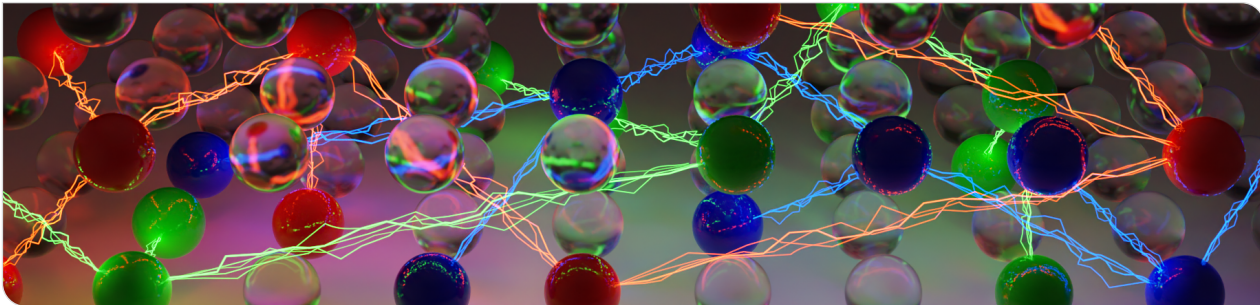


# Skalierbares SAT Solving und dessen Anwendung

*Scalable SAT Solving and its Application* | Promotionsprüfung

Dominik P. Schreiber | 17. November 2023



# Motivation: SAT Solving

## Das NP-vollständige Problem SAT [Cook 1971]

Gegeben eine aussagenlogische Formel  $F := \bigwedge_{c \in C} (\bigvee_{\ell \in c} \ell)$ , finde eine erfüllende Variablenbelegung für  $F$  oder melde **Unerfüllbarkeit**.



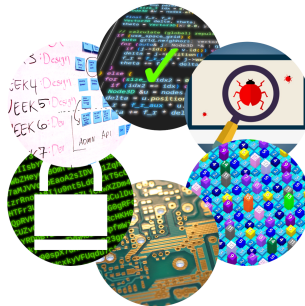
# Motivation: SAT Solving

## Das NP-vollständige Problem SAT [Cook 1971]

Gegeben eine aussagenlogische Formel  $F := \bigwedge_{c \in C} (\bigvee_{\ell \in c} \ell)$ , finde eine erfüllende Variablenbelegung für  $F$  oder melde **Unerfüllbarkeit**.

**SAT Solving:** Fundamentaler Baustein für eine Fülle von Anwendungen

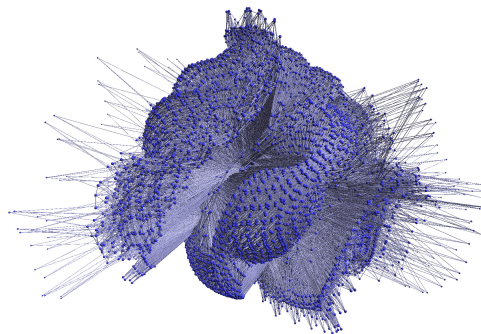
- Planung und Scheduling
- Formale Verifikation
- Testen und Debuggen
- Kryptographie
- Theorem-Beweise
- Schaltkreis-Design



# SAT: Grenzen der praktischen Lösbarkeit

**Beispiel:** Prüfe zwei **arithmetische Schaltkreise** auf Äquivalenz.

- Instanz A: 260k Variablen, 850k Klauseln
  - Schaltkreise sind **nicht äquivalent**
  - Gelöst in 1.33 s von KISSAT\_MAB\_PROP-NO\_SYM [Gao 2023]

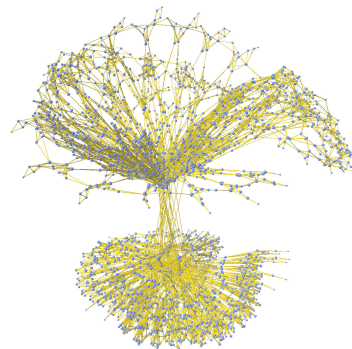


Knoten = Variablen;  
Kanten = gemeinsame Klausel(n);  
Variablen kontrahiert um Faktor  $\approx 16$

# SAT: Grenzen der praktischen Lösbarkeit

**Beispiel:** Prüfe zwei **arithmetische Schaltkreise** auf Äquivalenz.

- Instanz A: 260k Variablen, 850k Klauseln
  - Schaltkreise sind **nicht äquivalent**
  - **Gelöst in 1.33 s** von KISSAT\_MAB\_PROP-NO\_SYM [Gao 2023]
- Instanz B: 4k Variablen, 13k Klauseln
  - Schaltkreise sind **äquivalent**
  - **Ungelöst** von sequentiellen Solvern innerhalb von 5000 s



Knoten = Variablen;  
Kanten = gemeinsame Klausel(n)

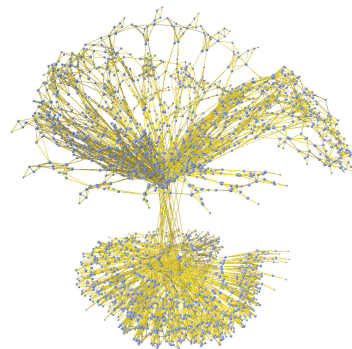
# SAT: Grenzen der praktischen Lösbarkeit

**Beispiel:** Prüfe zwei **arithmetische Schaltkreise** auf Äquivalenz.

- Instanz A: 260k Variablen, 850k Klauseln
  - Schaltkreise sind **nicht äquivalent**
  - **Gelöst in 1.33 s** von KISSAT\_MAB\_PROP-NO\_SYM [Gao 2023]
- Instanz B: 4k Variablen, 13k Klauseln
  - Schaltkreise sind **äquivalent**
  - **Ungelöst** von sequentiellen Solvern innerhalb von 5000 s

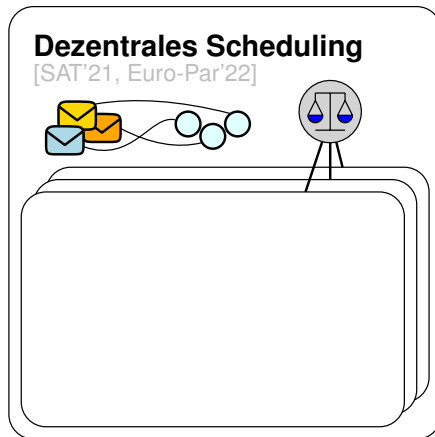
## Beobachtung

Es ergeben sich stets **praktisch relevante** Probleme, die mit aktuellen Solvern **nicht praktikabel lösbar** sind.

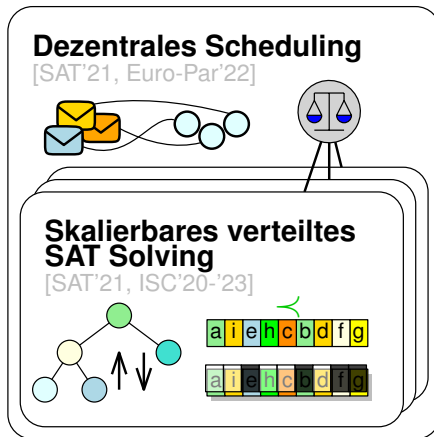


Knoten = Variablen;  
Kanten = gemeinsame Klausel(n)

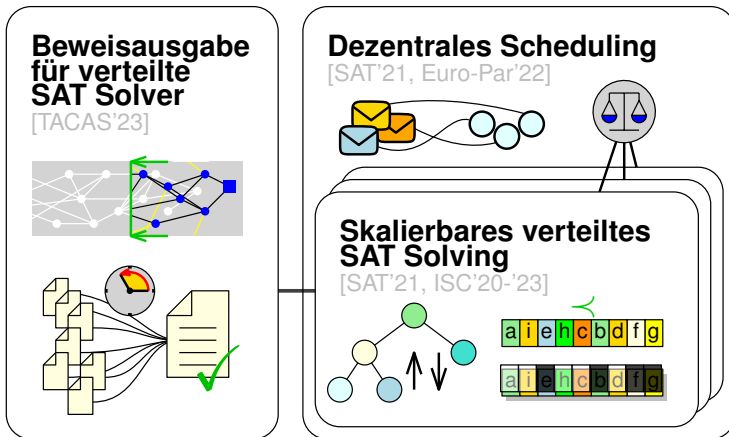
# Übersicht



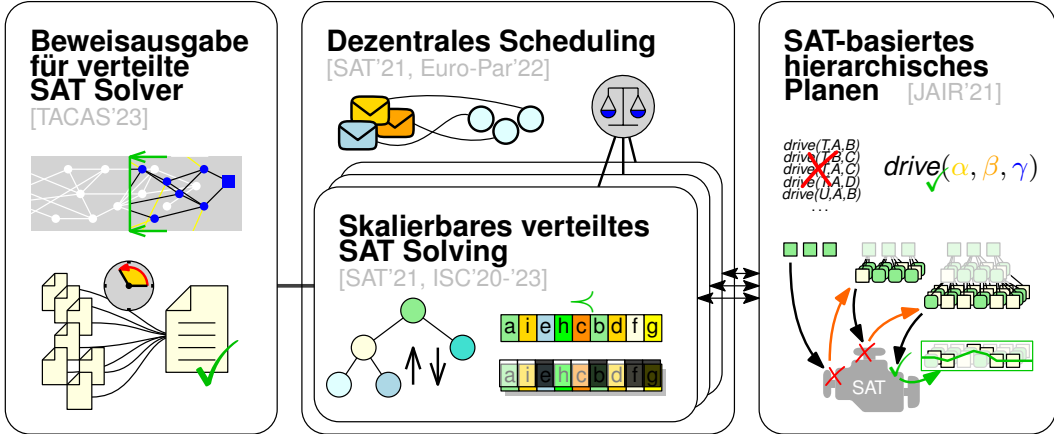
# Übersicht



# Übersicht

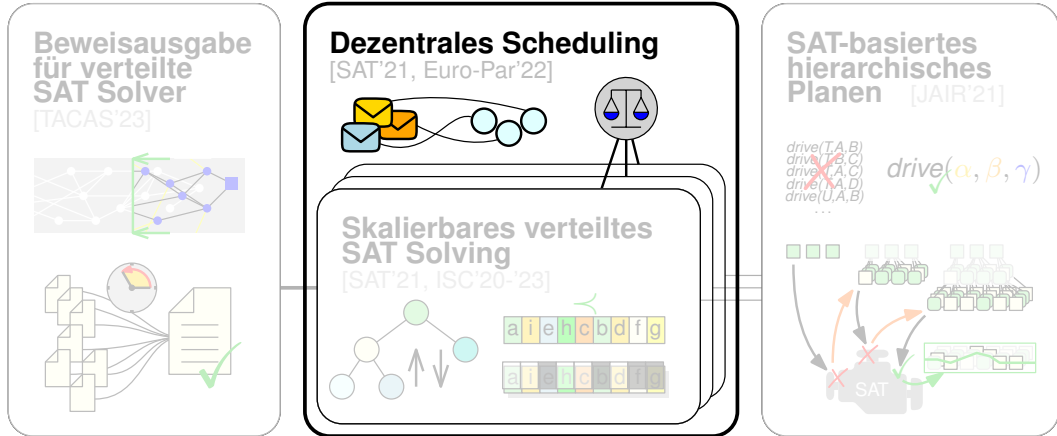


# Übersicht





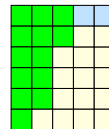
# Übersicht



# Scheduling: Motivation

Definition: *Verformbarkeit (Malleability)* [Feitelson 1997]

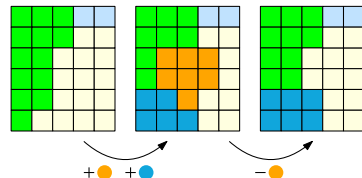
Eine parallele Rechnung ist **verformbar**, wenn sie zur Ausführungszeit eine **fluktuierende Anzahl von Recheneinheiten** unterstützt.



# Scheduling: Motivation

Definition: Verformbarkeit (*Malleability*) [Feitelson 1997]

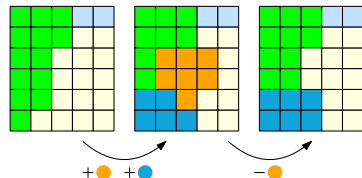
Eine parallele Rechnung ist **verformbar**, wenn sie zur Ausführungszeit eine **fluktuierende Anzahl von Recheneinheiten** unterstützt.



# Scheduling: Motivation

Definition: Verformbarkeit (*Malleability*) [Feitelson 1997]

Eine parallele Rechnung ist **verformbar**, wenn sie zur Ausführungszeit eine **fluktuierende Anzahl von Recheneinheiten** unterstützt.



## Warum verformbares Scheduling für SAT Solving?

- Ausführungszeit einer Aufgabe **unbekannt**  $\Rightarrow$  **Flexibles Reagieren** von Vorteil
- **Sublineare Skalierung**  $\Rightarrow$  Gleichzeitiges Bearbeiten mehrerer Probleme **erhöht Effizienz**
- Einfach umsetzbare **Verformbarkeit**  $\rightarrow$  Nächster Themenblock

# Ausführungsumgebung

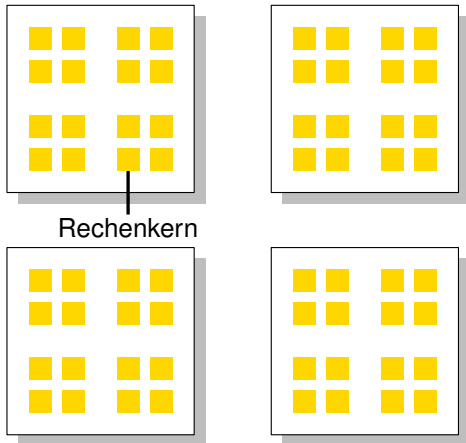
Maschine

Maschine

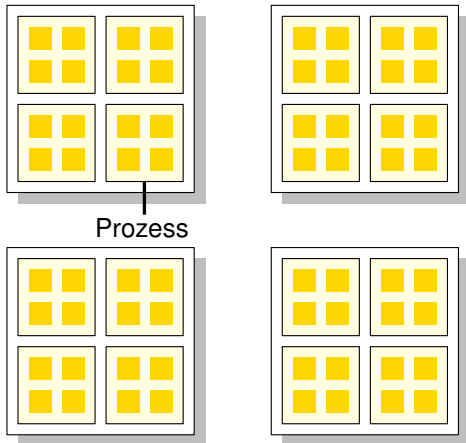
Maschine

Maschine

# Ausführungsumgebung



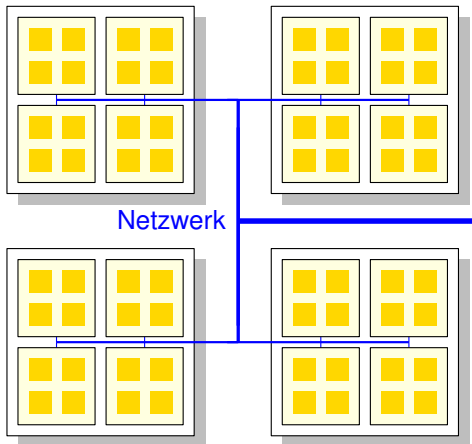
# Ausführungsumgebung



## Verteilte Rechenumgebung des Schedulers

- $m$  verteilte Prozesse

# Ausführungsumgebung

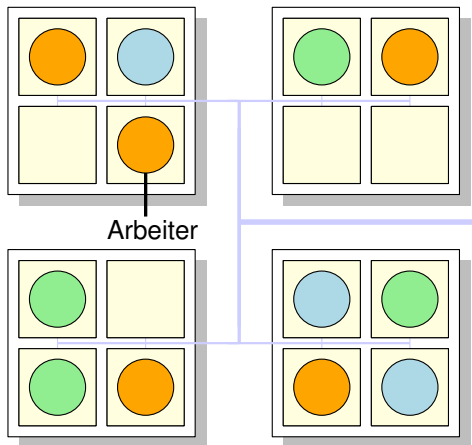


## Verteilte Rechenumgebung des Schedulers

- $m$  verteilte Prozesse



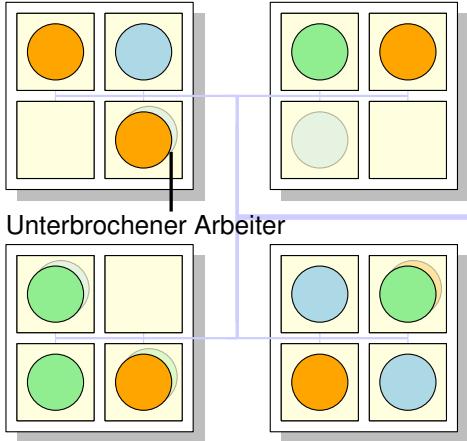
# Ausführungsumgebung



## Verteilte Rechenumgebung des Schedulers

- $m$  verteilte Prozesse
- **Arbeiter**: Ausführungskontext einer bestimmten Aufgabe auf einem bestimmten Prozess
- Je Prozess:
  - $\leq 1$  **aktive** Arbeiter
  - $\leq c$  unterbrochene Arbeiter

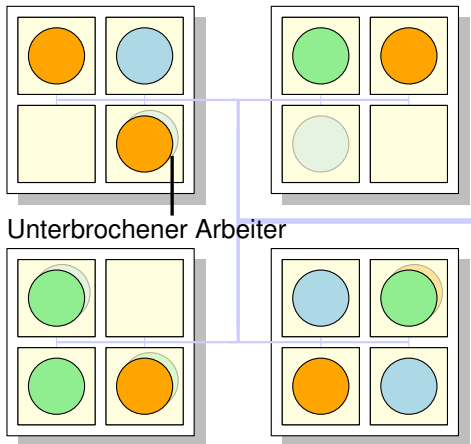
# Ausführungsumgebung



## Verteilte Rechenumgebung des Schedulers

- $m$  verteilte Prozesse
- **Arbeiter**: Ausführungskontext einer bestimmten Aufgabe auf einem bestimmten Prozess
- Je Prozess:
  - $\leq 1$  **aktive** Arbeiter
  - $\leq c$  unterbrochene Arbeiter

# Ausführungsumgebung



## Verteilte Rechenumgebung des Schedulers

- $m$  verteilte Prozesse
- **Arbeiter:** Ausführungskontext einer bestimmten Aufgabe auf einem bestimmten Prozess
- Je Prozess:
  - $\leq 1$  **aktive** Arbeiter
  - $\leq c$  unterbrochene Arbeiter
- Eigenschaften jeder Aufgabe  $j \in J$ :
  - **Priorität**  $p_j \in \mathbb{R}^+$
  - **Max. Ressourcen-Bedarf**  $d_j \in \mathbb{N}^+$

# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  **faire Anzahl**  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit **Tiefe**  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: **Aggregiere Ereignisse**, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  **faire Anzahl**  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit **Tiefe**  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: **Aggregiere Ereignisse**, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

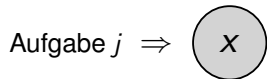
**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  **konkrete Prozesse** zu

# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  **faire Anzahl**  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit **Tiefe**  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: **Aggregiere Ereignisse**, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  **konkrete Prozesse** zu

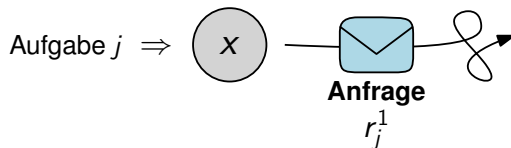


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  **faire Anzahl**  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit **Tiefe**  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: **Aggregiere Ereignisse**, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  **konkrete Prozesse** zu

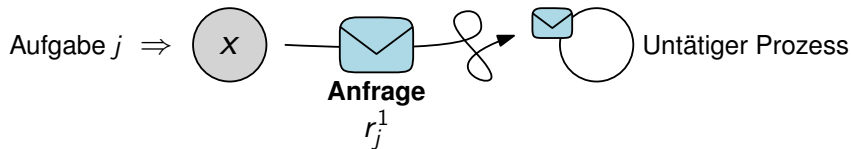


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  **faire Anzahl**  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit **Tiefe**  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: **Aggregiere Ereignisse**, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  **konkrete Prozesse** zu



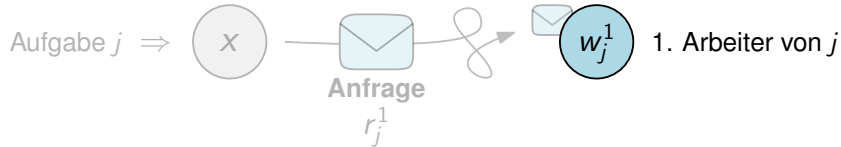


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  faire Anzahl  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit Tiefe  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: Aggregiere Ereignisse, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  konkrete Prozesse zu

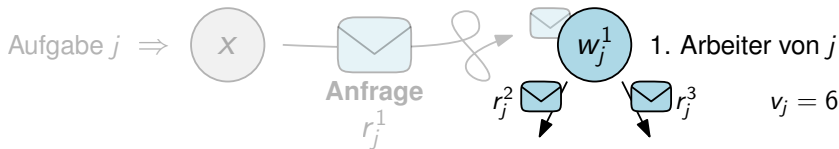


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  **faire Anzahl**  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit **Tiefe**  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: **Aggregiere Ereignisse**, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  **konkrete Prozesse** zu

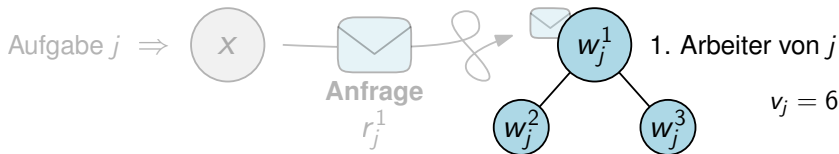


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  faire Anzahl  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit Tiefe  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: Aggregiere Ereignisse, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  konkrete Prozesse zu

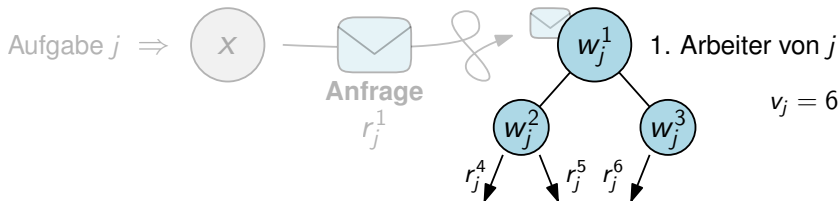


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  faire Anzahl  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit Tiefe  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: Aggregiere Ereignisse, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  konkrete Prozesse zu

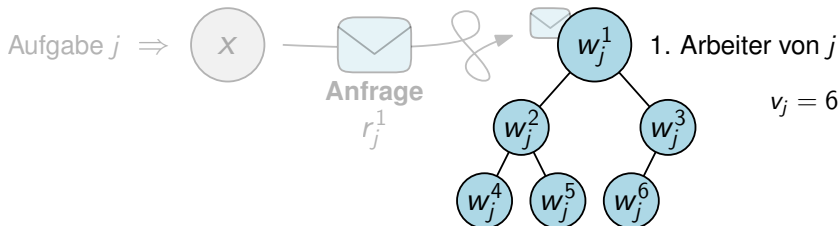


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  faire Anzahl  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

- Theorie: Voll skalierbarer Algorithmus mit Tiefe  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: Aggregiere Ereignisse, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  konkrete Prozesse zu

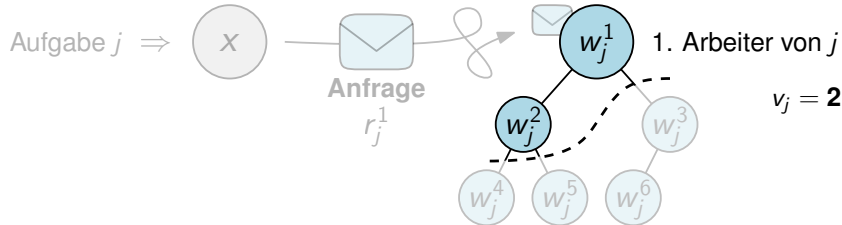


# Mallob: Scheduling-Ansatz [Euro-Par'22]

**Problem 1:** Bestimme für jede Aufgabe  $j$  faire Anzahl  $1 \leq v_j \leq d_j$  von Arbeitern, sodass  $v_j \propto p_j$

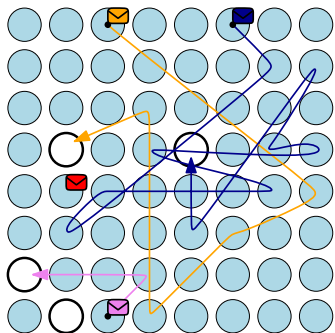
- Theorie: Voll skalierbarer Algorithmus mit Tiefe  $\mathcal{O}(\log m)$  durch kollektive Operationen
- Praxis: Aggregiere Ereignisse, die Systemzustand ändern  $\rightarrow$  berechne neue Zuteilung lokal

**Problem 2:** Ordne jeder Aufgabe  $j$   $v_j$  konkrete Prozesse zu



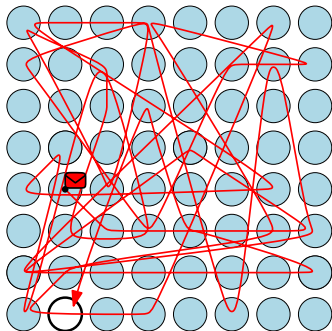
# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

## Random-Walk-Methode



# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

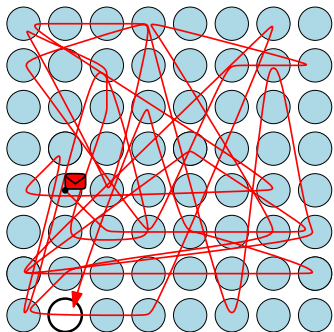
## Random-Walk-Methode



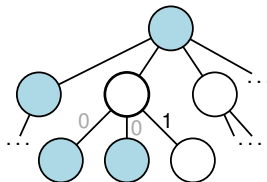


# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

## Random-Walk-Methode

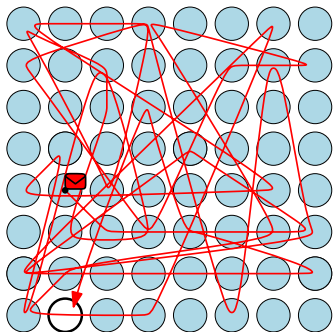


## Prozess-Baum

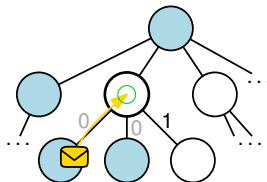


# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

## Random-Walk-Methode

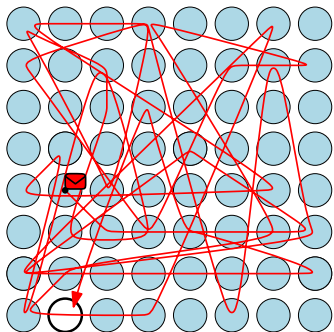


## Prozess-Baum

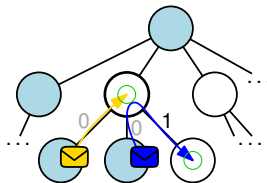


# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

## Random-Walk-Methode

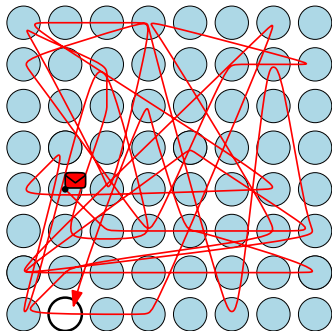


## Prozess-Baum

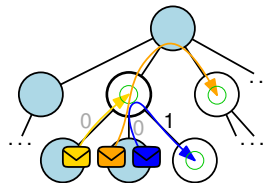


# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

## Random-Walk-Methode

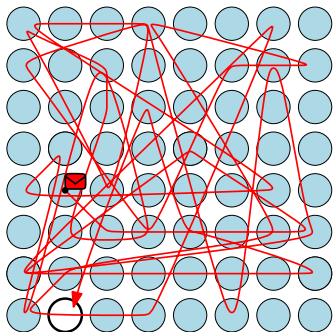


## Prozess-Baum

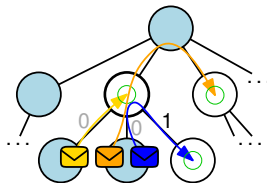


# Zuordnung von Anfragen und Prozessen [Euro-Par'22]

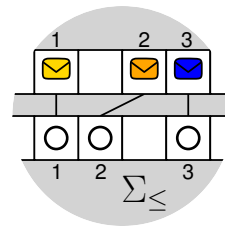
## Random-Walk-Methode



## Prozess-Baum



## Async. Präfixsummen



# Scheduling: Experimente [Euro-Par'22]

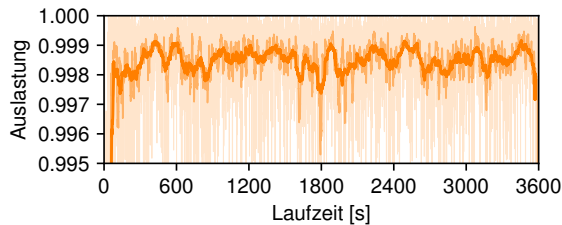
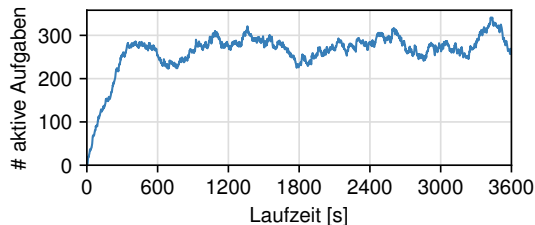
- 128 Maschinen von **SuperMUC-NG**
  - 1536 Prozesse × 4 Kerne
- Zufälliges Eintreffen von SAT-Aufgaben
- Bedarf, Priorität, max. Zeit-Budget zufällig

400 Probleme aus Int. SAT Competition 2020

# Scheduling: Experimente [Euro-Par'22]

- 128 Maschinen von **SuperMUC-NG**  
– 1536 Prozesse × 4 Kerne
- Zufälliges Eintreffen von SAT-Aufgaben
- Bedarf, Priorität, max. Zeit-Budget zufällig

400 Probleme aus Int. SAT Competition 2020



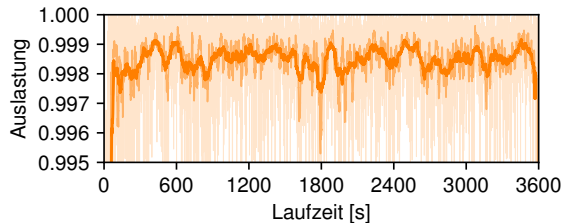
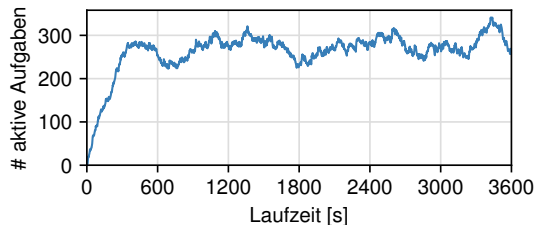
# Scheduling: Experimente [Euro-Par'22]

- 128 Maschinen von **SuperMUC-NG**
  - 1536 Prozesse × 4 Kerne
- Zufälliges Eintreffen von SAT-Aufgaben
- Bedarf, Priorität, max. Zeit-Budget zufällig

400 Probleme aus Int. SAT Competition 2020

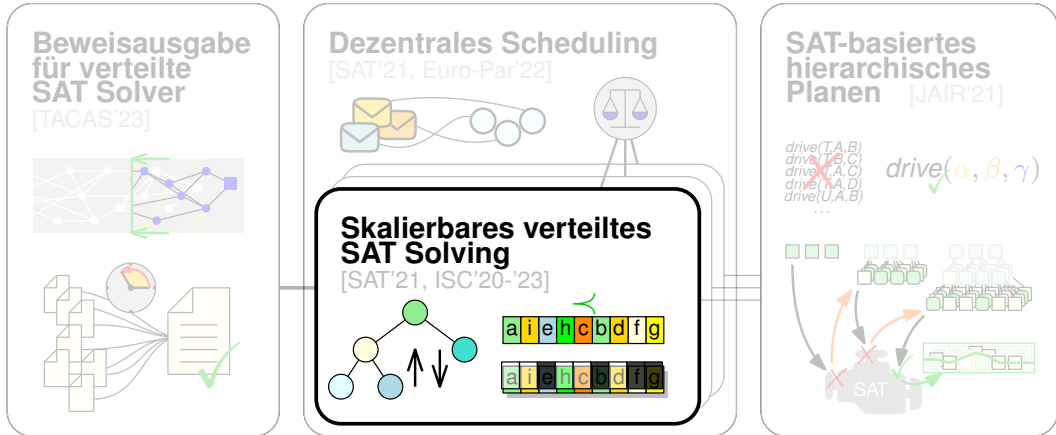
## Durchschnittliche Latenzen

- ≈ 10 ms für Scheduling des 1. Arbeiters
- ≈ 1 ms für Berechnung fairer Volumen
- ≈ 6 ms für Hinzufügen neuer Arbeiter





# Übersicht

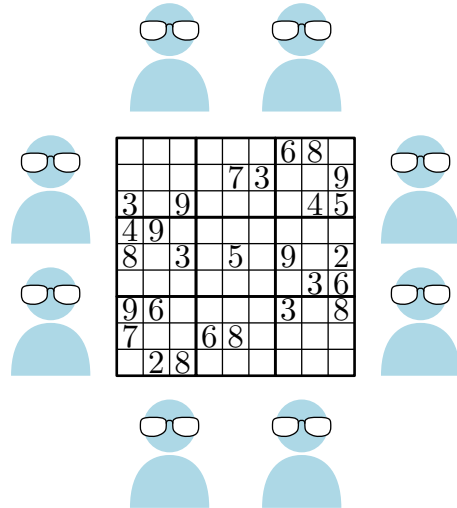


# Paralleles Logisches Schließen

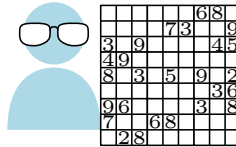
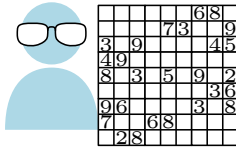
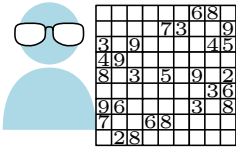
## Die Versammlung der Logik-Experten

- Komplexes Logik-Puzzle
- $n$  Puzzle-Experten möchten das Problem lösen
- Experten arbeiten **ungestört** am besten

Wie koordinieren wir unsere Experten?

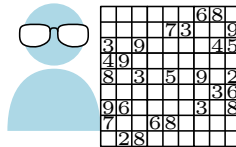
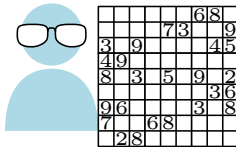
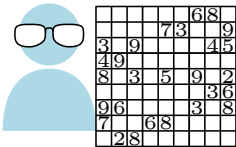
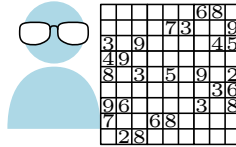
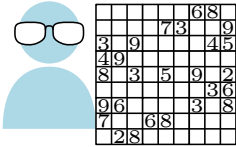


# Paralleles Logisches Schließen




## Paralleles Portfolio


- Alle bearbeiten **originales Problem** unabhängig voneinander




# Paralleles Logisches Schließen



					68		
6			73				9
3	9		6			4	5
4	9	6					
8	3	5	9		2		
						3	6
9	6				3		8
7		6	8				
	2	8				6	




					68		
			73				9
3	9			7		4	5
4	9						
8	3	5	9		2		
						4	3
9	6				3		8
7		6	8				
	2	8					




					68		
			73				9
3	9					4	5
4	9						
8	3	4	5	9	2		
						4	3
9	6				3		8
7	3	6	8				
	2	8					

## Paralleles Portfolio


- Alle bearbeiten **originales Problem** unabhängig voneinander




					68		
			73				9
3	9					4	5
4	9					8	5
8	3	5	9		2		
						3	6
9	6				3		8
7		6	8				
	2	8					




					68		
			73				9
3	9					4	5
4	9						
8	3	5	9		2		
9	6				3		8
7	3	6	8			9	
	2	8				6	



					68	3	
			73				9
3	9						4
4	9						5
8	3	5	9		2		
						3	6
9	6				3		8
7	3	6	8			9	
	2	8					

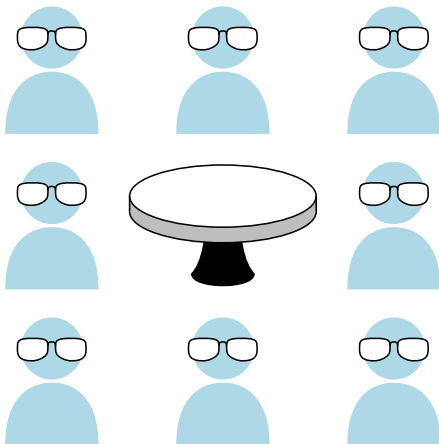


					68		
			73				9
3	9	6					4
4	9	6					5
8	3	4	5	6	9	2	
						3	6
9	6				3		8
7		6	8				
	2	8					



					68	3	
6			73				9
3	9	6		7			4
4	9						5
8	3	5	9		2		
						3	6
9	6				3		8
7		6	8				
	2	8					

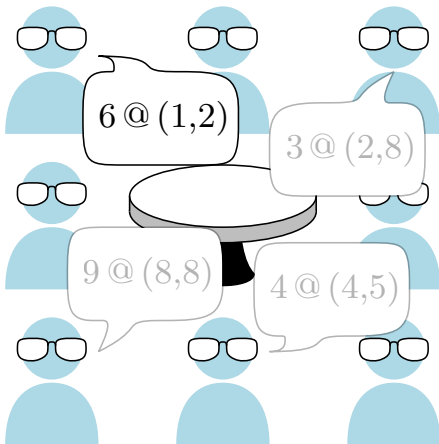
# Paralleles Logisches Schließen



## Paralleles Portfolio

- Alle bearbeiten **originales Problem** unabhängig voneinander
- Kurze Meetings, um wichtige **Erkenntnisse auszutauschen**


# Paralleles Logisches Schließen




## Paralleles Portfolio

- Alle bearbeiten **originales Problem** unabhängig voneinander
- Kurze Meetings, um wichtige **Erkenntnisse auszutauschen**


# Paralleles Logisches Schließen




					68				
6			73				9		
3	9		6	74	5				
49	6								
8	3	45	9	2					
							36		
96					3	8			
73	68			9					
28					6				




					68				
		6		73			9		
3	9		6	74	5				
49	6								
8	3	45	9	2					
							436		
96					3	8			
73	68			9					
28									




					68				
		6		73			9		
3	9		6	74	5				
49	6								
8	3	45	9	2					
							436		
96					3	8			
73	68			9					
28									




					68				
6			73				9		
3	9		6	74	5				
49	6				85				
8	3	45	9	2					
							36		
96					3	8			
73	68			9					
28									




					68				
		6		73			9		
3	9		6	74	5				
49	6								
8	3	45	9	2					
							36		
96					3	8			
73	68			9					
28							6		



					68	3			
6			73				9		
3	9		6	74	5				
49	6								
8	3	45	9	2					
							36		
96					3	8			
73	68			9					
28									



					68				
		6		73			9		
3	9		6	74	5				
49	6								
8	3	45	69	2					
							36		
96					3	8			
73	68			9					
28									

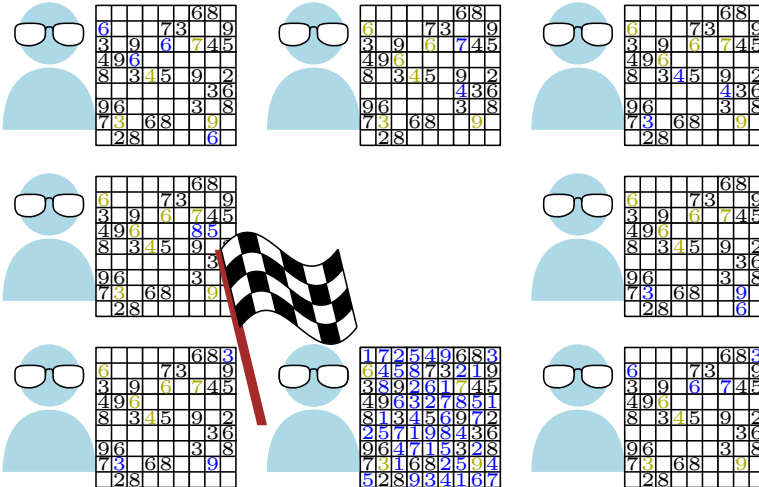


					68	3			
6			73				9		
3	9		6	74	5				
49	6								
8	3	45	9	2					
							36		
96					3	8			
73	68			9					
28									

## Paralleles Portfolio

- Alle bearbeiten **originales Problem** unabhängig voneinander
- Kurze Meetings, um wichtige **Erkenntnisse auszutauschen**
- Erkenntnisse **beschleunigen** das Lösen

# Paralleles Logisches Schließen



The diagram shows six solvers working on a 9x9 grid puzzle. The puzzle is divided into six regions, each assigned to a solver. The solvers are represented by icons of a person with glasses. The puzzle is shown in three stages: initial state, intermediate state, and final state. A checkered flag is placed over the final state, indicating the solution is found.

**Initial State:**

					6	8		
6			7	3			9	
3	9		6		7	4	5	
4	9	6						
8	3	4	5	9		2		
							3	6
9	6					3		8
7	3		6	8			9	
2	8							6

**Intermediate State:**

					6	8		
6			7	3			9	
3	9		6		7	4	5	
4	9	6						
8	3	4	5	9		2		
							4	3
9	6					3		8
7	3		6	8			9	
2	8							

**Final State:**

					6	8	3	
6			7	3			9	
3	9		6		7	4	5	
4	9	6						
8	3	4	5	9		2		
								3
9	6					3		8
7	3		6	8			9	
2	8							6

**Final Solution:**

1	7	2	5	4	9	6	8	3
6	4	5	8	7	3	2	1	9
3	8	9	2	6	1	7	4	5
4	9	6	3	2	7	8	5	1
8	1	3	4	5	6	9	7	2
2	5	7	1	9	8	4	3	6
9	6	4	7	1	5	3	2	8
7	3	1	6	8	2	5	9	4
5	2	8	9	3	4	1	6	7

## Paralleles Portfolio

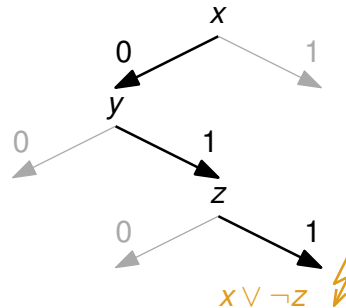
- Alle bearbeiten **originales Problem** unabhängig voneinander
- Kurze Meetings, um wichtige **Erkenntnisse auszutauschen**
- Erkenntnisse **beschleunigen** das Lösen
- Nur **ein Experte** muss eine Lösung finden!



# Verteiltes SAT Solving: Stand der Technik

## Paralleles SAT Solving [Hamadi et al. 2010]

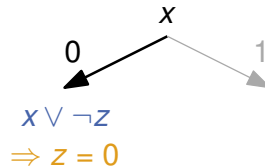
- Experten  $\equiv$  Sequentielle Suchalgorithmen



# Verteiltes SAT Solving: Stand der Technik

## Paralleles SAT Solving [Hamadi et al. 2010]

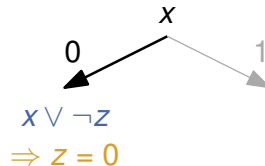
- Experten  $\equiv$  Sequentielle Suchalgorithmen



# Verteiltes SAT Solving: Stand der Technik

## Paralleles SAT Solving [Hamadi et al. 2010]

- Experten  $\equiv$  Sequentielle Suchalgorithmen
- Geteilte Informationen  $\equiv$  **gelernte Konfliktklauseln**



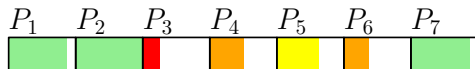
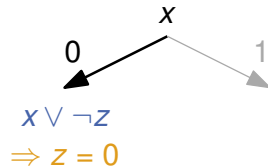
# Verteiltes SAT Solving: Stand der Technik

## Paralleles SAT Solving [Hamadi et al. 2010]

- Experten  $\equiv$  Sequentielle Suchalgorithmen
- Geteilte Informationen  $\equiv$  **gelernte Konfliktklauseln**

## Vorheriger Stand der Technik: **HordeSat** [Balyo et al. 2015]

- Periodischer **Klauselaustausch**
  - Konkatination von **Klausel-Buffern** fester Größe
  - **Duplikate, ungenutzter Platz** in Buffern



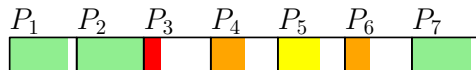
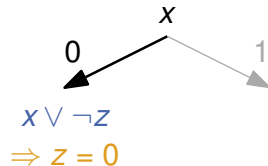
# Verteiltes SAT Solving: Stand der Technik

## Paralleles SAT Solving [Hamadi et al. 2010]

- Experten  $\equiv$  Sequentielle Suchalgorithmen
- Geteilte Informationen  $\equiv$  **gelernte Konfliktklauseln**

## Vorheriger Stand der Technik: **HordeSat** [Balyo et al. 2015]

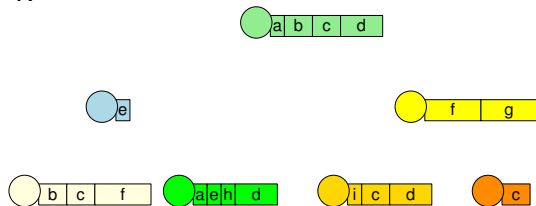
- Periodischer **Klauselaustausch**
  - Konkatination von **Klausel-Buffern** fester Größe
  - **Duplikate**, **ungenutzter Platz** in Buffern
- Experimente mit  $\leq 2048$  Kernen
  - Individuelle **super-lineare Speedups** ( $> 2048$ )
  - Median Speedup auf 2048 Kernen: **13** (**Effizienz 0.6%**)



# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

## Austausch nützlicher Klauseln

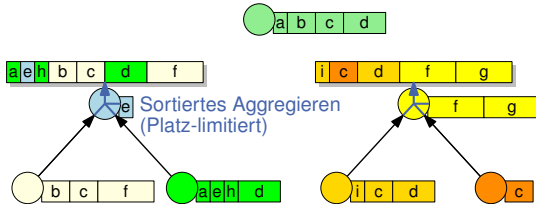
1.



# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

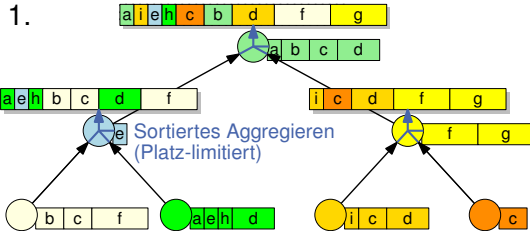
## Austausch nützlicher Klauseln

1.



# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

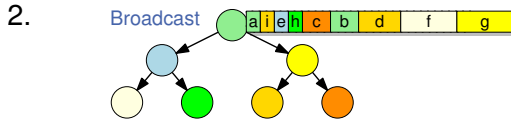
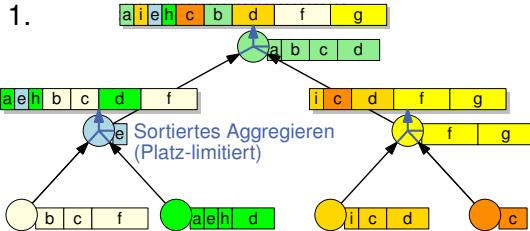
## Austausch nützlicher Klauseln





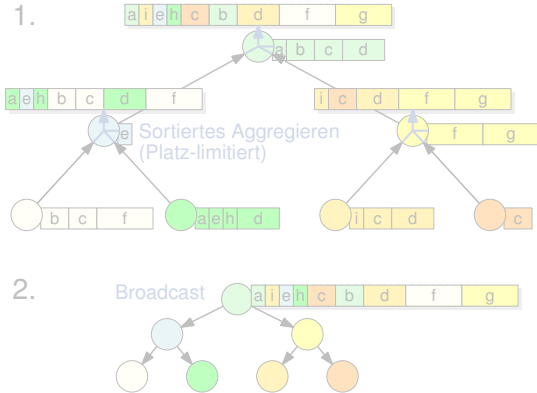
# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

## Austausch nützlicher Klauseln

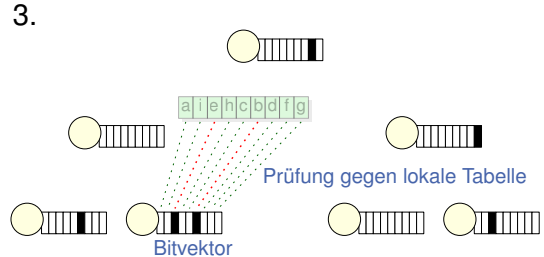


# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

## Austausch nützlicher Klauseln

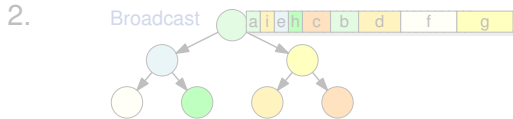
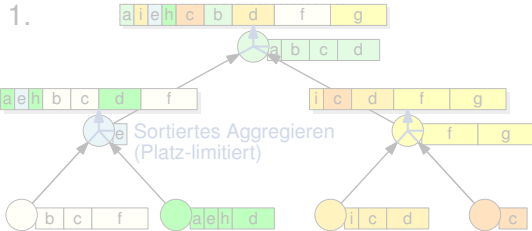


## Filtern zuvor geteilter Klauseln

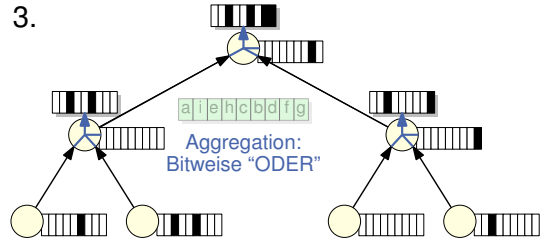


# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

## Austausch nützlicher Klauseln

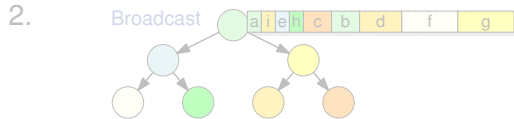
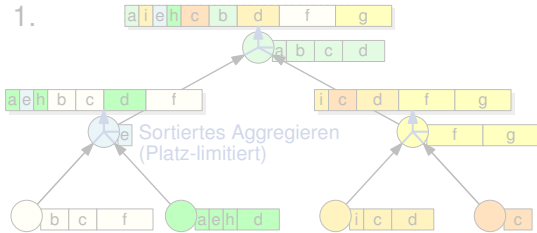


## Filtern zuvor geteilter Klauseln

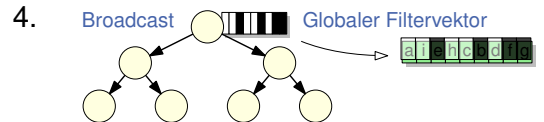
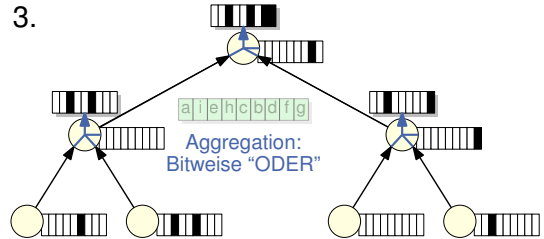


# Klauselaustausch: Unser Ansatz [SAT'21, ISC'20-23]

## Austausch nützlicher Klauseln



## Filtern zuvor geteilter Klauseln



# Unser System MALLOBSAT [SAT'21, ISC'20-23]

## Solver-Konfiguration

- Schnittstellen für **performante Solver** (KISSAT, CADICAL, LINGELING, GLUCOSE)  
[Biere et al. 2018, 2020; Audemard & Simon 2009]
- **Minimale Randomisierungen**  $\Rightarrow$  effektive Aufteilung per Klauselaustausch

# Unser System MALLOBSAT [SAT'21, ISC'20-23]

## Solver-Konfiguration

- Schnittstellen für **performante Solver** (KISSAT, CADICAL, LINGELING, GLUCOSE) [Biere et al. 2018, 2020; Audemard & Simon 2009]
- **Minimale Randomisierungen**  $\Rightarrow$  effektive Aufteilung per Klauselaustausch

## Verformbarkeit

- **Prozess-Baum** für jegliche Kommunikation
- Beliebiges **Hinzufügen, Entfernen** von Solver-Prozessen

# Unser System MALLOBSAT [SAT'21, ISC'20-23]

## Solver-Konfiguration

- Schnittstellen für **performante Solver** (KISSAT, CADICAL, LINGELING, GLUCOSE) [Biere et al. 2018, 2020; Audemard & Simon 2009]
- **Minimale Randomisierungen** ⇒ effektive Arbeitsaufteilung per Klauselaustausch

## Verformbarkeit

- **Prozess-Baum** für jegliche Kommunikation
- Beliebiges **Hinzufügen, Entfernen** von Solver-Prozessen

*“Mallob-mono is now, by a wide margin, the most powerful SAT solver on the planet.”*

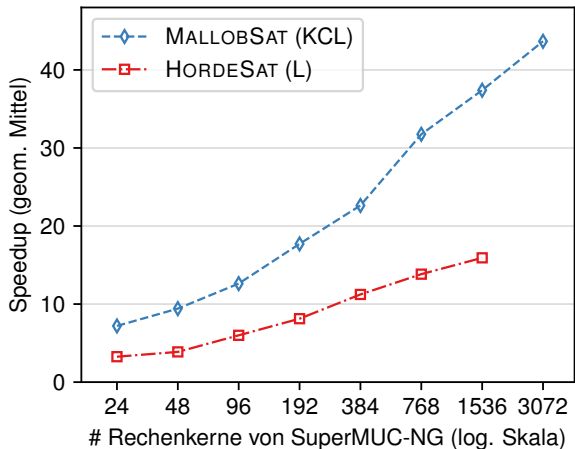
—Byron Cook, Amazon Distinguished Scientist, 2021

<https://www.amazon.science/blog/automated-reasonings-scientific-frontiers>



Int. SAT Competition  
2020–2023

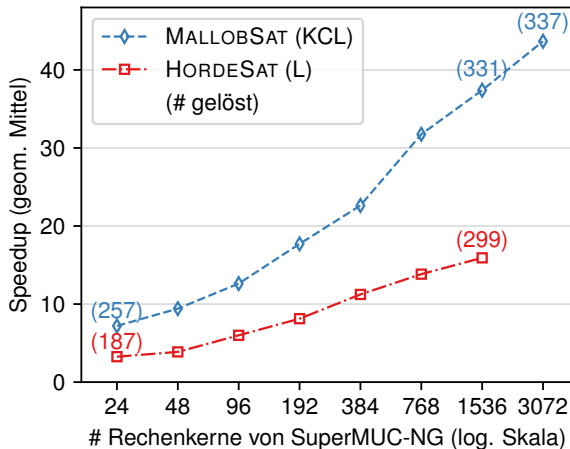
# MALLOBSAT: Performance und Skalierung



400 Probleme aus SAT Comp. 2021 · Seq. Baseline KISSAT\_MAB-HYWALK · Seq. Zeitlimit 32 h (gelöst: 331) · Par. Zeitlimit 300 s

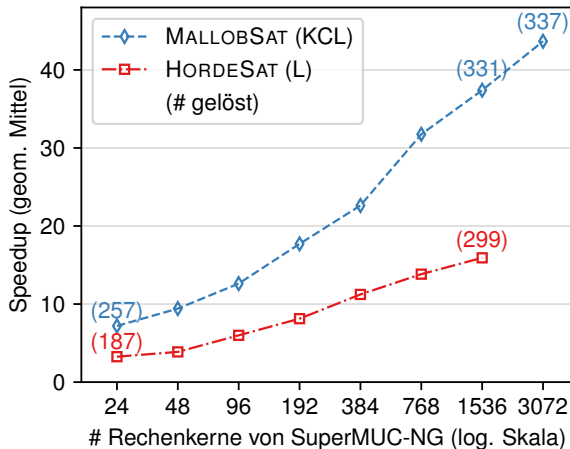


# MALLOBSAT: Performance und Skalierung



400 Probleme aus SAT Comp. 2021 · Seq. Baseline KISSAT\_MAB-HYWALK · Seq. Zeitlimit 32 h (gelöst: 331) · Par. Zeitlimit 300 s

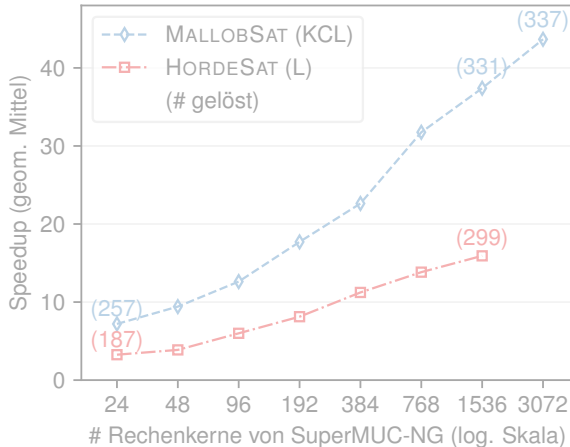
# MALLOBSAT: Performance und Skalierung



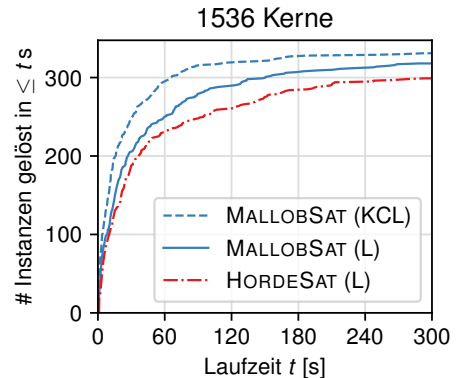
Seq. Laufzeit  $\geq 1$  h  $\Rightarrow$  Speedup 419 auf 3072 Kernen

400 Probleme aus SAT Comp. 2021 · Seq. Baseline KISSAT\_MAB-HYWALK · Seq. Zeitlimit 32 h (gelöst: 331) · Par. Zeitlimit 300 s

# MALLOBSAT: Performance und Skalierung



Seq. Laufzeit  $\geq 1$  h  $\Rightarrow$  Speedup **419** auf 3072 Kernen



400 Probleme aus SAT Comp. 2021 · Seq. Baseline KISSAT\_MAB-HYWALK · Seq. Zeitlimit 32 h (gelöst: 331) · Par. Zeitlimit 300 s

# MALLOBSAT: Gleichzeitiges Lösen vieler Aufgaben

## Scheduling von 400 SAT-Problemen

$x \in \{400, 1600, 6400\}$  Kerne, 2 h Laufzeit.

- **Starr:** Jede Aufgabe erhält  $\frac{x}{400} \in \{1, 4, 16\}$  Kerne
- **Flexibel:** Ressourcen von fertigen Aufgaben werden an verbleibende Aufgaben **umverteilt**

1/4/4 Kerne pro Prozess · 400 Probleme aus SAT Comp. 2021

# MALLOBSAT: Gleichzeitiges Lösen vieler Aufgaben

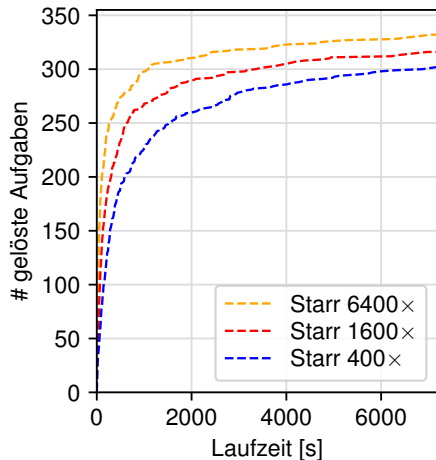
## Scheduling von 400 SAT-Problemen

$x \in \{400, 1600, 6400\}$  Kerne, 2 h Laufzeit.

- **Starr:** Jede Aufgabe erhält  $\frac{x}{400} \in \{1, 4, 16\}$  Kerne
- **Flexibel:** Ressourcen von fertigen Aufgaben werden an verbleibende Aufgaben **umverteilt**

1/4/4 Kerne pro Prozess · 400 Probleme aus SAT Comp. 2021  
Starr 400×  $\equiv$  400× KISSAT\_MAB-HYWALK [Zheng et al. 2022]

Animation: <https://dominikschreiber.de/animallob>



# MALLOBSAT: Gleichzeitiges Lösen vieler Aufgaben

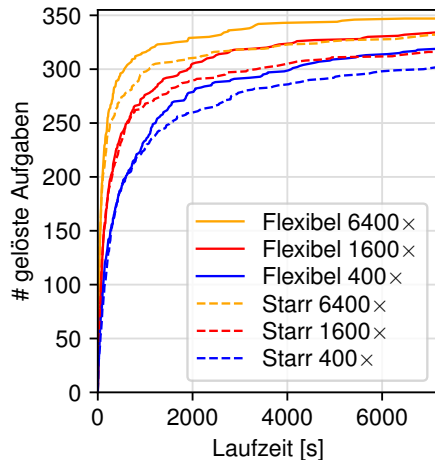
## Scheduling von 400 SAT-Problemen

$x \in \{400, 1600, 6400\}$  Kerne, 2 h Laufzeit.

- **Starr:** Jede Aufgabe erhält  $\frac{x}{400} \in \{1, 4, 16\}$  Kerne
- **Flexibel:** Ressourcen von fertigen Aufgaben werden an verbleibende Aufgaben **umverteilt**

1/4/4 Kerne pro Prozess · 400 Probleme aus SAT Comp. 2021  
 Starr 400x  $\equiv$  400x KISSAT\_MAB-HYWALK [Zheng et al. 2022]

Animation: <https://dominikschreiber.de/animallobsat>



# MALLOBSAT: Gleichzeitiges Lösen vieler Aufgaben

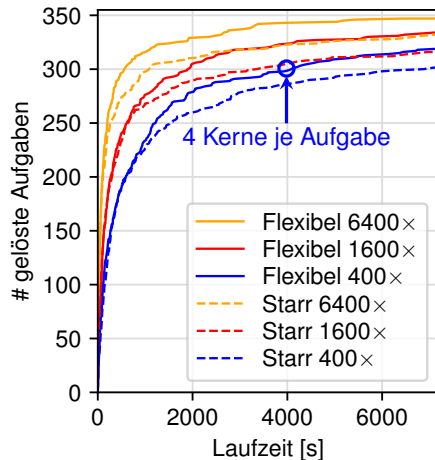
## Scheduling von 400 SAT-Problemen

$x \in \{400, 1600, 6400\}$  Kerne, 2 h Laufzeit.

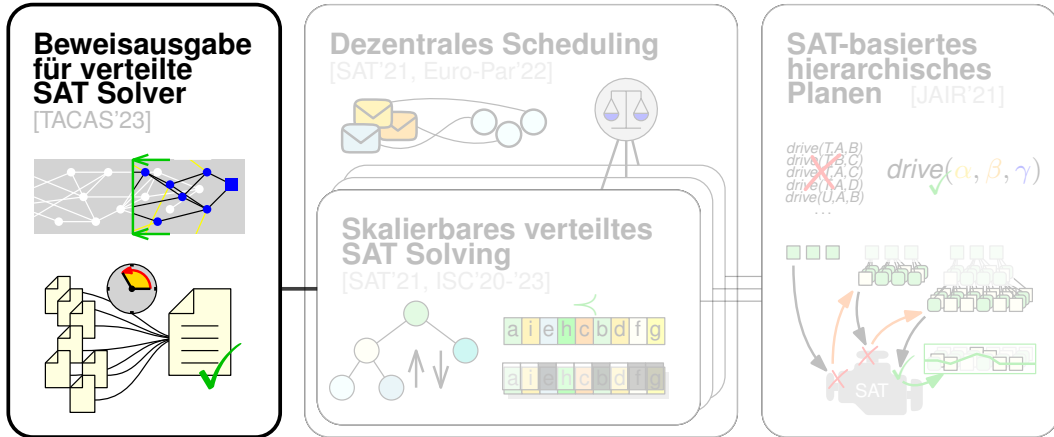
- **Starr**: Jede Aufgabe erhält  $\frac{x}{400} \in \{1, 4, 16\}$  Kerne
- **Flexibel**: Ressourcen von fertigen Aufgaben werden an verbleibende Aufgaben **umverteilt**

1/4/4 Kerne pro Prozess · 400 Probleme aus SAT Comp. 2021  
 Starr 400×  $\equiv$  400× KISSAT\_MAB-HYWALK [Zheng et al. 2022]

Animation: <https://dominikschreiber.de/animallobsat>



# Übersicht

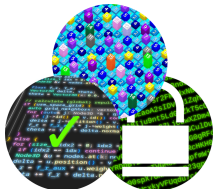




# Motivation: Beweise für Unerfüllbarkeit

## Fragestellung

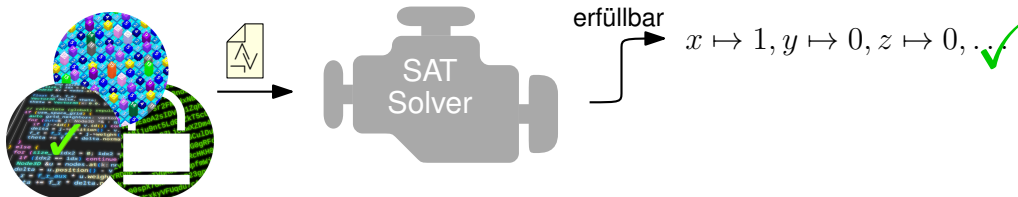
Wie können wir sicherstellen, dass die Ausgabe eines SAT Solvers **korrekt** ist?



# Motivation: Beweise für Unerfüllbarkeit

## Fragestellung

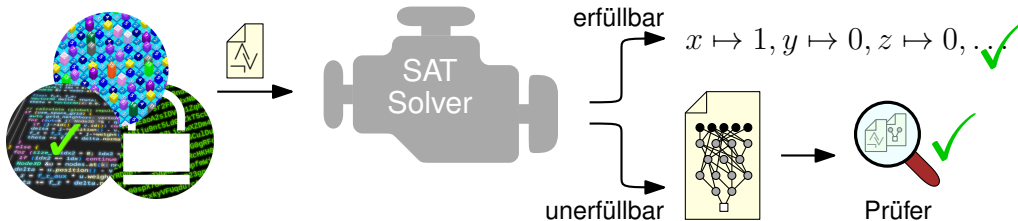
Wie können wir sicherstellen, dass die Ausgabe eines SAT Solvers **korrekt** ist?



# Motivation: Beweise für Unerfüllbarkeit

## Fragestellung

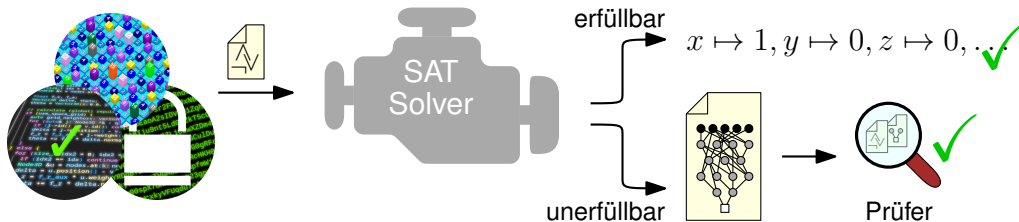
Wie können wir sicherstellen, dass die Ausgabe eines SAT Solvers **korrekt** ist?



# Motivation: Beweise für Unerfüllbarkeit

## Fragestellung

Wie können wir sicherstellen, dass die Ausgabe eines SAT Solvers **korrekt** ist?

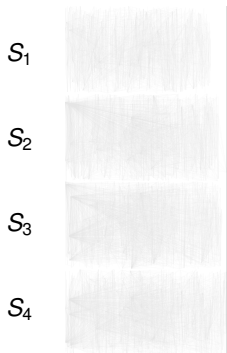


## Vorheriger Stand der Technik für Beweise

- Sequentielle Solver: **etablierter Standard**, **strikte Voraussetzung** in Wettbewerben
- Parallele/verteilte Solver mit Klauselaustausch: **keine praktikablen Ansätze**

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

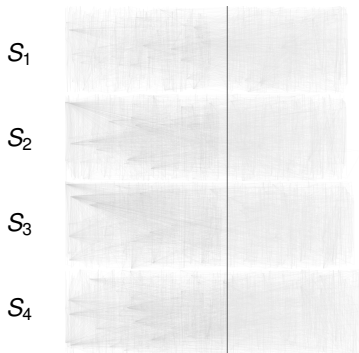


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

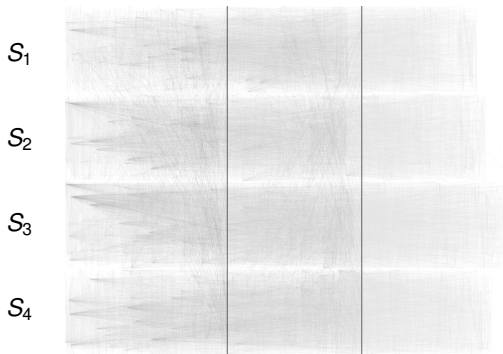


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

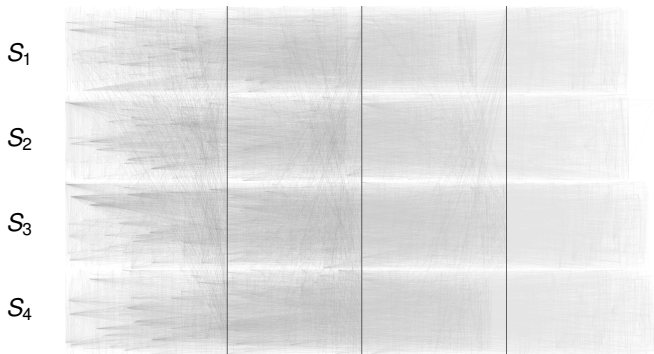


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →



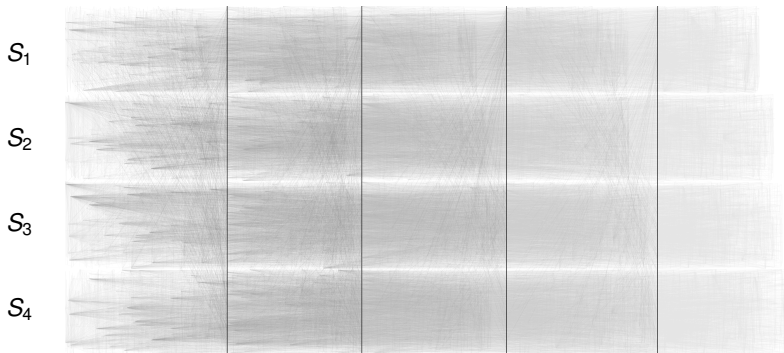
Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)



# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

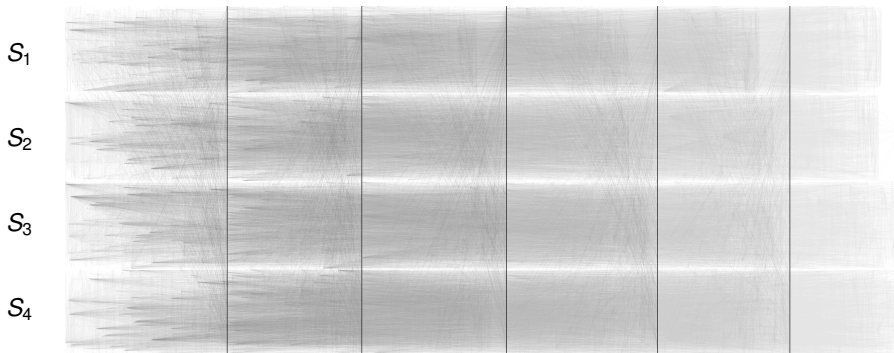


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

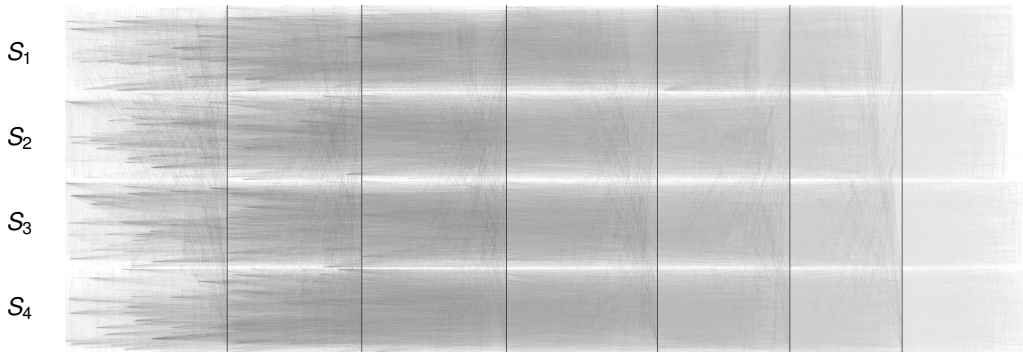


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

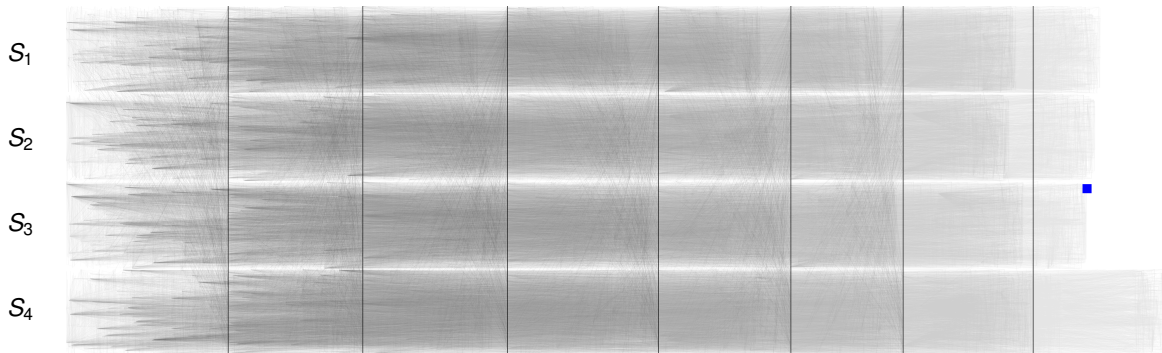


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

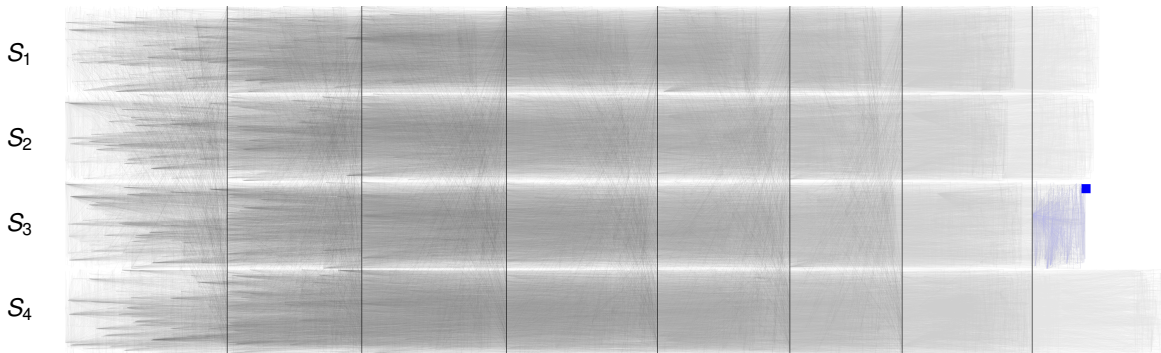


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Solving:** [Herleitung und Austausch](#) von Klauseln, Schreiben in [partielle Beweise](#)

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

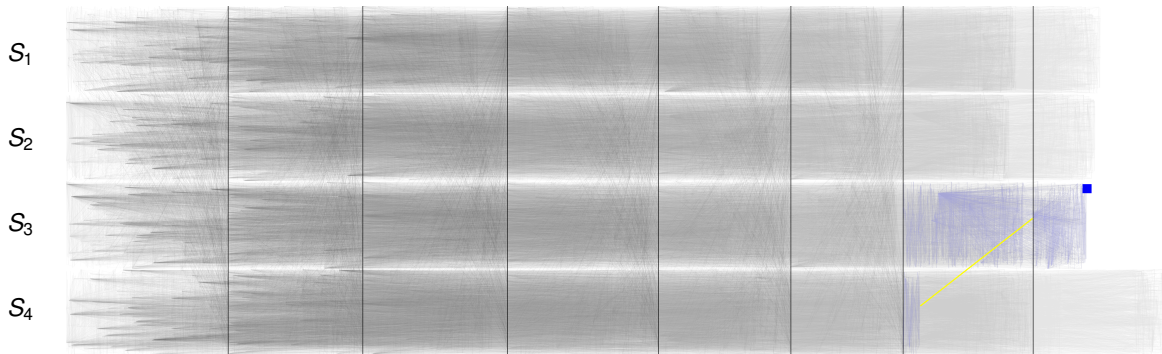


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

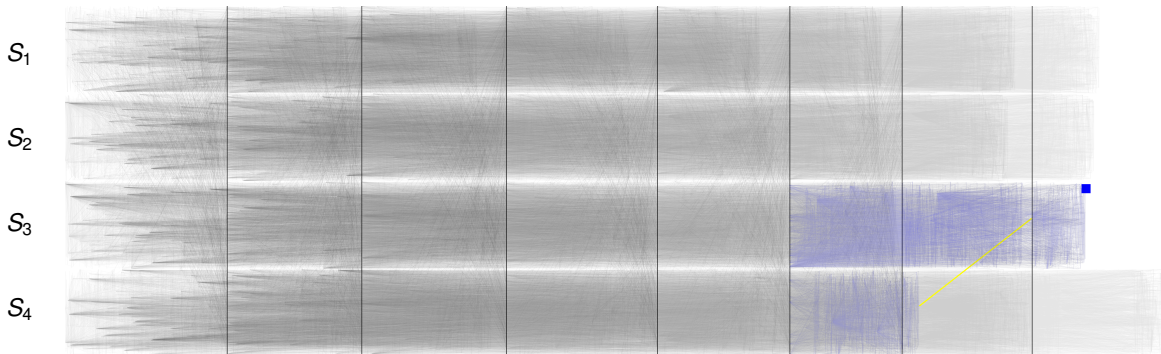


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

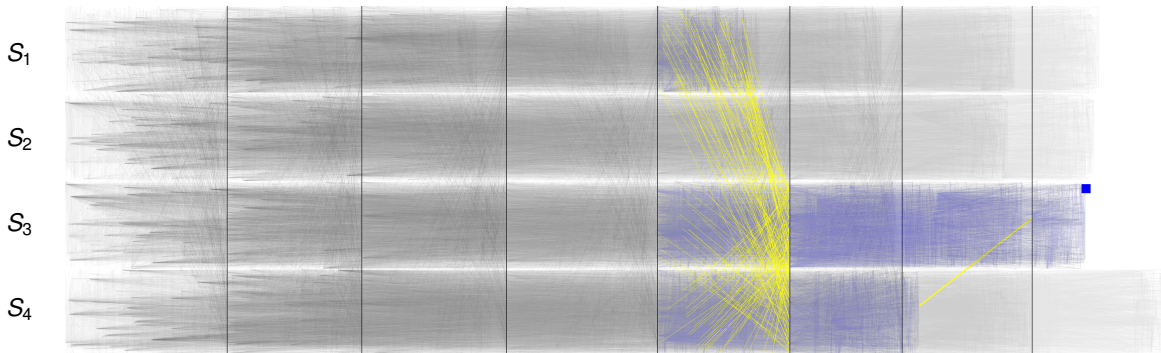


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →



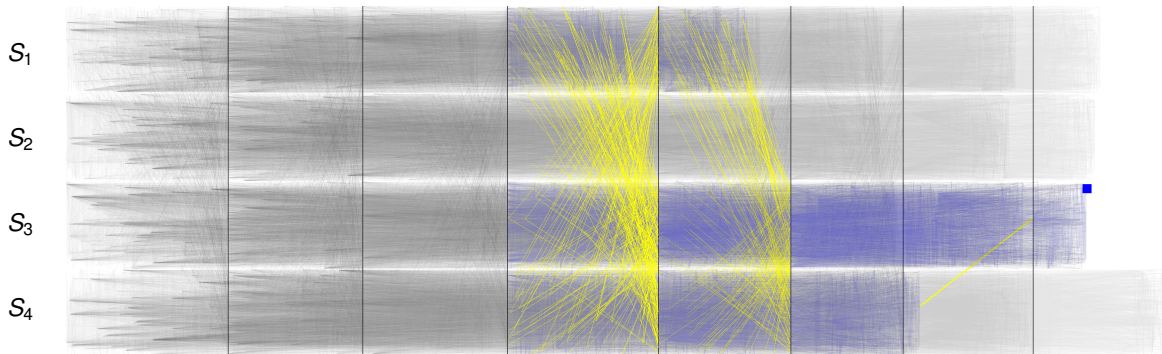
Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs



# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

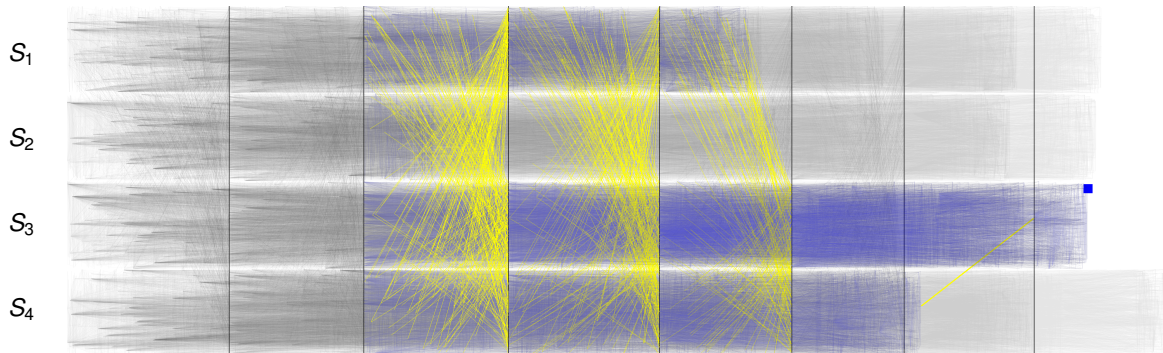


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

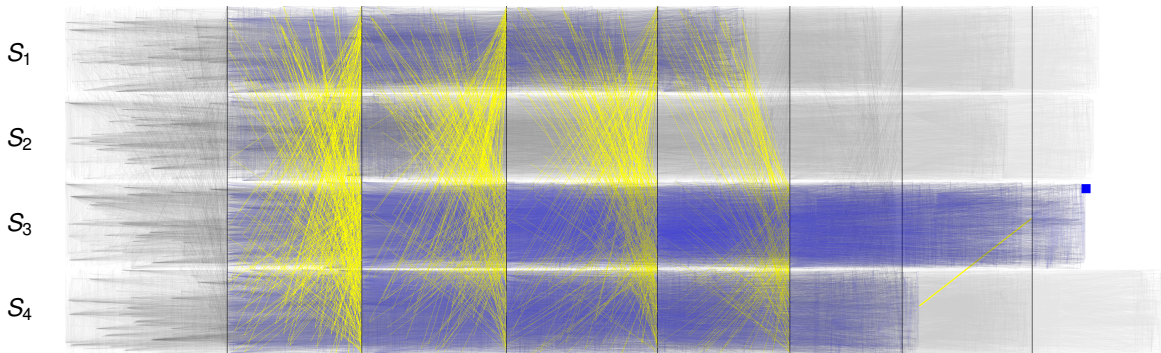


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

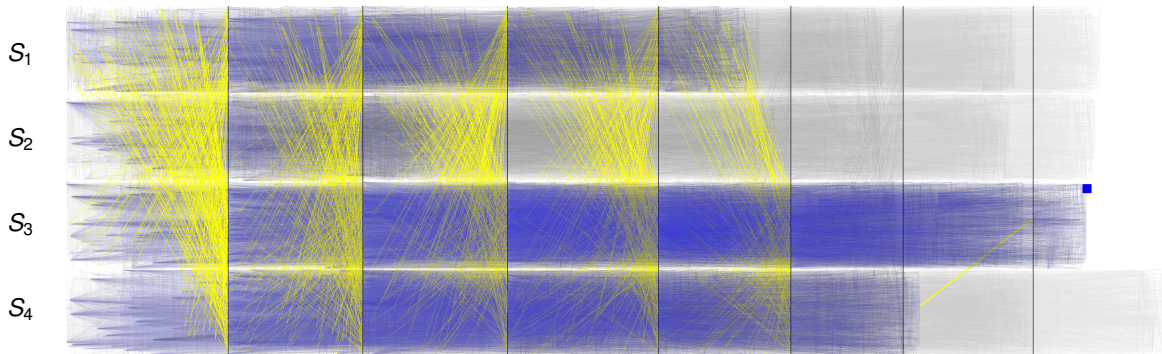


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Verteilte Beweis-Rekonstruktion [TACAS'23]

— Produzierte Klauseln →

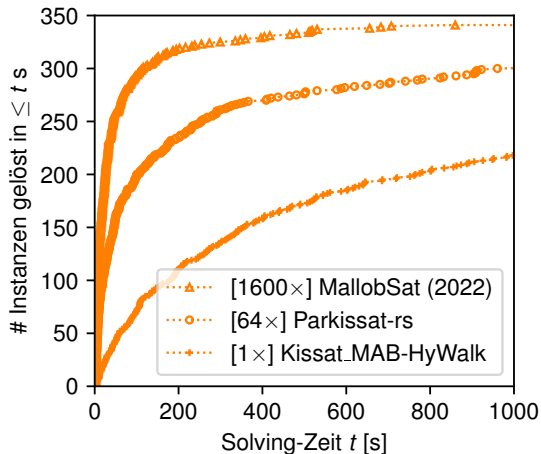


Zuf. 3-SAT Formel, 180 Variablen. 4 Notebook-Kerne  $\times$  1.7 s. 300k Abhängigkeiten (ohne orig. Klauseln).

**Rekonstruktion:** Rückverfolgung nötiger Klauseln, Umkehrung jedes Klauselaustauschs

# Experimente [TACAS'23]

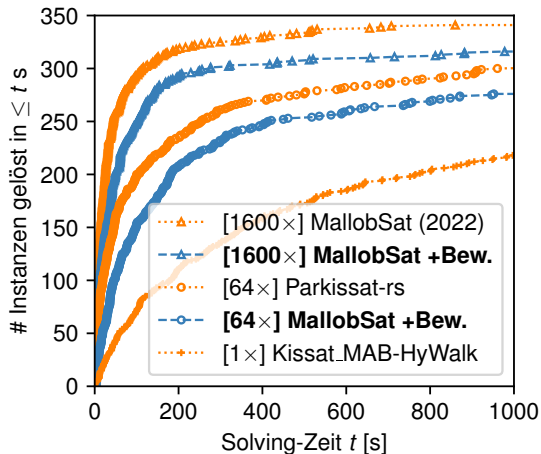
## Reine Solving-Performance



AWS Cloud · 400 Probleme aus SAT Comp. 2022

# Experimente [TACAS'23]

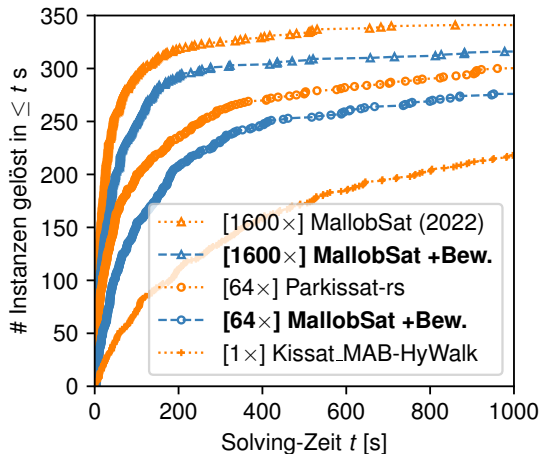
## Reine Solving-Performance



AWS Cloud · 400 Probleme aus SAT Comp. 2022

# Experimente [TACAS'23]

## Reine Solving-Performance



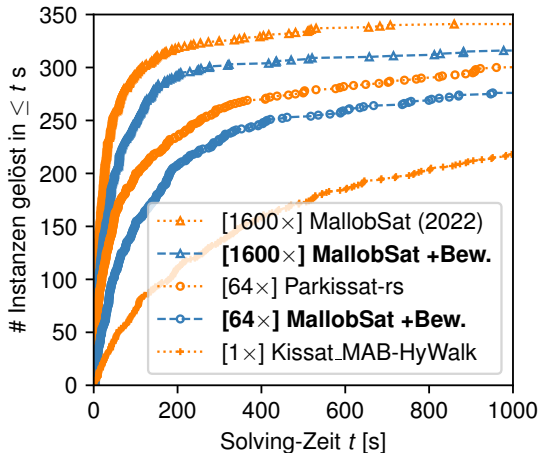
## Beweis-Rekonstruktion & -Prüfung

- **[64x]**: 135 Beweise rekonstruiert & geprüft  
in  $\approx 1.1 \times$  Solving-Zeit
- **[1600x]**: 154 Beweise rekonstruiert & geprüft  
in  $\approx 4.8 \times$  Solving-Zeit

AWS Cloud · 400 Probleme aus SAT Comp. 2022

# Experimente [TACAS'23]

## Reine Solving-Performance



## Beweis-Rekonstruktion & -Prüfung

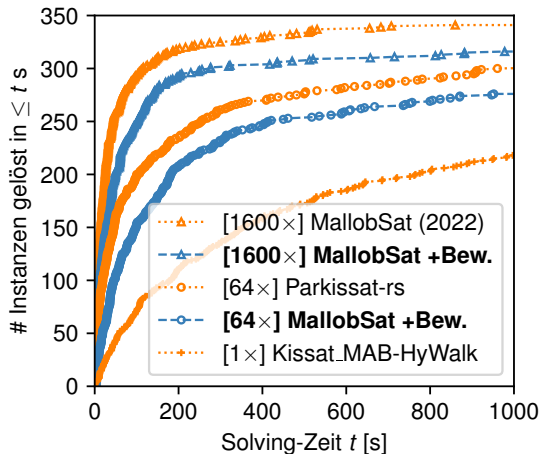
- **[64x]:** 135 Beweise rekonstruiert & geprüft  
in  $\approx 1.1 \times$  Solving-Zeit
- **[1600x]:** 154 Beweise rekonstruiert & geprüft  
in  $\approx 4.8 \times$  Solving-Zeit
- Vorheriger Ansatz (64x) [Heule et al. 2014]:  
81 Beweise geprüft mit Mehraufwand  $10.4 \times$

AWS Cloud · 400 Probleme aus SAT Comp. 2022



# Experimente [TACAS'23]

## Reine Solving-Performance



## Beweis-Rekonstruktion & -Prüfung

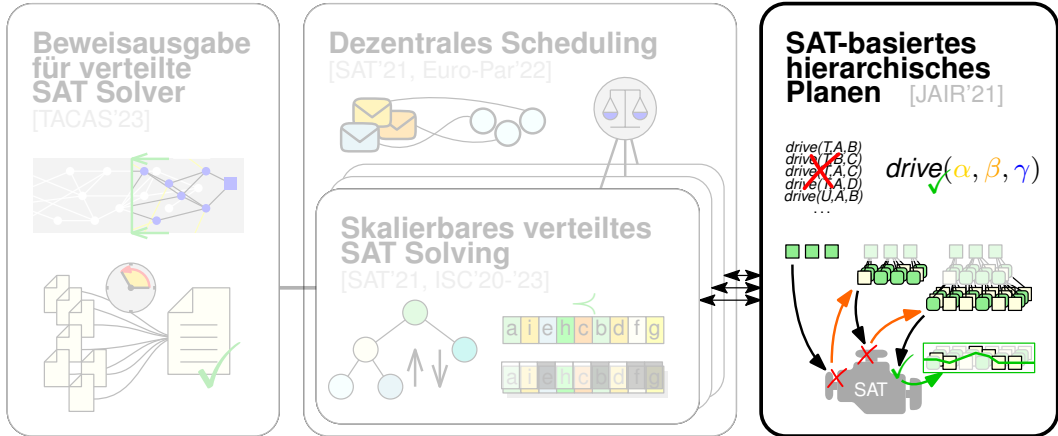
- **[64x]:** 135 Beweise rekonstruiert & geprüft  
in  $\approx 1.1 \times$  Solving-Zeit
- **[1600x]:** 154 Beweise rekonstruiert & geprüft  
in  $\approx 4.8 \times$  Solving-Zeit
- Vorheriger Ansatz (64x) [Heule et al. 2014]:  
81 Beweise geprüft mit Mehraufwand  $10.4 \times$

## Fazit

Erster praktikabler und skalierbarer Ansatz für verteilte Beweiskonstruktion mit Klauselaustausch

AWS Cloud · 400 Probleme aus SAT Comp. 2022

# Übersicht



# Hierarchisches Planen

$$t_1(A, B)$$

$$t_2(C)$$

$$t_3(E, F, G)$$

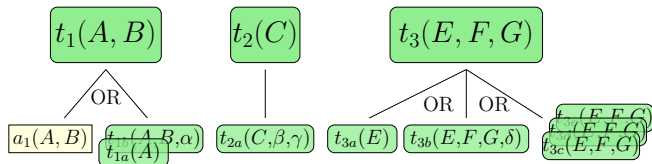
**Problemstellung** [Bercher et al. 2019]

- Erledige Sequenz von **Aufgaben** ...

# Hierarchisches Planen

**Problemstellung** [Bercher et al. 2019]

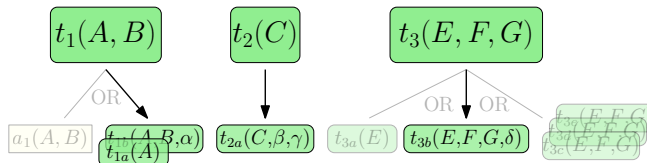
- Erledige Sequenz von **Aufgaben** ...
- durch **Zerlegen** jeder Aufgabe ...



# Hierarchisches Planen

**Problemstellung** [Bercher et al. 2019]

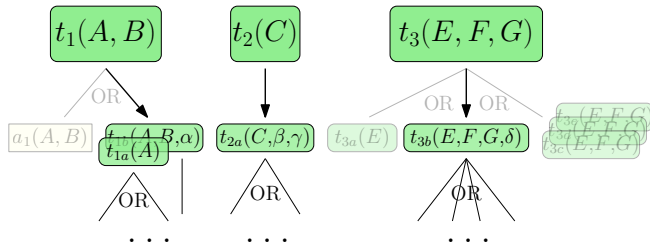
- Erledige Sequenz von **Aufgaben** ...
- durch **Zerlegen** jeder Aufgabe ...



# Hierarchisches Planen

**Problemstellung** [Bercher et al. 2019]

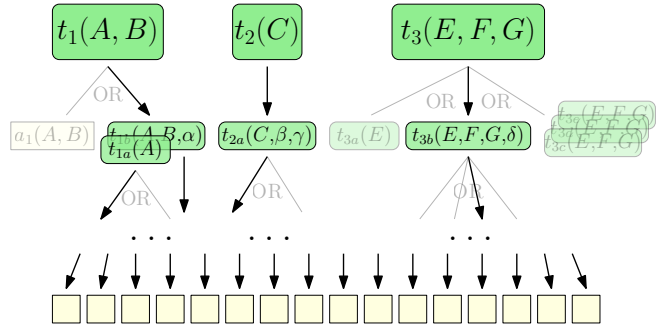
- Erledige Sequenz von **Aufgaben** ...
- durch **Zerlegen** jeder Aufgabe ...



# Hierarchisches Planen

## Problemstellung [Bercher et al. 2019]

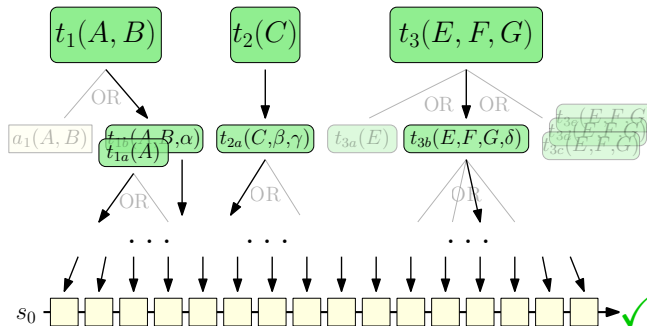
- Erledige Sequenz von **Aufgaben** ...
- durch **Zerlegen** jeder Aufgabe ...
- bis nur **primitive Aufgaben** verbleiben ...



# Hierarchisches Planen

## Problemstellung [Bercher et al. 2019]

- Erledige Sequenz von **Aufgaben** ...
- durch **Zerlegen** jeder Aufgabe ...
- bis nur **primitive Aufgaben** verbleiben ...
- die von **Startzustand** aus **ausführbar** sind.

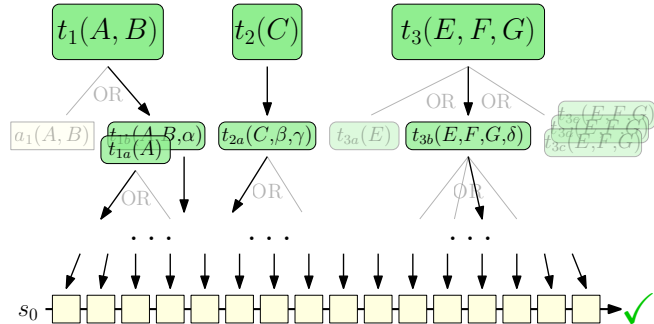




# Hierarchisches Planen

## Problemstellung [Bercher et al. 2019]

- Erledige Sequenz von **Aufgaben** ...
- durch **Zerlegen** jeder Aufgabe ...
- bis nur **primitive Aufgaben** verbleiben ...
- die von **Startzustand** aus **ausführbar** sind.



## Vorherige SAT-basierte Ansätze [ICAPS'19; Behnke et al. 2018]

- **Kodierte Hierarchie** bis zu **beschränkter Tiefe** — erhöhe Tiefe bis Problem **erfüllbar** wird
- Teures Ausinstanzieren (**Grounding**) aller parametrisierten Objekte  $\Rightarrow$  **kombinatorische Explosion**

# Unser Ansatz: Lilotane [JAIR'21]

## Zentrale Merkmale

- **Kein Grounding**: Parameter werden in Aussagenlogik kodiert

~~$drive(T,A,B)$   
 $drive(T,B,C)$   
 $drive(T,A,C)$   
 $drive(T,A,D)$   
 $drive(U,A,B)$   
...~~

$drive(\alpha, \beta, \gamma)$

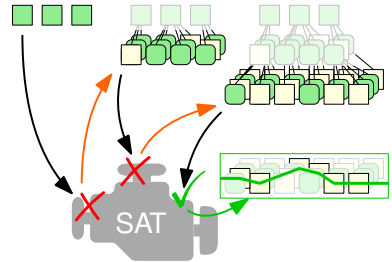
# Unser Ansatz: Lilotane [JAIR'21]

## Zentrale Merkmale

- **Kein Grounding**: Parameter werden in Aussagenlogik kodiert
- **Inkrementelles SAT Solving**: Wiederholte Interaktion mit einer **einzigsten Solver-Instanz** [ICAPS'19]

~~$drive(T,A,B)$~~   
 ~~$drive(T,B,C)$~~   
 ~~$drive(T,A,C)$~~   
 ~~$drive(T,A,D)$~~   
 ~~$drive(U,A,B)$~~   
...

$drive(\alpha, \beta, \gamma)$



# Unser Ansatz: Lilotane [JAIR'21]

## Zentrale Merkmale

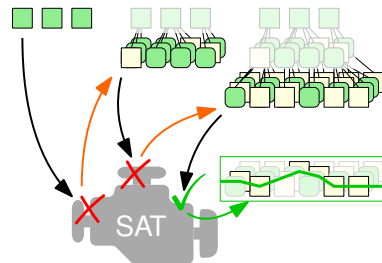
- **Kein Grounding**: Parameter werden in Aussagenlogik kodiert
- **Inkrementelles SAT Solving**: Wiederholte Interaktion mit einer **einzigsten Solver-Instanz** [ICAPS'19]

## Vergleich mit vorherigem Stand der Technik

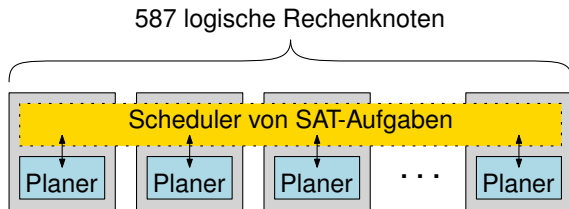
- Kleinere Formeln, oftmals um **1-2 Größenordnungen**
- **Effizientes Planen**
  - 2. Platz in Int. Planning Competition 2020
  - 2× unabhängige Nutzung in IPC 2023

~~$drive(T,A,B)$~~   
 ~~$drive(T,B,C)$~~   
 ~~$drive(T,A,C)$~~   
 ~~$drive(T,A,D)$~~   
 ~~$drive(U,A,B)$~~   
...

$drive(\alpha, \beta, \gamma)$



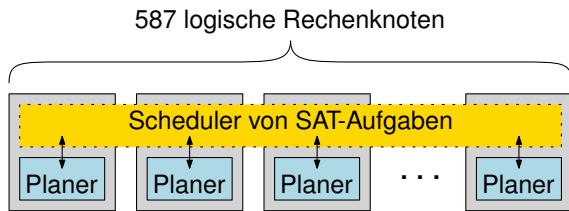
# Planung × Verteiltes SAT Solving × Scheduling



SuperMUC-NG · 2346 Kerne insgesamt

1 Kern pro Planer-Instanz,  
3 Kerne pro Mallob-Prozess

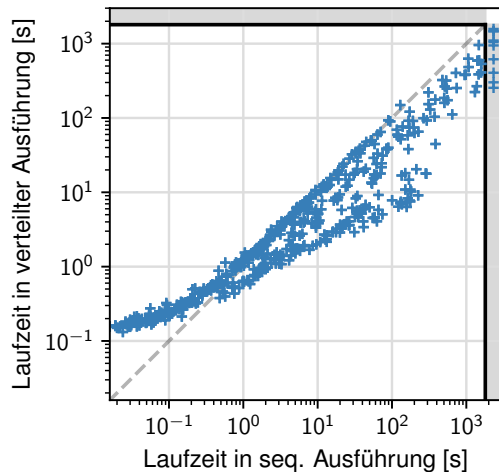
# Planung × Verteiltes SAT Solving × Scheduling



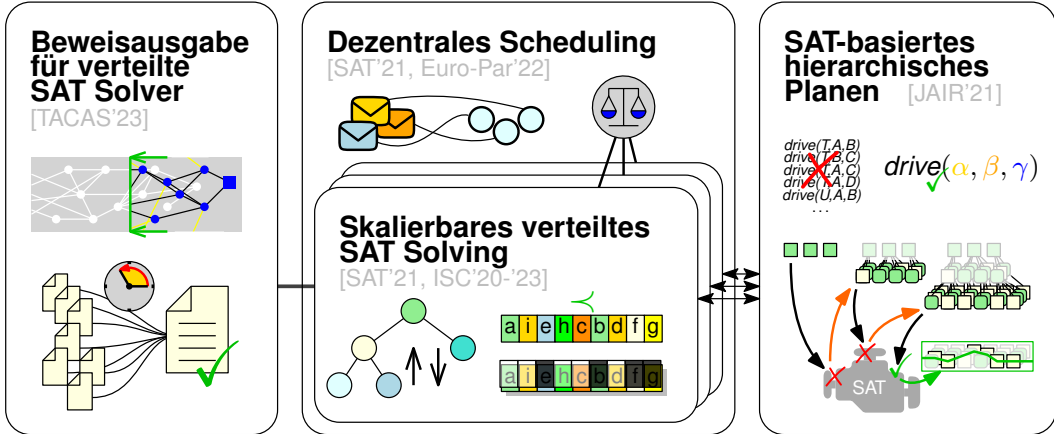
SuperMUC-NG · 2346 Kerne insgesamt

1 Kern pro Planer-Instanz,  
3 Kerne pro Mallob-Prozess

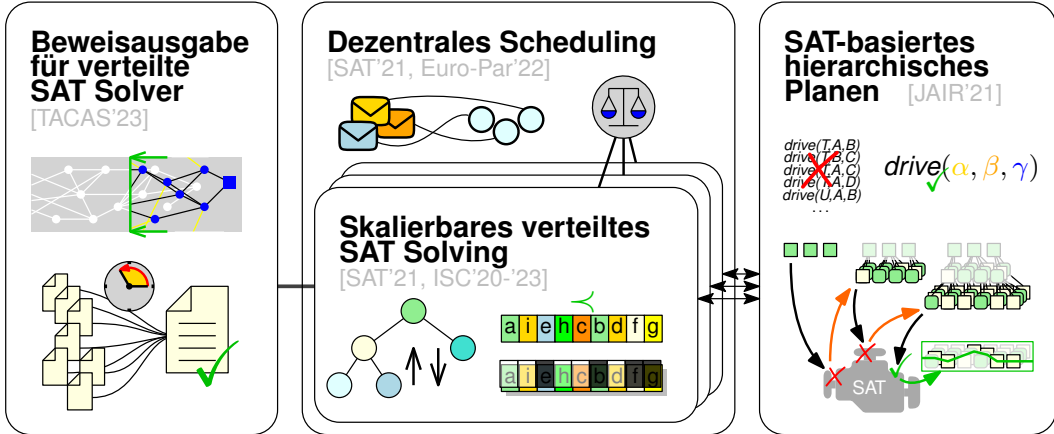
Animation: <https://dominikschreiber.de/animallob>



# Zusammenfassung



# Zusammenfassung



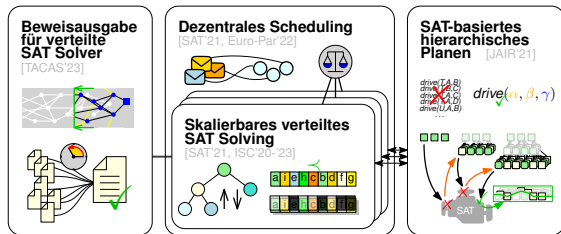
⇒ Horizont der praktisch & zuverlässig lösbaren Probleme **erfolgreich erweitert**.



# Abschluss

## Ausblick

- Nutzen-maximierende Algorithmen für Ressourcenzuteilung
- Verringerung **redundanter Arbeit** zwischen sequentiellen Solvern
- **Heterogene Systeme** (v.a. GPUs)



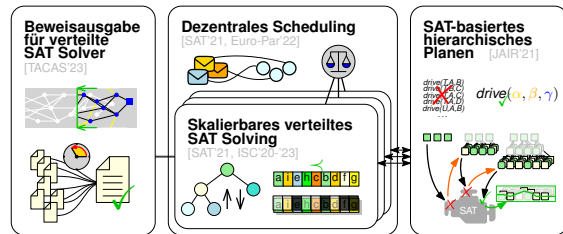
# Abschluss

## Ausblick

- Nutzen-maximierende Algorithmen für Ressourcenzuteilung
- Verringerung **redundanter Arbeit** zwischen sequentiellen Solvern
- **Heterogene Systeme** (v.a. GPUs)

## Einbezogene Publikationen

- [Euro-Par'22] Sanders, Schreiber: *Decentralized Job Scheduling of Malleable NP-hard Jobs* \*
- [SAT'21] Schreiber, Sanders: *Scalable SAT Solving in the Cloud*
- [TACAS'23] Michaelson, Schreiber et al.: *Unsatisfiability Proofs for Distributed Clause-Sharing Solvers* \*
- [JAIR'21] Schreiber: *Lilotane: A Lifted SAT-based Approach to Hierarchical Planning* (auch auf IJCAI'21)
- [ISC'20-23] Technische Berichte für Int. SAT Competitions



## Weitere Publikationen

- [ICAART'19, ICAPS'19] SAT-basiertes hierarchisches Planen mit Grounding
- [SoCS'19] Klassisches Planen mit SAT-basierten Techniken
- [HPCSE'21] Fortschrittsbericht zu Hochleistungsrechnen
- [JOSS'22] Artikel zu MALLOB(SAT) in Software-Journal
- [2xSparkle'19, 2xIPC'20] Diverse Technische Berichte

\* Für **Best Paper Award** nominiert

## Referenzen

- Behnke, Gregor, Daniel Höller, and Susanne Biundo. “totSAT – Totally-ordered hierarchical planning through SAT.” AAAI 2018.
- Bercher, Pascal, Ron Alford, and Daniel Höller. “A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations.” IJCAI 2019.
- Audemard, Gilles and Laurent Simon. “Predicting learnt clauses quality in modern SAT solvers.” IJCAI 2009.
- Balyo, Tomáš, Peter Sanders, and Carsten Sinz. “Hordesat: A massively parallel portfolio SAT solver.” SAT 2015.
- Biere, Armin. “CaDiCaL, Lingeling, Plingeling, Treengeling and YaSAT entering the SAT competition 2018.” SAT Competition 2018.
- Biere, Armin, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020.” SAT Competition 2020.
- Cook, Stephen. “The complexity of theorem proving procedure.” 3rd Symp. on Theory of Computing (1971).
- Feitelson, Dror G. “Job scheduling in multiprogrammed parallel systems.” IBM Research Report (1997).
- Gao, Yu. “Kissat MAB prop in SAT Competition 2023.” SAT Competition 2023.
- Hamadi, Youssef, Said Jabbour, and Lakhdar Sais. “ManySAT: a parallel SAT solver”. JSAT (2010).
- Heule, Marijn J. H., Norbert Manthey, and Tobias Philipp. “Validating Unsatisfiability Results of Clause Sharing Parallel SAT Solvers.” POS 2014.
- Zheng, Jiongzhi, Kun He, Zhuo Chen, et al. “Combining Hybrid Walking Strategy with Kissat MAB, CaDiCaL, and LStech-Maple.” SAT Competition 2022.

# Bildquellen

- 2 · Schaltkreis: <https://www.rawpixel.com/image/5907876/photo-image-background-public-domain-technology>
- 2 · Planung/Scheduling: <https://www.pexels.com/photo/blue-printer-paper-7376/>
- 2 · Kryptographie: <https://pixabay.com/vectors/computer-encrypt-encryption-1294045/>
- 2 · Gefärbtes Gitter:  
<https://www.quantamagazine.org/the-number-15-describes-the-secret-limit-of-an-infinite-grid-20230420/>
- 2 · Debugging: <https://technofaq.org/posts/2017/12/heres-everything-you-need-to-know-about-software-testing/>

# Anhang

(deutsch / englisch)

# Problemstellung

## Zentrale Problemstellung

Wie können wir den Horizont der praktisch lösbaren SAT-Instanzen erweitern?

Wichtige Aspekte: [Effiziente Kodierung](#) von Problemen; Nutzung von [paralleler Hardware](#).

# Problemstellung

## Zentrale Problemstellung

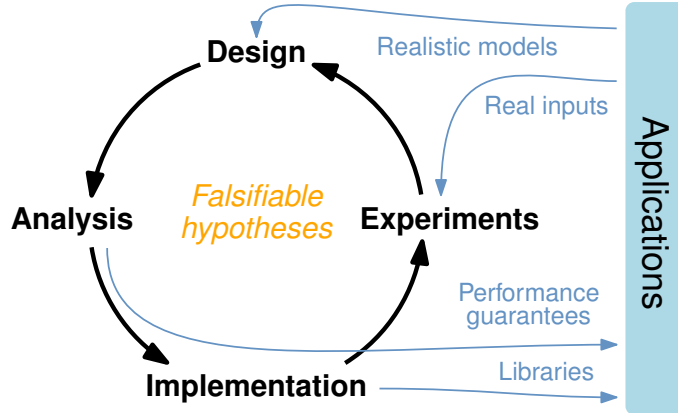
Wie können wir den Horizont der praktisch lösbaren SAT-Instanzen erweitern?

Wichtige Aspekte: **Effiziente Kodierung** von Problemen; Nutzung von **paralleler Hardware**.

### Leitfragen:

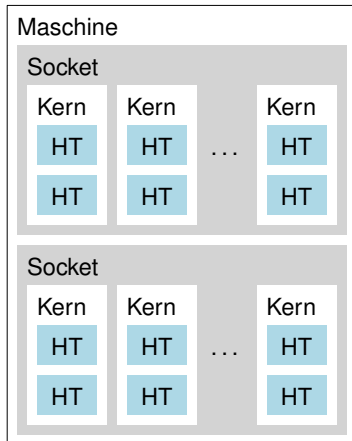
- 1 Wie können wir **moderne verteilte Rechenumgebungen** effizient für SAT Solving nutzen?
- 2 Wie können wir Systeme für SAT Solving in derartigen Umgebungen vollständig **vertrauenswürdig** machen, um deren Einsatz für **kritische Anwendungen** zu ermöglichen?
- 3 Wie können **komplexe Anwendungen** effizienteren Gebrauch von SAT-Solvern machen, damit zuvor unlösbare Probleme bewältigt werden können?

# Methodology: Algorithm Engineering

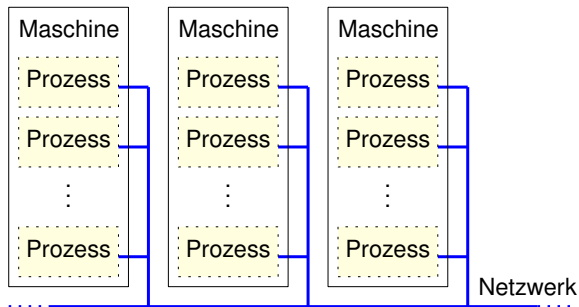
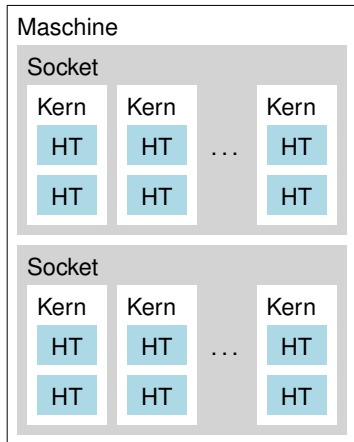




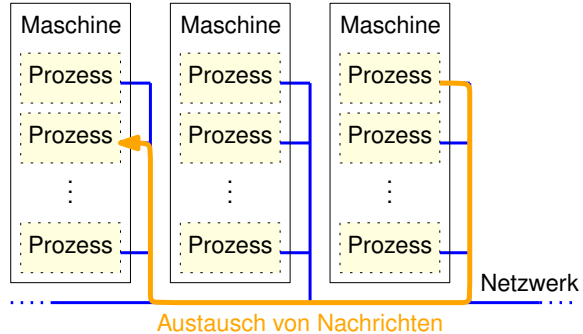
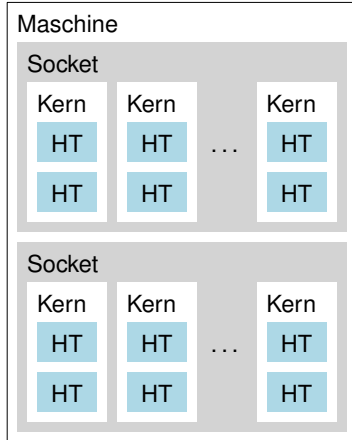
# Grundlage: Rechenumgebung



# Grundlage: Rechenumgebung



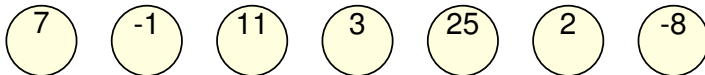
# Grundlage: Rechenumgebung



## Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  *span / depth*:

- **All-reduction** – e.g., broadcast value, compute maximum



## Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  *span* / *depth*:

- **All-reduction** – e.g., broadcast value, compute maximum



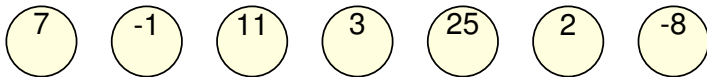
## Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  *span / depth*:

- **All-reduction** – e.g., broadcast value, compute maximum



- **Sorting**  $\mathcal{O}(m)$  scattered elements [Ajtai, Komlós, Szemerédi '83]



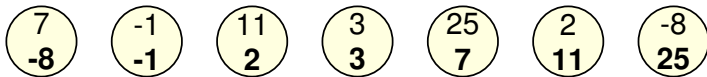
# Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  *span / depth*:

- **All-reduction** – e.g., broadcast value, compute maximum



- **Sorting**  $\mathcal{O}(m)$  scattered elements [Ajtai, Komlós, Szemerédi '83]



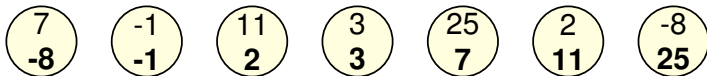
# Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  *span / depth*:

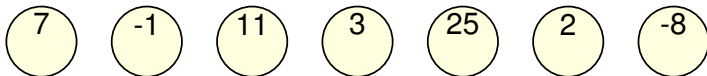
- **All-reduction** – e.g., broadcast value, compute maximum



- **Sorting**  $\mathcal{O}(m)$  scattered elements [Ajtai, Komlós, Szemerédi '83]



- **Prefix sum** (or “Scan”)





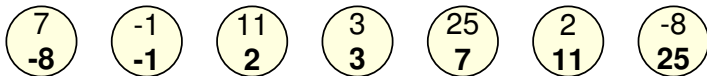
# Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  span / depth:

- **All-reduction** – e.g., broadcast value, compute maximum



- **Sorting**  $\mathcal{O}(m)$  scattered elements [Ajtai, Komlós, Szemerédi '83]



- **Prefix sum** (or “Scan”)



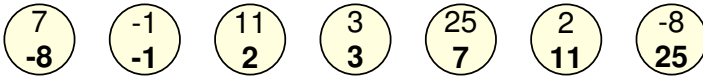
# Prerequisites: Communication Primitives

Collective operations with  $\mathcal{O}(\log m)$  span / depth:

- **All-reduction** – e.g., broadcast value, compute maximum



- **Sorting**  $\mathcal{O}(m)$  scattered elements [Ajtai, Komlós, Szemerédi '83]



- **Prefix sum** (or “Scan”)



# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

(C2) Each job has **at least one** worker and **at most**  $d_j$  workers.  $\forall j \in J : 1 \leq v_j \leq d_j$

# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

(C2) Each job has **at least one** worker and **at most**  $d_j$  workers.  $\forall j \in J : 1 \leq v_j \leq d_j$

(C3) The volume of each job  $j$  scales **proportionally with**  $p_j$  as far as (C2) allows.

# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

(C2) Each job has **at least one** worker and **at most**  $d_j$  workers.  $\forall j \in J : 1 \leq v_j \leq d_j$

(C3) The volume of each job  $j$  scales **proportionally with**  $p_j$  as far as (C2) allows.

## Theorem

Solving the above problem on  $m$  processes for  $n \leq m$  jobs is possible in  $\mathcal{O}(\log m)$  span.

# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

(C2) Each job has **at least one** worker and **at most**  $d_j$  workers.  $\forall j \in J : 1 \leq v_j \leq d_j$

(C3) The volume of each job  $j$  scales **proportionally with**  $p_j$  as far as (C2) allows.

## Theorem

Solving the above problem on  $m$  processes for  $n \leq m$  jobs is possible in  $\mathcal{O}(\log m)$  span.

## Central Idea

- Express job volumes  $v_j = v_j(\alpha) := \alpha p_j$ ,  $\alpha \geq 0$



# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

(C2) Each job has **at least one** worker and **at most**  $d_j$  workers.  $\forall j \in J : 1 \leq v_j \leq d_j$

(C3) The volume of each job  $j$  scales **proportionally with**  $p_j$  as far as (C2) allows.

## Theorem

Solving the above problem on  $m$  processes for  $n \leq m$  jobs is possible in  $\mathcal{O}(\log m)$  span.

## Central Idea

- Express job volumes  $v_j = v_j(\alpha) := \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$

# Assigning Fair Volumes to Jobs

## Volume Assignment Problem

Map each job  $j \in J$  to a fair number of workers, the **volume**  $v_j \in \mathbb{N}^+$ , s.t.

(C1) All job demands are fully met **or** all  $m$  processes are utilized.

$$(\forall j \in J : v_j = d_j) \quad \vee \quad \sum_{j \in J} v_j = m$$

(C2) Each job has **at least one** worker and **at most**  $d_j$  workers.  $\forall j \in J : 1 \leq v_j \leq d_j$

(C3) The volume of each job  $j$  scales **proportionally with**  $p_j$  as far as (C2) allows.

## Theorem

Solving the above problem on  $m$  processes for  $n \leq m$  jobs is possible in  $\mathcal{O}(\log m)$  span.

## Central Idea

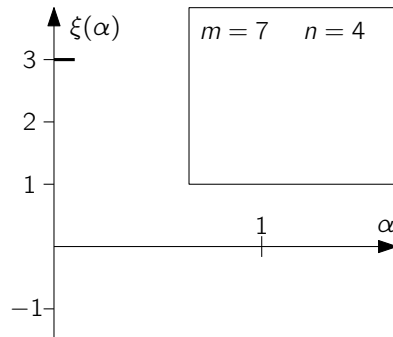
- Express job volumes  $v_j = v_j(\alpha) := \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Solve unused resources  $\xi(\alpha) := m - \sum_{j \in J} v_j(\alpha)$  for  $\xi(\alpha) = 0$

# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$  ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$

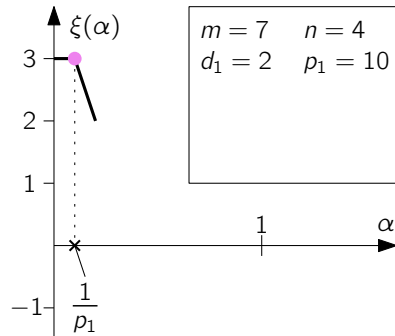
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$



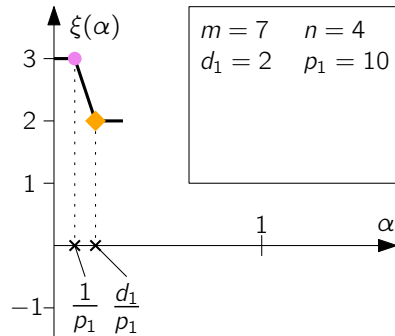
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$



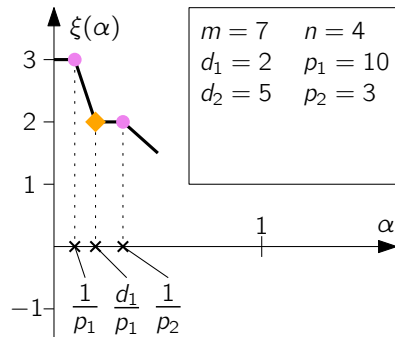
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$



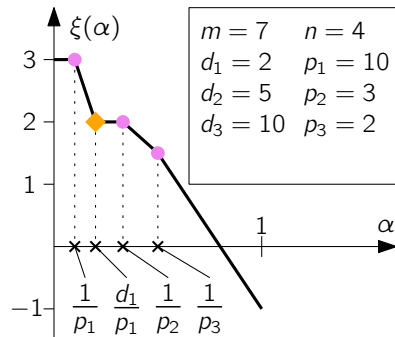
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$



# Volume Assignment Algorithm

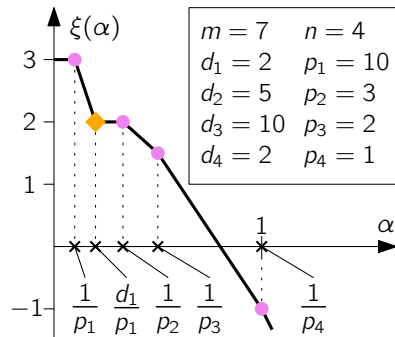
- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$





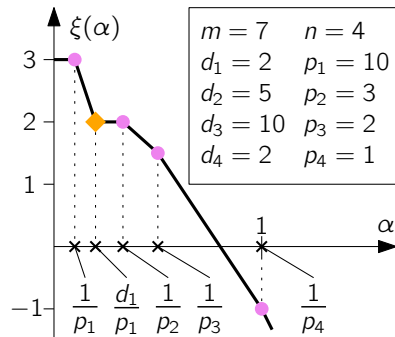
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$



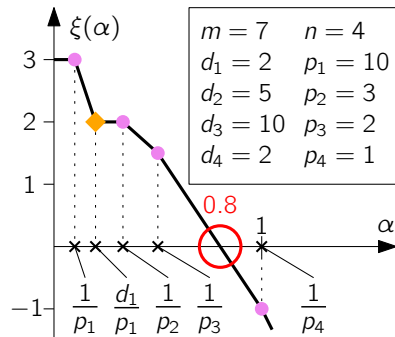
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes



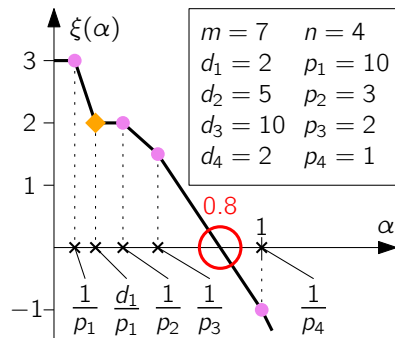
# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$



# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$



$$v_1(\hat{\alpha}) = d_1 = 2$$

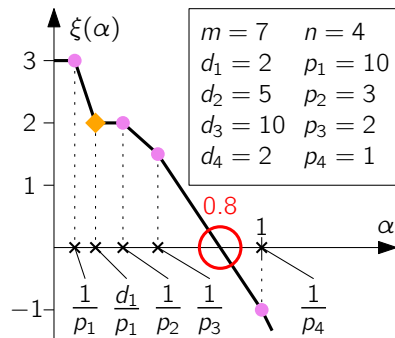
$$v_2(\hat{\alpha}) = 0.8 \cdot 3 = 2.4$$

$$v_3(\hat{\alpha}) = 0.8 \cdot 2 = 1.6$$

$$v_4(\hat{\alpha}) = 1$$

# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$
- Round the  $v_j(\hat{\alpha})$  to appropriate integers



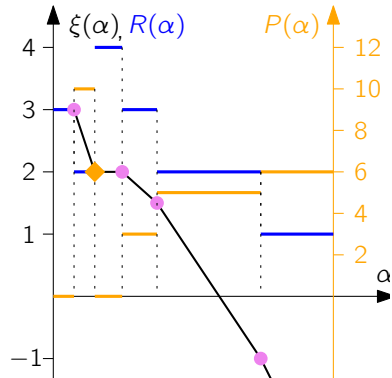
$$\begin{aligned}
 v_1(\hat{\alpha}) &= d_1 = 2 & v_1 &= 2 \\
 v_2(\hat{\alpha}) &= 0.8 \cdot 3 = 2.4 & v_2 &= 3 \\
 v_3(\hat{\alpha}) &= 0.8 \cdot 2 = 1.6 & v_3 &= 1 \\
 v_4(\hat{\alpha}) &= 1 & v_4 &= 1
 \end{aligned}$$

# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$
- Round the  $v_j(\hat{\alpha})$  to appropriate integers

## \*Parallel evaluation of $\xi$ :

- Compute  $\xi$  based on auxiliary terms  $R$  and  $P$

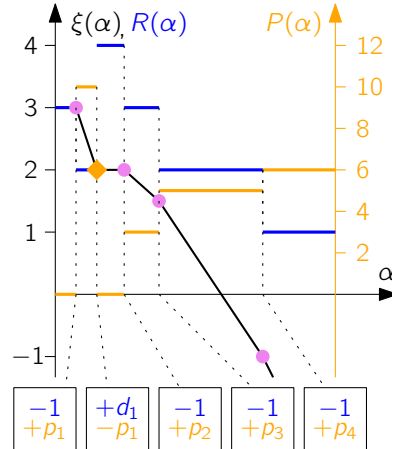


# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$
- Round the  $v_j(\hat{\alpha})$  to appropriate integers

## \*Parallel evaluation of $\xi$ :

- Compute  $\xi$  based on auxiliary terms  $R$  and  $P$
- Create, sort *events* which manipulate  $R$  and  $P$

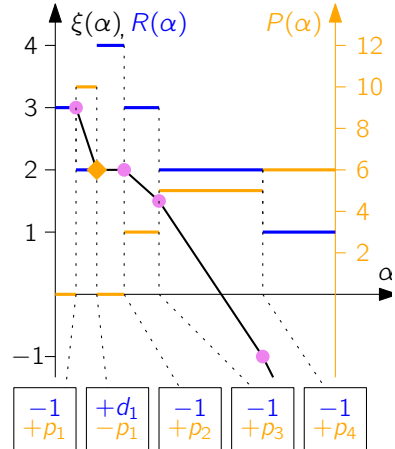


# Volume Assignment Algorithm

- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$
- Round the  $v_j(\hat{\alpha})$  to appropriate integers

## \*Parallel evaluation of $\xi$ :

- Compute  $\xi$  based on auxiliary terms  $R$  and  $P$
- Create, sort *events* which manipulate  $R$  and  $P$
- Compute *all intermediate values* of  $R$ ,  $P$  with prefix sum





# Volume Assignment Algorithm

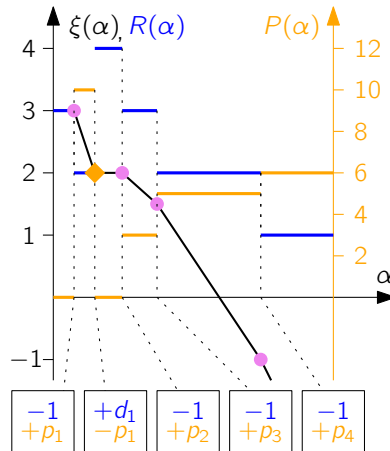
- Job volumes  $v_j = v_j(\alpha) = \max(1, \min(d_j, \alpha p_j))$ ,  $\alpha \geq 0$
- Excess resources  $\xi(\alpha) = m - \sum_{j \in J} v_j(\alpha)$
- Evaluate  $\xi$  in parallel\* at all  $2n$  values where  $\xi'$  changes
- Interpolate value  $\hat{\alpha}$  where  $\xi(\hat{\alpha}) = 0$
- Round the  $v_j(\hat{\alpha})$  to appropriate integers

## \*Parallel evaluation of $\xi$ :

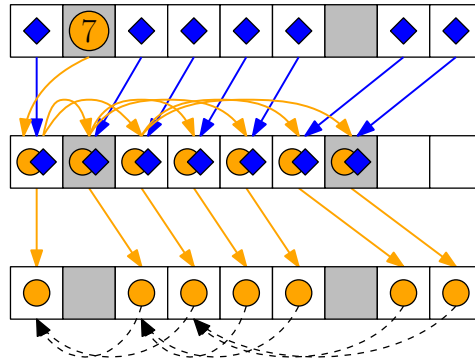
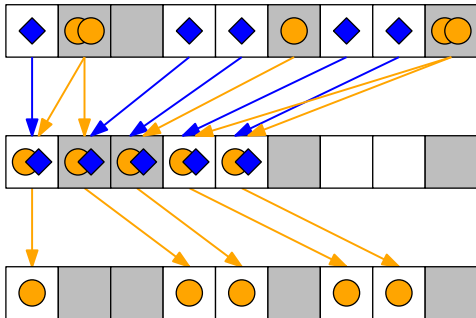
- Compute  $\xi$  based on auxiliary terms  $R$  and  $P$
- Create, sort *events* which manipulate  $R$  and  $P$
- Compute *all intermediate values* of  $R$ ,  $P$  with prefix sum

All-reductions, prefix sums, sorting  $\mathcal{O}(m)$  elements:

**Possible in  $\mathcal{O}(\log m)$  time** [Ajtai, Komlós, Szemerédi '83]

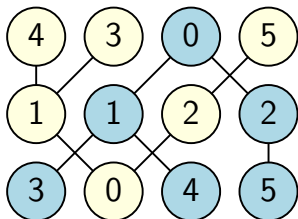


# Zuordnung von Anfragen via Präfixsummen



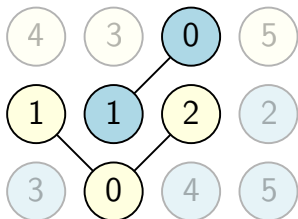
# Reusing Suspended Workers

## Naïve scheduling



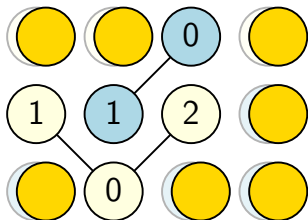
# Reusing Suspended Workers

## Naïve scheduling



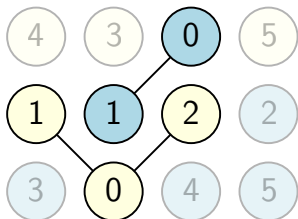
# Reusing Suspended Workers

## Naïve scheduling



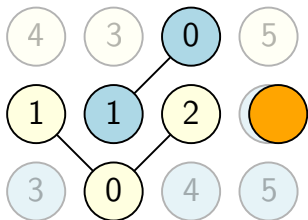
# Reusing Suspended Workers

## Naïve scheduling



# Reusing Suspended Workers

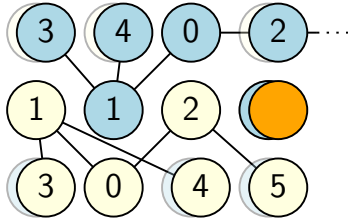
## Naïve scheduling



- Idle procs. can be seized by other jobs

# Reusing Suspended Workers

## Naïve scheduling

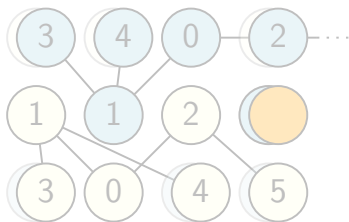


- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking



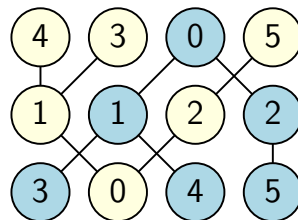
# Reusing Suspended Workers

## Naïve scheduling



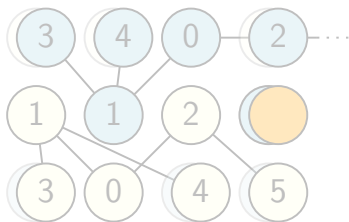
- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

## Improvements



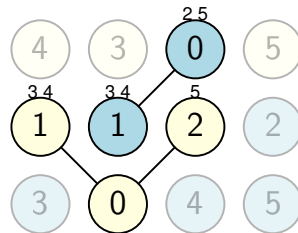
# Reusing Suspended Workers

## Naïve scheduling



- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

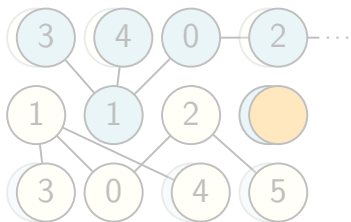
## Improvements



- Workers transitively remember past children

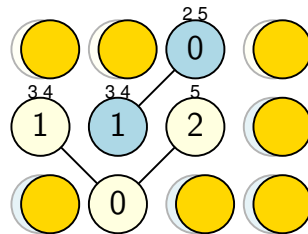
# Reusing Suspended Workers

## Naïve scheduling



- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

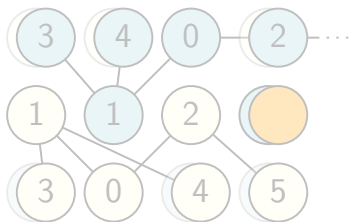
## Improvements



- Workers transitively remember past children

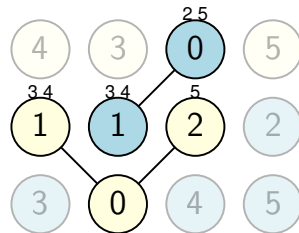
# Reusing Suspended Workers

## Naïve scheduling



- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

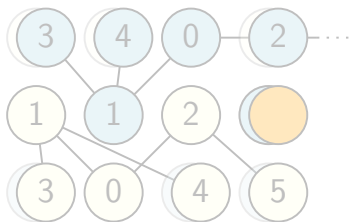
## Improvements



- Workers transitively remember past children

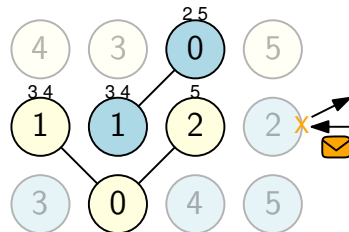
# Reusing Suspended Workers

## Naïve scheduling



- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

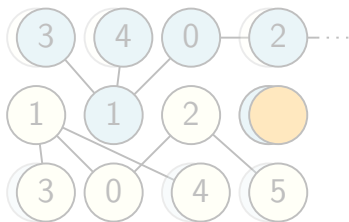
## Improvements



- Workers transitively remember past children
- Idle processes prefer to reactivate past child

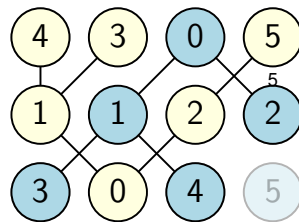
# Reusing Suspended Workers

## Naïve scheduling



- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

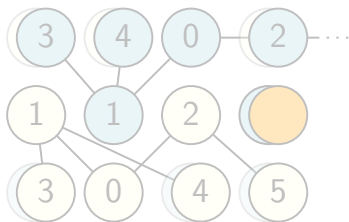
## Improvements



- Workers transitively remember past children
- Idle processes prefer to reactivate past child

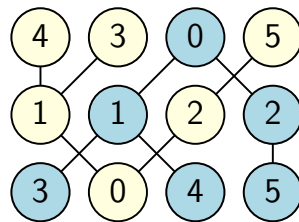
# Reusing Suspended Workers

## Naïve scheduling



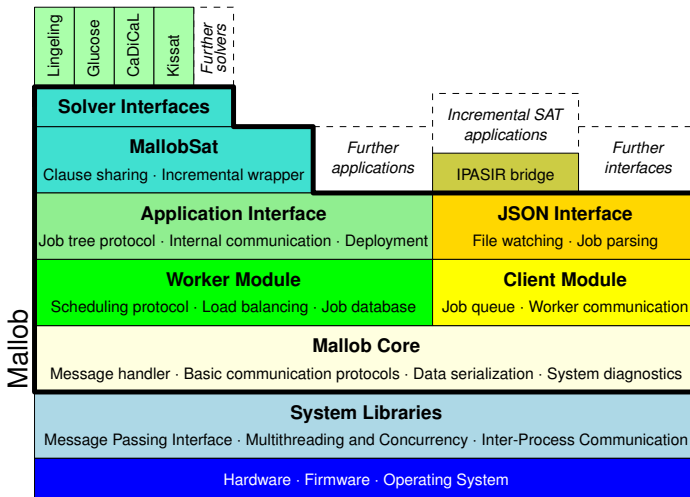
- Idle procs. can be seized by other jobs
- Jobs can re-grow differently after shrinking

## Improvements



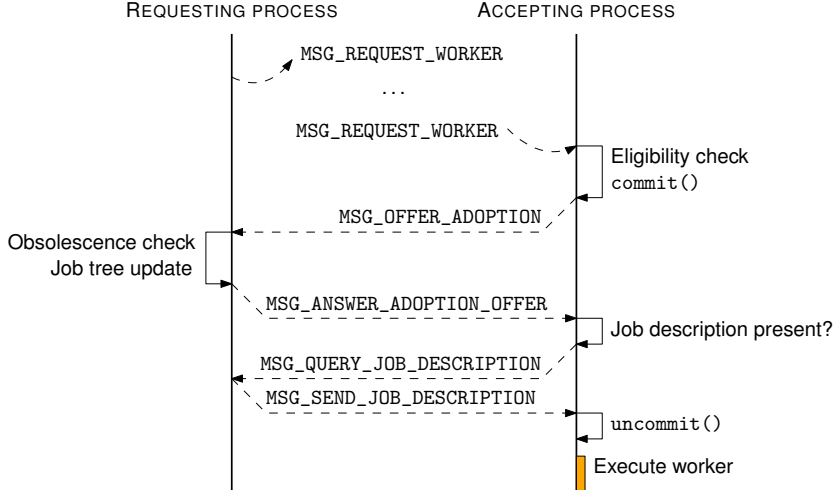
- Workers transitively remember past children
- Idle processes prefer to reactivate past child

# Mallob: Technology Stack

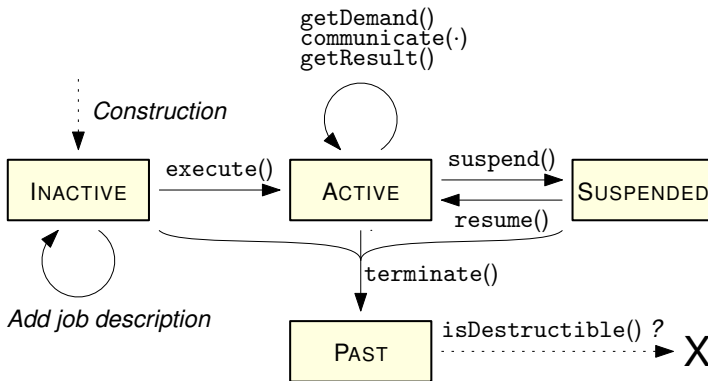




# Mallob: Example Protocol

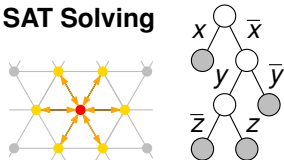


# Mallob: Worker Life Cycle



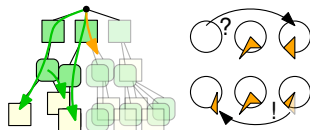
# Mallob: Anwendungsstudien

## SAT Solving



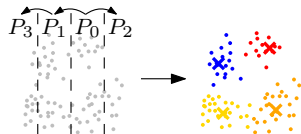
Malte Sönnichsen Maximilian Schick

## Hierarchisches Planen



Niko Wilhelm

## k-means Clustering

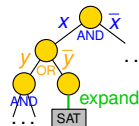


Michael Dörr

## QBF Solving (WIP)

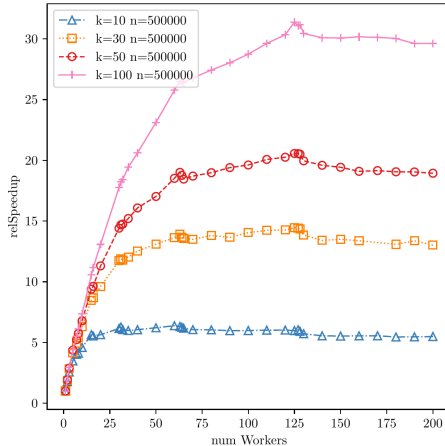
$$\forall x \exists y \forall z$$

$$F(x, y, z)$$

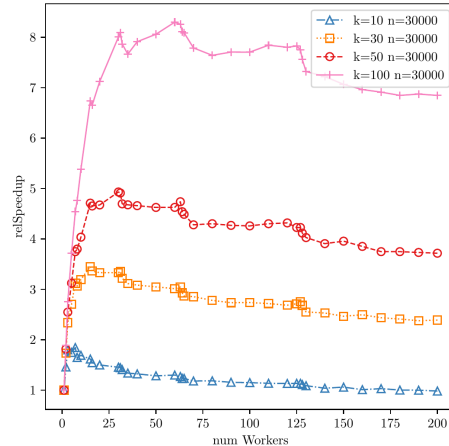


Kollab. Maximilian Heisinger

# Mallob: Fallstudie $k$ -means [Michael Dörr] (1/2)

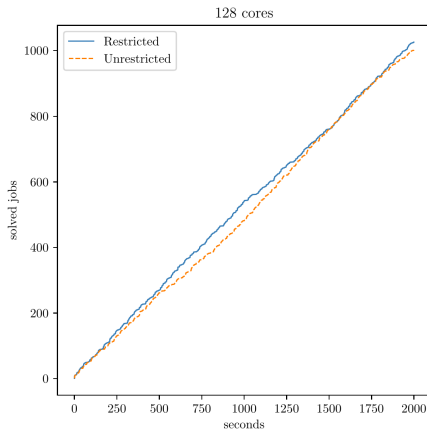


(a) 54 dimensional *covertype* self speedup

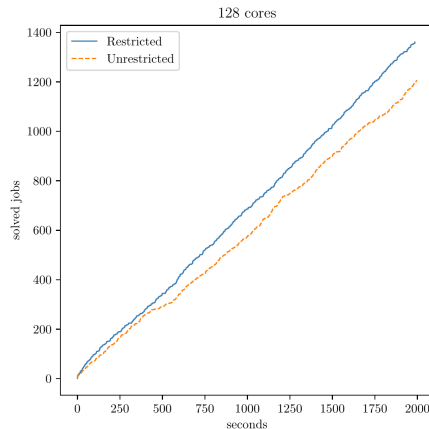


(b) 115 dimensional *benign\_traffic* self speedup

# Mallob: Fallstudie $k$ -means [Michael Dörr] (2/2)



**(a)** Solved jobs per time with maximum 8 concurrent jobs



**(b)** Solved jobs per time with maximum 6 concurrent jobs and more smaller sized jobs

# SuperMUC-NG

## Allgemein

- 311 040 Rechenkerne
- 719 TB Arbeitsspeicher
- Spitzenleistung: 26,9 PetaFLOPS
- SUSE Linux Enterprise Server 12 SP3

## 6 336 “dünne” Rechenknoten auf 8 Islands

- 2× Intel Skylake Xeon Platinum 8174 @ 2.7 GHz
- 2×24 = 48 **physische Kerne** (= 96 Hardwarethreads)
- 96 GB Arbeitsspeicher (= 1 GB pro Hardwarethread)
- Interconnect: Intel OmniPath, Topologie: Fat tree



31st @ TOP500  
(June 2023)

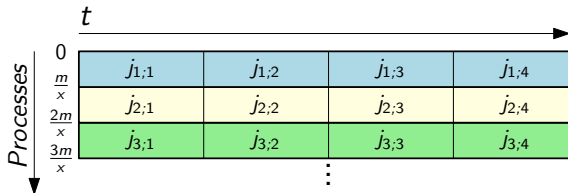
Leibniz Rechenzentrum

<https://doku.lrz.de/supermuc-ng-10745965.html>

# Efficiency for Uniform Jobs

**Setup:** 1536 processes  $\times$  four cores

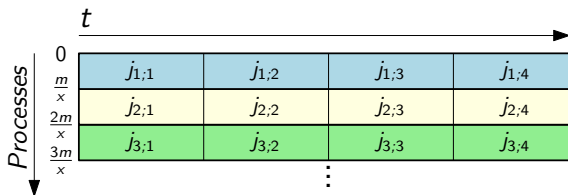
**Scenario:**  $x$  jobs in parallel with fixed CPU limit



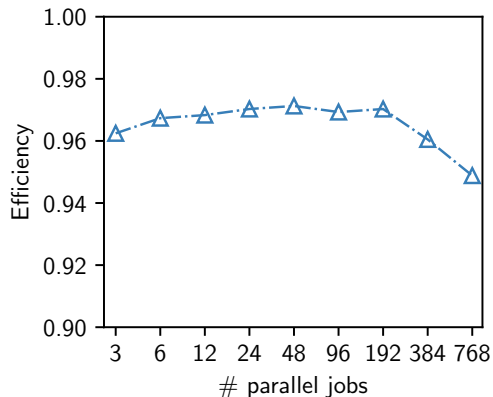
# Efficiency for Uniform Jobs

**Setup:** 1536 processes × four cores

**Scenario:**  $x$  jobs in parallel with fixed CPU limit



**Efficiency:** >96% for reasonable loads

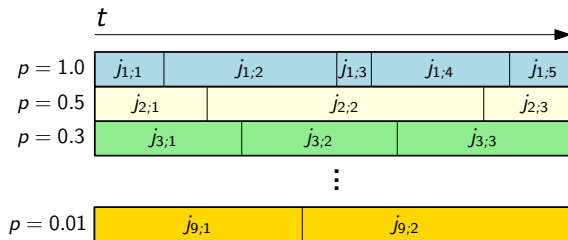




# Impact of Job Priorities

**Setup:** 384 processes  $\times$  four cores

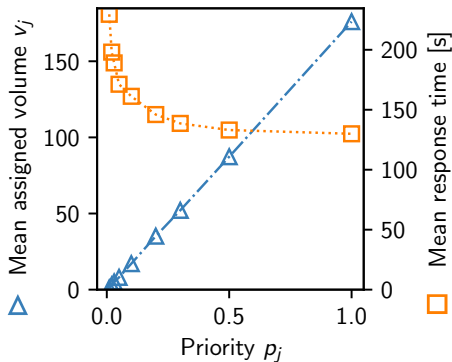
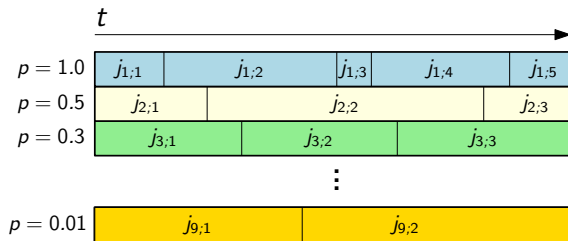
**Scenario:** 9 “clients” introducing jobs sequentially



# Impact of Job Priorities

**Setup:** 384 processes × four cores

**Scenario:** 9 “clients” introducing jobs sequentially



# Scheduling: Experimente [Euro-Par'22]+

## Durchschnittliche Latenzen

≈ 10 ms für Scheduling des 0. Arbeiters

≈ 1 ms für Berechnung fairer Volumen

≈ 6 ms für Verdoppelung der Arbeiter einer Aufgabe

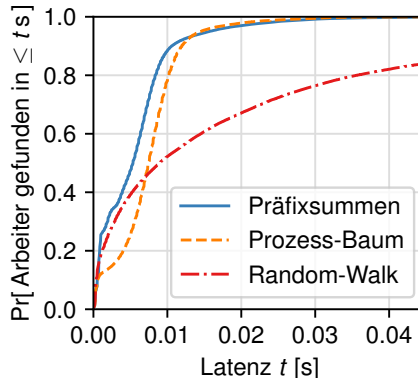
# Scheduling: Experimente [Euro-Par'22]+

## Durchschnittliche Latenzen

≈ 10 ms für Scheduling des 0. Arbeiters

≈ 1 ms für Berechnung fairer Volumen

≈ 6 ms für Verdoppelung der Arbeiter einer Aufgabe



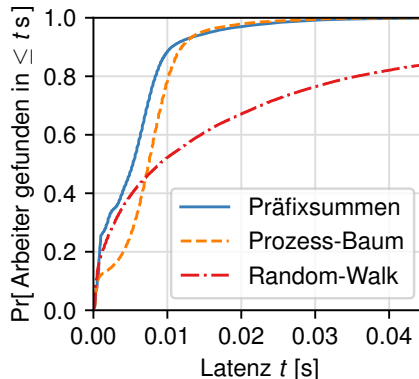
# Scheduling: Experimente [Euro-Par'22]+

## Durchschnittliche Latenzen

- ≈ 10 ms für Scheduling des 0. Arbeiters
- ≈ 1 ms für Berechnung fairer Volumen
- ≈ 6 ms für Verdoppelung der Arbeiter einer Aufgabe

## Stabilitäts-Metriken

Je Aufgabe werden **25%** mehr Arbeiter angelegt als mindestens nötig.



# Scheduling: Experimente [Euro-Par'22]+

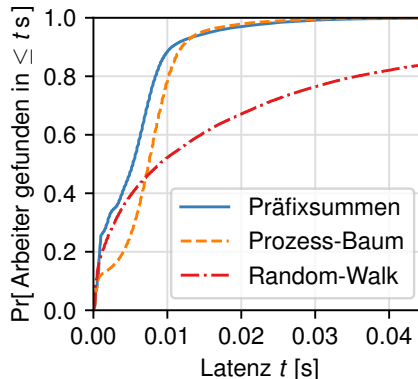
## Durchschnittliche Latenzen

- ≈ 10 ms für Scheduling des 0. Arbeiters
- ≈ 1 ms für Berechnung fairer Volumen
- ≈ 6 ms für Verdoppelung der Arbeiter einer Aufgabe

## Stabilitäts-Metriken

Je Aufgabe werden **25%** mehr Arbeiter angelegt als mindestens nötig.

**89%** aller Arbeiter  $w_j^i$  werden **nur ein mal initialisiert**.



# Realistic Arrival Rates: Worker Reuse

---

<b>Worker reuse strategy</b>	<u>Workers created</u> <u>Workers required</u>		
	median	max.	total
None	1.43	33.0	2.14
Basic [SAT'21]	1.40	31.5	2.07
Latest [Euro-Par'22]	<b>1.25</b>	<b>24.5</b>	<b>1.80</b>

---

# Realistic Arrival Rates: Worker Reuse

Worker reuse strategy	$\frac{\text{Workers created}}{\text{Workers required}}$			Pr [Worker created at $\leq X$ processes]				
	median	max.	total	1	2	5	10	25
None	1.43	33.0	2.14	0.87	0.90	0.94	0.97	0.992
Basic [SAT'21]	1.40	31.5	2.07	0.87	0.90	0.94	0.97	0.993
Latest [Euro-Par'22]	<b>1.25</b>	<b>24.5</b>	<b>1.80</b>	<b>0.89</b>	<b>0.91</b>	0.94	0.97	0.993



# Realistic Arrival Rates: Worker Reuse

Worker reuse strategy	Workers created Workers required			Pr [Worker created at $\leq X$ processes]				
	median	max.	total	1	2	5	10	25
None	1.43	33.0	2.14	0.87	0.90	0.94	0.97	0.992
Basic [SAT'21]	1.40	31.5	2.07	0.87	0.90	0.94	0.97	0.993
Latest [Euro-Par'22]	<b>1.25</b>	<b>24.5</b>	<b>1.80</b>	<b>0.89</b>	<b>0.91</b>	0.94	0.97	0.993

⇒ Most workers ( $\approx 90\%$ ) are **initialized only once**

# Benchmark-Instanzen: Anniversary'22

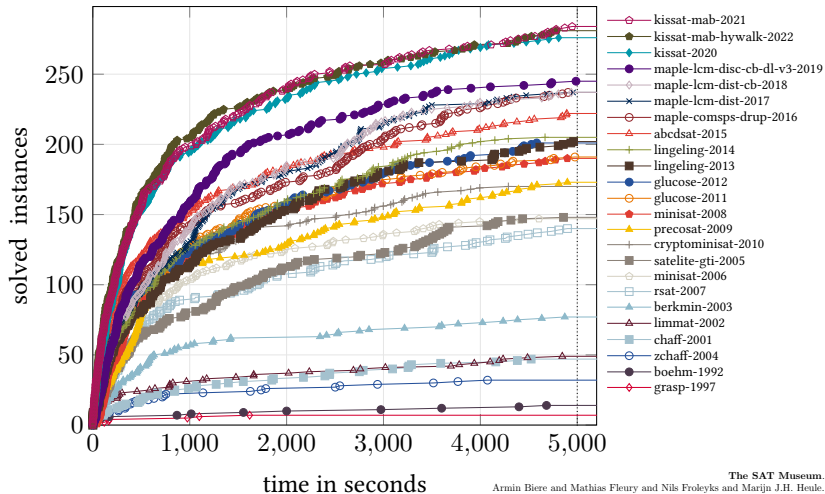


5355 Probleme,  
138 Familien + 72 unbekannt

Schriftgröße  $\propto$  # Probleme

via [wordclouds.com](https://wordclouds.com/),  
[GBD \(benchmark-database.de\)](https://github.com/benchmark-database)

## SAT Competition All Time Winners on SAT Competition 2022 Benchmarks



<https://cca.informatik.uni-freiburg.de/satmuseum>

The SAT Museum.  
 Armin Biere and Mathias Fleury and Nils Frolejks and Marijn J.H. Heule.  
 In *Proceedings 14th International Workshop on Pragmatics of SAT (POS'23)*,  
 vol. 3545, CEUR Workshop Proceedings, pages 72-87, CEUR-WS.org 2023.  
 [ paper - bibtex - data - zenodo - ceur - workshop - proceedings ]

# SAT Solving und Machine Learning: Schnittstellen

## ■ **Algorithmenselektion**

Xu, Lin, et al. “SATzilla: portfolio-based algorithm selection for SAT.” JAIR 2008.

Eggensperger, Katharina, Marius Lindauer, and Frank Hutter. “Neural networks for predicting algorithm runtime distributions.” IJCAI 2018.

## ■ **SAT Solving mit maschinellem Lernen**

Liang, Jia Hui, et al. “Learning rate based branching heuristic for SAT solvers.” SAT 2016.

Guo, Wenxuan, et al. “Machine learning methods in solving the boolean satisfiability problem.” Machine Intelligence Research (2023).

## ■ **Neuronale Netze per SAT/SMT Solving verifizieren**

Huang, Xiaowei, et al. “Safety verification of deep neural networks.” CAV 2017.

Ehlers, Ruediger. “Formal verification of piece-wise linear feed-forward neural networks.” ATVA 2017.

## ■ **Analysieren und Verstehen des Verhaltens von SAT Solvern und Instanzen**

Soos, Mate, Raghav Kulkarni, and Kuldeep S. Meel. “CrystalBall: gazing in the black box of SAT solving.” SAT 2019.

Fuchs, Tobias, Jakob Bach, and Markus Iser. “Active Learning for SAT Solver Benchmarking.” TACAS 2023.

# SAT Solving: Anwendungsbeispiele

## Synthese/Entwurf/Analyse von eingebetteten Systemen

- Design-Space-Exploration via **SAT-Decoding**

Lukasiewicz, Martin et al. “Sat-decoding in evolutionary algorithms for discrete constrained optimization problems.” IEEE Congress on Evolutionary Computation, 2007.

- Rekonstruktion von **verschleierte Schaltkreise**

Yu, Cunxi, et al. “Incremental SAT-based reverse engineering of camouflaged logic circuits.” IEEE Tr. Computer-Aided Design of Integrated Circuits and Systems, 2017.

- Testen von **Hardware-Neural-Networks**

Gebregiorgis, Anteneh, and Mehdi B. Tahoori. “Testing of neuromorphic circuits: Structural vs functional.” ITC 2019.

## Kryptographie

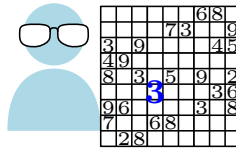
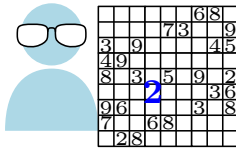
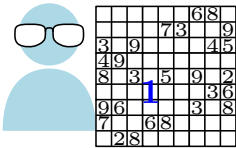
- **Urbild-Angriffe** auf Hashfunktionen, Schwachstellen in Blockziffern

Lafitte, Frédéric et al. “Applications of SAT solvers in cryptanalysis: finding weak keys and preimages.” JSAT, 2014.

- Analyse, Kontrolle, Verifikation von **kryptographischen Bausteinen**, z.B. Streamziffern

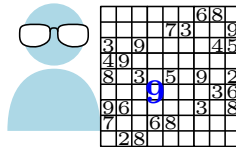
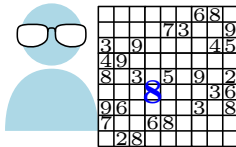
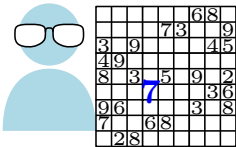
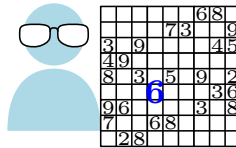
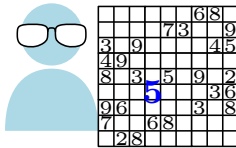
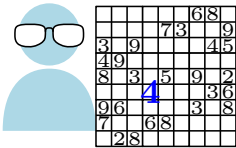
Soos, Mate et al. “Extending SAT solvers to cryptographic problems.” SAT, 2009.

# Paralleles Logisches Schließen

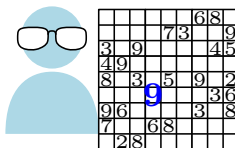
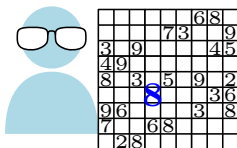
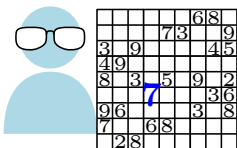
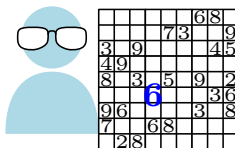
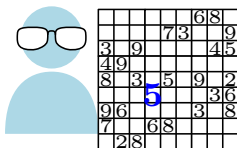
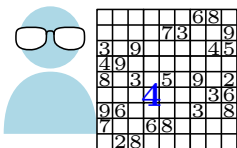
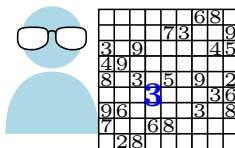
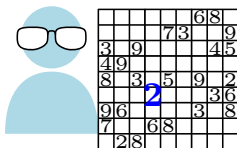
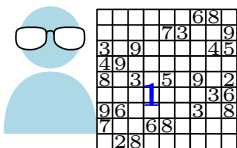


## Explizite Suchraum-Unterteilung

- Partitioniere Suchraum entlang einiger Entscheidungen  
⇒ Unabhängige Teilprobleme



# Paralleles Logisches Schließen



## Explizite Suchraum-Unterteilung

- Partitioniere Suchraum entlang einiger Entscheidungen  
⇒ Unabhängige Teilprobleme
- Schwierige Lastverteilung!

*Beispiele aus SAT Solving:*

- PSATO, PaSAT, MiraXT
- Treengeling
- Paracooba

# Verteiltes SAT Solving: Stand der Technik

**Beste Ansatz:** Portfolio verschiedener Solver

- Seq. Algorithmen produzieren Konflikt-Klauseln
- Solver tauschen Klauseln aus



# Verteiltes SAT Solving: Stand der Technik

**Beste Ansatz:** Portfolio verschiedener Solver

- Seq. Algorithmen produzieren Konflikt-Klauseln
- Solver tauschen Klauseln aus

## HordeSat [Balyo et al. 2015]

- Orchestriert Hunderte von sequentiellen Solvern
- Periodischer Klauselaustausch
  - Konkatenation von Klausel-Buffern fester Größe
  - Duplikate, ungenutzter Platz in Buffern

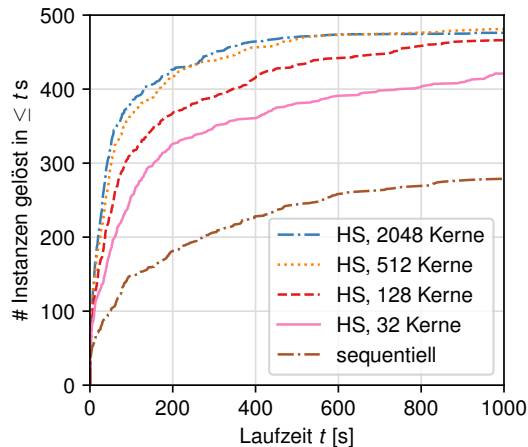
# Verteiltes SAT Solving: Stand der Technik

**Beste Ansatz:** Portfolio verschiedener Solver

- Seq. Algorithmen **produzieren Konflikt-Klauseln**
- Solver **tauschen Klauseln aus**

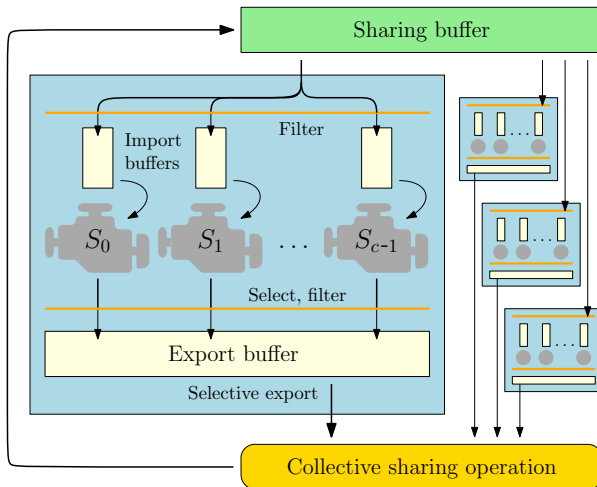
## HordeSat [Balyo et al. 2015]

- Orchestriert Hunderte von sequentiellen Solvern
- Periodischer **Klauselaustausch**
  - Konkatenation von Klausel-Buffern fester Größe
  - **Duplikate, ungenutzter Platz** in Buffern
- Experimente mit bis zu **2048 Kernen**
  - Median Speedup auf 2048 Kernen: **13**
  - **Effizienz: 0.6%!**



Daten von Balyo et al. (2015)

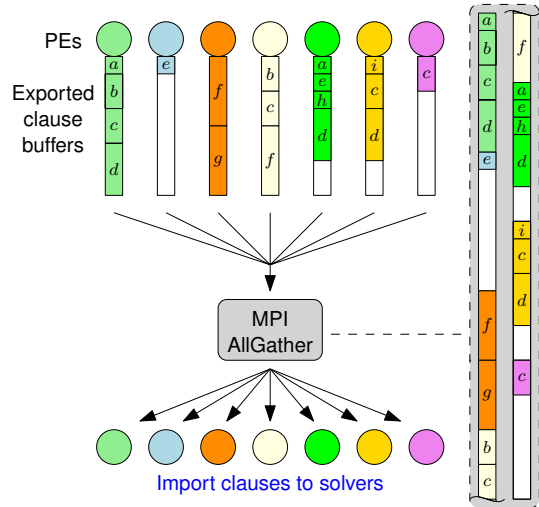
# Solver Architecture à la HordeSat



# Clause Exchange in HordeSat

## Periodic collective operation **AllGather**

- **Locally best clauses** are shared with everyone
- **Duplicate** clauses
- “**Holes**” in buffer carrying no information
- Buffer grows **proportionally** with # proc.  
⇒ **Bottleneck** w.r.t communication *and* local work



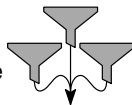
# MALLOBSAT: Overview

## Compact clause sharing



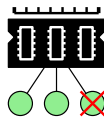
Hierarchical merging  
+ duplicate detection  
Sublinear buffer scaling  
Handling LBD values

## Distributed clause filtering



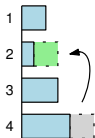
Exact filtering  
of clauses  
shared before  
/ from self

## Memory Awareness



Reduction of  
solver threads  
Negotiated  
memory panic

## Adaptive buffering



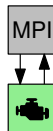
Keep best clauses  
at expense of  
worse clauses  
For export + import

## Diversification

Glucose, Lingeling,  
CaDiCaL, Kissat  
Input permutation  
Clause sharing!



## Controlling



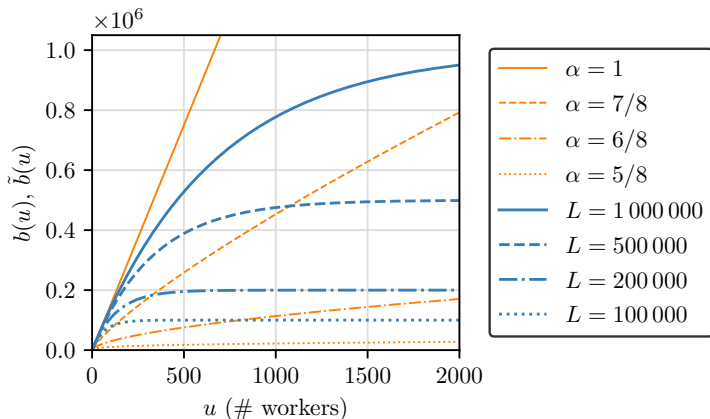
Subprocess for solvers  
Seamless preemption  
and termination  
Fault tolerance

# MALLOBSAT: Buffer-Limitierungen

**Max. Buffergröße** nach  $u$   
 aggregierten Buffern der Größe  $\leq \beta$ :

$$b_{\alpha}(u) = u \cdot \alpha^{\log_2(u)} \cdot \beta$$

$$\tilde{b}_L(u) = L - (L - \beta) \cdot e^{\frac{\beta}{\beta-L}(u-1)}$$



$$\beta = 1500$$

# MALLOBSAT: Handling LBD Values

- Clause quality metric, central for **whether to keep a clause**
- Some solvers **keep clauses with min. LBD indefinitely**  
— but expect **a single solver's clause volume!**

# MALLOBSAT: Handling LBD Values

- Clause quality metric, central for **whether to keep a clause**
- Some solvers **keep clauses with min. LBD indefinitely**  
— but expect **a single solver's clause volume!**
- Use **original LBD values** of imported clauses? [HordeSat]  
⇒ **Growing overhead** (time, space) from low-LBD clauses
- **Reset** LBD values to **maximum** at import? [TopoSAT2]  
⇒ Many clauses may be **discarded very quickly**

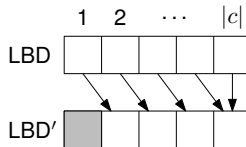


# MALLOBSAT: Handling LBD Values

- Clause quality metric, central for **whether to keep a clause**
- Some solvers **keep clauses with min. LBD indefinitely**  
— but expect **a single solver's clause volume!**
- Use **original LBD values** of imported clauses? [HordeSat]  
⇒ **Growing overhead** (time, space) from low-LBD clauses
- **Reset LBD values to maximum** at import? [TopoSAT2]  
⇒ Many clauses may be **discarded very quickly**

Our current approach: **Increment each LBD before import**

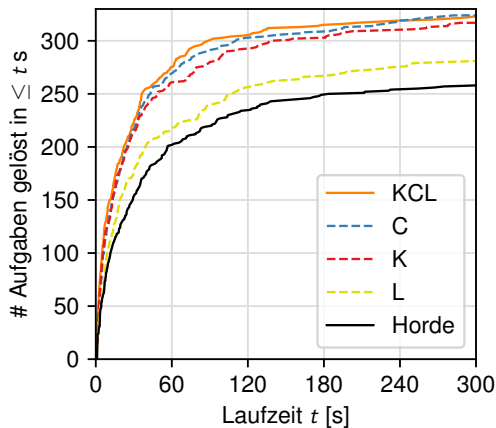
- Maintains **LBD-based prioritization** of clauses
- Solver keeps **full control** over its **LBD-2-clauses**
- “Regional clauses are the best!”



	Median RAM	PAR-2
Orig. LBD	108.8 GiB	75.7
Reset LBD	95.6 GiB	74.3
LBD++	97.3 GiB	72.9

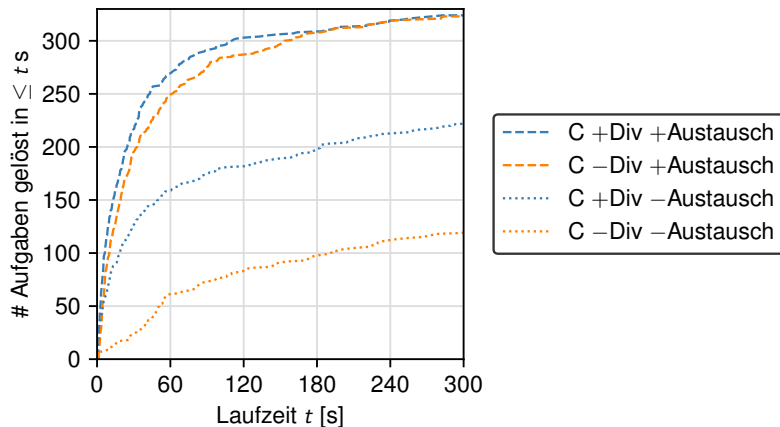
768 cores × 349 instances × 300 s

# MALLOBSAT: Solver-Auswahl und Diversifikation



768 Kerne (16 Maschinen) von SuperMUC-NG • 349 “lösbare” Probleme aus Int. SAT Competition 2022

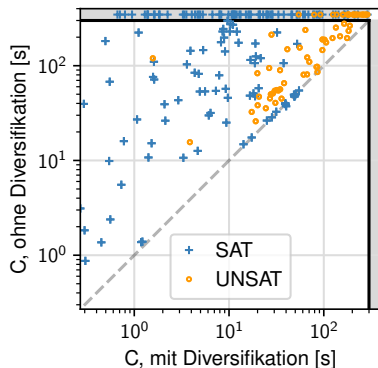
# MALLOBSAT: Diversifikation vs. Klauselaustausch



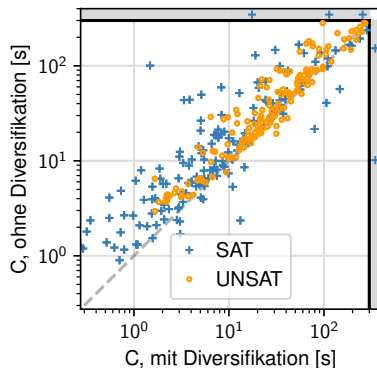
768 Kerne (16 Maschinen) von SuperMUC-NG • 349 “lösbare” Probleme aus Int. SAT Competition 2022

# MALLOBSAT: (k)ein Portfolio-Solver? (1/2)

Einfluss von Diversifikation  
ohne Klauselaustausch



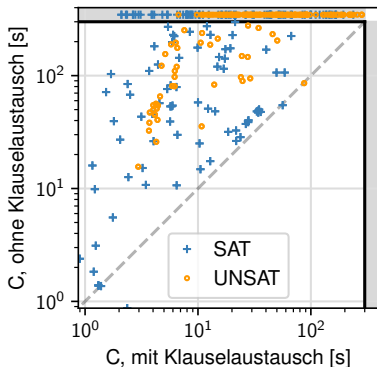
Einfluss von Diversifikation  
mit Klauselaustausch



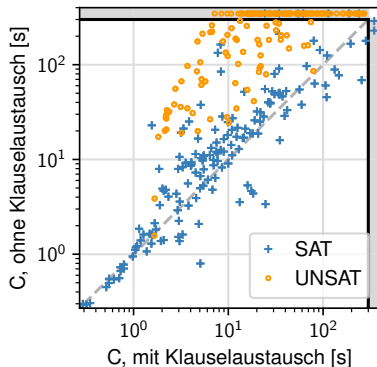
768 Kerne (16 Maschinen) von SuperMUC-NG • 349 "lösbare" Probleme aus Int. SAT Competition 2022

# MALLOBSAT: (k)ein Portfolio-Solver? (2/2)

Einfluss von Klauselaustausch  
ohne Diversifikation

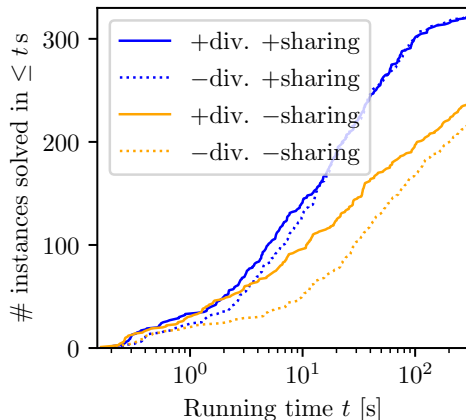


Einfluss von Klauselaustausch  
mit Diversifikation



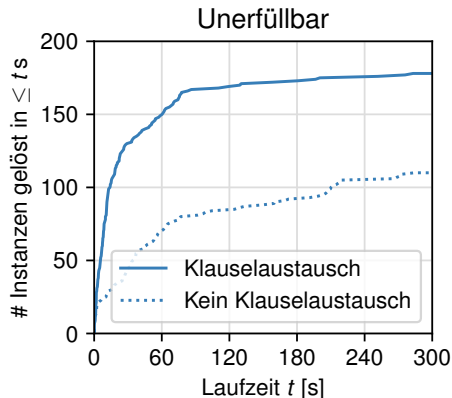
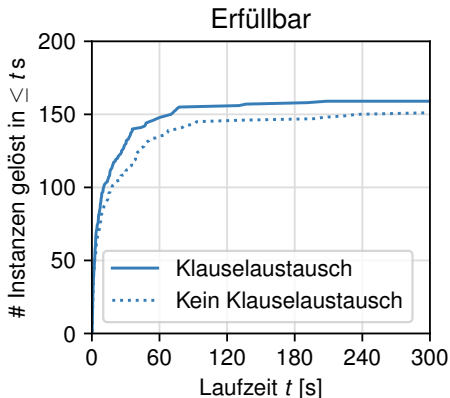
768 Kerne (16 Maschinen) von SuperMUC-NG • 349 "lösbare" Probleme aus Int. SAT Competition 2022

# MALLOBSAT: Diversification vs. Sharing



768 cores (16 machines) of SuperMUC-NG • 349 “solvable” problems from Int. SAT Competition 2022

# MALLOBSAT: Einfluss von Klauselaustausch

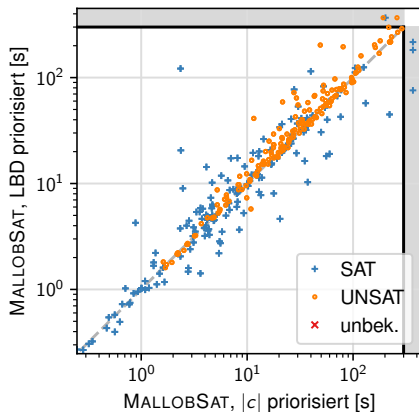


3072 Kerne (128 Maschinen) von SuperMUC-NG • 400 Probleme aus Int. SAT Competition 2021

# MALLOBSAT: Einfluss von LBD-Werten

Klausel-Qualität **per LBD primär?**

- Leicht schlechtere Performance
- Auch für  $LBD \leq 8, |c| \leq 20$  priorisiert



768 Kerne (16 Maschinen) von SuperMUC-NG • 349 “lösbare” Probleme aus Int. SAT Competition 2022



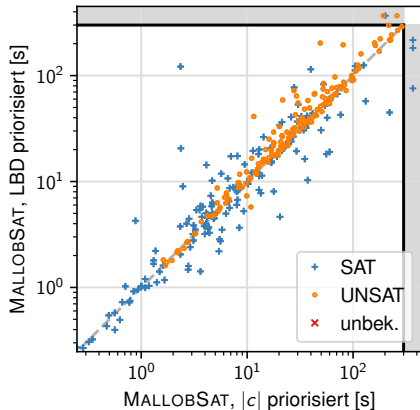
# MALLOBSAT: Einfluss von LBD-Werten

Klausel-Qualität *per LBD primär?*

- Leicht schlechtere Performance
- Auch für  $LBD \leq 8$ ,  $|c| \leq 20$  priorisiert

*Auswürfeln oder Reset aller LBDs* vor Import

- Leicht schlechtere Performance
- Führt zu veränderter LBD-Verteilung in Solvern!



768 Kerne (16 Maschinen) von SuperMUC-NG • 349 “lösbare” Probleme aus Int. SAT Competition 2022

# MALLOBSAT: Einfluss von LBD-Werten

Klausel-Qualität **per LBD primär?**

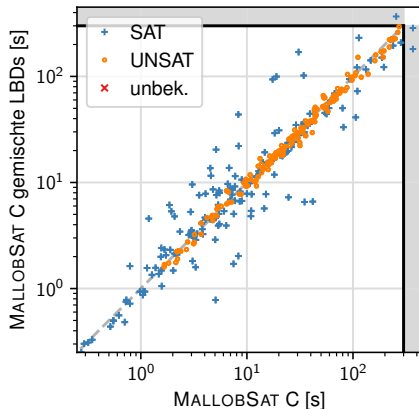
- Leicht schlechtere Performance
- Auch für  $LBD \leq 8$ ,  $|c| \leq 20$  priorisiert

**Auswürfeln oder Reset aller LBDs** vor Import

- Leicht schlechtere Performance
- Führt zu veränderter LBD-Verteilung in Solvern!

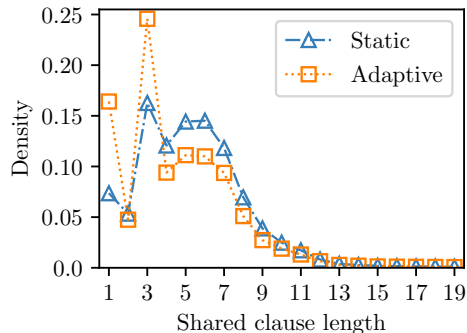
**Durchmischung von LBDs** geteilter Klauseln (rechts)

- LBDs innerhalb jeder Klausellänge permutieren
- **Gleiche Performance!**



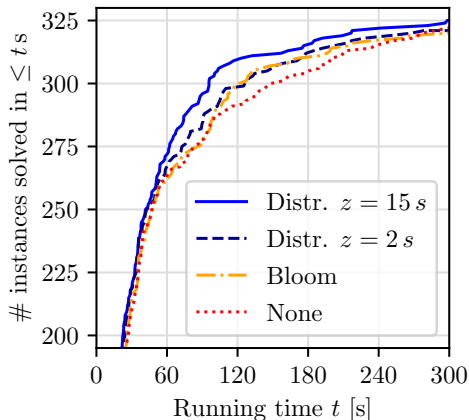
768 Kerne (16 Maschinen) von SuperMUC-NG • 349 "lösbare" Probleme aus Int. SAT Competition 2022

# MALLOBSAT: Clause Stores



768 cores (16 machines) of SuperMUC-NG • 349 “solvable” problems from Int. SAT Competition 2022

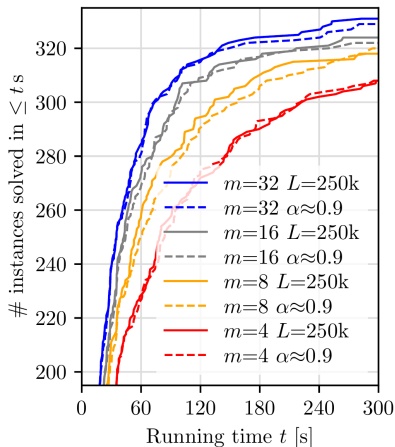
# MALLOBSAT: Clause Filtering



- $\geq 2\times$  acceleration on some instances:  
Unsatisfiable model checking via CBMC
- Instances have multiple connected components
- Only 15-35% of clauses admitted by filter  
(total average 47%)
- Solvers might explore components in differing order  
 $\Rightarrow$  High overlaps but temporally shifted

768 cores (16 machines) of SuperMUC-NG • 349 “solvable” problems from Int. SAT Competition 2022

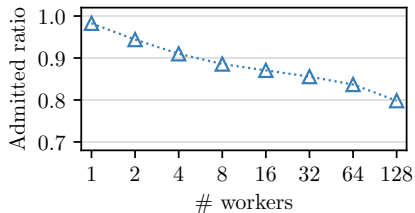
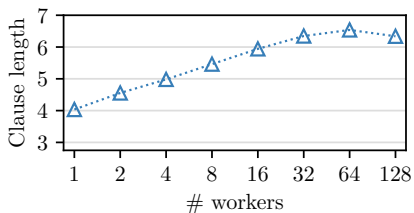
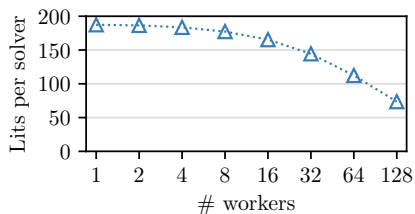
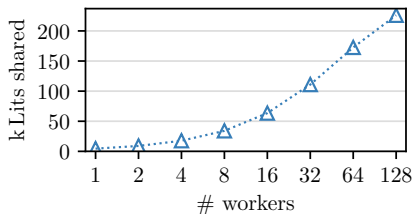
# MALLOBSAT: Sharing Buffer Scaling



$m$	Variant	Lits/s	#	PAR	CSAR
32	$L = 250k$	345 274	331	127.4	27.5
32	$\alpha \approx 0.9$	345 274	329	130.2	28.5
16	$L = 250k$	221 835	324	139.2	29.6
16	$\alpha \approx 0.9$	190 728	322	141.7	30.5
8	$L = 250k$	127 031	318	151.7	36.0
8	$\alpha \approx 0.9$	105 132	320	153.2	39.3
4	$L = 250k$	68 126	308	173.6	45.6
4	$\alpha \approx 0.9$	57 718	308	173.7	45.3

768 cores (16 machines) of SuperMUC-NG • 349 “solvable” problems from Int. SAT Competition 2022

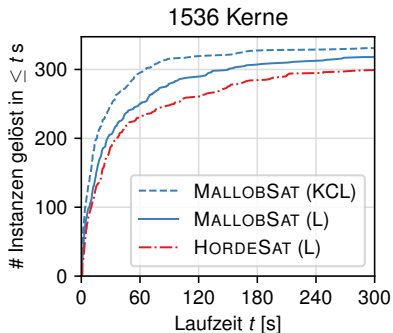
# MALLOBSAT: Clause Sharing Insights



24 cores per worker

400 problems from  
Int. SAT Competition 2021

# MALLOBSAT: Performance und Skalierung



Kerne	# gelöst		Speedup	
	H	M	H	M
24	187	257	3.3	7.2
48	217	279	3.9	9.4
96	244	305	6.0	12.6
192	270	312	8.1	17.7
384	287	321	11.2	22.6
768	297	328	13.8	31.8
1536	299	331	15.9	37.4
3072	–	337	–	43.7

400 Probleme aus SAT Comp. 2021

Seq. Baseline KISSAT\_MAB-HYWALK

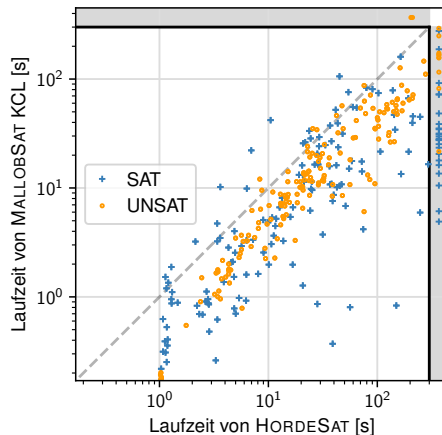
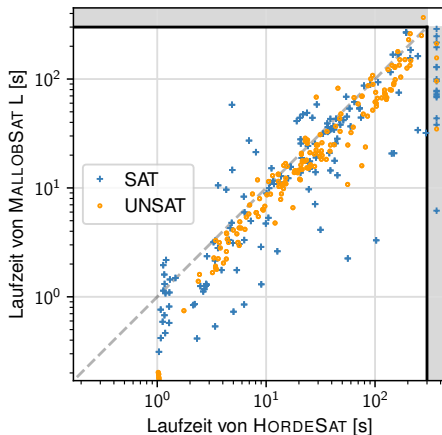
Seq. Zeitlimit 32 h (gelöst: 331)

Par. Zeitlimit 300 s

Geom. Mittel der Speedups auf beiderseits gelösten Instanzen

MALLOBSAT auf 3072 Kernen bei Problemen mit seq. Laufzeit  $\geq 1$  h: **Speedup 419** (Effizienz  $\approx 14\%$ )

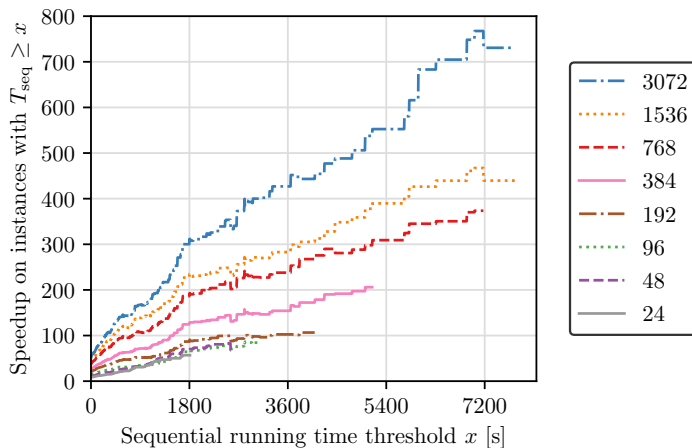
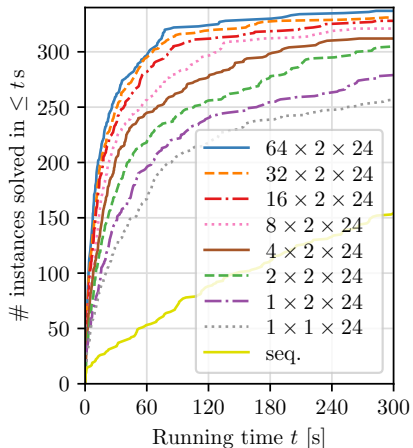
# Direktvergleich: MALLOBSAT vs. HORDESAT



1536 Kerne (64 Maschinen) von SuperMUC-NG • 400 Probleme aus Int. SAT Competition 2021

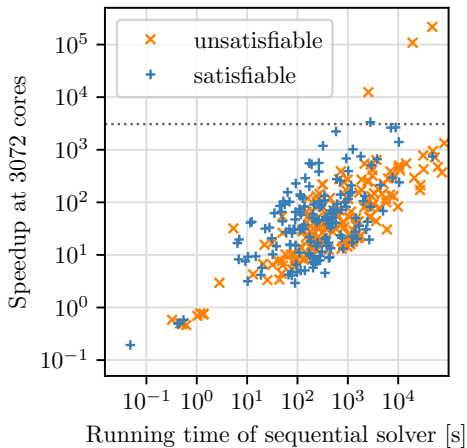
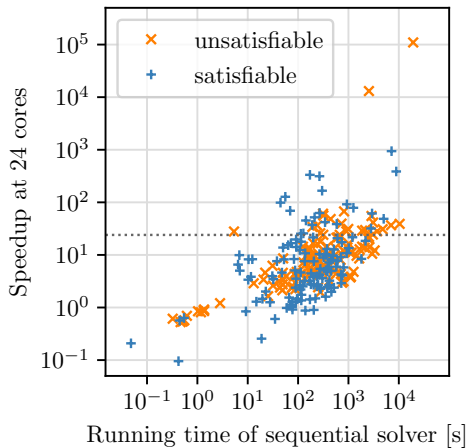


# MALLOBSAT: Strong + Weak Scaling



400 problems from Int. SAT Competition 2021

# MALLOBSAT: Per-Instance Speedups



400 problems from Int. SAT Competition 2021

# MALLOBSAT: Speedups je Familie

Ergebnis und Instanzfamilie		1. Autor	Speedups ▲
SAT	Hypertree Decomposition	Schidler	3, 5, 5, 5, 5, 7, 8, 9, 12, 13
SAT	Hamiltonkreis	Heule	4, 4, 7, 11, 17, 20, 21, 22, 24, 31, 33, 36, 42
SAT	Tree Decomposition	Ehlers	5, 7, 87
UNSAT	Zellularautomat	Chowdhury	5, 8, 8, 9, 9, 10, 22, 22, 66
			⋮
UNSAT	Relativiertes Pidgeon-Hole	Elffers	277, 542, 638
UNSAT	Bioinformatik	Bonet	292, 717
UNSAT	Balanciert zufällig	Spence	321, 388
SAT	Summe dreier Kubikzahlen	Riveros	384, 509, 1018, 3345
SAT	Schaltkreis-Multiplikation	Shunyang	17, 31, 32, 62, 105, 119, 213, 254, 393, 741, 746, 1401, 2650
UNSAT	Perfekte Matchings	Reeves	119, 413, 12 439, 108 593, 217 099

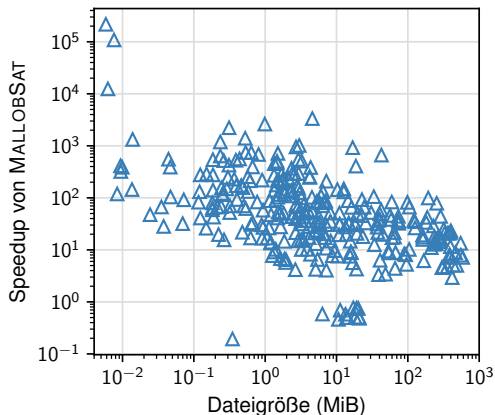
3072 Kerne (128 Maschinen) von SuperMUC-NG • 400 Probleme aus Int. SAT Competition 2021  
 Nur Instanzen mit seq. Laufzeit  $\geq 60$  s • Nur Familien mit  $\geq 2$  Instanzen

# MALLOBSAT: Konsistente Speedups

Instanz	Seq. (s)	Speedup auf $x$ Kernen							
		24	48	92	192	384	768	1536	3072
ctl_4201_555_unsat.cnf	1873.9	14.4	20.1	32.8	49.0	75.3	105.7	138.7	171.2
edit_distance031_283.cnf	2465.9	20.3	36.3	59.8	85.9	122.7	184.2	230.8	247.1
edit_distance031_284.cnf	2561.6	22.0	34.3	59.4	87.7	130.4	185.7	230.3	256.8
mp1-bsat201-707.cnf	3087.9	41.4	67.7	125.3	165.5	234.1	260.7	287.5	321.3
mp1-bsat210-739.cnf	7011.8	36.6	60.3	109.1	150.8	253.0	294.4	324.3	387.5
mp1-klieber2017s-2000-022-eq.cnf	908.0	28.1	49.0	65.8	81.6	92.7	110.6	149.7	172.7
randomG-B-Mix-n16-d05.cnf	3053.2	24.4	41.1	63.5	106.5	163.4	238.5	332.6	413.3
rphp4_065_shuffled.cnf	671.5	24.1	42.3	66.0	94.0	129.0	179.4	231.5	276.8
rphp4_080_shuffled.cnf	2145.0	29.7	53.1	85.3	161.1	229.6	295.1	416.3	542.1
rphp4_090_shuffled.cnf	3709.9	28.2	51.3	91.7	138.9	261.7	379.4	503.2	638.1
satch2ways14u.cnf	1868.9	29.0	45.9	60.8	87.7	120.7	183.0	218.1	270.9
satch2ways15.cnf	10383.0	39.0	59.6	78.5	106.8	180.0	265.9	369.4	506.1
sp4-33-one-stri-tree-noid.cnf	727.5	39.4	55.6	95.6	133.4	198.0	266.6	320.0	390.0

Probleme aus Int. SAT Competition 2021 mit  $> 5\%$  Beschleunigung pro Schritt,  $> 5\%$  Effizienz auf 3072 Kernen

# MALLOBSAT: Speedups vs. Dateigröße

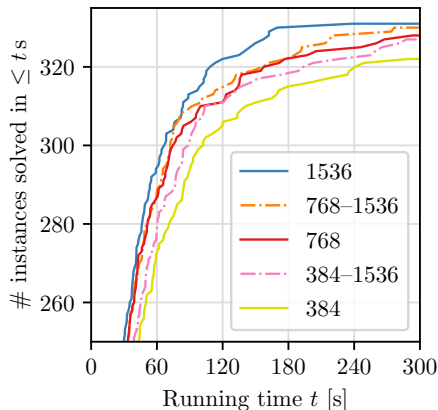


3072 Kerne (128 Maschinen) von SuperMUC-NG • 400 Probleme aus Int. SAT Competition 2021

# MALLOBSAT: Verformbarkeit

## Verformbarkeits-Test

- 1536 Kerne insgesamt
- **Benchmark-Stream**: Übliche sequentielle Verarbeitung von SAT-Benchmarks (2021)
- **Stör-Stream**: ||: 10 s “Stör-Aufgabe”, 10 s Pause :||



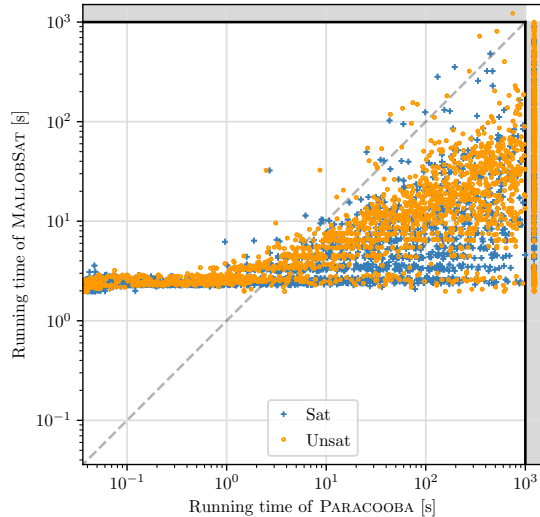
# MALLOBSAT: Wettbewerbe

Exklusiv  
gelöste  
Instanzen

- 2 — 2020: 1. Platz Cloud Track
- 42 { 2021: 2. Platz Parallel Track  
3. Platz SAT, 2. Platz UNSAT  
2021: 1. Platz Cloud Track
- 16 { 2022: 3. Platz Parallel Track  
2. Platz SAT, 1. Platz UNSAT  
2022: 1. Platz Anniversary Parallel Track  
1. Platz UNSAT
- 266 { 2022: 1. Platz Cloud Track  
2022: 1. Platz Anniversary Cloud Track  
2022: 1. Platz FLoC Olympic Games:  
Cloud, Anniv. Cloud, Anniv. Parallel
- 4 { 2023: 1. Platz Cloud Track  
2023: 2. Platz Parallel Track  
2. Platz SAT

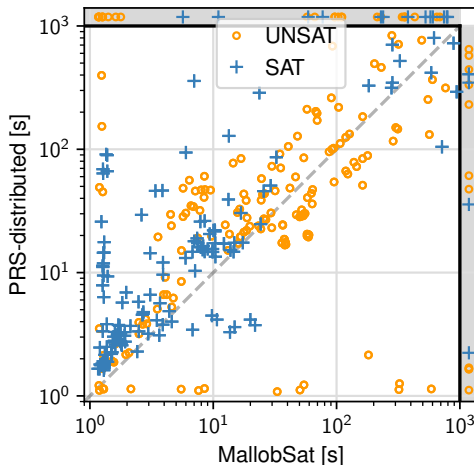


# MALLOBSAT vs. PARACOOBA (ISC'22 Anniversary)





# MALLOBSAT vs. PRS-DISTRIBUTED (ISC'23)



# Which Proof Format?

## DRAT proof format

```
add  $\overline{x_3}$   
add  $x_1 x_2$   
add  $\overline{x_1}$   
delete  $\overline{x_3}$   
add  $x_3 \overline{x_4}$   
add  $x_1 x_3$   
add  $\square$ 
```

# Which Proof Format?

## DRAT proof format

```
add  $\bar{x}_3$   
add  $x_1 x_2$   
add  $\bar{x}_1$   
delete  $\bar{x}_3$   
add  $x_3 \bar{x}_4$   
add  $x_1 x_3$   
add  $\square$ 
```

- + compact format
- + prevalent in solvers
- costly checking

# Which Proof Format?

## DRAT proof format

add  $\bar{x}_3$   
 add  $x_1 x_2$   
 add  $\bar{x}_1$   
 delete  $\bar{x}_3$   
 add  $x_3 \bar{x}_4$   
 add  $x_1 x_3$   
 add  $\square$

- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$   
 add  $c_{10} := x_1 x_2$  via  $c_3, c_2$   
 add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$   
 delete  $c_9$   
 add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$   
 add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$   
 add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$

# Which Proof Format?

## DRAT proof format

add  $\bar{x}_3$   
 add  $x_1 x_2$   
 add  $\bar{x}_1$   
 delete  $\bar{x}_3$   
 add  $x_3 \bar{x}_4$   
 add  $x_1 x_3$   
 add  $\square$

- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$   
 add  $c_{10} := x_1 x_2$  via  $c_3, c_2$   
 add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$   
 delete  $c_9$   
 add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$   
 add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$   
 add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

# Which Proof Format?

## DRAT proof format

```

add  $\bar{x}_3$ 
add  $x_1 x_2$ 
add  $\bar{x}_1$ 
delete  $\bar{x}_3$ 
add  $x_3 \bar{x}_4$ 
add  $x_1 x_3$ 
add  $\square$ 

```

- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

```

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$ 
add  $c_{10} := x_1 x_2$  via  $c_3, c_2$ 
add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$ 
delete  $c_9$ 
add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$ 
add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$ 
add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$ 

```

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

Unique LRAT IDs across solvers?

# Which Proof Format?

## DRAT proof format

```

add  $\bar{x}_3$ 
add  $x_1 x_2$ 
add  $\bar{x}_1$ 
delete  $\bar{x}_3$ 
add  $x_3 \bar{x}_4$ 
add  $x_1 x_3$ 
add  $\square$ 

```

- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

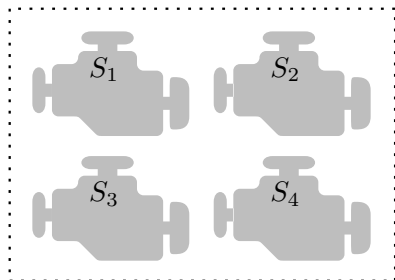
```

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$ 
add  $c_{10} := x_1 x_2$  via  $c_3, c_2$ 
add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$ 
delete  $c_9$ 
add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$ 
add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$ 
add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$ 

```

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

Unique LRAT IDs across solvers?



10 original clauses

# Which Proof Format?

## DRAT proof format

add  $\bar{x}_3$   
 add  $x_1 x_2$   
 add  $\bar{x}_1$   
 delete  $\bar{x}_3$   
 add  $x_3 \bar{x}_4$   
 add  $x_1 x_3$   
 add  $\square$

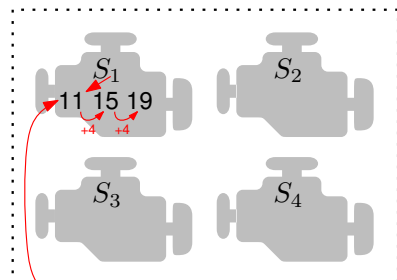
- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$   
 add  $c_{10} := x_1 x_2$  via  $c_3, c_2$   
 add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$   
 delete  $c_9$   
 add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$   
 add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$   
 add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

## Unique LRAT IDs across solvers?



10 original clauses



# Which Proof Format?

## DRAT proof format

add  $\bar{x}_3$   
 add  $x_1 x_2$   
 add  $\bar{x}_1$   
 delete  $\bar{x}_3$   
 add  $x_3 \bar{x}_4$   
 add  $x_1 x_3$   
 add  $\square$

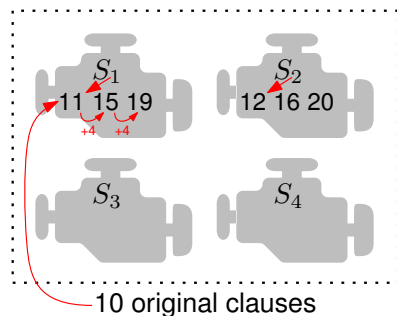
- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$   
 add  $c_{10} := x_1 x_2$  via  $c_3, c_2$   
 add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$   
 delete  $c_9$   
 add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$   
 add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$   
 add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

Unique LRAT IDs across solvers?



# Which Proof Format?

## DRAT proof format

add  $\bar{x}_3$   
 add  $x_1 x_2$   
 add  $\bar{x}_1$   
 delete  $\bar{x}_3$   
 add  $x_3 \bar{x}_4$   
 add  $x_1 x_3$   
 add  $\square$

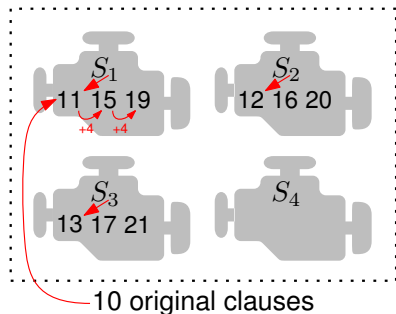
- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$   
 add  $c_{10} := x_1 x_2$  via  $c_3, c_2$   
 add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$   
 delete  $c_9$   
 add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$   
 add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$   
 add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

## Unique LRAT IDs across solvers?



# Which Proof Format?

## DRAT proof format

add  $\bar{x}_3$   
 add  $x_1 x_2$   
 add  $\bar{x}_1$   
 delete  $\bar{x}_3$   
 add  $x_3 \bar{x}_4$   
 add  $x_1 x_3$   
 add  $\square$

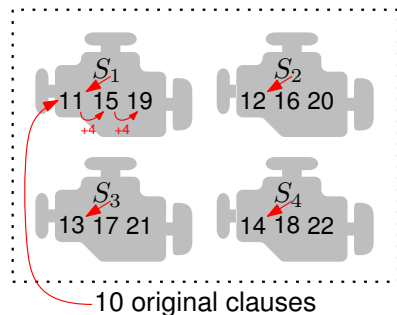
- + compact format
- + prevalent in solvers
- costly checking

## LRAT proof format

add  $c_9 := \bar{x}_3$  via  $c_5, c_4$   
 add  $c_{10} := x_1 x_2$  via  $c_3, c_2$   
 add  $c_{11} := \bar{x}_1$  via  $c_6, c_9$   
 delete  $c_9$   
 add  $c_{12} := x_3 \bar{x}_4$  via  $c_7, c_{11}$   
 add  $c_{13} := x_1 x_3$  via  $c_8, c_{12}$   
 add  $c_{14} := \square$  via  $c_{11}, c_{10}, c_1$

- + more efficient checking
- + unique IDs for clauses
- + explicit dependencies!

Unique LRAT IDs across solvers?



# Distributed Approach

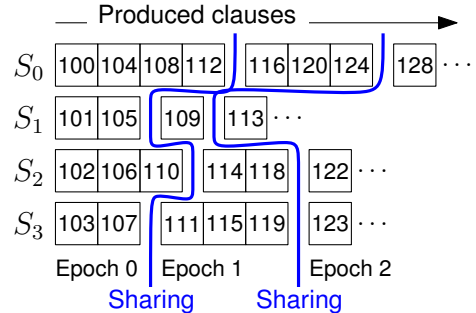
## Parallel processing + distributed memory?

- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**

# Distributed Approach

## Parallel processing + distributed memory?

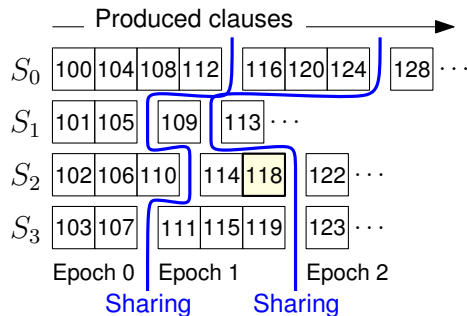
- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**



# Distributed Approach

## Parallel processing + distributed memory?

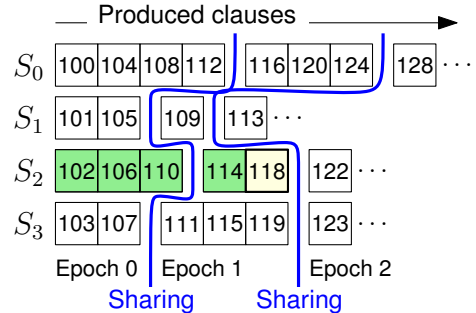
- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**



# Distributed Approach

## Parallel processing + distributed memory?

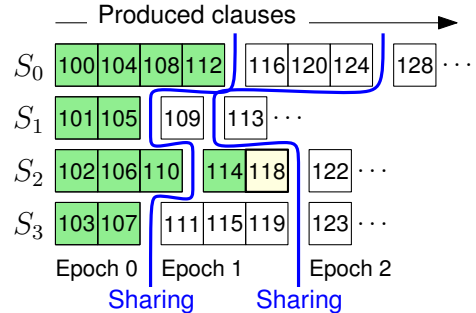
- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**



# Distributed Approach

## Parallel processing + distributed memory?

- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**

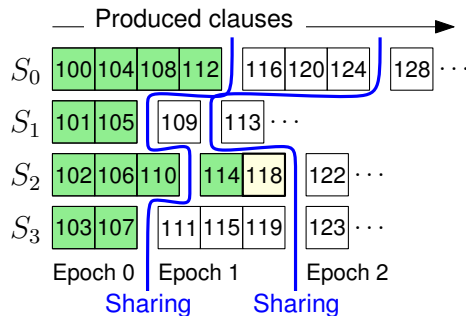




# Distributed Approach

## Parallel processing + distributed memory?

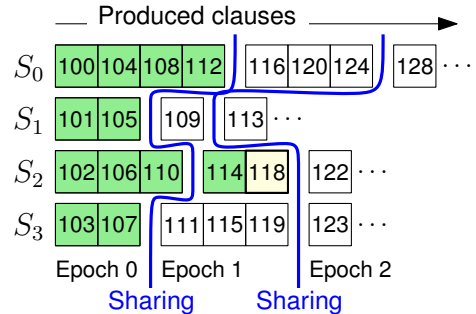
- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**
- Unroll needed dependencies **epoch by epoch**  
in **reverse chronological order**



# Distributed Approach

## Parallel processing + distributed memory?

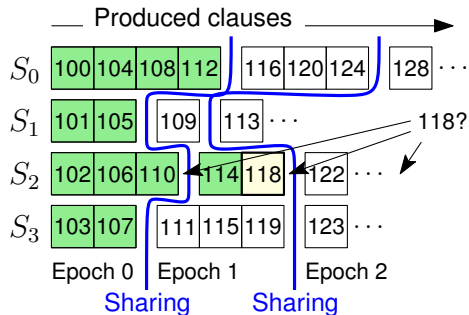
- Prune all partial proofs **in parallel**, **then** combine
  - read each partial proof **only once!**
- Unroll needed dependencies **epoch by epoch**
  - in **reverse chronological order**
- **Redistribute** each required clause to its producer
  - redistribute each ID **only once!**
  - just before processing its **originating epoch!**



# Distributed Approach

## Parallel processing + distributed memory?

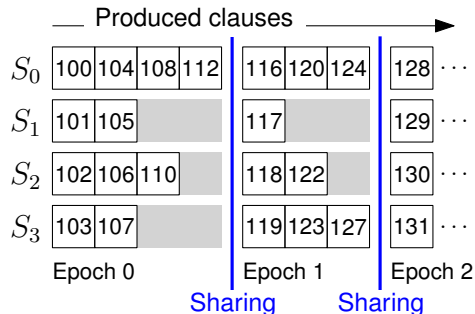
- Prune all partial proofs **in parallel**, **then** combine
  - read each partial proof **only once!**
- Unroll needed dependencies **epoch by epoch** in **reverse chronological order**
- **Redistribute** each required clause to its producer
  - redistribute each ID **only once!**
  - just before processing its **originating epoch!**



# Distributed Approach

## Parallel processing + distributed memory?

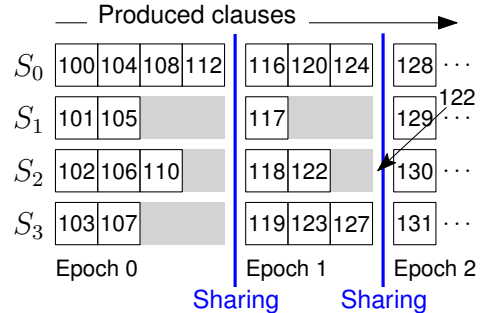
- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**
- Unroll needed dependencies **epoch by epoch**  
in **reverse chronological order**
- **Redistribute** each required clause to its producer  
— redistribute each ID **only once!**  
— just before processing its **originating epoch!**
- **Align clause IDs** to efficiently find epoch of a clause



# Distributed Approach

## Parallel processing + distributed memory?

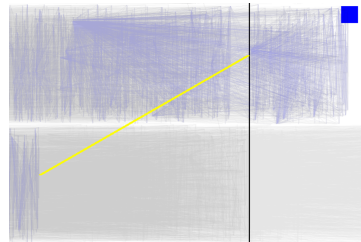
- Prune all partial proofs **in parallel**, **then** combine  
— read each partial proof **only once!**
- Unroll needed dependencies **epoch by epoch**  
in **reverse chronological order**
- **Redistribute** each required clause **to its producer**  
— redistribute each ID **only once!**  
— just before processing its **originating epoch!**
- **Align clause IDs** to efficiently find epoch of a clause



# Rewind: Realization

## Local Processing

- For each  $S_i$ : **Frontier**  $F_i$  of req. clause IDs *produced by*  $S_i$ ;  
**Backlog**  $B_i$  of *remote* req. clause IDs
  - **External-memory priority queues** partitioned by epoch
- Epoch  $e$ : Process proof parts from ep.  $e$
- Clause  $c$  with  $id(c) \in F_i$ : Insert each  $d \in deps(c)$  into  $F_i$  or  $B_i$



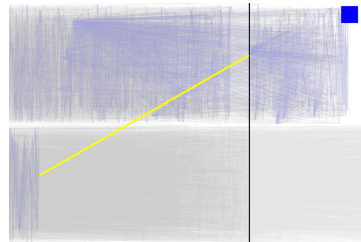
# Rewind: Realization

## Local Processing

- For each  $S_i$ : **Frontier**  $F_i$  of req. clause IDs *produced by*  $S_i$ ;  
**Backlog**  $B_i$  of *remote* req. clause IDs
  - **External-memory priority queues** partitioned by epoch
- Epoch  $e$ : Process proof parts from ep.  $e$
- Clause  $c$  with  $id(c) \in F_i$ : Insert each  $d \in deps(c)$  into  $F_i$  or  $B_i$

## Redistribution of Clause IDs

- After processing epoch  $e$ : Extract IDs from ep.  $e - 1$  from all  $B_i$
- All-reduction like Mallob's clause sharing, **detecting duplicate IDs**
- **Strictly less communication** than during solving



# Kontext: Verteilte Graph-Algorithmen

**Single-source Erreichbarkeit** *aller* Knoten  $t$  von Startknoten  $s$  in **verteiletem gerichtetem Graphen**  $G$   
 $n$  Knoten,  $m$  Kanten, Durchmesser  $D$ ;  $p$  Prozessoren

- Einfügen randomisierter **Durchmesser-reduzierender Shortcuts**  
Ghaffari, Mohsen, and Rajan Udwani. “Brief announcement: Distributed single-source reachability.” ACM Symp. on Principles of Distributed Computing, 2015.
- Stand der Technik (2019):  $\tilde{O}(m)$  **Arbeit**,  $n^{1/2+o(1)}$  **Tiefe** w.h.p. in  $n$   
Jambulapati, Arun et al. “Parallel reachability in almost linear work and square root depth.” FOCS 2019.

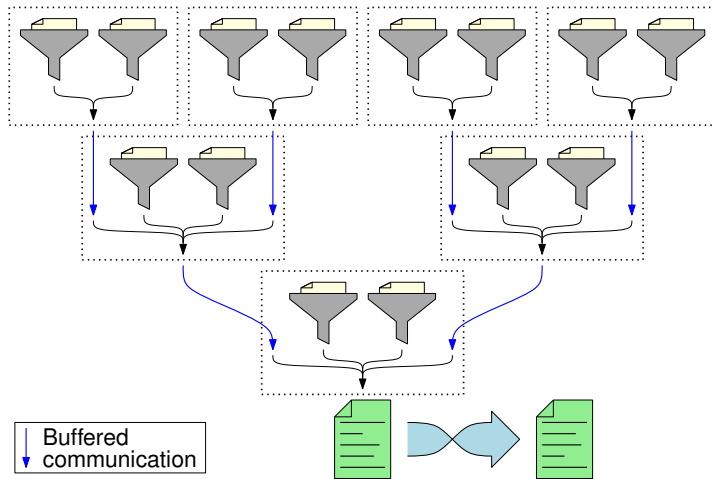
## Unser Ansatz

- Grob  $\mathcal{O}(m \log m + p \cdot \hat{\epsilon} K)$  **Arbeit**,  $\mathcal{O}(D \log m + \hat{\epsilon} \cdot K \log p)$  **Tiefe**  
 $\hat{\epsilon}$ : # Klauselaustausch-Operationen;  $K$ : # Klauseln je Austausch
- **Streaming-Algorithmus**: Greift auf jede Kante ein einziges Mal, in fester Reihenfolge zu
- **Externer Algorithmus**: Benötigt prinzipiell nur  $\mathcal{O}(K + \hat{\epsilon} p + \max. \text{Ausgangsgrad} + \text{ext. PQ})$  Speicher
- Gebündelte Kommunikation:  $\hat{\epsilon} \cdot 2(p - 1)$  **Nachrichten** der Länge  $\mathcal{O}(K)$



# Distributed Combination

- Hierarchically merge pruning output along **tree of processors**
- Root processor
  - 1 adds approximated “delete” lines
  - 2 writes stream into file
  - 3 reverses file



# Experimental Setup (1/2)

## Technology

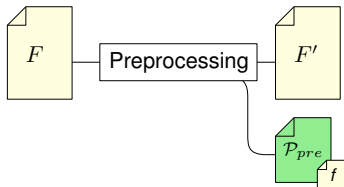
- Base SAT solver: [CaDiCaL](#) [Biere 2018] modified to output LRAT, **restricted portfolio**
- Distributed solver: [Mallob](#) [Schreiber+Sanders 2021] extended by clause IDs + proof production
- Proof checking: `lrat-check` from [drat-trim](#) tools (M. Heule)

# Experimental Setup (1/2)

## Technology

- Base SAT solver: [CaDiCaL](#) [Biere 2018] modified to output LRAT, **restricted portfolio**
- Distributed solver: [Mallob](#) [Schreiber+Sanders 2021] extended by clause IDs + proof production
- Proof checking: `lrat-check` from [drat-trim](#) tools (M. Heule)

## Pipeline

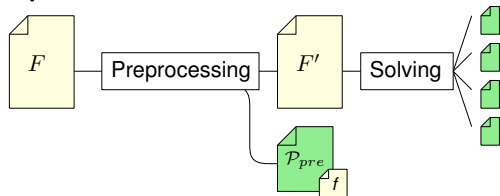


# Experimental Setup (1/2)

## Technology

- Base SAT solver: [CaDiCaL](#) [Biere 2018] modified to output LRAT, **restricted portfolio**
- Distributed solver: [Mallob](#) [Schreiber+Sanders 2021] extended by clause IDs + proof production
- Proof checking: `lrat-check` from [drat-trim](#) tools (M. Heule)

## Pipeline

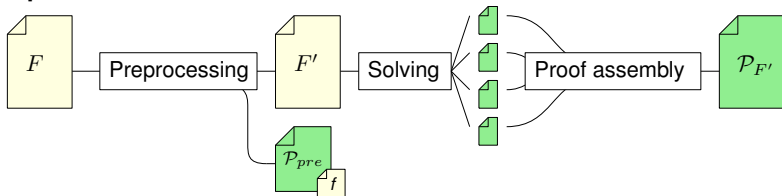


# Experimental Setup (1/2)

## Technology

- Base SAT solver: [CaDiCaL](#) [Biere 2018] modified to output LRAT, **restricted portfolio**
- Distributed solver: [Mallob](#) [Schreiber+Sanders 2021] extended by clause IDs + proof production
- Proof checking: `lrat-check` from [drat-trim](#) tools (M. Heule)

## Pipeline

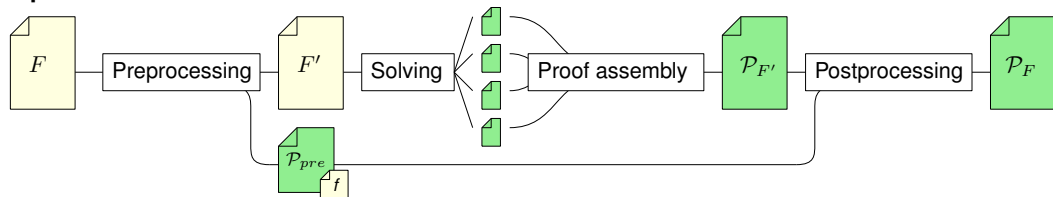


# Experimental Setup (1/2)

## Technology

- Base SAT solver: [CaDiCaL](#) [Biere 2018] modified to output LRAT, **restricted portfolio**
- Distributed solver: [Mallob](#) [Schreiber+Sanders 2021] extended by clause IDs + proof production
- Proof checking: `lrat-check` from [drat-trim](#) tools (M. Heule)

## Pipeline

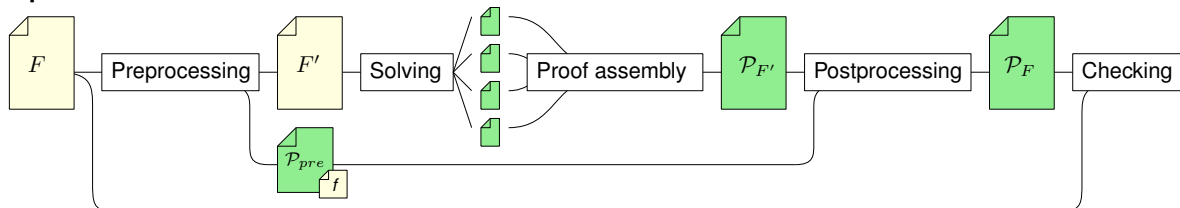


# Experimental Setup (1/2)

## Technology

- Base SAT solver: [CaDiCaL](#) [Biere 2018] modified to output LRAT, **restricted portfolio**
- Distributed solver: [Mallob](#) [Schreiber+Sanders 2021] extended by clause IDs + proof production
- Proof checking: `lrat-check` from [drat-trim](#) tools (M. Heule)

## Pipeline



## Experimental Setup (2/2)

### Comparison to prior work

- Shared-memory clause-sharing portfolios: [Heule, Manthey, Philipp @ POS'14](#)
  - Synchronized, **moderated** logging into shared [DRAT proof](#)
  - Solver not competitive  $\Rightarrow$  Simulate proof output, compare [checking times only](#)
- Sequential SAT solving: [Kissat\\_MAB-HyWalk @ SAT Comp. 2022](#)



## Experimental Setup (2/2)

### Comparison to prior work

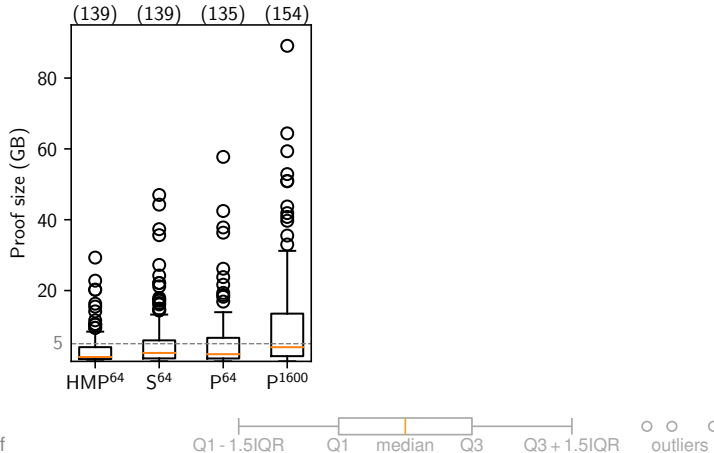
- Shared-memory clause-sharing portfolios: [Heule, Manthey, Philipp @ POS'14](#)
  - Synchronized, **moderated** logging into shared [DRAT proof](#)
  - Solver not competitive  $\Rightarrow$  Simulate proof output, compare [checking times only](#)
- Sequential SAT solving: [Kissat\\_MAB-HyWalk @ SAT Comp. 2022](#)

### Resources

- **1600× setup**: 100× m6i.4xlarge EC2 instances (16 hwthreads, 64 GB RAM)
  - **64× setup**: 1× m6i.16xlarge EC2 instance (64 hwthreads, 256 GB RAM)
  - Sequential setup: One m6i.4xlarge EC2 instance
- }  $\leq 1000$  s solving  
 $\leq 4000$  s proof prod.

# Evaluation: Proof Output

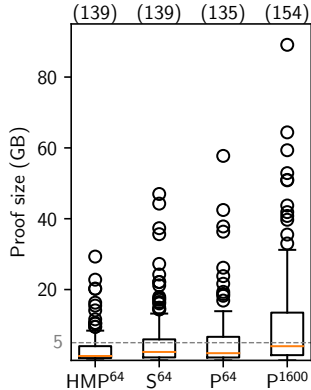
How large are the resulting proofs?



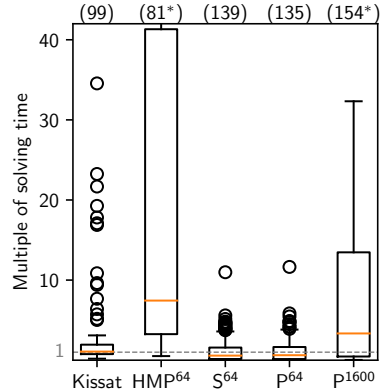
\*Some data cut off

# Evaluation: Proof Output

How large are the resulting proofs?



How fast can we check the proofs?

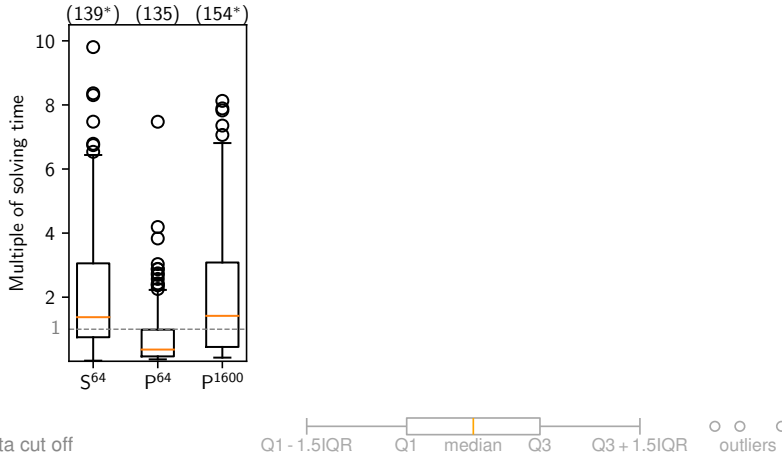


\*Some data cut off



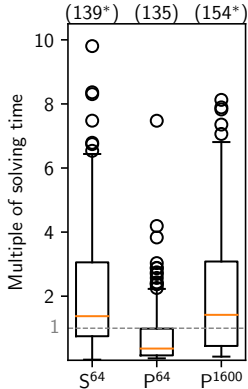
# Evaluation: Overhead

## Proof assembly

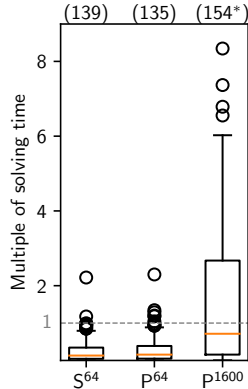


# Evaluation: Overhead

## Proof assembly



## Postprocessing

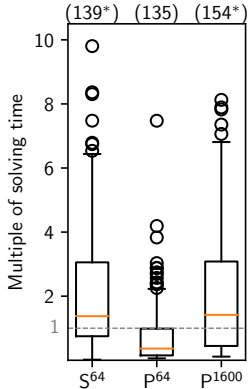


\*Some data cut off

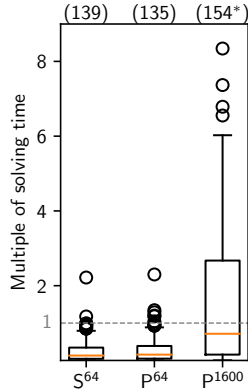


# Evaluation: Overhead

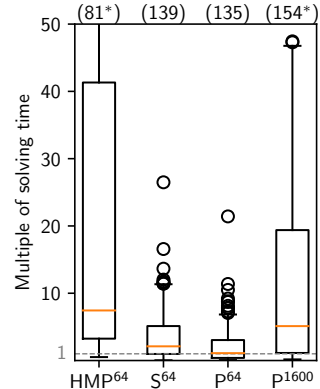
### Proof assembly



### Postprocessing



### Total (HMP: checking only)

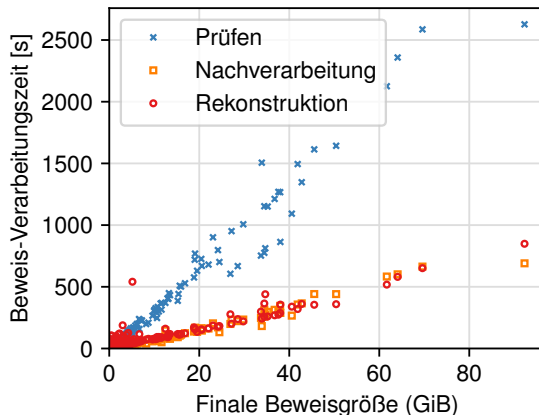


\*Some data cut off



# Beweise: Weitere Einblicke

- **Durchsatz** der Beweis-Rekonstruktion (median-Instanz): **114 MB/s**
- Pruning-Faktor: durchschnittl. **28×**, **korreliert stark mit 1/Solving-Zeit**
  - Je mehr Klauselaustausch-Operationen, desto **verknüpfter** wird der Beweis
- **Größte Beweise** ( $\geq 40$  GB):
  - mp1 Block-Puzzle
  - 2× div EPFL-Miter
  - 2× Sport-Turnierplanung
  - Sudoku-Problemgenerierung
  - Gruppentheorie (Hantzsche-Wendt)
  - Äquivalenz Pancake– vs. Selection-Sort
  - Kryptographie (MD5)



# Total-Order Hierarchical Task Network Planning

## Objective

- Achieve a given set of **tasks** ...

$t_1(A, B)$

$t_2(C)$

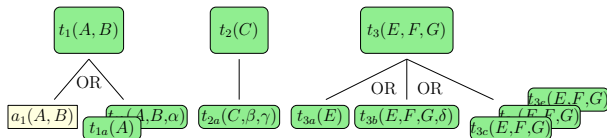
$t_3(E, F, G)$



# Total-Order Hierarchical Task Network Planning

## Objective

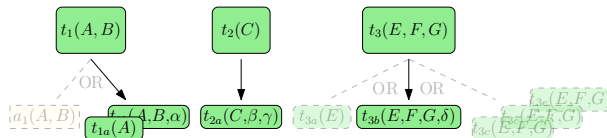
- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...



# Total-Order Hierarchical Task Network Planning

## Objective

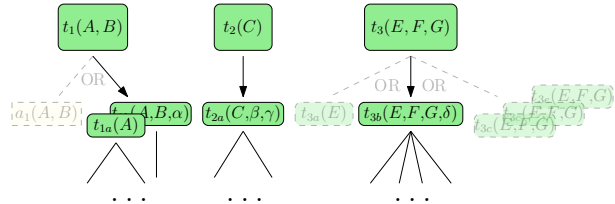
- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...



# Total-Order Hierarchical Task Network Planning

## Objective

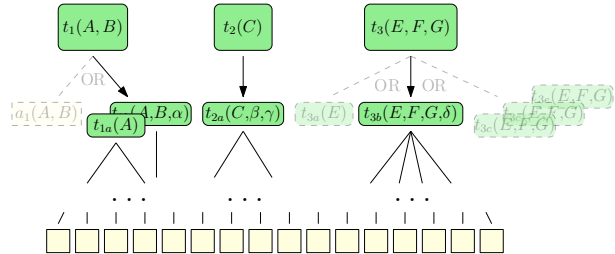
- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...



# Total-Order Hierarchical Task Network Planning

## Objective

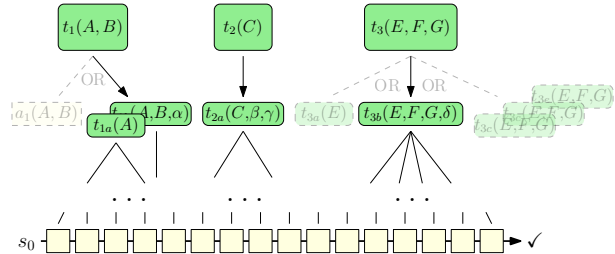
- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...



# Total-Order Hierarchical Task Network Planning

## Objective

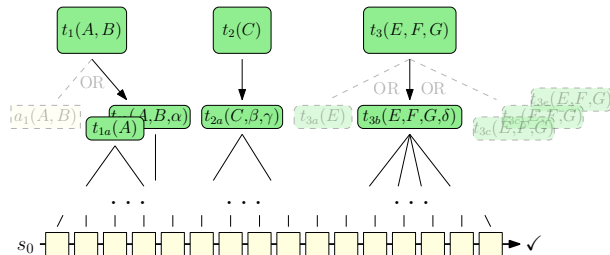
- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...
- which form a **plan**, a sequence of **executable actions** from the given **initial state**



# Total-Order Hierarchical Task Network Planning

## Objective

- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...
- which form a **plan**, a sequence of **executable actions** from the given **initial state**



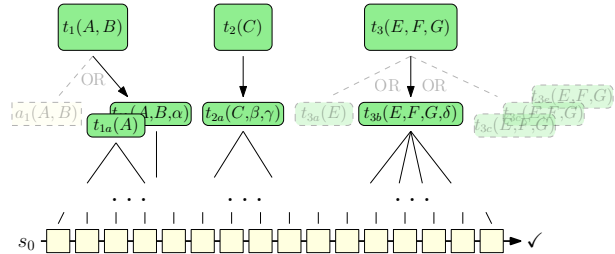
## Structures

- **Fact**: Boolean feature of world state

# Total-Order Hierarchical Task Network Planning

## Objective

- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...
- which form a **plan**, a sequence of **executable actions** from the given **initial state**



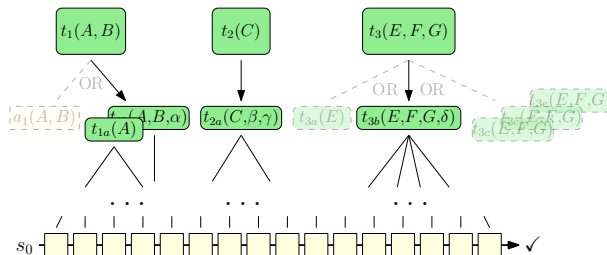
## Structures

- **Fact**: Boolean feature of world state
- **Task**: Footprint of sth. to achieve

# Total-Order Hierarchical Task Network Planning

## Objective

- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...
- which form a **plan**, a sequence of **executable actions** from the given **initial state**



## Structures

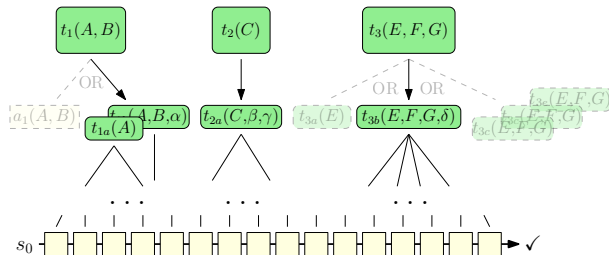
- **Fact**: Boolean feature of world state
- **Task**: Footprint of sth. to achieve
- **Method**: Recipe to achieve certain compound task  $\Rightarrow$  **preconditions**, **subtasks**



# Total-Order Hierarchical Task Network Planning

## Objective

- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...
- which form a **plan**, a sequence of **executable actions** from the given **initial state**



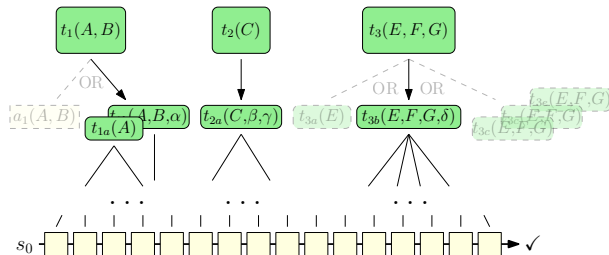
## Structures

- **Fact**: Boolean feature of world state
- **Task**: Footprint of sth. to achieve
- **Method**: Recipe to achieve certain compound task  $\Rightarrow$  **preconditions**, **subtasks**
- **Operator**: Recipe to execute primitive task  $\Rightarrow$  **preconditions**, **effects**

# Total-Order Hierarchical Task Network Planning

## Objective

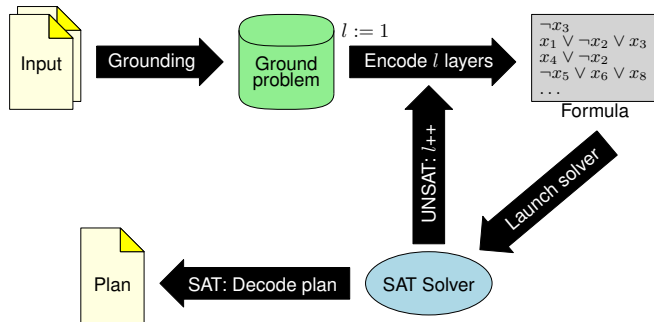
- Achieve a given set of **tasks** ...
- by recursively replacing each task with a specific set of **subtasks** ...
- until only “atomic” **primitive tasks** remain ...
- which form a **plan**, a sequence of **executable actions** from the given **initial state**



## Structures

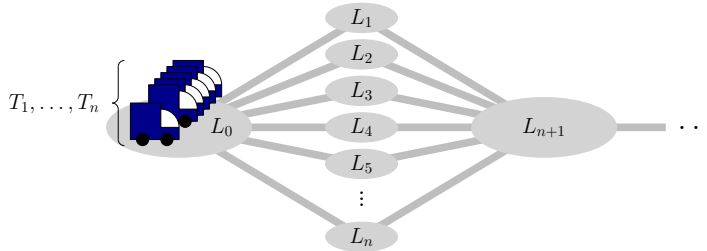
- **Fact**: Boolean feature of world state
- **Task**: Footprint of sth. to achieve
- **Method**: Recipe to achieve certain compound task  $\Rightarrow$  **preconditions**, **subtasks**
- **Operator**: Recipe to execute primitive task  $\Rightarrow$  **preconditions**, **effects**
- **Action (Reduction)**: *ground* operator (method) – no **free variables**

# SAT-based TOHTN Planning



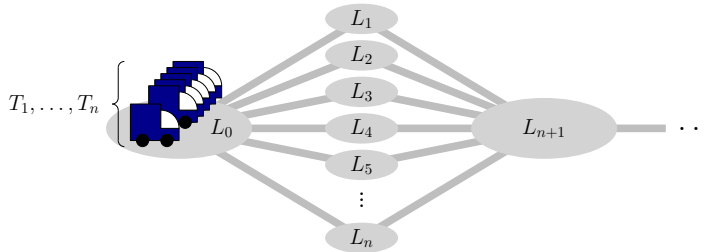
Behnke et al. (AAAI'18) with separate formulae & solver instances,  
Schreiber et al. (ICAPS'19) with [incremental SAT solving](#)

# The Problem with Grounding



$drive(T_1, L_0, L_1)$	$drive(T_1, L_0, L_2)$	$drive(T_1, L_0, L_3)$	$\dots$	$drive(T_n, L_0, L_n)$
$drive(T_2, L_0, L_1)$	$drive(T_2, L_0, L_2)$	$drive(T_2, L_0, L_3)$		
$drive(T_3, L_0, L_1)$	$drive(T_3, L_0, L_2)$			
$drive(T_4, L_0, L_1)$				$\vdots$
$drive(T_5, L_0, L_1)$		$\dots$		
$\vdots$				
$drive(T_n, L_0, L_1)$	$\dots$			$drive(T_n, L_0, L_n)$

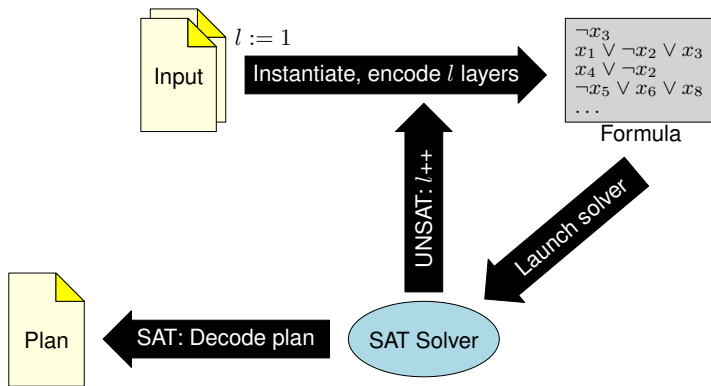
# The Problem with Grounding



$drive(T_1, L_0, L_1)$	$drive(T_1, L_0, L_2)$	$drive(T_1, L_0, L_3)$	$\dots$	$drive(T_n, L_0, L_n)$
$drive(T_2, L_0, L_1)$	$drive(T_2, L_0, L_2)$	$drive(T_2, L_0, L_3)$		
$drive(T_3, L_0, L_1)$	$drive(T_3, L_0, L_2)$			
$drive(T_4, L_0, L_1)$				$\vdots$
$drive(T_5, L_0, L_1)$		$\dots$		
$\vdots$				
$drive(T_n, L_0, L_1)$	$\dots$			$drive(T_n, L_0, L_n)$

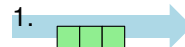
⇒ Combinatorial blowup!

# Our Approach



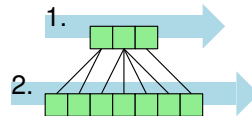
# Instantiation

- Layer 0: Instantiate operations matching initial tasks



# Instantiation

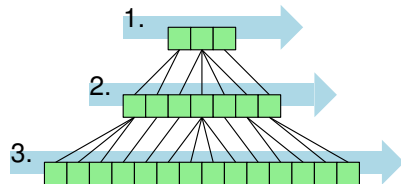
- Layer 0: Instantiate operations matching initial tasks
- Layer  $l$ : Instantiate possible children of op. at layer  $l - 1$





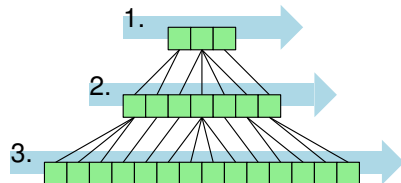
# Instantiation

- Layer 0: Instantiate operations matching initial tasks
- Layer  $l$ : Instantiate possible children of op. at layer  $l - 1$
- **Keep variables**, do not ground into separate operations
- Layer fully instantiated: **Encode, attempt to solve**

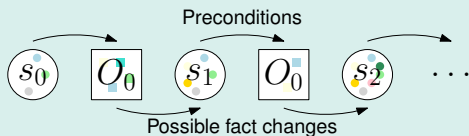


# Instantiation

- Layer 0: Instantiate operations matching initial tasks
- Layer  $l$ : Instantiate possible children of op. at layer  $l - 1$
- **Keep variables**, do not ground into separate operations
- Layer fully instantiated: **Encode, attempt to solve**

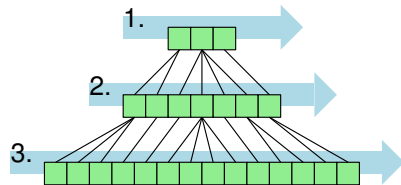


## Reachability Analysis

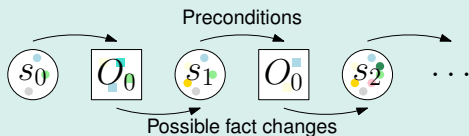


# Instantiation

- Layer 0: Instantiate operations matching initial tasks
- Layer  $l$ : Instantiate possible children of op. at layer  $l - 1$
- **Keep variables**, do not ground into separate operations
- Layer fully instantiated: **Encode, attempt to solve**



## Reachability Analysis



- ⇒ (Over-)approximate fact changes of instantiated operations, add to possible facts
- ⇒ Discard operations with impossible preconditions

# SAT Encoding

---

	<b>Tree-REX (ICAPS'19)</b>	<b>Lilotane (ours)</b>
Operation variables (per position)	$drive(T_1, L_0, L_1), drive(T_1, L_0, L_2),$ $drive(T_2, L_0, L_1), drive(T_2, L_0, L_2), \dots$	$drive(\alpha, \beta, \gamma)$

---

# SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$drive(T_1, L_0, L_1), drive(T_1, L_0, L_2),$ $drive(T_2, L_0, L_1), drive(T_2, L_0, L_2), \dots$	$drive(\alpha, \beta, \gamma)$
Fact variables (per position)	$at(T_1, L_0), \dots, road(L_0, L_1), \dots$	$at(T_1, L_0), \dots, road(L_0, L_1), \dots$ $at(\alpha, \beta), at(\alpha, \gamma), road(\beta, \gamma)$

# SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$drive(T_1, L_0, L_1), drive(T_1, L_0, L_2),$ $drive(T_2, L_0, L_1), drive(T_2, L_0, L_2), \dots$	$drive(\alpha, \beta, \gamma)$
Fact variables (per position)	$at(T_1, L_0), \dots, road(L_0, L_1), \dots$	$at(T_1, L_0), \dots, road(L_0, L_1), \dots$ $at(\alpha, \beta), at(\alpha, \gamma), road(\beta, \gamma)$
Substitution variables — (global)		$[\alpha/T_1], [\alpha/T_2], \dots$ $[\beta/L_0], [\beta/L_1], \dots$

# SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$drive(T_1, L_0, L_1)$ , $drive(T_1, L_0, L_2)$ , $drive(T_2, L_0, L_1)$ , $drive(T_2, L_0, L_2)$ , ...	$drive(\alpha, \beta, \gamma)$
Fact variables (per position)	$at(T_1, L_0)$ , ..., $road(L_0, L_1)$ , ...	$at(T_1, L_0)$ , ..., $road(L_0, L_1)$ , ... $at(\alpha, \beta)$ , $at(\alpha, \gamma)$ , $road(\beta, \gamma)$
Substitution variables (global)	—	$[\alpha/T_1]$ , $[\alpha/T_2]$ , ... $[\beta/L_0]$ , $[\beta/L_1]$ , ...
Selected clause schemes	$drive(T_1, L_0, L_1) \Rightarrow at(T_1, L_0)$	$drive(\alpha, \beta, \gamma) \Rightarrow at(\alpha, \beta)$ $exactly-one([\alpha/T_1], [\alpha/T_2], \dots, [\alpha/T_n])$ $[\alpha/T_1] \wedge [\beta/L_0] \Rightarrow (at(\alpha, \beta) \Leftrightarrow at(T_1, L_0))$

# Our Encoding

## Boolean variables

- $f_x^l$ : “At the  $l$ -th layer, fact  $f$  holds before the  $x$ -th operation”  *$f$  may contain variables (pseudo-constants)*
- $o_x^l$ : “At the  $l$ -th layer, the  $x$ -th operation is  $o$ ”  *$o$  may contain pseudo-constants*
- $prim_x^l$ : “At the  $l$ -th layer, the  $x$ -th operation is primitive”
- $[\alpha/c]$ : “Pseudo-constant  $\alpha$  is substituted with constant  $c$ ”

## Sparse Encoding

- Only encode variables which are not trivially true or trivially false
- **Reachability analysis** from top to bottom, left to right: Filter impossible operations, facts
- Retroactively prune subtrees which turned out to be impossible



# Our Encoding

## Clauses (1/2)

- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0$  ,  $\forall f \notin s_0 : \neg f_0^0$

# Our Encoding

## Clauses (1/2)

- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0$  ,  $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position

# Our Encoding

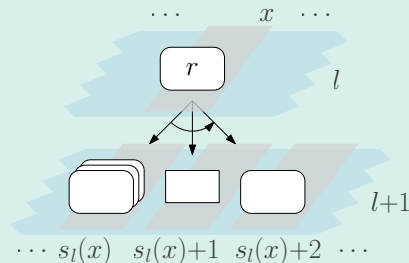
## Clauses (1/2)

- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0$  ,  $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects:  $o_x^l \Rightarrow \bigwedge_{f \in \text{pre}(o)} f_x^l \wedge \bigwedge_{f \in \text{eff}(o)} f_{x+1}^l$

# Our Encoding

## Clauses (1/2)

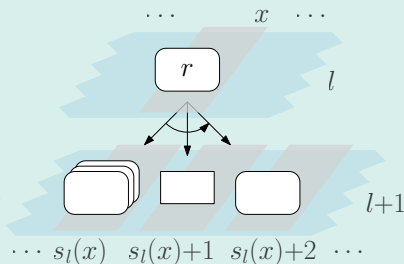
- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0$ ,  $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects:  $o_x^l \Rightarrow \bigwedge_{f \in \text{pre}(o)} f_x^l \wedge \bigwedge_{f \in \text{eff}(o)} f_{x+1}^l$
- Propagation of facts, actions:  $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}$ ,  $o_x^l \Rightarrow o_{s_l(x)}^{l+1}$



# Our Encoding

## Clauses (1/2)

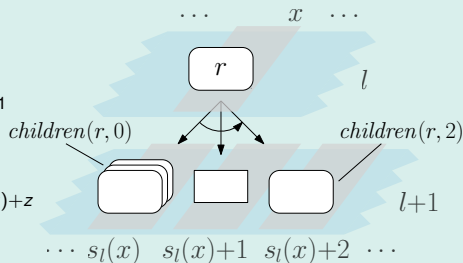
- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0$ ,  $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects:  $o_x^l \Rightarrow \bigwedge_{f \in \text{pre}(o)} f_x^l \wedge \bigwedge_{f \in \text{eff}(o)} f_{x+1}^l$
- Propagation of facts, actions:  $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}$ ,  $o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction:  $\forall z : o_x^l \Rightarrow \bigvee_{o' \in \text{children}(o,z)} (o')_{s_l(x)+z}^{l+1}$



# Our Encoding

## Clauses (1/2)

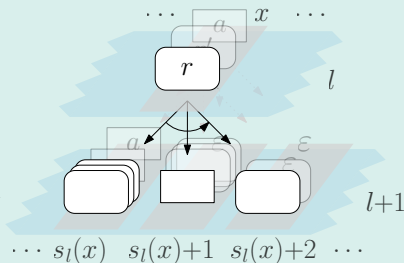
- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0, \forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects:  $o_x^l \Rightarrow \bigwedge_{f \in \text{pre}(o)} f_x^l \wedge \bigwedge_{f \in \text{eff}(o)} f_{x+1}^l$
- Propagation of facts, actions:  $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}, o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction:  $\forall z : o_x^l \Rightarrow \bigvee_{o' \in \text{children}(o,z)} (o')_{s_l(x)+z}^{l+1}$



# Our Encoding

## Clauses (1/2)

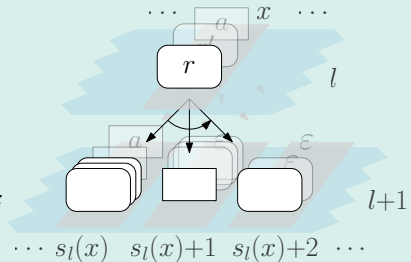
- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0, \forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects:  $o_x^l \Rightarrow \bigwedge_{f \in \text{pre}(o)} f_x^l \wedge \bigwedge_{f \in \text{eff}(o)} f_{x+1}^l$
- Propagation of facts, actions:  $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}, o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction:  $\forall z : o_x^l \Rightarrow \bigvee_{o' \in \text{children}(o,z)} (o')_{s_l(x)+z}^{l+1}$



# Our Encoding

## Clauses (1/2)

- Initial state  $s_0$ :  $\forall f \in s_0 : f_0^0, \forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects:  $o_x^l \Rightarrow \bigwedge_{f \in \text{pre}(o)} f_x^l \wedge \bigwedge_{f \in \text{eff}(o)} f_{x+1}^l$
- Propagation of facts, actions:  $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}, o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction:  $\forall z : o_x^l \Rightarrow \bigvee_{o' \in \text{children}(o,z)} (o')_{s_l(x)+z}^{l+1}$
- **Assume** fully expanded network at deepest layer  $l'$ :  
 $\text{prim}'_0, \text{prim}'_1, \text{prim}'_2, \dots$





# Our Encoding

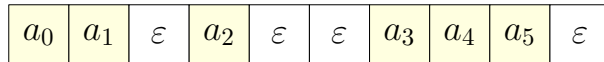
## Clauses (2/2)

- Domain of pseudo-constant  $\alpha$  in  $o$ :  $o'_x \Rightarrow \bigvee_{c \in \text{dom}(\alpha)} [\alpha/c]$   
+ at-most-one constraints over  $\{[\alpha/c] \mid c \in \text{dom}(\alpha)\}$
- Link between pseudo-atom  $f$  and actual atom  $f' := f[\alpha_1/c_1][\alpha_2/c_2]$ :  
 $([\alpha_1/c_1] \wedge [\alpha_2/c_2]) \Rightarrow (f'_x \Leftrightarrow (f')'_x)$
- Frame axioms:  $(f'_x \wedge \neg f'_{x+1}) \Rightarrow (\neg \text{prim}'_x \vee \bigvee_{o \in \text{supp}(\neg f)} o'_x \vee \bigvee_{o \in \text{isupp}(\neg f)} o'_x)$   
+ If  $o \in \text{isupp}(\neg f)$  is active, then active substitutions must unify an effect of  $o$  with  $f$ .

## Further Challenges

- Special handling of actions whose effects may become contradictory
- Enforce restrictions on a pseudo-constant's domain imposed by argument types
- Make “more general” operations subsume “less general” operations

# Anytime Plan Improvement



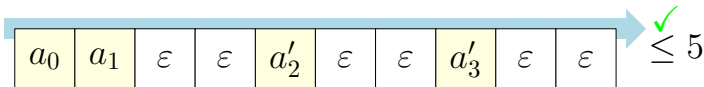
- Maximize number of  $\varepsilon$ -actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)

# Anytime Plan Improvement



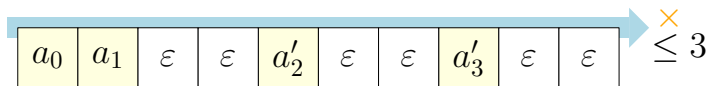
- Maximize number of  $\varepsilon$ -actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)

# Anytime Plan Improvement



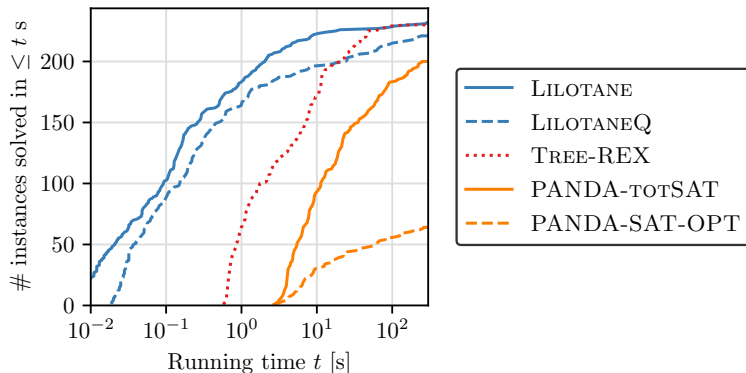
- Maximize number of  $\epsilon$ -actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)

# Anytime Plan Improvement

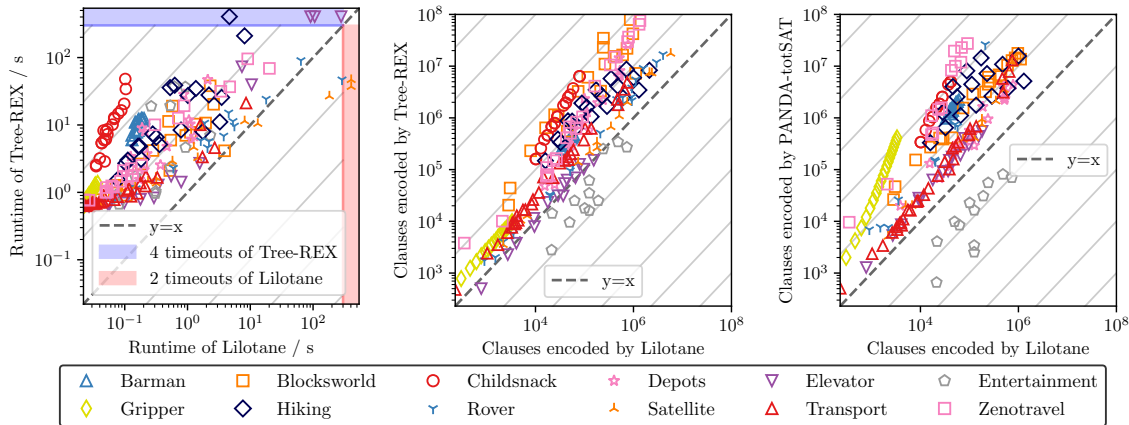


- Maximize number of  $\varepsilon$ -actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)
- Improved encoding exploiting incremental SAT
- Anytime procedure: Cancellable at any time, outputs best plan found

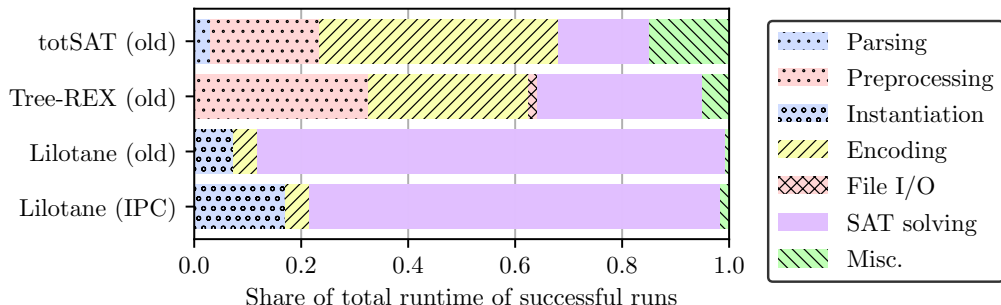
## Comparing SAT-based Planners (1/2)



# Comparing SAT-based Planners (2/2)



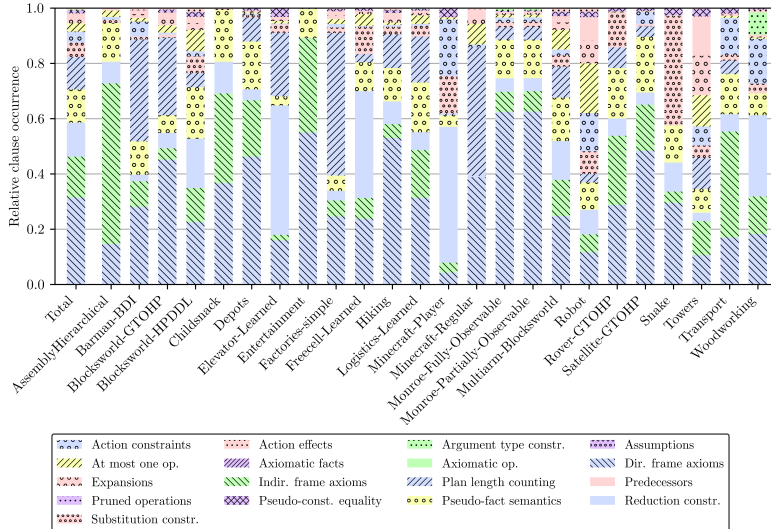
# Tree-REX, totSAT, Lilotane: Share of Runtimes



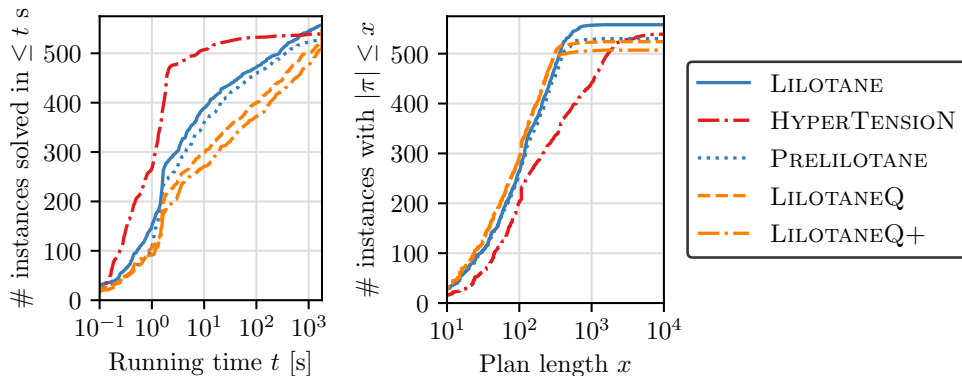
**(old)**: pre-IPC benchmark set (Behnke '18; Schreiber '19)      **(IPC)**: large IPC benchmark set



# Lilotane: Clause Categories

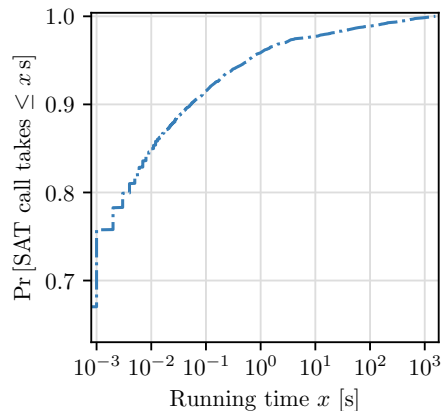
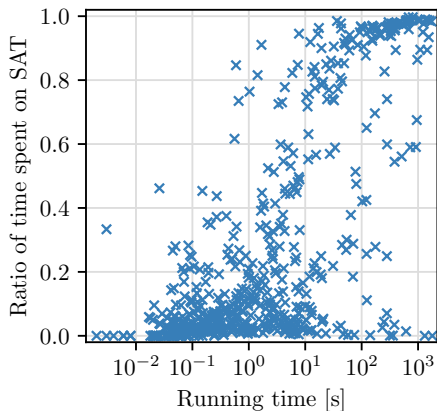


# Evaluation on IPC Benchmarks

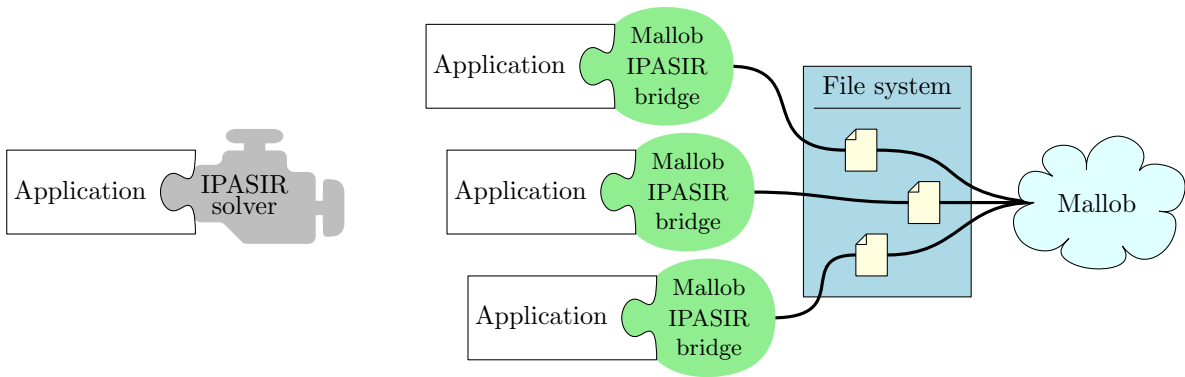


**IPC'20:** 82 problems across 7 domains solved exclusively by Lilotane (winner: 74 problems)

# MALLOTANE: Requirement Analysis



# MALLOTANE: Technology



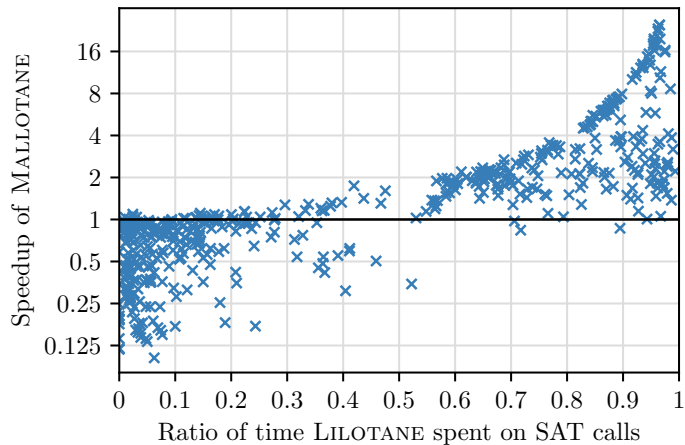
# MALLOTANE: Speedups

Domain	#	min.	med.	geom.	max.	total
Assembly	5	0.15	1.11	0.772	3.19	<b>2.821</b>
Barman	18	0.18	0.84	0.808	4.12	<b>2.069</b>
BlocksworldG	22	0.12	0.81	0.639	1.14	<b>0.946</b>
BlocksworldH	1	0.91	0.91	0.910	<b>0.91</b>	<b>0.914</b>
Childsnack	26	0.19	0.46	0.460	1.09	<b>0.930</b>
Depots	22	0.32	0.77	0.734	1.33	1.196
Elevator	146	0.14	<b>3.40</b>	<b>3.103</b>	24.82	<b>9.054</b>
Entertainment	2	0.93	0.99	0.960	<b>0.99</b>	<b>0.938</b>
Factories	4	0.16	1.75	0.839	2.06	1.835
Freecell	12	1.01	<b>2.28</b>	<b>2.429</b>	5.17	<b>3.228</b>
Hiking	23	0.29	0.82	0.815	2.06	1.455
Logistics	50	0.54	<b>2.02</b>	1.948	3.86	<b>2.171</b>
MinecraftP	4	0.78	<b>2.16</b>	1.717	2.67	<b>2.294</b>
MinecraftR	32	0.37	0.96	0.895	1.09	<b>0.990</b>
MonroeFO	19	0.58	0.91	0.899	1.14	<b>0.949</b>
MonroePO	19	0.83	0.94	1.014	2.68	1.058
Multiarms	4	1.02	<b>2.20</b>	1.649	3.17	<b>2.869</b>
Robot	8	0.10	0.34	0.375	1.37	1.249
Rover	24	0.12	1.89	1.528	5.85	<b>3.477</b>
Satellite	15	0.16	1.04	0.849	1.42	1.125
Snake	20	0.14	0.74	0.586	1.11	1.027
Towers	6	0.20	0.84	0.581	<b>0.89</b>	<b>0.864</b>
Transport	32	0.13	0.54	0.645	8.63	<b>3.207</b>
Woodworking	22	0.14	0.65	0.547	1.01	<b>0.824</b>

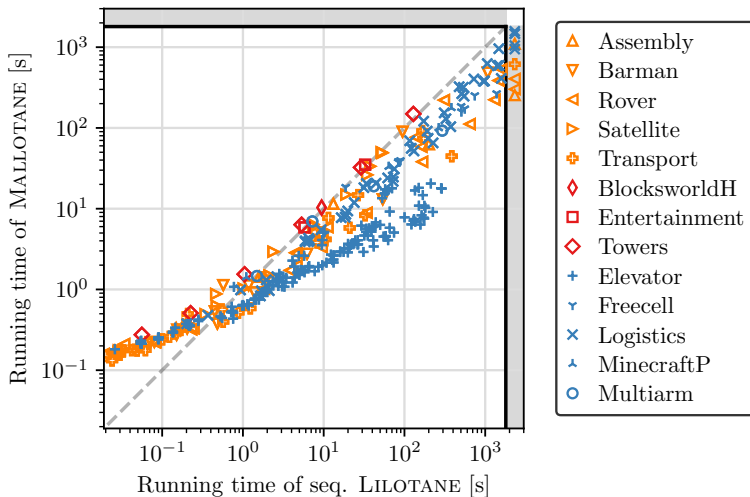
Med./geom./total speedups  $\geq 2$

Max./total speedups  $< 1$

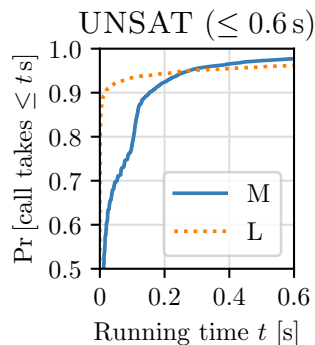
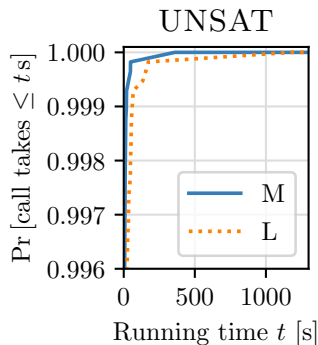
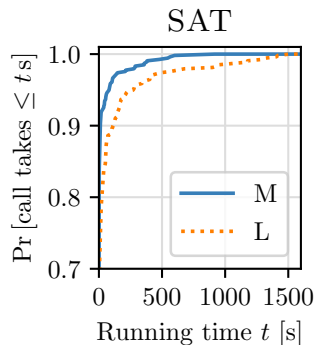
# MALLOTANE: Speedups vs. SAT Ratio



# MALLOTANE: Speedups by Domain

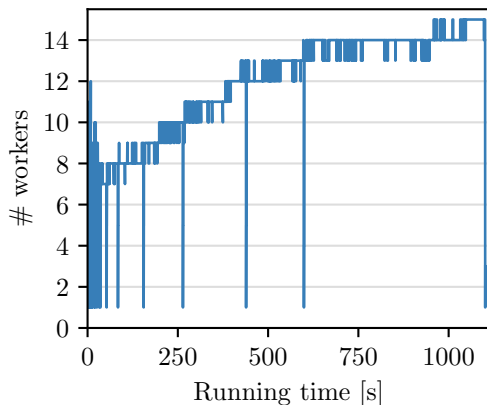
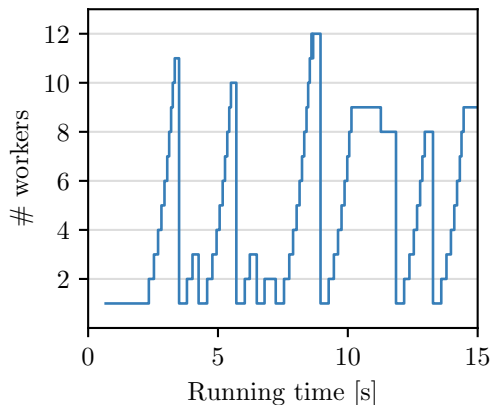


# MALLOTANE: Duration of SAT Calls





# MALLOTANE: Job Volume over Time



Problem from *Assembly* domain · Demand of a job increment doubles each 0.5 s