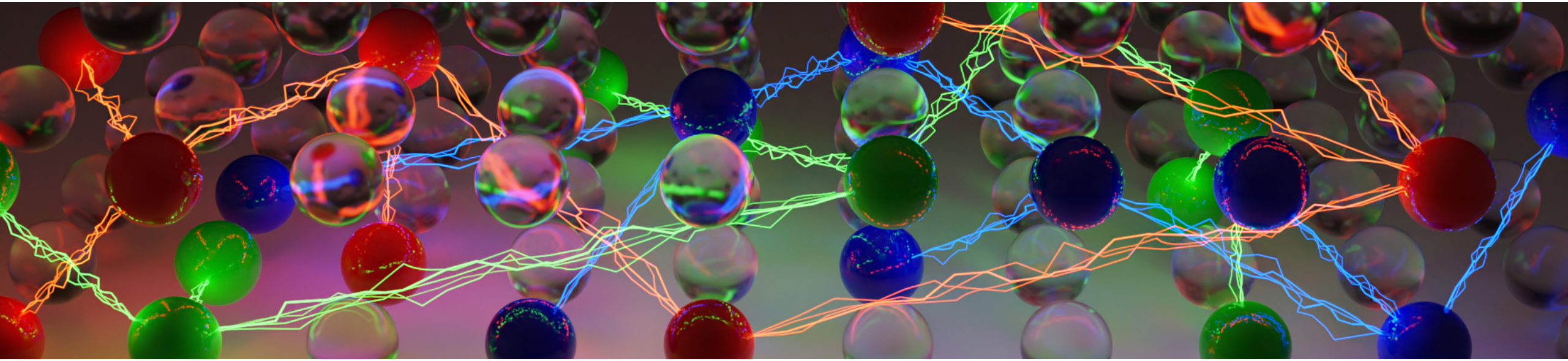


From Distributed SAT to Distributed SMT?

Successes and Challenges

Invited Talk @ SMT Workshop

Dominik Schreiber, Aina Niemetz, Mathias Preiner, and many others | July 25, 2026

SAtRes
Scalable Automated Reasoning

Outline

1. Parallel Constraint Solving – generally and in SMT
2. The Mallob system
3. Experimental methodology and distributed SAT experiments
4. Distributed SMT solving with Bitwuzllob
5. Outlook

Outline

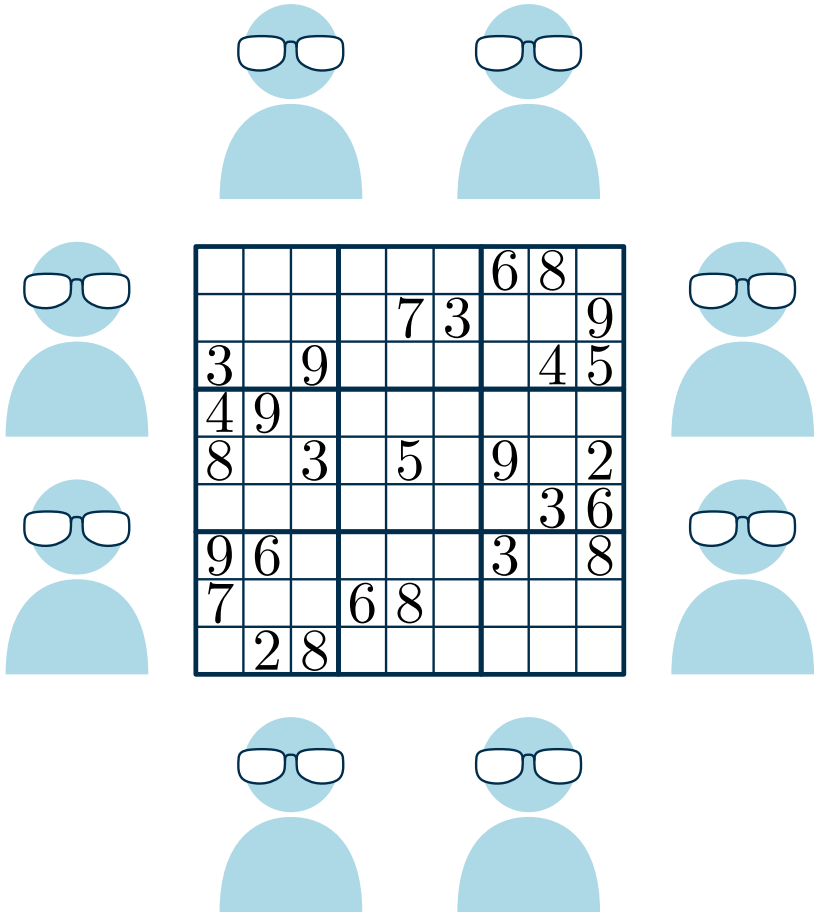
1. **Parallel Constraint Solving** – generally and in SMT
2. The Mallob system
3. Experimental methodology and distributed SAT experiments
4. Distributed SMT solving with Bitwuzllob
5. Outlook

Parallel Constraint Solving: An Analogy [JAIR'24]

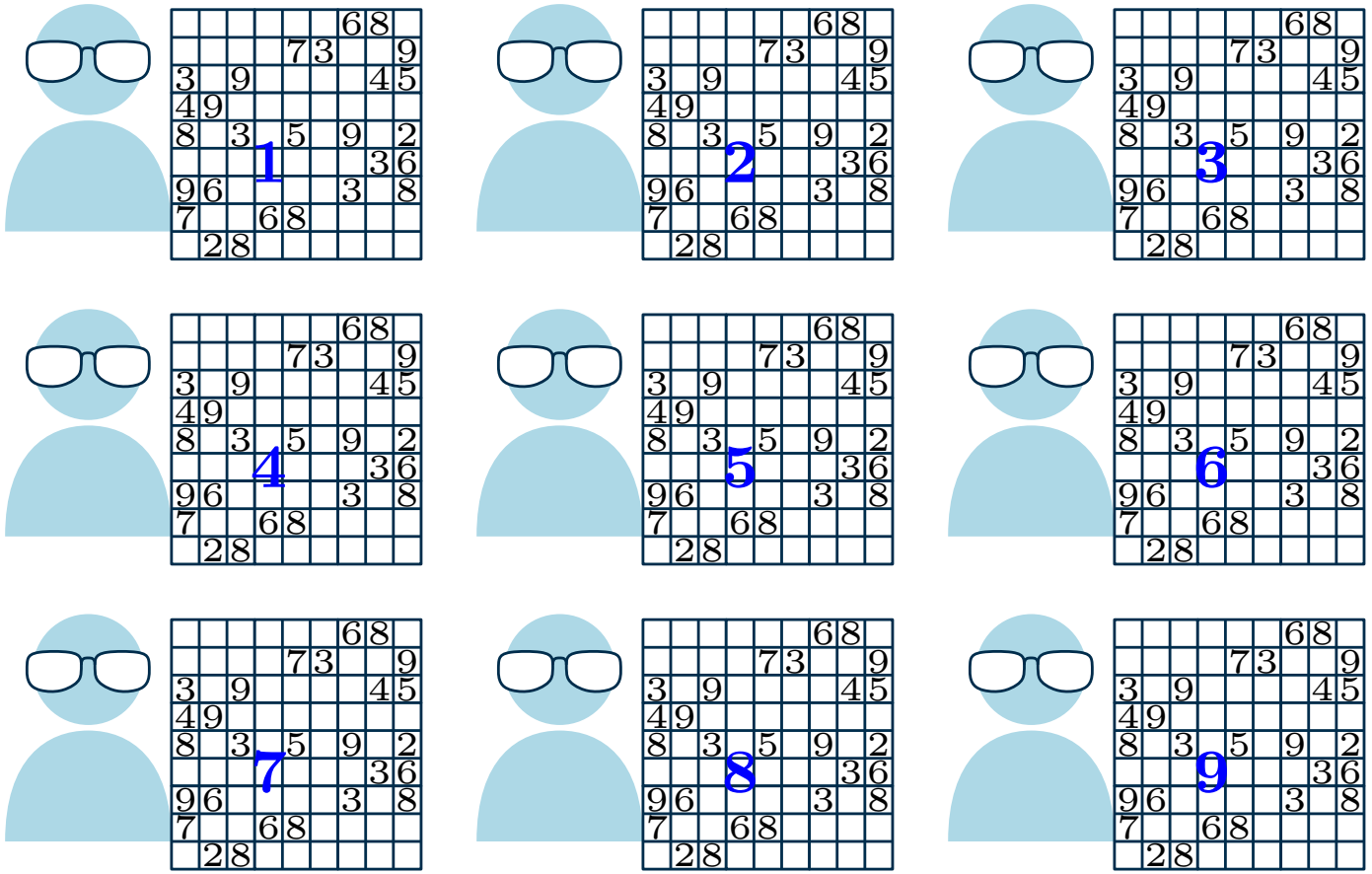
The Council of Nerds

- Complex and large logic puzzle
- *n* puzzle experts at your disposition
  — anti-social: work best if left undisturbed

How do we employ and "orchestrate" our experts?



Approach I: Search Space Partitioning



- Partition search space at some decisions
  $\Rightarrow$ Independent subproblems
- State-of-the-art for certain hard combinatorial problems

Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

- Cubes generated with Look-ahead solver (expensive but more informed branching heuristic)
- Unlikely that any of the workers will run out tasks $\Rightarrow$ Hope of good load balancing

Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

- Cubes generated with Look-ahead solver (expensive but more informed branching heuristic)
- Unlikely that any of the workers will run out tasks ⇒ Hope of good load balancing
- State-of-the-art for certain hard combinatorial problems – e.g., “Pythagorean Triples”, ∼200TB proof [3]

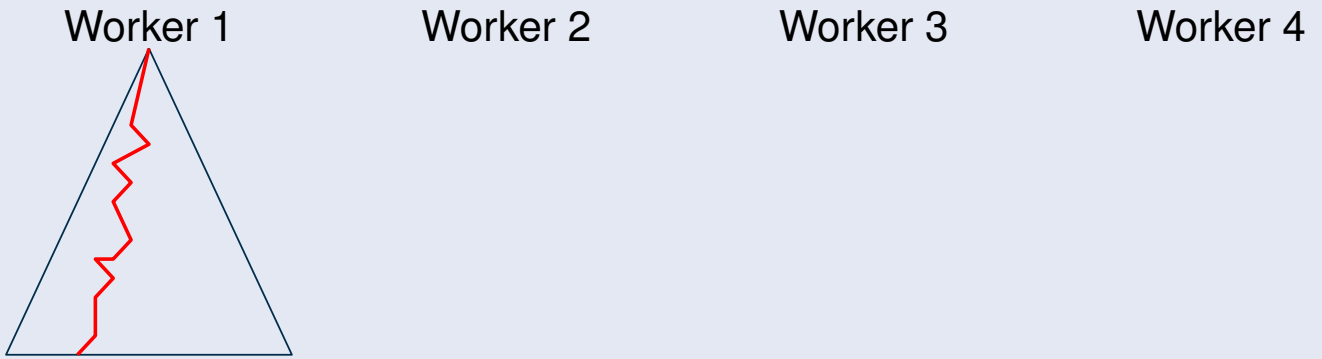
Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

- Cubes generated with Look-ahead solver (expensive but more informed branching heuristic)
- Unlikely that any of the workers will run out tasks ⇒ Hope of good load balancing
- State-of-the-art for certain hard combinatorial problems – e.g., “Pythagorean Triples”, ∼200TB proof [3]

Dynamic Load Balancing: Work stealing



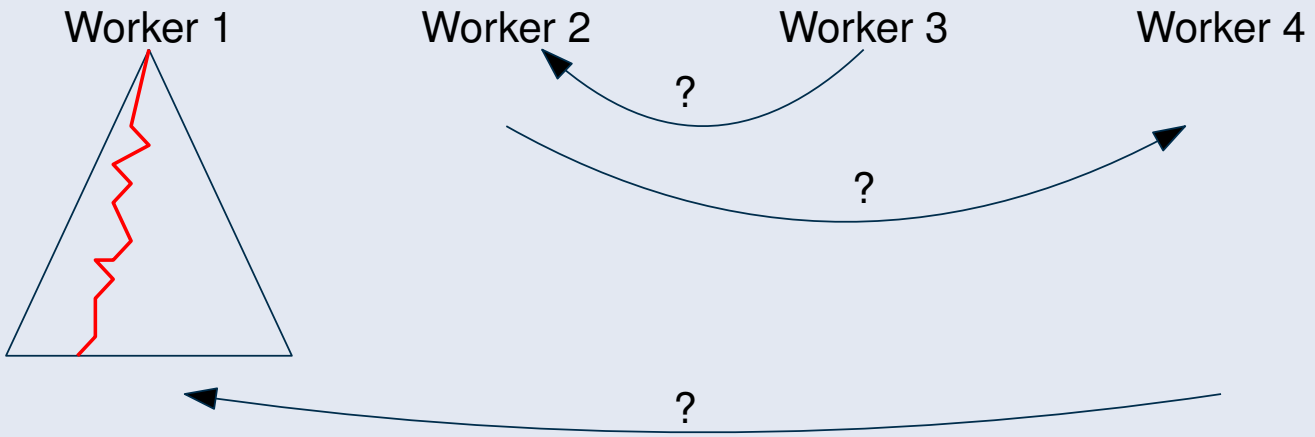
Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

- Cubes generated with Look-ahead solver (expensive but more informed branching heuristic)
- Unlikely that any of the workers will run out tasks ⇒ Hope of good load balancing
- State-of-the-art for certain hard combinatorial problems – e.g., “Pythagorean Triples”, ∼200TB proof [3]

Dynamic Load Balancing: Work stealing



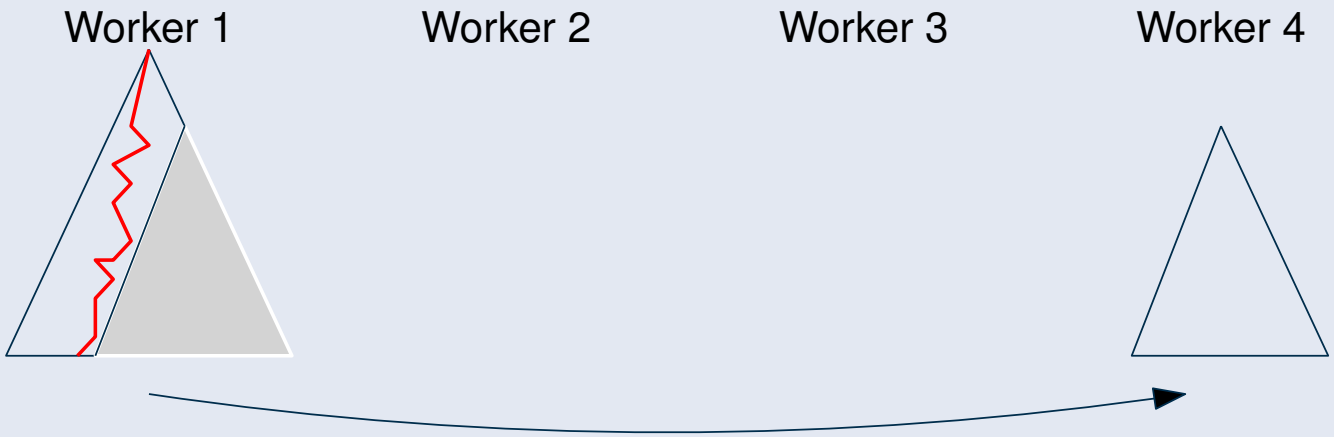
Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

- Cubes generated with Look-ahead solver (expensive but more informed branching heuristic)
- Unlikely that any of the workers will run out tasks ⇒ Hope of good load balancing
- State-of-the-art for certain hard combinatorial problems – e.g., “Pythagorean Triples”, ∼200TB proof [3]

Dynamic Load Balancing: Work stealing



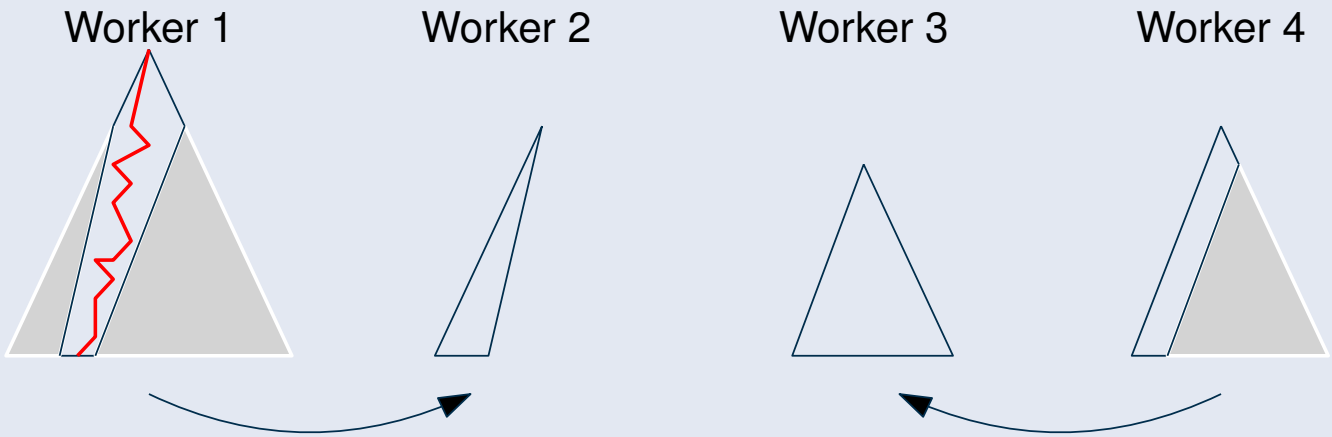
Partitioning Solvers: Cube&Conquer, Work Stealing

Static Load Balancing: Cube&Conquer (2011) [4]

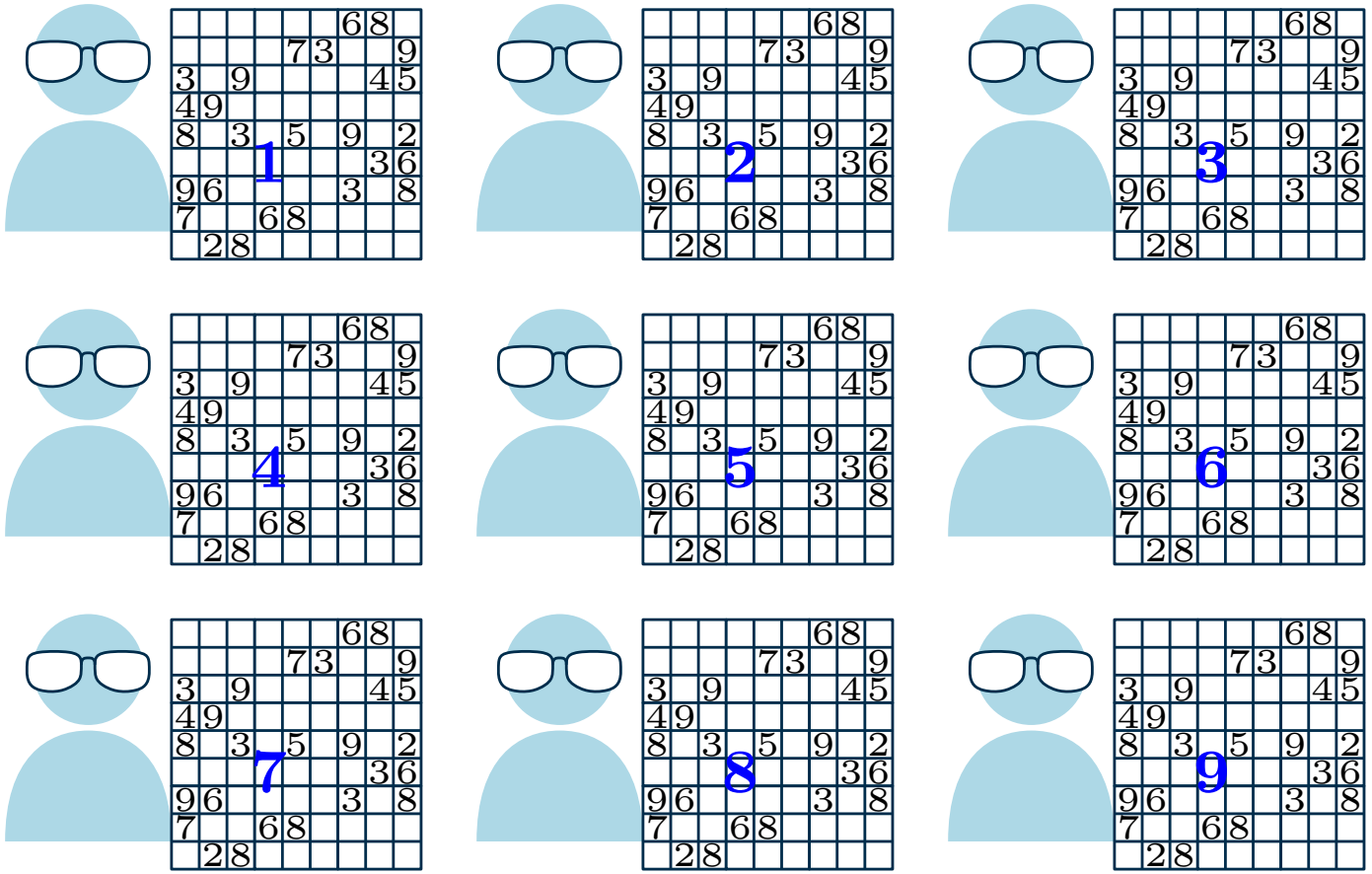
Generate a large amount (millions) of partial assignments (“**cubes**”) and randomly assign them to workers.

- Cubes generated with Look-ahead solver (expensive but more informed branching heuristic)
- Unlikely that any of the workers will run out tasks ⇒ Hope of good load balancing
- State-of-the-art for certain hard combinatorial problems – e.g., “Pythagorean Triples”, ∼200TB proof [3]

Dynamic Load Balancing: Work stealing

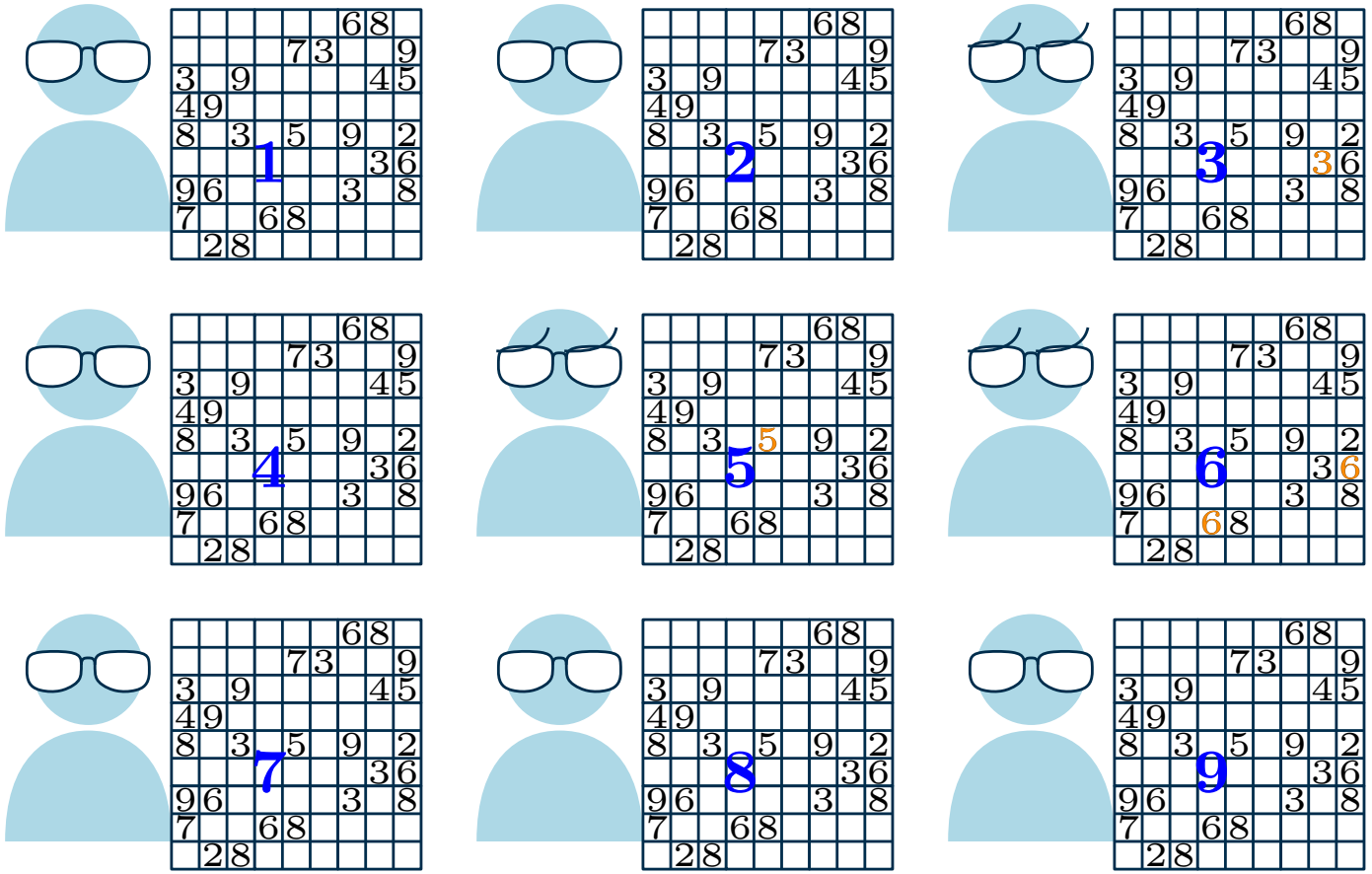


Approach I: Search Space Partitioning (cont'd)



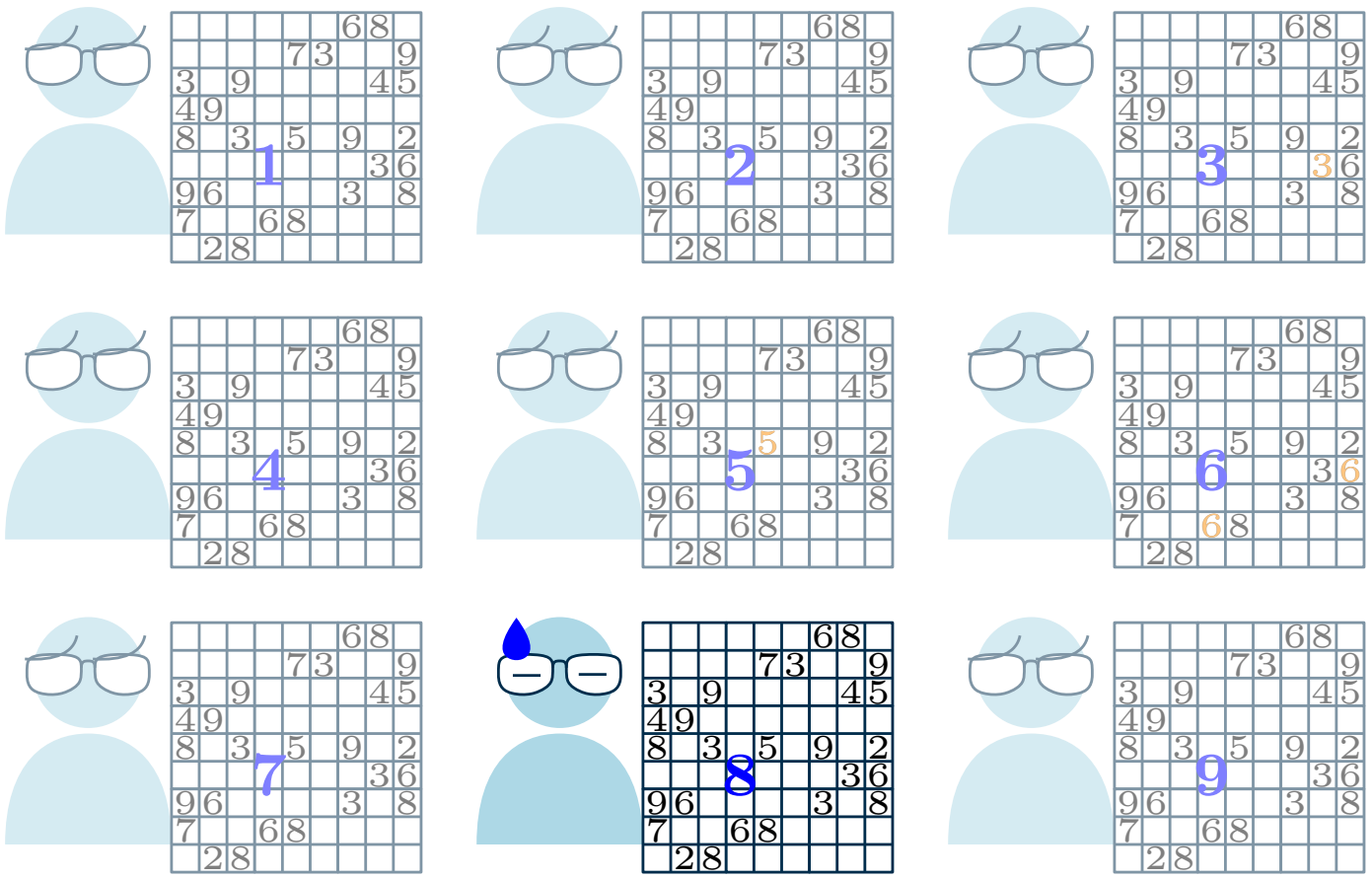
- Partition search space at some decisions
  $\Rightarrow$ Independent subproblems
- State-of-the-art for certain hard combinatorial problems
- Difficult to balance the load
  — uneven or useless splits
  — harder than the original problem!
- Subtle redundancies remain across sub-tasks!

Approach I: Search Space Partitioning (cont'd)



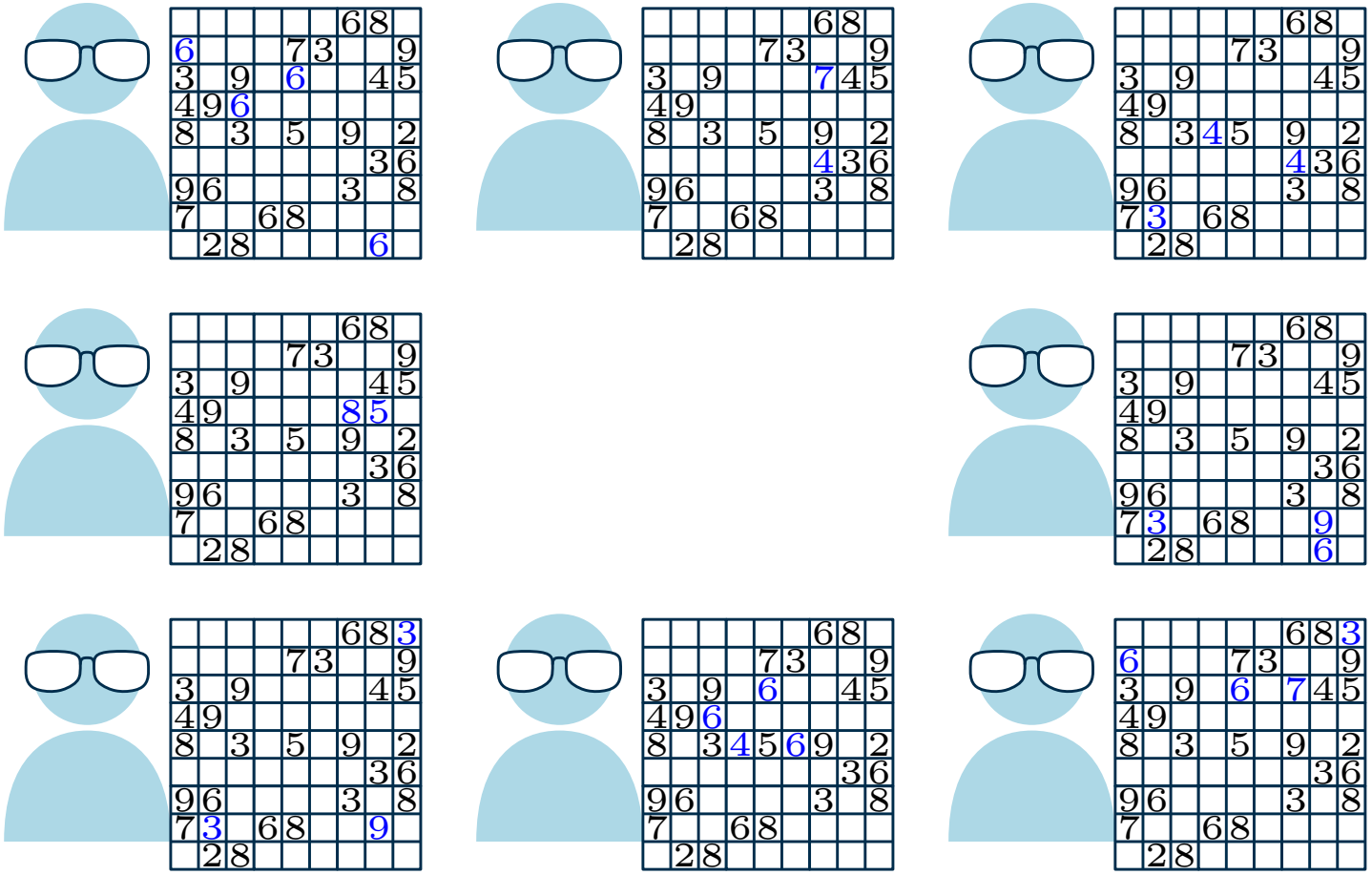
- Partition search space at some decisions
  $\Rightarrow$ Independent subproblems
- State-of-the-art for certain hard combinatorial problems
- Difficult to balance the load
  — uneven or useless splits
  — harder than the original problem!
- Subtle redundancies remain across sub-tasks!

Approach I: Search Space Partitioning (cont'd)

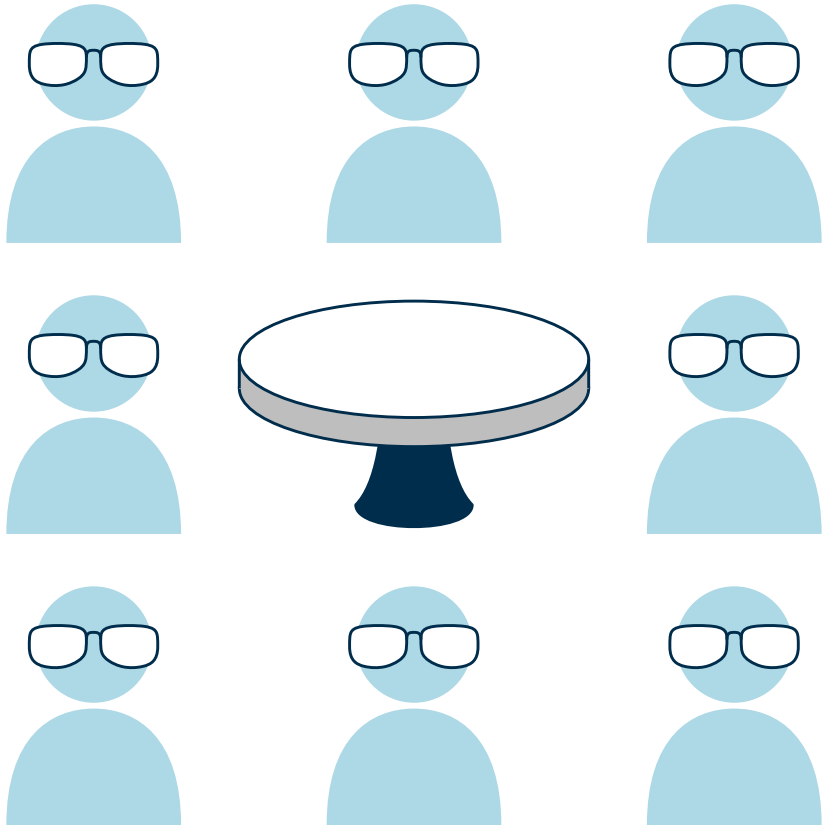


- Partition search space at some decisions
  $\Rightarrow$ Independent subproblems
- State-of-the-art for certain hard combinatorial problems
- Difficult to balance the load
  — uneven or useless splits
  — harder than the original problem!
- Subtle redundancies remain across sub-tasks!

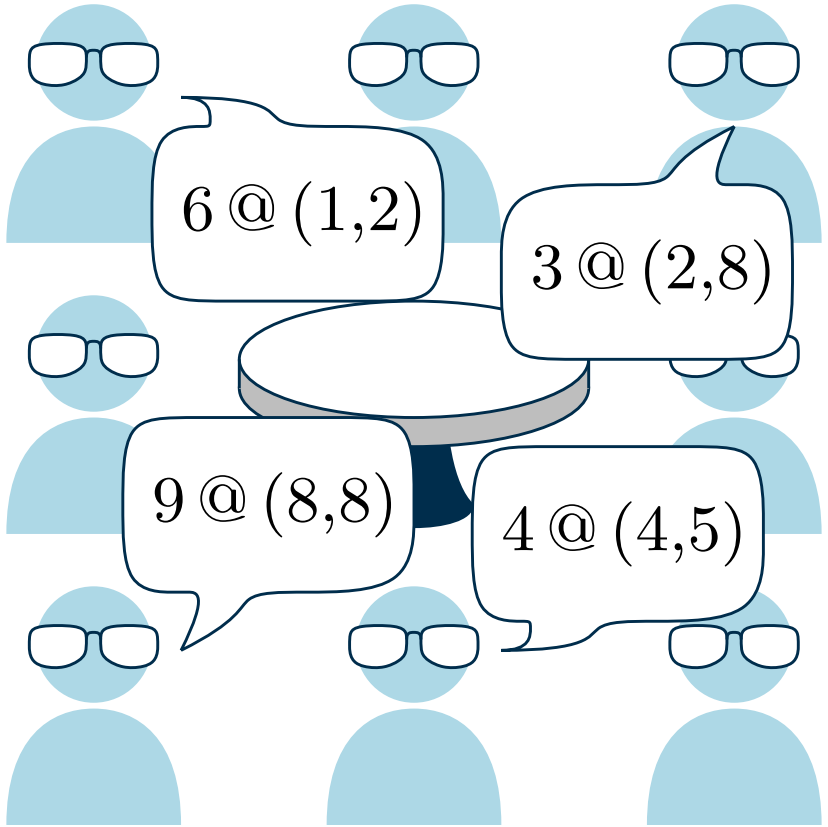
Approach II: Portfolio



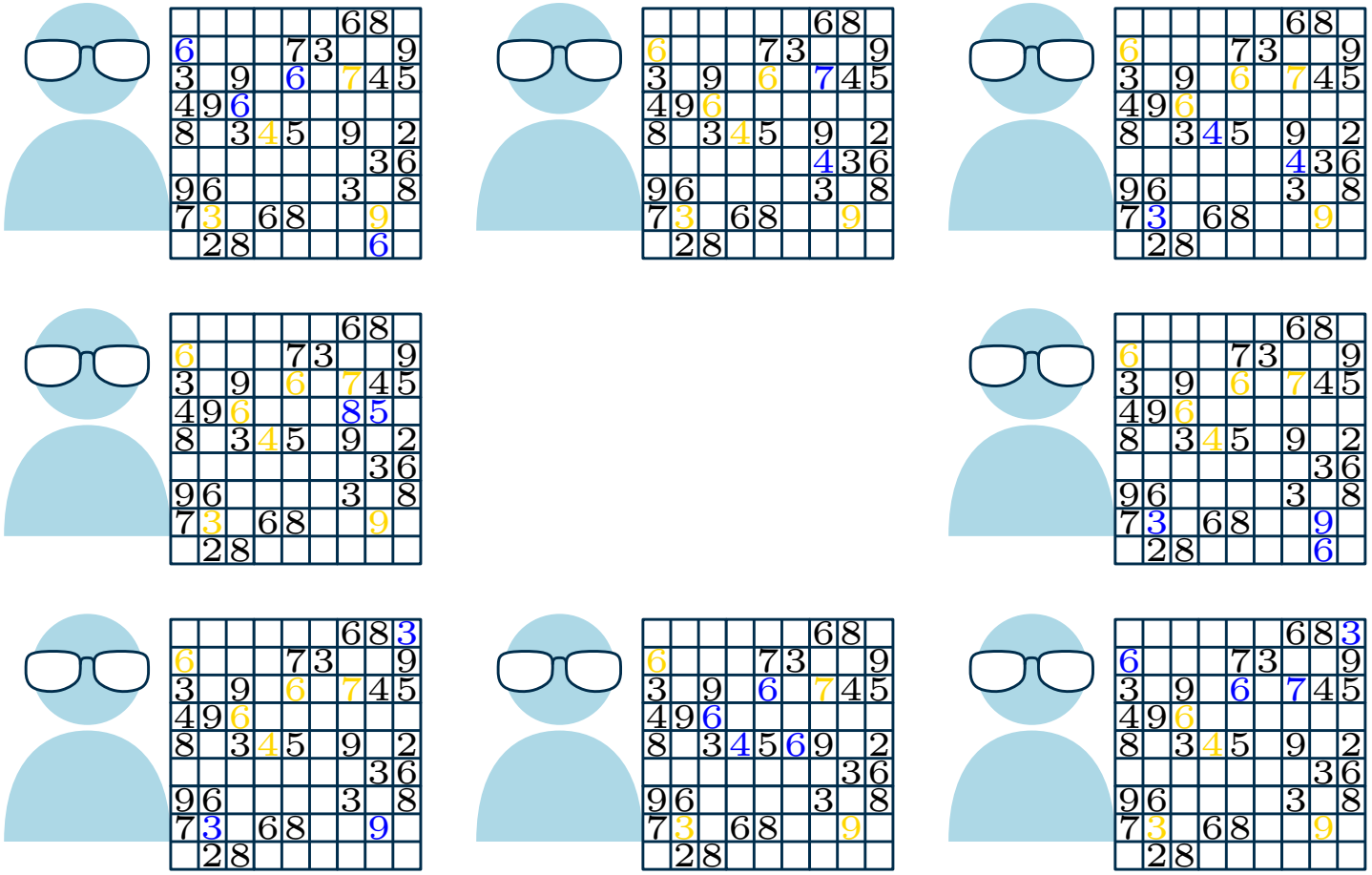
Approach II: Portfolio + Cooperation



Approach II: Portfolio + Cooperation



Approach II: Portfolio + Cooperation



Samples of Parallel SMT

A. Wilson, A. Noetzli, A. Reynolds, B. Cook, C. Tinelli, C. Barrett:
Partitioning Strategies for Distributed SMT Solving (FMCAD'23)
– Centralized partitioning + Portfolio
– Experiments: $\leq$ 256 cores

C. Barrett, P. Chen, B. Cook, B. Dutertre, R. Jones, N. Le, A. Reynolds, K. Sheth, C. Stephens, M. Whalen:
SMT-D: New Strategies for Portfolio-based SMT Solving (FMCAD'24)
– Portfolio + Centralized lemma sharing
– Experiments: $\leq$ 64 cores

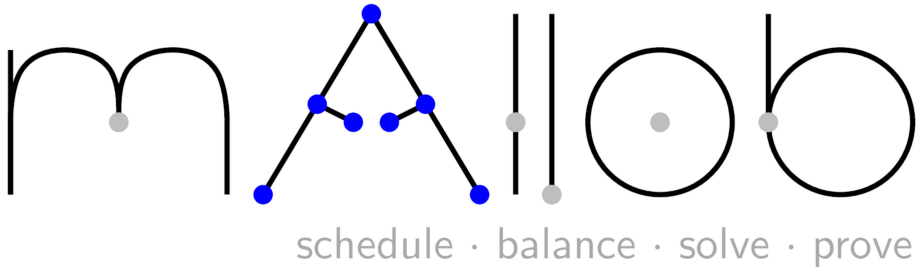
M. Zhao, S. Cai, & Y. Qian:
Distributed SMT Solving Based on Dynamic Variable-Level Partitioning (CAV'24)
– Dynamic partitioning
– Experiments: $\leq$ 32 cores

T. Kolárik, A. Hyvärinen, S. Asadzadeh, N. Sharygina:
Parallel SMT Solving via Iterative Tree Partitioning (TACAS'26)
– Dynamic partitioning + Centralized lemma sharing
– Experiments: $\leq$ 8 cores (+ SMT Comp results)

Outline

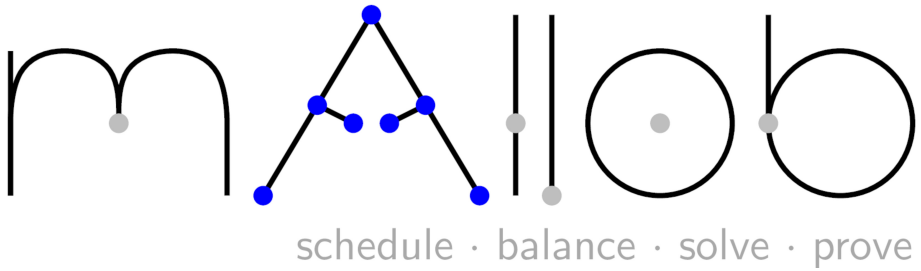
1. Parallel Constraint Solving – generally and in SMT
2. **The Mallob system**
3. Experimental methodology and distributed SAT experiments
4. Distributed SMT solving with Bitwuzllob
5. Outlook

The Mallob System in a Nutshell

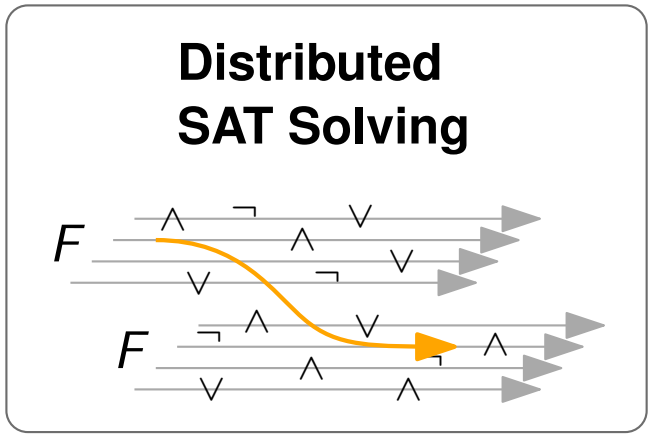


https://s.kit.edu/mallob

The Mallob System in a Nutshell



SAT'21 JAIR'24 SAT'25



https://s.kit.edu/mallob

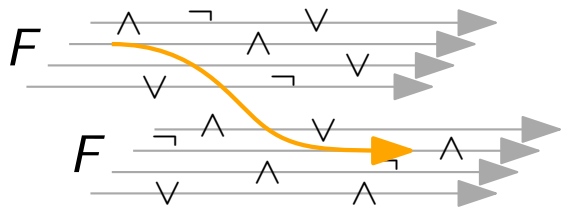
The Mallob System in a Nutshell

mallob

schedule · balance · solve · prove

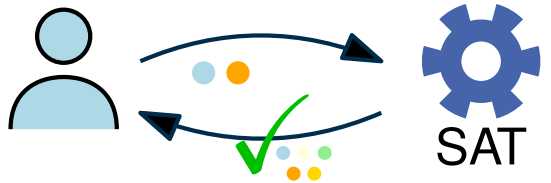
SAT'21 JAIR'24 SAT'25

Distributed SAT Solving



TACAS'26b CAV'26

Incremental Queries



https://s.kit.edu/mallob

The Mallob System in a Nutshell

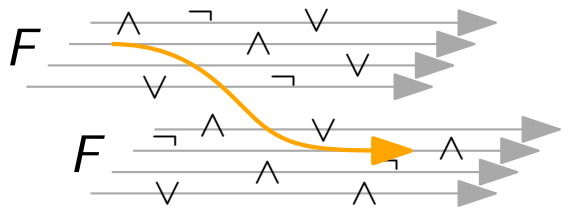
mallob

schedule · balance · solve · prove

https://s.kit.edu/mallob

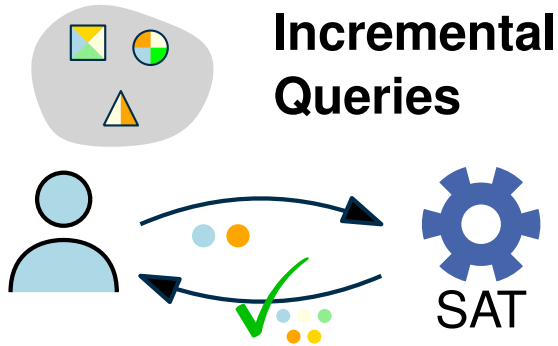
SAT'21 JAIR'24 SAT'25

Distributed SAT Solving



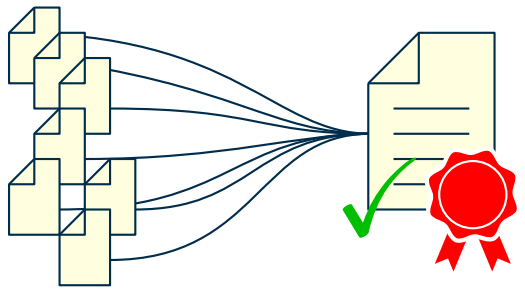
TACAS'26b CAV'26

Incremental Queries



TACAS'23 SAT'24 TACAS'26b FMCAD'26

Scalable Certification



The Mallob System in a Nutshell

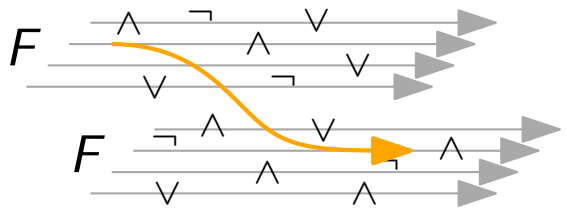
mallob

schedule · balance · solve · prove

https://s.kit.edu/mallob

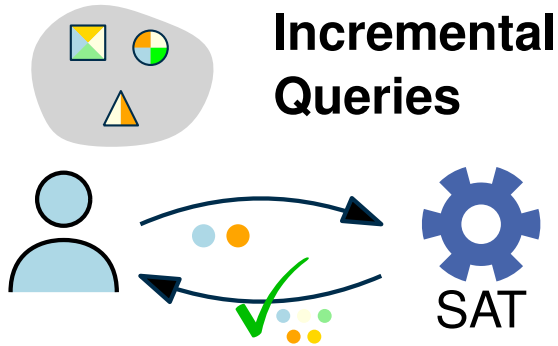
SAT'21 JAIR'24 SAT'25

Distributed SAT Solving



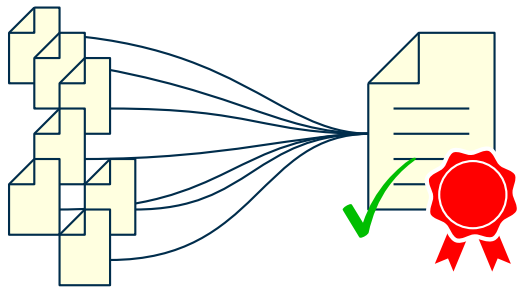
TACAS'26b CAV'26

Incremental Queries



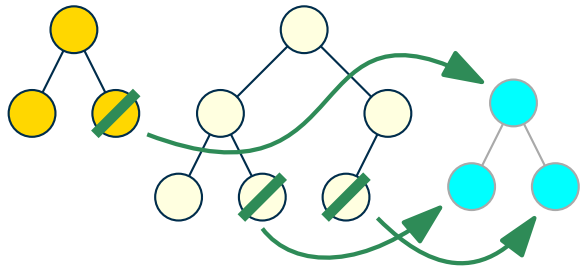
TACAS'23 SAT'24 TACAS'26b FMCAD'26

Scalable Certification

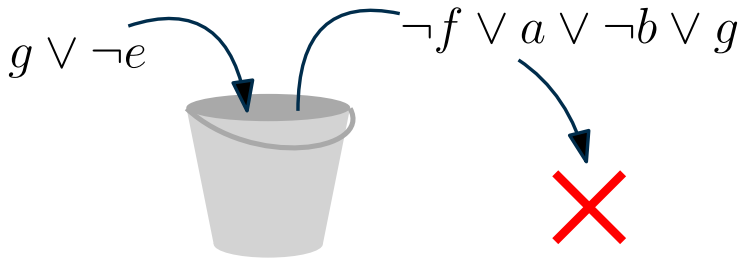


SAT'21 Euro-Par'22

Flexible Multitasking



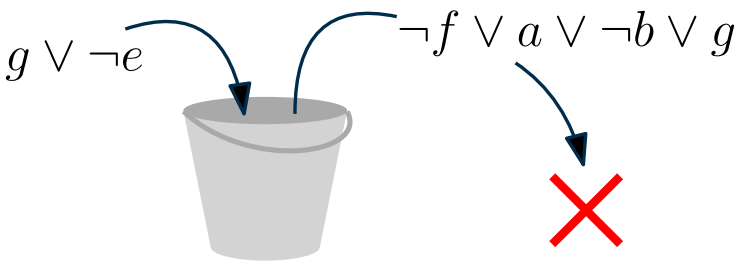
Robust Clause Sharing in MallobSat [JAIR'24]



No short clause left behind

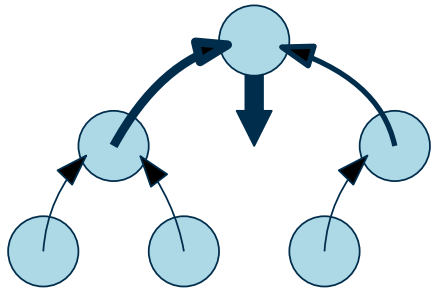
- Prioritize clauses by length at all stages
- Fixed-budget buffers discard longer clauses

Robust Clause Sharing in MallobSat [JAIR'24]



No short clause left behind

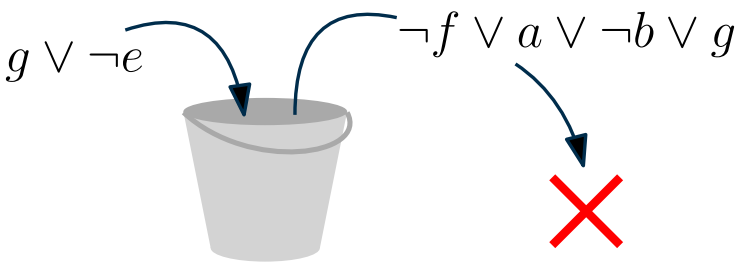
- Prioritize clauses by length at all stages
- Fixed-budget buffers discard longer clauses



Fixed budget for the globally best distinct clauses

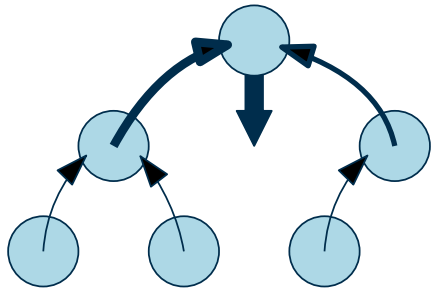
- "Bubble up" shortest clauses along process tree
- Length-one clauses can always be shared for free

Robust Clause Sharing in MallobSat [JAIR'24]



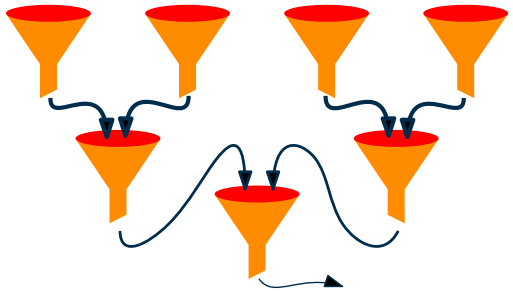
No short clause left behind

- Prioritize clauses by length at all stages
- Fixed-budget buffers discard longer clauses



Fixed budget for the globally best distinct clauses

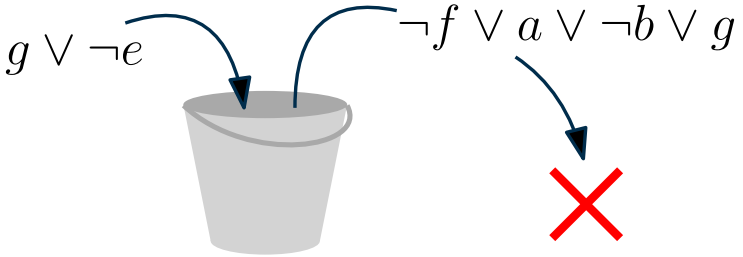
- "Bubble up" shortest clauses along process tree
- Length-one clauses can always be shared for free



Filter all duplicate and reflected clauses

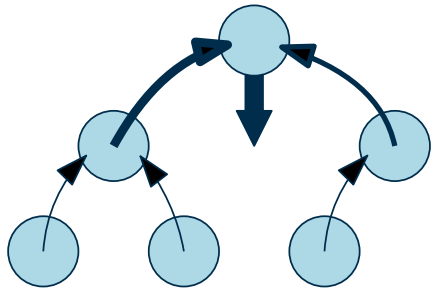
- Distribute filter memory across all processes
- Reset filters after some time period

Robust Clause Sharing in MallobSat [JAIR'24]



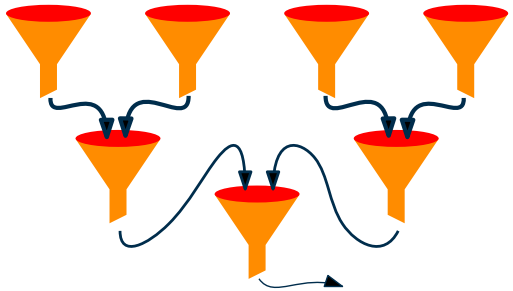
No short clause left behind

- Prioritize clauses by length at all stages
- Fixed-budget buffers discard longer clauses



Fixed budget for the globally best distinct clauses

- "Bubble up" shortest clauses along process tree
- Length-one clauses can always be shared for free



Filter all duplicate and reflected clauses

- Distribute filter memory across all processes
- Reset filters after some time period

Monitor and control sharing volume

- Sharing volume sublinear in # solver threads
- Compensate for unused sharing budget

Outline

1. Parallel Constraint Solving – generally and in SMT
2. The Mallob system
3. **Experimental methodology and distributed SAT experiments**
4. Distributed SMT solving with Bitwuzllob
5. Outlook

Best Practices for Parallel Solver Evaluation

Best Practices for Parallel Solver Evaluation

1. Benchmark against the **best available sequential solver** (based on the same paradigm as your solver), not just your parallel solver @ 1 core (“self-speedup”).

Best Practices for Parallel Solver Evaluation

1. Benchmark against the **best available sequential solver** (based on the same paradigm as your solver), not just your parallel solver @ 1 core (“self-speedup”).

2. **Report scaling** by exponentially increasing the number of cores / nodes.

Best Practices for Parallel Solver Evaluation

1. Benchmark against the **best available sequential solver** (based on the same paradigm as your solver), not just your parallel solver @ 1 core (“self-speedup”).

2. **Report scaling** by exponentially increasing the number of cores / nodes.

3. **Provide speedups** over the sequential baseline at different scales, aggregating per-input speedups via geometric mean and/or median (but never arithmetic mean!).

Best Practices for Parallel Solver Evaluation

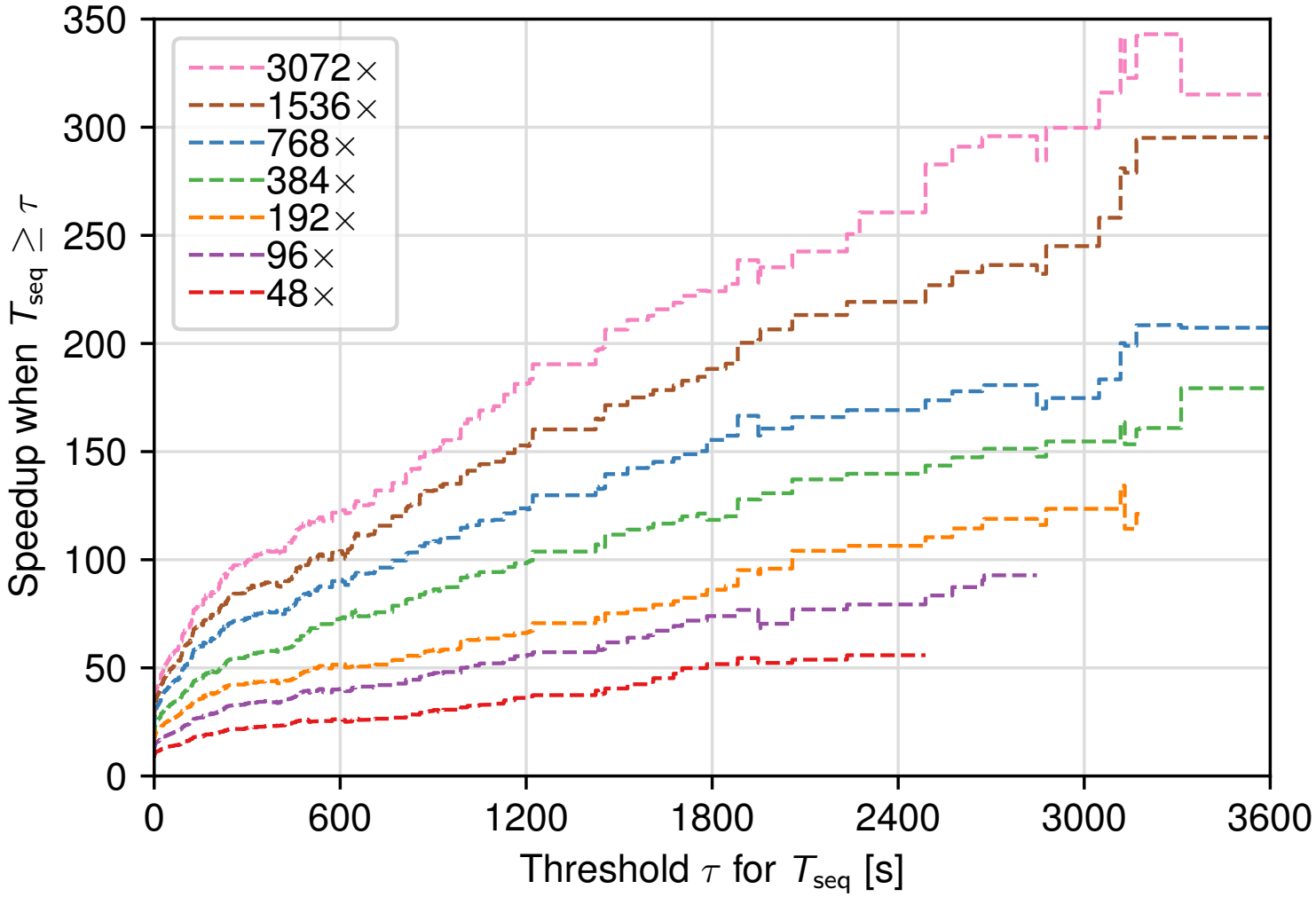
1. Benchmark against the **best available sequential solver** (based on the same paradigm as your solver), not just your parallel solver @ 1 core (“self-speedup”).

2. **Report scaling** by exponentially increasing the number of cores / nodes.

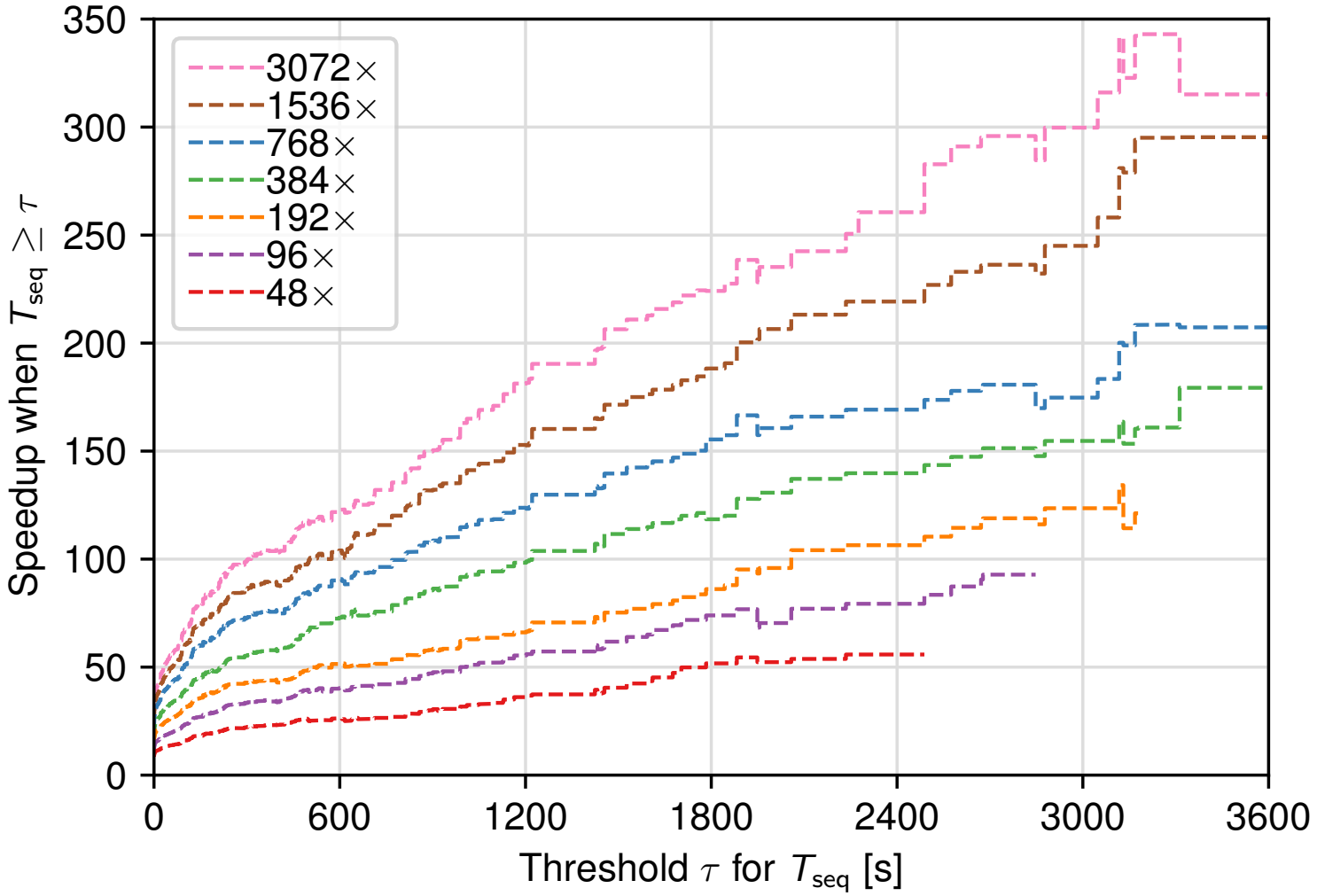
3. **Provide speedups** over the sequential baseline at different scales, aggregating per-input speedups via geometric mean and/or median (but never arithmetic mean!).

4. Omit inputs unsolved by sequential solver from speedup calculations to keep speedups interpretable and report difference in solved inputs separately.

MallobSat: Scaling [SAT'25]

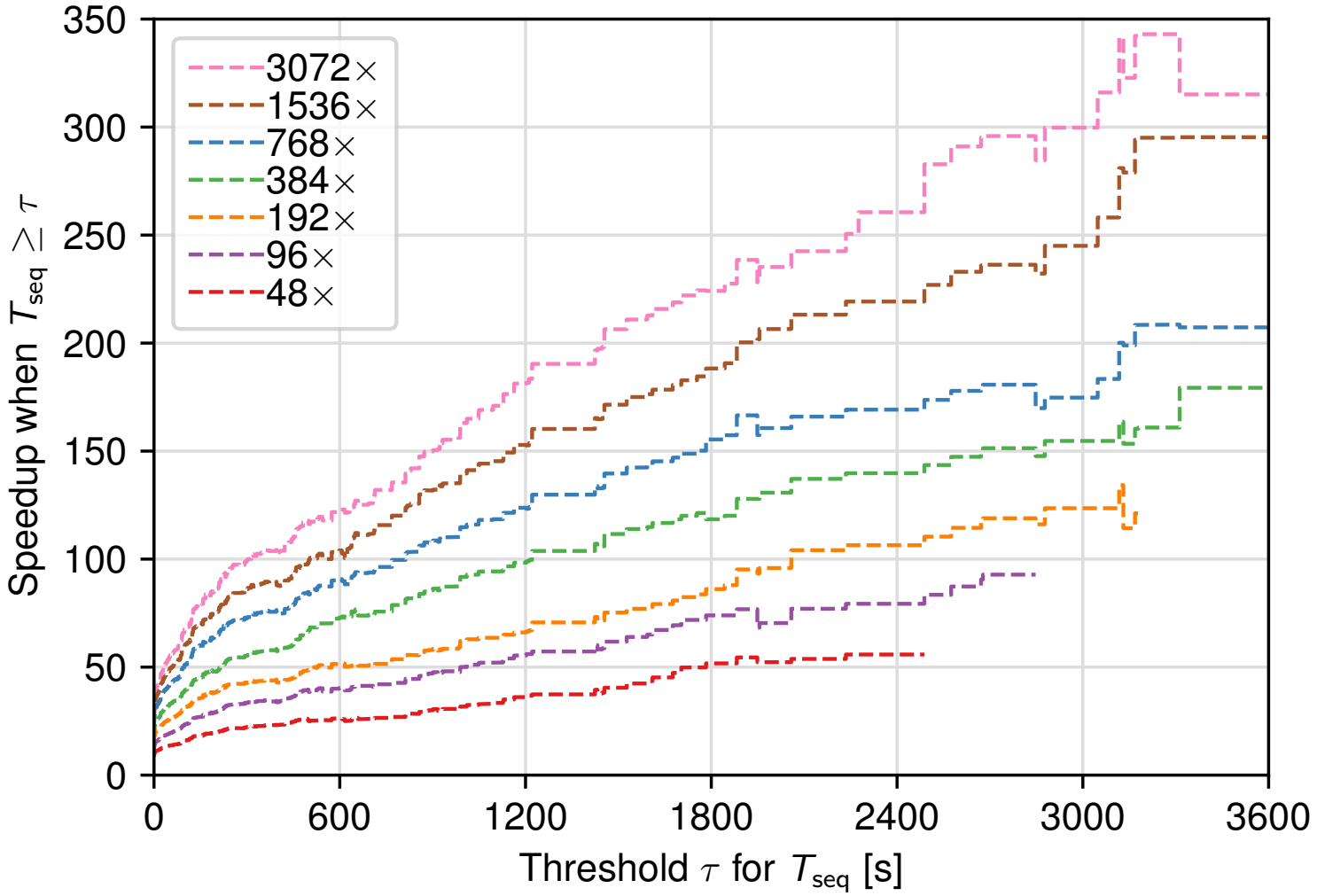


MallobSat: Scaling [SAT'25]



- Only using Kissat solver
- Only using very light, trivial diversification (random seeds, initial variable polarities)
- Not a portfolio solver but a **clause sharing solver** with uniform parallel CDCL!
  (Enabling all diversification does further improve performance)

MallobSat: Scaling [SAT'25]



- Only using Kissat solver
- Only using very light, trivial diversification (random seeds, initial variable polarities)
- Not a portfolio solver but a **clause sharing solver** with uniform parallel CDCL!
  (Enabling all diversification does further improve performance)

SAT Competition 2026:

- Best seq.: 276 solved within 5000 s
- MallobSat^{32}: 300 solved within 1000 s
- MallobSat^{800}: 301 solved within 200 s

Outline

1. Parallel Constraint Solving – generally and in SMT
2. The Mallob system
3. Experimental methodology and distributed SAT experiments
4. **Distributed SMT solving with Bitwuzllob**
5. Outlook

Towards SMT

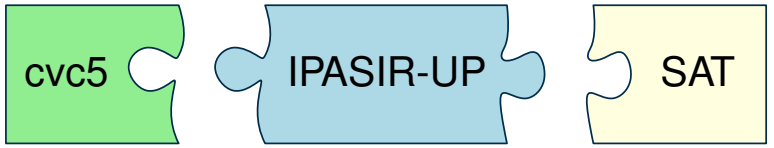
How can we exploit this technology for SMT?

Towards SMT

How can we exploit this technology for SMT?

CDCL($\mathcal{T}$)?

- Fine-grained interactions beetween solver and theory
- Spends large amounts of time outside of SAT
- Large sequence of trivial SAT queries
- IPASIR-UP as a bridge?

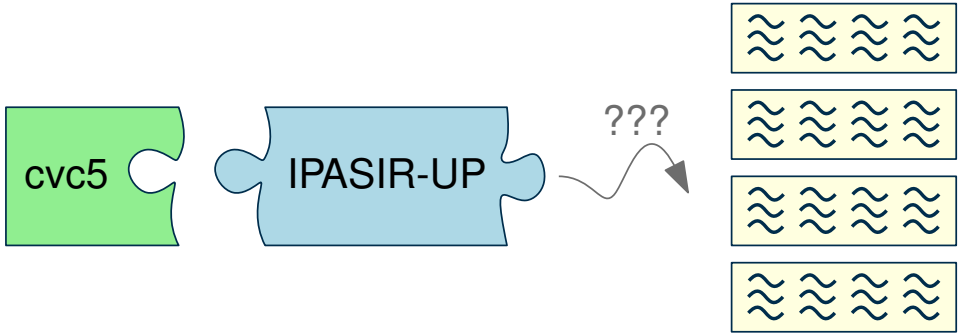


Towards SMT

How can we exploit this technology for SMT?

CDCL($\mathcal{T}$)?

- Fine-grained interactions beetween solver and theory
- Spends large amounts of time outside of SAT
- Large sequence of trivial SAT queries
- IPASIR-UP as a bridge? – semantic gap to parallel system!

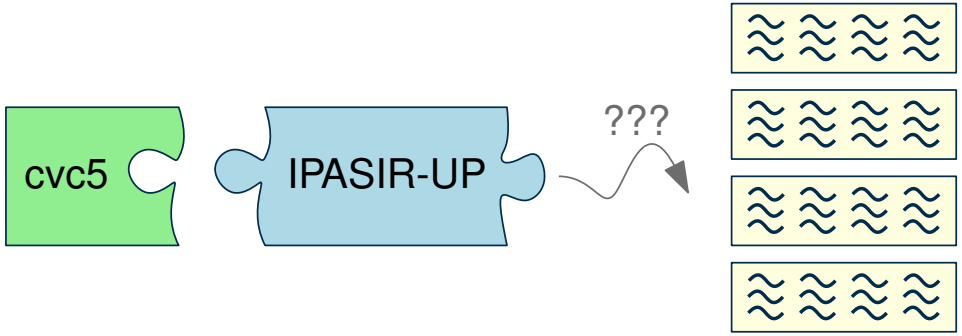


Towards SMT

How can we exploit this technology for SMT?

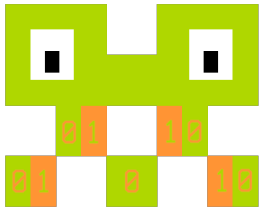
CDCL($\mathcal{T}$)?

- Fine-grained interactions beetween solver and theory
- Spends large amounts of time outside of SAT
- Large sequence of trivial SAT queries
- IPASIR-UP as a bridge? – semantic gap to parallel system!



More coarse grained SAT interactions? **Bitwuzla** to the rescue!

- Lemmas-on-demand paradigm with bit-vector abstraction
- Spends almost entire time in SAT solver
- Uses off-the-shelf incremental SAT solver
- Mallob covers all required features!



A. Niemetz & M. Preiner, CAV'23

Bitwuzla + Mallob = Bitwuzllob

How do we achieve an efficient integration?

Bitwuzla + Mallob = Bitwuzllob

How do we achieve an efficient integration?

- Link Mallob(Sat) into Bitwuzla like any other SAT solver?

Bitwuzla + Mallob = Bitwuzllob

How do we achieve an efficient integration?

- Link Mallob(Sat) into Bitwuzla like any other SAT solver?
  – impossible (isolated vs. distributed application)
- Let Bitwuzla talk to Mallob “deamon” via some interface?

Bitwuzla + Mallob = Bitwuzllob

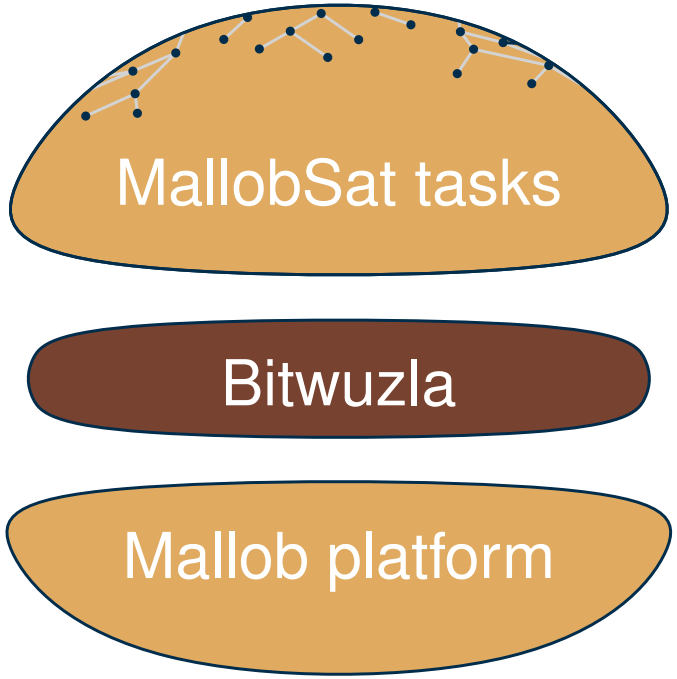
How do we achieve an efficient integration?

- Link Mallob(Sat) into Bitwuzla like any other SAT solver?
  – impossible (isolated vs. distributed application)
- Let Bitwuzla talk to Mallob “deamon” via some interface?
  – complicated / brittle, inefficient (per-call overheads!)

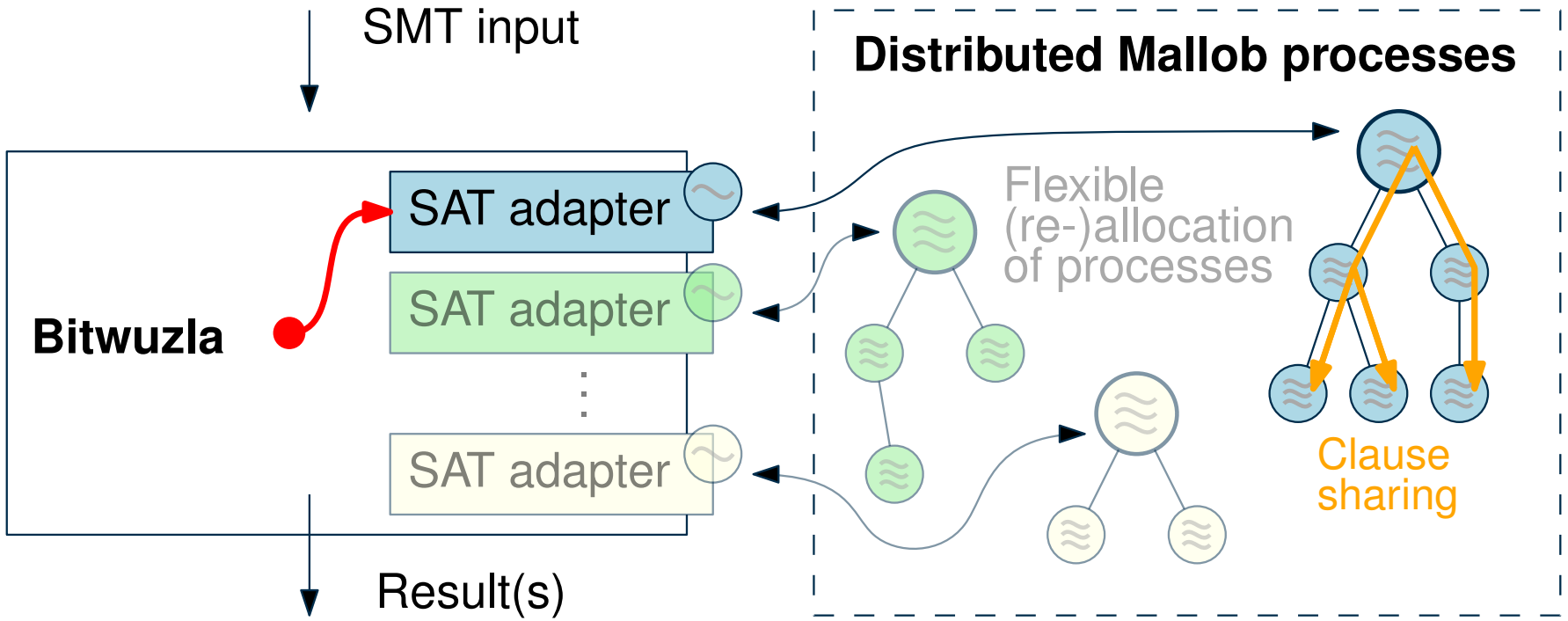
Bitwuzla + Mallob = Bitwuzllob

How do we achieve an efficient integration?

- Link Mallob(Sat) into Bitwuzla like any other SAT solver?
  – impossible (isolated vs. distributed application)
- Let Bitwuzla talk to Mallob “deamon” via some interface?
  – complicated / brittle, inefficient (per-call overheads!)
- The Mallob Burger
  – **Mallob** process receives and initializes SMT task
  – **Bitwuzla** runs and emits SAT queries
  – **Mallob** processes SAT queries by spawning MallobSat tasks

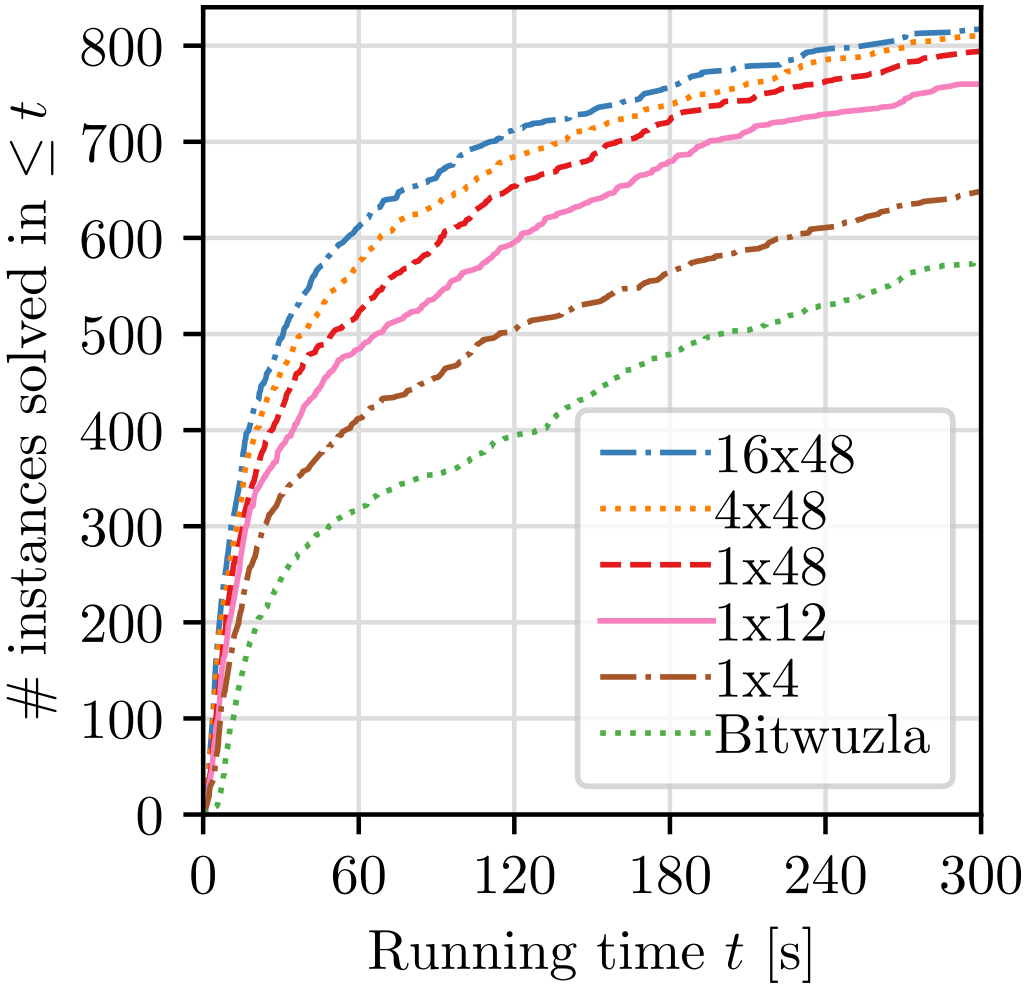


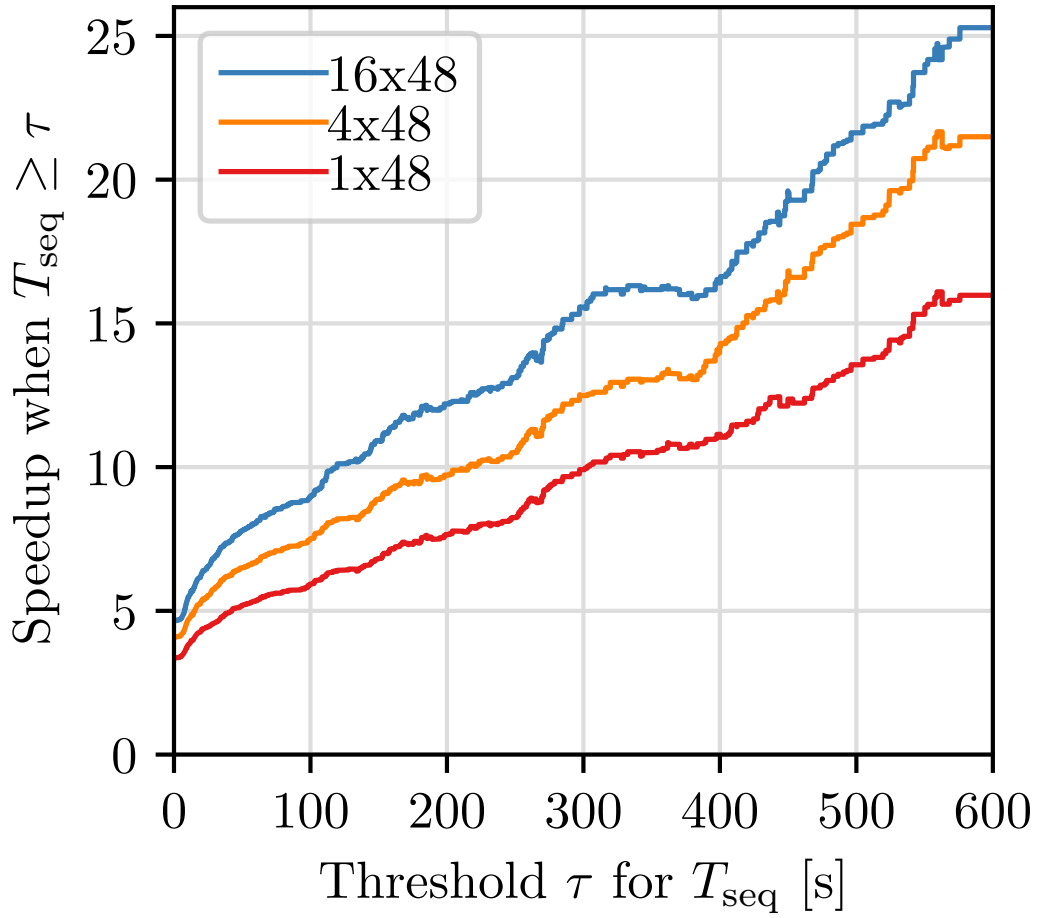
Bitwuzllob: Architecture



Each SAT adapter: Latency-hiding local SAT solver thread + distributed MallobSat task

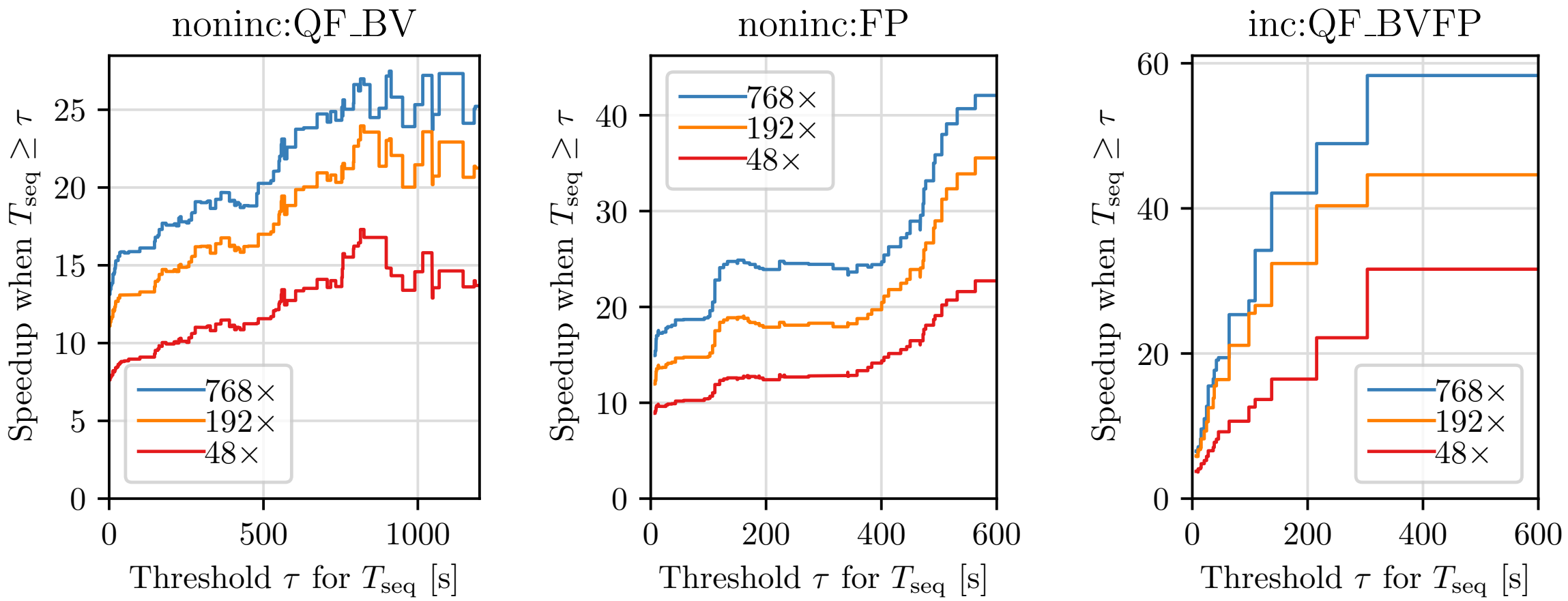
Bitwuzllob: Scaling





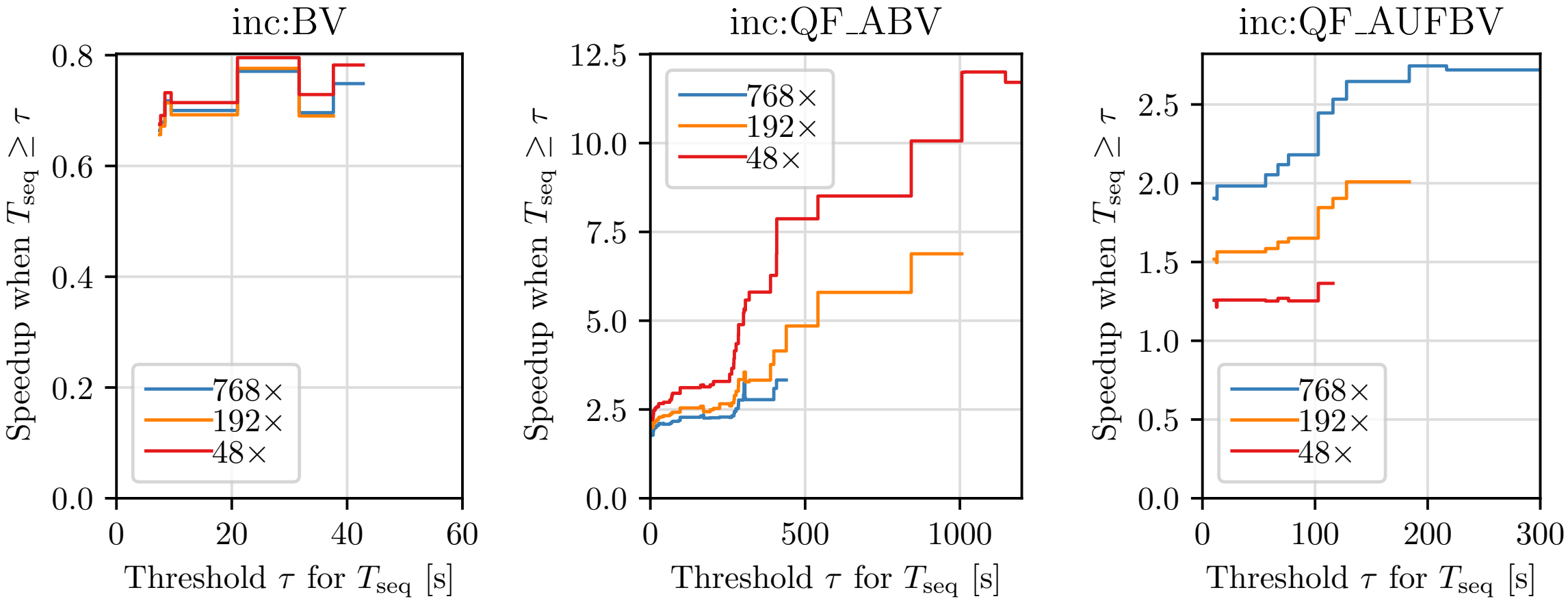
1386 inputs bucket-sampled from all SMT-LIB instances supported by Bitwuzla · Cutoff @ 5 inputs considered

Bitwuzllob: Scaling (The Good)



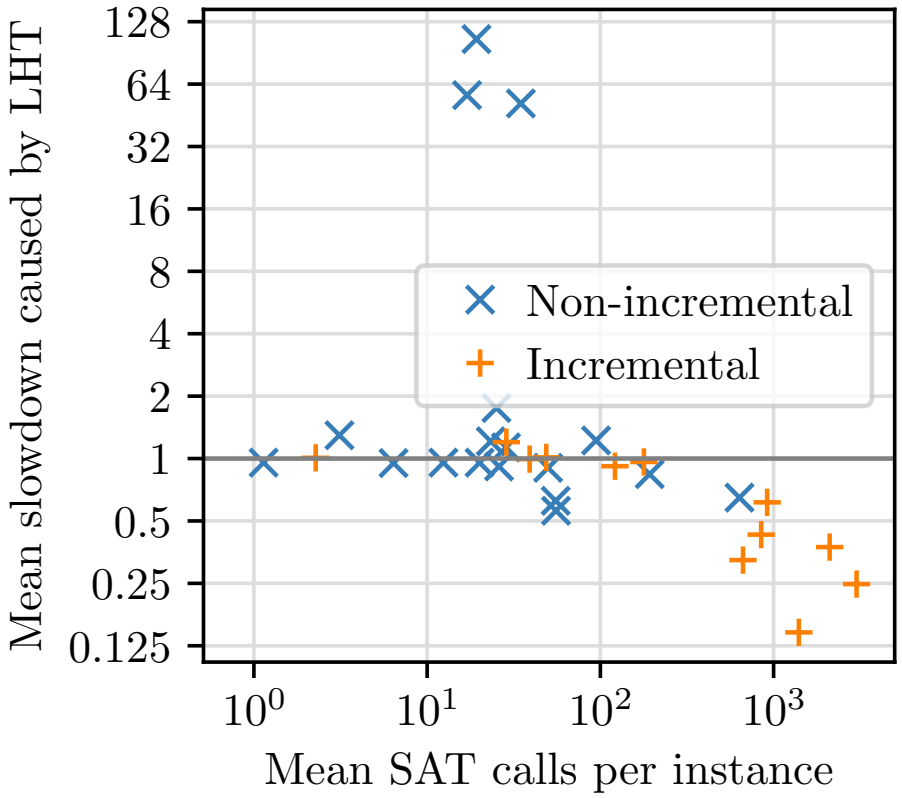
1386 inputs bucket-sampled from all SMT-LIB instances supported by Bitwuzla · Cutoff @ 5 inputs considered

Bitwuzllob: Scaling (The Bad & The Ugly)

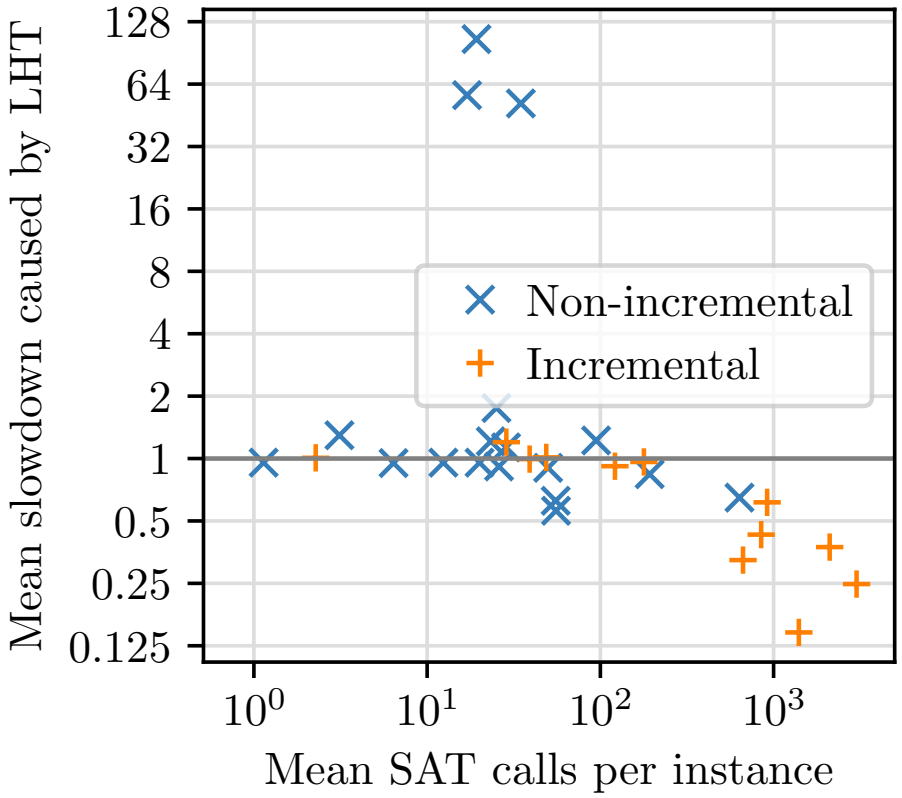


1386 inputs bucket-sampled from all SMT-LIB instances supported by Bitwuzla · Cutoff @ 5 inputs considered

Solver Configuration Matters

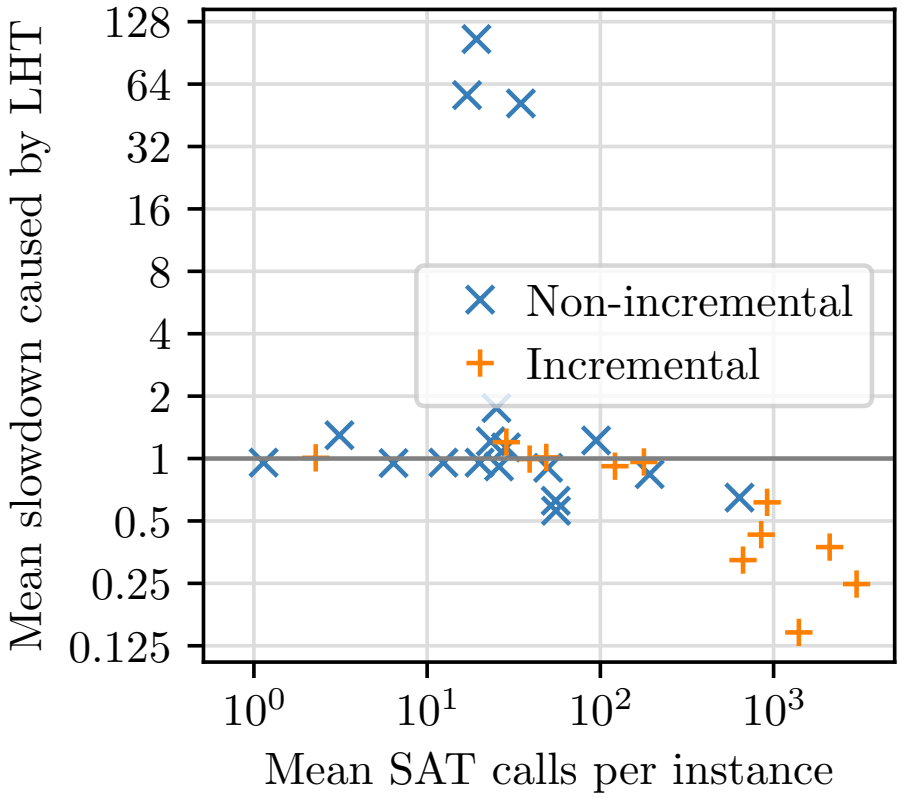


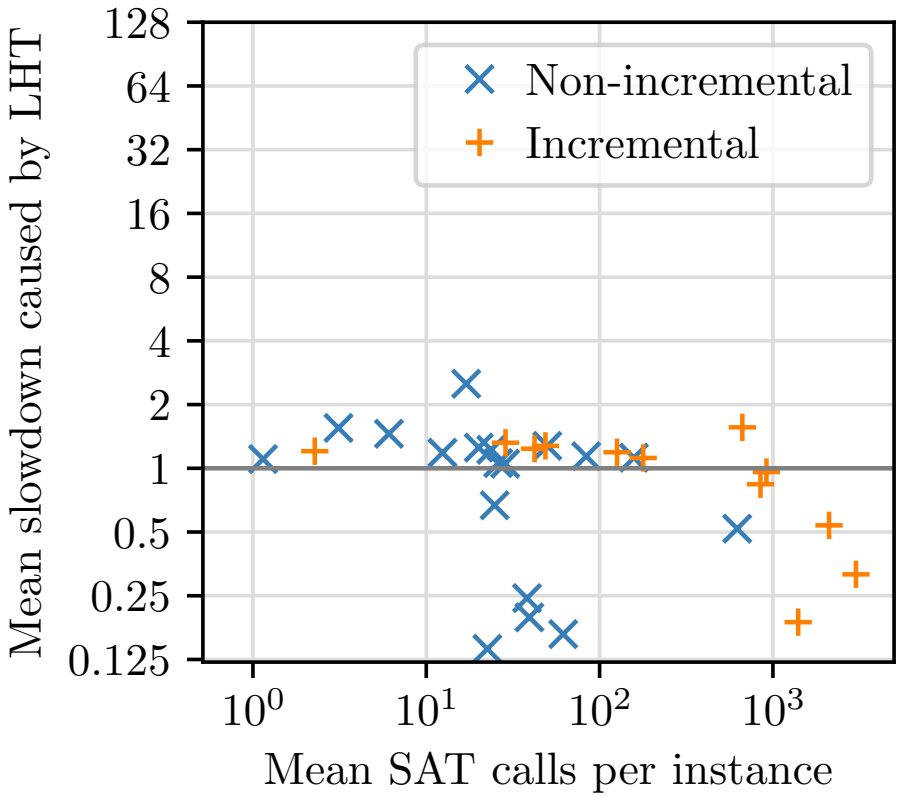
Solver Configuration Matters



Non-incremental theories BVFP, BVFPLRA, ABVFP:
Slowdowns of 50–100× through latency-hiding thread!
– Many SAT queries, expensive bit-blasting

Solver Configuration Matters





Non-incremental theories BVFP, BVFPLRA, ABVFP:
Slowdowns of 50–100× through latency-hiding thread!
– Many SAT queries, expensive bit-blasting

Remedy: Forcing initial variable phases to zero
(1st diversification in MallobSat!)
⇒ Refinement converges normally

Outline

1. Parallel Constraint Solving – generally and in SMT
2. The Mallob system
3. Experimental methodology and distributed SAT experiments
4. Distributed SMT solving with Bitwuzllob
5. **Outlook**

What's next?

Distributed "lemmas-on-demand" SMT via distributed SAT

- 1 SMT task $\rightsquigarrow$ several parallel SAT tasks
- Combine with partitioning strategies
- Integrate distributed SAT sweeping [FMCAD'26]

What's next?

Distributed "lemmas-on-demand" SMT via distributed SAT

- 1 SMT task $\rightsquigarrow$ several parallel SAT tasks
- Combine with partitioning strategies
- Integrate distributed SAT sweeping [FMCAD'26]

Distributed CDCL($\mathcal{T}$)

- Proper integration of IPASIR-UP into MallobSat
- Consequent, MallobSat-style clause sharing as central source of scalability
- Separate parallel/distributed algorithms for theory-specific subroutines

What's next?

Distributed "lemmas-on-demand" SMT via distributed SAT

- 1 SMT task $\rightsquigarrow$ several parallel SAT tasks
- Combine with partitioning strategies
- Integrate distributed SAT sweeping [FMCAD'26]

Distributed CDCL($\mathcal{T}$)

- Proper integration of IPASIR-UP into MallobSat
- Consequent, MallobSat-style clause sharing as central source of scalability
- Separate parallel/distributed algorithms for theory-specific subroutines

Mallob talk @ CAV on Monday, 11:20!

Joint work with:

Aina Niemetz
Mathias Preiner

Peter Sanders
Niccolò Rigi-Luperti
Ruben Götz

Armin Biere
Marijn Heule
Jeremias Berg
Christoph Jabs
Katalin Fazekas

and more

DFG Deutsche Forschungsgemeinschaft
German Research Foundation
Emmy Noether Project Scalable Automated Reasoning

mAllob
schedule · balance · solve · prove