Scalable and Provable Distributed SAT Solvers

Mike Whalen, Principal Applied Scientist, Automated Reasoning Group

Joint work with
Dawn Michaelson
Dominik Schreiber
Marijn Heule
Ben Kiesl-Reiter

Talk Takeaways

SAT solvers are general purpose **reasoning engines** that can be used for both **verification** and **optimization** purposes.

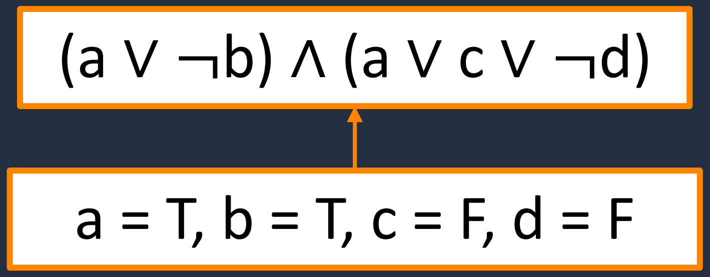
These solvers are the basis for **many of the automated reasoning tools** in use today.

Recent work in distributed **Clause Sharing Portfolio Solvers** has led to **dramatic performance improvements** over state-of-the-art sequential SAT solvers.

We have created a scalable approach for **proofs for distributed solvers** to ensure the correctness of solver results.

What is SAT?

Boolean satisfiability: Assign true/false to variables to make formula true



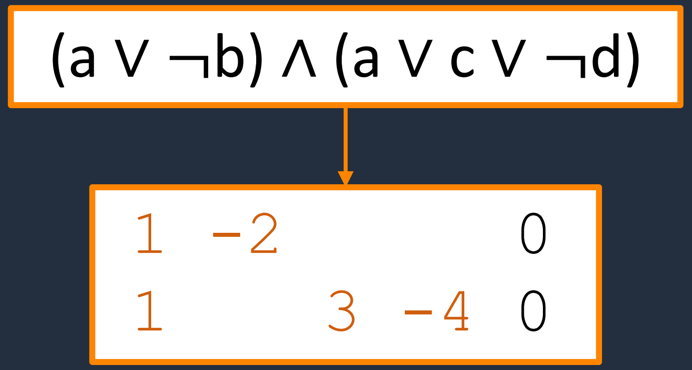
$(a \lor \neg b) \land (a \lor c \lor \neg d)$

$a = T, b = T, c = F, d = F$

Some formulas don’t have such an assignment

$\neg a \land (a \lor \neg b) \land (a \lor b)$

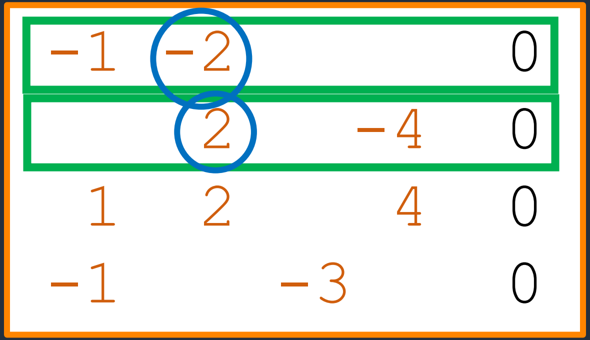
“standard” representation uses numbers for variables



$(a \lor \neg b) \land (a \lor c \lor \neg d)$

```
1 -2       0
1     3 -4 0
```

SAT Solvers



```
-1 -2       0
    2    -4 0
 1  2     4 0
-1    -3    0
```

Learn extra clauses to simplify search---prune search space

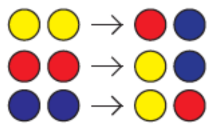
```
-1 -4 0
```

Problem is unsatisfiable if the empty clause is learned

```
0
```

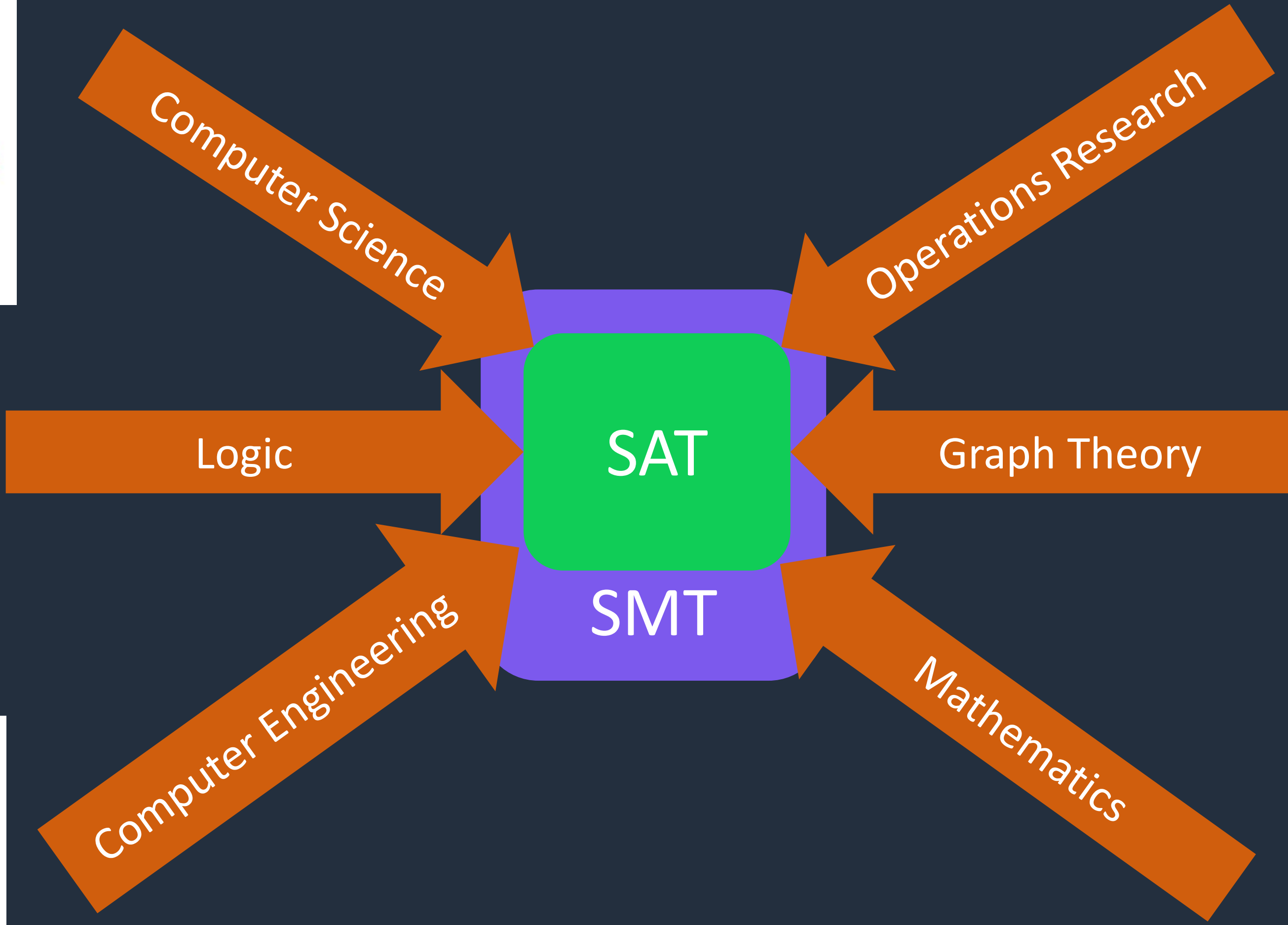
The Utility of SAT

exploit generation

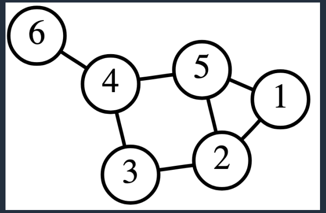


term rewriting termination

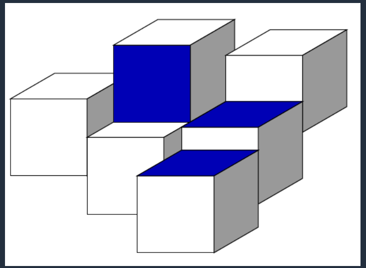
formal verification



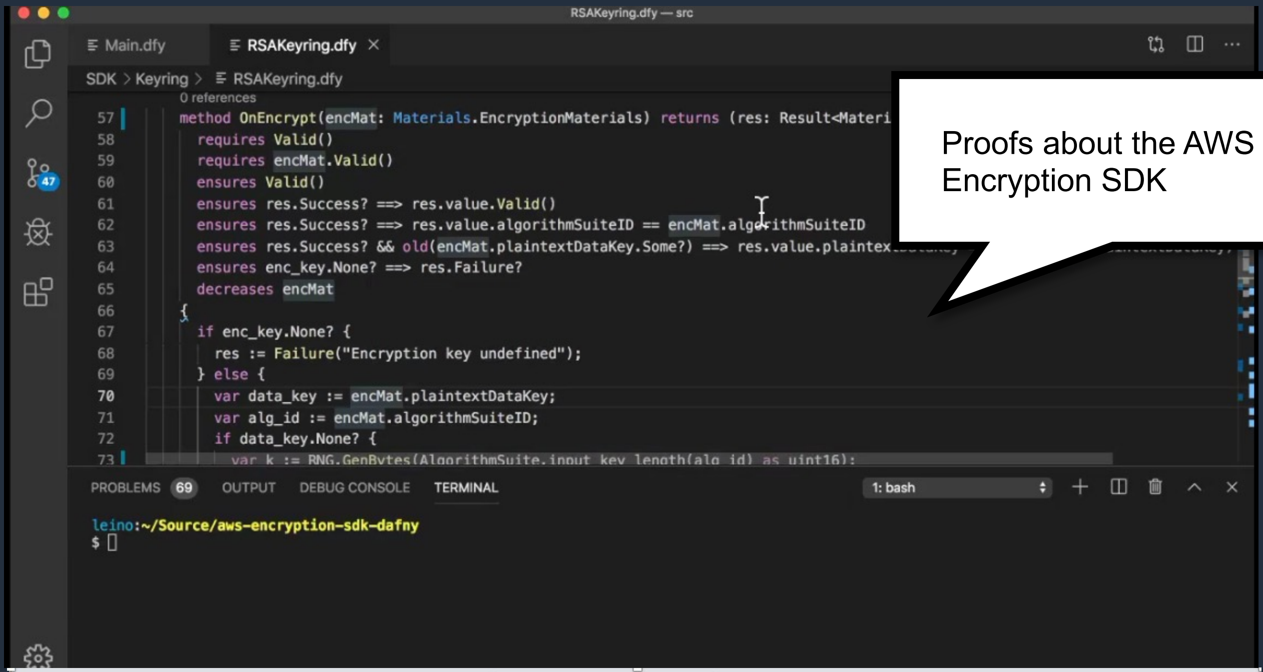
planning and scheduling



Reachability / cliques



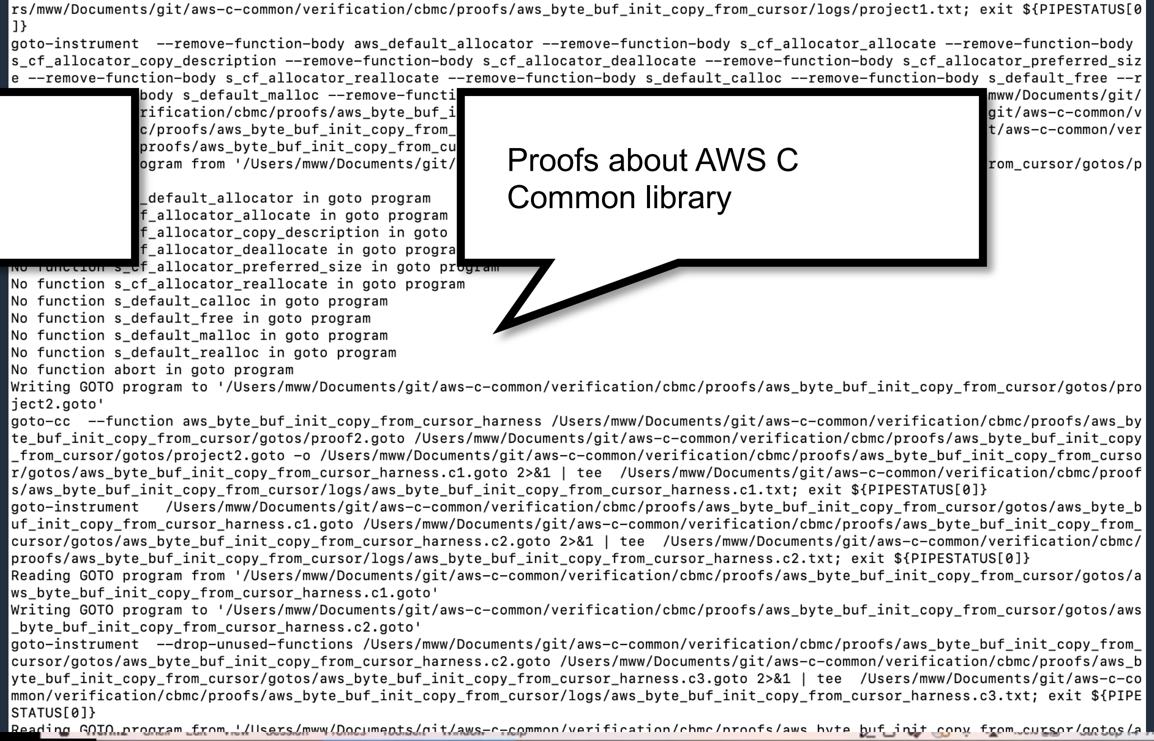
Keller's tiling conjecture



```
method OnEncrypt(encMat: Materials.EncryptionMaterials) returns (res: Result<Materi
  requires Valid()
  requires encMat.Valid()
  ensures Valid()
  ensures res.Success? ==> res.value.Valid()
  ensures res.Success? ==> res.value.algorithmSuiteID == encMat.algorithmSuiteID
  ensures res.Success? && old(encMat.plaintextDataKey.Some?) ==> res.value.plaintex
  ensures enc_key.None? ==> res.Failure?
  decreases encMat
{
  if enc_key.None? {
    res := Failure("Encryption key undefined");
  } else {
    var data_key := encMat.plaintextDataKey;
    var alg_id := encMat.algorithmSuiteID;
    if data_key.None? {
      var k := RNG.GenBytes(AlgorithmSuite.input_key_length(alg_id) as uint16);
```

```
leino:~/Source/aws-encryption-sdk-dafny
$
```

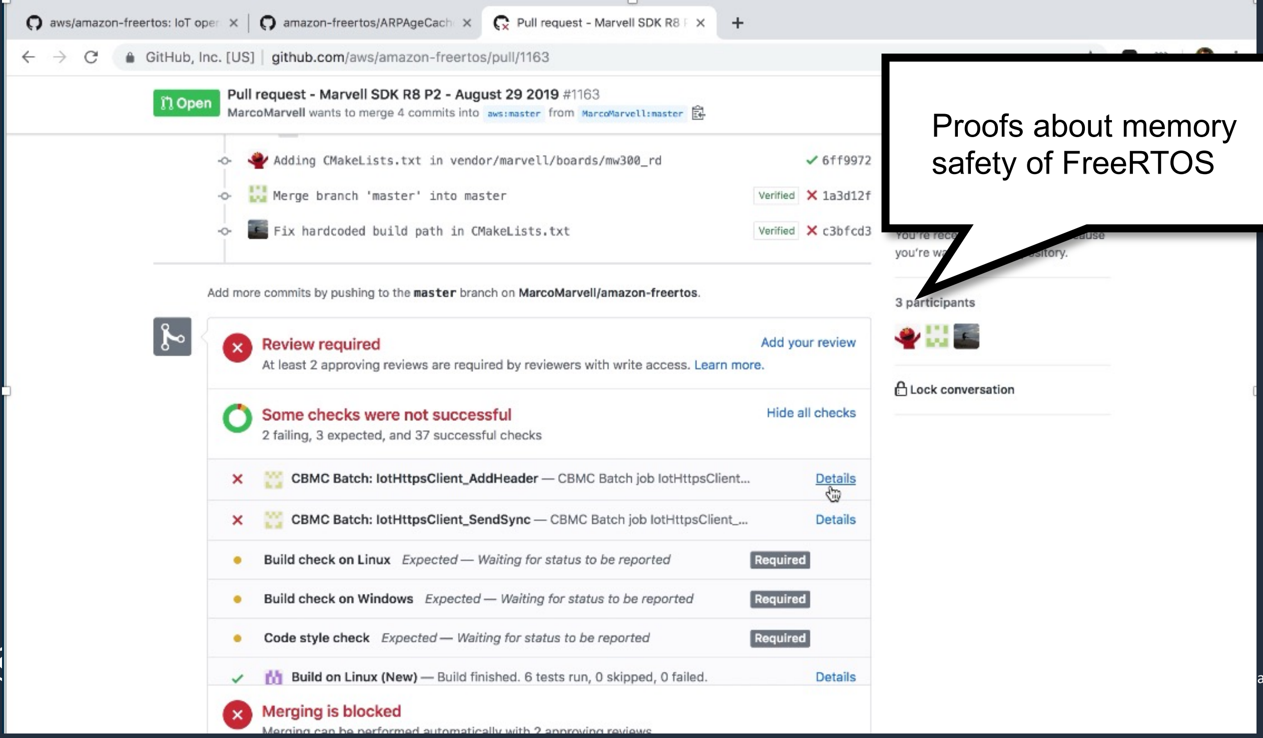
Proofs about the AWS Encryption SDK



```
rs/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/logs/project1.txt; exit ${PIPESTATUS[0
]}
goto-instrument  --remove-function-body aws_default_allocator --remove-function-body s_cf_allocator_allocate --remove-function-body
s_cf_allocator_copy_description --remove-function-body s_cf_allocator_deallocate --remove-function-body s_cf_allocator_preferred_siz
e --remove-function-body s_cf_allocator_reallocate --remove-function-body s_default_calloc --remove-function-body s_default_free --r
              body s_default_malloc --remove-functi                                              mww/Documents/git/
              rification/cbmc/proofs/aws_byte_buf_i                                              git/aws-c-common/v
              c/proofs/aws_byte_buf_init_copy_from_                                              t/aws-c-common/ver
              proofs/aws_byte_buf_init_copy_from_cu
              ogram from '/Users/mww/Documents/git/                                              rom_cursor/gotos/p

              _default_allocator in goto program
              f_allocator_allocate in goto program
              f_allocator_copy_description in goto
              f_allocator_deallocate in goto progra
No function s_cf_allocator_preferred_size in goto program
No function s_cf_allocator_reallocate in goto program
No function s_default_calloc in goto program
No function s_default_free in goto program
No function s_default_malloc in goto program
No function s_default_realloc in goto program
No function abort in goto program
Writing GOTO program to '/Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/gotos/pro
ject2.goto'
goto-cc  --function aws_byte_buf_init_copy_from_cursor_harness /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_by
te_buf_init_copy_from_cursor/gotos/proof2.goto /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy
_from_cursor/gotos/project2.goto -o /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_curso
r/gotos/aws_byte_buf_init_copy_from_cursor_harness.c1.goto 2>&1 | tee  /Users/mww/Documents/git/aws-c-common/verification/cbmc/proof
s/aws_byte_buf_init_copy_from_cursor/logs/aws_byte_buf_init_copy_from_cursor_harness.c1.txt; exit ${PIPESTATUS[0]}
goto-instrument   /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/gotos/aws_byte_b
uf_init_copy_from_cursor_harness.c1.goto /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_
cursor/gotos/aws_byte_buf_init_copy_from_cursor_harness.c2.goto 2>&1 | tee  /Users/mww/Documents/git/aws-c-common/verification/cbmc/
proofs/aws_byte_buf_init_copy_from_cursor/logs/aws_byte_buf_init_copy_from_cursor_harness.c2.txt; exit ${PIPESTATUS[0]}
Reading GOTO program from '/Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/gotos/a
ws_byte_buf_init_copy_from_cursor_harness.c1.goto'
Writing GOTO program to '/Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/gotos/aws
_byte_buf_init_copy_from_cursor_harness.c2.goto'
goto-instrument  --drop-unused-functions /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_
cursor/gotos/aws_byte_buf_init_copy_from_cursor_harness.c2.goto /Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_b
yte_buf_init_copy_from_cursor/gotos/aws_byte_buf_init_copy_from_cursor_harness.c3.goto 2>&1 | tee  /Users/mww/Documents/git/aws-c-co
mmon/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/logs/aws_byte_buf_init_copy_from_cursor_harness.c3.txt; exit ${PIPE
STATUS[0]}
Reading GOTO program from '/Users/mww/Documents/git/aws-c-common/verification/cbmc/proofs/aws_byte_buf_init_copy_from_cursor/gotos/a
```

Proofs about AWS C Common library


aws/amazon-freertos: IoT oper
amazon-freertos/ARPAgeCach
Pull request - Marvell SDK R8
GitHub, Inc. [US] | github.com/aws/amazon-freertos/pull/1163

Open **Pull request - Marvell SDK R8 P2 - August 29 2019** #1163
MarcoMarvell wants to merge 4 commits into aws:master from MarcoMarvell:master

Adding CMakeLists.txt in vendor/marvell/boards/mw300_rd		✓ 6ff9972
Merge branch 'master' into master	Verified	✗ 1a3d12f
Fix hardcoded build path in CMakeLists.txt	Verified	✗ c3bfcd3

Add more commits by pushing to the **master** branch on **MarcoMarvell/amazon-freertos**.

Review required — Add your review
At least 2 approving reviews are required by reviewers with write access. Learn more.

Some checks were not successful — Hide all checks
2 failing, 3 expected, and 37 successful checks

✗	**CBMC Batch: IotHttpsClient_AddHeader** — CBMC Batch job IotHttpsClient...		Details
✗	**CBMC Batch: IotHttpsClient_SendSync** — CBMC Batch job IotHttpsClient_...		Details
●	**Build check on Linux** *Expected — Waiting for status to be reported*	Required	
●	**Build check on Windows** *Expected — Waiting for status to be reported*	Required	
●	**Code style check** *Expected — Waiting for status to be reported*	Required	
✓	**Build on Linux (New)** — Build finished. 6 tests run, 0 skipped, 0 failed.		Details

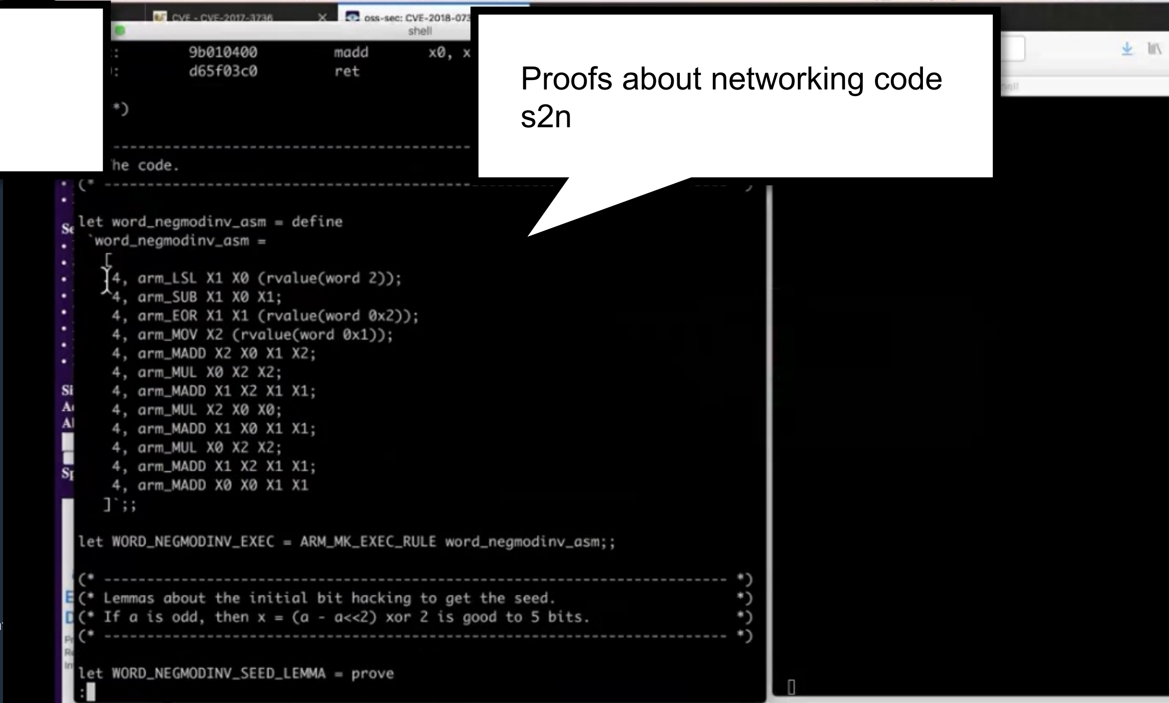
Merging is blocked
Merging can be performed automatically with 2 approving reviews.

You're rece... you're wa... ...sitory.

3 participants

Lock conversation

Proofs about memory safety of FreeRTOS



```
:       9b010400        madd      x0, x
:       d65f03c0        ret

 *)

 ---------------------------------------------
he code.
(* -------------------------------------------

let word_negmodinv_asm = define
 `word_negmodinv_asm =
  [
   4, arm_LSL X1 X0 (rvalue(word 2));
   4, arm_SUB X1 X0 X1;
   4, arm_EOR X1 X1 (rvalue(word 0x2));
   4, arm_MOV X2 (rvalue(word 0x1));
   4, arm_MADD X2 X0 X1 X2;
   4, arm_MUL X0 X2 X2;
   4, arm_MADD X1 X2 X1 X1;
   4, arm_MUL X2 X0 X0;
   4, arm_MADD X1 X0 X1 X1;
   4, arm_MUL X0 X2 X2;
   4, arm_MADD X1 X2 X1 X1;
   4, arm_MADD X0 X0 X1 X1
  ]`;;

let WORD_NEGMODINV_EXEC = ARM_MK_EXEC_RULE word_negmodinv_asm;;

(* ------------------------------------------------------------------------- *)
(* Lemmas about the initial bit hacking to get the seed.                     *)
(* If a is odd, then x = (a - a<<2) xor 2 is good to 5 bits.                 *)
(* ------------------------------------------------------------------------- *)

let WORD_NEGMODINV_SEED_LEMMA = prove
:
```

Proofs about networking code s2n

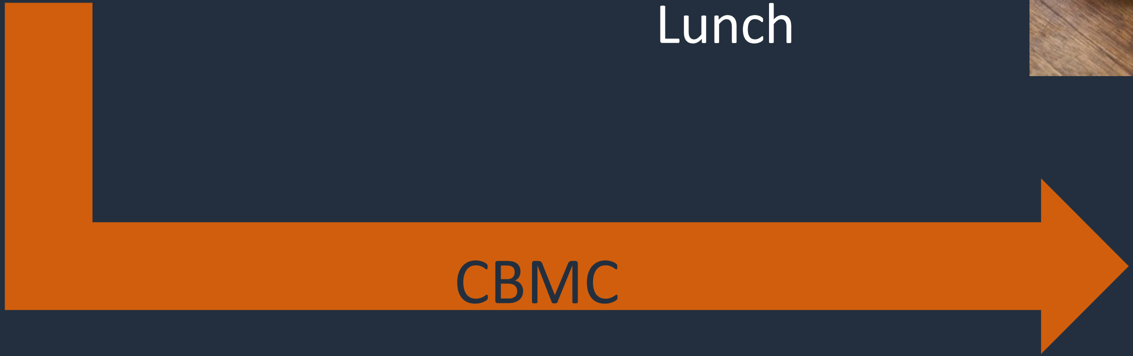
Scaling Proof Engines

Overnight

Lunch

Coffee

Squiggle



CBMC

Scaling Proof Engines

Overnight

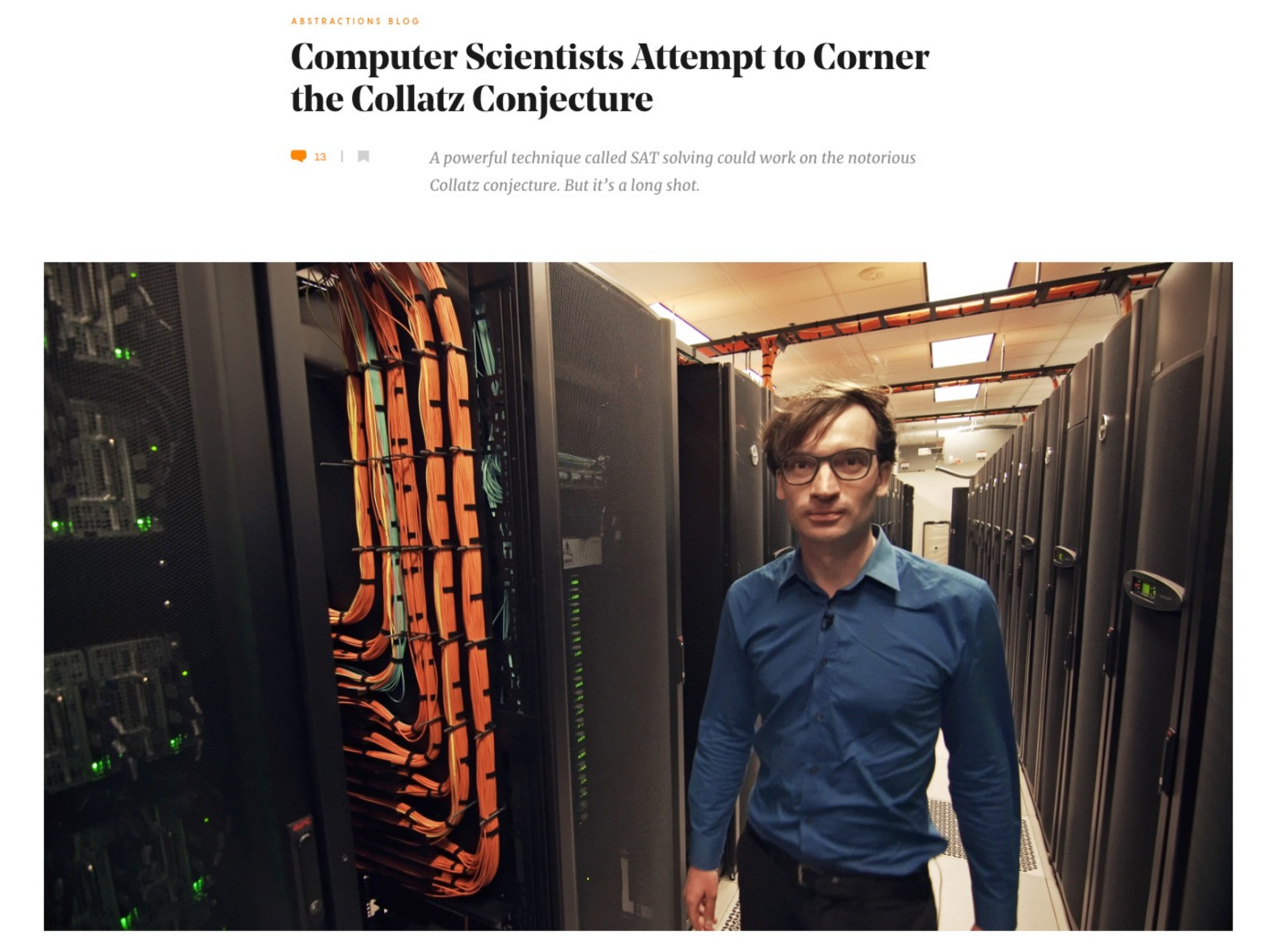
Lunch

Coffee

Squiggle

CBMC

Scaling Proof Engines



ABSTRACTIONS BLOG

Computer Scientists Attempt to Corner the Collatz Conjecture

13

A powerful technique called SAT solving could work on the notorious Collatz conjecture. But it's a long shot.

SAT Scalability

Mid 90s: **thousands** of clauses & variables

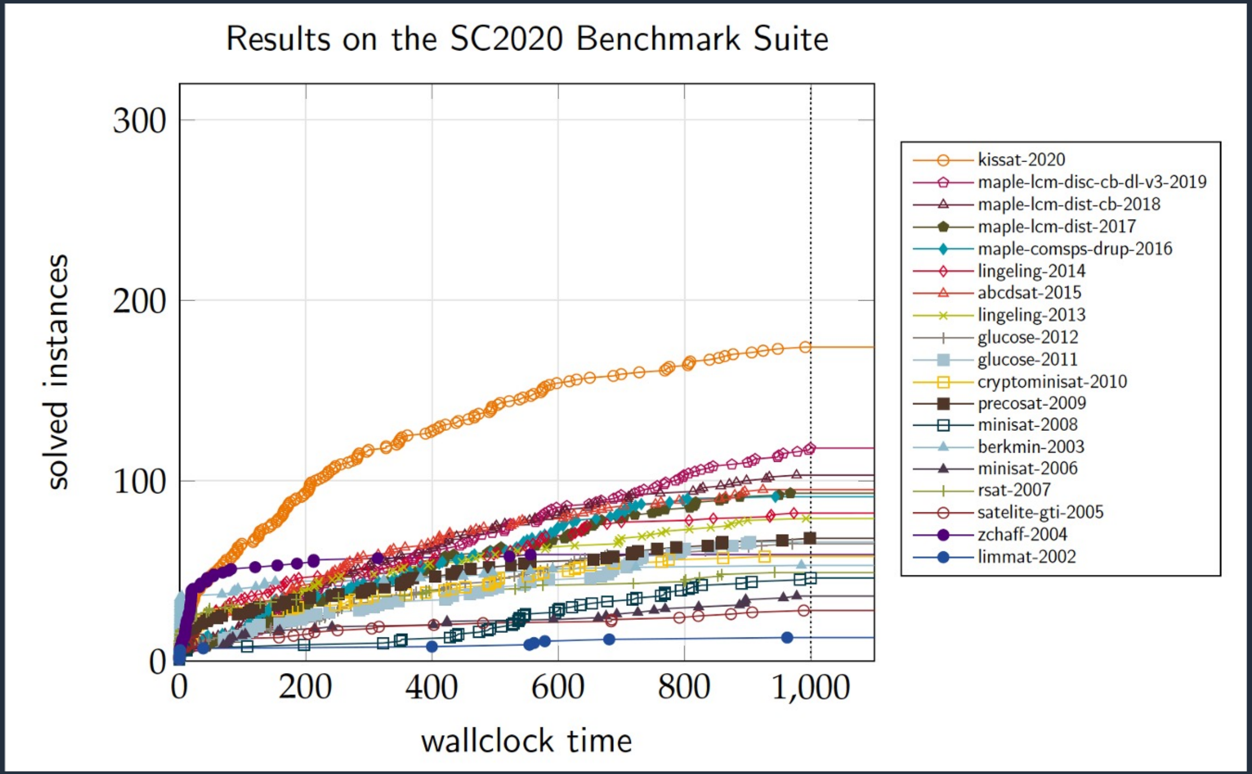
In 2020: **millions** of clauses & variables

SAT Scalability

Techniques for searching for answers improve over time

This graph describes the winning solver for each of the last 20 years, running on the same problems on the same hardware.

Benchmark suite contains 400 problems.

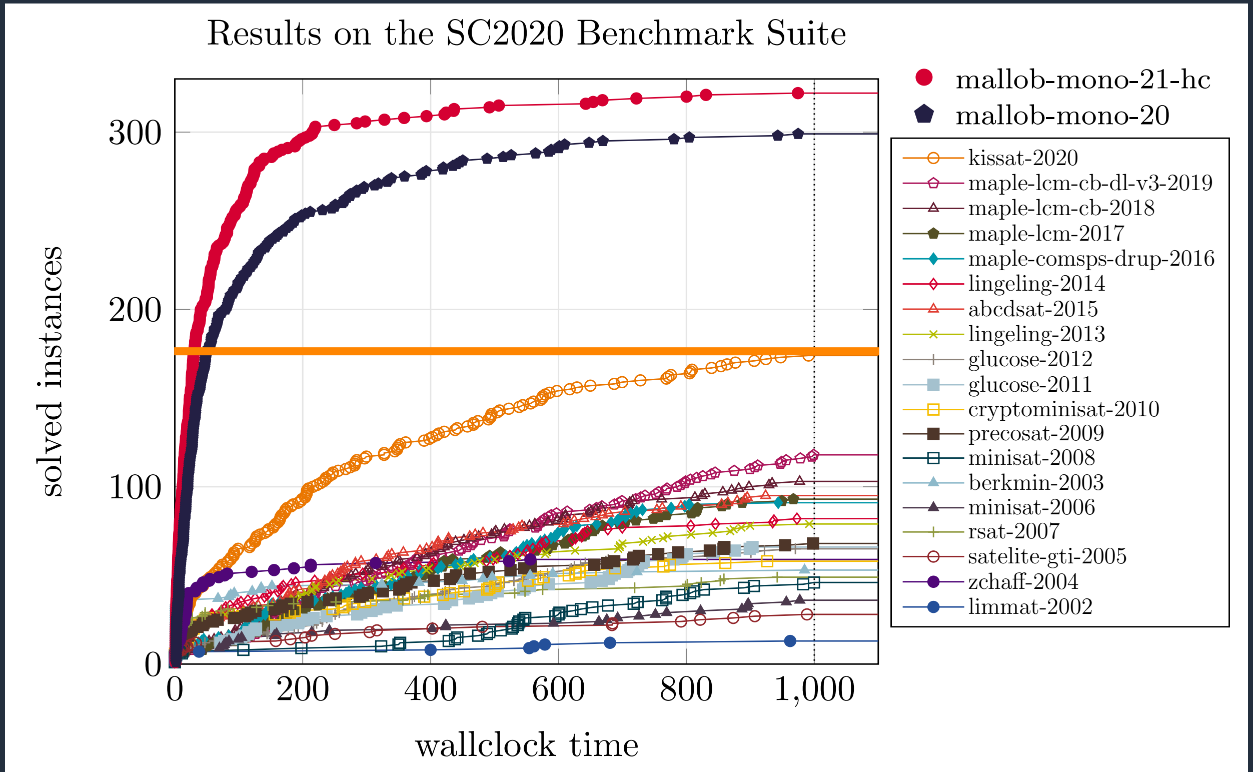


Distributed SAT Solvers

Distributed SAT solvers run multiple instances that work together

Still a new and improving technology

~30x faster than best sequential solver



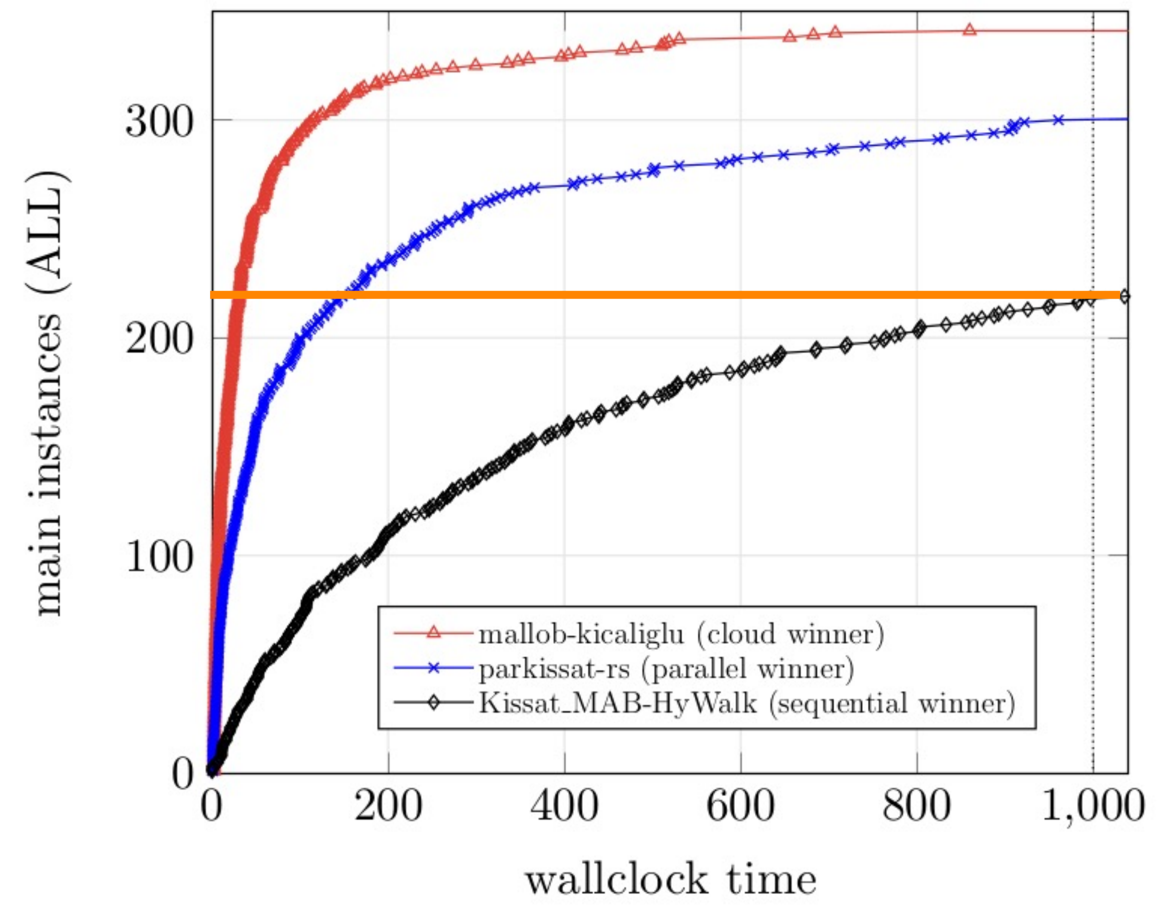
Distributed SAT Solvers

Distributed and parallel solvers improving more quickly than sequential solvers

2020-22: Distributed solver improvement: 56%

Sequential solver improvement: 18%

2022 SAT Competition



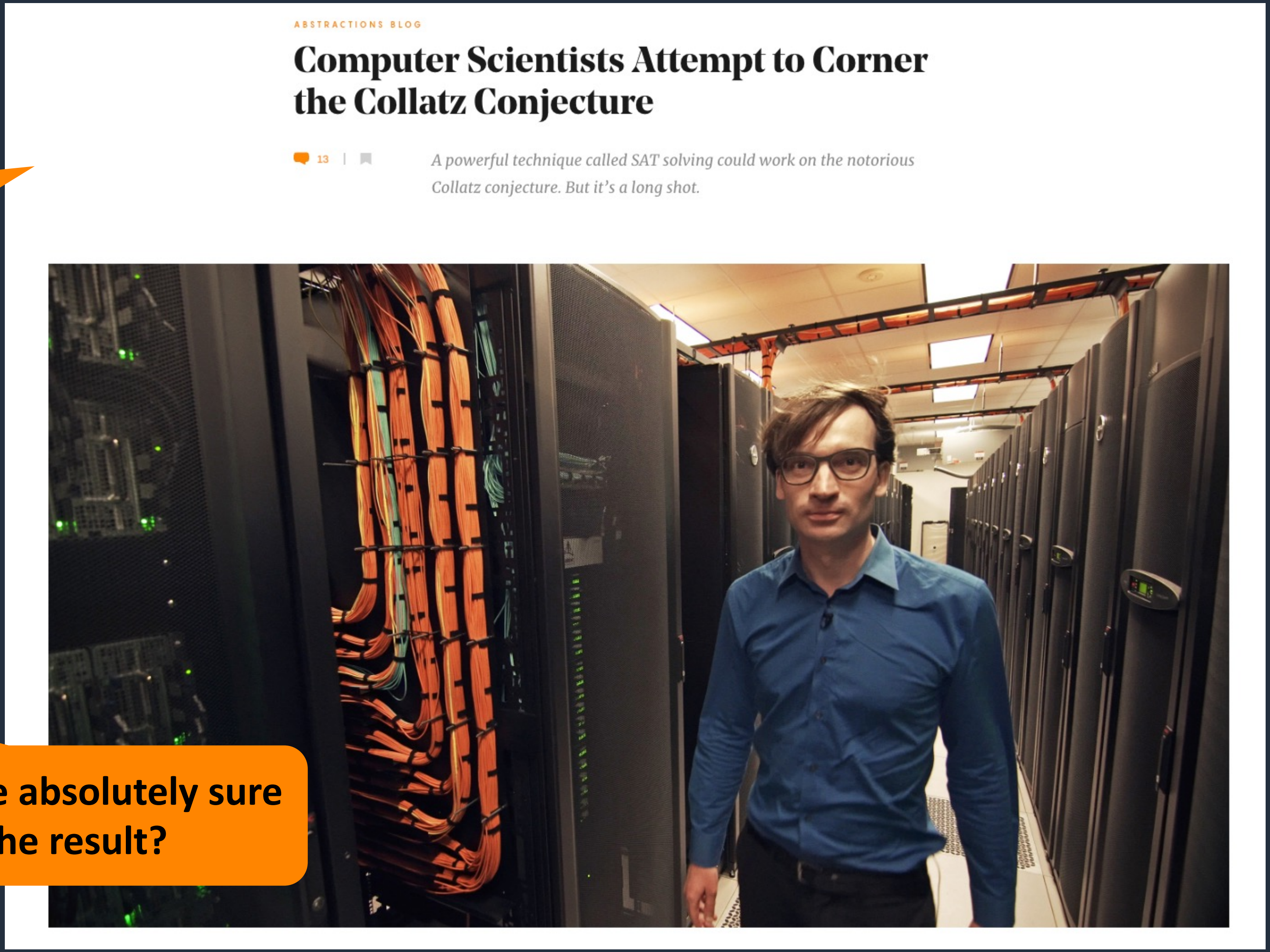
Using Distributed Solvers

Imagine a distributed SAT solver proves the Collatz Conjecture

This means the solver derives UNSAT for the negation of the conjecture

Fame! Glory!! Accolades!!!

But are we absolutely sure of the result?



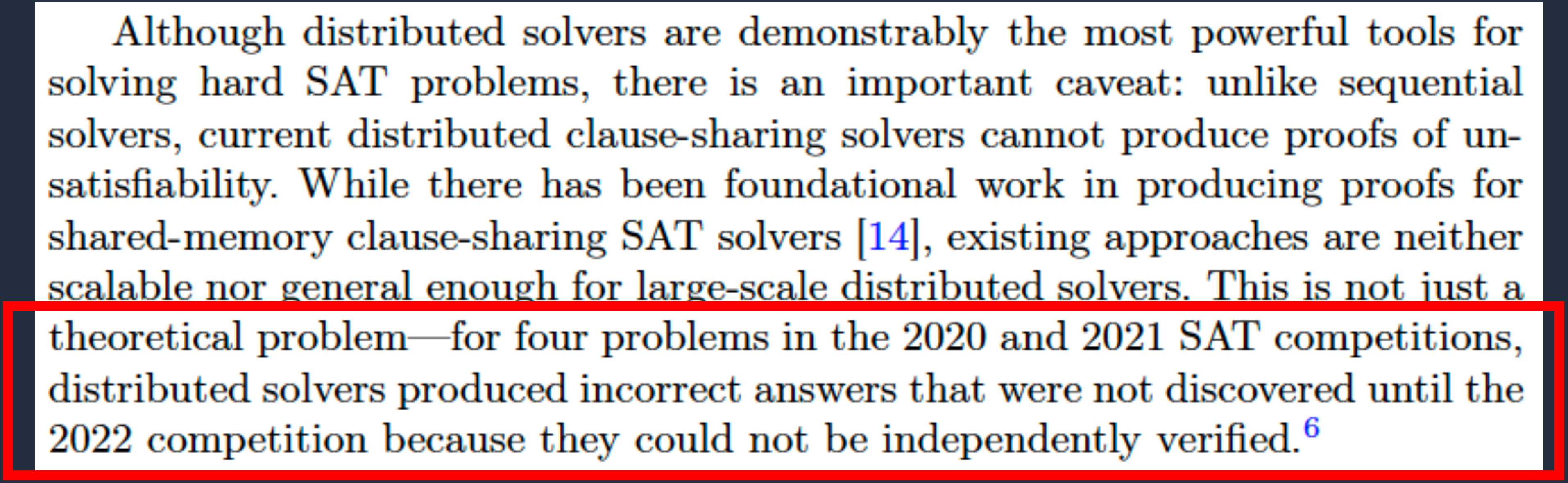
ABSTRACTIONS BLOG

Computer Scientists Attempt to Corner the Collatz Conjecture

13

A powerful technique called SAT solving could work on the notorious Collatz conjecture. But it's a long shot.

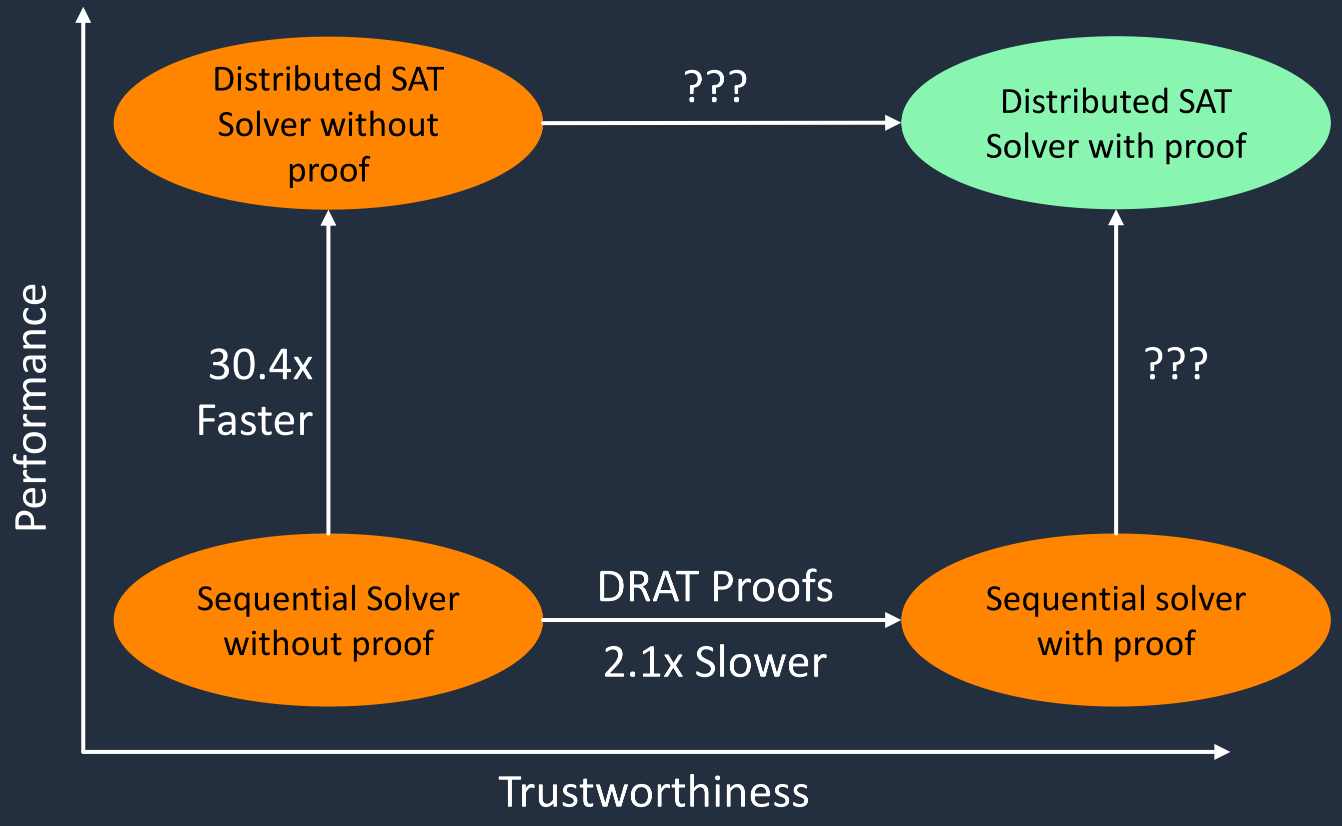
Trusting Results



Although distributed solvers are demonstrably the most powerful tools for solving hard SAT problems, there is an important caveat: unlike sequential solvers, current distributed clause-sharing solvers cannot produce proofs of unsatisfiability. While there has been foundational work in producing proofs for shared-memory clause-sharing SAT solvers [14], existing approaches are neither scalable nor general enough for large-scale distributed solvers. This is not just a theoretical problem—for four problems in the 2020 and 2021 SAT competitions, distributed solvers produced incorrect answers that were not discovered until the 2022 competition because they could not be independently verified.[6]

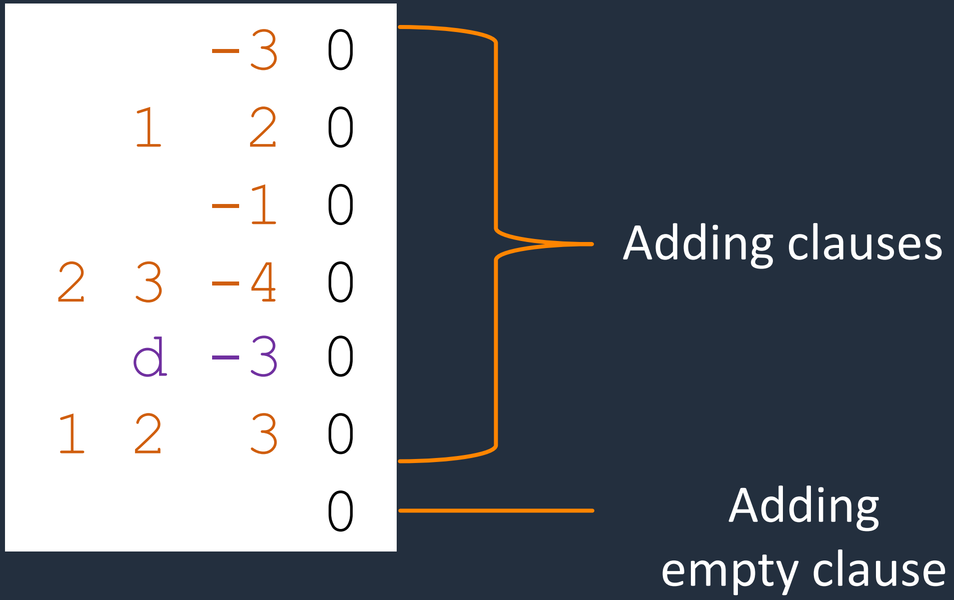
Unsatisfiability Proofs for Distributed Clause Sharing SAT Solvers, TACAS 2023

Solver Quadrants


Distributed SAT Solver without proof
???
Distributed SAT Solver with proof
30.4x Faster
???
Performance
Sequential Solver without proof
DRAT Proofs
2.1x Slower
Sequential solver with proof
Trustworthiness

Sequential Proof Format

Standard format: DRAT



```
      -3 0
    1  2 0
      -1 0
 2  3 -4 0
    d -3 0
 1  2  3 0
         0
```

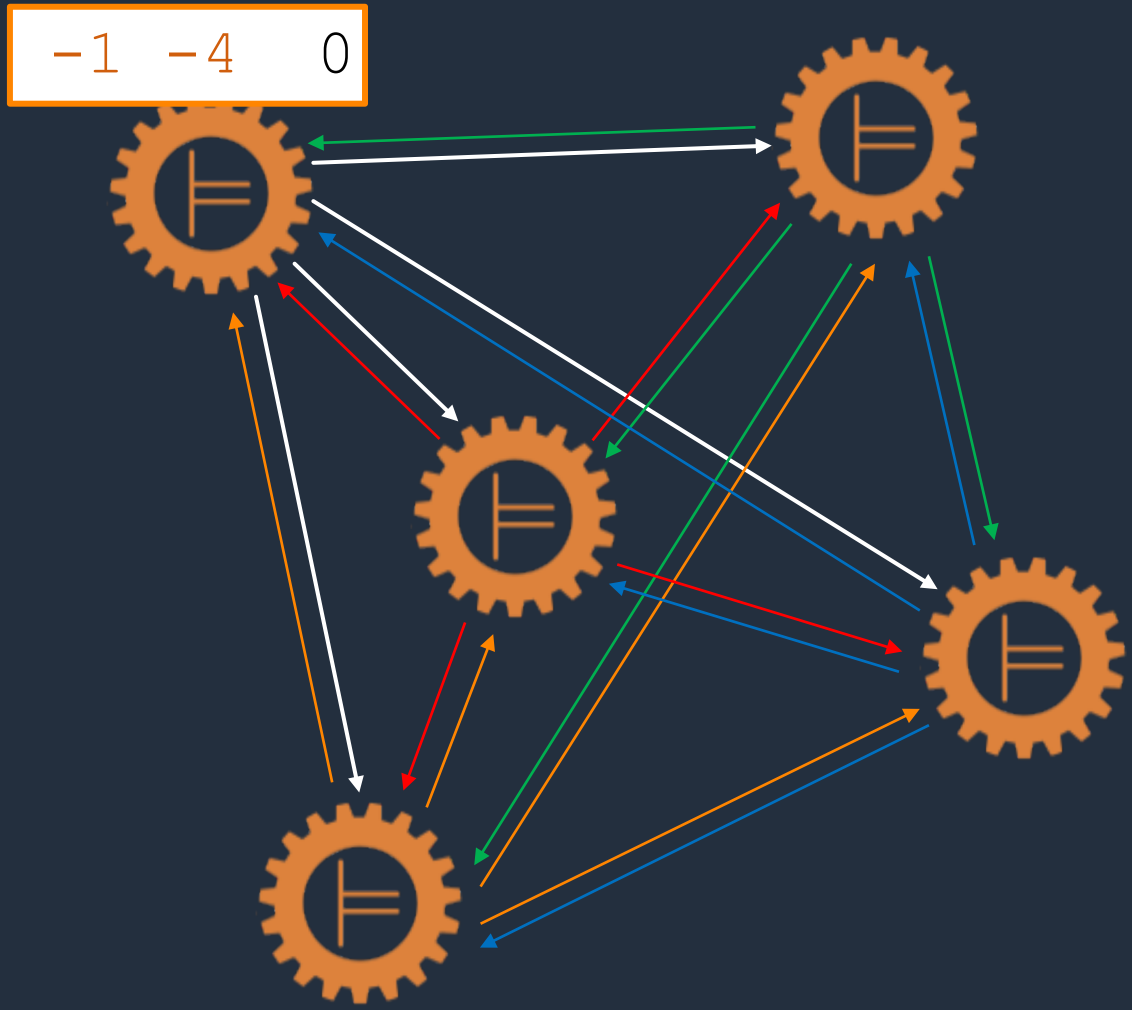
Original Problem

```
 1 -2    0
 2 -4    0
 1  2  4 0
-1 -3    0
 1 -3    0
-1  3    0
 1  3 -4 0
 1  3  4 0
```

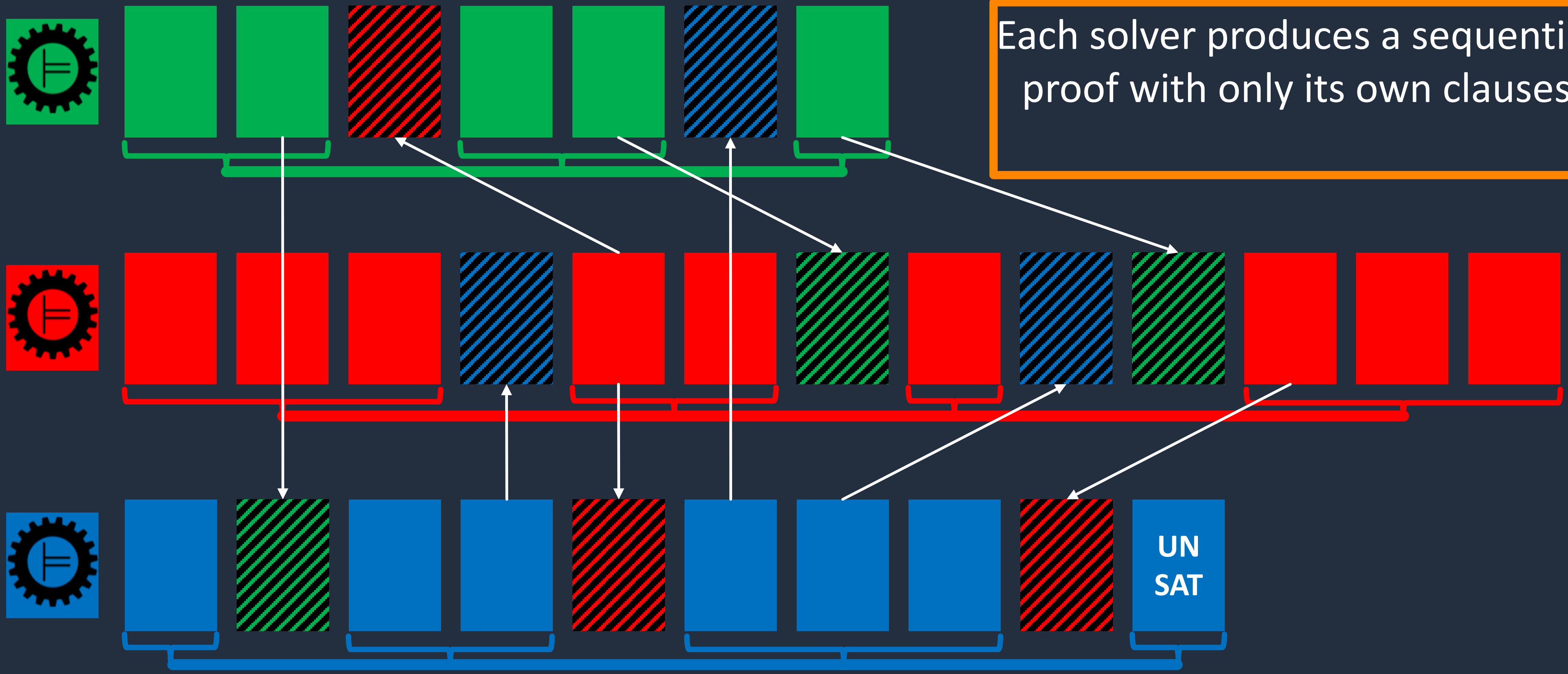
Distributed Solvers

Multiple solver instances running in parallel on same problem

Share learned clauses



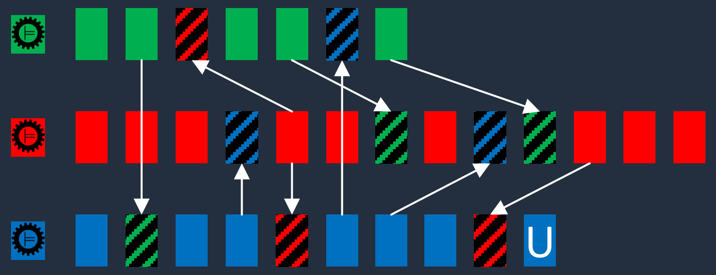
Distributed Solver Proof Challenges



Each solver produces a sequential proof with only its own clauses

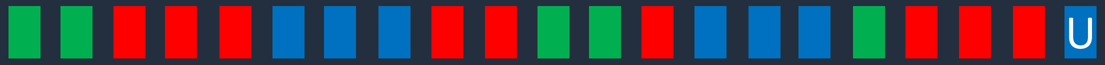
Combining Proofs

- Need dependency order of clauses between solvers
- Need to order clauses from different proofs relative to each other



Combining Proofs

- Need dependency order of clauses between solvers
- Need to order clauses from different proofs relative to each other



Technical Challenges

- Provide metadata to identify which solver produced each learned clause
- Efficiently sort learned clauses in dependency order across all solvers
- Reduce proof size by removing unnecessary clauses, so that proof can be efficiently checked.

Sequential Proof Format

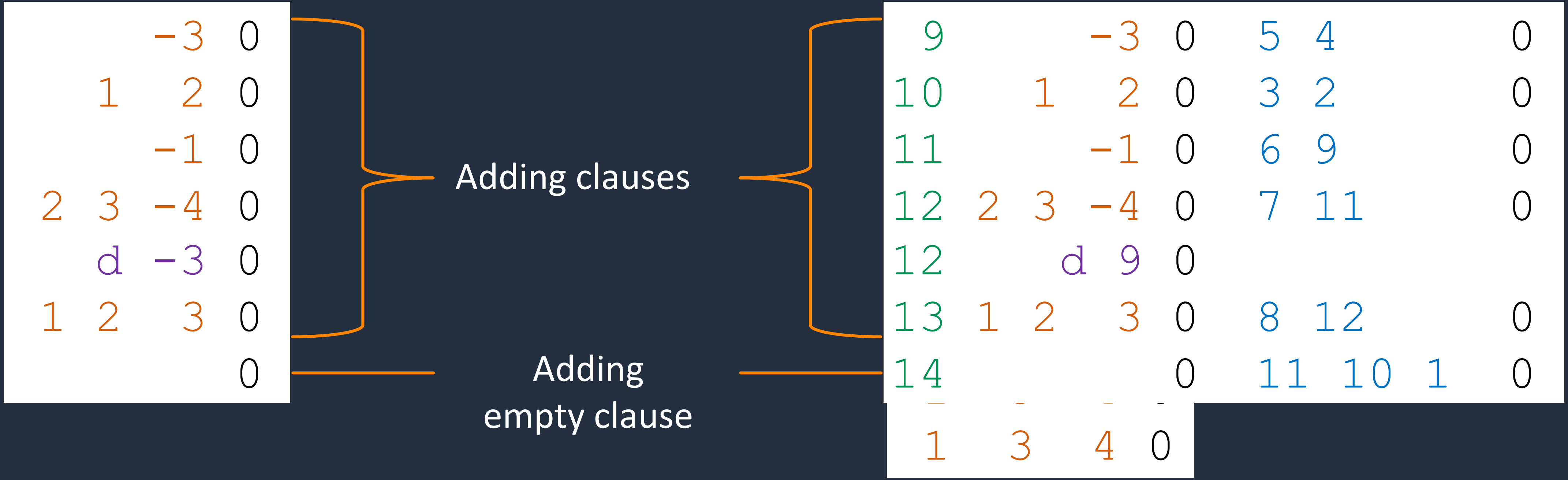
```
1
2
3
4
```

```
5
6
7
8
```

Standard format: DRAT

Original Problem:

Another format: LRAT



```
      -3  0
   1   2  0
      -1  0
2  3  -4  0
   d  -3  0
1  2   3  0
          0
```

Adding clauses

Adding empty clause

```
 9        -3  0   5  4         0
10     1   2  0   3  2         0
11        -1  0   6  9         0
12  2  3  -4  0   7 11         0
12       d 9  0
13  1  2   3  0   8 12         0
14            0  11 10  1      0
```

```
 1     3     4  0
```

Distributed Clause ID

- Each clause needs unique ID across all instances
- Needs to be unique without communication
- Formula:

$$o + i + t * k$$

where

- o: original clauses
- i: current instance num
- t: total instances
- k: number of clauses learned in current instance

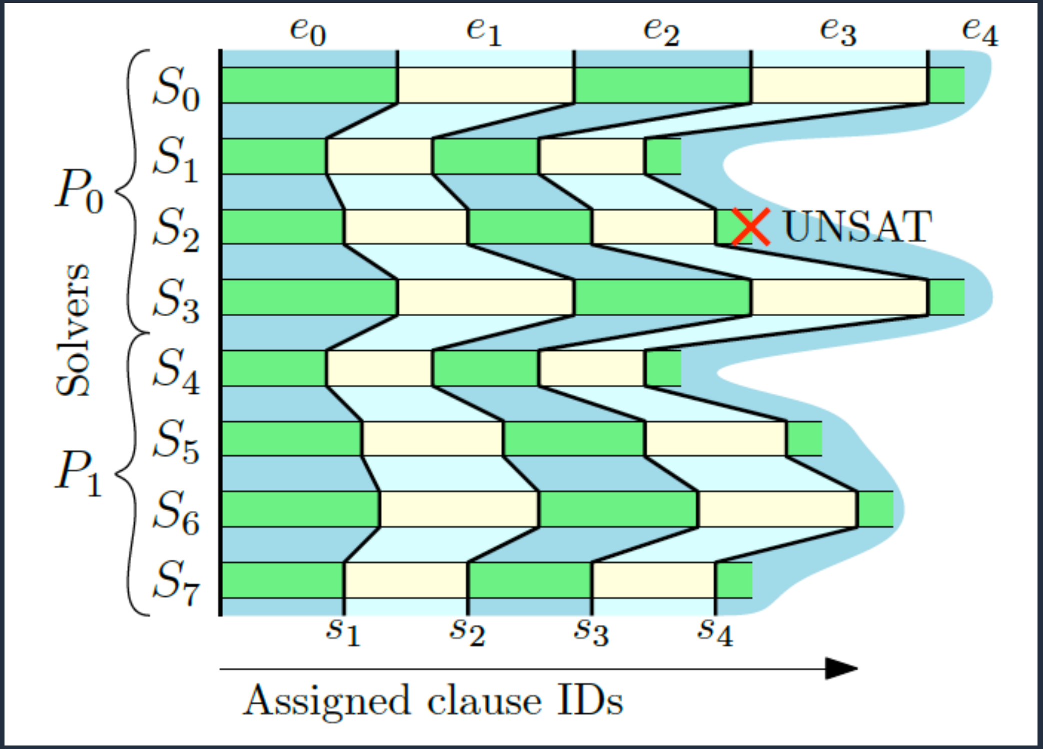
3 Instances
19 Original Clauses

1	2	3
20	21	22
23	24	25
26	27	28

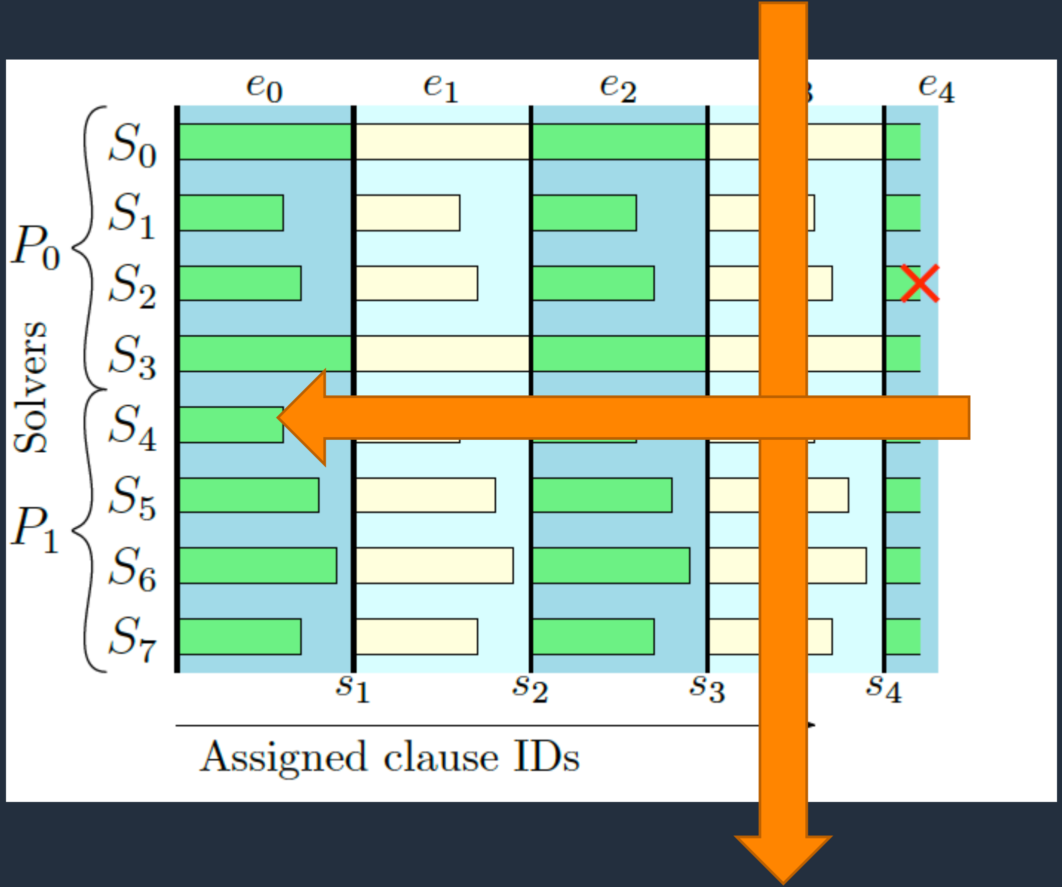
Technical Challenges

- Provide metadata to identify which solver produced each learned clause
  - Cross-Solver Unique Clause IDs used by LRAT format.
- Efficiently sort learned clauses in dependency order across all solvers
- Reduce proof size by removing unnecessary clauses, so that proof can be efficiently checked.

Using Epochs to Structure Proof Reconstruction

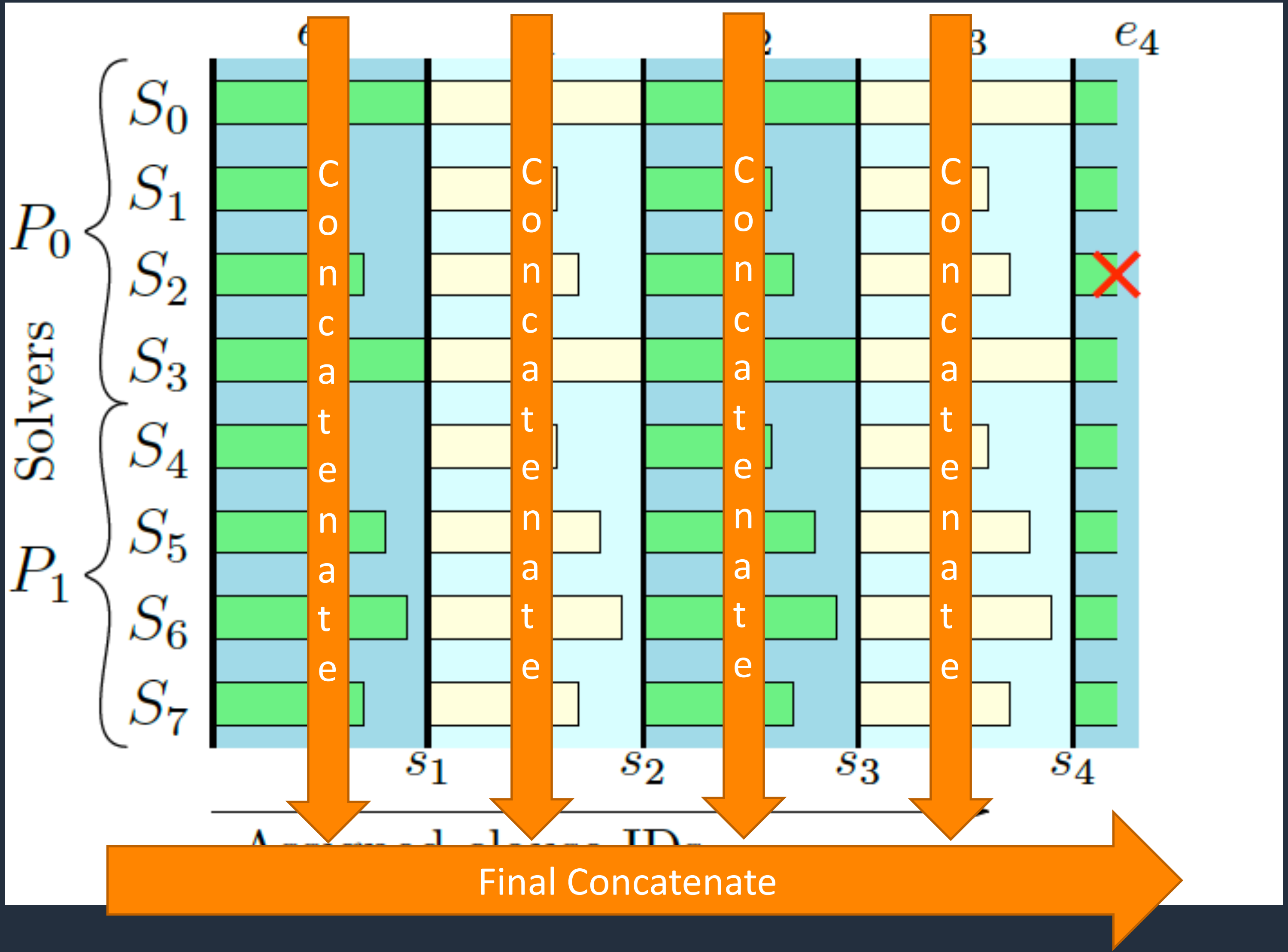


No dependencies



Later phases do not impact earlier phases

Naïve Approach



Because of epochs, could just merge all proofs together.

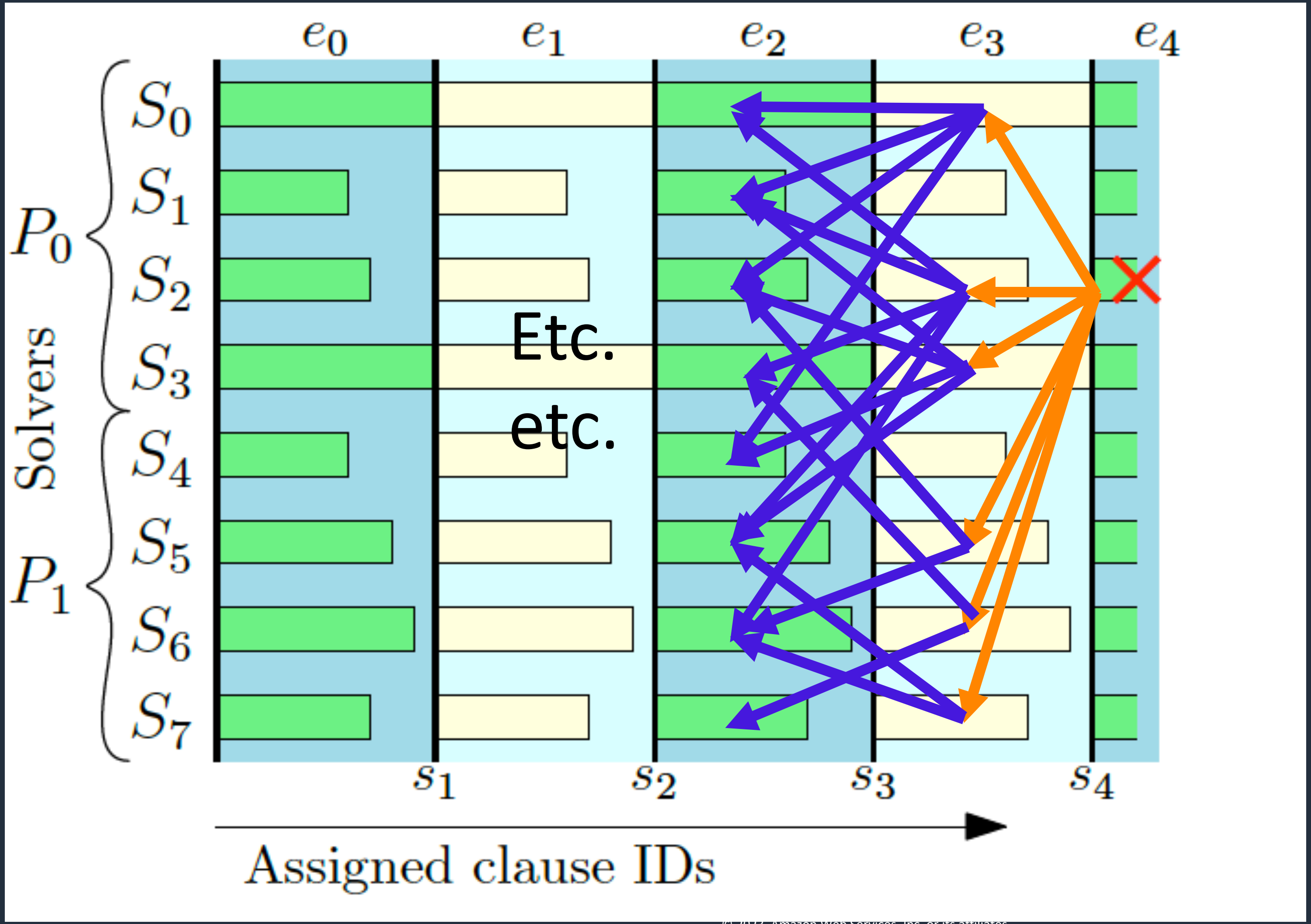
However, proofs are HUGE

Mean proof size: 65 GB

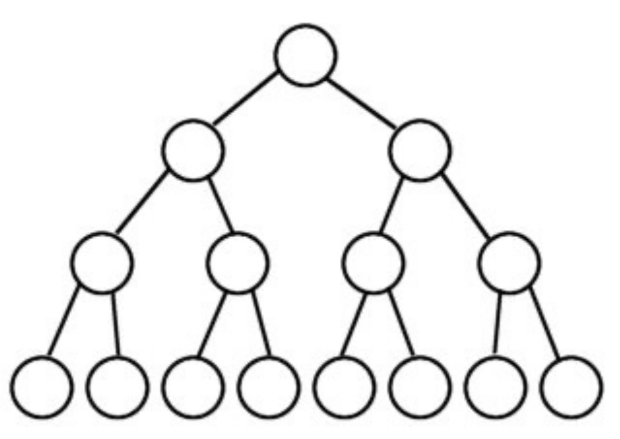
Largest proof size: >1 TB

Unreasonably large to check.

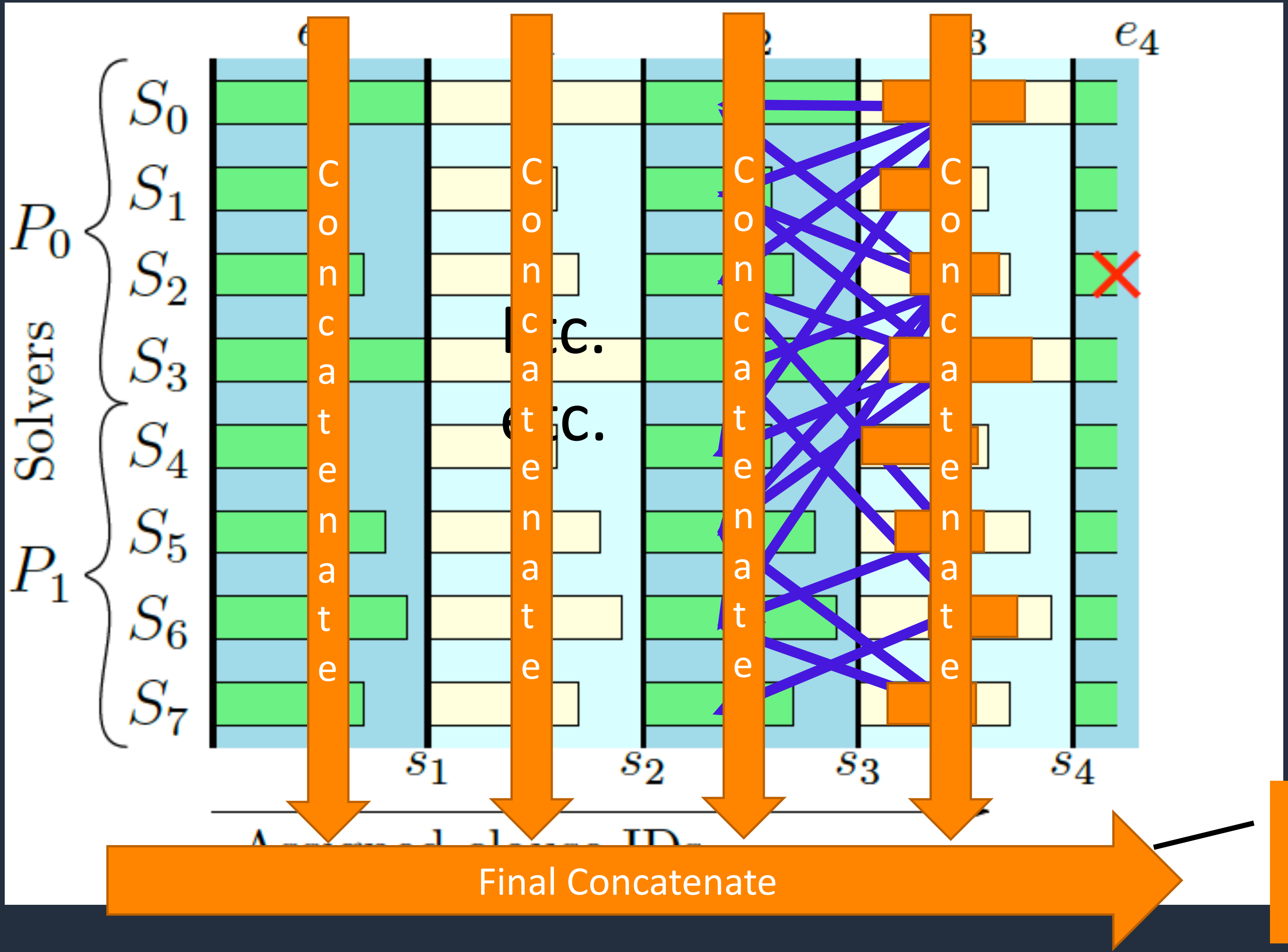
Using Epochs to Structure Proof Reconstruction



Removing duplicate requests for the same clause from multiple nodes: use same mechanism that Mallob uses for clause sharing.



Using Epochs to Structure Proof Reconstruction



Concatenate

Concatenate

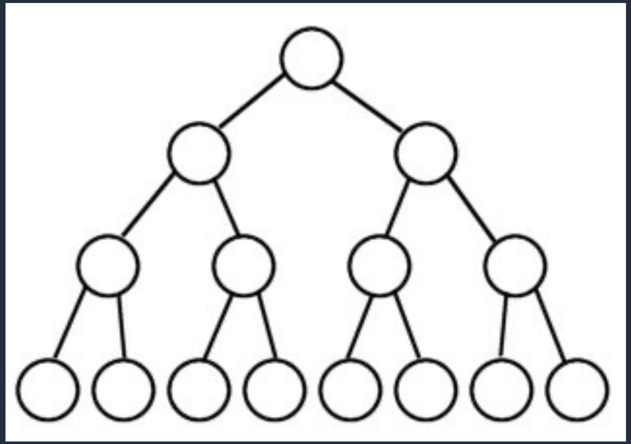
Concatenate

Concatenate

Etc. etc.

Final Concatenate

Removing duplicate requests for the same clause from multiple nodes: use same mechanism that Mallob uses for clause sharing.

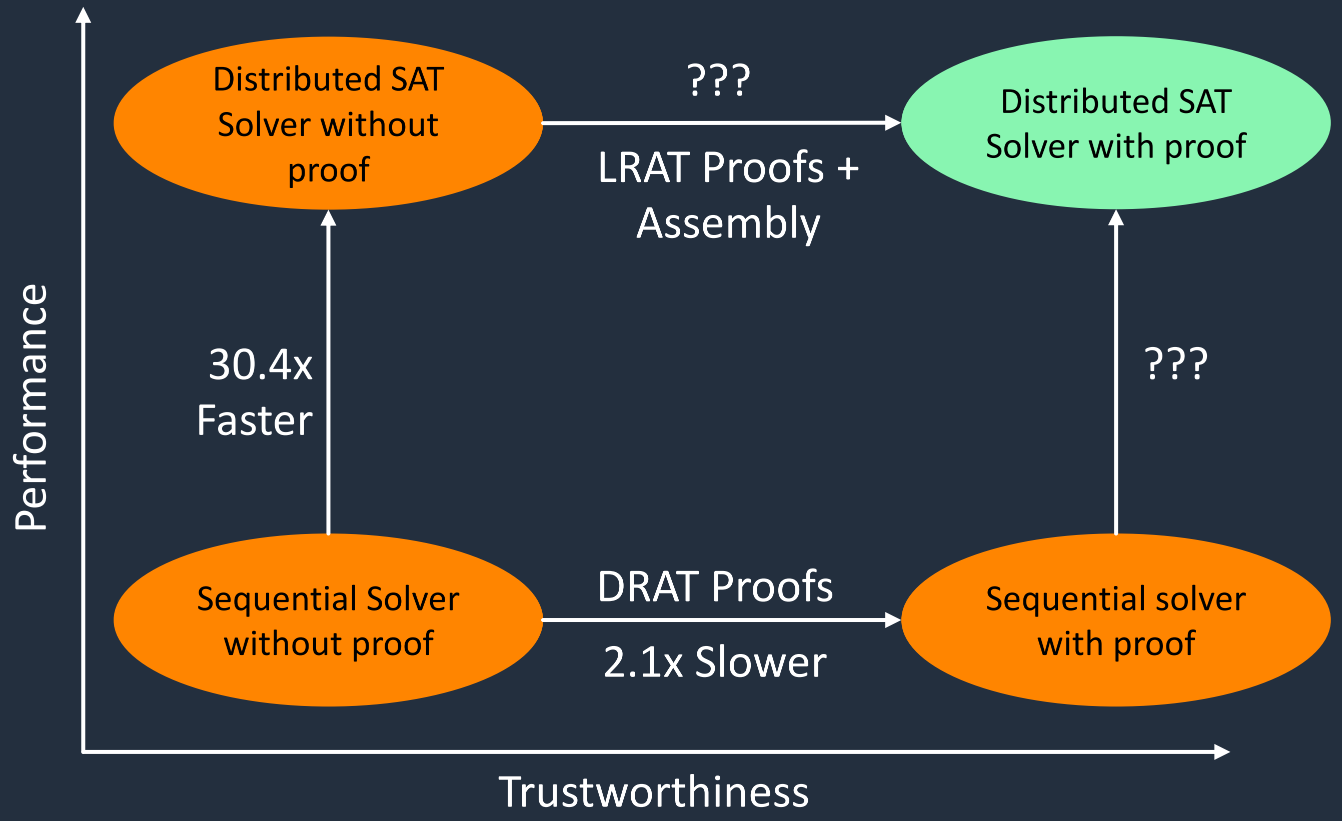


Mean proof size: 5.5 GB.
8% of Naïve approach

Technical Challenges

- Provide metadata to identify which solver produced each learned clause
  - Cross-solver unique clause IDs used by LRAT format.
- Efficiently sort learned clauses in dependency order across all solvers
  - Use the same techniques used to share clauses to share proof dependencies
- Reduce proof size by removing unnecessary clauses, so that proof can be efficiently checked.
  - Work backwards from the empty clause to establish results

Solver Quadrants


Distributed SAT Solver without proof
???
LRAT Proofs + Assembly
Distributed SAT Solver with proof
Performance
30.4x Faster
???
Sequential Solver without proof
DRAT Proofs
2.1x Slower
Sequential solver with proof
Trustworthiness

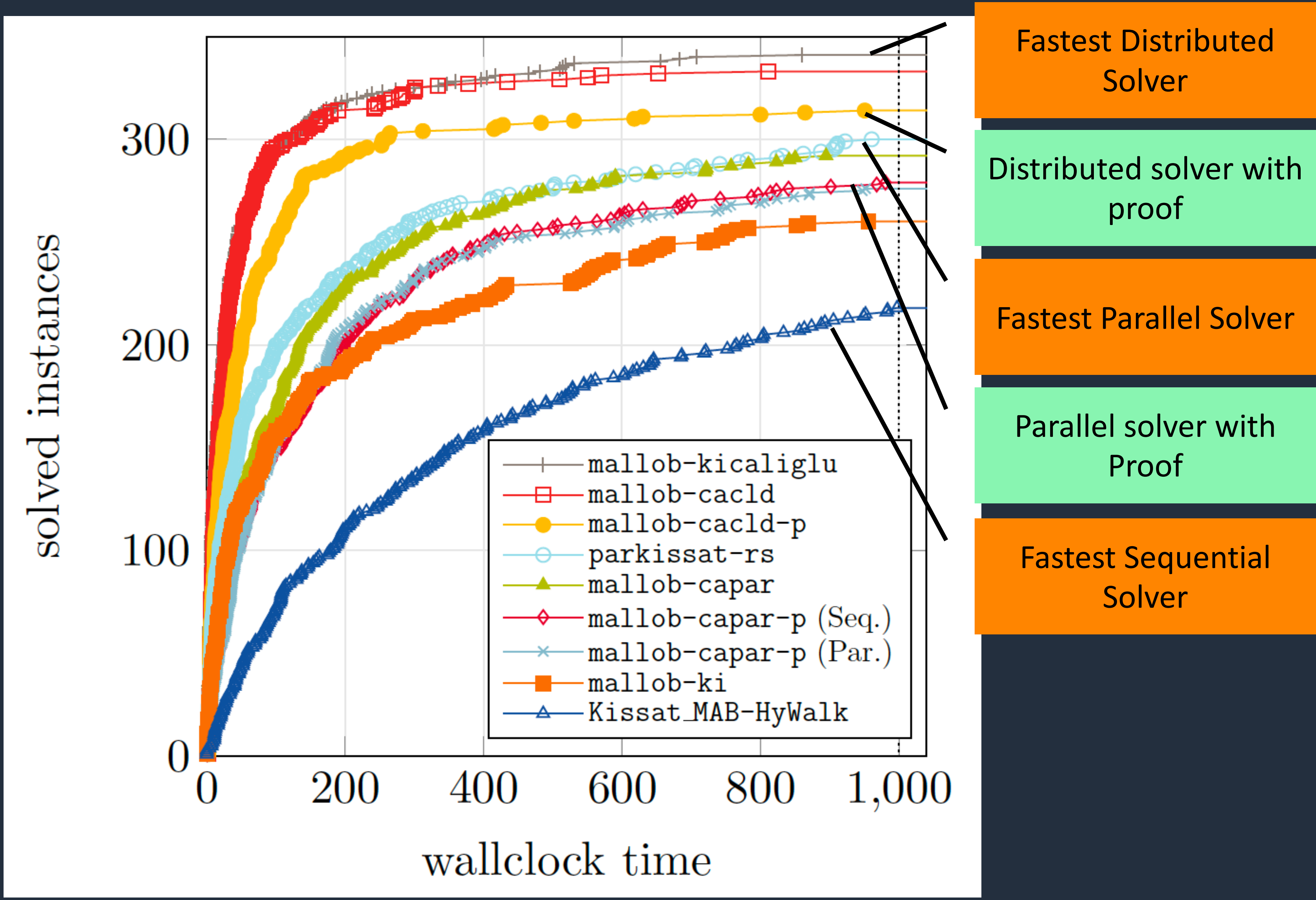
Experiment

- Run parallel and distributed solvers on SAT-Comp 2022 benchmarks (400 problems)
- Compare against the winners of the SAT-Comp 2022 tracks
  - Sequential track – running on m6i.4xlarge AWS instance w/16 cores, 64GB RAM
  - Parallel track - running on m6i.16xlarge AWS instance w/64 cores, 256 GB RAM
  - Distributed track – running on 100 * m6i.4xlarge AWS instances
- Track solving time and proof reconstruction time

Results: Solving

Distributed proof solver is ~1.8x slower (median) than best distributed solver

Distributed proof solver is ~17x faster (median) than best sequential solver

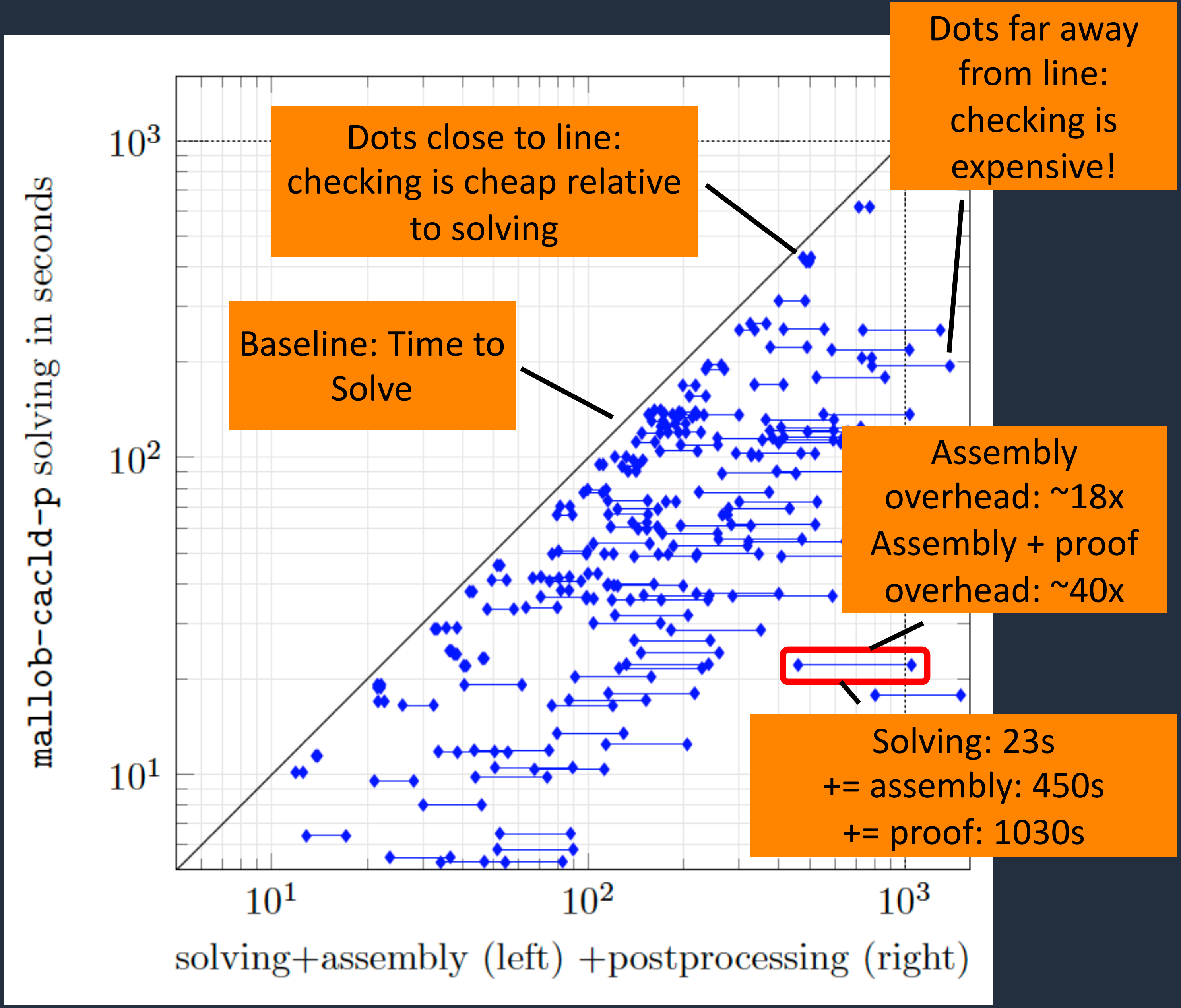


Results: Proof

(Not shown) Average overhead for proof checking of sequential solver: 1.1x solve time

Average overhead for proof assembly and postprocessing (proof checking) for distributed solver: 5.1x solve time

Distributed solver is *relatively slower* at performing proofs than sequential solver and has *substantial variance* with solving time.



Dots close to line: checking is cheap relative to solving

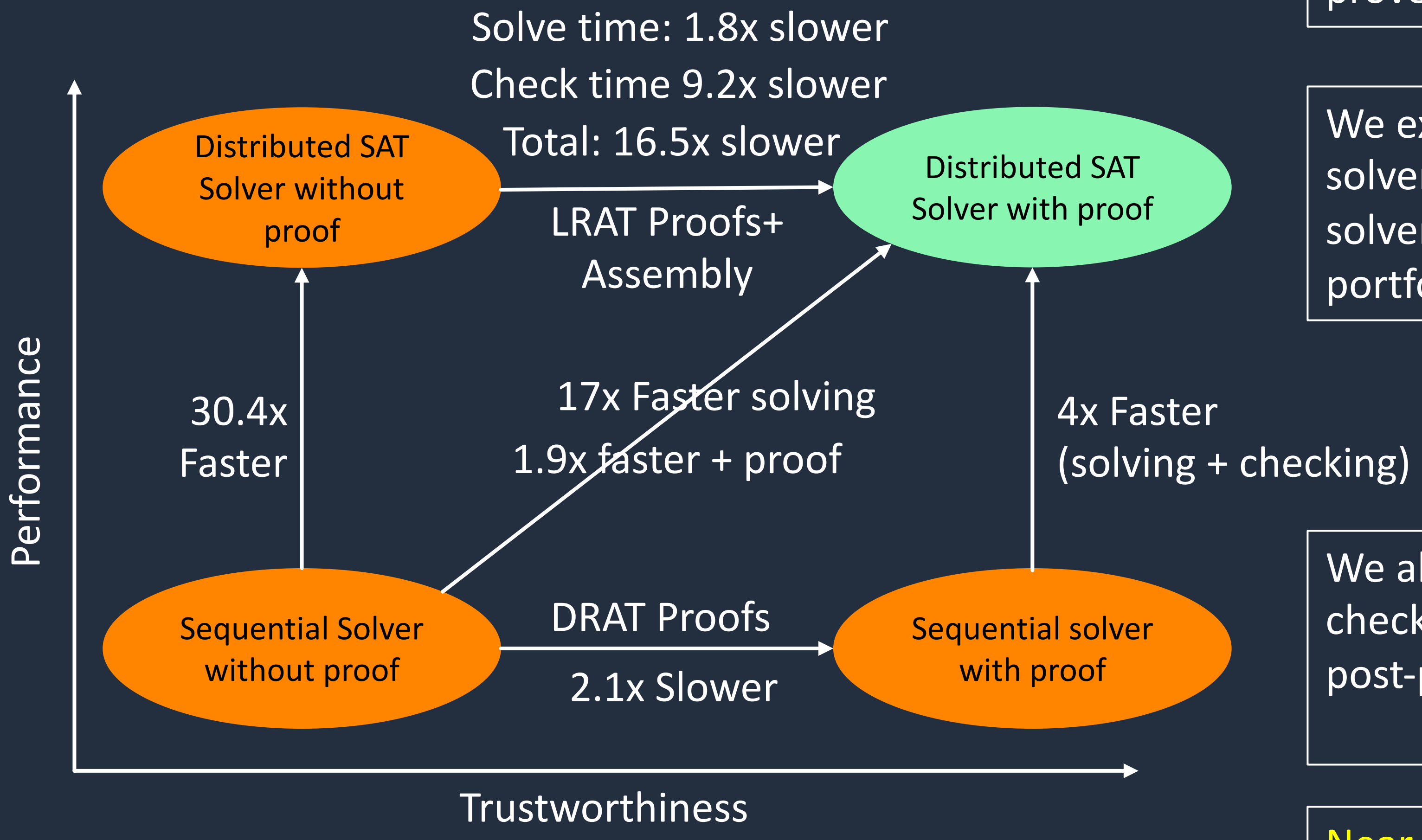
Dots far away from line: checking is expensive!

Baseline: Time to Solve

Assembly overhead: ~18x
Assembly + proof overhead: ~40x

Solving: 23s
+= assembly: 450s
+= proof: 1030s

Solver Quadrants



Even though proof checking adds more overhead than sequential solving proof checking, the total time improvement for the distributed prover is 4x.

We expect to be able to improve solver performance once more solvers support LRAT to diversify our portfolio.

We also expect to be able to improve checker performance using a parallel post-processor and proof checker.

Near term goal:
10x solving + checking

Talk Takeaways

SAT solvers are general purpose **reasoning engines** that can be used for both **verification** and **optimization** purposes.

These solvers are the basis for **most of the automated reasoning tools** in use today.

Recent work in distributed **Clause Sharing Portfolio Solvers** has led to **dramatic performance improvements** over state-of-the-art sequential SAT solvers.

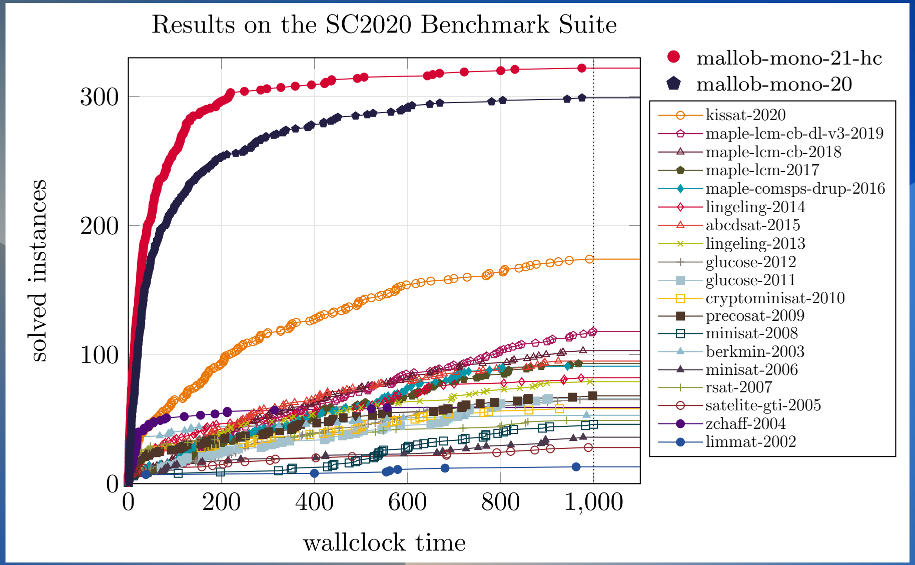
We have created a scalable approach for **proofs for distributed solvers** to ensure the correctness of solver results.

Thank you!

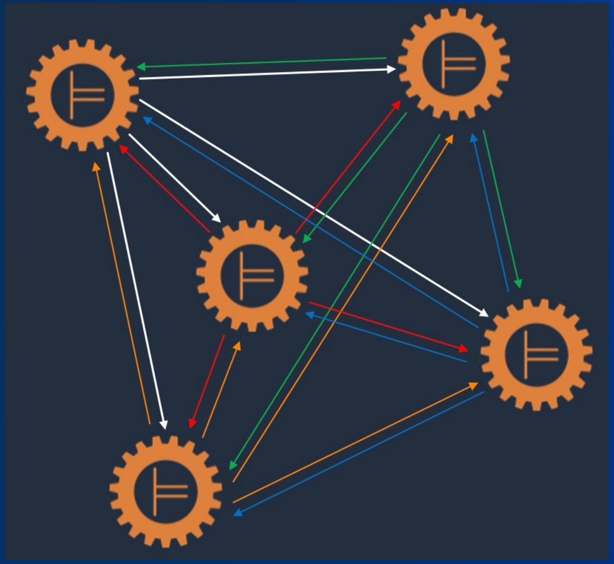
Mike Whalen

mww@amazon.com

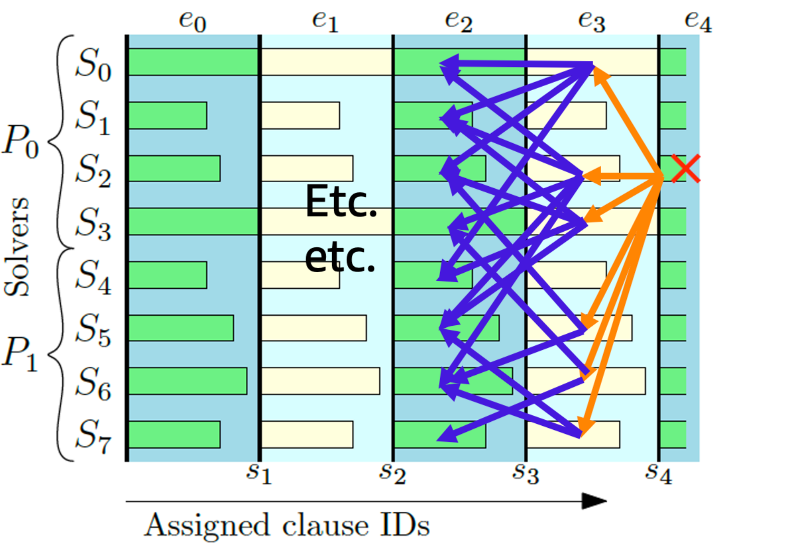
Performance on Solving:



Clause Sharing:



Reconstruction Mechanism:



Etc.
etc.