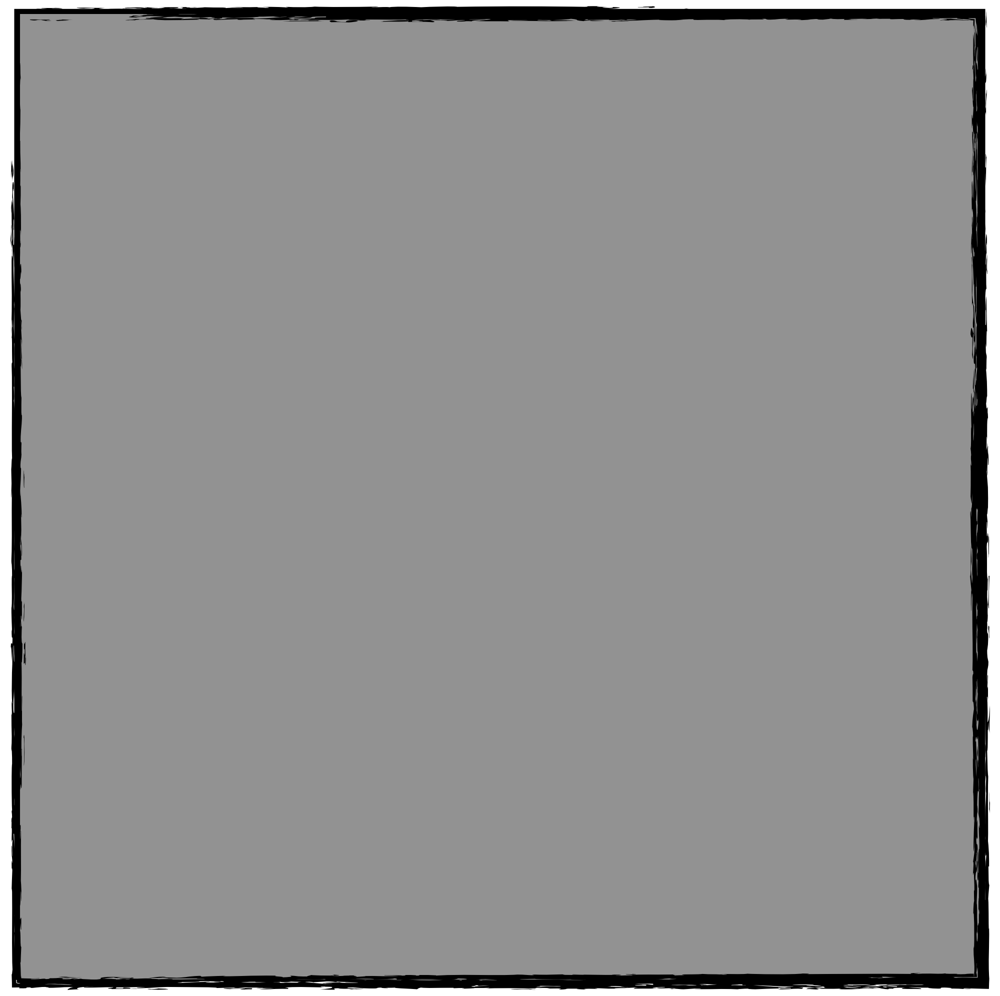
Deterministic State Space Exploration

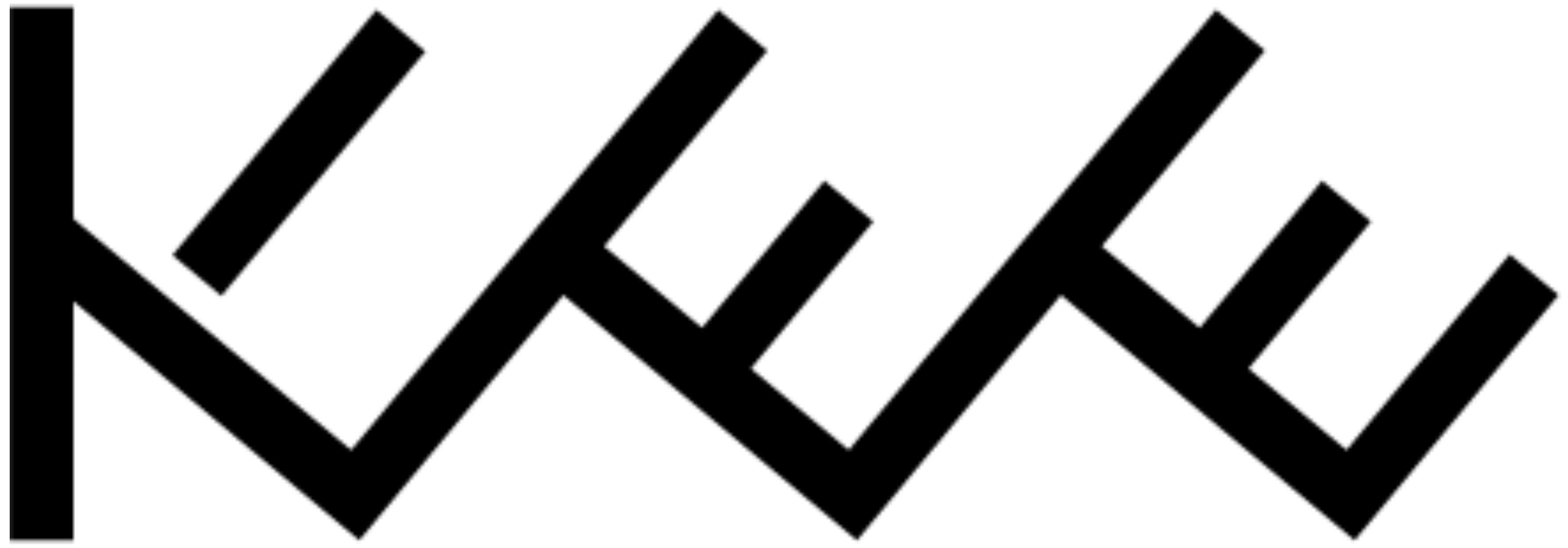
KLEE ‘24

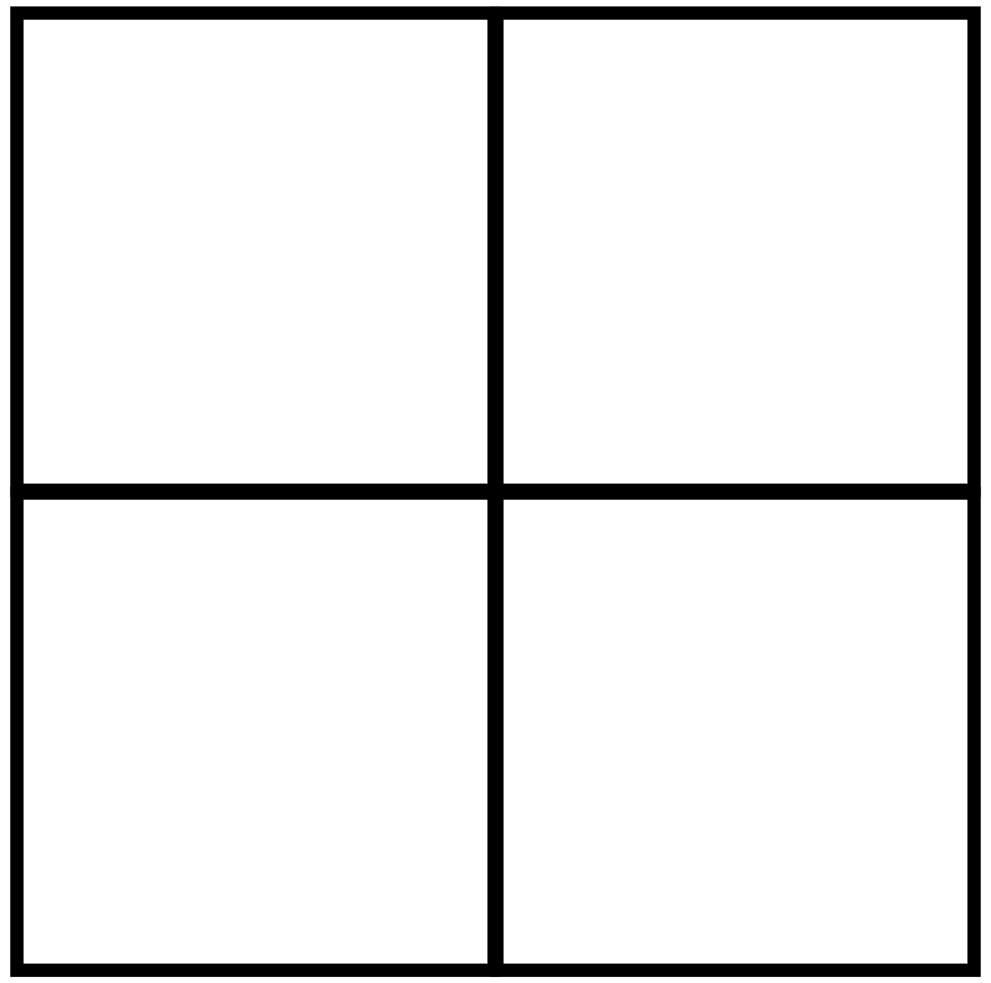
SOFTWARE RELIABILITY
GROUP

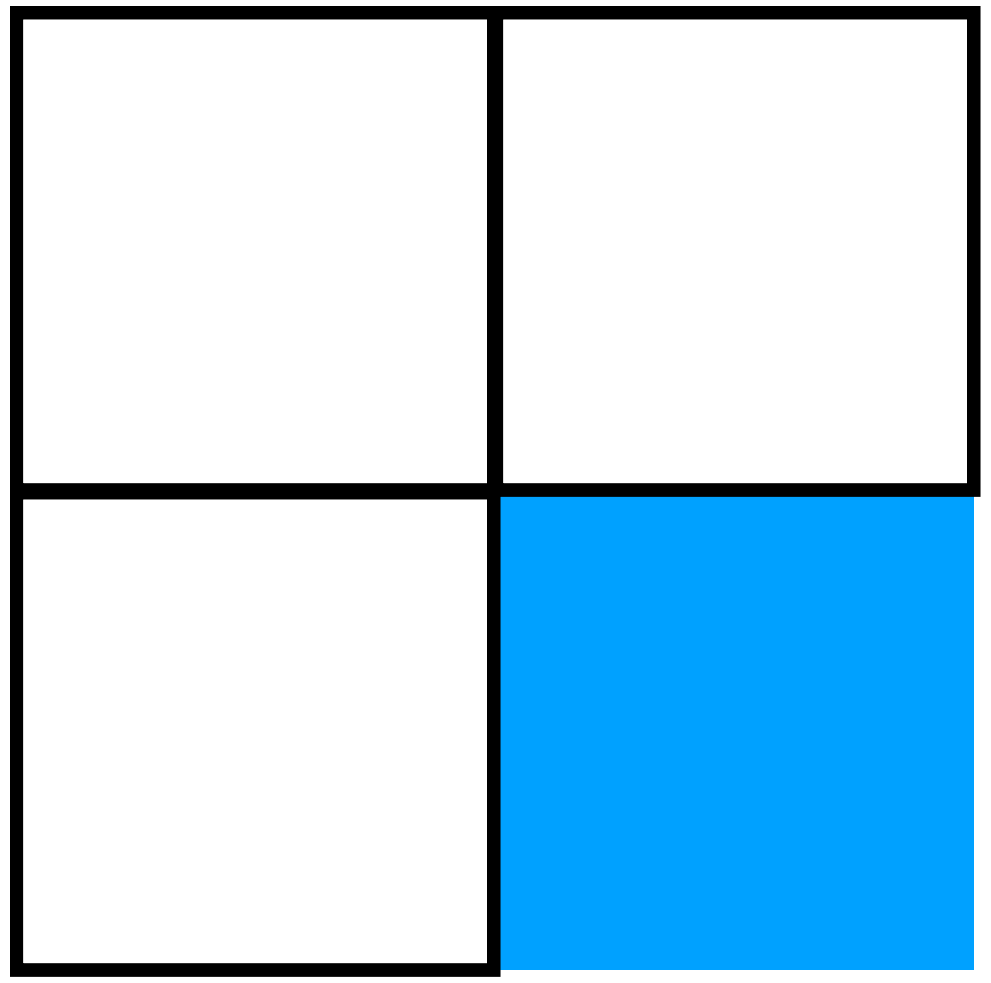
Under submission as Special Issue in STTT Journal
Based on 1st RRRR 2022, Workshop, Munich, Germany

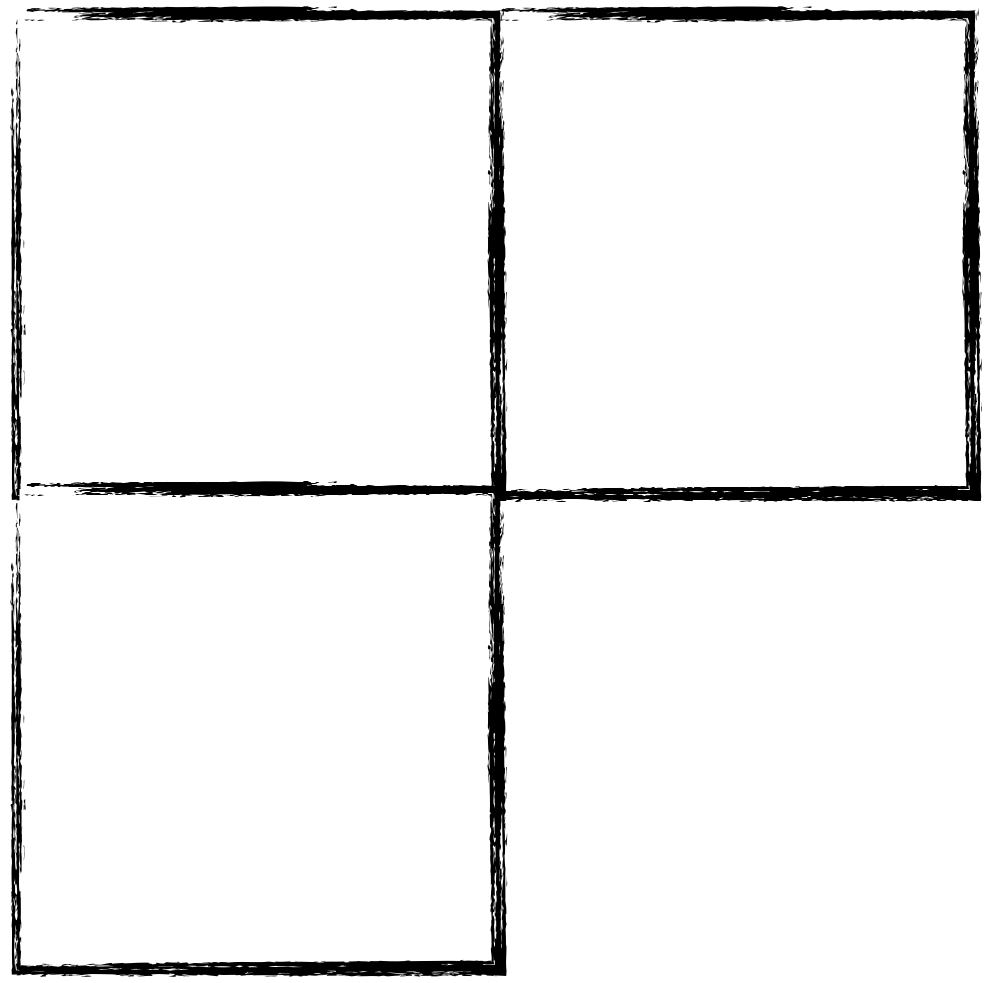
Martin Nowack

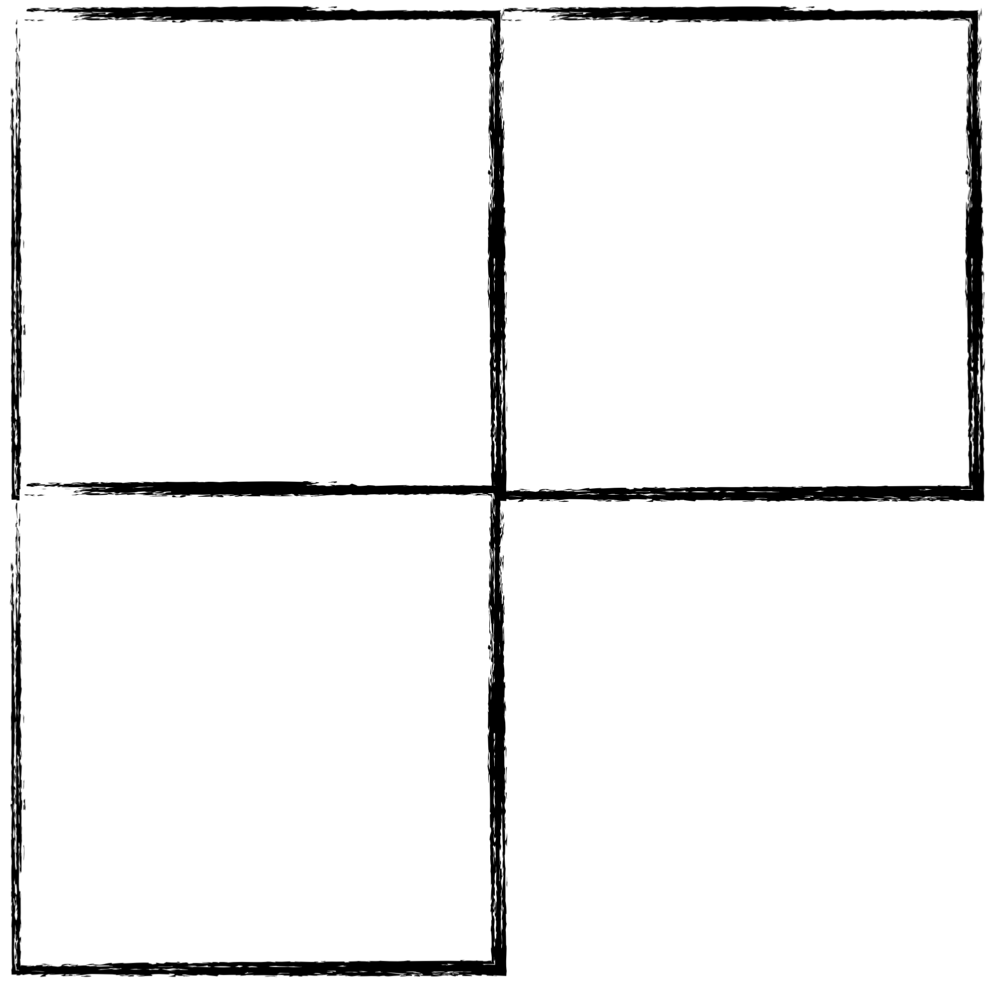


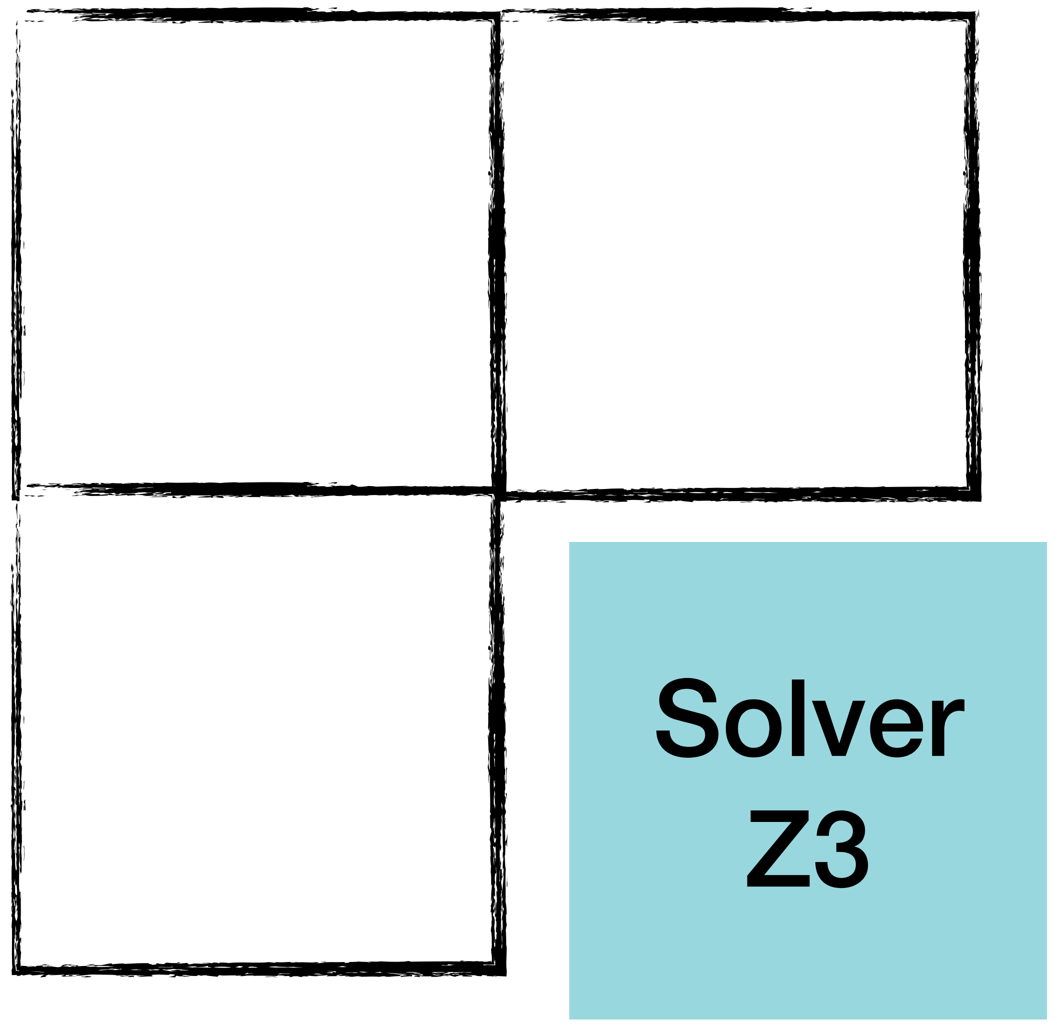




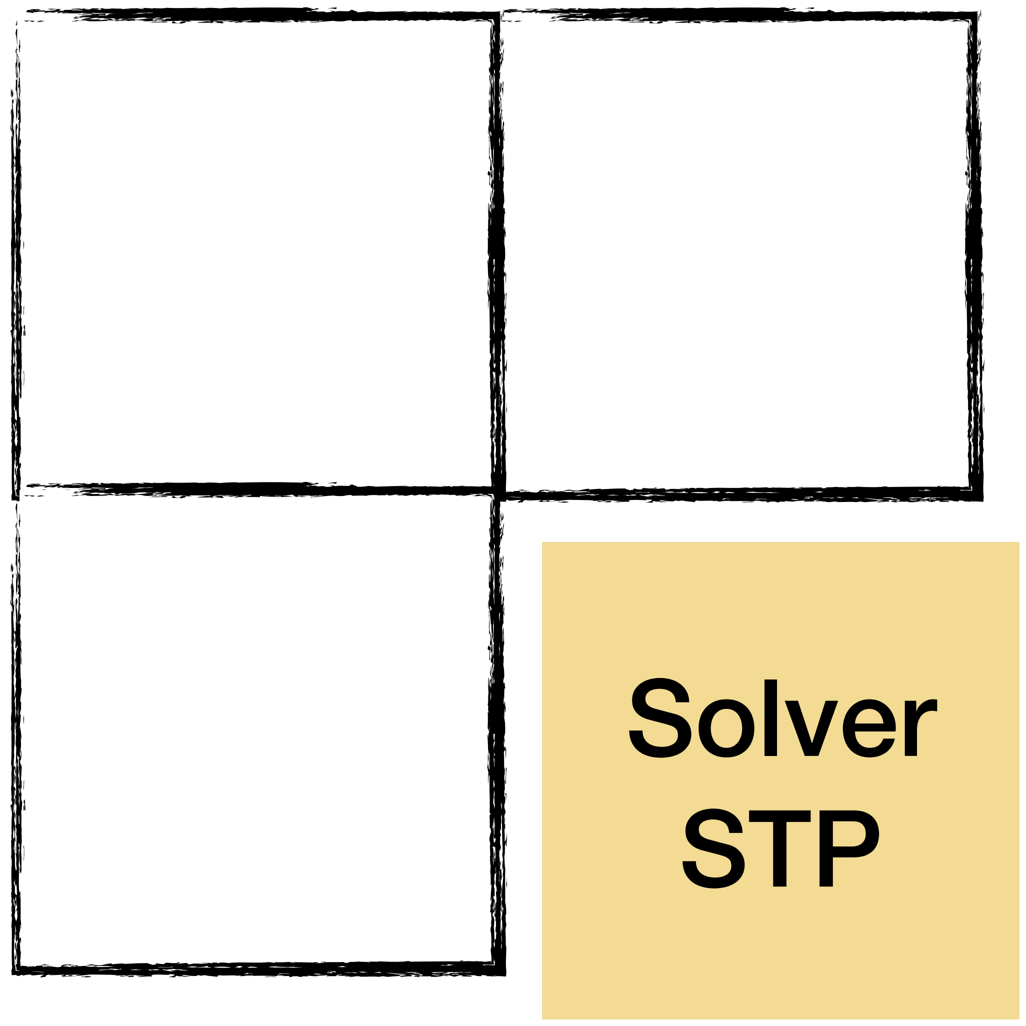




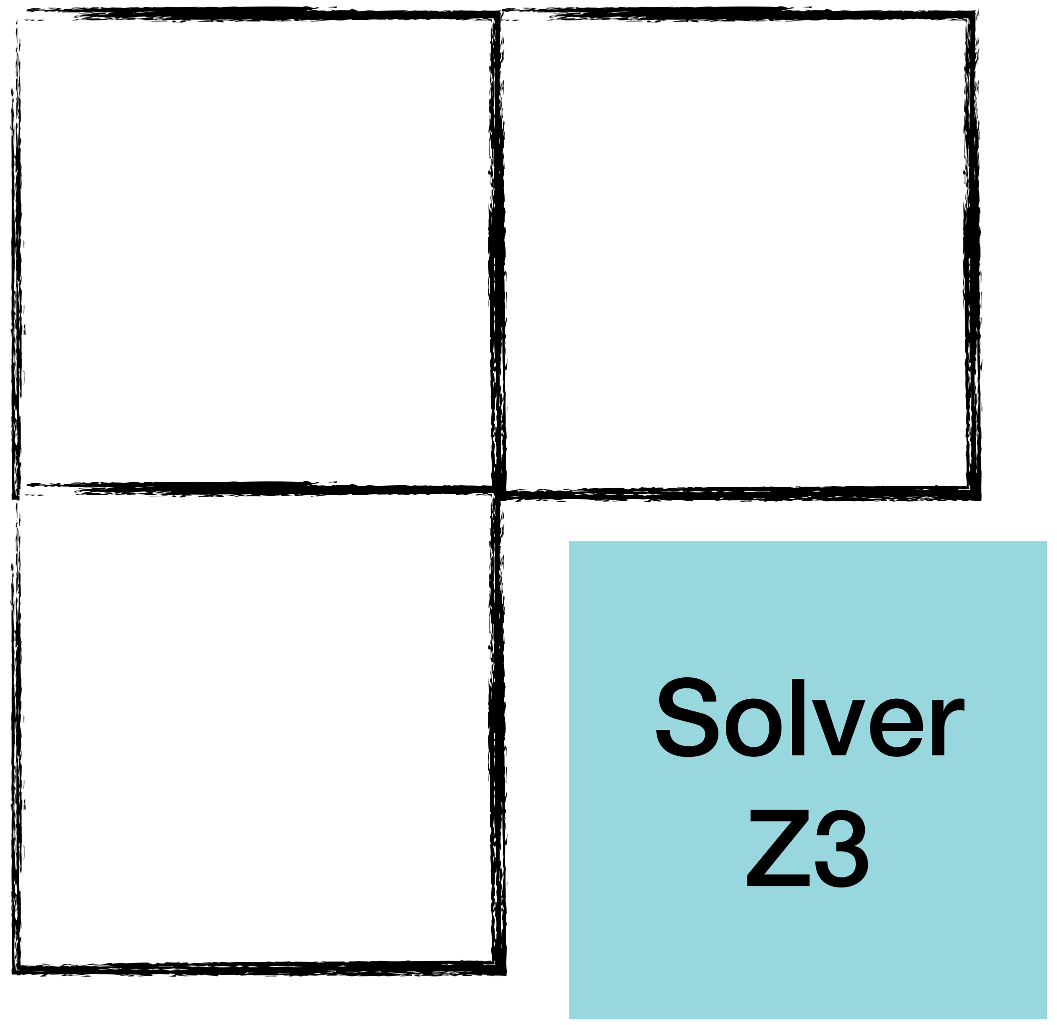




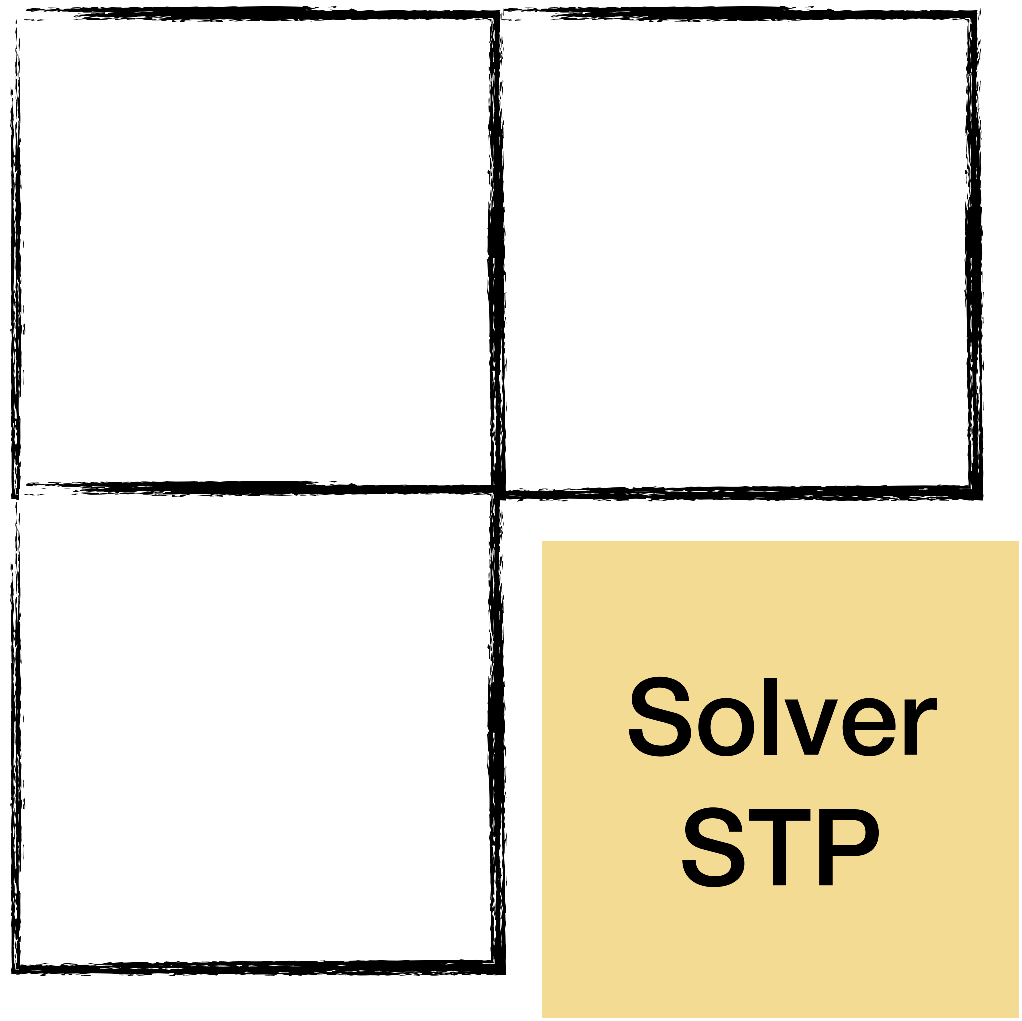
Solver
Z3



Solver
STP

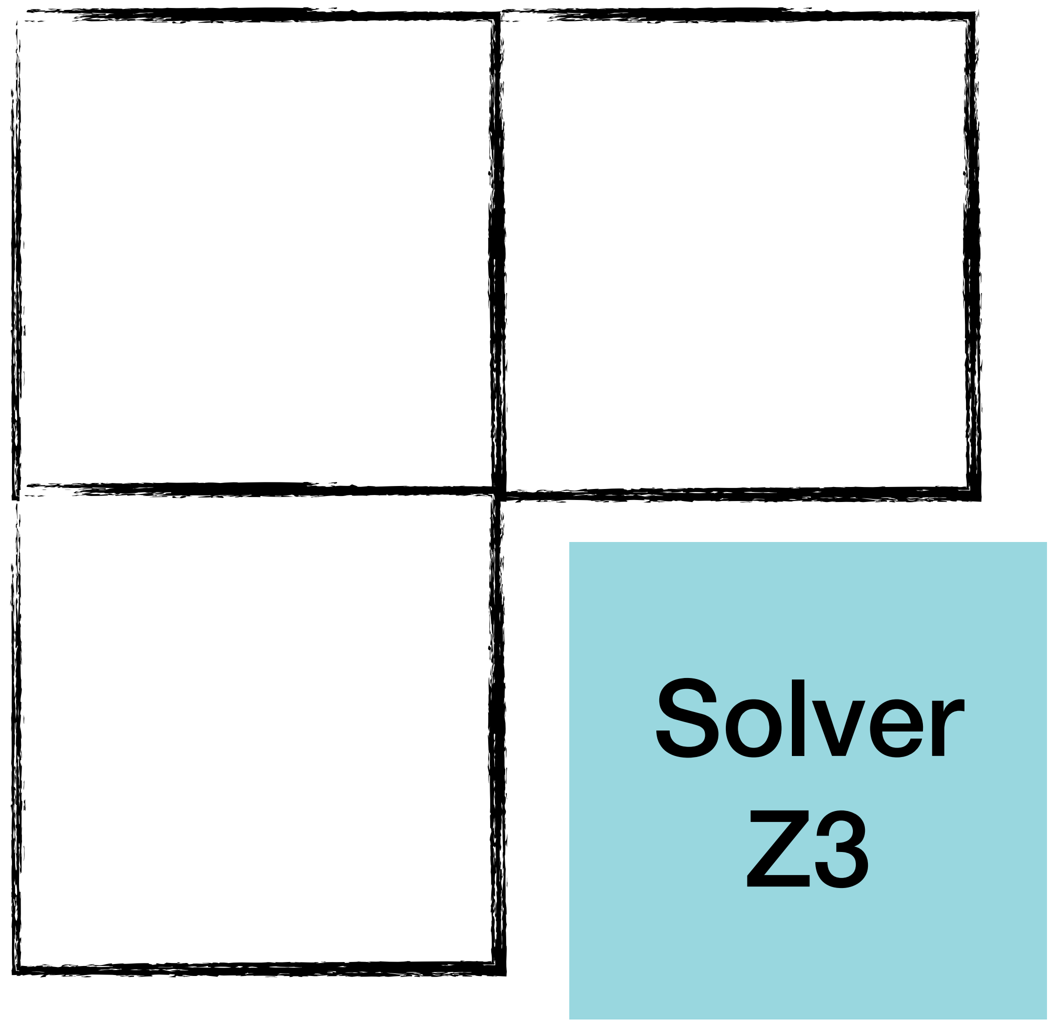


Solver Z3

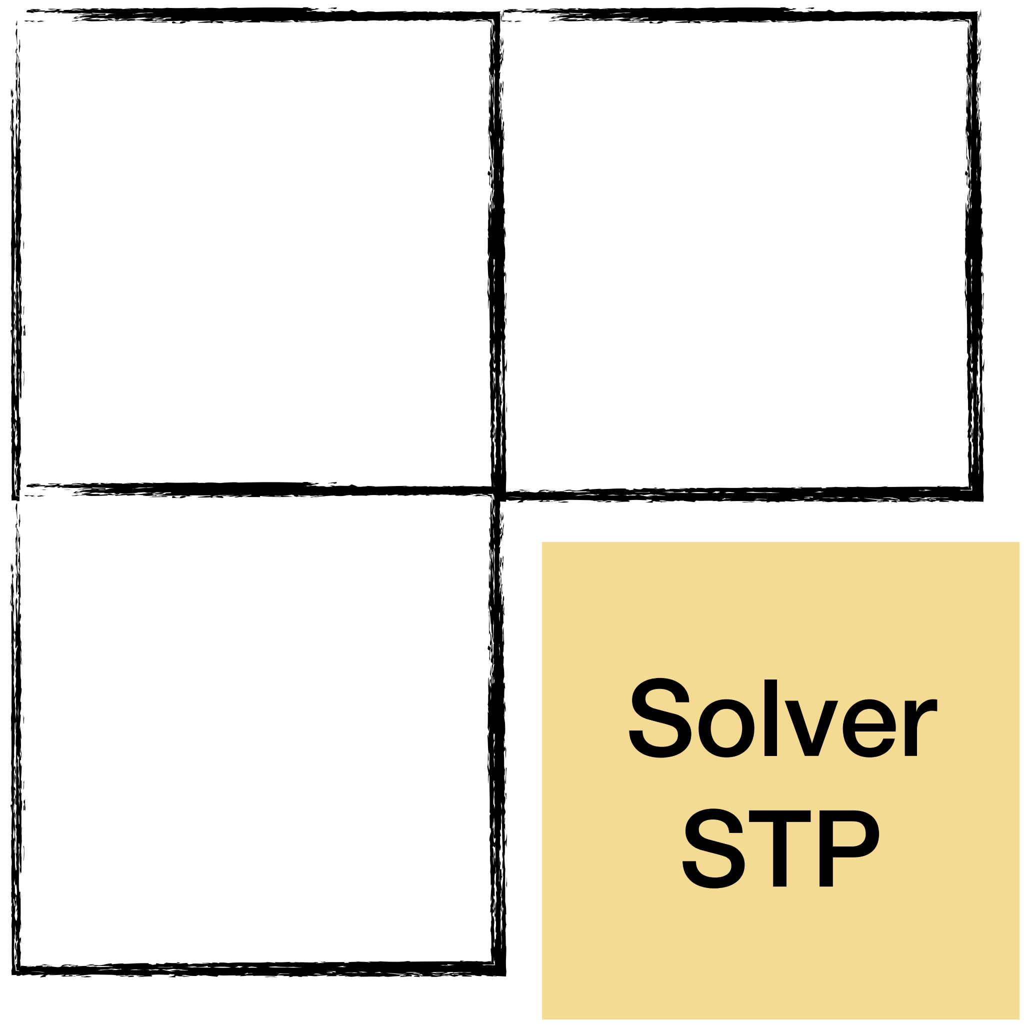


Solver STP

```
./TestApp 1 2 3
```



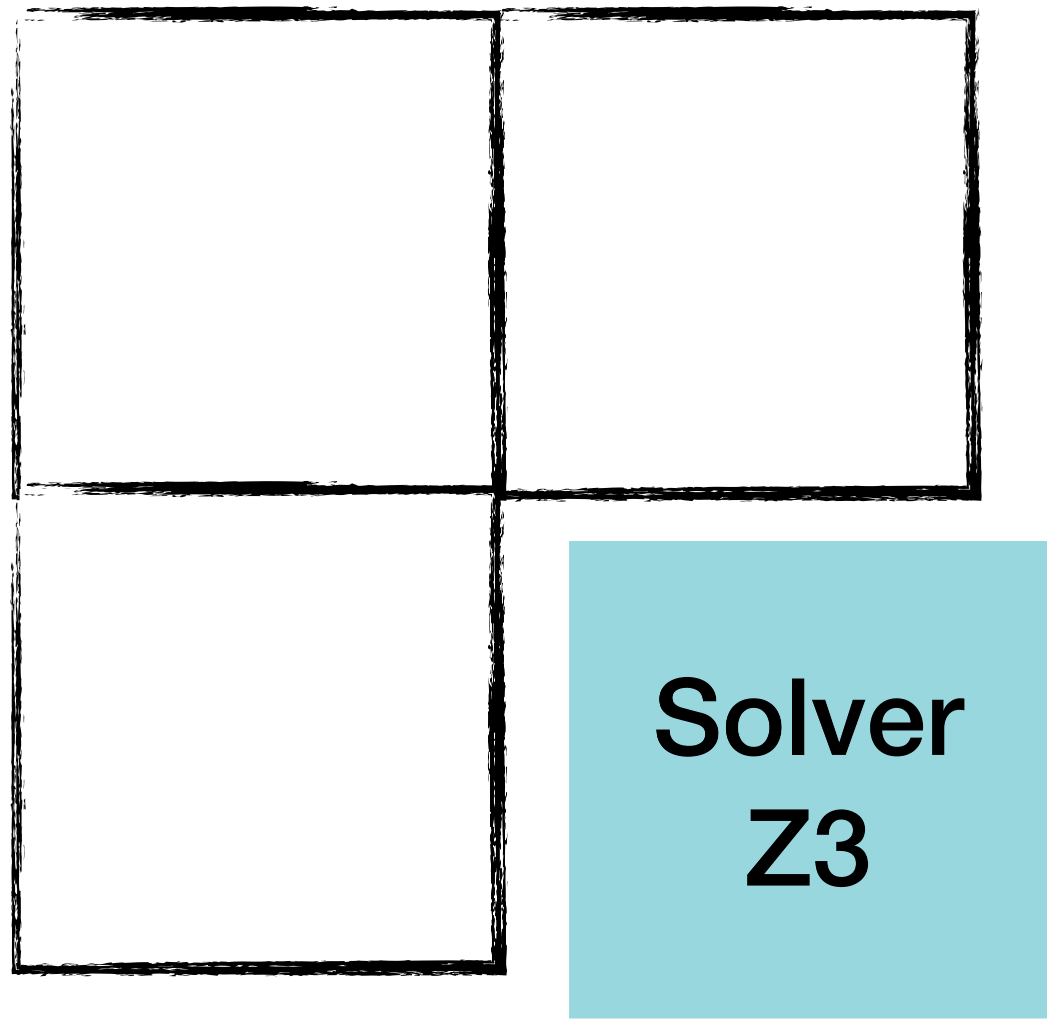
Solver Z3



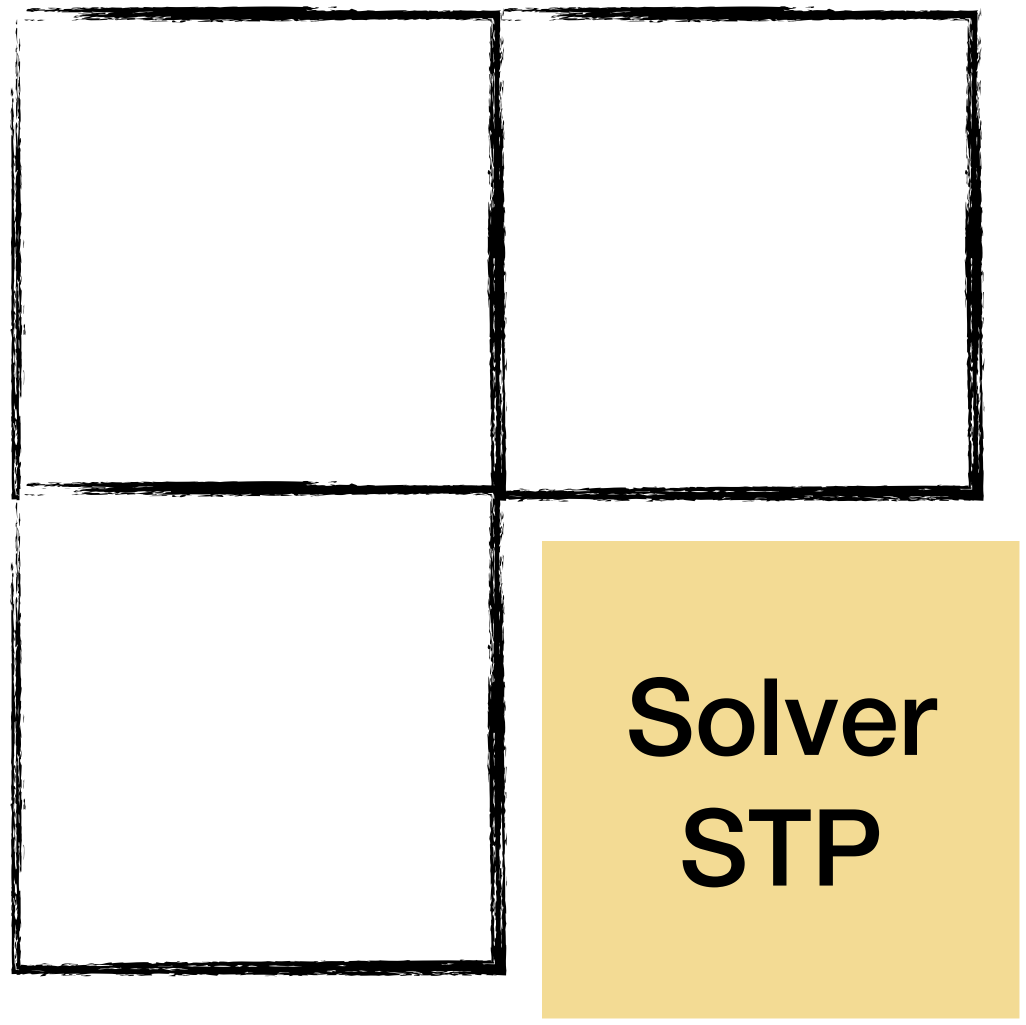
Solver STP

```
./TestApp 1 2 3
```

⌛



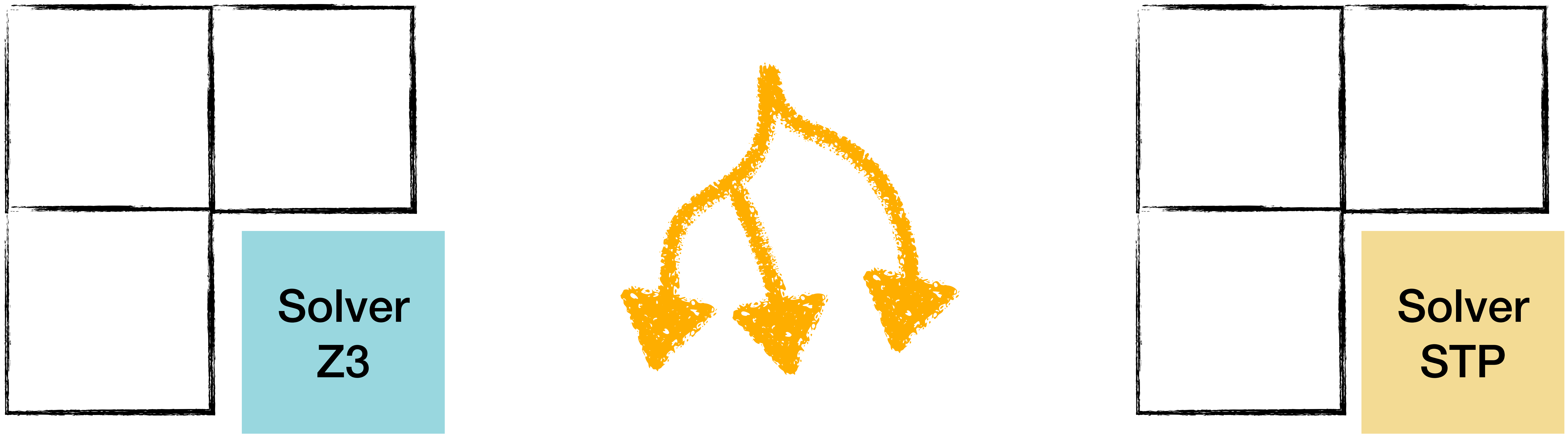
Solver
Z3



Solver
STP

./TestApp 1 2 3

GCov

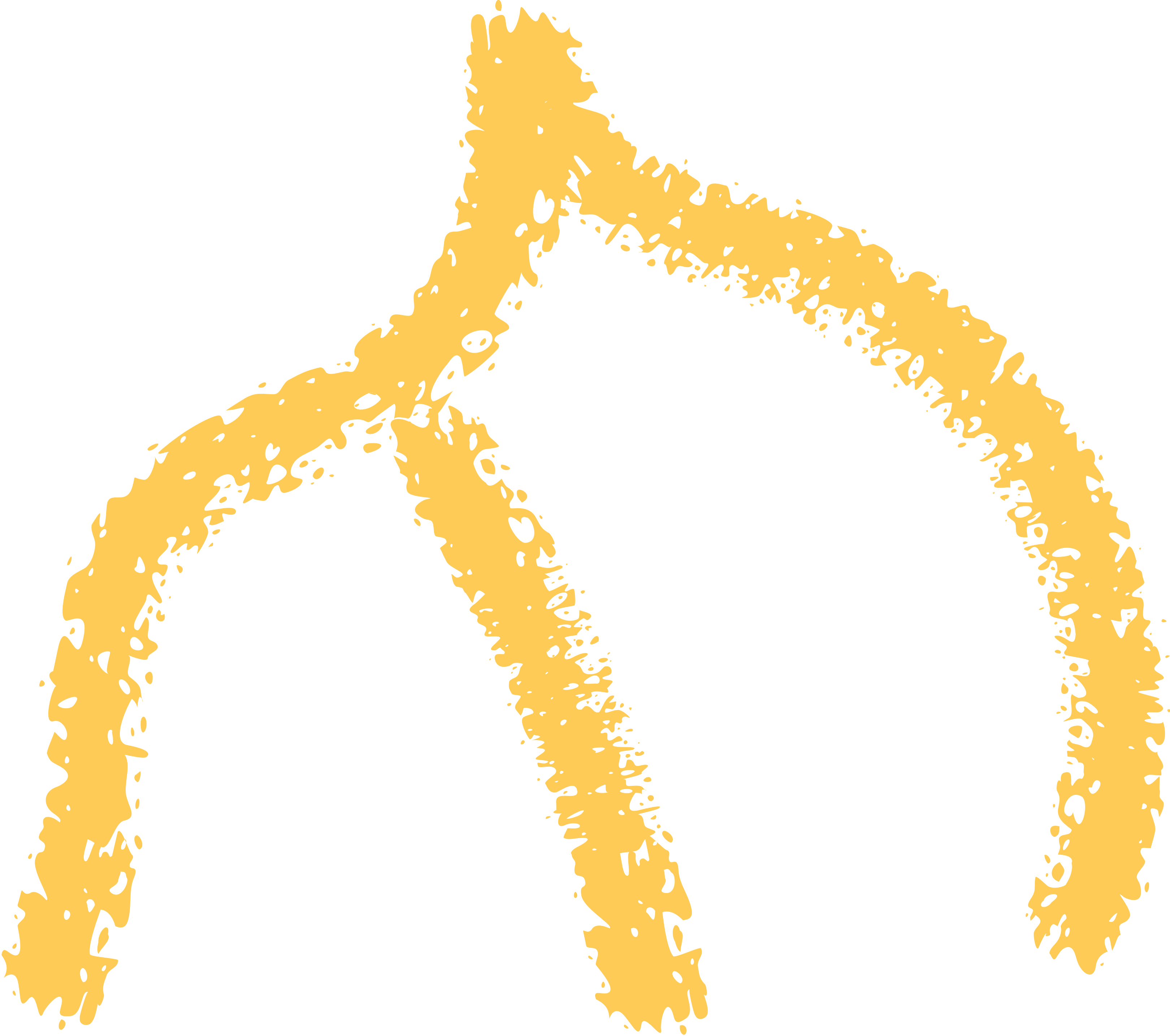


Solver Z3

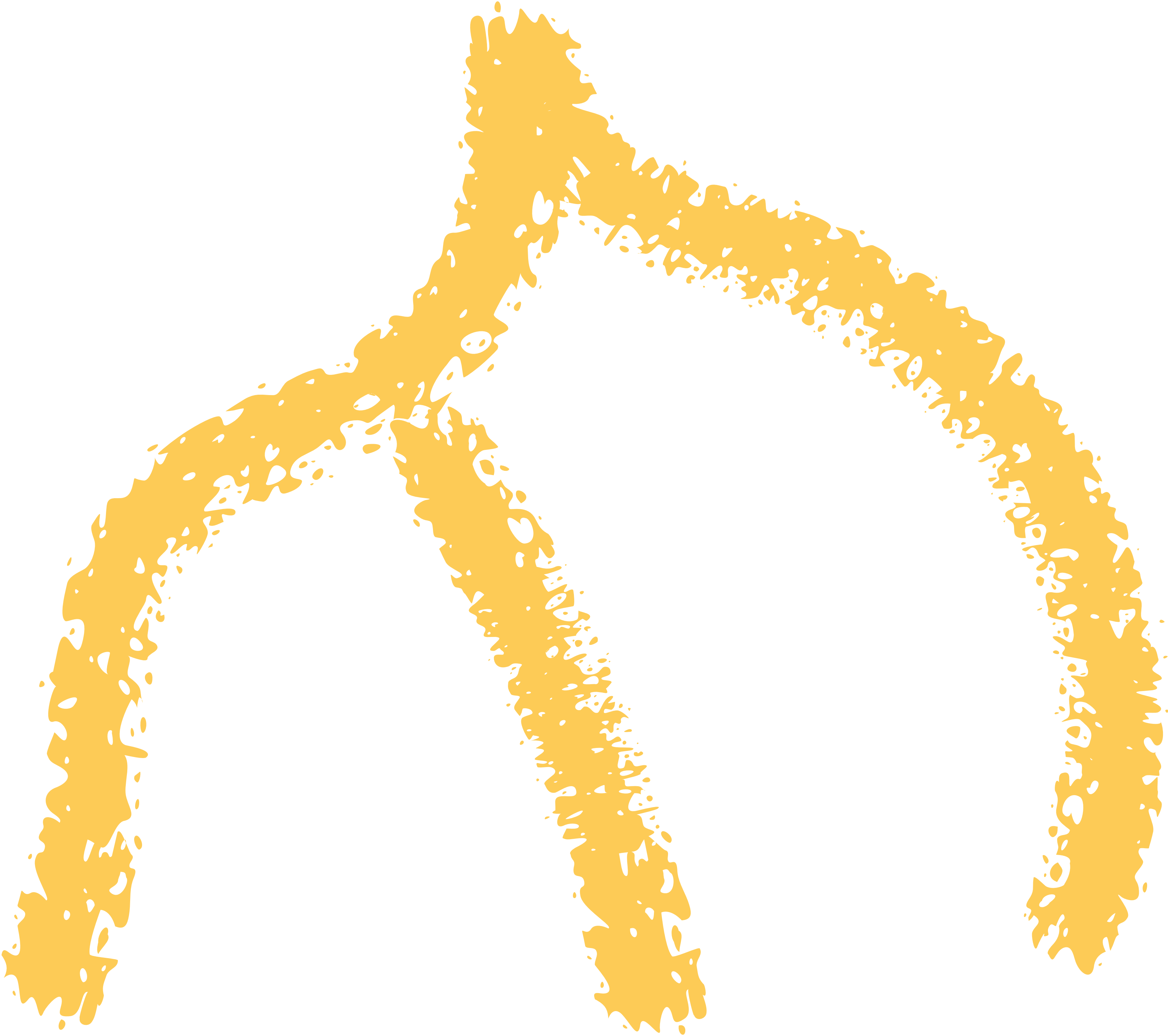
Solver STP

./TestApp 1 2 3

GCov

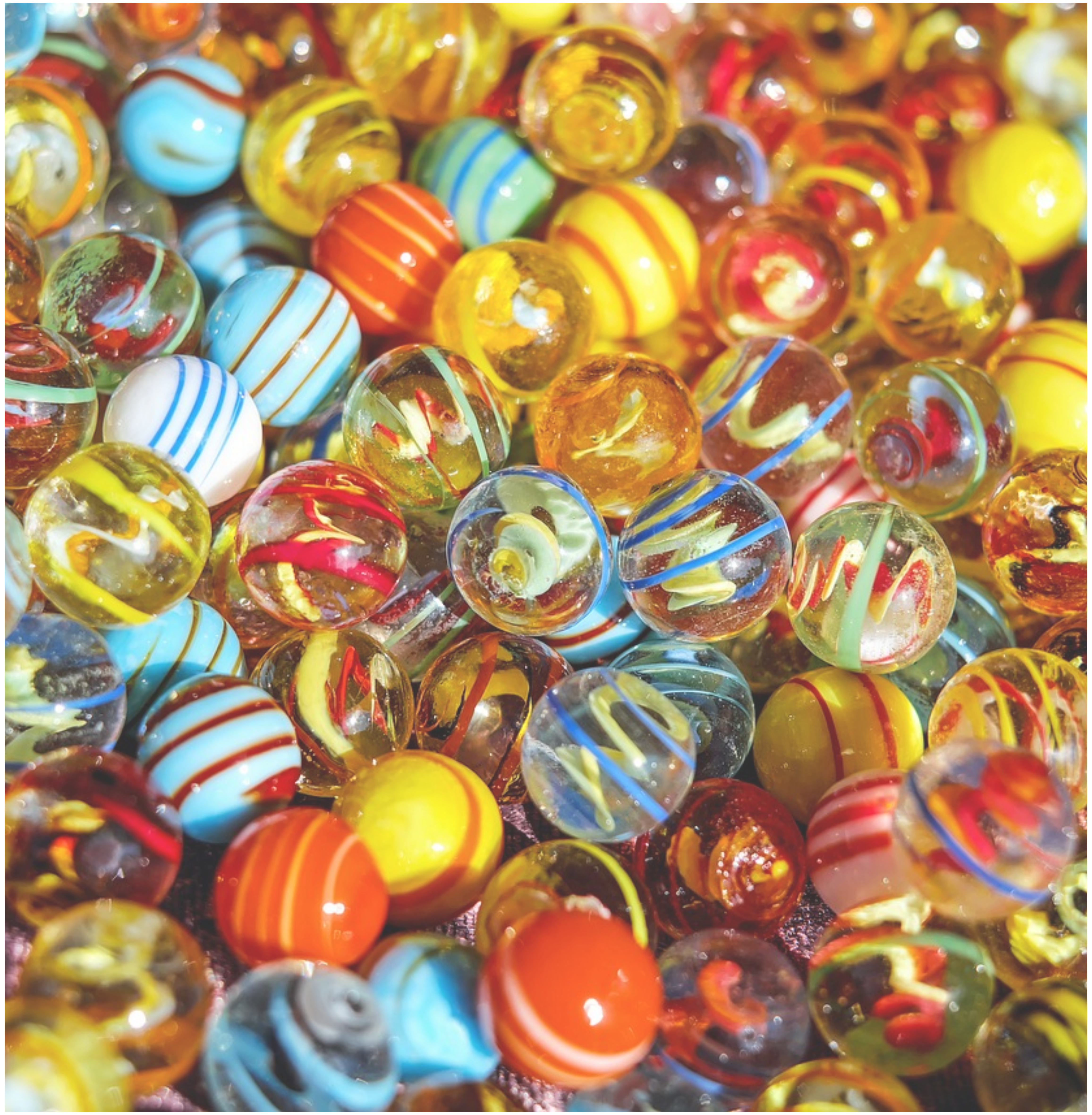


The many states …



The many states …

- Every path has a different cost:
    - Different number of instructions
    - Different constraints
    - Different costs solving them
- Too many paths



Goal: Fine-Grain Replication of Workload

... where appropriate

Deterministic State Space Exploration

Deterministic State Space Exploration

Each instruction is executed in the same order and by the same state.

Deterministic State Space Exploration

Each instruction is executed in the same order and by the same state.

Deterministic State Space Exploration

Each instruction is executed in the same order and by the same state.

```
char * a = malloc(1024);
int32 i = symbolic;

a[i]++;
if (i != 12345)
{
  a[i-2] = a[i] * 2;
} else {
```

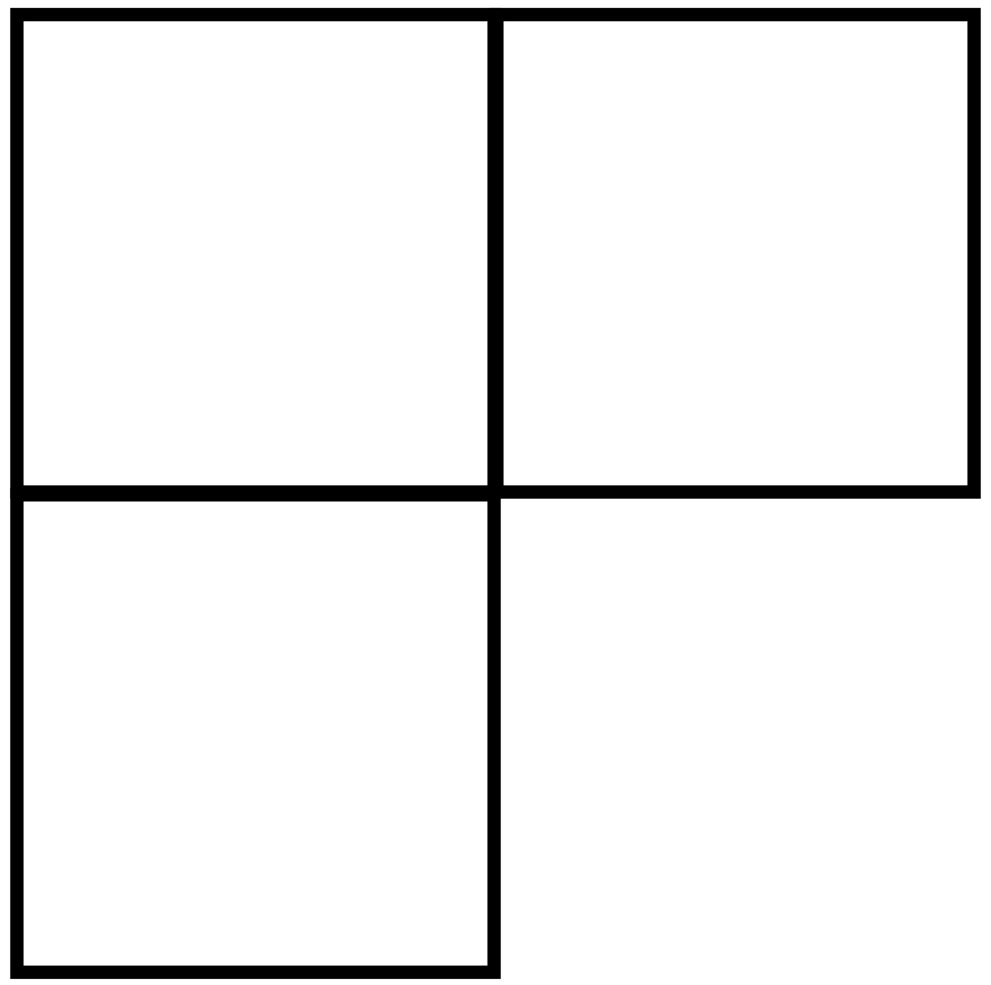
Deterministic State Space Exploration

Each instruction is executed in the same order and by the same state.

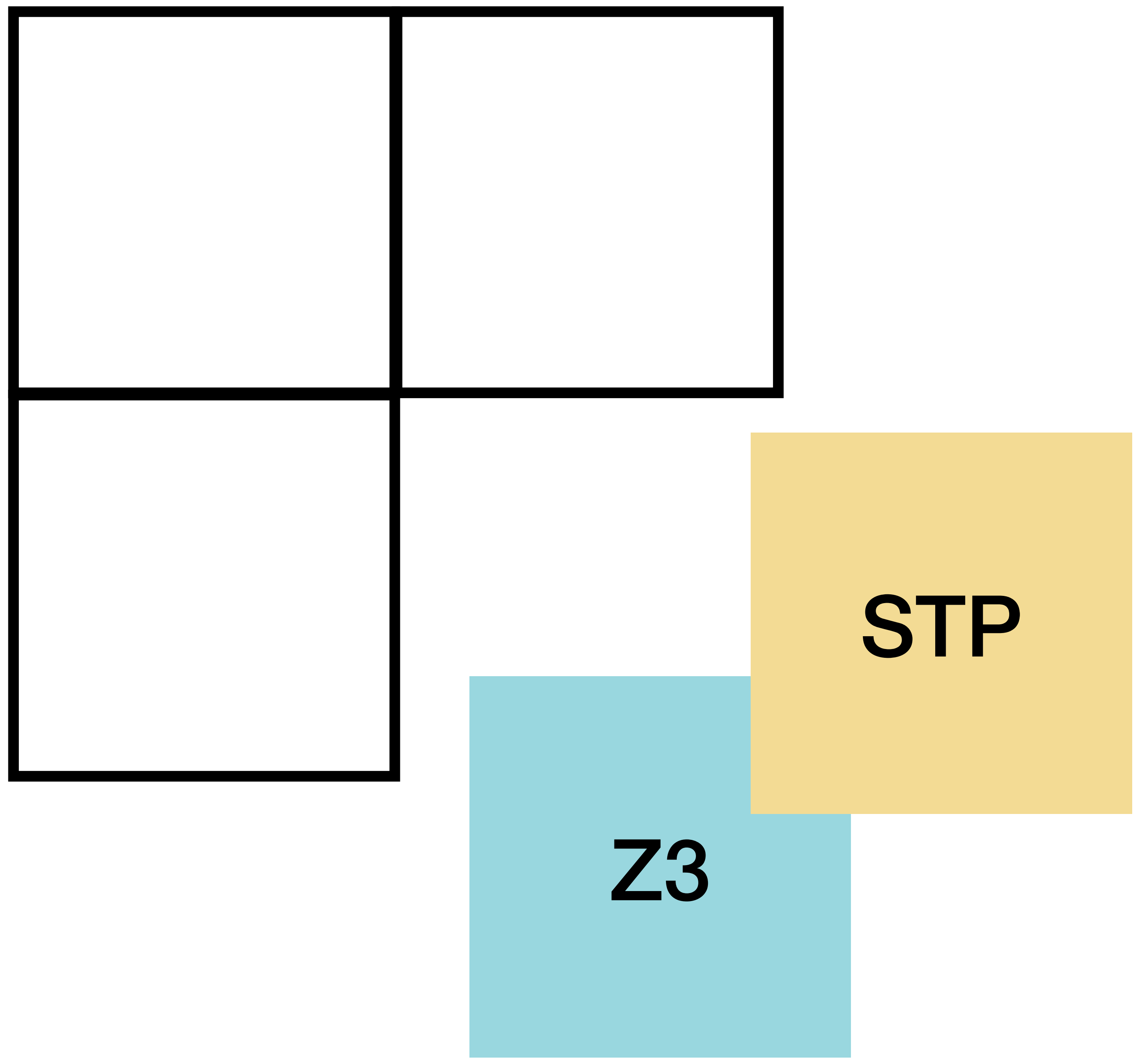
```
  %1 = alloca i8*, align 8
  %2 = alloca i32, align 4
  call void @llvm.dbg.declare(metadata i8** %1, metadata !13, metadata !DIExpression()), !dbg !16
  %3 = call i8* @malloc(i64 noundef 1024), !dbg !17
  store i8* %3, i8** %1, align 8, !dbg !16
  call void @llvm.dbg.declare(metadata i32* %2, metadata !18, metadata !DIExpression()), !dbg !20
  %4 = call i32 (...) @make_symbolic(), !dbg !21
  store i32 %4, i32* %2, align 4, !dbg !20
  %5 = load i8*, i8** %1, align 8, !dbg !22
  %6 = load i32, i32* %2, align 4, !dbg !23
  %7 = sext i32 %6 to i64, !dbg !22
  %8 = getelementptr inbounds i8, i8* %5, i64 %7, !dbg !22
  %9 = load i8, i8* %8, align 1, !dbg !24
  %10 = add i8 %9, 1, !dbg !24
  store i8 %10, i8* %8, align 1, !dbg !24
  %11 = load i32, i32* %2, align 4, !dbg !25
  %12 = icmp ne i32 %11, 12345, !dbg !27
  br i1 %12, label %13, label %27, !dbg !28

13:                                               ; preds = %0
  %14 = load i8*, i8** %1, align 8, !dbg !29
  %15 = load i32, i32* %2, align 4, !dbg !31
  %16 = sext i32 %15 to i64, !dbg !29
  %17 = getelementptr inbounds i8, i8* %14, i64 %16, !dbg !29
  %18 = load i8, i8* %17, align 1, !dbg !29
  %19 = sext i8 %18 to i32, !dbg !29
```

Example of its Application

Example: Evaluate Different Solvers

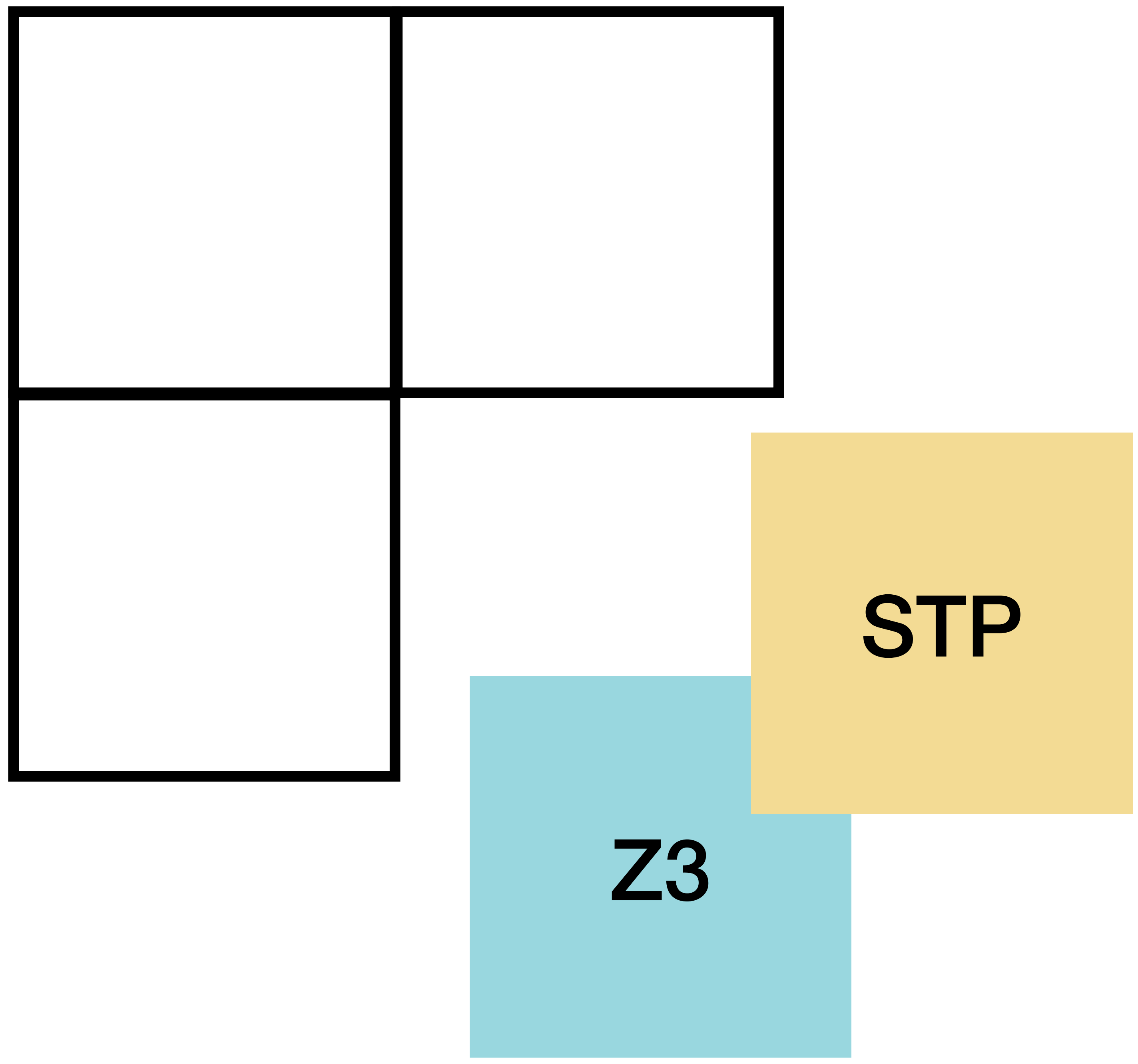


Example: Evaluate Different Solvers



Example: Evaluate Different Solvers

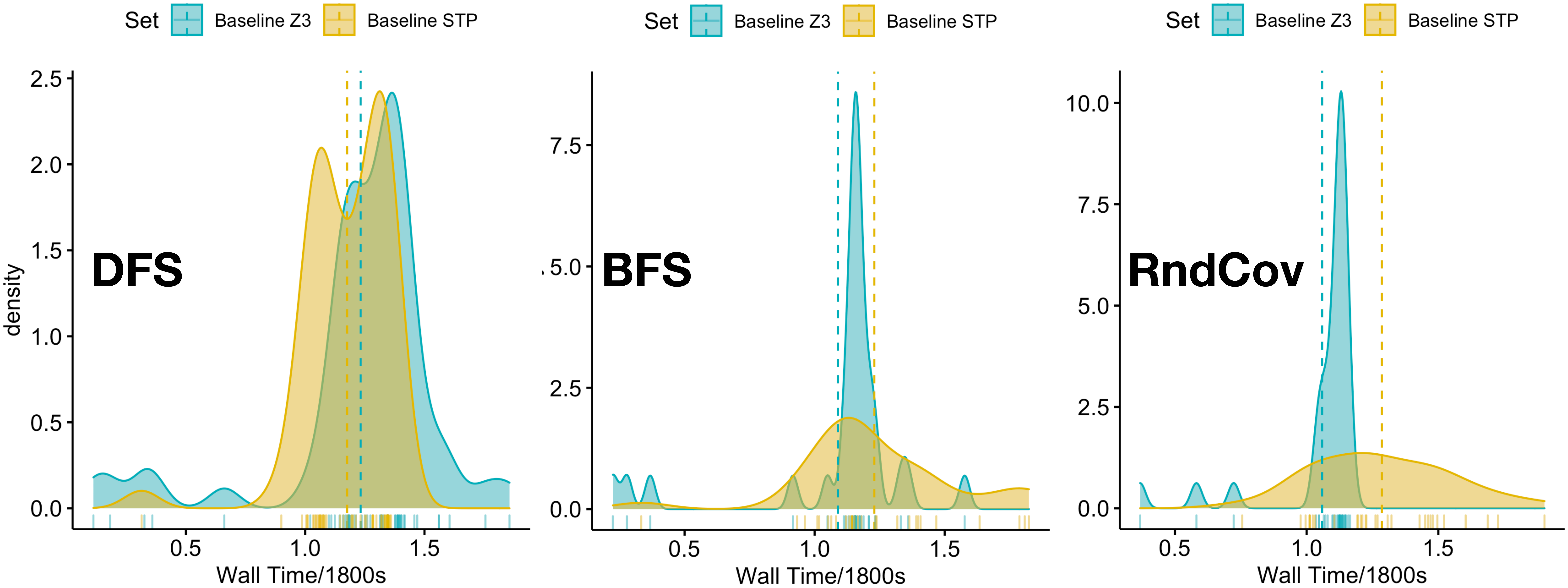
- 80 applications: GNU CoreUtils
- 3 different searchers:
  - DFS, BFS, Rnd+Cov
- Fixed number of instructions: ~30min



Measure:
Execution time -> normalised to 30 min

Solvers: Z3 vs STP

DFS vs. BFS vs. RndCov



BFS as an Example

Let’s Go Deterministic

BFS

Execution Time

Let’s Go Deterministic

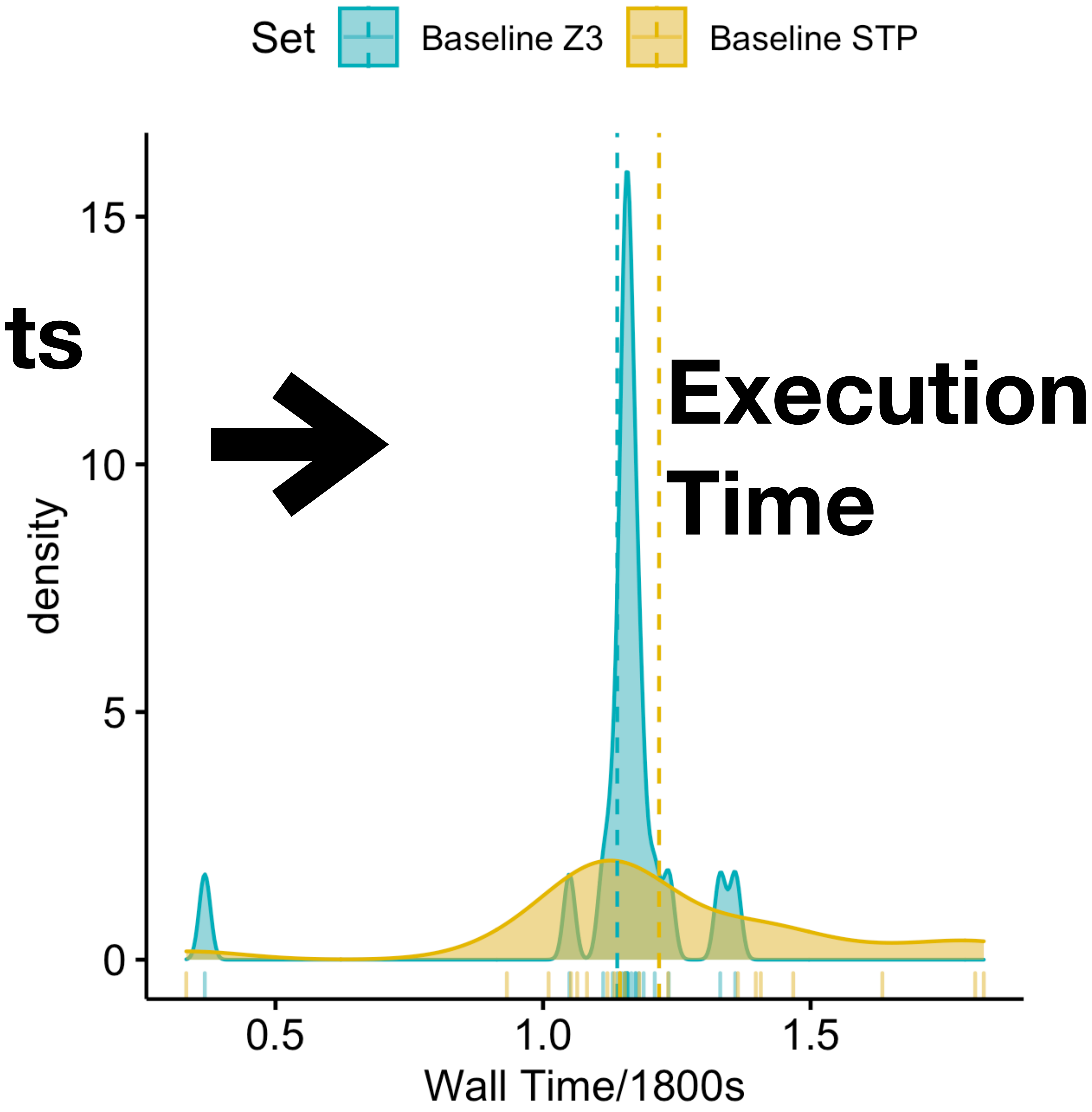
BFS

Constraints Generation Time + Constraints Solving Time → Execution Time

Let’s Go Deterministic

BFS

Constraints Generation Time + **Constraints Solving Time** → **Execution Time**

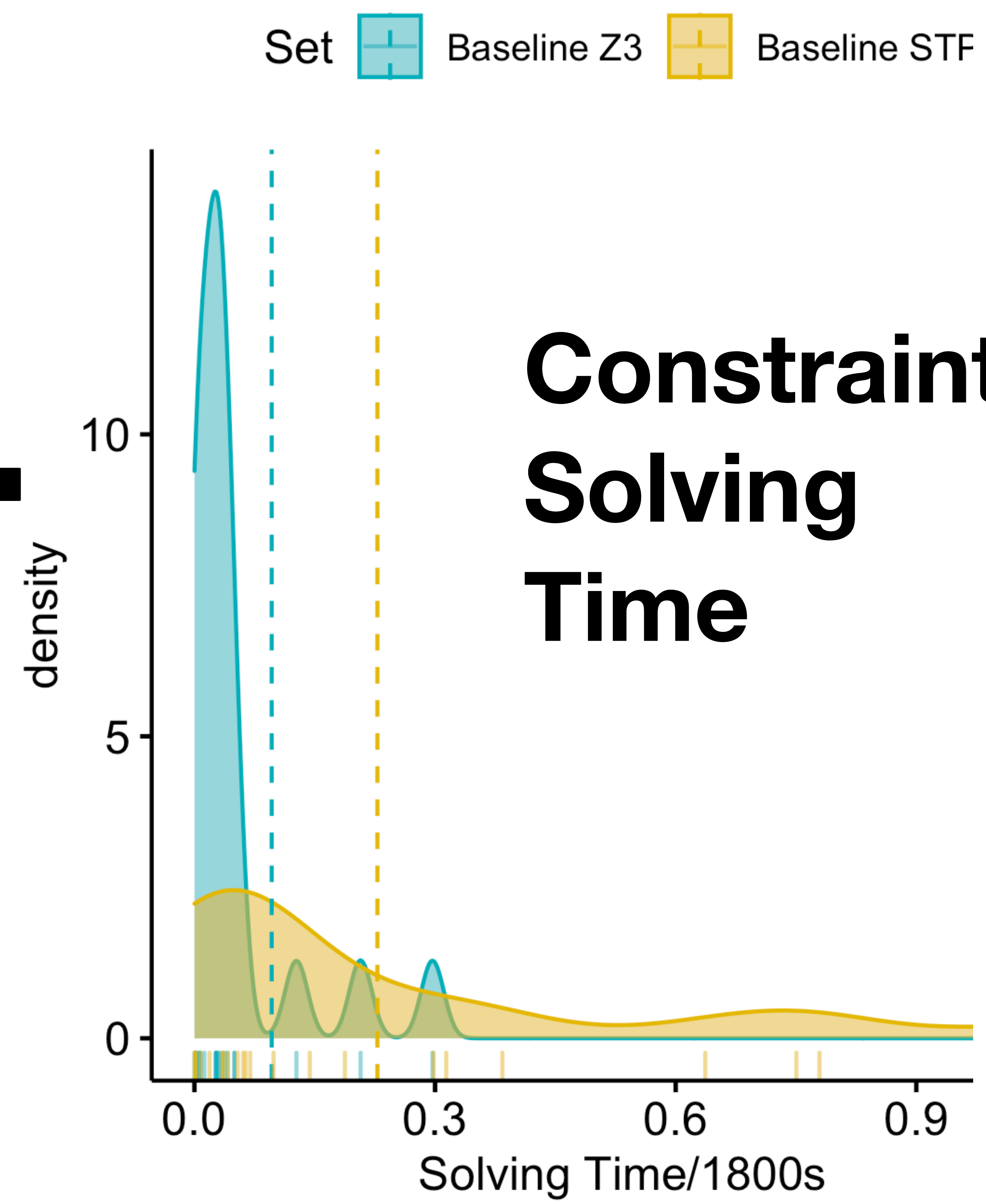


Let’s Go Deterministic

BFS

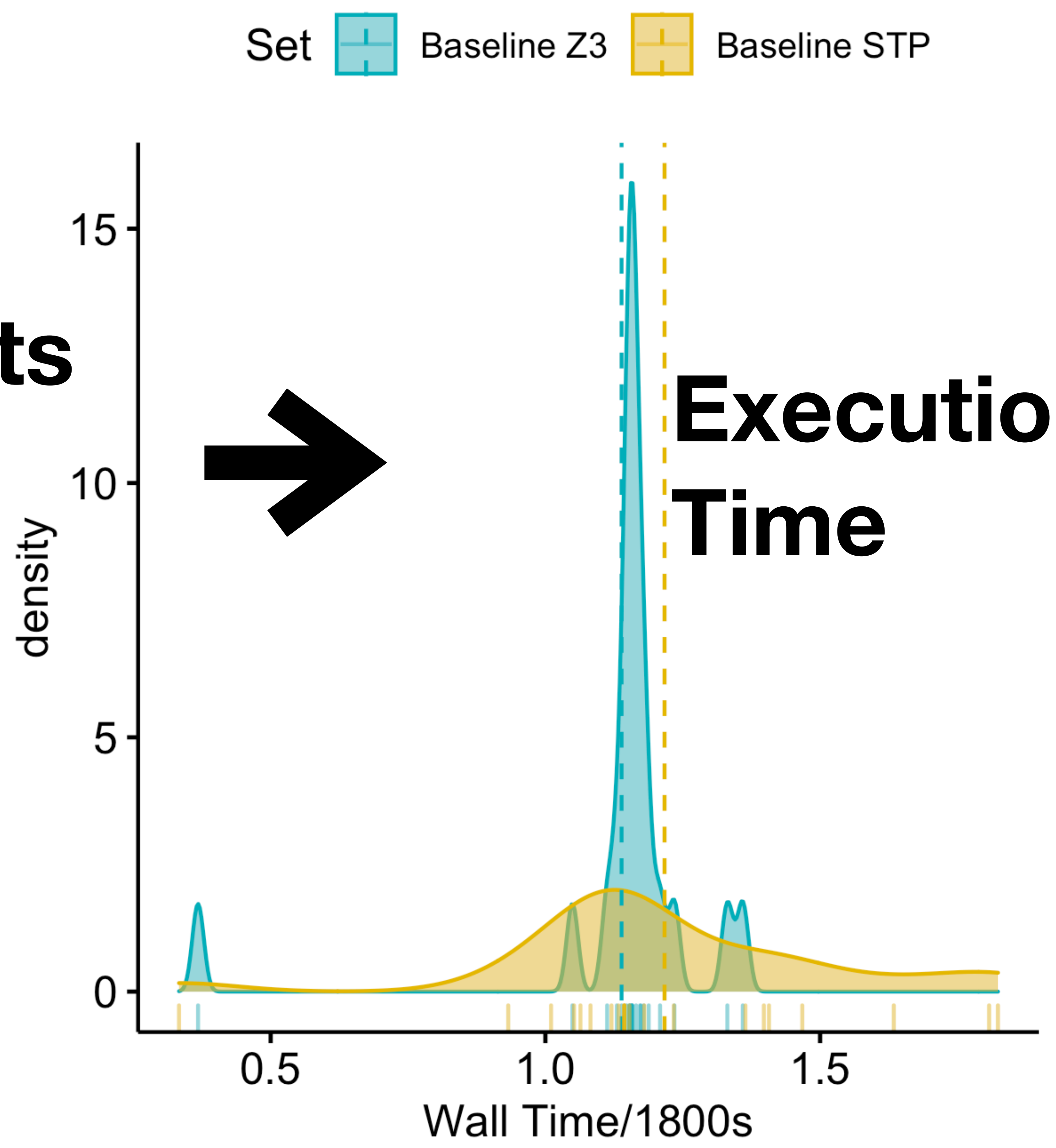
Constraints Generation Time

+



Constraints Solving Time

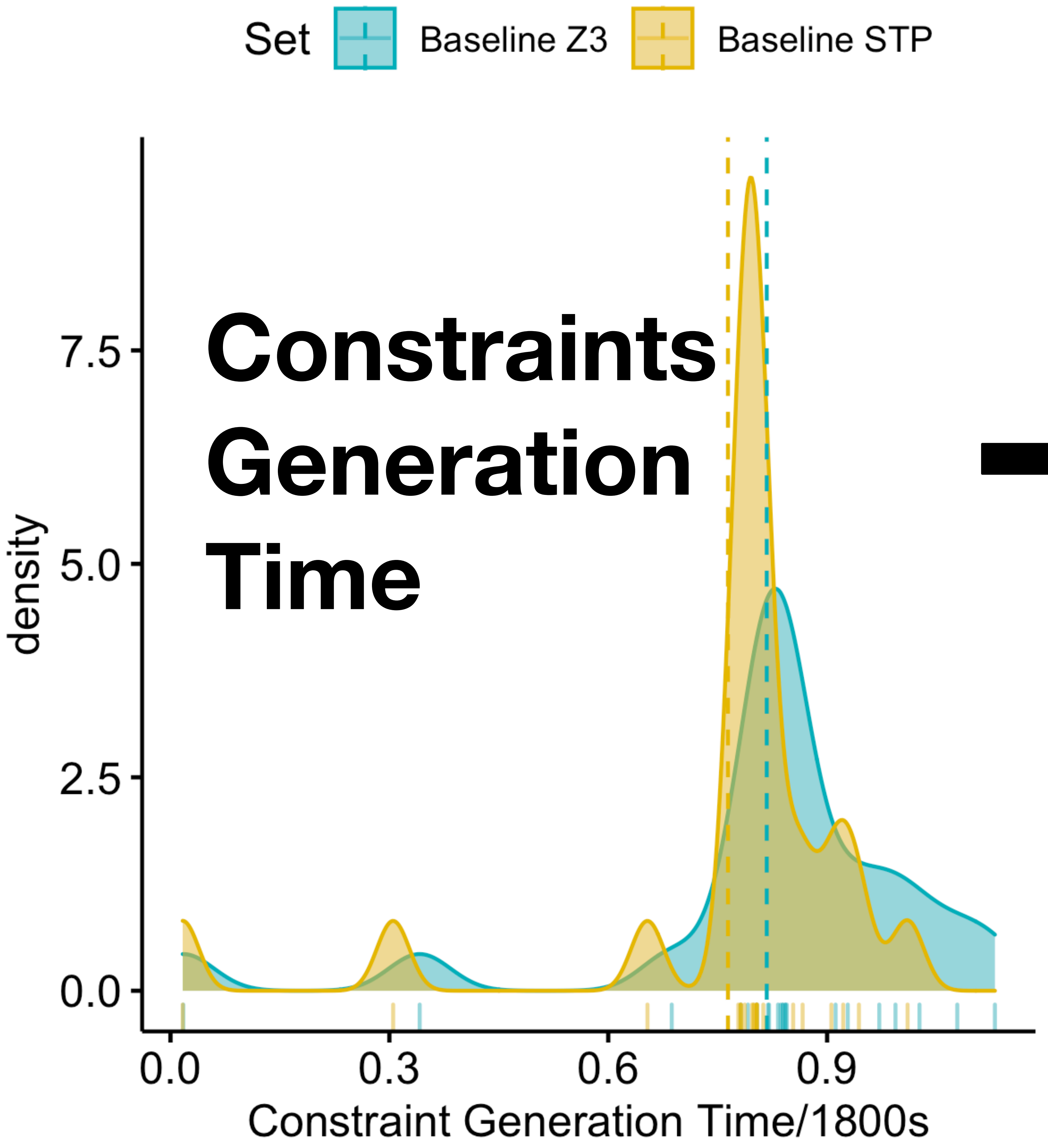
→



Execution Time

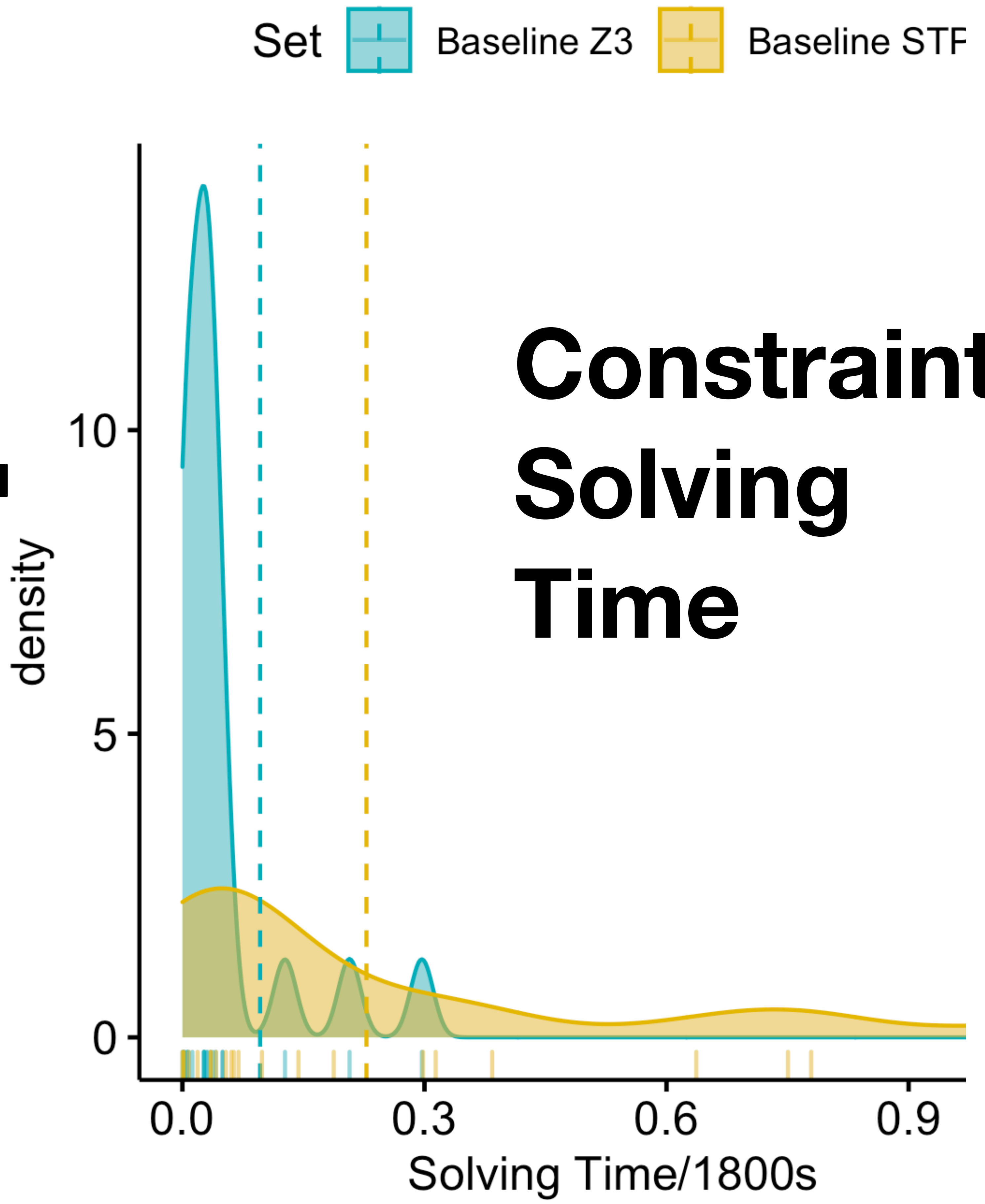
Let’s Go Deterministic

BFS



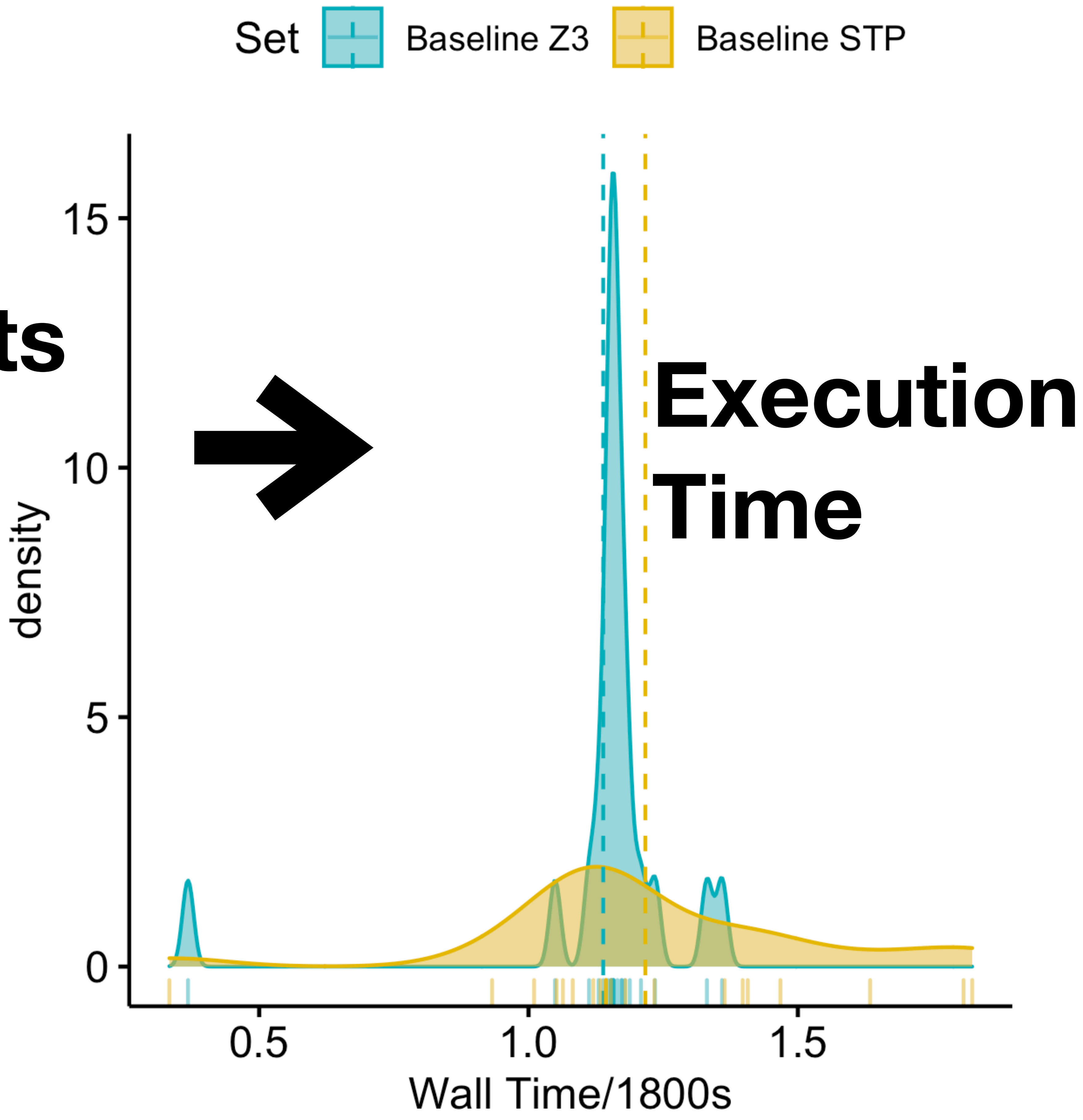
Constraints Generation Time

+



Constraints Solving Time

→



Execution Time

Let’s Go Deterministic

BFS

Constraints Generation Time

+

Constraints Solving Time

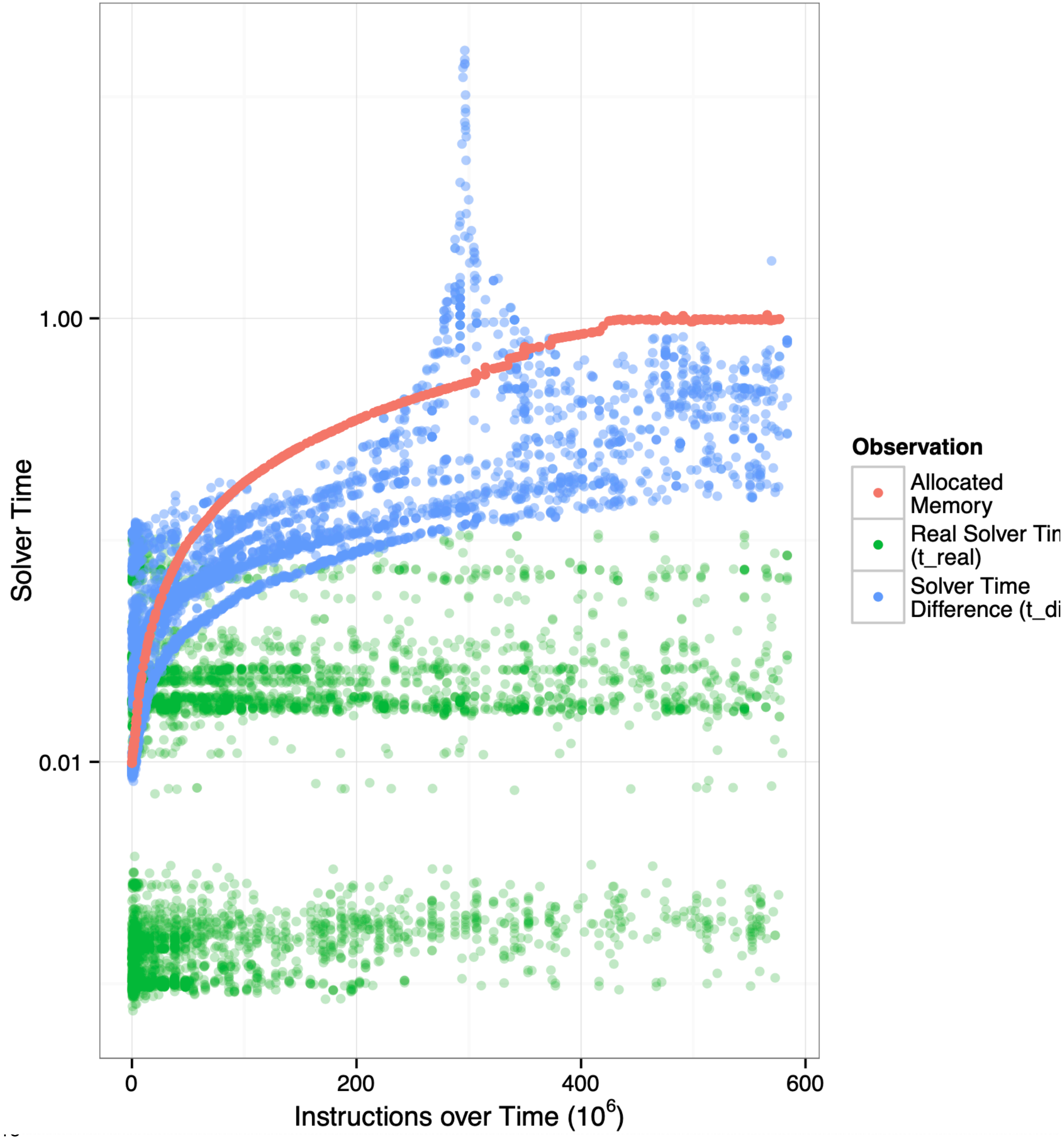
→

Execution Time

Set
Baseline Z3
Baseline STP
density
0.0
2.5
5.0
7.5
0.3
0.6
0.9
Constraint Generation Time/1800s
Set
Baseline Z3
Baseline STF
density
0
5
10
0.0
0.3
0.6
0.9
Solving Time/1800s
Set
Baseline Z3
Baseline STP
density
0
5
10
15
0.5
1.0
1.5
Wall Time/1800s

A `fork()` in the road ...*

- The higher the memory load the longer the system call `fork()` takes
- KLEE `forks()` and execute the solver in a child process for every solver call
- The more memory (i.e., the more states) the longer `fork()` will take



****Baumann et al. “A Fork() in the Road”, HotOS, 2019***

The Fixed Version*

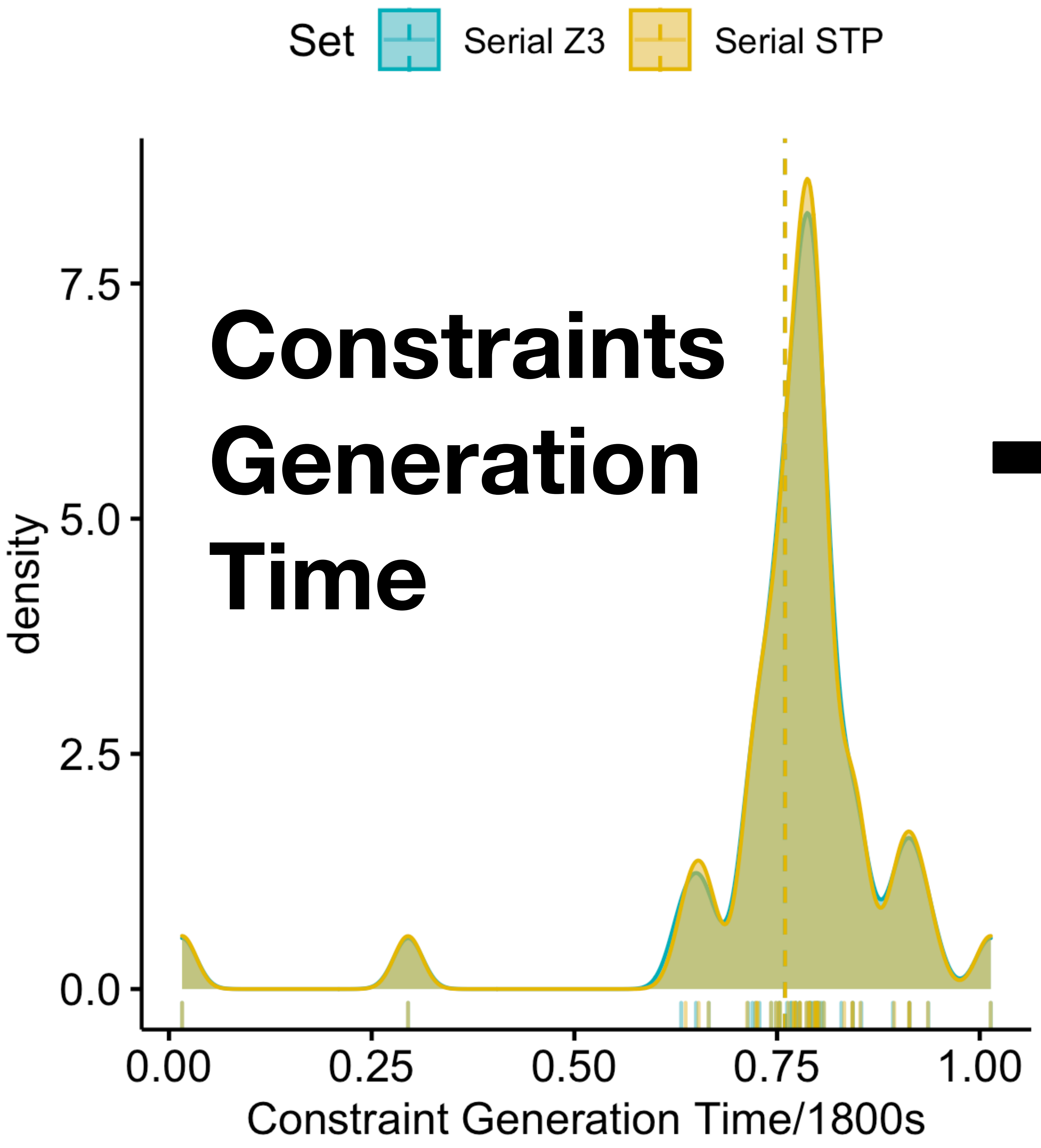
BFS

Constraints Generation Time + Constraints Solving Time → Execution Time

* *Rakadjiev et al, "Parallel SMT Solving and Concurrent Symbolic Execution", TrustCom 2015*

The Fixed Version*

BFS

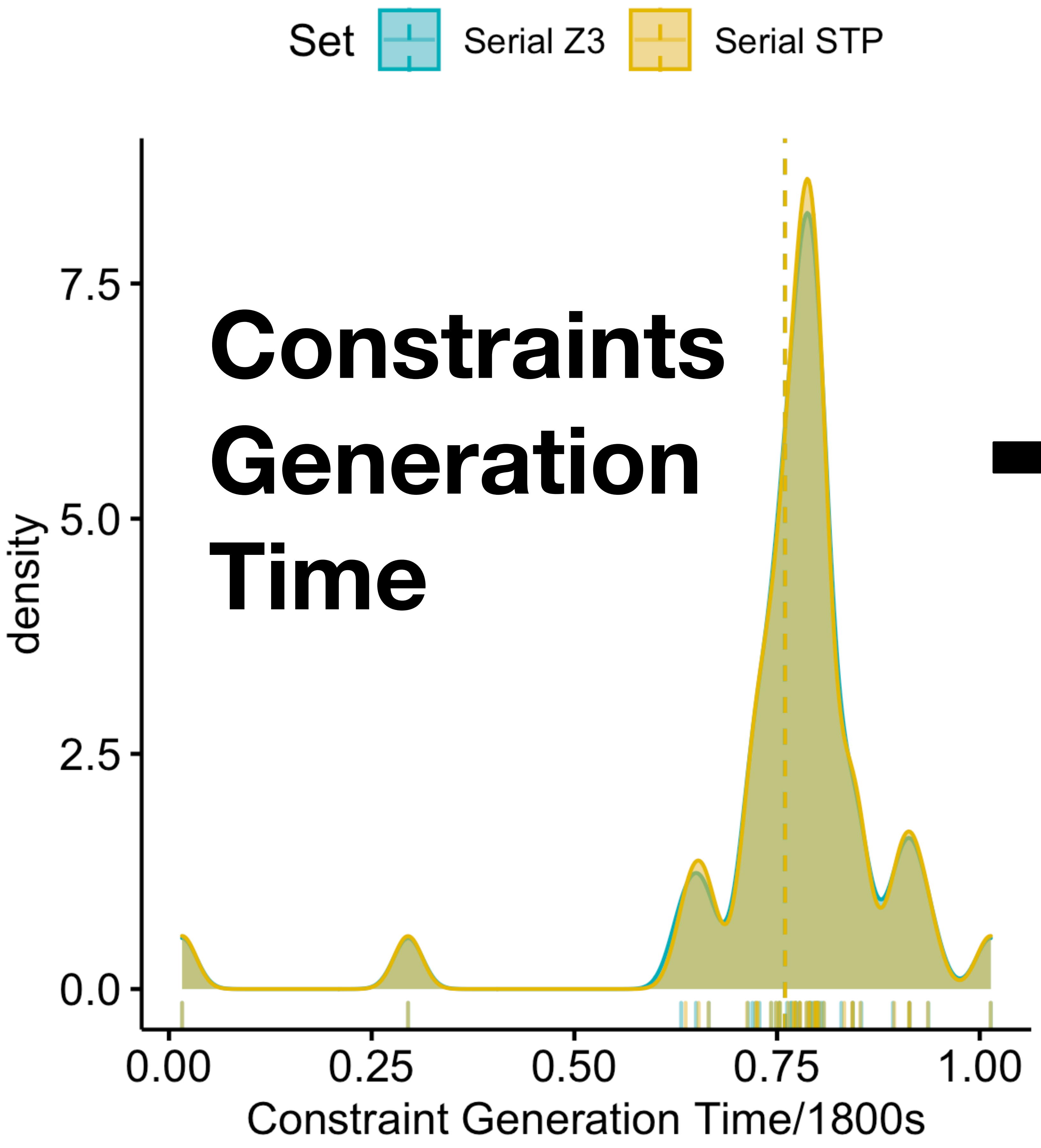


Constraints Generation Time + **Constraints Solving Time** → **Execution Time**

** Rakadjiev et al, "Parallel SMT Solving and Concurrent Symbolic Execution", TrustCom 2015*

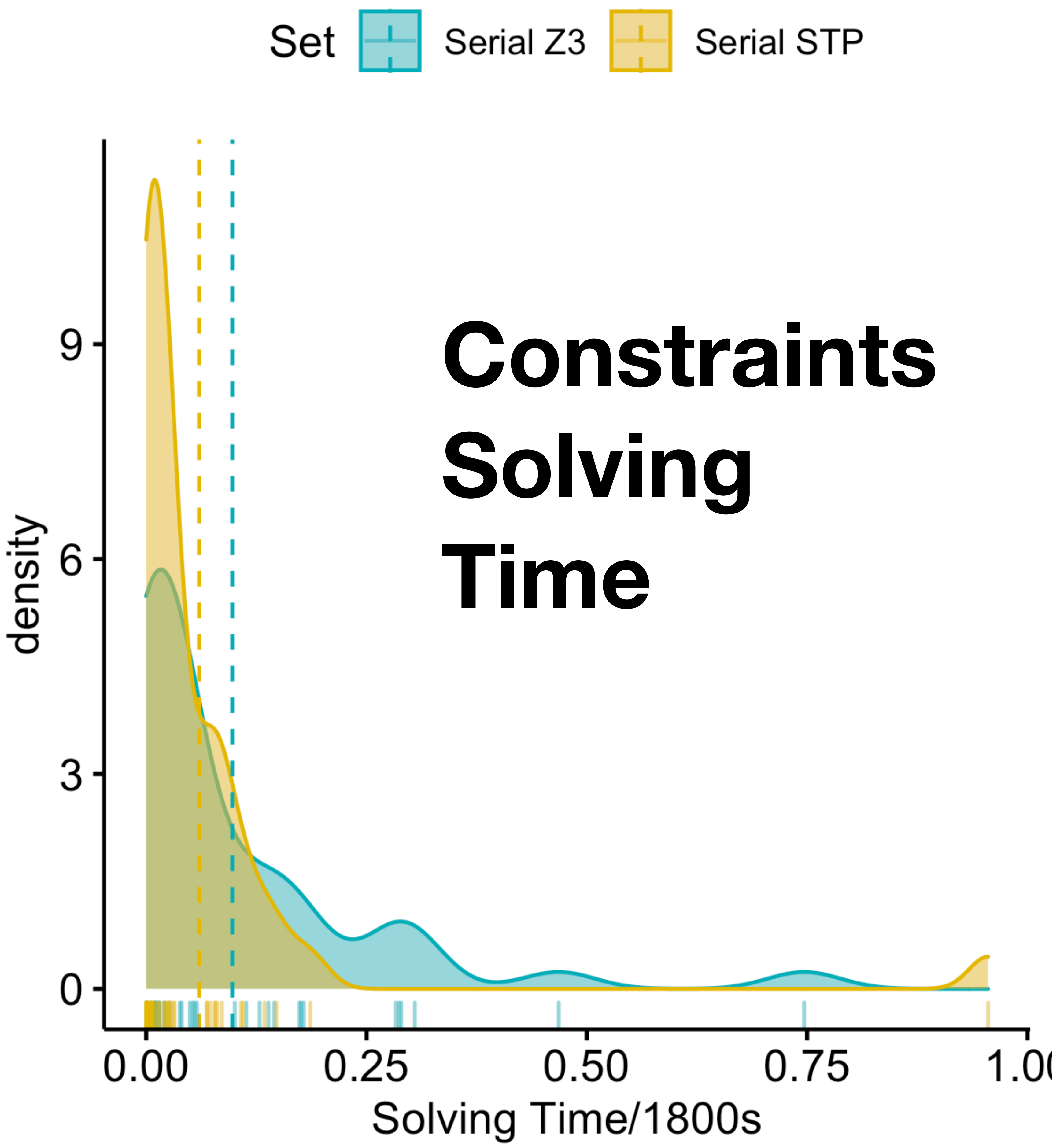
The Fixed Version*

BFS



Constraints Generation Time

+



Constraints Solving Time

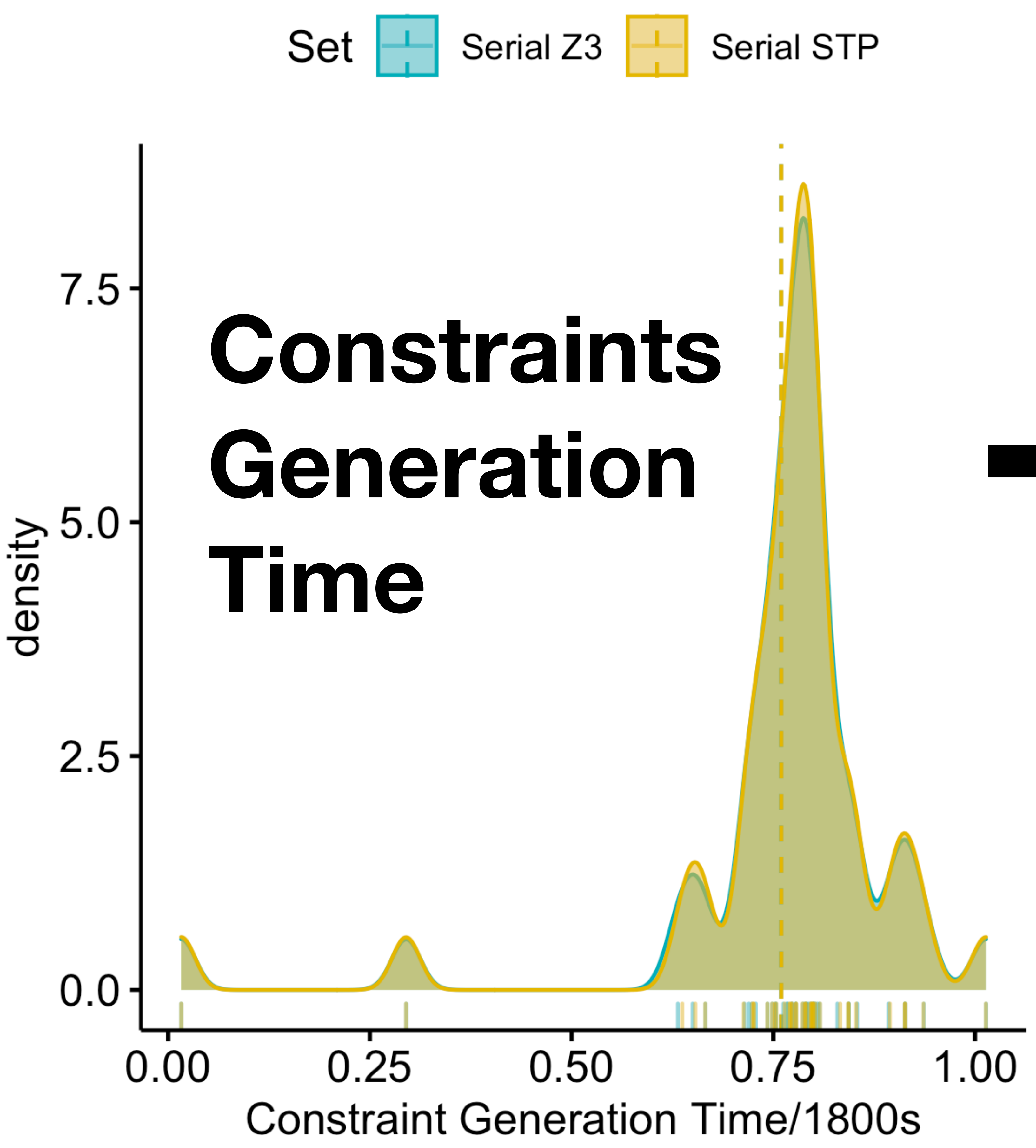
→

Execution Time

** Rakadjiev et al, "Parallel SMT Solving and Concurrent Symbolic Execution", TrustCom 2015*

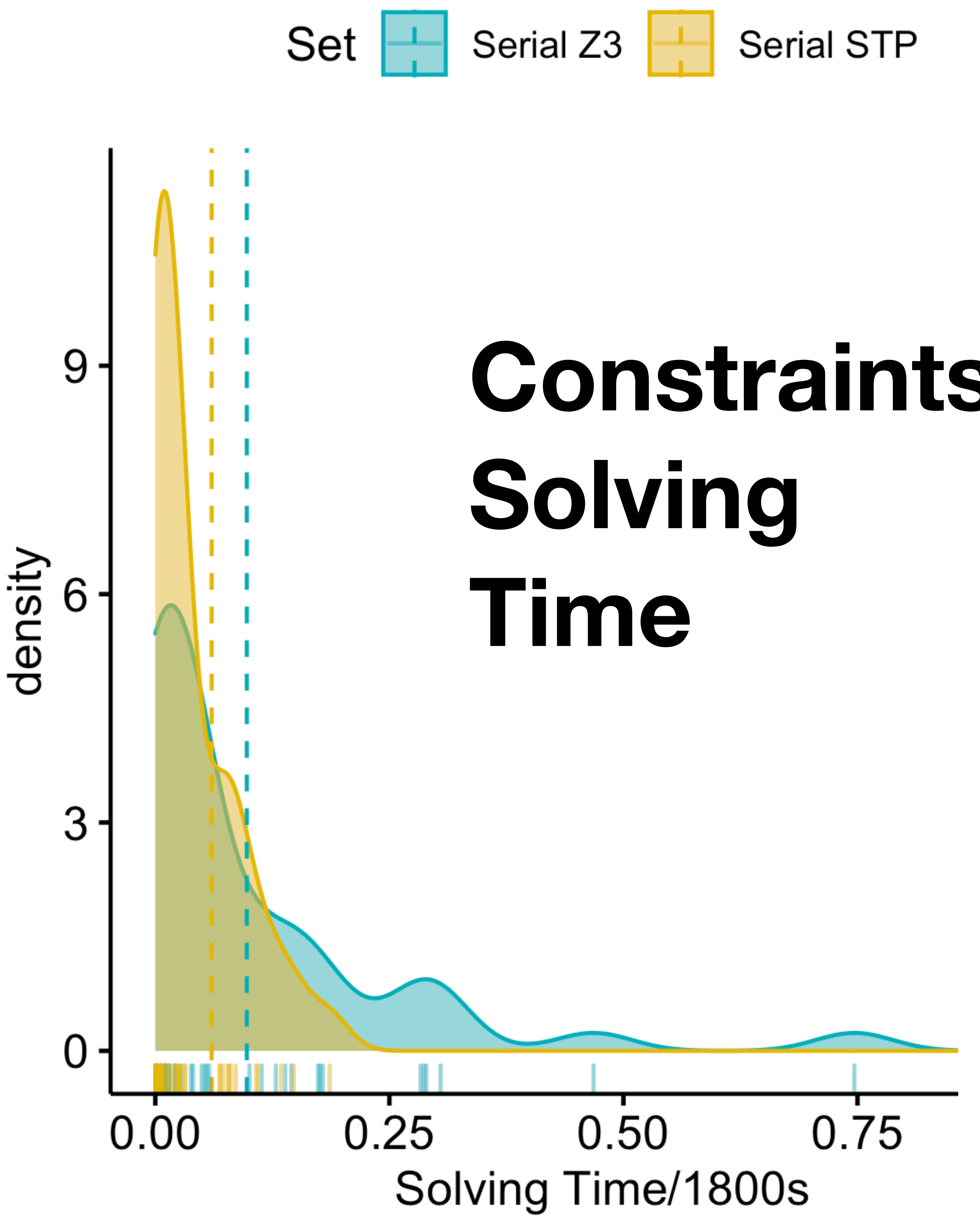
The Fixed Version*

BFS

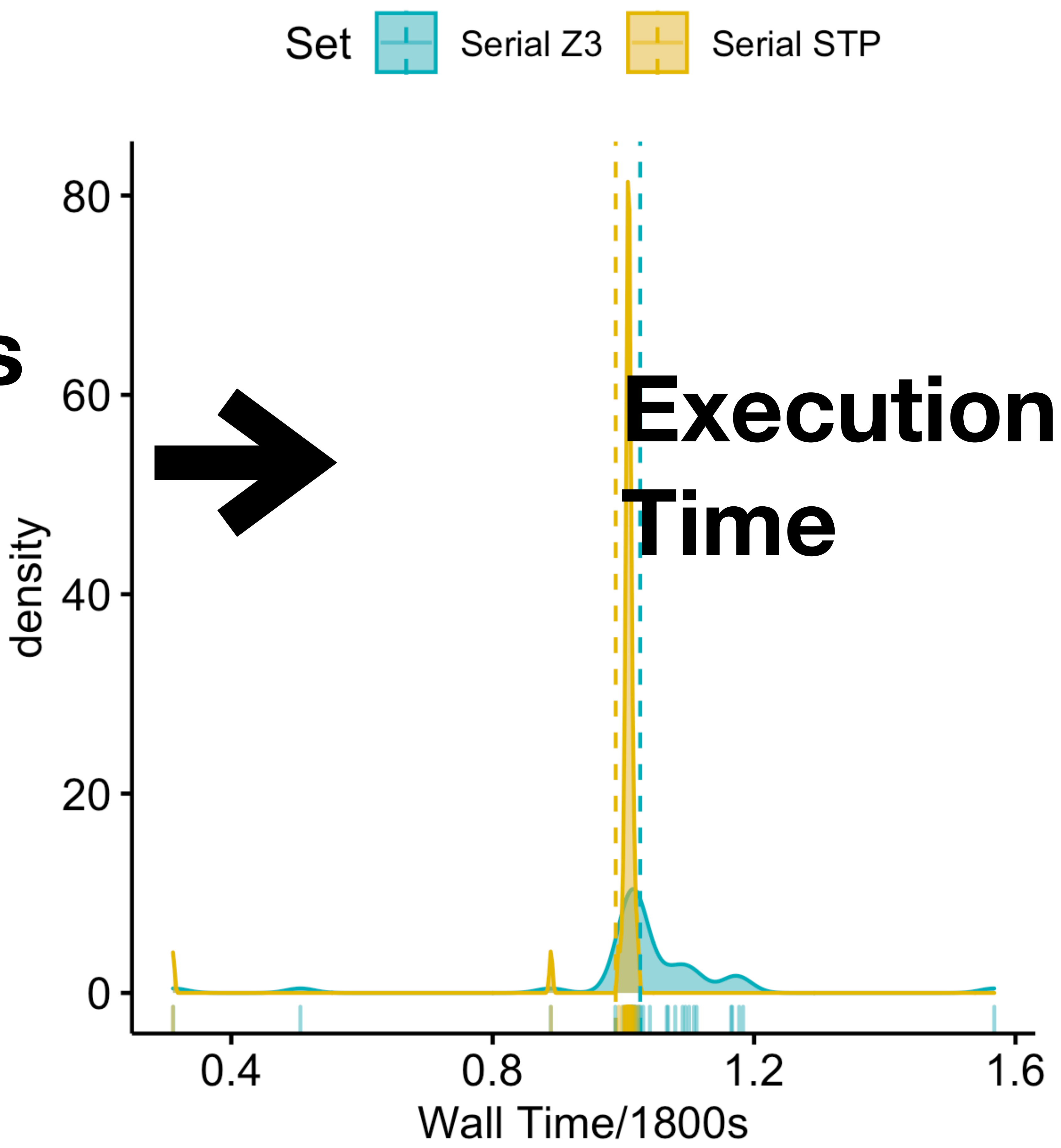


Constraints Generation Time

+



Constraints Solving Time



→ Execution Time

** Rakadjiev et al, "Parallel SMT Solving and Concurrent Symbolic Execution", TrustCom 2015*

How to use Deterministic State Space Exploration?

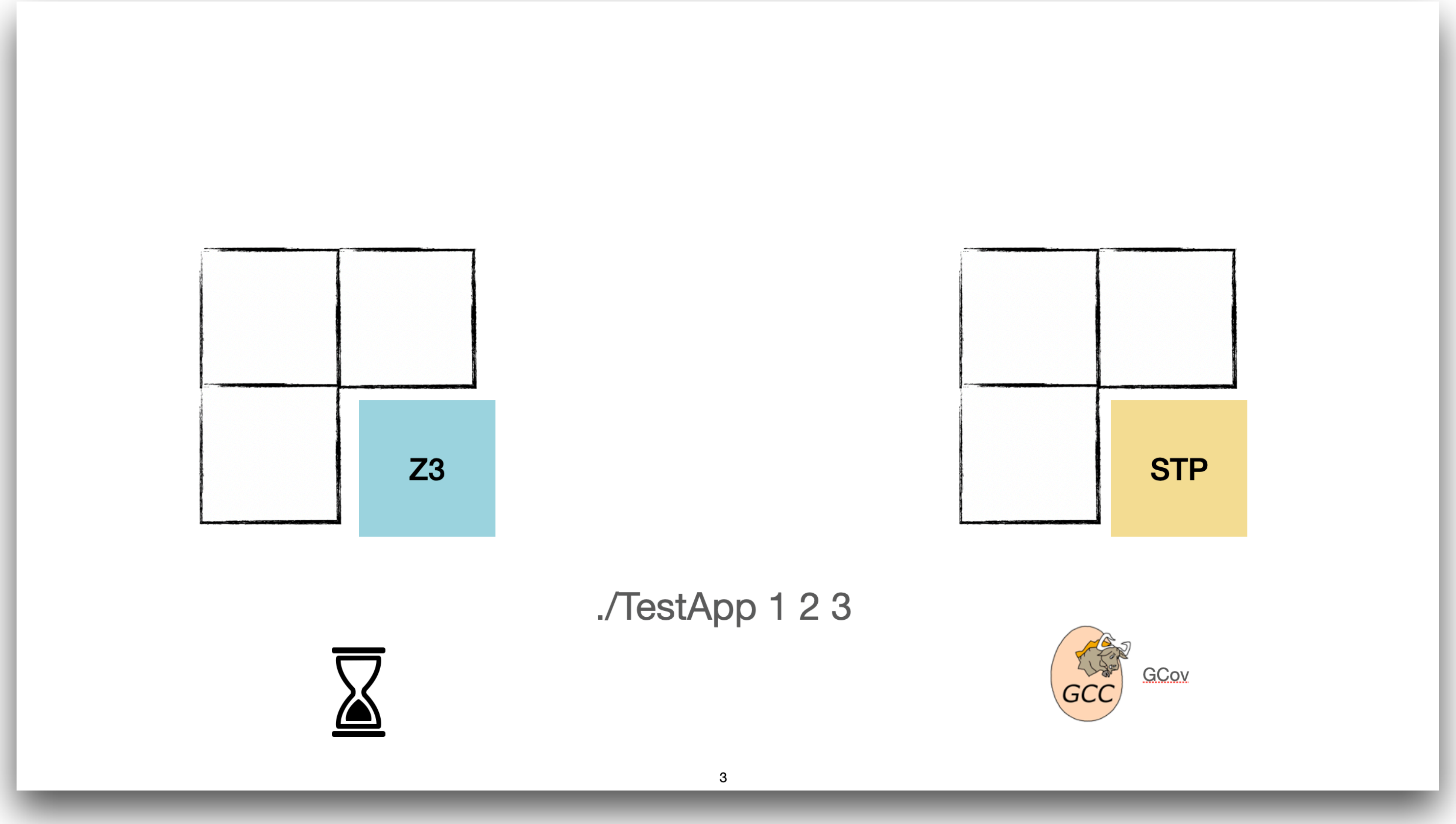
```
--debug-print-instructions=all:file
--debug-compress-instructions
--istats-write-after-instructions=<uint>
```

Deterministic State Space Exploration

$\subseteq$

Fine-Grain Replication of Workload

Evaluate Correctness



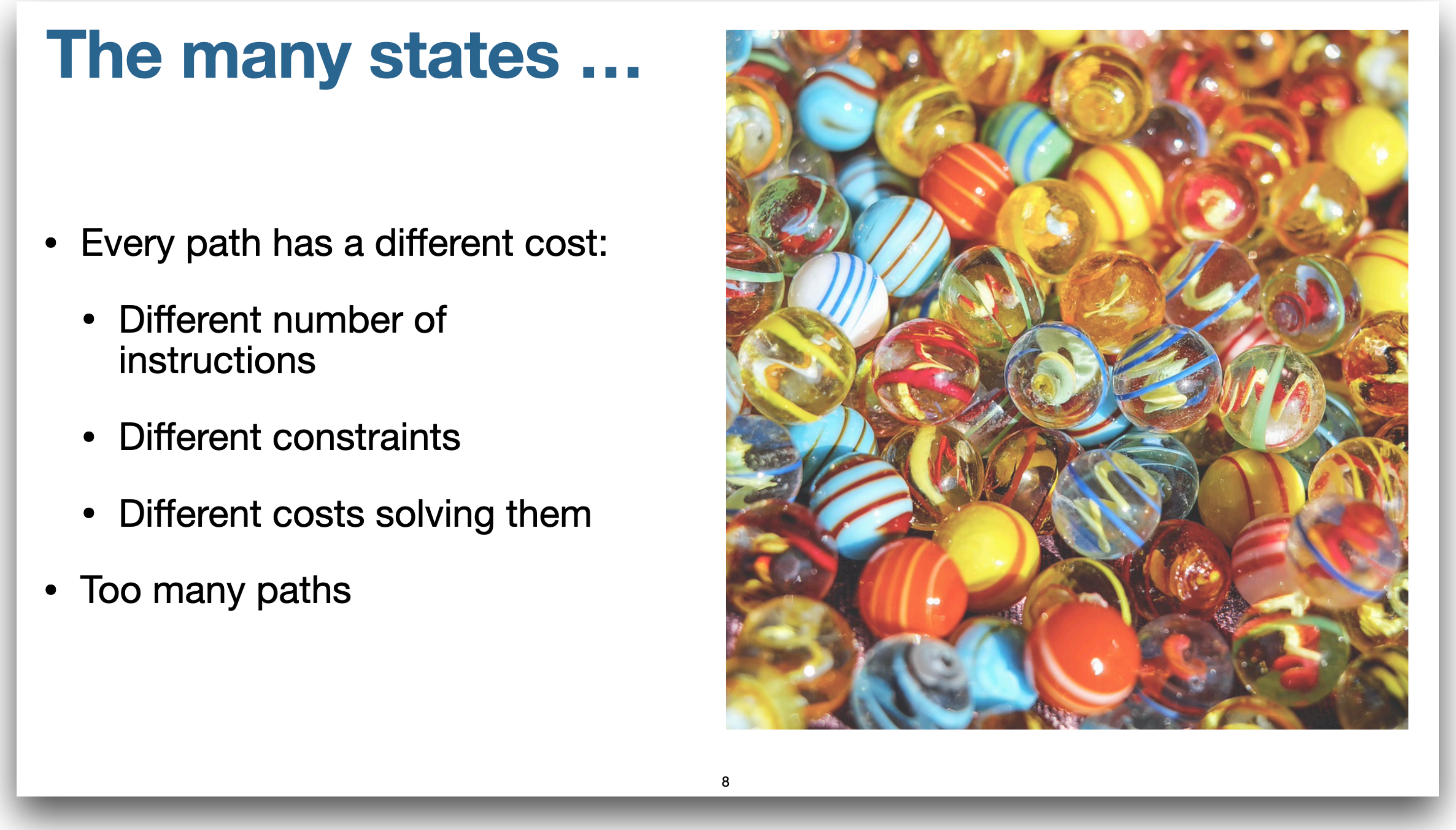
Summary

Deterministic State Space Exploration

$\subseteq$

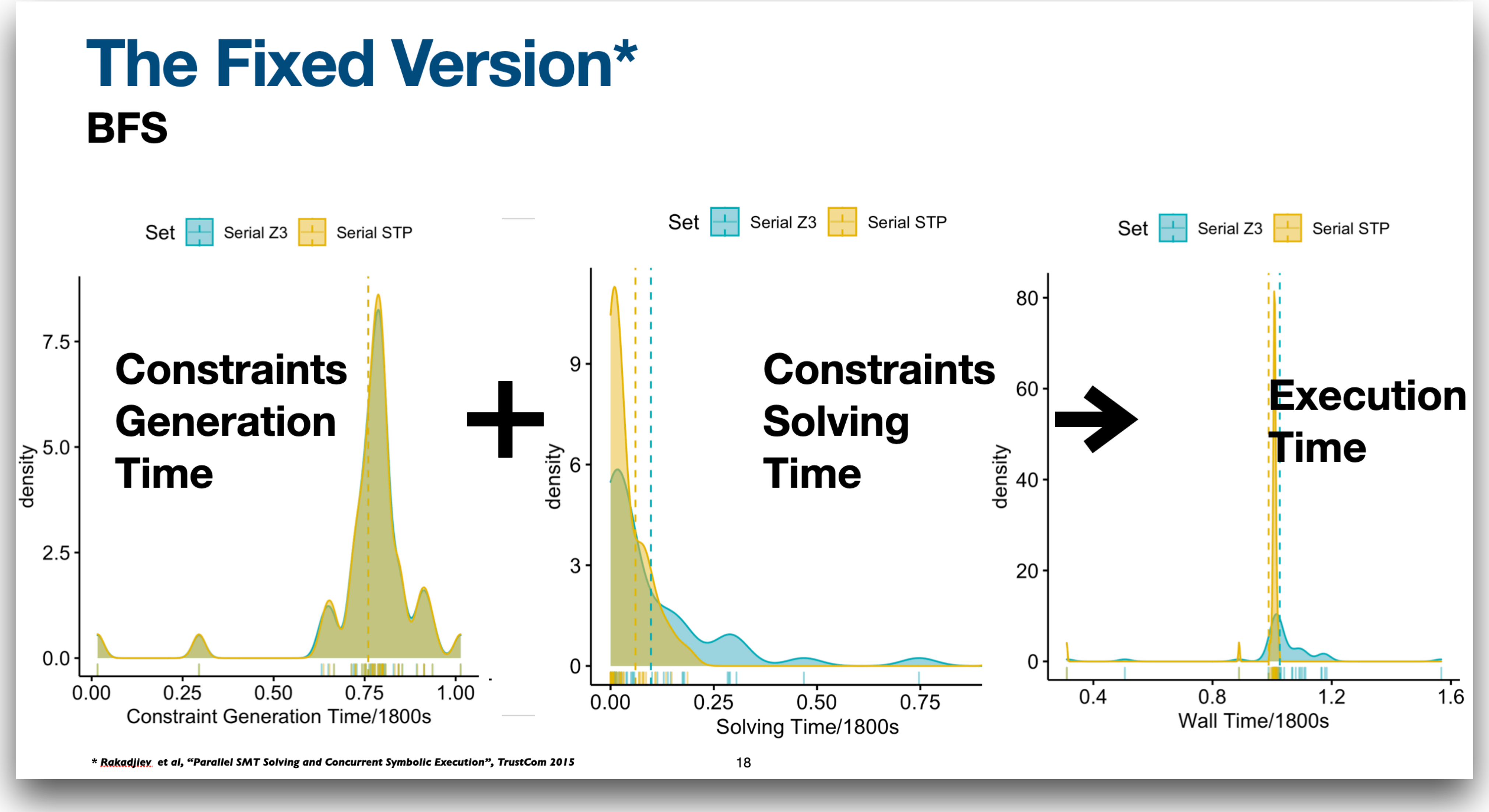
Fine-Grain Replication of Workload

Evaluate Correctness



The many states …

- Every path has a different cost:
  - Different number of instructions
  - Different constraints
  - Different costs solving them
- Too many paths



The Fixed Version*

BFS

Constraints Generation Time + **Constraints Solving Time** → **Execution Time**

* Rakadjiev et al, "Parallel SMT Solving and Concurrent Symbolic Execution", TrustCom 2015