Parallel and Selective Symbolic Execution

George Candea and Stefan Bucur
School of Computer & Communication Sciences

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

We take software reliability on faith.

010011010011
110101001010
010010000001
010111010011
110110010110
011011000011

RTL8029.SYS


Android SMS bug sends your messages to random c tacts
December 31, 201 :23am PST
A bug in GMail erases thousands of accounts
MARCH 1, 2011
Surprise! Your iPhone Is Tracking Your Every Move
April 20, 2011
Skype crash: Software bug and server overloads blamed
30 December 20
Hotmail bug means m efox 4 betas
February 3, 201
Google Docs Users Stranded by Bug
Fri 29th Oct
Skype bug gives attackers access to Mac OS X machines
'Extremely wormable and dangerous'
6th May 2011 19:40 GMT
Internet Explorer bug puts 900 million users at risk
31 Jan 2011 14:36

Rigorous Approaches ?

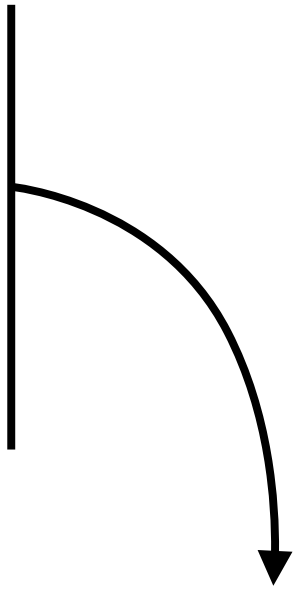
- Solid techniques for software construction
  - *special languages, model checkers, theorem provers, ...*
  - *static analysis, dynamic analysis, runtime verification, ...*
- But software systems are messy
  - *scale & economics hinder the use of rigorous approaches*

The Goal

Existing programming languages

Existing development processes

Tools to understand complex software



Significantly more reliable software
that is economically feasible

Outline

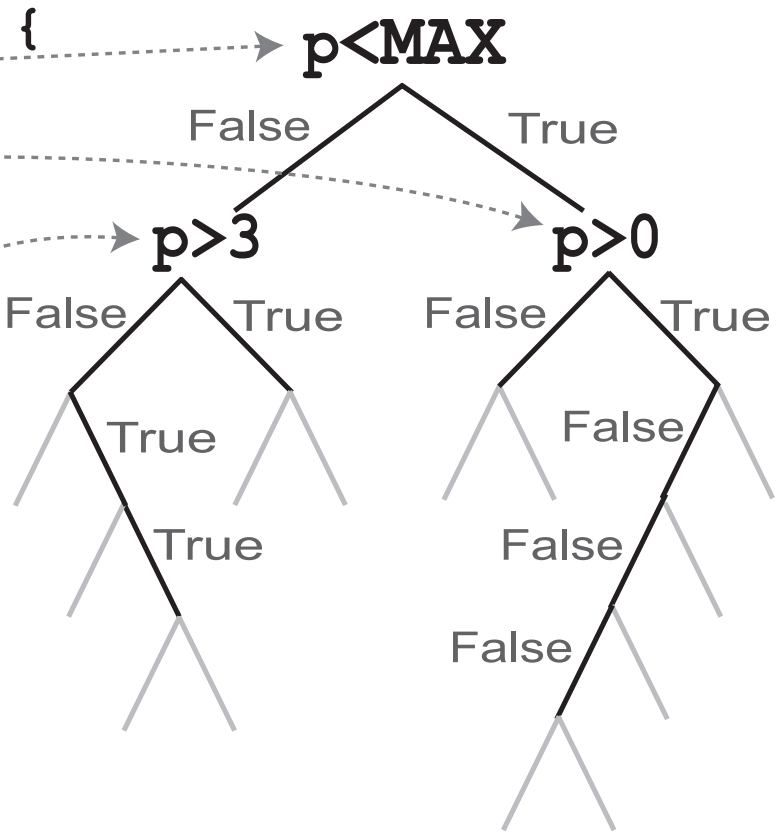
→ Cloud9

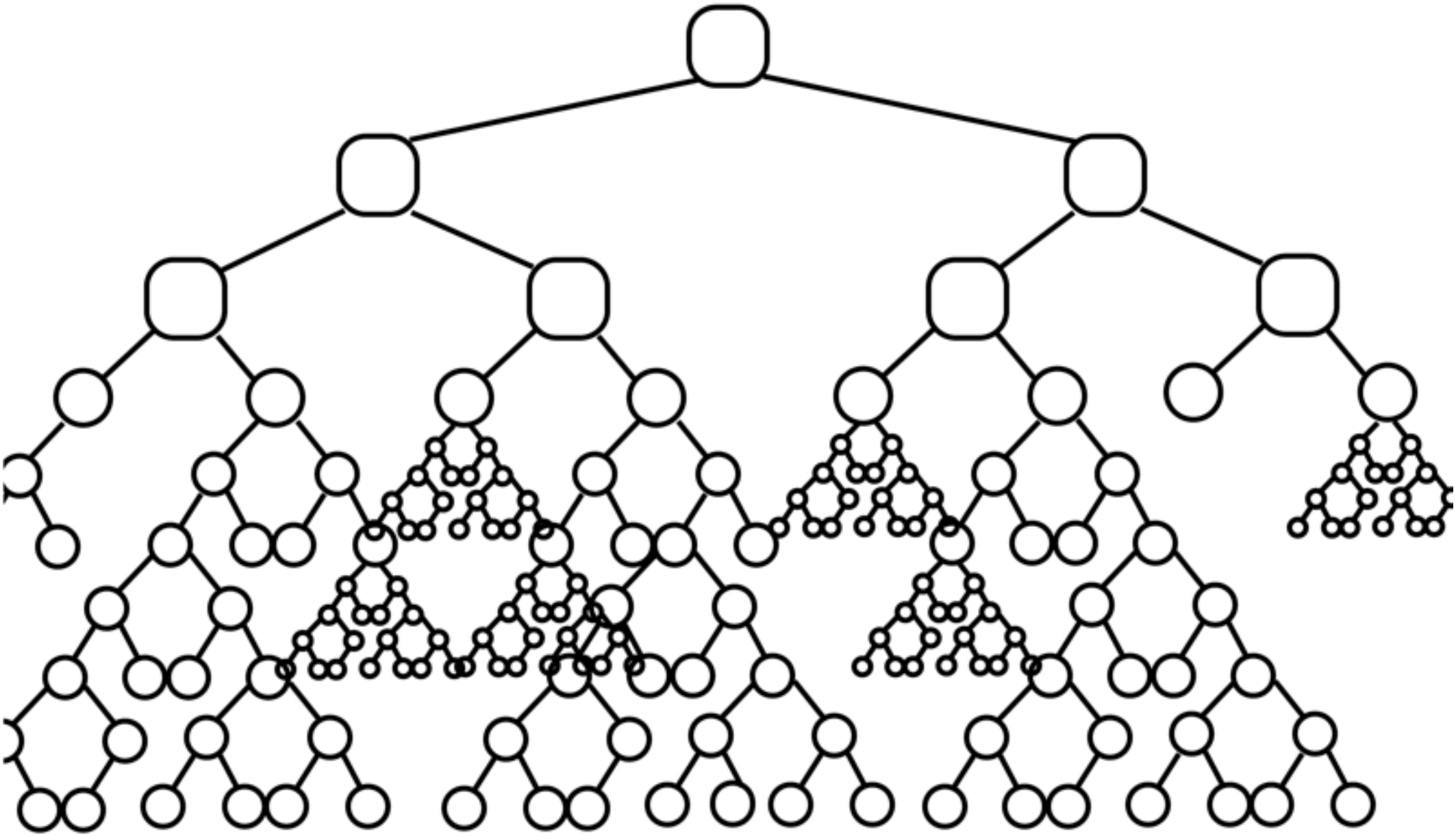
- *Parallel symbolic execution platform*

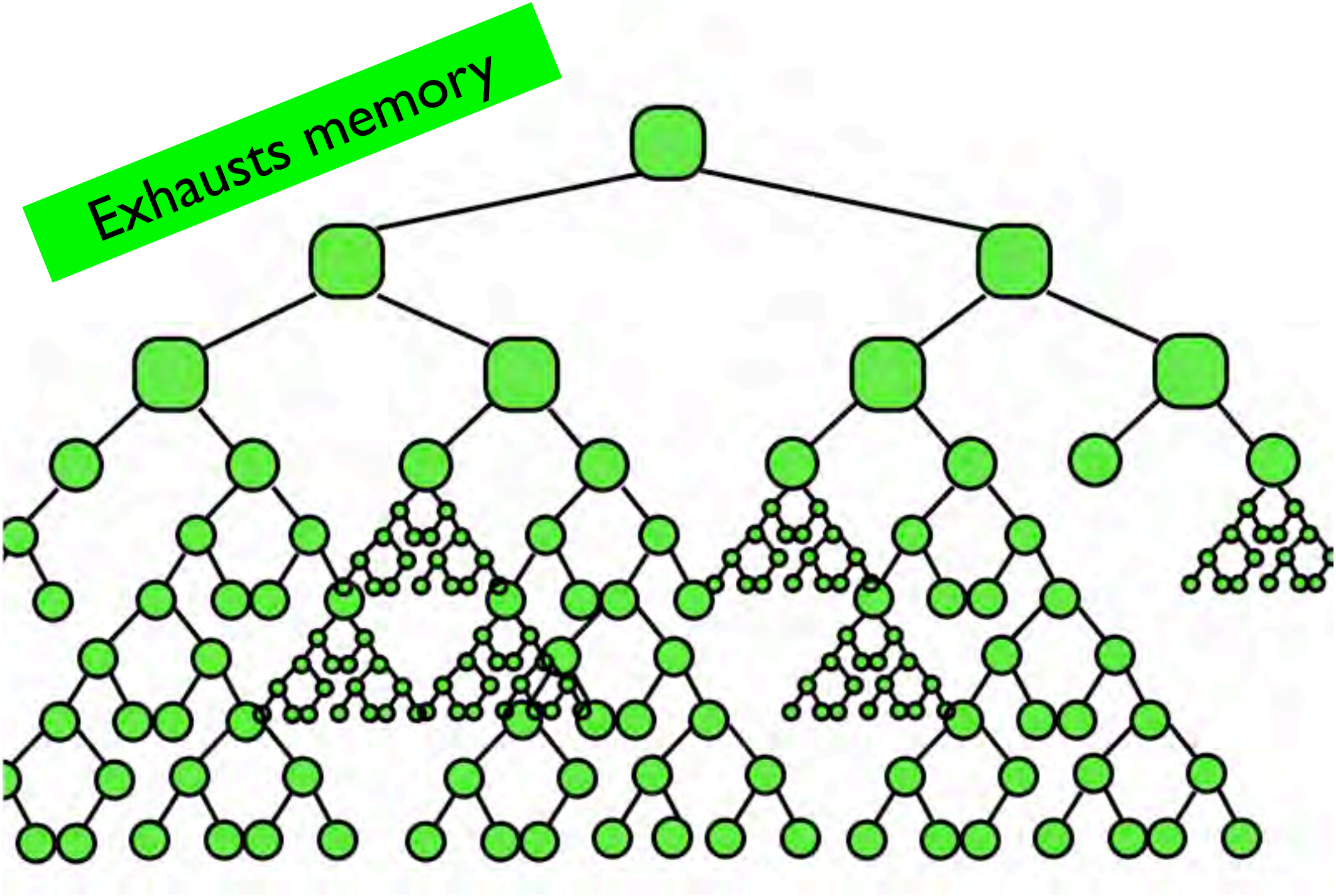
- S^2E
  - *Selective symbolic execution platform*
- Lessons for SMT/SAT Solvers
  - *Real data from running on real systems*

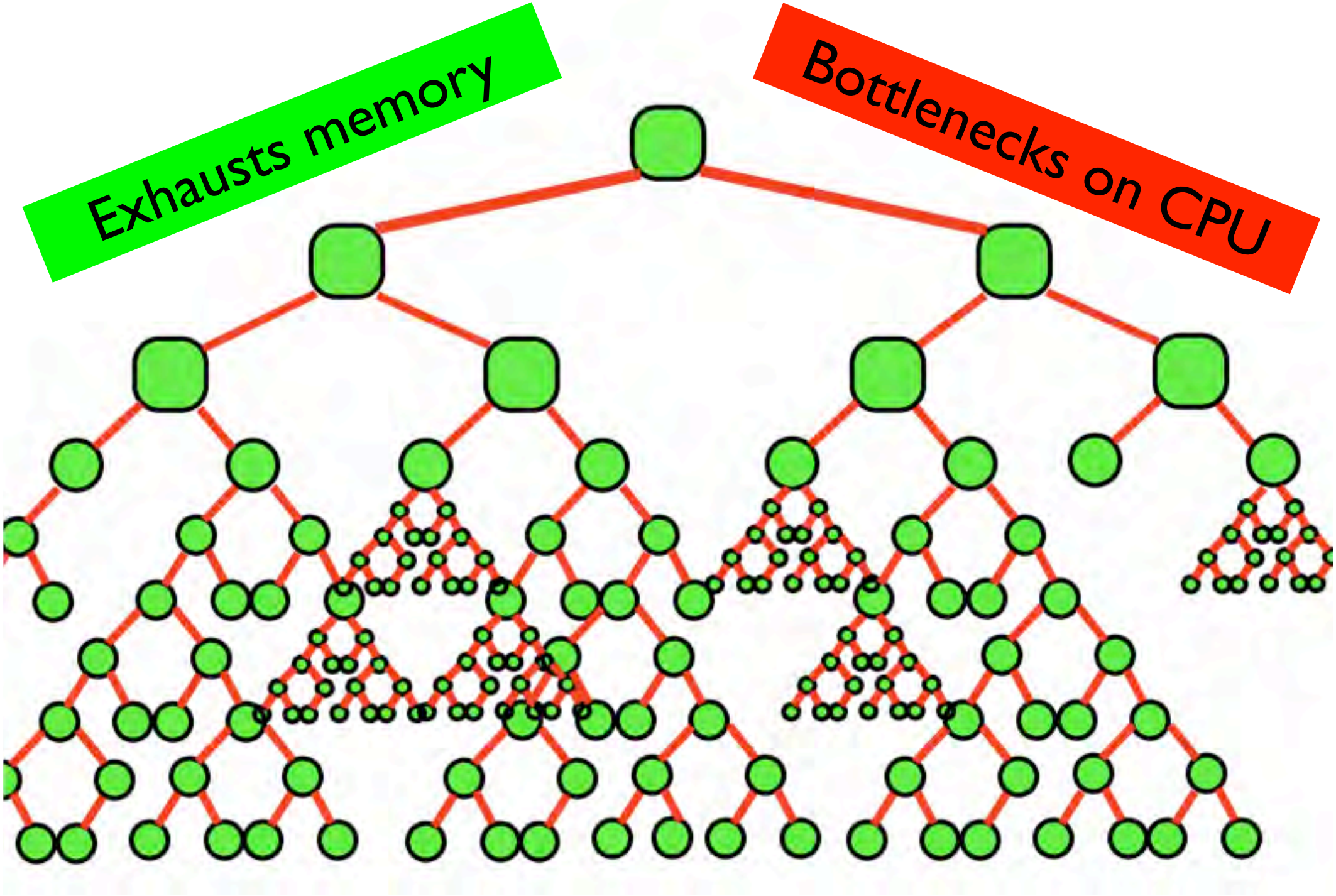
Symbolic Execution

```
void write( int p ){
  if (p < MAX) {
    if (p > 0)
      ...
    else {
      ...
    }
  } else {
    if (p > 3)
      close(p);
    else {
      ...
  ...
```



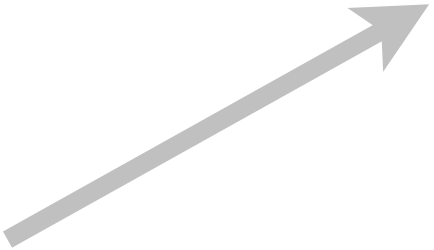



Exhausts memory


Exhausts memory
Bottlenecks on CPU

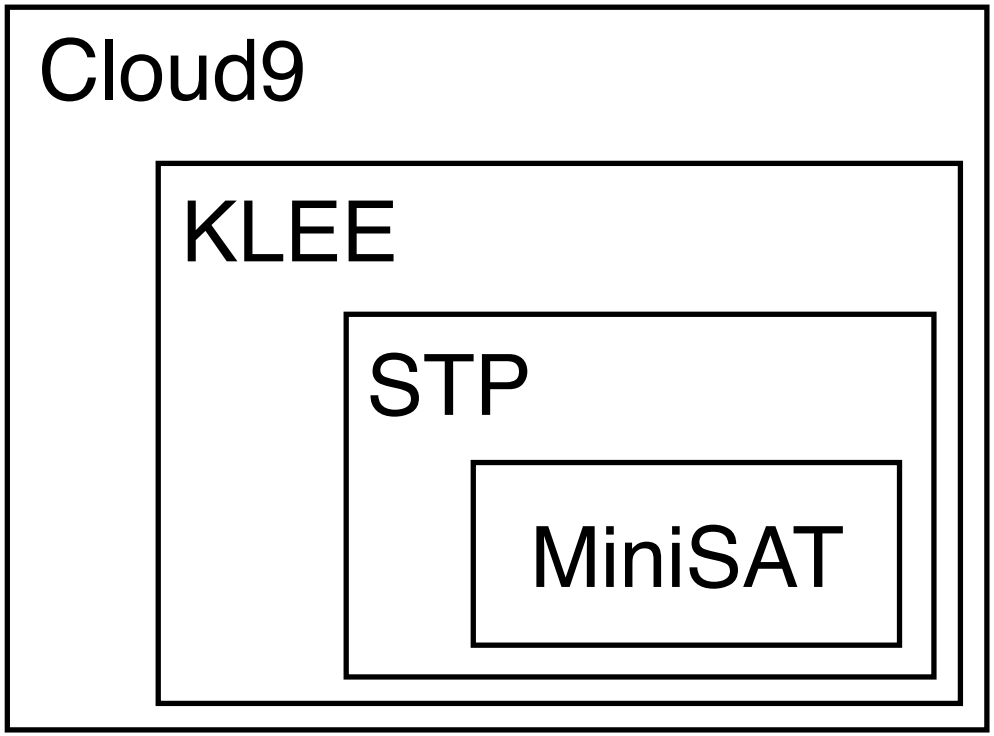
Apache

Memcached



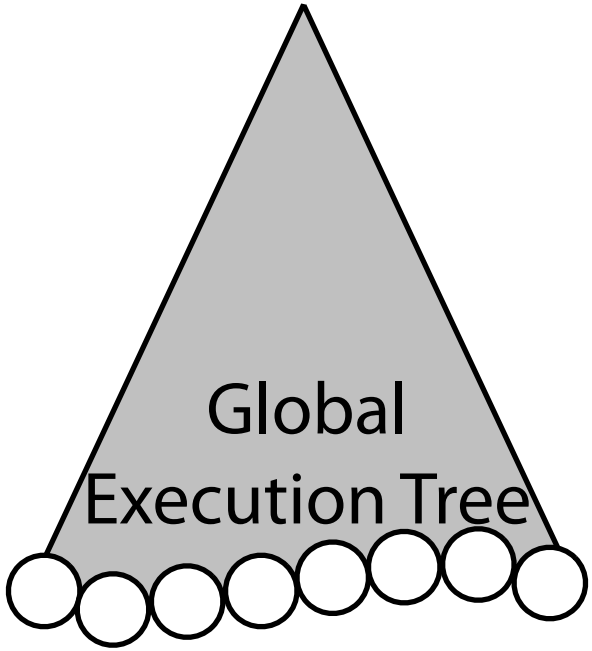
GNU Coreutils

Build
a Google
of symbolic execution.

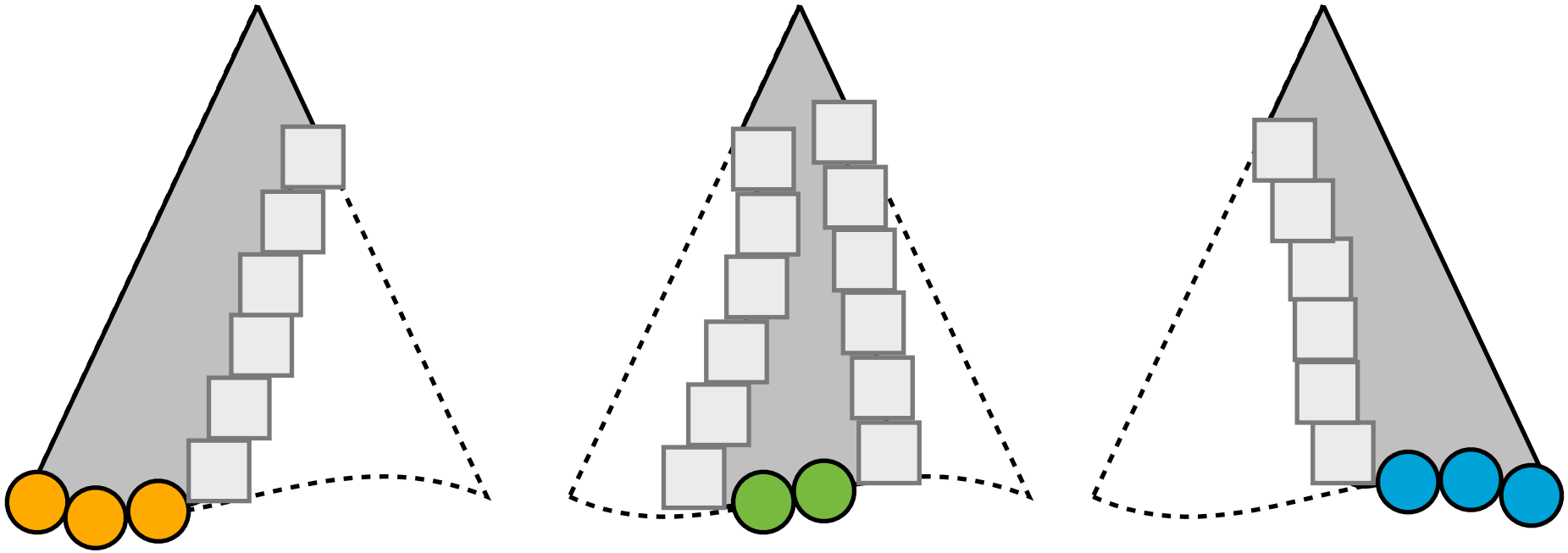


1. Parallel symbolic execution
2. Multithreaded support
3. Full model of POSIX environment
4. API for writing symbolic tests

Cloud9



Cloud9



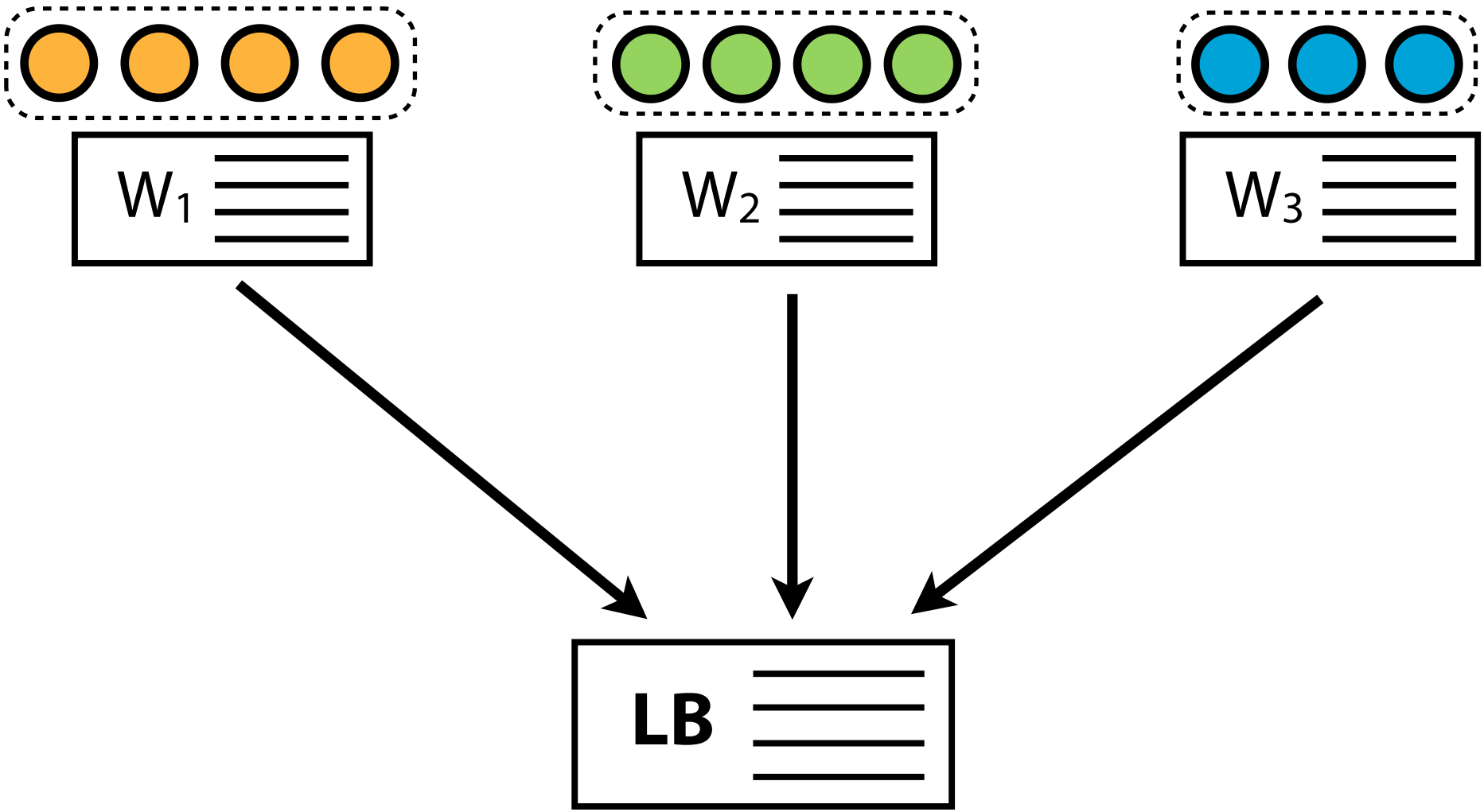
W_1's local tree (local KLEE)

W_2's local tree (local KLEE)

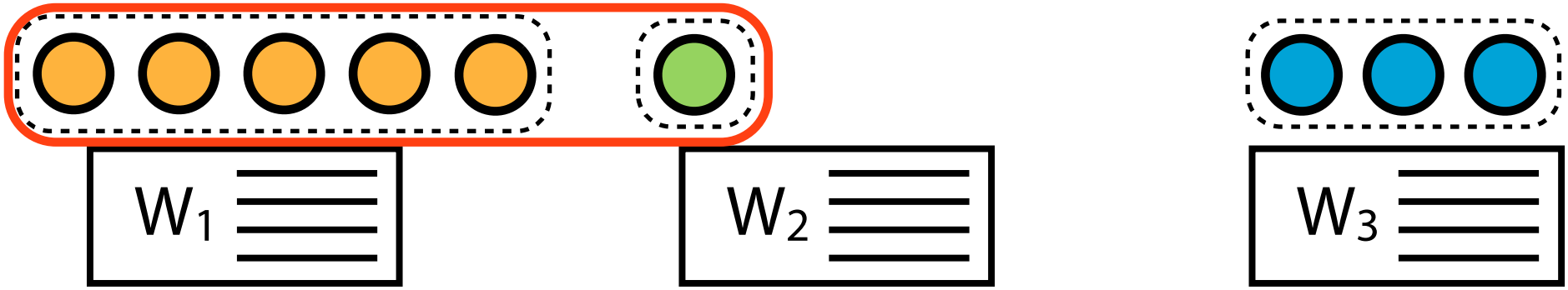
W_3's local tree (local KLEE)

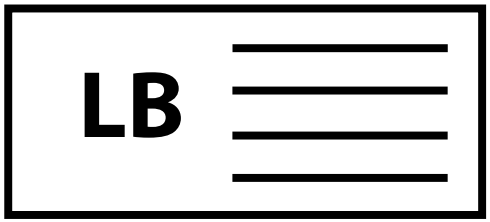
Disjoint + complete exploration of exec tree

Dynamic Parallelization

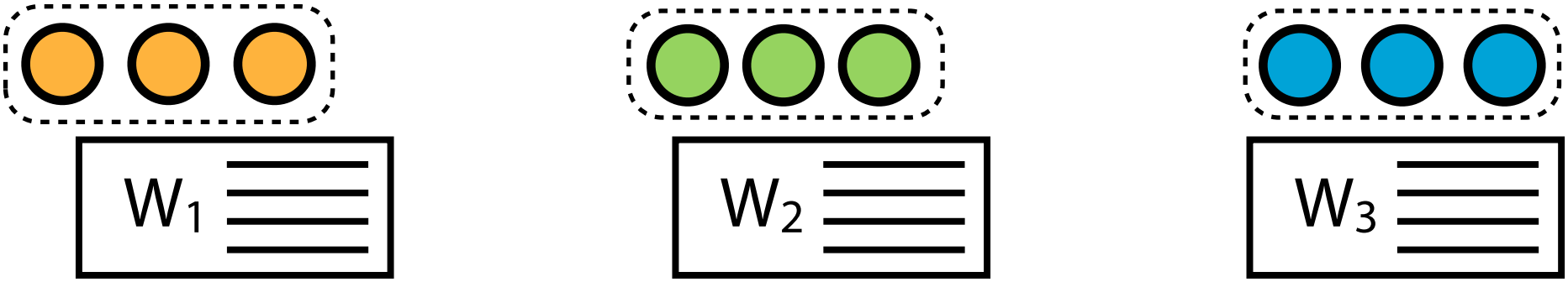


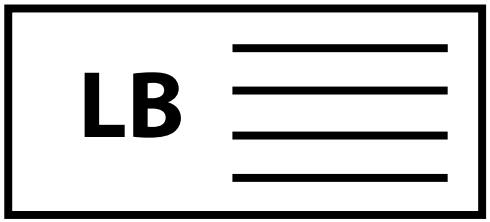
Dynamic Parallelization



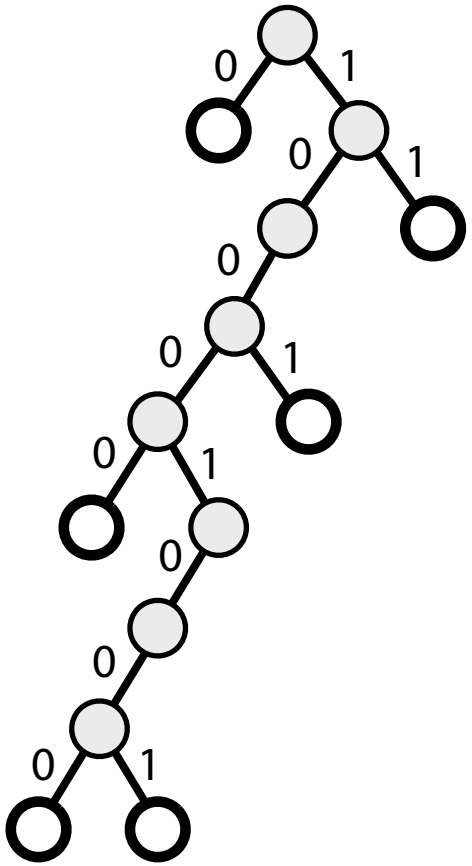


Dynamic Parallelization



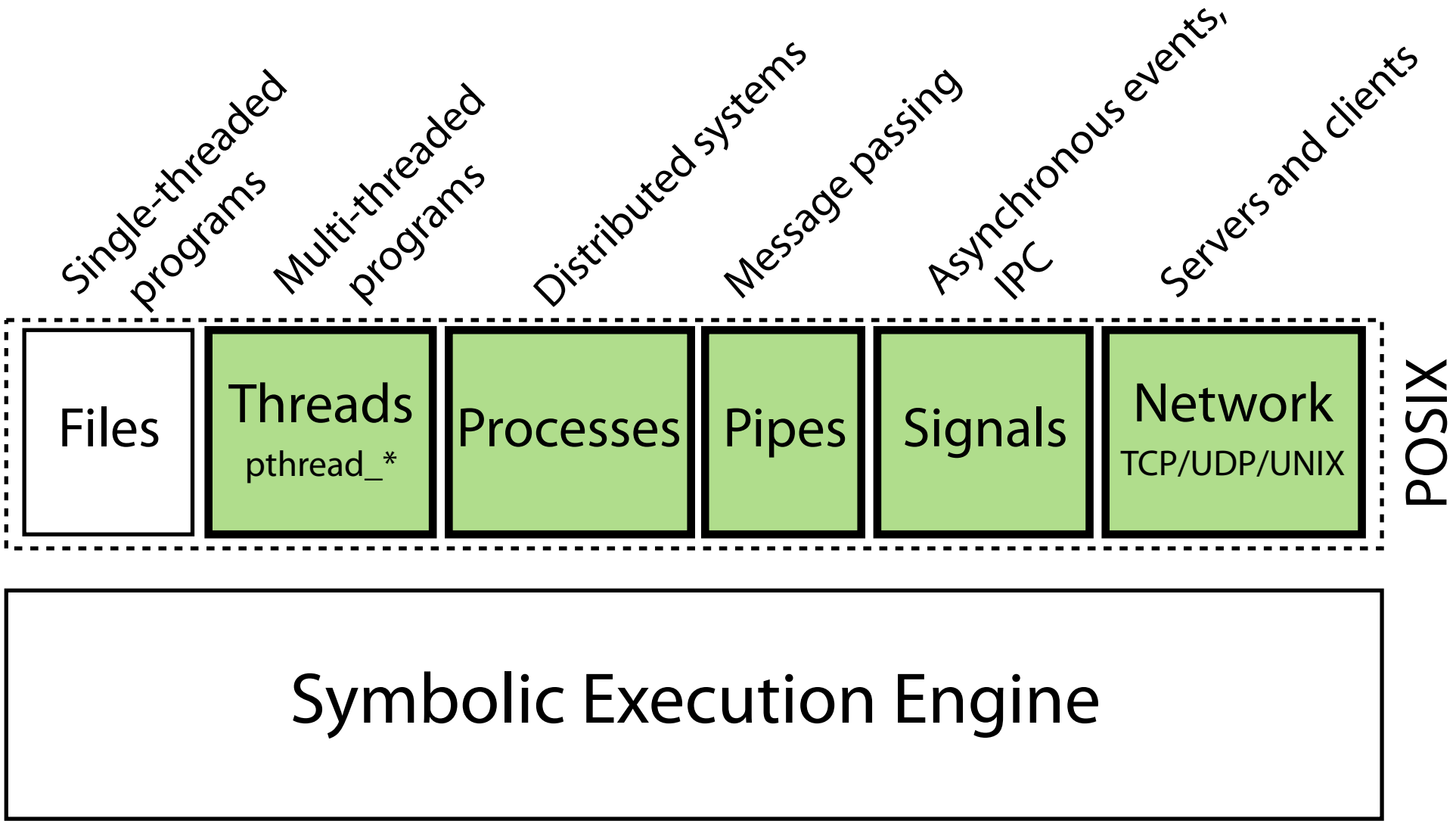


Path-based State Encoding



- Paths in tree = bitvectors
  - *Multiple paths share common prefix → tree of bitvectors*
- Compact encoding
  - 2^{100} *leaves →* <128 *bits*
- Requires deterministic replay

POSIX Environment Model



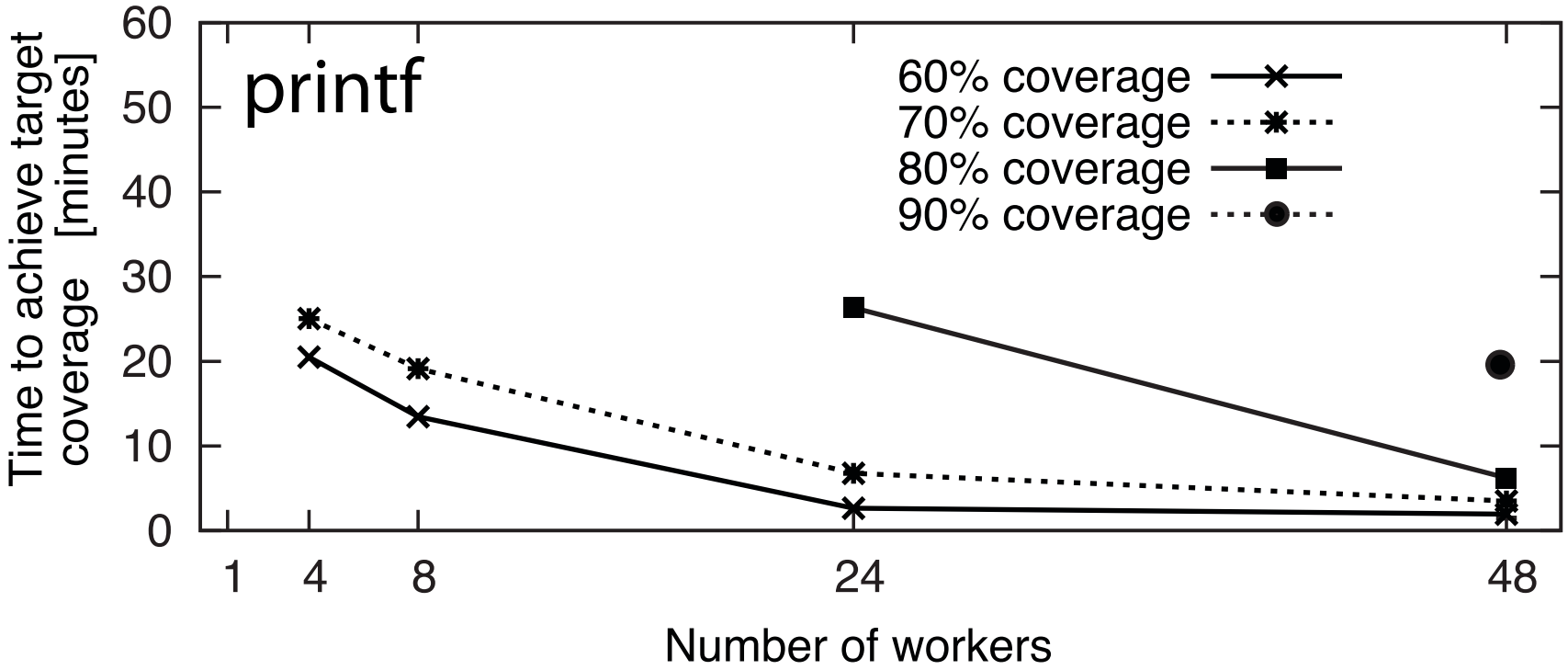
Test Real-World Software

Apache

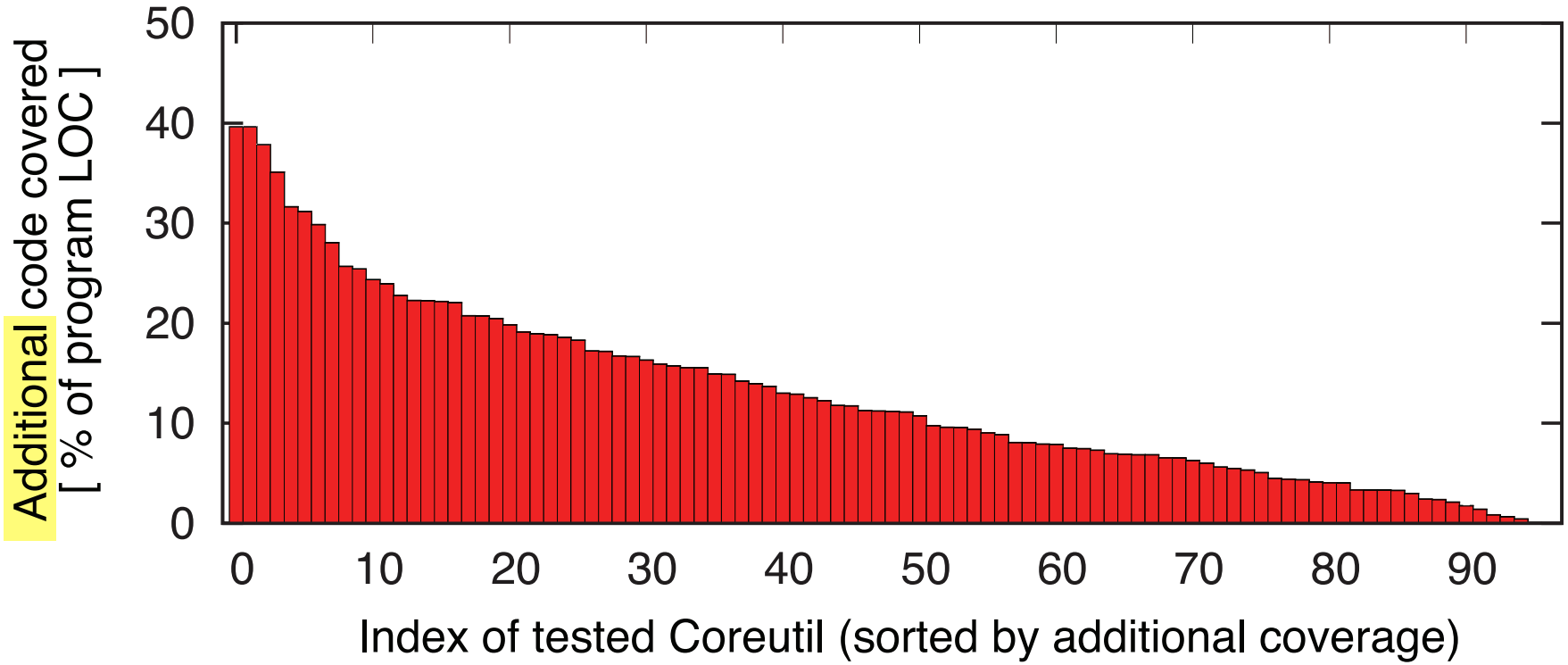
Memcached

GNU Coreutils

Time to Reach Target Coverage

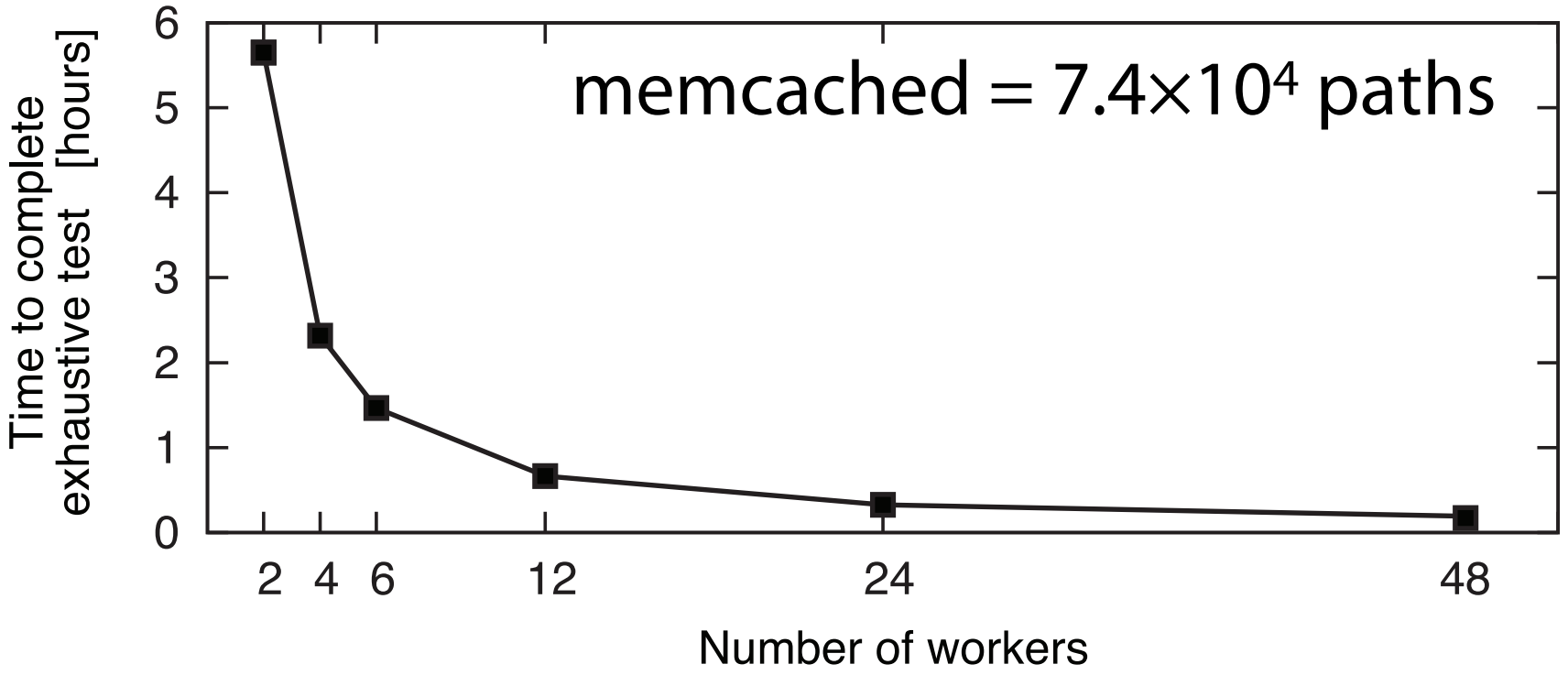


Increase in Code Coverage

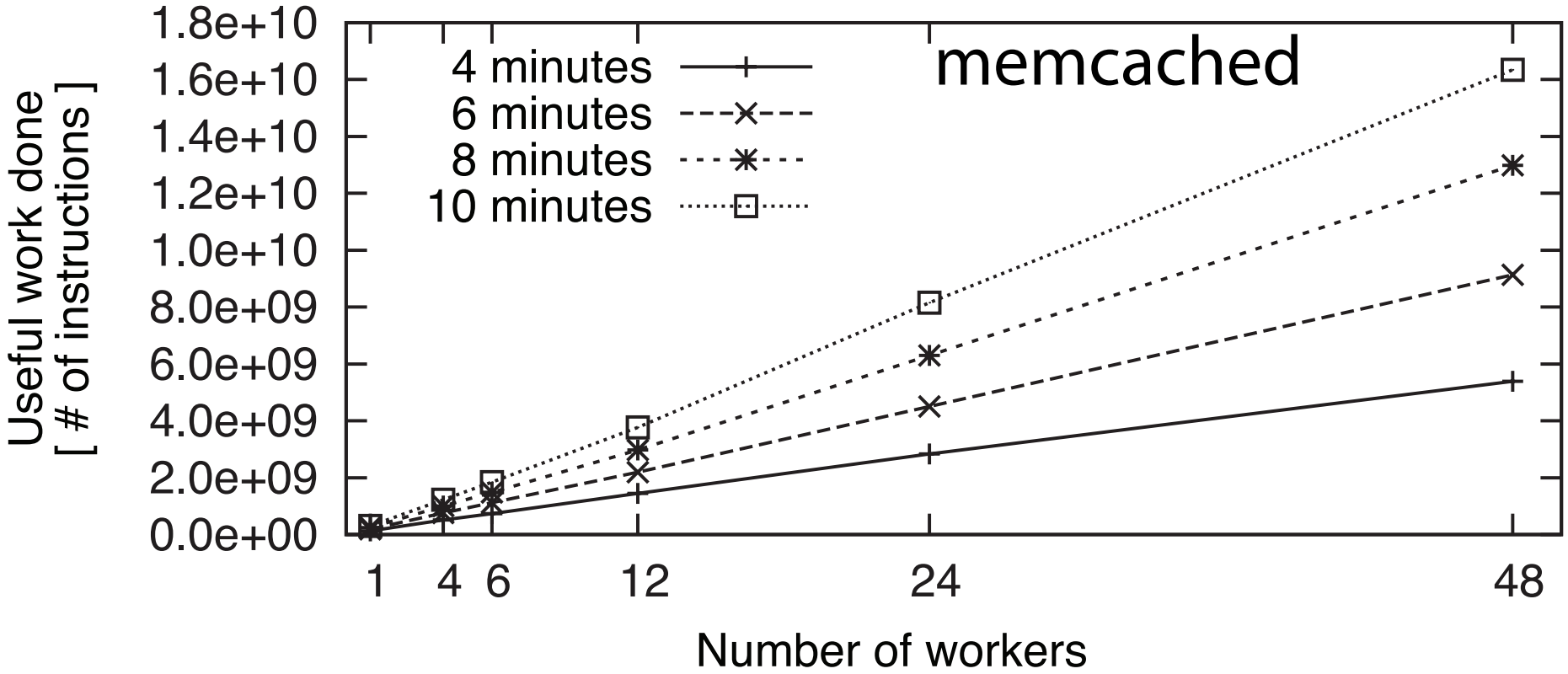


Coreutils suite (12 workers, 10 minutes)

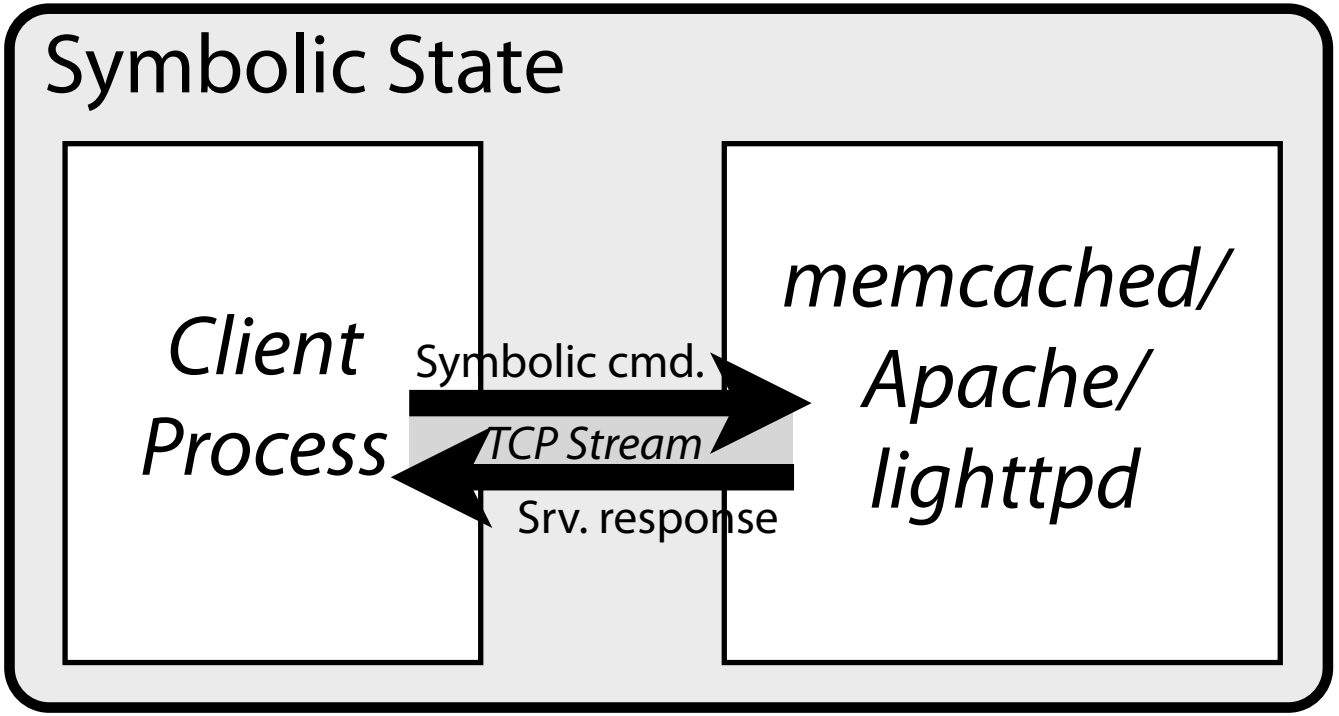
Exhaustive Exploration



Linear Scalability



Distributed Systems



Execute the "whole world" symbolically

Cloud9 in a Nutshell

- Platform for parallel symbolic execution
- Scales linearly on clusters + clouds
  - *throw hardware at the problem*
- Full POSIX model, incl. multithreading

http://cloud9.epfl.ch

Public release any time now
(contact us for private release)

Outline

- Cloud9
  - *Parallel symbolic execution platform*
- → S^2E
  - *Selective symbolic execution platform*
- Lessons for SMT/SAT Solvers
  - *Real data from running on real systems*

S^2E = platform for building analysis tools that are multi-path and in-vivo

Bug Finding

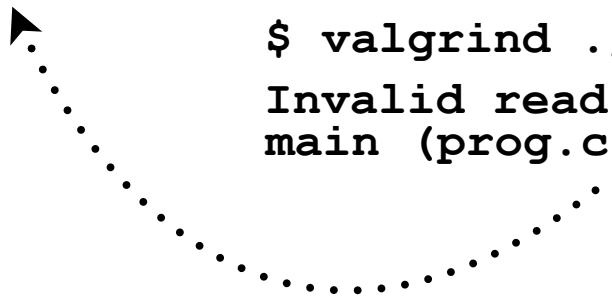
```
int main(argc, argv)
{
  if (argc == 2)
    printf("%c", *argv[2]);

  printf("OK");
}
```

```
$ ./prog
OK

$ ./prog ABC
Segmentation fault

$ valgrind ./prog ABC
Invalid read of size 1
main (prog.c:4)
```



Other Examples

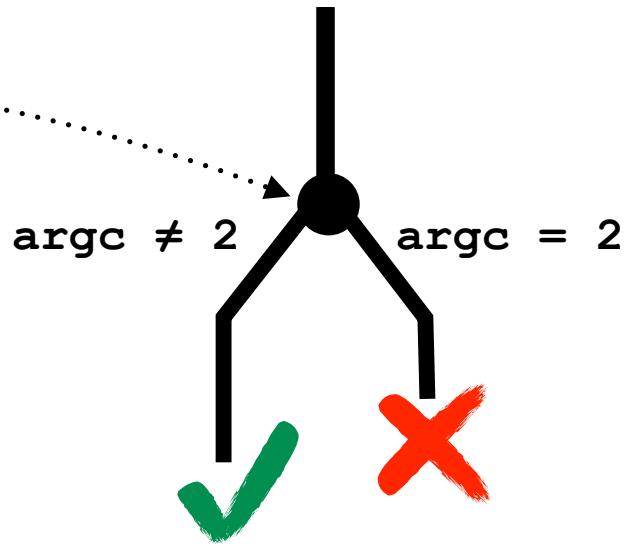
- Security analysis
- Program verification
- Performance profiling
- ...

Analysis = check properties of execution paths

Multi-Path Analysis

```
int main(argc, argv)
{
  if (argc == 2)
    printf("%c", *argv[2]);

  printf("OK");
}
```



= simultaneously analyze multiple paths

In-Vivo Analysis

In Vitro

In Vivo

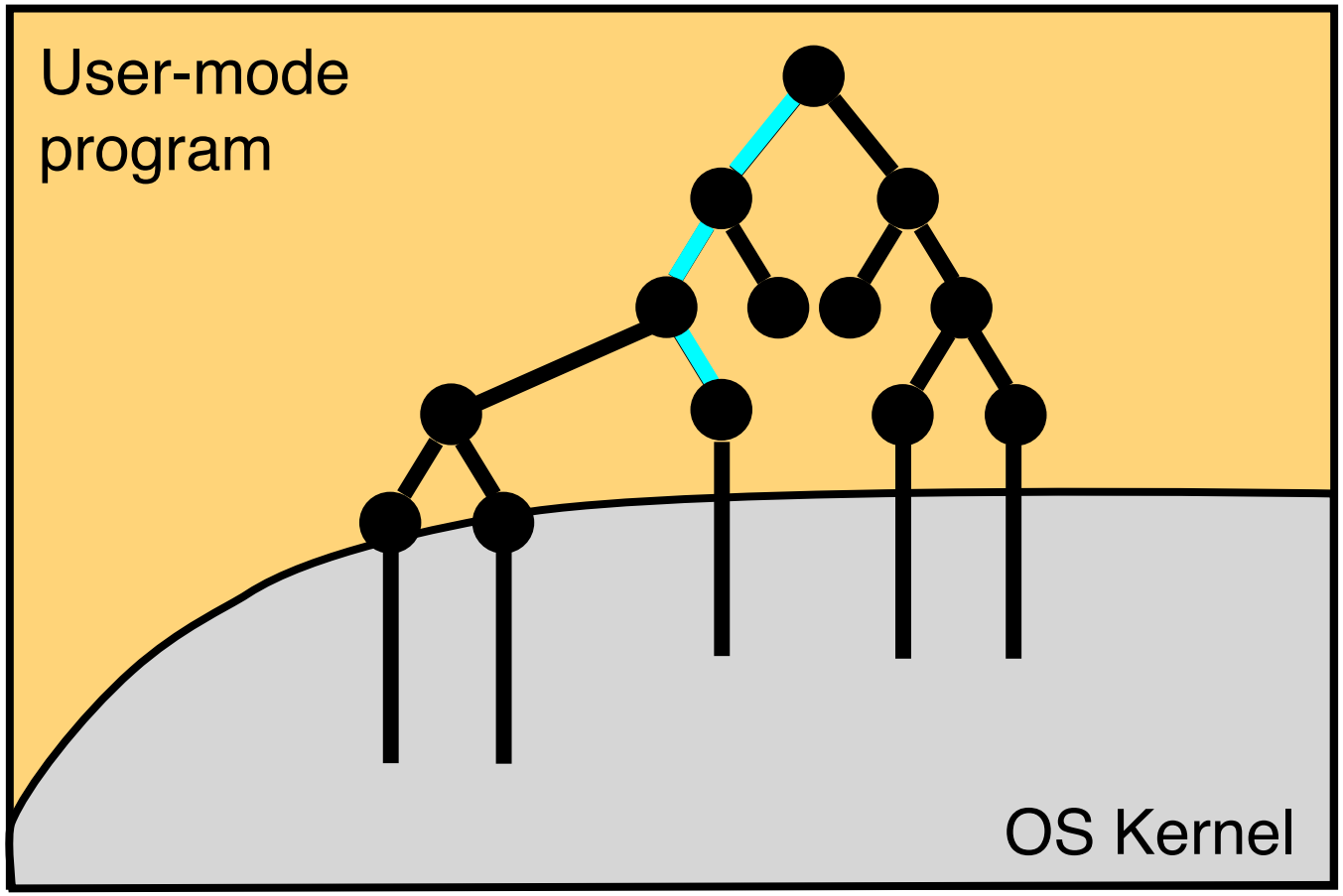
In-Vivo Analysis

In Vitro

In Vivo

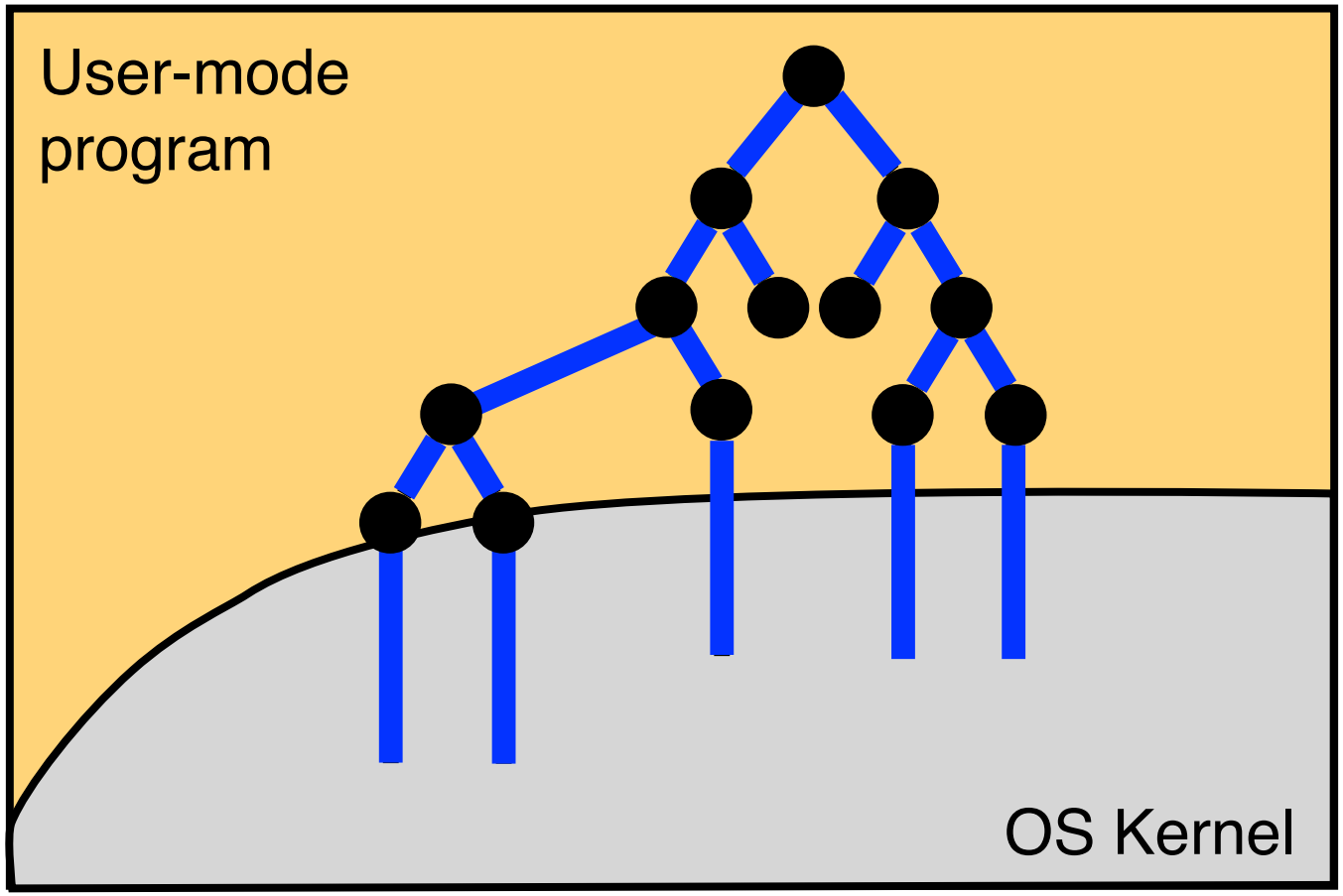
= analyze entire software stack

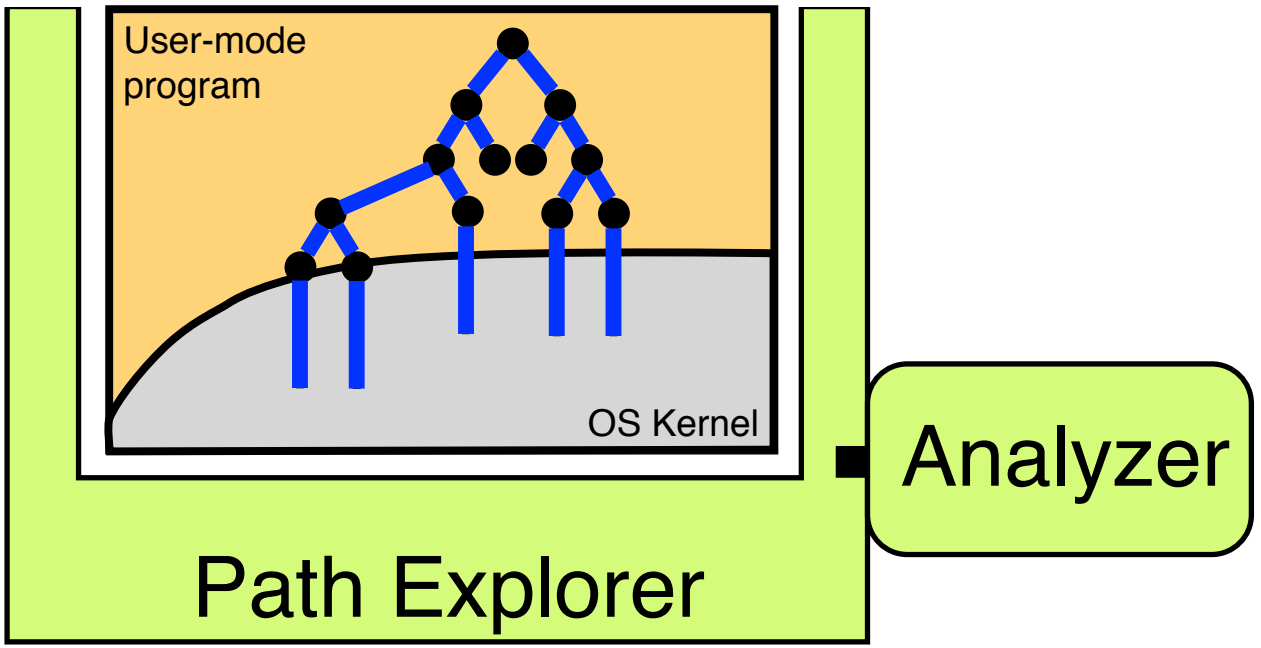
Single-Path In-Vitro



~~Single-Path In-Vitro~~

Multi-Path In-Vivo





S²E = "box" for automated path exploration

Challenge: Path Explosion

$$\#\ \text{of paths} \geq \mathbf{2}^{\text{program size}}$$

Challenge: Path Explosion

$$\#\text{ of paths} \geq 2^{\textbf{system size}}$$

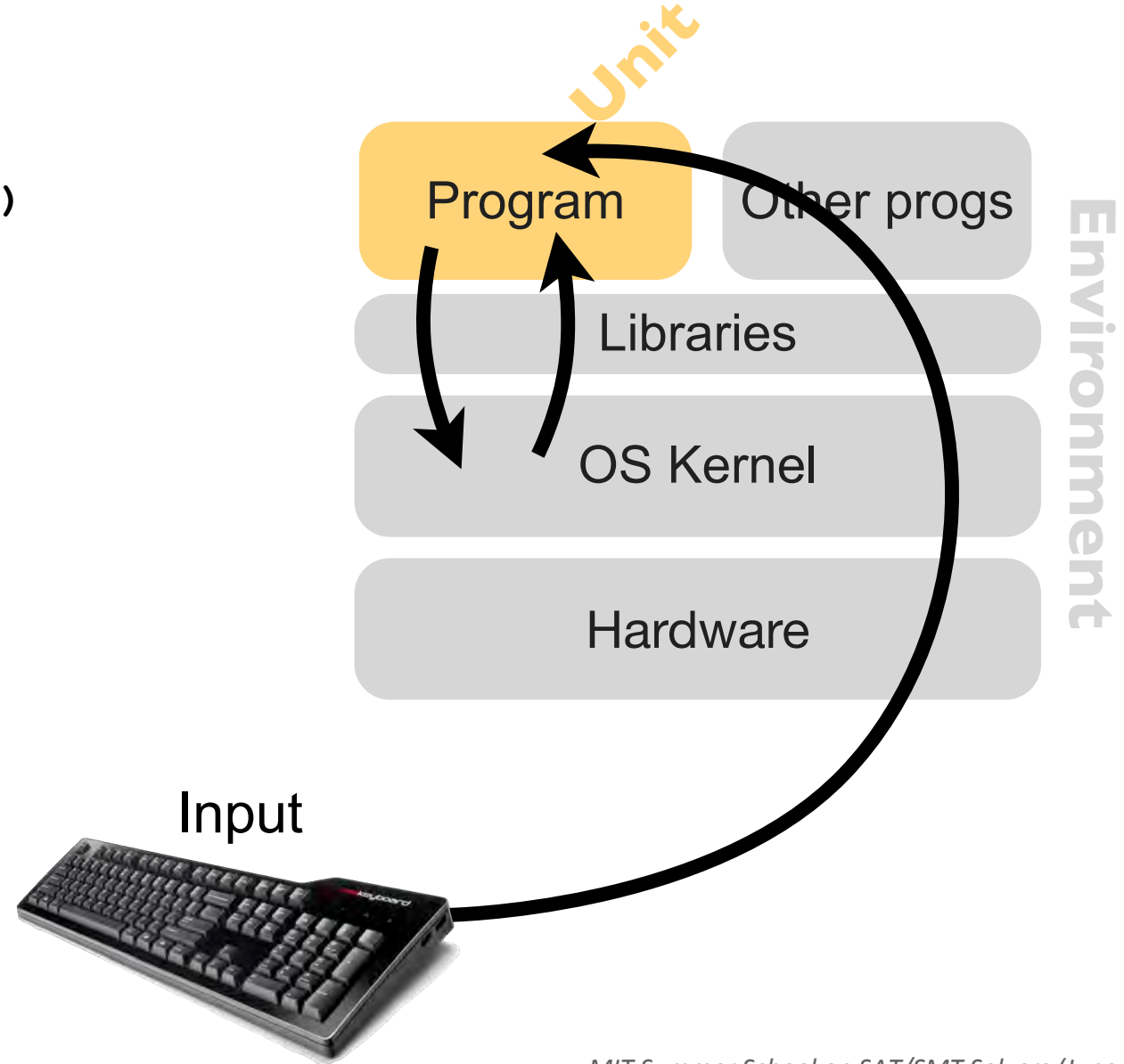
- Cannot analyze all paths ⇒ select only some
  - *automatically select only the paths that are relevant*

Outline

- Cloud9
- S^2E
  - *Concept: In-Vivo Multi-Path Analysis*
  - → *Idea #1: Selective Path Analysis*
  - *Idea #2: Analysis-specific Execution Consistency*
  - *Implementation: The S^2E System*
  - *Three Use Cases*
- Lessons for SMT/SAT Solvers

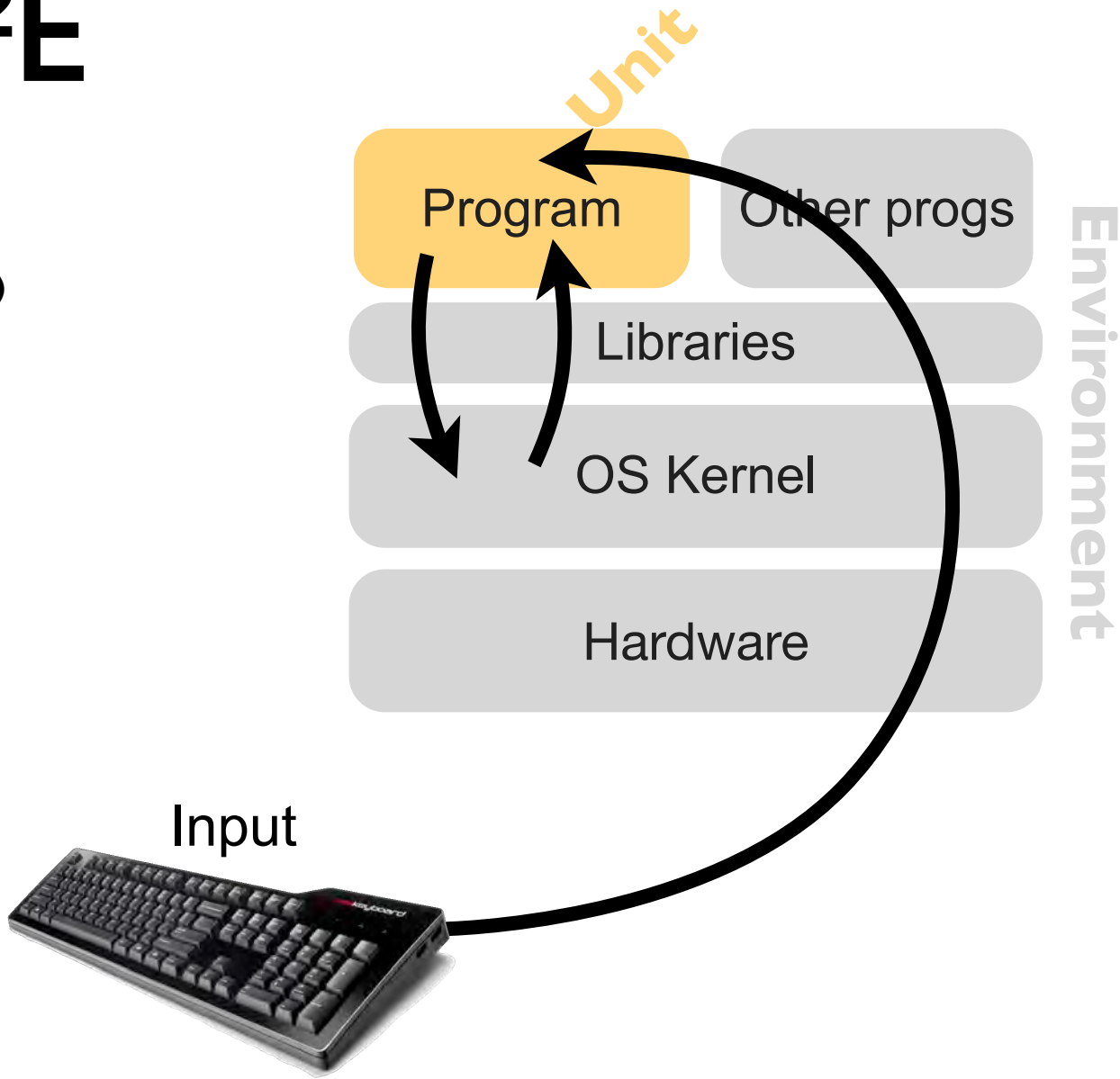
Unit vs. Environment

```
int main(argc, argv)
{
  if (argc == 0) {
      ...
  }
  p = malloc(...);
  if (p == NULL) {
      ...
  }
}
```



Goal in S²E

```
int main(argc, argv)
{
  if (argc == 0) {
     ...
  }
  p = malloc(...);
  if (p == NULL) {
     ...
  }
}
```



Provide illusion of full-system analysis

Concrete → Symbolic

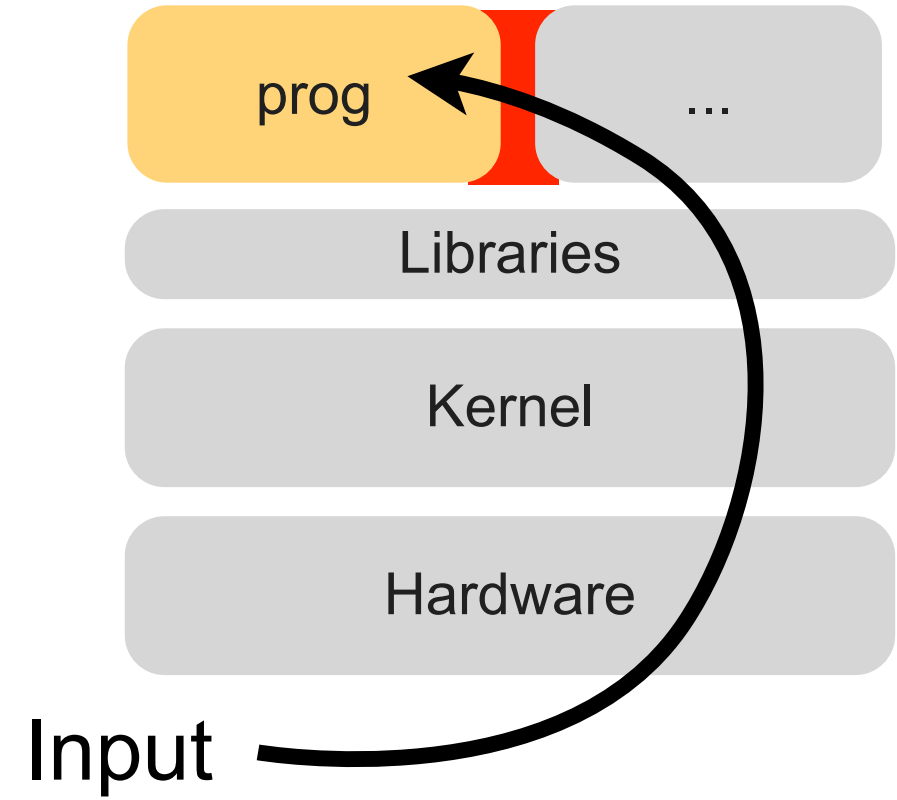
```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```

α β



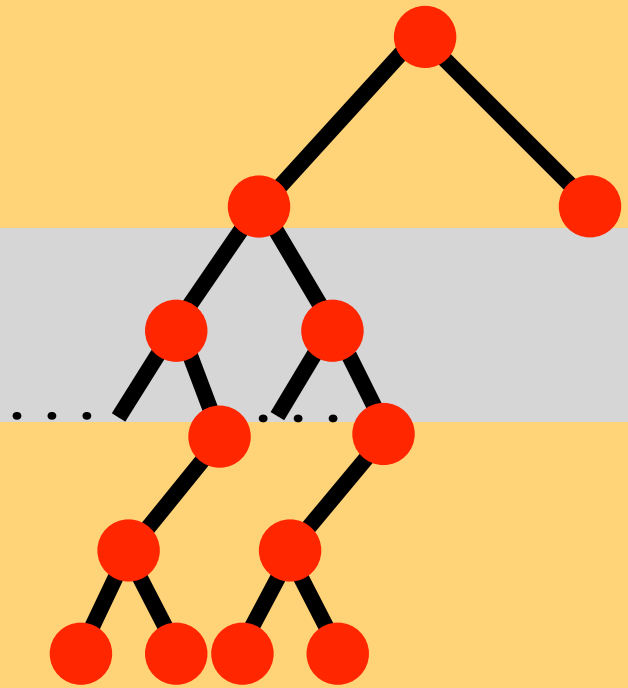
Symbolic → Concrete

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



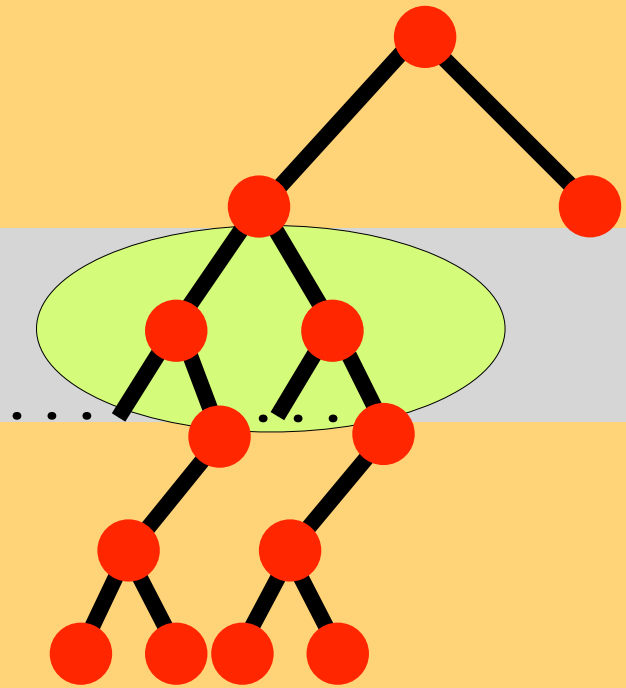
Symbolic → Concrete

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



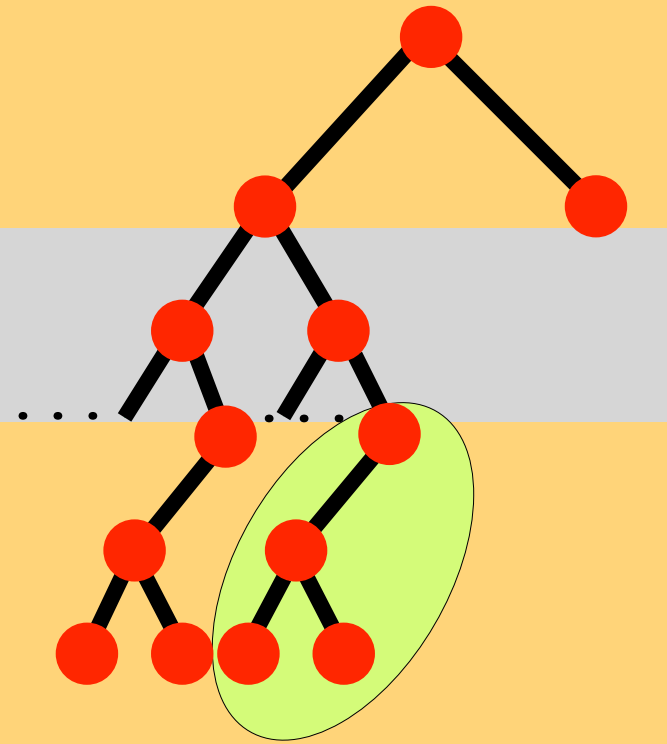
Symbolic → Concrete

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



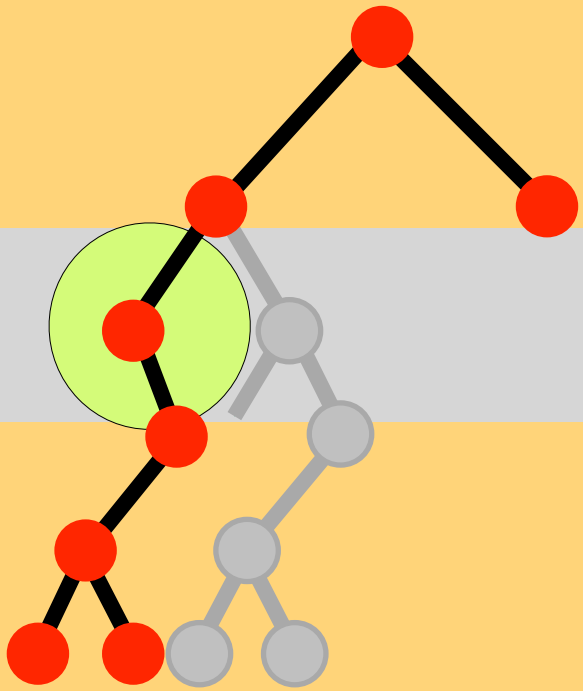
Symbolic → Concrete

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



Selective Symbolic Execution

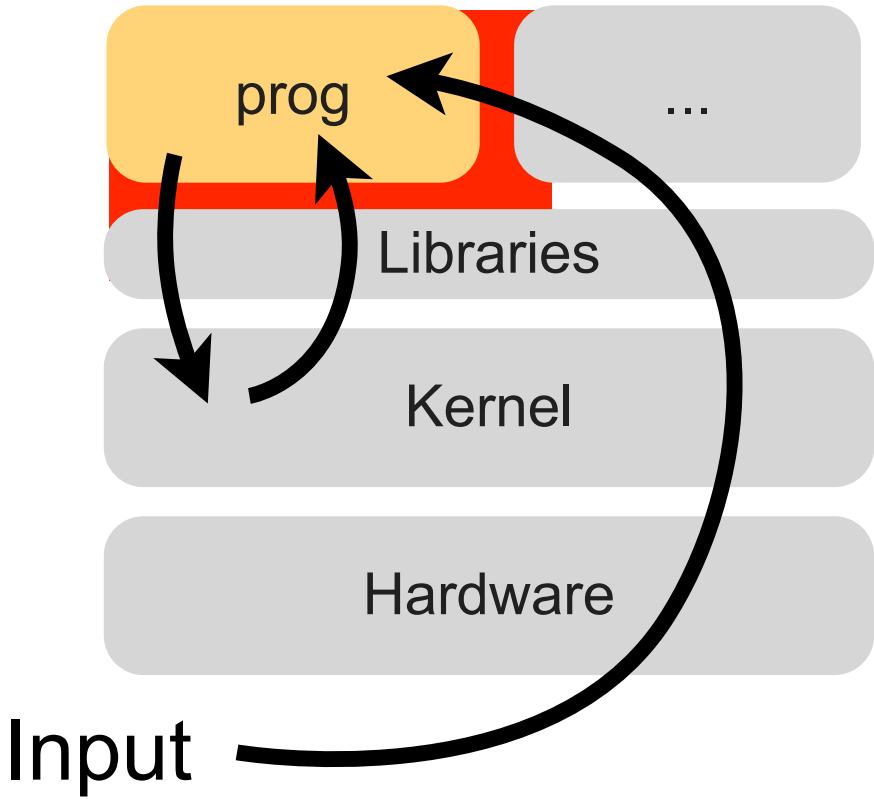
```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc( 0 );

  if (p == NULL) {
     ...
  }
  ...
}
```

α β 128



Consistent Execution

- Path selection must be done carefully...
  - *preserve illusion of full-system analysis*
  - *unit/environment interaction must be realistic*
  - *preserve efficiency (explore minimum # of paths necessary)*

Examples of Inconsistency

→ Paths modify each other's environment state

- *happens when environment changes are not isolated*
- *e.g., in test generation systems (DART, EXE, ...)*

- Use models of the environment
  - *models (by definition) exhibit behavioral differences*
  - *e.g., model checkers, symbex engines (SLAM, KLEE, ...)*

Examples of Inconsistency

- Paths modify each other's environment state
  - *happens when environment changes are not isolated*
  - *e.g., in test generation systems (DART, EXE, ...)*

→ Use models of the environment

  - *models (by definition) exhibit behavioral differences*
  - *e.g., model checkers, symbex engines (SLAM, KLEE, ...)*

In S²E, each path has its own real environment

Outline

- Cloud9
- S^2E
  - *Concept: In-Vivo Multi-Path Analysis*
  - *Idea #1: Selective Symbolic Execution*
  - → *Idea #2: Analysis-specific Execution Consistency*
  - *Implementation: The S^2E System*
  - *Three Use Cases*
- Lessons for SMT/SAT Solvers

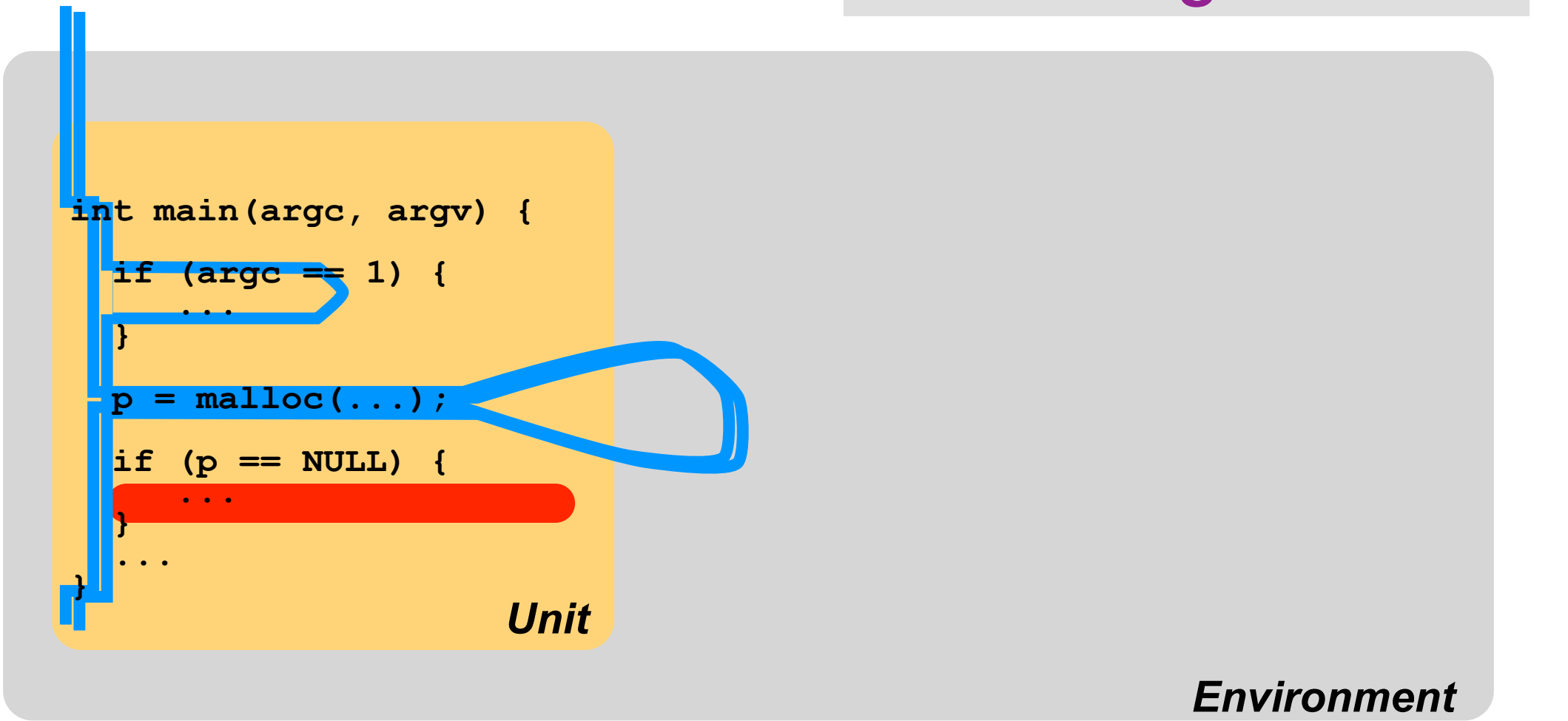
Execution Consistency Models

- Not the same as memory consistency models
  - *but similar in spirit*
- An ECM = set of paths to be analyzed
  - *i.e., abstract specification of which paths to analyze*
  - *execution is consistent iff its path is in the ECM set*

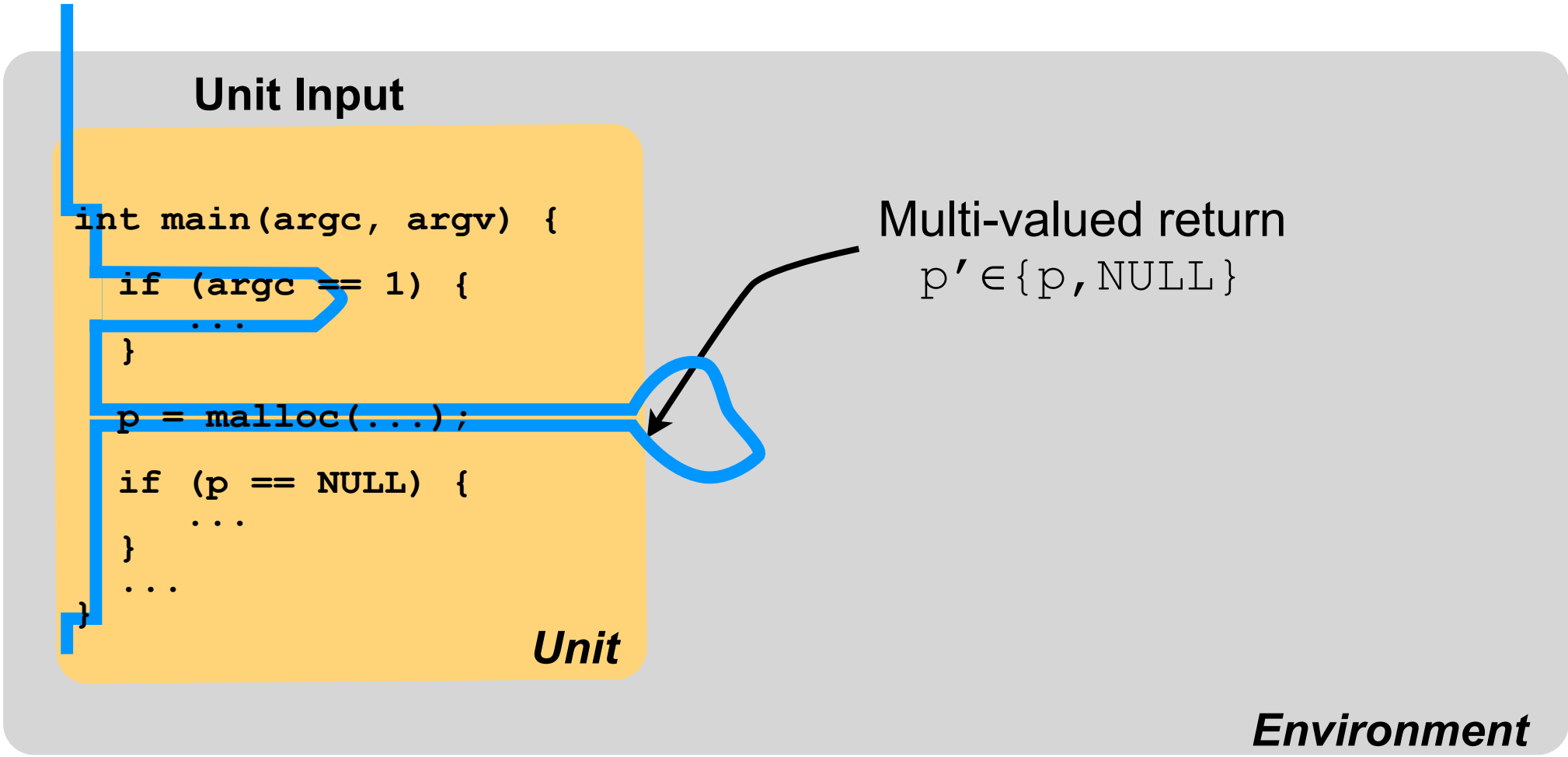
SC-UE (SC Unit-level Execution)

Much fewer paths

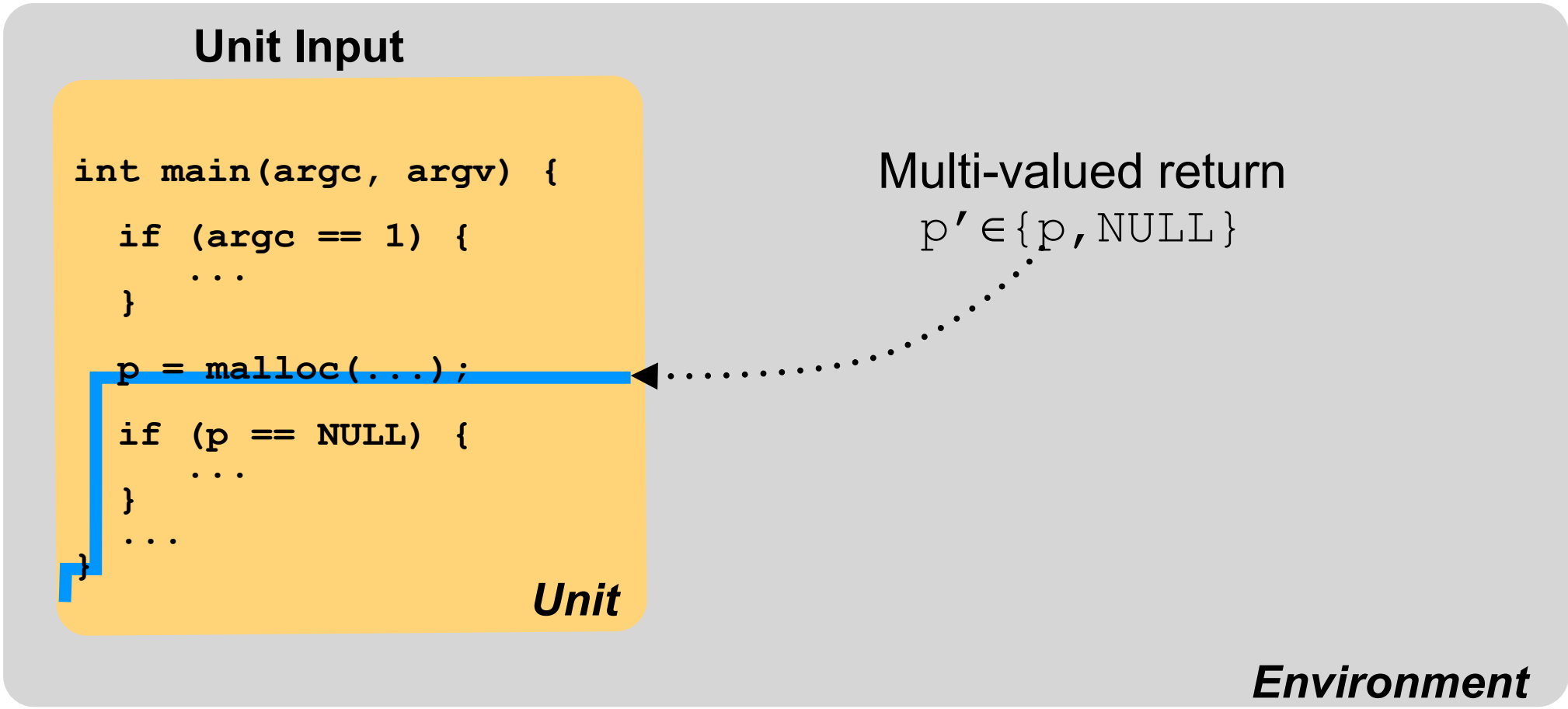
False negatives ?



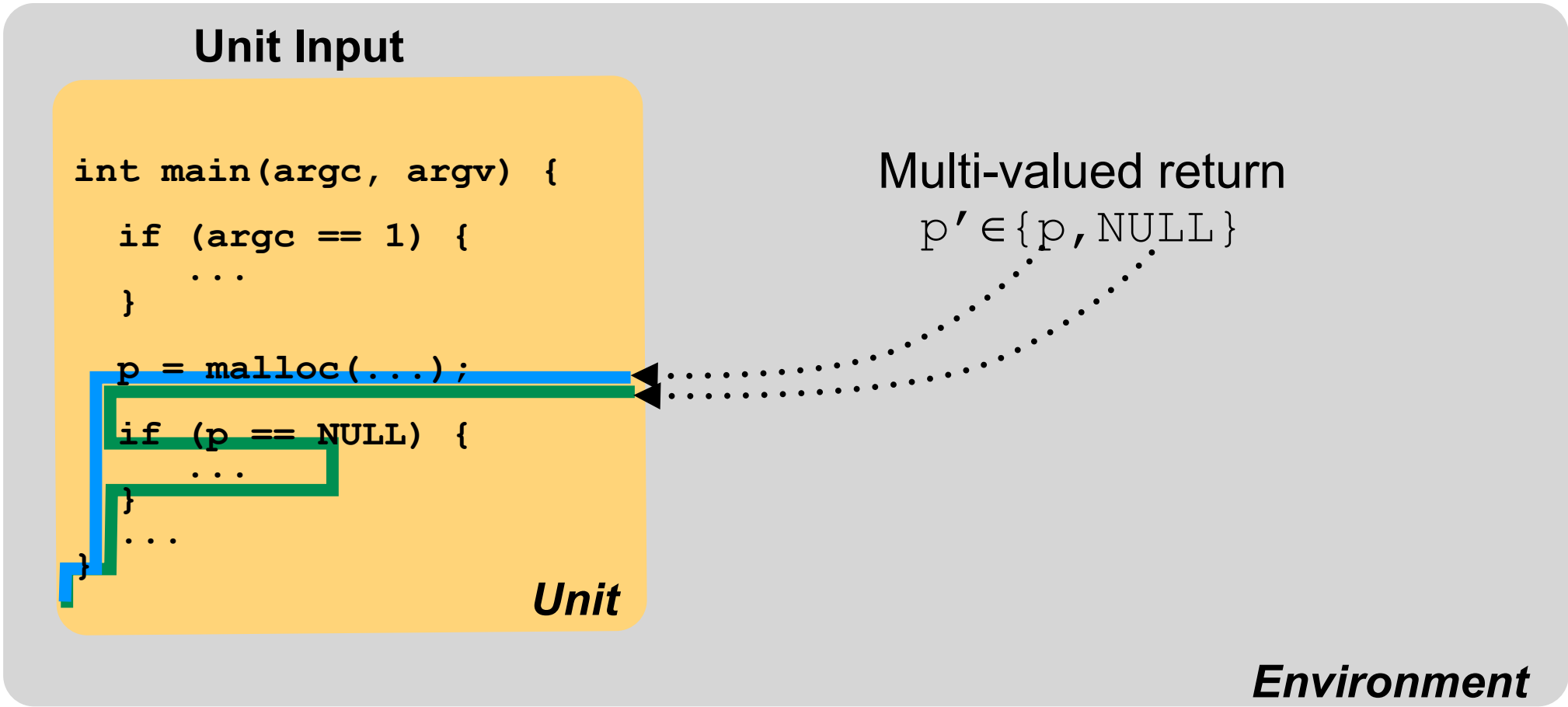
RC-OC (Relaxed Overapprox. Consistency)



RC-OC (Relaxed Overapprox. Consistency)

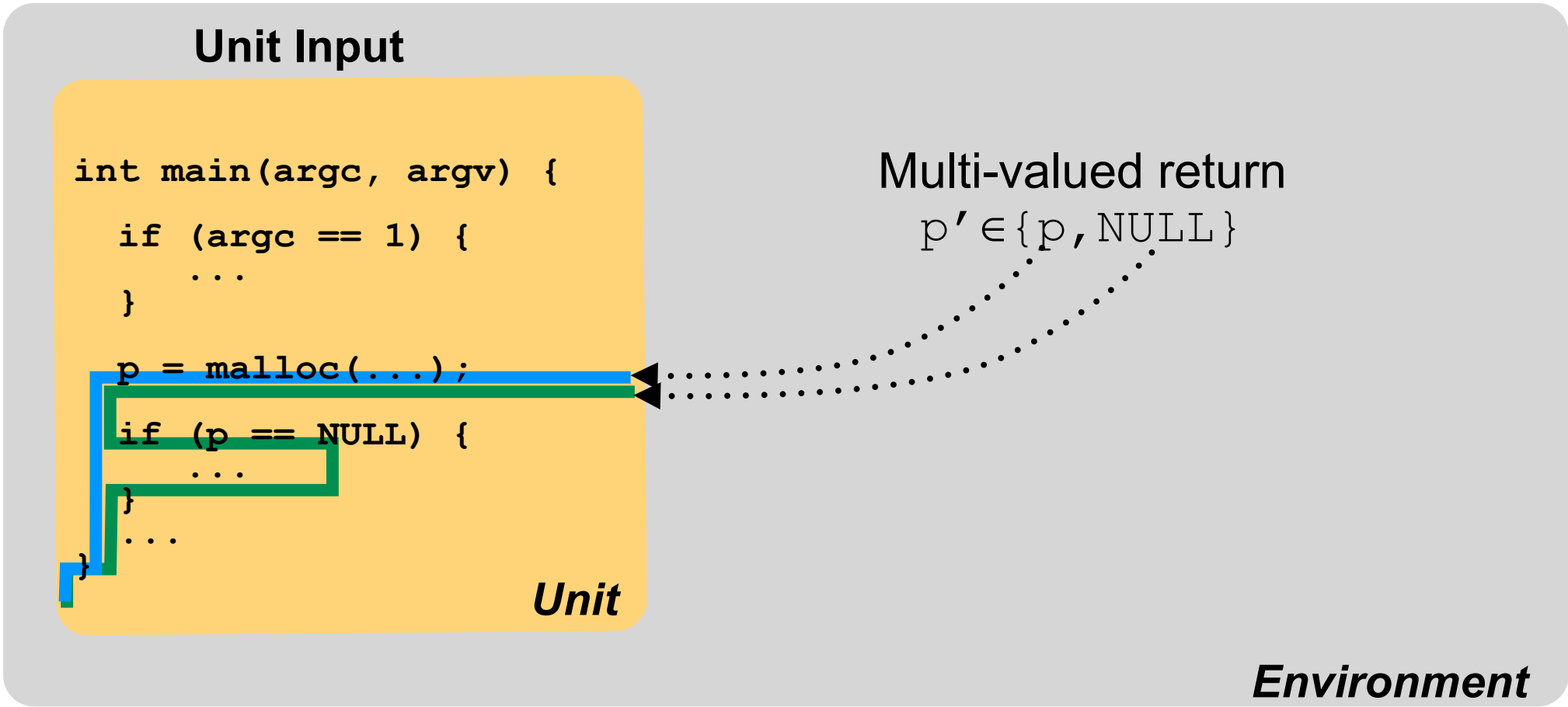


RC-OC (Relaxed Overapprox. Consistency)



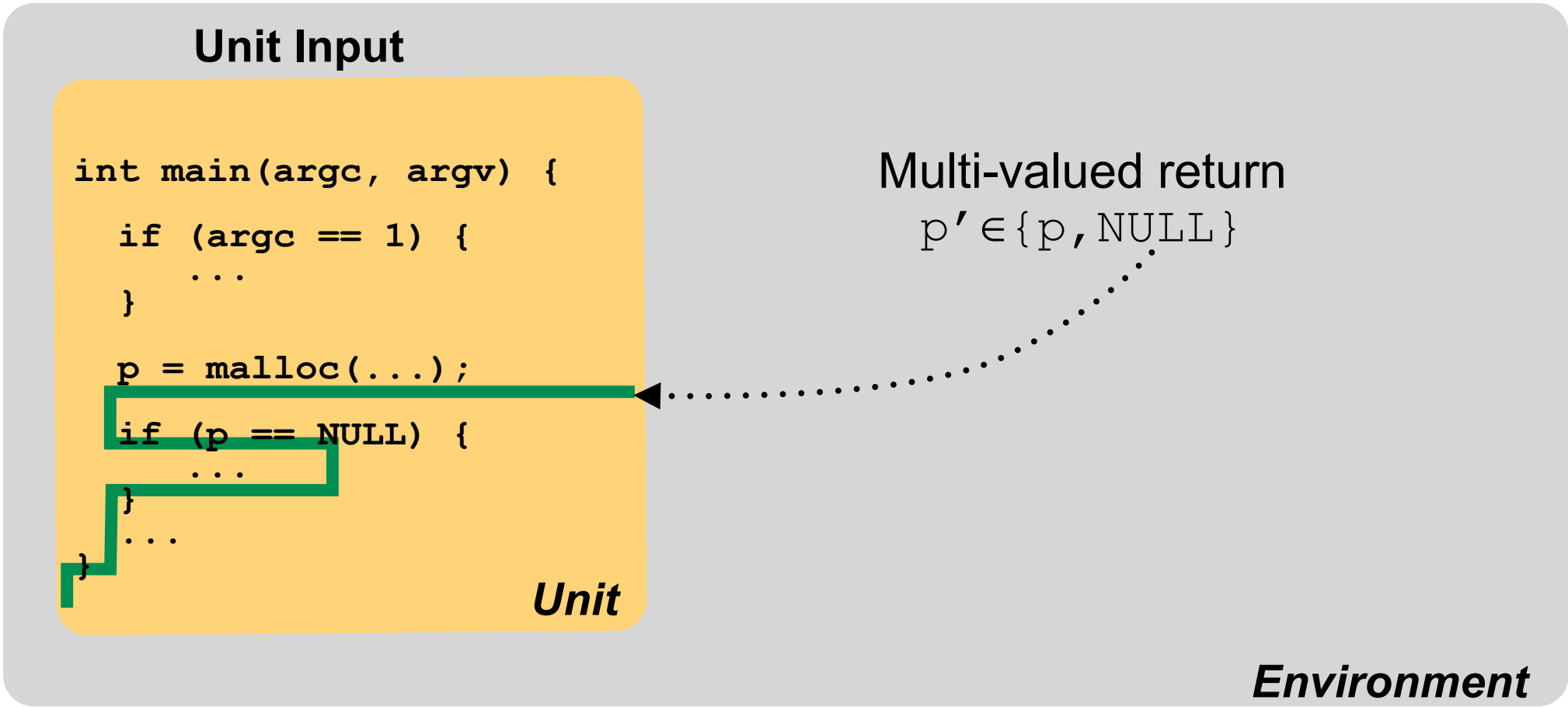
RC-OC (Relaxed Overapprox. Consistency)

No false negatives



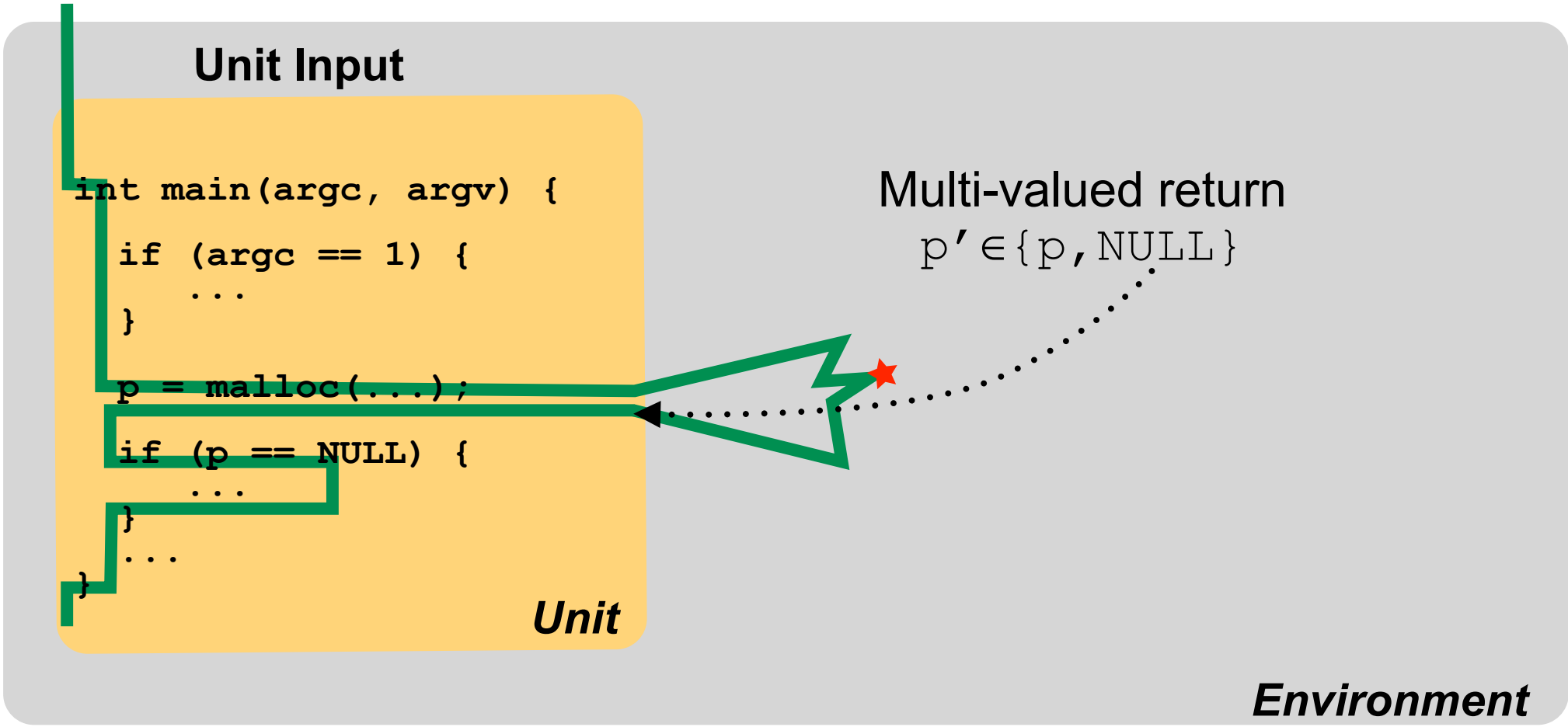
RC (Relaxed Consistency)

No false negatives



RC (Relaxed Consistency)

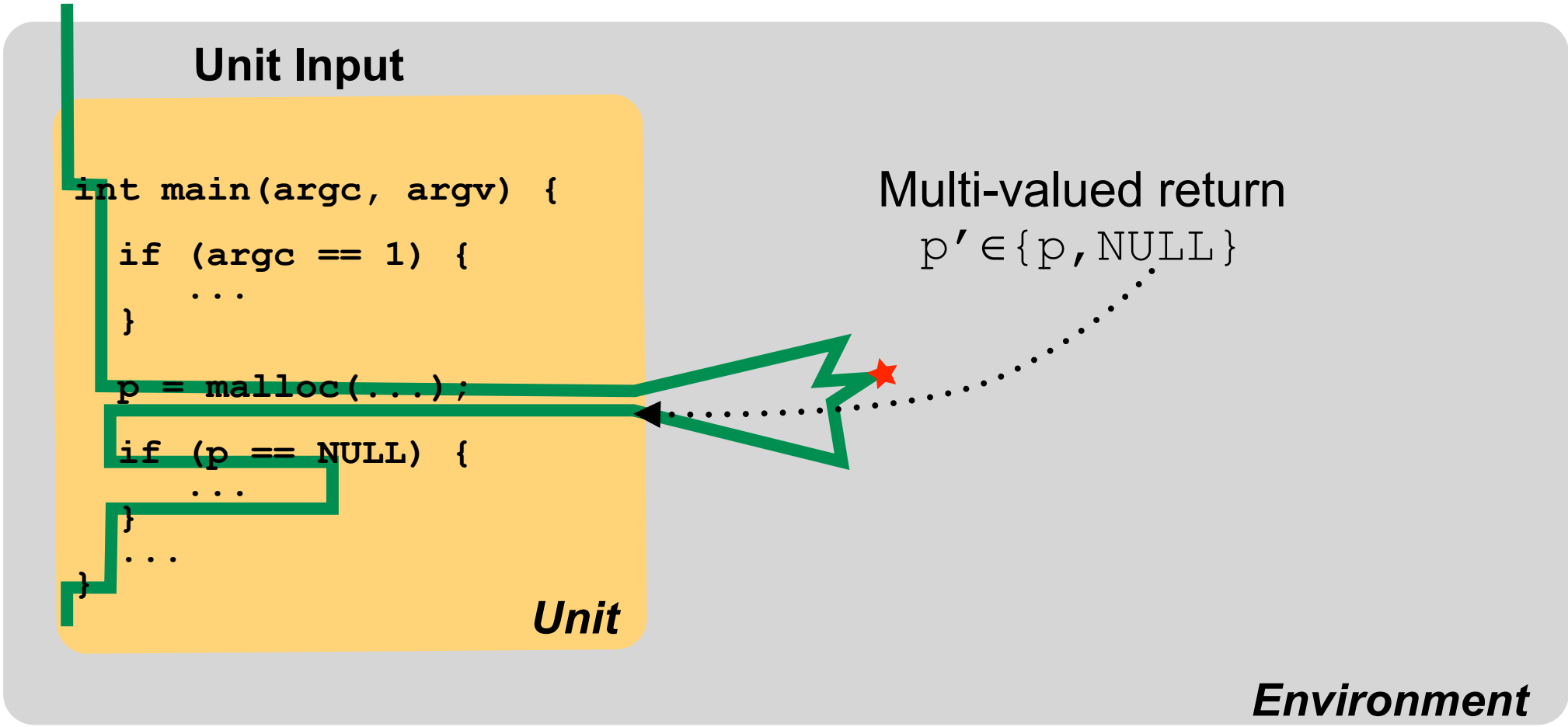
No false negatives



RC (Relaxed Consistency)

No false negatives

False positives ?



LC (Local Consistency)

Unit Input

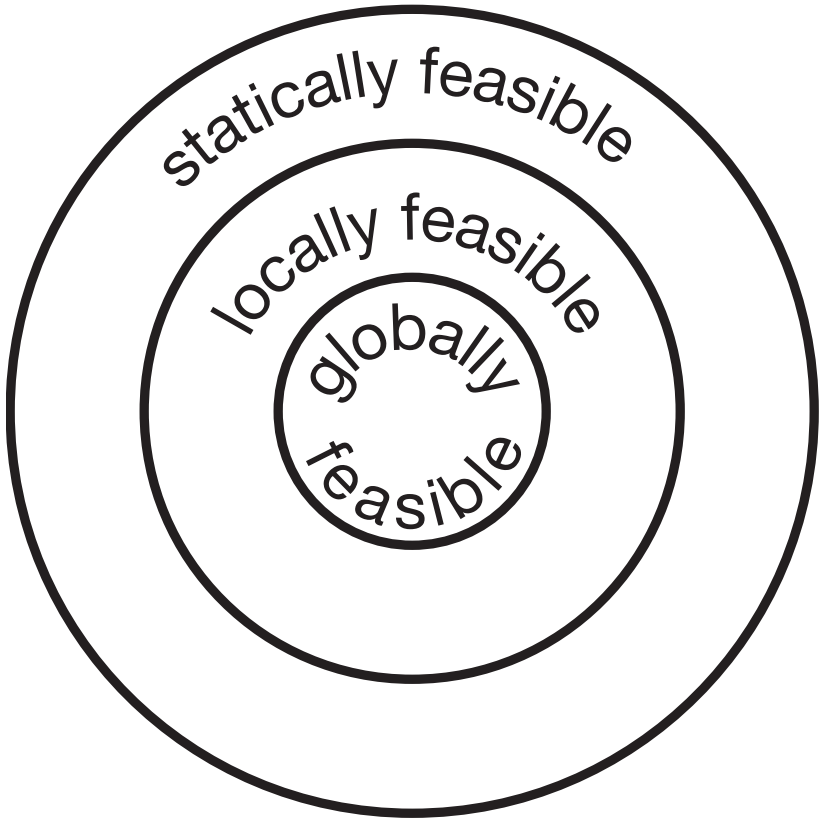
```
int main(argc, argv) {
  if (argc == 1) {
    ...
  }
  p = malloc(...);
  ...
}
```

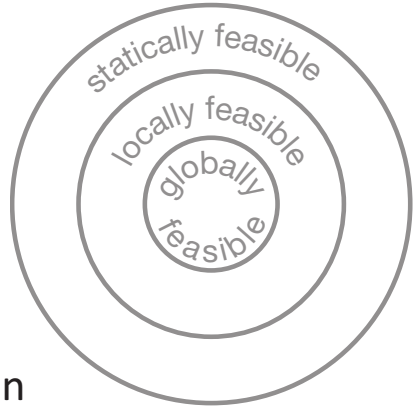
Unit

Interface annotation

`malloc() → {p,NULL}`

Environment


statically feasible
locally feasible
globally feasible



SC-CE
Strictly consistent concrete execution

SC-UE
Strictly consistent unit-level execution

SC-SE
Strictly consistent system-level execution

+ more inputs based on knowledge of constraints from within the unit

+ more inputs based on knowledge of constraints from environment

+ relax constraints at unit/environment boundary consistently with environment interface specification

+ arbitrarily relax constraints at unit/environment boundary

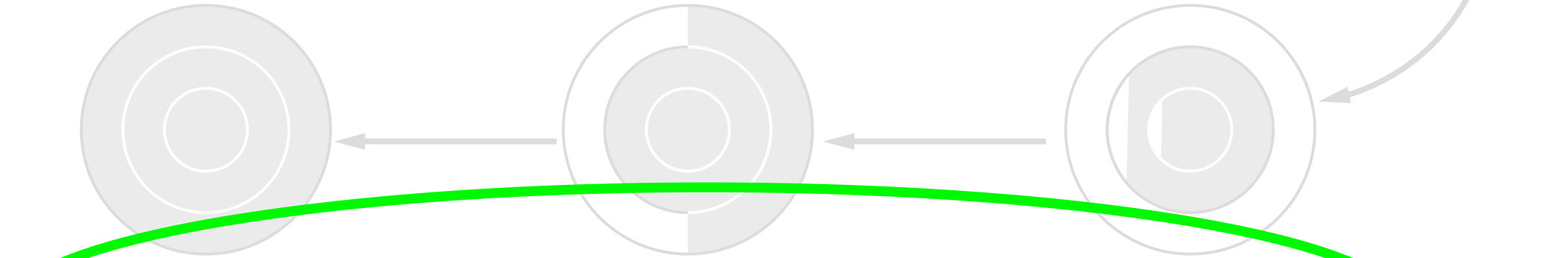
+ arbitrarily relax constraints anywhere

RC-CC
CFG consistency

RC-OC
Overapproximate consistency

LC
Local consistency

SC-CE Strictly consistent concrete execution	SC-UE Strictly consistent unit-level execution	SC-SE Strictly consistent system-level execution
Classic fuzzing (most real-world testing)	Dynamic symbolic & concolic execution engines (DART, EXE)	Symbolic execution engines with environment models (KLEE, Cloud9)



RC-CC
CFG consistency

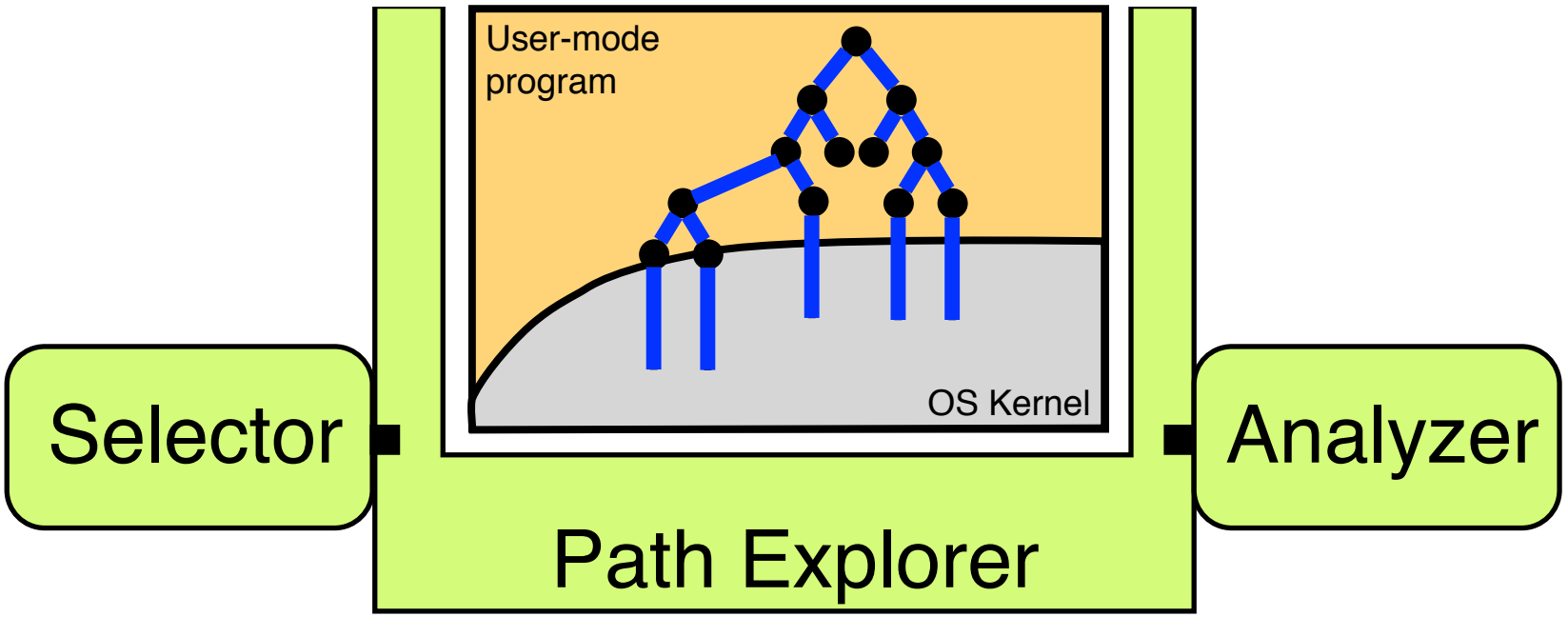
RC-OC
Overapproximate consistency

LC
Local consistency

Mix & Match

- ECM = specification of paths to be explored
  - *S2E underneath the covers explores the requested paths*
- Analyzer can make principled trade-offs
  - *FPs vs. FNs vs. exploration+analysis performance*
- Minimize the number of explored paths
  - *all the paths in the ECM set, but none extra*

Can implement any ECM



S²E = box for automated path exploration

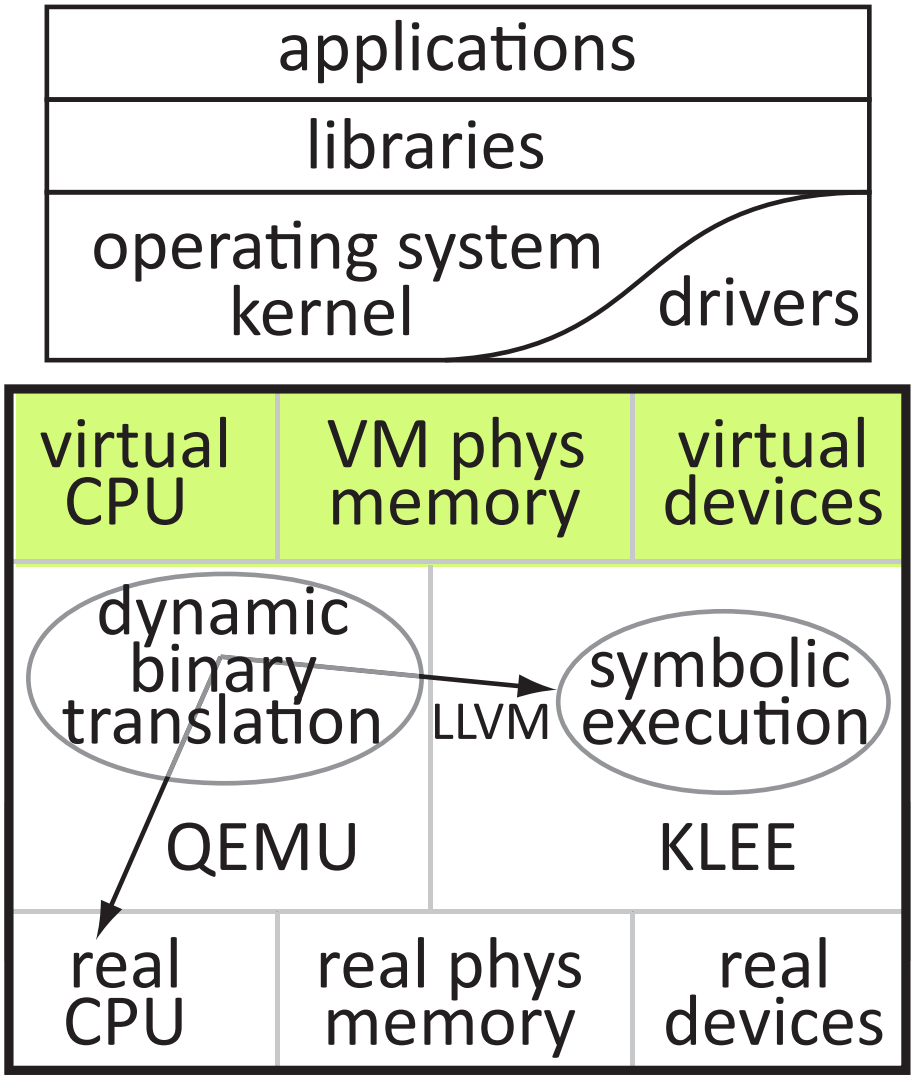
Outline

- Cloud9
- S^2E Platform
  - *Concept: In-Vivo Multi-Path Analysis*
  - *Idea #1: Selective Symbolic Execution*
  - *Idea #2: Analysis-specific Execution Consistency*
  - → *Implementation: The S^2E System*
  - *Three Use Cases*
- Lessons for SMT/SAT Solvers

The S²E System

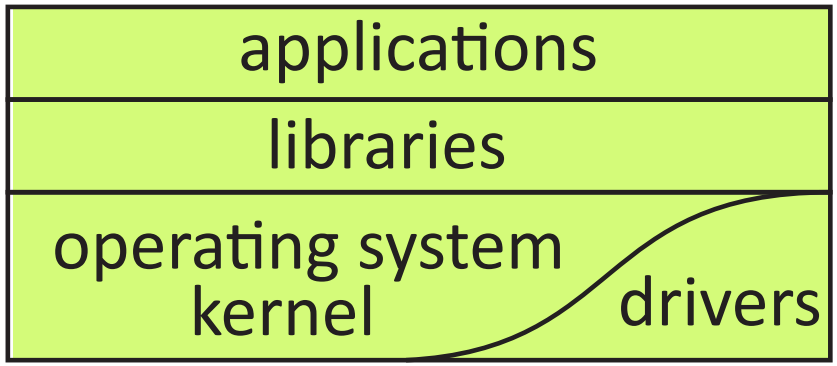
S²E = virtualization +
dynamic binary translation +
selective symbolic execution

The S²E System

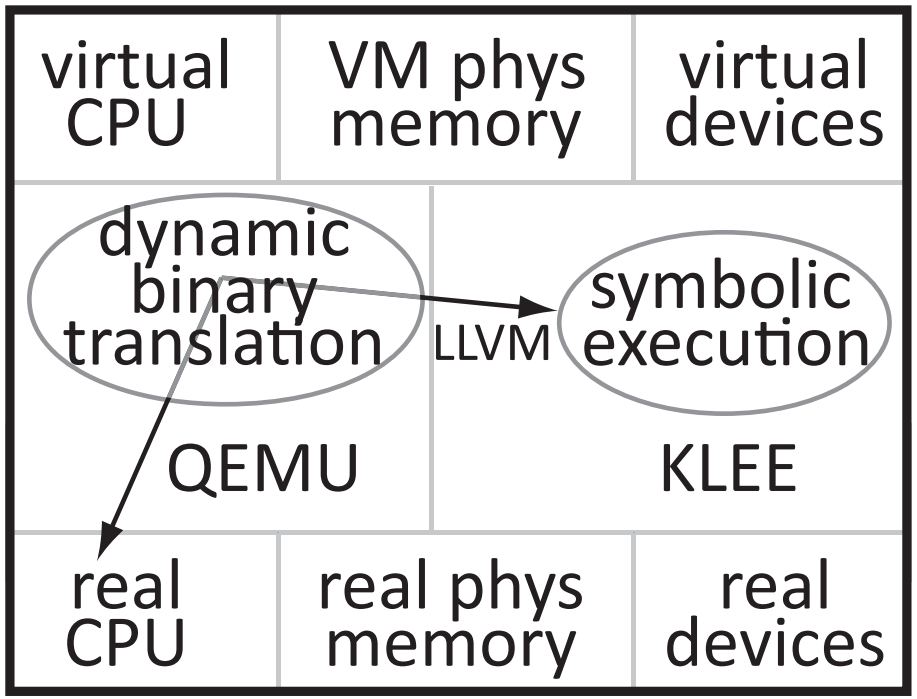


Customized virtual machine

The S²E System

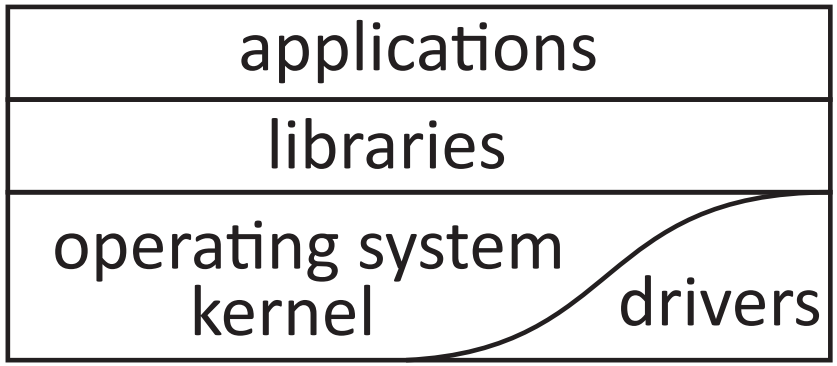


Runs unmodified x86 binaries (including proprietary, obfuscated, and encrypted binaries)

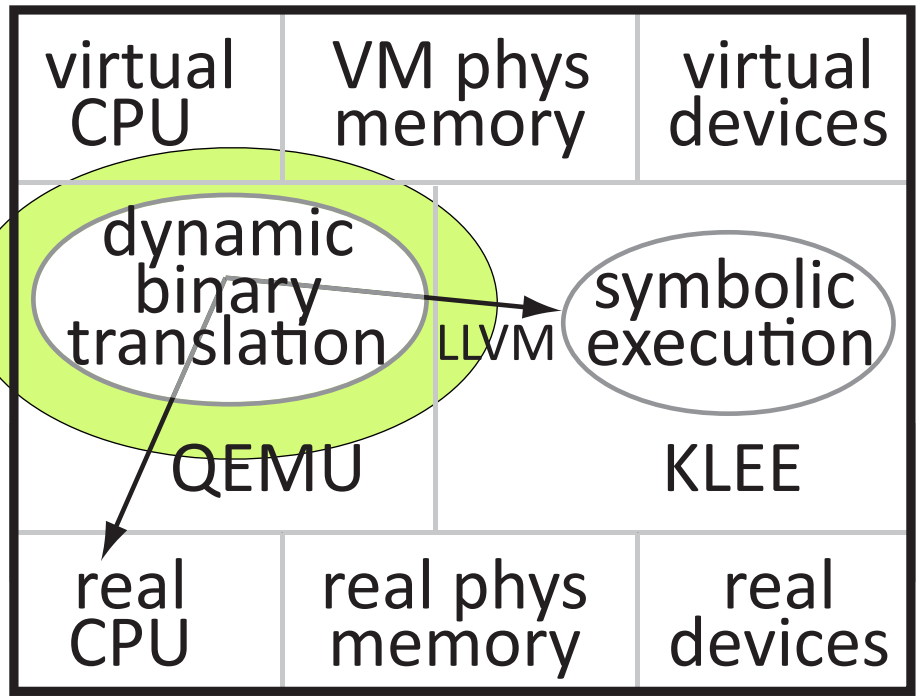


Customized virtual machine

The S²E System



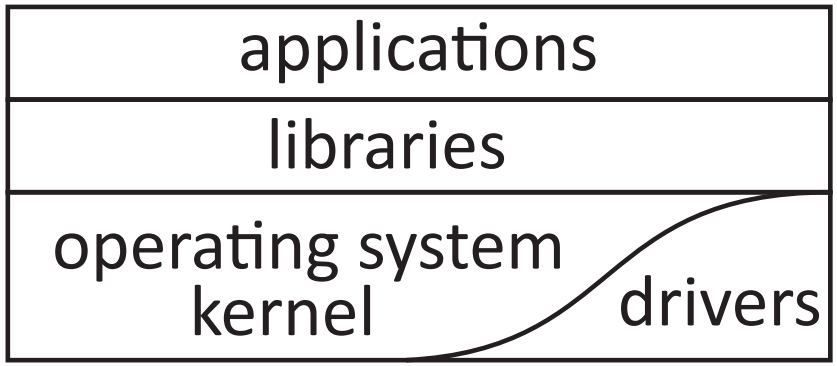
Runs unmodified x86 binaries (including proprietary, obfuscated, and encrypted binaries)



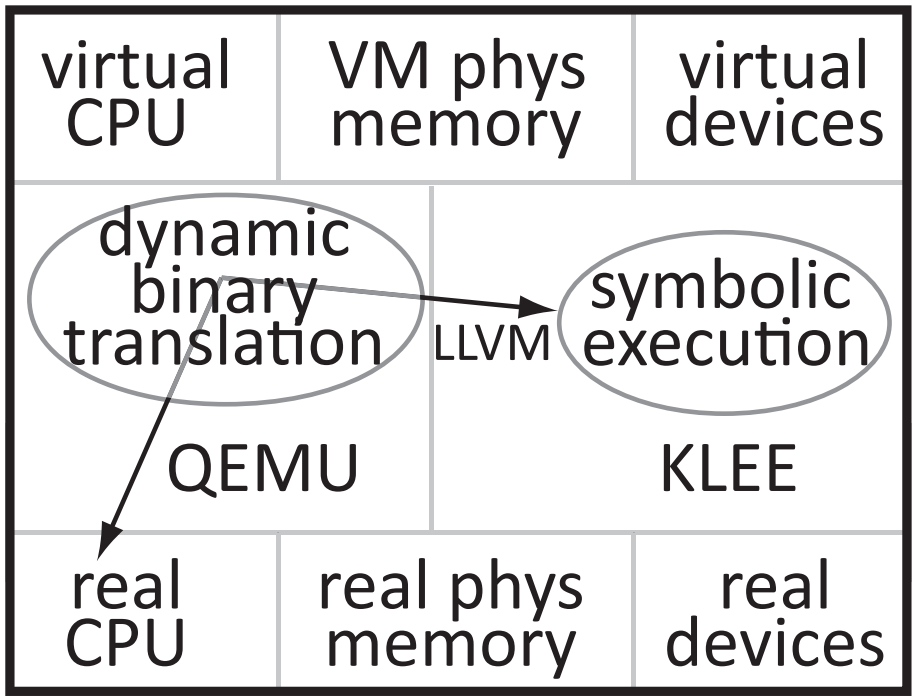
Customized virtual machine

Selection done at runtime
Most code runs “natively”

The S²E System



Runs unmodified x86 binaries (incl. proprietary/obfuscated/ encrypted binaries)

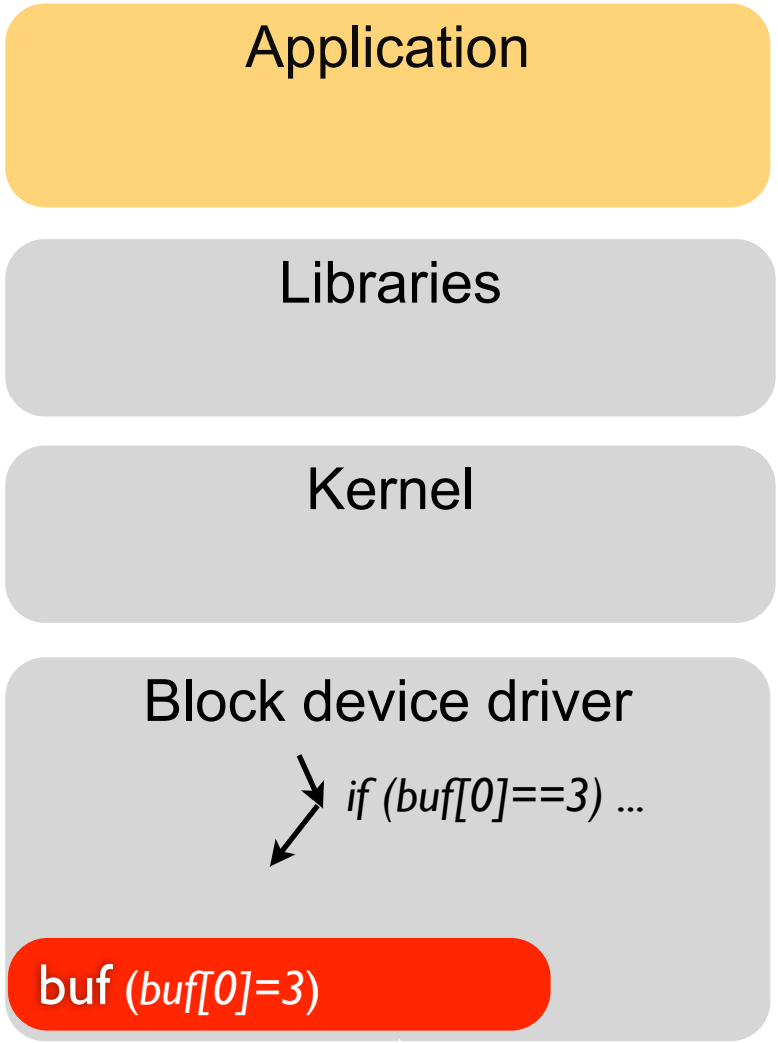


Customized virtual machine

Selection done at runtime
Most code runs “natively”

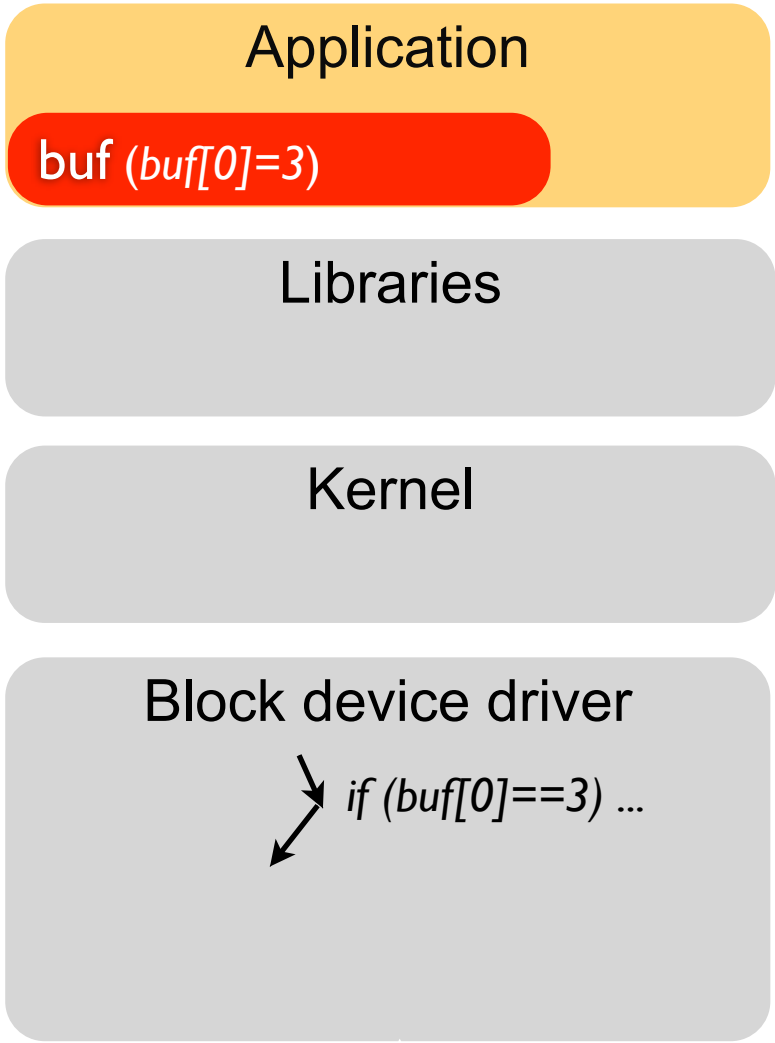
Shared concrete/symbolic state representation

Lazy+Selective Conversion



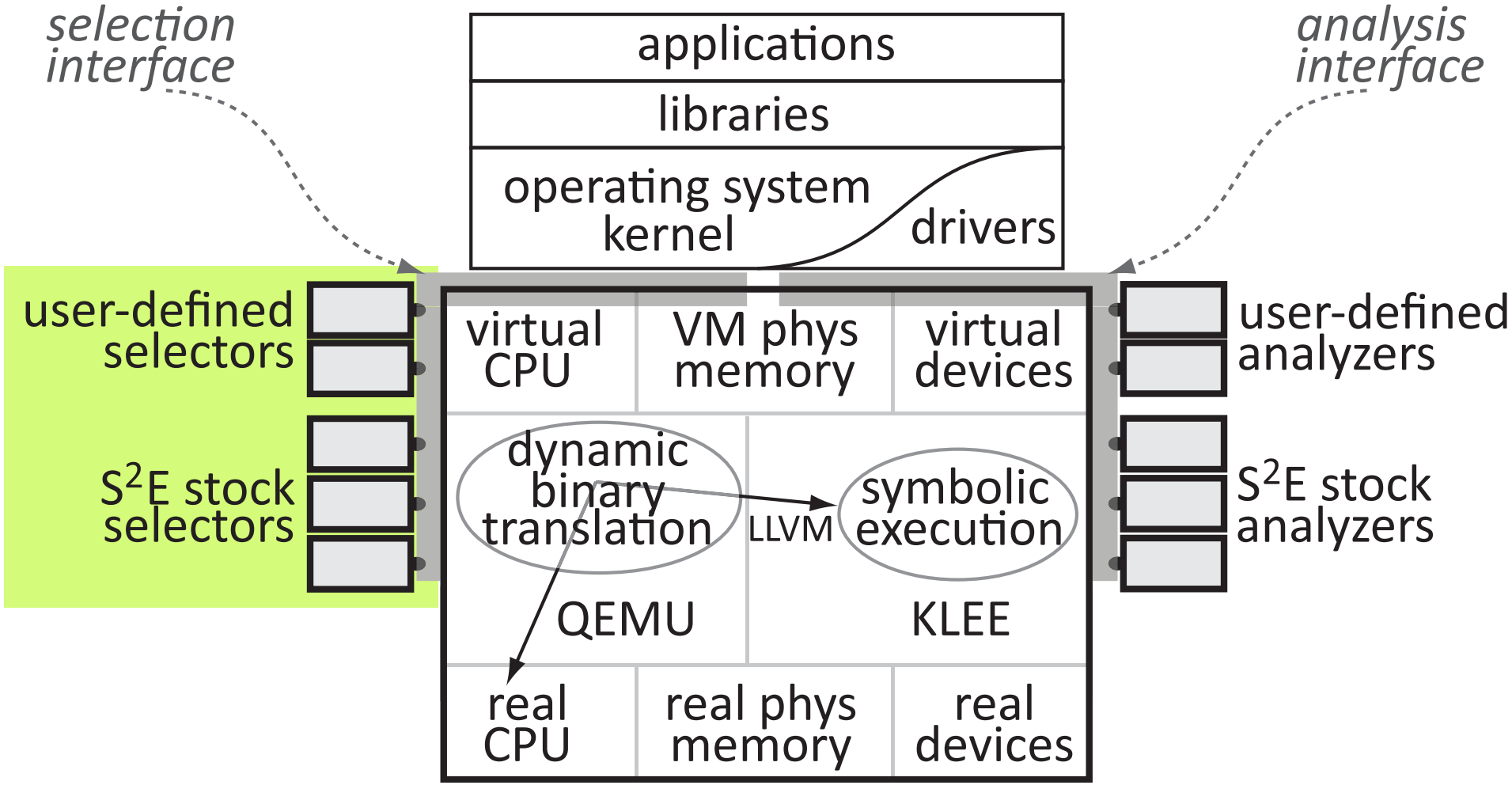
Avoids *unnecessary* concretization

Lazy+Selective Conversion

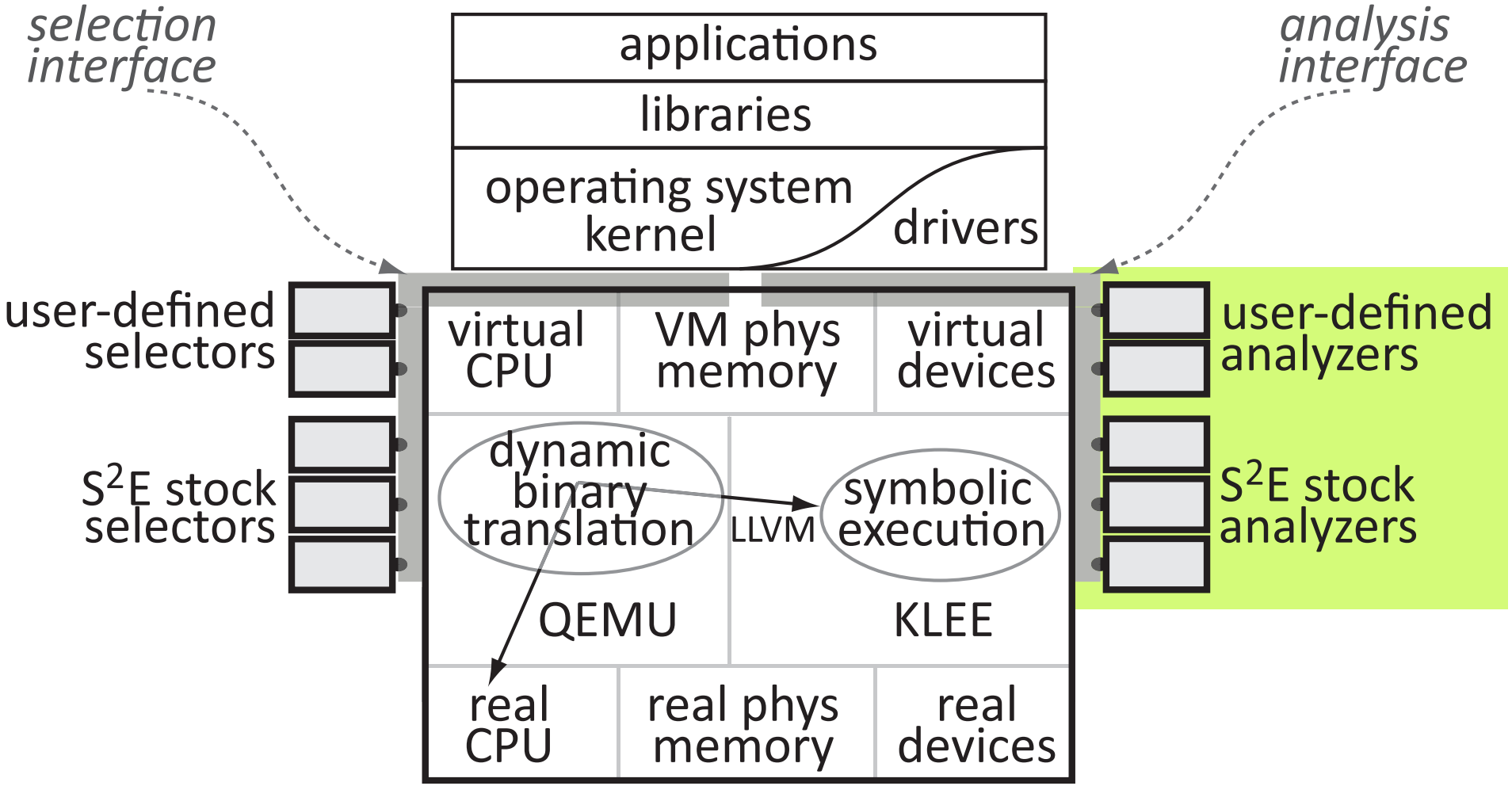


Avoids *unnecessary* concretization

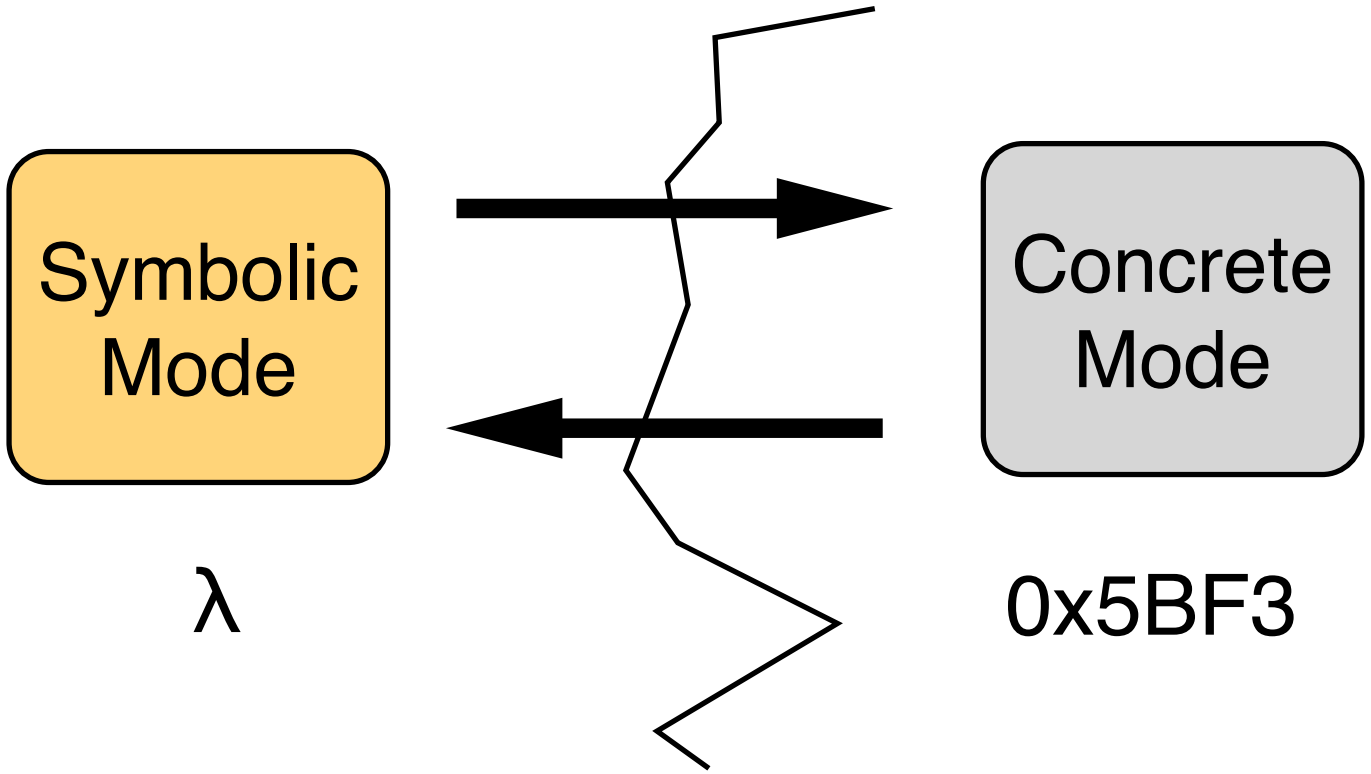
S2E User's View



S2E User's View



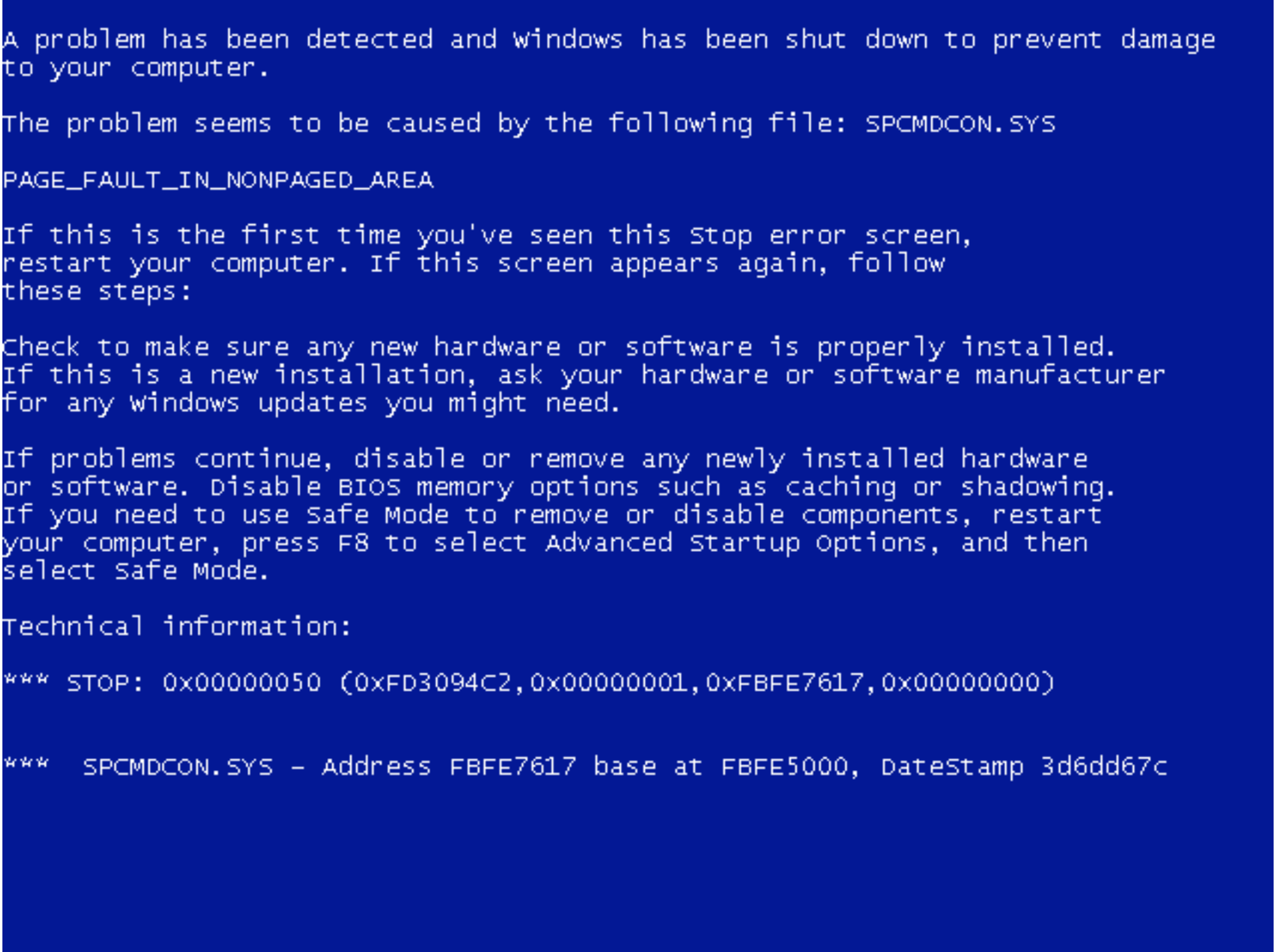
Key S²E Feature



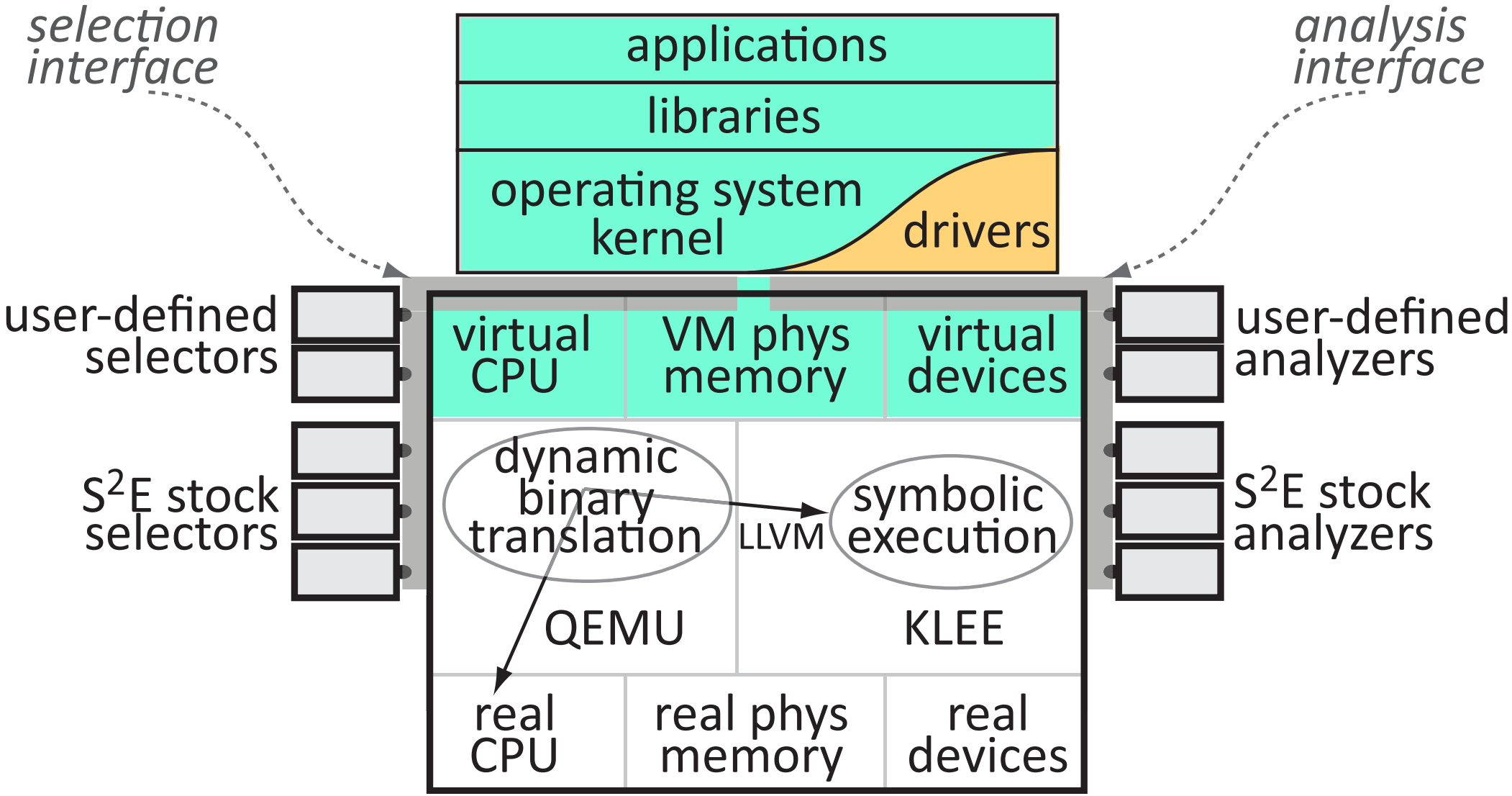
Weaves execution between symbolic/concrete transparently, efficiently, and consistently

Outline

- Cloud9
- S^2E Platform
  - *Concept: In-Vivo Multi-Path Analysis*
  - *Idea #1: Selective Symbolic Execution*
  - *Idea #2: Analysis-specific Execution Consistency*
  - *Implementation: The S^2E System*
  - → *Three Use Cases*
- Lessons for SMT/SAT Solvers


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
The problem seems to be caused by the following file: SPCMDCON.SYS
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)
*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

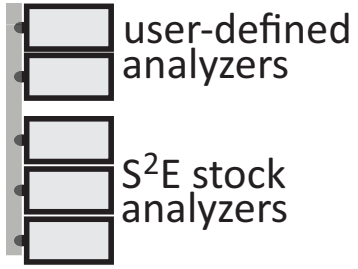
DDT+ Testing Tool



DDT+: Device Driver Tester

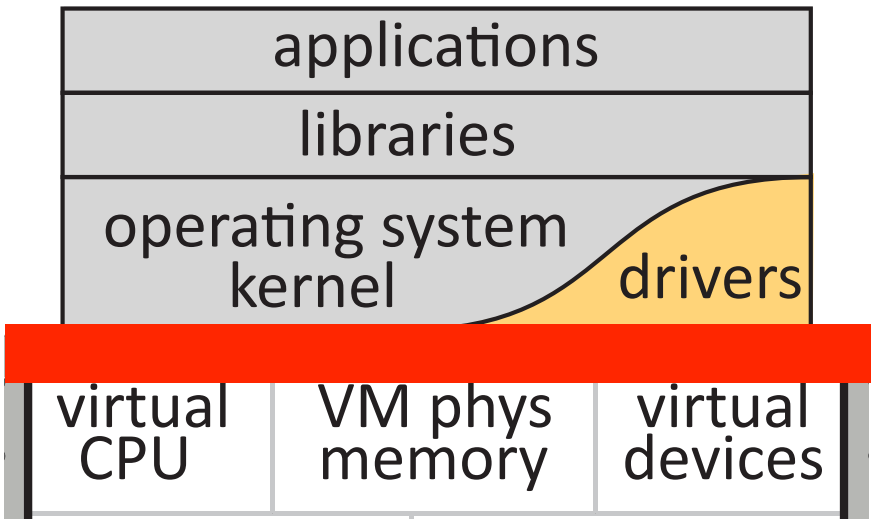
- Path exploration
  - *LC (local consistency) for OS/driver interface*
  - *RC (relaxed consistency) for driver/HW interface*

- Path Analysis
  - *Off-the-shelf <u>single-path</u> checkers*
  - *Our own VM-level analyzers*



Symbolic Hardware

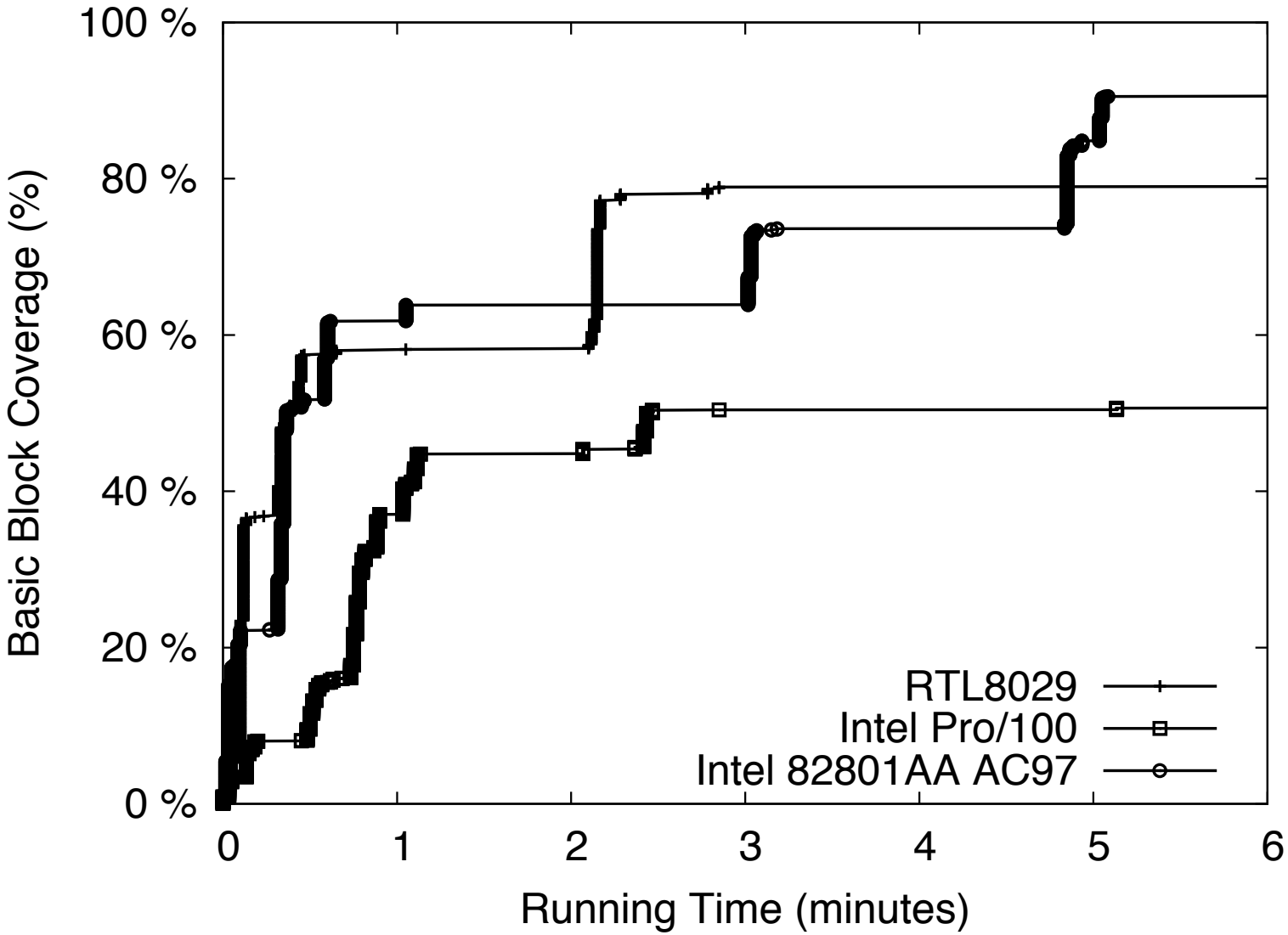
- Symbolic HW inputs, symbolic interrupts, etc.
- Test without having the real hardware
- Test for bad hardware behaviors



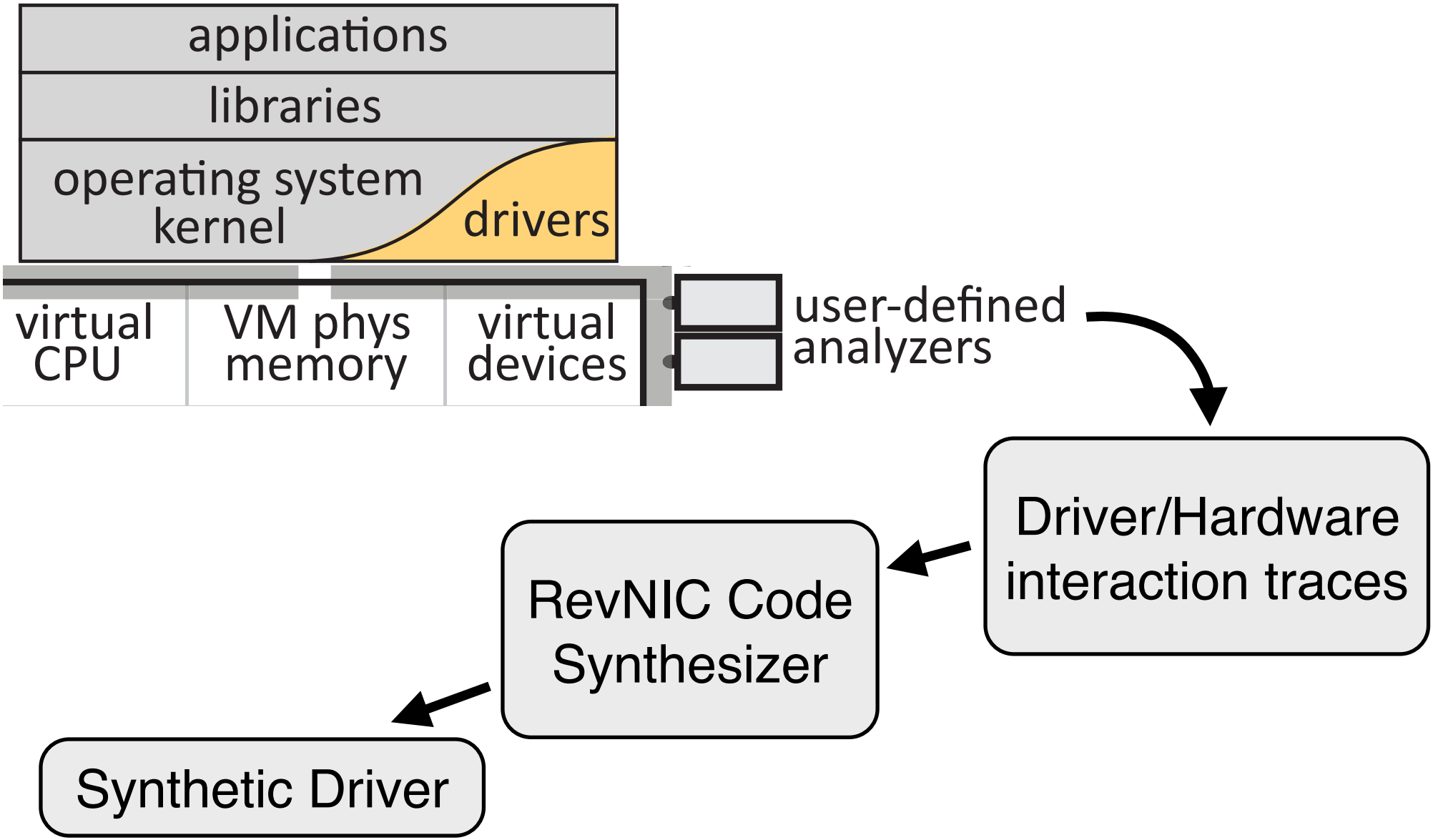
symbolic hardware

Tested Driver	Bug Type
RTL8029	Resource leak
RTL8029	Memory corruption
RTL8029	Race condition
RTL8029	Segmentation fault
RTL8029	Segmentation fault
AMD PCNet	Resource leak
AMD PCNet	Resource leak
Ensoniq AudioPCI	Segmentation fault
Ensoniq AudioPCI	Segmentation fault
Ensoniq AudioPCI	Race condition
Ensoniq AudioPCI	Race condition
Intel Pro/1000	Memory leak
Intel Pro/100 (DDK)	Kernel crash
Intel 82801AA AC97	Race condition

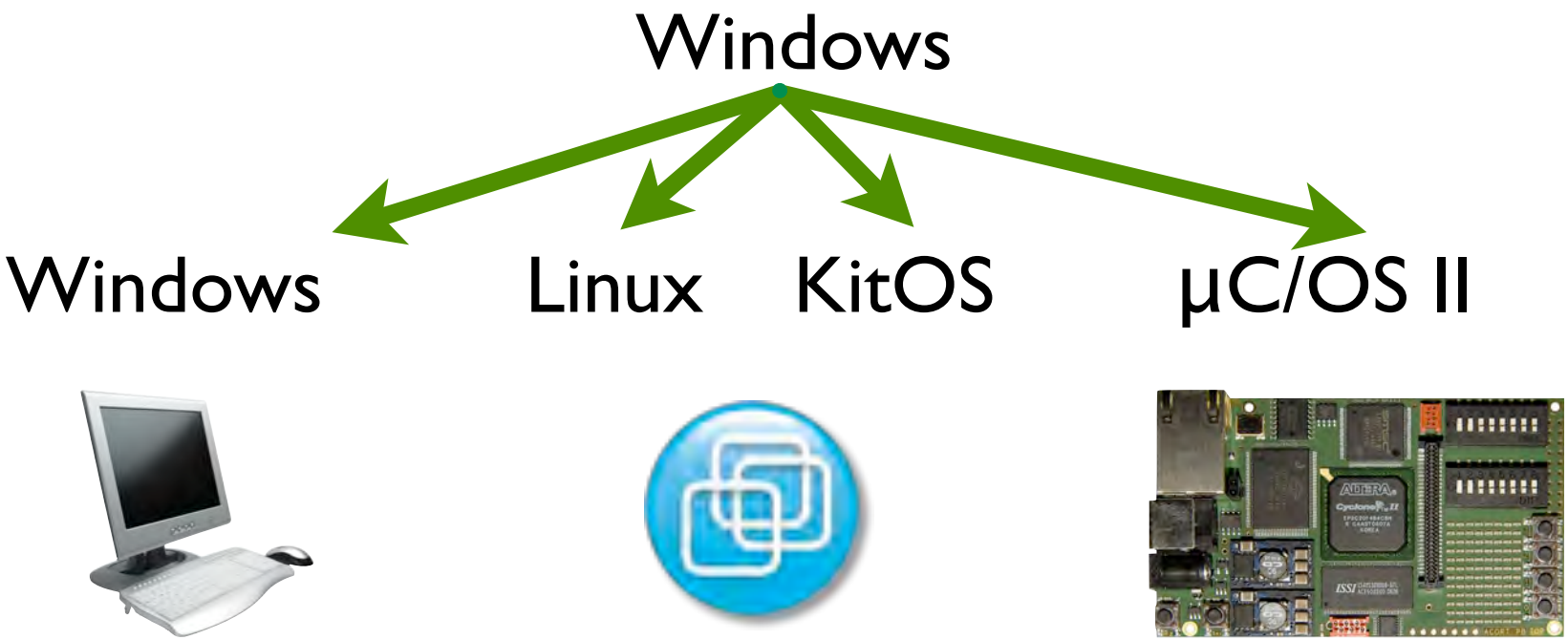
Analysis Time < 20 minutes



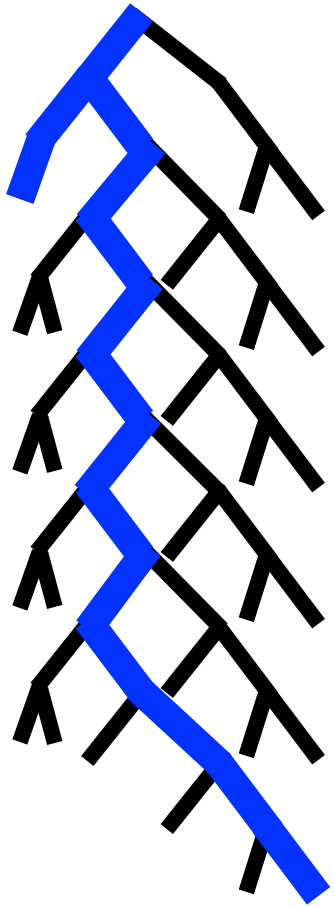
RevNIC+ Reverse Engineering



Automated Porting



PROFs: Performance Profiling



1,645 instructions

129,086 instructions

`ping`

- Apache URL parser
- Microsoft IIS SSL module
- Lua language interpreter
- Various utilities

S2E Improves Productivity

PROF$_S$

20 person-hours
767 lines of code

RevNIC$^+$

40 person-hours*
580 lines of code

DDT$^+$

38 person-hours
720 lines of code

S2E Platform

> 100,000 lines of code
47,000 lines of new code

S2E in a Nutshell

- Platform for building analysis tools that are multi-path and in-vivo
- Idea #1: Selective symbolic execution
- Idea #2: Execution consistency models

http://s2e.epfl.ch

Ready-to-use VM, demos, tutorials, source code, documentation

Outline

- Cloud9
  - *Parallel symbolic execution platform*
- S^2E
  - *Selective symbolic execution platform*

→ Lessons for SMT/SAT Solvers

  - *Real data from running on real systems*

Experiences with Constraint Solving

Evaluation Setup

GNU Coreutils

2/4500/620 LOC

1h x 96 tools

python™

380 KLOC

1h x 48 workers

Cloud 9

Symbolic execution engine (Klee)

Constraint cache & simplifications

STP

MiniSAT

Original Path Constraints

Simplified Path Constraints

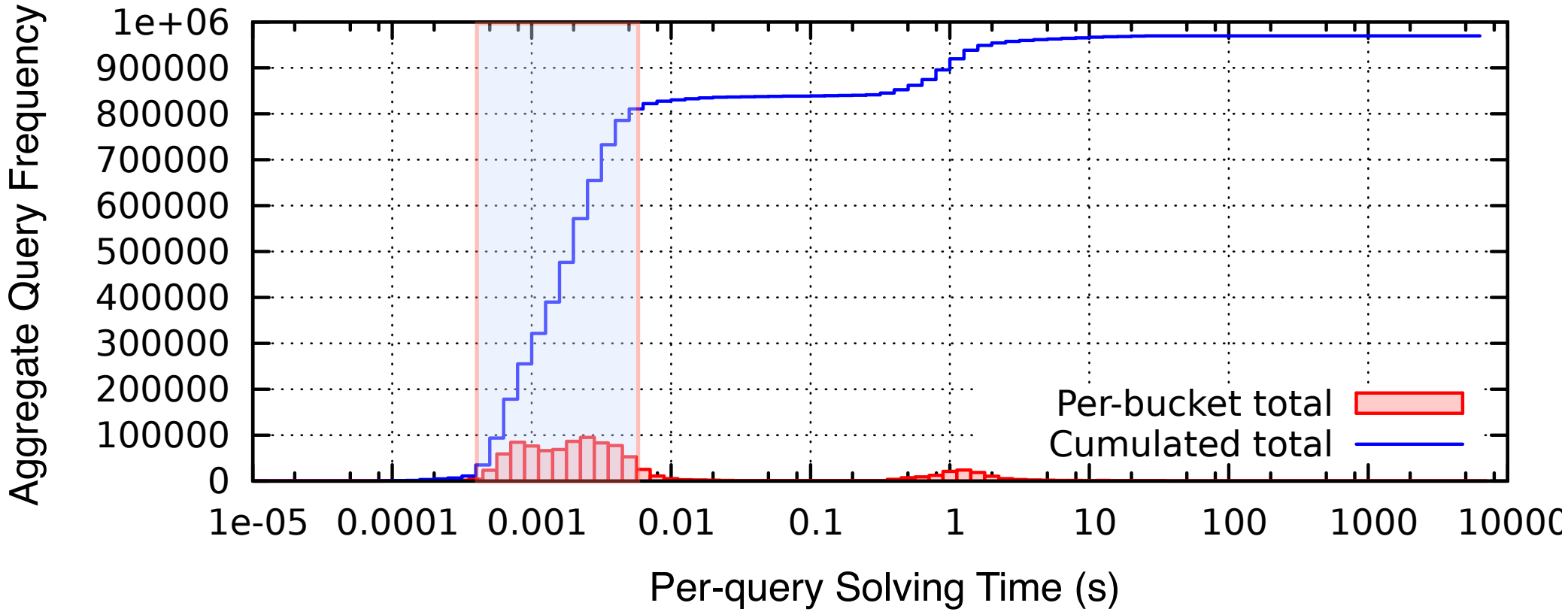
SAT representation

Queries & Statistics

CS in Numbers

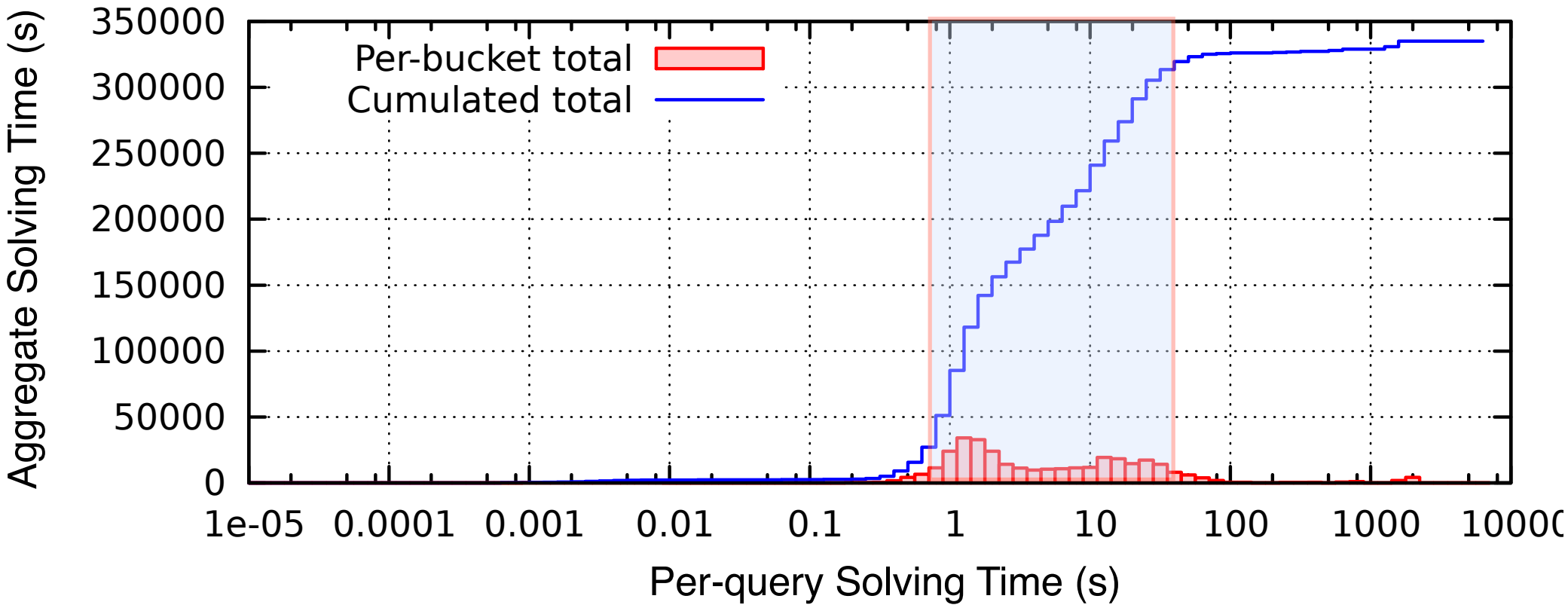
- Solving time characteristics
- STP time vs. SAT time
- CPU profile

Query Frequency Distribution Among Per-Query Solving Time for Coreutils



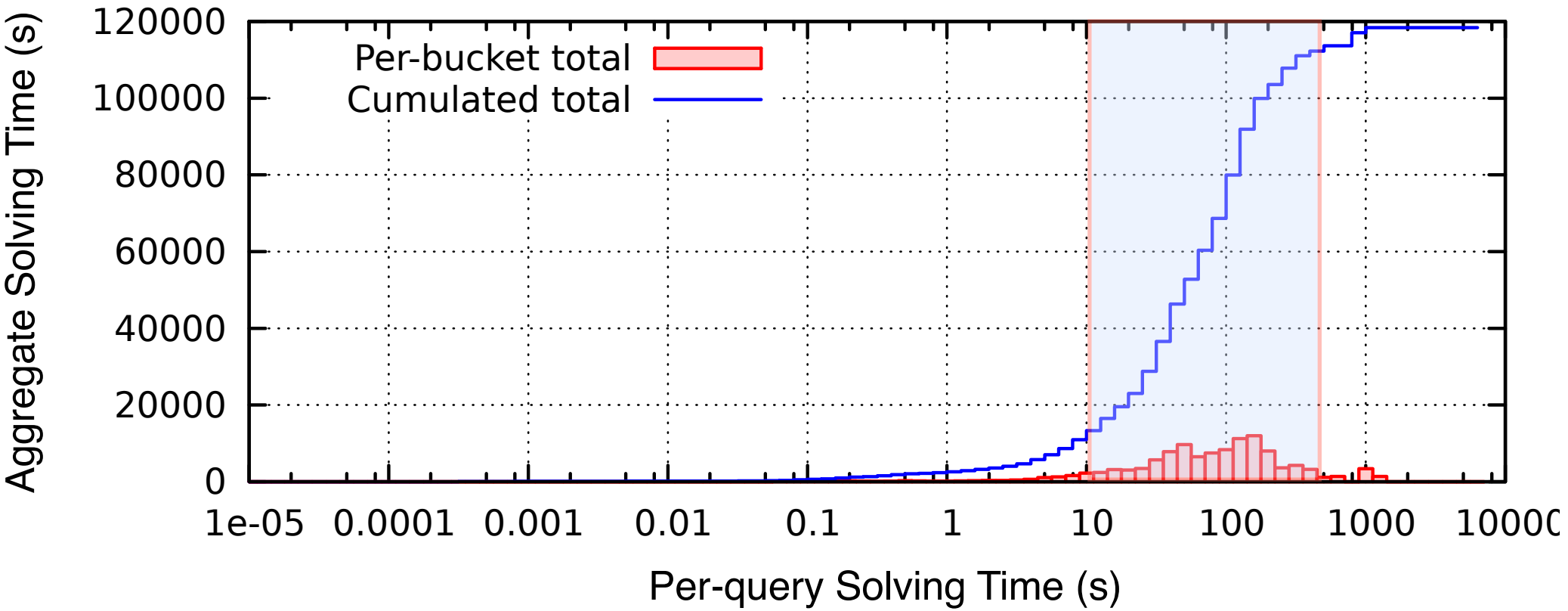
The small (<0.1s) queries are most numerous...

Aggregate Solving Time Distribution Among Per-Query Solving Time for Coreutils



... but queries >1s dominate total solving time

Aggregate Solving Time Distribution Among Per-Query Solving Time for Python

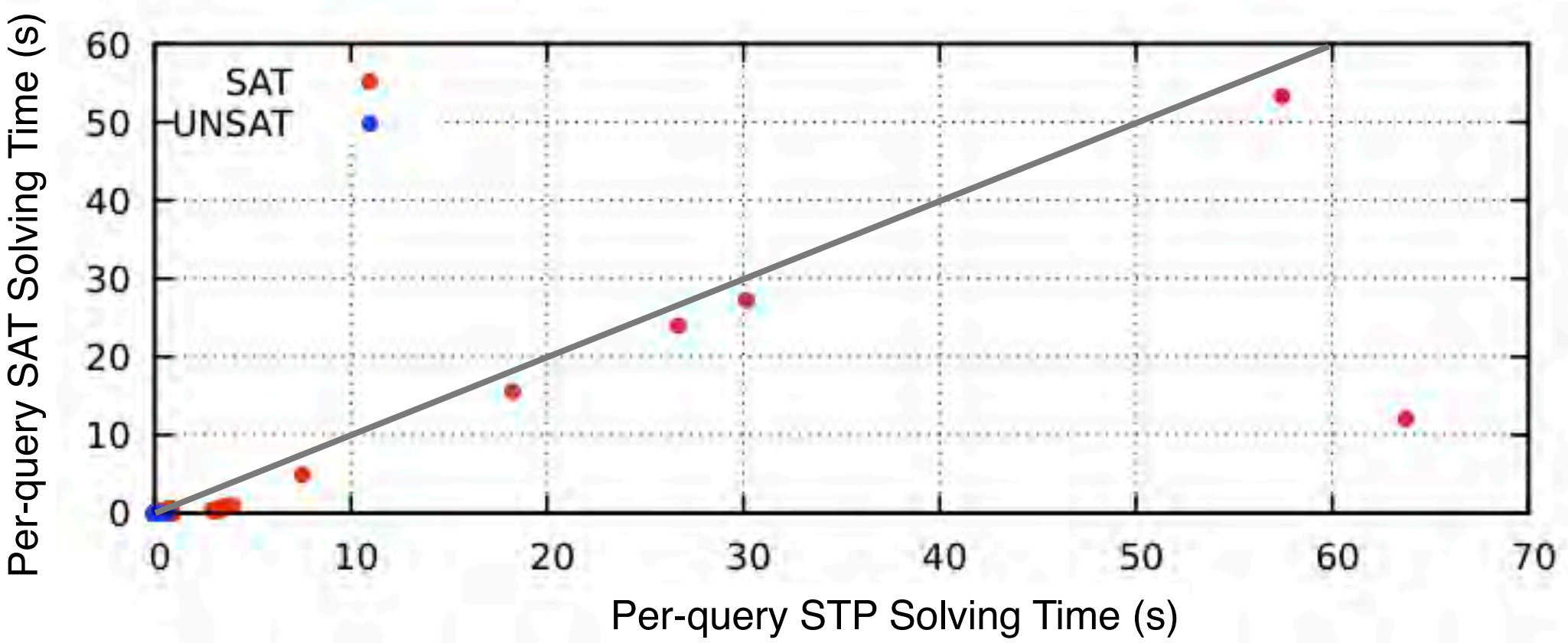


Program nature generates more complex constraints

CS in Numbers

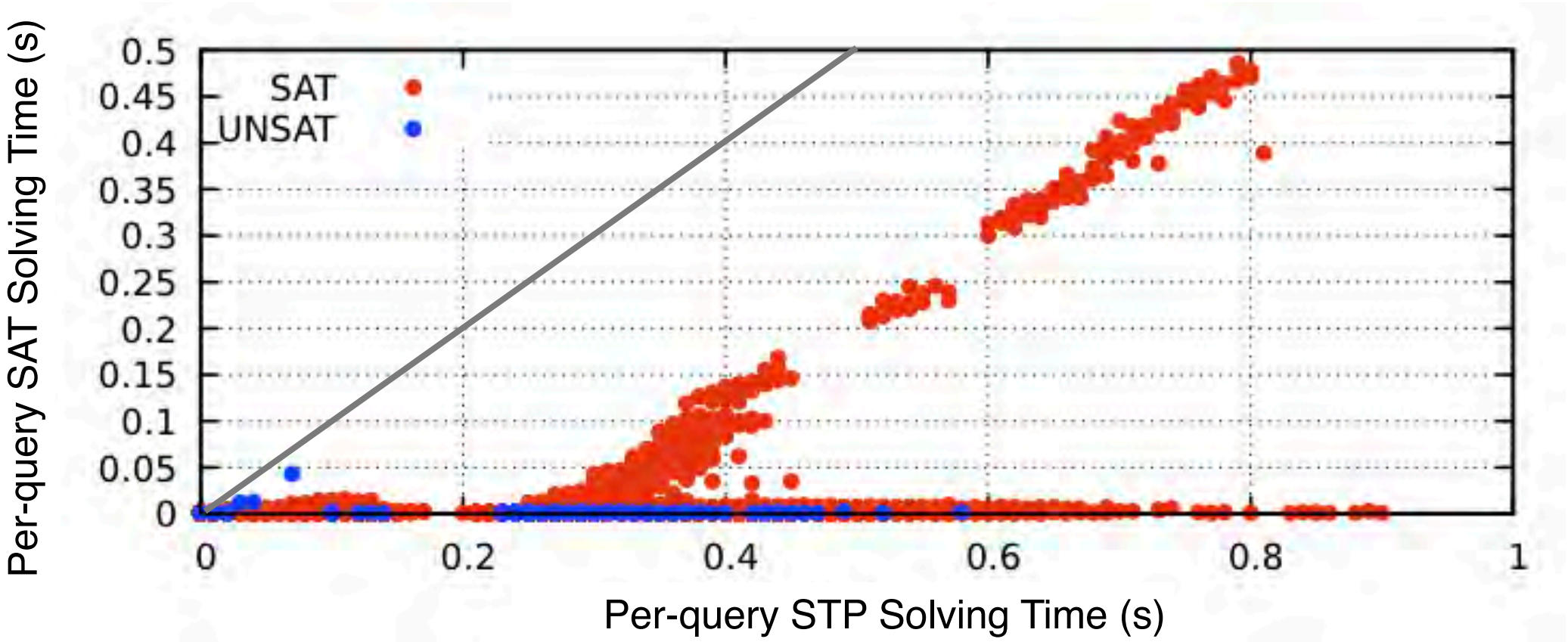
- Solving time characteristics
- STP time vs. SAT time
- CPU profile

Total STP Time - MiniSat Time Correlation For All Queries



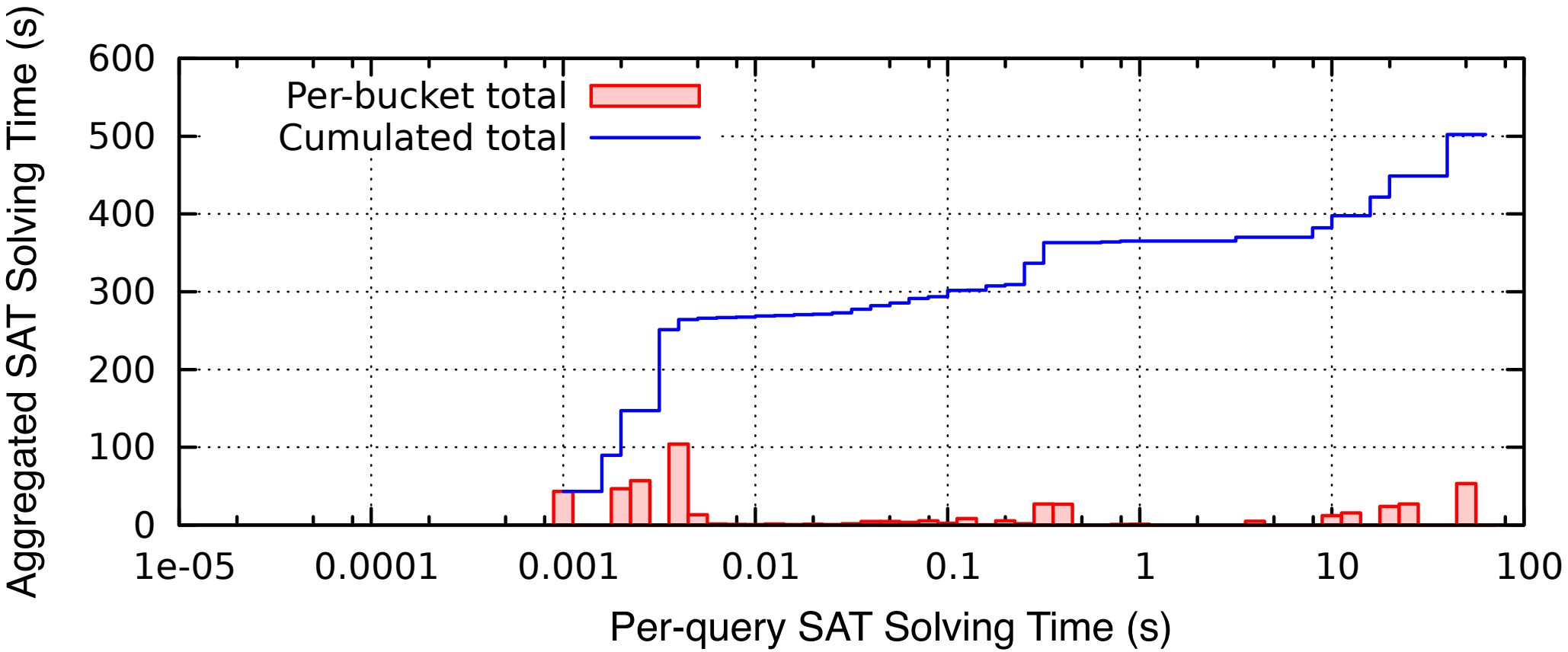
SAT operations dominant for large queries

Total STP Time - MiniSat Time Correlation For Queries <1s



Non-SAT operations dominant for small queries

Aggregate SAT Time Distribution Among Per-Query SAT Time



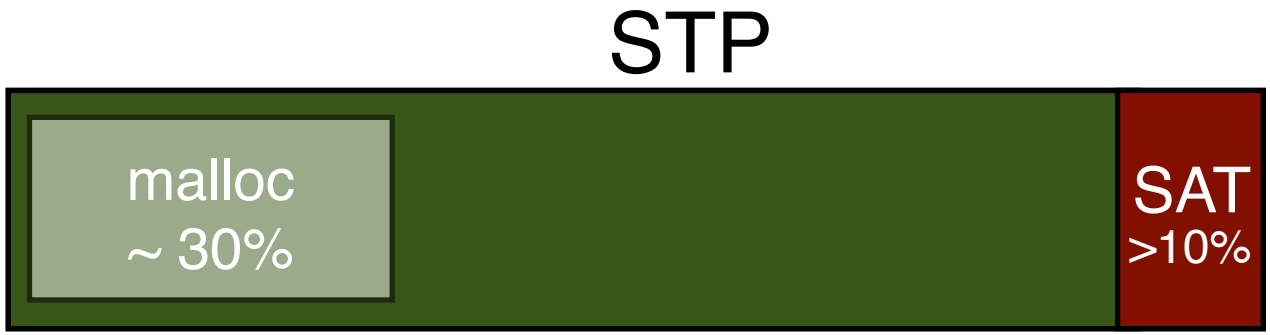
No particular bottleneck

CS in Numbers

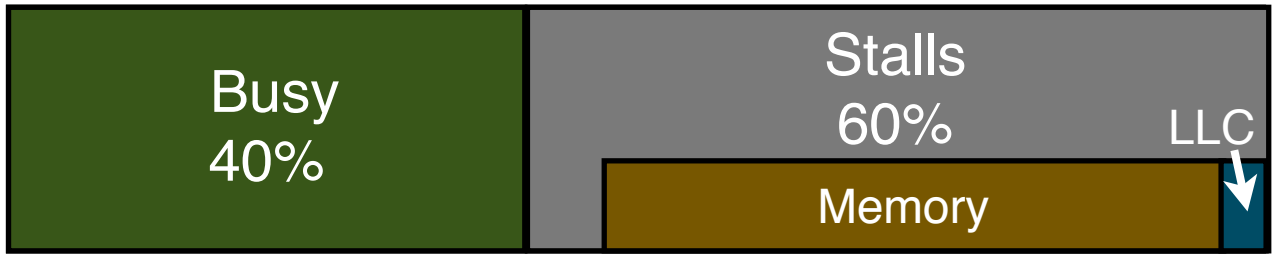
- Solving time characteristics
- STP time vs. SAT time
- CPU profile

CPU Profile

Function Profile



CPU Architecture Profile



Conclusions

- Target expensive (>1 s) queries for improving solving time
- Optimize SAT solver for expensive queries
- Low-level optimizations throughout the code

Outline

- Cloud9 ✓
  - *Parallel symbolic execution platform*
- S^2E ✓
  - *Selective symbolic execution platform*
- Lessons for SMT/SAT Solvers ✓
  - *Real data from running on real systems*

Acknowledgments

- Cloud9
  - *Vlad Ureche, Cristian Zamfir*
- S2E
  - *Vitaly Chipounov, Vova Kuznetsov*
- STP Measurements
  - *Ayrat Khalimov, Istvan Haller*