

Automated Software Reliability

George Candea

*School of Computer & Communication Sciences
EPFL (Lausanne, Switzerland)*



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

**We are forced to take
the reliability of software
on faith**

```
010011010011
110101001010
010010000001
010111010011
110110010110
011011000011
```

RTL8029.SYS



Computer image courtesy of the Hudson Library & Historical Society

Faith-based Industry

- Programmers write code, then mostly **hope** for the best
- Users **trust** software providers to test the code thoroughly
- Cannot **assess** the reliability of a software system before using it

Systems vs. Programs

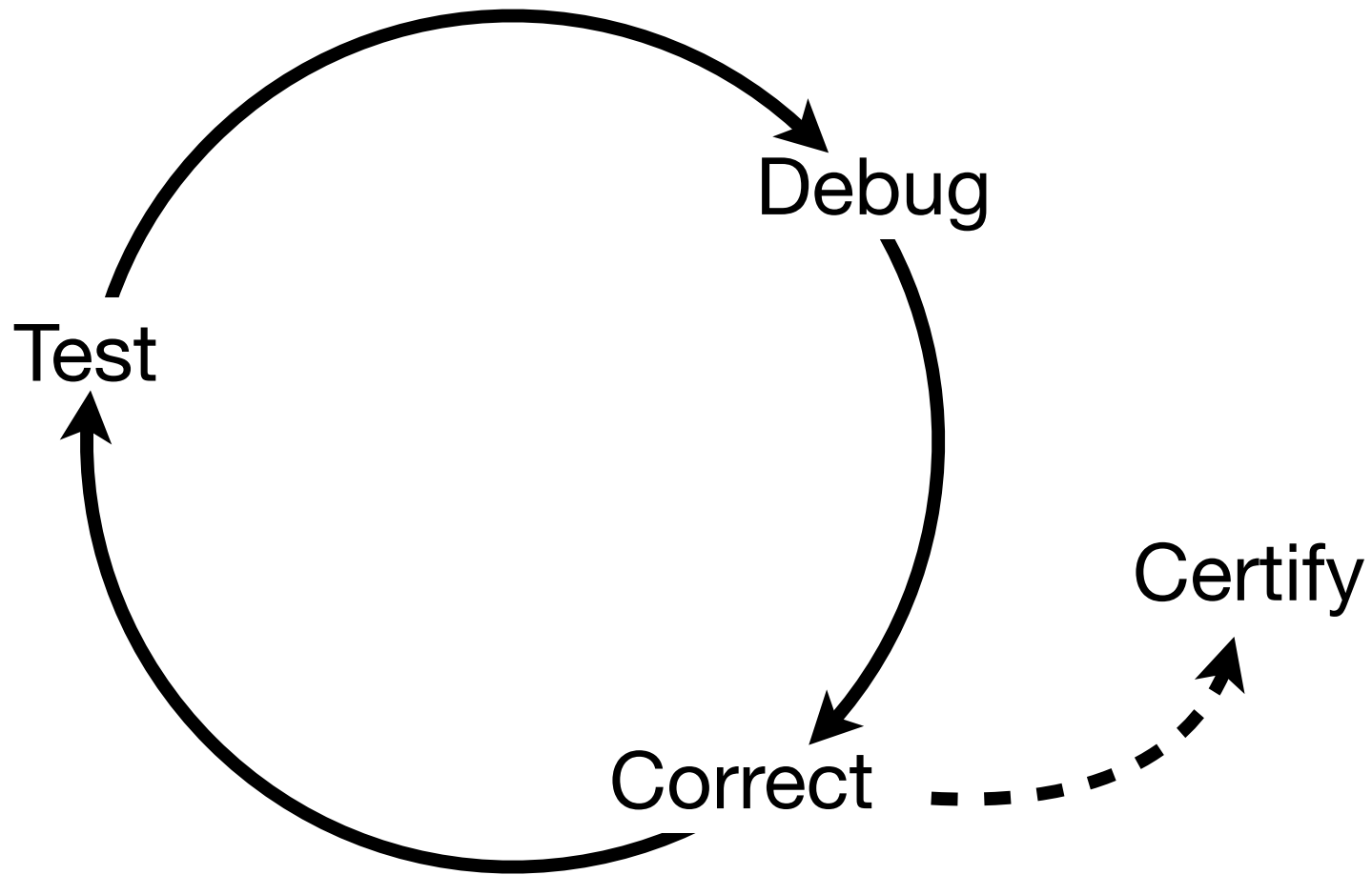
- We know how to build *small* things that work well
- But a real-world system is ...
 - complex – millions of lines of code (MLOC)
 - written by 100s of programmers in many languages
 - many threads running in parallel
 - implicit, vague specifications

Systems vs. Programs

- We know how to build *small* things that work well
- But a real-world system is ...
 - complex – millions of lines of code (MLOC)
 - written by 100s of programmers in many languages
 - many threads running in parallel
 - implicit, vague specifications

Bug-free programs \nRightarrow Bug-free systems





Automation

	<i>Automated Testing</i>	<i>Automated Debugging</i>	<i>Automated Correcting</i>	<i>Scalable Certification</i>
Tools & Systems	DDT Test proprietary code	ESD Execution synthesis	Dimmunix Deadlock immunity	iProve End-user verifiability
	LFI Test recovery code	Portend Data race classifier	RevNIC Reverse engineering	iQA Rating & certification
	ConfErr & WebErr Human error testing			
Core Techs	Cloud9 - Cluster-based parallel symbolic execution			
	S²E - Selective symbolic execution of full software stacks			

Outline

1. The S²E Platform

- *Background*
- *S²E: The Theory*
- *S²E: The System*

2. Three Use Cases

3. Automated Software Reliability Services (AutoSRS)

**S²E = 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
```

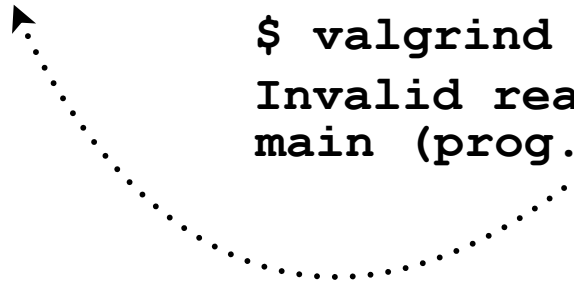
```
OK
```

```
$ ./prog ABC
```

```
Segmentation fault
```

```
$ valgrind ./prog ABC
```

```
Invalid read of size 1
main (prog.c:4)
```



Other Examples

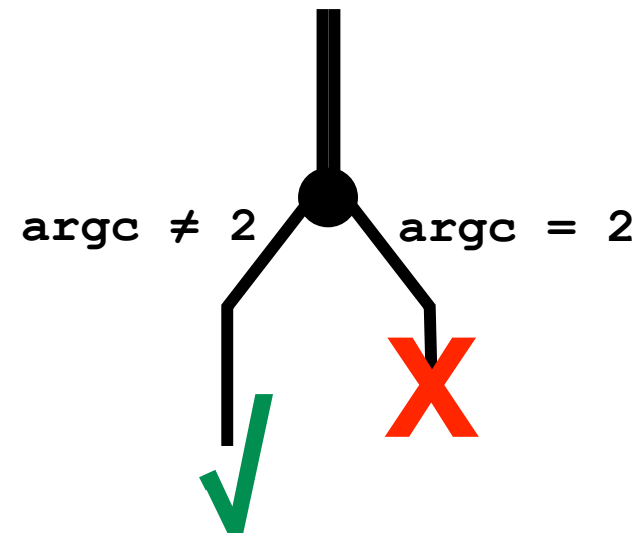
- Security analysis
- Program verification
- Performance profiling
- ...

**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



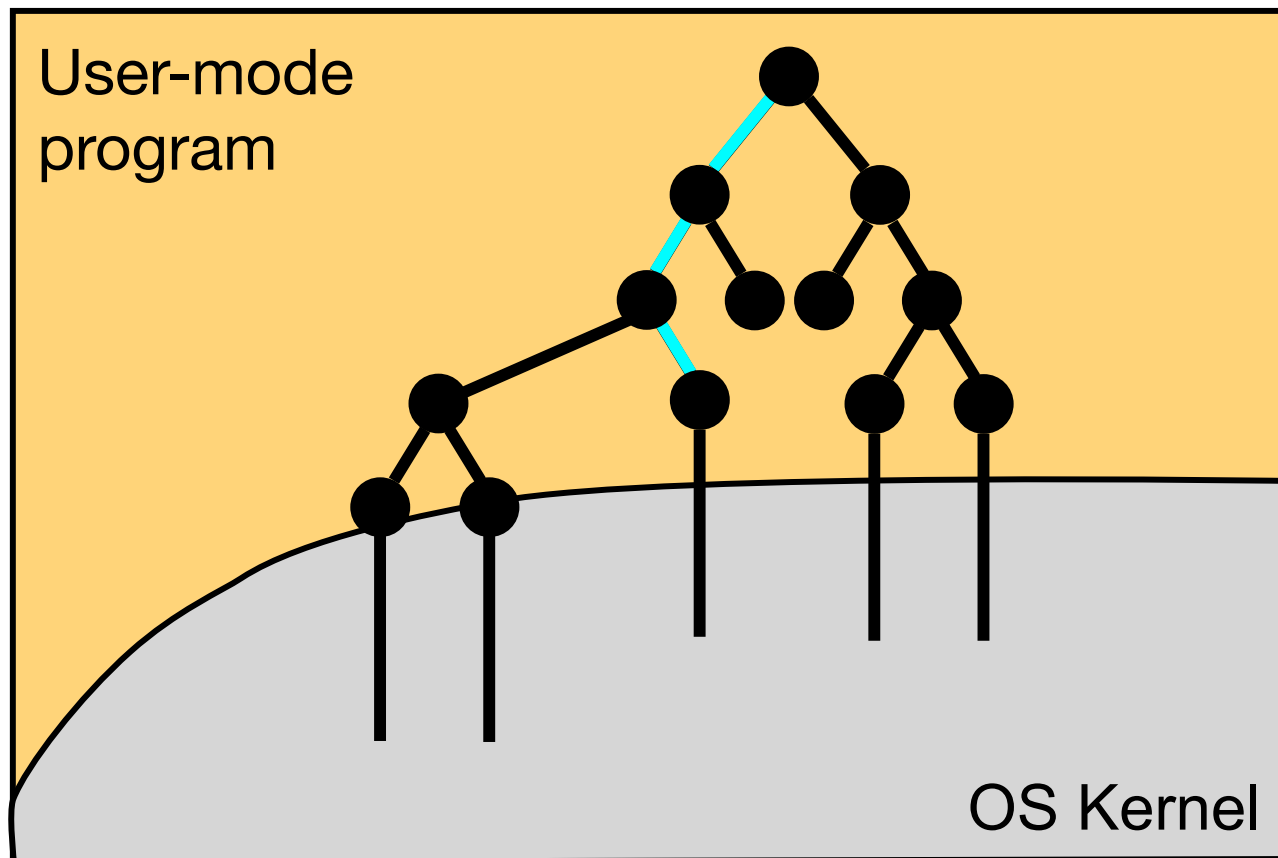
In Vitro



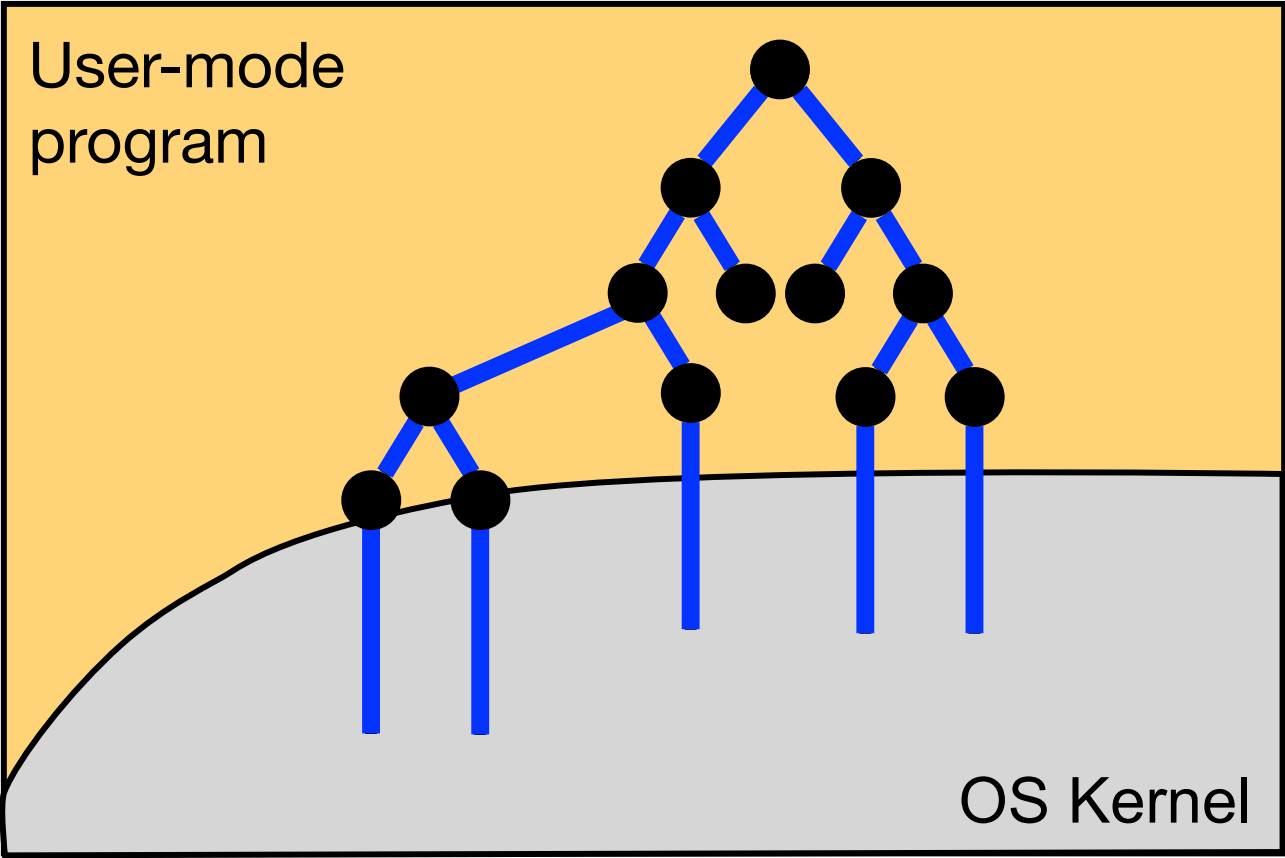
In Vivo

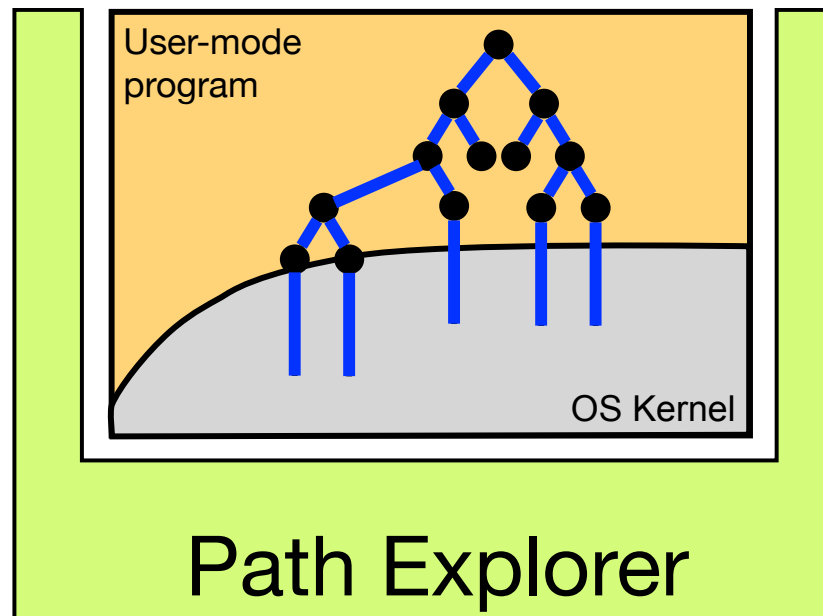
Analyze entire software stack

Single-Path In-Vitro



~~Single-Path In-Vitro~~
Multi-Path In-Vivo





**S²E platform =
path exploration + path analysis**

Challenge: Path Explosion

of paths $\approx 2^{\text{system size}}$

- Cannot analyze all paths \Rightarrow select only some
 - which paths you choose can make a big difference
 - S²E enables making this choice ***analysis-specific***

Outline

1. The S²E Platform

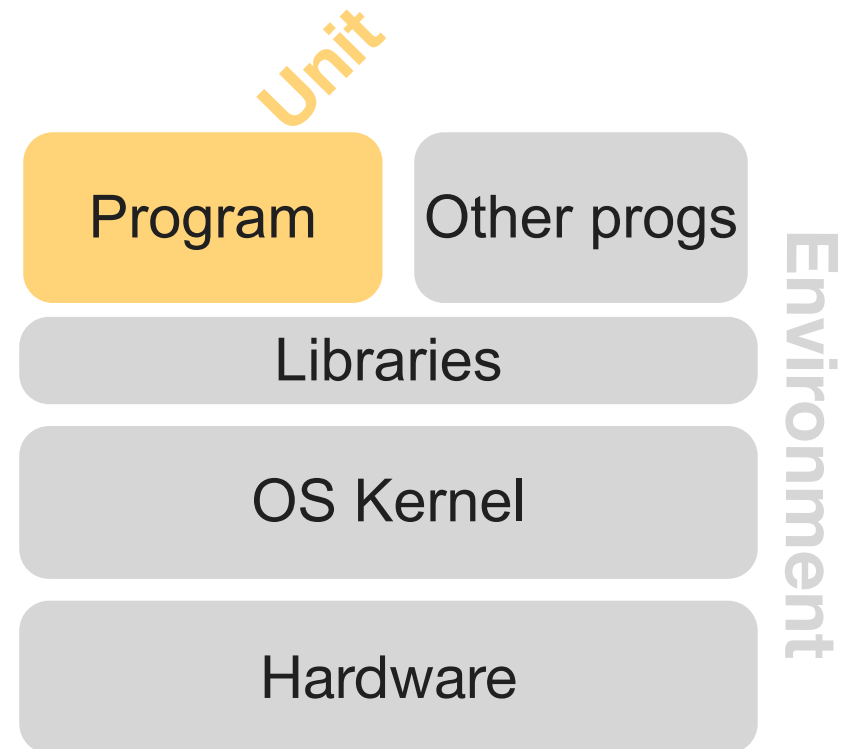
- *Background*
- • *S²E: The Theory*
- *S²E: The System*

2. Three Use Cases

3. Automated Software Reliability Services

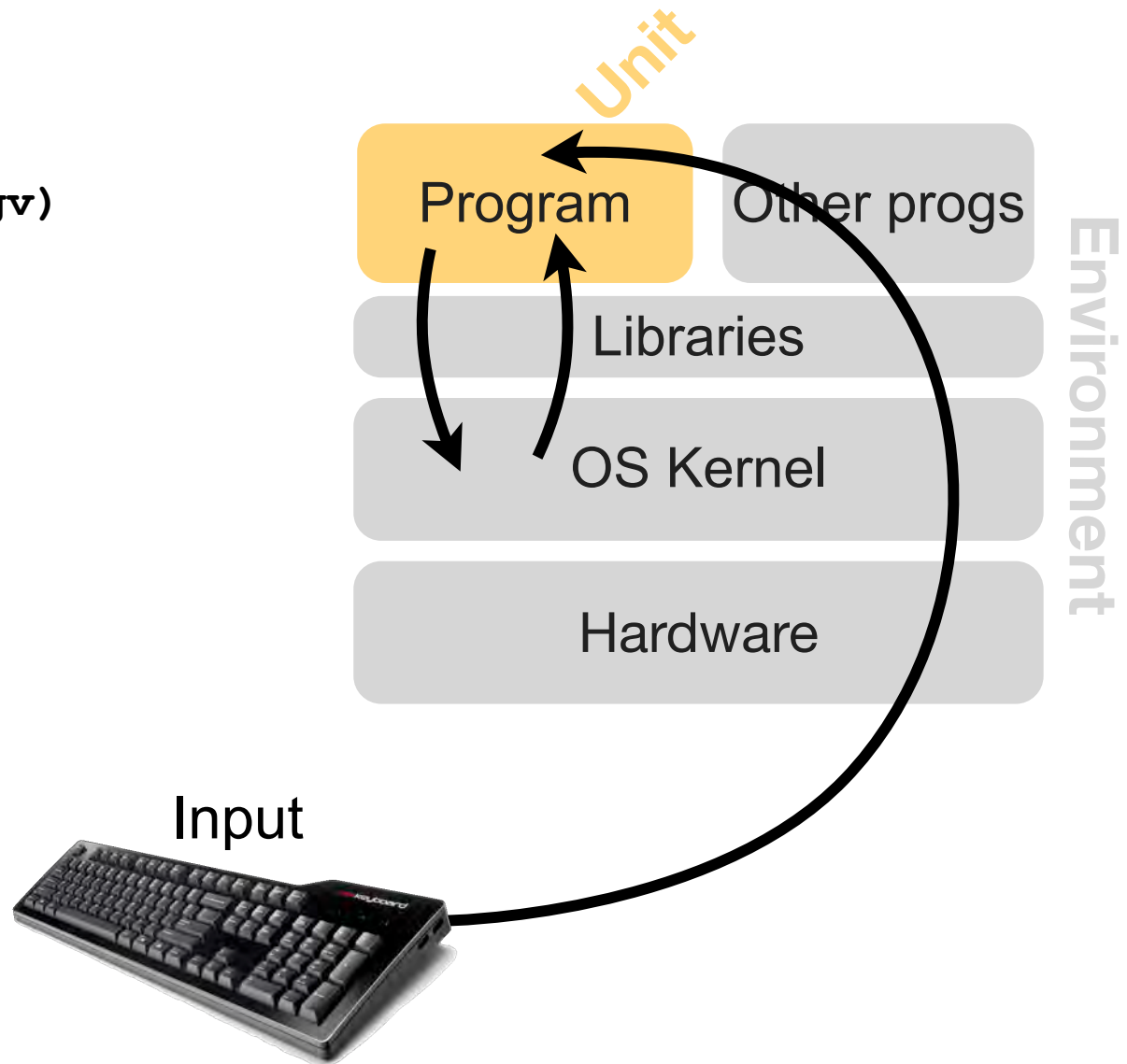
Unit vs. Environment

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

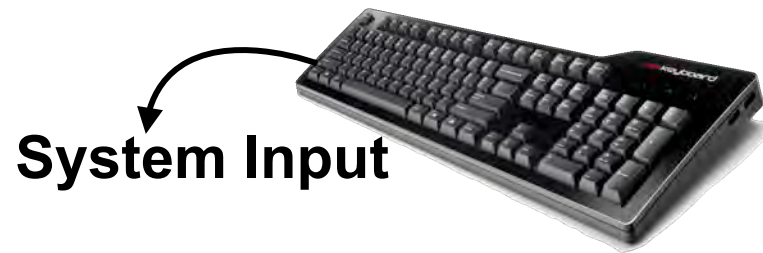


Unit vs. Environment

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



Thorough Automated Testing

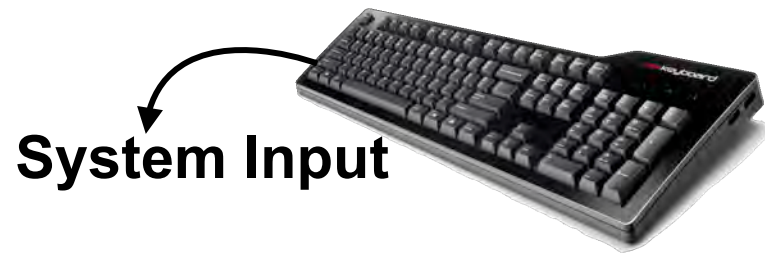


```
int main(argc, argv) {  
    if (argc == 0) {  
        ...  
    }  
    p = malloc(...);  
    if (p == NULL) {  
        ...  
    }  
    ...  
}
```

Unit

Environment

Thorough Automated Testing



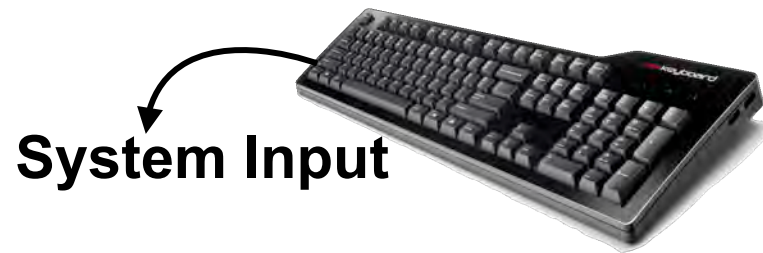
```
int main(argc, argv) {  
    if (argc == 0) {  
        ...  
    }  
    p = malloc(...);  
    if (p == NULL) {  
        ...  
    }  
    ...  
}
```

Unit



Environment

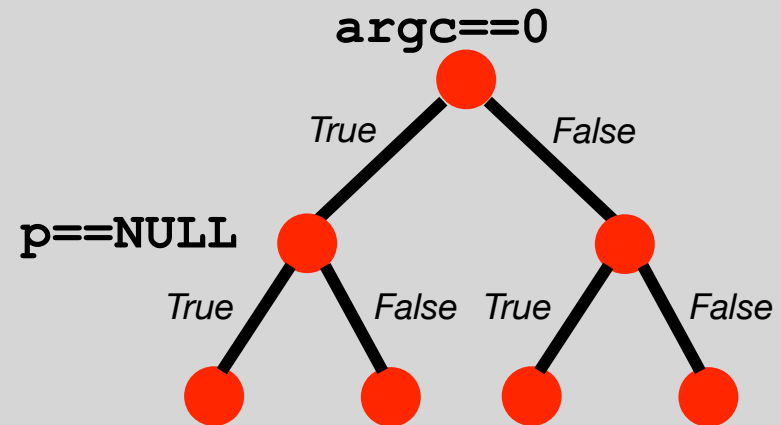
Thorough Automated Testing



Zero false negatives
Zero false positives

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

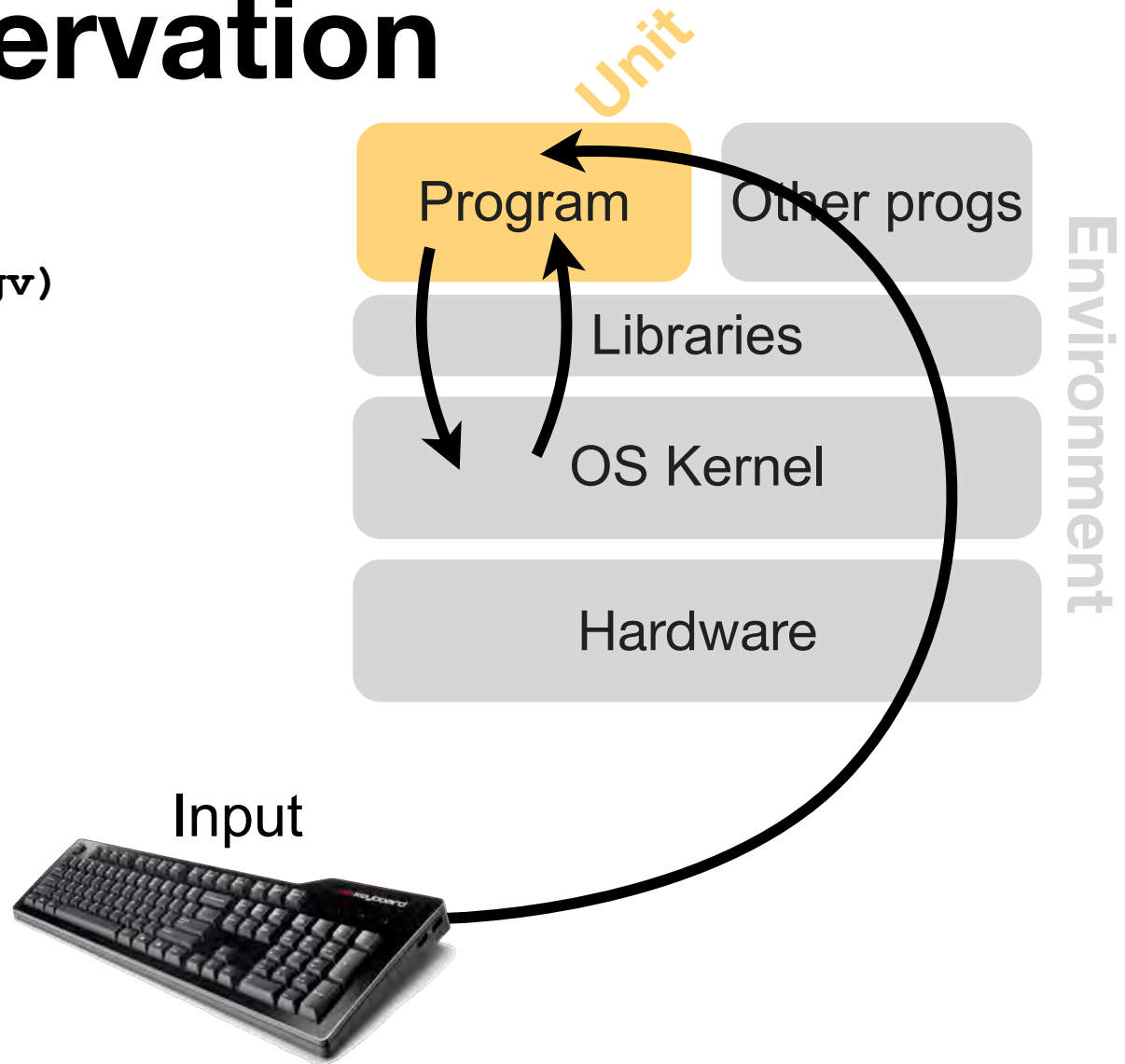
Unit



Environment

Key Observation

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



Provide illusion of full-system analysis

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

- Allow some paths to clobber other paths' state
 - *happens when environment changes are not isolated*
 - *occurs in test generation systems (DART, EXE, ...)*
- Use models of the environment
 - *models (by definition) exhibit behavioral differences*
 - *model checkers, symbolic exec engines (SLAM, KLEE, ...)*

In S²E, each path has its own real environment

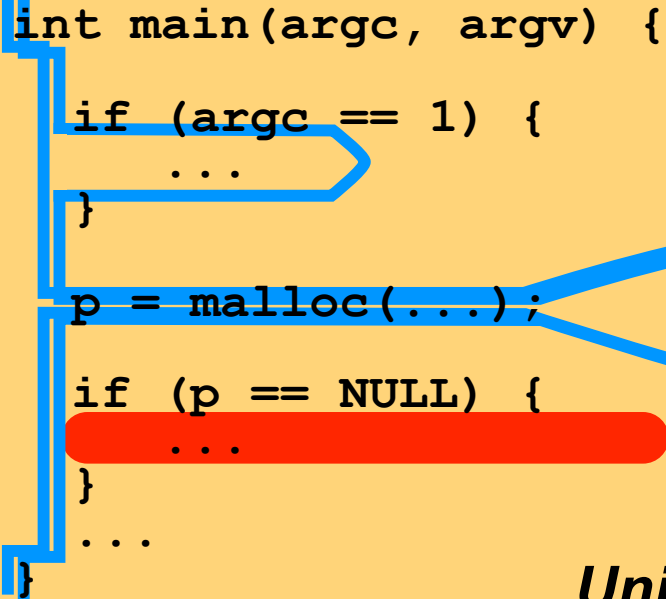
Execution Consistency Models

- Execution Consistency Model = set of paths
 - *i.e., abstract specification of which paths to analyze*
 - *i.e., grammar describing the paths of interest*
- Execution is consistent iff its path belongs to ECM set
- Not the same as memory consistency models
 - *but similar in spirit*

SC-UE (SC Unit-level Exec)

Much fewer paths
False negatives ?

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



Unit

Environment

Relaxed Consistency (RC)

Unit Input

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

Unit

Environment

Relaxed Consistency (RC)

No false negatives

Unit Input

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

Unit

Multi-valued return

$p' \in \{p, \text{NULL}\}$



Environment

Relaxed Consistency (RC)

No false negatives
False positives ?

Unit Input

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

Unit

Multi-valued return

$p' \in \{p, \text{NULL}\}$

Environment

Local Consistency (LC)

Unit Input

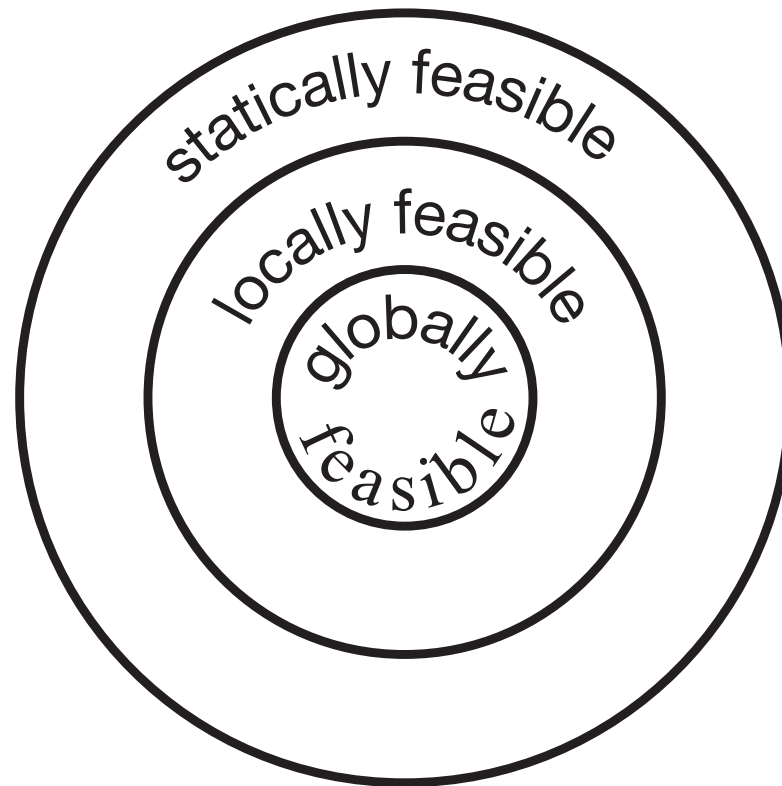
```
int main(argc, argv) {  
    if (argc == 1) {  
        ...  
    }  
  
    p = malloc(...);  
  
    ...  
}
```

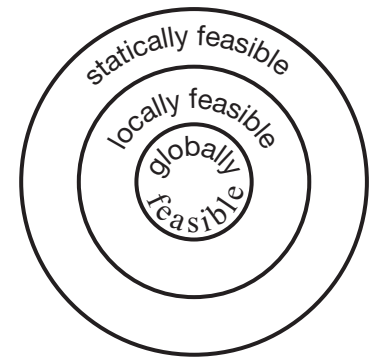
Unit

Interface annotation

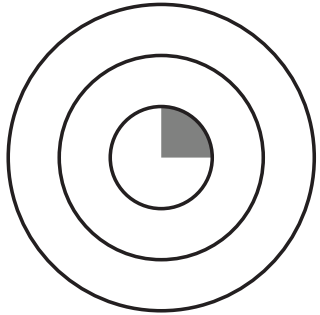
`malloc()` → {p, NULL}

Environment

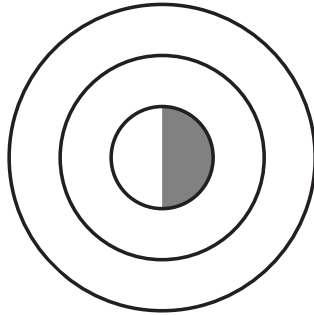




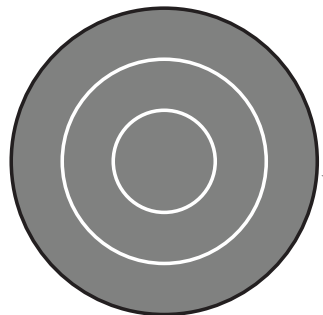
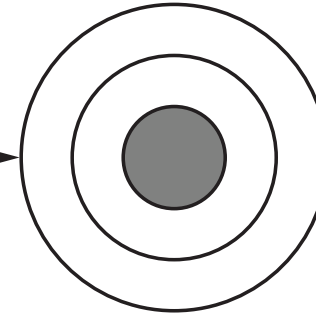
SC-CE
Strictly consistent
concrete execution



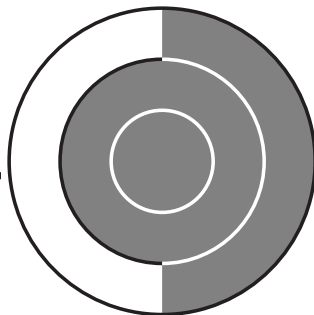
SC-UE
Strictly consistent
unit-level execution



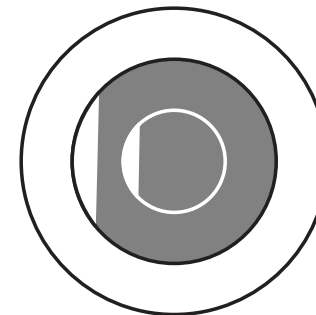
SC-SE
Strictly consistent
system-level execution



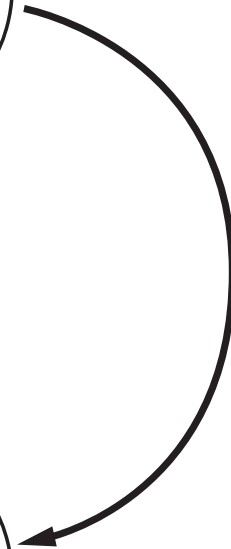
RC-CC
CFG consistency

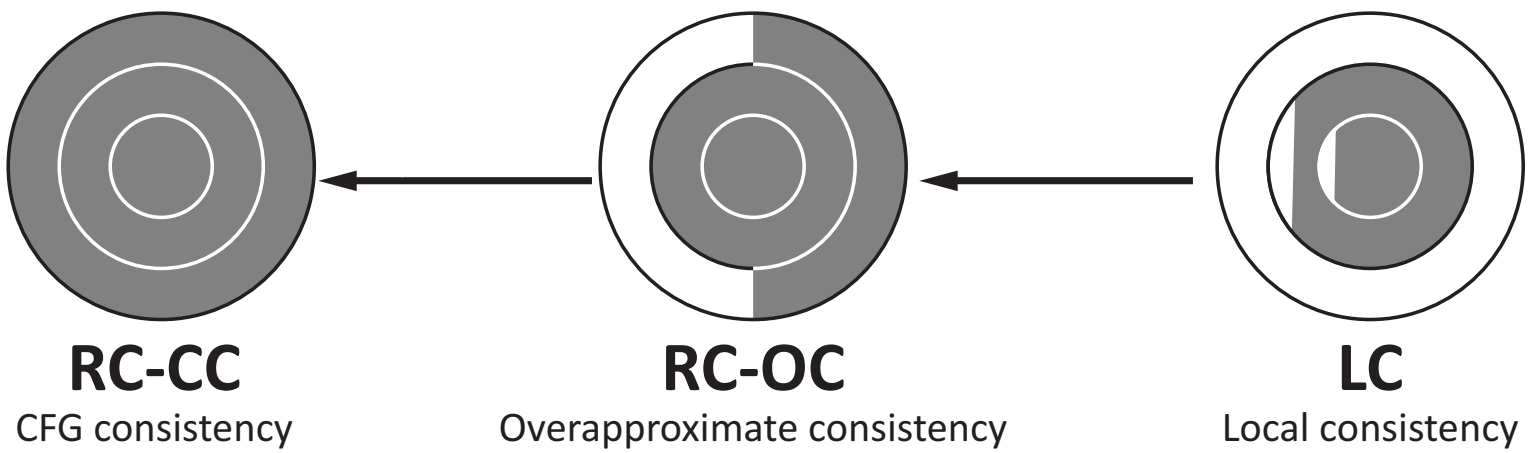
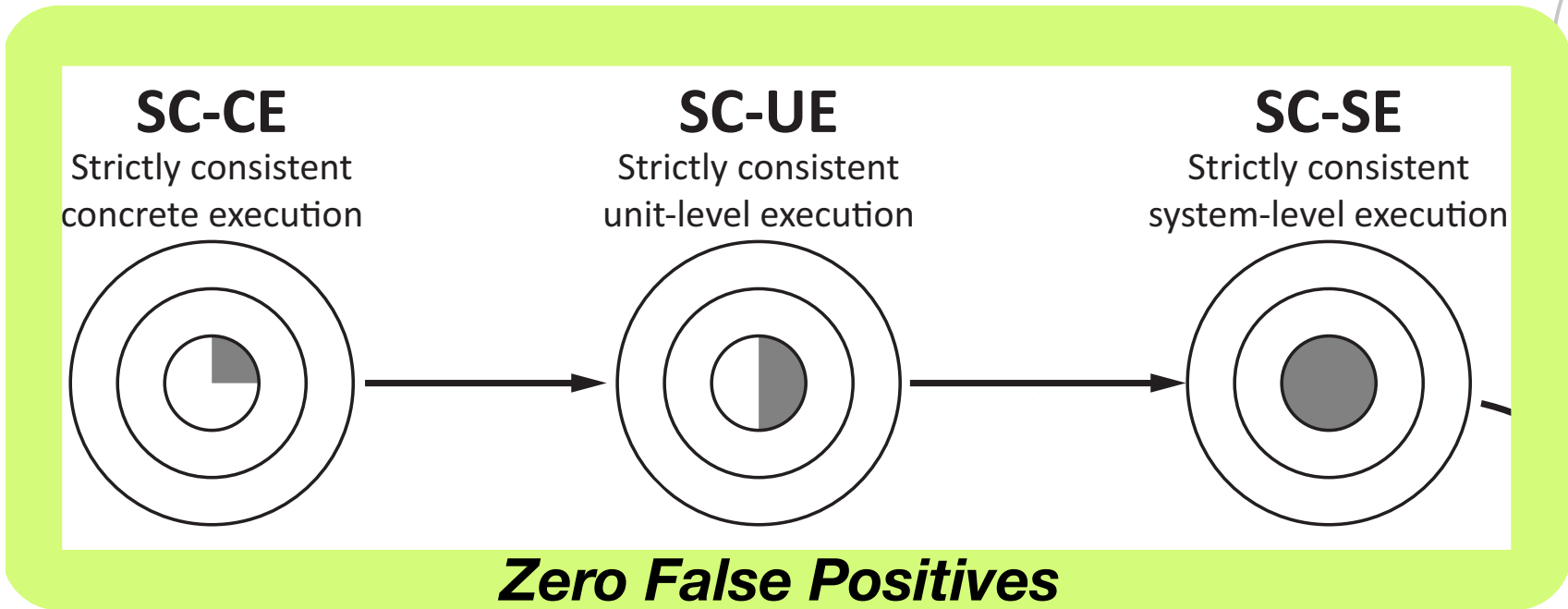


RC-OC
Overapproximate consistency



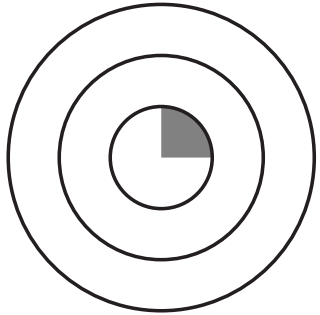
LC
Local consistency



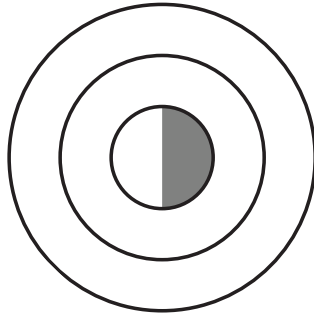




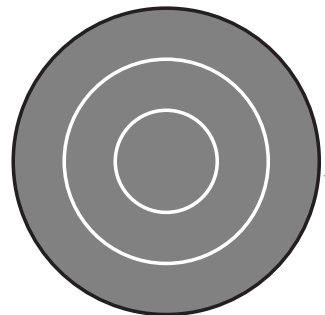
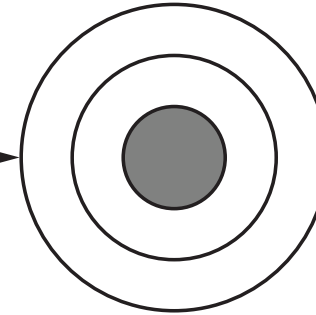
SC-CE
Strictly consistent
concrete execution



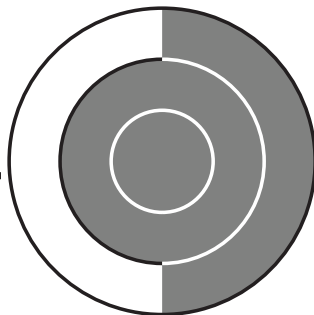
SC-UE
Strictly consistent
unit-level execution



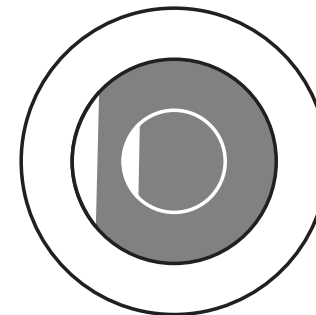
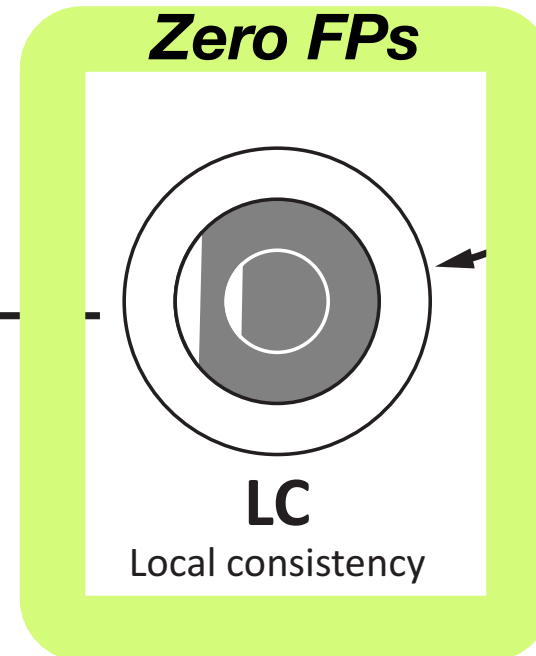
SC-SE
Strictly consistent
system-level execution



RC-CC
CFG consistency



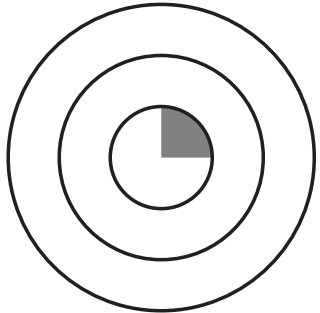
RC-OC
Overapproximate consistency



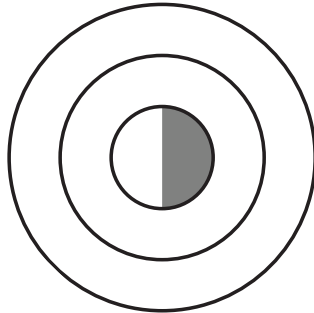
LC
Local consistency



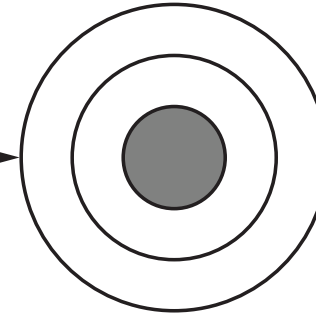
SC-CE
Strictly consistent
concrete execution



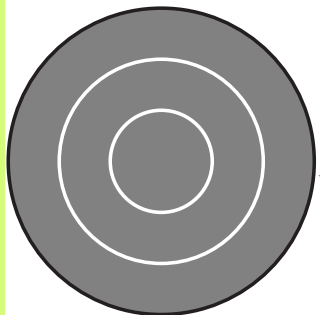
SC-UE
Strictly consistent
unit-level execution



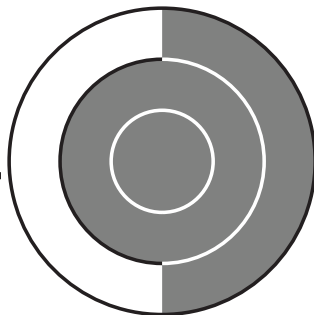
SC-SE
Strictly consistent
system-level execution



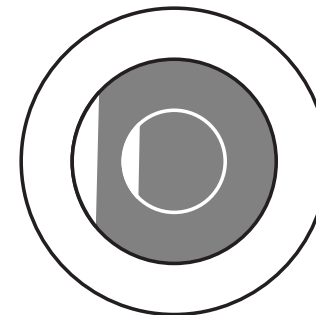
False Positives



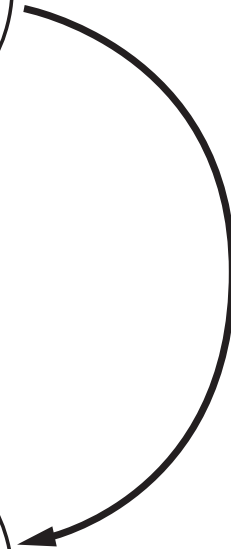
RC-CC
CFG consistency



RC-OC
Overapproximate consistency



LC
Local consistency



Mix & Match

- ECM = specification of paths to be explored
 - *S2E underneath the covers explores the requested paths*
- 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 exec. consistency model

Outline

1. The S²E Platform

- *Background*
- *S²E: The Theory*
- ➔• *S²E: The System*

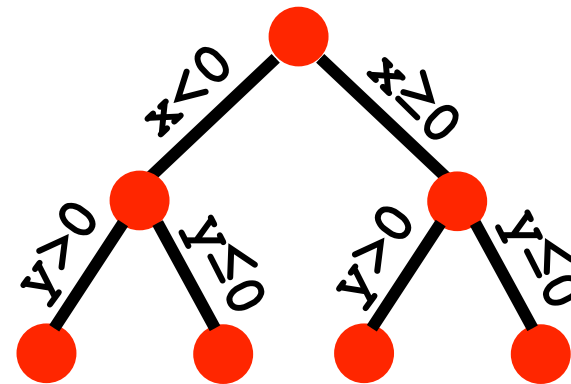
2. Three Use Cases

3. Automated Software Reliability Services

Symbolic Execution

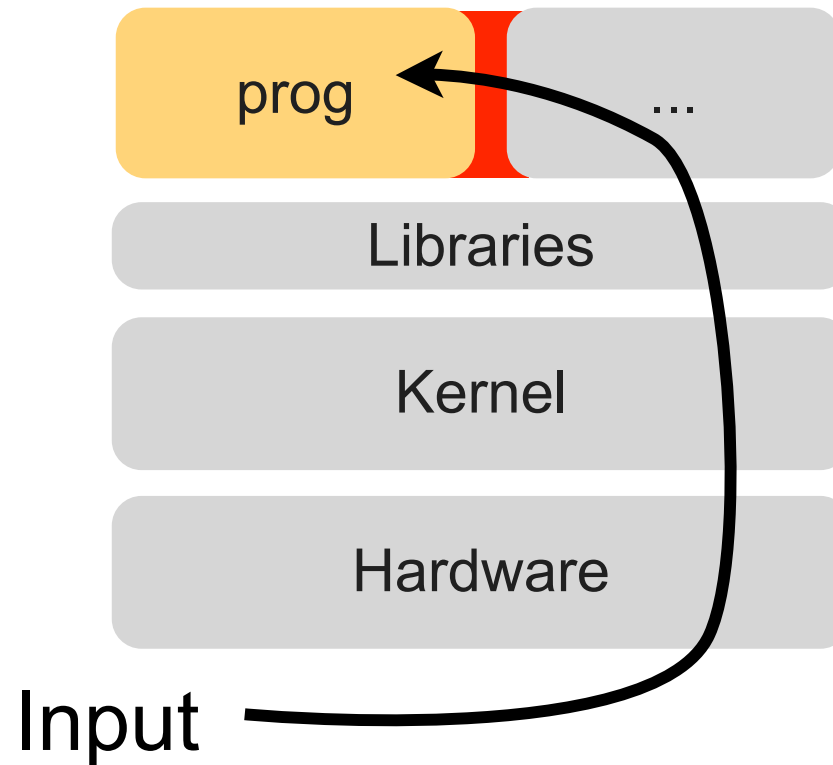
```
int func(int x, int y)
{
    if (x < 0) {
        ...
    }

    if (y > 0) {
        ...
    }
}
```



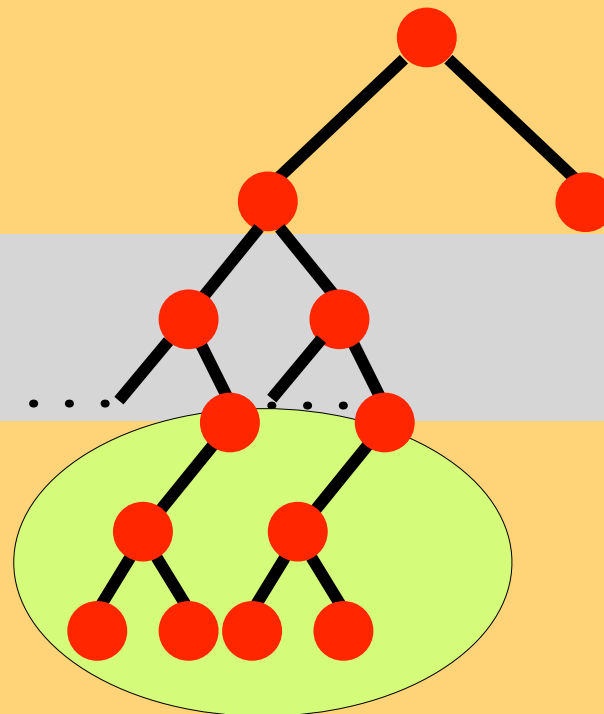
Concrete → Symbolic

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



Symbolic → Concrete

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



Unit

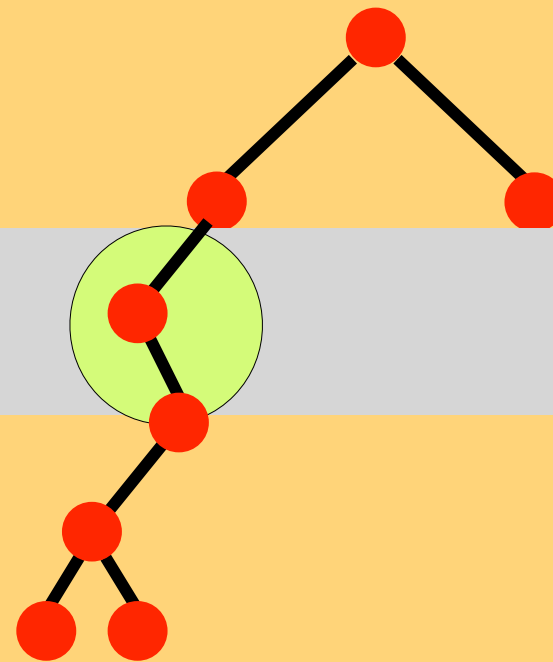
Environment

Symbolic → Concrete

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

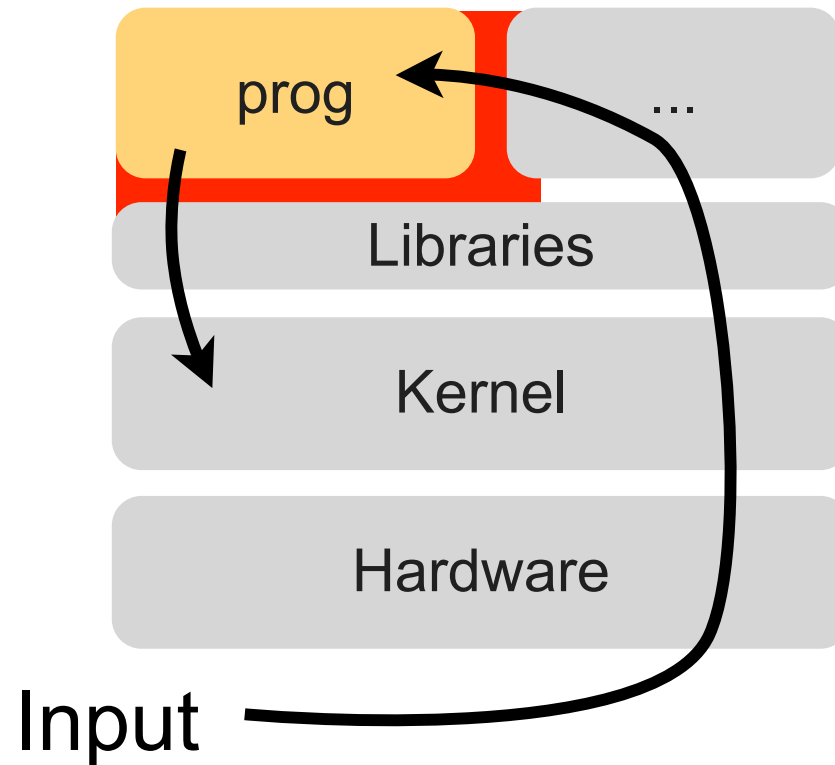
Unit

Environment



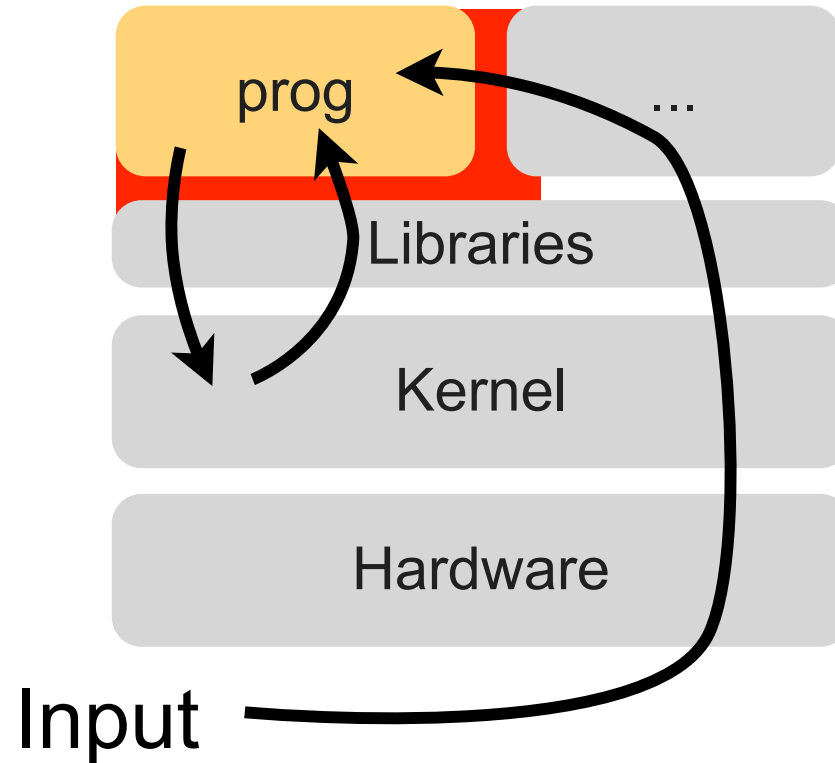
Selective Symbolic Execution

```
int main(argc $\alpha$ , argv $\beta$ ) {  
    if (argc == 0) {  
        ...  
    }  
    p = malloc(5128);  
    if (p == NULL) {  
        ...  
    }  
    ...  
}
```

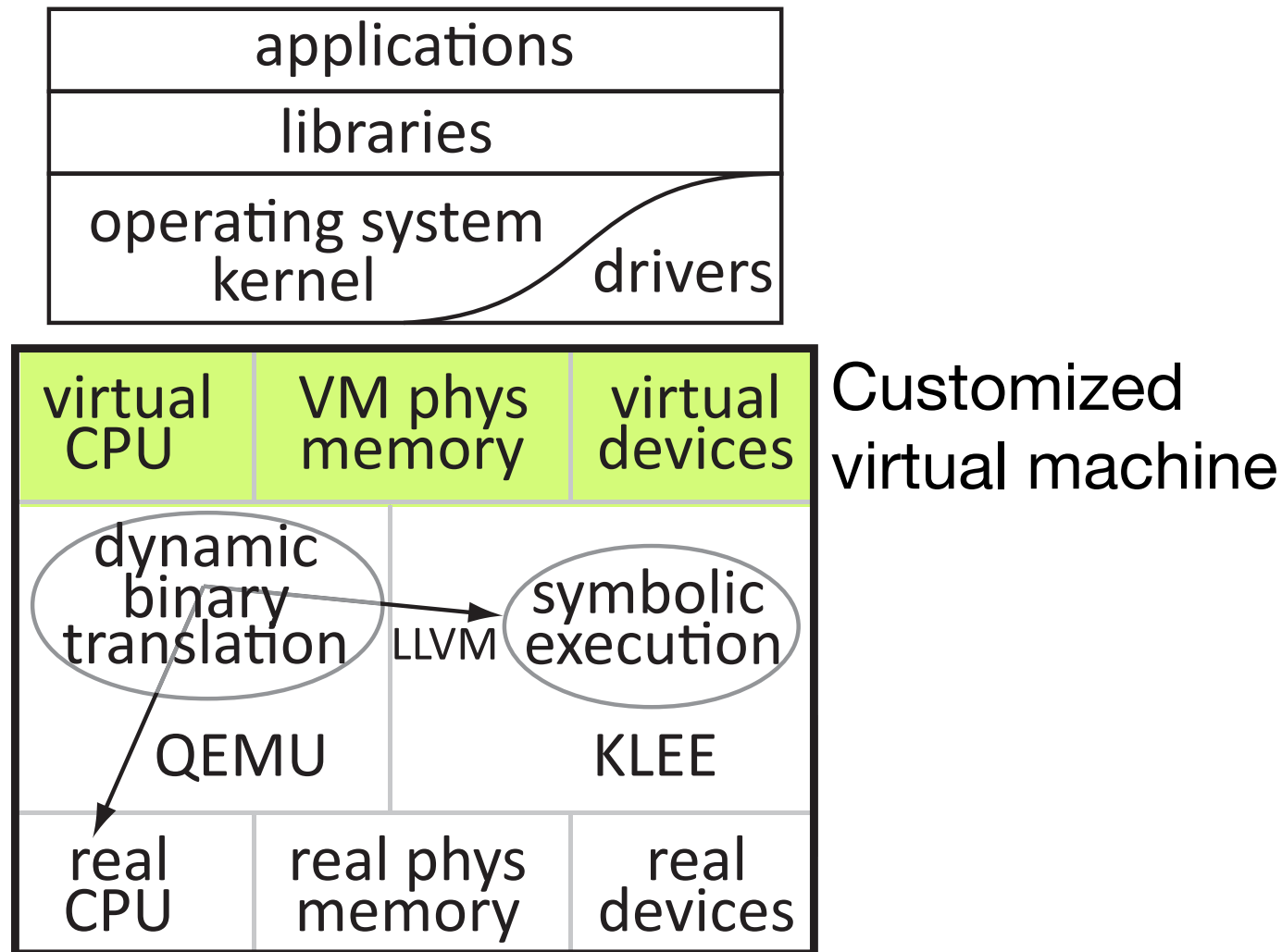


Selective Symbolic Execution

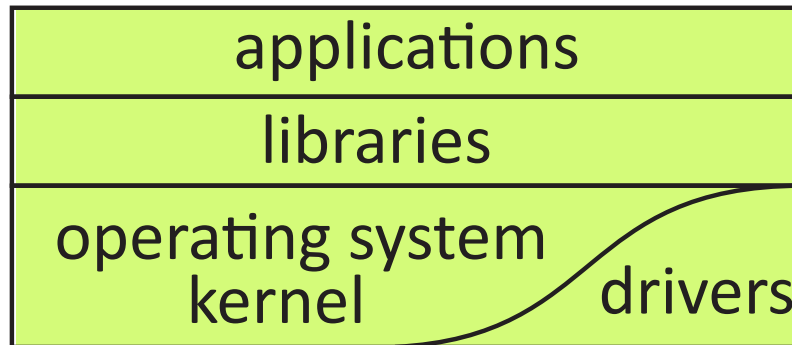
```
int main( $\alpha$ argc,  $\beta$ argv) {  
    if (argc == 0) {  
        ...  
    }  
    p = malloc(1285);  
    if (p == NULL) {  
        ...  
    }  
    ...  
}
```



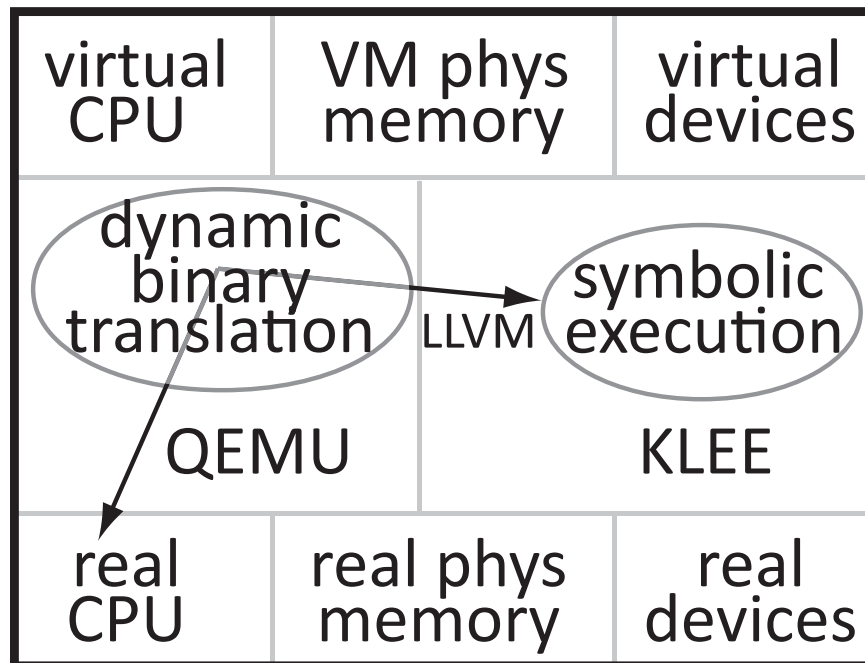
The S²E System



The S²E System

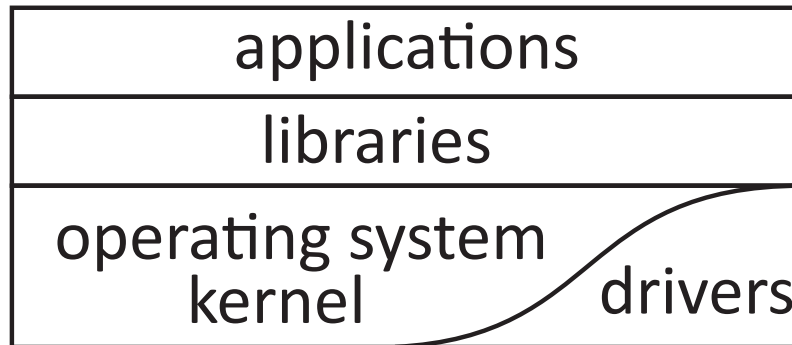


Runs unmodified x86 binaries (incl. proprietary/obfuscated/encrypted binaries)

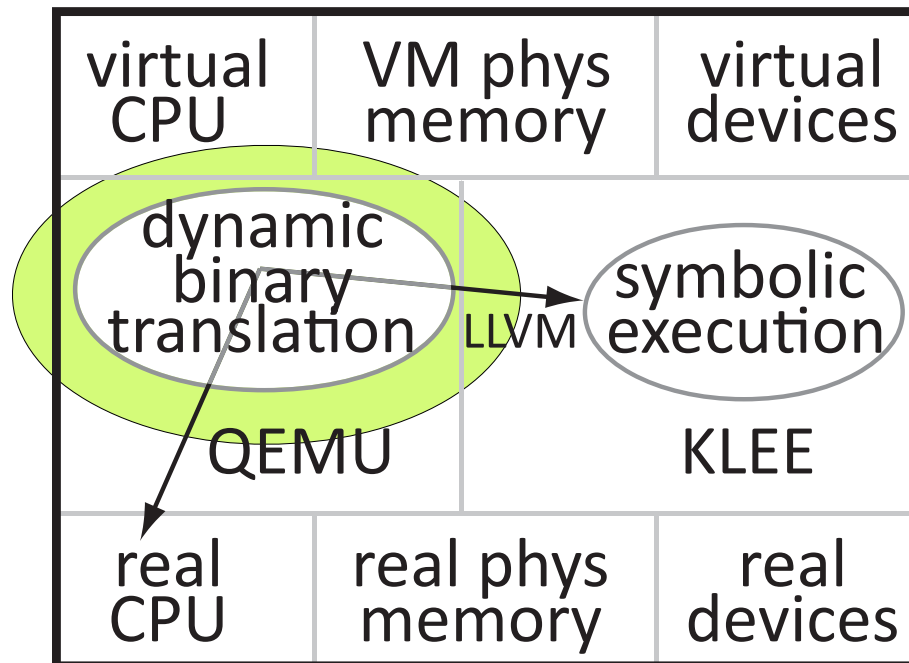


Customized virtual machine

The S²E System



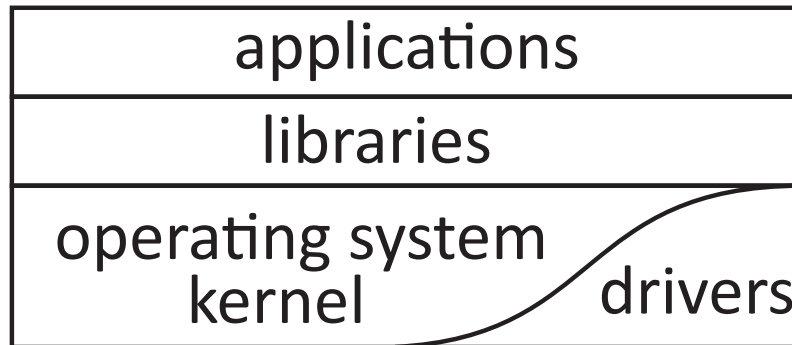
Runs unmodified x86 binaries (incl. proprietary/obfuscated/encrypted binaries)



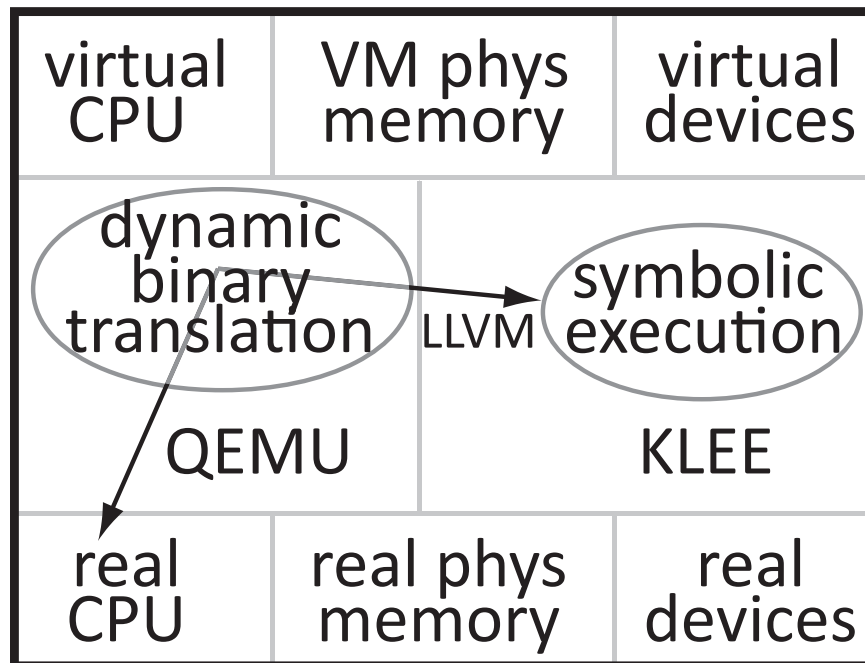
Customized virtual machine

Selection done at runtime
Most code runs “natively”

The S²E System



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

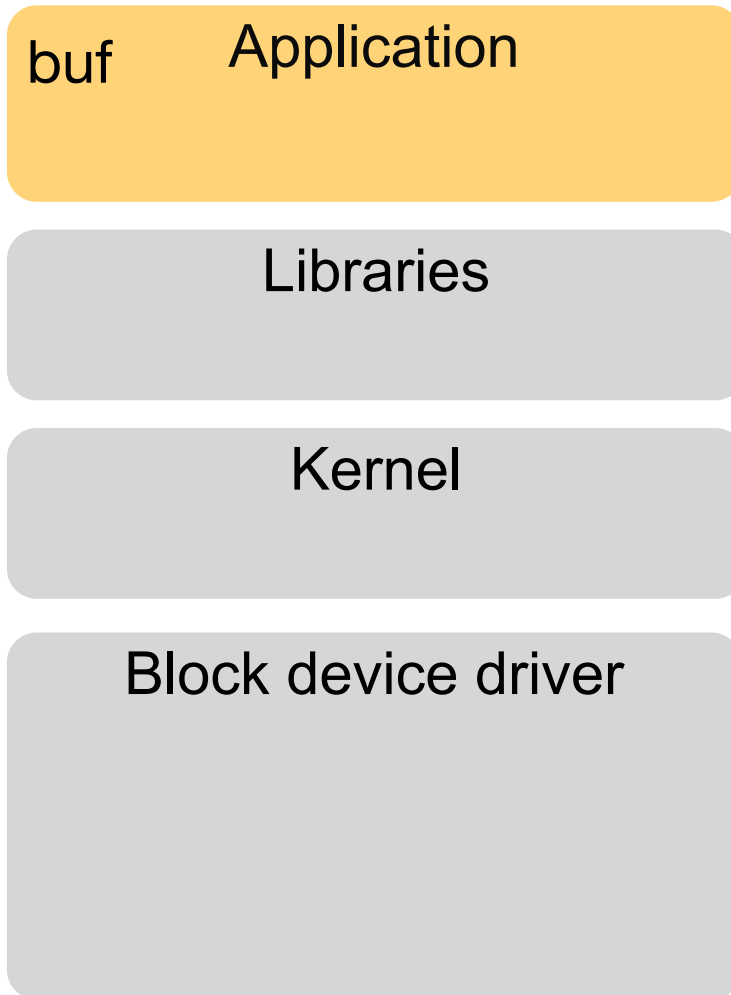


Customized virtual machine

Selection done at runtime
Most code runs “natively”

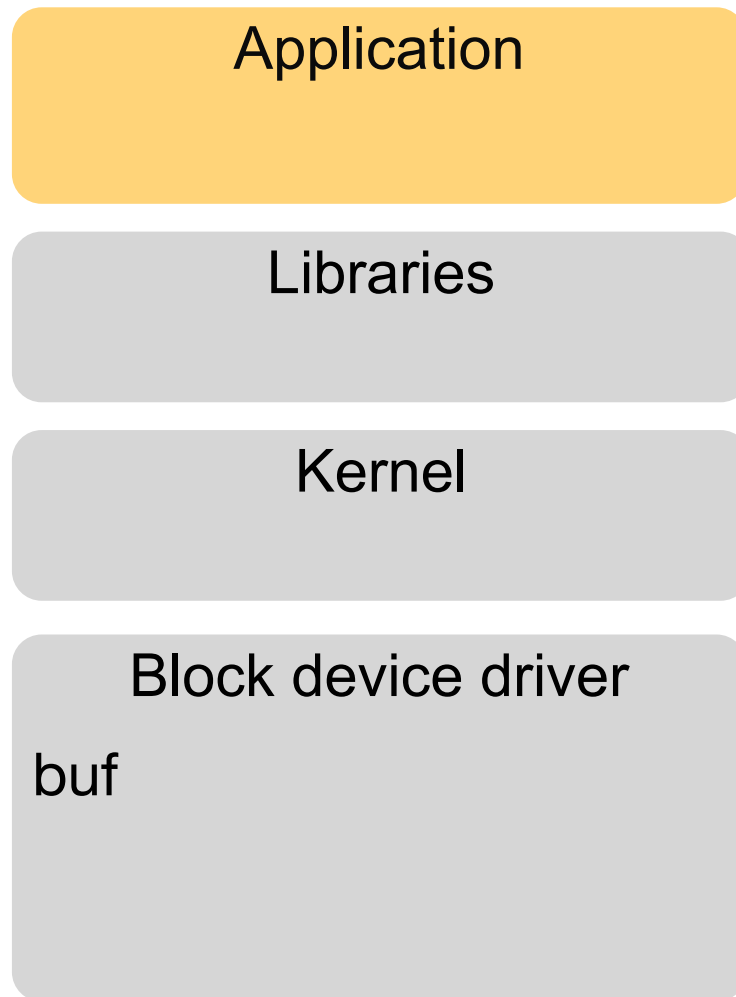
Shared concrete/symbolic state representation

Lazy+Selective Conversion



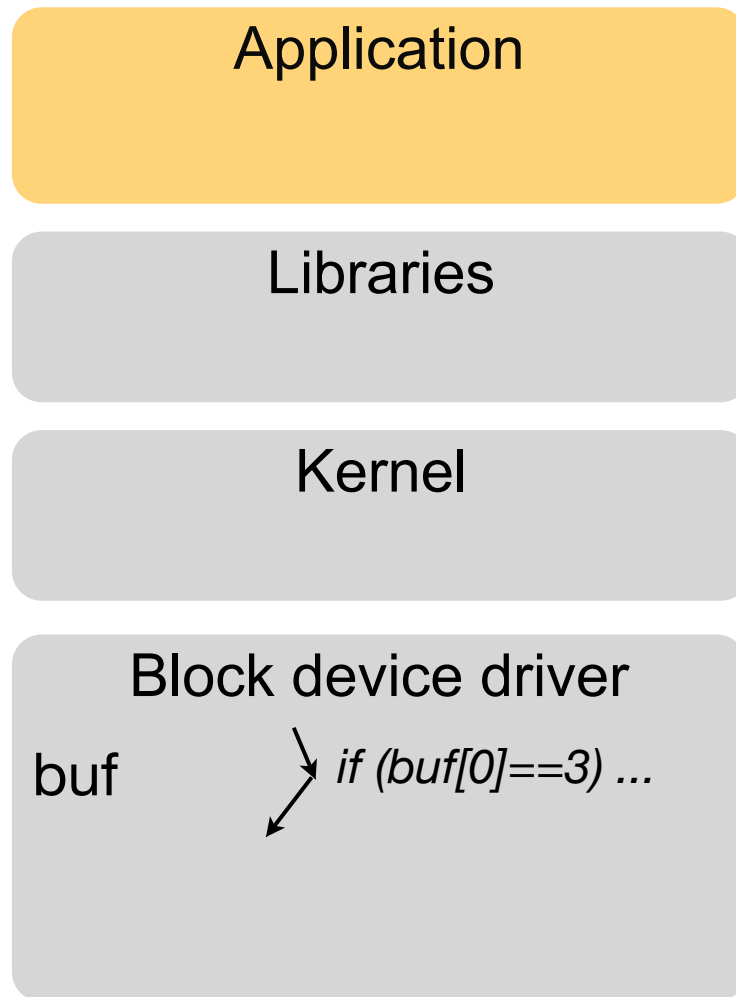
Avoids *unnecessary*
concretization

Lazy+Selective Conversion



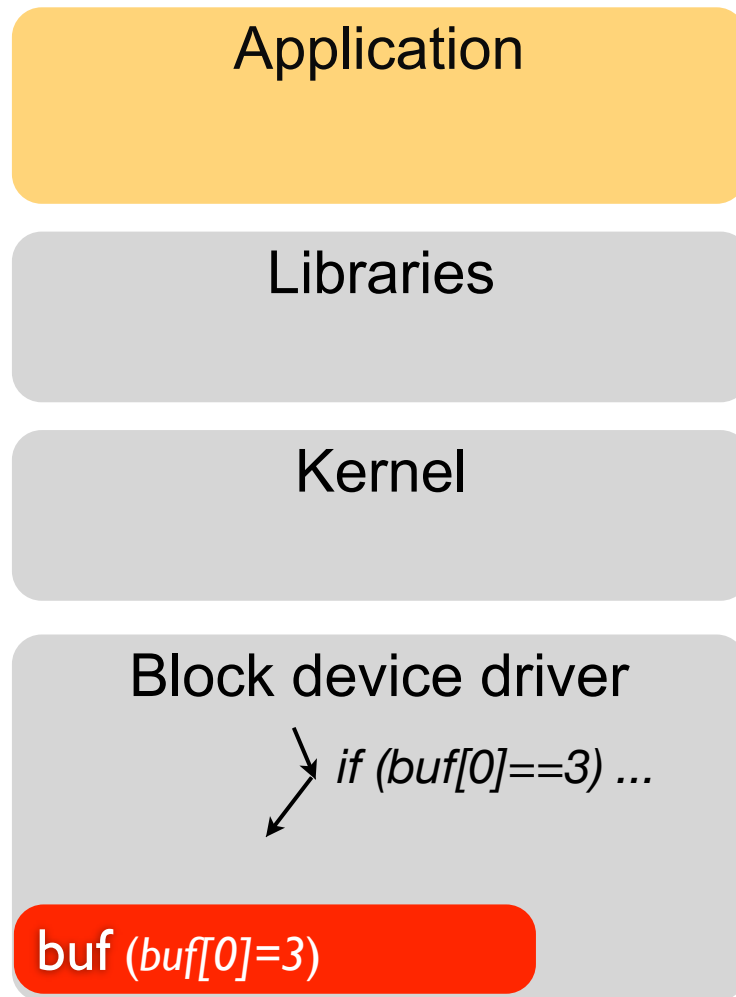
Avoids *unnecessary*
concretization

Lazy+Selective Conversion



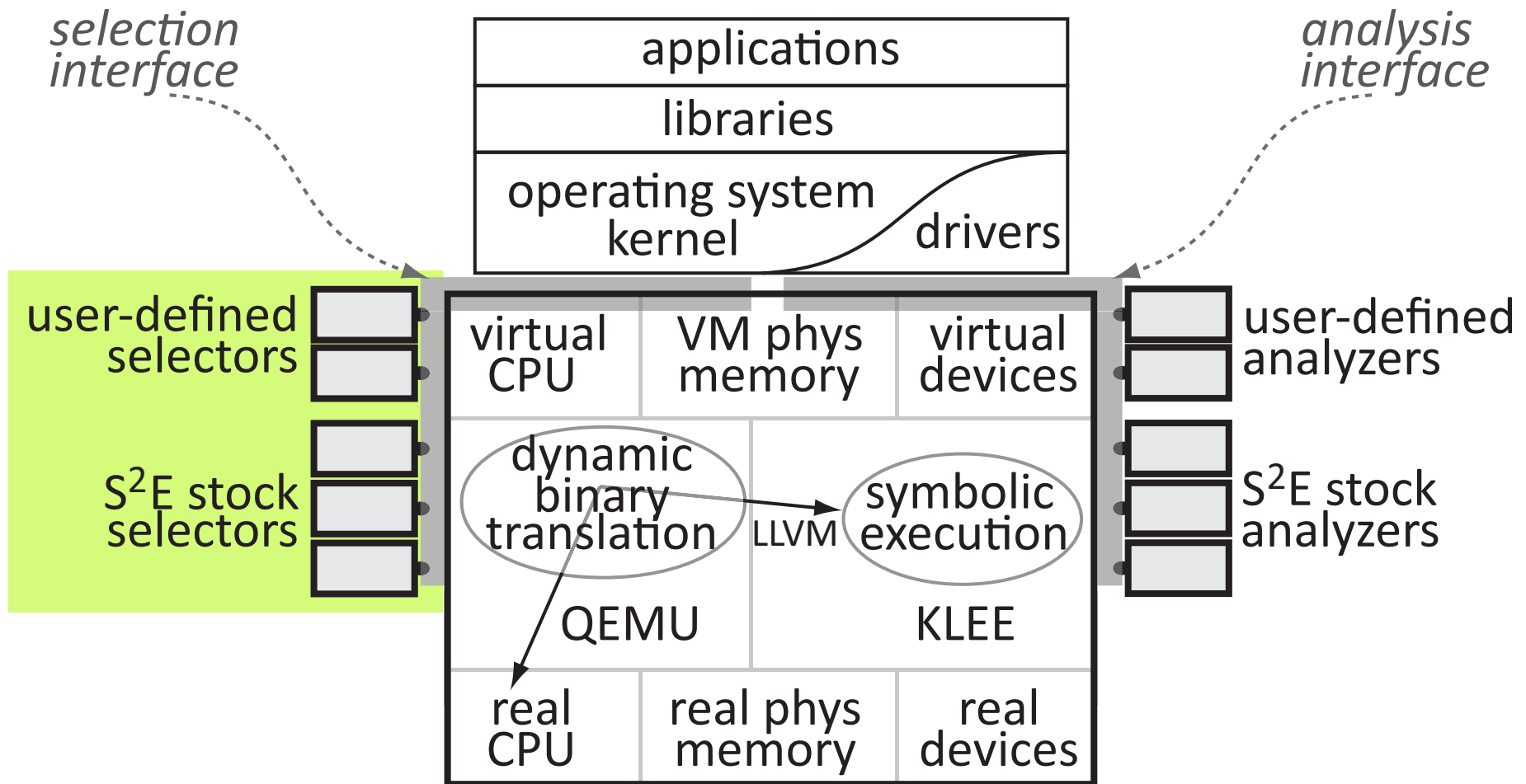
Avoids *unnecessary* concretization

Lazy+Selective Conversion

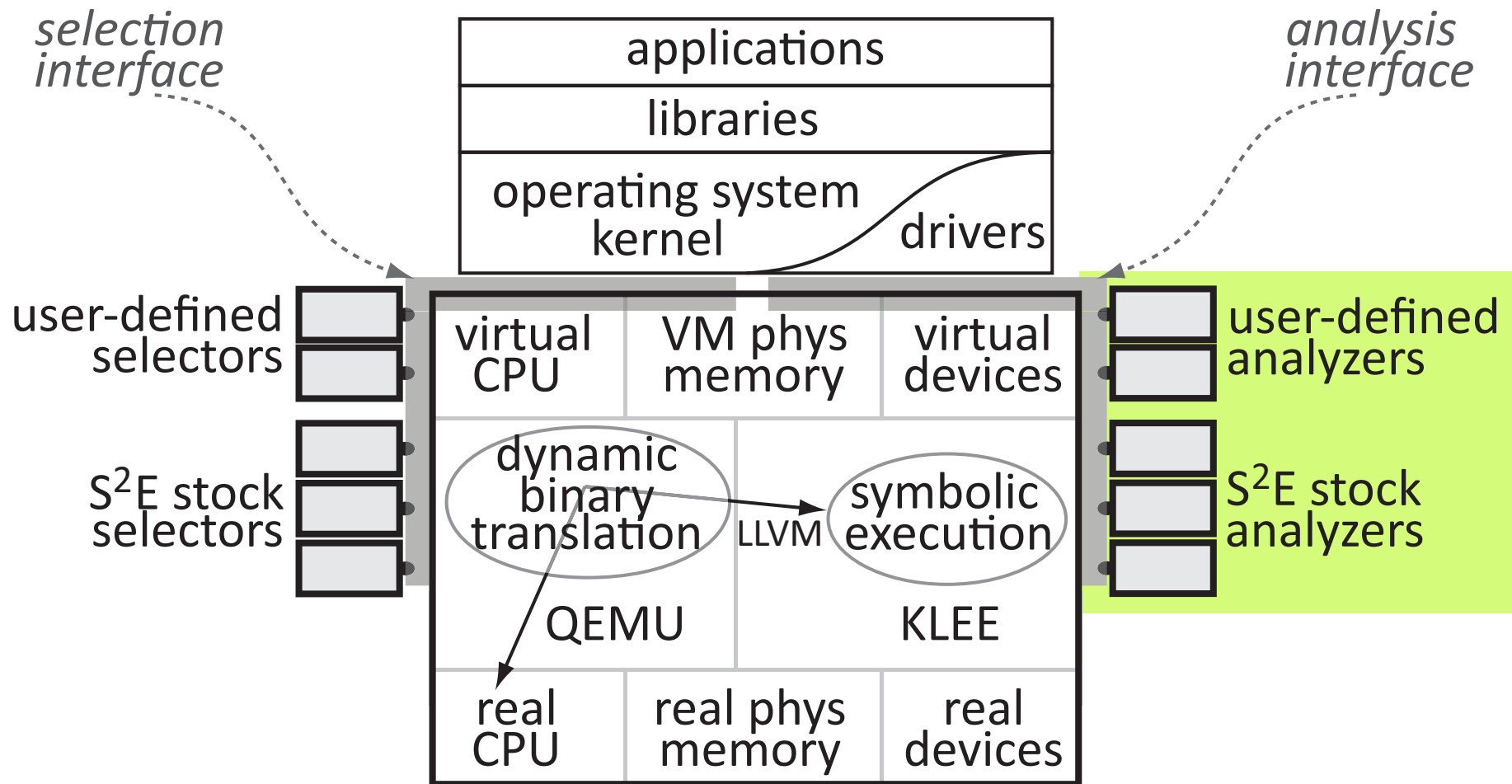


Avoids *unnecessary* concretization

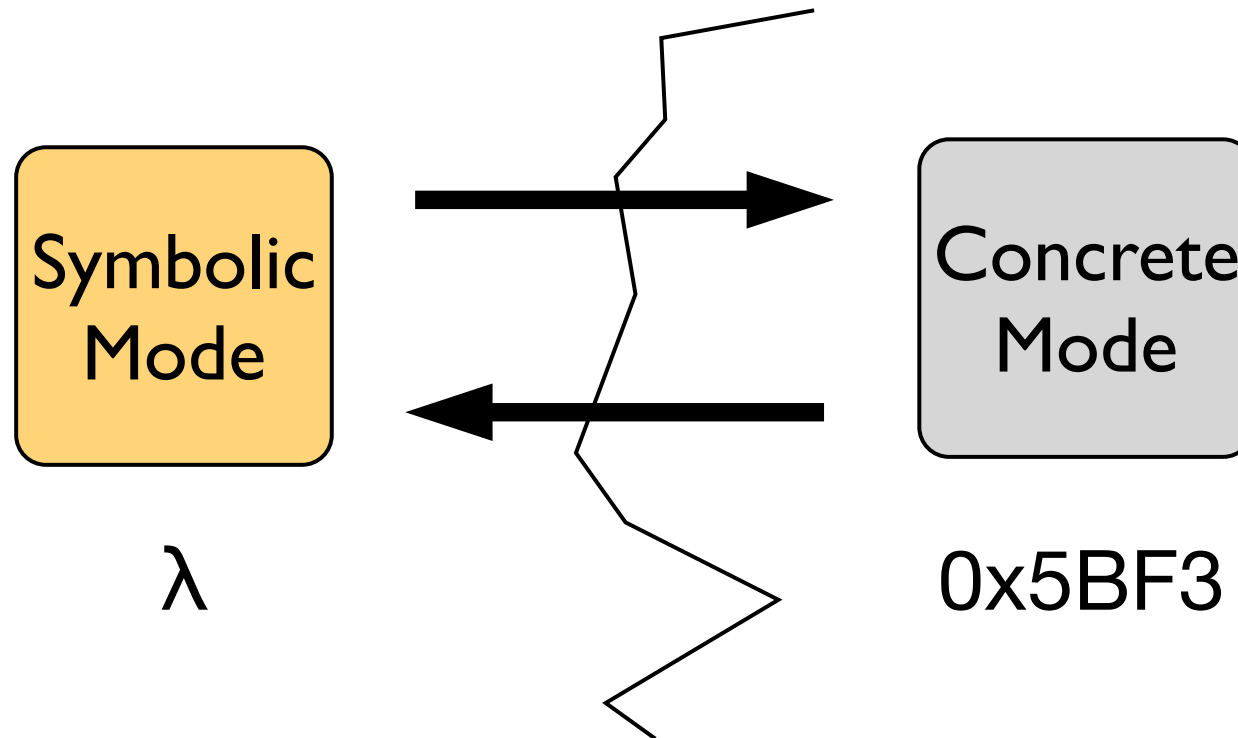
S2E User's View



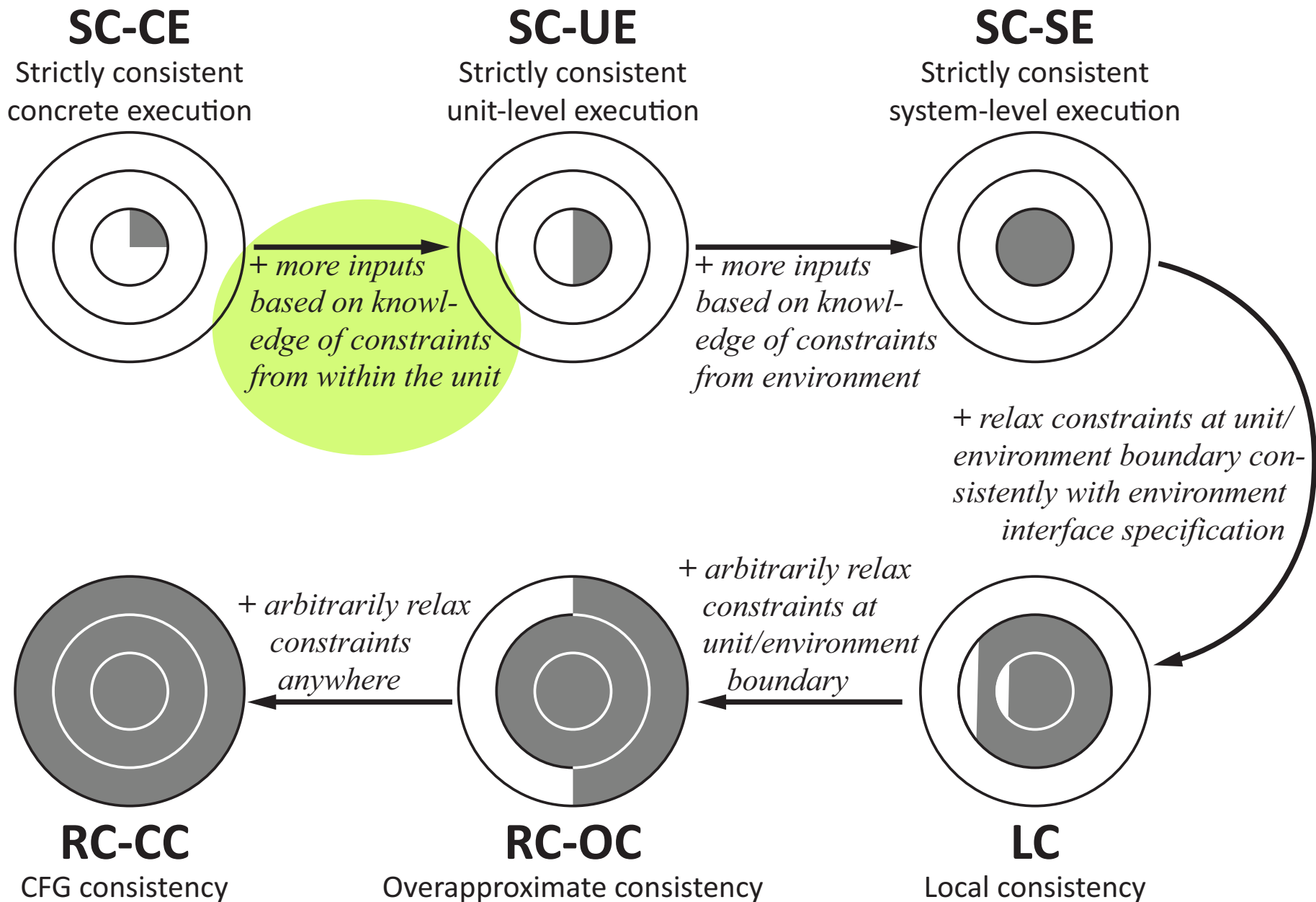
S2E User's View

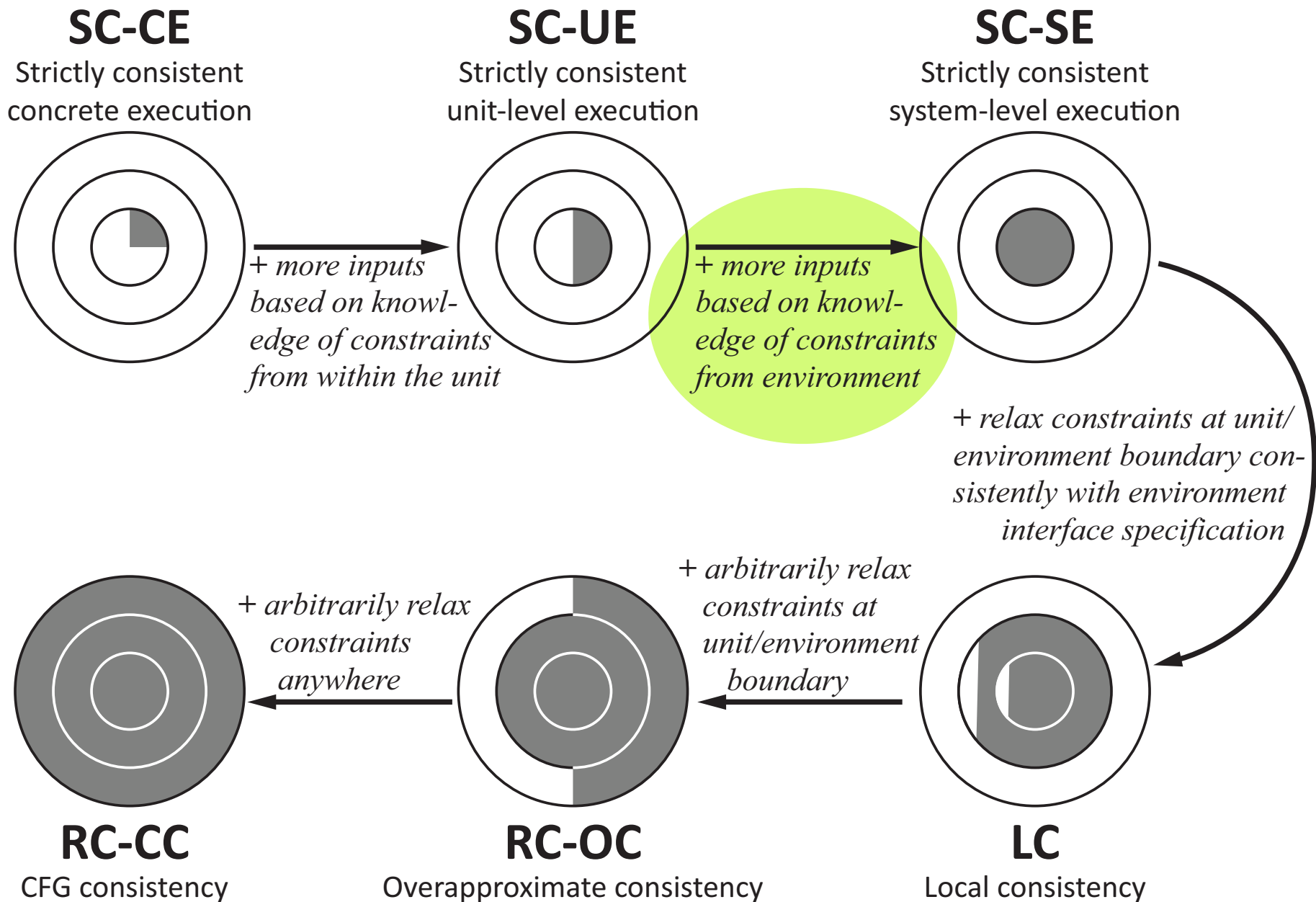


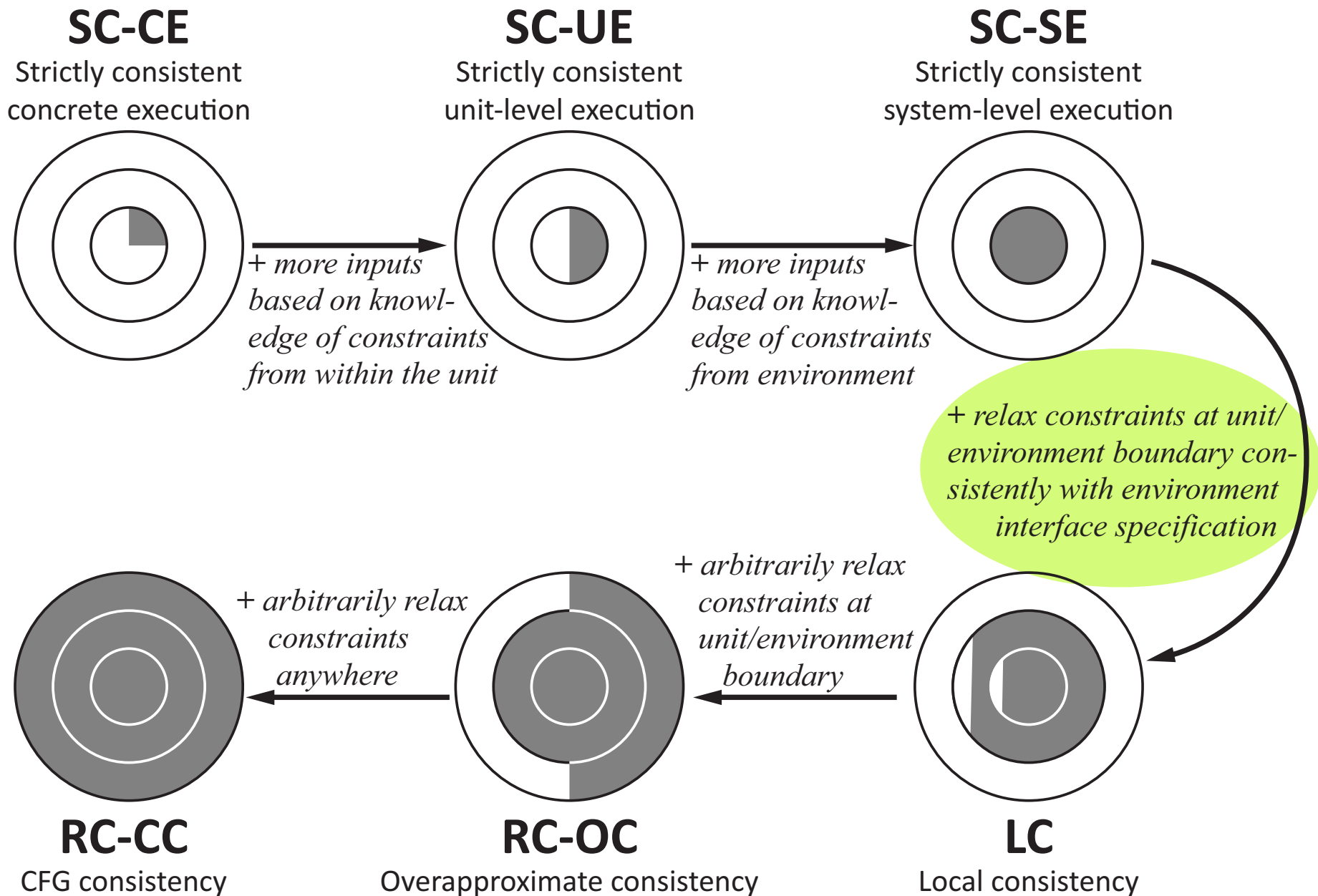
Key S²E Feature

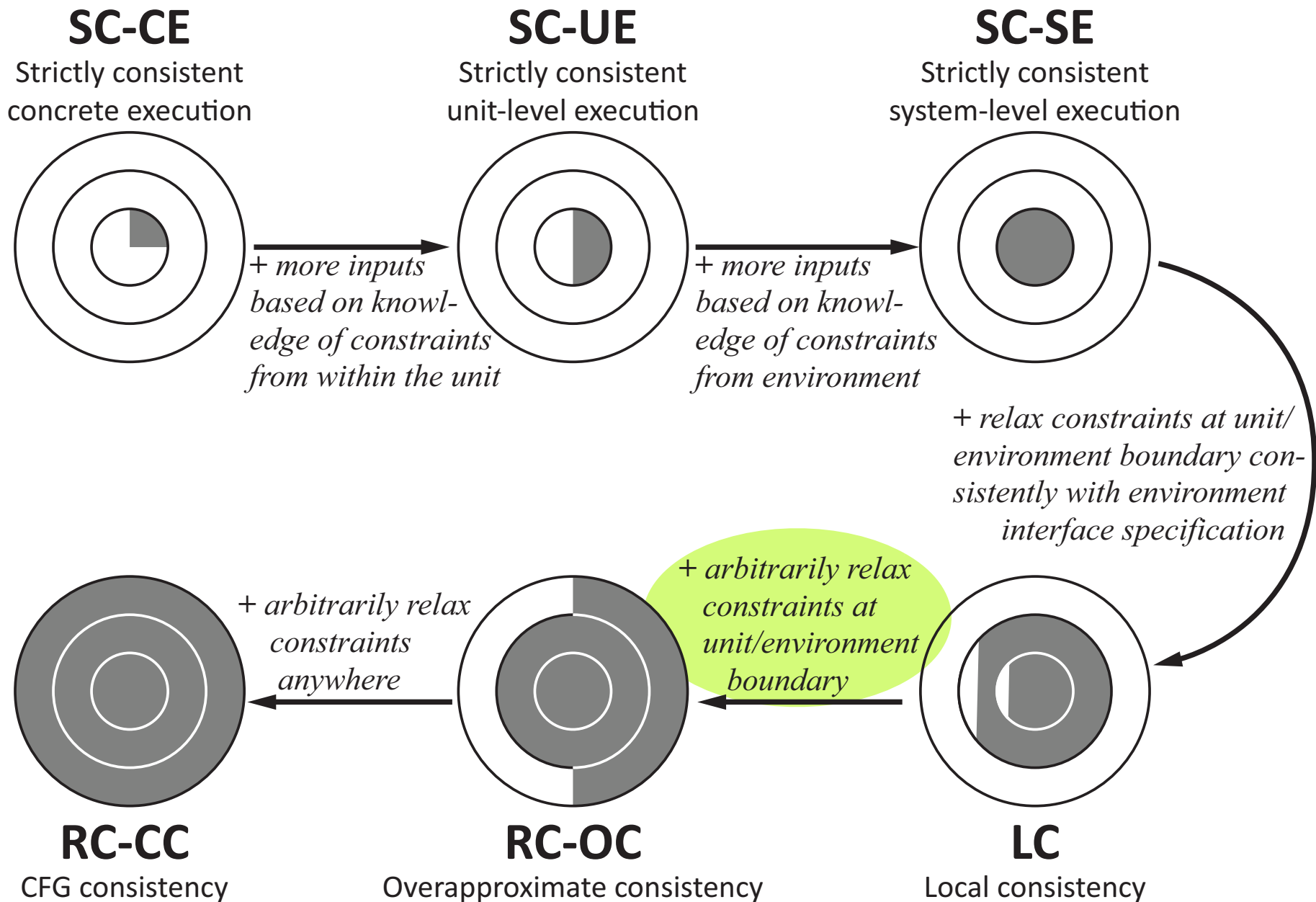


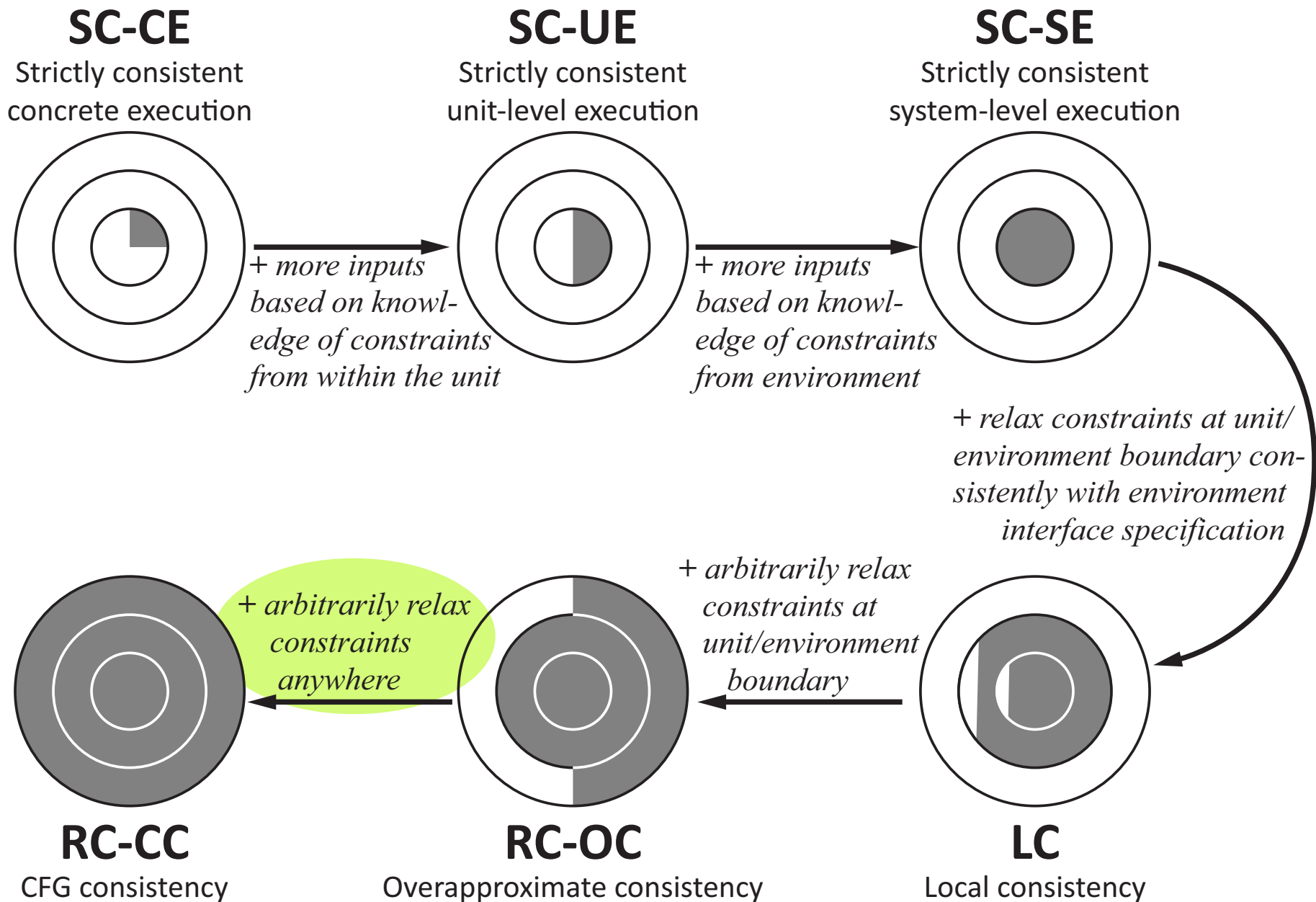
Execution weaves between symbolic/concrete transparently, efficiently, and consistently











SC-CE

Strictly consistent
concrete execution

Classic fuzzing
(most real-world
testing)

SC-UE

Strictly consistent
unit-level execution

Dynamic symbolic &
concolic execution
engines
(DART, EXE)

SC-SE

Strictly consistent
system-level execution

Symbolic execution
engines with
environment models
(KLEE)

Most disassemblers

Automated reverse
engineering (RevNIC)

Some automated
testing tools (DDT)

RC-CC

CFG consistency

RC-OC

Overapproximate consistency

LC

Local consistency

Outline

1. The S²E Platform

2. Three Use Cases

- ➔ ● *Finding bugs in proprietary software*
 - *Reverse engineering*
 - *Performance profiling*

3. Automated Software Reliability Services

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.

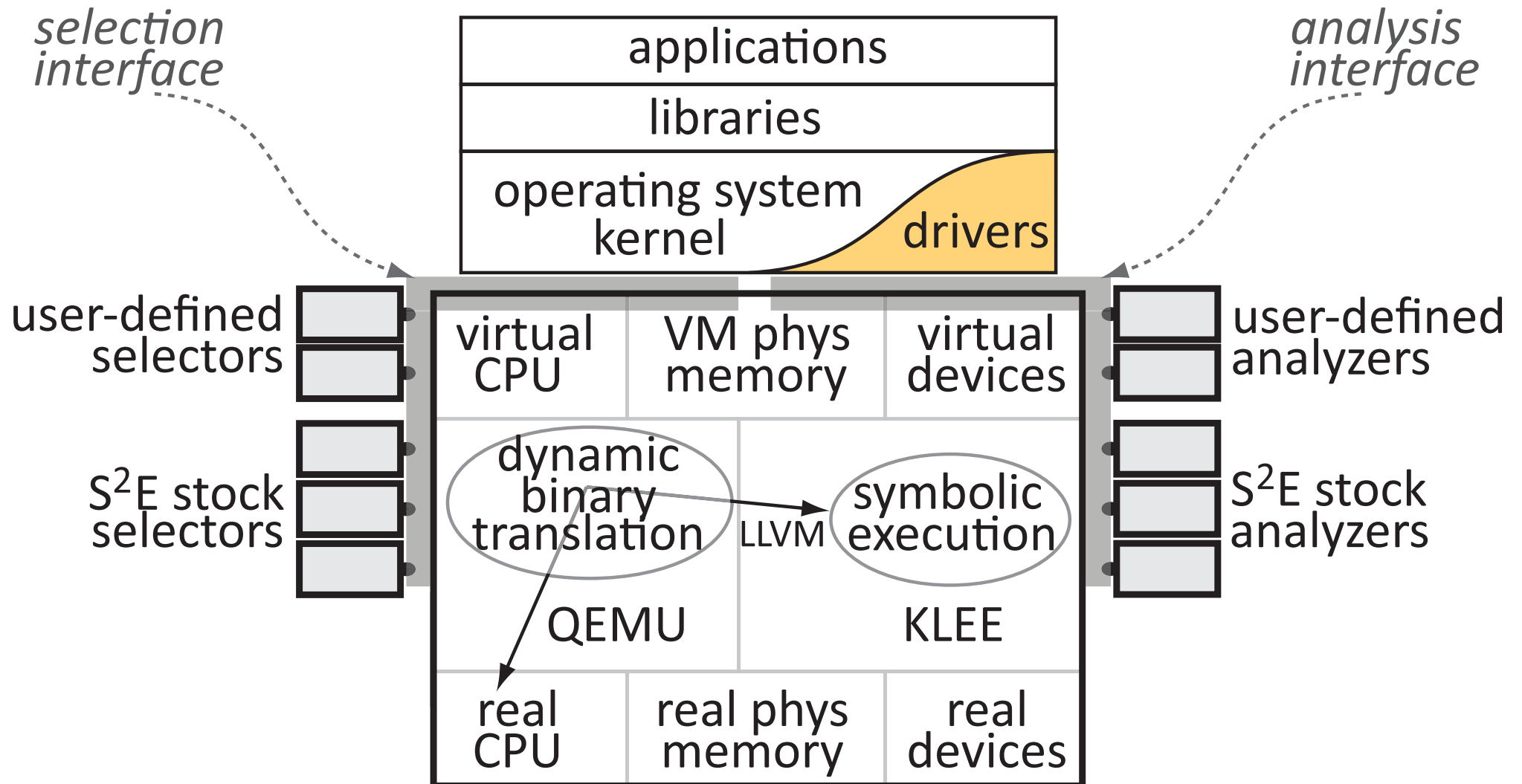
If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

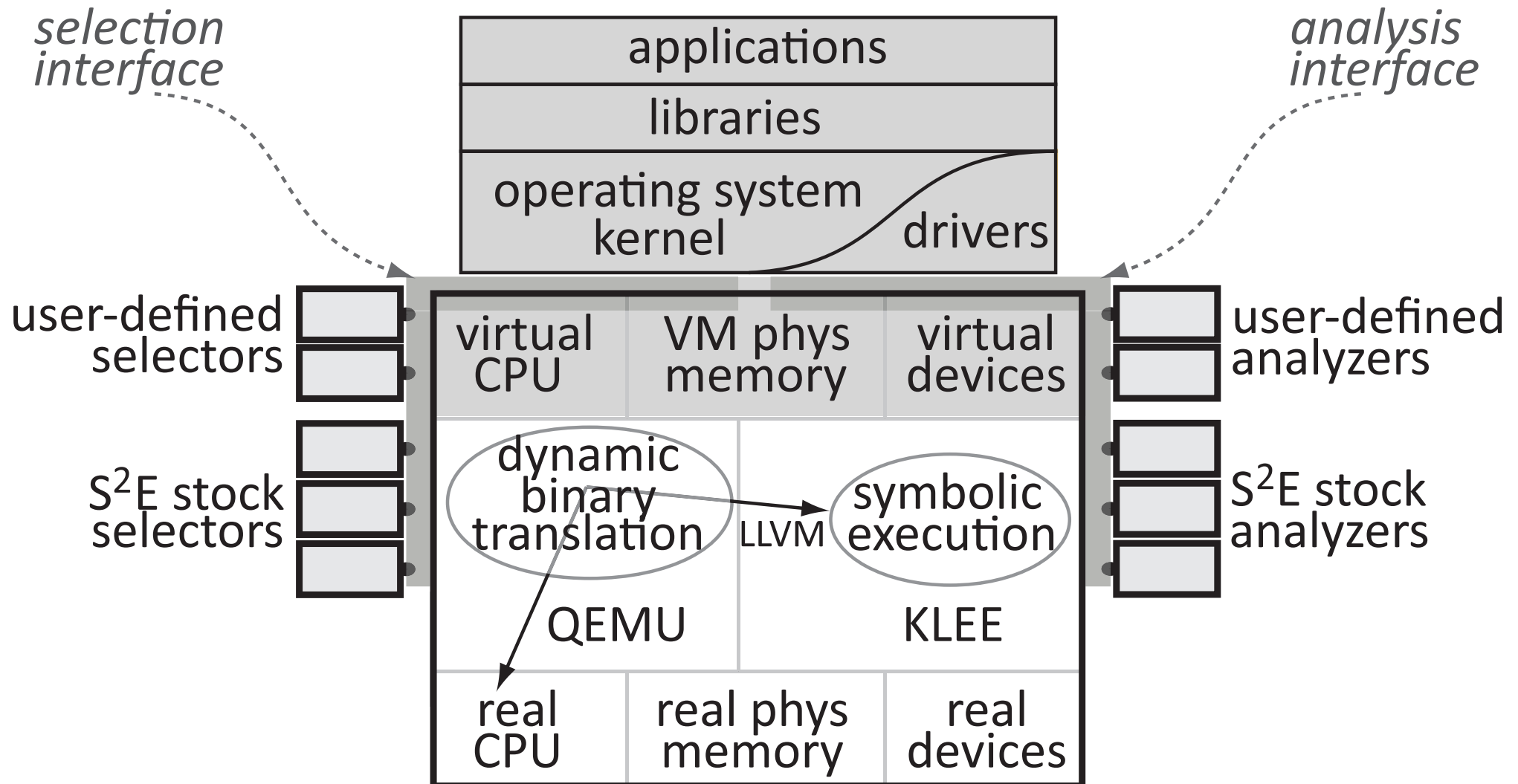
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

DDT+ Testing Tool



DDT+ Testing Tool



DDT+

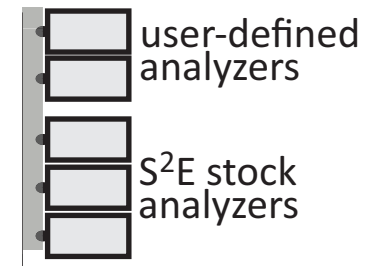
- Path exploration

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



- Path Analysis

- Off-the-shelf single-path checkers
- Our own VM-level analyzers (incl. coverage counter)

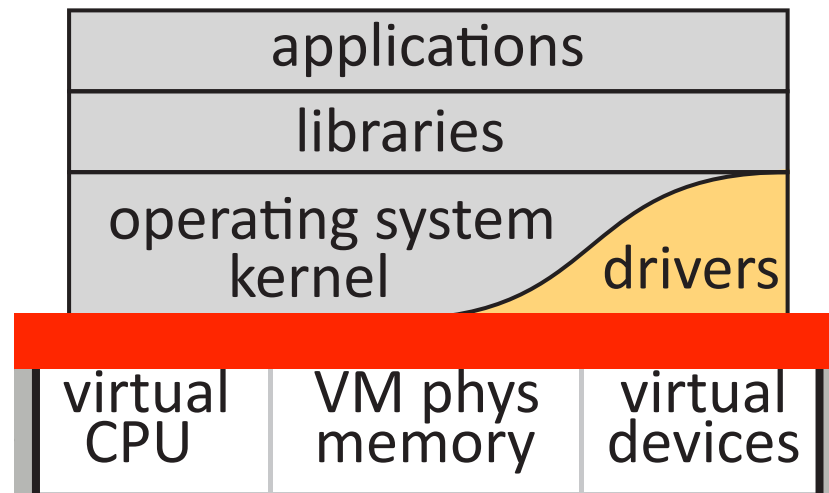


- Results of analysis

- find bugs \Rightarrow executable traces (inputs, instructions, ...)
- traces prove presence of the bugs + help fix the bugs

Symbolic Hardware

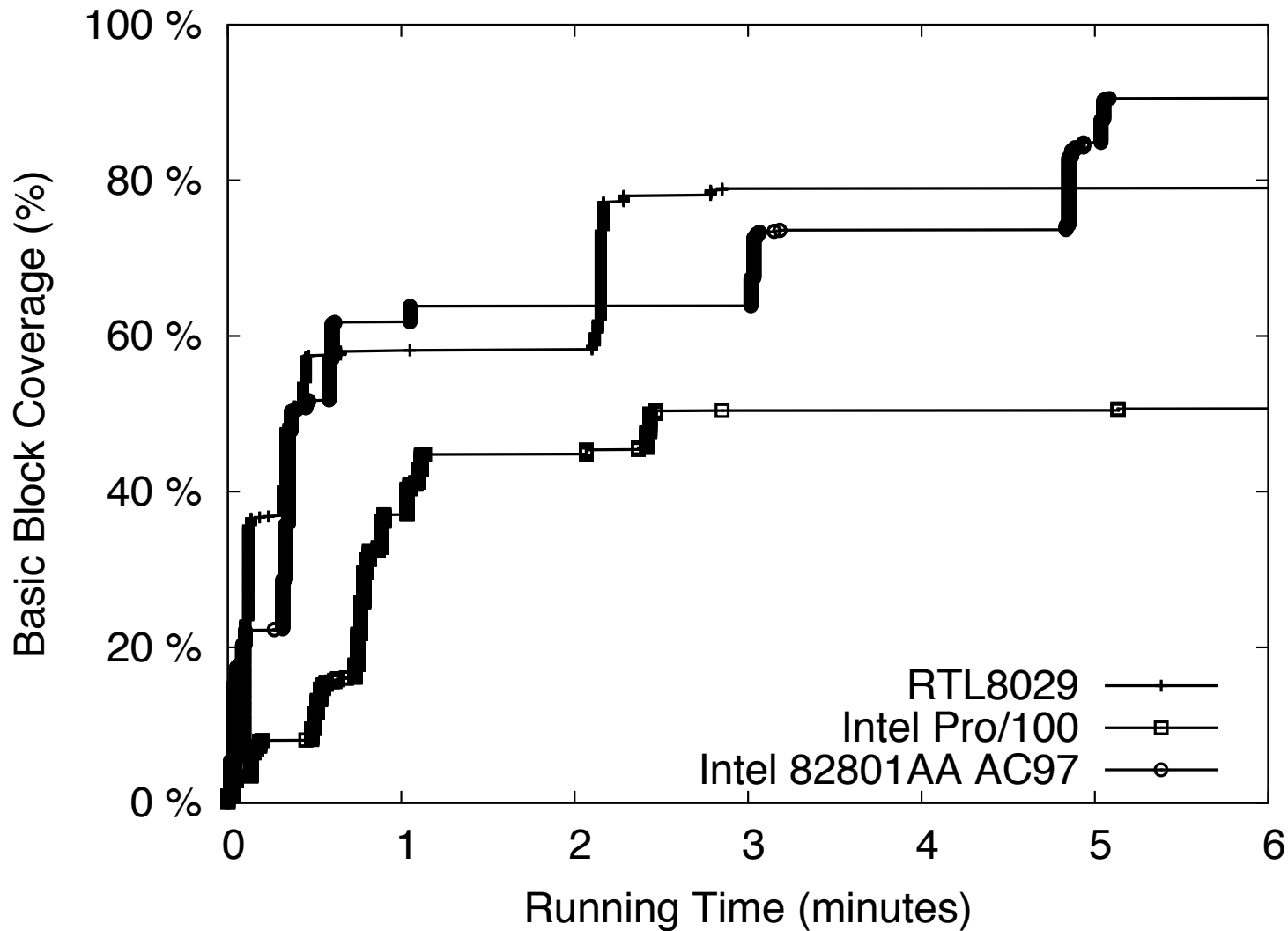
- Symbolic HW inputs, symbolic interrupts, etc.
- Test without having the real hardware
- Test for bad hardware behaviors



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



Outline

1. The S²E Platform

2. Three Use Cases

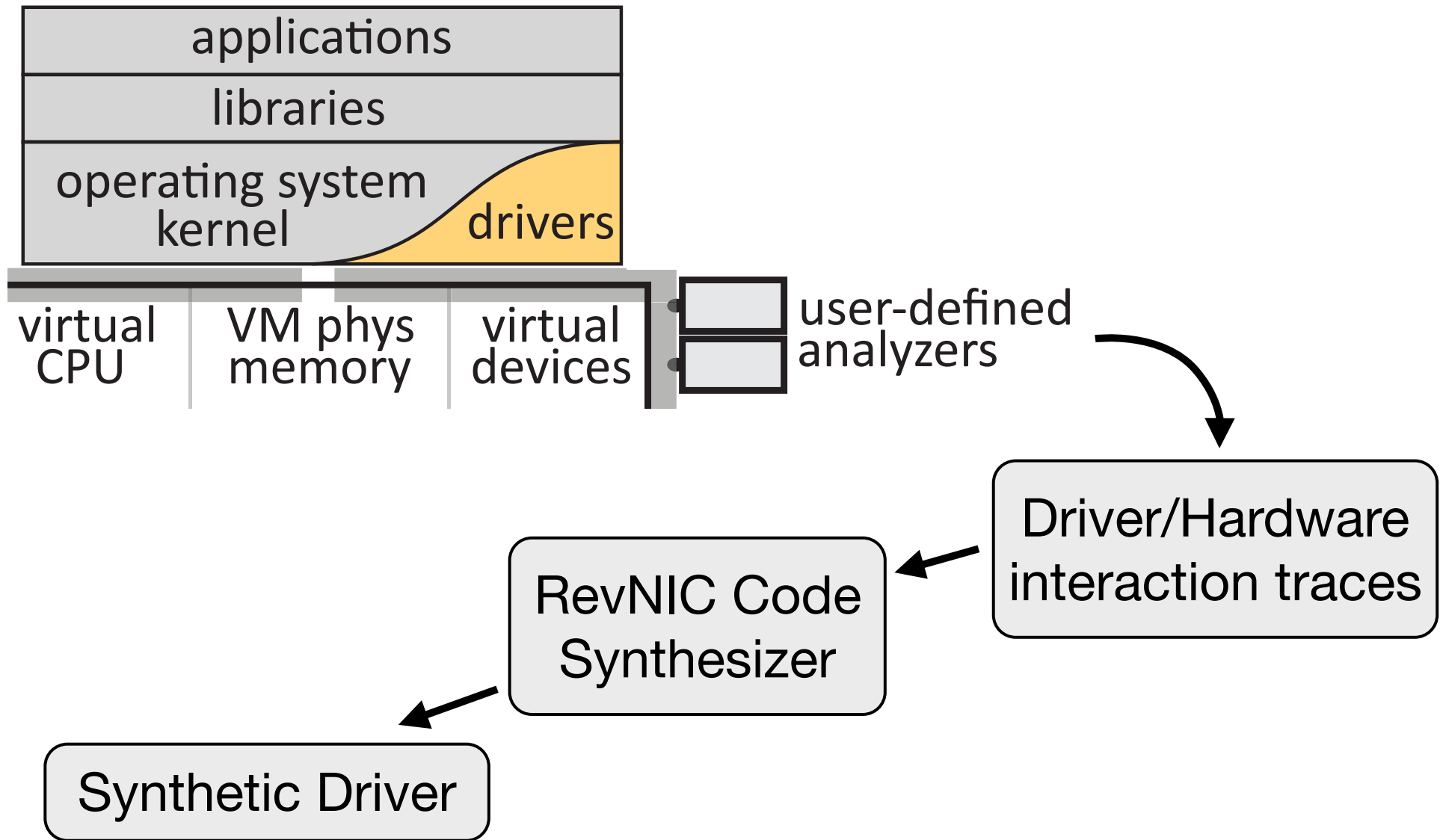
- *Finding bugs in proprietary software*

→• *Reverse engineering*

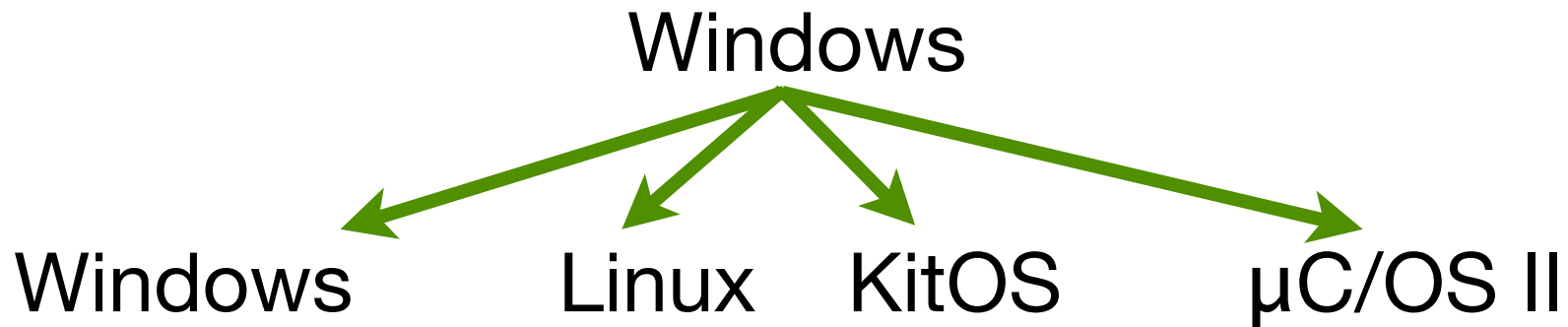
- *Performance profiling*

3. Automated Software Reliability Services

RevNIC+ Reverse Engineering



Automated Porting



x86 PC

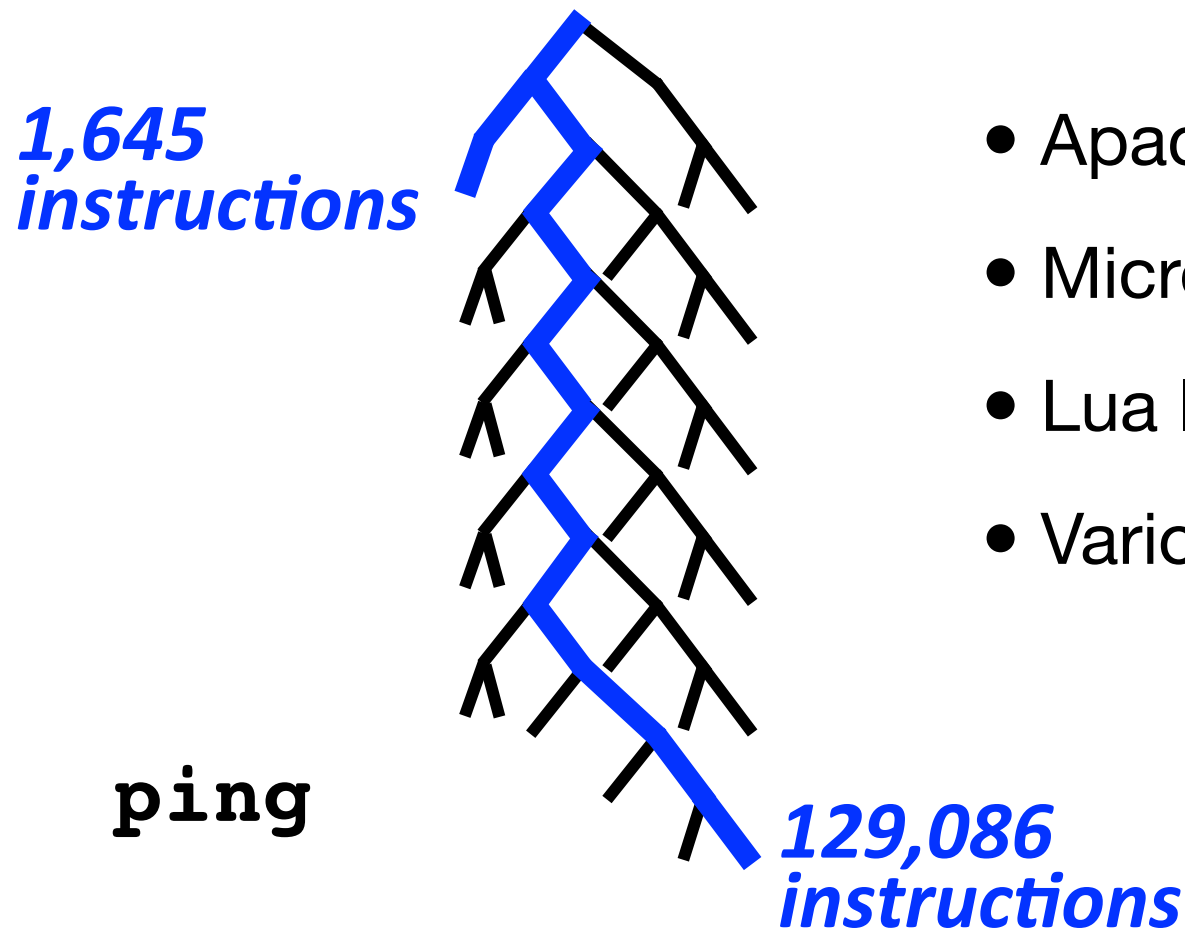


**VMware
QEMU**



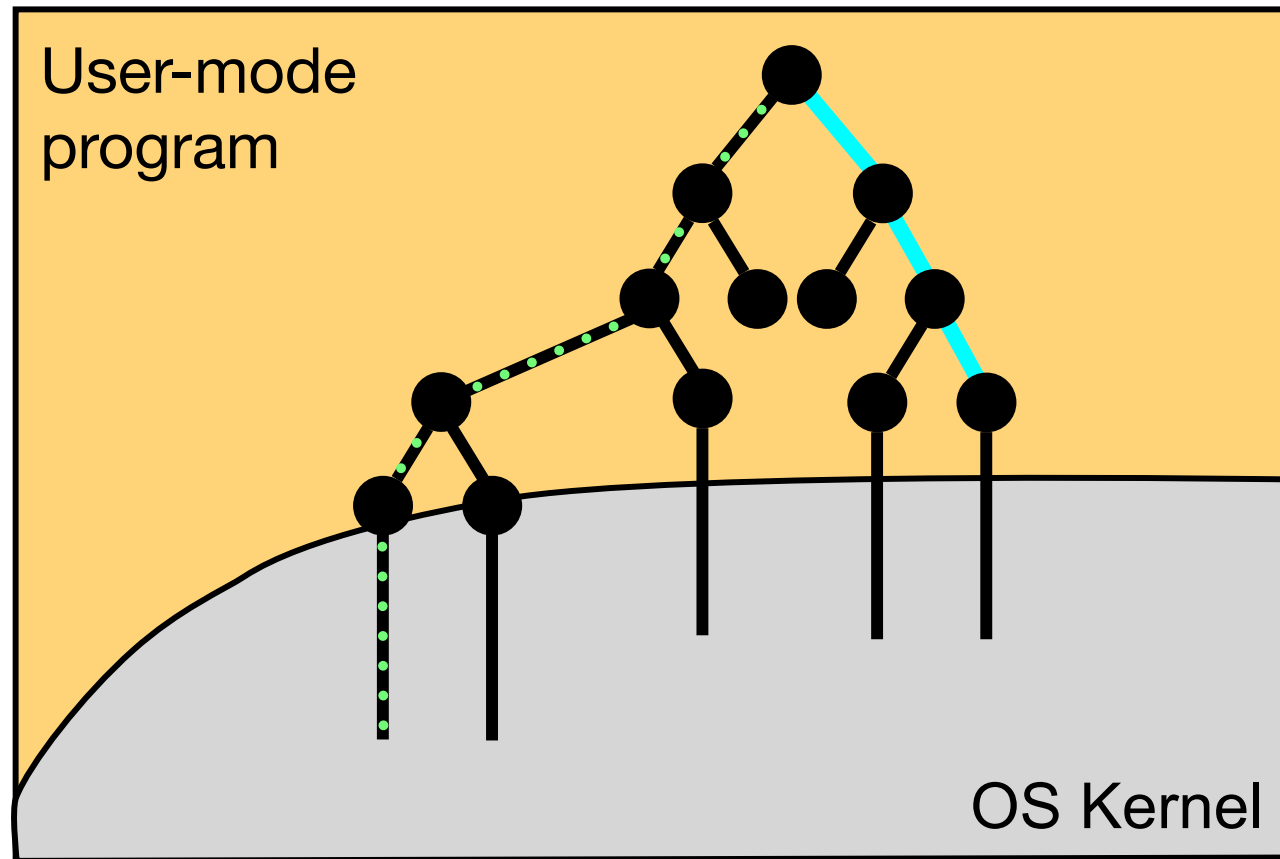
FPGA4U

PROFs: Performance Profiling



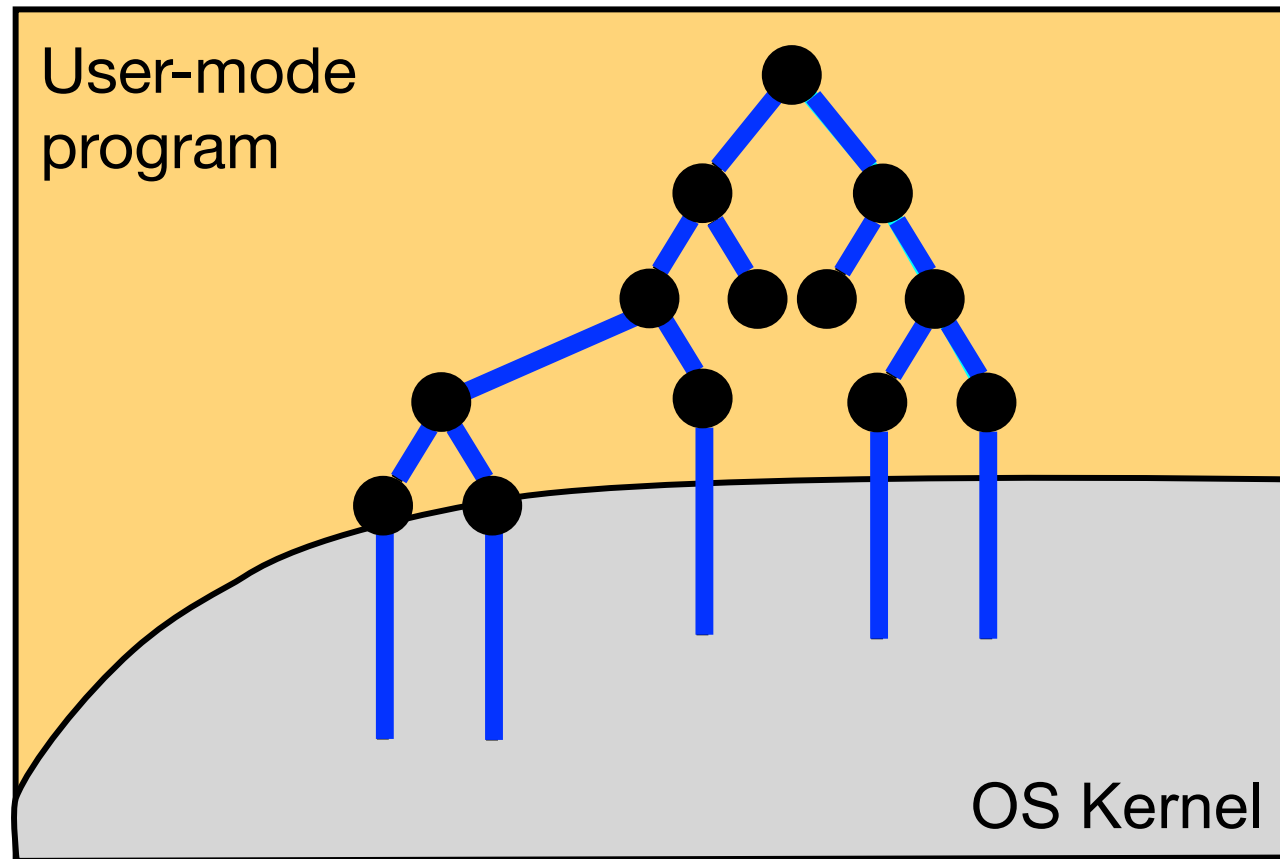
- Apache URL parser
- Microsoft IIS SSL module
- Lua language interpreter
- Various utilities

Multi-Path In-Vivo Profiling



— Valgrind
- - - Oprofile

Multi-Path In-Vivo Profiling



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 in-vivo multi-path analysis tools
- Selective symbolic execution of x86 binaries
- Execution consistency models



<http://s2e.epfl.ch>

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

Outline

1. The S²E Platform

2. Three Use Cases

- *Testing of proprietary software*
- *Reverse engineering*
- *Performance profiling*

3. Automated Software Reliability Services

July 4, 1996

The screenshot shows the Hotmail website interface. On the left, the Hotmail logo is displayed above the text "The World's FREE Web-Based Email". The main content area is a blue box with a dark blue header that reads "REGISTERED USERS". Below this header, there are two input fields for "Login Name" and "Password", followed by an "Enter" button. Below the input fields are three radio buttons: "Frames", "No Frames", and "My Default", with "My Default" selected. Below the radio buttons are two buttons: "Who Should Sign Up?" and "Sign Up Here!". To the right of these buttons are three links: "About Hotmail", "Email Safety", and "Privacy Statement". Below the links is a dark blue header that reads "AWARDS". Underneath the "AWARDS" header are two award banners. The left banner is for "The John C. Dvorak Telecommunications Excellence Award" and the right banner is for "PC Computing THE A LIST Top Pick for Internet Email".

January 12, 2011

- **107 trillion** – The number of emails sent on the Internet in 2010.
- **294 billion** – Average number of email messages per day.
- **1.88 billion** – The number of email users worldwide.
- **480 million** – New email users since the year before.
- **89.1%** – The share of emails that were spam.
- **262 billion** – The number of spam emails per day (assuming 89% are spam).
- **2.9 billion** – The number of email accounts worldwide.
- **25%** – Share of email accounts that are corporate.

27% of humanity uses email

<http://royalpingdom.com/2011/01/12/internet-2010-in-numbers/>

Webmail Is ...

Easy to use +
Accessible from anywhere +
Free

Used by billions of people

Recipe for miracles

Replicate the Miracle ?

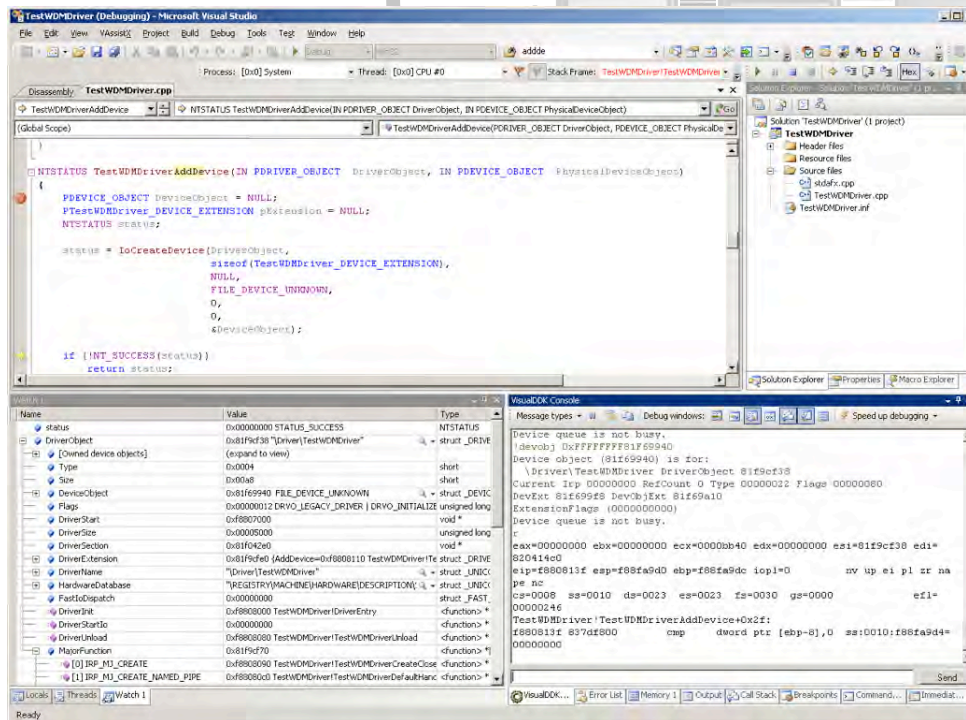
- How do we get all the world's programmers to use the very best testing tools?
- How do we empower end users to demand better quality from the software they use?

Software reliability as a service

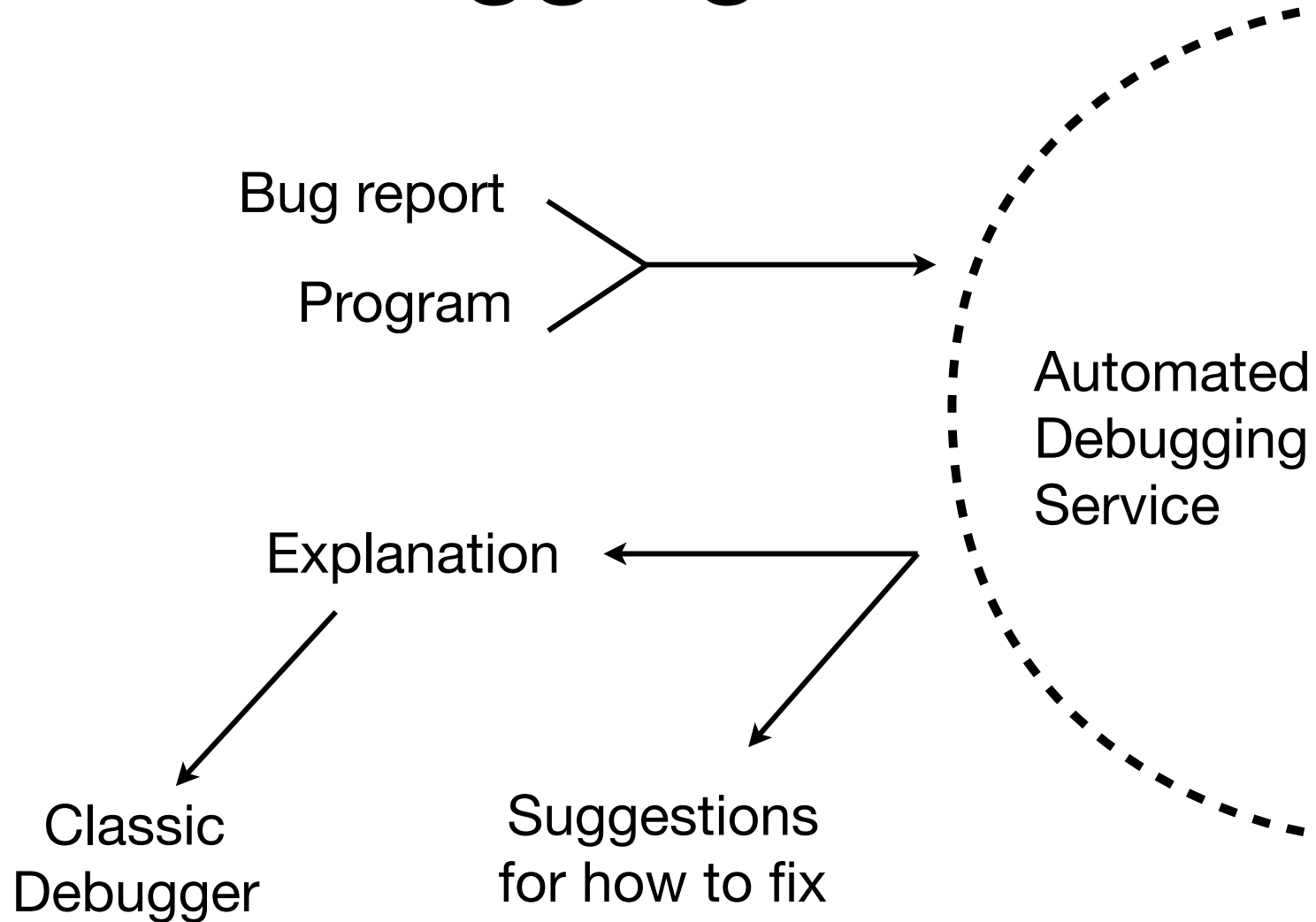
AutoSRS

1. Testing service
2. Debugging service
3. End-user service
4. Certification/ranking service

1. Testing Service



2. Debugging Service



3. End-User Service

Address: <http://test.epfl.ch>

Upload file

RTL8029.SYS

Type

Windows device driver

TEST

Bug Type	Severity	Description	Replay Trace
Resource leak	●●	Does not call NdisCloseConfiguration when initialization fails	<input type="button" value="DOWNLOAD"/>
Memory corruption	●●●●	Does not check range of MaximumMulticastList registry	<input type="button" value="DOWNLOAD"/>
Race condition	●●●●	Interrupt arriving before timer initialization uses uninitialized data	<input type="button" value="DOWNLOAD"/>
Segmentation fault	●●●●	Incorrectly handled unexpected OID in QueryInformation	<input type="button" value="DOWNLOAD"/>
Segmentation fault	●●●●	SetInformation does not handle unexpected OID	<input type="button" value="DOWNLOAD"/>

4. Certification/Rating Service

- Service for software consumers
 - evaluate software reliability automatically
 - publish results \Rightarrow enable product comparisons
 - explain & quantify software reliability to consumers
- No certification \Rightarrow liability for damages
 - do away with “AS IS” software licenses



Tools & Systems

Automated Testing

DDT

Test proprietary code

LFI

Test recovery code

ConfErr & WebErr

Human error testing

Automated Debugging

ESD

Execution synthesis

Portend

Date race classifier

Automated Correcting

Dimmunix

Deadlock immunity

RevNIC

Reverse engineering

Scalable Certification

iProve

End-user verifiability

iQA

Rating & certification

Core
Techs

Cloud9 - Cluster-based parallel symbolic execution

S²E - Selective symbolic execution of full software stacks

AutoSRS - Automated Software Reliability Services

<http://dslab.epfl.ch>