Towards Practical Application-level Support for Privilege Separation

(from ACSAC'22)

Nik Sultana
Illinois Institute of Technology

Henry Zhu
UIUC

Ke Zhong
University of Pennsylvania

Zhilei Zheng
University of Pennsylvania

Ruijie Mao
University of Pennsylvania

Digvijaysinh Chauhan
University of Pennsylvania

Stephen Carrasquillo
University of Pennsylvania

Junyong Zhao
University of Pennsylvania

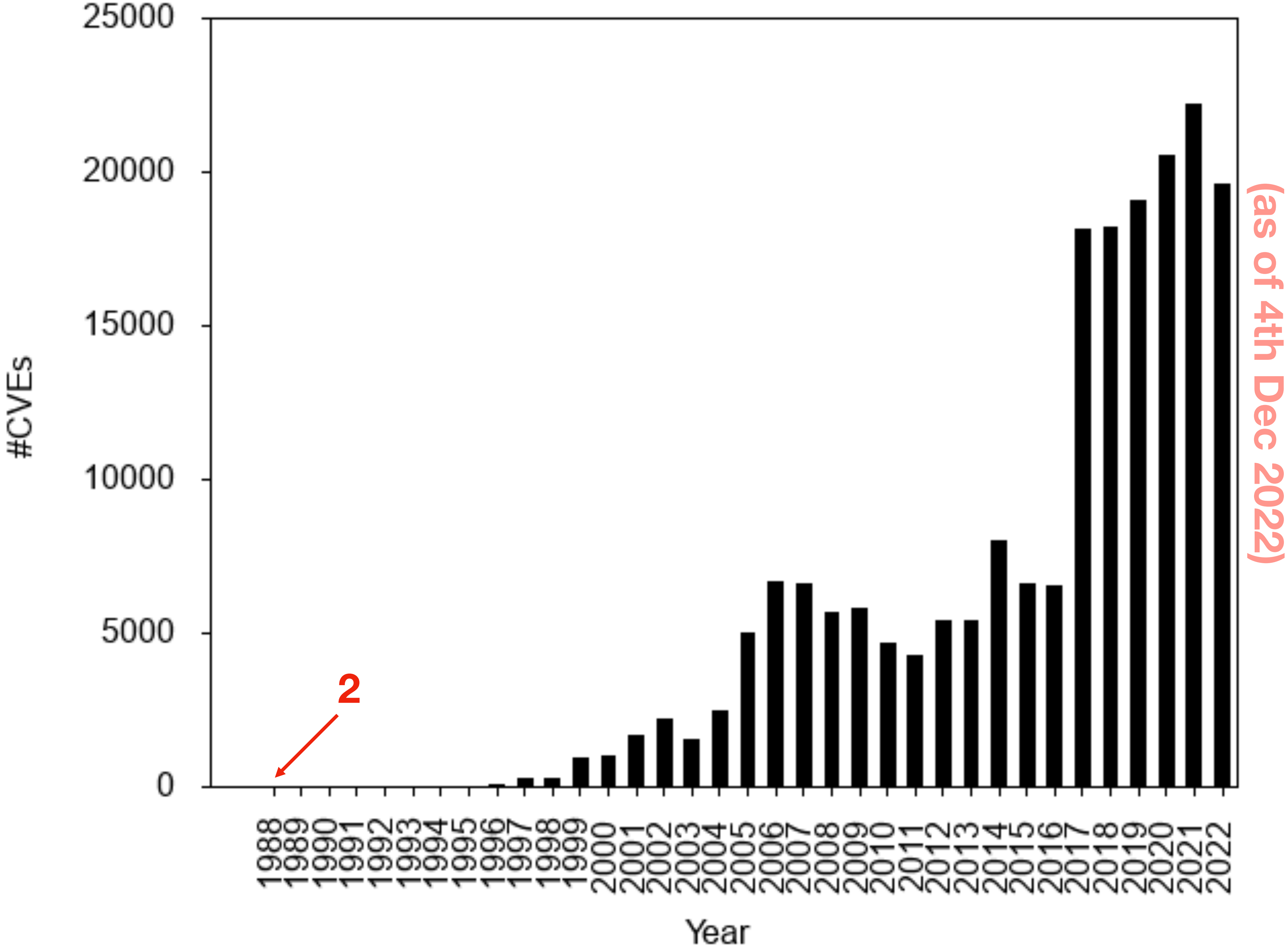
Lei Shi
University of Pennsylvania

Nikos Vasilakis
Brown University & MIT

Boon Thau Loo
University of Pennsylvania

HotSoS'23

Motivation: Software Security



Increased trend in # of CVEs:

Good: we know about problems.

Bad: there are more problems.

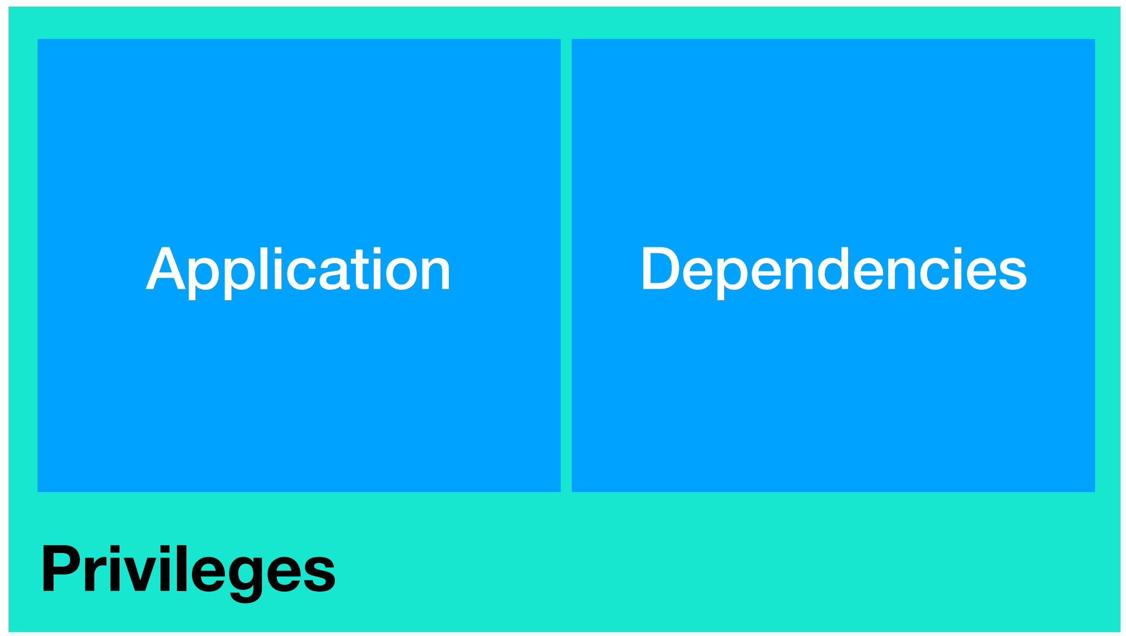
Ack: Graph generated using dataset from https://www.cve-search.org/dataset/

Software Security **Techniques**

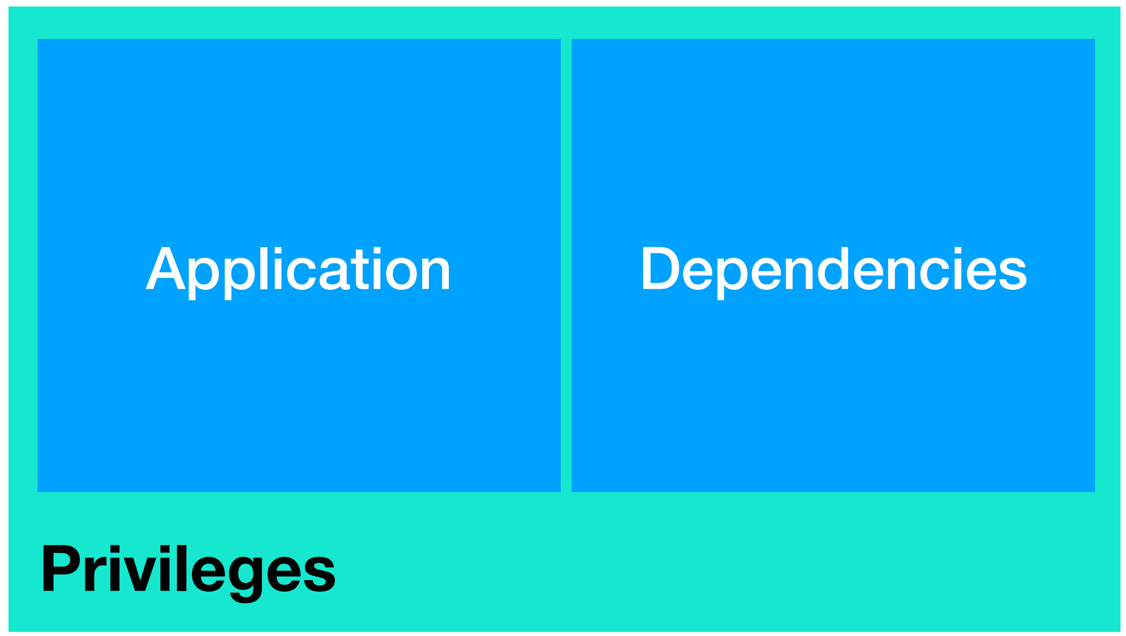
- Range of techniques available: ASLR, Stack canaries, Sandboxing, Soft/hard bounds checking, …
- Combining them is good practice.
  But some techniques are difficult to apply.

  We focus on one such technique: **privilege separation.**

What is Privilege Separation? (privsep)



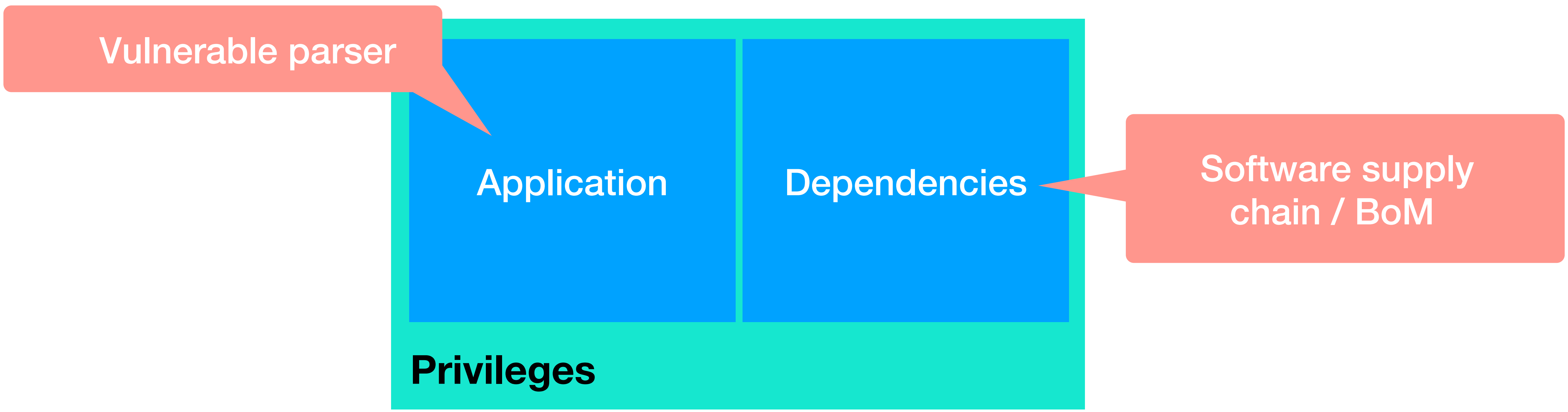
What is Privilege Separation? (privsep)



Heuristics for splitting software.

- **Compartmentalize code + data.** Early application: servers: SMTP, SSH.
- Monolithic application ➡ Concurrent set of cooperating programs.
    - Monolithic application: often common privileges throughout.
    - **Distributed system**: granularity of privilege allocation.

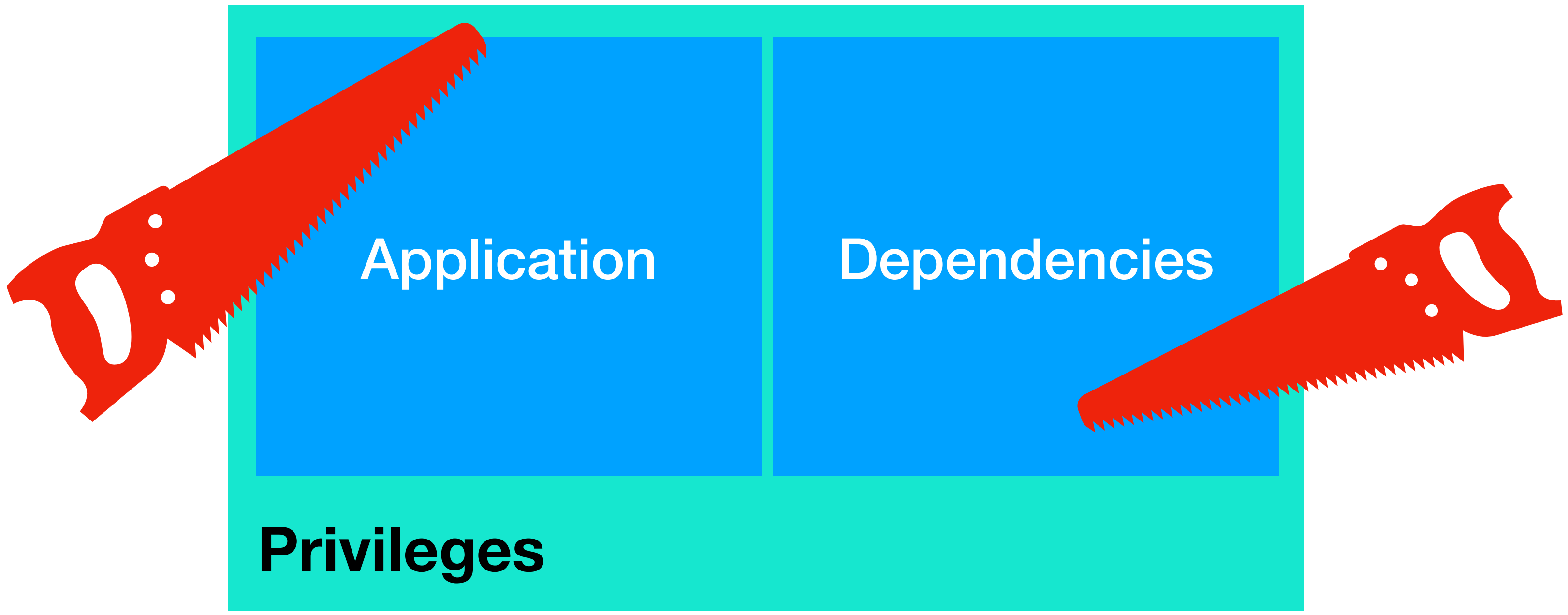
~~What is~~ Why Privilege Separation? (privsep)



- **Compartmentalize code + data.** Early application: servers: SMTP, SSH.
- Monolithic application ➡ Concurrent set of cooperating programs.

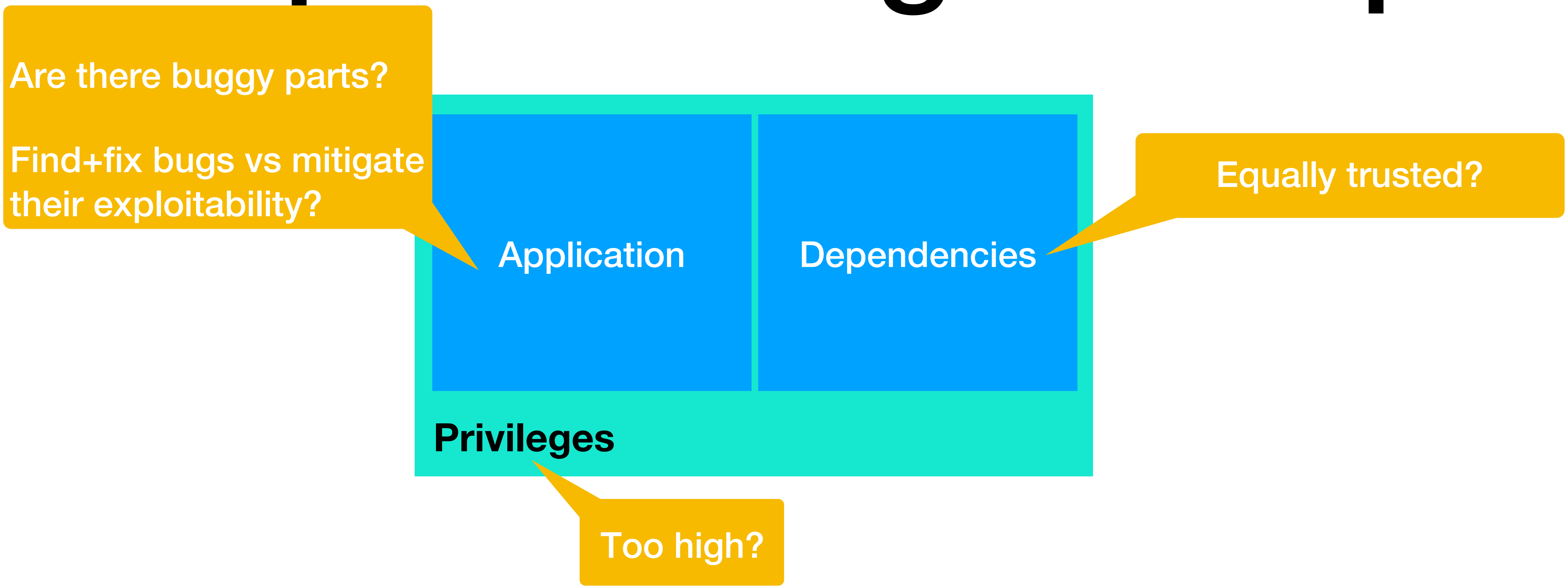
Main benefit: **vulnerability containment.**
Best case: if a vulnerability is exploitable, then fewer privileges can be abused.

Implementing Privsep



- **Implementing** privsep: usually a lot of work.
  Restructuring logic and code, positive and negative tests.
- Changing software without introducing bugs!
- There are many **decisions** to take (and retake later) wrt what+how to separate.

Implementing Privsep

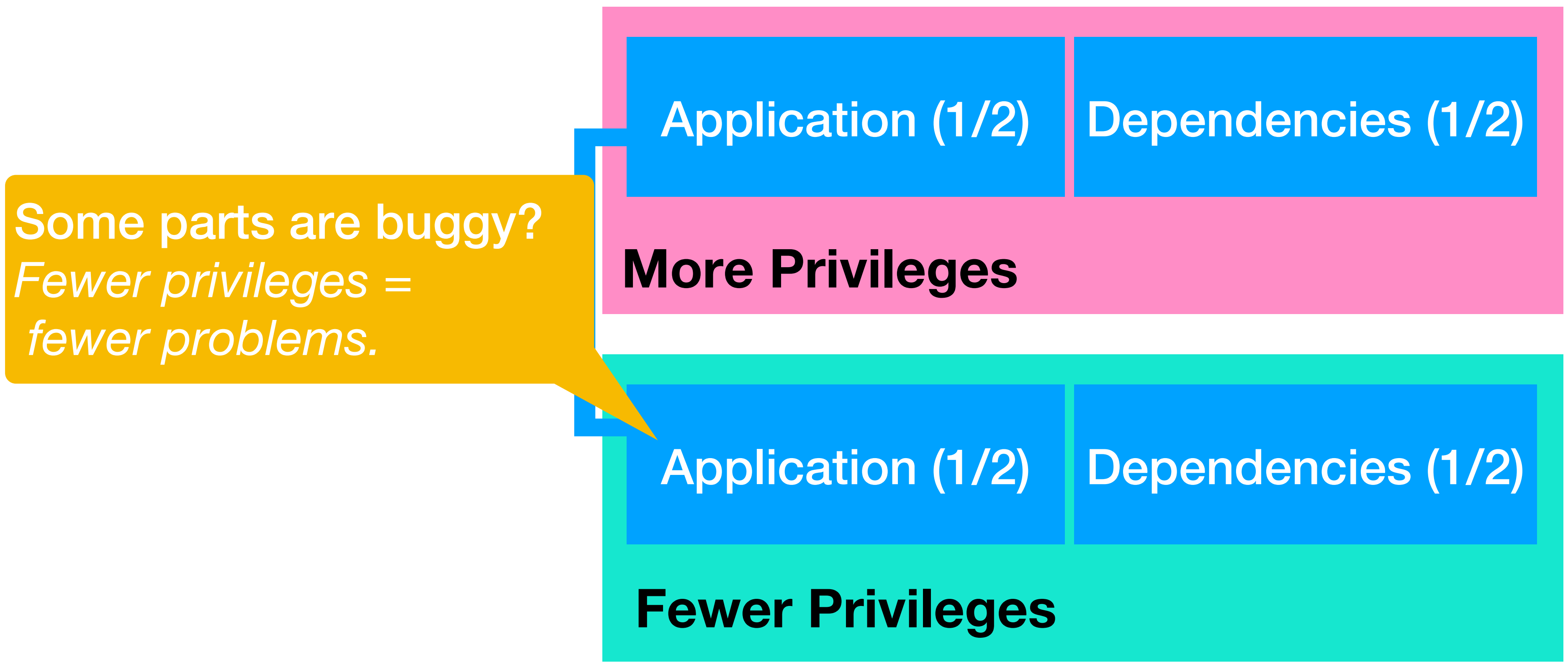


- **Implementing** privsep: usually a lot of work.
  Restructuring logic and code, positive and negative tests.

- Changing software without introducing bugs!

- There are many **decisions** to take (and retake later) wrt what+how to separate. (See yellow bubbles above)

What Privsep looks like

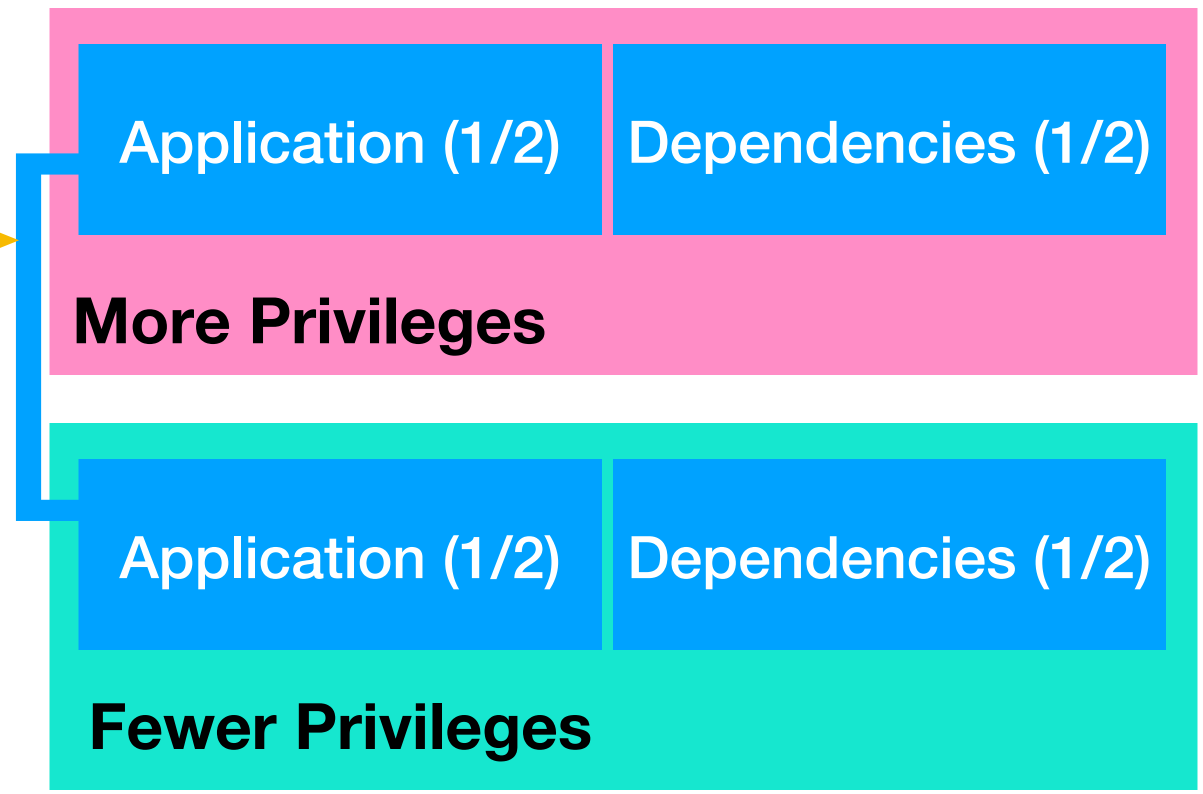


- Distributed system, heterogeneous privileges.

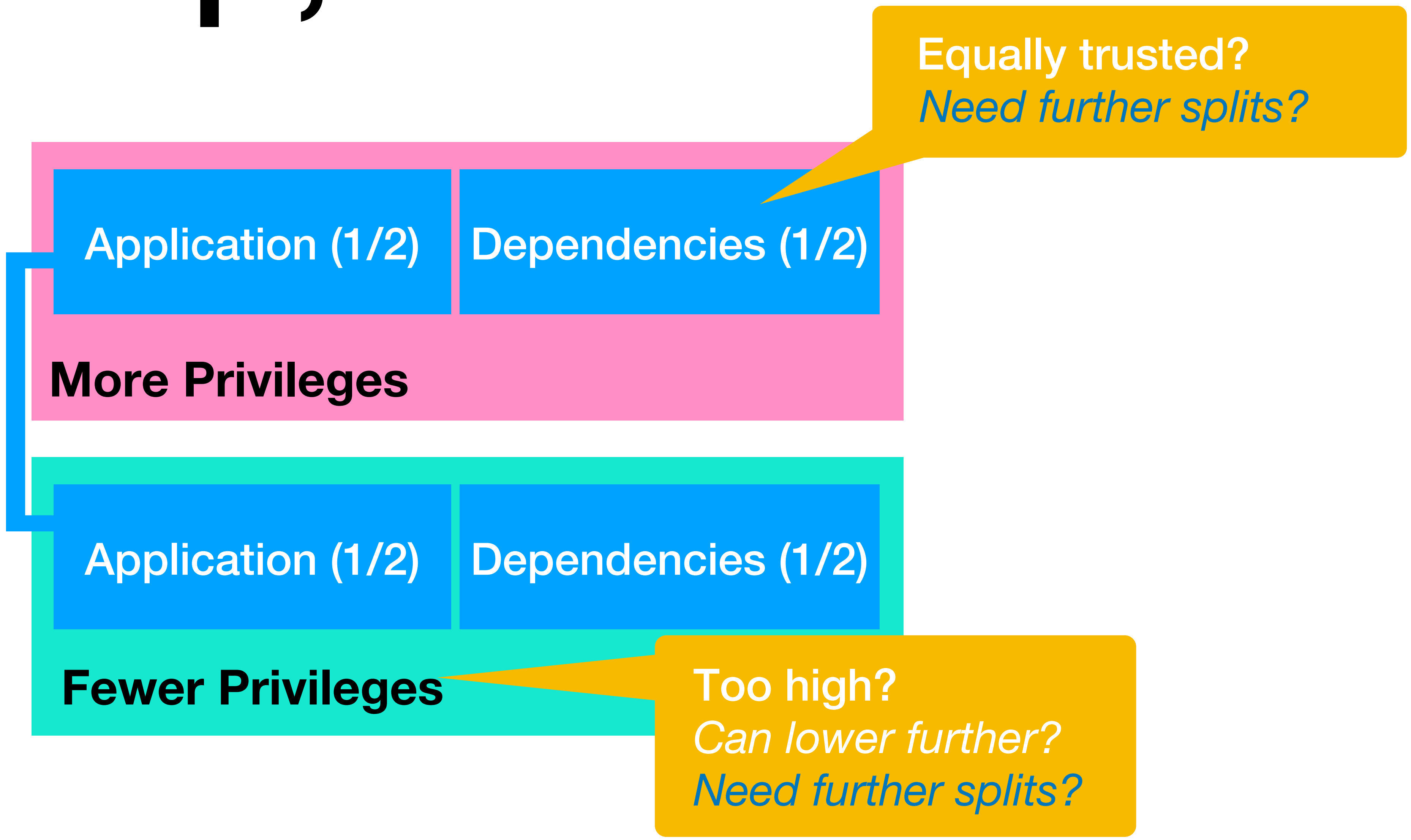
  Sometimes: separating between trusted vs untrusted.

What Privsep looks like

Heuristics:
- Components needing specific access.
- Dependencies incl. libraries.
- Cross-domain interfaces (e.g., parts of network, filesystem)



Privsep, *and then?*



- **Drawbacks** include:
  Inertia wrt **splitting software**, introduction of **new failure modes** (hello distributed systems), performance **overhead**, inertia wrt **maintainability and portability** (e.g., if use hardware enforcement).

Roles

2.1. People around Debian

There are several types of people interacting around Debian with different roles:

- **upstream author**: the person who made the original program.
- **upstream maintainer**: the person who currently maintains the program.
- **maintainer**: the person making the Debian package of the program.
- **sponsor**: a person who helps maintainers to upload packages to the official Debian package archive (after checking their contents).
- **mentor**: a person who helps novice maintainers with packaging etc.
- **Debian Developer** (DD): a member of the Debian project with full upload rights to the official Debian package archive.
- **Debian Maintainer** (DM): a person with limited upload rights to the official Debian package archive.

Please note that you can't become an official **Debian Developer** (DD) overnight, because it takes more than technical skill. Please do not be discouraged by this. If it is useful to others, you can still upload your package either as a **maintainer** through a **sponsor** or as a **Debian Maintainer**.

https://www.debian.org/doc/manuals/debmake-doc/ch02.en.html#reminders

Roles

Differences between **application** and **tool** *developers*.

Training

Generality, Accuracy, completeness

(Longstanding) Research Goal

Widely-applicable tool support for privsep

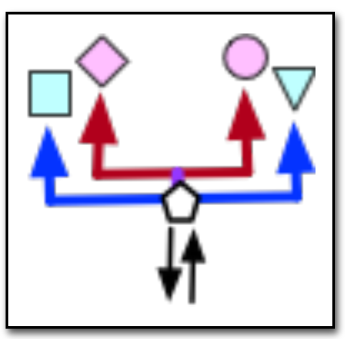
(This paper)

Foundations:
- compartment model
- tool infrastructure
- software-level

(Longstanding) Research Goal

Widely-applicable <u>tool support</u> for privsep

(This paper)



Artefacts:
+ tooling
+ several examples
+ supporting scripts
  & documentation

Foundations:
- compartment model
- tool infrastructure
- software-level

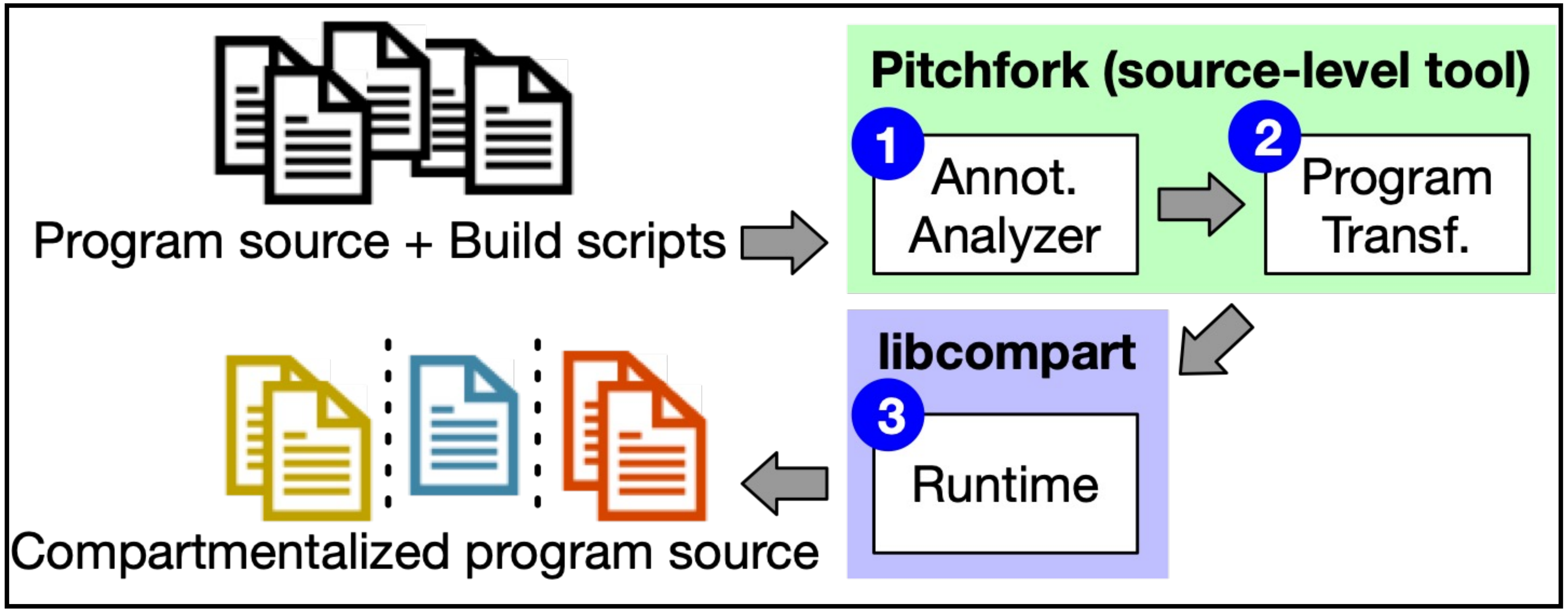
What’s different from prior art?

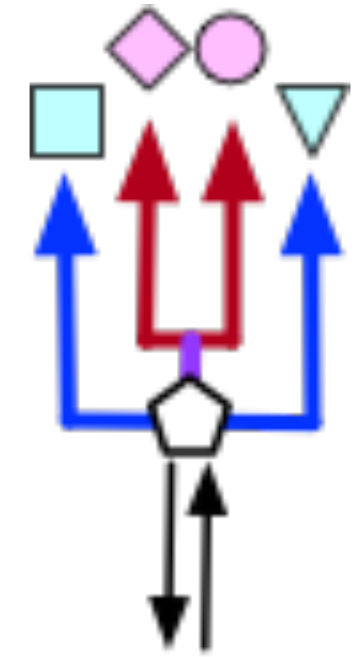
- **Separation “distance” + flexibility.**
  Separate binaries vs separate processes.
  Number of compartments.
  Commodity kernels and hardware.

- **Both tool and library.**
  Either can be used directly.
  Tool adapts code to use library.

- **Model-based approach.**
  Implemented abstractions provided/explained by the model.

Pitchfork

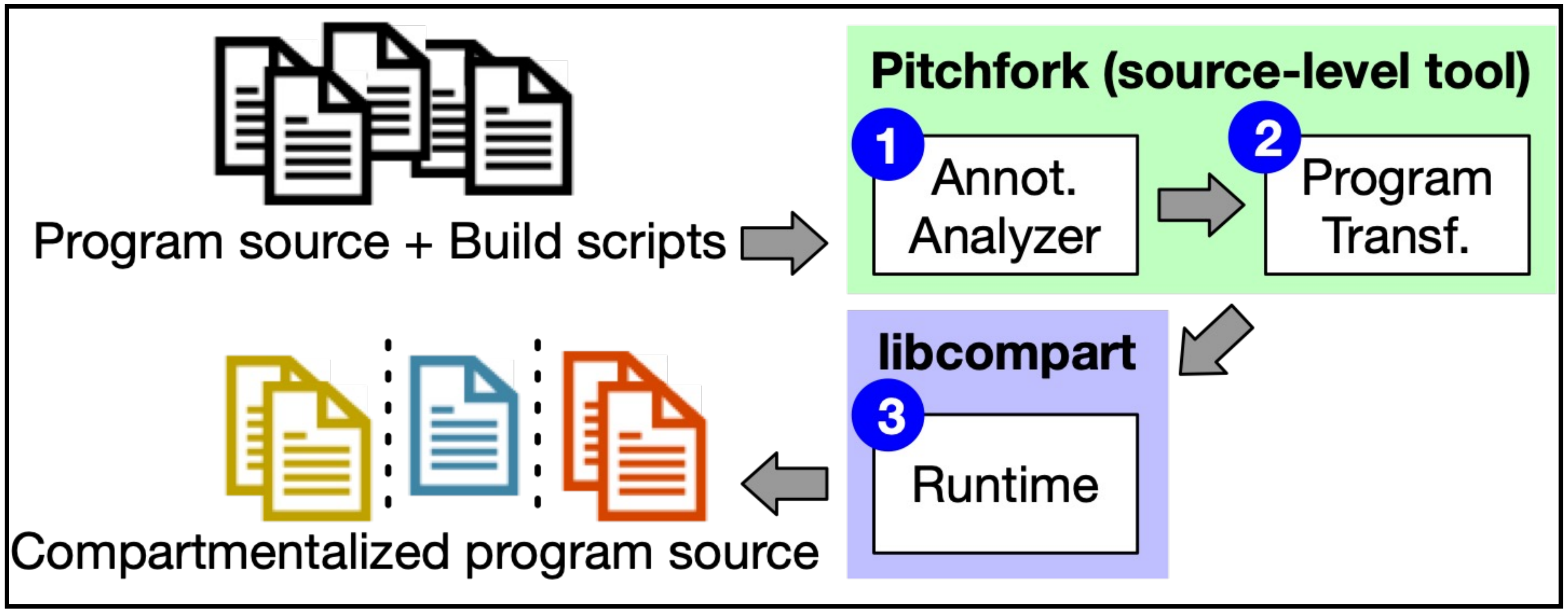


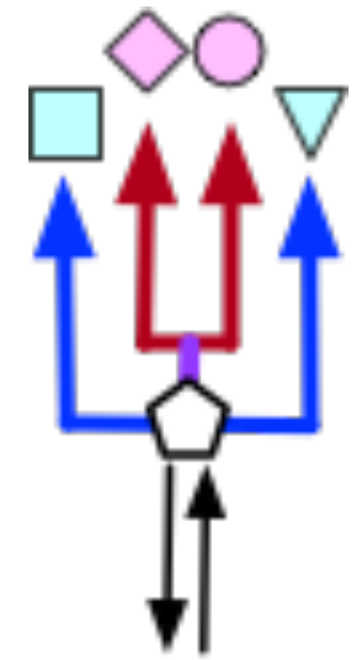


The **system** has two components based on a **model**:

- Pitchfork 1 2
- libcompart 3

Pitchfork





The **system** has two components based on a **model**:

- Pitchfork 1 2
- libcompart 3

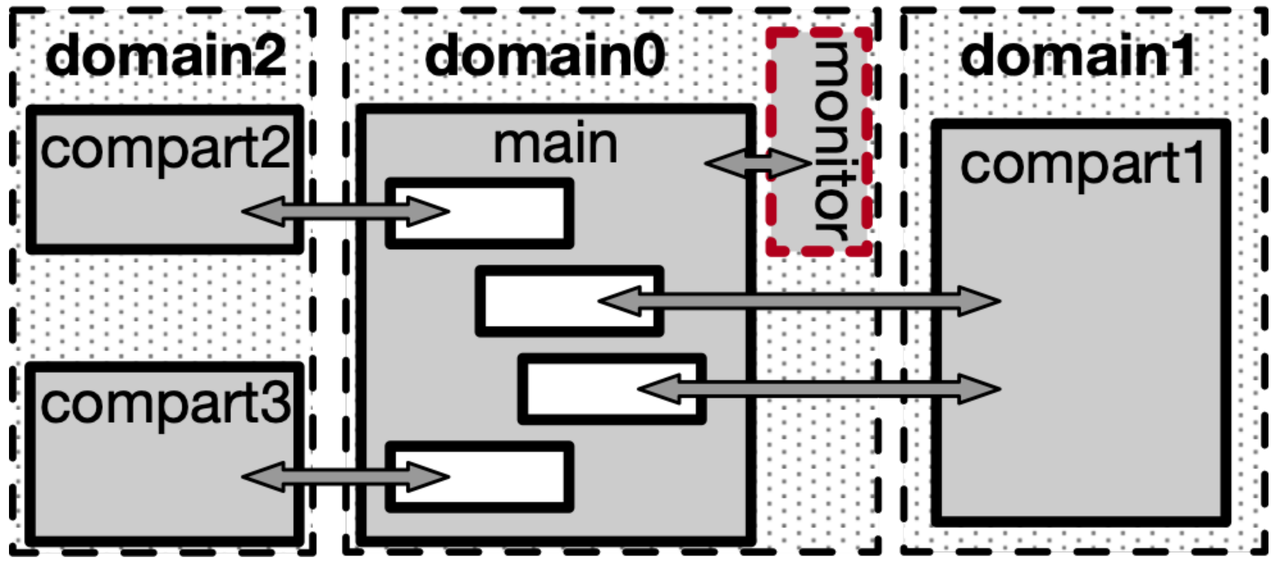
The **model** supports**:**

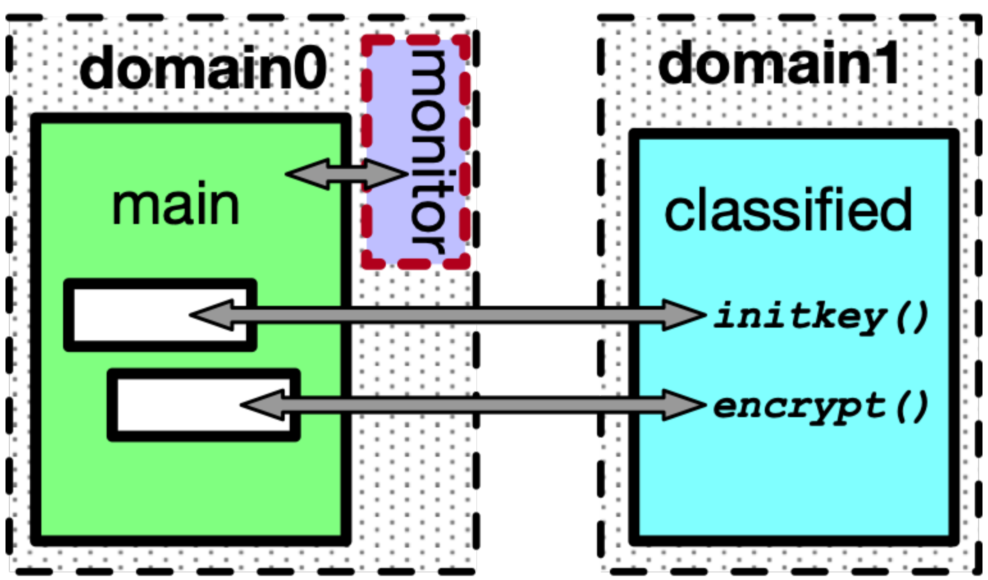
- Multiple compartments (different levels of trust)
- Synchronous communication
- Monitoring and failure-handling

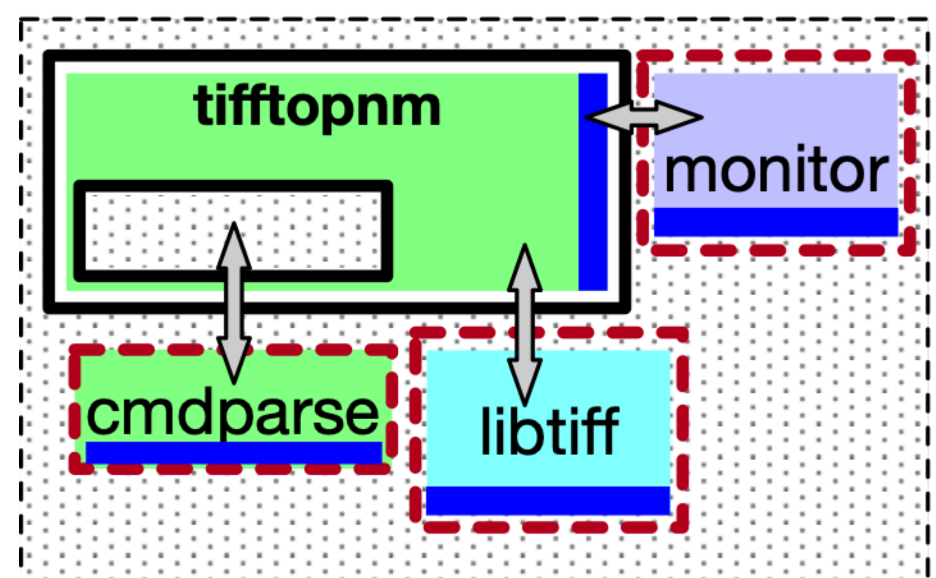
Pitchfork

```
if(console_type == BEEP_TYPE_CONSOLE) {
  pitchfork_start("Privileged");
  if(ioctl(console_fd, KIOCSOUND, period) < 0) {
    putchar('\a'); /* Output the only beep we can, in an effort to fall back on usefulness */
    perror("ioctl");
  }
  pitchfork_end("Privileged");
} else {
   /* BEEP_TYPE_EVDEV */
   struct input_event e;
   e.type = EV_SND;
   e.code = SND_TONE;
   e.value = freq;
   pitchfork_start("Privileged");
   if(write(console_fd, &e, sizeof(struct input_event)) < 0) {
     putchar('\a'); /* See above */
     perror("write");
   }
   pitchfork_end("Privileged");
}
```

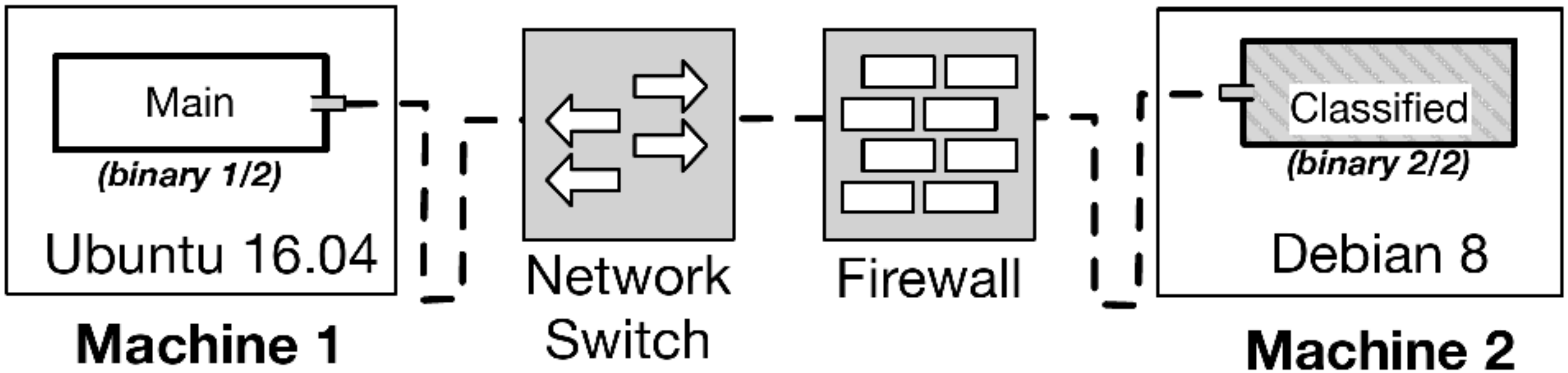
Compartment Model







Example of what's enabled



- Machine and network-level policy+enforcement.
- Communication channel over TCP.
- Organization:
  Domain: one on each machine
  Compartments: one in each domain.
  Segments: 2 in Classified, 1 in Main.

Evaluation

(Many more details in the paper)

- Applicability
    - Examples
    - Maintainability
    - Convenience
- Security
    - Known CVEs
    - Heuristics
- Overhead: running time, memory, binary size.

Evaluation

- Applicability
  - Examples
  - Maintainability
  - Convenience
- Security
  - Known CVEs
  - Heuristics
- Overhead: running time, memory, binary size.

Software	Plat.	Separation Goal
cURL	L	Command invocation, parsing, file transfer.
Evince	L	libspectre dependency—see §2.
git	L	Historical vulnerability [13].
ioquake3	m	Applying server updates.
tifftopnm	L	Separating parsers—see §C.
nginx	L	HTTP request parsing
redis	L	Isolating low-use commands.
tcpdump, uniq	F	Leveraging Capsicum [68].
Vitetris	L	Network-facing code—see §2.

Evaluation

- Applicability
    - Examples
    - Maintainability
    - Convenience
- Security
    - Known CVEs
    - Heuristics
- Overhead: running time, memory, binary size.

$$\text{SAR} = \frac{\text{\#LOC Synthesized}}{\text{\#Lines of Annotation}}$$

Soft.	#LOC	#Annot.	#LOC Synthesized		**SAR**
			Compart.	De/marsh.	
beep	372	9	133	245	42
PuTTY	123K	6	52	29	13.5
wget[6]	62.6K	3	65	168	77.7
wget[7]	62.8K	8	57	38	11.9

Evaluation

- Applicability
  - Examples
  - Maintainability
  - Convenience
- Security
  - Known CVEs
  - Heuristics
- Overhead: running time, memory, binary size.

Software	CVE-*-*	Vulnerability
beep	2018-0492	Race condition
PuTTY	2016-2563	Stack buffer overflow
wget	2016-4971	Arbitrary file writing
wget	2017-13089	Stack buffer overflow

System release

- http://pitchfork.cs.iit.edu
- Everything is released except for exploit code:
    - libcompart
    - Pitchfork
    - examples of applying libcompart & Pitchfork
    - FreeBSD ports analysis
- Apache 2.0 license

Follow-up work

A Domain-Specific Language for Reconfigurable, Distributed Software Architecture

Henry Zhu
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, IL, USA

Junyong Zhao
Department of Computer Science
University of Arizona
Tucson, AZ, USA

Nik Sultana
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA

***Abstract*—A program's architecture—how it organizes the invocation of application-specific logic—influences important program characteristics including its scalability and security. Architecture details are usually expressed in the same programming language as the rest of a program, and can be difficult to distinguish from non-architecture code. And once defined, architecture is difficult and risky to change because it couples tightly with application logic over time.**

We introduce C-Saw: an approach to express a software's architecture using a new embedded domain-specific language (EDSL) designed for that purpose. It *decouples* application-specific logic from architecture, making it easier to identify architectural details of software. C-Saw leverages three ideas: (i) introducing a new, formally-specified EDSL to separate an application's architecture description from its programming language; (ii) reducing architecture implementation to the definition and management of distributed key-value tables, and (iii) introducing an expressive state-management abstraction for distributed applications.

We describe a prototype implementation of C-Saw for C programs and use it to build end-to-end examples of expressing and changing the architecture of widely-used, third-party software. We evaluate this on Redis, cURL, and Suricata and find that C-Saw provides expressiveness and reusability, requires fewer lines of code when compared to directly using C to express architectural patterns, and imposes low performance overhead on typical workloads.

***Index Terms*—Key-Value Tables, Process Algebra, Coordination Language, Domain-Specific Language**

I. Introduction

Software's architecture describes its fundamental information-processing structure [1] and varies in its complexity. Examples of architecture include: a sequence of processing steps, a pipeline of concurrent stages, an event-handling system, a fan-out to worker instances, and a mix of these patterns [2].

The choice of architecture influences important software characteristics such as security [3] and performance [4]. For example, architecture affects how software can scale to meet demand by harnessing additional resources to distribute the

The blurring of architecture and logic complicates the implementation of important features that depend on architecture-level changes. Fig. 1 shows examples of such features which include caching and load-balancing.

As a result of architecture's poor visibility in source code and its coupling with non-architecture code, architecture-level changes are *high-friction*: they take effort, risk introducing bugs, and create a maintenance burden if the software diverges from an up-stream, canonical open-source version. One could avoid architecture-level change by designing an overly-general architecture to begin with, but this raises practitioners' red flags because it risks "premature optimization" [6], "creeping elegance" [7], and introducing a "bad smell" from needless complexity due to "speculative generality" [8]. Even then, general interfaces might not forestall the need for eventual revision since the software's requirements can evolve.

To avoid these problems, we need a low-friction method to express software's architecture. It needs to support a range of architecture patterns, be linguistically distinguished from application logic, and induce low overhead. New and existing software could then be adapted more easily to respond to new and changing needs that require architecture-level changes.

In this paper, we introduce C-Saw ("see-saw"): an approach to express a software's architecture using a new Embedded domain-specific language (DSL) designed for that purpose. C-Saw relies on distributed key-value tables to track both architecture-related state and application-logic state. These tables are managed by DSL expressions. The DSL is inlined into the application source-code and it is designed to work with existing software and languages—we prototyped this for the C language and developed usage examples involving widely-used, third-party applications.

The DSL can express a set of *architectures* that serve commonly-occurring *needs* such as those serviced by the examples in Fig. 1. These needs include: **(i)** *availability* through fail-over or replication; **(ii)** *manageability* through live migration or scale-out; **(iii)** *performance* through caching for

$\mathsf{InstanceTypes} = \{\tau_f,\ \tau_g\}$
$\mathsf{Instances} = \{f : \tau_f,\ g : \tau_g\}$
$\mathbf{def}\ \mathsf{main}()\ \blacktriangleleft\ \mathsf{start}\ f(g) + \mathsf{start}\ g(f)$
$\mathbf{def}\ \tau_f :: \mathit{junction}(\overline{g})\ \blacktriangleleft$
$\quad |\ \mathbf{init\ prop}\ \neg\mathrm{Work}$
$\quad |\ \mathbf{init\ data}\ n$
$\quad \lfloor H_1 \rceil;\ \mathsf{save}(\ldots,\ n);$
$\quad \mathsf{write}(n,\ \overline{g});$
$\quad \mathsf{assert}\ [\overline{g}]\ \mathrm{Work};$
$\quad \mathsf{wait}\ []\ \neg\mathrm{Work};$
$\mathbf{def}\ \tau_g :: \mathit{junction}(\overline{f})\ \blacktriangleleft$
$\quad |\ \mathbf{init\ prop}\ \neg\mathrm{Work}$
$\quad |\ \mathbf{init\ data}\ n$
$\quad |\ \mathbf{guard}\ \mathrm{Work}$
$\quad \mathsf{restore}(n,\ \ldots);$
$\quad \lfloor H_2 \rceil;$
$\quad \mathsf{retract}\ [\overline{f}]\ \mathrm{Work};$

Ack: **Henry Zhu, Junyong Zhao**

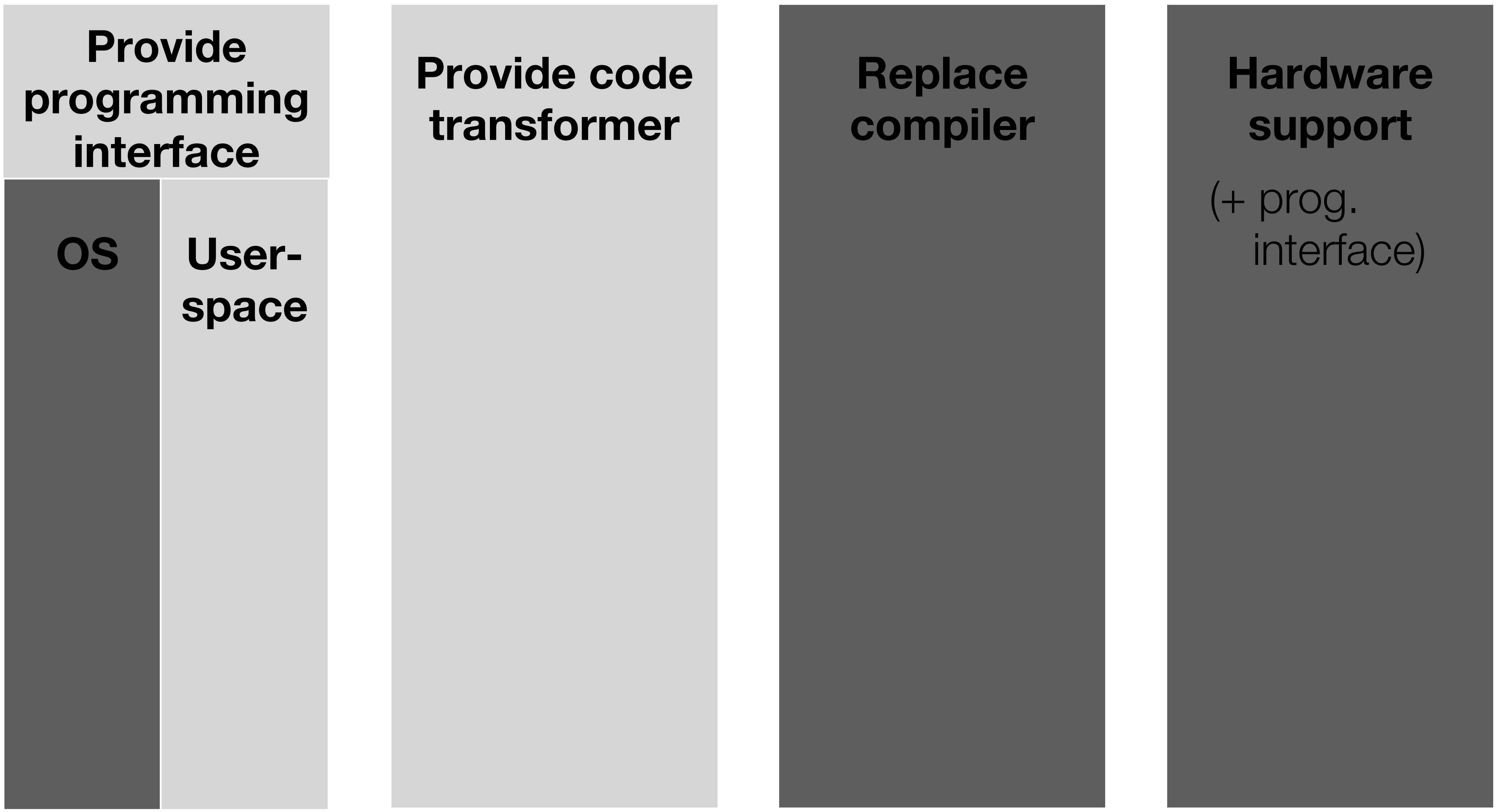

ILLINOIS TECH
Computer Science
Department
Nik Sultana
http://www.cs.iit.edu/~nsultana1

Extra slides

Related work

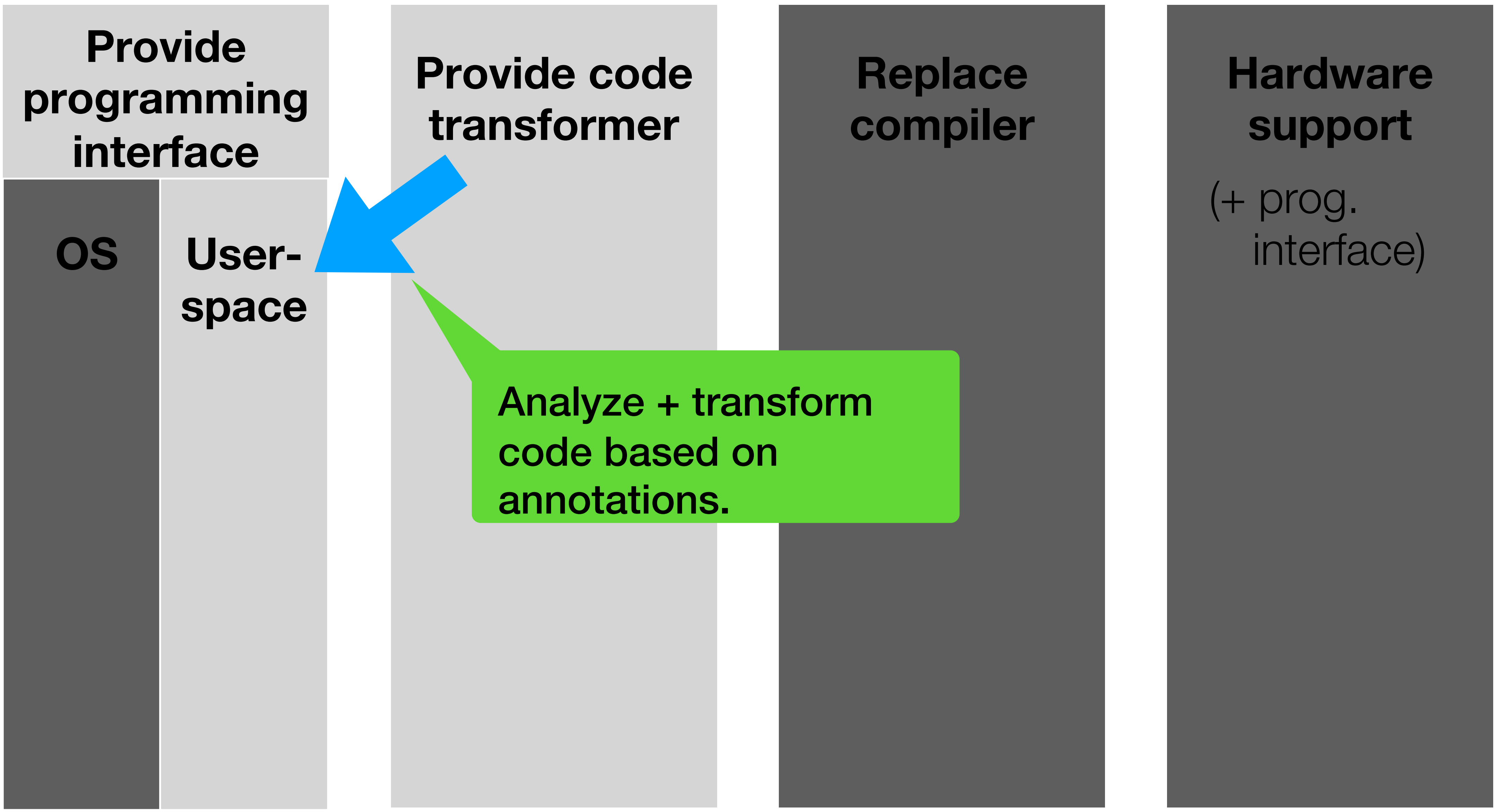
Provide programming interface		Provide code transformer	Replace compiler	Hardware support
OS	**User-space**			(+ prog. interface)

Related work



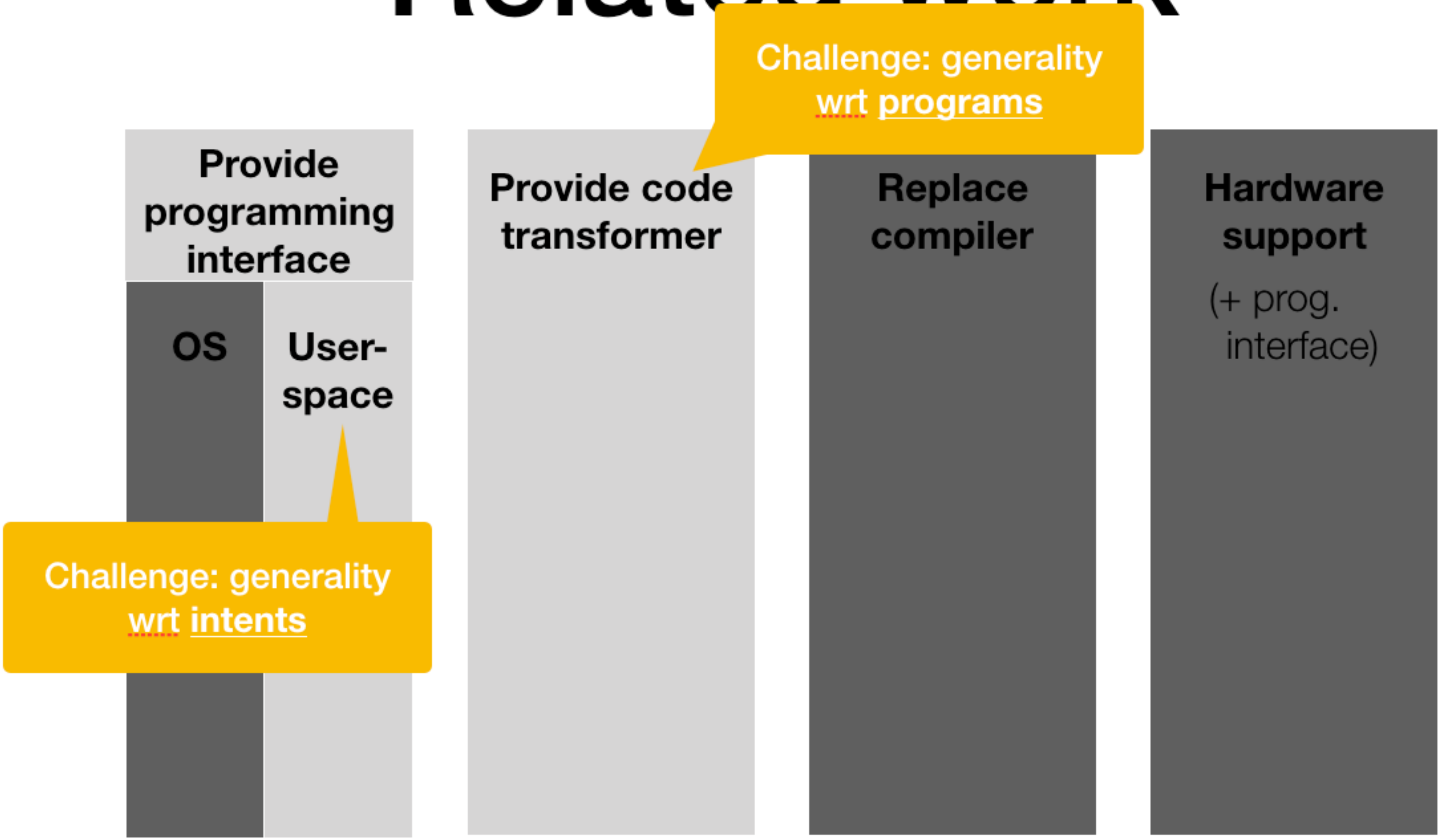
Focus: preserve portability,
lessen splitting/rewriting effort.

Related work



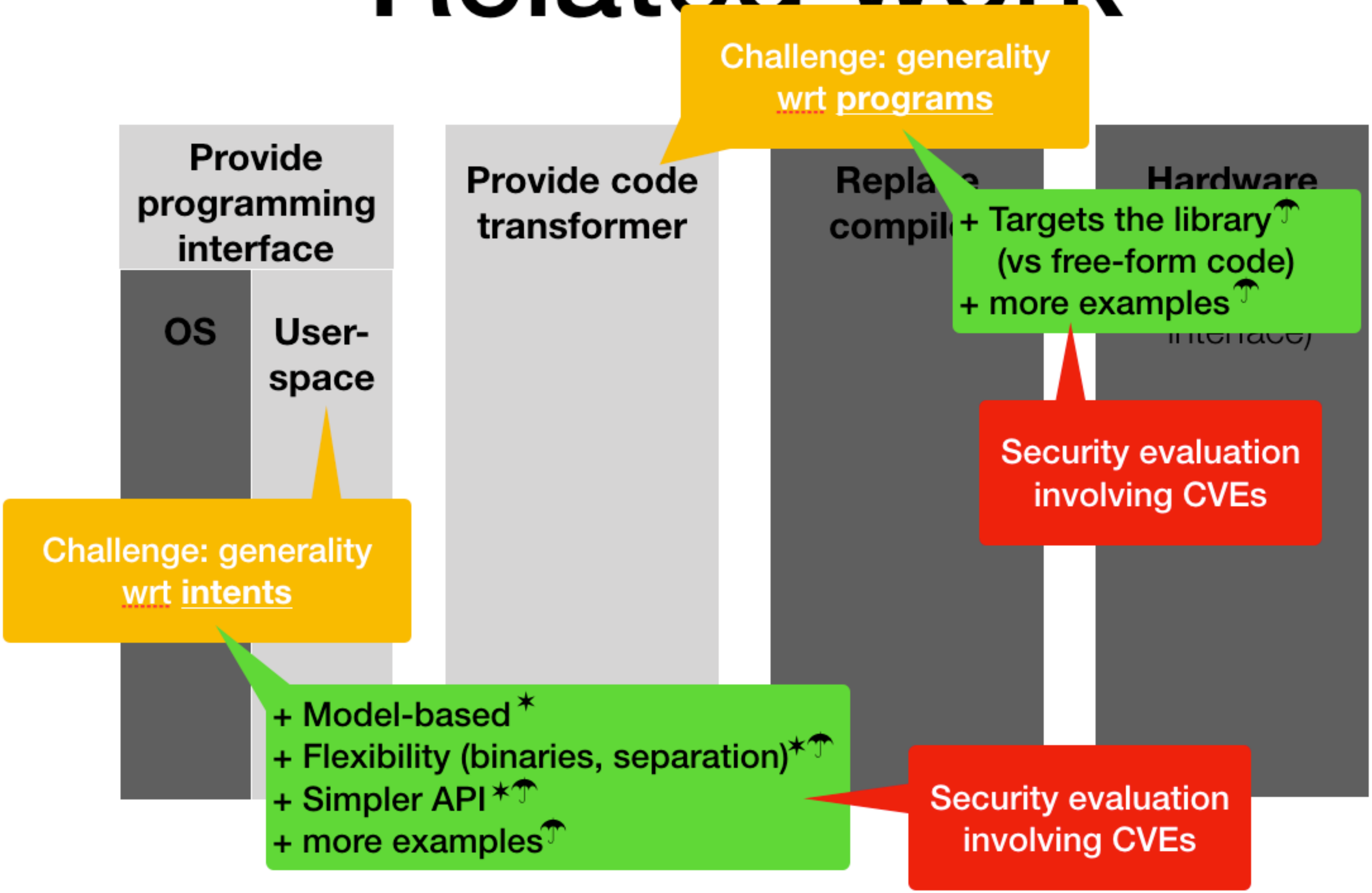
Best of both worlds: flexibility and convenience (automation).
Can inspect generated code.

Related work

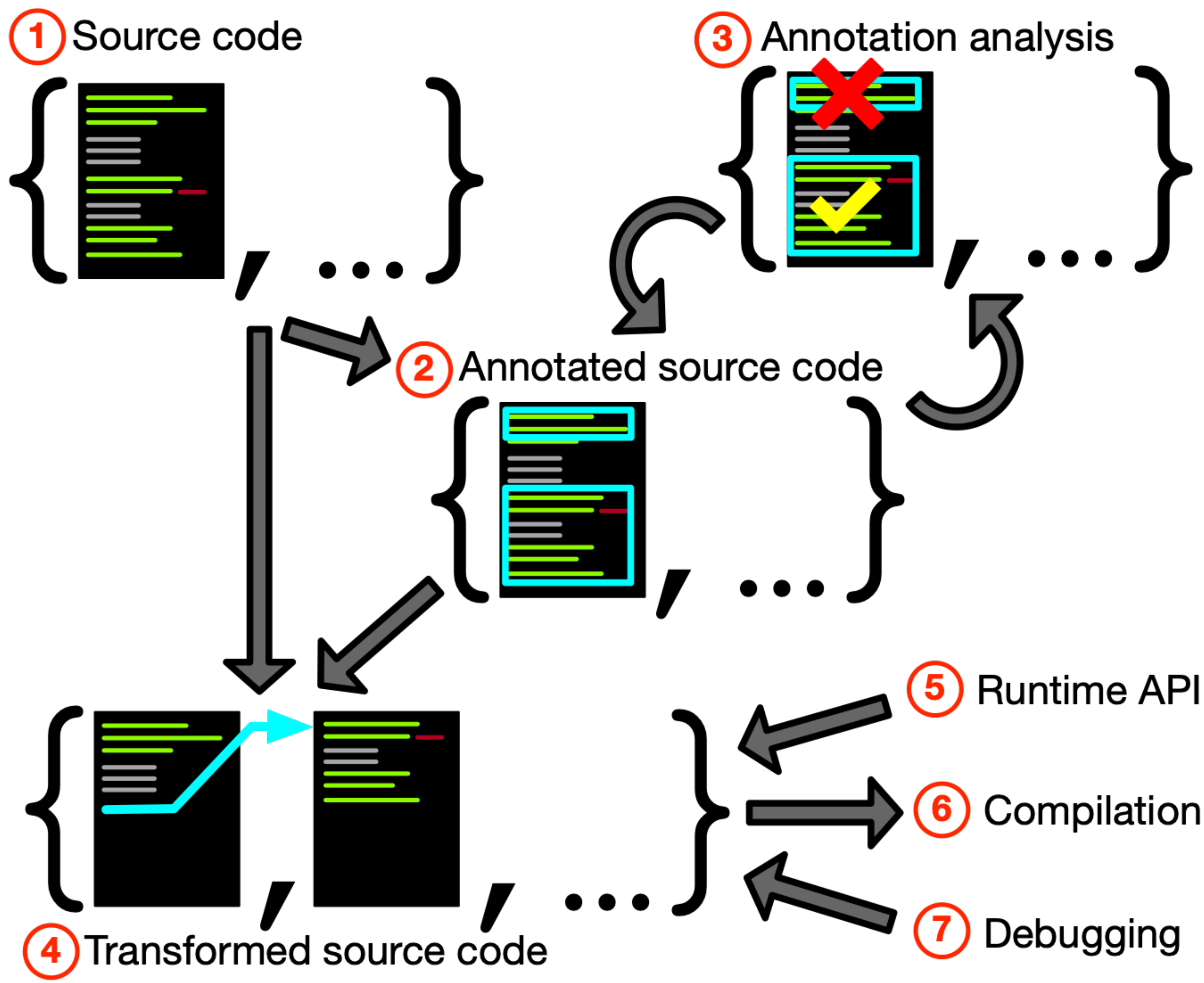


✶Technical contribution
☂Artifact contribution

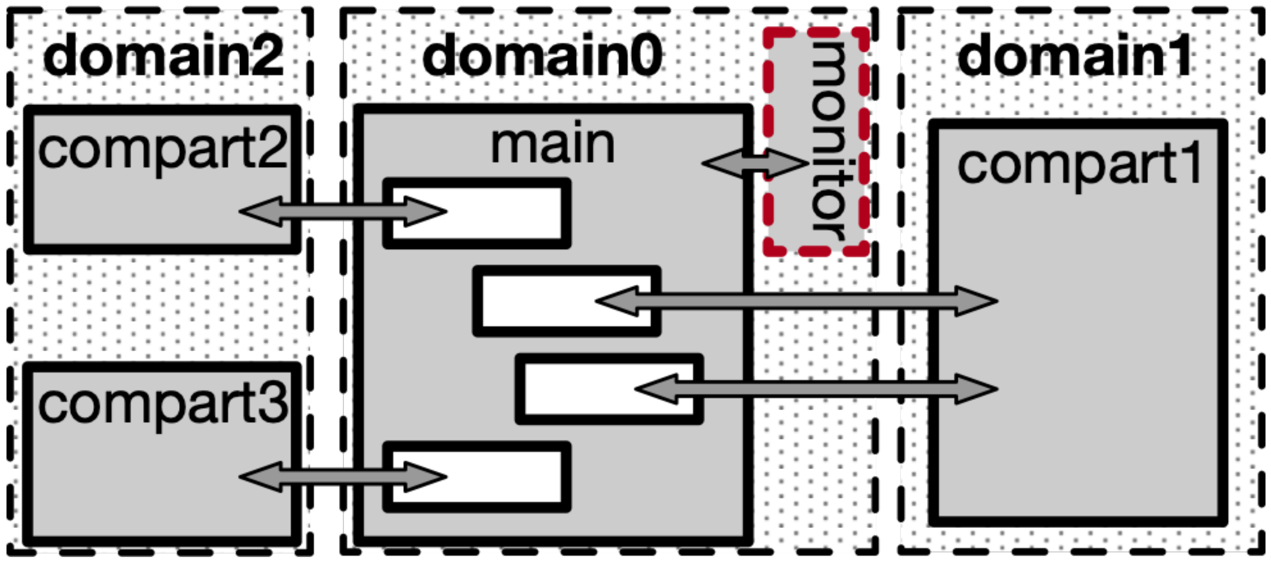
Related work



Pitchfork

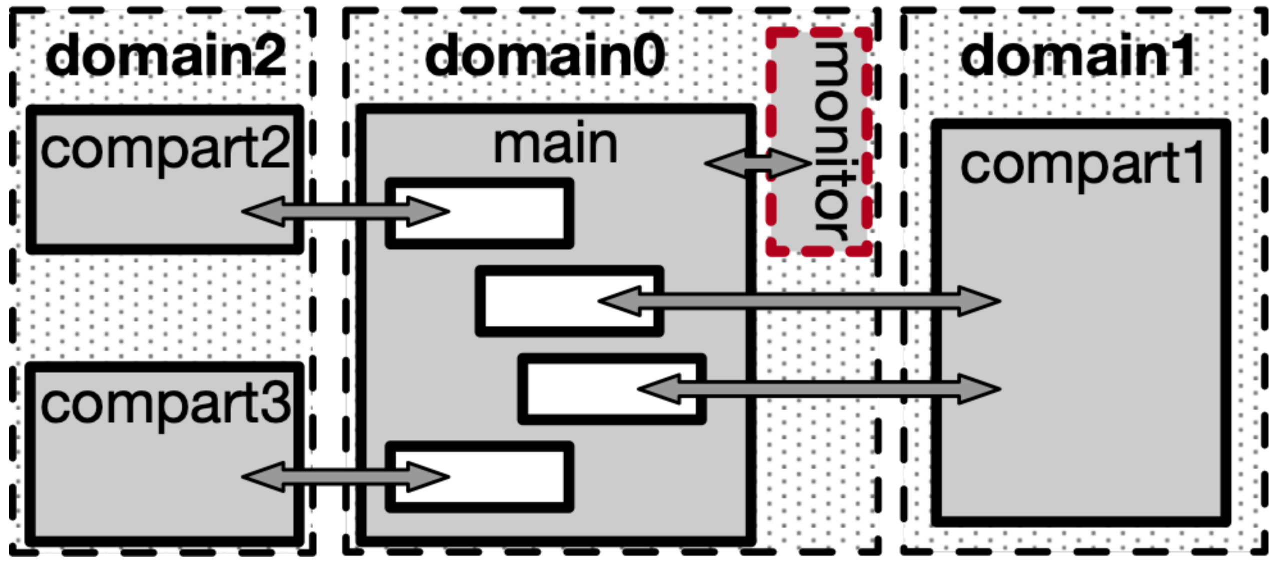

1 Source code
3 Annotation analysis
2 Annotated source code
5 Runtime API
6 Compilation
7 Debugging
4 Transformed source code

Compartment Model



- Organization:
  Domain: Shared memory/handles/resources across compartments
  Compartments: Sharing across segments.
  Segments: code + data.

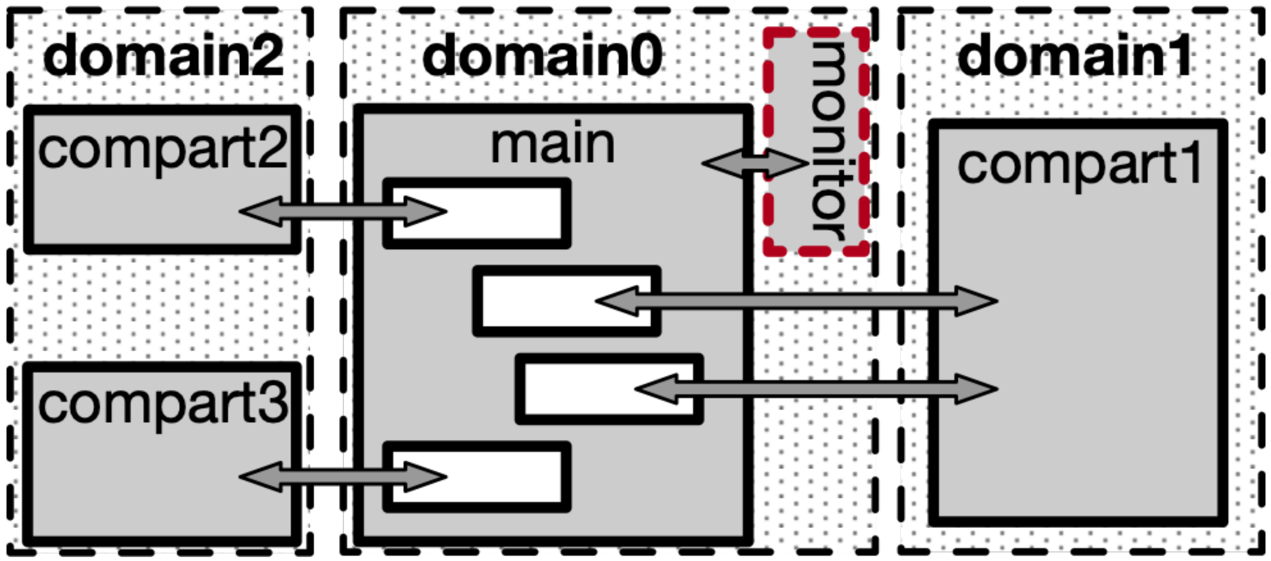
Compartment Model



$\in$

- Organization:
  Domain: Shared memory/handles/resources across compartments
  Compartments: Sharing across segments.
  Segments: code + data.

Compartment Model

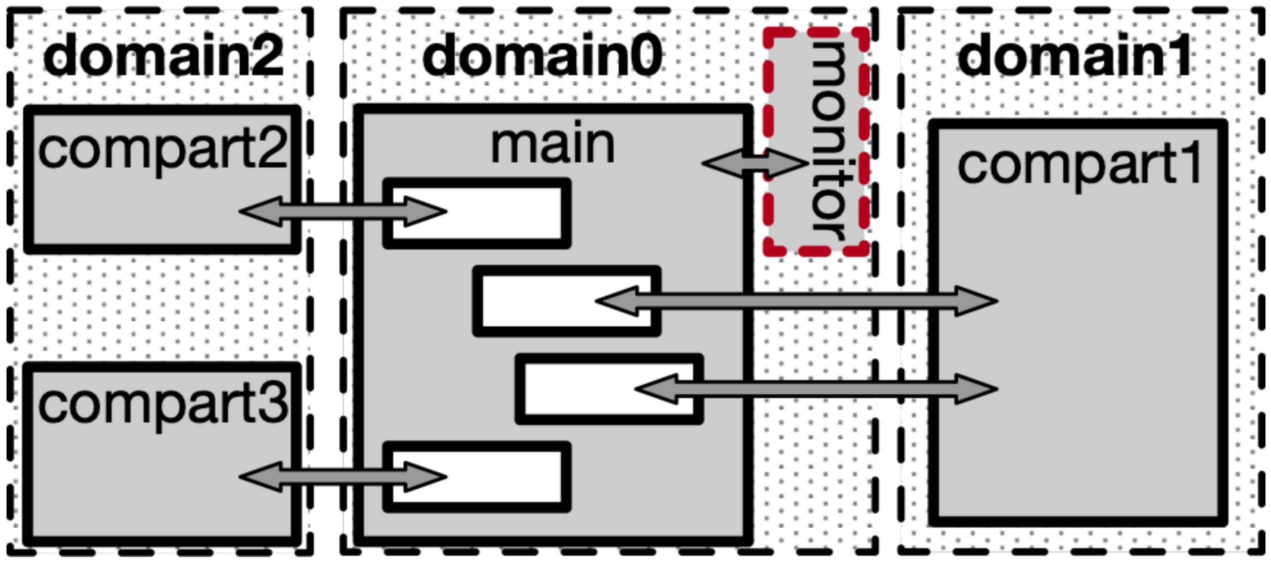


∈

- Organization:
  Domain: Shared memory/handles/resources across compartments
  Compartments: Sharing across segments.
  Segments: code + data.

∈

Compartment Model



- Organization:
  Domain: Shared memory/handles/resources across compartments
  Compartments: Sharing across segments.
  Segments: code + data.

- **Special compartments**: Main, Monitor — always in domain0.

- Implementation: pluggable API for communication, configuration and enforcement.

- Generalization and Tooling
  vs Flexibility:
  General but restrictive

libcompart

```
+#include "netpbm_interface.h"
int
main(int argc, const char * argv[]) {
  +compart_init(NO_COMPARTS, comparts, default_config);
  +convertTIFF_ext = compart_register_fn("libtiff", &
        ext_convertTIFF);
  +parseCommandLine_ext = compart_register_fn("cmdparse"
        , &ext_parseCommandLine);
  +compart_start("netpbm");

  struct CmdlineInfo cmdline;
  TIFF * tiffP;
  FILE * alphaFile;
  FILE * imageoutFile;

```

libcompart

```
pm_proginit(&argc, argv);
-parseCommandLine(argc, argv, &cmdline);
+struct extension_data arg;
+args_to_data_CommandLine(&arg, argc, argv);
+arg = compart_call_fn(parseCommandLine_ext, arg);
+args_from_data(&arg, &cmdline);
-tiffP = newTiffImageObject(cmdline.inputFilename);
-if (cmdline.alphaStdout)
...
-TIFFClose(tiffP);
+args_to_data(&arg, &cmdline);
+arg = compart_call_fn(convertTIFF_ext, arg);
pm_strfree(cmdline.inputFilename);
```