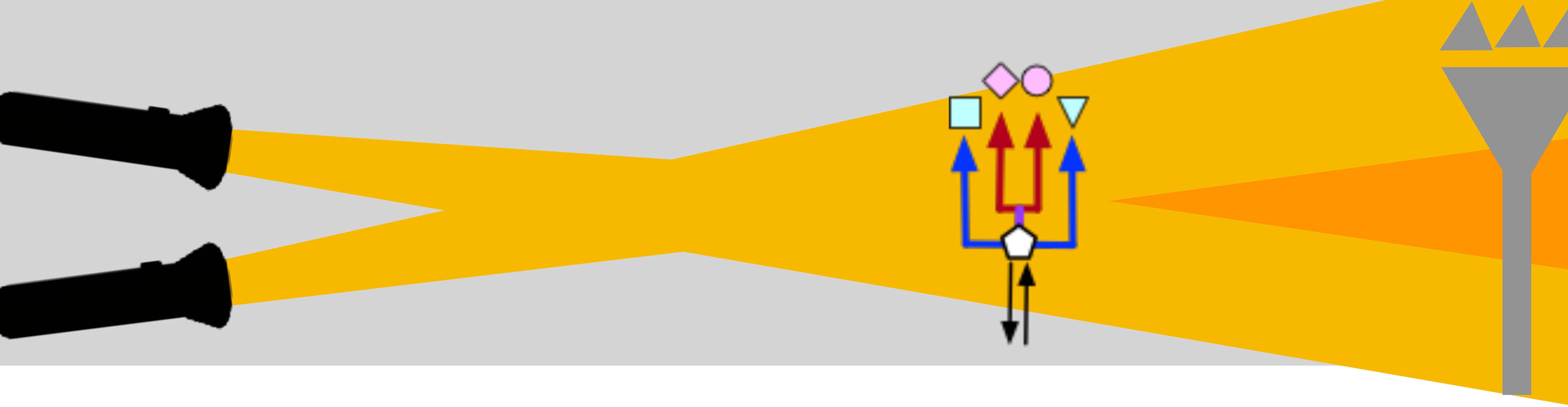


# Torches on Pitchfork: Multi-feature Evaluation of a Security-oriented Programming Toolchain

Nik Sultana

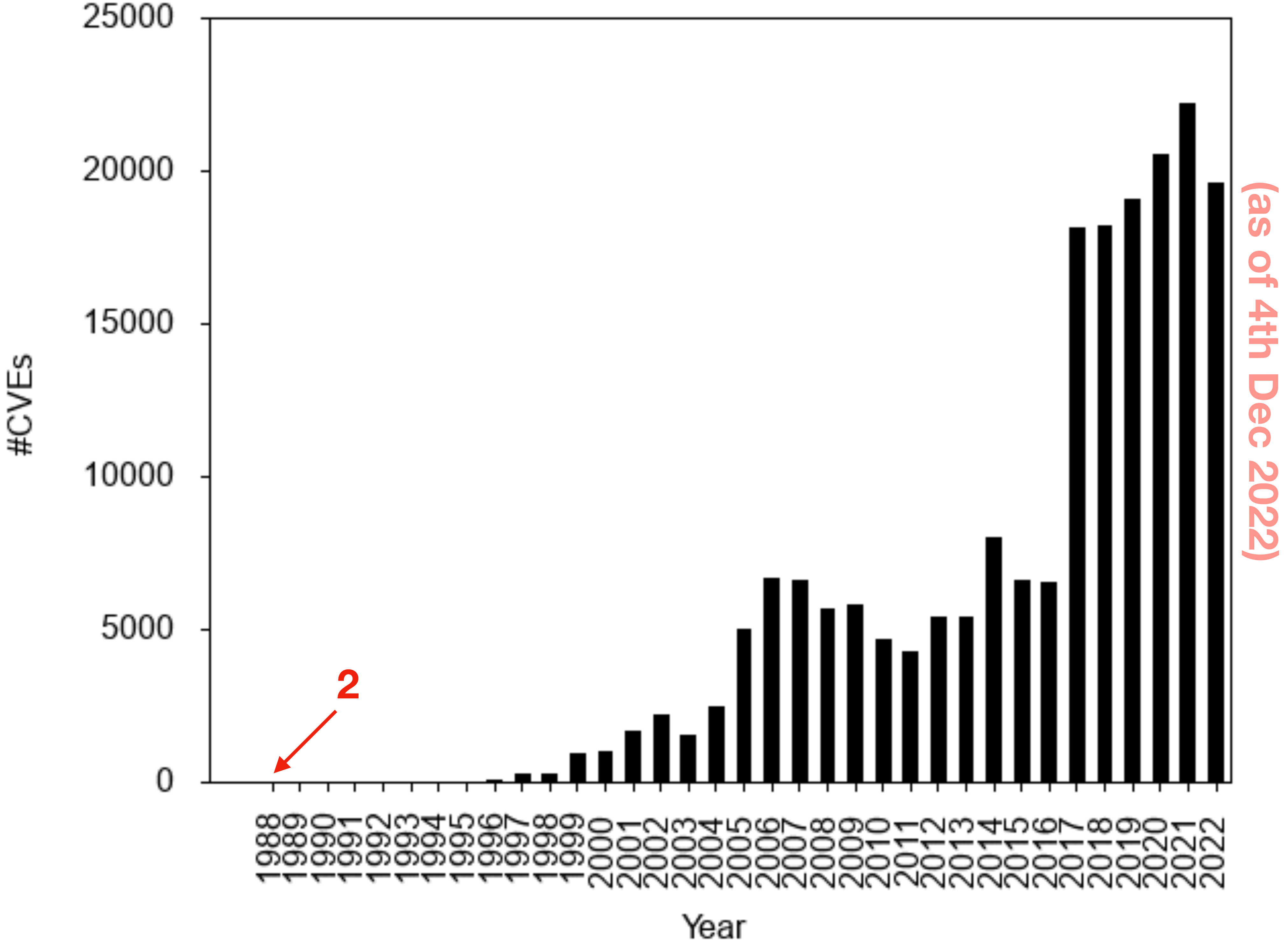
*Illinois Institute of Technology*

Learning from Authoritative Security Experiment Results (LASER) 2022

# 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

# Motivation: Software Security



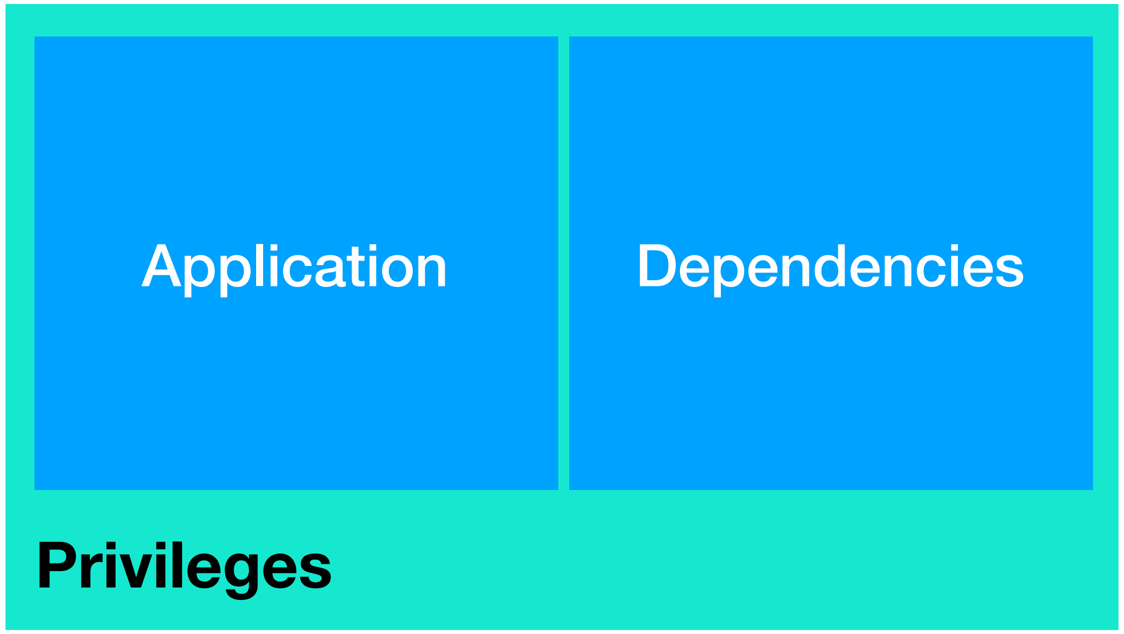
**Increased trend in # of CVEs:**

Good: we know about problems.

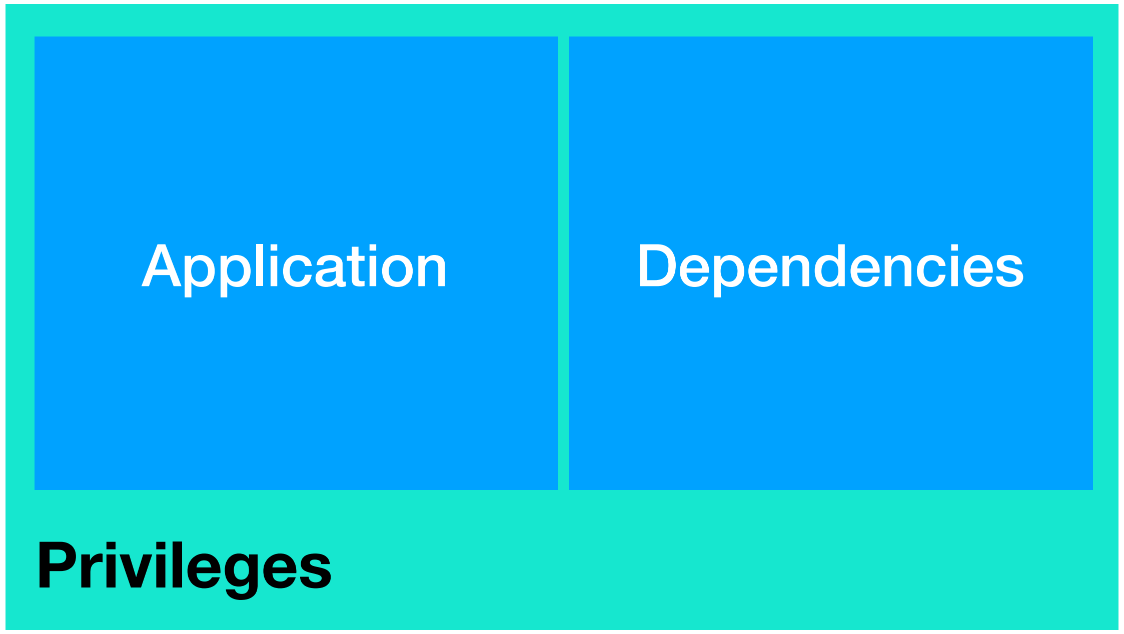
Bad: there are more problems.

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

# What is Privilege Separation? (privsep)

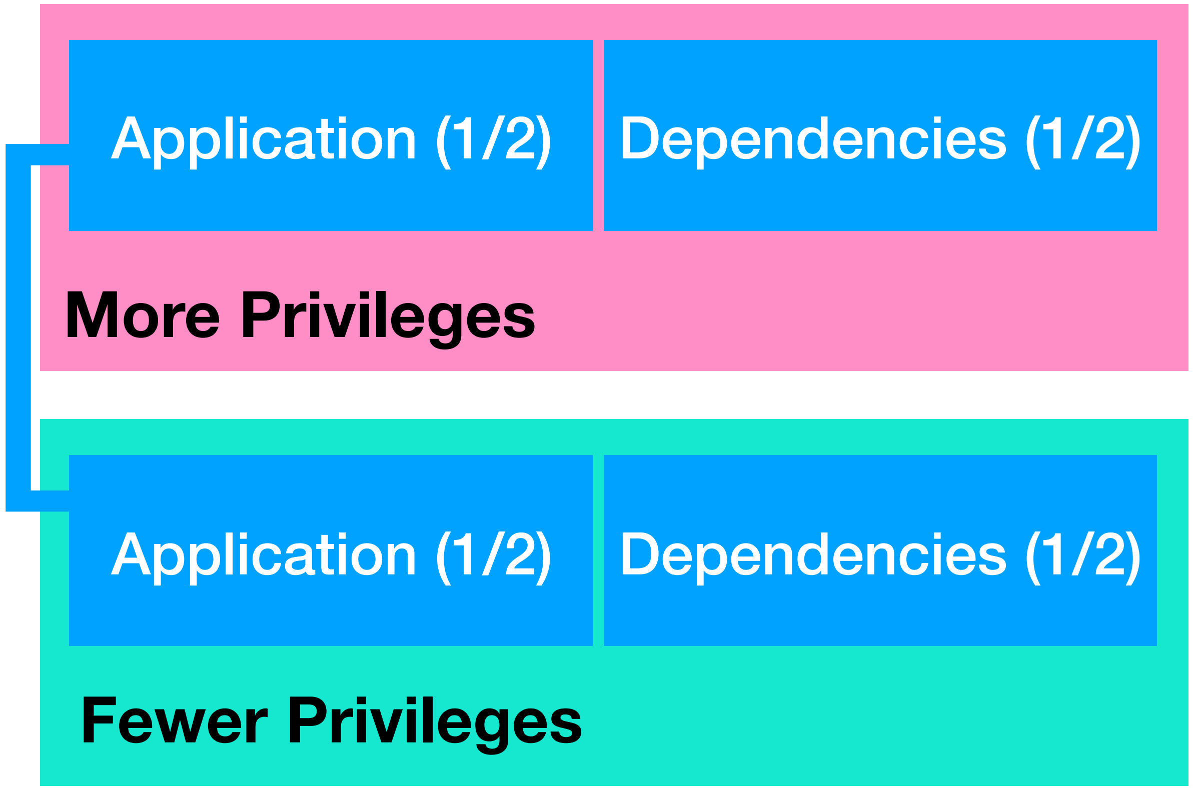


# What is Privilege Separation? (privsep)



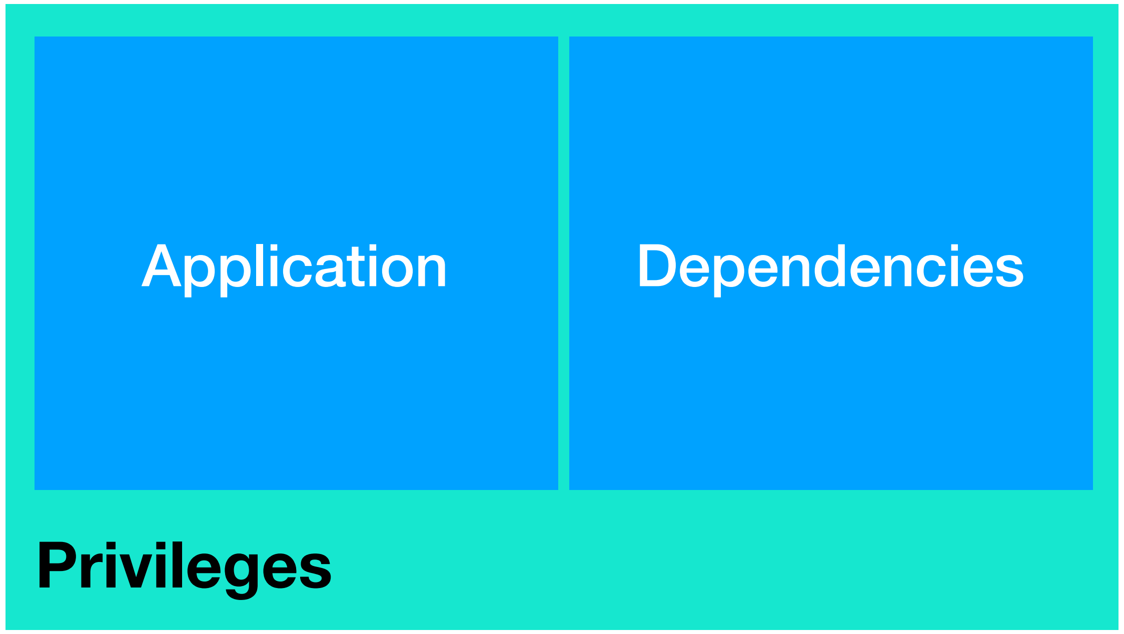
- **Compartmentalize code + data.** Early application: servers: SMTP, SSH.
- Monolithic application → Concurrent set of cooperating programs.
  - Monolithic application: often common privileges throughout.
  - **Distributed system**: granularity of privilege allocation.

# Privsep



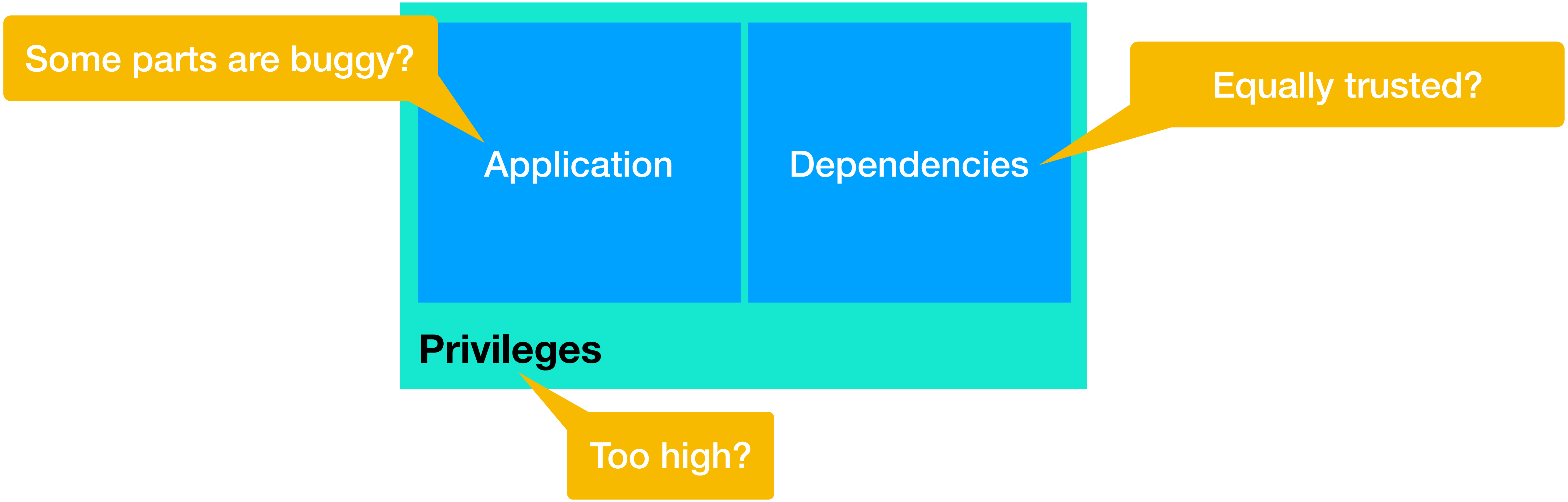
- Main benefit: **vulnerability containment.**
  Best case: if a vulnerability is exploitable, then fewer privileges can be abused.

# Privsep



- **Implementing** privsep: usually a lot of work. Changing software without introducing bugs.

# Privsep

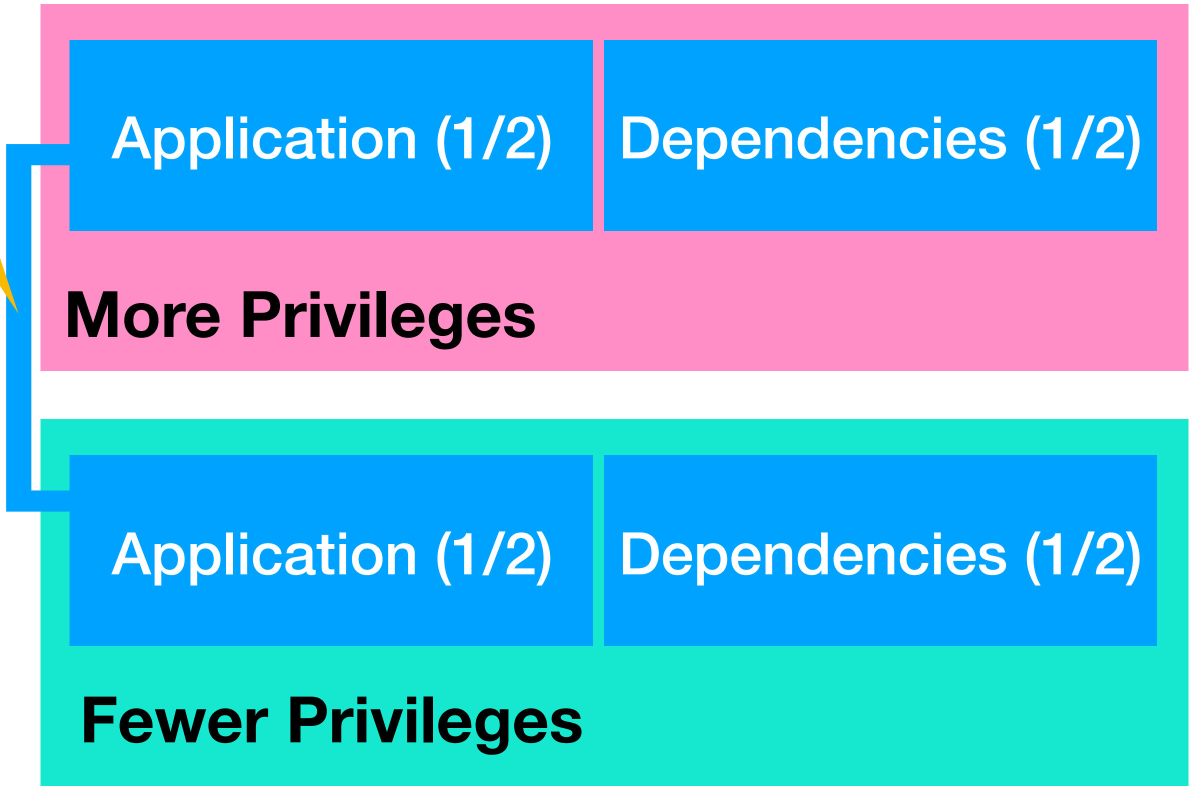


- **Implementing** privsep: usually a lot of work. Changing software without introducing bugs.

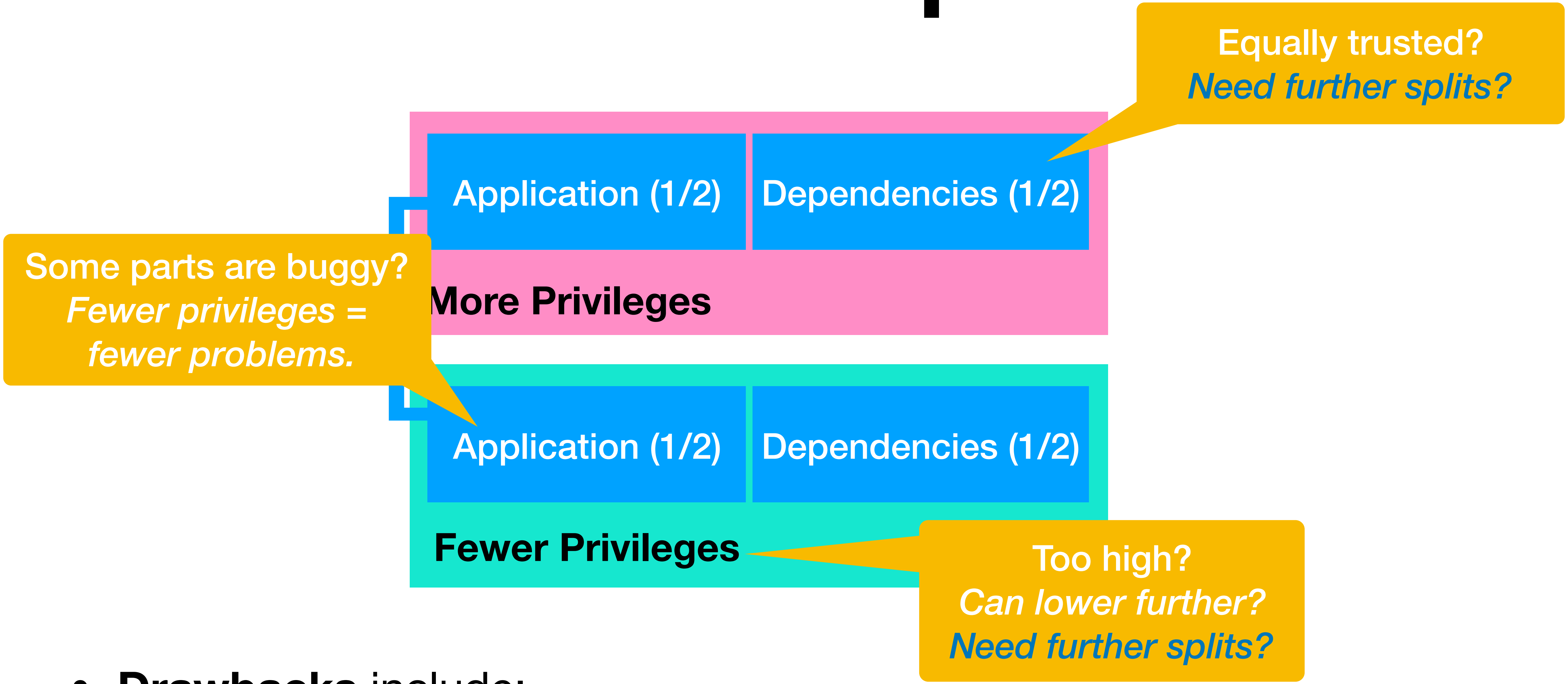
- There are many **decision** to take (and retake later) wrt what+how to separate (see yellow bubbles above).

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

# Privsep



# Privsep

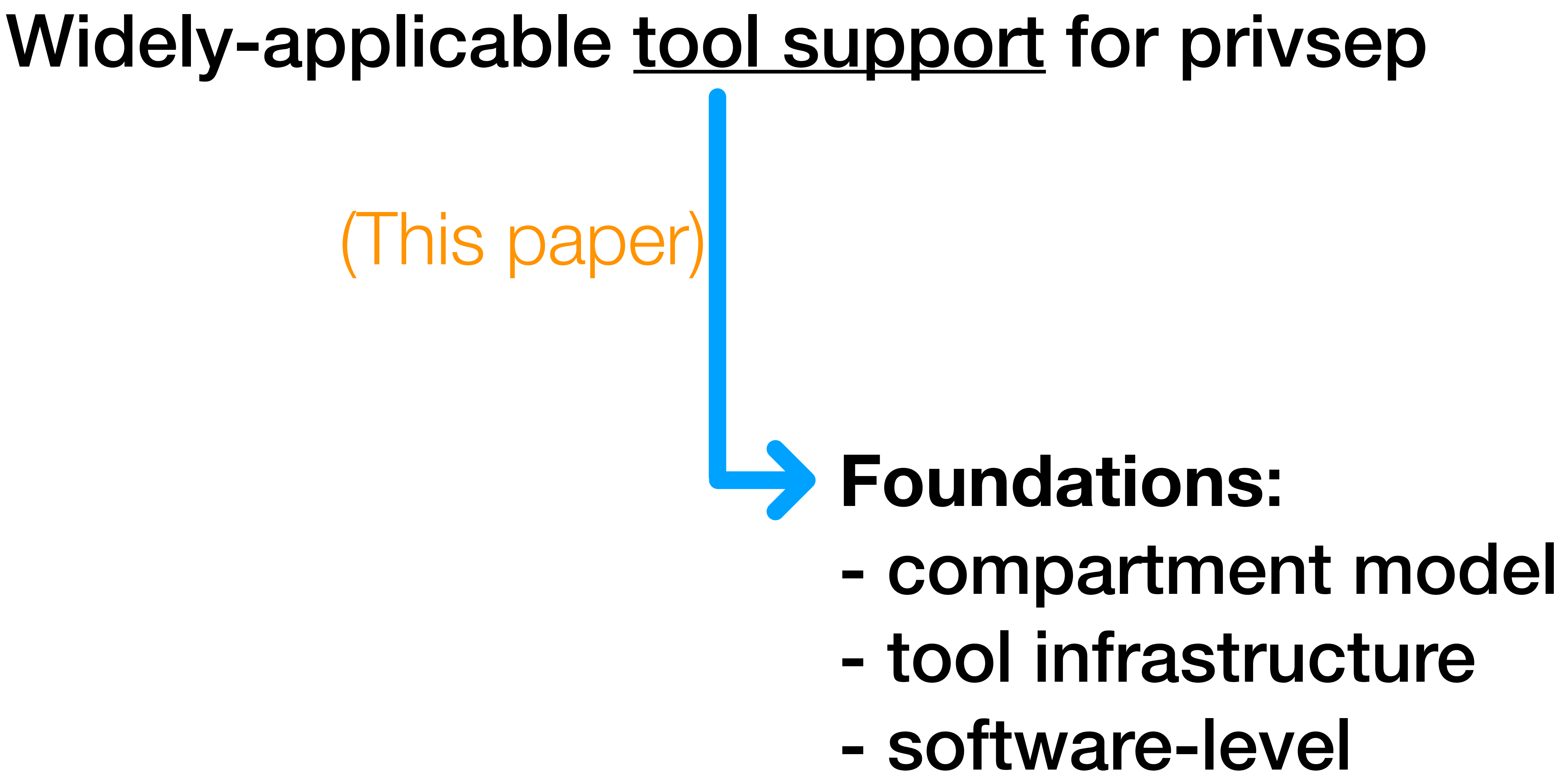


- **Drawbacks** include:
  Inertia wrt **splitting software**, introduction of **new failure modes** (hello distributed systems), performance **overhead**, inertia wrt **maintainability and portability** (e.g., if use hardware enforcement).

# (Longstanding) Research Goal

Widely-applicable tool support for privsep

# (Longstanding) Research Goal



Widely-applicable tool support for privsep

(This paper)

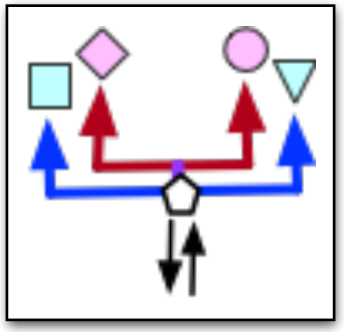
**Foundations:**

- compartment model
- tool infrastructure
- software-level

# (Longstanding) Research Goal

Widely-applicable tool support for privsep

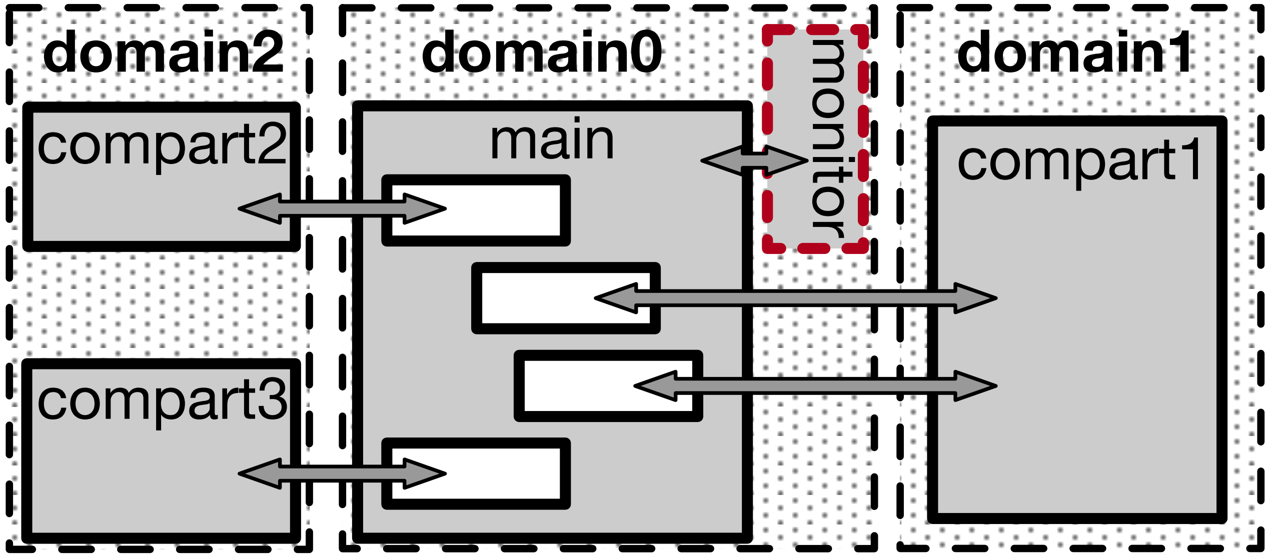
(This paper)



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

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

# Compartment Model



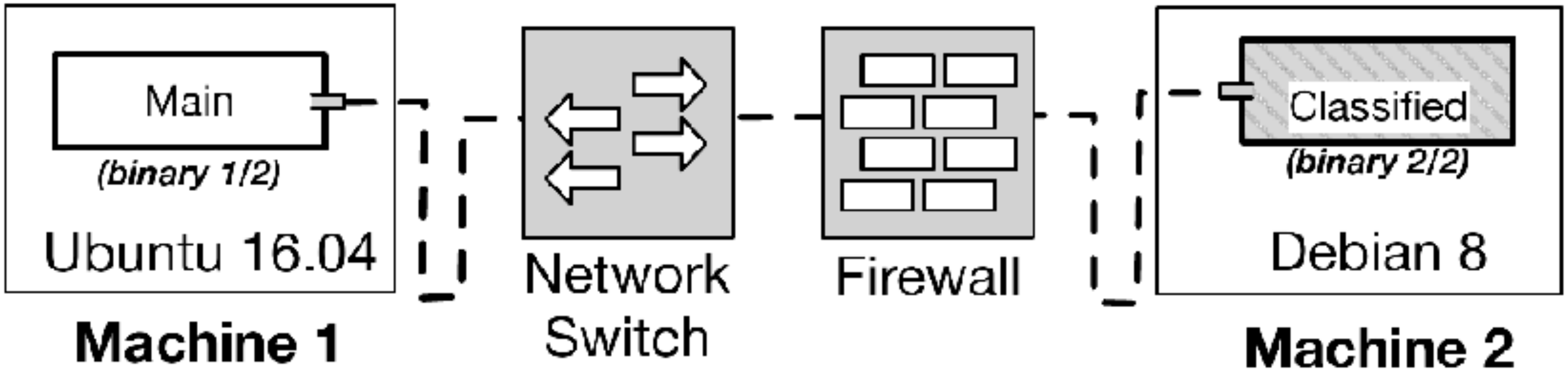
- Organization:
  **Domain**: Shared memory/handles/resources across compartments
  **Compartments**: Sharing across segments.
  **Segments**: code + data.

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

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

- Generalization and Tooling
  vs Flexibility:
  General but restrictive

# Example of what's enabled

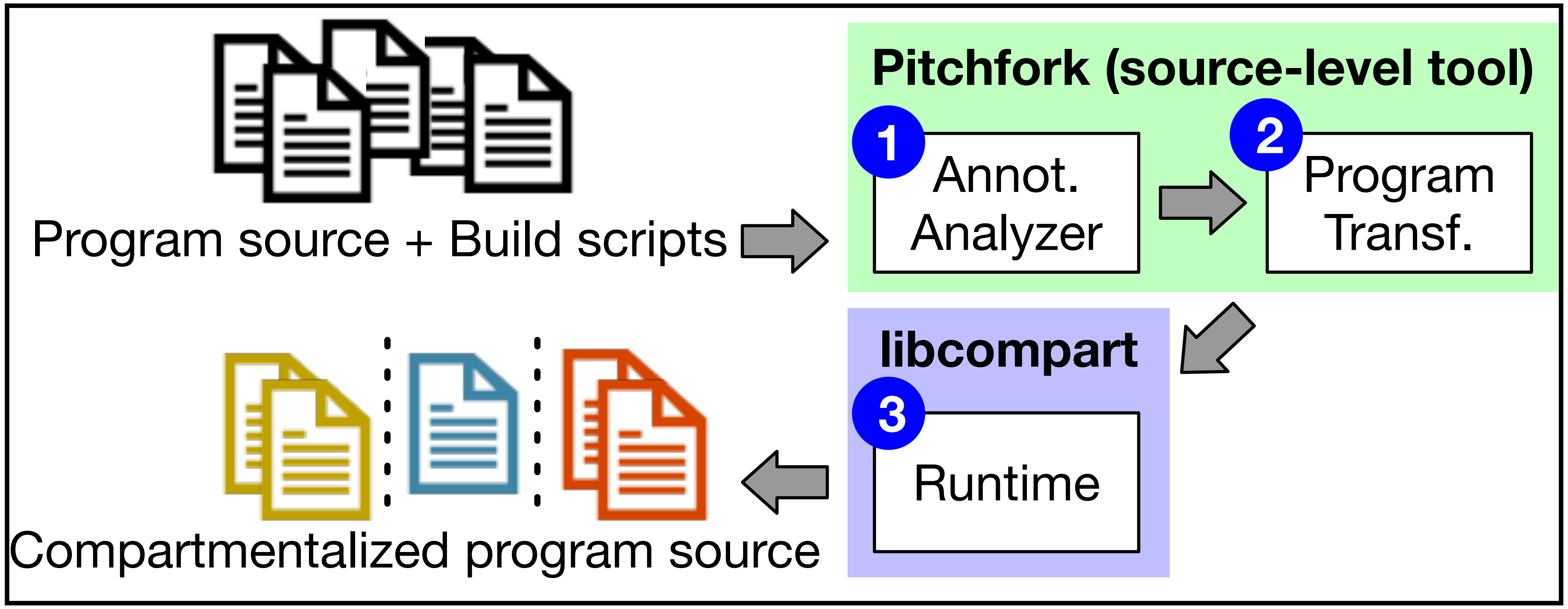


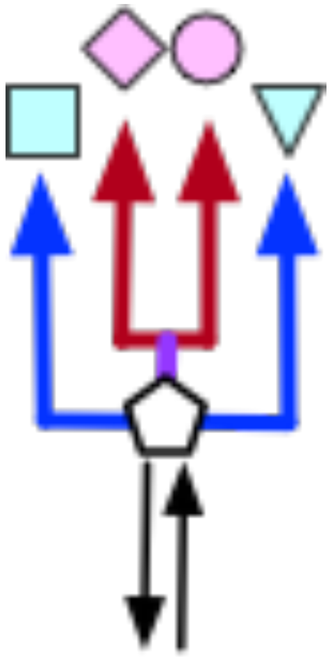
- Organization:
  **Domain**: one on each machine
  **Compartments**: one in each domain.
  **Segments**: 2 in Classified, 1 in Main.

- Communication channel over TCP.

- Machine and network-level policy+enforcement.

# Pitchfork

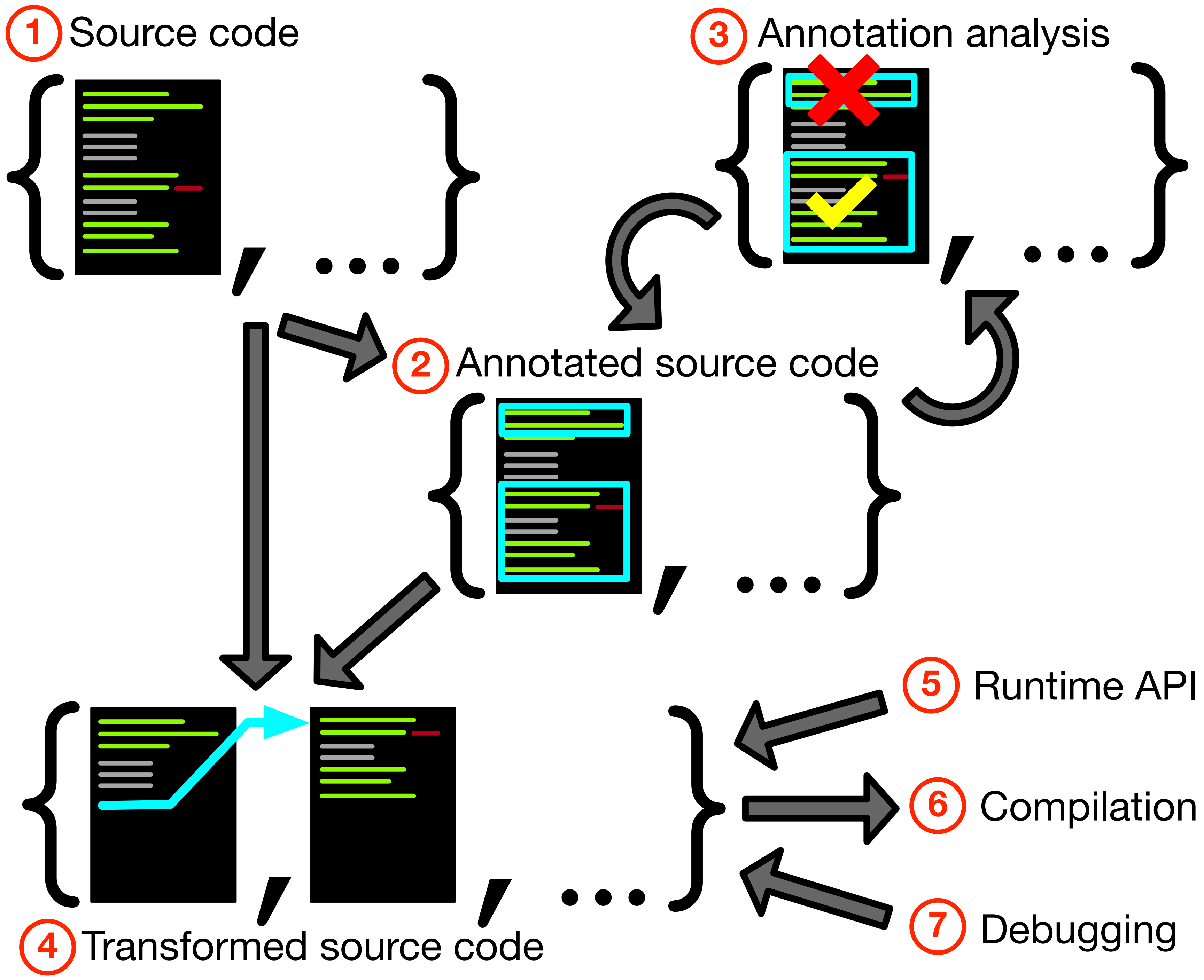

Program source + Build scripts
Pitchfork (source-level tool)
1
Annot. Analyzer
2
Program Transf.
libcompart
3
Runtime
Compartmentalized program source



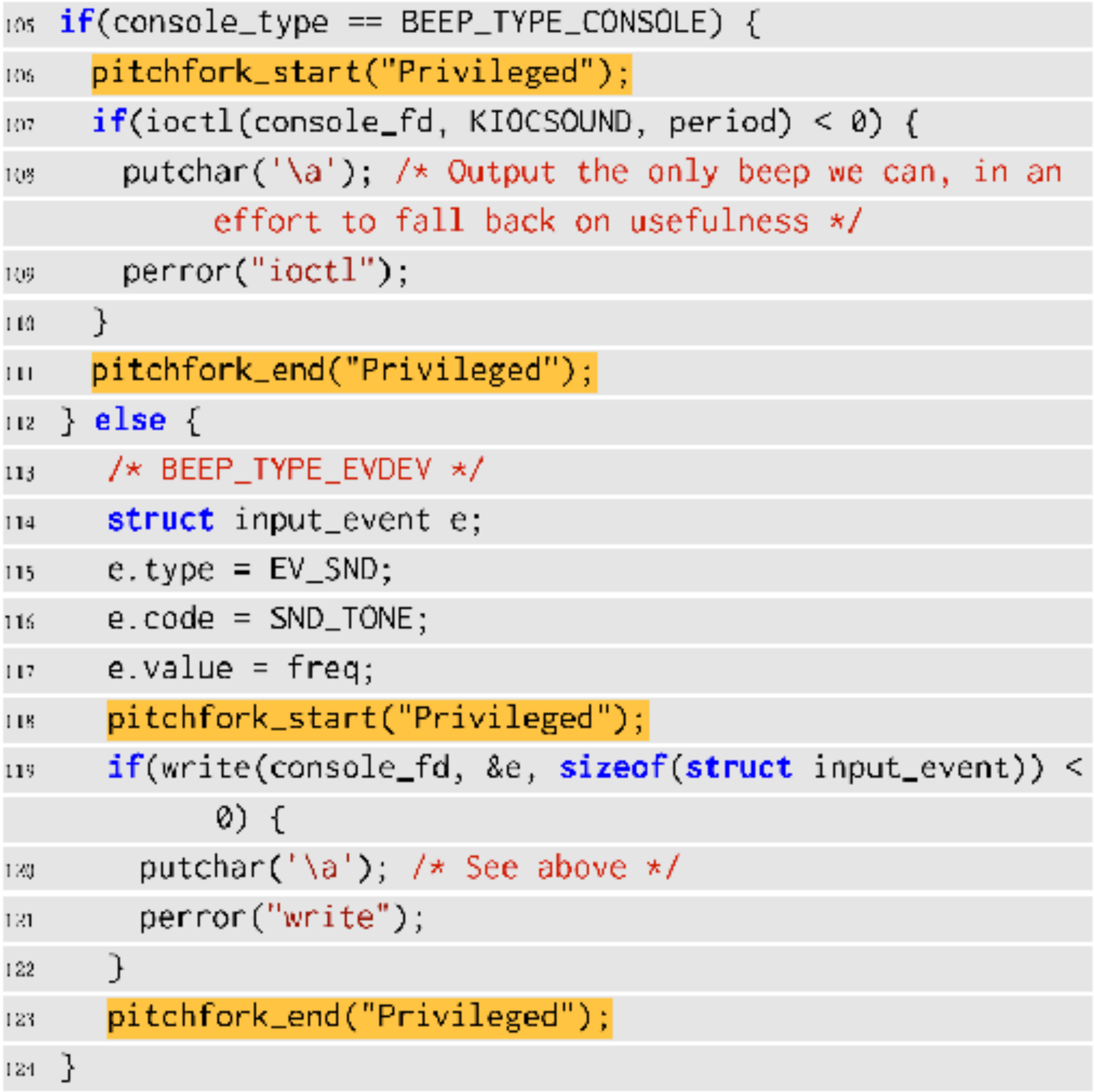
The **system** has two components based on a **model**:

- Pitchfork 1 2
- libcompart 3

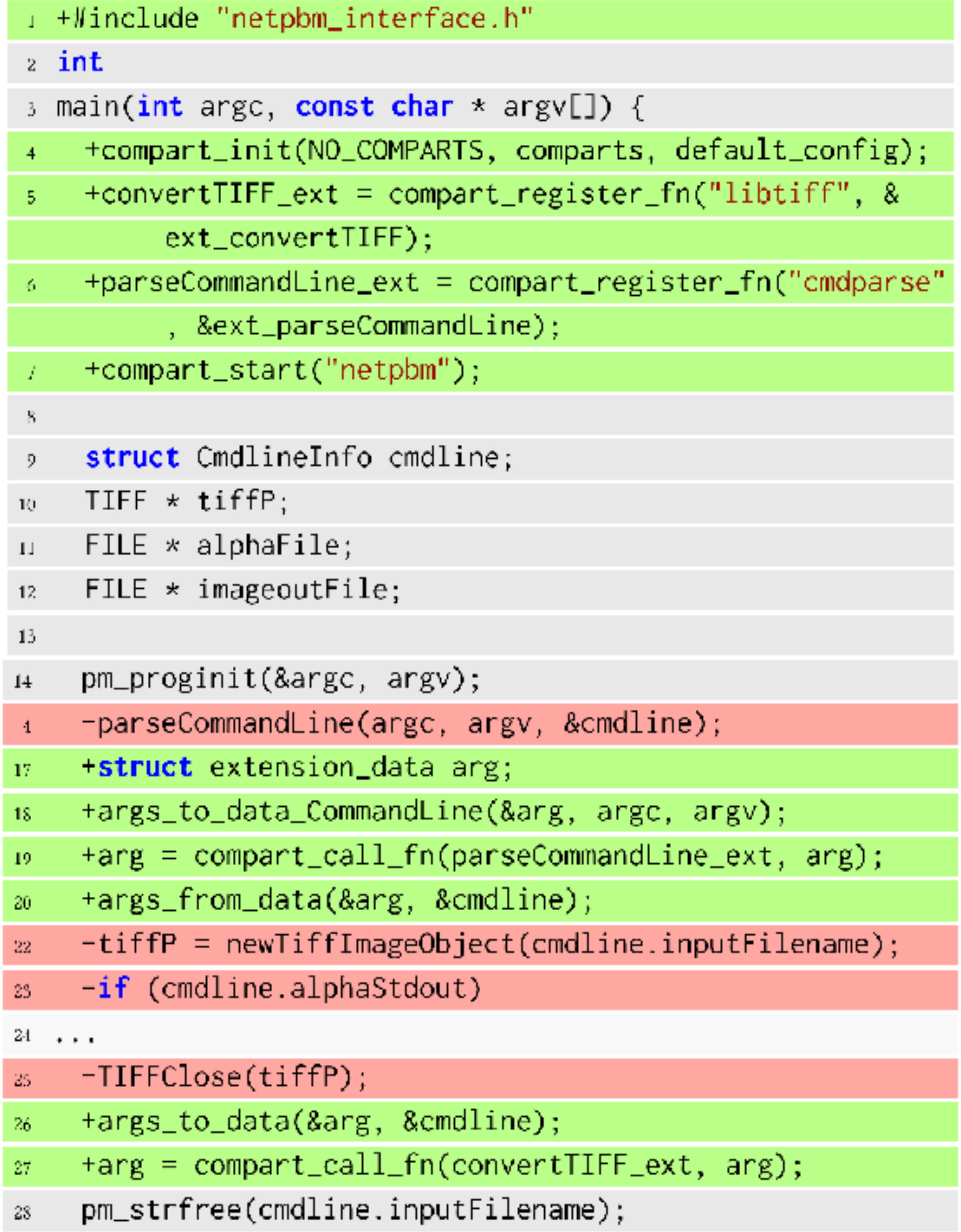
# Pitchfork


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

# Pitchfork



```c
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");
}
```

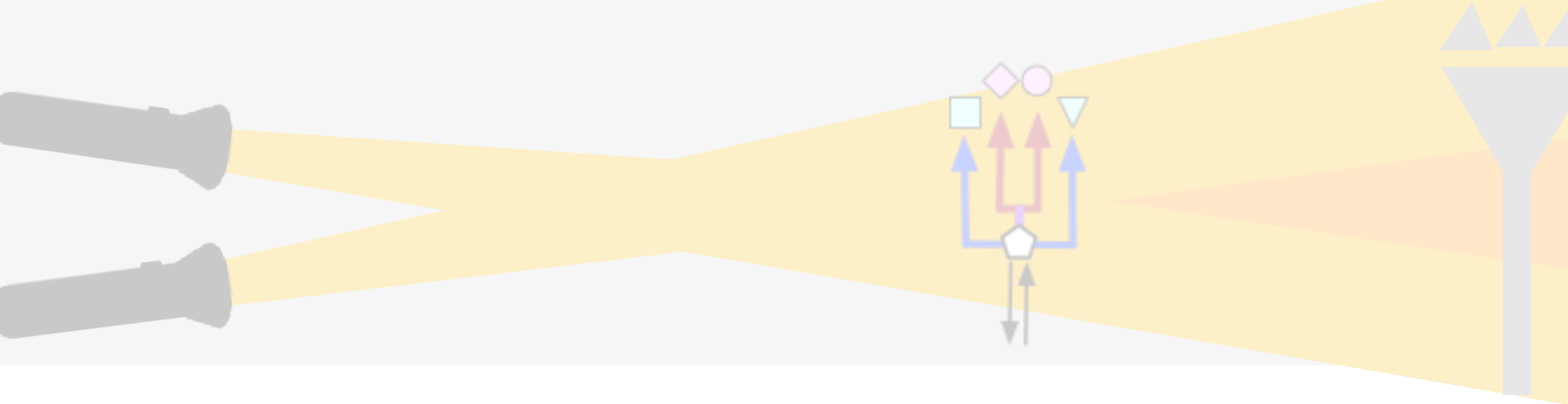


```
+#include "netpbm_interface.h"
int
main(int argc, const char * argv[]) {
  +compart_init(NO_COMPARTS, comparts, default_config);
  +convertTIFF_ext = compart_register_fn("libtiff", &
        ext_convertTIFF);
  +parseCommandLine_ext = compart_register_fn("cmdparse"
        , &ext_parseCommandLine);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

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

  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);
```

# libcompart



# Torches on Pitchfork: Multi-feature Evaluation of a Security-oriented Programming Toolchain

Nik Sultana

*Illinois Institute of Technology*

Learning from Authoritative Security Experiment Results (LASER) 2022

# Food for thought

- **How to identify+scope the security problem?**
- **How to show the problem begin solved?**
  Can this scale with size, complexity and variety of problem instances? (programs)
- **How to understand newly-introduced problems?**

# Food for thought

- Evaluation goals
- Evaluation process
- Challenges:
  - Skills + Time needed to reproduce exploit. Scaling the eval.
  - Generalizability of analysis + transformation.
  - User study.
  - Reasoning about incomplete info — likelihood of introducing bugs.

**Plans for post-workshop: above + more software analysis**

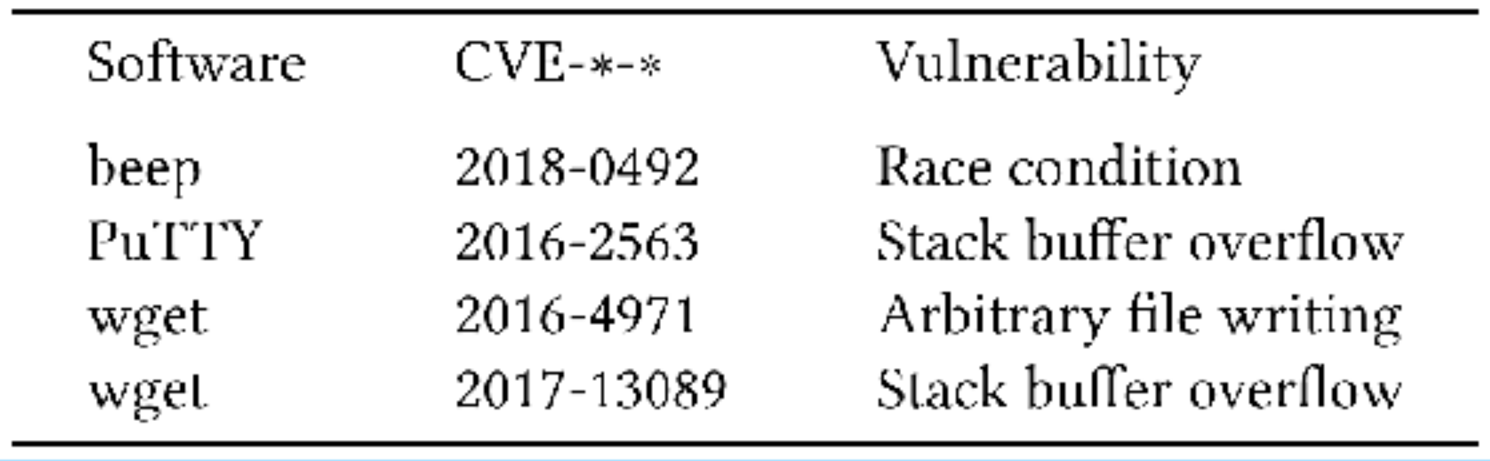
# Evaluation

**(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

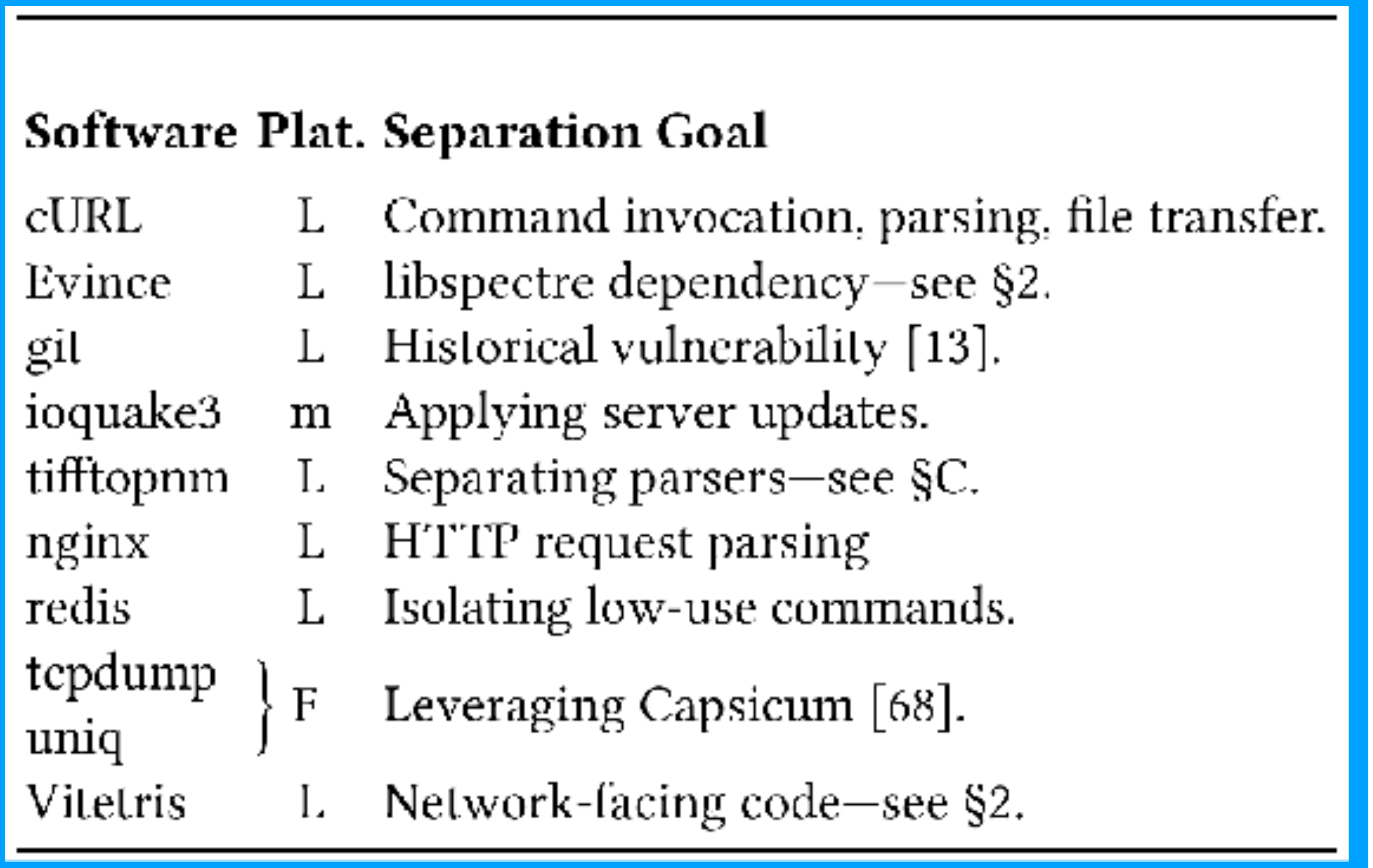


| Software | CVE-*-* | Vulnerability |
|---|---|---|
| beep | 2018-0492 | Race condition |
| PuTTY | 2016-2563 | Stack buffer overflow |
| wget | 2016-4971 | Arbitrary file writing |
| wget | 2017-13089 | Stack buffer overflow |

    - Heuristics
- Overhead: running time, memory, binary size.

# Evaluation

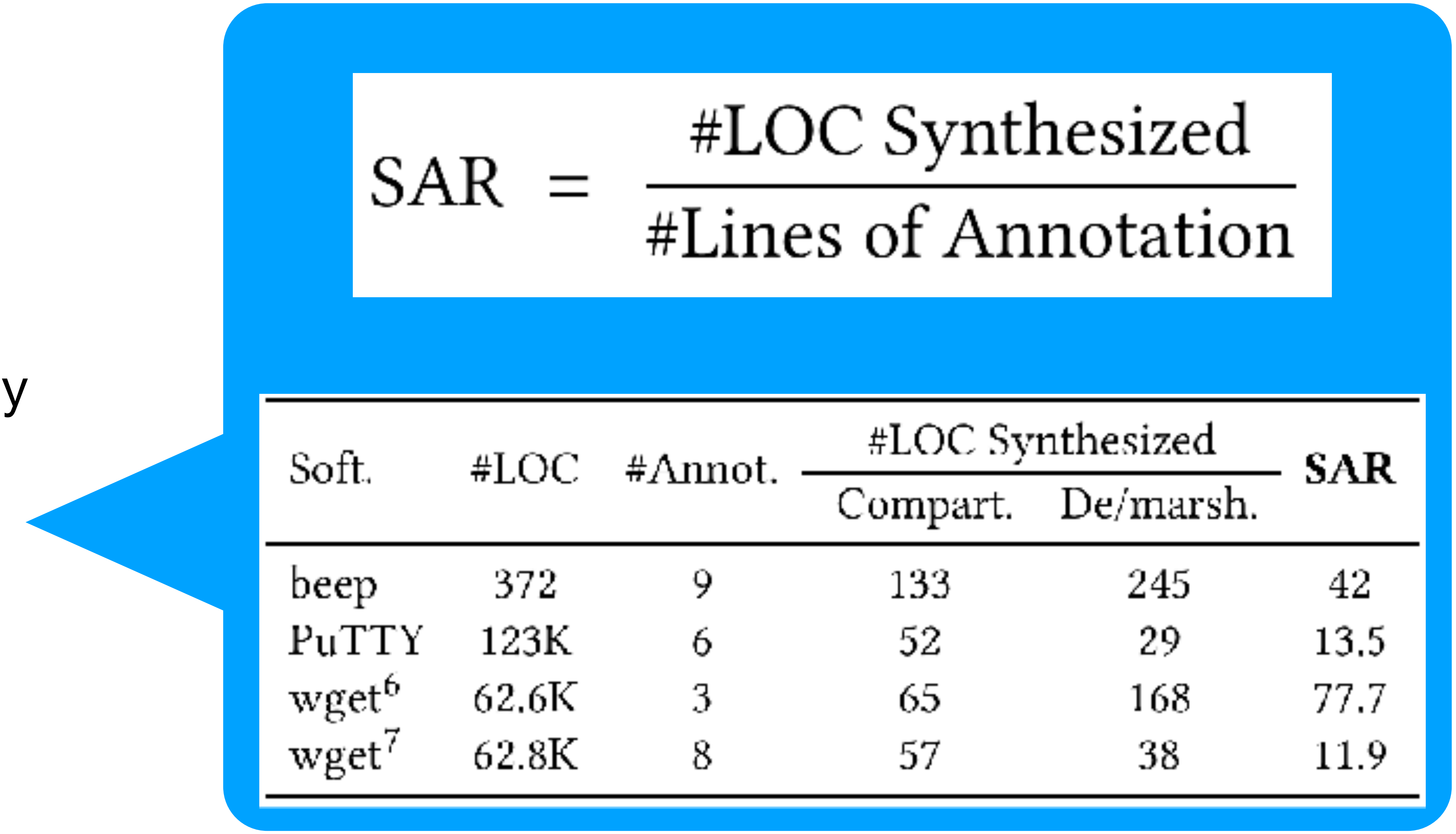
- Applicability
  - Examples
  - Maintainability
  - Convenience
- Security
  - Known CVEs
  - Heuristics
- Overhead: running time, memory, binary size.



| 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 | F | Leveraging Capsicum [68]. |
| uniq | | |
| Vitetris | L | Network-facing code—see §2. |

# Evaluation

- Applicability
  - Examples
  - Maintainability
  - Convenience



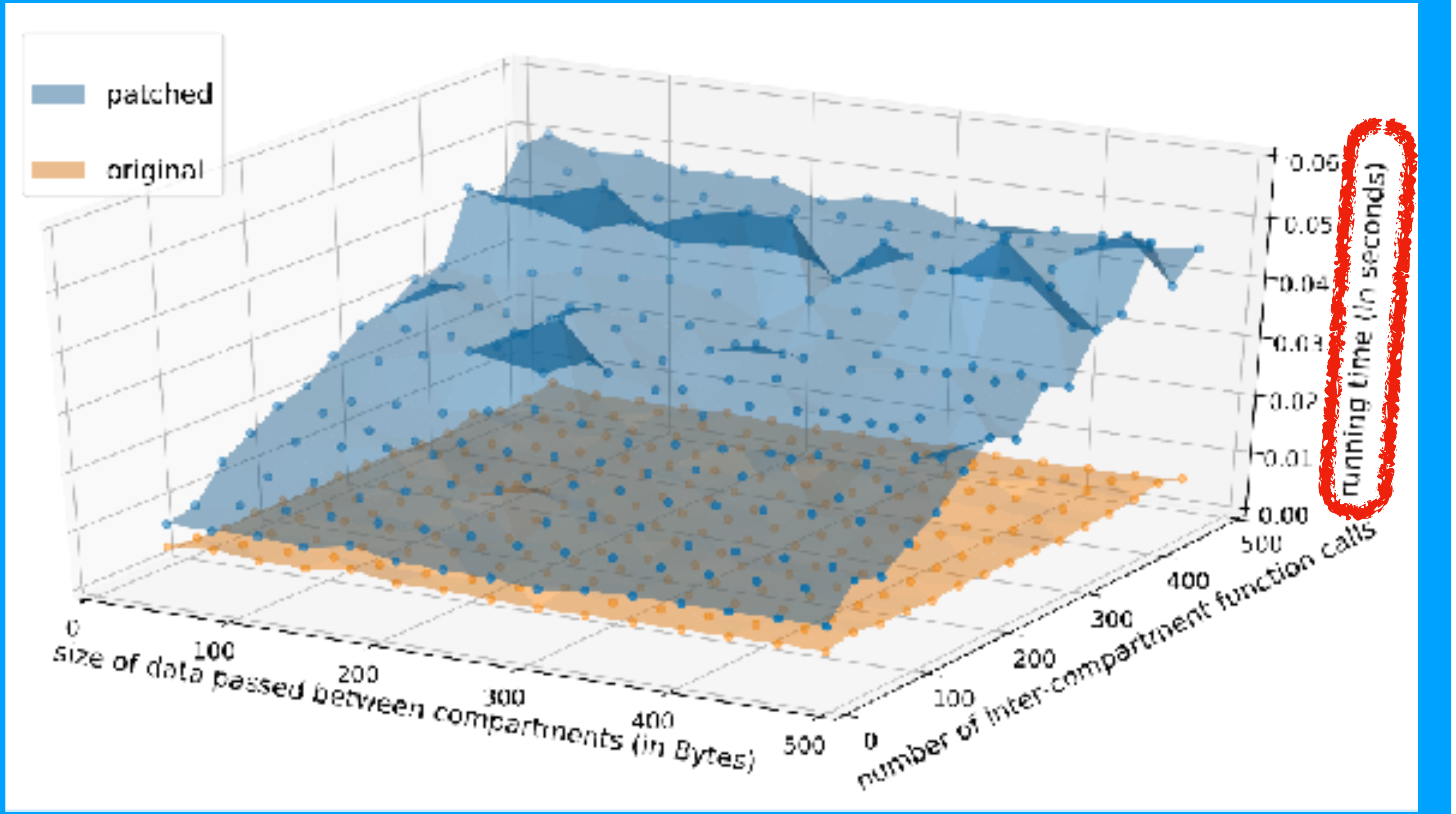
$$\text{SAR} = \frac{\text{\#LOC Synthesized}}{\text{\#Lines of Annotation}}$$

| 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 |

- Security
  - Known CVEs
  - Heuristics
- Overhead: running time, memory, binary size.

# Evaluation

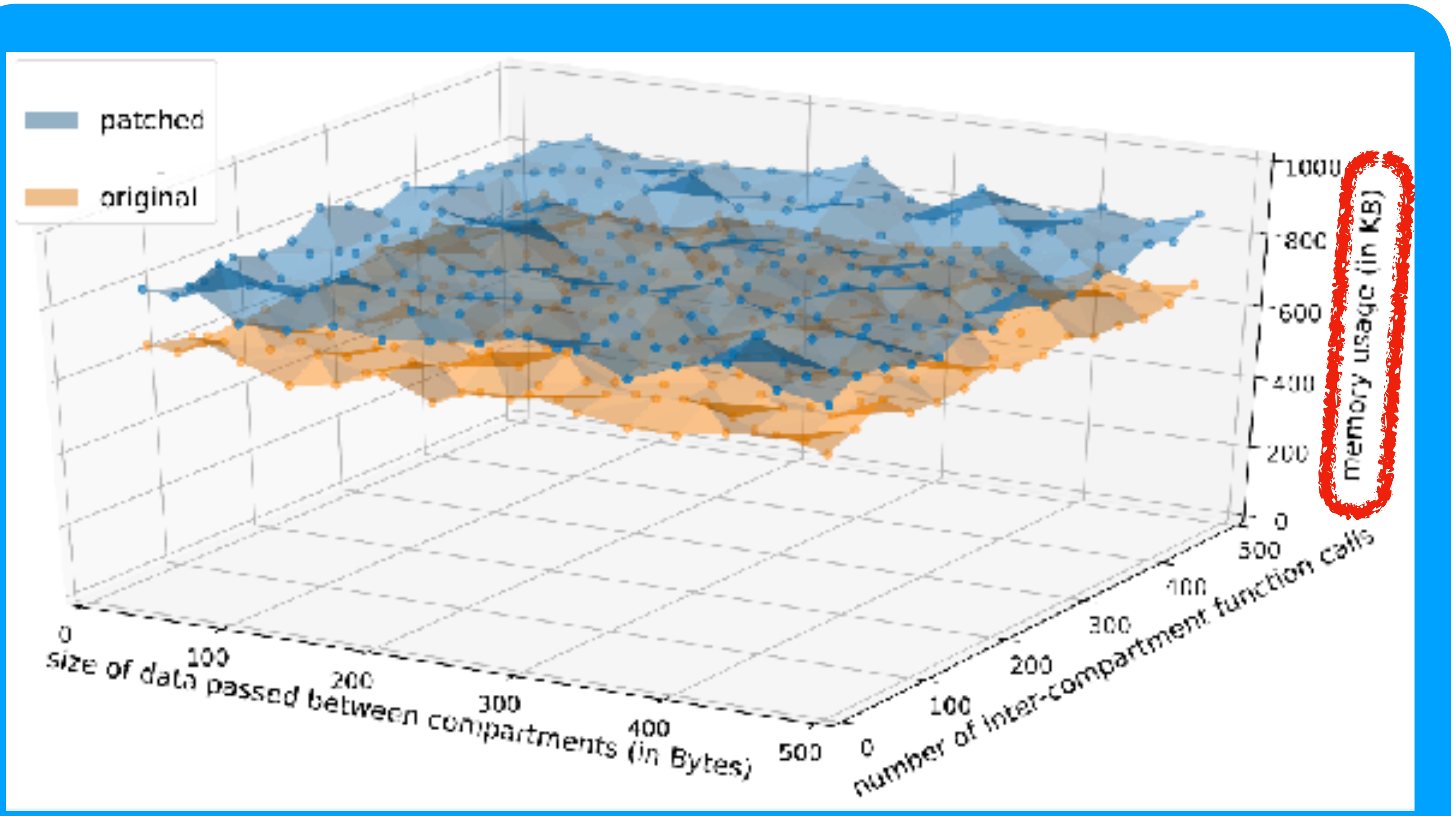
- Applicability
  - Examples
  - Maintainability
  - Convenience
- Security
  - Known CVEs
  - Heuristics
- Overhead: running time, memory, binary size.



**Different compartments in same domain.**

# Evaluation

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



**Memory stability wrt quantity & size of data exchange.**

# Food for thought

- **How to identify+scope the security problem?**

  Existing literature on privsep.

  Non-specialized, commodity hardware & kernel.
  “Realism”.

  CVEs in third-party, widely-used programs.
  (CVEs that allow code injection or exfiltration).

  Written in C, “warts and all”.
  Unmodified compiler toolchains.

Mul y-

C

# Food for thought

- **How to show the problem being solved?**

  Reproduce CVEs — not all attempts were productive for this research (discussed in an appendix).
  Classify CVEs?

  Trial and error. Starting with simple/short programs. Recreated problem from literature.

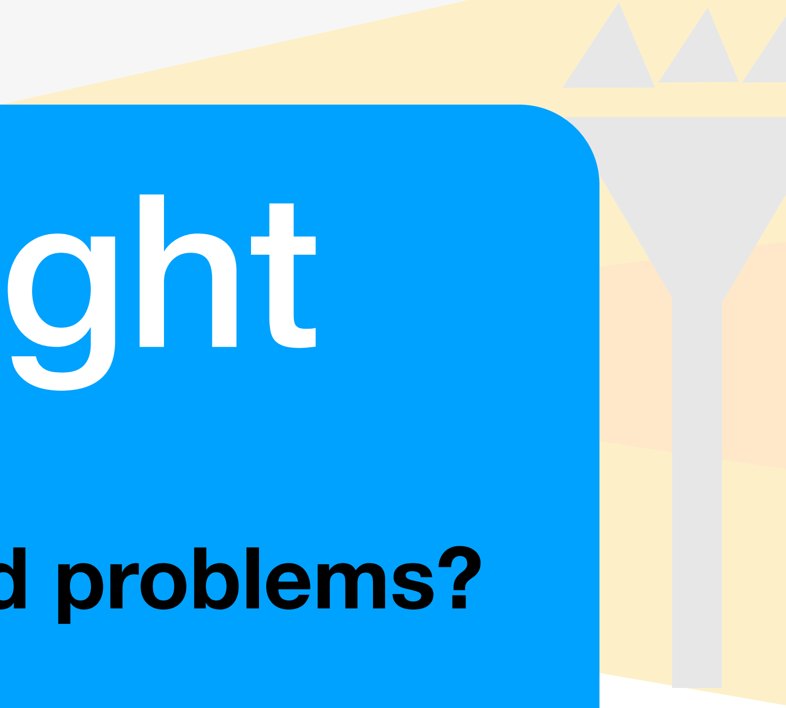
  **Thanks to community**

  Work up to more types of software.
  Generality analysis.

  **Different experiment methodologies: security, performance, applicability.**

Mul y-

C



# Food for thought

- **How to understand newly-introduced problems?**

  Very hard to prove a negative.

  Does this ultimately require verification?

  Practical under approximation : tests still run, usage still works (so no newly-introduced problems wrt those instances), but no airtight evidence that no problems have been introduced.

  **Other practical issues: build scripts, portability and complexity of the resulting system.**

Mul y-

c

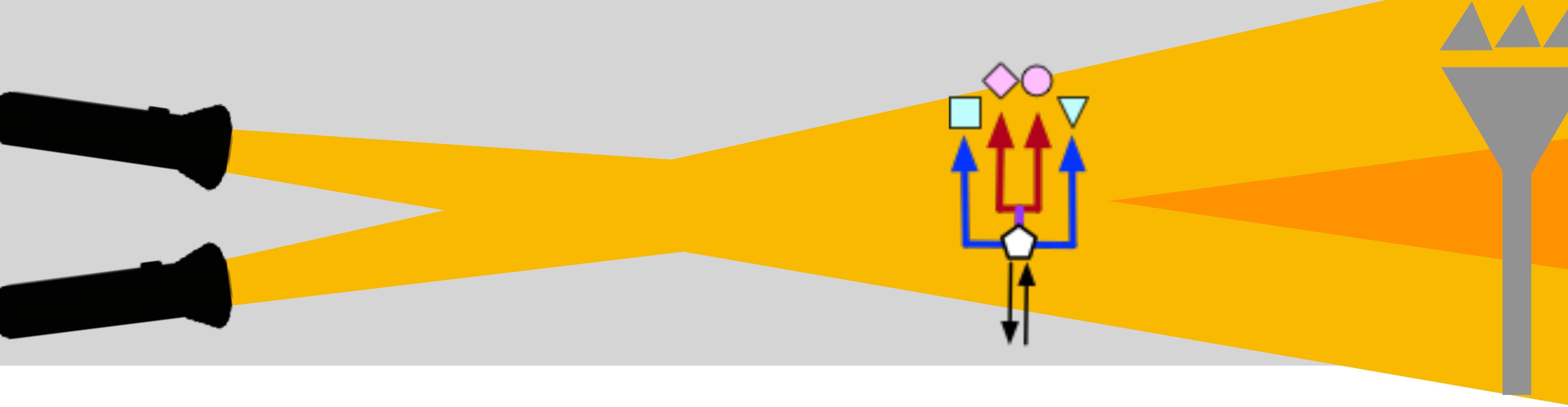
# Things that didn’t work

- Some partitionings: e.g.,
  - **CVE-2015-6565** (openssh) involved a bad permissioning decision. In general, can partitioning mitigate against bad configuration decisions? Doesn’t partitioning add another layer of configuration?
  - **CVE-2018-10933** (libssh) involved flawed state machine.
- Eval environment diversity: leads to complexity in the paper. Better to have a single environment for all use cases?
- Test setup inertia wrt some use-cases (library versioning) — this would have been easy to overcome, but at the cost of a little more engineering and fiddling.
- Conceptual/algebraic approach to describe partitions,
  too simplistic.

# Food for thought

- Evaluation goals
- Evaluation process
- Challenges:
  - Skills + Time needed to reproduce exploit. Scaling the eval.
  - Generalizability of analysis + transformation.
  - User study. **How to quantify benefit of using a specific defense?**
  - Reasoning about incomplete info — likelihood of introducing bugs.

**Plans for post-workshop: above + more software analysis**



# Torches on Pitchfork:

# Multi-feature Evaluation of a Security-oriented Programming Toolchain

Nik Sultana

*Illinois Institute of Technology*

Learning from Authoritative Security Experiment Results (LASER) 2022