When is logging sufficient?

Tracking event causality for improved forensic analysis and correlation

Johannes Olegård

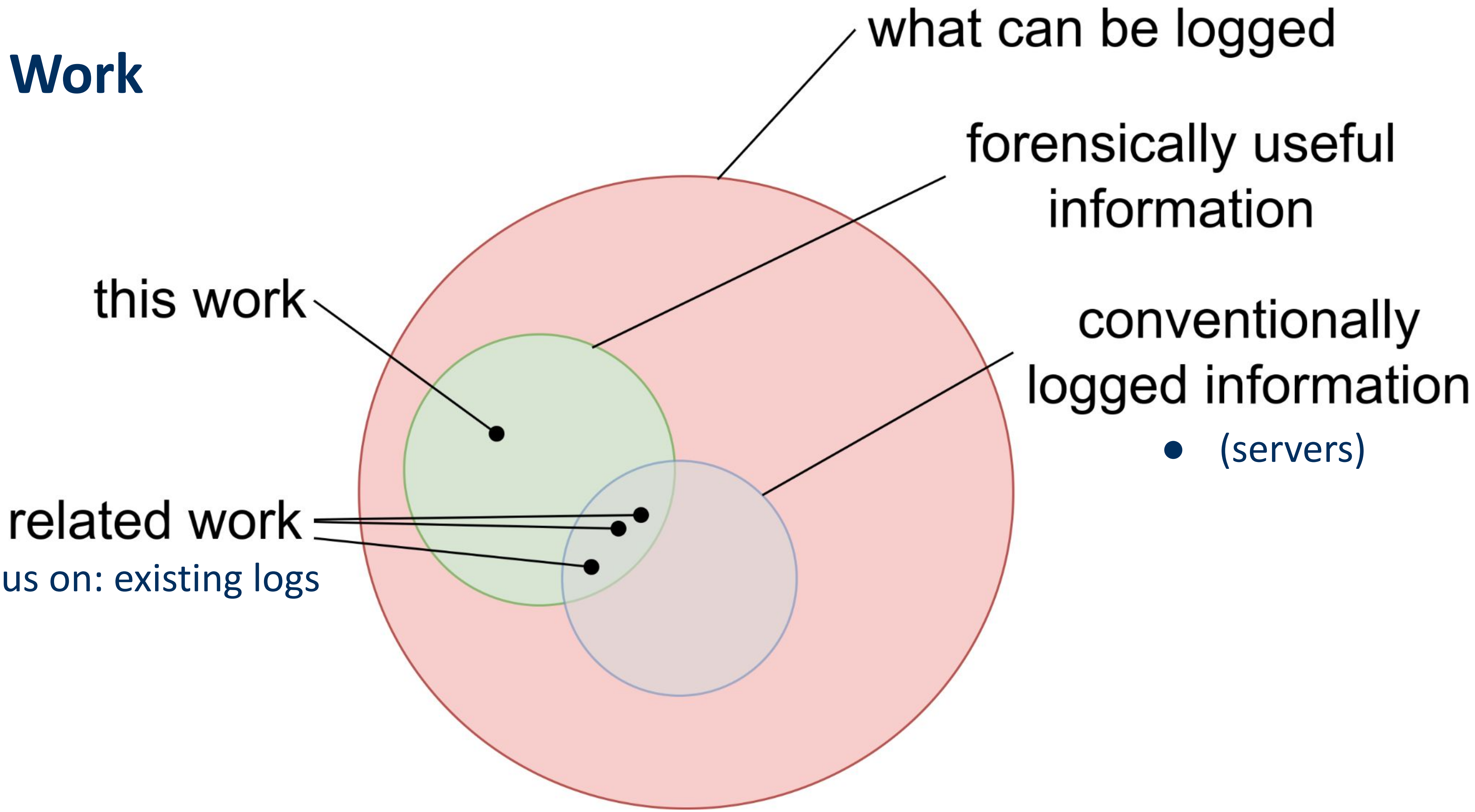
PhD Student

Stockholm University

“Insufficient” Logging

- Security Standards say: “Please do *sufficient* logging!” (e.g. ISO27k)
- Sure, but how!?
  - Developers don’t know.
  - Academics don’t know.
- which leads to logs with:
  - too little/much information
  - correlation difficulties

Related Work


what can be logged
forensically useful information
this work
conventionally logged information
related work

- (servers)

- Focus on: existing logs

Related Work: Forensic Readiness

- Forensic readiness of a **System**
  - Design it to be easier to investigate (reduce uncertainty)
  - Problem: existing research is too high-level
    - req.-engineering, threat modeling, etc.
- Proposed new definition:

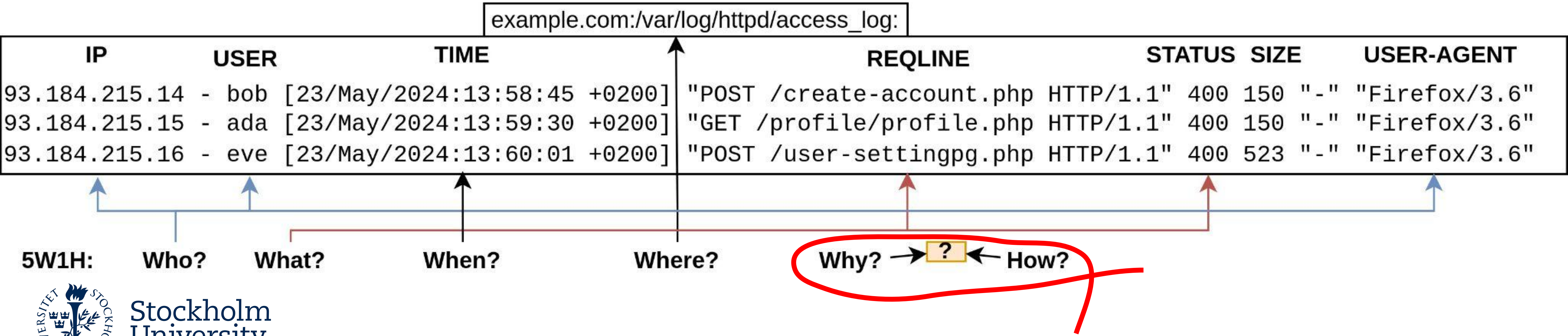
"A System is forensically ready with respect to a set of forensic questions."

The short answer

- “Logging is sufficient when it can sufficiently answer your (future) forensic questions”
- But then: what forensic questions do we typically ask?

5W1H

- a log entries provides “micro-answers” to 5W1H
  - (not always good answers)
- However: typically no answer to Why and How
  - what would these answers look like?



example.com:/var/log/httpd/access_log:

```
IP              USER         TIME                              REQLINE                         STATUS  SIZE      USER-AGENT
93.184.215.14 - bob [23/May/2024:13:58:45 +0200] "POST /create-account.php HTTP/1.1" 400 150 "-" "Firefox/3.6"
93.184.215.15 - ada [23/May/2024:13:59:30 +0200] "GET /profile/profile.php HTTP/1.1" 400 150 "-" "Firefox/3.6"
93.184.215.16 - eve [23/May/2024:13:60:01 +0200] "POST /user-settingpg.php HTTP/1.1" 400 523 "-" "Firefox/3.6"
```

Stockholm University

Correlation

- Log entries typically described message SENT and RECEIVED events.

Application A log:

- Sent a message to B.
- Sent a message to B.
- Sent a message to B.

Application B log:

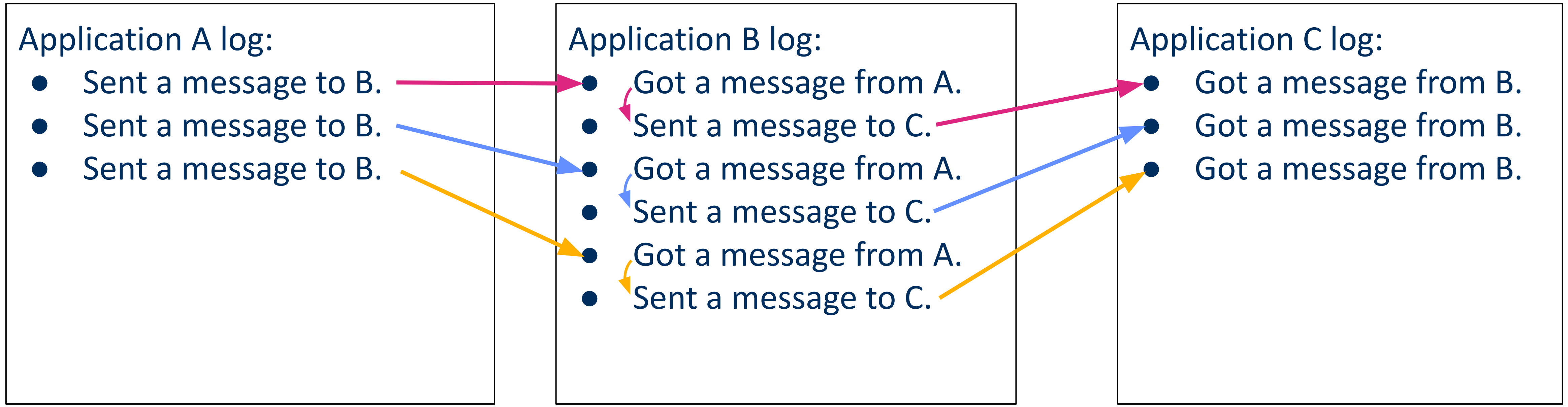
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.

Application C log:

- Got a message from B.
- Got a message from B.
- Got a message from B.

Correlation

- Correlation typically relies on timestamps and guesswork.



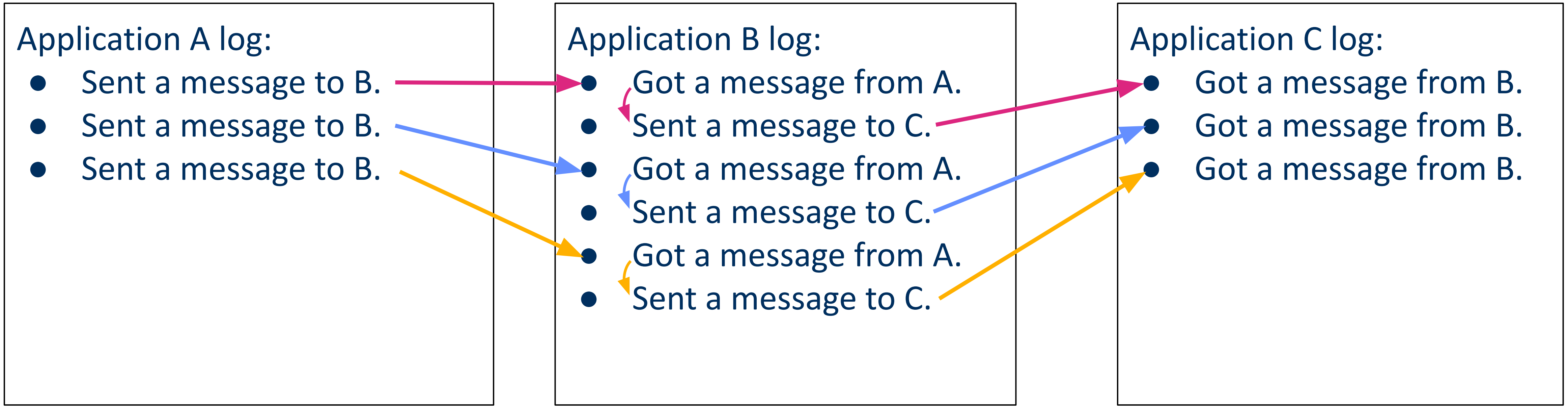
Application A log:
- Sent a message to B.
- Sent a message to B.
- Sent a message to B.

Application B log:
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.

Application C log:
- Got a message from B.
- Got a message from B.
- Got a message from B.

Correlation

- Correlation typically relies on timestamps and guesswork.
  - We could just store “causal information” in the first place!

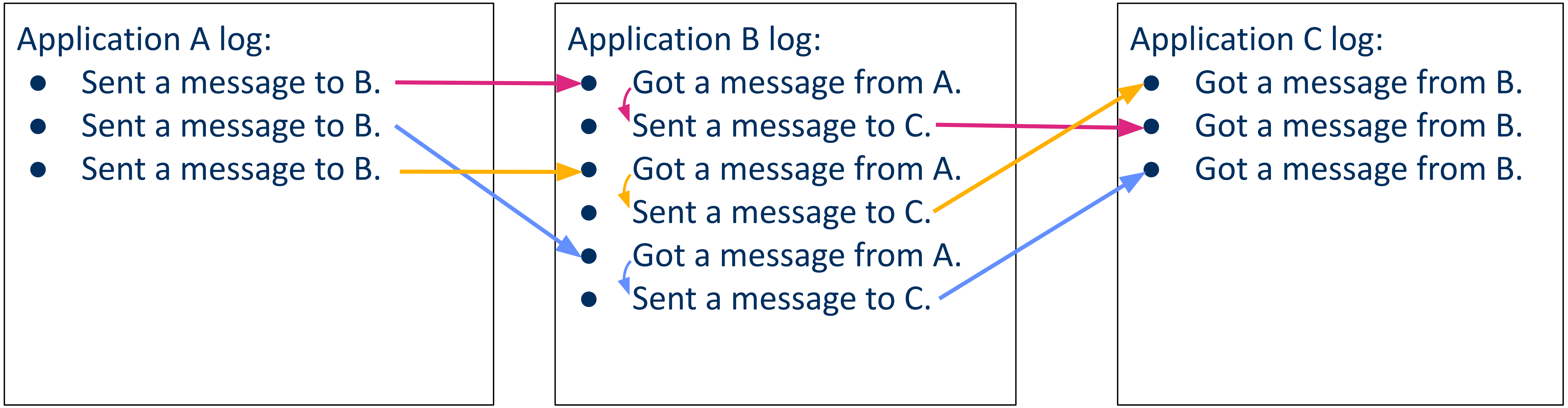

Application A log:
- Sent a message to B.
- Sent a message to B.
- Sent a message to B.

Application B log:
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.

Application C log:
- Got a message from B.
- Got a message from B.
- Got a message from B.

Correlation

- Correlation typically relies on timestamps and guesswork.
  - We could just store “causal information” in the first place!
  - also avoids common pitfalls: out-of-order events,   unsynced clocks



Application A log:
- Sent a message to B.
- Sent a message to B.
- Sent a message to B.

Application B log:
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.
- Got a message from A.
- Sent a message to C.

Application C log:
- Got a message from B.
- Got a message from B.
- Got a message from B.

Main idea of this paper

because it sounds cooler

- Each log entry SHOULD contain:
  - a unique **EventID** (the event's "gretel number")

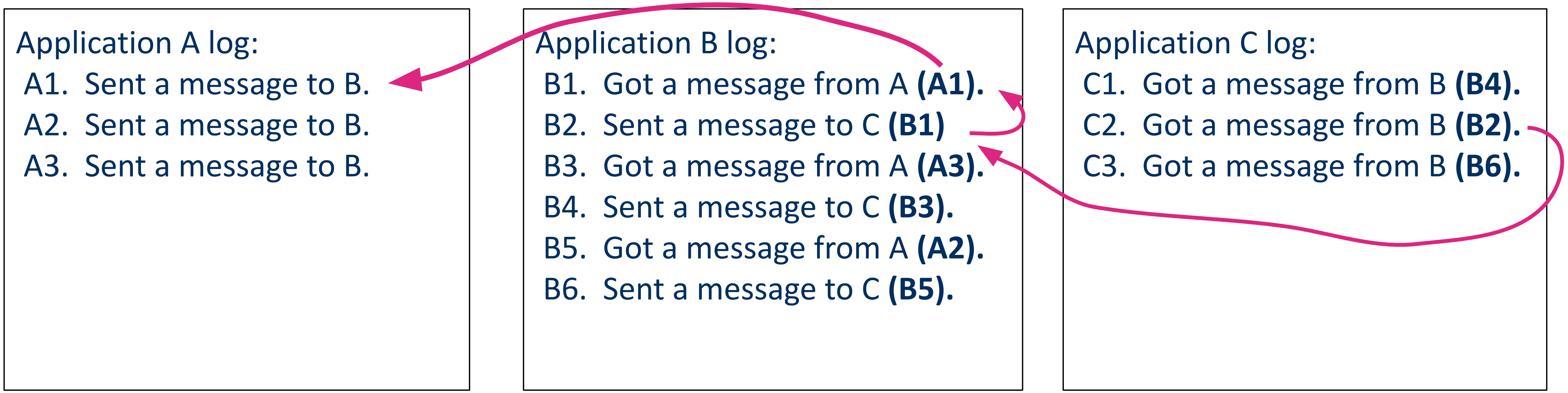
Application A log:
A1. Sent a message to B.
A2. Sent a message to B.
A3. Sent a message to B.

Application B log:
B1. Got a message from A.
B2. Sent a message to C.
B3. Got a message from A.
B4. Sent a message to C.
B5. Got a message from A.
B6. Sent a message to C.

Application C log:
C1. Got a message from B.
C2. Got a message from B.
C3. Got a message from B.

Main idea of this paper

- Each log entry SHOULD contain:
  - a unique **EventID** (the event's "gretel number")
  - at least one **predecessor EventID** ("What event(s) **caused** this one?")



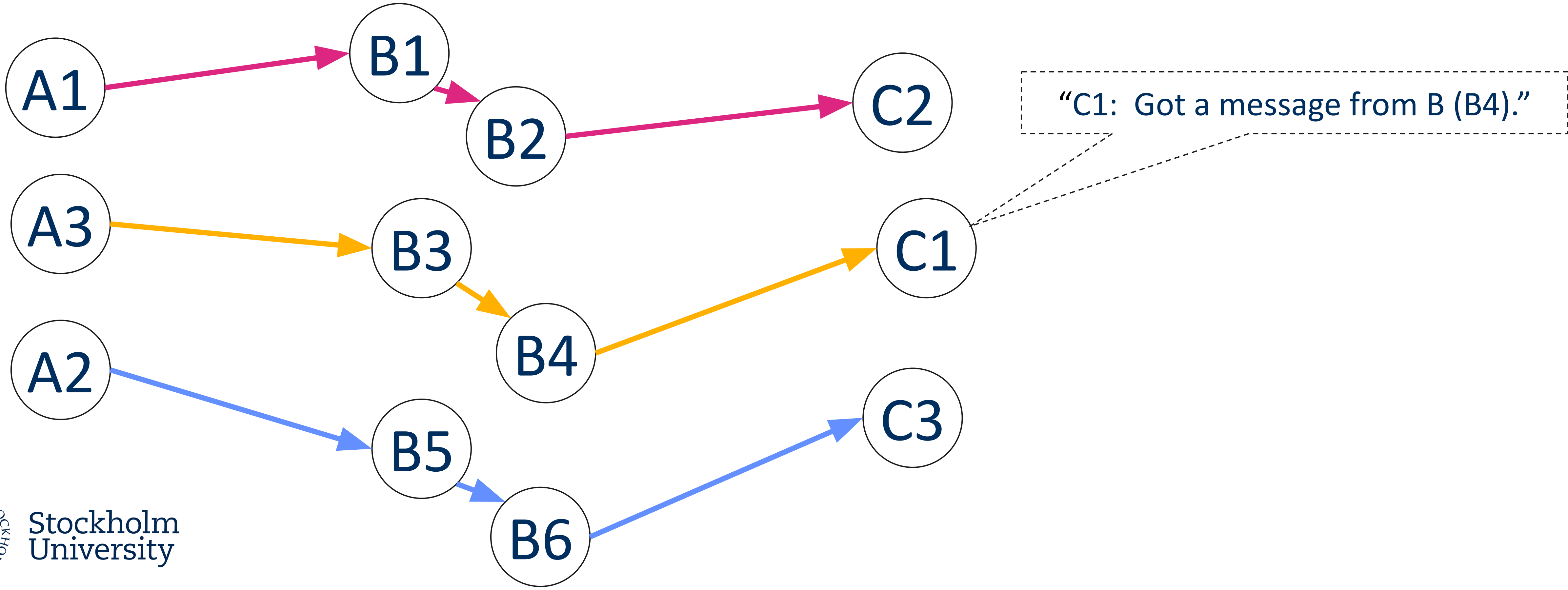
Application A log:
A1. Sent a message to B.
A2. Sent a message to B.
A3. Sent a message to B.

Application B log:
B1. Got a message from A **(A1).**
B2. Sent a message to C **(B1)**
B3. Got a message from A **(A3).**
B4. Sent a message to C **(B3).**
B5. Got a message from A **(A2).**
B6. Sent a message to C **(B5).**

Application C log:
C1. Got a message from B **(B4).**
C2. Got a message from B **(B2).**
C3. Got a message from B **(B6).**

Causal Graph

- (partial order)
- **anti-tampering (bonus):** detect anomalies in the graph



Proof-of-Concept

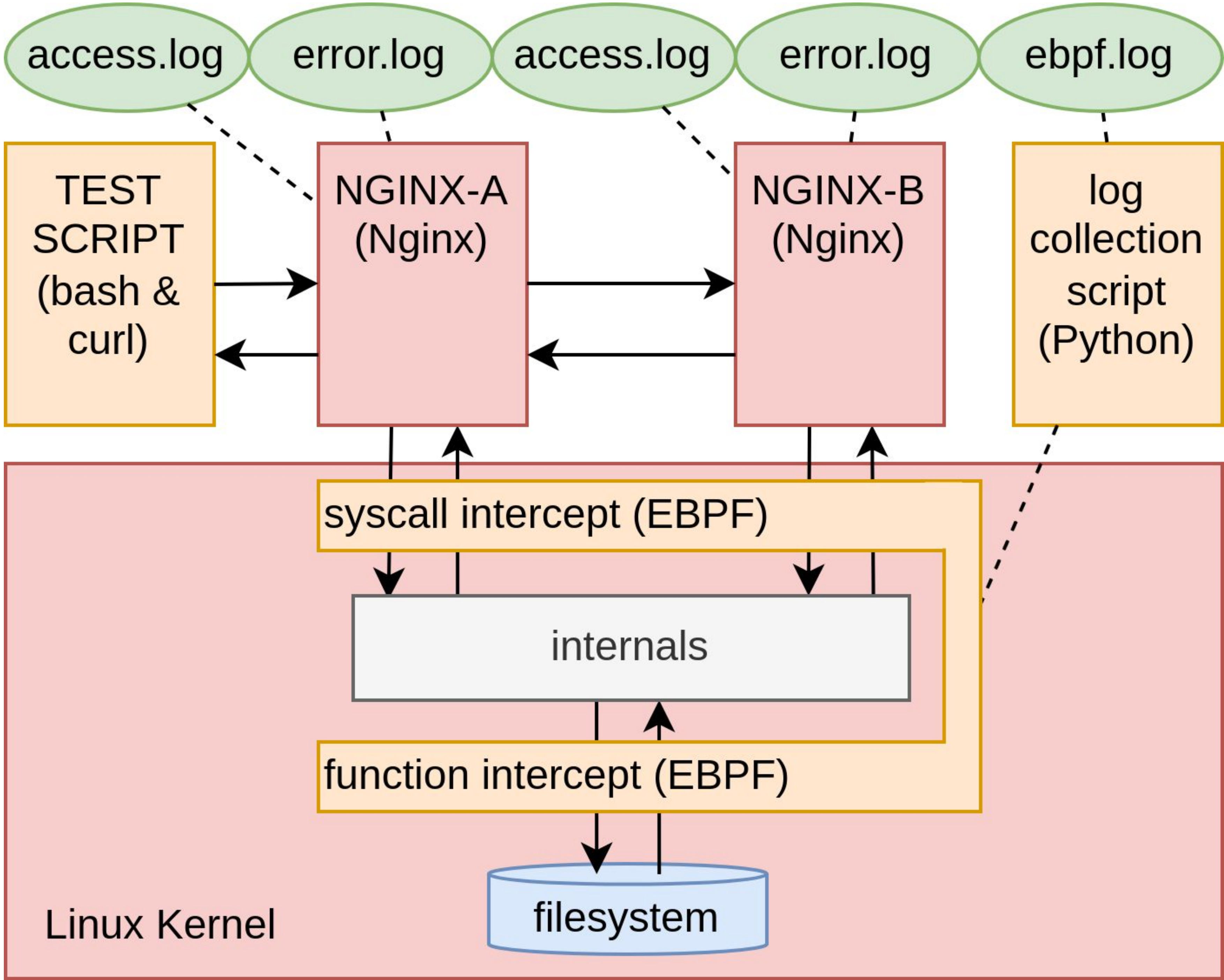
modify:

- Nginx
  - upstream-chaining
  - docker
- Linux kernel
  - (using eBPF)
  - log **some** syscalls

(nginx = a HTTP server)
(eBPF = runtime-injectable monitoring scripts for linux kernel)

github.com/jesajx/gretel

Stockholm University



Implementation: EventID

$$\underbrace{\text{0000000000000005}}_{\substack{\text{Scope} \\ \text{(Logical } \textbf{App ID}\text{)}}}\text{–}\underbrace{\text{64aa92aa321be38d–0551d49a4443f80f–74366e774f124ed7}}_{\substack{\text{Scope-specific Identifier} \\ \text{(}\textbf{random number}\text{)}}}$$

- 256-bit integer   (~67 hexchars)

- some options:
  - random number
  - hash
  - sequential ID

Stockholm
University

Implementation: algorithm

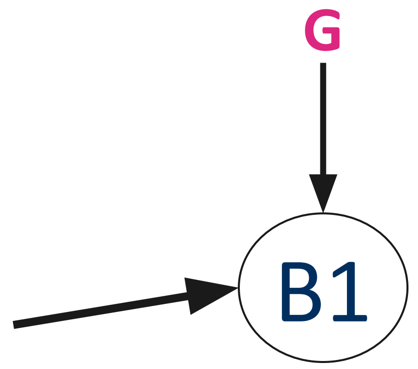
```
thread_local G = ...
```

```
print(f"{x} happened")
```

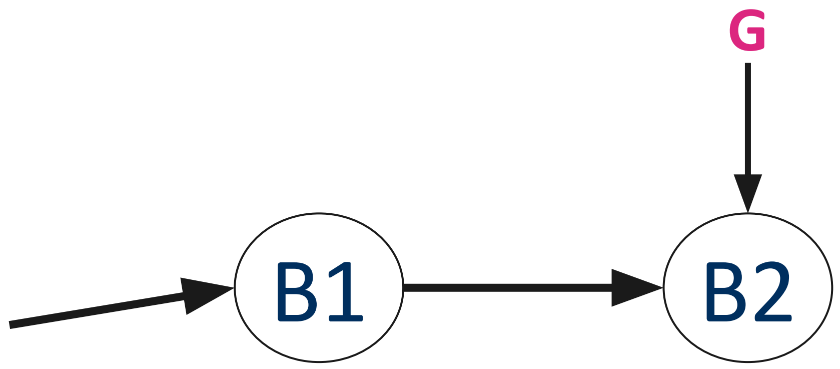
becomes:

```
(old, new) = advance()
print(f"{x} happened, ID={new} pred={old}")
```

before:



after:



Stockholm University

Implementation: Messages

```
GET /path HTTP/1.1
...
```

becomes:

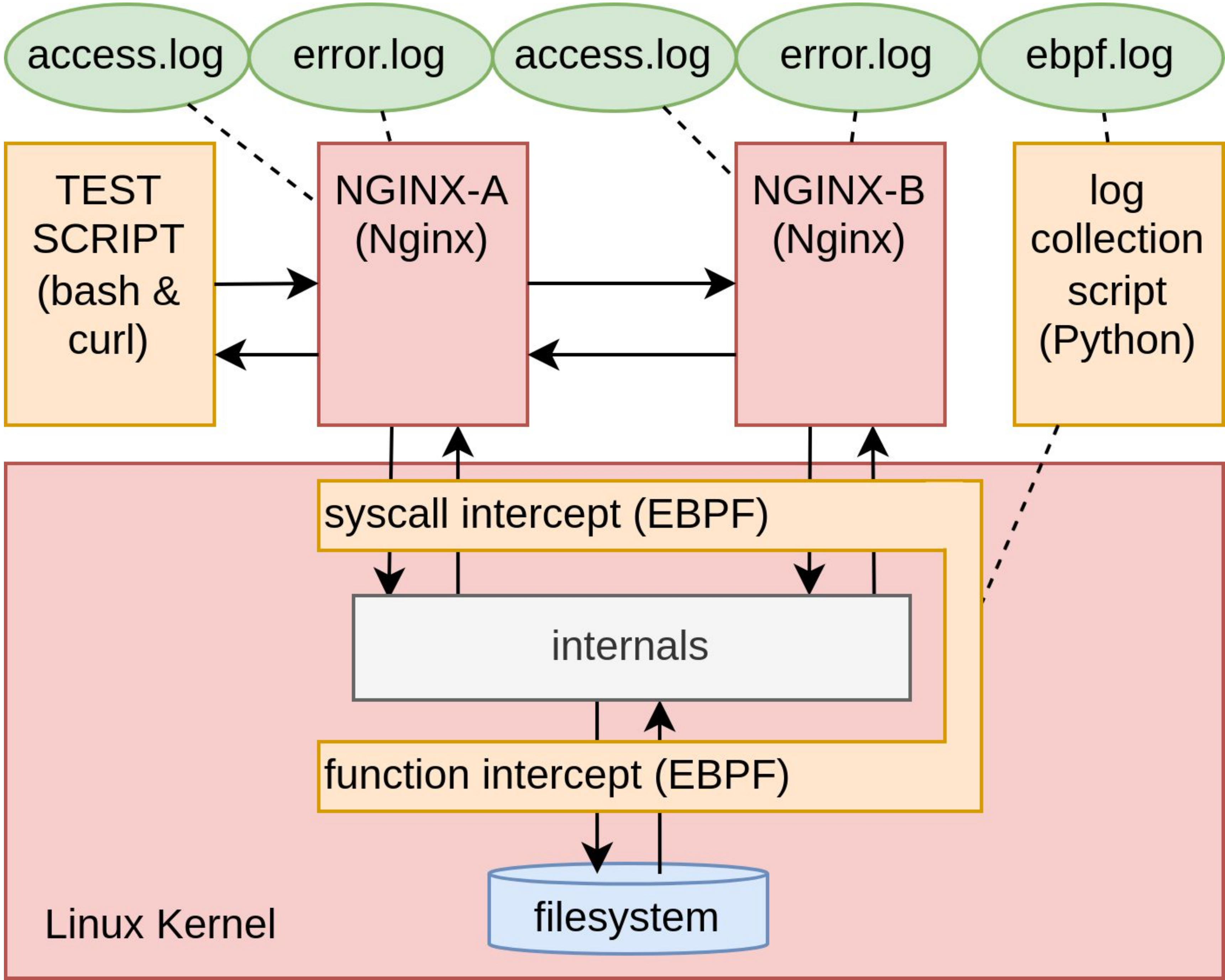
```
GET /path HTTP/1.1
gretel: G
...
```

systemcalls: prctl() + elbow grease

Stockholm University

Experiment

- send 10k GET-requests
  - repeat 10 times
  - with/without gretel
- measure:
  - CPU usage (user+sys)
  - memory usage
  - log file sizes
  - (message sizes)



Stockholm University

Result

- CPU/RAM usage increase: **tiny** (< 1%)
- HTTP header increase: “**medium**” (2x)
  - (likely dwarfed by body in real world.)
- log file size increase: **big** **(x2.5)**
  - “worst case scenario”: 256-bit gretel numbers
    - but this is adjustable! (also: text vs binary logs)

Stockholm University

Conclusion

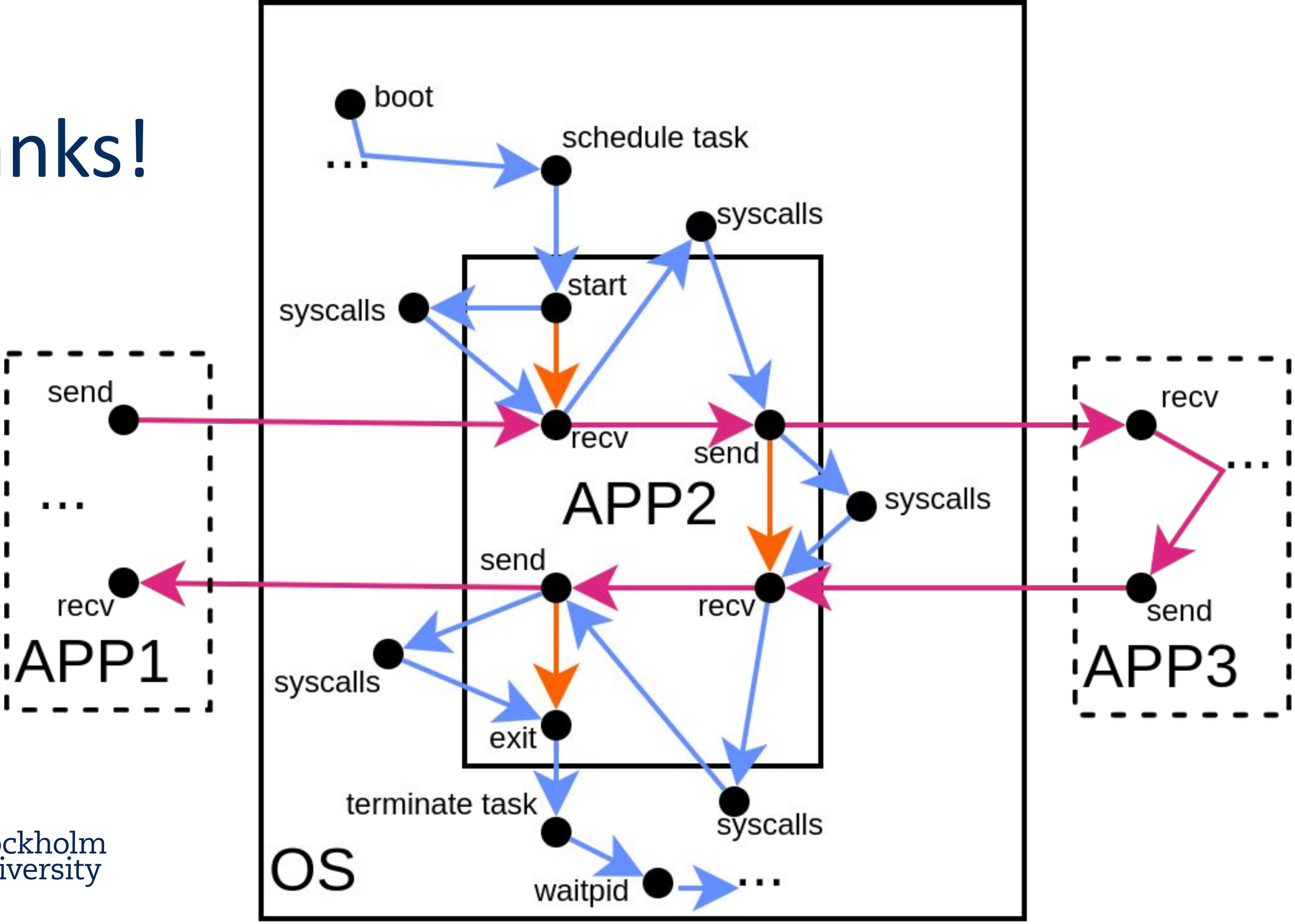
- **We proposed that:**
  - Causal Information should be stored in logs
  - (as a forensic readiness measure)
- **pros:**
  - improved log-correlation
  - anti-tampering mechanisms
- **cons:**
  - log file size

Future work

- Dig deeper
  - Anti-tampering / **attacks on this logging system**
  - **forensic datasets** with automatically labeled IoCs
- What else to put in logs
  - **Taxonomize forensic questions (+ answers)**

Stockholm University

Thanks!



Stockholm University