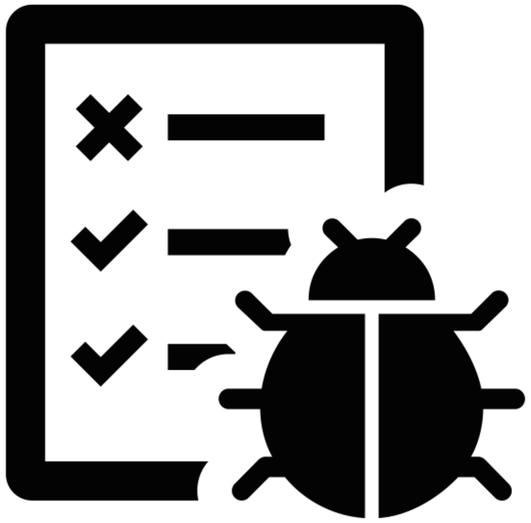
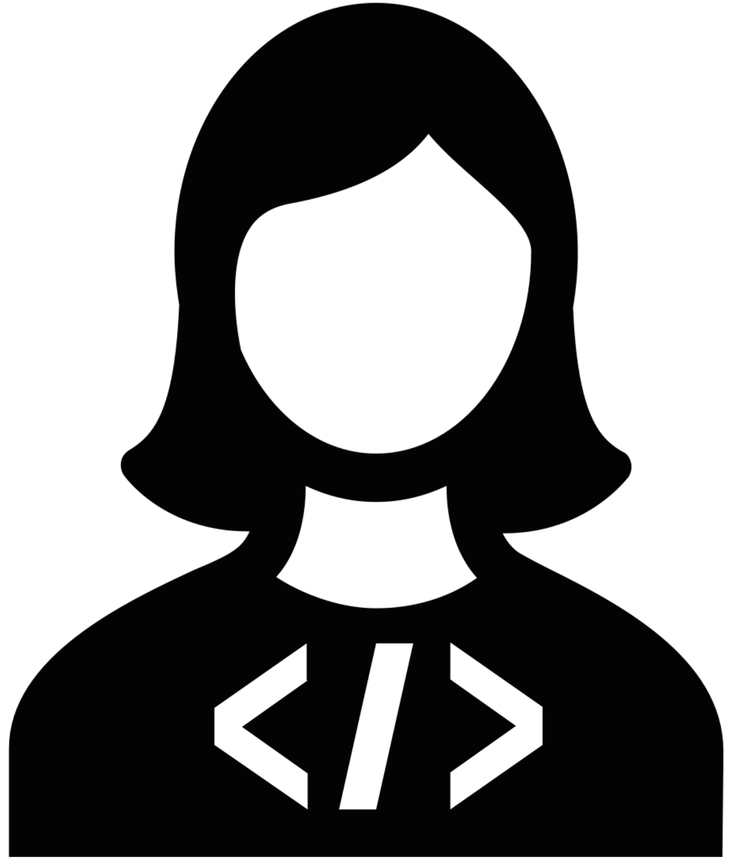
# Minimizing Faulty Executions of Distributed Systems

Colin Scott, Aurojit Panda, Vjekoslav Brajkovic, George Necula,
Arvind Krishnamurthy, Scott Shenker

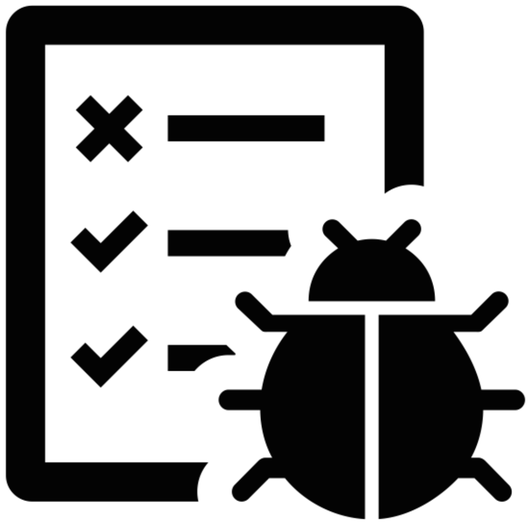






Software Developer









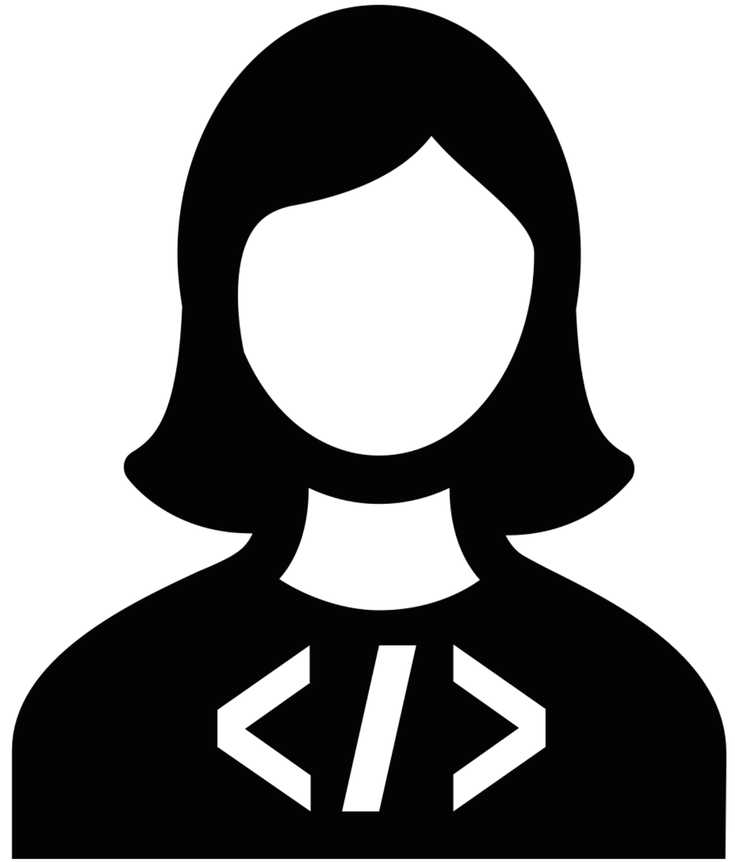




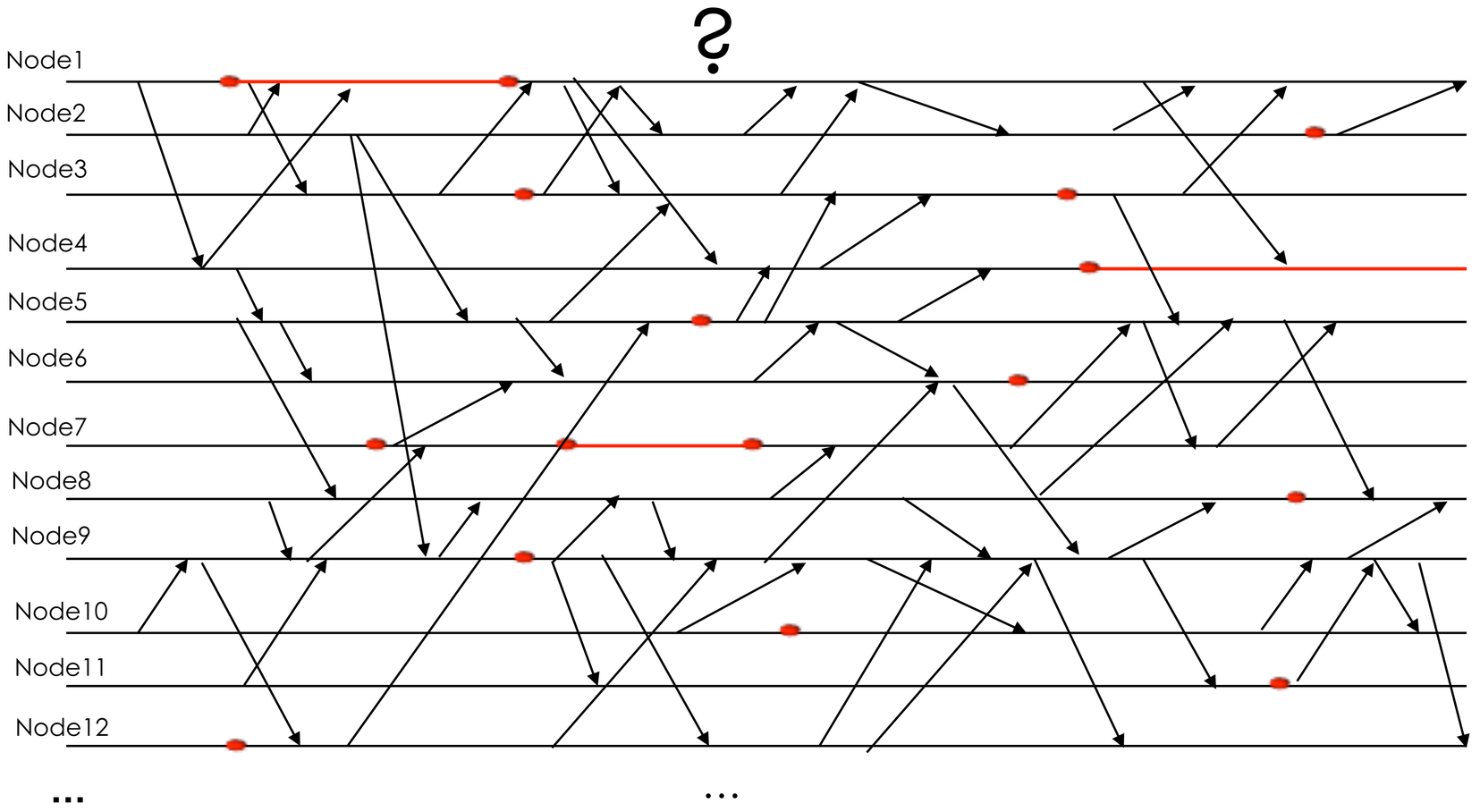


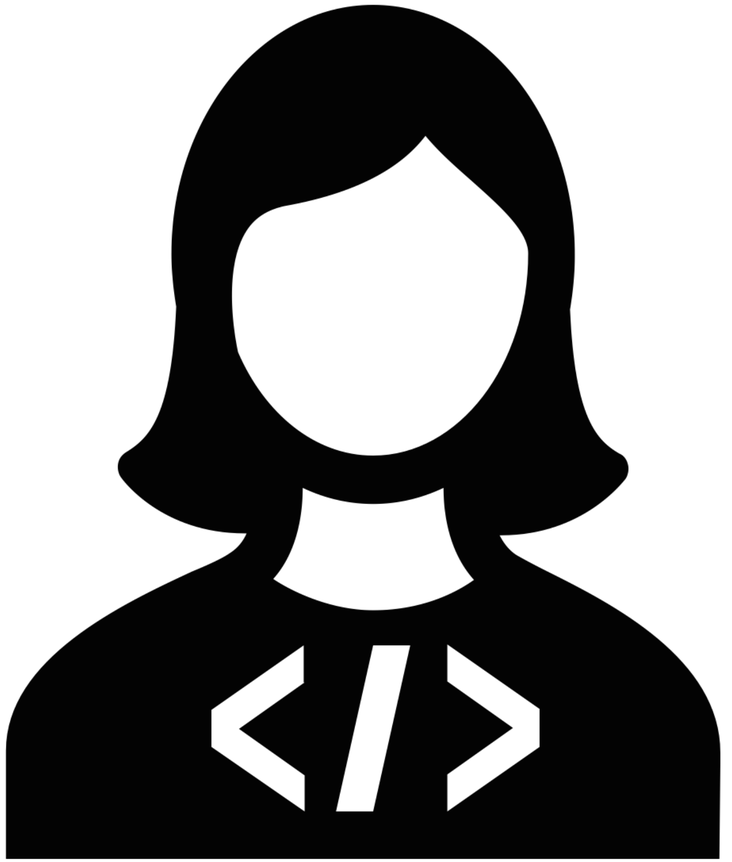



(GBs)



Software Developer



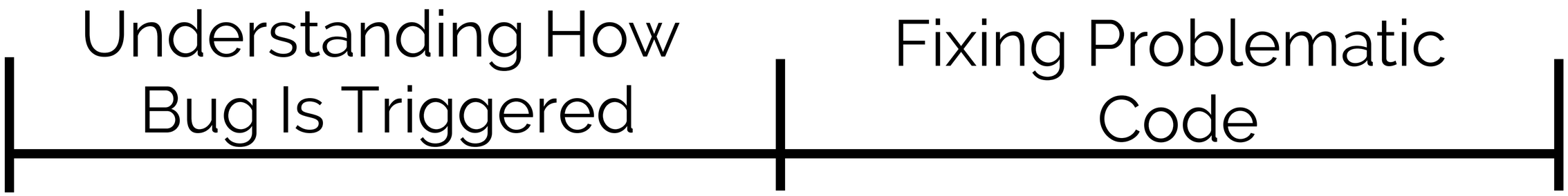



Software Developer

# 49% of developers' time spent on debugging![1]

[1] LaToza, Venolia, DeLine, ICSE' 06

# 49% of developers' time spent on debugging![1]




[1] LaToza, Venolia, DeLine, ICSE' 06

# Our Goal

Allow Developers To Focus on Fixing the Underlying Bug

# Problem Statement

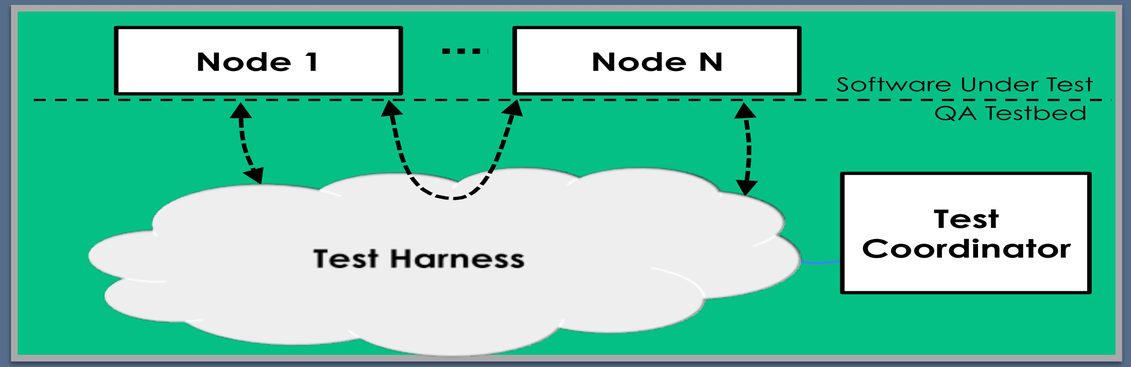
Identify a **minimal causal sequence** of events that triggers the bug

# Outline

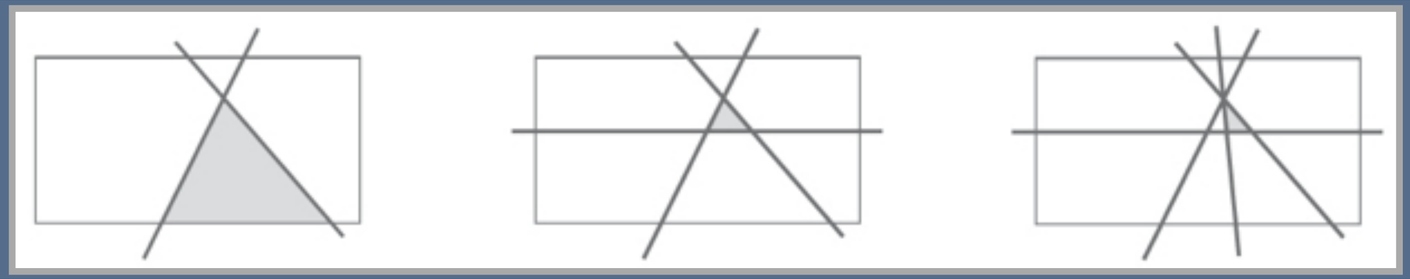
Introduction

Background

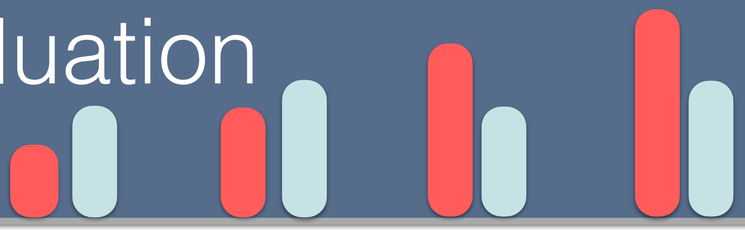
Randomized Testing with DEMi




Minimization



Evaluation



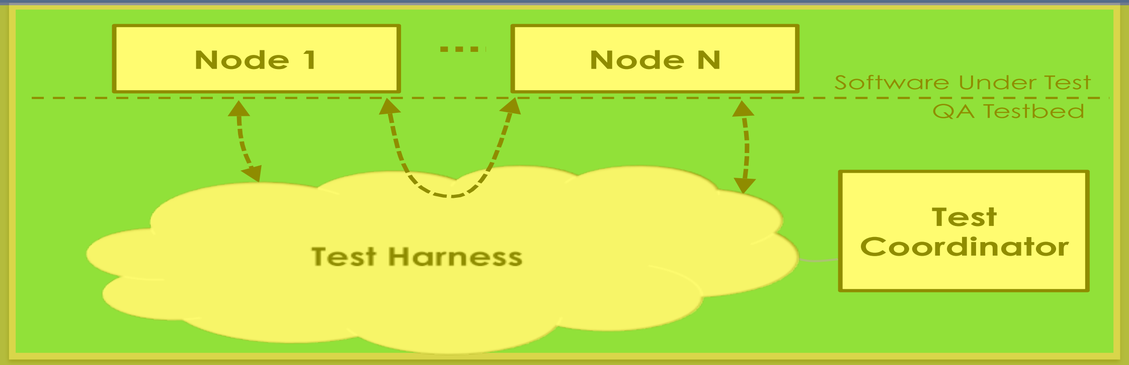
Conclusion

# Outline

Introduction

Background

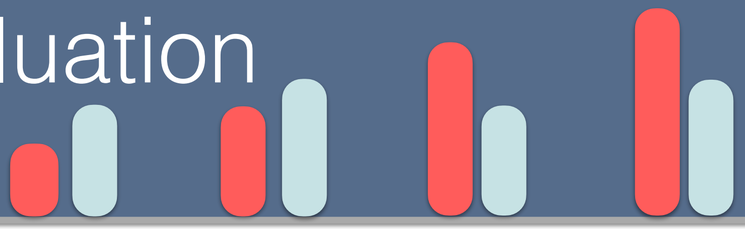
Randomized Testing with DEMi




Minimization



Evaluation



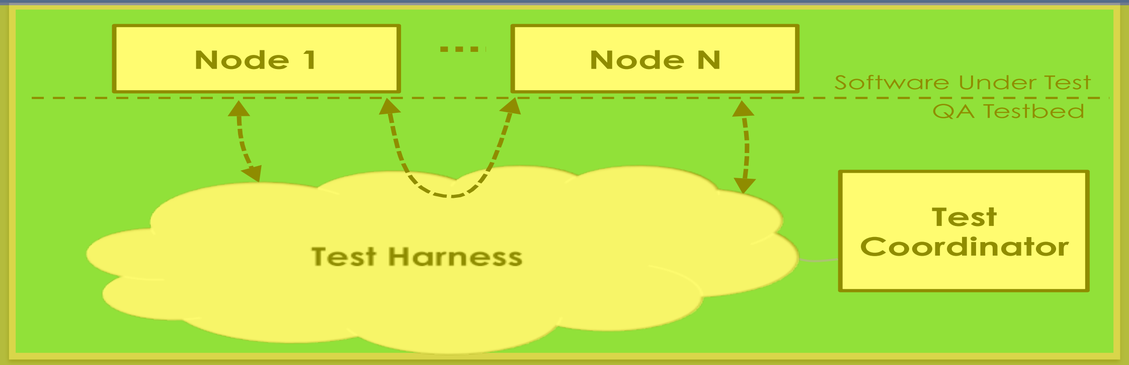
Conclusion

# Outline

Introduction

Background

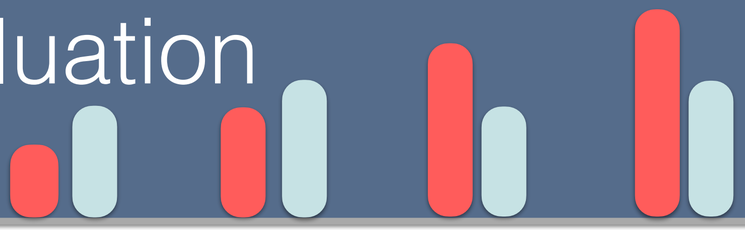
Randomized Testing with DEMi




Minimization



Evaluation



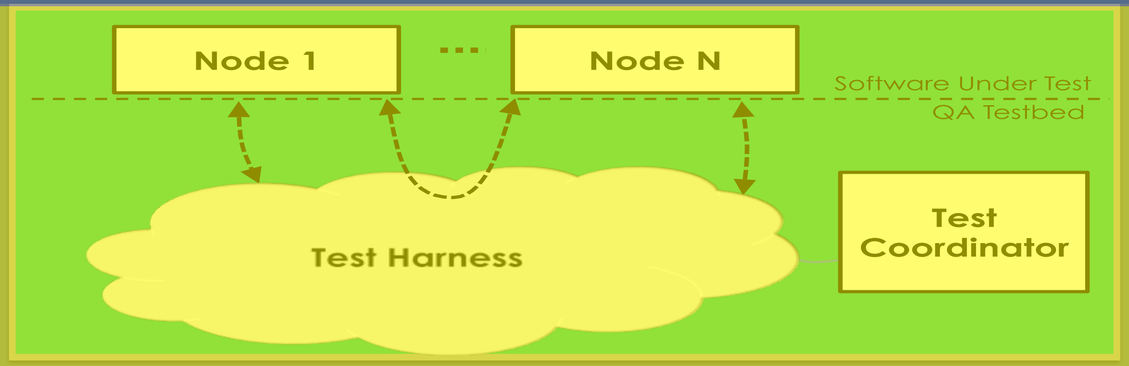
Conclusion

# Outline

Introduction

Background

Randomized Testing with DEMi




Minimization

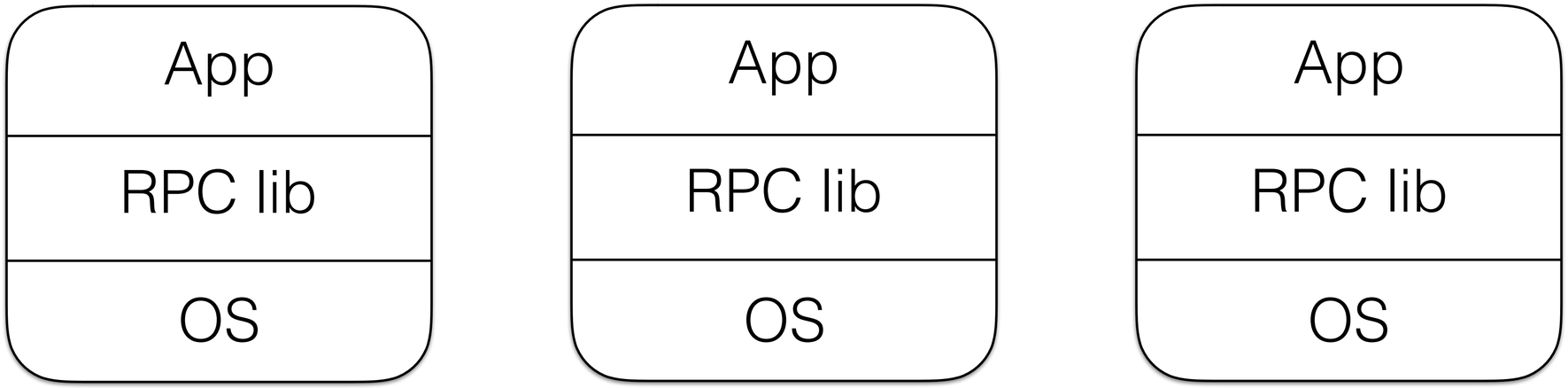


Evaluation

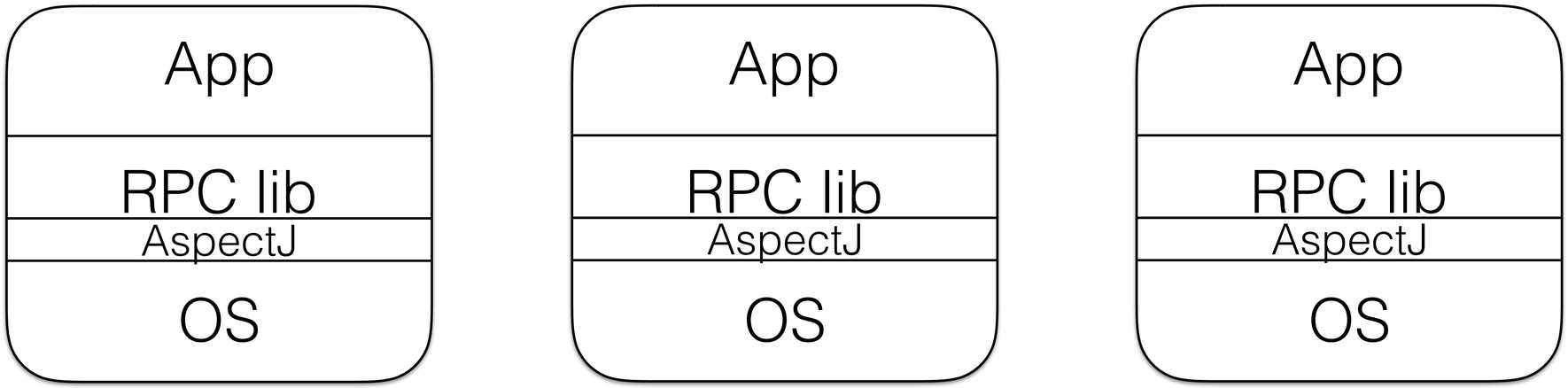


Conclusion

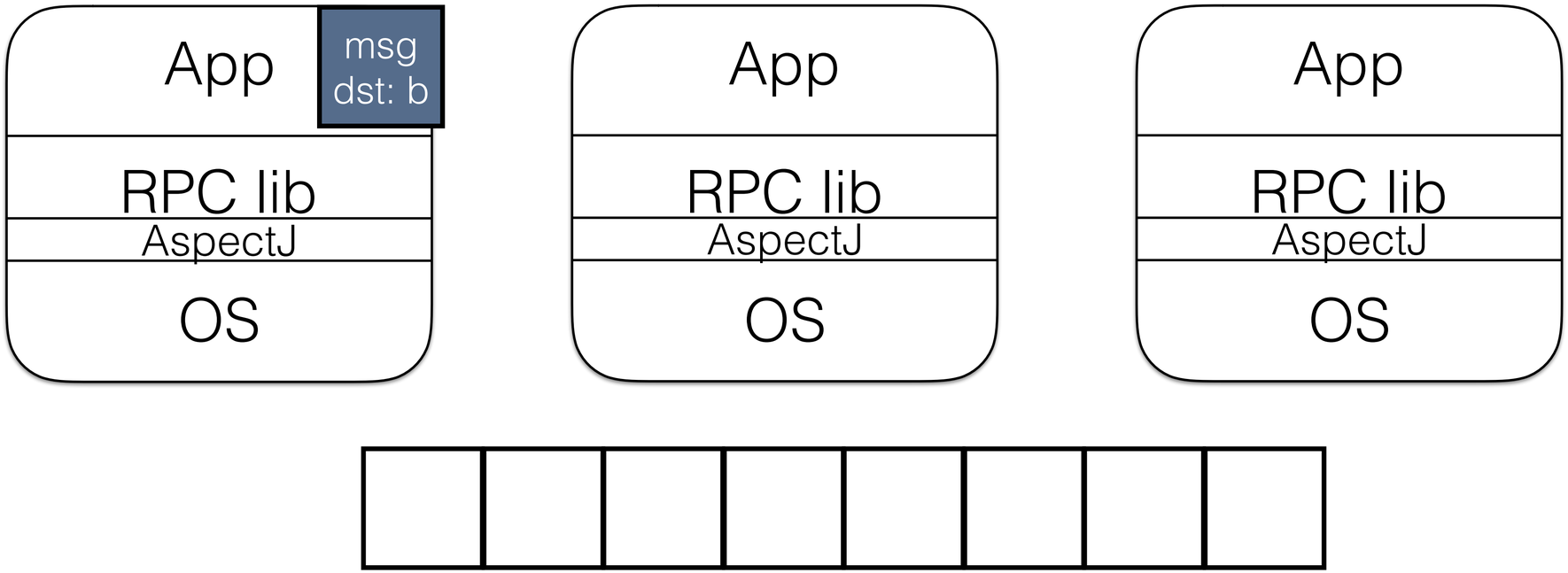
# Randomized Testing with DEMi



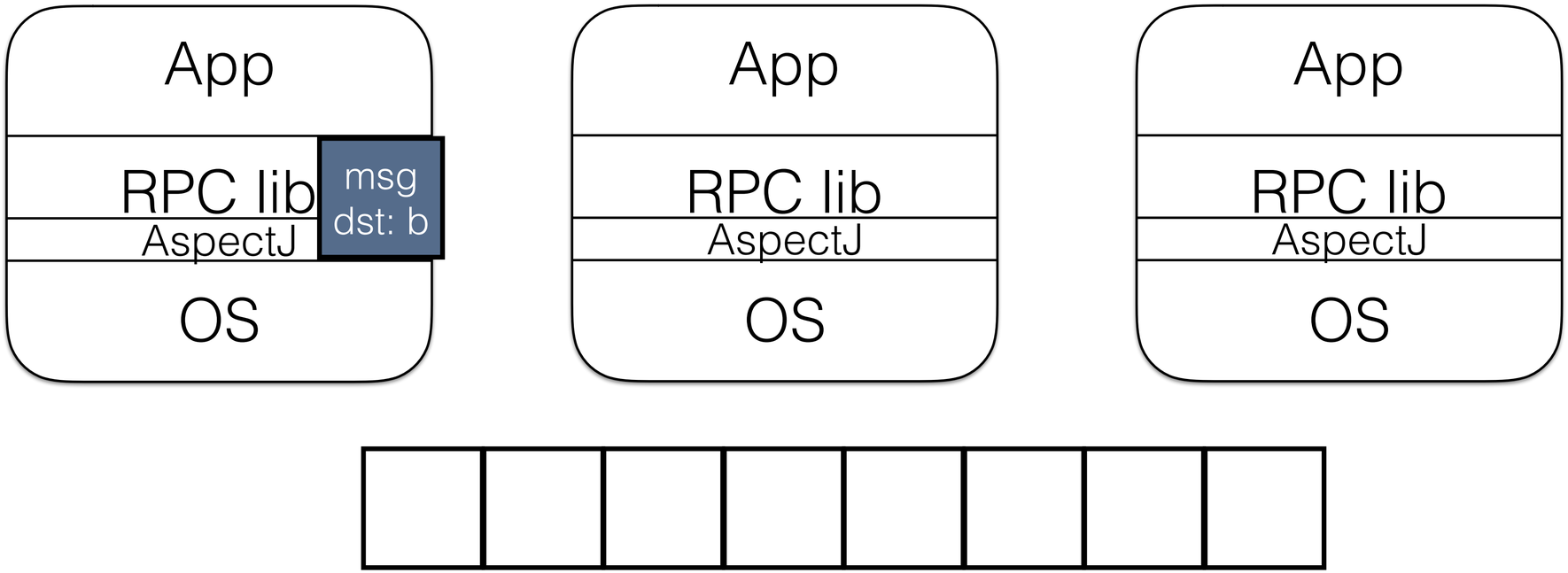
# Randomized Testing with DEMi



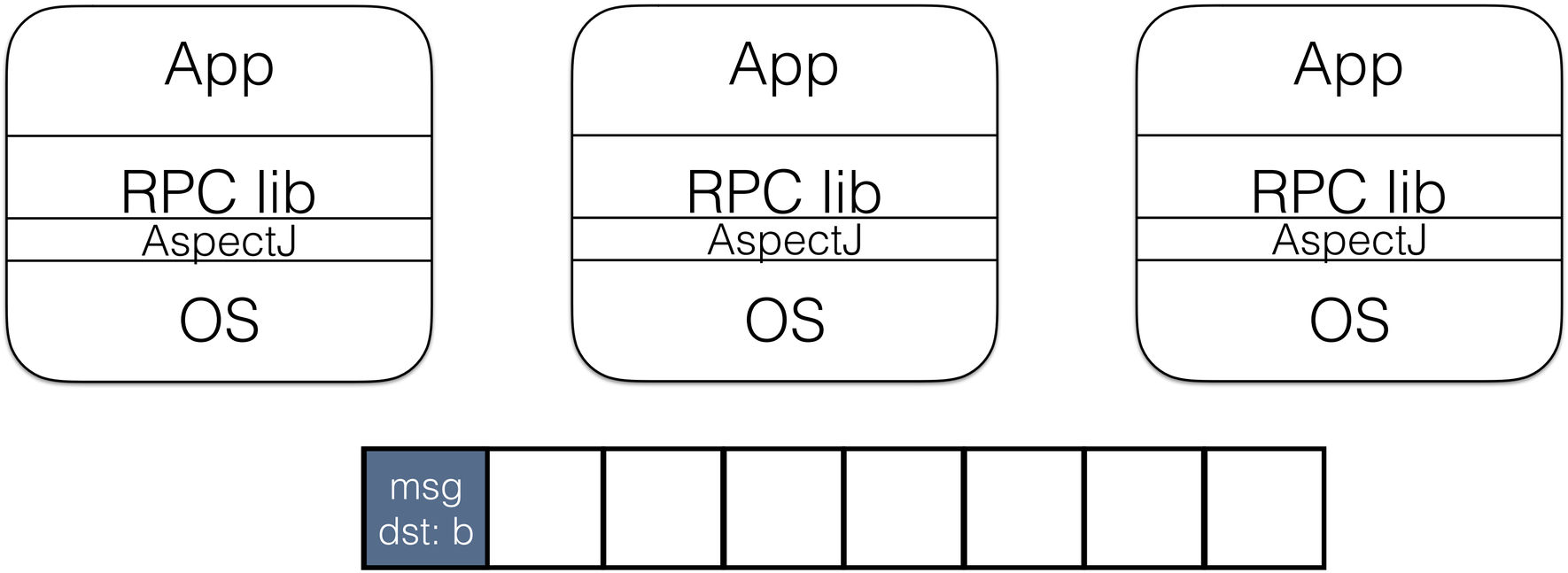
# Randomized Testing with DEMi



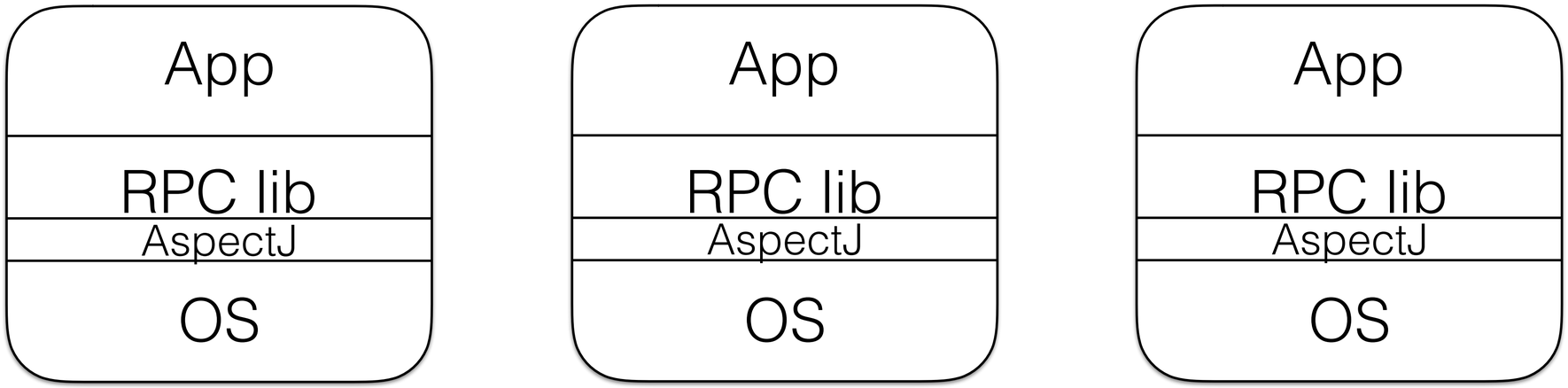
# Randomized Testing with DEMi



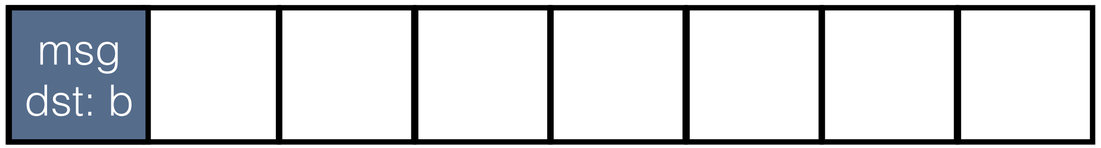
# Randomized Testing with DEMi



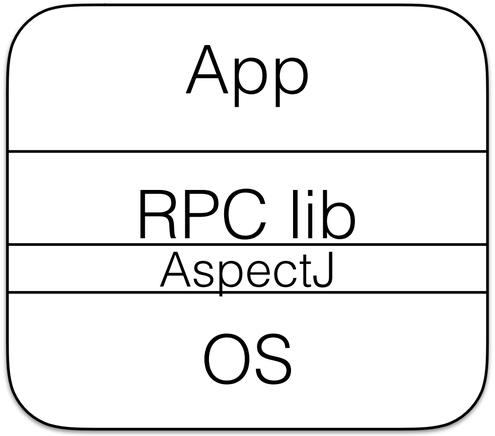
# Randomized Testing with DEMi

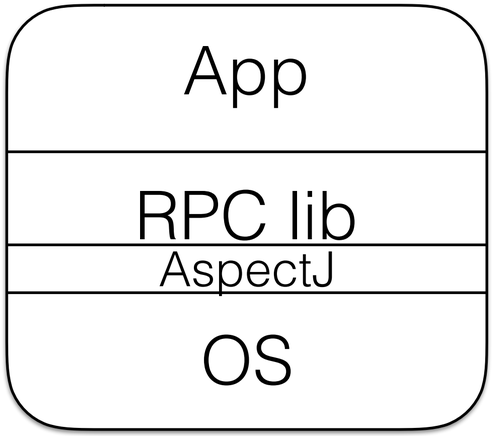


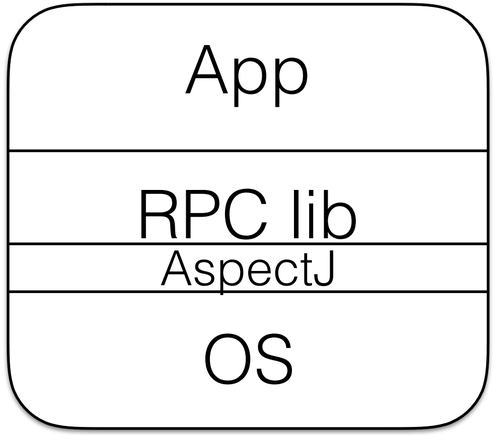


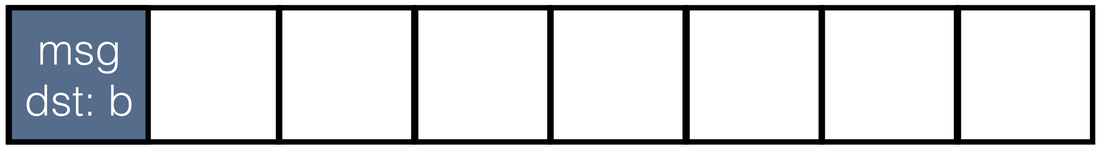
# Randomized Testing with DEMi





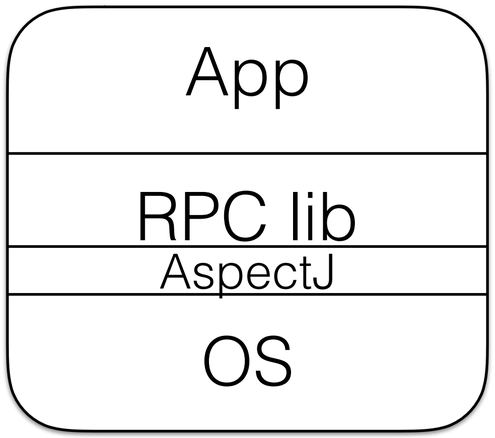



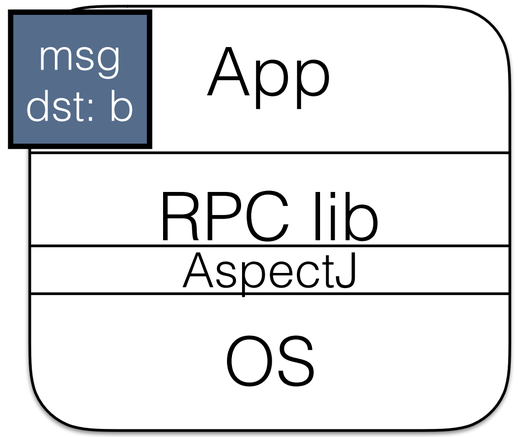


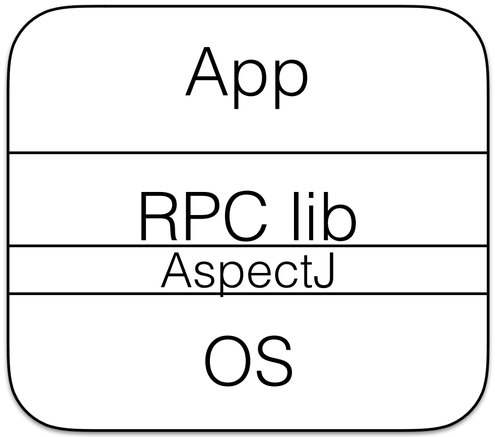



message delivery

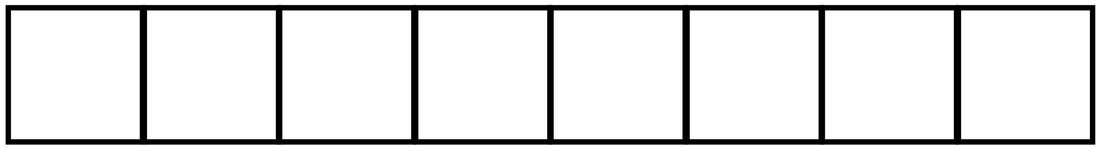
# Randomized Testing with DEMi





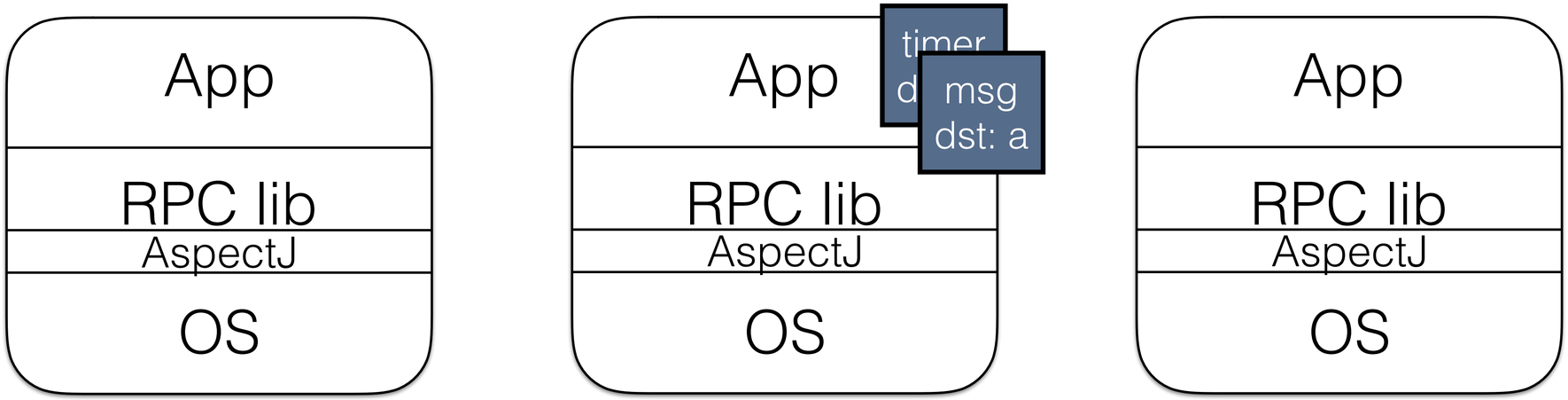





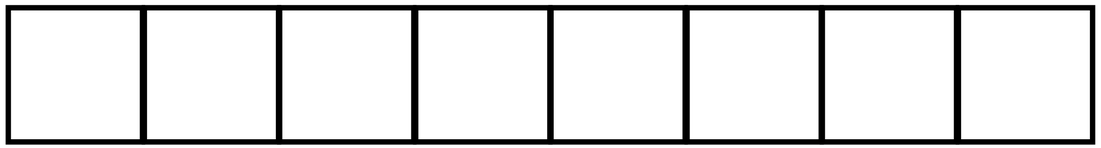


message
delivery

# Randomized Testing with DEMi





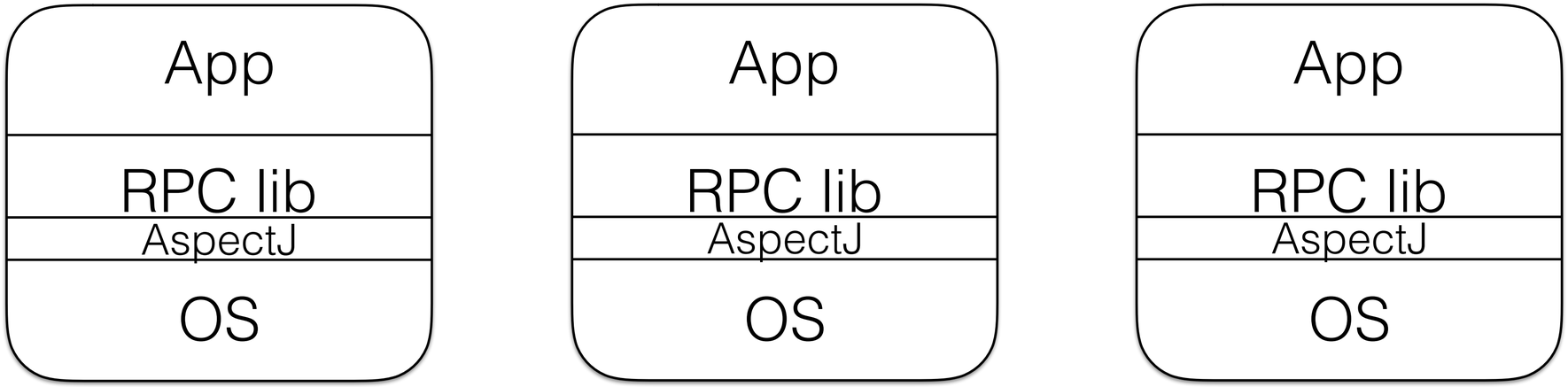





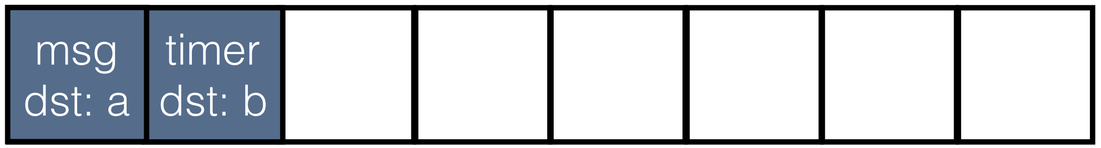


message
delivery

# Randomized Testing with DEMi







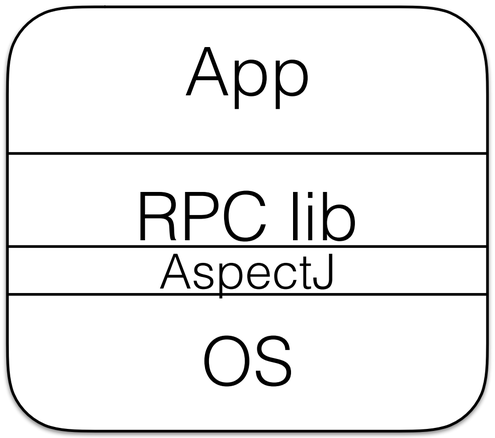



message
delivery

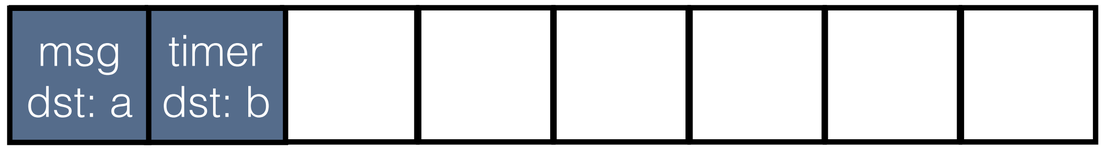
# Randomized Testing with DEMi

External events (events outside system's control):

- Crash-recovery
- Process creation
- External message







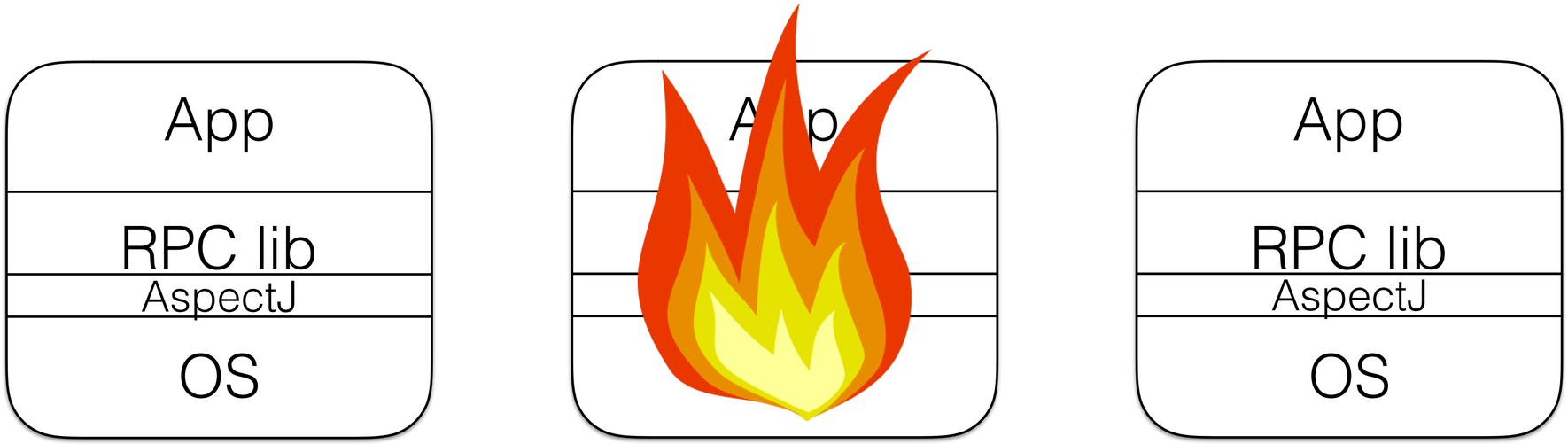



message delivery

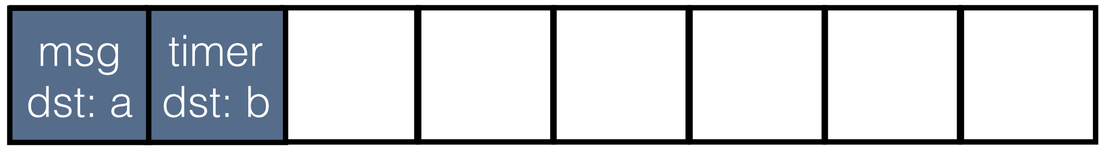
# Randomized Testing with DEMi

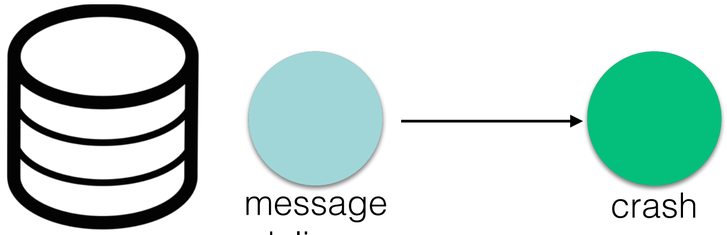
External events (events outside system's control):

- Crash-recovery
- Process creation
- External message





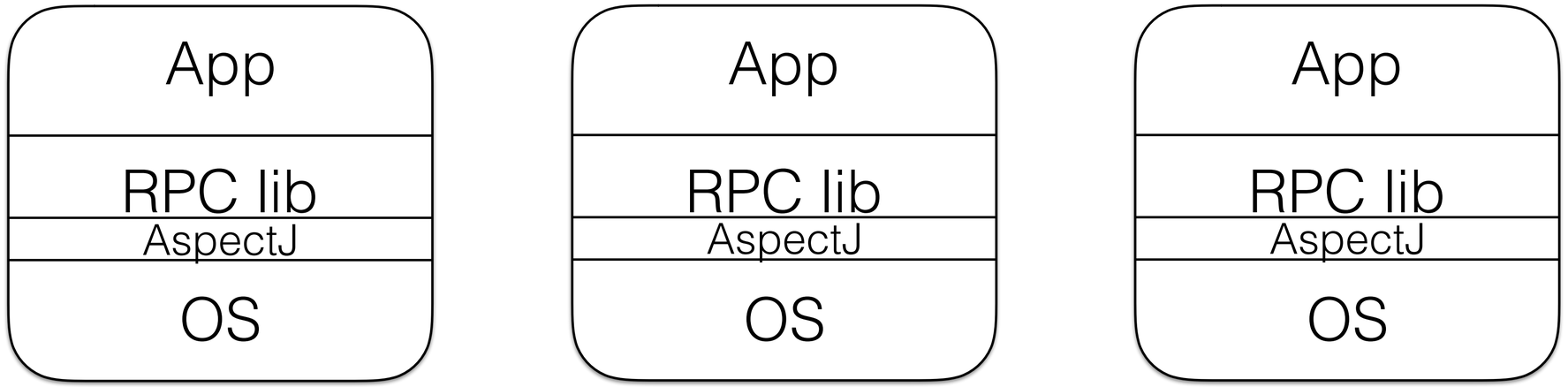



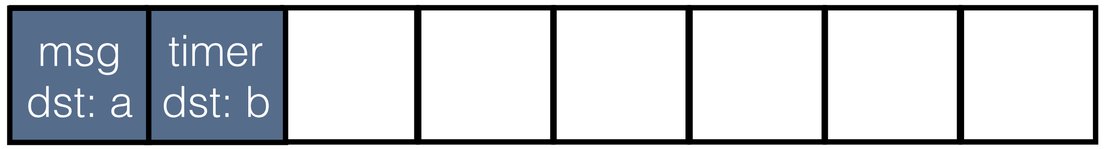
# Randomized Testing with DEMi

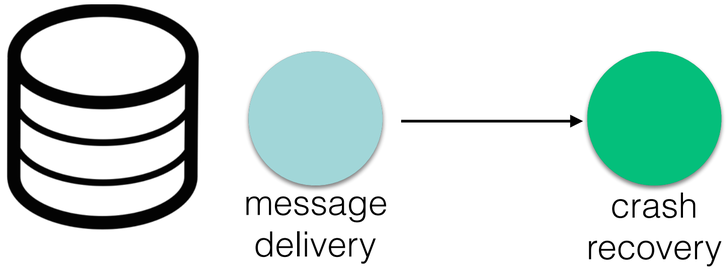
External events (events outside system's control):

- Crash-recovery
- Process creation
- External message



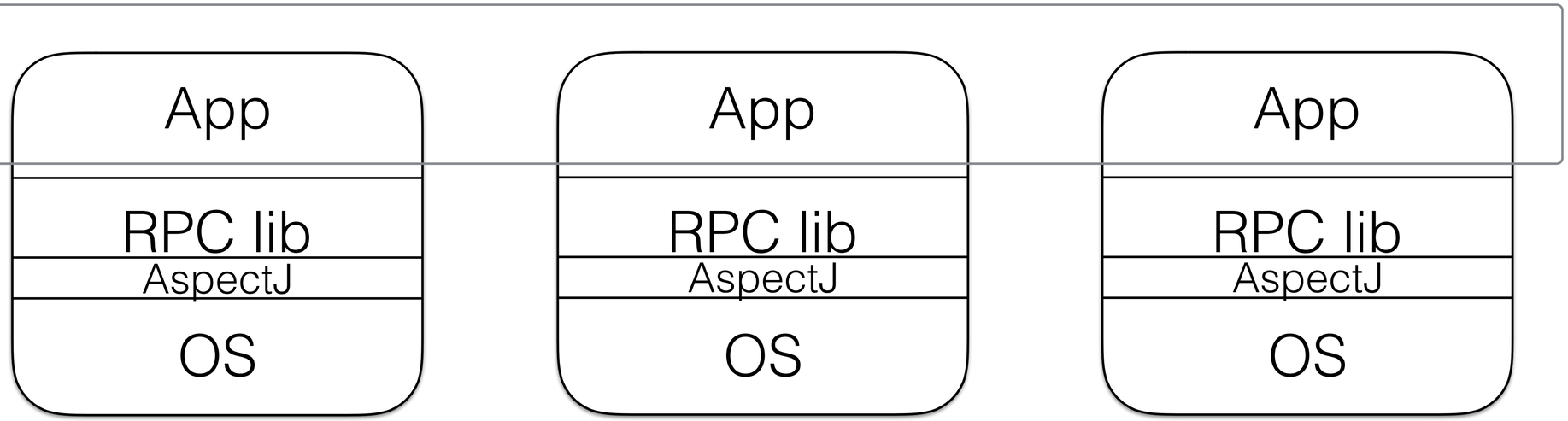


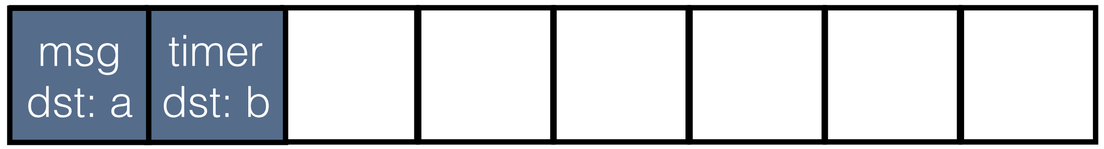



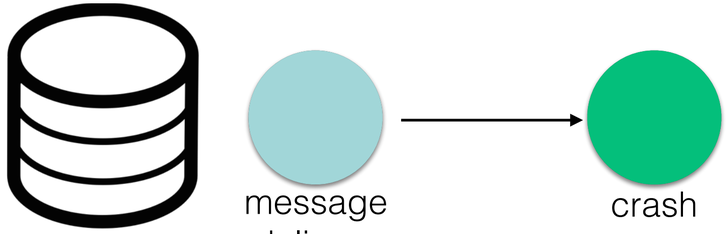
# Randomized Testing with DEMi



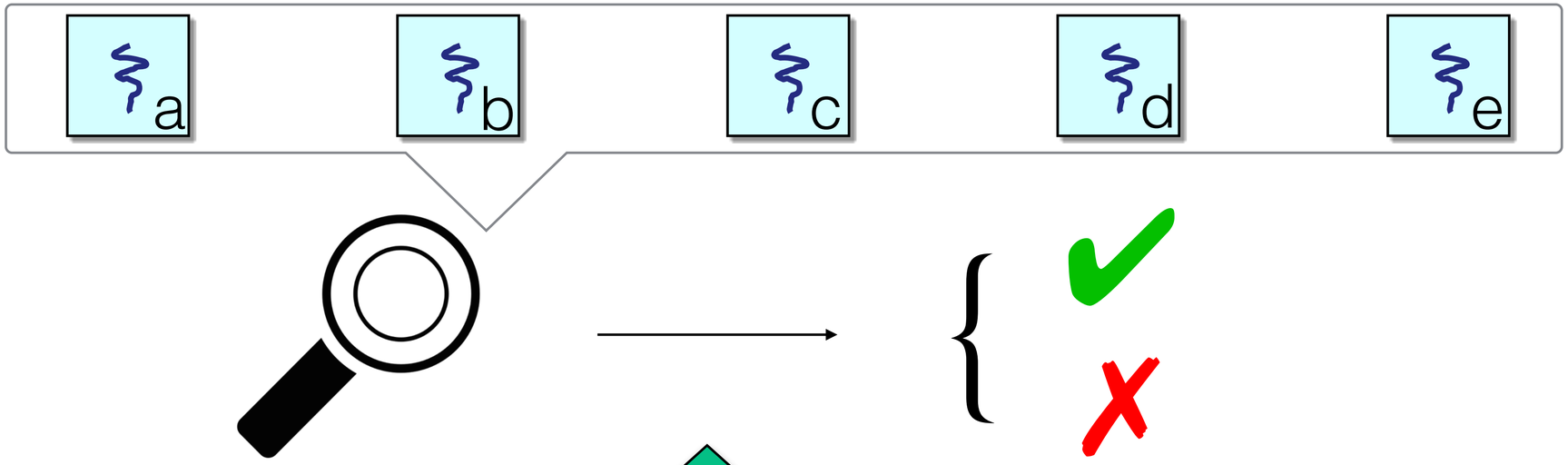




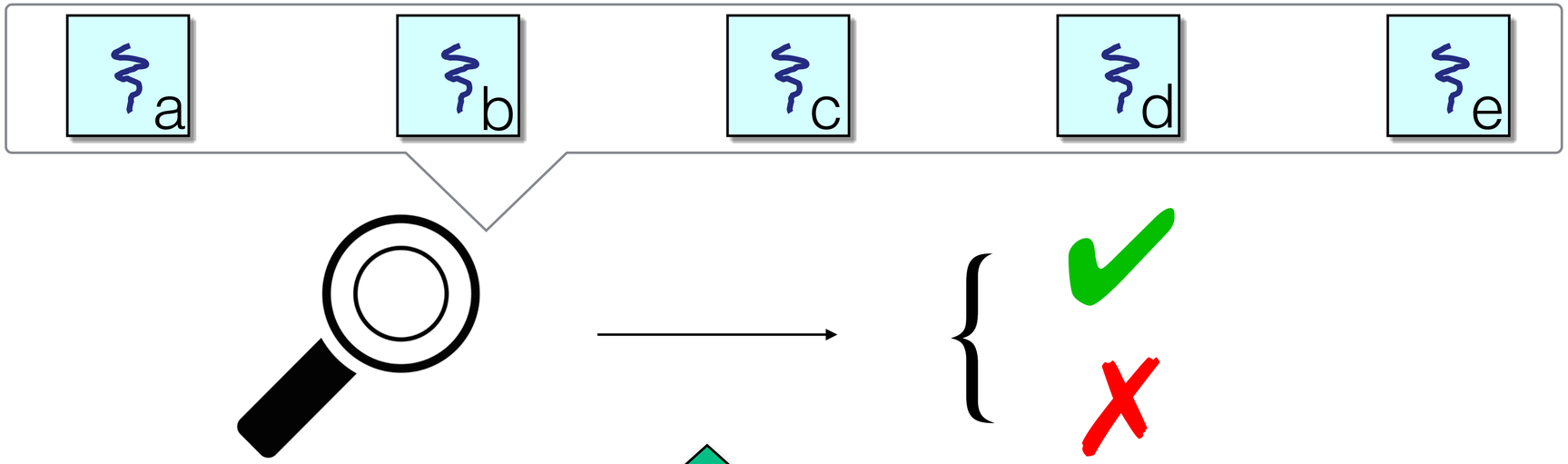



# Invariant Checking



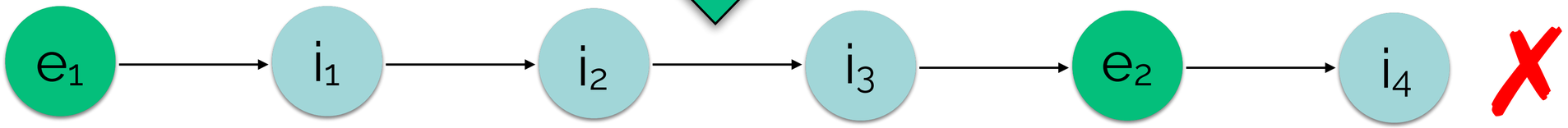

An invariant is a predicate *P* over the state of all processes.

# Invariant Checking




An invariant is a predicate *P* over the state of all processes.

A faulty execution is one that ends in an invariant violation.

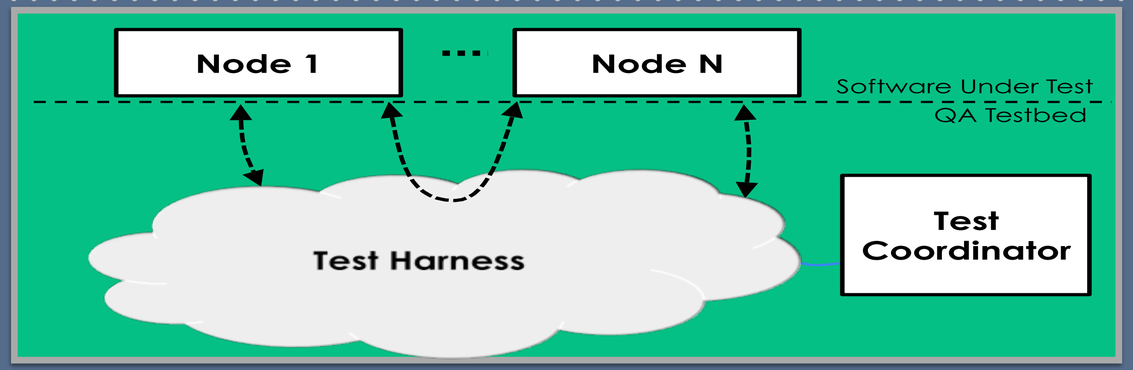


# Outline

Introduction

Background

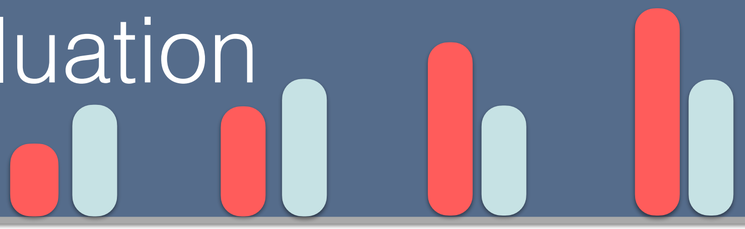
Randomized Testing with DEMi




Minimization



Evaluation



Conclusion

# Formal Problem Statement

Given: schedule $\tau$ that results in violation of P

Find: locally minimal reproducing sequence $\tau$':

- $\tau$' violates P, $|\tau'| \leq |\tau|$
- $\tau$' contains a subsequence of the external events of $\tau$
- if we remove any external event e from $\tau$',
- $\neg\exists\ \tau''$ containing same external events - e, s.t. $\tau''$ violates P

# Formal Problem Statement

After finding τ', minimize internal events:

- remove extraneous message deliveries from τ'

Log lines | Motifs

```
Sending to raft-member-4: RequestVote(Ter
Sending to raft-member-2: BeginElection

Received message from raft-member-3: Requ
Received timer: Timer(election-timer,Elec
Received message from raft-member-2: Begi

Sending to raft-member-3: VoteCandidate(T
Sending to raft-member-3: BeginElection
Sending to raft-member-1: RequestVote(Ter

Received message from raft-member-4: Vote
Received message from raft-member-2: Requ

Received message from raft-member-3: Begi
Sending to raft-member-2: VoteCandidate(T

Sending to raft-member-4: RequestVote(Ter
Received message from raft-member-1: Vote
Received timer: Timer(election-timer,Elec

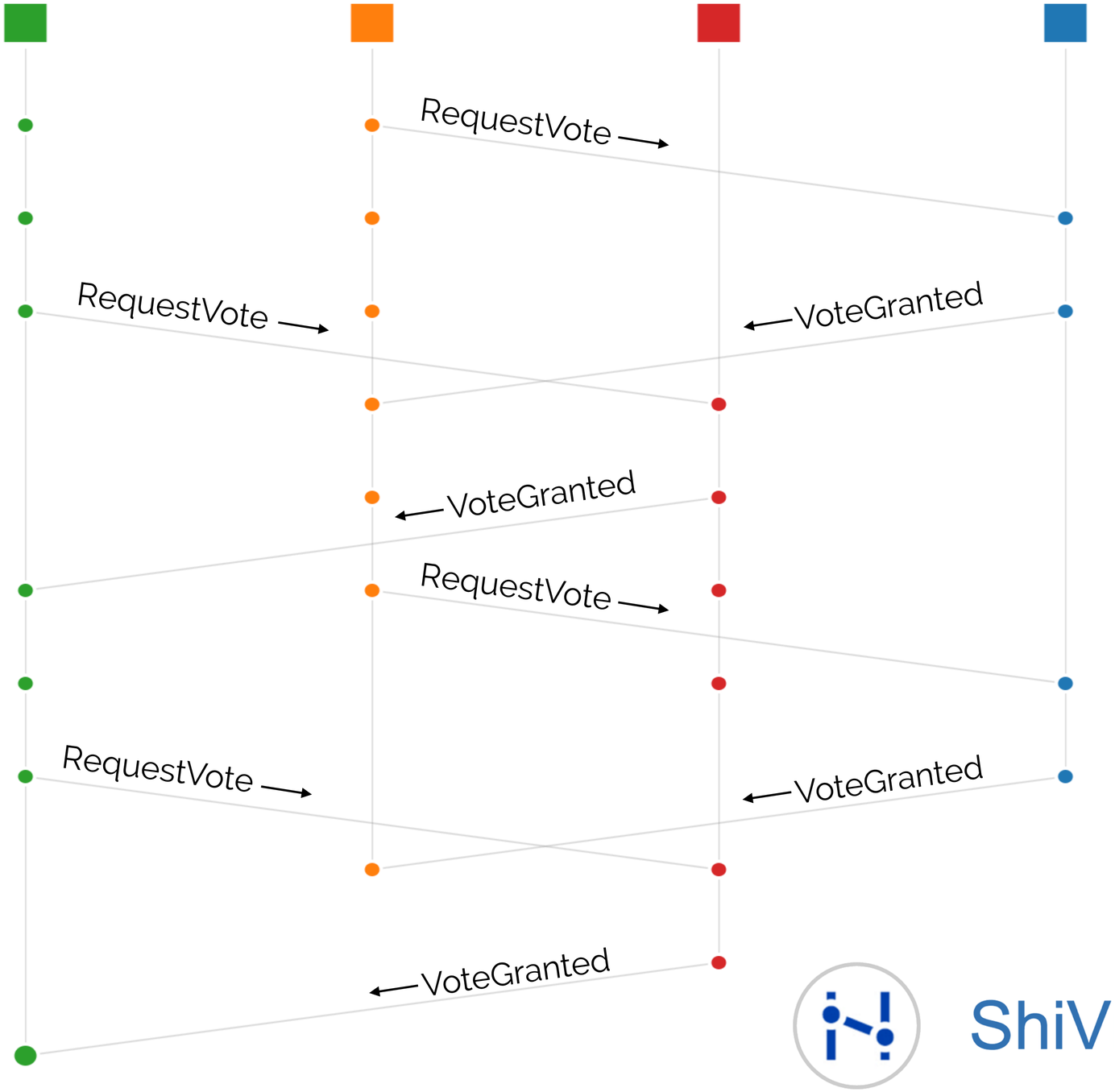
Received message from raft-member-3: Requ
Received message from raft-member-2: Begi
Received timer: Timer(election-timer,Elec

Sending to raft-member-3: VoteCandidate(T
Sending to raft-member-1: RequestVote(Ter

Received message from raft-member-4: Vote
Received message from raft-member-2: Requ

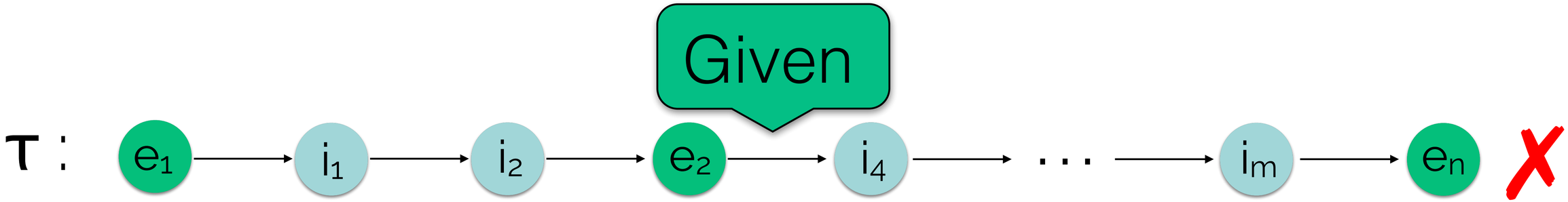
Sending to raft-member-2: VoteCandidate(T
```

```
34 Received message from raft-member-1: Vo
   teCandidate(Term(2))
```

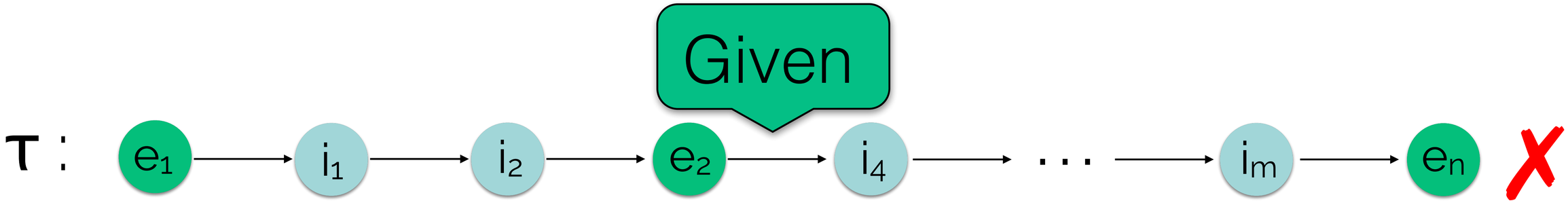



ShiViz

# Minimization

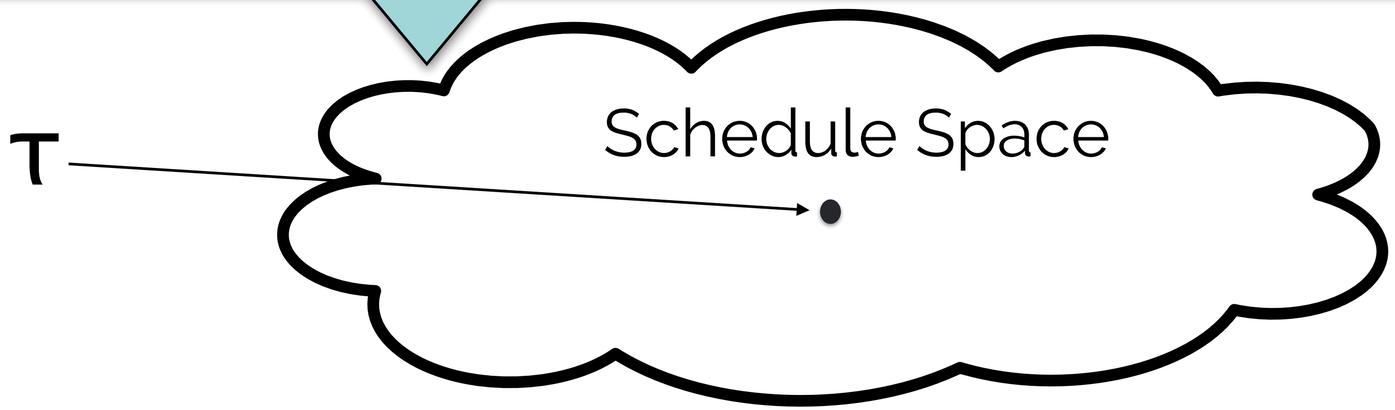


# Minimization



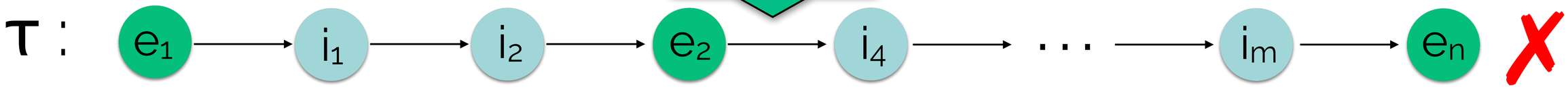

Straightforward approach:

- Enumerate all schedules $|\tau'| \leq |\tau|$,
- Pick shortest sequence that reproduces ✗



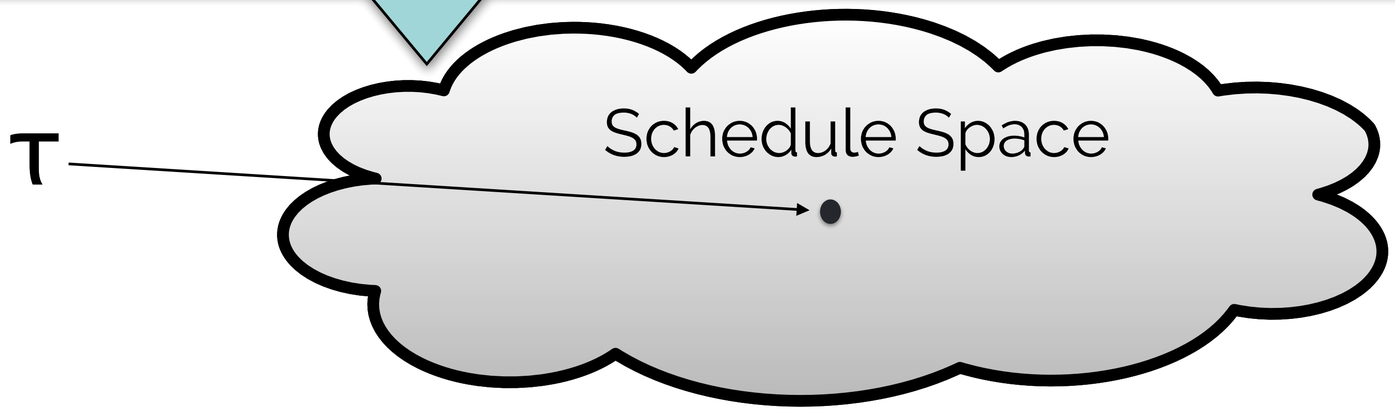
# Minimization

Given




Straightforward approach:

- Enumerate all schedules $|\tau'| \leq |\tau|$,
- Pick shortest sequence that reproduces ✗

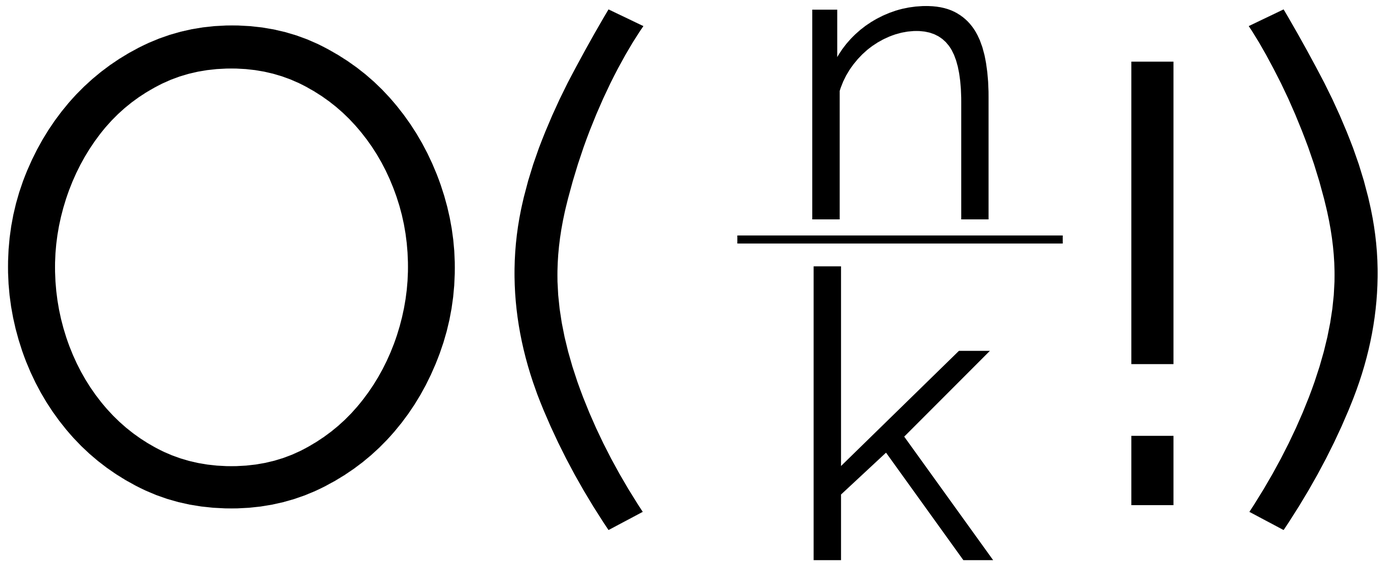


$$O(n!)$$

# Observation #1: many schedules are commutative

Adopt DPOR:
**D**ynamic **P**artial **O**rder **R**eduction

C. Flanagan, P. Godefroid, “Dynamic Partial-Order Reduction for Model Checking Software”, POPL ‘05

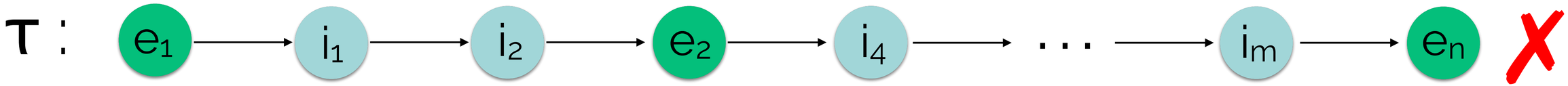

O( n/k ! )

# Approach: prioritize schedule space exploration

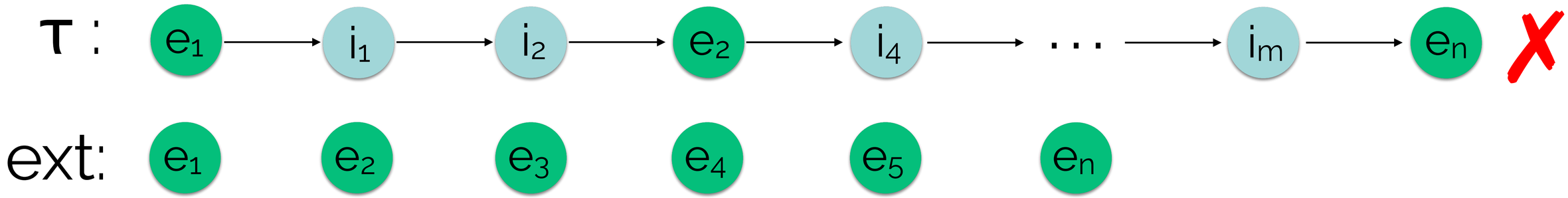
# Approach: prioritize schedule space exploration

Assume: fixed time budget
Objective: quickly find small failing schedules

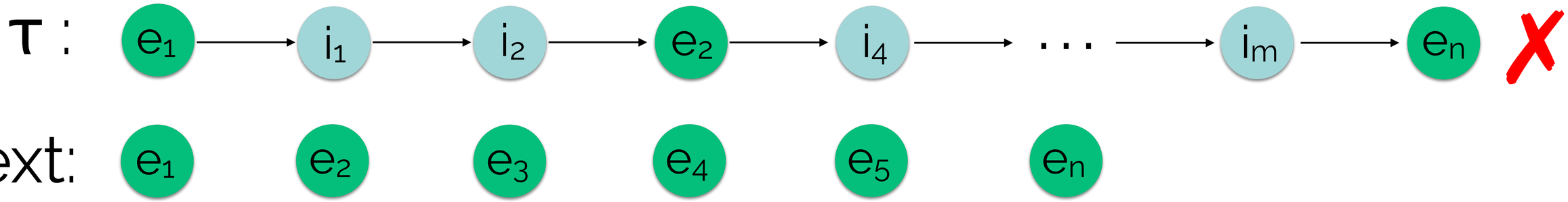
# Observation #2: selectively mask original events



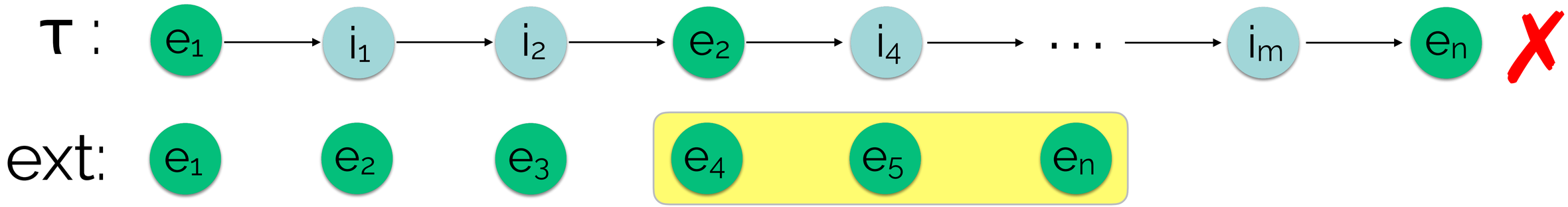
# Observation #2: selectively mask original events



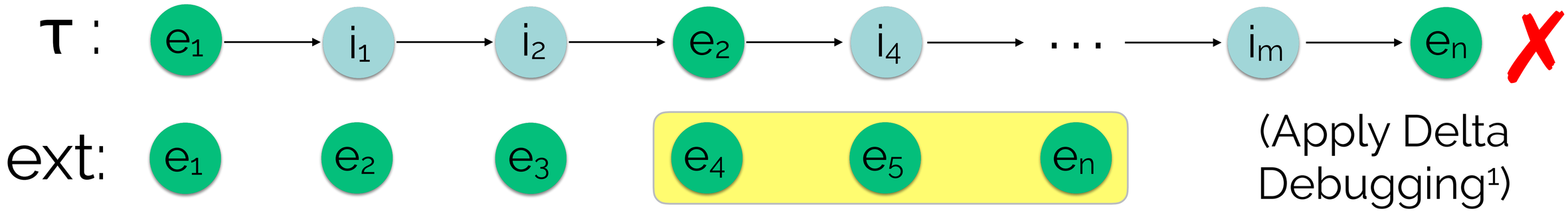
# Observation #2: selectively mask original events



# Observation #2: selectively mask original events

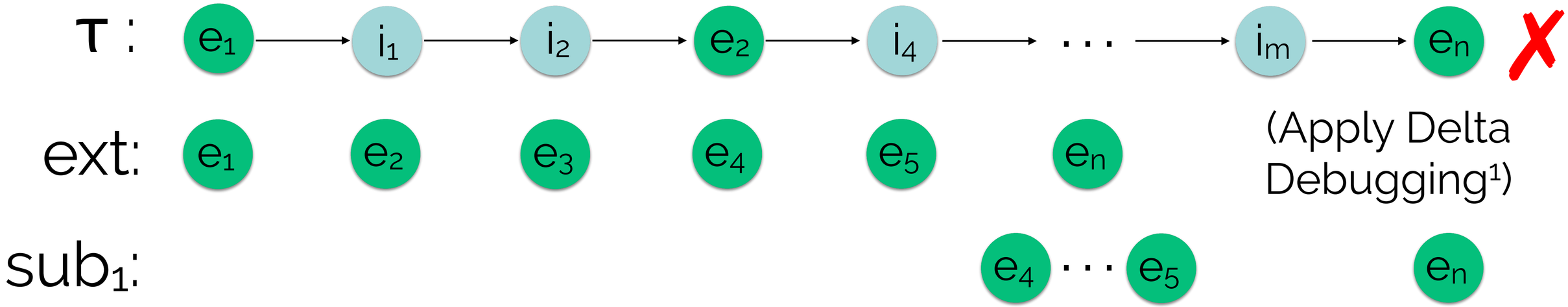


# Observation #2: selectively mask original events



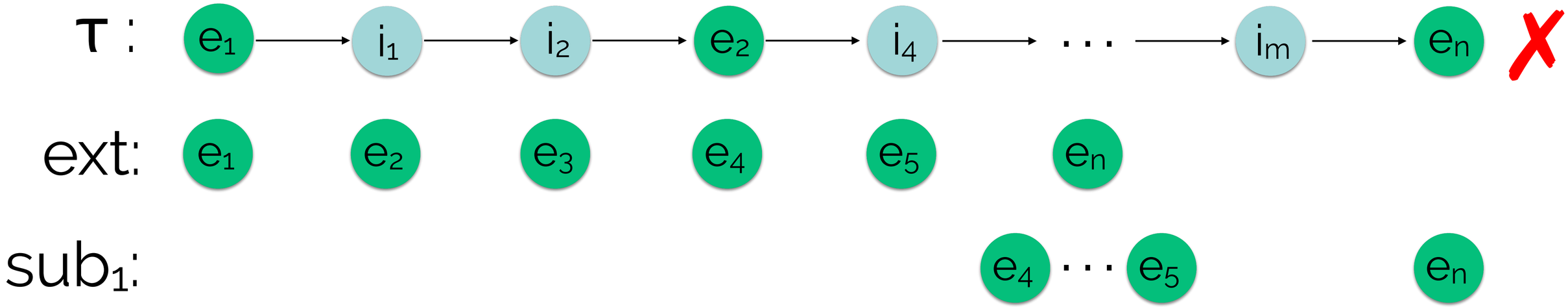

[1]A Zeller, R. Hildebrandt, “Simplifying and Isolating Failure-Inducing Input”, IEEE ‘02

# Observation #2: selectively mask original events



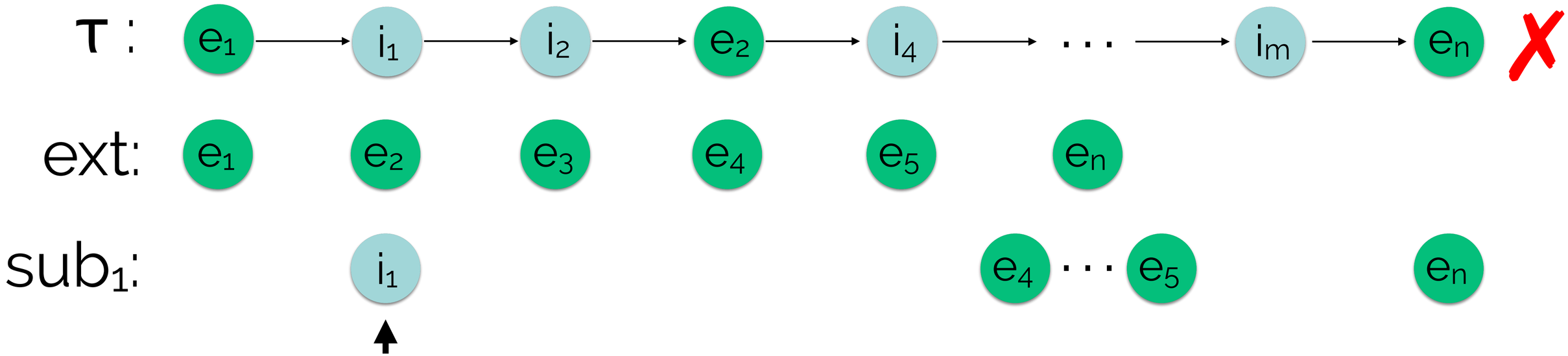

[1]A Zeller, R. Hildebrandt, “Simplifying and Isolating Failure-Inducing Input”, IEEE ‘02

# Observation #2: selectively mask original events



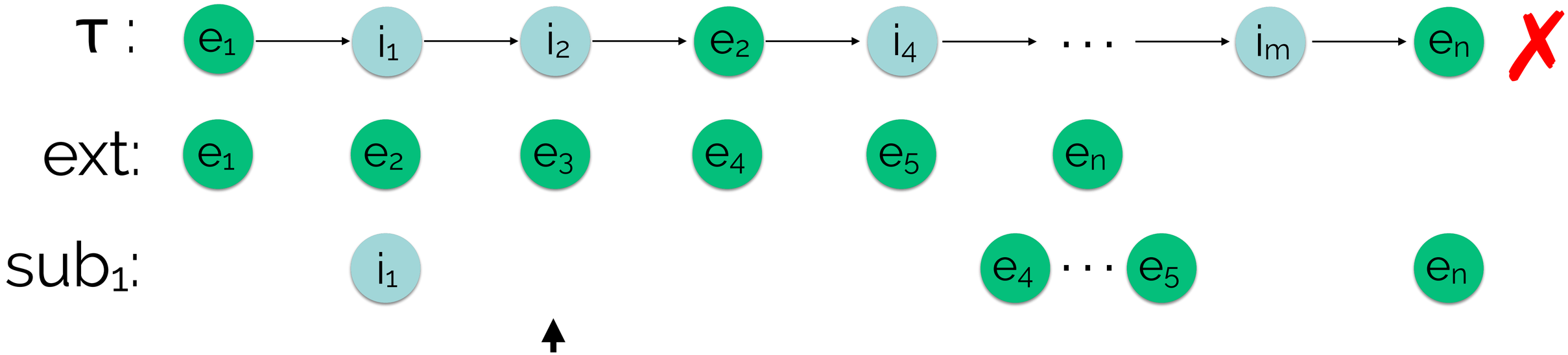

```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

# Observation #2: selectively mask original events



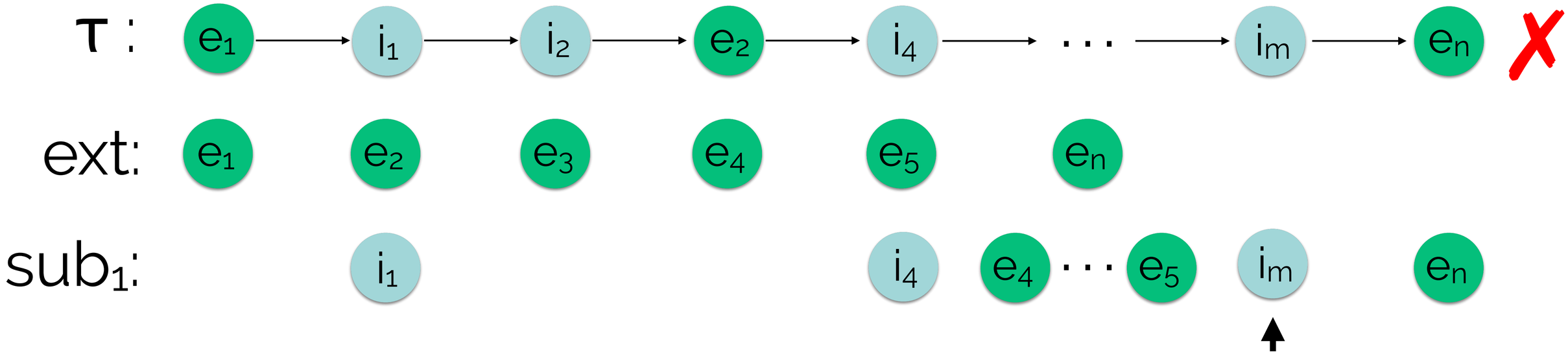

```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

# Observation #2: selectively mask original events



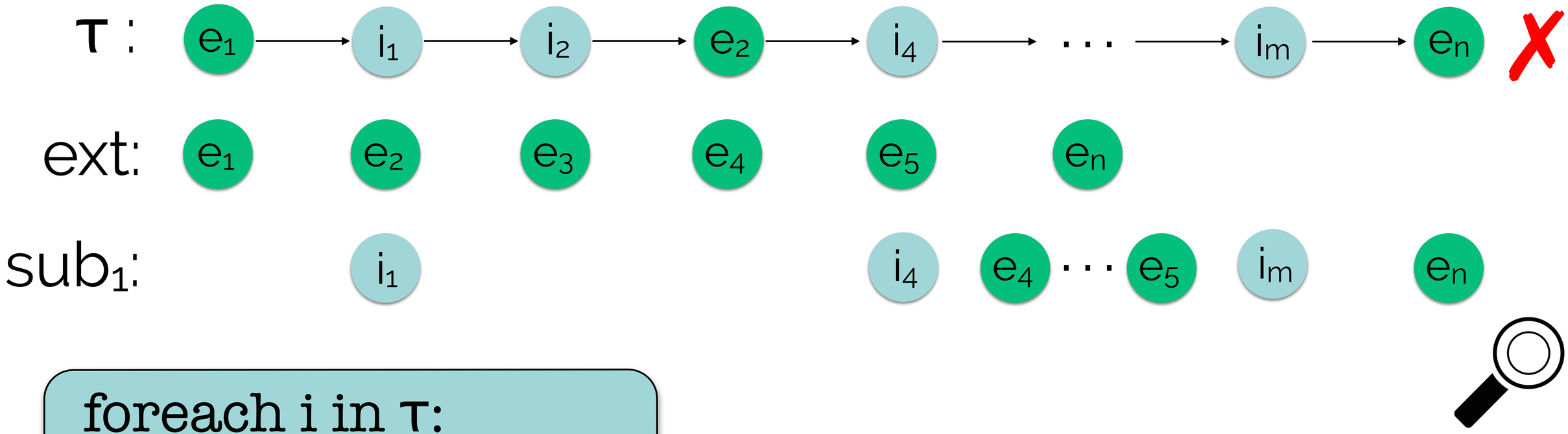

```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

# Observation #2: selectively mask original events




```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

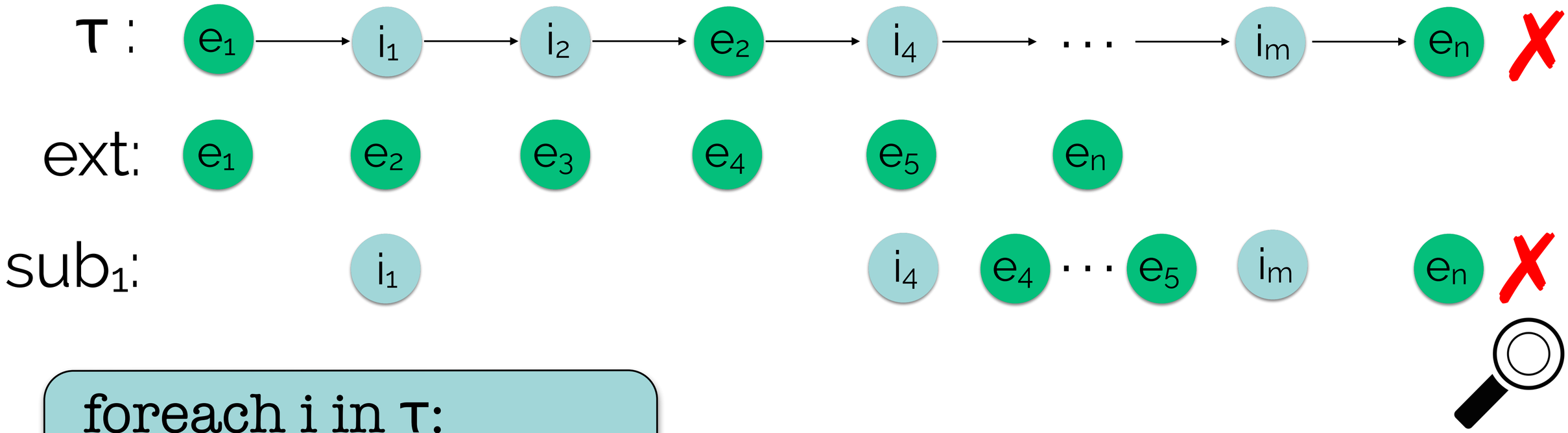
# Observation #2: selectively mask original events





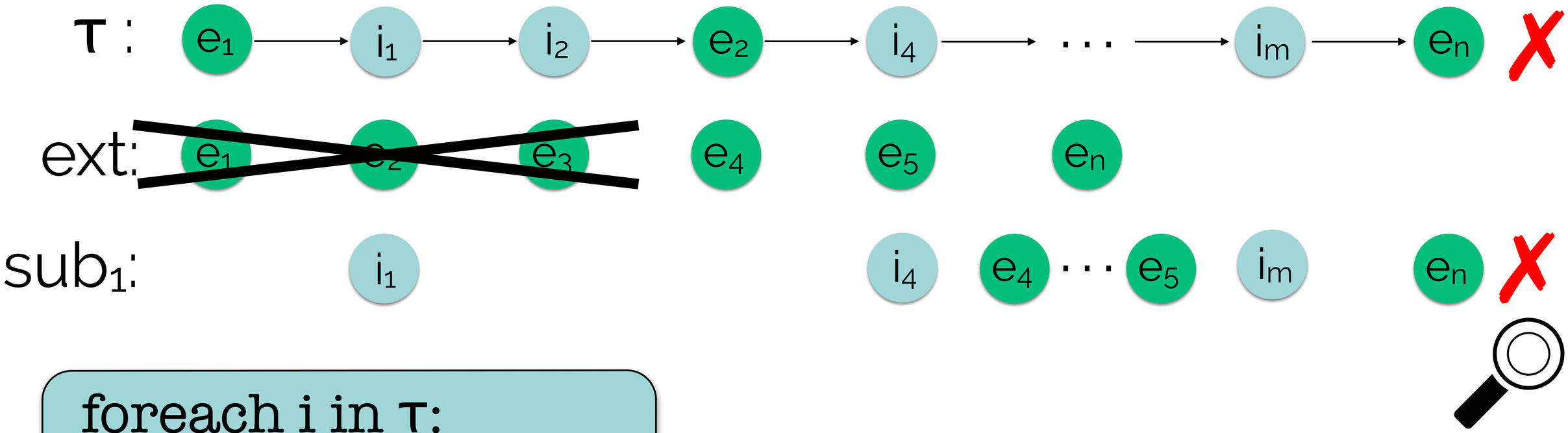

```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

# Observation #2: selectively mask original events



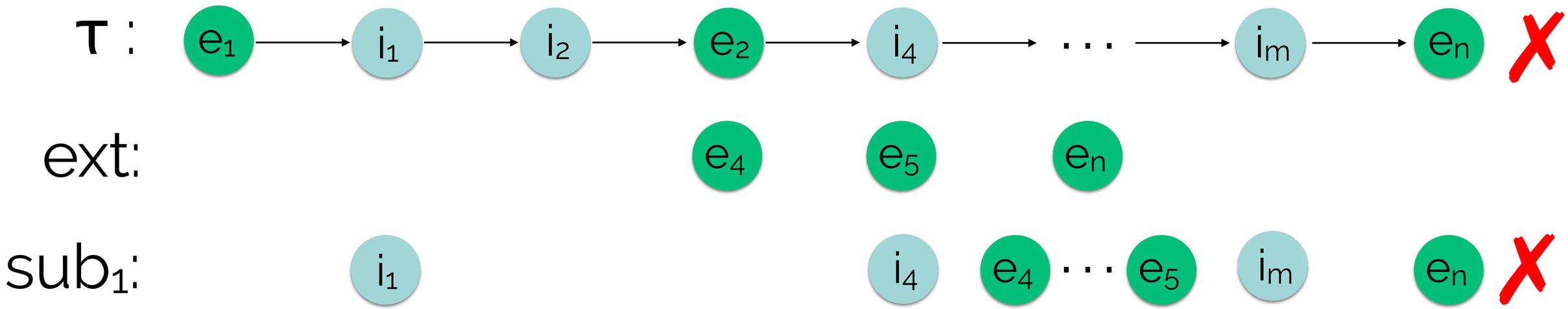

```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

# Observation #2: selectively mask original events

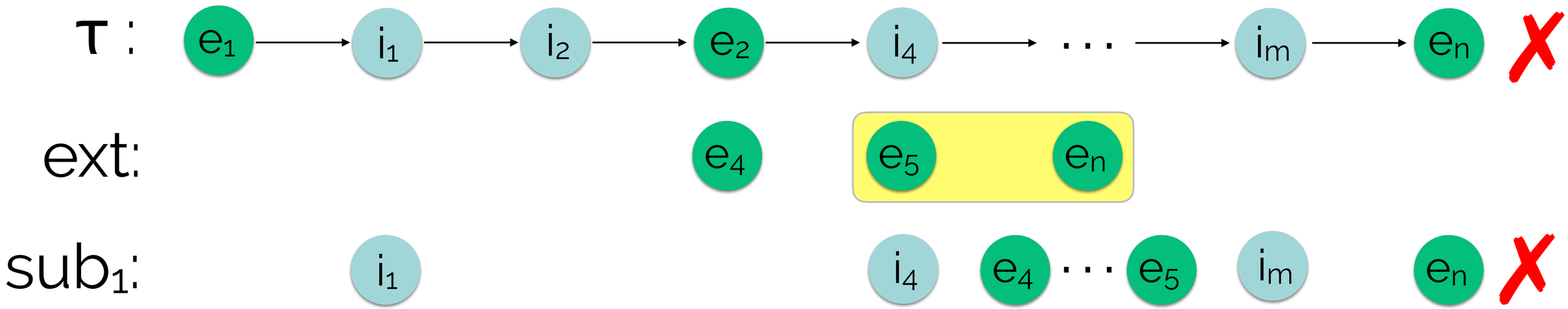



```
foreach i in τ:
   if i is pending:
     deliver i
# ignore unexpected
```

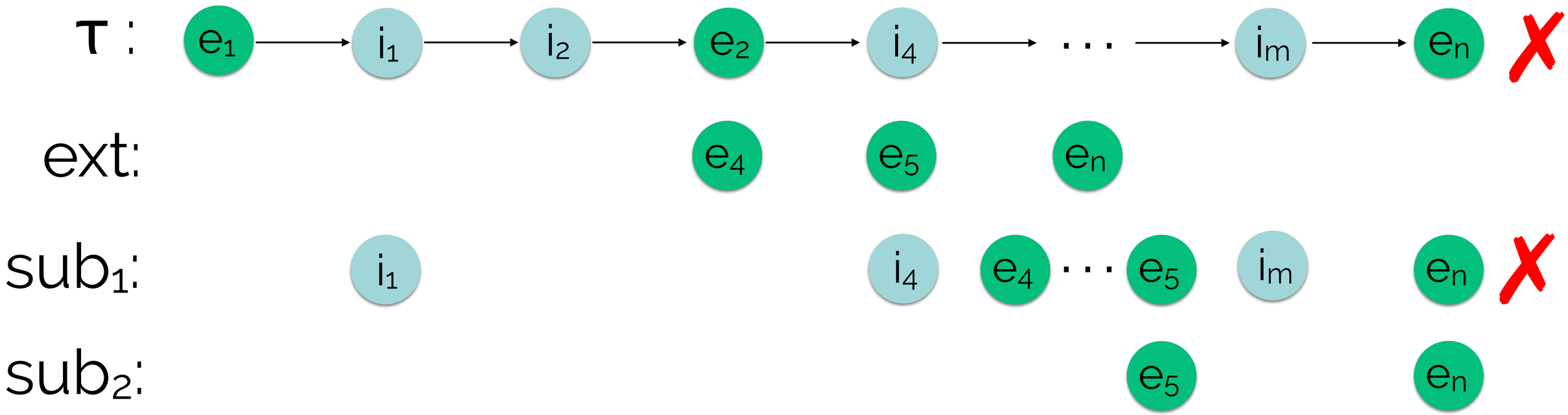
# Observation #2: selectively mask original events



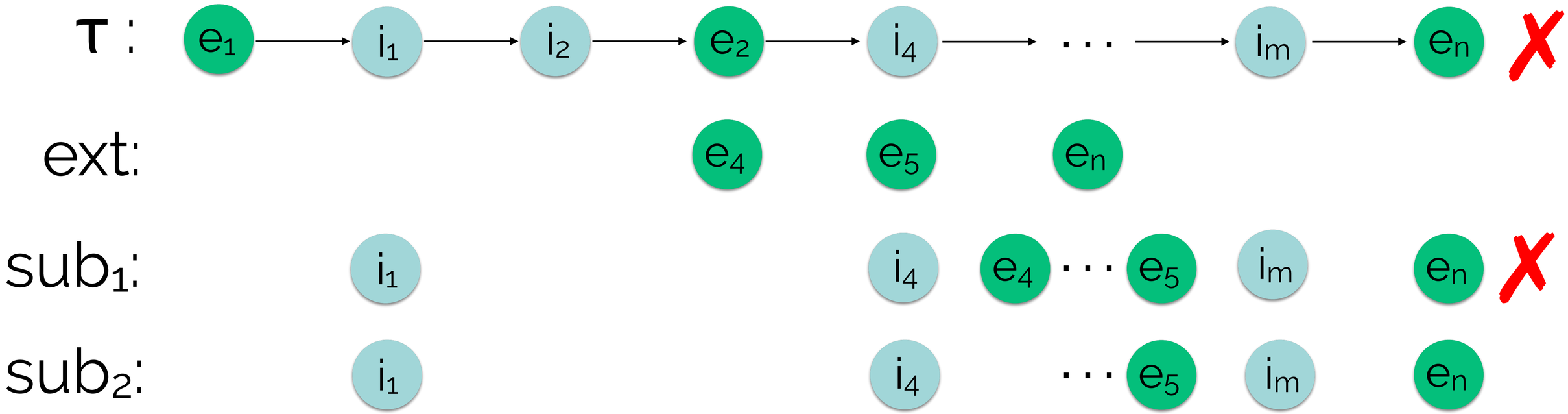
# Observation #2: selectively mask original events



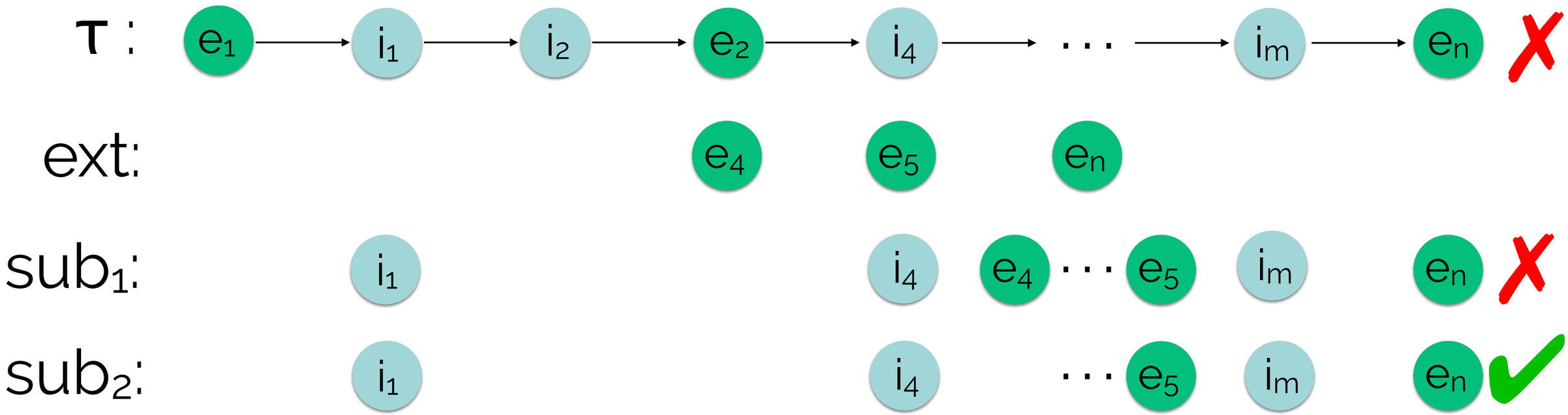
# Observation #2: selectively mask original events



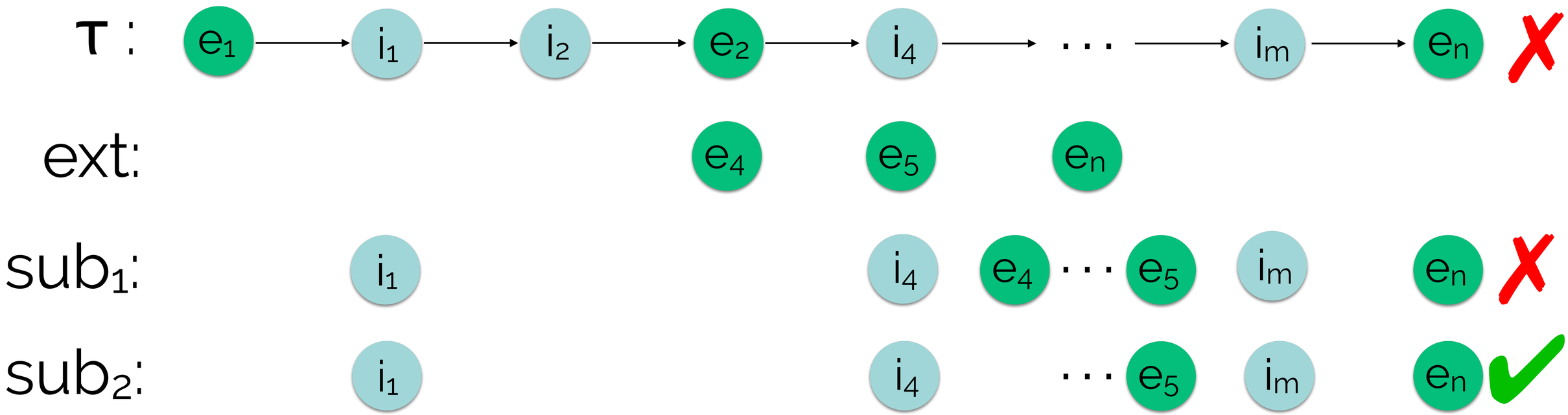
# Observation #2: selectively mask original events



# Observation #2: selectively mask original events



# Observation #2: selectively mask original events




Explore backtrack points until (i) ✗ or (ii) time budget for $sub_2$ expired

. . .

. . .

Goal: find minimal schedule that produces violation

Observation #1: many schedules are commutative

Approach: prioritize schedule space exploration

Observation #2: selectively mask original events

Observation #3: some contents should be masked

Observation #4: shrink external message contents

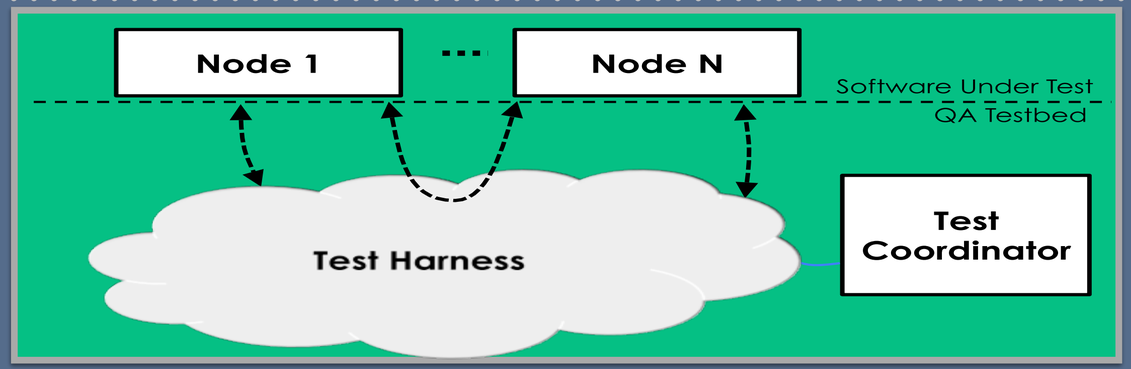
Minimize internal events after externals minimized

# Outline

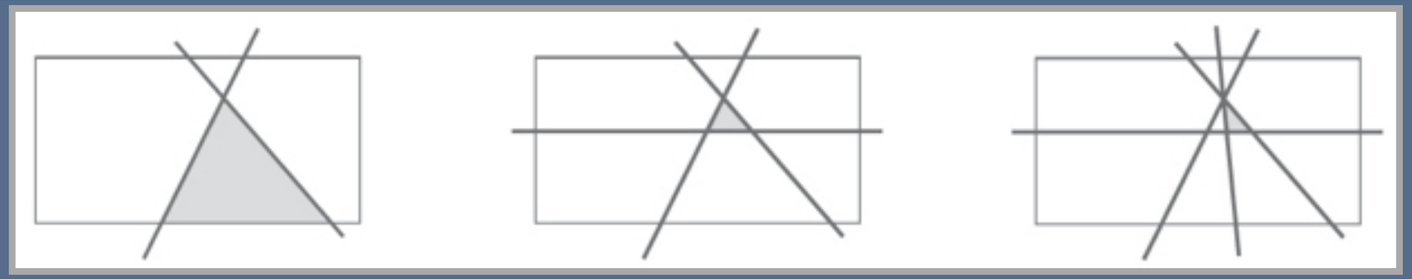
Introduction

Background

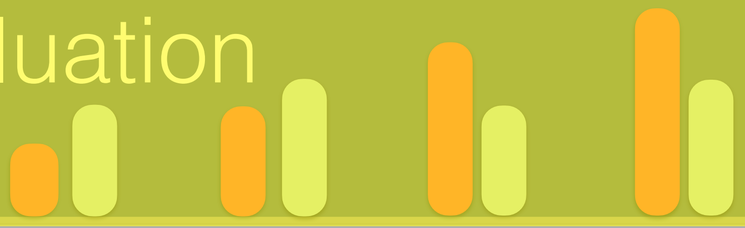
Randomized Testing with DEMi




Minimization



Evaluation



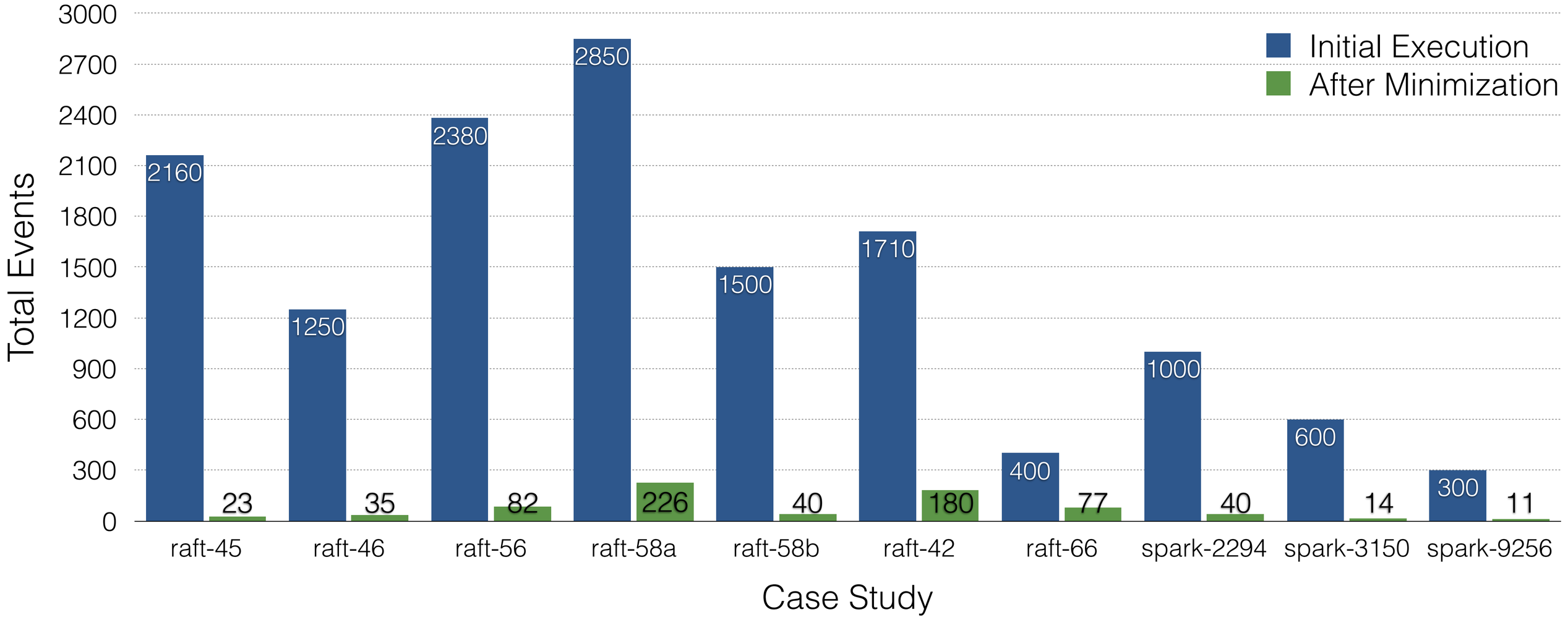
Conclusion

# Target Systems

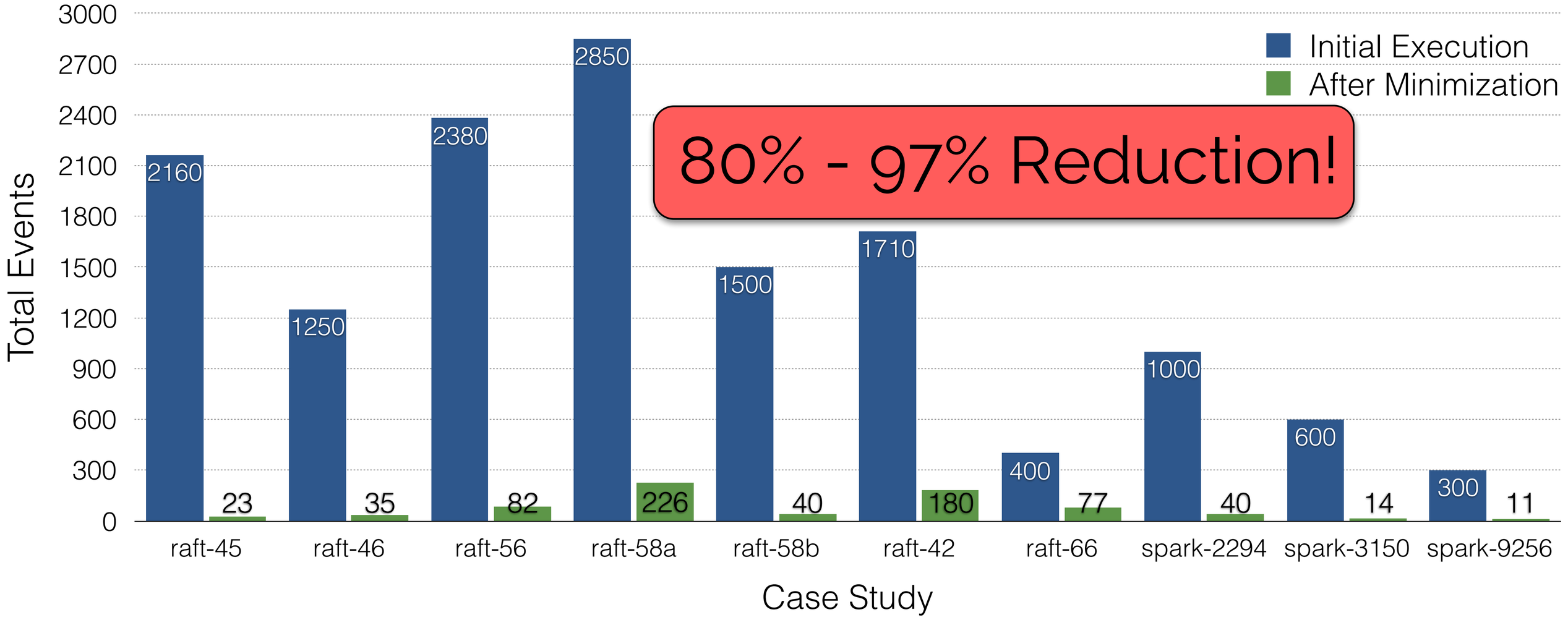




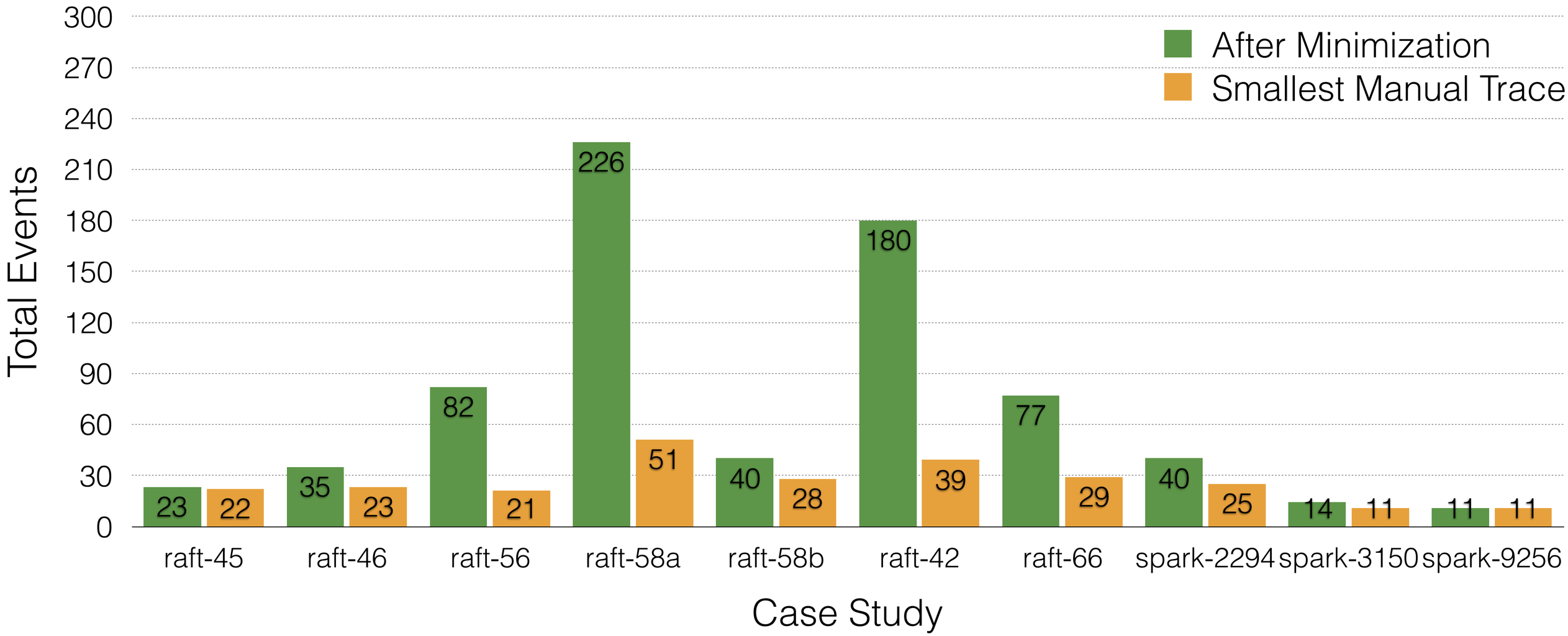

# How well does DEMi work?



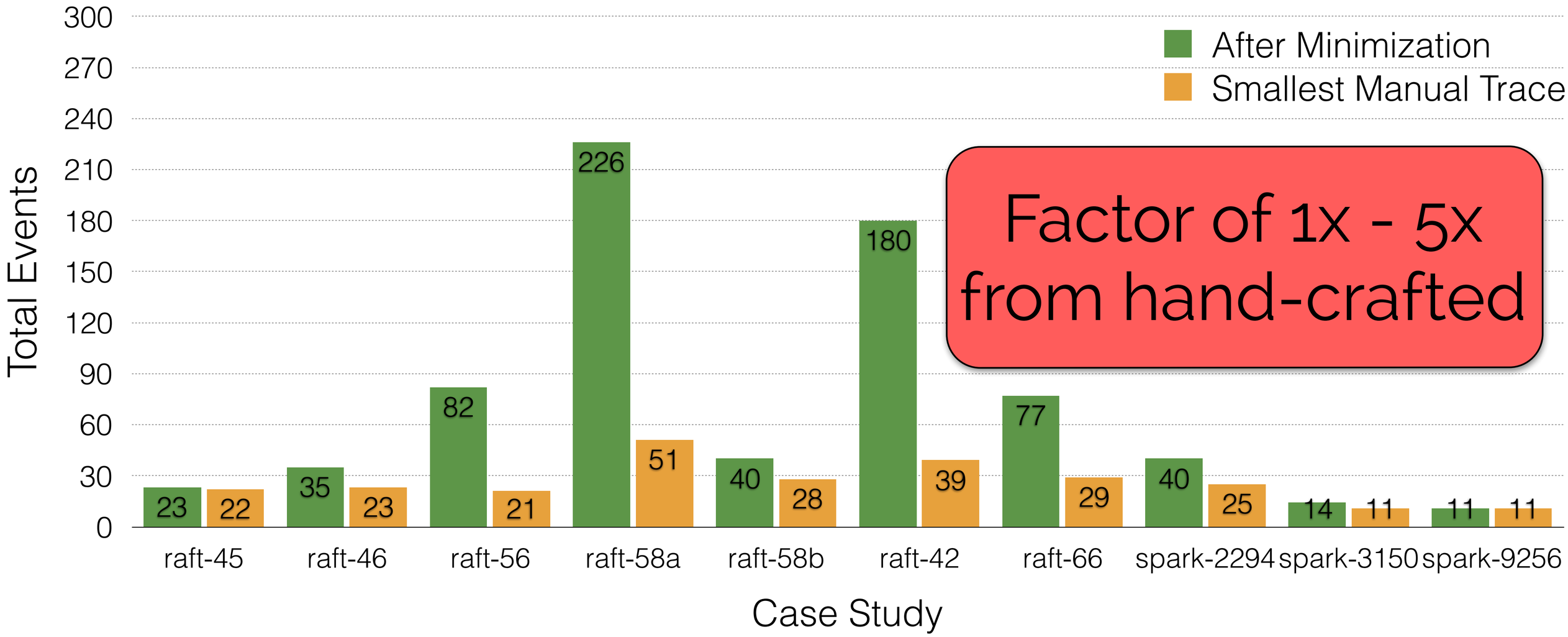
# How well does DEMi work?



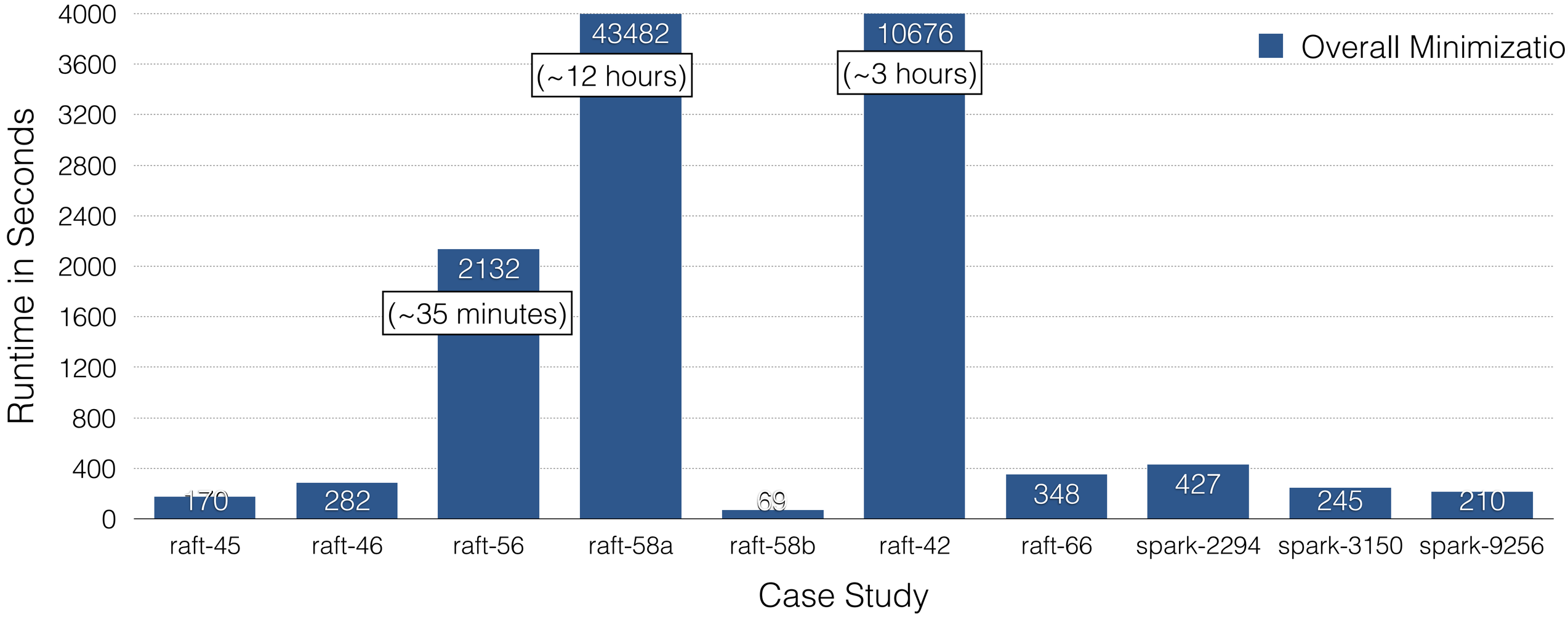
# How well does DEMi work?



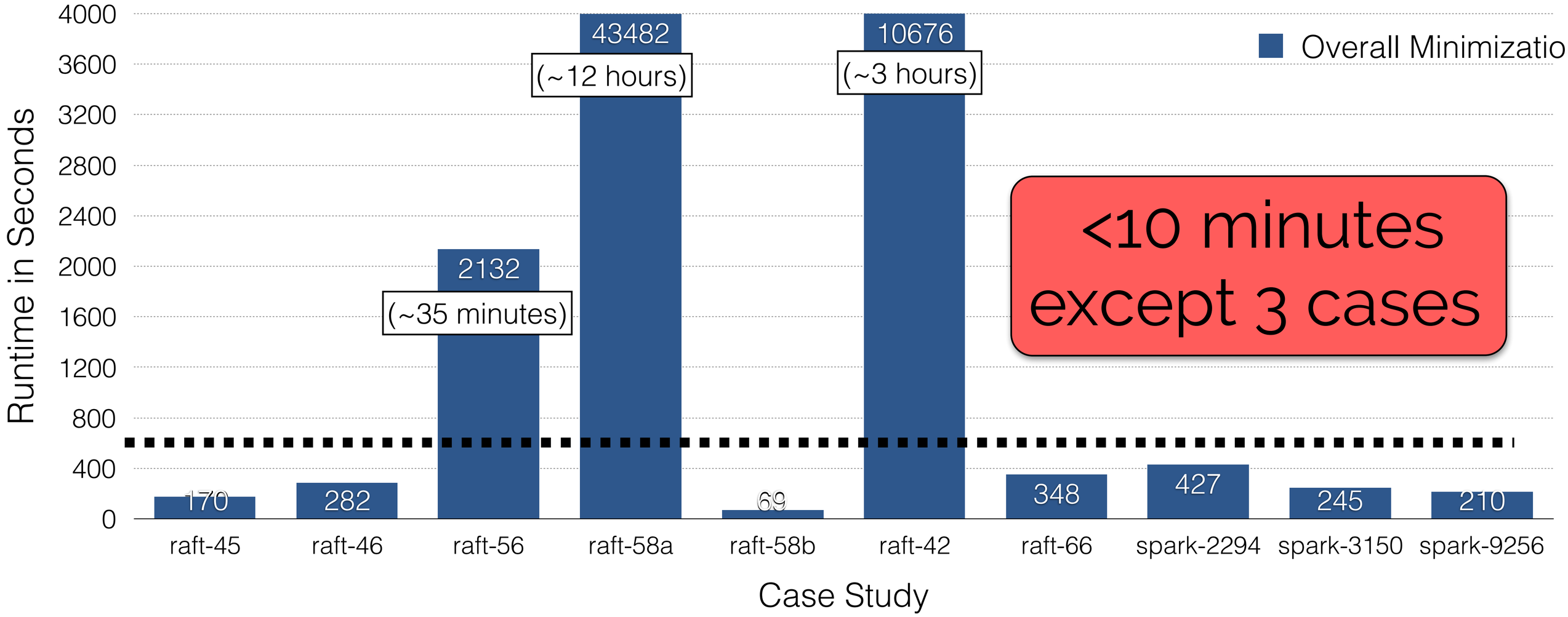
# How well does DEMi work?



# How quickly does DEMi work?



# How quickly does DEMi work?



# See the paper for…

- How we handle non-determinism
- Handling multithreaded processes
- Supporting other RPC libraries
- Sketch for minimizing production traces
- More in-depth evaluation
- Related work
- …

# Conclusion

Optimistic that these techniques can be successfully applied more broadly

Open source tool: github.com/NetSys/demi

Read our paper!
eecs.berkeley.edu/~rcs/research/nsdi16.pdf

Contact me! cs@cs.berkeley.edu

Thanks for your time!

# Attributions

Inspiration for slide design: Jay Lorch's IronFleet slides

Graphic Icons: thenounproject.org

logfile: mantisshrimpdesign

magnifying glass: Ricardo Moreira

disk: Anton Outkine

hook: Seb Cornelius

bug report: Lemon Liu

devil: Mourad Mokrane

Putin: Remi Mercier

# Production Traces

Model: feed partially ordered log into single machine DEMi

Require:

- Partial ordering of all message deliveries
- All crash-recoveries logged to disk

# Instrumentation Complexity

| | akka-raft | Spark |
|---|---|---|
| **Message Fingerprint** | 59 | 56 |
| **Non-Determinism** | 2 | ~250 |

**Table 4:** Instrumentation complexity (lines of code) needed to define message fingerprints, and to mitigate non-determinism.

# Related Work

- Thread Schedule Minimization
  - Isolating Failure-Inducing Thread Schedules. SIGSOFT '02.
  - A Trace Simplification Technique for Effective Debugging of Concurrent Programs. FSE '10.
- Program Flow Analysis.
  - Enabling Tracing of Long-Running Multithreaded Programs via Dynamic Execution Reduction. ISSTA '07.
  - Toward Generating Reducible Replay Logs. PLDI '11.
- Best-Effort Replay of Field Failures
  - A Technique for Enabling and Supporting Debugging of Field Failures. ICSE '07.
  - Triage: Diagnosing Production Run Failures at the User's Site. SOSP '07.

# DDmin in more detail

Input: $T_{\text{✘}}$ s.t. $T_{\text{✘}}$ is a trace and $\mathit{test}(T_{\text{✘}}) = \text{✘}$. Output: $T'_{\text{✘}} = \mathit{ddmin}(T_{\text{✘}})$ s.t. $T'_{\text{✘}} \subseteq T_{\text{✘}}$, $\mathit{test}(T'_{\text{✘}}) = \text{✘}$, and $T'_{\text{✘}}$ is minimal.

$$\mathit{ddmin}(T_{\text{✘}}) = \mathit{ddmin}_2(T_{\text{✘}}, \emptyset) \quad \text{where}$$

$$\mathit{ddmin}_2(T'_{\text{✘}}, R) = \begin{cases} T'_{\text{✘}} & \text{if } |T'_{\text{✘}}| = 1 \text{ (“base case”)} \\ \mathit{ddmin}_2(T_1, R) & \text{else if } \mathit{test}(T_1 \cup R) = \text{✘} \text{ (“in } T_1\text{”)} \\ \mathit{ddmin}_2(T_2, R) & \text{else if } \mathit{test}(T_2 \cup R) = \text{✘} \text{ (“in } T_2\text{”)} \\ \mathit{ddmin}_2(T_1, T_2 \cup R) \cup \mathit{ddmin}_2(T_2, T_1 \cup R) & \text{otherwise (“interference”)} \end{cases}$$

where $\mathit{test}(T)$ denotes the state of the system after executing the trace $T$, ✘ denotes an invariant violation, $T_1 \subset T'_{\text{✘}}$, $T_2 \subset T'_{\text{✘}}$, $T_1 \cup T_2 = T'_{\text{✘}}$, $T_1 \cap T_2 = \emptyset$, and $|T_1| \approx |T_2| \approx |T'_{\text{✘}}|/2$ hold.

# DDmin assumptions

- *Monotonic:*

$$P \oplus C = \chi \Rightarrow P \oplus (C \cup C') \neq \checkmark$$

- *Unambiguous:*

$$P \oplus C = \chi \wedge P \oplus C' = \chi \Rightarrow P \oplus (C \cap C') \neq \checkmark$$

- *Consistent*

$$P \oplus C \neq ?$$

# Local vs. global minima

**Definition 8 (Global minimum).** *A set* $c \subseteq c_\chi$ *is called the* global minimum *of* $c_\chi$ *if:* $\forall c' \subseteq c_\chi \cdot \big(|c'| < |c| \Rightarrow test(c') \neq \chi\big)$ *holds.*

**Definition 10 ($n$-minimal test case).** *A test case* $c \subseteq c_\chi$ *is* $n$-minimal *if:* $\forall c' \subset c \cdot |c| - |c'| \leq n \Rightarrow \big(test(c') \neq \chi\big)$ *holds. Consequently,* $c$ *is 1-minimal if* $\forall \delta_i \in c \cdot test\big(c - \{\delta_i\}\big) \neq \chi$ *holds.*

# Minimization Pace

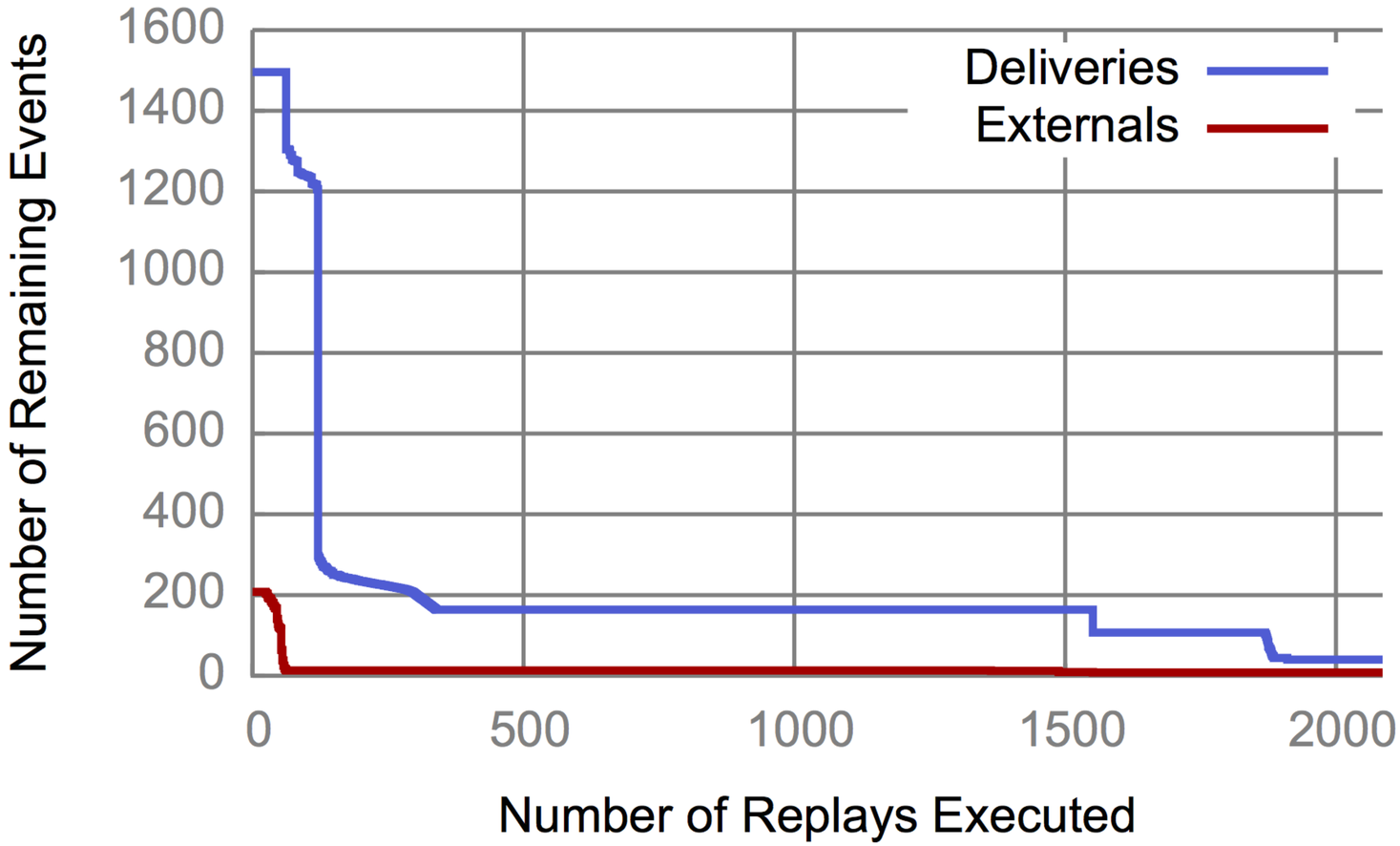



**Figure 2:** Minimization pace for raft-58b. Significant progress is made early on, then progress becomes rare.

# Dealing With Threads

If you're lucky: threads are largely independent (Spark)
If you're unlucky:  key insight:
A write to shared memory is equivalent to a message delivery

Approach:

- interpose on virtual memory, thread scheduler
- pause a thread whenever it writes to shared memory / disk

Cf. "Enabling Tracing Of Long-Running Multithreaded Programs Via Dynamic Execution Reduction", ISSTA '07

# Dealing With Non-Determinism

Interpose on:

- Timers
- Random number generators
- Unordered hash values
- ID allocation

Stop-gap: replay each schedule multiple times

# Complete Results

| Bug Name | Bug Type | Initial | Provenance | STSSched | TFB | Optimal | NoDiverge |
|---|---|---|---|---|---|---|---|
| raft-45 | Akka-FIFO, reproduced | 2160 (E:108) | 2138 (E:108) | 1183 (E:8) | 23 (E:8) | 22 (E:8) | 1826 (E:11) |
| raft-46 | Akka-FIFO, reproduced | 1250 (E:108) | 1243 (E:108) | 674 (E:8) | 35 (E:8) | 23 (E:6) | 896 (E:9) |
| raft-56 | Akka-FIFO, found | 2380 (E:108) | 2376 (E:108) | 1427 (E:8) | 82 (E:8) | 21 (E:8) | 2064 (E:9) |
| raft-58a | Akka-FIFO, found | 2850 (E:108) | 2824 (E:108) | 953 (E:32) | 226 (E:31) | 51 (E:11) | 2368 (E:35) |
| raft-58b | Akka-FIFO, found | 1500 (E:208) | 1496 (E:208) | 164 (E:13) | 40 (E:8) | 28 (E:8) | 1103 (E:13) |
| raft-42 | Akka-FIFO, reproduced | 1710 (E:208) | 1695 (E:208) | 1093 (E:39) | 180 (E:21) | 39 (E:16) | 1264 (E:43) |
| raft-66 | Akka-UDP, found | 400 (E:68) | 392 (E:68) | 262 (E:23) | 77 (E:15) | 29 (E:10) | 279 (E:23) |
| spark-2294 | Akka-FIFO, reproduced | 1000 (E:30) | 886 (E:30) | 43 (E:3) | 40 (E:3) | 25 (E:1) | 43 (E:3) |
| spark-3150 | Akka-FIFO, reproduced | 600 (E:20) | 536 (E:20) | 18 (E:3) | 14 (E:3) | 11 (E:3) | 18 (E:3) |
| spark-9256 | Akka-FIFO, found (rare) | 300 (E:20) | 256 (E:20) | 11 (E:1) | 11 (E:1) | 11 (E:1) | 11 (E:1) |

# Runtime Breakdown

| Bug Name | STSSched | TFB |
|---|---|---|
| raft-45 | 56s (594) | 114s (2854) |
| raft-46 | 73s (384) | 209s (4518) |
| raft-56 | 54s (524) | 2078s (31149) |
| raft-58a | 137s (624) | 43345s (834972) |
| raft-58b | 23s (340) | 31s (1747) |
| raft-42 | 118s (568) | 10558s (176517) |
| raft-66 | 14s (192) | 334s (10334) |
| spark-2294 | 330s (248) | 97s (78) |
| spark-3150 | 219s (174) | 26s (21) |
| spark-9256 | 96s (73) | 0s (0) |

# Integrating with other RPC libs

