University of Kent

Actors! And now?

An Implementer's Perspective on High-level Concurrency Models, Debugging Tools, and the Future of Automatic Bug Mitigation

THE ROYAL SOCIETY

UKRI Engineering and Physical Sciences Research Council

Stefan Marr

17 October 2021

Got a Question?
Feel free to interrupt me!

We're Looking for a Postdoc!

Job Ad

Project CaMELot: Catch and Mitigate Event-Loop Concurrency Issues

Please get in touch!

University of Kent

https://stefan-marr.de/2021/02/open-postdoc-position-on-language-implementation-and-concurrency/

Outcomes of Project MetaConc and work by

VRIJE UNIVERSITEIT BRUSSEL

C. Torres Lopez

D. Aumayr

JOHANNES KEPLER UNIVERSITY LINZ

Research Foundation Flanders
Opening new horizons

E. Gonzalez Boix

H. Mössenböck

FWF Der Wissenschaftsfonds.

Actors! What are Actors?

43 Years of Actors: A Taxonomy of Actor Models and Their Key Properties

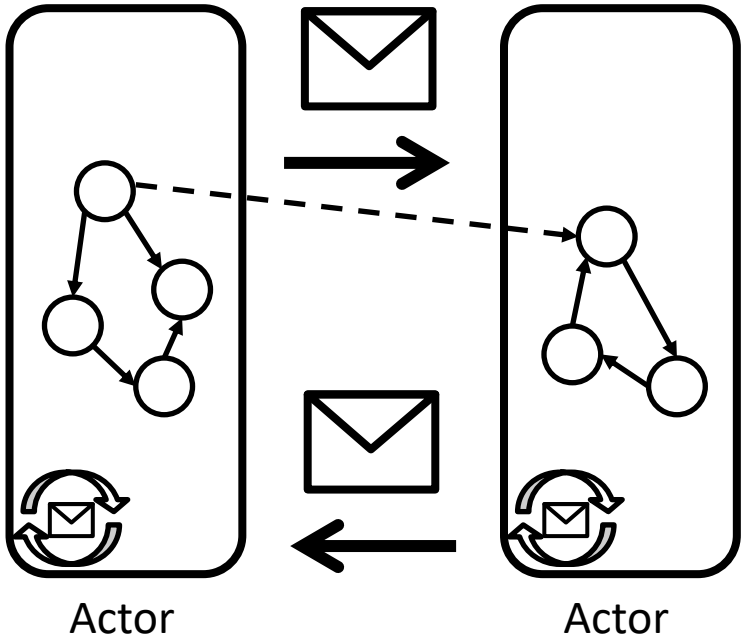
Joeri De Koster
Vriie Universiteit Brussel

Tom Van Cutsem
Nokia Bell Labs

Wolfgang De Meuter
Vriie Universiteit Brussel

- Many different variants
- For the 50 Years' Edition:
  - Which model is good for what?
    - Suitable problems/applications
    - Unsuitable problems per model
  - …

Communicating Event Loops



Concurrency Bugs are Common in Event Loop Systems

node.js	53 projects, 57 issues 12 projects, 1000 potential issues	2 studies
akka	12 projects 53 concurrency issues	1 study
HTML 5 / JS	Websites in top 500 ≈1-10 concurrency issues pe	
android	8-27 apps ≈2-20 concurrency issues pe	
C	6 projects 35 known event races	1 study

How to get rid of all these bugs?

Perhaps not a way to get rid of them all, but at least to make it easier

DEBUGGING ACTORS WITH SUITABLE BREAKPOINTS/STEPPING

Actor Breakpoints/Stepping

```
prom := aResult <-: get.

prom whenResolved: [:r |
  r println
].
```

Actor A

Actor Breakpoints/Stepping

msg send
msg receive
promise resolver
promise resolution

```
prom := aResult <-: get.

prom whenResolved: [:r |
  r println
].
```

1

Actor A

Actor Breakpoints/Stepping

msg send
msg receive
promise resolver
promise resolution

```
prom := aResult <-: get.

prom whenResolved: [:r |
  r println
].
```

Actor A

```
class Result = ()(

  public get = (
  ②| result |
    result := 42.
    ^ result
  )

)
```

Actor B

Actor Breakpoints/Stepping

msg send
msg receive
promise resolver
promise resolution

```
prom := aResult <-: get.

prom whenResolved: [:r |
  r println
].
```

Actor A

```
class Result = ()(

  public get = (
    | result |
    result := 42.
  ③ ^ result
  )

)
```

Actor B

Actor Breakpoints/Stepping

msg send
msg receive
promise resolver
promise resolution

```
prom := aResult <-: get.

prom whenResolved: [:r |④
  r println
].
```

Actor A

```
class Result = ()(

  public get = (
    | result |
    result := 42.
    ^ result
  )

)
```

Actor B

Actor Breakpoints/Stepping

```
prom := aResult <-: get.

prom whenResolved: [:r |
  r println
].
```

Actor A

```
class Result = ()(
```

before async

after async

```
  public get = (
  ①| result |
    result := 42.
  ②^ result
  )

)
```

Actor B

Actor Breakpoints/Stepping

```
prom := aResult <-: get.
```

promise resolver

promise resolution

```
prom whenResolved: [:r |
①r println
].
```

Actor A

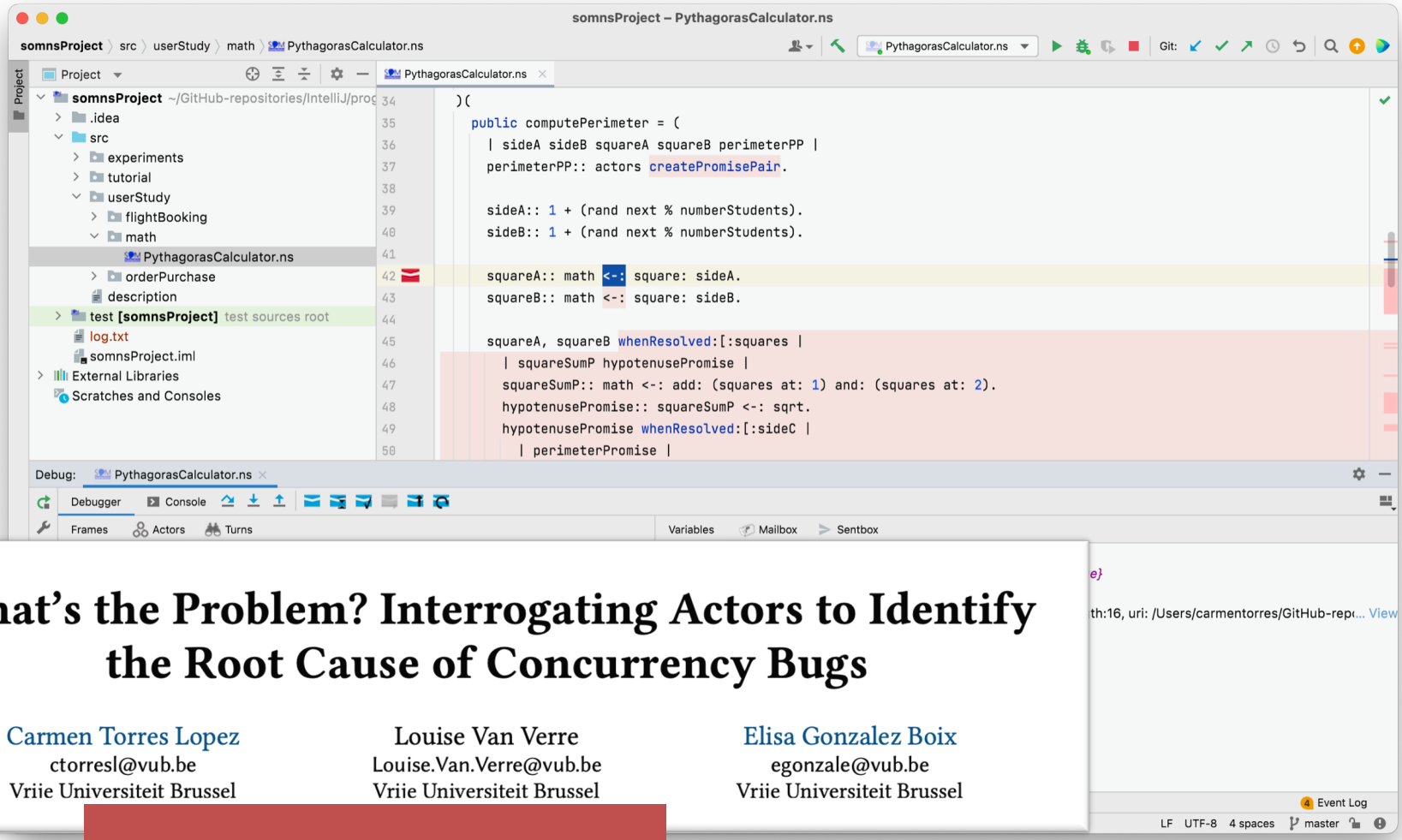
```
class Result = ()(

  public get = (
    | result |
    result := 42.
    ^ result
  )

)
```

Actor B

Apgar: A Debugger Made for Actor Programs



What's the Problem? Interrogating Actors to Identify the Root Cause of Concurrency Bugs

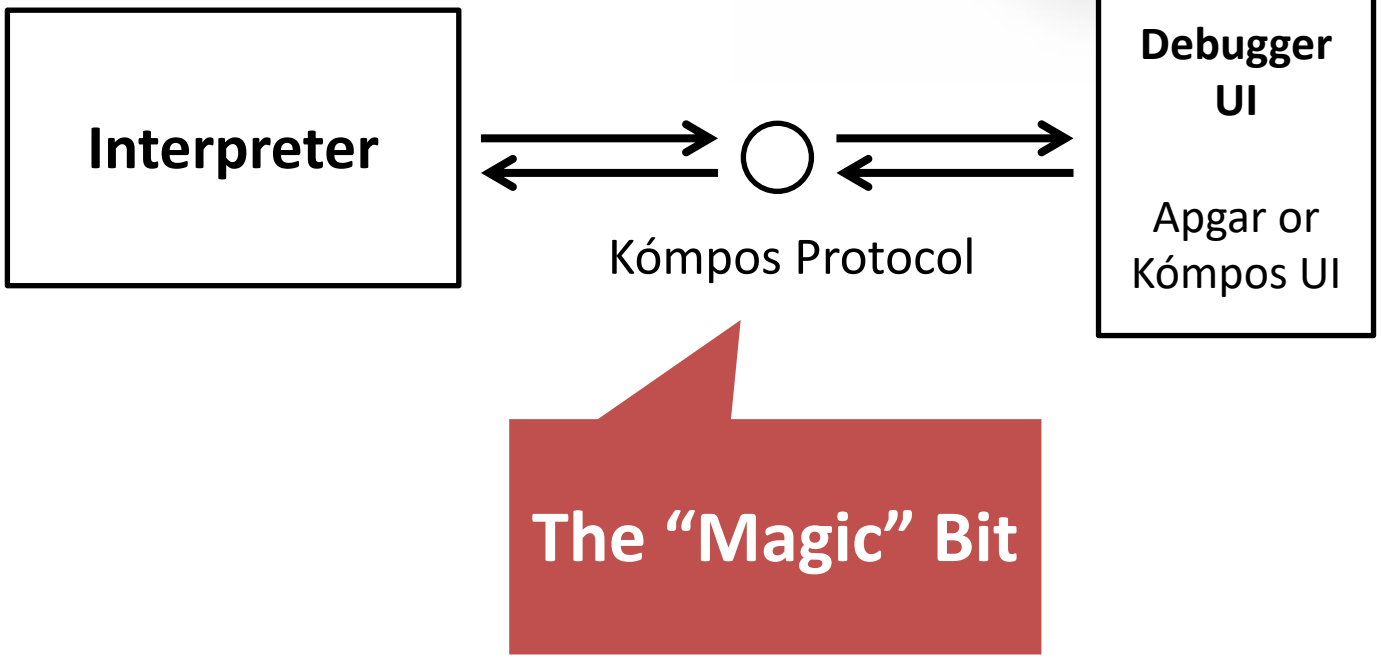
Carmen Torres Lopez
ctorresl@vub.be
Vriie Universiteit Brussel

Louise Van Verre
Louise.Van.Verre@vub.be
Vriie Universiteit Brussel

Elisa Gonzalez Boix
egonzale@vub.be
Vriie Universiteit Brussel

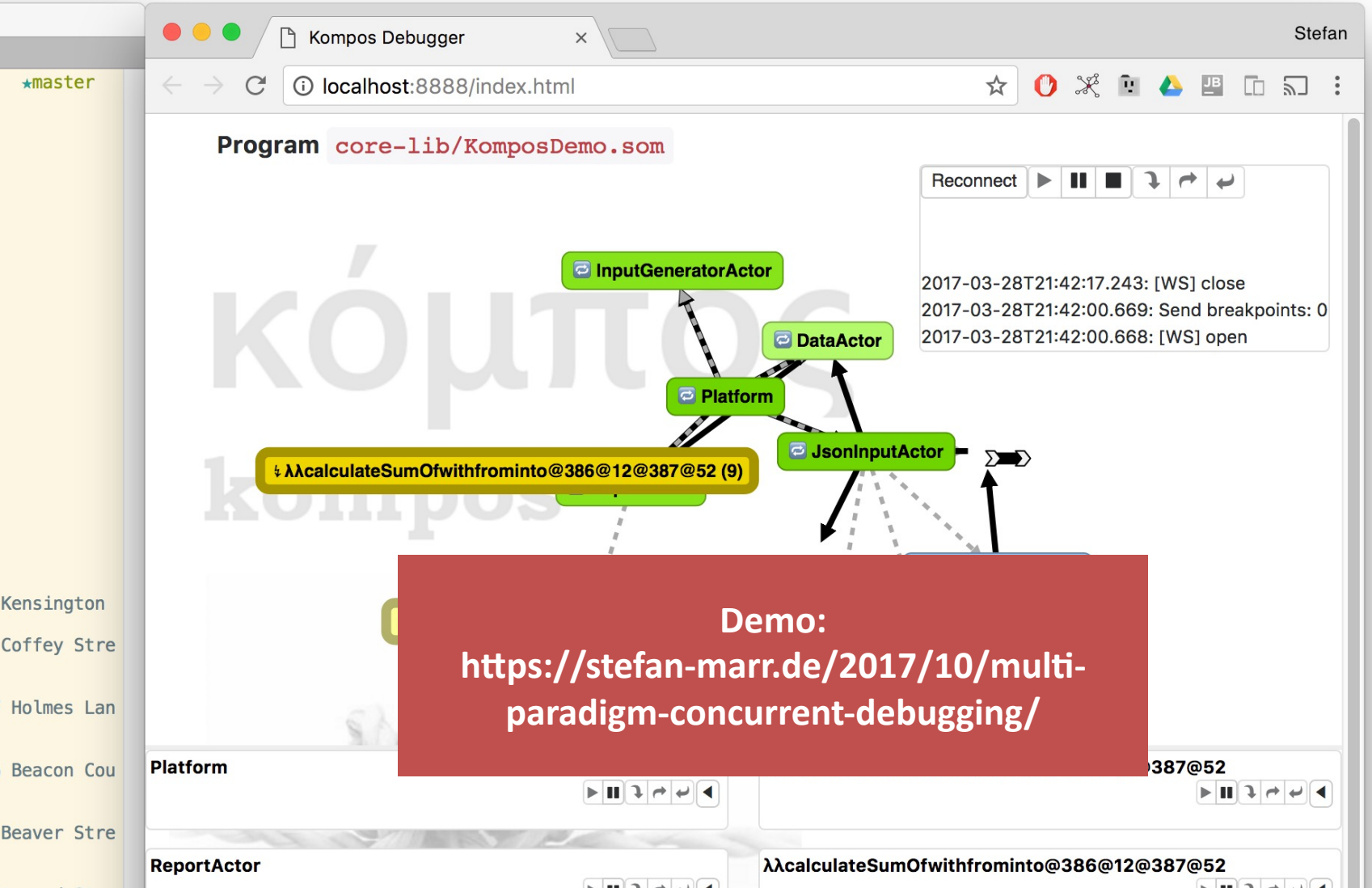
Carmen's presentation is in about 5.5h here at AGERE

Kómpos Architecture



https://stefan-marr.de/papers/dls-marr-et-al-concurrency-agnostic-protocol-for-debugging/

The Kómpos Debugger



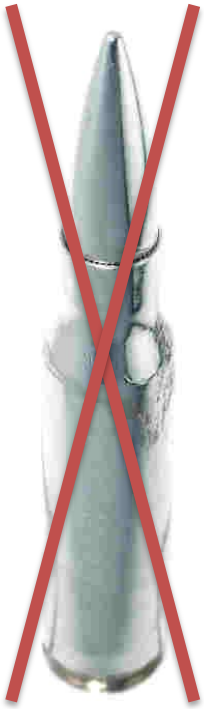
Demo:
https://stefan-marr.de/2017/10/multi-paradigm-concurrent-debugging/

Even with better debuggers, we'll still have concurrency bugs in our actor systems…

Maybe, just maybe!

Maybe Actors aren’t the best choice for every problem?

Maybe there are no Silver Bullets?

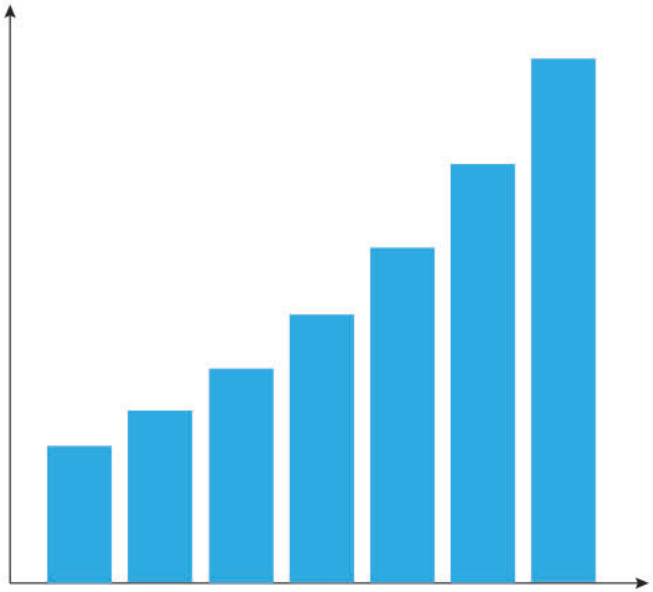


- **Actors**
- **CSP**
- **Locks, Monitors, …**
- **Fork/Join**
- **Transactional Memory**
- **Data Flow**

…

Building an Online Sales-Data Processor

```
{"item":     "beer",
 "price":    5.5,
 "quantity": 344,
 "customer": "<Prog>",
 "address":  "Pleinlaan 2"}
```



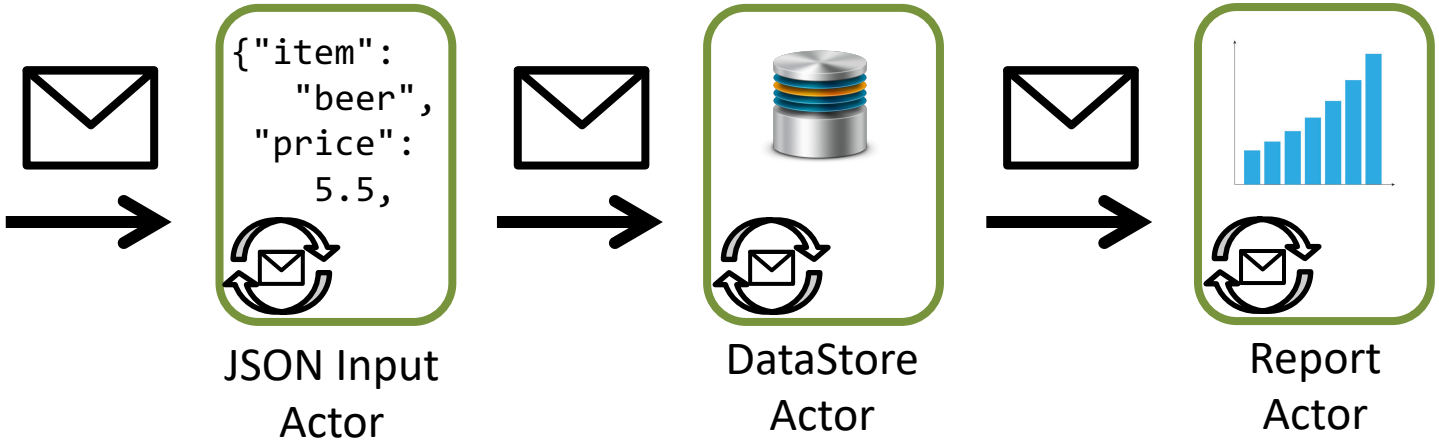
Stream of Sales Events

- Track revenue
- Report sales revenue over time

Subsystems as Asynchronous Activities

Use Actors as Main Abstraction

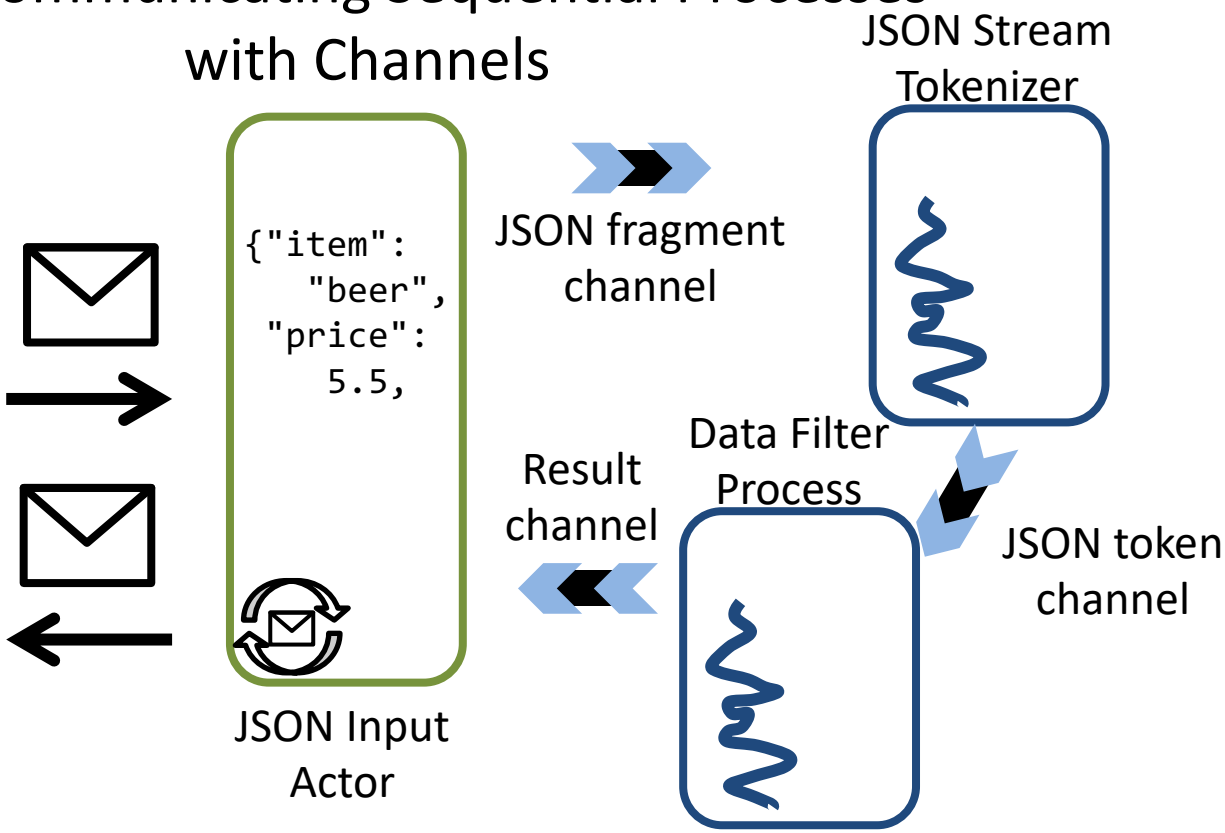
Event-Loop Model fits UI and System Paradigms



Parallelize JSON Processing

Using Communicating Sequential Processes with Channels

- Strict consumer/ producer relationship
- Allow for pipeline parallelism

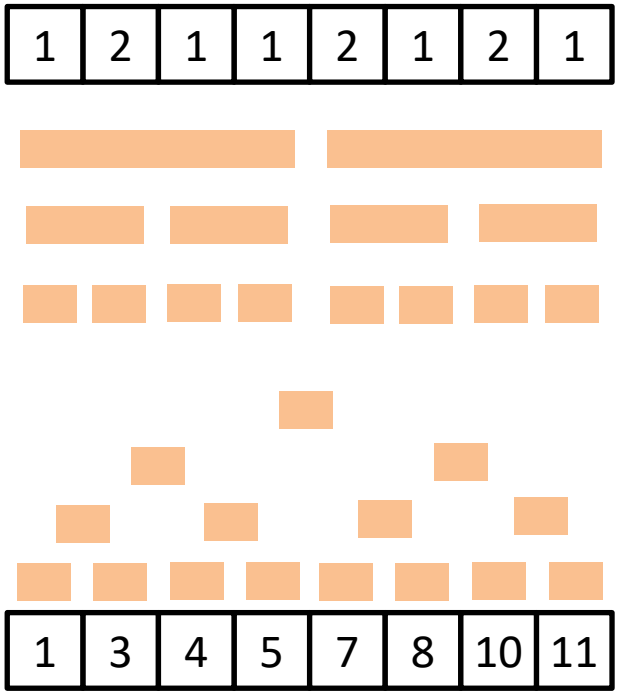


Sales Revenue Over Time based on Large Data Array

Report Actor

Construct Sum Tree
in parallel

Calculate Prefix Sum
in parallel



Parallel Prefix Sum Calculation with fork/join parallelism

How to build debuggers to support all the Concurrency Models?

κόμπος

kompos

Κόμπος: A PLATFORM FOR DEBUGGING COMPLEX CONCURRENT APPLICATIONS

The Kómpos Debugger

A Concurrency-Agnostic Protocol for Multi-Paradigm Concurrent Debugging Tools

Stefan Marr
Johannes Kepler University
Linz, Austria
stefan.marr@jku.at

Carmen Torres Lopez
Vrije Universiteit Brussel
Brussels, Belgium
ctorresl@vub.be

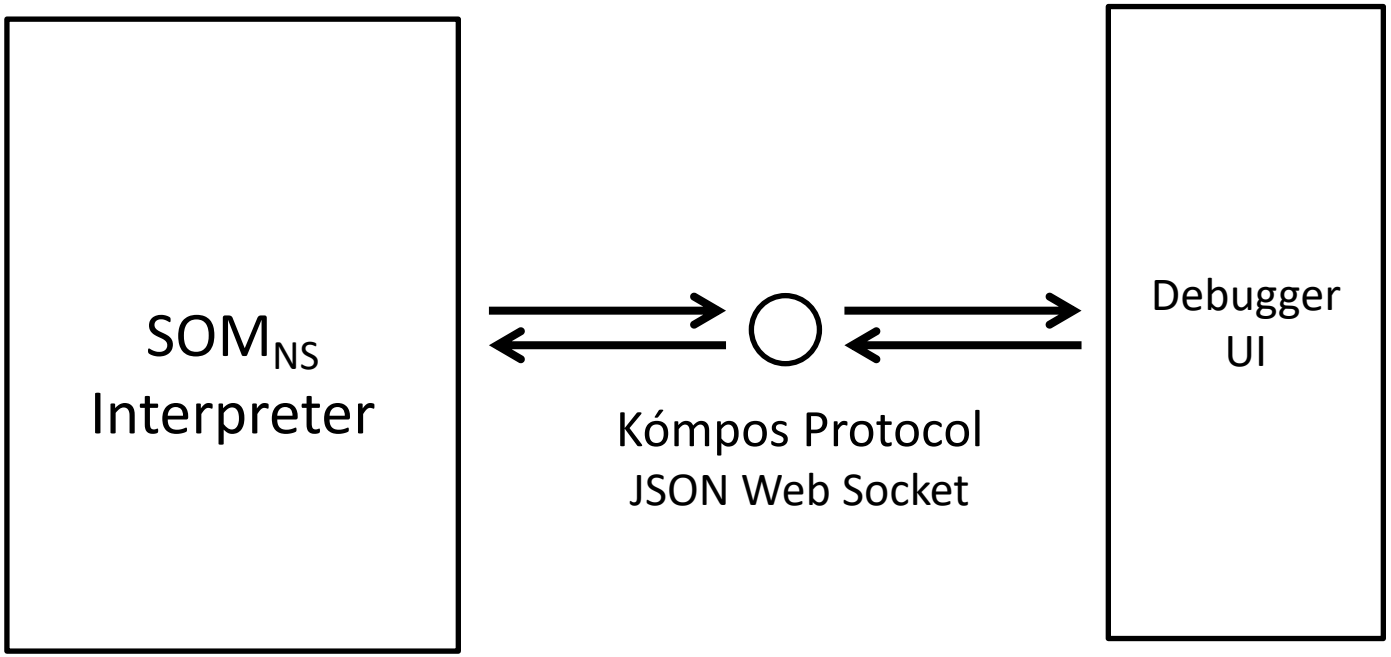
Dominik Aumayr
Johannes Kepler University
Linz, Austria
dominik.aumayr@jku.at

Elisa Gonzalez Boix
Vrije Universiteit Brussel
Brussels, Belgium
egonzale@vub.be

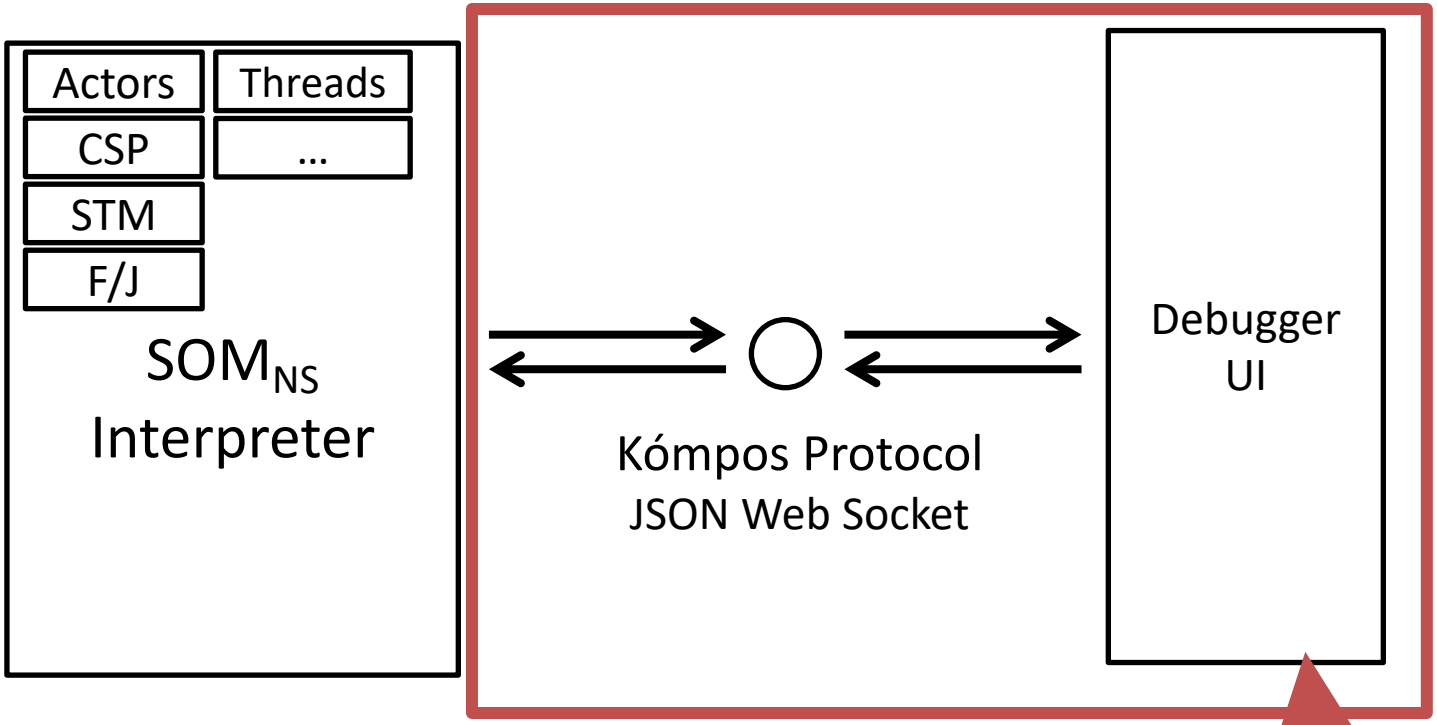
Hanspeter Mössenböck
Johannes Kepler University
Linz, Austria
hanspeter.moessenboeck@jku.at

https://stefan-marr.de/papers/dls-marr-et-al-concurrency-agnostic-protocol-for-debugging/

Kómpos Architecture



Kómpos Architecture

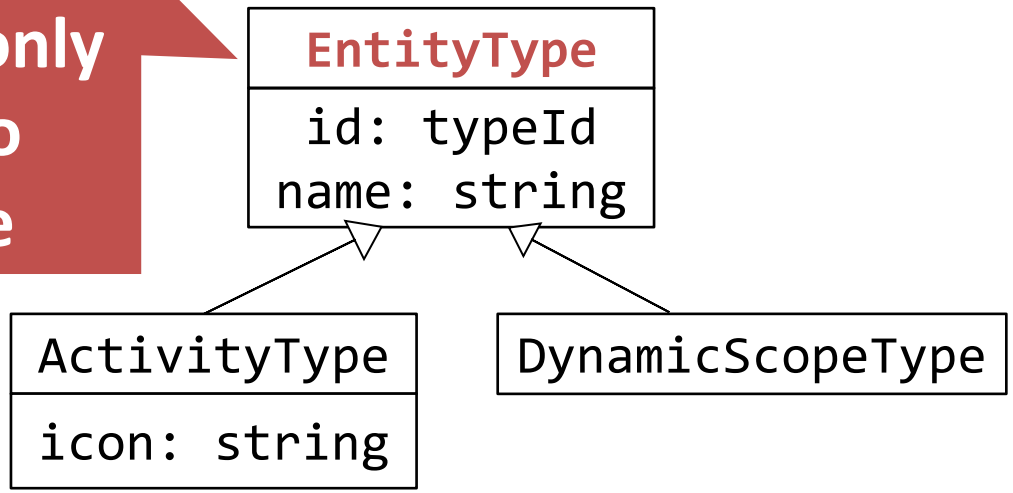


Agnostic of Concurrency Models

And we have two UIs! Apgar & Kómpos UI

Kómpos Protocol Metadata

Concurrency semantics only known to language



BreakpointType

name: string
label: string
applicableTo: Tag[]

SteppingType

name: string
label: string
applicableTo: Tag[]
activities: ActivityType[]
scopes: DynamicScopeType[]

Kómpos Protocol Messages

SetBreakpoint
location: Coord type: BreakpointType

DoStep
activityId: id type: SteppingType

Debugger UI just "lists" available types

Stopped
activityId: id location: Coord actType: ActivityType scopes: DynamicScopeType[]

A Model-Agnostic Debugger: Example Channel Breakpoints

```
channel out
1 write: 42 4
```

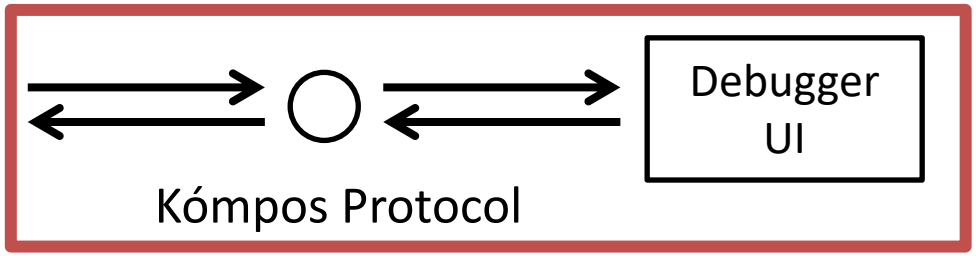
Process A

```
channel in
3 read 2
```

ss B

"just" source locations and ids! UI doesn't need to know these concepts!

Debuggers can be Great for High-level Concurrency Models!



promise resolver
promise resolution

```
prom whenResolved: [:r |
  r println ].
```

Make tools agnostic

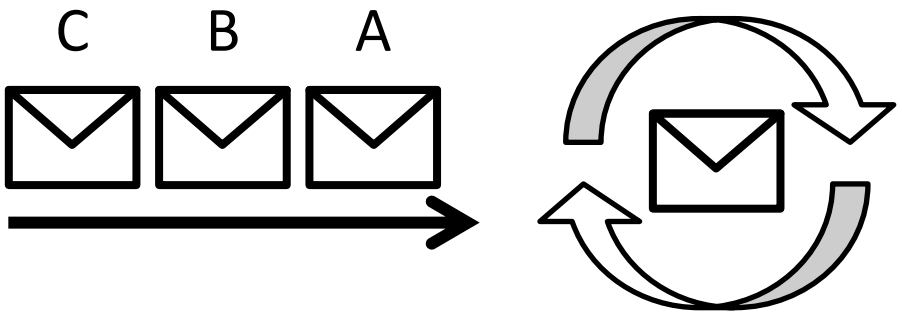
Offer the Key Features as Breakpoints/Steps

Reproduces only 1 in 10? How can I fix such a bug???

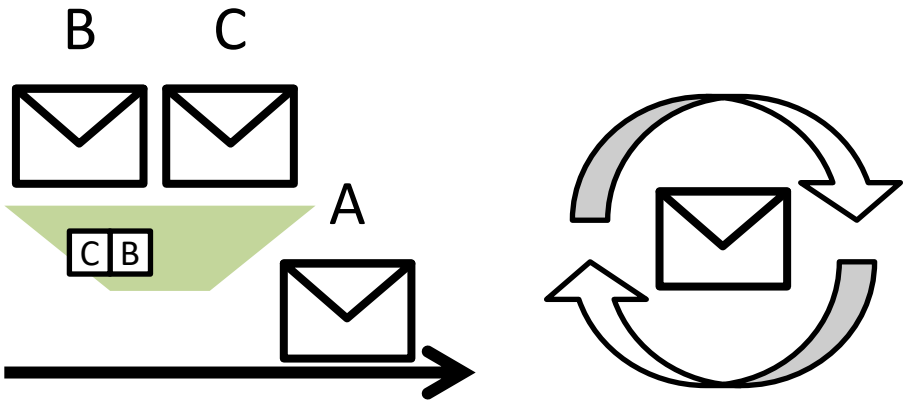
NON-DETERMINISM MAKES FOR UNHAPPY DEBUGGERS

One Solution: Record & Replay

- Record
  event order



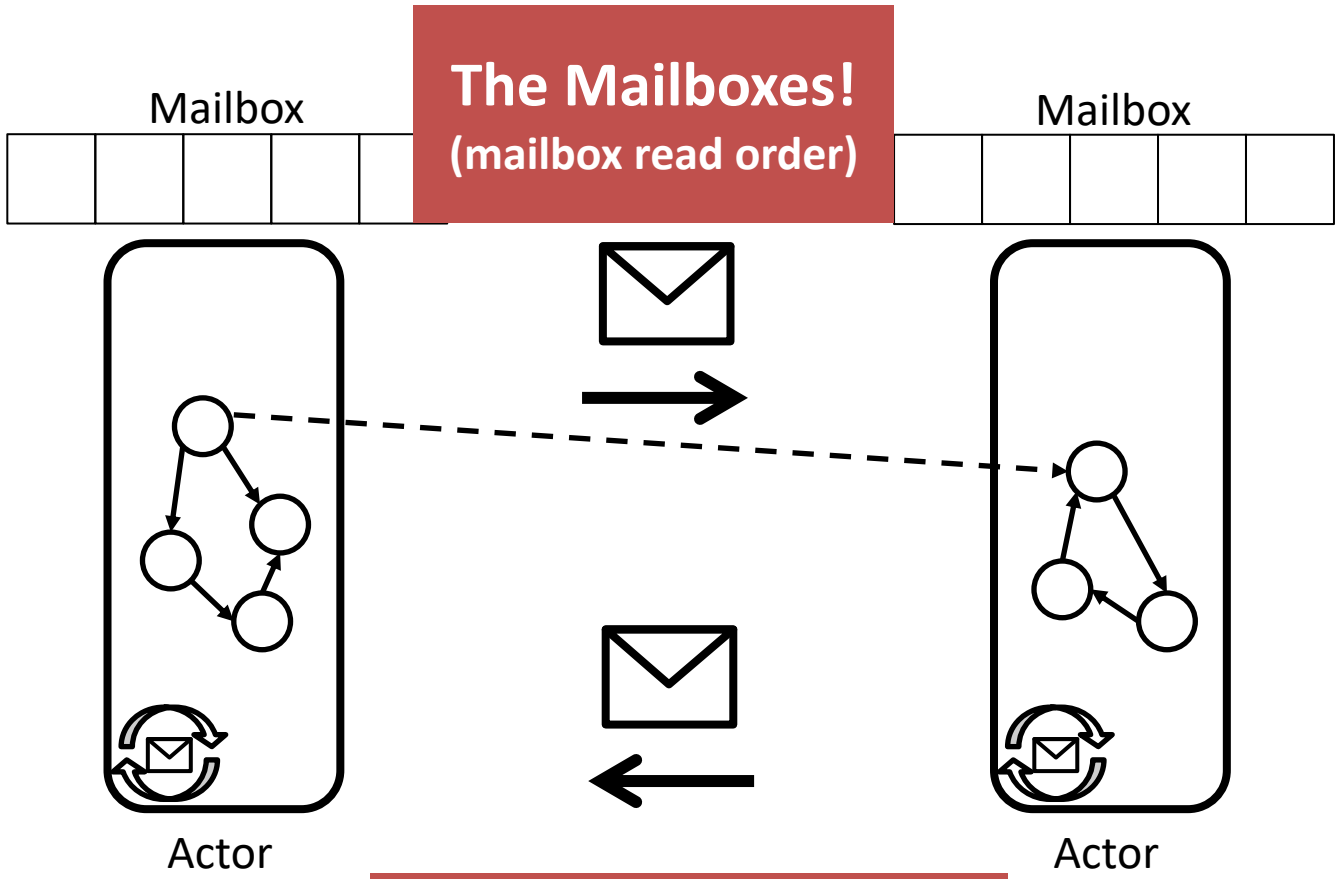
- Replay
  reorder to fit



Capturing High-level Nondeterminism in Concurrent Programs for Practical Concurrency
Model Agnostic Record & Replay D. Aumayr et al. The Art, Science, and Engineering of Programming, **Programming**, 2021.
Efficient and Deterministic Record & Replay for Actor Languages D. Aumayr et al. Proceedings of the 15th International Conference on Managed Languages and Runtimes, **ManLang'18.**

How is that going to work agnostic to concurrency models?

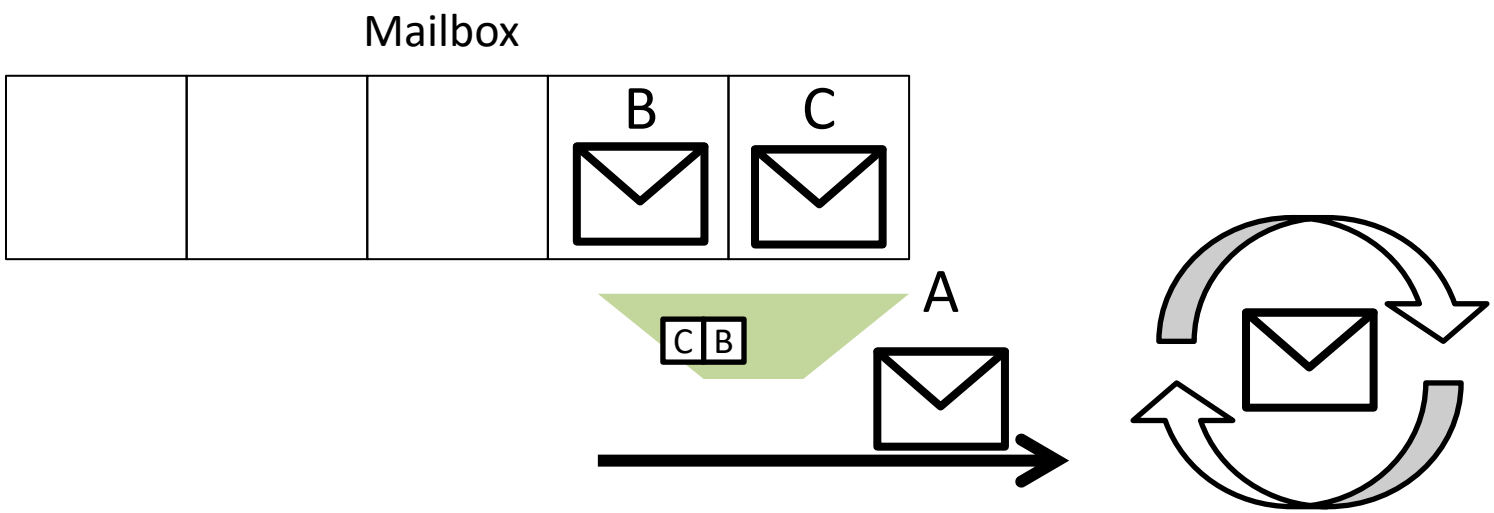
Looking at Communicating Event Loops



The Mailboxes!
(mailbox read order)

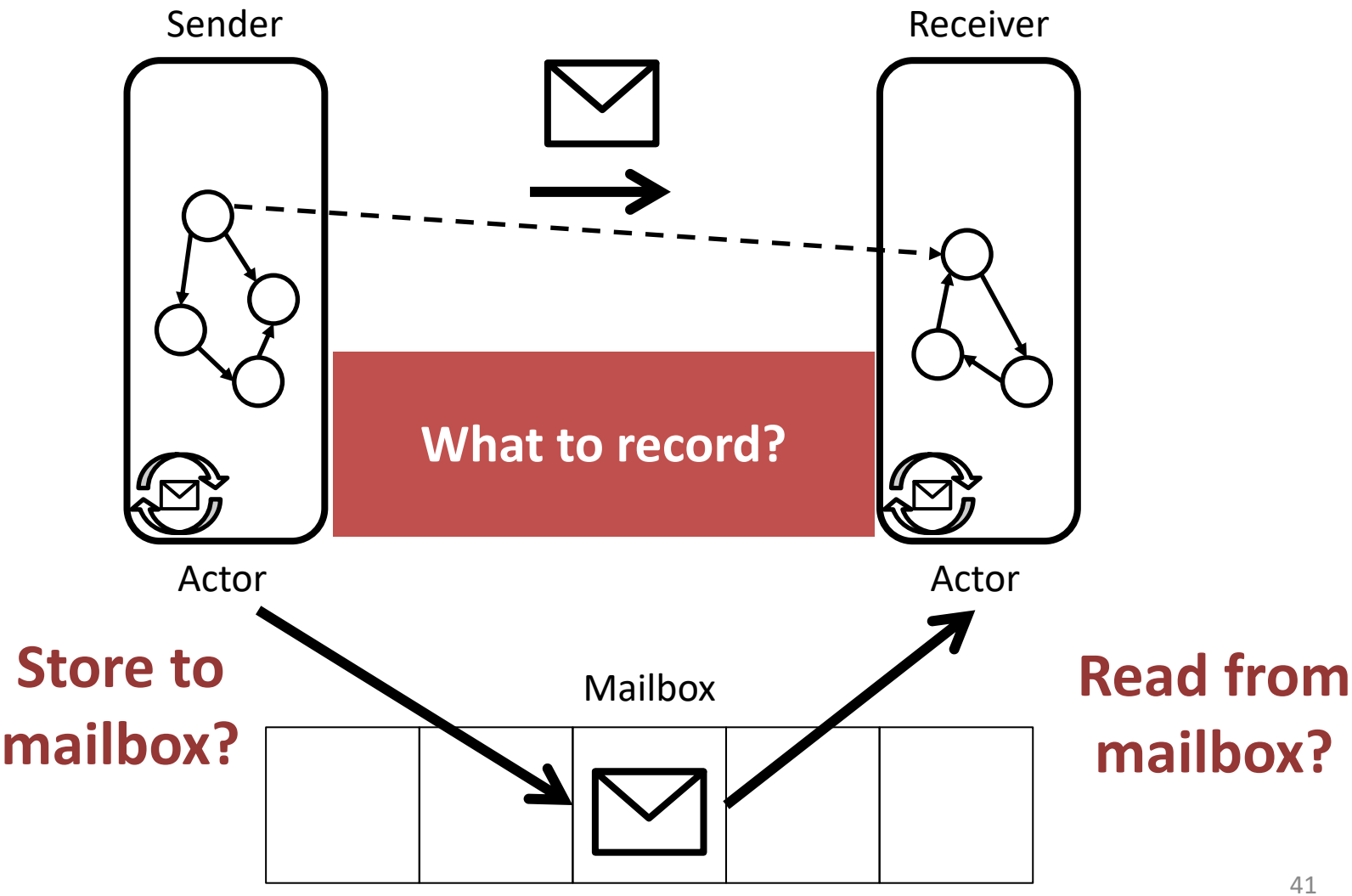
What are the Points of Non-determinism?

Communicating Event Loops



Replay messages in same order as originally

Recording Non-determinism in Communicating Event Loops



For Communicating Event Loops

Sender-side and Receiver-Side Recording are "Functionally Equivalent"

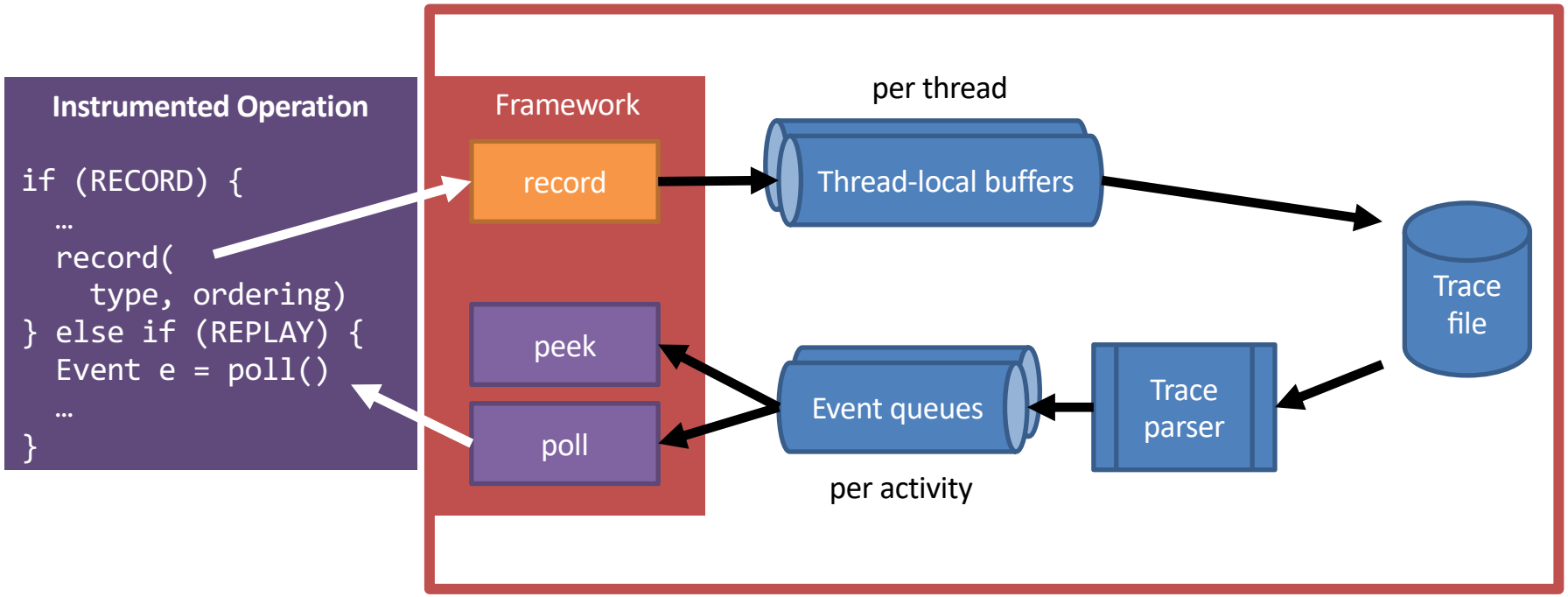
most interesting bit

with complexity
and performance trade-offs

Overview for Concurrency Models

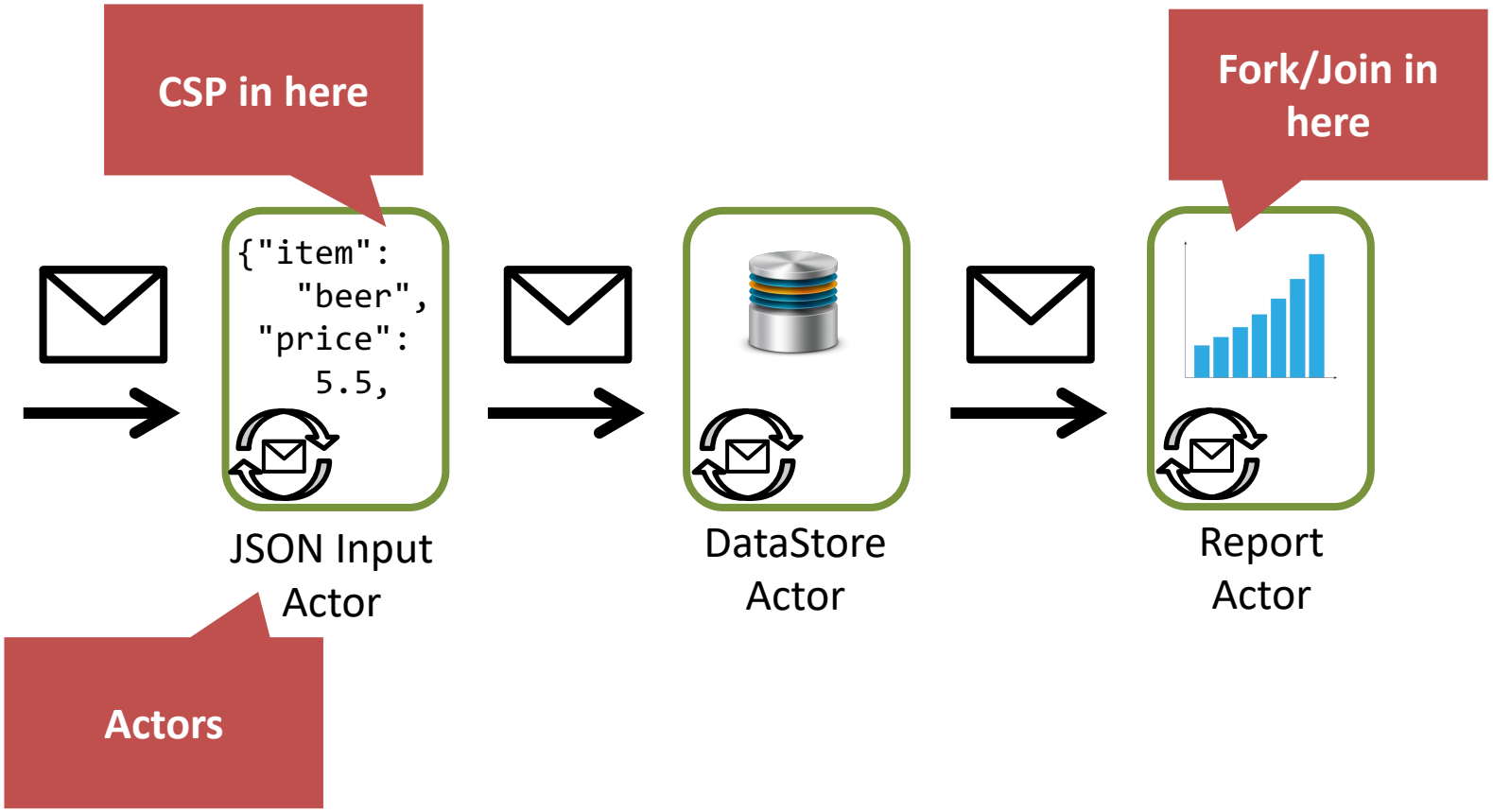
Model	Activities	Passive Entities	Non-determinism
Communicating Event Loops	Actor	Promise, Message	Message order per actor
Threads & Locks	Thread	Lock, Condition	Order of lock acquisitions
Communicating Sequential Processes	Process	Channel	Order of channel reads/writes
Software Transactional Memory	Transaction	-	Commit order

Model Agnostic Framework



Agnostic of Concurrency Models

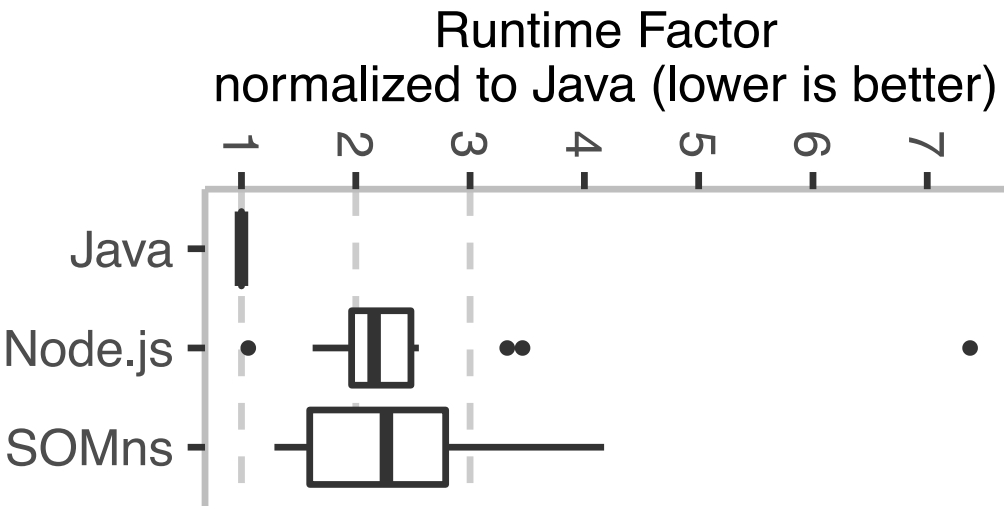
Allows us to Record&Replay a Multi-Paradigm Application



SOM_{NS}: A NEWSPEAK FOR CONCURRENCY RESEARCH

Newspeak: newspeaklanguage.org
SOM_{NS}: github.com/smarr/SOMns

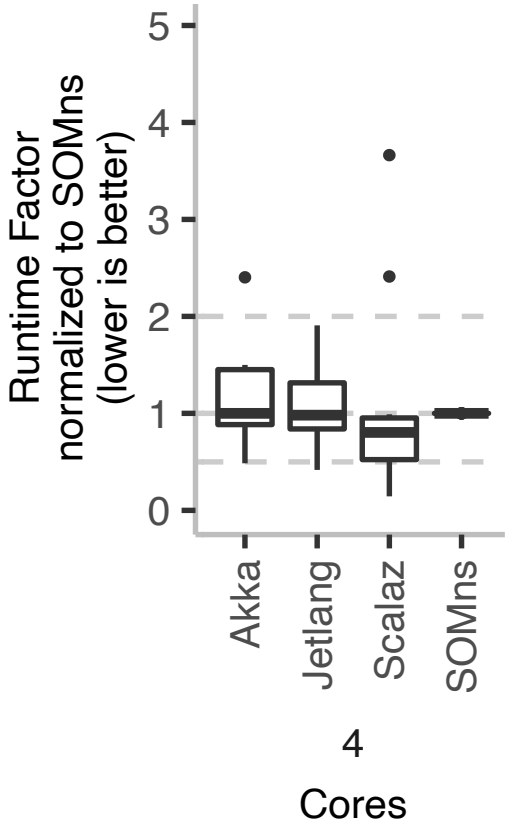
Performance: Baselines



SOM_{NS} is on level of optimized dynamic languages!

Are We Fast Yet: Cross-Language Comparison
https://github.com/smarr/are-we-fast-yet#readme

Performance: Baselines



Competitive with JVM actor frameworks!

Savina Actor Benchmark Suite
https://github.com/shamsimam/savina#readme

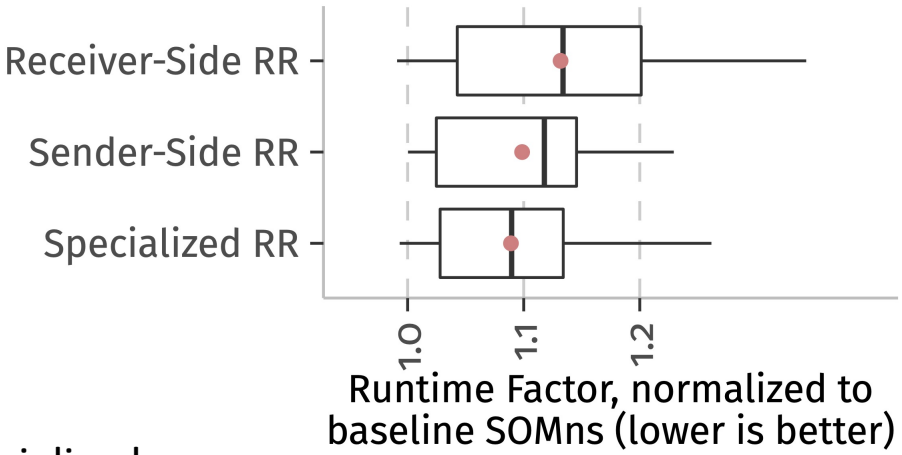
Overhead of Recording Actors for Replay

Overhead on Savina benchmarks
over execution without recording (geometric)

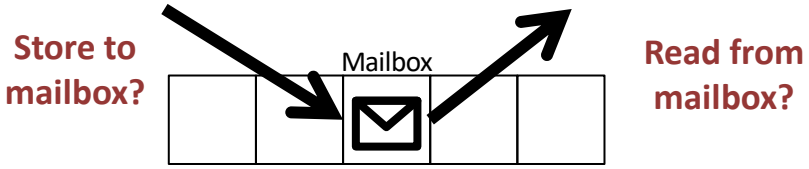
- Specialized: 7.89%
  min. -21.42%, max. 36.29%
  (specialized to actors,
  without support for
  other concurrency models)

- Sender-side: 7.82%
  min. -17.84%, max. 41.23%
  - Performance is competitive with specialized implementation

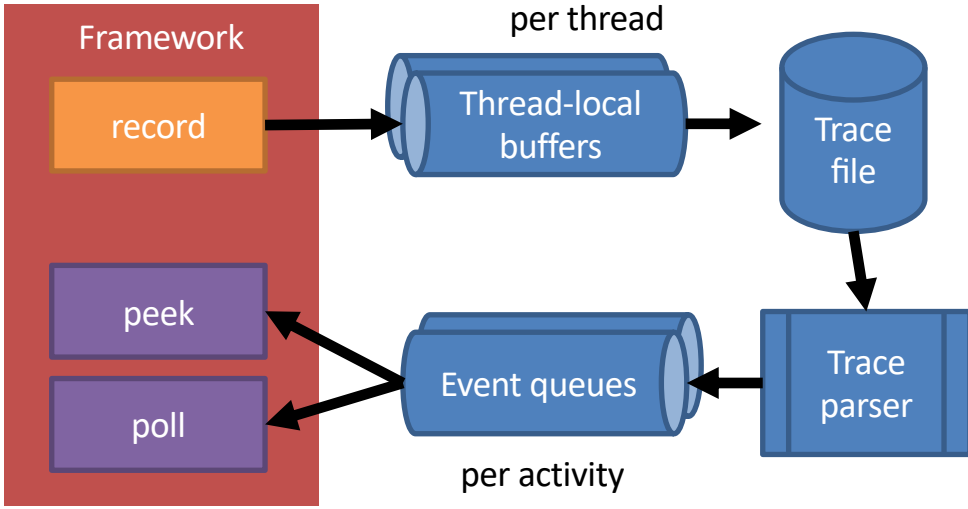
- Receiver-side: 13.23%
  min. -19.33%, max. 53.1%
  - Not as optimized as specialized



Agnostic Record&Replay is Practical!



Capture Non-determinism Per Concurrency Model

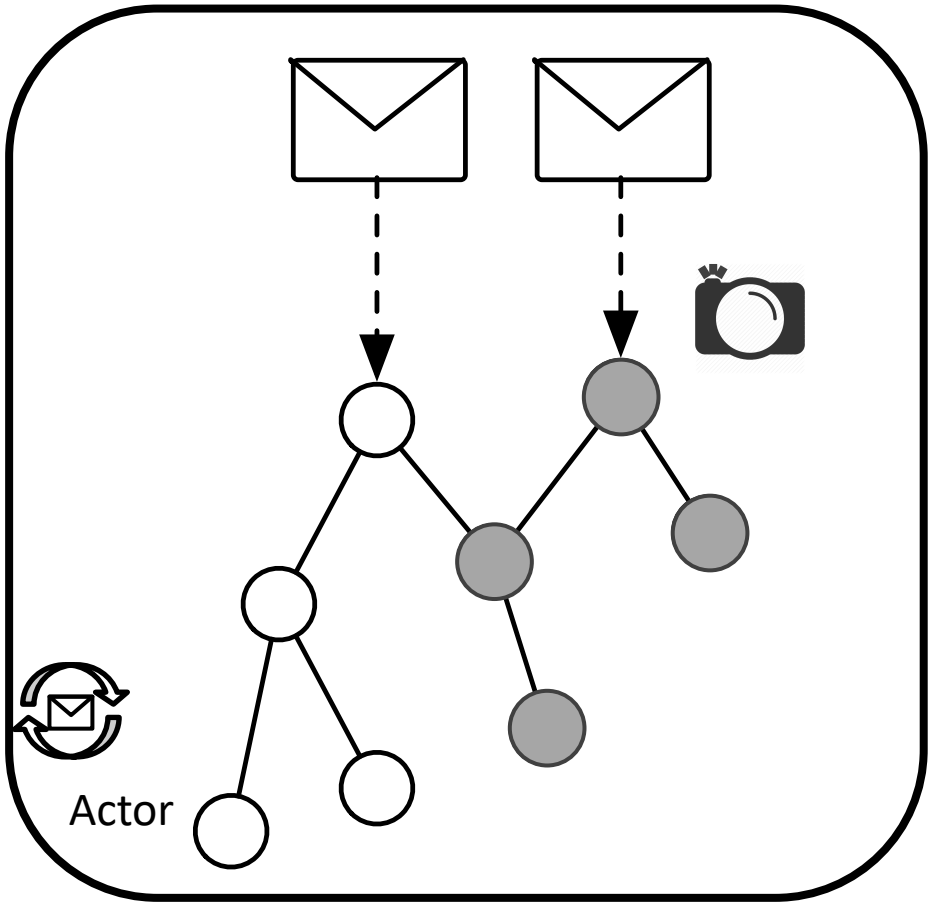


Keep Framework Agnostic

Snapshotting Actor Systems without Stopping Them

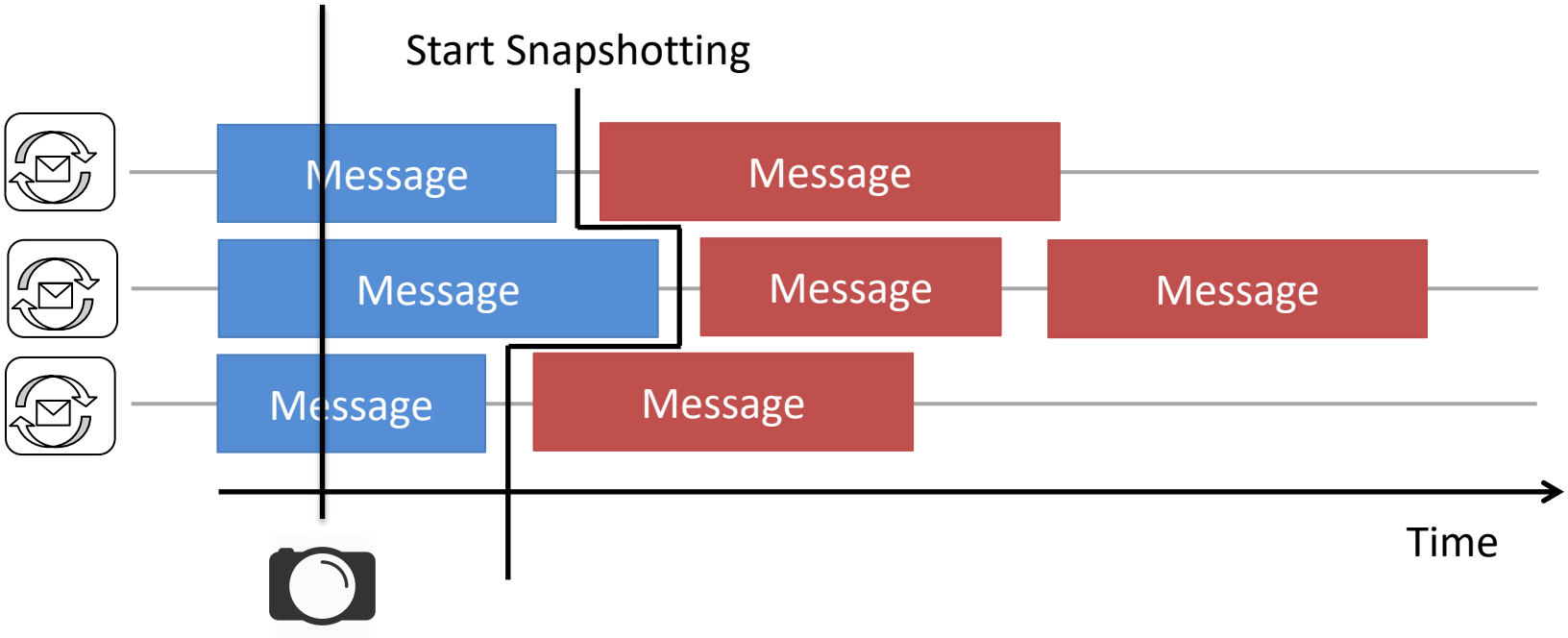
LONG AND HUGE TRACES MAKE REPLAY IMPRACTICAL

Asynchronous and Partial Heap Snapshots



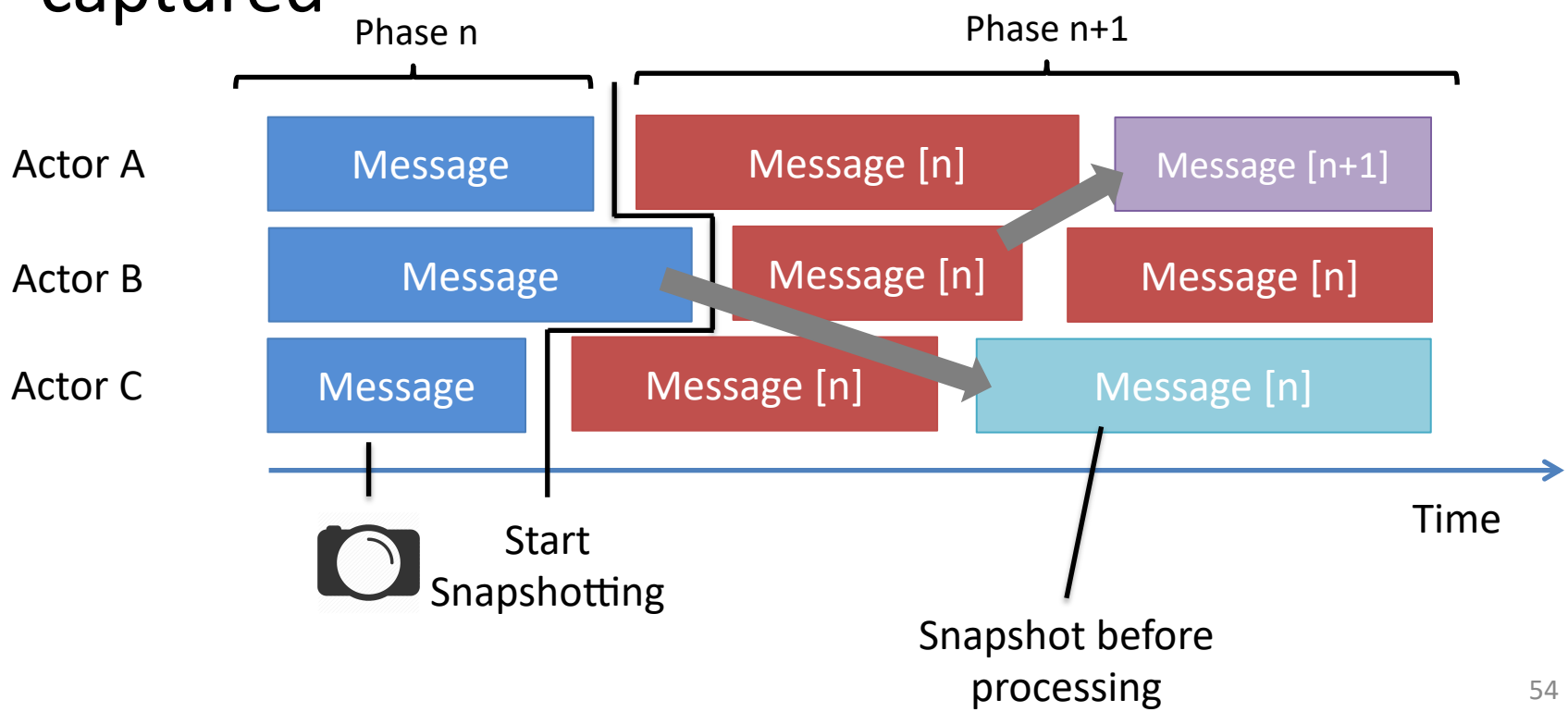
snapshot on message receive
but only objects reachable from a message

Snapshotting without Global Synchronization

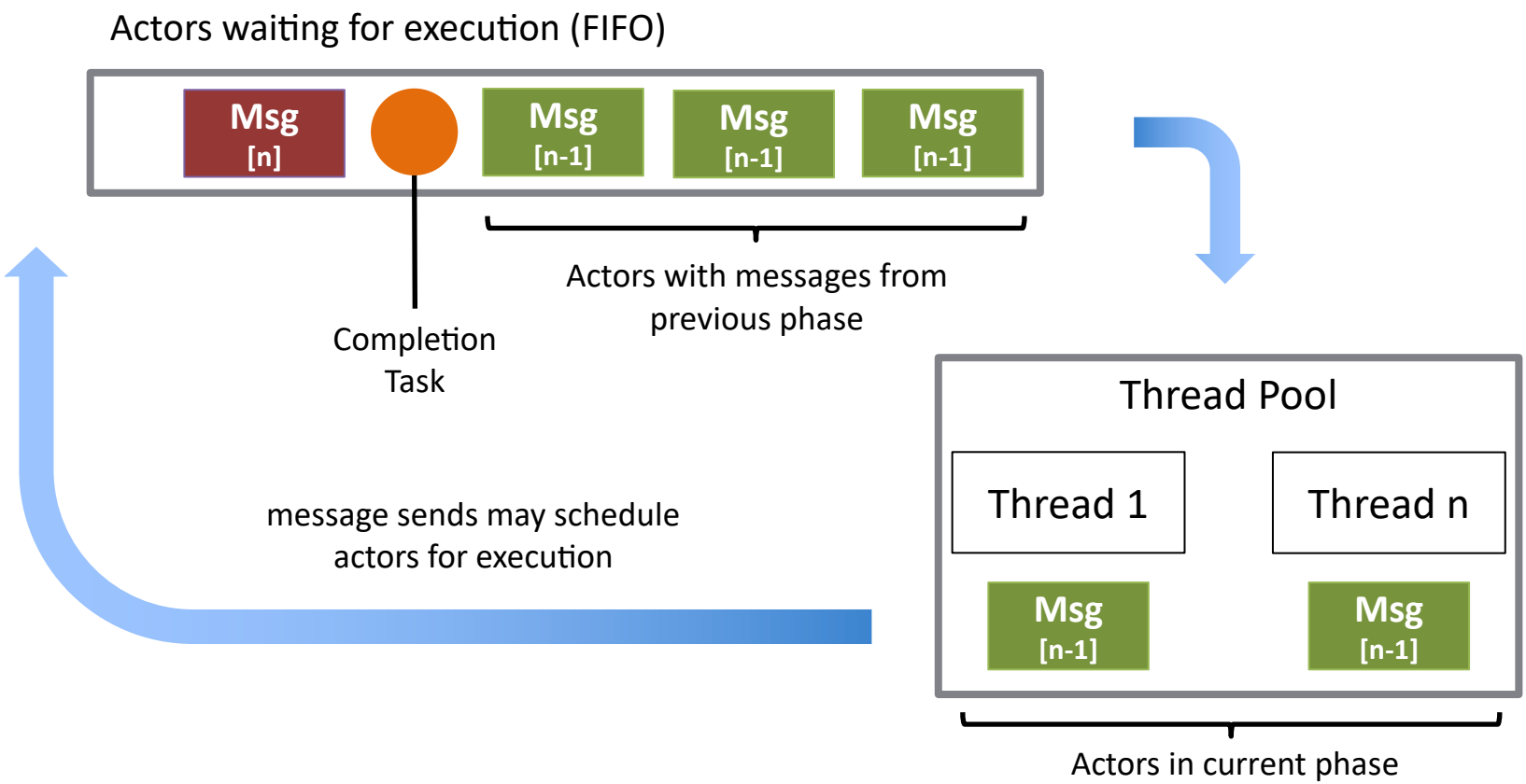


Detecting Message Crossovers

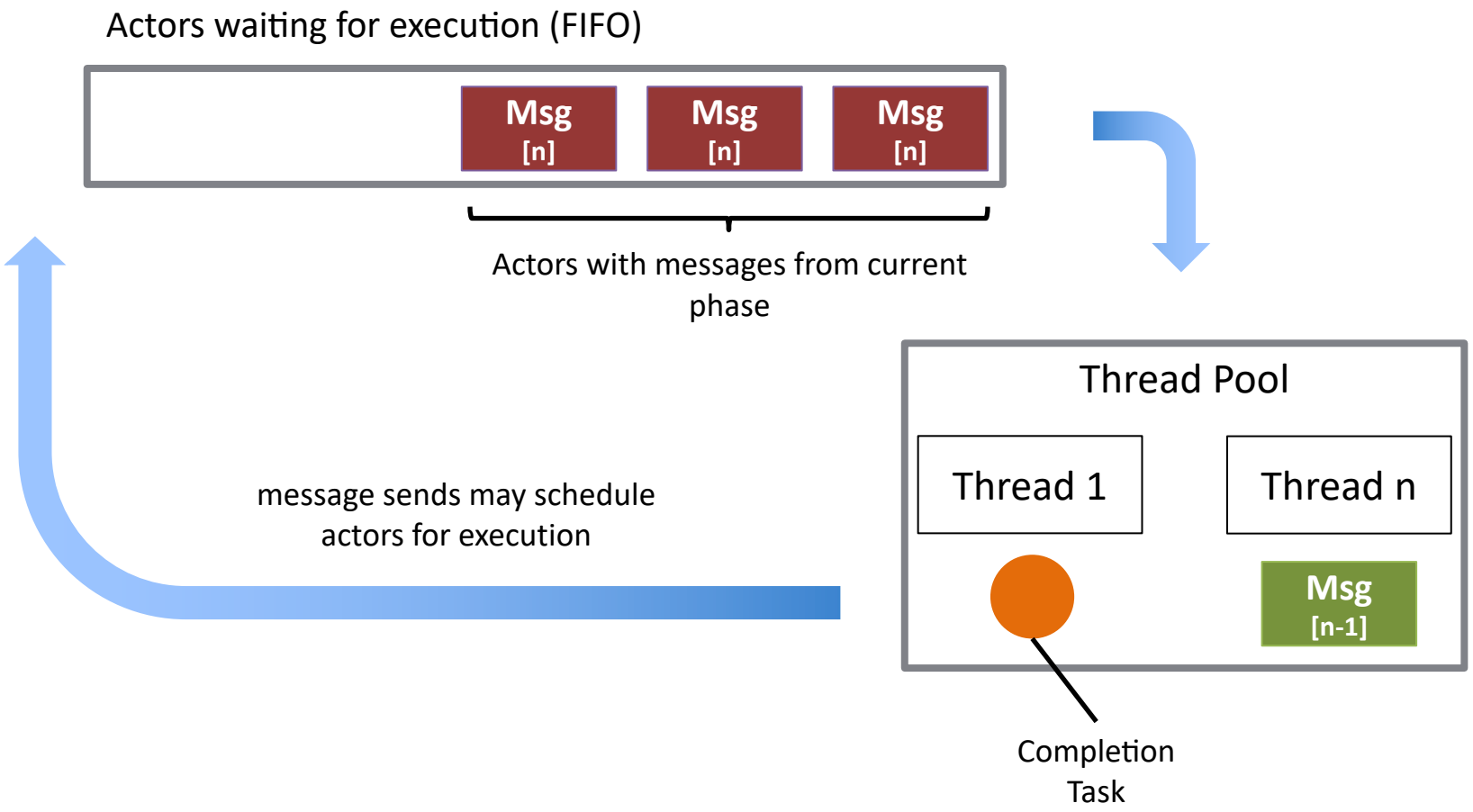
- Attach send phase number to messages
- Messages sent in Phase n (previous) are captured



Detecting Snapshot Completion (2)

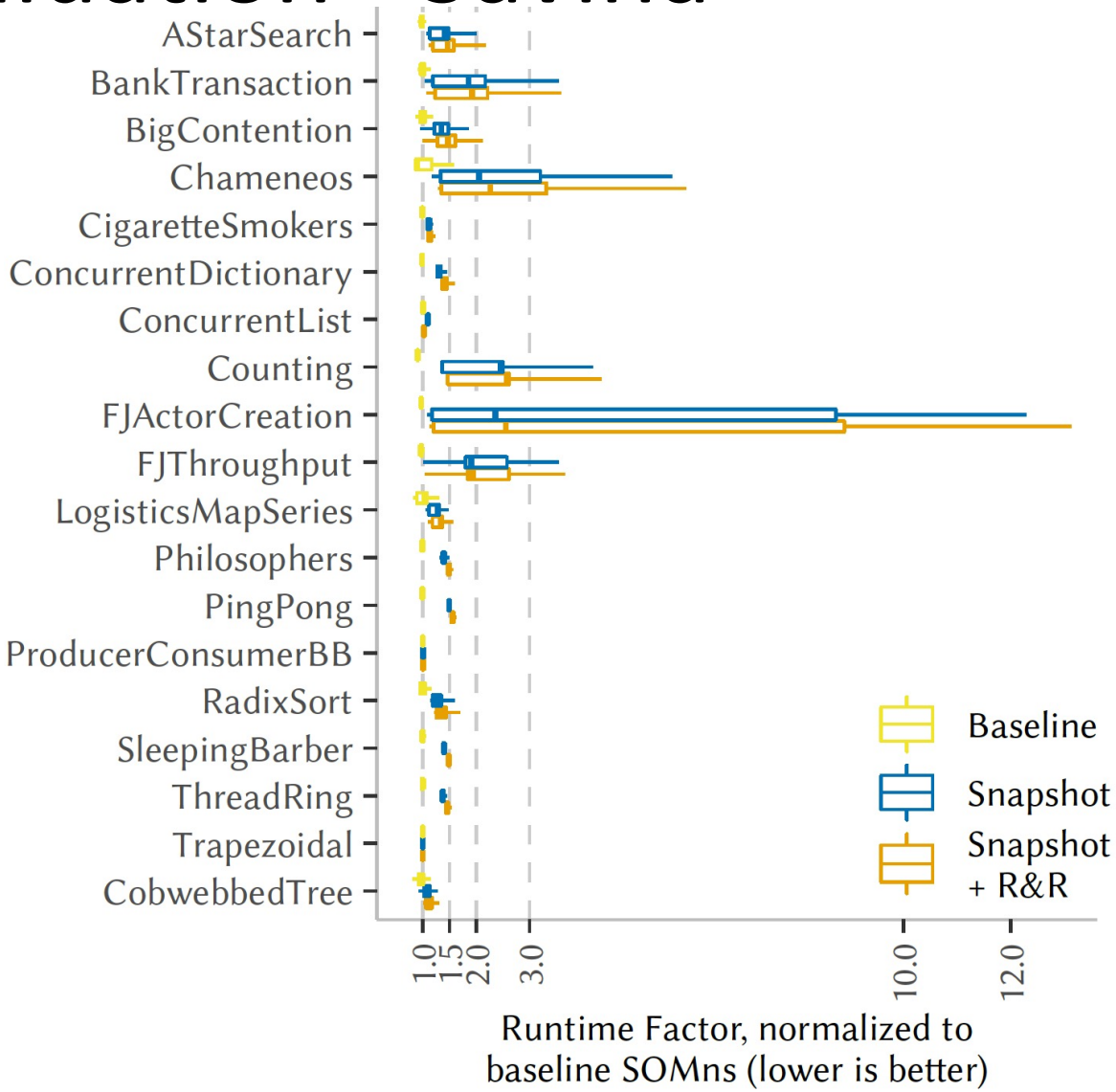


Detecting Snapshot Completion (3)



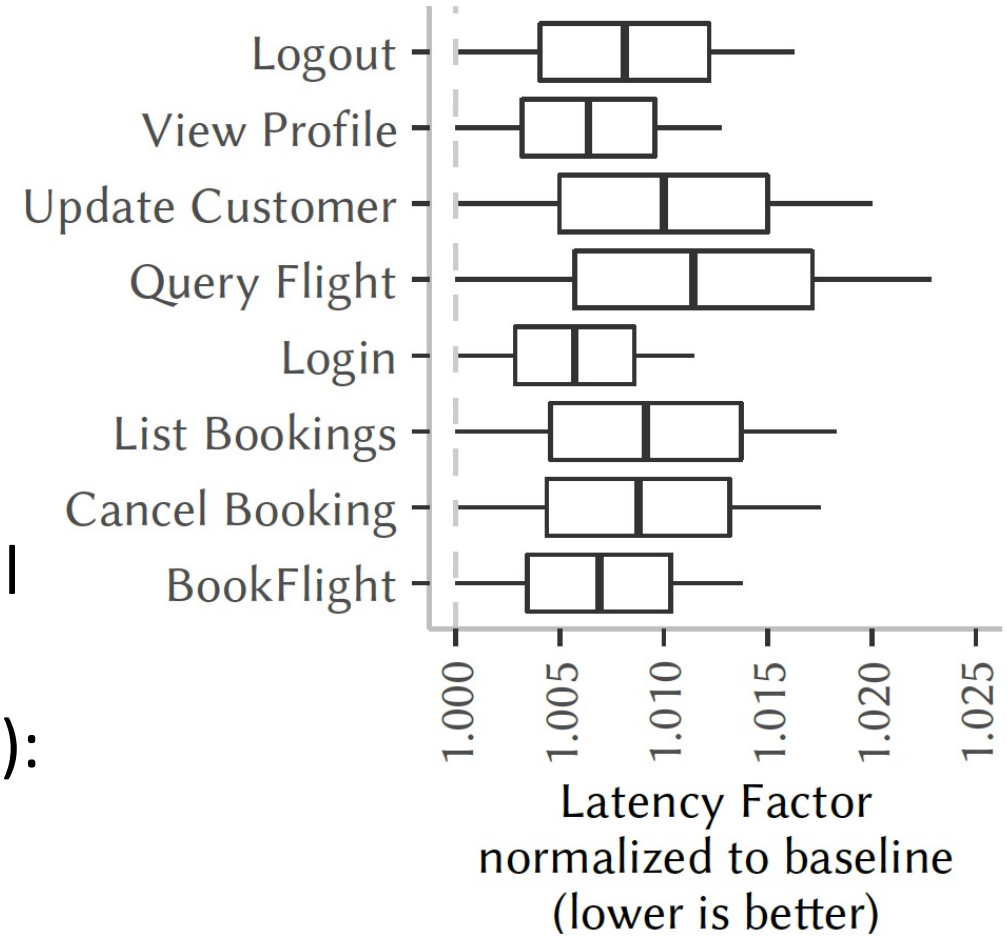
Evaluation - Savina

- Snapshot every second iteration
- Worst-case scenario

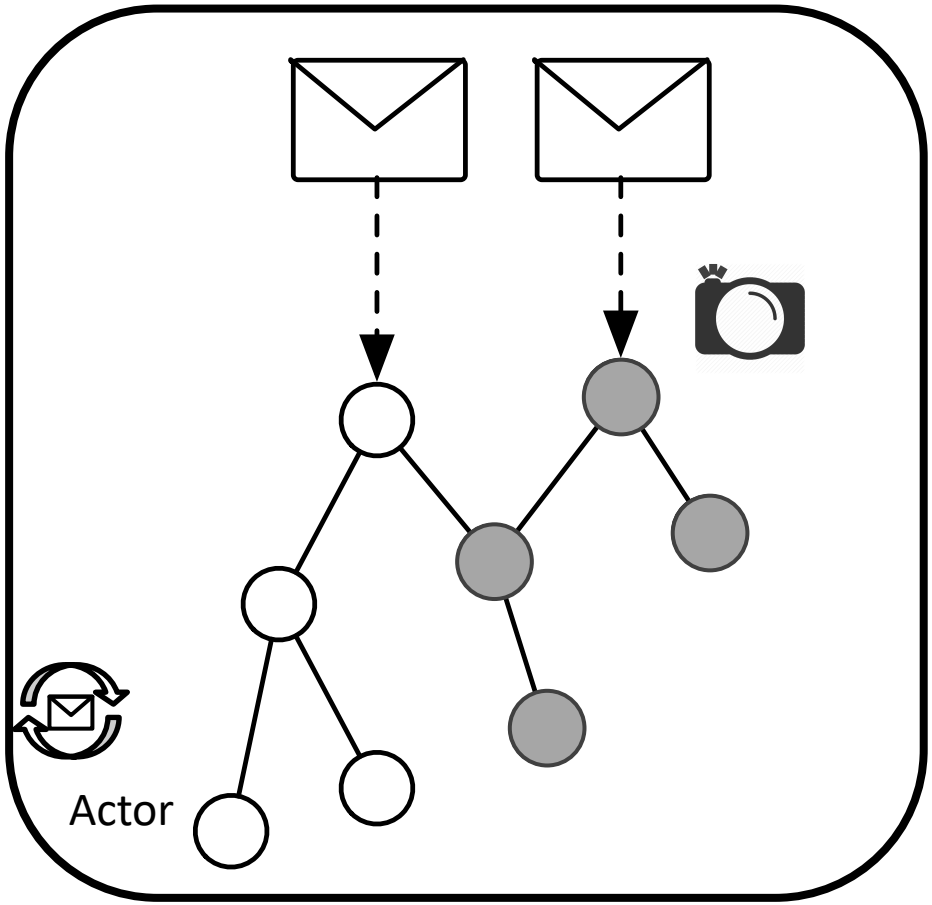


Evaluation – AcmeAir Web Application

- Snapshot every 1000 requests
- Latency increases minimally (1,66% geo mean)
- 20 Million requests total
- Slow requests (> 100ms): 5.43% increase (0.007% of total requests)



Snapshots can be Low-Overhead, Without Stop-the-World Pause



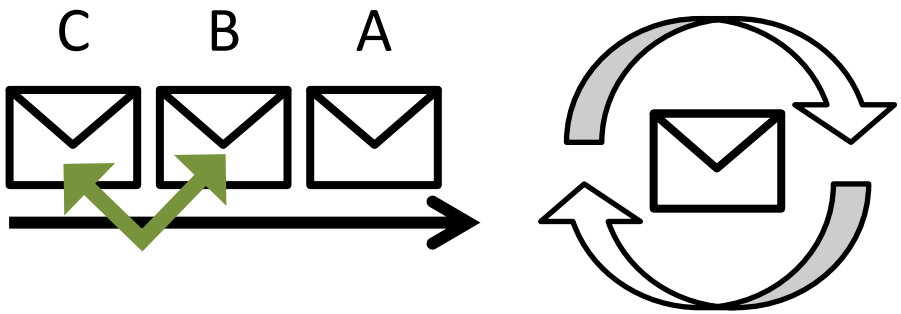
Looking for a PostDoc

If it fails only 1 in 10 times, can we avert failure?

BUG MITIGATION

Bug Mitigation: Basic Idea

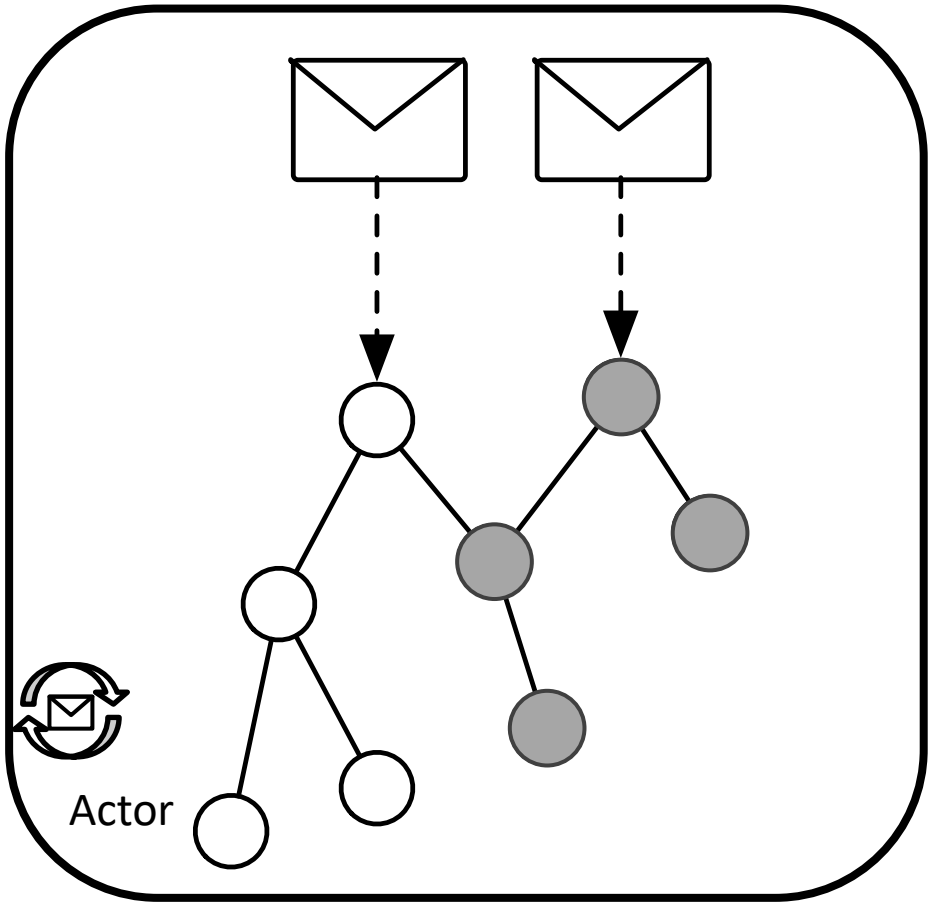
Detect Event Races At Run Time



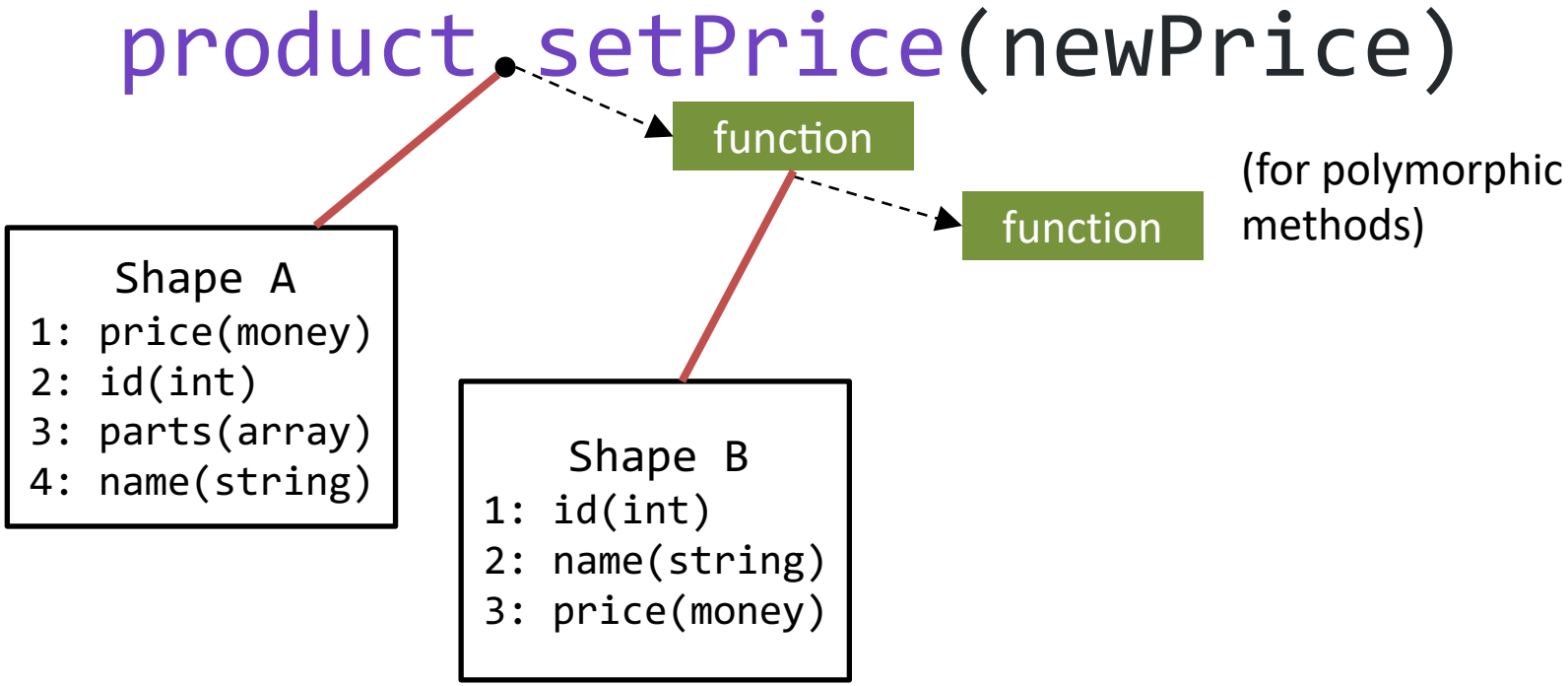
Order A -> B -> C problematic?

Let's swap them!

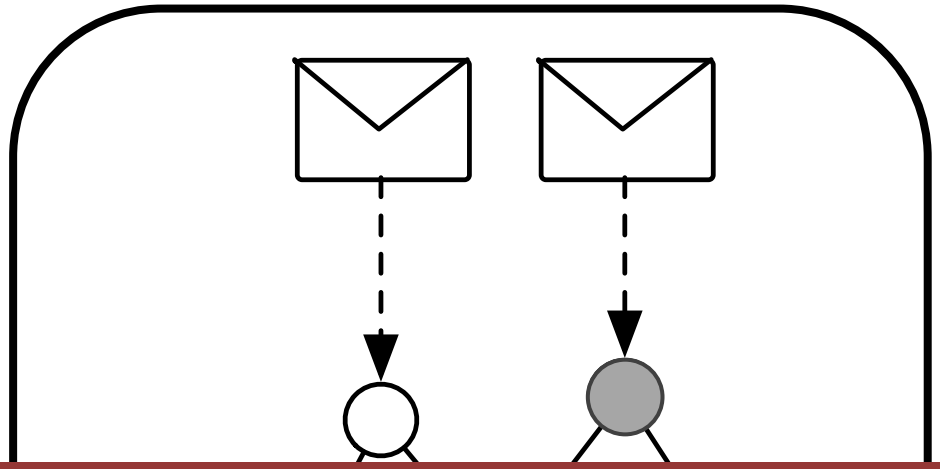
Messages Usually Access Predictable Parts of the Heap



Use Existing VM Techniques to Minimize Race Detection Overhead



Restrict Monitoring to Parts that can Race



Very Early, but:
Heap Access Patterns promising for light-weight, low-precision race-possibility detection

WRAP-UP/CONCLUSION

We're Looking for a Postdoc!

Job Ad

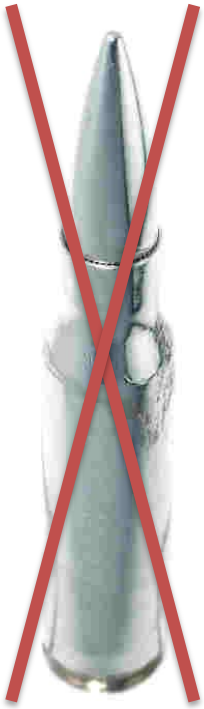
Project CaMELot: Catch and Mitigate Event-Loop Concurrency Issues

Please get in touch!

University of Kent

https://stefan-marr.de/2021/02/open-postdoc-position-on-language-implementation-and-concurrency/

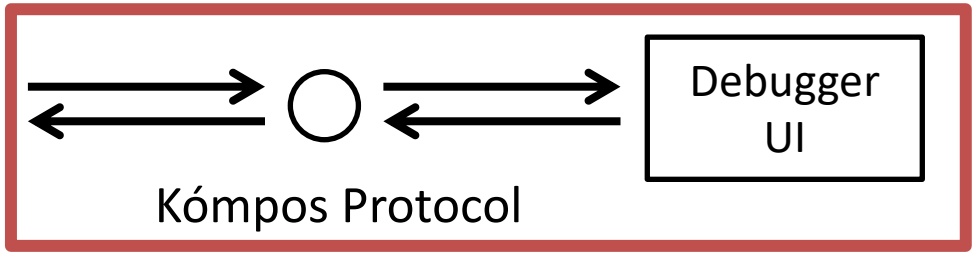
Maybe there are no Silver Bullets?



- **Actors**
- **CSP**
- **Locks, Monitors, …**
- **Fork/Join**
- **Transactional Memory**
- **Data Flow**

…

Debuggers can be Great for High-level Concurrency Models!



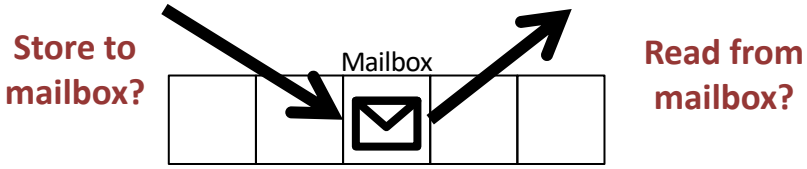
Make tools agnostic

promise resolver
promise resolution

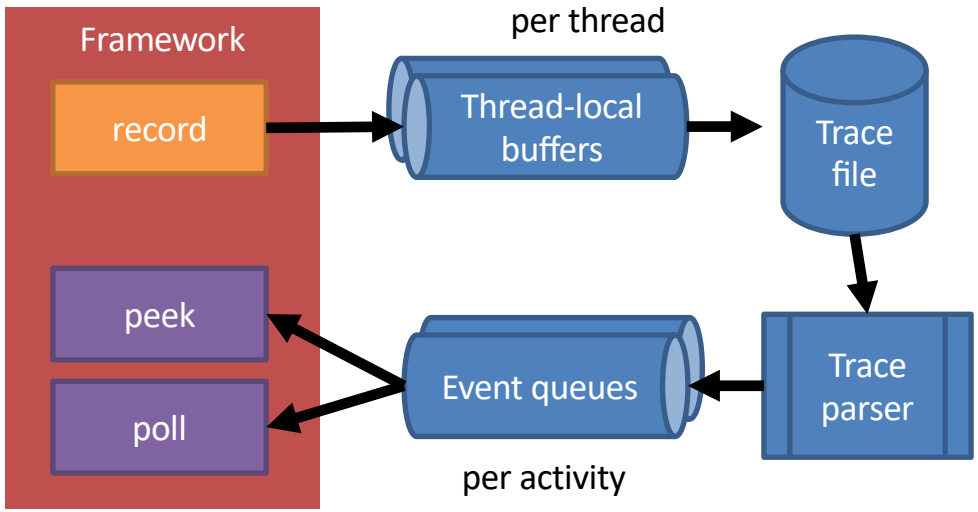
```
prom whenResolved: [:r |
  r println ].
```

Offer the Key Features as Breakpoints/Steps

Agnostic Record&Replay is Practical!

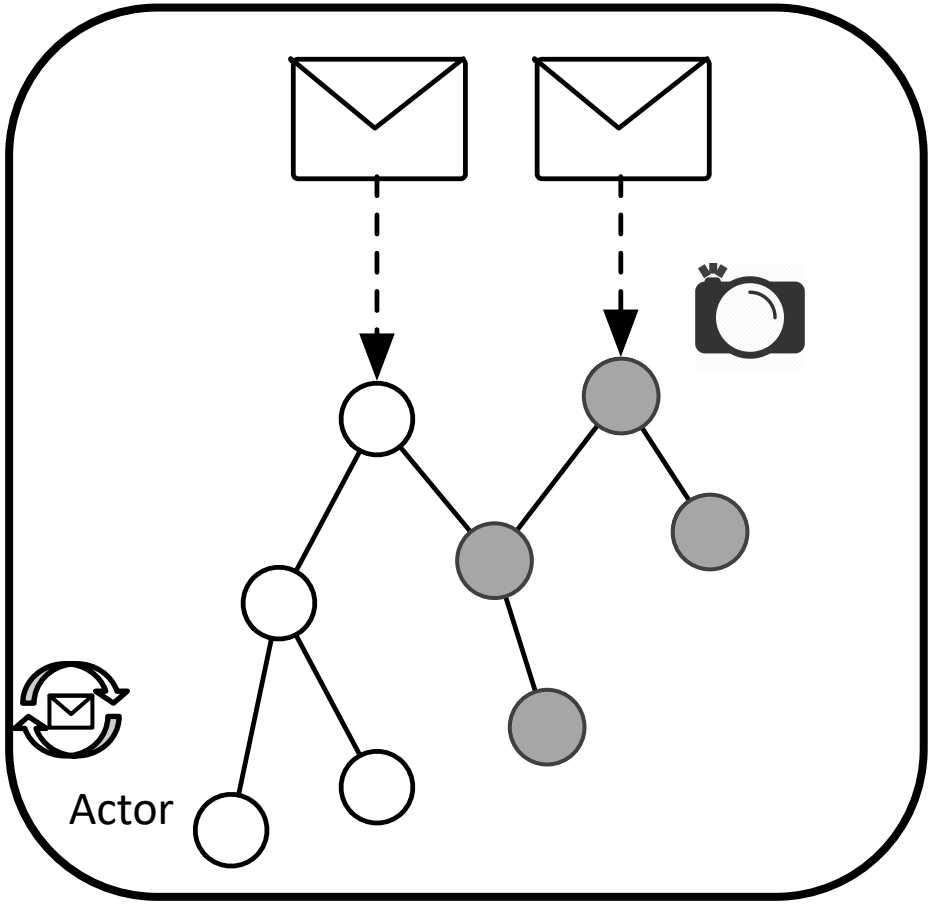


Capture Non-determinism Per Concurrency Model

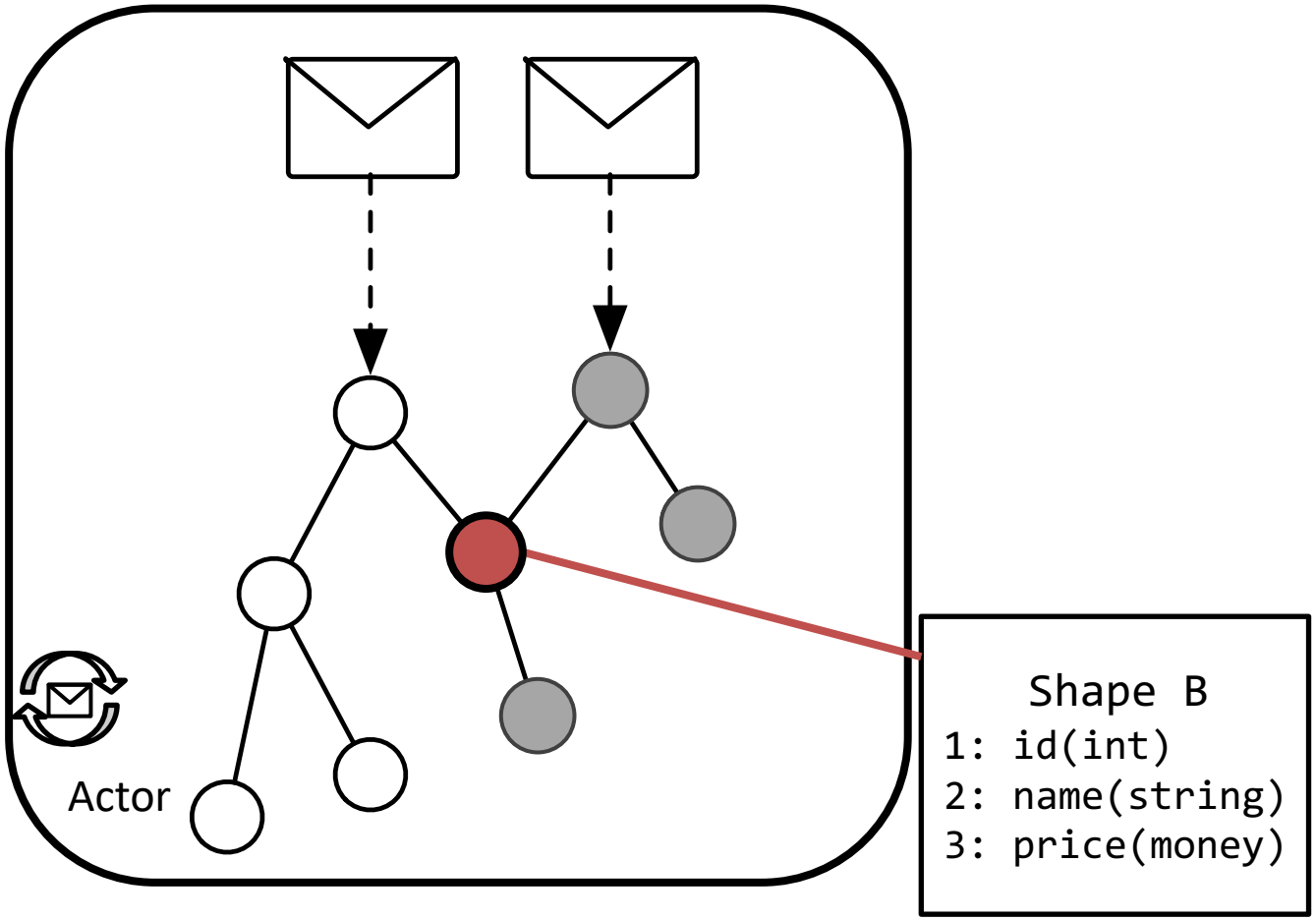


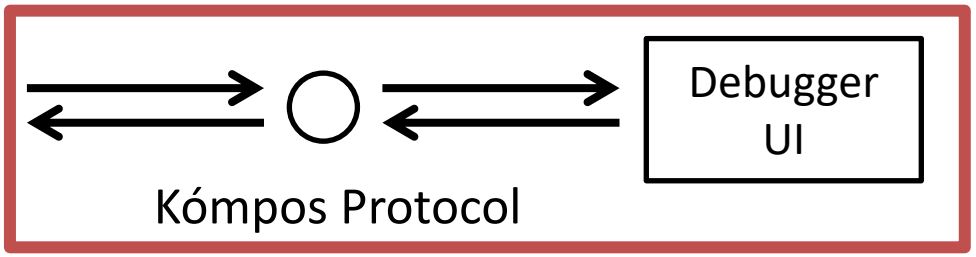
Keep Framework Agnostic

Snapshots can be Low-Overhead, Without Stop-the-World Pause

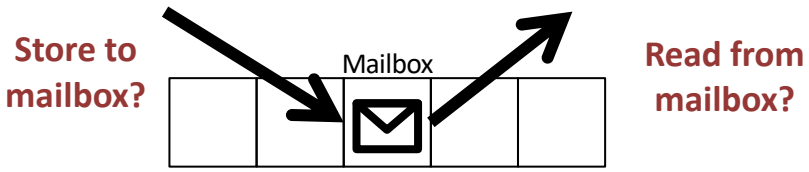


And maybe,
we can use it to do race-mitigation!

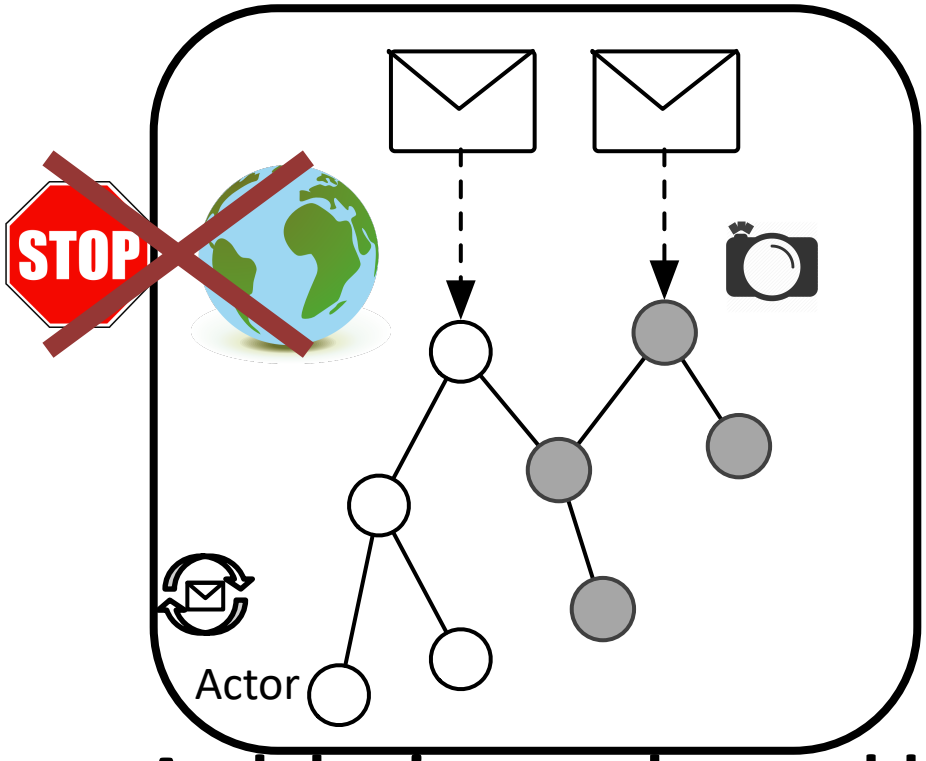




Make tools agnostic



Capture Non-determinism Per Concurrency Model



And don't stop the world for snapshotting!

References

- **Capturing High-level Nondeterminism in Concurrent Programs for Practical Concurrency Model Agnostic Record & Replay** (pdf)
  D. Aumayr, *S. Marr*, S. Kaleba, E. Gonzalez Boix, H. Mössenböck, <**Programming**>, p. 39, AOSA Inc., 2021. doi: 10.22152/programming-journal.org/2021/5/14
- **Asynchronous Snapshots of Actor Systems for Latency-Sensitive Applications** (pdf)
  D. Aumayr, *S. Marr*, E. Gonzalez Boix, H. Mössenböck, **MPLR'19**, p. 157–171, ACM, 2019. doi: 10.1145/3357390.3361019
- **Efficient and Deterministic Record & Replay for Actor Languages** (pdf)
  D. Aumayr, *S. Marr*, C. Béra, E. Gonzalez Boix, H. Mössenböck, **ManLang'18**, ACM, 2018. doi: 10.1145/3237009.3237015
- **A Concurrency-Agnostic Protocol for Multi-Paradigm Concurrent Debugging Tools** (pdf)
  S. Marr, C. Torres Lopez, D. Aumayr, E. Gonzalez Boix, H. Mössenböck, **DLS'17**, p. 3–14, ACM, 2017. doi: 10.1145/3133841.3133842
- **Kómpos: A Platform for Debugging Complex Concurrent Applications** (pdf)
  S. Marr, C. Torres Lopez, D. Aumayr, E. Gonzalez Boix, H. Mössenböck, <**Programming Demo'17**>, p. 2:1–2:2, ACM, 2017. Demo. doi: 10.1145/3079368.3079378
- **A Study of Concurrency Bugs and Advanced Development Support for Actor-based Programs** (pdf)
  C. Torres Lopez, *S. Marr*, H. Mössenböck, E. Gonzalez Boix, **AGERE!'16 (LNCS)**, p. 155–185, Springer, 2018. doi: 10.1007/978-3-030-00302-9_6
- **Towards Advanced Debugging Support for Actor Languages: Studying Concurrency Bugs in Actor-based Programs** (pdf)
  C. Torres Lopez, *S. Marr*, H. Mössenböck, E. Gonzalez Boix, **AGERE! '16**, 2016.