Explainable Root Cause Analysis for Failing Microservices

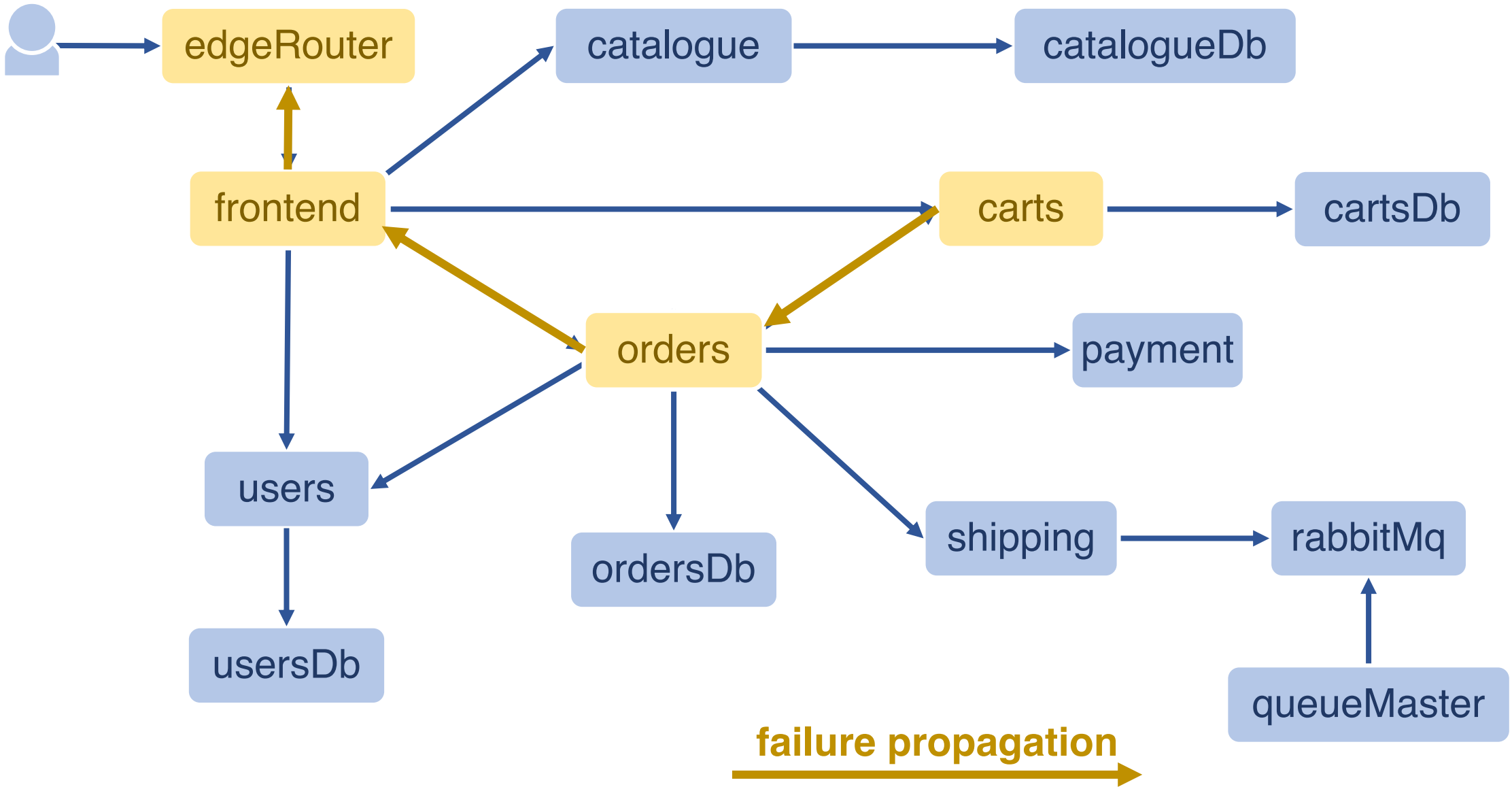
Jacopo Soldani, Stefano Forti, and Antonio Brogi

Department of Computer Science, University of Pisa, Pisa, Italy

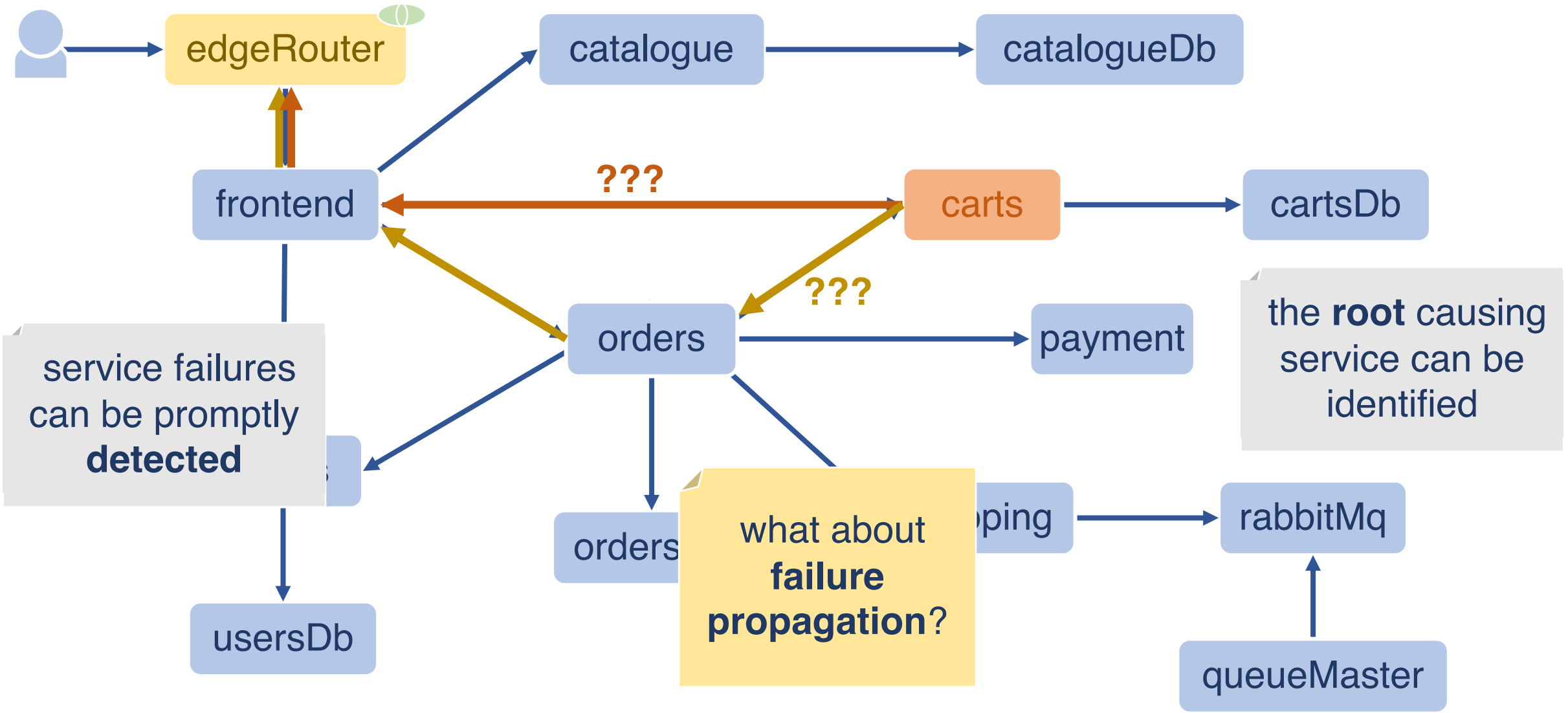
Microservices 2022 – MHS Paris Nord, Paris, France

May 10th, 2022

Failing microservices



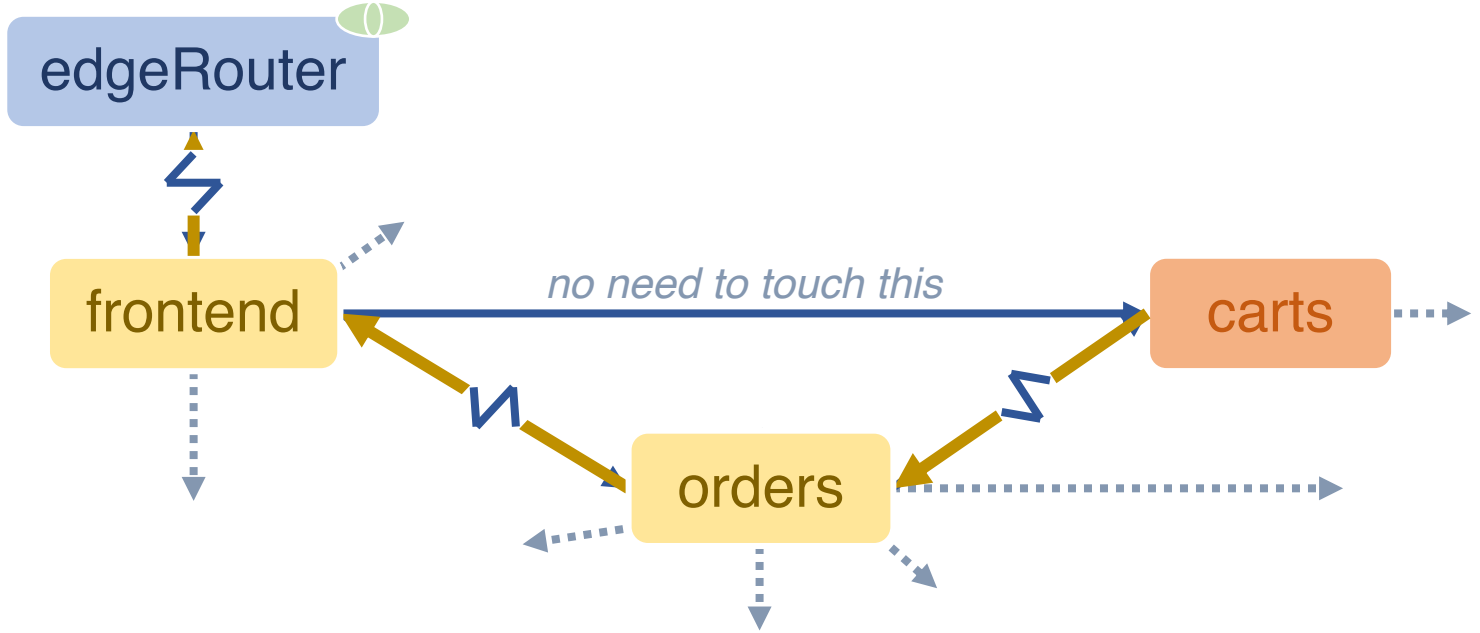
Failure detection & root cause analysis[1]



1. Soldani J, Brogi A. *Anomaly Detection and Failure Root Cause Analysis in (Micro)Service-Based Cloud Applications: A Survey*. CSUR 55(3), 2022

On the importance of explanations[1]

Existing root cause analysis techniques lack **explainability**



Explanations would enable intervening to avoid failure propagation

- on the failing service, // e.g., by adding a circuit breaker
- on the services failing in cascade,
- and *only* following the cascade

enabling **fault isolation**

1. Soldani J, Brogi A. *Anomaly Detection and Failure Root Cause Analysis in (Micro)Service-Based Cloud Applications: A Survey*. CSUR 55(3), 2022

Research question & contributions

How to determine the **cascading failures** that possibly caused an observed failure, **identifying (or starting from)** their **root cause**?

Declarative root cause analysis (RCA) technique for processing the events logged by microservice-based applications

yRCA, an open source **prototype** implementation of the proposed root cause analysis technique

Controlled **experiments**, applying our technique to an existing **chaos testbed**

Declarative RCA – Logged Events

Logged events modelled as **facts,** including

- logging **service** (name and instance)
- **timestamp**
- type of **event**
  - → `internal` // internal business logic
  - → `sendTo(Dest,Id)` // sent request (client-side)
  - → `received(Id)` // request received (server-side)
  - → `timeout(Dest,Id)` // expired timeout (client-side)
  - → `errorFrom(Dest,Id)` // error reply returned (client-side)
- **severity**, according to Syslog standard
  - → `emerg..err` // errors
  - → `warning` // warning
  - → `notice..debug` // info

Declarative RCA – Explanations

An explanation is found with

`causedBy(Event,E,RootCause)`

resolved recursively, by cases

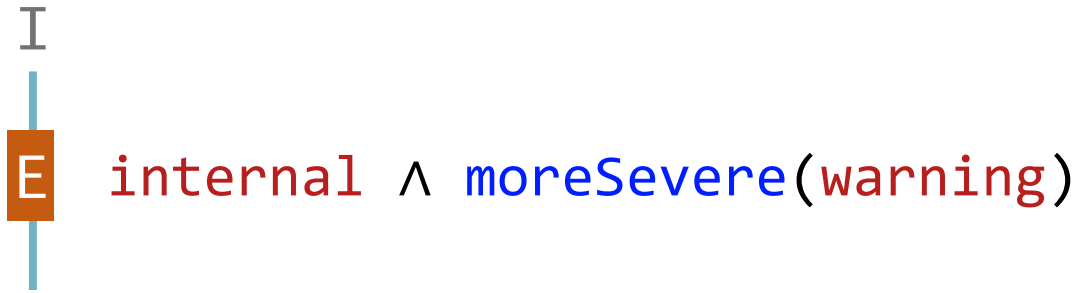
where

- `Event` is the **fact to be explained**
- `RootCause` is the **root causing service**
  - → fixed, if already known (e.g., thanks to other RCA techniques)
  - → automatically identified, otherwise
- `E` is a list of facts explaining the **propagation** of a `RootCause`'s failure to `Event`

All possible explanations = all possible solutions for `causedBy`

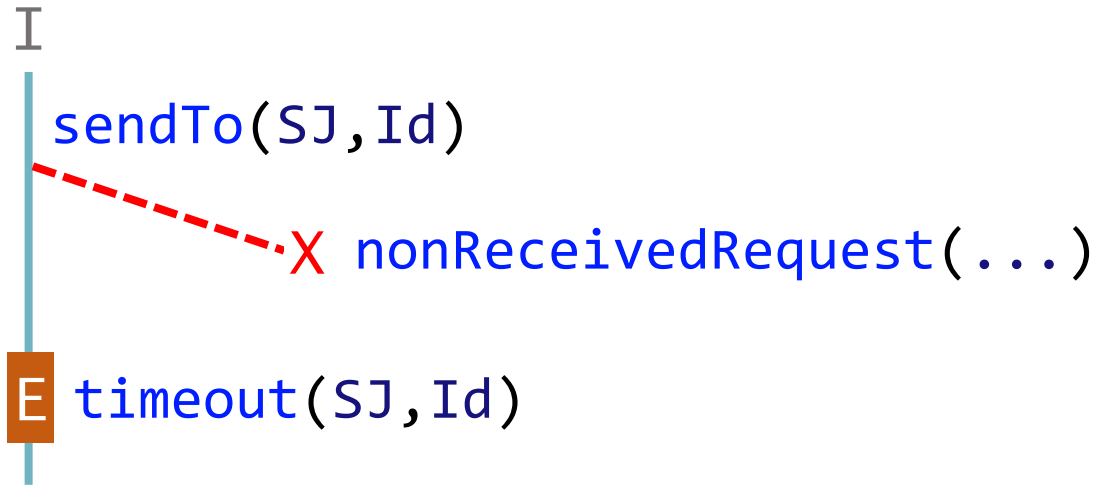
Base cases

internal error



unreachable

service invoked



Recursive Cases

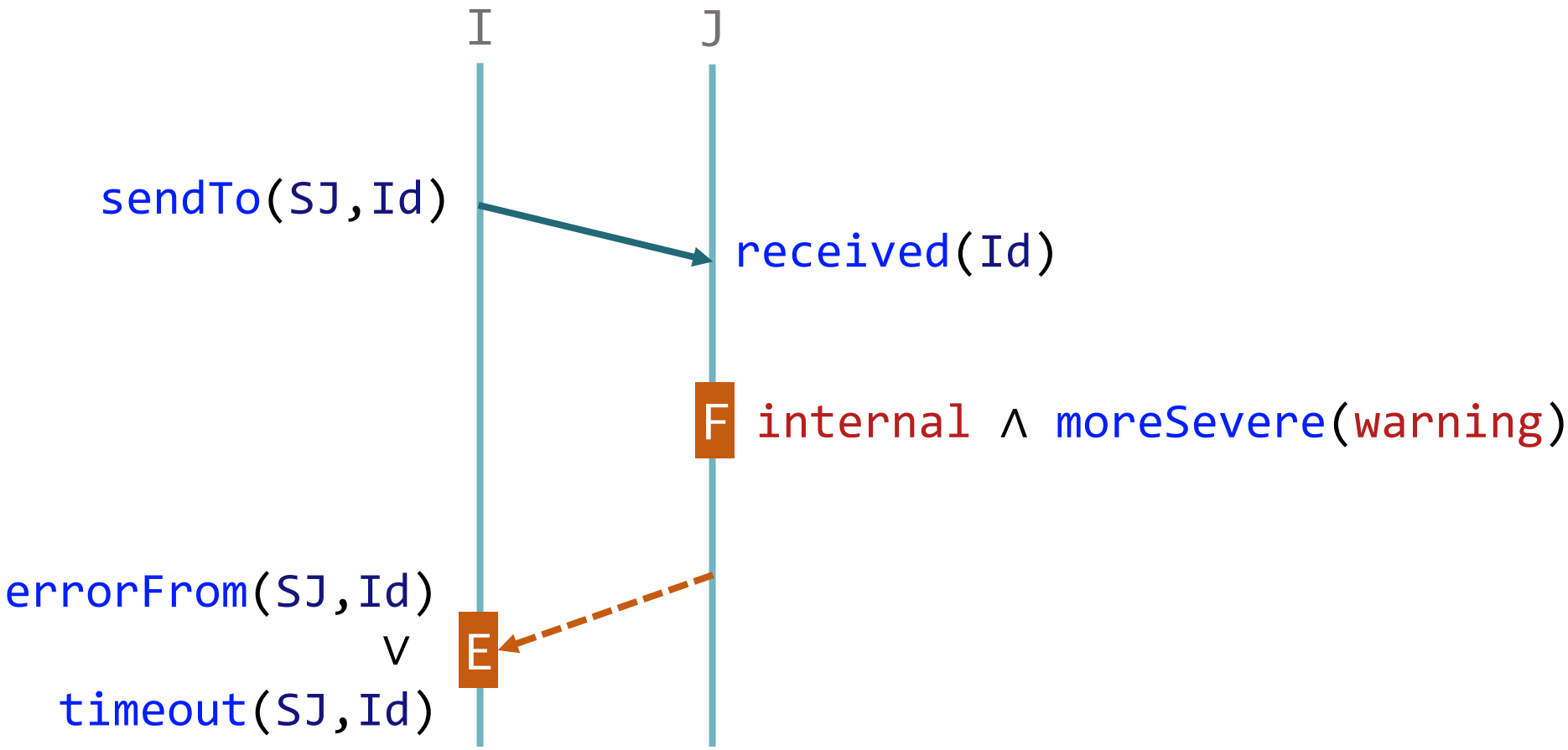
Error `E` (under explanation) **caused by** error **F** of invoked service

→ **recur** to explain `F`

Error `F` can be

- **internal** error

Case – Failure of invoked service



E caused by F and **recur** to explain F

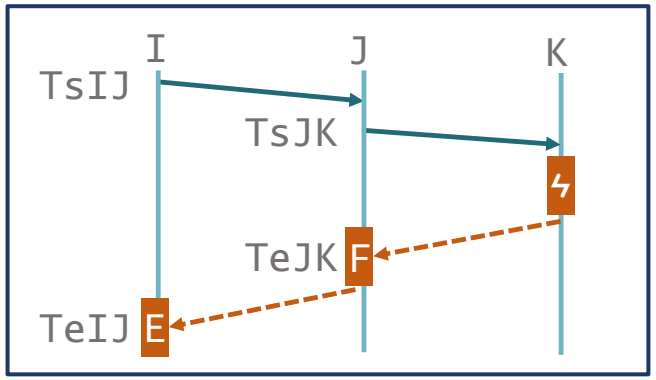
Recursive Cases

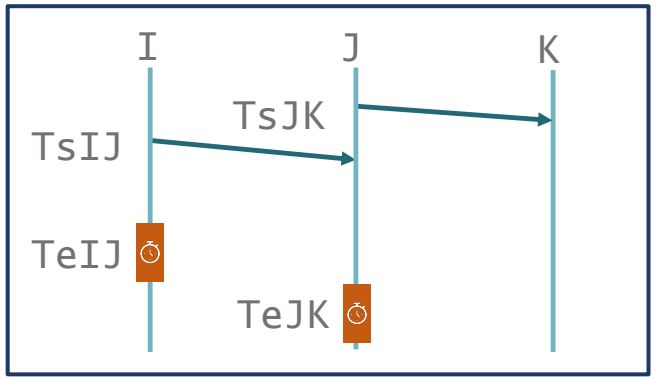
Error **E** (under explanation) **caused by** error **F** of invoked service

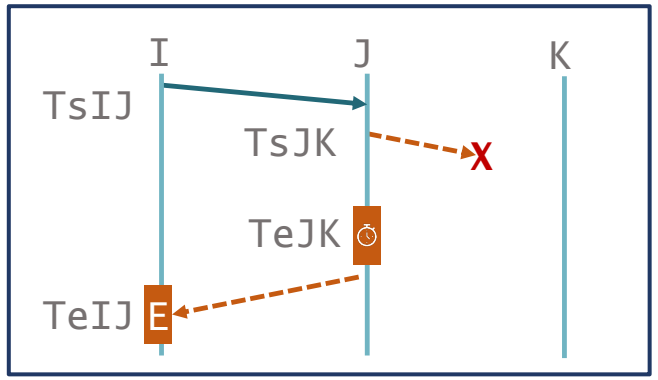
→ **recur** to explain **F**

Error **F** can be

- **internal** error
- **error response** received by another service
- **timed out** interaction with another (reachable) service
- **timeout** expiring since contacting an unreachable service





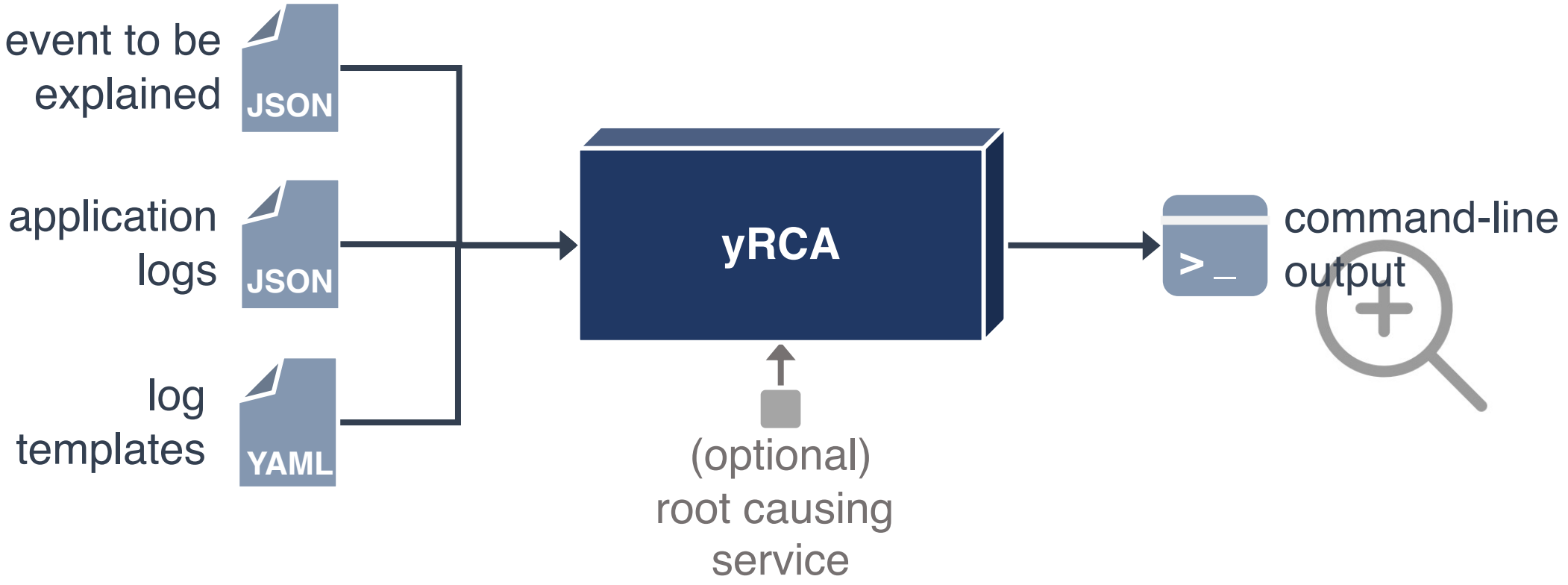


yRCA (disambiguation)

Even if it recalls this

we actually refer to our **prototype implementation** :-)

yRCA



https://github.com/di-unipi-socc/yRCA

yRCA (cont)

```
[0.615]: edgeRouter: Error response (code: 500) received from frontend
(request_id: [<requestId>])
      -> frontend: Error response (code: 500) received from orders
         (request_id: [<requestId>])
      -> orders: Failing to contact carts (request_id: [<requestId>]). Root
         cause: <exception>
      -> carts: unreachable
[0.385]: edgeRouter: Error response (code: 500) received from frontend
(request_id: [<requestId>])
      -> frontend: Failing to contact carts (request_id: [<requestId>]). Root
         cause: <exception>
      -> carts: unreachable
```

probability

root causes

explanation
(cascading failures)

Evaluation

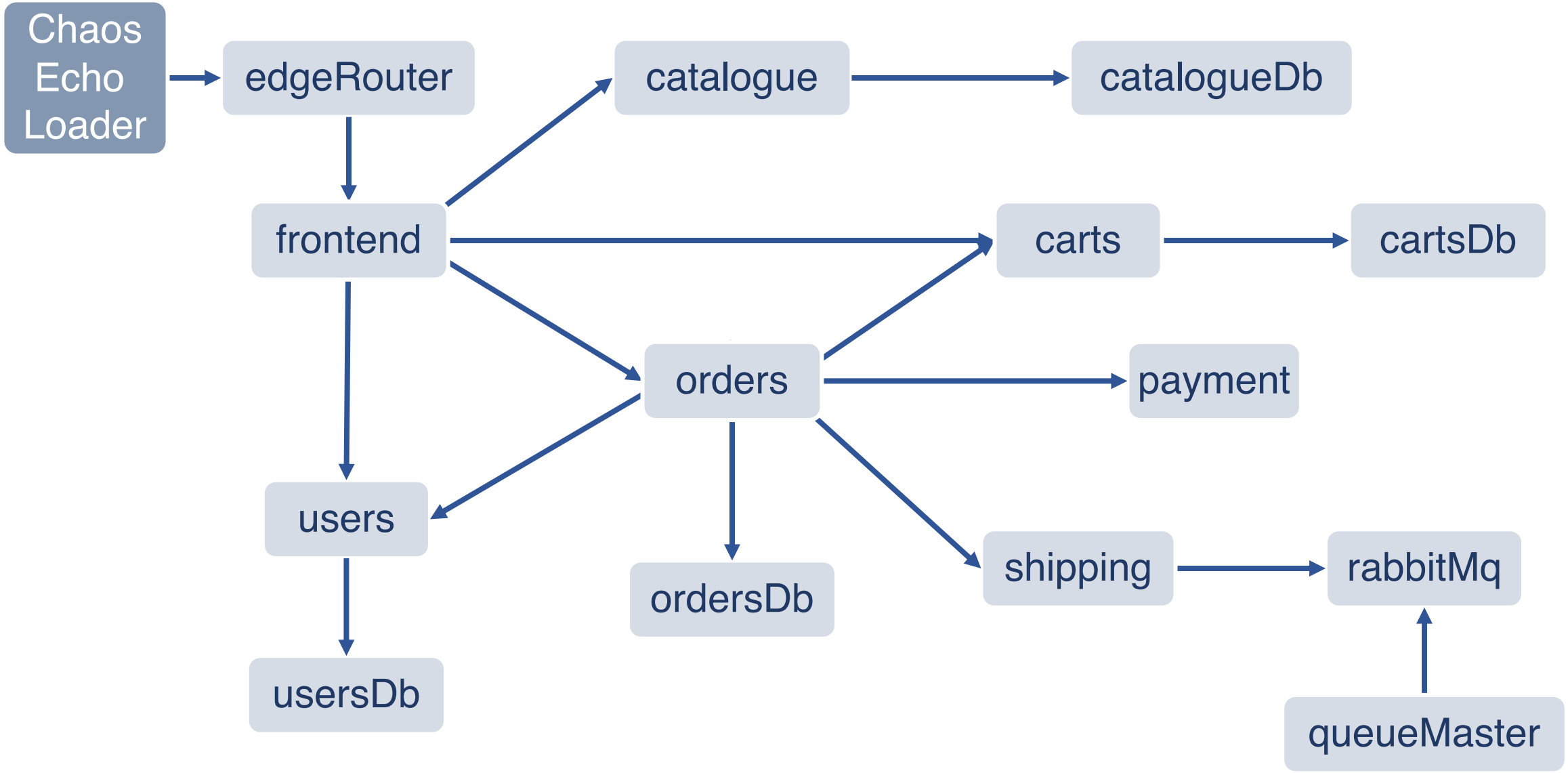
Controlled experiments to evaluate the performances of our RCA in determining the failure cascades that caused an observed failures

- **Reference application** based on the Chaos Echo testbed[1]

1. Soldani J, Brogi A. *Automated Generation of Configurable Cloud-Native Chaos Testbeds*. EDCC 2021 Workshops.

Reference Application

name = Chaos Echo Service



Evaluation

Controlled experiments to evaluate the performances of our RCA in determining the failure cascades that caused an observed failures

- **Reference application** based on the Chaos Echo testbed[1]

- Four different situations
  a) End-user **load** varying from 1 to 100 req/s
  b) Service **interaction probability** varying from 0.1 to 1
  c) Failure **cascade length** varying from 1 to 4
  d) Service **failure probability** varying from 0.1 to 1

  a–c: only one service set to fail on its own, giving a **ground thruth**

  d: any service can fail

- All combinations run to generate at least 200 failures in edgeRouter

1. Soldani J, Brogi A. *Automated Generation of Configurable Cloud-Native Chaos Testbeds*. EDCC 2021 Workshops.

Results

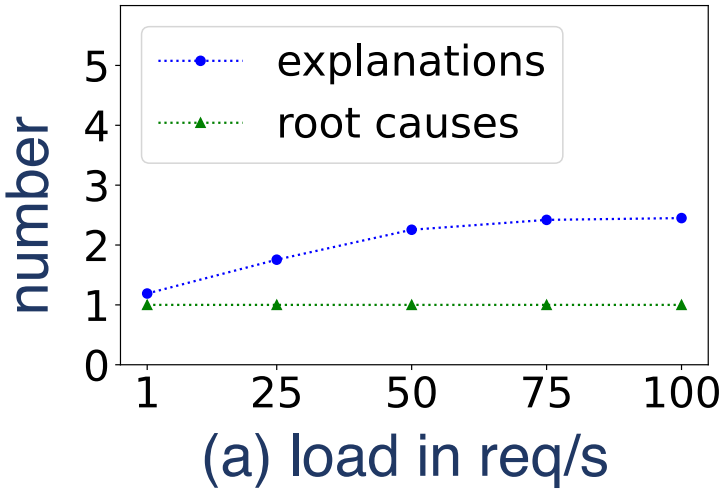
yRCA run on a random sample of 200 edgeRouter's failures (for each case)

99.74% of generated failures were **successfully** explained

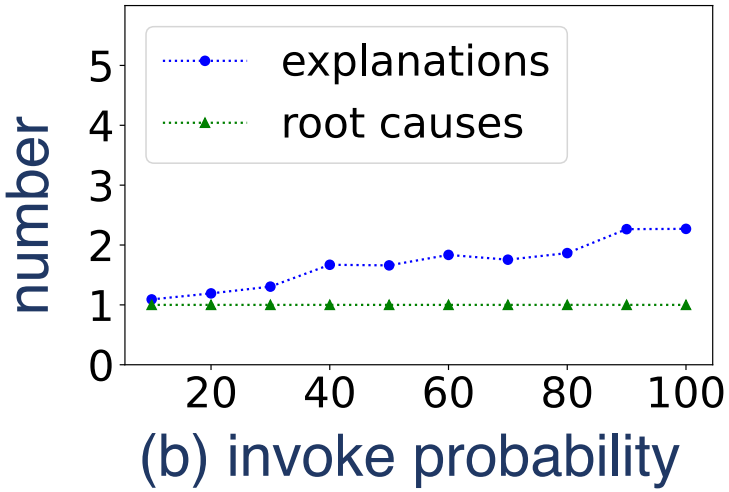
Returned solutions however contained also **false positives**, viz., failure cascades considered to have caused the failure even if this was not the case

Q1) How many **false positives**?

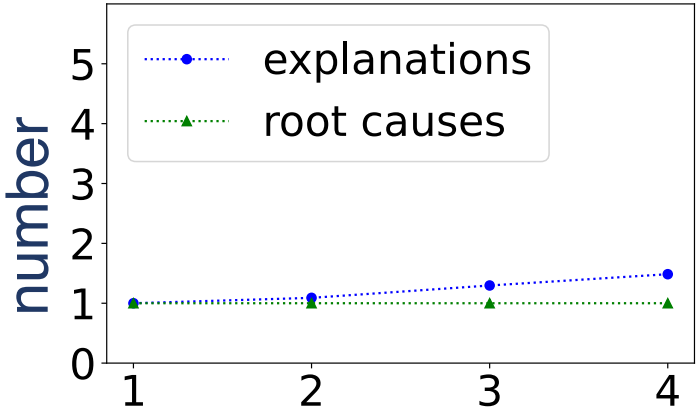
Results – False positives



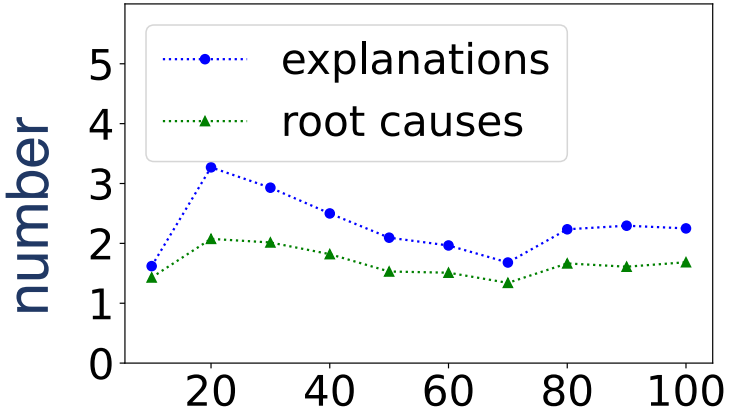
(a) load in req/s



(b) invoke probability



(c) cascade length



(d) fail probability

Results

yRCA run on a random sample of 200 edgeRouter's failures (for each case)

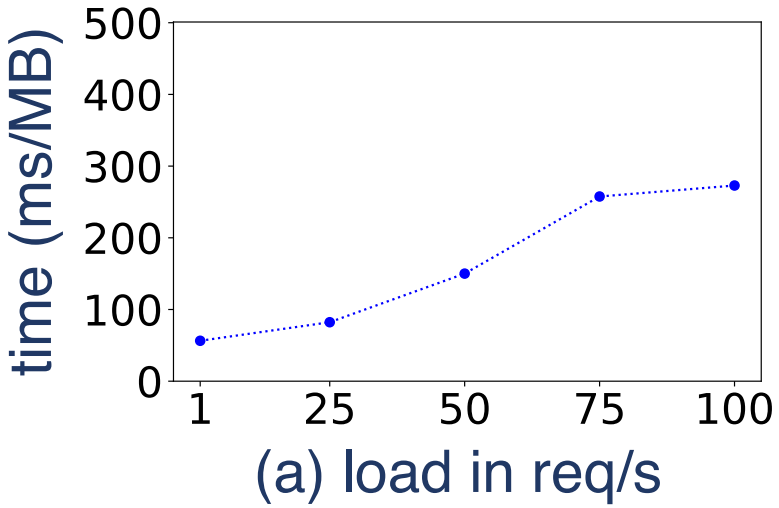
99.74% of generated failures were **successfully** explained

Returned solutions however contained also **false positives**, viz., failure cascades considered to have caused the failure even if this was not the case

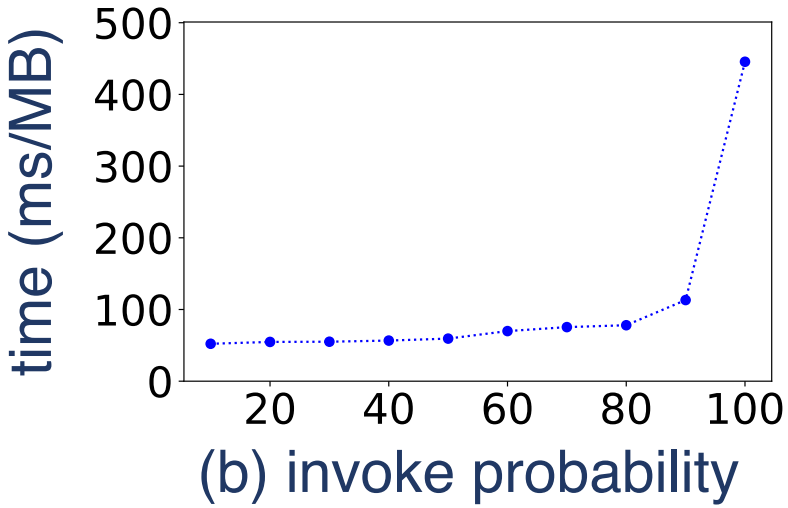
Q1) How many **false positives**?

Q2) How about **time**?

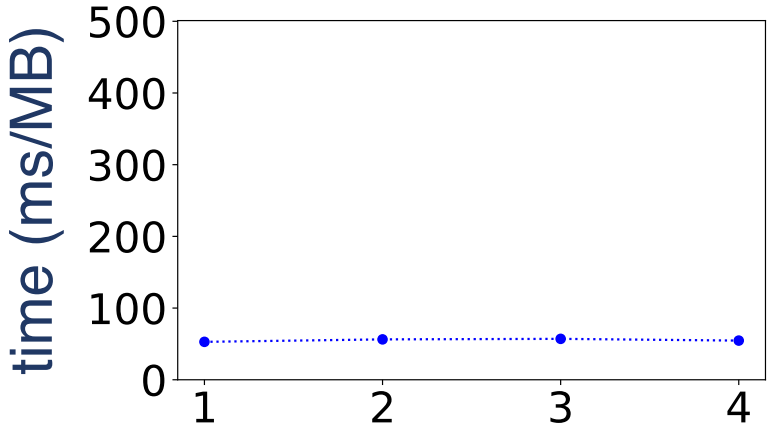
Results – Elapsed time



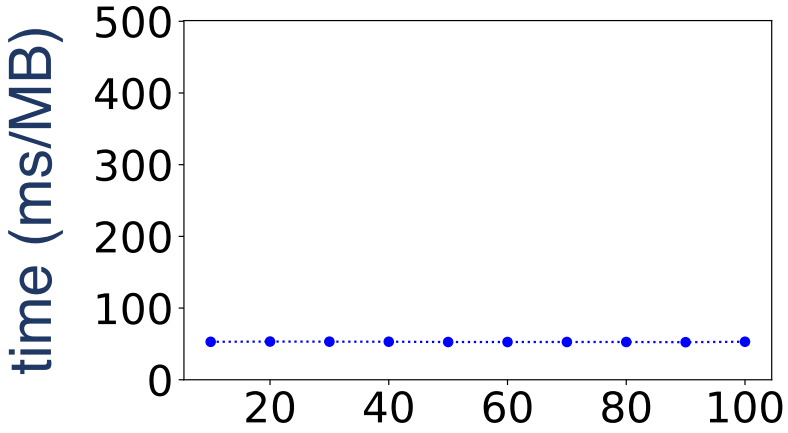
(a) load in req/s



(b) invoke probability



(c) cascade length



(d) fail probability

Conclusions

Our contributions

Declarative root cause analysis (RCA) technique for processing the events logged by microservice-based applications

yRCA, an open source **prototype** implementation of the proposed root cause analysis technique

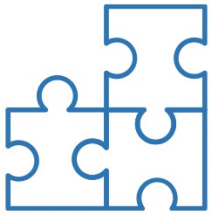
Controlled **experiments**, applying our technique to an existing **chaos testbed** (with 99.74% of generated failures successfully explained)

our declarative RCA can be used to **explain** solutions found with other RCA techniques

Future work

Assessment on industrial applications and/or based on different **chaos testing** approaches (e.g., Netflix's Chaos Monkey)

Graphical tool to visualise **failure cascades** and support reasoning on **countermeasures** to be enacted



Extend our declarative RCA to deal with **incomplete logs**, e.g., when logging driver fails or service get suddenly killed

Explainable Root Cause Analysis for Failing Microservices

Jacopo Soldani, Stefano Forti, and Antonio Brogi

Department of Computer Science, University of Pisa, Pisa, Italy


jacopo.soldani@unipi.it


http://pages.di.unipi.it/soldani