Explaining Cascading Failures in Microservices

Antonio Brogi

Department of Computer Science
University of Pisa

1. Software Products
2. Agile Software Engineering
3. Features, Scenarios and Stories
4. Software Architecture
5. Cloud-based Software
6. Microservices Architecture
7. Security and Privacy
8. Reliable Programming
9. Testing
10. DevOps and Code Management

8

Reliable Programming

"I focus here on techniques that help improve the overall reliability of a program

[…]

1. Fault avoidance - You should program in such a way that you avoid introducing faults into your program.

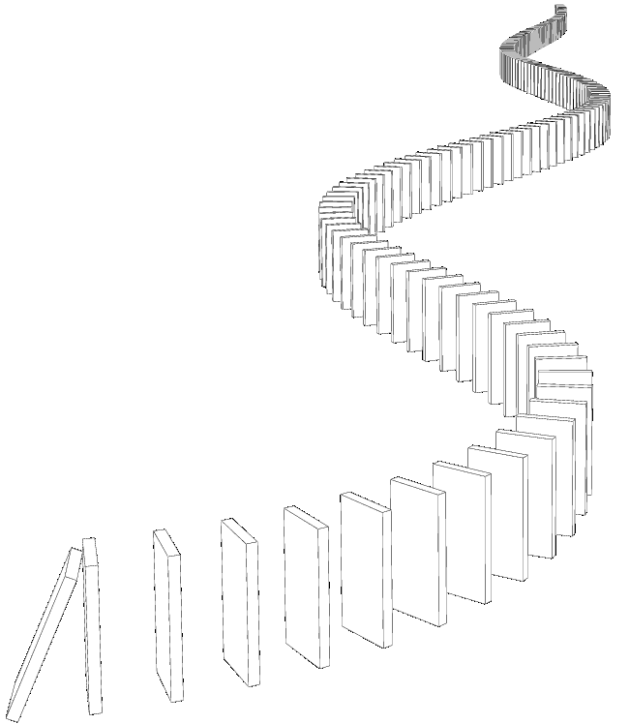
2. Input validation - You should define the expected format for user inputs and validate that all inputs conform to that format.

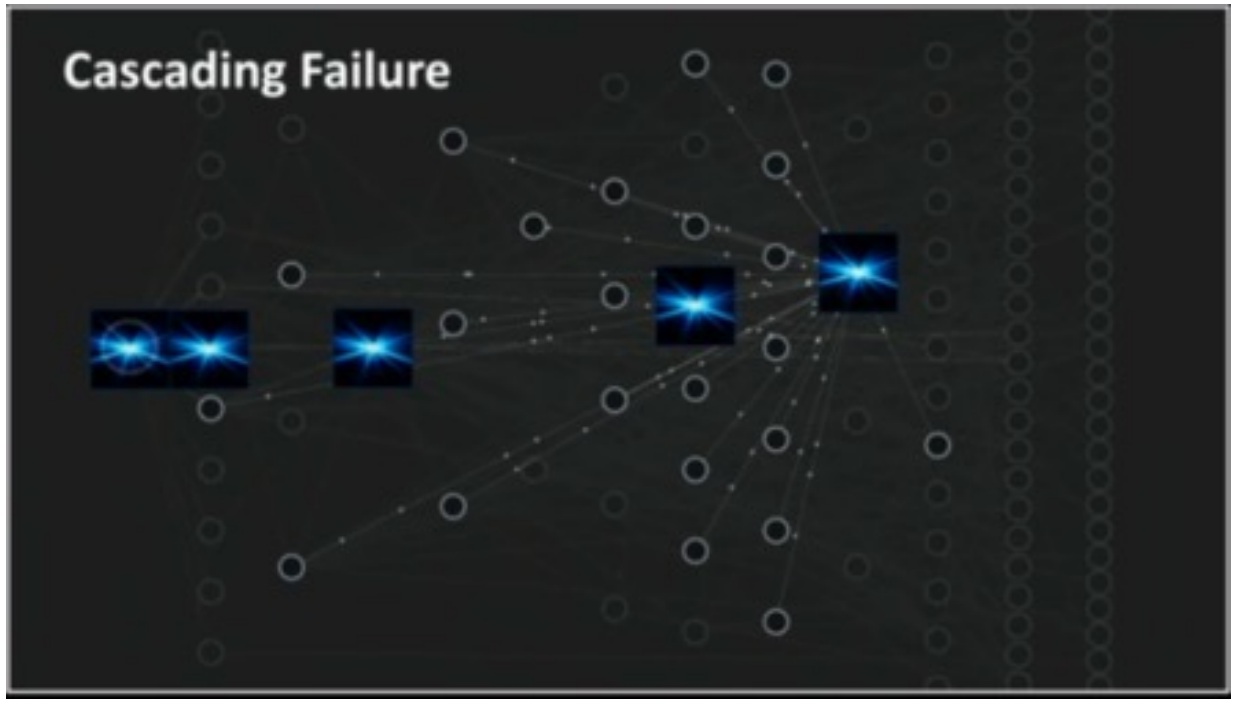
3. Failure management - You should implement your software so that program failures have minimal impact on product users."

Cascading failures and root causes

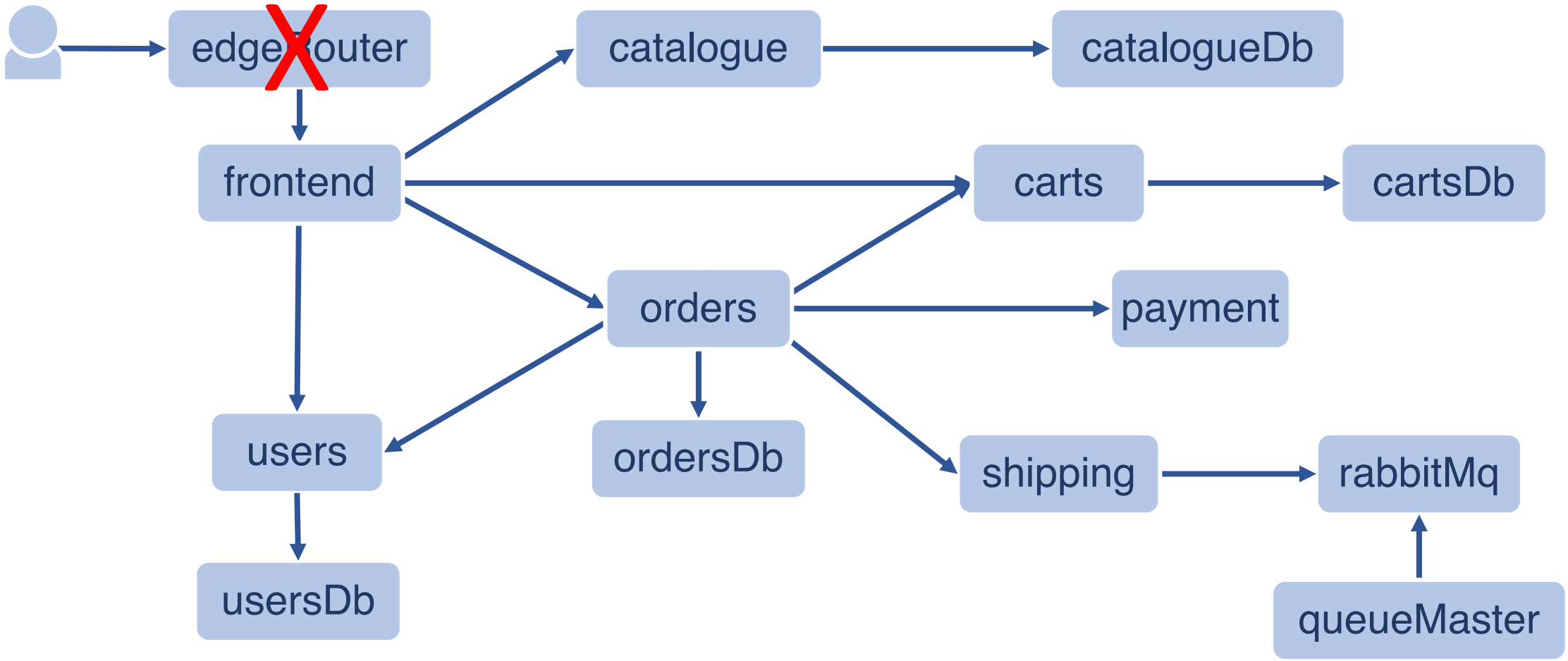
Cascading failures may occur in microservice-based applications

Determining the root cause of a cascading failure is crucial



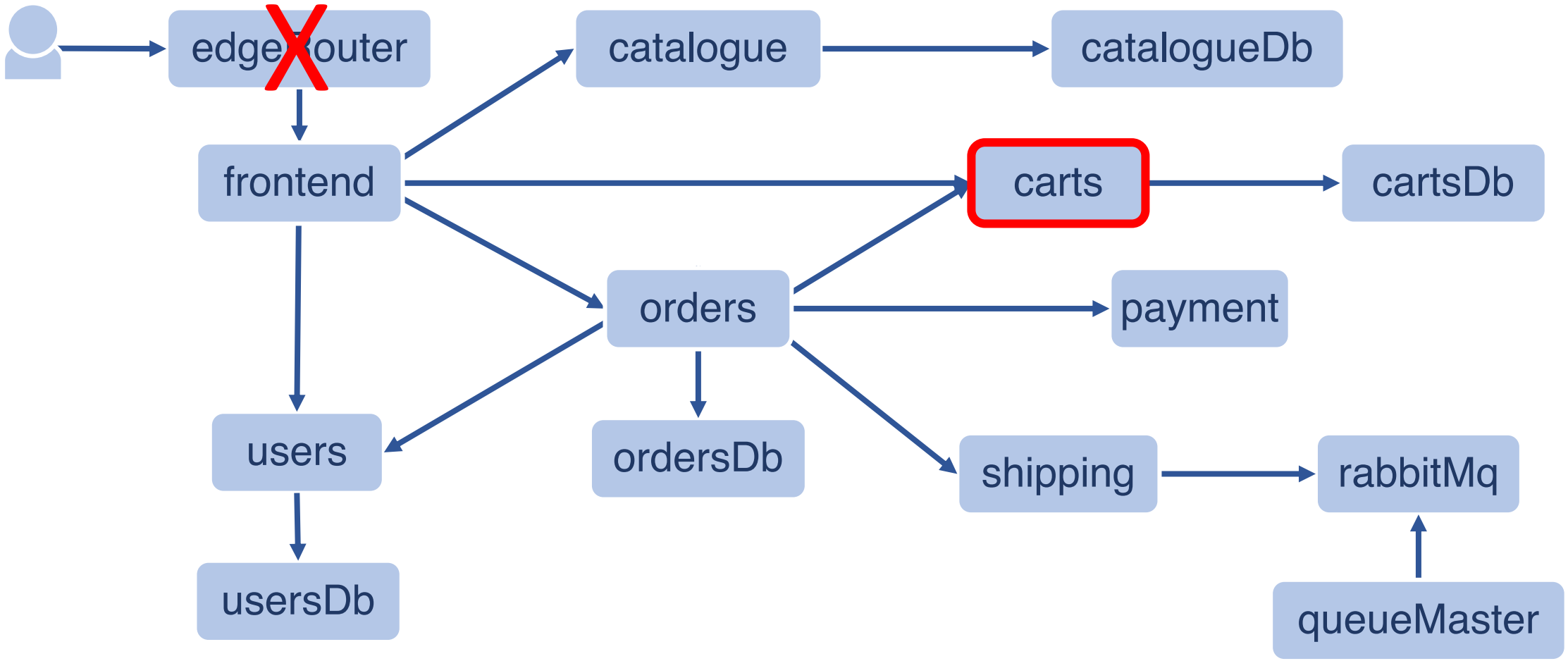


Example



edgeRouter service failed … why?

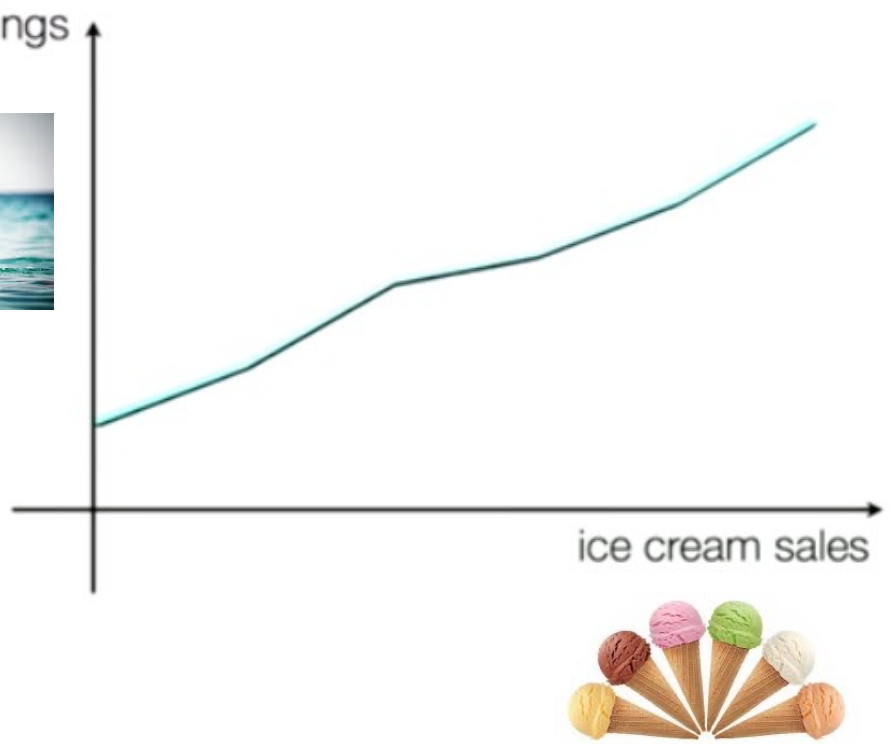
Example



ML-based approaches: … it was *carts*'s fault, most probably

Importance of explanations

Correlation is not causation



Yahoo's data center in Santa Clara witnessed a downtime for almost 12 hours in 2010
Squirrels chewed down the cables through which data got transferred

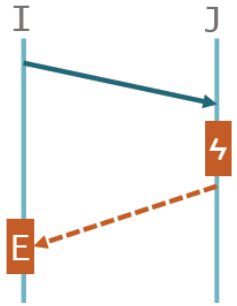
Importance of explanations (cont.)

The availability of **explanations** of root cause analyses would permit to avoid cascading failures by intervening

- both on the failing service and
- on (only) the services failing in cascade

A declarative approach to failure root cause analysis for microservices

1) Define causal relations between events



If *service I logged a failure/timeout E at the end of an interaction with J*

and *J logged an internal error during such interaction*

then *add E to the explanation and analyse the cause of the internal error at J*

If *service I logged an internal error E*

then *I is the root cause of E*

2) Let the inference engine work!

```
?- causedBy(event, Explanation, RootCause).
```

Logged events

Analysis of (distributed) applications logs containing

- The id of logging **service** (name and instance)
- A **timestamp**
- The logged **event**

```
internal                    // internal business logic
sendTo(Dst,Id)              // sent request
received(Id)                // received request
timeout(Dst,Id)             // expired timeout
errorFrom(Dst,Id)           // error reply returned
```

- And its (Syslog) **severity**

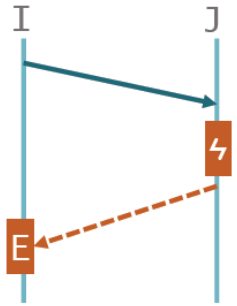
```
emerg/alert/crit/err        // errors
warning                     // warning
notice/info/debug           // info
```

Causal relations

Identify the causal relations among events occurring in separate service instances

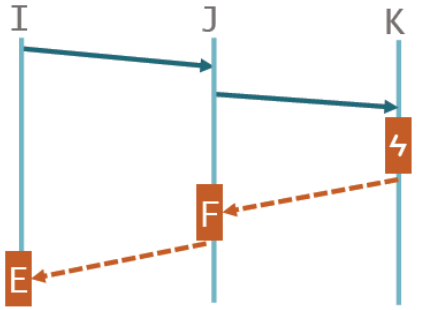
- 5 recursive cases
- 3 base cases

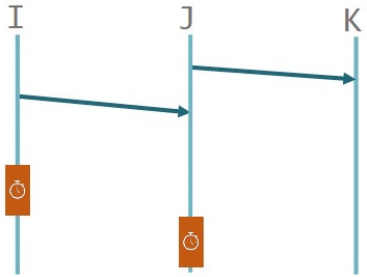
Recursive cases



Internal error of invoked service instance - *A failure/timeout event E at service I, occurring at the end of an interaction with J, was caused by an internal error at J. Recur to explain the internal error at J.*

Failed interaction of invoked service instance - *A failure/timeout event E at service I was caused by a failure event F at J, which in turn was caused by a failed interaction of J with K. Recur to explain the failure event at J.*

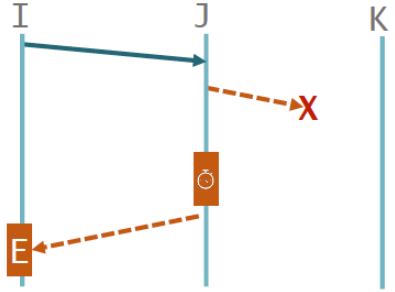


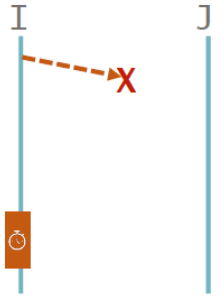


Timed-out interaction of invoked service instance - *A timeout event at service I was caused by a timeout at J, which in turn was caused by a timeout related to an interaction of J with K. Recur to explain the timeout at J.*

Recursive cases (cont.)

Unreachability of a service called by invoked service instance *– A failure/timeout event E at service I was caused by a timeout at J, which occurred since the request sent by J was never received by K. Recur to explain the timeout at J.*





Unreachability of invoked service instance *- A timeout event at I was caused by a non-received request in an interaction of I with J. Recur to explain why J was unreachable.*

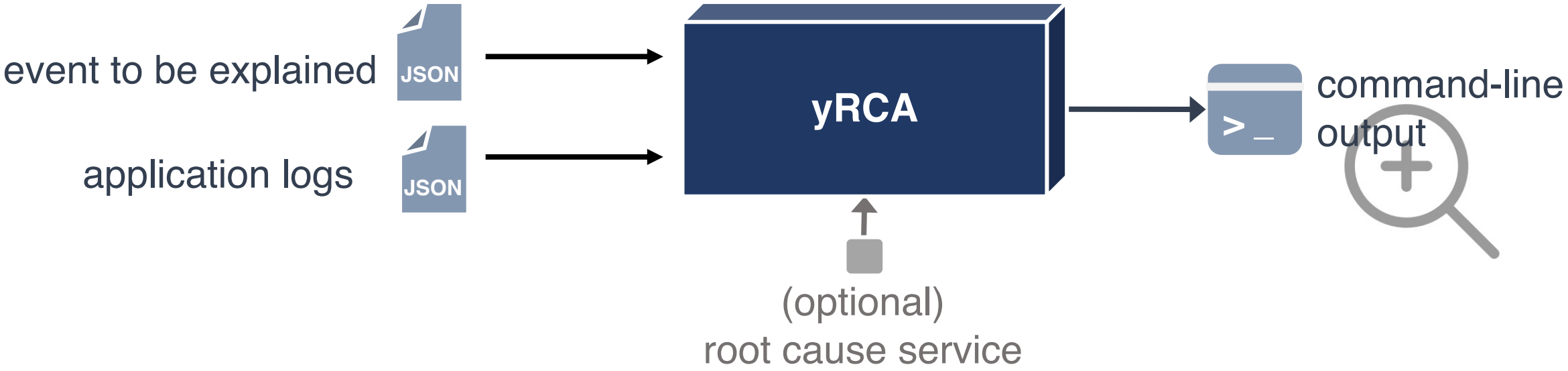
Base cases

Internal service error *- Internal failure event logged by a service, identifying the service itself as the root cause for such an event.*

Temporary service unreachability *- Service considered temporarily unreachable as it previously logged some information.*

Unstarted service *- Service never logged any information.*

yRCA prototype



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

yRCA on the example

```
[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
```

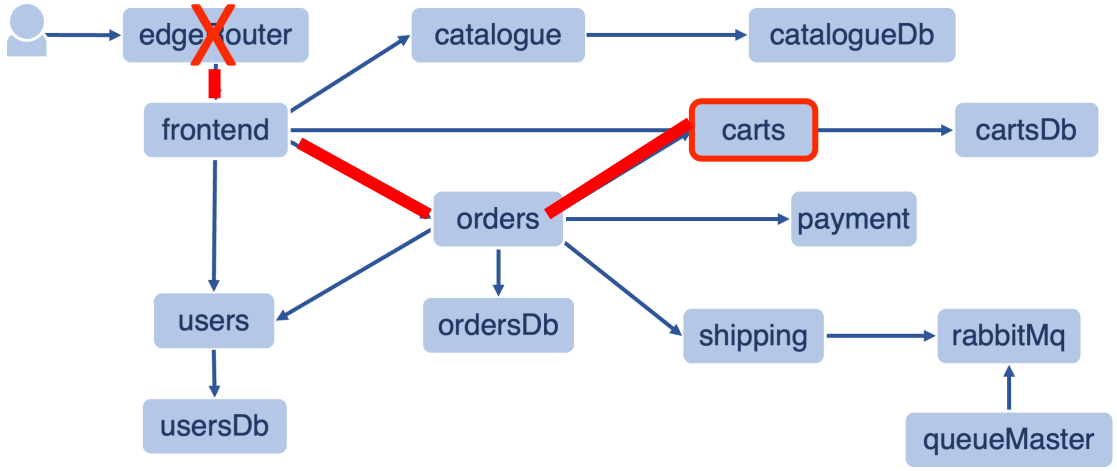
probability

root cause

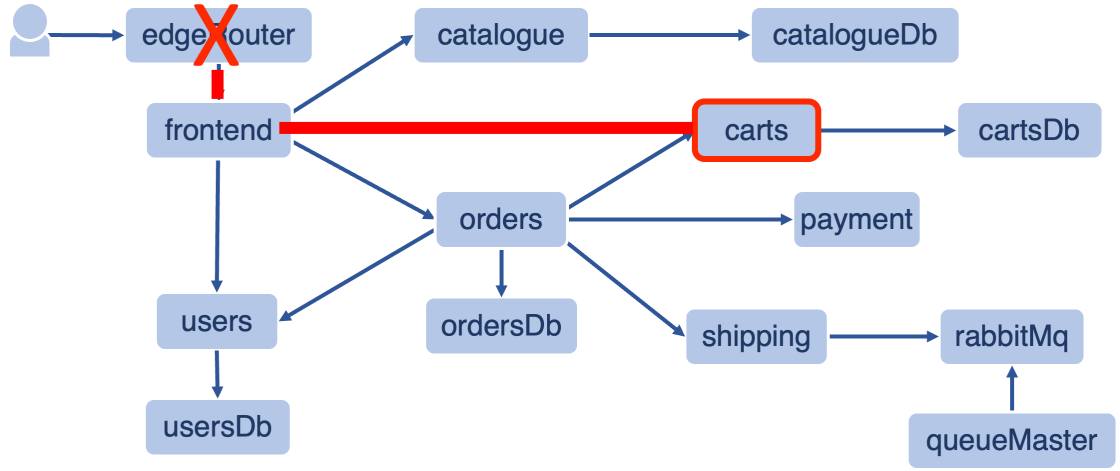
explanation
(cascading failures)

yRCA on the example

61.5%



38.5%



yRCA assessment

Controlled experiments (with Chaos Echo testbed)

- End-user **load** varying from 1 to 100 requests per second
- Service **interaction probability** varying from 0.1 to 1
- Failure **cascade length** varying from 1 to 4

} only one service set to fail on its own, giving a **ground thruth**

- Service **failure probability** varying from 0.1 to 1

} any service can fail

Successfull detection of failure root cause: 99.74% of cases

Low number of false positive (3 explanations in the worst case)

Conclusions

+ Declarative root cause analysis (RCA) technique capable of

  (1) determining root causes and of

  (2) explaining cascading failures

  in microservice-based applications

+ Open source prototype implementation available

+ Good experimental assessment

Future work

- Assessment with industrial applications and with other chaos testing approaches
- Graphical tool visualising cascading failures and suggesting countermeasures
- Deal with with incomplete logs

J. Soldani, S. Forti, A. Brogi. Failure Root Cause Analysis for Microservices, Explained. 22nd International Conference on Distributed Applications and Interoperable Systems (DAIS 2022).