Microservices beyond COVID-19

Antonio Brogi

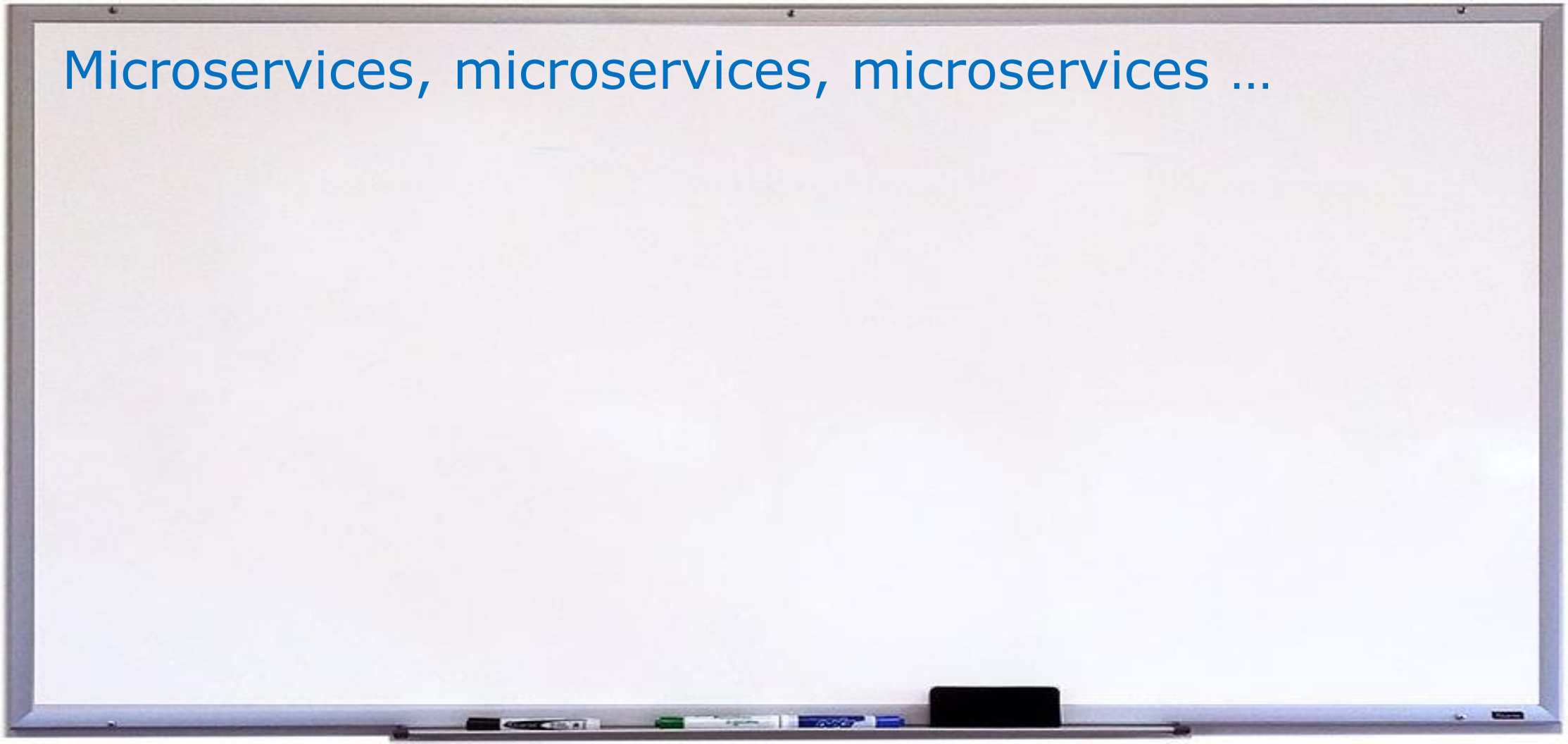
Department of Computer Science
University of Pisa, Italy

Q: Beyond COVID-19?

Nick

A: Sorry, just a dirty trick to attract audience :)

15:00 Coffee Break
15:30 Keynote: Microservices beyond COVID-19
16:30 Closing


Microservices, microservices, microservices …

Microservices

Main motivations

(1) Shorten lead time for new features/updates

- accelerate rebuild and redeployment
- reduce chords across functional silos

(2) Need to scale, effectively

- millions of users

amazon Google NETFLIX Spotify

LinkedIn twitter ebay

Uber zalando GROUPON ...

OK but … what are microservices?

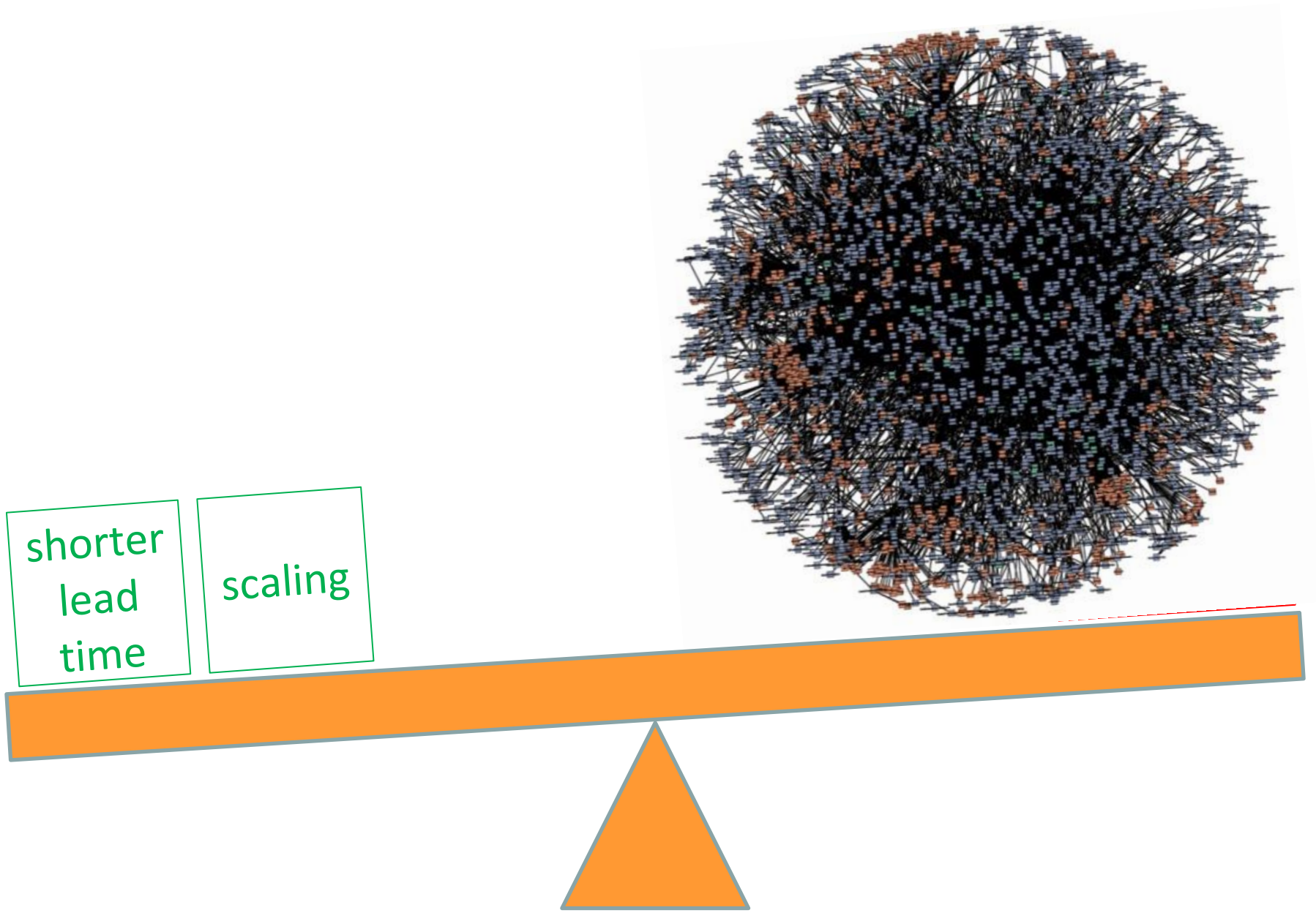
Microservices

Applications = sets of services

+ each running in its own ~~process~~ container
+ communicating with lightweight mechanisms
+ built around business capabilities
+ decentralizing data management
+ independently deployable
+ horizontally scalable
+ fault resilient
+ DevOps culture and tools!

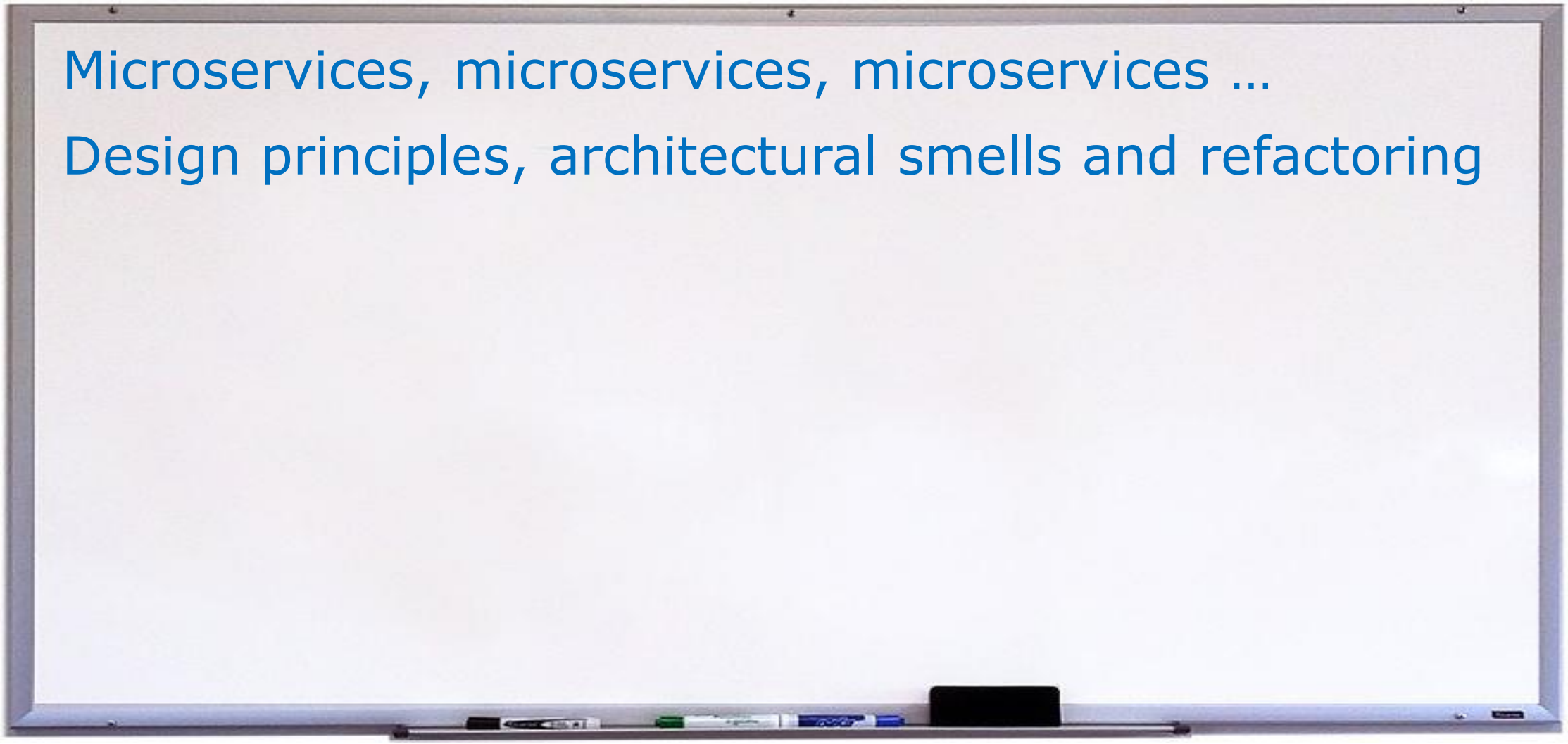
(service-orientation done right?)

Microservices



Does my app respect the "microservices principles"?

If not, how can I refactor it?


Microservices, microservices, microservices …
Design principles, architectural smells and refactoring

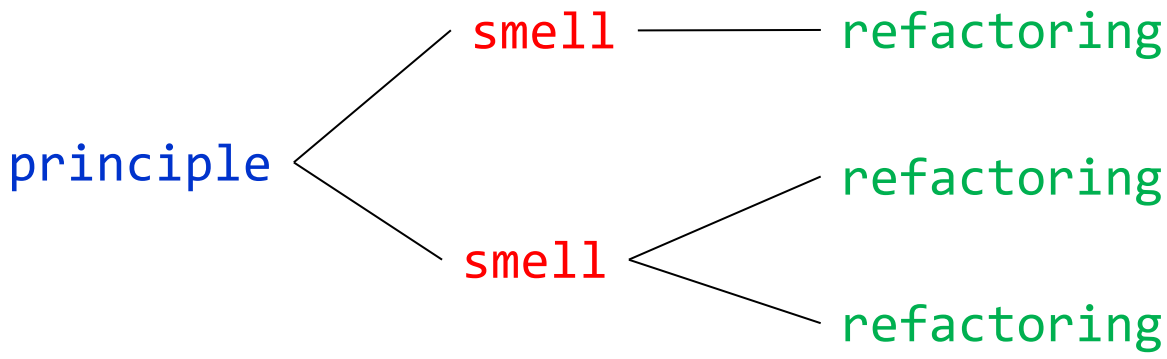
Question

How can architectural **smells** affecting design **principles** of microservices be detected and resolved via **refactoring**?

A multivocal review

Recent review of white and grey literature aimed at identifying

- the most recognised ***architectural smells*** for microservices, and
- the architectural ***refactorings*** to resolve them



(review of 41 studies presenting architectural smells & refactorings for resolving them)

D. Neri, J. Soldani, O. Zimmermann, A. Brogi. ***Design principles, architectural smells and refactorings for microservices: A multivocal review.*** *Software-Intensive Cyber-Physical Systems. 2020.*

Design principles

Independent deployability

The microservices forming an application should be independently deployable

Horizontal scalability

The microservices forming an application should be horizontally scalable

[= possibility of adding/removing replicas of single microservices]

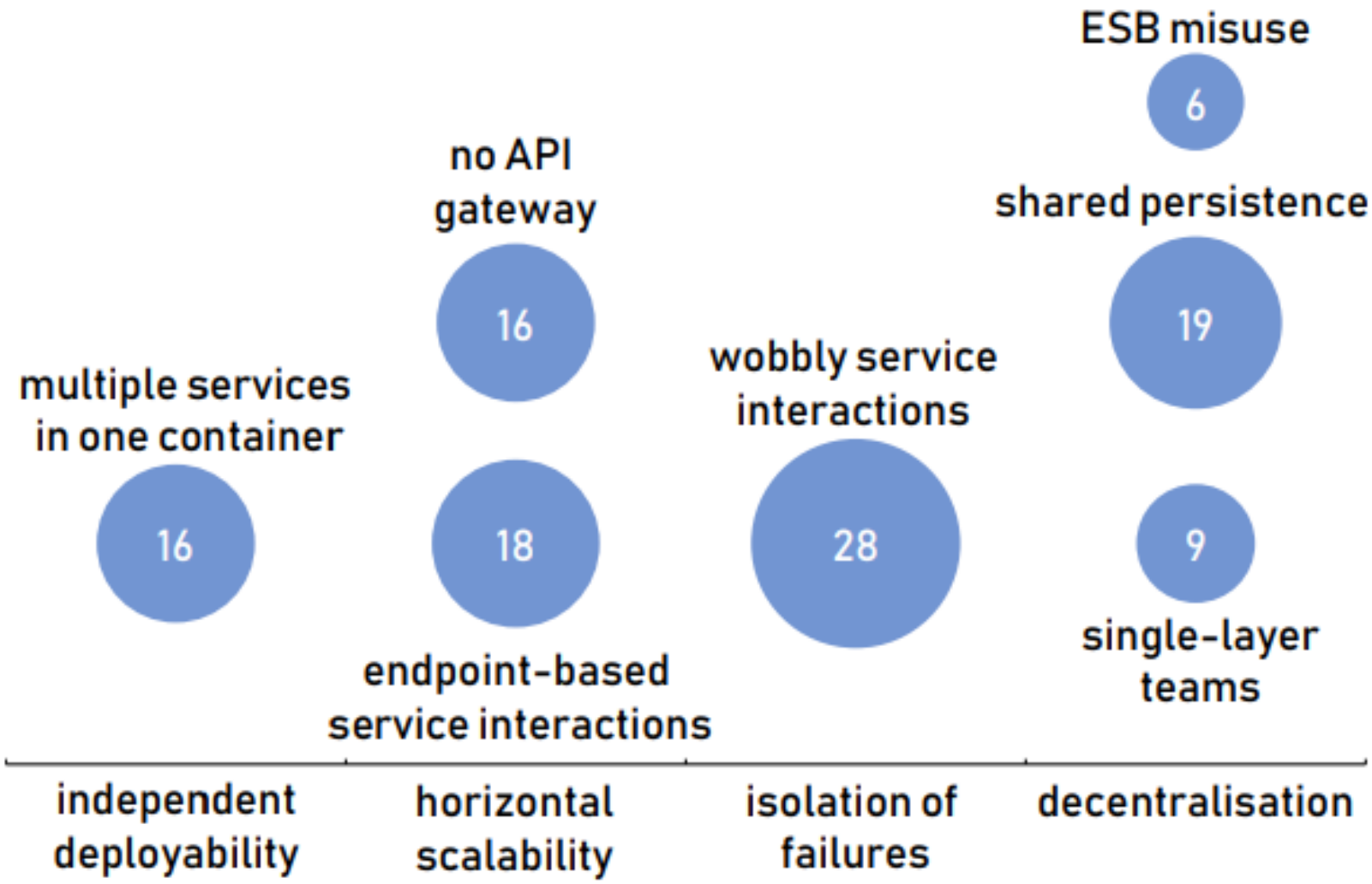
Isolation of failures

Failures should be isolated

Decentralization

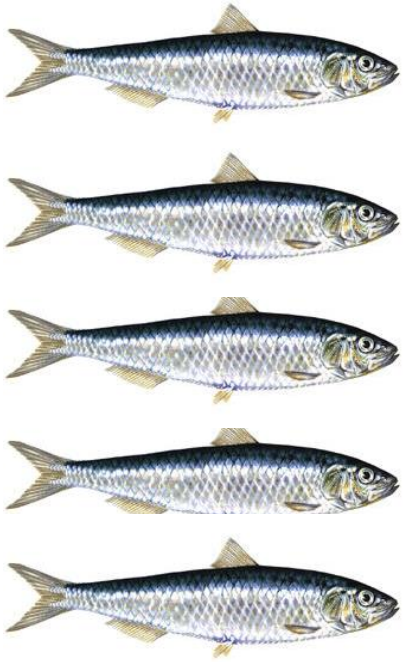
Decentralisation should occur in all aspects of microservice-based applications, from data management to governance

Architectural smells

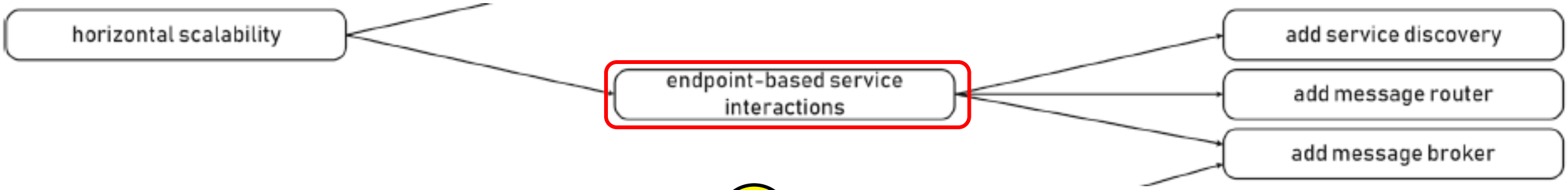


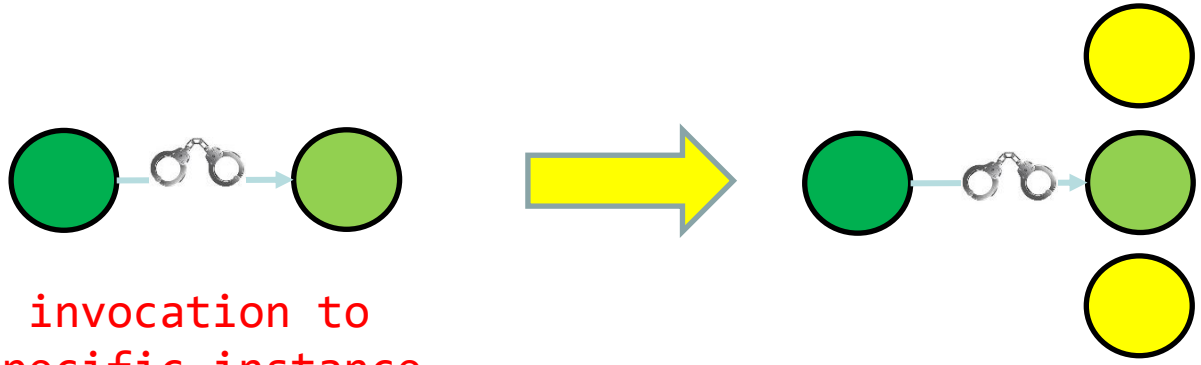
Multiple services in one container

independent deployability → multiple services in one container → package each service in a separate container



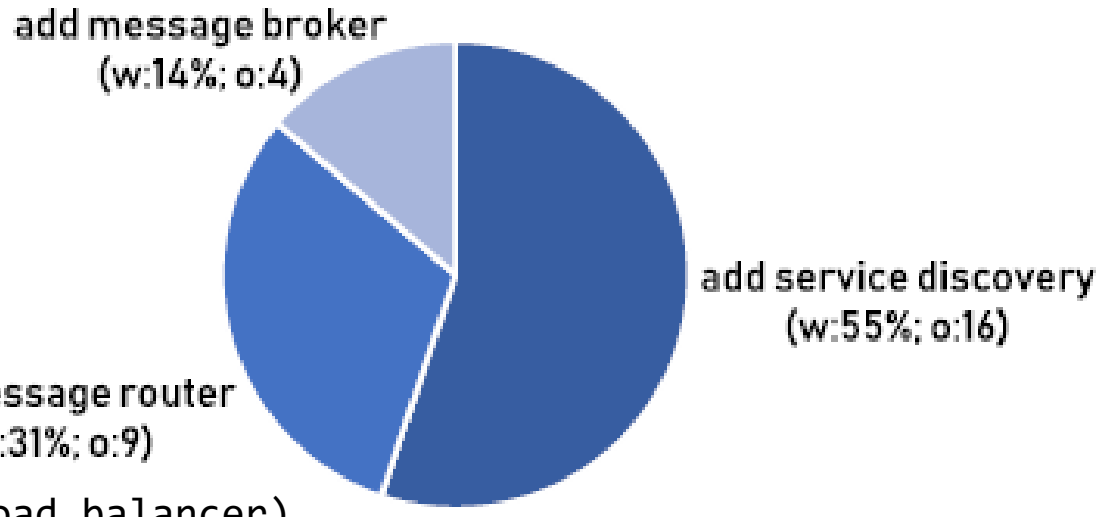
Endpoint-based service interactions





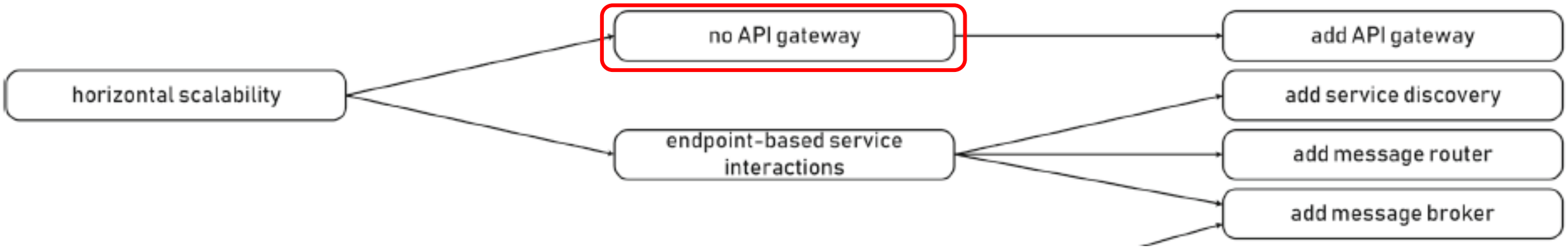
invocation to specific instance

(e.g. message queue)



(e.g. load balancer)

No API gateway



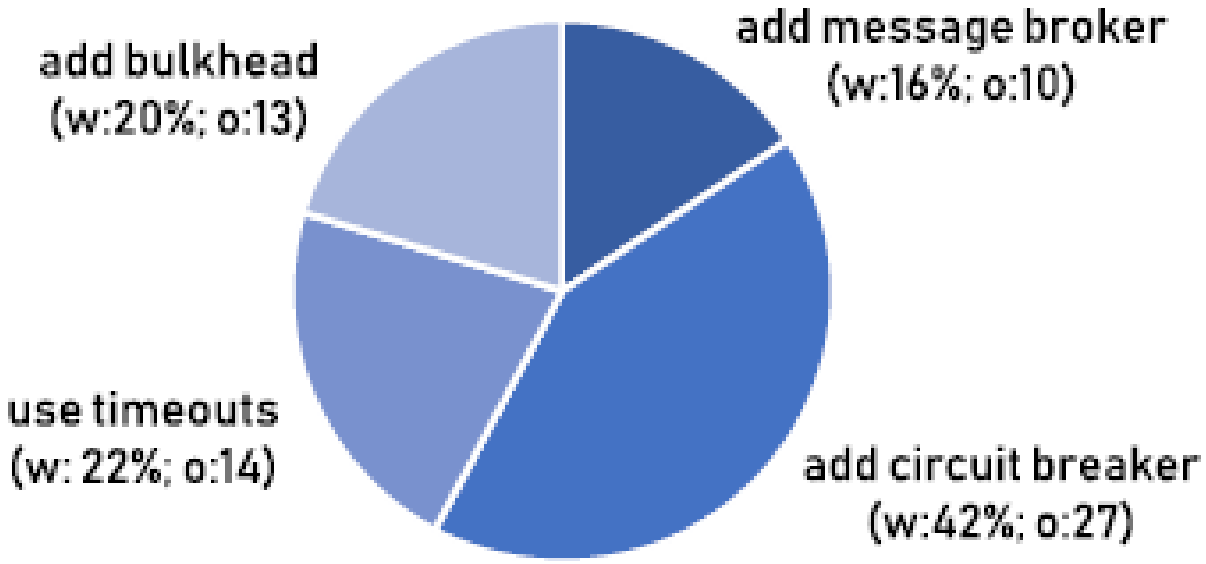
App clients must invoke directly app services

(similar to endpoint-based service interaction smell)

Refactoring: add API gateway (that can be useful also for authentication, throttling, …)

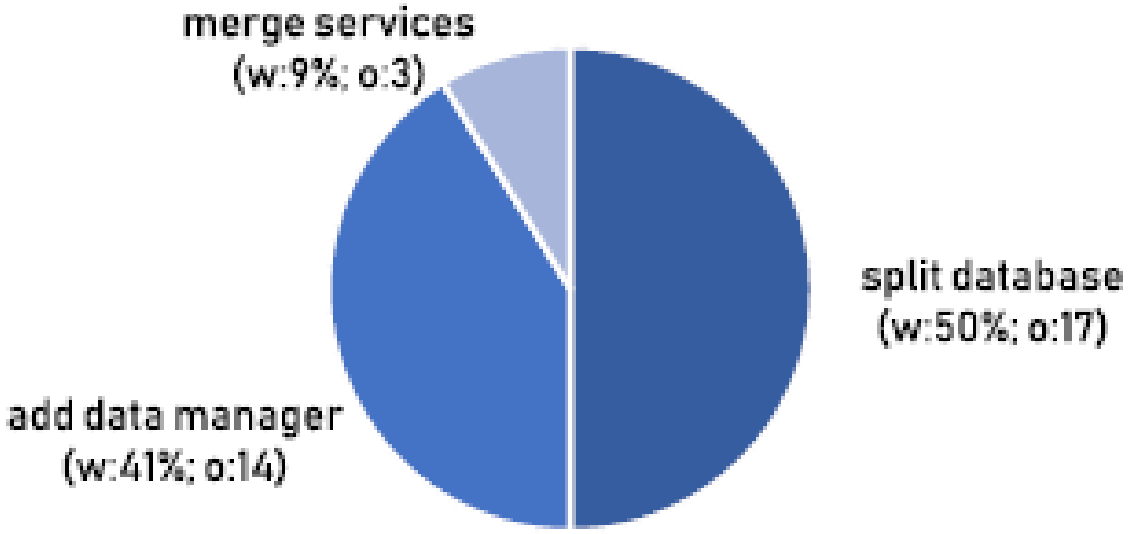
Wobbly service interactions

The interaction of m1 with m2 is *wobbly* when a failure of m2 can trigger a failure of m1

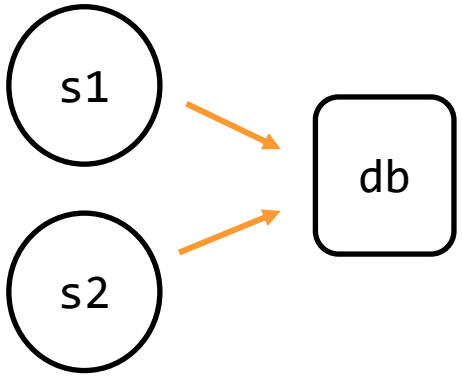


Shared persistence

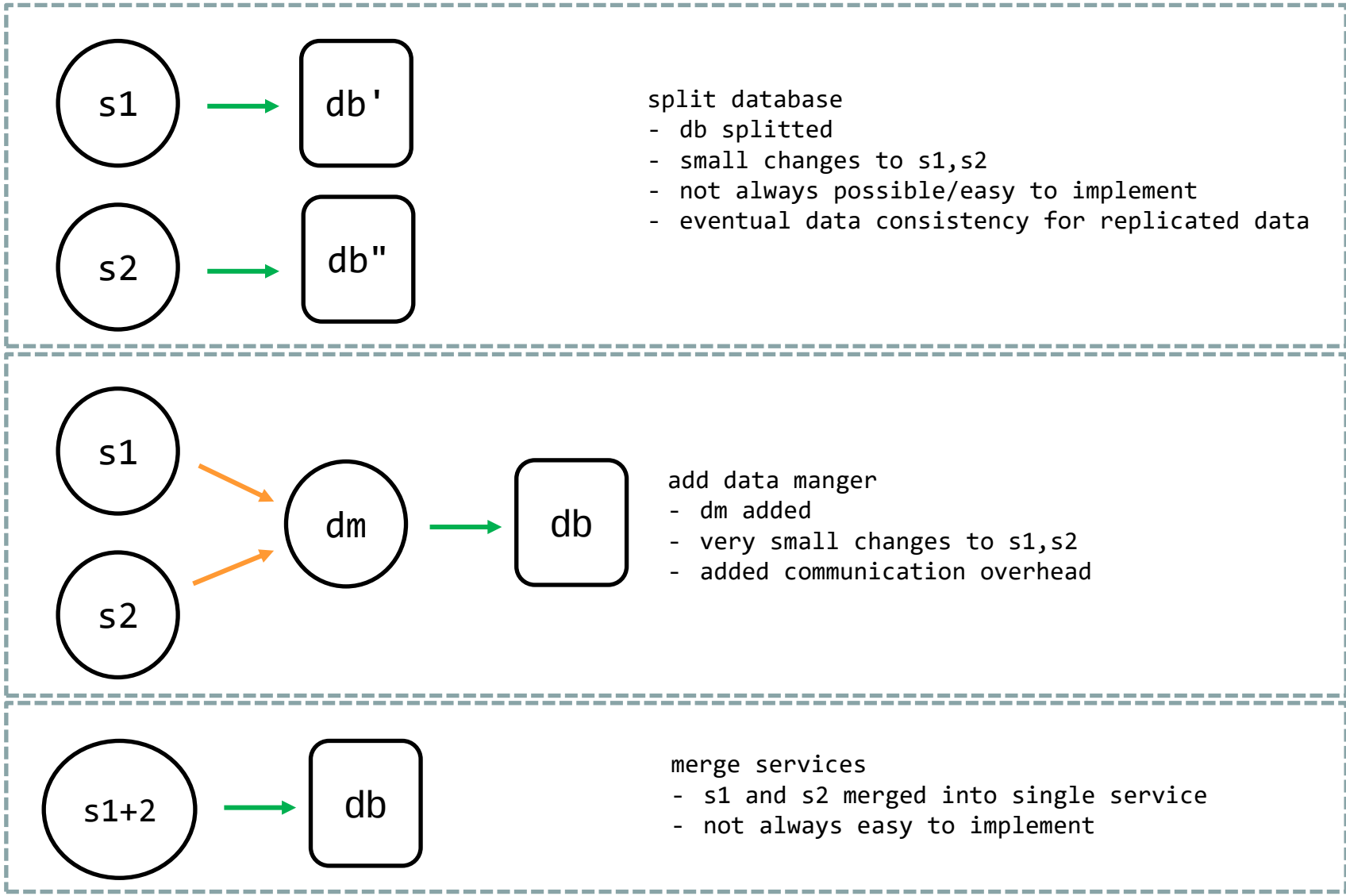
Multiple services access/manage the same DB



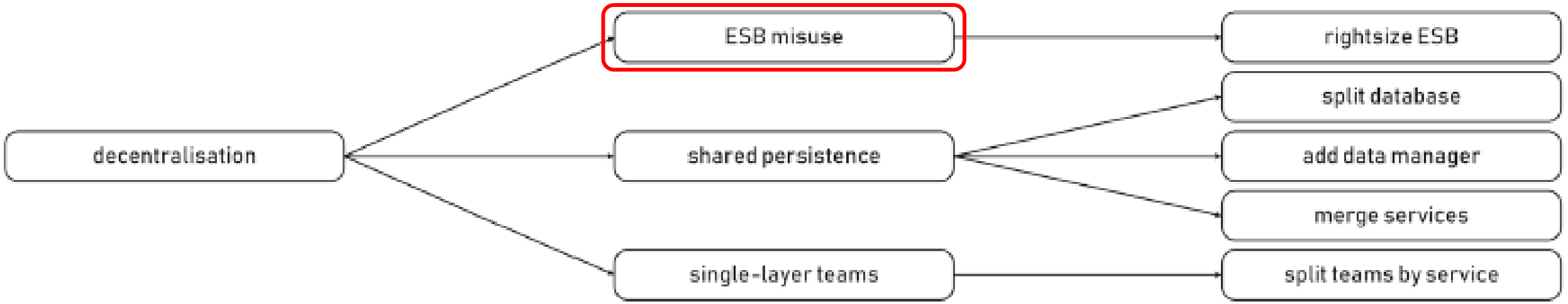
Shared persistence



db shared by multiple services



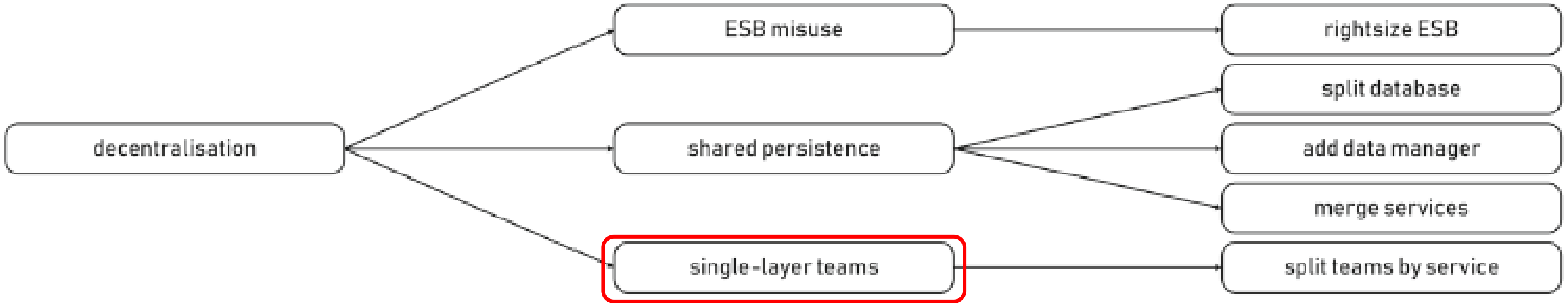
ESB misuse

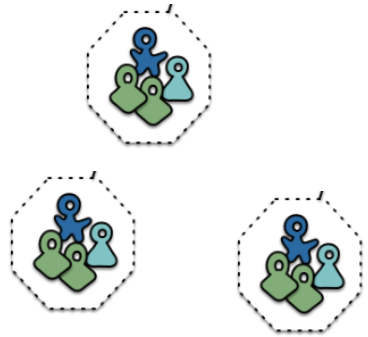


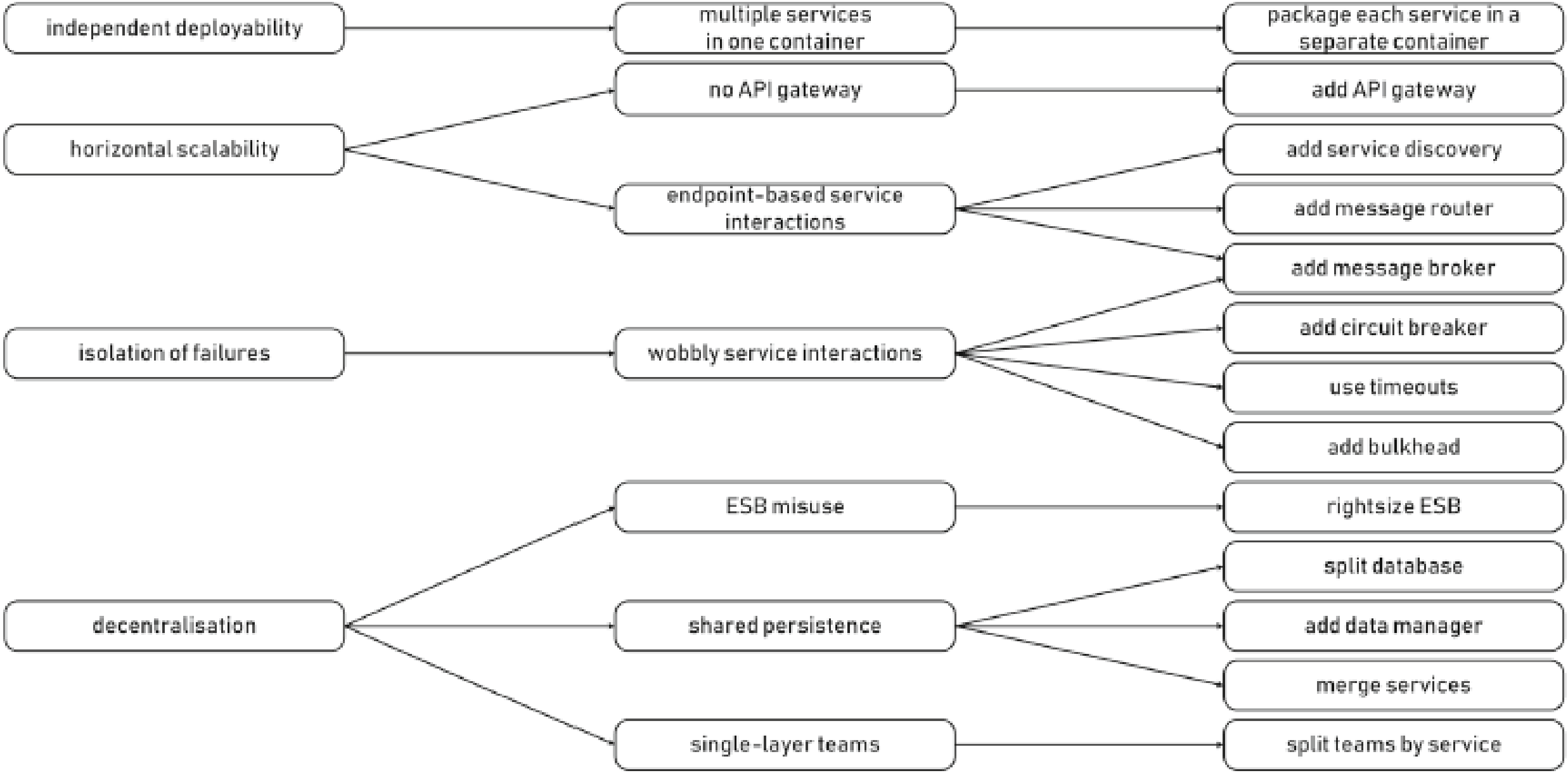
ESB misuse may lead to
undesired centralisation of
business logic and dumb services

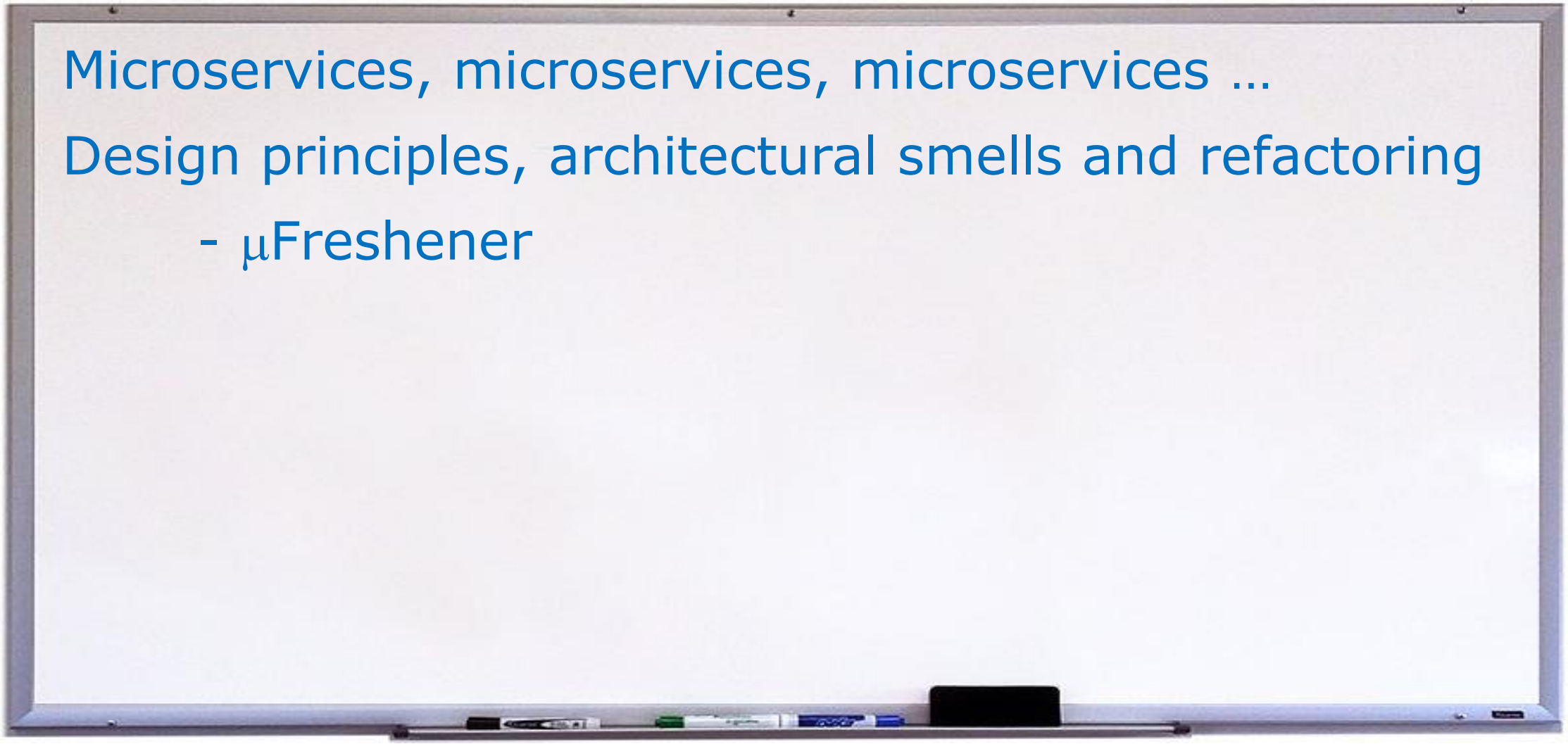
Smart endpoints & dumb pipes !

Single-layer teams






independent deployability
horizontal scalability
isolation of failures
decentralisation
multiple services in one container
no API gateway
endpoint-based service interactions
wobbly service interactions
ESB misuse
shared persistence
single-layer teams
package each service in a separate container
add API gateway
add service discovery
add message router
add message broker
add circuit breaker
use timeouts
add bulkhead
rightsize ESB
split database
add data manager
merge services
split teams by service


Microservices, microservices, microservices …
Design principles, architectural smells and refactoring
- μFreshener

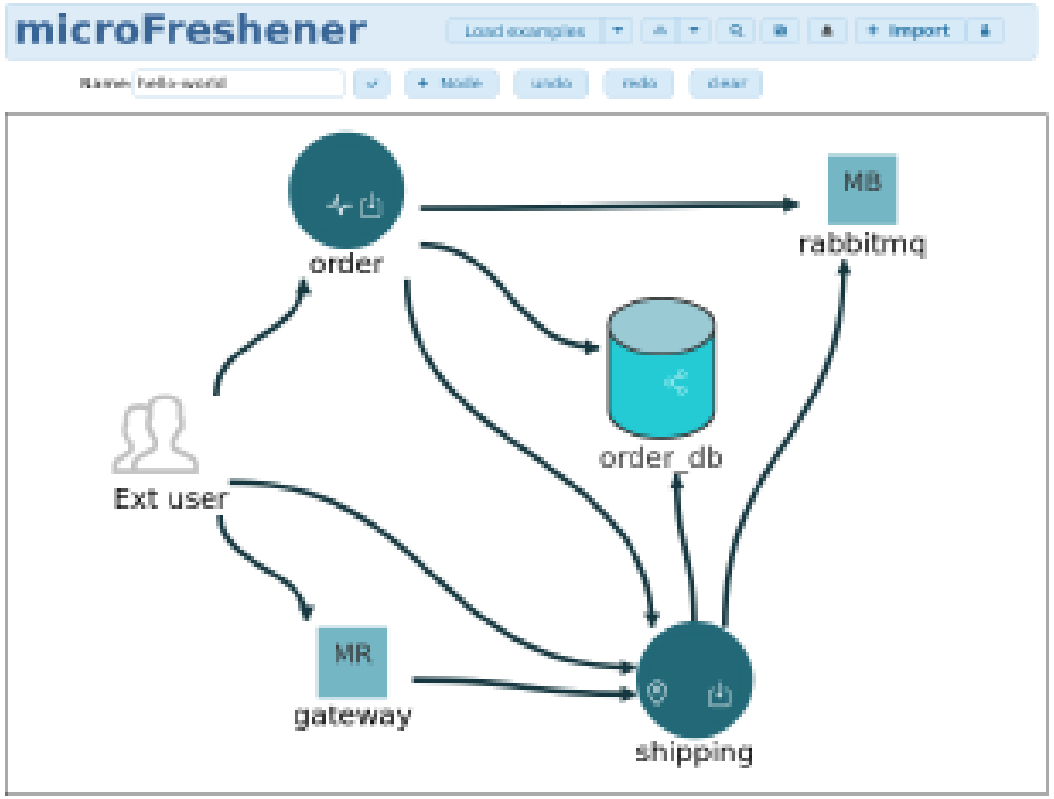
μFreshener

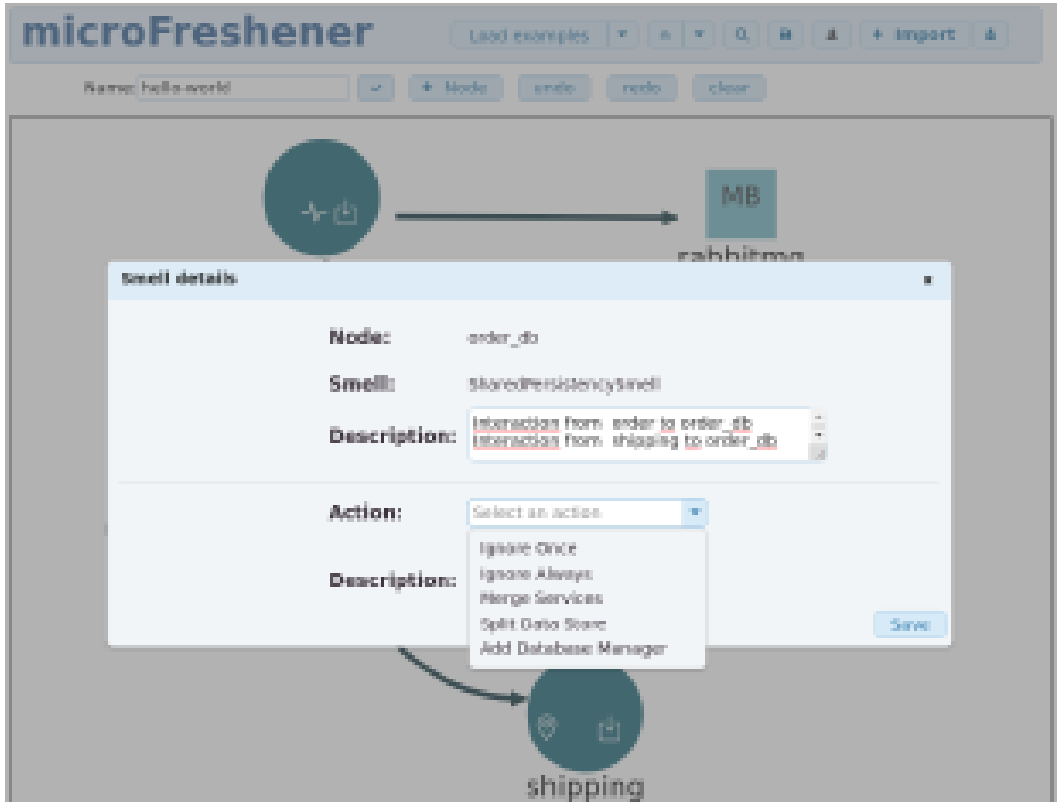
A web-based GUI for

- editing app specifications
- automatically identifying architectural smells
- applying architectural refactorings to resolve the identified smells

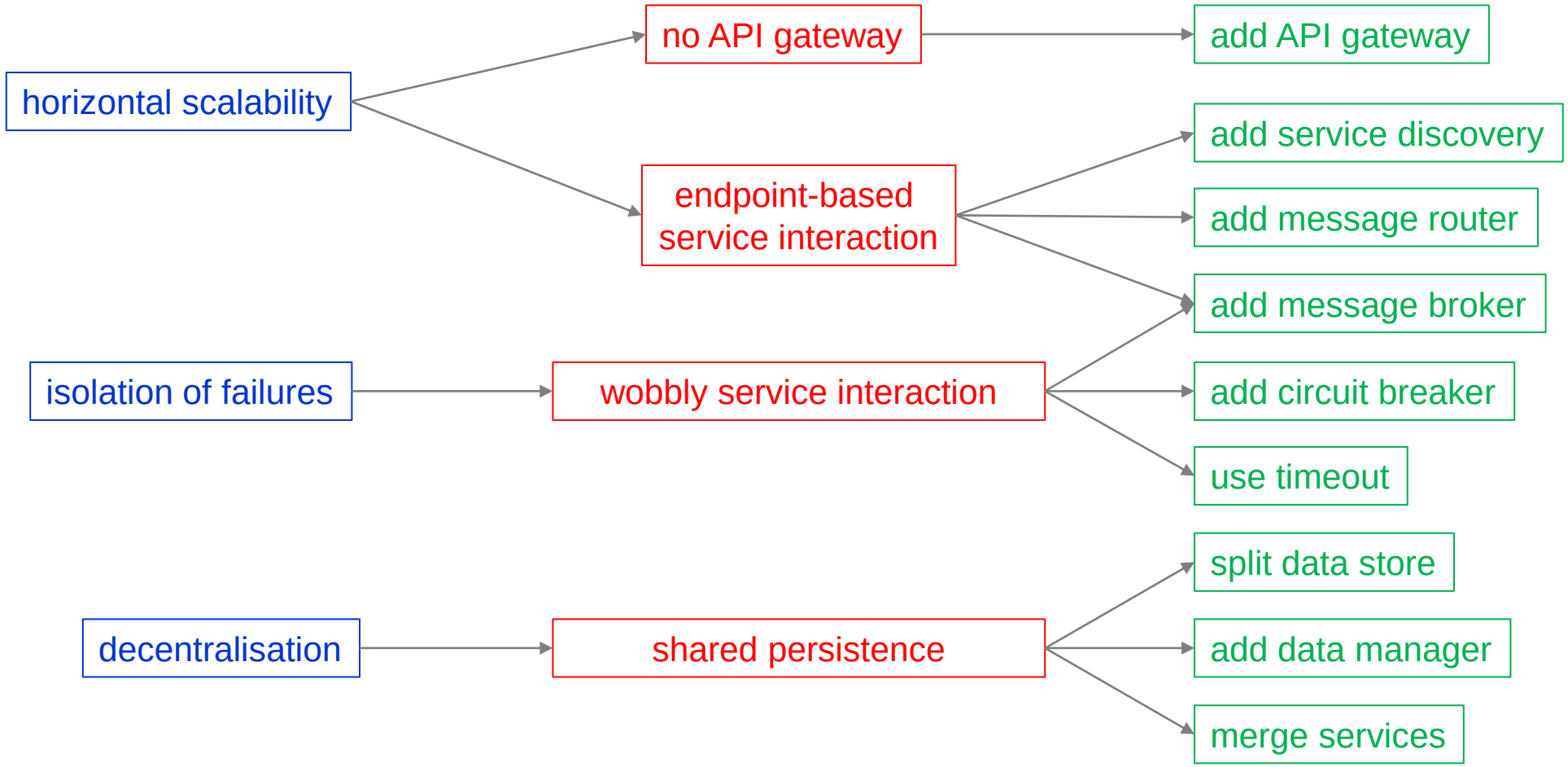
A. Brogi, D. Neri, J. Soldani. ***Freshening the air in microservices: Resolving architectural smells via refactoring****. WESOACS 2019.*

μFreshener


microFreshener
Load examples
Import
Name: hello-world
Node
undo
redo
clear
order
MB
rabbitmq
order_db
Ext user
MR
gateway
shipping


microFreshener
Load examples
Import
Name: hello-world
Node
undo
redo
clear
MB
Smell details
Node:
order_db
Smell:
SharedPersistencySmell
Description:
Interaction from order to order_db
Interaction from shipping to order_db
Action:
Select an action
Description:
Ignore Once
Ignore Always
Merge Services
Split Data Store
Add Database Manager
Save
shipping

Excerpted principle-smell-refactoring taxonomy



Modelling application architecture

Graphical representation (of μTOSCA model)

OASIS

service

data store

mB message broker

mR message router

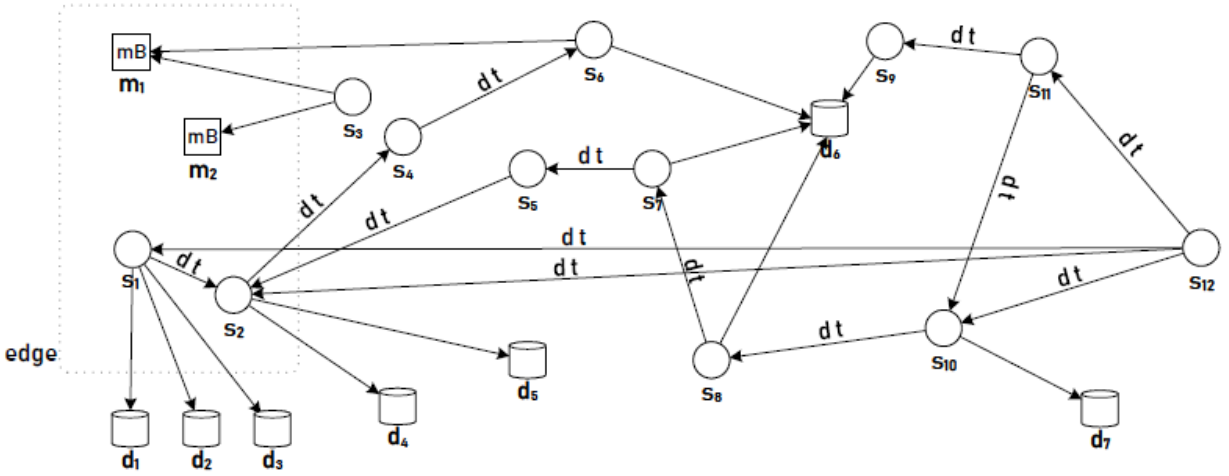
edge

d dynamic discovery

c circuit breaker

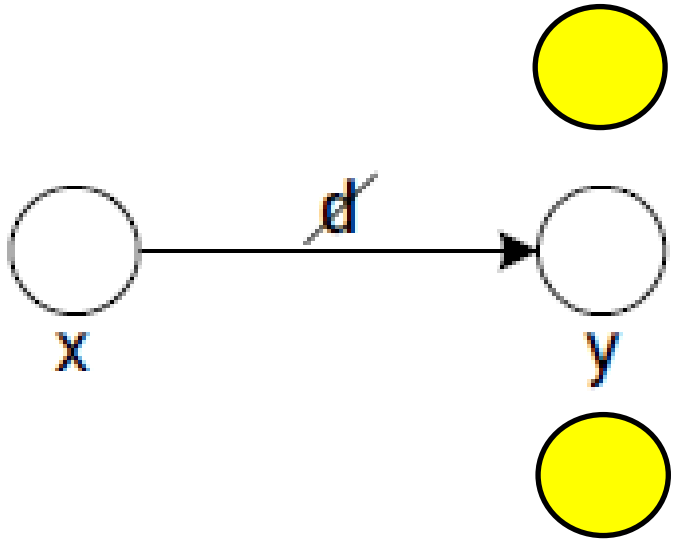
t timeout

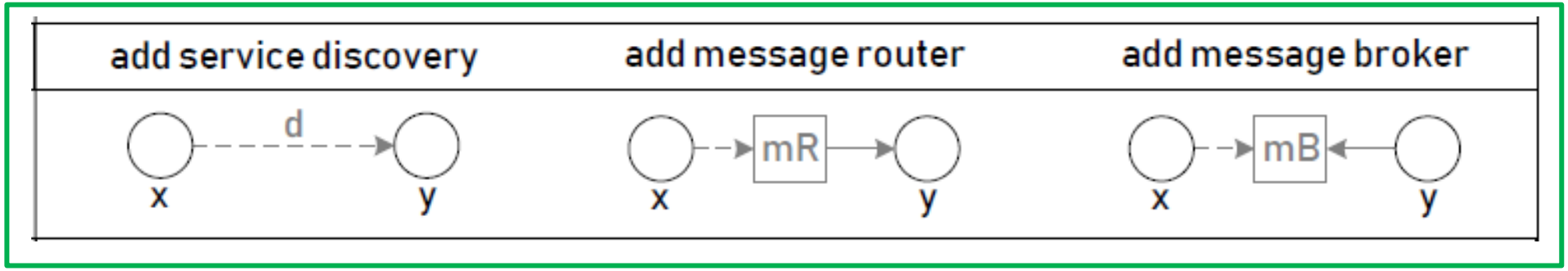
Example



µFreshener: horizontal scalability

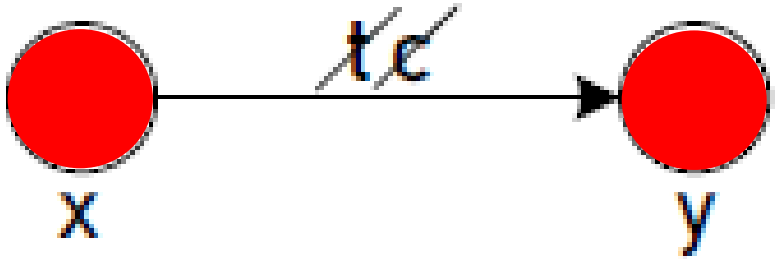
endpoint-based service interaction

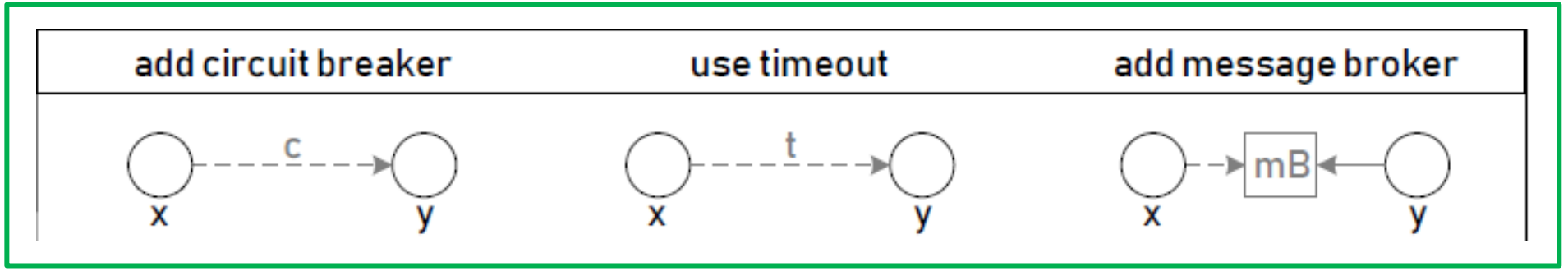




µFreshener: isolation of failures

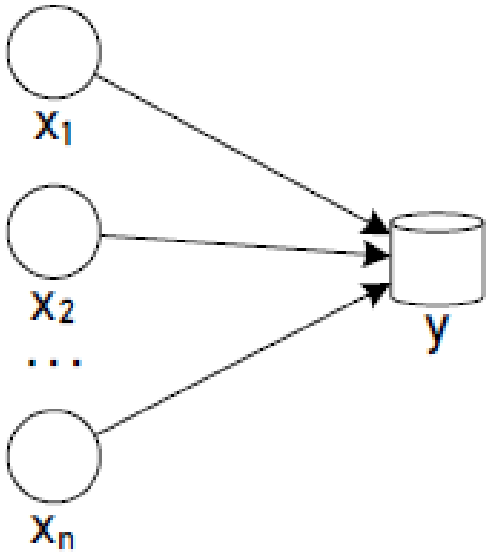
wobbly service interaction

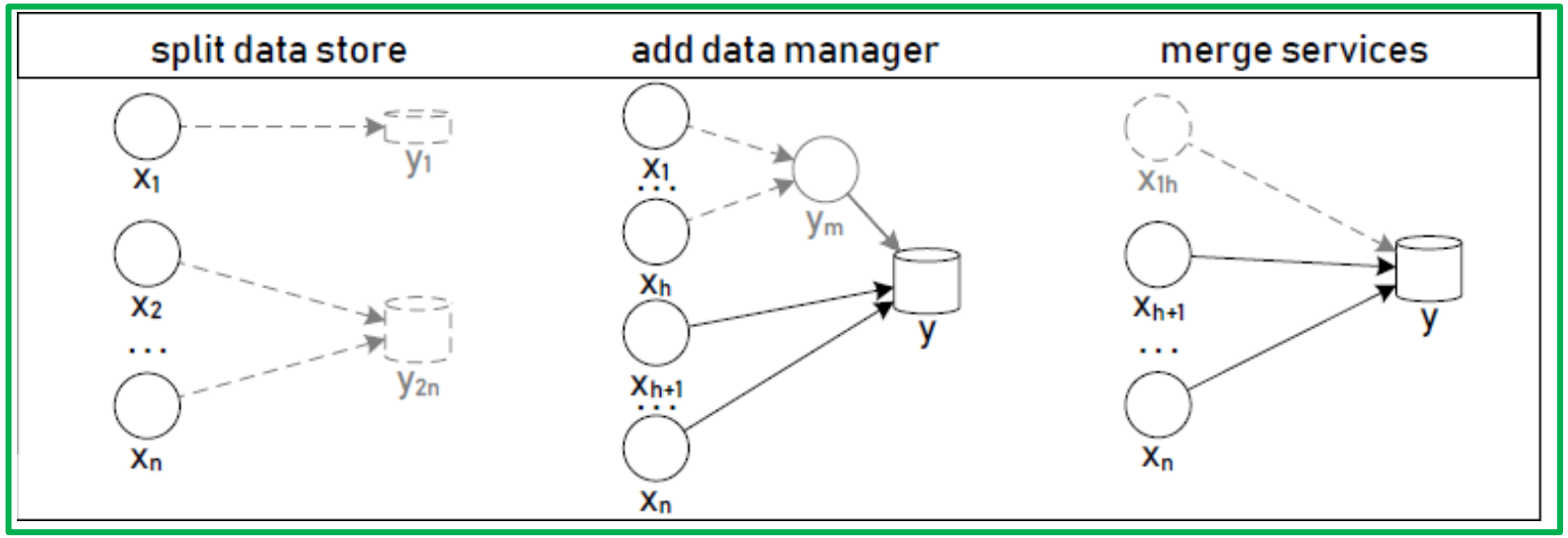




µFreshener: decentralisation

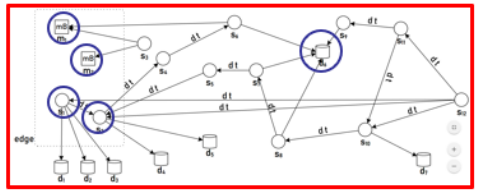
shared persistence



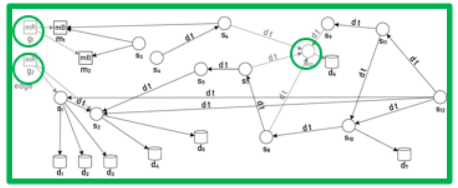


Remarks 1/2

- mFreshener (freely) usable to analyse & refactor microservice-based apps
  - industrial case study



- 4 no API gateway smells
- 1 shared persistence smell



- 2 API gateways added
- 1 data manager added

  - controlled experiment (100% vs. 49% smells identified, 83% vs. 1% resolved all smells)

- a smell is not necessarily a principle violation

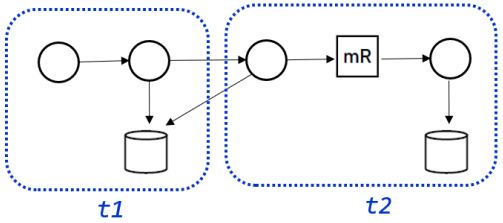
≠

- "let it be" refactoring supported

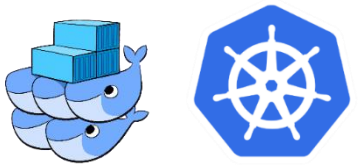
Remarks 2/2

- μFreshener works at the architecture level
  concrete implementation of refactoring left to application manager – much like in design patterns

- scalability: μFreshener features team-based view

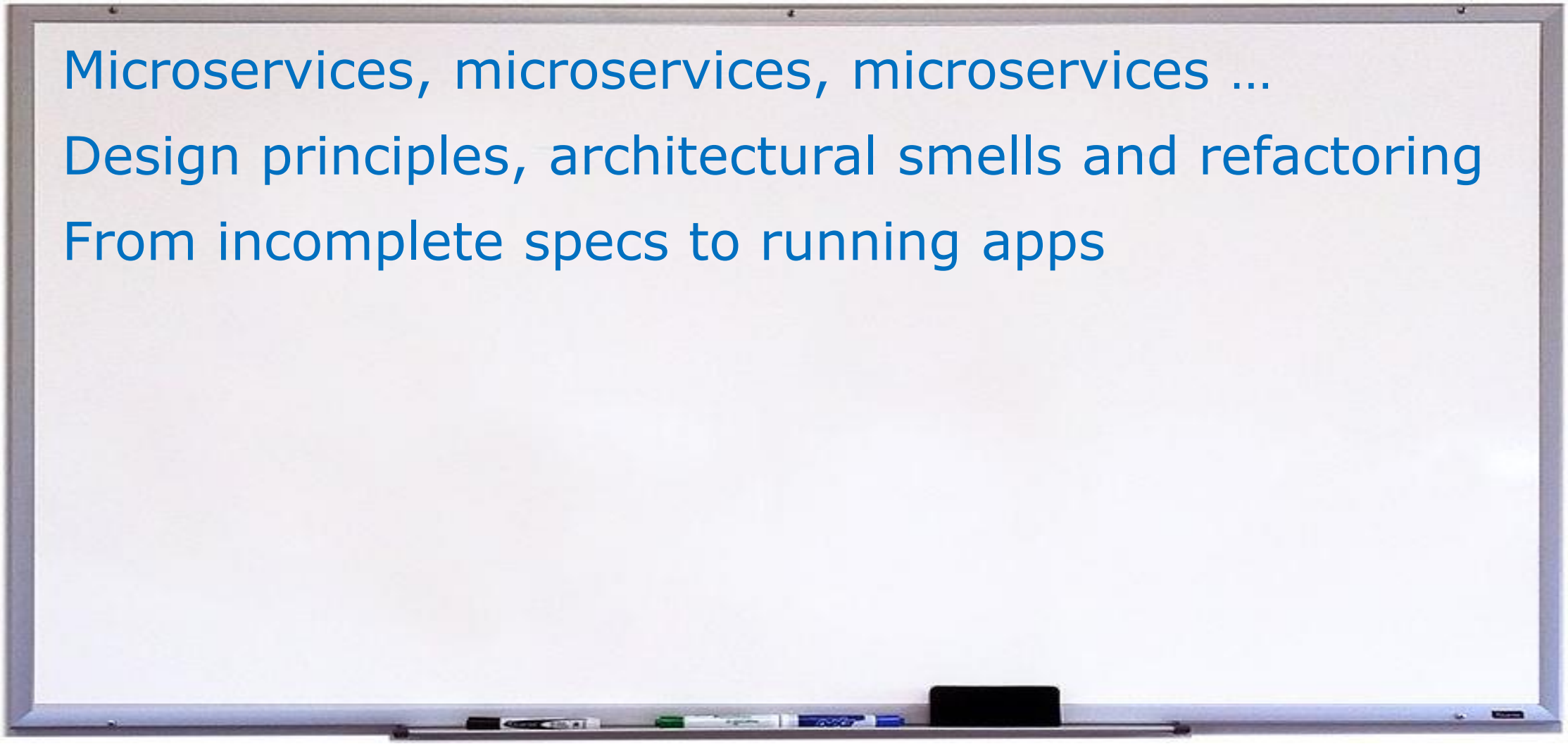


- ongoing work: dealing with container orchestration



Can I play with μFreshener?

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


Microservices, microservices, microservices …
Design principles, architectural smells and refactoring
From incomplete specs to running apps

Motivations

- Microservice-based applications integrate many interacting services

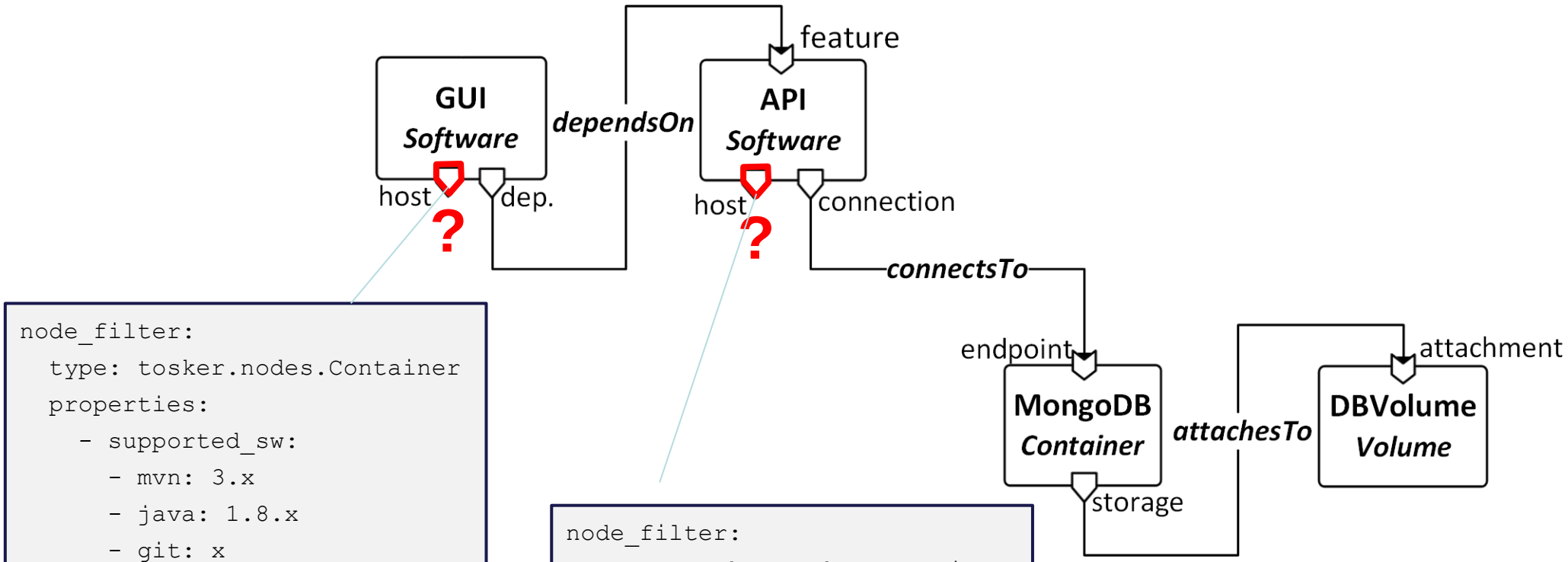
→ Need to select an appropriate runtime enviroment for each microservice

→ Need to package each microservice into the selected runtime environment

Idea (1/2)

Exploit the TOSCA-based representation of microservice-based applications to **specify only the application components and the software support they need**

Example



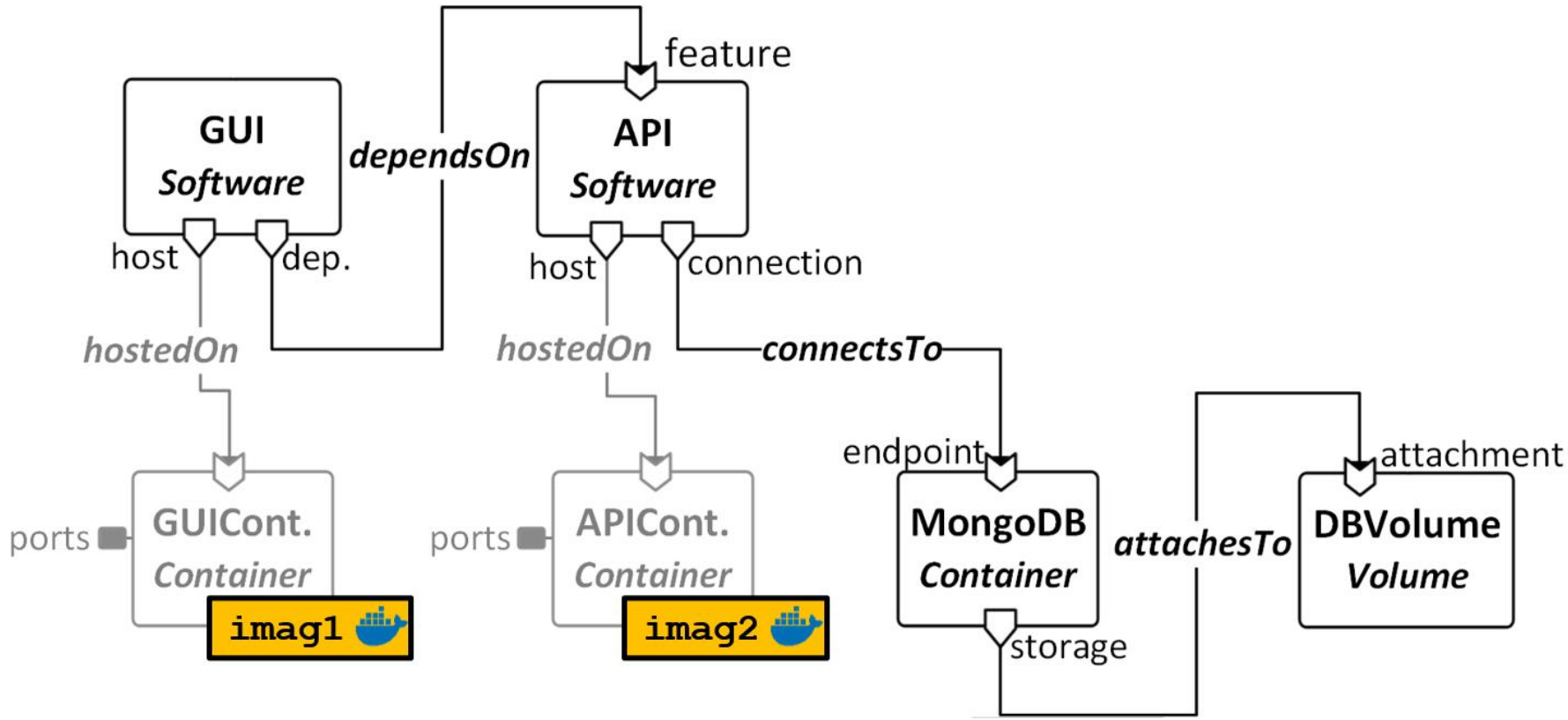

```
node_filter:
  type: tosker.nodes.Container
  properties:
    - supported_sw:
      - mvn: 3.x
      - java: 1.8.x
      - git: x
    - ports:
      - 8080: 8000
    - os_distribution: ubuntu
```

```
node_filter:
  type: tosker.nodes.Container
  properties:
    - supported_sw:
      - node: 6.x
      - npm: 3.x
      - git: x
    - ports:
      - 3000: 8080
```

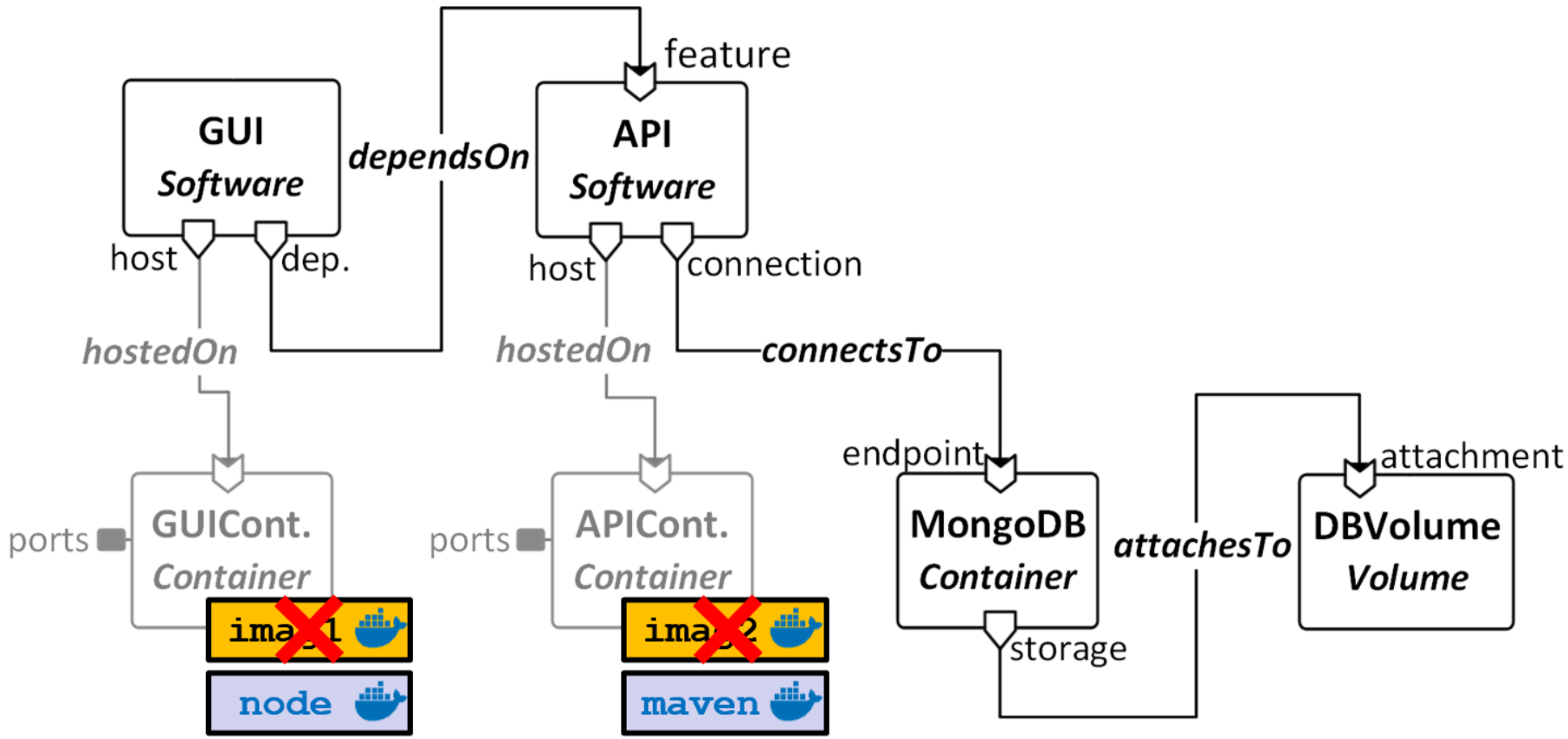
Idea (2/2)

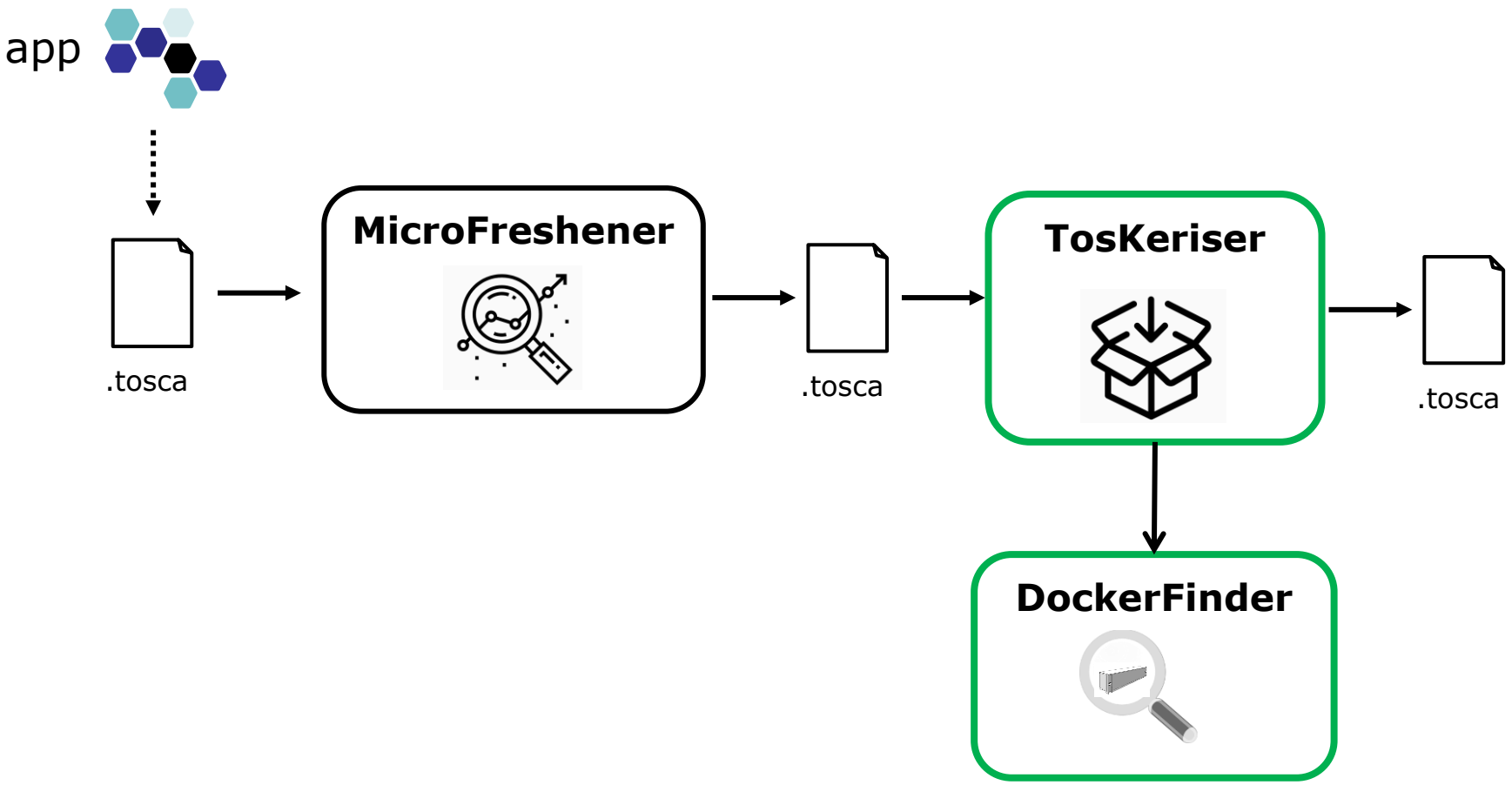
Develop a tool for **automatically completing (and updating) TOSCA application specifications by discovering and including Docker-based runtime environments** providing the software support needed by each microservice

```
$ toskerise thinking.csar --policy size
```



```
$ toskerise thinking.completed.csar -f --policy most_used
```





A. Brogi, D. Neri, L. Rinaldi, J. Soldani. ***Orchestrating incomplete TOSCA applications with Docker****. Science of Computer Programming. 2018.*

A. Brogi, D. Neri, J. Soldani. ***A microservice-based architecture for (customisable) analyses of Docker images****. Software: Practice and Experience. 2018.*

Motivations

- Microservice-based applications integrate many interacting services

→ Need to select an appropriate runtime enviroment for each microservice

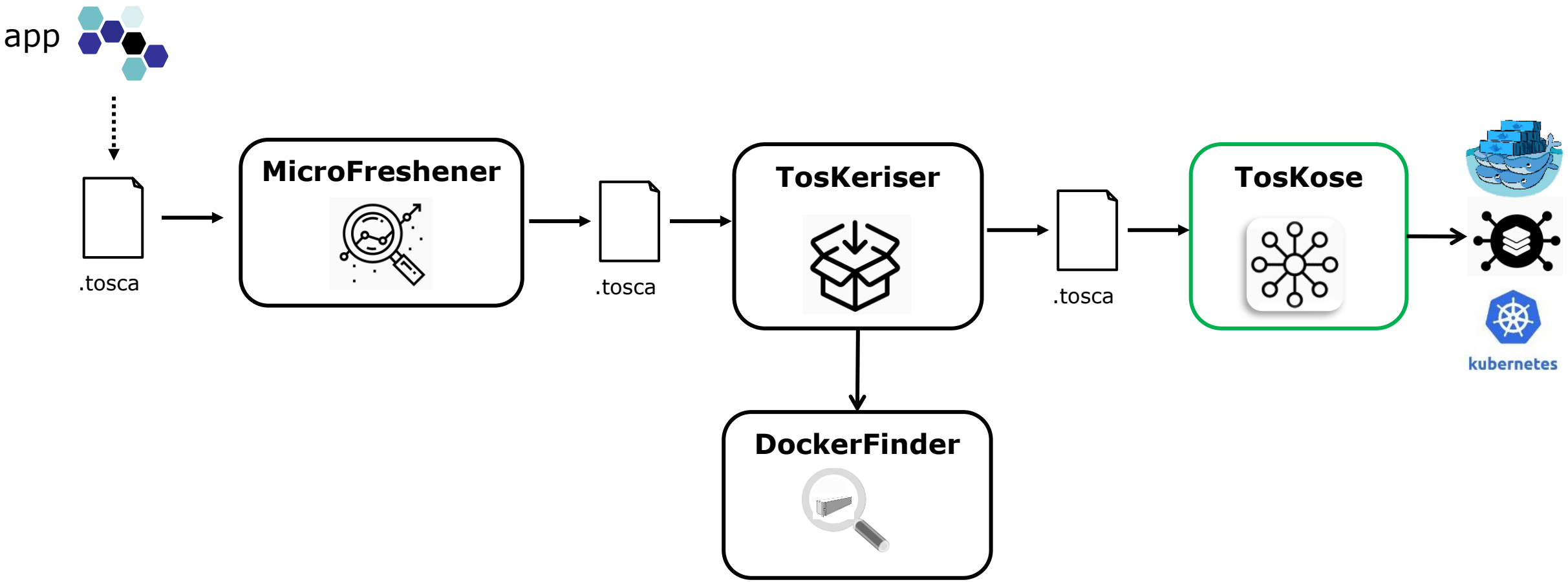
→ Need to package each microservice into the selected runtime environment

Idea

Develop a tool to **automate the deployment on top of existing container orchestrators**

Ingredients:

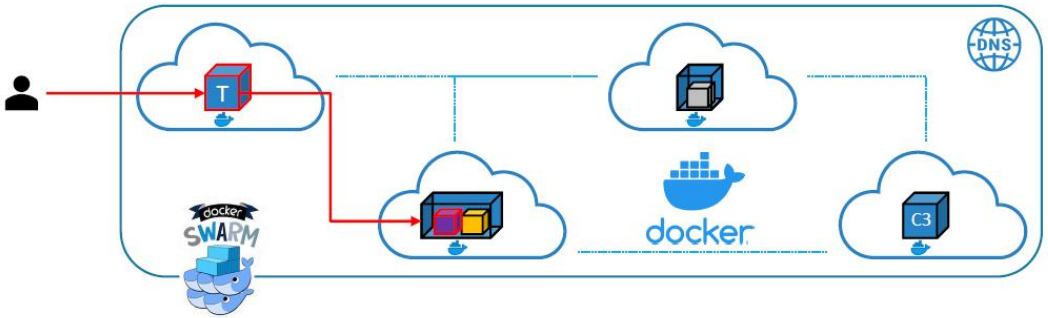
- a *process management* system inside containers
- a *service* for component-aware orchestration
- a *packager* capable of deploying on existing container orchestrators



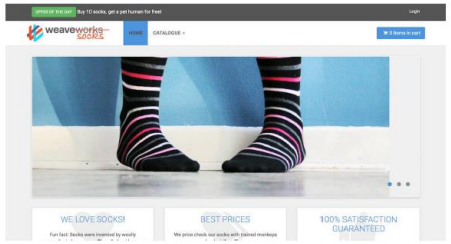
M. Bogo, J. Soldani, D. Neri, A. Brogi. ***Component-aware Orchestration of Cloud-based Enterprise Applications, from TOSCA to Docker and Kubernetes****. Software: Practice and Experience. 2020.*

Case studies

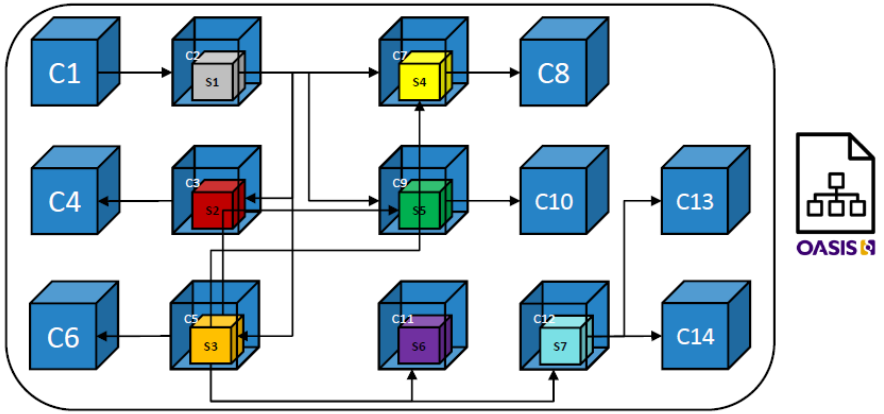
Thinking



Sock Shop



- 7 software components
- 14 containers (7 standalone)
- Deploy on Cluster of 4 VMs with Docker Swarm

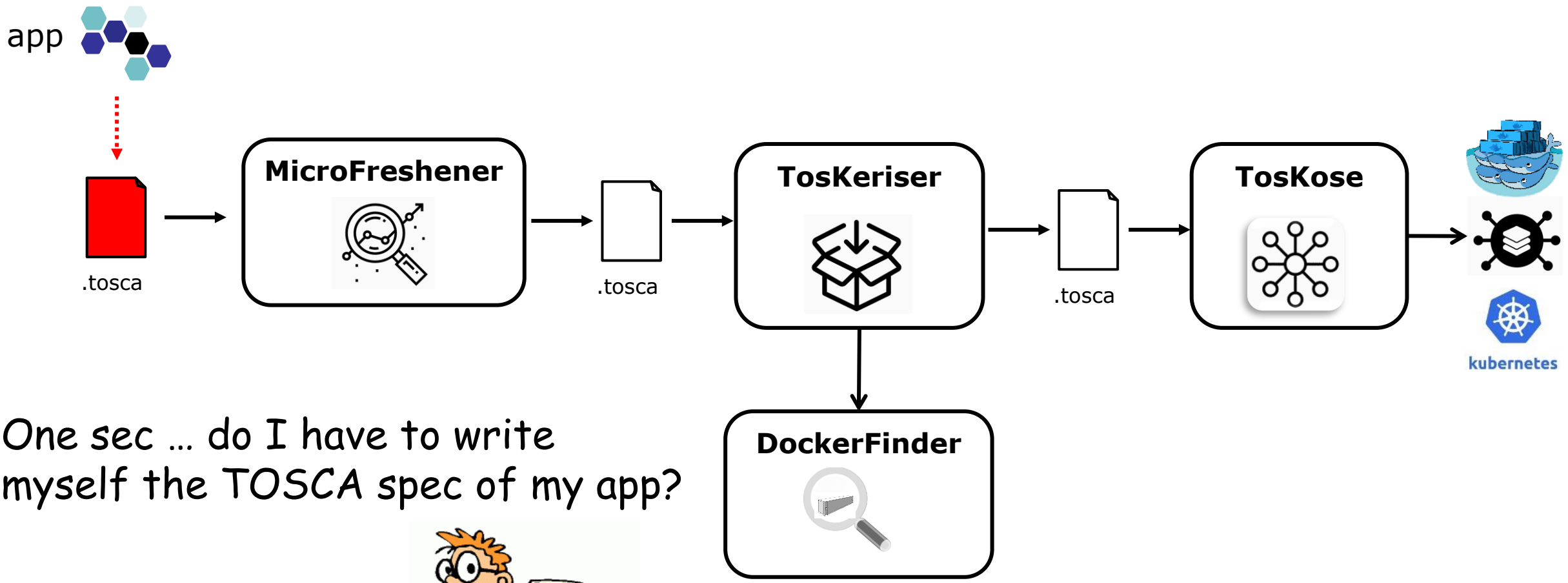


Can I play with these tools too?

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

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

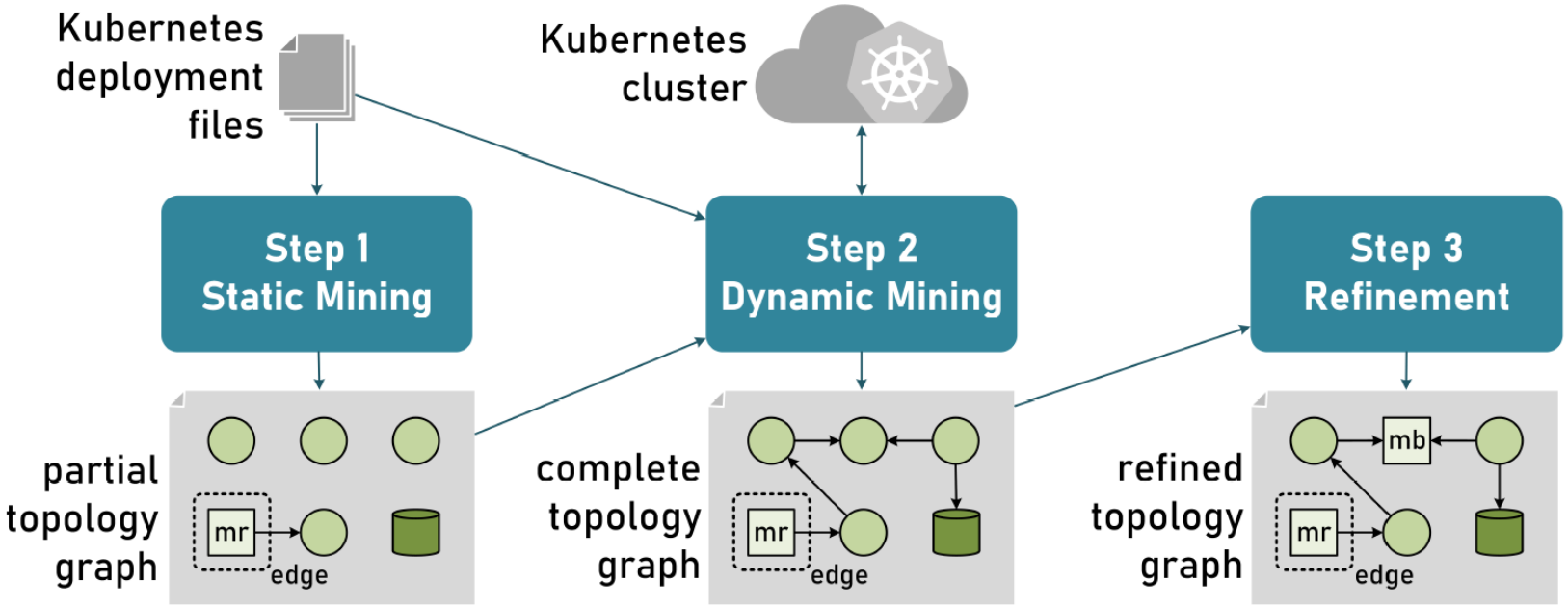
https://github.com/di-unipi-socc/toskose



One sec … do I have to write myself the TOSCA spec of my app?


Microservices, microservices, microservices …
Design principles, architectural smells and refactoring
From incomplete specs to running apps
Mining the architecture of microservice-based apps

Automatically deriving the architecture of black-box applications



eliciting services & databases

monitoring interactions among services & databases

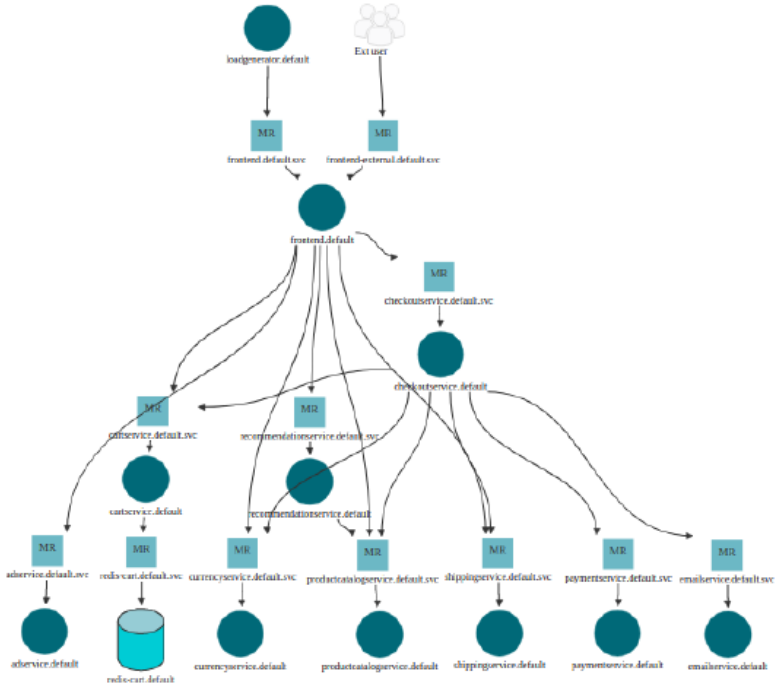
identifying integration components

+ marshalling obtained architecture to TOSCA

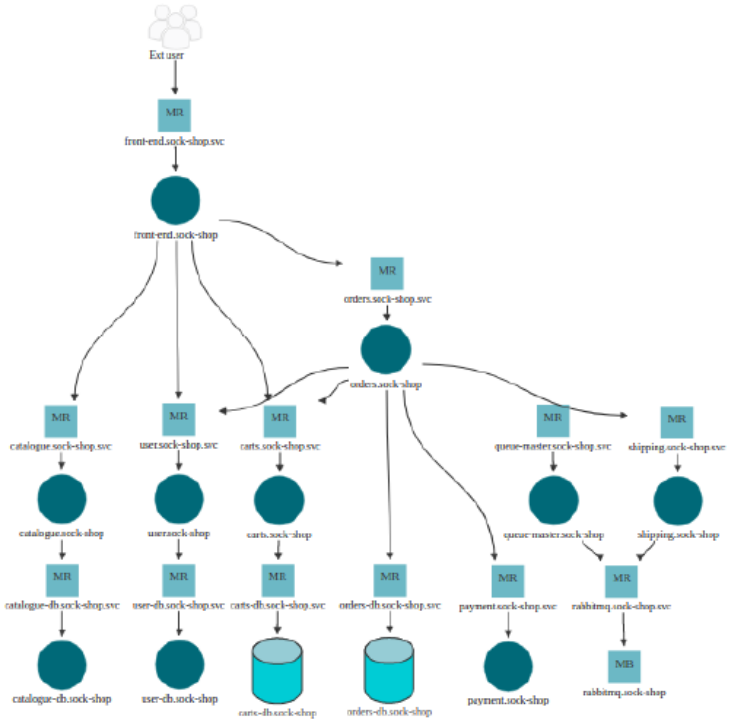
G. Muntoni, J. Soldani, A. Brogi. ***Mining the Architecture of Microservice-Based Applications from their Kubernetes Deployment.*** *WESOACS 2020.*

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

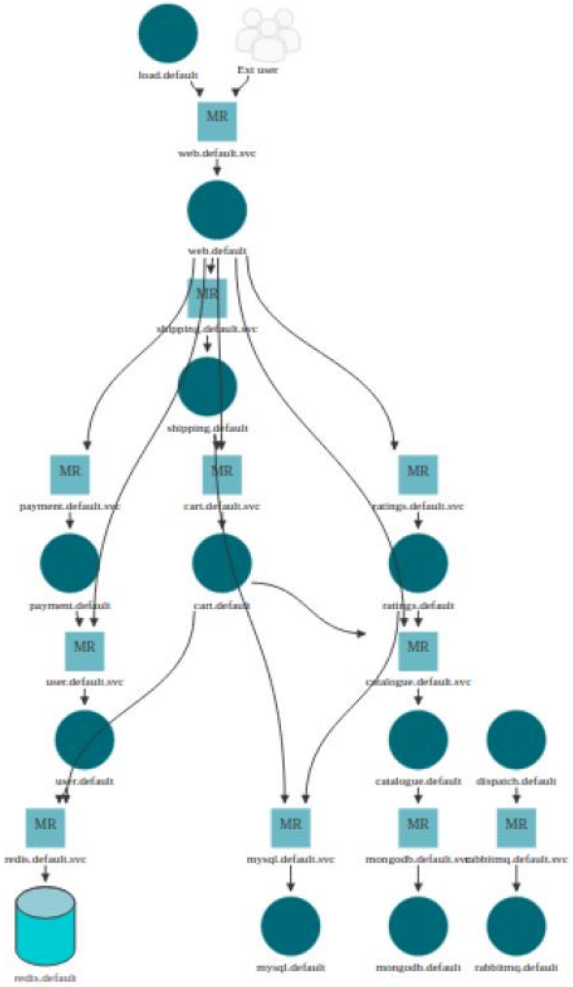
Case studies



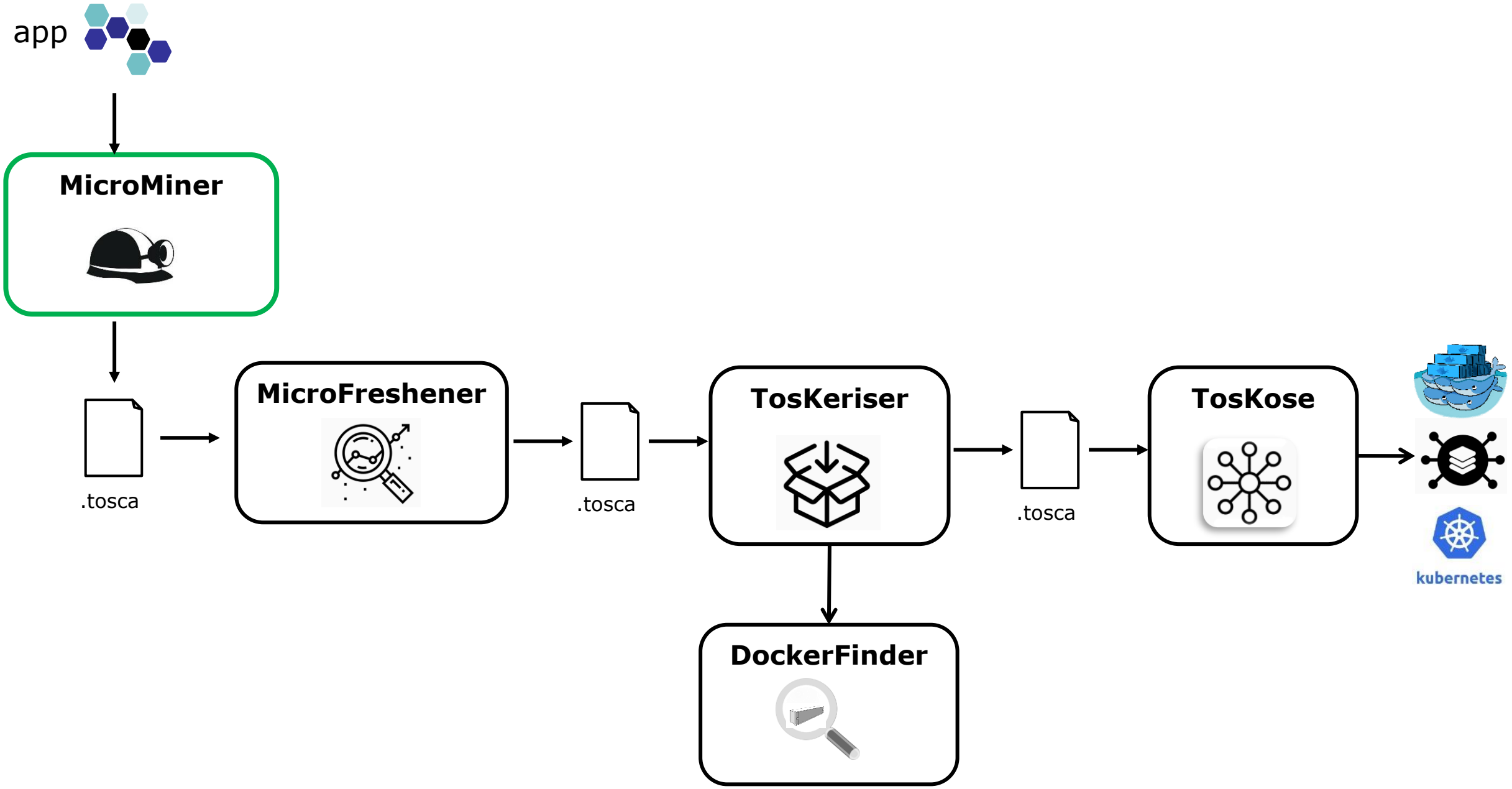
Online boutique



Sock shop



Robot shop


app
MicroMiner
.tosca
MicroFreshener
.tosca
TosKeriser
.tosca
TosKose
DockerFinder
kubernetes


Microservices, microservices, microservices …
Design principles, architectural smells and refactoring
From incomplete specs to running apps
Mining the architecture of microservice-based apps
Concluding remarks

Take-home message: A (minimal) modelling of microservice-based applications can considerably simplify their design and analysis and allow automating their container-based completion and deployment

Many interesting research directions on microservices (non-exhaustive, biased list):

- DSLs for microservices
- Security
- Monitoring
- Identifying failure causalities
- Continuous reasoning
- Green computing
- …

… and thanks to

J. Soldani

D. Neri

O. Zimmermann

M. Bogo

G. Muntoni

L. Rinaldi

Microservices beyond COVID-19

Antonio Brogi

Department of Computer Science
University of Pisa, Italy