

MICROSERVICES I PRAKTIKEN

*från tröga monoliter till en arkitektur för kortare ledtider,
högre skalbarhet och ökad feltolerans*

MAGNUS LARSSON

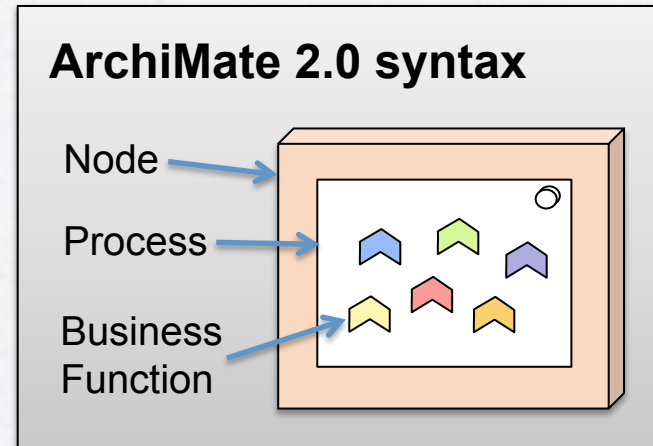
2015.05.21 | CALLISTAENTERPRISE.SE

AGENDA

- What's the problem?
- New solutions to old problems...
- What's a microservice?
- New challenges with microservices
- Implementing microservices
- Demonstration

WHAT'S THE PROBLEM??

- Well known problems with monolithic applications
 - Poor scalability and resilience
 - Long release cycles
- ...we had tried to solve these problems before (and failed?)...
- But there are new opportunities now!

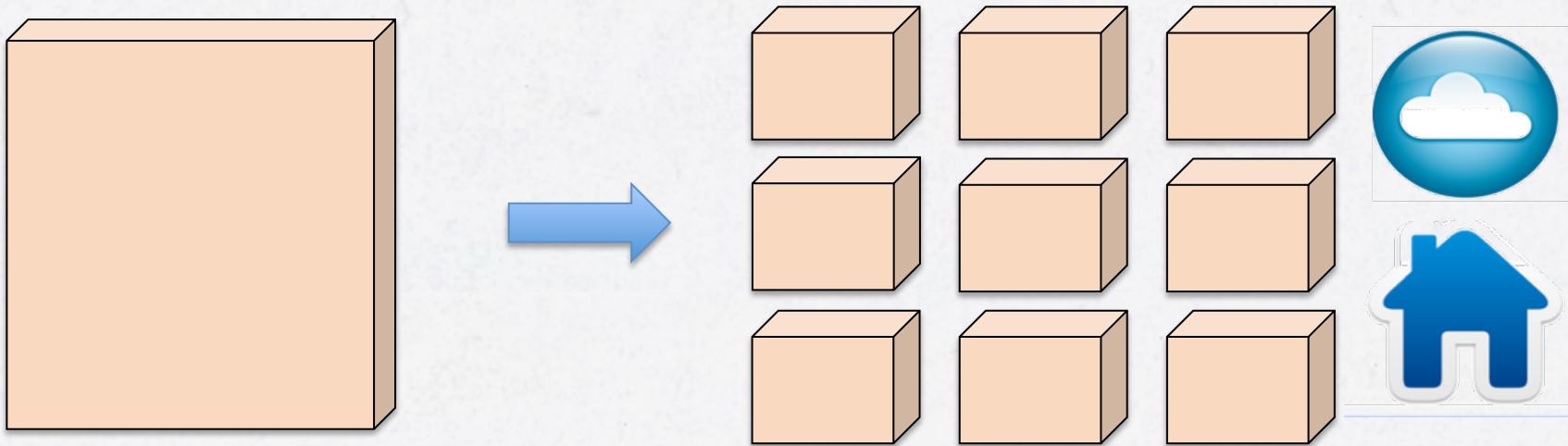


HISTORY OF MICROSERVICES

- Microservices was first heard of in May 2011
- Success stories from early adopters migrating from monoliths to microservices
 - Amazon (<http://goo.gl/LfsD67>)
 - eBay (<http://goo.gl/dodV2c>)
 - Gilt (<http://goo.gl/yVVox9>)
 - Groupon (<https://goo.gl/uKTtAs>)
 - Karma (<https://goo.gl/kXObAO>)
 - Netflix: Part 1, part 2 and “Fast Delivery” (<https://goo.gl/MVgHM1>, <https://goo.gl/fDeZ5A>, <https://goo.gl/hN6ZCL>)
 - SoundCloud: Part 1, part 2 and part 3 (<https://goo.gl/Xq0Cgm>, <https://goo.gl/swJ8Vt>, <https://goo.gl/J2oN8I>)

NEW SOLUTIONS TO OLD PROBLEMS

- Strong trend moving from “Big Iron” to many small servers
 - Typically virtual servers
 - In cloud or/and on premises
 - Better price/performance

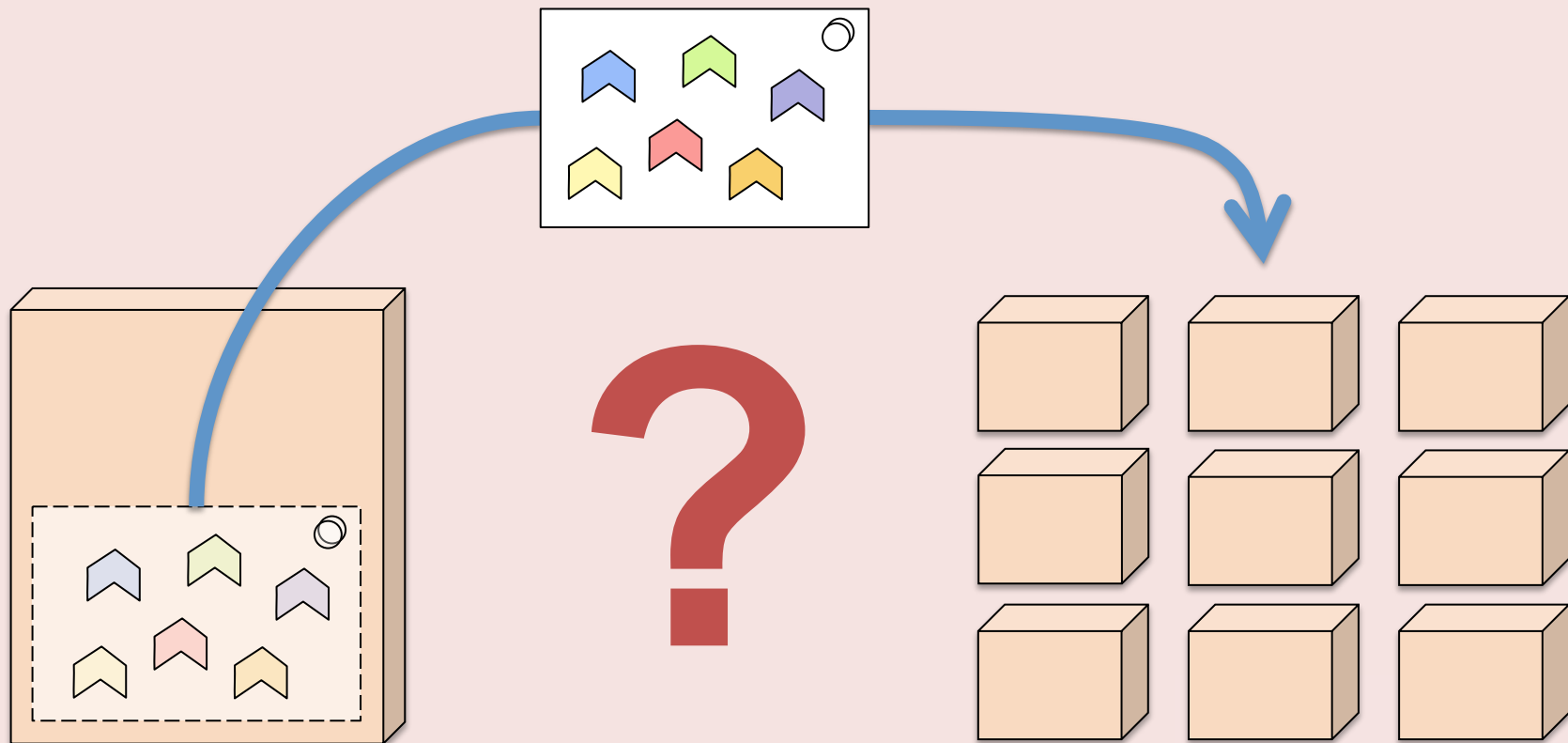


NEW SOLUTIONS TO OLD PROBLEMS

- *Cloud computing* makes it easier to manage many small servers
 - **IaaS: Infrastructure as a Service**
 - » Deliver virtual servers
 - » E.g. Amazon EC2, Microsoft Azure, Google Compute Engine et. al.
 - **PaaS: Platform as a Service**
 - » Deliver an application platform
 - » E.g. Heroku, Red Hat OpenShift, Pivotal Cloud Foundry et. Al.
 - » **Note:** Some PaaS can be used on premises, e.g. OpenShift and Cloud Foundry
 - **Docker, the Container revolution...**
 - » *IaaS + PaaS* → *CaaS* ?
 - » Windows Server Containers on its way (<http://goo.gl/ZmEkTS>)!

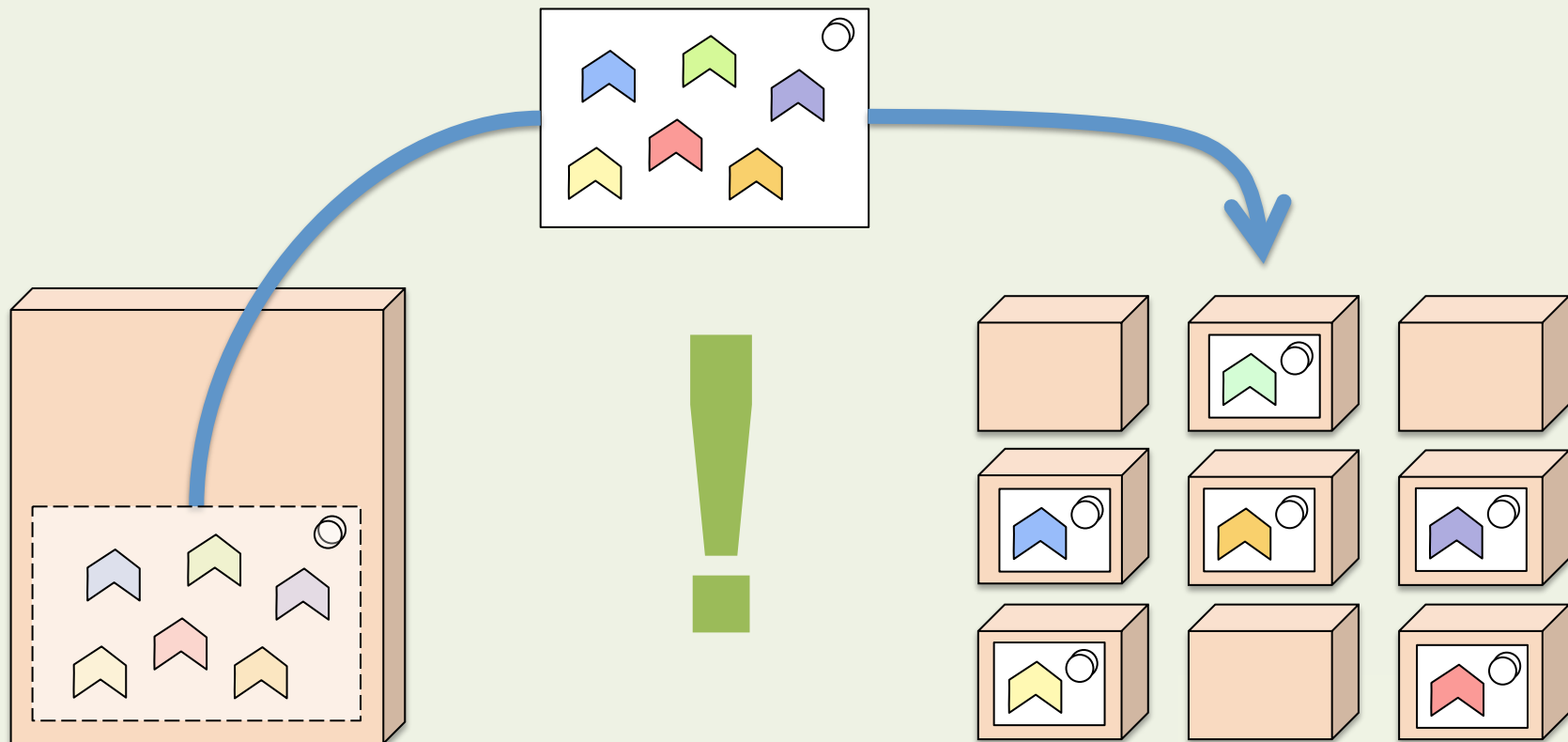
NEW SOLUTIONS TO OLD PROBLEMS

- How to fit monolithic applications in a number of small boxes?



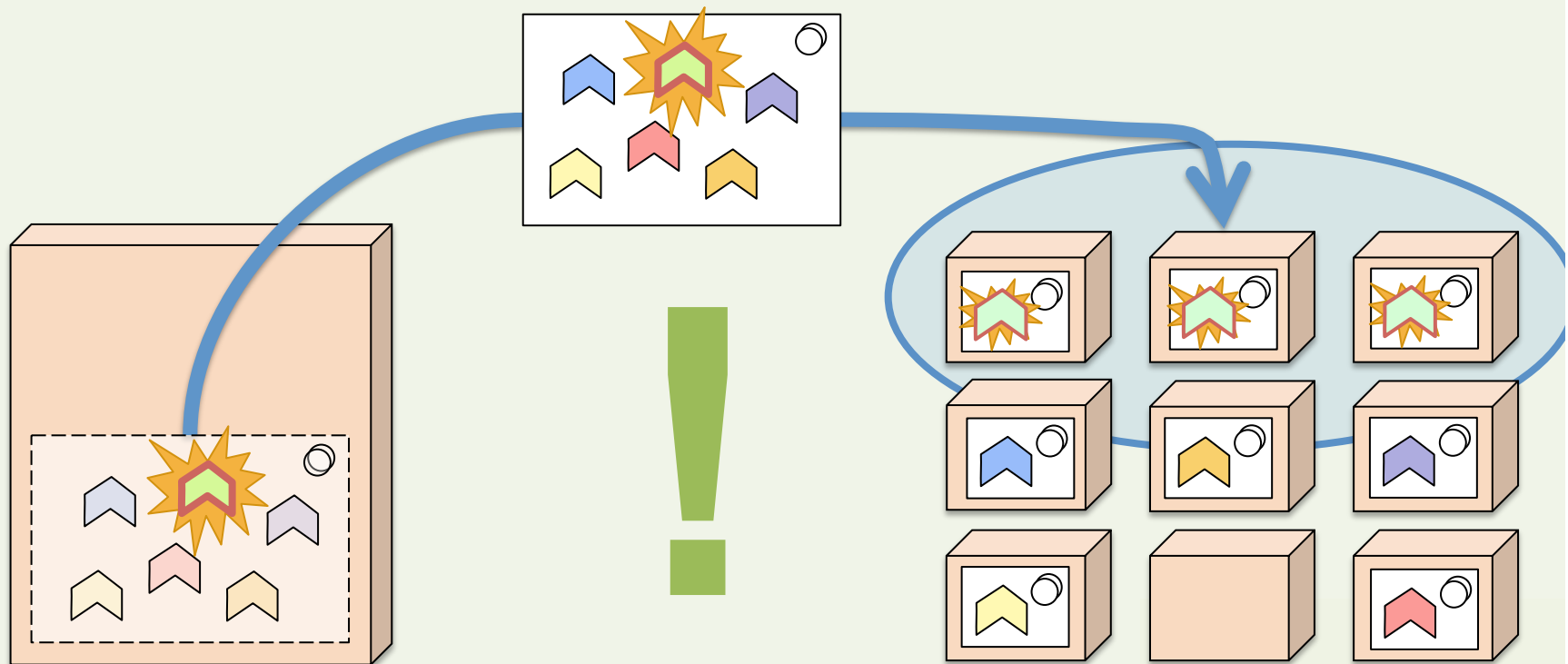
NEW SOLUTIONS TO OLD PROBLEMS

- We need to split the monolith to make it fit...



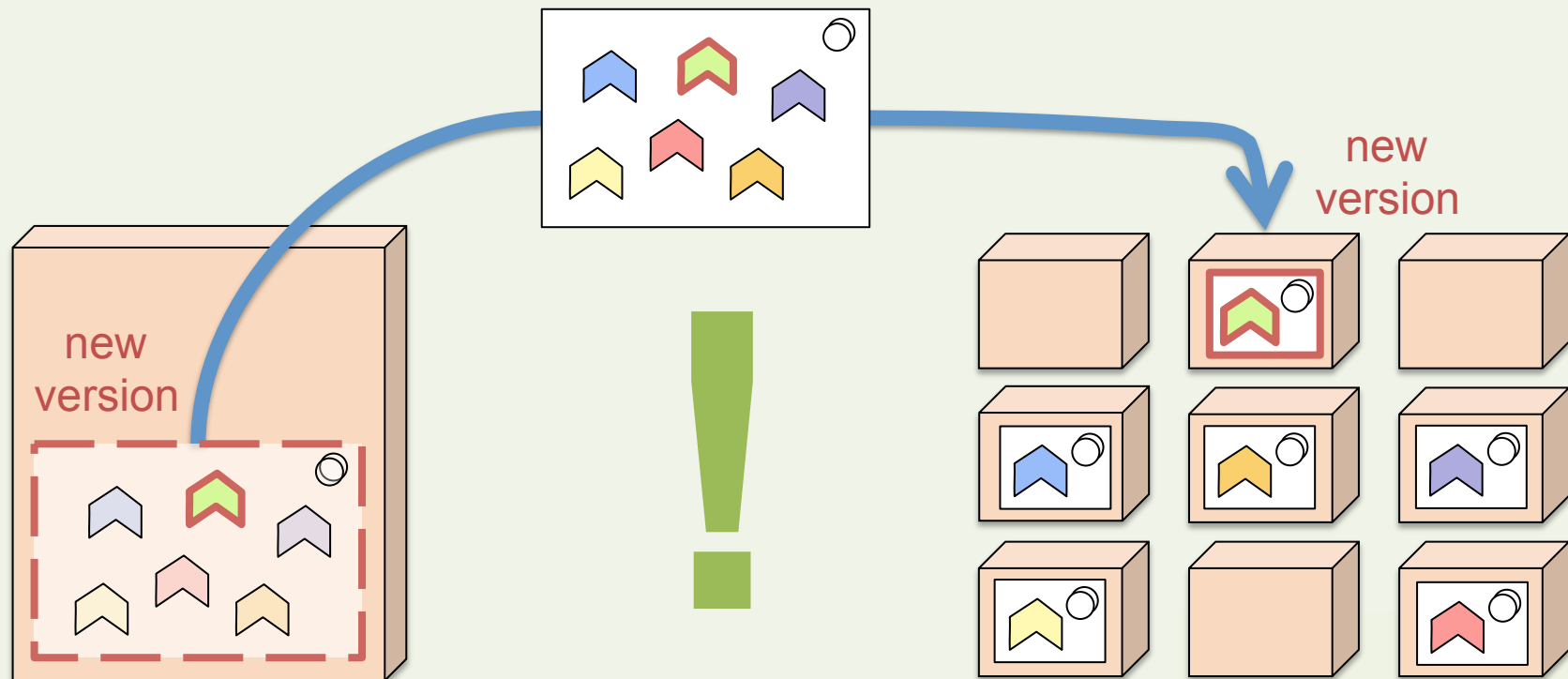
NEW SOLUTIONS TO OLD PROBLEMS

- Splitting the monolith also makes it easier to scale...
 - Auto scaling provided by platforms



NEW SOLUTIONS TO OLD PROBLEMS

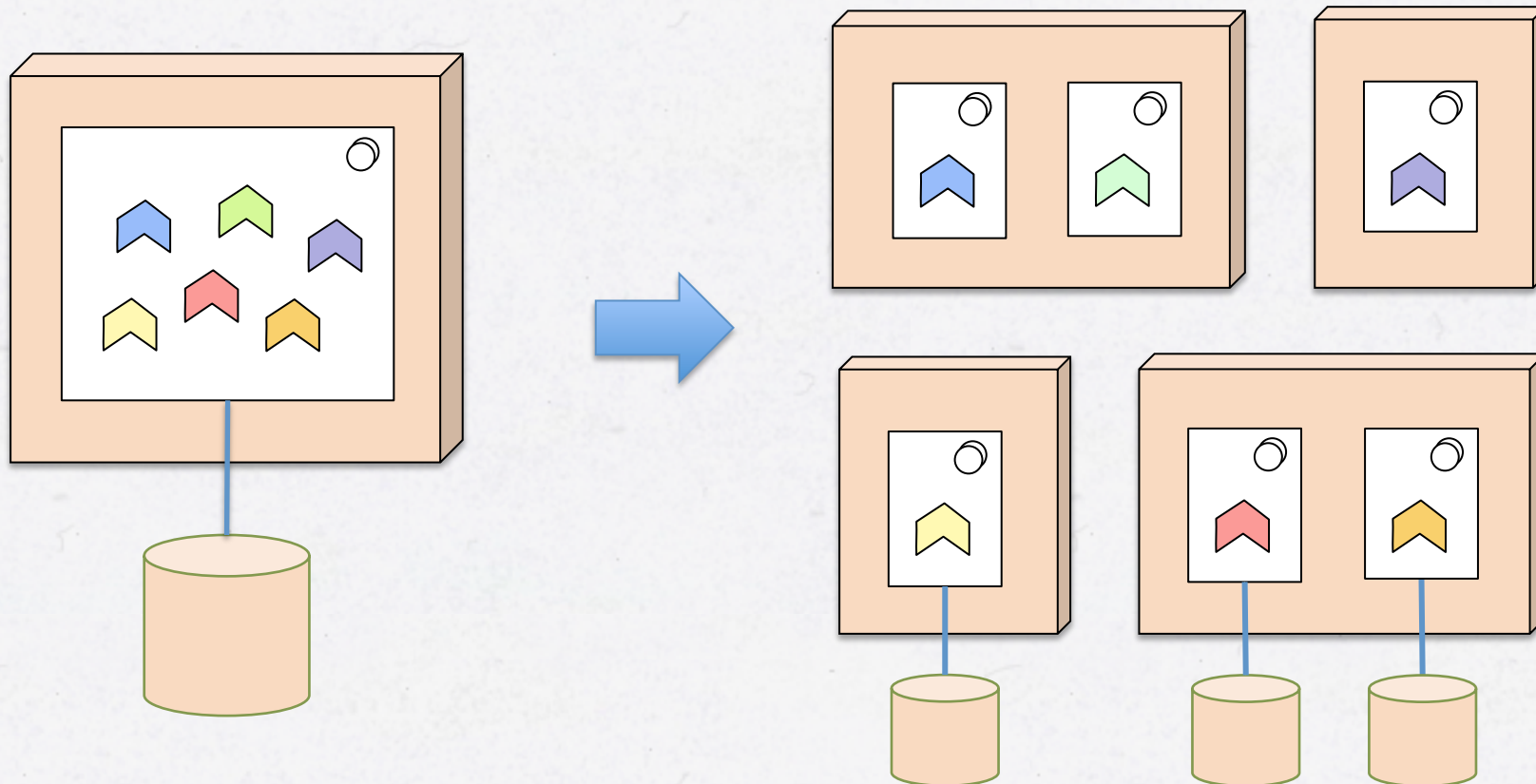
- Shorter release cycles
 - So much easier to update or replace a microservice compared to a monolith



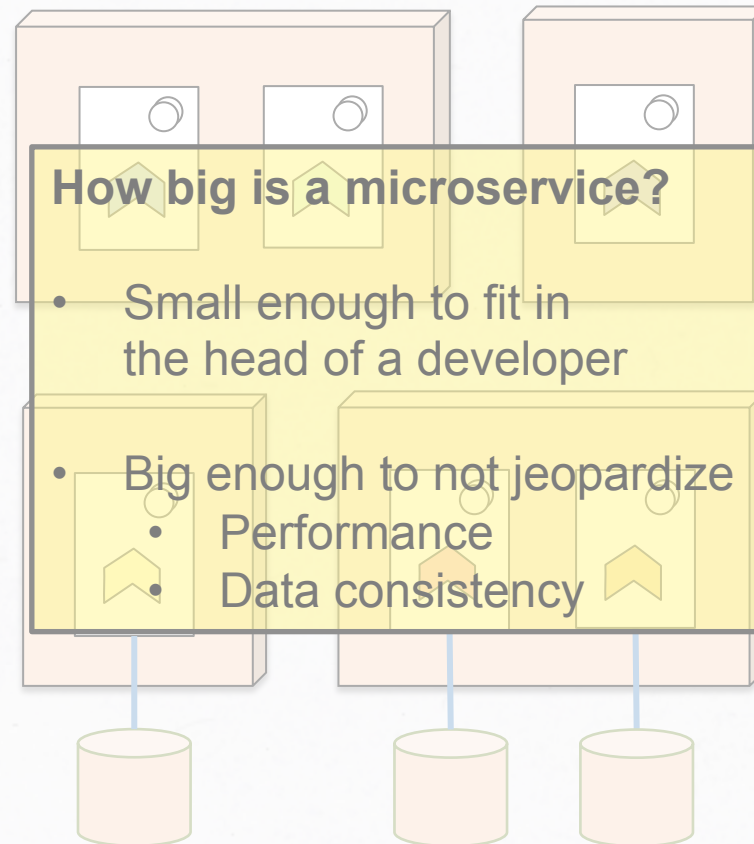
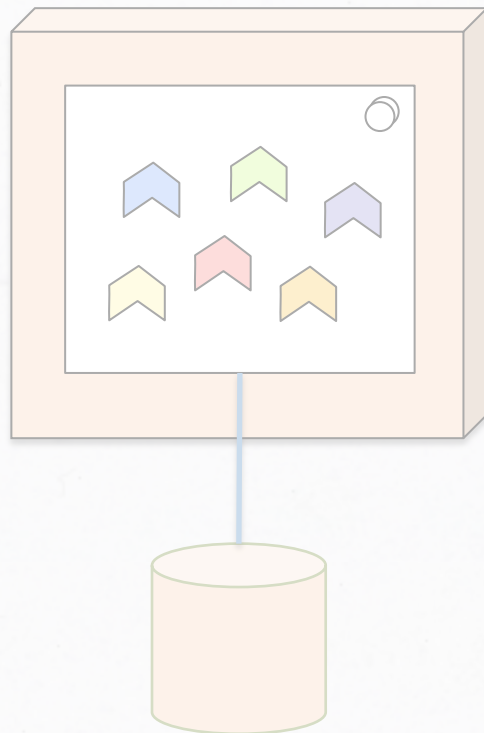
WHAT'S A MICROSERVICE?

- A software component that is independently replaceable and upgradeable
- Share nothing architecture
 - They don't share databases!
 - Only communicate through well defined interfaces,
 - » E.g. REST services or queuing mechanisms
- Typically deployed as separate runtime processes

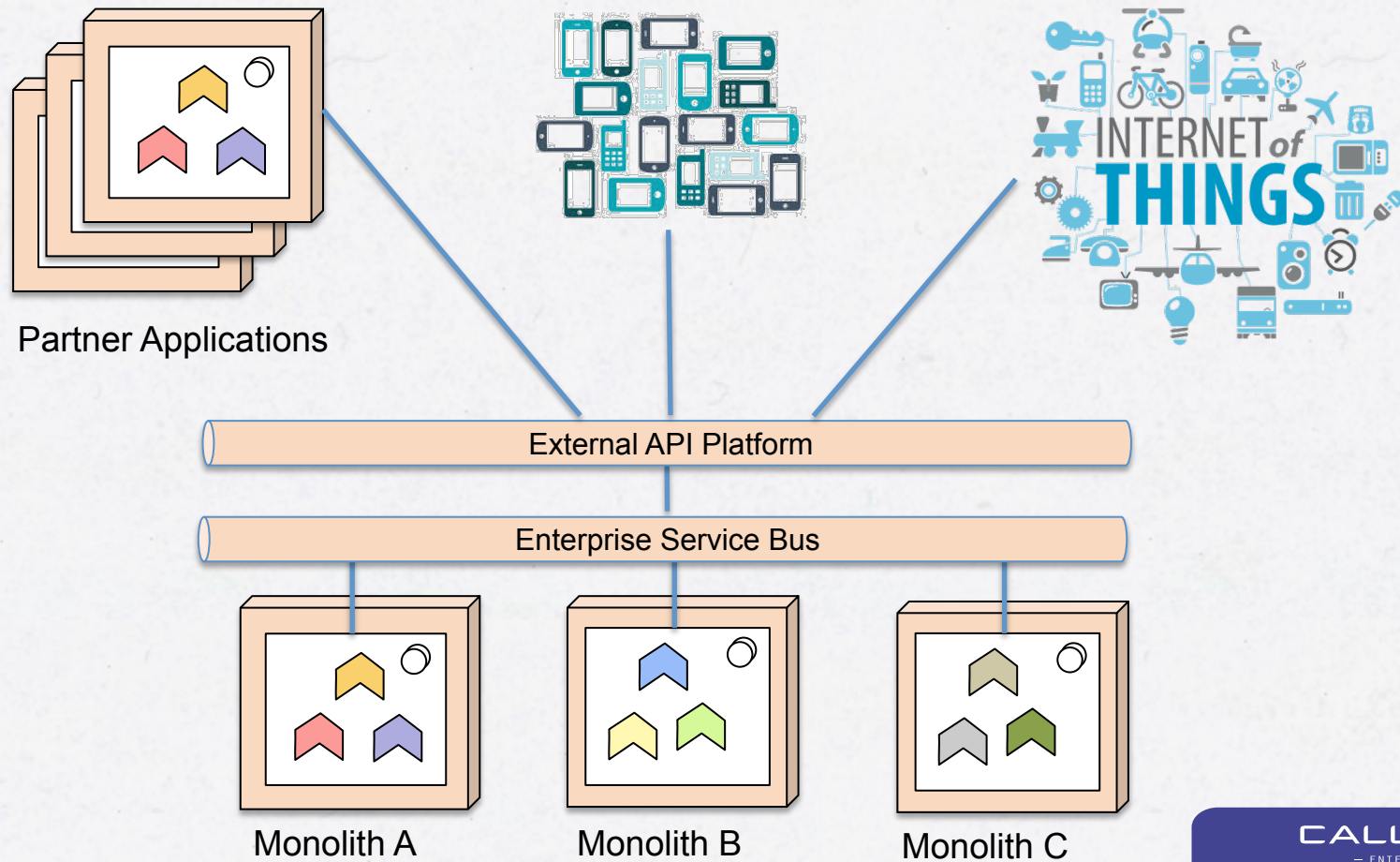
WHAT'S A MICROSERVICE?



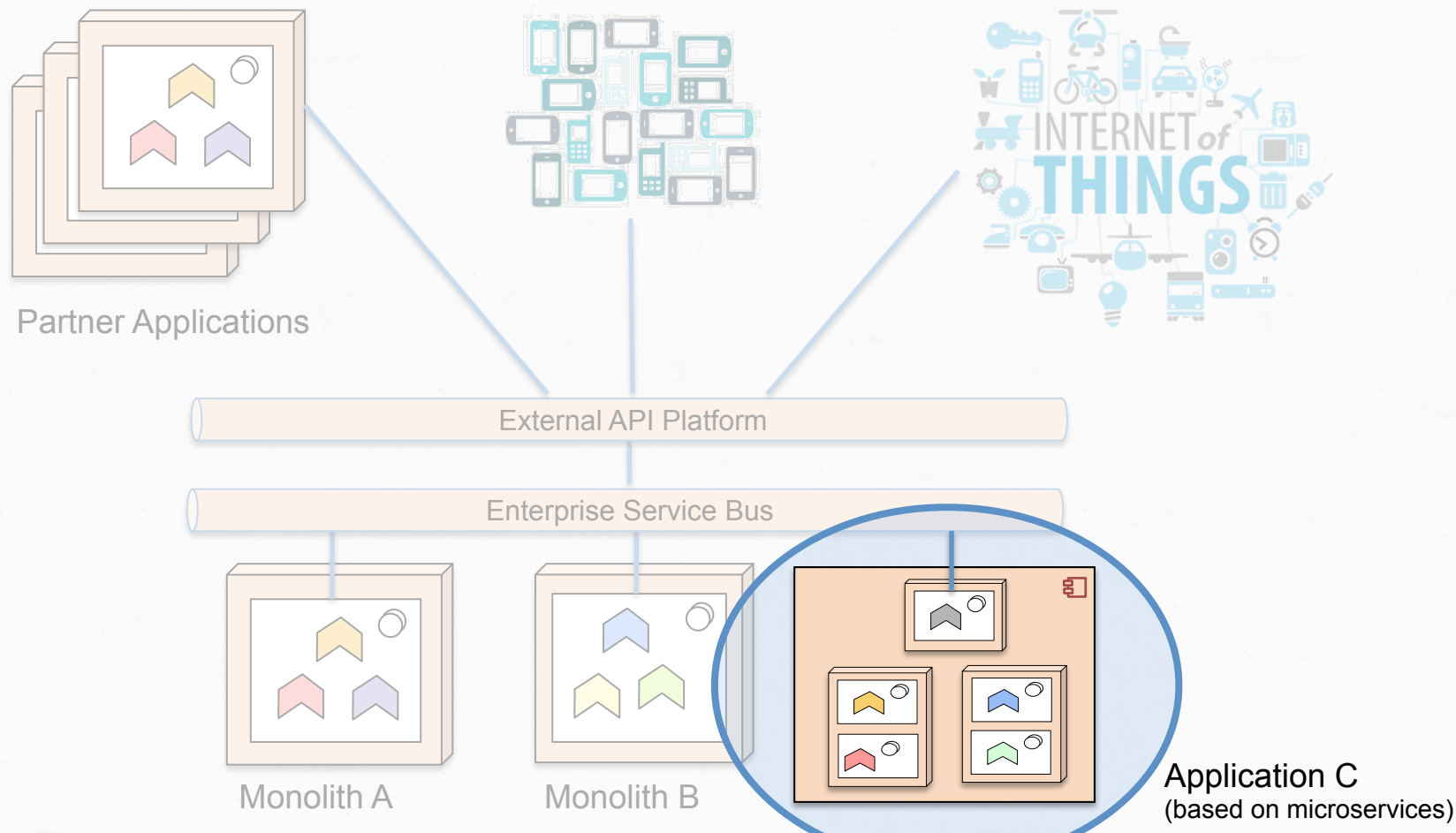
WHAT'S A MICROSERVICE?



HOW DOES MICROSERVICES FIT INTO AN EXISTING SYSTEM LANDSCAPE?

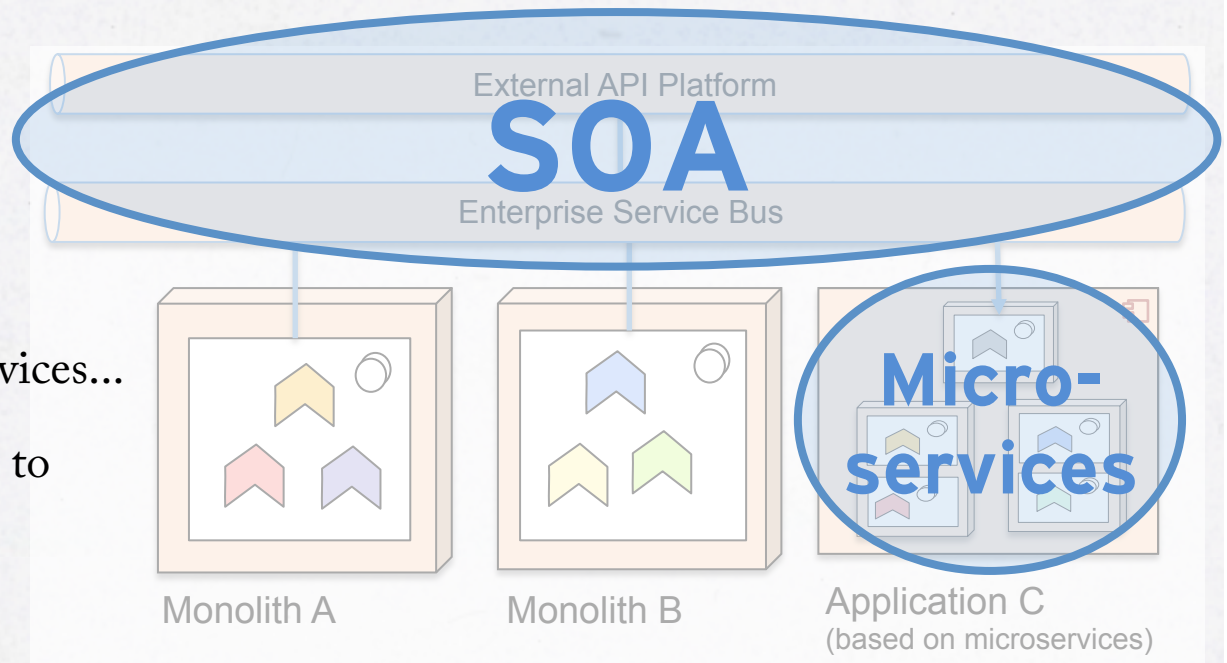


HOW DOES MICROSERVICES FIT INTO AN EXISTING SYSTEM LANDSCAPE?



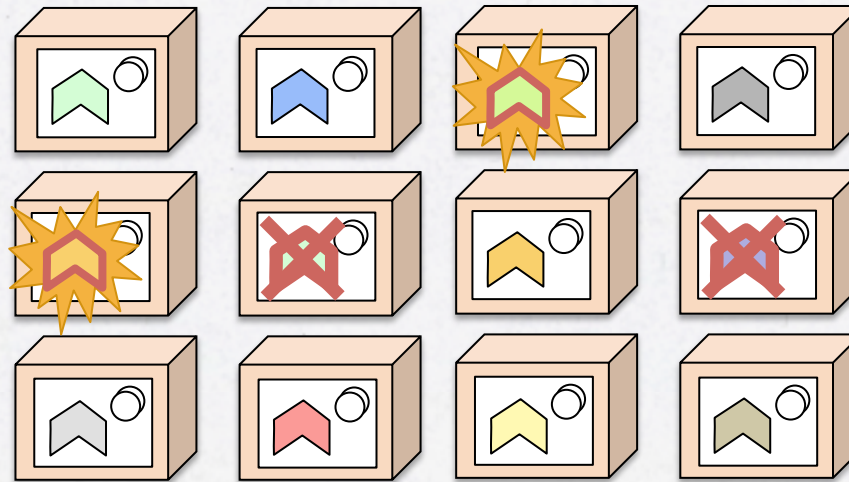
WHAT'S A MICROSERVICE?

- SOA vs. Microservices
 - SOA and microservices don't conflict, they complement each other!
 - SOA is about how to reuse existing functionality as services...
 - Microservices is about how to make functionality to scale better with high resilience and short release cycles...



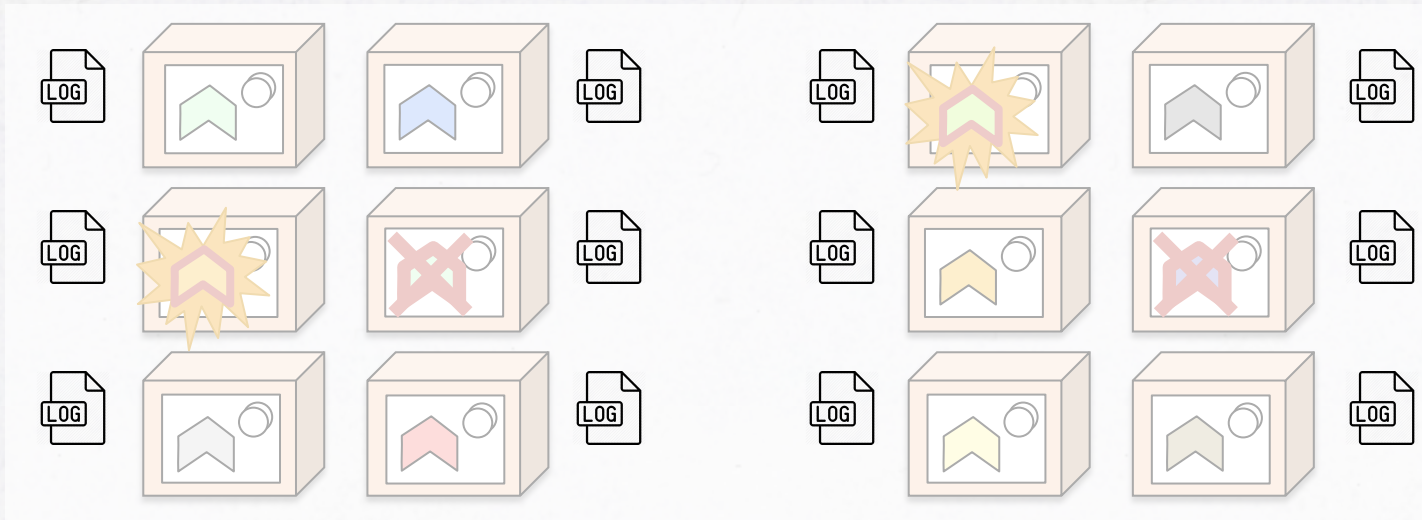
NEW CHALLENGES WITH MICROSERVICES

- Managing large numbers of microservices...
 - Where are they and are they ok???



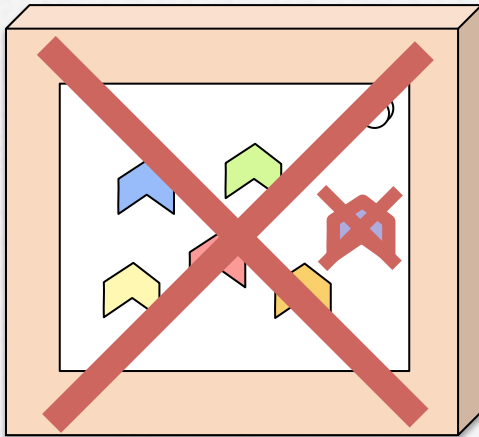
NEW CHALLENGES WITH MICROSERVICES

- What went wrong???

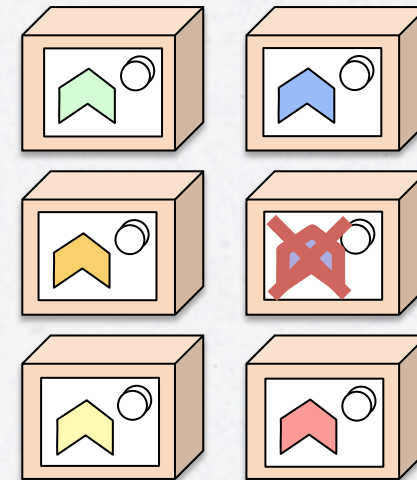


NEW CHALLENGES WITH MICROSERVICES - RESILIENCE

- Minor effect if a small microservice fails than a big monolith...



VS



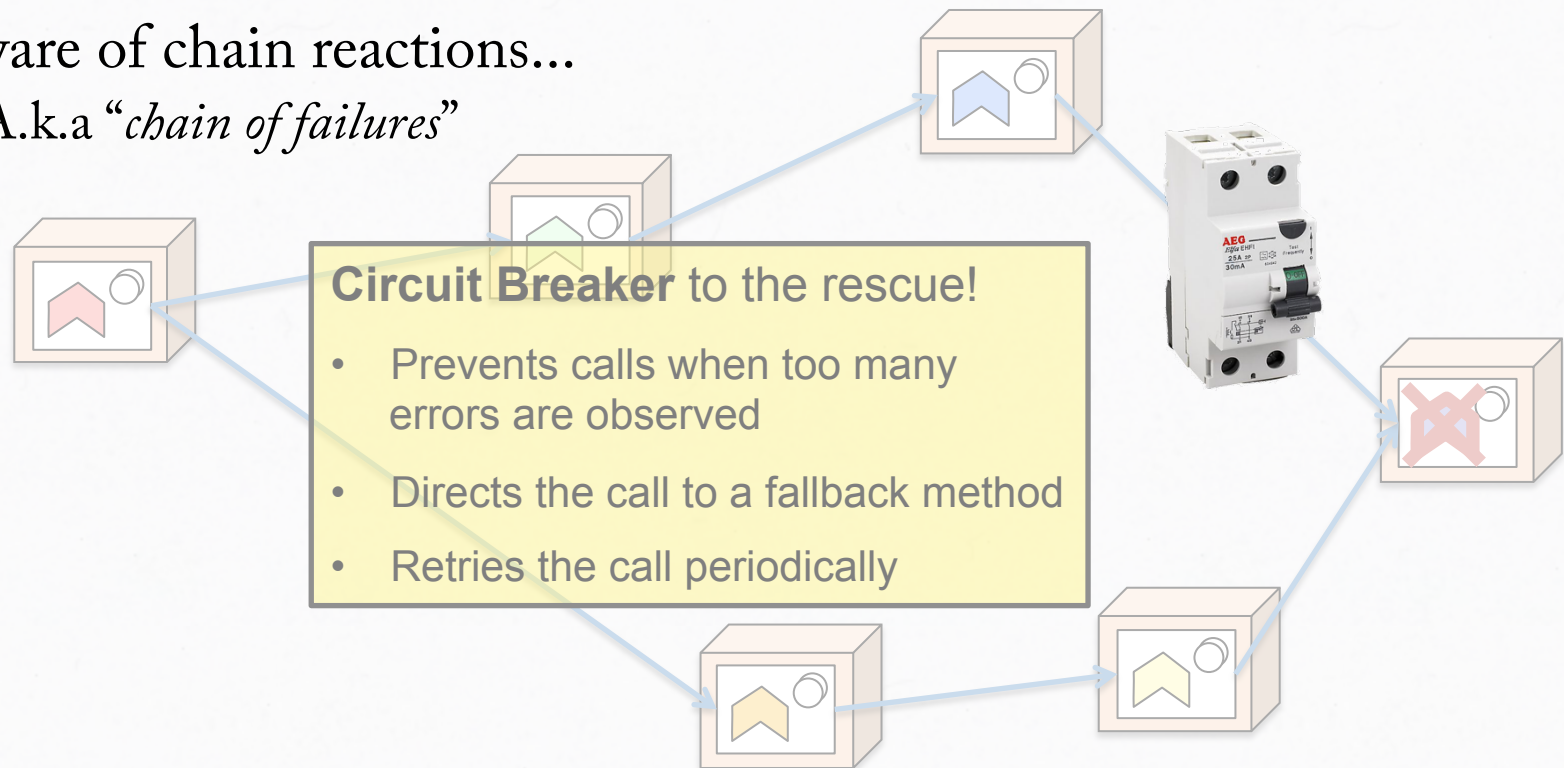
NEW CHALLENGES WITH MICROSERVICES - RESILIENCE

- Beware of chain reactions...
 - A.k.a “*chain of failures*”



NEW CHALLENGES WITH MICROSERVICES - RESILIENCE

- Beware of chain reactions...
 - A.k.a “*chain of failures*”



NEW CHALLENGES WITH MICROSERVICES

- Managing large numbers of microservices requires tools for
 1. **Runtime discovery of services**
 - » New services can auto-register at startup
 2. **Dynamic router and load balancer**
 - » Clients can detect new instances as they are started up
 3. **Centralized log management**
 - » Collects and visualize log events from distributed processes
 4. **Circuit breaker**
 - » Prevent problems with chain of failures
 5. **Protecting external API's**
 - » Secure external API's, e.g. using OAuth 2.0

NEW CHALLENGES WITH MICROSERVICES

- Managing large numbers of microservices requires tools for
 - Runtime discovery of services
 - New services can auto-register at startup
 - Dynamic router and load balancer
 - Clients can detect new instances as they are started up

NETFLIX
OSS



- Protecting external API's
 - Secure external API's using OAuth 2.0

AGENDA - WHERE ARE WE?

- What's the problem?
- New solutions to old problems...
- What's a microservice?
- New challenges with microservices
- **Implementing microservices**
- Demonstration

IMPLEMENT MICROSERVICES WITH OPEN SOURCE



- Netflix OSS (<http://goo.gl/DHOf4o>)
 - Since 2011, Netflix has been releasing components of their cloud platform as free and open source software
 - Obviously proven in battle...



SPRING CLOUD

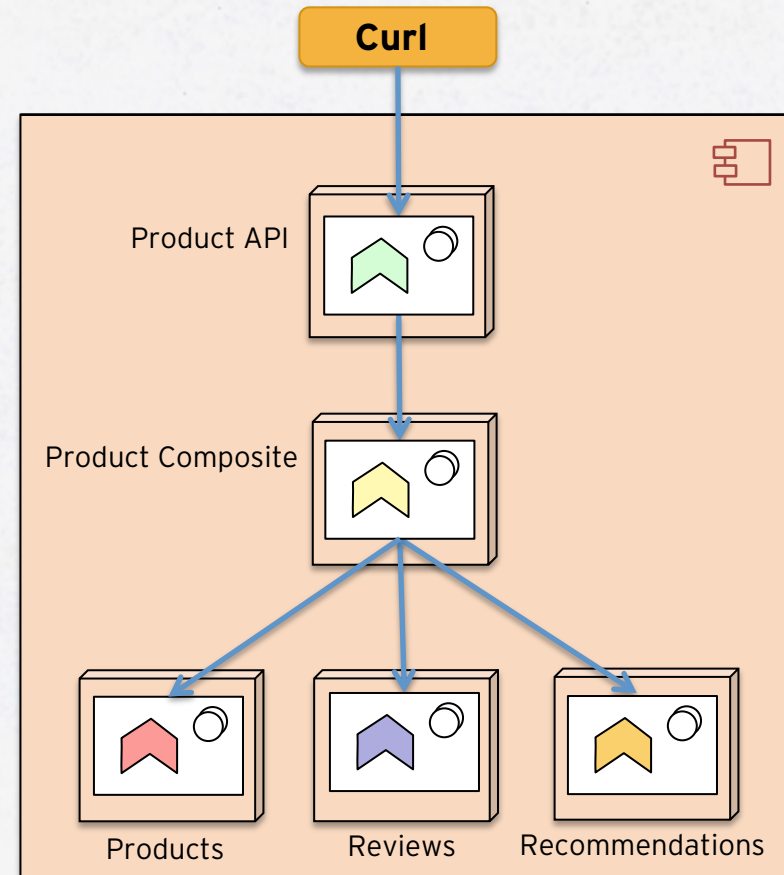
- Spring Cloud (<http://goo.gl/vHVdEp>)
 - Spring Cloud simplifies use of Netflix OSS
 - Add own components, e.g. OAuth 2.0 support
 - Based on Spring Boot and the “*convention over configuration*” paradigm



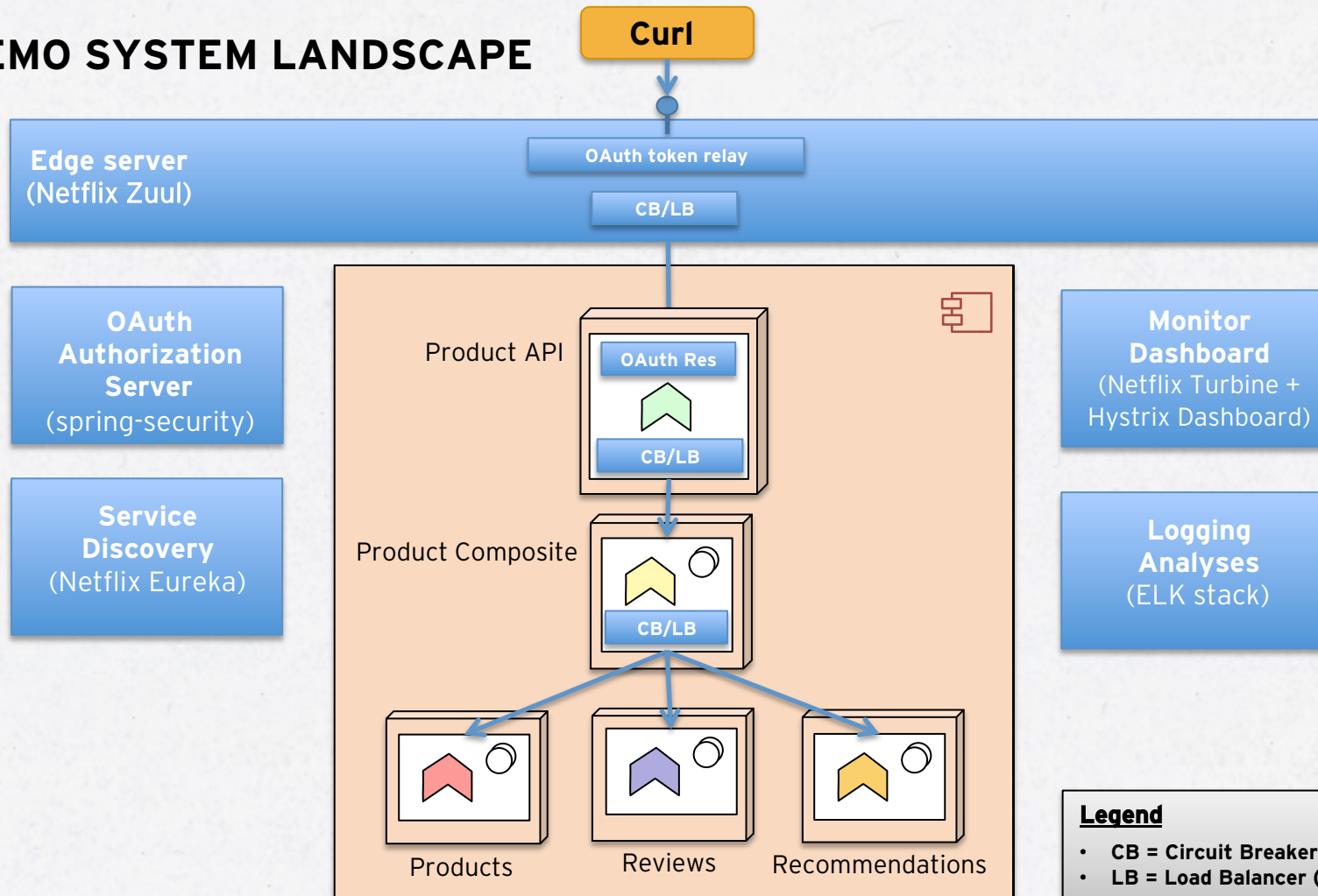
- The ELK stack (<https://goo.gl/aCHlhN>)
 - Elasticsearch, Logstash and Kibana
 - Used for centralized log analyses

DEMO SYSTEM LANDSCAPE

- An API for product-information
- A composite service aggregate information from three core-services
- Plus infrastructure services for OAuth, Discovery and Edge-servers...



DEMO SYSTEM LANDSCAPE



DEPLOY

- In cloud

- Using PaaS: Pivotal Web Services

```
$ cf push
```

 (<https://goo.gl/I3oDGt>)

- Sample configuration file

```
---  
memory: 512M  
instances: 1  
applications:  
- name: product-api-service  
  path: product-api.jar
```

- On premises

- Using Docker

```
$ docker-compose start
```

- Sample configuration file

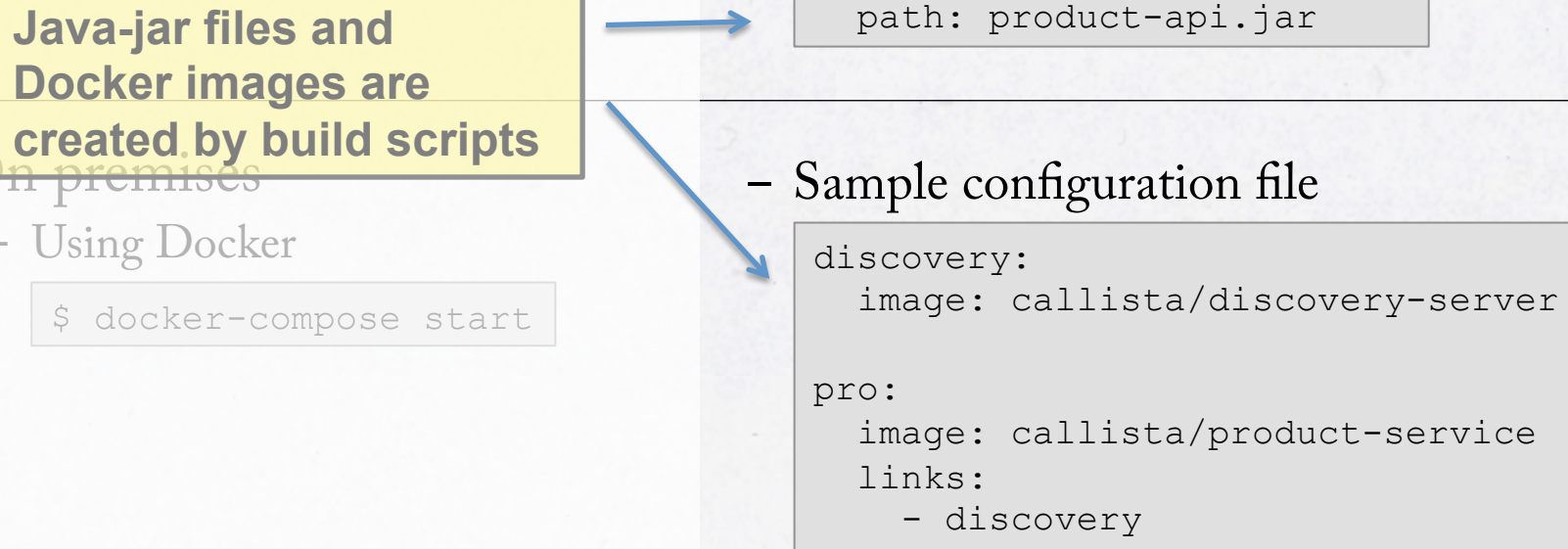
```
discovery:  
  image: callista/discovery-server  
  
pro:  
  image: callista/product-service  
  links:  
    - discovery
```

DEPLOY

- In cloud
 - Using PaaS: [Pivotal Web Services](#)

```
$ cf push
```

**Java-jar files and
Docker images are
created by build scripts**



- On premises
 - Using Docker

```
$ docker-compose start
```

– Sample configuration file

```
---  
memory: 512M  
instances: 1  
applications:  
- name: product-api-service  
  path: product-api.jar
```

– Sample configuration file

```
discovery:  
  image: callista/discovery-server  
  
pro:  
  image: callista/product-service  
  links:  
    - discovery
```


DEMO SLIDES

- Discovery server
- Centralized log analysis
- Scale up
- Resilience

THE DISCOVERY SERVER

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EDGESERVER	n/a (1)	(1)	UP (1) - 172.17.0.70:edgeserver:b74a3b6279298de049546f78f8cde438
PRODUCT	n/a (1)	(1)	UP (1) - 172.17.0.64:product:81409c2245b0135600a481972c9bfef8
PRODUCTAPI	n/a (1)	(1)	UP (1) - 172.17.0.68:productapi:9bb492a65a85c9e2d76e18adec3d5c09
PRODUCTCOMPOSITE	n/a (1)	(1)	UP (1) - 172.17.0.66:productcomposite:afb55f6fb35cd6a1fac33c6e0e1f6cd5
RECOMMENDATION	n/a (1)	(1)	UP (1) - 172.17.0.60:recommendation:56ba137a59ceeb7118f4431b90f76d1a
REVIEW	n/a (1)	(1)	UP (1) - 172.17.0.62:review:3db2f7d0117f6041e87359b6c25b29e6

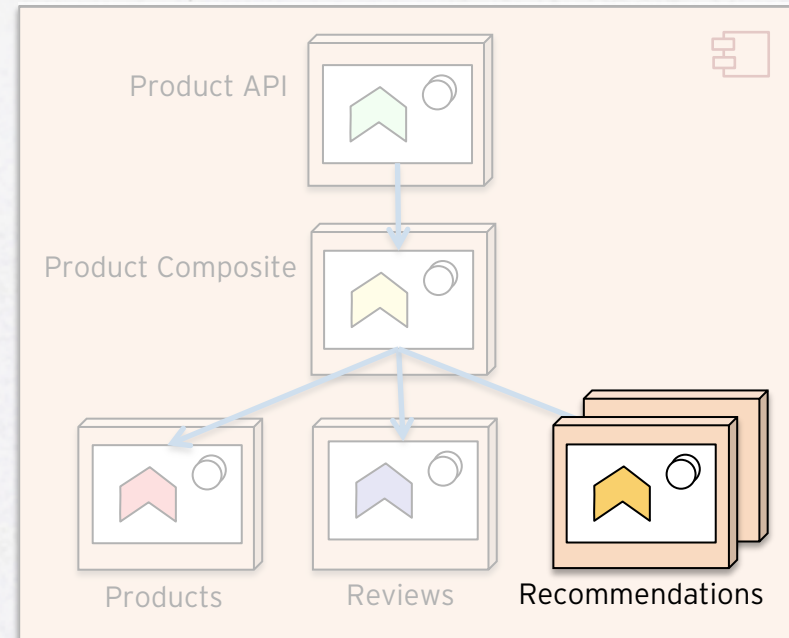
CENTRALIZED LOG ANALYSIS - KIBANA

@timestamp ^>	< corrlid >	< _type >	< message
2015-05-09T08:53:46.141+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-api	execute command: getProductComposite
2015-05-09T08:53:46.142+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-api	ProductApi: User=user, Auth=Bearer e272fb85-6b
2015-05-09T08:53:46.154+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-composite	execute command: getProduct
2015-05-09T08:53:46.163+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product	/product called
2015-05-09T08:53:46.170+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-composite	execute command: getRecommendations
2015-05-09T08:53:46.171+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-composite	GetRecommendations...
2015-05-09T08:53:46.177+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	recommendation	/recommendation called, processing time: 147
2015-05-09T08:53:46.326+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	recommendation	/recommendation response size: 3
2015-05-09T08:53:46.340+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-composite	execute command: getReviews
2015-05-09T08:53:46.341+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-composite	GetReviews...
2015-05-09T08:53:46.348+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	review	/reviews called, processing time: 109
2015-05-09T08:53:46.460+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	review	/reviews response size: 3
2015-05-09T08:53:46.473+02:00	5a1e7038-938e-44dc-bad1-18433c7fae3d	product-api	GetProductComposite http-status: 200

SCALE UP

- Let's scale up one of the services

```
$ docker-compose scale rec=2
...
$ docker-compose ps
Name
-----
api_1
rec_1
rec_2
...
```



@timestamp ^>	< HOSTNAME >	< _type >	< corrlid >	< message >
2015-05-09T08:53:42.539+02:00	8c0edf567efd	recommendation	623010bf-677d-4ea0-ae3c-0683550240e4	/recommendation called, processing time: 175
2015-05-09T08:53:42.717+02:00	8c0edf567efd	recommendation	623010bf-677d-4ea0-ae3c-0683550240e4	/recommendation response size: 3
2015-05-09T08:53:44.915+02:00	beea68b76d07	recommendation	09b176af-4093-48f4-973e-a6f0f8489726	/recommendation called, processing time: 122
2015-05-09T08:53:45.040+02:00	beea68b76d07	recommendation	09b176af-4093-48f4-973e-a6f0f8489726	/recommendation response size: 3

SCALE UP

- The new service instance in the discovery server

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EDGESERVER	n/a (1)	(1)	UP (1) - 172.17.0.23:edgeserver:bf311b440f4e4f66c87815173ec6787d
PRODUCT	n/a (1)	(1)	UP (1) - 172.17.0.17:product:572cd15b44ca1cfdc0ca2b23b885998f
PRODUCTAPI	n/a (1)	(1)	UP (1) - 172.17.0.21:productapi:6d9e4ec6da84fdb41701efd737e4fe51
PRODUCTCOMPOSITE	n/a (1)	(1)	UP (1) - 172.17.0.19:productcomposite:10bca9e845a5871cf6372fbea71105b0
RECOMMENDATION	n/a (2)	(2)	UP (2) - 172.17.0.13:recommendation:dd364a10b6e735e834821137ea8ffe62 , 172.17.0.11:recommendation:cae6e1ce5527cafab1bb854f2c93eac8
REVIEW	n/a (1)	(1)	UP (1) - 172.17.0.15:review:46fec4812d0971b45adaee0e0aef635c

CALL THE API

- Get an access token from the OAuth Authentication Server

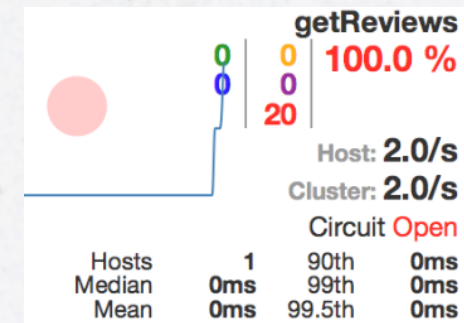
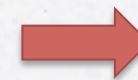
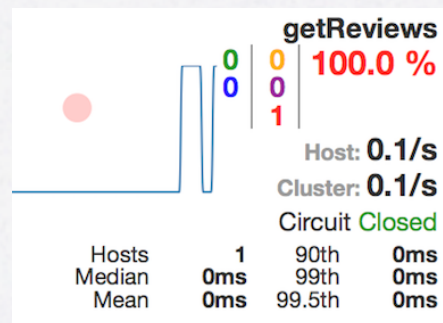
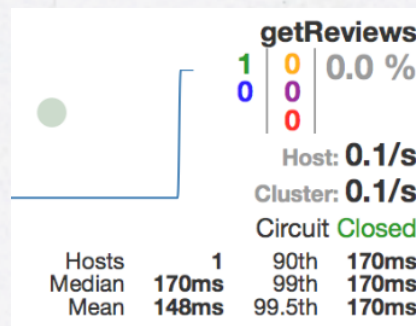
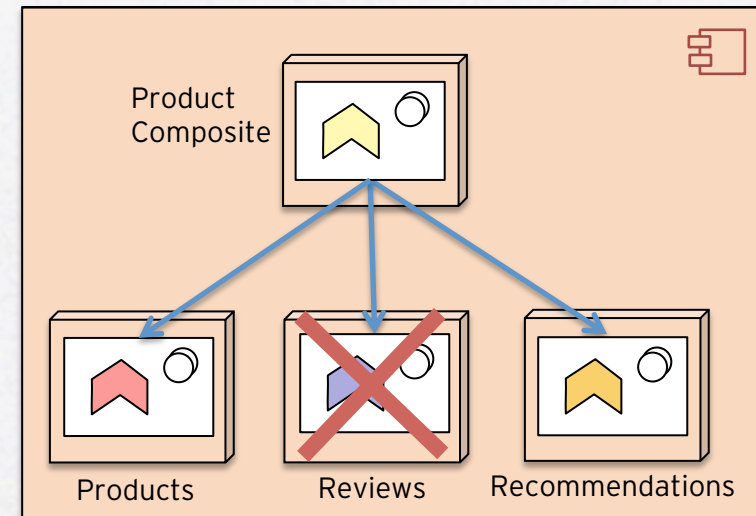
```
$ curl -s acme:acmesecret@docker:9999/uaa/oauth/token \
  -d grant_type=password -d client_id=acme \
  -d username=user -d password=password | jq .
{"access_token": "e5863174-6a25-4e4d-9fe0-32532a842d88", ...}
```

- Call the API with the access token

```
$ curl -s 'http://docker:8765/api/product/12345' \
  -H "Authorization: Bearer $TOKEN" | jq .
{
  "productId": 12345, "name": "name", ...
  "recommendations": [ {...}, {...}, {...} ],
  "reviews": [ {...}, {...}, {...} ]
}
```


CIRCUIT BREAKER

- Introduce an error
 - The review service stops to response, requests just hangs until requests timeout
- Try out
- Force the Circuit to open
 - Coming requests will fast-fail, i.e. not wait for the timeout!



CIRCUIT BREAKER

- Normal calls (circuit **closed**):

```
$ curl 'http://docker:8765/api/product/12345' ...  
{"productId": ..., "recommendations": [...], "reviews": [...]}  
0.398 ms
```

- Calls with a few timeouts (circuit still **closed**):

```
$ curl 'http://docker:8765/api/product/12345' ...  
{"productId": ..., "recommendations": [...], "reviews": null}  
3.295 ms
```

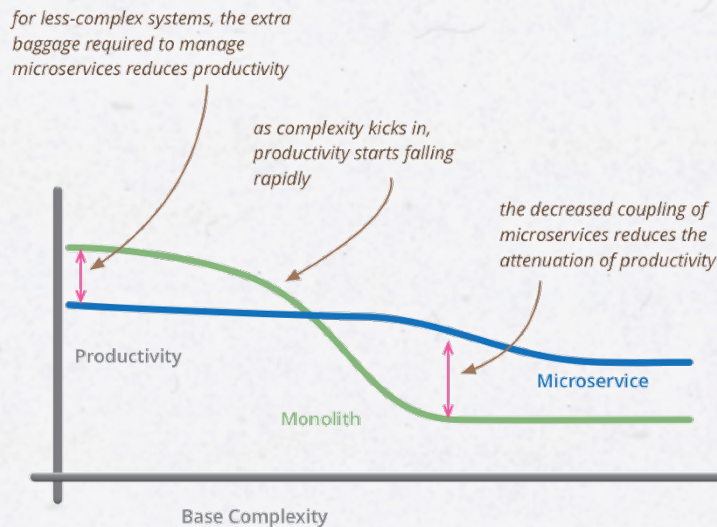
- Calls with a lot of timeouts (circuit **open**, i.e. it will **fast-fail**):

```
$ curl 'http://docker:8765/api/product/12345' ...  
{"productId": ..., "recommendations": [...], "reviews": null}  
0.239 ms
```

■ ...WHAT WE DIDN'T HAVE TIME TO TALK ABOUT (THIS TIME)

- The CAP theorem and distributed systems, eventual consistency...
- Conway's law requires organizational changes
- Continuous Delivery, a pre-requisite for large-scale use of microservices
- Building microservices, try out our blog series (<http://goo.gl/6nYXCD>)
- How to apply TDD for microservices?
- Configuration of microservices
- When to apply microservices?
 - See blog posts by Sam Newman and Martin Fowler (<http://goo.gl/e6A5Zb>, <http://goo.gl/ifKBmb>)

...WHAT WE DIDN'T HAVE TIME TO TALK ABOUT (THIS TIME)



Source: <http://goo.gl/ifKBmb>

but remember the skill of the team will outweigh any monolith/microservice choice



Kent Beck
@KentBeck

Follow

any decent answer to an interesting question begins, "it depends..."

7:45 PM - 6 May 2015



380



240

- When to apply microservices?
 - See blog posts by Sam Newman and Martin Fowler (<http://goo.gl/e6A5Zb>, <http://goo.gl/ifKBmb>)

SUMMARY

- Microservices use new solutions to old problems regarding
 - Scalability, resilience, release cycles
- Microservices is about splitting up monoliths in units of independently replaceable and upgradeable components
- Uses infrastructure for scaling out on many small servers
 - In cloud or on premises
- New advanced, battle-proven and open source tools for handling challenges with microservices
 - Netflix OSS, Spring Cloud and the ELK stack

WHAT TO DO NEXT?

- Short term:
 - Cherry pick specific components to address current problems, e.g.
 - » the ELK stack for improved log analyses
 - » a Circuit Breaker for improved resilience
 - Familiarize yourself with the microservices architecture
- Mid term:
 - Assess your application portfolio and identify pain points
 - Perform a pilot project with one prioritized application
- Long term:
 - Establish a strategic plan for microservices
 - Fully implement and deploy microservices for one application

Q&A

