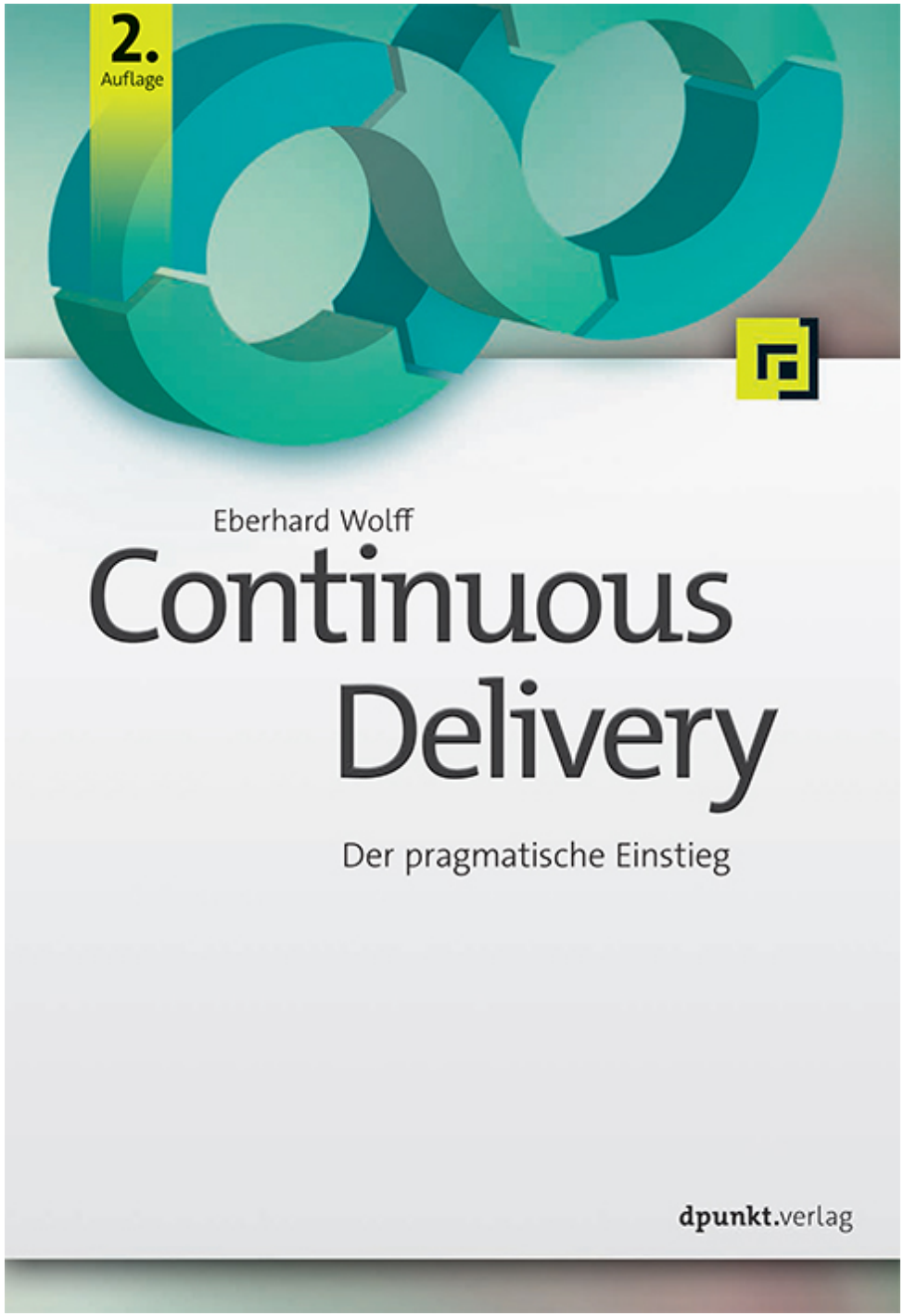
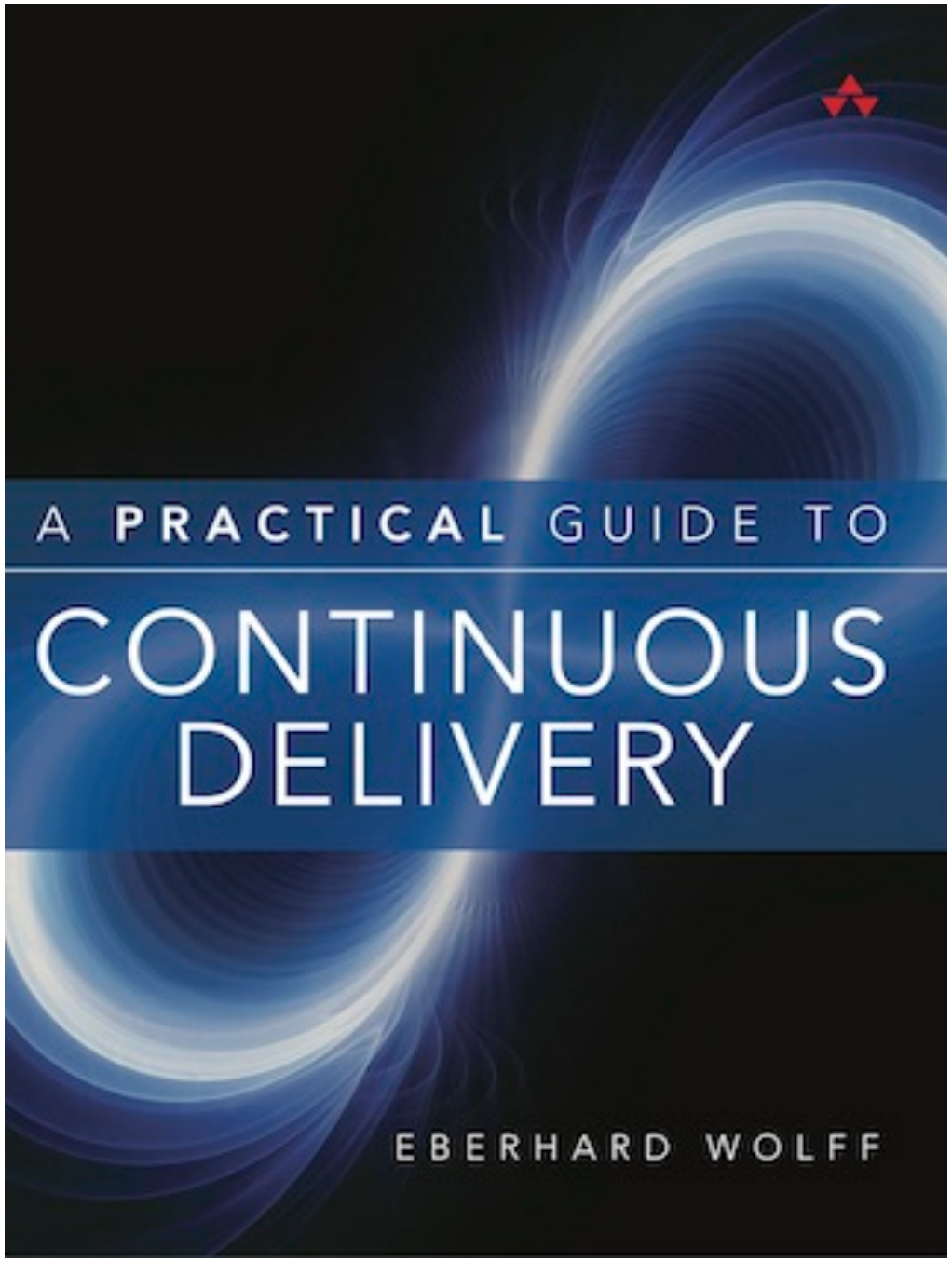
Four times Microservices: REST, Kubernetes, UI Integration, Async

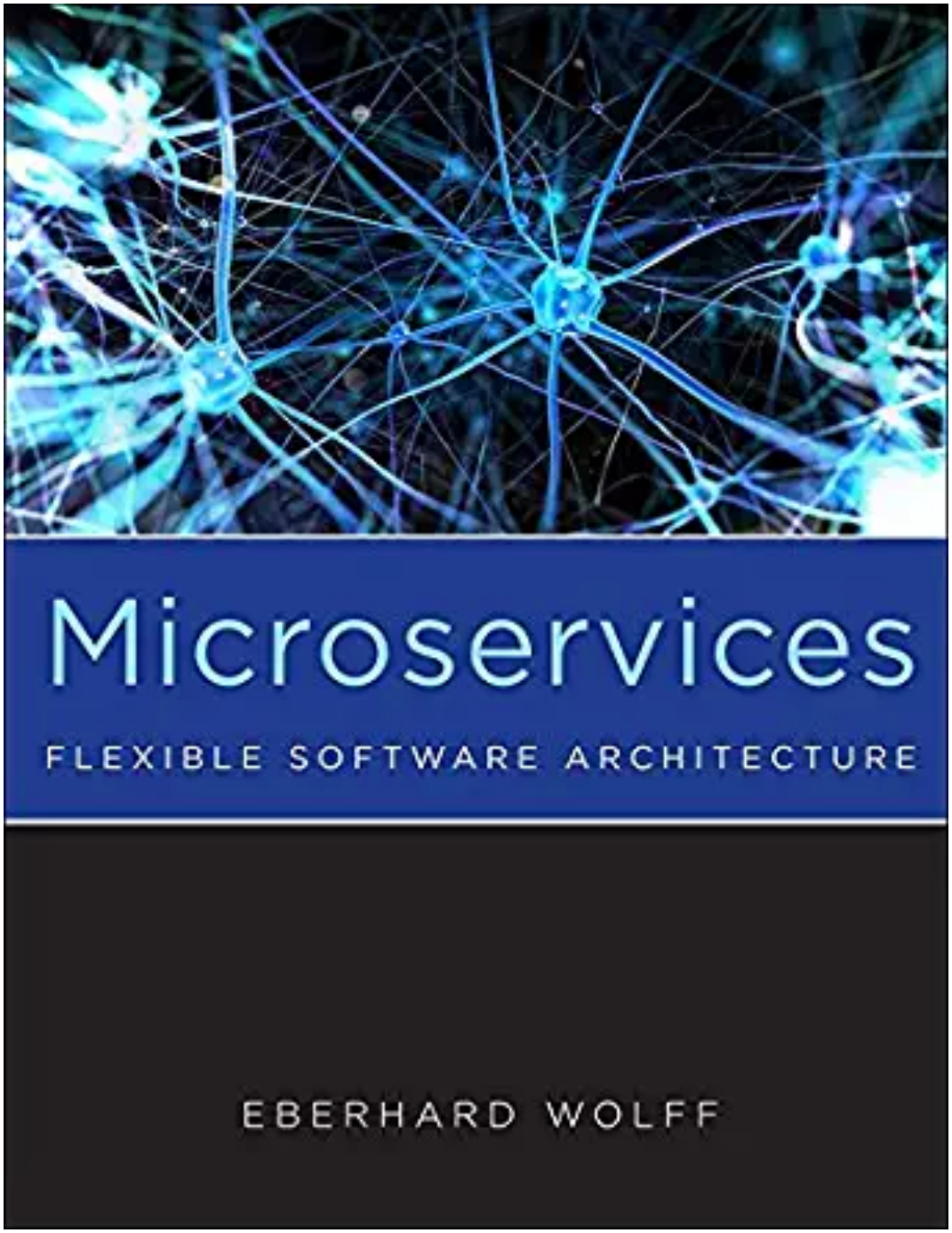
Eberhard Wolff
@ewolff
http://ewolff.com
Fellow





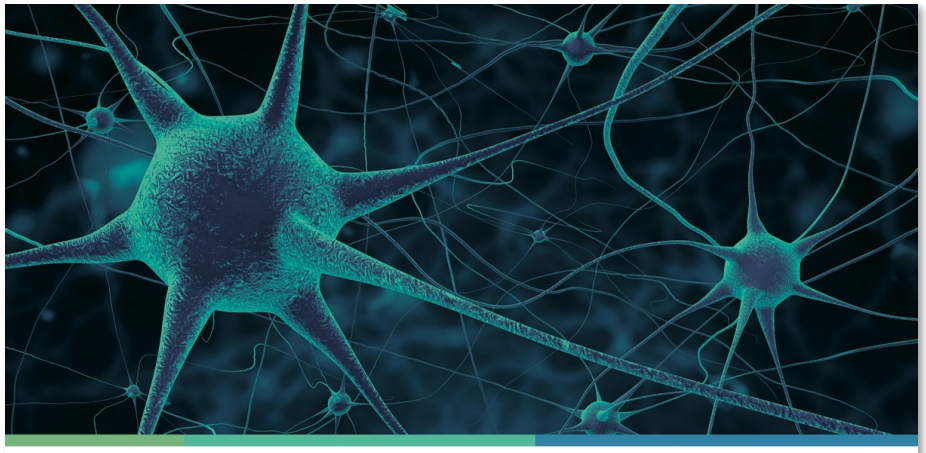
http://continuous-delivery-buch.de/

http://continuous-delivery-book.com/



http://microservices-buch.de/

http://microservices-book.com/

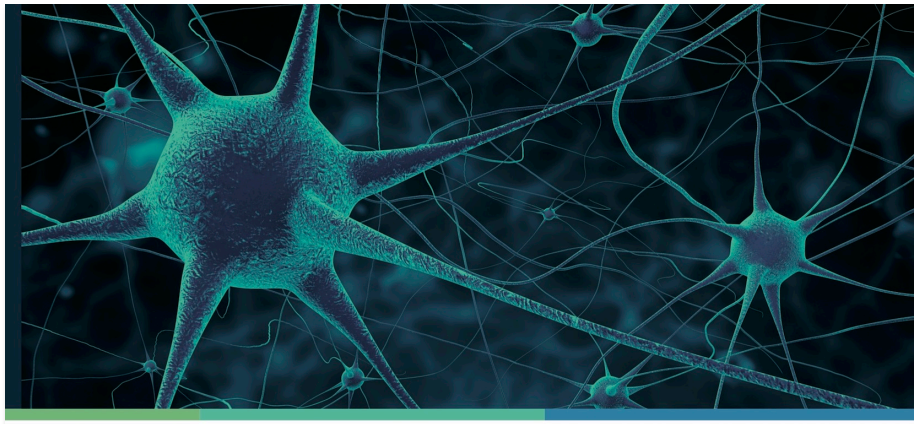


Eberhard Wolff

Microservices Primer

A Short Overview

innoQ



Eberhard Wolff

Microservices

Ein Überblick

innoQ

FREE!!!!

http://microservices-book.com/primer.html

http://microservices-buch.de/ueberblick.html

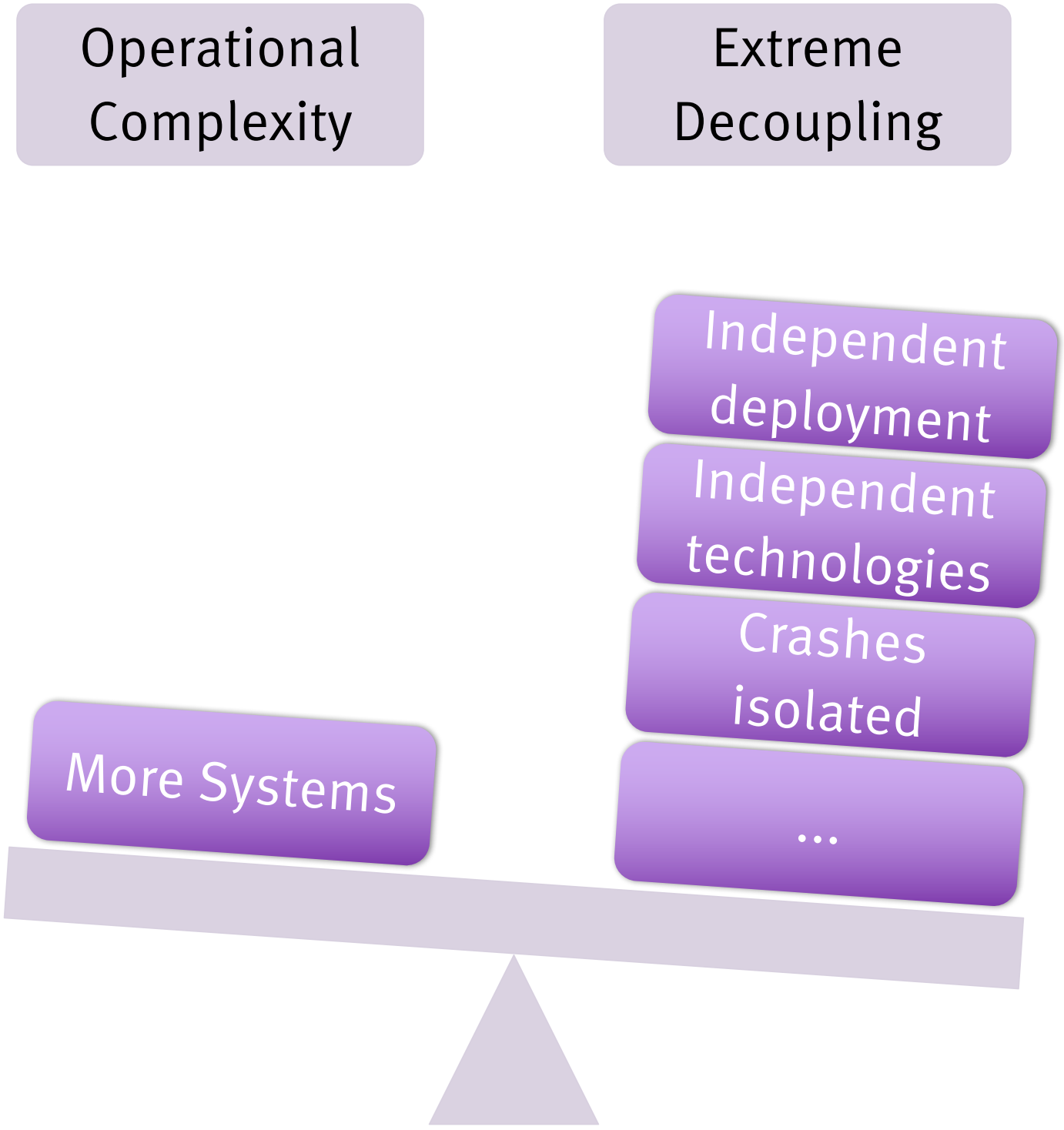
What are Microservices?

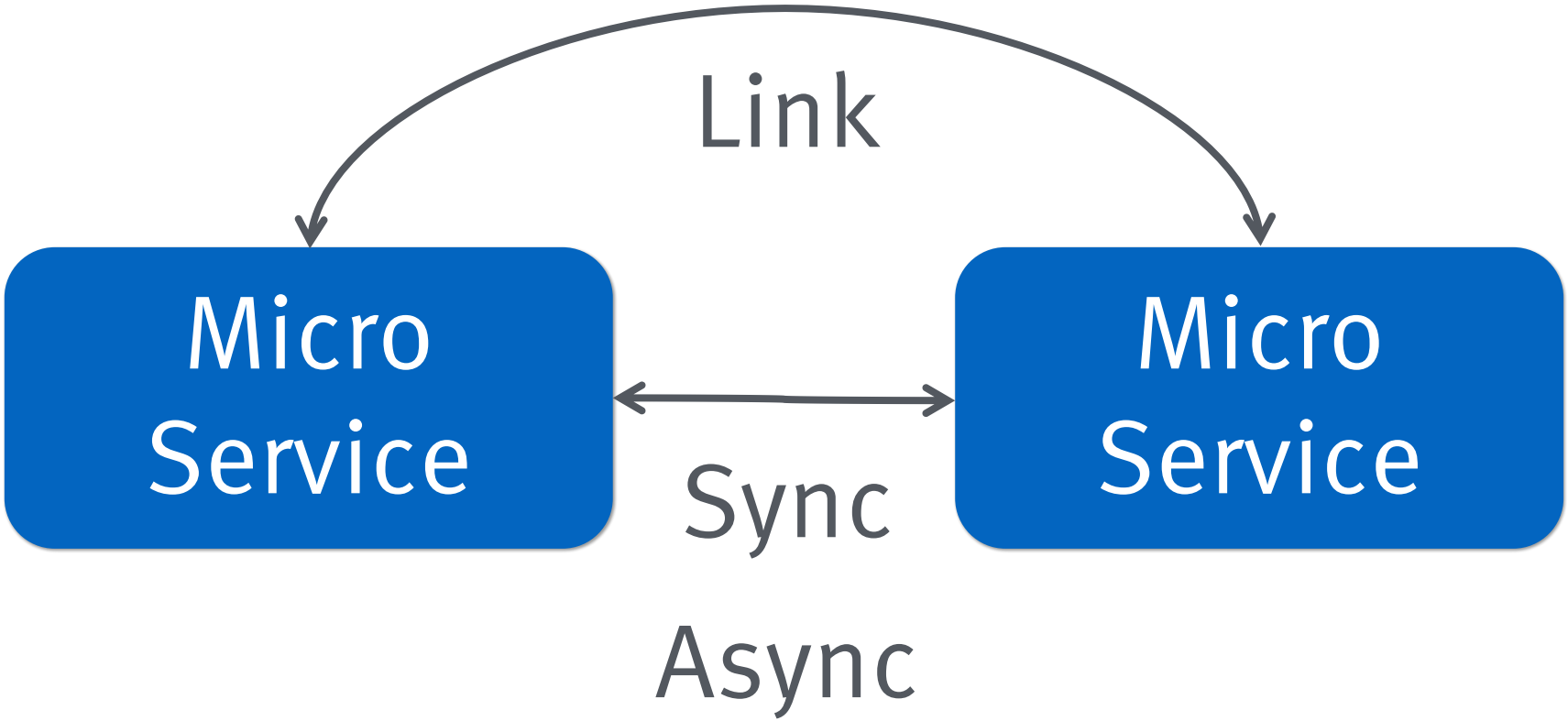
- *Modules* providing *interfaces*
- Processes, Containers, virtual machines

What are Microservices?

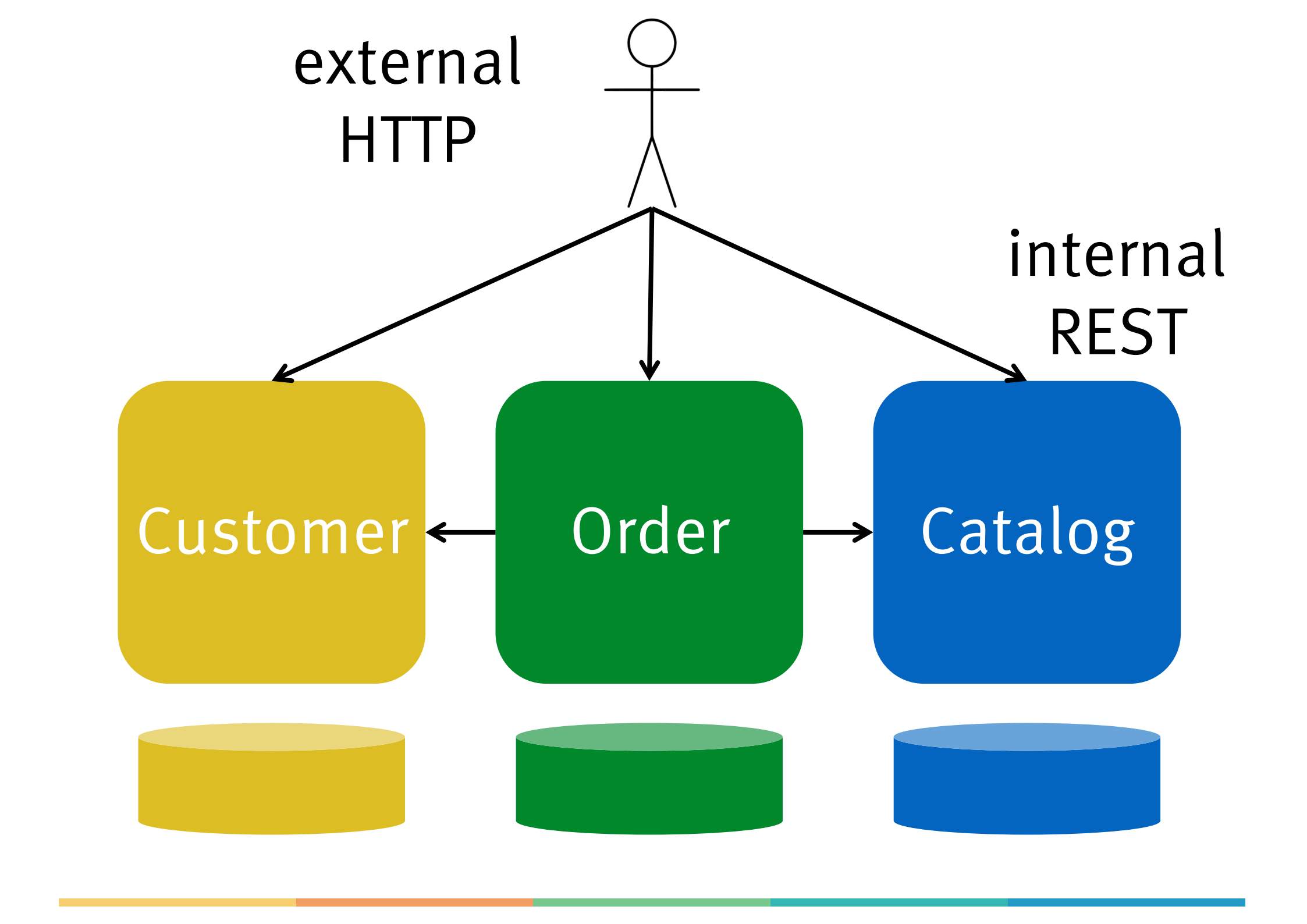
- *Independent* Continuous Delivery Pipeline including tests
- *Standardized* Operations (configuration, log analysis, tracing, monitoring, deployment)
- *Resilient* (compensate failure, crashes ...) therefore separate processes, VM, Docker containers

Microservices = Extreme Decoupling


Operational Complexity
Extreme Decoupling
Independent deployment
Independent technologies
Crashes isolated
...
More Systems


Link
Micro Service
Micro Service
Sync
Async

Synchronous REST Microservices


external
HTTP
internal
REST
Customer
Order
Catalog

Synchronous Microservices

- Service Discovery:
  IP / port?
- Load Balancing:
  Resilience and independent scaling
- Routing:
  Route external request to microservices
- Resilience

Synchronous Microservices with Java, Spring Boot / Cloud & the Netflix Stack

https://github.com/ewolff/microservice

Service Discovery
Eureka

Why Eureka for Service Discovery?

- REST based service registry
- Supports replication
- Caches on the client
- Resilient
- Fast

- Foundation for other services

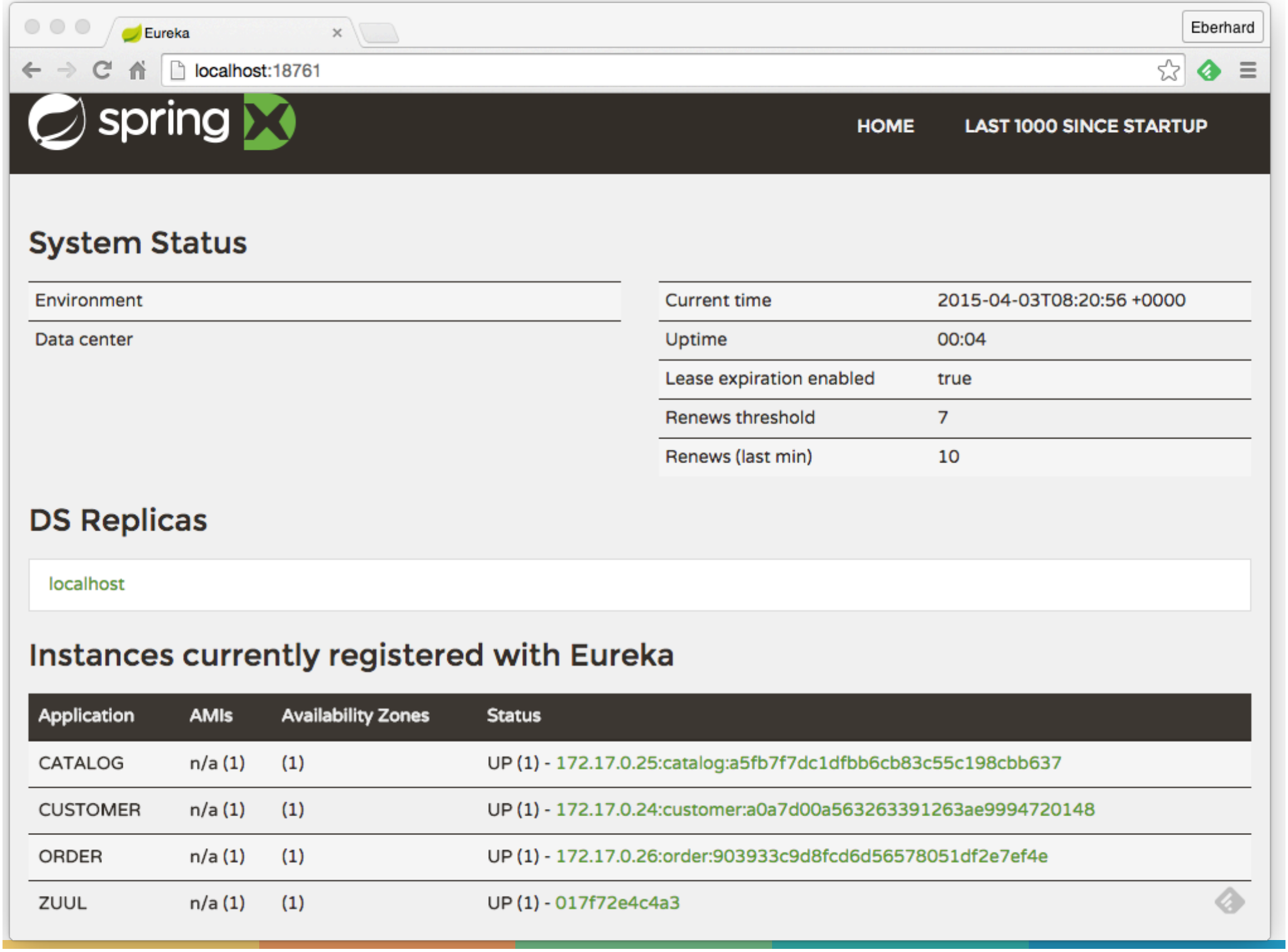
Eureka Client

- @EnableDiscoveryClient: generic
- @EnableEurekaClient: specific
- Dependency on spring-cloud-starter-eureka
- Automatically registers application

Eureka Server

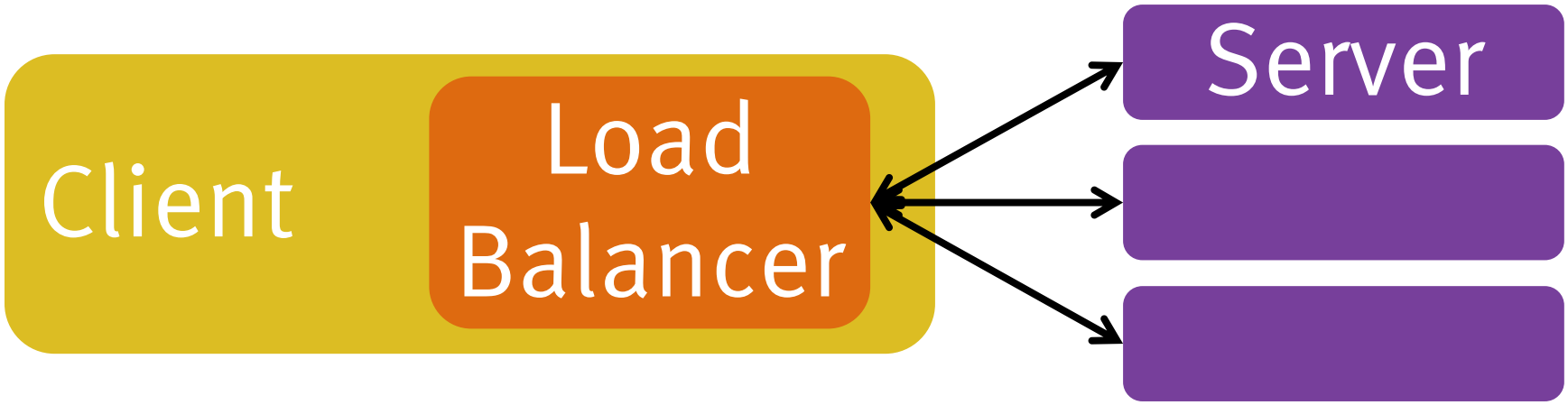
```
@EnableEurekaServer

@EnableAutoConfiguration

public class EurekaApplication {

    public static void main(String[] args) {

      SpringApplication.run(EurekaApplication.class, args);

    }

}
```

Add dependency to
spring-cloud-starter-eureka-server


Eureka
Eberhard
localhost:18761
spring X
HOME
LAST 1000 SINCE STARTUP
System Status
Environment
Data center
Current time
2015-04-03T08:20:56 +0000
Uptime
00:04
Lease expiration enabled
true
Renews threshold
7
Renews (last min)
10
DS Replicas
localhost
Instances currently registered with Eureka
Application
AMIs
Availability Zones
Status
CATALOG
n/a (1)
(1)
UP (1) - 172.17.0.25:catalog:a5fb7f7dc1dfbb6cb83c55c198cbb637
CUSTOMER
n/a (1)
(1)
UP (1) - 172.17.0.24:customer:a0a7d00a563263391263ae9994720148
ORDER
n/a (1)
(1)
UP (1) - 172.17.0.26:order:903933c9d8fcd6d56578051df2e7ef4e
ZUUL
n/a (1)
(1)
UP (1) - 017f72e4c4a3

Load Balancing Ribbon

Ribbon: Client Side Load Balancing



- Decentralized Load Balancing
- No bottle neck
- Resilient

Ribbon Example

Via Dependency Injection

```
private LoadBalancerClient loadBalancer;


…
ServiceInstance instance = loadBalancer.choose("CATALOG");
String url =  "http://" + instance.getHost() + ":”
 + instance.getPort() + "/catalog/";
```

Eureka name

Need dependency to spring-cloud-starter-ribbon

Resilience

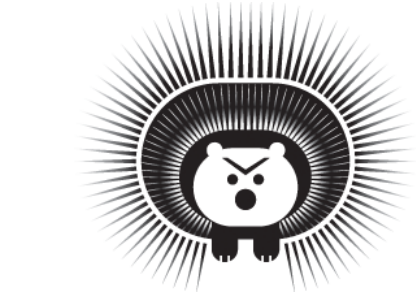
HYSTRIX

DEFEND YOUR APP

Timeout

- Do call in other thread pool
- Won’t block request handler
- Can implement timeout

Circuit Breaker



- Called system fails -› open
- Open -› do not forward call
- Forward calls after a time window

Hystrix & Spring Cloud

- Annotation based approach
- Annotations of javanica libraries

- Simplifies Hystrix
- No commands

Enable Hystrix

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableCircuitBreaker
public class OrderApp {


   public static void main(String[] args) {
     SpringApplication.run(OrderApp.class, args);
   }
}
```

Need spring-cloud-starter-hystrix

Fallback

```
@HystrixCommand(fallbackMethod = "getItemsCache",
commandProperties = { @HystrixProperty(name =
   "circuitBreaker.requestVolumeThreshold", value = "2") } )
public Collection<Item> findAll() {
   …
     this.itemsCache = pagedResources.getContent();
     return itemsCache;
}

private Collection<Item> getItemsCache() {
     return itemsCache;
}
```

Zuul Routing

Routing

- One URL to the outside
- Internal: Many Microservices
- REST
- Or HTML GUI

- Hides internal structure (sort of)
- Might add filters

Automatically maps route to server registered on Eureka

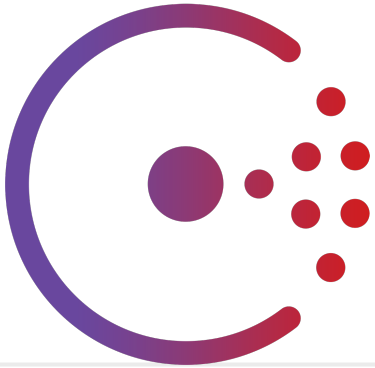
i.e. /customer/**
to CUSTOMER

No configuration

Netflix Stack

NETFLIX

- Service Discovery: Eureka
- Load Balancing: Ribbon
- Routing: Zuul
- Resilience: Hystrix
- Non-Java microservice:
  specific clients or sidecar



Consul

- Service Discovery by Hashicorp
- DNS interface
- Platform independent
- Consul Template can fill out config file templates
- e.g. for load balancer
- ...unaware of service discovery

https://github.com/ewolff/microservice-consul

Kubernetes

https://github.com/ewolff/microservice-kubernetes

Kubernetes

- Docker container on a single machine:
  Not resilient enough
- Kubernetes: Run Docker container in cluster
- ...and much more!

Pods

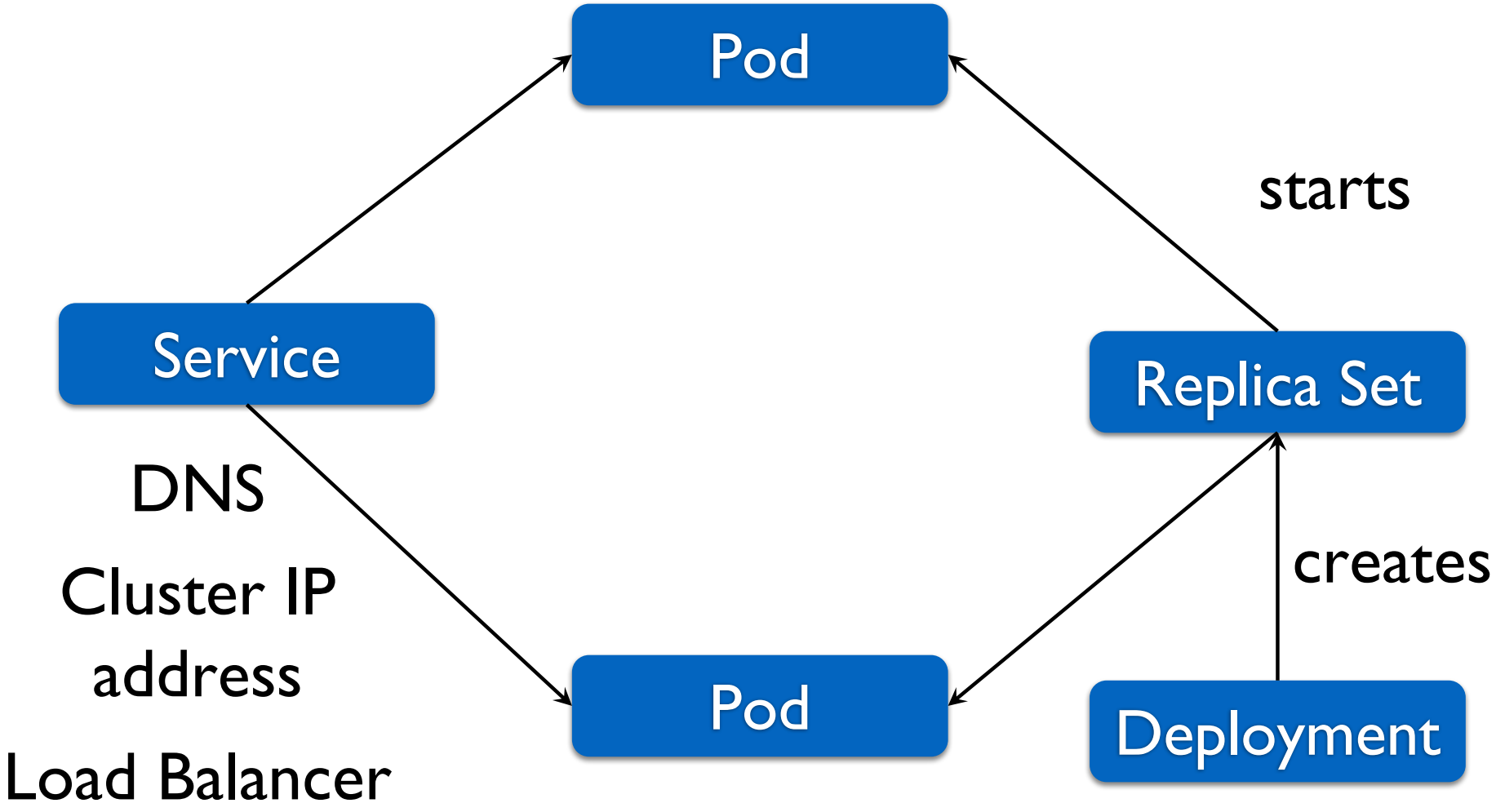
- Kubernetes runs *Pods*
- Pods = 1..n Docker containers
- Shared volumes
- ...and ports

Replica Sets

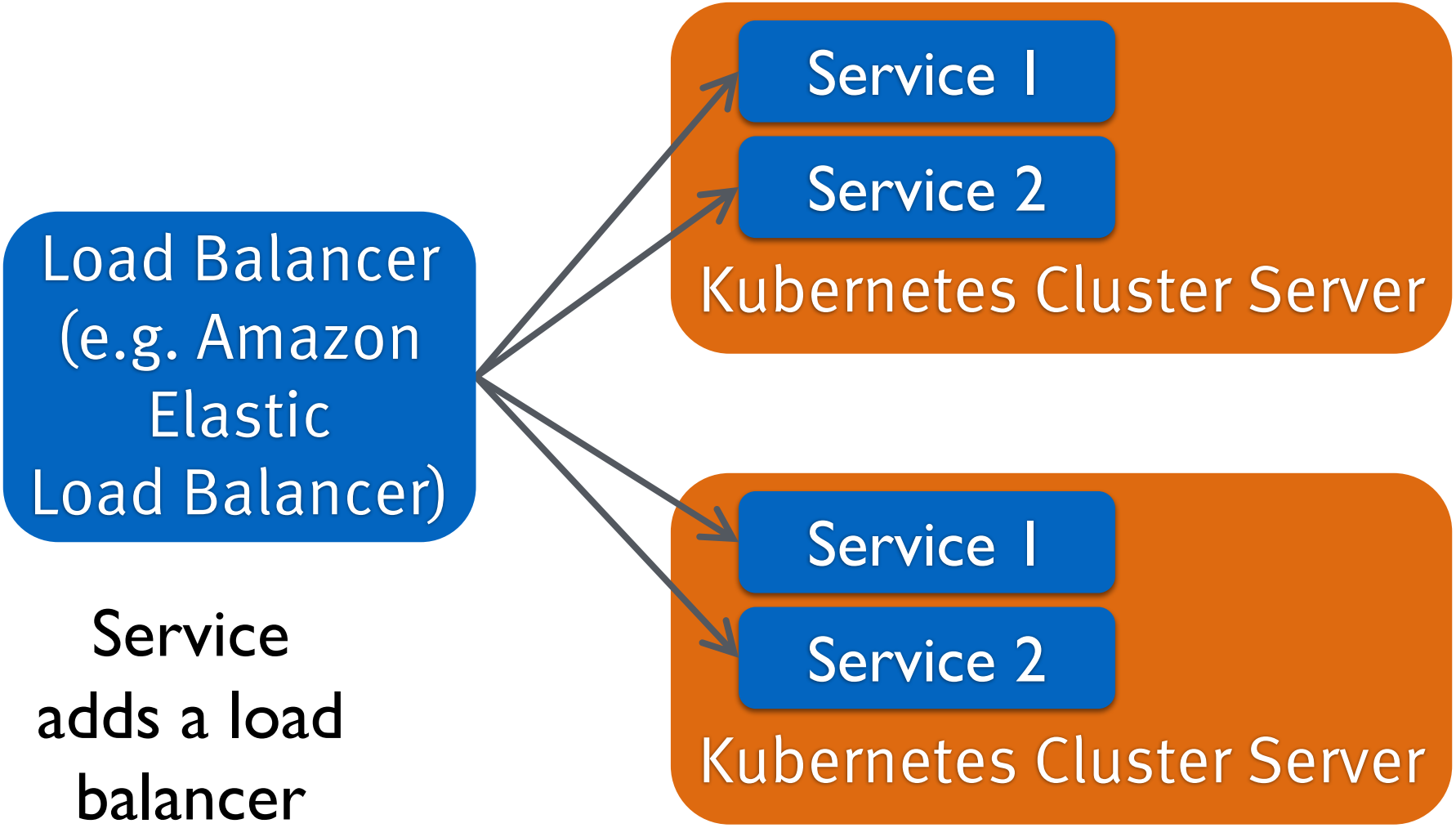
- *Deployment* creates replica set
- *Replica sets* ensure that a certain number of Pods run
- ...for load balancing / fail over

Services

- *Services* ensure access to the Pods
- DNS entry
- Cluster-wide unique IP address for internal access
- Node port …
- … or external load balancer for external access


Pod
starts
Service
Replica Set
DNS
Cluster IP
address
Load Balancer
Pod
creates
Deployment

Load Balancer



Service adds a load balancer

Kubernetes

- Service Discovery: DNS
- Load Balancing: IP
- Routing: Load Balancer or Node Port
- Resilience: Hystrix? envoy proxy?
- Can use any language

No code dependencies on Kubernetes!

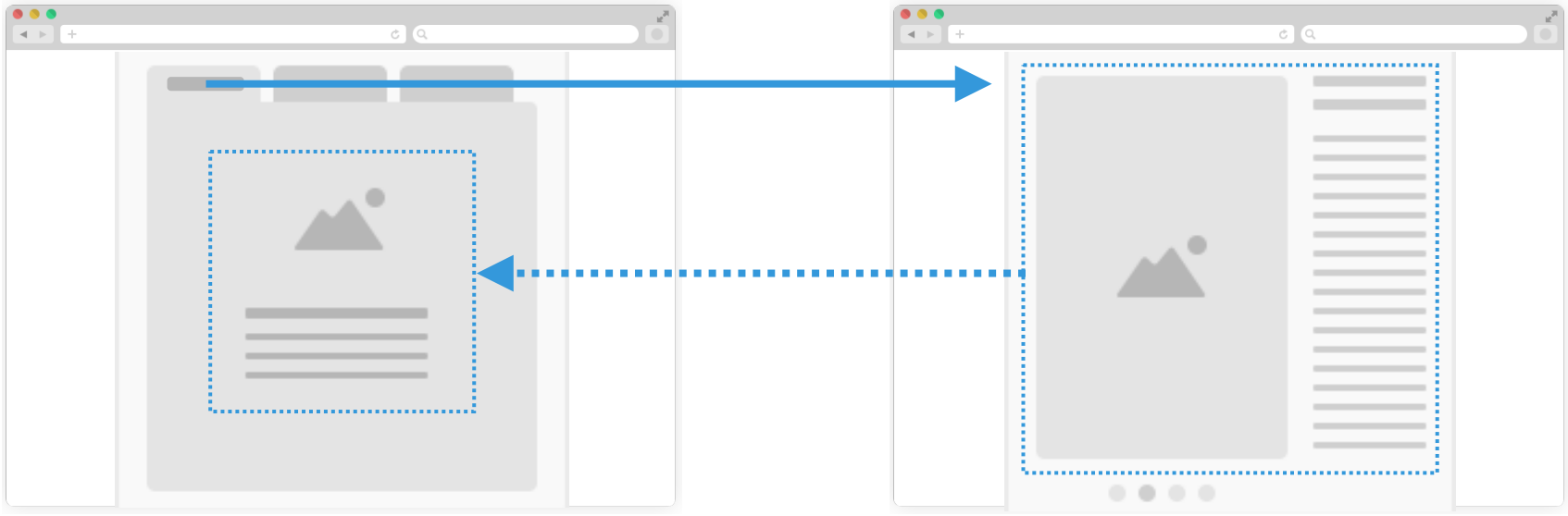
UI Integration

UI Integration

- Very powerful
- Often overlooked
- UI should be part of a microservices
- Self-contained System emphasize UI
- http://scs-architecture.org/

System 1 System 2

Hyperlinks to navigate between systems.



System 1 System 2

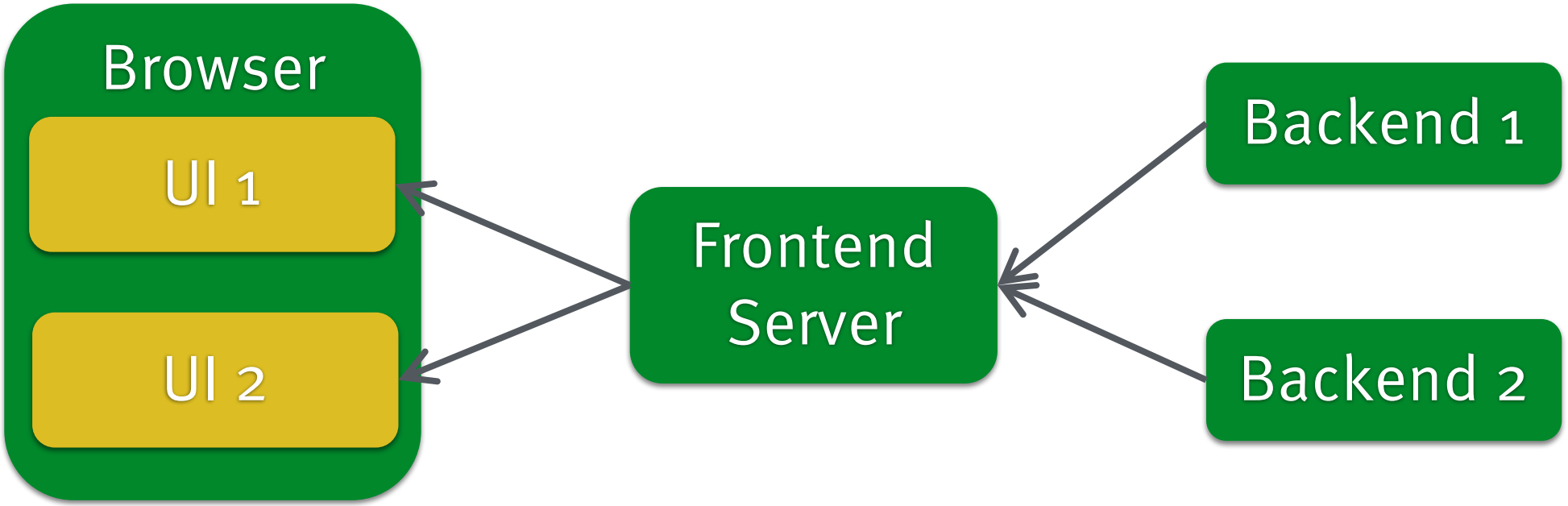
Transclusion of content served by another application

https://www.innoq.com/en/blog/transclusion/

UI Integration

- Extremely decoupled
- Here is a link: http://ewolff.com
- No need to know anything else about the system
- What about more complex transclusion?

Server-side integration



Server-side Integration: Technologies

- ESI (Edge Side Include)
- E.g. Varnish cache

- SSI (Server Side Include)
- E.g. Apache httpd, Nginx

ESI (Edge Side Includes)

```
...
<header>
... Logged in as: Ada Lovelace  ...
</header>
...
<div>
  ... a lot of static content and images ...
</div>
...
<div>
  some dynamic content
</div>
```

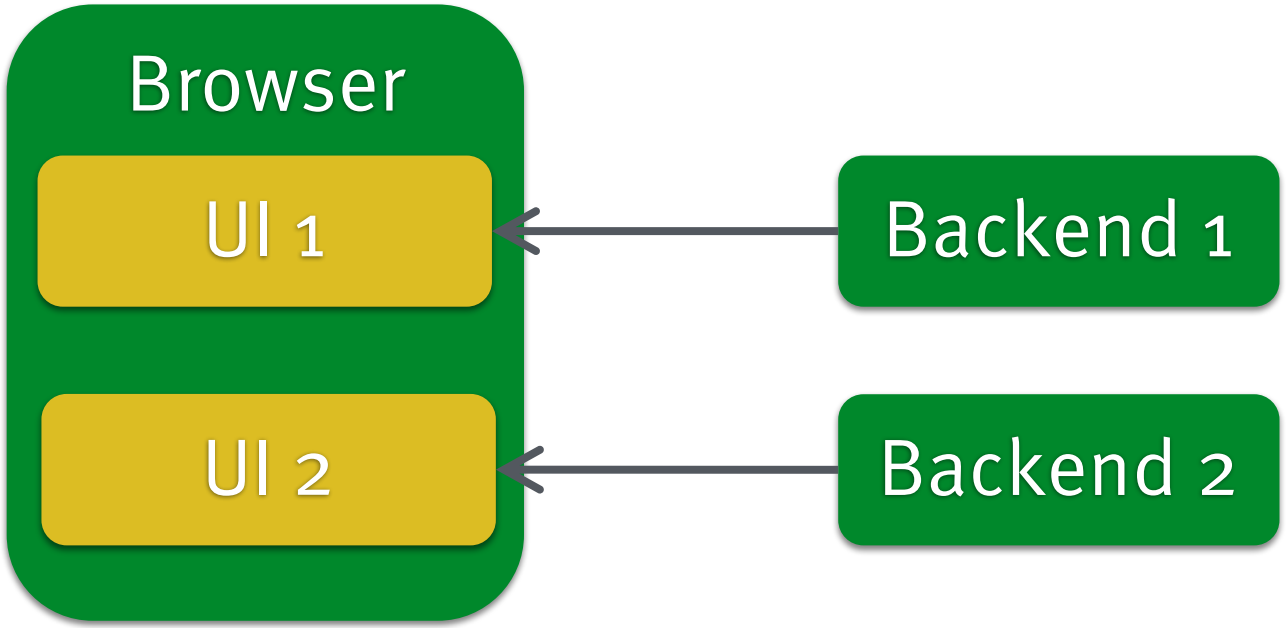
ESI (Edge Side Includes)

```
...

<esi:include src="http://example.com/header" />

...
<div>
  ... a lot of static content and images ...
</div>
...

<esi:include src="http://example.com/dynamic" />
```

http://www.w3.org/TR/esi-lang

https://github.com/ewolff/SCS-ESI

Client-side integration



Client-side Integration: Code

Replace ‹a href= " " class= "embeddable"›
with content of document in a ‹div› (jQuery)

```
$("a.embeddable").each(function(i, link) {
 $("‹div /›").load(link.href, function(data, status, xhr) {
  $(link).replaceWith(this);
  });
});
```

https://github.com/ewolff/SCS-jQuery

https://github.com/ewolff/crimson-insurance-demo

Asynchronous Microservices

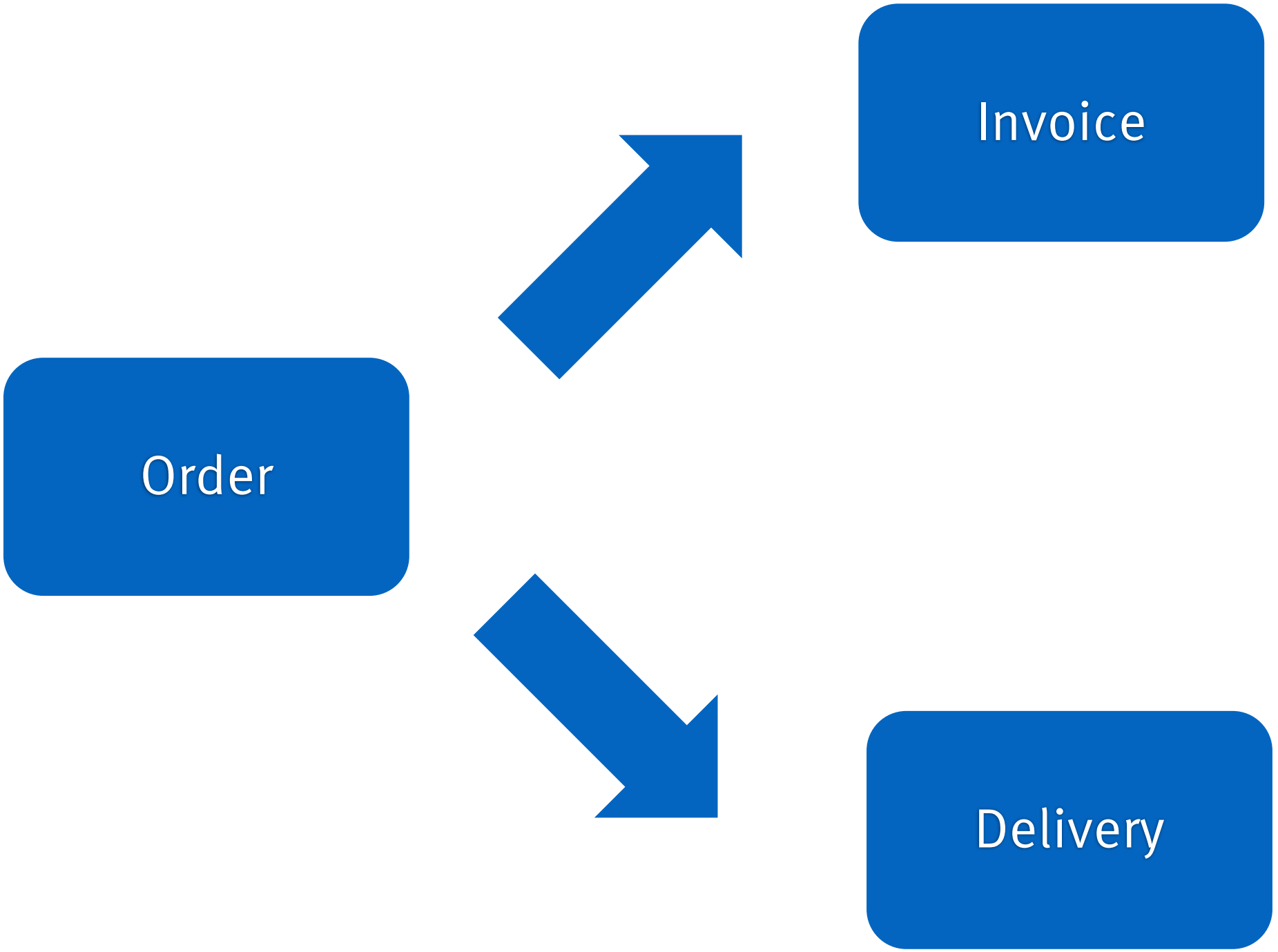
Asynchronous

- Do not talk to other microservices
- ...and wait for a reply
- ...while processing a request.

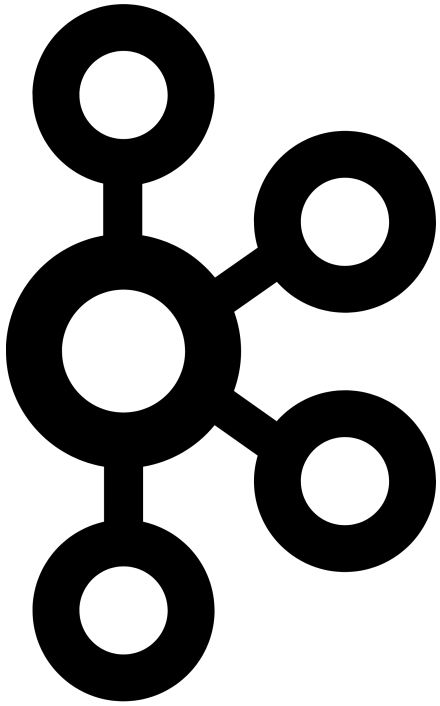
- Either don‘t wait for a reply
- ...or don‘t communicate while processing a request.

Asynchronous: Benefits

- Deals with unreliable systems
- Can guarantee delivery
- Good fit for events
- ...and Bounded Context / DDD

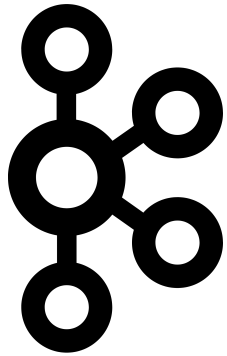

Invoice
Order
Delivery

Kafka



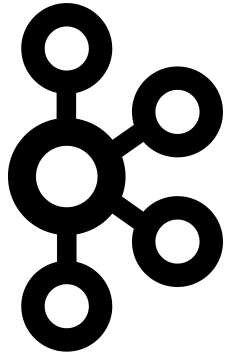
https://github.com/ewolff/microservice-kafka

Kafka API



- Producer API
- Consumer API
- Streams API (transformation)

Kafka Records

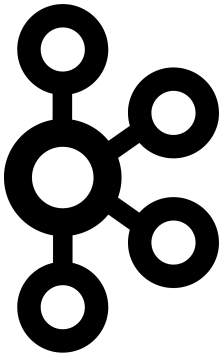


- Key
- Value
- Timestamp

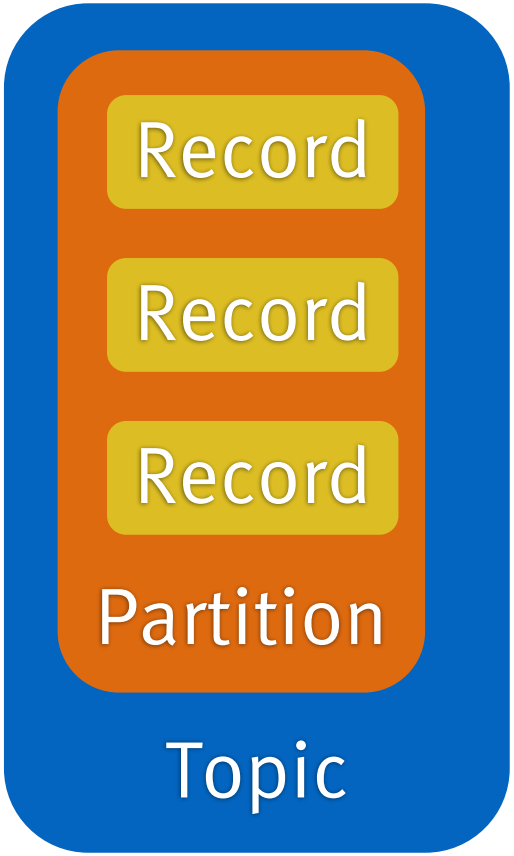
Record
Key
Value
Timestamp

- No headers
- Stored forever (!)

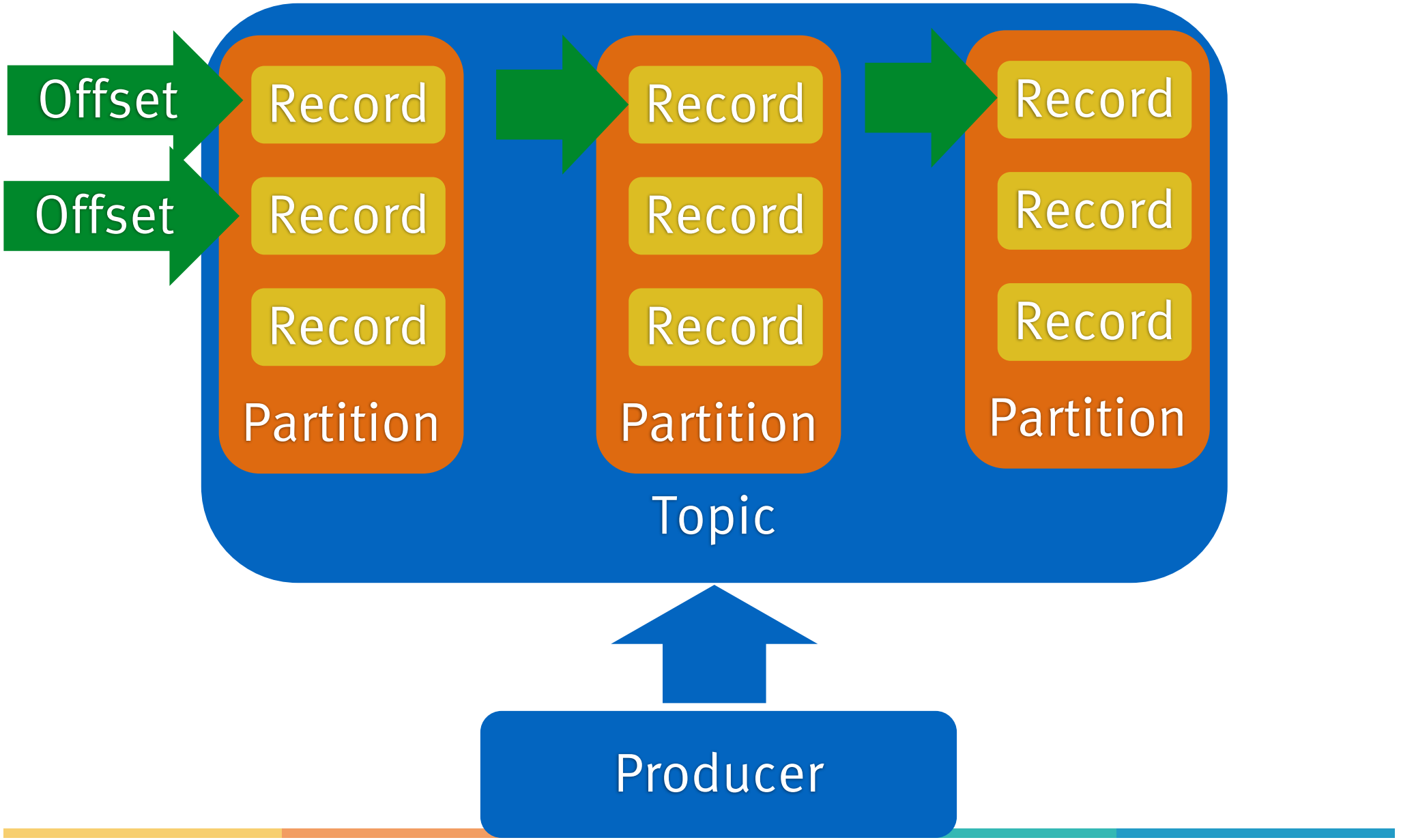
Kafka Topics



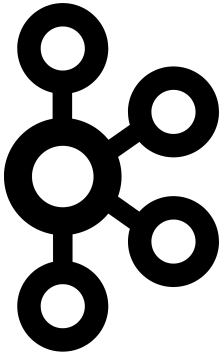
- Named
- Topics have partitions
- Order guaranteed per partition
- Consumer commits offset per partition
- Consumer group: One consumer per partition



Offset committed per consumer

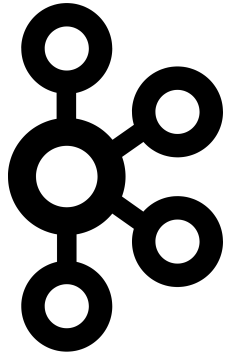


Log Compaction



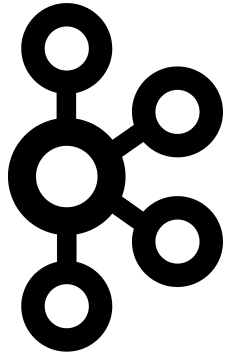
- Remove all but the latest record with a given key
- Idea: Old events not relevant
- ...if new events override
- Efficient storage in the long run

Kafka Replication



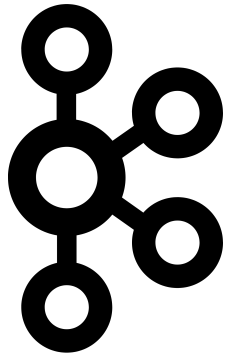
- N Replicas (configurable)
- In-sync replicas (configurable):
  write successful if all written

Asynchronous Microservices



- Service Discovery:
  via topic
- Load Balancing:
  via partitions / consumer groups
- Routing:
  ./.
- Resilience
  service fails – latency increases

Kafka Conclusion



- Provides access to old events
- Guaranteed order per key
- (Log compaction also per key)
- Consumer groups send records to one consumer
- Additional (complex) infrastructure

REST & ATOM

https://github.com/ewolff/microservice-atom

Atom

- Data format
- ...for feeds, blogs, podcasts etc
- Access through HTTP

- Idea: Provide a feeds of events / orders

```
<feed xmlns="http://www.w3.org/2005/Atom">

  <title>Order</title>

  <link rel="self"
     href="http://localhost:8080/feed" />

  <author><name>Big Money Inc.</name> </author>

  <subtitle>List of all orders</subtitle>

  <id>https://github.com/ewolff/microservice-
  atom/order</id>

  <updated>2017-04-20T15:28:50Z</updated>

  <entry> ... </entry>

</feed>
```

```
<entry>

  <title>Order 1</title>

  <id>https://github.com/ewolff/microservice-
   atom/order/1</id>

  <updated>2017-04-20T15:27:58Z</updated>

  <content type="application/json“
    src="http://localhost:8080/order/1" />

  <summary>This is the order 1</summary>

</entry>
```

Atom Feed

- Some unneeded information
- ...but not a lot

- Mostly links to details
- ...and timestamps
- Can use content negotiation to provide different data formats or details

Subscribing to Feed

- Poll the feed (HTTP GET)
- Client decides when to process new events
- ...but very inefficient
- Lots of unchanged data

HTTP Caching

- Client GETs feed
- Server send data
- + Last-Modified header
- Client sends get
- + If-Modified-Since header
- Server: 304 (Not modified)
- ...or new data

REST can be asynchronous.

Atom

- Service Discovery:
  REST
- Load Balancing:
  REST
- Routing:
  ./.
- Resilience
  failures just increase latency

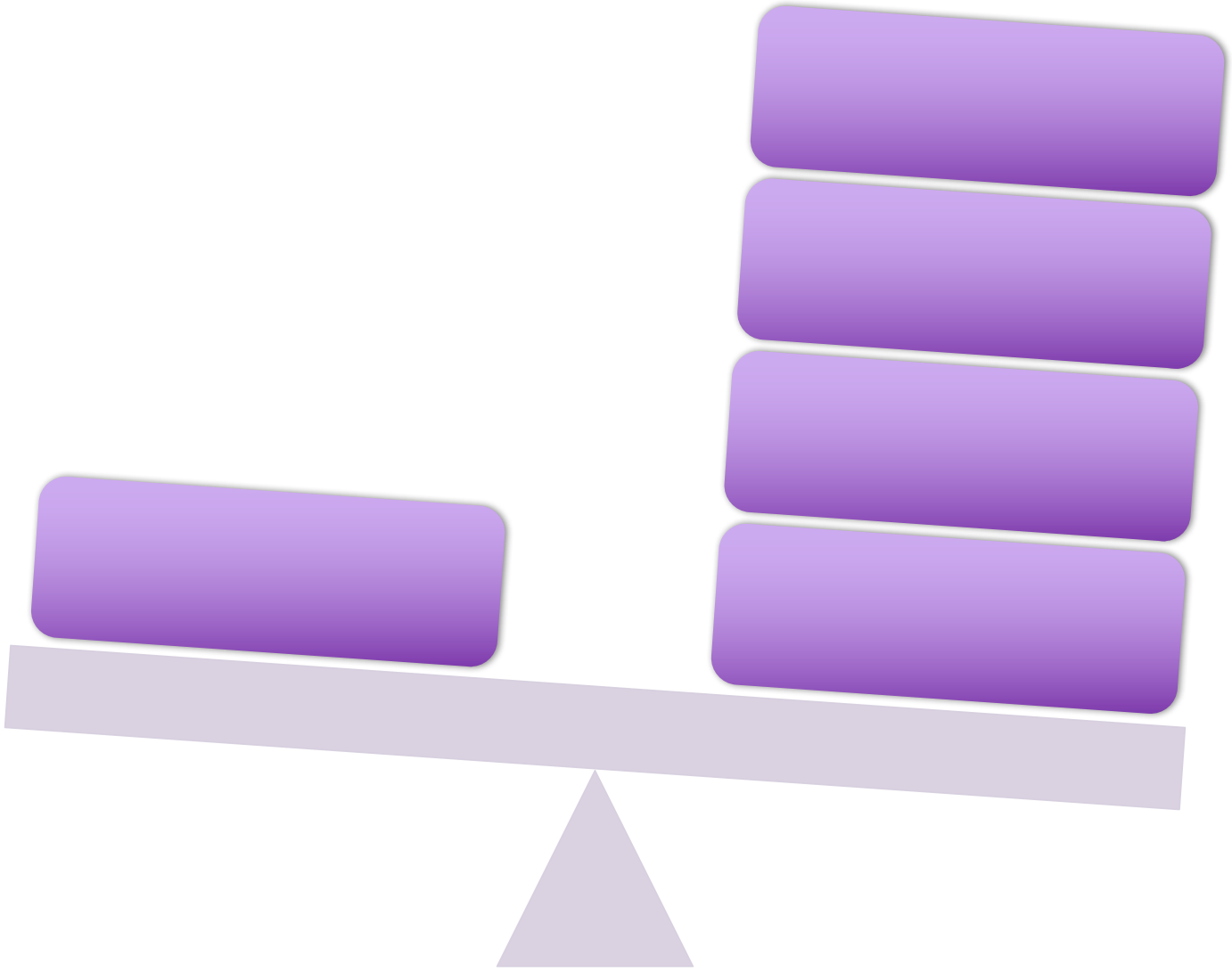
Atom Conclusion

- Can provides access to old events if they are stored anyway.
- One consumer: idempotency
- No additional infrastructure

Conclusion

Operational Complexity

Extreme Decoupling



Conclusion: Communication

- Netflix: Java focus, code dependencies
- Kubernetes: No code dependencies
- UI Integration: more decoupling
- Asynchronous deals with challenges in distributed systems

Links

- List of microservices demos (sync, async, UI): http://ewolff.com/microservices-demos.html
- REST vs. Messaging for Microservices https://www.slideshare.net/ewolff/rest-vs-messaging-for-microservices

EMail bedcon2017@ewolff.com to get:

Slides

+ Microservices Primer

+ Sample Microservices Book

+ Sample of Continuous Delivery Book

Powered by Amazon Lambda & Microservices