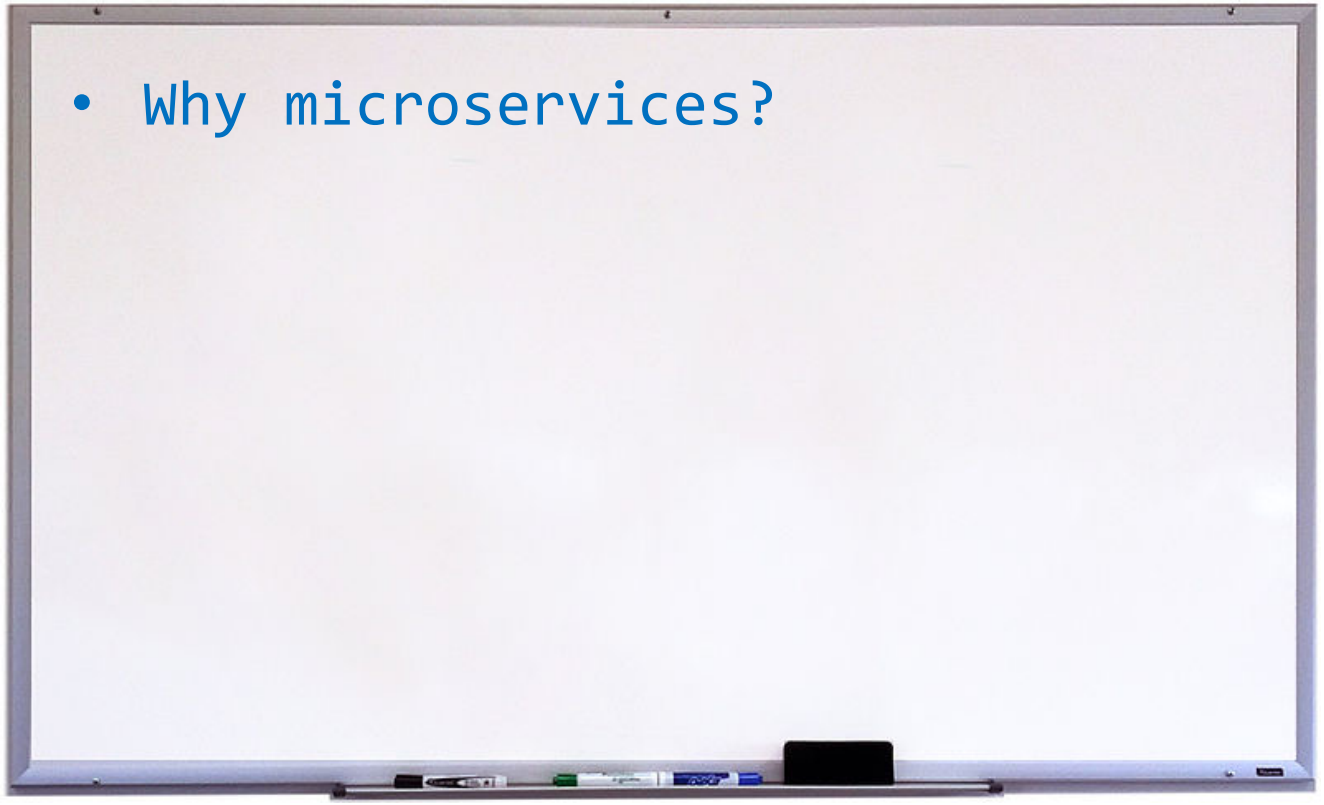
Microservices

Antonio Brogi

Department of Computer Science
University of Pisa


• Why microservices?

Why microservices?

Lead time

New feature’s design starts

First click of user

(1) Shorten lead time for new features (and updates)

→ accelerate rebuild and redeployment

→ reduce chords across functional silos

Why microservices?

(2) Need to scale, effectively

→ millions of users

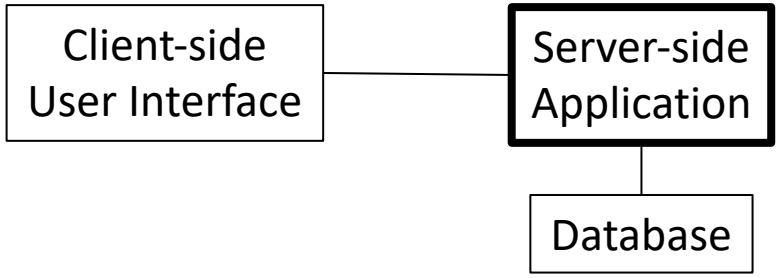
e.g. (source www.businessofapps.com)

- 422 million people use Spotify once a month, 182 million are subscribers
- In 2022, Netflix had 222 million subscribers worldwide


• Why microservices?
• Essence of microservices

~~Monoliths~~

- Typical Enterprise Application



Server-side app is a ***monolith*** (single logical executable)
  - handles HTTP requests
  - executes domain logic
  - queries/updates database
  - sends HTML views to client

- Cons of monoliths
  - changes made to small part of application require rebuilding and redeploying entire monolith
  - scaling requires scaling of entire application rather than parts of it that require more resources

J. Lewis, M. Fowler. Microservices – a definition of this new architectural term. March 2014.

Essence of microservices

1. Service-orientation

Develop applications as sets of stateless services:

- each running in its own ~~process~~ container
- communicating with lightweight mechanisms (RESTish protocols)
  - HTTP request-response with resource API
  - dumb message bus (e.g. asynchronous fabric like RabbitMQ)
- polyglotism
  - different languages, different storage technologies

~~ESBs~~ ("smart endpoints and dumb pipes")

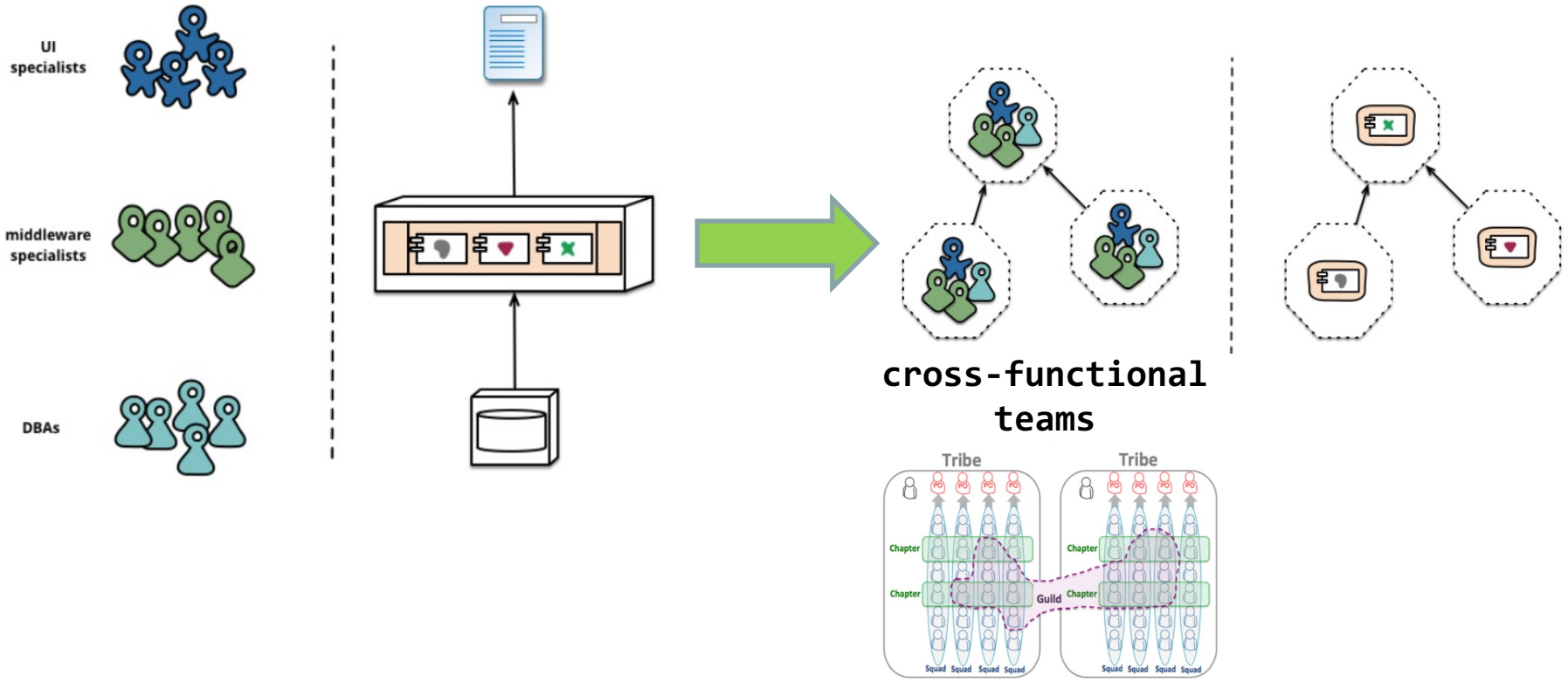
~~WS-* standards~~

Essence of microservices

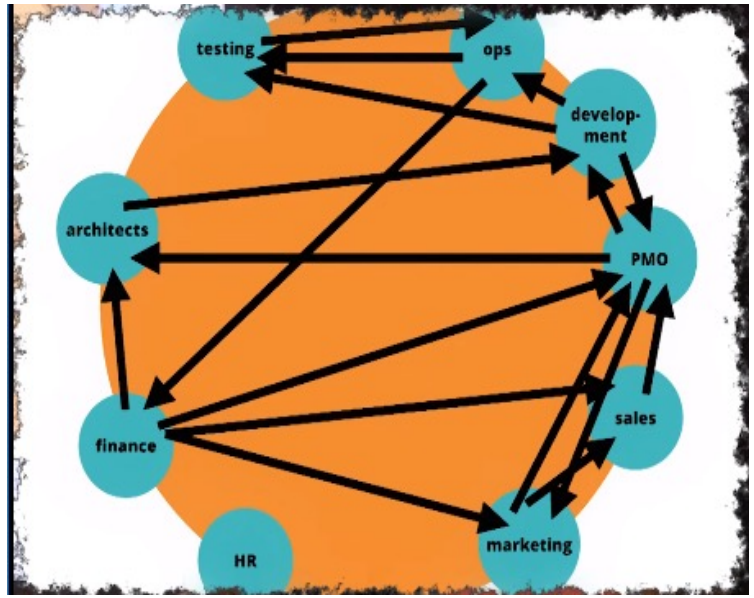
2. Organize services around business capabilities

"Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations"

M. Conway, 1968



Impact of organizational design on system architectures



Chord diagram of communication between functional silos

Each chord represents a delay in your process, due to crossing team boundaries

London’s Royal Free Hospital

London’s Royal Free Hospital - Infectious diseases team

- Cross-functional team of 33 people
- 12 clinicians
- 21 non-clinicians (nurses, pharmacists, physioherapists, …)
- “They all have to be involved”
- Eliminating delays due to inter-team communications

Coupling measures number of inter-component relationships

Low coupling → independent services, independent updates

Cohesion measures number of intra-component relationships

High cohesion → less inter-service communication overhead

How to decompose system into a set of microservices?

- Not too many (low cohesion -> communication overhead)
- Not too few (high coupling -> less independency for updates/deployment/..)

Difficult, tips:

Balance fine-grain functionality and system performance

Follow the “common closure principle”

(elements likely to be changed at the same time should stay in same service)

~~Micro~~: size doesn’t matter, really

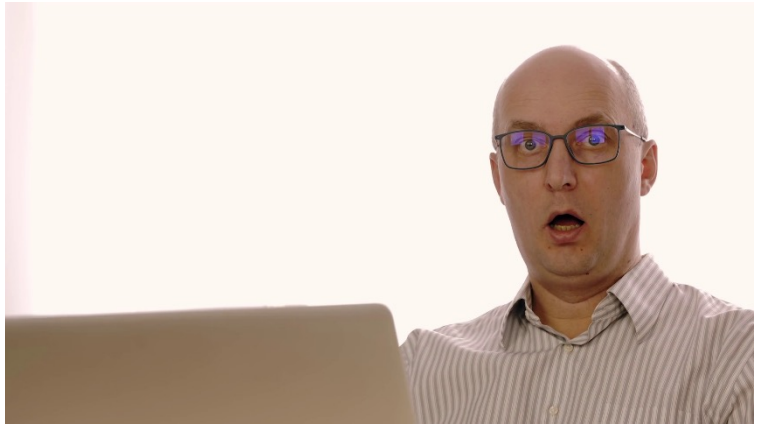
“The size of a microservice is the size
of the team that is building it”

Team can be fed with two large pizzas (8-10 people)

Essence of microservices

3. Decentralize data management 

- let each service manage its own database



- *eventual consistency* and *compensations* instead of distributed transactions

(cost of fixing mistakes vs. loosing business)

CAP theorem

In presence of a network Partition, you cannot have both Availability and Consistency.

In distributed systems we must trade-off data consistency and performance
→
Microservices must be designed to tolerate some degree of data inconsistency

Two types of inconsistency have to be managed:

1. Dependent data inconsistency
   - Actions/failures of one service can cause data managed by another service to become inconsistent
2. Replica inconsistency
   - Several replicas of the same service may be executing concurrently
   - Each updates its own db copy
   - → Need to make these dbs “eventually consistent”

Netflix approach

To replicate data in n nodes:

- «write to the ones you can get to, then fix it up afterwards»
- use quorum: e.g., (n/2 + 1) of the replicas must respond

Essence of microservices

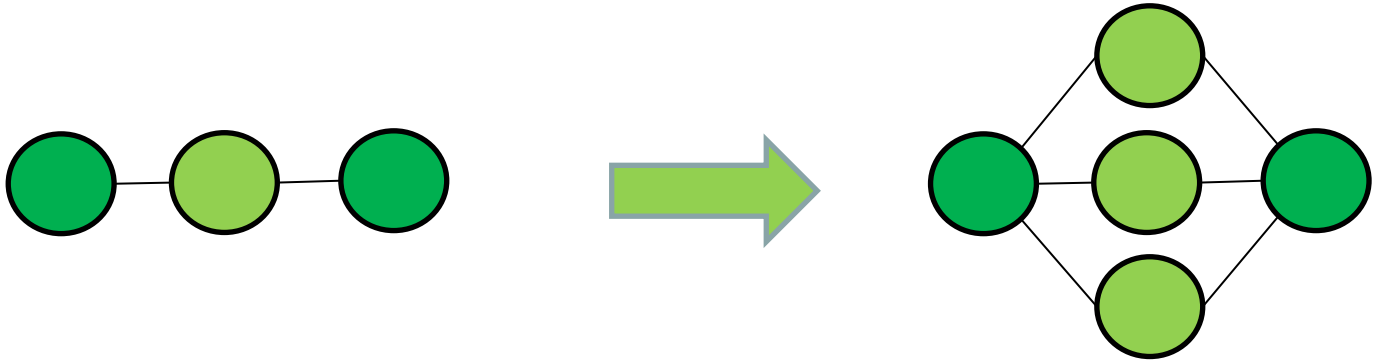
4. Independently deployable services

Objective: Be able to make a change to a single service and release it into production, without having to deploy other services

Pivotal to update/extend/restart/…

Essence of microservices

5. Horizontally scalable services 



vertical
scaling

horizontal
scaling

Add more power to «machine»

Add more «machines»

Essence of microservices

6. Fault resilient services

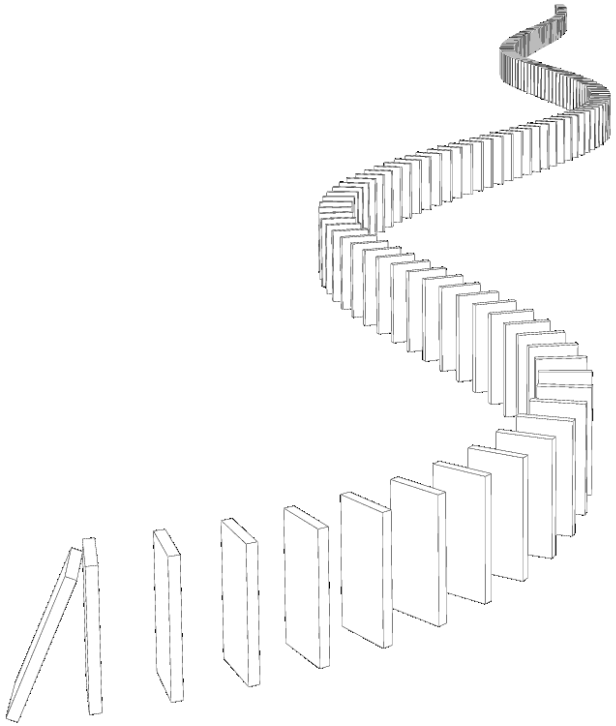
Any distributed system will experience failures - no doubt

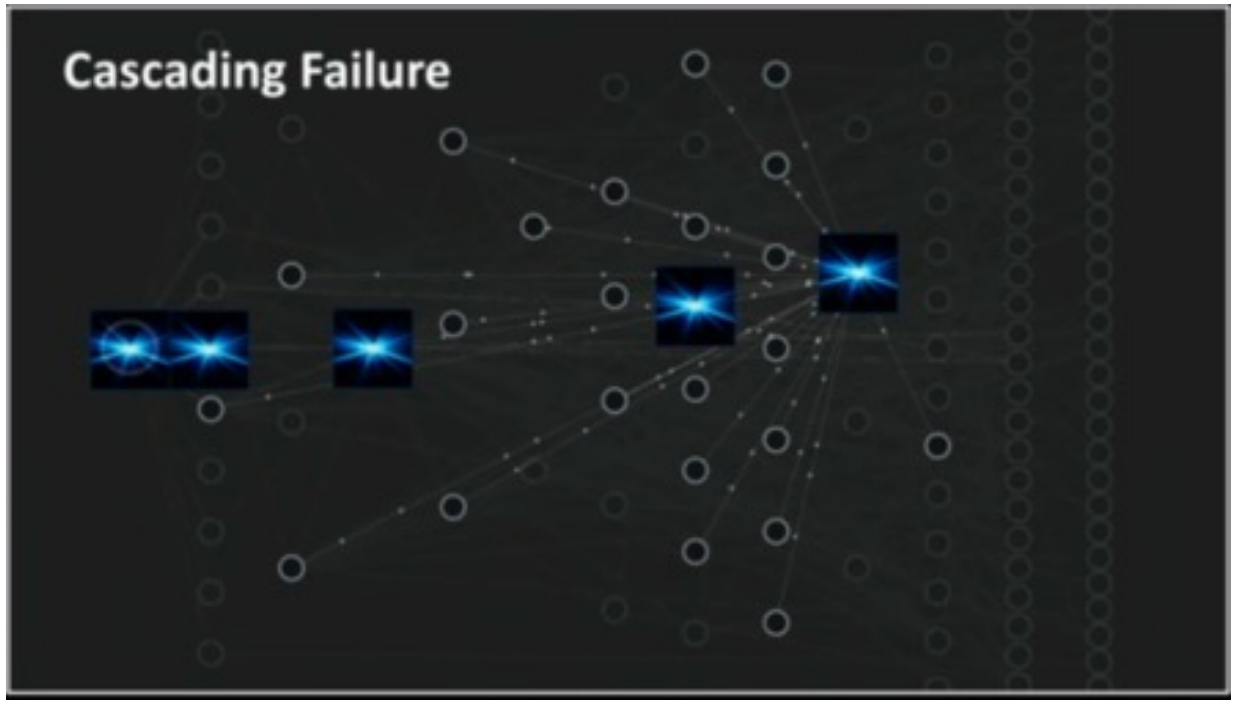
To offer a functionality F, a service synchronously invokes two other mutually independent services, each featuring 90% uptime.
We can predict that F will be probably NOT available for around ___ in a month.

- 6 minutes
- 6 hours
- 6 days

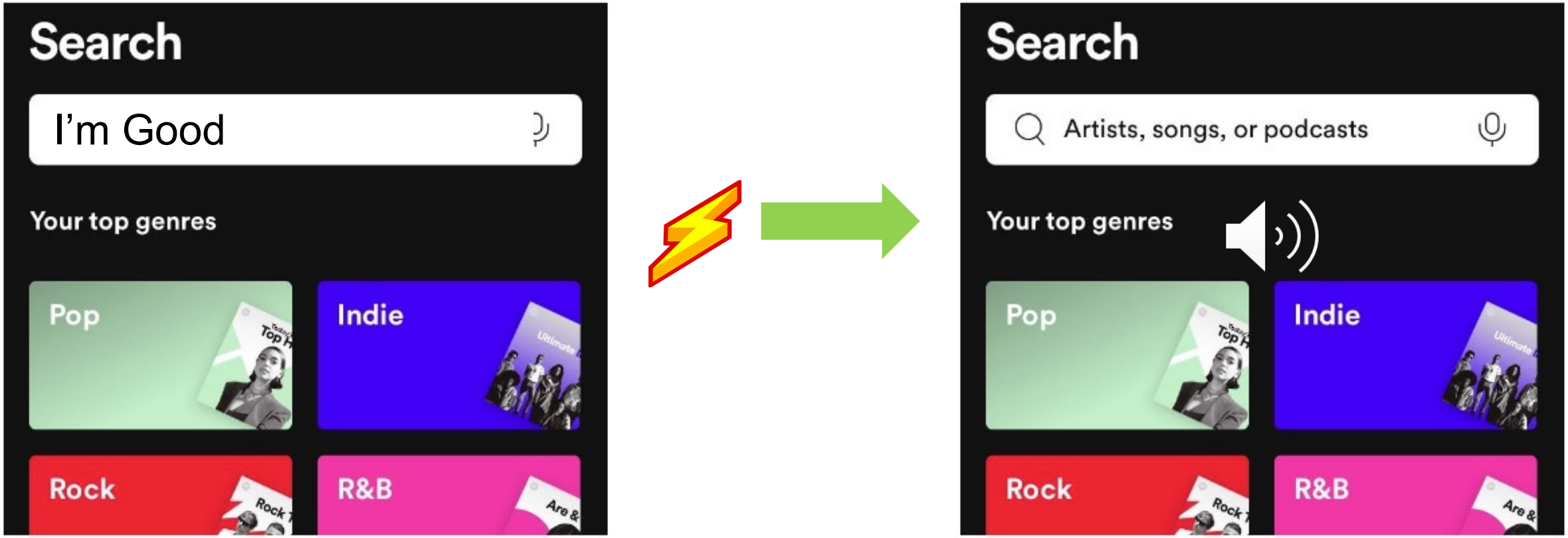
A: $p_{na} = 1 - (0.9 \times 0.9) = 0.19$
6 days!

Cascading failures may occur



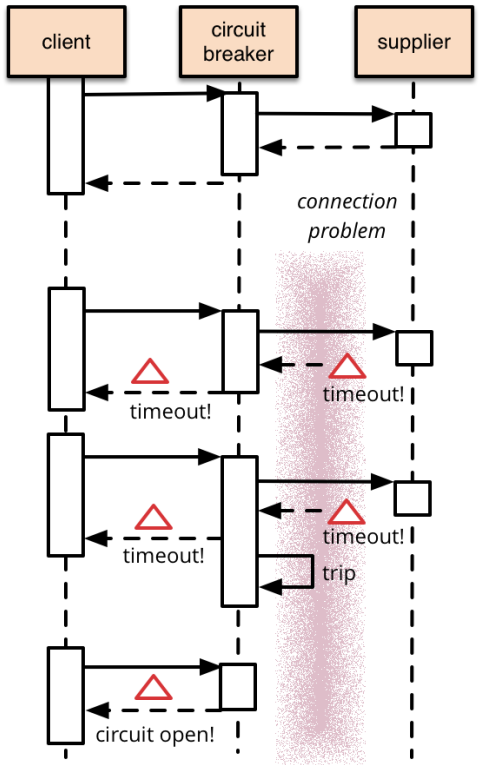


Application should tolerate failures



Isolate failures

Example: Circuit breaker



A service client invokes a remote service via a circuit breaker object that functions in a similar fashion to an electrical circuit breaker.

When the number of consecutive failures crosses a threshold, the circuit breaker trips, and for the duration of a timeout period all attempts to invoke the remote service will fail immediately.

After the timeout expires the circuit breaker allows a limited number of test requests to pass through. If those requests succeed the circuit breaker resumes normal operation. Otherwise, the timeout period begins again.

Test (bravely)

Chaos Monkey randomly terminates VM instances and containers that run inside your production environment

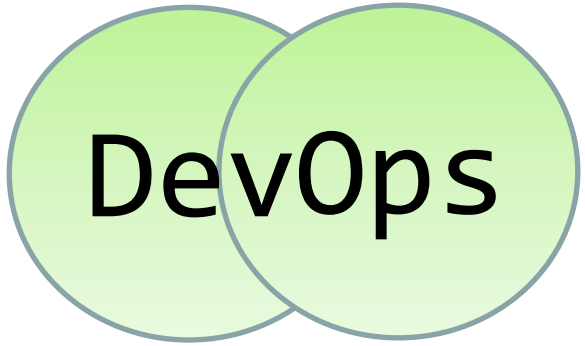
Essence of microservices (summary)

Develop applications as 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

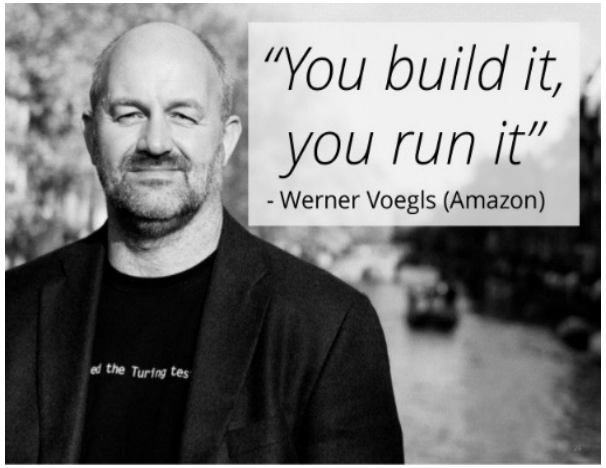
[Some of these ideas date back to Unix design principles]

[Service-orientation «done right»?]



culture and tools

Same team responsible for service development, deployment and managements



Being woken up at 3am by your pager is certainly a powerful incentive to focus on quality when writing your code

Tools

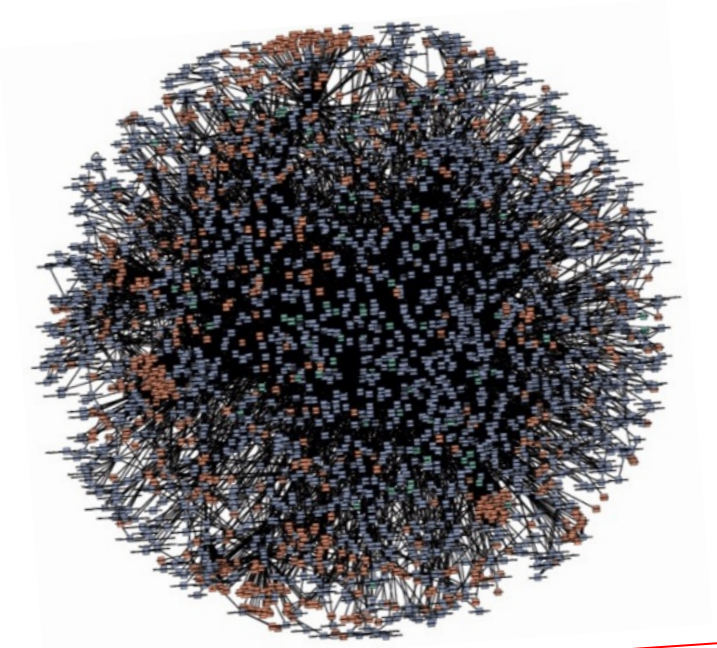
- VCS (Version Control Systems)
- CI/CD (Continuous Integration/Continuos Deployment)
- IaC (Infrastructure As Code)

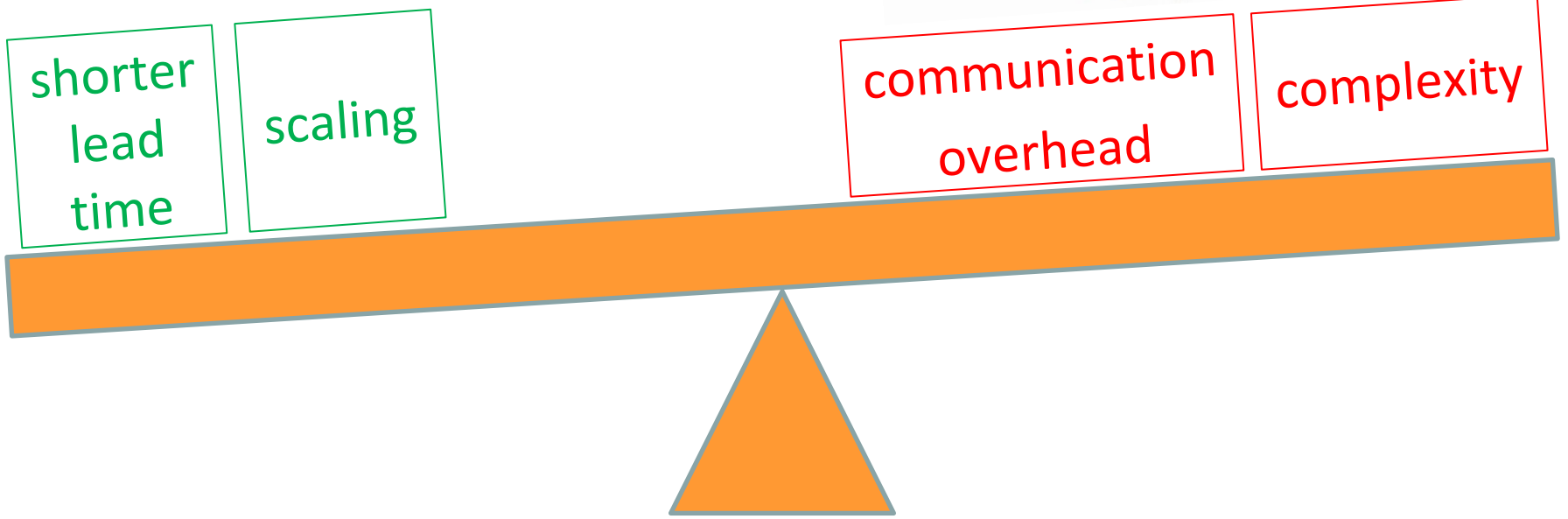

• Why microservices?
• Essence of microservices
• Concluding remarks

Concluding remarks

- Many pros, including
  - shorter lead time
  - effective scaling
- Cons
  - communication overhead
  - complexity (testing and monitoring included)
  - “wrong cuts”
  - “avoiding data duplication as much as possible while keeping microservices in isolation is one of the biggest challenges”
- “A poor team will always create a poor system”

Concluding remarks

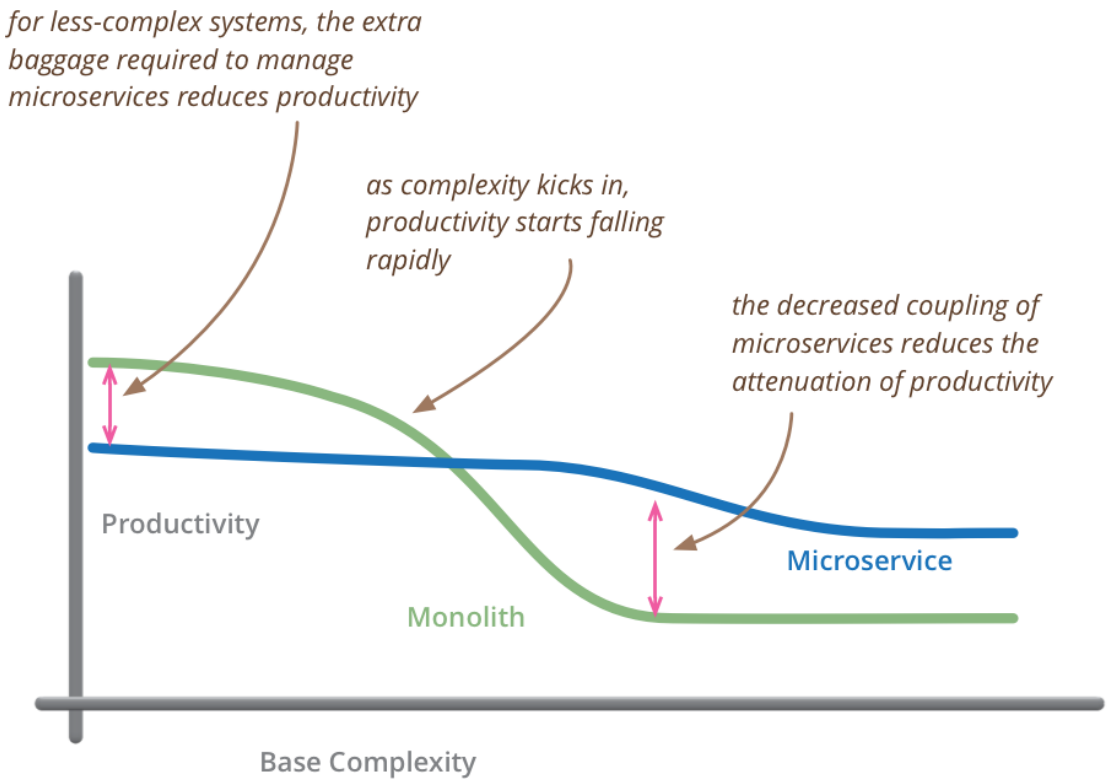




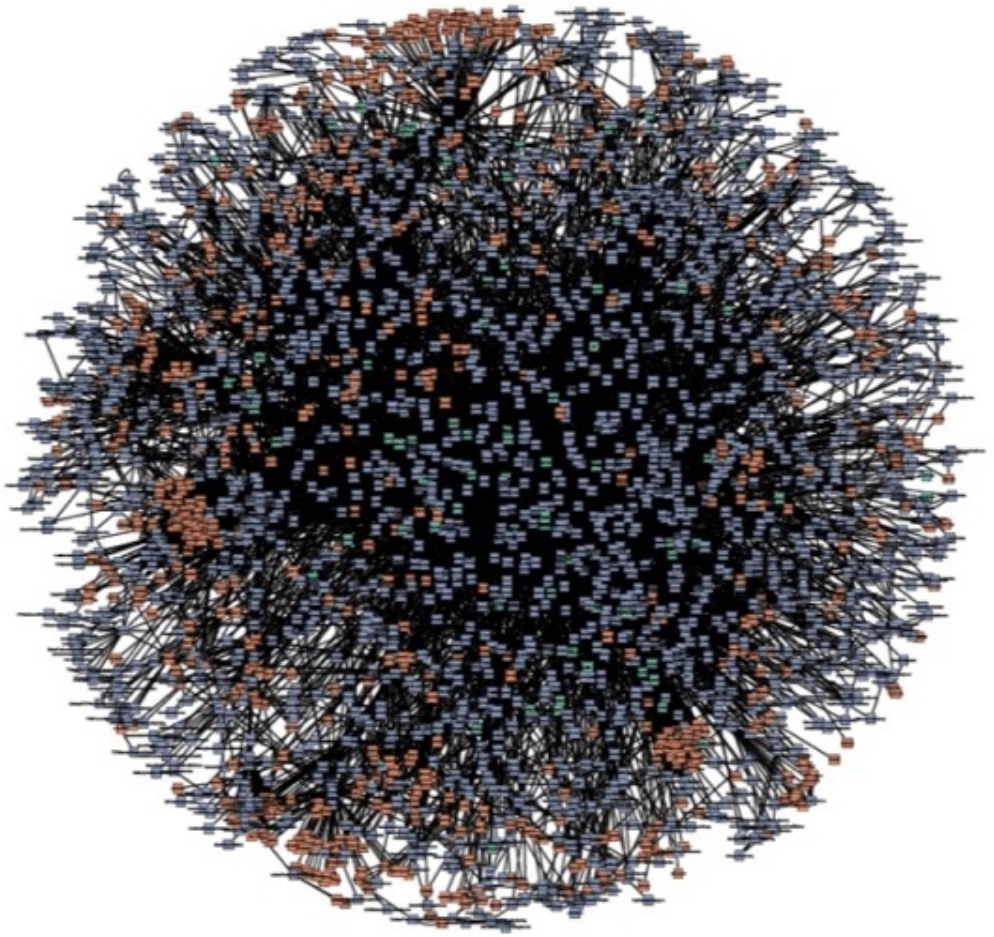
Concluding remarks

> Don't even consider microservices unless you have a system that's too complex to manage as a monolith

[M. Fowler]



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



amazon Google NETFLIX Spotify twitter LinkedIn ebay zalando GROUPON Uber ...

e.g. (source www.businessofapps.com)

- 422 million people use Spotify once a month, 182 million are subscribers
- In 2022, Netflix had 222 million subscribers worldwide

Can I play with microservices?