Microservices

Antonio Brogi

Department of Computer Science
University of Pisa

Microservices

- Microservices
  - microservices
  - microservices
- Microservices
  - microservices
  - microservices
  - microservices
- Microservices
  - microservices

Buzzword?

or reality?

amazon

NETFLIX

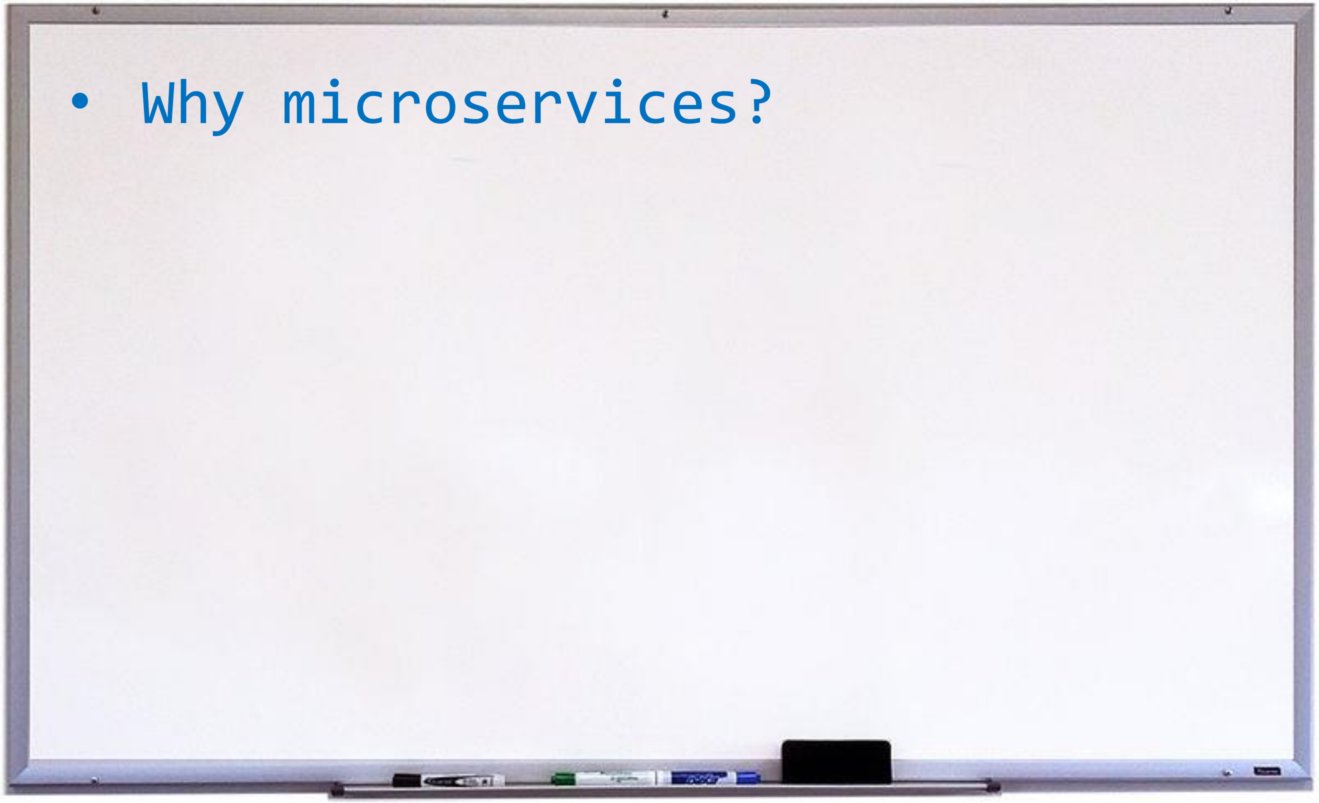
Spotify

ebay

Google

twitter

...



- Why microservices?

Why microservices?

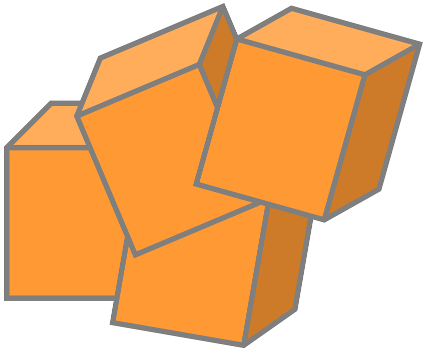
(1) Shorten lead time for new features and updates

→ accelerate rebuild and redeployment

→ reduce chords across functional silos

Why microservices?

(2) Need to scale, effectively



e.g., Spotify figures:

- 75M monthly active users
- average user session 23 min, white/pink noise all night
- 2B user-generated playlists, 75M playlists created by Spotify

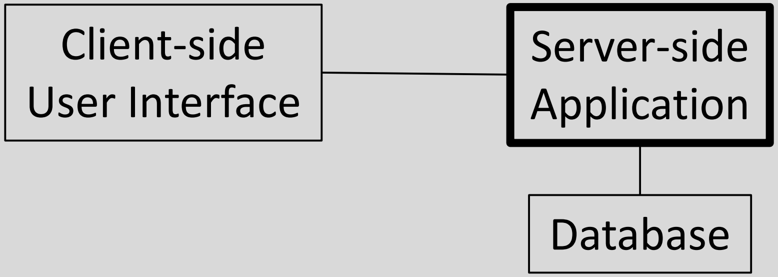
OK but … what are microservices?

James

- 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 databas
- 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 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~~

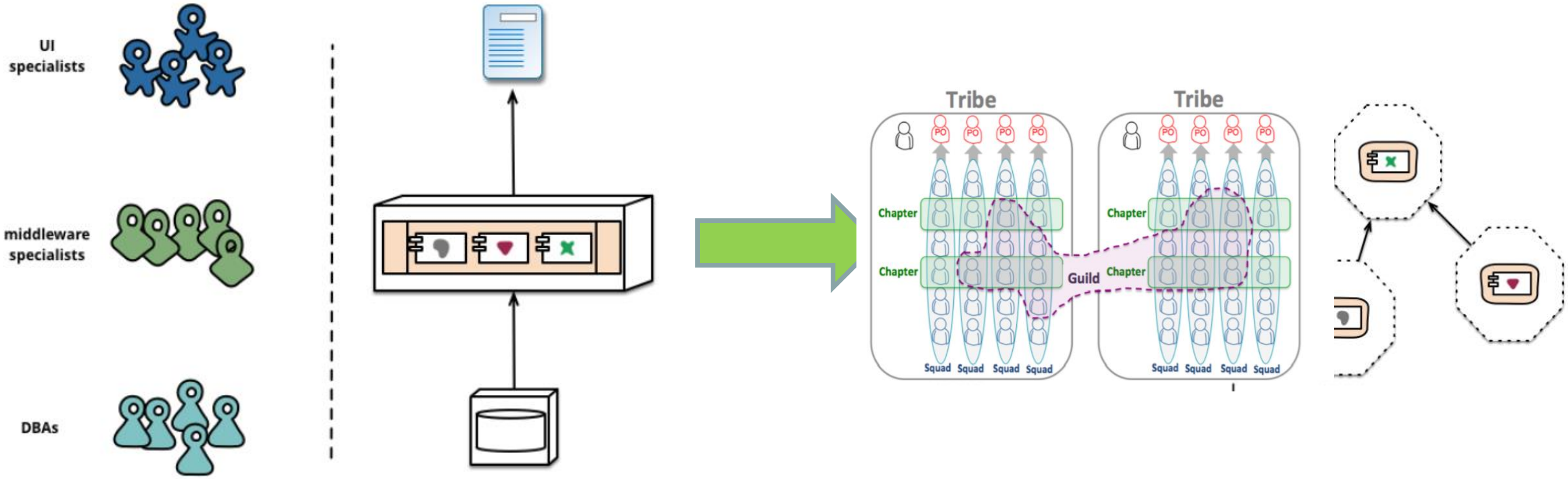
~~Micro~~: size doesn’t matter, really

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

Essence of microservices

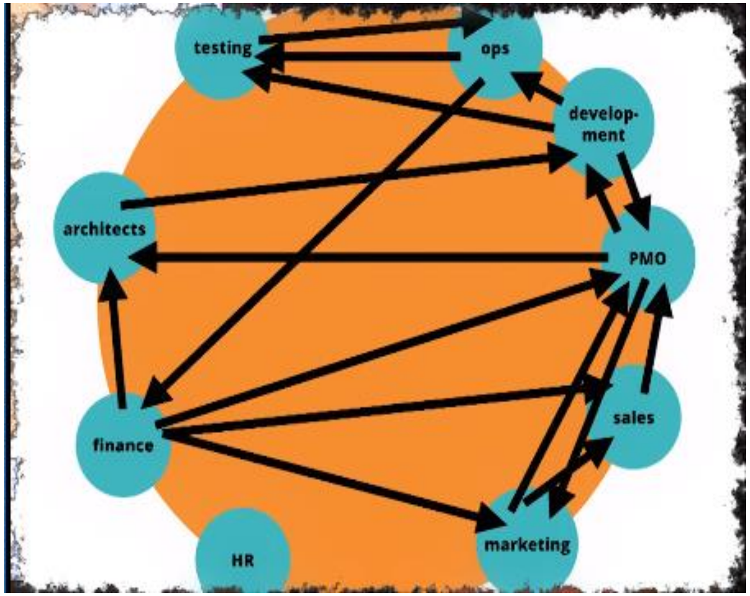
2. Organize services around business capabilities

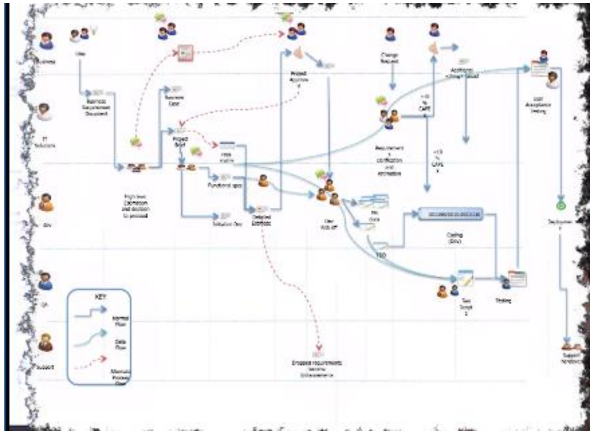
> "Organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations"
>
> *M. Conway, 1968*



cross-functional teams

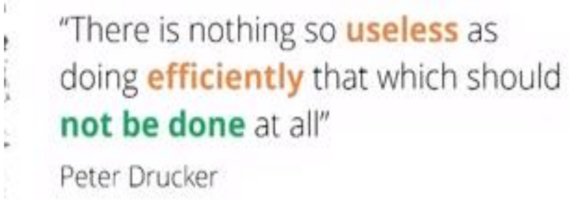
Impact of organizational design on system architectures





Chord diagram of communication between functional silos

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



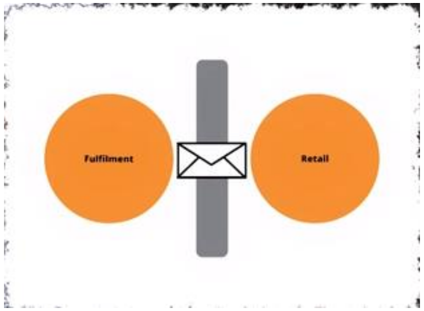
"There is nothing so **useless** as doing **efficiently** that which should **not be done** at all"

Peter Drucker

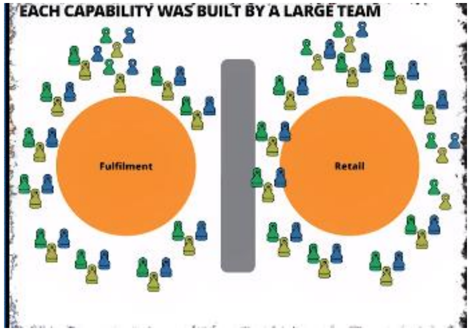
J. Lewis. How I Finally Stopped Worrying and Learnt to Love Conway's Law. GOTO 2015.

Example 1: Ecommerce site (selling tickets)

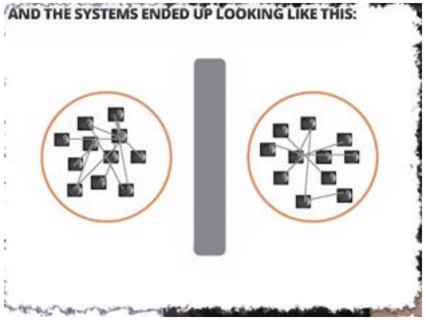
Big project teams in London (≈ 100 people) and in India (≈ 200 people)
India: building retail part (selling tickets)
London: fulfillment (get tickets to people)



Big message bus in the middle

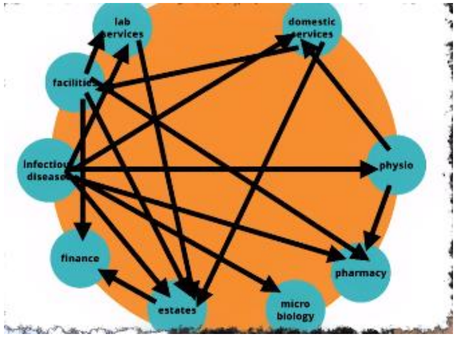


Functional silos, short lived project teams

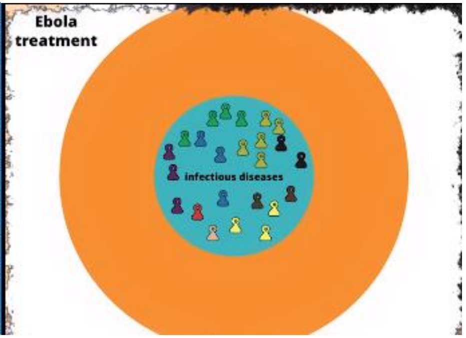


Tightly coupled services, at both sides
System had to be end-to-end tested (6 weeks) and deployed all together: distributed monolith

Example 2: London’s Royal Free Hospital



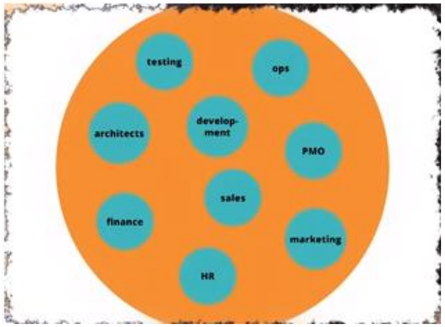
Hospital chord diagram (general)

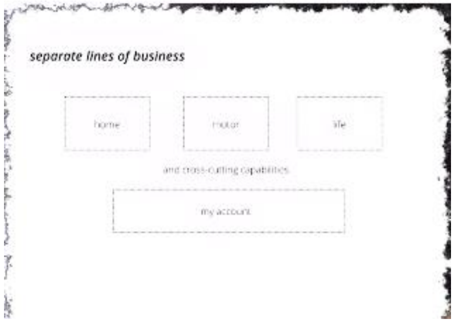


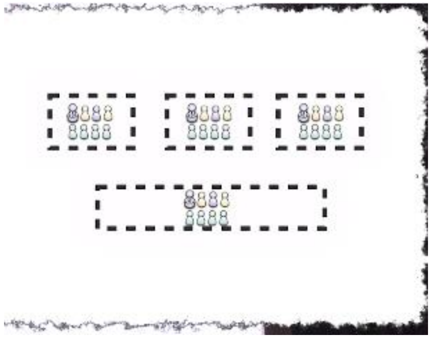
London’s Royal Free Hospital
Infectious deseases

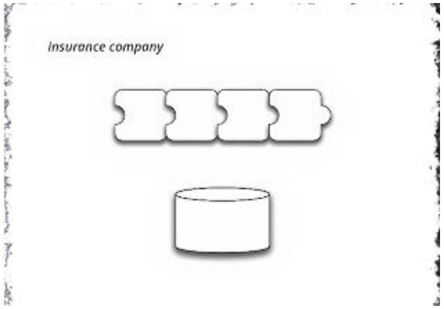
- Cross-functional team of 33 people
- 12 clinicians
- 21 non-clinicians (nurses, pharmacists, physioherapists, …)
- “They all have to be involved”
- Eliminating delays due to inter-team communications

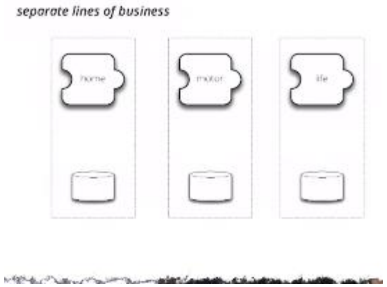
Example 3: Insurance company











Essence of microservices

3. Decentralize data management

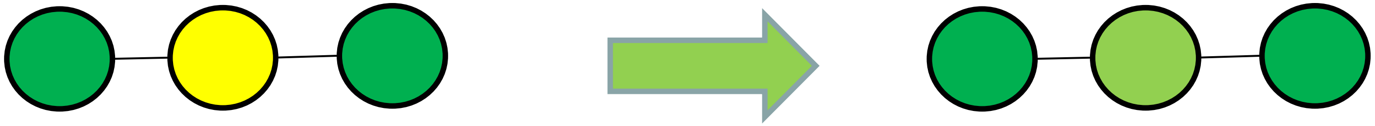
- let each service manage its own database

- eventual consistency and compensations instead of distributed transactions

(cost of fixing mistakes vs. loosing business)

Essence of microservices

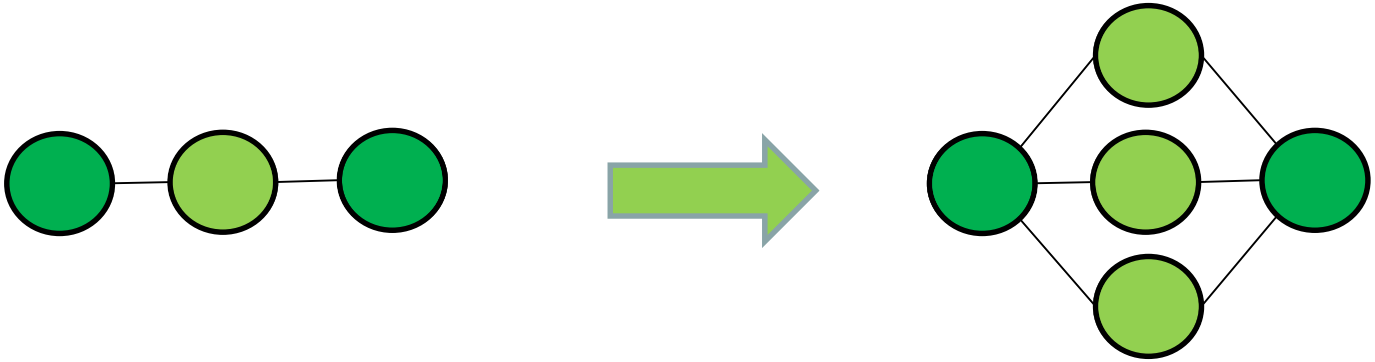
4. Independently deployable services



Pivotal to update/extend/restart/…

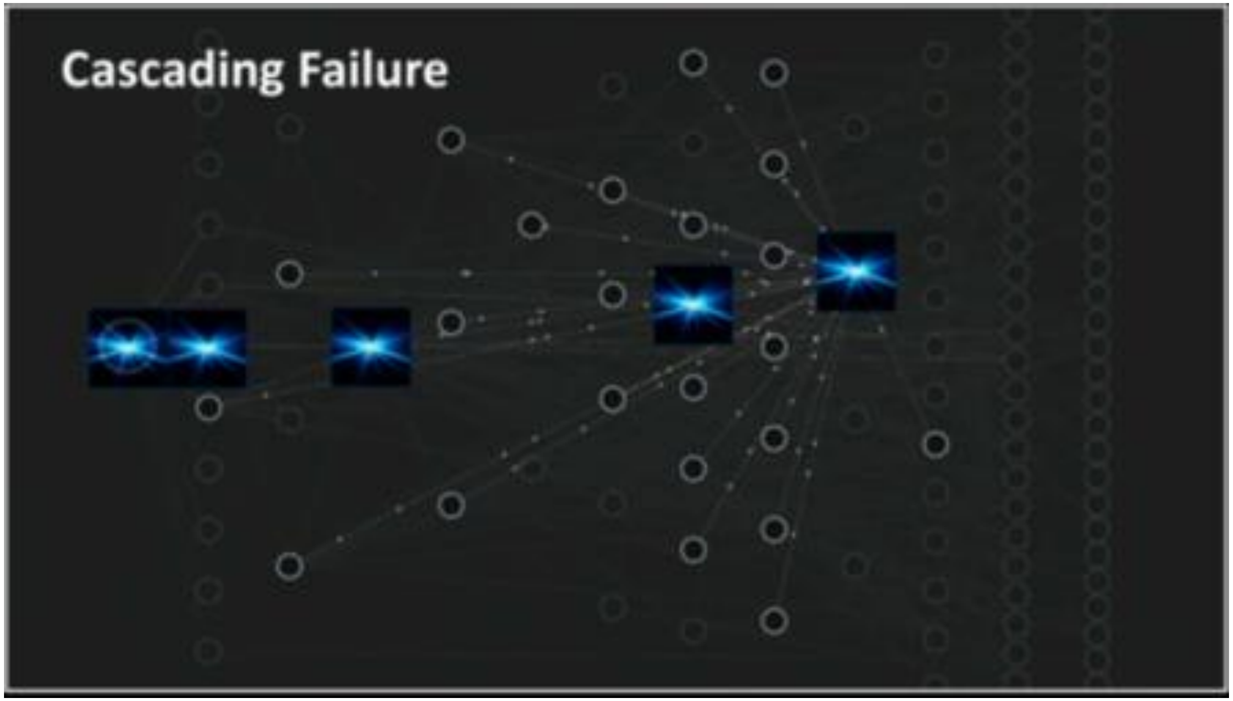
Essence of microservices

5. Horizontally scalable services



Essence of microservices

6. Fault resilient services



- Applications need to be designed so that they can tolerate failure of services

- Any service call could fail due to unavailability of supplier, client has to respond to this as gracefully as possible

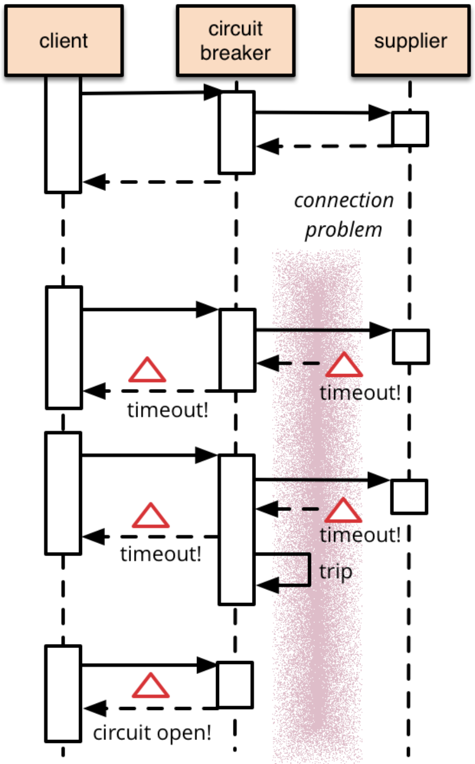
- Netflix API (data of 2012)
  - received more than 1 billion incoming calls per day
  - several billion outgoing calls (averaging a ratio of 1:6) to dozens of underlying subsystems with peaks of over 100k dependency requests per second
  - all this across thousands of cloud instances
  - intermittent failure guaranteed with this many variables, even if every dependency itself has excellent availability and uptime

synchronous calls between services induce multiplicative effect of downtime

30 dependencies each with 99.99% uptime → 2+ hours downtime/month

$99.99\%^{30}$ x 24 x 30 = 99.7% x 24 x 30 = 2+

Design for failure



Test (bravely)

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

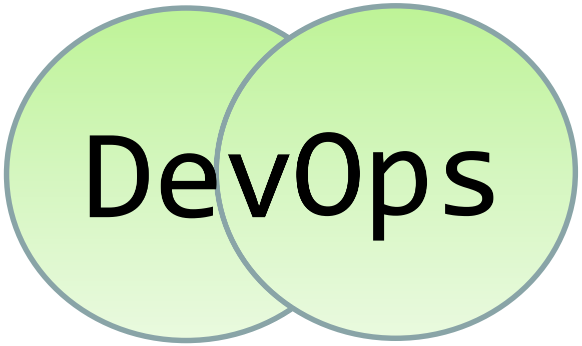
Essence of microservices (summary)

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

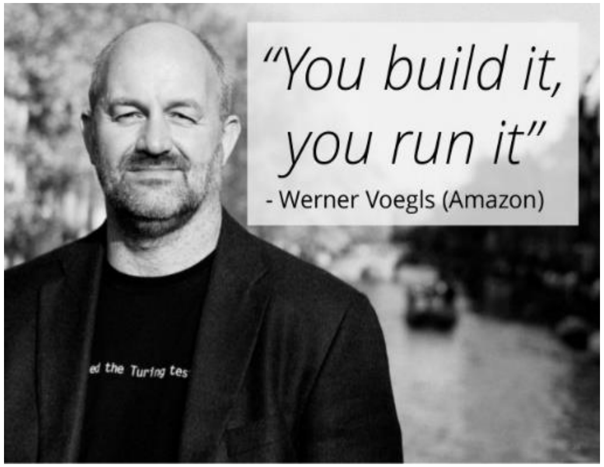
[Some of these ideas date back to Unix design principles]

[Service-orientation «done right»?]



DevOps

~~projects~~ products



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

VCS (e.g., Git & GitHub)
CI&CD (e.g., Jenkins)
IaC (e.g., Puppet, Chef)
APM (e.g., NewRelic)

OK, got it.

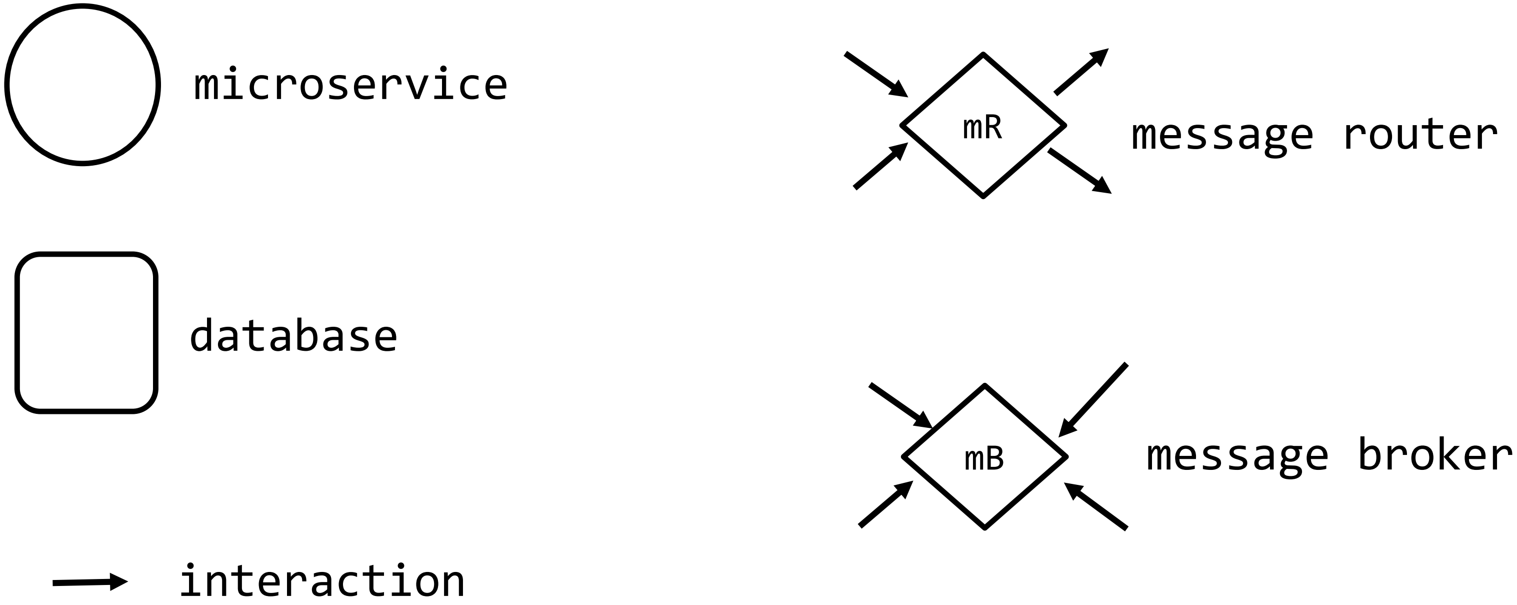
Now, does my application respect the «microservices principles»?

If not, how should I refactor it?

- Why microservices?
- Essence of microservices
- Refactoring microservice-based architectures

A simple modelling of microservice architectures can be fruitfully exploited to drive the refactoring of existing application

Simple model



microservice

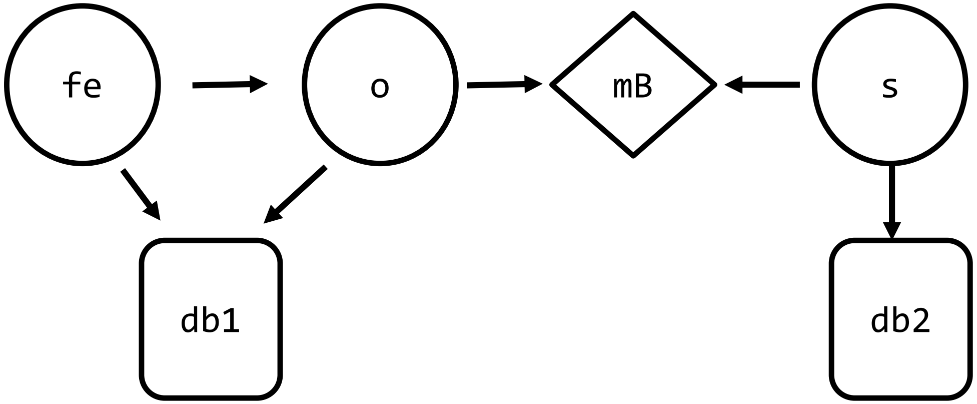
database

interaction

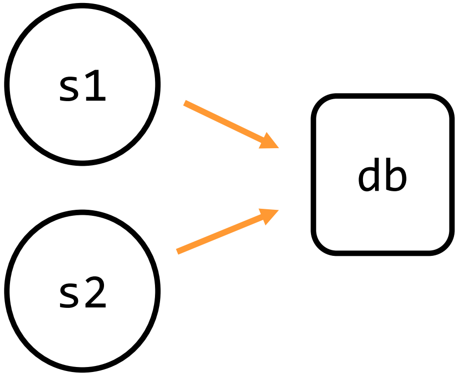
message router

message broker

Example

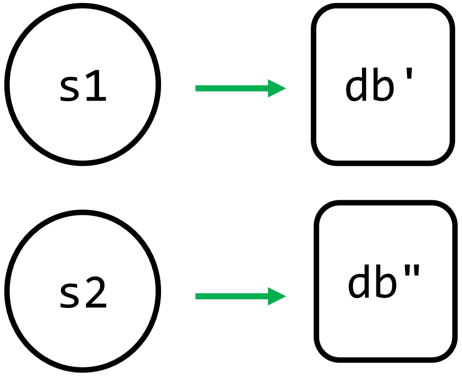


Anti-pattern



db shared by
multiple services

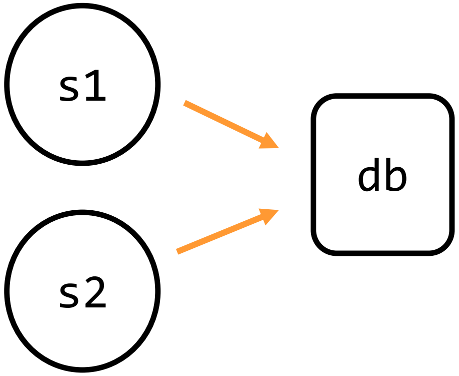
Refactoring



db split

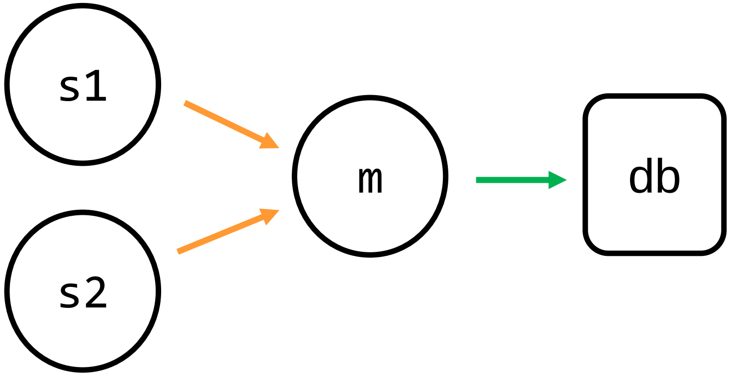
- split db
- small changes to s1,s2
- not always possible/easy to implement

Anti-pattern



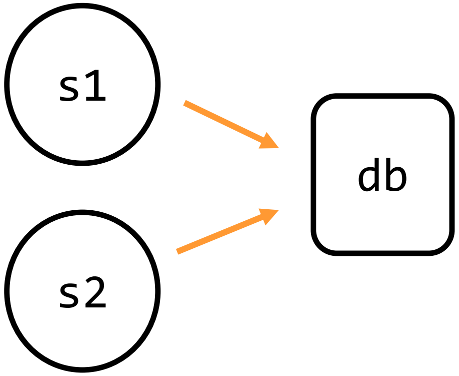
db shared by
multiple services

Refactoring



db manager introduction
- m added
- very small changes to s1,s2
(- direct inter-service
dependencies introduced)

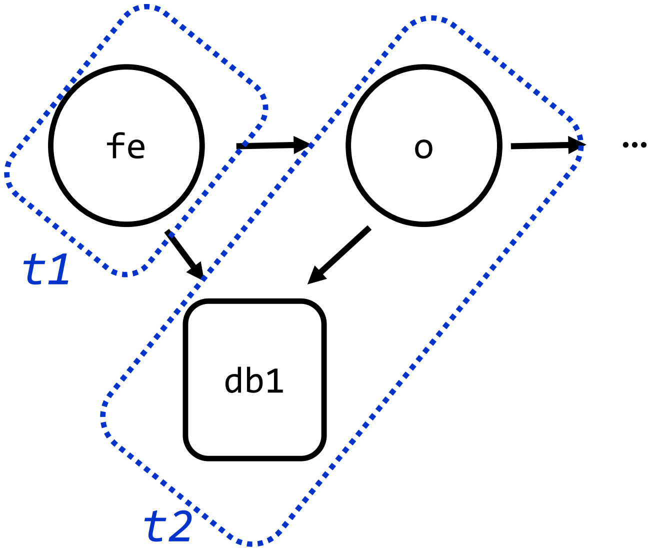
Anti-pattern



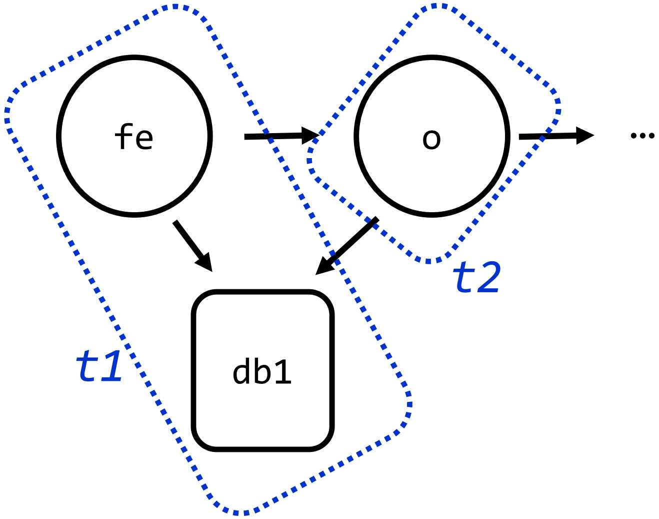
db shared by
multiple services

Refactoring

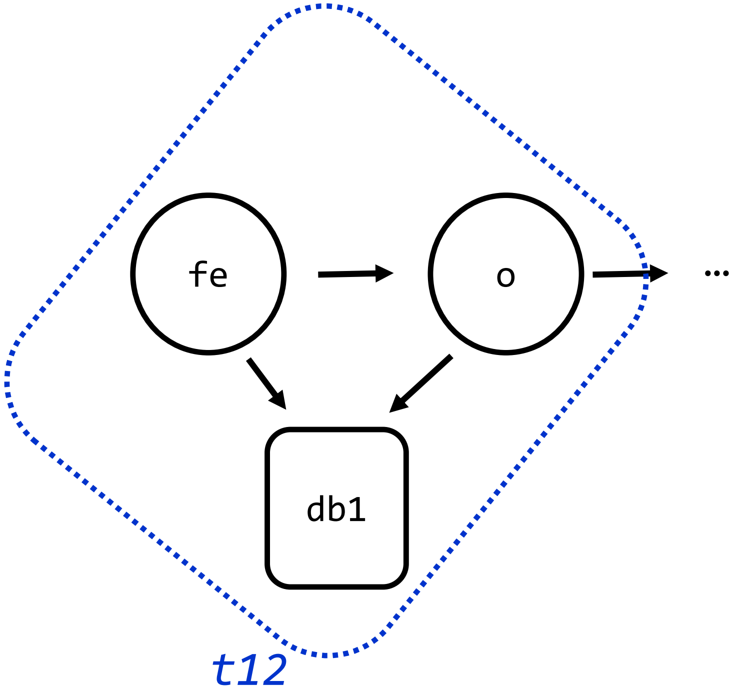
Anti-pattern



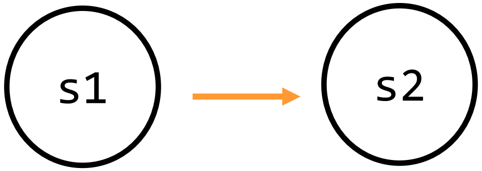
db shared by
multiple ~~services~~ **teams**



Organizational Refactoring



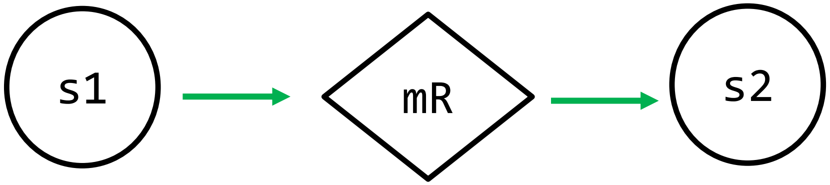
Anti-pattern



direct dependency (evil)

s2 not horizontally
scalable

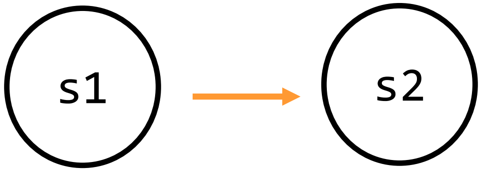
Refactoring



message router introduction

- mR added
- very small changes to s1

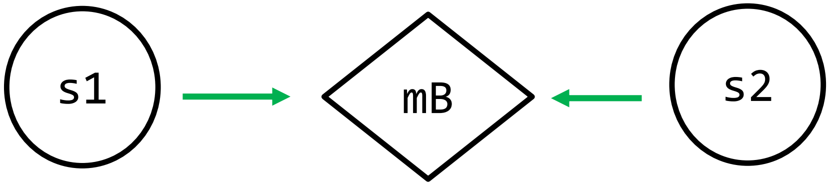
Anti-pattern



direct dependency (evil)

s2 not horizontally scalable

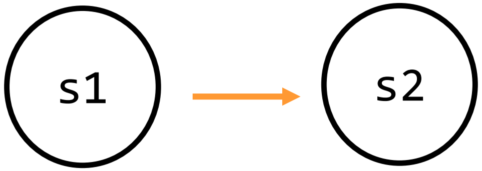
Refactoring



message broker introduction

- mB added
- very small changes to s1
- bigger changes to s2

Anti-pattern

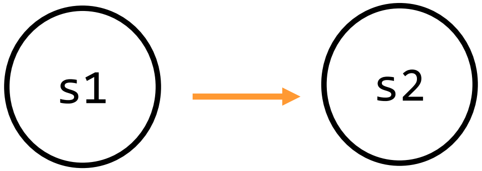


direct dependency (evil)

s2 not horizontally scalable

Refactoring

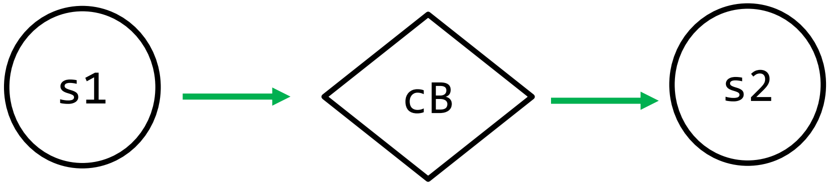
Anti-pattern



direct dependency (evil)

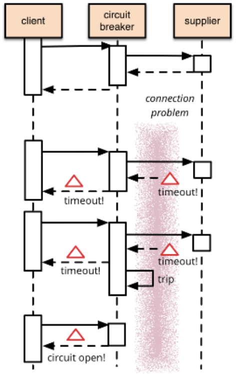
s1 not fault resilient

Refactoring



circuit breaker introduction

- cB added



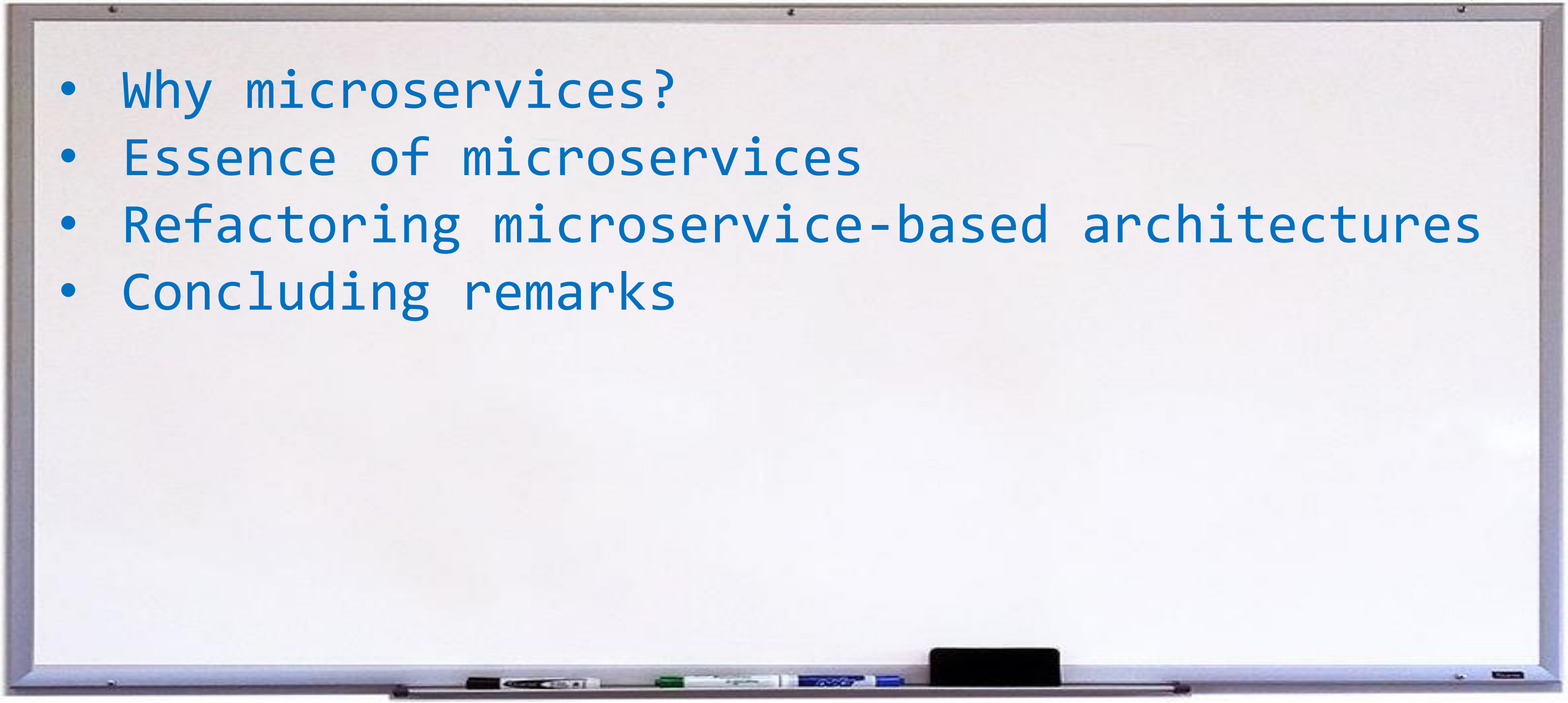
Remark

Microservice-based architectures
mapped onto
orchestrated **containers**

Orchestration does change the behaviour
of microservice based architecture!

Very interesting ☺

Where can I find tools supporting those analyses?



- Why microservices?
- Essence of microservices
- Refactoring microservice-based architectures
- Concluding remarks

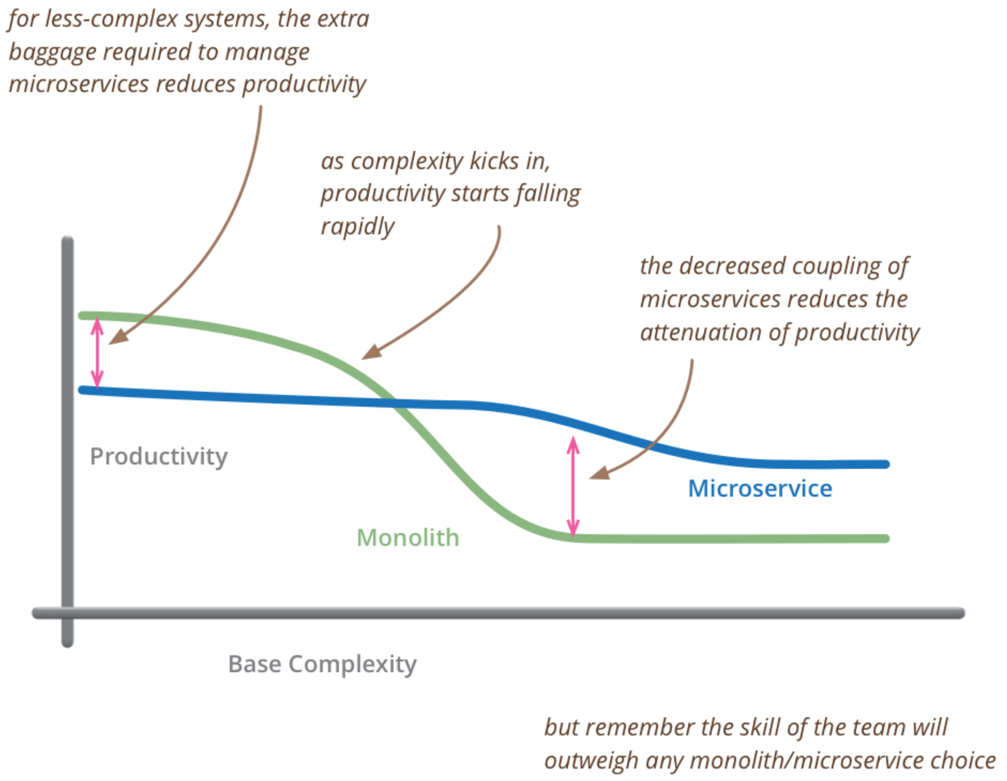
Concluding remarks

- Microservice architectural style is an important idea, worth serious consideration for enterprise applications
- Many pros, including
    - shorter lead time
    - effective scaling
- Cons
    - communication overhead
    - complexity
    - “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

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

[M. Fowler]



for less-complex systems, the extra baggage required to manage microservices reduces productivity

as complexity kicks in, productivity starts falling rapidly

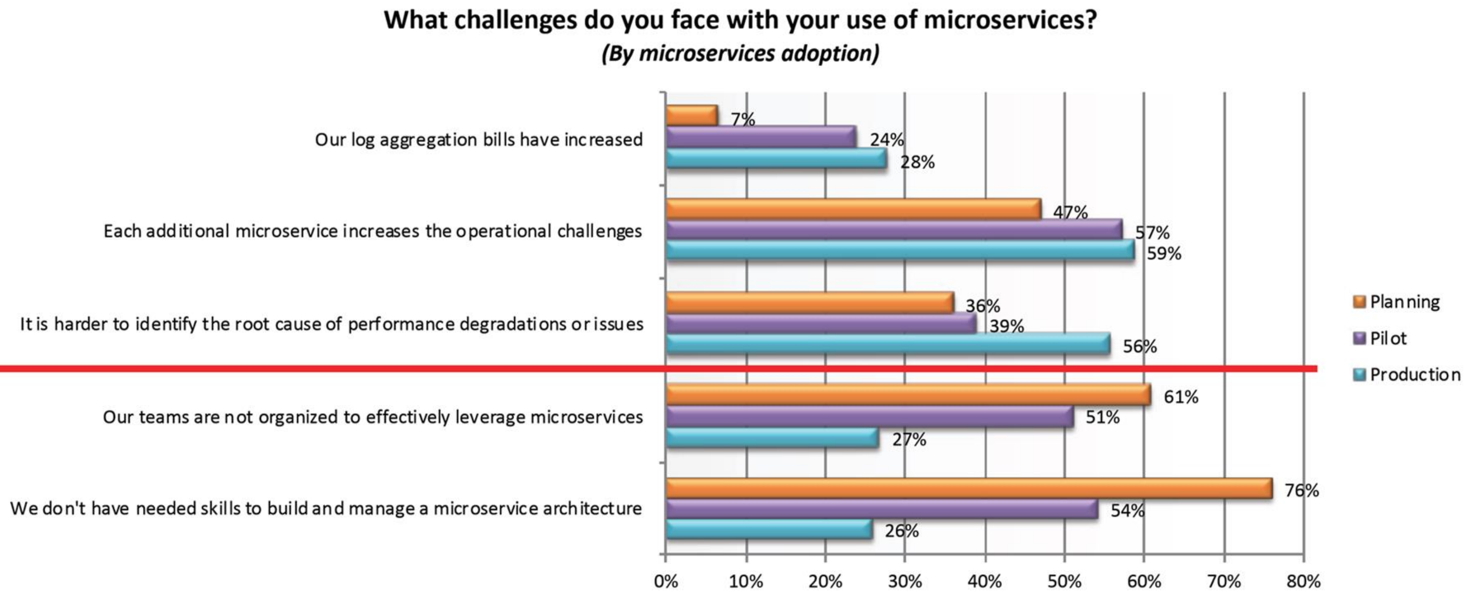
the decreased coupling of microservices reduces the attenuation of productivity

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

Data and Troubleshooting Problems Caused by Microservices

[thenewstack.io – May 2018]

Interviews to 300+ CIOs of companies with 500+ employees



What challenges do you face with your use of microservices?
(By microservices adoption)

Our log aggregation bills have increased
7%
24%
28%
Each additional microservice increases the operational challenges
47%
57%
59%
It is harder to identify the root cause of performance degradations or issues
36%
39%
56%
Our teams are not organized to effectively leverage microservices
61%
51%
27%
We don't have needed skills to build and manage a microservice architecture
76%
54%
26%
0%
10%
20%
30%
40%
50%
60%
70%
80%
Planning
Pilot
Production

Can I play with microservices?

Netflix: 80M members, 190 countries, 10s of languages, 1000s of device types

Spotify: 810 active services, 90+ squads, 600+ developers