Aalto University
School of Electrical
Engineering

Microservices and serverless: Overview

Santeri Paavolainen

10.1.2018

Overview

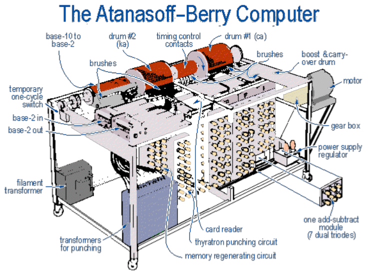
- Historical background and context
- Defining ”microservice”
- Why would one use microservices?
  - Pros and cons
- Some tools and terminology

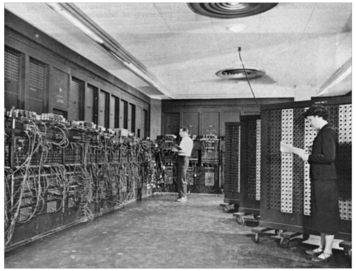
- All of this pretty broadly, will dig deeper in later lectures

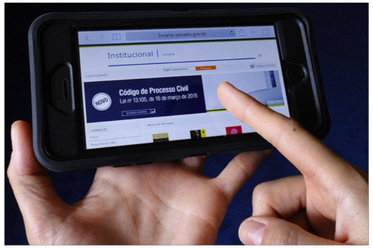
What is a microservice?

Some computing history …

Trends in computing



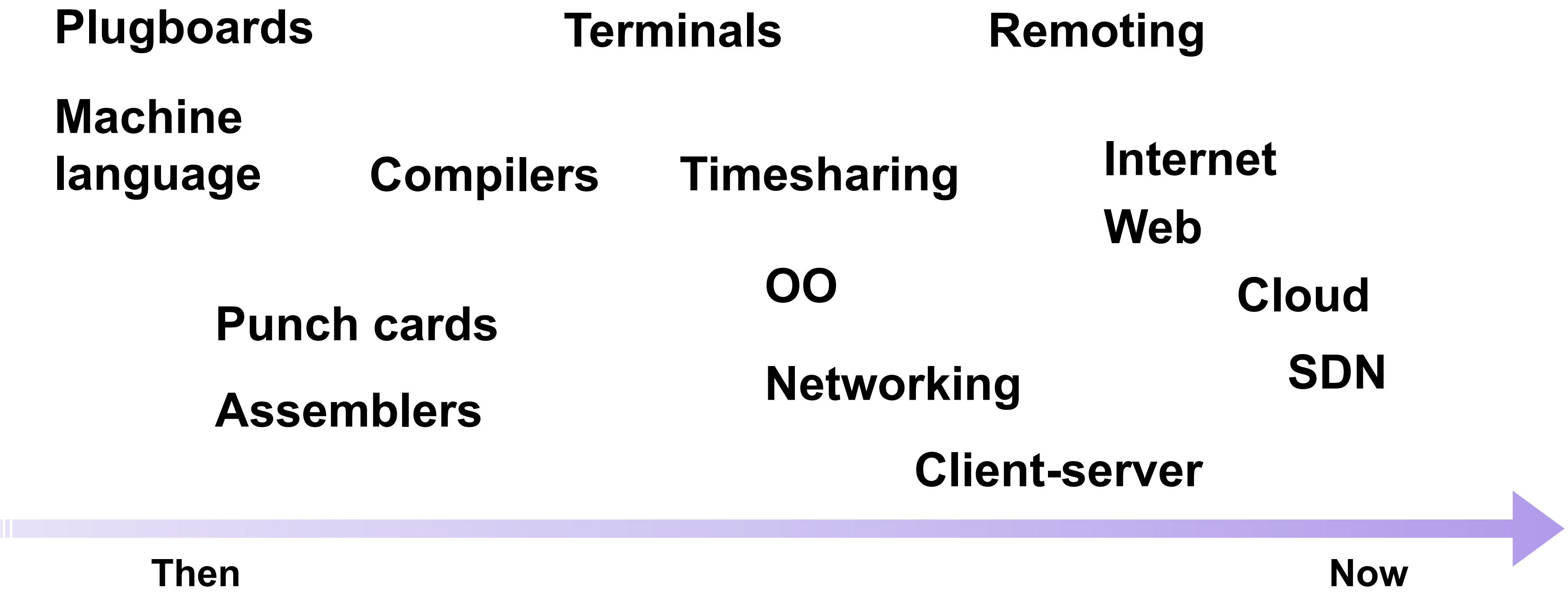




Genesis **Custom built** **Product** **Commodity**

Sources:Joi Ito (SAGE, CC BY 2.0), Arnold Reinhold (IBM 1401, CC BY-SA 3.0), Ben Franske (IBM S/360, CC BY 2.5), Veradrive (IBM PC/XT, CC BY 4.0), Senado Feredal (Smartphone, CC BY 2.0), Google (Google data center)

Trends in computing



Plugboards

Machine language

Punch cards

Assemblers

Compilers

Terminals

Timesharing

OO

Networking

Client-server

Remoting

Internet

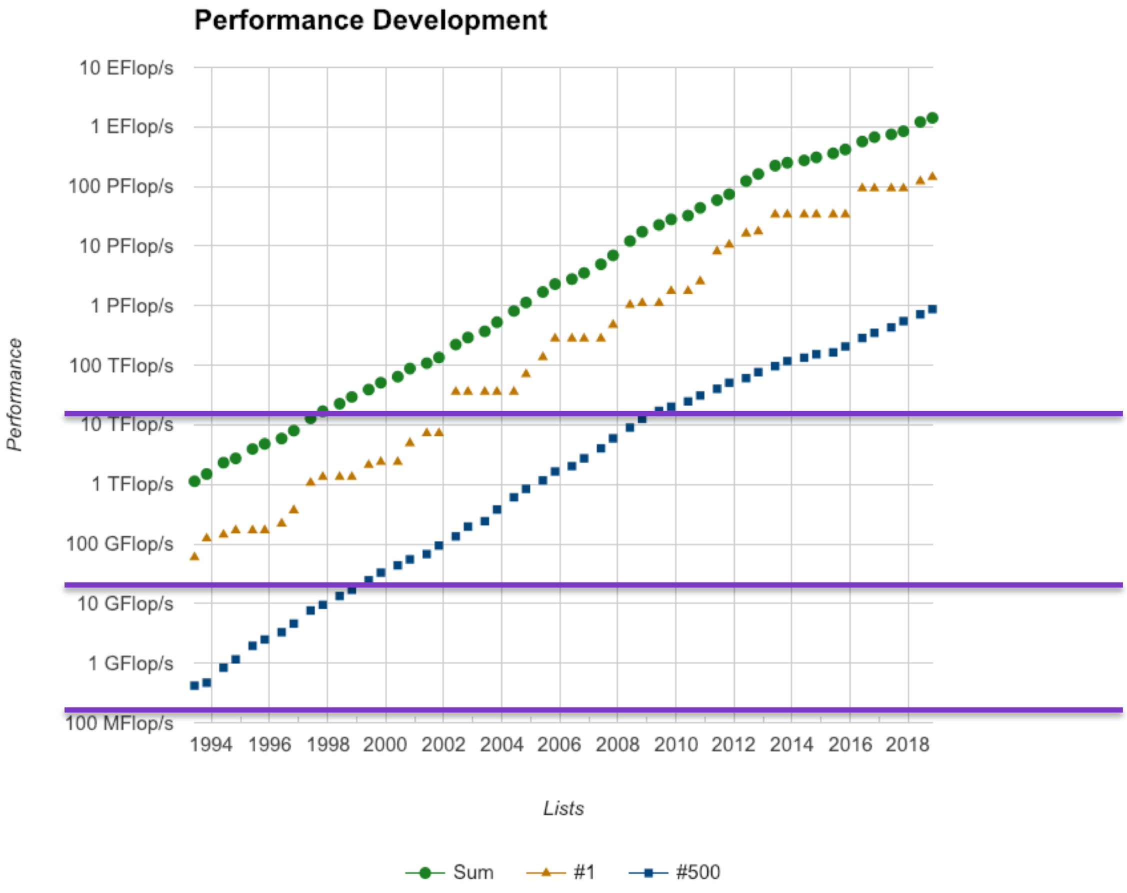
Web

Cloud

SDN

Then

Now



Nvidia Titan Xp
12 TFlop/s

Raspberry Pi
28 GFlop/s

Cray 1
160 MFlop/s

Source: top500.org

Scarcity	Abundance
Few users	World population
Little data	Big data

Then

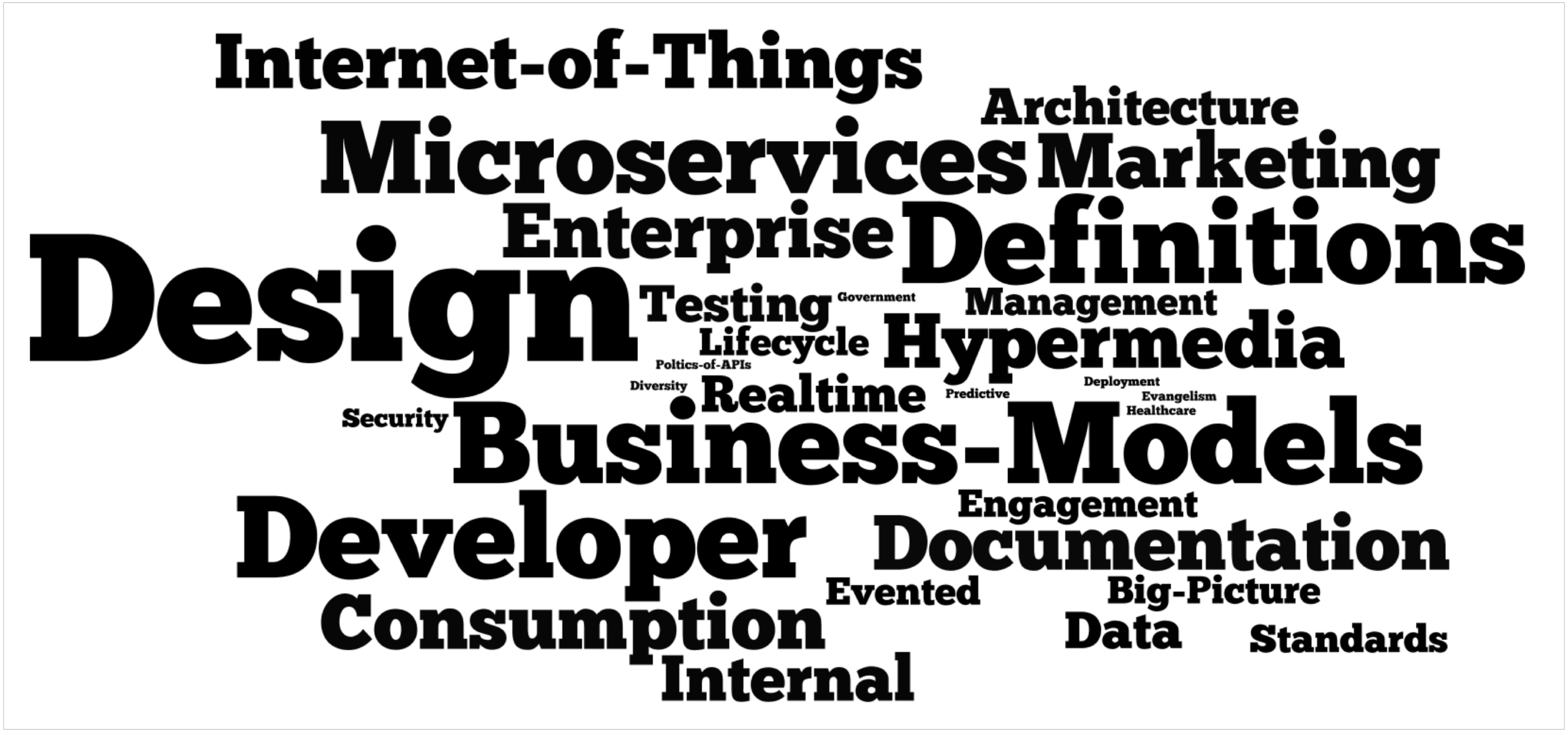
Now

Then

Now

Source: pngimg.com (CC BY-NC 4.0)

What is a microservice?



Source: Kin Lane (CC BY 3.0)

Defining microservices

- **“Like a service, just smaller”**
  - Not very useful …
- **Architectural design model**
  - Fine-grained separation of concerns
- **Implementation patterns for heterogenous systems**
  - How to deploy and manage hundreds of difference services?
- **Organizing human resources**
  - Microservices as an organizational e.g. management tool

Microservices as an architectural design model

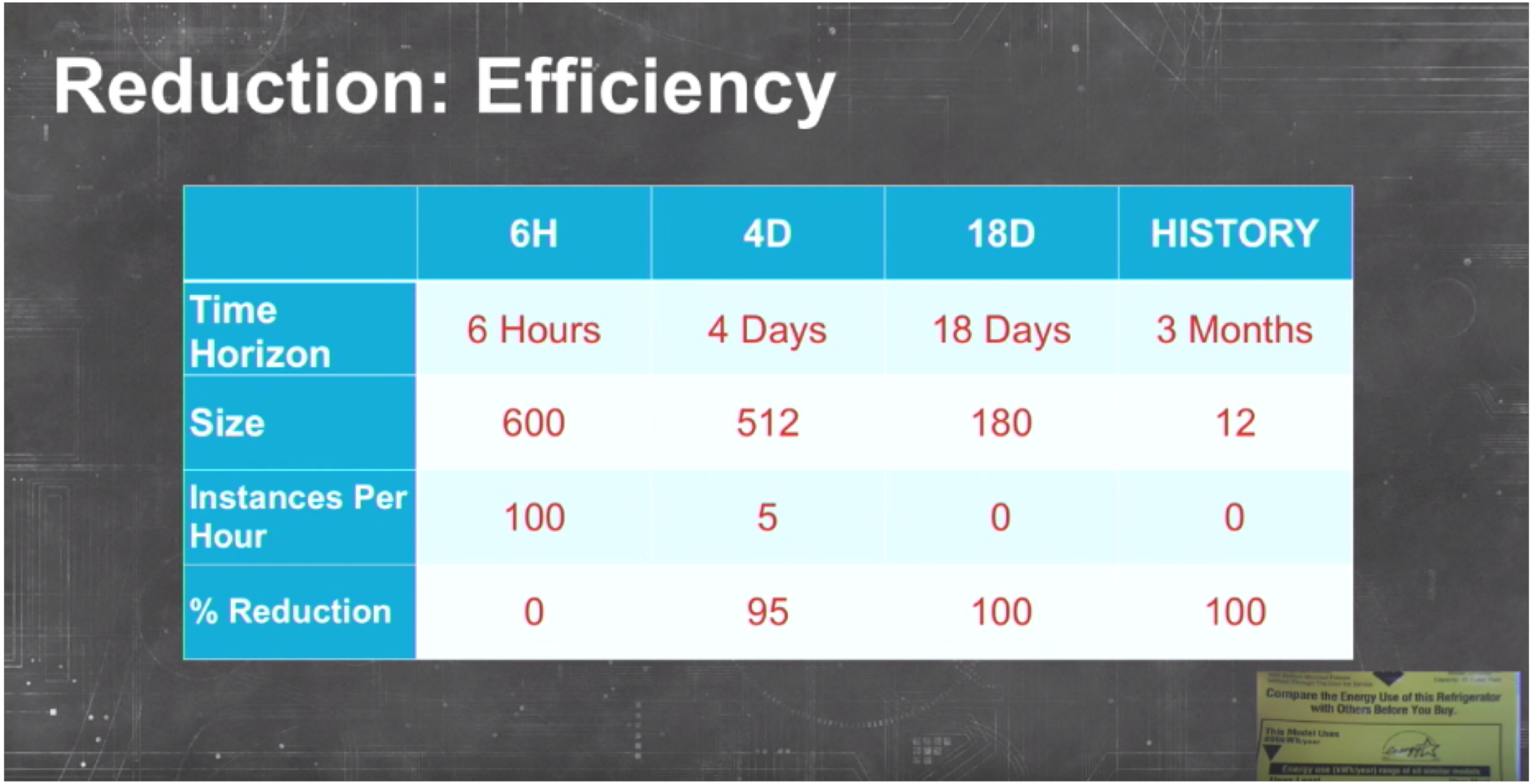
- **Loosely coupled architectures**
  - Parameterized configuration and service discovery
  - Independent component lifecycles
- **Fine grained component separation**
  - Identifying domains of logical responsibilities
- **Identifying and managing state**
  - Preference to purely stateless or purely stateful components

- **This is a high-level technology design viewpoint**

Microservices as implementation patterns

- **“Architecture astronauts” often overlook practical but important concerns**
  - Logging, tracing and monitoring
  - Edge cases such as cold restarts, bad nodes
  - Deployments and resource scaling
- **Operational and implementation patterns**
  - Logging sidecars, external services, distributed tracing
  - Blue/green deployments, gradual rollouts
  - Testing live systems

- **This is a practical / operational viewpoint**



Reduction: Efficiency

	6H	4D	18D	HISTORY
Time Horizon	6 Hours	4 Days	18 Days	3 Months
Size	600	512	180	12
Instances Per Hour	100	5	0	0
% Reduction	0	95	100	100

Compare the Energy Use of this Refrigerator with Others Before You Buy.
This Model Uses

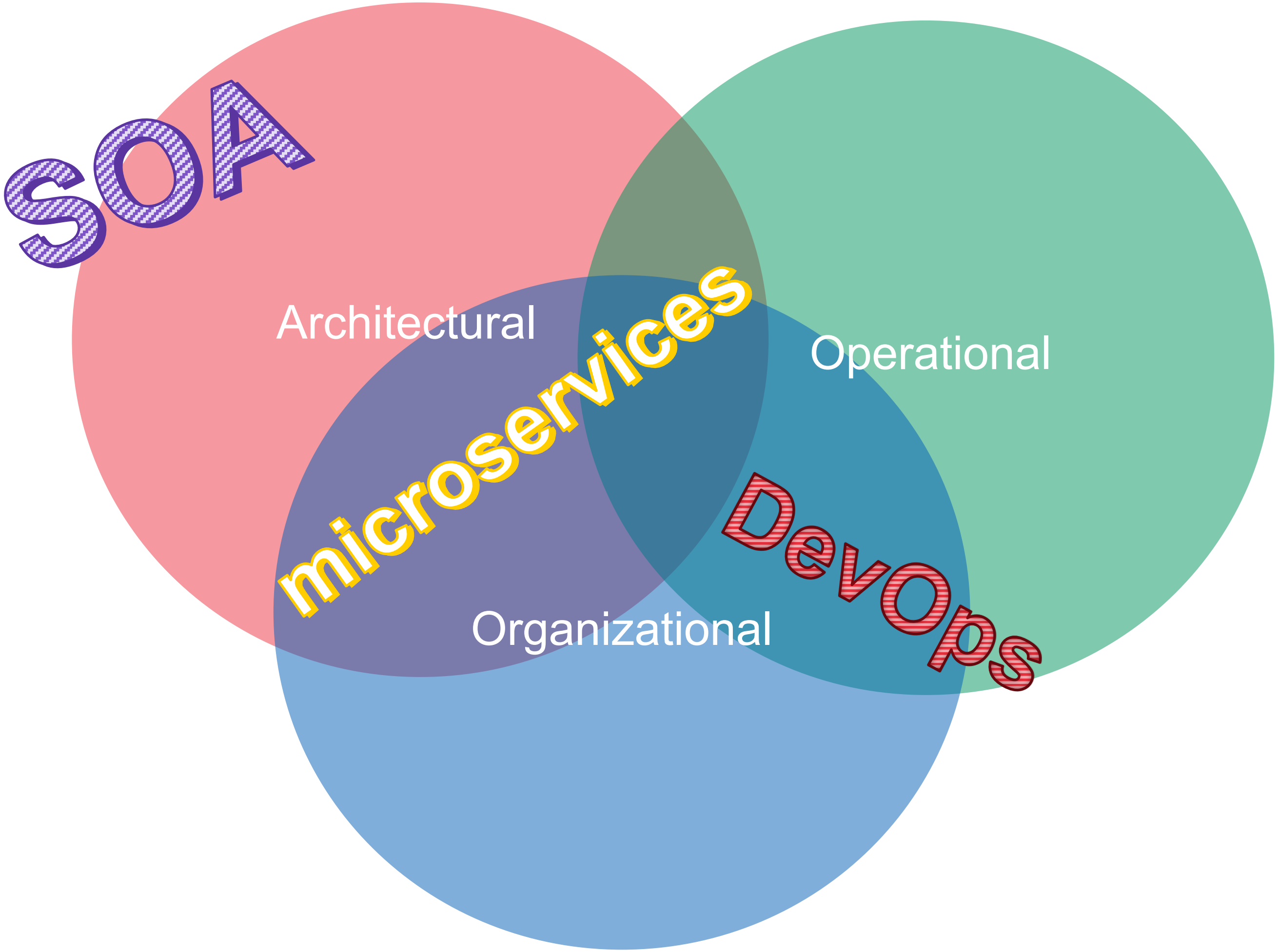
Source: YouTube / AWS

Microservices as organizational structure

- **Conway’s law**
  - "organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations.”
  - Define system by organization, or organize by system design
- **Two-pizza rule for team size (Bezos)**
  - Minimize friction on internal communication
- **Formalize external interfaces**
  - Service contracts, SLAs → DevOps

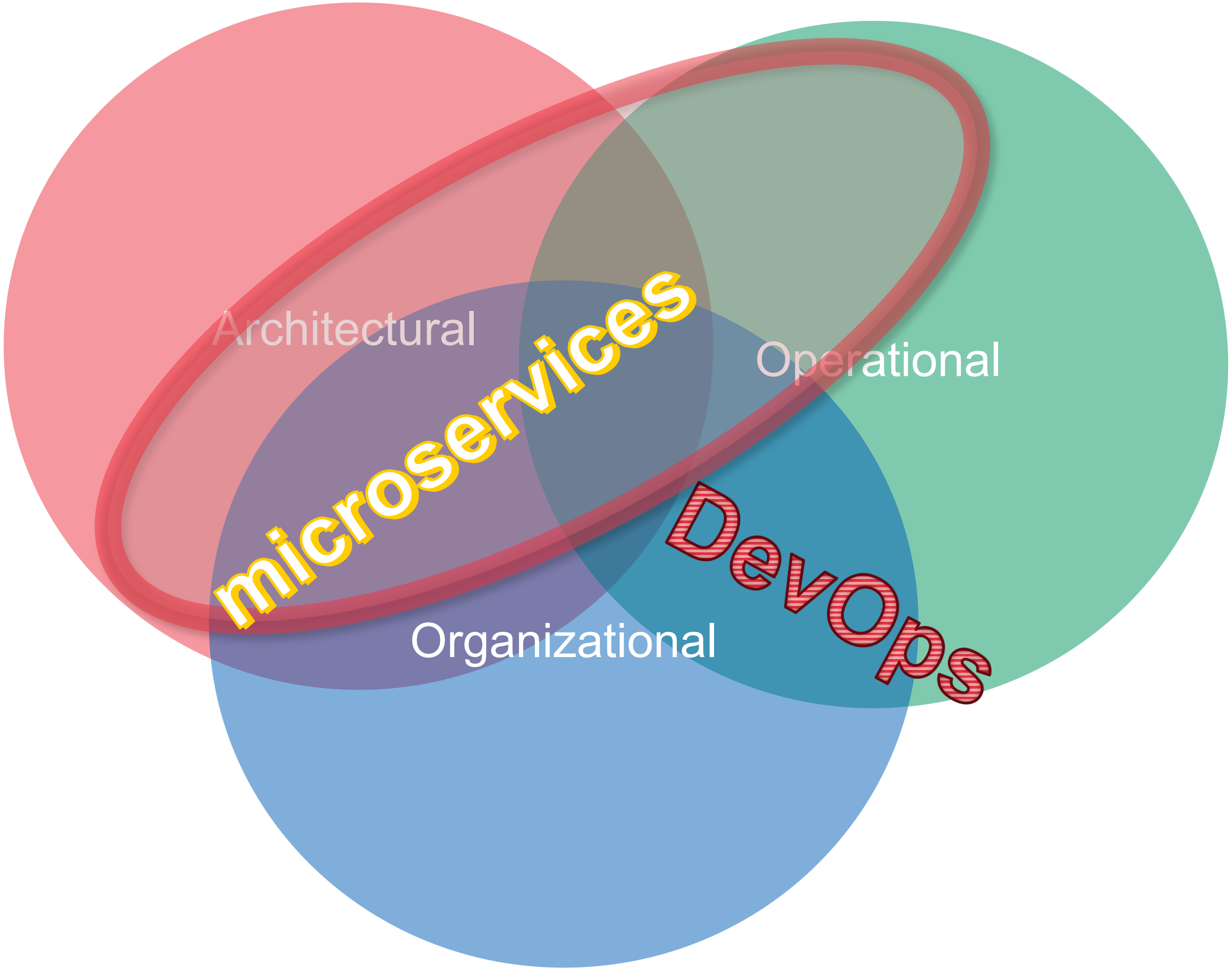
- **This is a management viewpoint**

What is a microservice?


SOA
Architectural
Operational
microservices
DevOps
Organizational

Just a word on DevOps

- **DevOps is the practice of integrating operational responsibility to development team**
  - ”You code it, you deploy it, you get the alarm call at night”
- **DevOps does not imply microservices nor vice versa**
  - OTOH, on a practical level, gaining maximum benefit from microservices (arch + ops + org) implies DevOps
- **Further elaboration as GitOps and infrastructure-as-code**
  - Repository-driven deployment model
  - Code as source of ground truth on infrastructure


Architectural
Operational
Organizational
microservices
DevOps

Serverless

- Serverless defined as a “Function-as-a-Service”
  - Service that runs functions when a request or event occurs
  - Not bound to any particular server or hardware, autoscaled
- Serverless more during later lectures
- Most of architectural and operational aspects of microservices apply to serverless as well

- Serverless as even more fine-grained evolution

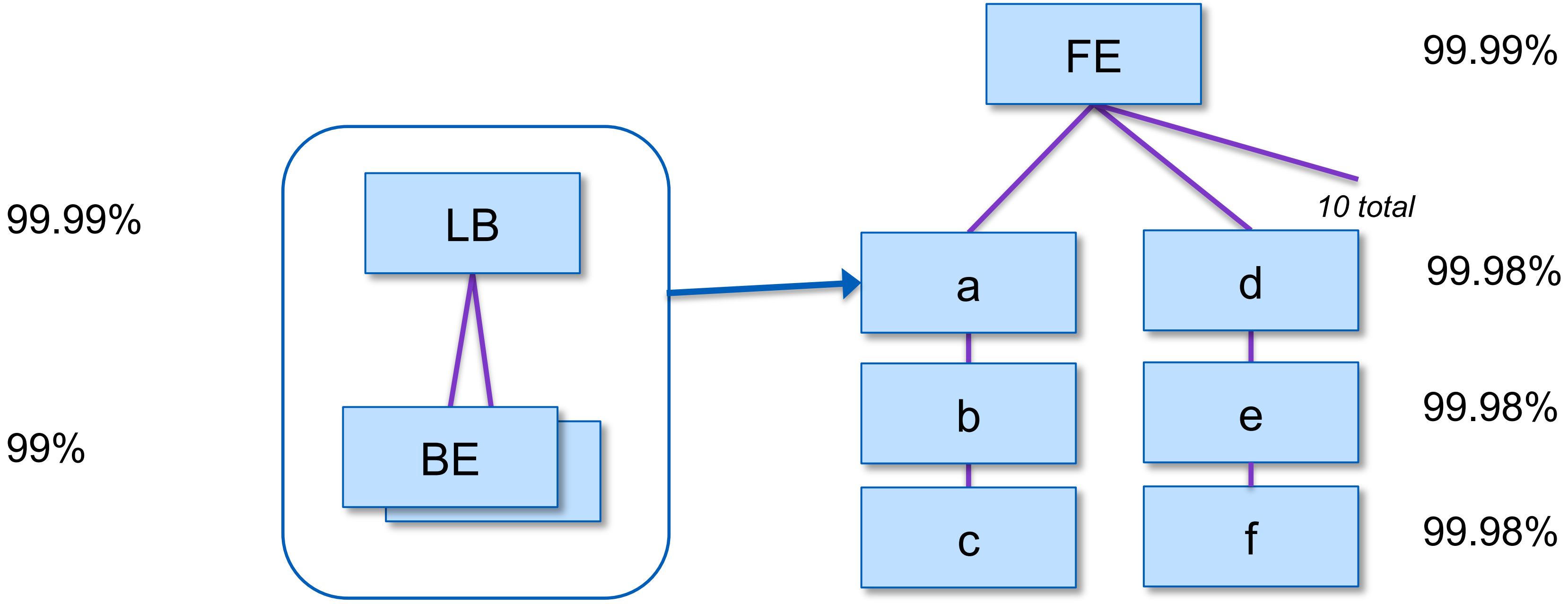
Why microservices?

Pros of microservices

- **Helps managing large development organizations**
  - Clearer responsibilities, divisions of labor
  - Easier to scale at team and individual level
- **Increases development velocity**
  - Independent decisions in teams, formal dependencies
  - Intra-team communications more focused
- **“Product” viewpoint (vs. “project”)**
  - Easier to focus on customer needs than managing schedules

Cons of microservices

- **Increases development overhead**
  - Repetition of code, configuration etc.
  - In practice, requires investment in automation (CI/CD)
  - Debugging distributed systems notoriously difficult
- **Changes usage patterns and increases operational risks**
  - Distributed services put more load on the network (vs. local IPC)
  - Authority on infrastructure open to misuse and accidents
  - Security harder to monitor and enforce
- **Dependencies between services**
  - Configuration management and versioning require effort
  - Increased number of services leads to lower availability, higher variance of many service level metrics

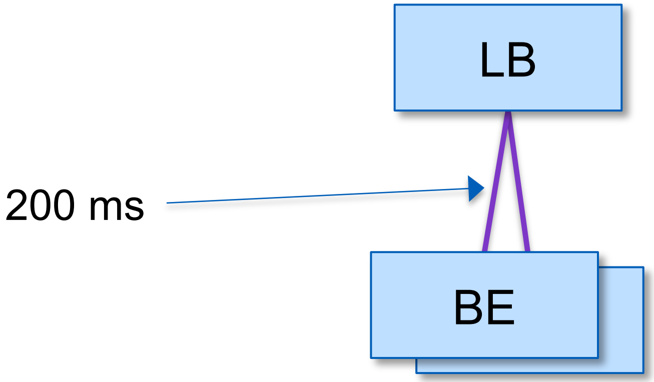


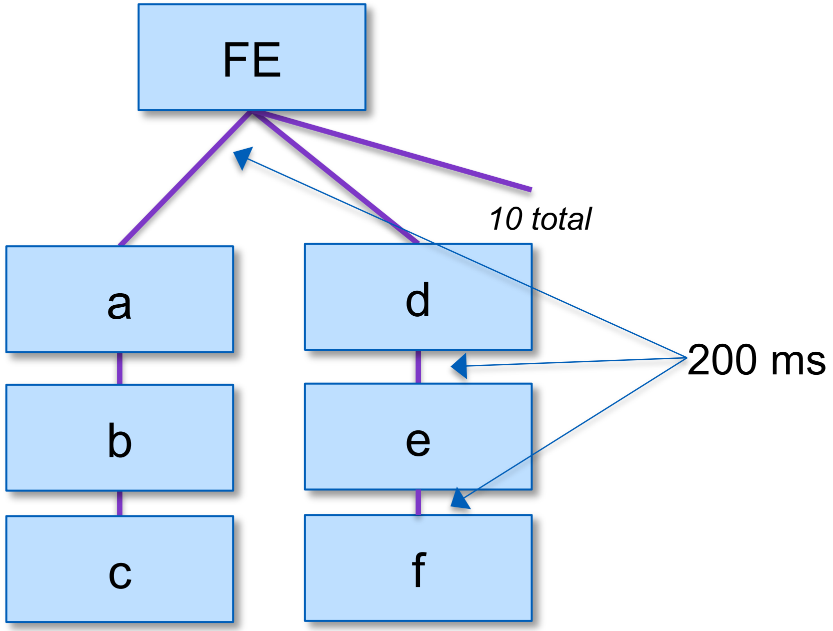
= .9999 * (1 - (1 - .99)^2)
= 99.98%

= 1.7 h / y

= .9999 * (.9998^3)^10
= 99.39%

= 53 h / y



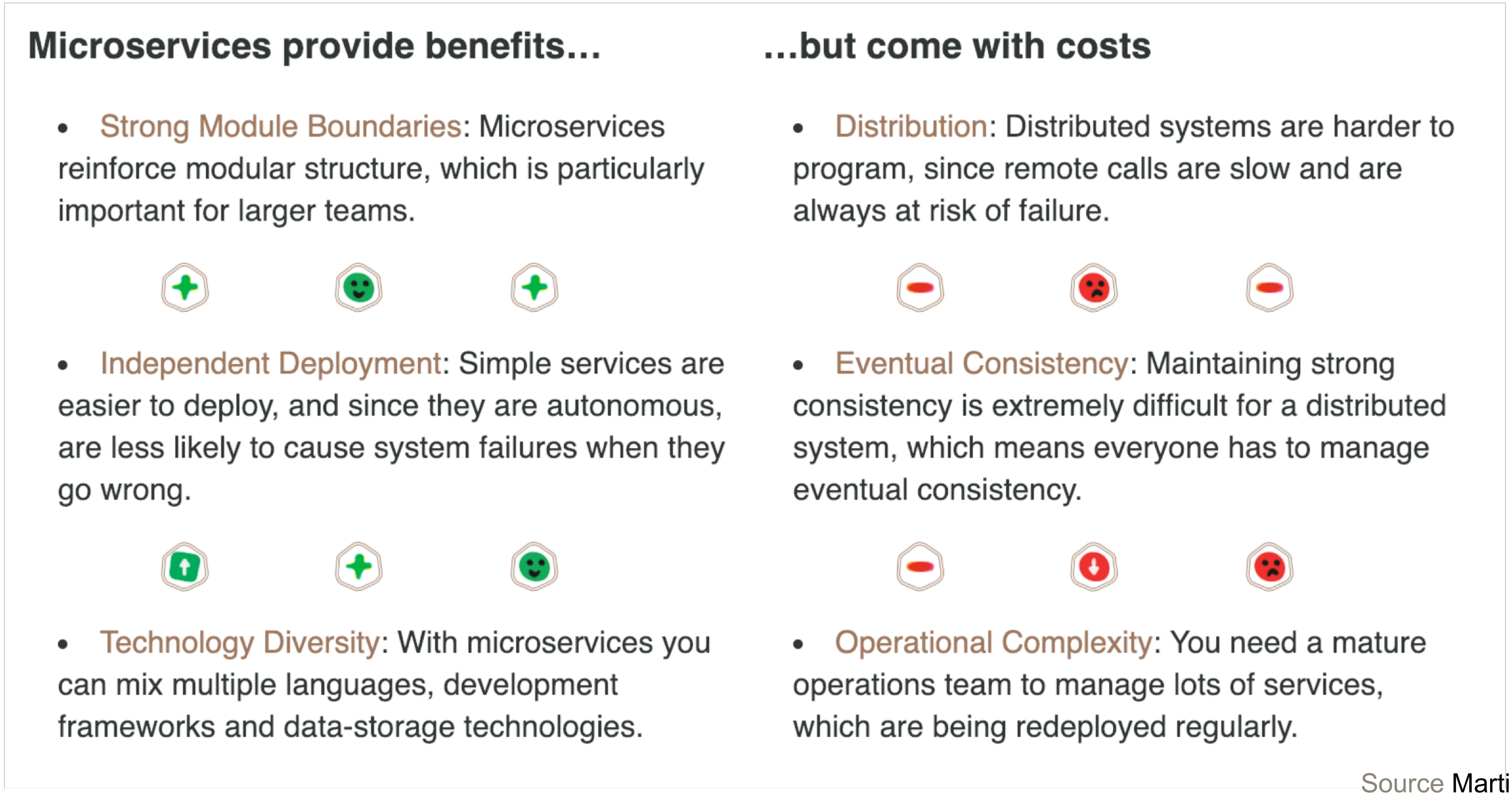


= 200 ms * 1
= 200 ms

= 200 ms * 3 * 10
= 6000 ms = 6 s

If for single inter-service request E[T] = 200 ms,
but P[T > 10000] = 0.01 then …
P[T > 10000 for any inter-service request] = 1 - (1 - .01)^30 = 26%

Summary in the words of Martin Fowler



Microservices provide benefits…

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.
- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
- Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

…but come with costs

- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

Source Martin Fowler

Tools and terminology

Aalto University
School of Electrical
Engineering

Containers

- **Containers commonly used for microservice development and deployment**
  - Not a requirement — more "infra" services often deployed directly on hypervisor or pure metal
  - Docker, Rocket, …
- **Orchestration systems for multiple containers**
  - Docker Compose & Swarm, Kubernetes, …
- **Even higher level orchestration workflows and infrastructure management (not container specific)**
  - Spinnaker, Terraform

Cloud container service providers

- **Amazon Web Services (AWS)**
  Google Cloud Platform (GCP)
  Microsoft Azure
  - All have functionally similar offerings
  - Virtual machines (compute), storage, networking, …
  - AWS Elastic Container Service (ECS) and Fargate for Docker
  - AWS Elastic Kubernetes Service (EKS),
    Google Kubernetes Engine (GKE),
    Azure Kubernetes Service (AKS)
  - Remember: Kubernetes runs Docker images

Docker and Kubernetes

- **Will have a tutorial in next lecture**
- **You can install Docker and Kubernetes locally**
  - Docker (Mac/Win): https://www.docker.com/get-started
  - Docker (Ubuntu): https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce
  - Minikube: https://kubernetes.io/docs/tasks/tools/install-minikube/
    - *Mac/OSX Docker has in-built Kubernetes, just not enabled by default …*
- **Can also use ECS/Fargate/EKS/GKE/AKS**
  - … but these require setting up an account, image registry, permissions, etc…