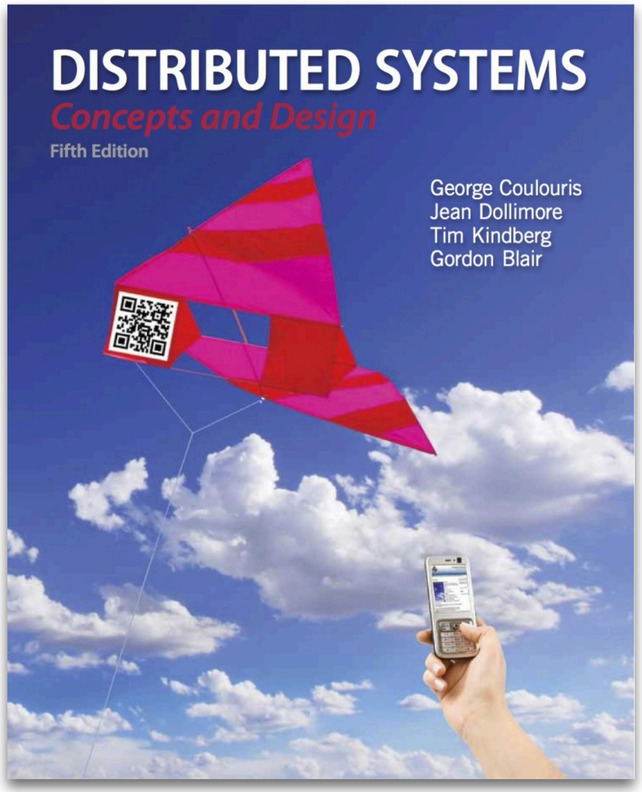
Distributed Systems 2023-2024: Introduction Part 1

Wouter Joosen & Tom Van Cutsem
DistriNet KU Leuven
September 2023

Learning resources

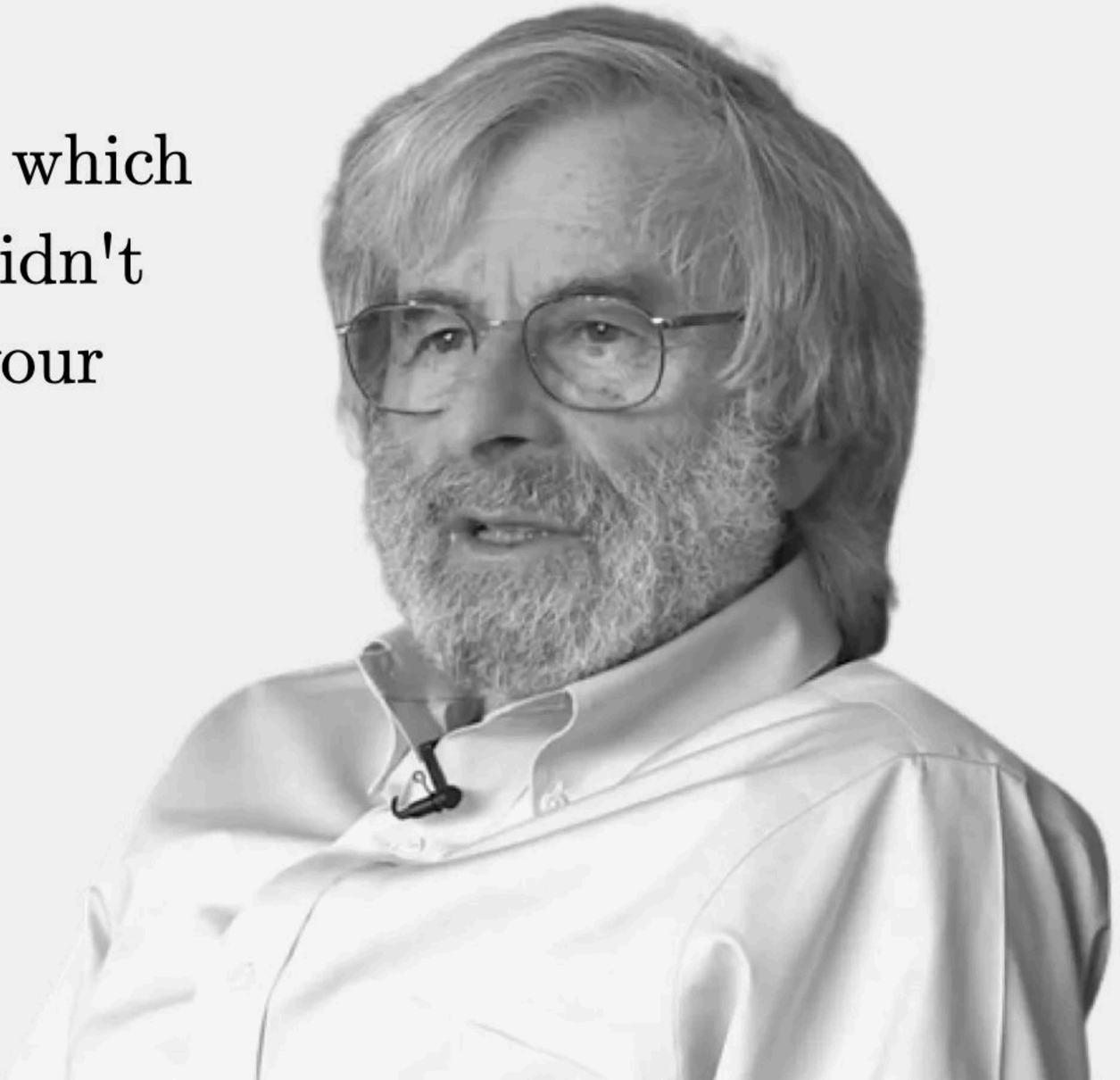
- Textbook, chapter 1



What is a distributed system?

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

- Leslie Lamport



(Image Credit: @FermatsLibrary)

Definitions

- Distributed system =
  – Hardware or software **components**…
  – **interconnected** by a Network…
  – that **communicate** through **message passing**

- Consequences:

  - Multiple components may access shared resources → must manage concurrency!

  - Every component has its own clock → no global clock → no global notion of time!

  - Every component may fail independently → must tolerate *partial* faults!

- Distributed software components are frequently called **processes**

Definitions

- Distributed algorithm =
  – a collection of **independently executing** algorithms …
  – that **cooperate** by sending and receiving **messages**

- Examples:

  - Mutual exclusion algorithms (e.g. “Lamport’s algorithm”): prevent different processes from using the same resource simultaneously

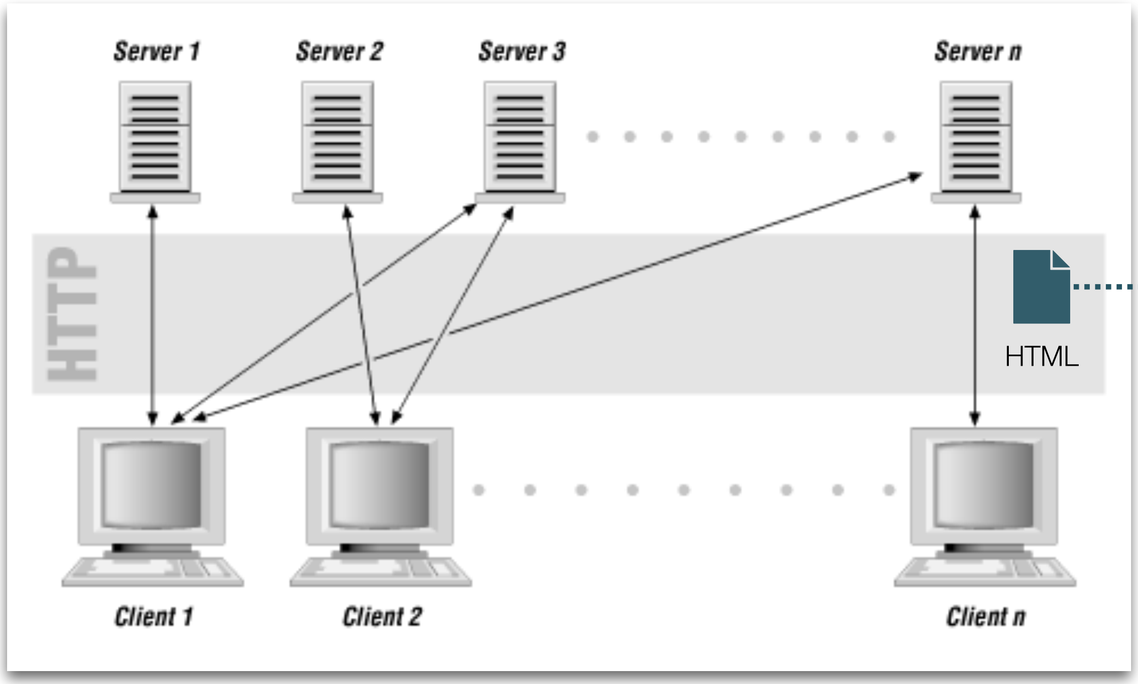
  - Distributed commit algorithms (e.g. “Two-phase commit”): ensure that *either all* processes update their state *or none* of the processes update their state

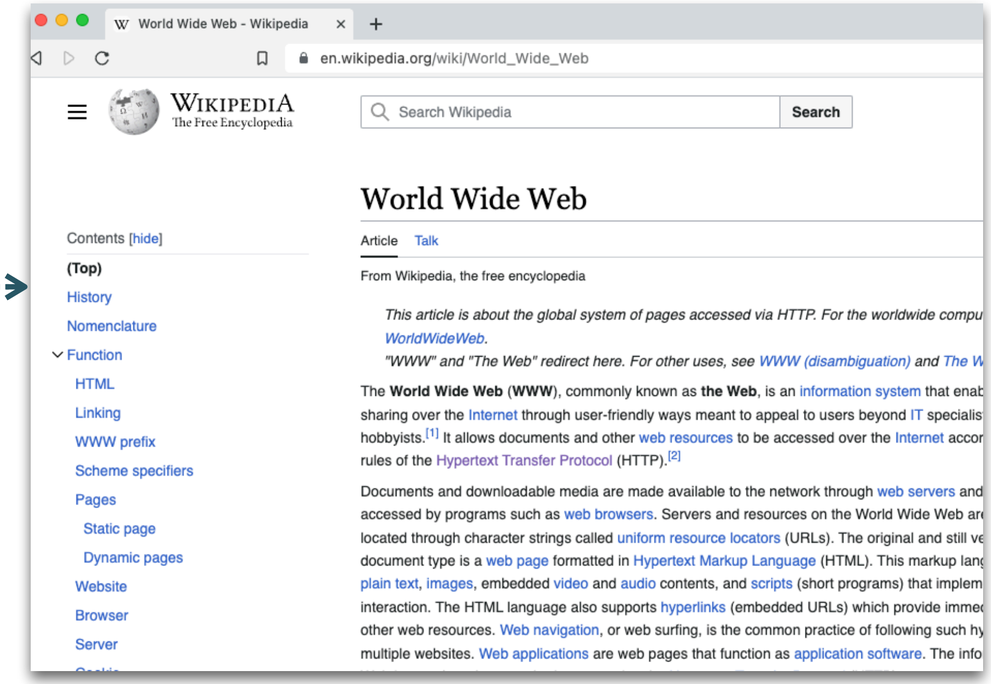
Why make a system distributed? (Motivation)

- The system may already be **inherently physically distributed** (e.g. communication between mobile phones, IoT sensors, browsers & Internet servers or even Low Earth Orbit satellites)
- To improve **reliability:** even if one node fails, the system as a whole keeps functioning (e.g. a replicated database)
- To **share resources** more efficiently (e.g. offload computations from many clients to a shared server in a datacenter to increase utilisation of CPU and disk - a.k.a. “utility computing” or “cloud computing”)
- To be able to **scale** the system with increased workload (e.g. a single server machine cannot serve millions of user requests in real-time, but a cluster of 100 server machines can)
    - Supercomputers are a special type of distributed system, where the hardware components are tightly interconnected with the goal of speeding up a single large computation through *parallel* execution.
- For **security & privacy** reasons, as different computers are administered by different people or organizations (e.g. the mobile banking app on your phone does not give you direct read/write access to your bank’s databases)
- Stop and think. Can you think of additional reasons to distribute a system?

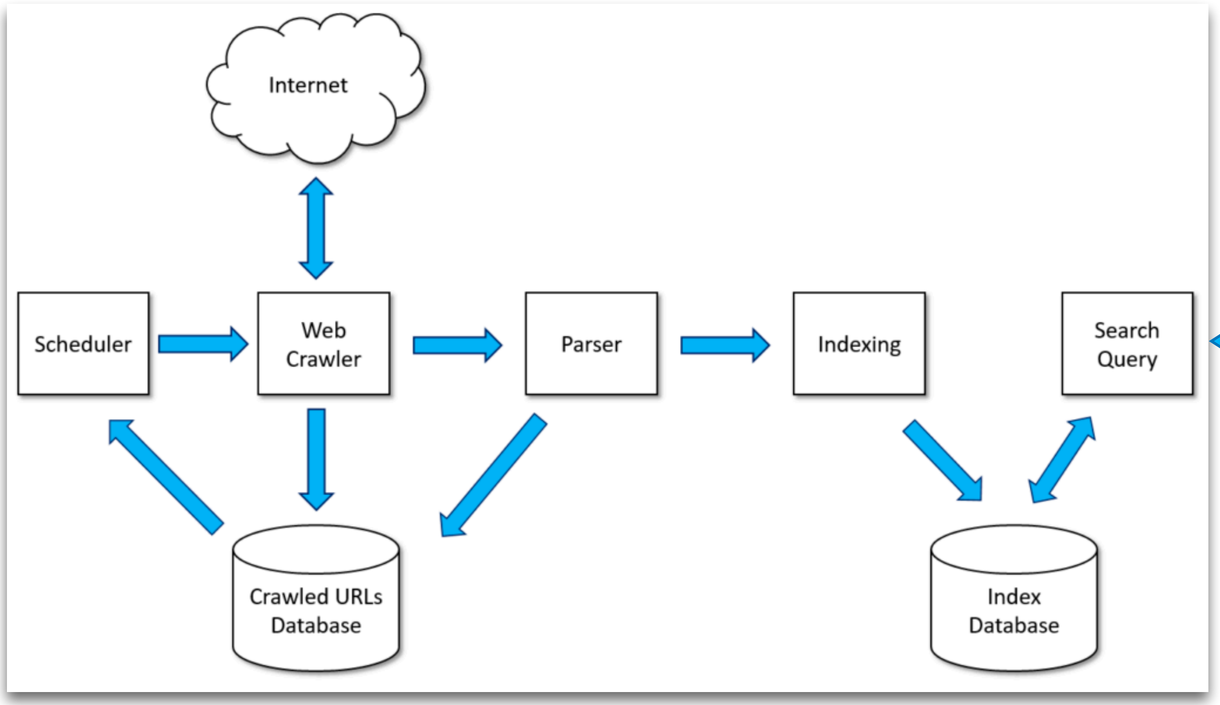
Examples of Distributed Systems

Information systems like the World Wide Web (“The Web”)

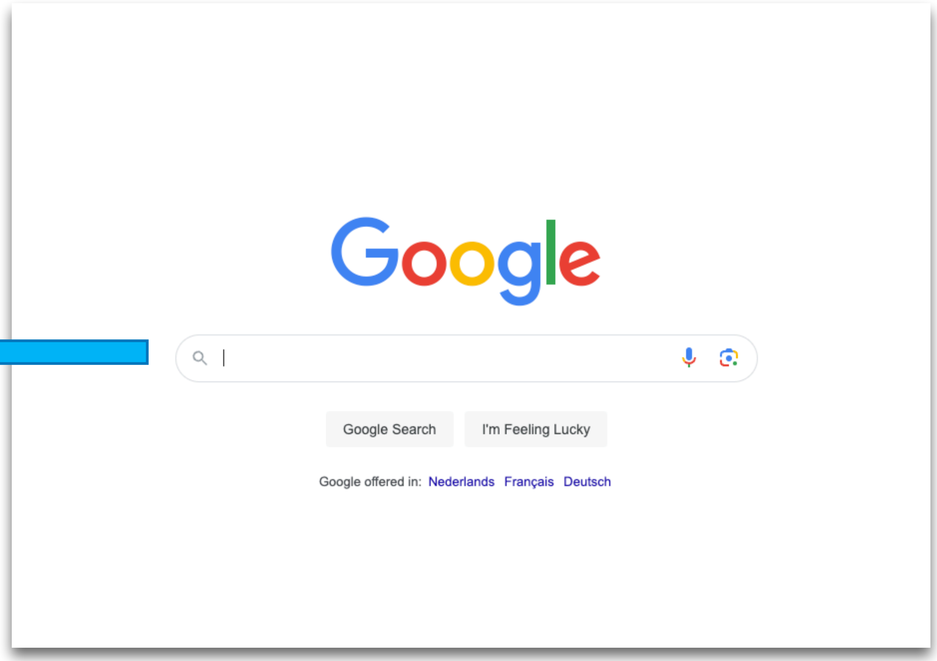




Search engines

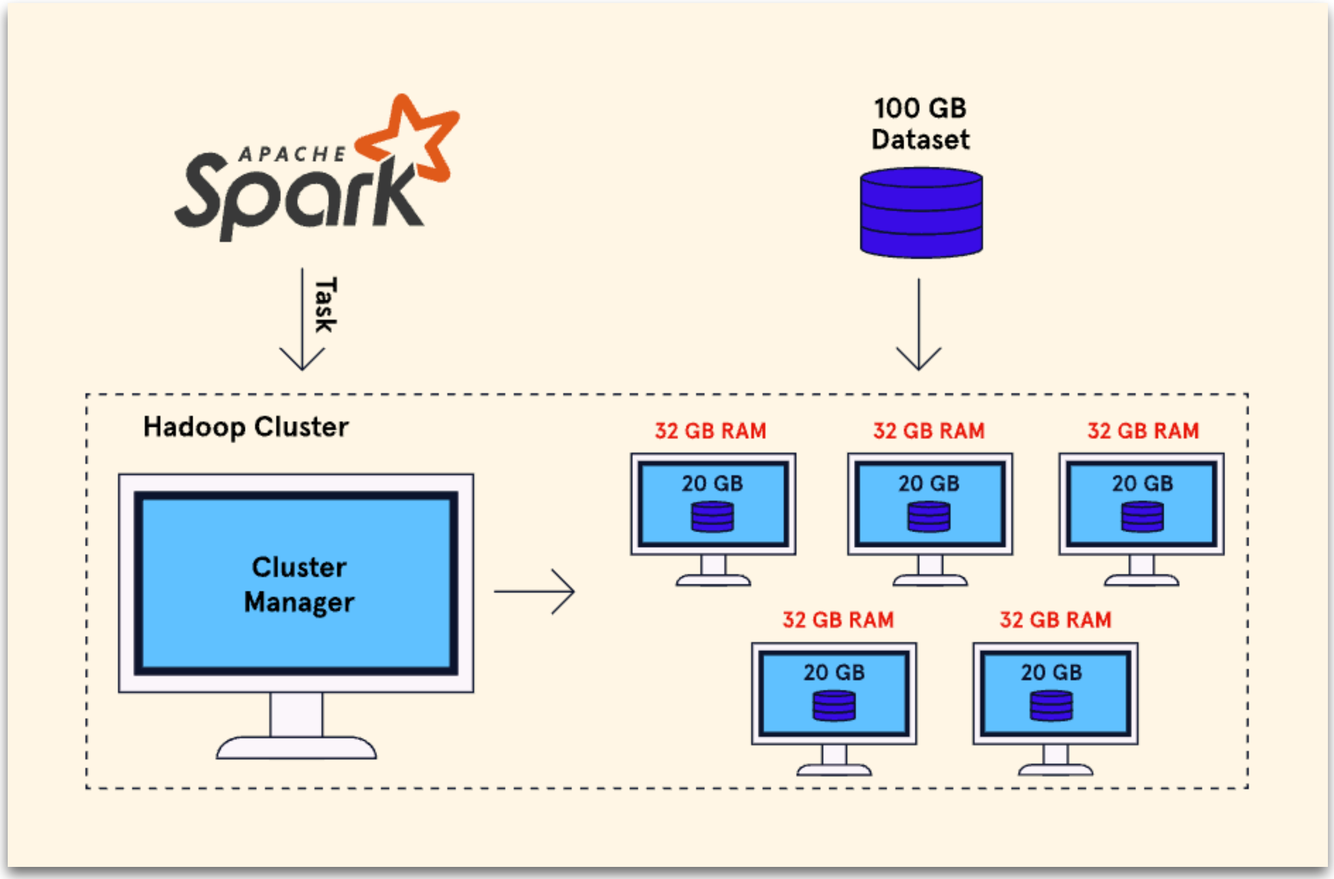


(source: https://aeroadmin.com/articles/en/2020/how-search-engines-work/)



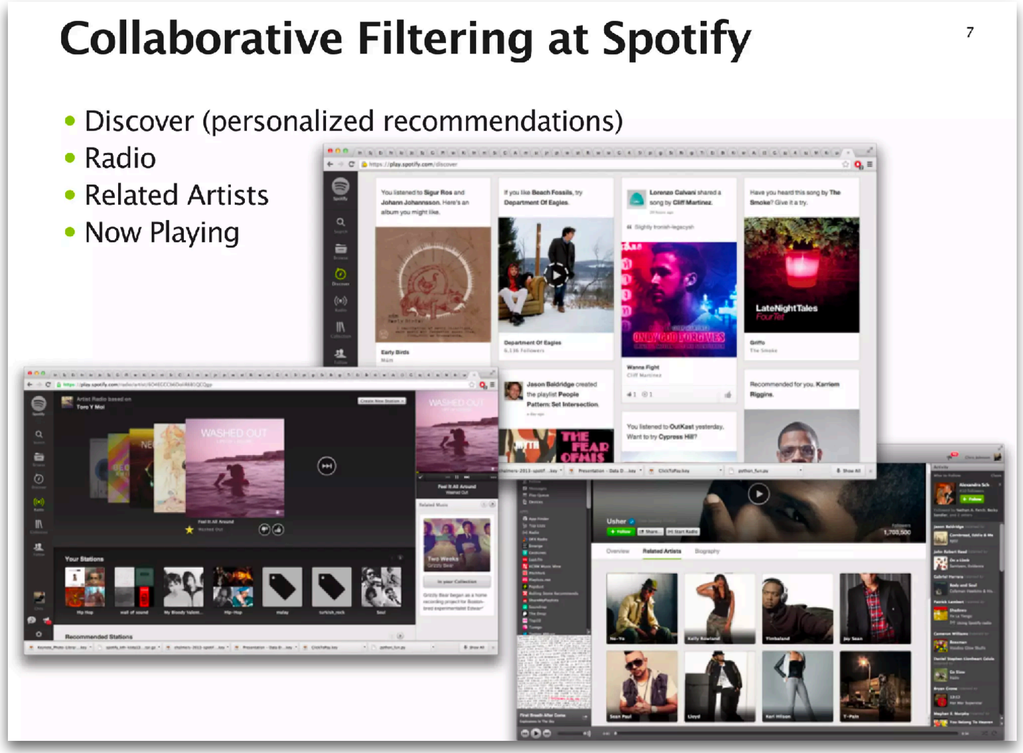
E.g. Google, Bing, Yahoo!, DuckDuckGo, …

“Big Data” processing systems



(Image source: https://www.codecademy.com/article/what-is-spark)

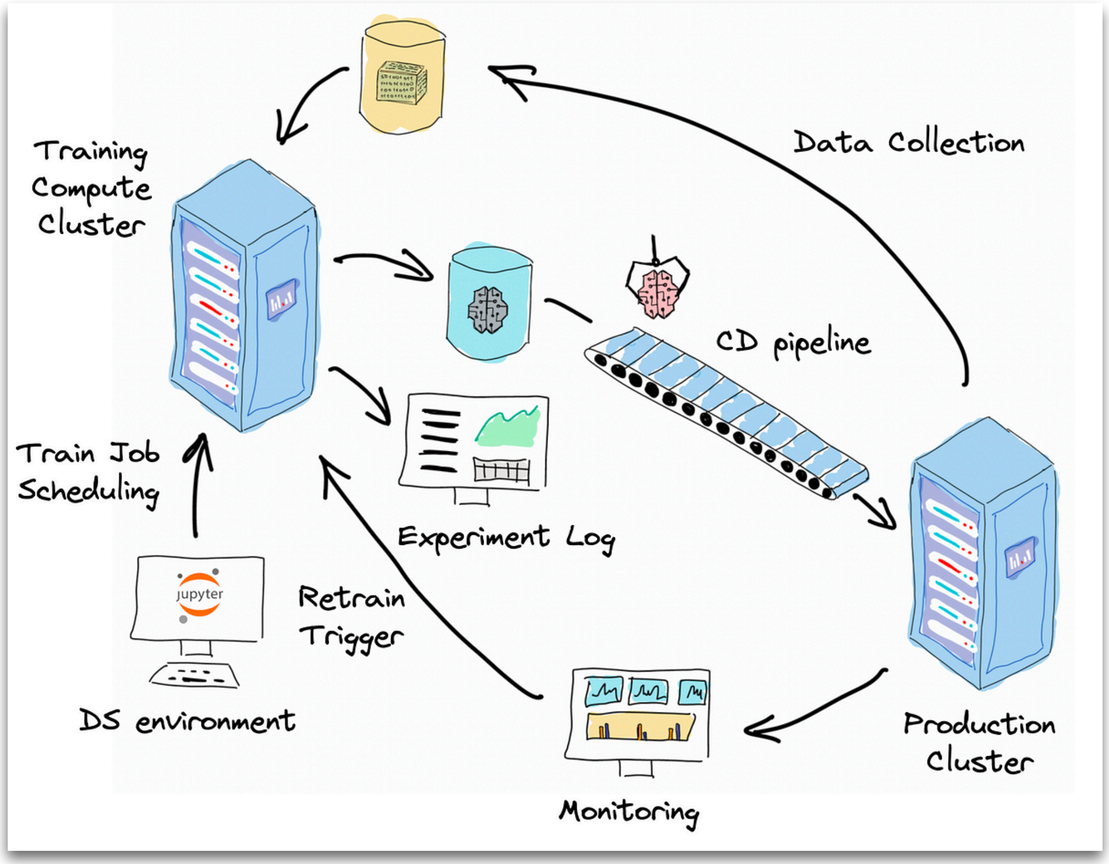
E.g. analyzing user actions to generate recommendations, analyzing credit card payments to detect fraud, etc.



(Source: https://www.slideshare.net/MrChrisJohnson/collaborative-filtering-with-spark)

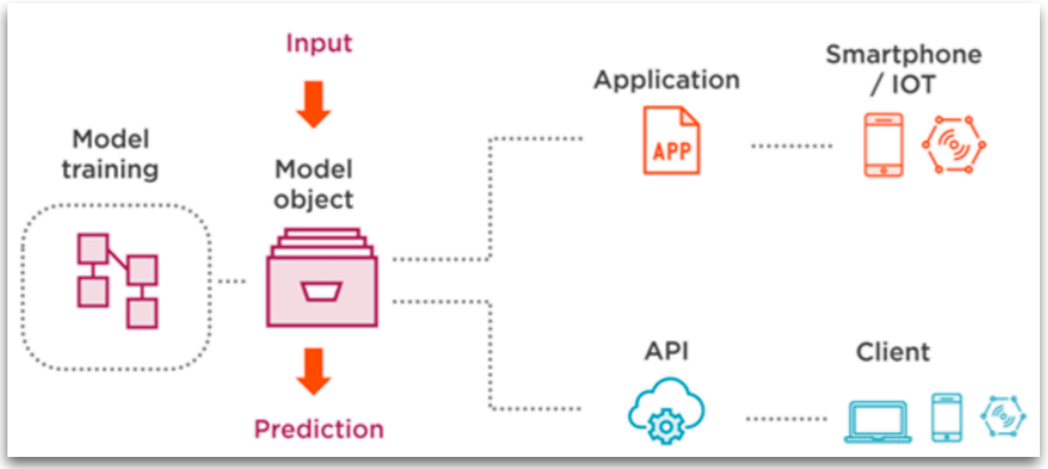
AI/ML Systems

Training ML models



(Source: https://towardsdatascience.com/machine-learning-systems-versus-machine-learning-models-3955d038ea1f)

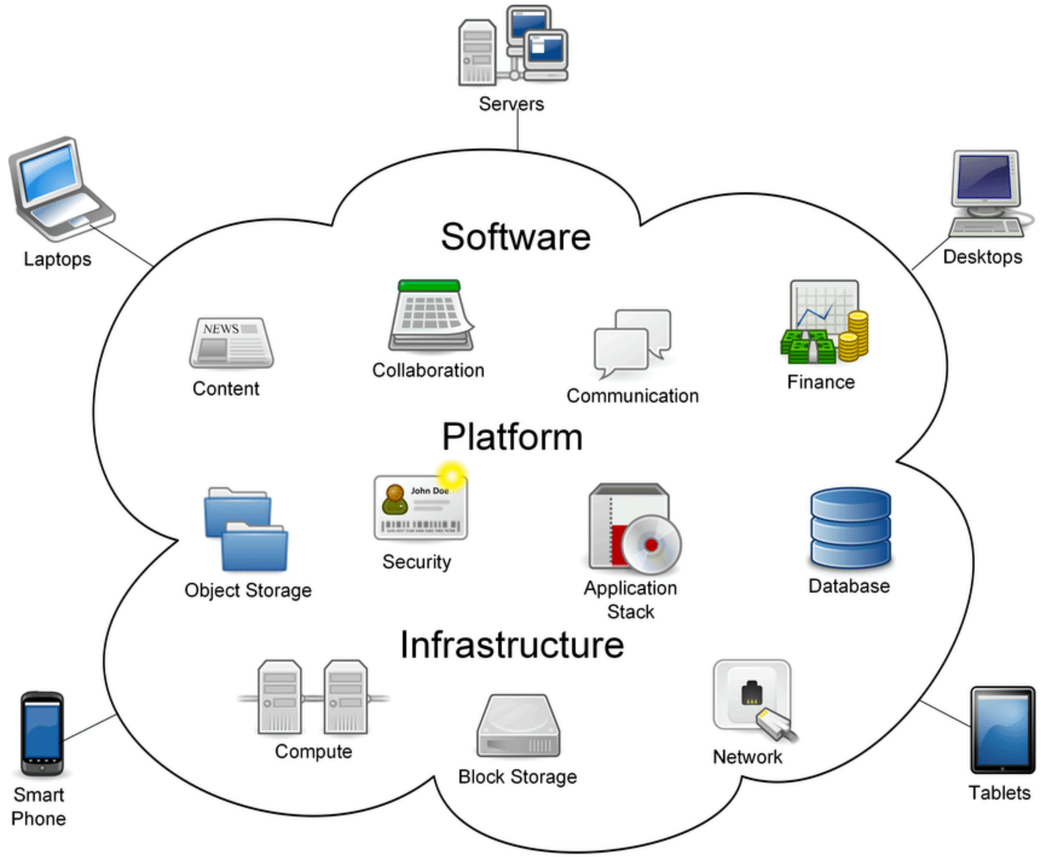
Serving ML models



(Source: https://ubuntu.com/blog/guide-to-ml-model-serving)

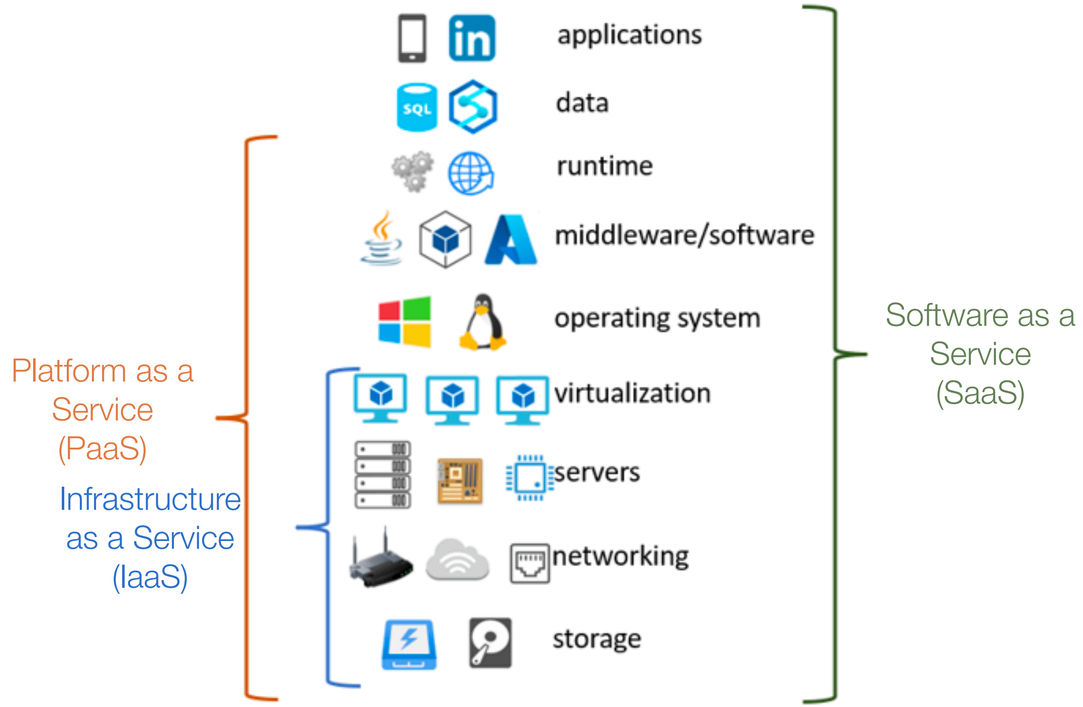
(ML = Machine Learning)

Cloud Computing (“The Cloud”)



(Image credit: Duggan, “The Application of Machine Learning to Optimise Live Migration in Cloud Data Centres”, 2019)

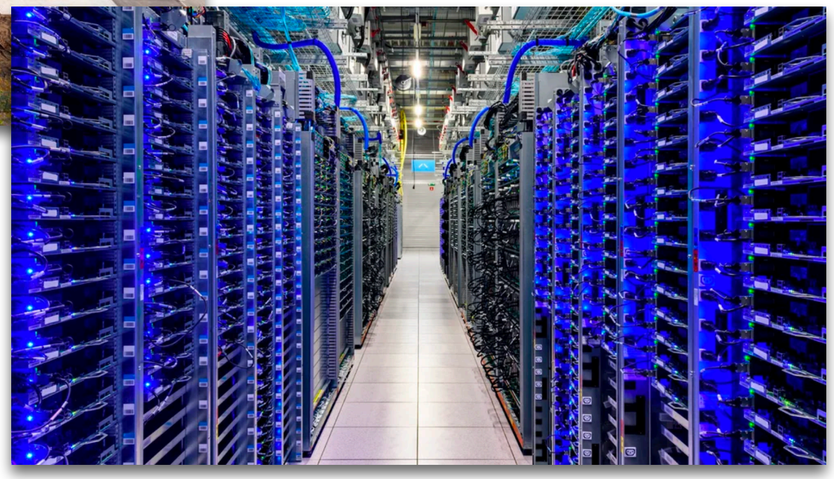
aws Azure Google Cloud



(Image credit: Wise Data Decision)

Cloud Computing (“The Cloud”)

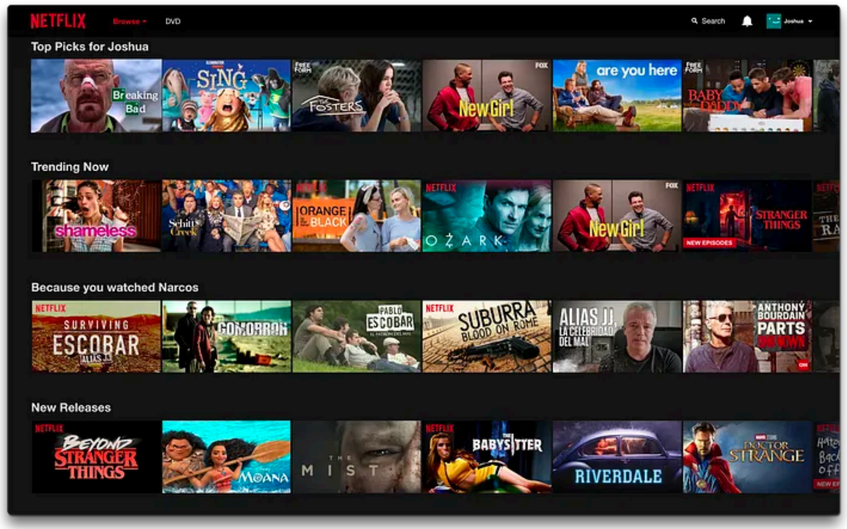
Google's data center in St. Ghislain, Belgium
(Image credit: datacenterknowledge.com)



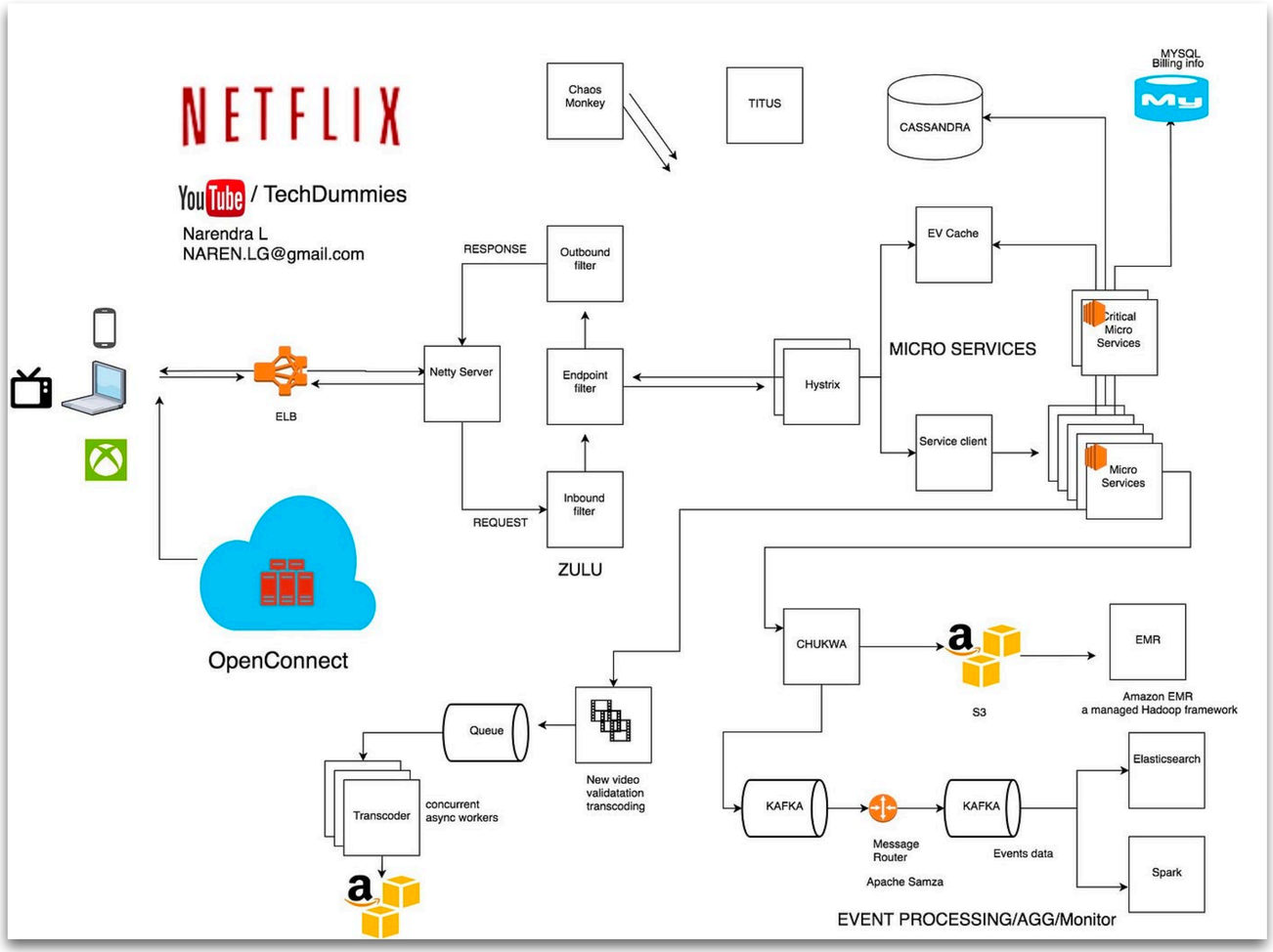
Server racks inside a data center
(Image credit: Google)

(Image credit: Free Software Foundation)

Complex services built on Cloud infrastructure (“Cloud-native”)

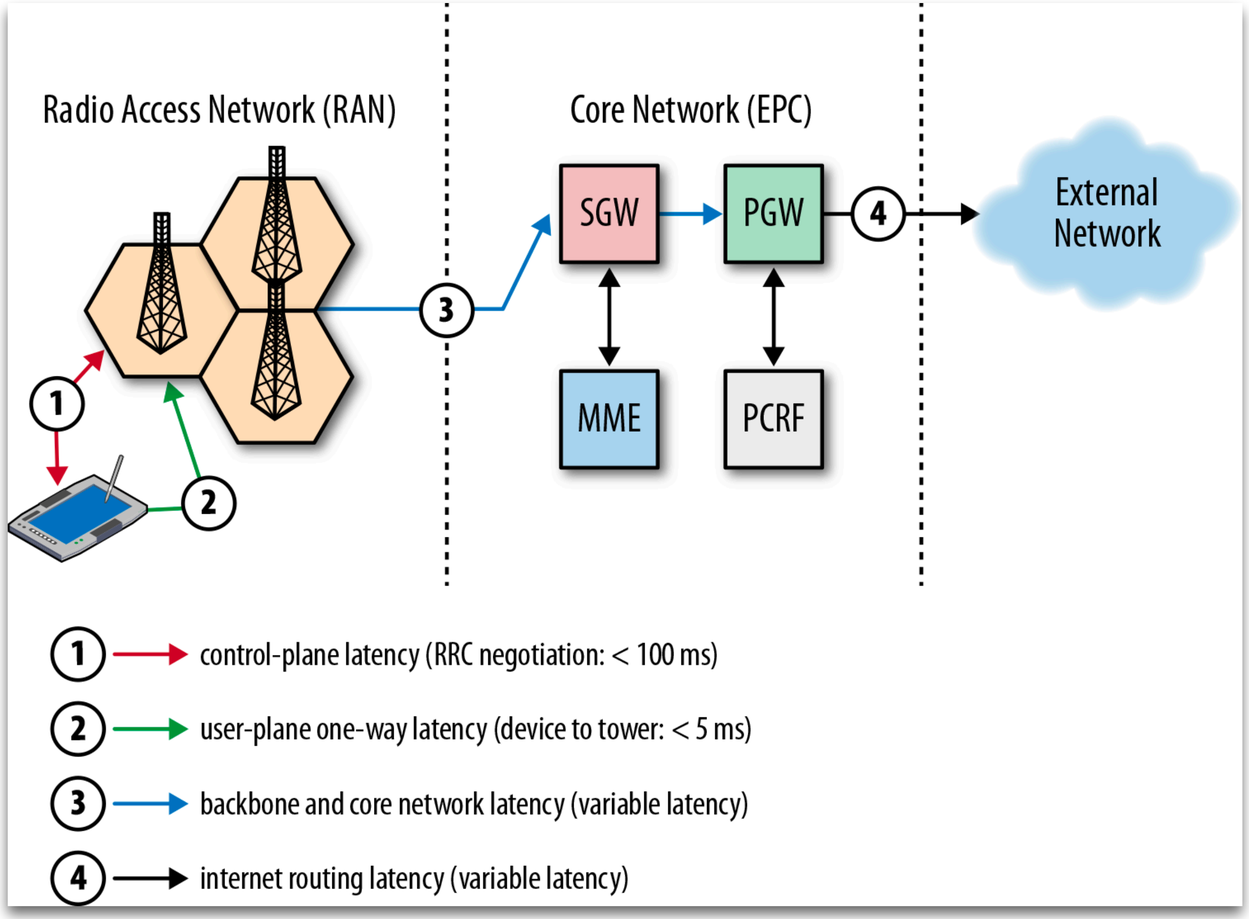


Example: Netflix on AWS



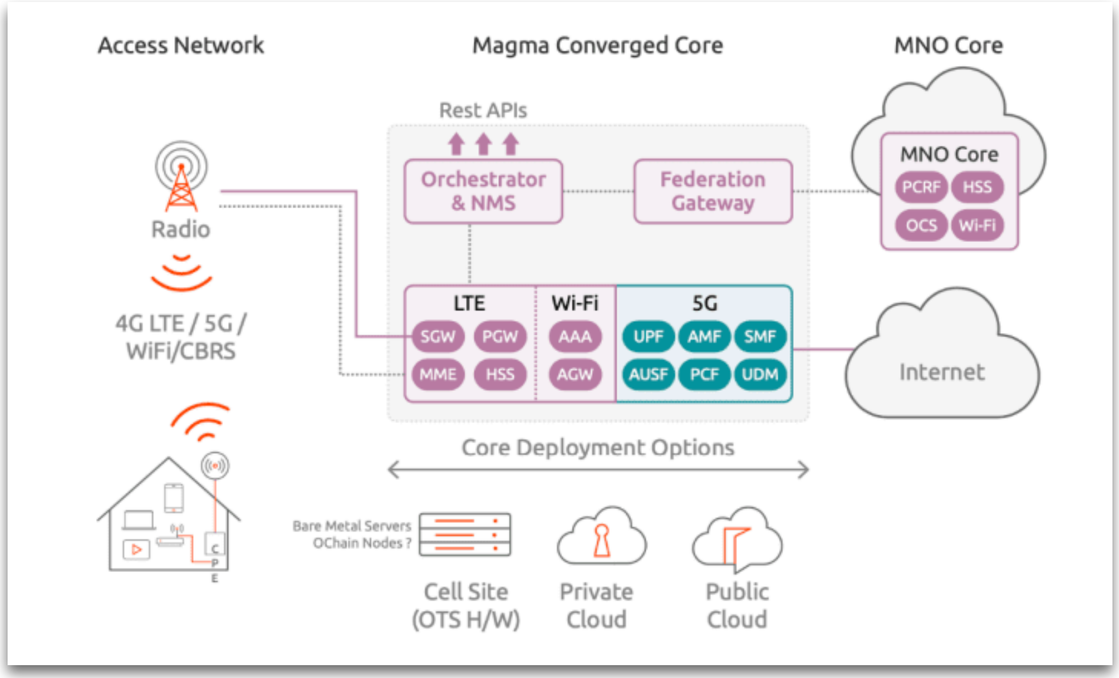
(Source: https://medium.com/@narengowda/netflix-system-design-dbec30fede8d)

Mobile (cellular) networks (e.g. 4G/LTE and 5G networks)



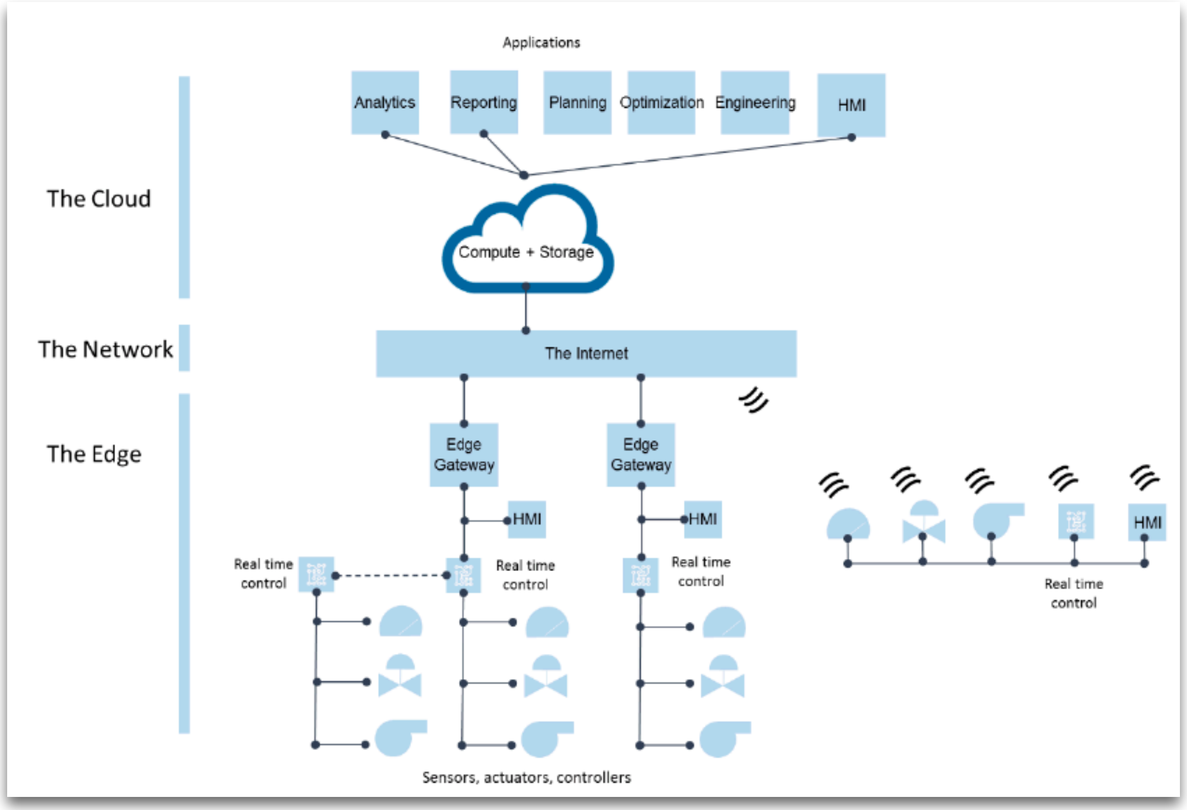
(Image credit: Ilya Grigorik, High Performance Browser Networking, O'Reilly)

Telco/IT convergence: run core network elements as services in the cloud (example: Magma)

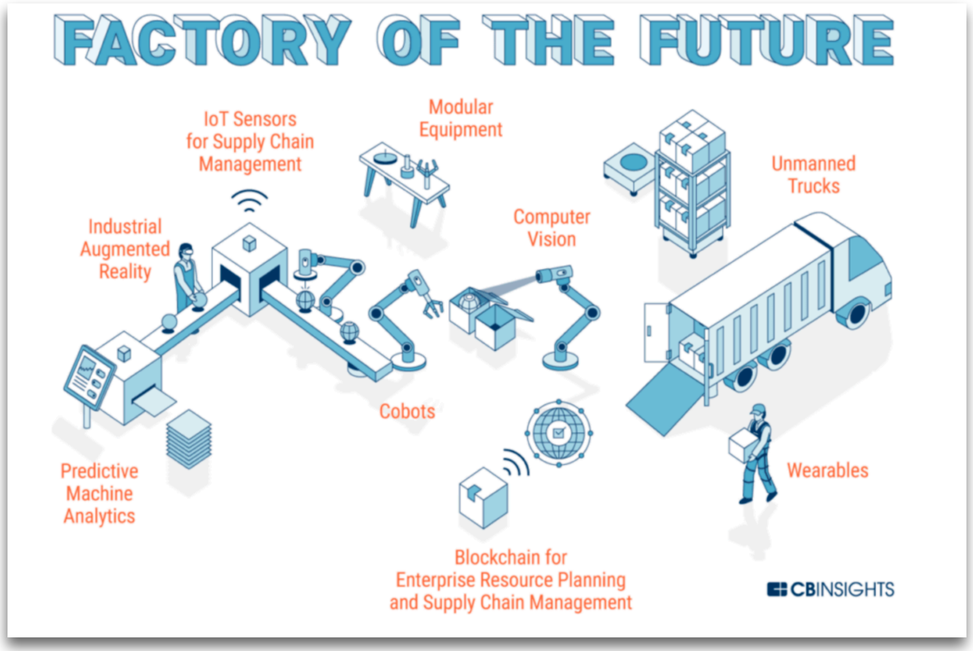


(Source: Ubuntu, https://ubuntu.com/blog/canonical-joins-magma-foundation)

The “Industrial Internet of Things” (IIoT)



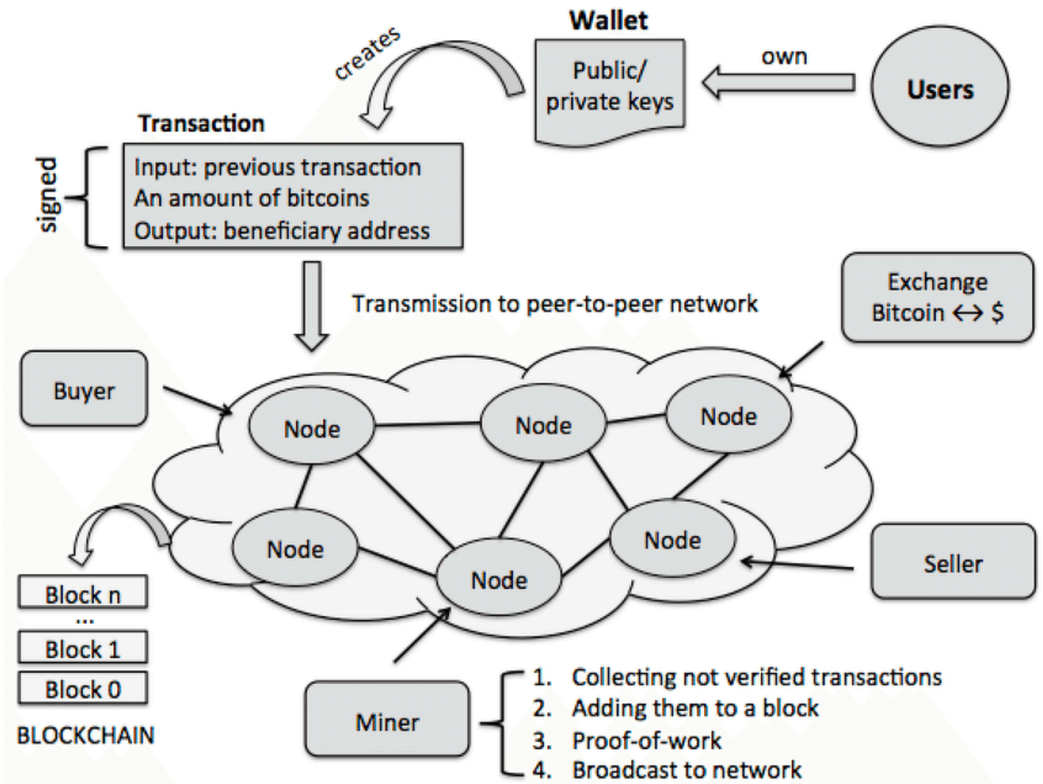
(Source: https://www.everythingrf.com/community/what-is-industrial-iot)
(Figure details not important!)



(Source: https://www.cbinsights.com/research/briefing/factory-of-the-future-manufacturing/)

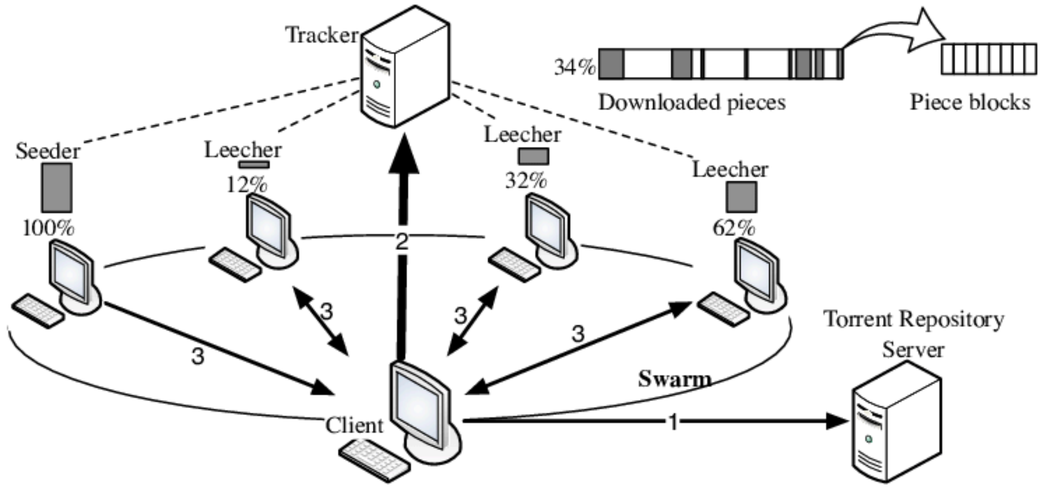
Blockchains and peer-to-peer (p2p) networks

Bitcoin: p2p electronic payments



(Image credit: Bistarelli et al., “An end-to-end voting-system based on bitcoin”, 2017)

Bittorrent: p2p file transfer



(Image credit: Evangelista et al., 2011)

More examples (from the textbook)

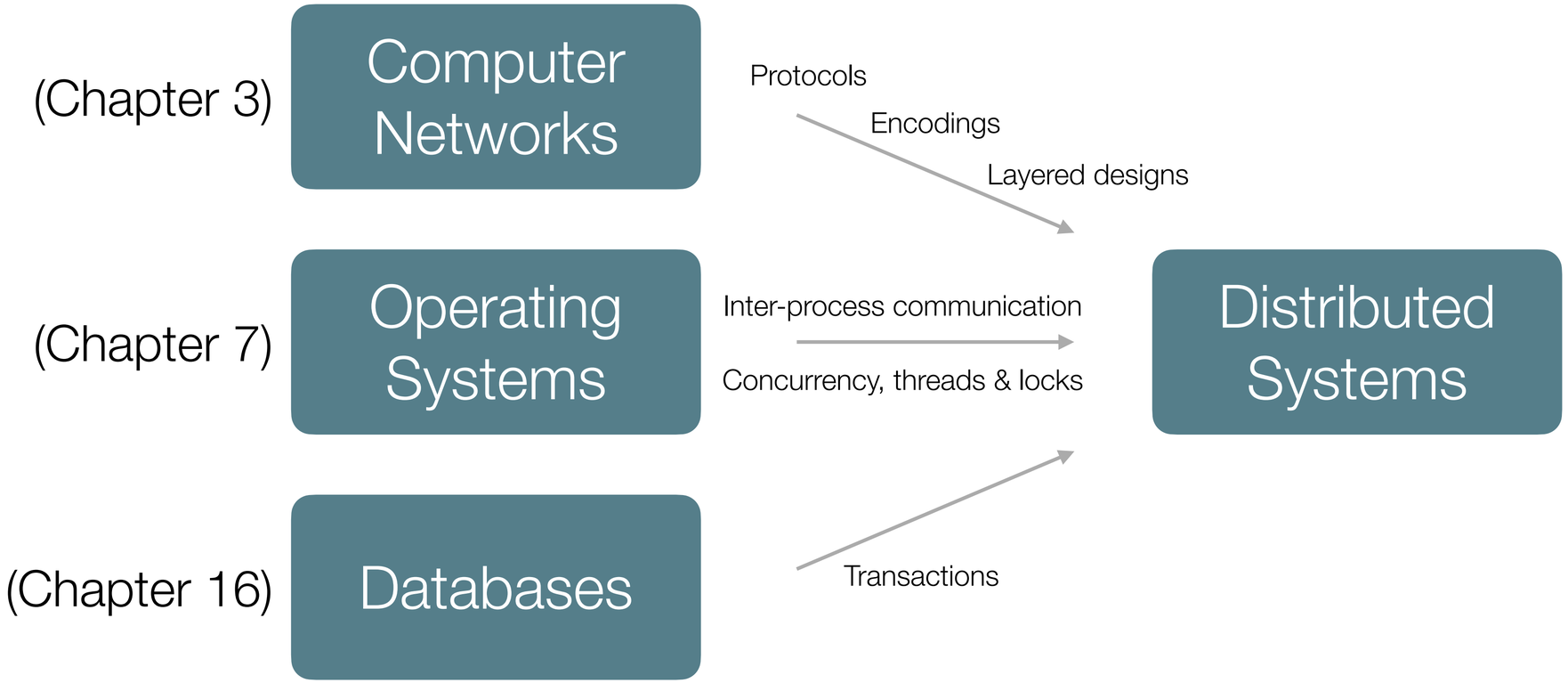
- Financial trading applications
- Multiplayer games
- Internet & intranets
- Mobile & ubiquitous computing
- …
- See textbook sections 1.2 and 1.3

Trends (and where to learn more about them…)

- From the textbook:
  - Pervasive Networking and Modern Internet => Today: WiFi and 5G networks (Prof. Mathy Vanhoef)
  - Mobile & ubiquitous computing => Today: Industrial Internet Infrastructure & sensor networks (Prof. Danny Hughes)
  - Multimedia systems => Today: AR/VR. Specialized courses by the HCI group!
  - Utility Computing (a.k.a. Cloud Computing) => This course! With electives to dive deeper:
    - Capital Selecta Distributed Systems (Topic: Containers & Kubernetes) (Dr. Eddy Truyen)
    - Gedistribueerde Softwarearchitecturen: Verdiepende Studie (Topic: Big Data & NoSQL) (Dr. Davy Preuveneers)
- More recently: AI and Cybersecurity! => entire new Master-after-Master Programmes devoted to these topics
  - Security & Privacy in Contemporary Distributed Systems (Topic: Web3 & Decentralized apps) (Prof. Tom Van Cutsem)

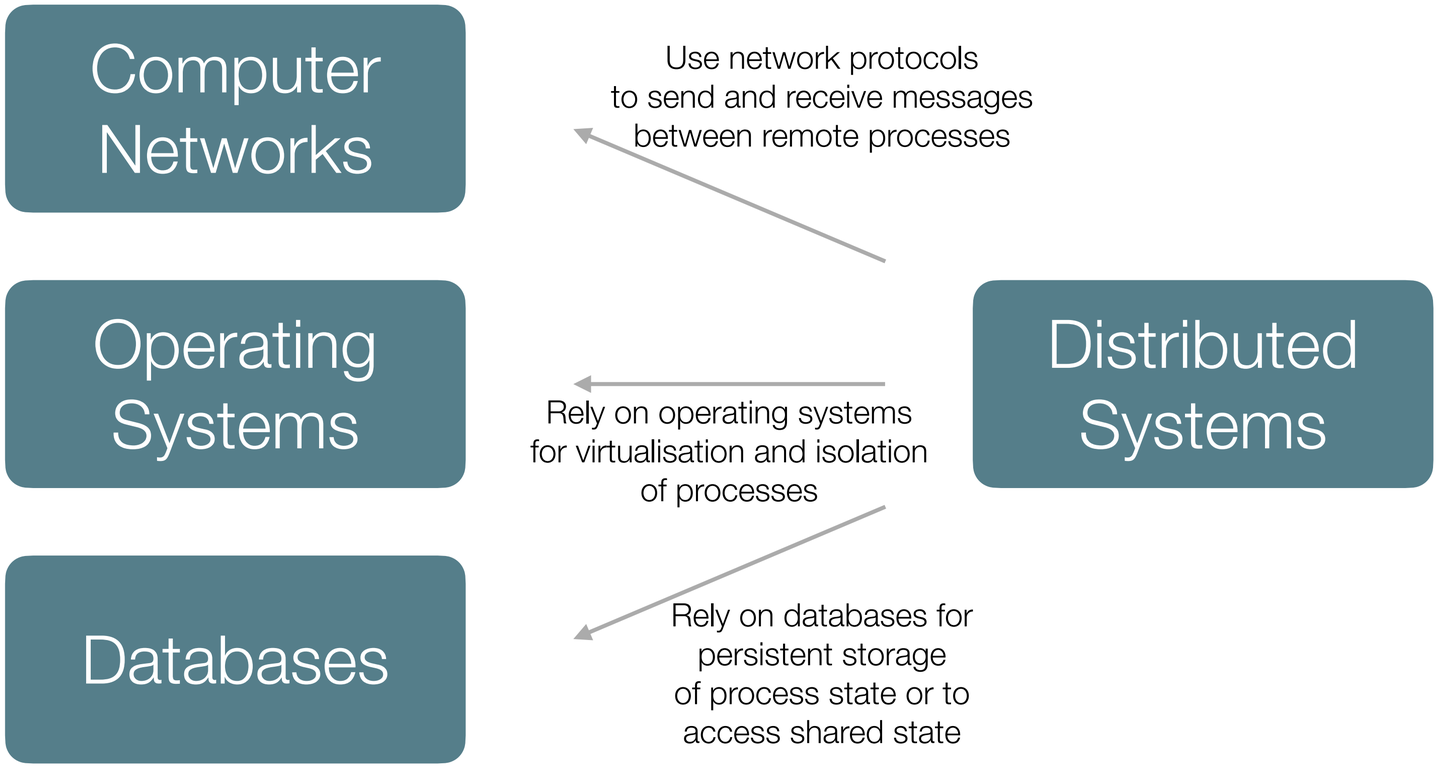
Distributed Systems and relation to other courses in higher education

- Borrow, use and extend basic concepts introduced elsewhere



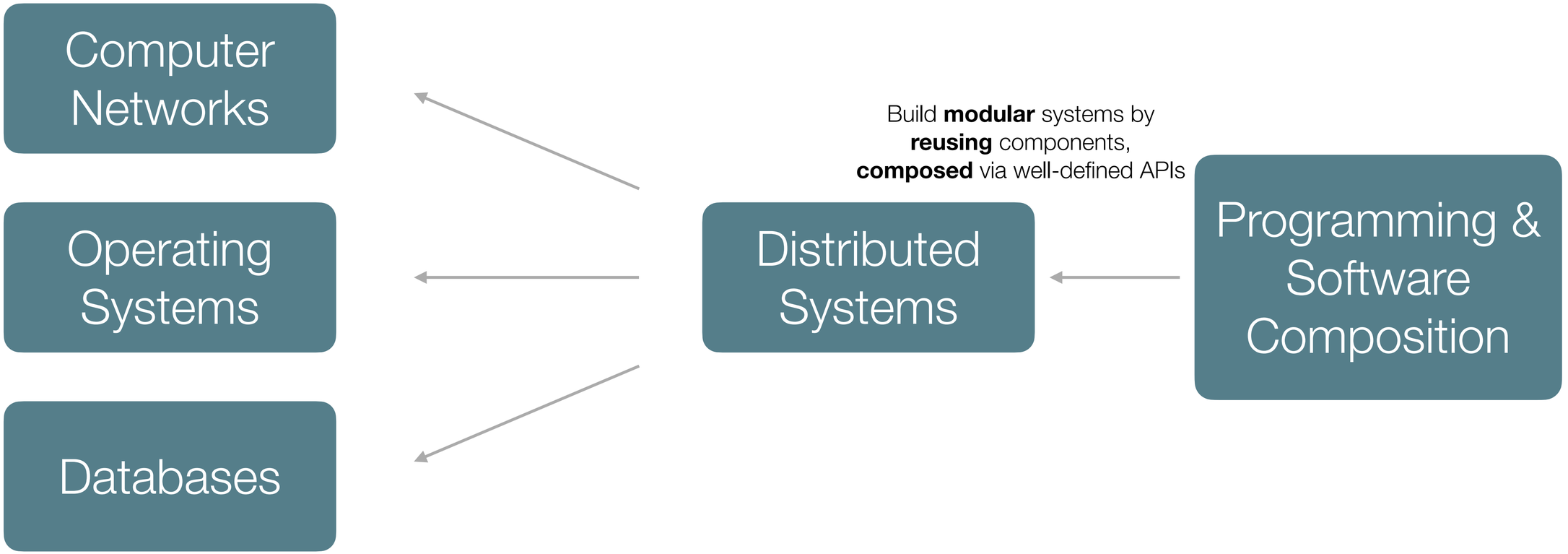
Distributed Systems and relation to other courses in higher education

- But also build on top of these concepts



Distributed Systems and relation to other courses in higher education

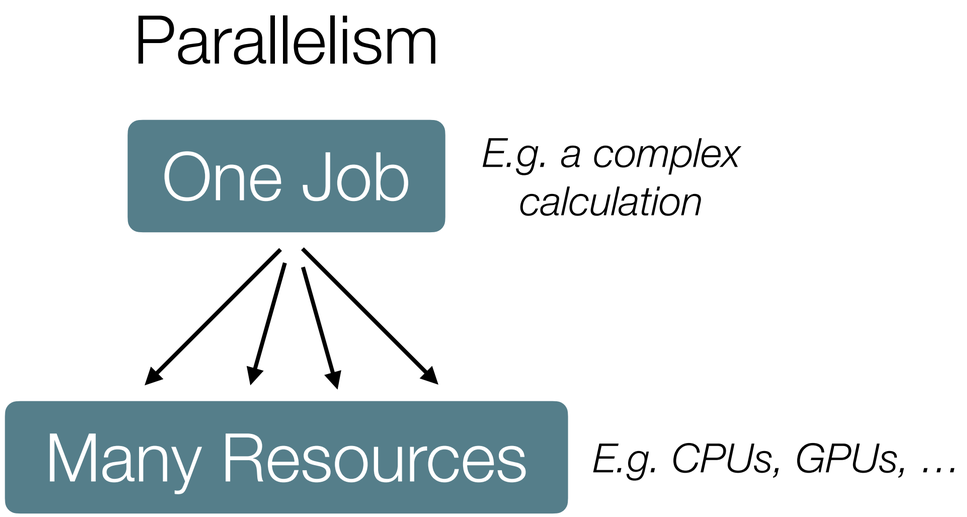
- Distributed Systems as a combination of systems reuse & software creation

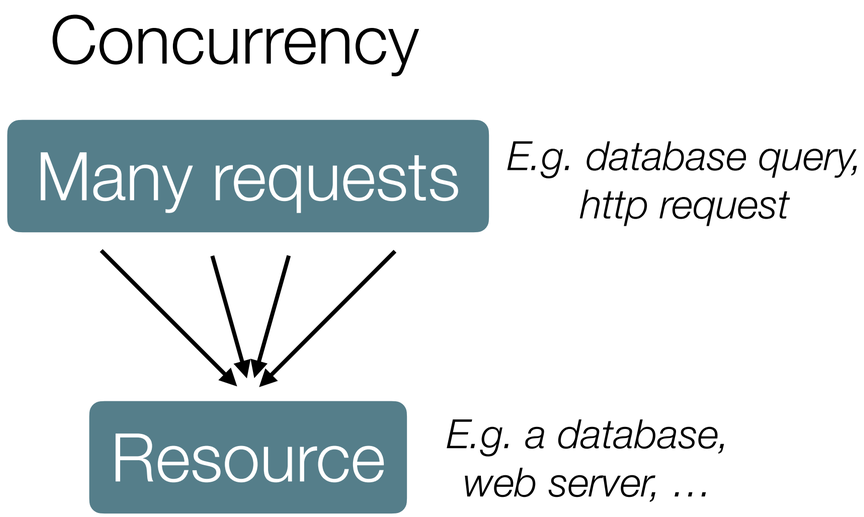


Parallel Systems versus Distributed Systems

Parallelism vs Concurrency

- Note: the terms are used ambiguously in practice, but the perspective is essential (many programmers confuse these concepts)





- There is a connection:
  - It is common to use threads for both
  - If parallel computations need access to shared resources, then the concurrency needs to be managed as well

Parallel versus Distributed Systems

- “**parallel**” hardware:
  - often identical servers / processors
  - regular interconnection structure, sometimes even shared-memory. Fast communication.
- frequent and regular communication patterns
- Homogeneous tasks: each machine performs similar functions
- Clock-synchronised (on a multi-core or multi-processor machine)
- Security: machines in trusted environment
- Failures are often correlated (e.g. entire cluster goes down)

- “**distributed**” hardware:
  - often different types of servers / processors
  - irregular interconnection, networks, no shared memory. Slow (high latency) communication.
- irregular communication patterns
- Heterogeneous tasks: each machine performs different functions
- Clocks are not synchronised
- Security: not all machines are equally trusted
- Failures are often uncorrelated (e.g. network partitions => some servers still reachable, others not)

Challenges in Distributed Systems

The Eight Fallacies of Distributed Computing (L. Peter Deutsch)

- Common false assumptions made by first time distributed systems developers:
  - The network is **reliable**.
  - The network is **secure**.
  - The network is **homogeneous**.
  - The **topology** does not change.
  - **Latency** is zero.
  - **Bandwidth** is infinite.
  - **Transport cost** is zero.
  - There is one **administrator**.

L. Peter Deutsch
(ACM Fellow, Sun Microsystems Fellow)

Challenges (cfr. textbook section 1.5)

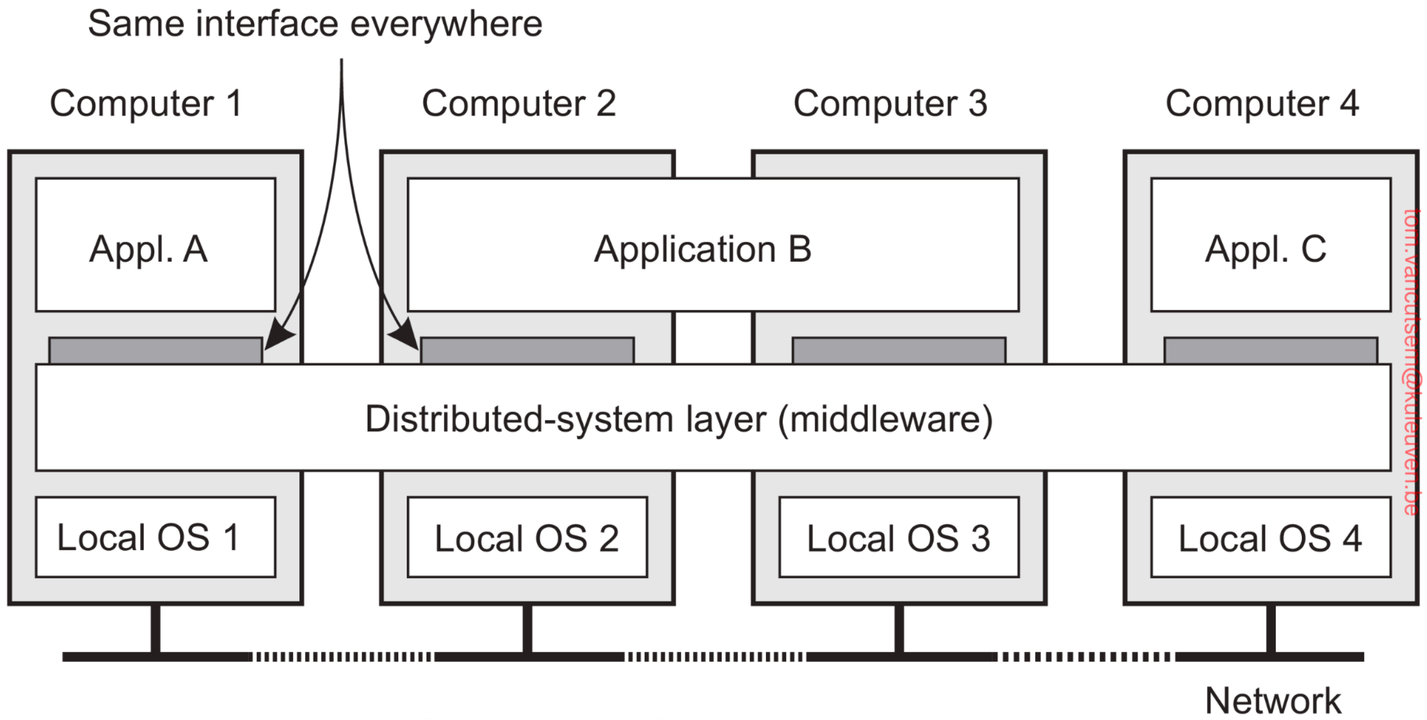
- Heterogeneity
- Openness
- Security
- Scalability
- Failure handling
- Concurrency
- Transparency

Challenges: Heterogeneity

- Heterogeneity exists at many levels:
  - Networks (ethernet, 4G/LTE, bluetooth, …)
  - Computer hardware (server, laptop, desktop, mobile, …)
  - Operating systems (e.g. network programming APIs on Linux vs Windows)
  - Programming languages and runtimes (e.g. Java on JVM, C# on .Net, JavaScript on Node.js)
  - Frameworks and libraries (e.g. Java Servlets vs Spring vs Struts …)
  - Data formats (e.g. XML vs JSON vs …)

Challenges: Heterogeneity => Solutions

- **middleware** provides a uniform high-level API
- Examples: remote method invocation (Java RMI), pub/sub message broker, …



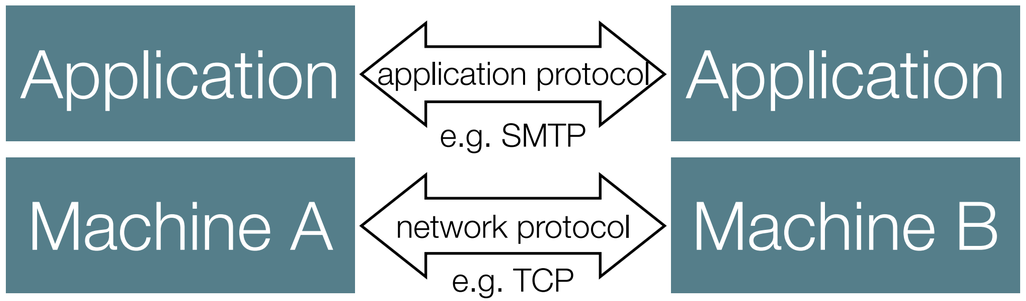
(Image credit: Maarten van Steen & Andrew Tanenbaum, "Distributed Systems", 4th edition)

Challenges: Openness

- Make systems **interoperable**: co-existence of different implementations
- Make systems **portable**: run old systems on top of newer systems
- Make systems **extensible**: easy to add/remove/upgrade existing components with minimal disruption or duplication of existing services

Challenges: Openness => Solutions

- Distributed components with well-defined (preferably open, standardized) interfaces
- Often described in a platform-neutral **Interface Definition Language** (IDL)
- Standardized network and application protocols



Challenges: Security

- Attacks against a system's
  - **confidentiality**: e.g. leaking personal user data, extracting secrets from a server
  - **integrity**: e.g. tampering with server data
  - **availability**: e.g. denial-of-service (make server unavailable by overwhelming it with requests)
- Need to **authenticate** incoming requests: on behalf of what user is the request sent? Attackers can spoof the identity of the sender!
- Need for **access control** on requests: who is allowed to access what resources? (e.g. files, printer, …)

Challenges: Scalability

- With respect to:
  - **Size**: make it easy to add more resources to the system over time, thus accommodating more users
  - **Geography**: users and the resources they access may lie far apart
  - **Administration**: make it easy to manage even if the system spans many independent administrative organizations

Challenges: Scalability => Solutions

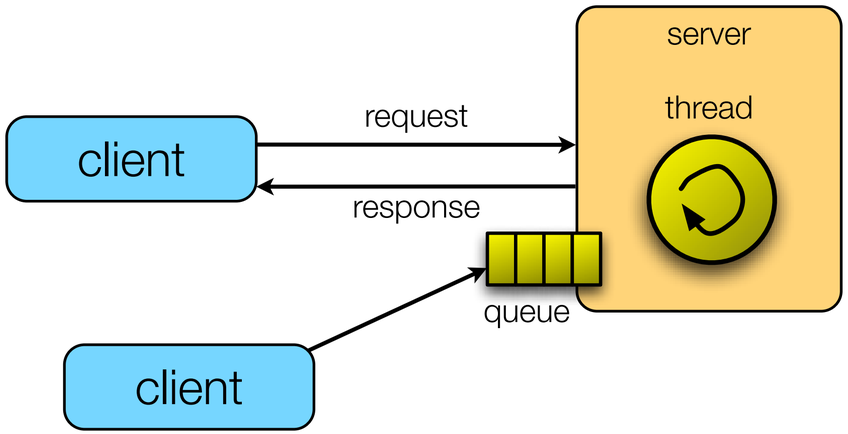
- General scaling techniques:
  - **Caching** of data
  - **Replication** & partitioning of data
  - **Load balancing** of requests across **more servers**
  - **Hierarchical** system organization (Example: Domain Name System organized as a hierarchical tree of servers, each responsible for a specific domain extension)

Challenges: Failure Handling

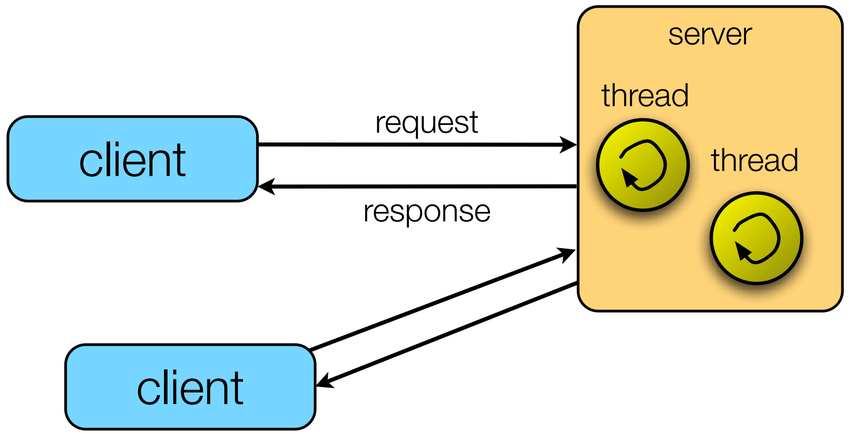
- Recall: distributed systems are subject to **partial** failures: failure is no longer “all-or-nothing”
  - Networks may partition, servers may be unresponsive (crashed or just slow? How would you tell the difference?)
- Some possible mitigations:
  - Detecting failures (e.g. use checksums to check message integrity, use heartbeat messages to detect liveness)
  - Masking failures (e.g. retransmit unacknowledged messages)
  - Tolerating failures (e.g. Google search: if ad recommendation subsystem does not respond, just don’t show any ads)
  - Recovering from failures (e.g. restore a process with data from a checkpoint that was saved to disk)
  - Make (parts of) the system redundant (e.g. replicate services across multiple server machines, e.g. in the Domain Name System store each name record in at least two different servers)

Challenges: Concurrency

- Client-server paradigm: many *client* processes connect to a shared *server* process



Single-threaded server:
serve one client request at-a-time
=> no concurrency, but limited throughput (requests per second)



Multi-threaded server:
serve multiple client requests simultaneously
=> must synchronize concurrent access to server data (e.g. using locks)

Challenges: Transparency

- A system is transparent for a feature if the feature is unobservable for the user
- Transparency can make the system easier to use (uniformity) or easier to cope with changes (adaptive)
- Most important forms of transparency:
  - **Access transparency**: local and remote resources are accessed in the same way (example: file systems may allow reading/writing of local files and remote files using the same API. Modern example: Dropbox file integration.)
  - **Location transparency**: allow resources to be accessed without the need to know their physical or network location (e.g. IP address)

Challenges: Transparency

- Additional forms of transparency:
  - **Concurrency transparency**: avoid interference from concurrent access to a resource (e.g. transactions allow each database client to pretend as if they have exclusive access to the database)
  - **Replication transparency**: allow a resource to be used without having to know whether the resource is replicated or how many replicas exist.
  - **Mobility transparency**: allow resources to be moved without affecting operations of users or programs (e.g. migrating a virtual machine from one server to another in a datacenter, or in the GSM system: allowing a phone to roam from one cell tower to another with uninterrupted service)
- But beware! Too much transparency can hurt system performance or stability. Example: remote file access can be orders of magnitude slower than local file access. Applications that are not designed to handle the difference can become slow or unresponsive.

Summary

- Distributed system = collection of computers or *processes* interconnected by *networks* that communicate using *messages*
  - Inherently concurrent (each process executes independently)
  - No common clock (each process observes time independently)
  - Subject to *partial* failures (each process or network link may fail independently)

Summary

- Challenges
  - Heterogeneity
  - Openness
  - Security
  - Scalability
  - Failure handling
  - Concurrency
  - Transparency

Mostly driven by the need to manage access to **shared resources**

Questions?

Remember: elective courses are available to deepen your knowledge

- G0K31B – Gedistribueerde Software Architecturen – Verdiepende Studie (Big Data & NoSQL)
- H04I0A – Industrial Internet Infrastructure (Sensor networks & IoT)
- H04G7A – Capita Selecta Distributed Systems (Cloud Orchestration & Kubernetes)