



# Introduction to Distributed Systems

## Sistemi Distribuiti e Cloud Computing

A.A. 2024/25

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

Advances in technology and services

Networking

Computing power

Storage

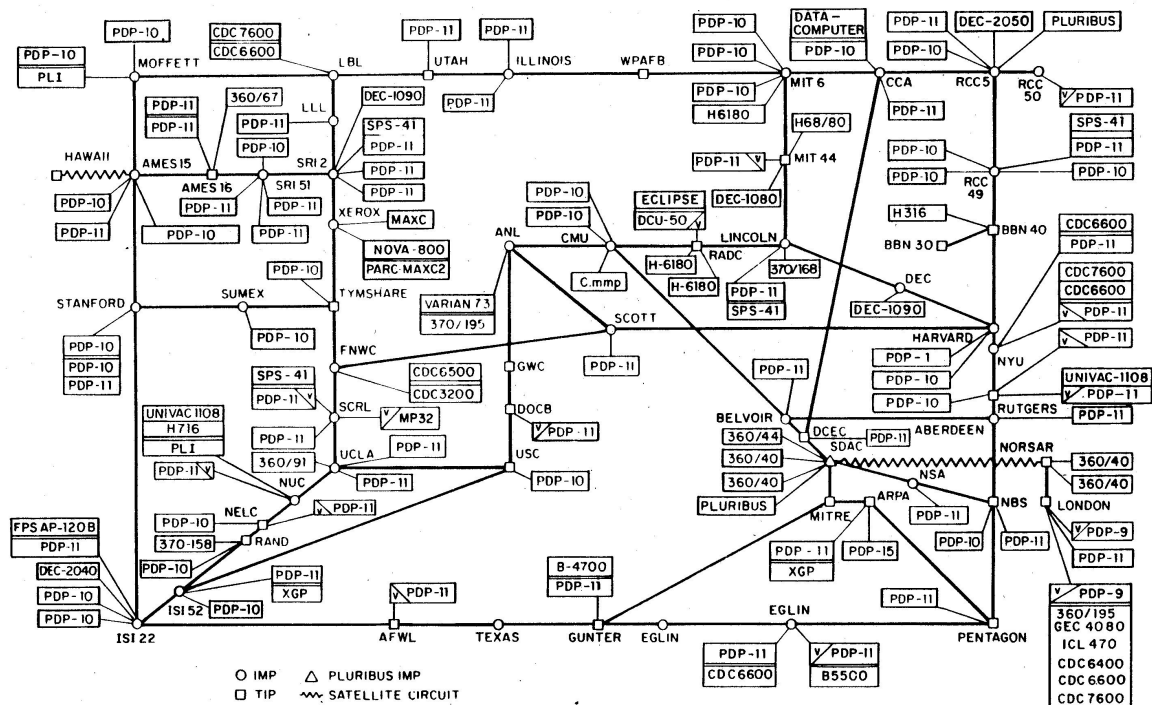
Memory

Protocols

Services and applications

# Internet evolution: 1977

ARPANET LOGICAL MAP, MARCH 1977

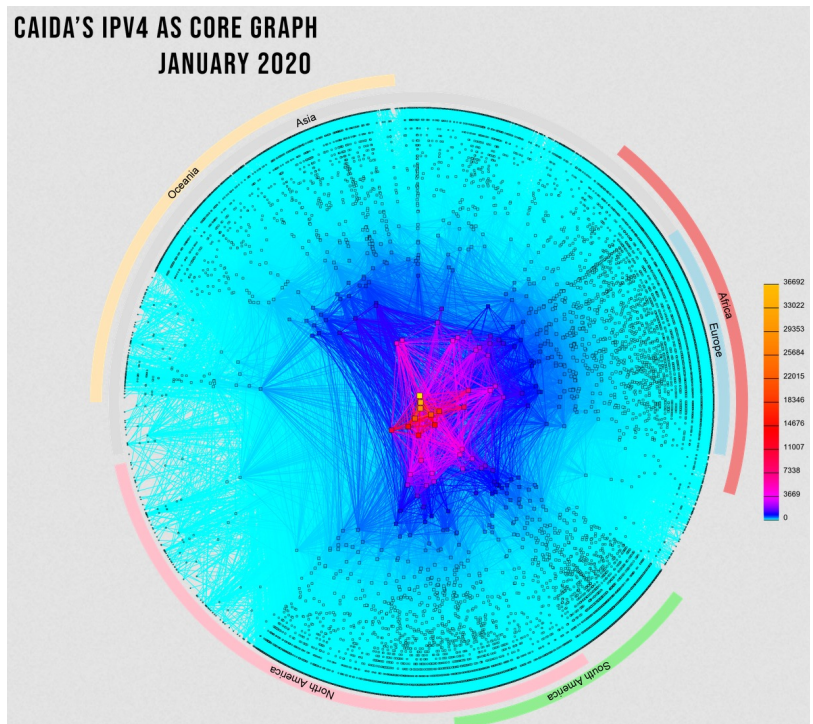


V. Cardellini - SDCC 2024/25

2

# Internet evolution: after 43 years (2020)

- IPv4 AS-level Internet graph
- Interconnections of ~47000 ASs, ~150K links



Source: <https://www.caida.org/projects/as-core/>

# Internet traffic in 2024

	BRAND	Fixed	Mobile
1	Alphabet	18%	20%
2	Meta	11%	22%
3	Netflix	9%	5%
4	Microsoft	7%	2%
5	Tik Tok	6%	12%
6	Apple	6%	5%
7	Amazon	5%	1%
8	Disney	4%	2%
		66%	69%

App Category and App	% of Users	User Volume
<b>Social Media</b>		
Facebook	35%	190 MB
Tik Tok	36%	173 MB
Instagram	35%	961 MB
Snapchat	25%	119 MB
X (Twitter)	16%	71 MB
<b>Communication</b>		
WhatsApp	34%	36 MB
Facebook Messenger	38%	28 MB
Telegram	7%	74 MB
Discord	2%	373 MB
FaceTime	8%	26 MB

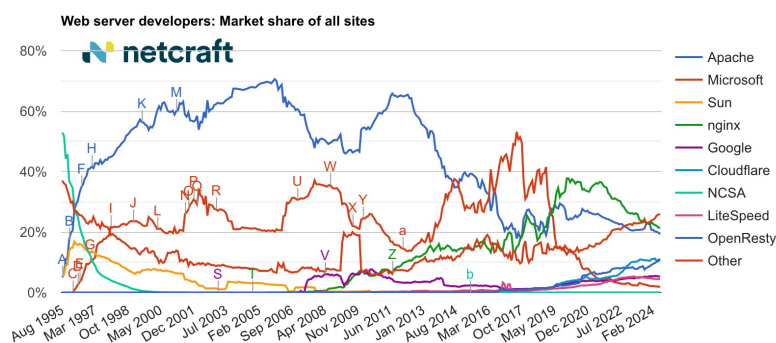
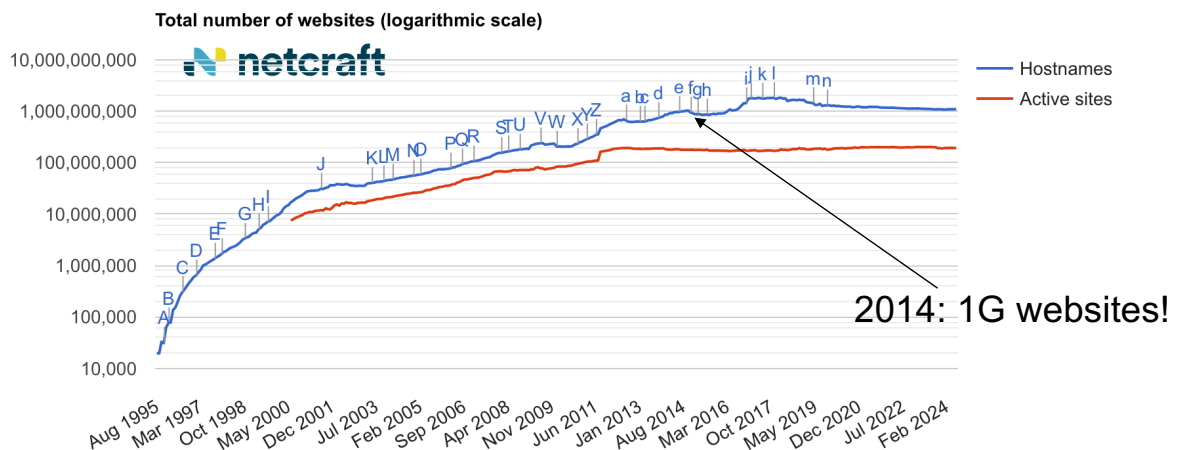
- Volume: 33 EB (22 fixed + 11 mobile) per day
- MAMAAN (Meta, Alphabet, Microsoft, Amazon, Alphabet and Netflix) generated 55% of Internet volume
- Video contributes 38% of downstream traffic
- Social media is the biggest app category in mobile networks' downstream and upstream traffic
- Expanding number of app categories and larger number of apps, which are producing more data

Source: <https://www.sandvine.com/phenomena>

V. Cardellini - SDCC 2024/25

4

## Web growth



V. Cardellini - SDCC 2024/25

5

# Metcalfe's law

---

*“The value of a telecommunications network is proportional to the square of the number of connected users of the system”.*

➔ Networking is *socially* and *economically* interesting

facebook

Google

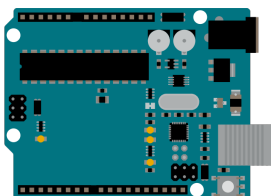
Instagram

TikTok

# Computing power

---

- Computers got...
  - Faster
  - Cheaper
  - Power efficient
  - Smaller
- 1974: Intel 8080
  - 2 MHz, 6K transistors
- 2004: Intel P4 Prescott
  - 3.6 GHz, 125 million transistors
- 2011: Intel 10-core Xeon Westmere-EX (**multicore CPUs**)
  - 3.33 GHz, 2.6 billion transistors
- 2019: NVIDIA Turing GPU
  - 14.2 TFLOPS of peak single precision (FP32) performance



Arduino UNO: weight=25 g, width=53.4 mm, length=68.6 mm

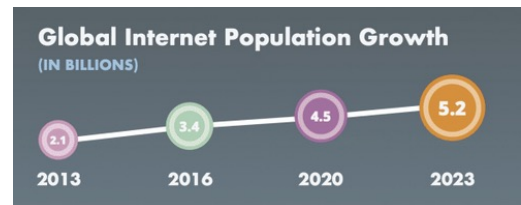
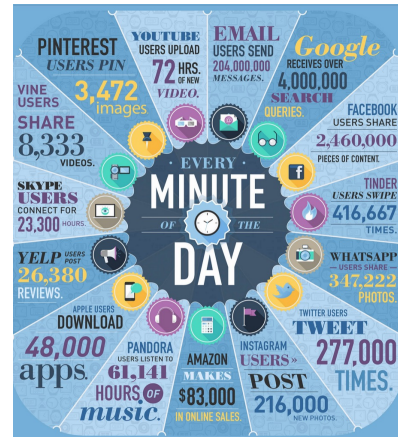


# Every minute in Internet

- Huge data volume (2023)
- Service evolution (how it was in 2014)



<https://www.domo.com/learn/infographic/data-never-sleeps-11>



V. Cardellini - SDCC 2024/25

8

## The challenge: scale of Internet services

- How to provide and manage these services?
- It is a problem of scale and scale changes everything!



Google in 1997

**Scale** has been the single most important force driving changes in system software over the last decade, and this trend will probably continue for the next decade.

Ousterhout, Is scale your enemy, or is scale your friend?: technical perspective, *Comm. ACM*, 2011

<https://dl.acm.org/doi/pdf/10.1145/1965724.1965748>

V. Cardellini - SDCC 2024/25

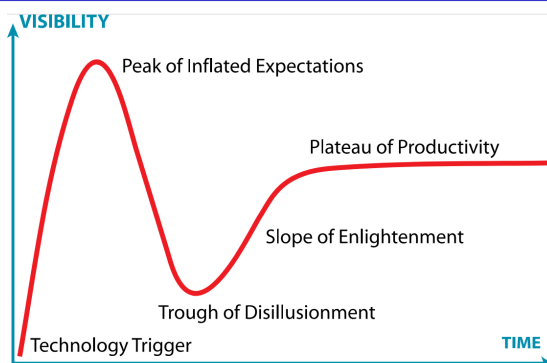
9

# Distributed systems: not only Internet and Web

- Internet and Web: two notable examples of distributed systems
- Other examples:
  - Cloud systems, HPC systems, ... sometimes accessible only through private networks
  - Peer-to-peer systems
  - Home networks (home entertainment, multimedia sharing)
  - Internet of Things (IoT)



## Gartner's IT hype cycle for emerging technologies

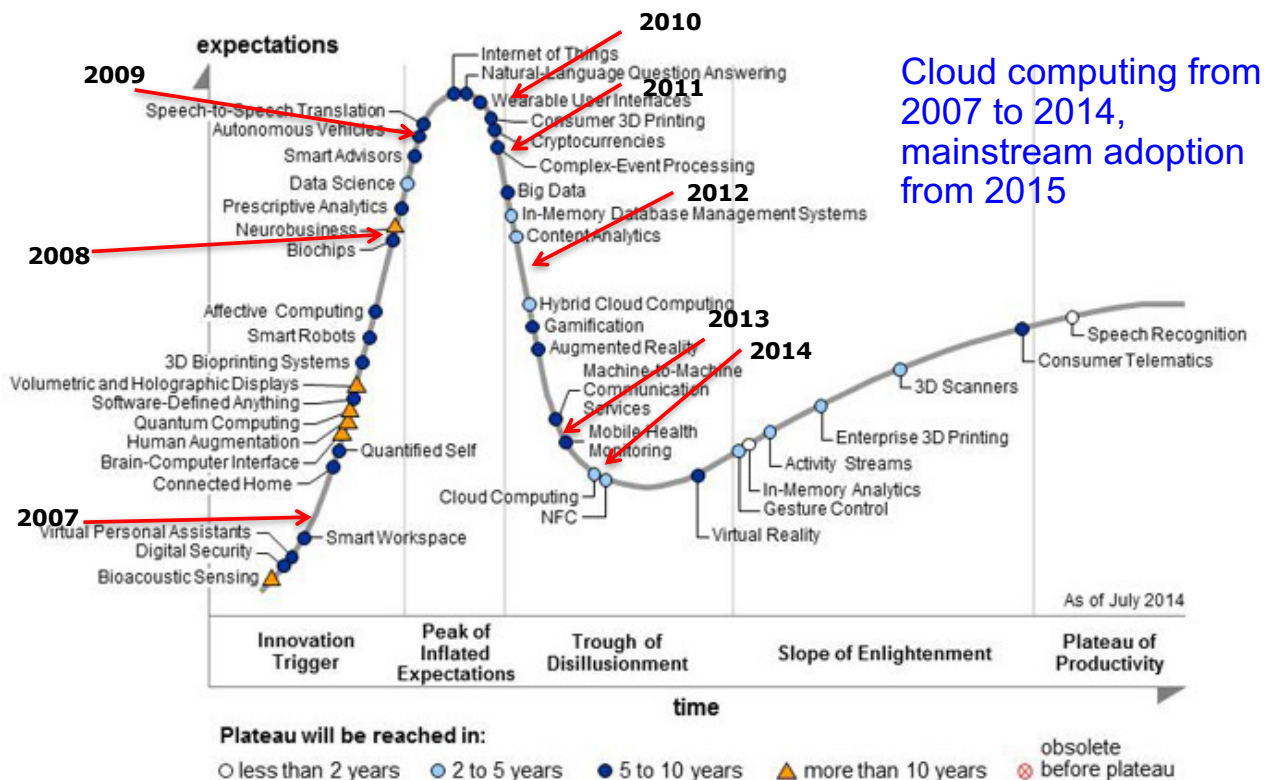


Graphical presentation developed by IT firm Gartner to represent maturity, adoption, and application of specific technologies

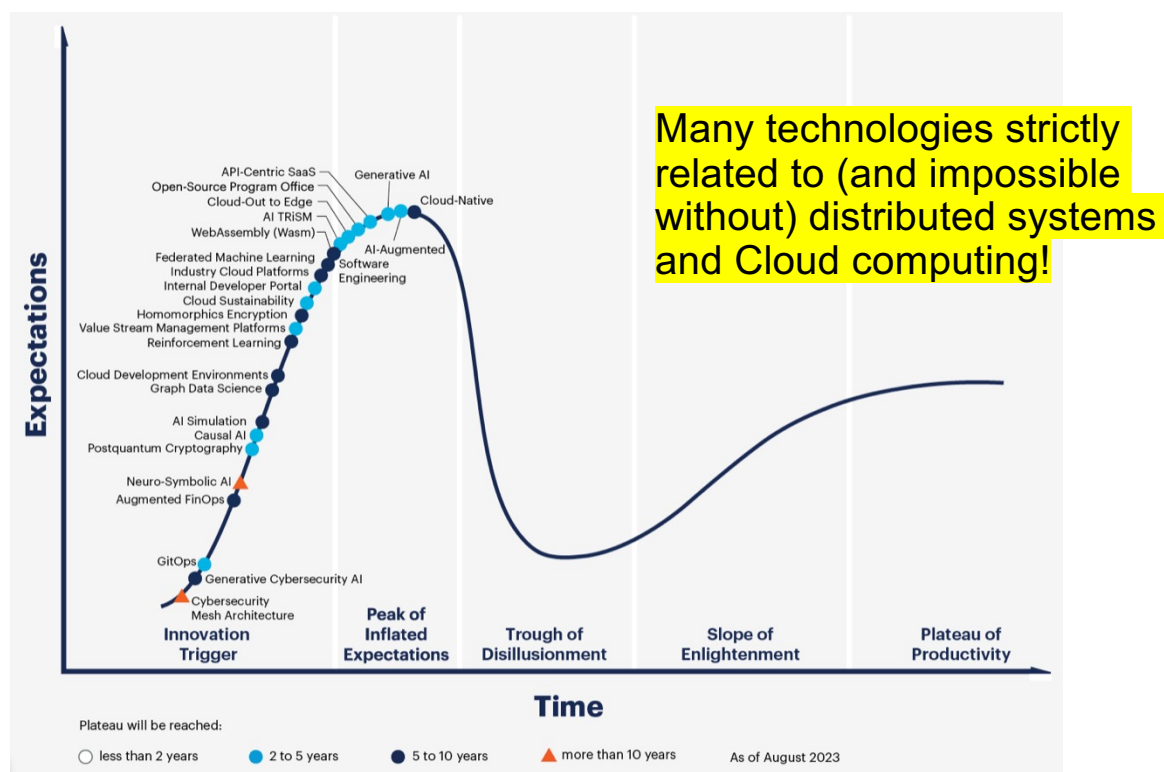
5 phases:

- 1) Innovation trigger: a new technology is introduced
- 2) Peak of inflated expectations: hype tends to be overhyped, with high and often unrealistic expectations about technology impact
- 3) Trough of disillusionment: interest wanes as reality sets in, possible decline in investment
- 4) Slope of enlightenment: more practical applications start to be found, with a more balanced and clearer understanding of its potential
- 5) Plateau of productivity: technology widely adopted, reaches its full potential, becoming a mainstream tool

# Gartner's IT hype cycle for emerging technologies and cloud computing



# Gartner's hype cycle for emerging technologies in 2023

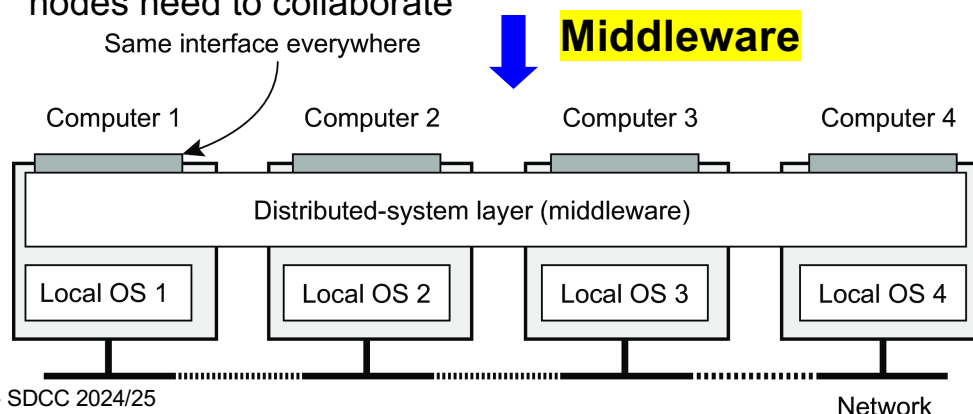


# Distributed systems, Cloud and AI

- Artificial Intelligence (AI) has recently become practical as result of:
  - Distributed computing
  - Affordable cloud computing and storage costs
  - Examples: federated learning, distributed training of foundation models (huge computational and data storage needs)
- Distribute = to divide and dispense in portions
- A foremost strategy used in distributed computing you already know
  - **Divide et impera**: break larger (computational) problems down into numbers of smaller, interrelated, “manageable” pieces

## Distributed system

- Multiple definitions of **distributed system (DS)**  
[van Steen & Tanenbaum] A distributed system is a collection of **autonomous computing elements** that appears to its users as a **single coherent system**
  - Consists of autonomous computing elements (i.e., **nodes**), can be hardware devices (computer, phone, car, robot, ...) or software processes
  - Users or applications perceive it as a **single system (how?)**: nodes need to collaborate



# Distributed system

---

[Lamport] A distributed system is one in which the **failure** of a computer you didn't even know existed can render your own computer unusable

- Emphasis on fault tolerance
- Who is Leslie Lamport?
  - Recipient of 2013 Turing award  
<https://www.youtube.com/watch?v=pgWTmOyUjtM>
  - His research contributions have laid the foundations of theory and practice of DS
    - Fundamental concepts such as **causality**, **logical clocks** and **Byzantine failures**
    - Algorithms to solve many fundamental problems in DS
  - Inventor of LaTeX

## Why make a system distributed?

---

- Share **resources**
  - Resource = computing node, data, storage, service, ...
- Lower **costs**
- Improve **performance**
  - e.g., get data from a nearby node rather than one halfway round the world
- Improve **availability** and **reliability**
  - even if one node fails, the system as a whole keeps functioning
- Improve **security**
- Solve bigger problems
  - e.g., huge amounts of data, can't fit on one machine
- Support Quality of Service (**QoS**) and Quality of Experience (**QoE**)



# Why to study distributed systems

---

- Distributed systems are **more complex** than centralized ones

- e.g., no global clock, group membership, ...

- Building them is **harder**... and building them correct is even much harder

*Distributed systems need radically different software than centralized systems do (Tanenbaum)*

- Managing, and, above all, testing them is **difficult**

## Some distinguishing features of DS

---

- **Concurrency**

- Many things happen “at the same time”
  - Centralized system: design choice
  - Distributed system: fact of life to be dealt with

- **No global clock**

- Centralized system: use computer’s physical clock for synchronization
  - Distributed system: many physical clocks and not necessarily synchronized among them

- **Independent and partial failures**

- Centralized system: fails completely
  - Distributed system: fails partially (i.e., only a part), often due to communication; hard (and in general impossible) to hide partial failures and their recovery

# Challenges and design goals

---

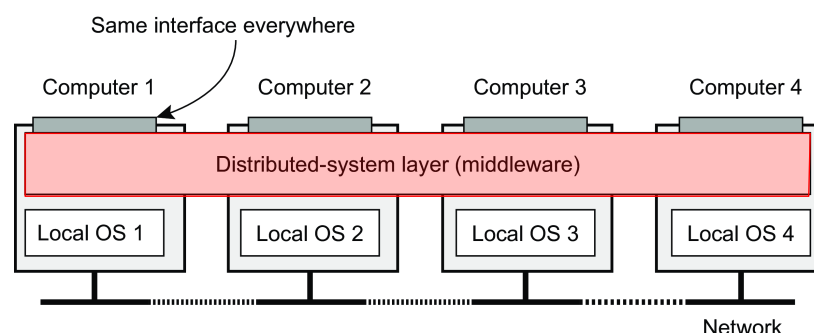
- Challenges and goals associated with designing distributed systems
  - Heterogeneity**
  - Distribution transparency**
  - Openness**
  - Scalability**
  - Dependability**
  - Security**

while improving **performance** and **energy efficiency**, reducing monetary cost, etc.

## Challenge 1: Heterogeneity

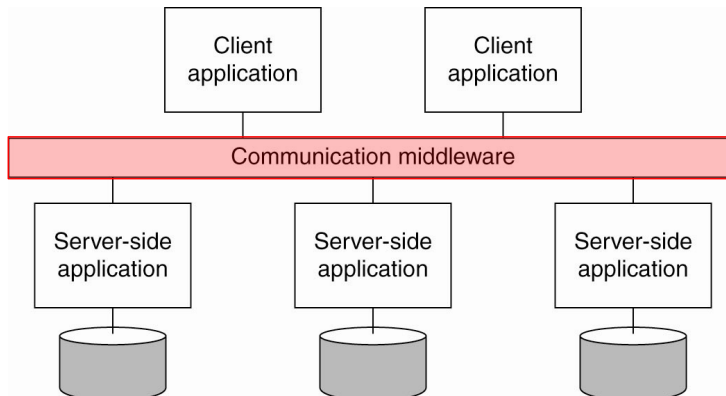
---

- Many sources of heterogeneity: network, hardware, operating system (OS), programming language, implementations by different developers
- How to address? **Middleware**: the “OS of a DS”
  - Sw layer placed on top of OS that provides a **programming abstraction** as well as **masks heterogeneity**
  - Contains commonly used components and functionalities (e.g., communication) thus avoiding developers to implement them again and from scratch



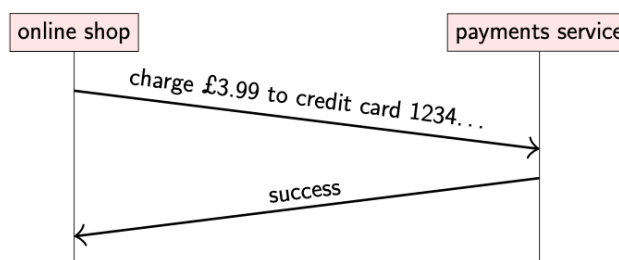
# Communication middleware

- Facilitates communication among (heterogeneous) DS components/apps
- We will study
  - Remote Procedure Call (RPC)
  - Message Oriented Middleware (MOM)



## Remote Procedure Call (RPC) example

- Online payment



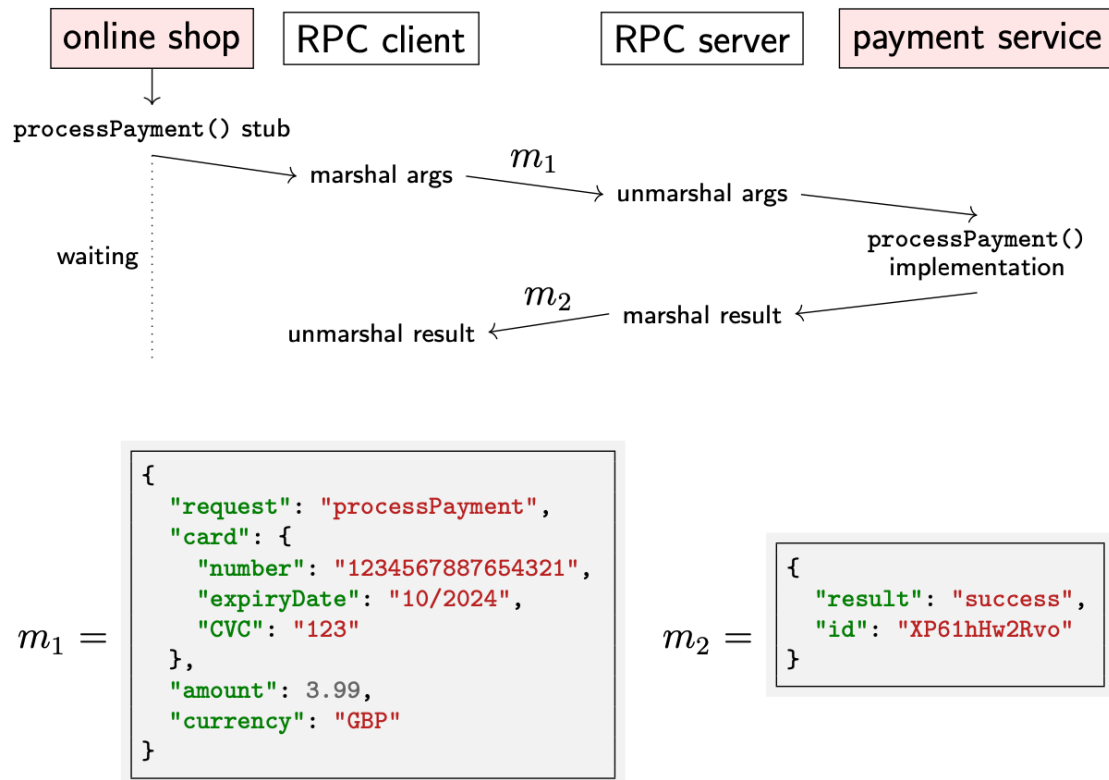
```
// Online shop handling customer's card details
Card card = new Card();
card.setCardNumber("1234 5678 8765 4321");
card.setExpiryDate("10/2024");
card.setCVC("123");

Result result = paymentsService.processPayment(card,
    3.99, Currency.GBP);

if (result.isSuccess()) {
    fulfilOrder();
}
```

Implementation of this function is on another node!

# RPC: Behind the curtains



## Challenge 2: Distribution transparency

- **Distribution transparency:** single coherent system where the distribution of its *objects* (processes and resources) is *transparent* (i.e., invisible) to users and apps
- **Types** of distribution transparency (ISO 10746, Reference Model of Open Distributed Processing)

### Access transparency

- Hide differences in data representation and how objects are accessed
  - e.g., use same mechanism for local or remote call

### Location transparency

- Hide where objects are located
  - e.g., URL hides IP address

### Relocation transparency

- Hide that objects may be moved to another location *while in use*

## Challenge 2: Distribution transparency

---

### Migration transparency

- Hide that objects may move to another location
  - e.g., communication between mobile phones

### Replication transparency

- Hide that multiple replicas of an object exist
  - How? Same name for all replicas
  - Example: type command `dig www.youtube.com`
  - Require also location transparency

### Concurrency transparency

- Hide that objects may be shared by several independent users
  - E.g.: concurrent access to same DB table by multiple users
  - Issue: leave shared object in a consistent state, e.g., by *locking* mechanisms

### Failure transparency

- Hide failure and recovery of objects (see Lamport's definition)

## Degree of distribution transparency

---

- Aiming at *full* distribution transparency may be too much
  - We cannot always hide *communication latency*: sending a message from Roma to New York requires ~23 ms
  - We cannot completely hide *failures* in a large-scale DS
    - Cannot distinguish a slow computer from a failing one
    - Cannot be sure that a server actually performed an operation before crashing
  - Price for achieving full transparency may be too high in term of *performance*
    - e.g., keeping data replicas *exactly* up-to-date *takes time*
    - e.g., *immediately flushing* write operations to disk for fault tolerance *affects performance*
    - **Trade-off** between consistency and performance



## Challenge 3: Openness

---

- Open DS: offers components that can easily be used by or integrated into other systems; consists of components that originate from elsewhere
- Systems should conform to well-defined **interfaces**
  - Defined through IDL (**Interface Definition Language**)
    - Nearly always capture only syntax, not semantics
    - **Complete** and **neutral**
    - IDL examples: XDR, Thrift, WSDL, OMG IDL
- Systems should
  - easily **interoperate**
  - support **portability** of applications
  - be easily **extensible**
- Examples: Jakarta EE, .Net, Web Services

*Practice shows that many distributed systems are not as open as we'd like (van Steen & Tanenbaum)*

## Separation of policies and mechanisms

---

- To implement open and flexible DS, we need to organize the DS as a collection of relatively small and easily replaceable or adaptable component rather than as a monolithic system
- How? **Separate policies from mechanisms**
  - A policy refers to the principles guiding the decisions, while mechanisms represent the means to implement policies
- E.g., caching in web browsers:
  - Mechanism: store data and allow (dynamic) setting of caching policies
  - Caching policies:
    - Where to cache data?
    - How to free space when cache fills up?
    - When to refresh cached data?
    - Private or shared cache?

# Separation of policies and mechanisms

---

- The other side of the coin
  - Strict separation can be counterproductive: the stricter the separation between policy and mechanism, the more we need to ensure proper mechanisms, potentially leading to many configuration parameters and complex management
- Need to find a balance
- Possible solution: *self-configurable systems*

*Finding the right balance in separating policies from mechanisms is one of the reasons why designing a distributed system is sometimes more an art than a science (van Steen & Tanenbaum)*

## Challenge 4: Scalability

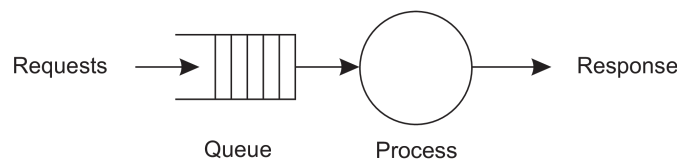
---

- **Scalability** is the property of a (distributed) *system* to keep an adequate level of performance notwithstanding a growing amount of:
  - Number of users and resources (**size** scalability)
  - Maximum distance between nodes (**geographical** scalability)
  - Number of administrative domains (**administrative** scalability)
- Most systems account only, to a certain extent, for size scalability

*Many developers of modern distributed systems easily use the adjective scalable without making clear why their system actually scales. (van Steen)*

# Size scalability

- Root causes for scalability problems in centralized system
  - Computational capacity, limited by CPUs
  - Storage capacity, including transfer rate between CPUs and disks
  - Network between user and centralized service
- Formal analysis (see PMCS course)

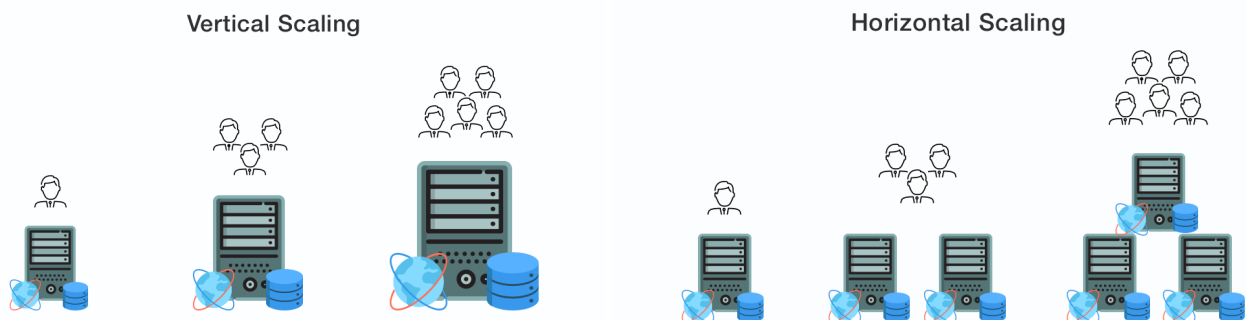


$$\frac{R}{S} = \frac{1}{1 - U}$$

- Service time  $S$
  - Utilization  $U$ : fraction of time the service is busy
  - Response time  $R$ : total time take to process a request its arrival
  - If  $U$  is small, response-to-service time is close to 1: request is immediately processed
  - If  $U$  goes up to 1, system comes to a grinding halt.
- Solution: decrease  $S$ . **How?**

# Size scalability

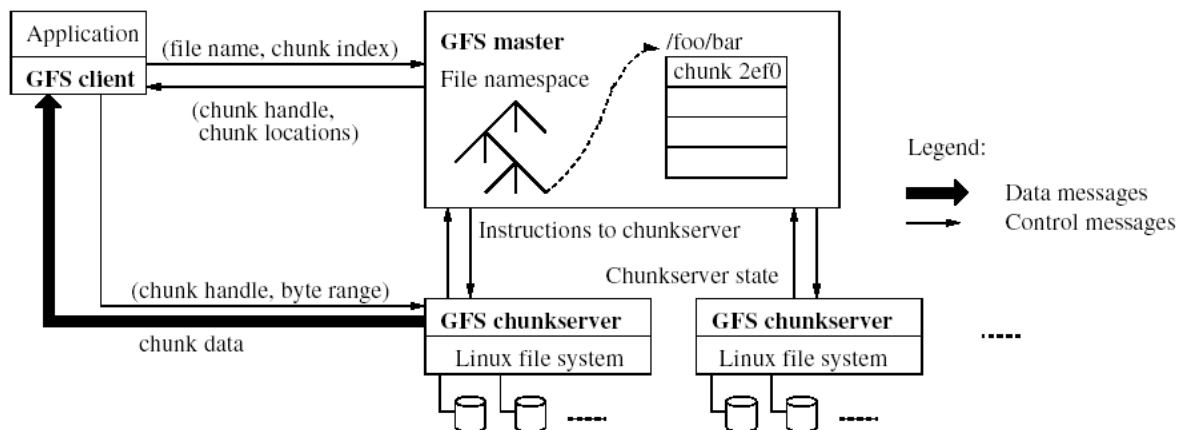
- Two directions for size scalability
  - **Vertical (scale-up)**: more powerful resources
  - **Horizontal (scale-out)**: more resources with same capacity



- Which resources? Physical, virtual (virtual machines, software containers, etc.)

# Size scalability: GFS example

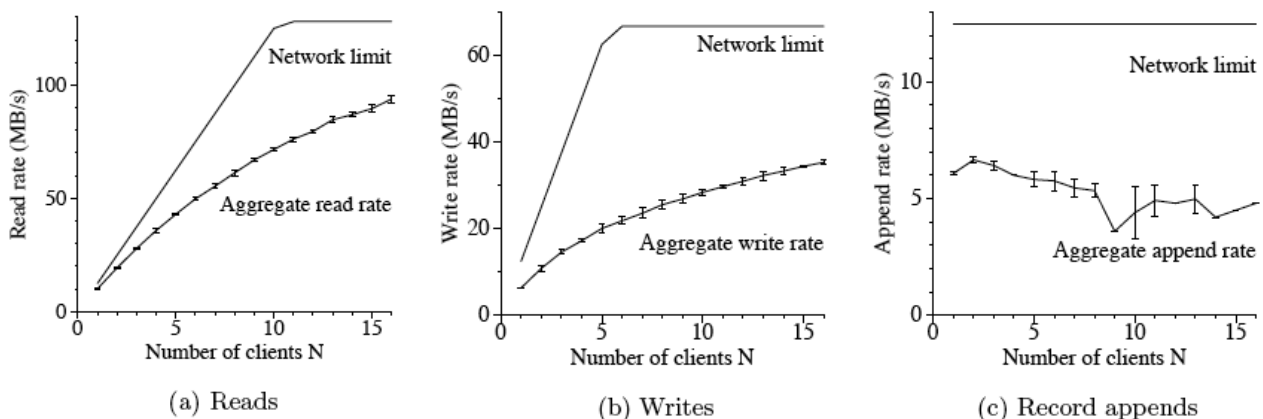
- Google File System (GFS)
  - Distributed file system based on divide et impera principle and horizontal scalability
  - Read, write, append operations
  - File divided into fixed blocks (**chunks**), which are distributed and replicated on multiple servers
  - Single **master**, multiple **workers** (chunkservers)



V. Cardellini - SDCC 2024/25

34

# Size scalability: GFS example



- See experimental results
  - **Scale parameter**: number of clients
  - **Scalability metric**: aggregated read/write/append throughput (random file access)
  - **Scalability criterion**: the closer to network limit, the better
- Scalability is challenging!

V. Cardellini - SDCC 2024/25

35

# Techniques for scaling

---

## 1. Hide communication latency

- Make use of **asynchronous communication**
- Have separate handler for incoming response
- Problem: not every app fits this model

## 2. Facilitate solution by moving computations to clients

## 3. Partition data and computation across multiple resources

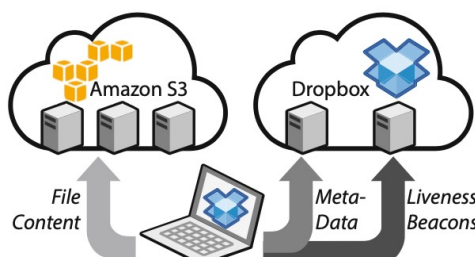
- **Divide et impera**: partition data and computation into smaller parts and distribute them across multiple DS resources
- E.g.: decentralized naming service (DNS), data-intensive distributed computation (Hadoop MapReduce and Spark)

# Techniques for scaling

---

## 4. Replicate DS resources and data

- Make copies of data available at different DS nodes
- Distribute processing on multiple DS nodes
- Examples:
  - Distributed file systems and databases
  - Replicated Web servers
  - Web caches (in browsers and proxies)
- Practical example: in a cloud storage service (e.g., Dropbox, OneDrive, GDrive) data are locally cached on your device and replicated across multiple cloud servers





# The problem with replication

---

- Applying replication is easy, but
- Having multiple copies leads to **inconsistency**: modifying one copy makes that copy different from the rest
- Trade-off: depending on application type, a certain degree of **inconsistency can be tolerated**
  - Blog, shared file, shopping cart, on-line auction, air traffic control
- We will study different **consistency models** to choose from

## Challenge 5: Dependability

---

- **Dependability** refers to the degree that a computer system can be relied upon to operate as expected
  - **partial failures** make it intricate for distributed systems
- Requirements related to dependability
  - **Availability**: readiness for usage
  - **Reliability**: continuity of service delivery
  - **Safety**: very low probability of catastrophic consequences
  - **Maintainability**: how easily a failed system can be repaired

# Dependability: availability

## Can I use the system now?

- System is ready to use (i.e., operational) immediately
  - Availability  $A(t)$  of component C: probability that C is functioning correctly at time  $t$
- Availability = uptime / (uptime + downtime)
  - Normally expressed as number of 9's
    - $A = 99\%$ : two nines  
downtime per year =  $0.01 * 365.2425 \text{ d} = 3 \text{ d } 15 \text{ h } 39 \text{ m } 29.5 \text{ s}$   
*1-A=1% average days per year*
    - $A = 99.99\%$ : four nines  
downtime per year =  $0.0001 * 365.2425 \text{ d} = 52 \text{ m } 35.7 \text{ s}$
    - See [uptime.is](https://uptime.is)

# Dependability: availability

## Metrics

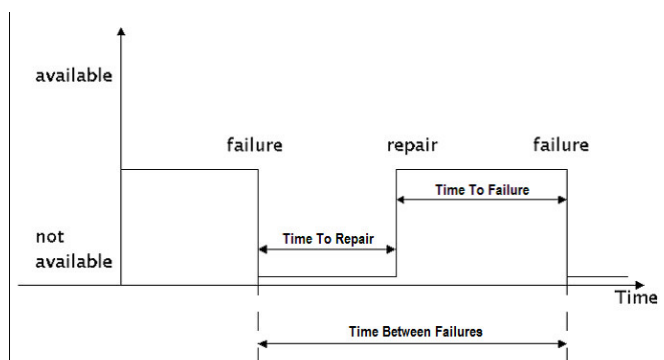
$$A = \text{MTTF} / (\text{MTTF} + \text{MTTR})$$

**Mean Time To Failure (MTTF)**: average time until a component fails

**Mean Time To Repair (MTTR)**: average time needed to repair a component

**Mean Time Between Failures (MTBF)**:  $\text{MTTF} + \text{MTTR}$

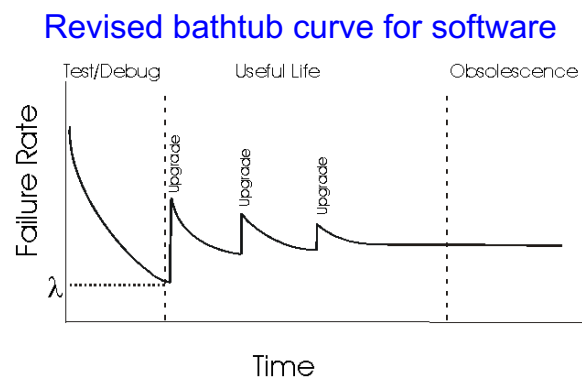
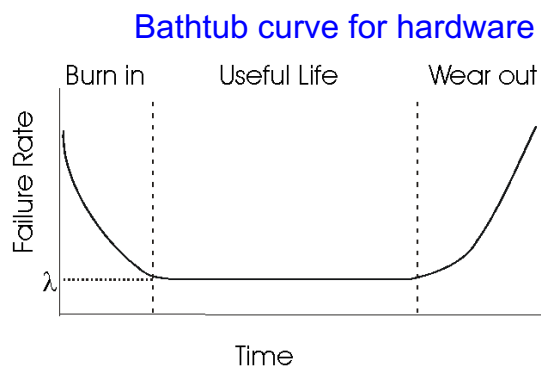
$\text{MTBF} = \text{total operating time} / \text{number of failures}$



# Dependability: reliability

*Will the system be up as long as I need it?*

- System will run continuously without failure
- Reliability  $R(t)$  of component  $C$ : conditional probability that  $C$  has been functioning correctly during  $[0,t)$  given  $C$  was functioning correctly at the time  $T = 0$
- Metrics: MTTF (and failure rate)



# Availability vs reliability

- **Availability  $\neq$  reliability** (when the system is repairable)
- Example A: system that goes down for 1 ms every hour
  - Highly available:  $> 99,9999\%$  ( $= 1 - 1/(3600 \cdot 1000)$ )
  - Unreliable, because MTBF = 1 hour and there are  $24 \cdot 365 = 8780$  failures per year
- Example B: system that never crashes but is shutdown for 2 weeks every year
  - Highly reliable, because MTBF = 1 year and there is only 1 failure per year
  - But only 96% available ( $= 1 - 14/365$ )

# Failure, error and fault

---

- **Failure**: a component is not living up to its specifications
  - Example: crashed program
- **Error**: part of a component that can lead to a failure
  - Example: programming bug
- **Fault**: cause of an error
  - Can be: transient, intermittent, permanent
  - Example: sloppy programmer

Chain **fault** → **error** → **failure**

# Dependability: tools

---

- **Fault prevention**
  - Prevent fault occurrence
- **Fault tolerance**
  - Build component and make it mask fault occurrence
- **Fault removal**
  - Reduce presence, number, or seriousness of faults
- **Fault forecasting**
  - Estimate current presence, future incidence, and consequences of faults

## Challenge 6: Security

---

- A distributed system that is not secure, is not dependable
- What we need
  - **Confidentiality**: information is disclosed only to authorized parties
  - **Integrity**: ensure that alterations to system assets can be made only in an authorized way
- Authorization, authentication, trust
  - **Authentication**: verifying correctness of claimed identity
  - **Authorization**: does an identified entity has proper access rights?
  - **Trust**: one entity can be assured that another will perform particular actions according to a specific expectation

## Security mechanisms

---

- Keeping it simple: it is all about encrypting and decrypting data using security keys
- **Symmetric** vs **asymmetric** cryptosystem
  - Symmetric: same encryption and decryption key, both keys should be kept secret
  - Asymmetric: public key and private key
- **Secure hashing**
  - In practice, we use **secure hash functions**:  $H(data)$  returns a fixed-length string
    - Any change from  $data$  to  $data^*$  will lead to a completely different string  $H(data^*)$
    - Given a hash value  $h$ , it is computationally impossible to find a  $data$  with  $h = H(data)$

# Two categories of distributed systems

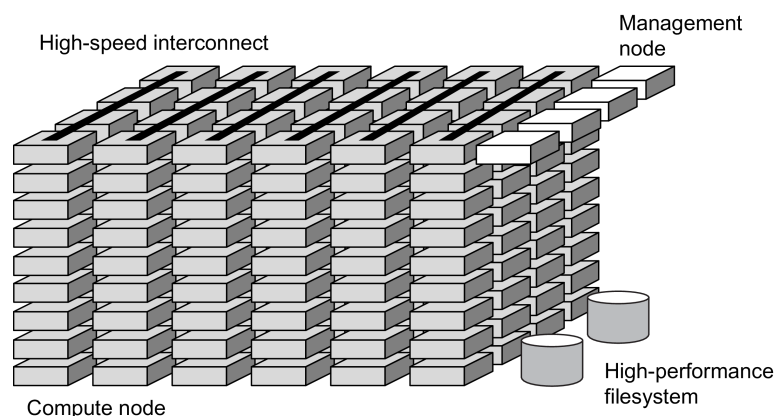
---

- High-performance distributed computing systems
  - Cluster computing
  - Cloud computing
  - Edge computing
- Distributed pervasive systems

## Cluster computing

---

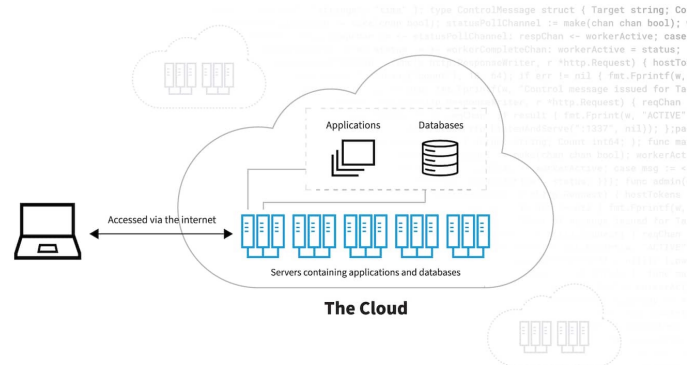
- Cluster: group of high-end systems connected through a LAN
  - Typically **homogeneous**: same OS, near-identical hardware
  - Single, or tightly coupled managing node(s)



- Clusters dominate TOP500 architectures  
[www.top500.org](http://www.top500.org)

# Cloud computing

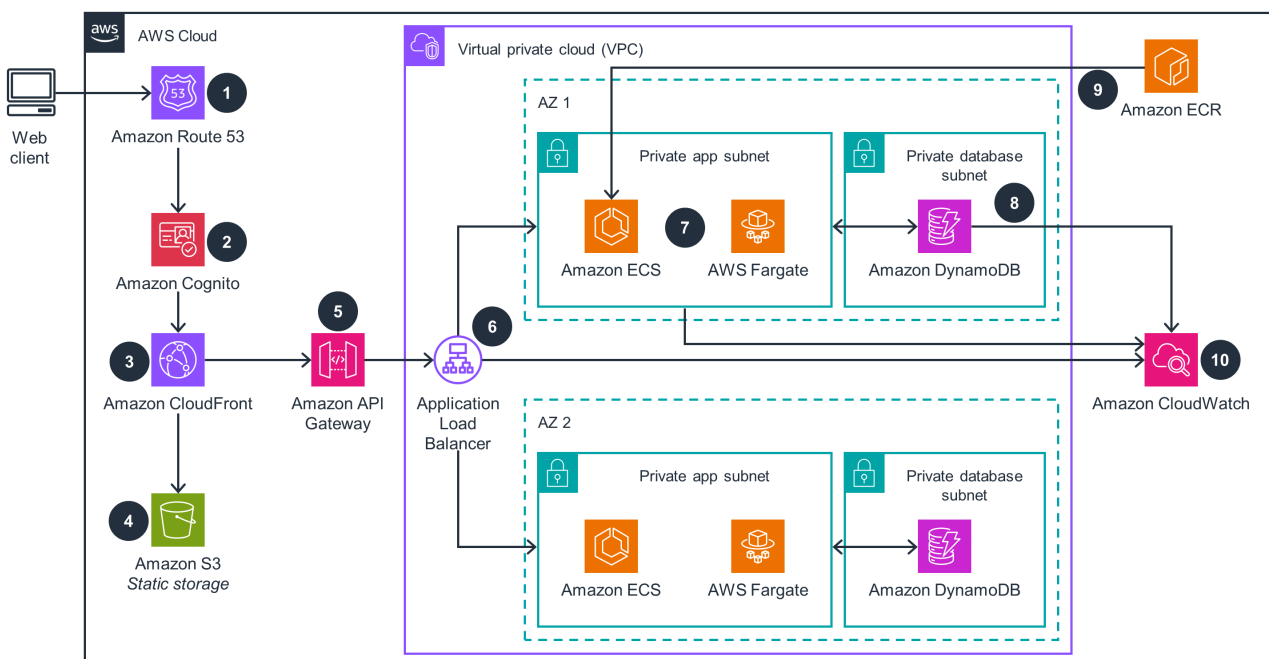
- Cluster computing is a major milestone that lead to Cloud computing
- But Cloud is:
  - available to anyone
  - on a much wider scale
  - does not require users to physically own or use hardware



V. Cardellini - SDCC 2024/25

50

## A distributed app in the Cloud



V. Cardellini - SDCC 2024/25

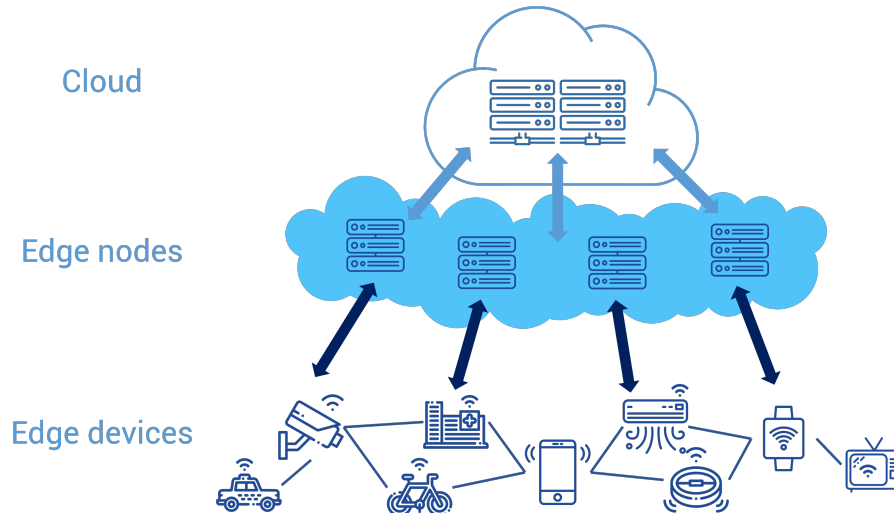
51



# Edge computing

---

- Brings computation and storage at the **network edges**, in proximity of data producers (e.g., IoT devices) and consumers (e.g., users)



V. Cardellini - SDCC 2024/25

52

# Distributed pervasive systems

---

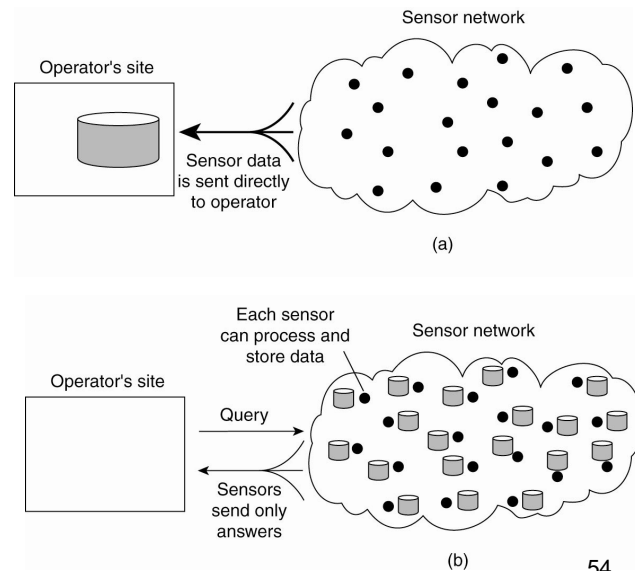
- Distributed systems whose nodes are
  - **small, mobile, battery-powered** and often **embedded** in a larger system
  - characterized by the fact that the system **naturally blends into the user's environment**
- Three (overlapping) subtypes of pervasive systems
  - **Ubiquitous computing systems**: pervasive and **continuously present**, i.e. continuous interaction between system and users
  - **Mobile computing systems**: pervasive, with emphasis on the fact that devices are **inherently mobile**
  - **Sensor networks**: pervasive, with emphasis on the actual (collaborative) **sensing** and **actuation** of the environment

V. Cardellini - SDCC 2024/25

53

# Sensor networks

- Characteristics of nodes
  - Many:  $10-10^3$
  - Simple: small memory, compute and communication capacity
  - Often battery-powered (or even battery-less)
- Sensor networks as distributed databases: two extremes
  - (a) Store and process data in a **centralized** way only on the **sink** node
  - (b) Store and process data in a **distributed** way on the sensors (active and autonomous)



V. Cardellini - SDCC 2024/25

54

## Pitfalls in realizing distributed systems

- Many distributed systems are needlessly complex because of errors in design and implementation that were patched later
- Common wrong assumptions by architects and designers of distributed systems ("The Eight Fallacies of Distributed Computing", Peter Deutsch, 1991-92):
  1. The network is reliable
    - "You have to design distributed systems with the expectation of failure" (Ken Arnold)
  2. Latency is zero
    - Latency is more problematic than bandwidth
    - "At roughly 300,000 km/s, it will always take at least 30 ms to send a ping from Europe to the US and back, even if the processing would be done in real time." (Ingo Rammer)
  3. Bandwidth is infinite

V. Cardellini - SDCC 2024/25

55

# Pitfalls in realizing distributed systems

---

4. The network is secure
5. The topology does not change
  - That's right, it doesn't--as long as it stays in the test lab!
6. There is one administrator
7. Transport cost is zero
  - Going from the application level to the transport level is not free
  - Costs for setting and running the network are not free
8. The network is homogeneous

Technology is not the solution to everything!

## References

---

- Chapter 1 of van Steen & Tanenbaum book
- A brief introduction to distributed systems <https://www.distributed-systems.net/my-data/papers/2016.computing.pdf>
- Fallacies of distributed computing explained <https://arnon.me/wp-content/uploads/Files/fallacies.pdf>
- Episode 470: L. Peter Deutsch on the fallacies of distributed computing <https://www.se-radio.net/2021/07/episode-470-l-peter-deutsch-on-the-fallacies-of-distributed-computing/>