TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Introduction to Distributed Systems

Corso di Sistemi Distribuiti e Cloud Computing

A.A. 2021/22

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

Technology advances

Networking

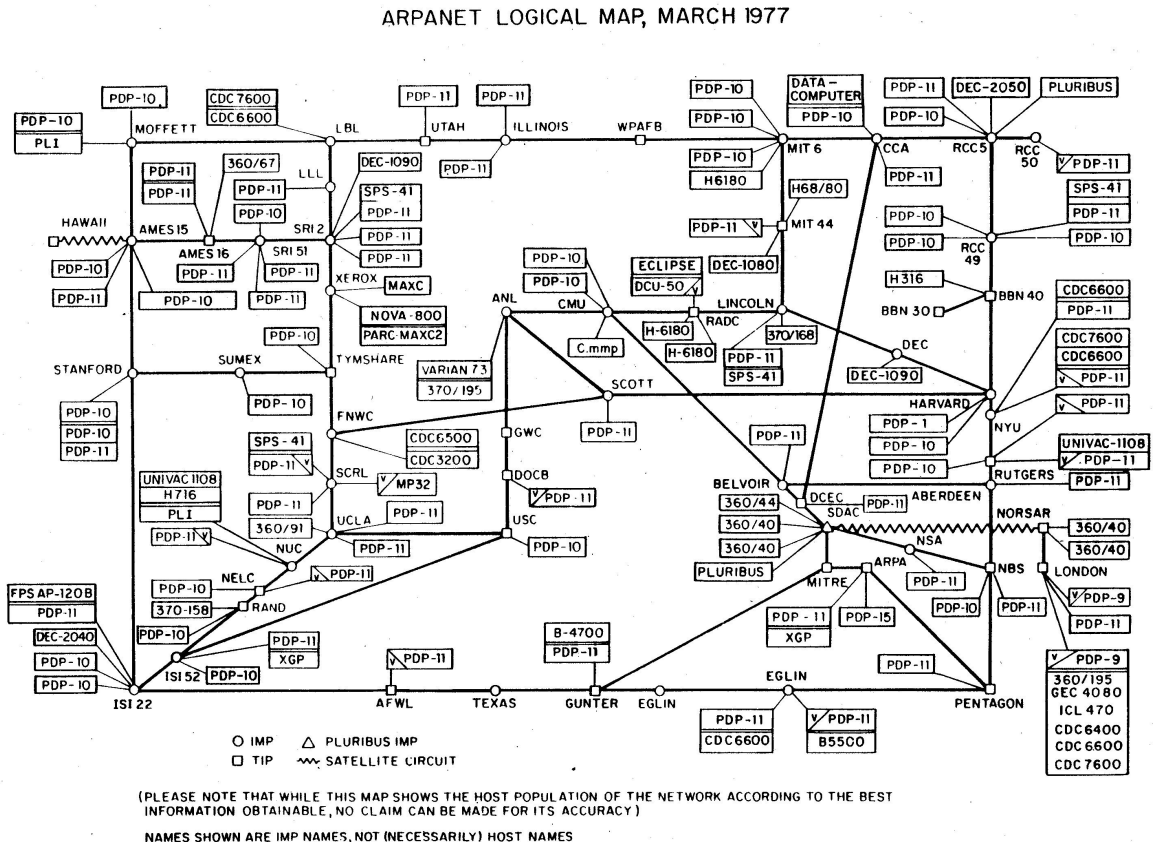
Computing power

Memory

Protocols

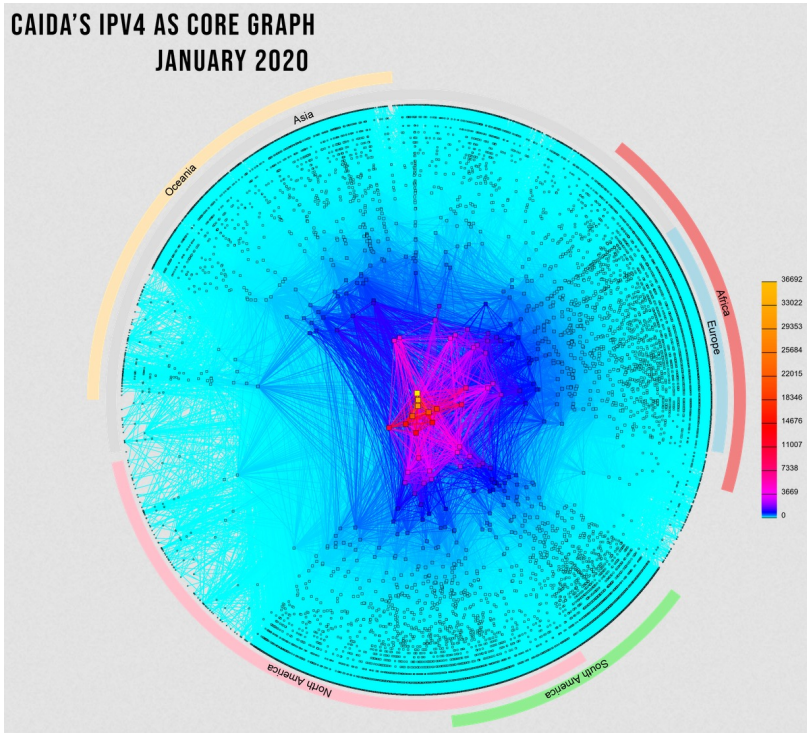
Storage

Internet evolution: 1977



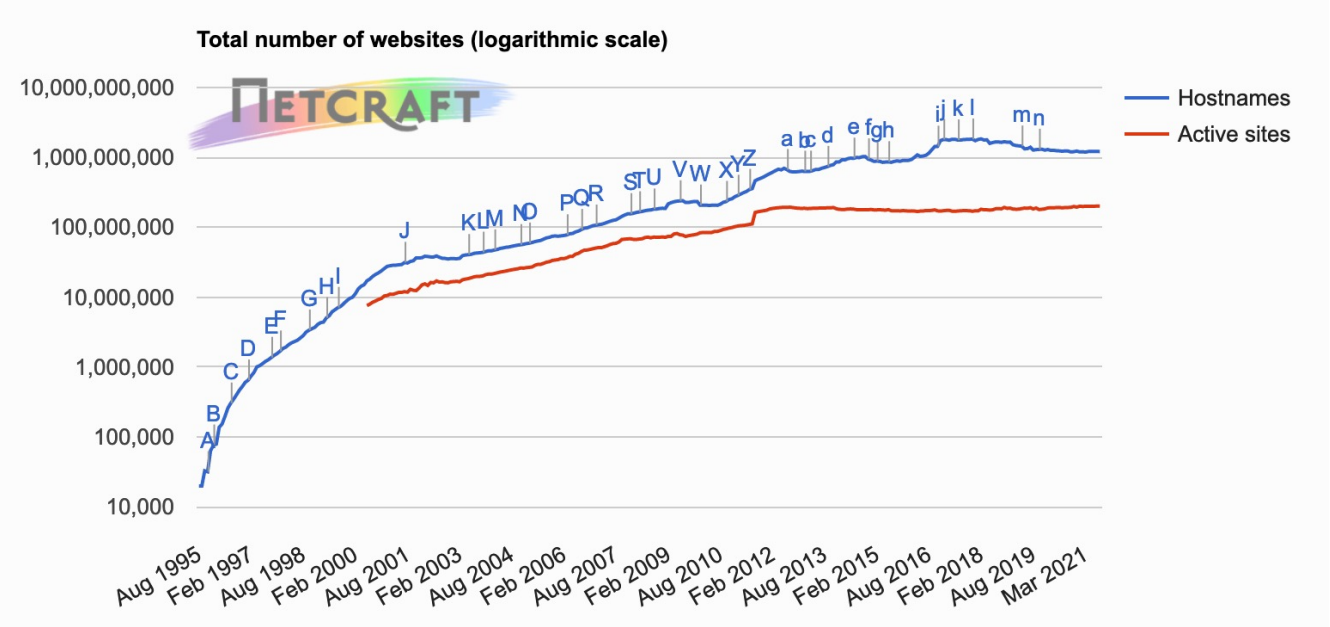
Internet evolution: after 43 years (2020)

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



Source: www.caida.org/research/topology/as_core_network/

Web growth: number of Web servers



In 2014 it was the first time the survey measured a billion websites: a milestone achievement that was unimaginable two decades ago

Source: Netcraft Web server survey

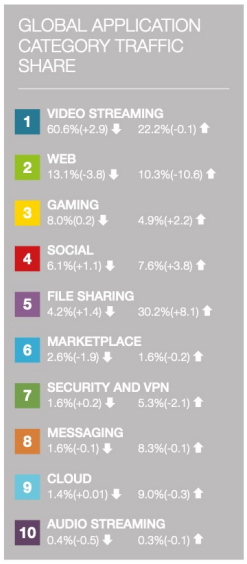
news.netcraft.com/archives/category/web-server-survey/

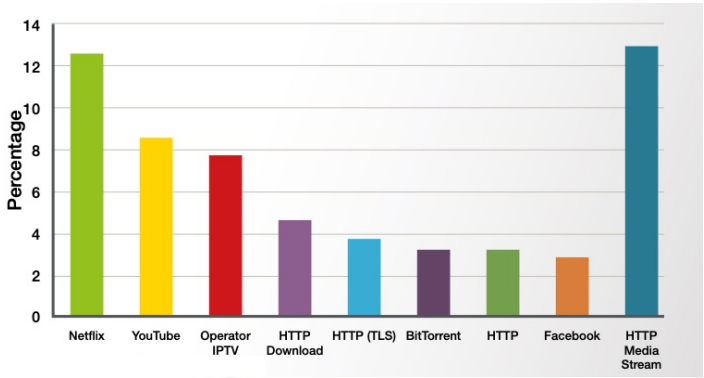
Metcalfe's law

"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

Internet traffic in 2019 and new trends





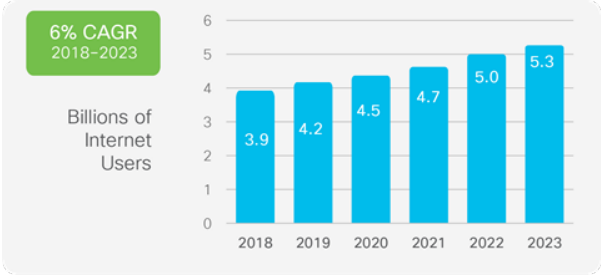
Traffic generated by IoT devices, voice assistants, mobile advertising, mobile crashes, cryptocurrencies, …

Source: sandvine, http://www.sandvine.com/

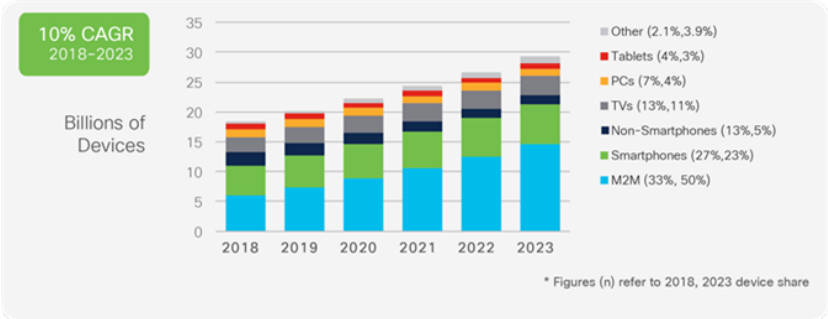
Future Internet traffic

Source: Cisco Internet report 2018-2023 bit.ly/3iQOjsN

- Growth in Internet users



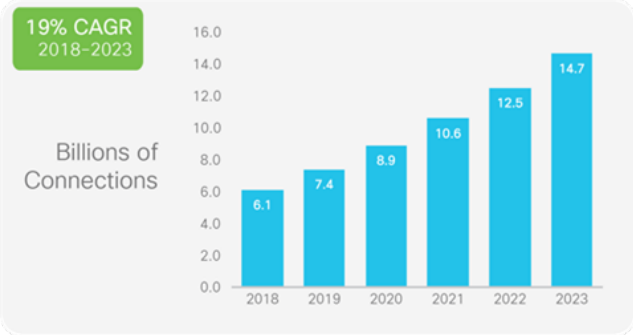
- Device and connection growth



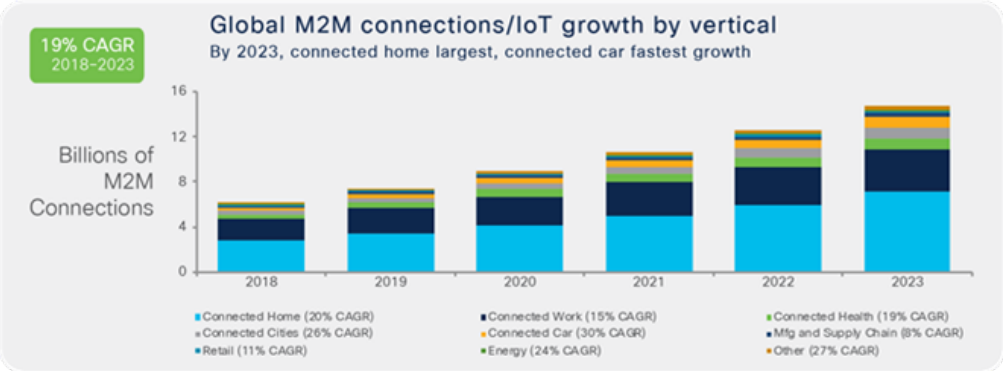
Implication of this growth: Internet is replacing voice telephony, television... will be the dominant transport technology for everything

Future Internet traffic

- Machine-2-machine (M2M) connection growth



- M2M apps across many industries accelerate **Internet of Things (IoT)** growth

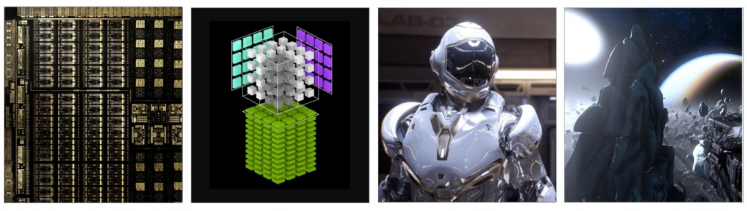


Computing power

- Computers got…
  - Smaller
  - Cheaper
  - Power efficient
  - Faster

- 1974: Intel 8080
  - 2 MHz, 6K transistors
- 2004: Intel P4 Prescott
  - 3.6 GHz, 125 million transistors
- 2011: Intel 10-core Xeon Westmere-EX
  - 3.33 GHz, 2.6 billion transistors
- GPUs scaled as well: in 2019 NVIDIA Turing GPU
  - 14.2 TFLOPS1 of peak single precision (FP32) performance

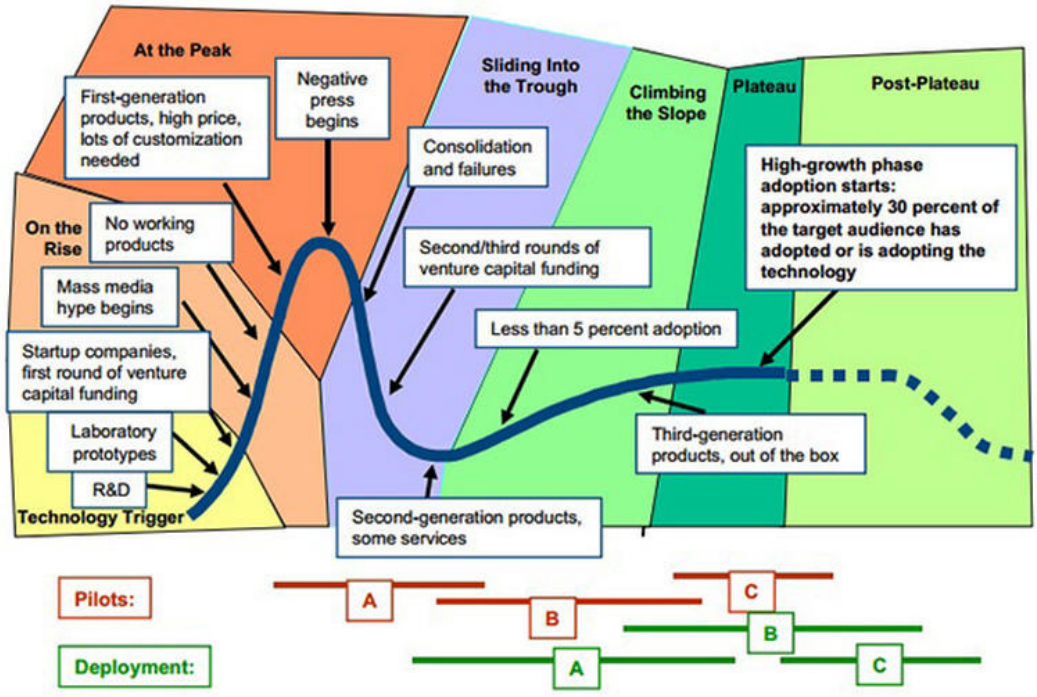
Multicore architectures



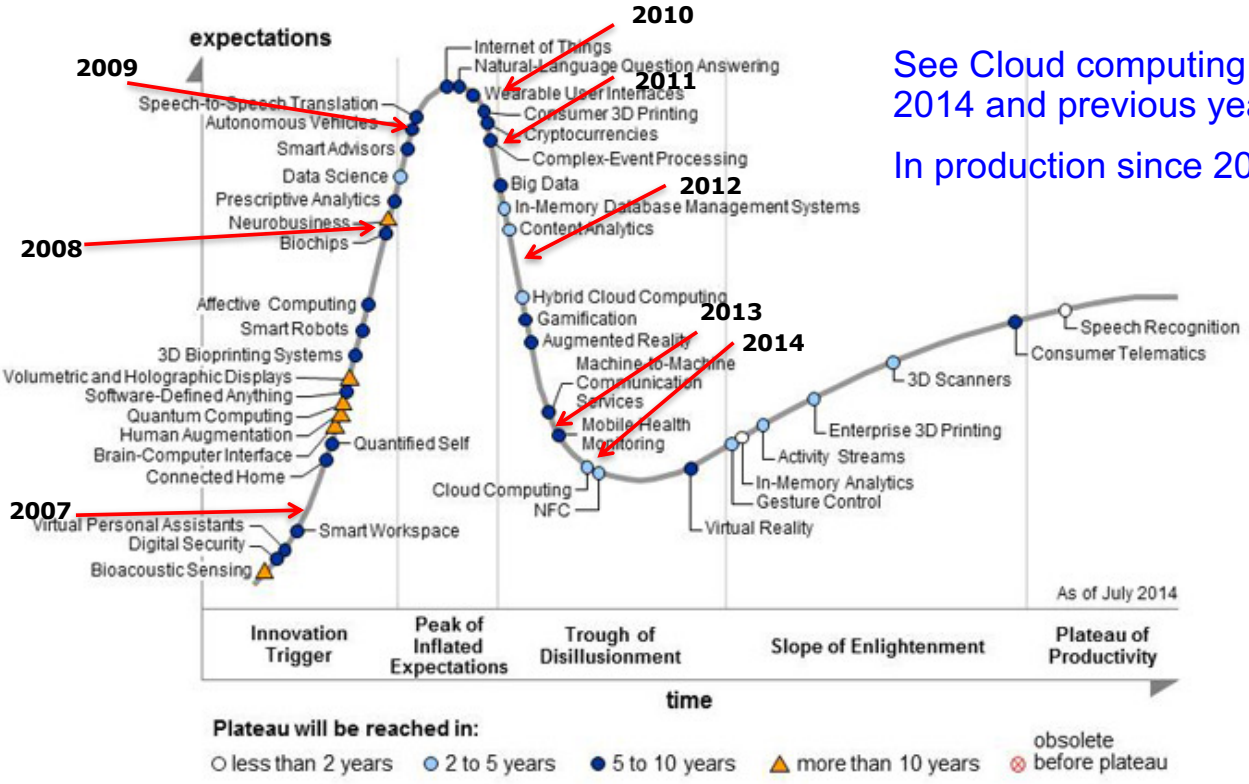
Distributed systems: not only Internet and Web

- Internet and Web: two notable examples of distributed systems
- Others include:
  - Cloud systems, HPC systems, … sometimes only accessible through private networks
  - Peer-to-peer (P2P) systems
  - Home networks (home entertainment, multimedia sharing)
  - Wireless sensor networks
  - Internet of Things (IoT)

Gartner's annual IT hype cycle for emerging technologies



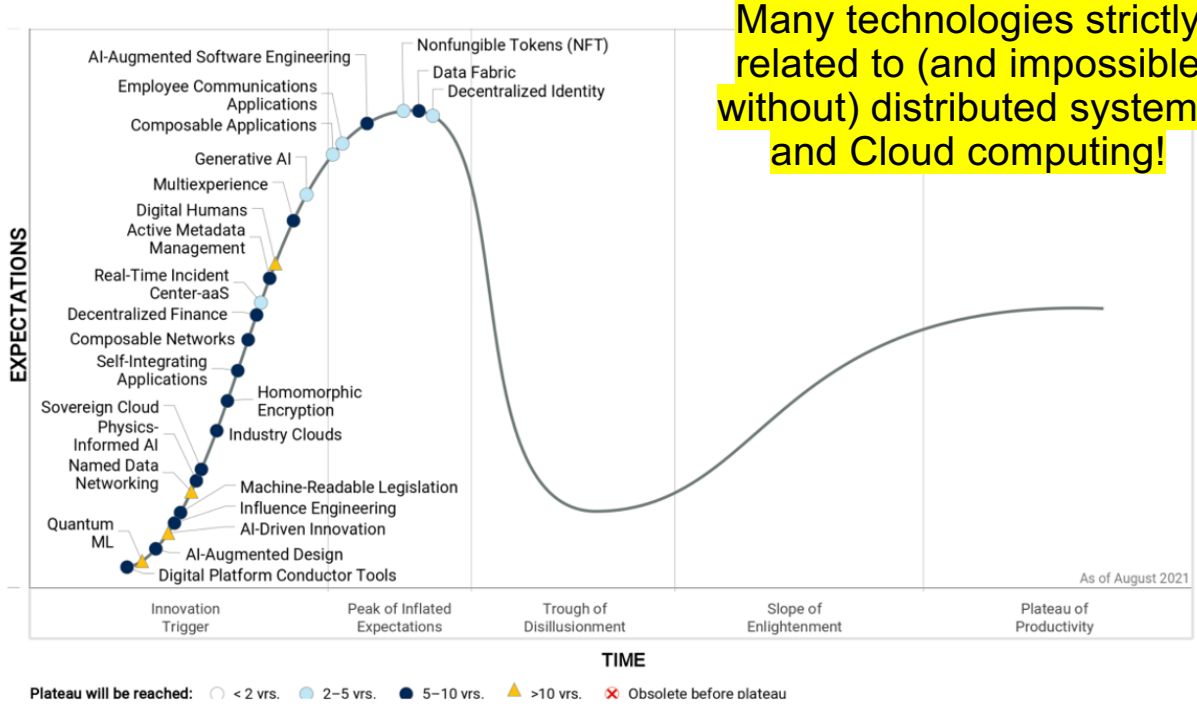
Hype cycle and cloud computing



See Cloud computing in 2014 and previous years

In production since 2015

Hype cycle for emerging technologies in 2021



Many technologies strictly related to (and impossible without) distributed systems and Cloud computing!

Source: Gartner (August 2021)

747576

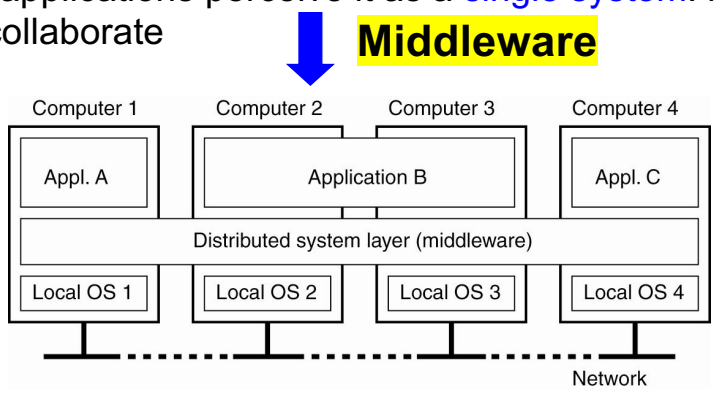
Distributed systems and AI

- Artificial Intelligence (AI) has become practical as the result of:
  - distributed computing
  - affordable cloud computing and storage costs

- 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

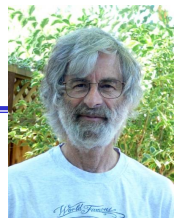
- Multiple definitions of **distributed system (DS)**, not always coherent with each other
- [van Steen & Tanenbaum] A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system
  - Consists of autonomous computing elements (i.e., **nodes**), can be hardware devices (computer, phone, car, robot, …) or software processes
  - Users or applications perceive it as a single system: nodes need to collaborate



Distributed system

- [Coulouris & Dollimore] A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages
  - If components = CPUs we have the definition of MIMD (Multiple Instruction stream Multiple Data stream) parallel architecture
- [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 bit.ly/2ZWaG8R
- His research contributions have laid the foundations of the theory and practice of distributed systems
  - Fundamental concepts such as causality, logical clocks and Byzantine failures; some notable papers:
    - "Time, Clocks, and the Ordering of Events in a Distributed System"
    - "The Byzantine Generals Problem"
    - "The Part-Time Parliament"
  - Algorithms to solve many fundamental problems in distributed systems, including:
    - Paxos algorithm for consensus
    - Bakery algorithm for mutual exclusion of multiple threads
    - Snapshot algorithm for consistent global states
- Initial developer 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)

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

Some distinguishing features of DS

- Concurrency
  - Many things are occurring “at the same time”
  - Centralized system: design choice
  - Distributed system: fact of life to be dealt with
- Absence of global clock
  - Centralized systems: use the computer’s physical clock for synchronization
  - Distributed systems: many physical clocks and not necessarily synchronized
- Independent and partial failures
  - Centralized systems: fail completely
  - Distributed systems: fail partially (i.e., only a part), often due to communication; hard (and in general impossible) to hide partial failures and their recovery

Challenges in distributed systems

- Many challenges associated with designing distributed systems
  - **Heterogeneity**
  - **Distribution transparency**
  - **Openness**
  - **Scalability**

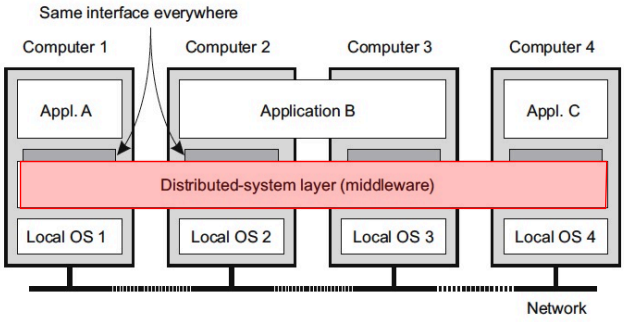
while improving performance and availability, guaranteeing security, energy efficiency, …

Heterogeneity

- Levels:
  - Networks
  - Computer hardware
  - Operating systems
  - Programming languages
  - Multiple implementations by different developers
- Solution? **Middleware**: the OS of DSs

  Middleware: software layer placed on top of OSs providing a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages

  Contains commonly used components and functions that need not be implemented by applications separately

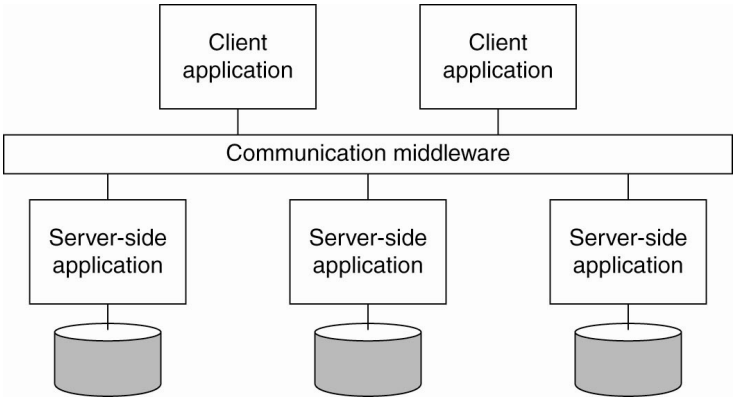


Some middleware services

- Communication
- Transactions
- Service composition
- Reliability

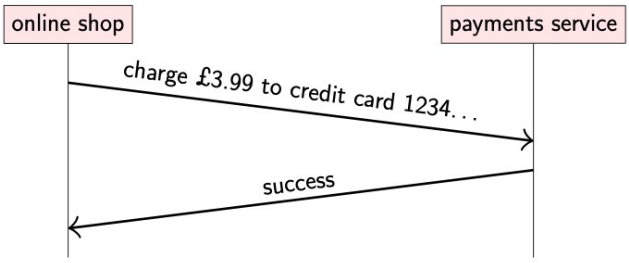
Communication middleware

- Communication middleware: to facilitate communication among (heterogeneous) DS components/apps
- We will study
  - Remote Procedure Call (RPC)
  - Remote Method Invocation (RMI)
  - Message Oriented Middleware (MOM)



Remote Procedure Call (RPC) example

- Online payment



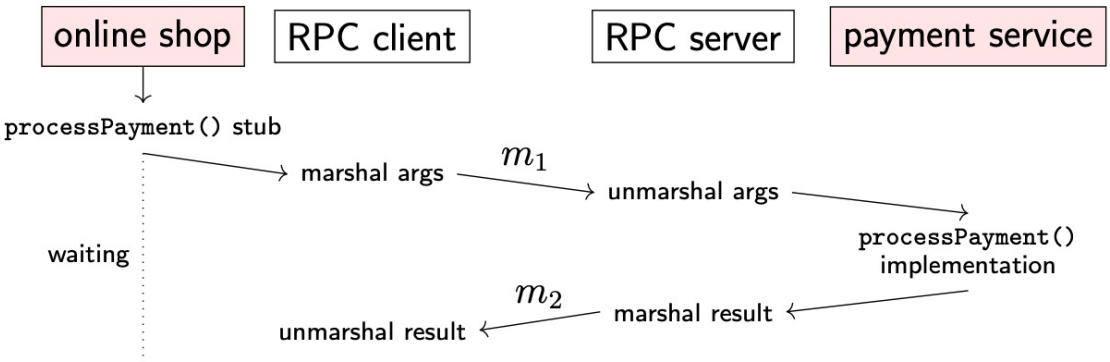

```
// Online shop handling customer's card details
Card card = new Card();
card.setCardNumber("1234 5678 8765 4321");
card.setExpiryDate("10/2024");
card.setCVC("123");

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



$m_1 =$

```
{
  "request": "processPayment",
  "card": {
    "number": "1234567887654321",
    "expiryDate": "10/2024",
    "CVC": "123"
  },
  "amount": 3.99,
  "currency": "GBP"
}
```

$m_2 =$

```
{
  "result": "success",
  "id": "XP61hHw2Rvo"
}
```


Distribution transparency

- Distribution transparency: single coherent system where the distribution of 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 resources are accessed
    - e.g., use same mechanism for local or remote call

Location transparency
  - Hide where resources are located
    - e.g., URL hides IP address
  - Access + location transparency = network transparency

Migration transparency
  - Hide that resources may move to another location (even at runtime) without affecting operativeness

Distribution transparency

Replication transparency

- Hide that there are multiple replicas of the same resource
  - Each replica should have the same name, e.g., type in terminal `$ dig www.youtube.com`
  - Require also location transparency

Concurrency transparency

- Hide that resources may be shared by several independent users
  - E.g.: concurrent access of multiple users to the same DB table
  - Concurrent access to shared resource should leave it in a consistent state; e.g., by using *locking* mechanisms

Failure transparency

- Hide failure and recovery of resources
- See DS definition by Lamport

Degree of distribution transparency

- Aiming to *full* distribution transparency is often too much
  - Communication latencies cannot be always hidden: access from Rome to a resource located on a server in New York requires ~23 ms
  - Impossible to completely hide failures in a large-scale DS
    - You cannot distinguish a slow computer from a failing one
    - You can never be sure that a server actually performed an operation before a crash
  - Full transparency costs in terms of performance
    - E.g.: keeping data replicas exactly up-to-date takes time
    - Tradeoff between degree of consistency and system performance

Openness

- Open DS: able to interact with services from other open systems, irrespective of the underlying environment
- Systems should conform to well-defined interfaces
  - Service interface defined through IDL (Interface Definition Language)
    - Nearly always capture only syntax, not semantics
    - Complete and neutral
    - IDL examples: Sun RPC, Thrift, WSDL, OMG IDL
- Systems should easily interoperate
- Systems should support portability of applications
- Systems should be easily extensible
- Examples: Java EE, .Net, Web Services

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

Separating policies from mechanisms

- To implement open and flexible DS: separate policies from mechanisms
- DS provides only mechanisms
  - E.g., mechanisms for Web browser
    - Support for data caching
  - E.g., policies for Web browser
    - Which resources in cache?
    - How long in cache?
    - When to refresh?
    - Private or shared cache?
- As a result, many parameters to be configured: 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)

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/or processes (**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

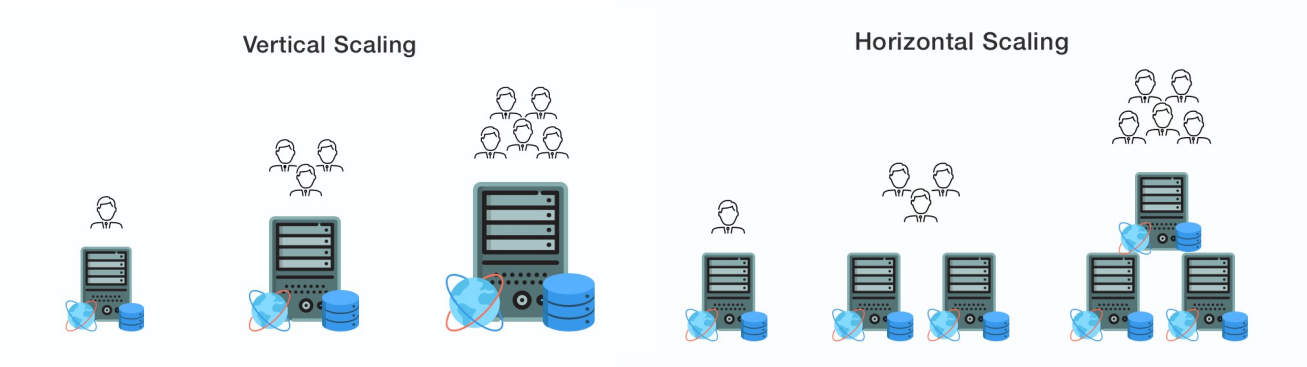
> "Many developers of modern distributed systems easily use the adjective scalable without making clear why their system actually scales." (van Steen)

Scalability

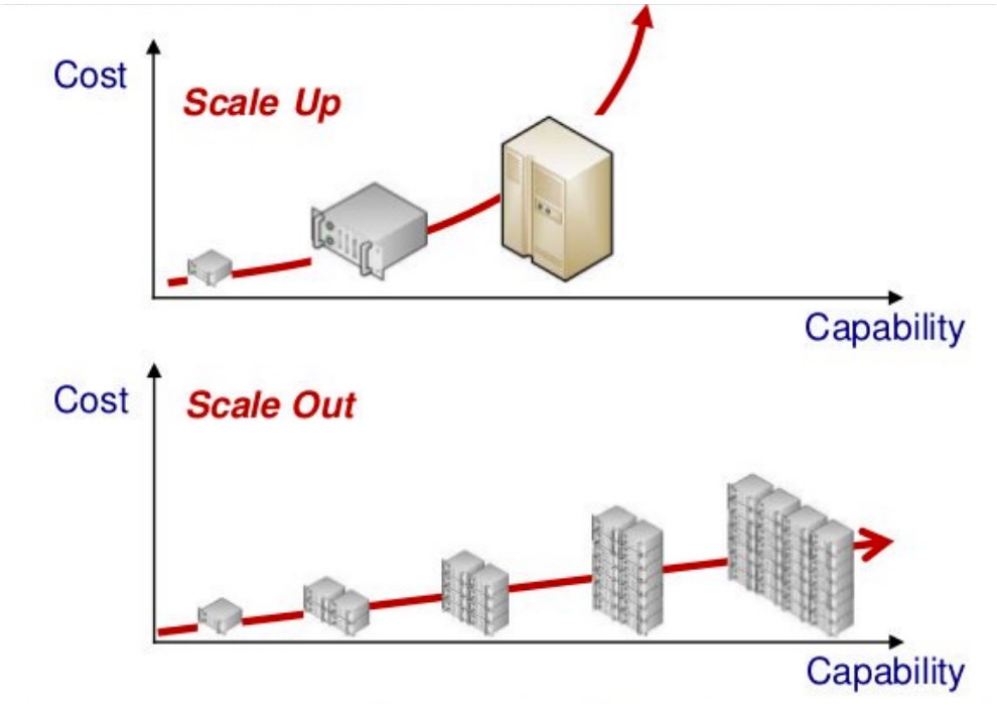
- Root causes for scalability problems with centralized solutions
  - Computational capacity, limited by CPUs
  - Storage capacity, including transfer rate between CPUs and disks
  - Network between user and centralized service

Size scalability

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

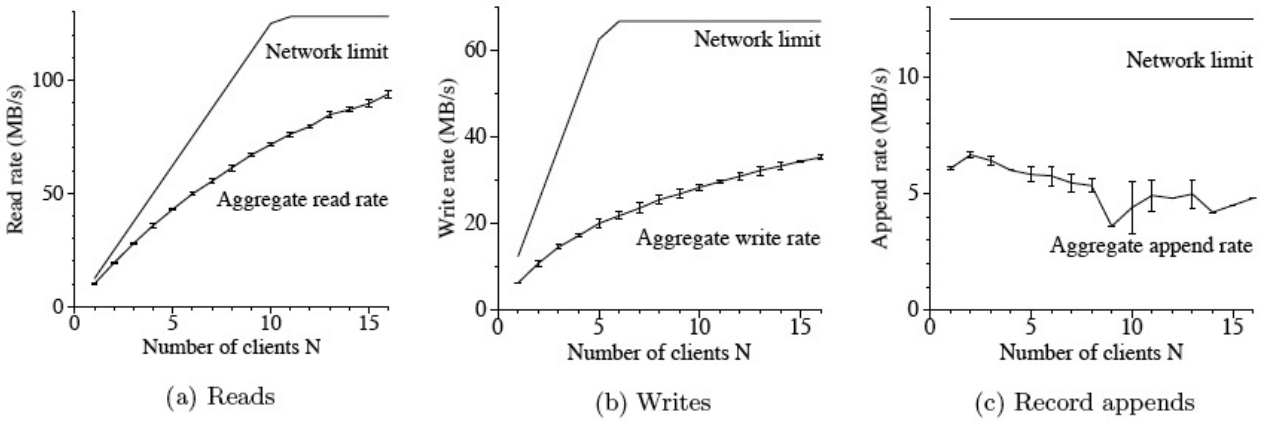


Scale-up vs. scale-out



Size scalability: example

- Google File System
  - Distributed file system realized by Google's researchers



- Scale parameter: number of clients
- Scalability metric: aggregated read/write/append throughput, assuming random file access
- Scalability criterion: the closer to network limit, the better

Techniques for scaling

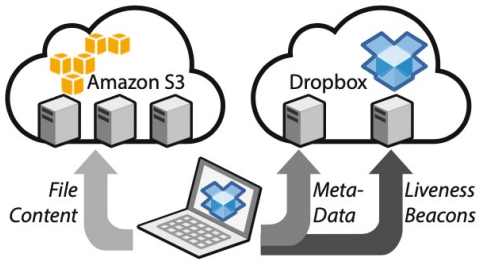
1. Hide communication latency
   - Make use of **asynchronous communication**
   - Make separate handler for incoming response
   - Problem: not every app fits this model (e.g., highly interactive ones)

2. Facilitate solution by moving computations to client

3. Partition data and computation across multiple resources
   - Divide et impera: partition data and computation into smaller parts and distributed 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
   - Distribute processing on multiple resource replicas
   - Maintain a copy of the same data on multiple 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



Consistency problems

- Applying data replication seems easy to apply, but…
- One replica should be "consistent" with another replicas
- We will study a variety of consistency models to choose from
  - Strict consistency requires global synchronization
- Depending on the application type, we can tolerate a certain degree of inconsistency
  - E.g.,: blog, shared file, electronic shopping cart, on-line auction, air traffic control

Fallacies in realizing distributed systems

- Many distributed systems are needlessly complex because of errors in design and implementation that were patched later
- Many 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 kilometers per second, it will always take at least 30 milliseconds 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

Fallacies in realizing distributed systems

4. The network is secure
5. 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 environment is homogeneous

Do not think that technology solves everything!

See Fallacies of Distributed Computing Explained

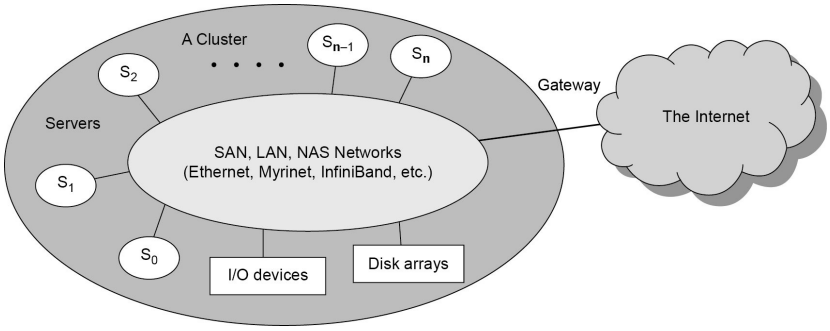
Listen to Episode 470: L. Peter Deutsch on the Fallacies of Distributed Computing

Three types of distributed systems

- High-performance distributed computing systems
  - Cluster computing
  - Cloud computing
  - Edge/fog computing
- Distributed information systems
- Distributed pervasive systems

Cluster computing

- Computer cluster: group of high-end servers connected through a LAN
  - Homogeneous: same OS, near-identical hardware
- Main goals: HPC (High Performance Computing) and/or HA (High Availability)
- Typical cluster architecture



Clusters dominate TOP500 architectures www.top500.org

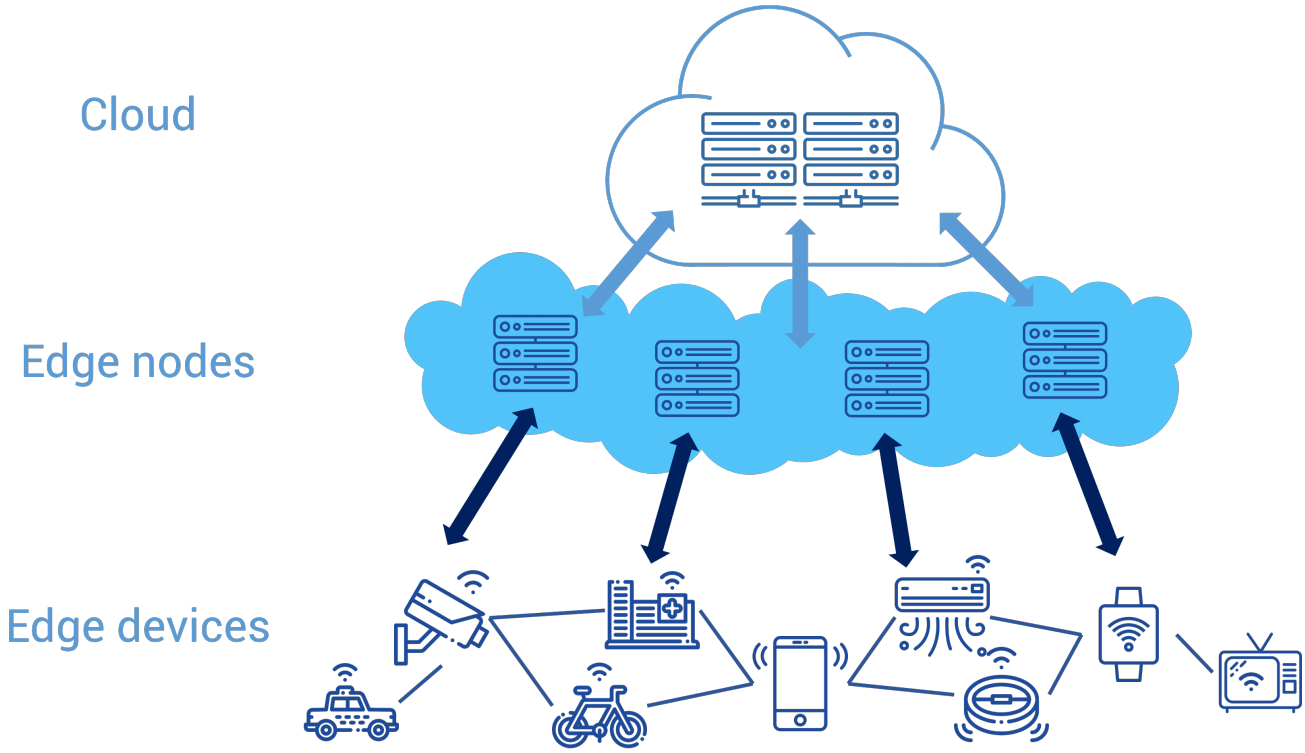
Cluster computing

- Often organized with a master/worker architecture
  - E.g., Beowulf cluster using MPI library
- Can be controlled by specific software tools that manage them as a single system
  - E.g., Mosix: cluster management system that provides a single-system image
    - Among features: automatic resource discovery and workload distribution by process migration

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 the user to physically see or use hardware

Decentralization: from Cloud to fog/edge computing



Distributed information systems

- Among distributed information systems let us consider transaction processing systems

```
BEGIN_TRANSACTION(server, transaction);
READ(transaction, file1, data);
WRITE(transaction, file2, data);
newData := MODIFIED(data);
IF WRONG(newData) THEN
  ABORT_TRANSACTION(transaction);
ELSE
  WRITE(transaction, file2, newData);
  END_TRANSACTION(transaction);
END IF;
```

  - The effect of all `READ` and `WRITE` operations become permanent only with `END_TRANSACTION`
  - A transaction is an atomic operation ("**all**-or-**nothing**")

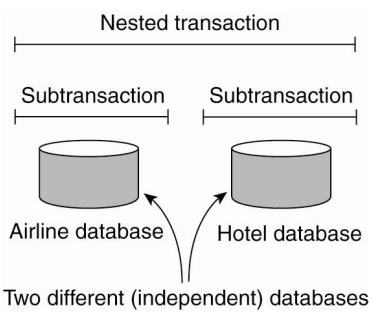
Transaction

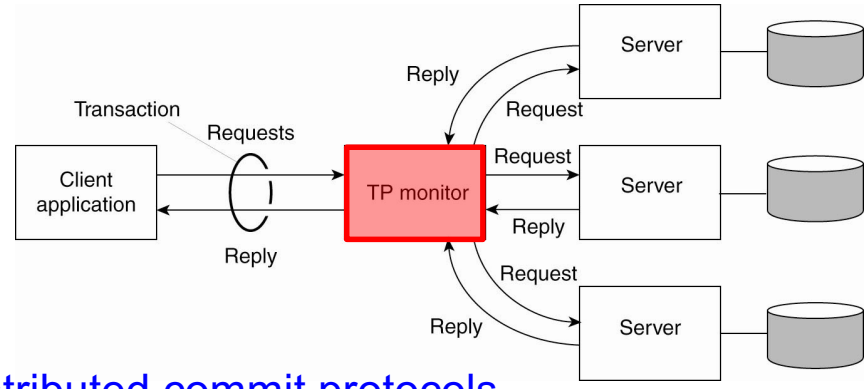
- Transaction: unit of work that you want to see as a whole and is treated in a coherent and reliable way independent of other transaction

- **ACID** properties
  - **Atomic**: happens indivisibly (seemingly)
  - **Consistent**: does not violate system invariants
  - **Isolated**: no mutual intereferences
  - **Durable**: commit means changes are durable

Distributed transactions

- Distributed (or nested) transaction: composed by multiple sub-transactions which are distributed across several servers
  - *Transaction Processing (TP) Monitor*: responsible for coordinating the execution of the distributed transaction
  - Example: Oracle Tuxedo





- We'll study distributed commit protocols

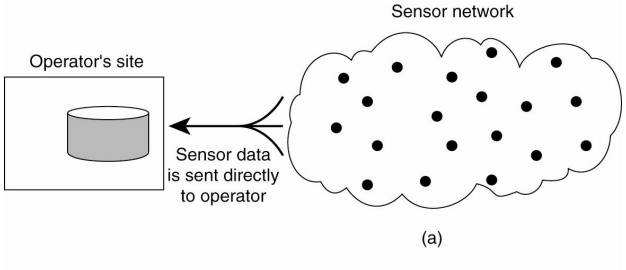
Distributed pervasive systems

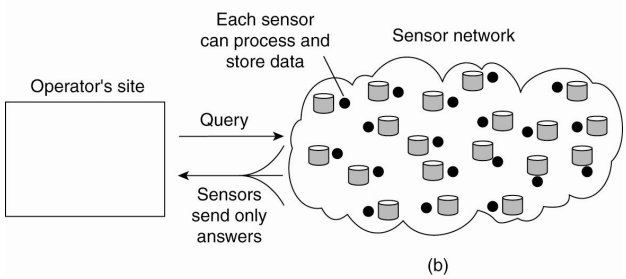
- Distributed systems whose nodes are often
  - small, mobile, battery-powered and often embedded in a larger system
  - characterized by the fact that the system naturally blends into the user's environment

- 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

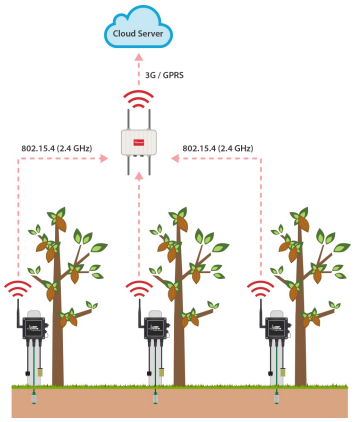
Sensor networks

- Sensors
  - Many (10-10^6)
  - Simple: limited computing, memory and communication capacity
  - Often battery-powered (or even battery-less)
  - Failures are frequent
- Sensor networks as distributed systems: 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)





Wireless sensor networks (WSNs)



Agricultural WSNs

Underwater WSNs