Distributed Systems: classification

Agenda

Distributed systems: examples

Trends

Resource sharing

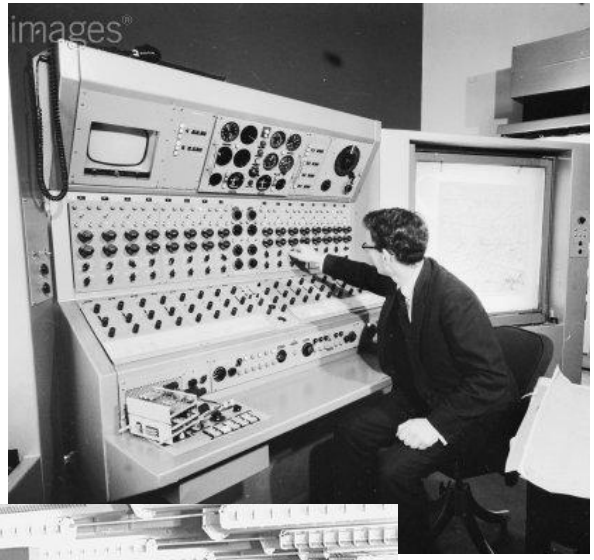
Challenges

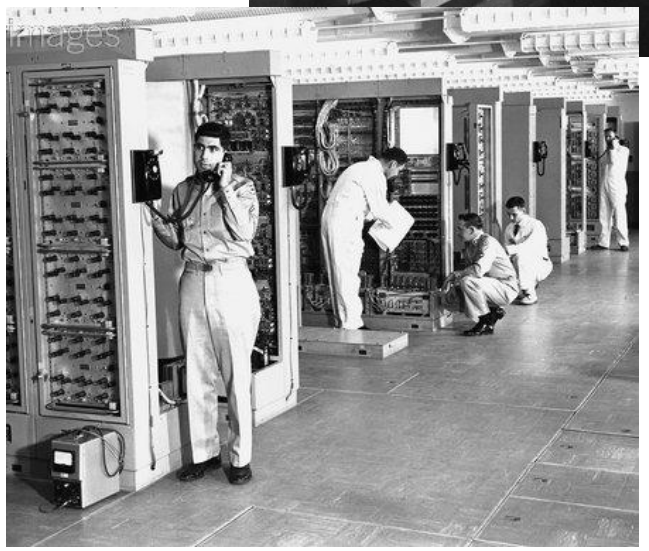
Case study

Early centralized systems

In the early days, computing systems were expensive and monolithic mainframe computers

The internet was developed for military stakeholders: distributed routing was invented in case of nuclear attacks





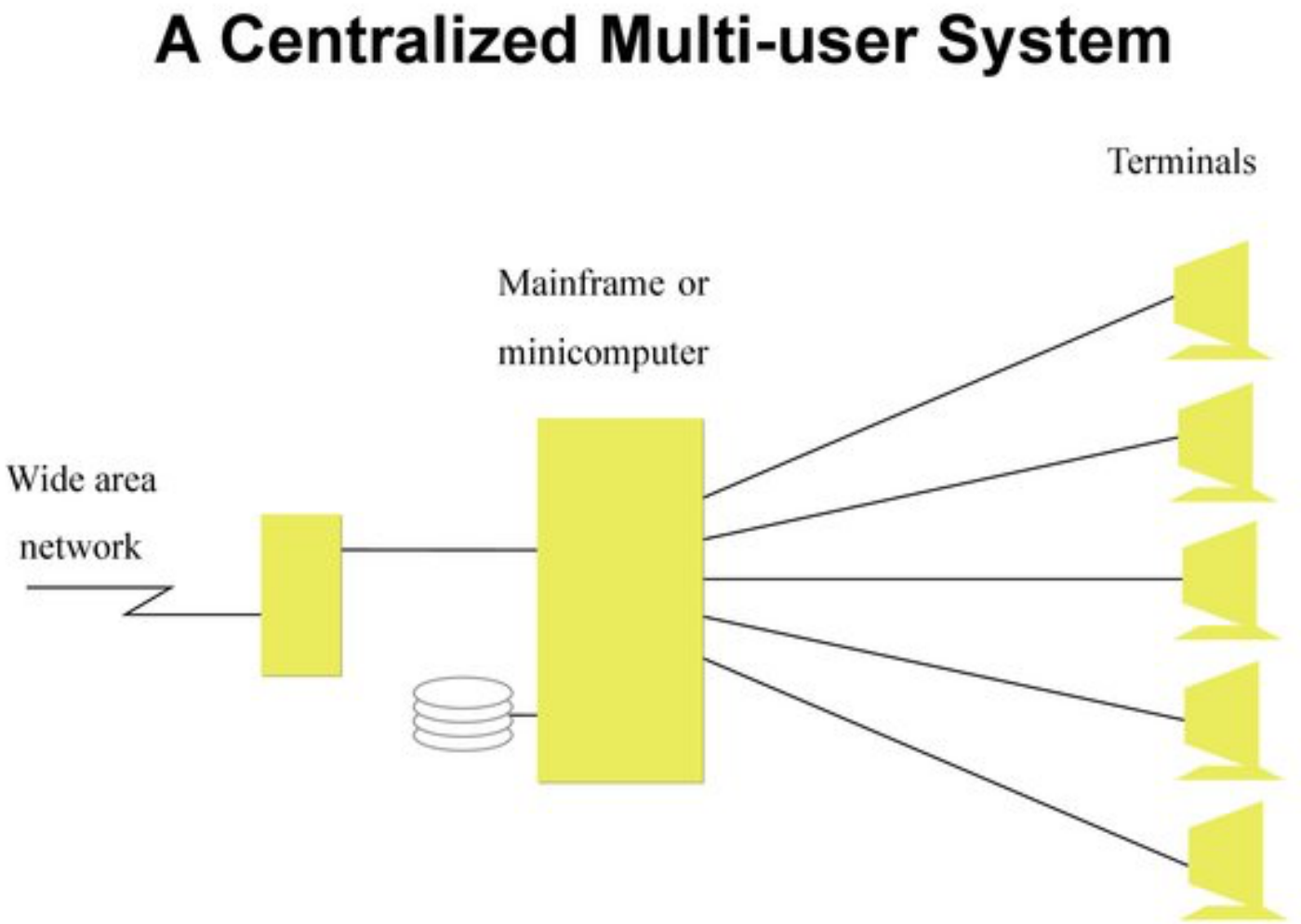
Control rooms

Centralized controls

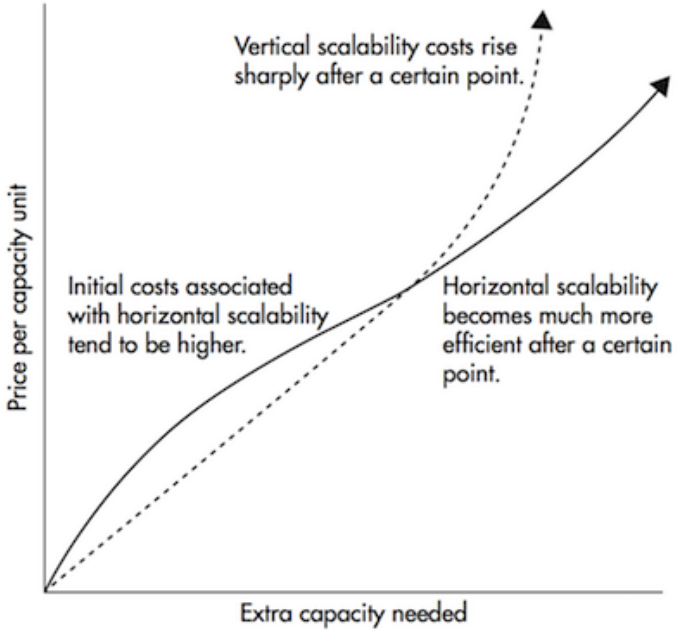
Complex interfaces

Very complicate

Centralized system: example



Why distribute a computing system?



The only way to handle more traffic with a single computer consists of upgrading the hardware a database is running on: this is called **scaling vertically**.

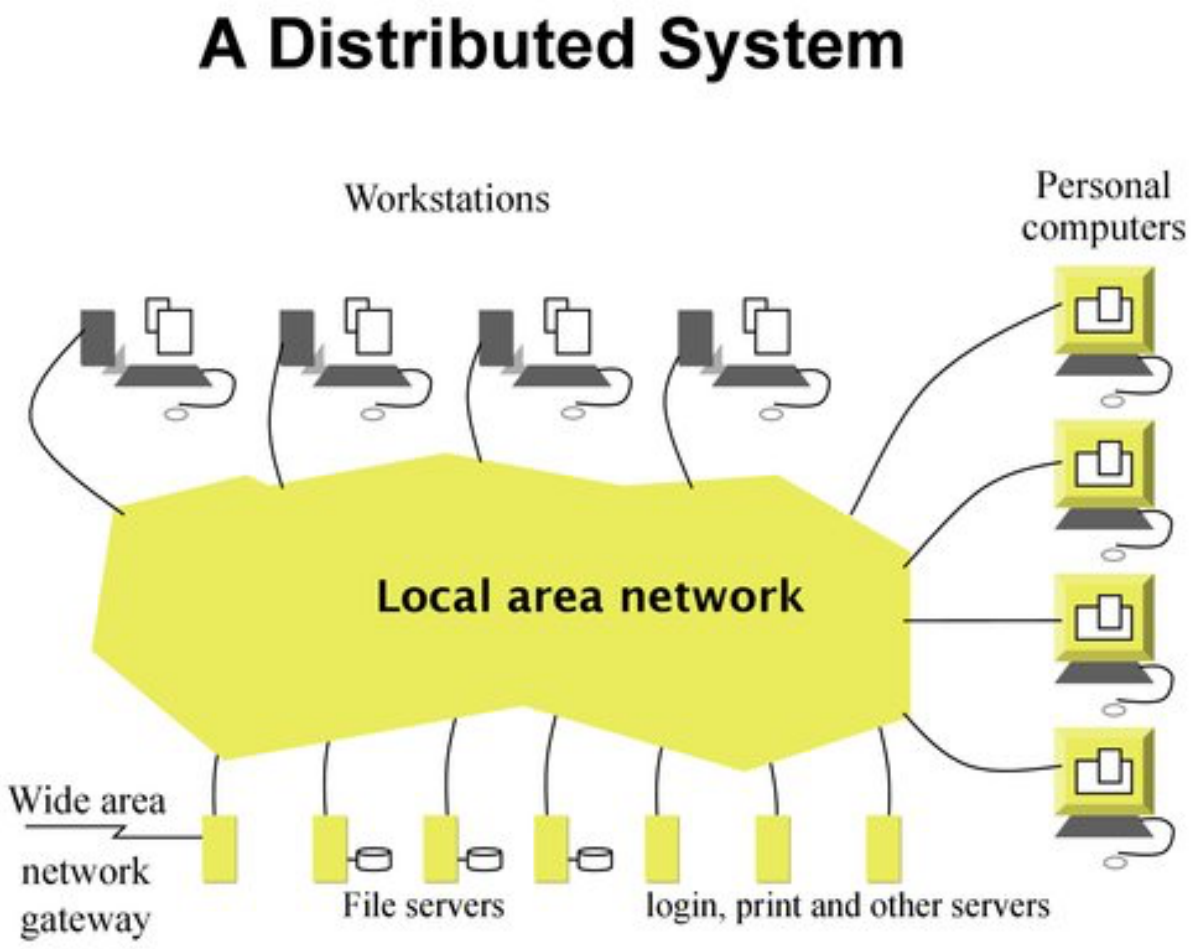
Scaling vertically is good, but after a certain point you will see that even the best hardware is not sufficient for traffic, not to mention impractical to host.

A distributed system can **scale horizontally**.

Scaling horizontally simply means adding more computers rather than upgrading the hardware of a single one.

https://medium.freecodecamp.org/a-thorough-introduction-to-distributed-systems-3b91562c9b3c

Distributed system: example



Distributed systems

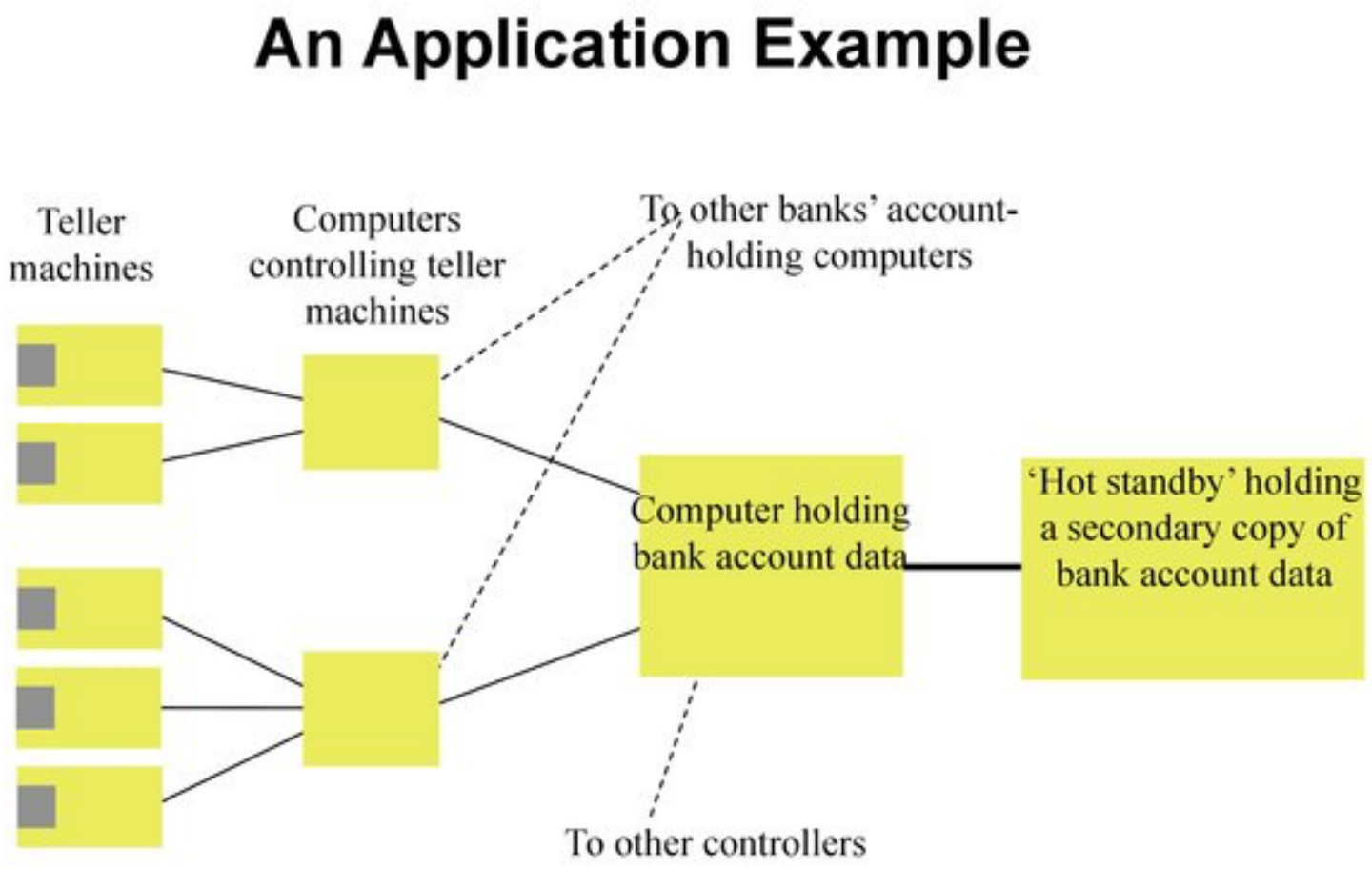
Modern businesses are decentralized

Distributed computing moves information and services closer to the customers and users who need them.

Personal and mobile computers and network servers are less expensive than mainframe computers

–Though total cost of ownership is still expensive

Distributed application: example



What is a distributed system

A distributed system is
a collection of autonomous computing elements
that appears to its users
as a single coherent system

Examples

The Web

Facebook

A cloud service (eg AWS)

An IoT system (eg. Domotics)

Skype

Telegram

Ethereum

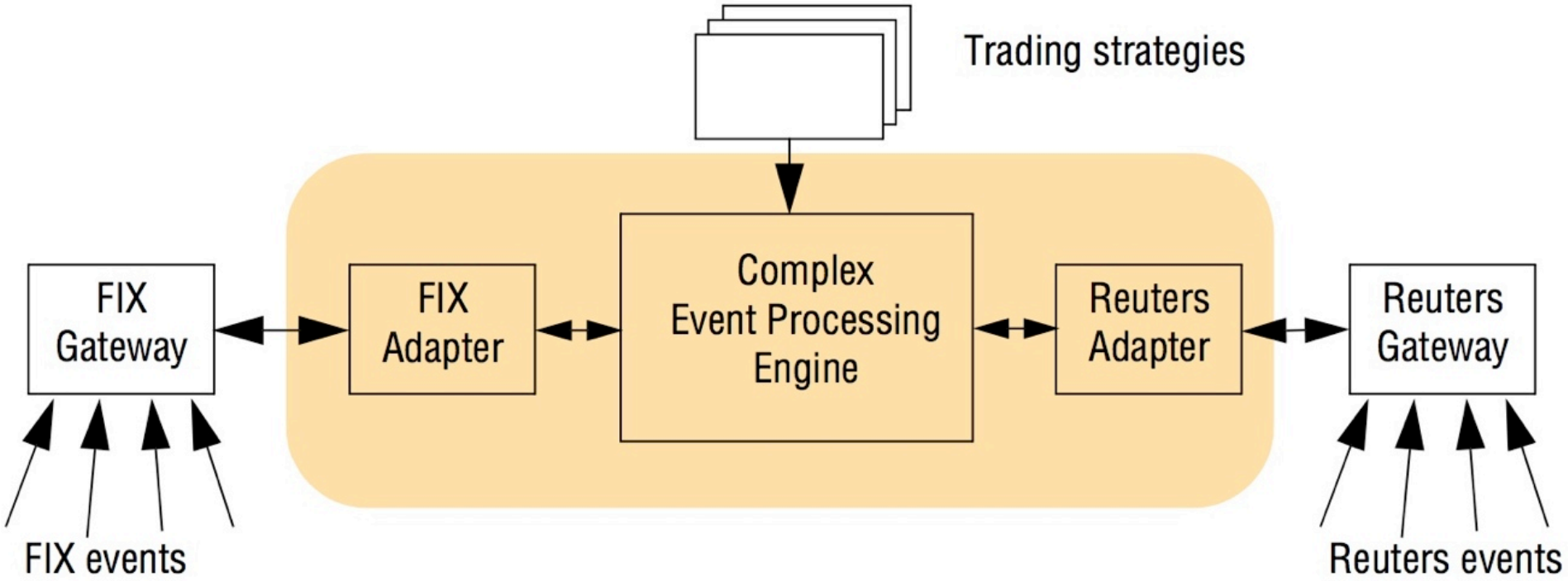
A massive multiuser game

…

Selected application domains and associated networked applications

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading, cryptocurrency
The information society	Web search engines, Wikipedia; social networking: Facebook Tumblr and Twitter.
Creative industries and entertainment	online gaming, music and movies at home, user-generated content, e.g. YouTube, Instagram
Healthcare	health informatics, on online patient records, monitoring wearable sensors
Education	e-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth, IoT
Science	The Grid for collaboration between scientists; github as collaboration between developers
Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

An example financial trading system



Important non functional property: **simultaneity** (very difficult)

See https://queue.acm.org/detail.cfm?id=2745385

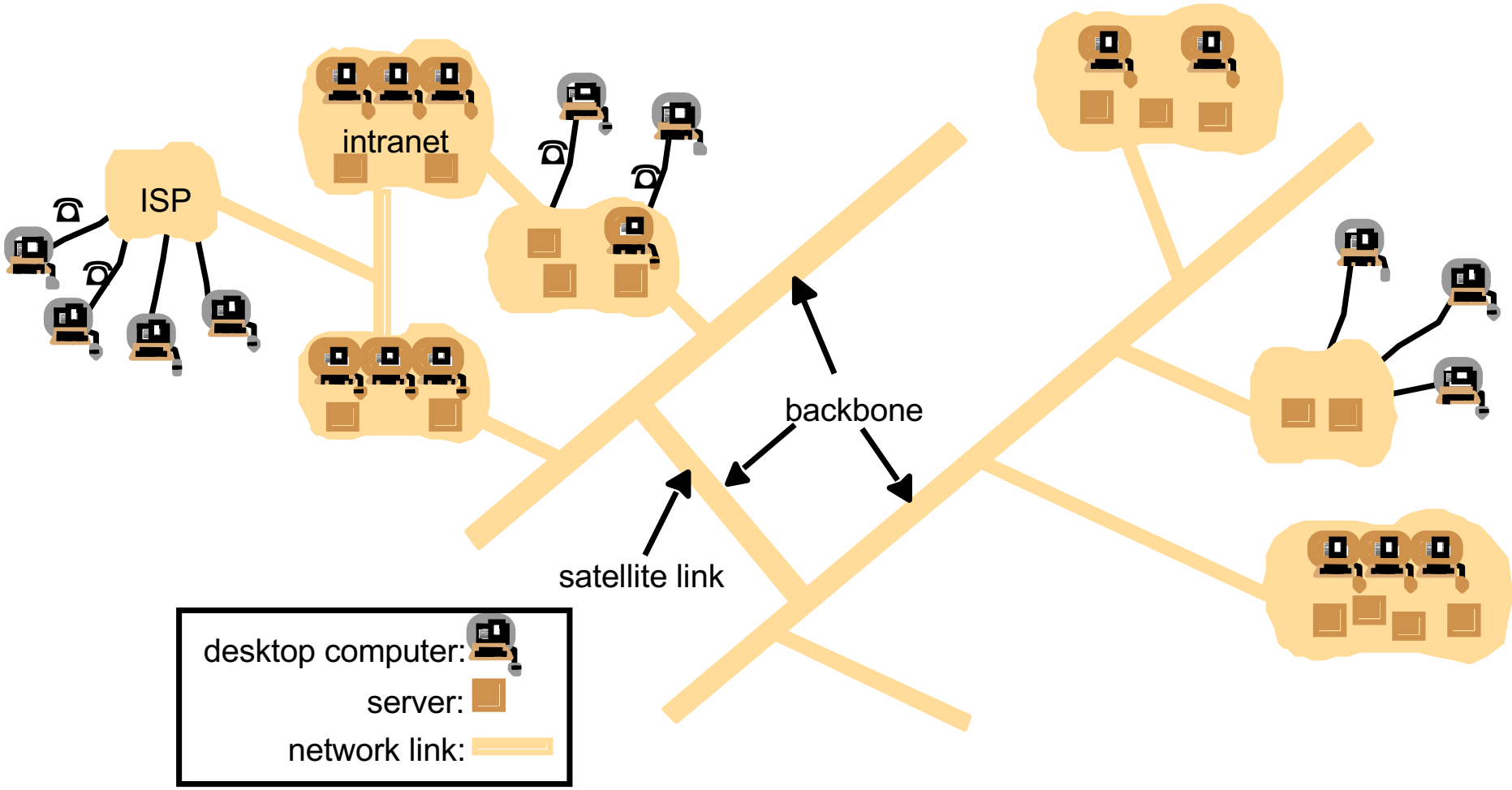
Collection of autonomous computing elements

- In a distributed system nodes act independently from each other
- They are programmed (usually by different people) to reach common goals by exchanging messages
- **IMPORTANT: Each node has its own notion of time** (the system has no global clock)

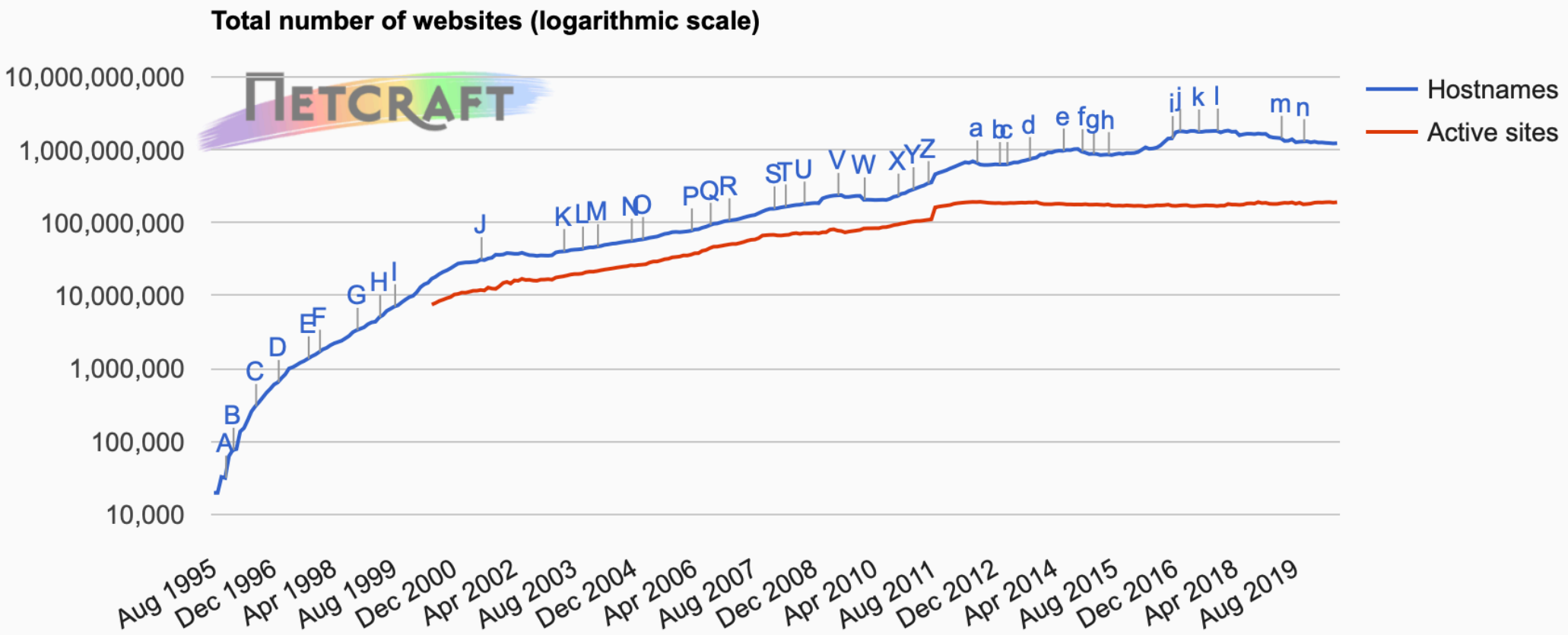
Distributed systems vs networks of computers

- A computer network is an interconnected collection of autonomous computers able to exchange data.
- A computer network usually requires users to explicitly login on one machine, explicitly submit jobs remotely, explicitly move files/data around the network.
- In a distributed system the existence of multiple autonomous computers is transparent to the user.
- The middeware automatically allocates jobs to processors, moves files among various computers without explicit user intervention

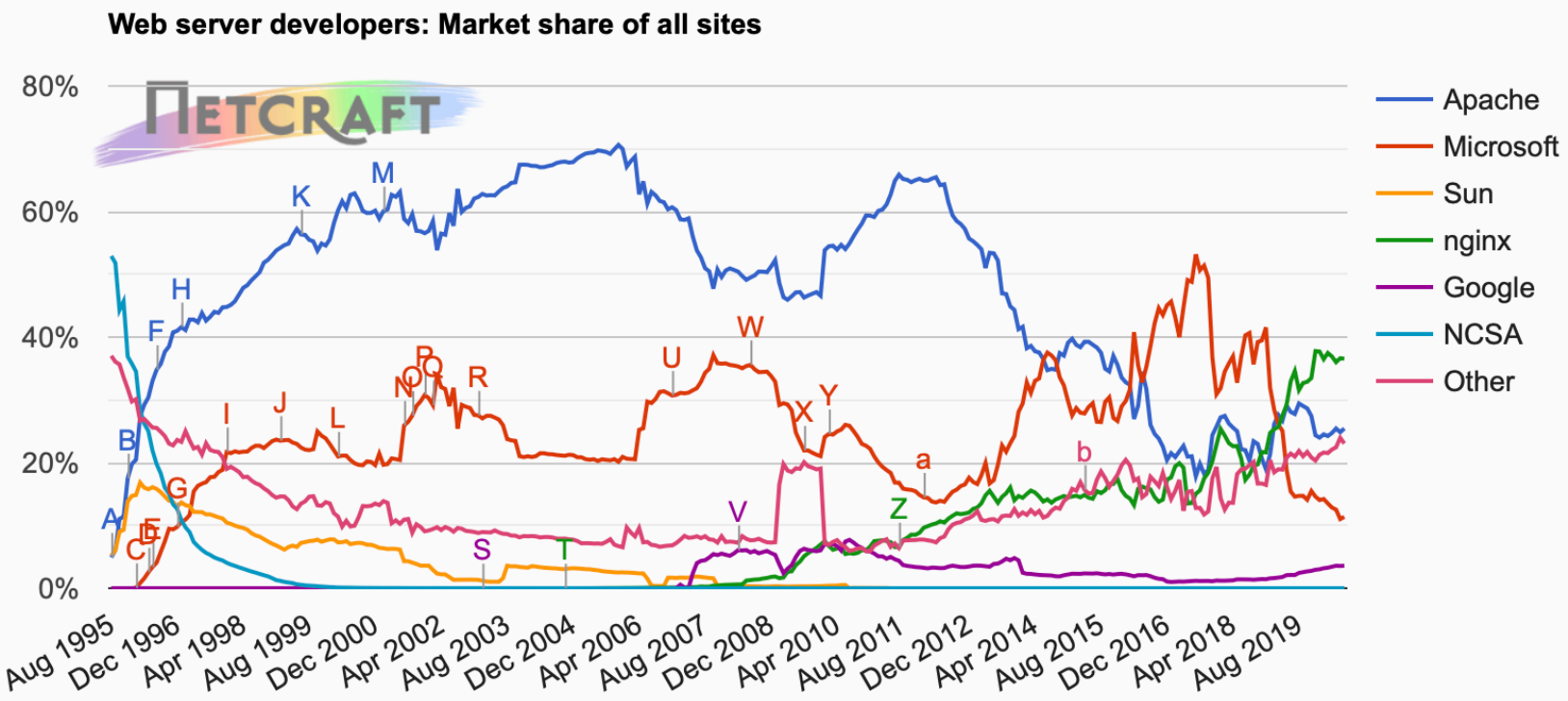
A typical portion of the Internet



https://news.netcraft.com/archives

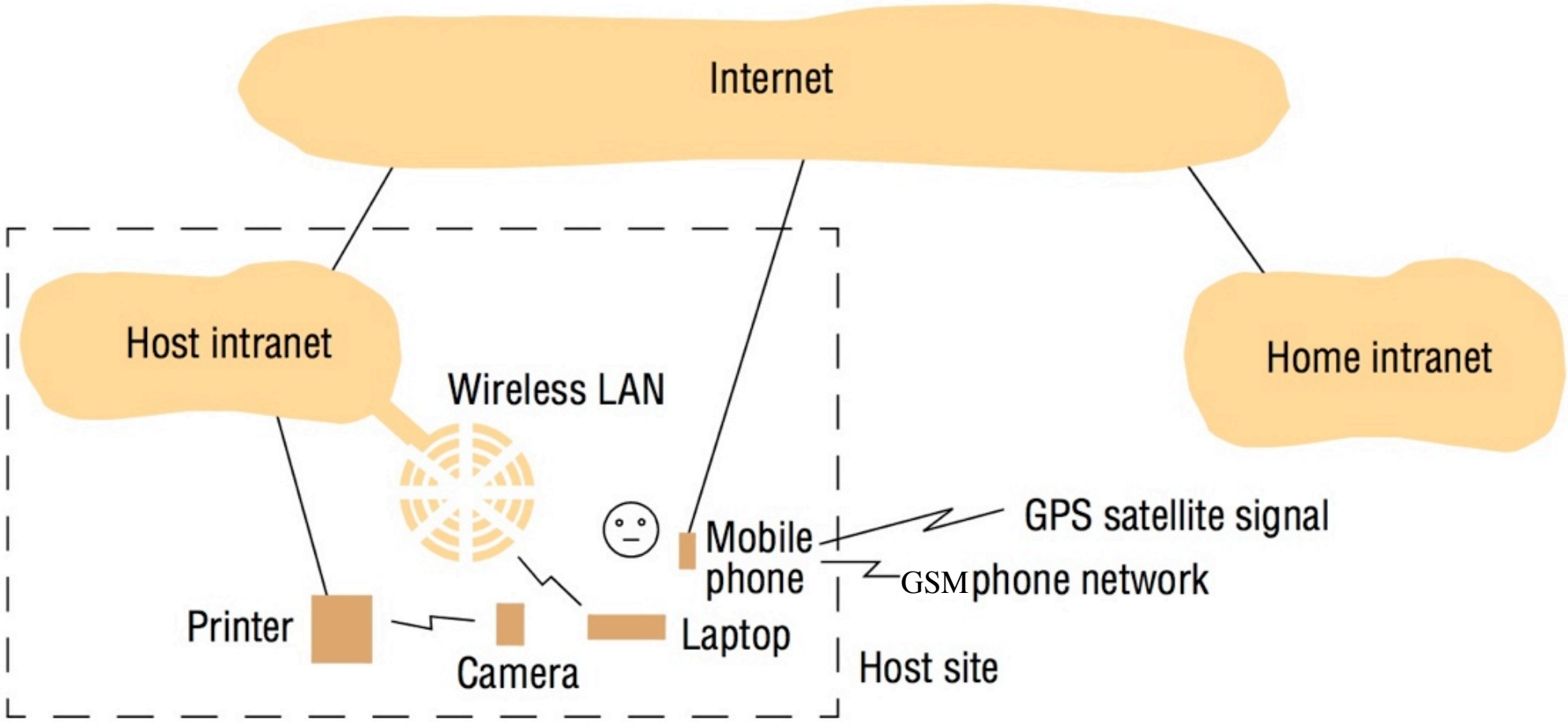


https://news.netcraft.com/archives

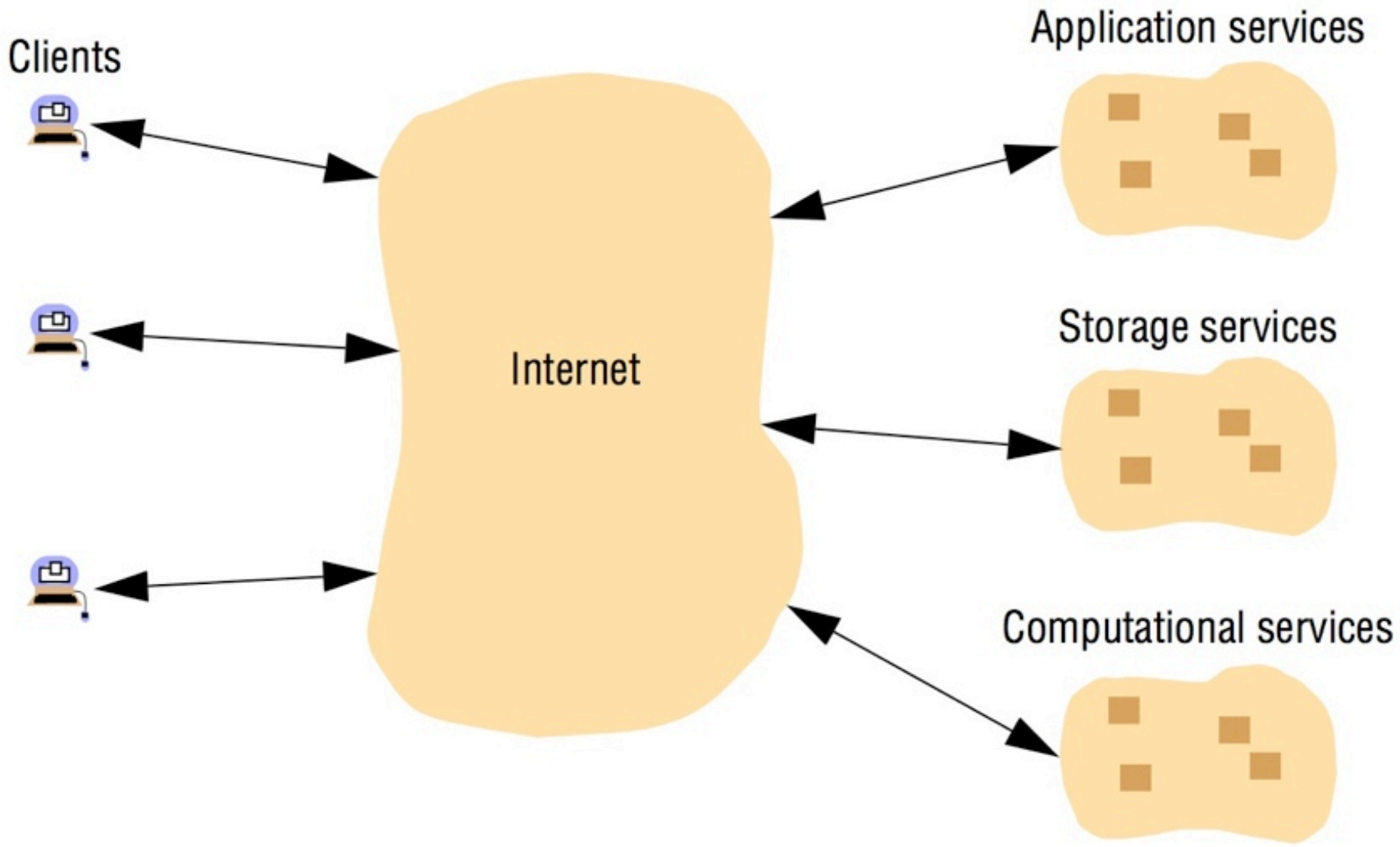


Developer	June 2020	Percent	July 2020	Percent	Change
nginx	448,673,487	36.63%	451,156,878	36.55%	-0.08
Apache	304,288,405	24.84%	314,054,523	25.45%	0.60
Microsoft	134,874,928	11.01%	140,264,332	11.36%	0.35
Google	43,449,240	3.55%	44,290,430	3.59%	0.04

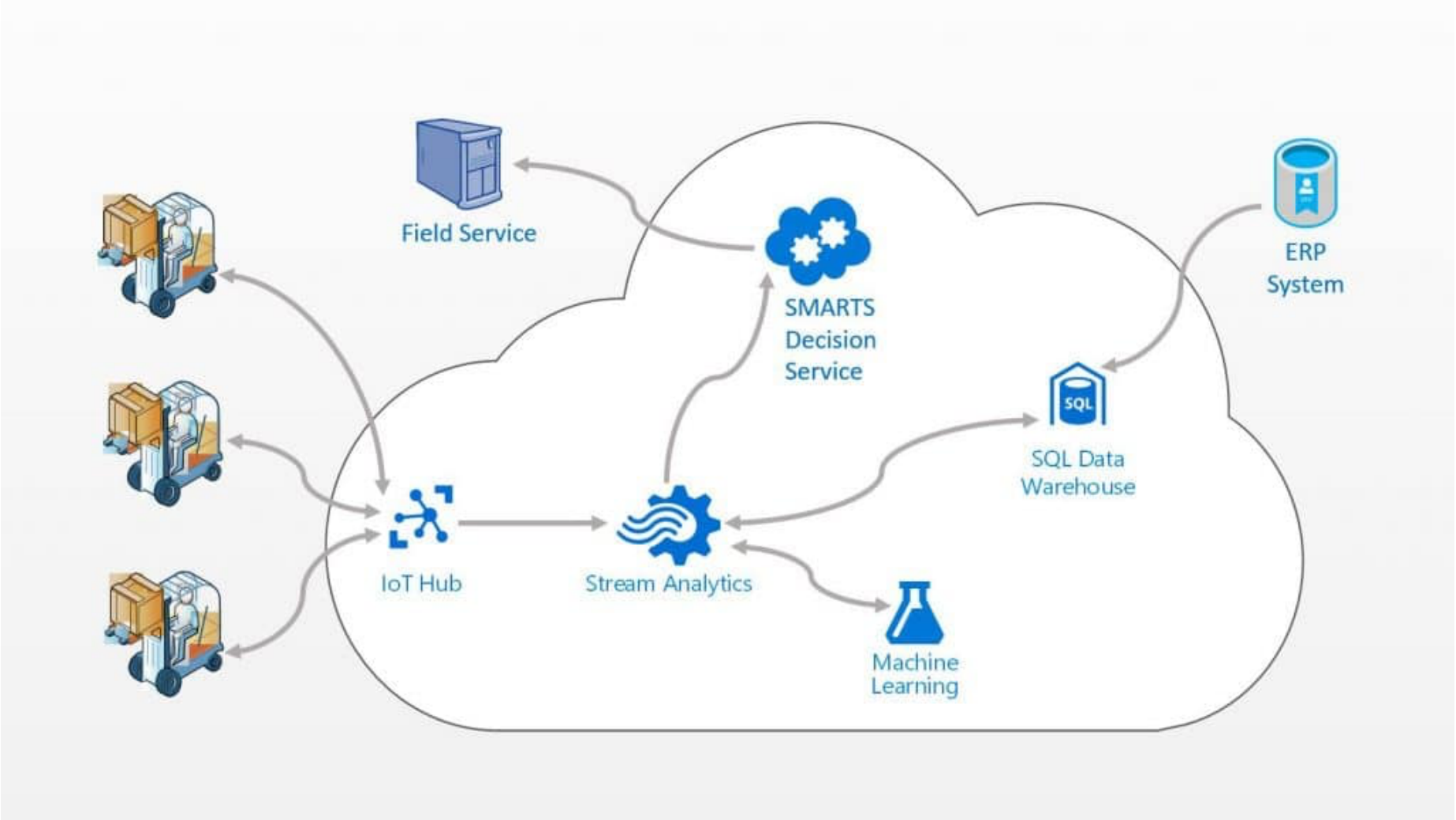
Portable and handheld devices in a network



Cloud computing



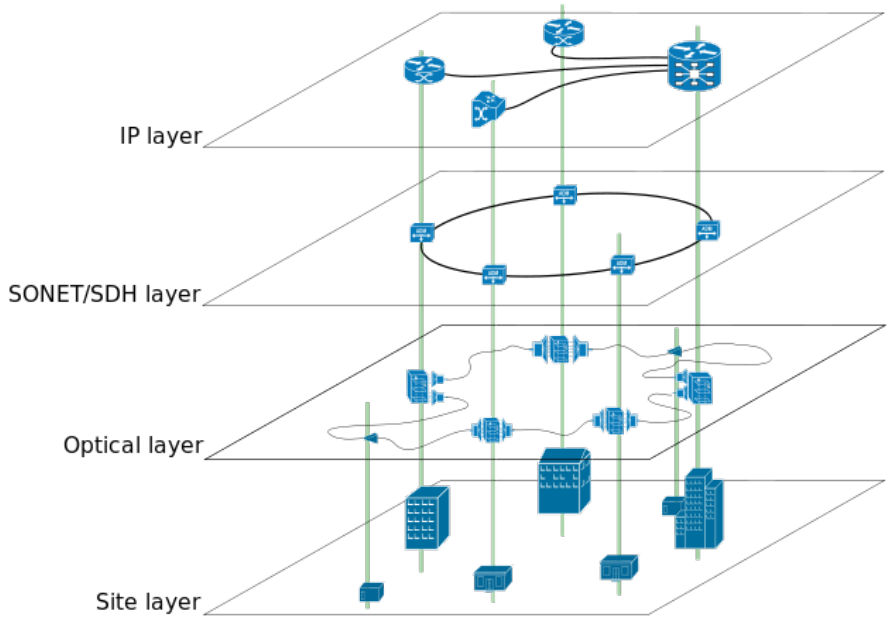
A distributed system: IoT network

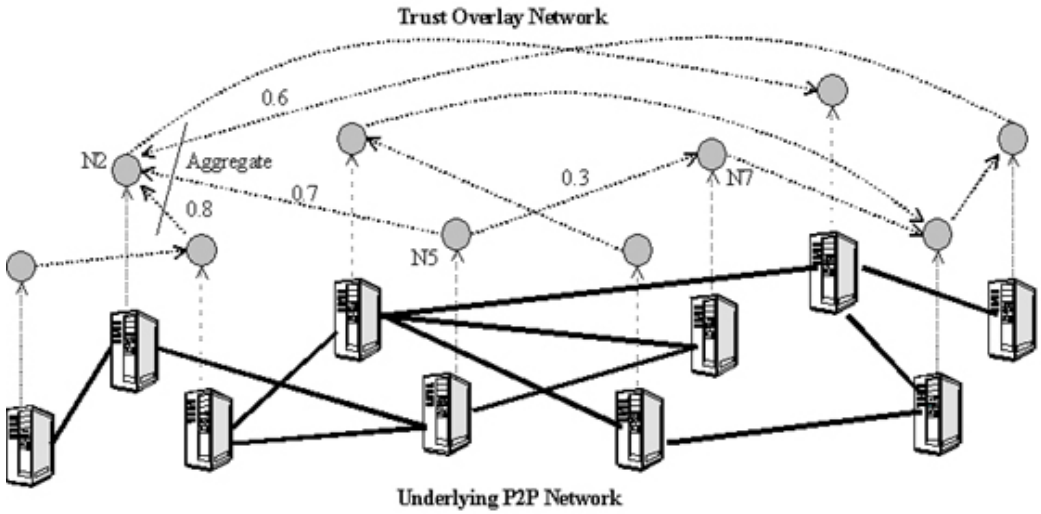


Overlay network

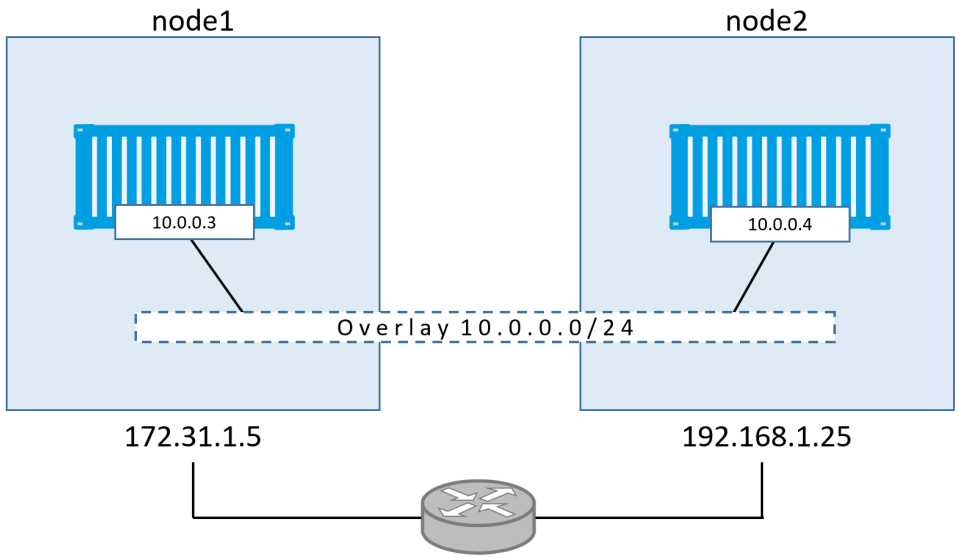
A distributed system is often organized as an overlay network, that is a network built on top of another network

It is a method of using software to create layers of network abstraction that can be used to run multiple separate, discrete virtualized network layers on top of the physical network, often providing new applications or security benefits.





Overlay networks



Each node in an overlay network communicates only with other nodes in the same network, its *neighbors*.

The set of neighbors may be dynamic, or may even be known only implicitly (i.e., it requires a lookup).

Well-known example of overlay network: peer-to-peer systems.

Overlay network types

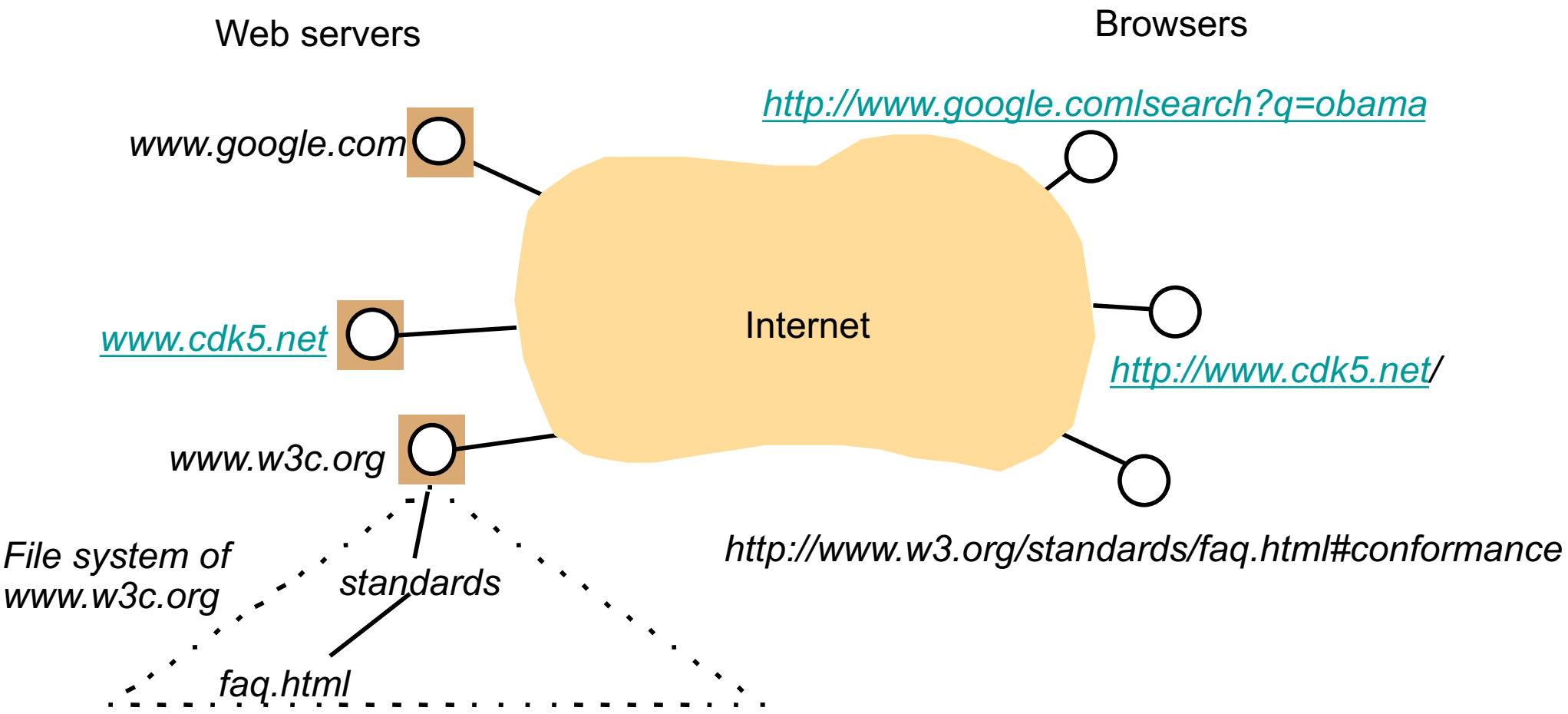
- **Structured**: each node has a well-defined set of neighbors with whom it can communicate (tree, ring).
- **Unstructured**: each node has references to randomly selected of the nodes from the system

http://blog.nigelpoulton.com/demystifying-docker-overlay-networking/

Single coherent system

- **Single system view**, behaving according to the expectations of users, that can be ubiquitous
- The collection of nodes **as a whole** operates the same, no matter where, when, and how user-system interactions take place
- Data can be replicated: **distribution transparency**
- **Middleware** to support programming distributed applications

Web servers and web browsers



Scalability

- Scalability is the capability of a system to handle a growing amount of work
- For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (hardware) are added.
- A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system.
- An algorithm, design, networking protocol, program, or other system is said to scale if it is suitably efficient and practical when applied to large situations (e.g. a large input data set, a large number of outputs or users, or a large number of participating nodes in the case of a distributed system).
- If the system fails when a quantity increases, it does not scale.

Scaling

- Horizontal scaling: you increase the number of machines
- Vertical scaling: you increase the performance of your system by increasing the performance of one machine

- Scaling is limited by Amdhal's law

The Amdhal law

The Amdhal law is a formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.
It is often used in distributed computing to predict the theoretical speedup when using multiple processors.

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where:

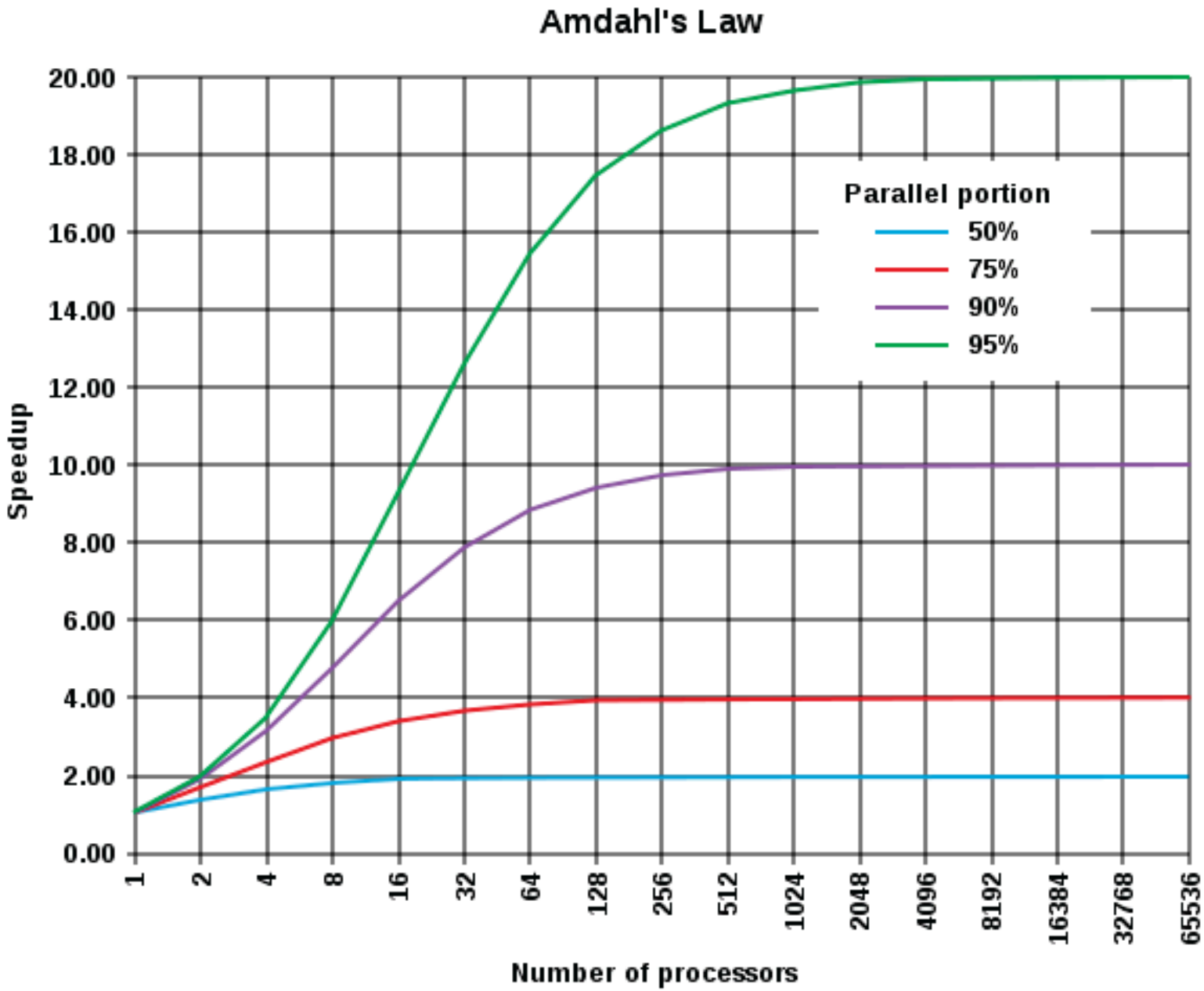
- S_{latency} is the theoretical improvement of the execution time (*latency*) of the complete task
- *s* is the improvement in execution (*speedup*) of the part of the task that benefits from the improved system resources.
- *p* is the proportion of the execution time that the part benefiting from improved resources actually occupies.

Example (source: Wikipedia, article on Amdhal law)

If a program needs 20 hours using a single processor core, and a particular part of the program which takes one hour to execute cannot be parallelized, while the remaining 19 hours ($p = 0.95$) of execution time can be parallelized, then regardless of how many processors are devoted to a parallelized execution of this program, the minimum execution time cannot be less than that critical one hour.

Hence, the theoretical speedup is limited to at most 20 times ($1/(1 - p) = 20$). For this reason, computing with many processors is useful only for highly parallelizable programs

The Amdhal law



Consequences of the Amdhal’s law

- An implication of Amdahl's law is that to speedup real applications which have both serial and parallel portions, heterogeneous computing techniques are required.
- For example, a CPU-GPU heterogeneous processor may provide higher performance and energy efficiency than a CPU-only or GPU-only processor
- Amdahl's law applies only to the cases where the problem size is fixed. In practice, as more computing resources become available, they tend to get used on larger problems (larger datasets), and the time spent in the parallelizable part often grows much faster than the inherently serial work. In this case, Gustafson's law gives a less pessimistic and more realistic assessment of the parallel performance.

Challenges of distributed systems

Heterogeneity of computing resources

Openness

Security

Scalability

Failure handling

Quality of Service

Transparency

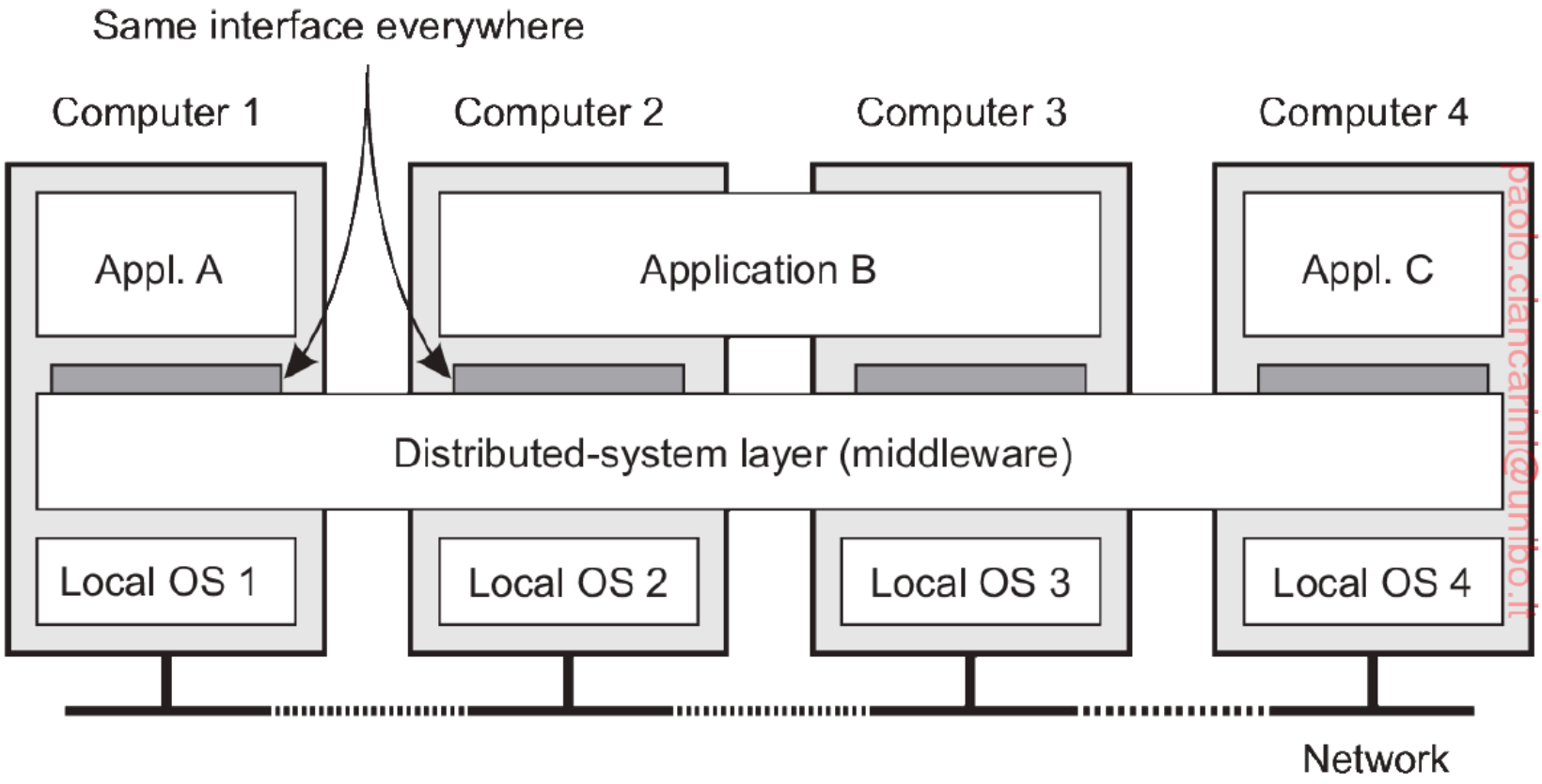
Absence of absolute time

Challenge: Heterogeneity of resources

- Networks (both physical media and protocols)
- Computer hardware;
- Operating systems;
- Programming languages;
- Implementations by different developers;
- Use cases: eg mobility vs home computing
- Systems of systems

Solution: Using middleware

1. Communications, eg. RPC
2. Transactions, eg. ACID
3. Service composition, eg. Web services
4. Reliability, eg. Horus



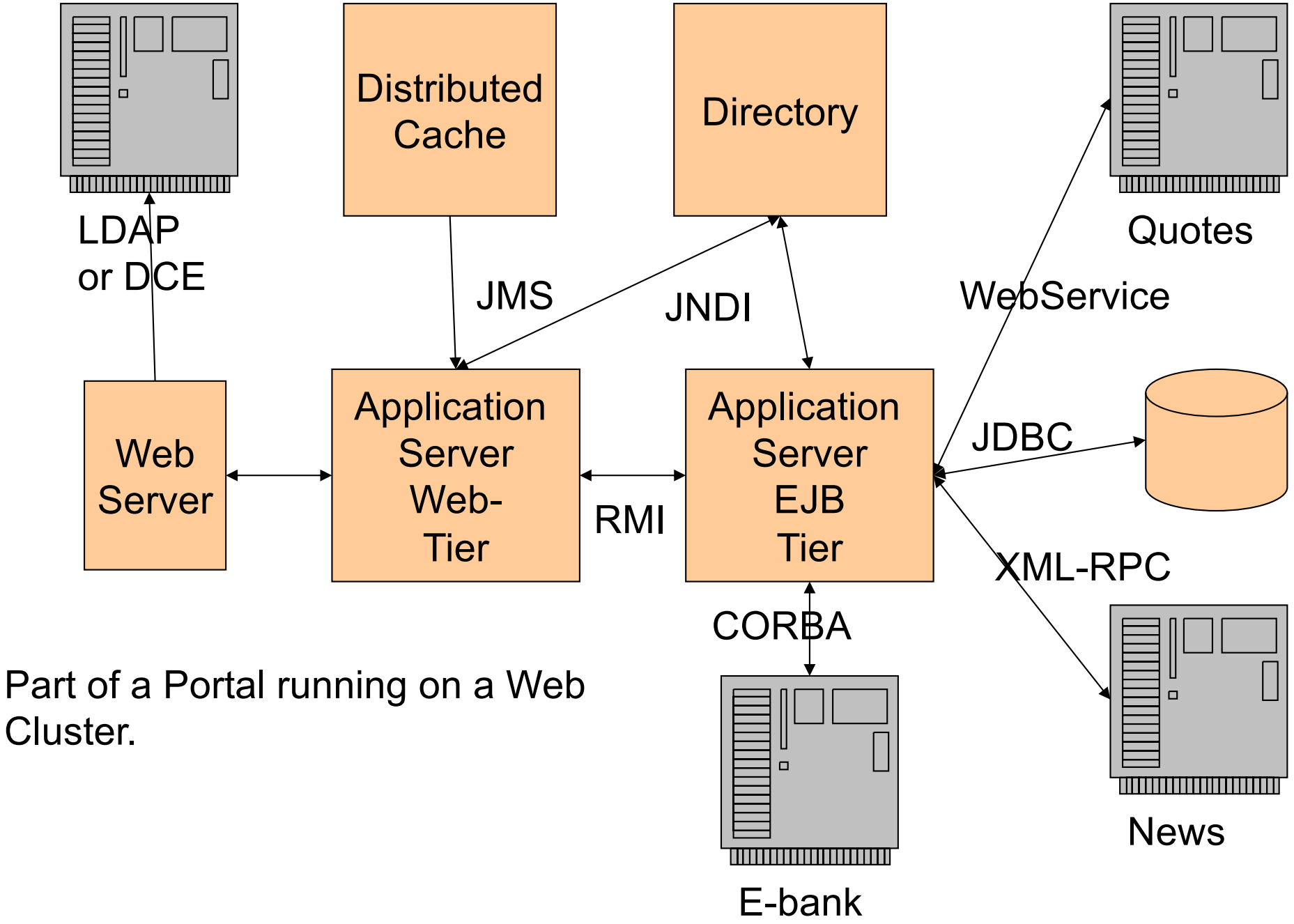
The Transparency Dogma

Middleware is supposed to hide remote-ness and concurrency by hiding distribution behind local programming language constructs

Critique: Jim Waldo, SUN

Full transparency is impossible and the price is too high

Where do we find Middleware?



Part of a Portal running on a Web Cluster.

Challenge: Openness

- Open systems are characterized by the fact that their key interfaces are published (eg.: API, application programming interface).
- Open distributed systems are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources.
- Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors.
- The conformance of each component to the published standard (we call this interoperability) must be carefully tested and verified if the system is to work correctly.

Challenge: Scalability

Scalability is the property of a system to handle a growing amount of work by adding resources to the system

- Controlling the cost of physical resources
- Controlling the performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks

Challenge: Failure handling

Computer systems sometimes fail.

When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation

- Detecting failures (eg. Checksum)
- Masking failures (eg. retransmit msgs, data replication)
- Tolerating failures
- Recovery from failures
- Redundant components

Transparencies

- *Access transparency*: local and remote resources accessed using identical operations
- *Location transparency*: resources accessed without knowledge of their physical or network location (for example, which building or IP address).
- *Concurrency transparency*: processes operate concurrently using shared resources without interference between them.
- *Replication transparency*: multiple instances of resources used to increase reliability and performance without knowledge of the replicas by users or programmers.
- *Failure transparency*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- *Mobility transparency*: allows the movement of resources and clients within a system
- without affecting the operation of users or programs.
- *Performance transparency*: allows the system to be reconfigured to improve performance as loads vary.
- *Scaling transparency*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Network transparency

- The two most important transparencies are *access* and *location* transparency: they strongly affect the usage of distributed resources
- They are referred to together as *network transparency*
- *Network transparency* is the situation in which an operating system or other service allows a user to access a resource (such as an application program or data) without the user needing to know, and usually not being aware of, whether the resource is located on the *local machine* (i.e., the computer which the user is currently using) or on a *remote machine* (i.e., a computer elsewhere on the network).

Absence of absolute time

- A distributed system has no central reference or management of time
- This is both a theoretical problem and a practical challenge:
  - distributed algorithms cannot exploit any notion of centralized clock
  - distributed applications must manage time-related operations in very specific ways

Middleware Models/Paradigms

Distributed File Systems

The Remote Procedure Call (RPC)

Distributed Objects

Distributed Documents

[All of which we return to in detail later in this course …]

Comparing DOS/NOS/Middleware

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed systems

Conclusions: trends

Distributed systems are always evolving. Some trends:

- Support user mobility
- Pervasive networking technology
- Ubiquitous computing
- Multimedia services
- Autonomous systems
- IoT

Exercise

- Name five types of hardware resource and five types of data or software resource that can usefully be shared
- Give examples of their sharing as it occurs in distributed systems.

Exercise

What is an open distributed system and what benefits does openness provide?

Exercise

What is meant by a scalable system?

Give examples

Exercise

- How might the clocks in two computers that are linked by a local network be synchronized without reference to an external time source?
- What factors limit the accuracy of the procedure you have described?
- How could the clocks in a large number of computers connected by the Internet be synchronized?
- Discuss the accuracy of that procedure.

Cristian's protocol

- The round trip time t to send a message and a reply between computer A and computer B is measured by repeated tests; then computer A sends its clock setting T to computer B. B sets its clock to T +t /2. The setting can be refined by repetition.
- The procedure is subject to inaccuracy because of contention for the use of the local network from other computers and delays in the processing the messages in the operating systems of A and B. For a local network, the accuracy is probably within 1 ms.
- For a large number of computers, one computer should be nominated to act as the time server and it should carry out Cristian's protocol with all of them. The protocol can be initiated by each in turn.
- Additional inaccuracies arise in the Internet because messages are delayed as they pass through switches in wider area networks.
- For a wide area network the accuracy is probably within 5-10 ms. These answers do not take into account the need for fault-tolerance