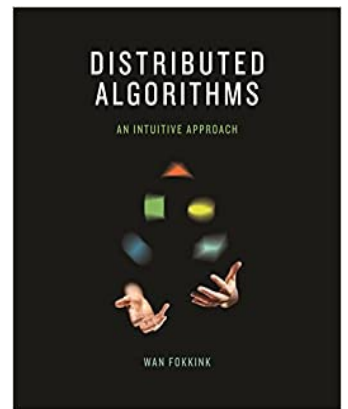
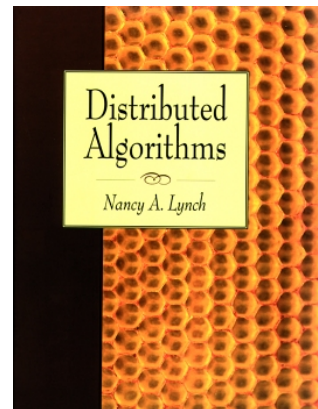
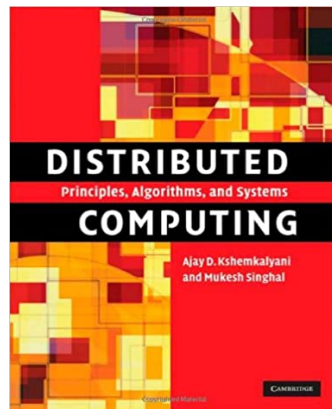
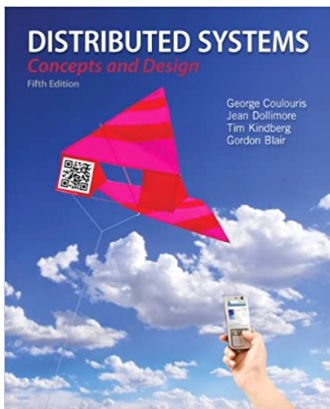


An introduction

# Course material

Some textbooks:

- **Distributed Systems: Concepts and Design** by Coulouris, Dollimore, Kindberg, Blair
- **Distributed Computing: Principles, Algorithms, and Systems** by Kshemkalyani and Singhal
- **Distributed Algorithms + Lecture notes** by Nancy Lynch
- **Distributed Algorithms: An Intuitive Approach** by Wan Fokkink



# A distributed system is..

... *“a collection of independent computers that appear to the user as one computer.”* — Andrew Tannenbaum

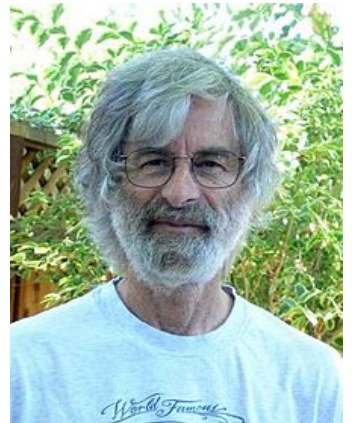
- multiple nodes communicating through a network
- trying to achieve some task together
- nodes being generic computing devices (computer, mobile phone, robot, car, ...)



# A distributed system is..

... “a system in which the failure of a computer  
you didn’t even know existed  
can render your own computer unusable.” — Leslie Lamport

- multiple nodes communicating through a network
- trying to achieve some task together
- nodes being generic computing devices  
(computer, mobile phone, robot, car, ...)



## Why make a system distributed?

1. *It's inherently distributed:*  
e.g. sending a message from your phone to your friend's phone
2. *For better reliability:*  
even if one node fails, the system as a whole keeps functioning
3. *For better performance:*  
get data from a nearby node, rather than from other side of world
4. *To solve bigger problems:*  
e.g. huge amounts of data can't fit on one machine

## Challenges & troubles

*Heterogeneity*

*Openness*

*Transparency*

*Concurrency*

*Consistency*

*Scalability*

*Security*

*Availability*

*Fault tolerance*

# *Heterogeneity*

There's a variety of

- hardware (e.g. in IoT)
- operating systems
- virtualization environments (Microsoft Hyper-V, OracleVM, docker ecosystem, ...)
- programming languages (Java, javascript, erlang, ...)
- mobile code
- networks
- data representation
- middleware (CORBA, XML/RPC, SOAP, Java RMI, ...)

It is very hard to guarantee correct interactions between all these components.

# *Openness*

Systems can be extended and their components gets re-implemented in various ways.

This requires:

- uniform communication mechanisms
- published interfaces
- open standards
- documentation

# Transparency

The system should be perceived as a whole, which requires *concealment of separation of components* (this indeed distinguishes a distributed system from a networked system).

The more transparencies are implemented, the more “distributed” is the system!

ISO Reference Model of Open Distributed Processing has defined several forms of transparencies, e.g.:

<i>Transparency</i>	<i>Main concern</i>	<i>Result</i>
<b>Access</b>	Methods used to access and manipulate resources/services.	Clients use same operations to access local and remote objects.
<b>Concurrency</b>	Concurrent access to resources/services, race conditions.	Clients are masked from effects of concurrency (e.g. linearizability).
<b>Location</b>	Location of resources/services in the distributed system.	Clients are unaware of the physical location of objects (e.g. IP).
<b>Migration</b>	Dynamic relocation of resources/services.	Clients are unaware of the dynamic relocation of resource/service.
<b>Replication</b>	Access to replicated objects, multiple responses, consistency.	Clients access replicated objects as if they were a single object. Distribution of requests, robustness to failures.
<b>Resource/liveness</b>	Resource management (e.g. deactivation/reactivation).	Clients are unaware of the deactivation/reactivation of a resource.
<b>Failure</b>	Partial failure of resource/service in a node.	Clients are unaware of a failure in a node.
<b>Federation</b>	Pan-organizational boundaries.	Clients are unaware of administrative and technology boundaries.

# Concurrency

Services and shared resources should be accessed in a concurrent way.

This poses problems w.r.t.

- **ordering of events**  
when can we say that an event has happened before another one?
- **concept of global state**  
there is no such concept in the presence of concurrent non-commutative events
- **consistency of local states**  
e.g. use of distributed commit protocols, conflict resolution, ...
- **mutual exclusion**  
e.g. locking mechanisms required
- **reaching consensus**  
e.g. use of quorums and gossiping algorithms, broadcasts, ...

# Consistency

Nodes in the systems should have a *consistent view* of the data.

In this respect challenges are posed by

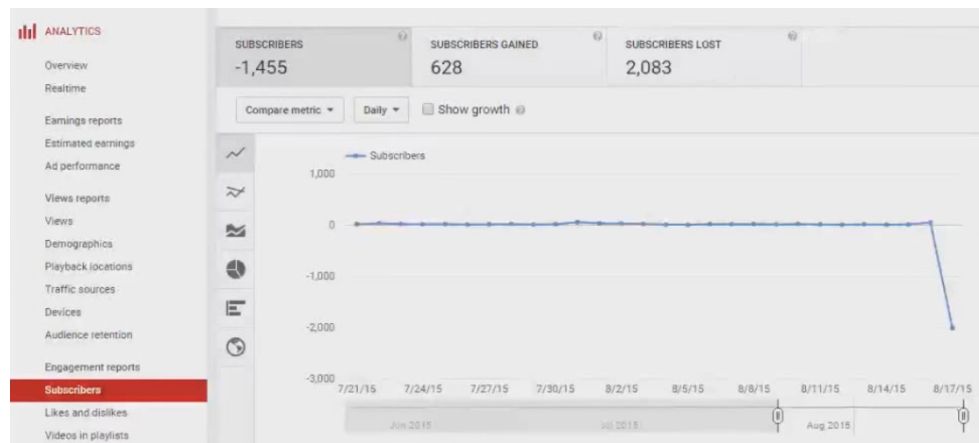
- data being **distributed** or **replicated** over several nodes
- **concurrent operations** on data (e.g. accesses and updates)
- **network failures / inefficiencies** (e.g. retry).

Several *models of consistency*, e.g.

- serializability
- read-after-write consistency
- eventual consistency
- ...

Several *techniques*, e.g.

- idempotency & commutativity
- quorums
- locking mechanisms
- optimistic transactions
- ...



# *Scalability*

As the number  $N$  of resources (e.g. files) becomes larger and larger, one requires:

- **scalability of performance**  
e.g. time to access/update/process/add/remove one resource should be *sublinear* in  $N$  (e.g.  $O(\log N)$ )
- **scalability of storage**  
e.g. size of total storage (including indexing data) should be *almost linear* in  $N$  (e.g.  $O(N \log N)$ )
- **scalability of costs**  
costs for development, deployment, maintenance, etc. should be *almost linear in*  $N$

## Growth of the Internet (computers vs web servers)

<i>Date</i>	<i>Computers</i>	<i>Web servers</i>	<i>Percentage</i>
1993, July	1 777 600	130	0.008%
1995, July	6 642 000	23 500	0.4%
1997, July	19 540 000	1 203 096	6%
1999, July	56 218 000	6 598 697	12%
2001, July	125 888 197	31 299 592	25%
2003, July	~ 200 000 000	42 298 371	21%
2005, July	353 284 187	67 571 581	19%
2014, July	996 106 380	178 558 568	18%

# Scalability

As the number  $N$  of resources (e.g. files) becomes larger and larger, one requires:

- **scalability of performance**  
e.g. time to access/update/process/add/remove one resource should be *sublinear* in  $N$  (e.g.  $O(\log N)$ )
- **scalability of storage**  
e.g. size of total storage (including indexing data) should be *almost linear* in  $N$  (e.g.  $O(N \log N)$ )
- **scalability of costs**  
costs for development, deployment, maintenance, etc. should be *almost linear in  $N$*

Growth of the Internet (computers vs web servers)

Year	Computers	Web servers	Percentage
1993, Jan	1 777 000	138	0.00008%
1995, Sep	4 643 000	21 566	0.46%
1997, Jul	15 540 000	1 203 000	0.8%
1999, Jun	52 210 000	6 538 000	1.2%
2001, Jun	135 588 100	31 293 500	2.3%
2003, Jun	280 980 000	65 200 000	2.3%
2005, Jun	592 280 000	167 000 000	2.8%
2014, Jul	1 065 560 000	370 500 000	3.5%

Question:

Consider two different network topologies, e.g. a *ring* and a *mesh*.  
Which one is better in terms of scalability of performance / storage?

# Security

The system is likely to be exposed to an *adversarial environment*.

It should then guarantee, for instance:

- **confidentiality** (e.g. against sniffing attacks)
- **integrity** (e.g. against forgery of messages and data)
- **security of mobile code** (e.g. cross-origin access policy)

# Availability

The system should always be accessible (e.g. tolerant against DOS attacks), and should function correctly.

Example: online shop wants to sell stuff 24/7 (downtime = money loss).

In practice, this requirement can only be approximated:

**uptime** = fraction of time that a service is accessible and functioning correctly

- “Two nines” = 99% up = down 3.7 days/year (e.g. batch processing, data extraction)
- “Three nines” = 99.9% up = down 8.8 hours/year (e.g. project management tools)
- “Four nines” = 99.99% up = down 53 minutes/year (e.g. video streaming, online gaming)
- “Five nines” = 99.999% up = down 5.3 minutes/year (e.g. ATM, telecommunications)

**Service-Level Objective** = uptime combined with latency constraints  
e.g. “99.9% of requests get a response within 200ms”

# *Fault tolerance*

We want the system as a whole to continue working *even when a fraction of its components is faulty*, e.g. \*

- messages may be lost
- nodes may crash, or temporarily freeze, while others keep running
- network may be temporarily partitioned

Handling principles:

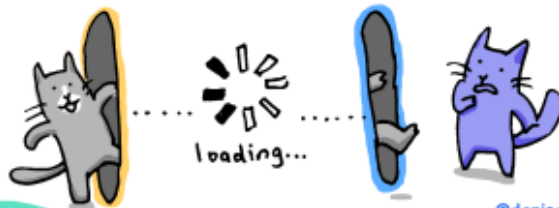
- *detecting*  
not easy, often impossible, but if detected a failure can be managed to some extent
- *masking*  
e.g. by hiding the crashed node, or retransmitting a lost message (e.g. TCP)
- *tolerating*  
e.g. by ignoring and reporting an abnormal behaviours (e.g. delivery failure in email servers)
- *recovery*  
e.g. by logs and rollbacks (e.g. DBMS)
- *redundancy*  
e.g. by keeping replicas (but this raises a consistency problem...)

\* “*All these problems will happen, simultaneously and non-deterministically*” — Murphy

① The network is reliable



② Latency is ZERO



③ Bandwidth is infinite



⑧ The network is homogeneous



# the 8 Fallacies of Distributed Computing

Originally formulated by L. Peter Deutsch & colleagues at Sun Microsystems in 1994; #8 added in 1997 by James Gosling

④ The network is secure



⑦ Transport costs \$0



⑥ There is only one administrator



⑤ Topology doesn't change



# Examples

## The web as a distributed system

The image shows a screenshot of the Wikipedia article titled "Distributed computing". The article text is partially visible, defining distributed computing as a field of computer science that studies distributed systems. A diagram is overlaid on the right side of the page, consisting of two boxes: "Client (your browser)" on the left and "Server (en.wikipedia.org)" on the right. A large arrow points from the client box to the server box, with the text "GET wiki/Distributed\_computing" written along the arrow's path. The Wikipedia article content includes a table of contents with sections like Introduction, Parallel and distributed computing, History, Architectures, Applications, Examples, Theoretical foundations, Properties of distributed systems, See also, Notes, References, Further reading, and External links.

Client (your browser)

Server (en.wikipedia.org)

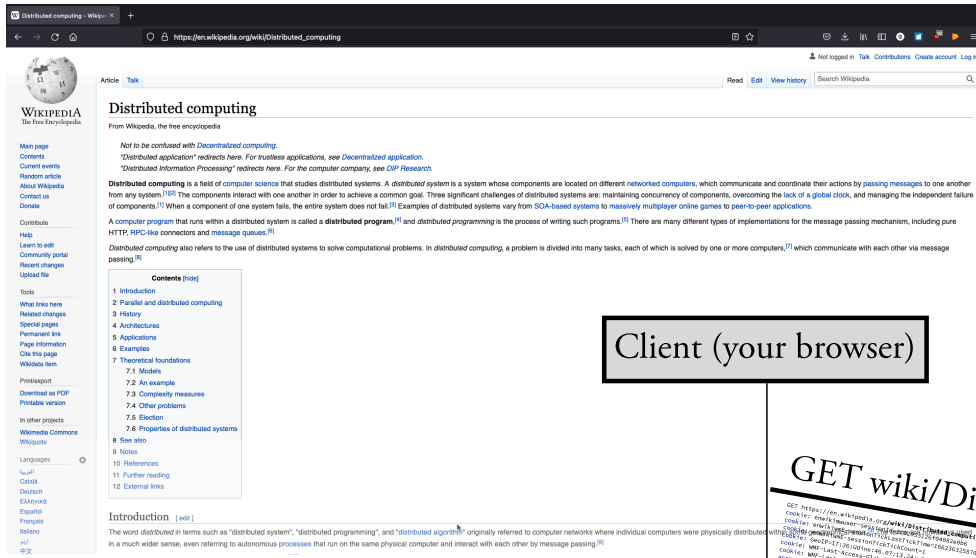
GET wiki/Distributed\_computing

# GET wiki/Distributed

GET https://en.wikipedia.org/wiki/Distributed HTTP/2.0  
cookie: enwikimwuser-sessionId=2c02953126f04082e0b6  
cookie: enwikiwmE-sessionTickLastTickTime=1662363258761  
cookie: enwikiwmE-sessionTickTickCount=1  
cookie: GeoIP=IT:36:Udine:46.07:13.24:v4  
cookie: WMF-Last-Access-Global=05-Sep-2022  
cookie: WMF-Last-Access=05-Sep-2022  
accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
accept-encoding: gzip, deflate, br  
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7)  
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.6.1 Safari/605.1.15  
accept-language: en-us

# Examples

## The web as a distributed system



Client (your browser)

Server (en.wikipedia.org)



<!DOCTYPE html> <html>

HTTP/2.0 200  
date: Mon, 05 Sep 2022 07:34:17 GMT  
server: mw1351.eqiad.wmnet  
x-content-type-options: nosniff  
content-language: en  
vary: Accept-Encoding, Cookie, Authorization  
last-modified: Sun, 04 Sep 2022 14:38:04 GMT  
content-type: text/html; charset=UTF-8  
content-encoding: gzip  
age: 1008  
x-cache: cp6014 miss, cp6015 hit/6  
x-cache-status: hit-front  
...  
content-length: 44497

decoded gzip:

```
<!DOCTYPE html>  
<html class="client-nojs" lang="en" dir="ltr">  
<head>  
  <meta charset="UTF-8"/>  
  <title>Distributed computing - Wikipedia</title>  
  ...  
</head>  
<body class="...">  
  ...  
</body>  
</html>
```

# Examples

## The web as a distributed system

The screenshot shows the Wikipedia article for "Distributed computing". The article text includes:

Not to be confused with [Decentralized computing](#).  
"[Distributed application](#)" redirects here. For [frustrated applications](#), see [Decentralized application](#).  
"[Distributed Information Processing](#)" redirects here. For the computer company, see [DIP Research](#).

**Distributed computing** is a field of [computer science](#) that studies distributed systems. A distributed system is a system whose components are located on different [networked computers](#), which communicate and coordinate their actions by [passing messages](#) to one another from any system.<sup>[1][2]</sup> The components interact with one another in order to achieve a common goal. Three significant challenges of distributed systems are: maintaining concurrency of components, overcoming the lack of a [global clock](#), and managing the independent failure of components.<sup>[1][2]</sup> When a component of one system fails, the entire system does not fail.<sup>[1]</sup> Examples of distributed systems vary from [SOA-based systems](#) to [massively multiplayer online games](#) to [peer-to-peer applications](#).

A computer program that runs within a distributed system is called a [distributed program](#),<sup>[1]</sup> and distributed programming is the process of writing such programs.<sup>[1]</sup> There are many different types of implementations for the message passing mechanism, including pure [HTTP](#), [RPC](#) like connectors and message queues.<sup>[1]</sup>

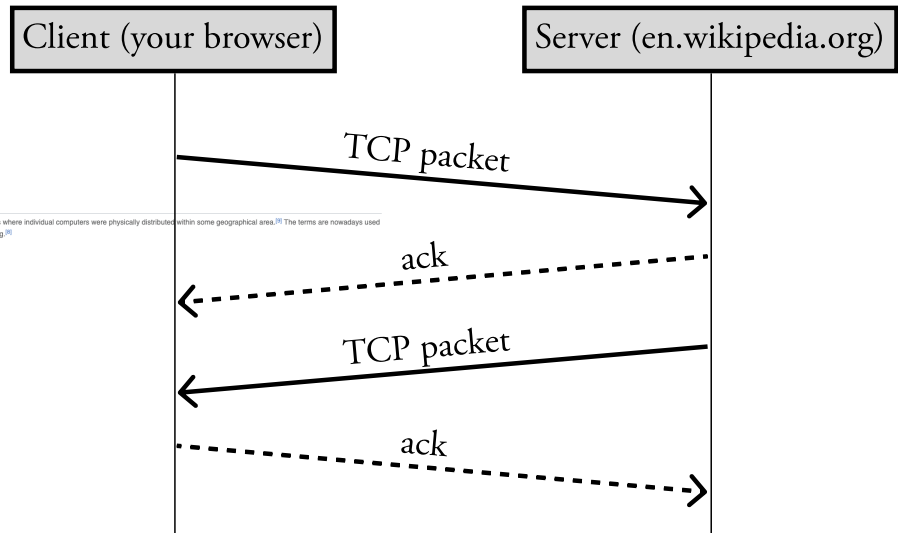
Distributed computing also refers to the use of distributed systems to solve computational problems. In distributed computing, a problem is divided into many tasks, each of which is solved by one or more computers,<sup>[1]</sup> which communicate with each other via message passing.<sup>[1]</sup>

**Contents** [hide]

- Introduction
- Parallel and distributed computing
- History
- Architectures
- Applications
- Examples
- Theoretical foundations
  - Models
  - An example
  - Complexity measures
  - Other problems
  - Election
  - Properties of distributed systems
- See also
- Notes
- References
- Further reading
- External links

**Introduction** [edit]

The word distributed in terms such as "distributed system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area.<sup>[1]</sup> The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing.<sup>[1]</sup>



Protocol	Length	Info
TLSv1.3	146	Change Cipher Spec, Application Data
TLSv1.3	1490	Application Data, Application Data, Application Data, Application Data, Application Data, Application Data
TLSv1.3	81	Application Data
TLSv1.3	1212	Application Data
TCP	66	55785 → 443 [ACK] Seq=2608 Ack=55367 Win=129920 Len=0 TSval=1947913124 TSecr=639616138
TLSv1.3	1266	Application Data
TCP	66	55785 → 443 [ACK] Seq=2608 Ack=56567 Win=131072 Len=0 TSval=1947913124 TSecr=639616138
TLSv1.3	353	Application Data
TCP	66	55786 → 443 [ACK] Seq=2037 Ack=4135 Win=130752 Len=0 TSval=671597563 TSecr=1054372489
TLSv1.3	353	Application Data
TCP	66	55786 → 443 [ACK] Seq=2037 Ack=4422 Win=130752 Len=0 TSval=671597563 TSecr=1054372489
TCP	66	443 → 55786 [ACK] Seq=4422 Ack=2037 Win=41984 Len=0 TSval=1054372490 TSecr=671597547
TLSv1.3	118	Application Data
TLSv1.3	1351	Application Data
TCP	66	55786 → 443 [ACK] Seq=2037 Ack=4474 Win=131008 Len=0 TSval=671597564 TSecr=1054372490
TLSv1.3	97	Application Data
TLSv1.3	1388	Application Data
TCP	1514	443 → 55786 [ACK] Seq=7081 Ack=2037 Win=42496 Len=1448 TSval=1054372491 TSecr=671597547 [TCP segment of
TLSv1.3	1210	Application Data
TCP	1514	443 → 55786 [ACK] Seq=9673 Ack=2037 Win=42496 Len=1448 TSval=1054372491 TSecr=671597547 [TCP segment of
TLSv1.3	1184	Application Data
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=5759 Win=129728 Len=0 TSval=671597564 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=7081 Win=129728 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=8529 Win=129600 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=9673 Win=129920 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=11121 Win=129600 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=12239 Win=128448 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	443 → 55786 [ACK] Seq=12239 Ack=2068 Win=42496 Len=0 TSval=1054372549 TSecr=671597564
TCP	66	[TCP Retransmission] 443 → 55784 [FIN, ACK] Seq=1829 Ack=1484 Win=68352 Len=0 TSval=2196911746 TSecr=131
TCP	66	[TCP Dup ACK 114#1] 55784 → 443 [ACK] Seq=1484 Ack=1830 Win=131072 Len=0 TSval=1310589679 TSecr=21969115
TLSv1.3	140	Application Data
TCP	66	443 → 55783 [ACK] Seq=13528 Ack=20276 Win=109568 Len=0 TSval=1578706227 TSecr=4284084843
TLSv1.3	140	Application Data
TCP	66	55783 → 443 [ACK] Seq=20276 Ack=13602 Win=130944 Len=0 TSval=4284085021 TSecr=1578706396
TLSv1.3	158	Application Data
TCP	66	443 → 55783 [ACK] Seq=13602 Ack=20368 Win=109568 Len=0 TSval=1578706787 TSecr=4284085403
TLSv1.3	173	Application Data

## Other examples of distributed systems

- The information society                      WWW, search engines (Google), cloud computing (AWS),  
Wikipedia, social networking (e.g. Facebook, MySpace)

Google web search engine indexes the whole content of the internet:  
- more than  $16 \cdot 10^9$  web pages

It includes:

- a physical infrastructure of *thousands of computers*, located all around the world
- a *distributed file system*, heavily optimized
- a service implementing *distributed locking, consensus, and conflict resolution*
- a programming model supporting *parallel and distributed computations*

(Google)

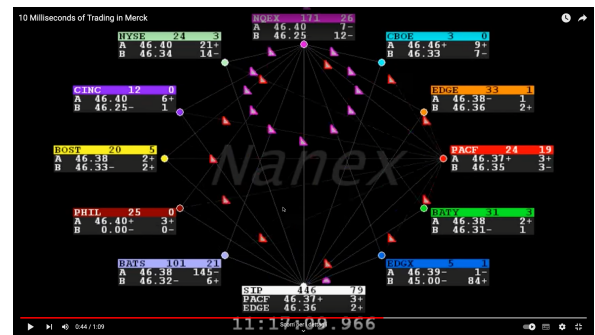
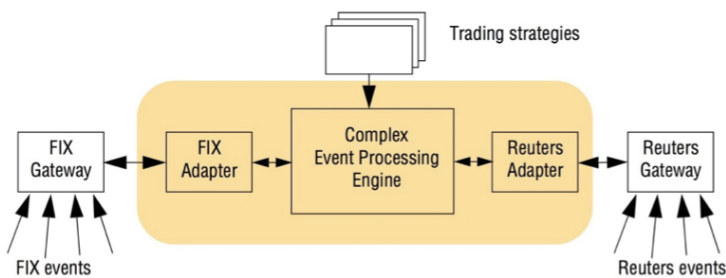
## Other examples of distributed systems

- The information society  
WWW, search engines (Google), cloud computing (AWS),  
Wikipedia, social networking (e.g. Facebook, MySpace)
- 
- Finance & commerce  
e-commerce (e.g. Amazon, eBay), PayPal, trading,  
online banking

# trading,

Distributed *event-based* system, programmed by Event Processing Languages

- events are exchanged in a *standard format and protocol* (Financial Information eXchange, Reuter events)
- *real-time access* to wide range of information sources (e.g. share prices, trends, economics, political developments)
- *automated monitoring* of events and trading applications



<https://www.youtube.com/watch?v=L5cZaIZ5bWc>

## Other examples of distributed systems

• The information society	WWW, search engines (Google), cloud computing (AWS), Wikipedia, social networking (e.g. Facebook, MySpace)
• Finance & commerce	e-commerce (e.g. Amazon, eBay), PayPal, trading, online banking
• Creative industries & entertainment	online gaming, audio and video streaming (Netflix), user-generated content (e.g. YouTube, Flickr)
• Healthcare	health informatics, clinic data, monitoring (e.g. Immuni)
• Education	e-learning, virtual learning environments, distance learning
• Transport & logistics	GPS in-route finding systems, map services (e.g. Google Maps)
• Science	distributed databases (e.g. GRID - Global Research Identifier Database, ROR, ...)
• Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

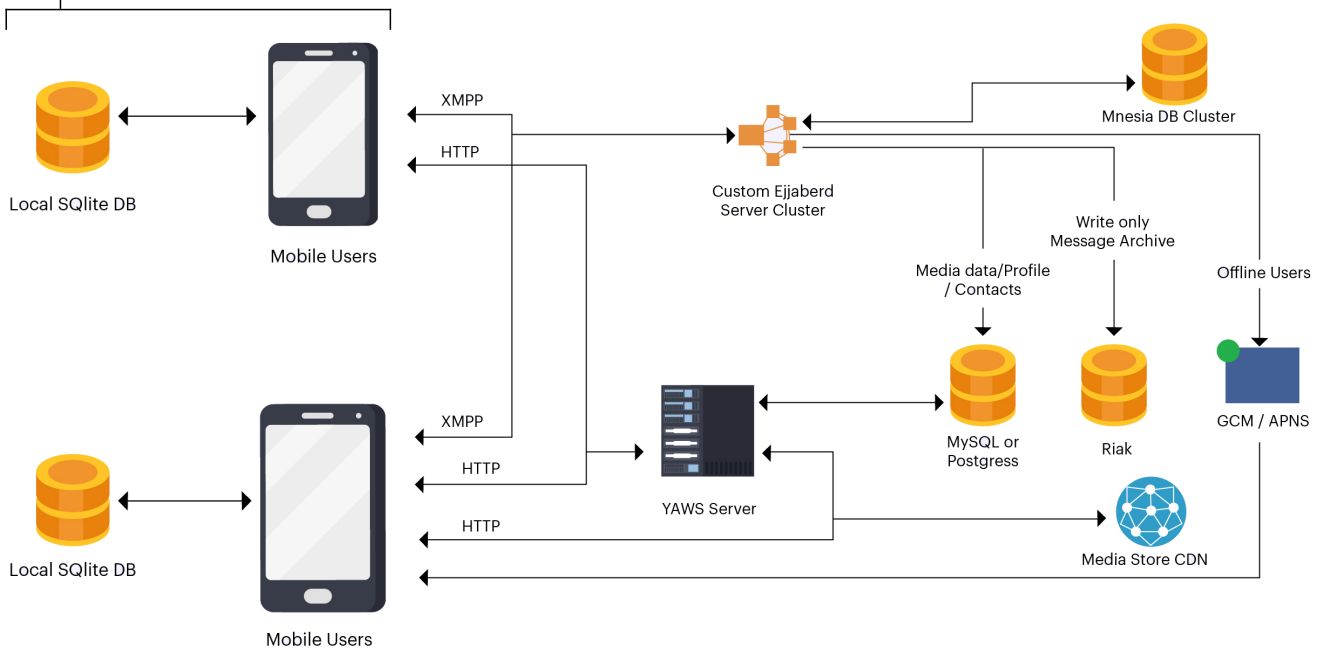
## Generations of distributed systems

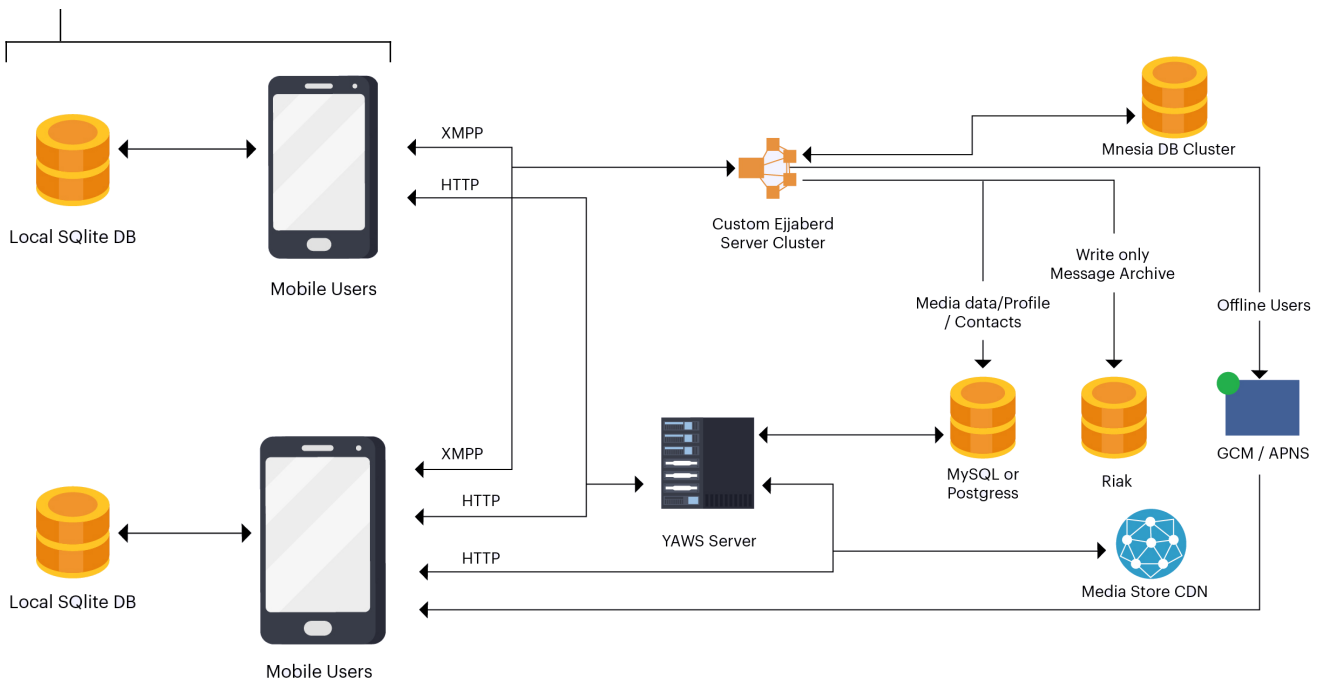
<i>Generation</i>	<i>Scale</i>	<i>Heterogeneity</i>	<i>Openness</i>	<i>Quality of service</i>
<b>Early generation</b> LANs, shared printers, file servers, emails	Small	Limited (typically homogeneous configurations)	Not a priority	In its infancy
<b>Internet era</b> WWW, networks of networks	Large	Significant in terms of platforms, languages, and middleware	Significant, with a range of standards introduced	Significant, with a range of services introduced
<b>Contemporary D.S.</b> mobiles systems, cloud computing, virtualised systems	Huge	High (radically different types of components)	Major research challenge, with existing standards not able to embrace the complexity of the systems	Major research challenge, with existing services not able to embrace the complexity of the systems

# A glimpse at Whatsapp architecture

2.5 billion active users, 100 billion messages sent every day ...with just 50 engineers!

- frontend:
- *mobile code* on Android (Java), iOS (Swift), Windows phones (C#)
  - *web app code* with Javascript / HTML / CSS
  - *desktop app code* on MacOS (Swift / Objective-C), Windows (C / C# / Java)
  - SQLite DB for storing chats





backend:

- Erlang (main programming language on servers)
- XMPP (open standard chat protocol for exchanging small XML data)
- Ejaberd (open-source XMPP server, written in Erlang)
- Mnesia (DBMS, written in Erlang)
- Riak (distributed NoSQL key-value DBMS)
- YAWS (web server in Erlang, used for managing multimedia data)
- GCM (Google Cloud Messaging) / APNS (Apple Push Notification Service)

# An introduction

[back]

## Course material

Some textbooks:

- **Distributed Systems: Concepts and Design** by Coulouris, Dollimore, Kindberg, Mei
- **Distributed Computing: Principles, Algorithms, and Systems** by Kulkarni and Singhal
- **Distributed Algorithms - Lecture notes** by Nancy Lynch
- **Distributed Algorithms: An Iterative Approach** by Wan Fokkink



## A distributed system is...

...*"a system in which the failure of a computer you didn't even know existed can render your own computer unusable"*...

— Leslie Lamport

- multiple nodes communicating through a network
- trying to achieve some task together
- nodes being generic computing devices (computer, mobile phone, robot, etc...)



### Why make a system distributed?

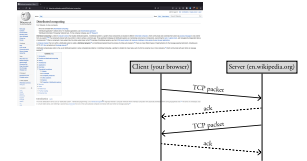
1. **It's inherently distributed**  
e.g. sending a message from your phone to your friend's phone
2. **For better reliability**  
even if one node fails, the system as a whole keeps functioning
3. **For better performance**  
get data from a nearby node, rather than from other side of world
4. **To solve bigger problems**  
e.g. huge amounts of data can't fit on one machine

### Challenges & troubles

Homogeneity	Openness	Transparency
<p><b>Homogeneity</b></p> <p>Uniformity in the way services are provided across the system.</p> <p>Uniformity in the way services are provided across the system.</p>	<p><b>Openness</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>	<p><b>Transparency</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>
<p><b>Consistency</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>	<p><b>Consistency</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>	<p><b>Scalability</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>
<p><b>Security</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>	<p><b>Availability</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>	<p><b>Fault tolerance</b></p> <p>Ability to interact with other systems.</p> <p>Ability to interact with other systems.</p>

## Examples

The web as a distributed system



### Other examples of distributed systems

- **The information society**: WWW, search engines (Google), cloud computing (AWS), Wikipedia, social networking (e.g. Facebook, MySpace)
- **Finance & commerce**: e-commerce (e.g. Amazon, eBay), PayPal, trading, online banking
- **Creative industries & entertainment**: online gaming, audio and video streaming (Netflix), user-generated content (e.g. YouTube, Flickr)
- **Healthcare**: health informatics, clinical data, monitoring (e.g. Fitbit)
- **Education**: e-learning, virtual learning environments, distance learning
- **Transport & logistics**: GPS in road finding systems, map services (e.g. Google Maps)
- **Science**: distributed databases (e.g. GRID - Global Research Identifier Database, BOSS...)
- **Environmental management**: sensor technology to monitor earthquakes, floods or tsunamis

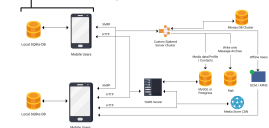
### Generations of distributed systems

Generations	Scale	Heterogeneity	Openness	Quality of service
<b>1st generation</b> Mainframes, distributed systems, the client-server model	Small	Low	Low	High reliability
<b>2nd generation</b> WWW, e-commerce, distributed systems	Large	High	High	High reliability, high availability
<b>3rd generation</b> Cloud computing, virtualized services	Very large	Very high	Very high	High reliability, high availability, high performance

### A glimpse at Whatsapp architecture

2.5 billion active users, 100 billion messages sent every day... with just 50 engineers

- multiplatform (Android, iOS, Symbian, Windows phone (CE))
- end-to-end encryption (HELM, CAS)
- desktop app (not on Mac OS, Symbian, Windows CE / C# / Java)
- SQLite DB for storing chats



- back-end:
- Erlang (multi programming language on servers)
  - SMTP (open standard chat protocol for exchanging mail XML data)
  - LDAP (open source SMTP server, written in Erlang)
  - Maria DBMS (written in Erlang)
  - Berk DBMS (NoSQL, key-value DBMS)
  - FUSE (web server in Erlang, used for managing multimedia data)
  - GCM (Google Cloud Messaging)
  - MMS (Apple Push Notification Service)

