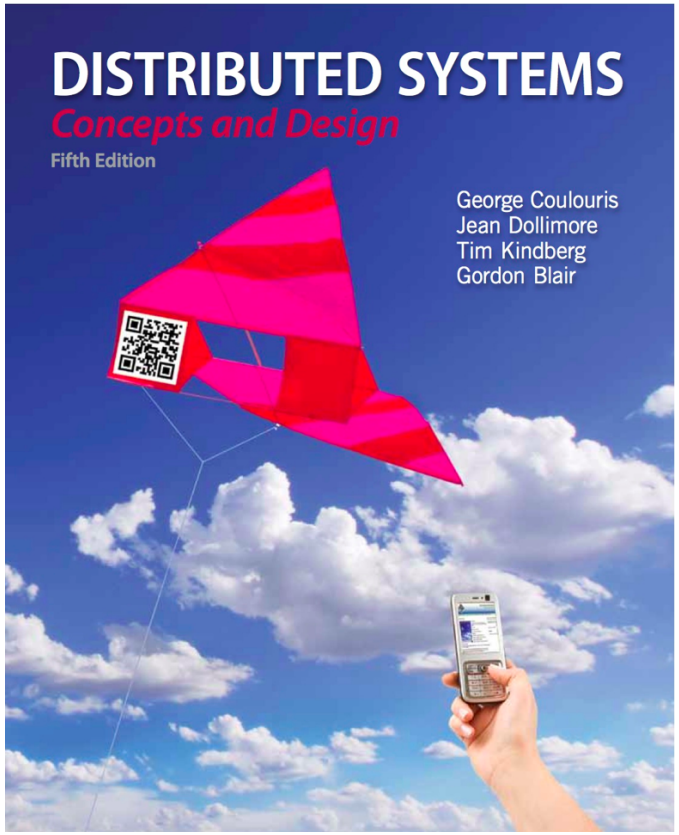
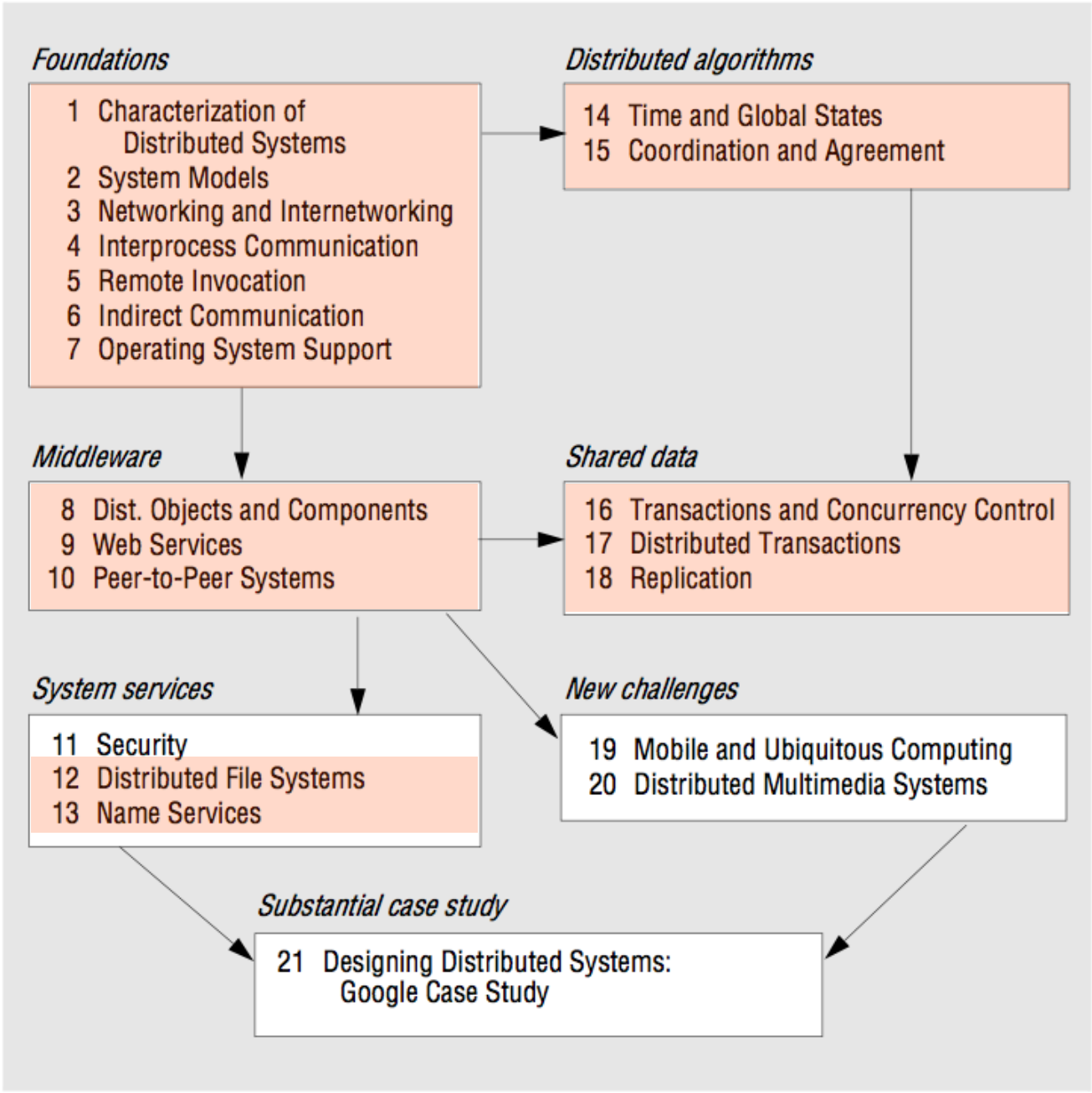
Book and Organization of the course

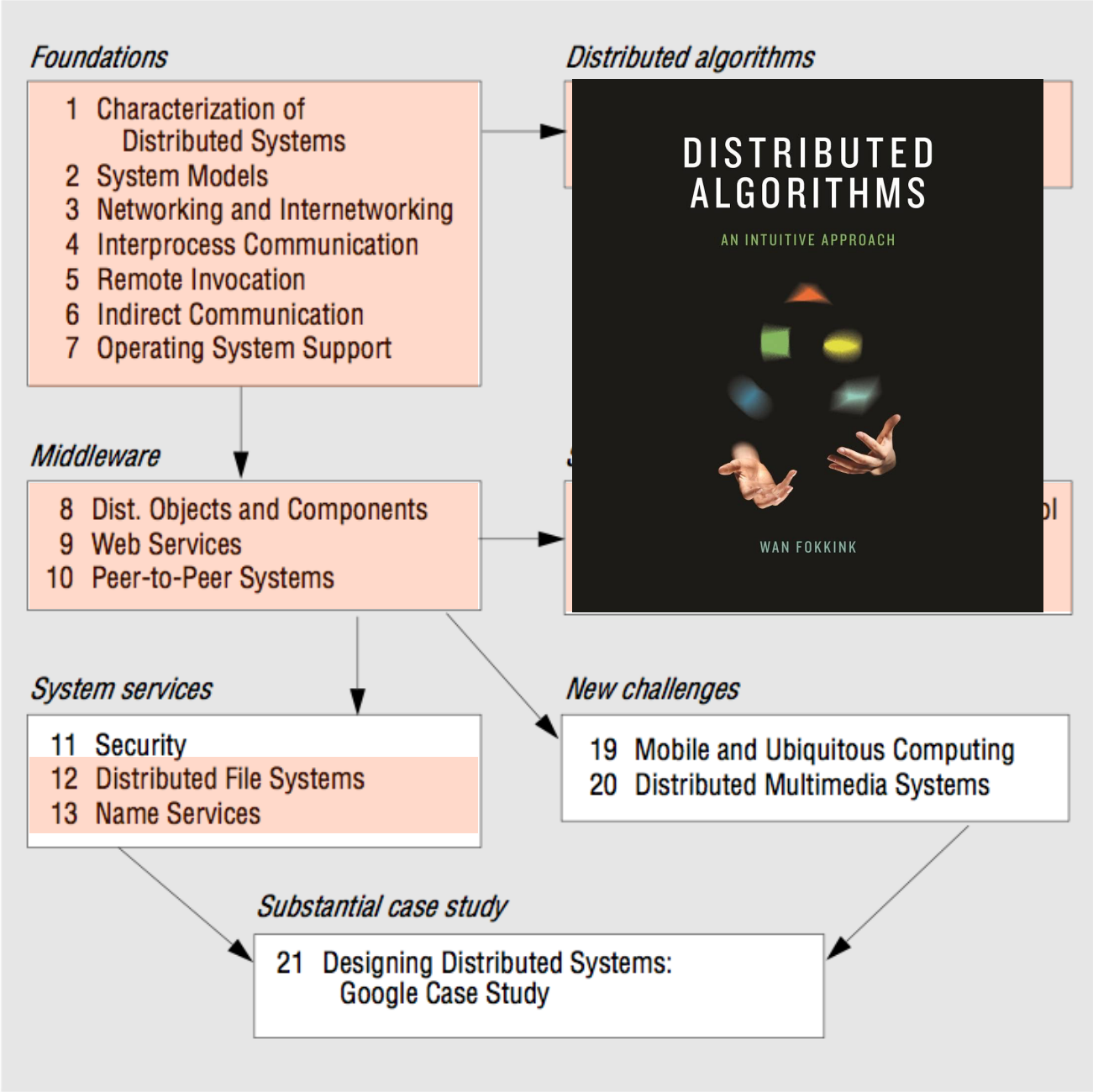
Book: Coulouris, Dollimore, Kindberg, Blair: Distributed Systems - Concepts and Design - 5 Edition. Addison Wesley, 2012



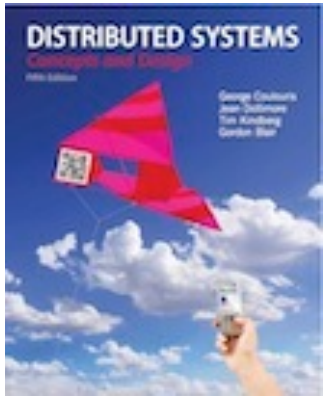


Book: Wan Fokkink: Distributed Algorithms - An intuitive approach - MIT Press, 2018

This book will be used as main reference for distributed algorithms (Coulouris' is quite "high-level" on this part)



Chapter 1
Characterization of Distributed Systems



From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2012

Modified by M. Miculan

Networking and Parallel Computing

Computer networking

- Hardware that connects computers
- Software that sends/receives messages from one computer to another, which might be on different networks (end to end delivery)
- Goal is to transmit messages reliably and efficiently

Parallel Computing

- Multiple homogeneous processors in “one” computer
  - Shared or distributed memory
  - Range from multi-processor systems to GPUs to clusters
- Goal is to execute a program faster by division of labor

Distributed Computing

- Networked computers that could be far apart
  - rely on computer networking - ANY kind of networking

- **Definition:**
  A distributed system is a system in which hardware or software components communicate and coordinate *only* by passing messages
  - No shared memory
  - Communication channels are important

- **Prime motivation for distributed system is to share (access/provide) *distributed* resources,**
  not to parallelize computation

Issues in Distributed Computing

- **Concurrent execution of processes**, with access to shared resources
- Processes have **partial/incomplete/inconsistent views** of resources and of system state (actually, there is no “global system state”)
- **No global clock** for coordination: timing problems (what comes before what?)
- More components, more **independent failures**: each component can fail independently leaving others running. Even failure detection alone is a problem

All these issues are usually ignored in local, stand-alone computational models.

Figure 1.1 (see book for the full text)
Selected application domains and associated networked applications

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

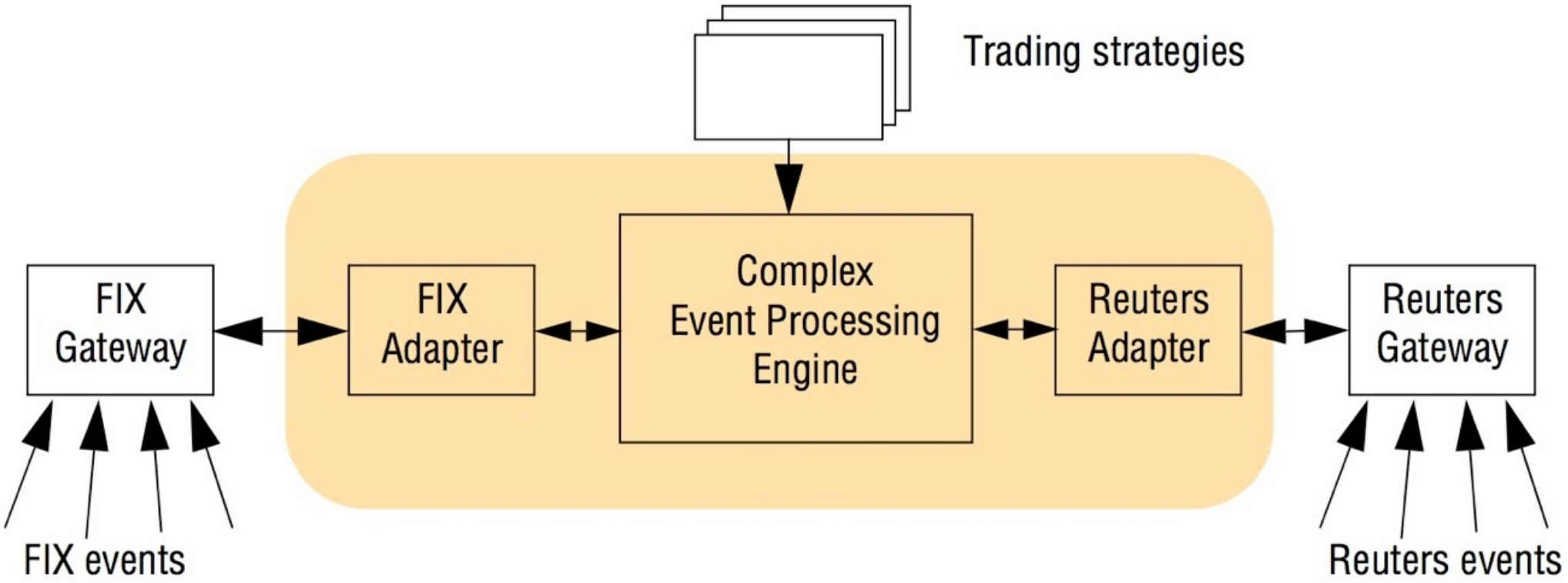
Example: Google

- A web search engine indexes the whole content of the internet
  - More than $16*10^9$ web pages
- Google infrastructure include:
  - an underlying **physical infrastructure of thousands of computers, located all around the world**
  - a **distributed file system**, heavily optimized
  - a **distributed structured storage system**
  - a lock service implementing **distributed locking and agreement**
  - a **programming model** supporting the management of very large parallel and distributed computations

Example: financial trading system

- Real time access to wide range of information sources (e.g. share prices, trends, economics and political developments,...)
- (Similar situations in logistics, military, disaster monitoring...)
- Automated monitoring of events and trading applications
- *Distributed event-based system*, programmed by *Event Processing Languages*
- Events are exchanged in a standard format and protocol (Financial Information eXchange, or Reuter events)

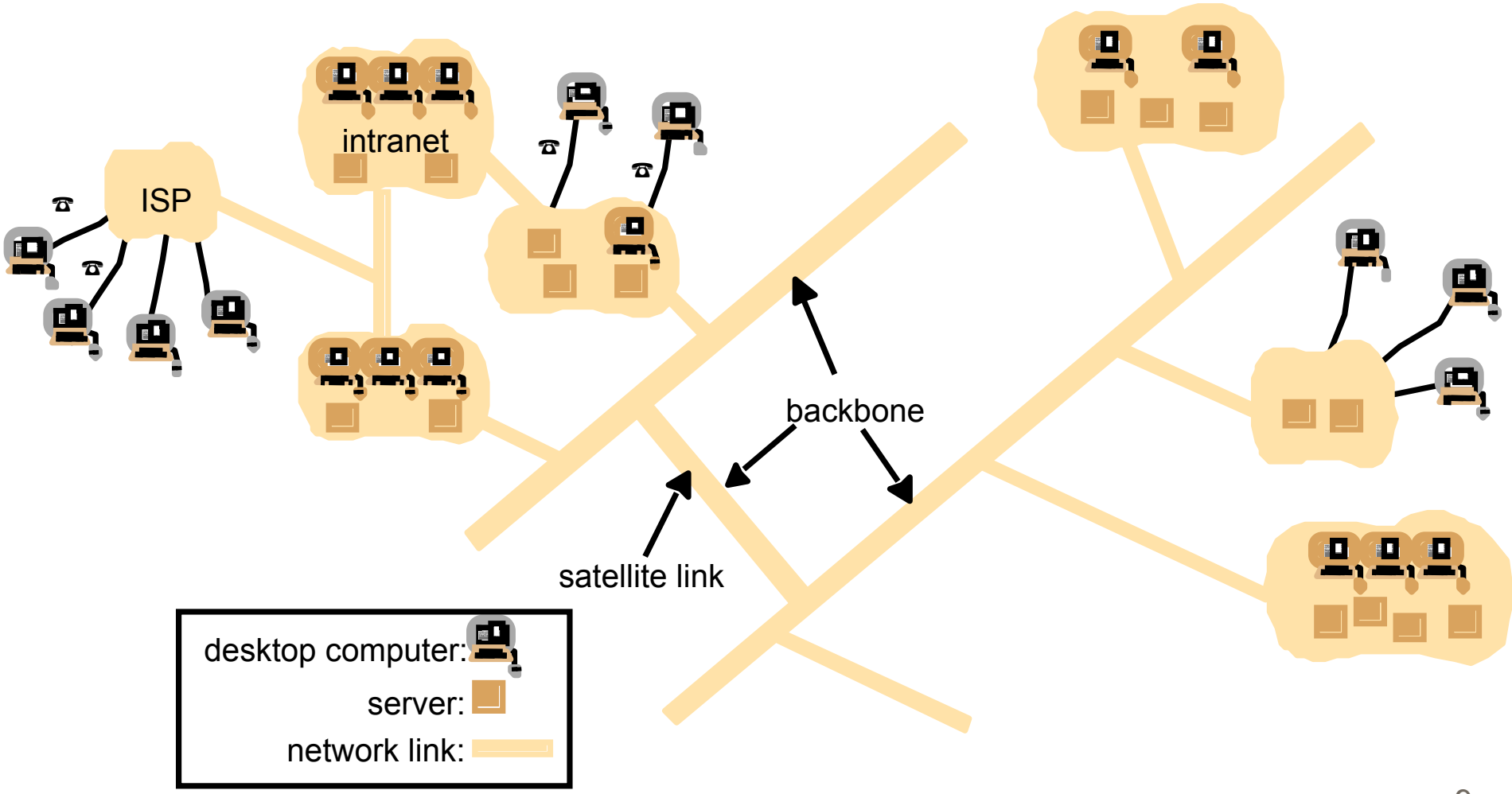
Example: financial trading system



Example animation: https://www.youtube.com/watch?v=L5cZaIZ5bWc

Trends in distributed systems:
Emergence of pervasive network technology

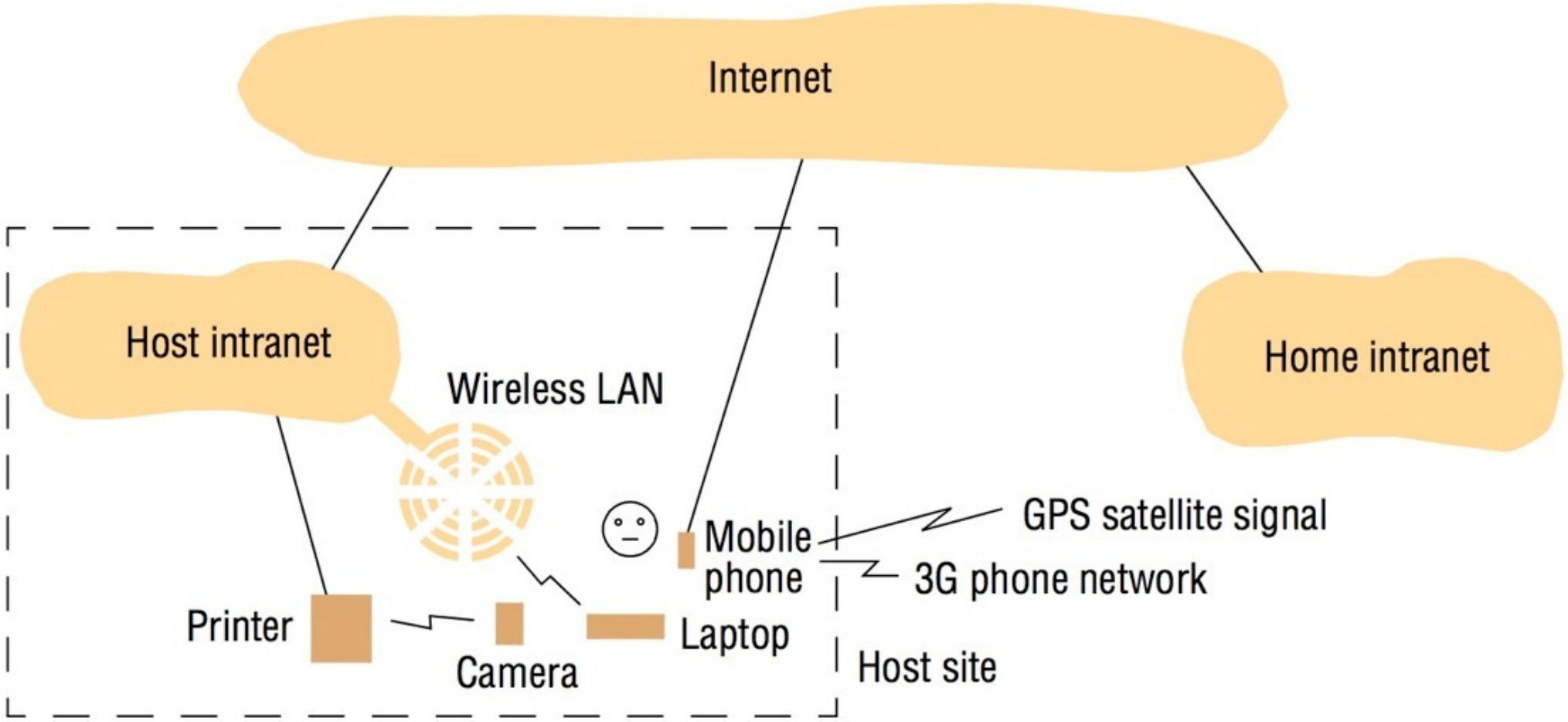
- Networking is a pervasive resource and devices can be connected anywhere, anytime



Trends in distributed systems: Emergence of ubiquitous and mobile computing

- laptop computers, handheld devices, GPS, phones, video cameras, wearable devices, embedded devices, ...
- **Mobile computing**: performing computing task while the user is moving.
  - Leads to *location-aware* or *context-aware* computing
- **Ubiquitous computing**: small, cheap computing devices are lurking everywhere!
  - self-configuration, mesh, autonomic systems, emergent behaviors
  - "Internet of things"
- the two concepts overlap but are different

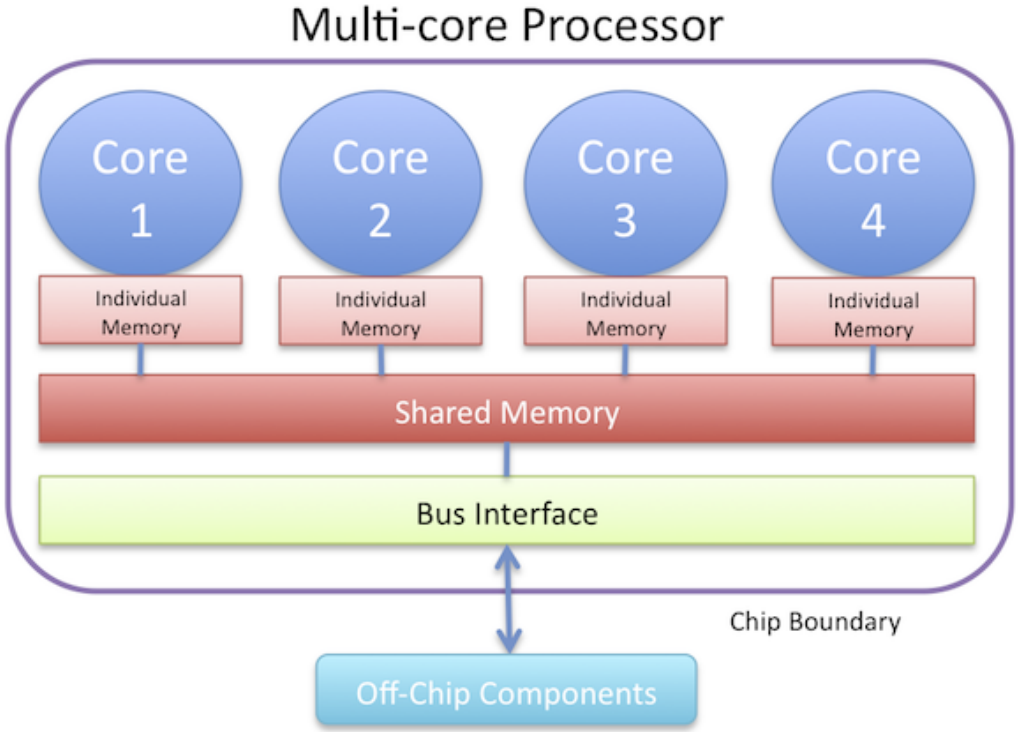
Portable and handheld devices in a distributed system



Multi-core processors as distributed systems

Cores work almost independently

As number of cores increases, this system is easier to understand (and program) as a set of message-passing components



Trends in distributed systems:
Increasing demand for multimedia services

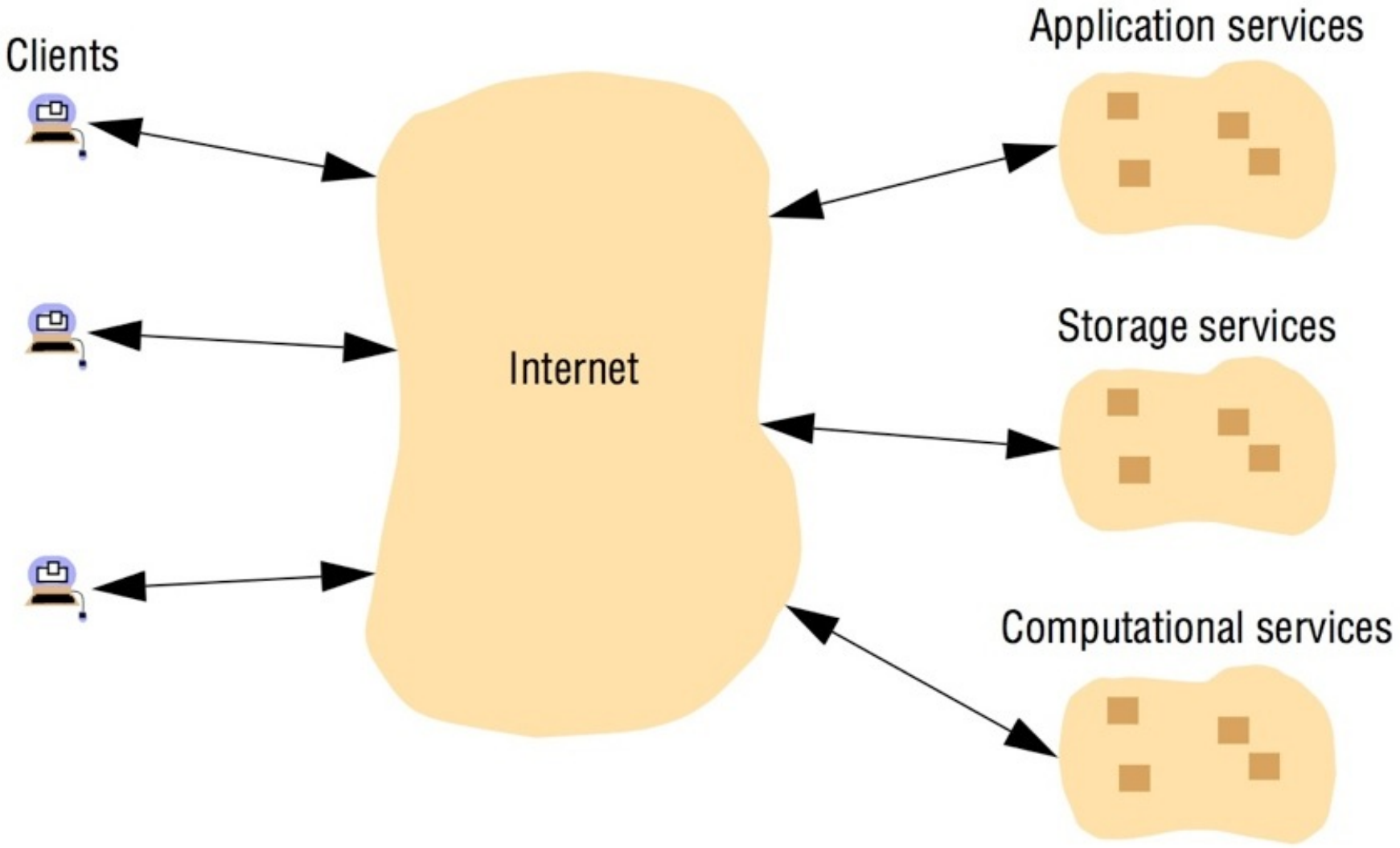
- continuous data streams with stringent real-time requirements
- (video)chat, webcasting, webradio,…
- very important for next consumer applications

Trends in distributed systems:
Distributed systems as a utility (Cloud Computing)

- Companies are moving to using computing resources as a commodity (like water, energy,...)
- Physical resources are concentrated in few big datacentres, with various degrees of virtualization, ranging from specific resources (e.g. storage (Dropobox)) to whole virtual machines in data centers
- Software services can be offered across the Internet
  - Google Apps
- Key: virtualization (VMware and Xen) and containerization (Docker)
- Clouds are implemented by large clusters
  - Big players: Amazon AWS, Google, MS Azur, Rackspace ...

Figure 1.5

Cloud computing



Focus of Distributed Systems is on Resource Sharing

- Resource sharing is standard, nowadays
  - useful for reducing costs
  - but by now part of everyday life and work activity
- **Service**: part of the system that manages a collection of related resources, offering their functionalities to users and applications
- **Server**: process that implements a service. Has direct, reliable and often exclusive access to resources
- **Client**: process that requests services from a server
  - by means of message exchange
  - according to a defined set of rules (protocol)
- *Caveat: many but **not all** distributed systems can be constructed in the form of interacting client & servers*

Fallacies of distributed computing [Peter Deutsch, 1994]

Usual wrong assumptions made by newbie programmers of distributed systems:

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

See http://www.rgoarchitects.com/Files/fallacies.pdf

If a CPU cycle were a heart beat…

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

Challenges and Issues (1)

- **Heterogeneity**: variety and differences in
  - networks, hardware, os, languages, data representation...
  - middleware: CORBA, XML/RPC, SOAP, Java RMI
  - mobile code, virtual machines
- **Openness**: systems can be extended and re-implemented in various ways
  - standard published interfaces, uniform communication mechanisms
  - open published standards: RFC, request for comments
- **Security**: more important in distributed systems
  - Confidentiality (sniffing attacks)
  - Integrity
  - Availability (DOS attacks)
  - Security of mobile code

Challenges and Issues (2)

- **Scalability**: effective with significant increase in resources
  - cost
    - in general, the cost for n users should be O(n)
  - performance
    - time for accessing data set of size n should be O(log n), e.g. using hierarchical data structures
  - not easy to achieve
  - preventing software resources running out
  - avoiding bottlenecks

Figure 1.6
Growth of the Internet (computers and web servers)

Source: http://www.isc.org/services/survey/

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	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

Challenges and Issues (3)

- **Failure handling**: distributed systems fail, often and partially: parts are failing while others keep running
  - detecting: often not easy, or impossible
  - masking: once detected, can be managed at some extent
    - hide: less severe (retransmit)
  - tolerating: ignore, timeout
  - recovery: logs, rollback
  - redundancy - but keeping replicas in sync is challenging
- **Availability**: measure of time the service is available for use
- **Concurrency**: shared resources must be accessed in a coherent way, so that the state remains consistent
  - semaphores, mutual exclusions, distributed commit

Brewer's (CAP) Theorem

- *Consistency*: all nodes have a consistent view of the data
- *Availability:* the service is always accessible: every request receives an answer (success or failure)
- *Partition Tolerance*: no set of failures less than total network failure is allowed to cause the system to respond incorrectly

Ideally, we would a distributed system to enjoy all these properties at the same time!

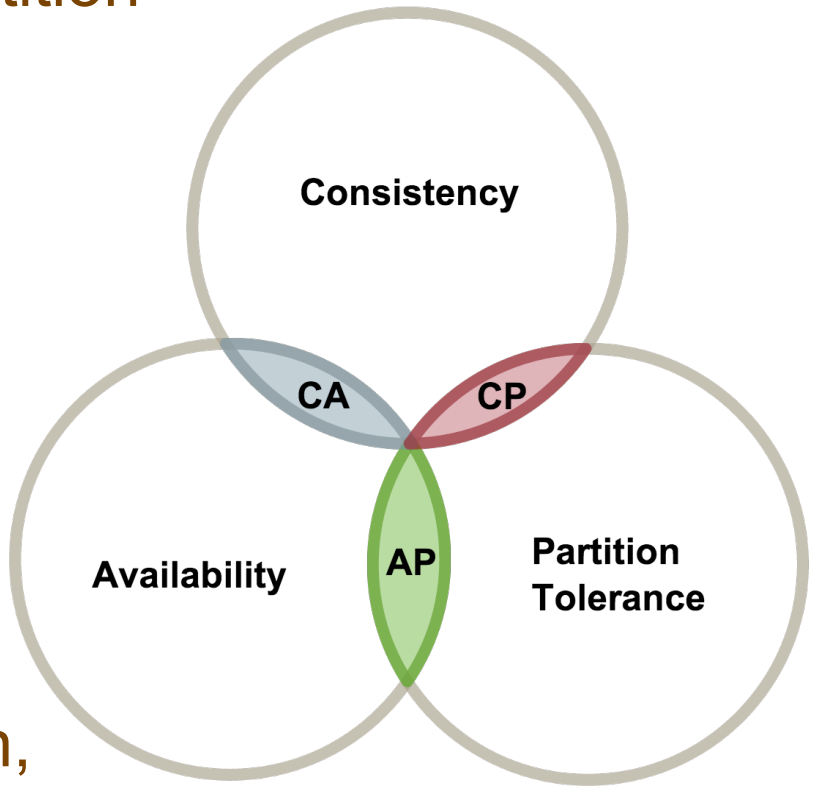
Theorem ("CAP Theorem", Gilbert, Lynch, 2002): A distributed system cannot satisfy all the above requirements at the same time.

See http://ksat.me/a-plain-english-introduction-to-cap-theorem/ for a plain English example of the CAP theorem.

Consequences of Brewer's (CAP) Theorem

We cannot have a system which is consistent, available and partition tolerant. Only two out of three properties can be achieved at the same time. Which one we have to give up depends on the situation.

- **Consistent-Available systems** but not partition tolerant: service works fine as long as the network has no failures
  - Not a real distributed system, then!
- **Consistent-Partition Tolerant systems**: service is suspended in case of partition, in order to avoid inconsistencies.
  - Example: ATMs, most RDBMSs
- **Available-Partition Tolerant systems**: service is not suspended in case of partition, but different hosts can see different informations
  - Example: git, many NoSQL databases



Consequences of Brewer's (CAP) Theorem

- The theorem doesn't rule out other forms of consistency in a distributed system (with availability and partition tolerance)
  - e.g. *eventual consistency:* if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value
  - Very often implemented in distributed systems, if (short) periods of inconsistency can be accepted
- "*More generally, maintaining invariants during partitions might be impossible, thus the need for careful thought about which operations to disallow and how to restore invariants during recovery*" (Brewer, 2012)
  - The Blockchain implements eventual consistency: it has precise rules for restoring invariants during recovery

Challenges and Issues (4)

- **Transparency**: concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than a collection of independent components
- The ISO Reference Model of Open Distributed Processing has defined these forms of transparencies
  - ***Access transparency***: enables local and remote resources to be accessed using identical operations.
  - ***Location transparency***: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
  - ***Concurrency transparency***: enables several processes to operate concurrently using shared resources without interference between them.

Transparencies (cont'd)

- ***Replication transparency***: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application 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.

Quality of Service (QoS)

- Nonfunctional properties of systems that affect the quality of the service provided
  - Reliability, Security, Adaptability
  - Performance: the ability to meet timeliness guarantees
- Time critical data: streams to be processed at a fixed rate
  - Example: audio/video streaming, audio/video conferencing
- Ability of meeting deadlines depends upon availability of adequate computing and network resources
- Each critical resource must be reserved by the applications requiring QoS
- Resource managers and protocols for providing guarantees