Distributed Systems Lecture 1

Davide Frey, WIDE Team, Inria Rennes

davide.frey@inria.fr

https://people.irisa.fr/Davide.Frey

Classical Distributed Algorithms

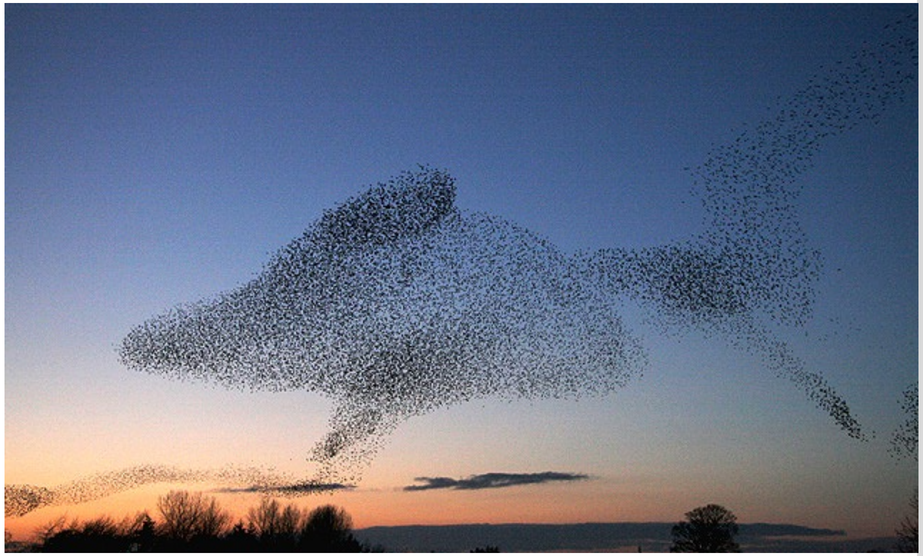
- Fully connected graph
  - every process can interact with every other process
- Communication Models
  - Message Passing
  - Shared Memory
- Timing Assumptions
  - Synchronous
  - Asynchronous
- Fault Models
  - No Faults
  - Crash Failures
  - Byzantine Failures (if time allows)

Some Terminology

Parallel Computing

Ben Freeman / https://www.flickr.com/photos/brf/3102459936/

Distributed Computing

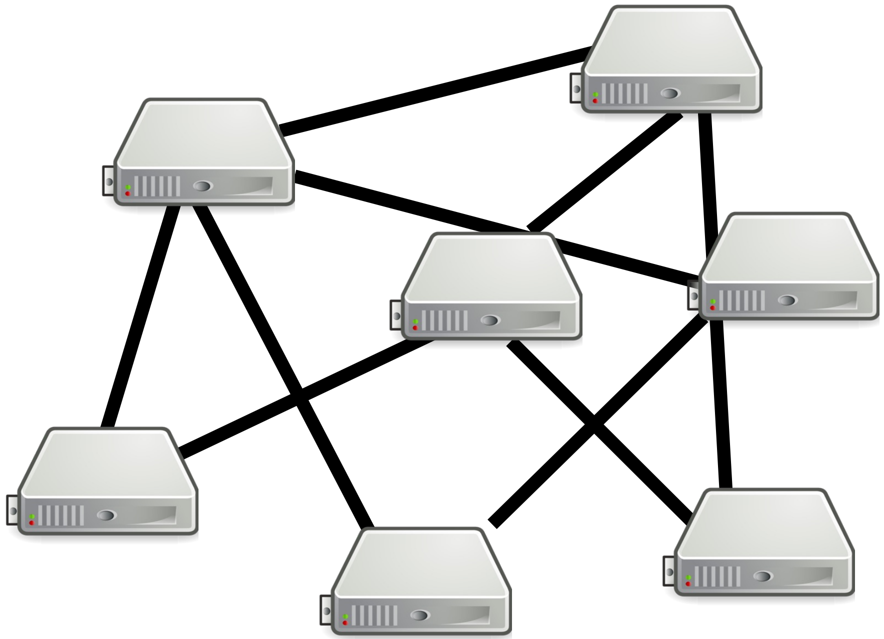


Walter Baxter / *Starling shapes in the evening sky* /

Some Terminology

Parallel Computing

Distributed Computing



More Terminology

- Distributed System

> “A collection of independent computers that appears to its users as a single coherent system. ” Tanembaum & Van Steen [DS Book]

> “A distributed [computer] system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. ” Leslie Lamport [1987, e-mail]

Examples of Distributed Systems

- **Web search**
  - Google : over 130 trillion indexed web pages (2016), over 105 billion queries per month (2020)
    - Major distributed systems challenges
- **Massively multiplayer online games**
  - Fortnite: 250M players (2019), up to 8.3M concurrent players
  - Need for very low latencies to support game
- **Financial trading**
  - Support for financial trading systems
  - Dissemination and processing of events
- **Mobile Apps**

From Systems to Algorithms

- Distributed Algorithms refer to the abstract methods we use to build distributed systems

Sequential Algorithm	:	Sequential Program
	=	
Distributed Algorithm	:	Distributed System

From Systems to Algorithms

- Distributed Algorithms refer to the abstract methods we use to build distributed systems

Sequential Algorithm	:	Sequential Program
	=	
Distributed Algorithm	:	Distributed System

(Historical) Examples of Distributed Problems

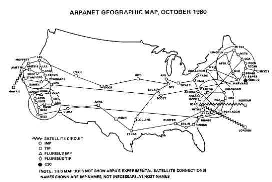
- Distributed *Computer* Systems are largely concerned with:
  - Data processing/management/presentation (“**computing**” side)
  - Communication/ coordination (“**distributed** side”)

- Those concerns existed well before computers were invented
  - **Ancient empires** needed efficient communication systems:
    - cf. the Postal Service of the Persian Empire (6th century BC), cf. the Roman roads (many still visible today), etc.
    - max message speed: ~ 300 km/day in the Persian system
    - assuming a good infrastructure (roads, horses, staging posts)
  - **Delays** impose distributed organisations
    - Persian and Roman empires extended over 1000s of miles
  - **Trust / secrecy / reliability** issues
    - Am I sure Governor X is doing what he says he is?

- This all has not really changed! Things have only speeded up!

(Historical) Examples of Distributed Algorithms

- Computers are far more **recent** than empires
  - The first "modern" computers appeared just after WWII
  - They were slow, bulky, and incredibly expensive
    - The ENIAC (1945), used by the US army: 30 tons, 170 m^2 footprint, 180 kilowatts, 18,000 vacuum tubes, and 5,000 additions/second (5KHz),
    - Price: $500,000 (in US$ of the time, would be roughly $5,000,000 today)
- And distributed computing is **even more** recent
  - For a long time, only very few computers around anyway
  - No practical technology to connect them
  - This all changed in the 80's:
    - The rise of the **micro-computers** (PC, Mac, etc.)
    - The launch of the **"Internet"** (1982, TCP/IP), after 10 years of development
  - (Almost) everybody could have a computer
  - And there was a way to connect them!



Back to Systems

- During the 80's computer networks mainly remained an **academic** affair
  - Competing networks and technologies, not always compatible
    - ARPANET/Internet was one of them, but not always the biggest.
    - Who remembers BITNET? Was quite big at the time.

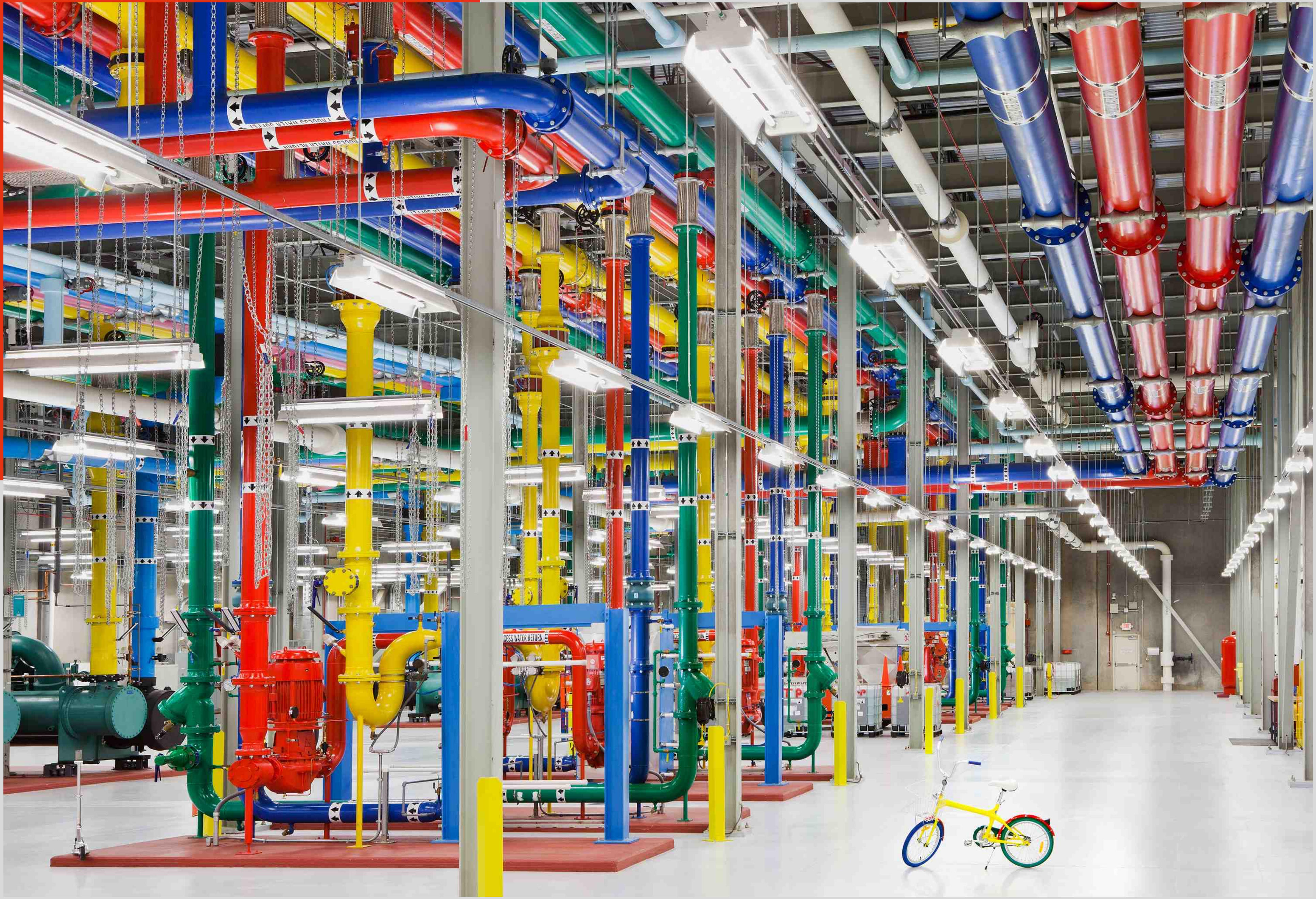
  - Not particularly user friendly
    - User programs were text based (news (Usenet), e-mail)
    - You had to know on which "network" a recipient was to sent her a e-mail.
- And then in 1990 came the **Web**
  - At **CERN** (European Organization for Nuclear Research)
  - Internet + hypertext (hyperlink) - allowed text-based browsing!
  - Triggered developments that made the modern web
    - HTML, HTTP, Graphical browsers, search engines, XML, RSS, blogs,….
  - Assured the domination of the Internet over other networks

Recent Examples

- **Client-Server** Applications
  - Distributed Databases (end of the 80’s)
  - The Web (at least until WebRTC)
- **Peer-to-Peer**
  - File Sharing, Video Streaming, countless applications
- **Grid Computing**
  - Computational grid akin to electrical grid
- **Cloud Computing**
  - Build on many previous technologies
  - Key role of **virtualisation**, **web**, **networking**
- **Fog/Edge Computing**
  - A more peer-to-peer cloud

http://www.google.com/about/datacenters/gallery/#/

http://www.google.com/about/datacenters/gallery/#/



http://www.google.com/about/datacenters/gallery/#/

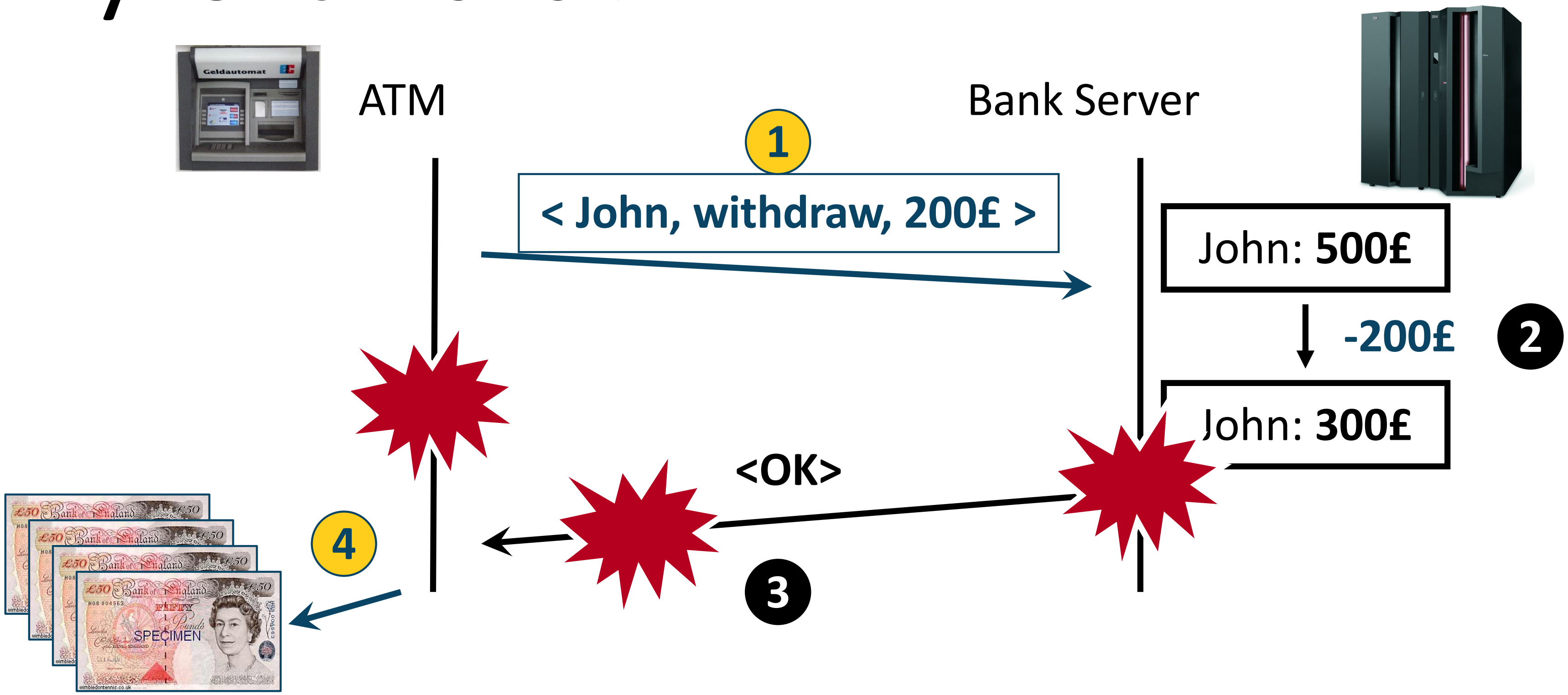
Why Study Distribution

- **Distribution** at the core of almost all recent ICT **revolutions**
  - mobile telephony (Nokia, iPhone)
  - search (Google)
  - social computing (Facebook, Twitter, Instagram)
  - cloud computing (AWS, Azure, but also OVH, Scaleway)
  - mobile apps (TikTok, WhatsApp)
  - big data
  - Machine learning and artificial intelligence
- But **developing** good distributed systems is terribly hard
  - DS are software intensive
  - Developing good distributed software is tough (even for Google)

Why is it Hard?

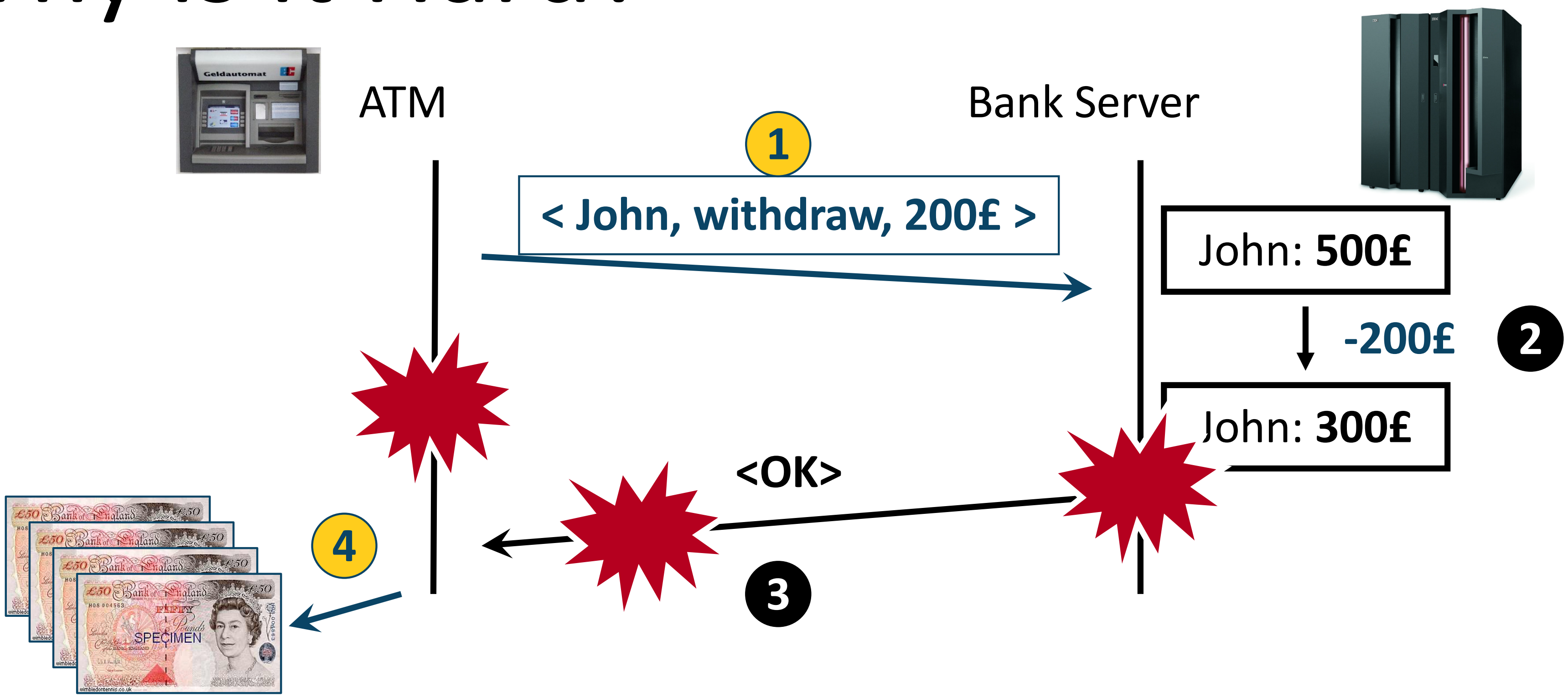
- A bank asks you to program their new **ATM software**
  - Central bank computer (server) stores account information
  - Remote ATMs authenticate customers and deliver money
- **A first version** of the program
  - ATM: (ignoring authentication and security issues)
    1. Ask customer how much money s/he wants
    2. Send message with **<customer ID, withdraw, amount>** to bank server
    3. Wait for bank server answer: **<OK>** or **<refused>**
    4. If **<OK>** give money to customer, else display error message
  - Central Server:
    1. Wait for messages from ATM: **<customer ID, withdraw, amount>**
    2. If enough money withdraw money, send **<OK>**, else send **<refused>**

Why is it Hard?



- But ...
  - What if the bank server crashes just after 2 and before 3?
  - What if the <OK> message gets lost? Takes days to arrive?
  - What is the ATM crashs after 1, but before 4?

Why is it Hard?



- This problem is known as the **distributed atomic commit problem**
  - Everybody act or nobody does (atomicity), even if problems
- Requires **fault-tolerance**
  - System keeps working even when subcomponents fail

Why is it hard? Other Issues

- Other fault-tolerance/availability concerns
  - Replication, caching & consistency issues
  - Reliable communication (multi-cast, message ordering, etc.)
- But fault-tolerance/availability not the only concerns in DS:
  - **Heterogeneity**: How to “glue” different applications on different OS, written in different languages, from different vendors?
  - **Evolvability**: How to change parts of a DS or add new parts without stopping the whole system?
  - **Scalability**: Can a DS grow smoothly without disruption? Are they inherent size limitation in the techniques involved?
  - **Separation of Concerns**: Can development effort be split easily between teams?
  - **Security**: Risks? Vulnerabilities? Which level of integrity, confidentiality, robustness does the system present?

From Systems to Algorithms

- Distributed Algorithms refer to the abstract methods we use to build distributed systems

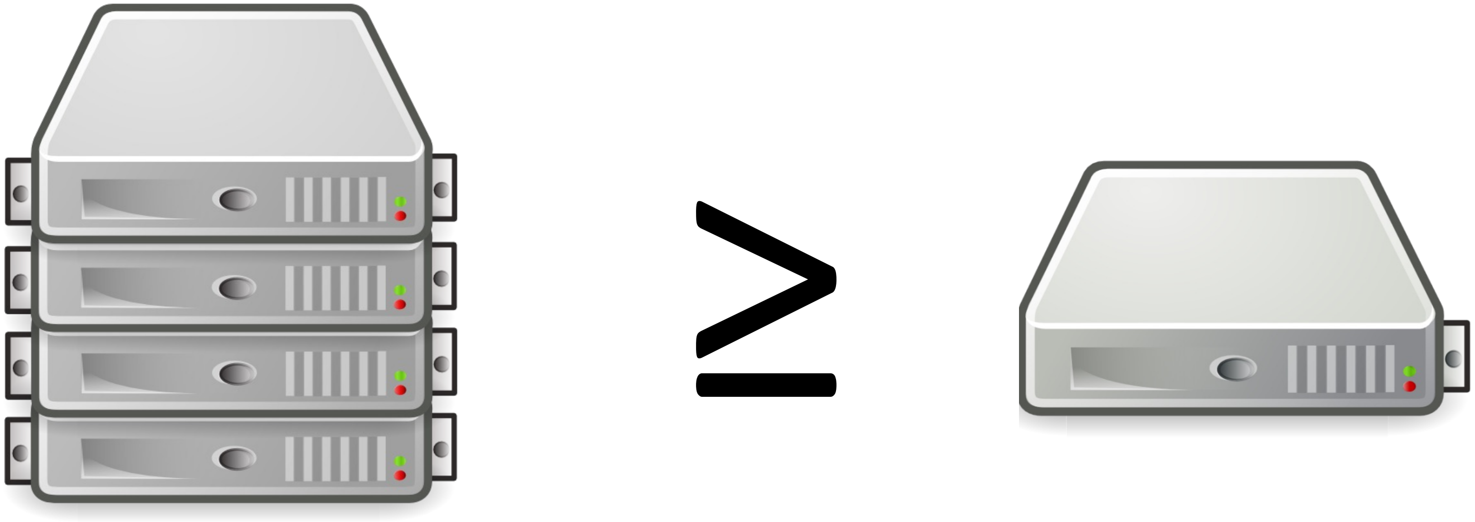
Sequential Algorithm	:	Sequential Program
	=	
Distributed Algorithm	:	Distributed System

In The Sequential World

Alan Turing (1912-1954)

Kurt Gödel (1906-1978)

In The Parallel World



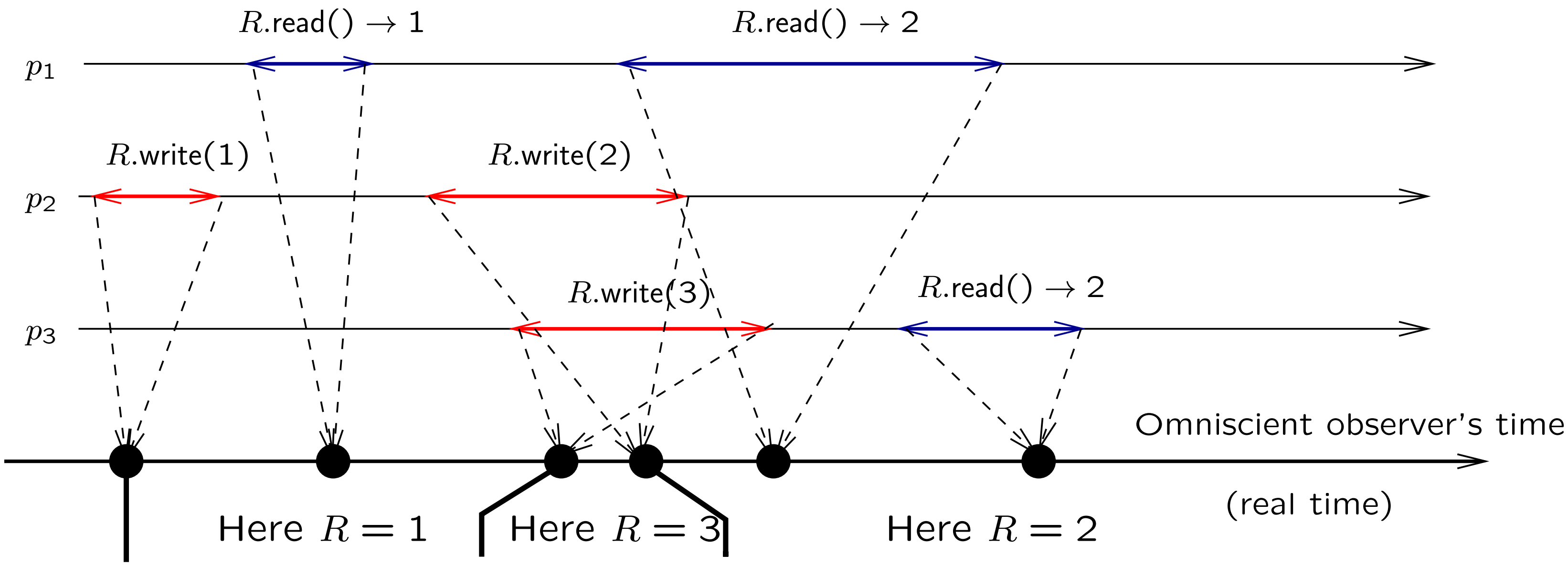
- The objective is speed and performance.
- Close to sequential world in terms of computability
  - Modulo “some” synchronization issues
    (threads, locks, barriers, ...)

In the Distributed World

- New enemies
  - Faults (machines, links)
  - (Unpredictable) Delays (messages, machines)

Example: Read-Write Register

- Peer-to-Peer Model
  - All processes $\{p_1,p_2,..,p_n\}$ = P are equal.
- Distributed RW register
  - Host a copy of a memory register. Two operations: read, write
  - Should behave atomically ("one copy semantics")



Read-Write Register (cont.)

- Fault model
  - Any number of processes may crash (up to |P|-1)
  - Message do arrive, but may take arbitrary long (asynchrony)
- Question
  - Can we implement a shared atomic RW register in this model ?

Distributed Algorithms

- Distributed Algorithms look at
  - fundamental problems of **distributed coordination**
  - for instance: agreement, mutual exclusion, leader election...
  - in an **abstract way** (abstract model of reality)
- Sometimes assuming some **adverse conditions**
  - participants may behave somewhat erratically
  - messages may get lost
- Goal of the study of distributed algorithms
  - find out **whether something is possible** under which conditions
  - for solvable problems, **prove** that a particular solution works
  - **compare** correct solutions to the same problems

Example of a distributed algorithm: Boolean OR

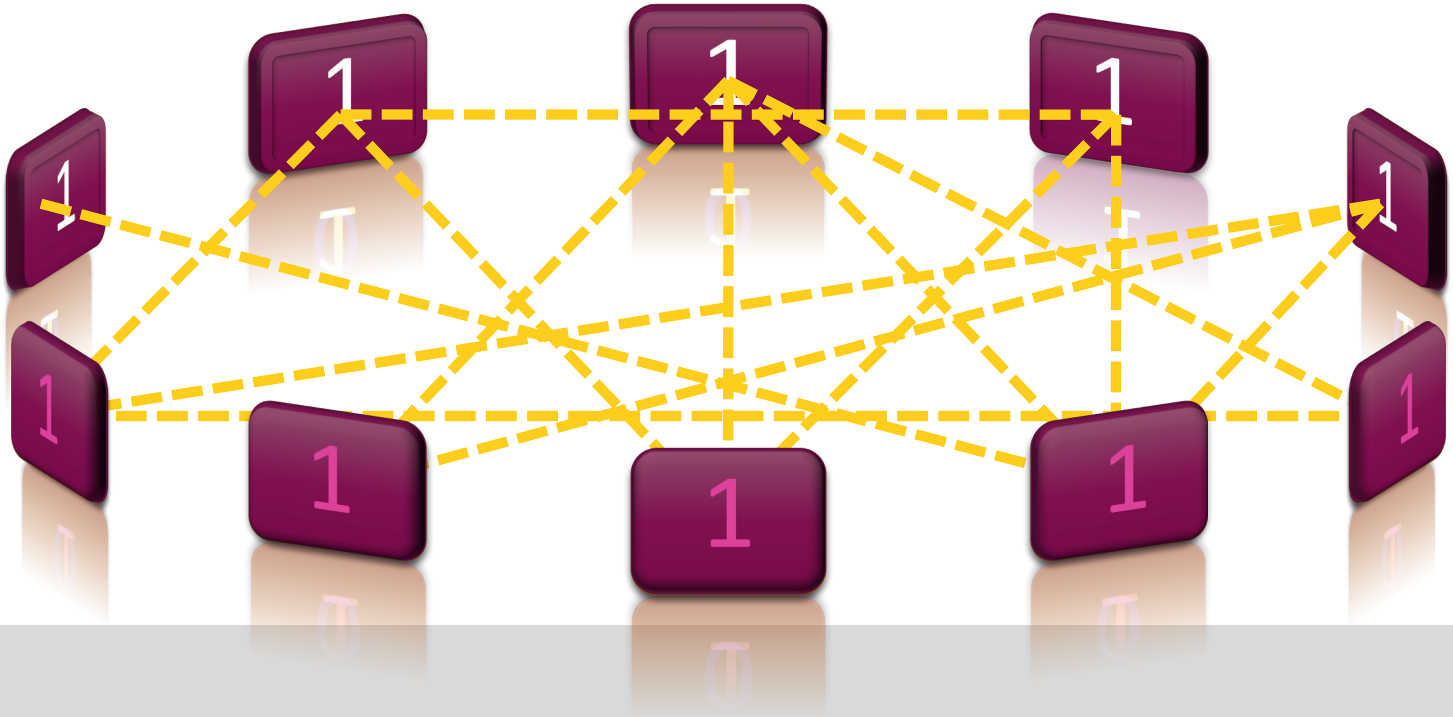
- Each actor has an initial value
  - *True* represented by 1
  - *False* represented by 0

A	B	A OR B
False	False	False
False	True	True
True	False	True
True	True	True

- **If** at least one actor is *True* at the beginning **then** the global result must be *True*

Example:

- Interaction only 2 by 2, random actors
- After interaction, both actors will share the same value, result of the OR between their previous values



Correctness proof

- If all actors have value 0 at the begining
  - No one should become 1 after an interaction
  - The protocol will return 0 (*i.e., false*)
- If at least one actor has value 1
  - Once an actor has value 1, it cannot go back to 0
  - If actors are randomly and fairly chosen, any actor with 0-value has a non-null probability to interact with an actor that owns a 1-value
  - The number of actors with 1-value can only increase
  - This number converges to *n* with probability equal to 1
  - The protocol will the return 1 (*i.e., true*)

Group Solving Session

The Cursed Monastery

- A visitor comes to a remote monastery and announces:
  - " *Some of the monks have been cursed by the local wizard and marked by a point on their forehead. They must all leave the monastery, or the whole community will perish.* "
- This monastery obeys a very strict rule:
  - There are no mirrors in the monastery.
  - Monks do not communicate in any way.
  - They only meet once a day for dinner.
- The visitor makes his announcement at dinner.
- How many days does it take for all the cursed monks to leave the monastery and why?
  - Hint: the monks have studied distributed algorithms

The Royal Wedding

- A king would like to marry his son to the princess of a neighbouring kingdom
- By tradition, if the alliance is agreed, the wedding will take place in a remote monastery, on the border between the two kingdoms
- It is all right if the parties do not arrive at the same time at the monastery
- Messengers travel by horses, and may get lost to thugs
  - however they have a non-zero chance of getting through
- Design an algorithm that allows the wedding to take place if both parties agree (if not, nothing should happen)

The 2 Generals

- 2 allied generals have surrounded their common enemy
- Their camps are 1 day apart by horse from each other
- They want to agree on when to attack
- Each can send the other one only one message per day
- Messengers might get attacked by thugs and get lost
- Design an algorithm for the 2 generals to reach an agreement and attack simultaneously

What have we learnt?

- Whether you know **how long your messages will take** makes a huge difference
  - No bound on communication delays: **asynchronous** systems
  - Bounded communication delays: **synchronous** systems
- With bounded delays + global clock (monastery)
  - **Not doing something** can mean a lot
- Some problems have **no solution**
  - Coordination with lossy channels impossible (the generals)
- If **communication** channels are **faulty**
  - possible to **make** them **perfect** (the royal wedding)
  - but a **price** to pay: communication delays can get **arbitrary long**
  - this is how **Ethernet** & **TCP/IP** work (with some timeouts)
  - does **not** work for **real time** systems