Fault-Tolerant Distributed Systems

Rui Carlos Oliveira
rco@di.uminho.pt

Informatics Department
Minho University

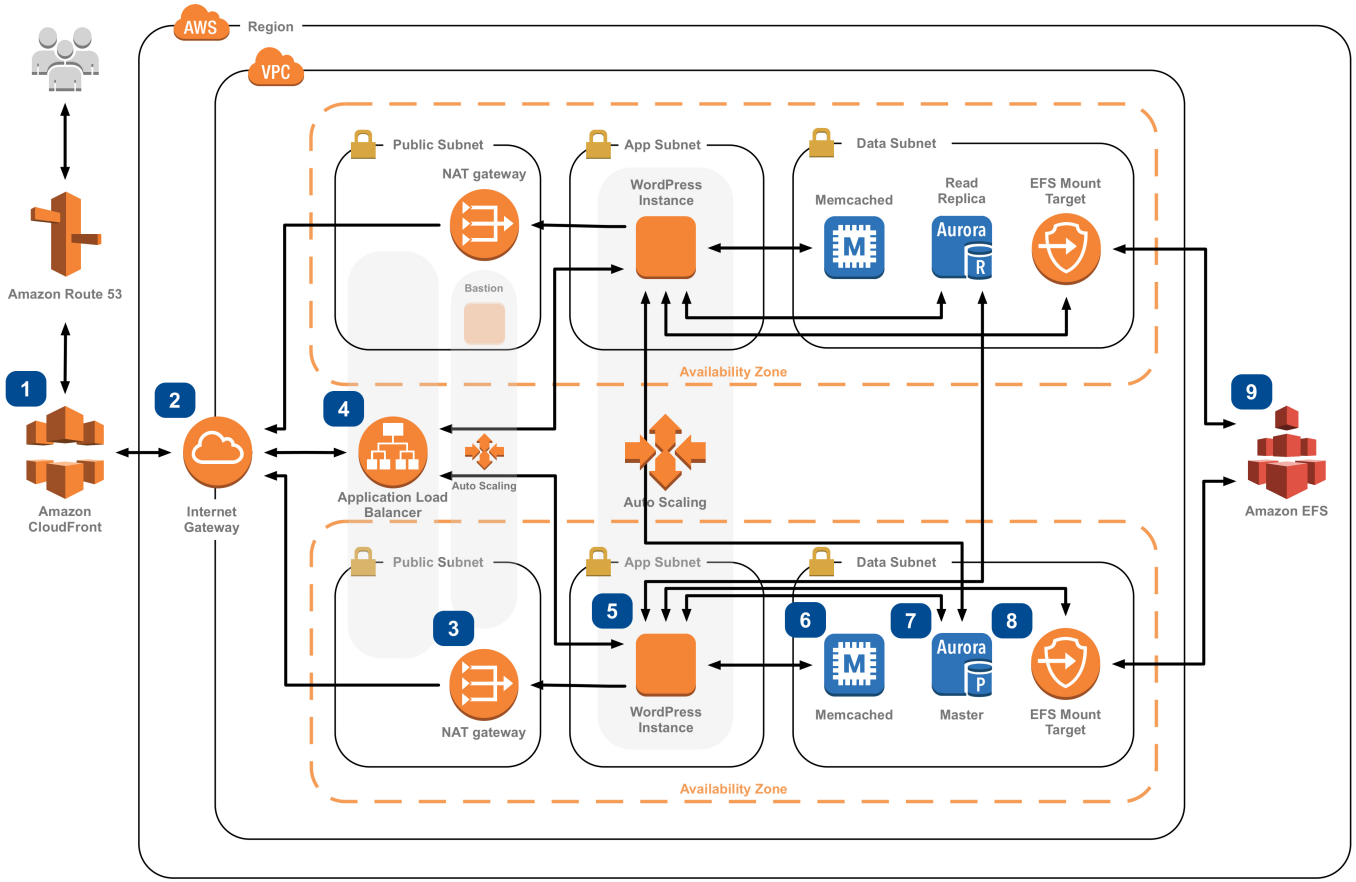
2022/23

Syllabus

- **Introduction to fault-tolerant distributed systems**
- Models of distributed systems and related faults
- Data replication
- Distributed consensus
- State machine replication
- Database replication

Introduction to fault-tolerant distributed systems

- Software systems are ever more complex, bigger, and full of intricate dependencies



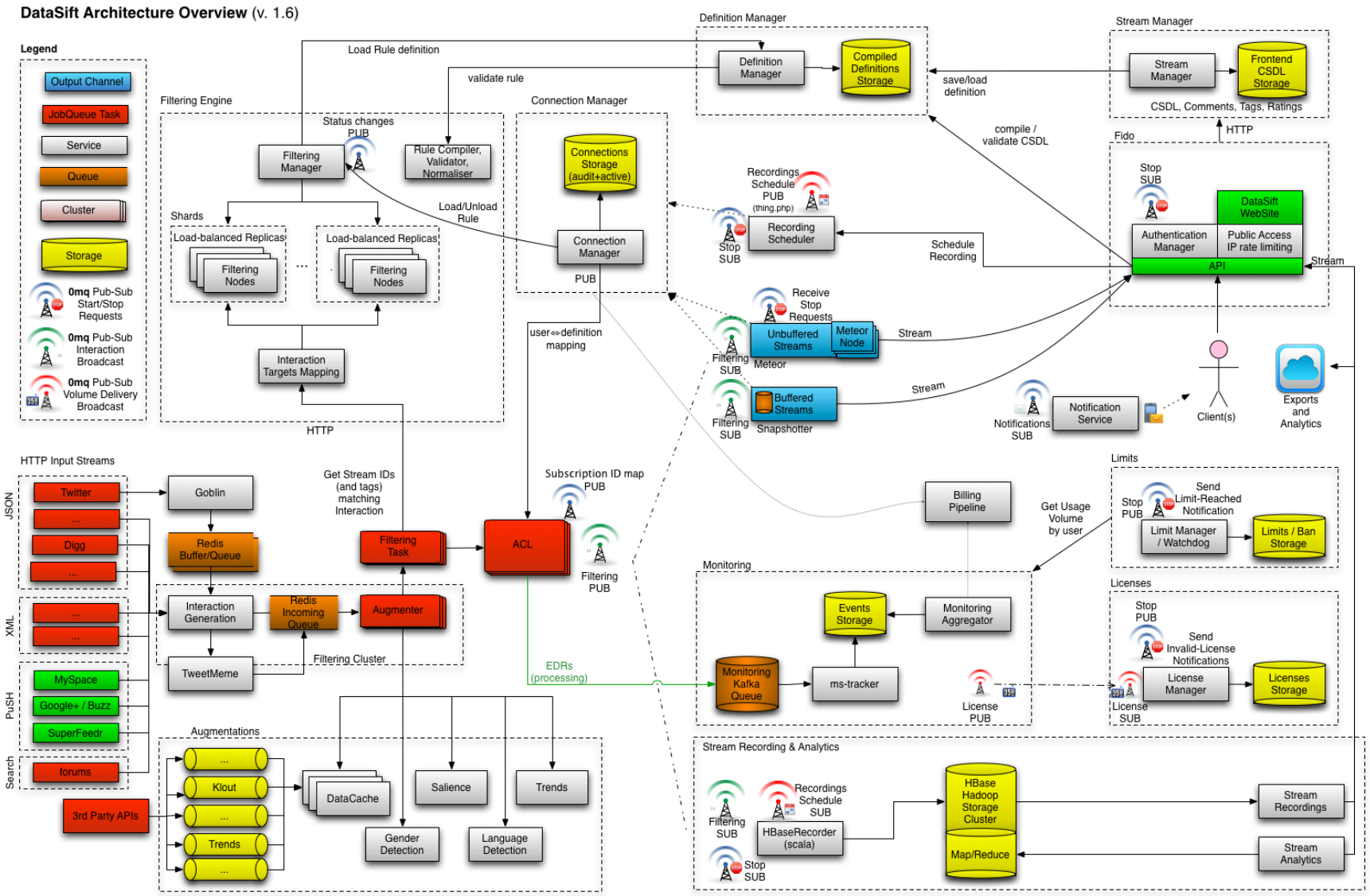
1. Static and dynamic content is delivered by **Amazon CloudFront**.
2. An **Internet gateway** allows communication between instances in your VPC and the Internet.
3. **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like **Amazon ElastiCache** (**Memcached**) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using **Mount Targets** in each AZ in your VPC.
9. Use **Amazon EFS**, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.

amazon web services

AWS Reference Architectures

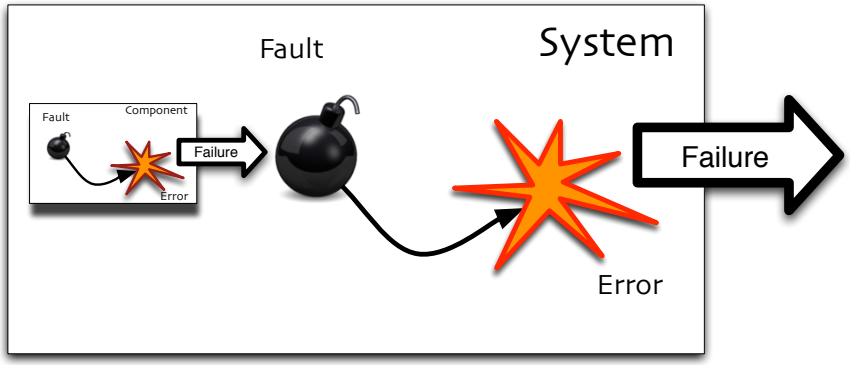
Introduction to fault-tolerant distributed systems

- Software systems are ever more complex, bigger, and full of intricate dependencies



Introduction to fault-tolerant distributed systems

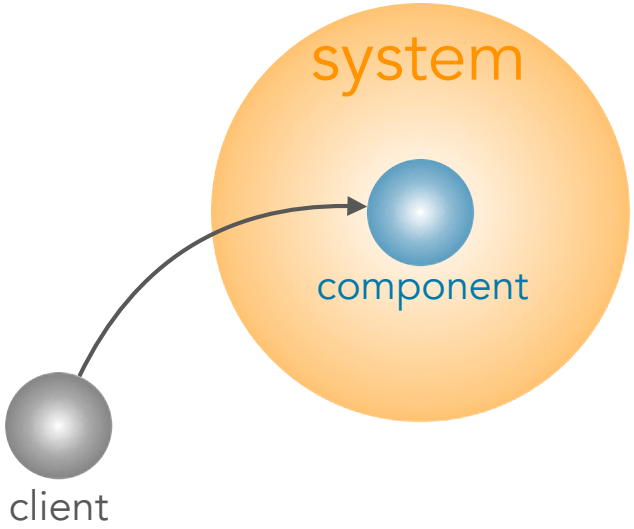
- Goal of the course
  - Learn how to make a software system fault-tolerant
- Impact of faults
  - Performance
  - Unavailability
  - Misbehaviour



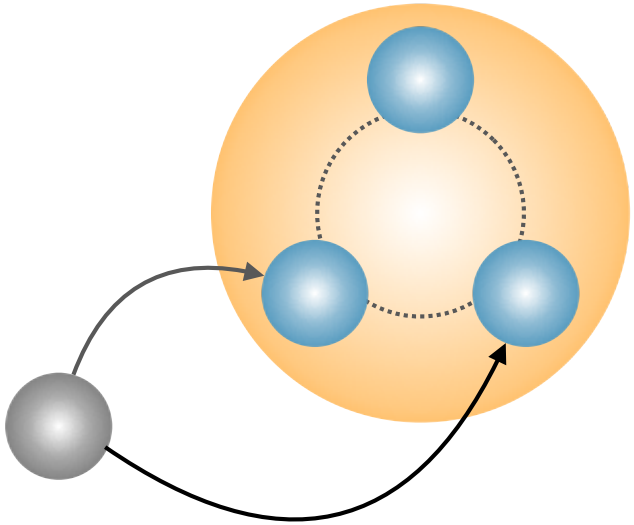
- The challenge
  - Make *fault-tolerant X* service as available as possible (availability)
    - 99% to 99.999% of uptime (3.65 days to 5.2 minutes of downtime)
  - Any behaviour of *fault-tolerant X*, needs to be a behaviour of X (safety)

Introduction to fault-tolerant distributed systems

- Fault tolerance requires replication
  - As any system component is prone to fail
  - Any vital component needs to be replicated
    - Remove any *Single Point Of Failure (SPOF)*

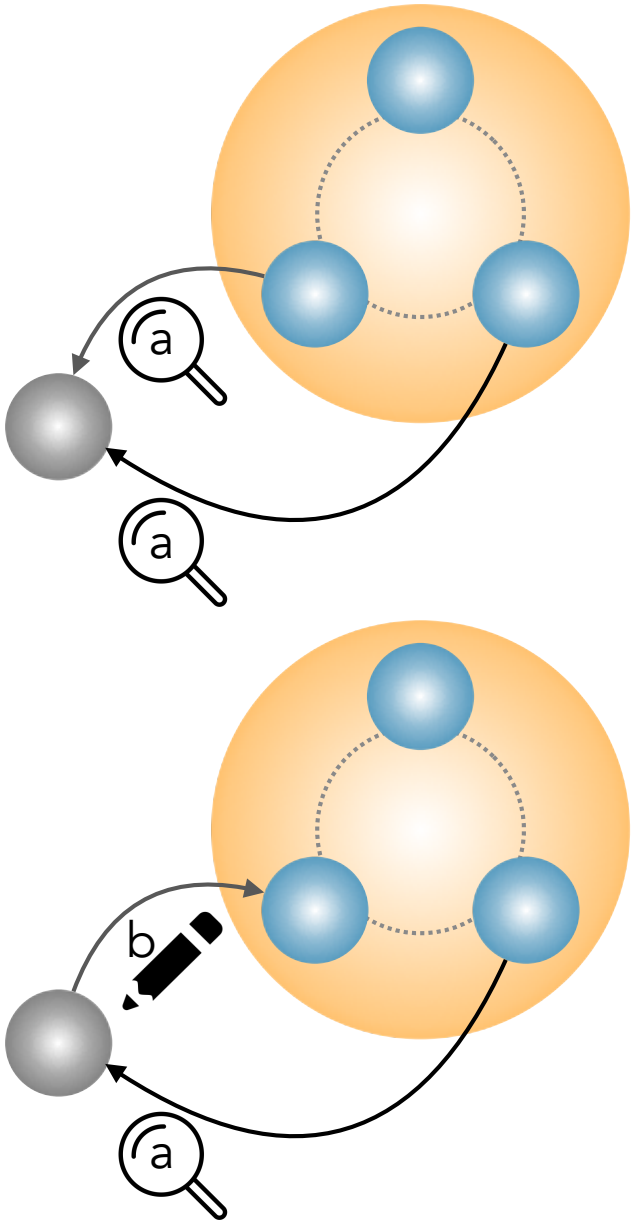


- Distributed systems suit replication well
  - Modularity
  - Incremental redundancy
  - Graceful degradation
  - Heterogeneity
  - Encapsulation



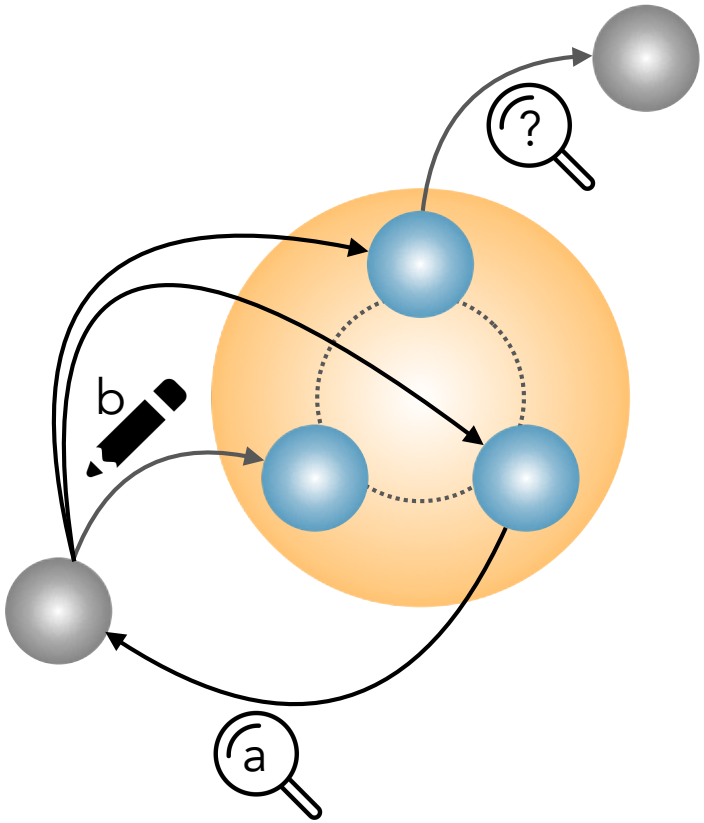
Introduction to fault-tolerant distributed systems

- Any replication approach depends on
  - The component's state mutability



Introduction to fault-tolerant distributed systems

- Any replication approach depends on
  - The component's state mutability
  - The concurrency model



Introduction to fault-tolerant distributed systems

- Any replication approach depends on
  - The component's state mutability
  - The concurrency model
  - The type and number of faults (fault model)
  - The workload bias (usually between reads and writes)

Introduction to fault-tolerant distributed systems

Reading material

- A. Avizienis, J-C Laprie, B. Randell, and C. Landwehr
  "Basic Concepts and Taxonomy of Dependable and Secure Computing",
  IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 1, NO. 1, JANUARY-MARCH 2004

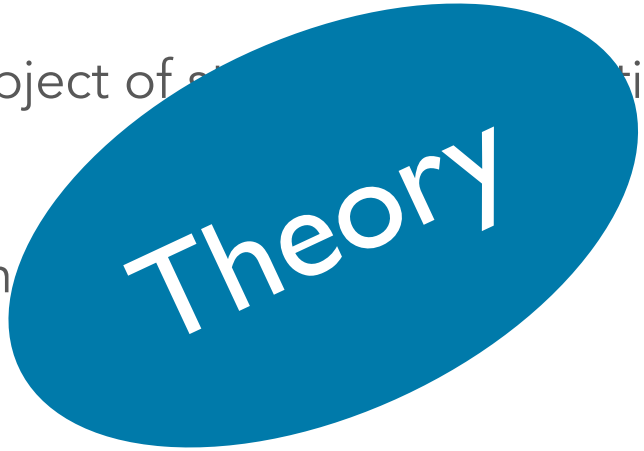
Syllabus

- Introduction to fault-tolerant distributed systems
- **Models of distributed systems and related faults**
- Data replication
- Distributed consensus
- State machine replication
- Database replication

Models of distributed systems and related faults

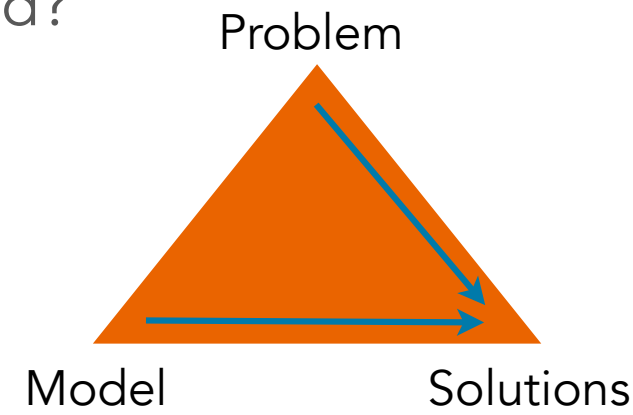
- Experimental observation
  - Build things and observe how they behave in va
  - Experience enables us to build things for that have been studied

- Modelling and analysis
  - Formulate a model by simplifying the object of s ting a set of rules to define its behaviour
  - Analyze the model and infer consequen



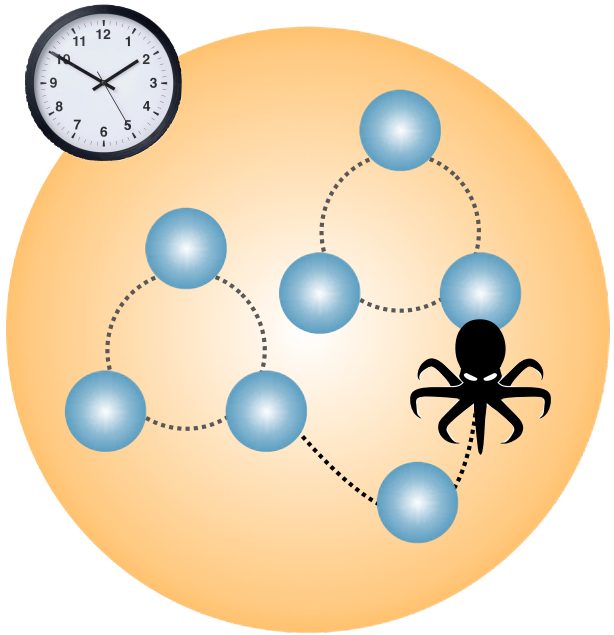
Models of distributed systems and related faults

- Modelling a system consists of identifying and precisely characterising its entities and the behavioural rules that are relevant for the problems in hand
- A model needs to be tractable in that it does not contain uninteresting details that would impair the desired analysis and, at the same time, it needs to be accurate so that it captures all attributes affecting the phenomena of interest
- We seek answers to two fundamental questions:
  - Feasibility: What kind of problems can be solved?
  - Cost: What is the relative cost of solutions?



Models of distributed systems and related faults

- A minimal model for a distributed system consists of a finite set of autonomous sequential processes connected through a finite set of point-to-point communication channels
- We picture the existence of an adversary with a limited power to impair the correct behaviour of processes or channels
- For the problems we will be considering we model the system as:
  - A finite set of sequential processes
  - A finite set of communication point-to-point channels
  - An adversary
  - Rules governing the behaviour of the above



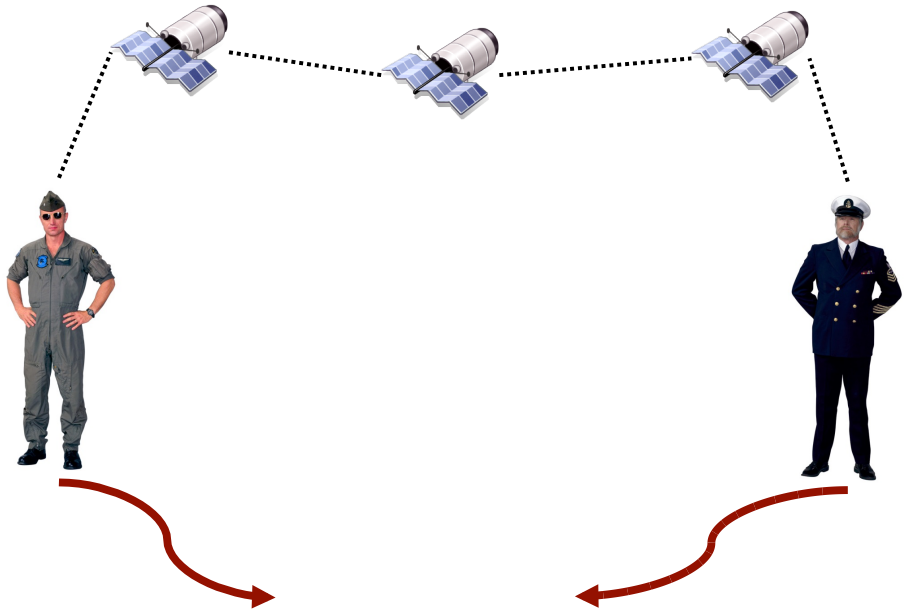
Models - What kind of problems can be solved?

- System model:
  - Two processes, A and B, communicate by sending and receiving messages
  - Neither process can fail. However, the channel can experience transient failures, resulting in the loss of a subset of the messages that have been sent

- A **Coordination problem**:
  - Devise a protocol where either of two actions α and β are possible, but (i) both processes take the same action and (ii) neither takes both actions

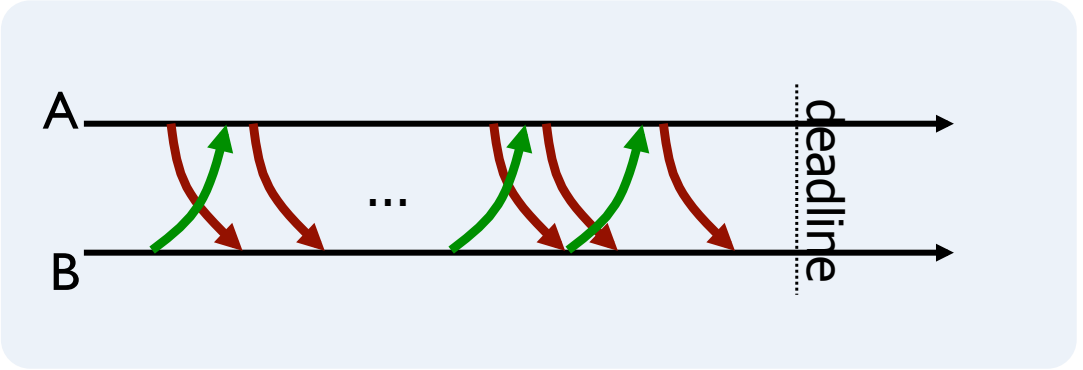
Problem: Agree on a simultaneous attack

Both attack: Win!
Only one attacks: Dead
No one attacks: Safe but…



Models - What kind of problems can be solved?

- Any solution will consist in a message exchange between A and B
- Without channel failures the solution would be straightforward
- Interesting case: Messages received is a subset of messages sent
- Consider an hypothetical solution:



  - Does the last message matter?
  - And the second last?
  - What if the "solution" was the one with the minimum message exchange?

Models - Synchrony and Asynchrony

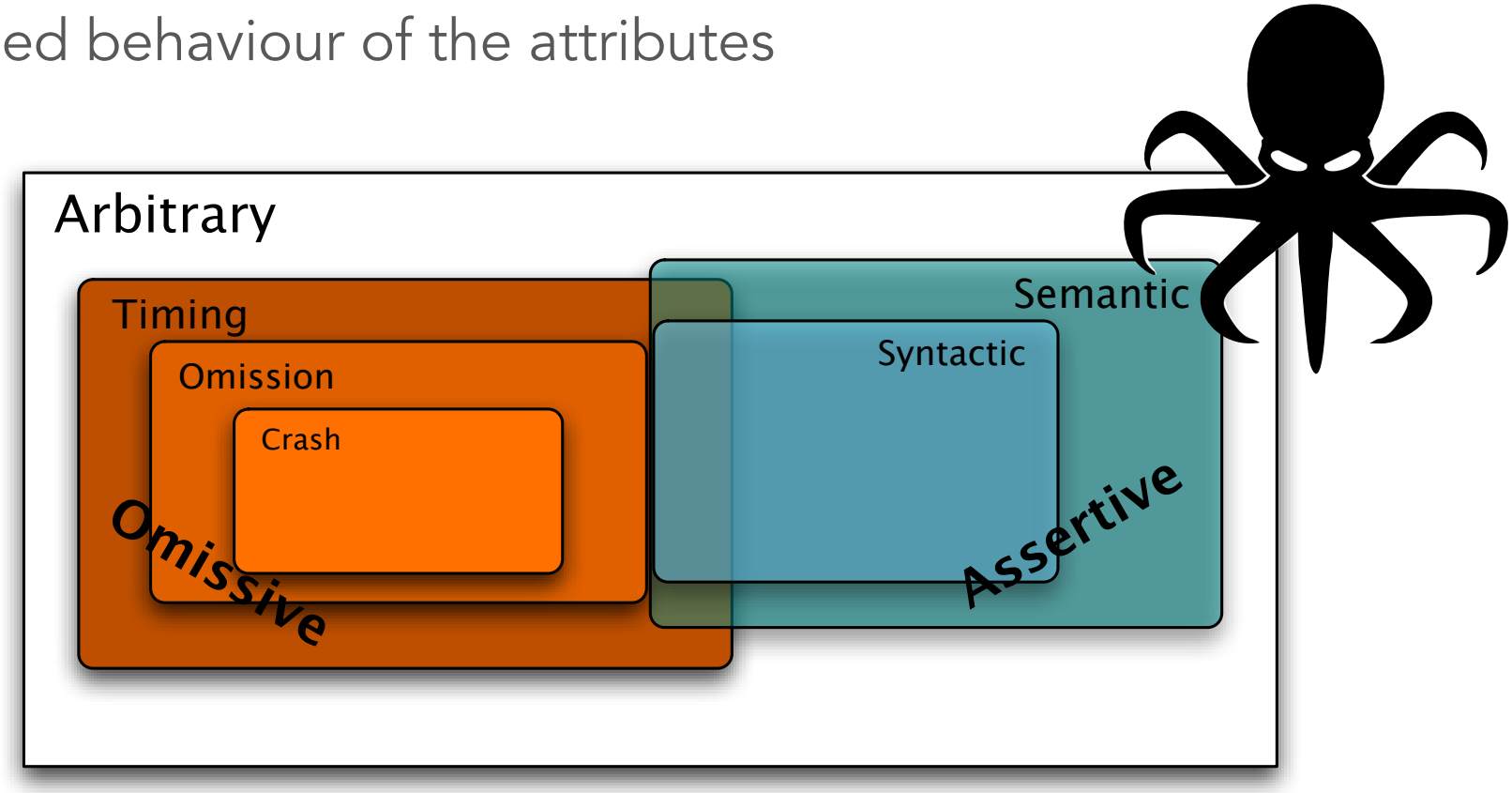
- Of major importance when modelling a distributed system are the assumptions we make regarding the synchrony or asynchrony of processes and communication channels
- Modelling an attribute as asynchronous means that we make no assumptions, absolute or relative, with respect to the time it takes to perform an action. An asynchronous model leads to universal solutions with respect time
- An attribute is synchronous if we assume it take at most t time units to perform an action
- Models can be semi-synchronous in that the attributes are synchronous
  - But *t* is unknown
  - But only eventually

Models - What is the relative cost of solutions?

- System model A:
  - A set of processes P_1,..., P_n. Each process P_i has a unique integer identifier id_i
  - There is no adversary
  - All processes start executing at the same time but **no synchrony assumptions**
- An Election problem:
  - Devise a protocol to elect a unique leader eventually known by all processes
- System model S:
  - Processes take at most t_p per step and messages take at most t_m to be delivered
- Homework:
  - Devise a solution in S that is not a solution in A. How do they compare?

Models of distributed systems and related faults

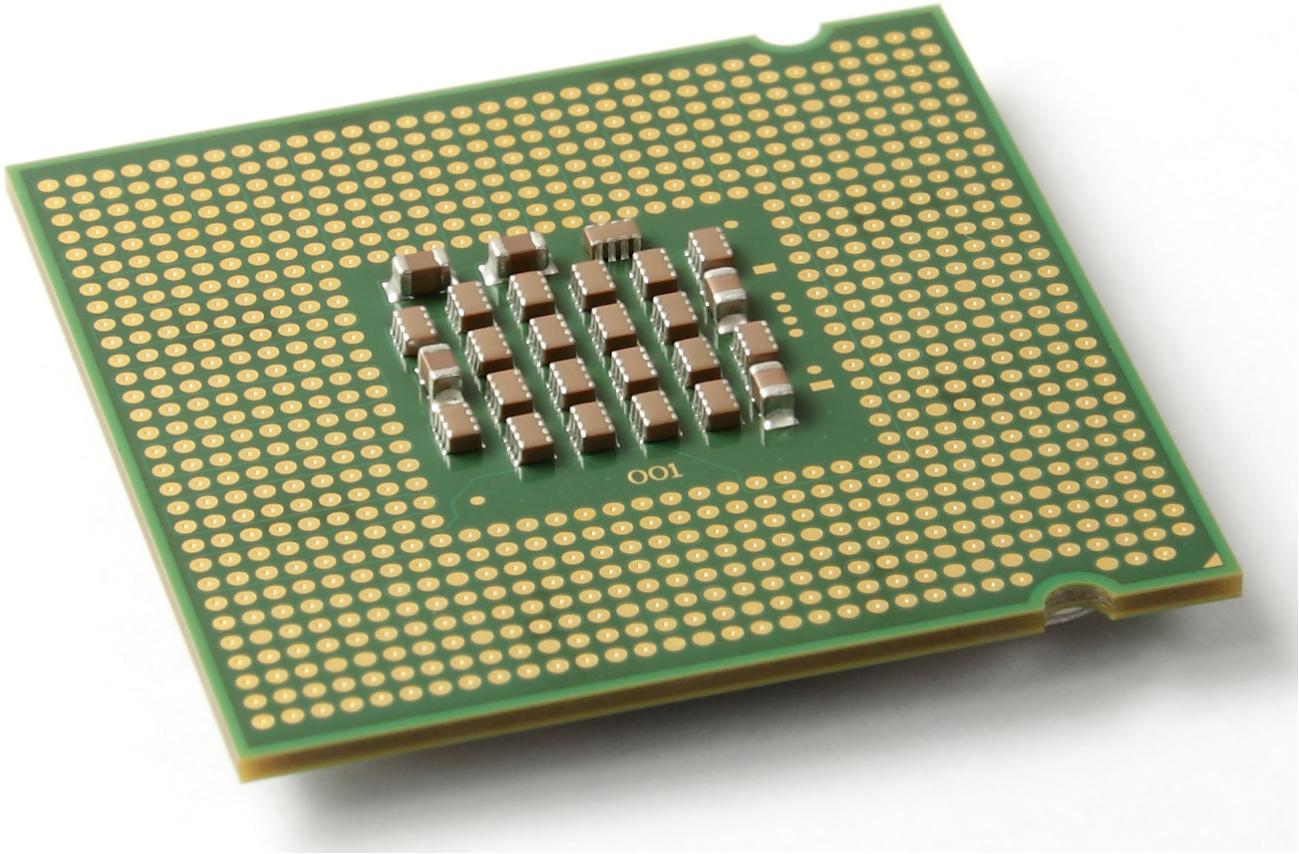
- The characterisation of the adversary can be done through the identification of the type, number and frequency of the deviations to the specified behaviour of the attributes



  - Omissive: fail-stop, crash-stop, crash-recovery, crash-link, receive, send and general omissions
  - Assertive: Syntactic/Semantic, Byzantine

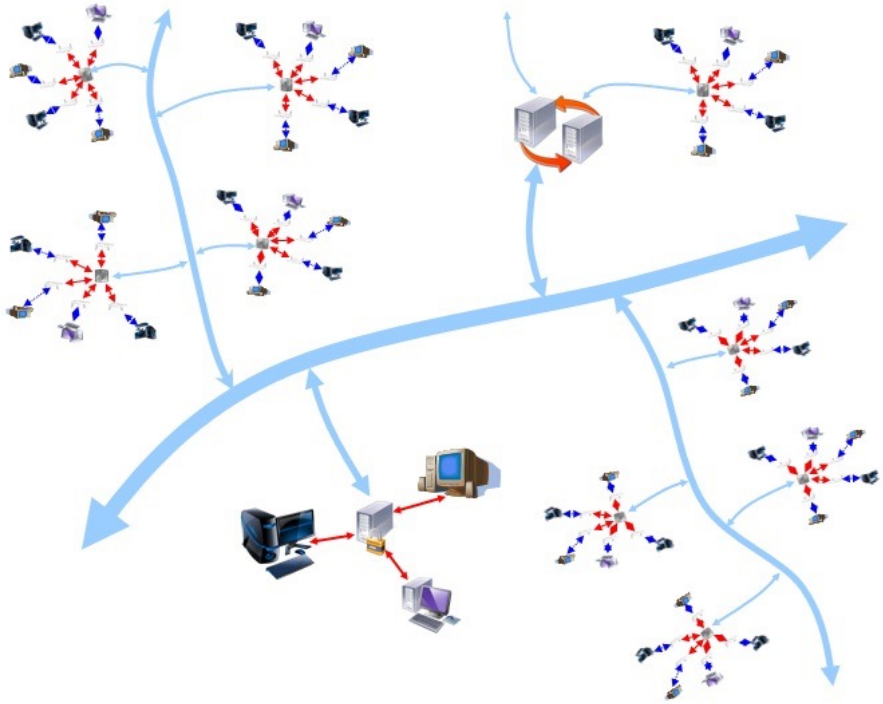
Models of distributed systems - Processing

- Set of processes
  - Unknown
  - Known size
  - Known ids
- Process faults
  - Crash
  - Crash and recovery
  - Misbehave arbitrarily



Models of distributed systems - Communication

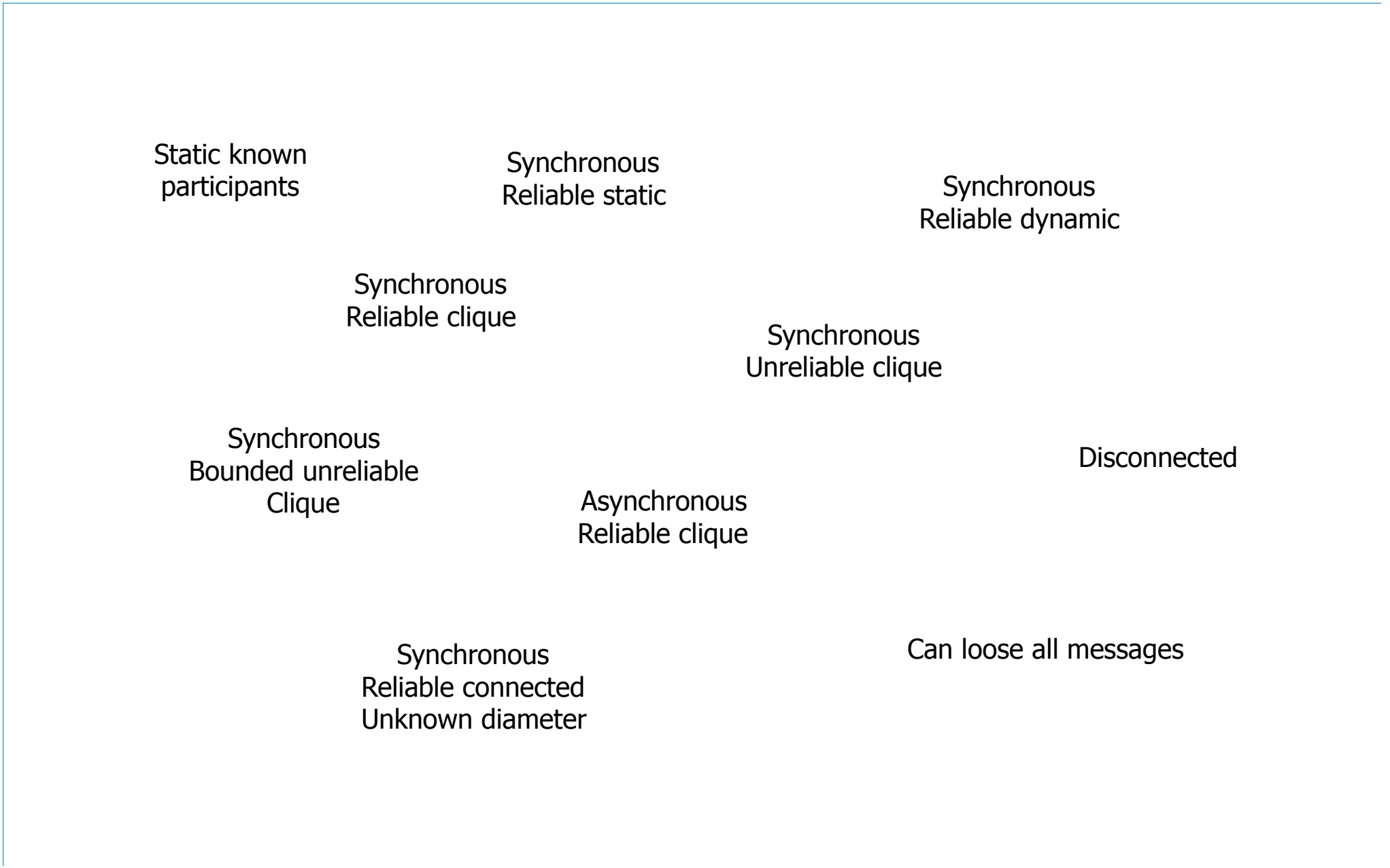
- Topology:
  - Fully connected (clique)
- Connected graph:
  - Static or Dynamic
  - Partitions
    - Transient or Permanent
- Reliability:
  - From all messages lost to no messages lost
  - Can messages be created?



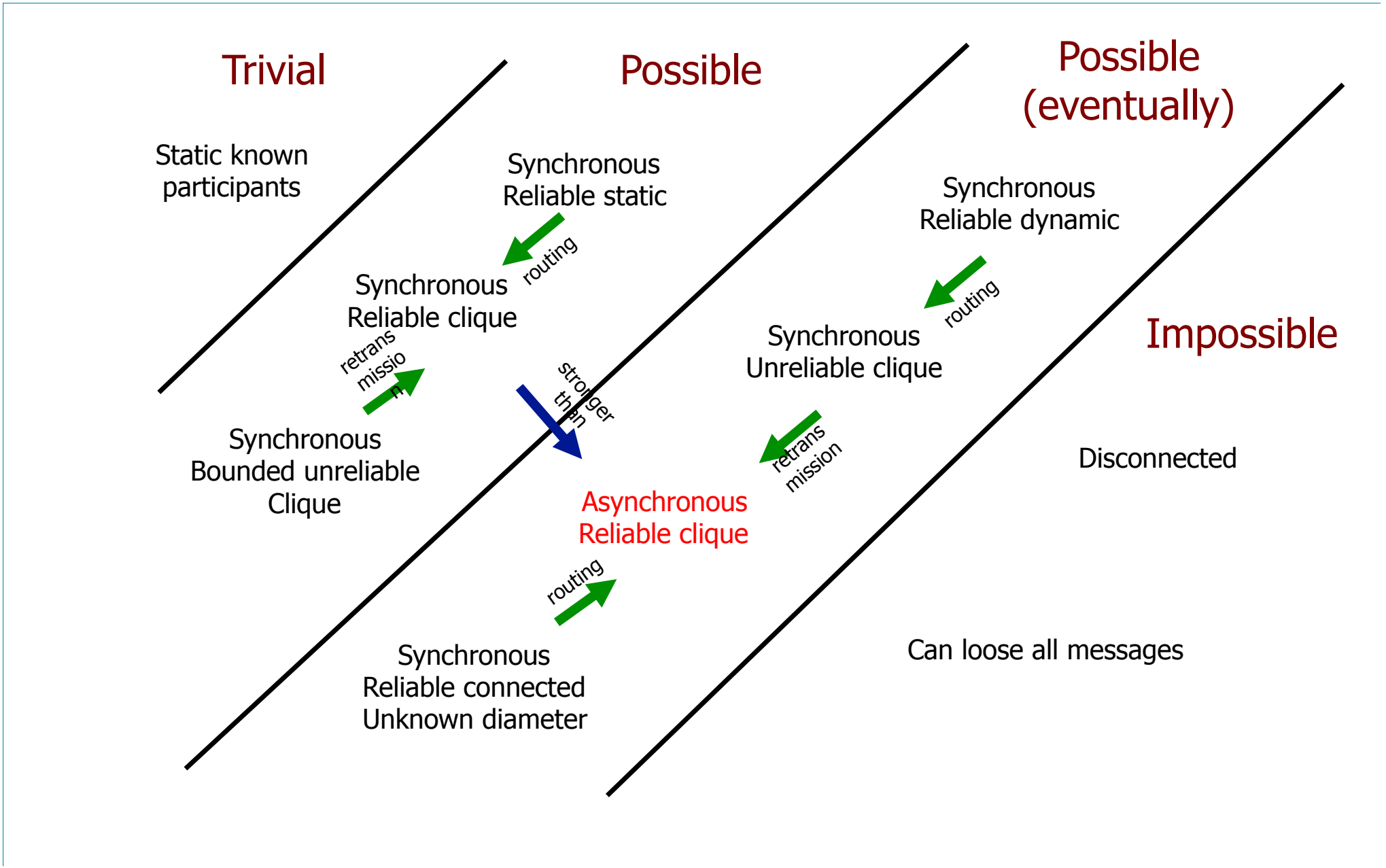
Models of distributed systems - Time

- Synchronous:
  - Known upper bounds on message delays, processing delays, and clock differences
- Partially synchronous:
  - Unknown fixed bounds
  - Known eventual bounds
- Asynchronous:
  - No assumptions on bounds and clocks

Models - What kind of problems can be solved?



Feasibility: Leader Election



Models of distributed systems and related faults

Reading material

- Fred. B. Schneider

  “What good are models and what models are good?”

  DISTRIBUTED SYSTEMS (2ND ED), SAPE MULLENDER (ED.), ADDISON-WESLEY 1993