UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Distributed Systems

Introduction and basic concepts

Contents

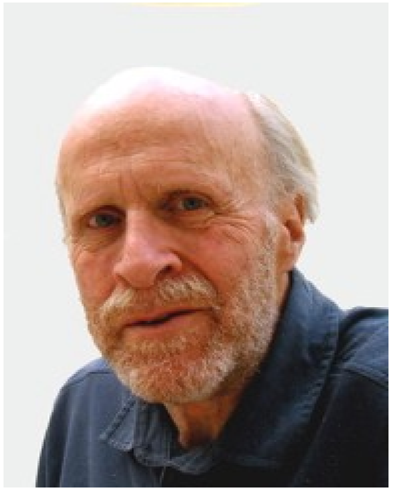
- Distributed System concepts and examples
- Features, pros and cons.
- Nomenclature: protocols, clients, services and servers
- Middleware
- Distributed computing paradigms

UNIVERSIDAD DE CASTILLA-LA MANCHA

Definitions

A **distributed system** is one in which components located at networked computers communicate and coordinate their actions **only** by passing messages.

__G. Coulouris__, J. Dollimore & T. Kindberg (2001)



The main characteristics of distributed systems:

- Concurrent execution of processes
- Lack of a global clock
- Independent failures

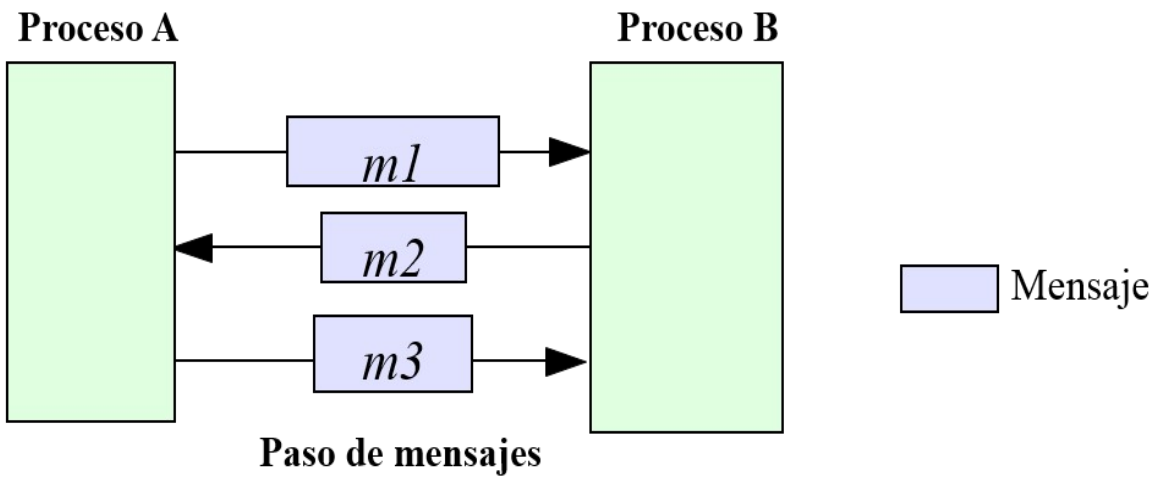
Message passing

Fundamental model of distributed applications:

- A sending process sends a request message.
- The message arrives at the receiving process, which processes the request and returns a reply message.
- This reply can generate further requests from the sending process.

Services exchange messages defined for each specific application.

- Data Oriented
- Each message must be interpreted



Independent failures?

- Client and server are independent processes potentially running on different nodes. They may have particular circumstances and restrictions.
- Example:

> RPC termination
>
> In gRPC, both the client and server make independent and local determinations of the success of the call, and their conclusions may not match. This means that, for example, you could have an RPC that finishes successfully on the server side (“I have sent all my responses!”) but fails on the client side (“The responses arrived after my deadline!”). It’s also possible for a server to decide to complete before a client has sent all its requests.
>
> https://grpc.io/docs/what-is-grpc/core-concepts/

Distributed System

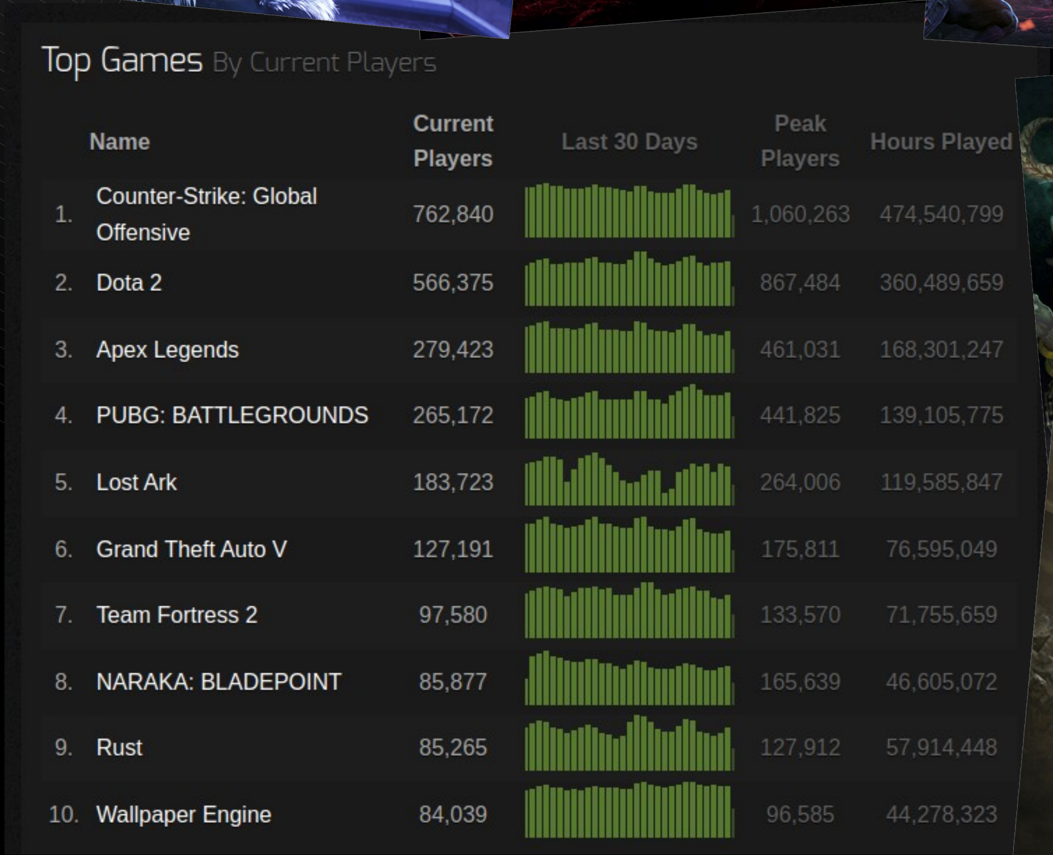
System consists of:

- computer resources (hardware and software)
- physically distributed
- connected through a computer network
- that communicate and coordinate their actions by passing messages and
- cooperate to perform a task, providing an integrated service.

Distributed System

Examples?

Example: MMOs
Massively Multi. Online Games

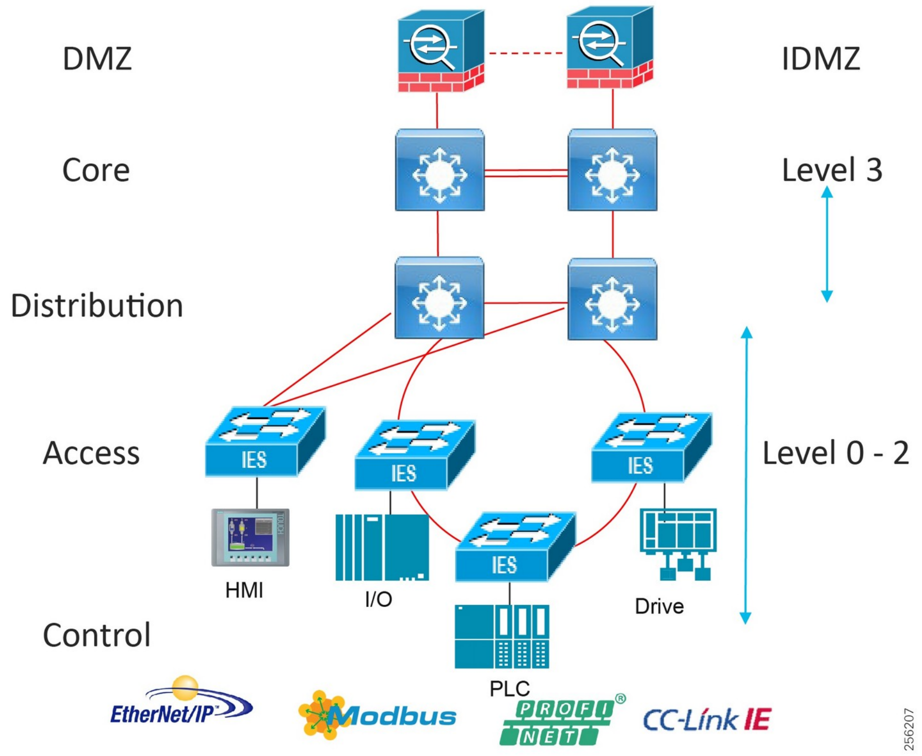


Top Games By Current Players

	Name	Current Players	Last 30 Days	Peak Players	Hours Played
1.	Counter-Strike: Global Offensive	762,840		1,060,263	474,540,799
2.	Dota 2	566,375		867,484	360,489,659
3.	Apex Legends	279,423		461,031	168,301,247
4.	PUBG: BATTLEGROUNDS	265,172		441,825	139,105,775
5.	Lost Ark	183,723		264,006	119,585,847
6.	Grand Theft Auto V	127,191		175,811	76,595,049
7.	Team Fortress 2	97,580		133,570	71,755,659
8.	NARAKA: BLADEPOINT	85,877		165,639	46,605,072
9.	Rust	85,265		127,912	57,914,448
10.	Wallpaper Engine	84,039		96,585	44,278,323

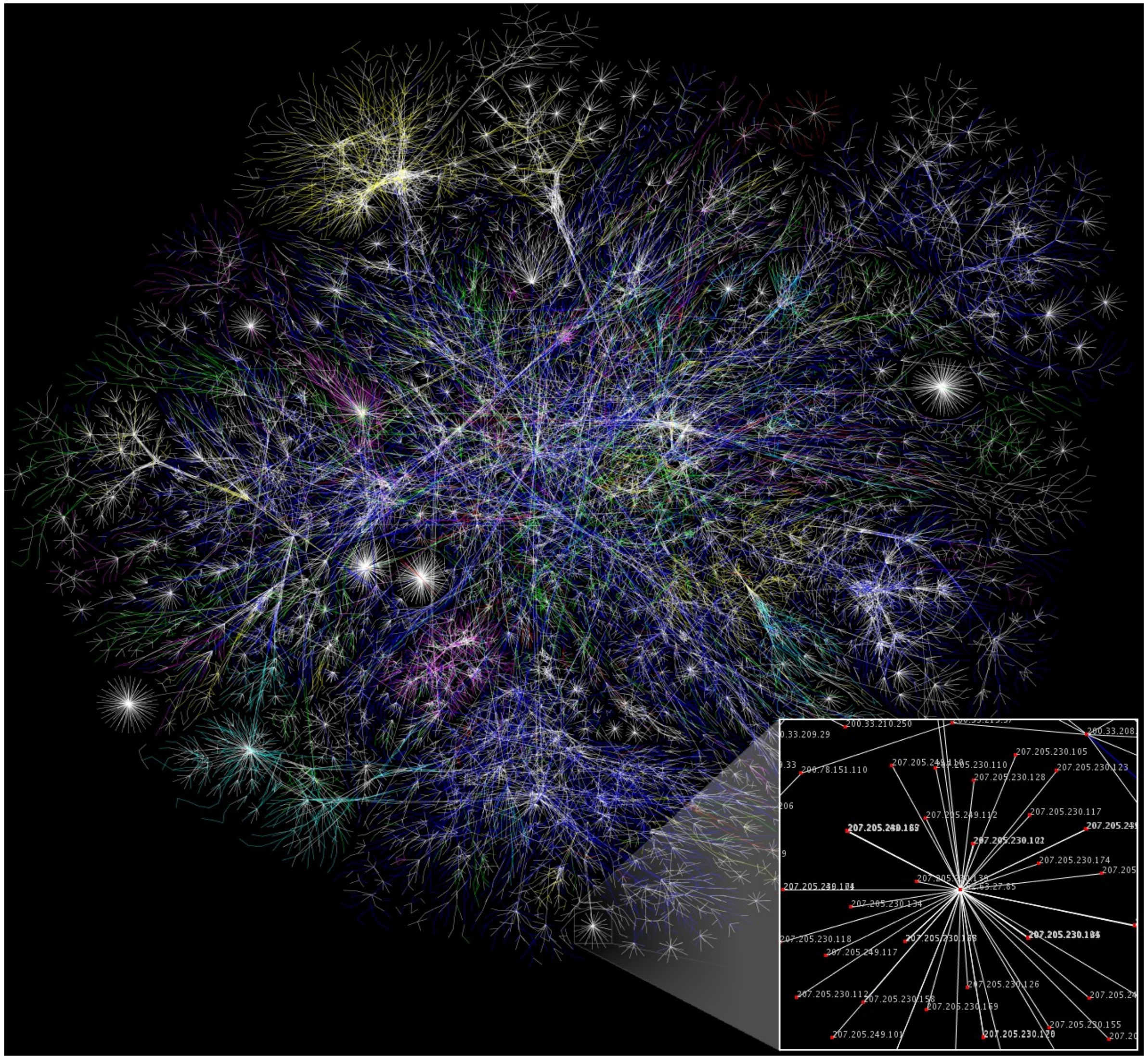
Source:https://steamcharts.com/

Example: plant control



Example

Internet and web search



- $63 \cdot 10^9$ web pages
- $1 \cdot 10^{12}$ URLs

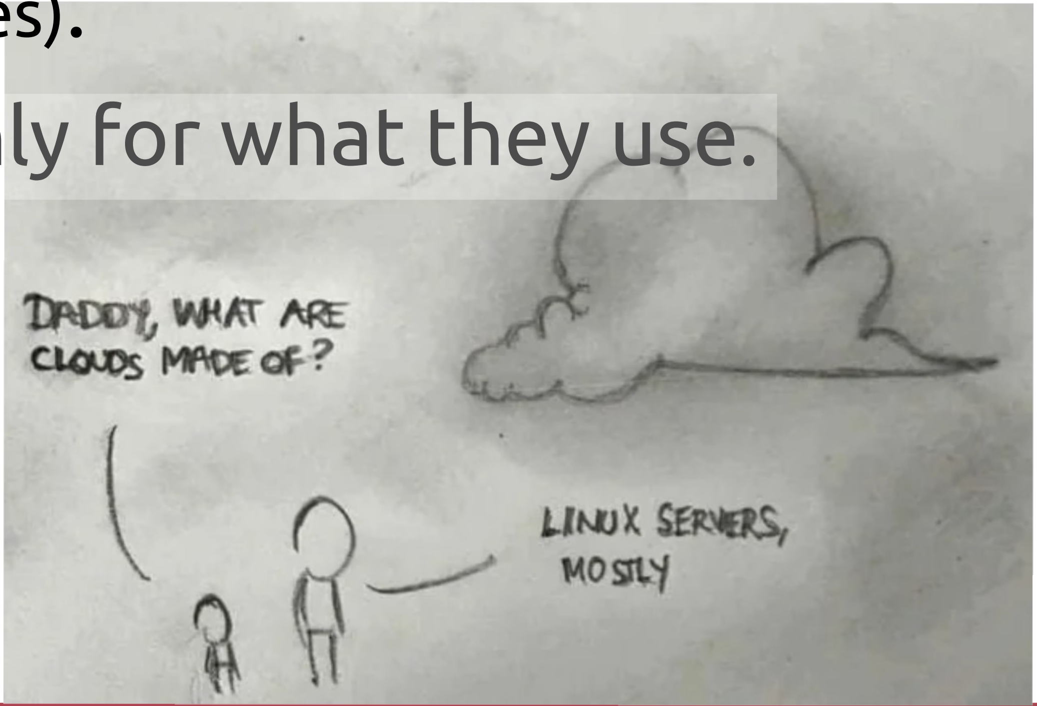
Google: 40,000 search queries every second

http://www.internetlivestats.com/google-search-statistics/

Example
Cloud Computing

Cloud computing provides arbitrary software and hardware IT resources over the Internet:

- On-demand.
- That scale with the workload (they are "elastic" resources).
- The customer pays only for what they use.



Example
Cloud Computing types

Infrastructure as a Service (IaaS): It provides IT resources (nodes, storage, networks), but the customer manages them.

- *Examples: AWS, Azure, Google Cloud*

Platform as a Service (PaaS): The customer is only concerned with the application build. The provider manages the IT resources.

- *Examples: Heroku, Google App Engine*

Software as a Service (SaaS): Both the infrastructure and the application itself are managed by the provider.

- *Examples: Microsoft 365, Slack, Dropbox*

Pros and cons

Pros:

- Sharing resources (hardware, software, data)
- Good cost/performance ratio
- Growth capacity (scalability)
- Higher availability (fault tolerance)
- Concurrency: simultaneous service to multiple users
- Speed: increased processing capacity

Cons

- Interconnection of resources (cost, reliability, saturation, etc.)
- Communications security
- More complex software

Nomenclature

- **SERVICE** – a component managing a **collection of related resources** and provides functionality to users and applications
- The only access to the service is via a well-defined **set of operations** that it exports.
  - **Example**: A file server just provides *read*, *write* and *delete* operations.
- For efective sharing, each resource must be managed by a program that offers a communication interface. This **communication interface** composed by a **set of operations** is often called **Application Programming Interface (API)**
- The APIs are a key concept in new data-based companies
- Example: Instagram

https://blog.api.rakuten.net/

Nomenclature

- **SERVER** – A running program (a *process*) on a networked computer that accepts **requests** from another program running on other computer to perform a **service** and **responds** appropriately.
- The requesting processes are referred as **clients**, and the overall approach is known as **client-server computing**.
- When a client sends a request for an operation, we say that the client invokes an operation upon the server.
- A **server** can implement/provide one or several **services** using, usually, an API for each service.

Nomenclature

- A protocol is the **set of rules** that enable two entities to connect and transmit data to one another (message exchange).
- The **set of messages** that a server can accept refers to a concrete protocol.
- A **protocol** specification requires:
  - Syntax
  - Semantics
  - Synchronization / Timing
- Many RFCs contains protocol specifications

Client and server is a …

DIY

- program | role
- process | device
- always running | just a while
- passive | active
- invokes | replies
- heavy | light
- simple | complex

Client or Server?

DIY

- ping
- Firefox (web browser)
- Squid (web proxy)
- Dropbox
- *Other?*

Designing a distributed system
The 8 Fallacies of DS

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

— Peter Deutsch & James Gosling

UNIVERSIDAD DE CASTILLA-LA MANCHA

Designing a distributed system **Challenges**

- **Heterogeneity** of hardware, software, networks and uses.
- **Openness**: ready to extend, provides open interfaces.
- **Security (CIA)**: Confidentiality, Integrity, Availability
- **Scalability**: Remain effective when system grows (more resources and users)
- **Failure handling**: Ready to manage failures in any process, computer or network.
- **Concurrency**: Resources may be requests concurrently.
- **Transparency**: Show the system as a whole rather than a set of components. Hiding details to users/programmers.
- **Quality of service**: how the service should be provided:
  - Reliability, security and performance.

UCLM UNIVERSIDAD DE CASTILLA-LA MANCHA

Designing a distributed system
Challenges: **Heterogenety**

Variety and difference in its components:

- Networks: topology, technology, configuration, etc.
- Hardware: servers, mobiles, laptops, printers, robots, motors, sensors, etc.
- Operating Systems: Windows, Linux, Android, FreeRTOS, VxWorks, etc.
- Programming languages: Java, C++, Python, C, Lisp, etc.
- Programmers, experience, know-how...
- E.g. KPI: Time of integration of a new technology.

How do we solve heterogeneity?

- Using open systems

Designing a distributed system

Challenges: **Heterogenety**

OPEN SYSTEM

Property that determines if the system can be extended and reimplemented

- Specifications, APIs, public, accessible, well documented protocol stacks.
- Public and open standards
  - E.g. RFCs

http://www.omg.org/spec/index.htm

Designing a distributed system
Challenges: **Scalability**

A system has the capacity to grow (**scalability**) if it maintains its effectiveness when the number of resources or users increases significantly

- Example: Internet growth
- Ex. KPI: Number of users served per minute/hour/.

Aspects to be considered:

- Cost/Performance
- Resource availability (ex. IPv4 addresses)

The concept of **cloud** was born, among other reasons, to provide ad-hoc scalability.

https://www.crn.com

Designing a distributed system Challenges: **Concurrency**

- **Simultaneous** execution of processes
- A resource can be shared by several clients concurrently.
  - Same issues addressed in concurrent and real-time scheduling
    - If the resource is replicated, it must preserve consistency

Designing a distributed system Challenges: **Concurrency**

The consistency problem (coherence) arises when several processes access and update data concurrently:

- Consistency of updates
- Coherence of replication
- Cache consistency
- Consistent clocks

Concurrency → **Consistency**

Designing a distributed system Challenges: **Reliability**

- The probability that a system will work (meet your requirements) for a period of time
- To achieve reliability you need to guarantee:
  - Failure treatment
  - Consistency
  - Security

Designing a distributed system
Challenges: **Reliability**

FAILURE HANDLING

Distributed systems often exhibit **partial failure**:

- The objective is to increase the availability of the distributed system
- Availability: percentage of time a system is available for use

Fault tolerance:

- Detection
- Failure toleration
- Failure recovery
- Redundancy

Designing a distributed system Challenges: **Security**

Security has three components:

- Confidentiality
  - Unauthorized access
- Integrity
  - Data corruption or alteration
- Availability
  - System or resource not accessible (DoS attacks)

Tools:

- Security at different levels
- SSL, Public Key Infrastructure, VPN, etc.

Designing a distributed system Challenges: **Transparency**

Hiding details of the system components from the user:

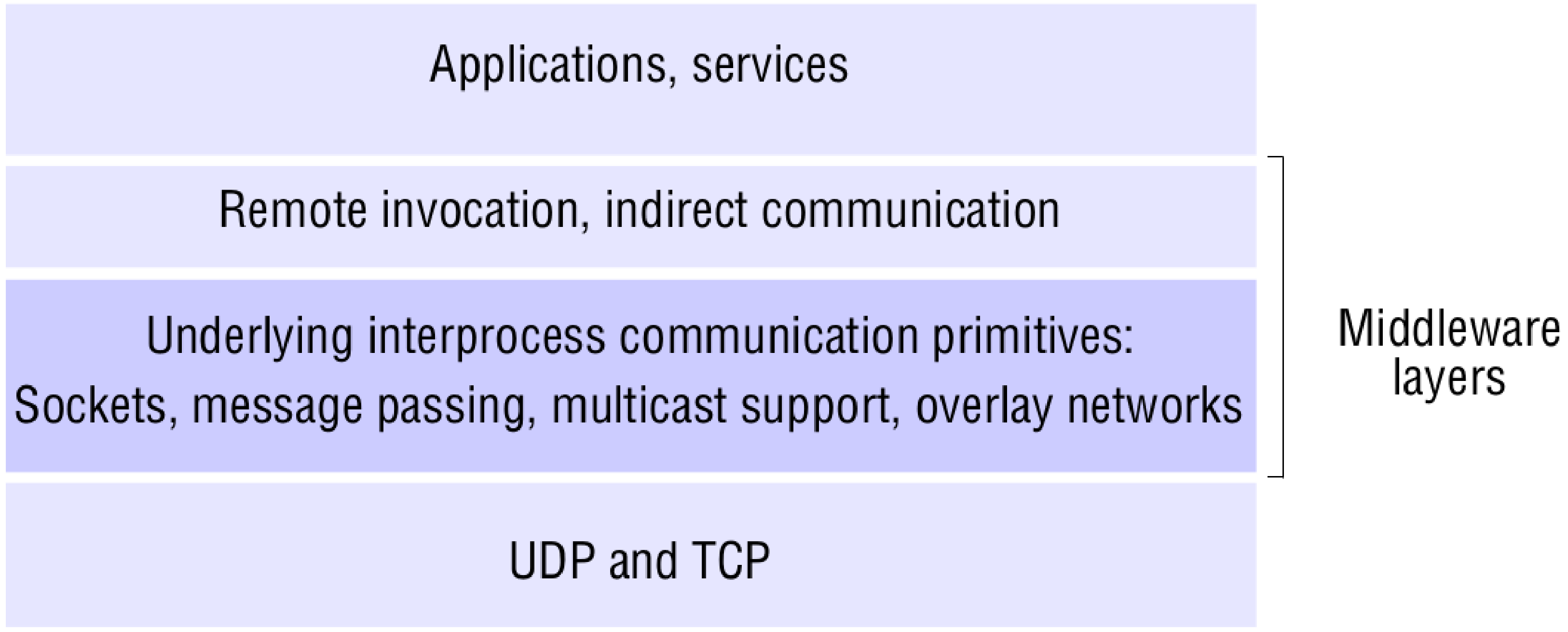
- **Access**: The same interface (operations) to access local and remote resources.
- **Location**: Access is independent of the location of the resource.
- **Concurrency**: Several processes access the same resource concurrently.
- **Replication**: Users are not aware of the existence of replicas.
- Of **failures**: Fails are hidden from clients (but not all).
- **Mobility**: of both resources and user without affecting their operations.
- **Performance**: allowing optimization according to the load.
- **Scalability**: it works the same way regardless of the amount of resources or clients.

What kind(s) of transparency provides?

DIY

- e-mail
- RAID (Redundant Array of Independent Disks)
- URL
- DNS
- VRRP (Virtual Router Redundancy Protocol)

Distributed Computing: **communication paradigms**


Applications, services
Remote invocation, indirect communication
Underlying interprocess communication primitives: Sockets, message passing, multicast support, overlay networks
UDP and TCP
Middleware layers

Source: Colouris

- Client/Server: Sockets
- Remote invocation: RPC & RMI
- Indirect communication

Middleware

A middleware is a software infrastructure (which may include tools, libraries, protocols, services, etc.) that aims to facilitate the development of applications in distributed environments.

- It allows **masking the heterogeneity** of the underlying networks:
  - It supports various programming languages
  - It supports various operating systems
  - Etc.

<u>Provide a common programming model for developers.</u>

RabbitMQ GRPC ZeroC mosquitto ORACLE FUSION MIDDLEWARE

Middleware
Increases developer productivity

The main goal of middleware is to isolate the developer from the non-functional features of his application/service.

- Ideally, problems with resource location, failures, performance, scalability, etc. should be transparent to the developer.

From the developer's point of view:

- Isolate yourself from the SD particularities
- Make your development transparent
- Focus on functional requirements

The term *middleware* is coined.

How to face heterogeneity

Universal references

- A unique non-ambiguous way to locate resources.
- Resource Naming.

Canonical representation

- Common network-level data representation and format.

Language bindings

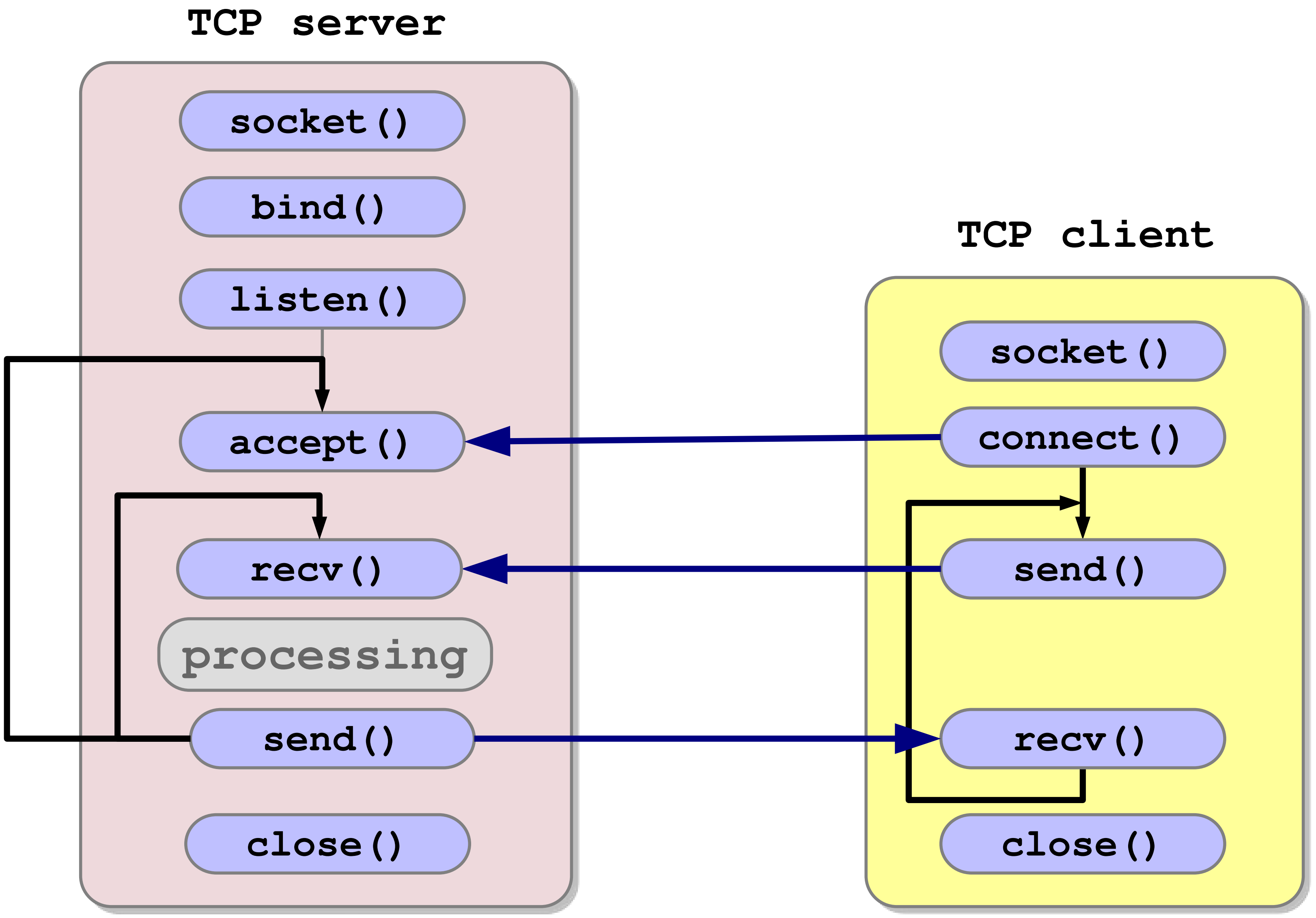
- Definition of common interfaces.
  - The contract between peers.
- How to map representations to different programming languages.

Sockets

IPC mechanism (ie, provides communication between processes) eventually running on different computers.

- Virtually universal low-level interface
- Access to link, network and transport layers
- Very efficient, but complex and error-prone
- Distributed application features are implemented ad-hoc
  - A protocol is designed for each application.

Client/Server with sockets
Example: TCP server

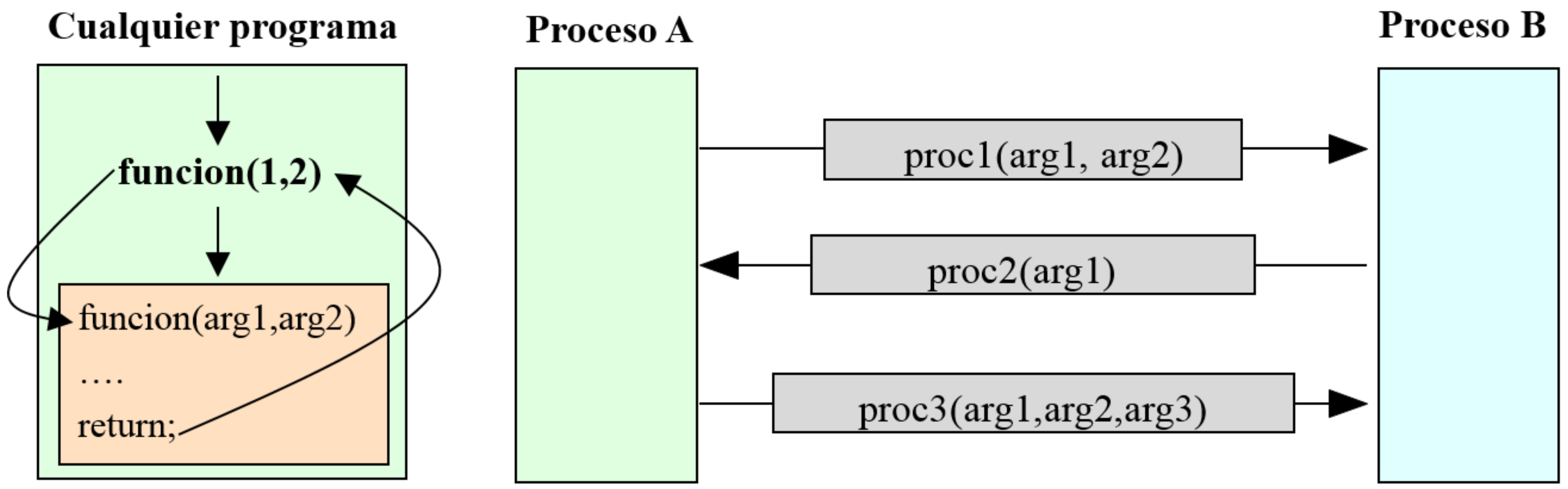


Transparency?
Reliability?
Scalability?

RPC: Remote Procedure Call

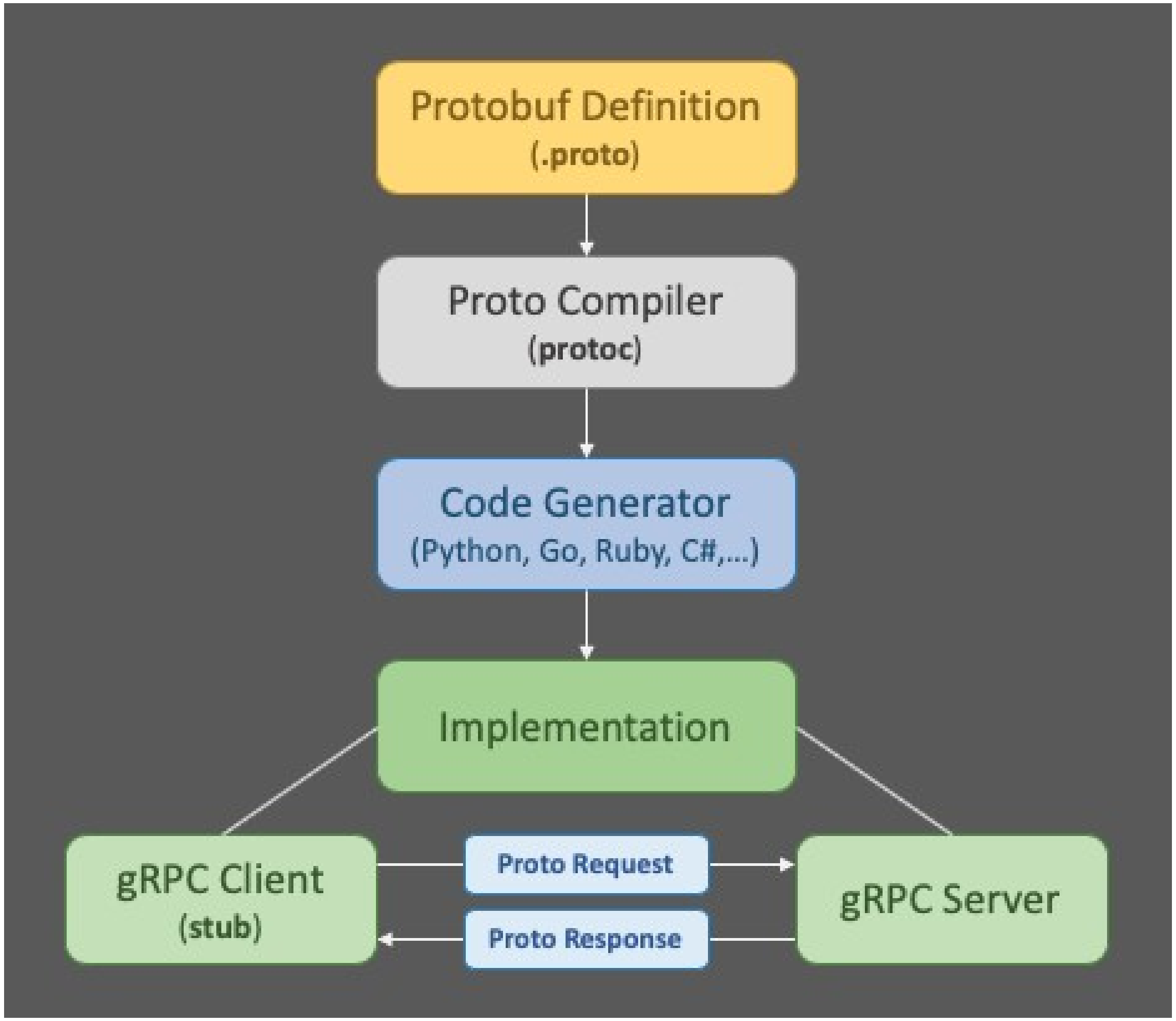
Idea: Make distributed software programming the same way a conventional (non-distributed) application.

- Conceptually the same as invoking a local (same node) procedure



Google gRPC

github.com/grpc



Fuente https://geeks.ms

RMI: Distributed Objects

- RPC counterpart for OOP.
- Resources are encapsulated as objects that are accessed by clients
  - An object is an abstraction of **operations + state**

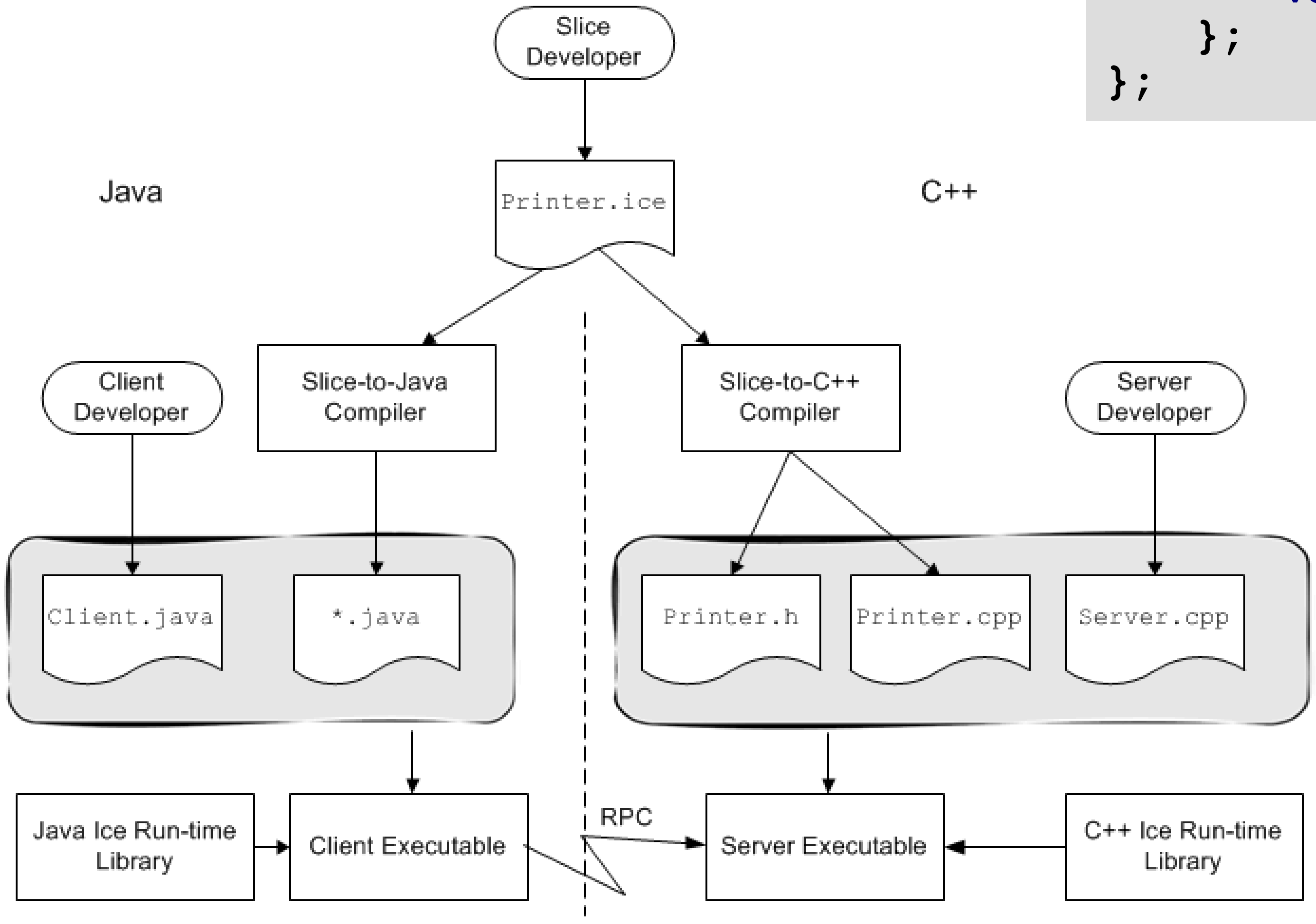
- Execute object methods as if they were local.
  - Client's Stubs: Representation of a remote object in local (including proxies).
  - Server's stubs: executes the method on the invoked object (performs the dispaching).

RMI: Distributed Objects
ZeroC Ice

- Internet Communications Engine: distributed object oriented middleware.
- Include “common services” for:
  - persistence
  - location of object references
  - replication
  - load balancing
  - service management
  - event-oriented programming, etc.

ZeroC Ice
Development model

```
module Example {
    interface Printer {
        void write(string s);
    };
};
```



http://doc.zeroc.com/display/Ice/Slice+Compilation

Indirect communication

Some applications are better suited to an event-driven paradigm.

- Events distributed by publishers are generated in event channels that are...
  - filtered (source vs. destination)
  - grouped
  - analyzed
  - replicated
- Received by subscribers
- An event management application (broker) manages the distribution of events and subscriptions

Scalability and **efficiency** are the key.

Indirect communication

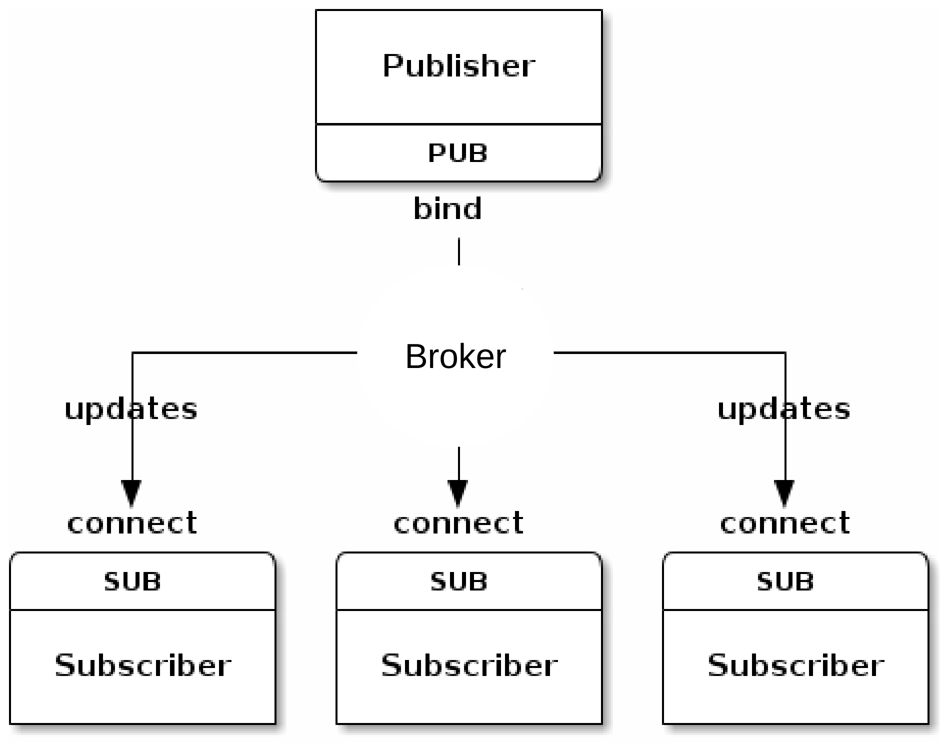
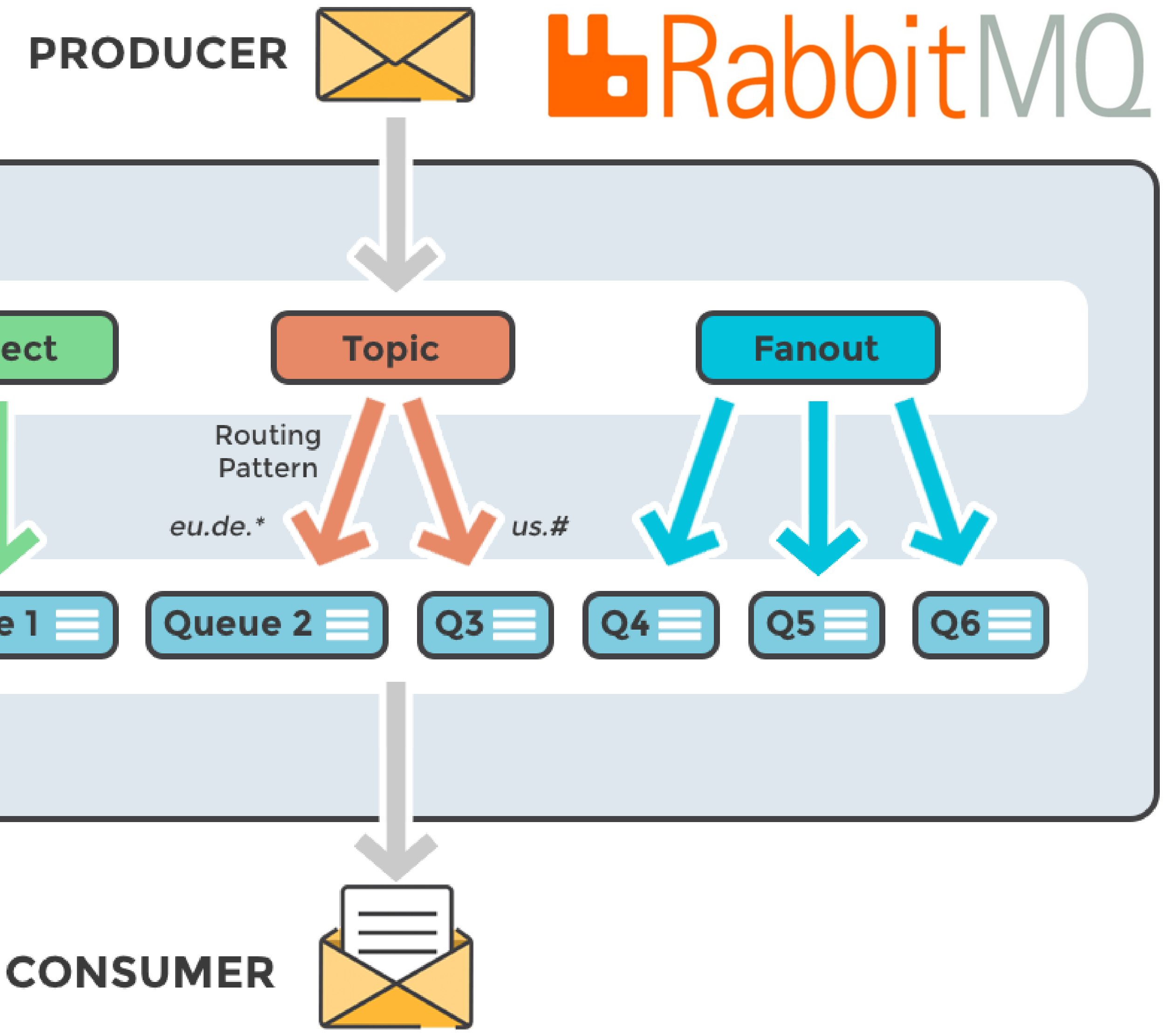


Figure 4 – Publish-Subscribe

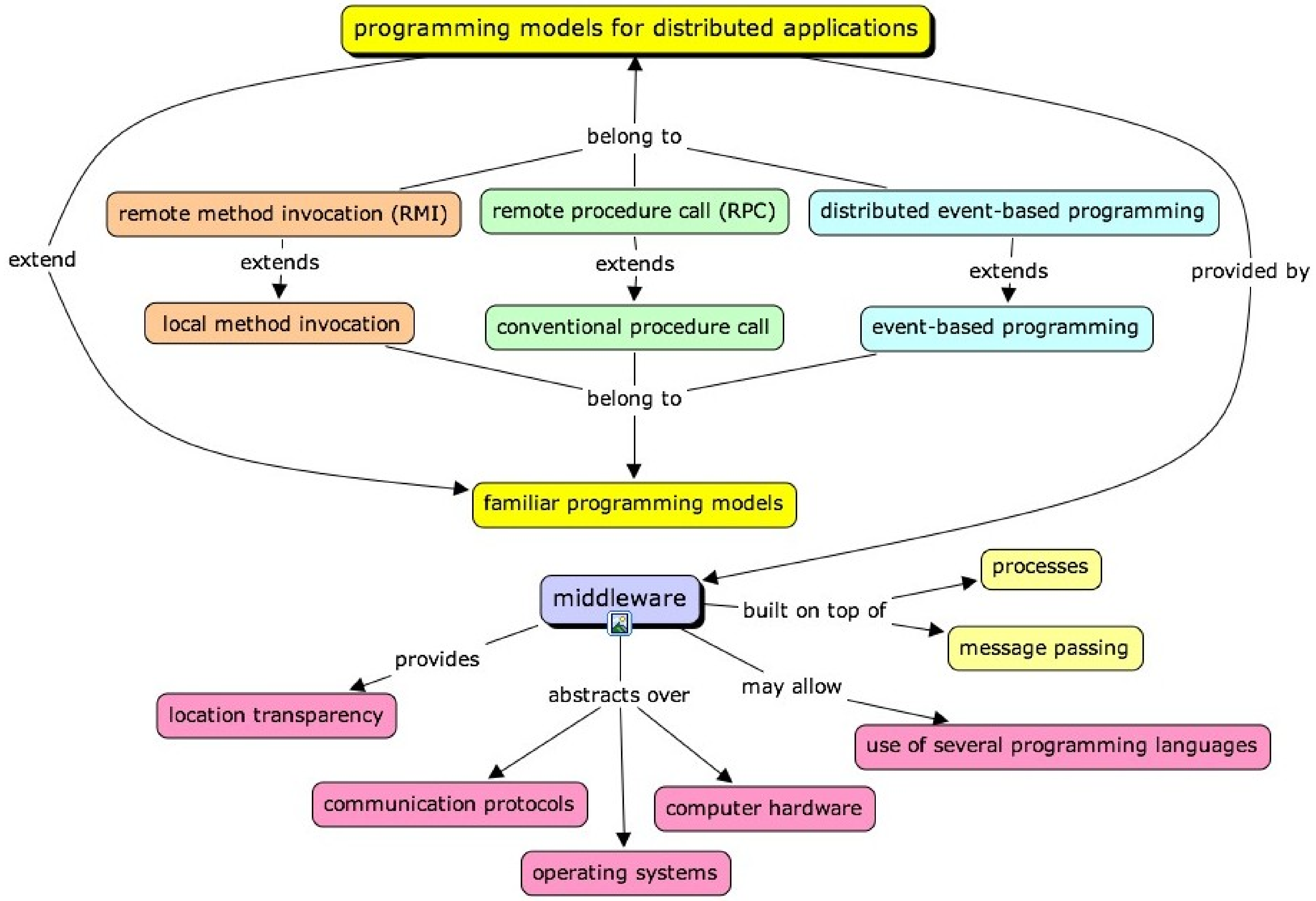
- pull/push model
- Event channel management
- Filtering: source/destination, type, content, etc.

Indirect communication
RabbitMQ



Source: https://cloudamqp.com

Middleware overview



Fundamental models

Models that allow us to focus on fundamental aspects of DS to be more specific:

- Interaction Model
- Failure Model
- Security Model

Interaction model

Any DS involves process communication by passing messages to achieve a goal:

- Communication: interaction flow
- Coordination: synchronization and order

Two crucial factors in the interaction:

- Communication channel performance (latency)
- Lack of global clock (clock drift and synchronization): two processes running on different computers could associate different timestamps to the events

Two models: synchronous vs. asynchronous

Interaction model

Synchronous distributed systems

- Minimum and maximum time limit for each process execution step
- Each transmitted message is received in a limited and known time.
- Each process has its local clock and its known drift

Asynchronous distributed systems

- In practice, most of them.
- They don't make any assumptions about the relative speeds of the processes, nor the delays of the messages, nor the drifts
- Channels are reliable, but there is no time limit to message delivery
- Communication between processes is **the only way to synchronize**

Failure model

Definition and classification of the failures that can affect a DS:

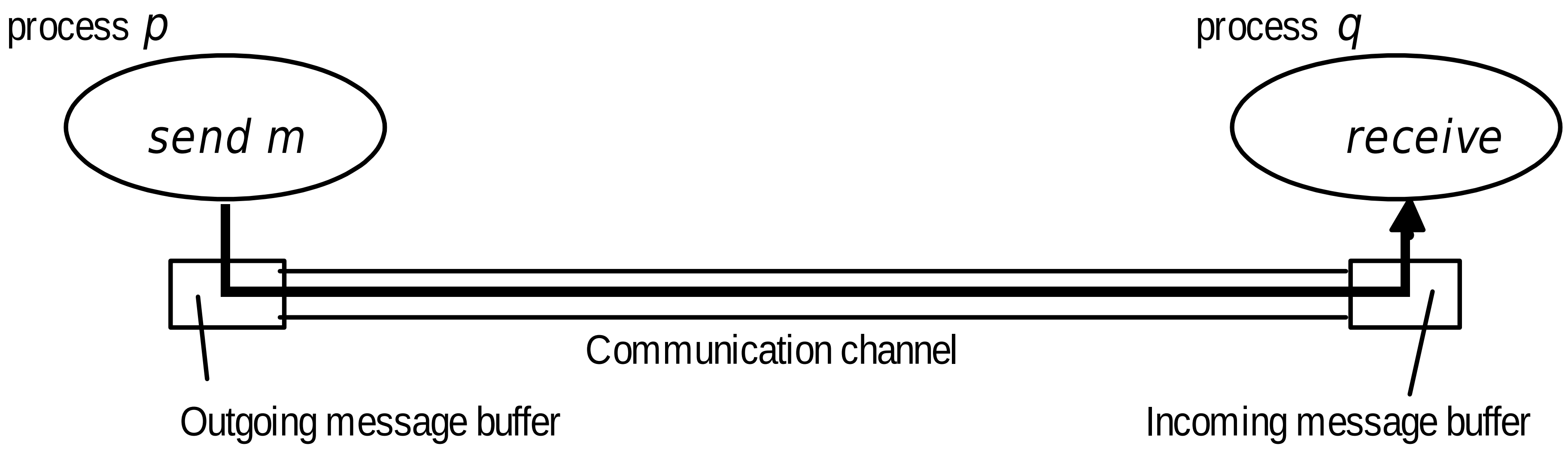
- Basis for the analysis of potential effects and for the design of fault-tolerant systems
- Types of failure:
  - Omission: Process or communication channel does not do what it is supposed to do
  - Time: not done in time
    - Only synchronous distributed systems
  - Arbitrary (Byzantine): any kind of error can occur

Failure model

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send* operation but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

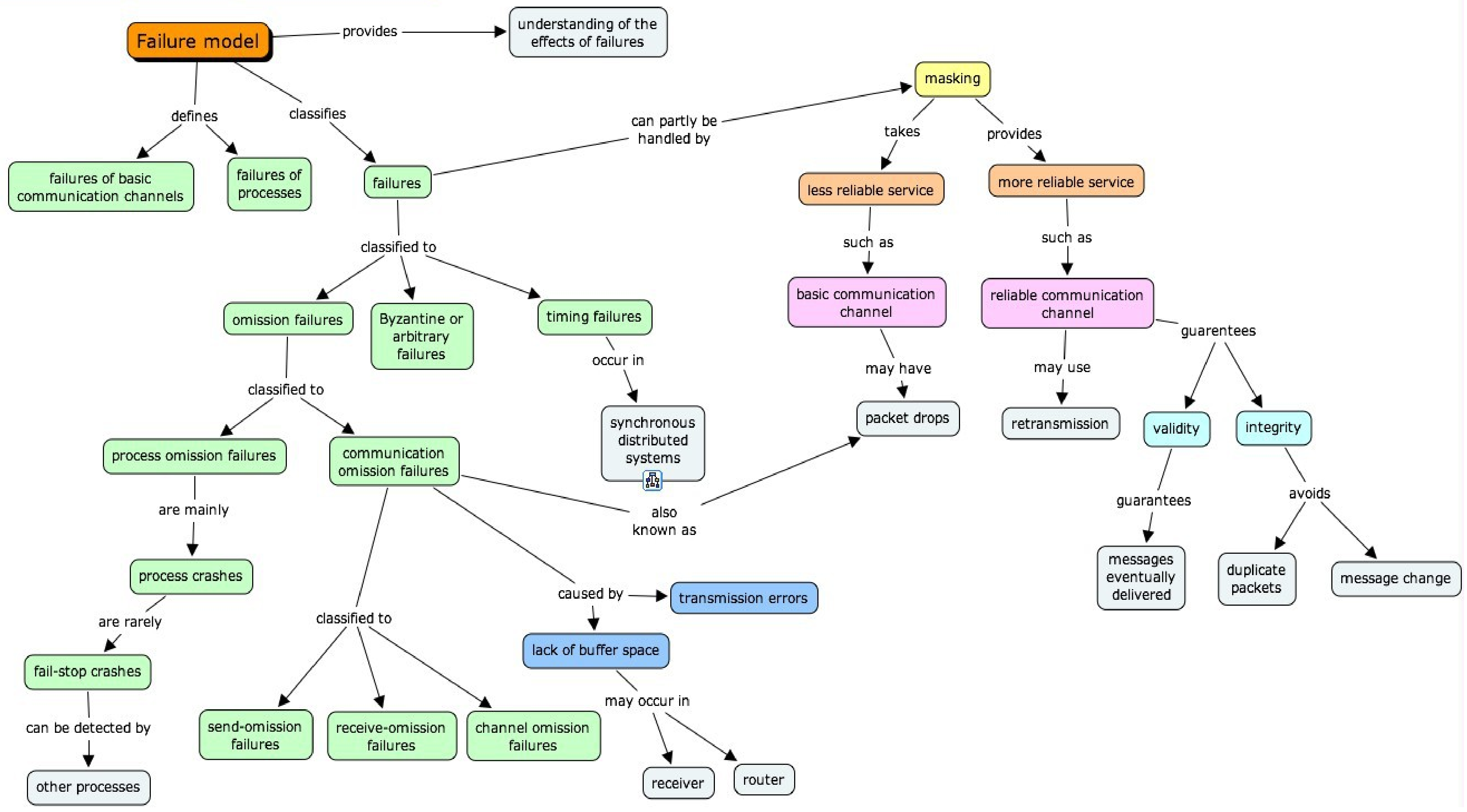
Failure model

- Channel failure vs. process failure



Failure model

Note! This is not independent study material. Read the book.



http://www.cs.hut.fi/Opinnot/T-106.5250/Cmaps/Ch05/Programming%20models%20for%20distributed%20systems.html

Security model

Definition and classification of attacks/threats that can affect a DS:

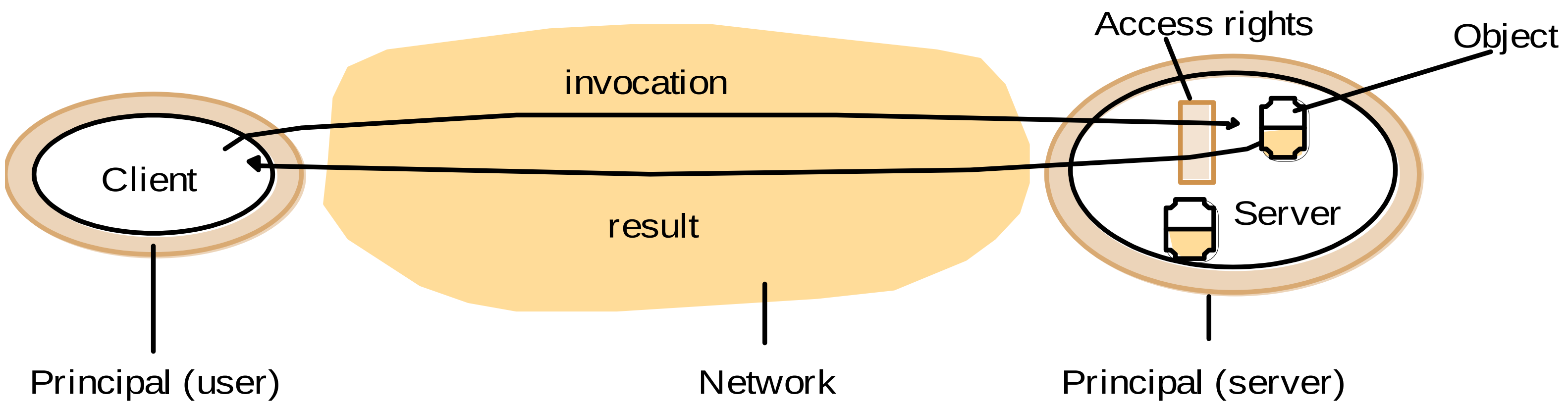
- Basis for analysis of potential threats and for building robust systems

Safety can be achieved:

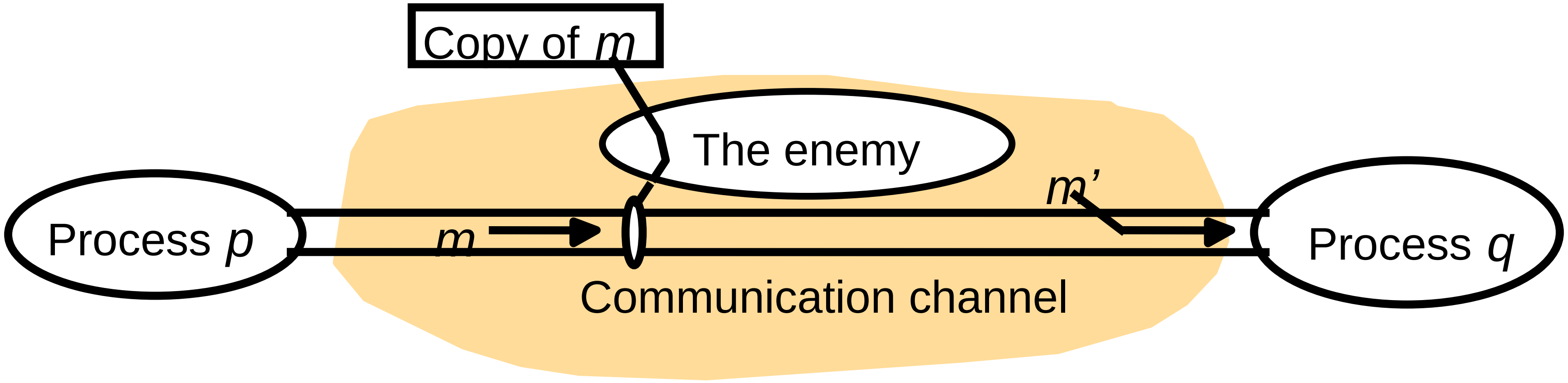
- Protecting objects
  - Authentication: third party/mainframe identification
  - Authorization: only to beneficiaries of the (main) rights
- Ensuring processes and their interactions (communication channels)

Security model

- Protecting resources

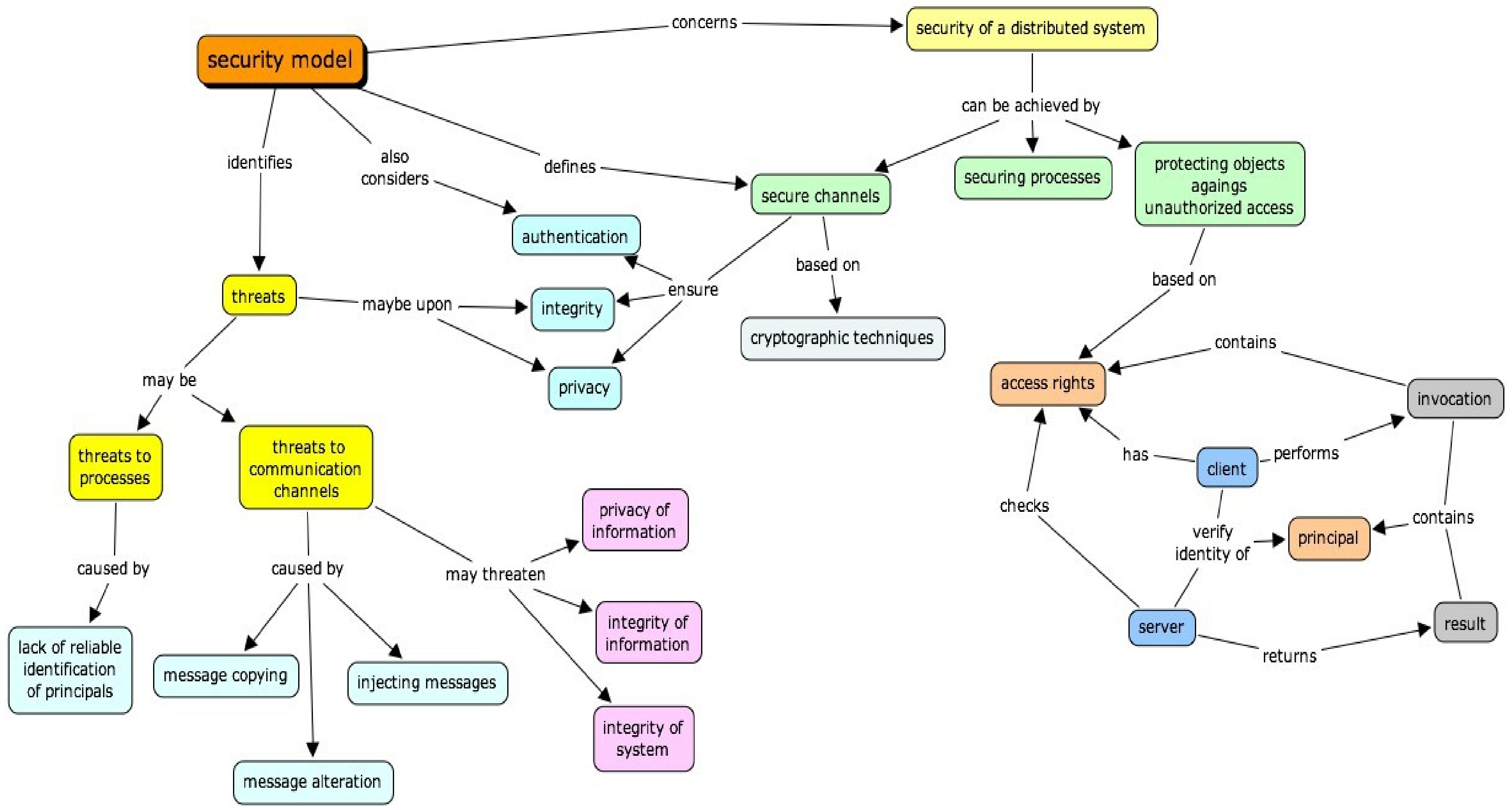


- Securing processes and their interactions



Security model

Note! This is not independent study material. Read the book.



http://www.cs.hut.fi/Opinnot/T-106.5250/Cmaps/Ch05/Programming%20models%20for%20distributed%20systems.html

References

G. Coulouris, *Distributed Systems: Concepts and Design*, Addison Wesley 2011

- Chapter 1 – Characterization of distributed systems.
- Chapter 2 – System models