Motivation

So far in this course

- We have solved *concurrency problems* using techniques for *concurrent programming with shared memory*

- But many systems nowadays are *distributed*:
  - Geographic distribution
  - *no shared memory*
Motivation

Distributed system. Definition

- Distributed system (def from wikipedia):

A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages.
The components interact with each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.
Motivation

Distributed system. Examples

- Computer networks. The Internet.
- Current supercomputers
- Banking systems
- Ubiquitous systems: collections of automobiles, appliances, buildings...
- Mobile phone networks
- Online games
- Social networks
- Cloud storage
Motivation

Distributed systems. **Advantages**

- **Economy**: scaling is much cheaper than in centralized computing.
- **Speed**: Computing power is easier to scale.
- **Inherent distribution**: Solving distributed problems requires distributed computing.
- **Reliability**: Fault tolerance (failure independence).
- **Flexibility**: Growth can be incremental.
- **Flexibility**: Different resources (like old computers) can be reused more easily.
Motivation

Distributed systems. Disadvantages and challenges

- **Efficiency**: a centralized system with the same resources is typically more efficient

- **Sharing resources**: imbalances in the use of resources may lead to inefficiencies

- **Consistency**: Data consistency is hard to impose

- **Software**: more complex and error-prone

- **Networks**: a new source of problems (message losses, latency, etc)

- **Security**: management of roles, agents, attacks, etc is more complex
Motivation

Distributed systems. Models

- **Cluster computing**: high-performance design for supercomputing
- **Grid computing**: Extension of clusters, with geographic distribution
- **Utility computing**: Sharing of external resources from providers
- **Volunteer computing**: donating computing power (e.g. SETI)
- **Cloud computing**: resources as services, based on virtualization
- **Autonomic computing**: no human intervention to steer and correct the evolution
- **Mobile Computing**: remote devices move, enter and exit the network
- **Ubiquitous Computing**: connected computers present in many objects interactively
Motivation

Distributed systems. For this course

- Distributed systems are **important**

- In terms of concurrent programming:
  - Threads **do not share memory**
  - Each thread accesses its own memory space
Motivation

Distributed systems. For this course

- Distributed systems are **important**

- In terms of concurrent programming:
  - Threads **do not share memory**
  - Each thread accesses its own memory space

- Mapa Conceptual:
  
  Concurrency = Simultaneous + Nondeterminism + **Interaction**

  Interaction = Communication | Synchronization

  Synchronization = Mutual Exclusion | Conditional Synchronization
Motivation

Distributed systems. For this course

- Distributed systems are important

- In terms of concurrent programming:
  - Threads do not share memory
  - Each thread accesses its own memory space

- Mapa Conceptual:
  
  \[
  \text{Concurrency} = \text{Simultaneous} + \text{Nondeterminism} + \text{Interaction}
  \]

  \[
  \text{Interaction} = \text{Communication} \mid \text{Synchronization}
  \]

  \[
  \text{Synchronization} = \text{Mutual Exclusion} \mid \text{Conditional Synchronization}
  \]

- Now threads do not share variables.
- New mechanism: message passing
Message Passing

Interaction between processes

Typical architecture: client / server

- A (server) thread offers services.
- **Client** thread: send messages requesting a service
- **Server**: repeatedly:
  - receive service requests from clients
  - performs the service
  - responds by sending a message to the client (if necessary)
- for implementing **resources**, servers will implement the resource.
Message Passing

- How is message passing implemented? Two operations:
  - `address.send(msg)`
  - `msg = address.receive()`
Message Passing

- How is message passing implemented? Two operations:
  - address.send(msg)
  - msg = address.receive()

- Two main issues
  1. **Addressing**: How to specify where to send messages.
  2. **Synchronization**: Are these new operations blocking?
Message Passing

Addressing

To send a message, we need to specify a **destination**.
- To perform a phone call, we dial a number.
- To send an email, we specify an email address.

Two main addressing schemes

- **by process**
  - Advantage: simple
  - Disadvantage: not flexible (not dynamic, force to need to fix which process does what).
  - Disadvantage: Communication 1:1

- **by channel**
  - Advantage: clean separation of processes and channels
  - Advantage: dynamic, different channels for different purposes (even between the same processes)
  - Communication 1:1, n:1 or even n:m.

We will use this
Message Passing

Synchronization

- Are operations blocking?
Message Passing

Synchronization

- Are operations blocking?

- Typically receiving is blocking:
  ```python
  msg = channel.receive()
  ```
Message Passing

Synchronization

- Are operations blocking?
- Typically receiving is blocking:
  
  ```python
  msg = channel.receive()
  ```

- For sending
  
  ```python
  channel.send(msg)
  ```
  
  *Synchronous communication*: sending is blocking (until receiver has received)
  
  ○ No buffer is needed
  
  ○ Easier to reason about the program

  *Asynchronous communication*: Sending is **not** blocking
  
  ○ More performance
  
  ○ Less latency

We will use this to implement resources
Message Passing

Example

prod$_1$
prod$_2$
prod$_3$

Almacen$_1$

put(x)

hayDato?

d

get

cons$_1$
cons$_2$
cons$_3$
Message Passing

Example

![Diagram showing message passing example]
Message Passing

Example

prod_1
prod_2
prod_3

Almacen1

put(x)

hayDato?

get

cons_1
cons_2
cons_3

d 3
Message Passing

Example

prod₁
prod₂
prod₃

Almacen₁

put(x)

hayDato?

get

d 3

cons₁
cons₂
cons₃

ch ch
Message Passing

Example