# Introduction to Distributed Systems

Fabienne Boyer, LIG, fabienne.boyer@imag.fr

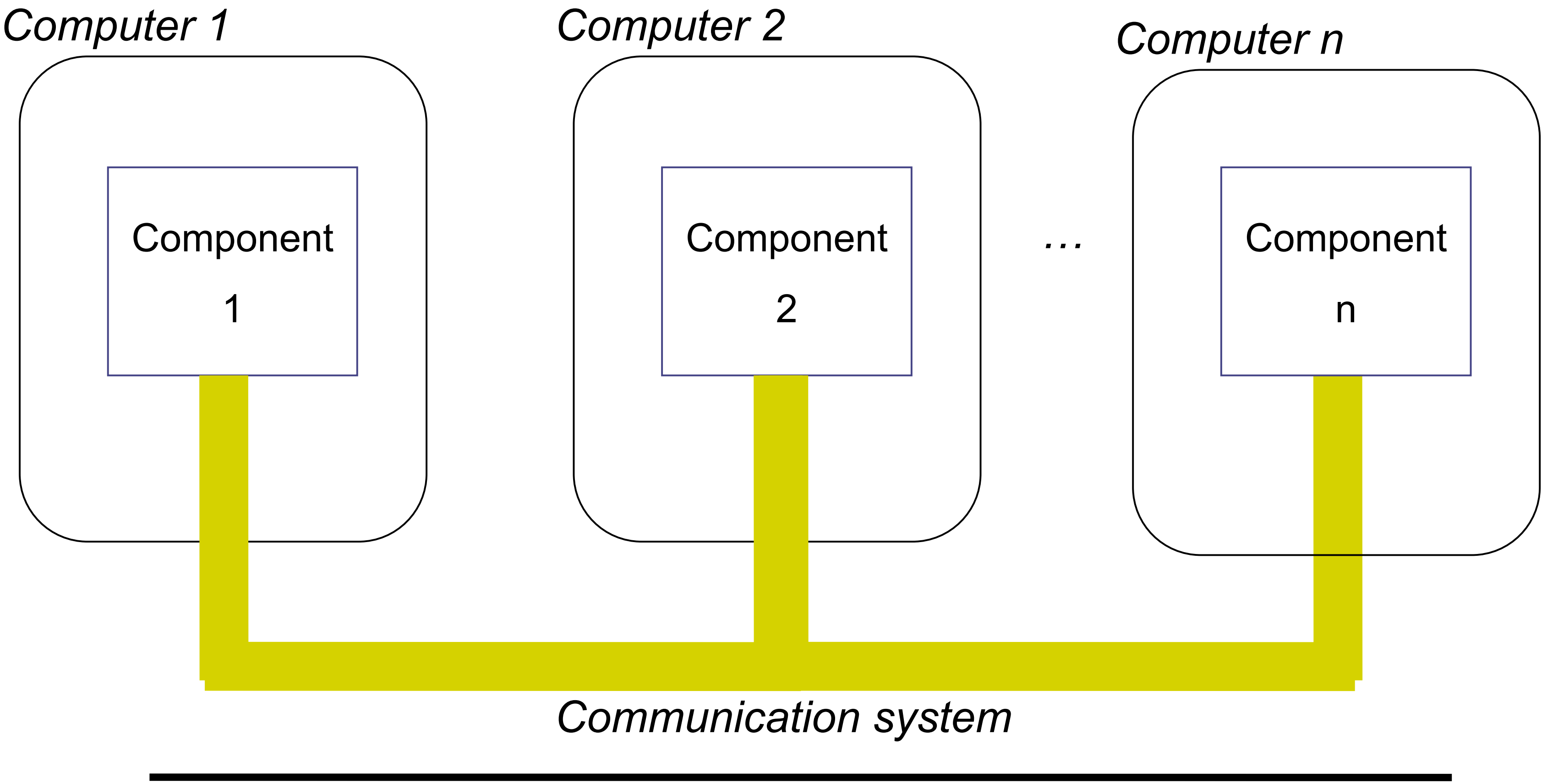
# What is a distributed system?

- Set of software components
- Running in separate address spaces
- Communicating through a network
- Collaborating to a common goal

***A distributed system is one that stops you from getting any work done when a machine you've never heard of crashes.***

***Leslie Lamport***

# What is a distributed system?



# Examples of distributed systems

- Information sharing
  - Wikipedia
- Collecticiels
  - Teleconference
  - Cooperative edition
  - Workflow (BPM)
- Real-time systems
  - Flying control systems
- Business
  - E-commerce

- Distributed Games
- Naming Servers
  - DNS
- Network File Servers
  - AFS, NFS
- Printing Servers

# Objectives of the course

*Study conceptual and practical aspects of distributed systems*

We will characterize a distributed system by

- Its properties
- Its architecture
- Its distributed protocol
- Its programming artefact
- Its execution model

# Expected properties of distributed systems

## Expected properties

- Availability
- Scalability
- Manageability
- Security

## Main issues

- Asynchronism
- Dynamicity
- Heterogeneity
- Size

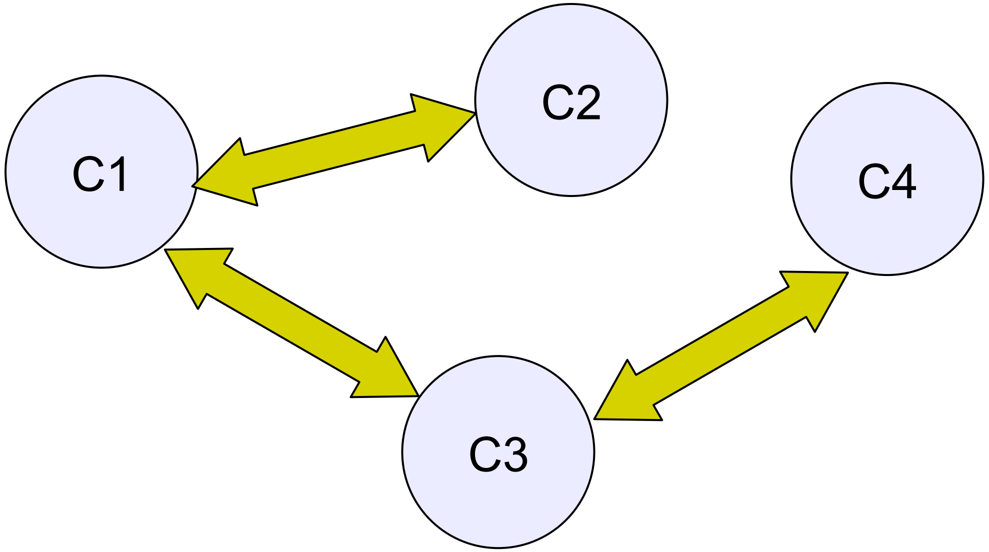
# Characterization of distributed systems

## *Distributed Programming Model*

| Architecture | Distributed Protocols | Programming Artefacts | Execution Model |
|---|---|---|---|
| Client-Server | Synchronous | Message | Thread |
| Multi-Tiers | Asynchronous | Procedure | Process |
| Peer-to-Peer | … | Object | Event |
| … | | Service | |
| | | Agent | |
| | | … | |

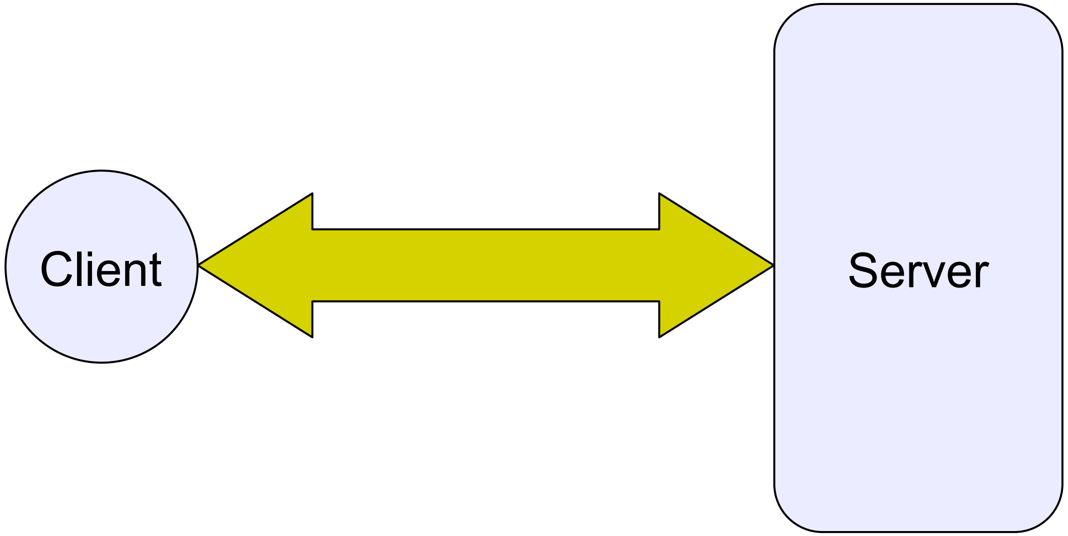
# Architecture of distributed systems

- What software components (processes) form the system
- How are they connected
- What are their roles



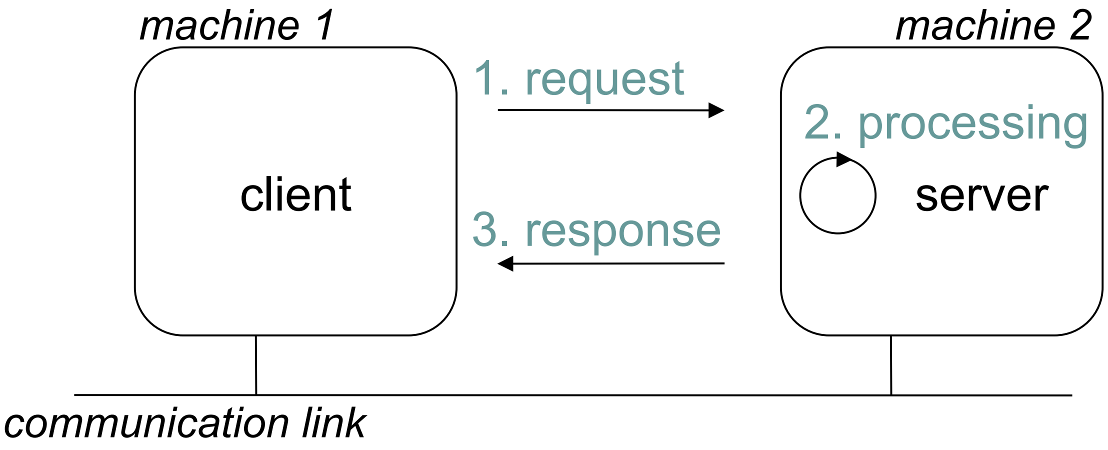
# Client-Server architecture

- A client uses some services provided by a server



# Client-Server architecture

- General client-server distributed architecture
  - The server provides a service
  - A client may request that service
- The client and the server are usually hosted by two distinct machines



# Client-Server architecture

- Request message
  - Sent by the client to the server
  - Specifies the requested service (a server may provide several services)
  - Contains parameters of the requested service
- Response message
  - Sent by the server to the client
  - Results of service execution, or error message

# Client-Server Contract

- 3 main aspects: Interface + Behavior + QoS

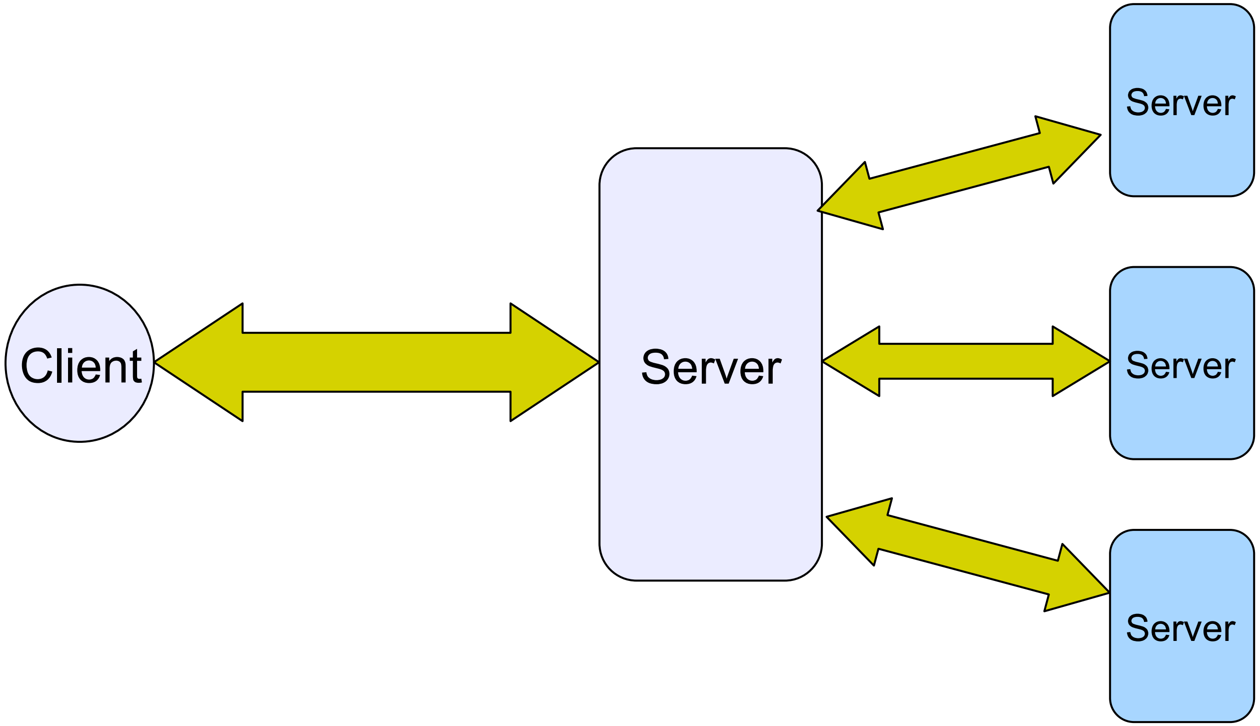
- The interface defines the format of the information exchanged between a service provider and its client
  - IDL (Interface Definition Language)
  - Examples: IDL Corba, Java interfaces, …

- The behavior drives the exchanges between a service provider and its client
  - Example: session types (state charts)

- The QoS defines non-functional aspects
  - Atomicity, reliability (MTBF), execution time, price, security, ..

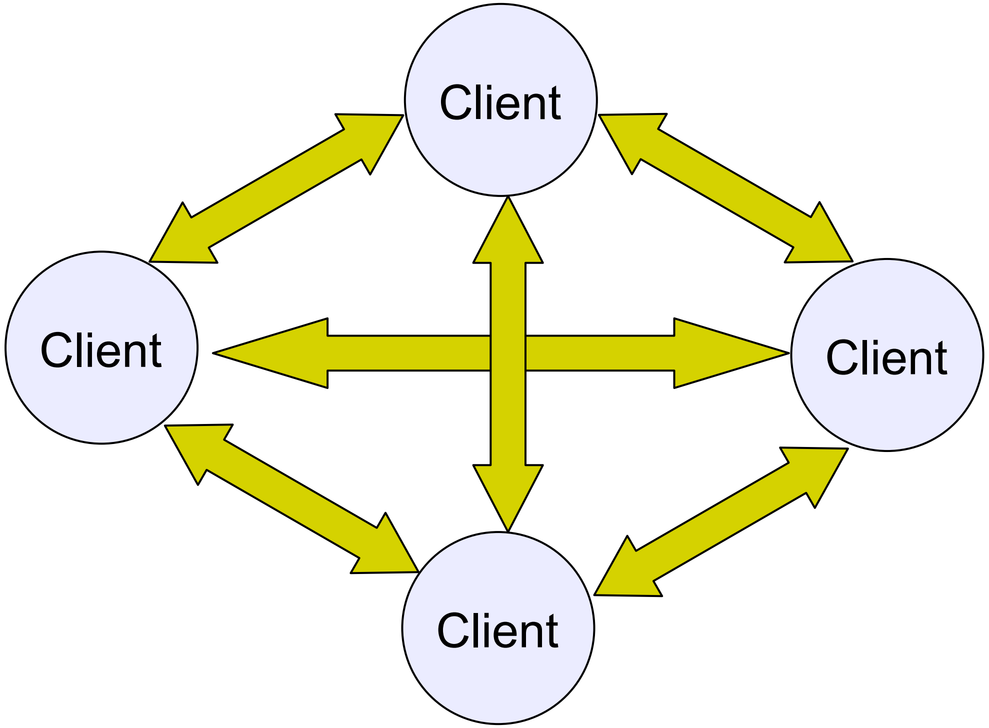
# Multi-tiers architecture

- The multi-Tiers client-server architecture allows to parallelize client requests



# Peer-to-peer architecture

- Software components play the role of both clients and servers



# Distributed Programming Model

- The distributed programming model mainly determines
  - What information is exchanged,
  - From who to who,
  - How it is exchanged,
  - When it is exchanged
  - How it is managed at sending and receiving times

# Distributed Programming Model

- What kind of information is exchanged
  - Data
  - Request (procedure call, object invocation)
  - Code
  - Object
  - Agent / Thread

- From who to who
  - Direct (process to process)
  - Indirect (exchanges go through a shared area)

- With what protocol
  - Synchronous (either blocking or non-blocking)
  - Asynchronous (generally non-blocking)
  - + Order, + reliability,+  atomicity

# Distributed protocol

- Blocking
  - When the client sends a request, it waits until the server replies to its request
- Non-blocking
  - A client does not wait for the result, it will be notified of (using client callbacks)

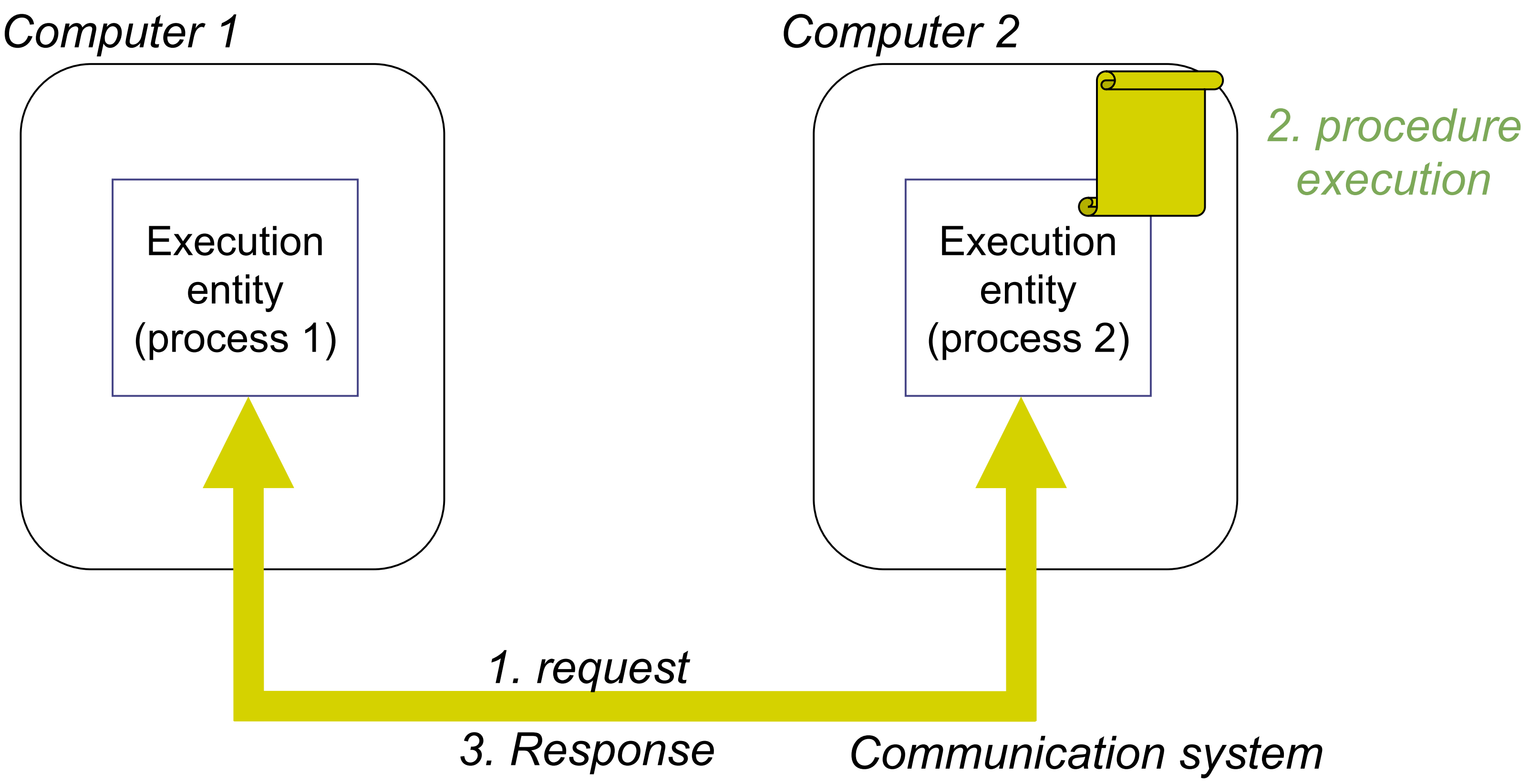
- Synchronous
  - When the client (resp. server) sends a request (resp. a reply), the server (resp. client) should be running
- Asynchronous
  - When the client (resp. server) sends a request (resp. a reply), the server (resp. client) may not be running

# Distributed protocol

- Ordered (or not)
  - Requests are *delivered* to the server in the order they are sent to the client
- Reliable (or not)
  - No request lost
- Atomic (or not)
  - A request is either entirely executed at the server side, or not at all

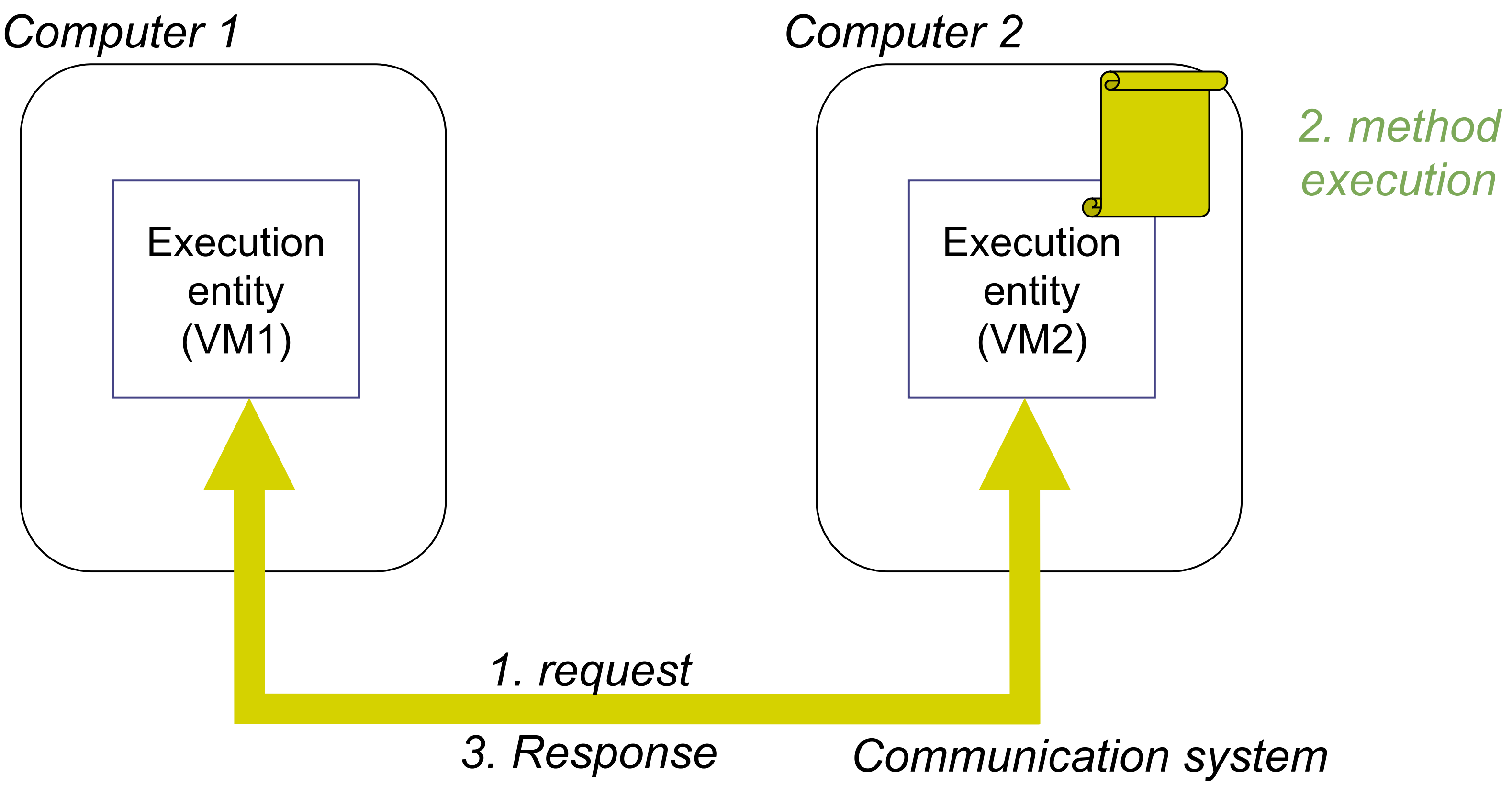
# Distributed programming model

- Remote procedure call



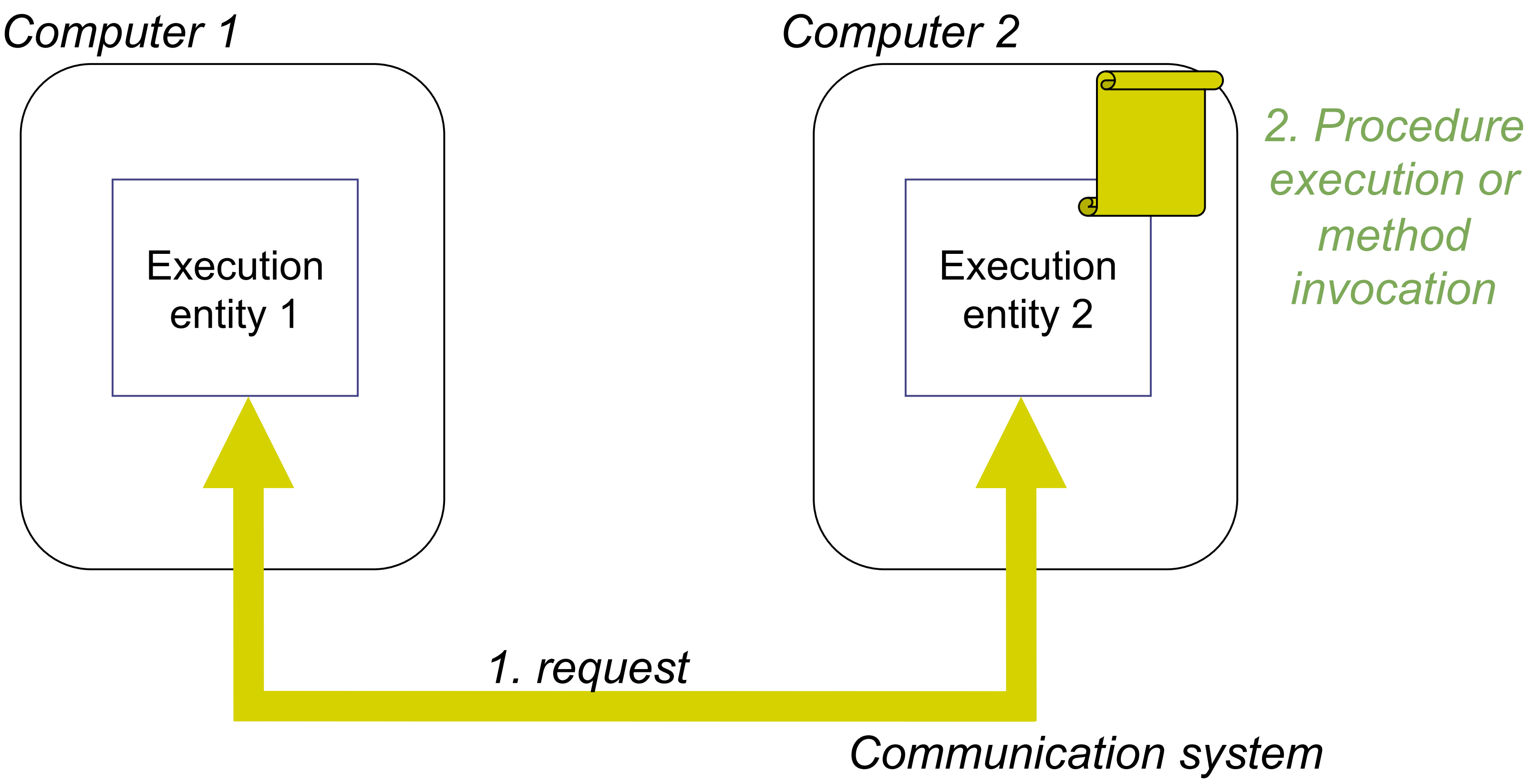
# Distributed programming model

- Remote method invocation



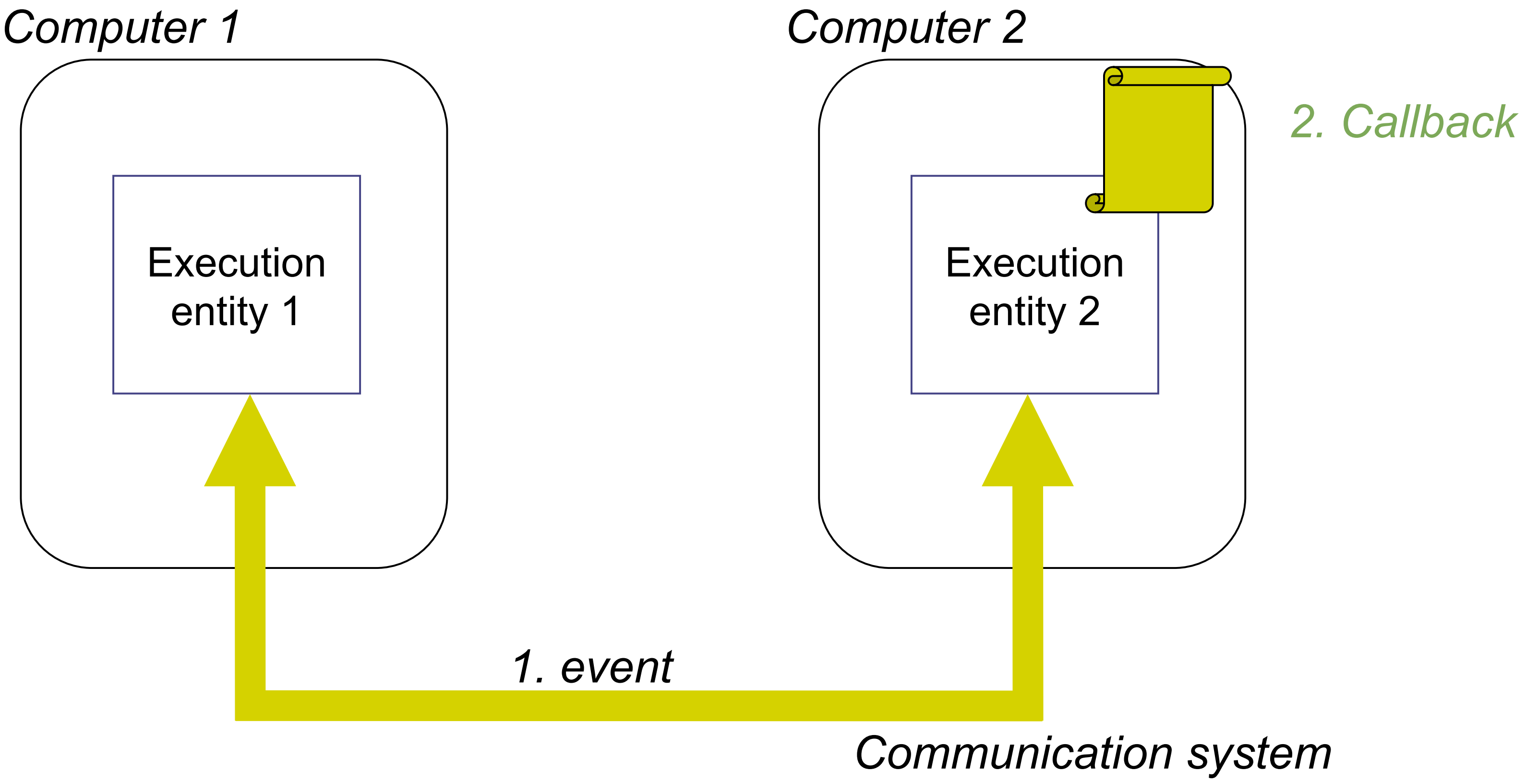
# Distributed programming model

- Non-blocking procedure call or method invocation



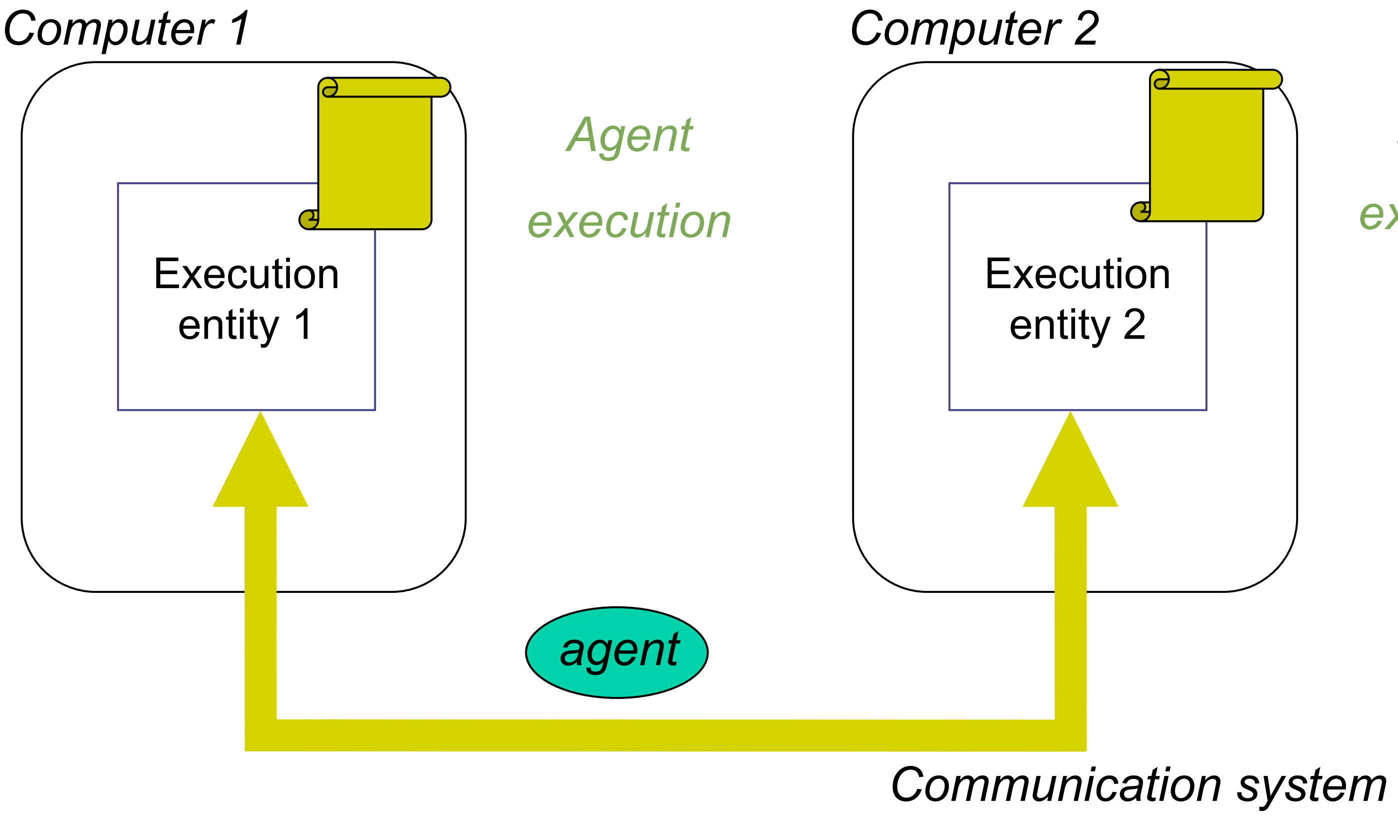
# Distributed programming model

- Event sending



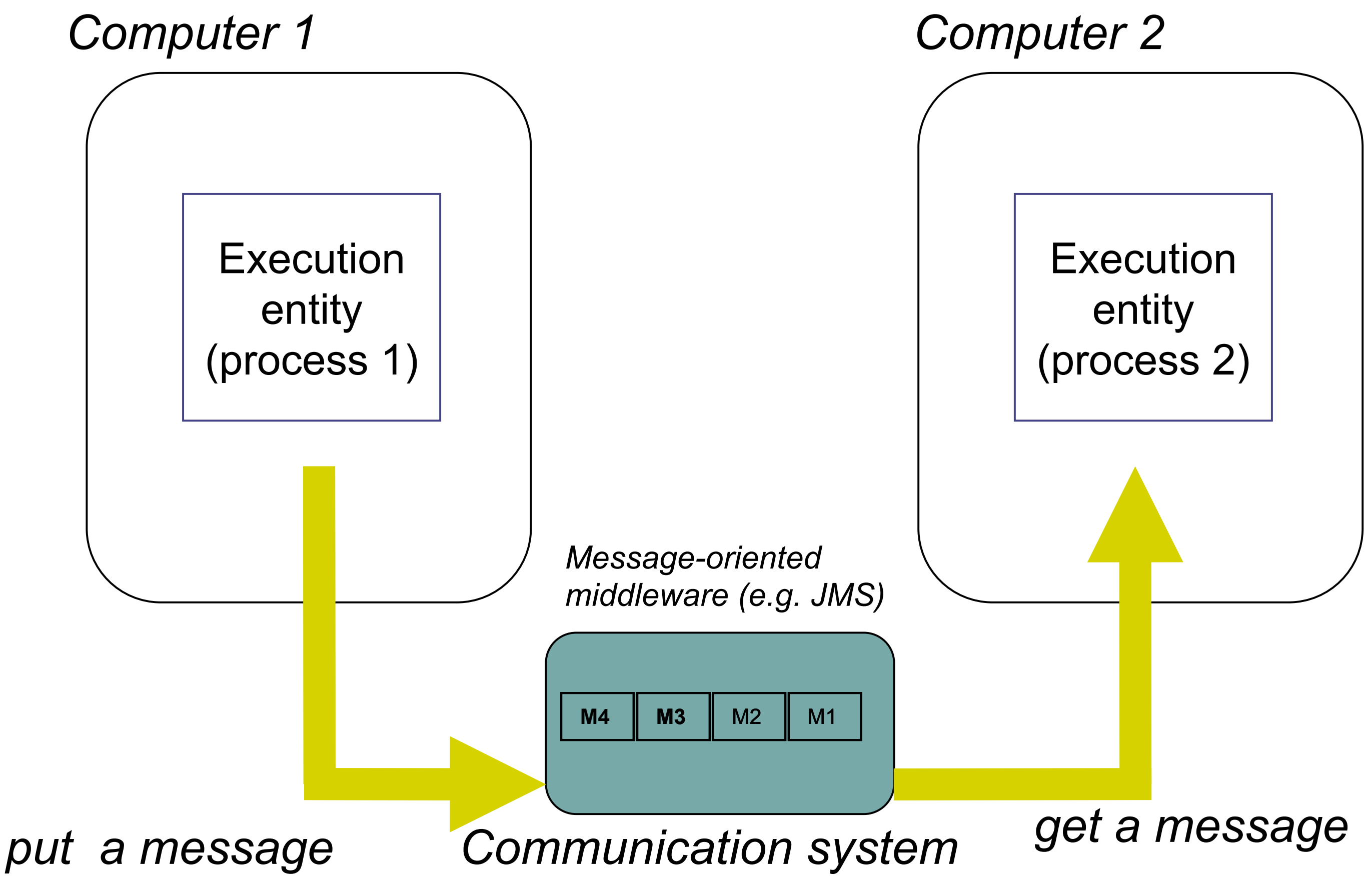
# Distributed programming model

- Agents



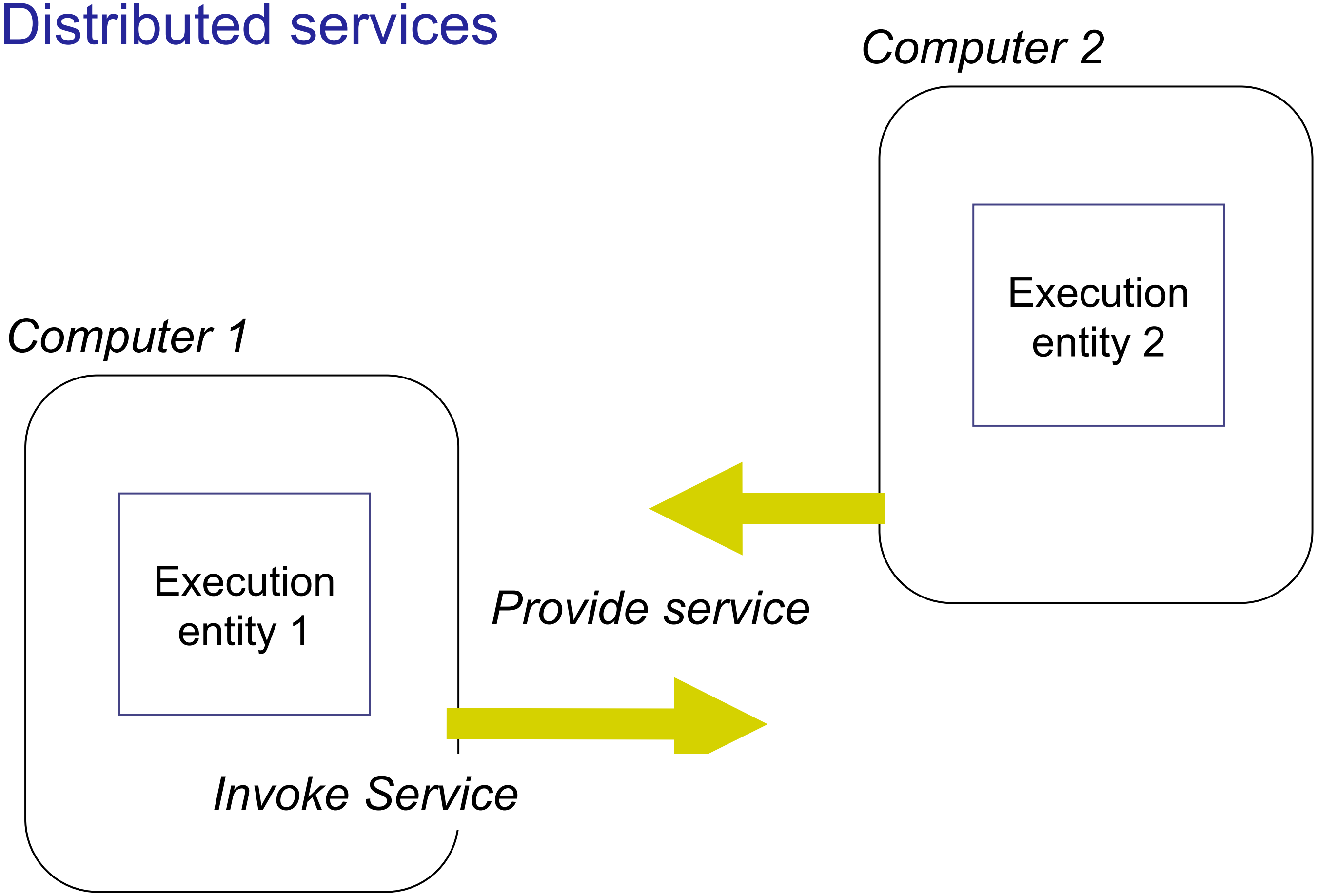
# Distributed programming model

- Message queuing


Computer 1
Computer 2
Execution entity (process 1)
Execution entity (process 2)
Message-oriented middleware (e.g. JMS)
M4
M3
M2
M1
put a message
Communication system
get a message

# Distributed programming model

- Distributed services



# Service-oriented distributed programming

- Base for service-oriented frameworks
  - A client specifies a required service
  - A server specifies a provided service
  - Static and/or dynamic matching between clients and servers

  *A client does not have to identify a server, only a service*

- Service orientation

  **A service is a contractually defined behavior that can be implemented and provided by any component for use by any component, based solely on the contract.**

  Bieber el. al., Service oriented programming, http://www.openwings.org/

# Distributed systems core layers

- Sockets (TCP, UDP)
  - Low-level protocol (based on messages)
  - Allows for application-specific optimizations
  - Data heterogeneity often managed by the application

- RPC (Remote procedure calls)
  - Procedural programming
  - Data heterogeneity managed by the application

- RMI (Remote Method Invocations)
  - Object-oriented programming
  - Data heterogeneity managed by the JVM

# Planning

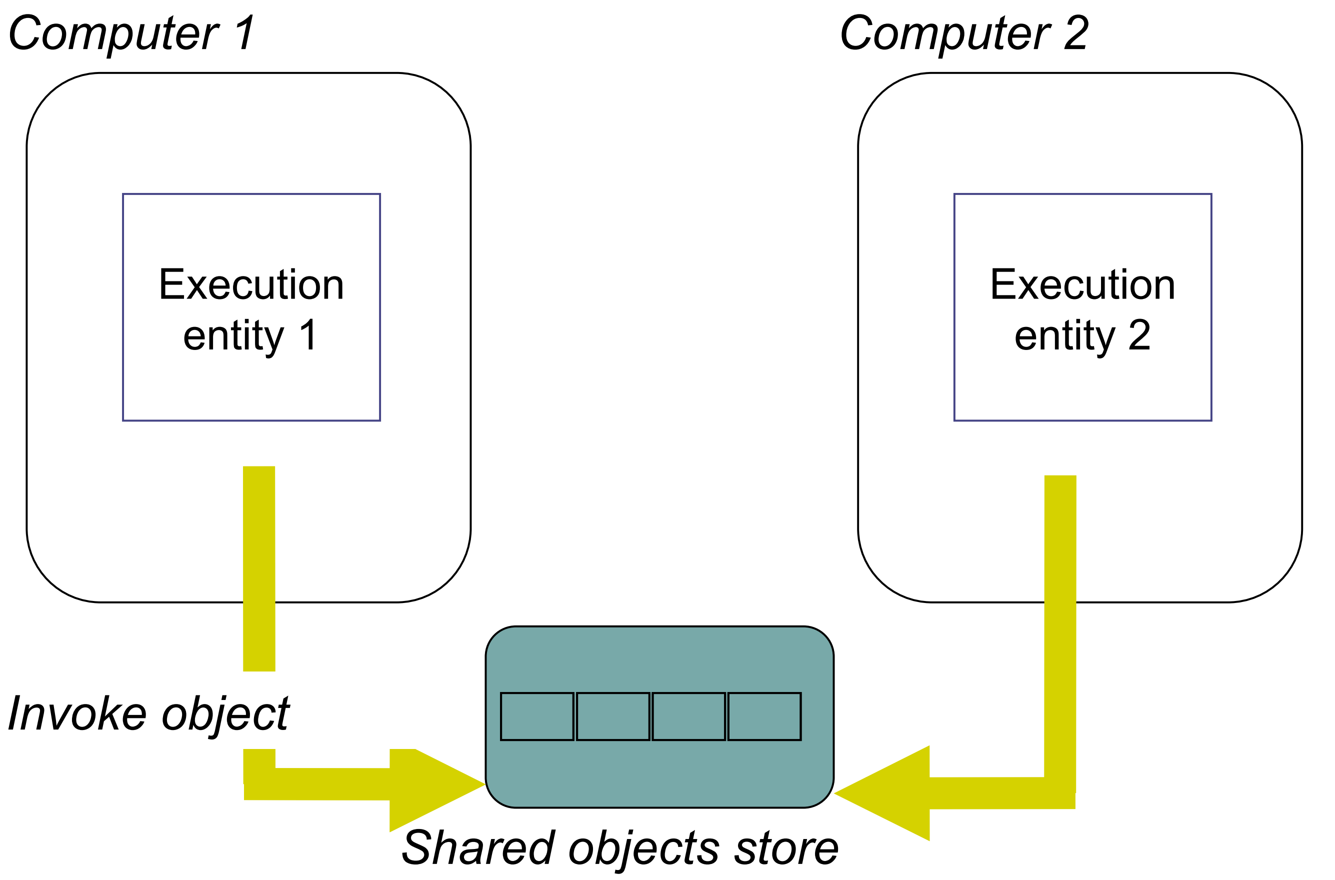
| Séance | Cours | TD / TP |
|---|---|---|
| 1 | Introduction &<br>Message-based distributed systems | TCP Sockets |
| 2 | Asynchronous message-based distributed systems | TCP Sockets |
| 3 | Asynchronous message-based distributed systems | Java NIO |
| 4 | Object-based distributed systems | Java NIO |
| 5 | Object-based distributed systems | Java RMI |
| 6 | Asynchronous object-based distributed systems | Java RMI<br>Distribution projet Mobile agents |
| 7 | .. | Travail projet  Mobile agents |
| 8 | Web protocols | AAA |
| 9 | Service-oriented distributed systems | Servlets |
| 10 | Evaluation projet Mobile agents | Jini |

# References

- Chris Britton, Peter Bye. *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems (2nd Edition)*. Addison-Wesley, 2004.
- George Coulouris, Jean Dollimore, Tim Kindberg. *Distributed Systems: Concepts and Design (4th Edition)*. Addison Wesley, 2005.
- Arno Puder, Kay Römer, Frank Pilhofer. *Distributed Systems Architecture: A Middleware Approach*. Morgan Kaufmann, 2005.
- Andrew S. Tanenbaum, Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall, 2006.

# Distributed programming model

- Shared distributed objects


Computer 1
Computer 2
Execution entity 1
Execution entity 2
Invoke object
Shared objects store