Sistemas Distribuídos

José Orlando Pereira

Departamento de Informática
Universidade do Minho

Distributed system

- Collection of autonomous computing elements
- Single coherent system

Design goals

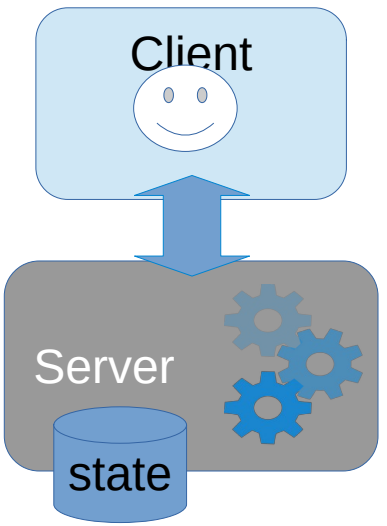
- Share resources: state, function, hardware, ...
- Achieve scale:
  - Numerical (size)
  - Geographical (distance)
  - Administrative
- Provide openness: interoperability between multiple vendors
- Transparency: do not show distribution boundaries

System architectures

- How are distributed components organized
- Centralized architectures:
  - Asymmetric / special roles
  - Planned organization
- Decentralized architectures:
  - Symmetric / equal peers
  - Self-organizing

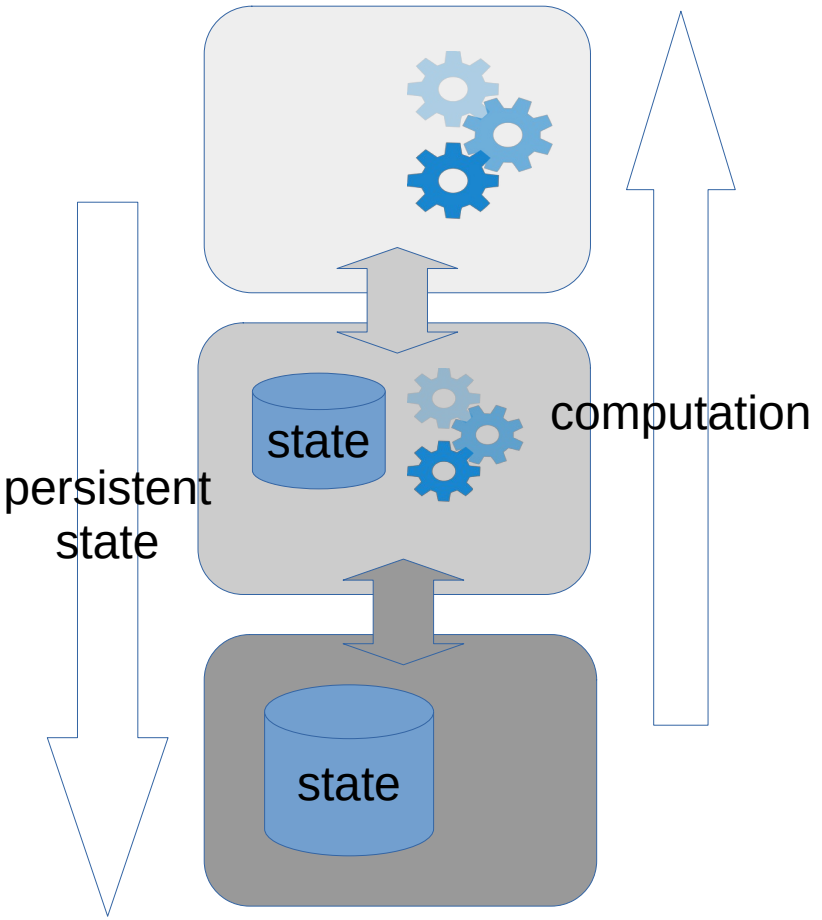
Client-Server

- Server encapsulates resources and function
- Server is a well known centralized entity
- Anonymous clients initiate synchronous interactions
- Example: NFS



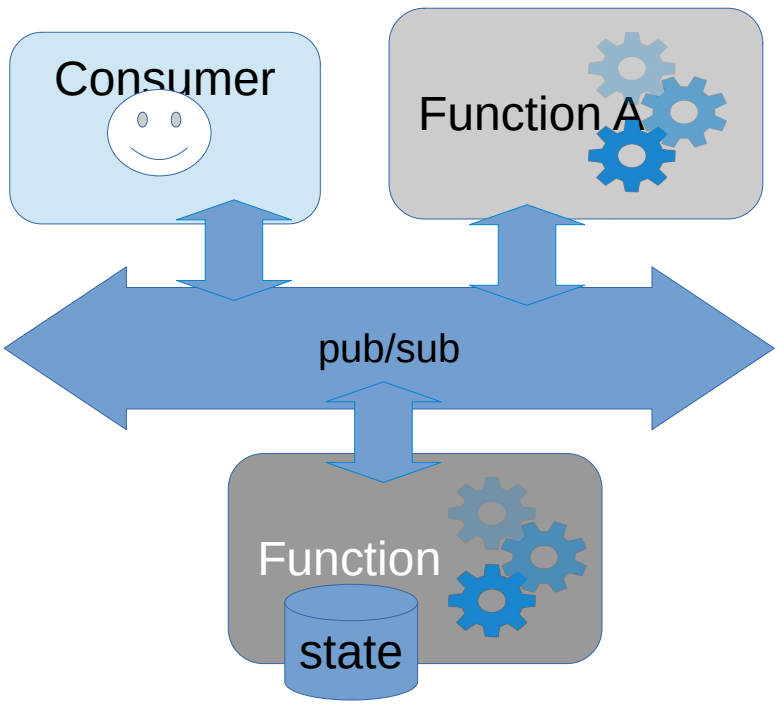
Layered

- Extends client-server
- Standard interfaces and interchangeable layers
- Separation of concerns:
  - Computation
  - Persistence
- Example: 3-tier Web application architectures



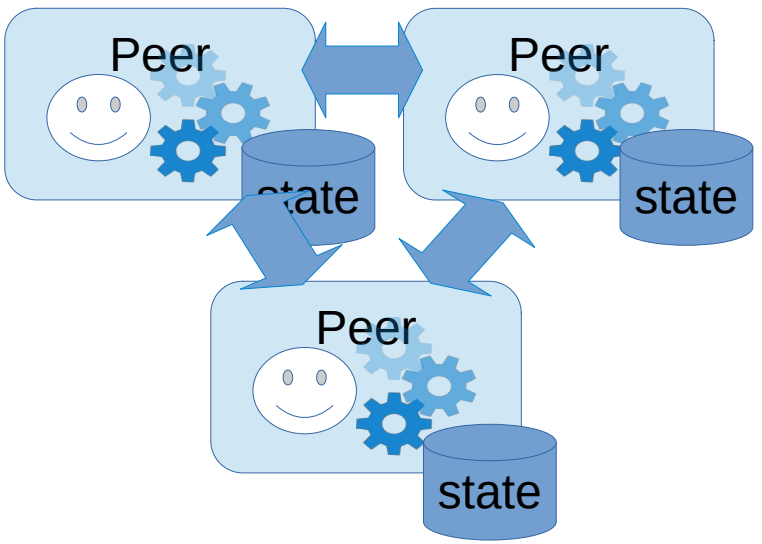
Event-based

- Referential decoupling with publish-subscribe
- Temporal decoupling with with store&forward
- Example: EAI



Peer-to-peer

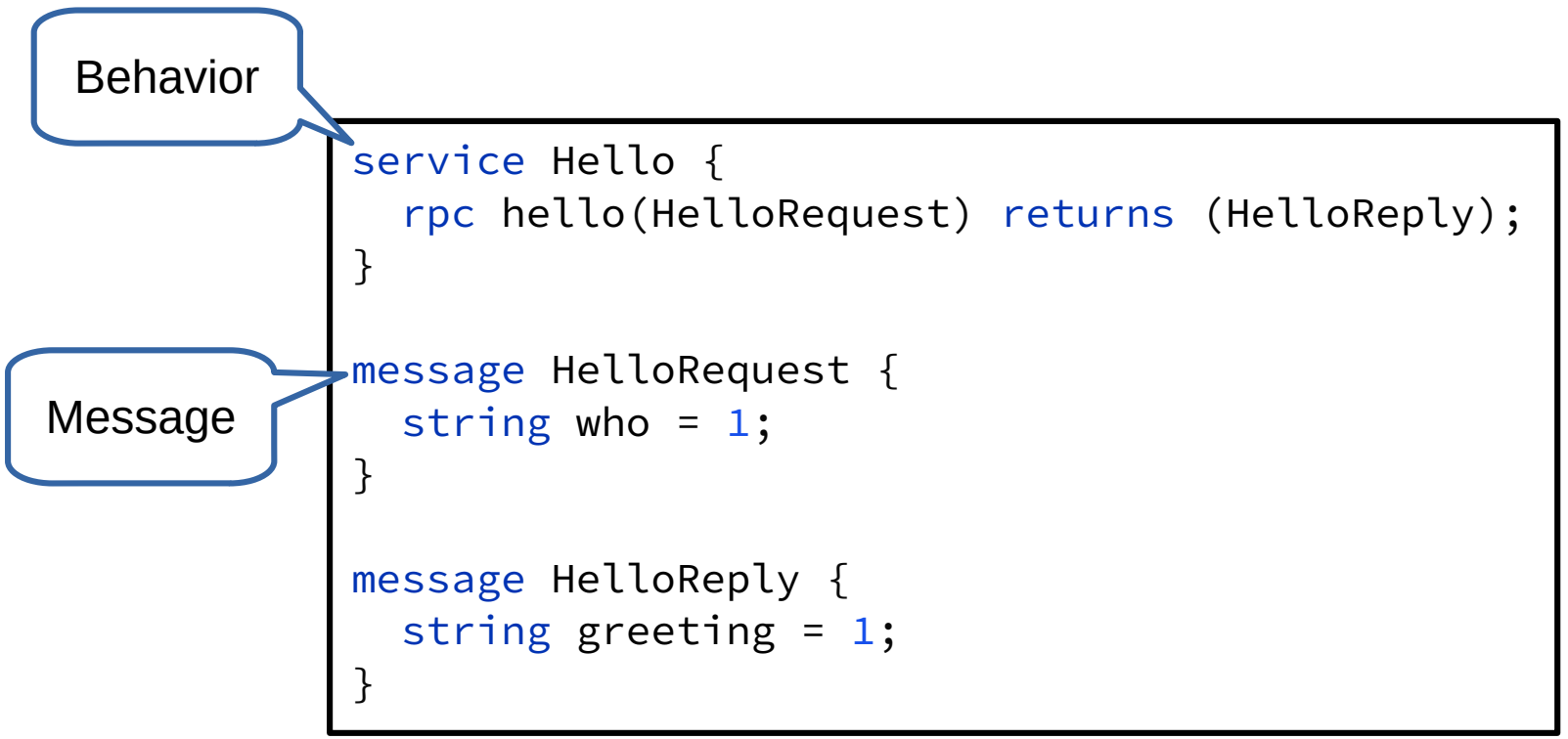
- Equal peers
- Decentralized and self-organizing
  - Overlay network
- Example: BitTorrent



Protocols

- We focus on protocols:
  - What messages are exchanged
  - What behaviors are accepted from each participant
- We resort to:
  - Specialized languages to specify protocols
  - Software tools to implement them
- Example: gRPC for client-server architectures
  https://github.com/grpc/grpc-java

Protobuf language



Protobuf language

- Scalar data types:
  - int32, int64, float, double, bool, string, bytes, …
- Composite data types:
  - optional, repeated, oneof, …
- Notice that:
  - There is no functionality / code!
  - There are no pointers / references!

- Reference documentation: https://protobuf.dev/programming-guides/proto3/

Middleware and functionality

- Middleware provides:
  - an API for clients to make use of the server
  - an API for servers to expose funcionality
- It is up to us to implement client and server functionality

- (Warning: We start by c&p configuration and setup code. Later we implement our own middleware and understand in detail what gRPC is doing.)

Example

- Turn queue system
  - Multiple queues
  - Average waiting time
- Client-server implementation:
  - Shared state and function in the server
  - Client for obtaining a ticket
  - Client for advancing a turn

Summary

- Definition and main goals of distributed systems
- Client-server as the main architecture and protocols as the key concept
- RPC middleware for client-server implementation