Object Interaction: RMI and RPC

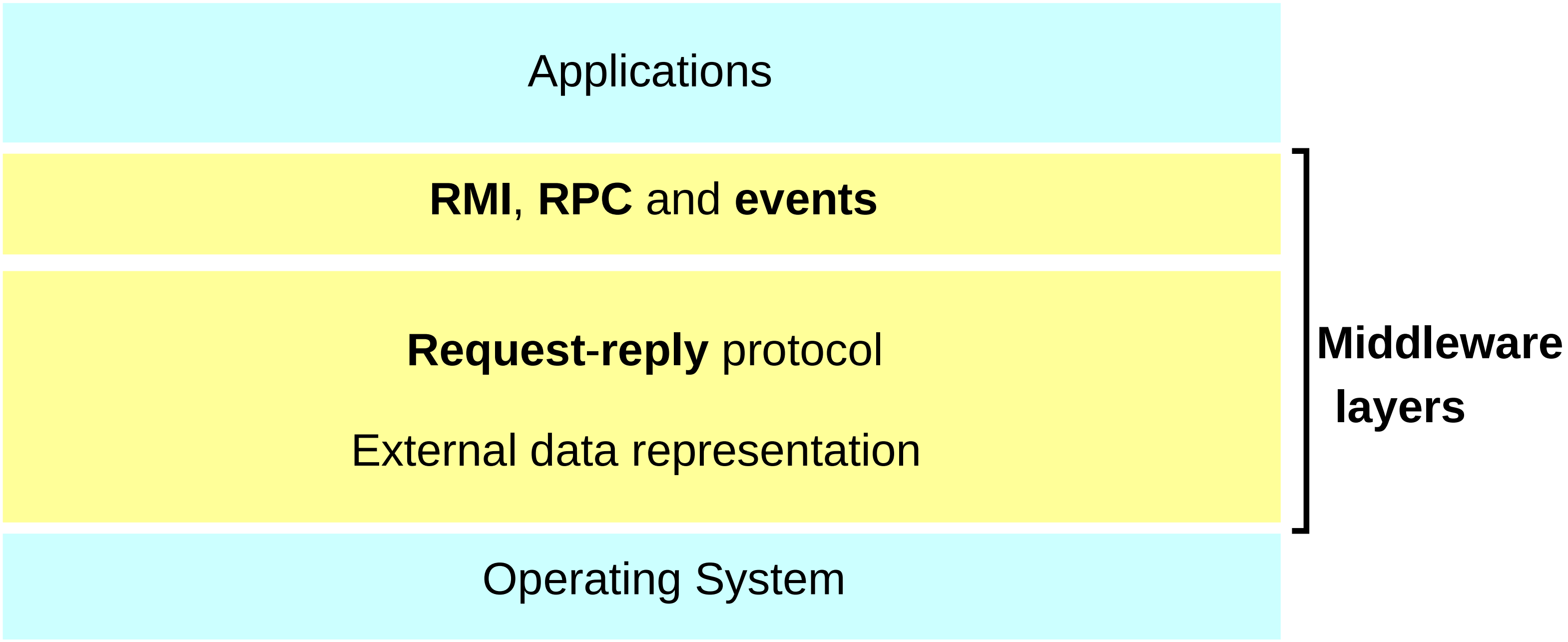
Overview

- Distributed applications programming
  - distributed objects model
  - RMI, invocation semantics
  - RPC
  - events and notifications
- Products
  - Java RMI, CORBA, DCOM
  - Sun RPC
  - Jini

Why Middleware?

- Location transparency
  - client/server need not know their location
- Sits on top of OS, independent of:
  - communication protocols:
    use abstract request-reply protocols over UDP, TCP
  - computer hardware:
    use external data representation e.g. CORBA CDR
  - operating system:
    use e.g. socket abstraction available in most systems
  - programming language:
    e.g. CORBA supports Java, C++

Middleware layer



Applications

RMI, **RPC** and **events**

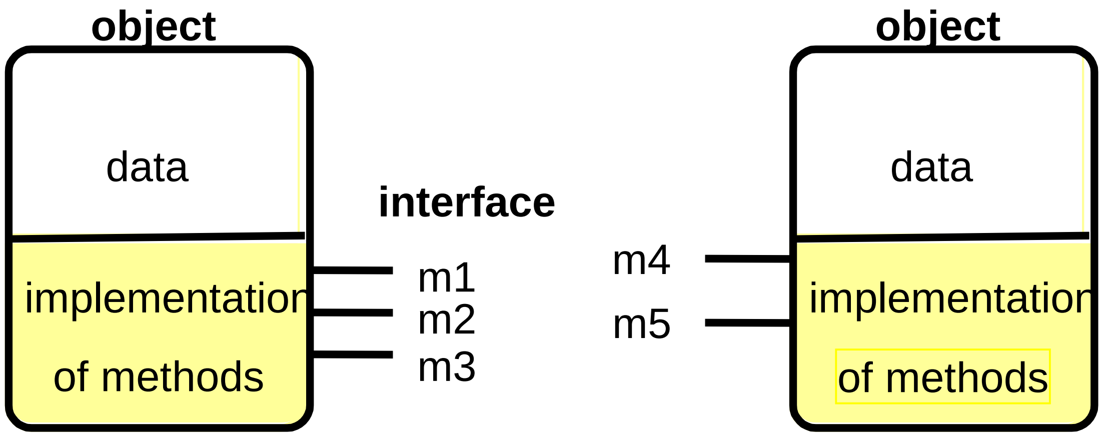
Request-reply protocol

External data representation

Operating System

Middleware layers

Objects

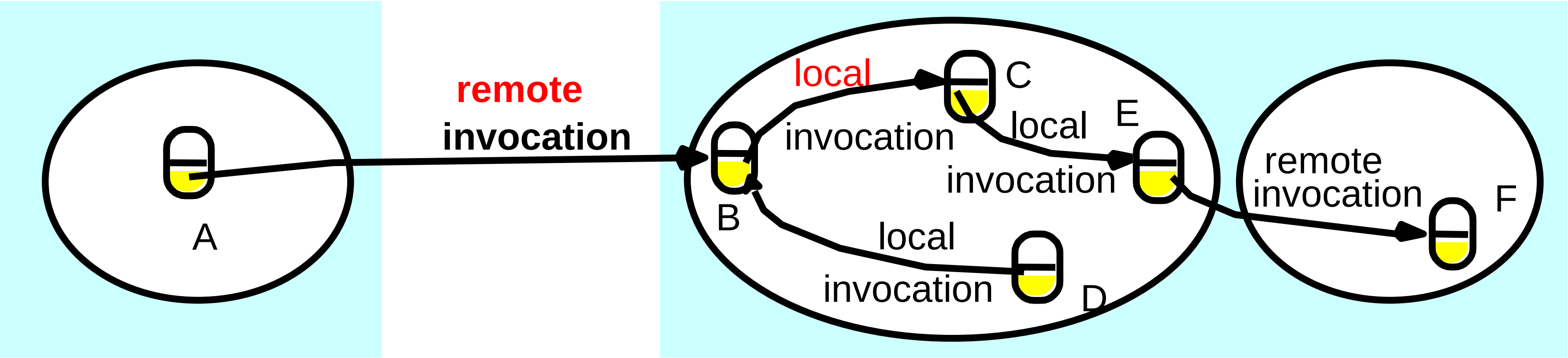


- Objects = data + methods
  - logical and physical nearness
  - first class citizens, can be passed as arguments
- Interact via interfaces:
  - define types of arguments and exceptions of methods

The object model

- Programs logically partitioned into objects
  - distributing objects natural and easy
- Interfaces
  - the only means to access data, make them remote?
- Actions
  - via method invocation
  - interaction, chains of invocations
  - may lead to exceptions, part of interface
- Garbage collection
  - reduced effort, error-free (Java, not C++)

The distributed object model



- Objects distributed (client-server models)
- Extend with
  - Remote object reference
  - Remote interfaces
  - Remote Method Invocation (RMI)

Advantages of distributed objects

- Data encapsulation gives better protection
  - concurrent processes, interference
- Method invocations
  - can be remote or local
- Objects
  - can act as clients, servers, etc
  - can be replicated for fault-tolerance and performance
  - can migrate, be cached for faster access

Remote object reference

- Object references
  - used to access objects which live in processes
  - can be passed as arguments, stored in variables,...
- Remote object references
  - object identifiers in a distributed system
  - must be unique in space and time
  - error returned if accessing a deleted object
  - can allow relocation (see CORBA case study)

Remote object reference

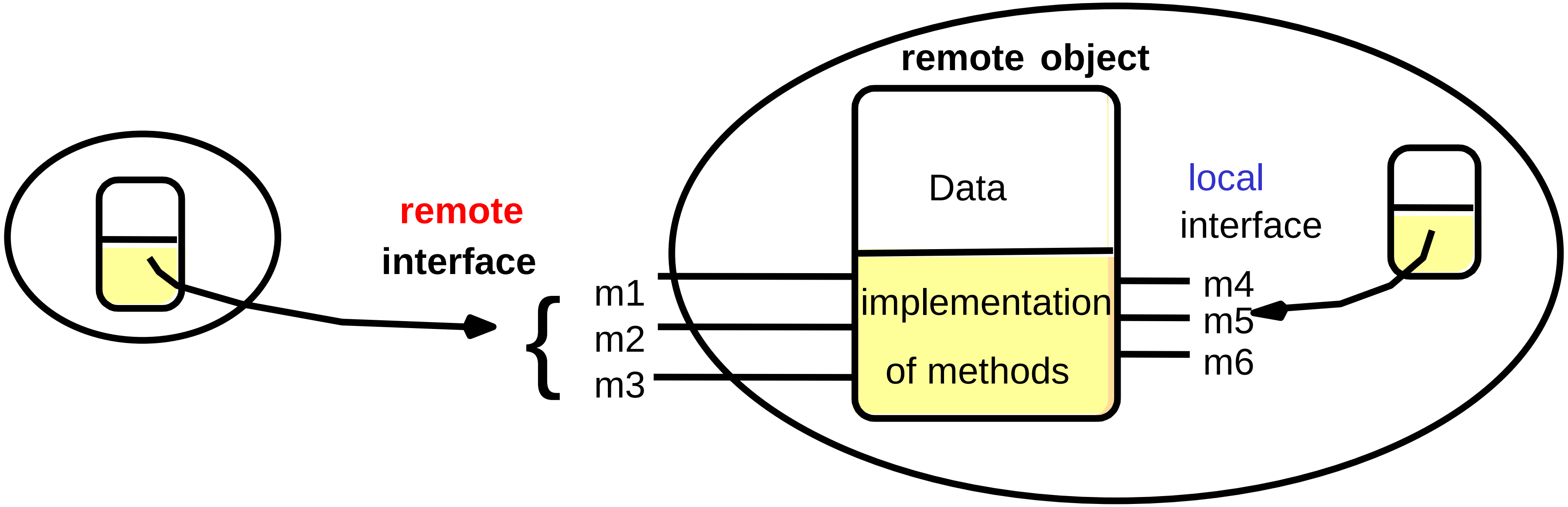
- Constructing unique remote object reference
  - IP address, port, interface name
  - time of creation, local object number (new for each object)
- Use the same as for local object references
- If used as addresses
  - cannot support relocation (alternative in CORBA)

32 bits	*32 bits*	*32 bits*	*32 bits*	
Internet address	port number	time	object number	interface of remote object

Remote interfaces

- Specify externally accessed
  - variables and procedures
  - no direct references to variables (no global memory)
  - local interface separate
- Parameters
  - input, output or both,
  - instead of call by value, call by reference
- No pointers
- No constructors

Remote object and its interfaces



- CORBA: Interface Definition Language (IDL)
- Java RMI: as other interfaces, keyword *Remote*

Handling remote objects

- Exceptions
  - raised in remote invocation
  - clients need to handle exceptions
  - timeouts in case server crashed or too busy
- Garbage collection
  - distributed garbage collection may be necessary
  - combined local and distributed collector
  - cf Java reference counting

RMI issues

- Local invocations
  - executed exactly once
- Remote invocations
  - via Request-Reply (see *DoOperation*)
  - may suffer from communication failures!
    - retransmission of request/reply
    - message duplication, duplication filtering
  - no unique semantics…

Invocation semantics summary

Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	***Maybe***
Yes	No	Re-execute procedure	***At-least-once***
Yes	Yes	Retransmit reply	***At-most-once***

Re-executing a method sometimes dangerous...

Maybe invocation

- Remote method
  - may execute or not at all, invoker cannot tell
  - useful only if occasional failures
- Invocation message lost...
  - method not executed
- Result not received...
  - was method executed or not?
- Server crash...
  - before or after method executed?
  - if timeout, result could be received after timeout...

At-least-once invocation

- Remote method
  - invoker receives result (executed exactly) or exception (no result, executed once or not at all)
  - retransmission of request messages
- Invocation message retransmitted...
  - method may be executed more than once
  - arbitrary failure (wrong result possible)
  - method must be idempotent (repeated execution has the same effect as a single execution)
- Server crash...
  - dealt with by timeouts, exceptions

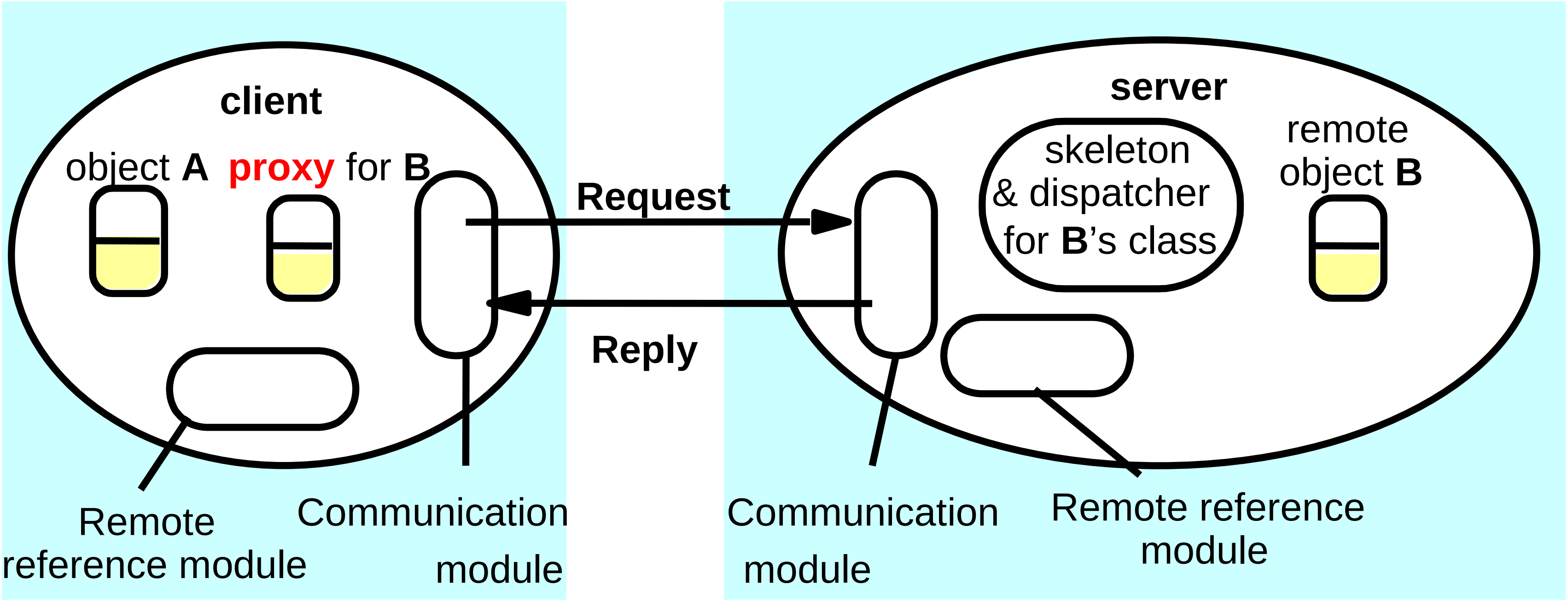
At-most-once invocation

- Remote method
  - invoker receives result (executed once) or exception (no result)
  - retransmission of reply & request messages
  - duplicate filtering
- Best fault-tolerance...
  - arbitrary failures prevented if method called at most once
- Used by CORBA and Java RMI

Transparency of RMI

- Should remote method invocation be same as local?
  - same syntax, see Java RMI (keyword *Remote*)
  - need to hide
    - data marshalling
    - IPC calls
    - locating/contacting remote objects
- Problems
  - different RMI semantics? susceptibility to failures?
  - protection against interference in concurrent scenario?
- Approaches (Java RMI)
  - transparent, but express differences in interfaces
  - provide recovery features

Implementation of RMI



Object A invokes a method in a remote object B:
communication module, remote reference module, RMI software.

Communication modules

- Reside in client and server
- Carry out Request-Reply jointly
  - use unique message ids (new integer for each message)
  - implement given RMI semantics
- Server’s communication module
  - selects dispatcher within RMI software
  - converts remote object reference to local

Remote reference module

- Creates remote object references and proxies
- Translates remote to local references (object table):
  - correspondence between remote and local object references (proxies)
- Directs requests to proxy (if exists)
- Called by RMI software
  - when marshalling/unmarshalling

RMI software architecture

- Proxy
  - behaves like local object to client
  - forwards requests to remote object
- Dispatcher
  - receives request
  - selects method and passes on request to skeleton
- Skeleton
  - implements methods in remote interface
    - unmarshals data, invokes remote object
    - waits for result, marshals it and returns reply

Binding and activation

- The binder
  - – mapping from textual names to remote references
  - – used by clients as a look-up service (cf Java RMIregistry)
- Activation
  - – objects active (available for running) and passive (=implementation of methods + marshalled state)
  - – activation = create new instance of class + initialise from stored state
- Activator
  - – records location of passive and active objects
  - – starts server processes and activates objects within them

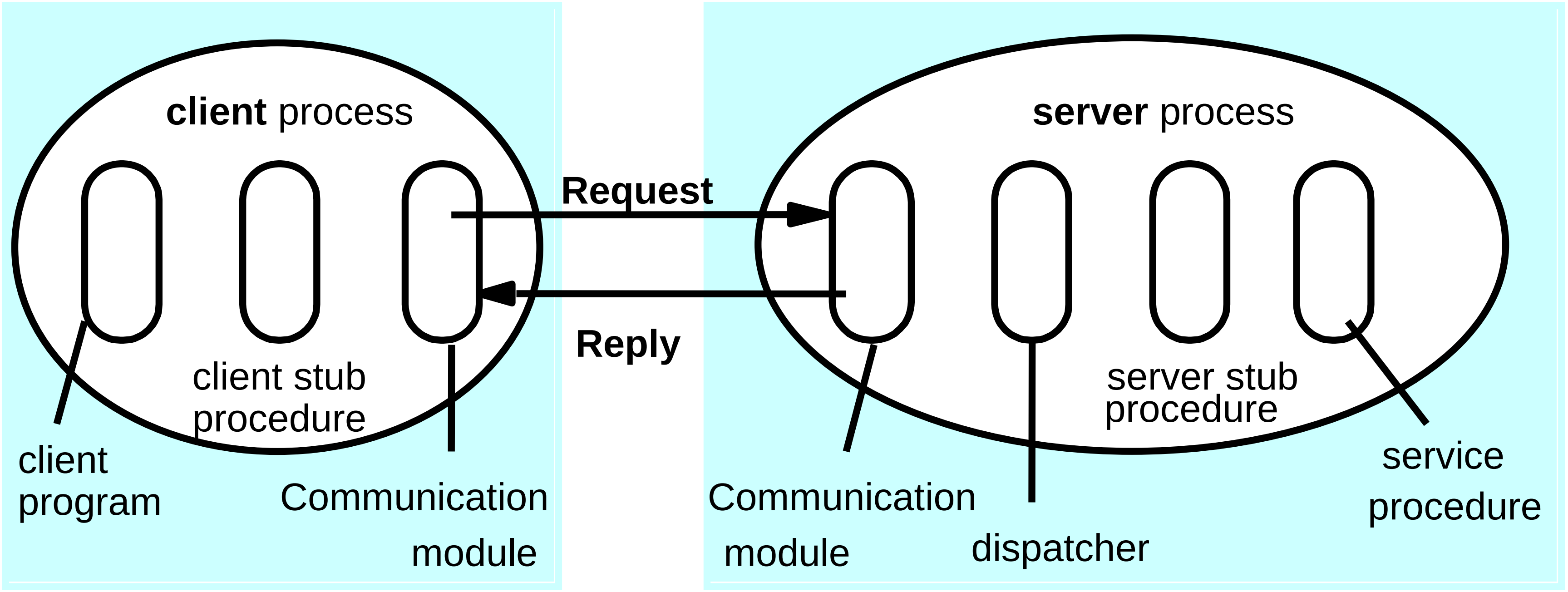
Object location issues

- Persistent object stores
  - stored on disk, state in marshalled form
  - readily available
  - cf Persistent Java
- Object migration
  - need to use remote object reference and address
- Location service
  - assists in locating objects
  - maps remote object references to probable locations

Remote Procedure Call (RPC)

- RPC
  - historically first, now little used
  - over Request-Reply protocol
  - usually at-least-once or at-most-once semantics
  - can be seen as a restricted form of RMI
  - cf Sun RPC
- RPC software architecture
  - similar to RMI (communication, dispatcher and stub in place of proxy/skeleton)

RPC client and server

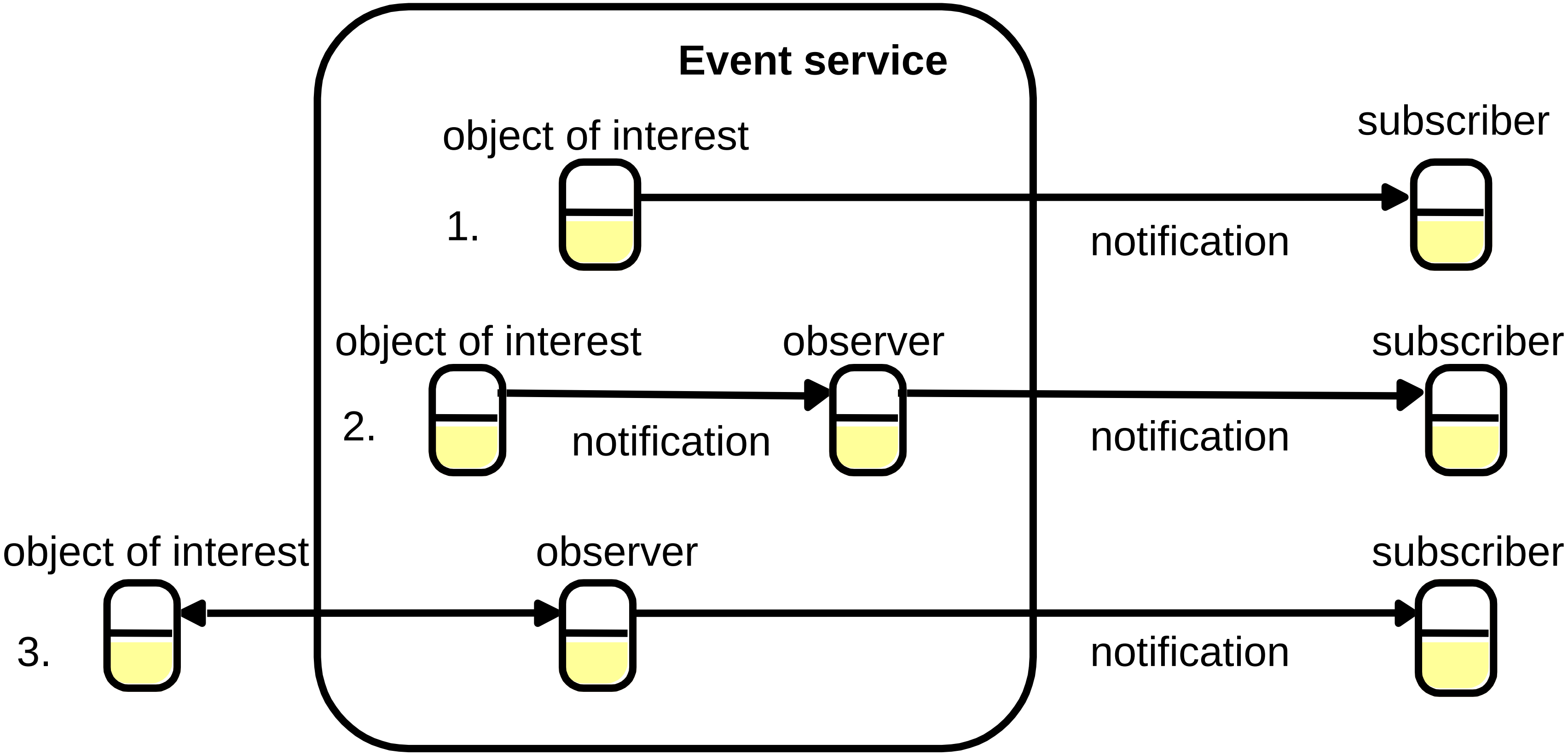


Implemented over Request-Reply protocol.

Event notification

- Distributed event-based systems (cf Jini)
  - object of interest, several interested parties
  - for heterogeneous systems
  - asynchronous model
- Based on Publish-Subscribe paradigm
  - publish type of event
  - subscribe to event notification
  - various delivery semantics (multicast, etc)
- Applications
  - financial information systems
  - real-time systems (hospital monitoring, powerstation)

Architecture for event notification



Summary

- Distributed object model
  - capabilities for handling remote objects (remote references, etc)
  - RMI: maybe, at-least-once, at-most-once semantics
  - RMI implementation, software architecture
- Other distributed programming paradigms
  - RPC, restricted form of RMI, less often used
  - event notification (for heterogeneous, asynchronous systems)