TDTS04/TDDE35: Distributed Systems

Instructor: Niklas Carlsson

Email: niklas.carlsson@liu.se

Notes derived from "Distributed Systems: Principles and Paradigms", by Andrew S. Tanenbaum and Maarten Van Steen, Pearson Int. Ed.

The slides are adapted and modified based on slides used by other instructors, including slides used in previous years by Juha Takkinen, as well as slides used by various colleagues from the distributed systems and networks research community.

Communication in distributed systems

- How do distributed components talk to each other?
  - "Distributed" processes located on different machines
  - Need communication mechanisms
- **Goal:** Hide distributed nature as far as possible

Communication in distributed systems

- Networking primitives and protocols (e.g., TCP/IP)
- Advanced communication models: Built on networking primitives
  - Messages
  - Streams
  - Remote Procedure Calls (RPC)
  - Remote Method Invocation (RMI)

Transport Layer

Important

The transport layer provides the actual communication facilities for most distributed systems.

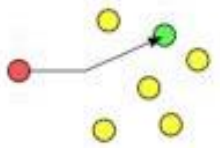
Standard Internet protocols

- TCP: connection-oriented, reliable, stream-oriented communication
- UDP: unreliable (best-effort) datagram communication

Example paradigms

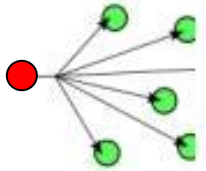
- Unicast
  - One to one

e.g. downloading file from webserver



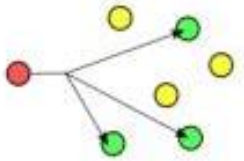
- Broadcast
  - One to many (everyone)

e.g. radio transmission



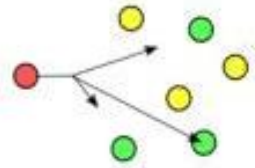
- Multicast
  - One to many (groups)

e.g. streaming video, IRC



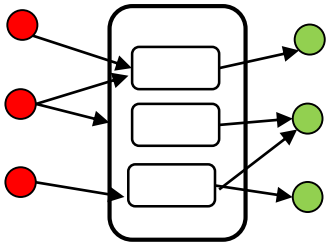
- Anycast
  - One to one of many

e.g. server selection



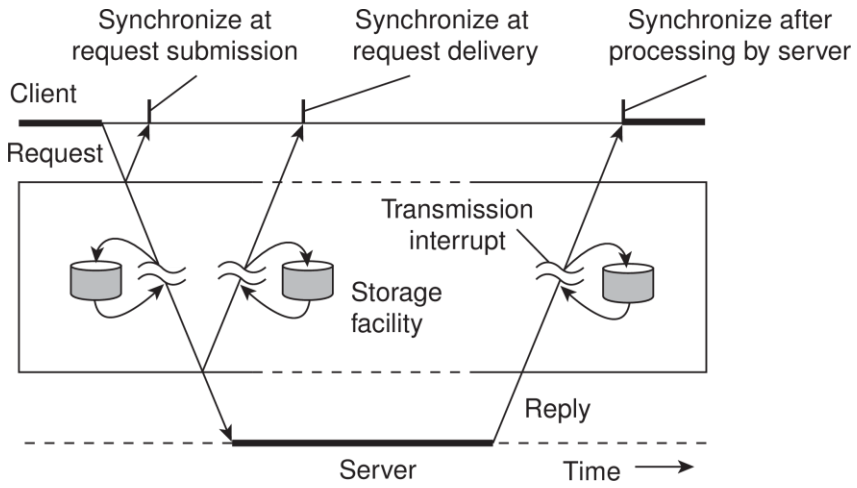
Publish subscribe
- Many to many

e.g., RSS and messaging



Types of communication

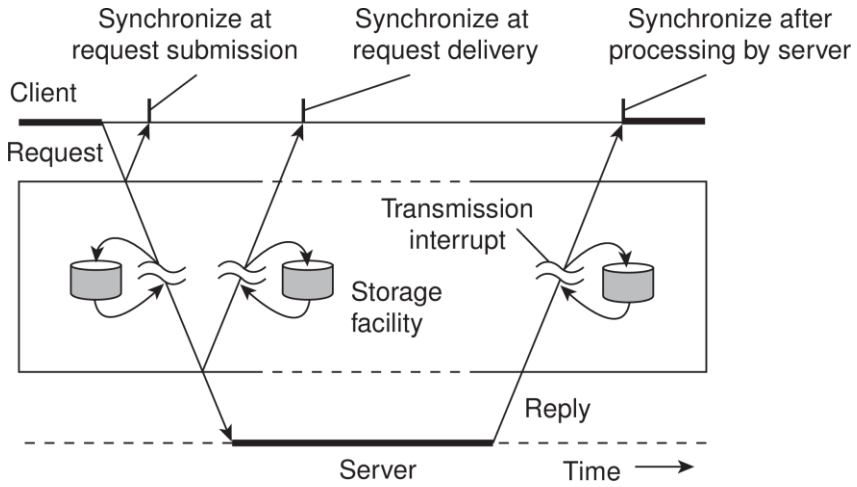
Distinguish...



- Transient versus persistent communication
- Asynchronous versus synchronous communication

Types of communication

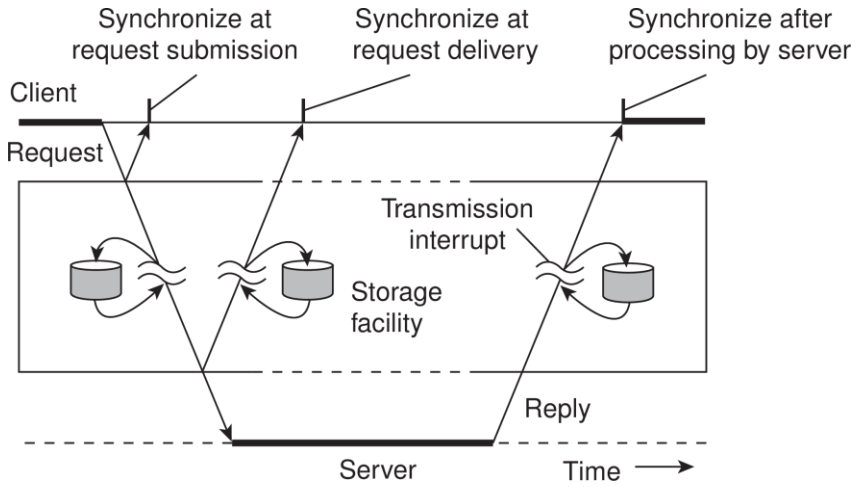
Transient versus persistent



- Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.
- Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

Types of communication

Places for synchronization



- At request submission
- At request delivery
- After request processing

Client/Server

Some observations

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at the time of communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

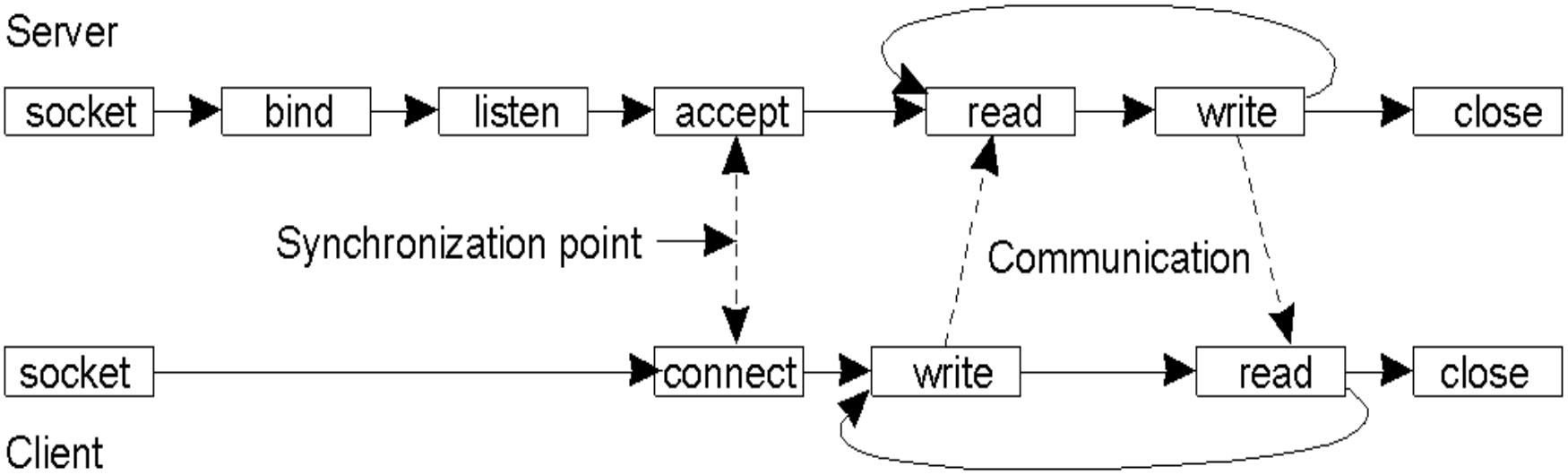
Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

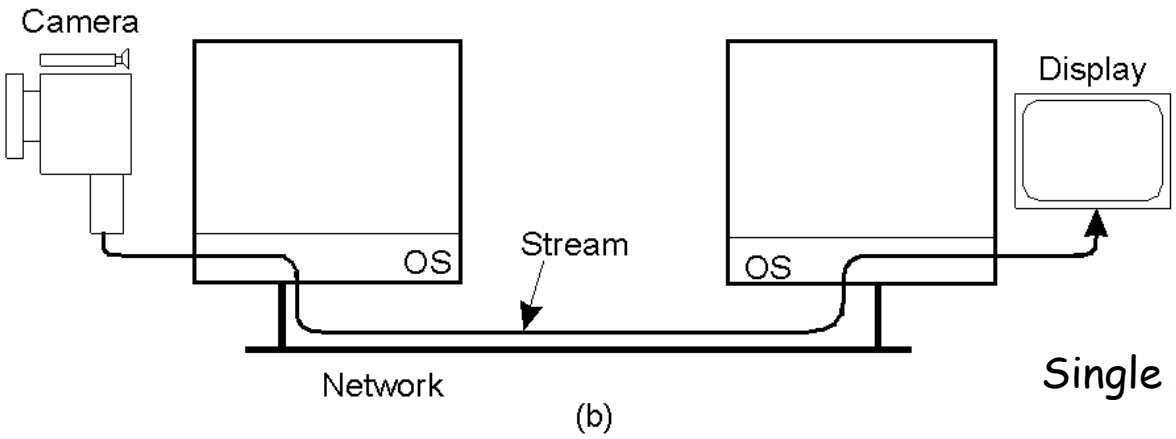
Message-oriented Transient Communication

Many distributed systems built on top of simple message-oriented model

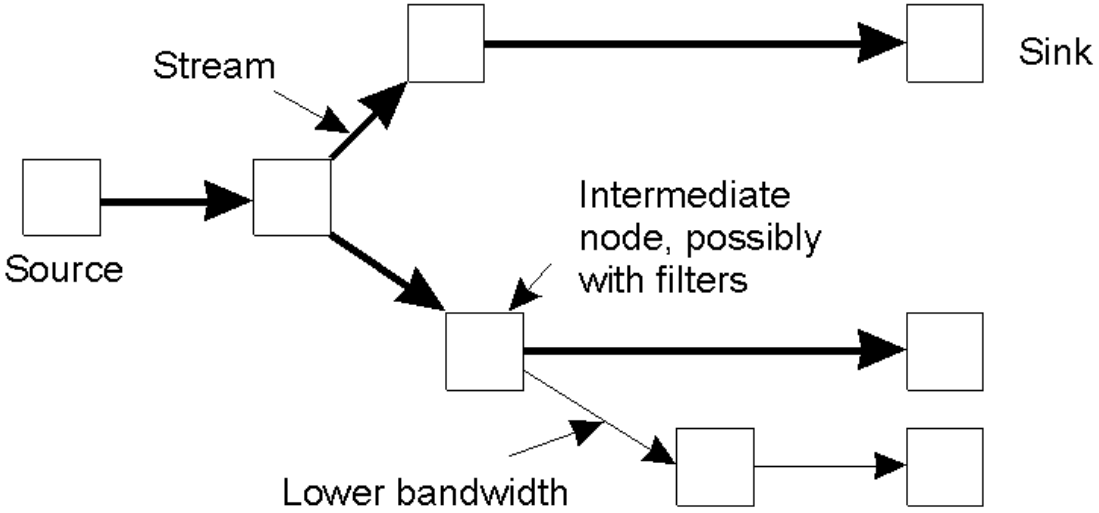
- Example: Berkeley sockets



Stream examples



Single sender and receiver



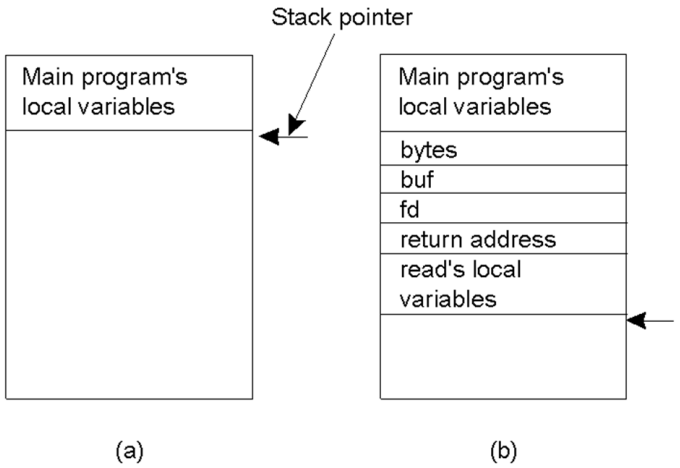
One sender
Multiple receivers

Remote procedure calls (RPC)

- **Goal:** Make distributed computation look like centralized computation
- **Idea:** Allow processes to call procedures on other machines
  - Make it appear like normal procedure calls

a) Parameter passing in a local procedure call: the stack before the call to read

b) The stack while the called procedure is active



Parameter Passing

Local procedure parameter passing

- Call-by-value
- Call-by-reference: arrays, complex data structures

Remote procedure calls simulate this through:

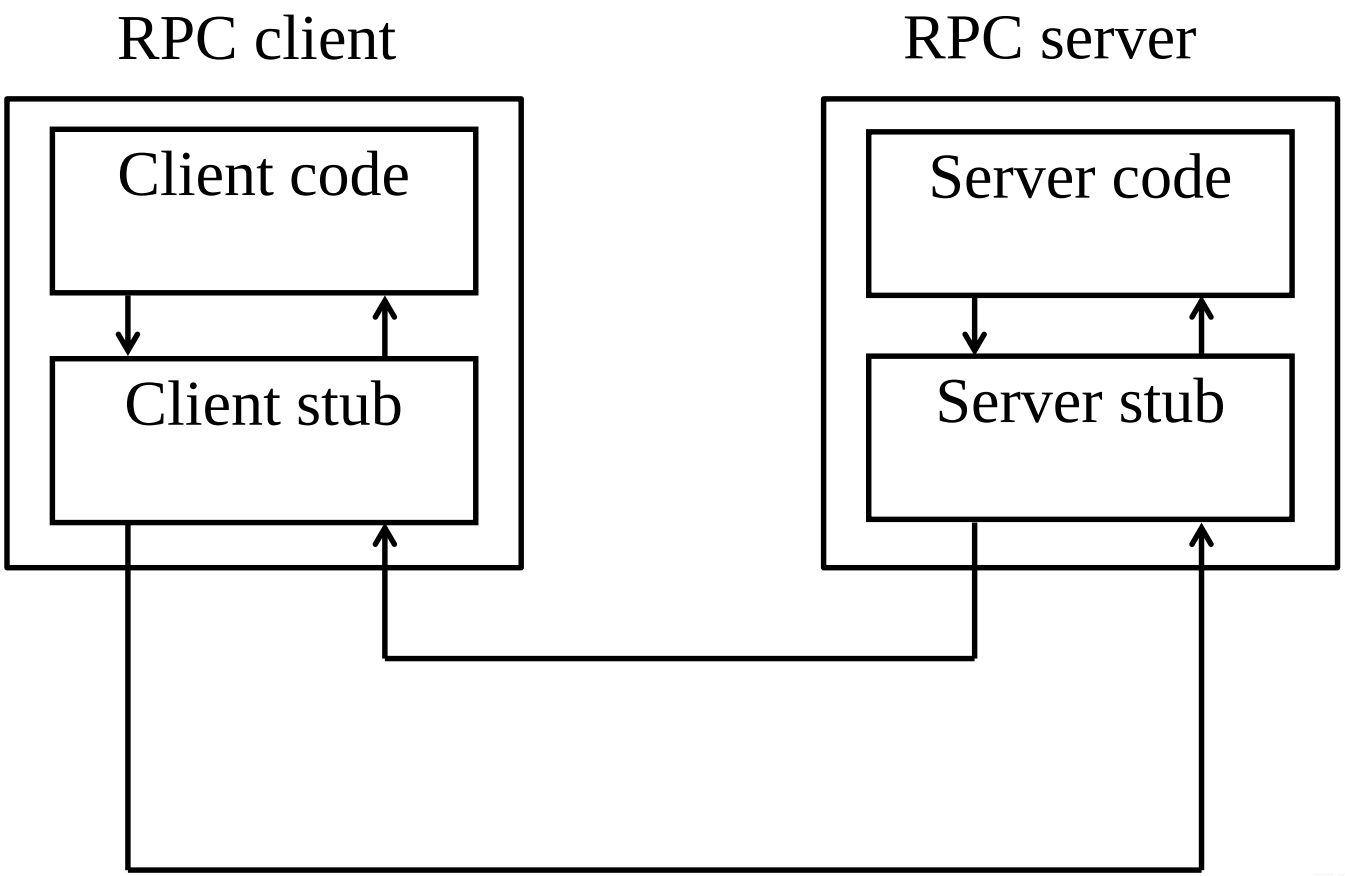
- Stubs – proxies
- Flattening – marshalling

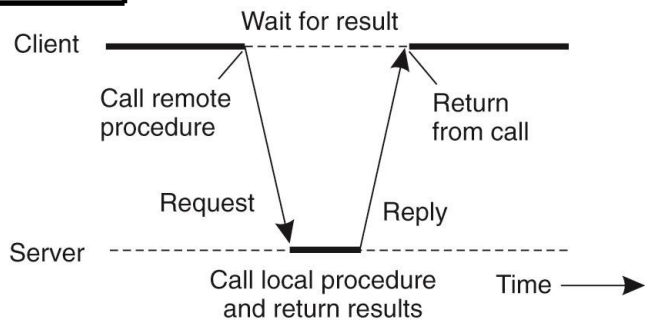
Related issue: global variables are not allowed in RPCs

RPC operation

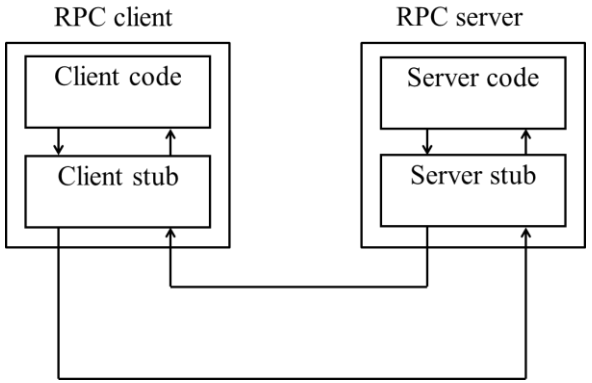
- Challenges:
  - Hide details of communication
  - Pass parameters transparently
- Stubs
  - Hide communication details
  - Client and server stubs
- Marshalling
  - Flattening and parameter passing

RPC operation





Stubs



- Code that communicates with the remote side
- Client stub:
  - Converts function call to remote communication
  - Passes parameters to server machine
  - Receives results
- Server stub:
  - Receives parameters and request from client
  - Calls the desired server function
  - Returns results to client

Passing value parameters

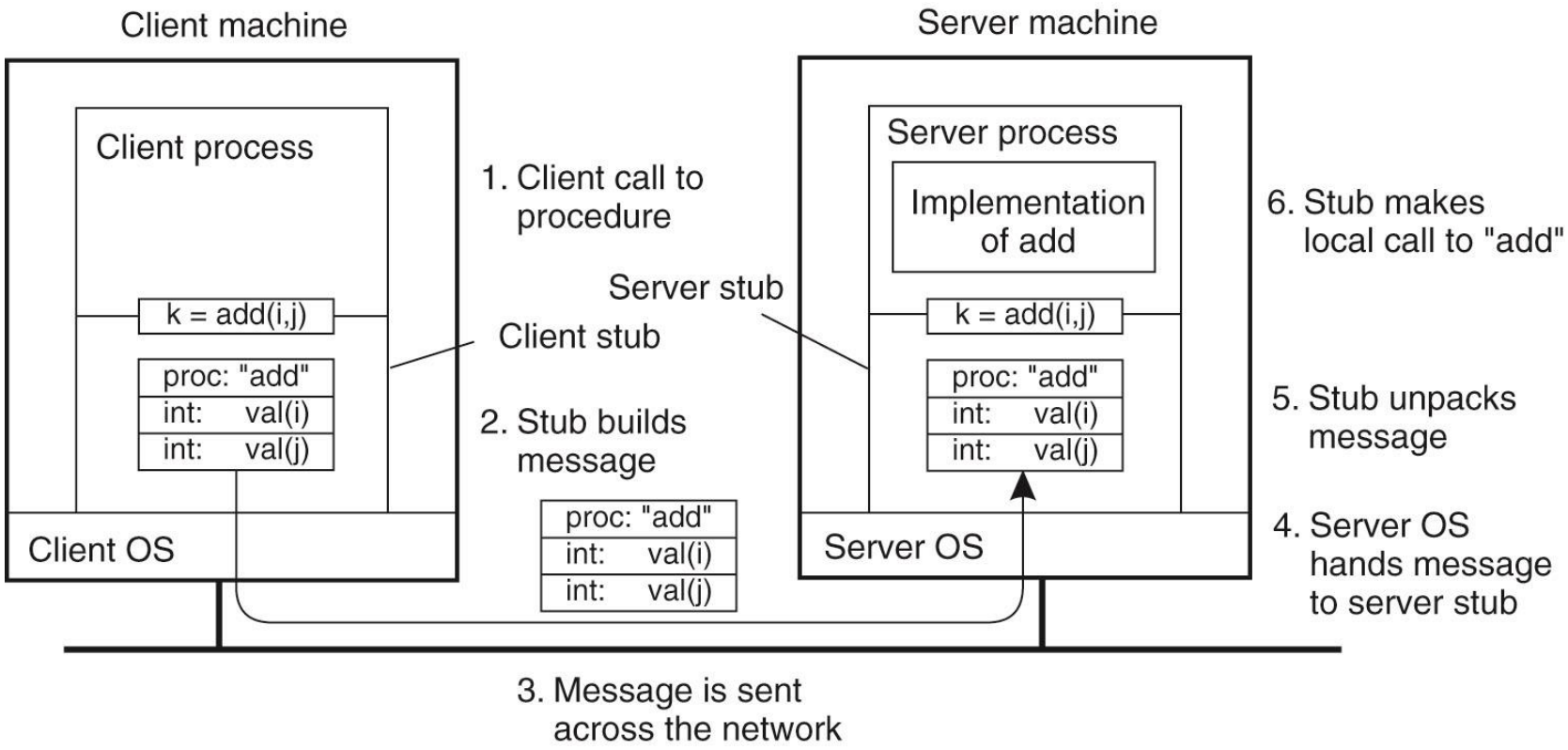


Figure 4-7. The steps involved in a doing a remote computation through RPC.

Stub generation

- Most stubs are similar in functionality
  - Handle communication and marshalling
  - Differences are in the main server-client code
- Application needs to know only stub interface
- Interface Definition Language (IDL)
  - Allows interface specification
  - IDL compiler generates the stubs automatically

Writing a Client and a Server

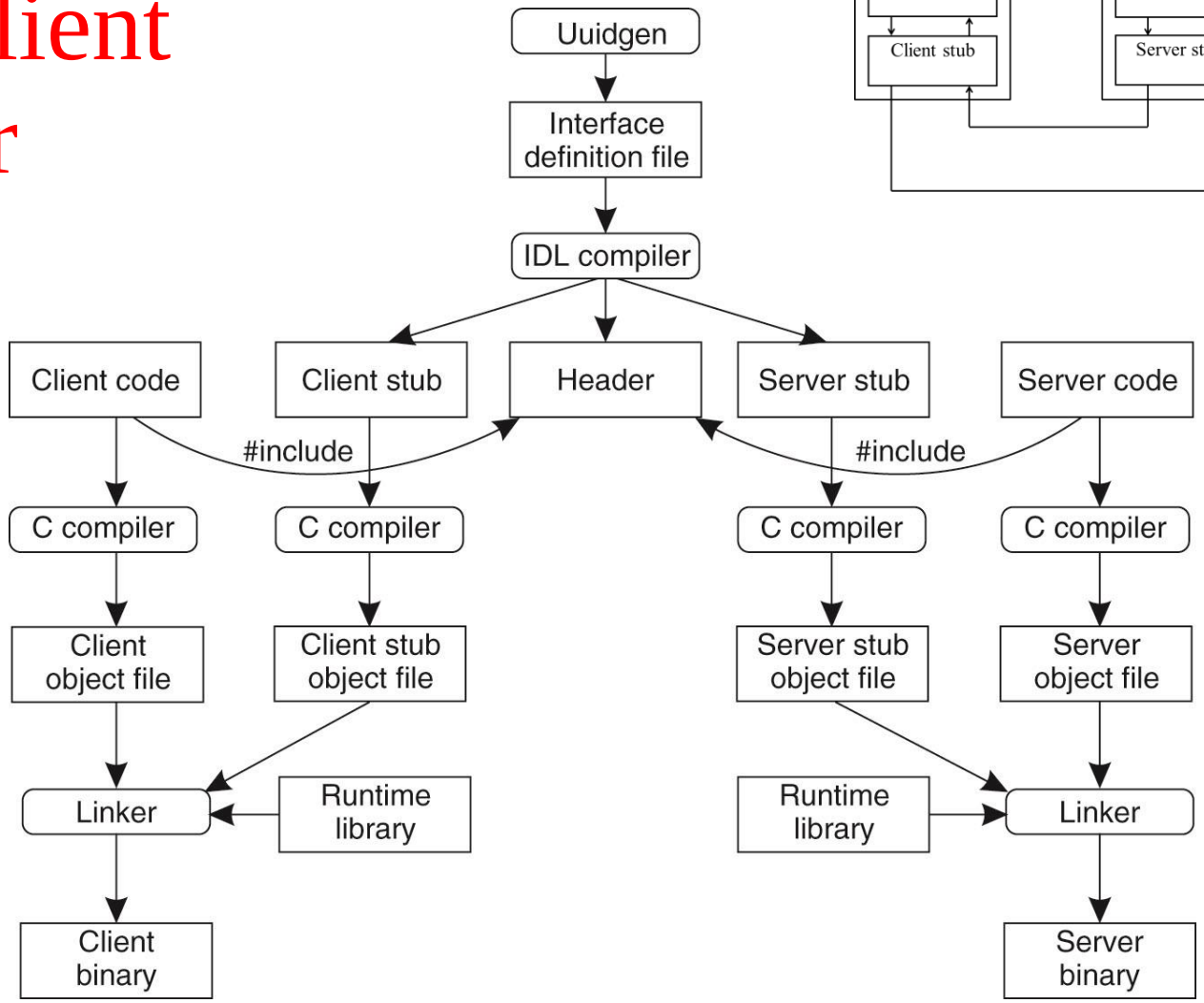


Figure 4-12. The steps in writing a client and a server in DCE RPC.

Asynchronous RPC

- Basic RPC
  - Client blocks until results come back

- What if client wants to do something else?
- What if things fail?

Asynchronous RPC

- Basic RPC
  - Client blocks until results come back
- Asynchronous RPC
  - Server sends ACK as soon as request is received
  - Executes procedure later
- Deferred synchronous RPC
  - Use two asynchronous RPCs
  - Server sends reply via second asynchronous RPC
- One-way RPC
  - Client does not even wait for an ACK from the server

Client and Server Stubs

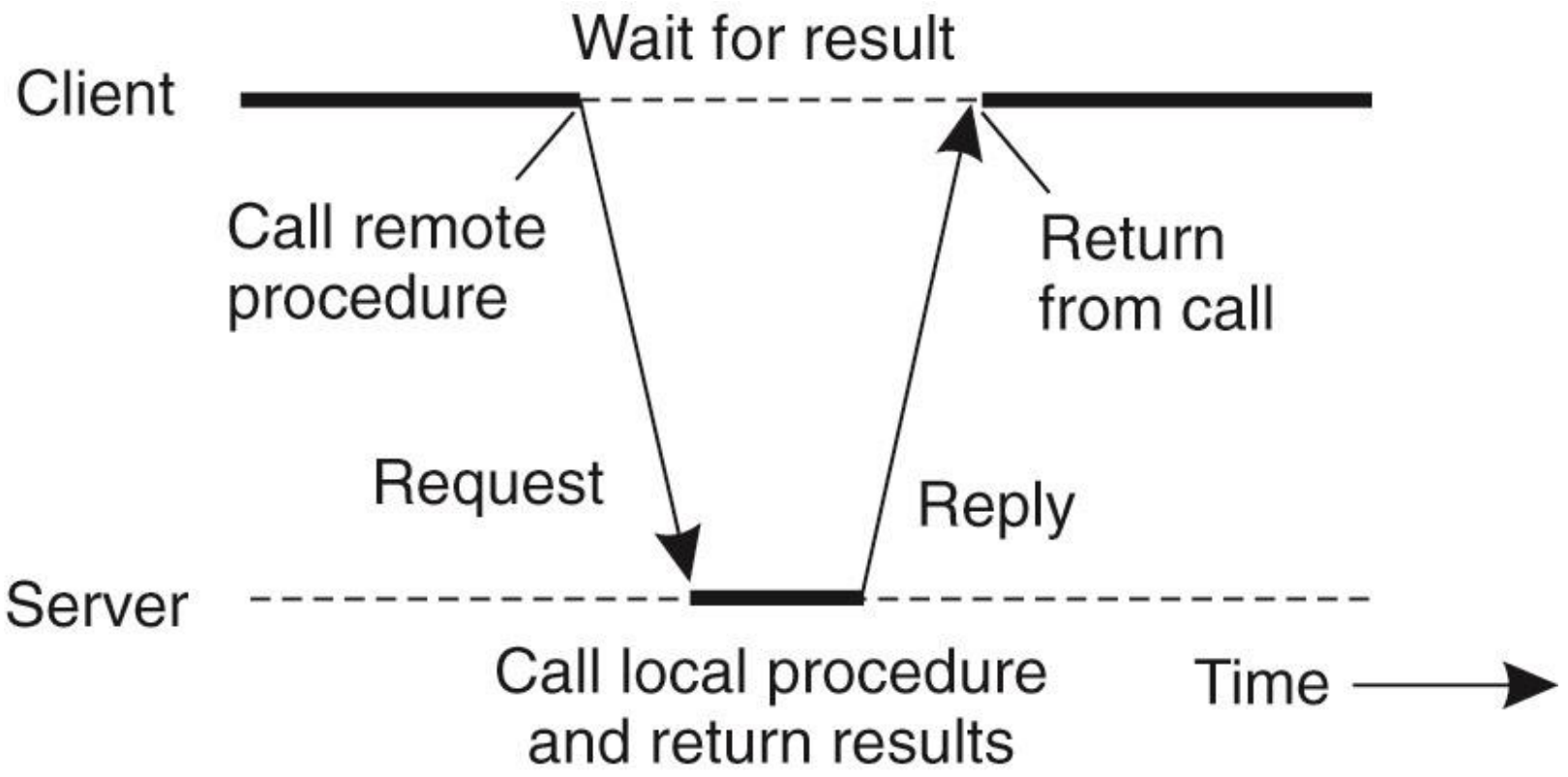


Figure 4-6. Principle of RPC between a client and server program.

Asynchronous RPC (2)

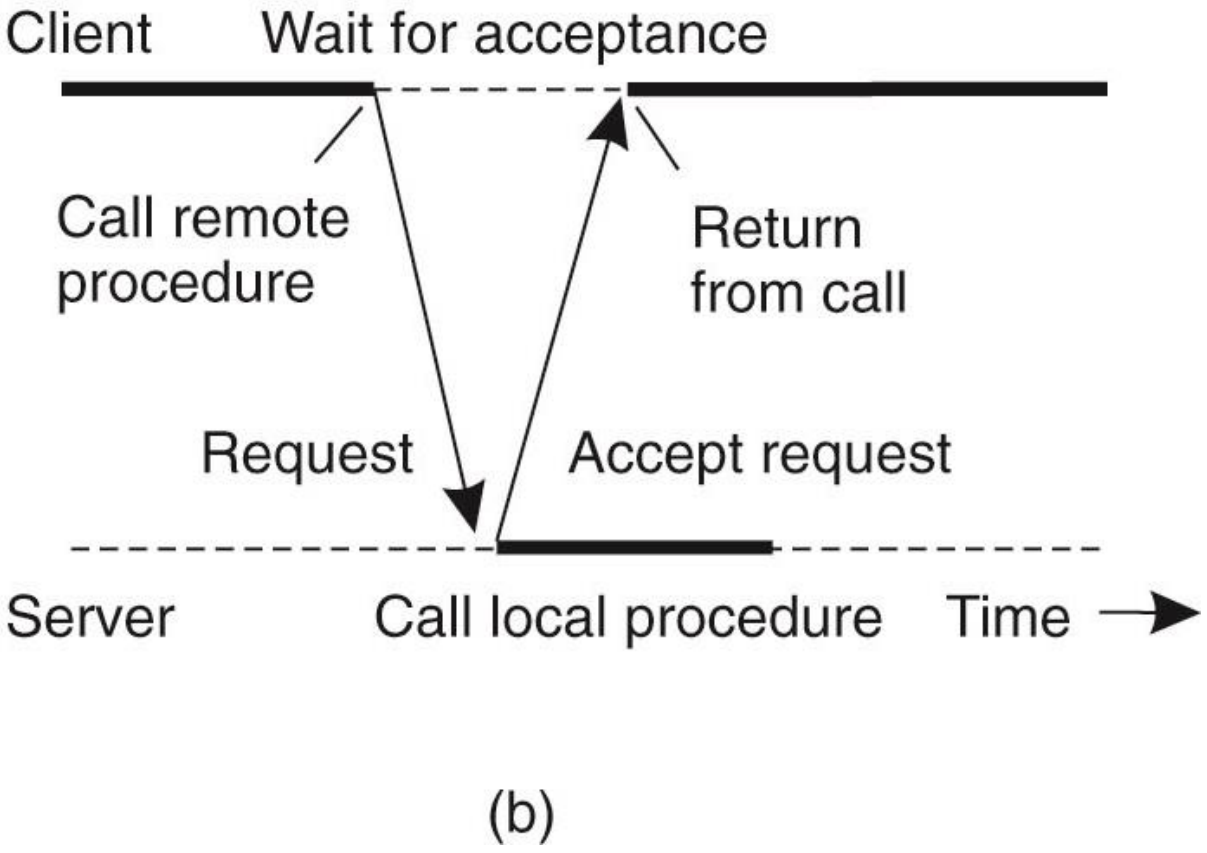


Figure 4-10. (b) The interaction using asynchronous RPC.

Asynchronous RPC (3)

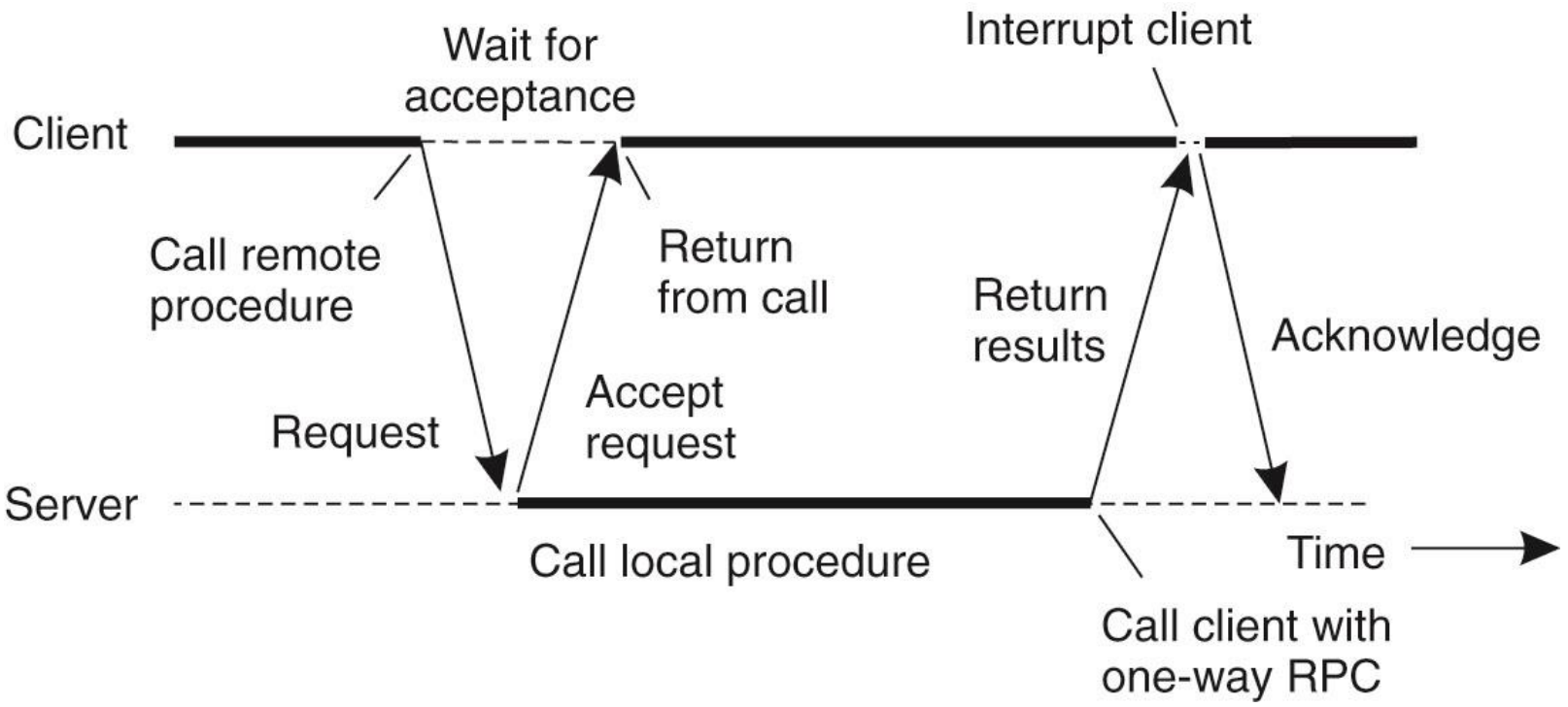


Figure 4-11. A client and server interacting through two asynchronous RPCs.

RPC: Network failure

- Client unable to locate server:
  - Return error or raise exception
- Lost requests/replies:
  - Timeout mechanisms
  - Make operation idempotent (does not change the results beyond initial operation)
  - Use sequence numbers, mark retransmissions

RPC: Server failure

- Server may crash during RPC
  - Did failure occur before or after operation?
- Operation semantics
  - Exactly once: desirable but impossible to achieve
  - At least once
  - At most once
  - No guarantee

RPC: Client failure

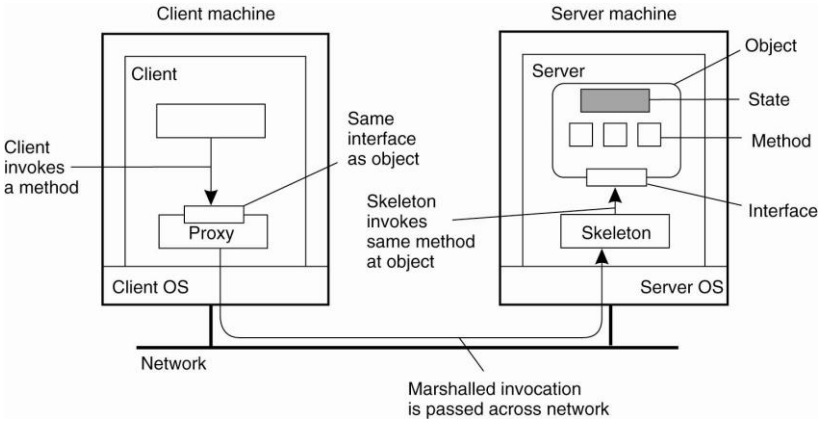
- Client crashes while server is computing
  - Server computation becomes orphan
- Possible actions
  - Extermination: log at client stub and explicitly kill orphans
  - Reincarnation: Divide time into epochs between failures and delete computations from old epochs
  - Expiration: give each RPC a fixed quantum T; explicitly request extensions

Remote method invocation (RMI)

- RPCs applied to distributed objects
- Class: object-oriented abstraction
- Object: instance of class
  - Encapsulates data
  - Export methods: operations on data
  - Separation between interface and implementation

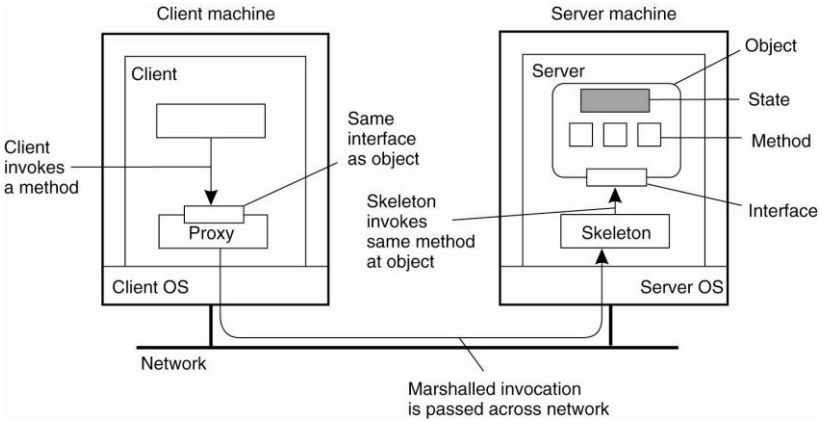
Proxies and skeletons

- Proxy: client stub
  - Maintains server ID, endpoint, object ID
  - Does parameter marshalling
  - In practice, can be downloaded/constructed on the fly
- Skeleton: server stub
  - Does demarshalling and passes parameters to server
  - Sends result to proxy



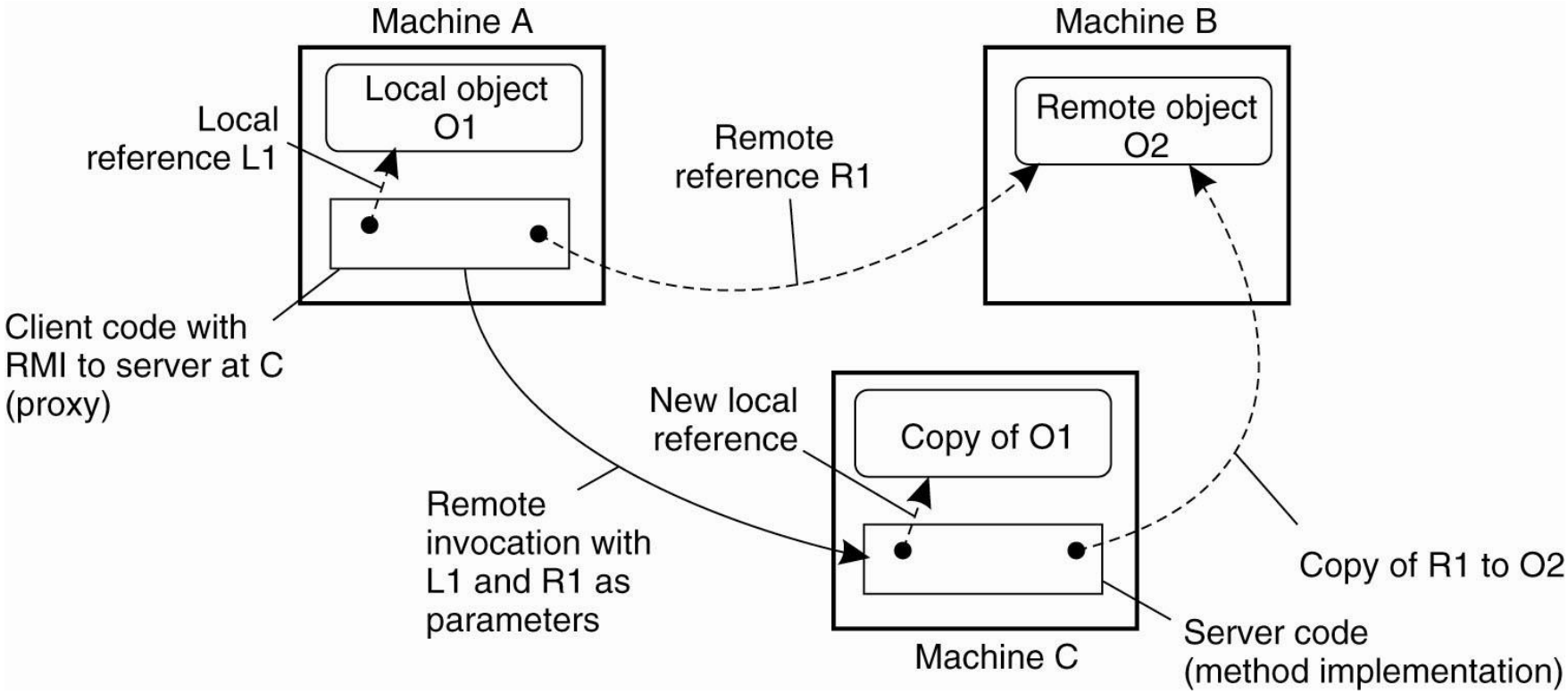
Binding a client to an object

- Loading a proxy in client address space
- Implicit binding:
  - Bound automatically on object reference resolution
- Explicit binding:
  - Client has to first bind object
  - Call method after binding

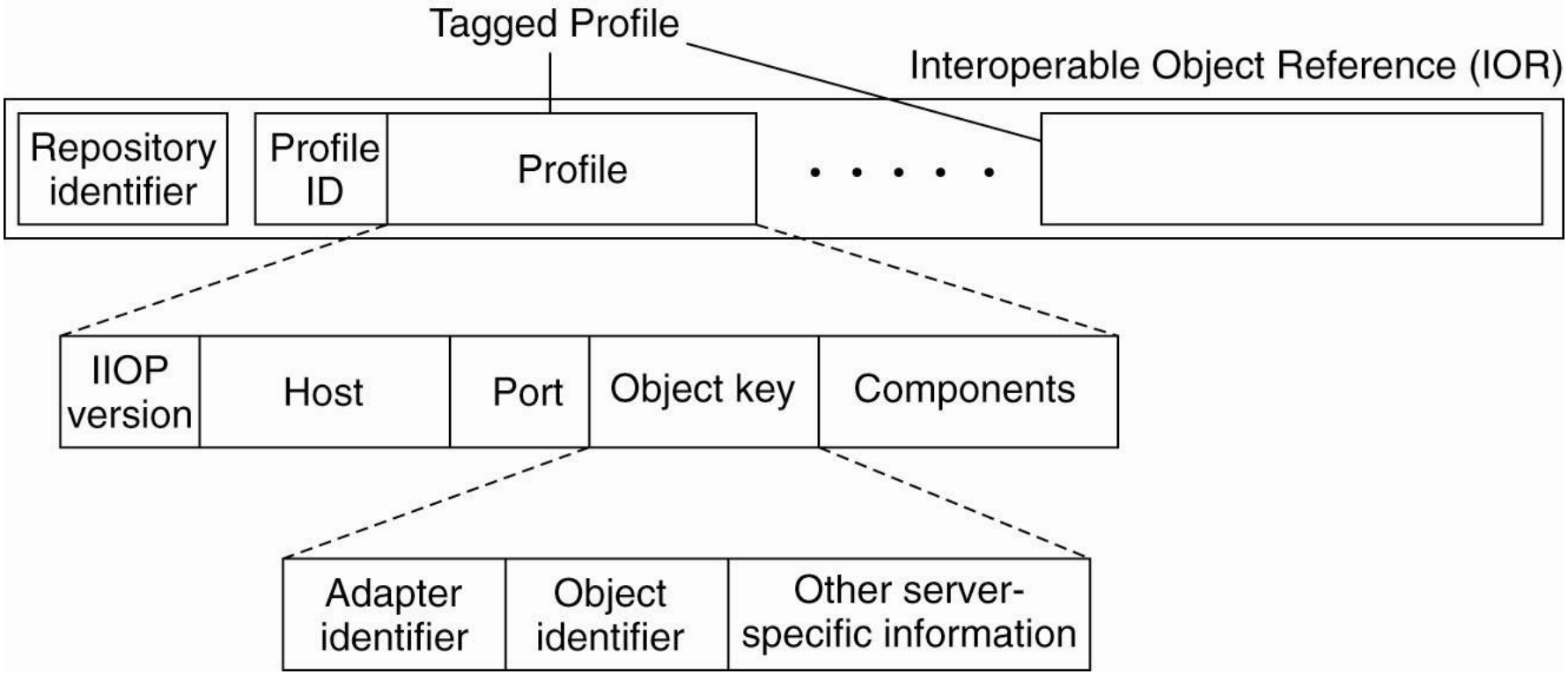


Parameter passing

- Less restrictive than RPCs
  - Supports system-wide object references
  - Pass local objects by value, remote objects by reference

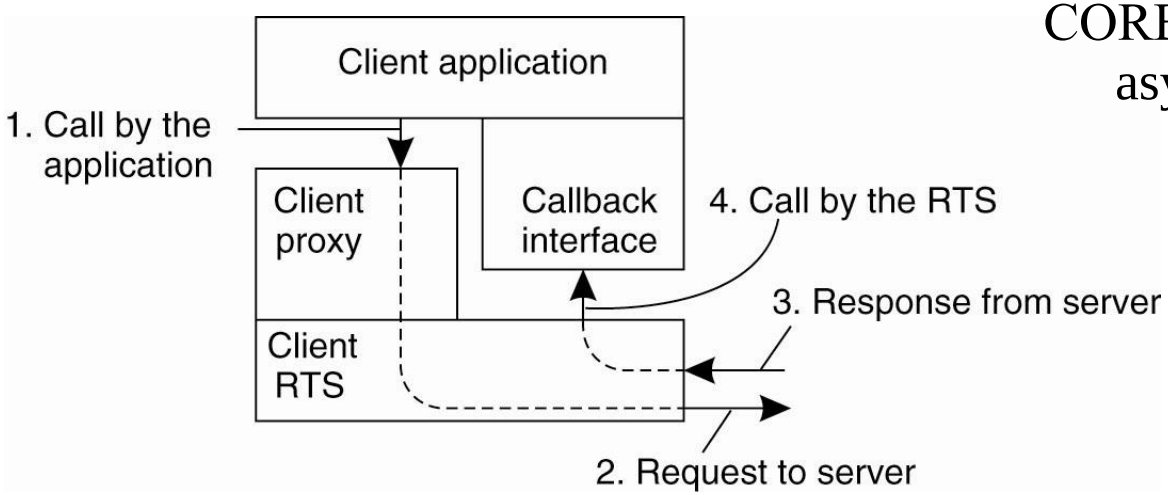


Naming: CORBA Object References

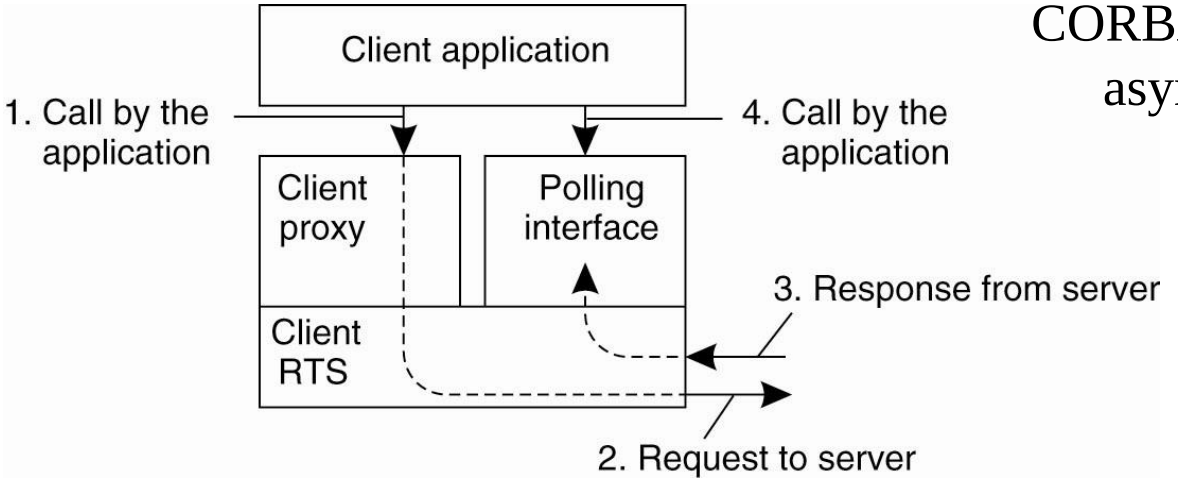


The organization of an IOR ...

Object-based messaging



CORBA's callback model for asynchronous method invocation.



CORBA's polling model for asynchronous method invocation.