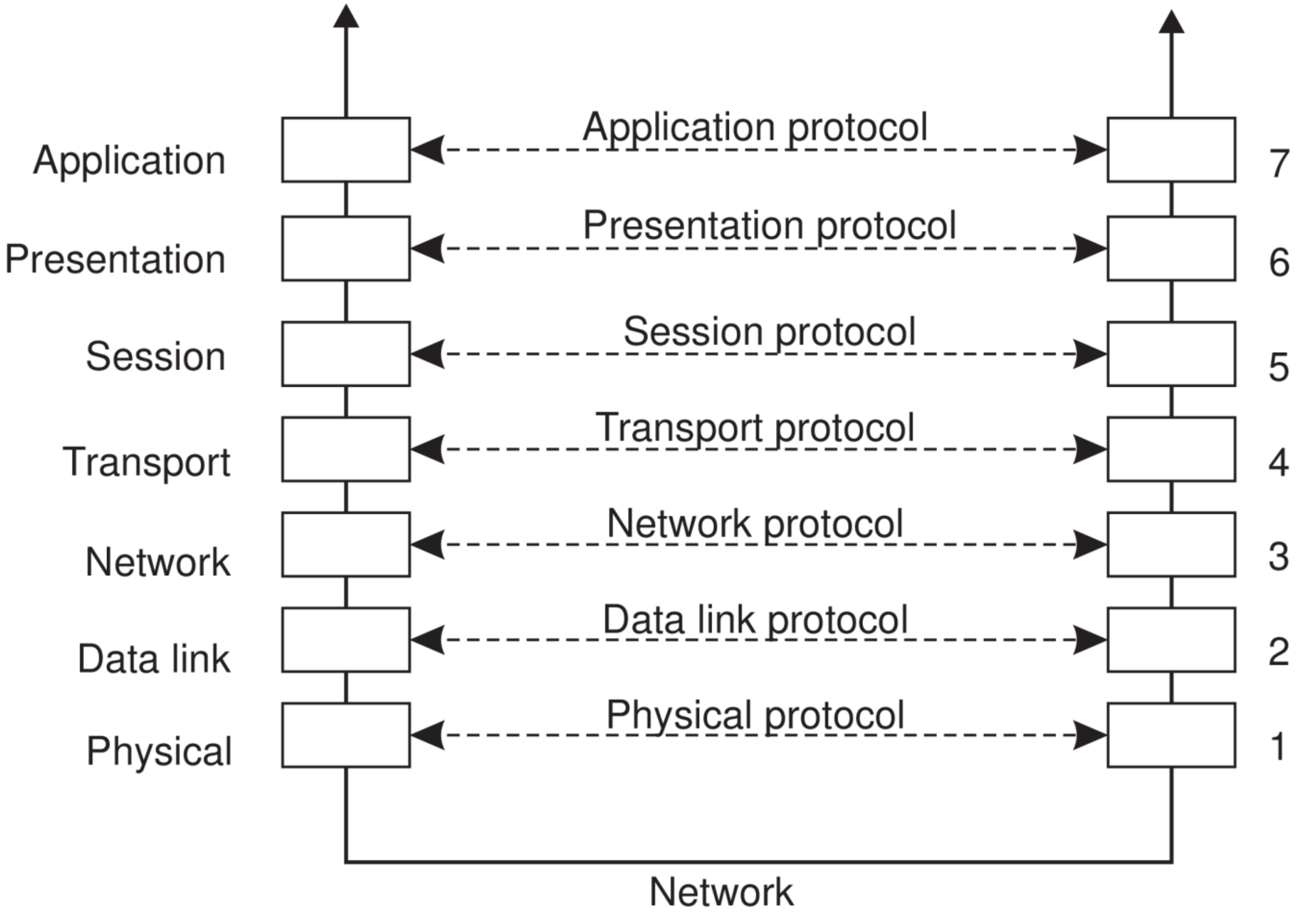
Distributed Systems

Basic networking model



Low-level layers

- Physical layer: contains the specification and implementation of bits, and their transmission between sender and receiver
- Data link layer: prescribes the transmission of a series of bits into a frame to allow for error and flow control
- Network layer: describes how packets in a network of computers are to be routed.

Transport Layer

- 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

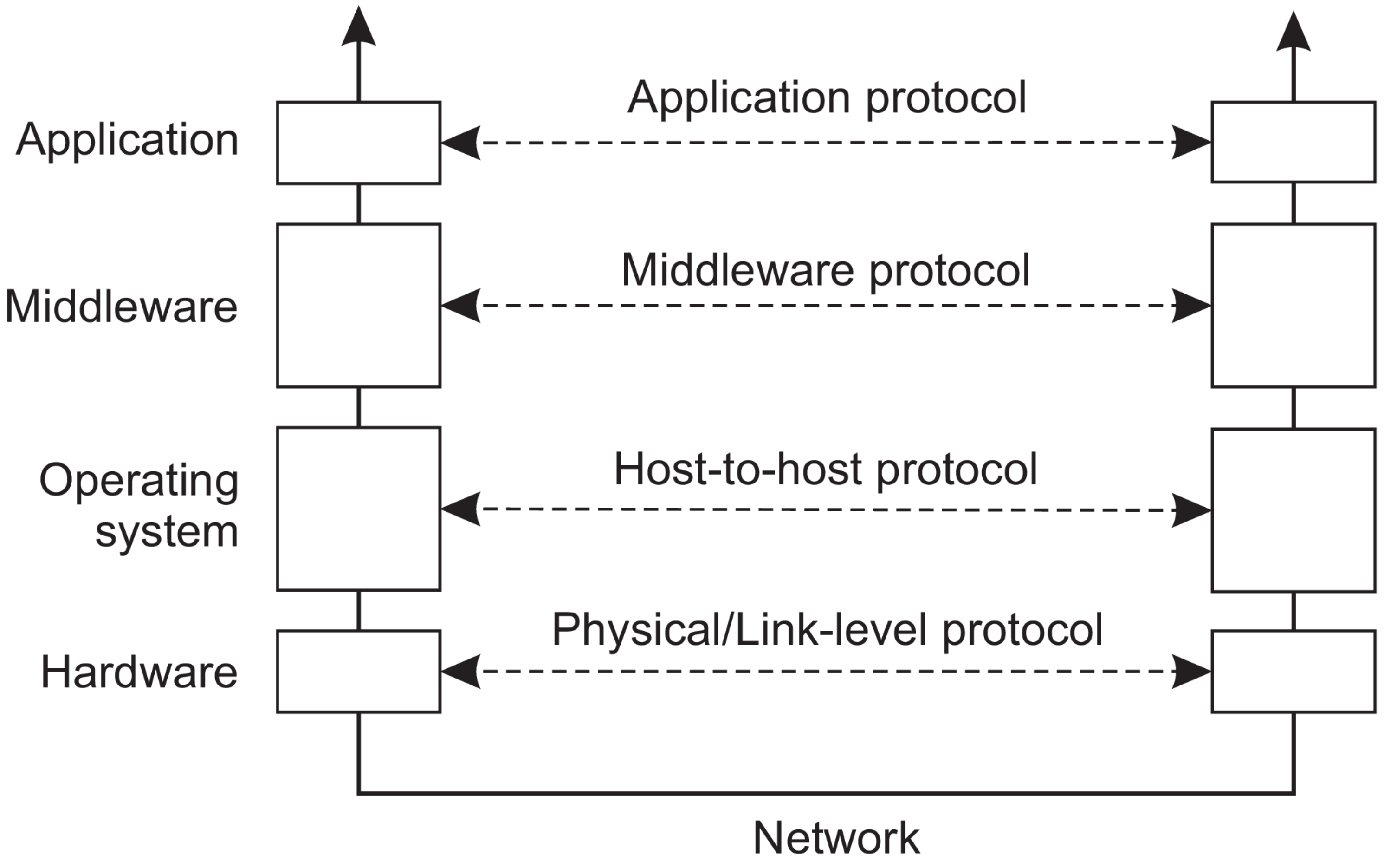
Middleware layer

- Middleware is invented to provide common services and protocols that can be used by many different applications
- A rich set of communication protocols
- (Un)marshaling of data, necessary for integrated systems
- Naming protocols, to allow easy sharing of resources
- Security protocols for secure communication
- Scaling mechanisms, such as for replication and caching

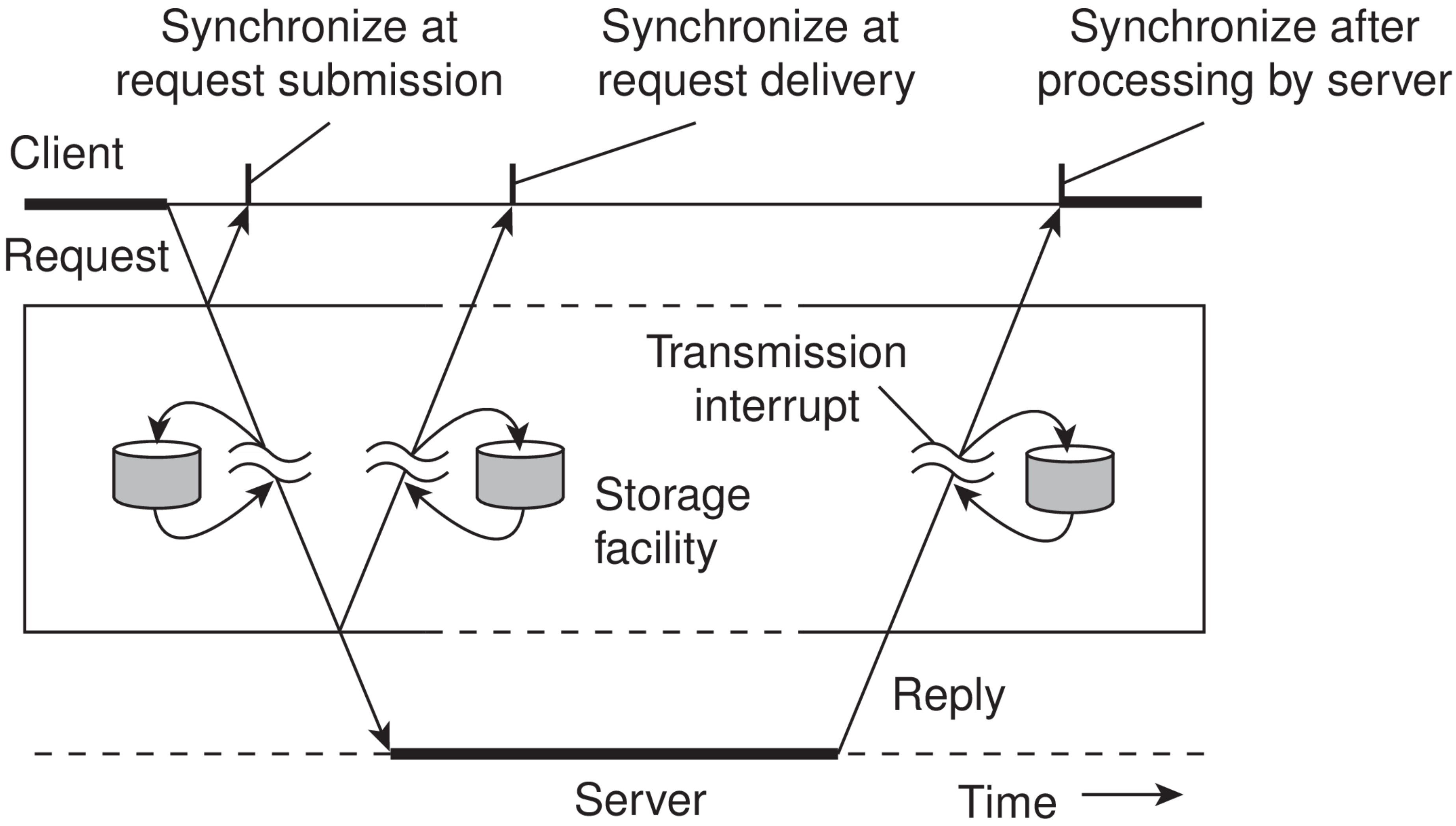
Note

- What remains are truly application-specific protocols... such as?

An adapted layering scheme



Types of communication


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

Types of communication

- Transient versus persistent communication
- Asynchronous versus synchronous communication

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.

At request submission

At request delivery

After request processing

Client/Server

- Client/Server computing is generally based on a model of transient synchronous communication:
- Client and server have to be active at 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)

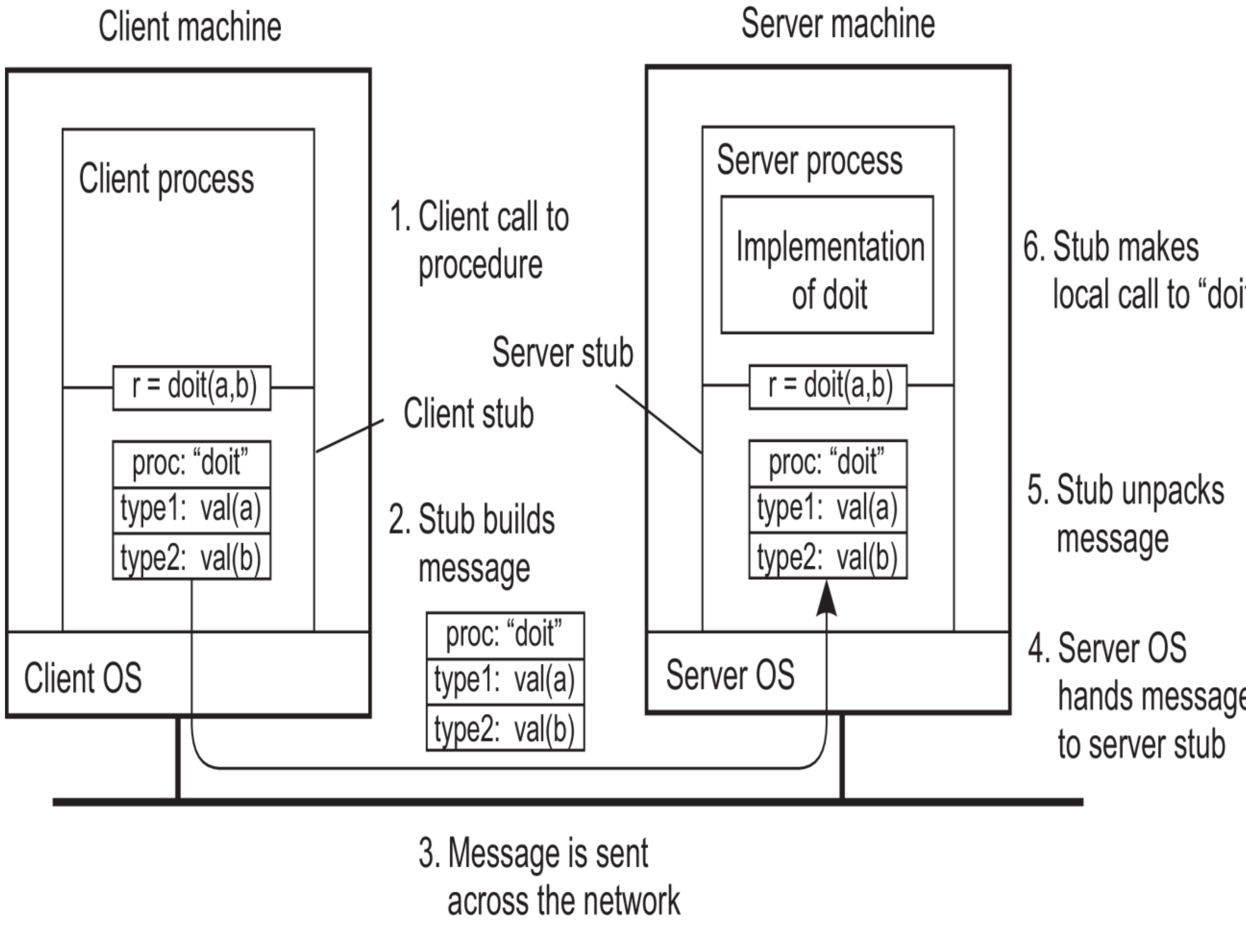
Messaging

Message-oriented middleware

- Aims at high-level persistent asynchronous communication:
- Processes send each other messages, which are queued
- Sender need not wait for immediate reply, but can do other things
- Middleware often ensures fault tolerance

Basic RPC operation

1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters; calls server.
6. Server does local call; returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result; returns to client.



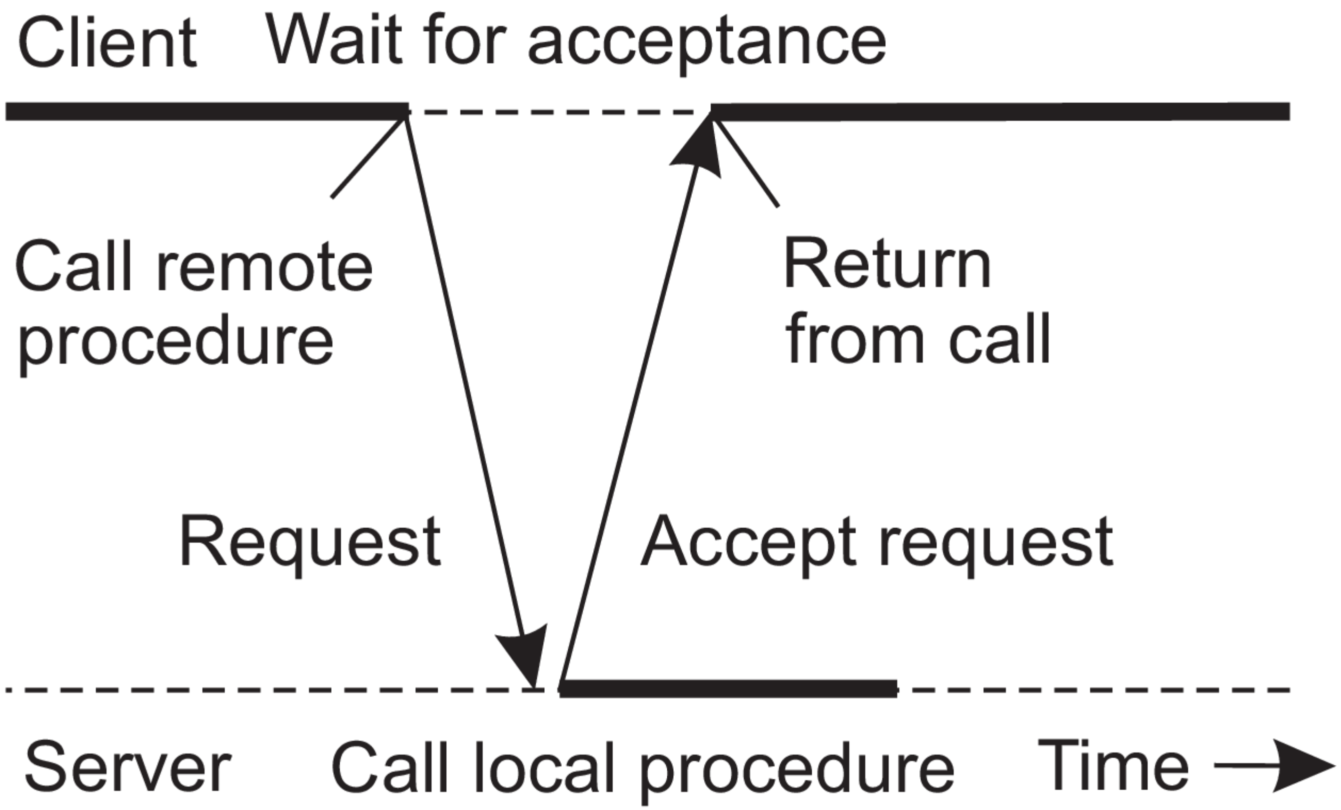
RPC: Parameter passing

There's more than just wrapping parameters into a message

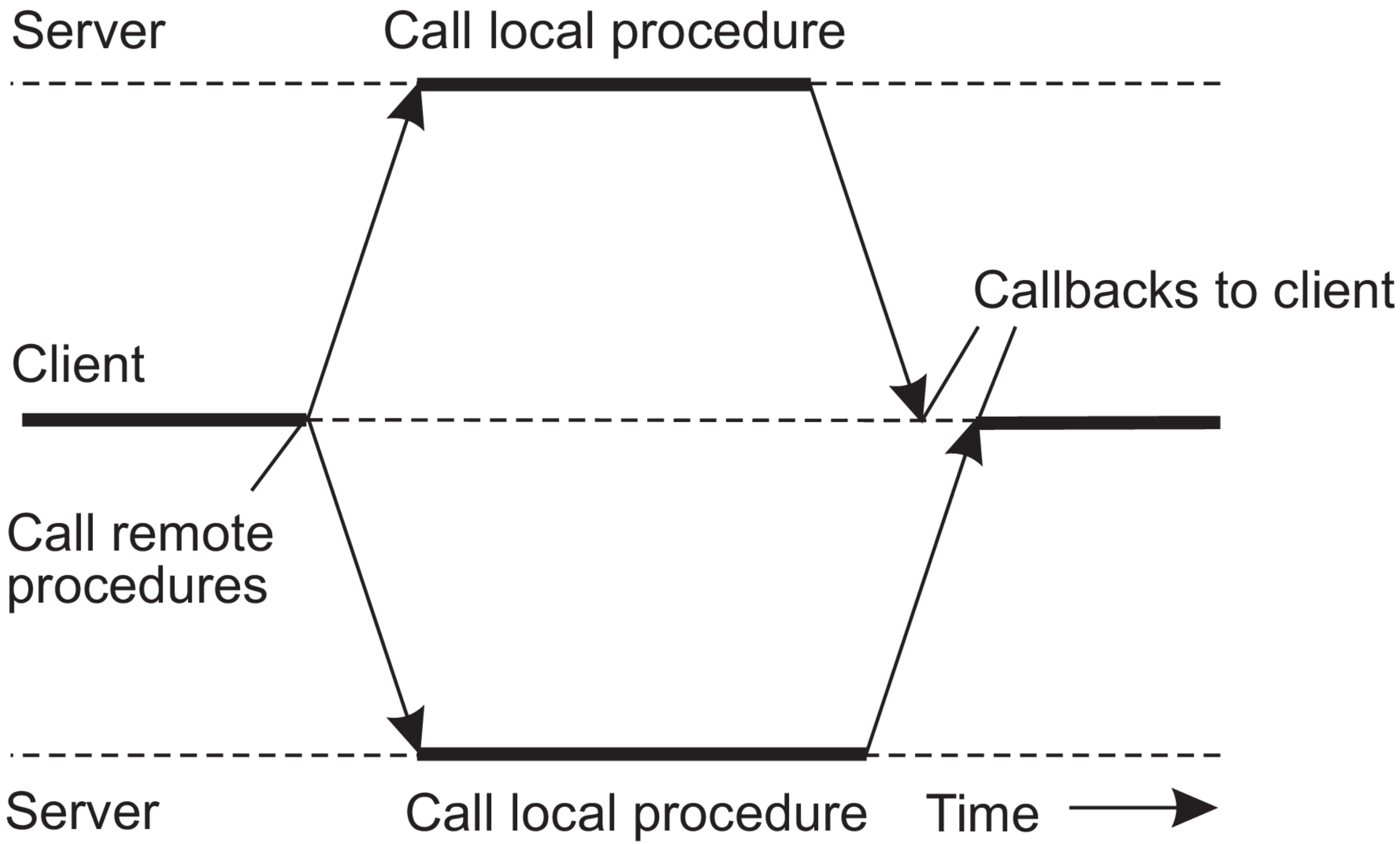
- Client and server machines may have different data representations (think of byte ordering)
- Wrapping a parameter means transforming a value into a sequence of bytes
- Client and server have to agree on the same encoding:
- How are basic data values represented (integers, floats, characters)
- How are complex data values represented (arrays, unions)

* Client and server need to properly interpret messages, transforming them into machine-dependent representations.

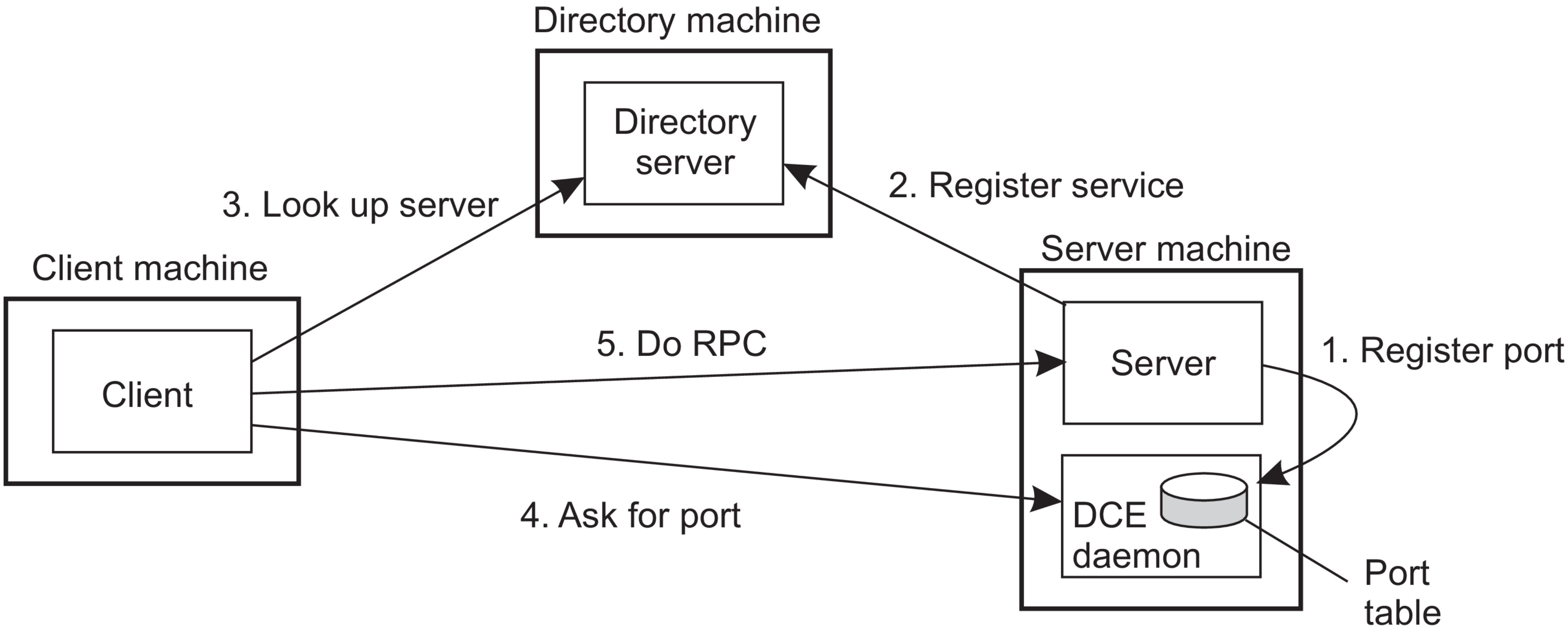
Asynchronous RPCs



Sending out multiple RPCs



Client-to-server binding



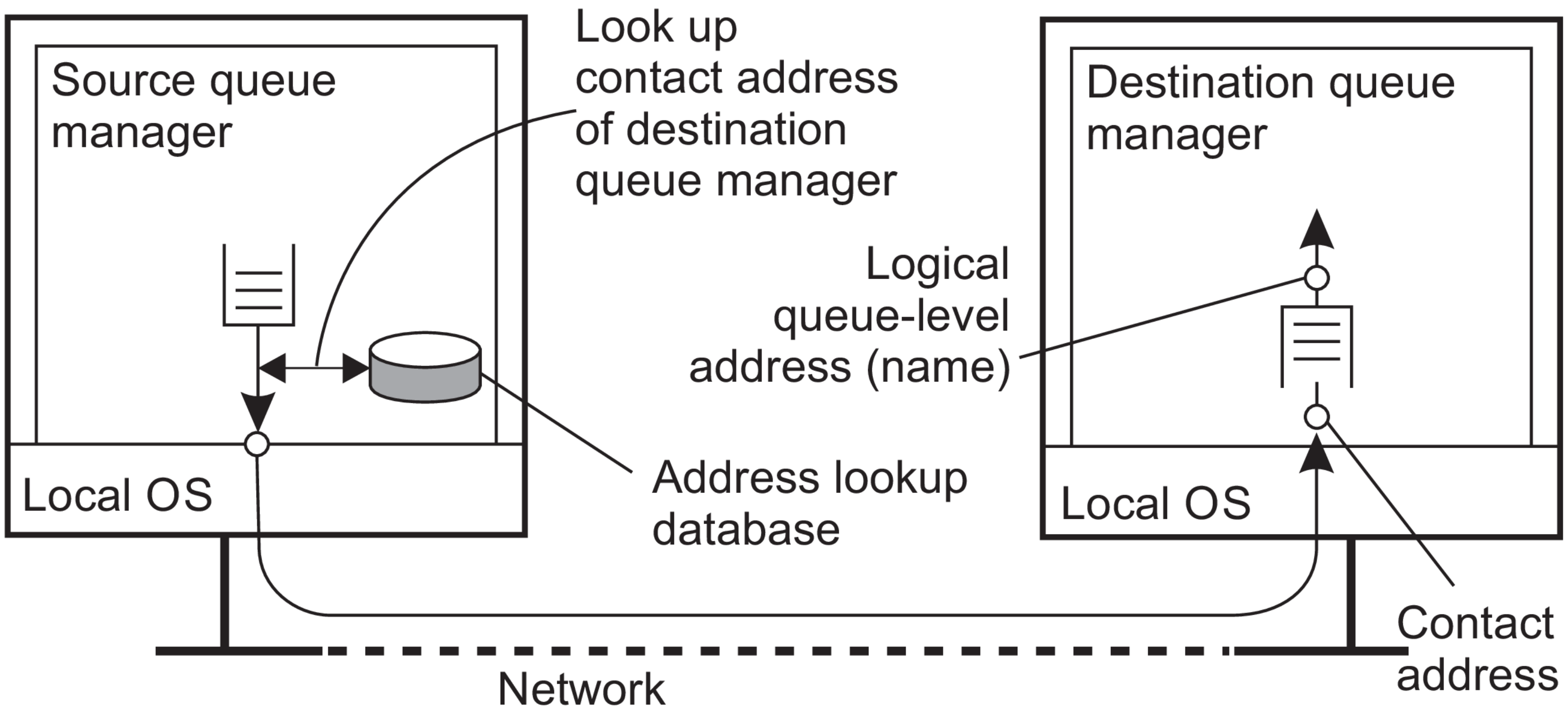
Message-oriented middleware

- Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers.

Operation	Description
put	Append a message to a specified queue
get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
notify	Install a handler to be called when a message is put into the specified queue

General model Queue managers

- Queues are managed by queue managers. An application can put messages only into a local queue. Getting a message is possible by extracting it from a local queue only ⇒ queue managers need to route messages.



Source queue manager
Look up contact address of destination queue manager
Destination queue manager
Logical queue-level address (name)
Local OS
Address lookup database
Local OS
Contact address
Network

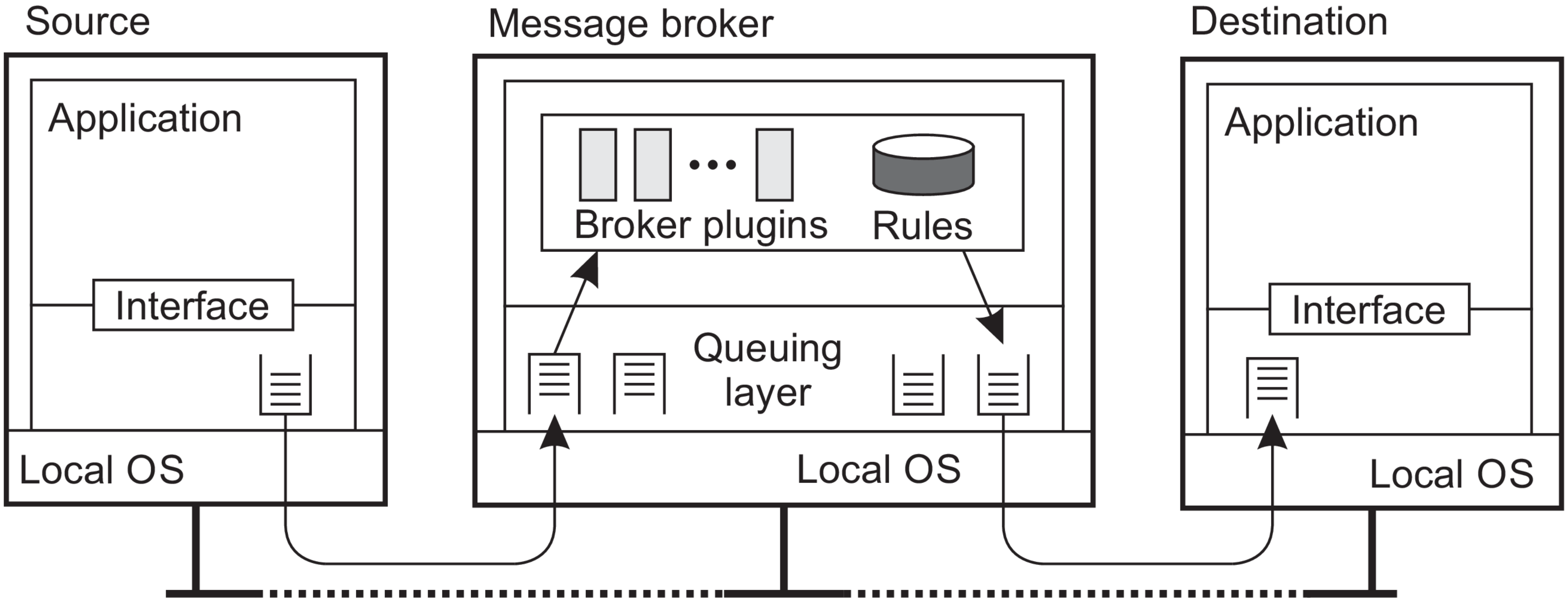
Message broker

- Message queuing systems assume a common messaging protocol: all applications agree on message format (i.e., structure and data representation)

Broker handles application heterogeneity in an MQ system

- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities (i.e., publish-subscribe capabilities)

Message broker: general architecture


Source
Message broker
Destination
Application
Broker plugins
Rules
Application
Interface
Queuing layer
Interface
Local OS
Local OS
Local OS