Distributed Systems

Mahmoud Abou El-Magd Soliman
Department of Computer Science
Faculty of Computers and Artificial Intelligence
Sohag University

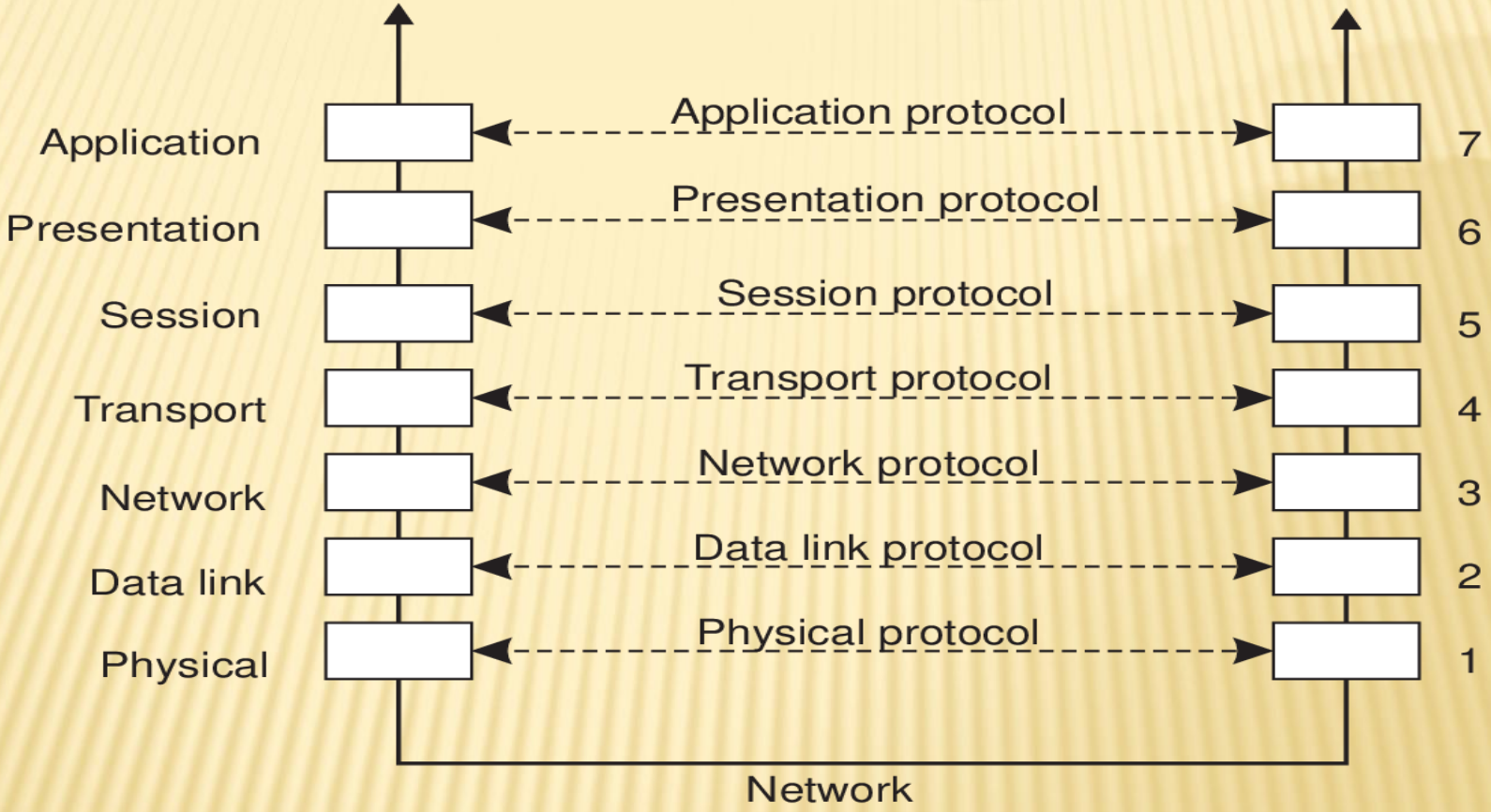
Chapter Four

Communication

Introduction

Interprocess communication is at the heart of all distributed systems. It makes no sense to study distributed systems without carefully examining the ways that processes on different machines can exchange information. Communication in distributed systems is always based on low-level message passing as offered by the underlying network. Expressing communication through message passing is harder than using primitives based on shared memory

Basic Networking Model



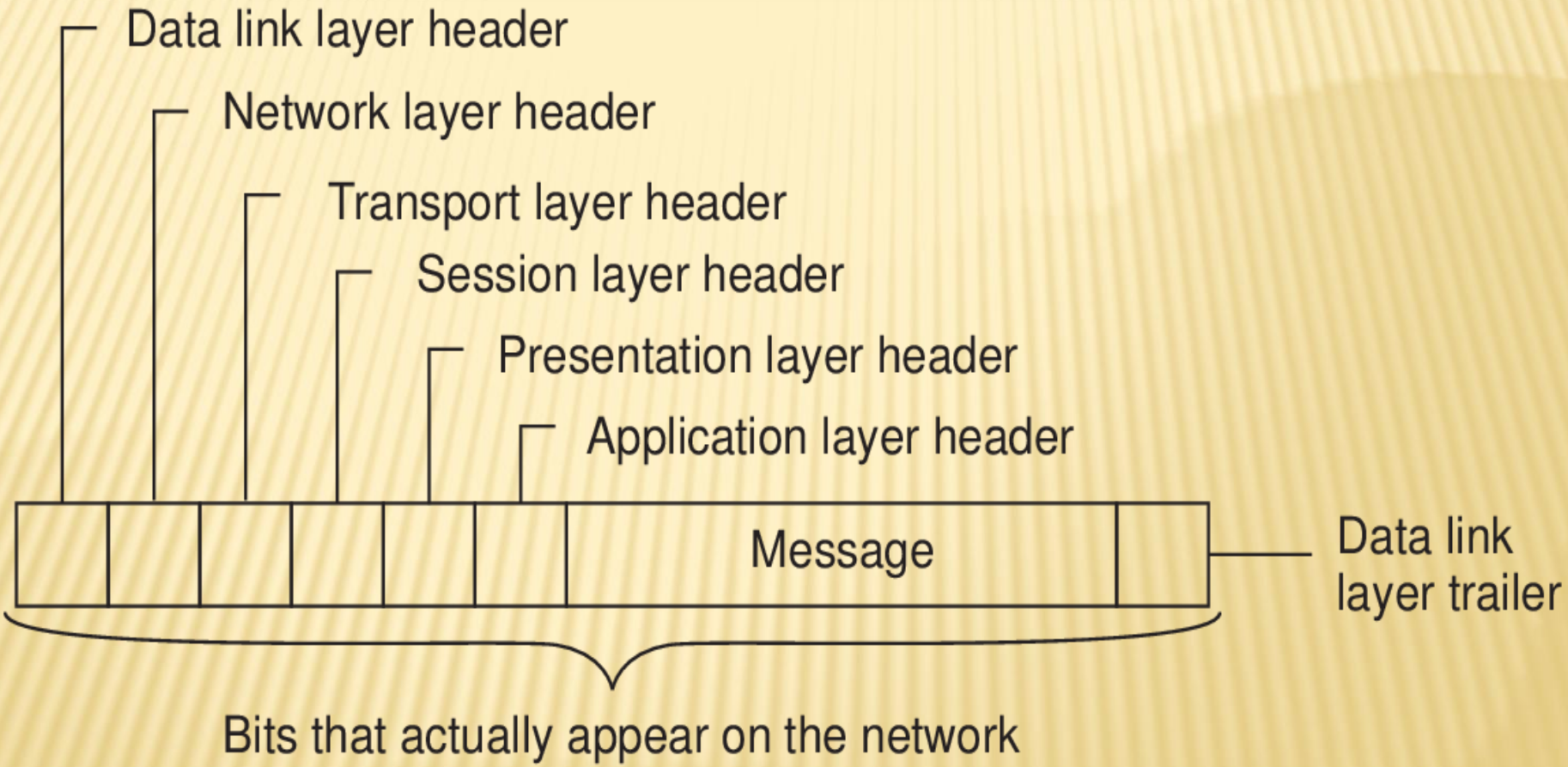
Drawbacks

- Focus on message-passing only
- Often unneeded or unwanted functionality
- Violates access transparency

A Typical Message

In the OSI model, communication is divided up into seven levels or layers, as shown in Figure. Each layer deals with one specific aspect of the communication. In this way, the problem can be divided up into manageable pieces, each of which can be solved independent of the others. Each layer provides an interface to the one above it. The interface consists of a set of operations that together define the service the layer is prepared to offer its users.

A Typical Message



A typical message as it appears on the network.

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.

Observation

For many distributed systems, the lowest-level interface is that of the network layer.

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

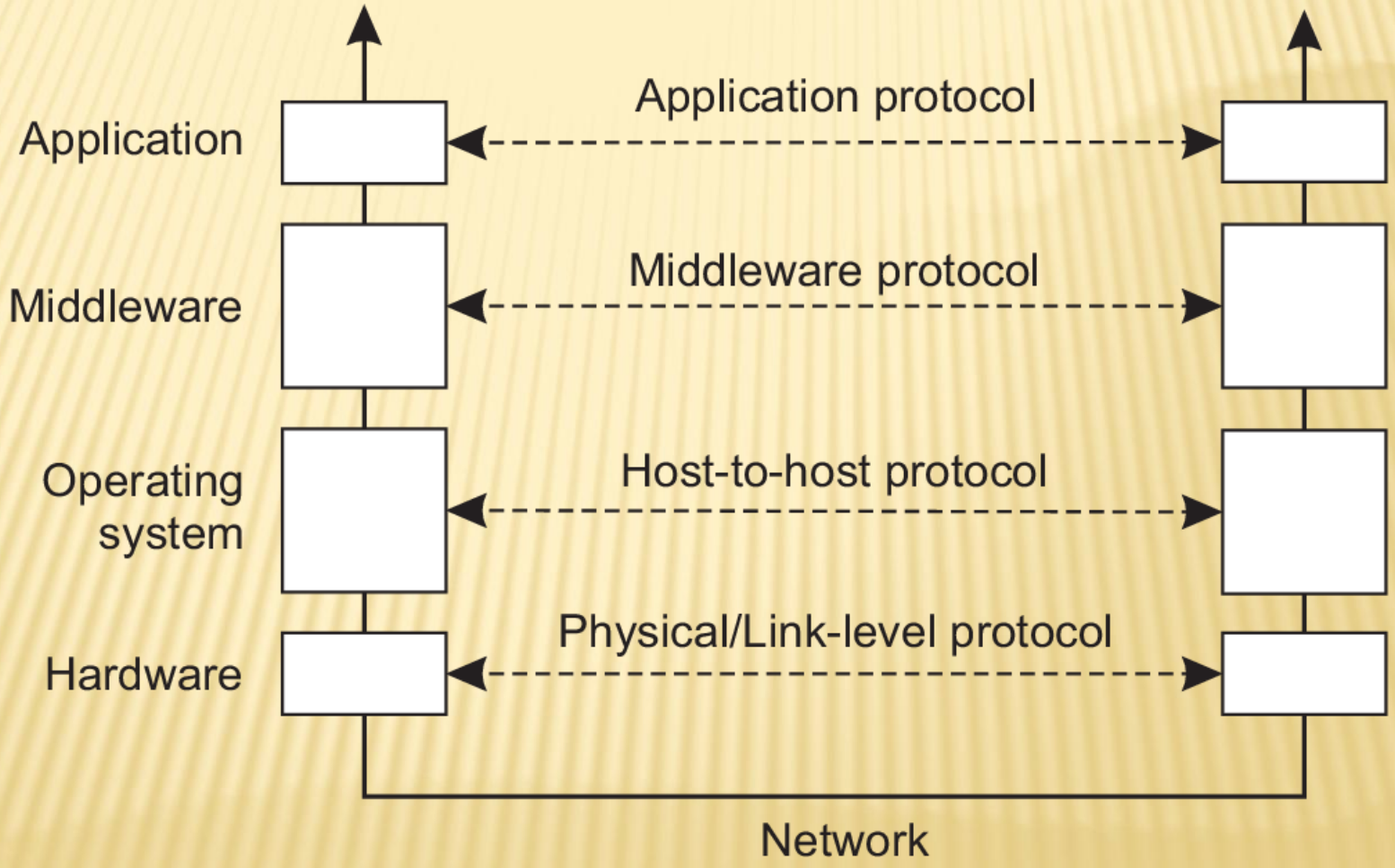
Middleware layer

Observation

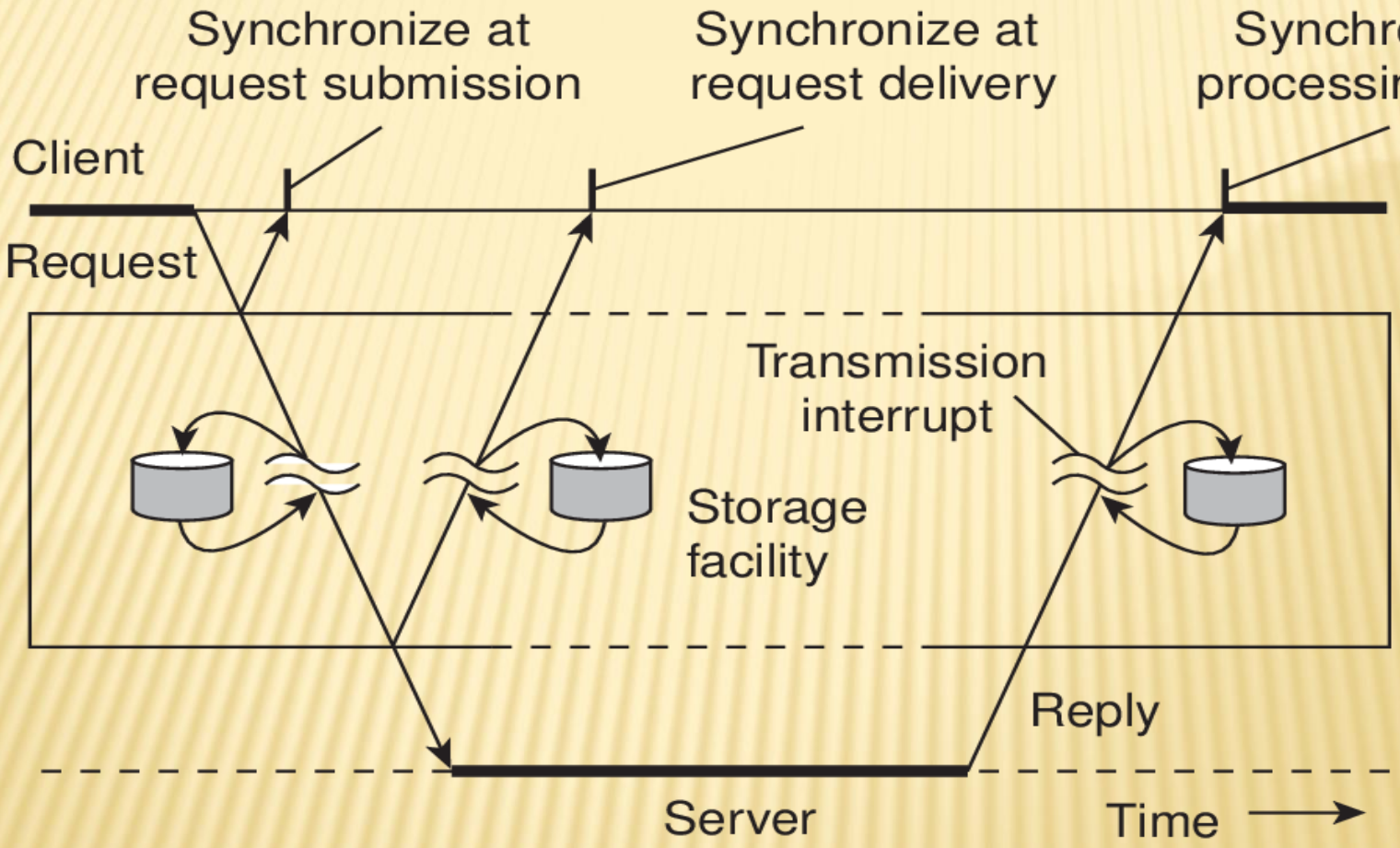
Middleware is invented to provide common services and protocols that can be used by many different applications

- 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

An Adapted Layering Scheme



Types of Communication



- 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

Some observations

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

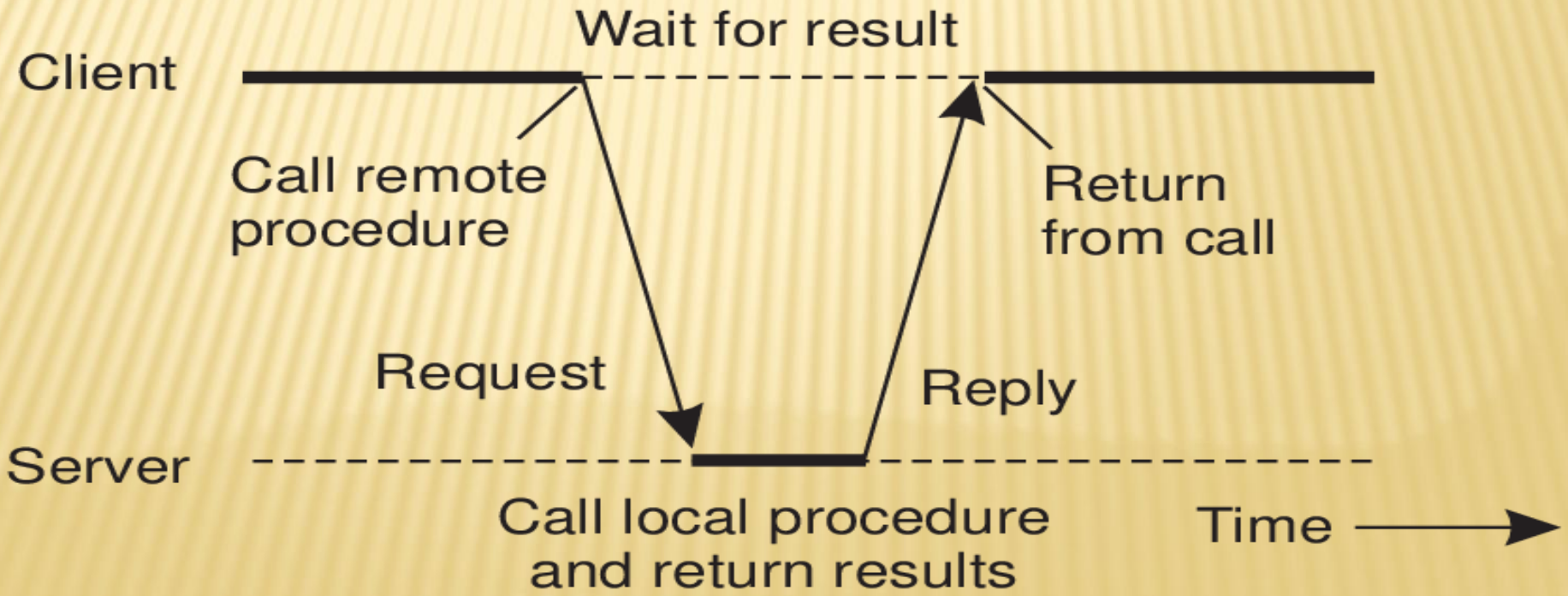
Observations

- Application developers are familiar with simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machine

Basic RPC Operation

Conclusion

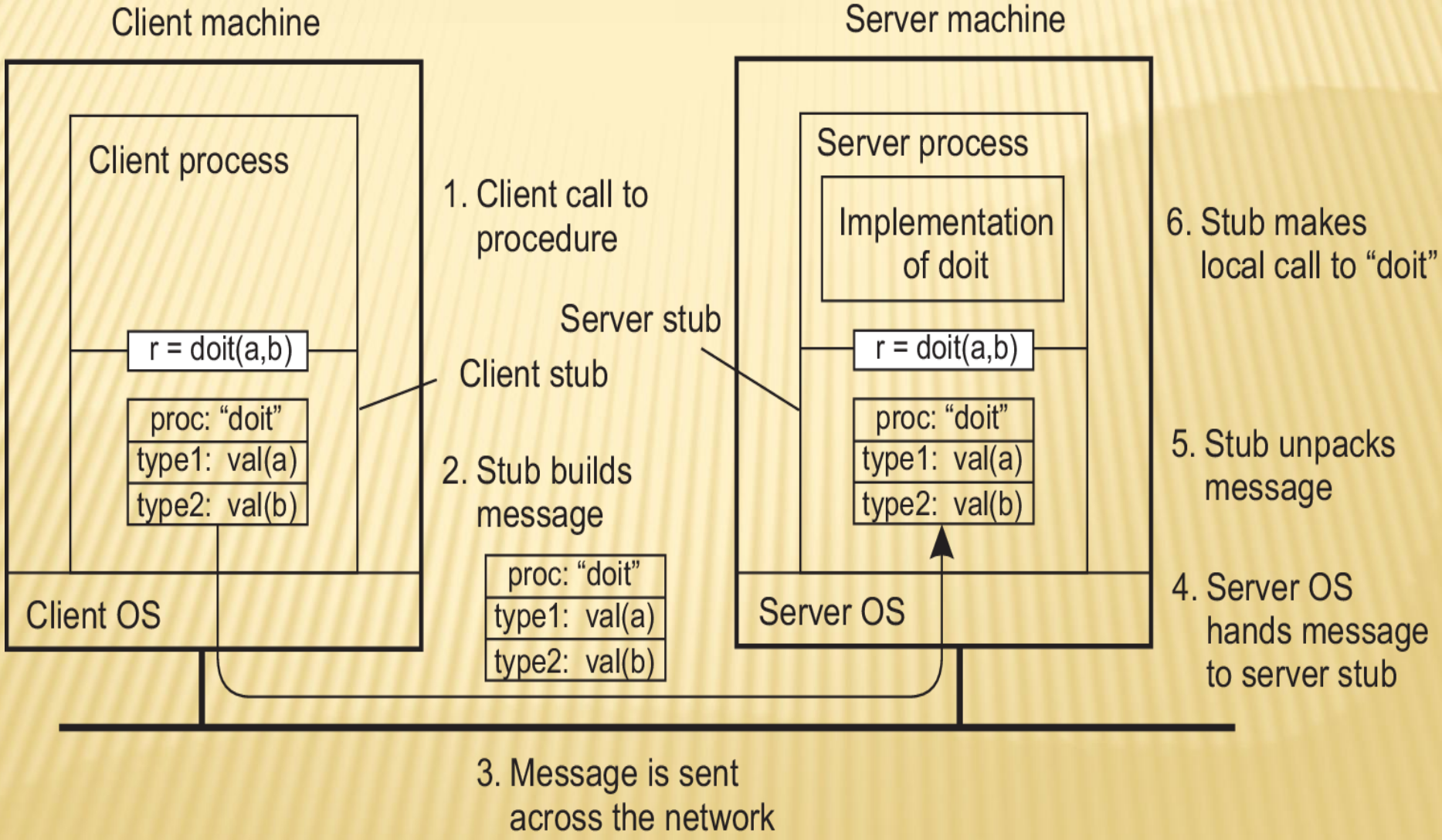
Communication between caller & callee can be hidden by using procedure-call mechanism



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) Server does local call; returns result to stub.
6) Stub builds message; calls OS.
7) OS sends message to client's OS.
8) Client's OS gives message to stub.
9) Client stub unpacks result; returns to

Basic RPC Operation



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)

RPC: Parameter Passing

Some assumptions

- Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
- All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.

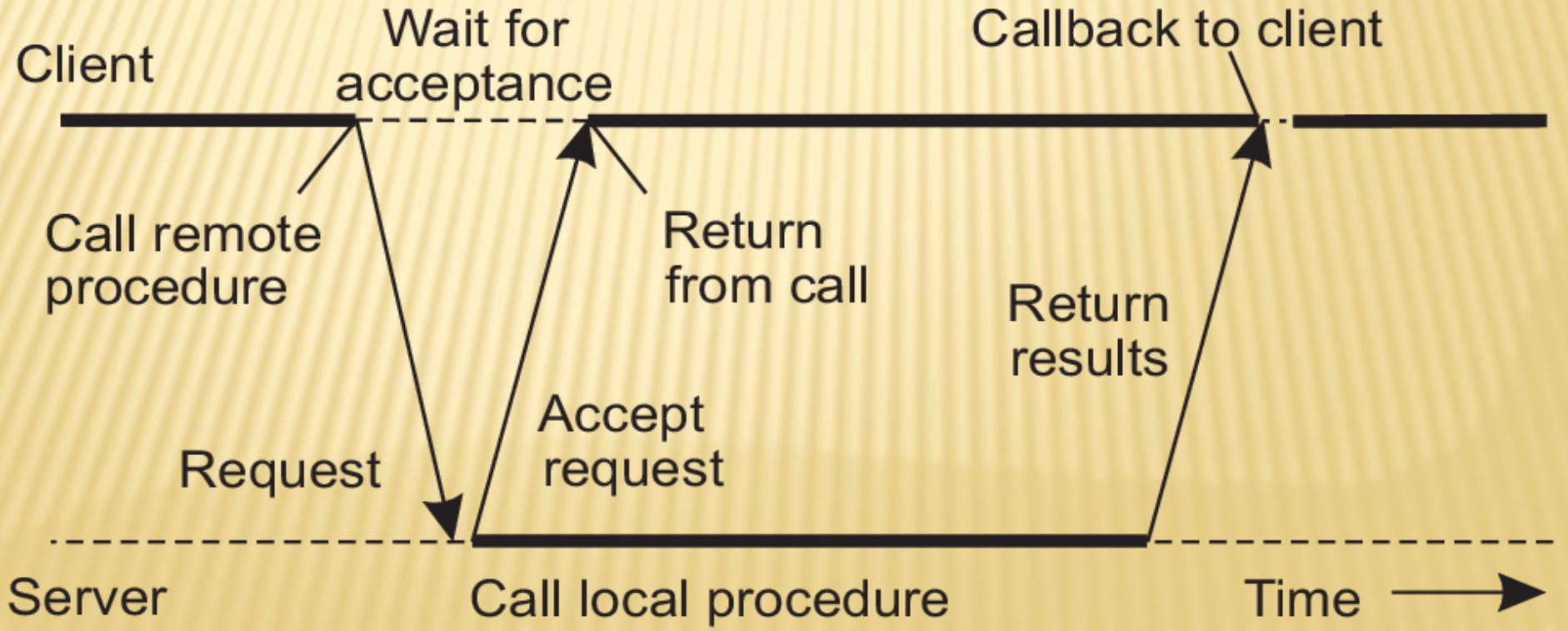
Conclusion

Full access transparency cannot be realized.

Asynchronous RPCs

Essence

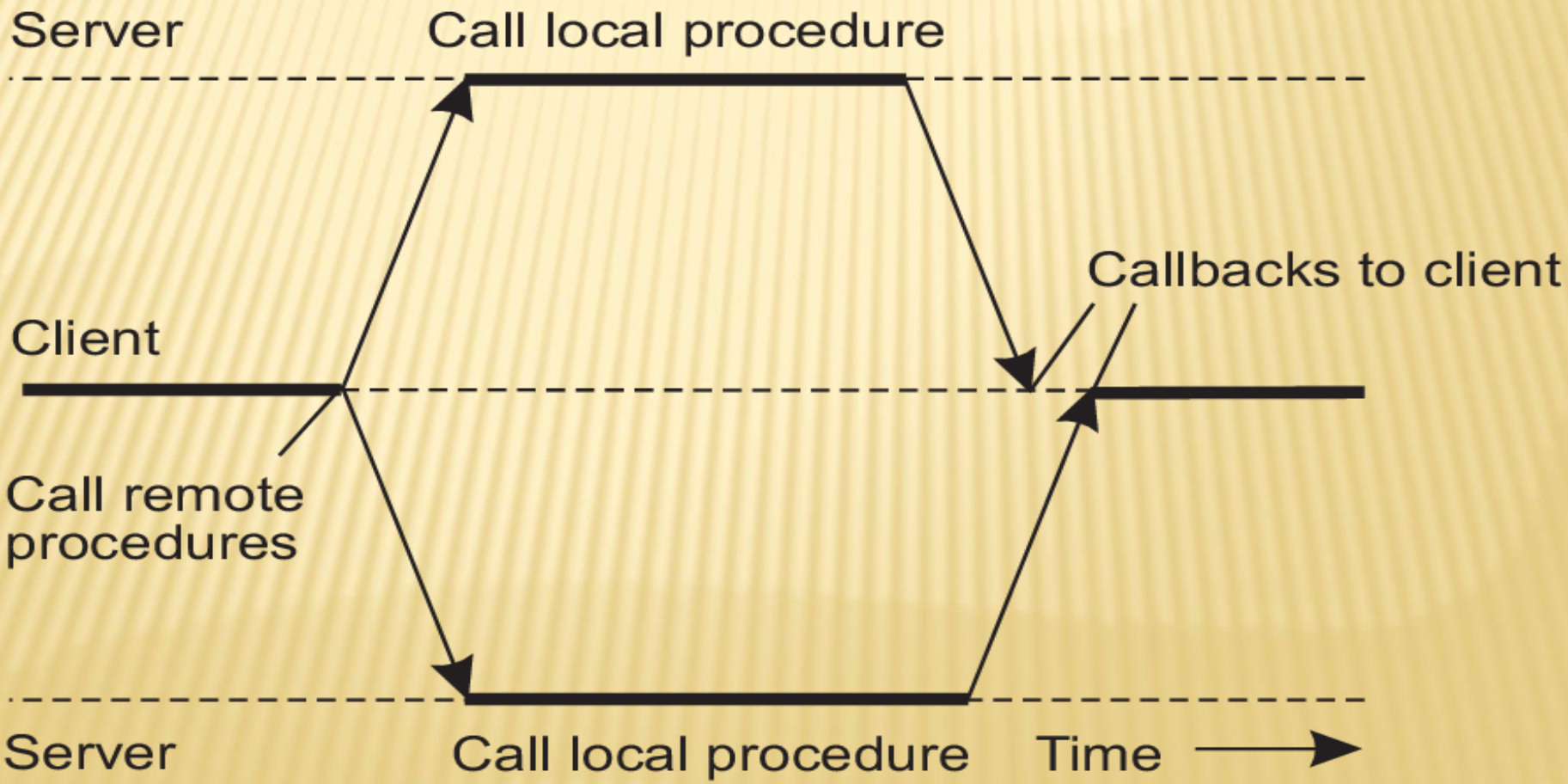
Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



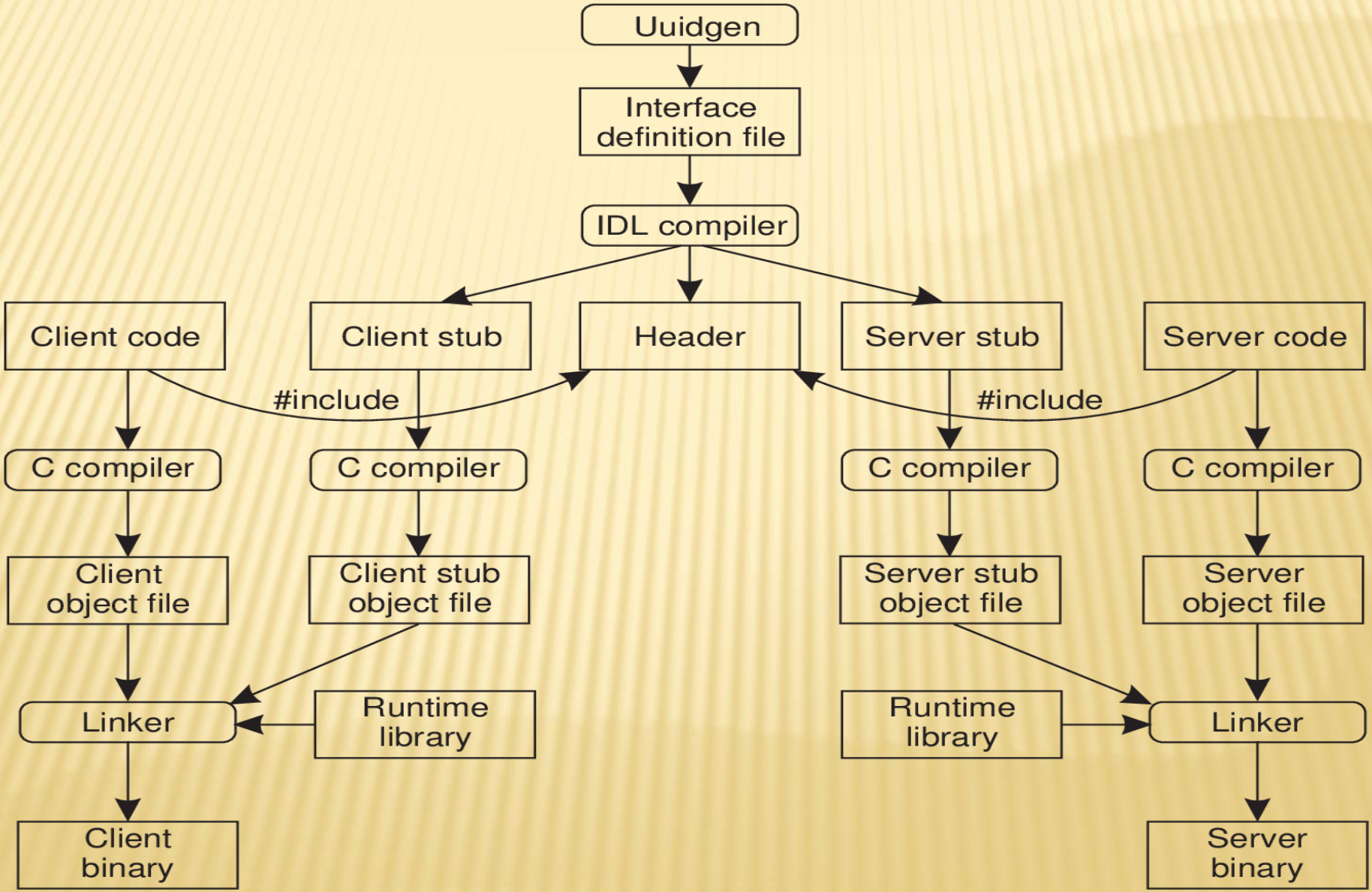
Sending out Multiple RPCs

Essence

Sending an RPC request to a group of servers.



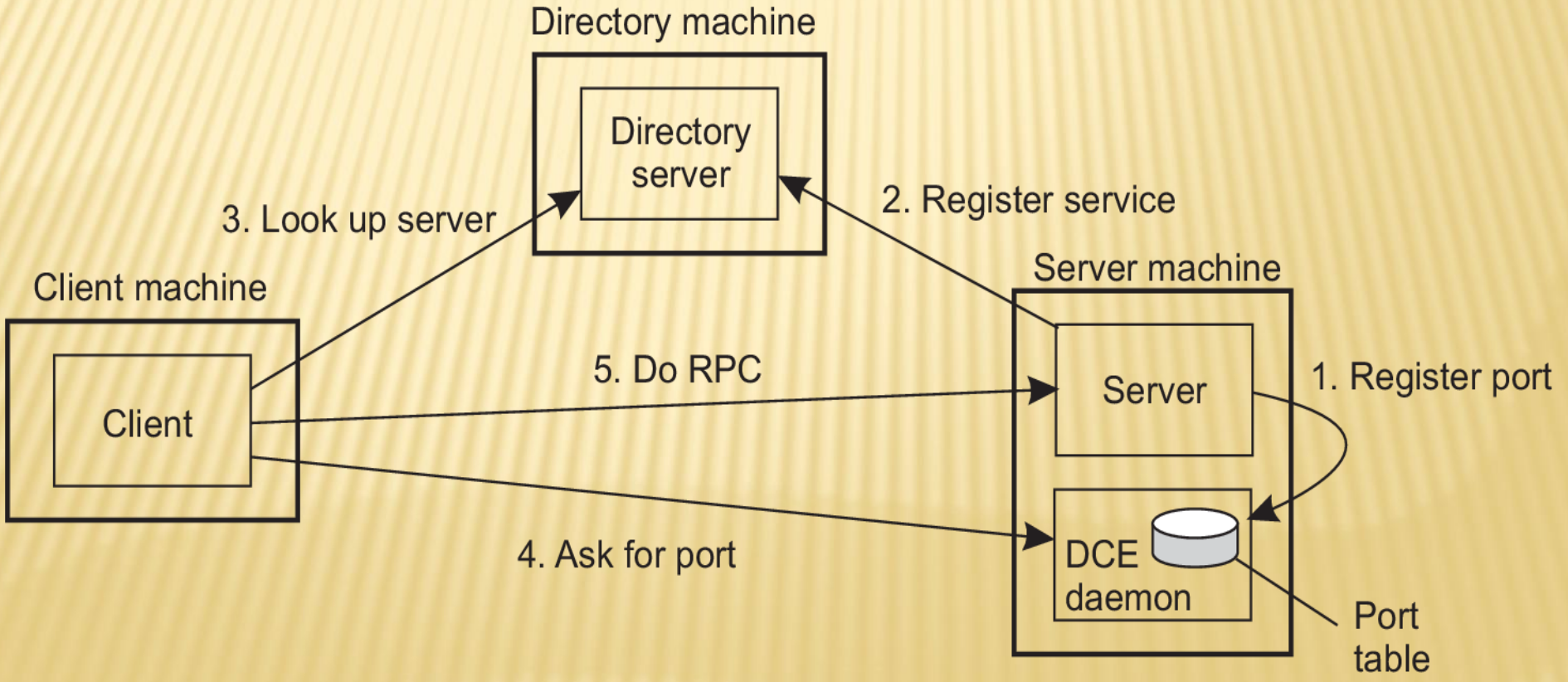
RPC in Practice



Client-to-Server Binding (DCE)

Issues

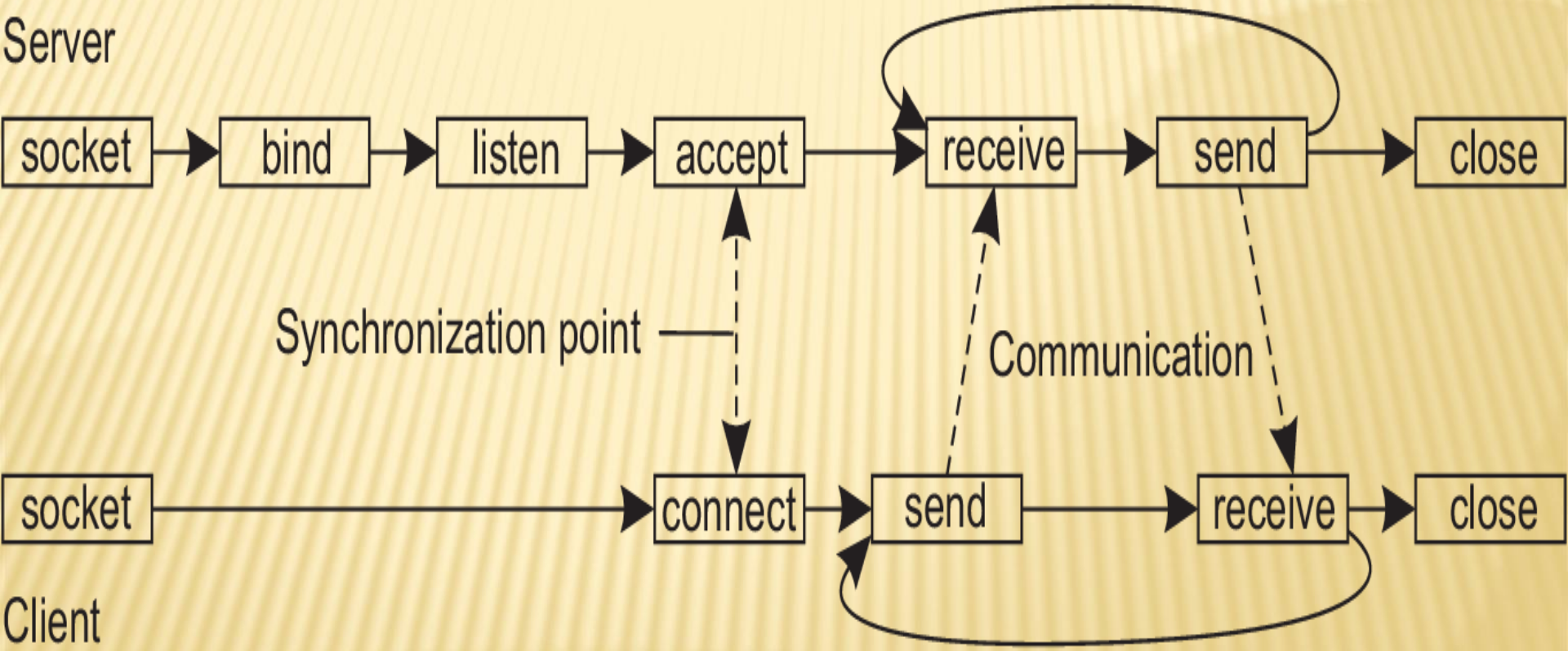
(1) Client must locate server machine, and (2) locate the server.



Transient Messaging: Sockets

Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Tell operating system what the maximum number of pending connection requests should be
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection

Transient Messaging: Sockets



Making Sockets Easier to Work with

Observation

Sockets are rather low level and programming mistakes are easily made. However, the way that they are used is often the same (such as in a client-server setting).

Alternative: ZeroMQ and MPI

Provides a higher level of expression by pairing sockets: one for sending messages at process P and a corresponding one at process Q for receiving messages. All communication is asynchronous.

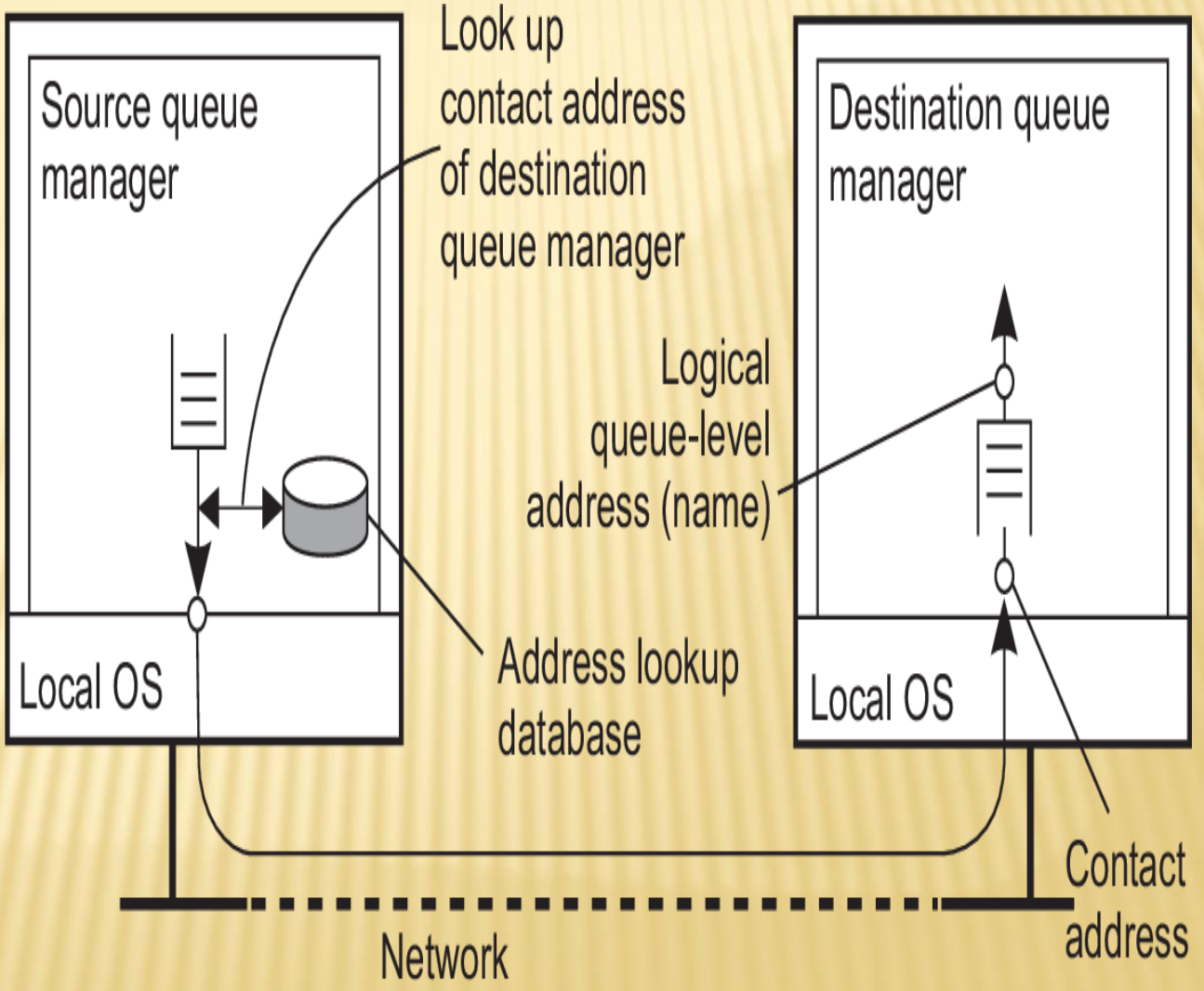
Three patterns

- Request-reply
- Publish-subscribe
- Pipeline

Message-oriented middleware

Essence

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



Message Broker

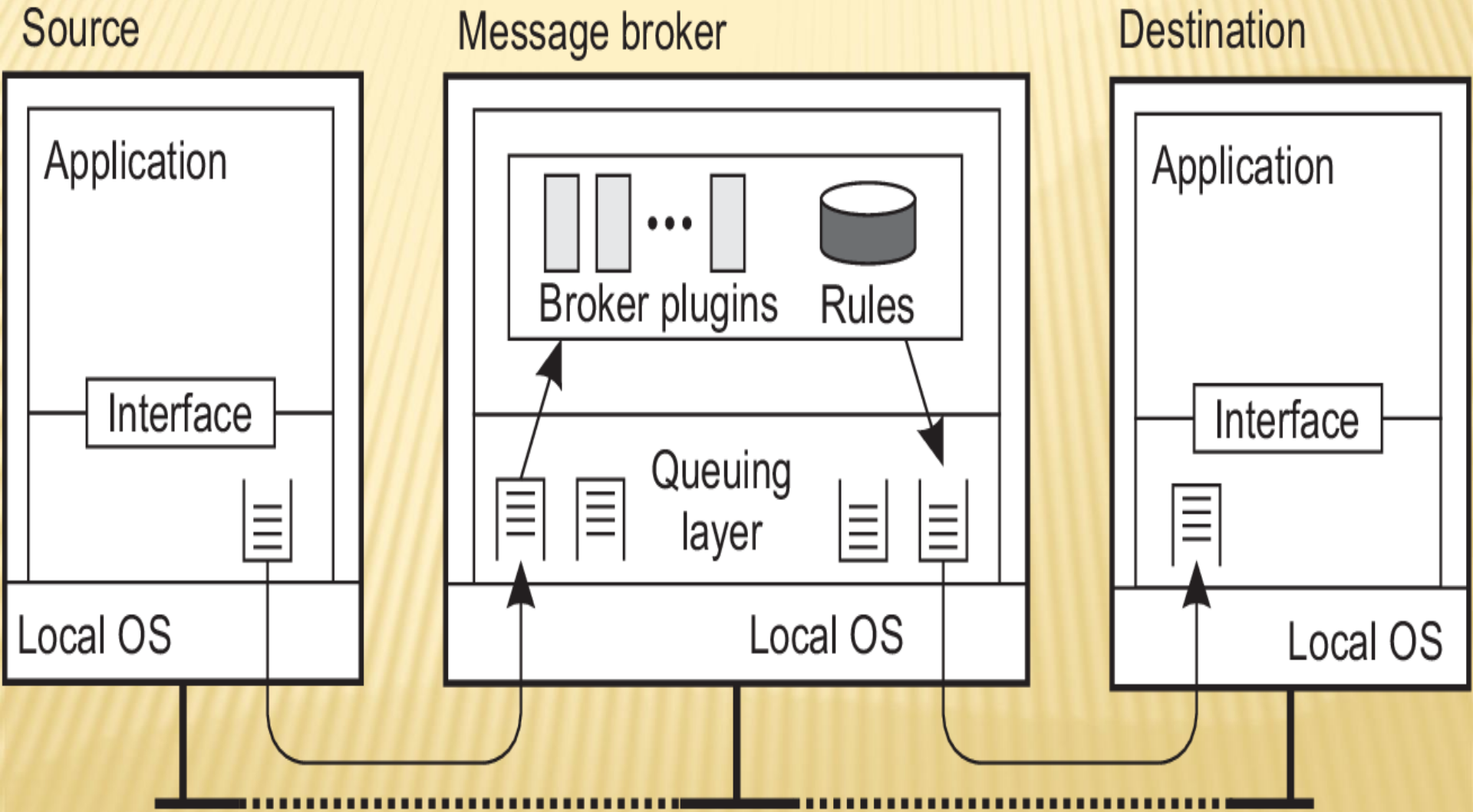
Observation

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



Application-Level Multicasting

Essence

Organize nodes of a distributed system into an overlay network and use that network to disseminate data:

- Oftentimes a tree, leading to unique paths
- Alternatively, also mesh networks, requiring a form of routing. Flooding.