



Systemy rozproszone

Informatyka, sem. 6
część II



Communication

na podstawie:

A.S. Tanenbaum, M. van Steen

Distributed Systems Principles and Paradigms

z modyfikacjami P. Kaczmarek



Middleware communication protocols



- Middleware
 - an application that logically lives in the application layer (ISO-OSI), but which contains many general-purpose protocols that warrant their own layers, independent of other, more specific applications
 - authentication protocols, commit, distributed locking, reliable multicast
- Four high-level middleware communication protocols
 - RPC, RMI, MQ services, stream-oriented

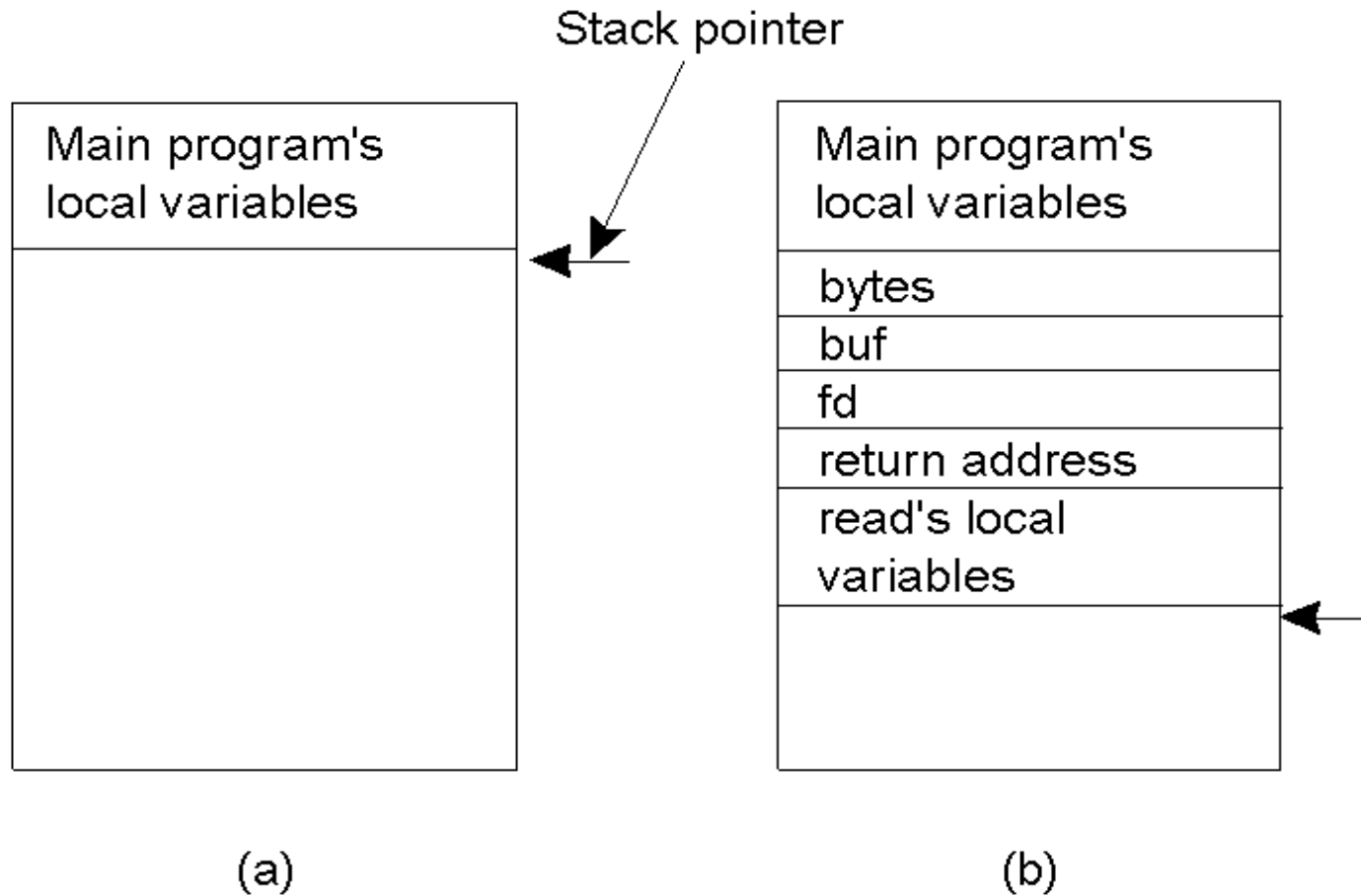


Remote Procedure Call

- The procedures "send" and "receive" do not conceal communication
- Solution: call procedures located on other machines
 - no message passing is visible to the programmer
- Basic idea is refreshingly simple, but
 - procedures execute in different name spaces
 - parameters and results have to be passed
 - both machines can crash



Conventional Procedure Call



- `count = read (fd, buf, nbytes);`



Conventional Procedure Call (2)

- a)Parameter passing in a local procedure call: the stack before the call to read
- b)The stack while the called procedure is active

Call types: call-by-value, call-by-reference, call-by-copy/restore (not in C)

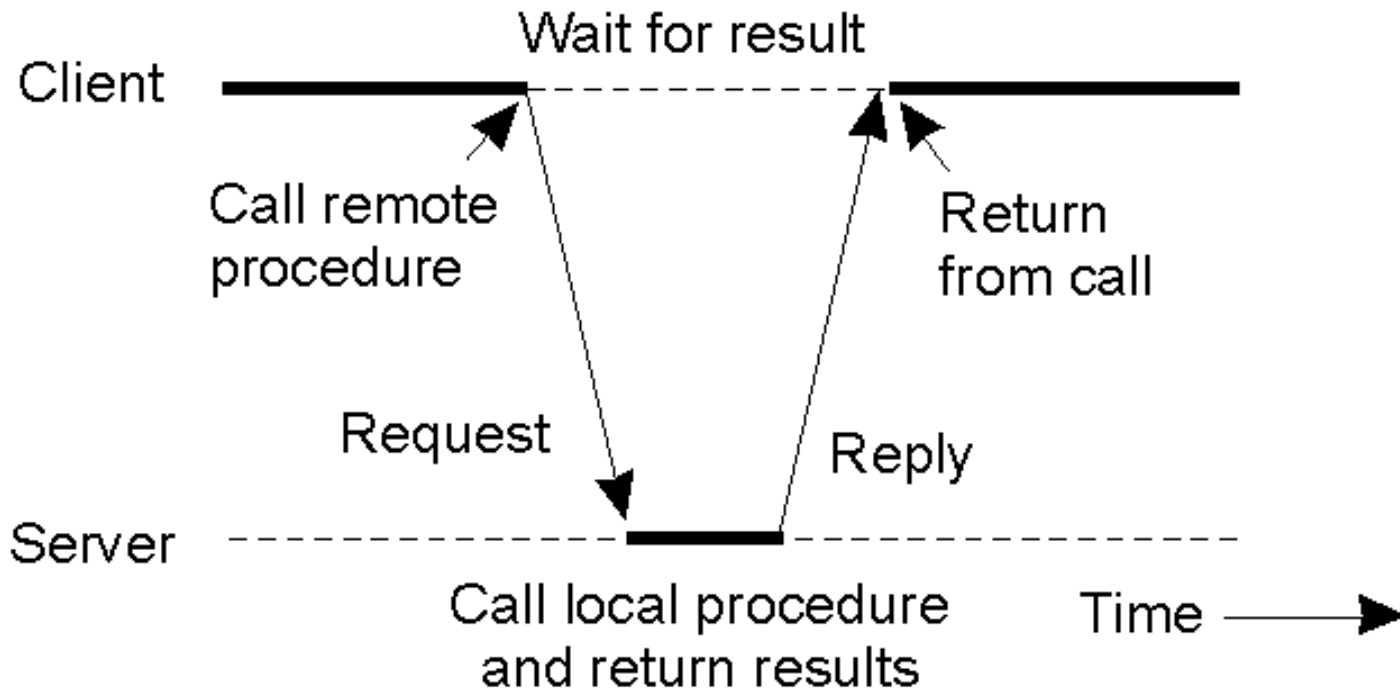


Client and Server Stubs

- We want RPC to be transparent
- Traditional call
 - prepare parameters, invoke a system call (e.g. read)
 - the method is extracted from the library, method internals are hidden from programmer
- RPC call - an analogous way
 - a client stub is invoked by the program
 - the stub calls the operating system, packs parameters and sends message to the server (server stub)



Client and Server Stubs



- Principle of RPC between a client and server program.
- RPC achieves transparency by creating client and server stubs
- (like calls to library programs on single-processor system)



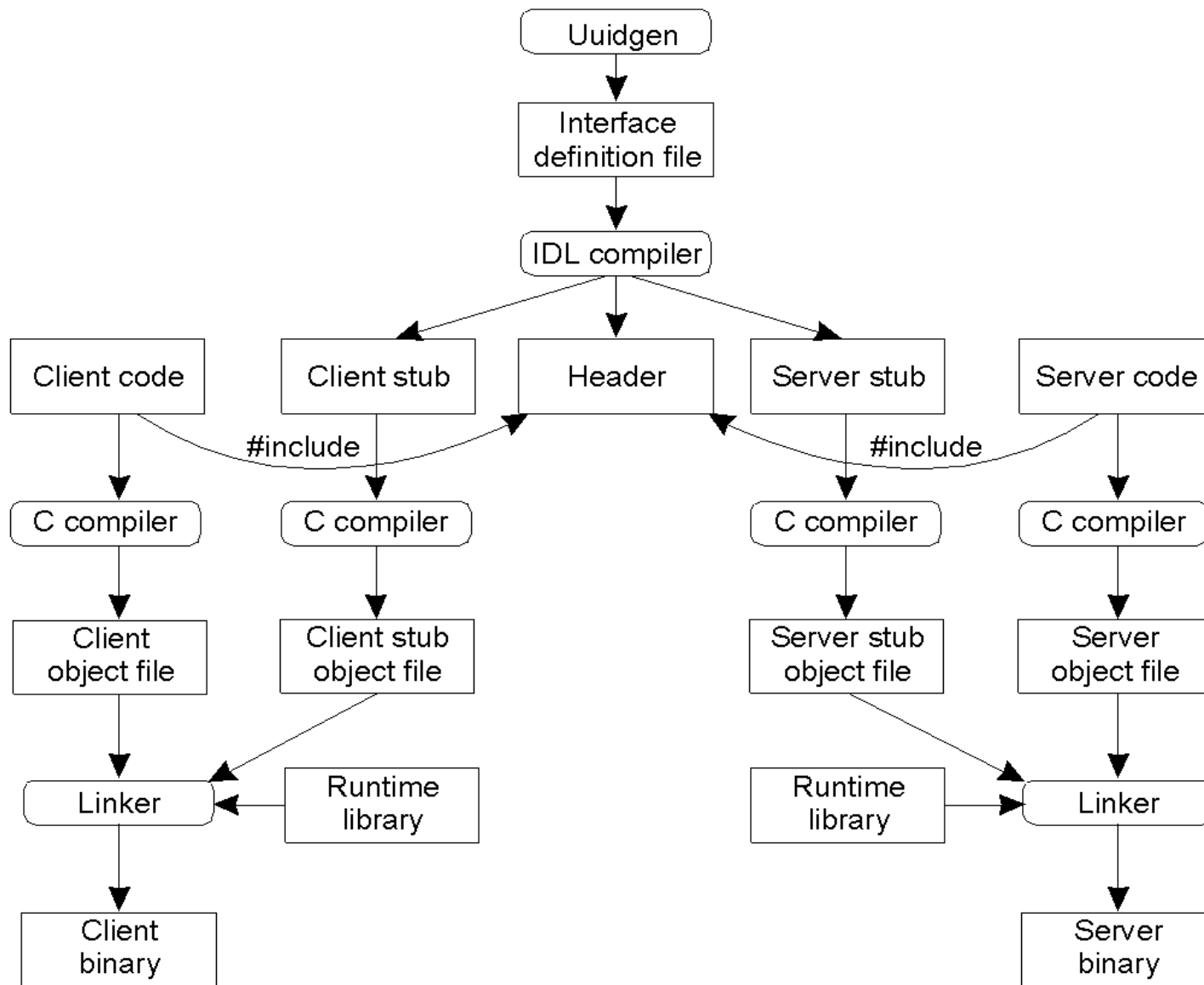
Steps of a Remote Procedure Call



1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client



Writing a Client and a Server





Writing a Client and a Server (2)



- Interface definition language
 - procedure definitions (a form close to ANSI C)
- A global unique identifier for the specified interface
 - the right server, the right version
- Compile IDL
 - a header file - unique identifier, types, constants, function prototypes
 - the client stub - actual procedures that the client program will call
 - the server stub - procedures called by the runtime system (when a message arrives)



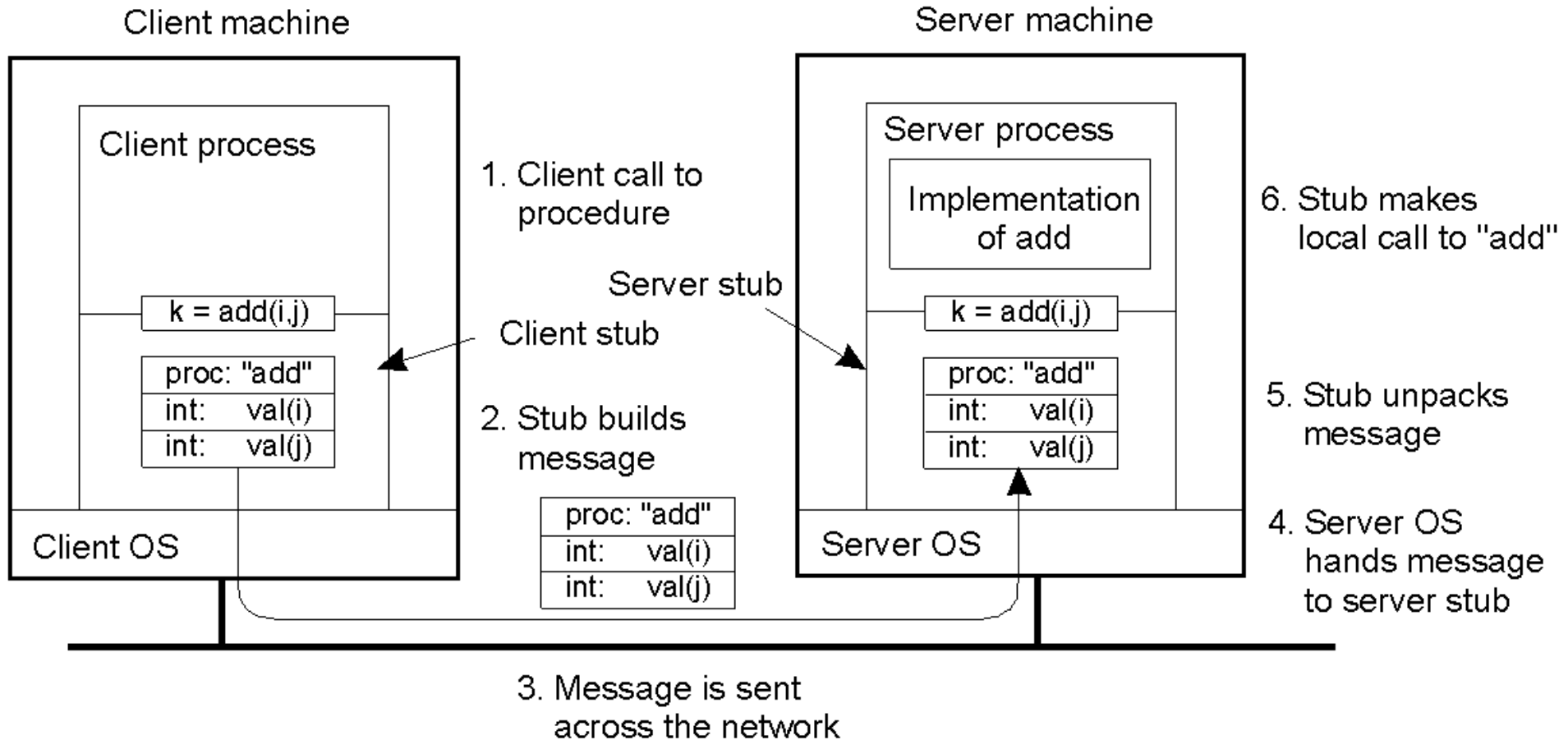
Writing a Client and a Server (3)



- Write client and server functions
- Server registers in the runtime system as an RPC server
- Client invokes server procedures, needs to know an endpoint
 - locate the server's machine
 - locate the server (the correct process) on that machine
- Performing an RPC
 - at-most-once / idempotent



Passing Value Parameters (1)



- Steps involved in doing remote computation through RPC



Passing Value Parameters (2)

- Parameter marshalling - packing parameters into a message
- Client side:
 - parameters, procedure number or name
- Server side:
 - stub identifies procedure, unpacks parameters, invokes procedure
- Difficulties:
 - different types, representations, character codes (EBCDIC / ASCII), big / small endian



Passing Value Parameters (2)

3	2	1	0
0	0	0	5
7	6	5	4
L	L	I	J

(a)

0	1	2	3
5	0	0	0
4	5	6	7
J	I	L	L

(b)

0	1	2	3
0	0	0	5
4	5	6	7
L	L	I	J

(c)

a) Original message on the Pentium

b) The message after receipt on the SPARC

c) The message after being inverted. The little numbers in boxes indicate the address of each byte

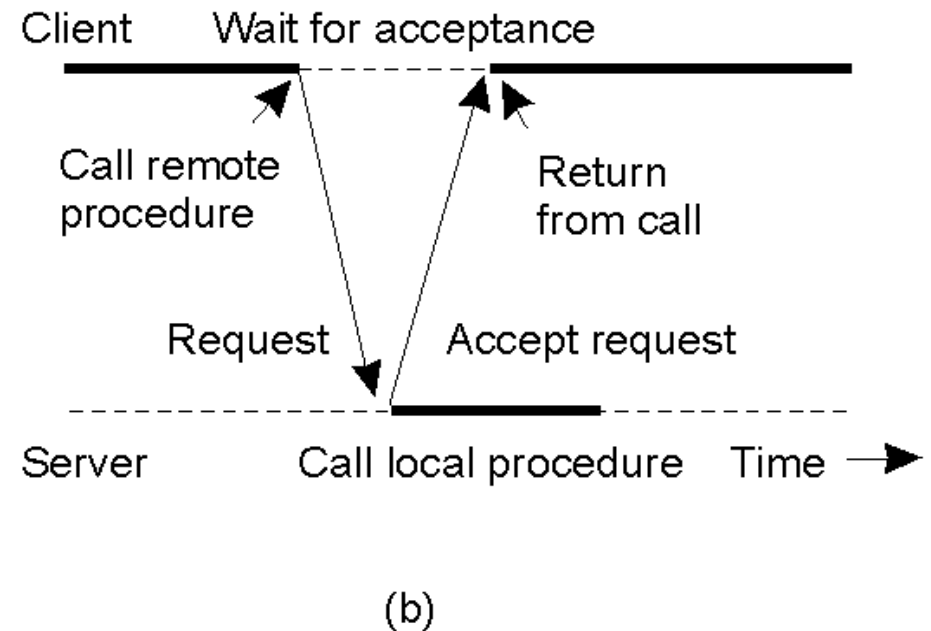
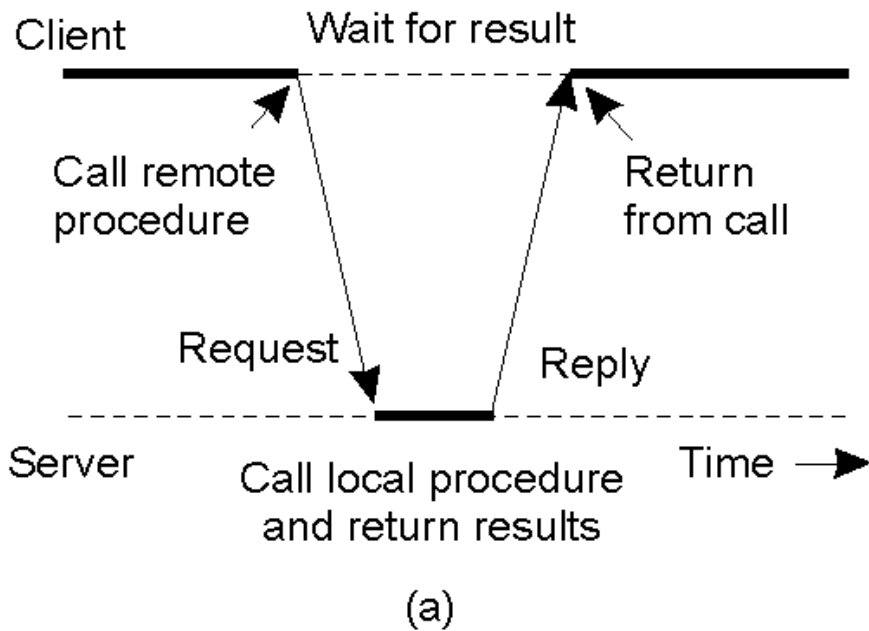


Passing Reference Parameters

- Only with greatest difficulty
- Alternative solutions
 - Forbid pointers and reference parameters - difficult to accept by developers
 - Copy an array into the message and send it (send back when server finishes)
 - Optimize for input / output arrays



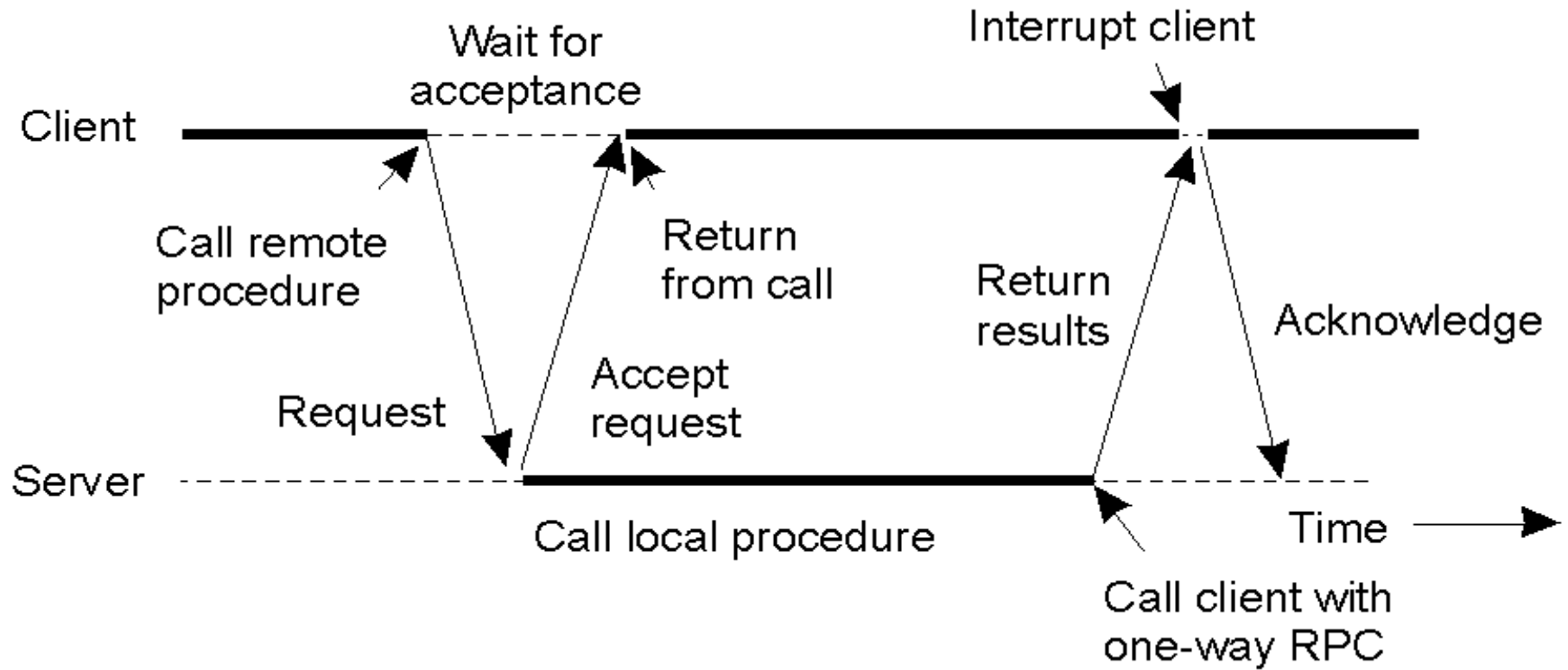
Asynchronous RPC (1)



- a) The interconnection between client and server in a traditional RPC
- b) The interaction using asynchronous RPC- a client immediately continues after issuing the RPC request



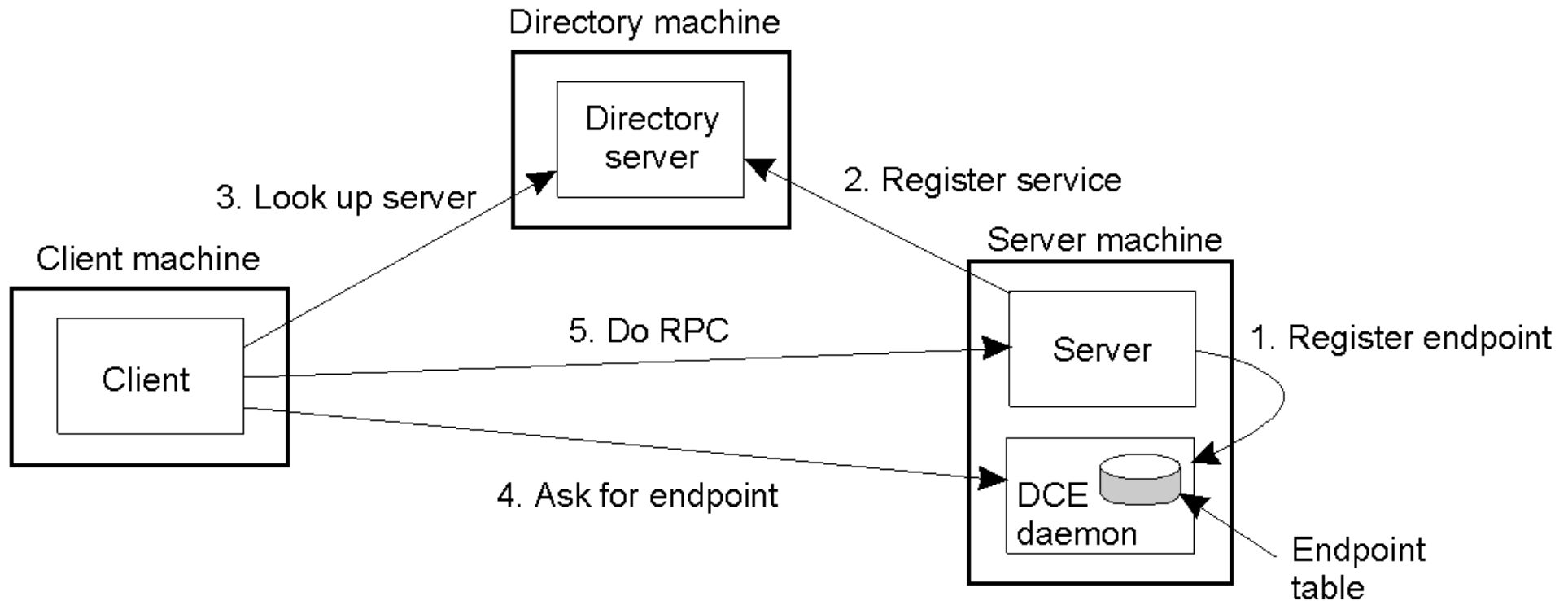
Asynchronous RPC (2)



- A client and server interacting through two asynchronous RPCs - deferred synchronous RPC



Binding a Client to a Server



- Client-to-server binding in DCE RPC



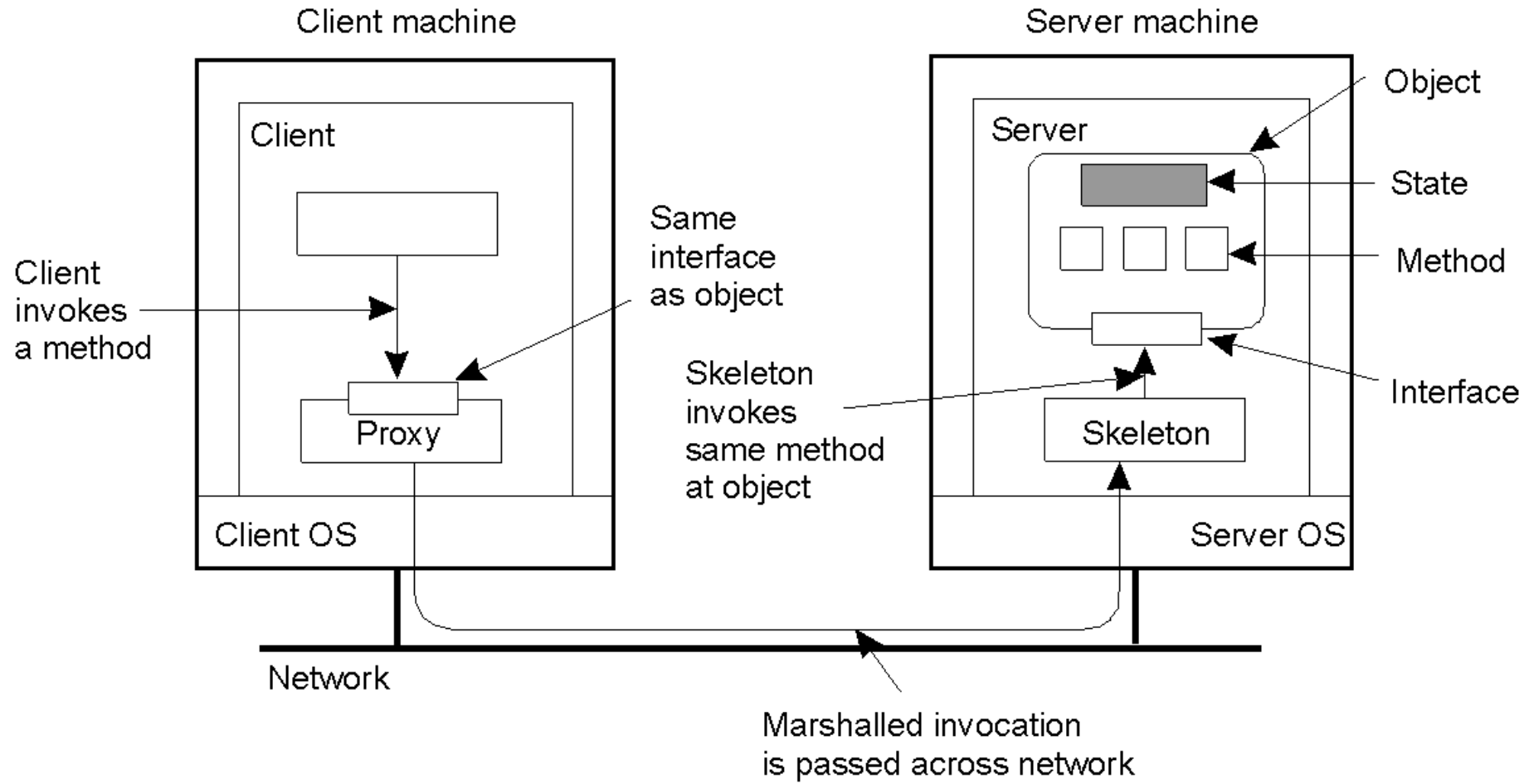
Remote Method (Object) Invocation



- RPC model applied to objects
- Object
 - encapsulates data (called state) and the operations on those data (called the methods), methods are made available through an interface
- Distributed objects: an interface at one machine and object itself on another
- A client binds to a distributed object, an implementation of the object's interface, called a proxy, is loaded into the client's address space - analogous to a client stub in RPC
- Server stub (skeleton) processes requests - server side
- Examples: CORBA, DCOM, Java RMI



Distributed Objects



- Common organization of a remote object with client-side proxy.



Persistent and Transient Objects



- A persistent object
 - continues to exist even if it is currently not contained in the address space of a server process
 - a server can store object's state on secondary storage and exit, a newly started server can read the state in its own address space
- A transient object
 - exists as long as the server manages the object



Binding a Client to an Object

```
Distr_object* obj_ref;           //Declare a systemwide object reference
obj_ref = ...;                   // Initialize the reference to
                                 // a distributed object
obj_ref-> do_something();         // Implicitly bind and invoke a method
(a)
```

```
Distr_object objPref;           //Declare a systemwide object reference
Local_object* obj_ptr;          //Declare a pointer to local objects
obj_ref = ...;                  //Initialize the reference to a distributed object
obj_ptr = bind(obj_ref);        //Explicitly bind and obtain a pointer to
                                 // the local proxy
obj_ptr -> do_something();       //Invoke a method on the local proxy
(b)
```

(a) Example with implicit binding using only global references

(b) Example with explicit binding using global and local references

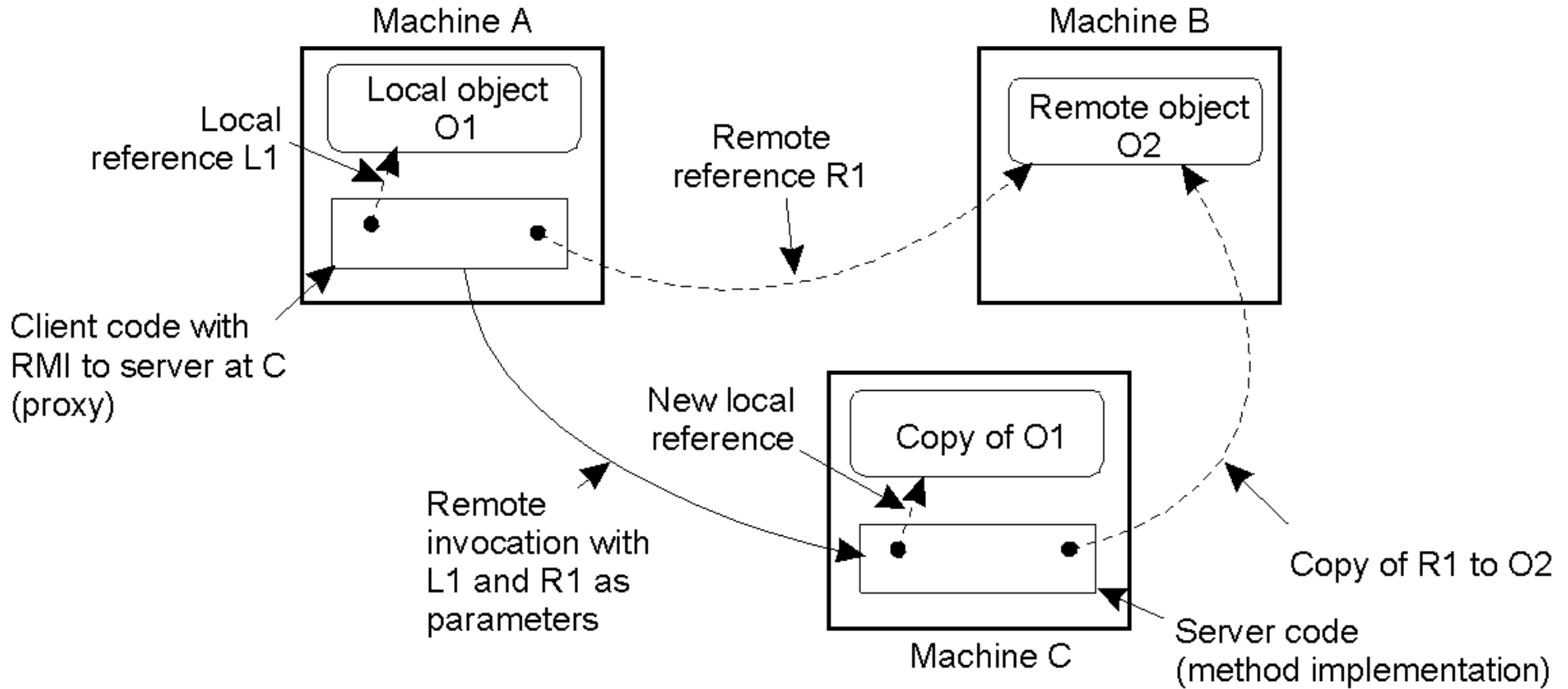


Static and Dynamic RMI

- Static invocation
 - `object.method(input_parameter)`
- Dynamic invocation – selects at runtime which objects will be invoked
 - `invoke (object, method, input_parameters, output_parameters)`
 - for example appending integer to a file
 - `fobject.append(int)`
 - `invoke (fobject, id(append), int)`



Parameter Passing



The situation when passing an object by reference or by value



Java RMI - demo



- Uwagi do demo
 - Parametry programu vs. parametry VM
 - Java security policy
 - Ustawienia firewalla
 - rmiregistry / CLASSPATH



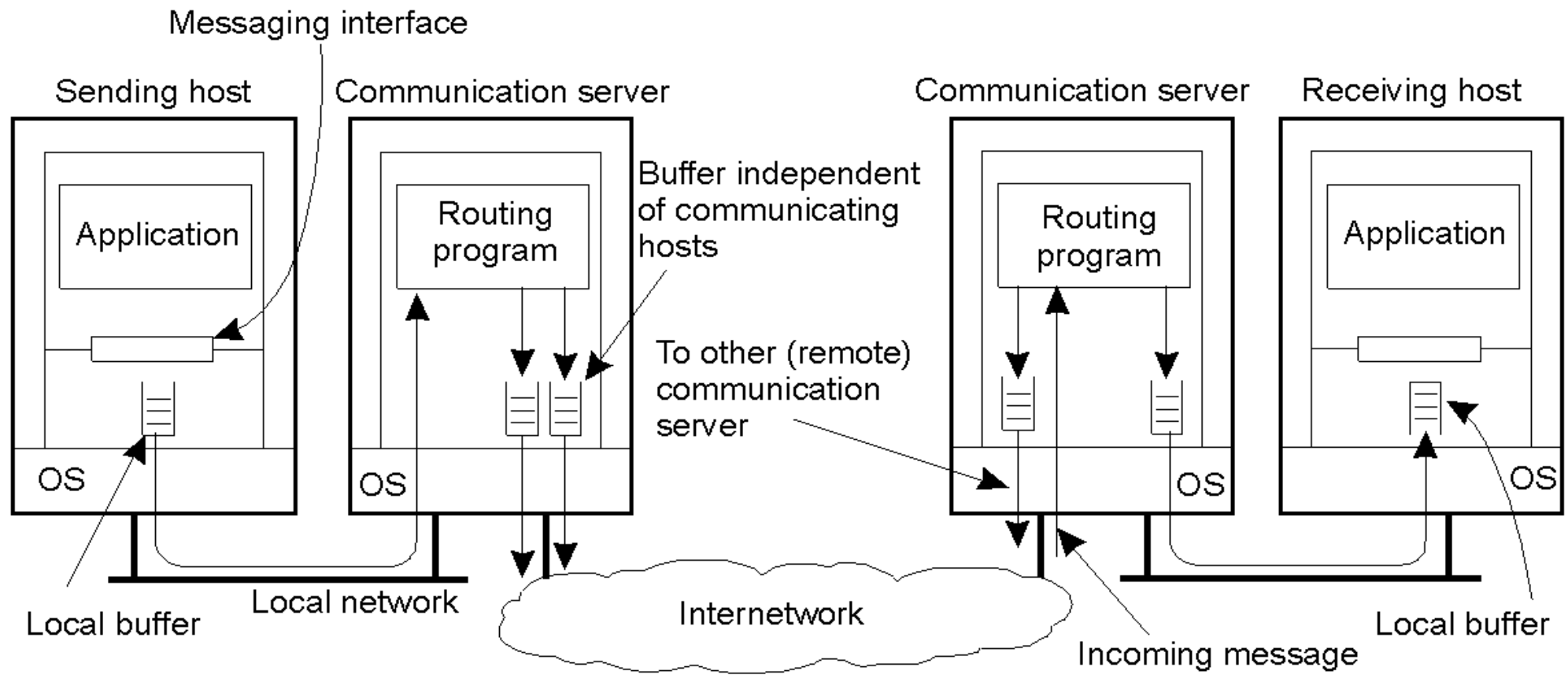
Message-Oriented Communication



- RPC / RMI hide communication, but neither mechanism is always appropriate
 - when it cannot be assumed that the receiving side is executing at the time a request is issued
- Alternative solution: messaging
 - message-queuing systems: allow processes to exchange information, even if the other party is not executing at the time communication is issued



Persistence and Synchronicity in Communication (1)



General organization of a communication system in which hosts are connected through a network

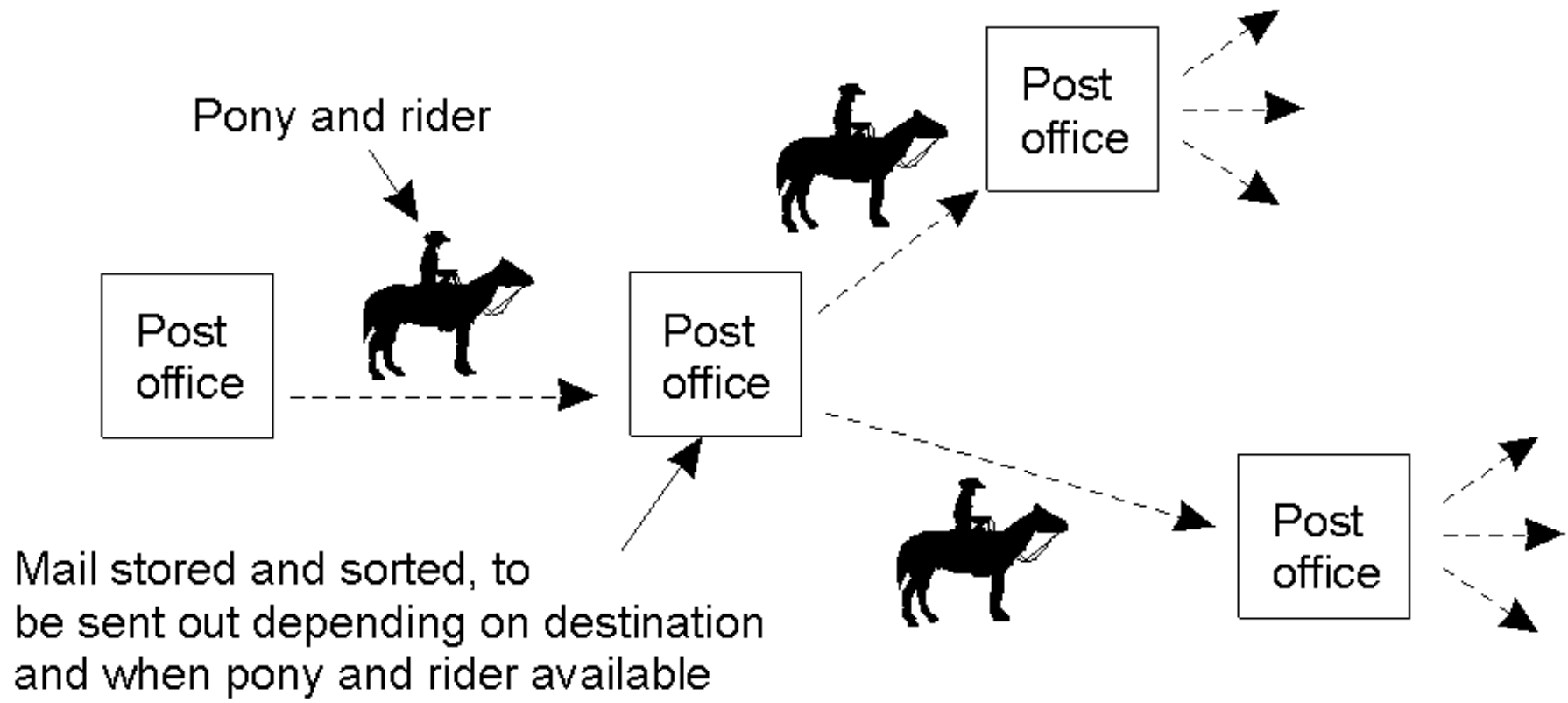


Persistence and Synchronicity in Communication

- Persistent communication - a message that has been submitted for transmission is stored by the communication system as long as it takes to deliver it to the receiver
- Transient communication - a message is stored by the communication system only as long as the sending and receiving applications are executing
- Asynchronous communication - a sender continues immediately after it has submitted its message for transmission
- Synchronous communication - a sender is blocked until its message is stored in a local buffer at the receiving host, or actually delivered to the receiver.



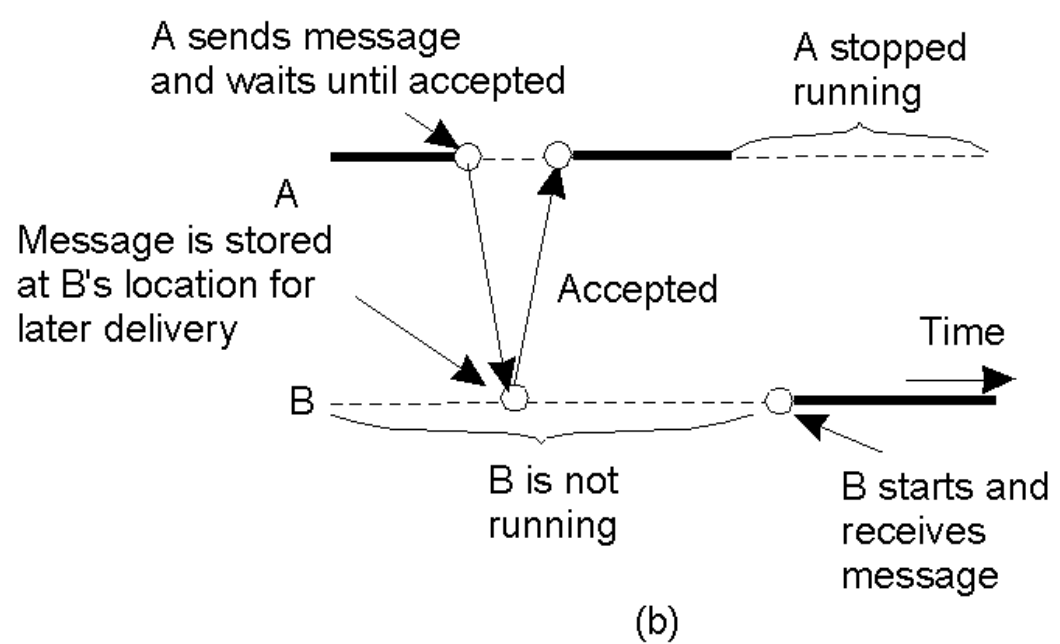
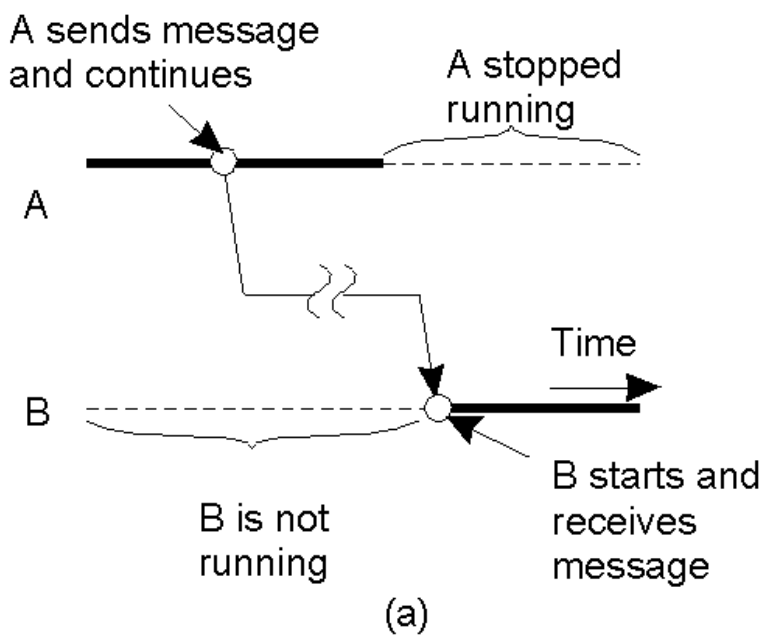
Persistence and Synchronicity in Communication (2)



Persistent communication of letters back in the days of the PonyExpress



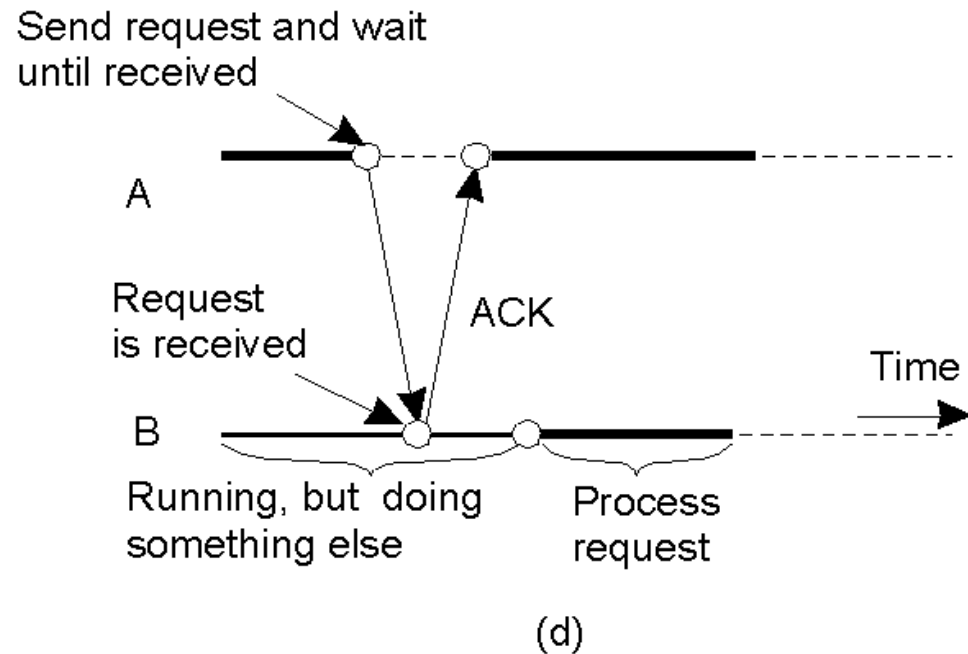
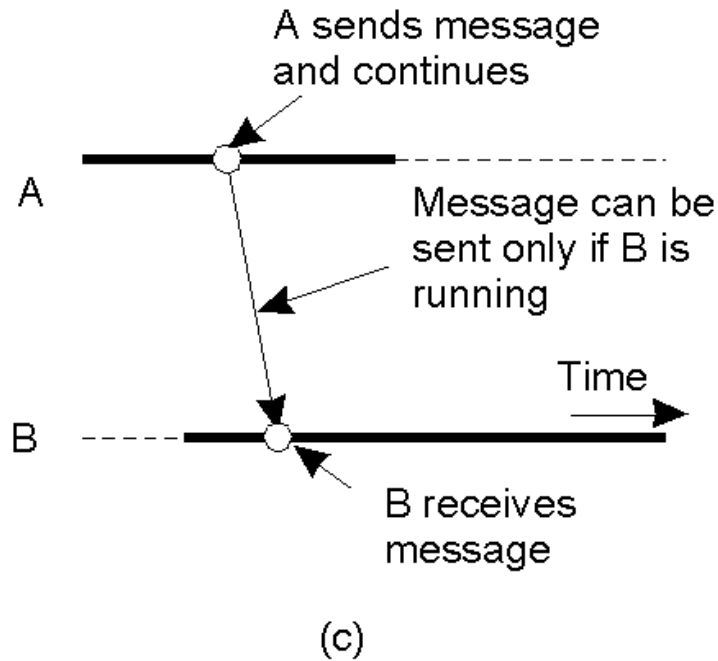
Persistence and Synchronicity in Communication (3)



- a) Persistent asynchronous communication
- b) Persistent synchronous communication



Persistence and Synchronicity in Communication (4)

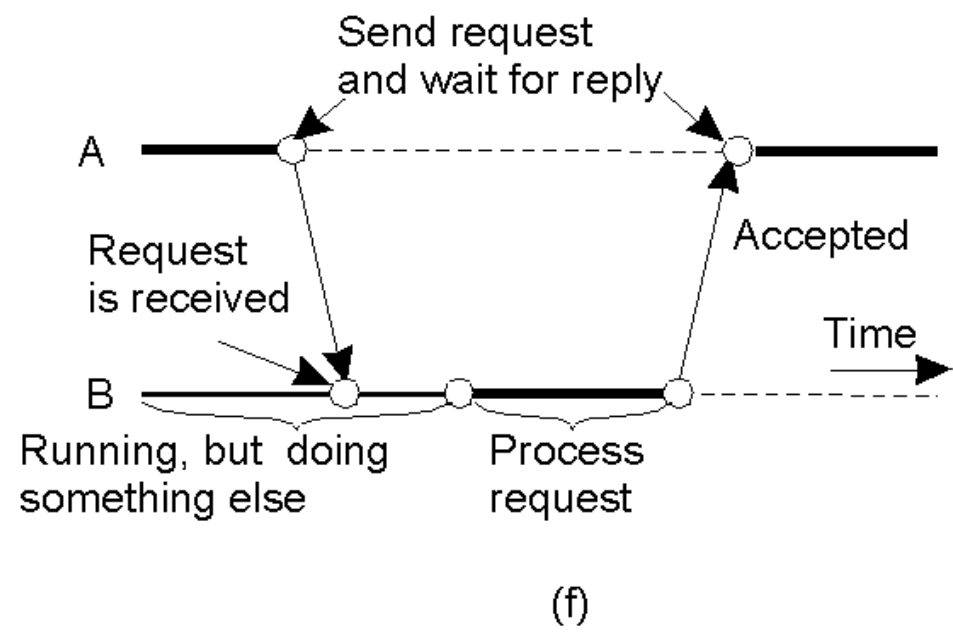
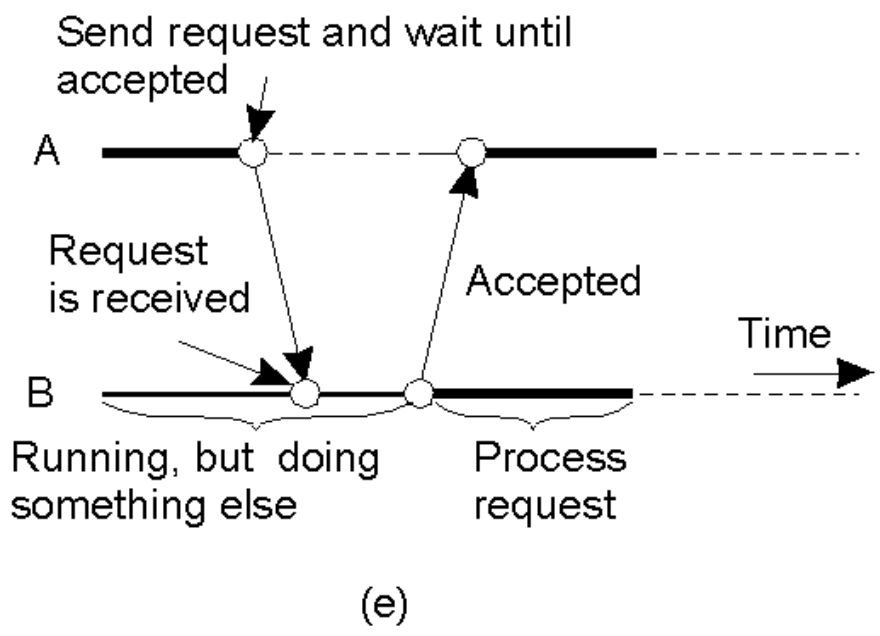


c) Transient asynchronous communication

d) Receipt-based transient synchronous communication



Persistence and Synchronicity in Communication (5)



e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication



Message Oriented Middleware

- Offer intermediate-term storage capacity for messages
 - targeted to support message transfers that are allowed to take minutes instead of sec. or milis.
- Applications communicate by inserting messages in specific queues
 - communication servers (mq systems)
 - private queues of applications
- Loosely coupled applications
 - no guarantees about reading a message by recipient



Java Messaging Service Demo



- Architektura
 - JMS provider (system kolejkowy), klienci JMS (wysyłają i odbierają wiadomości), wiadomości, prekonfigurowane obiekty JMS
- Modele komunikacji
 - point-to-point (queue destination)
 - publish-subscribe (topic destination)



JMS Demo - cd

- Wymagane "jary" z serwera J2EE - img, jms
- Utworzenie kolejki w serwerze
 - uruchomienie serwera: `asadmin start-domain --verbose domain1`
 - `http://localhost:4848/asadmin/`
 - jeżeli używane JNDI: utworzenie kolejki i "factory":
Resources -> JMS resources
- Uruchomienie aplikacji
 - parametry: kolejka
 - typy połączenia: synchroniczne, asynchroniczne

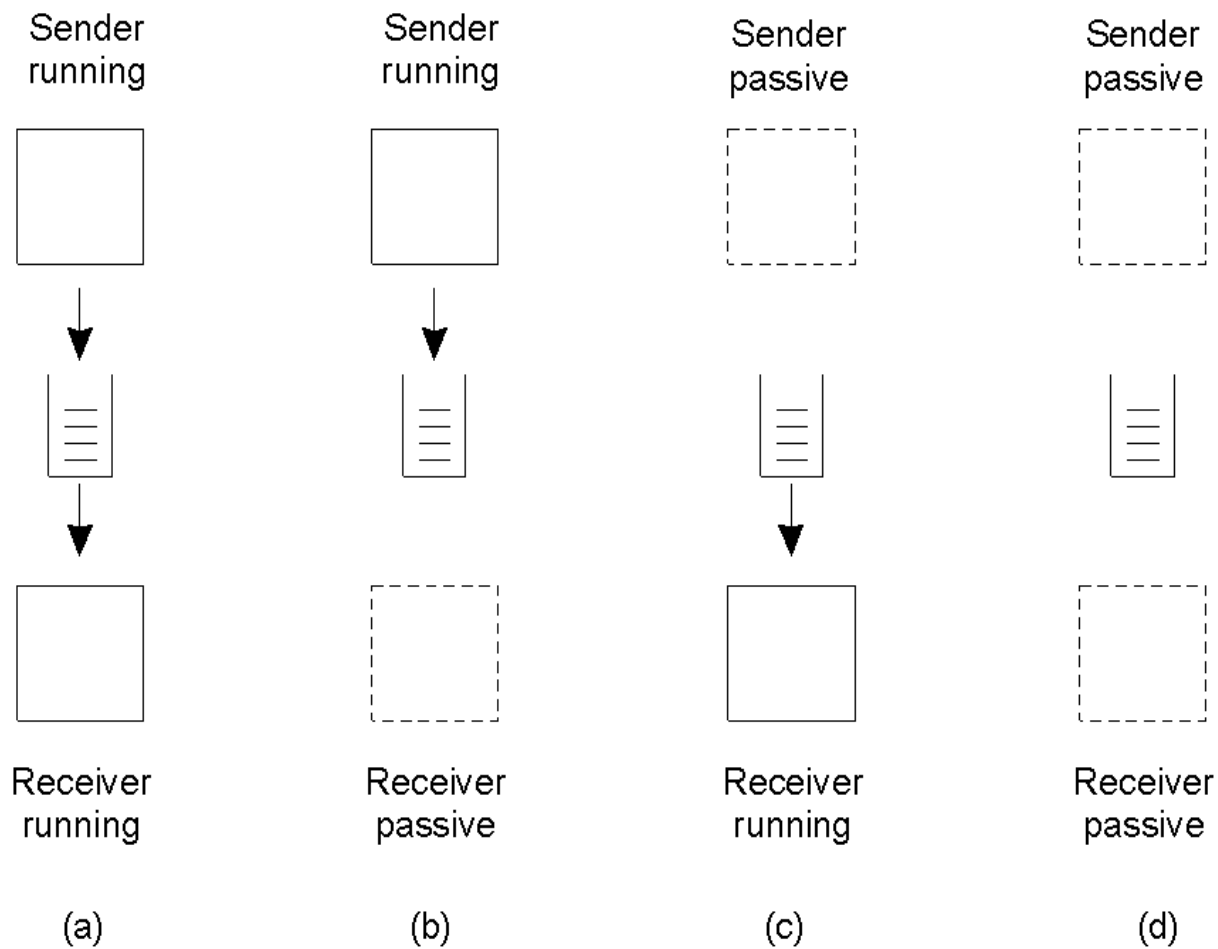


JMS cd

- Producent wiadomości
 - Utworzenie ConnectionFactory lub odnalezienie w JNDI
 - Utworzenie Connection wykorzystując ConnectionFactory
 - Utworzenie Session
 - Odszukanie lub utworzenie obiektu Destination
 - Utworzenie MessageProducer i wysyłanie wiadomości
- Konsument wiadomości
 - Utworzenie MessageConsumer
 - W przypadku komunikacji asynchronicznej utworzenie MessageListener
 - Rozpoczęcie odbierania wiadomości



Message-Queuing Model (1)



- Four combinations for loosely-coupled communications using queues.



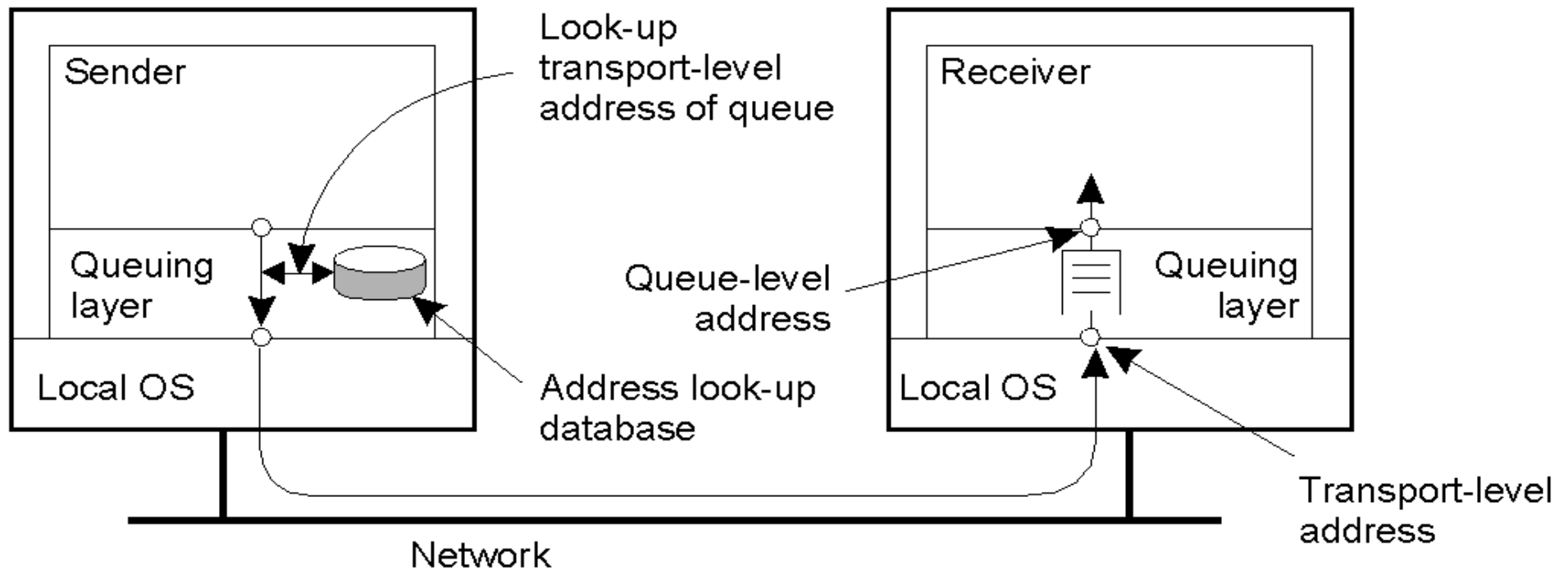
Message-Queuing Model (2)

Primitive	Meaning
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.

- Basic interface to a queue in a message-queuing system.
- Usually, a callback function is called whenever a message is put into the queue



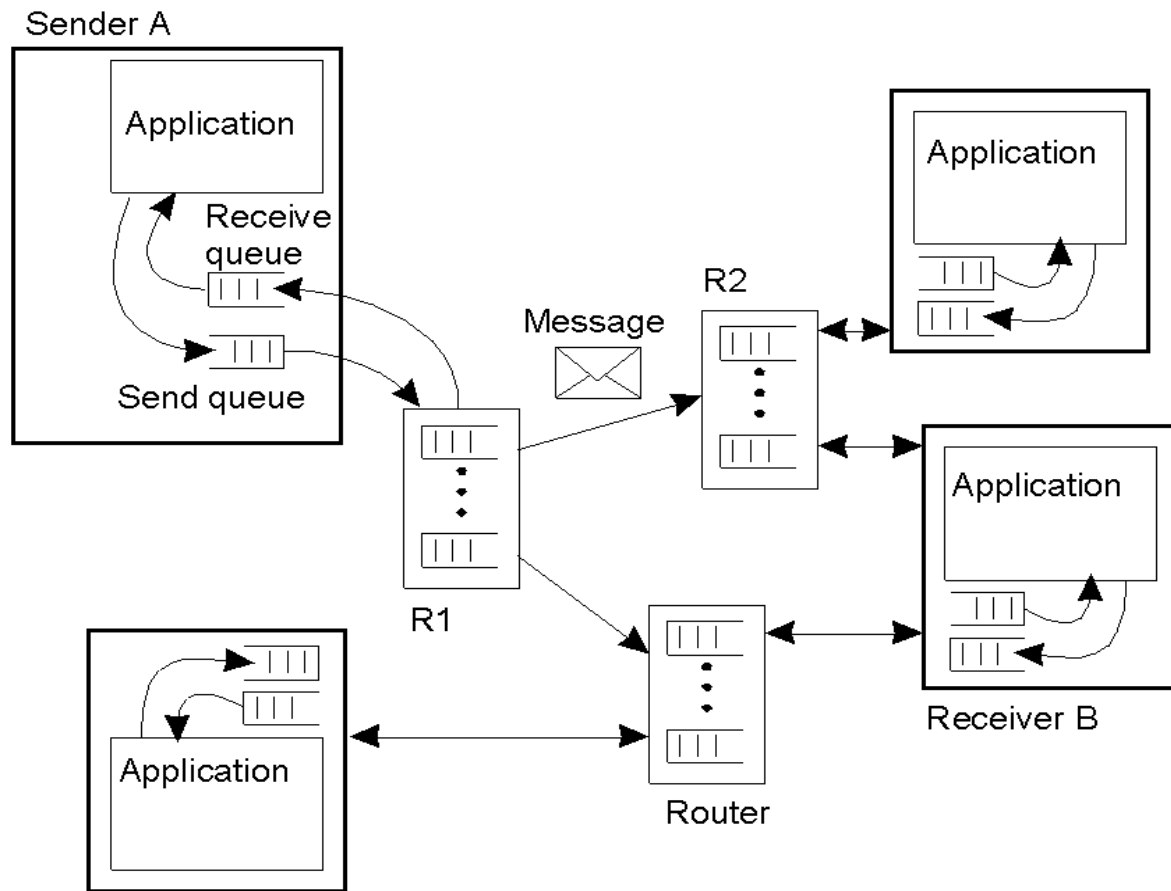
General Architecture of a Message-Queuing System (1)



- The relationship between queue-level addressing and network-level addressing.
- Queues: source, destination, queue names



General Architecture of a Message-Queuing System (2)



- The general organization of a message-queuing system with routers.
- Queues are managed by queue managers (special: relays, overlay)

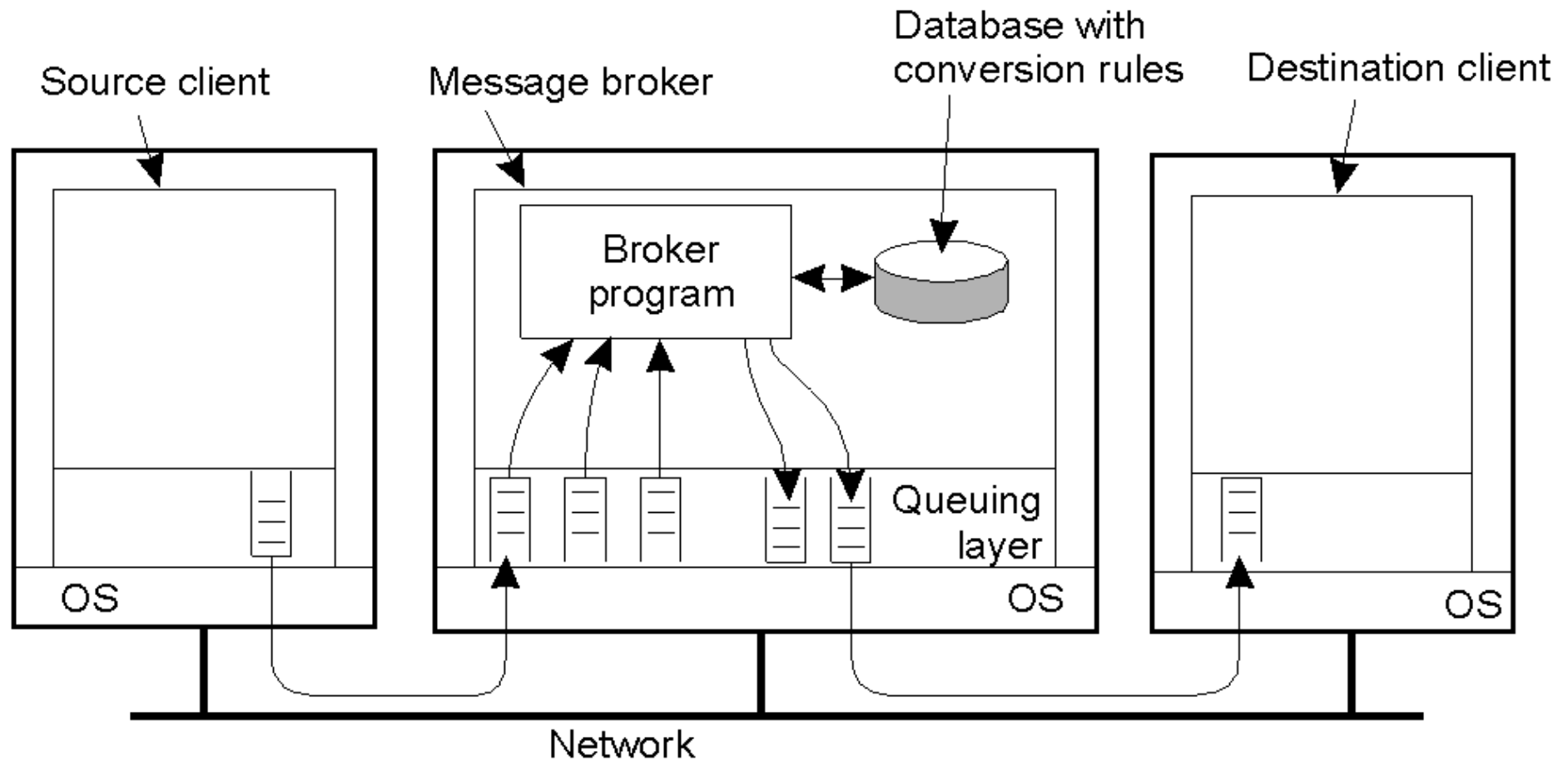


Message Brokers

- Integrate existing and new applications
 - message format incompatibilities
 - either adjust receivers or agree on a common message format - both difficult
- Message brokers
 - special nodes that handle conversion between different formats
 - simple reformat, application level gateway
 - a database of rules for conversion



Message Brokers



- The general organization of a message broker in an mq system.



Stream-Oriented Communication



- Support for time dependent information
 - continuous media
 - the temporal relationships between different data items are fundamental to correctly interpreting what the data actually means.
 - discrete media
- Data stream
 - a sequence of data units



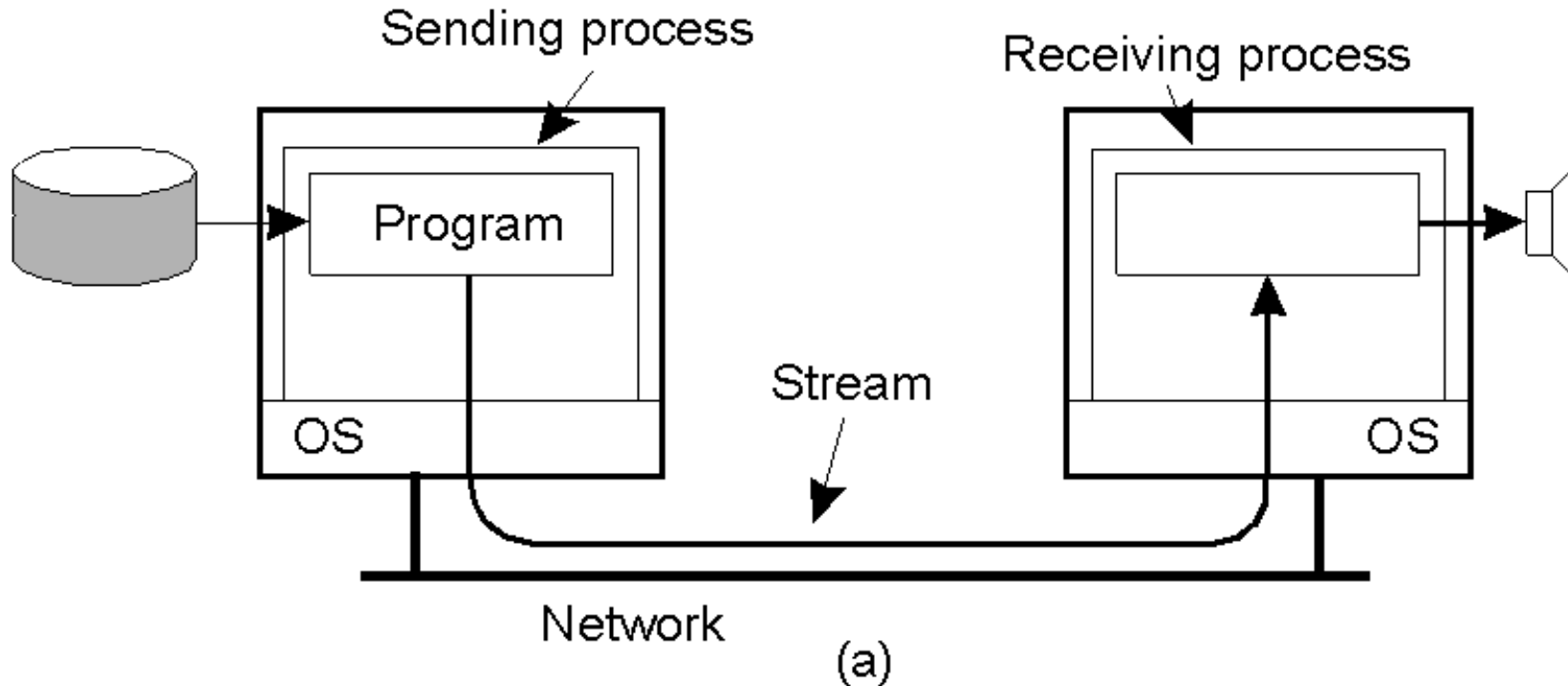
Stream-Oriented Communication cont



- Transmission modes
 - asynchronous - data items transmitted one after another, no further timing constraints
 - synchronous - a maximum end-to-end delay defined for each unit
 - isochronous - on time, maximum and minimum end-to-end delays
- Simple / complex streams
 - substreams



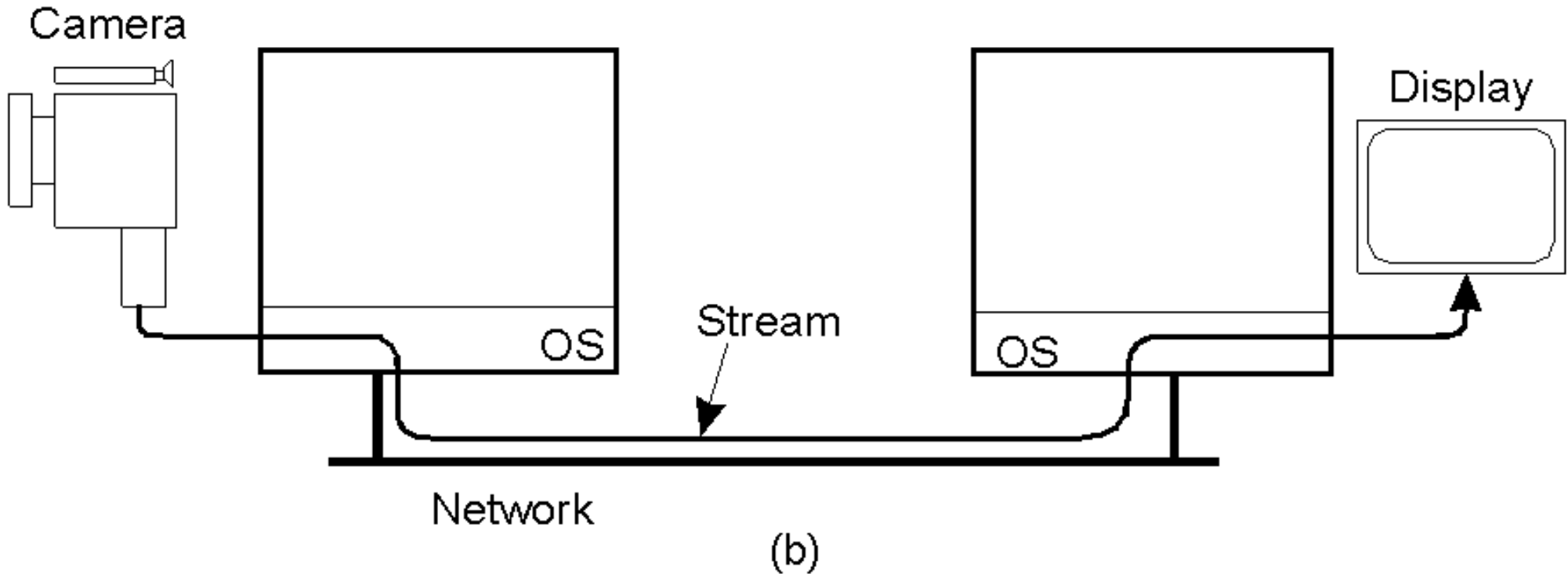
Data Stream (1)



- Setting up a stream between two processes across a network.



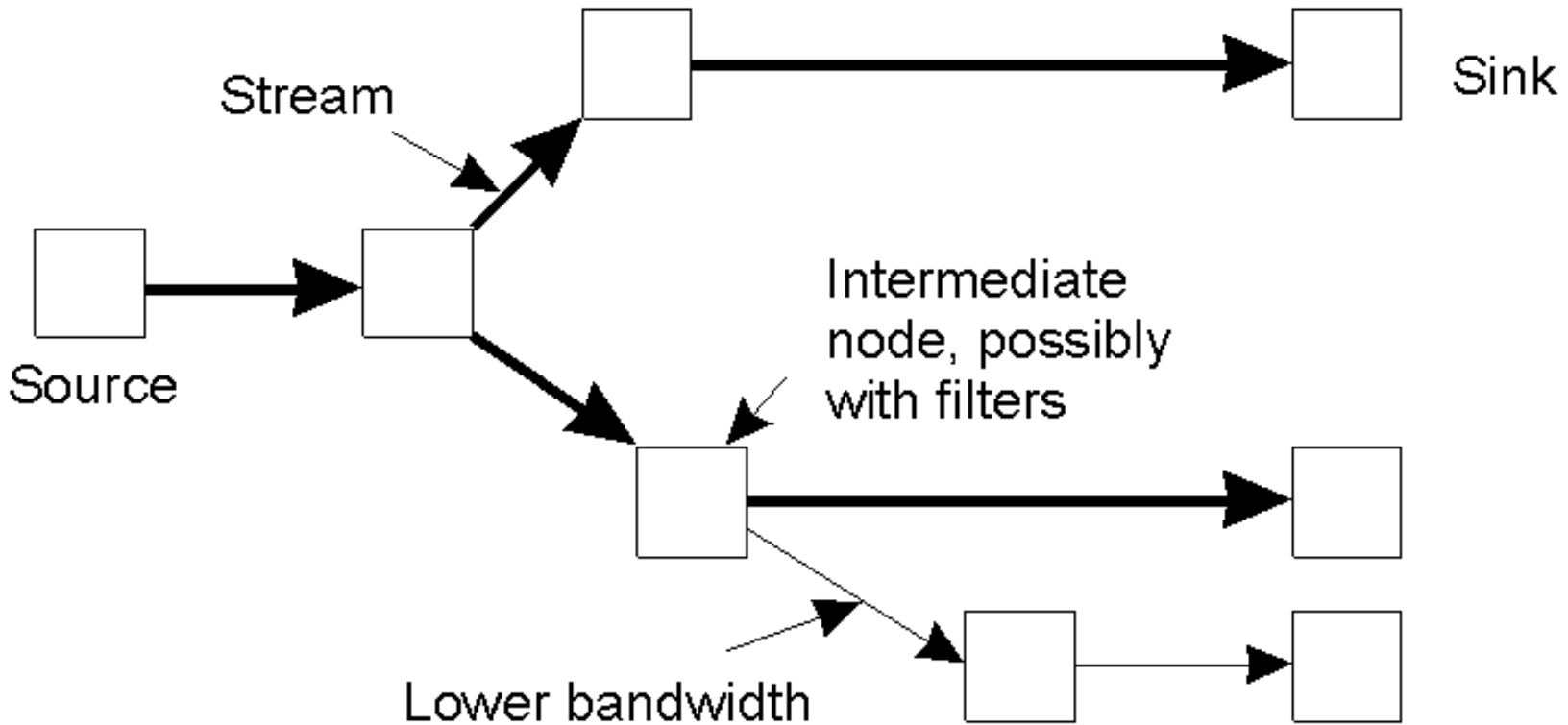
Data Stream (2)



- Setting up a stream directly between two devices.



Data Stream (3)



- An example of multicasting a stream to several receivers.