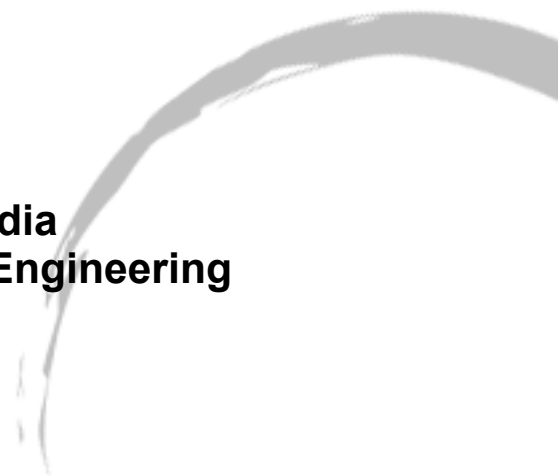




# Communication

**Prof. Chi-Sheng Shih**  
**Graduate Institute of Network and Multimedia**  
**Department of Computer Science and Information Engineering**  
**National Taiwan University**

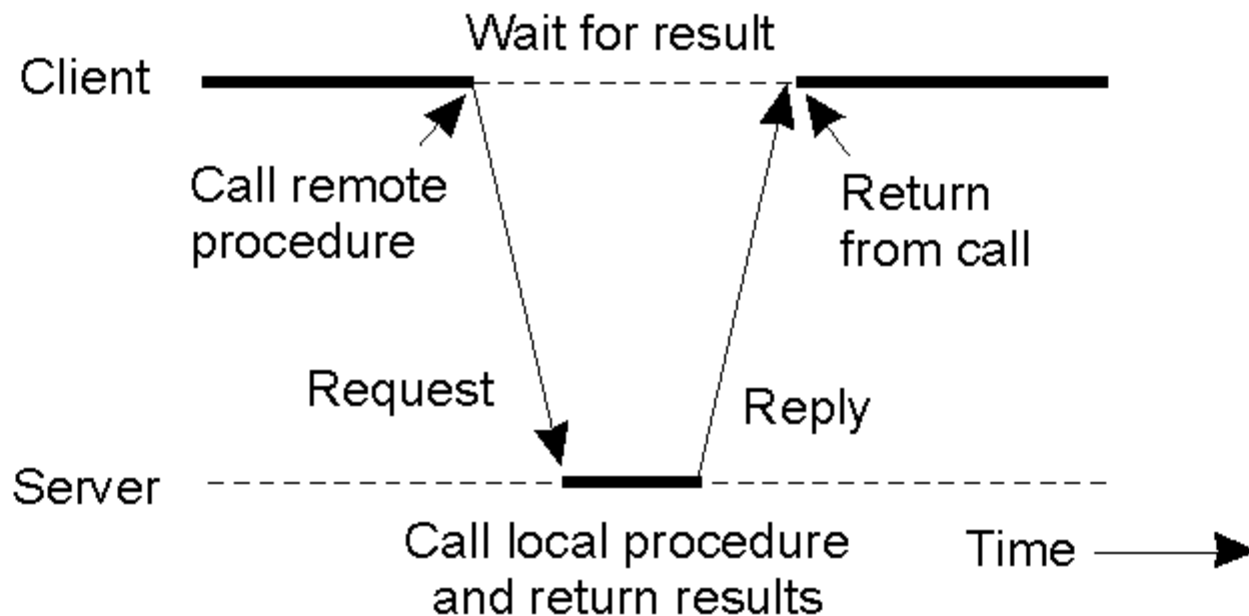


# Topics to be Covered

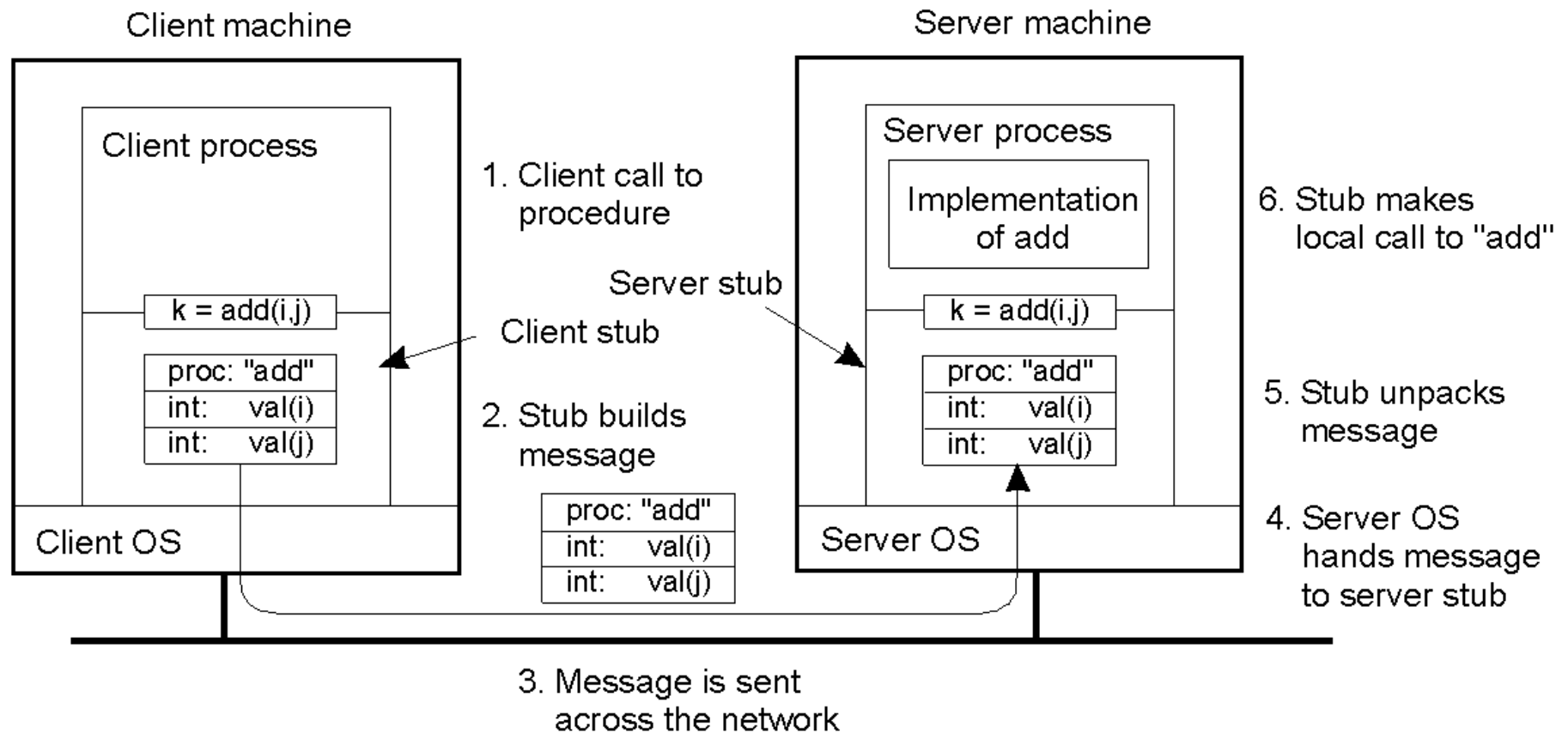
- Remote Procedure Calls
- Message Passing
- Remote Object Invocation
- Message-Oriented Communication
- Stream-Oriented Communication

# Remote Procedure Call

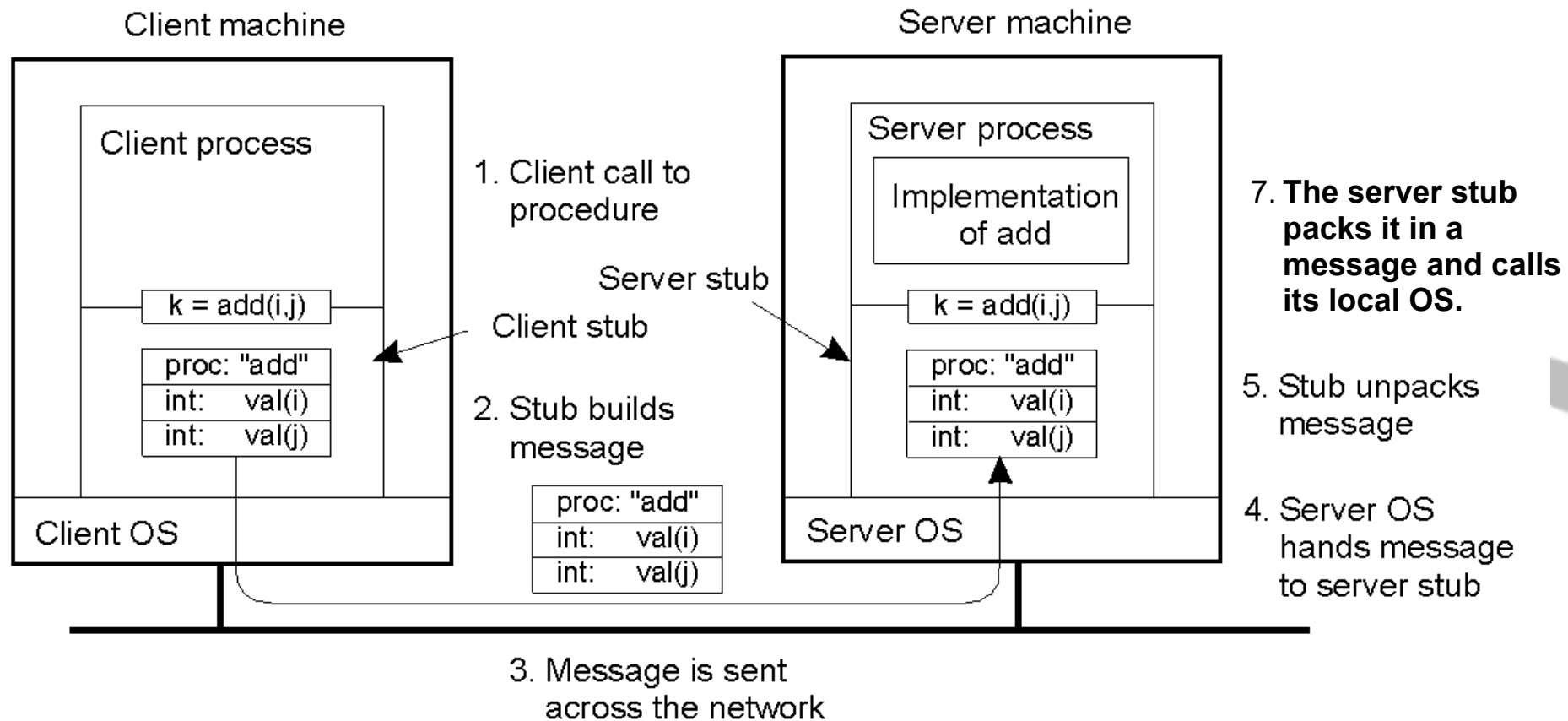
- The caller and called procedures are located on different processors/machines.
- No message passing is visible to the programmers.



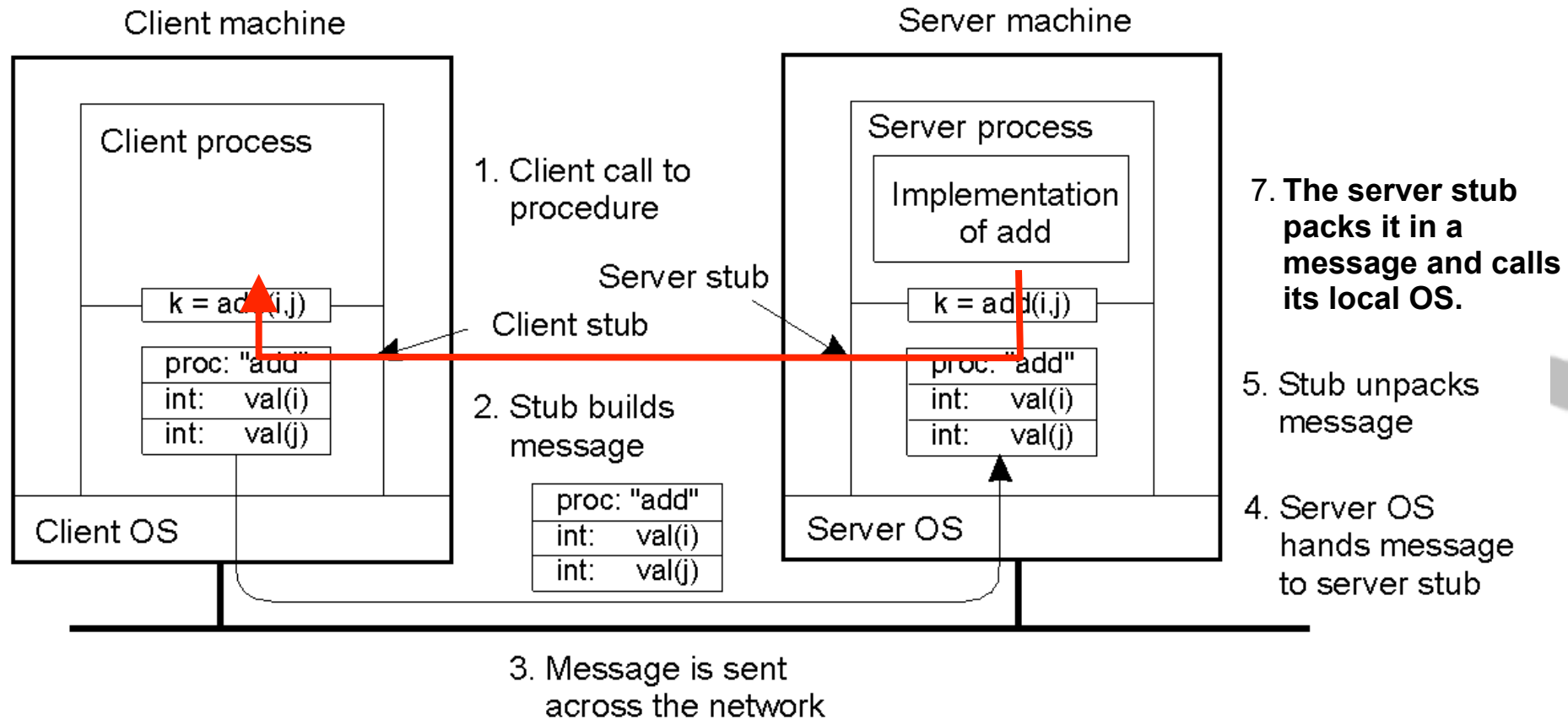
# Steps of a Remote Procedure Call



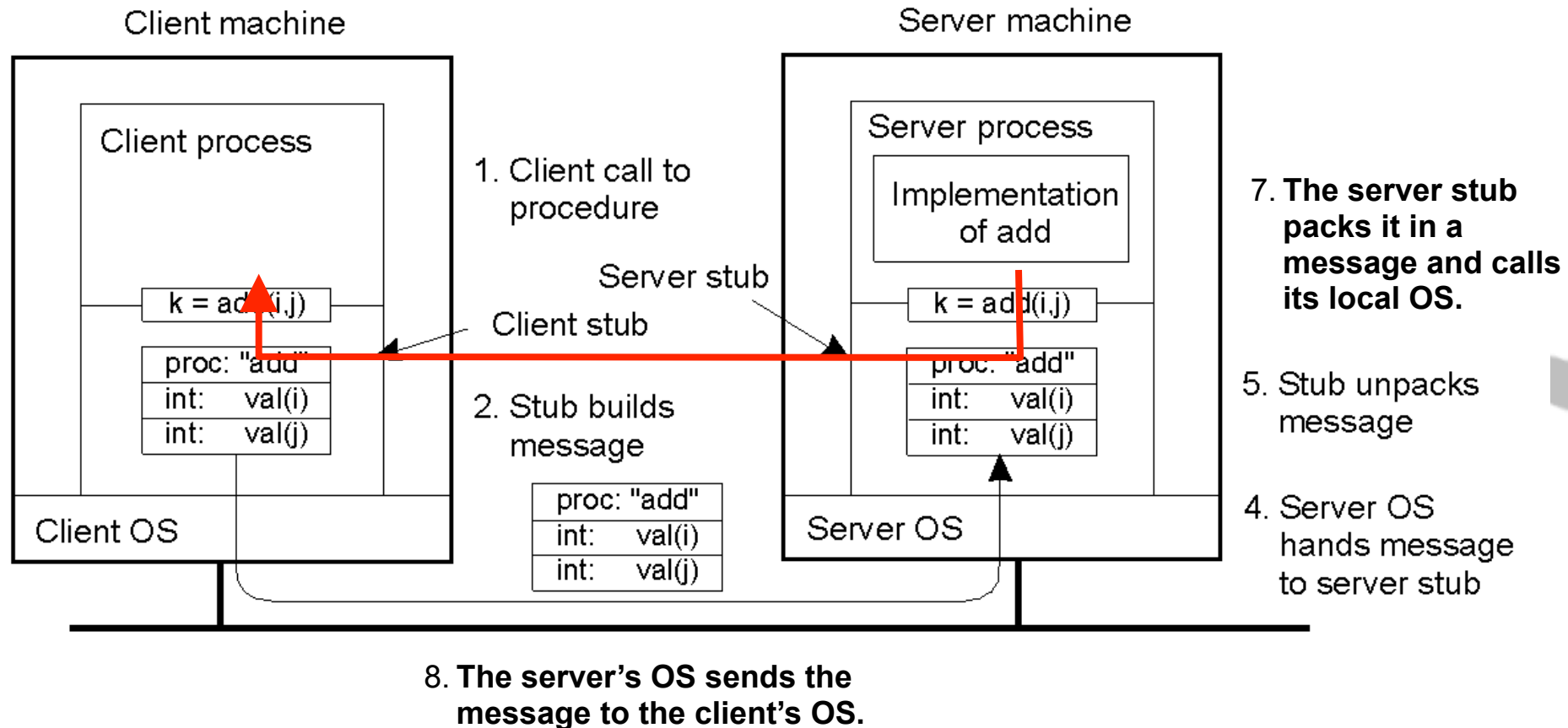
# Steps of a Remote Procedure Call



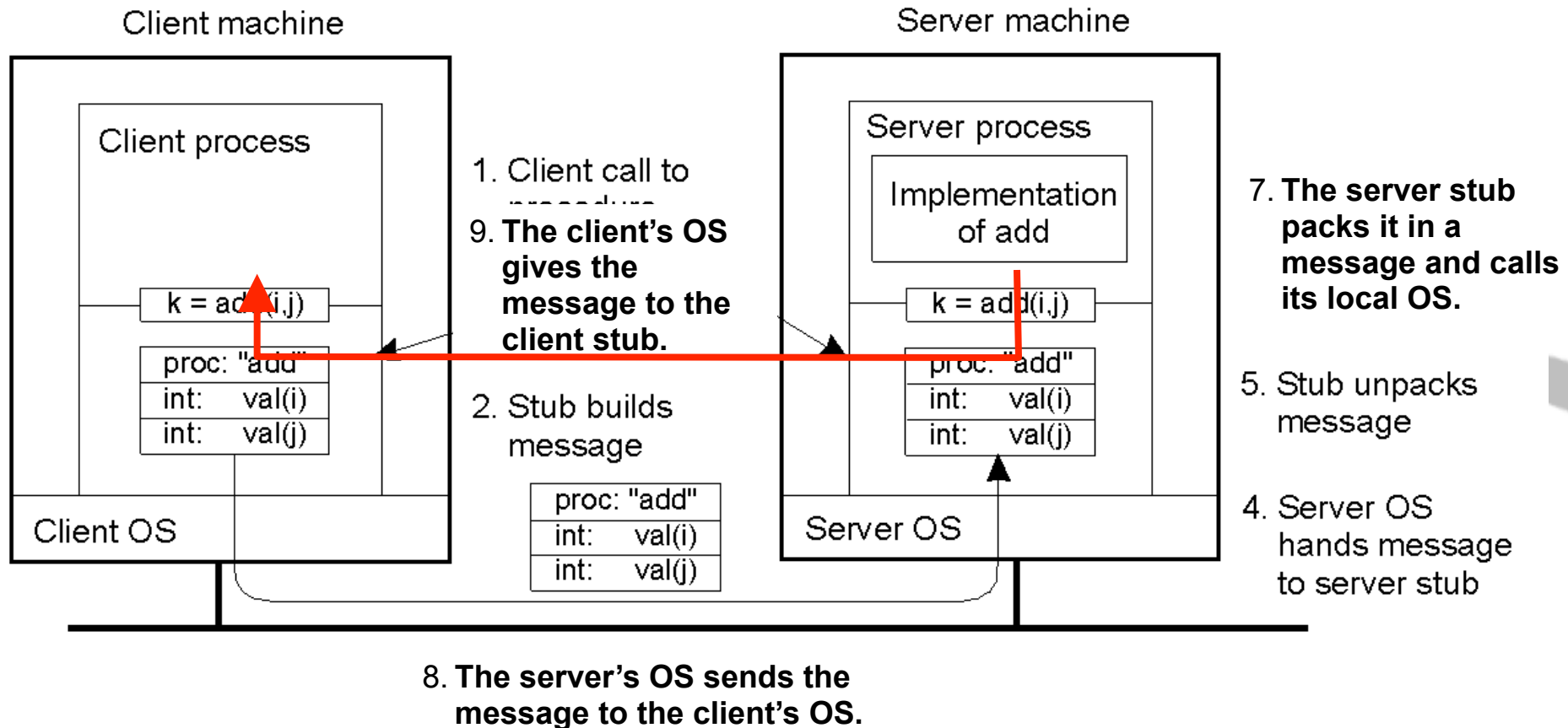
# Steps of a Remote Procedure Call



# Steps of a Remote Procedure Call

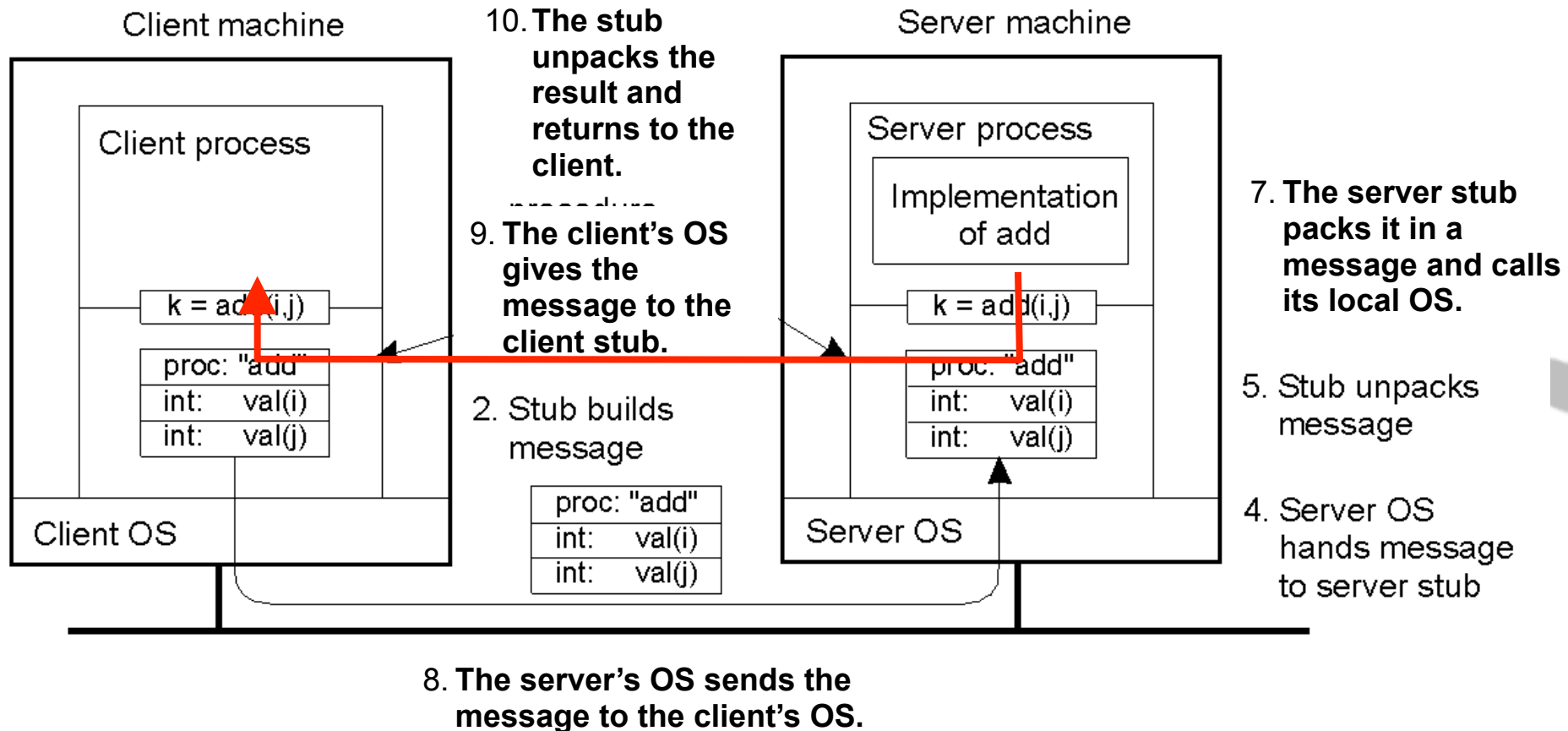


# Steps of a Remote Procedure Call





# Steps of a Remote Procedure Call



# 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

# Parameters Marshaling for Value Parameters

- Remember the text encoding problem?
  - On a PC, read the file stored on IBM Mainframe?
- Same problem on heterogeneous distributed systems.
- How to fix it?

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

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

- a) **Original message on the x86 processor**
- b) **The message after receipt on the SPARC**

(The little numbers in boxes indicate the address of each byte)

# Parameters Marshaling for Value Parameters

- Remember the text encoding problem?
  - On a PC, read the file stored on IBM Mainframe?
- Same problem on heterogeneous distributed systems.
- How to fix it?

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

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

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

- a) **Original message on the x86 processor**
- 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)

# Reference Parameters

- One can forbid using reference parameters.
- Simple solution:
  - Copy the referenced memory to the server and change the pointer.
  - Send back the procedure results AND the referenced memory.
  - **How about the synchronization issue?**
- Another solution:
  - Send the pointer to the server
  - Change the server to reference the memory on the client.

# Parameter Specification and Stub Generation

Parameters Specification includes

- Message format
- Data representation
  - Integers are in two's complement
  - How data are encoded?
- Communication Protocol
  - Using TCP or UDP?
  - Error correction?
- Interface Definition Language

```
foobar( char x; float y; int z[5] )  
{  
    ....  
}
```

(a)

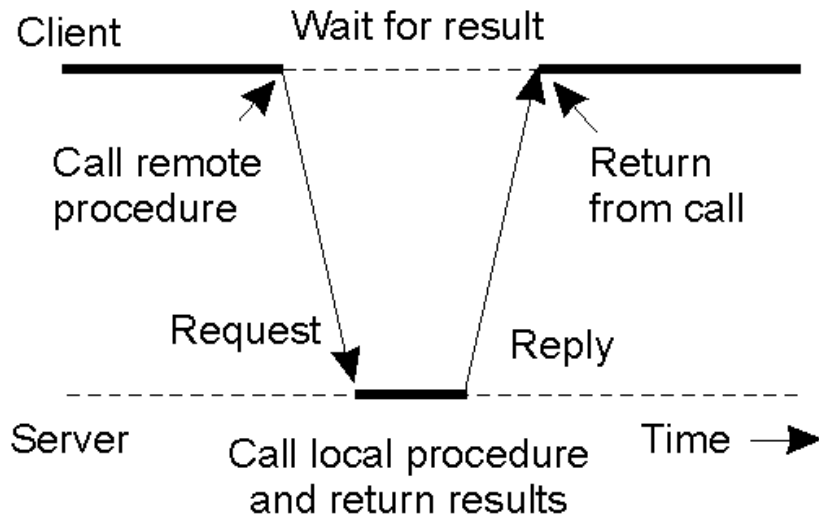
foobar's local variables	
	x
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

(b)

# Extended RPC Models

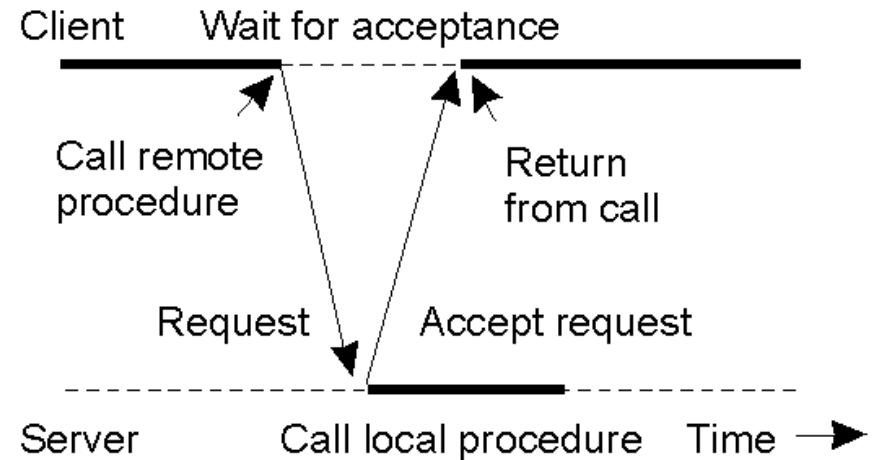
- Doors
- Asynchronized RPC
- Deferred synchronized RPC
- One-way RPC

# Asynchronous RPC



(a)

RPC



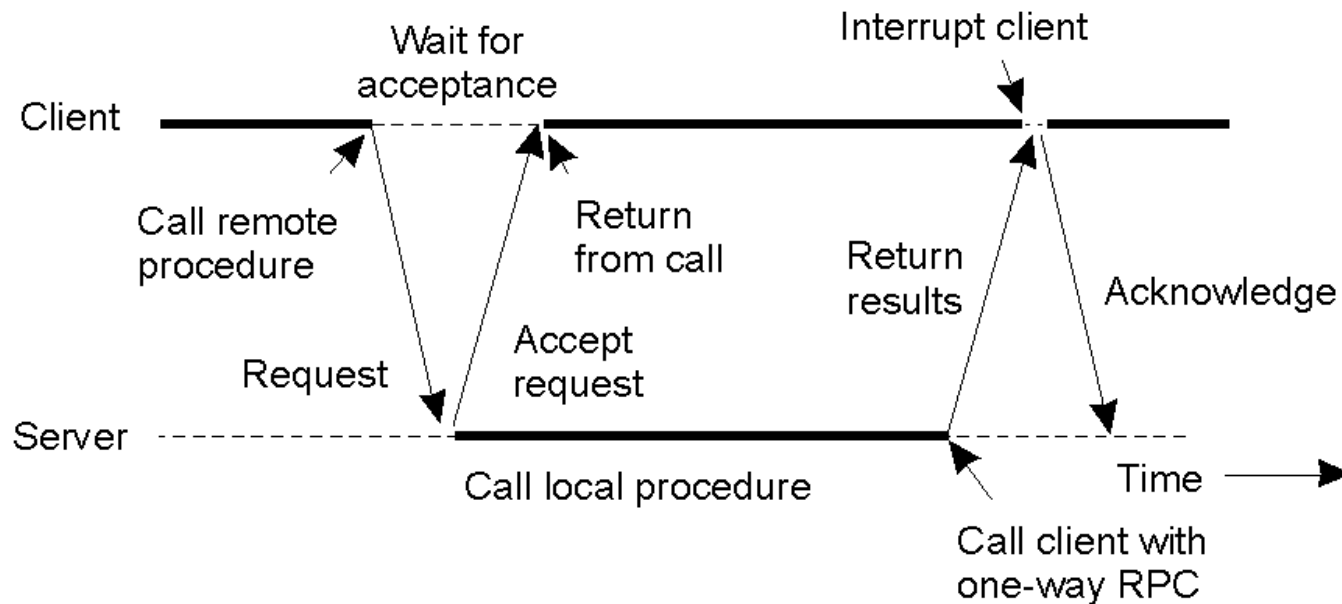
(b)

Asynchronous RPC



# Deferred synchronous RPC

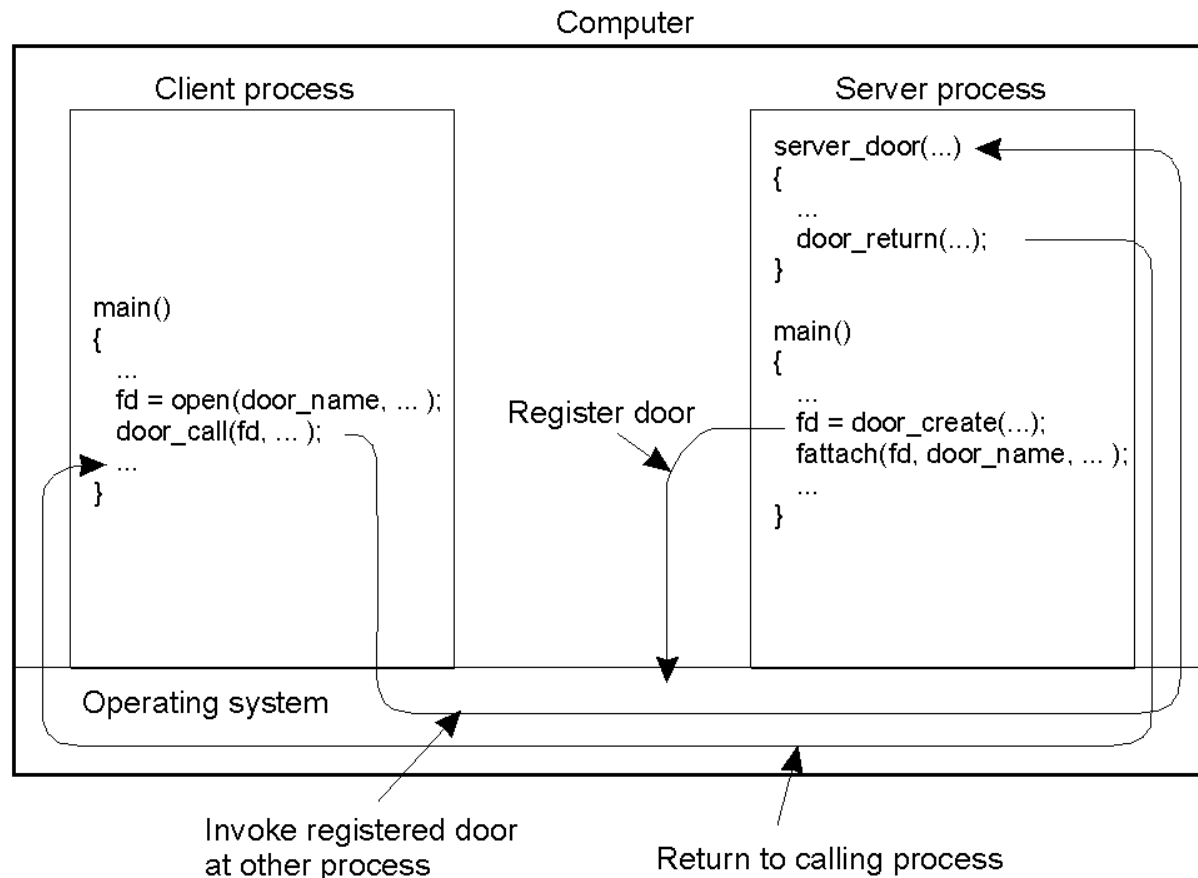
- A client and server interacting through two asynchronous RPCs



# Doors

What if the client and server are on the same machine?

- Should the message be sent as generic RPC?
- What can be done to optimize the performance?



# Lightweight RPC

## (Example Leading Discussion)

- Who wrote this paper?
- What are the background of this paper?
- What are the motivation of this paper?
- New terms:
  - Protection domain
  - A-Stacks/E-Stacks
  - Control transfer
- Performance results
- Uncommon cases:
- How to support Lightweight RPC crossing dockers?

# Lightweight RPC

## (Example Leading Discussion)

- New terms:
  - Protection domain: Domains are designed to protect each process from being attacked. One can consider single address space operating system. Within which, each process has its protection domain. Protected procedure call can transfer the control of object from one domain to another domain.
  - A-Stacks/E-Stacks: stacks for arguments and execution pointer
  - Control transfer: the requests are executed by client thread on Server's domain. Similar to executing an interrupt service route (ISR) at current process in modern operating systems.

# Performance Results

- (Don't copy the figures from paper and summarize the results in your words.)
- Compared to RPC, LRPC conducts more than 2.5 time operations per second.
- Compared to Taos, LRPC took 1/3 to 1/4 of time to complete.
- Performance under worst/uncommon case, claimed to be negligible.

# Lesson Learned for paper critics

- Lesson learned from this work:
  - Identify the performance bottleneck
  - Tackle the bottleneck with specialized solution, not general solution.

# Discussion

- Is this mechanism suitable for layer-designed or single domain operating systems?
- eRPC (Embedded Remote Procedure Call) is a Remote Procedure Call (RPC) system created by NXP use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

# Discussion


- On multi-core systems, the messages could be sent from one core to another core. Can we follow the same approach to transfer the control from one core to another one?



# RPC for dockers

- Is there any benefit of using RPC crossing dockers on the same machine?
- What's the challenge of enabling it?

# Who Wrote the Paper



[Thomas E. Anderson](#)

No contact information provided yet.

**Bibliometrics:** publication history

Publication years	1989-2012
Publication count	152
Citation Count	4,823
Available for download	70
Downloads (6 Weeks)	918
Downloads (12 Months)	7,586
Downloads (cumulative)	68,912
Average downloads per article	984.46
Average citations per article	31.73

[View colleagues](#) of Thomas E. Anderson



[Brian N. Bershad](#)

No contact information provided yet.

**Bibliometrics:** publication history

Publication years	1985-2007
Publication count	79
Citation Count	1,954
Available for download	39
Downloads (6 Weeks)	258
Downloads (12 Months)	2,150
Downloads (cumulative)	47,805
Average downloads per article	1,225.77
Average citations per article	24.73

[View colleagues](#) of Brian N. Bershad




[Henry M. Levy](#)

No contact information provided yet.

**Bibliometrics:** publication history

Publication years	1976-2011
Publication count	113
Citation Count	3,899
Available for download	72
Downloads (6 Weeks)	1,491
Downloads (12 Months)	6,843
Downloads (cumulative)	71,859
Average downloads per article	998.04
Average citations per article	34.50

[View colleagues](#) of Henry M. Levy



[Edward D. Lazowska](#)

[Home page](#)  
lazowska@cs.washington.edu

**Bibliometrics:** publication history

Publication years	1974-2012
Publication count	83
Citation Count	2,038
Available for download	56
Downloads (6 Weeks)	228
Downloads (12 Months)	1,793
Downloads (cumulative)	33,535
Average downloads per article	598.84
Average citations per article	24.55

[View colleagues](#) of Edward D. Lazowska

# Background of the paper

- Year: 1990:
  - Background: 10Mbs network, Intel 386/486: IA-32 (32-bits processors)
  - RPC was developed in '80
  - Small/Micro-kernel was targeted in this paper.
  - Borrow disjoined domain concept from distributed computing systems.
  - On 6 August 1991, CERN, a pan European organization for particle research, publicized the new World Wide Web project. The Web was invented by British scientist Tim Berners-Lee in 1989.
- Observations:
  - Most communication traffic is between domains in the same operating systems.
  - Most of communication traffic do not contain complex data structure.
  - Conventional RPC has high overhead on small/micro kernel systems.

# Motivation

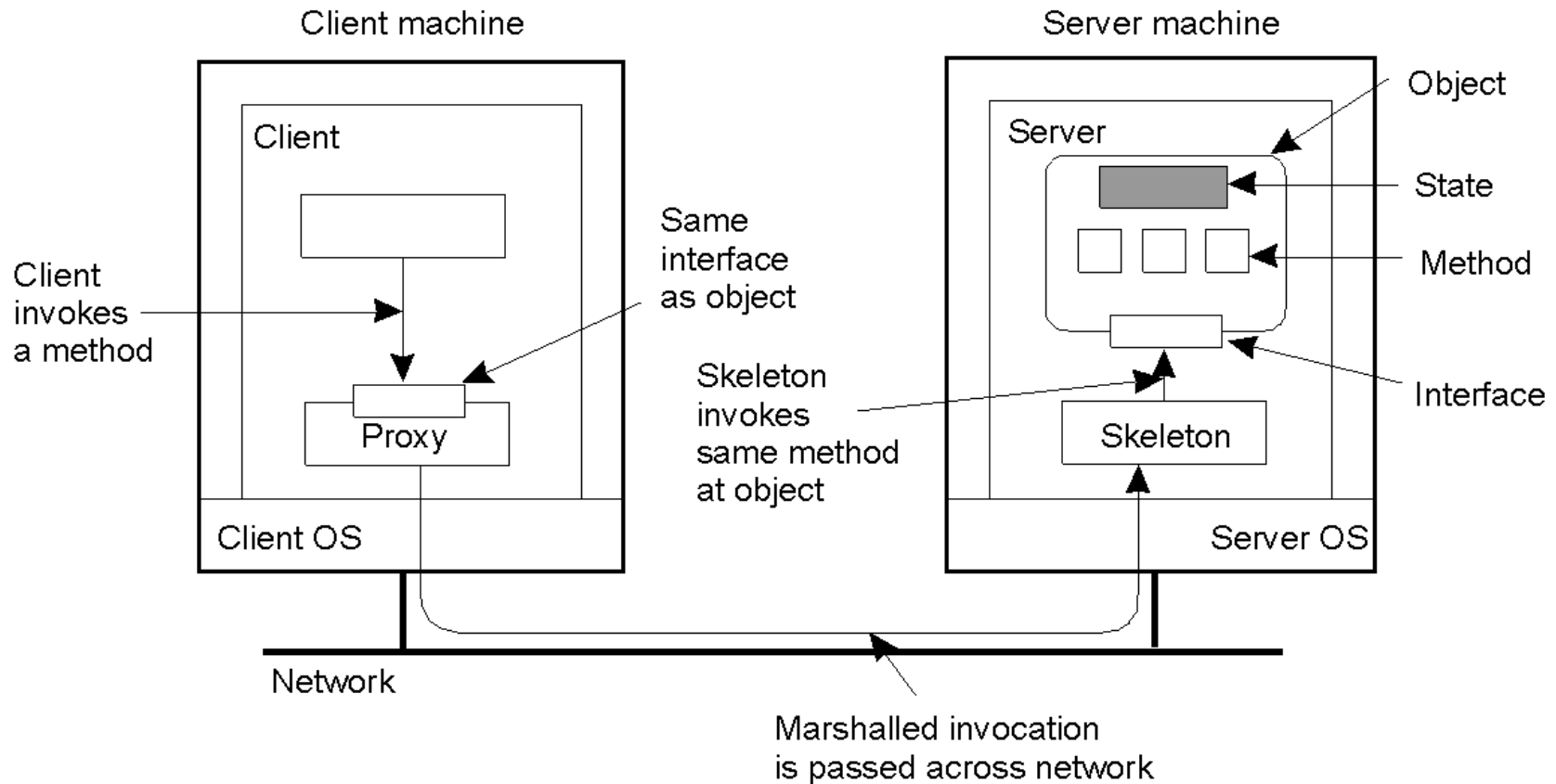
- Message-based mechanism can serve the communication needs of both local and remote subsystems.
- However, it violates a basic tenet of basic system design by failing to isolate the common case.
  - Ensure protection
  - Isolating failure
- Separate normal case and worst case:
  - Fast for normal cases; make progress for worst case

# Remote Object Invocation

- An object hides its internal from the outside world by means of well-defined interface.
  - **State**: encapsulated data
  - **Method**: operations, accessible through **interface**
- RPC principles could be applied to objects to make the life easier.
- CORBA, COM/DCOM, and JINI are mature object-based distributed system.

# Remotes Objects

- Data (state) are kept on the server (remote site), not sent to the client.



- Consistence vs. Performance

# Objects in Distributed Systems

- Compile-time Objects vs. Runtime Objects
  - Compile-time objects are language dependent and easier to build distributed systems.
  - Run-time objects are language independent and are more flexible.
    - Adaptors are needed for run-time objects.
- Persistent Objects vs. Transient Objects
  - A persistent object is not dependent on its current server.
  - A transient object exists as long as the server that manages the object.

# Binding a Client to an Object

```
Distr_object* obj_ref;  
obj_ref = ...;  
obj_ref-> do_something();
```

```
//Declare a system-wide object reference  
// Initialize the reference to a distributed object  
// Implicitly bind and invoke a method
```

(a) Implicit Binding

```
Distr_object objPref;  
Local_object* obj_ptr;  
obj_ref = ...;  
obj_ptr = bind(obj_ref);  
obj_ptr -> do_something();
```

```
//Declare a system-wide object reference  
//Declare a pointer to local objects  
//Initialize the reference to a distributed object  
//Explicitly bind and obtain a pointer to the local proxy  
//Invoke a method on the local proxy
```

(b) Explicit Binding



# Implementation of Object Reference

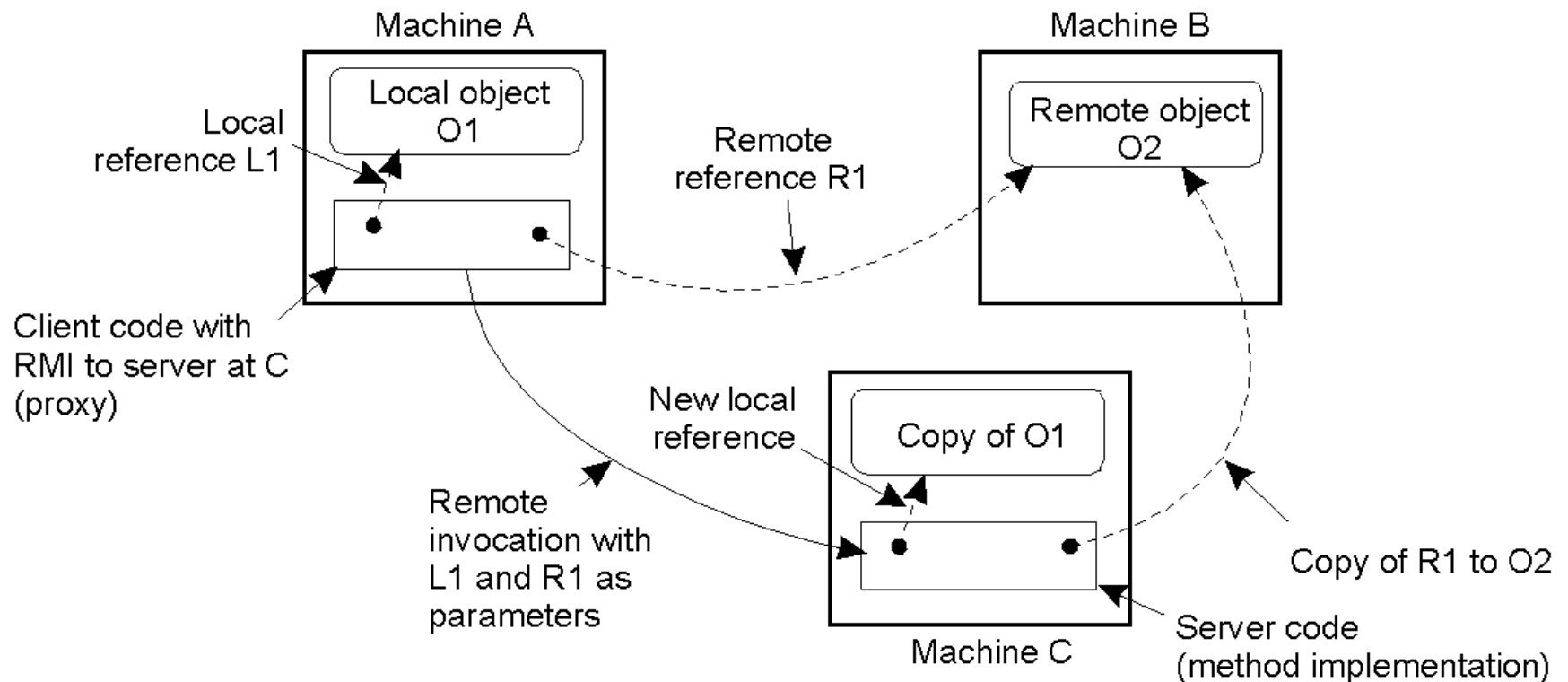
- How can the clients in a distributed system find the remote objects?
- Simple solution:
  - Giving each server and object a system-wide IDs.
  - Build a reference table.
- What trouble you may find?
  - The servers may crash.
  - The servers may move from one machine to another one.

# Remote Method Invocation

- RMI is similar to RPC but it provides system-wide method invocation.
- Static invocation
  - Using interface definition language or
  - Using object-based language and predefined interface definitions.
- Dynamic invocation
  - (Why/When do we need dynamic invocation?)

# Parameter Passing

- The scenario when passing an object by reference or by value.

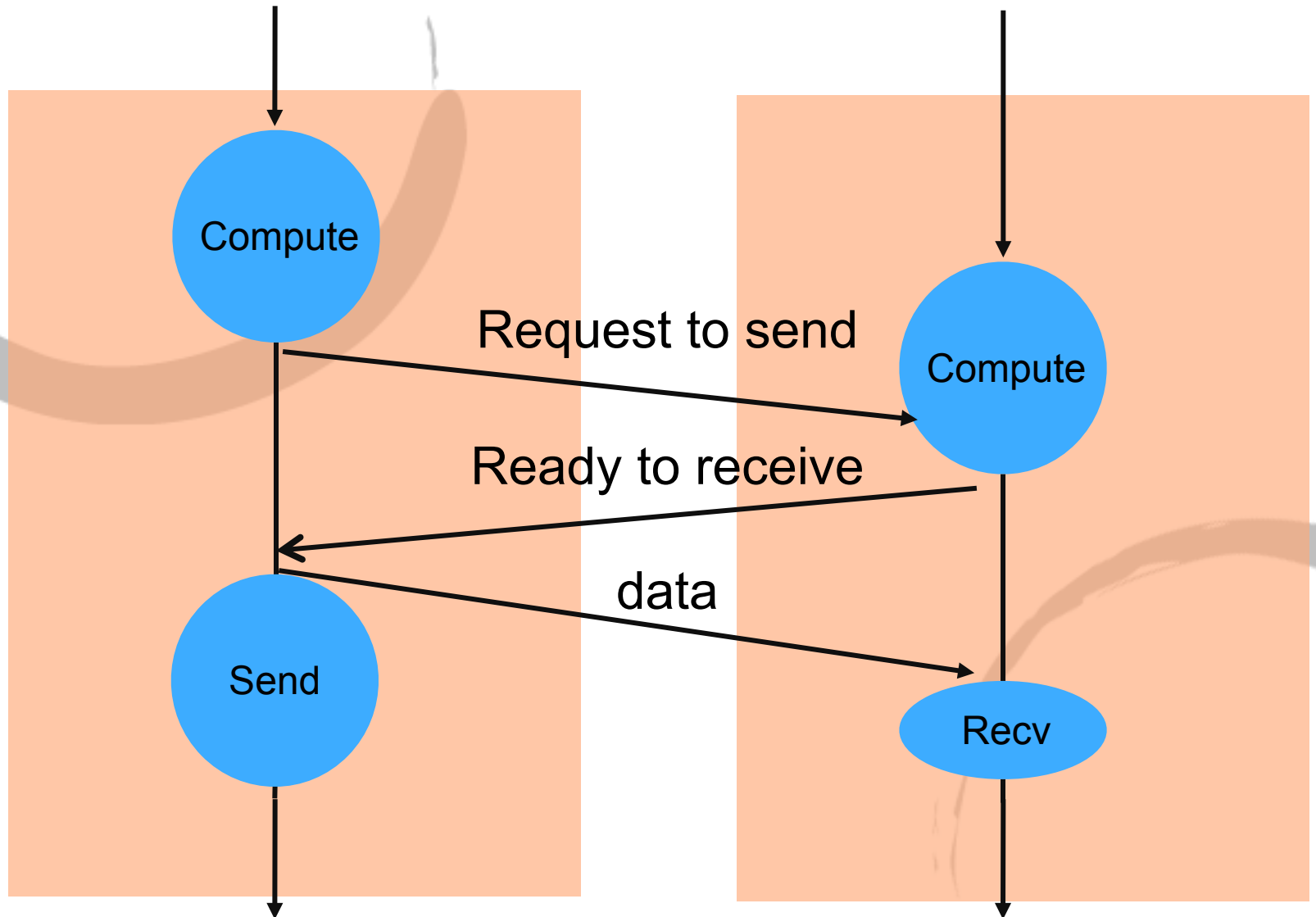


# How to Improve Performance of RPC?

- With RPC, a three-phase protocol lowers the processor performance ratio:
  - Processor performance ratio:  $T_{\text{comp}}/T_{\text{all}} = T_{\text{comp}}/(T_{\text{comp}} + T_{\text{comm}})$
  - When the ratio is 0.1, the processor is idle for 90% of the time.
- How to increase the ratio?

Von Eicken, T., Culler, D. E., Goldstein, S. C., & Schauser, K. E. (1992). Active messages: a mechanism for integrated communication and computation. ACM SIGARCH Computer Architecture News, 20(2), 256–266.

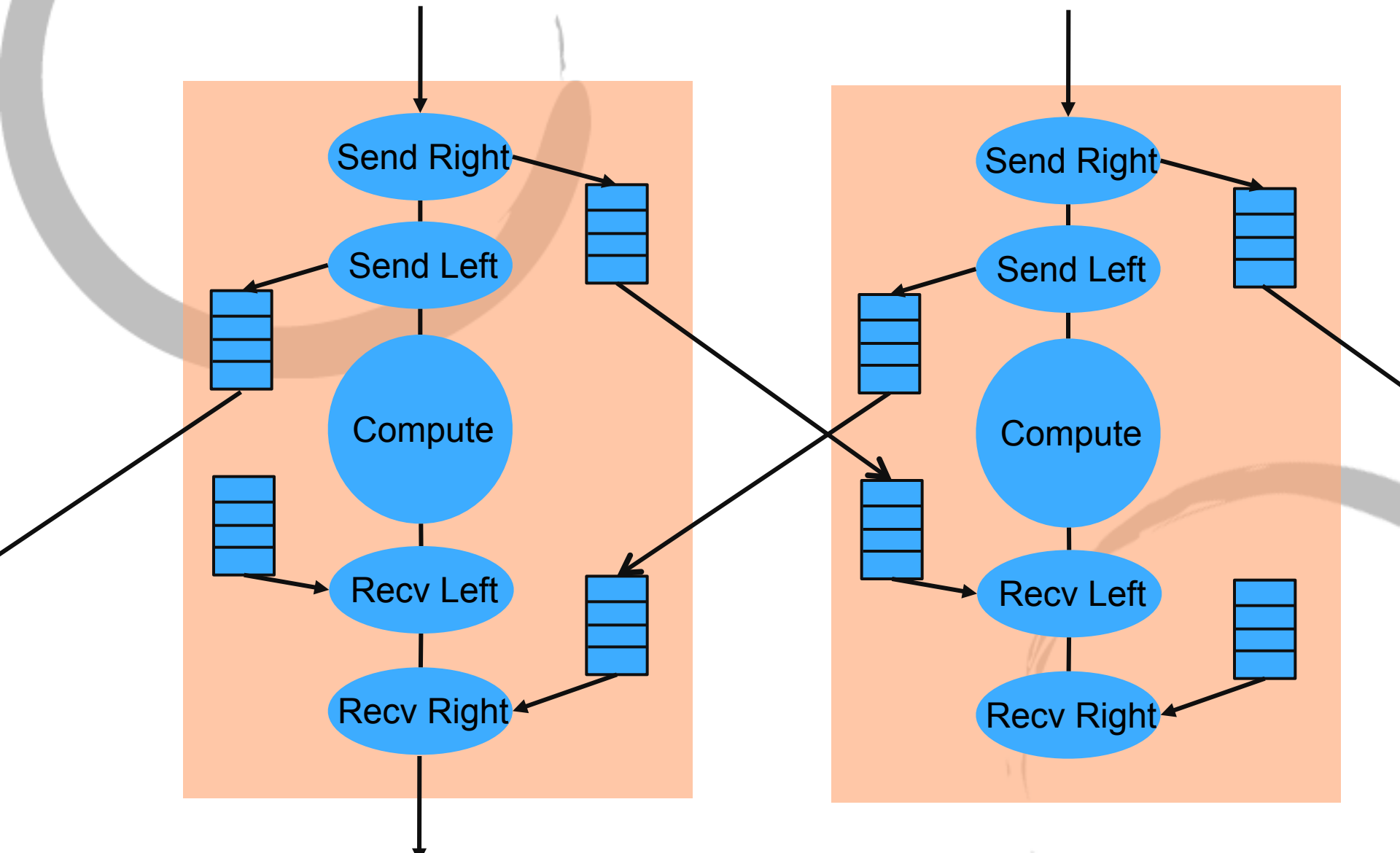
# Traditional Messaging



# Active Messages

- Who wrote this paper?
  - Berkeley CS Division – Thorsten Von Eicken, David E. Culler (Professor in Computer Science at UC Berkeley),
- What are the background of this paper?
  - 1992, Similar to Lightweight RPC. Work on large-scale multiprocessor
- What are the motivation of this paper?
  - Overlap communication and computation and avoid buffering on receiver
- New terms:
  - Start-up Cost
  - Three-phase Control
- How is the Active Messages different from RPC?
  - RPC processes the enclosed data; active message extracts the enclosed into proper location
- Limitation: Limitation: SPMD (Single Process, Multiple Data) /SMP (Symmetric Multiple Processor)
- Performance results

# Active Message

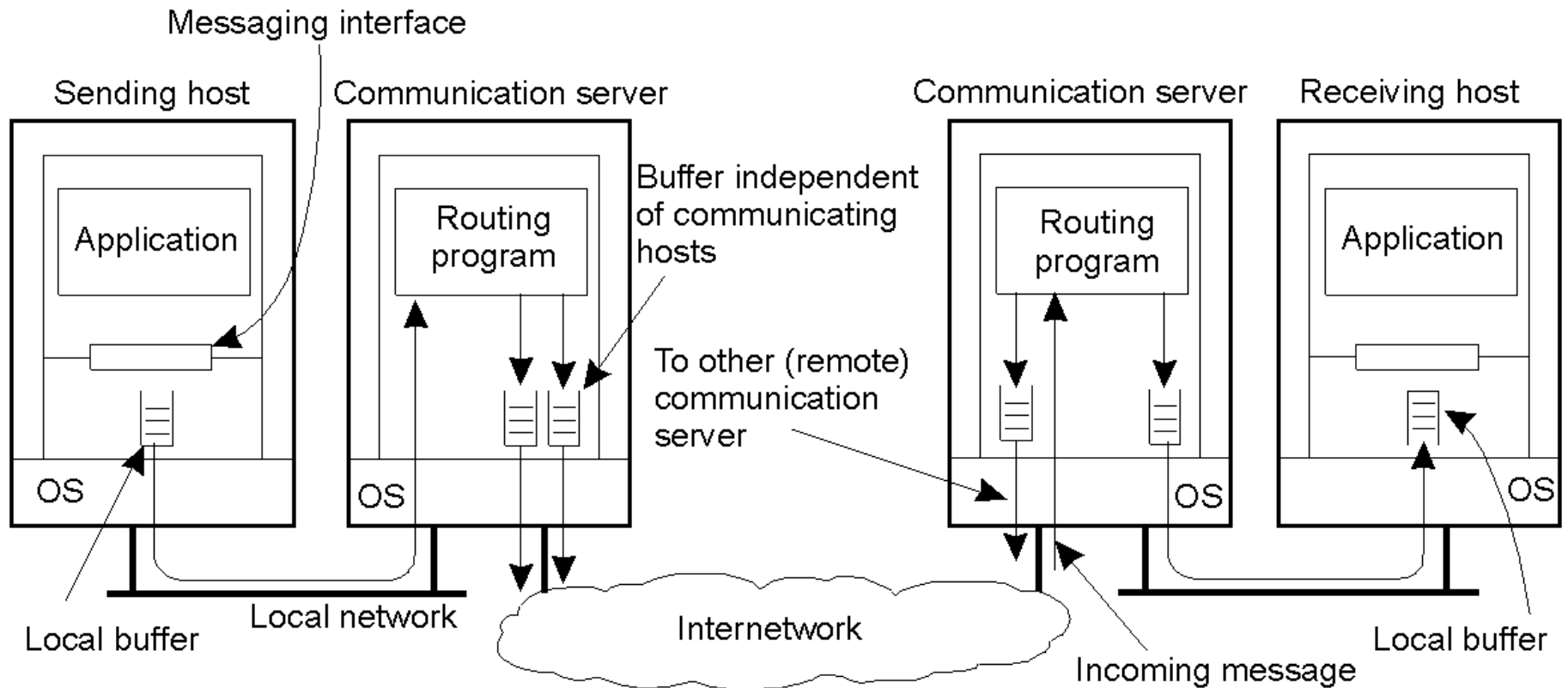


# Message-Oriented Communication

- RPC and RMI hide communication in distributed systems and enhance access transparency.
- But,
  - Client and server have to be on at the same time.
  - Synchronous mechanism blocks the clients.
- So, here is message-oriented communication.



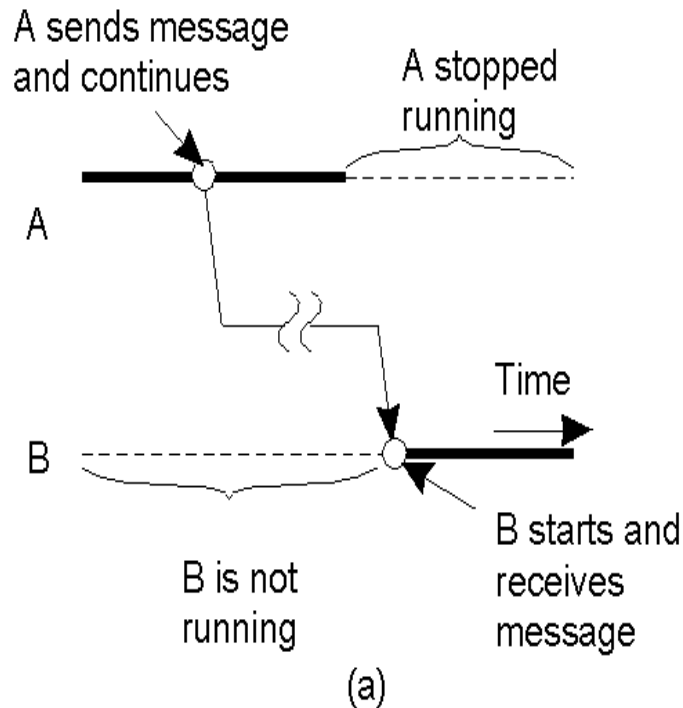
## Persistence and Synchronicity in Communication



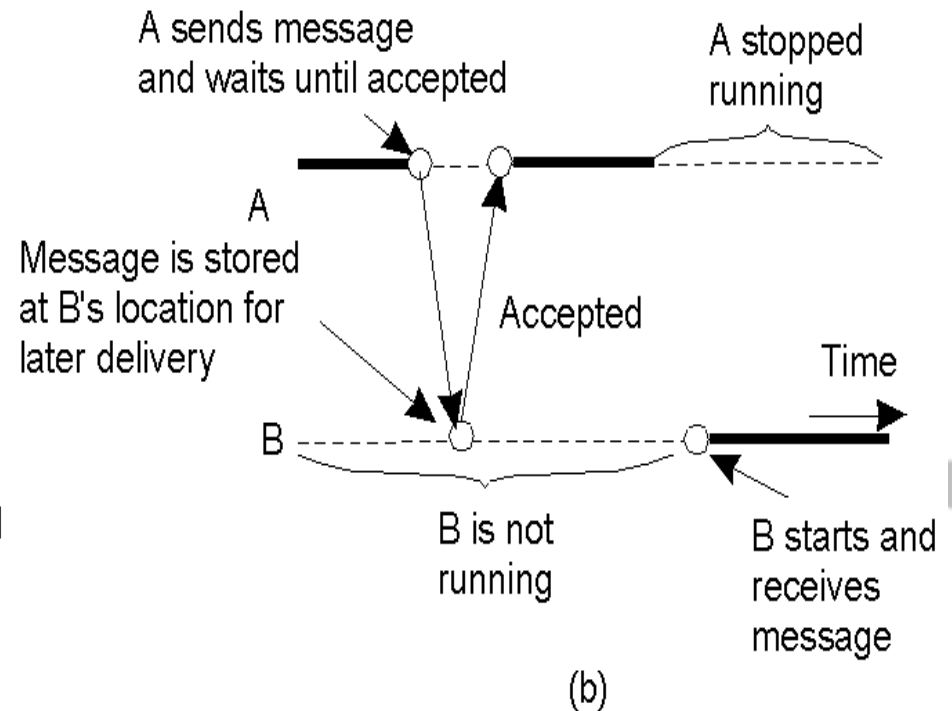
# Message-Oriented Communication

- Alternatives
  - Persistence: persistent communication vs. transient communication.
  - Synchronicity: asynchronous communication vs. synchronous communication
- As an engineer, how do you choose the communication protocol for message-oriented communication application?
  - Persistent asynchronous
  - Transient asynchronous
  - ...

# Persistence and Synchronicity in Communication

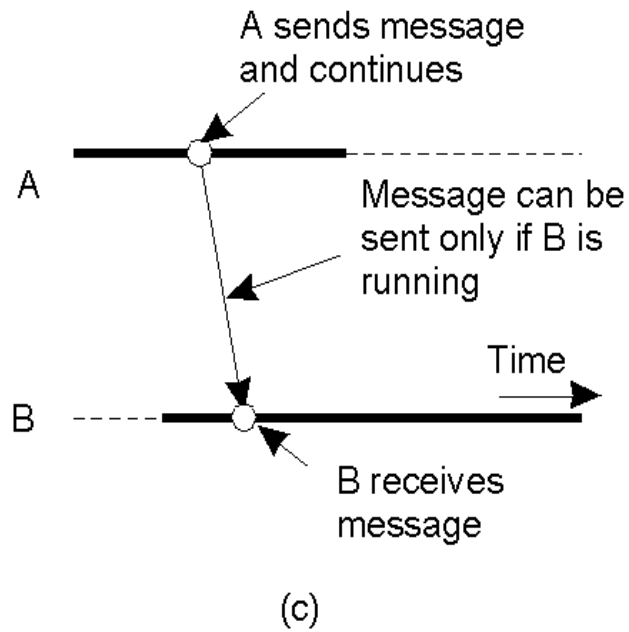


**Persistent asynchronous communication**

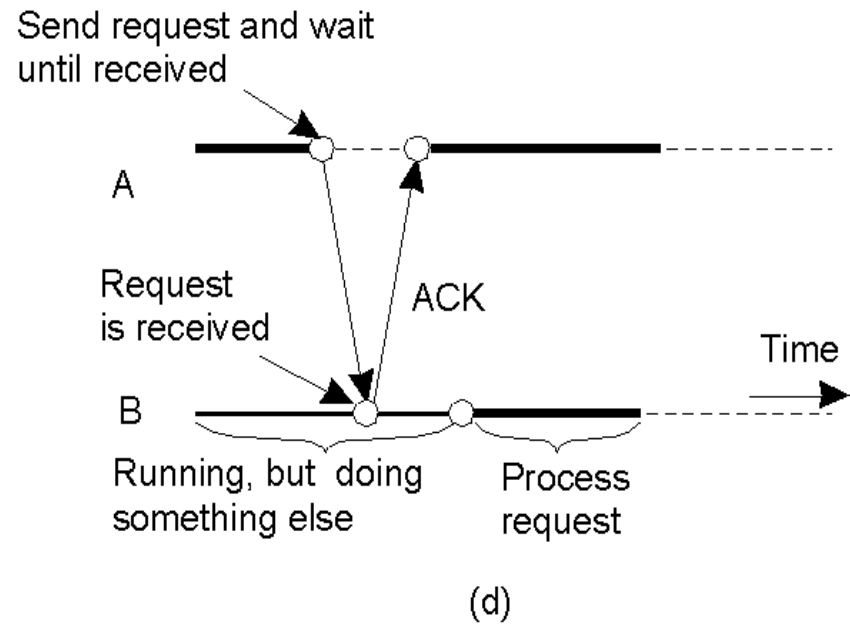


**Persistent synchronous communication**

# Persistence and Synchronicity in Communication

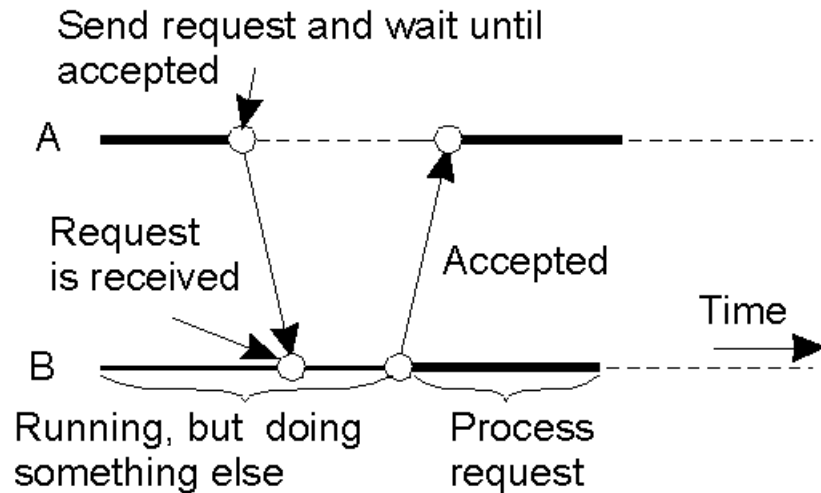


Transient asynchronous communication



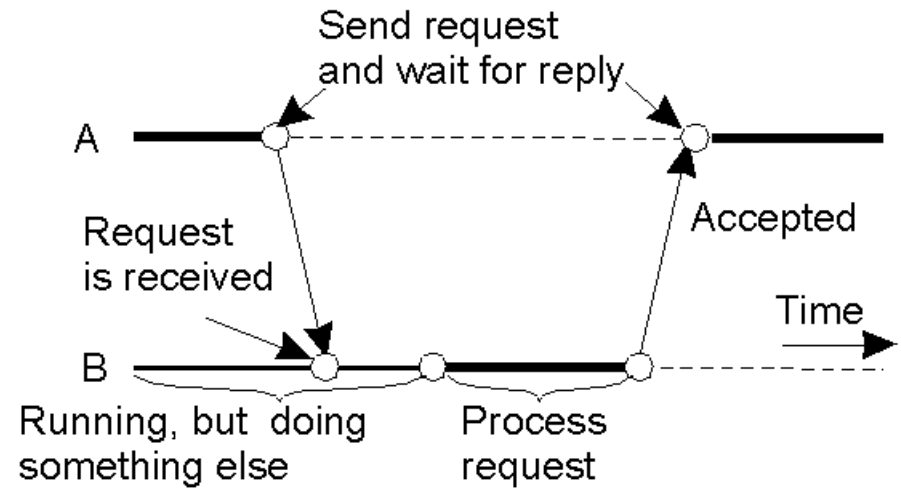
**Receipt-based transient synchronous communication**

# Persistence and Synchronicity in Communication



(e)

Delivery-based transient synchronous communication at message delivery

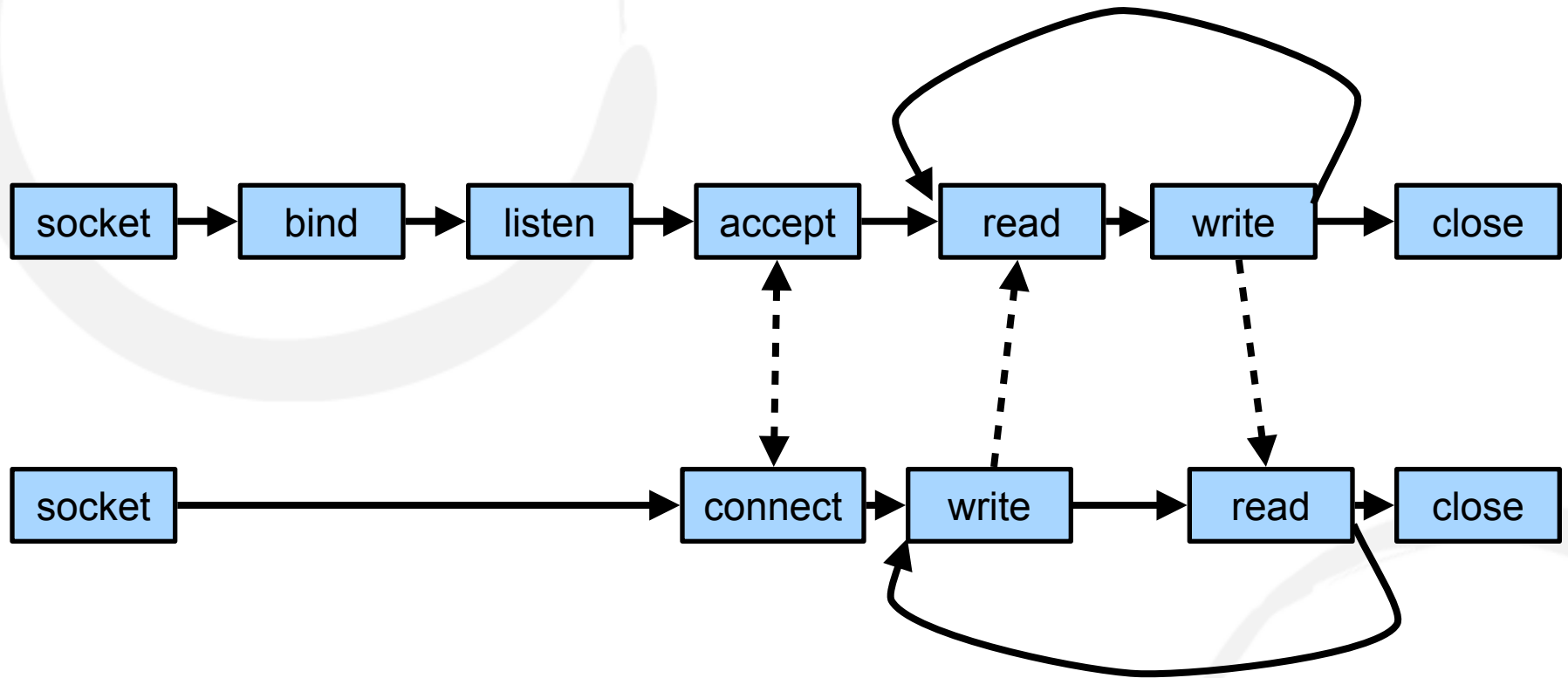


(f)

Response-based transient synchronous communication

# Message-Oriented Transient Communication

- Socket is used for one-to-one communication.



- Message-passing-Interface (MPI) is used for one-to-many communication.

# Message Passing Interface

- A standard message passing specification for the vendors to implement
- **Context: distributed memory parallel computers**
  - Each processor has its own memory and cannot access the memory of other processors
  - Any data to be shared must be explicitly transmitted from one to another
- Most message passing programs use the *single program multiple data (SPMD)* model
  - Each processor executes the same set of instructions
  - Parallelization is achieved by letting each processor operate a different piece of data
  - MIMD (Multiple Instructions Multiple Data): multiple program to process multiple data

# Example for SPMD

```
main(int argc, char **argv){
    if(process is assigned as Master role){
        /* Assign work and coordinate workers and collect results */
        MasterRoutine(/*arguments*/);
    } else { /* it is worker process */
        /* interact with master and other workers. Do the work and send
        results to the master*/
        WorkerRoutine(/*arguments*/);
    }
}
```



# The Message-Passing Interface (MPI)

- Some of the most intuitive message-passing primitives of MPI.

Primitive	Meaning
<b>MPI_bsend</b>	<b>Append outgoing message to a local send buffer</b>
<b>MPI_send</b>	<b>Send a message and wait until copied to local or remote buffer</b>
<b>MPI_ssend</b>	<b>Send a message and wait until receipt starts</b>
<b>MPI_sendrecv</b>	<b>Send a message and wait for reply</b>
<b>MPI_isead</b>	<b>Pass reference to outgoing message, and continue</b>
<b>MPI_issend</b>	<b>Pass reference to outgoing message, and wait until receipt starts</b>
<b>MPI_recv</b>	<b>Receive a message; block if there are none</b>
<b>MPI_irecv</b>	<b>Check if there is an incoming message, but do not block</b>

# MPI Basic Send/Recv

```
int MPI_Send( void *buf, int count, MPI_Datatype datatype,  
int dest, int tag, MPI_Comm comm )
```

**buf:** initial address of send buffer

**dest:** rank of destination (integer)

**tag:** message tag (integer)

**comm:** communicator (handle)

**count:** number of elements in send buffer (nonnegative integer)

**datatype:** datatype of each send buffer element (handle)

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype,  
int source, int tag, MPI_Comm comm, MPI_Status *status )
```

**status:** status object (Status)

**source:** rank of source (integer)

- status is mainly useful when messages are received with **MPI\_ANY\_TAG** and/or **MPI\_ANY\_SOURCE**

# Information about a Message

- `count` argument in `recv` indicates maximum length of a message
- Actual length of message can be got using `MPI_Get_Count`

```
MPI_Status status;
```

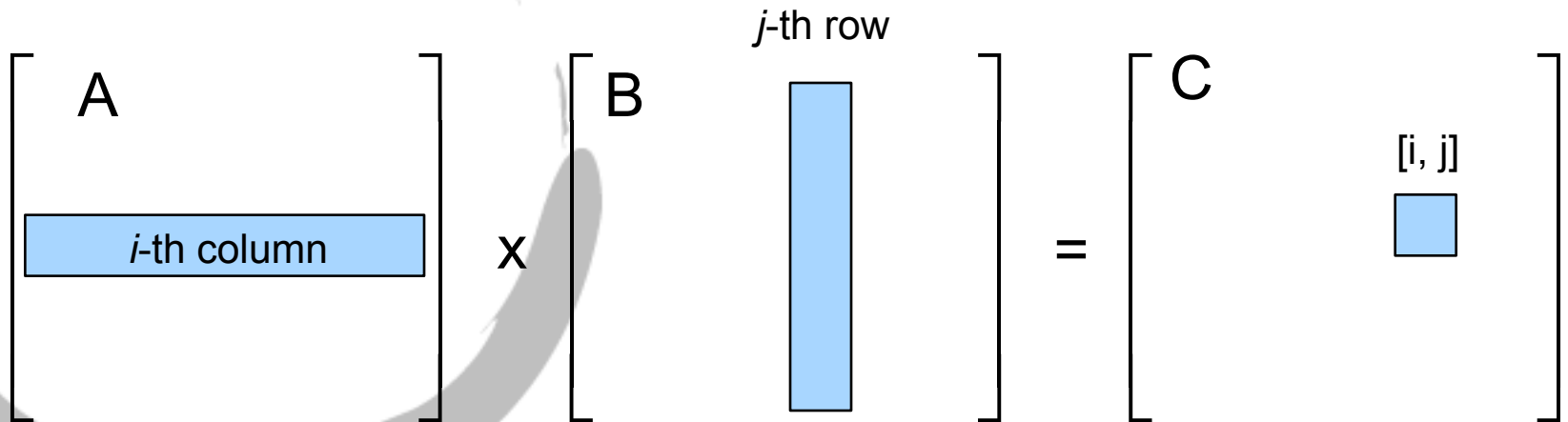
```
MPI_Recv( ..., &status );
```

```
... status.MPI_TAG;
```

```
... status.MPI_SOURCE;
```

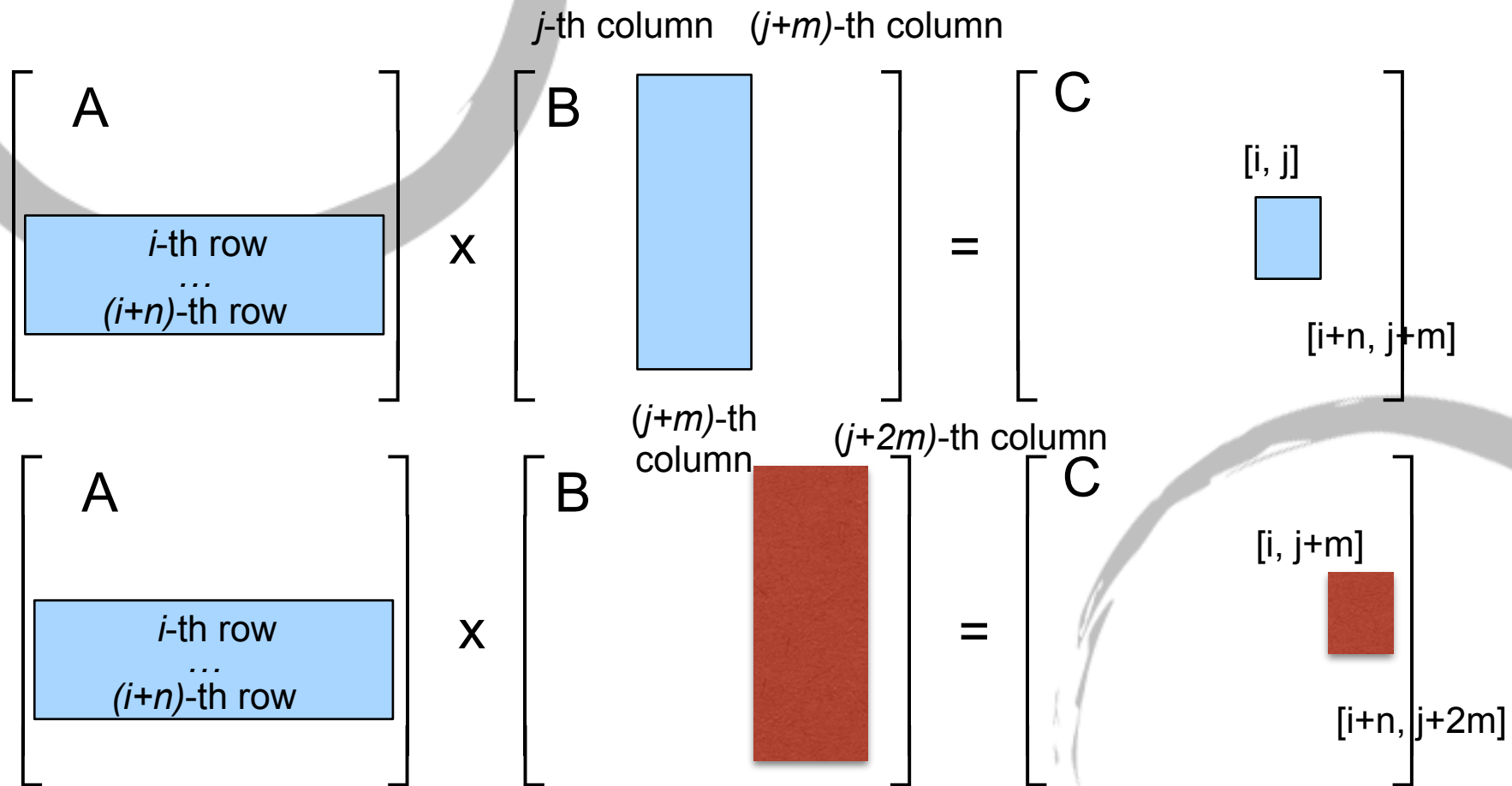
```
MPI_Get_count( &status, datatype, &count );
```

# Matrix Multiplication



```
do i = 1, n
  do k = 1, l
    do j = 1, m
      c(i,k) = c(i,k) + a(i,j)*b(j,k)
    end do
  end do
end do
```

# Matrix Multiplication - Divide the work for distributed computing



# Example: Matrix Multiplication Program

```
/* send matrix data to the worker tasks */
```

MASTER  
SIDE

```
averow = NRA/numworkers;
extra = NRA%numworkers;
offset = 0;
mtype = FROM_MASTER;
for (dest=1; dest<=numworkers; dest++) {
    rows = (dest <= extra) ? averow+1 : averow;           // If # rows not divisible absolutely by # workers
    printf("sending %d rows to task %d\n",rows,dest);    // some workers get an additional row
    MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD); // Starting row being sent
    MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);  // # rows sent

    count = rows*NCA;                                     // Gives total # elements being sent
    MPI_Send(&a[offset][0], count, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);

    count = NCA*NCB;                                     // Equivalent to NRB * NCB; # elements in B
    MPI_Send(&b, count, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);

    offset = offset + rows;                               // Increment offset for the next worker
}
```

# Example: Matrix Multiplication Program (contd.)

MASTER  
SIDE

```
/* wait for results from all worker tasks */
```

```
mtype = FROM_WORKER;
```

```
for (i=1; i<=numworkers; i++)
```

```
{
```

```
    source = i;
```

```
    MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
```

```
    MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
```

```
    count = rows*NCB;
```

```
    MPI_Recv(&c[offset][0], count, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD, &status);
```

```
}
```

```
/* print results */
```

```
} /* end of master section */
```

```
// Get results from each worker
```

```
// #elements in the result from the worker
```

# Example: Matrix Multiplication Program (contd.)

WORKER  
SIDE

```
if (taskid > MASTER)
{
    // Implies a worker node
    mtype = FROM_MASTER;
    source = MASTER;
    printf ("Master =%d, mtype=%d\n", source, mtype);

    // Receive the offset and number of rows
    MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
    printf ("offset =%d\n", offset);
    MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
    printf ("row =%d\n", rows);

    count = rows*NCA;
    // # elements to receive for matrix A
    MPI_Recv(&a, count, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD, &status);
    printf ("a[0][0] =%e\n", a[0][0]);

    count = NCA*NCB;
    // # elements to receive for matrix B
    MPI_Recv(&b, count, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD, &status);
```



# Example: Matrix Multiplication Program (contd.)

WORKER  
SIDE

```
for (k=0; k<NCB; k++)
    for (i=0; i<rows; i++) {
        c[i][k] = 0.0;
        for (j=0; j<NCA; j++)
            c[i][k] = c[i][k] + a[i][j] * b[j][k];
    }

    // Do the matrix multiplication for the # rows you are assigned to

mtype = FROM_WORKER;
printf ("after computing \n");

MPI_Send(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
MPI_Send(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);

MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD); // Sending the actual result

printf ("after send \n");

} /* end of worker */
```

# Asynchronous Send/Receive

- `MPI_Isend()` and `MPI_Irecv()` are non-blocking; control returns to program after call is made

```
int MPI_Isend( void *buf, int count, MPI_Datatype datatype,  
              int dest, int tag, MPI_Comm comm, MPI_Request  
              *request )
```

```
int MPI_Irecv( void *buf, int count, MPI_Datatype datatype,  
              int source, int tag, MPI_Comm comm, MPI_Request  
              *request )
```

**request:** communication request (handle); output parameter

# Detecting Completions

- Non-blocking operations return (immediately) “request handles” that can be waited on and queried
- `MPI_Wait` waits for an MPI send or receive to complete  
`int MPI_Wait ( MPI_Request *request, MPI_Status *status)`
  - request matches request on `Isend` or `Irecv`
  - status returns the status equivalent to status for `MPI_Recv` when complete
  - blocks for send until message is buffered or sent so message variable is free
  - blocks for receive until message is received and ready

# Detecting Completions (contd.)

- **MPI\_Test** tests for the completion of a send or receive  
`int MPI_Test ( MPI_Request *request, int *flag,  
MPI_Status *status)`
  - request, status as for **MPI\_Wait**
  - does not block
  - flag indicates whether operation is complete or not
  - enables code which can repeatedly check for communication completion

# Multiple Completions

- Often desirable to wait on multiple requests; ex., A master/slave program

```
int MPI_Waitall( int count, MPI_Request array_of_requests[],  
                MPI_Status array_of_statuses[] )
```

```
int MPI_Waitany( int count, MPI_Request array_of_requests[], int  
                *index, MPI_Status *status )
```

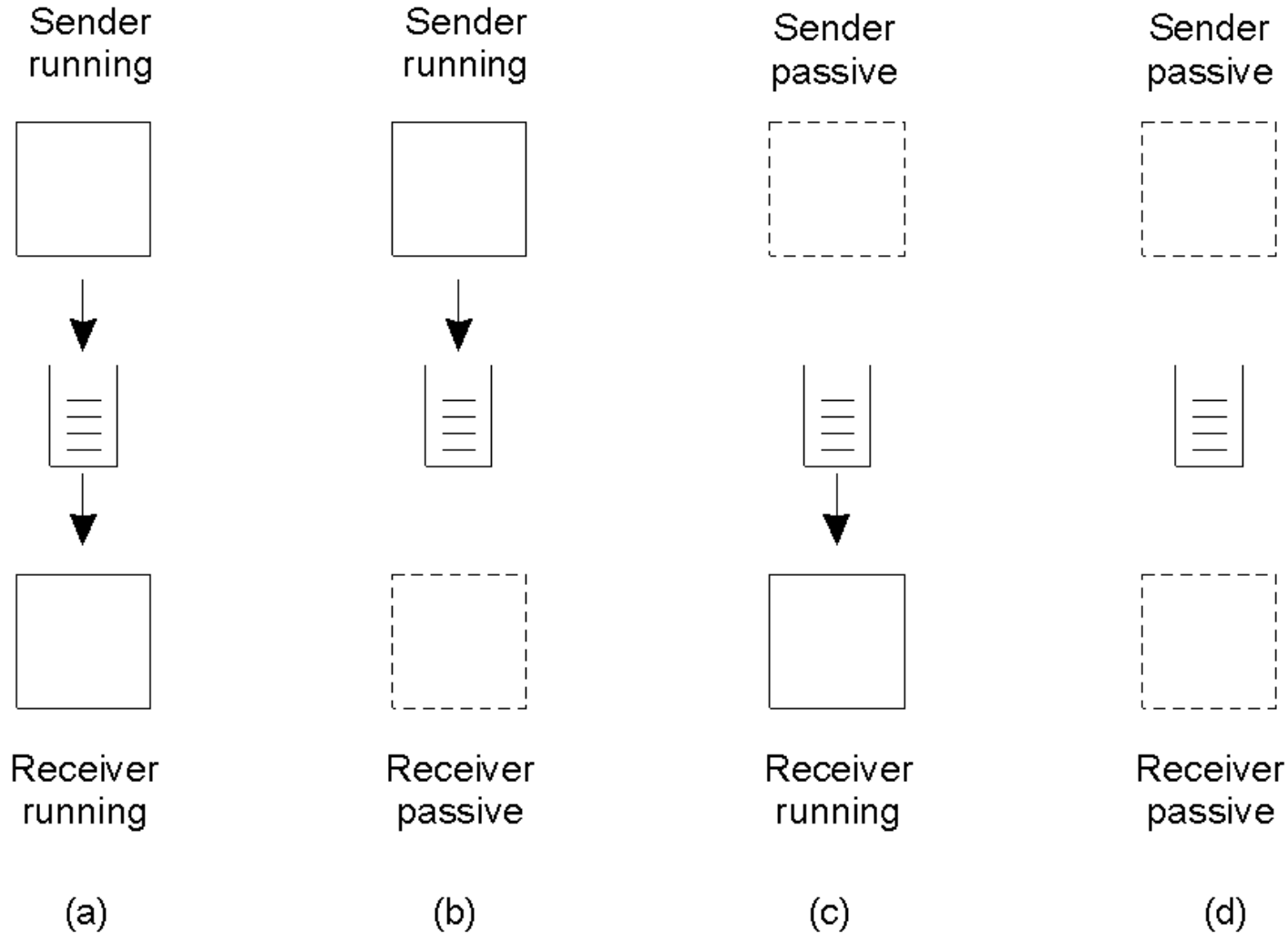
```
int MPI_Waitsome( int incount, MPI_Request array_of_requests[],  
                  int *outcount, int array_of_indices[], MPI_Status  
                  array_of_statuses[] )
```

- There are corresponding versions of test for each of these

# Message-Oriented Persistent Communication

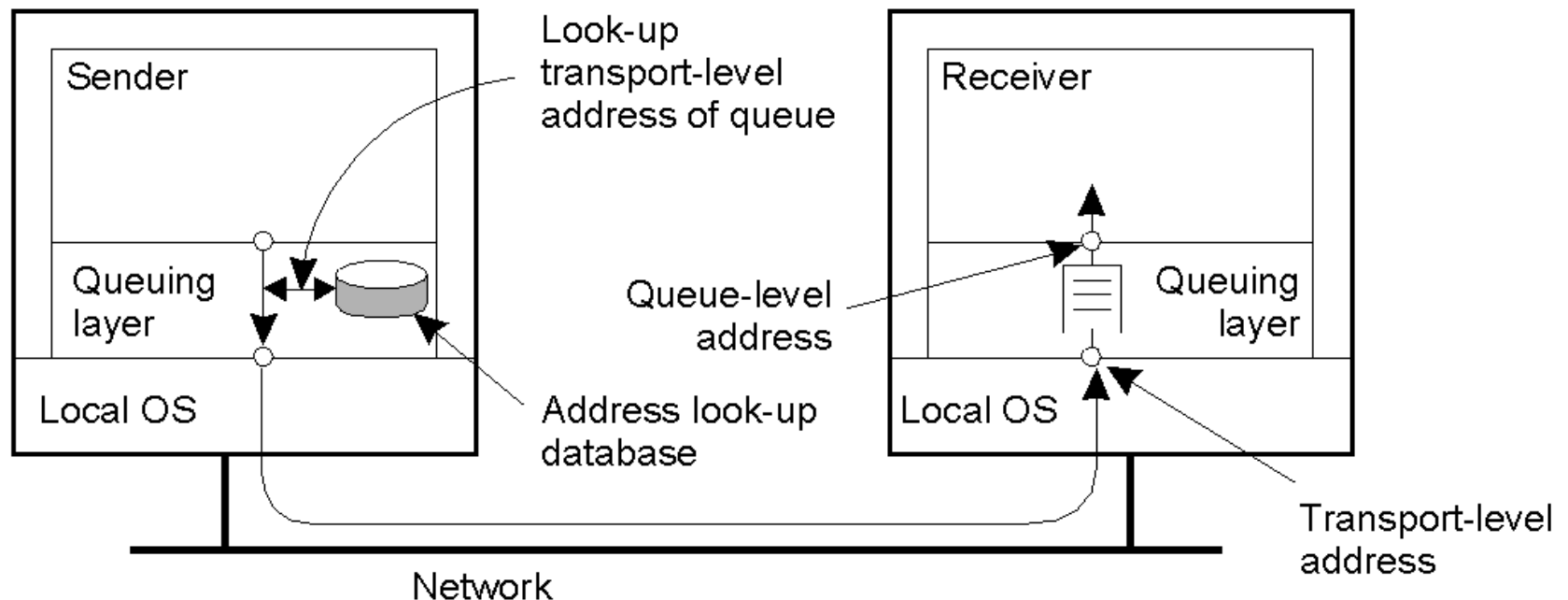
- Message-oriented middleware service does not require either the sender or receiver to be active during message transmission.
- What applications are suitable for message-oriented persistent communication?
  - what can wait for minutes.
  - what may not always be on and require robust communication
- Message-Queuing Model
  - Messages are inserted into specific queues, forwarded over a series of communication servers, and eventually delivered to the destination.
  - No guarantee are given about when, or even if the message will actually be read.

# Message-queuing Systems – a persistent asynchronous communication



## General Architecture of a Message-Queuing System (1)

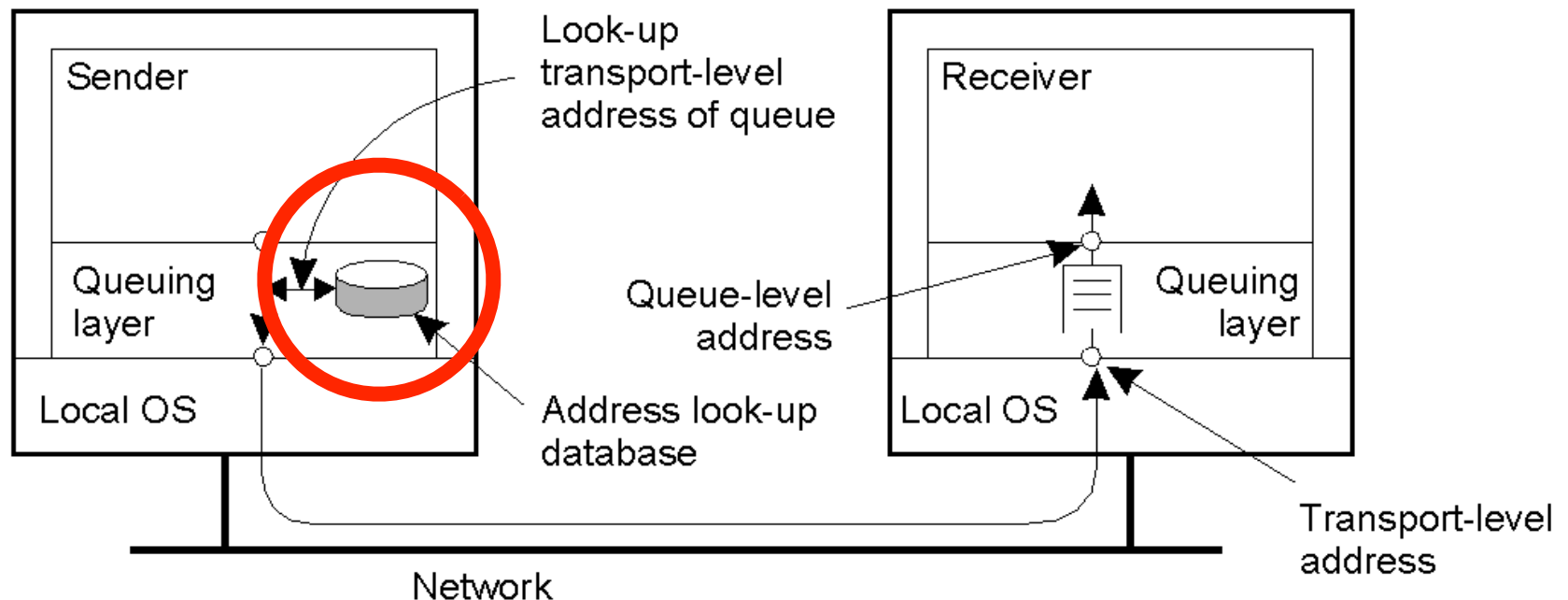
The relationship between queue-level addressing and network-level addressing.





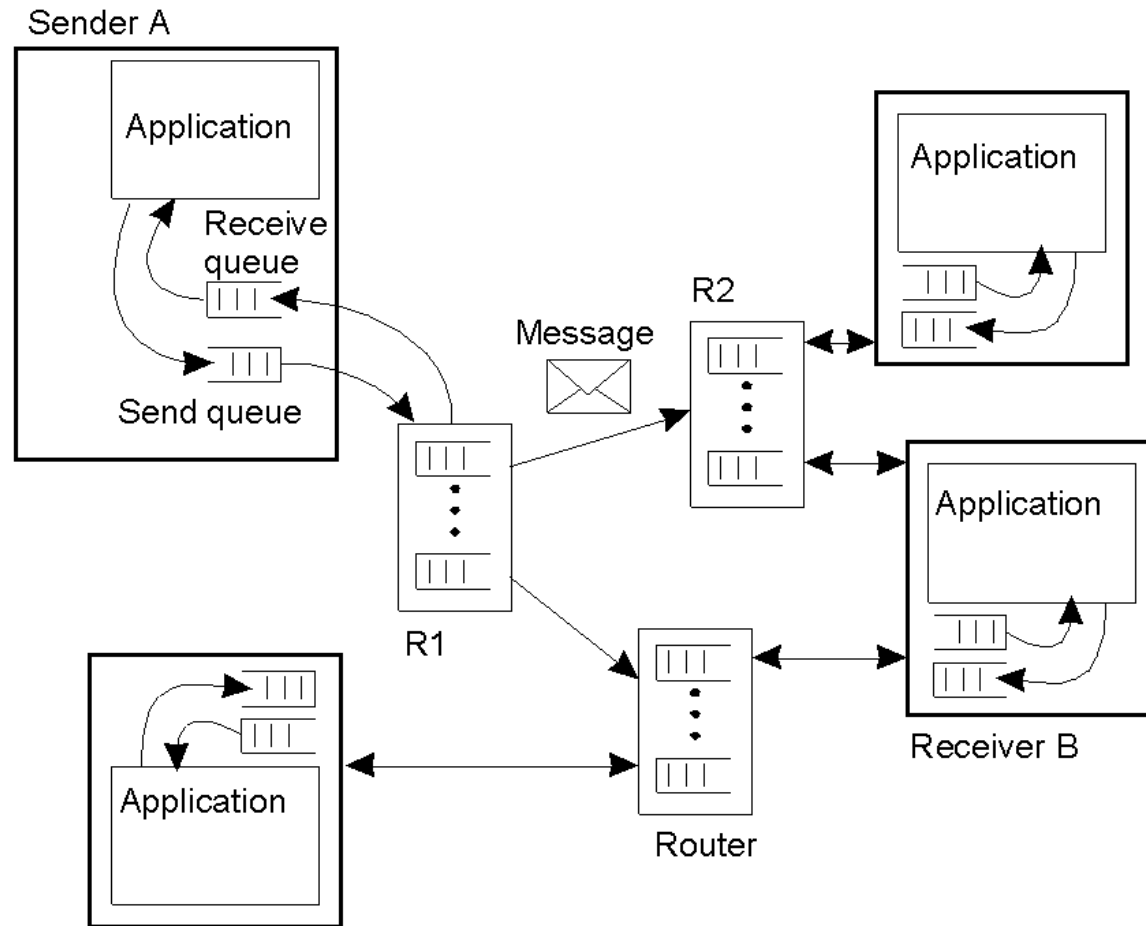
## General Architecture of a Message-Queuing System (1)

The relationship between queue-level addressing and network-level addressing.



**The database could cause network problem when the system is scaled up.**

## General Architecture of a Message-Queuing System (2)

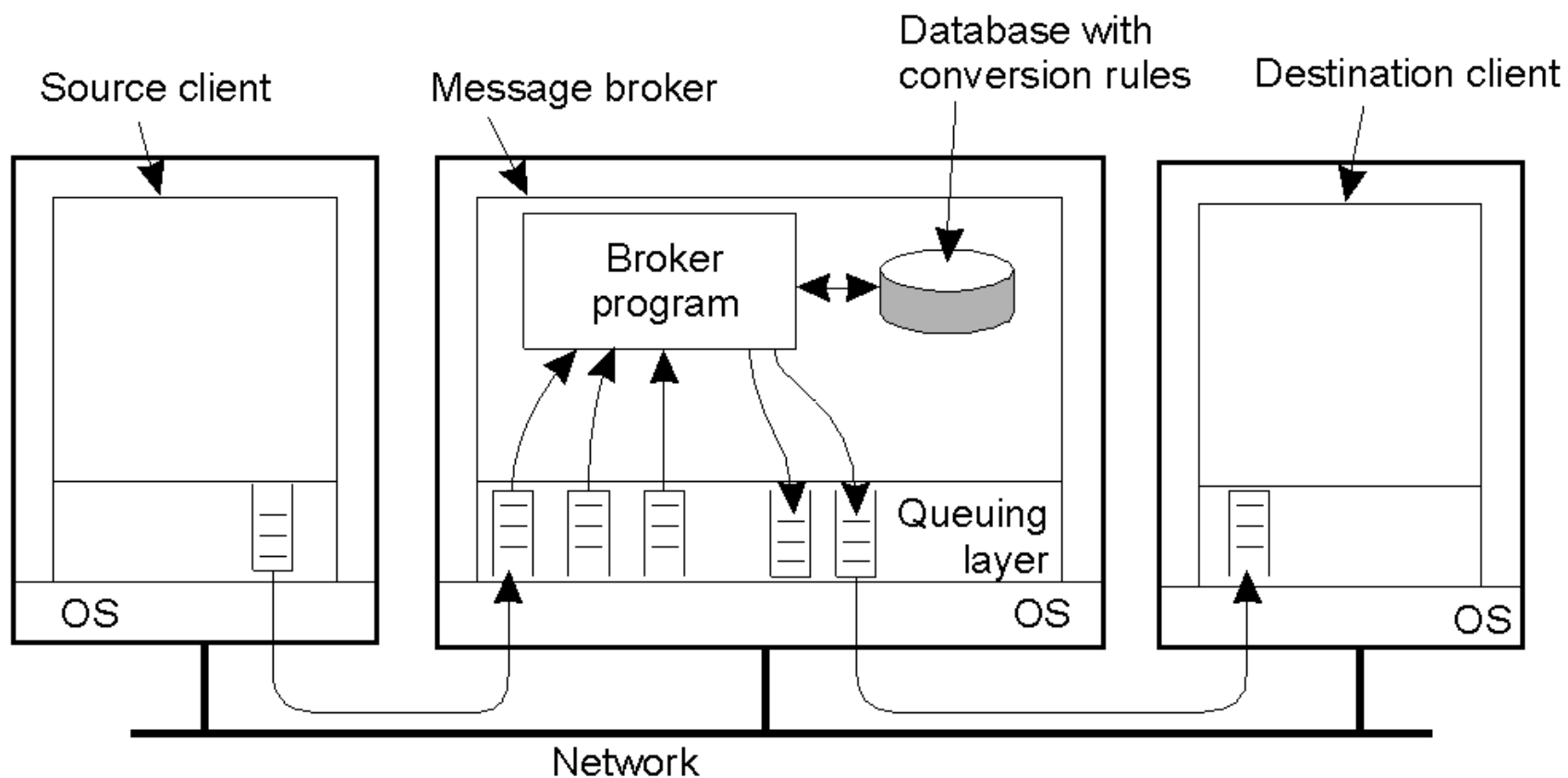


The general organization of a message-queuing system with routers.

# Heterogeneous Message Format

- The sender and receiver have to agree on the message format.
- This approach may work for other protocols but not for message passing.
- When it is not possible to have a common format, a re-formatter is needed.
- One example is the distributed heterogeneous database systems.
  - A query may need to be executed on more than one database.
  - The query is divided into several sub-queries and the results are collected at the end.

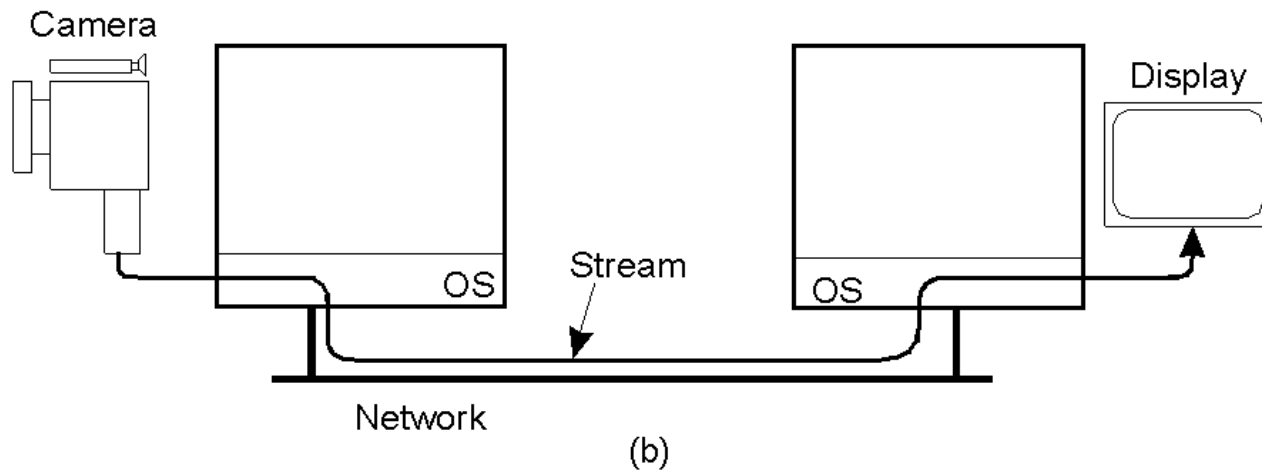
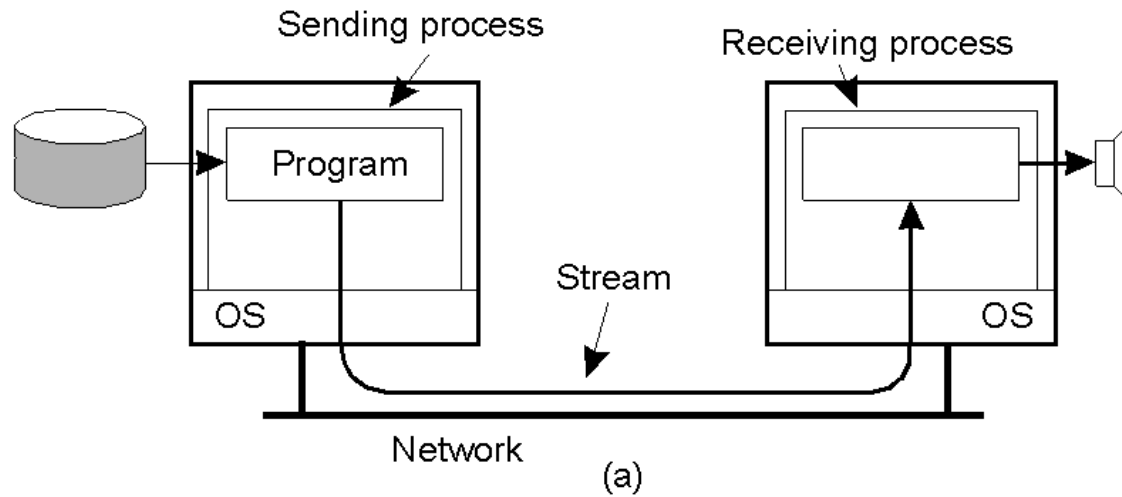
# Message Brokers

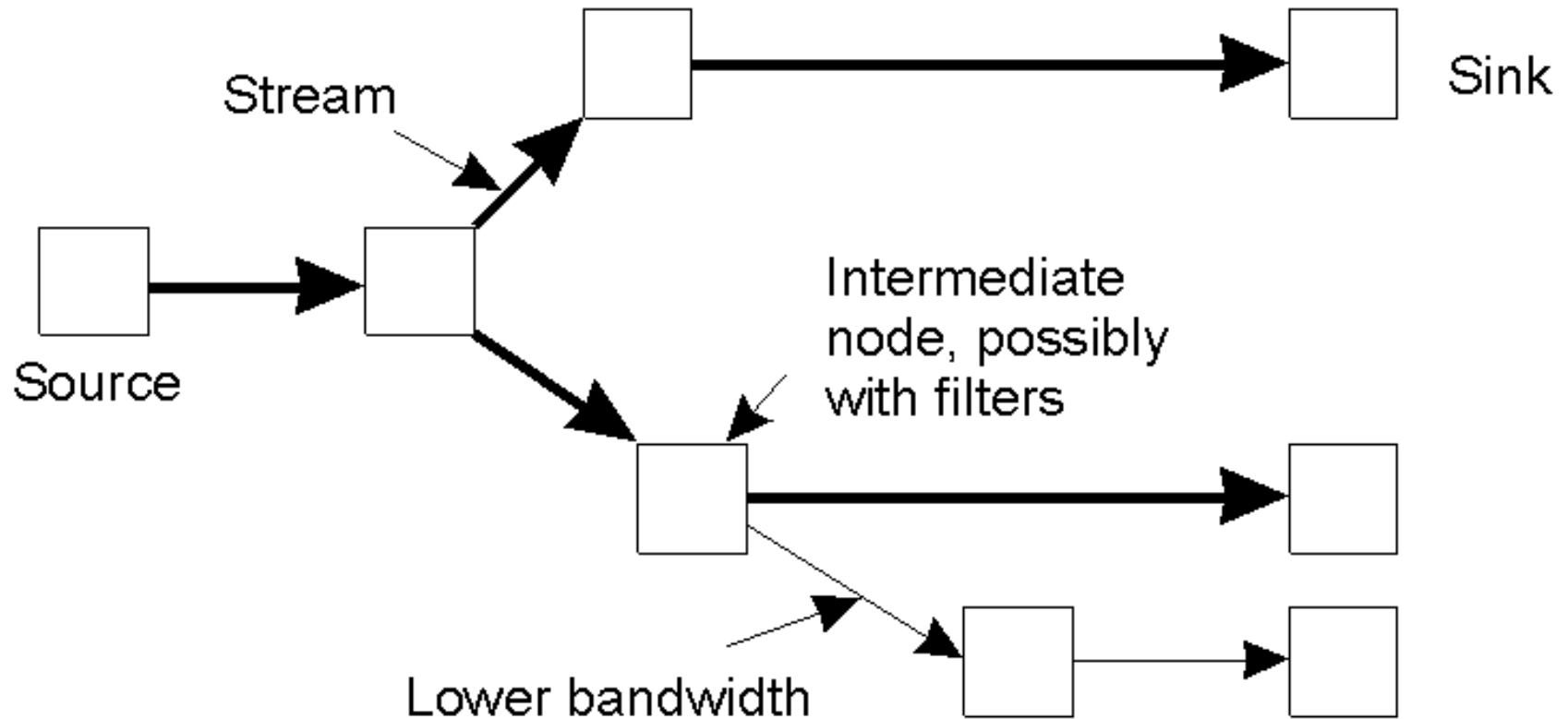


# Stream-Oriented Communications

- Timing has no effect on correctness for RPC, RMI, and message passing.
- In many applications, timing plays a crucial role. One example is audio/video stream.
- Transmission modes:
  - Asynchronous transmission mode
  - Synchronous transmission mode
  - Isochronous transmission mode
- Stream types:
  - Simple stream
  - Complex stream

# Data Streams





- Filters can be used to adjust the QoS level of the streams from high quality to low quality.

# Quality of Service for Streams

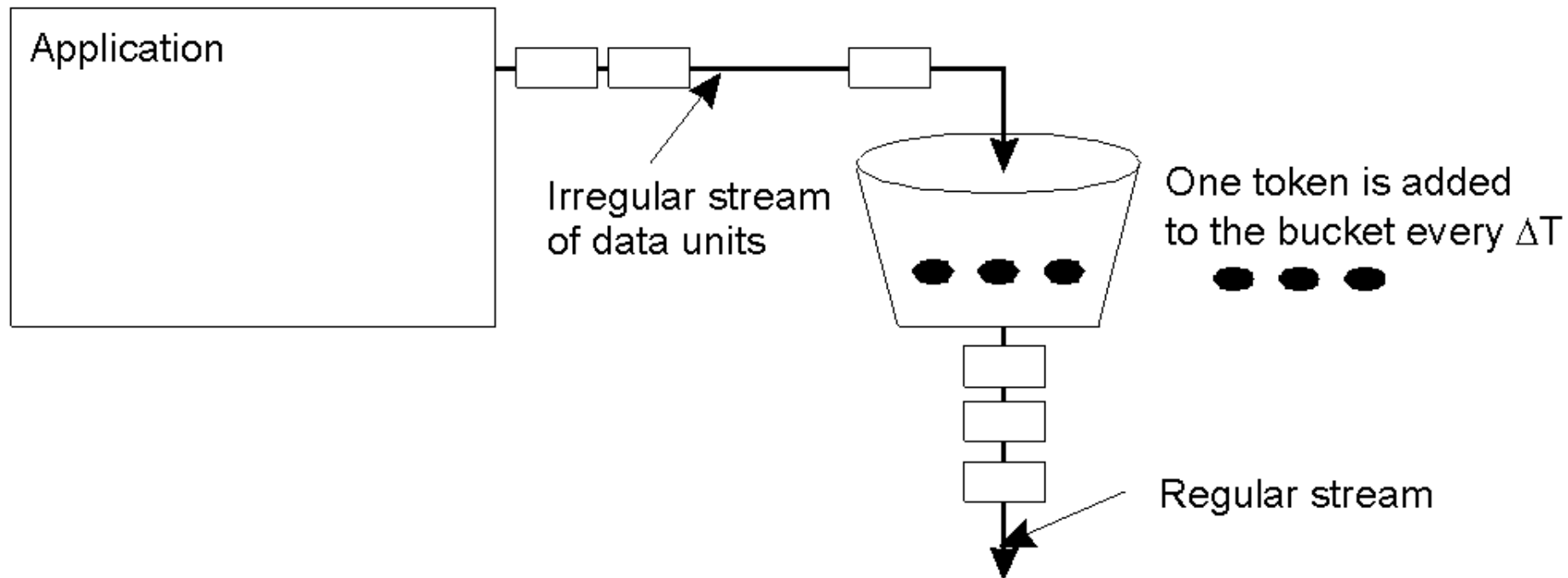
- QoS can be used to specify the time-dependent (or other non-functional) requirements.
- QoS Application Level: Flow Specification
  - Time-dependent requirements
  - Services requirements:
    - Loss Sensitivity
      - Loss interval
      - Loss sensitivity
    - Minimal delay noticed
    - Maximum delay variation
    - Quality of guarantee:

Characteristics of the Input	Service Required
<ul style="list-style-type: none"><li>? <b>Maximum data unit size (bytes)</b></li><li>? <b>Token bucket rate (bytes/sec)</b></li><li>? <b>Token bucket size (bytes)</b></li><li>? <b>Maximum transmission rate (bytes/sec)</b></li></ul>	<ul style="list-style-type: none"><li>? <b>Loss sensitivity (bytes)</b></li><li>? <b>Loss interval (<math>\mu</math>sec)</b></li><li>? <b>Burst loss sensitivity (data units)</b></li><li>? <b>Minimum delay noticed (<math>\mu</math>sec)</b></li><li>? <b>Maximum delay variation (<math>\mu</math>sec)</b></li><li>? <b>Quality of guarantee</b></li></ul>

## Flow Specification



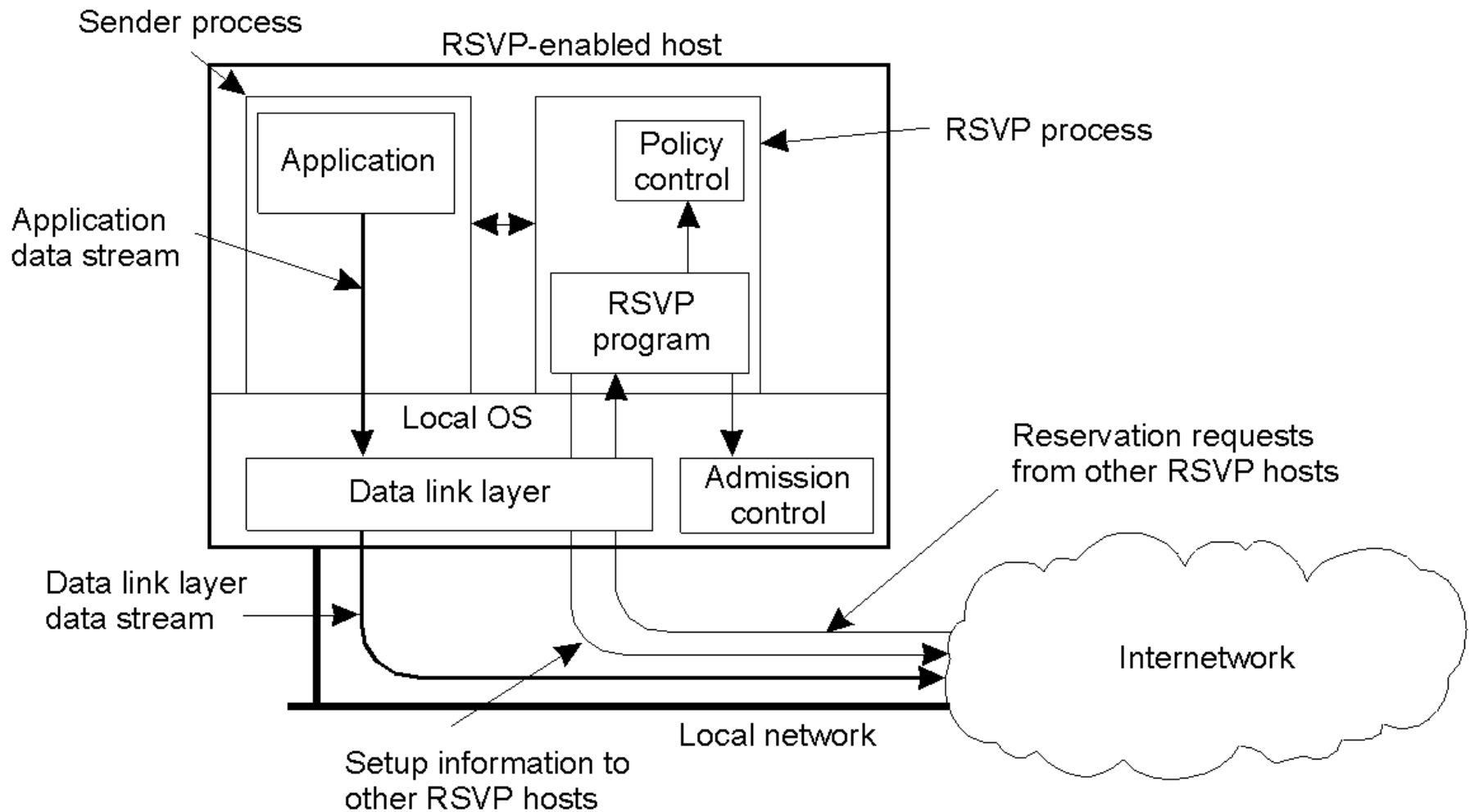
# Token Bucket Algorithm



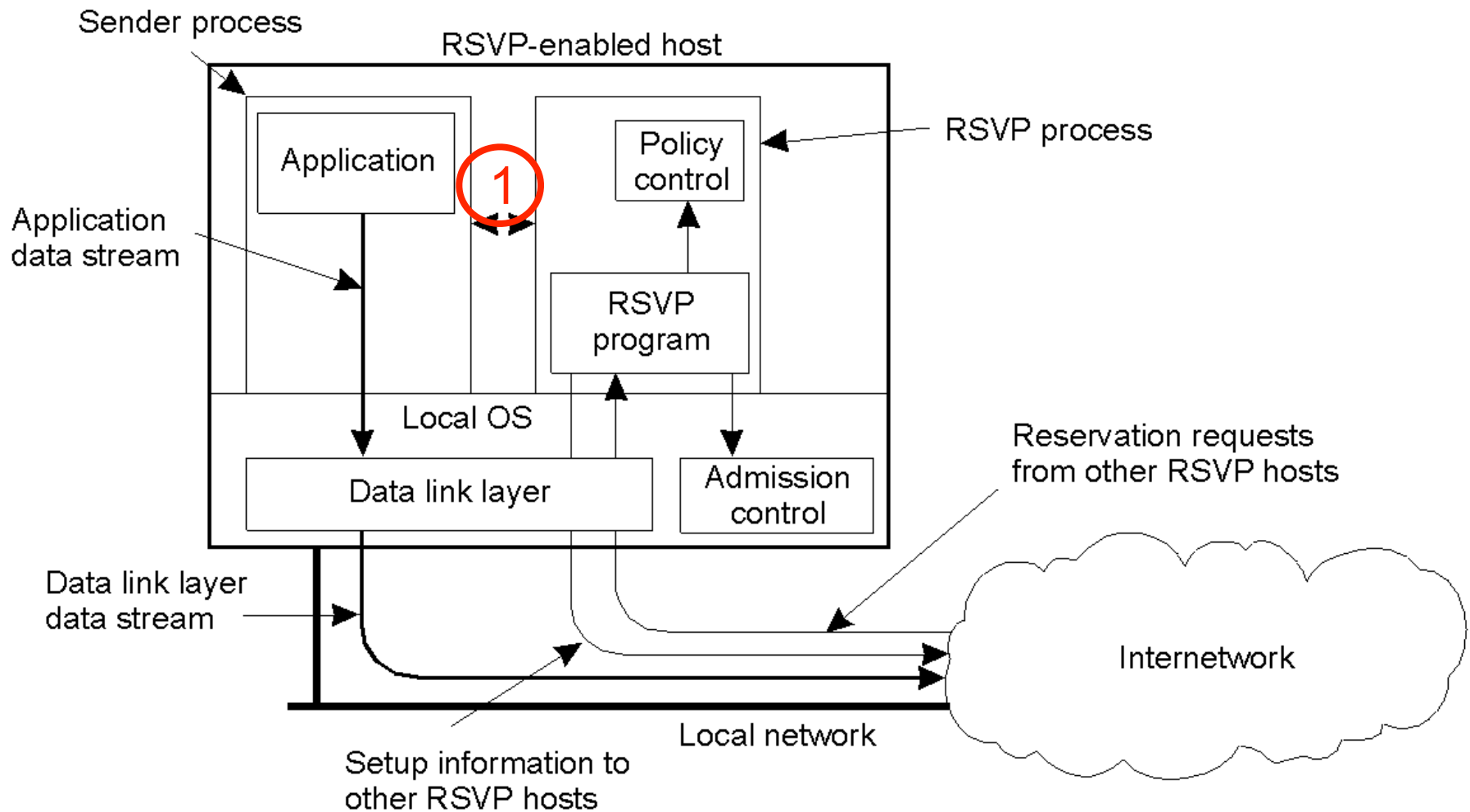
# Quality of Service Specification

- Such detailed flow specification will never be acceptable for end-users.
- DSLR vs. Point-and-Shot (or latest digital camera)
- To be a good engineer, you have to talk to the end-users.
- To guarantee the QoS, the resources have to be reserved. However, there is
  - no universal model for specifying QoS parameters,
  - no generic model for describing the resources, and
  - no model for translating QoS parameters to resource usage.

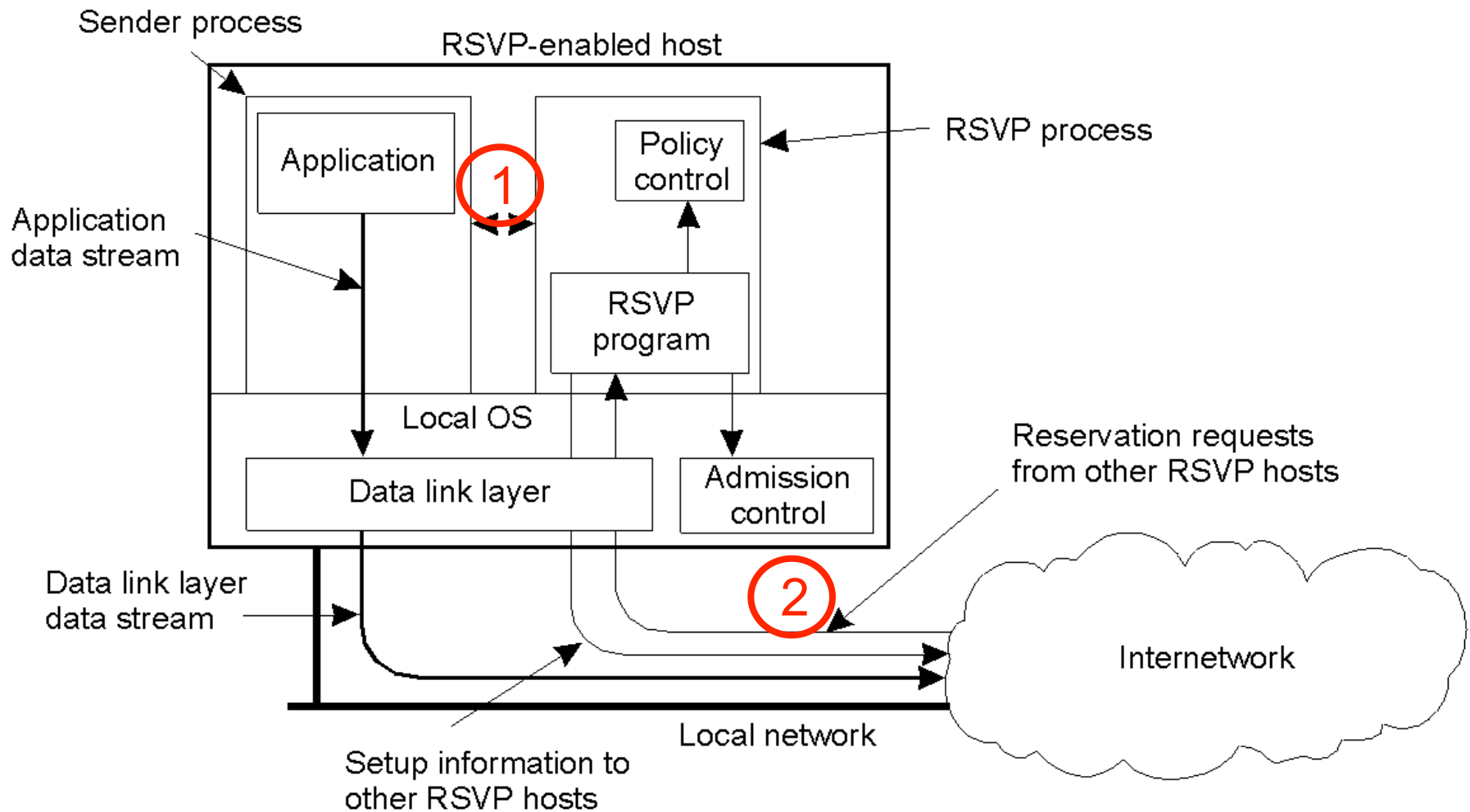
# Resource Reservation Model - RSVP



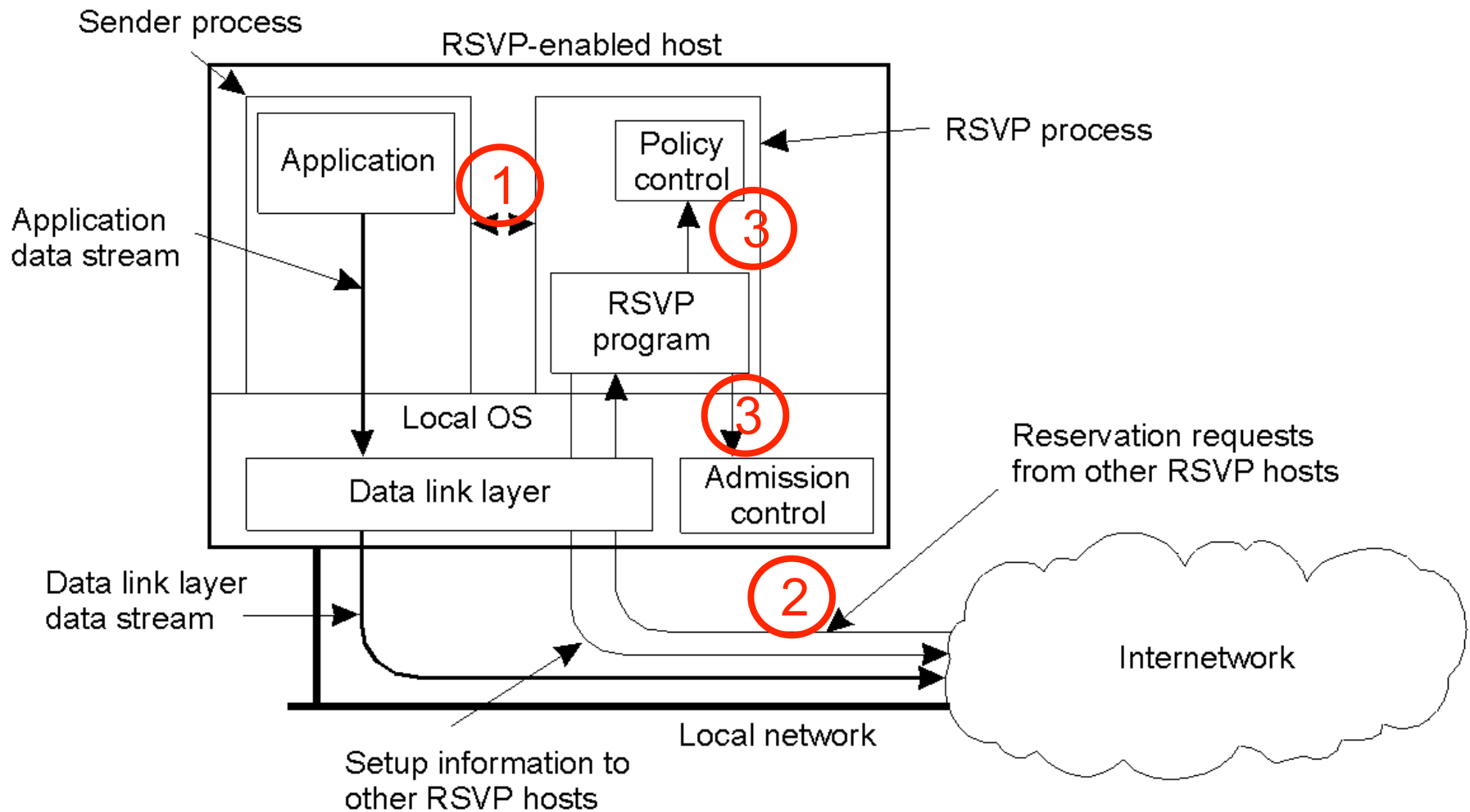
# Resource Reservation Model - RSVP



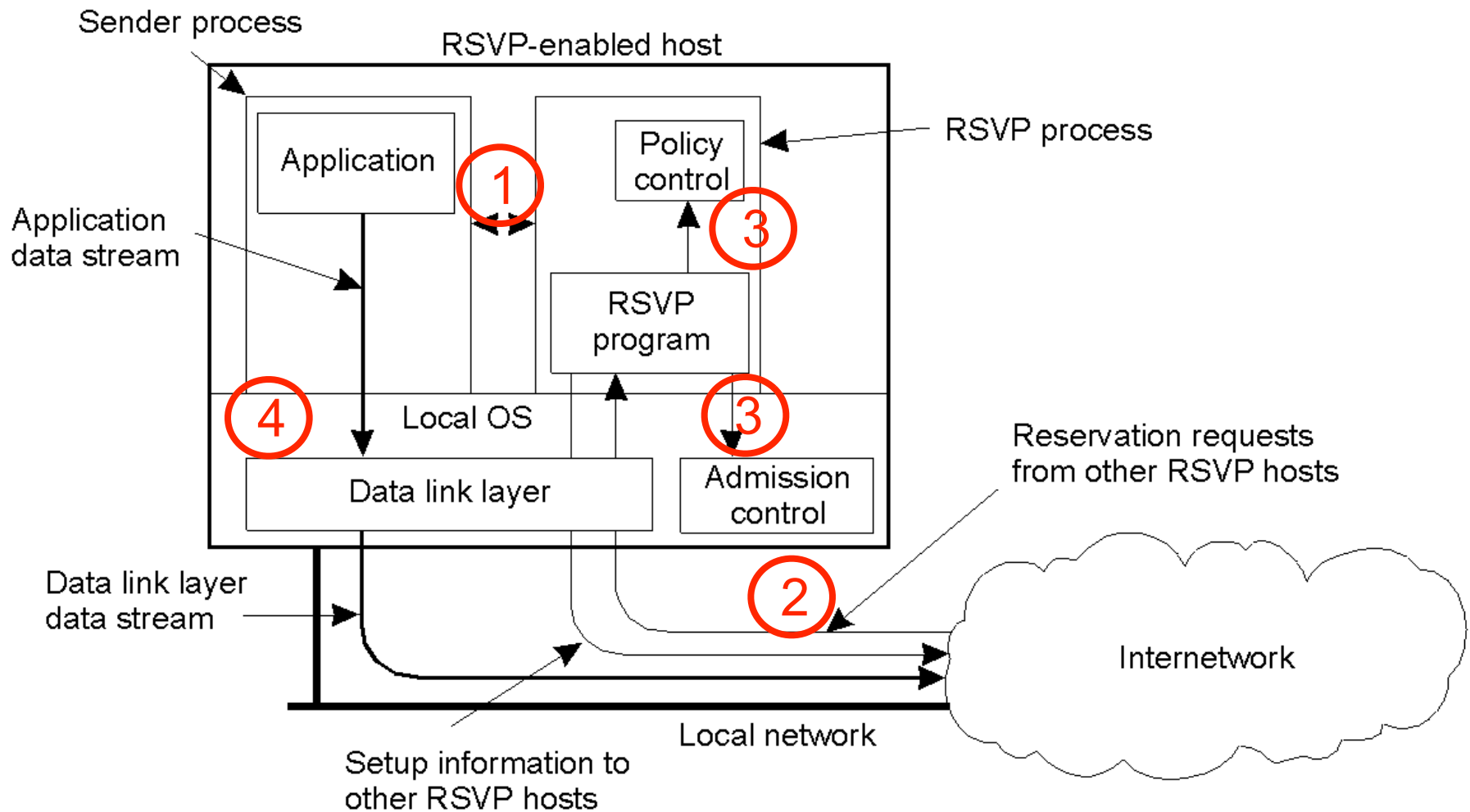
# Resource Reservation Model - RSVP



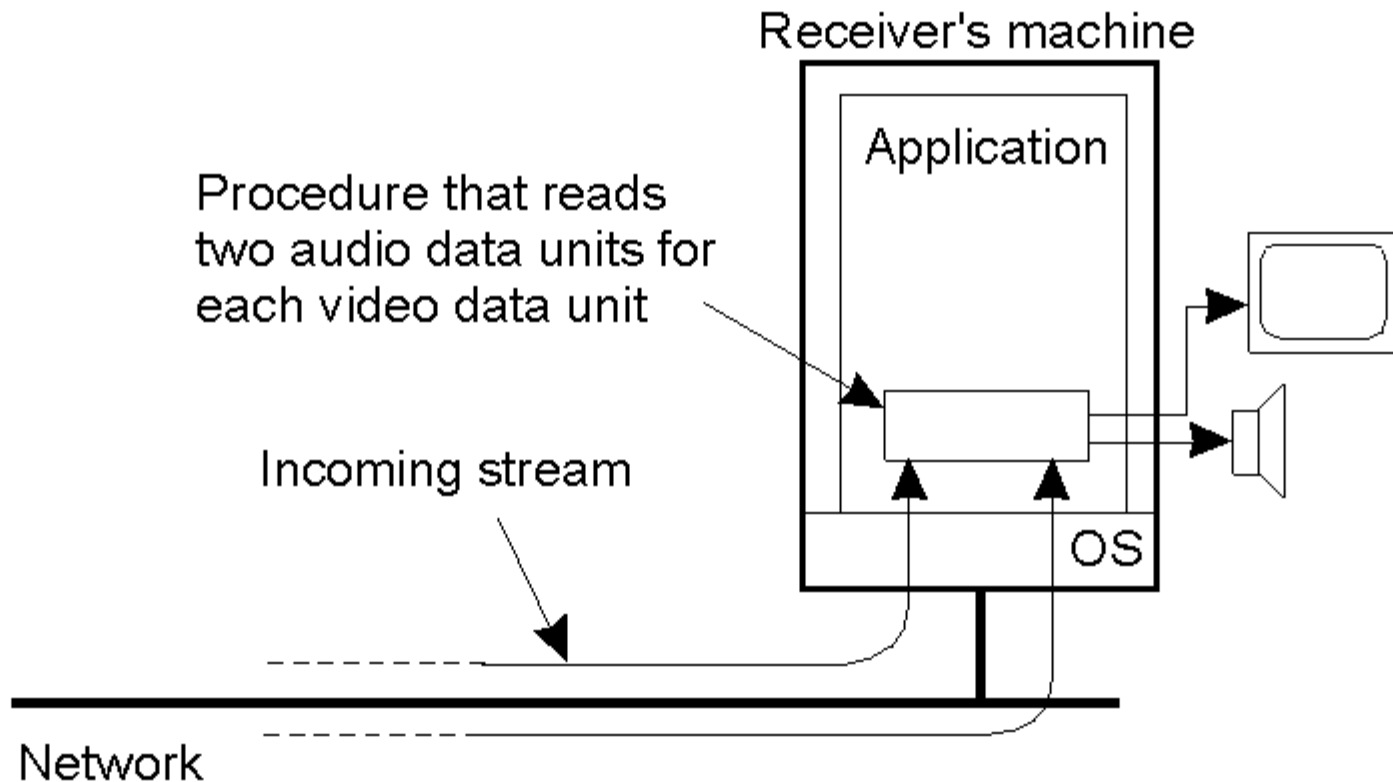
# Resource Reservation Model - RSVP



# Resource Reservation Model - RSVP



# Synchronization Mechanisms

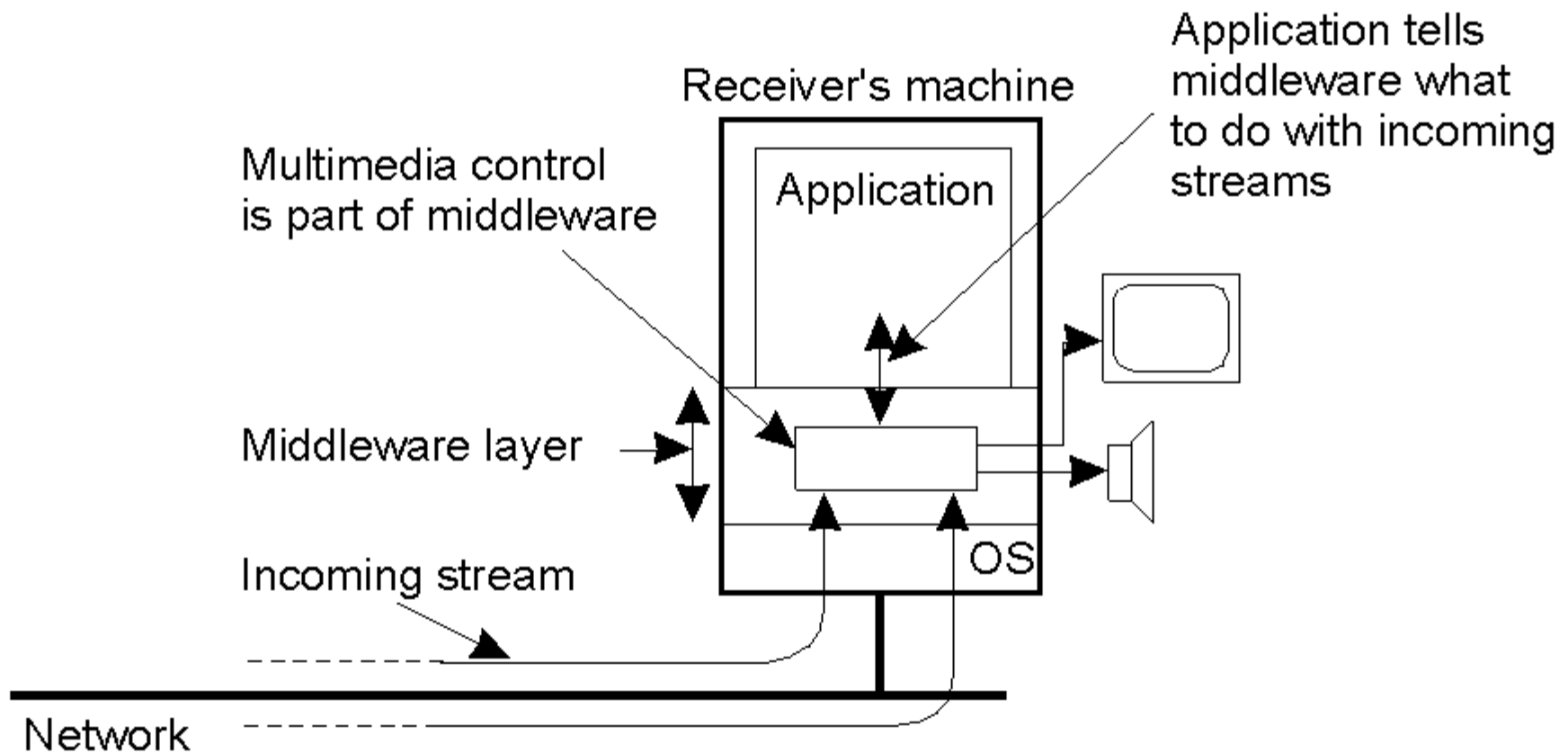


- The image and audio have to read alternatively for every 33 msec.
- The application is responsible for synchronizing the streams.

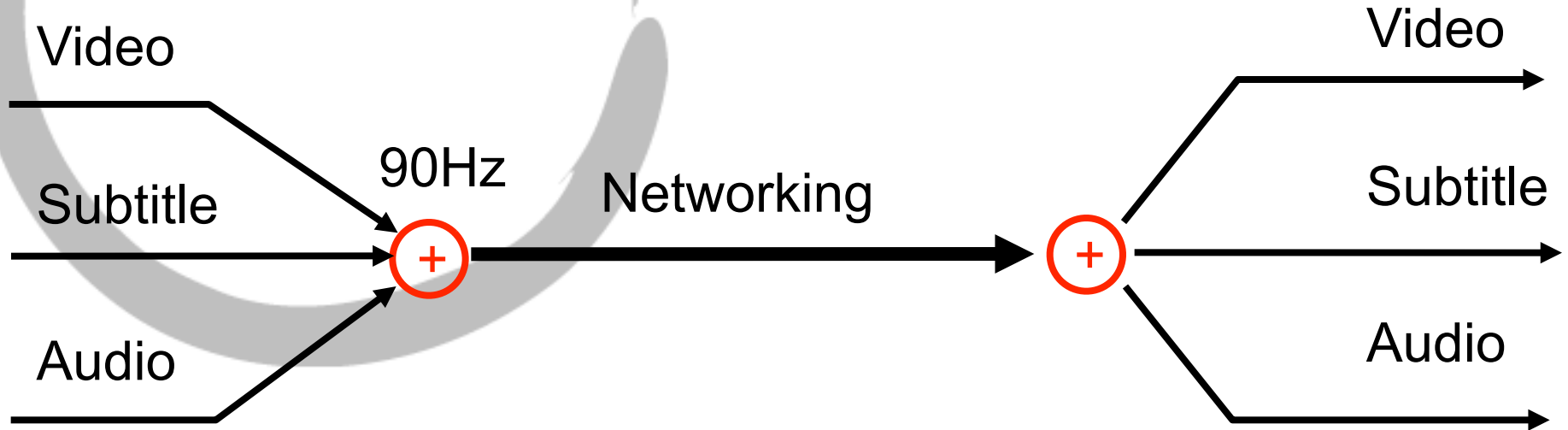


# Synchronization Mechanisms (2)

- The principle of synchronization as supported by high-level interfaces.



# MPEG2 Synchronization Protocol



# Summary

- Powerful and flexible facilities for communication are essential for any distributed system.
- RPCs are aimed at achieving access transparency.
- RMI allows system-wide object reference to be passed as parameters.
- Message-oriented protocol provides more general purpose and high-level model.
- Communication could be
  - Persistent or temporary
  - Synchronous or Asynchronous
- Stream-oriented model aims at timing requirement for end-to-end communication.

# Reading Lists

- Reading Lists for RPC and Message passing:
  - Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy. 1990. Lightweight remote procedure call. *ACM Trans. Comput. Syst.* 8, 1 (February 1990), 37-55.
  - D. D. Clark and D. L. Tennenhouse. 1990. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures & protocols (SIGCOMM '90)*. ACM, New York, NY, USA, 200-208.
  - T. von Eicken, A. Basu, V. Buch, and W. Vogels. 1995. U-Net: a user-level network interface for parallel and distributed computing (includes URL). *SIGOPS Oper. Syst. Rev.* 29, 5 (December 1995), 40-53.
  - Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. 1992. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th annual international symposium on Computer architecture (ISCA '92)*. ACM, New York, NY, USA, 256-266.
  - R. Graham, T. Woodall, and J. Squyres, Open MPI: A Flexible High Performance MPI, *Lecture Notes in Computer Science*, 2006, Volume 3911, pp. 228-239 (2006)
  - Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. *ACM Comput. Surv.* 35, 2 (June 2003), 114-131.
  - Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur, Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming, *International Journal of High Performance Computing Applications* February 2010 vol. 24 no. 1 49-57.
  - Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named data networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 66-73.

# Efficiently Reading Scientific Papers (1/2)

- Read for breadth
  - **What did they do?**
  - **Skim introduction, headings, graphics, definitions, conclusions and bibliography.**
  - **Consider the credibility.**
  - **How useful is it?**
  - **Decide whether to go on.**
- Read in depth
  - **How did they do it?**
  - **Challenge their arguments.**
  - **Examine assumptions.**
  - **Examine methods.**
  - **Examine statistics.**
  - **Examine reasoning and conclusions.**
  - **How can I apply their approach to my work?**

# Efficiently Reading Scientific Papers (2/2)

- Take notes
  - **Make notes as you read.**
  - **Highlight major points.**
  - **Note new terms and definitions.**
  - **Summarize tables and graphs.**
  - **Write a summary.**
- Reference:
  - “How to read a computer science research paper” by Admanda Stent.
  - 如何閱讀一本書，Mortimer J. Adler and Charles Van Doren 著，郝明義與朱衣譯，台灣商務印書館。

# Writing Critics

- You may want to including the following topics but not limited to.
  - **How useful is it? What's its contribution?**
  - **Highlight major points/contribution:**
  - **Questioning and discussing its**
    - Assumptions,
    - Methods,
    - Statistics, and
    - reasoning and conclusions.
- Proof read it before you submit.