Communication in Distributed Systems – Programming

Hong-Linh Truong
Distributed Systems Group,
Vienna University of Technology

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong

Learning Materials

- Main reading:
  - Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall
    - Chapters 3 & 4
- Others
  - George Coulouris, Jean Dollimore, Tim Kindberg, „Distributed Systems – Concepts and Design“, 2nd Edition
    - Chapter 5.
  - Sukumar Ghosh, “Distributed Systems: An Algorithmic Approach”, Chapman and Hall/CRC, 2007
    - Chapter 15
  - Papers referred in the lecture
- Test the examples in the lecture

Outline

- Recall
- Message-oriented Transient Communication
- Message-oriented Persistent Communication
- Remote Procedure Call
- Streaming data programming
- Group communication
- Gossip-based Data Dissemination
- Summary

Recall

- One-to-one versus group communication
- Transient communication versus persistent communication
- Message transmission versus procedure call versus object method calls
- Physical versus overlay network

MESSAGE-ORIENTED TRANSIENT COMMUNICATION

Message-oriented Transient Communication at Transport Layer

- How an application uses the transport layer communication to send/receive messages?

Transport-level socket programming via socket interface

- Socket interface – Socket APIs
  - Very popular, supported in almost all programming languages and operating systems
  - Berkeley Sockets (BSD Sockets)
    - Java Socket, Windows Sockets API/WinSock, etc.

Message-oriented Transient Communication at Transport Level (2)

What is a socket: a communication end point to/from which an application can send/receive data through the underlying network.

- Client
  - Connect, send and then receive data through sockets
- Server:
  - Bind, listen/accept, receive incoming data, process the data, and send back to the client the result

Q: Which types of information are used to describe the identifier of the “end point”?

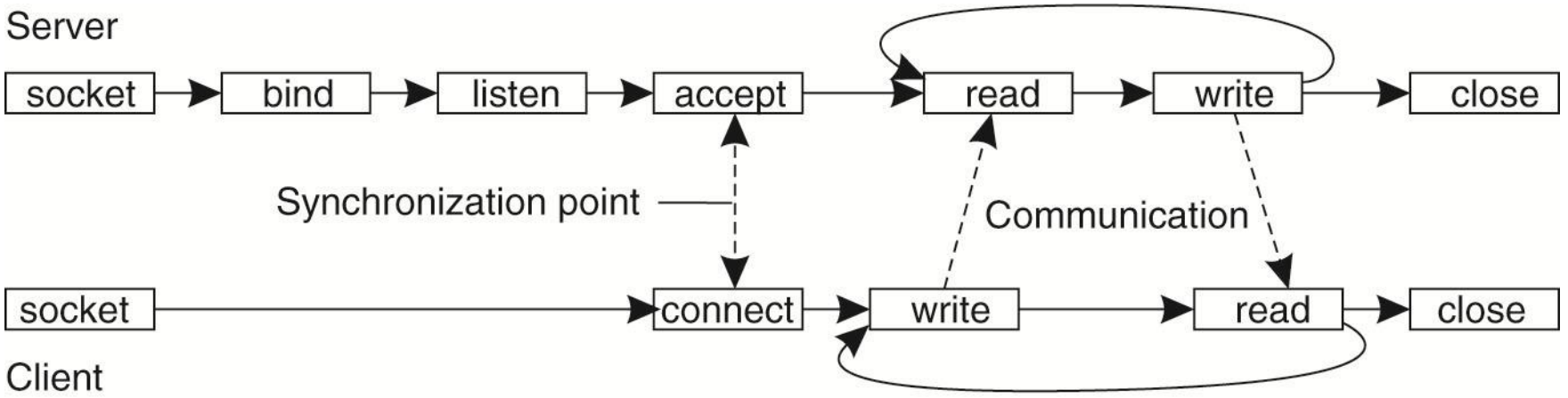
Socket Primitives

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
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

Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Client-server interactions

Connection-oriented communication interaction



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

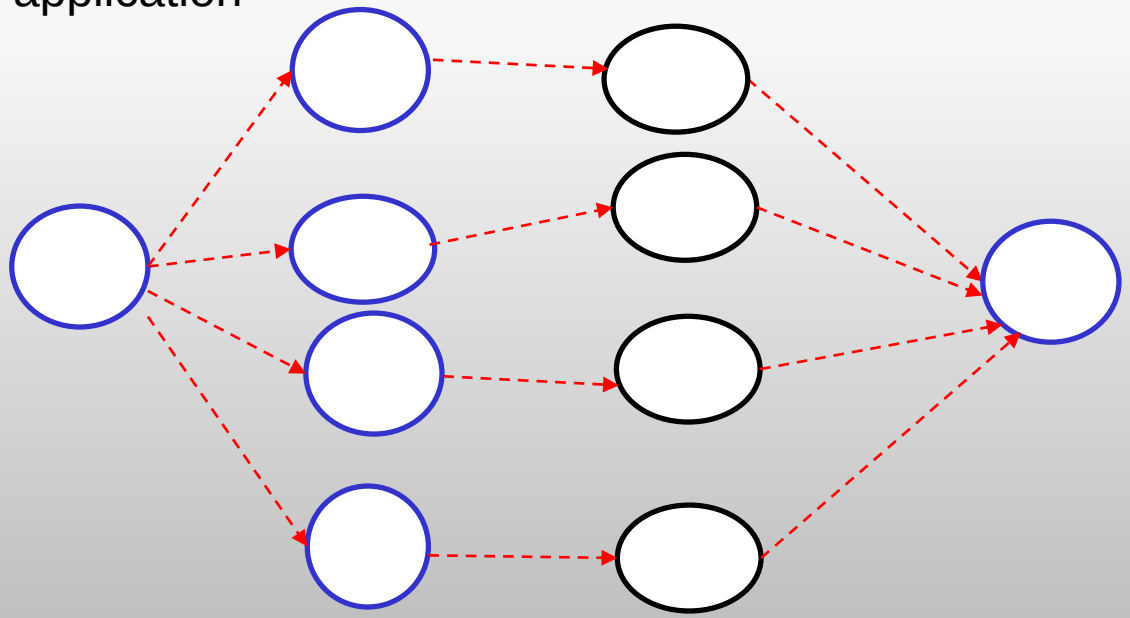
Q1: How to implement a multi-threaded server?
Q2: What if connect() happens before listen()/accept()?

Example

- Simple echo service
  - Client sends a message to a server
  - Server returns the message
- Source code: http://www.infosys.tuwien.ac.at/teaching/courses/VerteilteSysteme/exs/socket-ex.tar.gz

Message-oriented Transient Communication at the Application level

Complex communication, large-scale number processes in the same application



Why transport level socket programming primitives are not good enough?

Message-passing Interface (MPI)

- Designed for parallel processing: http://www.mpi-forum.org/
- Well supported in clusters and high performance computing systems
- One-to-one/group and synchronous/asynchronous communication

- Basic MPI concepts
  - Communicators/groups to determine a set of processes that can be communicated: MPI_COMM_WORLD represents all mpi processes
  - Rank: a unique identifier of a process
  - A set of functions to manage the execution environment
  - Point-to-point communication functions
  - Collective communication functions
  - Functions handling data types

Message-passing Interface (MPI)

Function	Description
MPI_Init	Initialize the MPI execution environment
MPI_Comm_size	Determine the size of the group given a communicator
MPI_Comm_rank	Determine the rank of the calling process in group
MPI_Send()	Send a message, blocking mode
MPI_Recv()	Receive a message, blocking mode
…	
MPI_Bcast()	Broadcast a message from a process to others
MPI_Reduce()	Reduce all values from all processes to a single value
…	
MPI_Finalize()	Terminate the MPI execution environment

Example

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if(myid == 0)  {
    printf("I am %d: We have %d processors\n", myid,
      numprocs);
    sprintf(output, "This is a message sending from %d",
      i);
    for(i=1;i<numprocs;i++)
      MPI_Send(output, 80, MPI_CHAR, i, 0,
       MPI_COMM_WORLD);
  }
  else {
     MPI_Recv(output, 80, MPI_CHAR, i, 0,
      MPI_COMM_WORLD, &status);
     printf("I am %d and I receive: %s\n", myid, output);
  }
```

```
source=0;
  count=4;
  if(myid == source){
    for(i=0;i<count;i++)
      buffer[i]=i;
  }

    MPI_Bcast(buffer,count,MPI_INT,source,MPI_COM
    M_WORLD);
  for(i=0;i<count;i++) {
    printf("I am %d and I receive: %d \n",myid, buffer[i]);
  }
  printf("\n");
  MPI_Finalize();
```

Code: http://www.infosys.tuwien.ac.at/teaching/courses/VerteilteSysteme/exs/mpi-ex.tar.gz

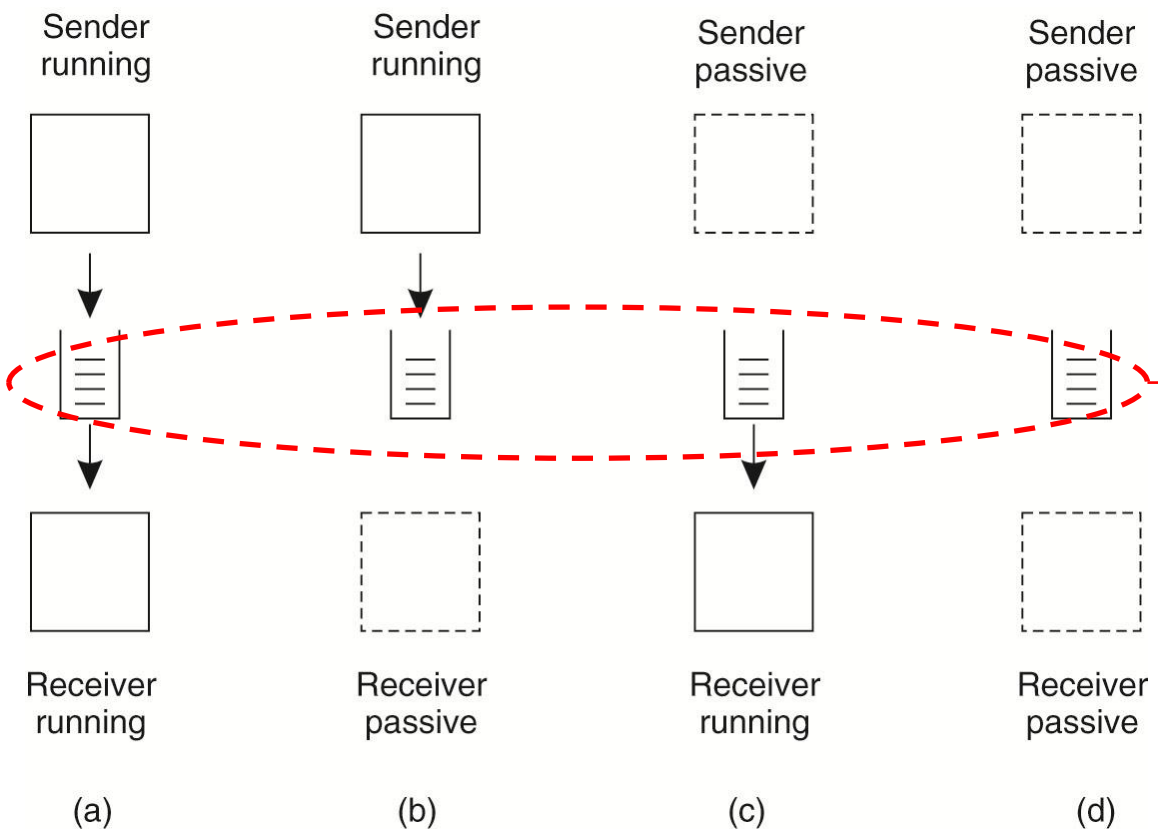
MESSAGE-ORIENTED PERSISTENT COMMUNICATION

Message-oriented Persistent Communication – Queuing Model

- Message-queuing systems or Message-Oriented Middleware (MOM)
- Well-supported in large-scale systems for
  - Persistent but asynchronous messages
  - Scalable message handling
- Several Implementations

Message-oriented Persistent Communication – Queuing Model

Communication models

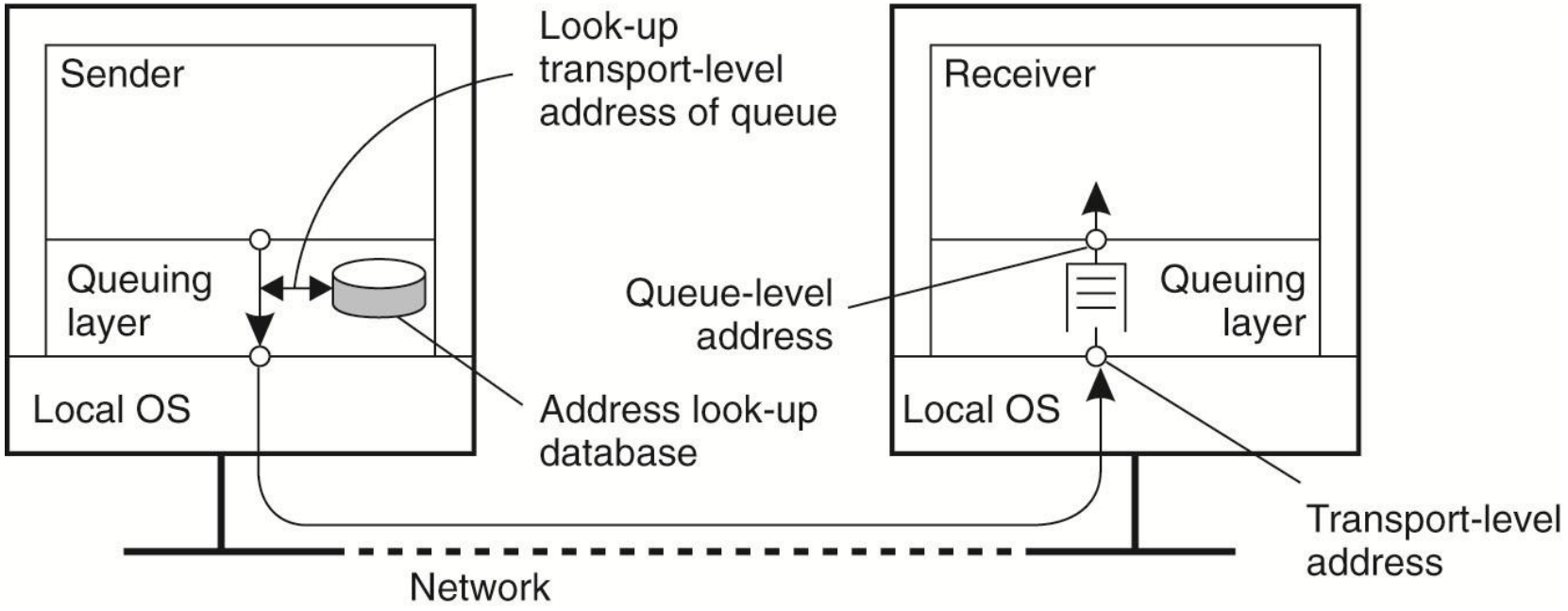


Q1: Give an example of case (d)

Operations
PUT
GET
POLL
NOTIFY

Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

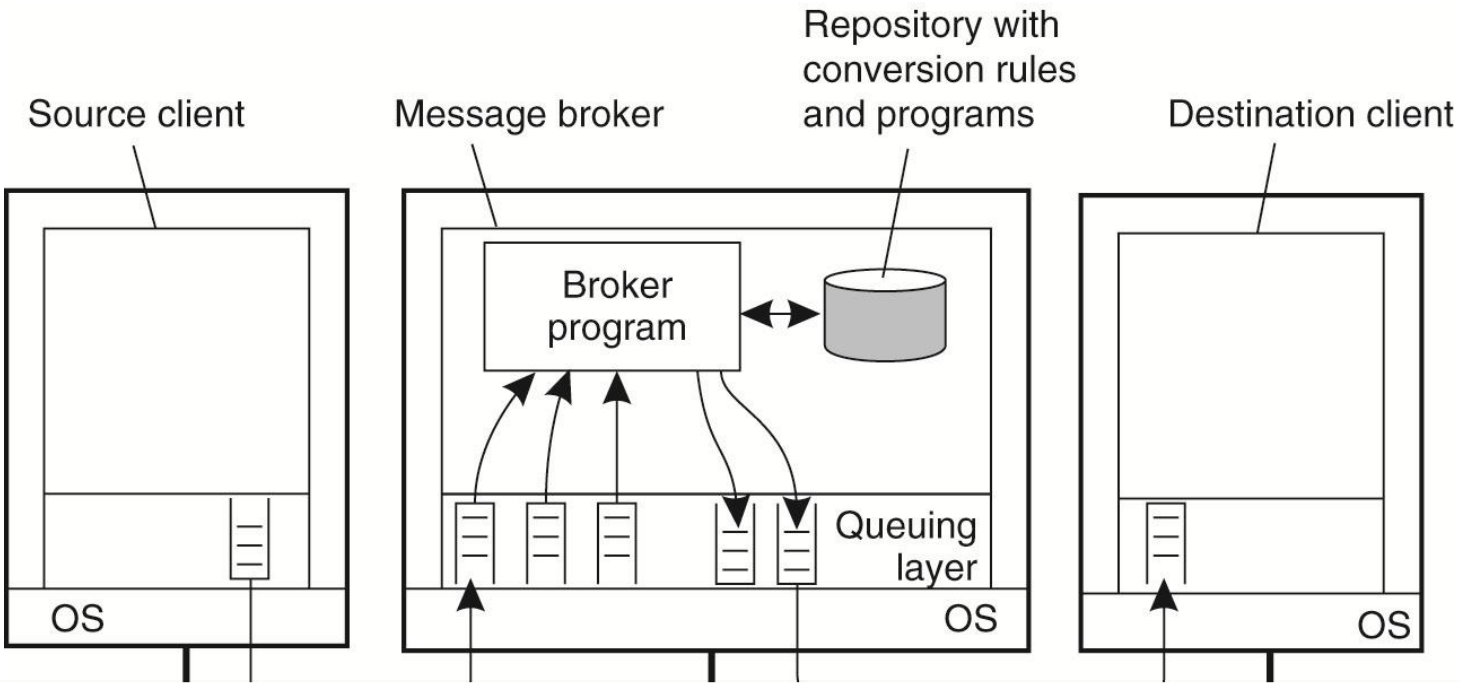
Message-oriented Persistent Communication – Queuing Model



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Message Brokers

- Publish/Subscribe: messages are matched to applications
- Transform: messages are transformed from one format to another one suitable for specific applications



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Example – Advanced Message Queuing Protocol (AMQP)

- http://www.amqp.org

Apache Qpid™

Example: AMQP

```
ConnectionFactory factory = new ConnectionFactory();
    factory.setUri(uri);
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();


    channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    for (int i=0; i<100; i++) {
        String message = "Hello distributed systems guys:  "+i;
        channel.basicPublish("", QUEUE_NAME, null,
        message.getBytes());
        System.out.println(" [x] Sent '" + message + "'");
        new Thread().sleep(5000);
    }


    channel.close();
    connection.close();
```

Source code: https://github.com/cloudamqp/java-amqp-example, see also the demo in the lecture 2

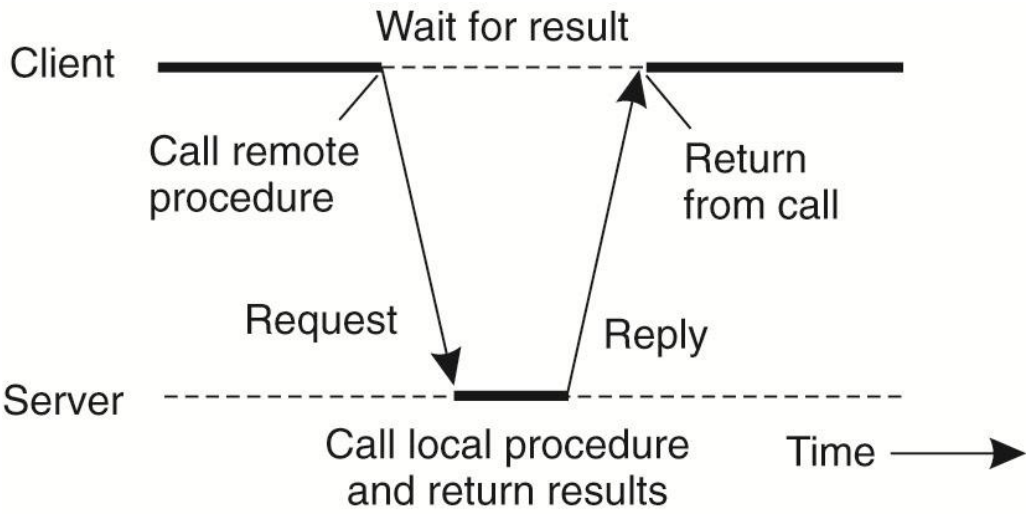
```
ConnectionFactory factory = new ConnectionFactory();
    factory.setUri(uri);
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();


    channel.queueDeclare(QUEUE_NAME, false, false,
       false, null);
    System.out.println(" [*] Waiting for messages");


    QueueingConsumer consumer = new
       QueueingConsumer(channel);
    channel.basicConsume(QUEUE_NAME, true,
       consumer);


    while (true) {
      QueueingConsumer.Delivery delivery =
       consumer.nextDelivery();
      String message = new String(delivery.getBody());
      System.out.println(" [x] Received '" + message + "'");
    }
```

REMOTE PROCEDURE CALL

Remote Procedure Call

How can we call a procedure in a remote process in a similar way to a local procedure?

Remote Procedure Call (RPC): hide all complexity in calling remote procedures



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

- Well support in many systems and programming languages

Q1: Which types of applications are suitable for RPC?

Message format and data structure description

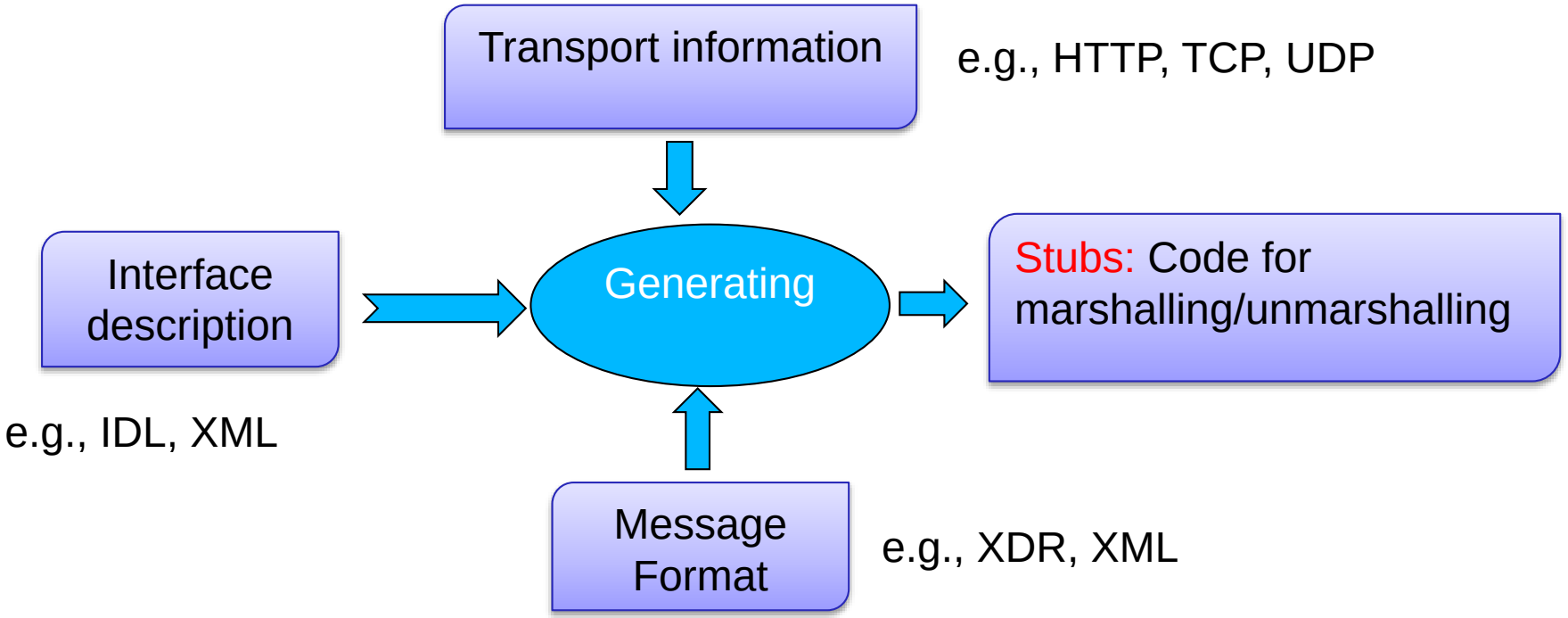
- Passing parameters and results needs agreed message format between a client and a server

Marshaling/unmarshalling describes the process packing/unpacking parameters into/from messages
(note: encoding/decoding are also the terms used)

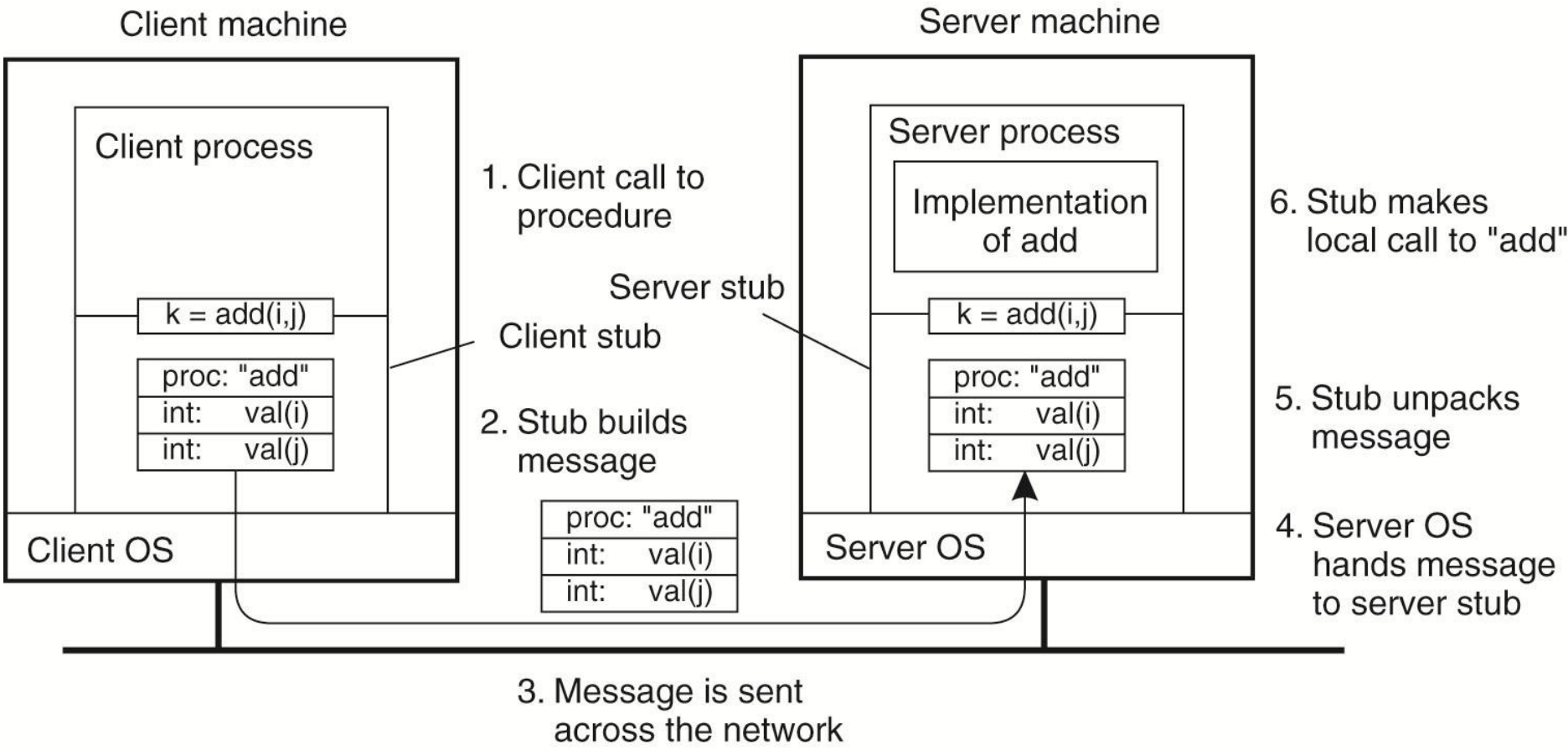
- Data types may have different representations due to different machine types (e.., SPARC versus Intel x86)

Interface languages can be used to describe the common interfaces between clients and server

Generating stubs

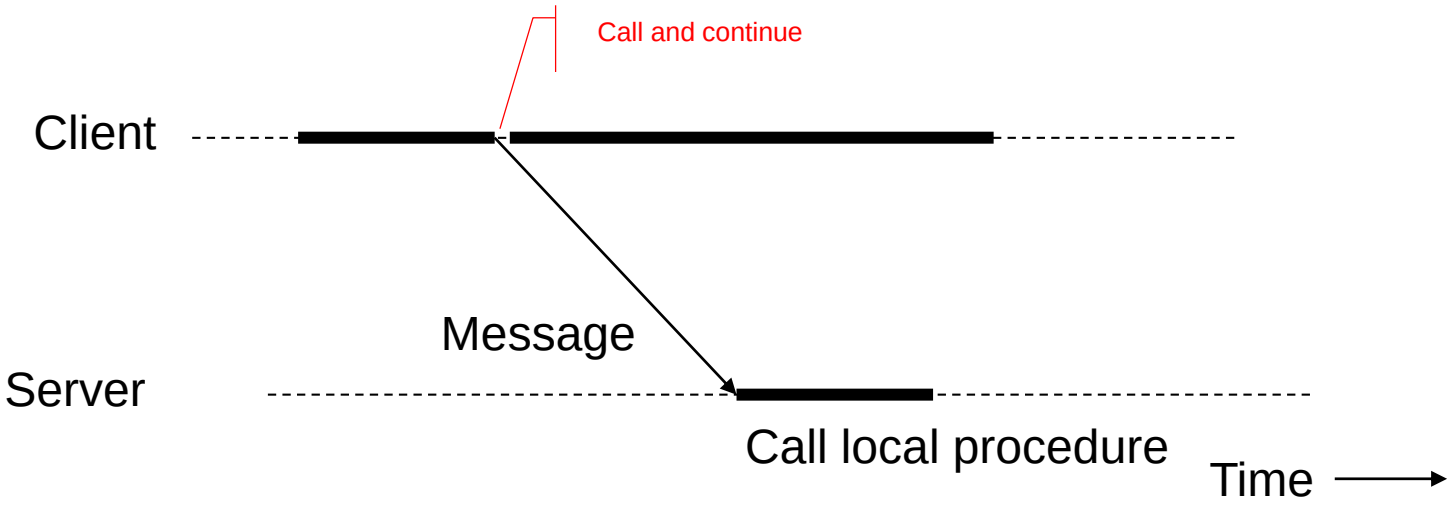


Detailed Interactions



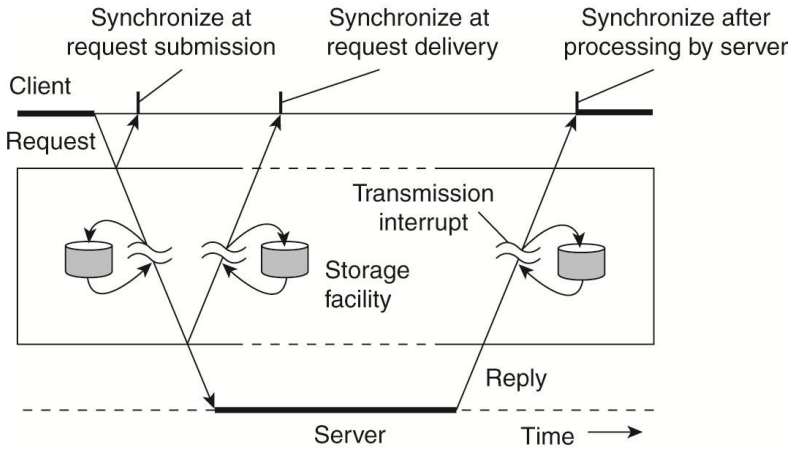
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

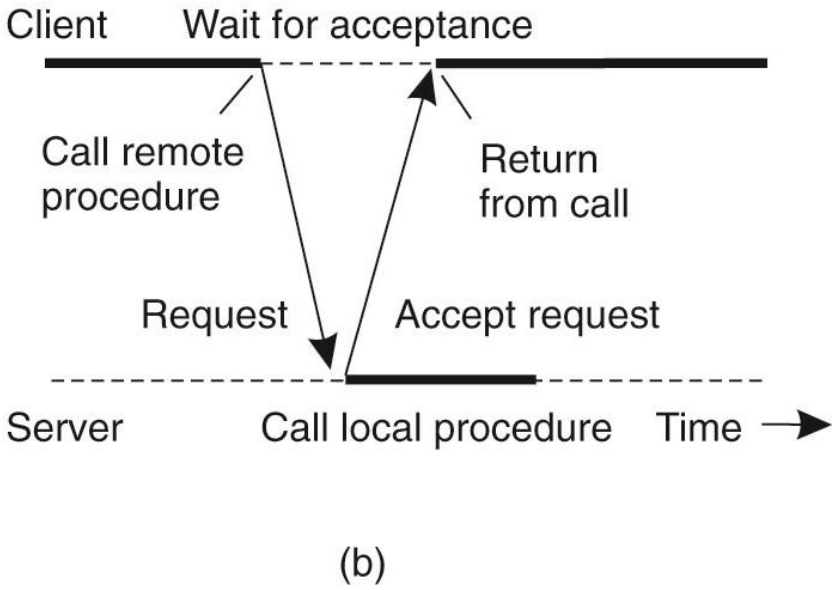
One-way RPC


Call and continue
Client
Message
Server
Call local procedure
Time

Asynchronous RPC

Recall: (A)synchronous communication
Q1: How can we implement asynchronous RPC



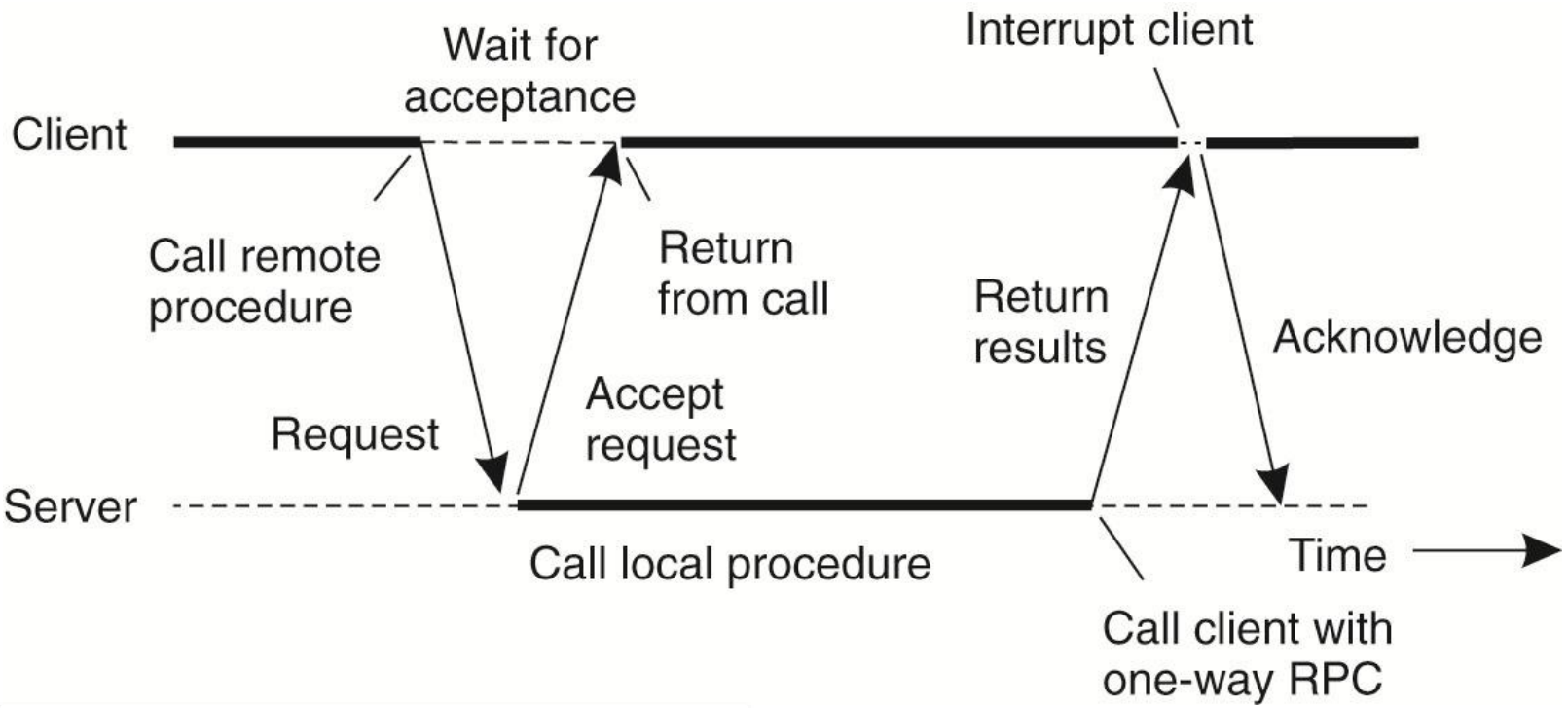


(b)

Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Asynchronous RPC

Two asynchronous RPC/ Deferred synchronous RPC



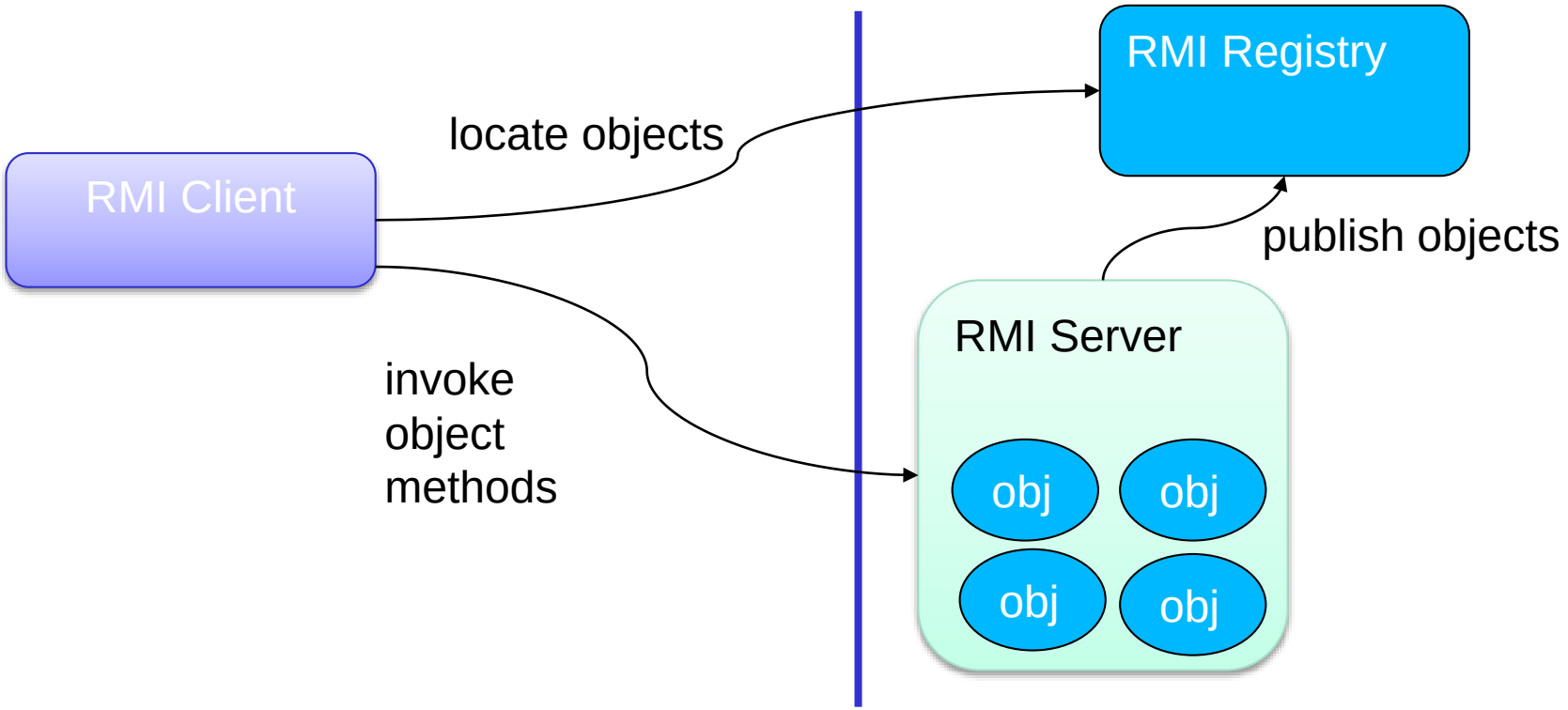
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Some RPC implementations

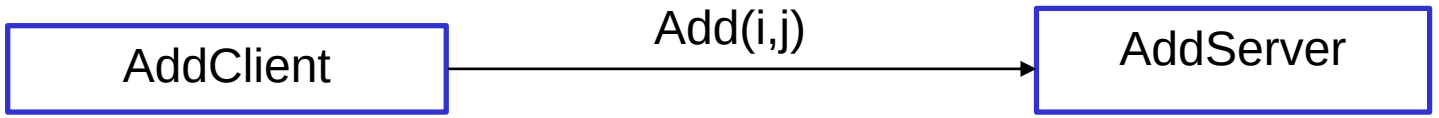
- rpcgen – SUN RPC
  - IDL for interface description
  - XDR for messages
  - TCP/UDP for transport
- XML-RPC
  - XML for messages
  - HTTP for transport
- JSON-RPC
  - JSON for messages
  - HTTP and/or TCP/IP for transport

Remote Meothd Invocation/Remote Object Call

- RPC style in Java
  - Remote object method invocation/call



Example of RPC




```
program ADD_PROG {
version ADD_VERS {
    int add(int , int ) = 1;
} = 1;
} = 0x23452345;
```

→ $rpcgen –N –a add.x →

- add.h
- add_xdr.c
- add_client.c
- add_clnt.c
- add_server.c
- add_svc.c

Code: http://www.infosys.tuwien.ac.at/teaching/courses/VerteilteSysteme/exs/rpcadd-ex.tar.gz

STREAMING DATA PROGRAMMING

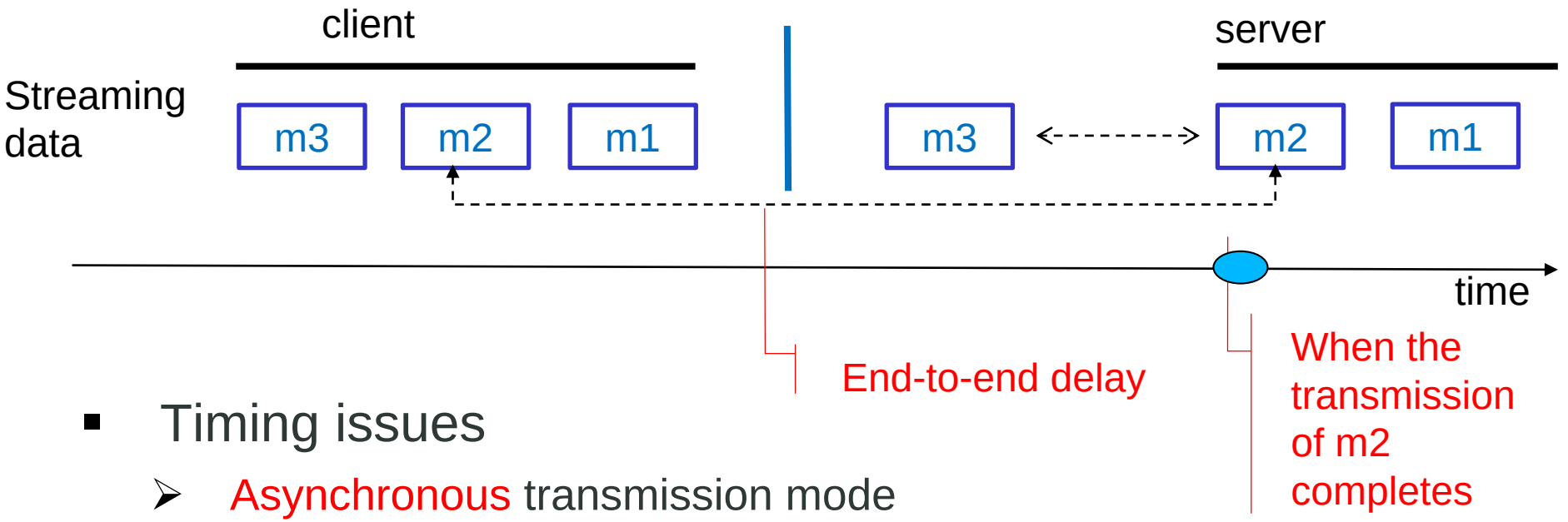
Data stream programming

Data stream: a sequence of data units

e.g. reading bytes from a file and send bytes via a TCP socket

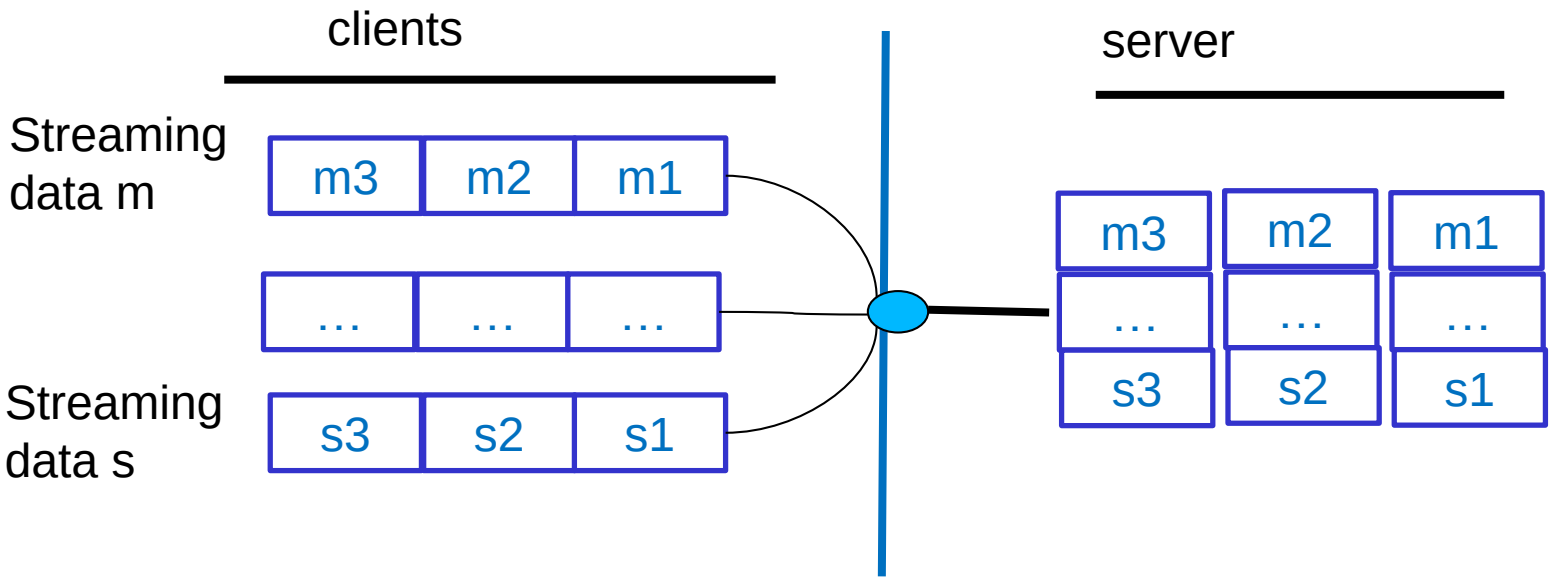
- Data streams can be used for
  - Continuous media (e.g., video)
  - Discrete media (e.g., stock market events/twitter events)

Timing issues



- Timing issues
  - Asynchronous transmission mode
    - no constraints on when the transmission completes
  - Synchronous transmission mode:
    - maximum end-to-end delay defined for each data unit
  - Isochronous transmission
    - maximum and minimum end-to-end delay defined

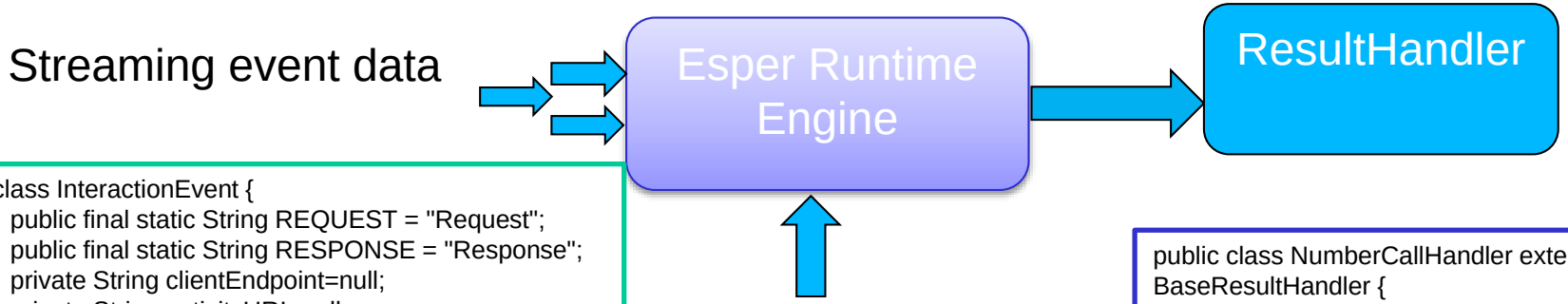
Multiple streams



Complex stream/multiple streams data processing

Example: Complex event processing with Esper

http://esper.codehaus.org/esper.




```
public class InteractionEvent {
        public final static String REQUEST = "Request";
        public final static String RESPONSE = "Response";
        private String clientEndpoint=null;
        private String activityURI=null;

        private String serviceEndpoint=null;

        private String messageCorrelationID=null;

        private String messageType=null;
       ///….
}
```

```
public class NumberCallHandler extends
BaseResultHandler {


   @Override
   public void update(Map[] insertStream,
Map[] removeSteam) {
   ///….
   }
}
```

```
select  clientEndpoint, serviceEndpoint
from InteractionEvent.win:length(100)
where messageType="Request"
```

GROUP COMMUNICATION

Group communication

- Group communication use multicast messages
  - E.g., IP multicast or application-level multicast

Atomic Multicast: Messages are received either by every member or by none of them

Reliable multicast: messages are delivered to all members in the best effort – but not guaranteed.

Atomic Multicast

Q1: Give an example of atomic multicast

Example of implementing multicast using one-to-one communication

Sender's program

```
i:=0;
do  i ≠ n →
   send message to member[i];
   i:= i+1
od
```

Receiver's program

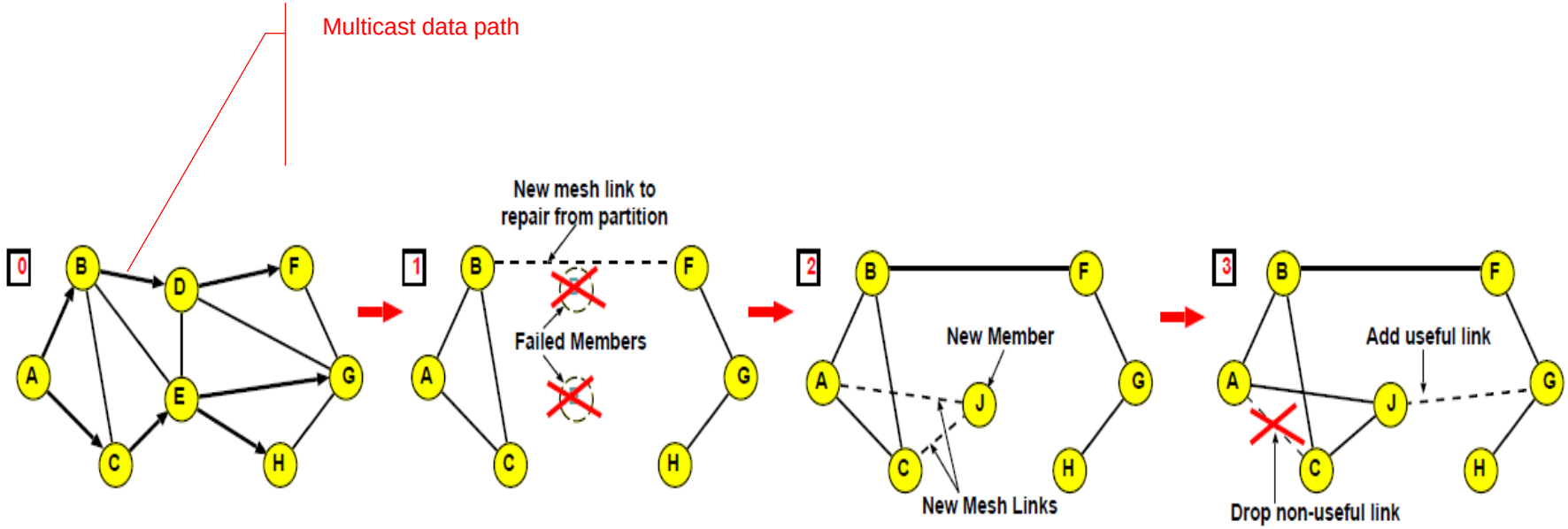
```
if   m is new →
          accept it;
          multicast m;
[] m is duplicate → discard m
fi
```

Source: Sukumar Ghosh, Distributed Systems: An Algorithmic Approach,Chapman and Hall/CRC, 2007

Q2: How to know "m is new"?

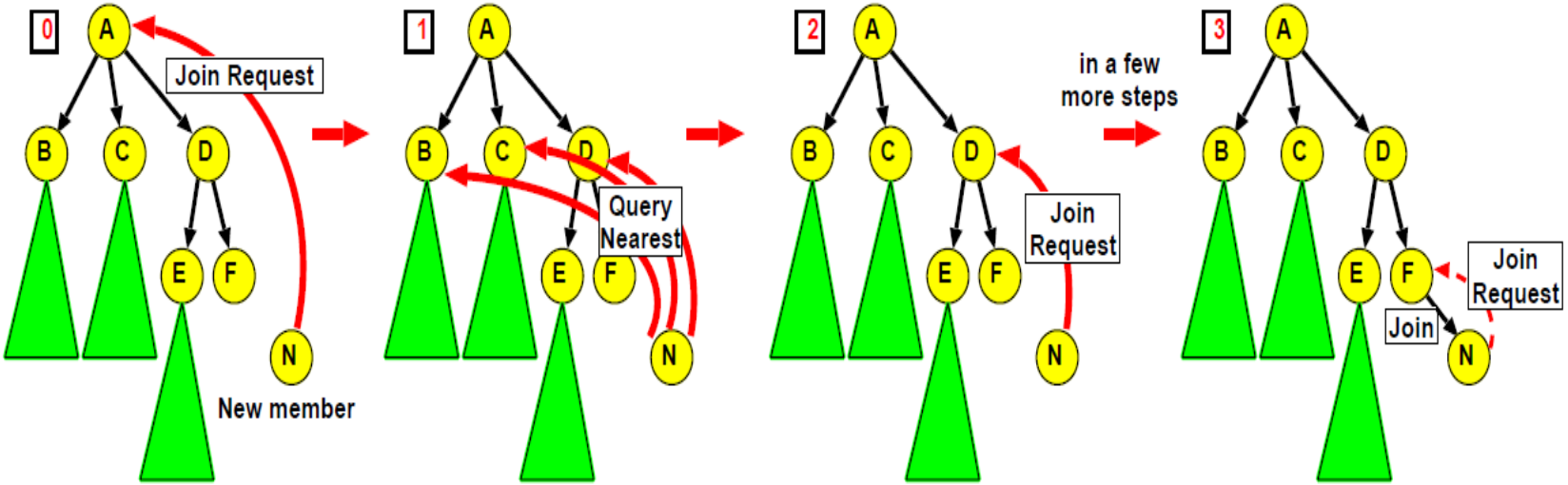
Application-level Multicast Communication (1)

- Application processes are organized into an overlay network, typically in a tree or a mesh



Source: Suman Banerjee , Bobby Bhattacharjee , A Comparative Study of Application Layer Multicast Protocols (2001) , http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.2832

Application-level Multicast Communication (2)



Sources: Suman Banerjee , Bobby Bhattacharjee , A Comparative Study of Application Layer Multicast Protocols (2001) , http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.2832

Gossip-based Data Dissemination (1)

Why gossip?

It can spread messages fast and realiable

Active thread (peer P):

```
(1) selectPeer(&Q);
(2) selectToSend(&bufs);
(3) sendTo(Q, bufs);              ----->
(4)
(5) receiveFrom(Q, &bufr);        <-----
(6) selectToKeep(cache, bufr);
(7) processData(cache);
```

Passive thread (peer Q):

```
(1)
(2)
(3) receiveFromAny(&P, &bufr);
(4) selectToSend(&bufs);
(5) sendTo(P, bufs);
(6) selectToKeep(cache, bufr);
(7) processData(cache)
```

Source: Anne-Marie Kermarrec and Maarten van Steen. 2007. Gossiping in distributed systems. SIGOPS Oper. Syst. Rev. 41, 5 (October 2007), 2-7. DOI=10.1145/1317379.1317381 http://doi.acm.org/10.1145/1317379.1317381

Gossip-based Data Dissemination (2)

- Give a system of N nodes and there is the need to send some data items
- Every node has been updated for data item x
  - Keep x in a buffer whose maximum capability is b
  - Determine a number of times t that the data item x should be forwarded
  - Randomly contact f other nodes (the fant-out) and forward x to these nodes

Different configurations of (b,t,f) create different algorithms

Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, Laurent Massoulieacute;, "Epidemic Information Dissemination in Distributed Systems," Computer, vol. 37, no. 5, pp. 60-67, May 2004, doi:10.1109/MC.2004.1297243

Summary

- Various techniques for programming communication in distributed systems
  - Transport versus application level programming
  - Transient versus persistent
  - Procedure call versus messages
  - Streaming data
  - Multicast and gossip-based data dissemination
- Dont forget to play some simple examples to understand existing concepts

Thanks for your attention

Hong-Linh Truong
Distributed Systems Group
Vienna University of Technology
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong