Communication in Distributed Systems – Programming

What is this lecture about?

- Examine and study main frameworks, libraries and techniques for programming communication in distributed systems
- Understand pros and cons of different techniques for different layers and purposes
- Be able to select the right solutions for the right systems
- Be able to combine different techniques for a complex problem

Learning Materials

- Main reading:
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair„Distributed Systems – Concepts and Design“, 5nd Edition
    - Chapters 4,5,6 and 9
  - Sukumar Ghosh, “Distributed Systems: AnAlgorithmic 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 Invocation
- Web Services
- Streaming data programming
- Group communication
- Gossip-based Data Dissemination
- Summary

MESSAGE-ORIENTED TRANSIENT COMMUNICATION

Message-oriented Transient Communication at Transport Layer

- How does an application use 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.
- Designed for low-level system, high-performance, resource-constrained communication

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 the result back to the client

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

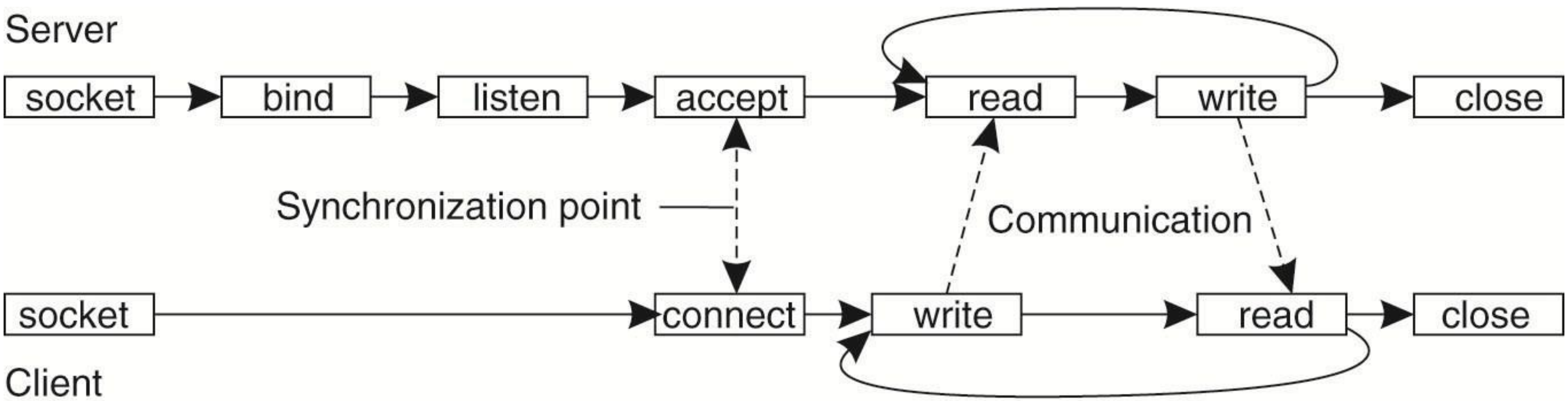
Socket Primitives

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 interaction

Connection-oriented communication interaction

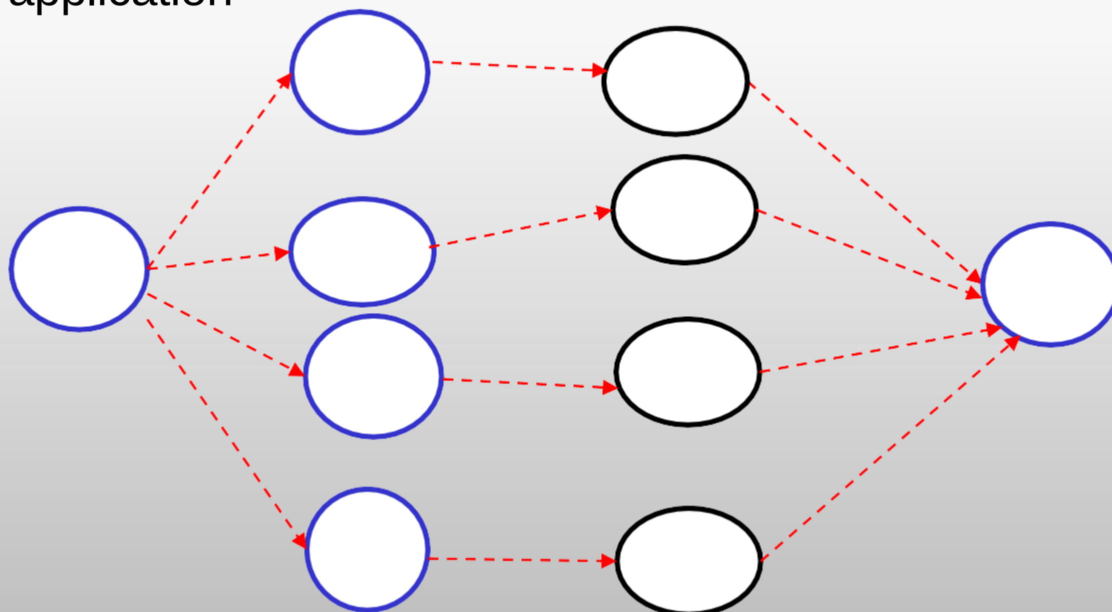


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007,Prentice-Hall

Q: How can a multi-threaded server be implemented?

Message-oriented Transient Communication at the Application level

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



Why are transport level socket programming primitives 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();
```

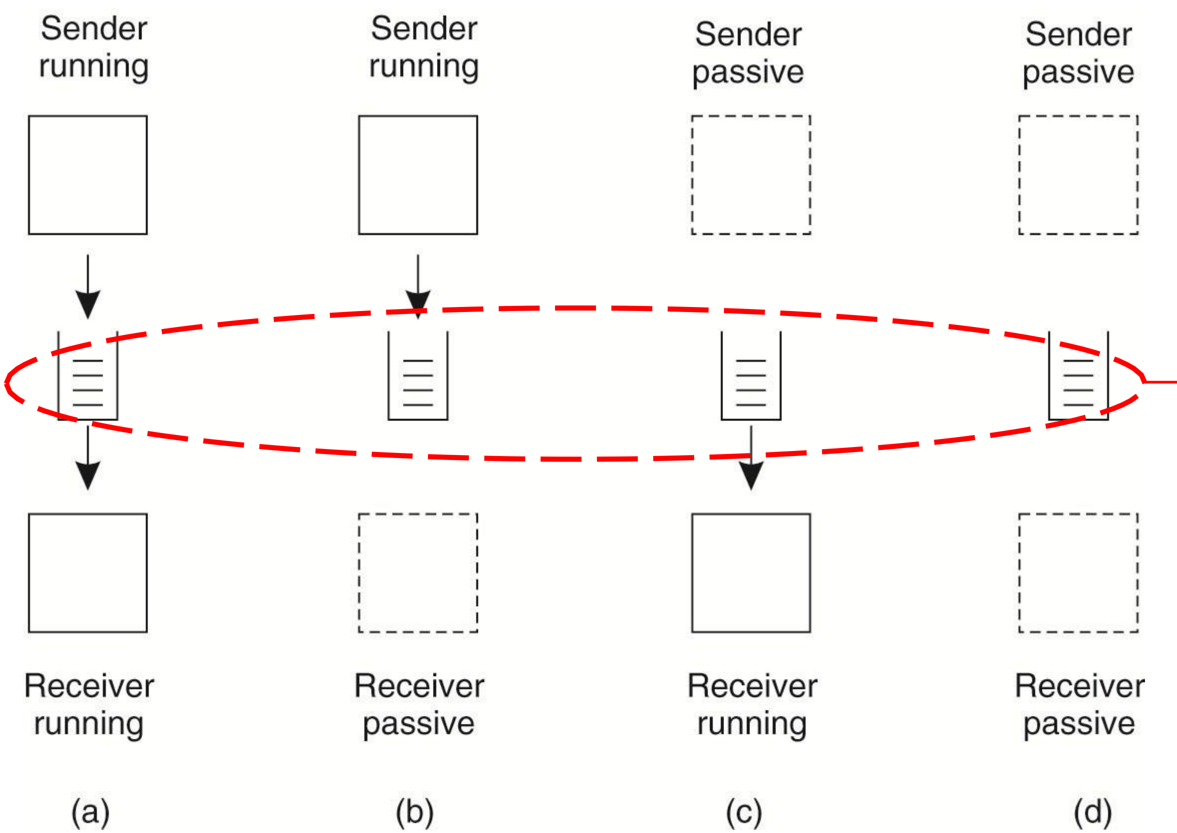
MESSAGE-ORIENTED PERSISTENT COMMUNICATION

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
  - Different communication patterns
- Several Implementations

Message-oriented Persistent Communication – Queuing Model

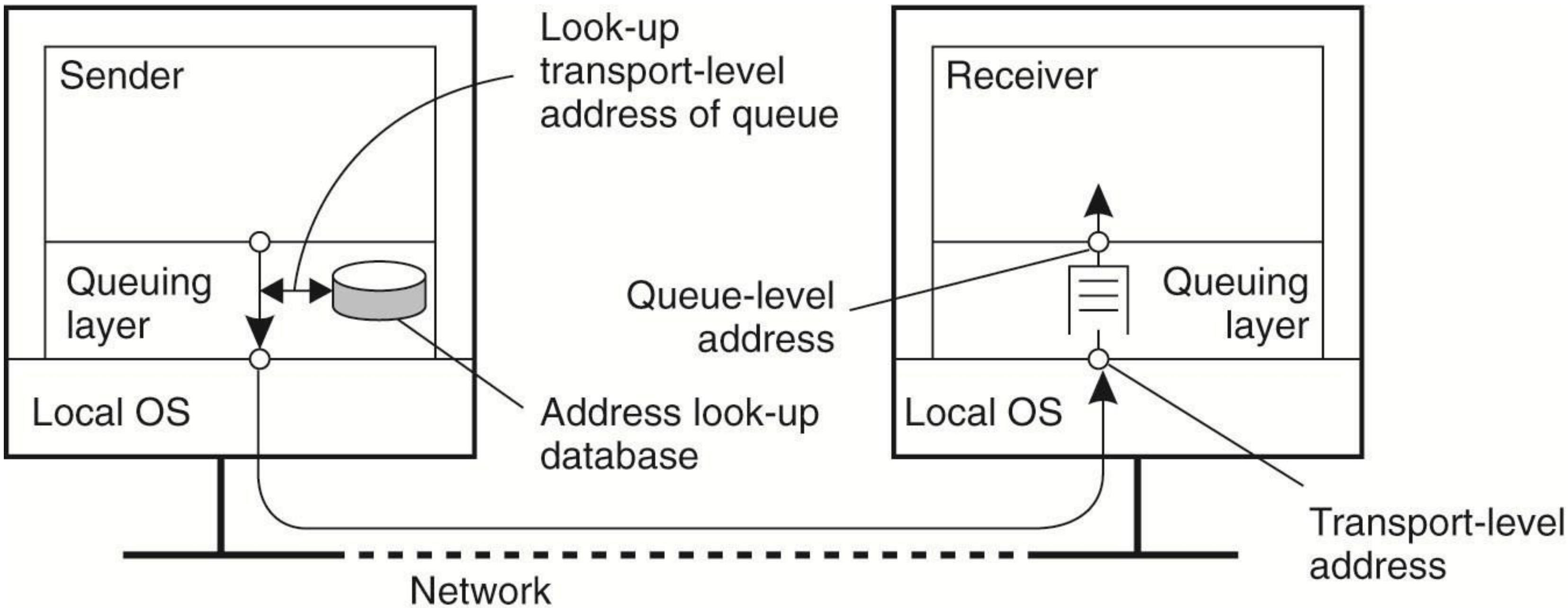
Communication models with time (un)coupling



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

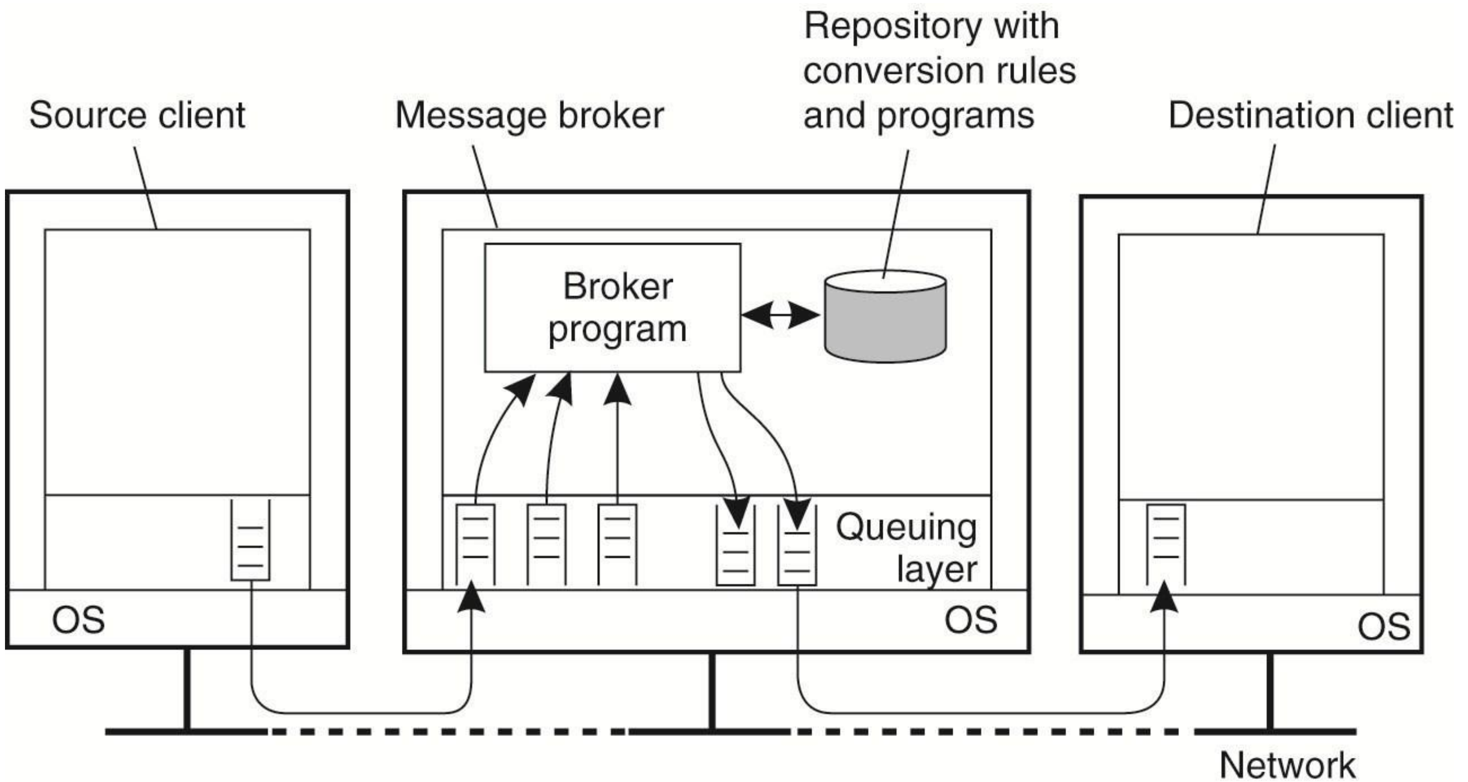


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Practical work: JMS - http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html

Message Brokers

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



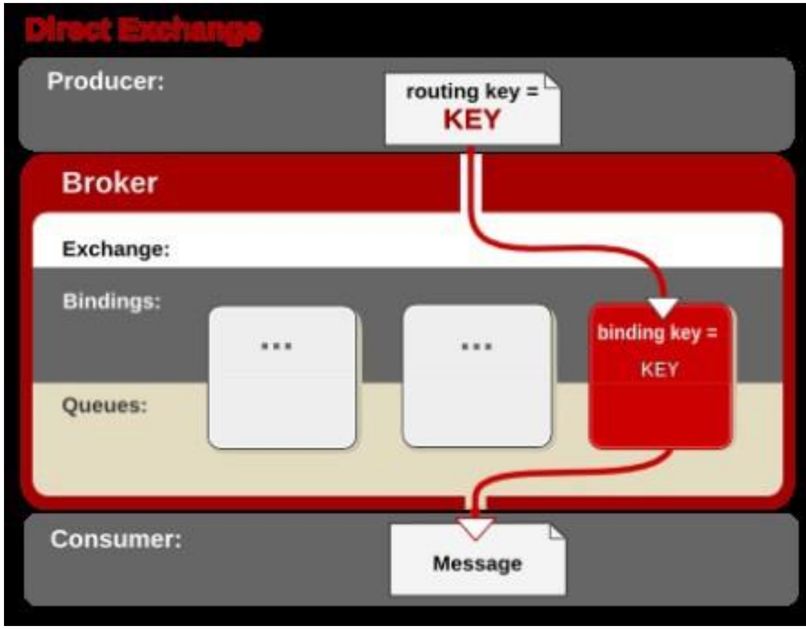
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles

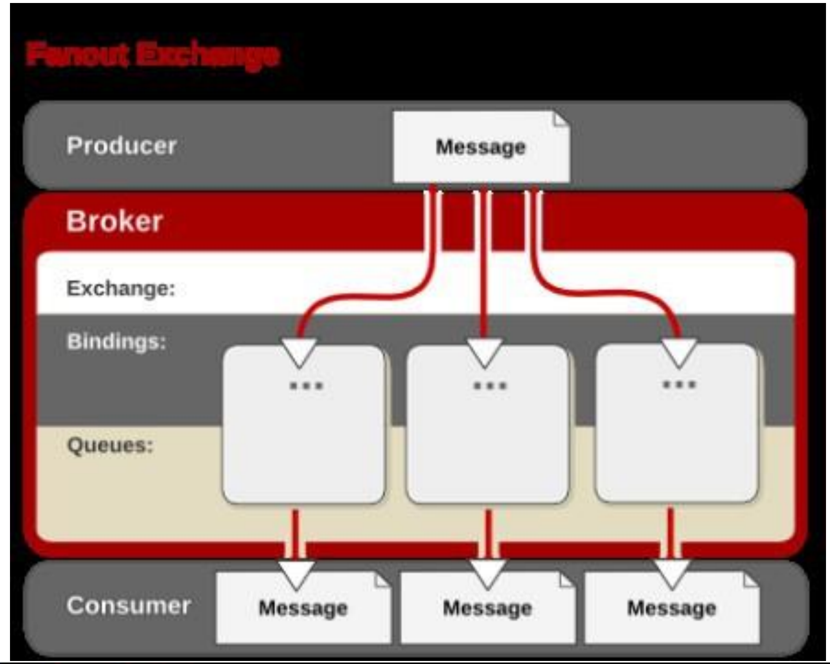
Example – Advanced Message Queuing Protocol (AMQP)

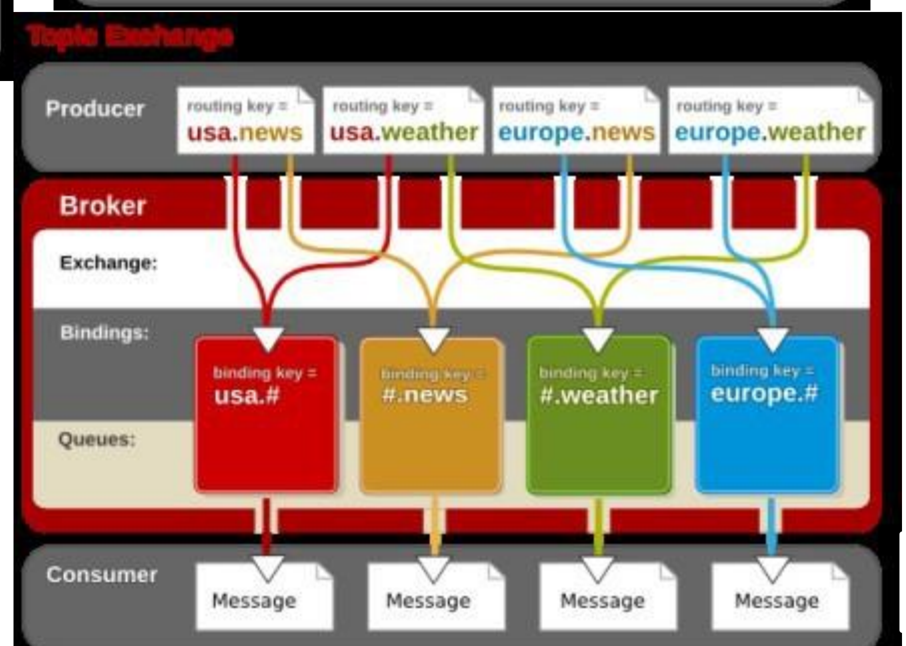
- http://www.amqp.org

Apache Qpid™

Content-Based Message Routing: AMQP







Note: defined in AMQP 0-10
But not in AMQP 1.0

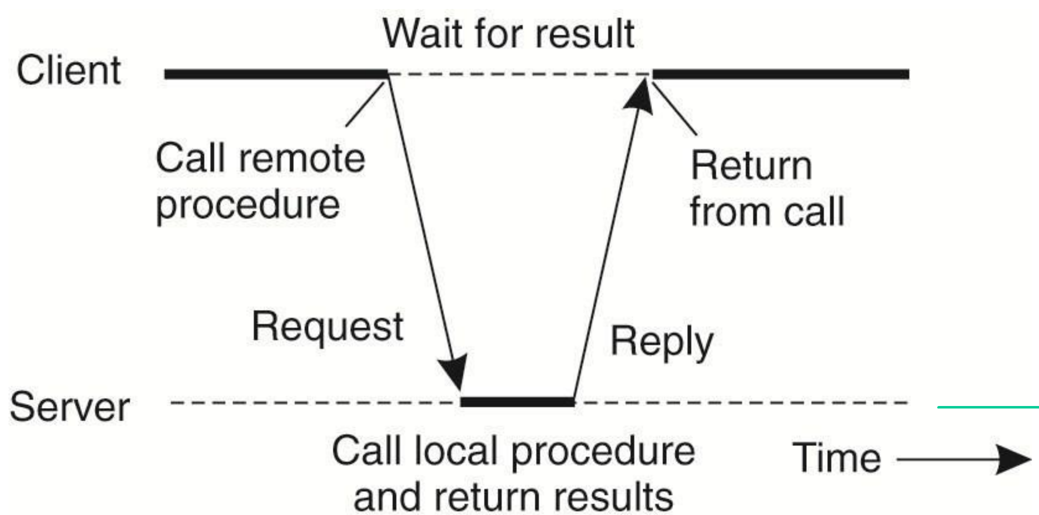
Figs source: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/1.1/html/Messaging_User_Guide/chap-Messaging_User_Guide-Exchanges.html

REMOTE INVOCATION

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): hides 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

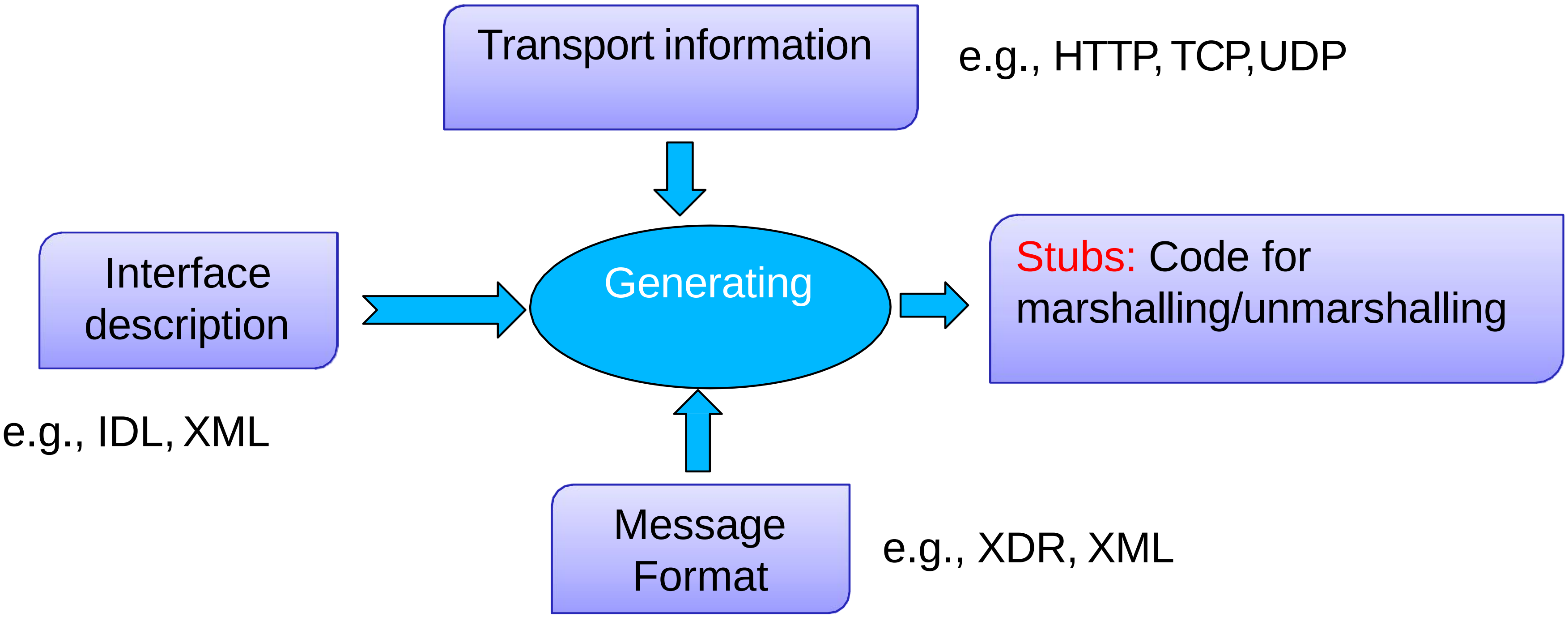
- Passing parameters and results needs agreed message format between a client and a server

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

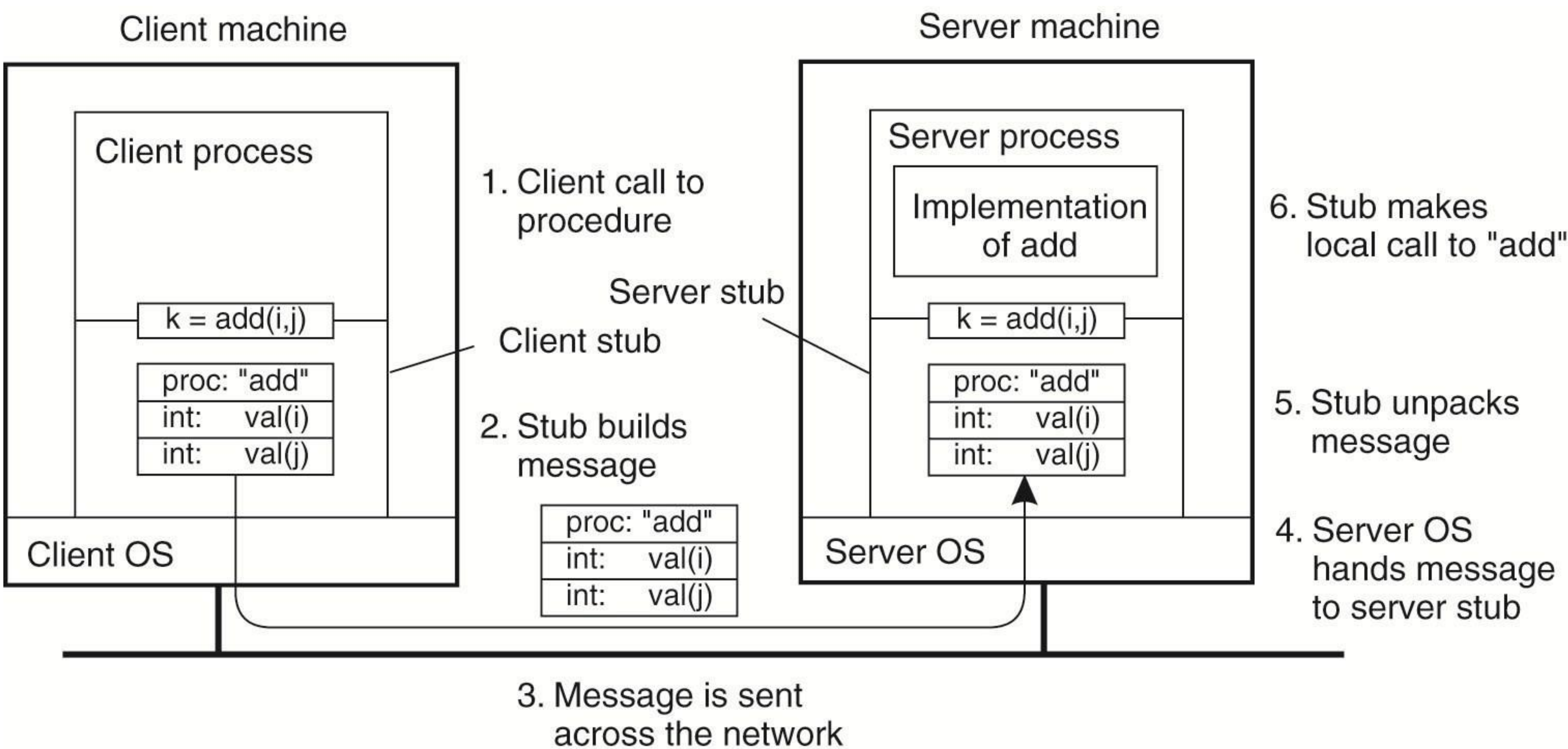
- Data types may have different representations due to different machine types (e.., SPARC versus Intel x86)

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

Generating stubs

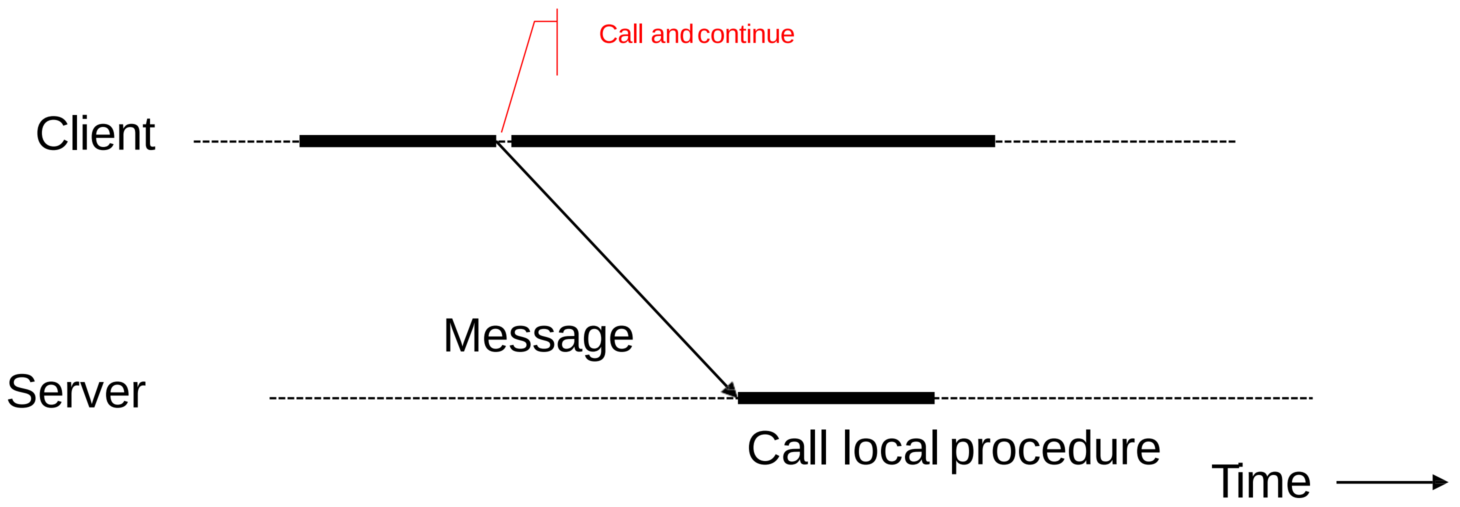


Detailed Interactions



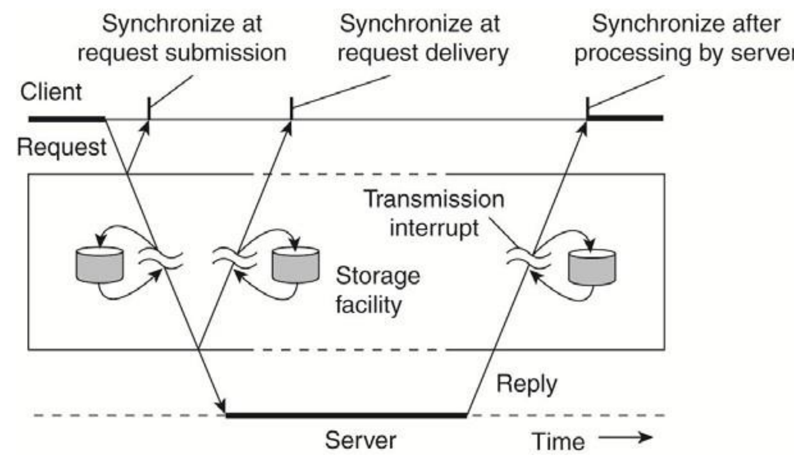
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

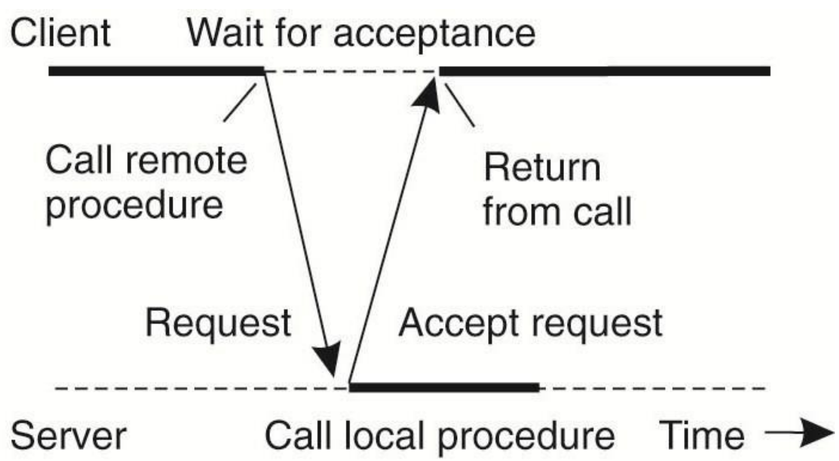
One-way RPC



Call and continue
Client
Message
Server
Call local procedure
Time

Asynchronous RPC



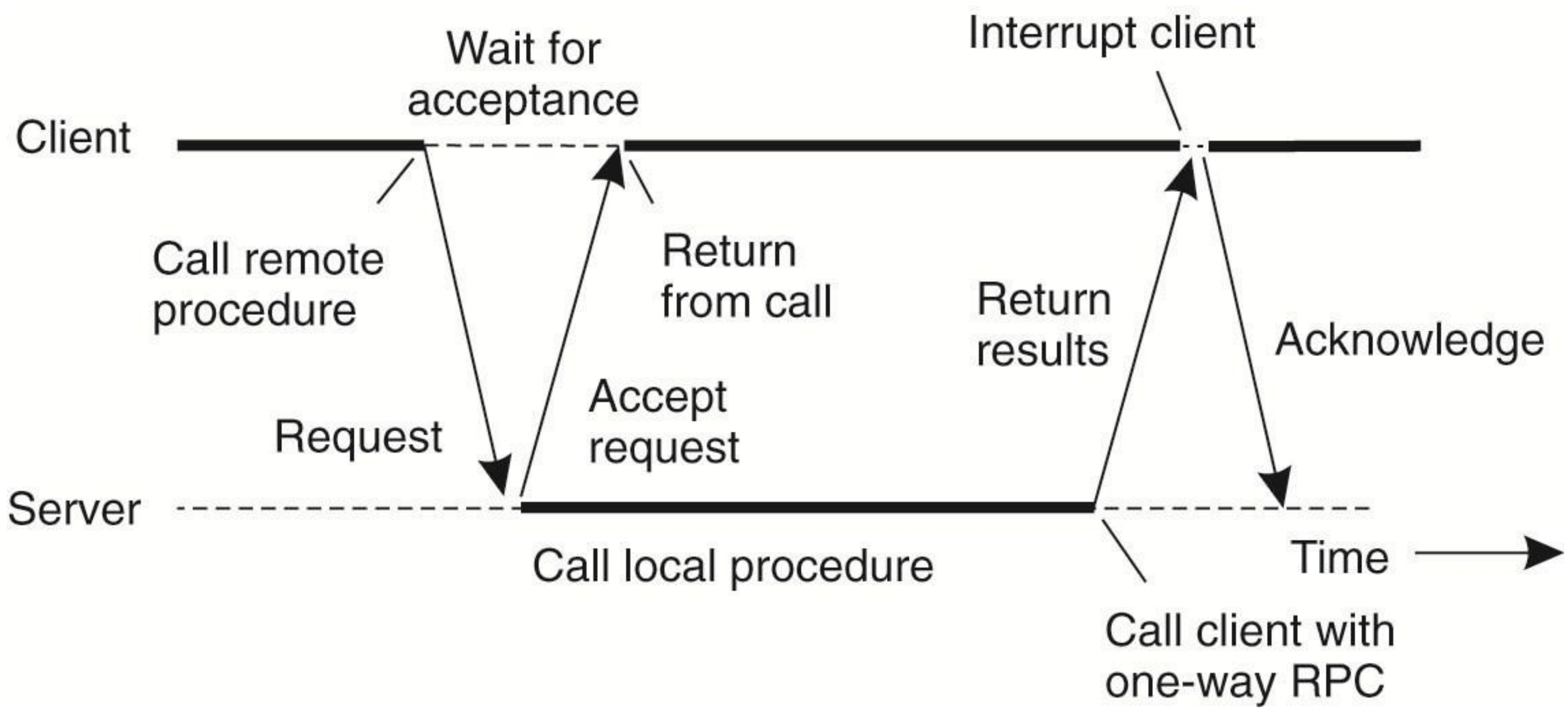


(b)

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

Asynchronous RPC

Two asynchronous RPCs/ Deferred synchronous RPC



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

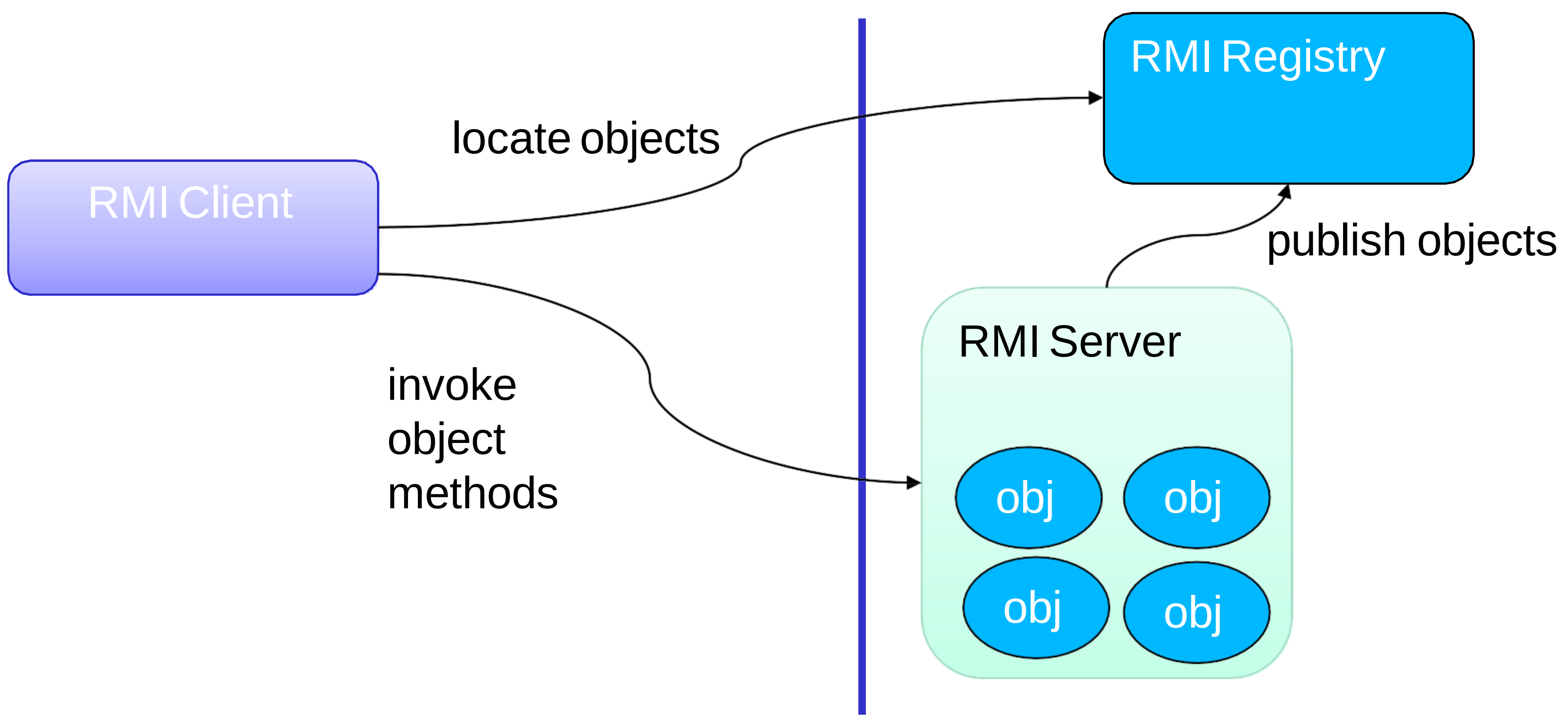
Q: List some possible failures in RPC interactions.

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
- Tools: Apache Thrift - http://thrift.apache.org/

Remote Meothd Invocation/Remote Object Call

- Remote object method invocation/call
  - RPC style in object-oriented programming



WEB SERVICES

Web services (1)

- Service: common software functionalities/capabilities offered through well-defined interfaces and consistent usage policies
- Socket APIs, RPC, or RMI can be used to implement "services", but
  - Do not work very well in the Web/Internet environment
  - Do not support well the integration of different software systems

Web Services: **"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."** -- http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis

Web services (2)

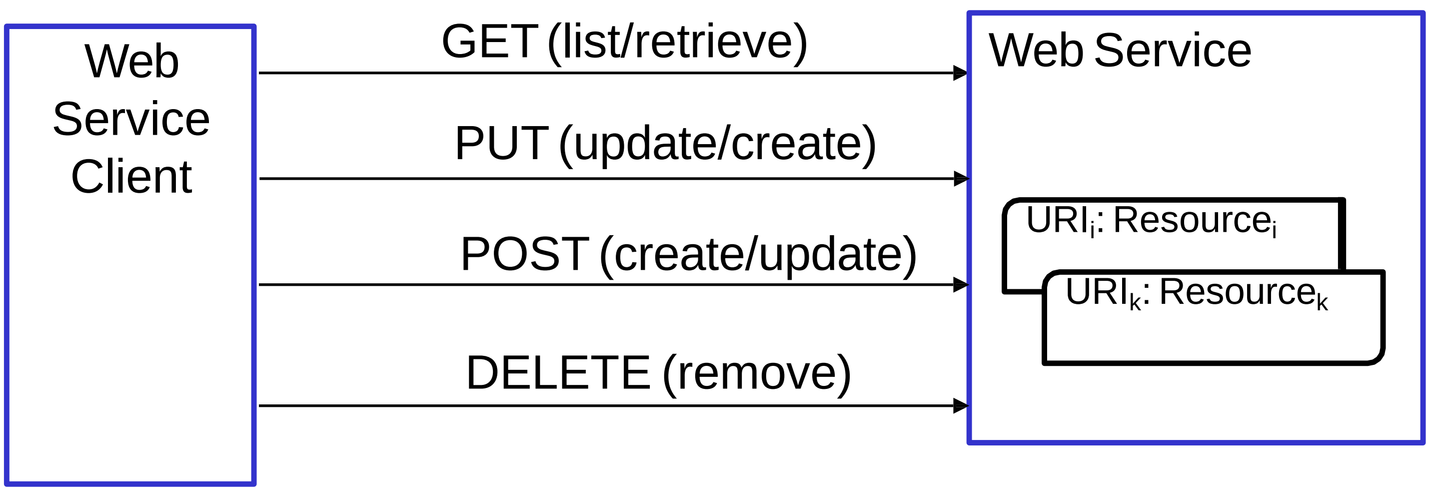
Applications			
Web Services			Services and descriptions
Web API/REST			Protocols/interfaces
URIs	**XML, JSON, etc.**	**HTTP, SMTP, RMI, …**	Identifiers, data format, transportation

- Why Web services are important in distributed systems?
  - Support interoperability
  - Hide system complexity and implementation detail
  - Enable easy integration of diverse and distributed software components

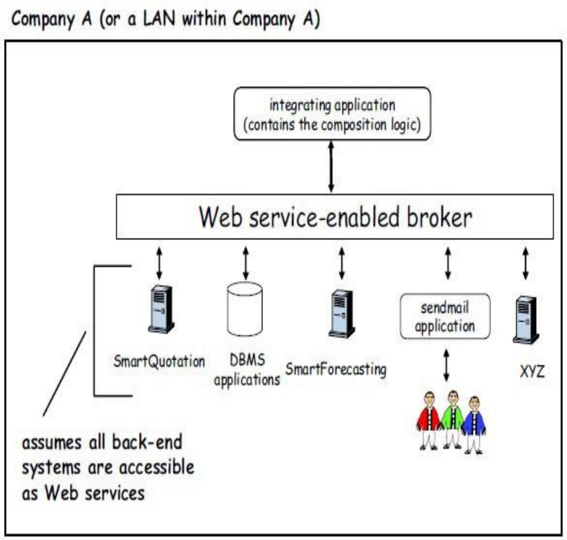
SOAP versus REST: http://wwwconference.org/www2008/papers/pdf/p805-pautassoA.pdf

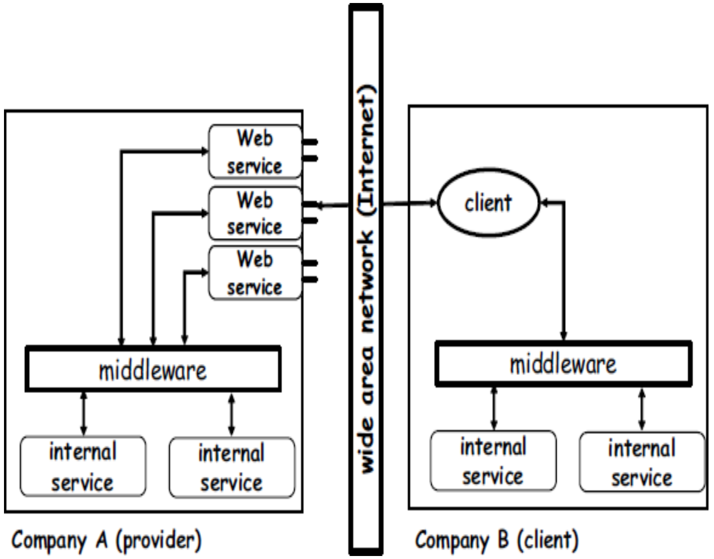
Architectural Design - REST

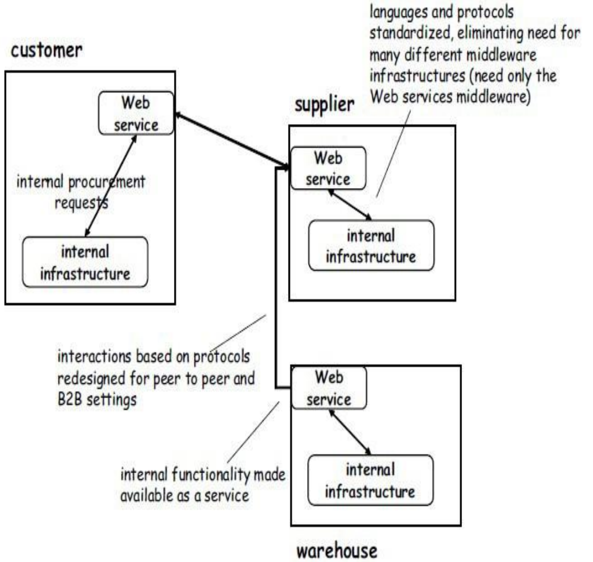
- Resources are identified and accessed through URIs
- Resources are data and functionality
- A Web service manages a set of resources
- A client and a service exchange representations of resources via standardized interface and protocols
  - Assume one-to-one communication/client-server model



Applications: Service-oriented Architecture/Computing







Source: Web Services: Concepts, Architecture and Applications,
G. Alonso, F. Casati, H. Kuno, V. Machiraju
Springer Verlag 2004
Chapter 5,

Applications – Cloud Computing

- Cloud resources
  - Files, storage, compute machines, middleware, etc.
  - Resources offered via RESTful models
- Many cloud services support REST APIs
- Examples

Compute & Networking

Amazon EC2
Auto Scaling
Elastic Load Balancing
Amazon VPC
Amazon Route 53
AWS Direct Connect

Storage & Content Delivery

Amazon S3
Amazon Glacier
Amazon EBS
AWS Import/Export
AWS Storage Gateway
Amazon CloudFront

Database

Amazon RDS
Amazon DynamoDB
Amazon ElastiCache
Amazon Redshift
Amazon SimpleDB

Analytics

Amazon EMR
Amazon Kinesis
AWS Data Pipeline

Deployment & Management

AWS Identity & Access Management
AWS CloudTrail
Amazon CloudWatch
AWS Elastic Beanstalk
AWS CloudFormation
AWS OpsWorks
AWS CloudHSM

App Services

Amazon AppStream
Amazon CloudSearch
Amazon Elastic Transcoder
Amazon SES
Amazon SQS
Amazon SWF

Mobile Services

Amazon Cognito
Amazon Mobile Analytics
Amazon SNS
AWS Mobile SDK for Android
AWS Mobile SDK for iOS

Resources

AWS Billing and Cost Management
AWS Marketplace

Getting Started with AWS

Getting Started with AWS
Computing Basics (Linux)
Computing Basics (Windows)
Web App Hosting (Linux)
Web App Hosting (Windows)
Deploying a Web Application
Analyzing Big Data with AWS
Static Website Hosting

Tools for Amazon Web Services

AWS Management Console
AWS SDK for Java
AWS SDK for JavaScript
AWS SDK for .NET
AWS SDK for PHP
AWS SDK for Python (boto)
AWS SDK for Ruby
AWS Toolkit for Eclipse
AWS Toolkit for Visual Studio
AWS Command Line Interface
AWS Tools for Windows PowerShell

Additional Software & Services

Alexa Top Sites
Alexa Web Information Service
Amazon Mechanical Turk
Amazon Silk

STREAMING DATA PROGRAMMING

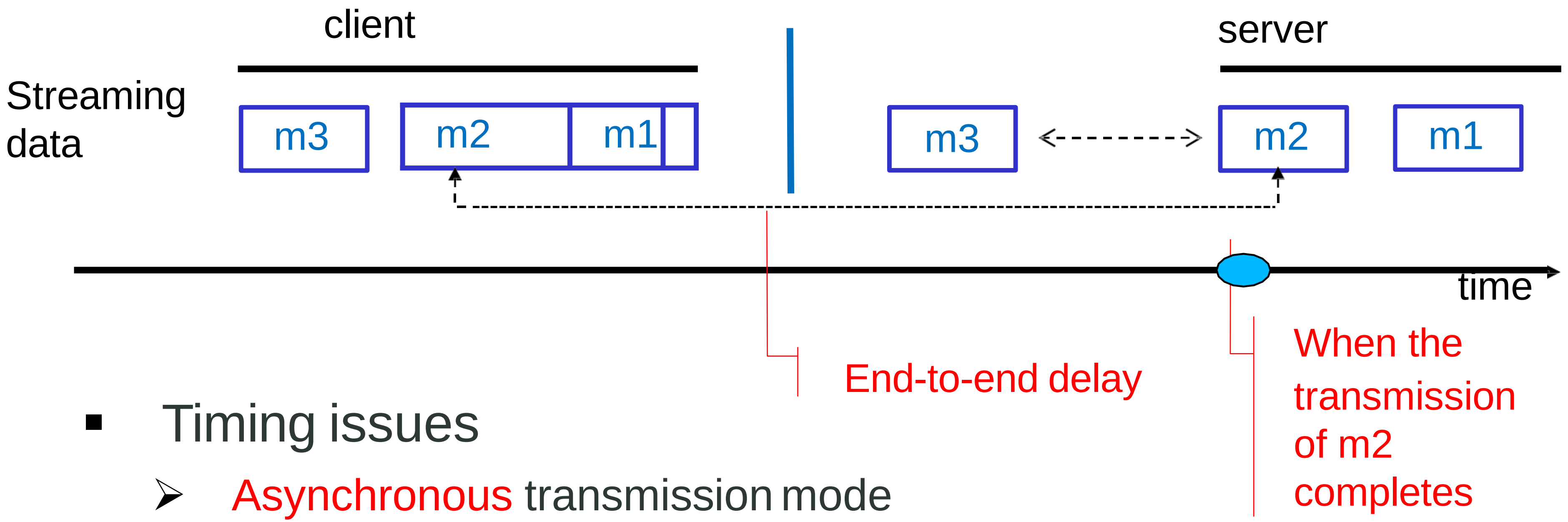
Data stream programming

Data stream: a sequence of data units

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

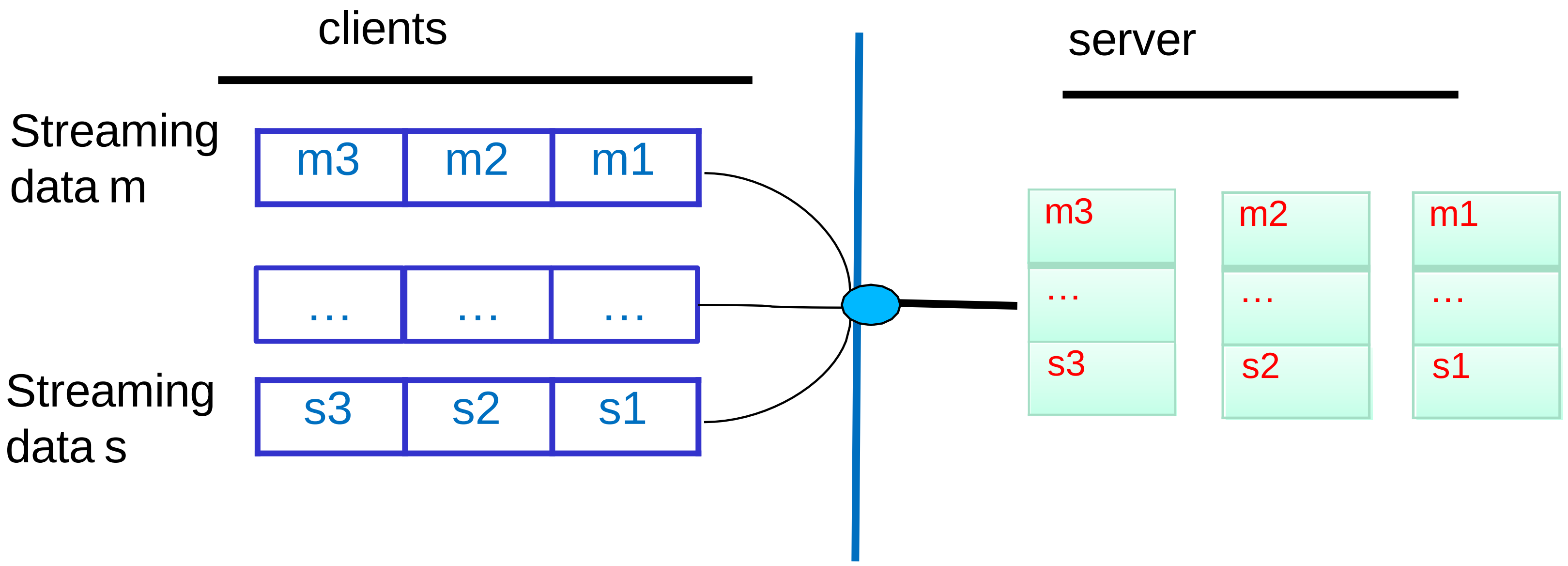
- Data streams can be used for
  - Continuous media (e.g., video)
  - Discrete media (e.g., stock market events/twitter events)

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

Tools: Esper | Storm | S4 | Gigaspaces XAP | Streambase

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

<u>Sender's program</u>

i:=0;

do i $\neq$ n $\rightarrow$

send message to member[i];

i:= i+1

od

<u>Receiver's program</u>

if **m** is new $\rightarrow$

accept it;

multicast **m**;

[] **m** is duplicate $\rightarrow$ discard m

fi

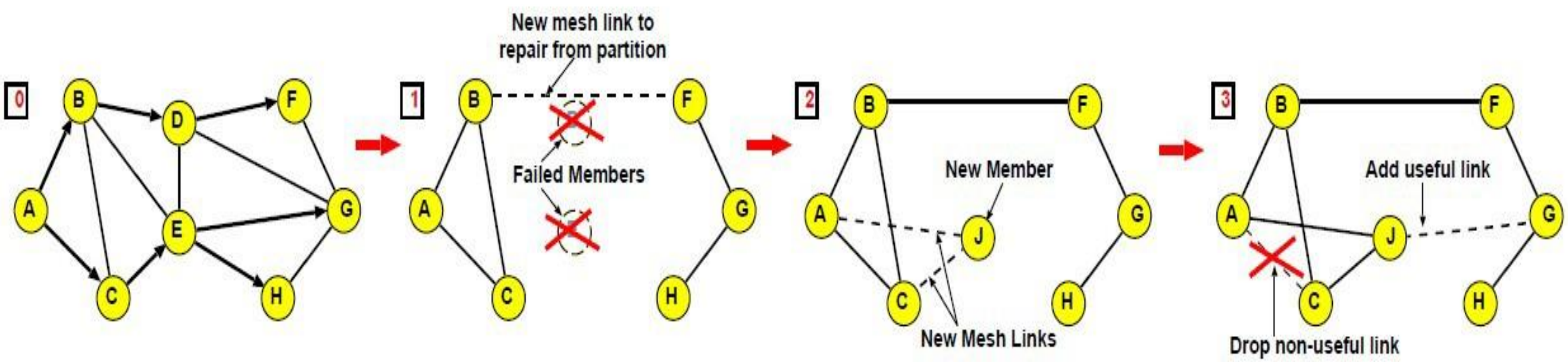
Source: Sukumar Ghosh, Distributed Systems: An Algorithmic Approach,Chapman and Hall/CRC, 2007

Q2: How do we know "m is new"?

Application-level Multicast Communication (1)

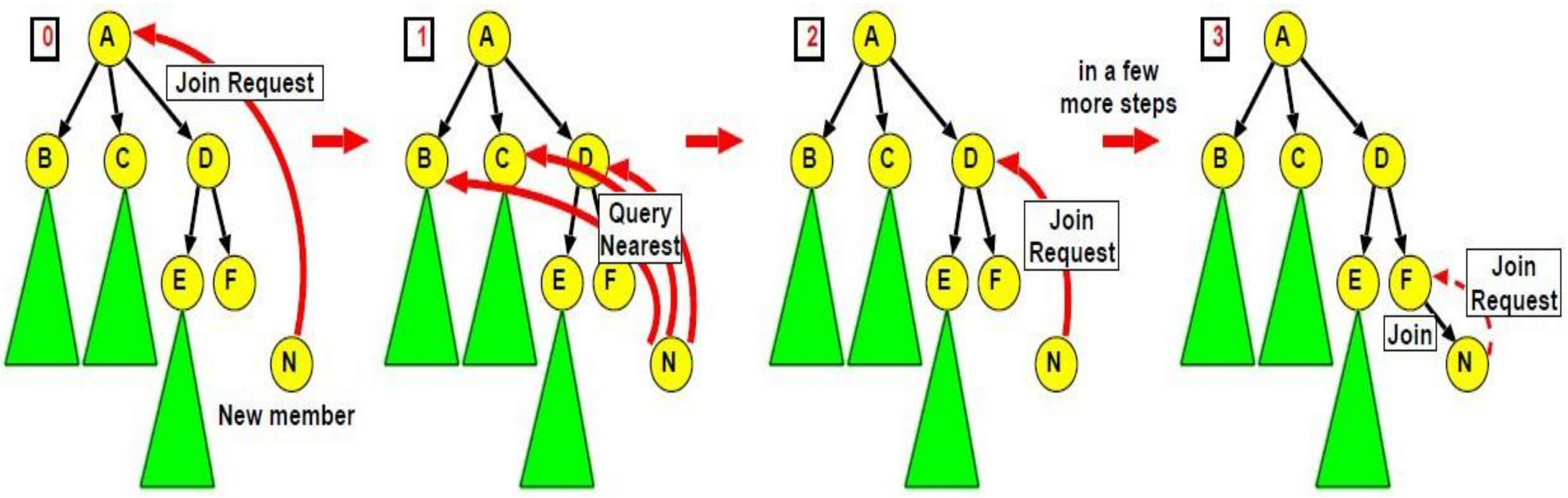
- Application processes are organized into an overlay network, typically in a mesh or a tree

Multicast data path



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? E.g., https://www.youtube.com/watch?v=OPYhk_NbEtA#t=22

It can spread messages fast and reliably

Active thread (peer P):		Passive thread (peer Q):
(1) selectPeer(&Q);		(1)
(2) selectToSend(&bufs);		(2)
(3) sendTo(Q, bufs);	----->	(3) receiveFromAny(&P, &bufr);
(4)		(4) selectToSend(&bufs);
(5) receiveFrom(Q, &bufr);	<-----	(5) sendTo(P, bufs);
(6) selectToKeep(cache, bufr);		(6) selectToKeep(cache, bufr);
(7) processData(cache);		(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 communication
  - Procedure call versus messages
  - Web Services
  - Streaming data
  - Multicast and gossip-based data dissemination
- Dont forget to play with some simple examples to understand existing concepts