Start Recording

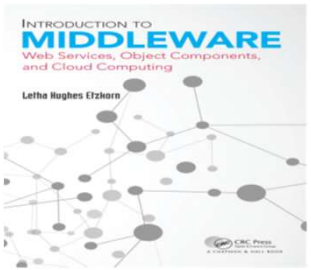
IMP Note to Students

- It is important to know that just login to the session does not guarantee the attendance.
- Once you join the session, continue till the end to consider you as present in the class.
- IMPORTANTLY, you need to make the class more interactive by responding to Professors queries in the session.
- **Whenever Professor calls your number / name ,you need to respond, otherwise it will be considered as ABSENT**

Course Name :
Middleware Technologies
CSIW ZG524

BITS Pilani
Pilani Campus

Textbooks



T1: Letha Hughes Etzkorn - Introduction to middleware _ web services, object components, and cloud computing-Chapman and Hall_CRC (2017).

T2: William Grosso - Java RMI (Designing & Building Distributed Applications)

R1: Gregor Hohpe, Bobby Woolf - Enterprise Integration Patterns_ Designing, Building, and Deploying Messaging Solutions -Addison-Wesley Professional (2003)

R2: MongoDB in Action

Note: In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

Evaluation Components

Evaluation Component	Name	Type	Weight	Duration	Schedule
EC – 1	Quiz I, II & III	Individual / Take-home	15%		Pre/Post Mid-Sem
EC – 2	Assignment/ Laboratory Exercises	Practical	10%		TBA
EC – 3	Mid-Semester Examination	Closed Book	30%	2 Hrs.	TBA
EC – 4	End-Semester Examination	Open Book	45%	3 Hrs.	TBA

Modular Structure

No	Title of the Module
M1	Introduction and Evolution
M2	Enterprise Middleware
M3	Middleware Design and Patterns
M4	Middleware for Web-based Application and Cloud-based Applications
M5	Specialized Middleware

innovate achieve lead

BITS Pilani

Pilani Campus

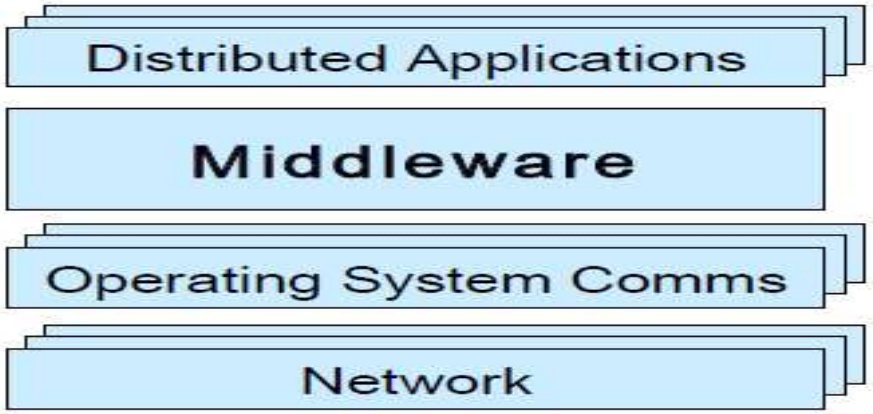
CS1: Introduction and Evolution

Agenda

- View of Middleware
- Forms of Middleware

What is Middleware?

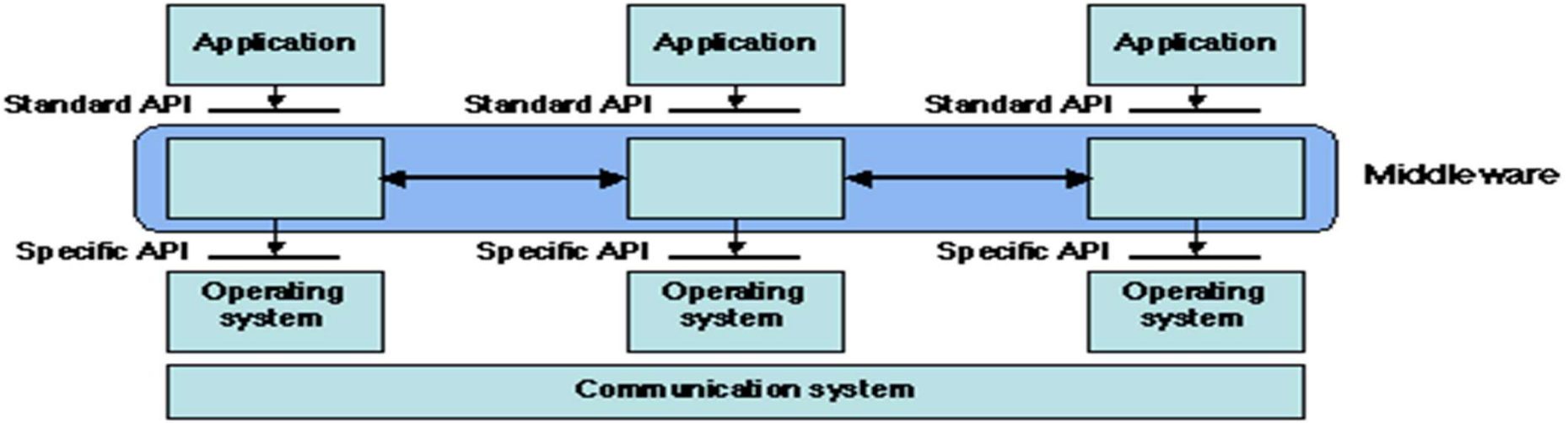
- Layer between OS and distributed applications
- Hides complexity and heterogeneity of distributed system
- Bridges gap between low-level OS communications and programming language abstractions
- Provides common programming abstraction and infrastructure for distributed applications
- Overview at: http://www.middleware.org



What is Middleware?

- "Middleware is the software that connects software components or enterprise applications in a distributed system".

- Examples: Enterprise Application Integration software, telecommunications software, transaction monitors, and messaging-and-queueing software.



Cont.

- Middleware provides support for (some of):
  - Naming, Location, Service discovery, Replication
  - Protocol handling, Communication faults, QoS
  - Synchronization, Concurrency, Transactions, Storage
  - Access control, Authentication

- Middleware dimensions:
  - Request/Reply vs. Asynchronous Messaging
  - Language-specific vs. Language-independent
  - Proprietary vs. Standards-based
  - Small-scale vs. Large-scale
  - Tightly-coupled vs. Loosely-coupled components

Common Forms of MW

- Sockets
- Remote Procedure Calls
- Distributed Object-Oriented Components (Ex: ORB)
- Message Oriented Middleware (Message Queues/Enterprise Message Bus etc.)
- Service Oriented Architectures
- Web services (Arbitrary / RESTful)
- SQL-oriented data access
- Embedded middleware
- Cloud Computing

Sockets

- Socket is an internal endpoint for sending/receiving data within a node on network
- Berkeley/POSIX sockets defined API for Inter Process Communication (IPC) within same host (BSD 4.2 – circa 1983)
- Early form of Middleware (limited to same host systems)
- Windows variant (WinSock) based on BSD Sockets.
- Treated similar to files in BSD/POSIX
- Maintained in File Descriptor table
- Supported protocols
  - TCP/IP – IPv4, IPv6
  - UDP

Sockets API

- **socket** —creates a descriptor for use in network communications
- **connect** —connect to a remote peer (client)
- **write** —send outgoing data across a connection
- **read** —acquire incoming data from a connection
- **close** —terminate communication and deallocate a descriptor
- **bind** —bind a local IP address and protocol port to a socket
- **listen** —set the socket listening on the given address and port for connections from the client and set the number of incoming connections from a client (backlog) that will be allowed in the listen queue at any one time
- **accept** —accept the next incoming connection (server)
- **recv** —receive the next incoming datagram
- **recvmsg** —receive the next incoming datagram (variation of recv)
- **recvfrom** —receive the next incoming datagram and record its source endpoint address
- **send** —send an outgoing datagram
- **sendmsg** —send an outgoing datagram (variation of send)
- **sendto** —send an outgoing datagram, usually to a prerecorded endpoint address
- **shutdown** —terminate a TCP connection in one or both directions
- **getpeername** —after a connection arrives, obtain the remote machine's endpoint address from a socket
- **getsockopt** —obtain the current options for a socket
- **setsockopt** —change the options for a socket

Sockets - Life Cycle

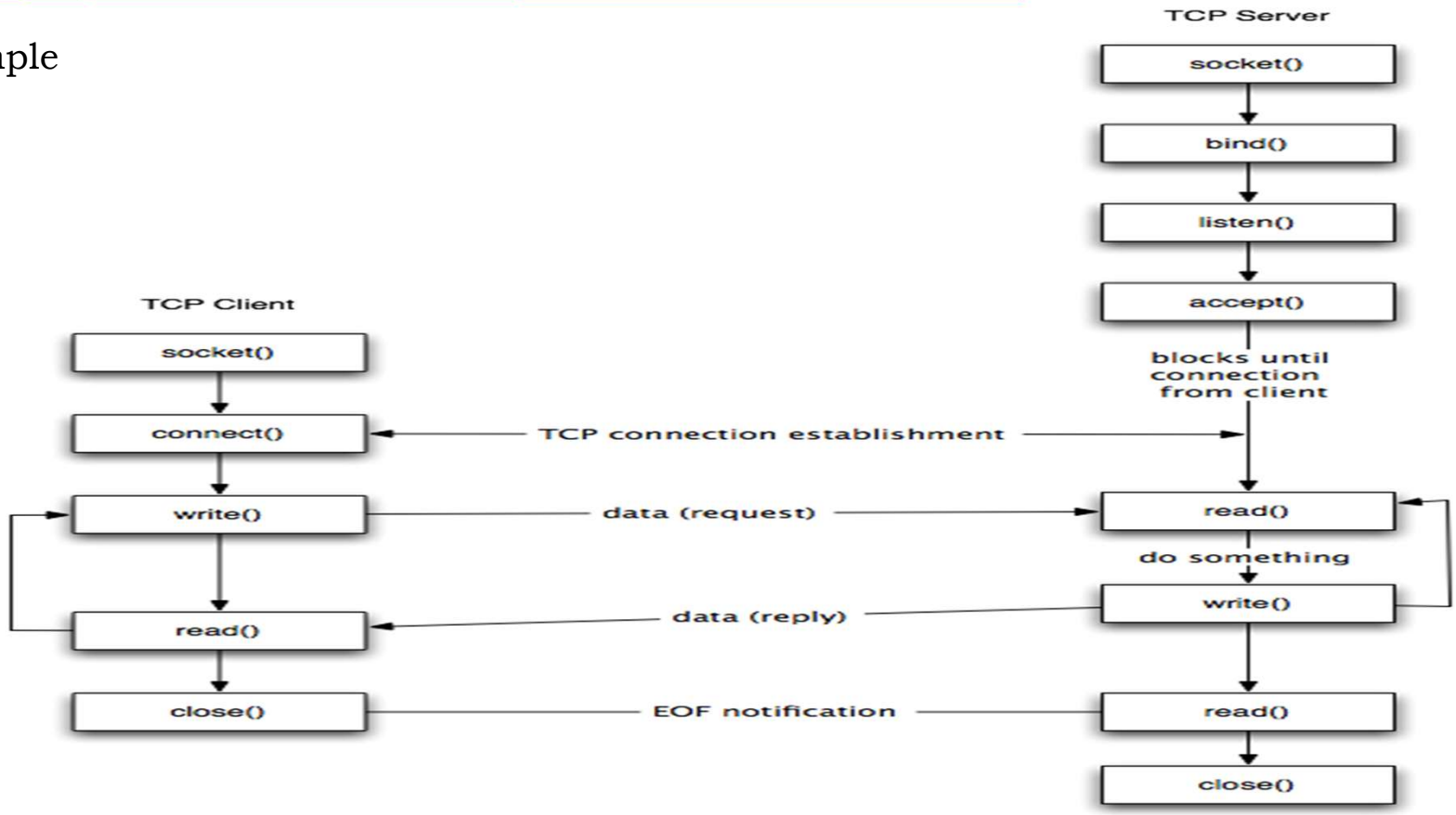
TCP Client – Server example

#Server

server.c

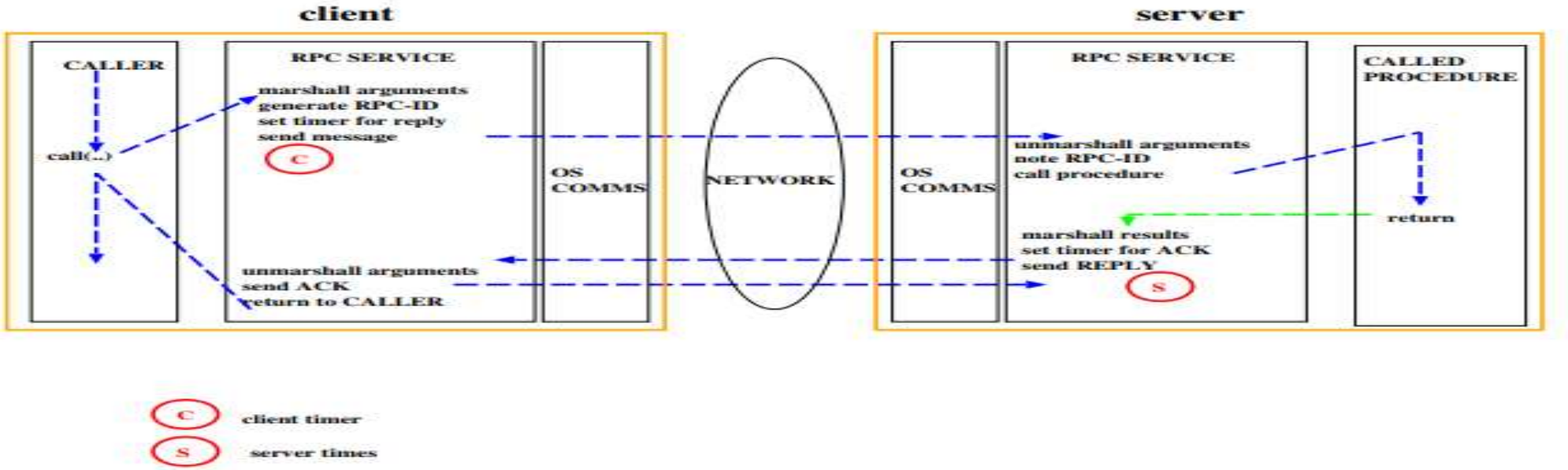
#Client

client.c



Remote Procedure Call (RPC)

- Mask's remote function calls as being local Client/server model
- Request/reply paradigm usually implemented with message passing in RPC service
- Marshalling of function parameters and return value



Remote Procedure Call (RPC)

Properties of RPC

- Language-level pattern of function call
  - easy to understand for programmer
- Synchronous request/reply interaction
  - natural from a programming language point-of-view matches replies to requests
  - built in synchronization of requests and replies
- Distribution transparency (in the no-failure case)
  - hides the complexity of a distributed system
- Various reliability guarantees
  - deals with some distributed systems aspects of failure

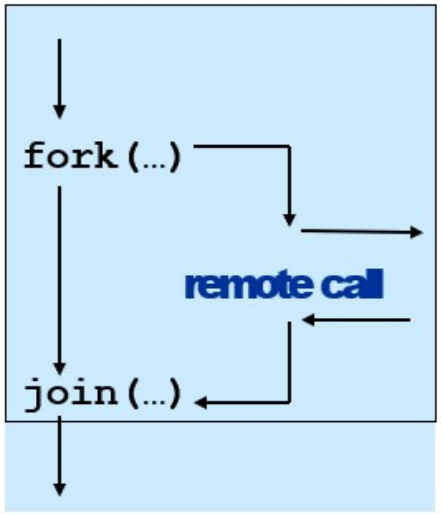
Failure Modes of RPC

- Invocation semantics supported by RPC in the light of : network and/or server congestion, client, network and/or server failure. **Note** DS independent failure modes
- RPC systems differ, many examples, local was Mayflower
- May be or at most once(RPC system tries once) Error return – programmer may retry
- Exactly once (RPC system retries a few times)
  - Hard error return – some failure most likely note that “exactly once” cannot be guaranteed

Remote Procedure Call (RPC)

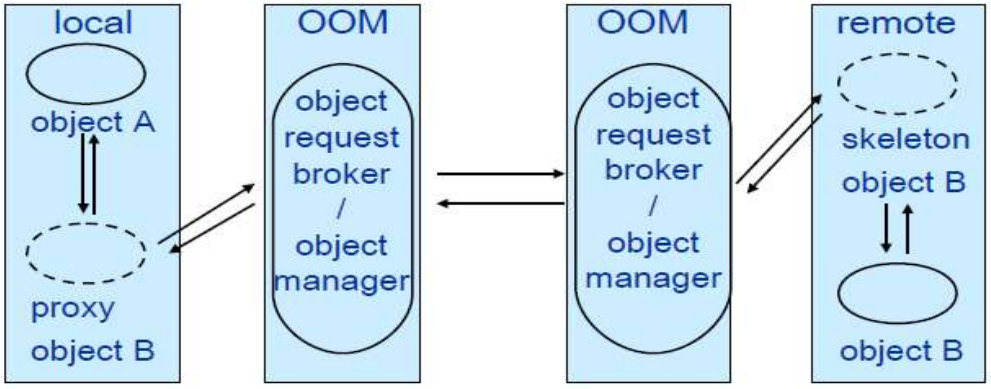
Disadvantages of RPC

- Synchronous request/reply interaction
  - tight coupling between client and server
  - client may block for a long time if server loaded leads to multi-threaded programming at client
  - slow/failed clients may delay servers when replying multi-threading essential at servers
- Distribution Transparency
  - Not possible to mask all problems
- RPC paradigm is not object-oriented
  - invoke functions on servers as opposed to methods on objects



Object-Oriented Middleware (OOM)

- **Objects** can be *local* or *remote*
- **Object references** can be *local* or *remote*
- Remote objects have visible **remote interfaces**
- Masks remote objects as being local using proxy objects Remote method invocation



Properties of OOM

- Support for object-oriented programming model
  - objects, methods, interfaces, encapsulation…
  - exceptions (were also in some RPC systems e.g. Mayflower)
- Synchronous request/reply interaction
  - same as RPC
- Location Transparency
  - system (ORB) maps object references to locations
- Services comprising multiple servers are easier to build with OOM
  - RPC programming is in terms of server-interface (operation)
  - RPC system looks up server address in a location service

Java Remote Method Invocation (RMI)

```
public interface PrintService extends Remote
        {
        int print(Vector printJob) throws RemoteException;
        }
```

- Distributed objects in Java
- RMI compiler creates proxies and skeletons
- RMI registry used for interface lookup
- Entire system written in Java (single-language system)

CORBA

- Common Object Request Broker Architecture
  - Open standard by the OMG (Version 3.0)
  - Language- and platform independent
- Object Request Broker (ORB)
  - General Inter-ORB Protocol (GIOP) for communication
  - Interoperable Object References (IOR) contain object location
  - CORBA Interface Definition Language (IDL)
    - Stubs (proxies) and skeletons created by IDL compiler
  - Dynamic remote method invocation
- Interface Repository
  - Querying existing remote interfaces
- Implementation Repository
  - Activating remote objects on demand

CORBA IDL

- Definition of language-independent remote interfaces
  - Language mappings to C++, Java, Smalltalk, …
  - Translation by IDL compiler
- Type system
  - basic types: long (32 bit),
  - long long (64 bit), short, float, char, boolean, octet, any, …
  - constructed types: struct, union, sequence, array, enum
  - objects (common super type Object)
- Parameter passing
  - in, out, inout
  - basic & constructed types passed by value
  - objects passed by reference

```
typedef sequence<string> Files;  interface
PrintService : Server {
void print(in Files printJob); };
```

CORBA Services

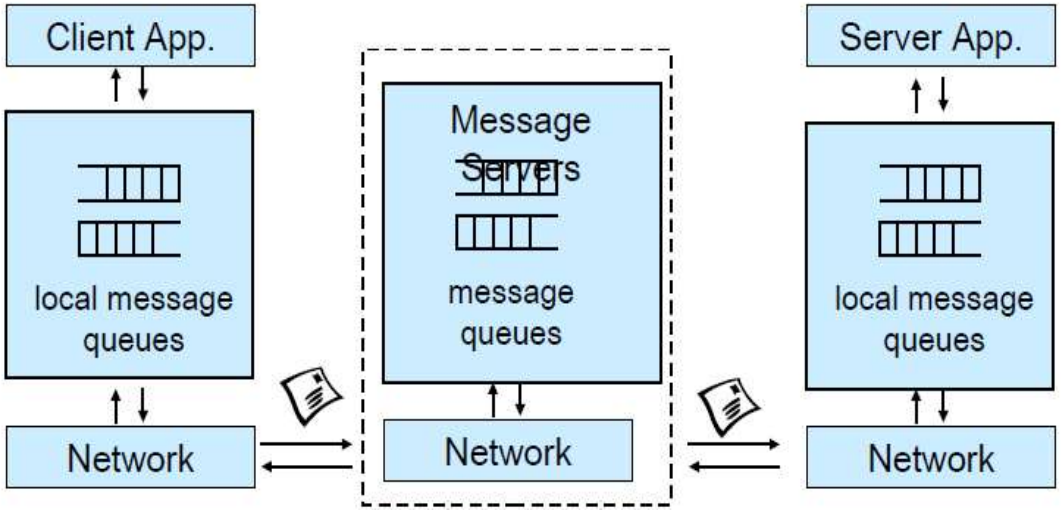
- Naming Service
  - Names → remote object references
- Trading Service
  - Attributes (properties) → remote object references
- Persistent Object Service
  - Implementation of persistent CORBA objects
- Transaction Service
  - Making object invocation part of transactions
- Event Service and Notification Service
  - In response to applications' need for asynchronous communication
  - built above synchronous communication with push or pull options
  - not an integrated programming model with general IDL messages

CORBA - Disadvantages

- Synchronous request/reply interaction only
  - So CORBA oneway semantics added and -
  - Asynchronous Method Invocation (AMI)
  - But implementations may not be loosely coupled
- Distributed garbage collection
  - Releasing memory for unused remote objects
- OOM rather static and heavy-weight
  - Bad for ubiquitous systems and embedded devices

Message-Oriented Middleware (MOM)

- Communication using messages
- Messages stored in message queues
- message servers decouple client and server
- Various assumptions about message content



Properties of MOM

- Asynchronous interaction
  - Client and server are only loosely coupled
  - Messages are queued
  - Good for application integration
- Support for reliable delivery service
  - Keep queues in persistent storage
- Processing of messages by intermediate message server(s)
  - May do filtering, transforming, logging, …
  - Networks of message servers
- Natural for database integration

Message-Oriented Middleware (MOM)

IBM MQ-Series

- One-to-one reliable message passing using queues
  - Persistent and non-persistent messages
  - Message priorities, message notification
- Queue Managers
  - Responsible for queues
  - Transfer messages from input to output queues
  - Keep routing tables
- Message Channels
  - Reliable connections between queue managers

Java Messaging Service (JMS)

- API specification to access MOM implementations
- Two modes of operation *specified*:
- **Point-to-point:** one-to-one communication using queues
- **Publish/Subscribe:** Event-Based Middleware
- JMS Server implements JMS API
- JMS Clients connect to JMS servers
- Java objects can be serialised to JMS messages
- A JMS interface has been provided for MQ
- pub/sub (one-to-many) - just a specification?

MOM - Disadvantages

- Poor programming abstraction (but has evolved)
  - Rather low-level (cf. Packets)
  - Request/reply more difficult to achieve, but can be done
- Message formats originally unknown to middleware
  - No type checking (JMS addresses this – implementation?)
- Queue abstraction only gives one-to-one communication
  - Limits scalability (JMS pub/sub – implementation?)

Web Services

- Use well-known web standards for distributed computing

- **Communication**
  - Message content expressed in **XML**
  - **S**imple **O**bject **A**ccess **P**rotocol (**SOAP**)
    – Lightweight protocol for sync/async communication
- **Service Description**
  - **W**eb **S**ervices **D**escription **L**anguage (**WSDL**)
    – Interface description for web services
- **Service Discovery**
  - **U**niversal **D**escription **D**iscovery and **I**ntegration (**UDDI**)
    – Directory with web service description in WSDL

Properties of Web Services

- Language-independent and open standard
- SOAP offers OOM and MOM-style communication:
  - Synchronous request/reply like OOM
  - Asynchronous messaging like MOM
  - Supports internet transports (http, smtp, …)
  - Uses XML Schema for marshalling types to/from programming language types
- WSDL says how to use a web service
  - http://api.google.com/GoogleSearch.wsdl
- UDDI helps to find the right web service
  - Exports SOAP API for access

Web Services

Disadvantages of WS

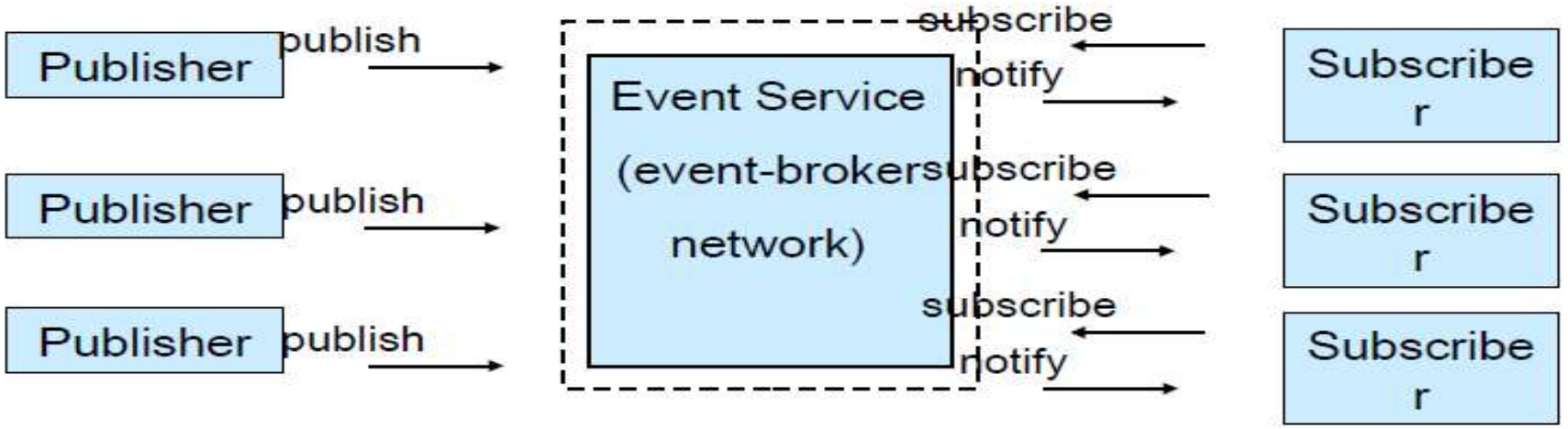
- Low-level abstraction
  - leaves a lot to be implemented
- Interaction patterns have to be built
  - one-to-one and request-reply provided
  - one-to-many?
  - still synchronous service invocation, rather than notification
  - No nested/grouped invocations, transactions, …
- No location transparency

What we lack so far ?

- General interaction patterns
  - we have one-to-one and request-reply
  - one-to-many? many to many?
  - notification?
  - dynamic joining and leaving?
- Location transparency
  - anonymity of communicating entities
- Support for pervasive computing
  - data values from sensors
  - lightweight software

Event-Based Middleware a.k.a. Publish/Subscribe

- **Publishers** (*advertise* and) *publish* **events** (messages)
- **Subscribers** express interest in events with *subscriptions*
- **Event Service** *notifies* interested subscribers of published events
- Events can have arbitrary content (typed) or name/value pairs



Topic-Based and Content-Based Pub/Sub

- Event Service matches events against subscriptions
  - What do subscriptions look like?
- Topic-Based Publish/Subscribe
  - Publishers publish events belonging to a topic or subject
  - Subscribers subscribe to a topic subscribe(PrintJobFinishedTopic, …)
- (Topic and) Content-Based Publish/Subscribe
  - Publishers publish events belonging to topics and
  - Subscribers provide a filter based on *content* of events
    subscribe(type=printjobfinished, printer='aspen', …)

Properties of Pub/Sub

- Asynchronous communication
  - Publishers and subscribers are loosely coupled
- Many-to-many interaction between pubs. and subs.
  - Scalable scheme for large-scale systems
  - Publishers do not need to know subscribers, and vice-versa
  - Dynamic join and leave of pubs, subs,
- (Topic and) Content-based pub/sub very expressive
  - Filtered information delivered only to interested parties
  - Efficient content-based routing through a broker network

Summary

- Middleware is an important abstraction for building distributed systems
  1. Remote Procedure Call
  2. Object-Oriented Middleware
  3. Message-Oriented Middleware
  4. Event-Based Middleware
- Synchronous vs. asynchronous communication
- Scalability, many-to-many communication
- Language integration
- Ubiquitous systems, mobile systems

Thank You


STOP REC