IS 651: Distributed Systems Chapter 2: The Evolution of Distributed Systems

Jianwu Wang

Spring 2021

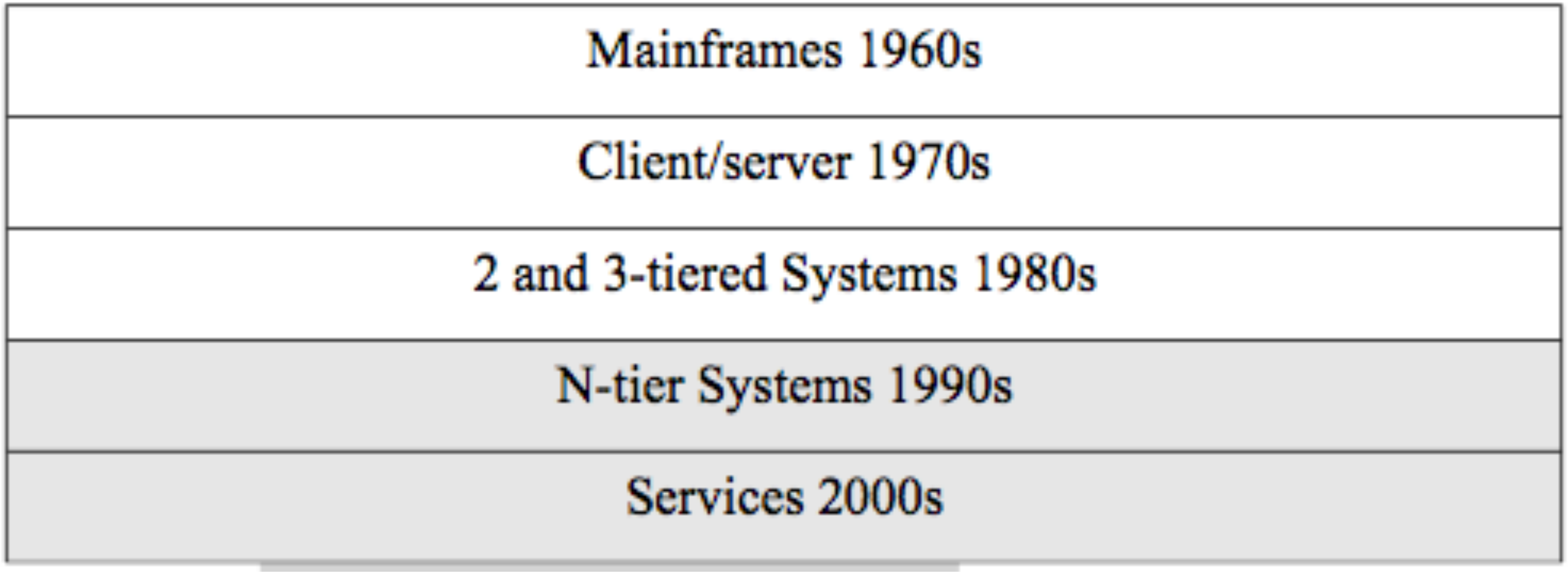
Notes

- Please use Piazza, not email, for questions
    - You are very welcome to reply others' questions
    - You can have public/private posts
    - You can have pictures in your posts
    - Do not ask how to do the exercise/homework, ask clarification questions
    - Do not share your solutions with others
    - Show courtesy and update whether the help you got works
- Lecture videos will be on blackboard (Course Videos) once they are available
- Assignment presentation: To show diverse solutions, members in the same team should avoid presenting the same assignment
- All homework/exercises are due on Thursday
- Course website is https://userpages.umbc.edu/~jianwu/is651/651.syll.s21.html, and reference page is https://userpages.umbc.edu/~jianwu/is651/651.ref.s21.html

Learning Outcomes

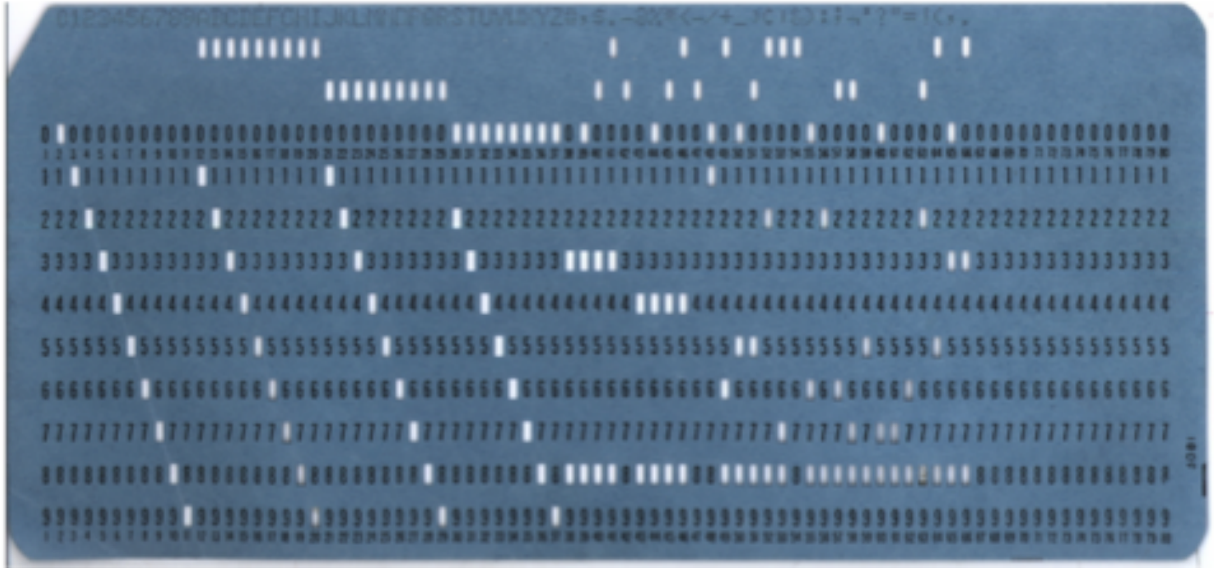
- After learning chapter 2, you should be able to
  - Understand the different generations of distributed systems and the reason for the evolution
  - Understand new terms from the chapter: middleware, remote procedure call (RPC), message-oriented, Transaction, etc.
  - Write XML documents and validate them using DTD

Basic Timeline of Distributed System Evolution

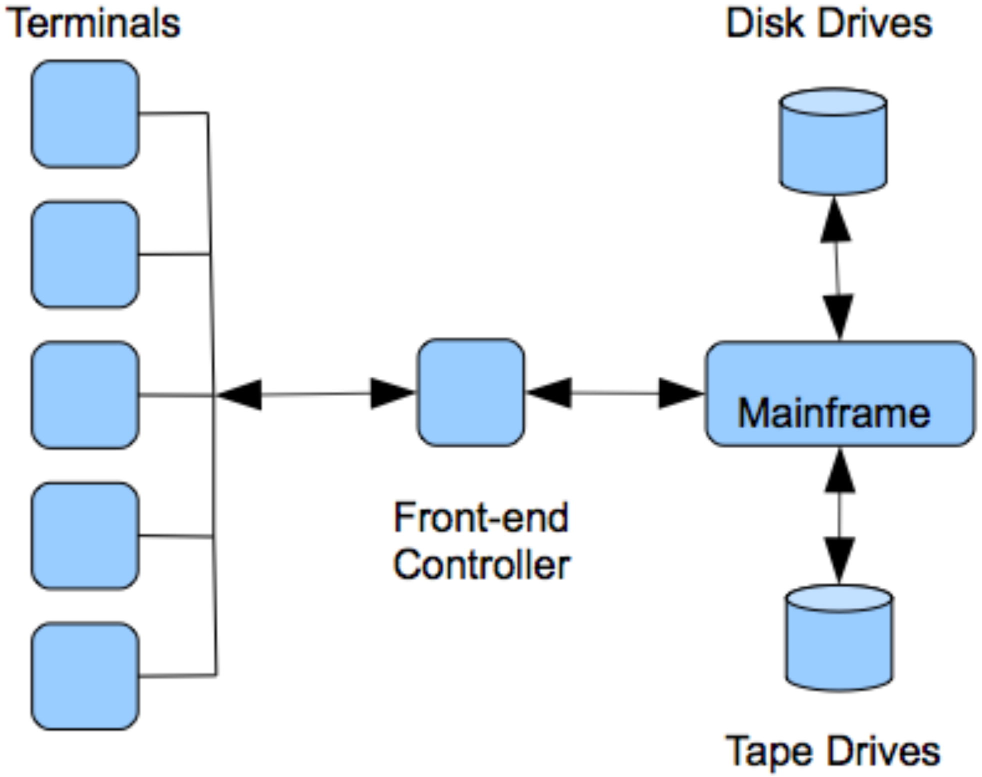


Mainframes 1960s
Client/server 1970s
2 and 3-tiered Systems 1980s
N-tier Systems 1990s
Services 2000s

Mainframe (1960s)



Punch Card



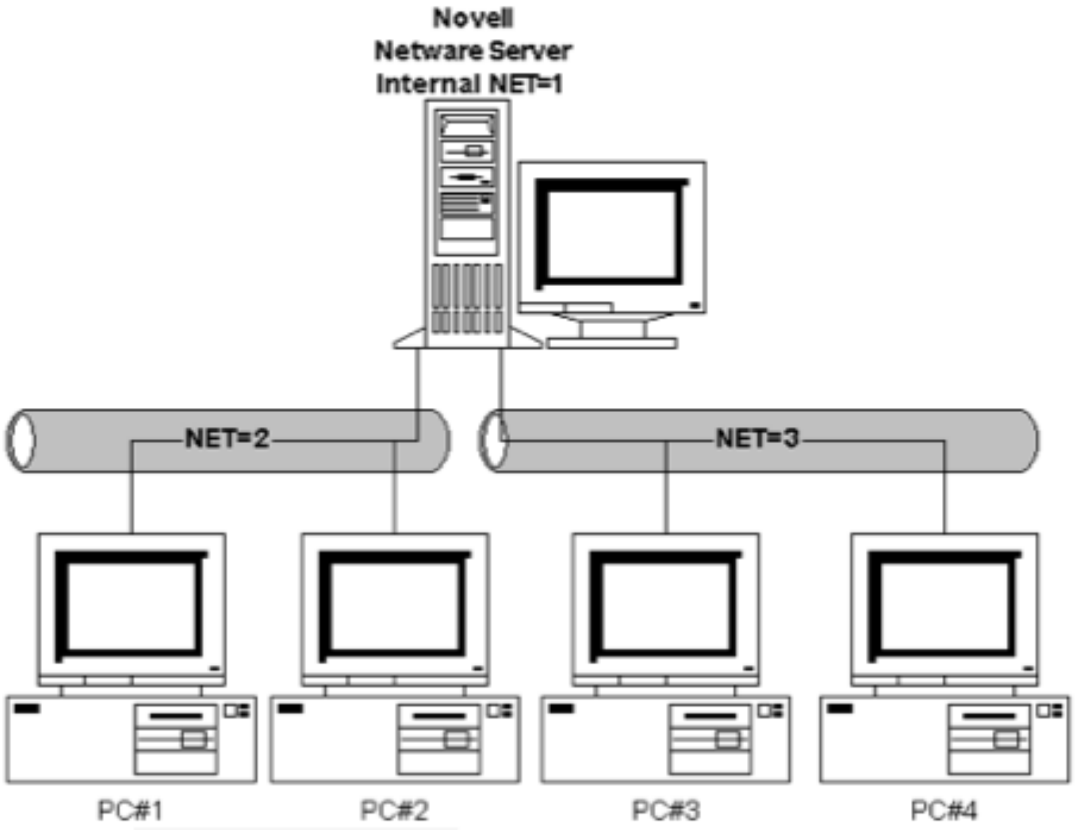
A typical mainframe architecture

Client/Server (1970s)

- Minicomputers: smaller computers (not Personal Computer yet)
- Ethernet: form local-area networking (LAN)
- X.25: wide-area networking (WAN) service
- Client server architecture
  - A client is a requestor process and a server is a responder process
  - One machine could be both client and server
- Beginnings of the Internet
  - ARPANET
  - TCP/IP stack

2 and 3-tier Systems (1980s)

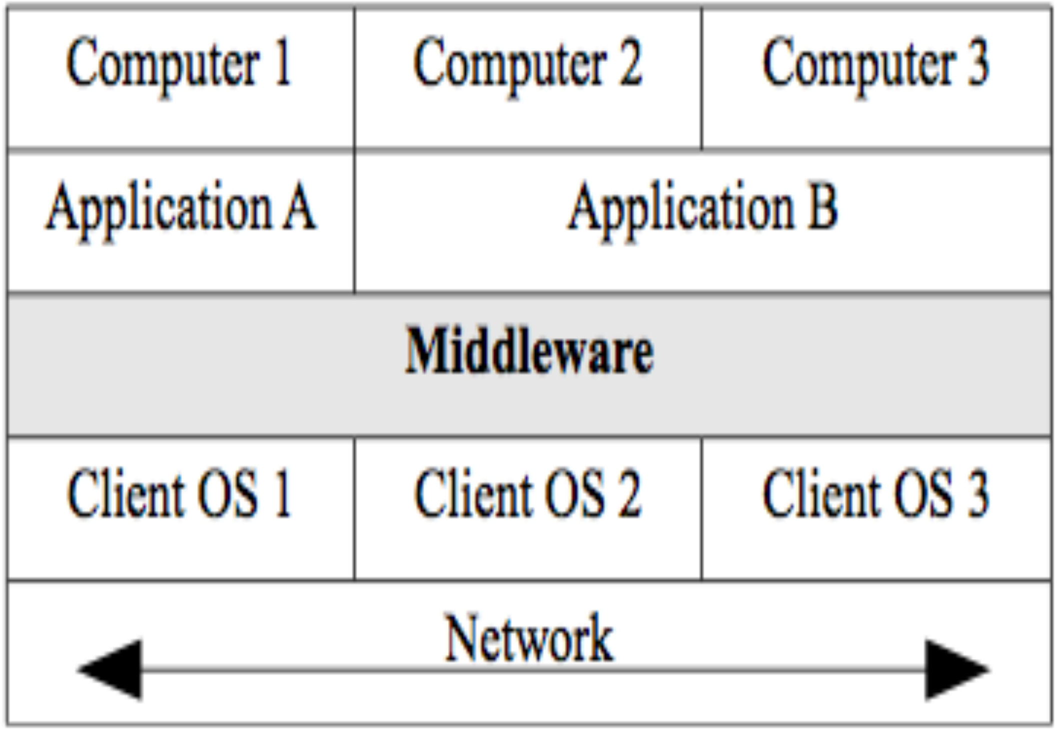
- Personal Computer
- NetWare file servers
- Network file system (NFS)
- Remote procedure call (RPC)
- 3-tier system: with an architectural middle tier, called the application server, and associated middleware
- Middleware



A 2-tier system

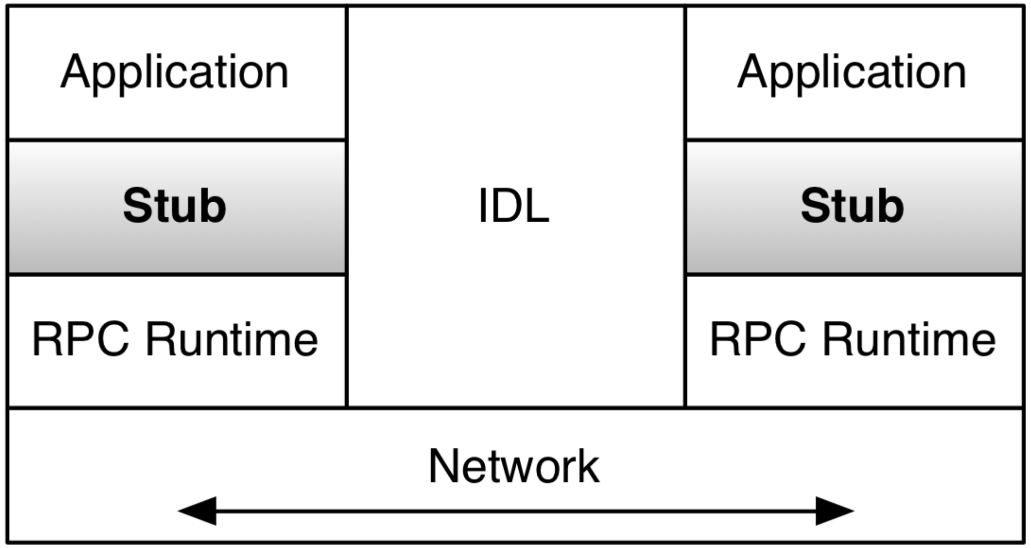
Middleware

- Middleware is the software layer that lies between the operating system and the applications on each side of a distributed computer network
- Middleware offers **general services** that can be used by many applications
  - Remote procedural call (RPC)
  - Distributed cache
  - Message queue
- Major types of middleware
  - Remote procedural call middleware
  - Message-oriented middleware



Remote Procedure Call (RPC) Middleware

- The application calls the remote procedure locally at the stub
- The stub intercepts calls that are for remote servers
  - Marshalling: pack the parameters into a message
  - Make a system call to send the message
- The RPC Runtime handles message sending
- The interface definition language (IDL) handles message translation
- RPC hides heterogeneity among the computers and handles the communication across network



Application

Stub

RPC Runtime

IDL

Application

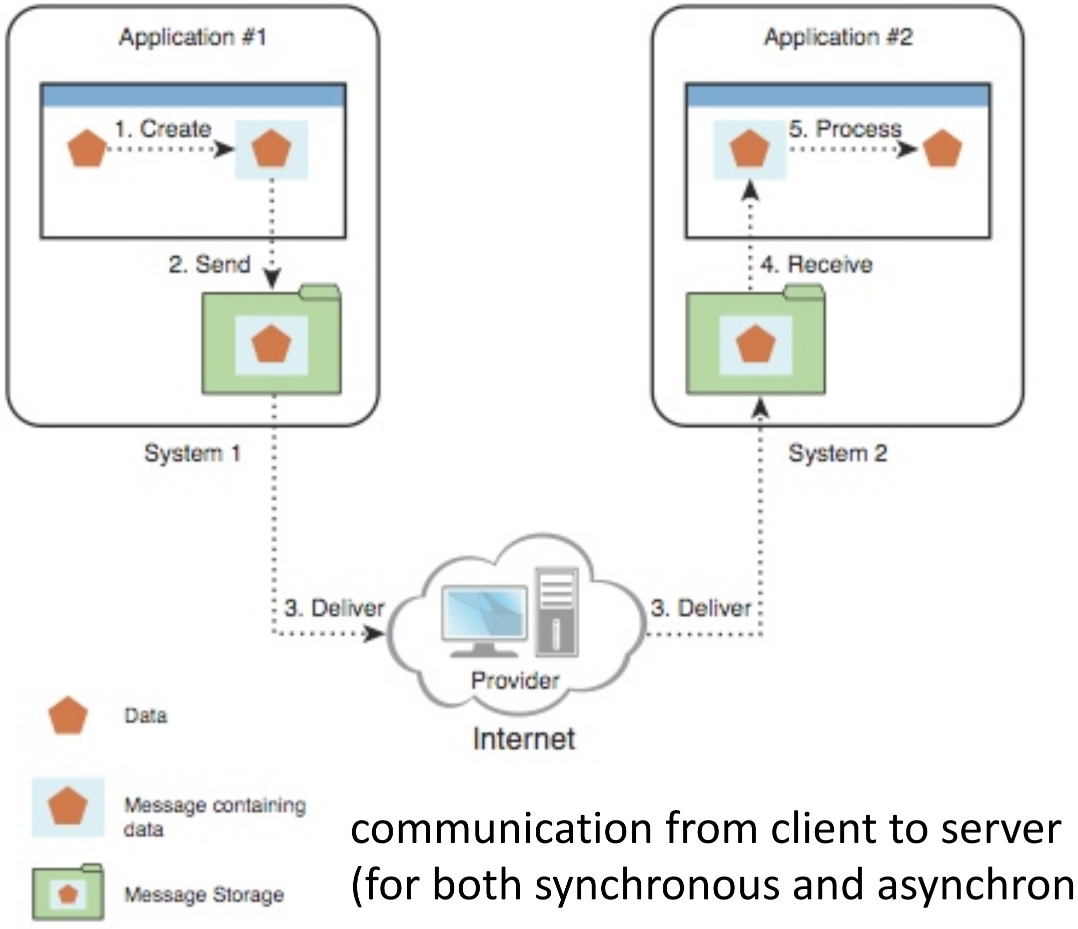
Stub

RPC Runtime

Network

Messaging Modes of Communication

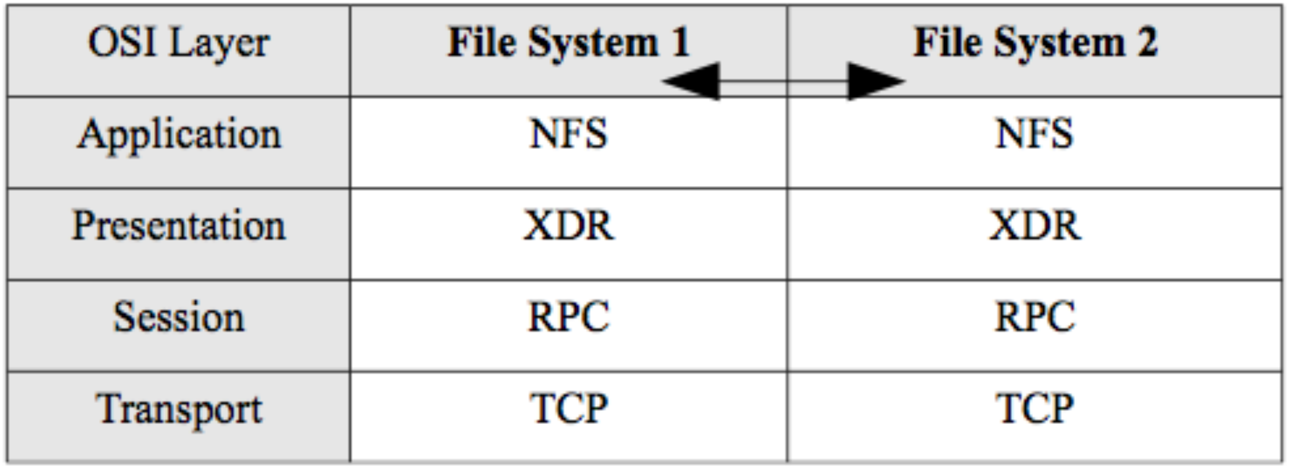
- Synchronous (blocking)
  - RPC protocol is synchronous
  - When a client makes a remote call, the calling process blocks or waits until it gets a reply
- Asynchronous
  - The calling process just goes back to processing and is interrupted with a callback message when it does get the response
  - Message-oriented protocol supports it



communication from client to server (for both synchronous and asynchronous)

Distributed File Systems

- A type of RPC middleware
- Allows users to mount directories from remote computers into their own local directory, so they appear as local
- NFS: network file system
- XDR: external data representation

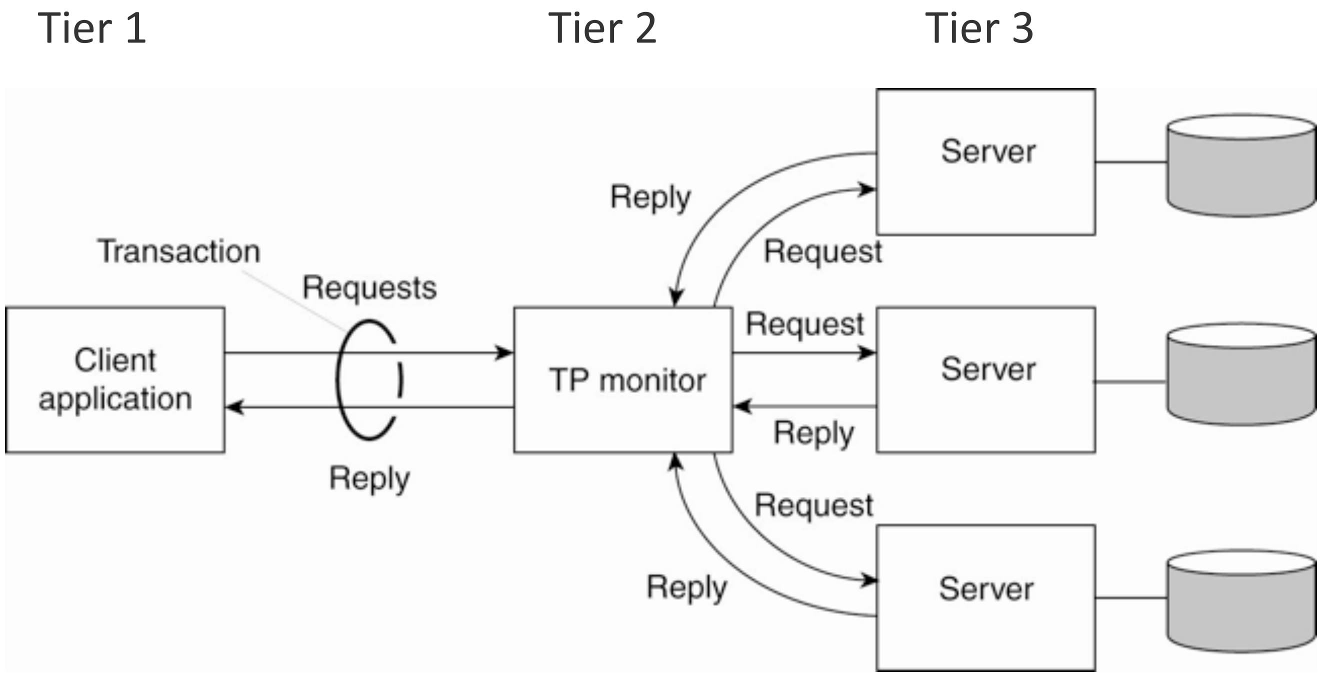


OSI Layer	**File System 1**	**File System 2**
Application	NFS	NFS
Presentation	XDR	XDR
Session	RPC	RPC
Transport	TCP	TCP

NFS distributed file system protocol stack

Transaction Middleware

- A database RPC middleware uses an explicit 3-tier architecture
- Transaction processing monitor (TPM) at middleware tier



Transactions

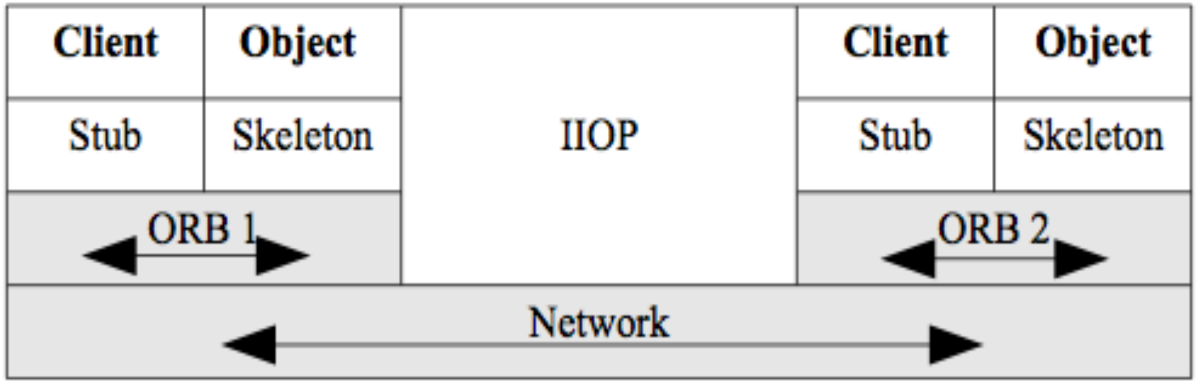
- All the participating operations on (distributed) resources should either *succeed* or *fail and recover* together
- 2-Phase Commit
  - commit-request phase: TPM request all the servers to commit and wait responses
  - commit phase: TPM decides either commit or abort based on responses
- A transaction is a unit of work with the following ACID properties
  - ATOMICITY: A transaction should be done or undone completely and unambiguously
  - CONSISTENCY: A transaction should transform the system from one consistent state to another consistent state
  - ISOLATION: Each transaction should appear to execute independently of other transactions that may be executing concurrently in the same environment
  - DURABILITY: The effects of a completed transaction should always be persistent

Object-Oriented RPC Middleware

- RPC-based distributed systems based on object-oriented programming principles
- Two main technologies
    - Common Object Request Broker Architecture (CORBA): a standard designed to facilitate the communication of systems that are deployed on diverse platforms
    - Distributed Component Object Model (DCOM): a proprietary Microsoft technology for software distributed across several networked computers to communicate with each other

CORBA

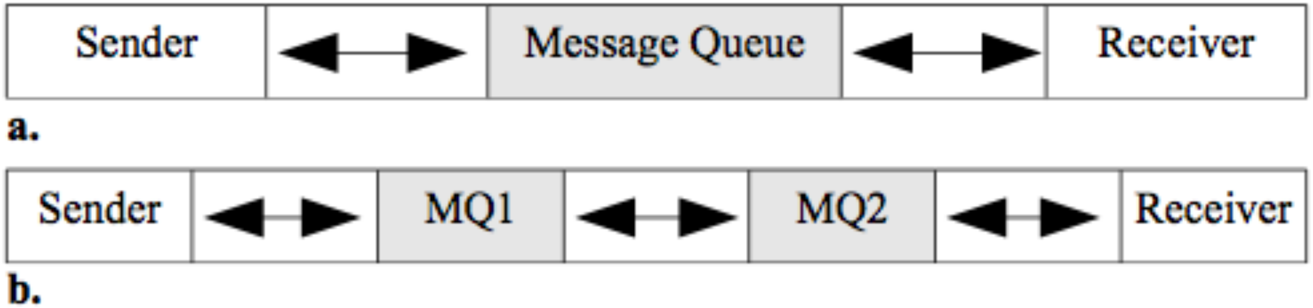
- Skeleton: stub for server object
- Object request broker (ORB)
  - Mediates a method call from one object to another local/remote object
- Internet inter-ORB protocol (IIOP)
  - Allows ORBs from different vendors to communicate over the Internet
- The client cannot tell whether the target object it communicates with is local or remote



common object request broker architecture (CORBA)

Message-Oriented Middleware (MOM)

- Point-to-point messaging (PTP): 1 to 1
  - Messages are sent to a queue, rather than directly to the intended receiver

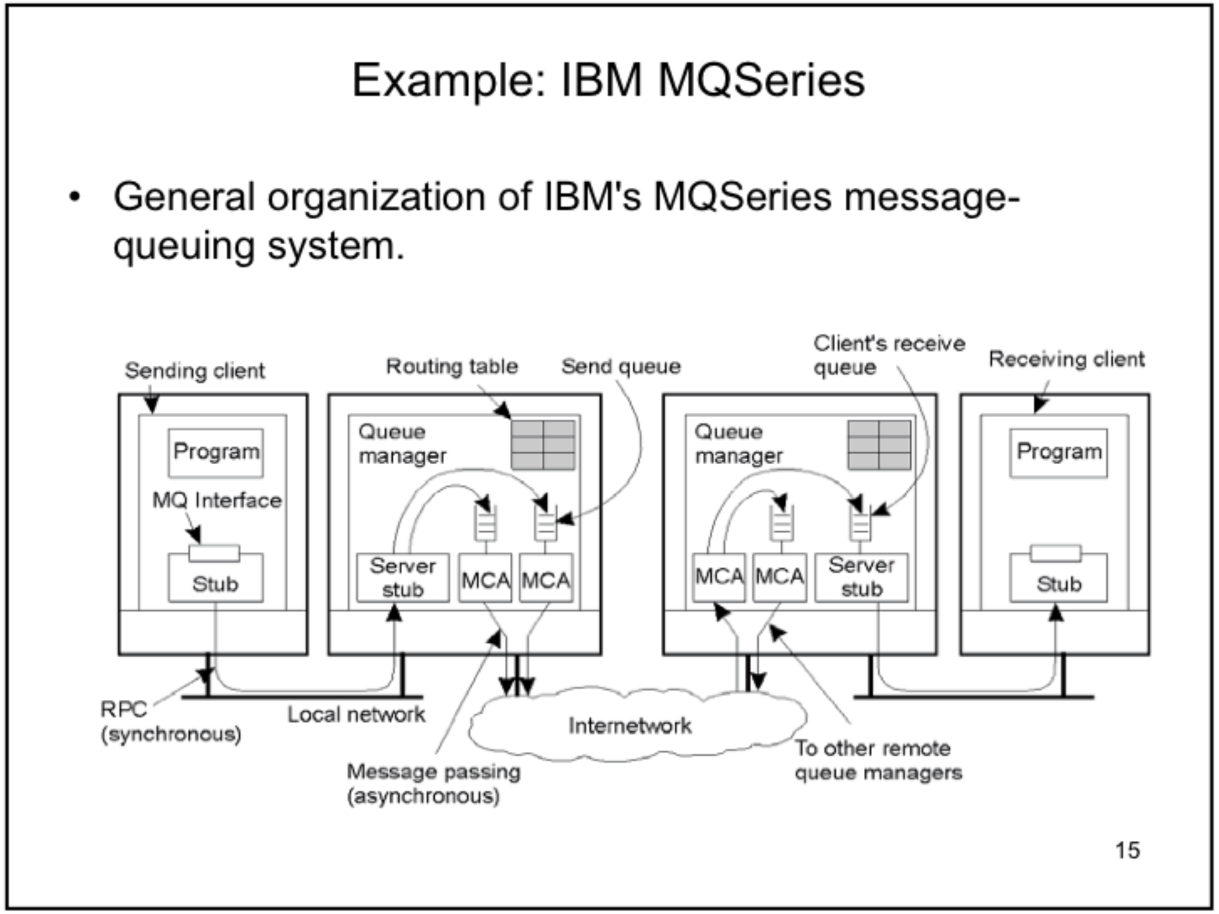


Message Queues (MQ).

- Publish/Subscribe messaging (pub/sub): M to N
- MOM is based on RPC
  - MOM uses queues to give **asynchronous** communication from the viewpoint of the sender and receiver

Point-to-point Messaging (P2P) Example

- MQSeries shows how P2P architecture and asynchronous communication are achieved using RPC protocols
- Message channel agent (MCA): controls message sending and receiving



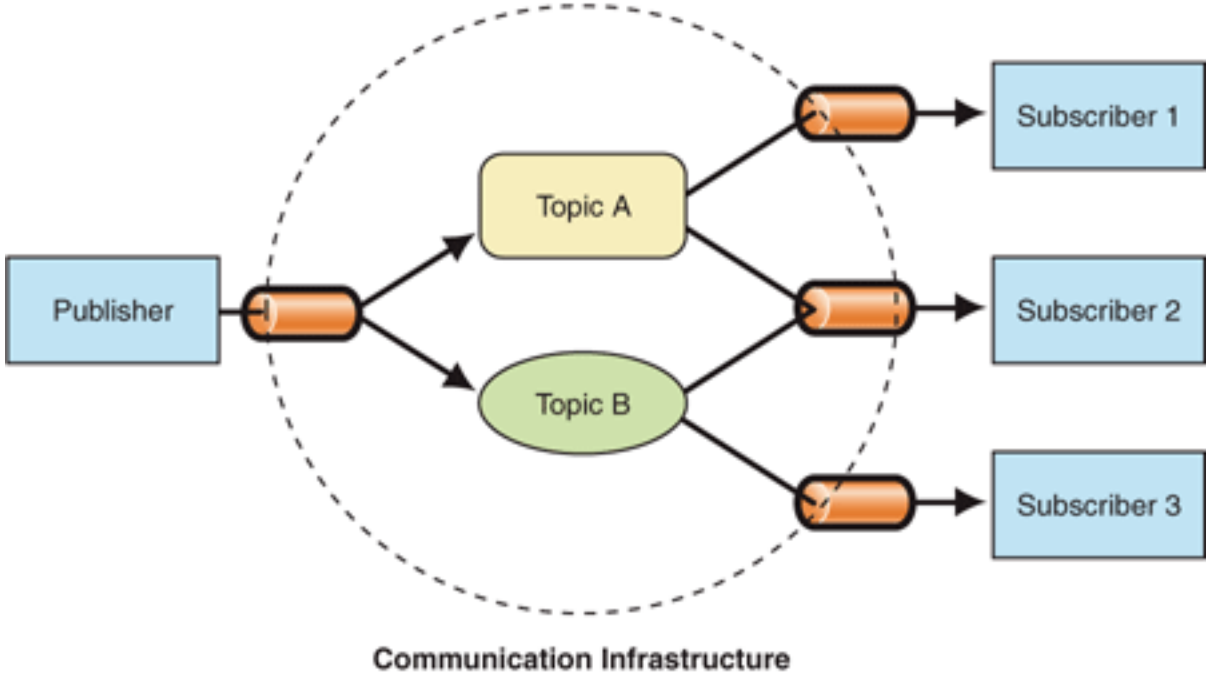
Example: IBM MQSeries

- General organization of IBM's MQSeries message-queuing system.

15

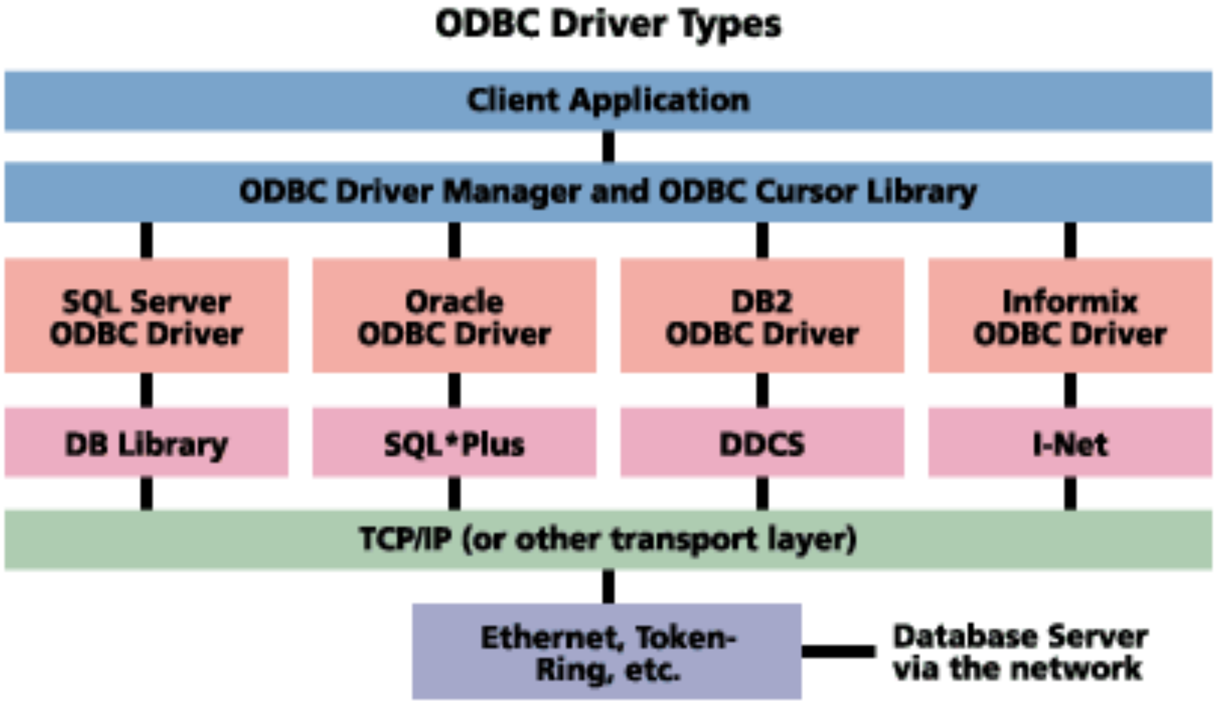
Pub/Sub

- The Pub/Sub model is an excellent message delivery model appropriate for multiple senders and multiple recipients
  - Each publisher can send out messages for multiple topics
  - Each subscriber can decide which topics he/she is interested



Database Access via ODBC

- It shows a **client-side** type of middleware
- Open Database Connectivity (ODBC)
  - A standard programming language middleware API for accessing database management systems
  - The same client application uses the different ODBC drivers to access different types of databases
- Java Database Connectivity (JDBC): an API for Java



ODBC Driver Types

Client Application

ODBC Driver Manager and ODBC Cursor Library

SQL Server ODBC Driver

Oracle ODBC Driver

DB2 ODBC Driver

Informix ODBC Driver

DB Library

SQL*Plus

DDCS

I-Net

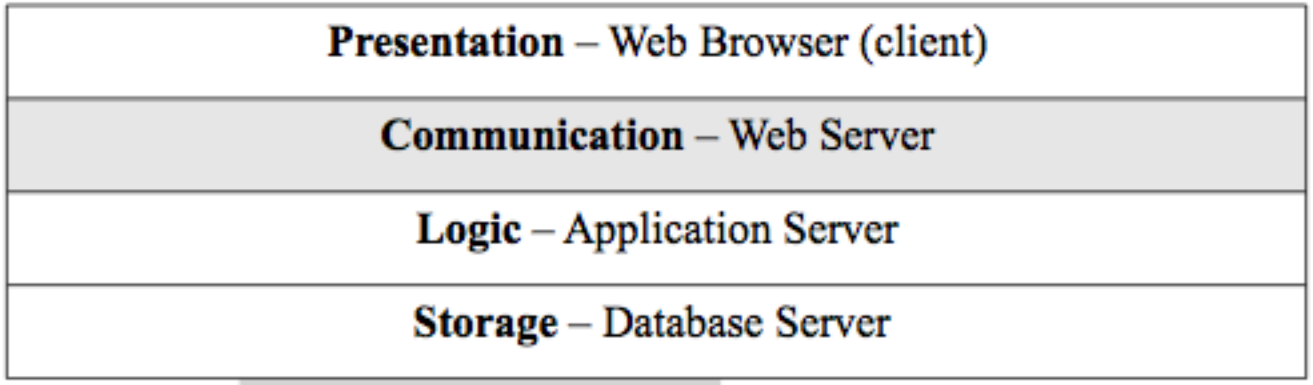
TCP/IP (or other transport layer)

Ethernet, Token-Ring, etc.

Database Server via the network

N-tier Systems (1990s)

- N-tier systems are not a different approach than 3-tier systems, they are just an elaboration of the same pattern
- Web server: serves content to the web using http protocol
- Application server: hosts and exposes business logic and processes



Presentation – Web Browser (client)
Communication – Web Server
Logic – Application Server
Storage – Database Server

LAMP Web Scripting with N-tier Systems

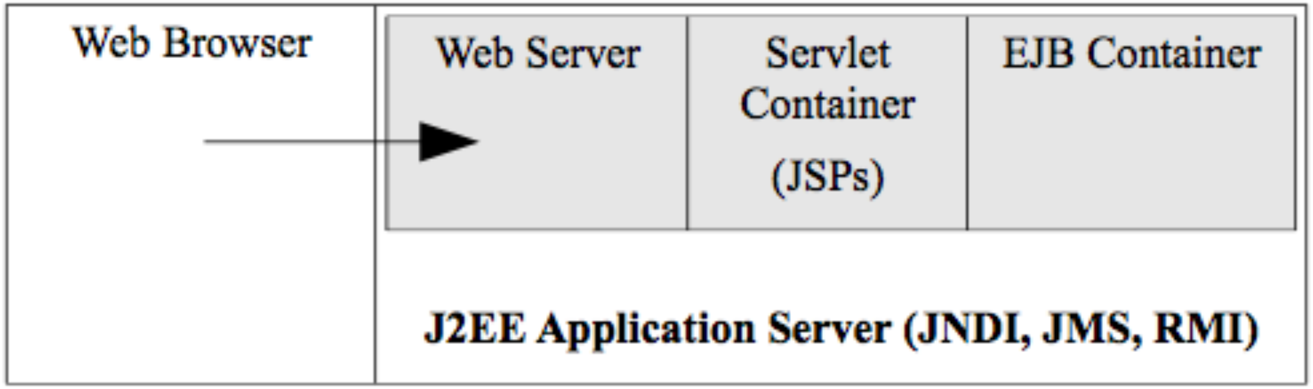
- L – the operating system. L stands for Linux as the most common one, but any operating system can be used such as Windows.
- A – the web sever. A stands for Apache HTTP Server, as the most popular open source web server, but any web server may be used.
- M – the database. M stands for MySql as a popular open-source relational database, but any database may be used.
- P – the scripting language. P originally stood for Perl which is a popular scripting language and oddly enough, many scripting languages begin with P such as Python and PHP.
    - Any scripting language may be used, however, such as Ruby and JavaScript.
    - Scripting languages are characterized as interpreted and dynamically typed.

MEAN Web Scripting with N-tier Systems

- M – MongoDB, a NoSQL database
- E – Express.js, a web application framework that runs on Node.js
- A – Angular, a JavaScript MVC (model, view, control) framework that runs in browser JavaScript engines
- N – Node.js, an execution environment for event-driven server-side and networking applications
- MEAN applications can be written in one language, namely JavaScript, for both server-side and client-side execution environments.
  - An open source project by IS students: https://github.com/rogueriderhood/mean-project/

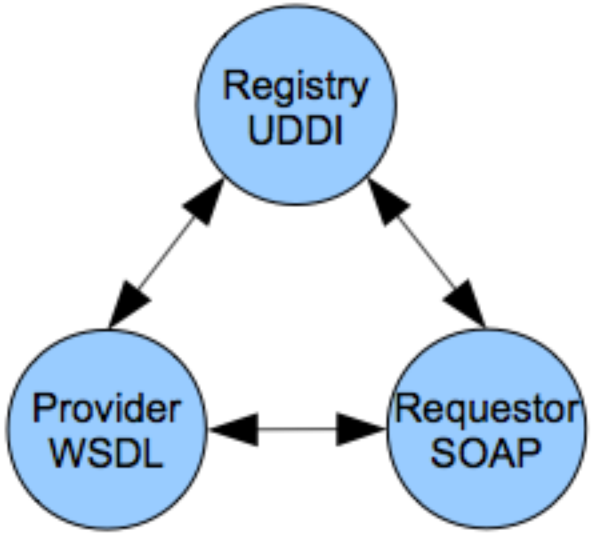
J2EE (Enterprise Edition) Application Server

- Java naming and directory interface (JNDI): A naming service for containers
- The Java messaging service (JMS): MOM service offered by Java frameworks
- Remote method invocation (RMI): Java framework version of object-oriented RPC
- Servlet Container: a server-side software component for objects to receive a requests and generate responses. A servlet is often built as Java Server Pages (JSPs)
- Enterprise JavaBeans (EJB) : a server-side software component for business logic
- It is still widely used and can provide Web services



Services (2000s)

- Standard service contracts
  - Participants have agreements
  - They should also be discoverable by using some kind of registry or directory
- Loose coupling
  - The participants have minimal dependencies on each other
- Encapsulation
  - Services should hide their logic from the outside world as a black box
  - This increases flexibility, reusability and increases composability
  - Services should also have location transparency where users do not care where the services are located
- Statelessness
  - Keep track of as little state as possible
  - This is a requirement for loose coupling and encapsulation



Principles in Distributed System Evolution

- A lot of concepts/components were developed to enable network-based communication among distributed computers via messages
  - Marshalling, RPC Runtime, IIOP, IDL, message queue, MCA, etc.
- Some new techniques/models are built on top of existing techniques/models
  - RPC -> MOM -> Pub/Sub
  - Many seemingly different techniques/models (NFS, ODBC, transaction middleware, CORBA and DCOM) are all built on top of RPC
- Some new techniques/models are extensions of existing techniques/models
  - 2 tier -> 3 tier -> n tier -> service

Extensible Markup Language (XML)

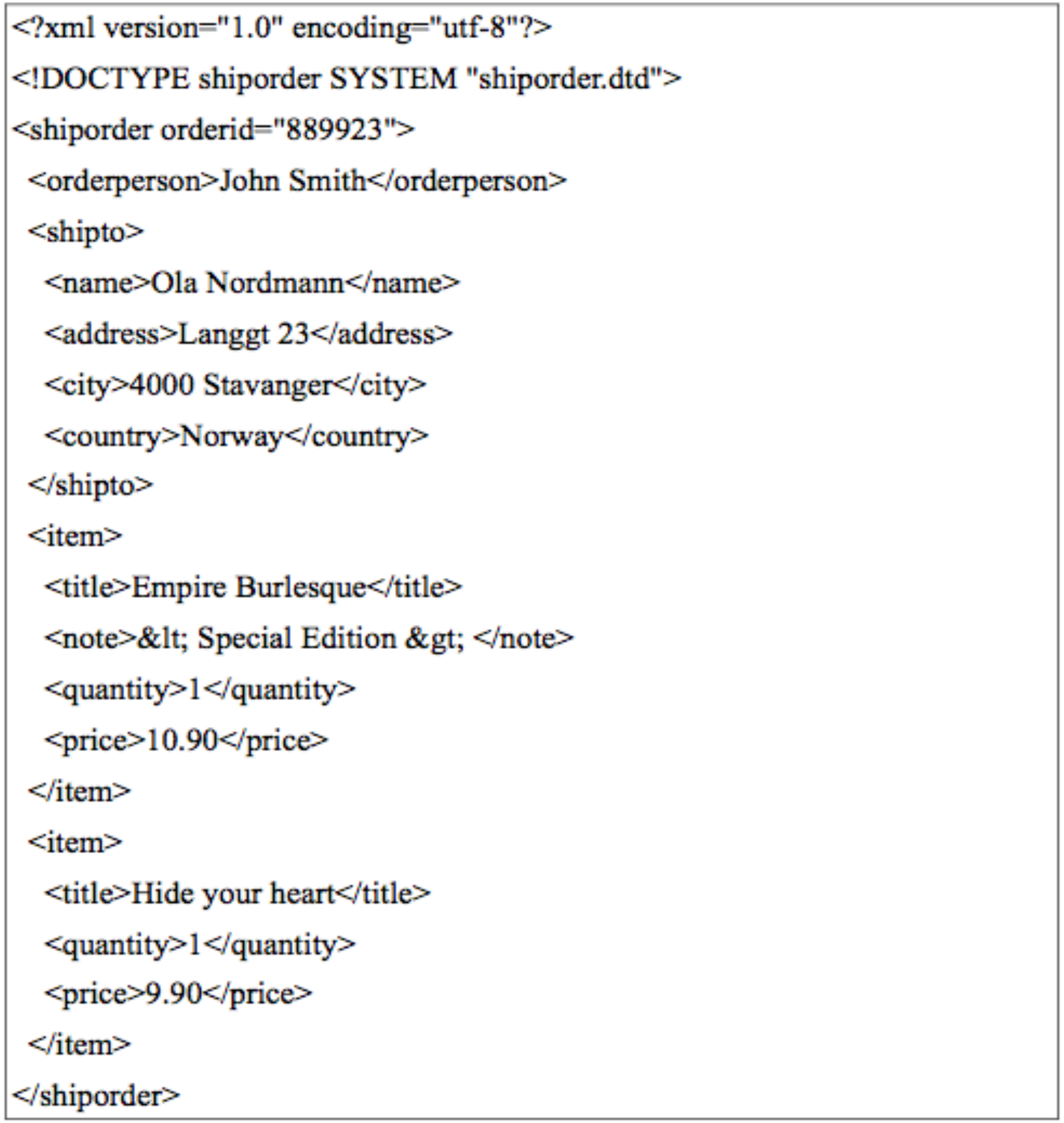
- Markup language: text document with annotation (normally using tags)
  - **H**yper**T**ext **M**arkup **L**anguage (HTML)
  - E**X**tensible **M**arkup **L**anguage (XML)
  - E**X**tensible **H**yper**T**ext **M**arkup **L**anguage (XHTML)
  - etc.
- XML documents form a tree structure
- Well-formed XML VS. Valid XML
- XML validation
  - Document type definition (DTD)
  - XML Schema

Well-formed XML

- It contains only properly encoded, legal Unicode characters
- None of the special syntax characters (<, &) appear except when performing their markup-delineation roles
- The begin, end, and empty-element tags that delimit the elements are correctly nested, with none missing and none overlapping
- The element tags are case-sensitive - the beginning and end tags must match exactly
- There is a single "root" element that contains all the other elements
- Well-form check command: $>xmllint --noout shiporder.xml

Valid XML

- The declaration in line 1 is contains question mark characters and is called a **processing instruction**. It refers the version and encoding for the XML document
- Line 2 has a reference to an external DTD file that contains the DTD
  - It can be replaced by embedding DTD content
- Line 3 is the root **tag** for the document. Note that it contains an **attribute**. Any XML tag may have an attribute and it must be quoted.
- Note that even though *item* is repeated, it uses the same tag. Never create tags like item1, item2, etc

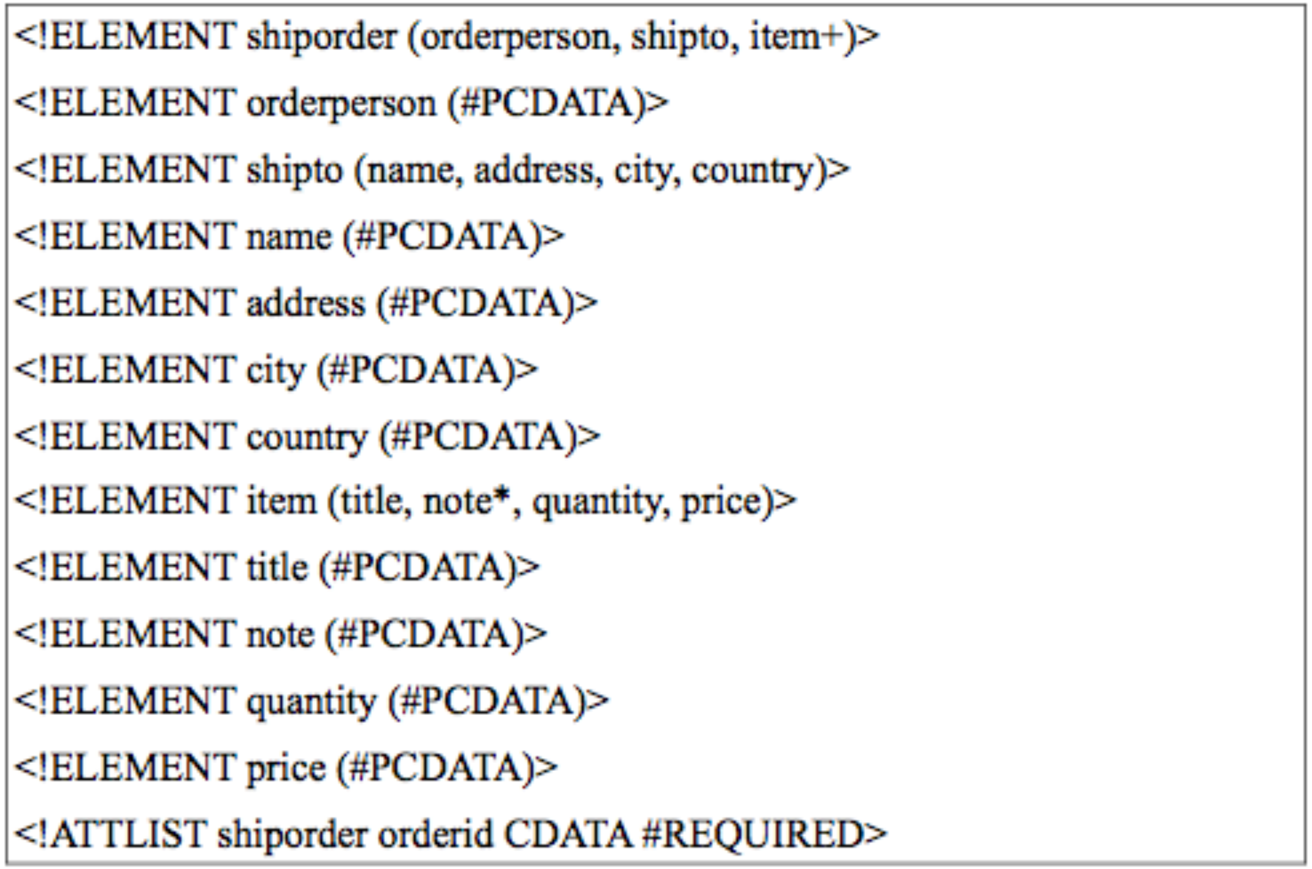


```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE shiporder SYSTEM "shiporder.dtd">
<shiporder orderid="889923">
 <orderperson>John Smith</orderperson>
 <shipto>
  <name>Ola Nordmann</name>
  <address>Langgt 23</address>
  <city>4000 Stavanger</city>
  <country>Norway</country>
 </shipto>
 <item>
  <title>Empire Burlesque</title>
  <note>&lt; Special Edition &gt; </note>
  <quantity>1</quantity>
  <price>10.90</price>
 </item>
 <item>
  <title>Hide your heart</title>
  <quantity>1</quantity>
  <price>9.90</price>
 </item>
</shiporder>
```

Document Type Definition (DTD)

- The declaration of the DTD in the XML document has the syntax where *SYSTEM* refers to that fact that the DTD is a private implementation for this document rather than a standard. It would change to PUBLIC if it was a standard.
    - <!DOCTYPE root-element SYSTEM "file.dtd" >
- DTDs do not have XML syntax. They have their own syntax.
- The !ELEMENT declares an element (also called a tag).
- The child elements of a tag are declared as an ordered list in parentheses. If an element can be repeated 1 or more times, it must have a plus sign (+) after it. The character star (*) means 0 or more and so makes elements optional.
- A leaf node of the hierarchy is declared #PCDATA which means **parsed character data** and it is the text of the content.
- The *&lt; and &gt;* are XML built-in entities for the less than and greater than (< >) characters. XML markup characters cannot be used because they would confuse a parser, so these pre-defined entities must replace them.
- There are no data types in DTDs. Everything is text.
- The !ATTLIST declares an attribute for an element and typically declares it as CDATA which means **character data**. This means that the XML parser does not parse it.
- One can require a document to have an attribute in order to be valid by using #REQUIRED.

DTD Example



```
<!ELEMENT shiporder (orderperson, shipto, item+)>
<!ELEMENT orderperson (#PCDATA)>
<!ELEMENT shipto (name, address, city, country)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT item (title, note*, quantity, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST shiporder orderid CDATA #REQUIRED>
```

- Validation command: $>xmllint --noout --valid shiporder.xml

Demo

- Well-form check command
  - xmllint --noout shiporder.xml
  - xmllint --noout shiporder-not-well-formed.xml
  - xmllint --noout shiporder-well-formed-not-valid.xml
- Validation check command
  - xmllint --noout --valid shiporder.xml
  - xmllint --noout --valid shiporder-well-formed-not-valid.xml