

Middleware and Interprocess Communication

Reading

- Coulouris (5th Edition): 4.1, 4.2, 4.6
- Tanenbaum (2nd Edition): 4.3

Outline

- **Middleware**
- Introduction to Interprocess Communication
- MPI

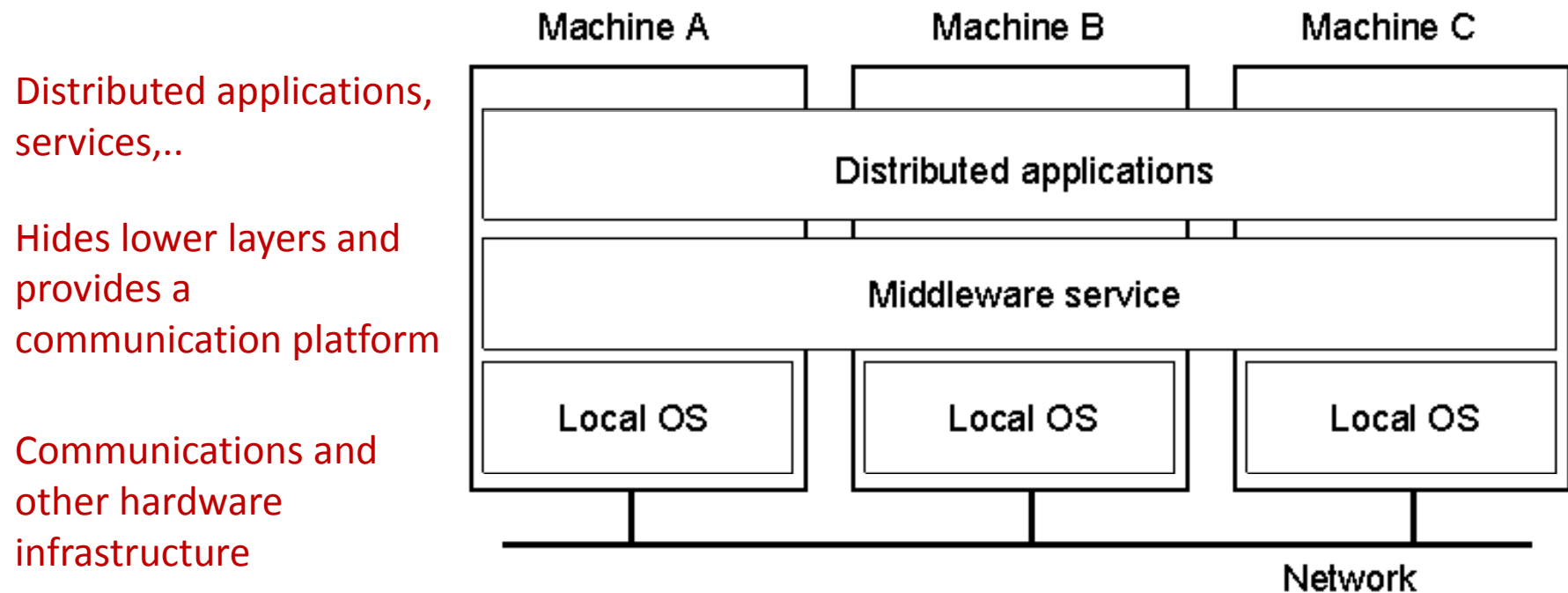
Middleware

- A software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages.

Properties of Middleware

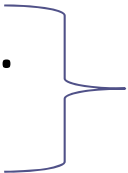
- It mostly, refers to the distributed system layer that enables communication between distributed systems.
- Mask the heterogeneity of the operating system, hardware, and network layers.
- Provides a uniform computational model for use by the programmers of servers and distributed applications.

Middleware Layer



Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Middleware Examples

- Common Object Request Broker (CORBA). It provides remote object invocation, which allows an object in a program running on one computer to invoke a method of an object in a program running on another computer.
 - Java Remote Method Invocation (RMI).
 - Remote Procedure Call (RPC).
 - Message Oriented Middleware (MOM).
 - Web services.
 - Distributed transaction processing.
- 
- Tight coupling.
Both caller and
callee need to
be running

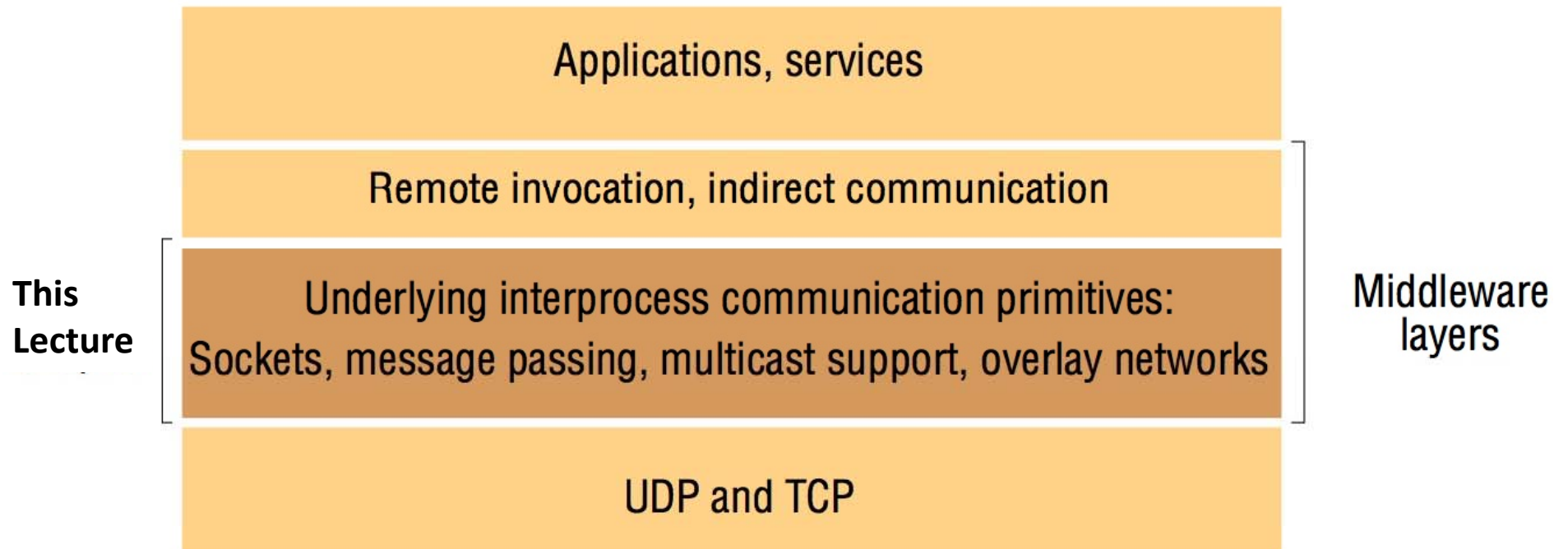
Categories of Middleware

<i>Major categories:</i>	<i>Subcategory</i>	<i>Example systems</i>
<i>Distributed objects (Chapters 5, 8)</i>	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
<i>Distributed components (Chapter 8)</i>	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
<i>Publish-subscribe systems (Chapter 6)</i>	-	CORBA Event Service
	-	Scribe
	-	JMS
<i>Message queues (Chapter 6)</i>	-	Websphere MQ
	-	JMS
<i>Web services (Chapter 9)</i>	Web services	Apache Axis
	Grid services	The Globus Toolkit
<i>Peer-to-peer (Chapter 10)</i>	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Outline

- Middleware
- **Introduction to Interprocess Communication**
- MPI

Middleware Layers



Interprocess Communication

- Shared storage:
 - Shared memory
 - Shared files
- Message passing:
 - Sockets
 - Pipes
 - MPI
 -
- Others
 - Overlay networks
 - Multicasting
 -

Message Passing

- Two operations: *send* and *receive*.
- To communicate:
 - One process send a message (sequence of bytes) to a destination.
 - A process at the destination receives the message.
- Characteristics:
 - Synchronous and asynchronous
 - Destination of a message
 - Reliability
 - Ordering

Synchronous and Asynchronous Communication

- Messages are received in a queue at the destination.
- Synchronous: the sending and receiving processes synchronize at every message.
 - Both send and receive are blocking.
 - A sending process (thread) blocks until the message is received.
 - A receiving process (thread) blocks until a message arrive.
- Asynchronous:
 - The sending operation is non-blocking. Sender proceeds while the message is being transmitted.
 - The receiving process (thread) can be either blocking or non-blocking.

Message Destinations

- Messages are sent to (*Internet address, local port*) *pairs*.
- A port has exactly one receiver but can have many senders.
 - Multicast ports are exception.
- Fixed location: client uses a fixed Internet address to refer to a service, then the service has to always run on the same computer.
- Location transparency: Client programs refer to services by name and use a name server to translate their names into server locations at runtime.

Reliability

- Validity: messages are guaranteed to be delivered despite a 'reasonable' number of packets being dropped or lost.
- Integrity: messages must arrive uncorrupted and without duplication.

Ordering

- Some applications require that messages be delivered in sender order.
- These application will consider it as a failure if a sender messages are received out of order.

Outline

- Middleware
- Introduction to Interprocess Communication
- **MPI**

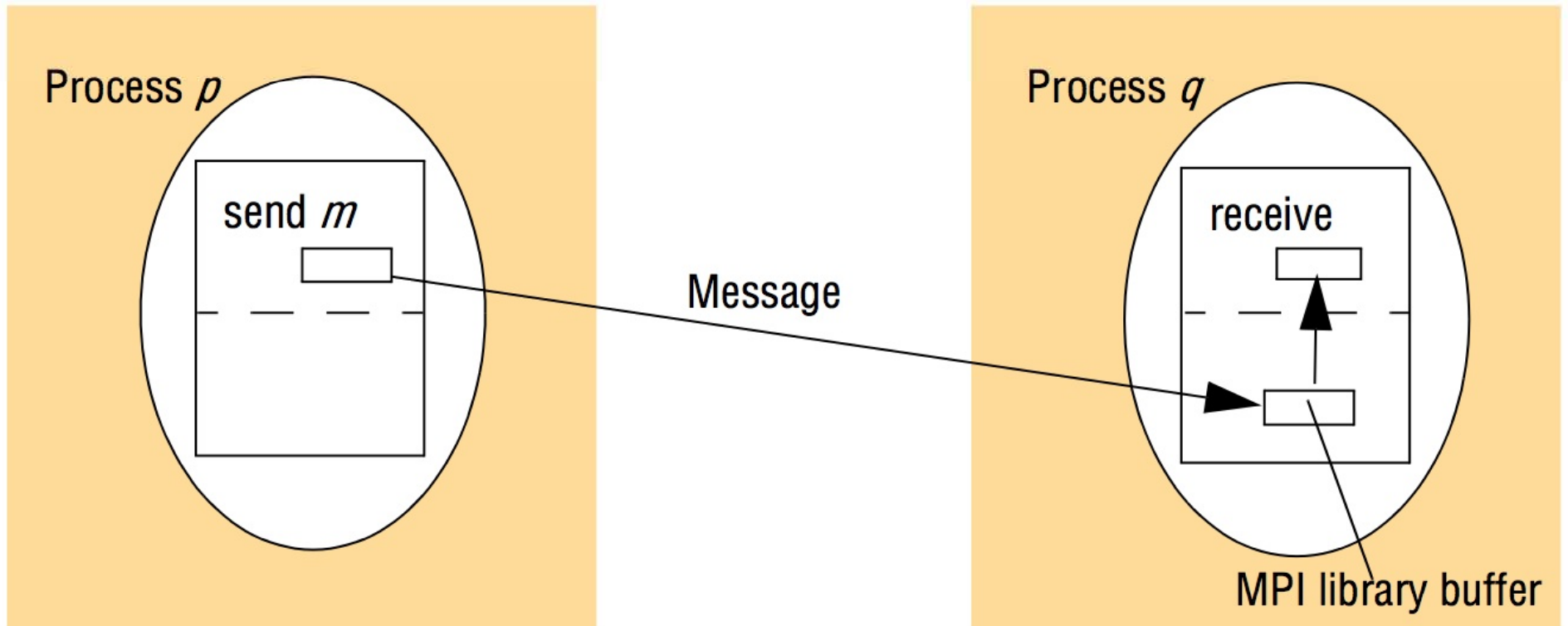
Message Passing Interface (MPI)

- Used when performance is paramount, for instance in high performance computing (HPC).
- Objective: portability through presenting a standardized interface independent of the operating system or programming language-specific socket interface.
- MPI is flexible.
- Interface is available as a message-passing library available for a variety of operating systems and programming languages, including C++ and Fortran.

MPI vs. Sockets

Sockets	MPI
Support only simple send and receive primitives	Provide more variations of send and receive operations that handle advanced features such as buffering and synchronization
Designed for TCP/IP	Suitable for other protocols that are typically used for HPC clusters. Example: infiniband.

Point-to-Point Communication in MPI



- An MPI library buffer in both the sender and the receiver is used to hold data in transit.

Blocking in MPI

- Blocking = ‘blocked until it is safe to return’
 - application data has been copied into the MPI system and hence is in transit or delivered
 - application buffer can be reused (for example, for the next *send* operation).
- Various interpretation of ‘safe to return’ are used:
 - *MPI_Ssend* is the synchronous blocking send. Safety is interpreted as delivered.
 - *MPI_Bsend* interprets safety as allocating and copying data to the library buffer.
 - *MPI_Bsend* interprets safety as the receiver is ready to accept the message and hence can be removed from library buffer.

Send Operations in MPI

<i>Send operations</i>	<i>Blocking</i>	<i>Non-blocking</i>
<i>Generic</i>	<i>MPI_Send</i> : the sender blocks until it is safe to return – that is, until the message is in transit or delivered and the sender's application buffer can therefore be reused.	<i>MPI_Isend</i> : the call returns immediately and the programmer is given a communication request handle, which can then be used to check the progress of the call via <i>MPI_Wait</i> or <i>MPI_Test</i> .
<i>Synchronous</i>	<i>MPI_Ssend</i> : the sender and receiver synchronize and the call only returns when the message has been delivered at the receiving end.	<i>MPI_Issend</i> : as with <i>MPI_Isend</i> , but with <i>MPI_Wait</i> and <i>MPI_Test</i> indicating whether the message has been delivered at the receive end.
<i>Buffered</i>	<i>MPI_Bsend</i> : the sender explicitly allocates an MPI buffer library (using a separate <i>MPI_Buffer_attach</i> call) and the call returns when the data is successfully copied into this buffer.	<i>MPI_Ibsend</i> : as with <i>MPI_Isend</i> but with <i>MPI_Wait</i> and <i>MPI_Test</i> indicating whether the message has been copied into the sender's MPI buffer and hence is in transit.
<i>Ready</i>	<i>MPI_Rsend</i> : the call returns when the sender's application buffer can be reused (as with <i>MPI_Send</i>), but the programmer is also indicating to the library that the receiver is ready to receive the message, resulting in potential optimization of the underlying implementation.	<i>MPI_Irsend</i> : the effect is as with <i>MPI_Isend</i> , but as with <i>MPI_Rsend</i> , the programmer is indicating to the underlying implementation that the receiver is guaranteed to be ready to receive (resulting in the same optimizations),

Thank You