Distributed Systems (5DV147)

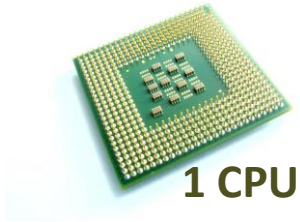
Fundamentals

Fall 2013

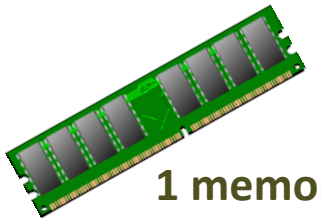
…basics

Single process

```
int i;
i=i+1;
   …
```



1 CPU



1 memory

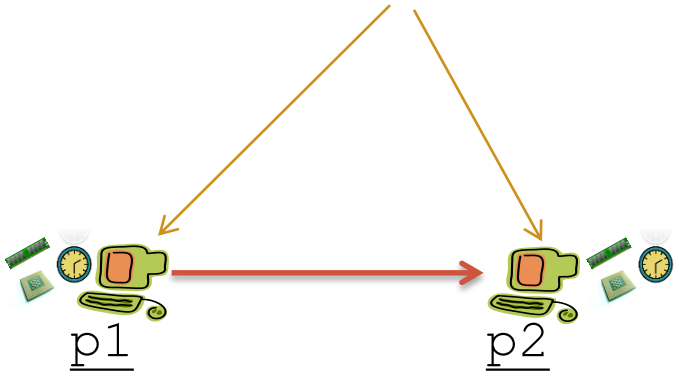
- Steps are strictly sequential
- Program behavior & variables' state determined by sequence of operations

Distributed processes

"A distributed system is one in which components located at networked computers **communicate and coordinate their actions only by passing messages**."

Coulouris, Dollimore, Kindberg, and Blair, 2012

State is private and hidden




```
int i;          int a;
i=i+1;          …
send(i,p2);     receive(p1,a);
…               …
```

Definition of steps of each process including transmission of messages between processes (computation, send, receive)

Can't predict the rate at which each process executes nor the timing of transmissions

Progress depends on internal local state and of messages received

Impossible to describe all states of a distributed algorithm because of failure of any of the processes or of message transmissions

Models

Why models?

- to abstract underlying properties that are common to a large range of systems so that it enables to distinguish the fundamental from the accessory
- to make explicit all relevant assumptions about the system
- to discover common design problems
- to prevent reinvent the wheel for every minor variant of the problem

A model abstracts away the key components and the way they interact

Message-passing model

Characteristics:

- Synchrony of the system
- Type of communication network
  - point-to-point or broadcast channel
- Model of process and communication failures that may occur

Synchronous systems

- There is a known upper bound on the time required by any process to execute a step
- Every process has a clock with known bound rate of drift with respect to real time
- There is a known bound on message delay – time to send, transport, and receive a message over a link

(Hadzilacos and Toueg, 1994)

Asynchronous systems

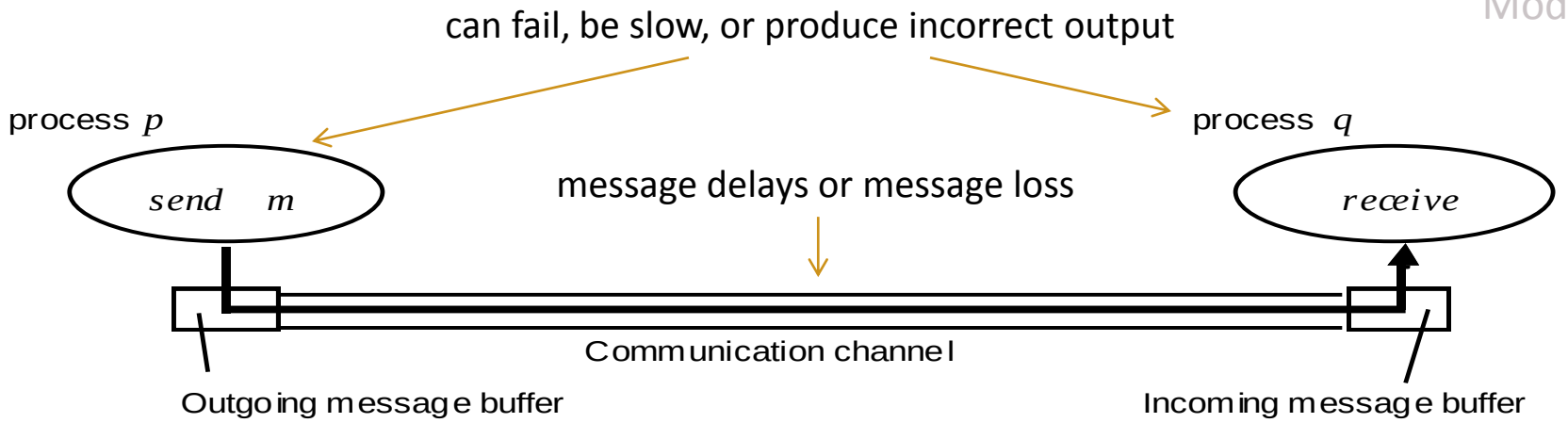
No timing assumptions, in particular on the maximum message delay, clock drift, or the time needed to execute a step

(Hadzilacos and Toueg, 1994)

Attractive because of:

- Simple semantics
- Applications are easier to port
- Variable or unexpected workloads are sources of asynchrony
- Processes share resources and communications channels share the network

Point-to-point networks



Two assumptions:

1. If p sends a message m to q then q eventually receives m
2. Every process executes an infinite sequence of steps

Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 2.14**

Failure model

“A distributed system is one in which the **failure of computer you didn't even know existed** can render your own computer unusable.”

Leslie Lamport, 1987

Why computer systems fail?

- Hardware reliability
  - Dominant until late in the 1980s
  - Minor component of overall reliability concerns
  - Not so for mechanical devices, e.g., networks or disks
- Software reliability
  - Software bugs account for a substantial fraction of unplanned downtime (25-35%)
    - Bohrbugs – easily localized and reproducible
    - Heisenbug – symptoms of other bugs that don't cause an immediate crash (extremely hard to fix, also known as transient bugs)
- Planned maintenance and environmental factors

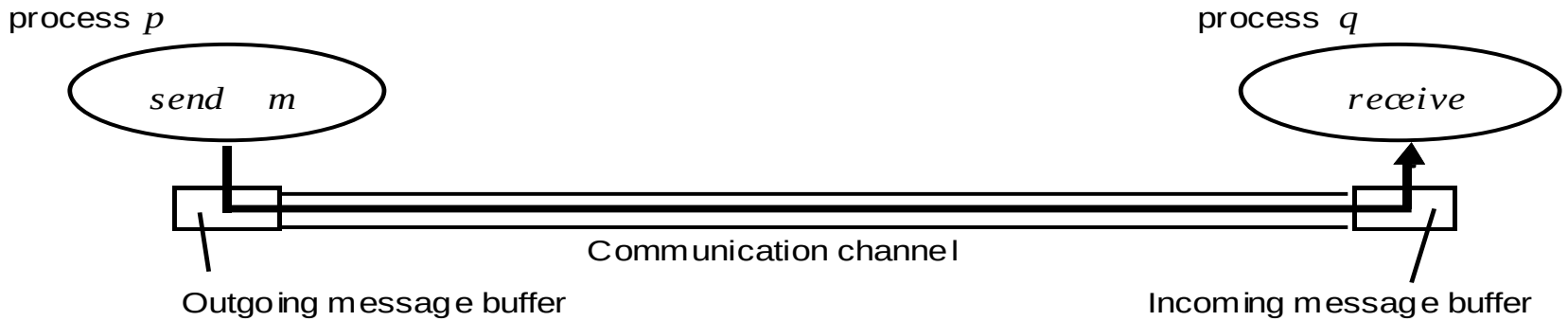
Processes and communication channels may fail, i.e., deviation from correct behavior.

The failure model defines ways in which failures may occur in order to understand the effects of failures.

Omission failures

Process or communication channel fails to do what it is supposed to do.

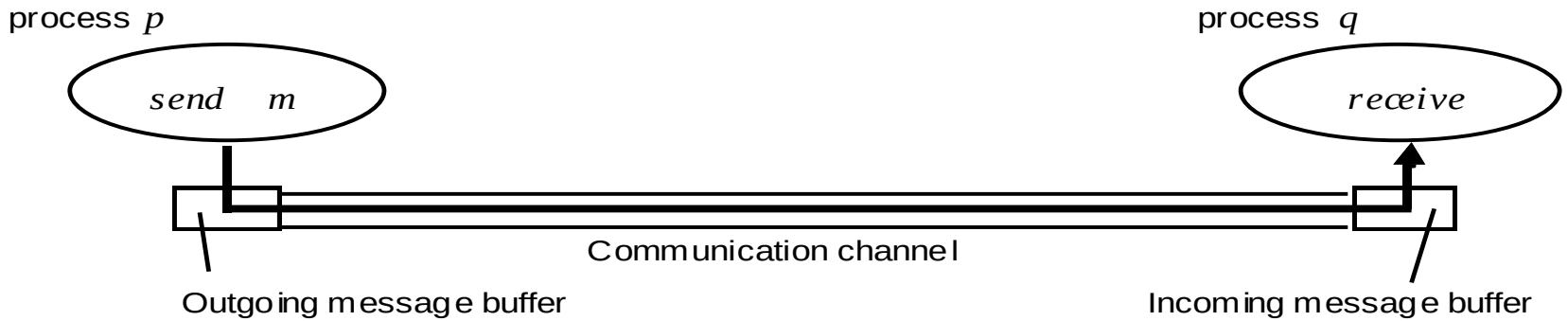
Process Omission failures:



1. Crash
2. Fail-stop
3. Send-omission
4. Receive-omission

Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 2.14**

Channel Omission failures:



m is inserted into $p's$ outgoing message buffer but m is not transported into $q's$ incoming message buffer

Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 2.14**

Timing failures

(only for synchronous systems)

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Benign failures

- Omission failures (asynchronous networks)
- Timing failures (synchronous networks)

Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 2.11**

Arbitrary failures

(Byzantine or malicious failures)

- Any type of error may occur
- If the sequence of steps executed by a process deviates arbitrarily from what it should do, e.g., change its state arbitrarily or send a message that it was not supposed to send
- In communication channels messages may be corrupted, duplicated, or non-existent messages be delivered

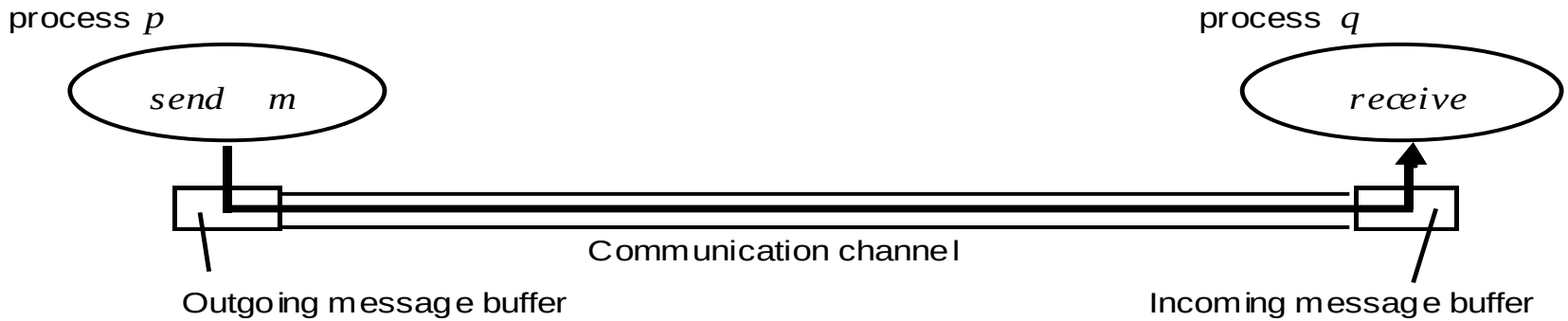
Why do we want to know the failure characteristics of a component?

- We can mask the failure of the components on which it depends by either hiding it or converting it into a more acceptable type of failure

Failure masking by Redundancy

- Information redundancy
- Time redundancy
- Physical redundancy

Reliability of one-to-one communication:

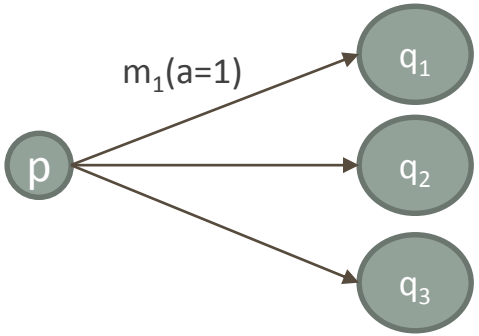


Validity: any message in the outgoing buffer is eventually delivered to the incoming message buffer

Integrity: The message received is identical to the one sent, and no messages are delivered twice

Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 2.14**

Reliability of broadcast communication:



What happens if p fails in the middle?

Reliable broadcast: all ... agree ... messages delivered ... failure

Consensus: all processes ... common decision depending on initi... despite failures

applied research

theoretical research

How would you design a reliable broadcast protocol?

End-to-end reliability or hop-by-hop?

End-to-end: only between source process and target process of a message

Hop-by-hop: reliability at every intermediate node

Hop-by-hop reliability increases complexity and reduces performance, and end-to-end reliability is still required.

End-To-End Arguments in System Design
J. H. SALTZER, D. P. REED, and D. D. CLARK
ACM Transactions on Computer Systems,
Vol. 2, No. 4, November 1984, Pages 277-288

http://groups.csail.mit.edu/ana/Publications/PubPDFs/End-to-End%20Arguments%20in%20System%20Design.pdf

Summary

- Difference between single process and distributed algorithms
- Message-passing model
  - Synchrony of the system
  - Type of communication network
  - Model of process and communication failures that may occur
- Differences between synchronous and asynchronous systems

Failure Model

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Figure adapted rom Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – based on **Figure 2.11 and Figure 2.15**

- Failure masking
  - Failure masking by redundancy
- Reliable point-to-point and broadcast communication
- End-to-end arguments in system design

Communication paradigms

Inter-process communication
&
remote invocation

Inter-process communication

Low level support for communication between processes

message passing
socket programming
multicast communication

Remote invocation

Two-way exchange between communication entities

calling of a remote operation, procedure or method

Request-reply protocols – pairwise exchange of messages from client to server

Remote procedure calls – procedures in processes on remote computers can be called as if they were in the local address space

Remote method invocation – a calling object can invoke a method in a remote object

Communicating entities and communication paradigms

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 2.2**

Next Lecture

Security