Lecture 02 · SENG 41283 · Distributed and Cloud Computing

System Models

Physical Models · Architectural Models · Fundamental Models · Interaction · Failure · Security

1. Why System Models?

Systems must function correctly in the widest possible range of circumstances and threats: widely varying workloads, heterogeneous hardware/OS/networks, internal problems (unsynchronised clocks, conflicting updates, hardware/software failures), and external threats (denial-of-service, data attacks). Three categories of model capture the key properties and design issues.

2. Physical Models

A physical model represents the underlying hardware elements, abstracting away specific technology details. The baseline: an extensible set of computer nodes interconnected by a network for passing messages.

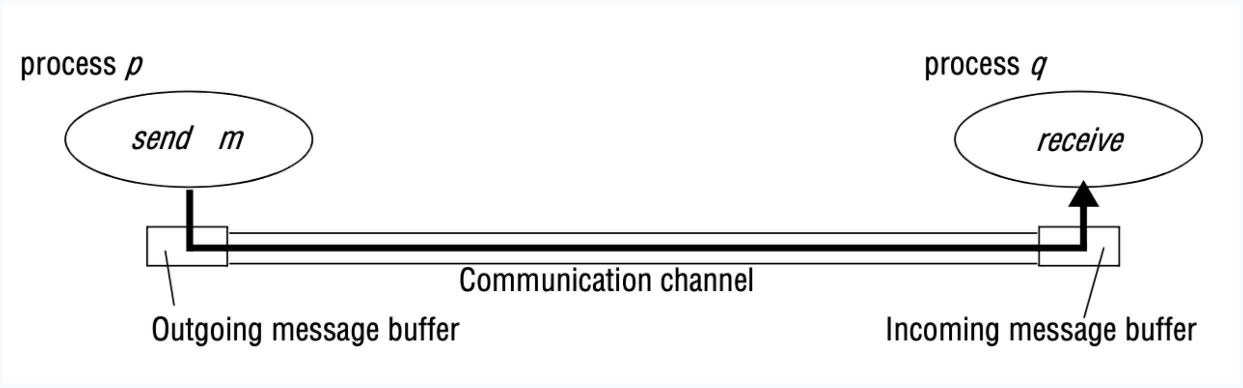


Figure 2.1 Baseline physical model: computer nodes interconnected by a network.

Generation	Period	Description
Early Systems	1970s–80s	Emerged with LAN technology (Ethernet). 10–100 nodes; limited Internet. Services: shared printers, file servers, email, file transfer.
Internet-Scale	1990s+	Dramatic Internet growth (Google 1996). Significant heterogeneity. Emphasis on open standards and middleware.
Contemporary	Present	Mobile computing (service discovery), ubiquitous computing (everyday objects), cloud computing (pools of nodes providing a single service).

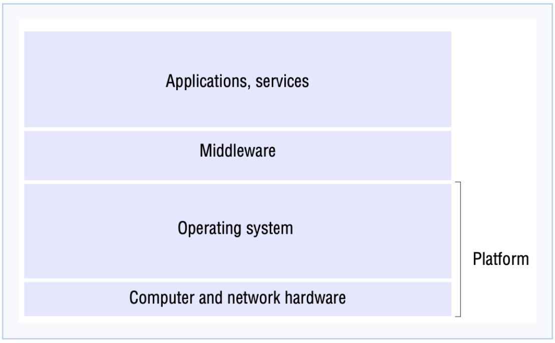


Figure 2.2 Early distributed systems: LAN-based, limited Internet connectivity.

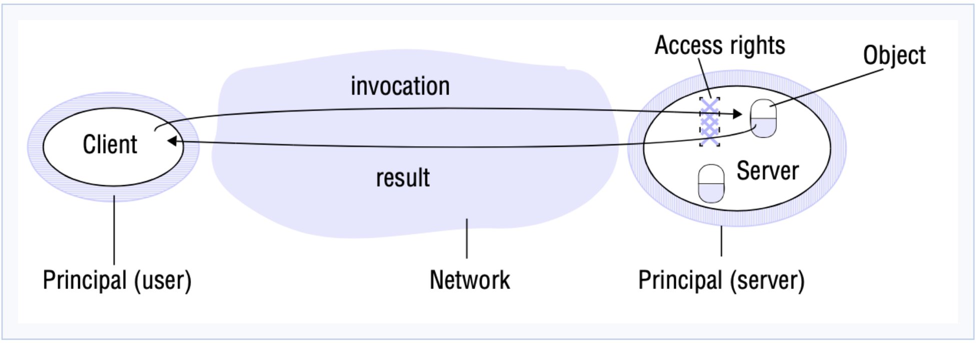


Figure 2.3 Contemporary distributed systems: mobile, ubiquitous, and cloud.

3. Architectural Models

The architecture of a system is its structure in terms of separately specified components and their interrelationships. Four key questions: (1) What entities communicate? (2) How do they communicate? (3) What roles and responsibilities do they have? (4) How are they mapped onto the physical infrastructure?

3.1 Communicating Entities

- **Processes:** the primary communicating entity; a distributed system is processes coupled with IPC paradigms.
- **Objects:** accessed via interfaces defined in IDL; natural units of decomposition.
- **Components:** like objects but also specify required interfaces (dependencies).
- **Web Services:** integrated into the WWW; use HTTP, WSDL, SOAP, and REST.

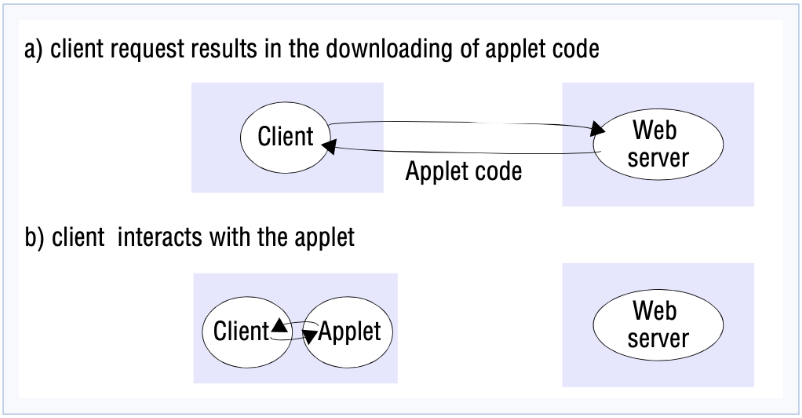

a) client request results in the downloading of applet code
Client
Web server
Applet code
b) client interacts with the applet
Client
Applet
Web server

Figure 3.1 Communicating entities: processes, objects, components, and web services.

3.2 Communication Paradigms

Interprocess Communication (IPC)

Low-level message-passing primitives; direct access to socket API (UDP/TCP); multicast support. IPC is the building block for all higher-level mechanisms.

Remote Invocation

The most common paradigm. Covers request–reply protocols, Remote Procedure Call (Birrell and Nelson, 1984), and Remote Method Invocation.

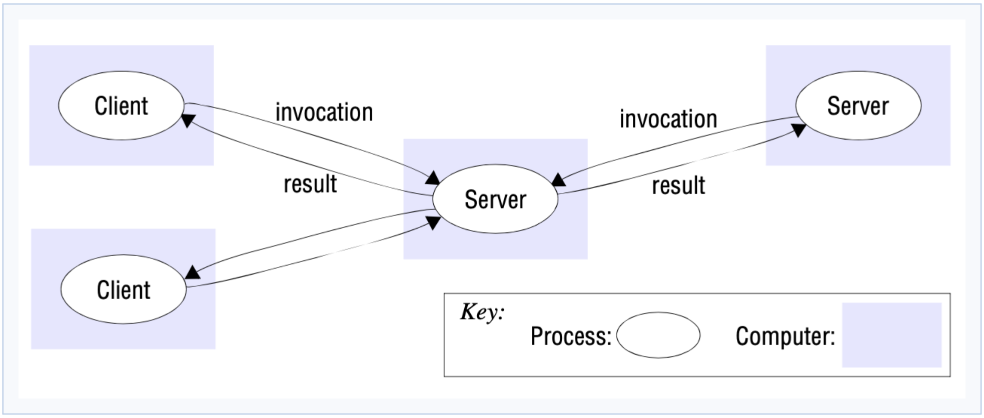


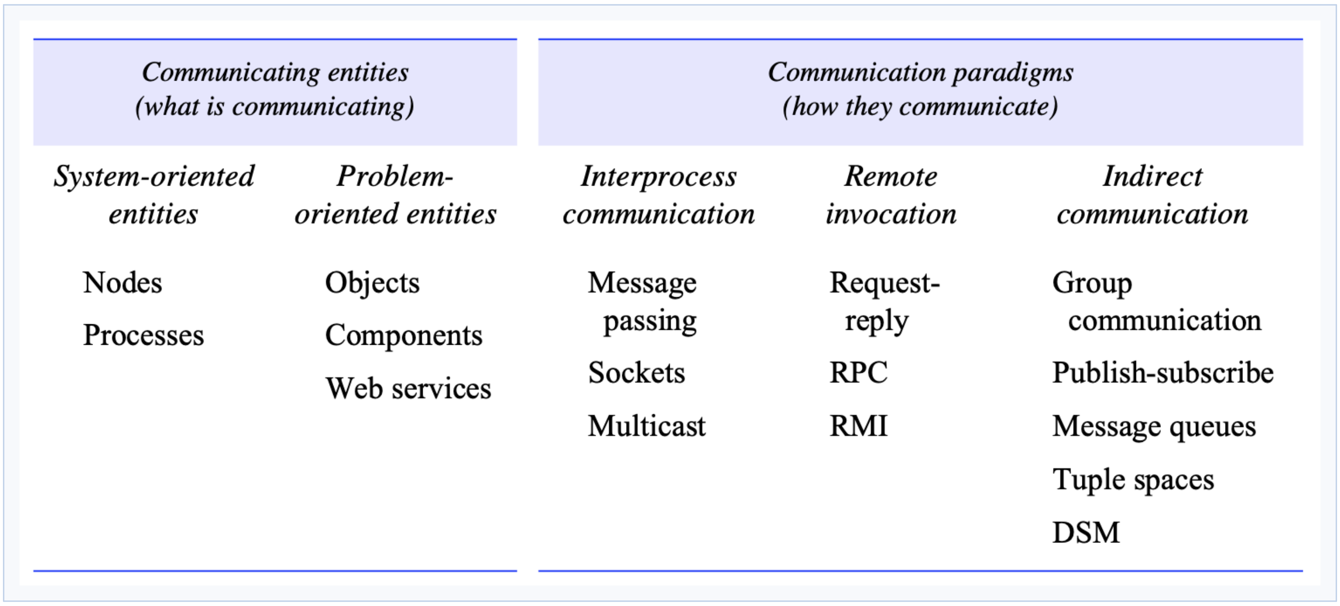
Figure 3.2 Communication paradigms: IPC, remote invocation, and indirect communication.

Indirect Communication

Through a third entity, allowing strong decoupling. Space uncoupling: senders need not know the identity of receivers. Time uncoupling: sender and receiver need not exist simultaneously. Mechanisms: group communication, publish–subscribe, message queues, tuple spaces, DSM.

3.3 Roles and Responsibilities

Client–Server: clients interact with server processes to access shared resources. Servers may themselves be clients of other servers.



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 3.3 Client–server model: clients access shared resources held by servers.

Peer-to-Peer: all processes play similar roles; each holds a small part of the total data and acts as both client and server.

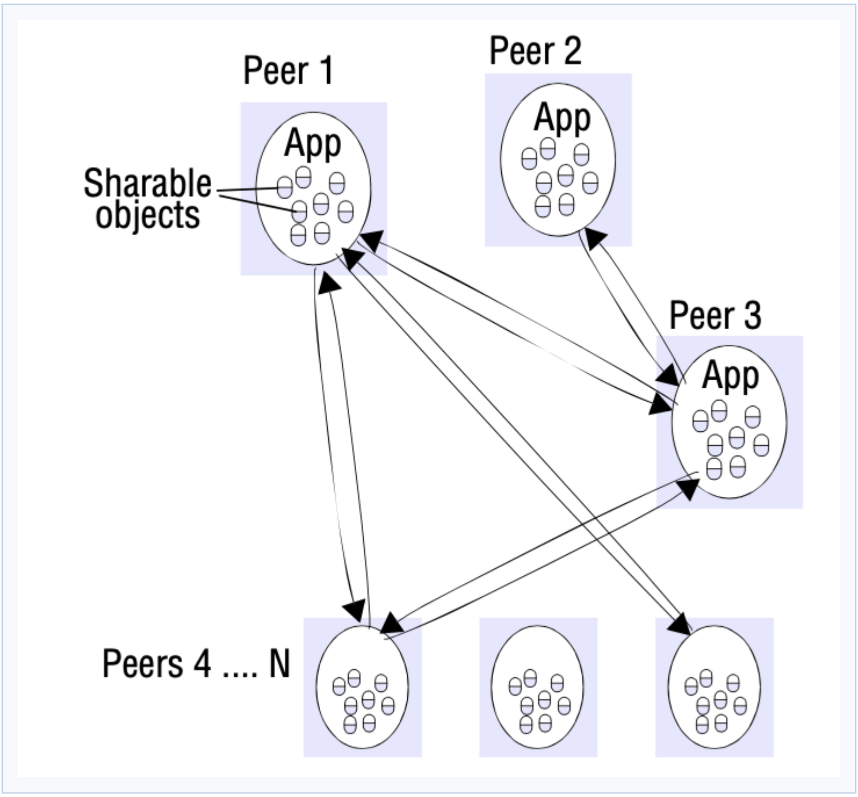


Figure 3.4 Peer-to-peer model: processes play equivalent roles.

3.4 Placement Strategies

- **Multiple servers:** partition or replicate objects across servers.
- **Caching:** store recently used data closer to clients (web browser cache, proxy servers).

- **Mobile code:** applets downloaded from server and executed at client.
- **Mobile agents:** running programs that travel between computers to collect information.

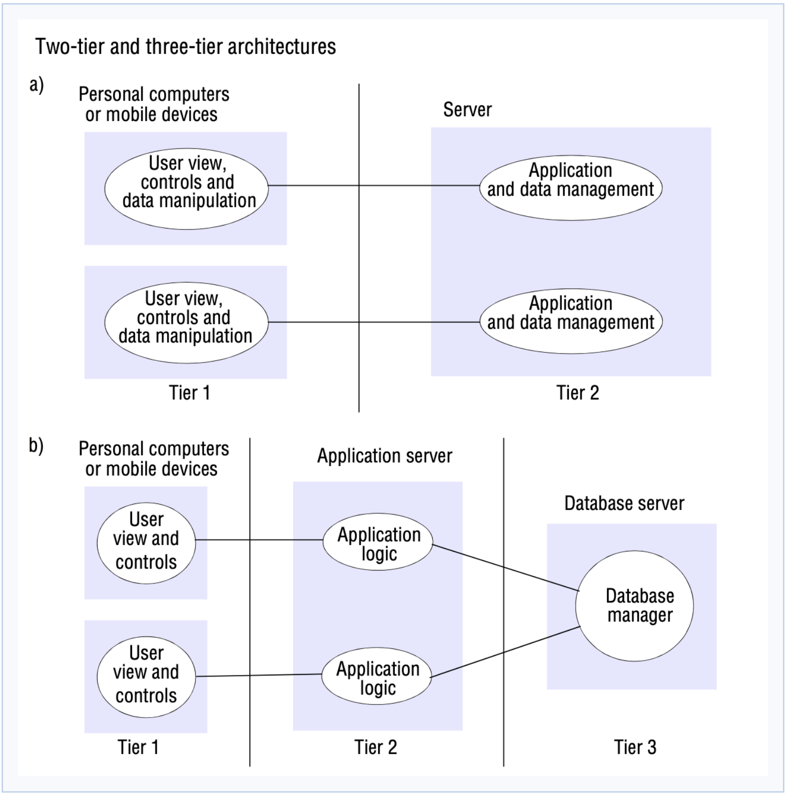
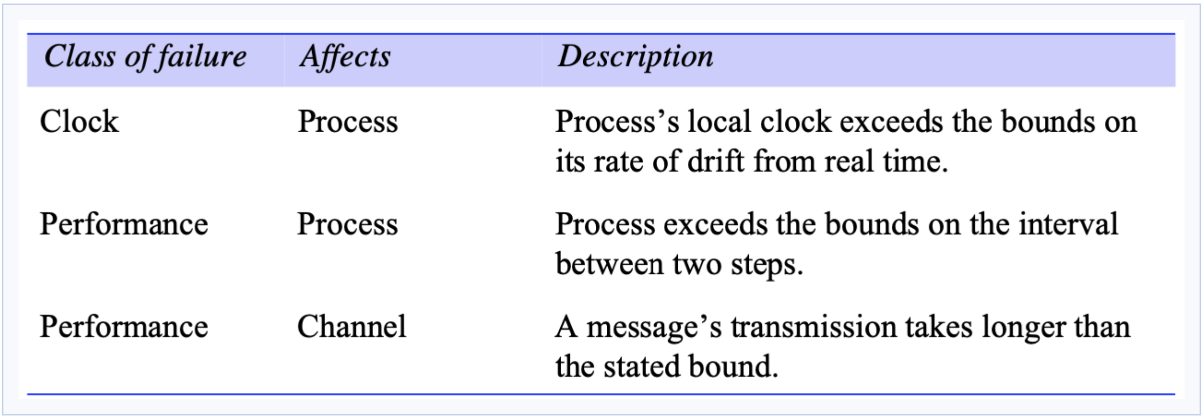

Two-tier and three-tier architectures
a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

Figure 3.5 Layering and tiered architectural patterns.

4. Fundamental Models

4.1 Interaction Model

Communication performance is characterised by latency (delay from start of transmission to beginning of receipt), bandwidth (total information transmissible in a given time), and jitter (variation in delivery time, critical for multimedia). Synchronous systems have known bounds on execution steps, message delays, and clock drift. Asynchronous systems (the Internet model) have no such bounds.

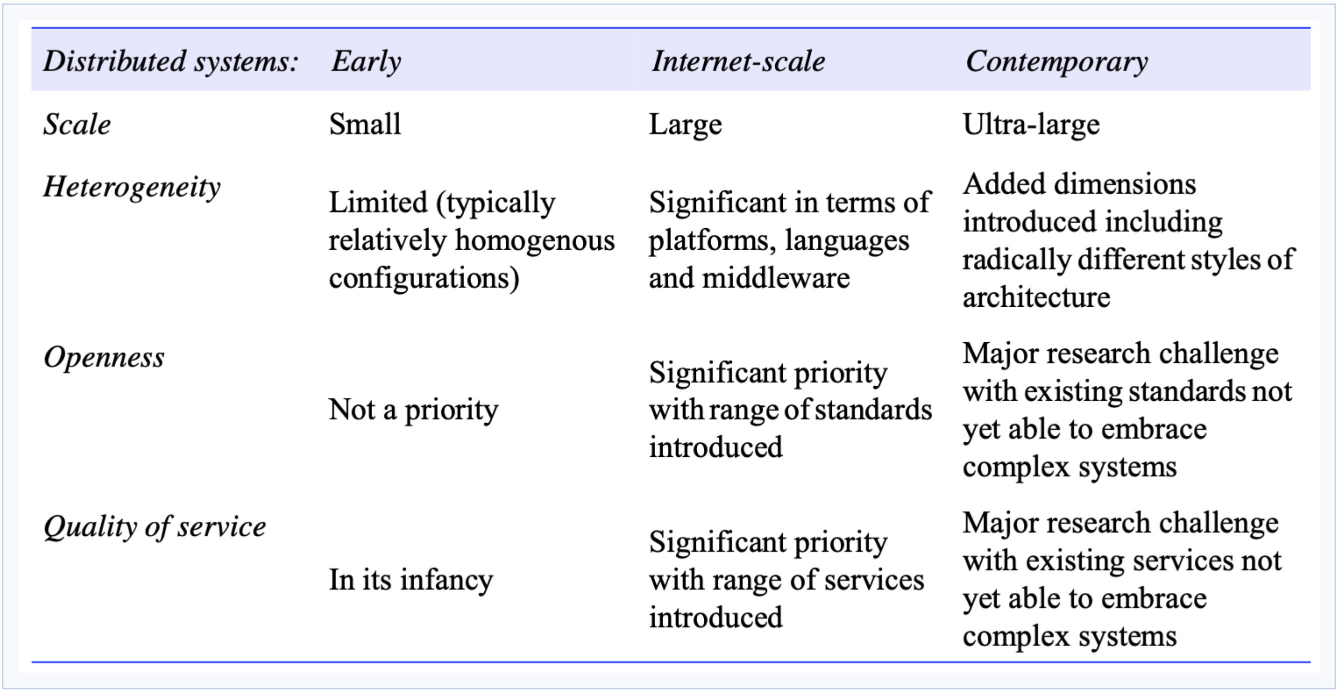


Class of failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Figure 4.1 Interaction model: performance characteristics of network communication.

4.2 Failure Model

- **Process omission (crash):** halts and executes no further steps. Fail-stop: detectable by other processes.
- **Communication omission:** message not transported from sender’s outgoing to receiver’s incoming buffer (buffer overflow or checksum error).
- **Arbitrary (Byzantine):** any type of error; process may return wrong values; undetectable by checking responses.
- **Timing failures:** response falls outside the specified time window (synchronous systems only).
- **Masking:** build reliable services from failing components; checksums convert arbitrary → omission failures.

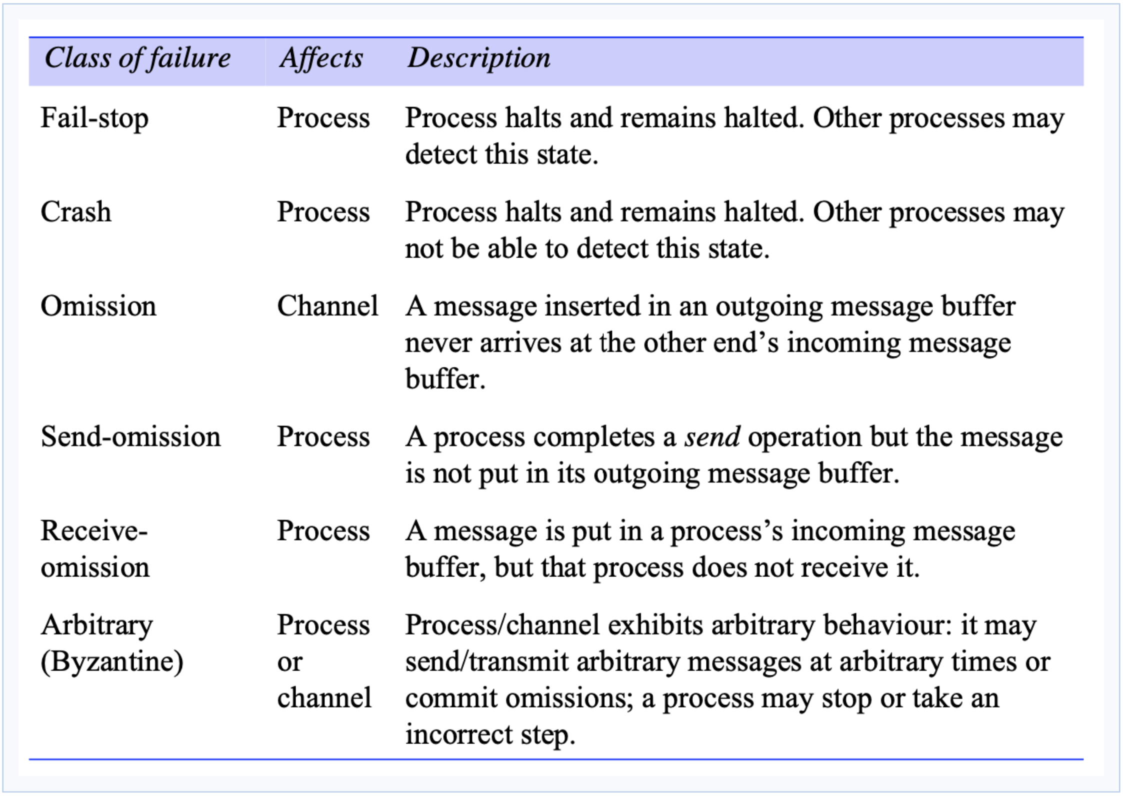


Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Figure 4.2 Failure model: taxonomy of process and communication failures.

4.3 Security Model

Access rights specify who is allowed to perform operations on an object. A principal (user or process) must be authenticated and checked against access rights. Defeating threats requires: cryptography and shared secrets, authentication, and secure channels (known identities, privacy, integrity, timestamps to prevent replay).



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* operation 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 or commit omissions; a process may stop or take an incorrect step.

Figure 4.3 Security model: protecting objects and securing process interactions.