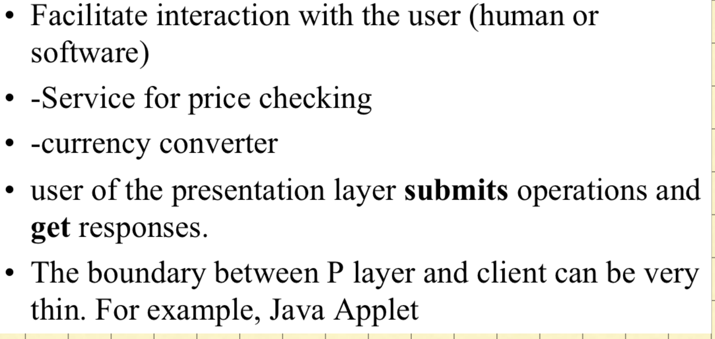
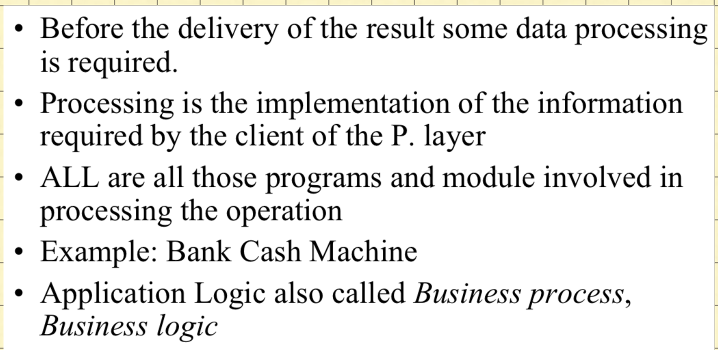
Presentation Layer



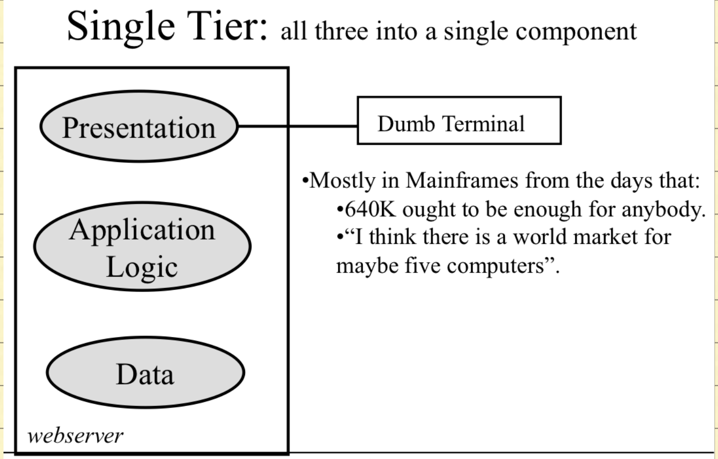
- Facilitate interaction with the user (human or software)
- -Service for price checking
- -currency converter
- user of the presentation layer **submits** operations and **get** responses.
- The boundary between P layer and client can be very thin. For example, Java Applet

Application Logic Layer (ALL)



- Before the delivery of the result some data processing is required.
- Processing is the implementation of the information required by the client of the P. layer
- ALL are all those programs and module involved in processing the operation
- Example: Bank Cash Machine
- Application Logic also called *Business process*, *Business logic*

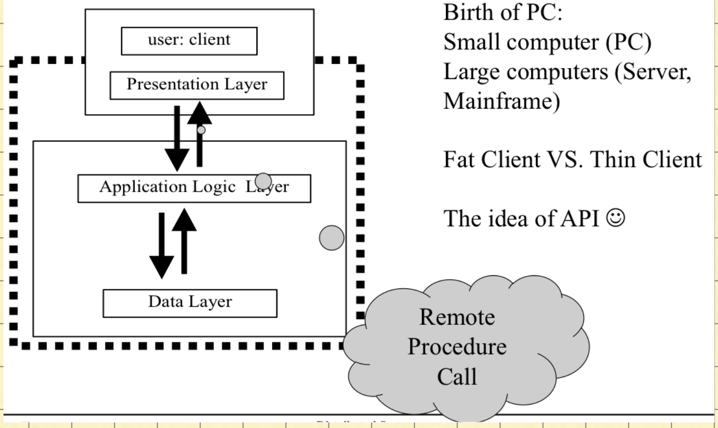
Single Tier



Single Tier: all three into a single component

- Mostly in Mainframes from the days that:
  - 640K ought to be enough for anybody.
  - "I think there is a world market for maybe five computers".

Two Tier

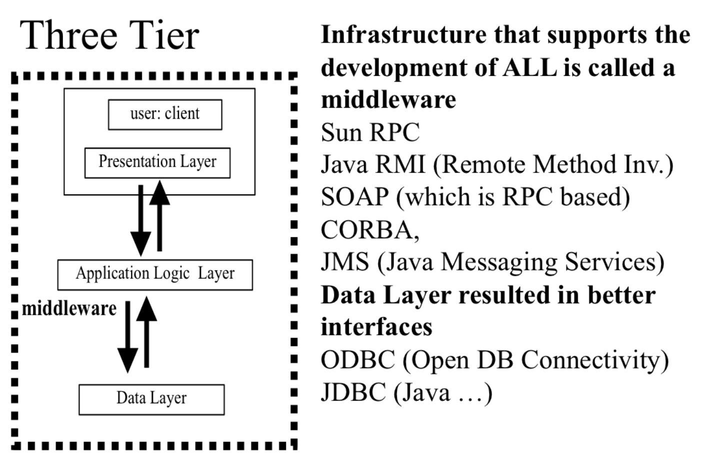


Birth of PC:
Small computer (PC)
Large computers (Server, Mainframe)

Fat Client VS. Thin Client

The idea of API ☺

Three Tier



user: client
Presentation Layer
Application Logic Layer
middleware
Data Layer

Three Tier

Infrastructure that supports the development of ALL is called a middleware

Sun RPC

Java RMI (Remote Method Inv.)

SOAP (which is RPC based)

CORBA,

JMS (Java Messaging Services)

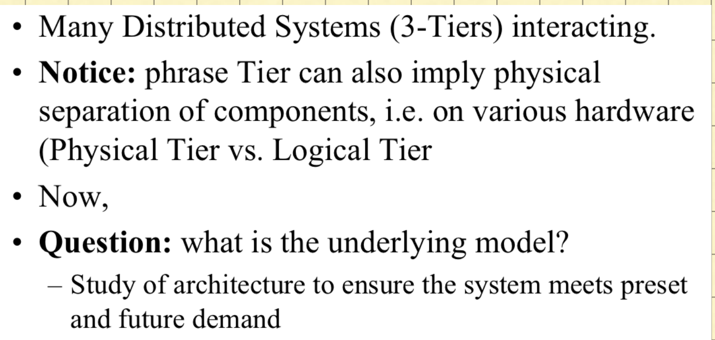
Data Layer resulted in better interfaces

ODBC (Open DB Connectivity)

JDBC (Java …)

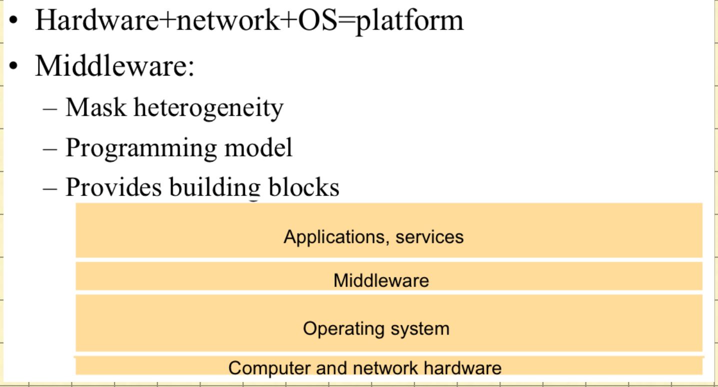
Application Logic Layer 中的软件称为中间件

N Tier



- Many Distributed Systems (3-Tiers) interacting.
- **Notice:** phrase Tier can also imply physical separation of components, i.e. on various hardware (Physical Tier vs. Logical Tier
- Now,
- **Question:** what is the underlying model?
  - Study of architecture to ensure the system meets preset and future demand

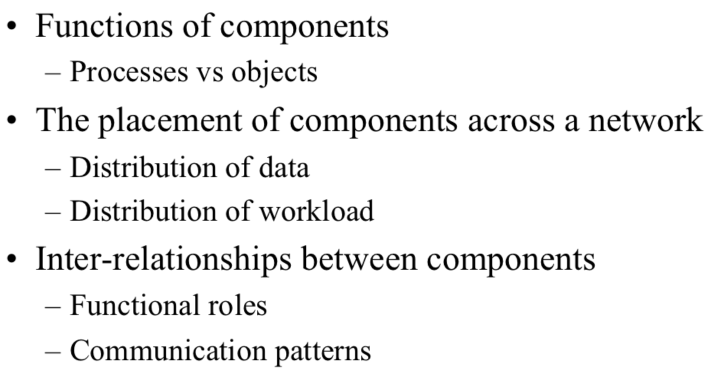
Service Layers in a Distributed System



- Hardware+network+OS=platform
- Middleware:
  - Mask heterogeneity
  - Programming model
  - Provides building blocks

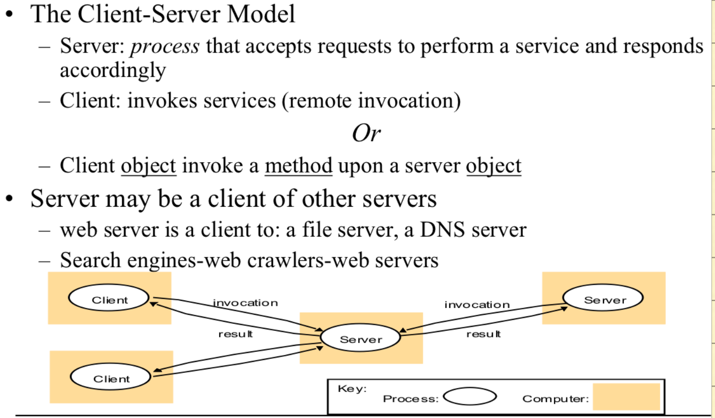
Applications, services
Middleware
Operating system
Computer and network hardware

Architectural Models of Distributed Systems



- Functions of components
  - Processes vs objects
- The placement of components across a network
  - Distribution of data
  - Distribution of workload
- Inter-relationships between components
  - Functional roles
  - Communication patterns

System Architectures

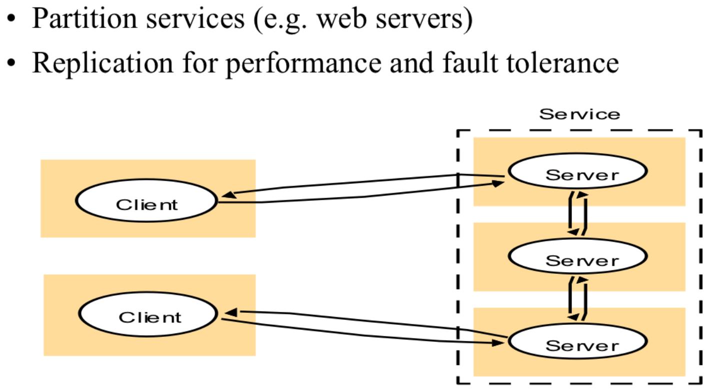


- The Client-Server Model
  - Server: *process* that accepts requests to perform a service and responds accordingly
  - Client: invokes services (remote invocation)

Or

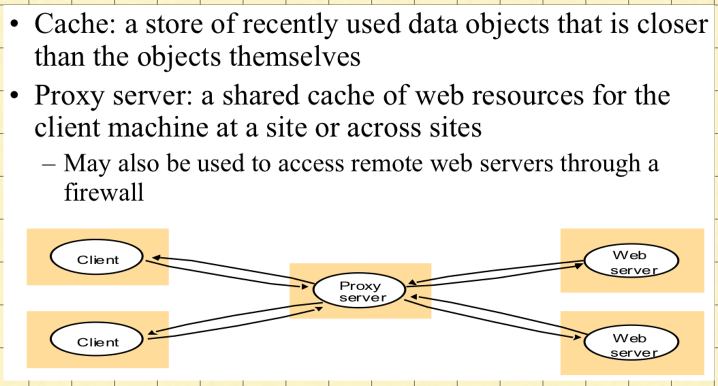
  - Client object invoke a method upon a server object
- Server may be a client of other servers
  - web server is a client to: a file server, a DNS server
  - Search engines-web crawlers-web servers

Multiple Servers



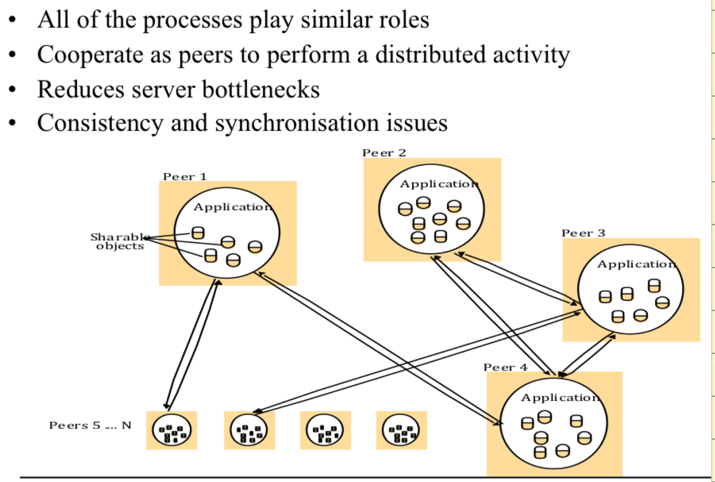
- Partition services (e.g. web servers)
- Replication for performance and fault tolerance

Proxy Servers and Caches



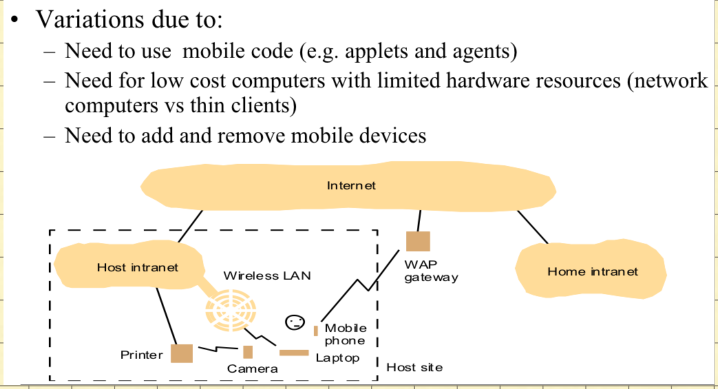
- Cache: a store of recently used data objects that is closer than the objects themselves
- Proxy server: a shared cache of web resources for the client machine at a site or across sites
  - May also be used to access remote web servers through a firewall

Peer processes



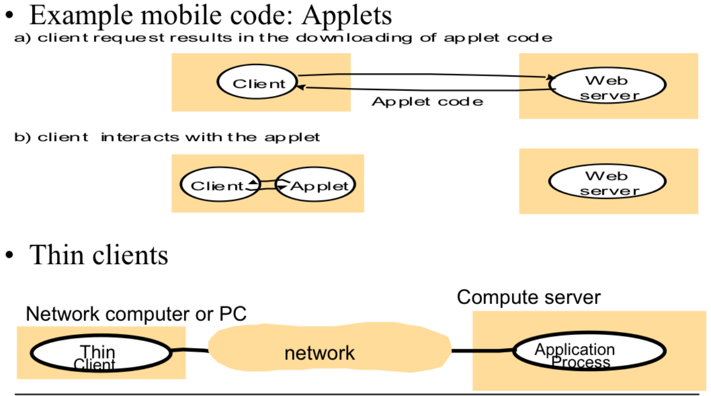
- All of the processes play similar roles
- Cooperate as peers to perform a distributed activity
- Reduces server bottlenecks
- Consistency and synchronisation issues

Variations of the client-server model



- Variations due to:
  - Need to use mobile code (e.g. applets and agents)
  - Need for low cost computers with limited hardware resources (network computers vs thin clients)
  - Need to add and remove mobile devices

Variations

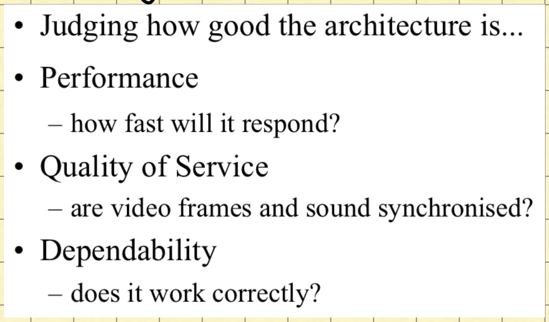


- Example mobile code: Applets

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

- Thin clients

Design Requirements for DSs



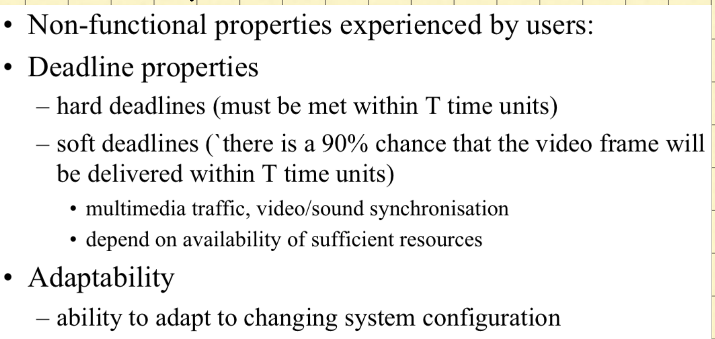
- Judging how good the architecture is...
- Performance
  - how fast will it respond?
- Quality of Service
  - are video frames and sound synchronised?
- Dependability
  - does it work correctly?

性能

QoS 服务质量

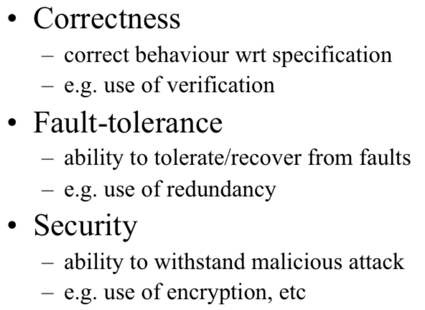
可靠性

Quality of Service (QoS)



- Non-functional properties experienced by users:
- Deadline properties
  - hard deadlines (must be met within T time units)
  - soft deadlines (`there is a 90% chance that the video frame will be delivered within T time units)
    - multimedia traffic, video/sound synchronisation
    - depend on availability of sufficient resources
- Adaptability
  - ability to adapt to changing system configuration

Dependability



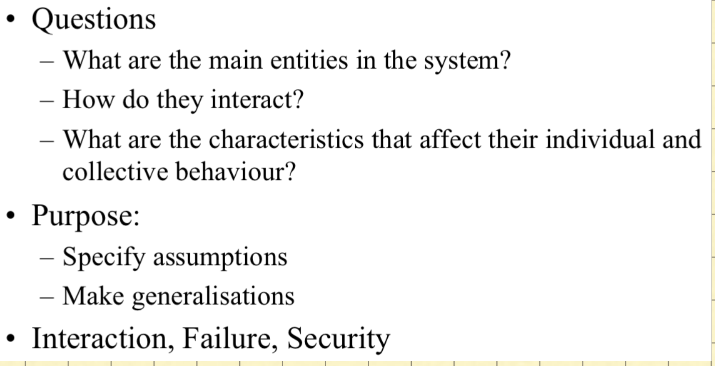
- Correctness
  - correct behaviour wrt specification
  - e.g. use of verification
- Fault-tolerance
  - ability to tolerate/recover from faults
  - e.g. use of redundancy
- Security
  - ability to withstand malicious attack
  - e.g. use of encryption, etc

正确性

错误容忍

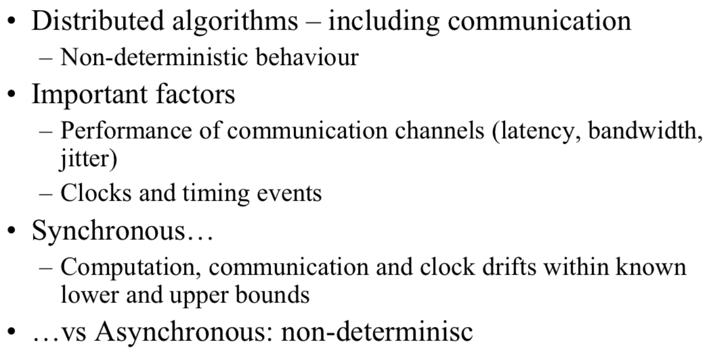
安全

Fundamental Models



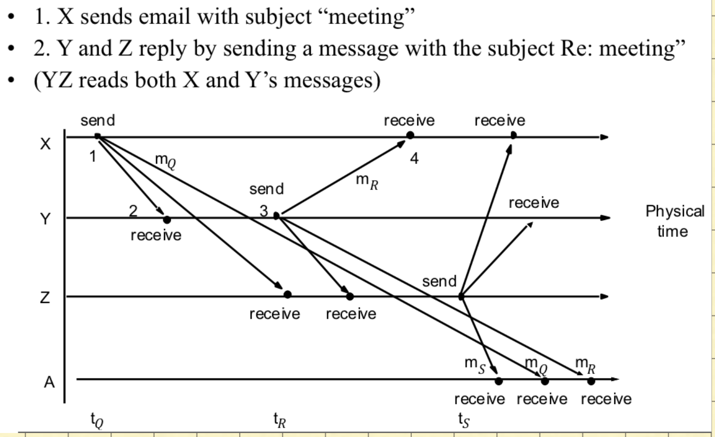
- Questions
  - What are the main entities in the system?
  - How do they interact?
  - What are the characteristics that affect their individual and collective behaviour?
- Purpose:
  - Specify assumptions
  - Make generalisations
- Interaction, Failure, Security

Interaction Model



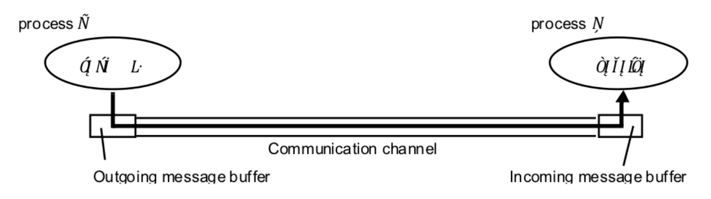
- Distributed algorithms – including communication
  - Non-deterministic behaviour
- Important factors
  - Performance of communication channels (latency, bandwidth, jitter)
  - Clocks and timing events
- Synchronous…
  - Computation, communication and clock drifts within known lower and upper bounds
- …vs Asynchronous: non-determinisc

Event Ordering

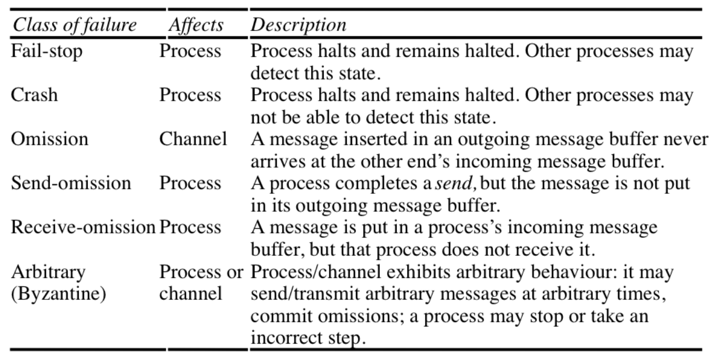


- 1. X sends email with subject "meeting"
- 2. Y and Z reply by sending a message with the subject Re: meeting"
- (YZ reads both X and Y's messages)

Failures

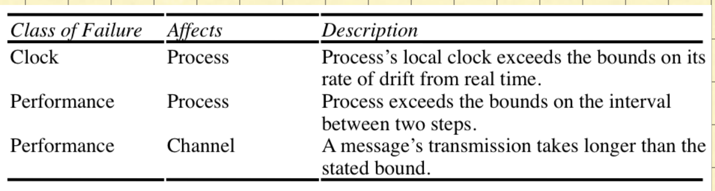


Omission and arbitrary failures



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.

Timing failures



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.