Ciências ULisboa

Programação em Sistemas Distribuídos

MEI-MI-MSI

2018/19

1. Review of Fundamental Distributed Systems Concepts

Prof. António Casimiro

What is a distributed system?

- A computer network ***is not*** a distributed system
- Computer network
  - infrastructure serving a set of computers interconnected through communication links of possibly diverse media and topology, and using a common set of communication protocols
- Distributed system
  - system composed of several computers which communicate through a computer network, hosting processes that use a common set of distributed protocols to assist the coherent execution of distributed activities

Characteristics of Distributed Systems

- multiple computers
- interconnected by a network
- sharing state

→

- independent failures
- communication is unreliable
- has variable delays
- speed and bandwidth are moderate
- investment costs often lower than mainframes
- management costs are higher
- partial ordering of events only
- difficult to assess global state

Centralized versus Distributed Systems

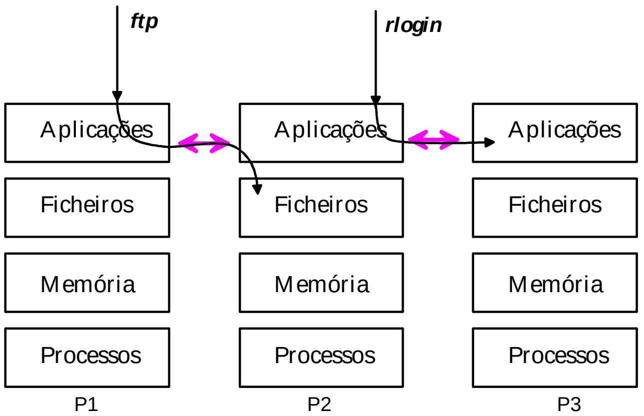
- Centralized :
  - Accessibility
    - to resources and info
  - Homogeneity
    - of procedures and technologies
  - Manageability
    - single, domain
  - Consistency
    - global state
  - Security, due to:
    - threat reduction
- Distributed :
  - Scalability, helped by:
    - geographical scope
    - heterogeneity
    - modularity
  - Sharing:
    - of common resources and info
  - Reliability and availability, due to:
    - redundancy and replication
    - graceful degradation
    - failure independence
  - Security, due to:
    - vulnerability reduction
  - Low cost factor

Distributed Systems Evolution

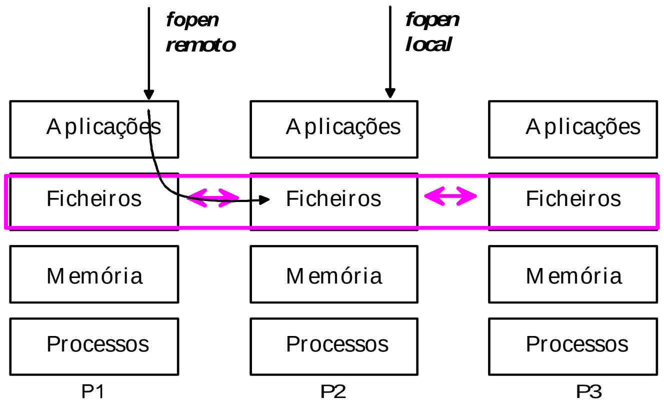
TECHNIQUES AND ARCHITECTURES

Distributed Systems Evolution
Techniques

- Resource sharing:
  - Remote login (rlogin)
  - File transfer

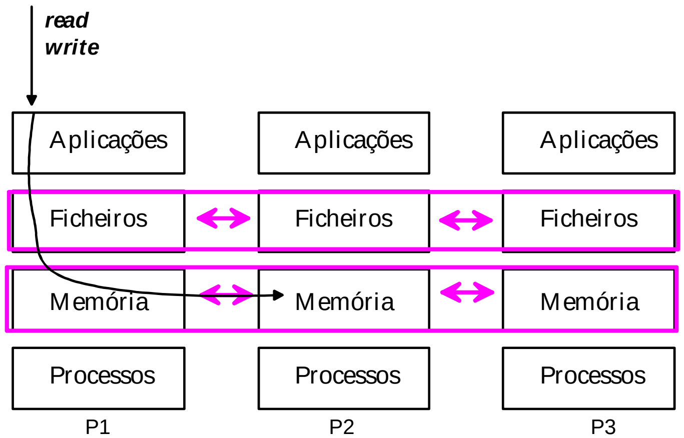


- Distributed file system:
  - Avoiding explicit file transfer
  - Efficient information sharing

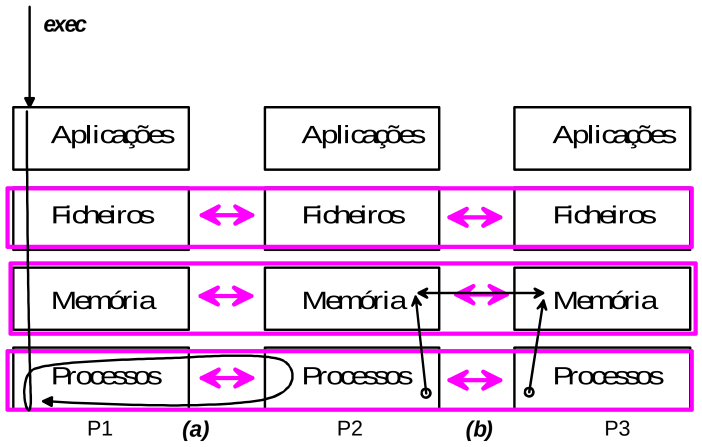


Distributed Systems Evolution
Techniques

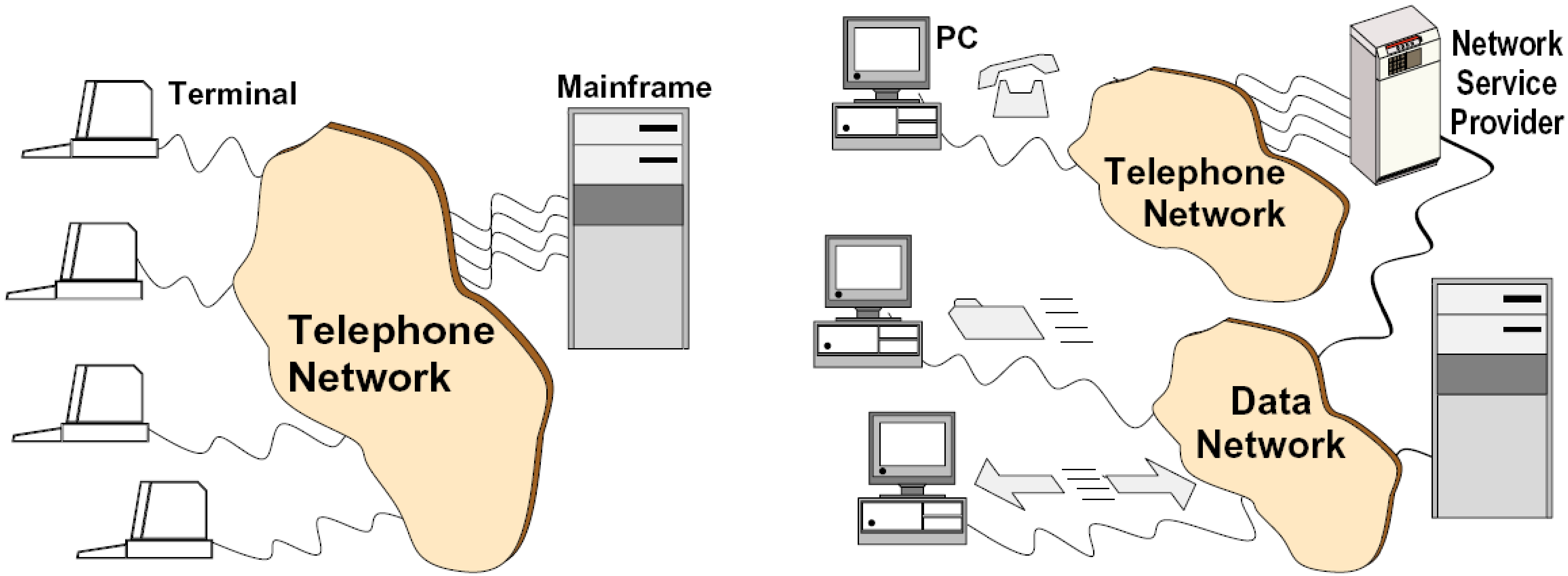
- Distributed memory:
  - Exploit network resources
  - Support (distributed) shared memory
  - Step towards distributed processing



- Distributed processing:
  - (a) Activities executed in several machines
  - (b) Concurrent activities synchronized over shared memory

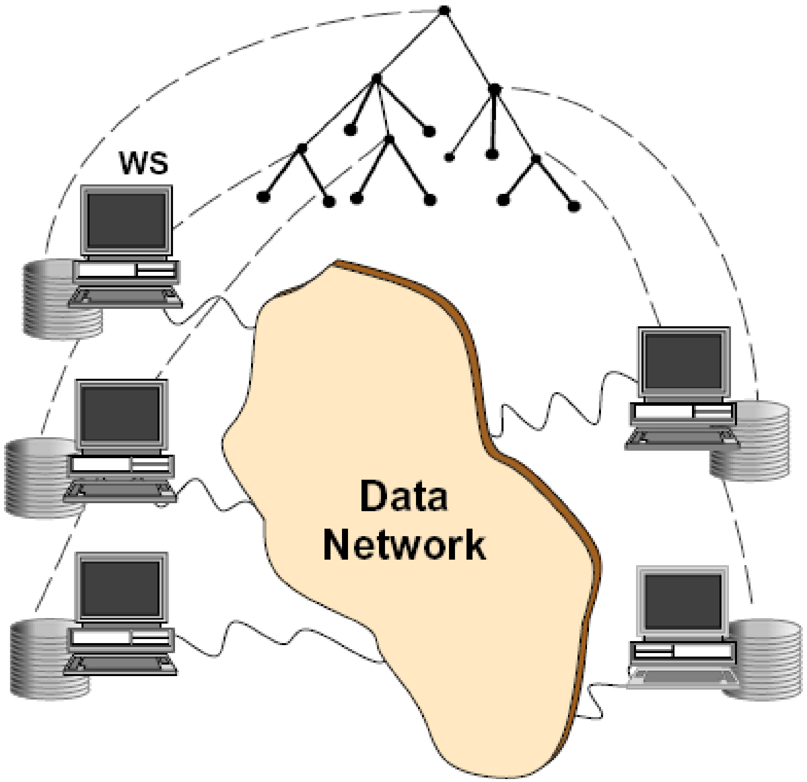


Remote Access Architectures:
(a) Plain Telephone Line; (b) Data Network

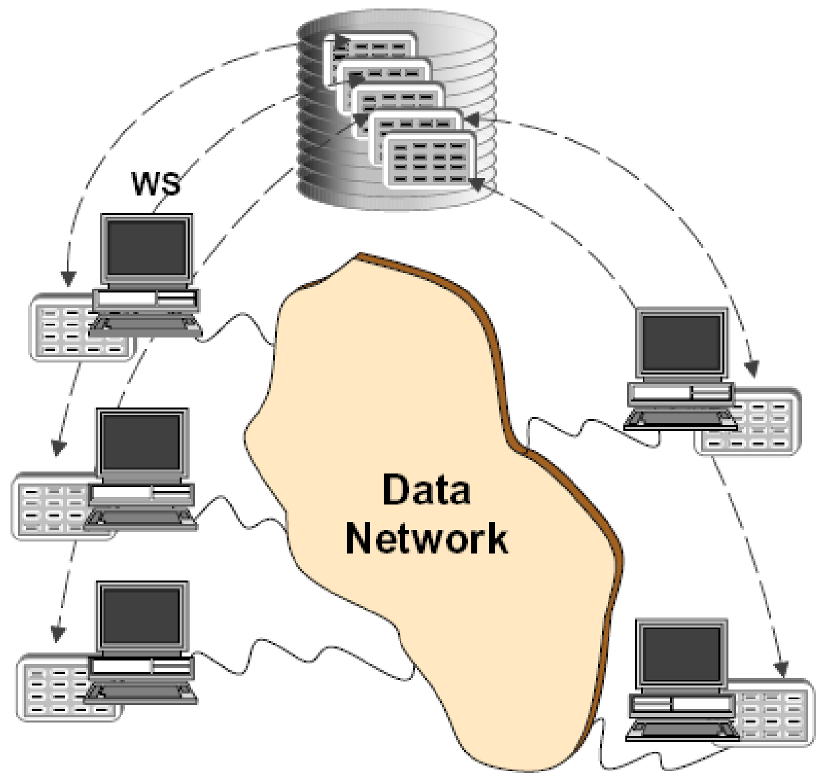


(a) (b)

Distributed File and Memory Architect.

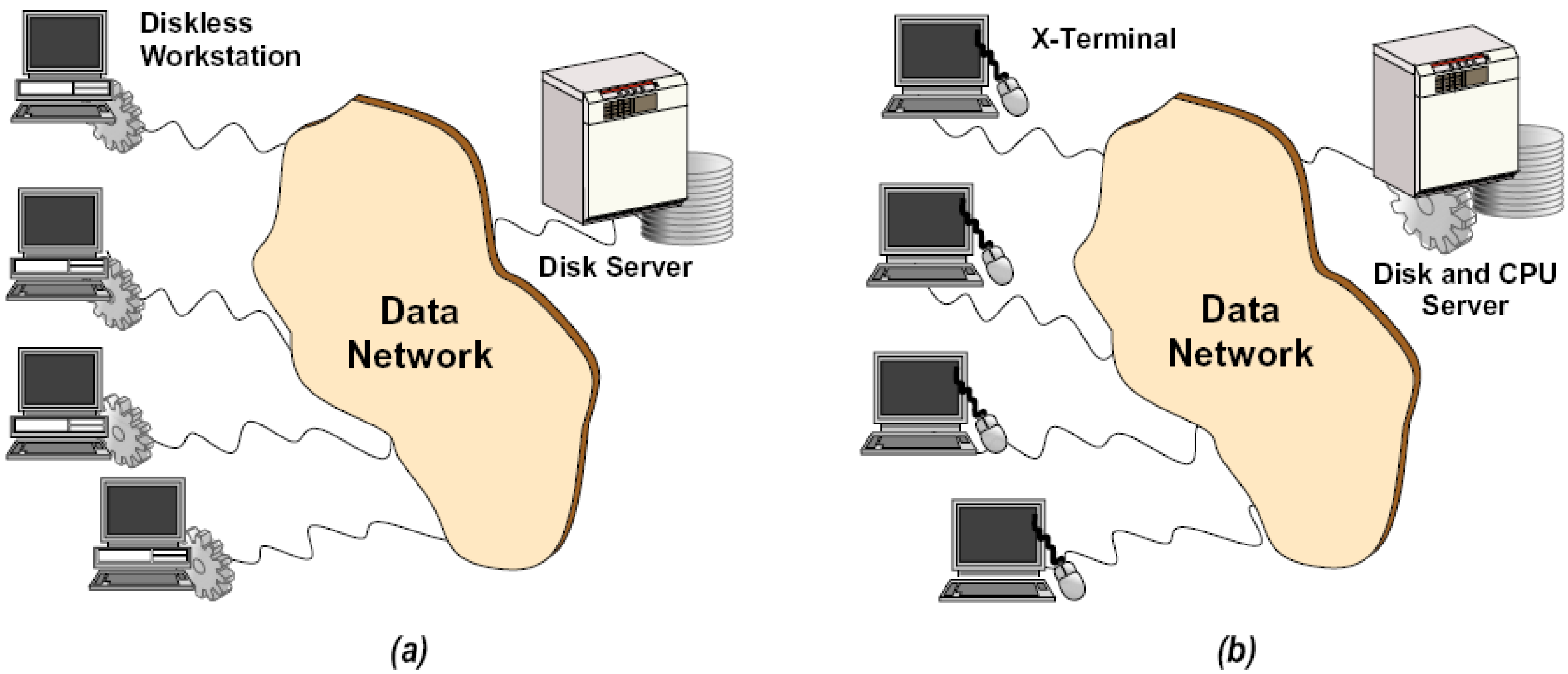


(a)

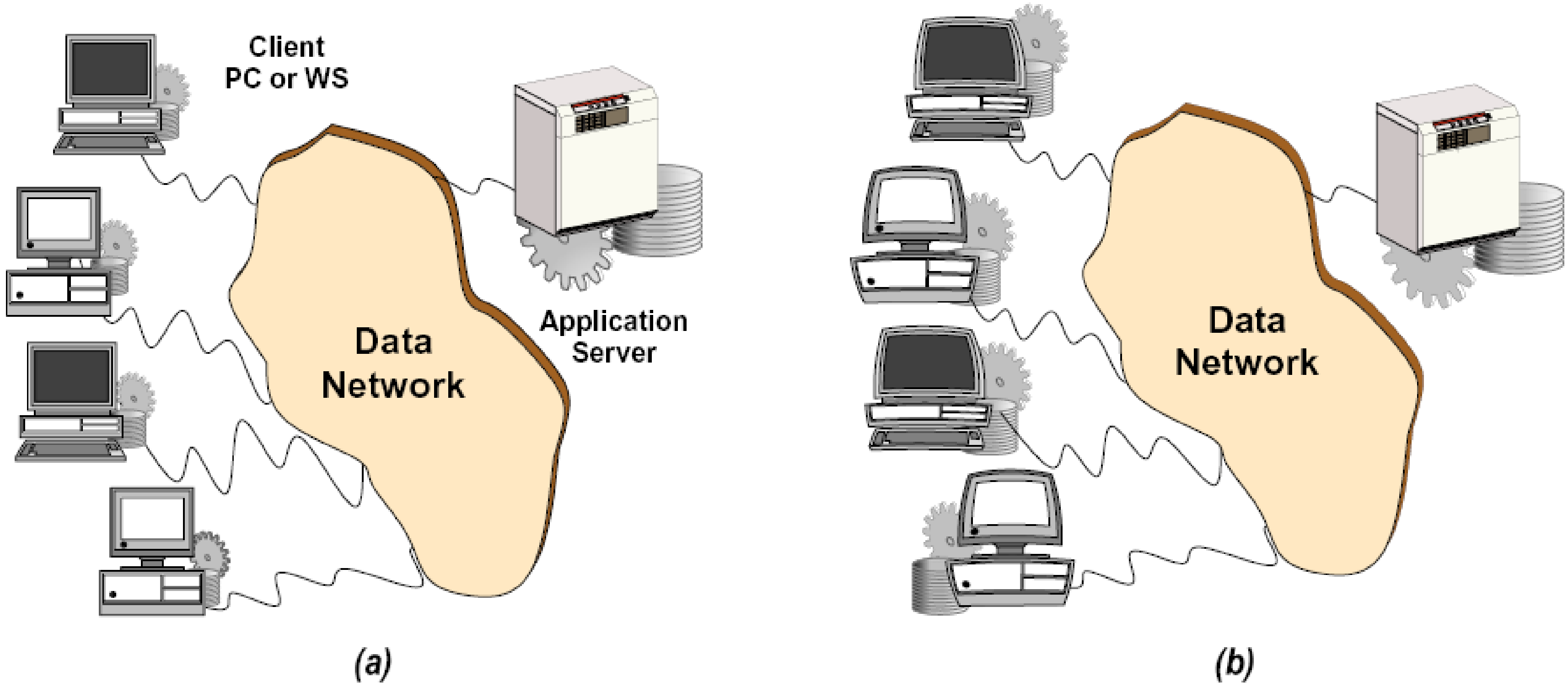


(b)

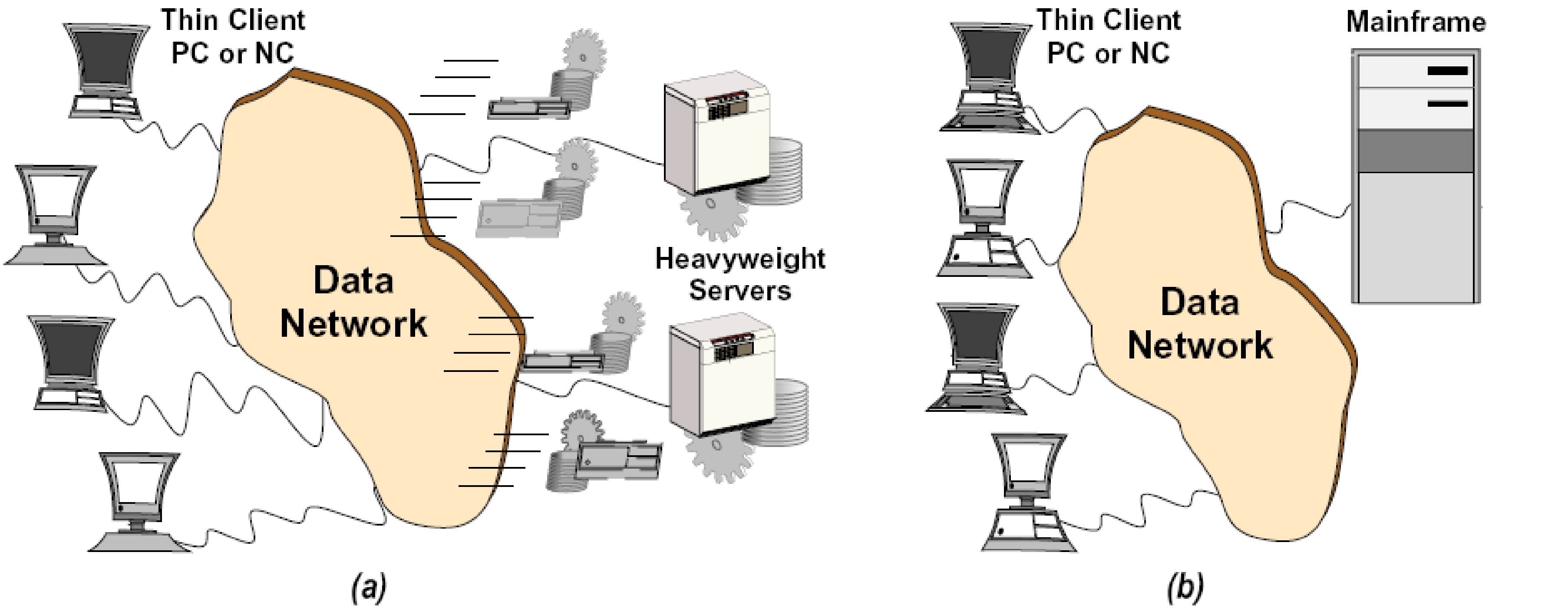
Diskless and X-Terminal Architect.
(on disk and/or CPU servers)



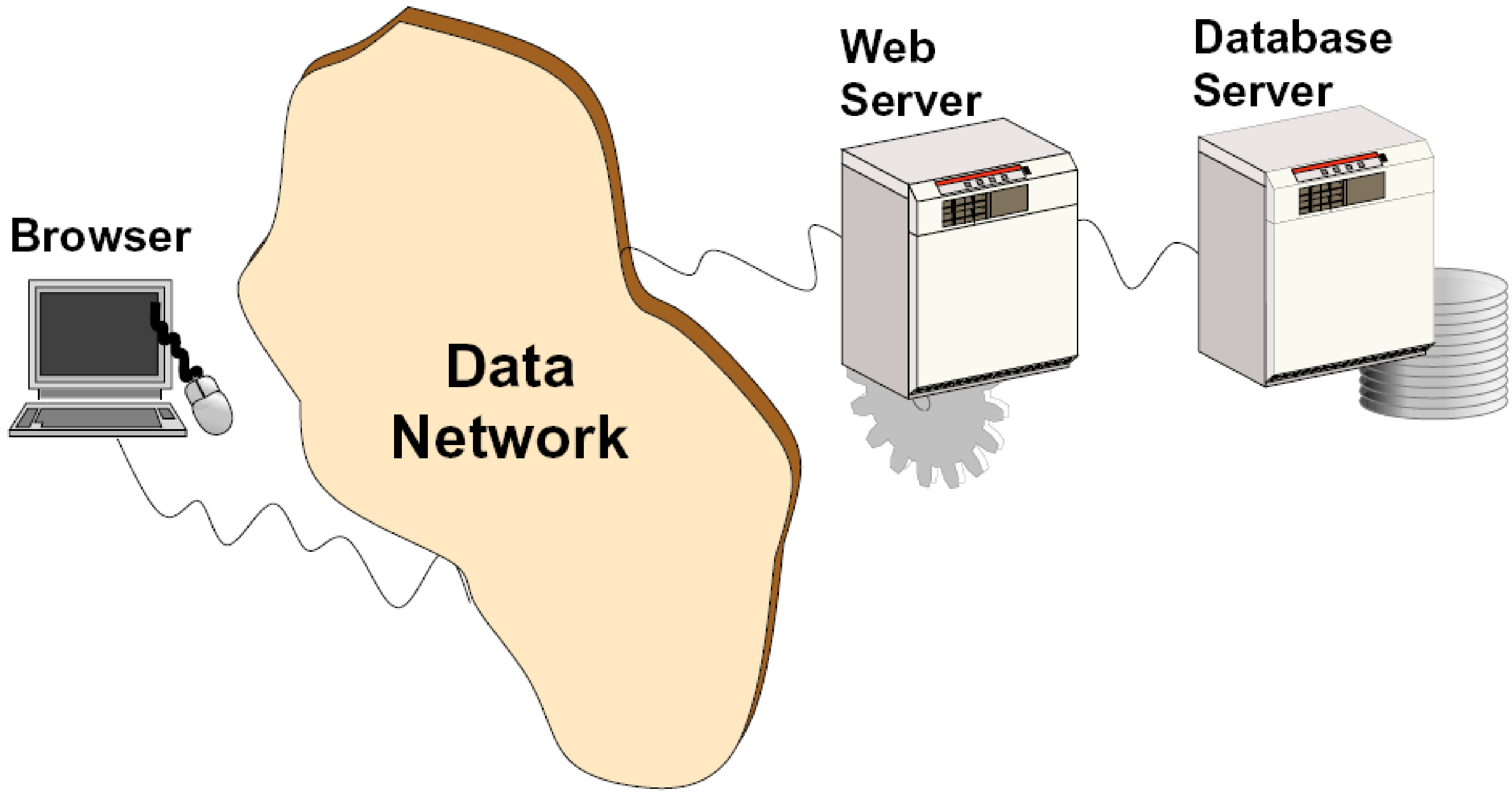
Client-Server Architectures
(Clients became “fat”, perhaps too fat…)



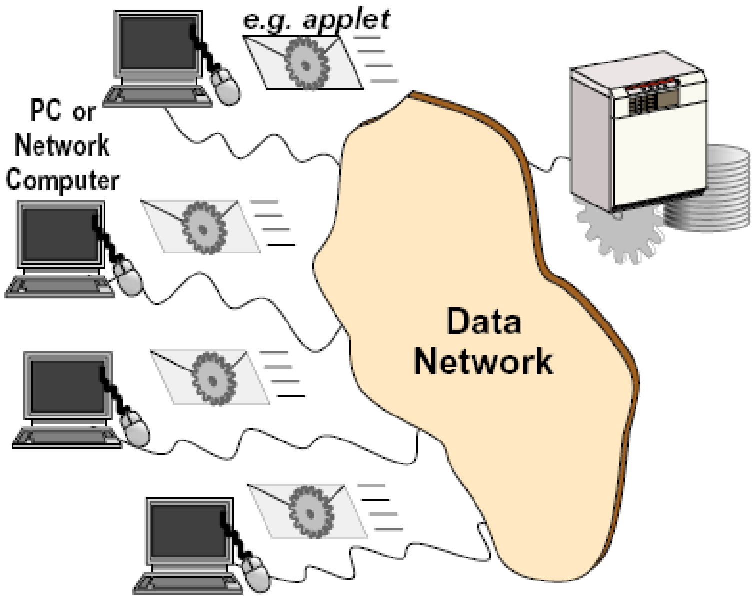
3-tier Client-Server or Thin-Client Arch.
(brought mainframes back, perhaps unnecessarily)



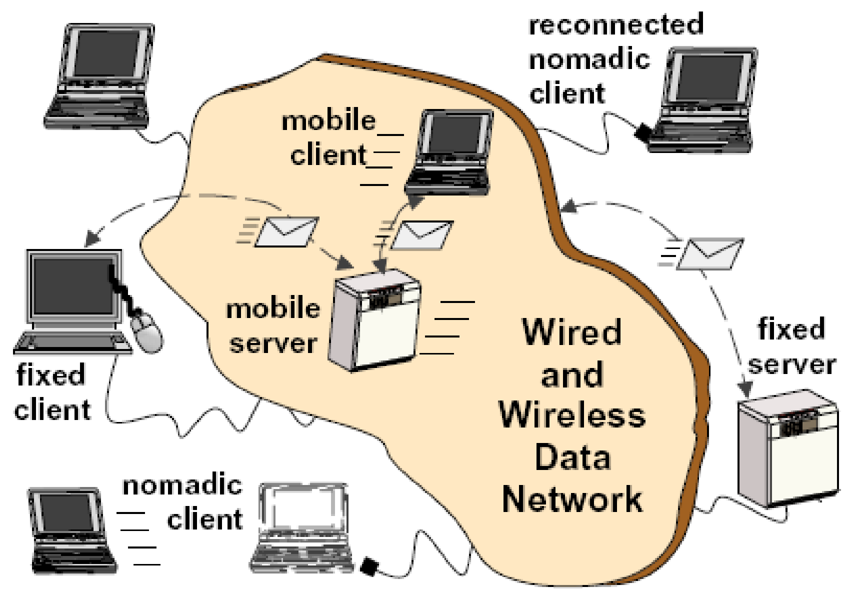
3-tier Client-Server Web-based Archit.



Mobile Code Architectures:
(a) Portable and Mobile Code; (b) Mobile Nodes

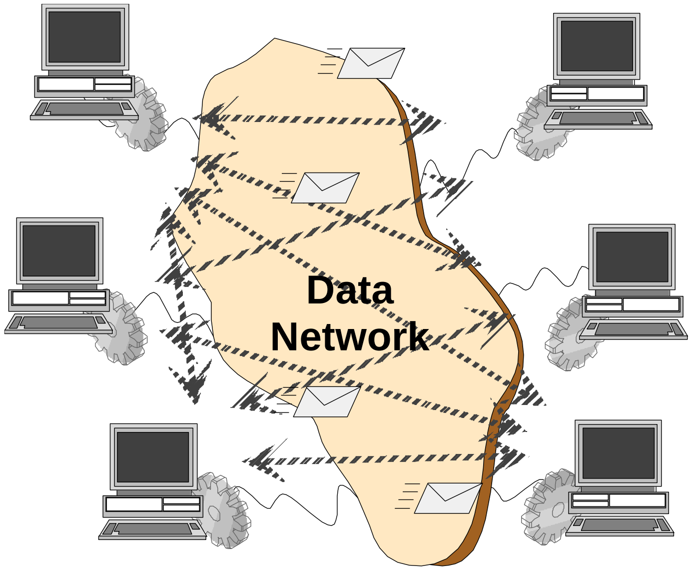


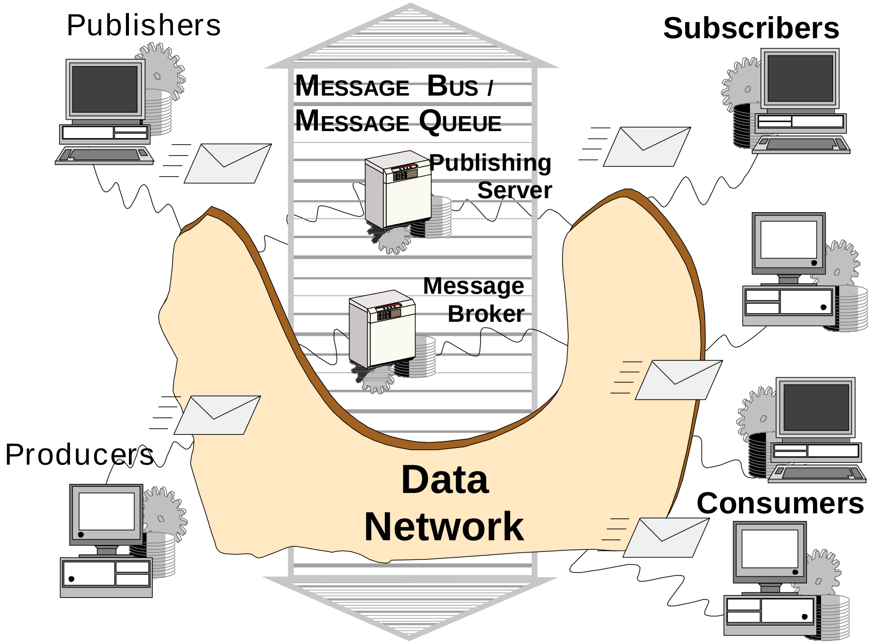
(a)



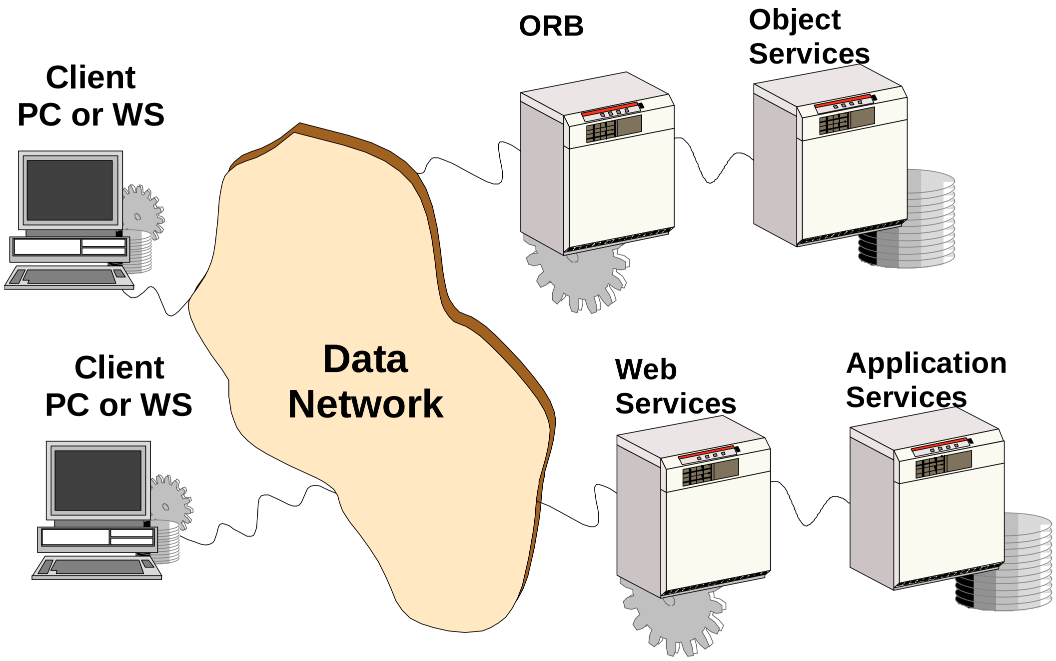
(b)

Message and Event-based Architectures:
(a) Multipeer; (b) Publisher-subscriber, Message Queue

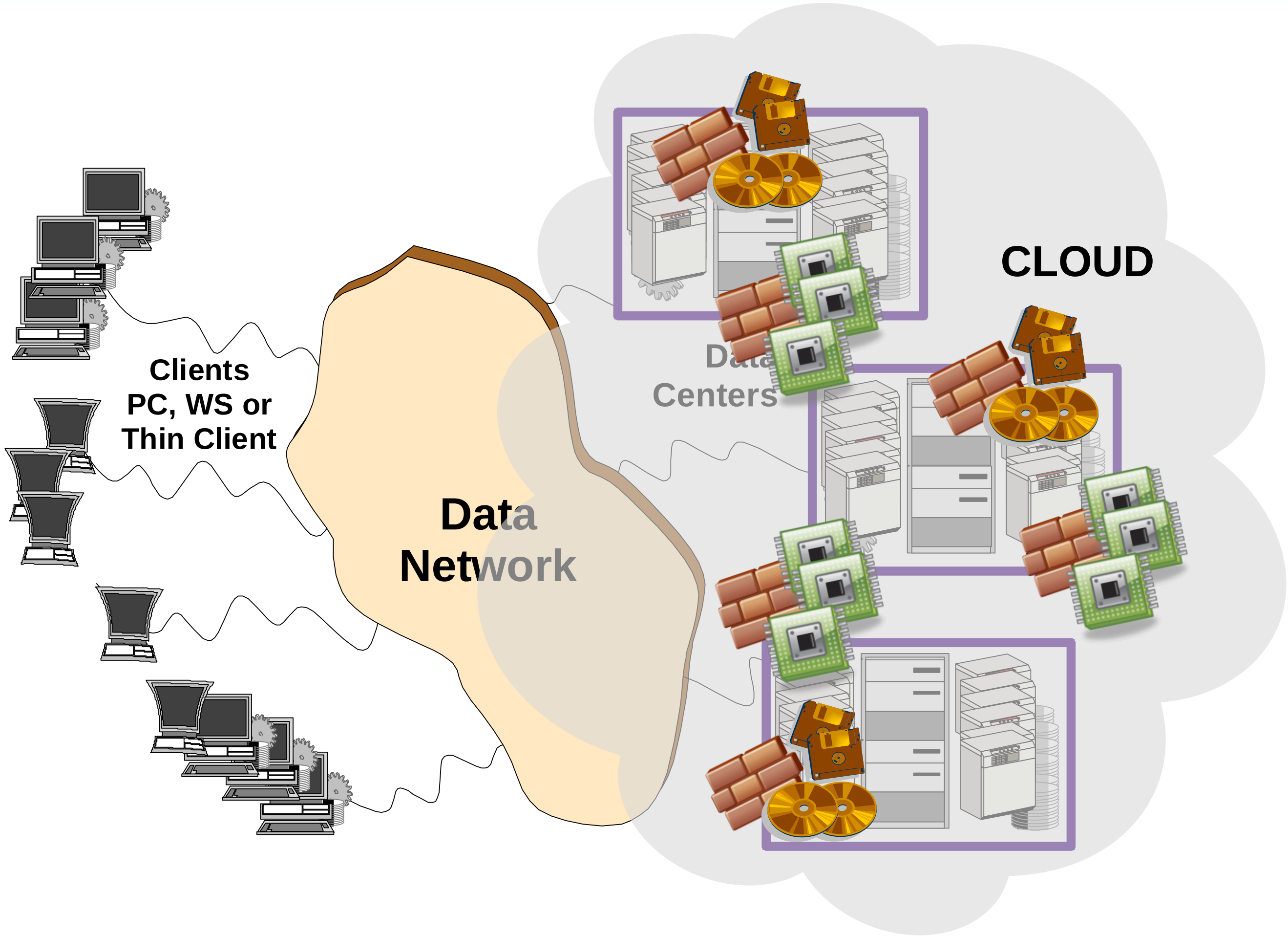




Distributed Objects, Web Services
(Web-based Architectures episode II)



Cloud Computing



Rudiments of: Dependability, Real-Time, Security

Non-functional properties of Distributed Systems

- What systems **are**, rather than what **they do**
- Some familiar names:
  - Dependability and fault-tolerance: **reliability** and **availability**
  - Real-time: **timeliness** and **predictability**
  - Security: **confidentiality** and **integrity**

- **Dependability** **(section 6.1)**
- **Timeliness** **(section 11.1)**
- **Security** **(section 16.1)**

Rudiments of:
Dependability

How reliable are distributed systems?

A distributed system is one that prevents you from working due to the failure of a machine we even had never heard of.

[L. Lamport]

- Since:
  - machines may fail independently (and still…),
  - but influence each other,
  - communicating through non-reliable networks with unpredictable delays
- ... “gathering machines” just worsens situation:
  - reliability (<1) of a system is equal to the product of the reliabilities of all individual components needed for the system to work correctly
  - R(10 machines@ 0.99)= 0.99^{10}= 0.90
  - R(10 machines @ 0.90)= 0.90^{10}= 0.35

How do distributed systems fail?

Why do computers fail and what can we do about it?

[J. Gray]

- Why:
  - because all that works fails
  - maybe they look like failing too much…
  - but we have a tendency of overestimating our HW and SW
- So?:
  - better prevent than remedy
- Dependability is ...
  - the trustworthiness of a computing system in fulfilling a set of properties, which allows justifiable trust on the service it delivers
- How?
  - by appropriate mechanisms that allow the prevention or tolerance (as well as quantification or prediction) of disturbances

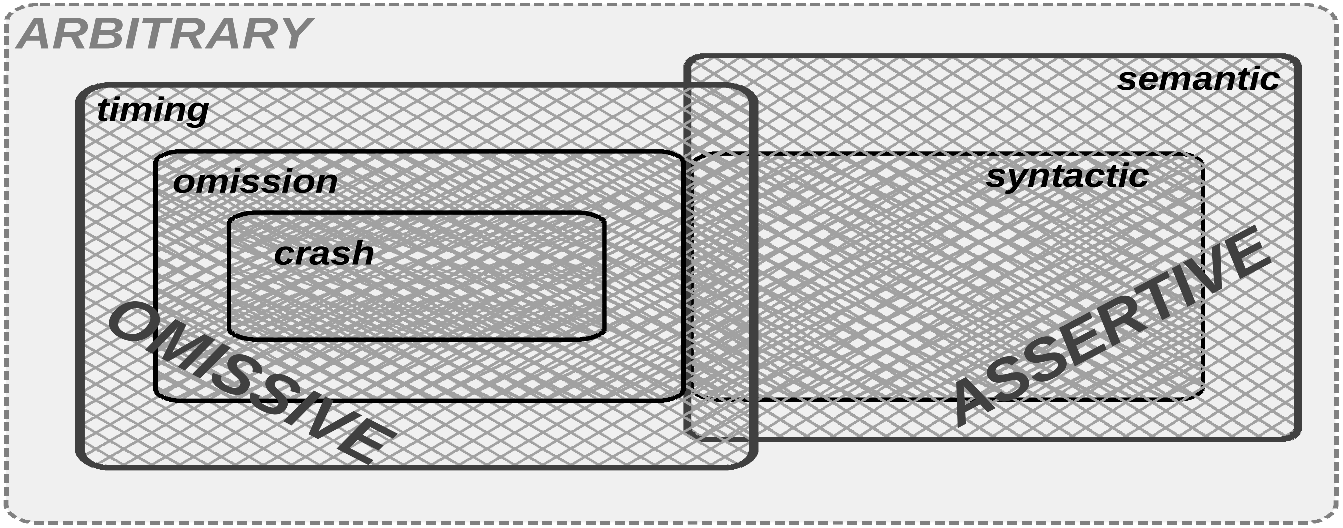
Threats to dependability

They upset the service execution during the lifetime of the system.

- Types:
  - **Falta (fault)** – cause of an error
  - **Erro (error)** – manifestation of a fault in the state of the system
  - **Falha (failure)** – consequence of an error on the service of system
- Simple Examples:
  - **Fault-** bit stuck at zero in a RAM; a packet is lost in transmission
  - **Error** – reading zero, after having written one; message never arrives
  - **Failure-** the wrongly read bit is delivered to the user; message never delivered to user

- **Possible solutions to the examples above:**
  - **Detection- memory with parity bit; timeout set for packet arrival**
  - **Recovery- memory with ECC; repeat transmission if timeout expires**
  - **Masking- replicate memory; systematically repeat a transmission**

Interaction Faults classification
(specially important in distributed systems)



- **Omissive**
  - **Crash**
    - Interactions stop
  - **Omission**
    - some interactions are missed
  - **Timing**
    - interactions out of time (early or late)
- **Assertive**
  - **Syntactic**
    - temperature sensor that indicates +ab degrees
  - **Semantic**
    - temperature sensor that indicates 26º when it is 30º

Dependability properties

- **Reliability**
  - The measure of the continuous delivery of correct service
  - Can be expressed in terms of Mean Time To Failure (MTTF) or **Mean Time Between Failures (MTBF)**
- **Maintainability**
  - The measure of the time to restoration of correct service
  - Can be expressed in terms of **Mean Time To Repair (MTTR)**
- **Availability**
  - The measure of delivery of correct service with respect to alternation between correct and incorrect service
  - Can be expressed in terms of a ratio between uptime and total time, or using the previous two metrics: **Availability = MTBF / (MTBF+MTTR)**
- **Safety**
  - The degree to which a system, upon failing, does so in a non-catastrophic manner

Means to Attain Dependability

- Fault **prevention** – how to prevent the occurrence of faults?
  - Avoid design defects (good process; tools; etc.)
  - Shielding, robust design, good operational procedures etc. for runtime faults

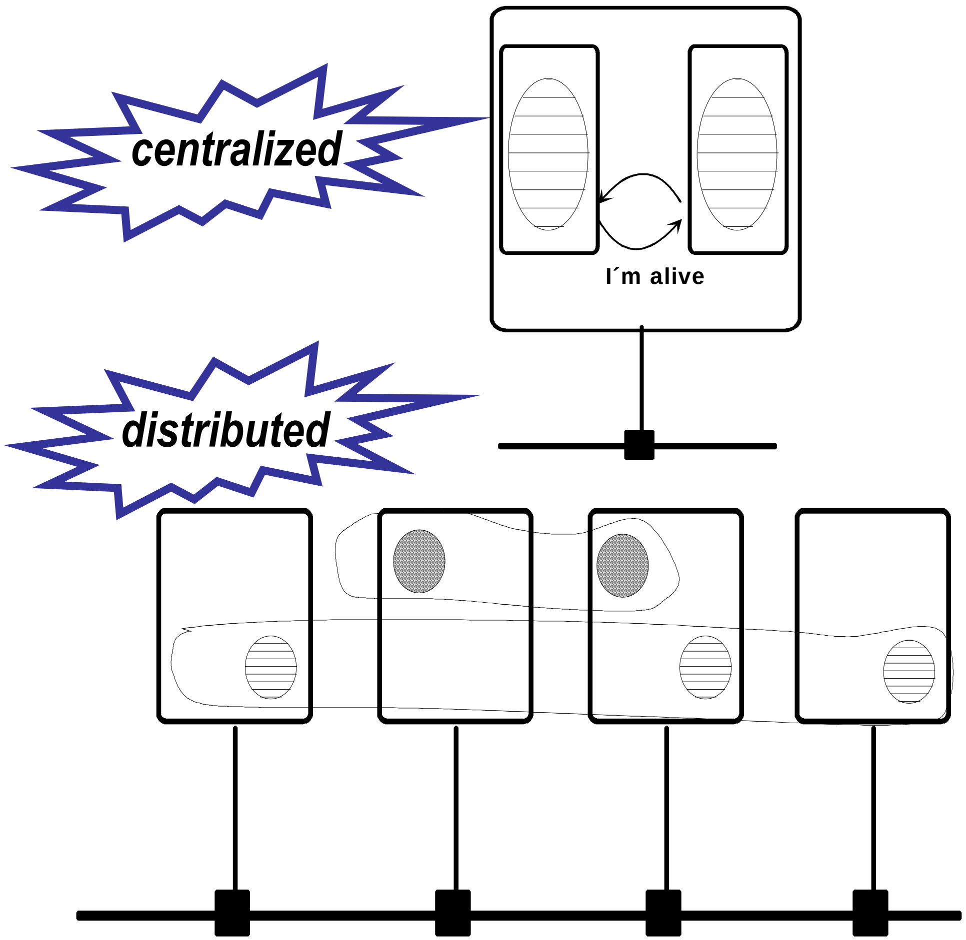
- Fault **tolerance** – how to deliver correct service in the presence of faults?
  - Error detection
  - Recovery
  - Fault handling

- Fault **removal** – how to reduce the number/severity of faults?
  - Verification & validation (design phase)
  - Corrective & preventive maintenance (operations)

- Fault **forecasting** – how to estimate the future incidence of faults?
  - Simulation, modeling, prediction from process metrics (data from models)
  - Historical data, accelerated life testing, etc. (data from real systems)

Foundations of modular and distributed fault tolerance

- **Topological separation**
  - Failure independence
  - Graceful degradation
- **Replication**
  - Software vs. hardware
  - Fine granularity
  - Resource optimization
- **Incremental F/T by**:
  - Class (omissive, semantic)
  - Number of faults
  - Number of replicas
    - pairs, triples, etc.
  - Type of replica control
    - active, passive
    - round robin, voting



Rudiments of: Real-Time

Real-Time (RT) Systems

- **Real-time (RT) system**: system which has at least one service whose progression is specified in terms of timeliness requirements dictated by the environment
  - The “real-time problem” is dictated by the need to synchronize our actions with the environment.
  - Real-time is about predictability (timeliness)
- Some **misconceptions**:
  - RT is ad-hoc design, assembly program., interrupts, and so forth;
  - RT systems are automata, pre-programmed and static;
  - RT is about having enough speed, and ever-increasing MIPs and Mbauds will solve all “performance" problems;
  - RT deadlines do not make sense, since they will be missed because failures occur, messages get lost, software has bugs, etc.
- **Examples of real-time systems**:
  - Oven controller, Manufacturing cell, Traffic Light Controller, Air traffic control system

Classes of RT Systems
importance of scheduling

- Hard real-time systems, where timing failures are to be **avoided**
  - E.g. on-board flight control system (fly-by-wire)

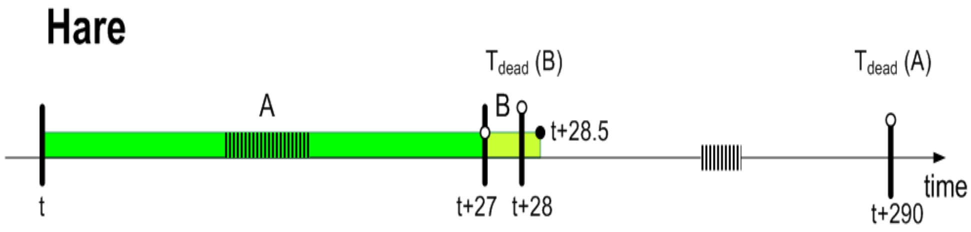
- Soft real-time systems, where **occasional** timing failures are accepted
  - E.g. on-line flight reservation system

- Mission-critical real-time systems, where timing failures **should be avoided** and occasional failures are handled as exceptional events
  - E.g. air-traffic control system

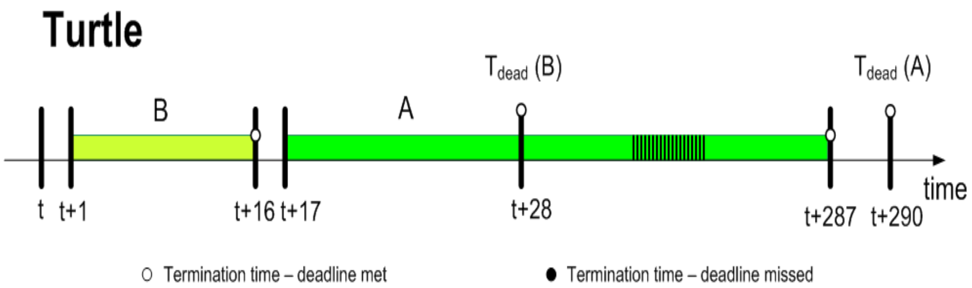
Illustrating the scheduling problem:
the Fable of the Hare and the Turtle...

Fable of La Fontaine, adapted to a R/T scheduling problem by G. LeLann

- HARE System:
  - Speed = 10;
  - Context switch = 0
  - Scheduling: Task A is started first



- TURTLE System:
  - Speed = 1;
  - Context switch = 1
  - Scheduling: Task B is started first

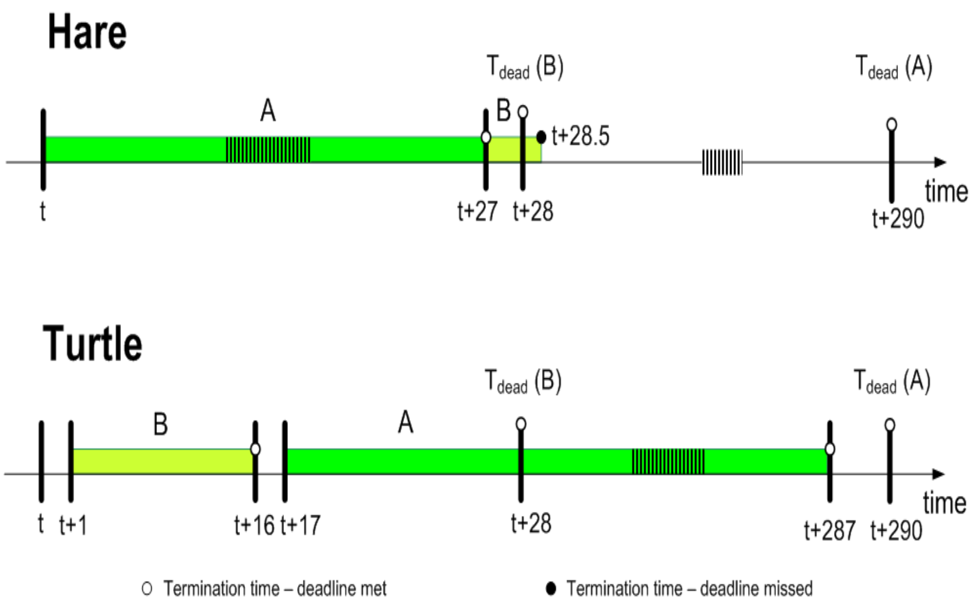


- Problem Definition:
  - Task A: duration= 270; deadline= t+290
  - Task B: duration= 15; deadline= t+28
  - Tasks A and B released at same time

Illustrating the scheduling problem:
the Fable of the Hare and the Turtle...

Fable of La Fontaine, adapted to a R/T scheduling problem by G. LeLann

- HARE System:
  - Speed = 10;
  - Context switch = 0
  - Scheduling: Task A is started first
- TURTLE System:
  - Speed = 1;
  - Context switch = 1
  - Scheduling: Task B is started first



- Problem Definition:
  - Task A: duration= 270; deadline= t+290
  - Task B: duration= 15; deadline= t+28
  - Tasks A and B released at same time

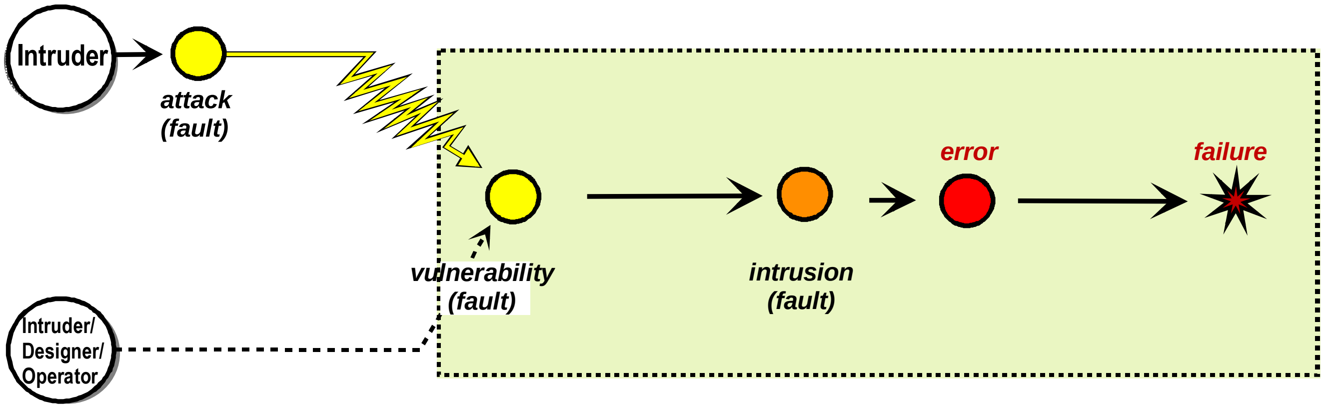
- **Results:**
  - **HARE, optimistic, starts to serve A and misses the deadline of B**
  - **TURTLE, realistic, starts to serve B (closer deadline) and meets both deadlines.**

Rudiments of:
Security

Insecurity, People and Computers

- Insecurity is as concerned with technical deficiencies as with **people's attitudes**
- Insecurity is quantitatively caused to a great extent by the actions of people who must be **educated** about the seriousness of their deeds
- It is better to invest than to spend: **improve your system before intrusion**
- **Don’t use crypto “just because”:** cryptocracy (a form of technocracy) is enemy of good systems practice

A systematic view of security failure
sequence Attack-Vulnerability-Intrusion



AVI sequence : *attack + vulnerability→ intrusion → error → failure*

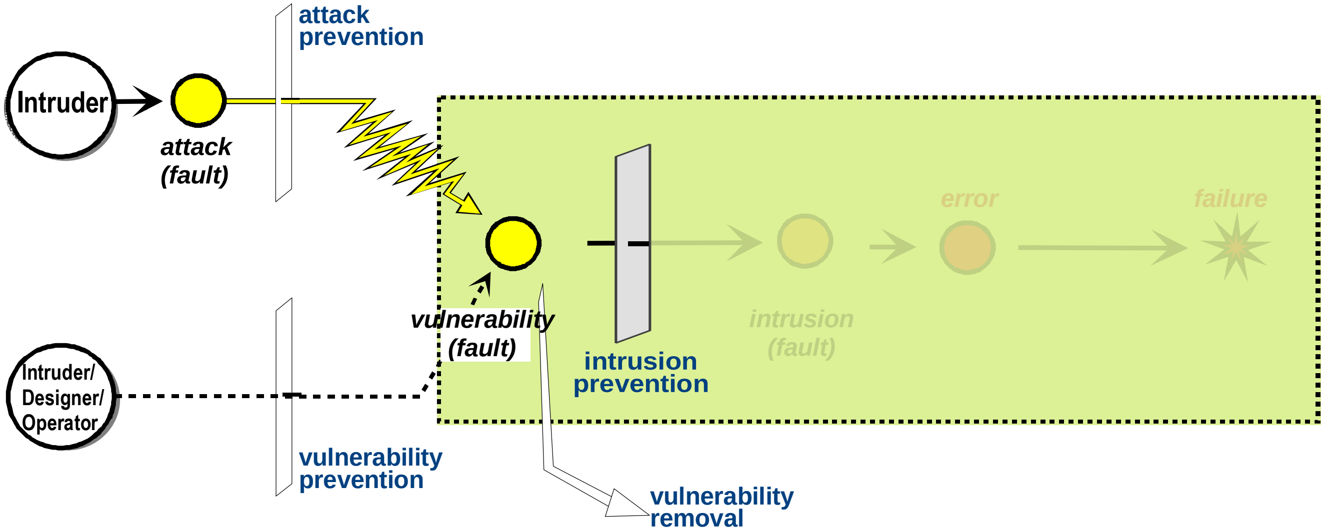
Security Properties

- **Confidentiality**
  - The measure in which a service or piece of information is protected from unauthorized disclosure
- **Integrity**
  - The measure in which a service or piece of information is protected from illegitimate and/or undetected modification
- **Authenticity** (*)
  - The measure in which a service or piece of information is genuine, and thus protected from personification or forgery
- **Availability**
  - The measure in which a service or piece of information is protected from denial of authorized provision or access

(*) Considered integrity of meta-information by some authors

Avoiding security failure and achieving security properties

canonical track: intrusion prevention



sequence : *attack + vulnerability→ intrusion→ failure*

Some formal notions

Formal system predicates
a colloquial definition

- **Objective**:
  - Specify in a formal manner, including formulas containing logic (*and, or, exists, forall*), temporal logic (*eventually, always*) and time (*until/from*) operators

- We can specify the properties of any program or protocol in terms of properties of: **safety** and/or **timeliness** , **liveness**

Formal system predicates
a colloquial definition

- **Safety**
  - the measure in which a service/program **does not do bad things**
  - safety properties specify that wrong events never take place
  - a safety property specifies that a predicate P is always true
  - example, "any delivered message is delivered to all correct participants" is a safety property
  - If it is not secured, the system becomes incorrect
  - however, it does not impose that messages are delivered at all…

  - So, what do you think is missing here?

Formal system predicates
a colloquial definition

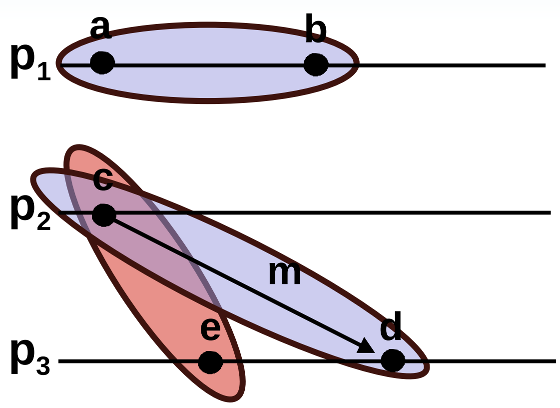
- **Liveness**
  - the measure in which a service/program **does good things**
  - liveness properties specify that good events eventually take place
  - a liveness property specifies that predicate P will eventually be true
  - example, "any message sent is delivered to at least one participant" is a liveness property
  - If it is not secured, the system may not progress (messages are not delivered)
  - With the previous safety property, "any delivered message is delivered to all correct participants" , the system might not do anything at all and still be correct!
  - Together with the current liveness property, "any message sent is delivered to at least one participant" , the system will both be correct and make progress!
  - Why couldn’t we have said: “any message sent is delivered to all correct participants“ ??

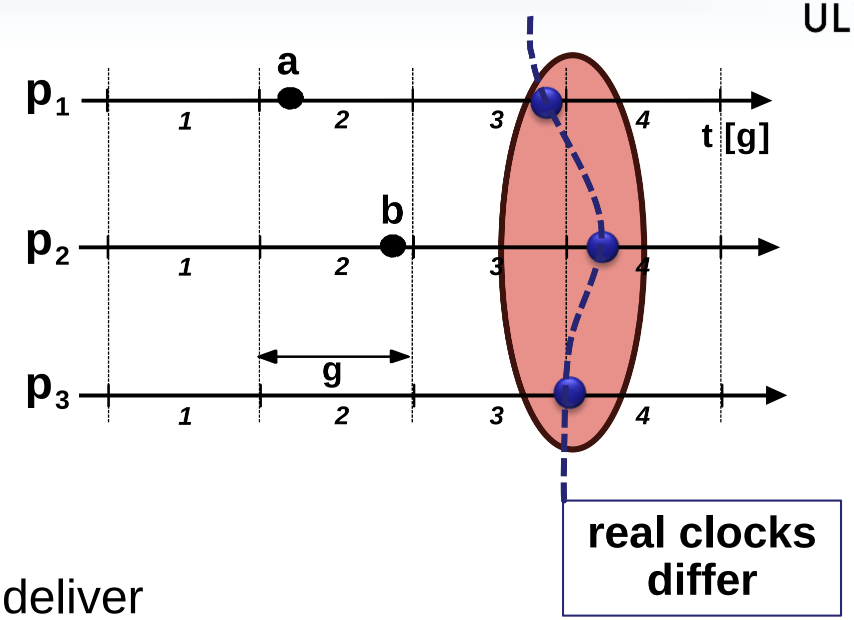
Formal system predicates

a colloquial definition

- **Timeliness**
  - a sub-class of safety property is timeliness, which specifies that **a predicate P will be true in relation to a given instant of real time** (until, before, at)
  - "any transaction completes until Tt from the start“ is a timeliness property
  - for the property to be secured, all transactions must execute within Tt time units

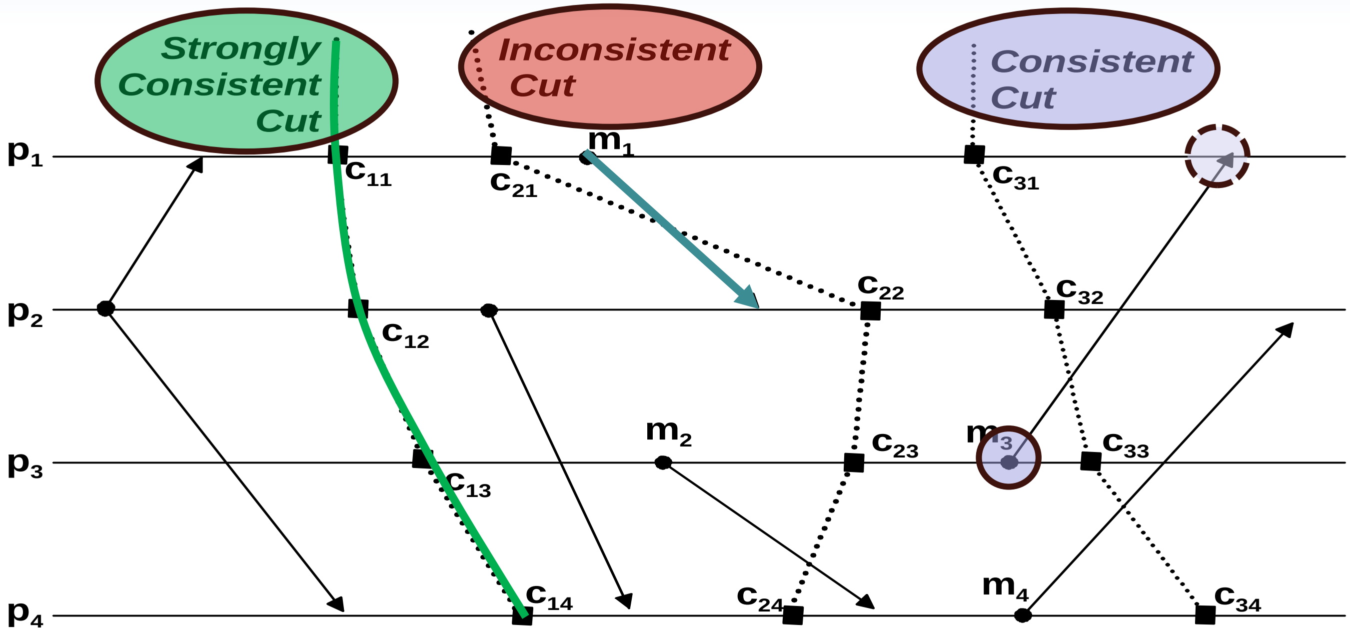
Space-Time and Lattice Diagrams





- Event types:
  - local execution, send, receive, deliver
- Precedence:
  - a < b ; c < d
  - is c < e ?
- Time lattices:
  - based on notion of global time with granularity (tick) g
  - does 4:00 tick at exactly the same place everywhere?

Cuts and Global States (GS)



- Types of cults:
  - **inconsistent** cut: snapshot gives invalid picture of GS
  - **consistent** cut: snapshot gives correct but possibly incomplete picture of GS (e.g., ignores messages in transit)
  - **strongly consistent** cut: snapshot faithfully represents GS