Distributed Systems Foundations

Prof. Carl Hauser

Cpt. S 464/564 Lecture

Chapter 1 from Text

August 26 & 31 2004

 Distributed Systems Foundations: © 2001-2003 Carl Hauser & David E. Bakken. Some figures © 2001 Veríssimo and Rodrigues

Housekeeping Items

Introductions

- Name, class (junior, senior, grad, etc.), employment (current or former)

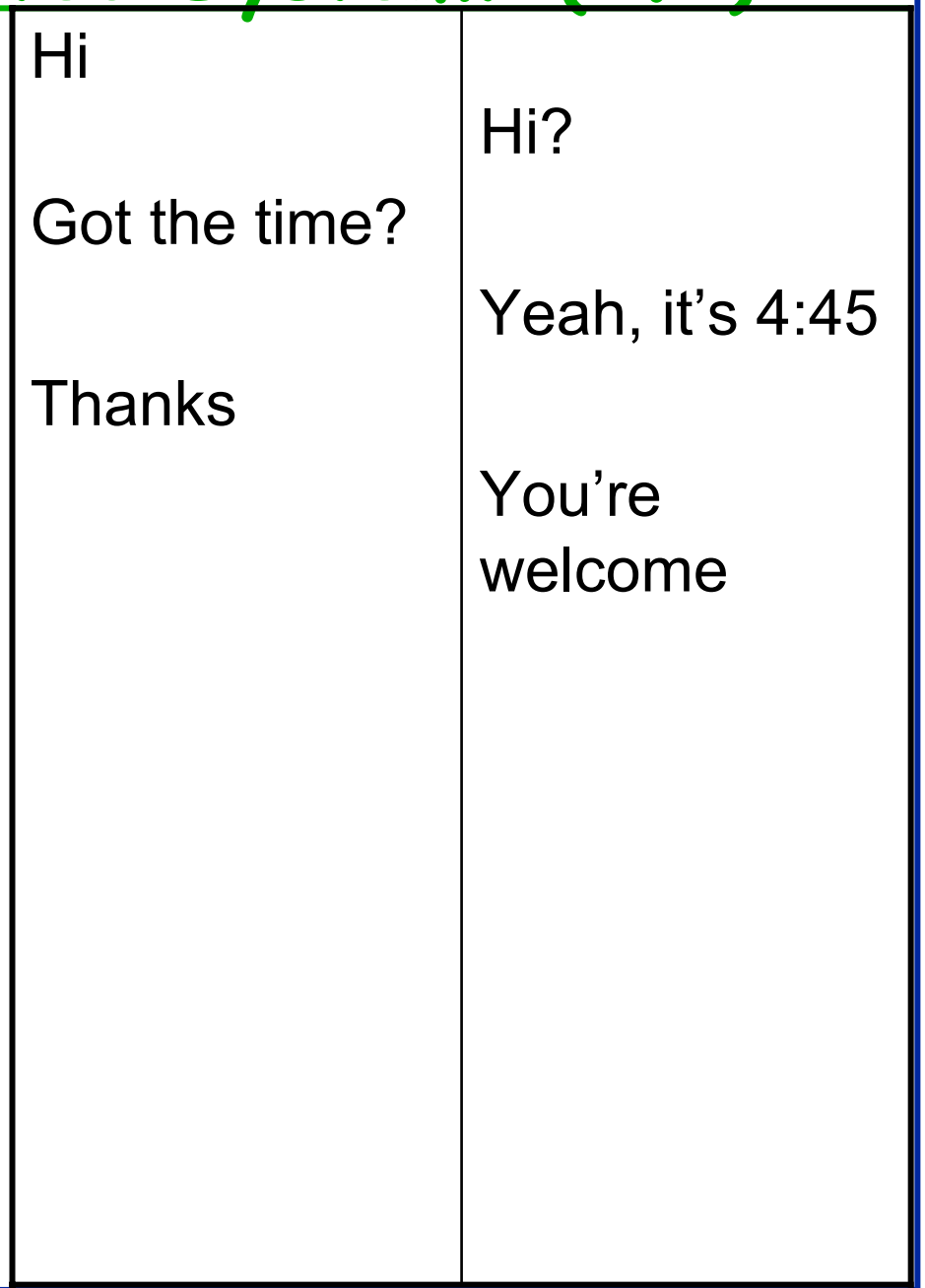
- Roll call

Agenda

- Introduce some terms and the big picture
  - What is a distributed system?
  - When are DS warranted?
  - How did DS evolve?
  - What are some architectures for DS?
- Review some concepts from other courses along the way
- Purpose: provide the foundation for the inquiry and investigation that goes on throughout this class

What is a Distributed System? (1.1)

- Many different kinds of DS, so one definition is not adequate!
- Much easier to talk about characteristics/attributes
- Presence of a **network** is necessary
- DS has more than just a network: set of distributed **protocols** to assist the distributed applications and services
  - What is a protocol? "An agreement that specifies the format [, timing] and meaning of messages [that] computers exchange" [Comer2001]
  - What time is it protocol?



DS Attributes (cont.)

- DS has processes that cooperate to achieve some common goal
- Is the Internet a distributed system: many computers act alone or interact only trivially
  - No: Internet contains protocols and services that are needed for building DSs, but is not, itself, a DS; attaching a computer to the Internet does not automatically make it part of a distributed system
  - Yes: the Internet contains processes such as routers and name servers that use distributed protocols to achieve a common purpose (albeit a low-level one)

DS Attributes (cont.)

- Schoeder's definition of a DS
  - Multiple computers
  - Interconnected by a network
  - Sharing some state
    - What is state?
    - Sharing accomplished across time and space
- Lamport's definition
  - A DS is the one that prevents you from working because of the failure of a machine that you never heard of

DS Attributes (cont.)

- Tanenbaum and van Steen's definition
  - A collection of independent computers that appears to its users as a single coherent system

DS Attributes (cont.):

- Computers/CPUs fail independently
- Communication
  - Is unreliable – Unreliable how?
  - Has moderate latency – What is latency? What is moderate?
  - Has variable latency
  - Has moderate bandwidth – What is bandwidth? What is moderate?
- Hard to determine the relative ordering of events
  - Clocks not synchronized perfectly - example
  - Communication latency
  - Result: only partial ordering on events is known
- Very hard to assess the global state at any one time
  - Messages can be in transit
  - Clocks are not synchronized perfectly

Transparency?

- Hide differences in data representation (Access Transparency)
- Hide where a resource is located (Location T.)
- Hide that a resource may move (Migration T.)
- Hide that a resource may move while in use (Relocation T.)
- Hide that a resource is replicated (Relocation T.)
- Hide that a resource is shared (Concurrency T.)
- Hide failure and recovery of a resource (Failure T.)
- Hide whether a resource is in memory or on disk (Persistence T.)

How much transparency is possible?

- Location – distance matters
- Failure – can affect time taken
- Replication – can affect time taken

Centralized vs. Distributed Sys.

Centralized Systems	Distributed Systems
Local scope and resources	Geographic scope/resources
Homogeneity	Heterogeneity
Manageability	Modularity
	Scalability
Consistency	Sharing
All or none	Graceful degradation/ Partial failure
Security - boundary	Security - permeating

When to Distribute?

- If you don't need a DS, then don't distribute!!!
- Be able to answer "Why do we need a distributed system to solve this problem?"
- Main reasons: distribute when
  1. Problem has a decentralized nature
  2. Distribution techniques provide useful services even if problem isn't naturally decentralized
  3. Problem requires adapting to changes and evolution in the activity and location of organizations

When to Distribute? (cont.)

- #1: Problem has a decentralized nature
- Many locations collaborating on a problem (ex?)
  - Inventory/ordering for a large retailer
  - Financial reporting
  - Distributed astronomy (coordinated use of radio telescopes)
  - Federated computational resources SETI@home, …

When to Distribute (cont.)

- #2: DS techniques can provide useful services
  - Sometimes naturally centralized: e.g. bank database
  - Sometimes centralized business model
  - But even so, sometimes distribution can help…
- Bank account database (DB) example
  - Naturally centralized: easy consistency and performance
  - Fragment DB among regions: exploit locality of reference, **reduce reliance on network for remote access**
  - Replicate each fragment for **fault tolerance**
- Now need DS techniques
  - Route request to right fragment
  - Maintain consistency of fragments as a whole database
  - Maintain consistency of each fragment's replicas

When to Distribute (cont.)

- #3: Adaptability to quickly changing business scenario
  - Allow autonomy in managing services and resources across domains
  - Allow incremental and locally-driven expandability
  - Much easier to add a new branch office (create another replica, etc) if already distributed

Services of Distributed Systems (1.2)

- DSs have a set of basic services (Table 1.3)
  - Often augmented with other specific services useful for particular applications
- Name service
- Registration, Authentication, and Authorization Services
- File Service
- Networking Service
- Remote Invocation Service
- Brokerage Service (fancy name service)
- Time Service
- Administration Services: management tasks

Evolution of Distributed Computing

- Evolving fast since early 1970s
  - Milestone event: ARPANET (BBN, 1969), first internet
  - Not always clear what should and would be distributed (which resources): apps, files, memory, processing
  - Key enablers: **local** and **wide-area** networking
- First stage
  - Sharing files: ftp
  - Remote access to CPUs: telnet
- Ftp is a pain, ergo distributed file systems
  - Same system call (fopen) works local and remote
- Distributed concurrent processing: divide and conquer across machines
- Remote execution: call non-local procedure/method
  - Gave birth to client-server paradigm
  - Remote procedure call: RPC …. then remote objects

Distributed System Architectures (1.3)

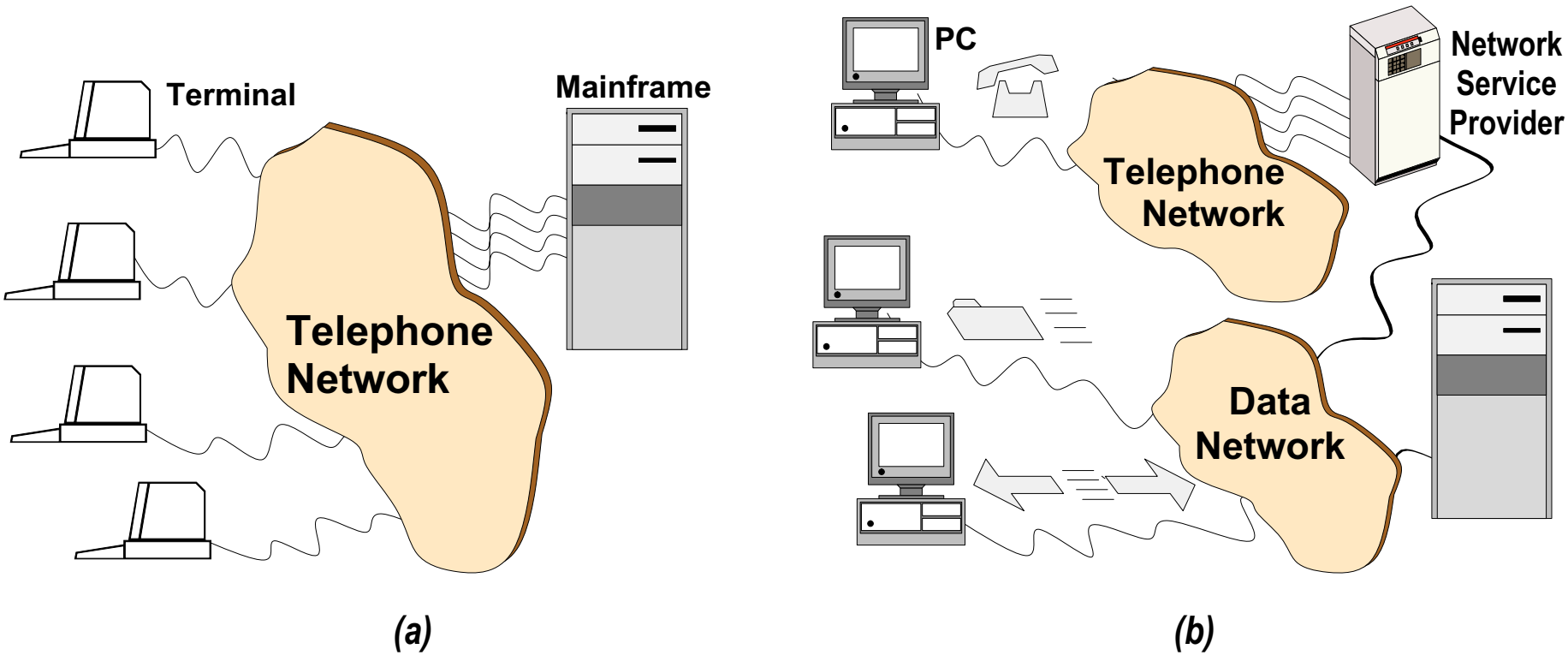
- Q: what is an architecture?
- Why talk about architecture?
  - Concise naming of *some* common combinations of characteristics/features
  - Higher level than DS *models*
  - Less specific than *framework* – a specific realization of an architecture to support building DSs
- DS architectures have evolved as hardware and software available became more sophisticated
- Presented now in rough order of evolution
- High-level architectures given, many ways to implement (we will study some….)

Remote Access

- The most basic form of distribution
- Goal: provide distributed access to centralized facilities

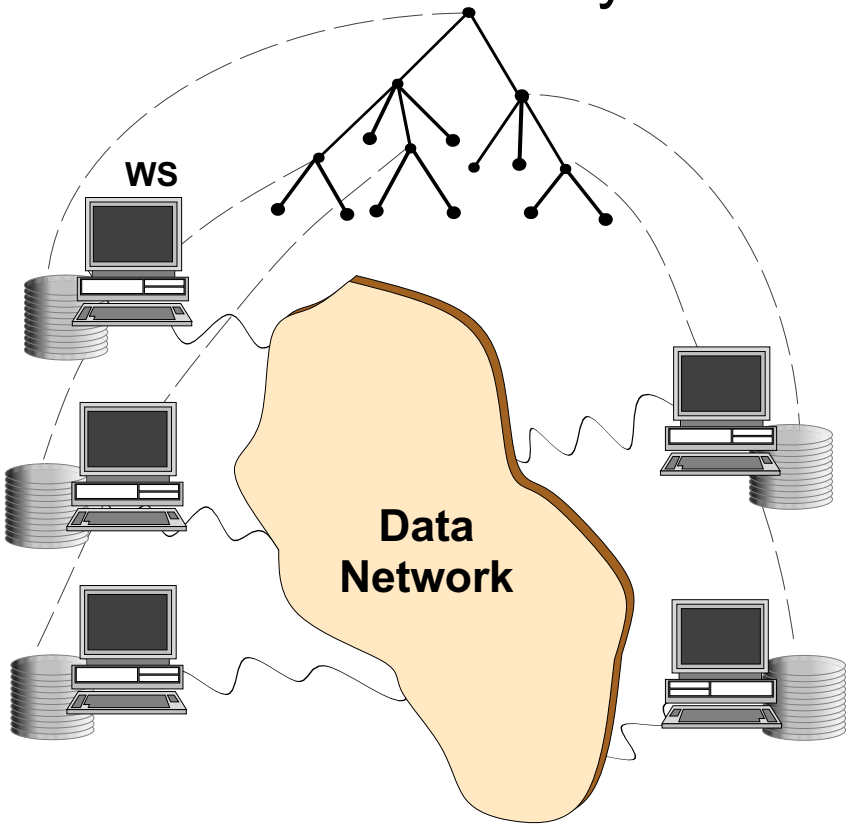
(a) terminal access over POTS: most basic form of DS

(b) remote sessions over data network: files, email, …

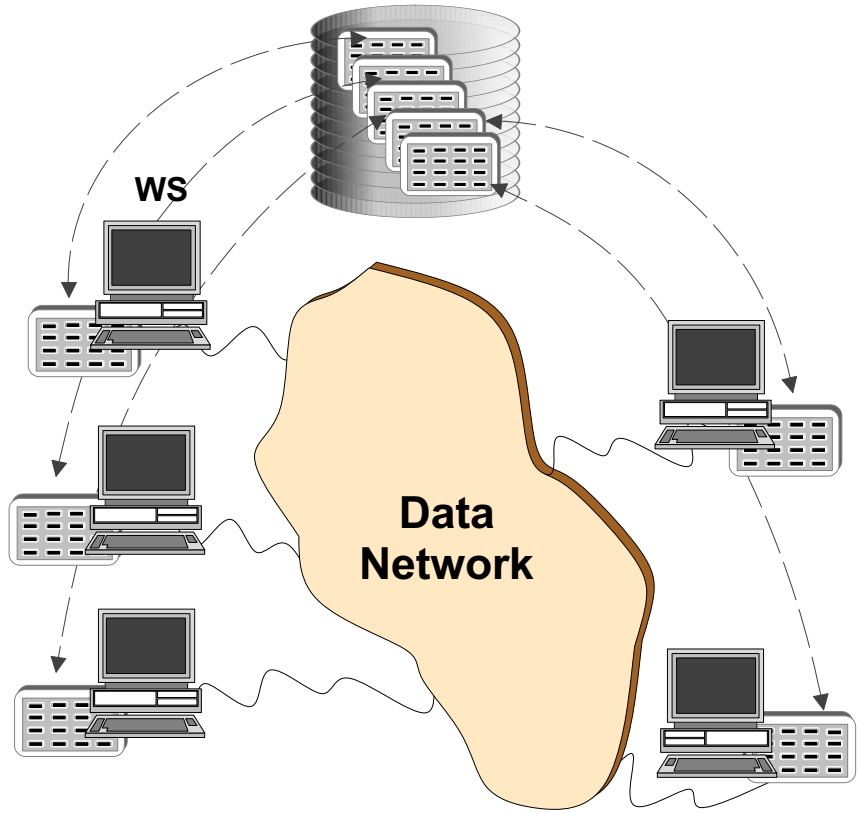


File and Memory Distribution

- Coming of workstations allowed computing power to be spread out
  - Added up to a lot of resources
  - File and memory distribution share on a peer basis



(a)



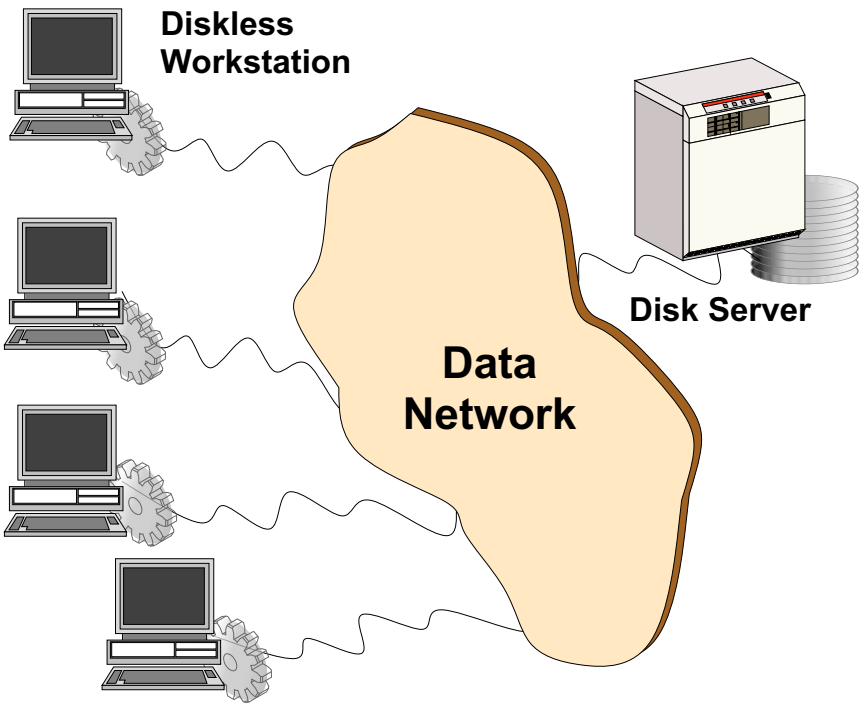
(b)

Remote Access II

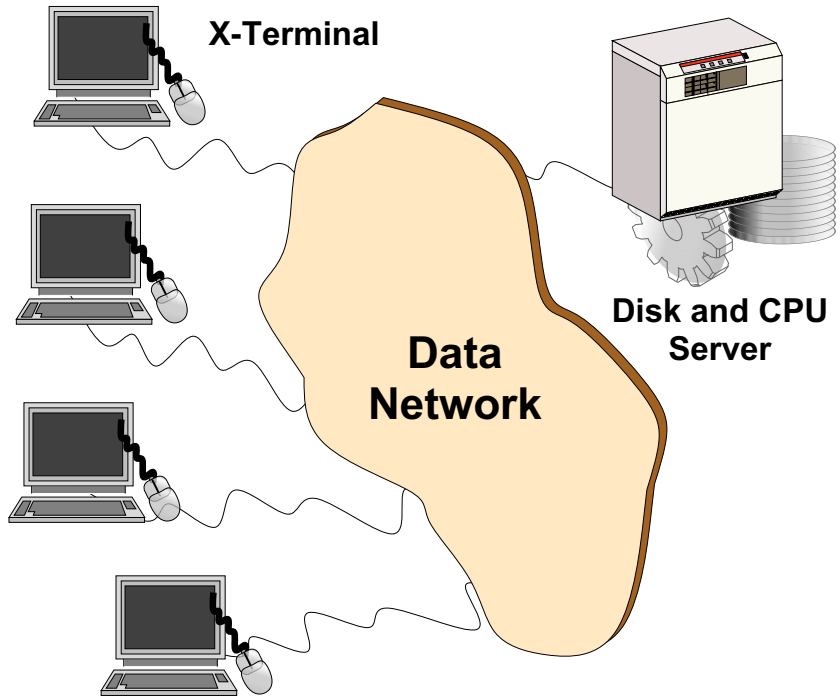
- Oops: DSs not so easy to administer & share perfectly….
- Backlash: re-centralize some resources

(a) Diskless workstations

(b) X-Terminals



(a)



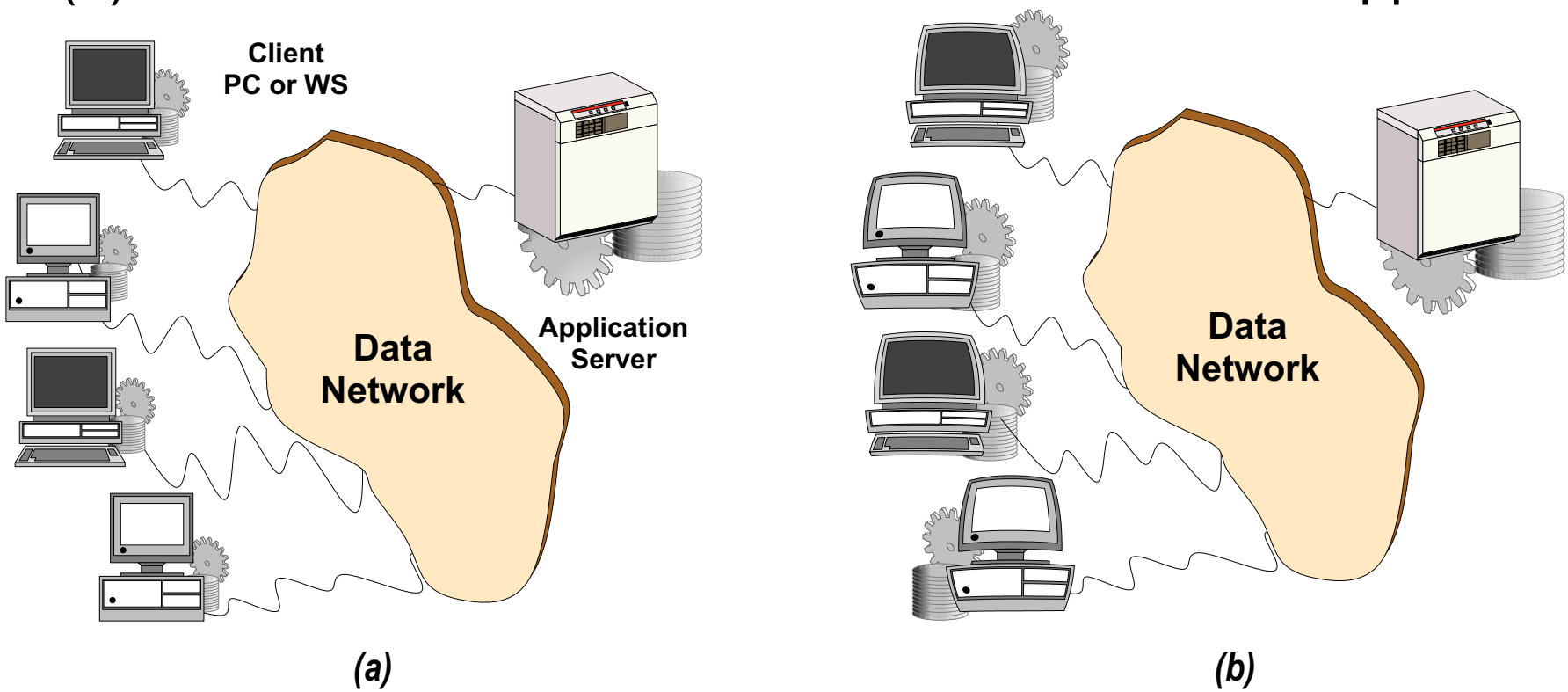
(b)

Client-Server Architectures

- Oops: X-terminals and diskless WSs unsatisfactory
- Client-server architectures: break up application into two remote pieces
  - Most widely-deployed architecture today

(a) Client-Server Architecture

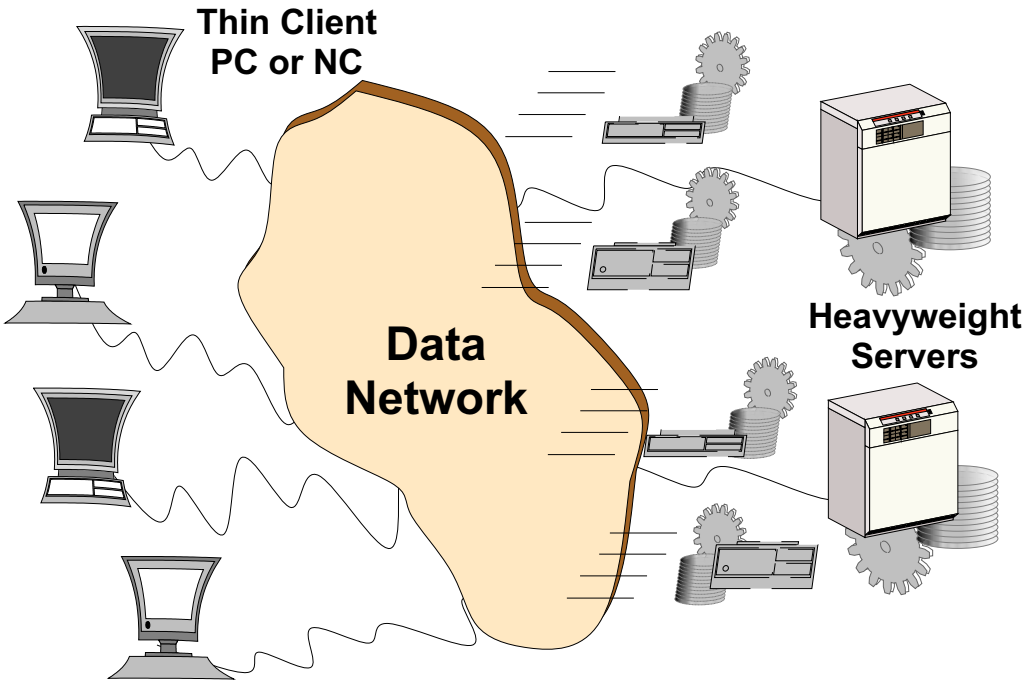
(b) Fat clients: client machines have code for each app.



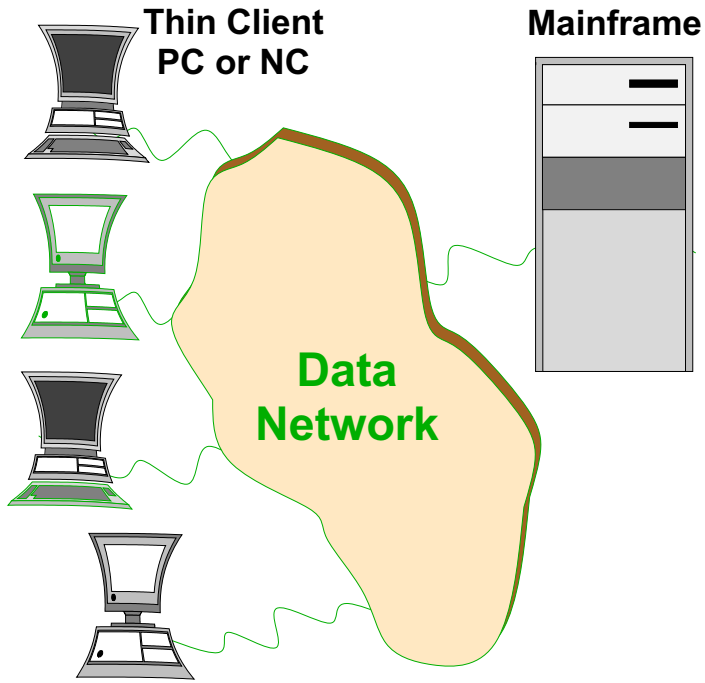
3-Tier Client-Server Architectures

- Oops, fat clients a big pain (code, files, config)
- Oops, too much distribution at server level
- Thin clients: reduced functionality (but more than X-Terminals)
- 3-Tier architecture
  1. Human-client interface (GUI machine)
  2. Application server
  3. Database or secondary servers “behind” app server

3-Tier Client-Server Architectures (cont.)



(a)



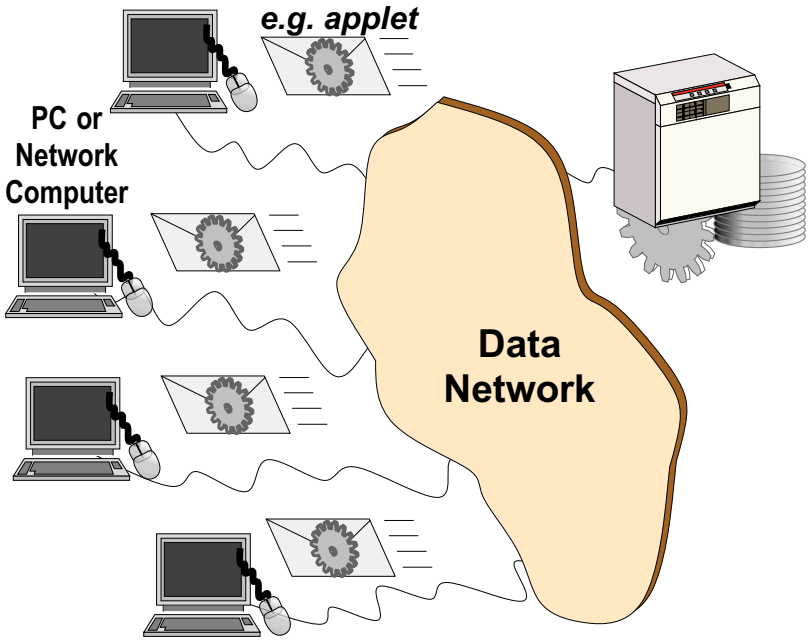
(b)

Mobile Code and Site Architectures

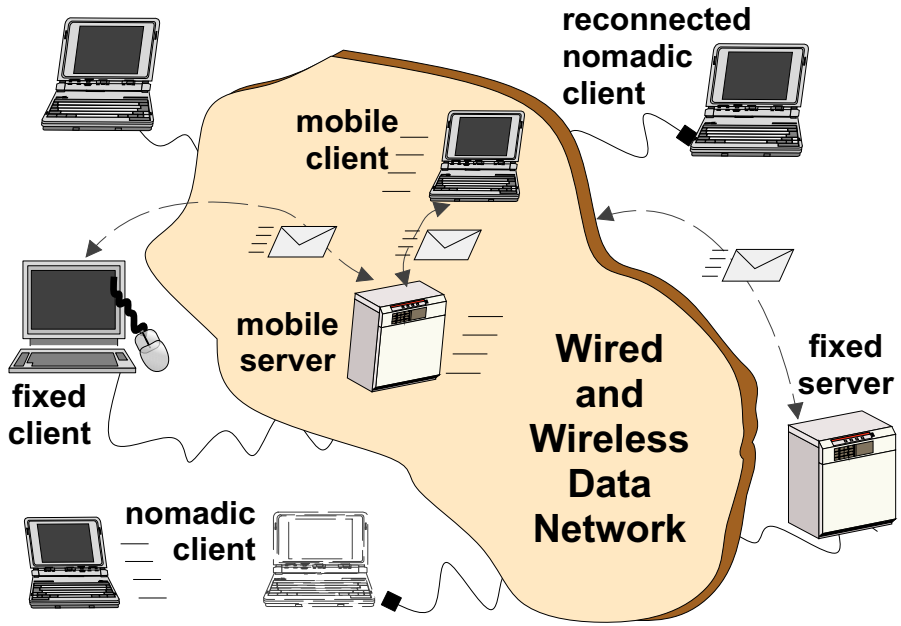
- Oops:
  - thin clients too limited
  - non-dedicated cycles
- Mobile code: ship code when needed to client

  (a) Portable and mobile code

  (b) Mobile Site Architectures: clients and servers can move



(a)



(b)

Mobile Site Architectures (cont.)

- Lots of possible applications
  - military
  - emergency networks
  -
- Variations
  - Only allow client to move (mobile cell phone networks)
  - Only allow sites to move while offline, then reappear ("nomadic computing")… very popular today, growing wildly fast…

Event-Based Architectures

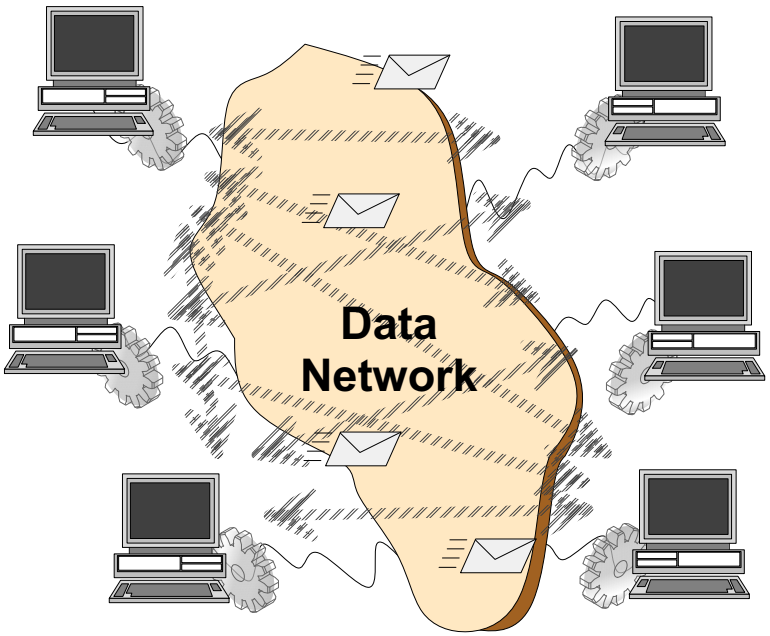
- Oops: client-server paradigm does not represent all kinds of useful distributed applications
- Some are more peer-peer
- Some require handling unsolicited events
- Example: client server
  - client: initiates request, gets reply
  - server: maintains state, gets request, sends reply based on state
- Event-based (AKA message-based) architectures
  - Server generally maintains state, but can initiate events / messages / invocations
  - E.g., when its state crosses a threshold
  - Examples?

Event-Based Architectures (cont.)

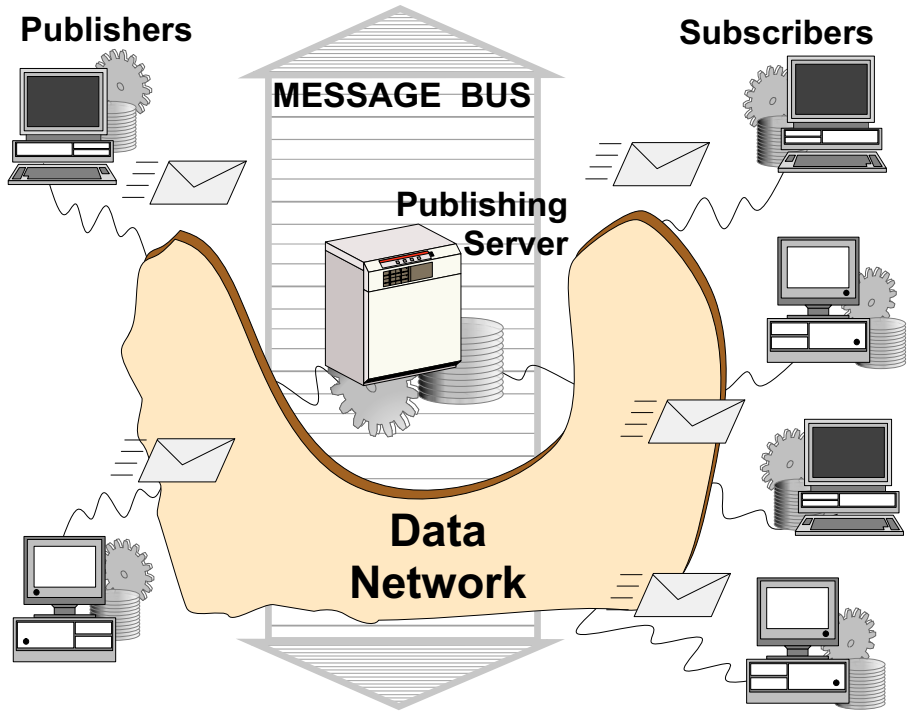
- Event-based communication often multicast/group
- Can run multipeer conversations: directly sending messages to another
- Asymmetric example: producer-consumer
- Example apps: email, USENET news
- “Push” technologies are producer-consumer
- Enhancement: let producers and consumers not have be online at same time: message queues
- Enhancement: let consumers specify (subscribe) only what (type,value) of messages they receive: “publish-subscribe” (AKA “message bus”)

Event-Based Architectures (cont.)

- Figures
  - (a) Multipeer
  - (b) Publish-subscribe



(a)



(b)

Formal Notions for DSs (1.4): DS Models

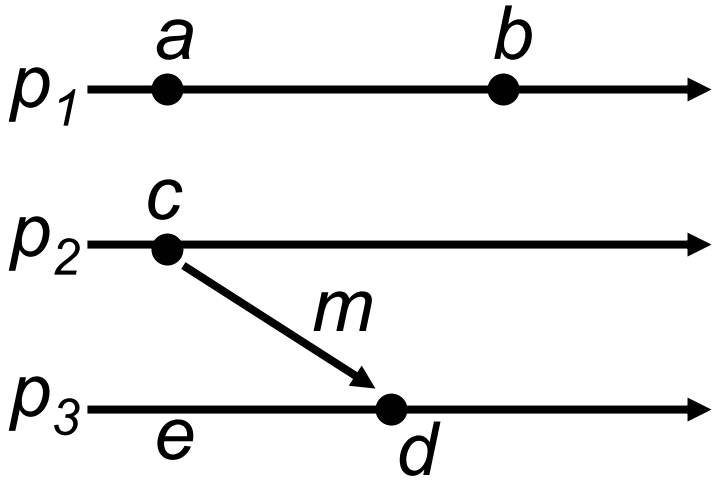
- Warning: heavy notation ahead
  - Only a small part of the class!
- DSs modeled as set of *N* processes or participants *p* residing on *M* processors or sites denoted *s*.
- Sites interconnected by some network, topologies vary
- Evolution of a system modeled by events e^i_p
  - i^{th} event of timeline of each process *p*
  - Omit subscript or superscripts when not ambiguous
- When needed: associate physical timestamps with events: *t(e)* is the real time instant when *e* took place
  - Alternately, t_0, t_a, t_b, …

DS Models (cont.)

- State of a process, *S*, is modified upon the occcurence of each event
- Evolution of state of *S* is its history, *H*
- *H* is an ordered set of tuples: <event, event state>
- A run is an ordered set of events in a process execution, described by a history
- A distributed run is a partially ordered set of events in the execution of several processes

Representing Distributed Computations

- Events at a process can be
  - execution events: internal computations
  - send events: sending a message to another process
  - receive events: receiving a message from another process
- Message exchanges depicted with timelines: e.g.

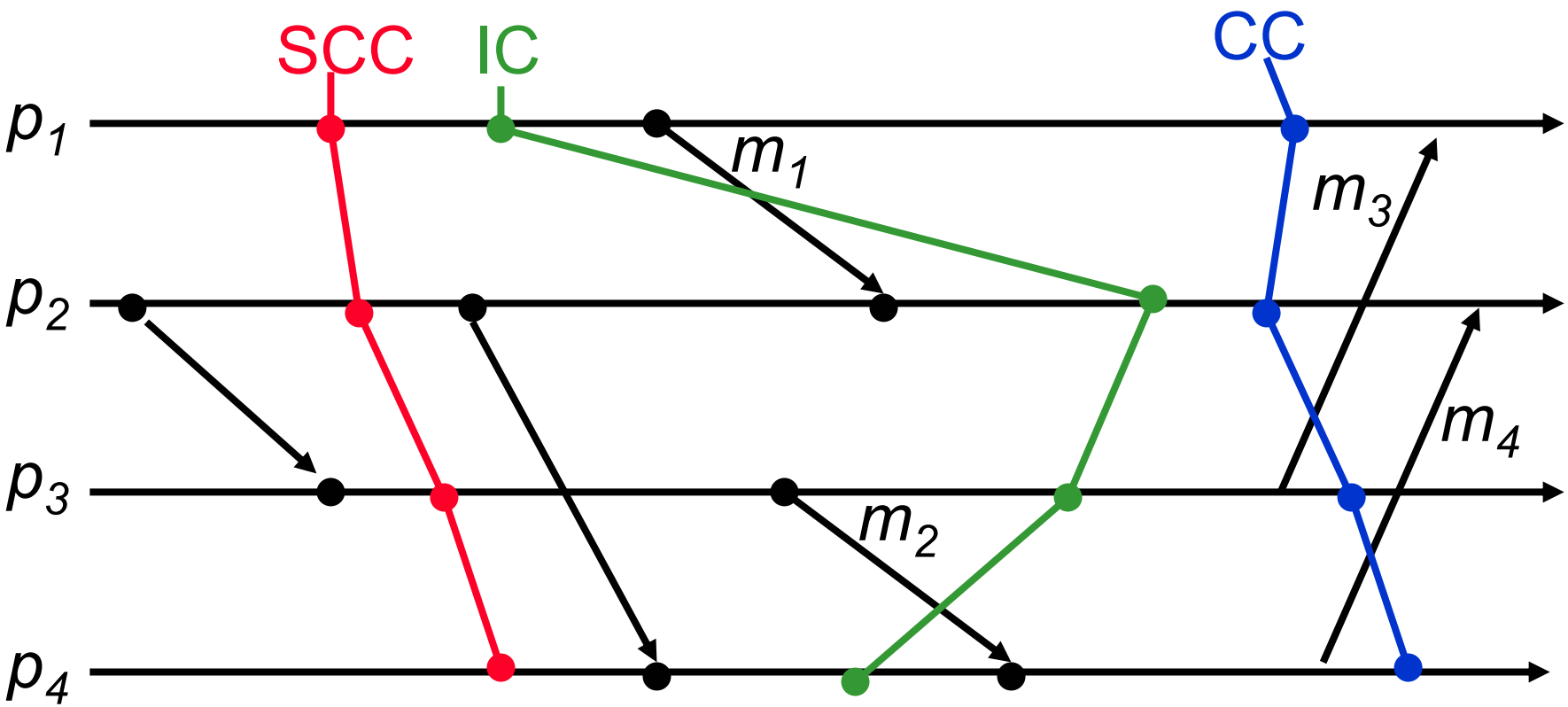


Distributed Systems Foundations: © 2001-2003 Carl Hauser & David E. Bakken. Some figures © 2001 Veríssimo and Rodrigues

Global States

- Sometimes very useful to get a global "picture" of a distributed system
- Global state (GS) of a DS at any point is a vector of its individual process states: $S = \{S_1, S_2, \ldots, S_M\}$
- Two viewpoints of system evolution:
  - Interleaving view: system goes through a succession of states
  - Space-time view: system goes through a partially ordered set of events occuring in several processes in the system
- A cut (in space-time view) is a segment intersecting the timelines of all processes.

Global States (cont.)



- Strongly consistent cut (SCC): faithfully represents GS of the system
- Inconsistent cut (IC): gives invalid picture of GS
- Consistent cut (CC): gives valid but possibly incomplete picture of the GS of the system

DS Properties

- Goal: specify a system with high-level properties
- Safety properties: something bad (wrong events) never takes place
  - Specification form: predicate *P* is true throughout every execution
- Liveness properties: something good (positive event) eventually takes place
  - Specification form: every execution can be extended in a way that makes *P* become true
- "any delivered message is delivered to all correct participants": safety property (NOT!)
- "any message sent is delivered to at least one participant": liveness property
- Timeliness properties relate the times at which events occur – e.g. event a occurs < .1 second after event b.