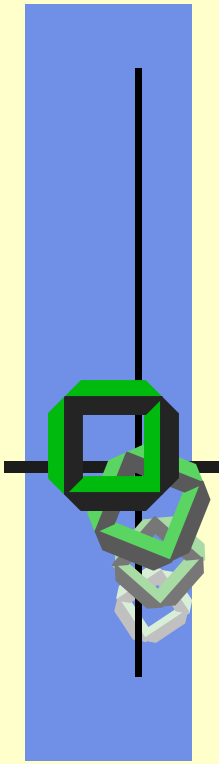


Distributed Systems

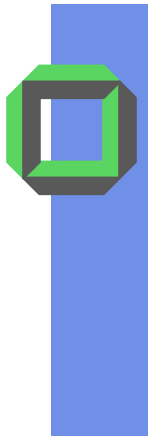
3 Problems & Examples

Inherent Problems, DS Examples

April-27-2009

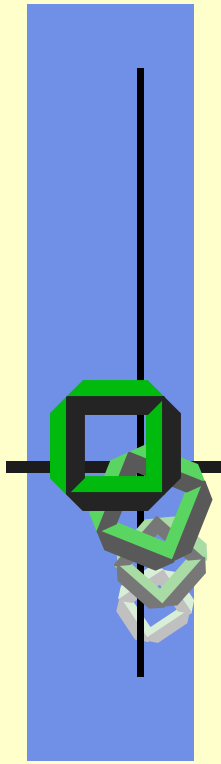
Summer Term 2009

System Architecture Group

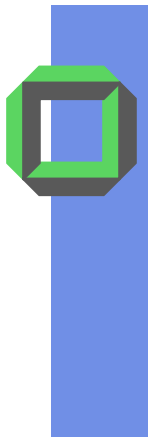


Schedule of Today

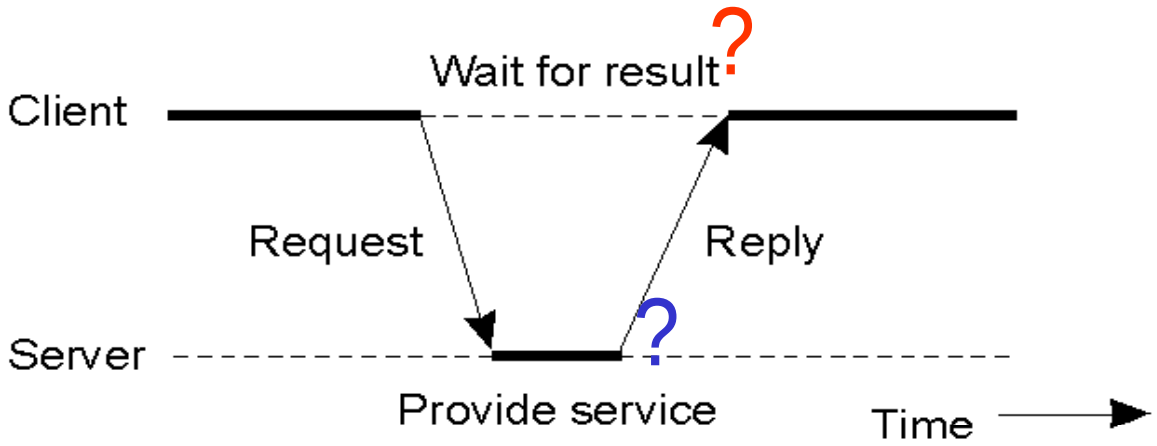
- Inherent Problems
- Conceptual Problems
- Motivation by Examples
- HW Architetures



Inherent Problems

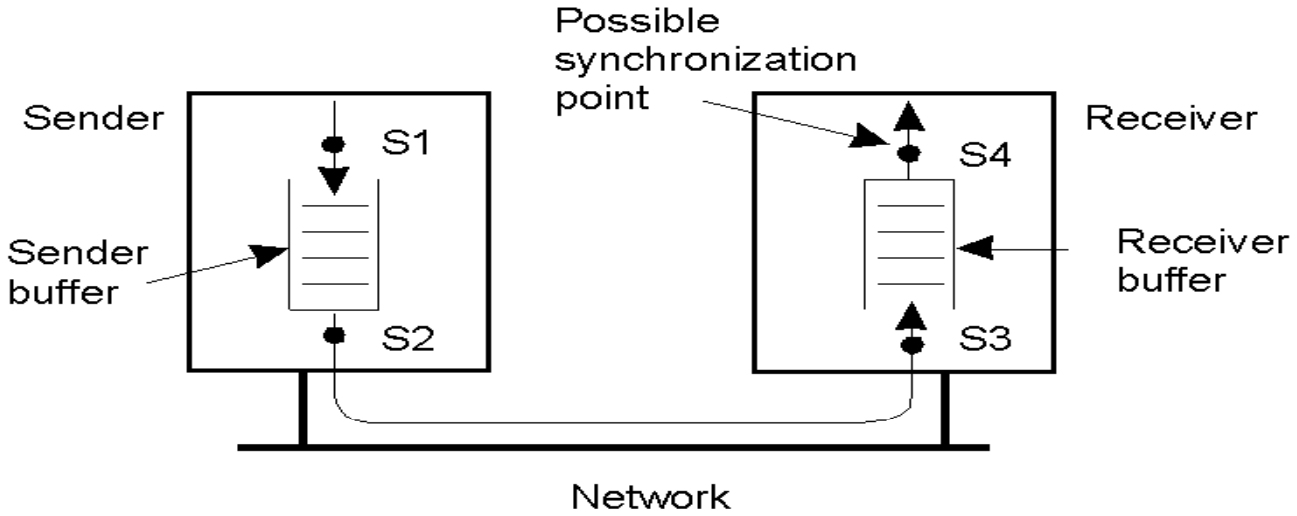


Interaction Problem



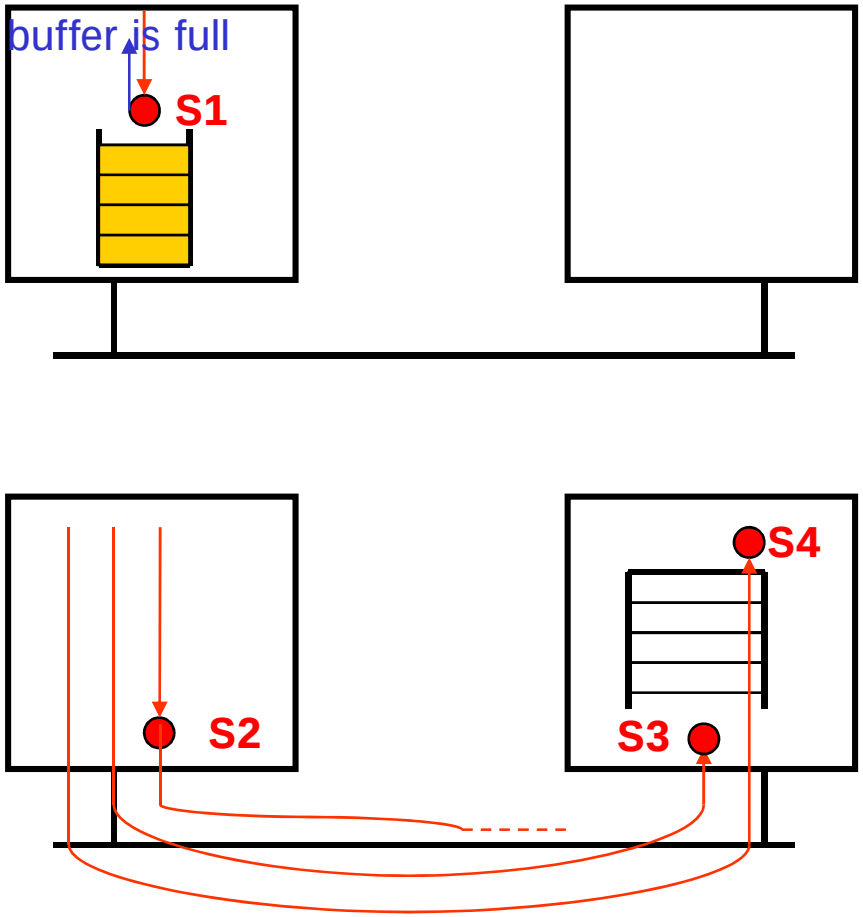
- *What to do if the server does not reply?*
  - Wait forever or wait only some limited time?
  - Send request again?
    - sometimes OK
    - sometimes, severe DS error
- *What can happen if client does not take the reply?*
  - In case of a synchronous reply, server is blocked forever!!

Basic Mechanism: Communication

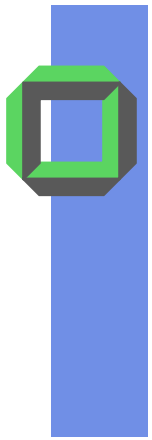


- Three orthogonal design parameters for a simple IPC
  - with or without buffering
    - at sender and/or at receiver site
  - with or without blocking
    - sender and/or receiver
  - reliable or unreliable

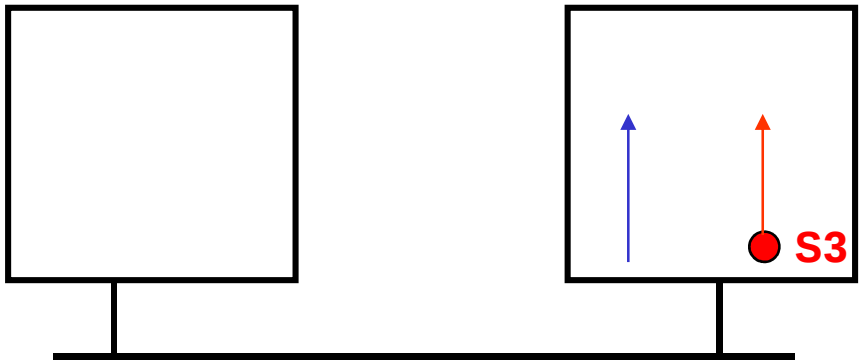
Potential Blocking of Sender



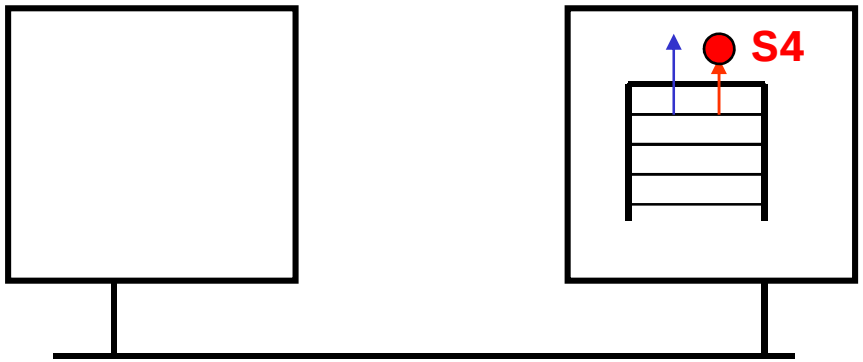
- ∃ sender buffer
  - block sender at S1 if buffer is full, or
  - avoid blocking with a trying send
- ∃ receiver buffer
  - block sender at S2 until message is sent
  - block sender at S3 until message was received at receiver's site
  - Block sender until message was delivered to the receiver process



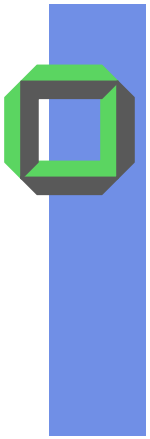
Potential Blocking of Receiver



- ∃ no receiver buffer
  - block receiver at S3
  - avoid blocking with polling until message has arrived



- ∃ receiver buffer
  - block receiver at S4 if buffer is empty
  - avoid blocking with polling until buffer no longer empty



Communication & Reliability

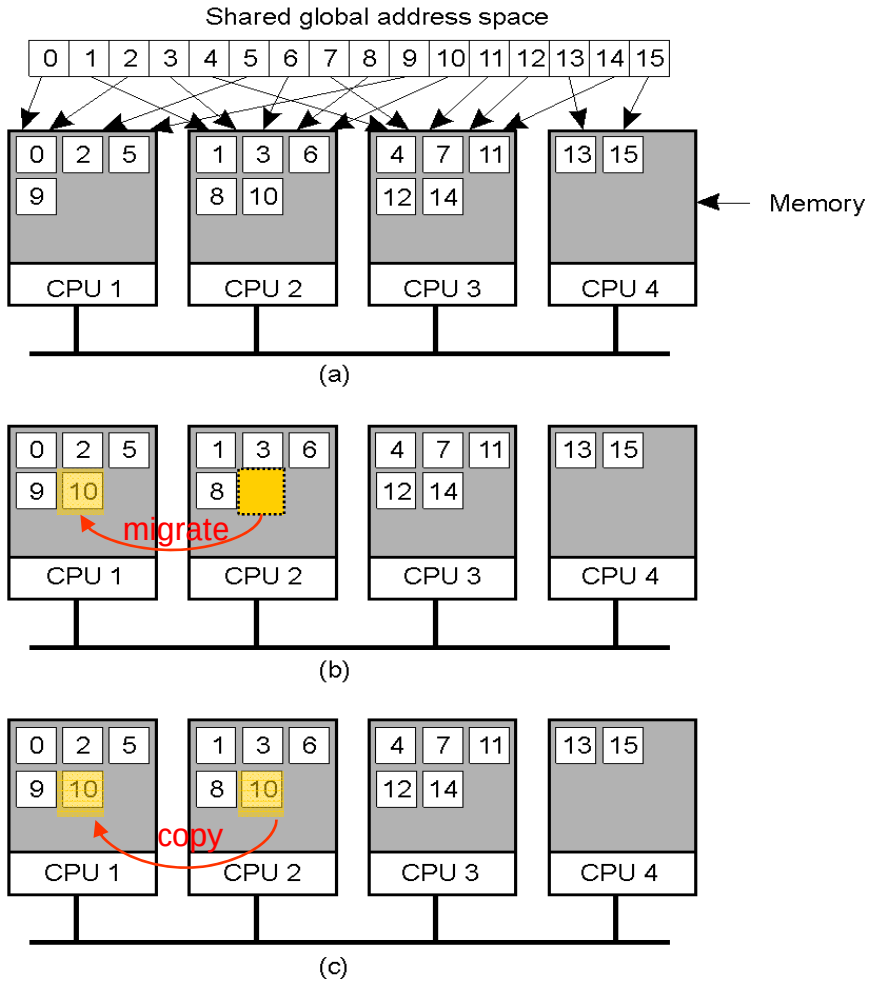
How to deal with communication?

- Reliable communication
  - Sender gets an information, that its message did arrive at the receiver's side
- Unreliable communication
  - Sender has no guarantee whether its message did arrive at the receiver's side

How to combine reliability with the 2 previous discussed IPC design parameters? (assignment, Ch.1 Tanenbaum)

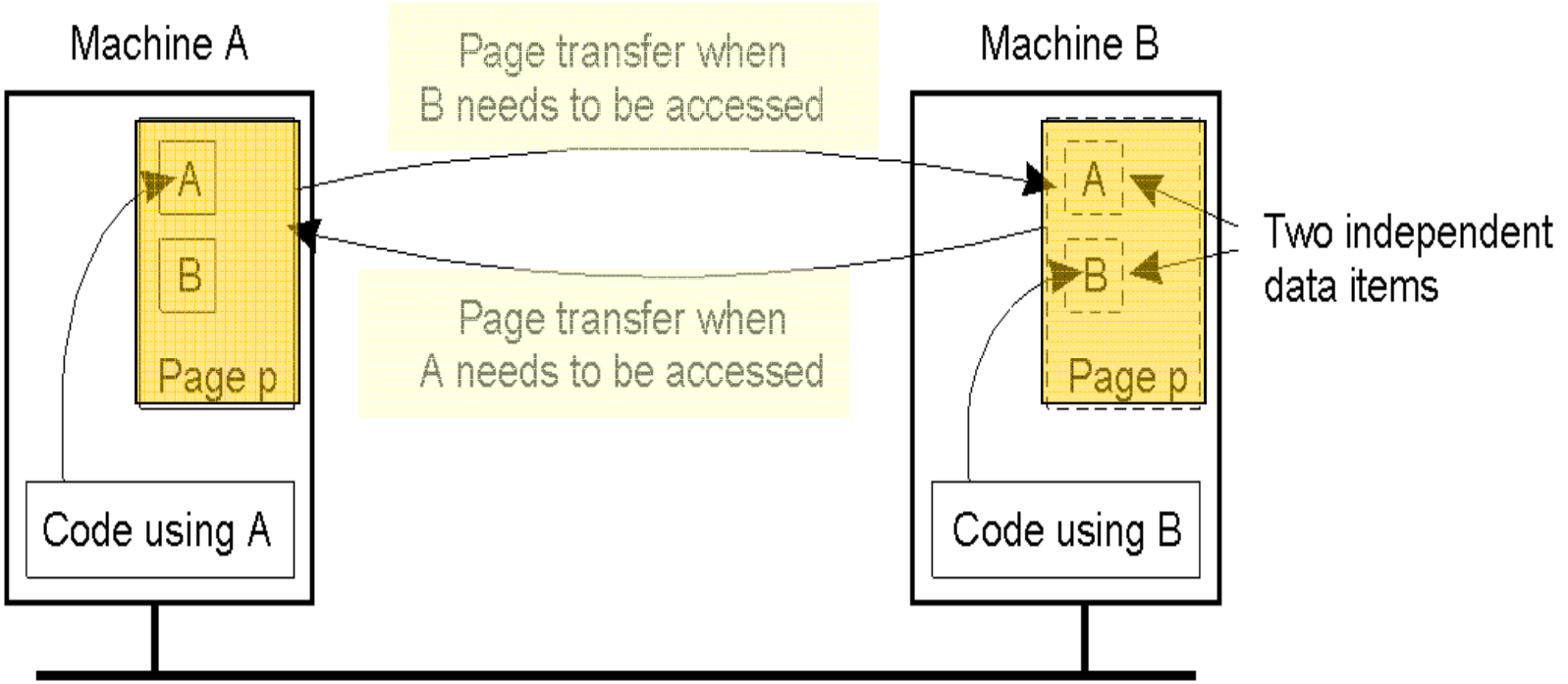
Distributed Shared Memory Systems

- Pages of an address space are distributed over 4 machines
- Situation after CPU 1 references page 10
- Situation if page 10 is read only and replication is used

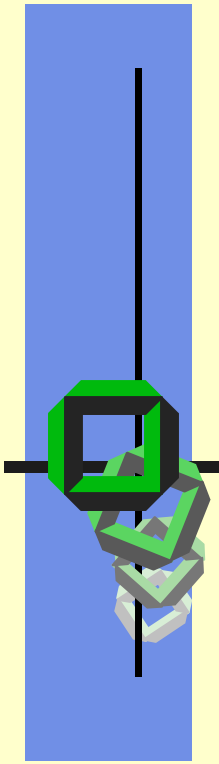


Question: Can we replicate read-write pages, too?

False Sharing in a DSM



- False sharing between 2 loosely coupled KLTs

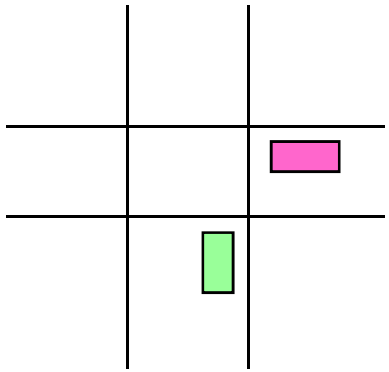


Conceptual Problems

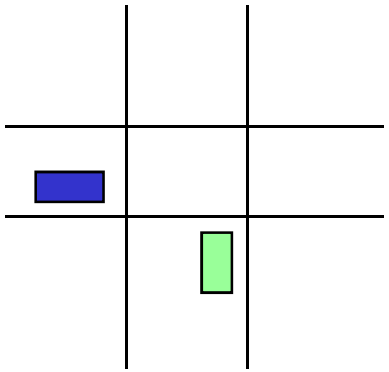
- Phantom Deadlocks
- Clock Synchronization
- Causally Ordered Events
- Covered Channels

Phantom Deadlock

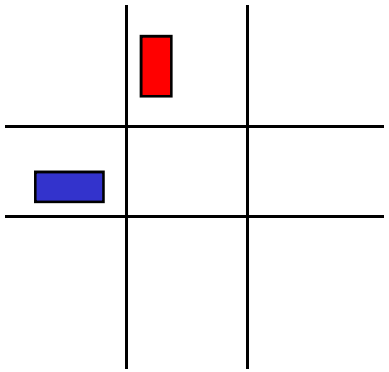
4 isolated views of drivers of the cars S,E,N,W result in:



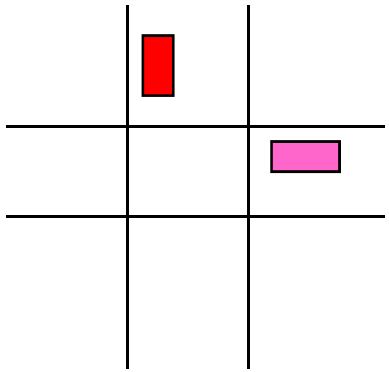
S waits for E



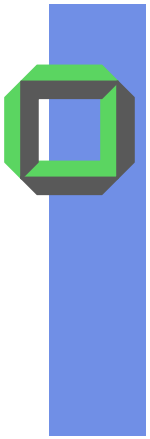
W waits for S



N waits for W



E waits for N

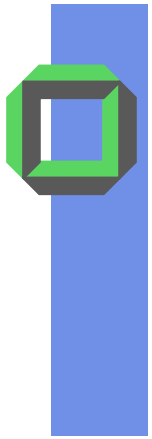


Time in DS

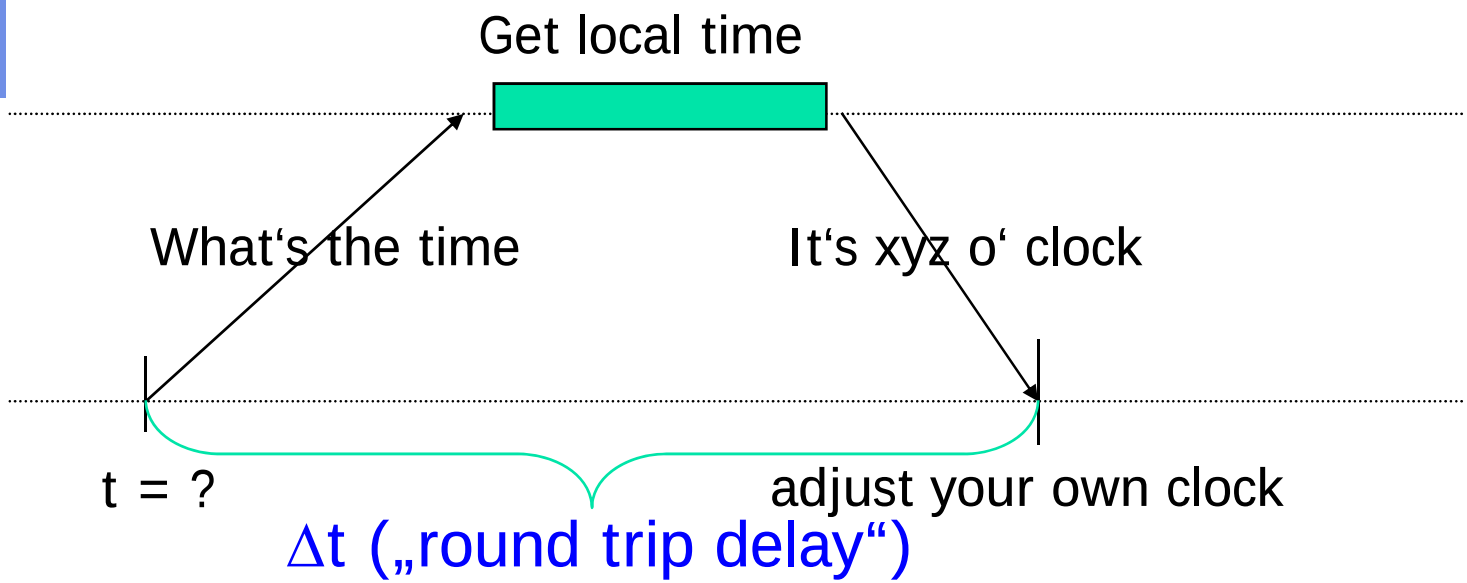
- Absence of a global time
- Every computer has its own local clock
- Assume: these clocks have different times

How to solve this timing problem?

- Very expensive precise physical clocks per node
- Cheaper logical clocks established via software protocols

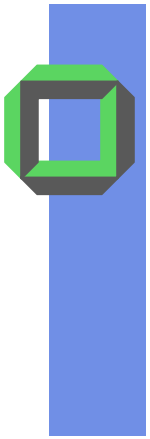


Synchronization of Clocks



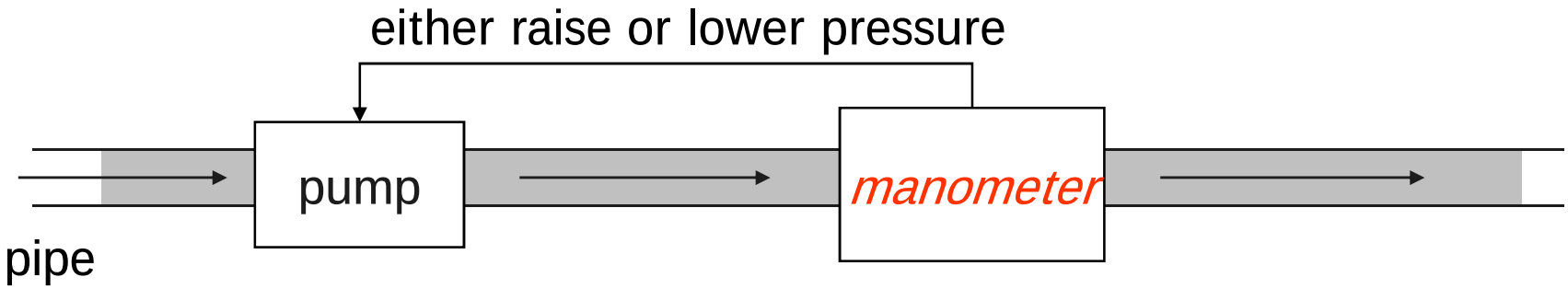
Problems:

- message transfer time is load dependant
- non symmetric message-transfer times
- how to get information about message transfer times?



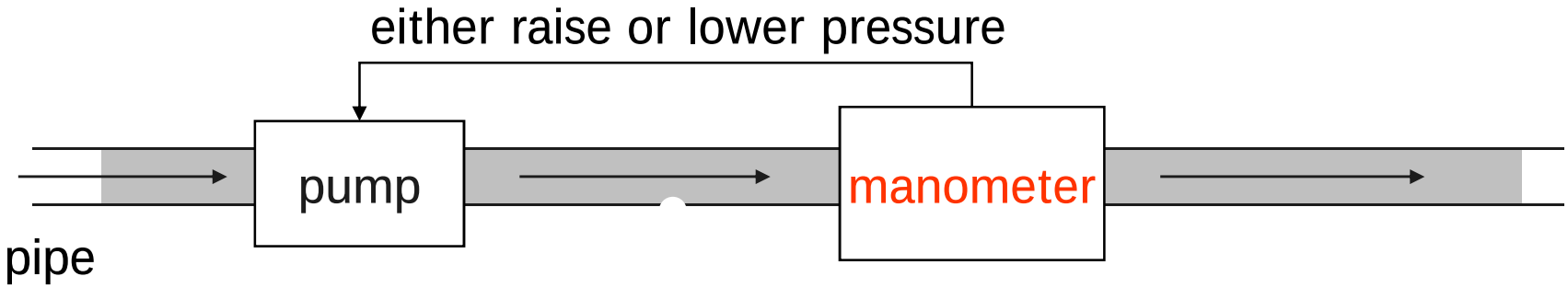
Causal Inconsistent Observations

Desired: Observe origin before its impact

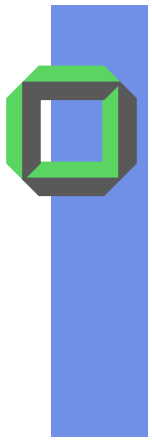


Depending on the value of the manometer
we‘ll accelerate or slow down the pump, but …

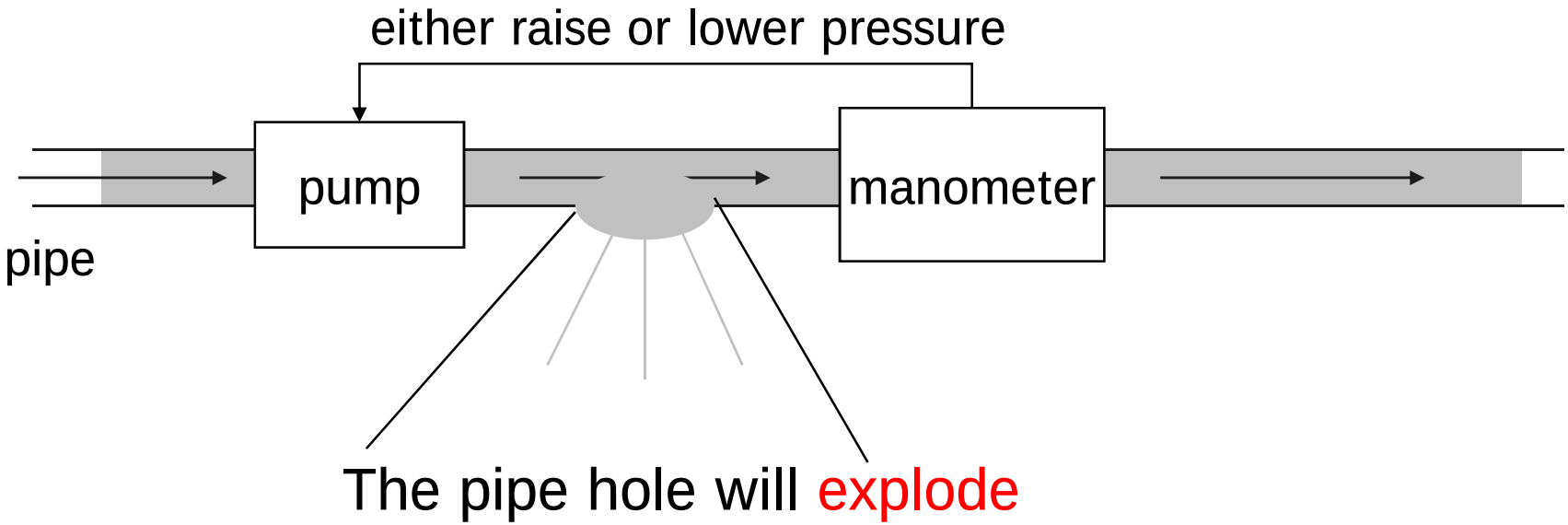
Causal Inconsistent Observations

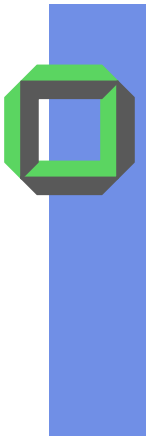


However, if ∃ small hole in the pipe, what will happen?
Manometer will decrease ⇒
we will accelerate the pump, and then ...



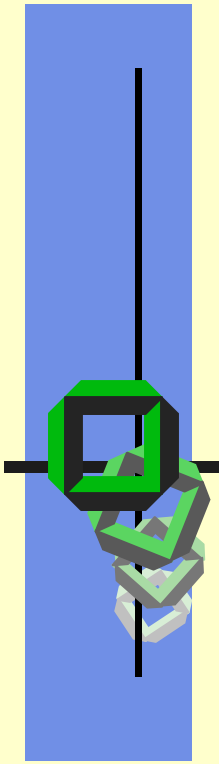
Causal Inconsistent Observations





Summary: Rules of Thumbs

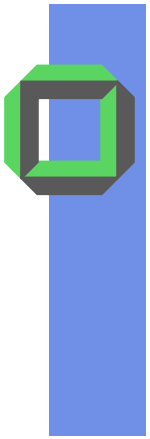
- Trade-offs:
  - Some challenges of DS provide conflicting requirements, e.g. scalability and performance
- Separation of Concerns:
  - Split problem into individual concerns and address each problem separately
- End-to End argument:
  - Often a reliable communication can only be implemented at application level
- Policy v. Mechanism:
  - System architects should implement base mechanisms that allow flexible policies⇒ AVOID: BUILT-IN POLICIES



Motivation by Example

Air Traffic Control

(slides from K. Birman,
chief architect of ISIS &
author of “Reliable DS”)



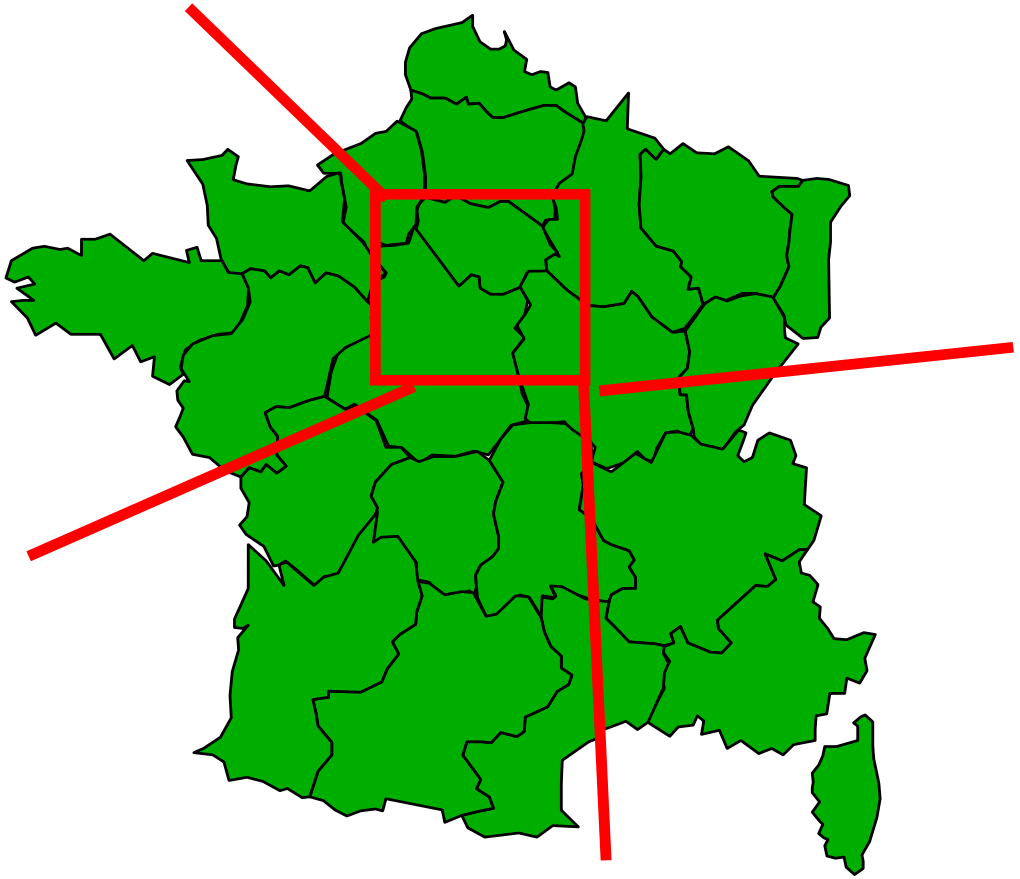
Air Traffic Control using Web Techn.

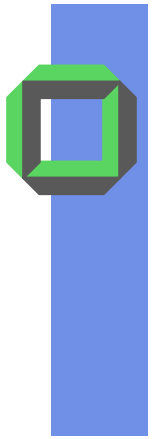
- Assume a “private” network
- Web browser could easily show planes, natural for controller interactions
- *What “properties” would the system need?*
  - Clearly need to know that trajectory and flight data is current and consistent
  - We expect it to give sensible advice on routing options (e.g. do not propose dangerous routes)
  - Continuous availability is vital: we need zero downtime
  - Expect a soft form of real-time responsiveness
  - Security and privacy also required (post 9/11!)

ATC Systems divide Country up

e.g. France

One major Design approach: “Divide an conquer”



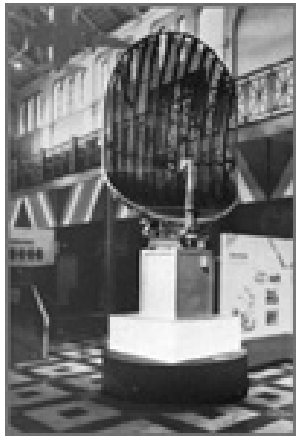


Issues with Old Systems

- Overloaded computers that often crashed
  - Attempt to build a replacement system failed (1994!!!)
- Getting slow as volume of air traffic rises
- Inconsistent displays a problem resulting in
  - phantom planes
  - missing planes
  - stale information
- Some major outages recently (and some near-miss stories associated with them)
  - TCAS saved the day: collision avoidance system of last resort... and it works....

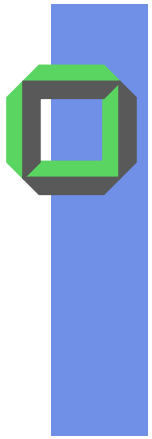
ATC News

- The FAA[1] recognized the need for further modernization of air traffic control, and in July 1988, selected IBM to develop the new multi-billion-dollar Advanced Automation System (AAS) for the Nation's en route ATC centers. AAS would include controller workstations, called "sector suites," that would incorporate new display, communications and processing capabilities. The system would also include new computer hardware and software to bring the air traffic control system to higher levels of automation.



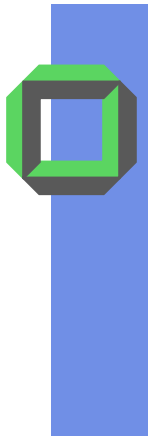
This ASR-1 (Airport Surveillance Radar) antenna was part of an air traffic system used beginning in the early 1950s.

[1]Federal Aviation Administration



ATC News

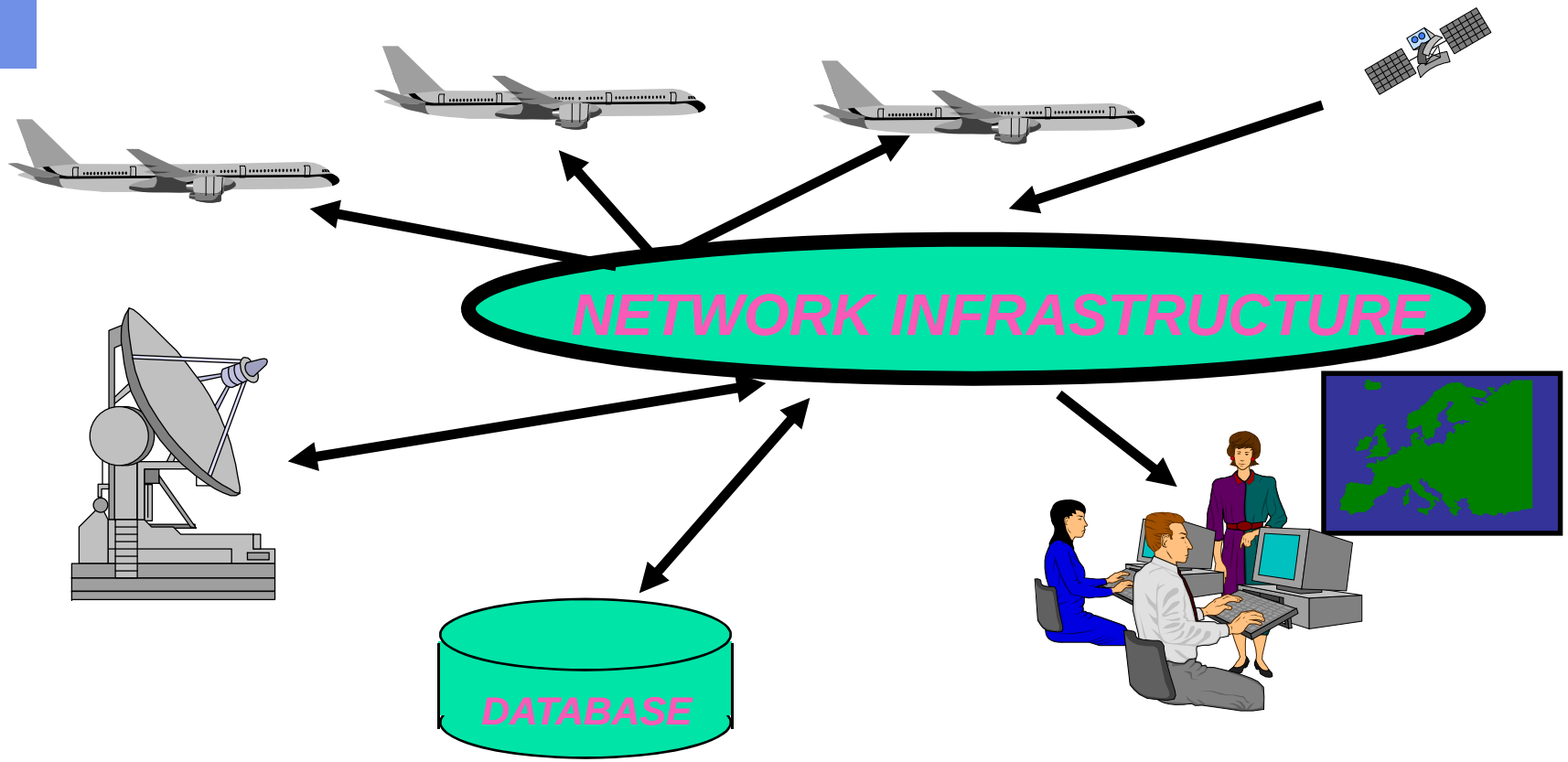
- In December 1993, the FAA reviewed its order for the planned AAS. IBM was far behind schedule and had major cost overruns.
- In 1994 the FAA simplified its needs and picked new contractors. The revised modernization program continued under various project names. Some elements met further delays.
- In 1999, controllers began their first use of an early version of the Standard Terminal Automation Replacement System, which included new displays and capabilities for approach control facilities. During the following year, FAA completed deployment of the Display System Replacement, providing more efficient workstations for en route controllers.

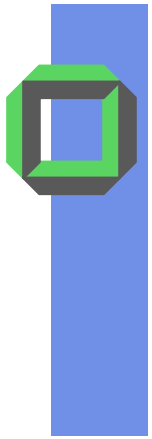


Concept of IBM's 1994 System

- Replace video terminals with workstations
- Build a highly available RT-system that guarantees no more than 3 seconds downtime per year
- Offer better user interface to ATC controllers, with intelligent course recommendations and warnings about future course changes that will be needed

ATC Architecture

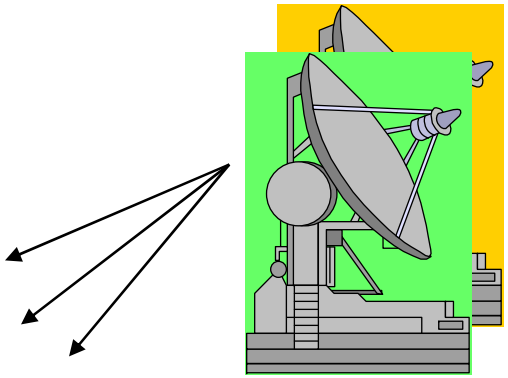




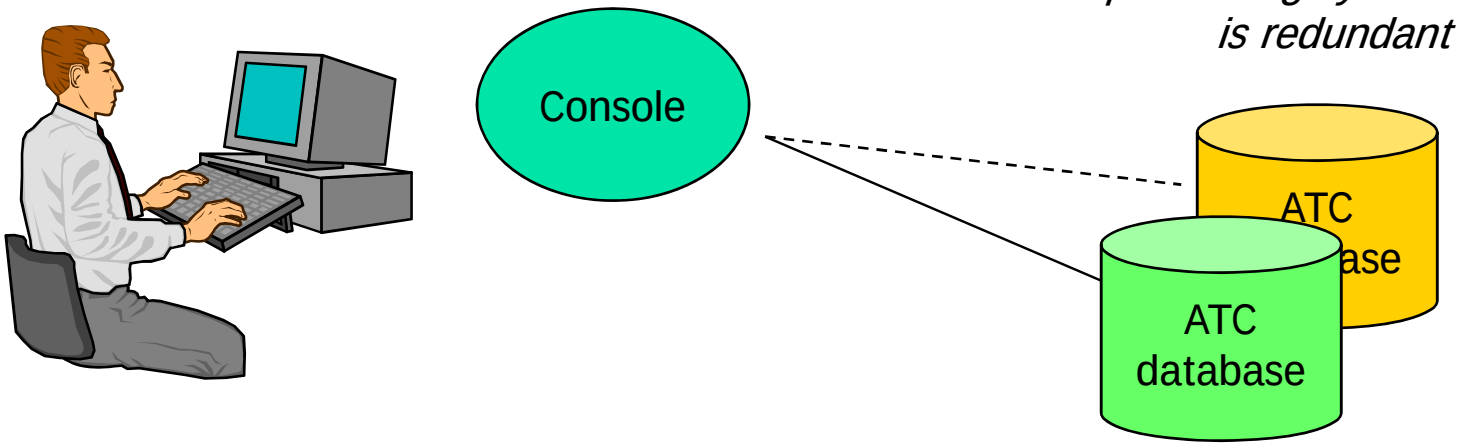
So… how to build it?

- In fact IBM project was just one of two at the time; the French had one too
  - IBM approach was based on lock-step replication
    - Replace **every major component** of the system with a fault-tolerant component set
    - Replicate entire programs (“state machine” approach)
  - French approach used replication selectively
    - As needed, replicate specific data items
    - Program “hosts” a data replica but isn’t itself replicated

IBM: Independent Consoles... backed by Ultra-Reliable Components

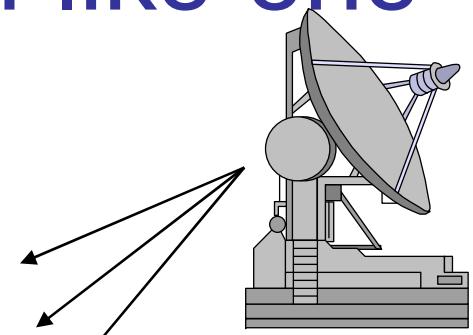


Radar processing system is redundant



ATC database is really a high-availability cluster

France: Multiple Consoles... but in some ways they function like one



Radar updates sent with hardware broadcasts

Console A

Console B

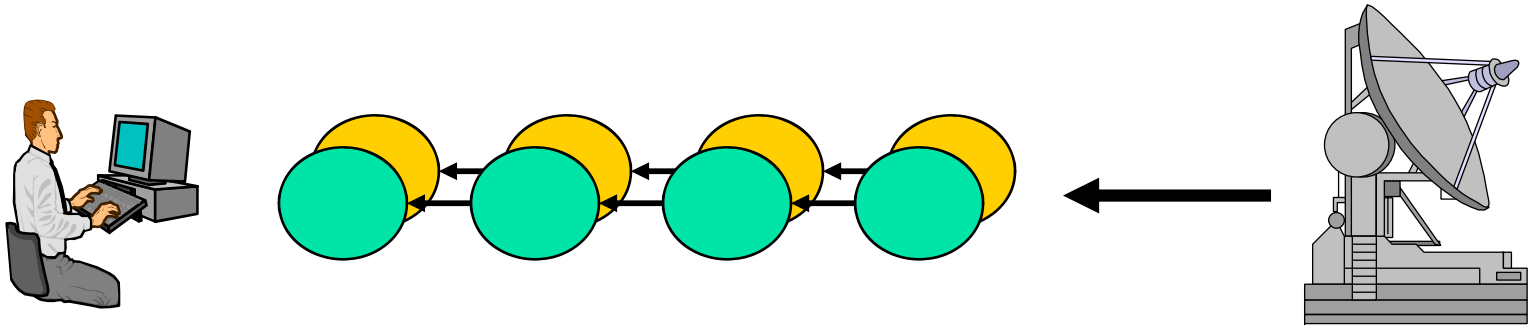
Console C

ATC database

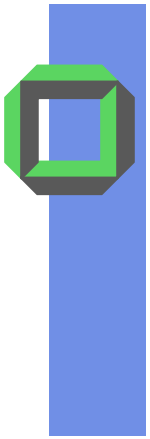
ATC database only sees one connection

Different Emphasis

- IBM imagined pipelines of processing with replication used throughout. “Services” did much of the work.

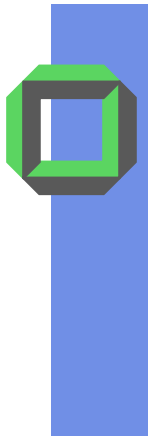


- French imagined selectively replicated data, for example “list of planes currently in sector A.17”
  - E.g. controller interface programs could maintain replicas of certain data structures or variables with system-wide value
  - Programs did computing on their own helped by databases



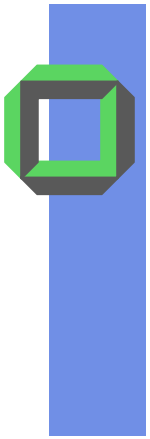
Other Technologies Used

- Both used standard off-the-shelf workstations (easier to maintain, upgrade, manage)
  - IBM proposed their own software for fault-tolerance and consistent system implementation
  - French used ISIS software developed at Cornell
- Both developed fancy graphical user interface much like the Web, pop-up menus for control decisions etc.
- Both used state-of-the-art “cleanroom” development techniques



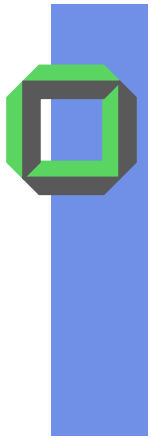
IBM Project: Another Fiasco

- IBM was unable to implement their fault-tolerant software architecture
- Problem was much harder than they expected.
  - Even a non-distributed interface turned out to be very hard, major delays, scaled back goals
  - And performance of the replication scheme turned out to be terrible for reasons they didn't anticipate
- The French project was a success and never even missed a deadline... In use today.



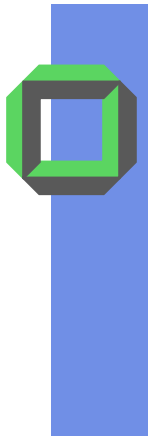
Where did IBM go wrong?

- Their software “worked” correctly
  - The replication mechanism wasn’t flawed, although it was much slower than expected
- But somehow it didn’t fit into a comfortable development methodology
  - Developers need to find a good match between their goals and the tools they use
  - IBM never reached this point
- The French approach matched a more standard way of developing applications



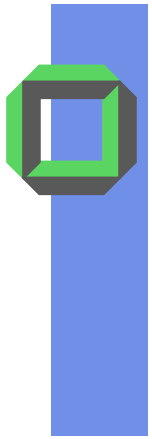
The French ATC System

- Teams of 3-5 air traffic controllers on a cluster of desktop consoles
- 50-200 of these console clusters in an air traffic control center
  - Radar Image
  - Weather Alert
  - Track Updates
  - Updates to Flight Plans
  - Console to Console State Updates
  - System Management and Monitoring
  - ATC center to center Updates
- Multicast ubiquitous...



Two Kinds of Multicast

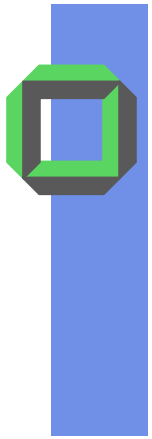
- Virtually Synchronous Multicast: very reliable, not particularly fast
- Unreliable Multicast: very fast, not particularly reliable
- Nothing in between!



Two Kinds of Subsystems

- Category 1: Complete reliability (virtual synchrony) e.g: routing decisions

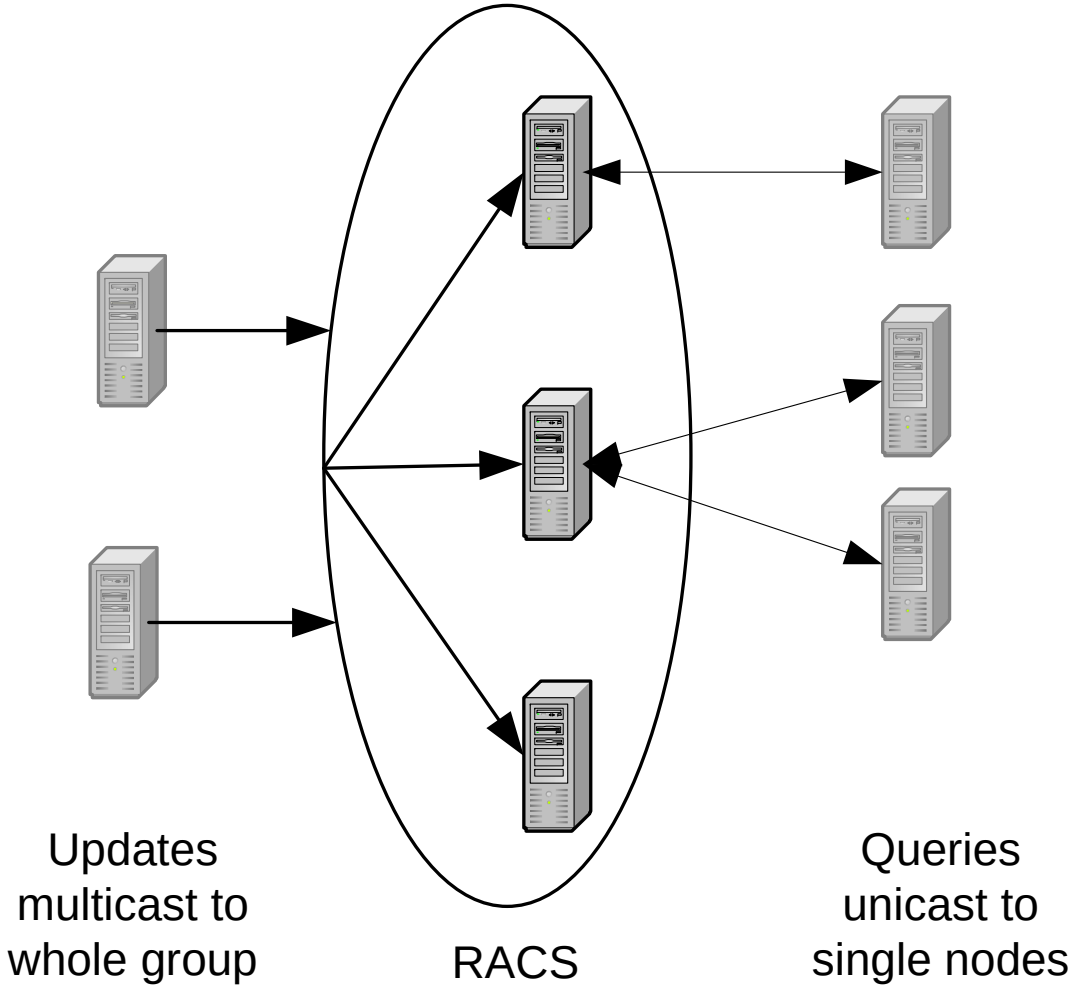
- Category 2: Careful application design + natural hardware properties + management policies. e.g: radar

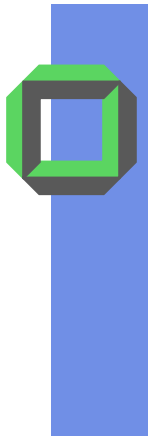


Multicast in the French ATC

- Engineering Lessons:
  - Structure application to tolerate partial failures
  - Exploit natural hardware properties
- *Can we generalize to modern systems?*
- Research Direction: Time-Critical Reliability
  - *Can we design communication primitives that encapsulate these lessons?*

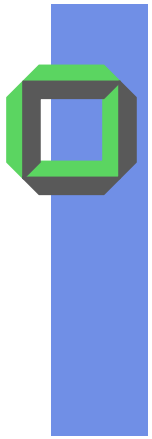
Anatomy of Cloned Service





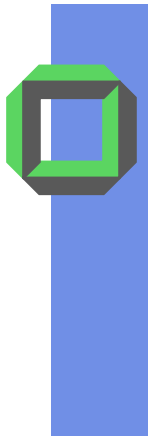
Examples of Mission-Critical Appl.

- Banking, stock markets, stock brokerages
- Heath care, hospital automation
- Control of power plants, electric grid
- Telecommunications infrastructure
- Electronic commerce and electronic cash on the Web (very important emerging area)
- Corporate “information” base: a company’s memory of decisions, technologies, strategy
- Military command, control, intelligence systems



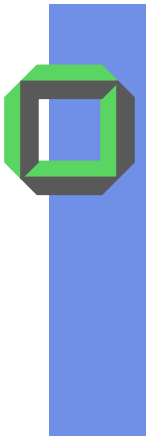
We depend on Reliable DS

- If these critical systems don’t work
  - when we need them
  - correctly
  - fast enough
  - securely and privately
- … then revenue, health and safety, and national security may be at risk!



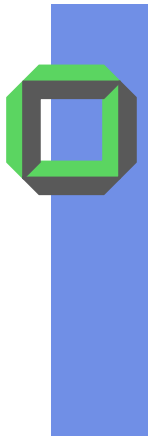
Critical Needs of Critical Applications

- Fault-tolerance: many flavors
  - Availability: System is continuously “up”
  - Recoverability: Can restart failed components
- Consistency:
  - Actions at different locations are consistent with each other.
  - Sometimes use term “single system image”
- Automated Self-Management:
  - Adaptivity
  - Load Control
- Security, privacy, etc....:
  - Vital, but not our topic in this course



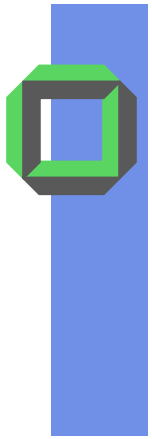
So what makes it hard?

- ATC example illustrated a core issue
- Existing platforms
  - Lack automated management features
  - Handle errors in ad-hoc, inconsistent ways
  - Offer one form of fault-tolerance mechanism (transactions), and it isn't compatible with high availability
- Developers often forced to step outside of the box... and might stumble.
  - But why don't platforms standardize such things?



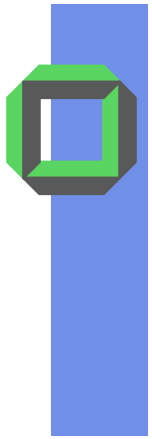
Generalized End-to-End View?

- Low-level mechanisms should focus on speed, not reliability
- The application should worry about “properties” it needs
- OK to violate the E2E philosophy if E2E mechanism would be much slower



E2E is visible in J2EE and .NET

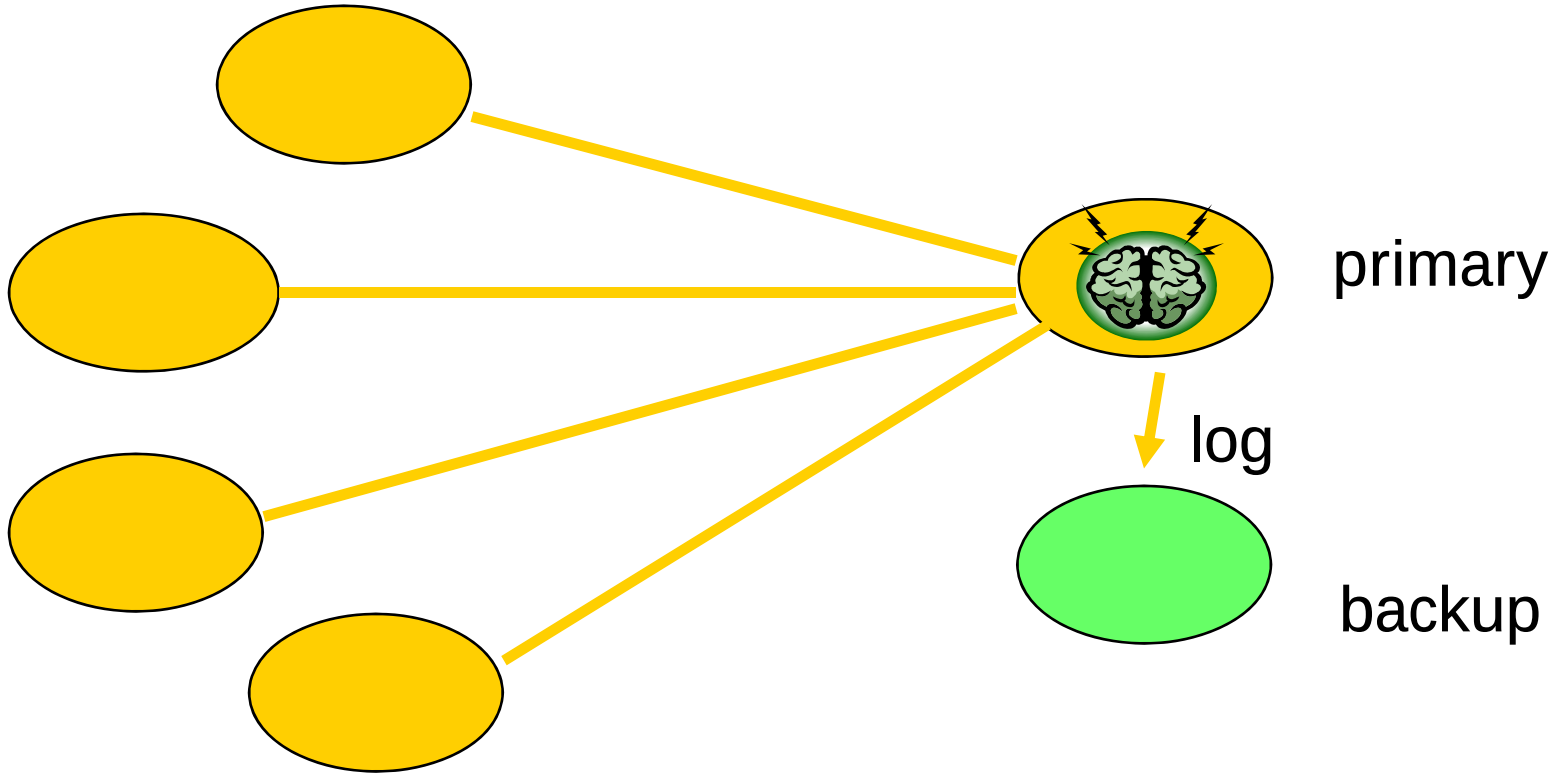
- If something fails, these technologies report timeouts, but
  - they also report timeouts when nothing has failed
  - when they report timeouts, they don’t tell you what failed
  - they don’t offer much help to fix things up after the failure, either



Example: Server Replication

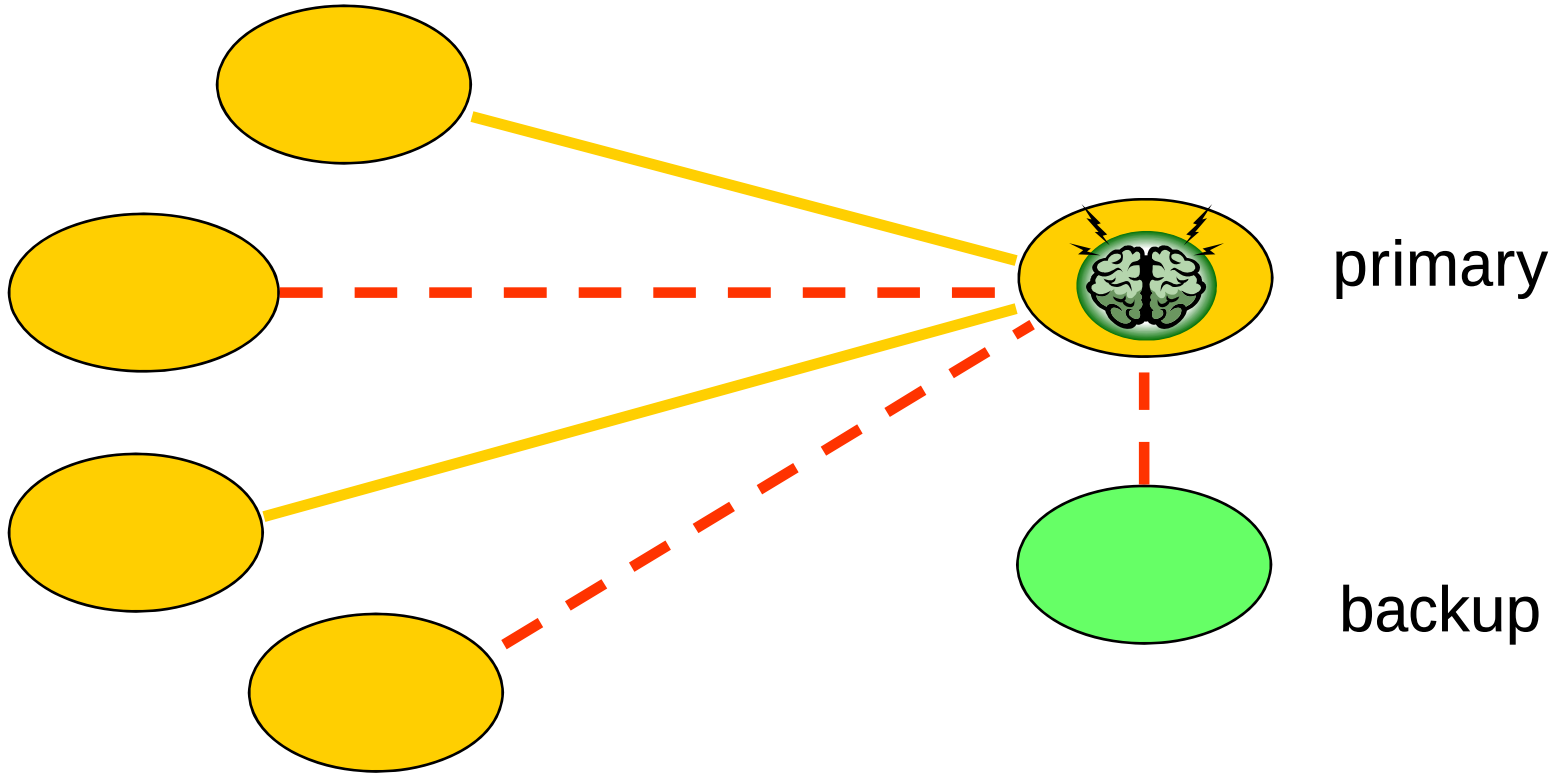
- Suppose that our ATC needs a highly available server.
- One option: “primary/backup”
  - We run two servers on separate platforms
  - The primary sends a log to the backup
  - If primary crashes, the backup soon catches up and can take over

Split Brain Syndrome...



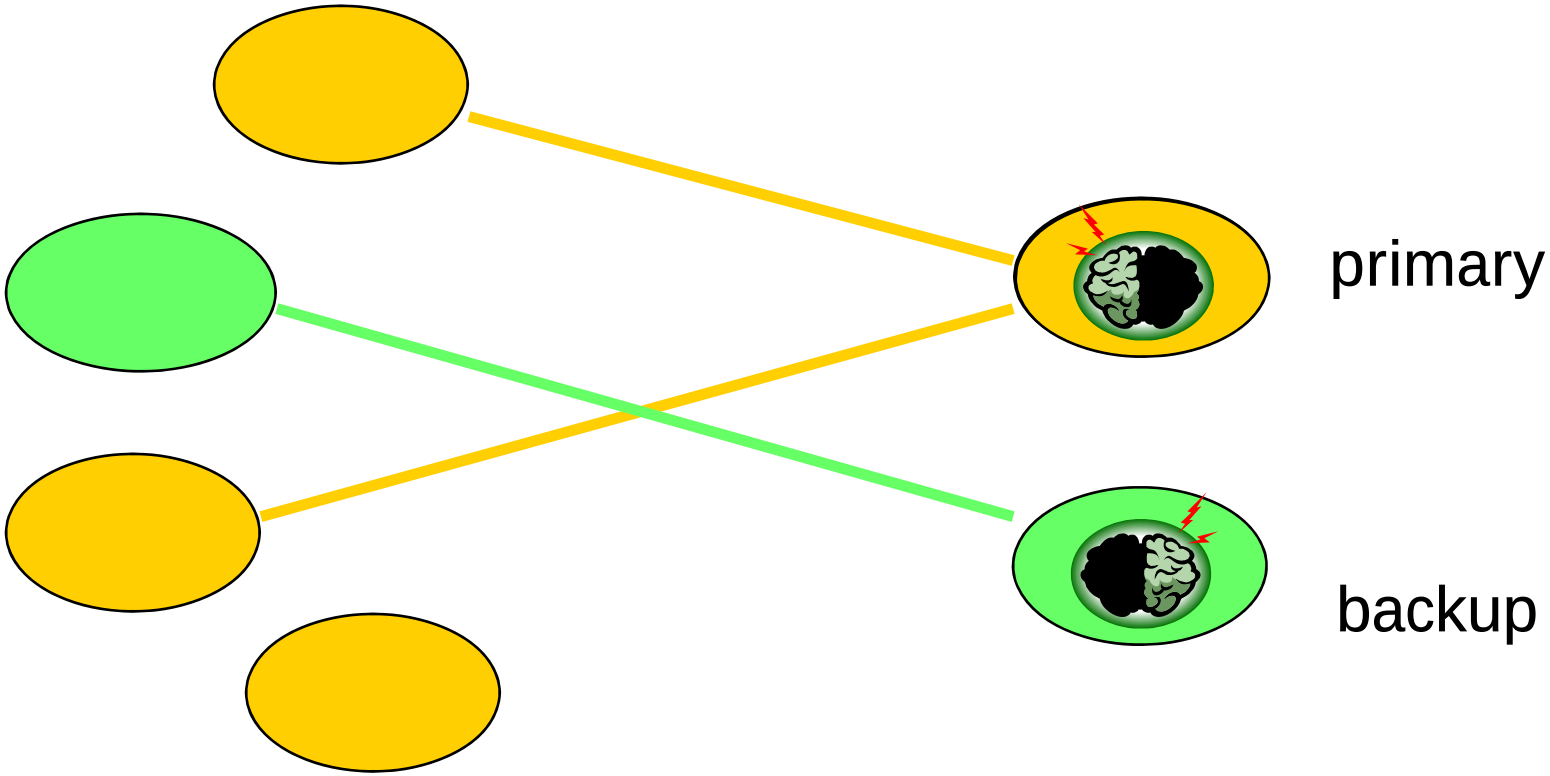
Clients initially connected to primary, which keeps backup up to date. Backup collects the log

Split brain Syndrome...

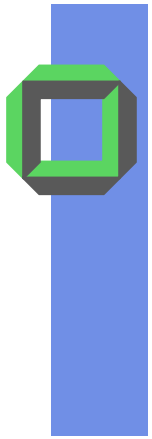


Transient problem causes some links to break but not all.
Backup thinks it is now primary, primary thinks backup is dowr

Split brain Syndrome

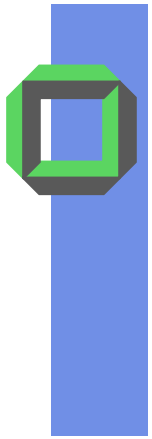


Some clients still connected to primary, but one has switched to backup and one is completely disconnected from both



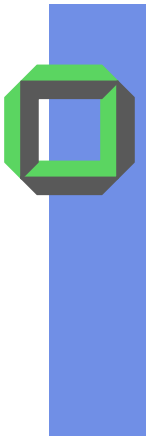
Implications?

- Air Traffic System with a split brain could malfunction disastrously!
  - For example, suppose the service is used to answer the question “is anyone flying in such-and-such a sector of the sky”
  - With the split-brain version, each half might say “nope”... in response to different queries!



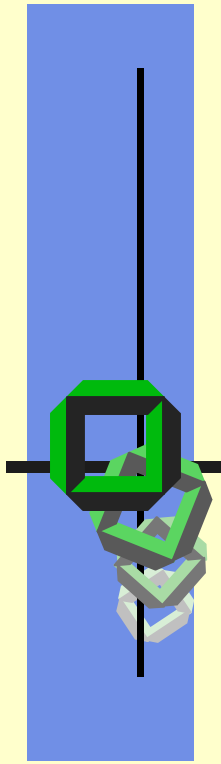
Can we fix this Problem?

- No, if we insist on an end-to-end solution
  - We'll look at this issue later in the class
  - But the essential insight is that we need some form of "agreement" on which machines are up and which have crashed
  - Can't implement "agreement" on a purely 1-to-1 (hence, end-to-end) basis.
    - Separate decisions can always lead to inconsistency
    - So we need a "membership service"... and this is fundamentally not an end-to-end concept!



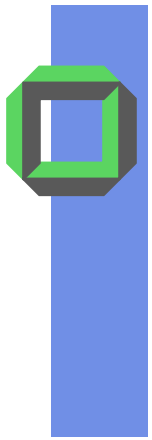
Can we fix this Problem?

- Yes, many options, once we accept this
  - Just use a single server and wait for it to restart
    - This is common today, but too slow for ATC
  - Give backup a way to physically “kill” the primary, e.g. unplug it
    - If backup takes over… primary shuts down
  - Or require some form of “majority vote”
    - maintains agreement on system status
- Bottom line? You need to anticipate the issue… and to implement a solution.

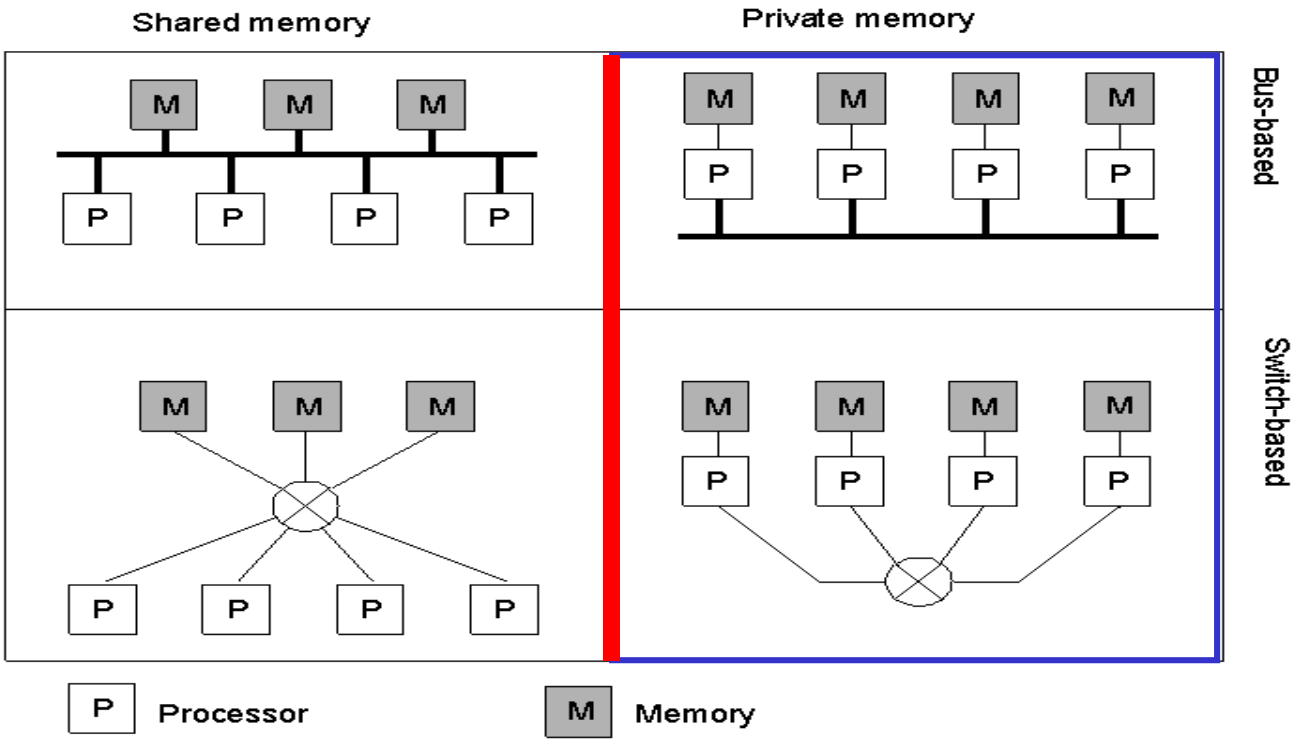


HW Architectures

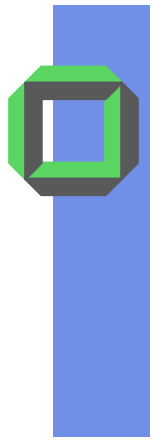
- Local versus Distributed Systems
- Range of DS
- Architectural Style



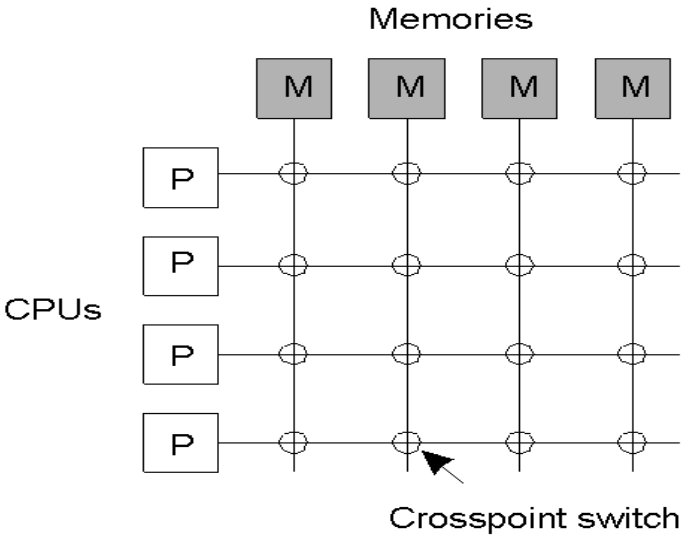
Local versus Distributed Systems



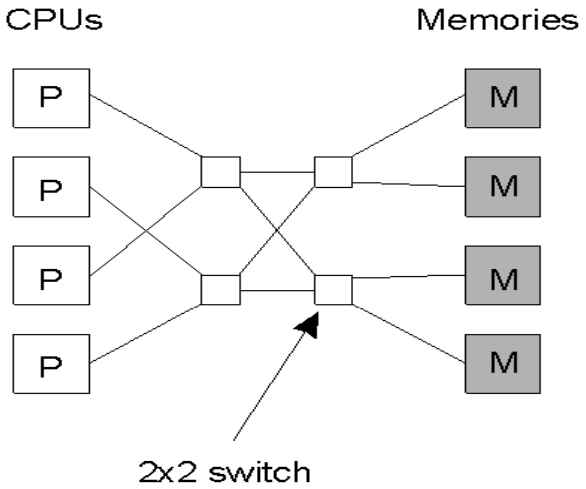
- Orthogonal HW design parameters
  - shared/private memory
  - bus-based/switch-based interconnection



Switched Multiprocessor



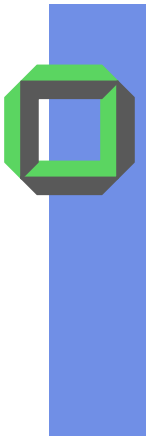
(a)



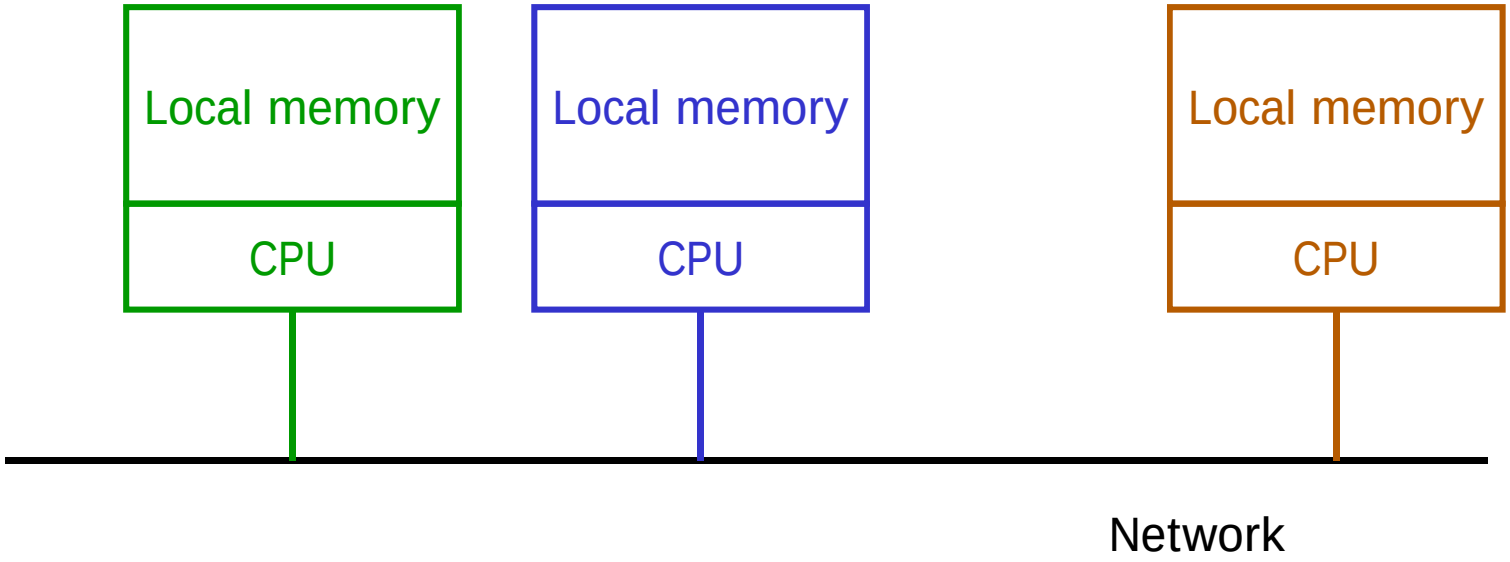
(b)

(a) Low scalability due to *cost* (crossbar)

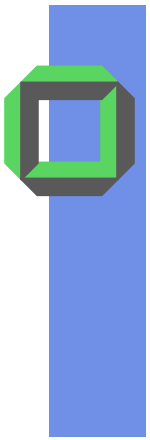
(b) Low scalability due to *switching latency*
alternative approach: *NUMA architecture*



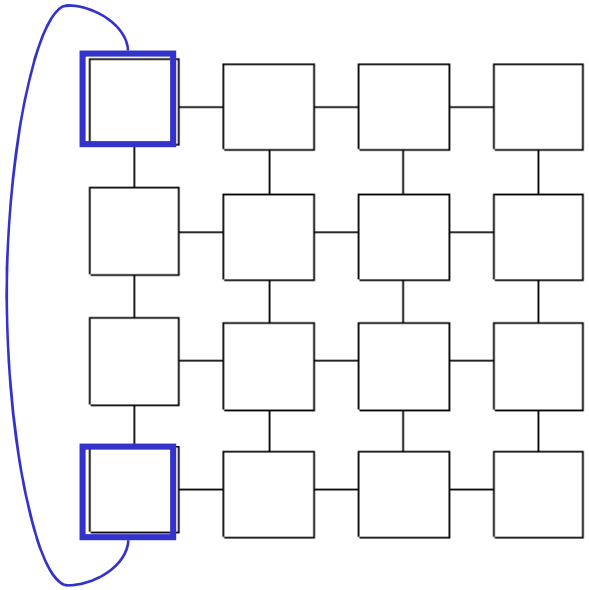
Bus-Based Multi-Computers



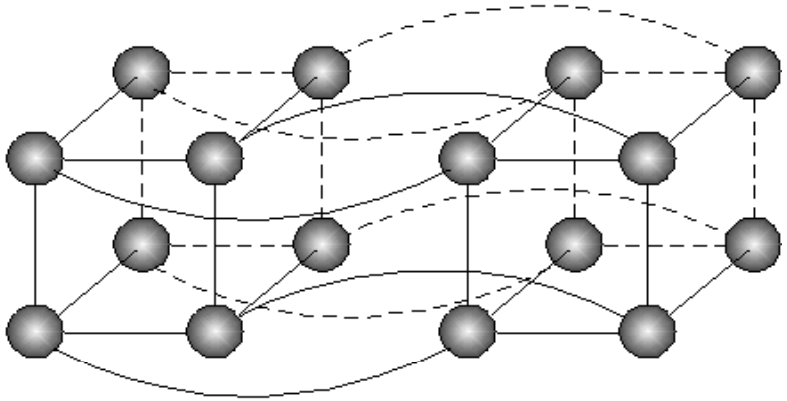
- Similar to a bus-based multi-processor, but hopefully with less network traffic



Homogeneous Multi-Computer



(a)



(b)

- Grid (well suited to meet 2-dimensional problems, e.g. analyzing photographs)
- Torus (~ Grid + interconnections of border nodes)
- Hypercube

  Switched MC between MPPs (> 1000 CPUs) or Cluster of WSs

Typical Range of DS

WAN Wide Area Network (e.g. Internet, worldwide)

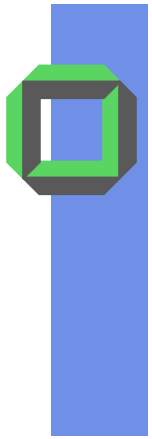
MAN Metropolitan Area Network (within about 10-30 km)

LAN Local Area Network
(company or university, within about 1-3 km)

SAN Storey/Storage* Area Network
(within a room etc.)

SAN = System Area Network

*Stockwerk



Preview

- SW Architectures
- Architectural Styles
- System Architectures
  - Centralized SA
  - Decentralized SA
  - Hybrid SA
- SA versus Middleware
  - Interceptors
  - Approach to Adaptive Software
- Self Management in DS
  - Feedback Control System
  - Astrolabe
  - Globule
  - Jade