L6 – DISTRIBUTED PROCESS MANAGEMENT

BMCS3003 DISTRIBUTED SYSTEMS AND PARALLEL COMPUTING

CONTENTS

INTRODUCTION

- Need for good resource allocation scheme for DS
- Distributed scheduler:

  A resource management component of a distributed operating system that focuses on judiciously and transparently redistributing the load of the system among the computers such that the overall performance of a system is maximized.

- More suitable for LANs than WANs

PROCESS MIGRATION

- Transfer of sufficient amount of the state of a process from one computer to another
- The process executes on the target machine

MOTIVATION

- Process migration is desirable in distributed computing for several reasons including:
  - Load sharing
  - Communications performance
  - Availability
  - Utilizing special capabilities

LOAD SHARING

- Move processes from heavily loaded to lightly load systems
  - Significant improvements are possible
  - Must be careful that the communications overhead does not exceed the performance gained.

COMMUNICATIONS PERFORMANCE

- Processes that interact intensively can be moved to the same node to reduce communications cost
- May be better to move process to the data than vice versa
  - Especially when the data is larger than the size of the process

AVAILABILITY AND SPECIAL CAPABILITIES

- Availability
  - Long-running process may need to move because of faults or down time
  - OS must have advance notice of fault
- Utilizing special capabilities
  - Process can take advantage of unique hardware or software capabilities

MIGRATION ISSUES

- For process migration to work we need to satisfy a few issues:
  - Who initiates the migration?
  - What is involved in a Migration?
  - What portion of the process is migrated?
  - What happens to outstanding messages and signals?

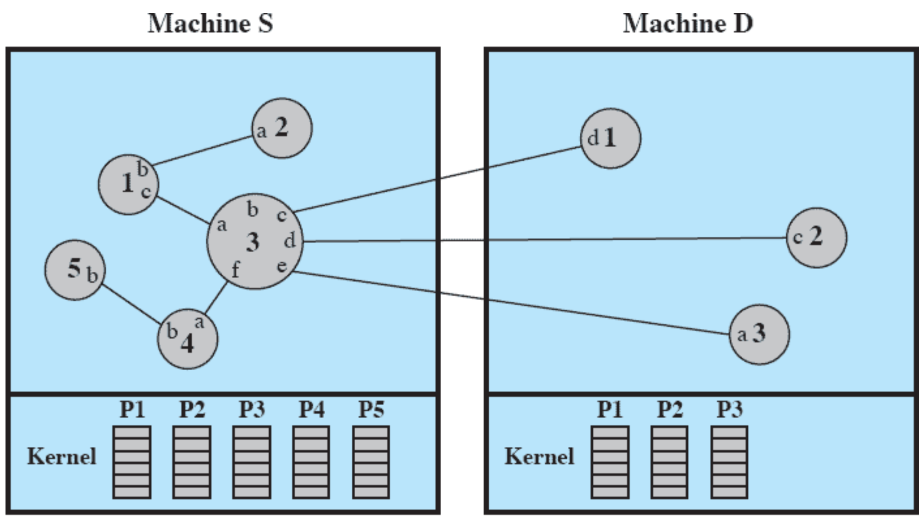
WHO INITIATES MIGRATION?

- Depends on the goal or reason for migration
- OS initiates
  - if the goal is load balancing.
  - May be transparent to process
- Process initiates
  - If the goal is to access a particular resource
  - Process must be aware of the distributed system

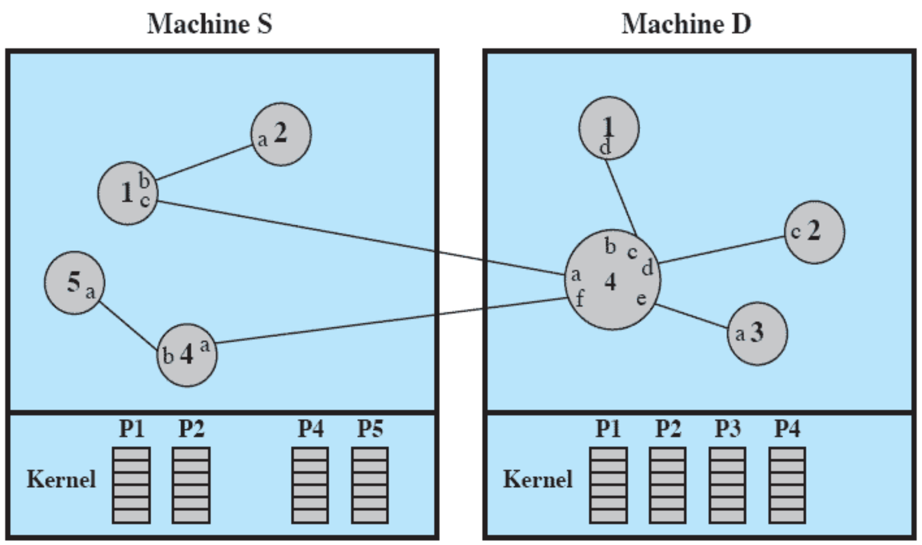
WHAT IS INVOLVED IN MIGRATION?

- Must destroy the process on the source system and create it on the target system
  - Process movement, not replication.
- Process image and process control block and any links must be moved

EXAMPLE OF PROCESS MIGRATION



(a) Before migration



(b) After migration

WHAT IS MIGRATED?

- Moving the process control block is simple
- Several strategies exist for moving the address space and data including:
  - – Eager (All)
  - – Precopy
  - – Eager (dirty)
  - – Copy-on-reference
  - – Flushing

EAGER (ALL)

- Transfer entire address space
  - No trace of process is left behind
  - If address space is large and if the process does not need most of it, then this approach my be unnecessarily expensive (taking minutes)
- Checkpoint/restart capability is useful.

PRECOPY

- Process continues to execute on the source node while the address space is copied
  - Pages modified on the source during precopy operation have to be copied a second time
  - Reduces the time that a process is frozen and cannot execute during migration

EAGER (DIRTY)

- Transfer only that portion of the address space that is in main memory and have been modified
  - – Any additional blocks of the virtual address space are transferred on demand
- The source machine is involved throughout the life of the process
  - – Maintains page and/or segment table entries.

COPY-ON-REFERENCE

- Variation of Eager(Dirty)
- Pages are only brought over when referenced
  - Has lowest initial cost of process migration

FLUSHING

- Pages are cleared from main memory by flushing dirty pages to disk
- Pages are accessed as needed from disk
  – Relieves the source of holding any pages of the migrated process in main memory

CHOOSING A STRATEGY

- If the process is not using much address space while on the target machine then better to use
  - Eager (dirty)
  - Copy-on-reference
  - Flushing
- Otherwise use
  - Eager (All)
  - Precopy

DISCUSSION

- If the file is initially on the same system as the process to be migrated and if the file is locked for exclusive access by that process, what strategy is recommended?

WHAT HAPPENS TO MESSAGES AND SIGNALS?

- Need to have a way to temporarily store outstanding messages and signals during the migration activity and then direct them to the new destination.
  - May need to maintain forwarding details at the initial site to ensure outstanding messages and signals get through

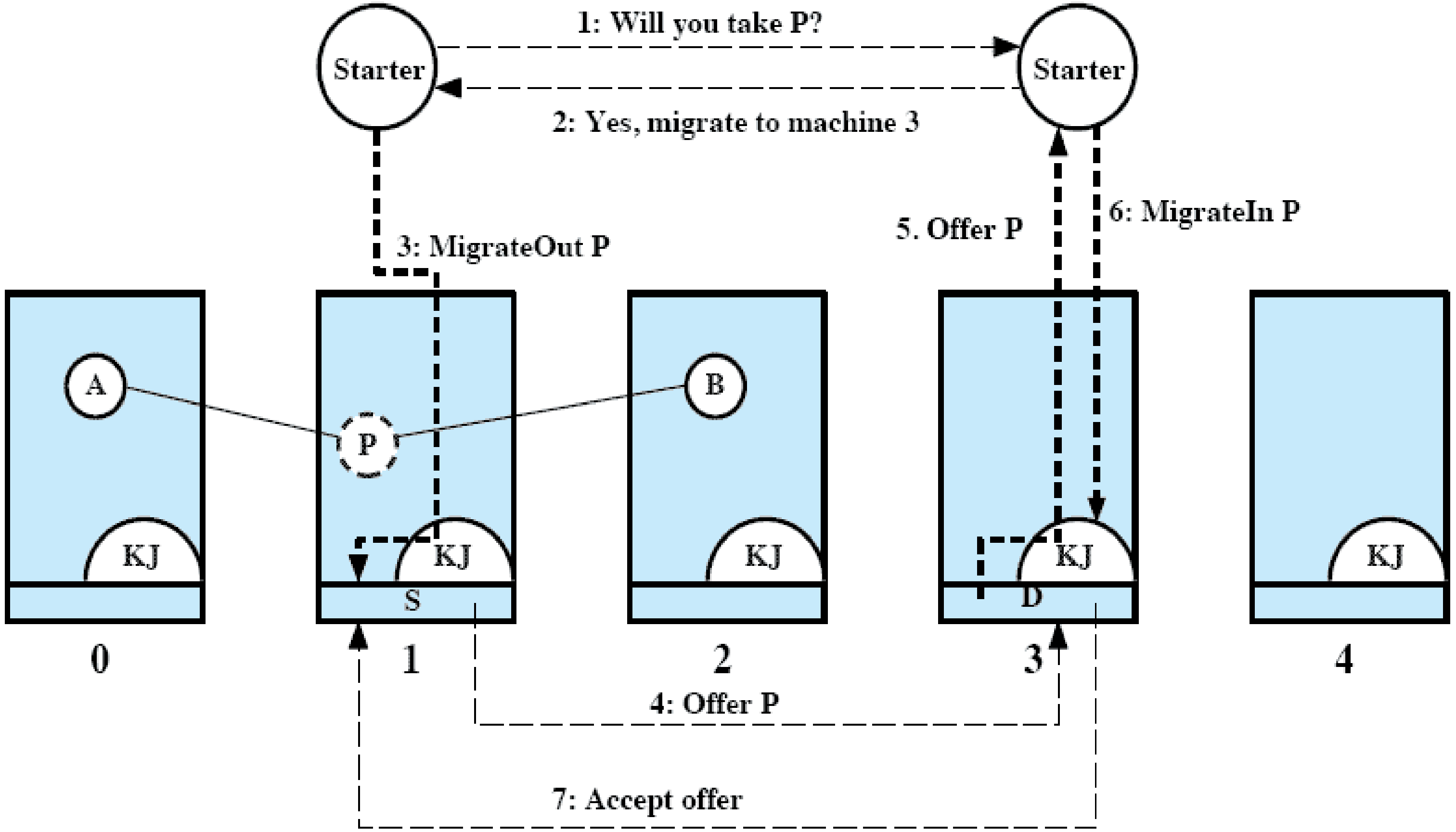
DECISION TO MIGRATE

- Decision to migrate may be made by a single entity
  - OS may decide based on load monitoring module
  - Process may decide based on resource needs
- Some systems let the target system participate in the decision.
  - Negotiated migration

MIGRATION BY NEGOTIATION

- Migration policy is responsibility of a Starter utility
  - Starter utility is also responsible for long-term scheduling and memory allocation
- Migration decision must be reached jointly by two Starter processes
  - one on the source and one on the destination

NEGOTIATION OF PROCESS MIGRATION



EVICTION

- Destination system may refuse to accept the migration of a process to itself
- If a workstation is idle, process may have been migrated to it
  - Once the workstation is active, it may be necessary to evict the migrated processes to provide adequate response time

PREEMPTIVE VS. NONPREEMPTIVE TRANSFERS

- Previous points related to preemptive processes
  - Process has been created and may have begun executing
- Nonpreemptive process transfers involve processes which have not yet begun
  - So have no state to transfer
  - Useful in load balancing.

DISTRIBUTED GLOBAL STATE

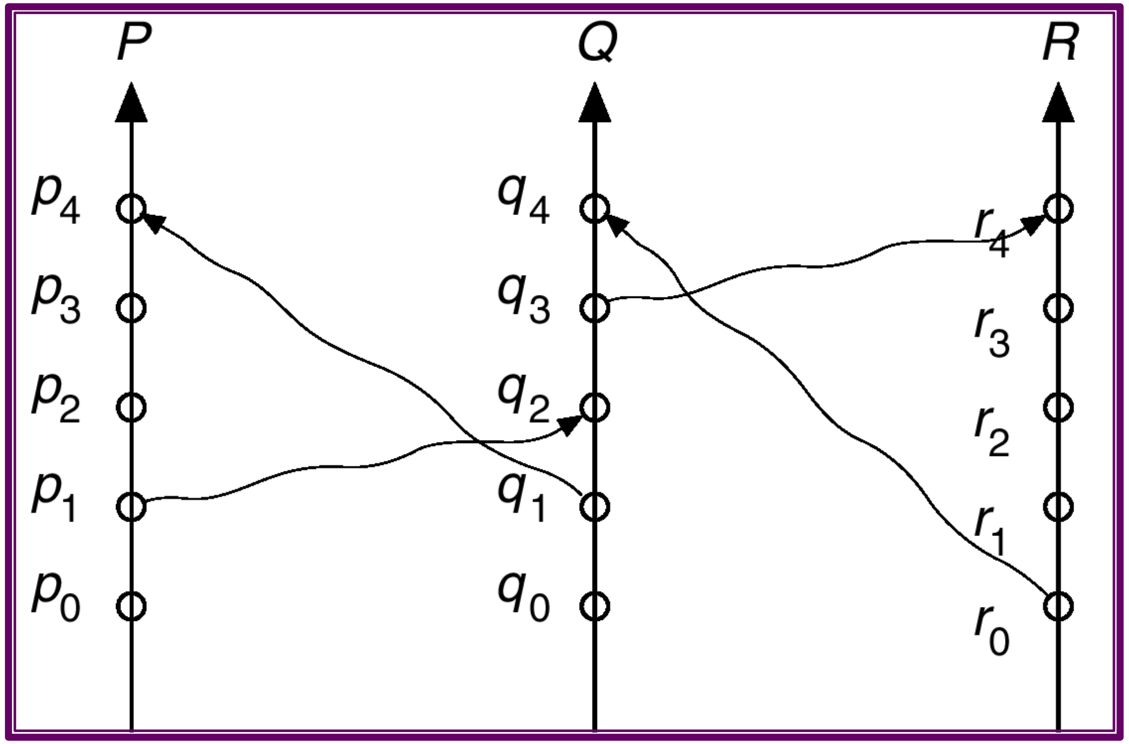
- Operating system cannot know the current state of all process in the distributed system
- A process can only know the current state of all processes on the local system
- Remote processes only know state information that is received by messages

DISTRIBUTED COORDINATION

EVENT ORDERING

- *Happened-before* relation (denoted by →).
  - If *A* and *B* are events in the same process, and *A was* executed before *B*, then *A* → *B*.
  - If *A* is the event of sending a message by one process and *B* is the event of receiving that message by another process, then *A* → *B*.
  - If *A* → *B* and B → *C* then *A* → *C*.

RELATIVE TIME FOR THREE CONCURRENT PROCESSES



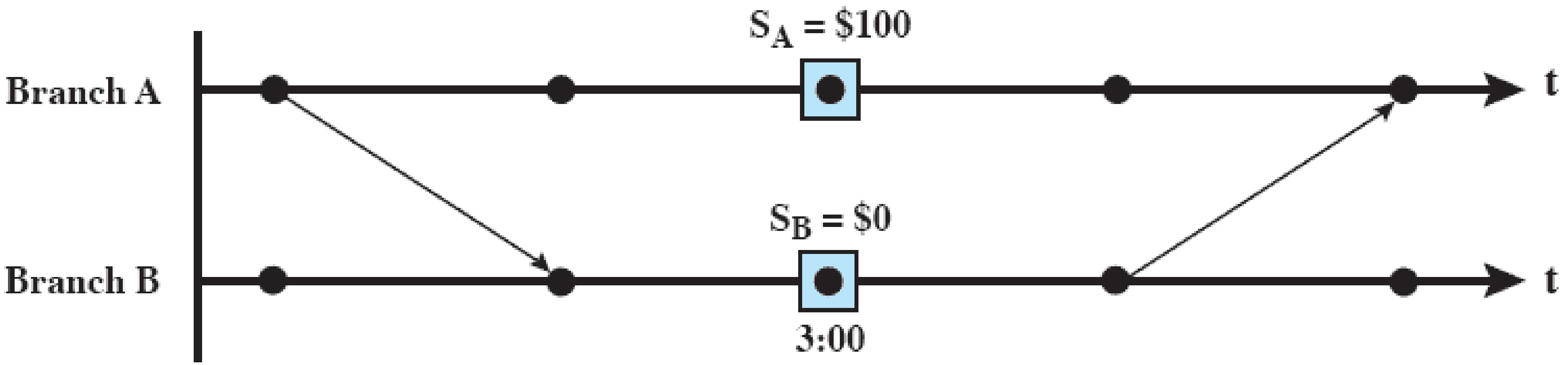
IMPLEMENTATION OF $\rightarrow$

- Associate a timestamp with each system event. Require that for every pair of events *A* and *B*, if $A \rightarrow B$, then the timestamp of *A* is less than the timestamp of *B*.
- Within *each* process P_i a *logical clock*, LC_i is associated. The logical clock can be implemented as a simple counter that is incremented between any two successive events executed within a process.
- A process advances its logical clock when it receives a message whose timestamp is greater than the current value of its logical clock.
- If the timestamps of two events *A* and *B* are the same, then the events are concurrent. We may use the process identity numbers to break ties and to create a total ordering.

EXAMPLE

- Bank account is distributed over two branches
- The total amount in the account is the sum at each branch
- At 3 PM the account balance is determined
- Messages are sent to request the information

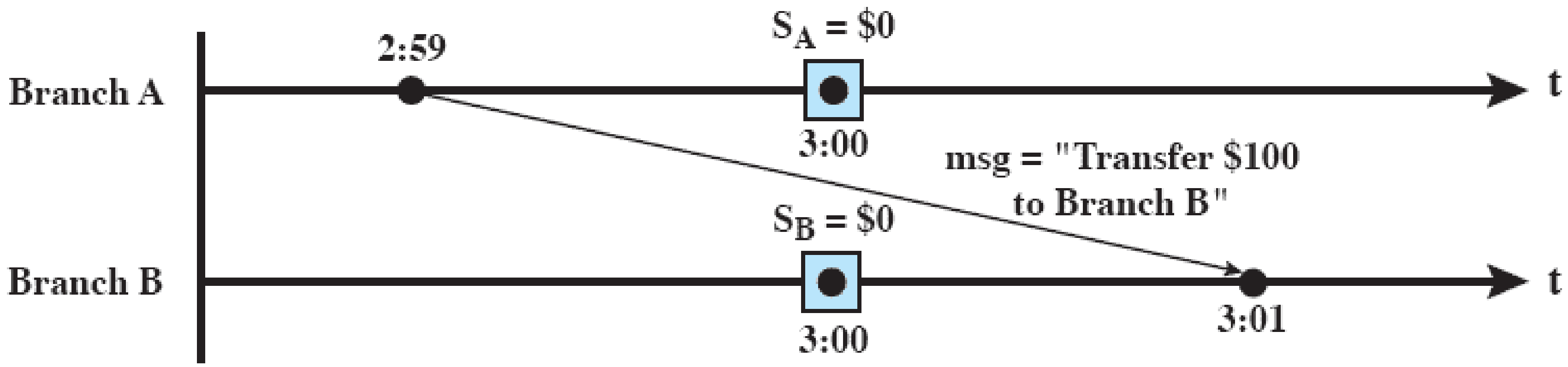
EXAMPLE 1



(a) Total = \$100

EXAMPLE 2

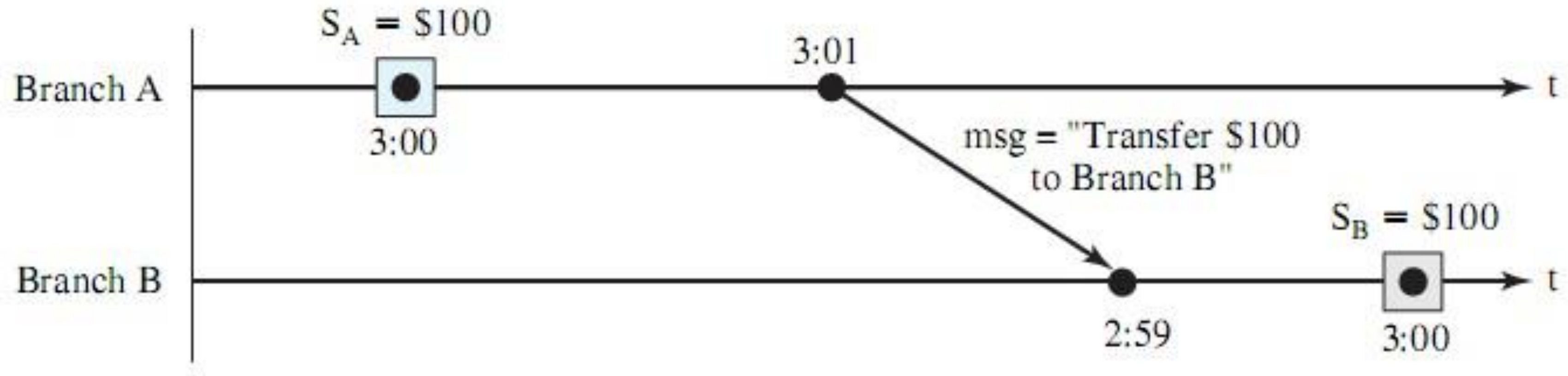
- If at the time of balance determination, the balance from branch A is in transit to branch B
- The result is a false reading



(b) Total = $0

EXAMPLE 3

- All messages in transit must be examined at time of observation
- Total consists of balance at both branches and amount in message



(c) Total = $200

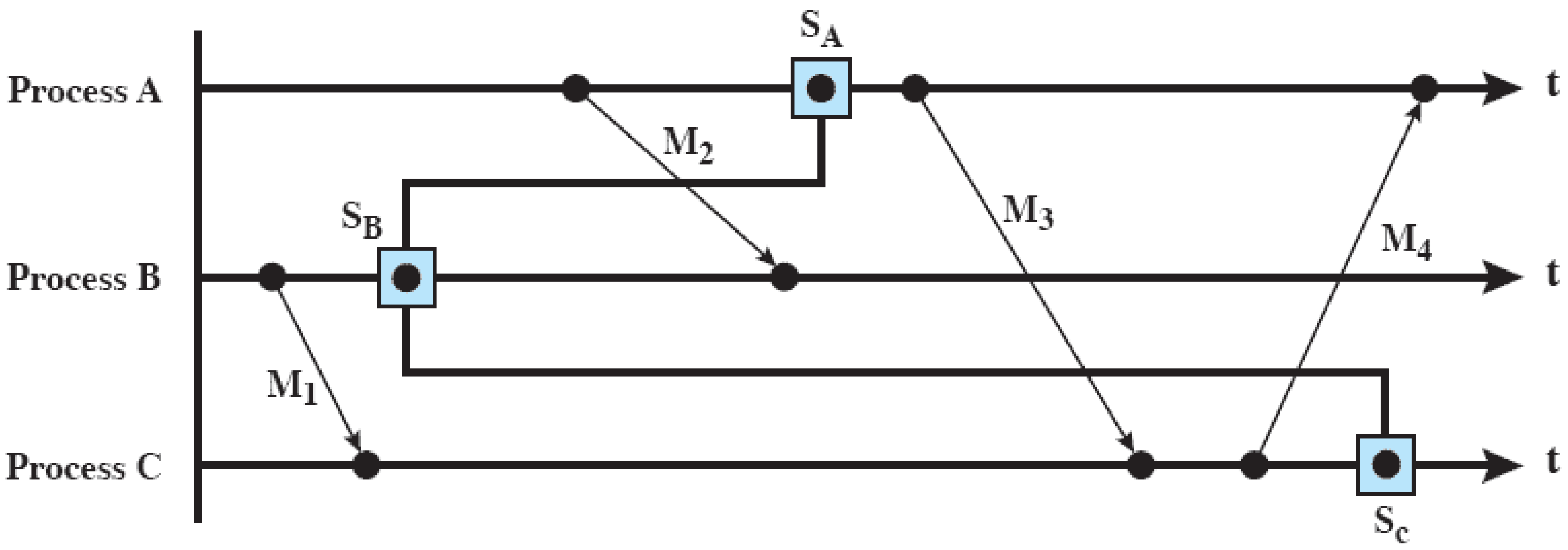
SOME TERMS

- Channel
  - Exists between two processes if they exchange messages
- State
  - Sequence of messages that have been sent and received along channels incident with the process

SOME TERMS

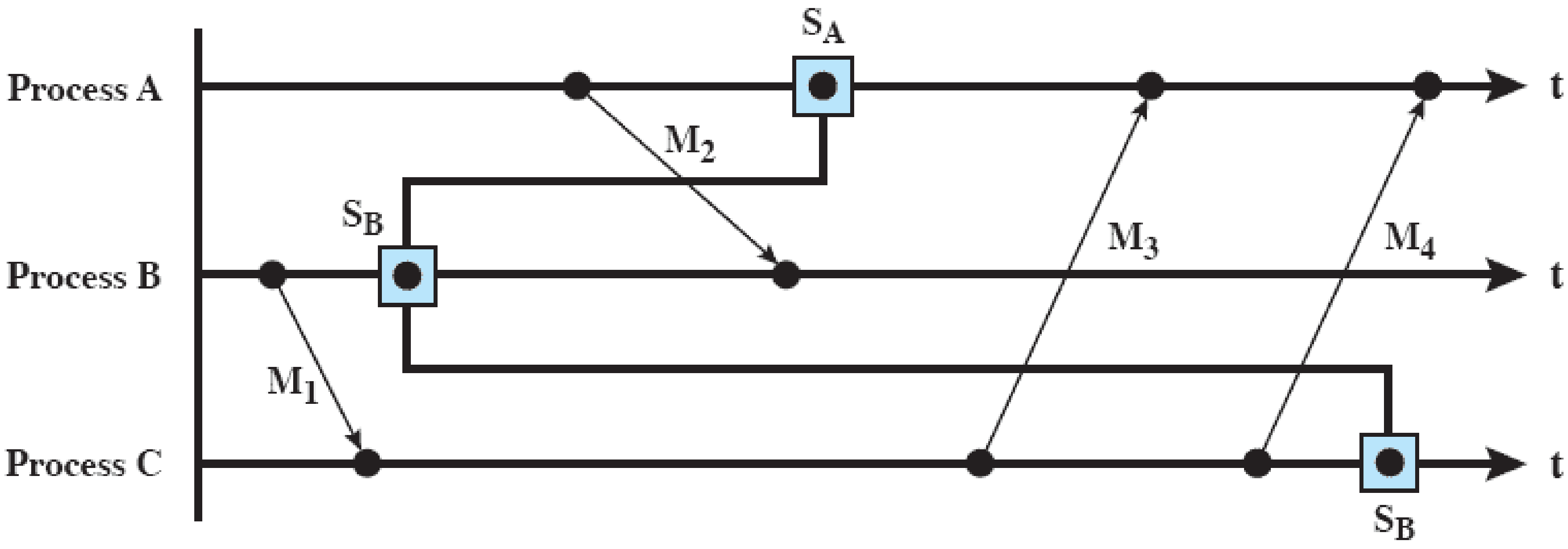
- Snapshot
  - Records the state of a process
- Global state
  - The combined state of all processes
- Distributed Snapshot
  - A collection of snapshots, one for each process

INCONSISTENT GLOBAL STATE



(a) Inconsistent Global State

CONSISTENT GLOBAL STATE



(b) Consistent Global State

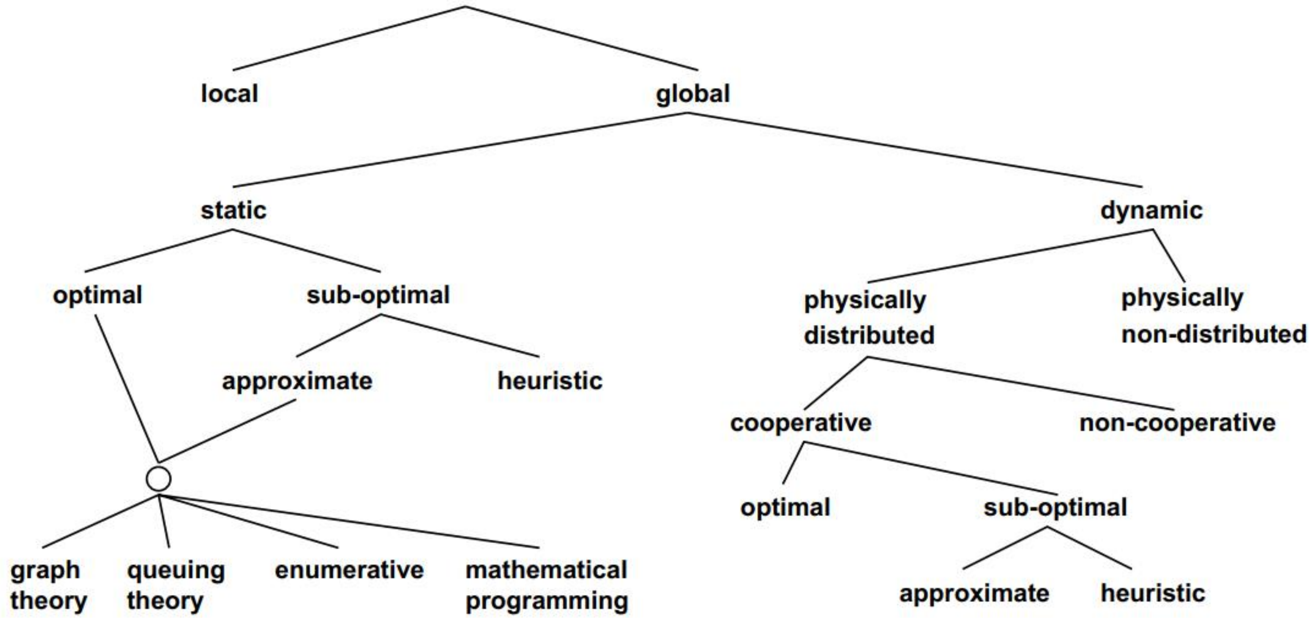
DISTRIBUTED SNAPSHOT ALGORITHM

- Records a consistent global state
- Assumes messages are delivered in order that they were sent
  - And no messages are lost
  - TCP satisfies requirements
- Uses a special control message
  - ***Marker***

DISTRIBUTED SCHEDULING ALGORITHM CHOICES

- Level of scheduling
  - local scheduling
  - global scheduling
- Load distribution goals
  - load balancing
  - load sharing

A TAXONOMY OF DISTRIBUTED SCHEDULING ALGORITHMS



CLASSIFICATION OF GLOBAL SCHEDULING

- Goal – To transfer load from heavily loaded computers to idle or lightly loaded computers
- Broadly characterized as :
  - Static: Decision is hard wired in the algorithm using apriori knowledge of the system
  - Dynamic: Make use of system state information to make load distributing decisions
  - Adaptive: Special class of dynamic algorithm, they adopt their activities by dynamically changing the parameters of the algorithm to suit the changing system state

LOAD BALANCING VS. LOAD SHARING

- unshared state :
  - A state in which one computer lies idle while at the same time tasks contend for service at another computer
- to reduce the likelihood of unshared state
- Load balancing algorithms
  - Attempt to equalize the loads at all computers
  - Higher overhead than load sharing algo
- anticipatory task transfer
  - To reduce the duration of unshared state

DISTRIBUTED PROCESS MANAGEMENT ISSUES

DEADLOCK PREVENTION

- Resource-ordering deadlock-prevention – define a *global* ordering among the system resources.
  - Assign a unique number to all system resources.
  - A process may request a resource with unique number *i* only if it is not holding a resource with a unique number grater than *i*.
  - Simple to implement; requires little overhead.

- Banker's algorithm – designate one of the processes in the system as the process that maintains the information necessary to carry out the Banker's algorithm.
  - Also implemented easily, but may require too much overhead

TIMESTAMPED DEADLOCK-PREVENTION SCHEME

- Each process P_i is assigned a unique priority number
- Priority numbers are used to decide whether a process P_i should wait for a process P_j; otherwise P_i is rolled back.
- The scheme prevents deadlocks. For every edge $P_i \rightarrow P_j$ in the wait-for graph, P_i has a higher priority than P_j. Thus a cycle cannot exist.

WAIT-DIE SCHEME

- Based on a nonpreemptive technique.
- If P_i requests a resource currently held by P_j, P_i is allowed to wait only if it has a smaller timestamp than does P_j (P_i is older than P_j). Otherwise, P_i is rolled back (dies).
- Example: Suppose that processes P_1, P_2, and P_3 have timestamps 5, 10, and 15 respectively.
  - if P_1 request a resource held by P_2, then P_1 will wait.
  - If P_3 requests a resource held by P_2, then P_3 will be rolled back.

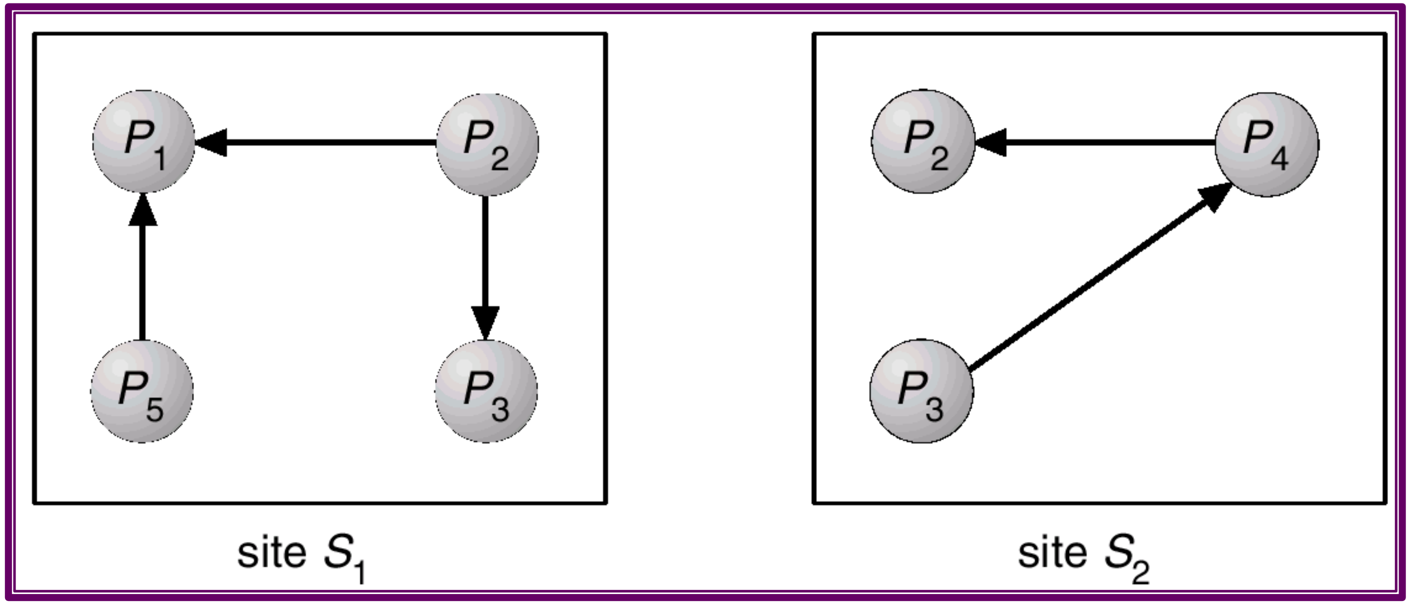
WOULD-WAIT SCHEME

- Based on a preemptive technique; counterpart to the wait-die system.

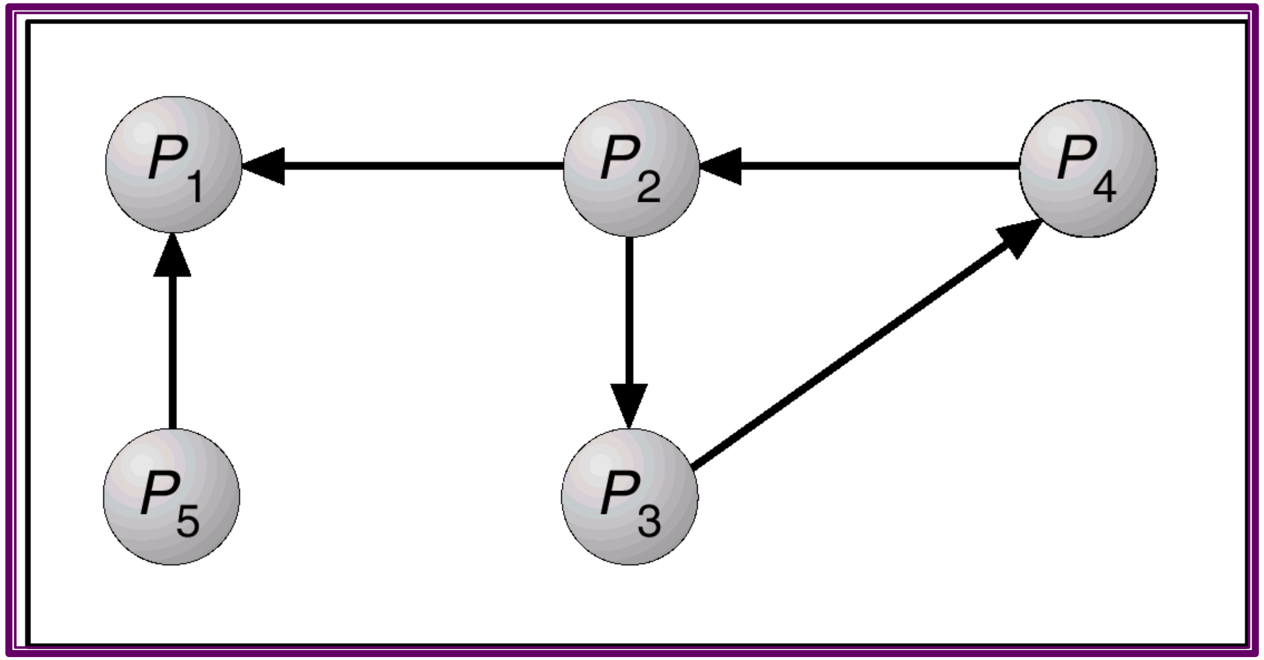
- If P_i requests a resource currently held by P_j, P_i is allowed to wait only if it has a larger timestamp than does P_j (P_i is younger than P_j). Otherwise P_j is rolled back (P_j is wounded by P_i).

- Example: Suppose that processes P_1, P_2, and P_3 have timestamps 5,10 and 15 respectively
  - If P_1 requests a resource held by P_2, then the resource will be preempted from P_2 and P_2 will be rolled back.
  - If P_3 requests a resource held by P_2, then P_3 will wait.

TWO LOCAL WAIT-FOR GRAPHS



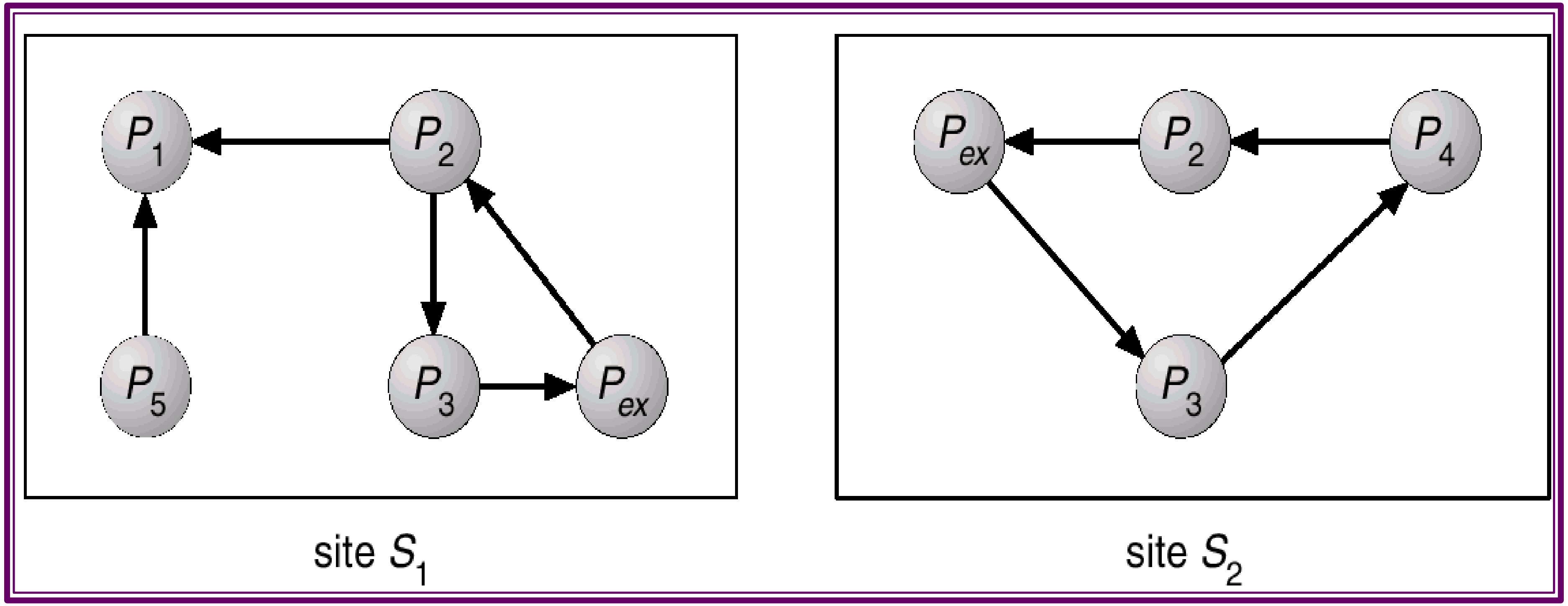
GLOBAL WAIT-FOR GRAPH



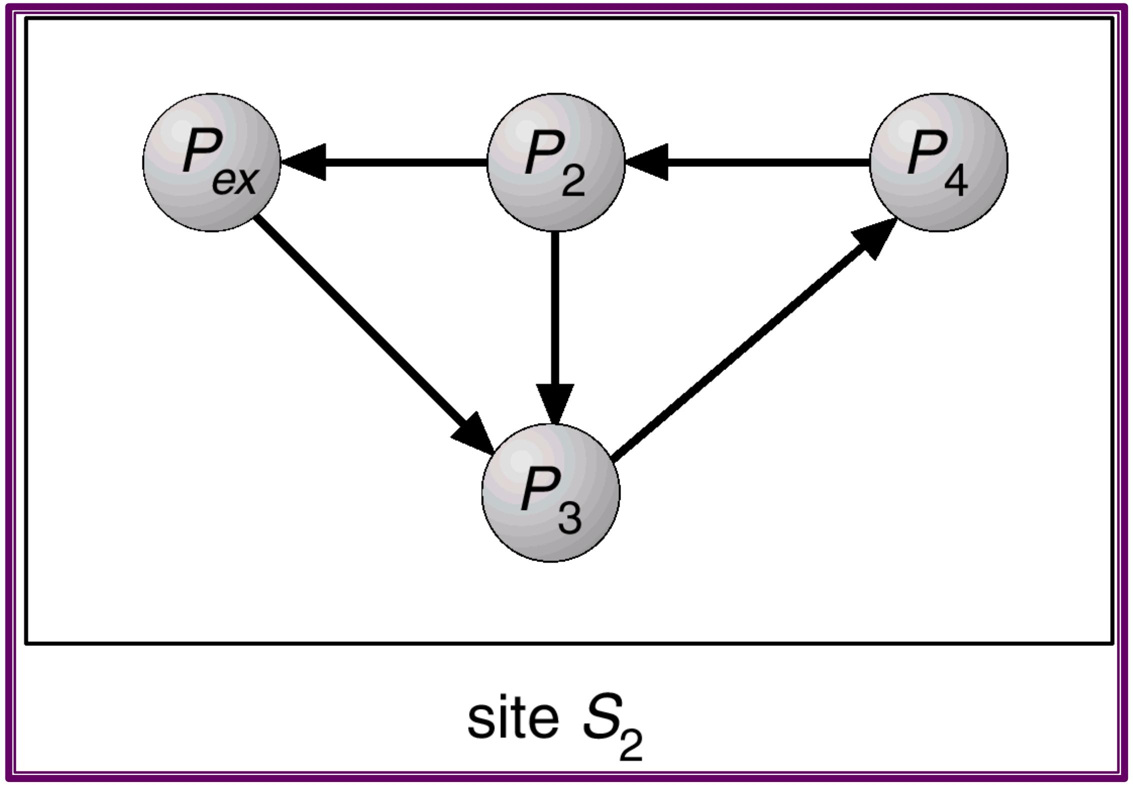
FULLY DISTRIBUTED APPROACH

- All controllers share equally the responsibility for detecting deadlock.
- Every site constructs a wait-for graph that represents a part of the total graph.
- We add one additional node P_{ex} to each local wait-for graph.
- If a local wait-for graph contains a cycle that does not involve node P_{ex}, then the system is in a deadlock state.
- A cycle involving P_{ex} implies the possibility of a deadlock. To ascertain whether a deadlock does exist, a distributed deadlock-detection algorithm must be invoked.

AUGMENTED LOCAL WAIT-FOR GRAPHS



AUGMENTED LOCAL WAIT-FOR GRAPH IN SITE



ELECTION ALGORITHMS

- Determine where a new copy of the coordinator should be restarted.
- Assume that a unique priority number is associated with each active process in the system, and assume that the priority number of process P_i is i.
- Assume a one-to-one correspondence between processes and sites.
- The coordinator is always the process with the largest priority number. When a coordinator fails, the algorithm must elect that active process with the largest priority number.
- Two algorithms, the bully algorithm and a ring algorithm, can be used to elect a new coordinator in case of failures.

BULLY ALGORITHM

- Applicable to systems where every process can send a message to every other process in the system.
- If process P_i sends a request that is not answered by the coordinator within a time interval T, assume that the coordinator has failed; P_i tries to elect itself as the new coordinator.
- P_i sends an election message to every process with a higher priority number, P_i then waits for any of these processes to answer within T.

BULLY ALGORITHM (CONT.)

- If no response within *T*, assume that all processes with numbers greater than i have failed; P_i elects itself the new coordinator.
- If answer is received, P_i begins time interval T', waiting to receive a message that a process with a higher priority number has been elected.
- If no message is sent within T', assume the process with a higher number has failed; P_i should restart the algorithm

BULLY ALGORITHM (CONT.)

- If P_i is not the coordinator, then, at any time during execution, P_i may receive one of the following two messages from process P_j.
  - P_j is the new coordinator ($j > i$). P_i, in turn, records this information.
  - P_j started an election ($j > i$). P_i, sends a response to P_j and begins its own election algorithm, provided that *Pi* has not already initiated such an election.

- After a failed process recovers, it immediately begins execution of the same algorithm.

- If there are no active processes with higher numbers, the recovered process forces all processes with lower number to let it become the coordinator process, even if there is a currently active coordinator with a lower number.

RING ALGORITHM

- Applicable to systems organized as a ring (logically or physically).
- Assumes that the links are unidirectional, and that processes send their messages to their right neighbors.
- Each process maintains an *active list*, consisting of all the priority numbers of all active processes in the system when the algorithm ends.
- If process P_i detects a coordinator failure, I creates a new active list that is initially empty. It then sends a message *elect(i)* to its right neighbor, and adds the number *i* to its active list.

RING ALGORITHM (CONT.)

- If P_i receives a message elect(j) from the process on the left, it must respond in one of three ways:

1. If this is the first *elect* message it has seen or sent, P_i creates a new active list with the numbers i and j. It then sends the message *elect(i),* followed by the message *elect(j).*

- If $i \neq j$, then the active list for P_i now contains the numbers of all the active processes in the system. P_i can now determine the largest number in the active list to identify the new coordinator process.
- If $i = j$, then P_i receives the message *elect(i)*. The active list for P_i contains all the active processes in the system. P_i can now determine the new coordinator process.

REACHING AGREEMENT

- There are applications where a set of processes wish to agree on a common "value".

- Such agreement may not take place due to:
  - Faulty communication medium
  - Faulty processes
    - Processes may send garbled or incorrect messages to other processes.
    - A subset of the processes may collaborate with each other in an attempt to defeat the scheme.

FAULTY COMMUNICATIONS

- Process P_i at site *A*, has sent a message to process P_j at site *B*; to proceed, P_i needs to know if P_j has received the message.
- Detect failures using a time-out scheme.
  - When P_i sends out a message, it also specifies a time interval during which it is willing to wait for an acknowledgment message form P_j.
  - When P_j receives the message, it immediately sends an acknowledgment to P_i
  - If P_i receives the acknowledgment message within the specified time interval, it concludes that P_j has received its message. If a time-out occurs, P_j needs to retransmit its message and wait for an acknowledgment.
  - Continue until P_i either receives an acknowledgment, or is notified by the system that *B* is down.

FAULTY COMMUNICATIONS (CONT.)

- Suppose that P_j also needs to know that P_i has received its acknowledgment message, in order to decide on how to proceed.

  - In the presence of failure, it is not possible to accomplish this task
  - It is not possible in a distributed environment for processes P_i and P_j to agree completely on their respective states.

FAULTY PROCESSES (BYZANTINE GENERALS PROBLEM)

- Communication medium is reliable, but processes can fail in unpredictable ways.
- Consider a system of n processes, of which no more than m are faulty. Suppose that each process P_i has some private value of V_i.
- Devise an algorithm that allows each nonfaulty P_i to construct a vector $X_i = (A_{i,1}, A_{i,2}, \ldots, A_{i,n})$ such that::
  - If P_j is a nonfaulty process, then $A_{ij} = V_j$.
  - If P_i and P_j are both nonfaulty processes, then $X_i = X_j$.
- Solutions share the following properties.
  - A correct algorithm can be devised only if $n \geq 3 \times m + 1$
  - The worst-case delay for reaching agreement is proportionate to $m + 1$ message-passing delays.

FAULTY PROCESSES (CONT.)

- An algorithm for the case where m = 1 and n = 4 requires two rounds of information exchange:
  - Each process sends its private value to the other 3 processes.
  - Each process sends the information it has obtained in the first round to all other processes.
- If a faulty process refuses to send messages, a nonfaulty process can choose an arbitrary value and pretend that that value was sent by that process.
- After the two rounds are completed, a nonfaulty process P_i can construct its vector $X_i = (A_{i,1}, A_{i,2}, A_{i,3}, A_{i,4})$ as follows:
  - $A_{i,j} = V_i$.
  - For $j \neq i$, if at least two of the three values reported for process P_j agree, then the majority value is used to set the value of A_{ij}. Otherwise, a default value (*nil*) is used.