Table of Contents

Distributed Operating Systems

- Operating Systems: provide problem-oriented abstractions of the underlying physical resources.
- Files (rather than disk blocks) and sockets (rather than raw network access).

Selected Issues

- Mutual exclusion and election
  - Non-token-based vs. token-based
  - Election and bidding
- Detection and resolution of deadlock
  - Four conditions for deadlock: mutual exclusion, hold and wait, no preemption, and circular wait.
  - Graph-theoretic model: wait-for graph
  - Two situations: AND model (process deadlock) and OR model (communication deadlock)
- Task scheduling and load balancing
  - Static scheduling vs. dynamic scheduling

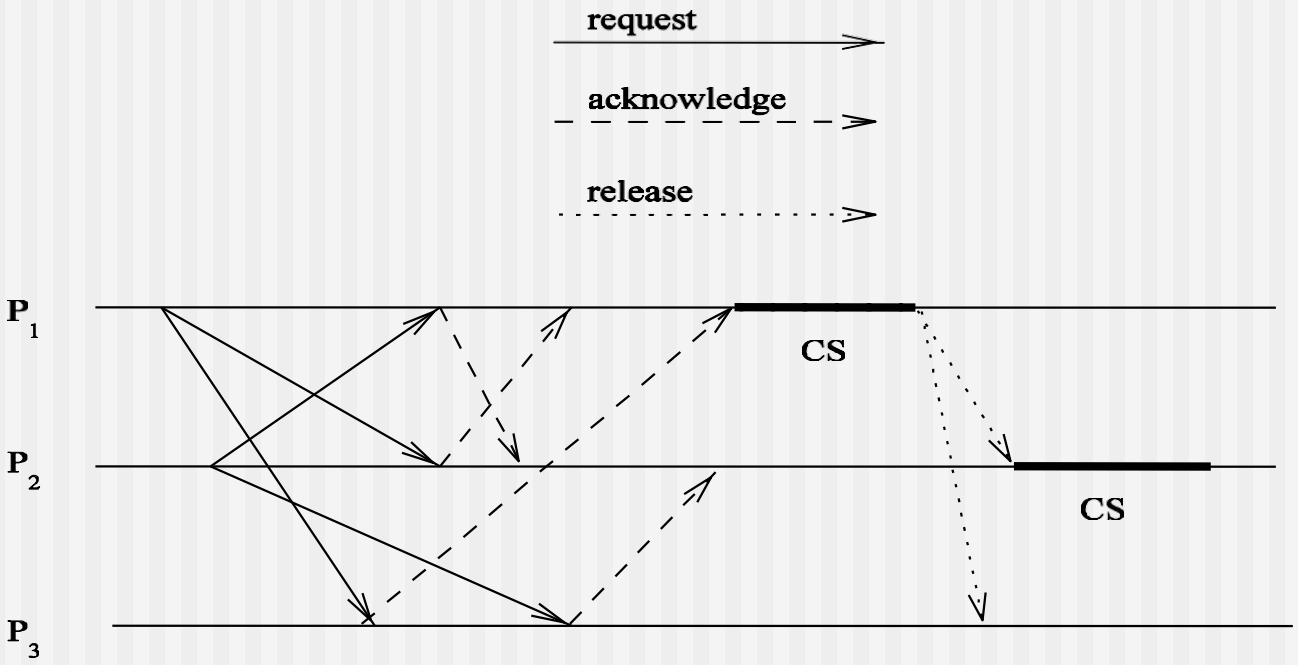
Mutual Exclusion and Election

- Requirements:
  - Freedom from deadlock.
  - Freedom from starvation.
  - Fairness.
- Measurements:
  - Number of messages per request.
  - Synchronization delay.
  - Response time.

Non-Token-Based Solutions: Lamport's Algorithm

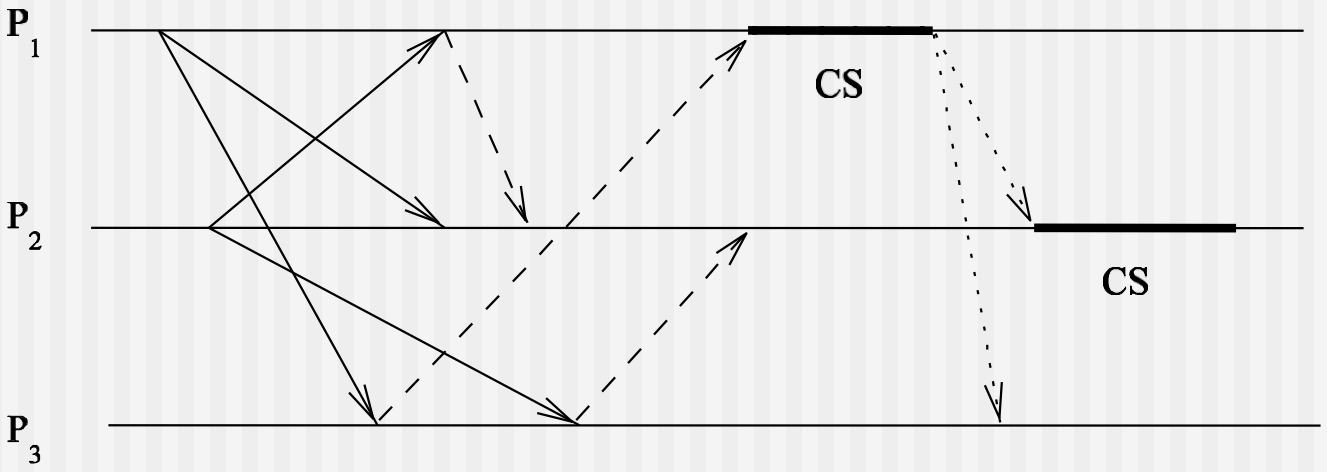
- To request the resource process P_i sends its timestamped message to all the processes (including itself).
- When a process receives the request resource message, it places it on its local request queue and sends back a timestamped acknowledgment.
- To release the resource, P_i sends a timestamped release resource message to all the processes (including itself).
- When a process receives a release resource message from P_i, it removes any requests from P_i from its local request queue. A process P_j is granted the resource when
  - Its request r is at the top of its request queue, and,
  - It has received messages with timestamps larger than the timestamp of r from all the other processes.

Example for Lamport's Algorithm



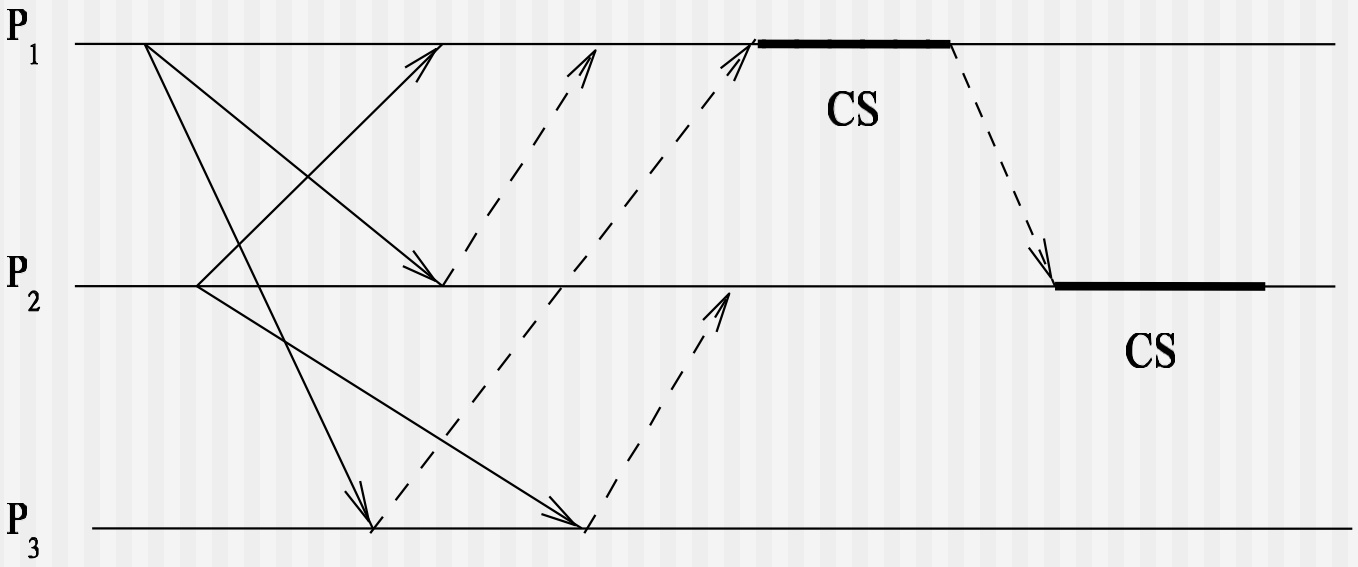
Extension

- There is no need to send an acknowledgement when process P_j receives a request from process P_i after it has sent its own request with a timestamp larger than the one of P_i's request.
- An example for Extended Lamport's Algorithm



Ricart and Agrawala's Algorithm

It merges acknowledge and release messages into one message *reply*.

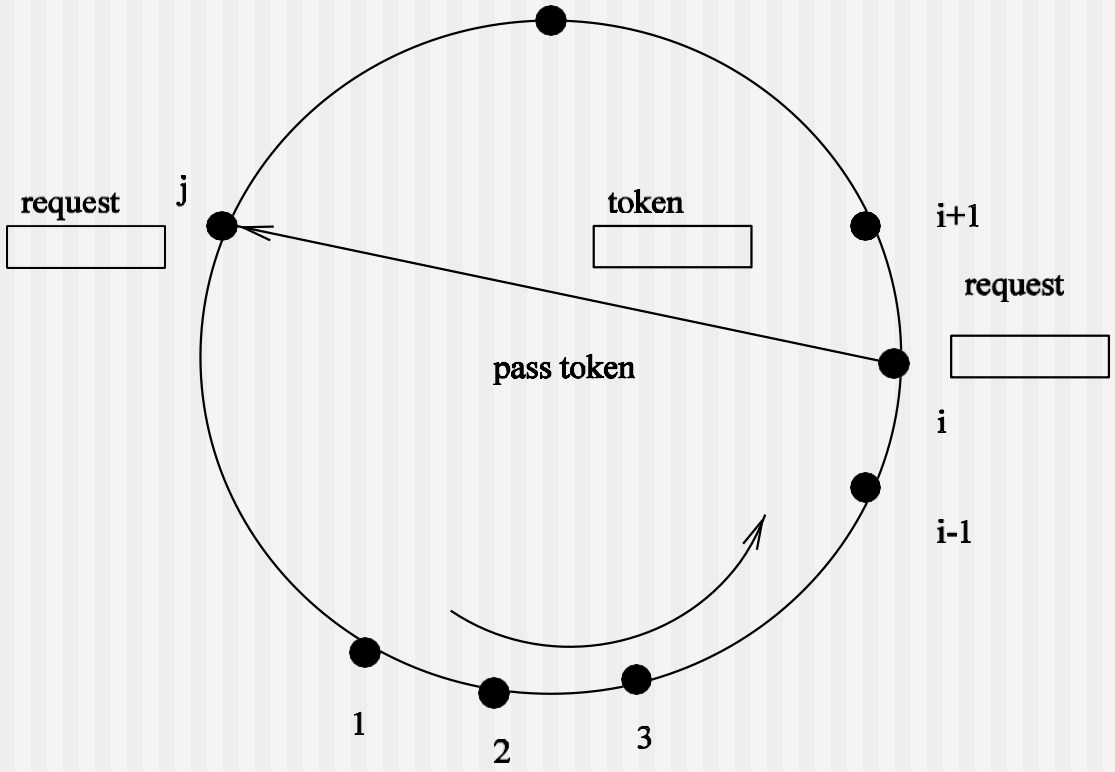


An example using Ricart and Agrawala's algorithm.

Token-Based Solutions: Ricart and Agrawala's Second Algorithm

- When token holder P_i exits CS, it searches other processes in the order $i + 1, i + 2, \ldots, n, 1, 2, \ldots, i - 1$ for the first j such that the timestamp of P_j 's last request for the token is larger than the value recorded in the token for the timestamp of P_j 's last holding of the token.

Token-based Solutions (Cont'd)



Ricart and Agrawala's second algorithm.

Pseudo Code

P(*i*)::=*[request-resource
□ consume
□ release-resource
□ treat-request-message
□ others
]

distributed-mutual-exclusion ::= ||*P*(*i*:1..n)

clock: 0,1,…, (initialized to 0)
token-present: **Boolean** (*F* for all except one process)
token-held: **Boolean** (*F*)
token: **array** (1..*n*) of *clock* (initialized 0)
request: **array** (1..*n*) **of** *clock* (initialized 0)

Pseudo Code (Cont'd)

- others::= all the other actions that do not request to enter the critical section.
- consume::= consumes the resource after entering the critical section
- request-resource::=

[*token present* = F
→ [**send** (*request-signal, clock, i*) **to all**;
receive (*access-signal, token*);
token-present:= T;
token-held:= T
]
]

Pseudo Code (Cont'd)

release-resource::=

[*token* (*i*):=*clock*;

token-held:= F;

min *j* in the order [*i* + 1,… *n*,1,2,…,*i* – 2, *i* – 1]

∧ (*request*(*j*) > *token*(*j*))

→ [*token-present*:= F;

send (*access-signal*, *token*) **to** P_j

]

]

Pseudo Code (Cont'd)

treat-request-message::=

[**receive** (*request-signal, clock; j*)

→[*request*(*j*):=max(*request*(*j*),*clock*);

token-present ∧¬*token-held* → release-resource

]

]

Ring-Based Algorithm

P(*i*:0..*n*-1)::=

[**receive** *token* **from** *P*((*i*-1) mod *n*);

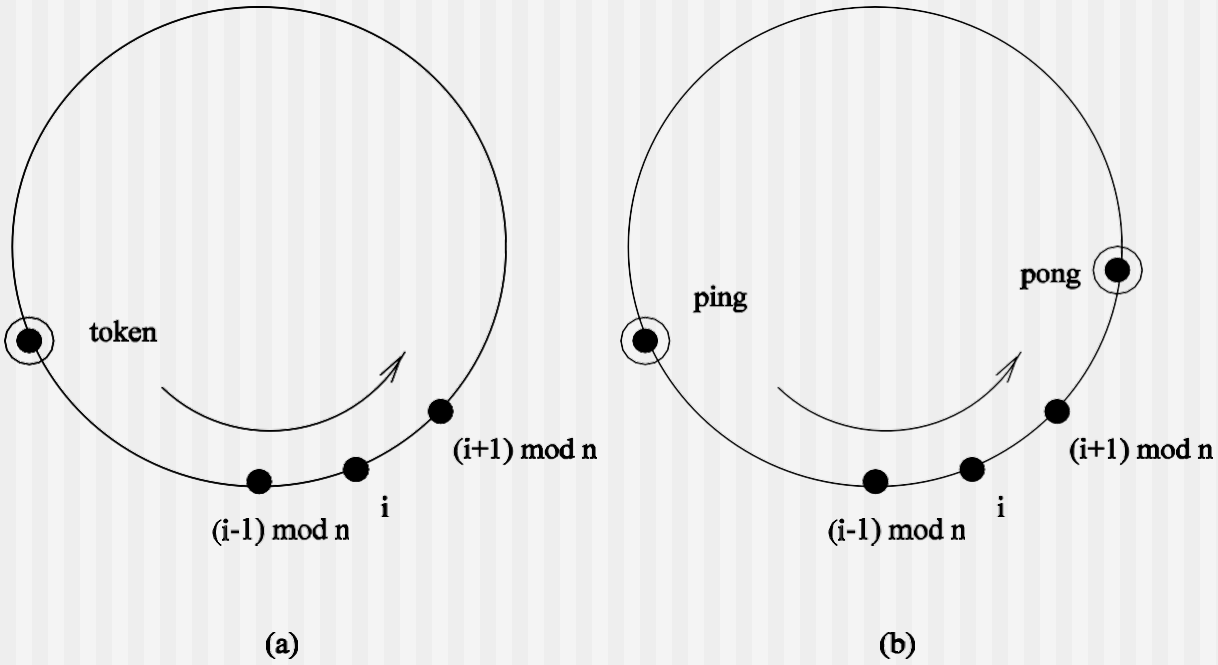
consume the resource if needed;

send token **to** P ((i + 1) mod n)

]

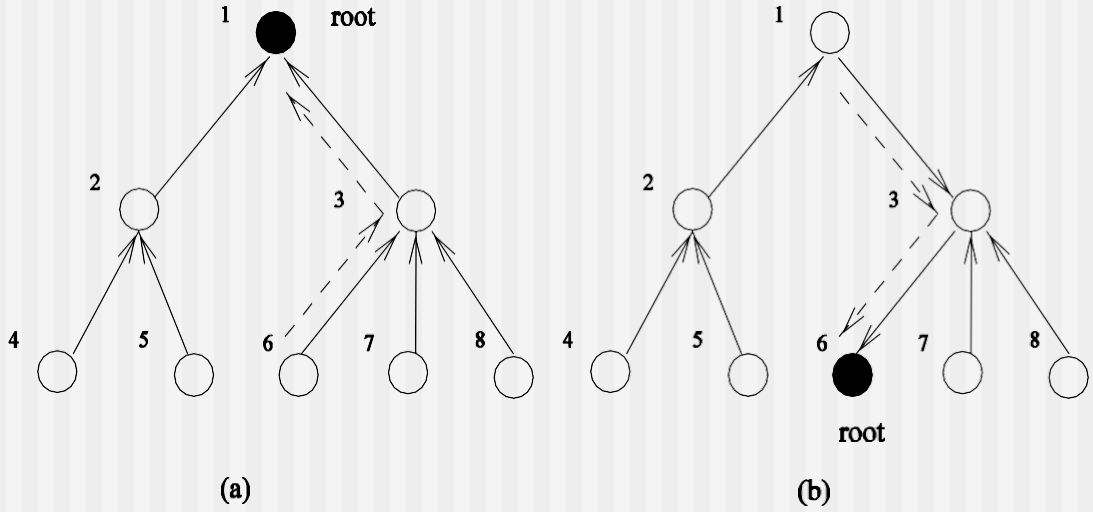
distributed-mutual-exclusion::= ||*P*(i:0..*n*-1)

Ring-Based Algorithm (Cont'd)



The simple token-ring-based algorithm (a) and the fault-tolerant token-ring-based algorithm (b).

Tree-Based Algorithm



A tree-based mutual exclusion algorithm.

Maekawa's Algorithm

- Permission from every other process but only from a subset of processes.
- If R_i and R_j are the request sets for processes P_i and P_j , then $R_i \cap R_j \neq \varphi$.

Example 11

$R_1 : \{P_1; P_3; P_4\}$

$R_2 : \{P_2; P_4; P_5\}$

$R_3 : \{P_3; P_5; P_6\}$

$R_4 : \{P_4; P_6; P_7\}$

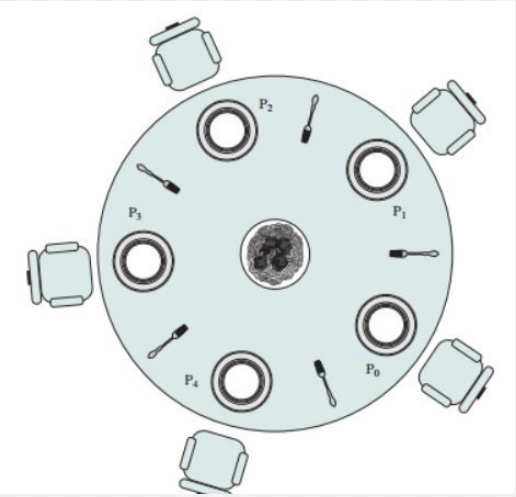
$R_5 : \{P_5; P_7; P_1\}$

$R_6 : \{P_6; P_1; P_2\}$

$R_7 : \{P_7; P_2; P_3\}$

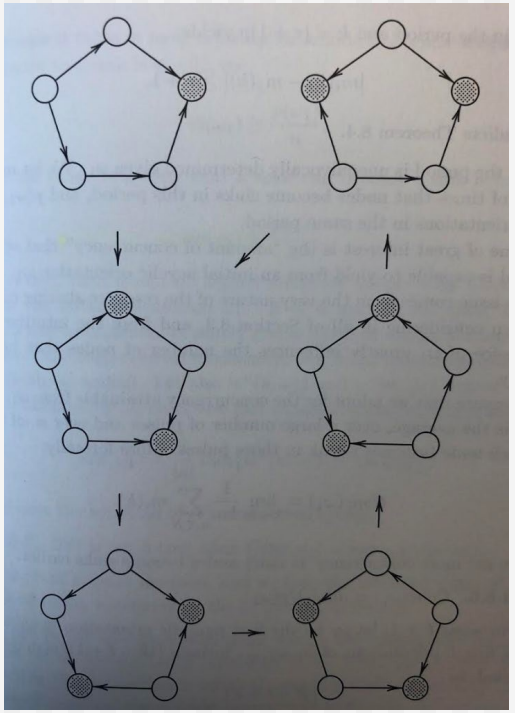
The Dining Philosophers Problem

- Sharing multiple resources
  - Mutual exclusion
  - Deadlock prevention
  - Amount of concurrency
  - Two neighbors i and j: holds_fork$_i^j$ and holds_turn$_i^j$



- edge reversal:

  set up a proper wait-for structure



- Drinking Philosophers Problem

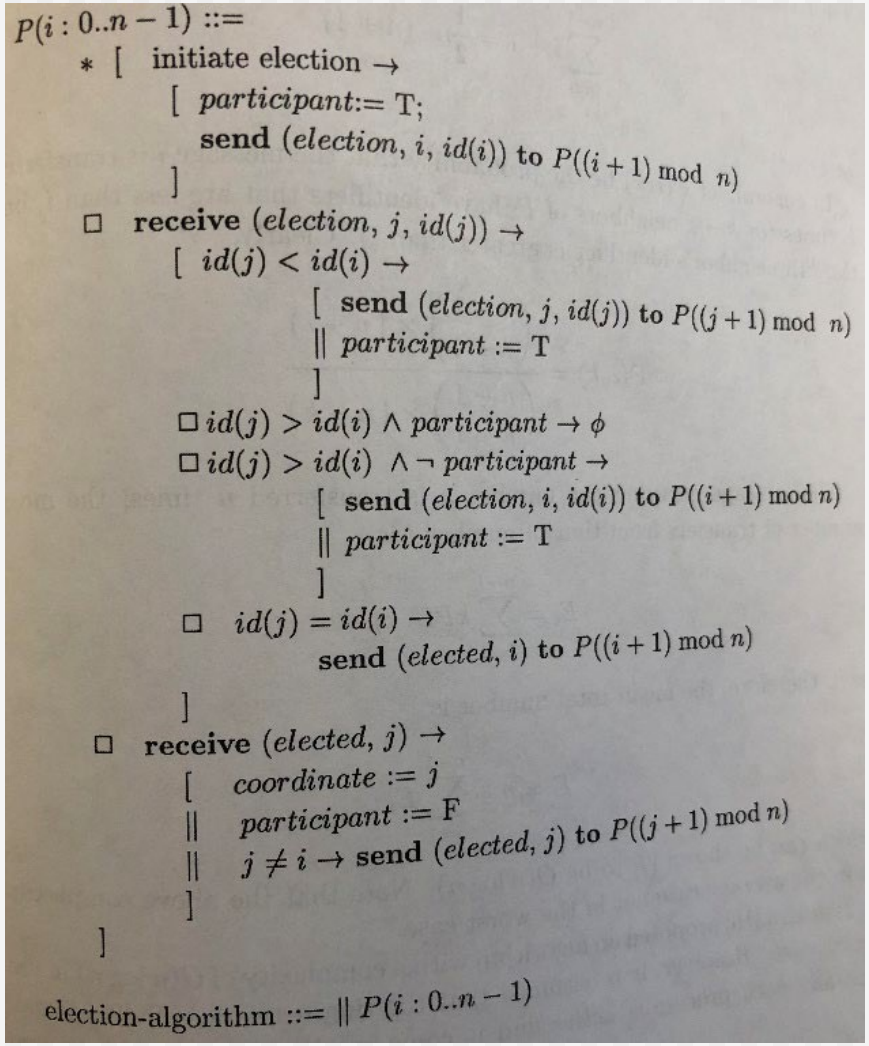
Related Issues

- **Election**: After a failure occurs in a distributed system, it is often necessary to reorganize the active nodes so that they can continue to perform a useful task.
- **Bidding**: Each competitor selects a bid value out of a given set and sends its bid to every other competitor in the system. Every competitor recognizes the same winner.
- **Self-stabilization**: A system is self-stabilizing if, regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps.

Focus 11: Chang and Robert's algorithm

Election on a ring

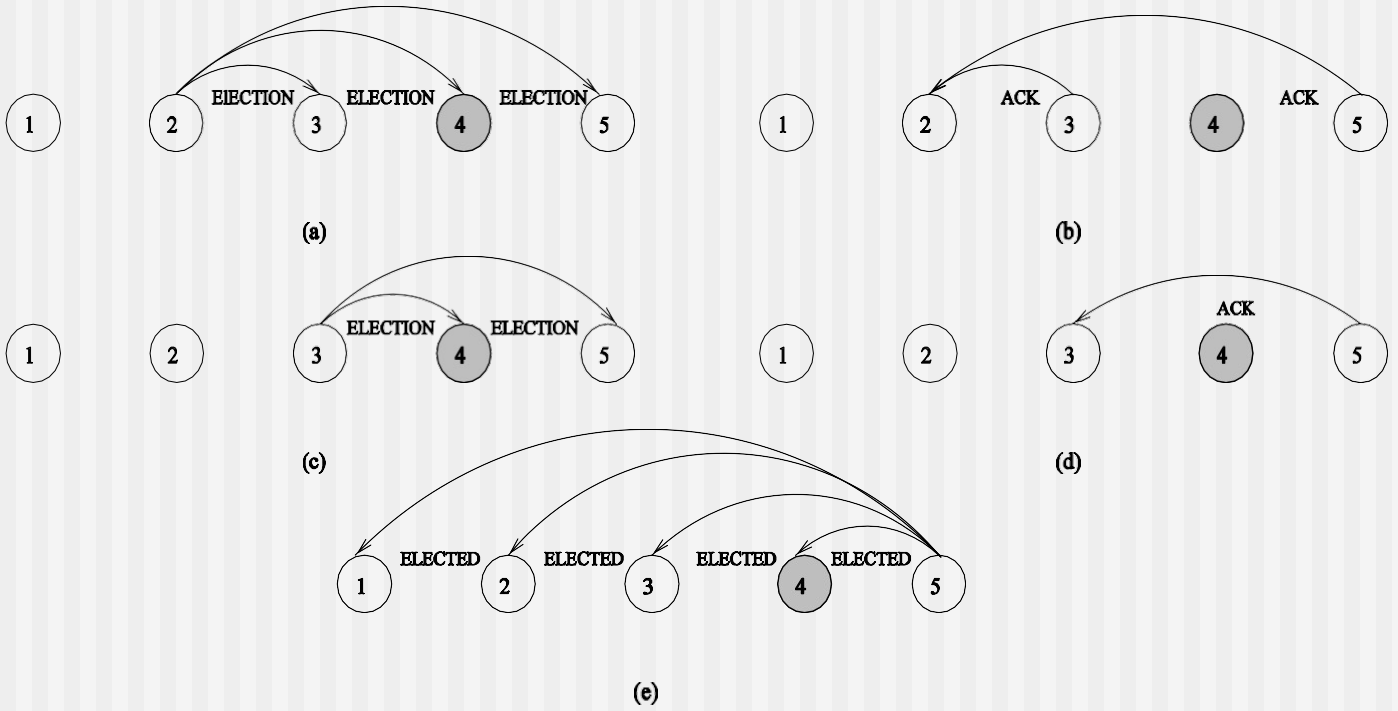
- Election and elected signals
- Smallest ID is the winner
- Two rounds of circulation
- O(n log n) messages



Garcia-Molina's Bully Algorithm for Election

- When *P* detects the failure of the coordinator or receives an ELECTION packet, it sends an ELECTION packet to all processes with higher priorities.
- If no one responds (with packet ACK), *P* wins the election and broadcast the ELECTED packet to all.
- If one of the higher processes responds, it takes over. *P*'s job is done. This higher process will repeat the same action as *P*.

Focus 11 (Cont'd)



Bully algorithm.

Lynch's Non-Comparison-Based Election Algorithms

- Process id is tied to time in terms of rounds.
- *Time-slice algorithm*: (n, the total number of processes, is known)
  - Process Pi (with its id(i)) sends its id in round id(i)2n, i.e., at most one process sends its id in every 2n consecutive rounds.
  - Once an id returns to its original sender, that sender is elected. It sends a signal around the ring to inform other processes of its winning status.
  - message complexity: O(n)
  - time complexity: min{id(i)} n

Lynch's Algorithms (Cont'd)

- *Variable-speed algorithm*: (n is unknown)
  - When a process Pi sends its id (id(i)), this id travels at the rate of one transmission for every 2id(i) rounds.
  - If an id returns to its original sender, that sender is elected.
- message complexity: $n + n/2 + n/2^2 + \ldots + n/2^{(n-1)} < 2n = O(n)$
- time complexity: $2^{\min\{id(i)\}}n$

Dijkstra's Self-Stabilization

- *Legitimate state P* : A system is in a legitimate state P if and only if one process has a privilege.
- *Convergence*: Starting from an arbitrary global state, S is guaranteed to reach a global state satisfying P within a finite number of state transitions.

Example 12

- A ring of finite-state machines with three states. A privileged process is the one that can perform state transition.
- For P_i, $0 < i \leq n - 1$,
  - $P_i \neq P_{i-1} \rightarrow P_i := P_{i-1}$,
  - $P_0 = P_{n-1} \rightarrow P_0 := (P_0 + 1) \bmod k$

Theorem: If $k > n$, then Dijkstra's token ring for mutual exclusion always eventually reaches a correct configuration.

For $n > 2$, theorem also hold if $k = n-1$.

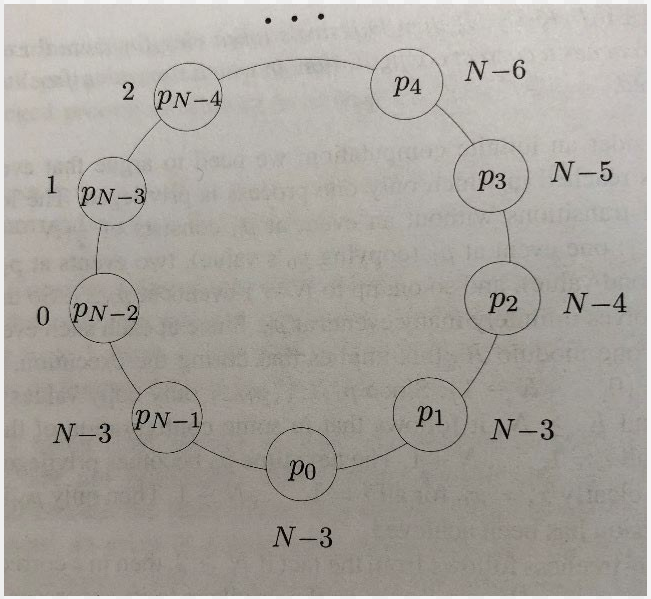
P0	P1	P2	Privileged processes	Process to make move
2	1	2	P0,P1,P2	P0
3	1	2	P1,P2	P1
3	3	2	P2	P2
3	3	3	P0	P0
0	3	3	P1	P1
0	0	3	P2	P2
0	0	0	P0	P0
1	0	0	P1	P1
1	1	0	P2	P2
1	1	1	P0	P0
2	1	1	P1	P1
2	2	1	P2	P2
2	2	2	P0	P0
3	2	2	P1	P1
3	3	2	P2	P2
3	3	3	P0	P0

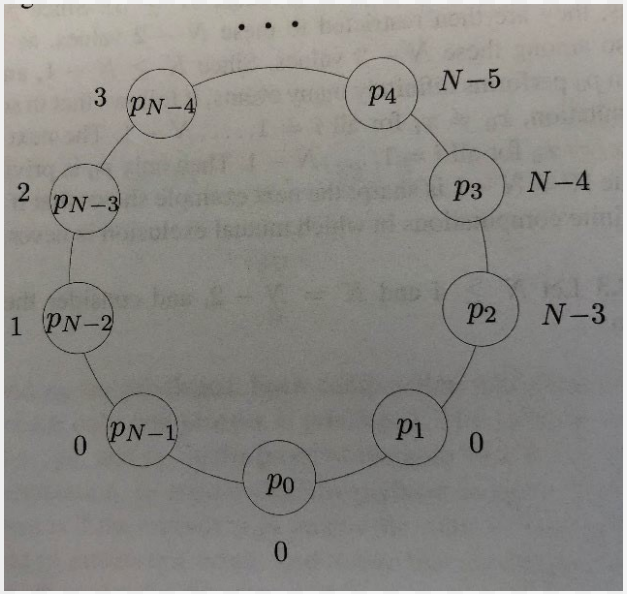
Table 1: Dijkstra's self-stabilization algorithm (n =3 and k =4).

Non-Convergence Example

When n > 3, k = n-2.

Infinite computation exists in which always n-1 processes are privileged





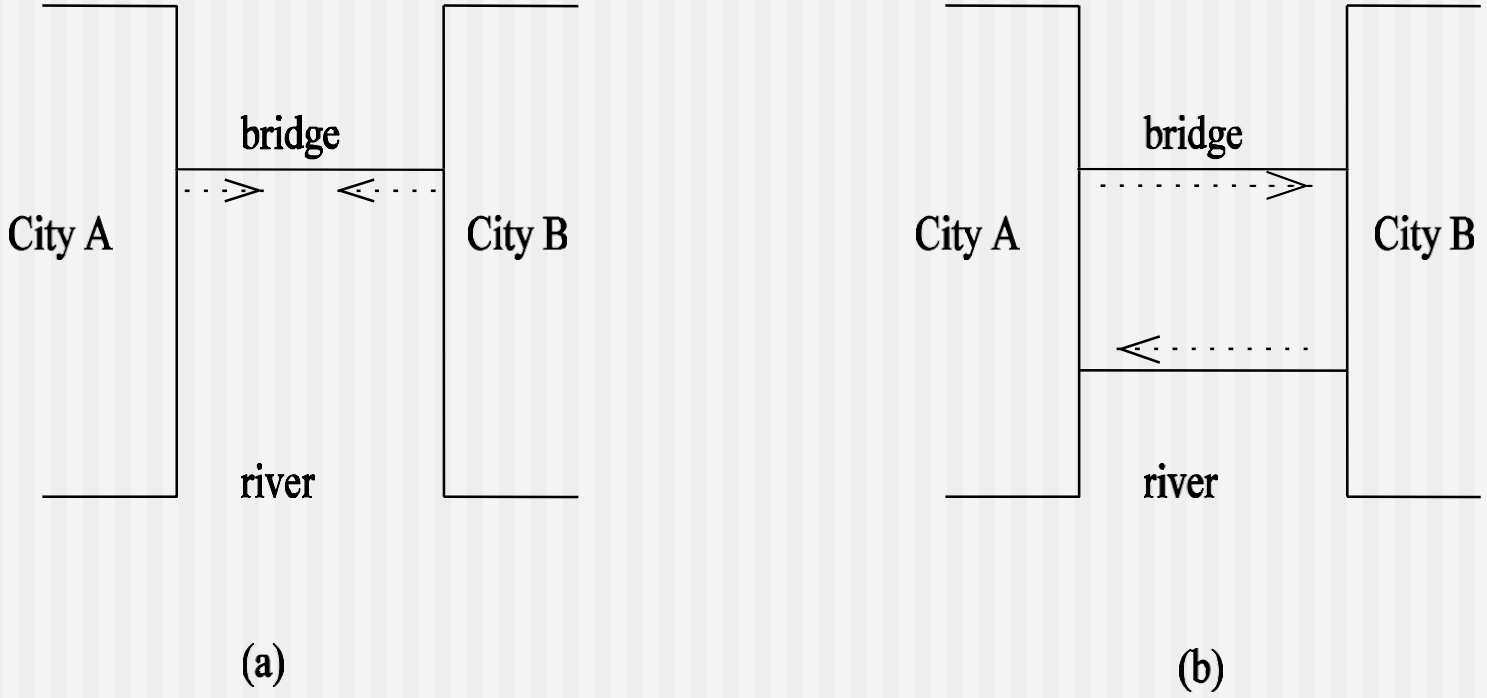
Extensions

- The role of demon (that selects one privileged process)
- The role of asymmetry.
- The role of topology.
- The role of the number of states

Detection and Resolution of Deadlock

- **Mutual exclusion**. No resource can be shared by more than one process at a time.
- **Hold and wait**. There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
- **No preemption**. A resource cannot be preempted.
- **Circular wait**. There is a cycle in the wait-for graph.

Detection and Resolution of Deadlock (Cont'd)



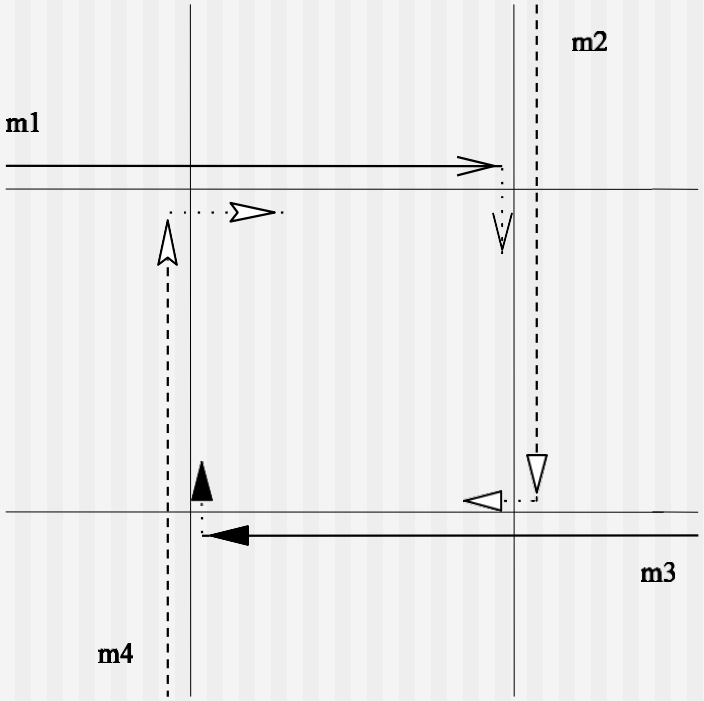
Two cities connected by (a) one bridge and by (b) two bridges.

Strategies for Handling Deadlocks

- Deadlock prevention
- Deadlock avoidance (based on "safe state")
- Deadlock detection and recovery
- Different Models
  - AND condition
  - OR condition

Types of Deadlock

- Resource deadlock
- Communication deadlock



An example of communication deadlock

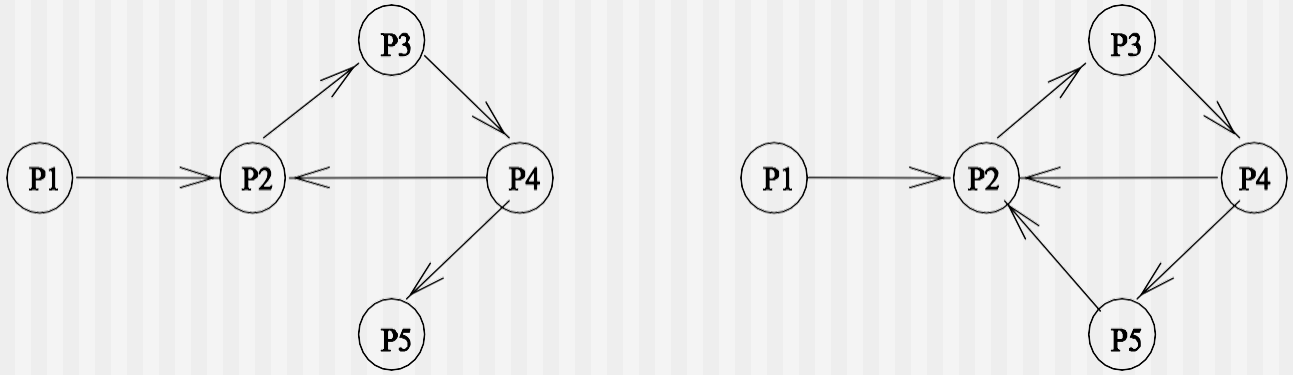
Conditions for Deadlock

- AND model: a **cycle** in the wait-for graph.
- OR model: a **knot** in the wait-for graph.

Wait-for graph is a special case of resource-allocation graph: each resource has exactly one instance.

Conditions for Deadlock (Cont'd)

A knot (K) consists of a set of nodes such that for every node a in K , all nodes in K and only the nodes in K are reachable from node a.



Two systems under the OR condition with
(a) no deadlock and without (b) deadlock.

Focus 12: Rosenkrantz' Dynamic Priority Scheme (using timestamps)

T1:

lock A;
lock B;
transaction starts;
unlock A;
unlock B;

wait-die (non-preemptive method)

[$LC_i < LC_j \rightarrow$ **halt** P_i (wait)
□ $LC_i \geq LC_j \rightarrow$ **kill** P_i (die)
]

wound-wait (preemptive method)

[$LC_i < LC_j \rightarrow$ **kill** P_j (wound)
□ $LCi \geq LCj \rightarrow$ **halt** P_i (wait)
]

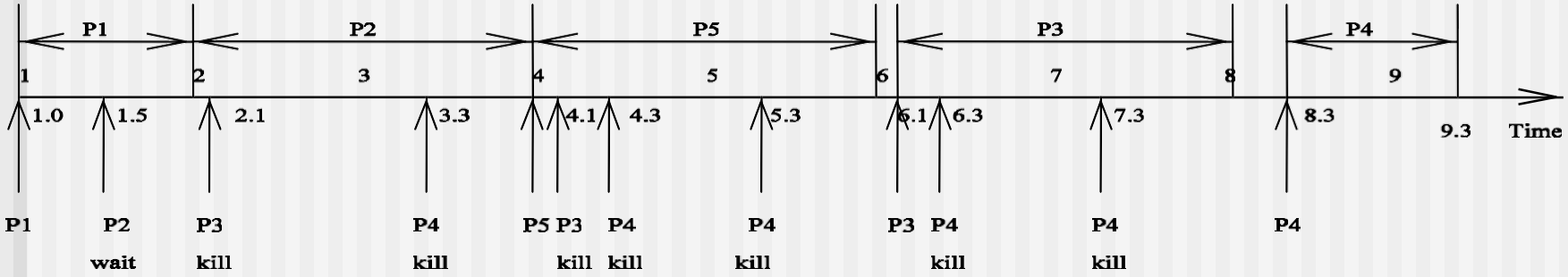
Example 13

Process id	Priority	1st request time	Length	Retry interval
P1	2	1	1	1
P2	1	1.5	2	1
P3	4	2.1	2	2
P4	5	3.3	1	1
P5	3	4.0	2	3

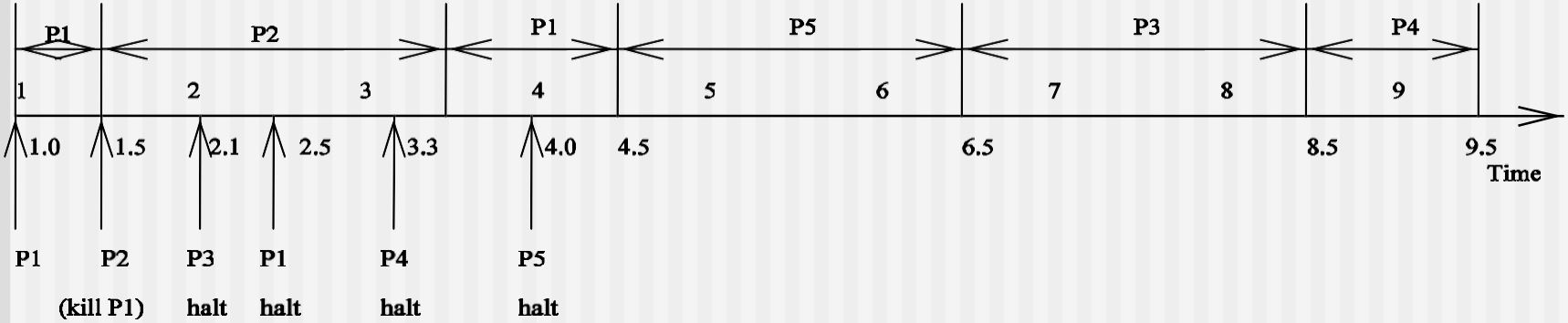
A system consisting of five processes.

Example 13 (Cont'd)

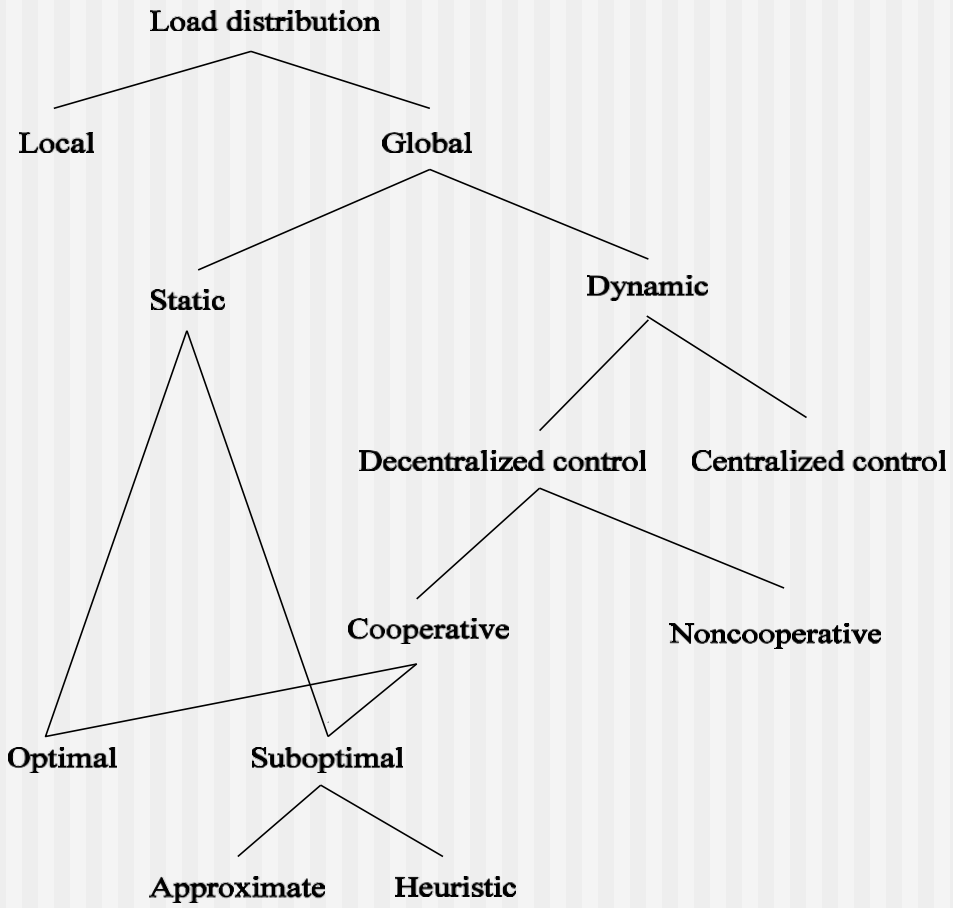
wait-die :



wound-wait:



Load Distribution



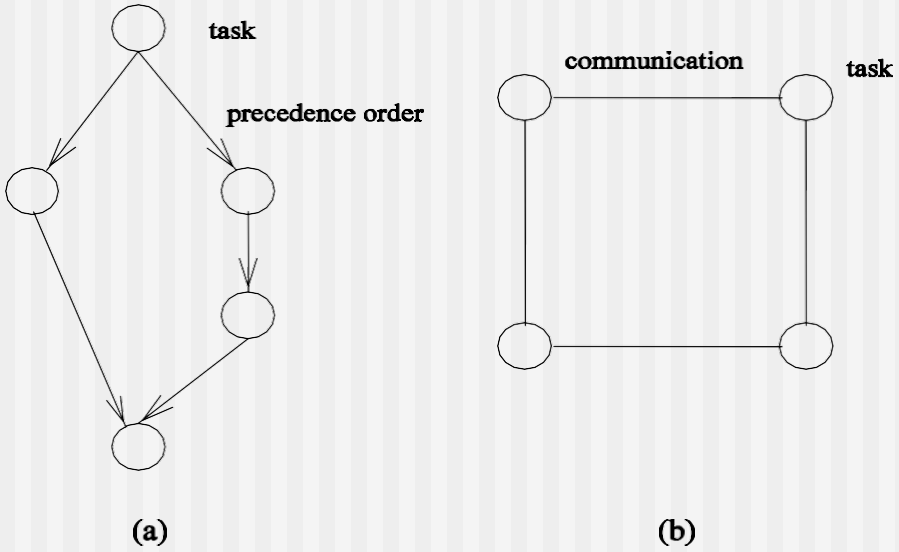
A taxonomy of load distribution algorithms.

Static Load Distribution (task scheduling)

- Processor interconnections
- Task partition
  - Horizontal or vertical partitioning.
  - Communication delay minimization partition.
  - Task duplication.
- Task allocation

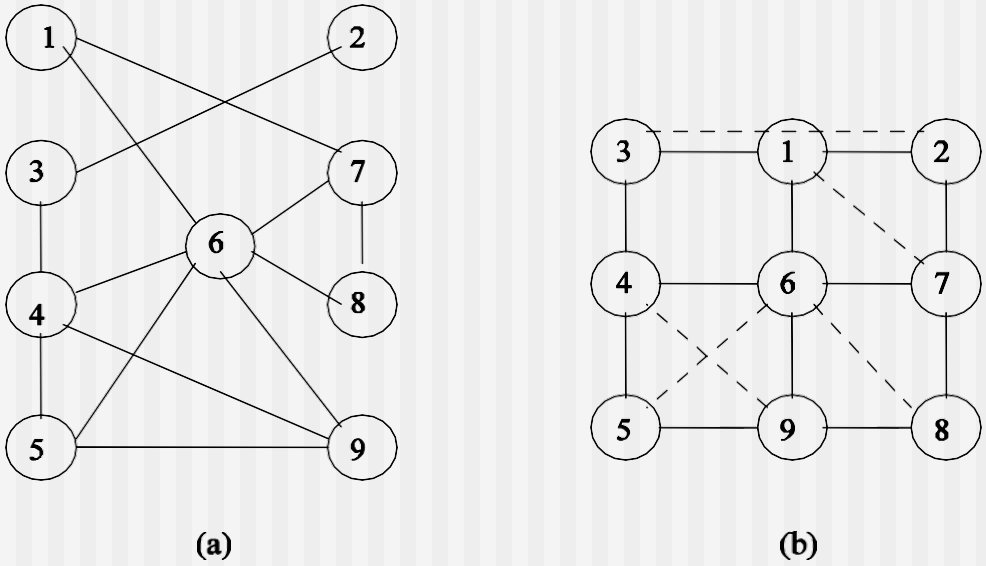
Models

- **Task precedence graph**: each link defines the precedence order among tasks.
- **Task interaction graph**: each link defines task interactions between two tasks.



(a) Task precedence graph (reducing critical path) and (b) task interaction graph (balancing comp. and comm.)

Example 14



Mapping a task interaction graph (a) to a processor graph (b).

Example 14 (Cont'd)

- The *dilation* of an edge of G_t is defined as the length of the path in G_p onto which an edge of G_t is mapped. The dilation of the embedding is the maximum edge dilation of G_t.
- The *expansion* of the embedding is the ratio of the number of nodes in G_t to the number of nodes in G_p.
- The *congestion* of the embedding is the maximum number of paths containing an edge in G_p where every path represents an edge in G_t.
- The *load* of an embedding is the maximum number of processes of G_t assigned to any processor of G_t.

Periodic Tasks With Real-time Constraints

- Task T_i has request period t_i and run time c_i.
- Each task has to be completed before its next request.
- All tasks are independent without communication.

Liu and Layland's Solutions

- *Rate monotonic scheduling* (RMS, fixed priority assignment). Tasks with higher request rates will have higher priorities.
- *Deadline driven scheduling* (DDS, dynamic priority assignment). A task will be assigned the highest priority if the deadline of its current request is the nearest.

Both are preemptive

RMS is easier to implement compared to DDS.

Schedulability

- Deadline driven schedule: iff

$$\sum_{i=0}^{n} c_i/t_i \leq 1$$

- Rate monotonic schedule: if

$$\sum_{i=0}^{n} c_i/t_i \leq n(2^{1/n} - 1);$$

may or may be not when

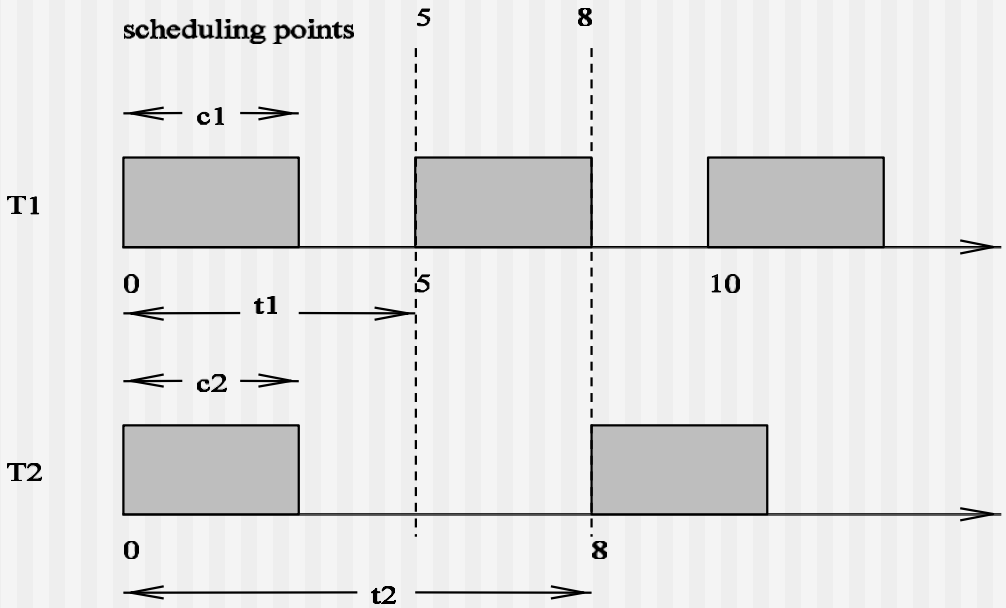
$$n(2^{1/n} - 1) < \sum_{i=0}^{n} c_i/t_i \leq 1$$

Example 15 (schedulable)

- T_1: $c_1 = 3$, $t_1 = 5$ and T_2: $c_2 = 2$, $t_2 = 7$ (with the same initial request time).
- The overall utilization is $0.887 > 0.828$ (bound for $n = 2$),
- but, it is schedulable.

Example 16 (un-schedulable under rate monotonic scheduling)

- T_1: $c_1 = 3$, $t_1 = 5$ and T_2: $c_2 = 3$, $t_2 = 8$ (with the same initial request time).
- The overall utilization is $0.975 > 0.828$



An example of periodic tasks that is not schedulable.

Example 16 (Cont'd)

- If each task meets its first deadline when all tasks are started at the same time then the deadlines for all tasks will always be met for any combination of starting times.
- *scheduling points* for task T : T 's first deadline and the ends of periods of higher priority tasks prior to T 's first deadline.
- If the task set is schedulable for one of scheduling points of the lowest priority task, the task set is schedulable; otherwise, the task set is not schedulable.

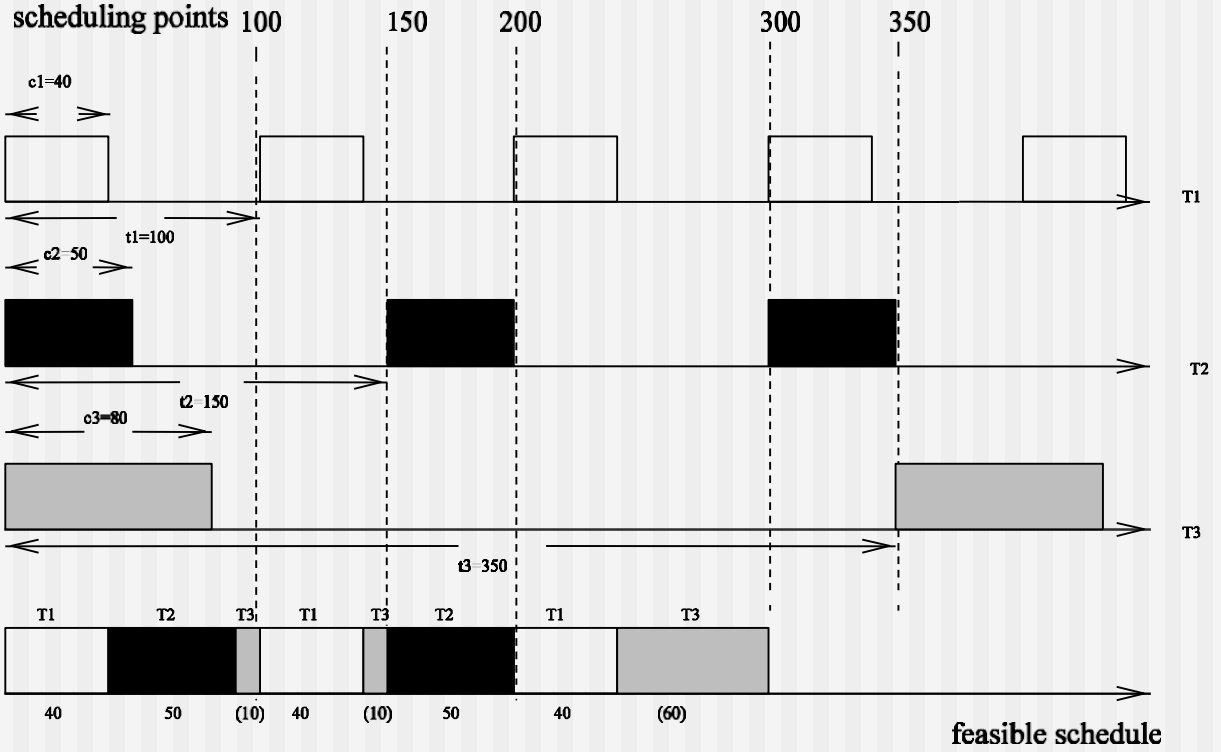
Example 17 (schedulable under rate monotonic schedule)

- $c_1 = 20$, $t_1 = 100$, $c_2 = 50$, $t_2 = 150$, and $c_3 = 80$, $t_3 = 350$.
- The overall utilization is $0.2 + 0.333 + 0.229 = 0.762 < 0.779$ (the bound for $n > 3$).
- c_1 is doubled to 40. The overall utilization is $0.4+0.333+0.229 = 0.962 > 0.779$.
- The scheduling points for T_3: 350 (for T_3), 300 (for T_1 and T_2), 200 (for T_1), 150 (for T_2), 100 (for T_1).

Example 17 (Cont'd)

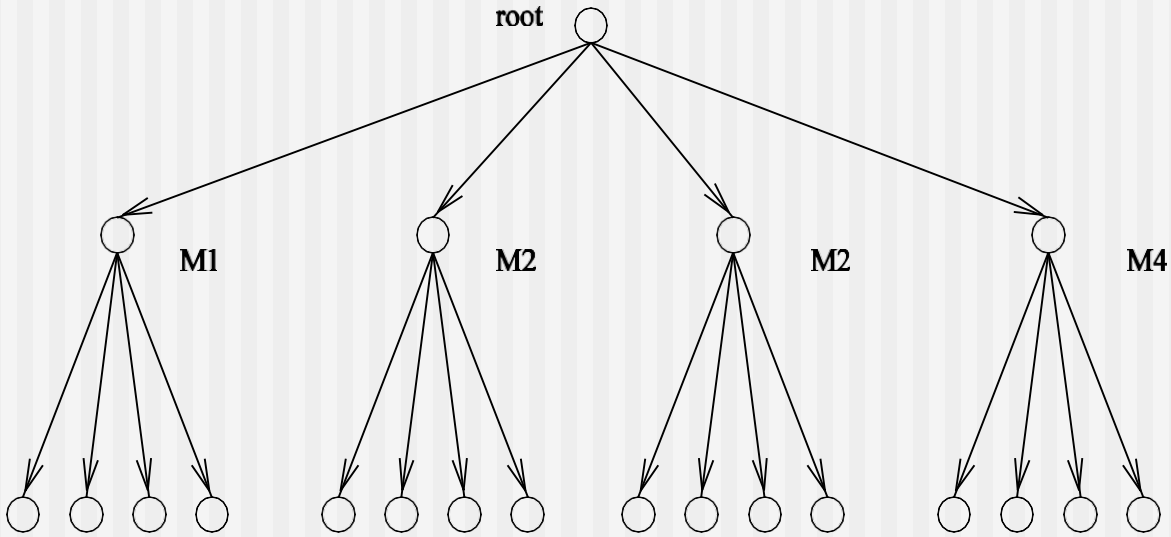
$c_1 + c_2 + c_3 \leq t_1$,
$40 + 50 + 80 > 100$;
$2c_1 + c_2 + c_3 \leq t_2$,
$80 + 50 + 80 > 150$;
$2c_1 + 2c_2 + c_3 \leq 2t_1$,
$80 + 100 + 80 > 200$;
$3c_1 + 2c_2 + c_3 \leq 2t_2$,
$120 + 100 + 80 = 300$;
$4c_1 + 3c_2 + c_3 \leq t_3$,
$160 + 150 + 80 > 350$.

Example 17 (Cont'd)



A schedulable periodic task.

Dynamic Load Distribution (load balancing)



A state-space traversal example.

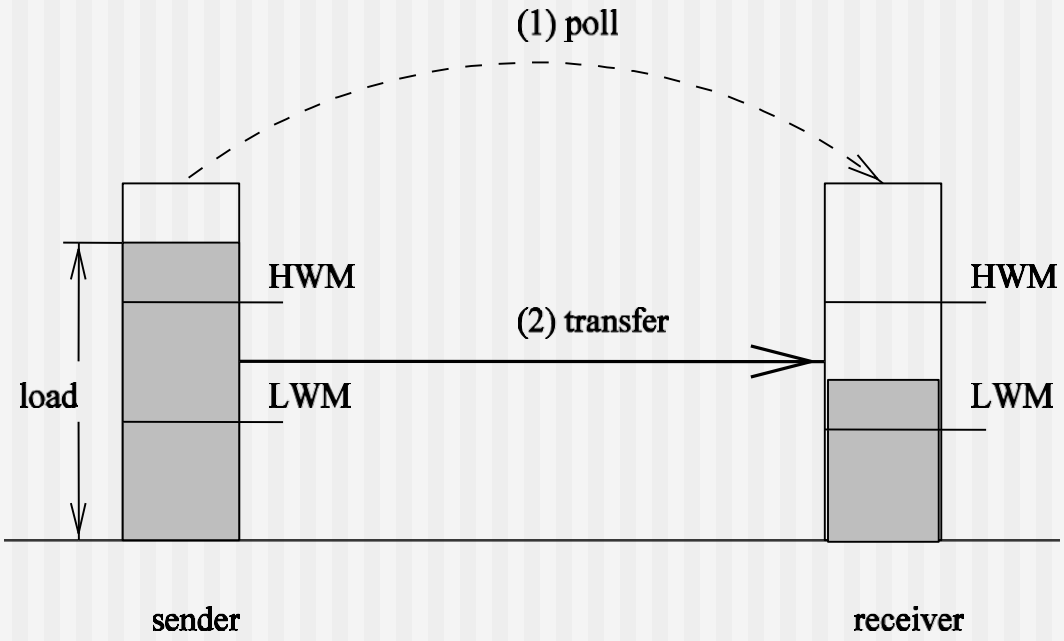
Dynamic Load Distribution (Cont'd)

A dynamic load distribution algorithm has six policies:

- Initiation
- Transfer
- Selection
- Profitability
- Location
- Information

Focus 13: Initiation

Sender-initiated approach:



Sender-initiated load balancing.

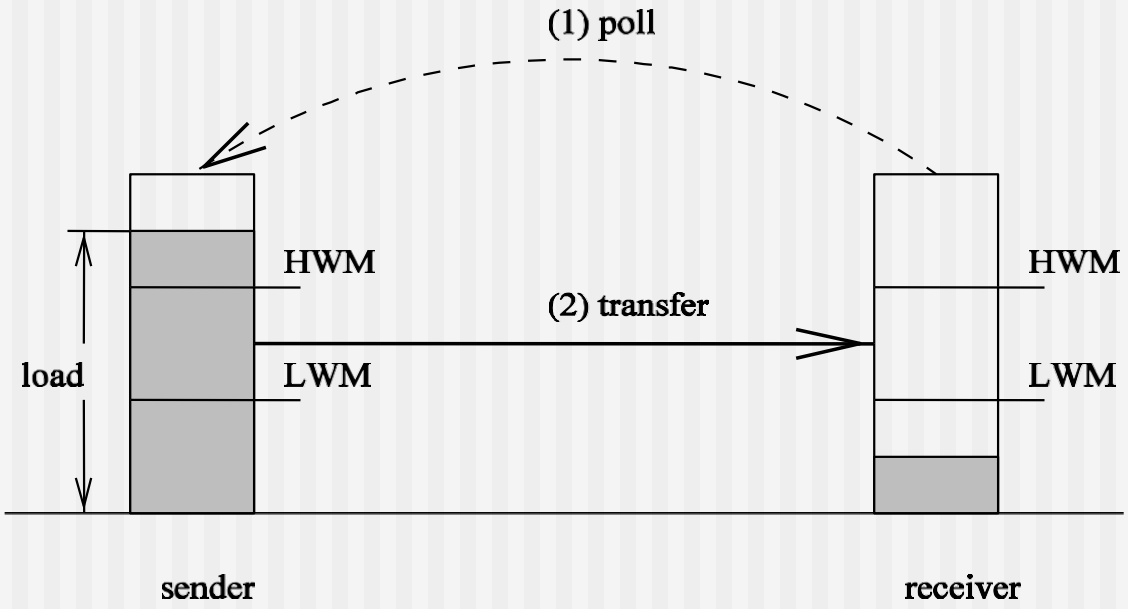
Focus 13 (Cont'd)

```
/* a new task arrives */
 queue length ≥ HWM →

* [ poll_set := φ;

    [| poll_set | < poll_limit →
          [ select a new node u randomly;
            poll_set := poll_set ∪ node u;
            queue_length at u < HWM →
                  transfer a task to node u and stop
          ]
    ]
 ]
```

Receiver-Initiated Approach

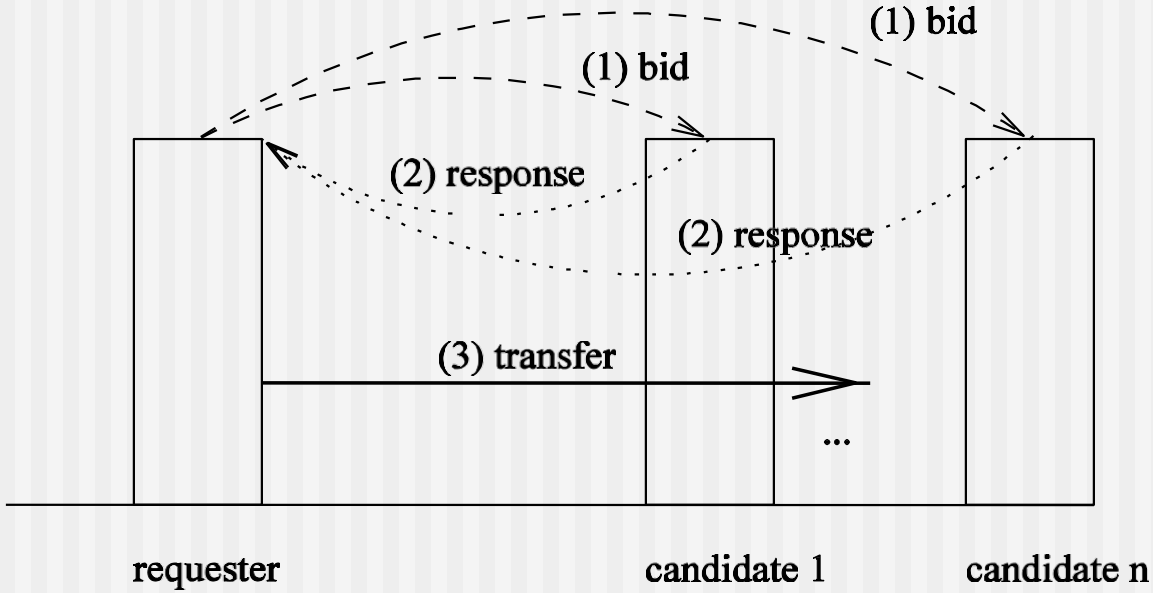


Receiver-initiated load balancing.

Receiver-Initiated Approach (Cont'd)

```
/* a task departs */
queue length < LWM →
[ poll limit:=φ ;
    *  [ | poll_set | < poll limit →
            [ select a new node u randomly;
              poll_set := poll set ∪ node u;
              queue_length at u > HWM →
                                  transfer a task from node u and stop
            ]
       ]
]
```

Bidding Approach



Bidding algorithm.

Focus 14: Sample Nearest Neighbor Algorithms

Diffusion

- At round t + 1 each node u exchanges its load $L_u(t)$ with its neighbors' $L_v(t)$.
- $L_u(t + 1)$ should also include new incoming load $\varphi_u(t)$ between rounds t and t + 1.
- Load at time t + 1:

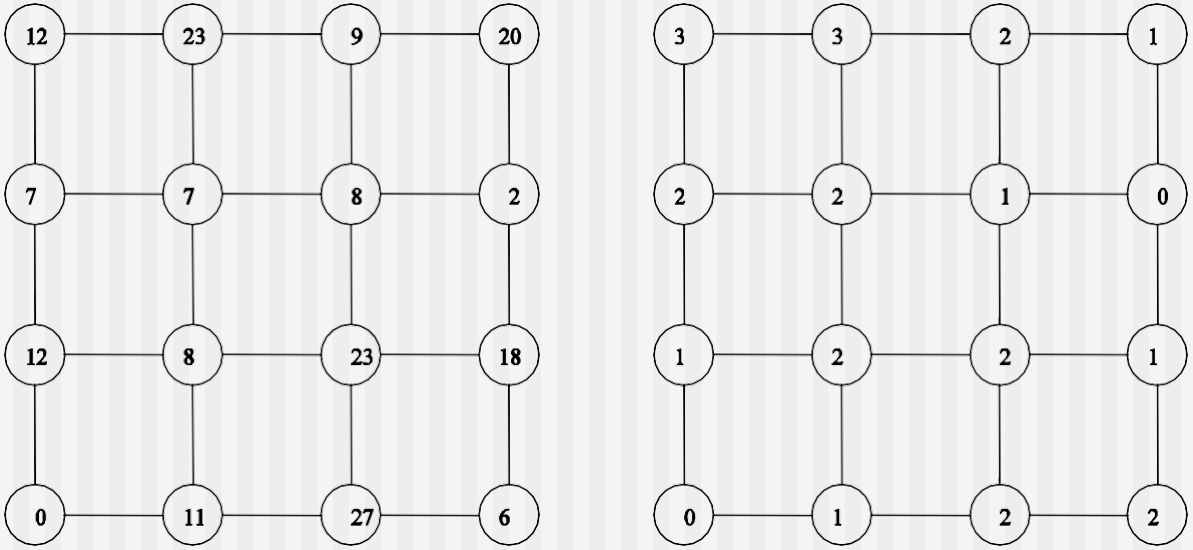
$$L_u(t + 1) = L_u(t) + \sum_{v \in A(u)} \alpha_{u,v}(L_v(t) - L_u(t)) + \varphi_u(t)$$

where $0 \leq \alpha_{u,v} \leq 1$ is called the diffusion parameter of nodes u and v.

Gradient

- Maintain a contour of the gradients formed by the differences in load in the system.
- Load in high points (overloaded nodes) of the contour will flow to the lower regions (underloaded nodes) following the gradients.
- The *propagated pressure* of a processor u, $p(u)$, is defined as p(u) =
  - 0 (if u is lightly loaded)
  - $1 + \min\{p(v) | v \in A(u)\}$ (otherwise)

Gradient (Cont'd)

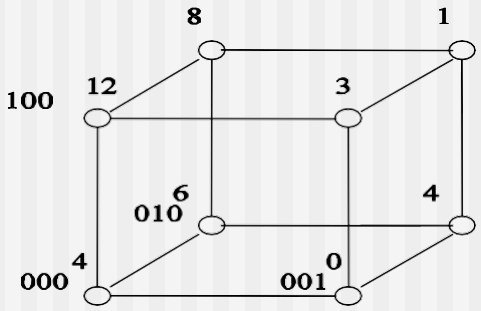


(a) A 4 x 4 mesh with loads. (b) The corresponding propagated pressure of each node (a node is lightly loaded if its load is less than 3).

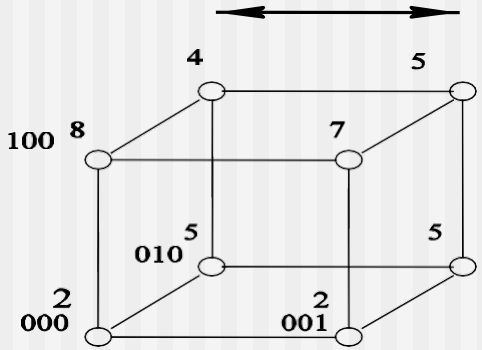
Dimension Exchange: Hypercubes

- A sweep of dimensions (rounds) in the n-cube is applied.
- In the i^{th} round neighboring nodes along the i^{th} dimension compare and exchange their loads.

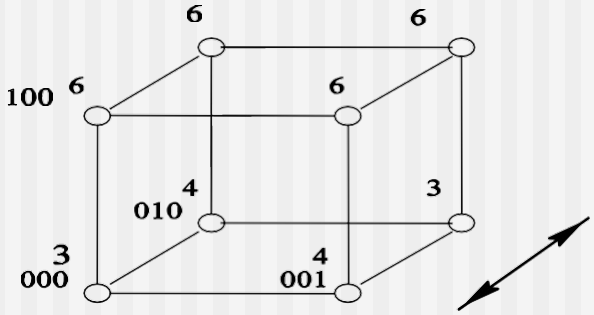
Dimension Exchange: Hypercubes (Cont'd)



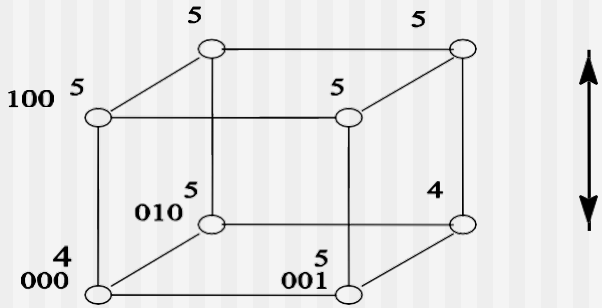
(a) initial load distribution



(b) dimension one exchange



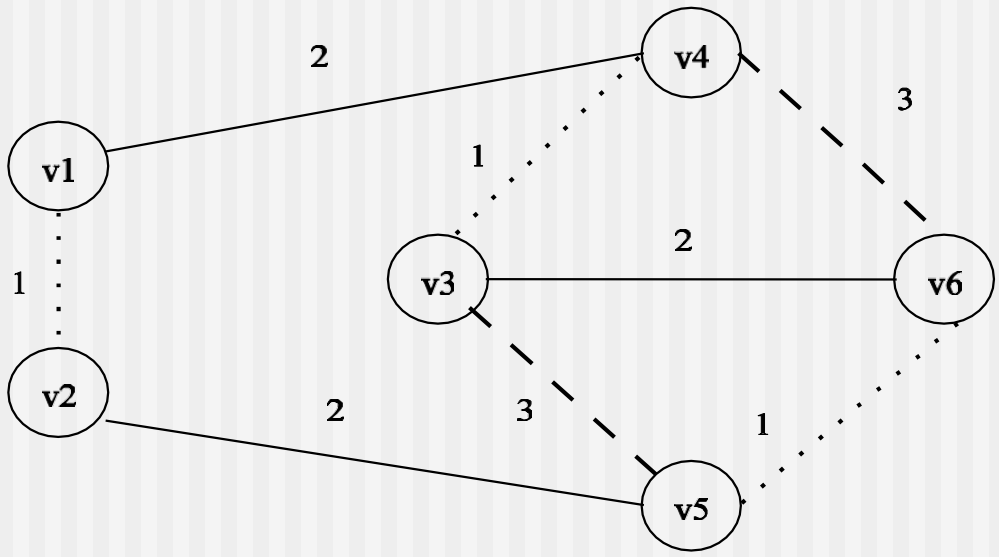
(c) dimension two exchange



(d) dimension three exchange

Load balancing on a healthy 3-cube.

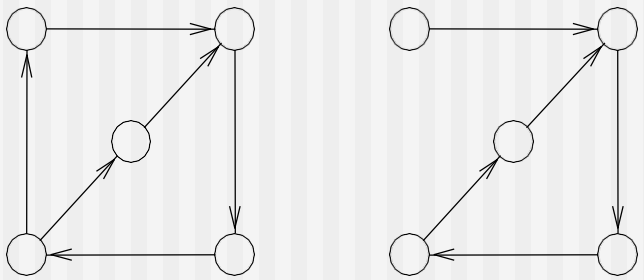
Extended Dimension Exchange: Edge-Coloring



Extended dimension exchange model through edge-coloring.

Exercise 4

1. Apply *wound-wait* and *wait-die* schemes to the example shown in Table 2.
2. Show the state transition sequence for the following system with $n = 3$ and $k = 5$ using Dijkstra's self-stabilizing algorithm. Assume that $P_0 = 3$, $P_1 = 1$, and $P_2 = 4$.
3. Determine if there is a deadlock in each of the following wait-for graphs assuming the OR model is used.



(a) (b)

Exercise 4 (Cont'd)

Process id	Priority	1st request time	Length	Retry interval	Resource(s)
P1	3	1	1	1	A
P2	4	1.5	2	1	B
P3	1	2.5	2	2	A,B
P4	2	3	1	1	B,A

Table 2: A system consisting of four processes.

4. Consider the following two periodic tasks (with the same request time)

- Task T_1: $c_1 = 4$, $t_1 = 9$
- Task T_2: $c_2 = 6$, $t_2 = 14$

(a) Determine the total utilization of these two tasks and compare it with Liu and Layland's least upper bound for the fixed priority schedule. What conclusion can you derive?

Exercise 4 (Cont'd)

(b) Show that these two tasks are schedulable using the rate-monotonic priority assignment. You are required to provide such a schedule.

(c) Determine the schedulability of these two tasks if task T_2 has a higher priority than task T_1 in the fixed priority schedule.

(d) Split task T_2 into two parts of 3 units computation each and show that these two tasks are schedulable using the rate-monotonic priority assignment.

(e) Provide a schedule (from time unit 0 to time unit 30) based on deadline driven scheduling algorithm. Assume that the smallest preemptive element is one unit.

Exercise 4 (Cont'd)

5. For the following 4 x 4 mesh find the corresponding propagated pressure of each node. Assume that a node is considered lightly loaded if its load is less than 2.