LECTURE 19
PROCESSORS ALLOCATION

Processor Allocation

- determine which process is assigned to which processor. Also called load distribution.
- Two categories:
- Static load distribution-nonmigratory, once allocated, can not move, no matter how overloaded the machine is.
- Dynamic load distribution-migratory, can move even if the execution started. But algorithm is complex.

The goals of allocation

- 1 Maximize CPU utilization
- 2 Minimize mean response time/ Minimize response ratio

  Response ratio-the amount of time it takes to run a process on some machine, divided by how long it would take on some unloaded benchmark processor. E.g. a 1-sec job that takes 5 sec. The ratio is 5/1.

Design issues for processor allocation algorithms

- Deterministic versus heuristic algorithms
- Centralized versus distributed algorithms
- Optimal versus suboptimal algorithms
- Local versus global algorithms
- Sender-initiated versus receiver-initiated algorithms

How to measure a processor is overloaded or underloaded?

- 1 Count the processes in the machine? Not accurate because even the machine is idle there are some daemons running.
- 2 Count only the running or ready to run processes? Not accurate because some daemons just wake up and check to see if there is anything to run, after that, they go to sleep. That puts a small load on the system.
- 3 Check the fraction of time the CPU is busy using time interrupts. Not accurate because when CPU is busy it sometimes disable interrupts.

Cont..

- How to deal with overhead?

  A proper algorithm should take into account the CPU time, memory usage, and network bandwidth consumed by the processor allocation algorithm itself.

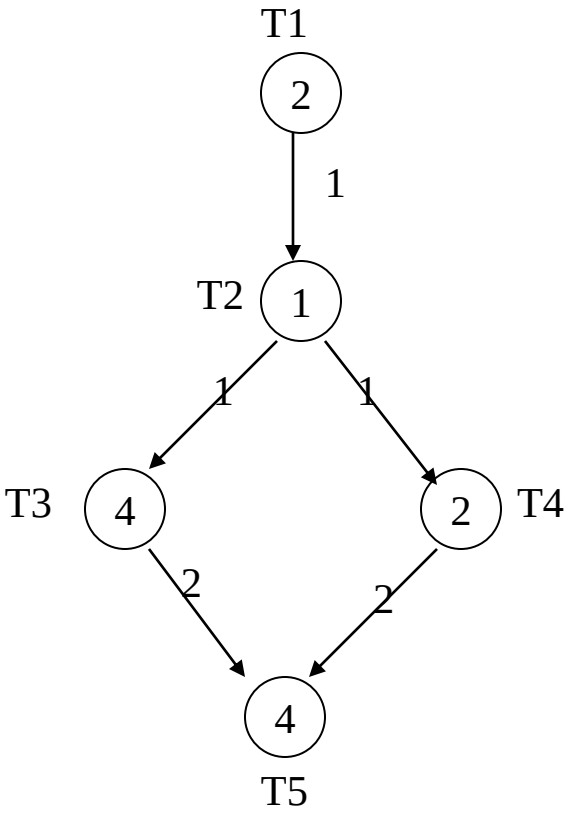
- How to calculate complexity?

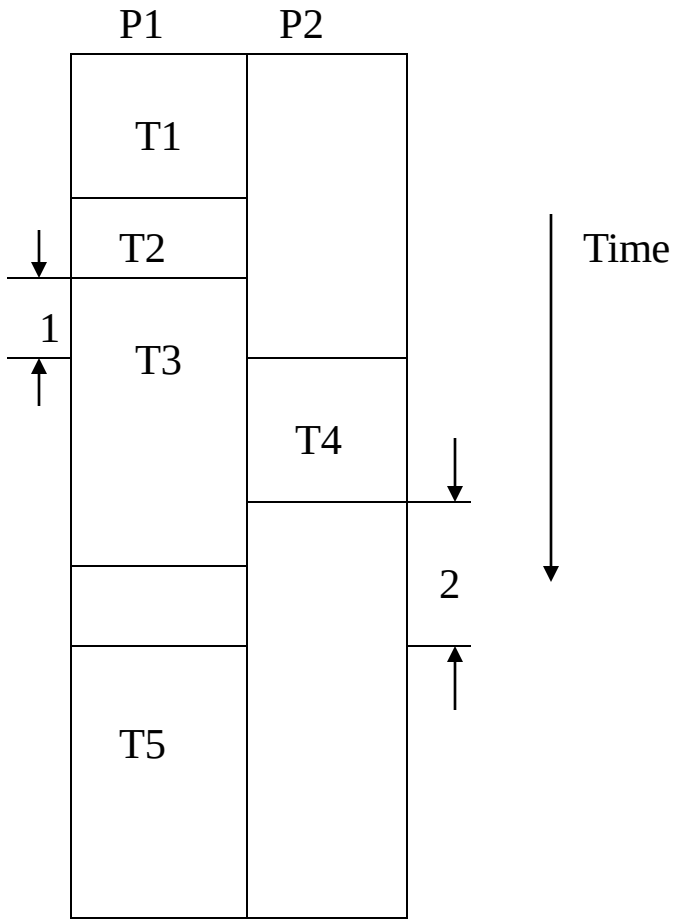
  If an algorithm performs a little better than others but requires much complex implementation, better use the simple one.

- How to deal with stability?

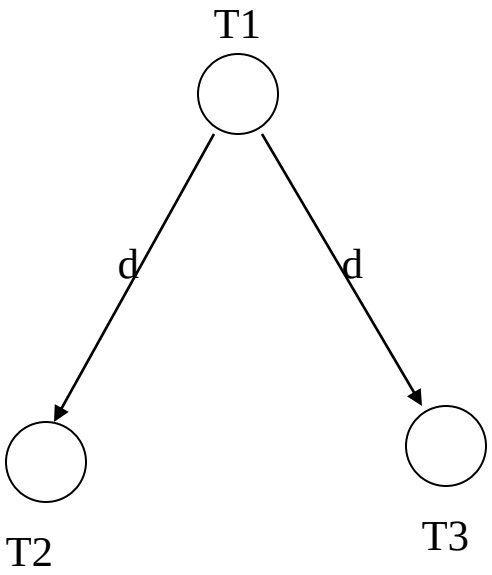
  Problems can arise if the state of the machine is not stable yet, still in the process of updating.

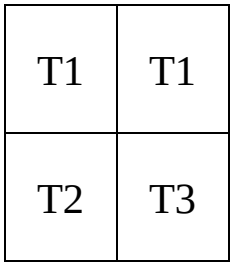
Load distribution based on precedence graph

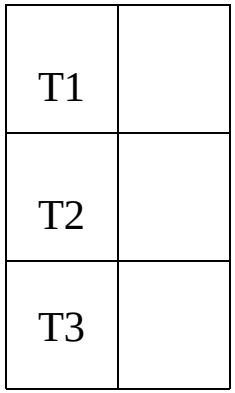


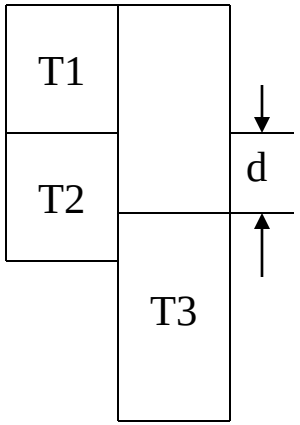


Cont..



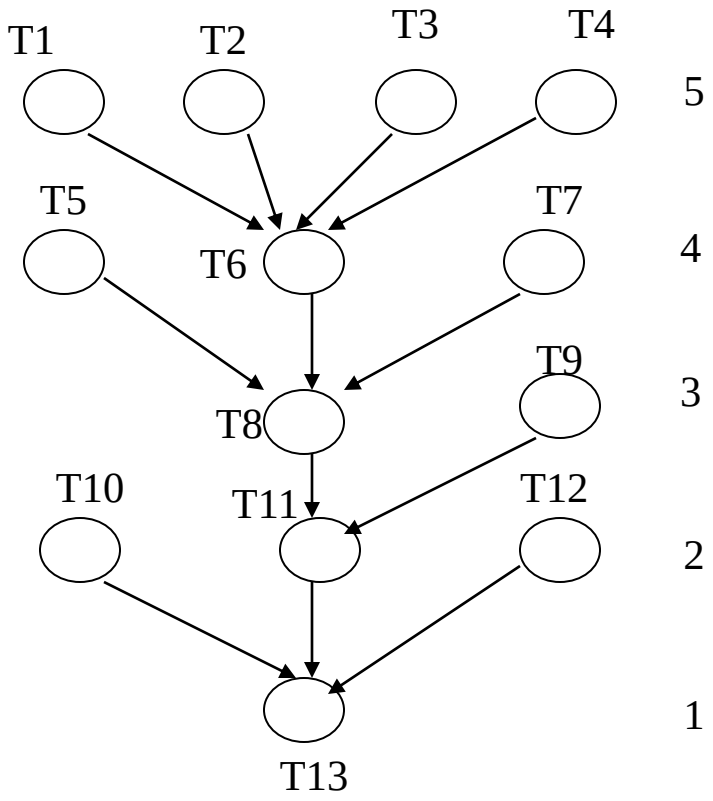






Two Optimal Scheduling Algorithms

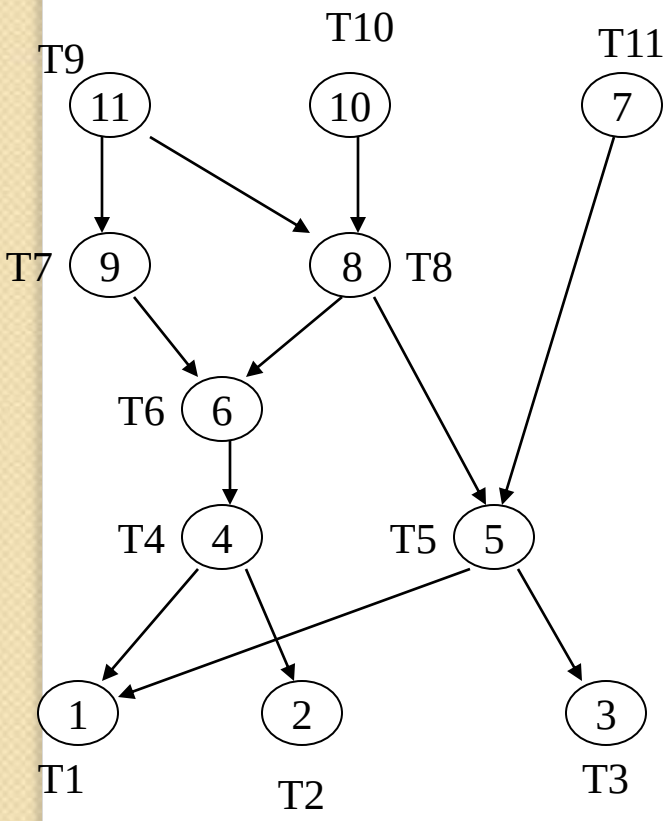
- The precedence graph is a tree



T1	T2	T3
T4	T5	T7
T6	T9	T10
T8	T12	
T11		
T13		

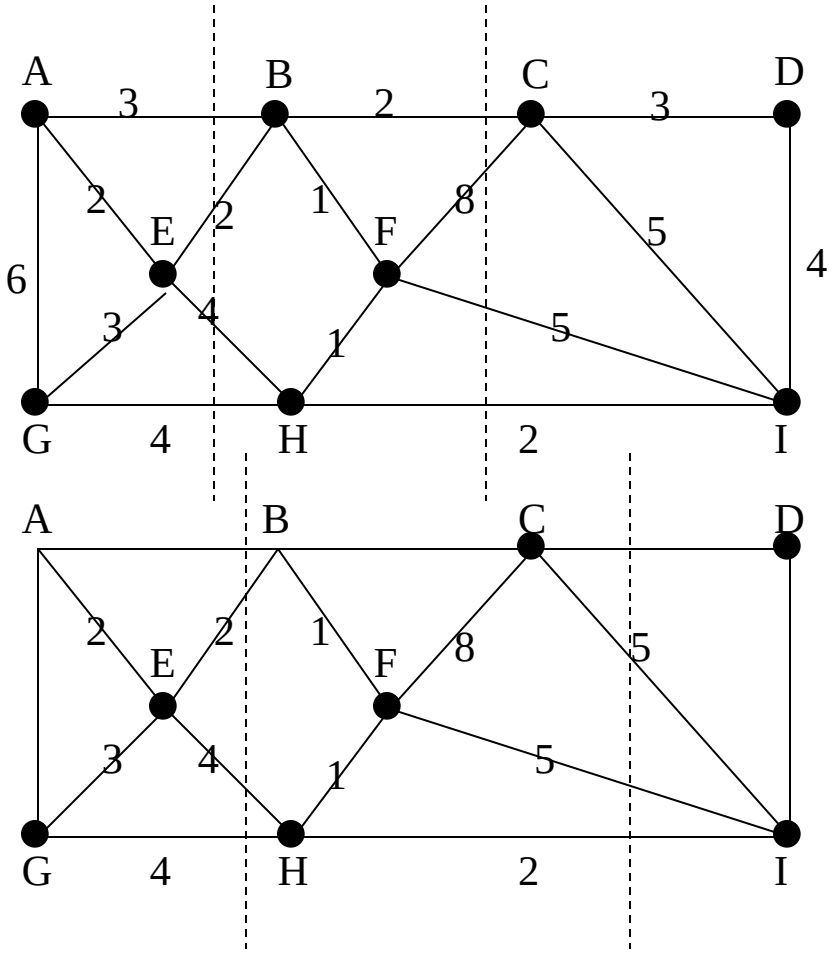
Cont..

- There are only two processors



T9	T10
T7	T8
T11	T6
T5	T4
T3	T2
T1	

A graph-theoretic deterministic algorithm



Total network traffic: 2+4+3+4+2+8+5+2 =30

Total network traffic: 3+2+4+4+3+5+5+2 = 28

Dynamic Load Distribution

- **Components of dynamic load distribution**
- Initiation policy
- Transfer policy
- Selection policy
- Profitability policy
- Location policy
- Information policy

Dynamic load distribution algorithms

- Load balancing algorithms can be classified as follows:
- Global vs. Local
- Centralized vs. decentralized
- Noncooperative vs. cooperative
- Adaptive vs. nonadaptive

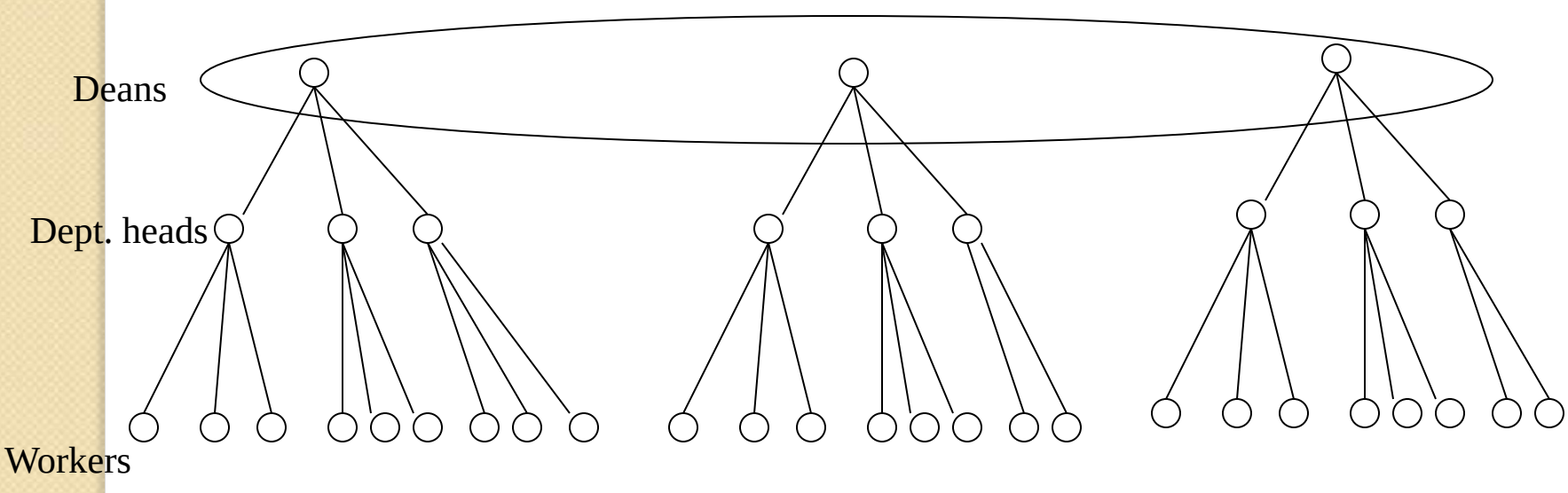
A centralized algorithm

- **Up-down algorithm**: a coordinator maintains a **usage table** with one entry per personal workstation.

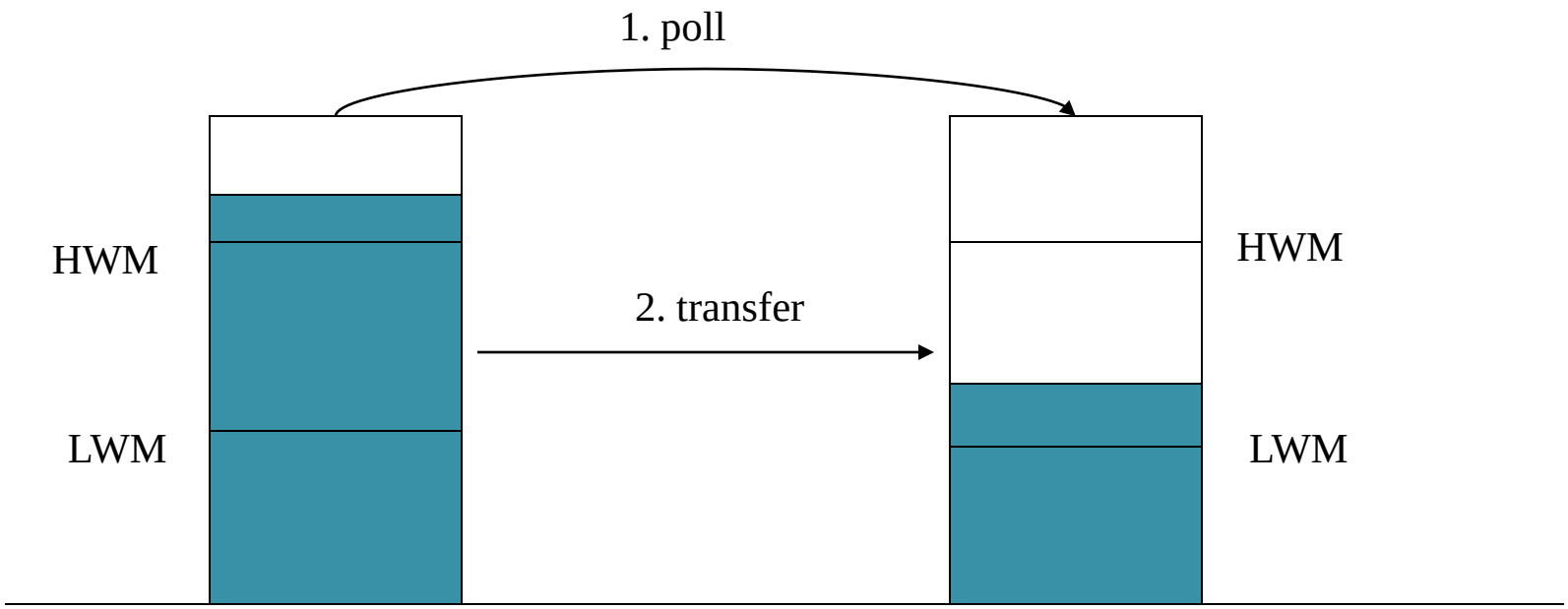
1. When a workstation owner is running processes on other people's machines, it accumulates penalty points, a fixed number per second. These points are added to its usage table entry.
2. When it has unsatisfied requests pending, penalty points are subtracted from its usage table entry.

- A positive score indicates that the workstation is a net user of system resources, whereas a negative score means that it needs resources. A zero score is neutral.
- When a processor becomes free, the pending request whose owner has the lowest score wins.

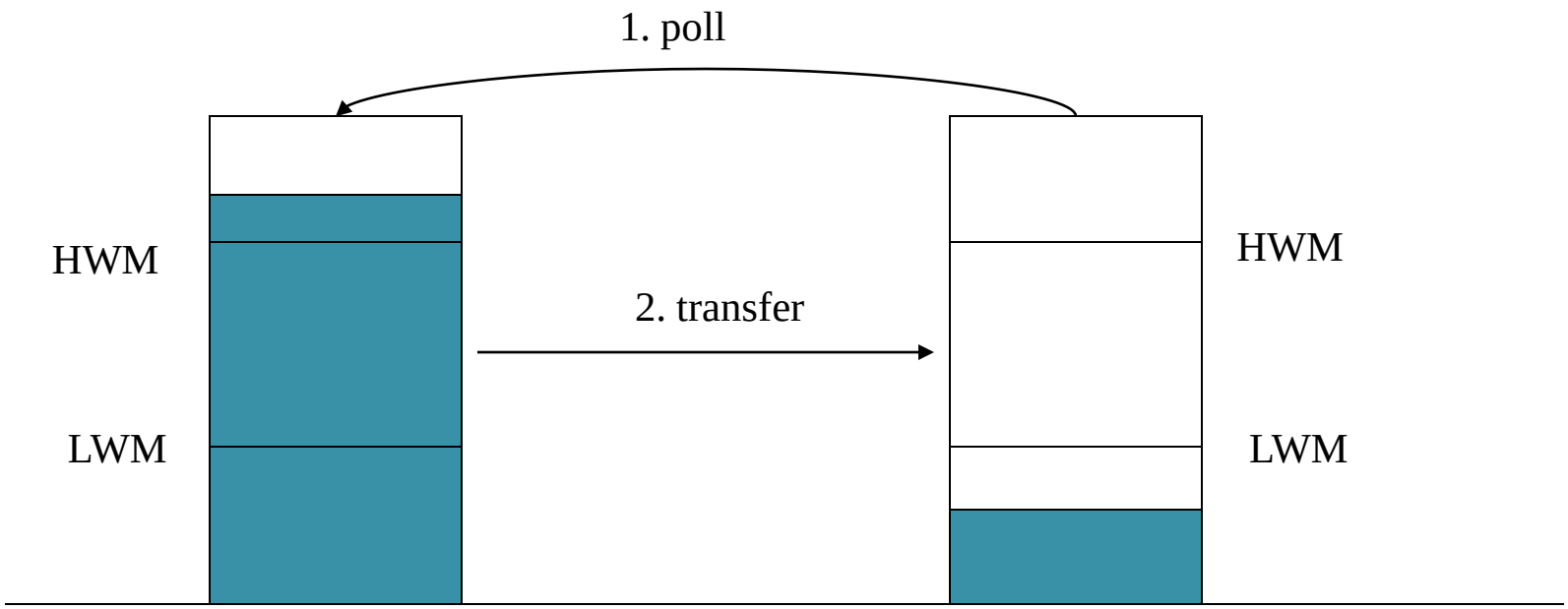
A hierarchical algorithm



A sender-initiated algorithm



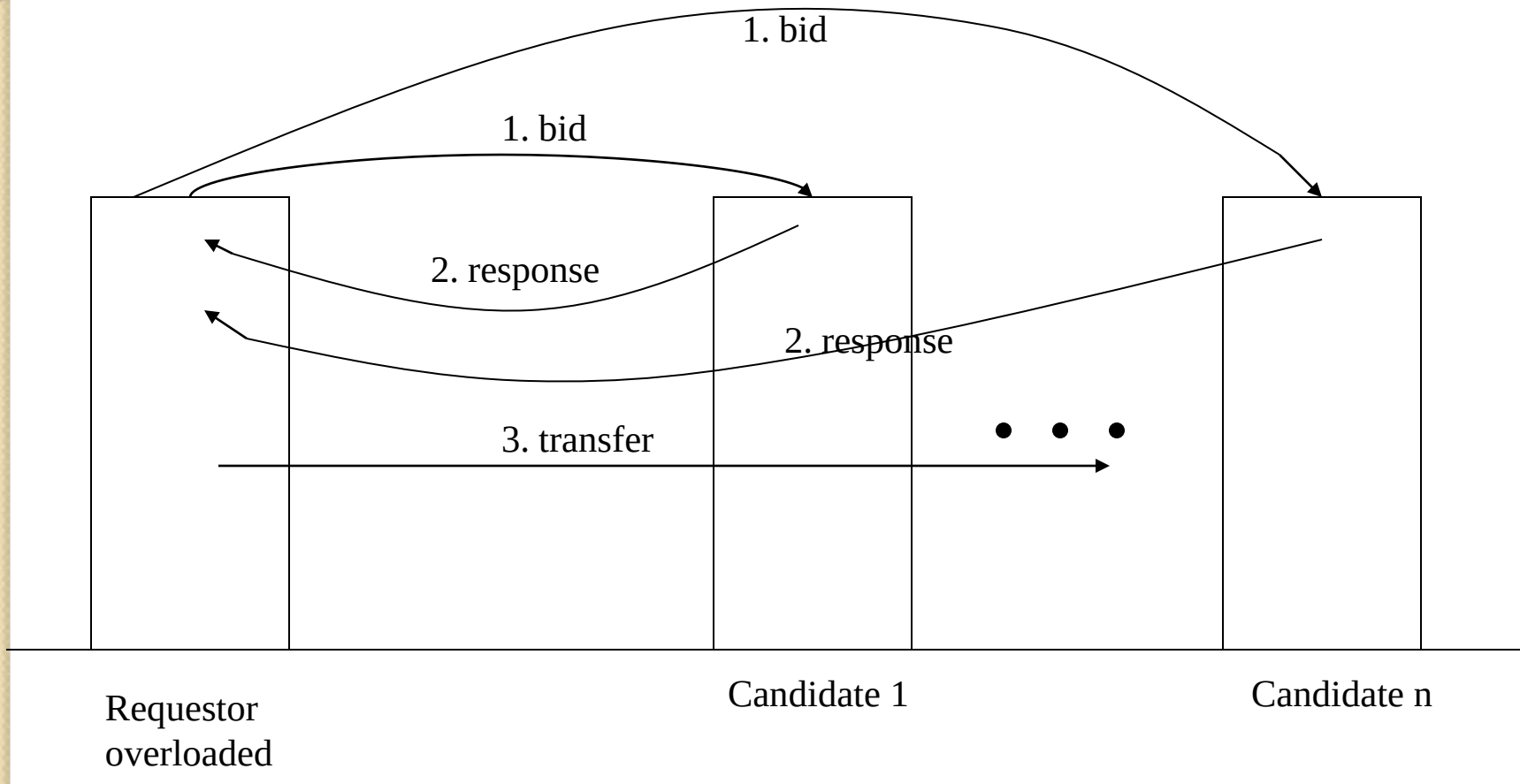
A receiver-initiated algorithm



A bidding algorithm

- This acts like an economy. Processes want CPU time. Processors give the price. Processes pick up the process that can do the work and at a reasonable price and processors pick up the process that gives the highest price.

Bidding algorithm



Cont..

- *Iterative* (also called *nearest neighbor*) *algorithm*: rely on successive approximation through load exchanging among neighboring nodes to reach a global load distribution.
- *Direct algorithm:* determine senders and receivers first and then load exchanges follow.

Direct algorithm

- the average system load is determined first. Then it is broadcast to all the nodes in the system and each node determines its status: overloaded or underloaded. We can call an overloaded node a *peg* and an underloaded node a *hole*.
- the next step is to fill holes with pegs preferably with minimum data movements.

Nearest neighbor algorithms: diffusion

- $L_u(t+1)=L_u(t)+\Sigma_{v\ \varepsilon\ A(u)}\ (\alpha_{u,v}\ (L_v(t)-L_u(t))+\Phi_u(t))$

- A(u) is the neighbor set of u.

  $0<=\alpha_{u,v}<=1$ is the diffusion parameter which determines the amount of load exchanged between two neighboring nodes u and v.

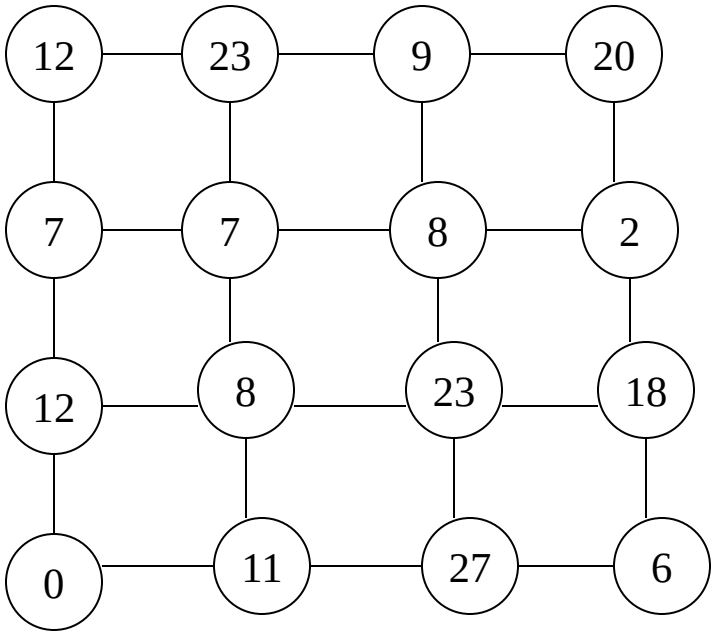
$\Phi_u(t))$ is the new incoming load between t and t+1.

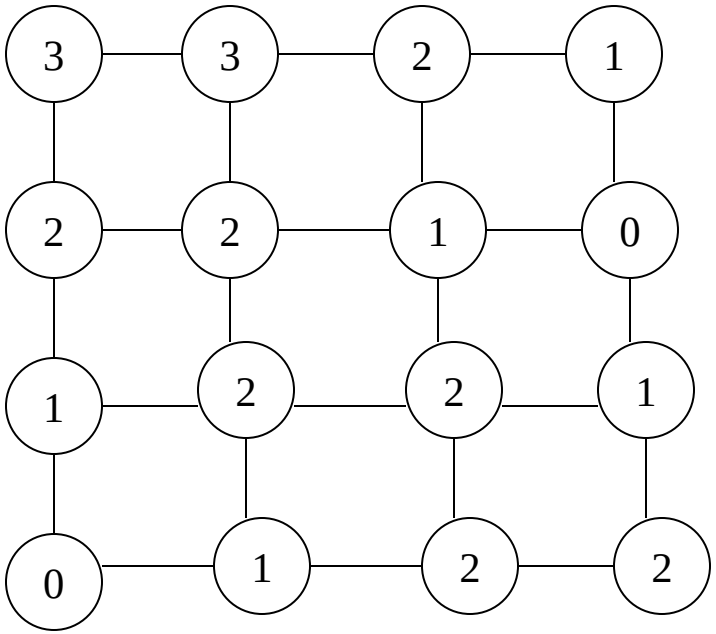
Nearest neighbor algorithm: gradient

- One of the major issues is to define a reasonable contour of gradients. The following is one model. The *propagated pressure* of a processor u, p(u), is defined as

- If u is lightly loaded, p(u) = 0
- Otherwise, p(u) = 1 + min{p(v)|v ϵ A(u)}

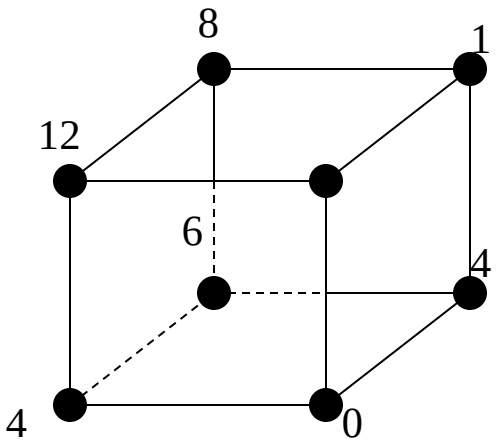
Nearest neighbor algorithm: gradient

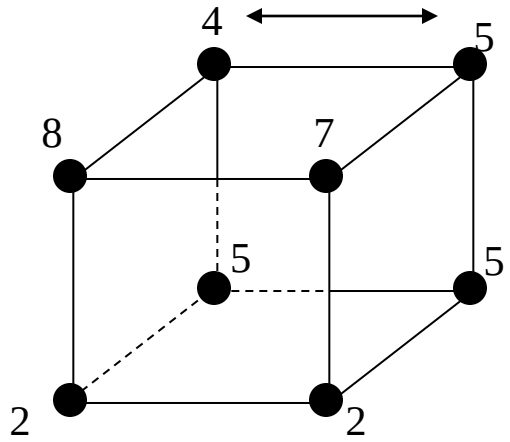


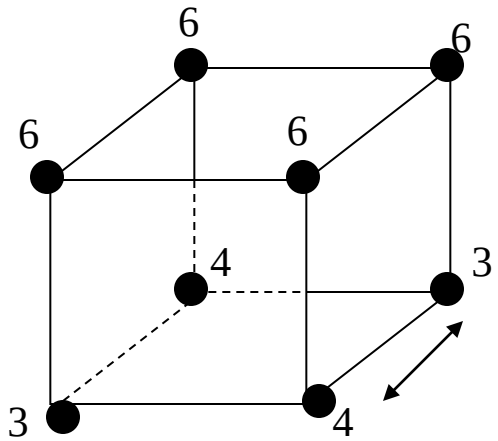


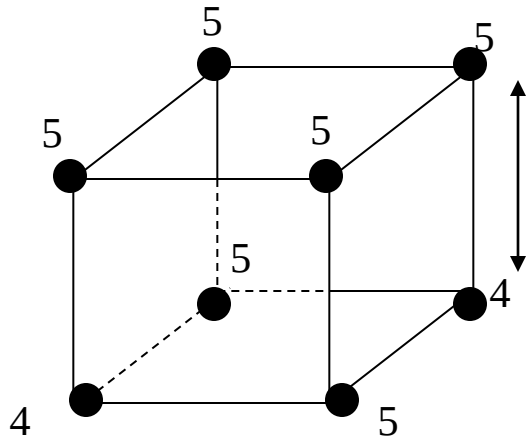
A node is lightly loaded if its load is < 3.

Nearest neighbor algorithm: dimension exchange

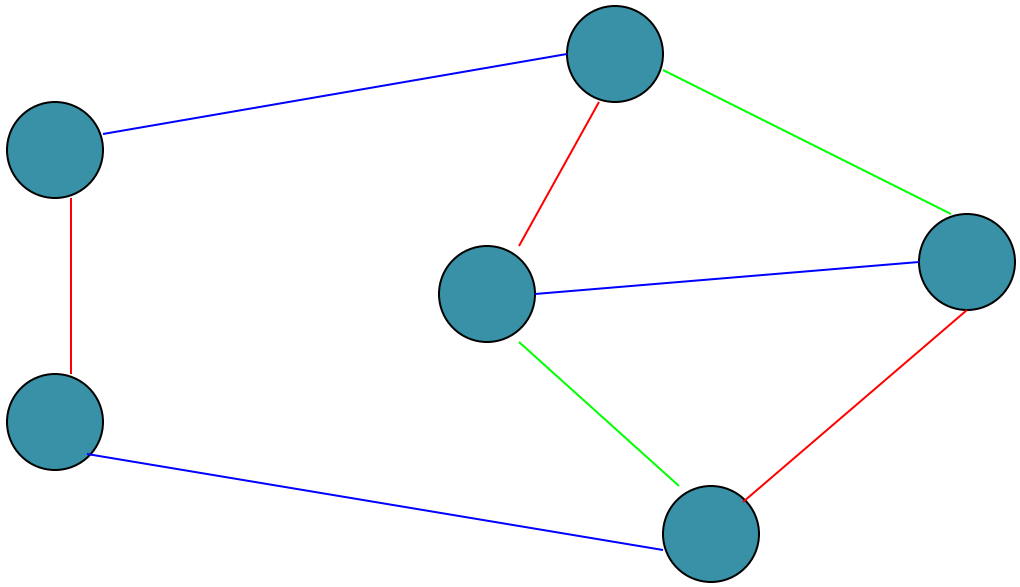








Nearest neighbor algorithm: dimension exchange extension



ASSIGNMENT

- Q: Explain various processor allocation algorithms in distributed system.