# Chapter 9

Load Balancing Problem

## Assumptions

- Programs are dynamic, i.e. threads can be created and deleted at any time and at any place.
- The parallel machine is not empty.
- We employ multiprogramming operation without partitioning, i.e. threads of different programs may share the processors.
- Load consists of machine instructions to be executed, i.e. transferable load consists of
  - threads that are migrated
  - work packets that are sent to server threads.

- **Goal:** Even distribution with low communication cost

# Load distribution versus Load balancing



- With load balancing between processors we want to achieve the following goals:

  1. Avoid unused processor capacity
  2. Equal progress of all threads of a parallel program
  3. *Fairness* in case of multiprogramming

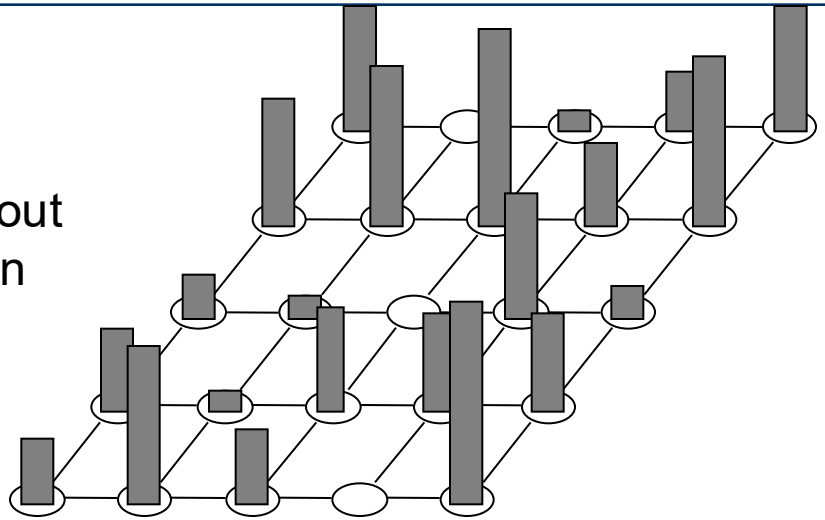
- Pursuing goal 1 only is called **load sharing**.

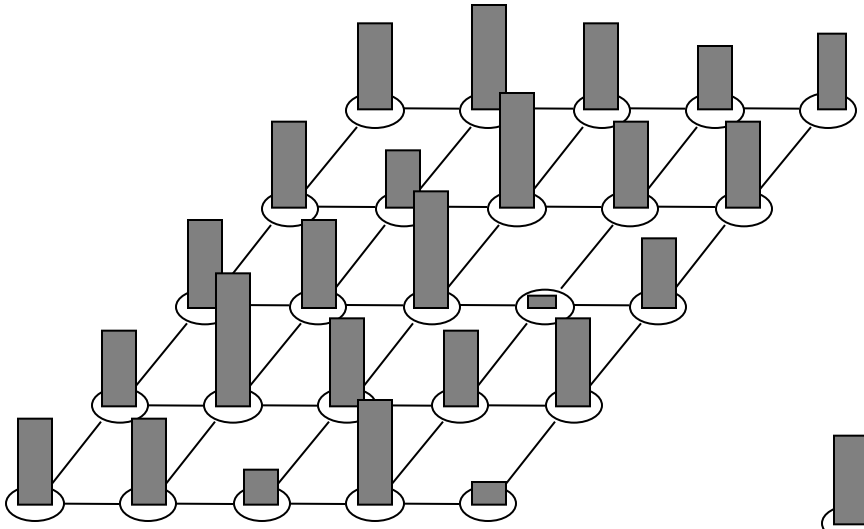
- Load sharing is finished when each processor has at least one thread to execute.

- Goals 2 and 3 are dealt with by **load balancing** which tries to equally utilize all processors.

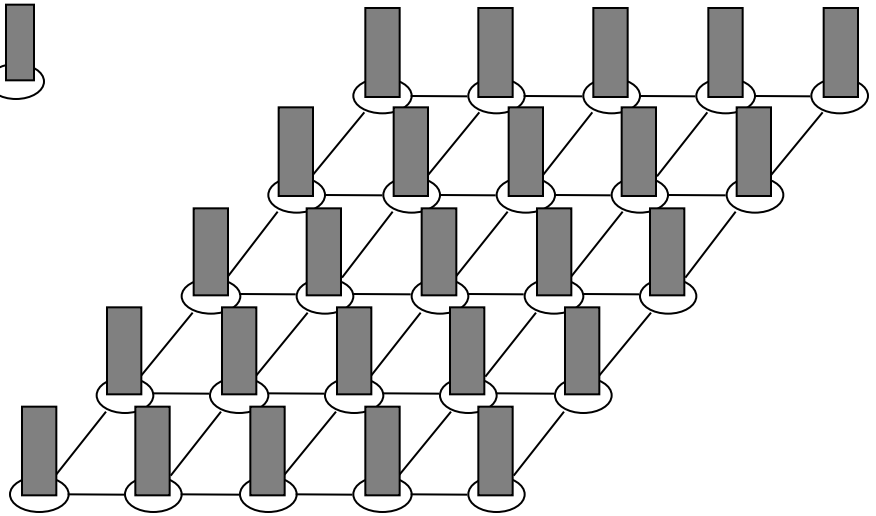
# Load sharing vs. Load balancing

Load situation without any load distribution





After load sharing



After load balancing

# Load balancing using thresholds



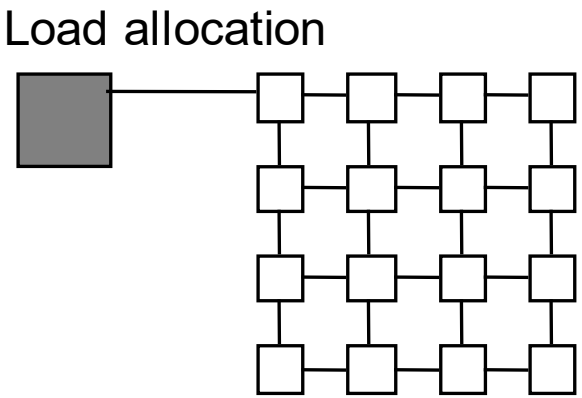
- Between load sharing and load balancing there is a pragmatic compromise that differentiates only between three possible load states:
  - underloaded: node can accept more threads.
  - well loaded: node can neither take more threads nor does it want to give away.
  - overloaded: node wants to give away threads.

- Usually we define two threshold values $s_u$ and $s_o$ , that are used to specify the load state:

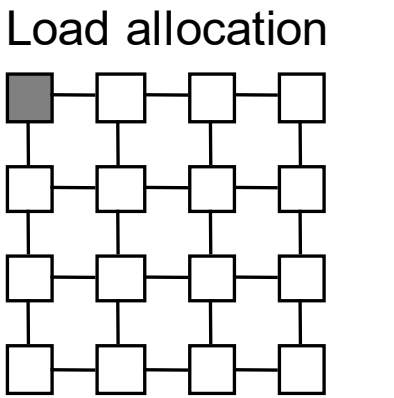
$$state_k := \begin{cases} \text{underloaded, if } B_k < s_u \\ \text{well loaded, if } s_u \le B_k \le s_o \\ \text{overloaded, if } s_o < B_k \end{cases}$$

- A load distribution mechanism is then limited to only those nodes that are either under- or overloaded, which may reduce the number of nodes involved significantly.

# Centralized Load Allocation



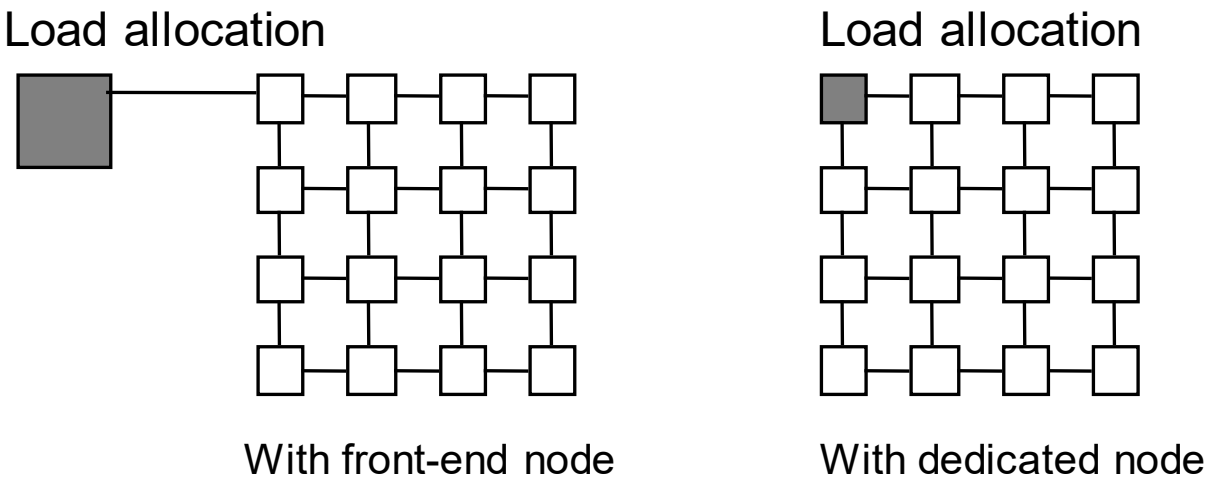

With front-end node




With dedicated node

Disadvantages?

# Centralized Load Allocation

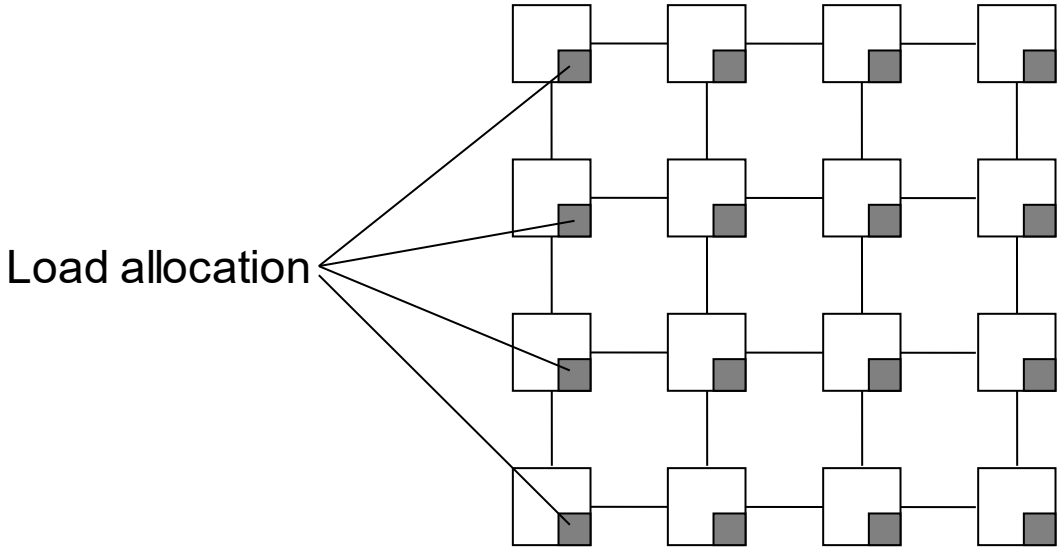



Disadvantages of central load distribution

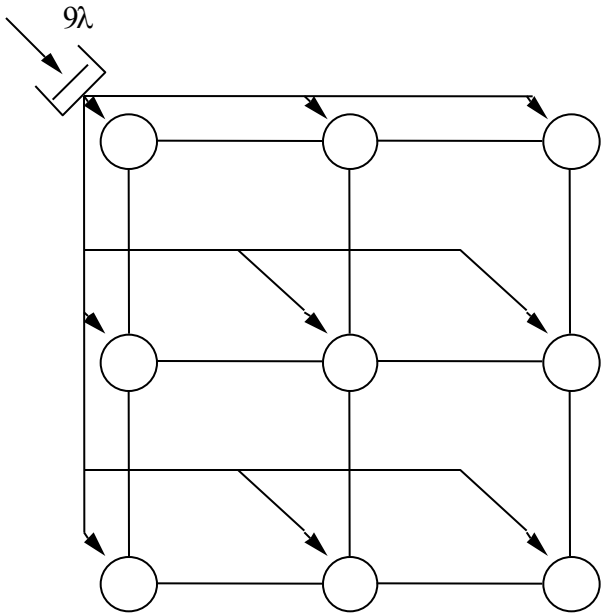
- Aging of global information
- Danger of bottleneck
- Failure problem
- Locality of load distribution activities
- Lacking parallelism

# Decentralized Load Allocation

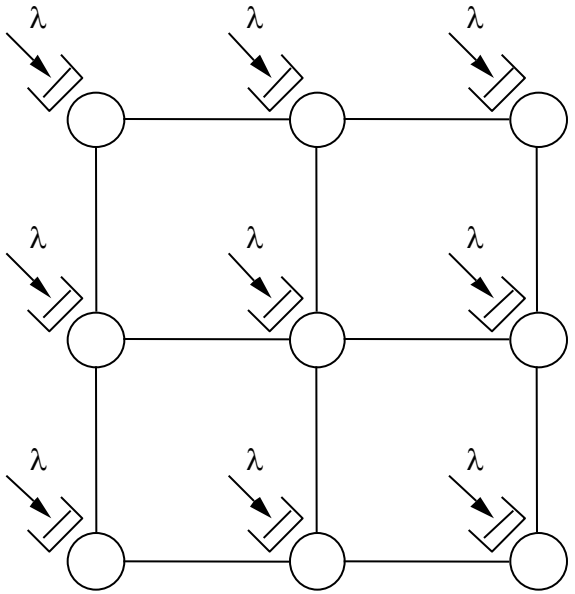
- All nodes run identical agents that pursue a balanced load cooperatively.
- *Heuristic distributed* algorithms are applied (due to the incompleteness of the available information).
- Interplay of agents is governed by a protocol.



# Reference models for load distribution (Example 3x3-mesh)



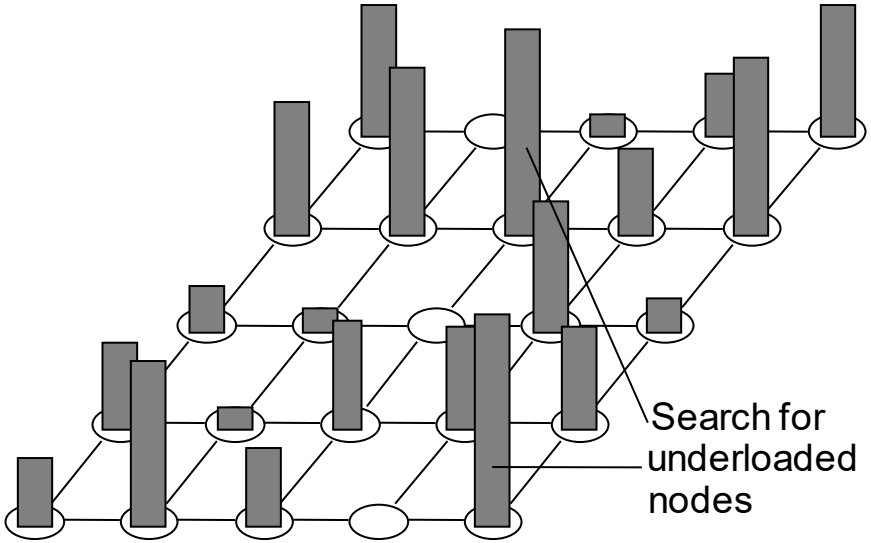

M|M|9-System



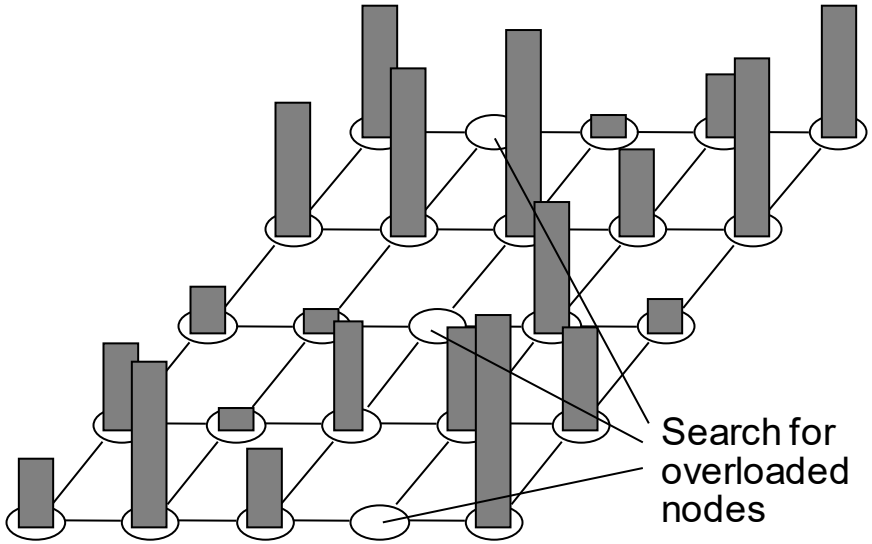

9 M|M|1-Systems

- Multiprocessor systems with shared memory and central ready list exactly correspond to M|M|n models, which is why a balanced load will develop on its own.
- Multicomputer systems rather correspond to a collection of M|M|1 systems with independent local arrival streams.

# Triggering load balancing



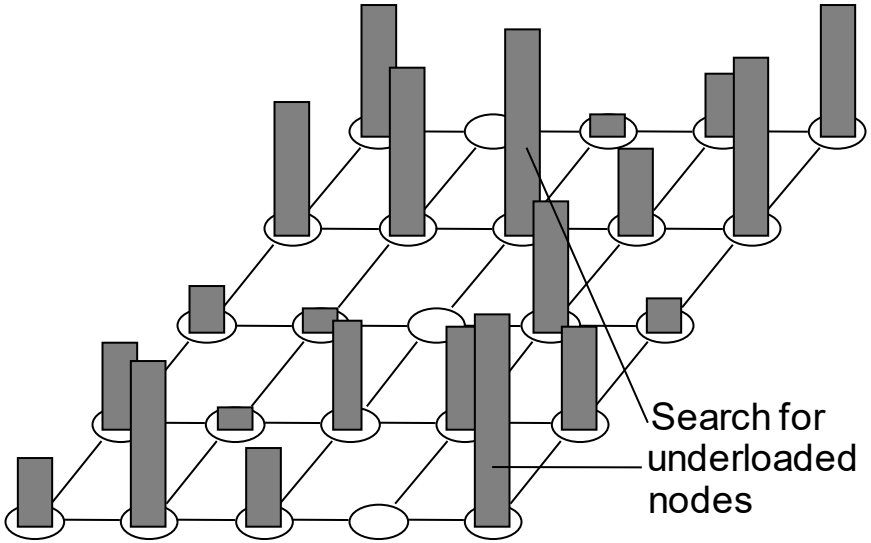

sender initiated (bidding)
(drafting)



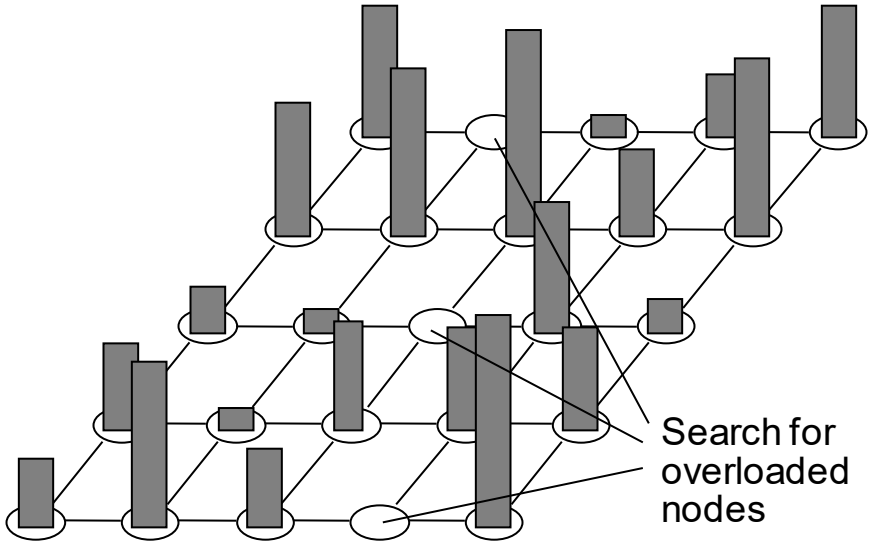

receiver initiated

***Which is better?***

# Triggering load balancing




sender initiated (bidding)
(drafting)

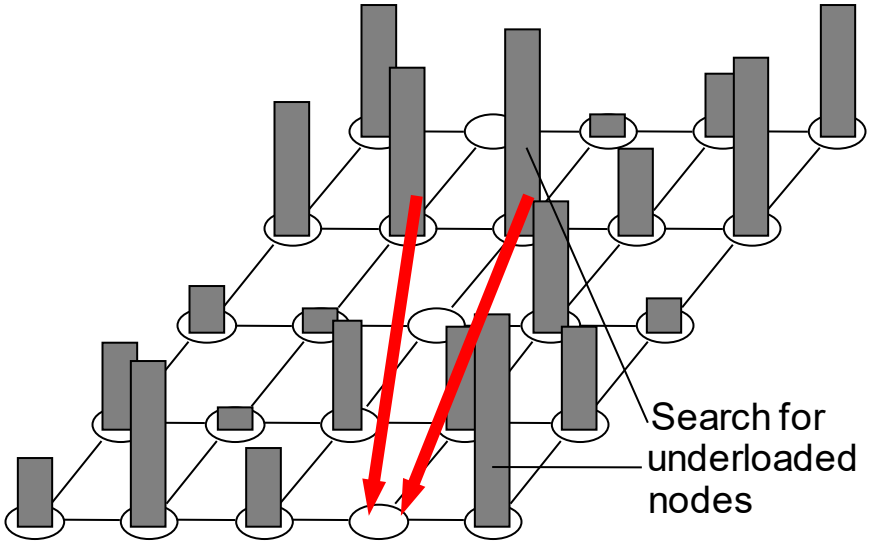



receiver initiated

***Which is better?***

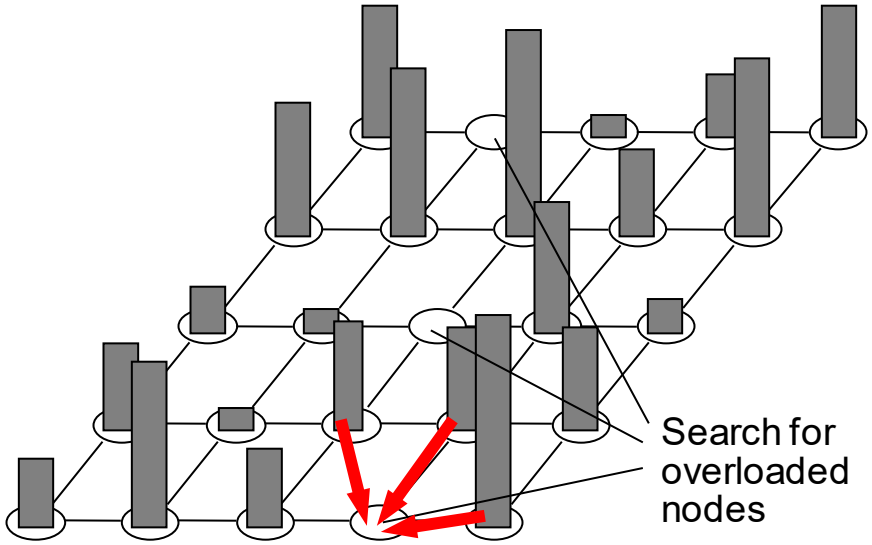
At high load the receiver initiation is better, at low loads the sender initiation („few select from many")

# Load transfer




**Global Transfer**

Load balancing between arbitrary nodes

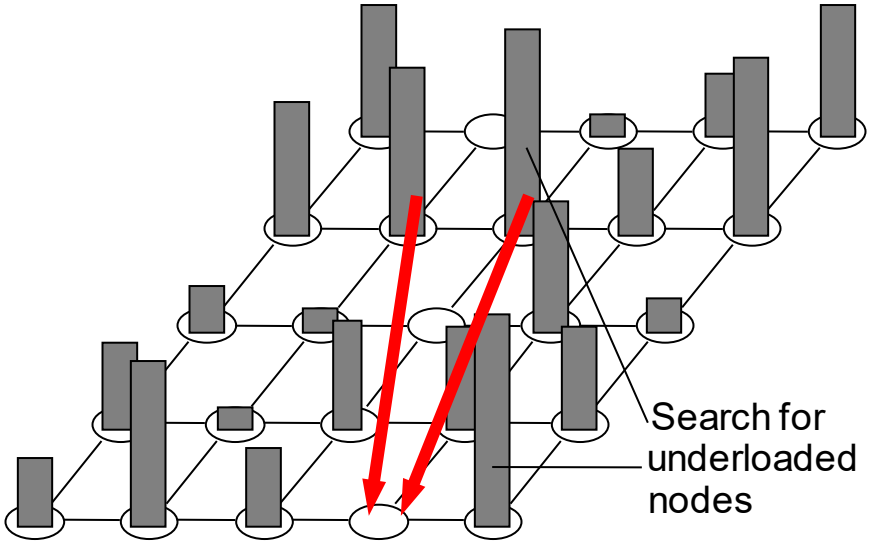



**Local Transfer**

Load balancing between neighbors only

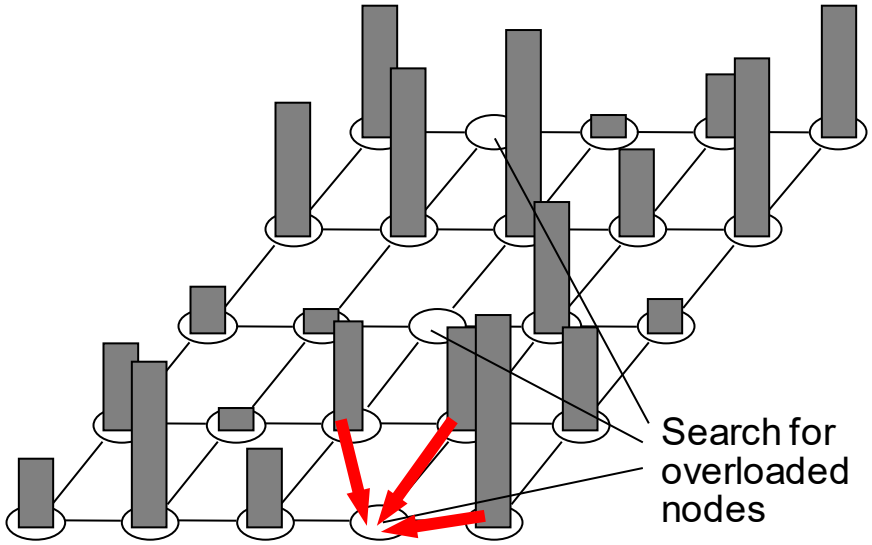
***Which is better?***

# Load transfer




**Global Transfer**

Load balancing between arbitrary nodes




**Local Transfer**

Load balancing between neighbors only

***Which is better?*** Better distribution vs. knowledge needed.

# Information dissemination

**Information problem**

- The more accurate the knowledge of a node about the load situation of all other nodes, the better the decisions that are made cooperatively.
- Dissemination and acquisition of information means significant overhead that may destroy the benefit of load balancing.

**When to send load information messages?**

# Information dissemination

## Information problem

- The more accurate the knowledge of a node about the load situation of all other nodes, the better the decisions that are made cooperatively.
- Dissemination and acquisition of information means significant overhead that may destroy the benefit of load balancing.

## When to send load information messages

- *Periodic* messages
  - Problem of proper selection of period
- *Message on load change*
  - Reduces number of messages sent
- *Message on request*
  - Information needs only be sent when demanded
  - Reduces the number of sent messages (if demanded not too often) but means additional delay when needed

# Number and selection of addressees



- *Broadcast-based solutions (all-to-all-broadcast)*
  - High overhead: $O(n^2)$ messages
- *Random selection*
  - Information diffuses as a stochastic process through the network.
  - Reduces number of messages, but slows down the dissemination process.
- *Broker*
  - Collects offers and requests and helps to find a match
  - Nodes send their own load state periodically or at changes
  - Nodes can request load information about other nodes from Broker

# Example of random selection:

- Each node maintains a state vector $Z$ of length $k < n$, in which the own state $Z(0)$ and states of other $k$-1 randomly selected nodes are stored.

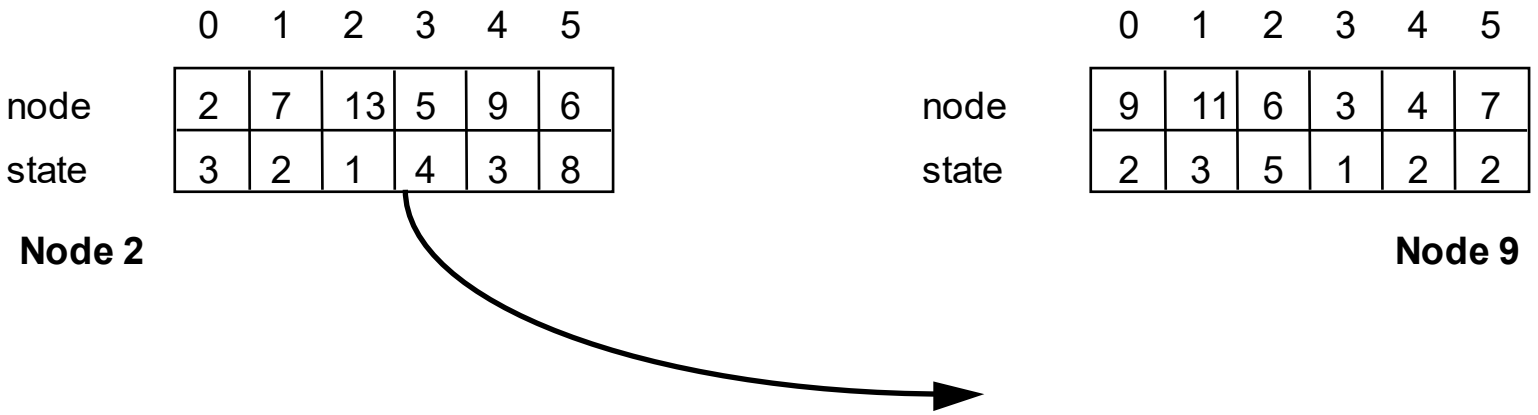
- Periodically each node performs the following actions:
  - 1. Update own state $Z(0)$
  - 2. Select processor $r$ randomly ($1 \leq r \leq n$)
  - 3. Send first half of state vector Z to node $r$

- Upon reception of a half state vector $Z_r$ merge the first half of the own vector with the received half according to the following rule:

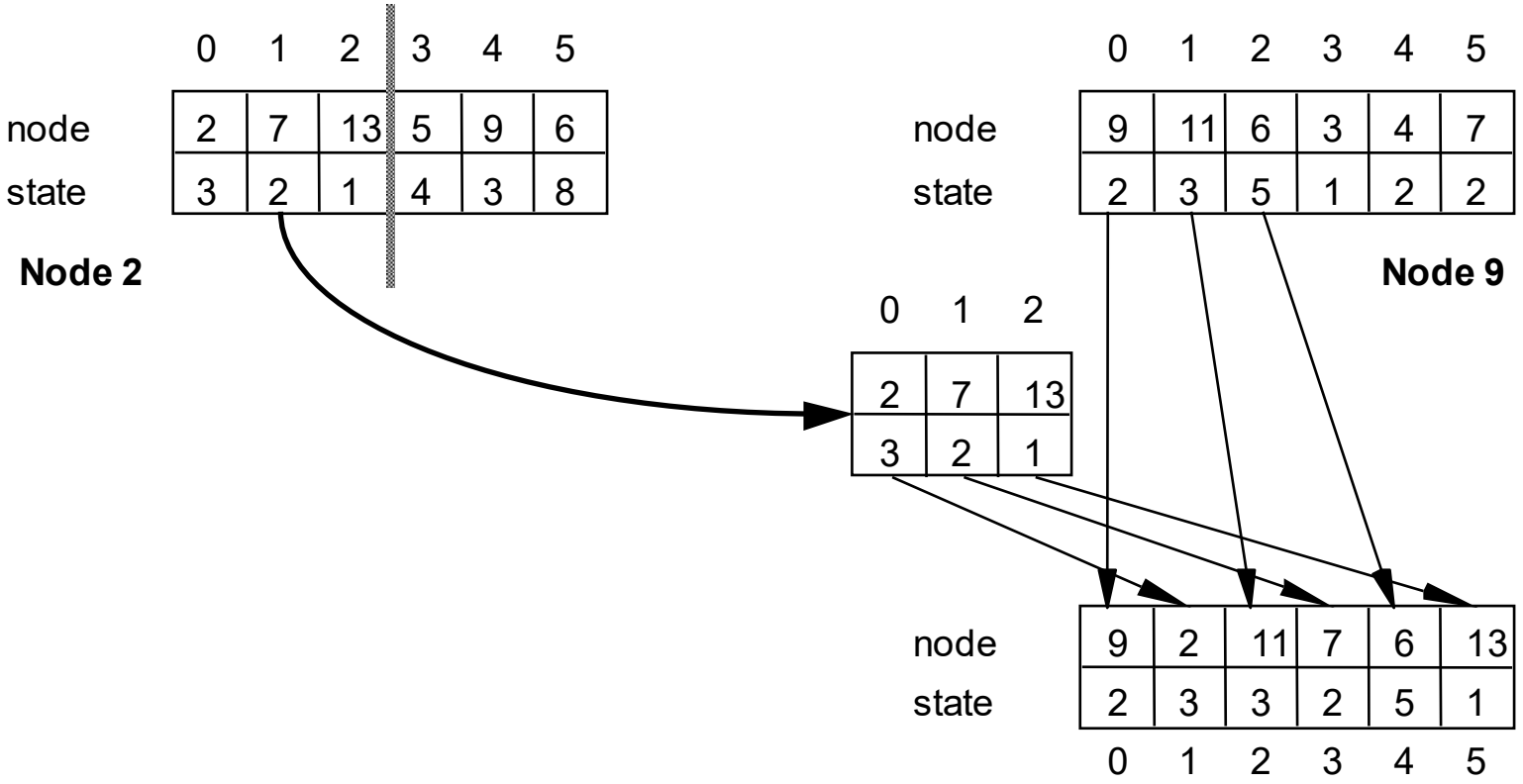
$$Z(2i) \leftarrow Z(i) \quad , \quad 1 \leq i \leq k/2 - 1$$

$$Z(2i+1) \leftarrow Z_r(i) \quad , \quad 0 \leq i \leq k/2 - 1$$

# Example of random selection



# Example of random selection



# 9.2 Microeconomic approaches

- Microeconomic approaches try to solve the allocation problem using market mechanisms.
- Resources (computational- and transmission capacity) are considered as scarce goods on a electronic market place.
- Applications or threads are buyers or consumers of these resources.
- Bids are managed by performing auctions.
- After specific rules a buyer is awarded a contract.
- Both sides (buyer and seller) try to maximize their individual utility function.

# Properties of microeconomic approaches

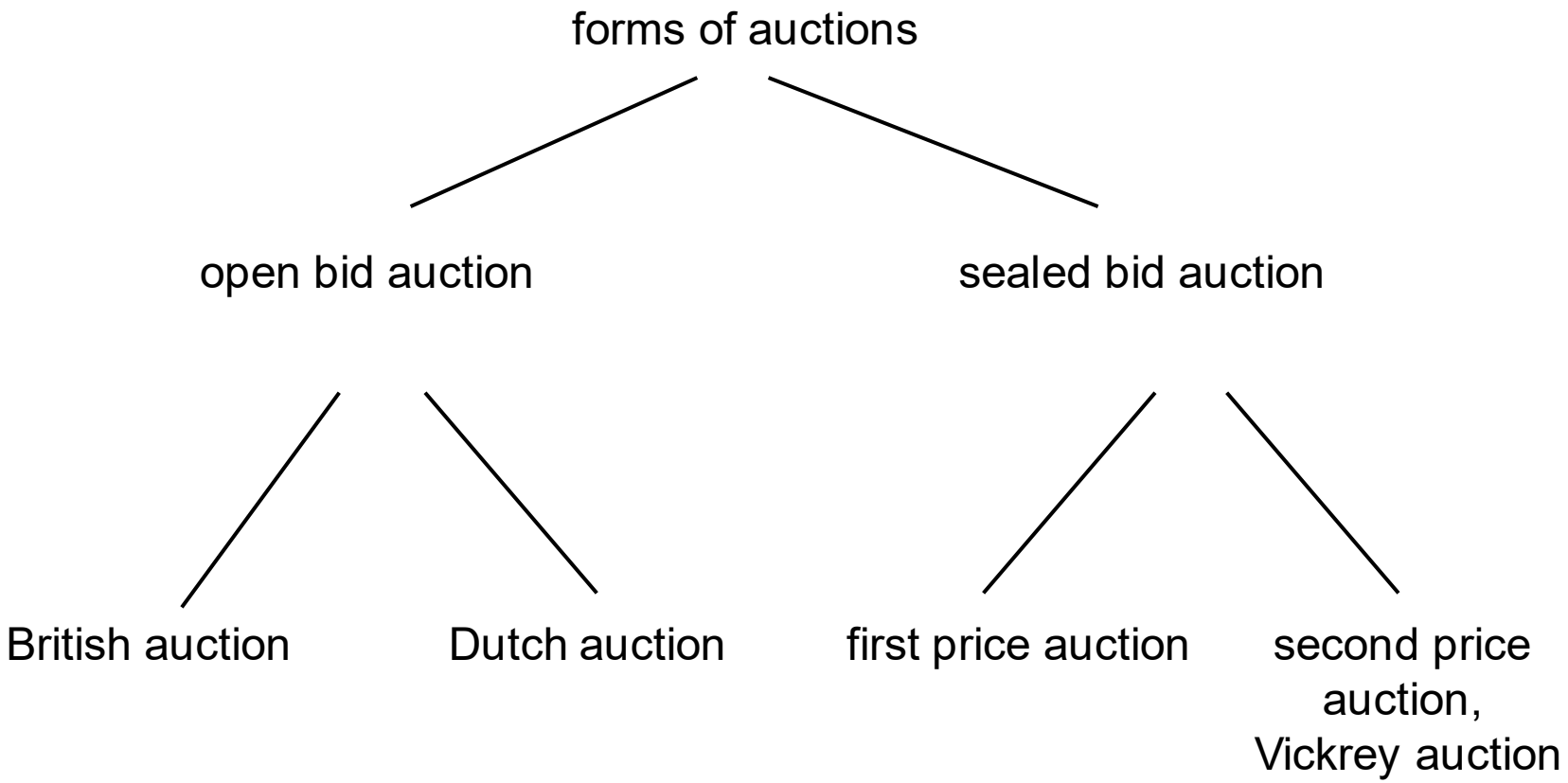


- Different kinds of interest (necessity, utility) of a requester for a resource can be modeled.
- Often, usage of resources has to be paid for anyway (accounting).
- Scarcity leads to price increase and to demand decrease.
  - Overload prevention
  - Hot-Spot avoidance
- Idle capacities result in decrease of prices:
  - Demand is stimulated
- Automatic adaptation
- Good theoretic foundation (microeconomics, game theory)

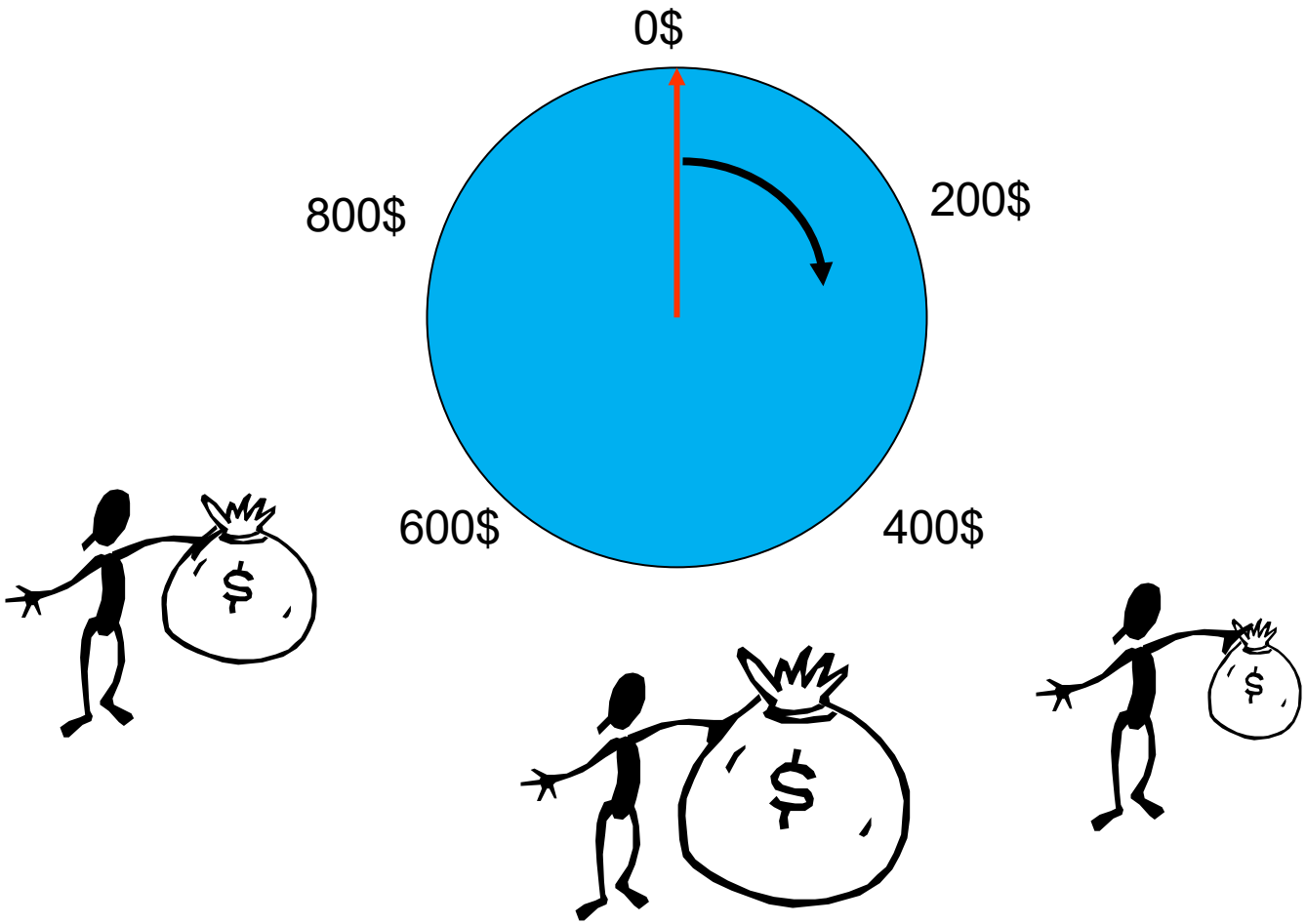
# 9.2.1 Auctions

forms of auctions?

# 9.2.1 Auctions

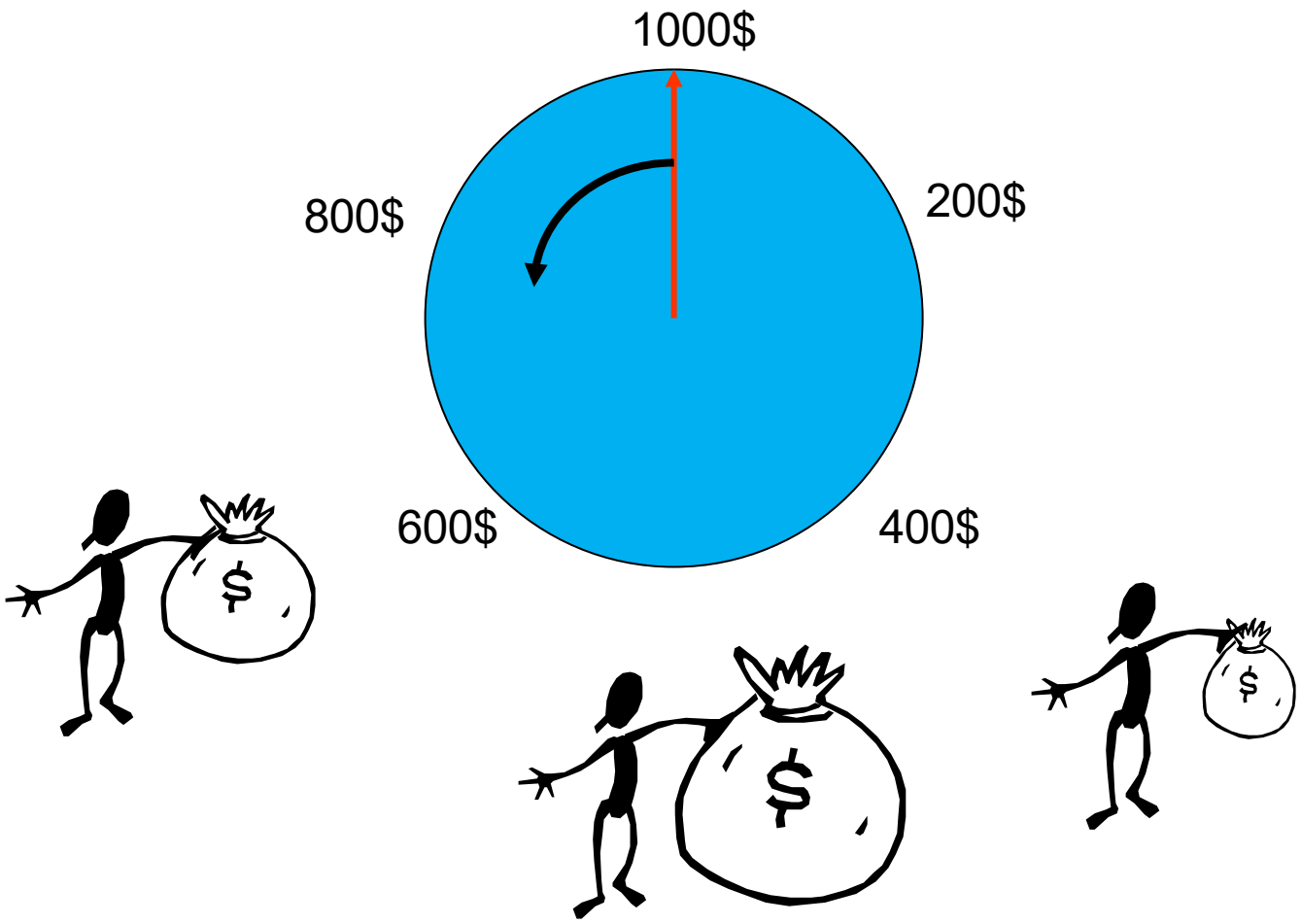


# British Auction (Ascending Bid)



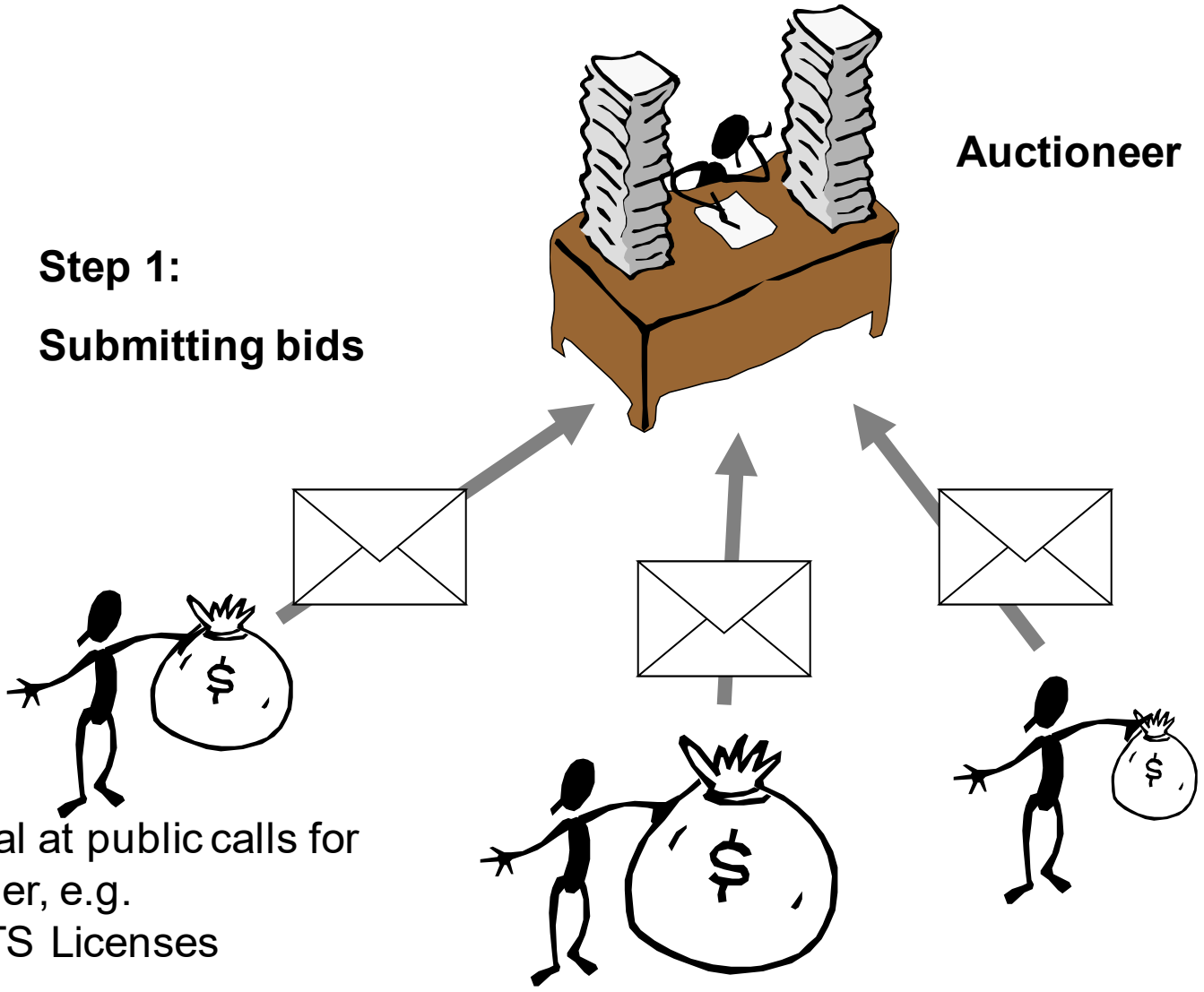

Usual in auction houses: Sotheby's, Christie's

# Dutch Auction (Descending Bid)




Usual at central markets (e.g. Dutch flower market)

# First Price Sealed Bid Auction




Usual at public calls for tender, e.g.
UMTS Licenses

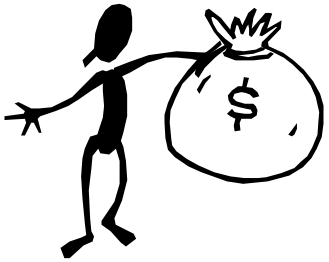
# First Price Sealed Bid Auction

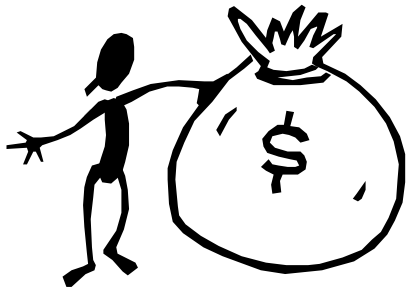


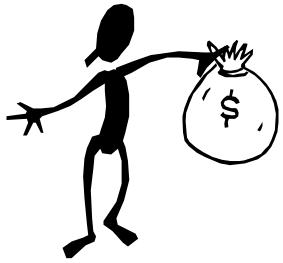
**Step 2:**

**Selection of best bid**

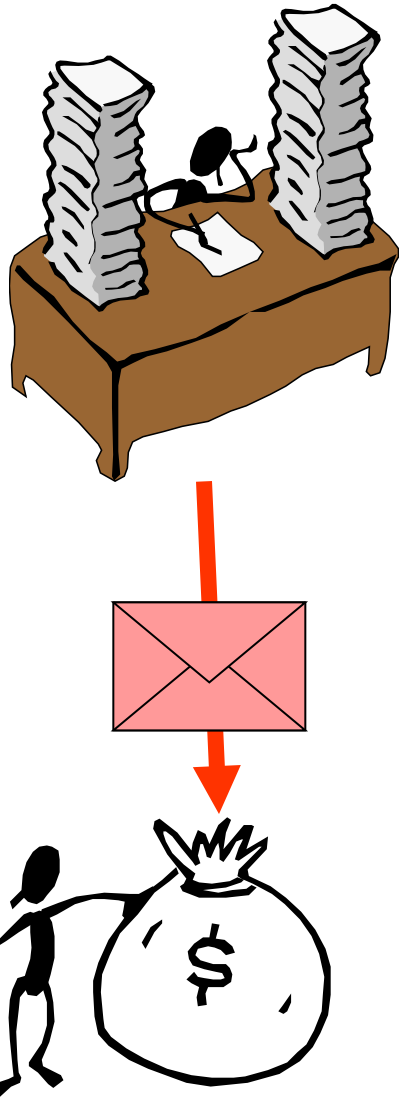
**Auctioneer**







# First Price Sealed Bid Auction



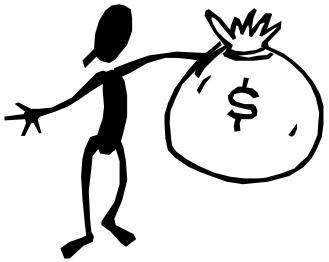
# Variant: Second Price Sealed Bid Auction

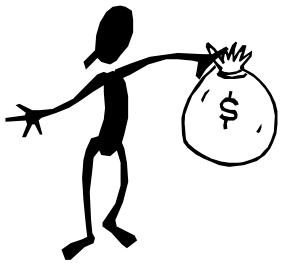


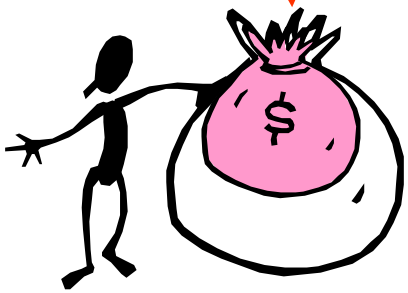
**Step 3:**

**award**

**Auctioneer**





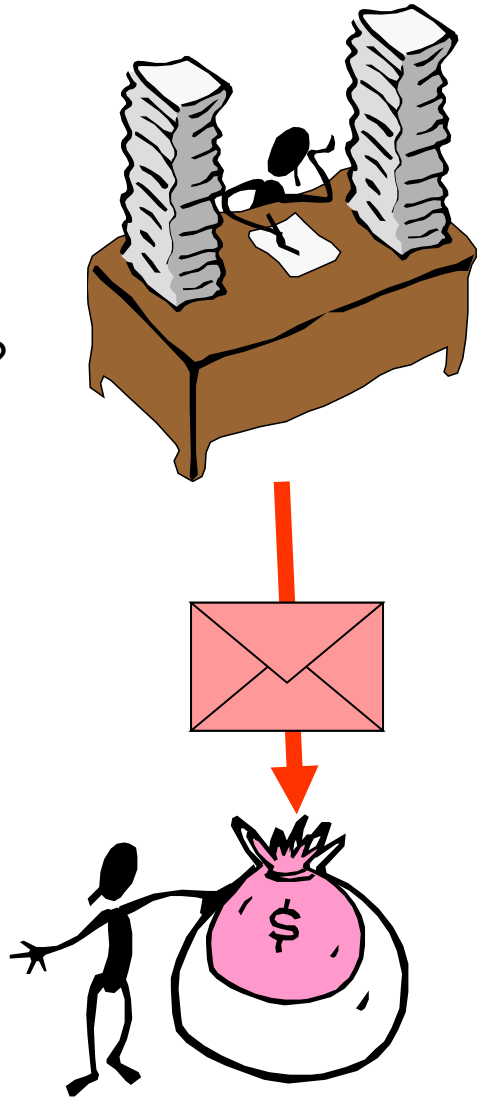


Winner pays only the
Amount of second highest bid
(Vickrey, Nobel prize 1996)

# Variant:
# Second Price Sealed Bid Auction

Winner pays only the
Amount of second highest bid

Why would an auctioneer use this?



**Auctioneer**

# Theoretic result

- Under some assumptions (individual preferences, symmetric bidder,...) the following theorem holds:

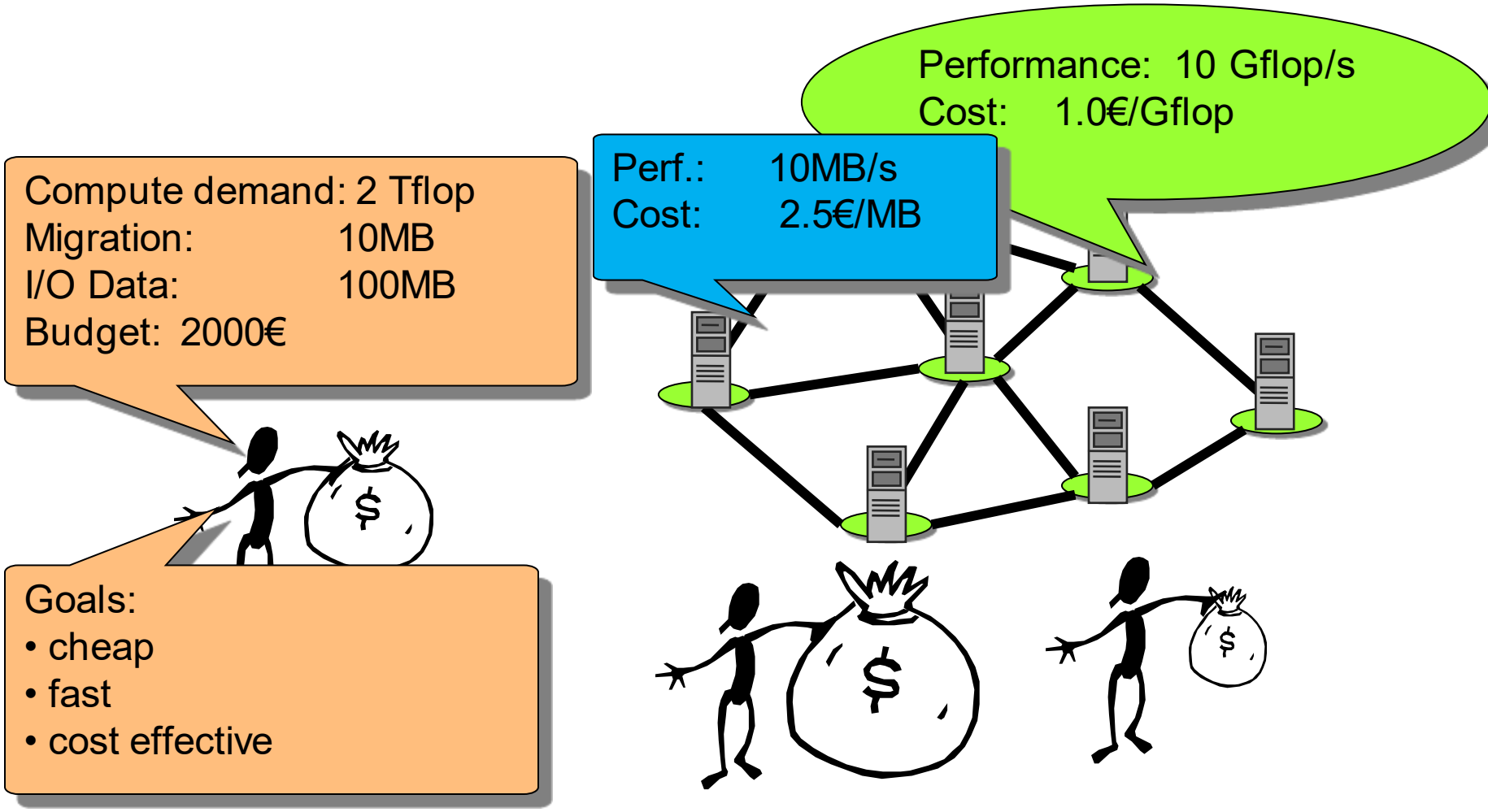
- **Revenue Equivalence Theorem (Vickrey):**
  All four forms of auctions lead to the same expected (ex ante) revenue.

  „Ex ante" means: on the basis of information available before the auction.

# Other Variants / Aspects

- Double-sided auctions: many seller - many buyer (stock exchange)
- Public sale of bundles of goods/resources (bundle bidding, combinatoric auctions)
- Continuing auctions (no punctual auctions, but continuous submission of bids)
- Volume discount auction (discount for larger amounts)
- Multiattribute auctions (besides the price other attributes (timeliness, reliability, QoS) have to be considered)

# 9.2.2 Example: Processes in a distributed system



# Model Framework

Let be $GP = (P, E_P)$ the processor graph with

- $n$ Number of processors $|P|$
- $d_{kl}$ Distance between processor $P_k$ and $P_l$, (transmission time per data unit)
- $\mu_k$ processing speed of processor $P_k$ (instructions / time)

Let $T$ *be the* (dynamic) set of processes with

- $m$ number of processes $|T|$
- $\beta_i$ compute demand of process $T_i$ (number of instructions)
- $b_i$ size of description of process $T_i$ (number of data units to be migrated)
- $e_i$ size of result of process $T_i$ (number of data units to be transmitted to the location of origin)

# Microeconomic Model

- At its creation, each process gets some amount of money $B_i$, its *Budget*.
- Each requested service (computation, transmission) needs to be paid for.
- Example:

  $T_i$ needs $\beta_i$ Instructions:

  for its execution on processor $P_k$ it must buy

  $\beta_i / \mu_k$ CPU seconds.
- The processors announce their current prices at their locations.
- These price lists can be seen from all neighboring nodes.

## (1) Calculation of action space

- The action space $A_i$ of a process $T_i$ consists of the set of actions it can still afford, based on its current budget
- Formally:

  action space = set of pairs ($k$, $C_k$) with $C_k < B$.
- $C_k$ are the costs of a service (e.g. computation) on $P_k$, composed of:
  - Execution cost:

    If $p_k$ *is the current* price for a CPU second on processor $P_k$ *then the* execution costs are $p_k$ ($\beta_i$ / $\mu_k$)
  - Migration cost:

    If a process $T_i$ is currently residing on processor $P_l$ , then $b_i$ data must be transmitted from $P_l$ to $P_k$, which results in costs of $b_i$ $p_{lk}$ if the current transmission price is $p_{lk}$ .
  - Cost for transmission of results to $P_l$: $e_i$ $p_{lk}$

**(2) Defining preferences (Ordering the action space)**

- Depending on its preferences, each process selects from its action space the „best" action. Different criteria or preferences are possible:

Which ones?

**(2) Defining preferences (Ordering the action space)**

- Depending on its preferences, each process selects from its action space the „best" action. Different criteria or preferences are possible:
- **Price**:
  - The cheapest action is selected.
- **Performance**:
  - The action with the shortest execution time $S_k$ is selected:

    $S_k := \beta_i / \mu_k + b_i \; d_{lk}$

    if process $T_i$ currently resides on processor $P_l$ .
- **Cost-effectiveness**:
  - The action with the best price-performance-ratio (PPR) is selected.
  - Since the execution time is inversely proportional to "performance", we define:

    $PPR_k := C_k \; S_k$

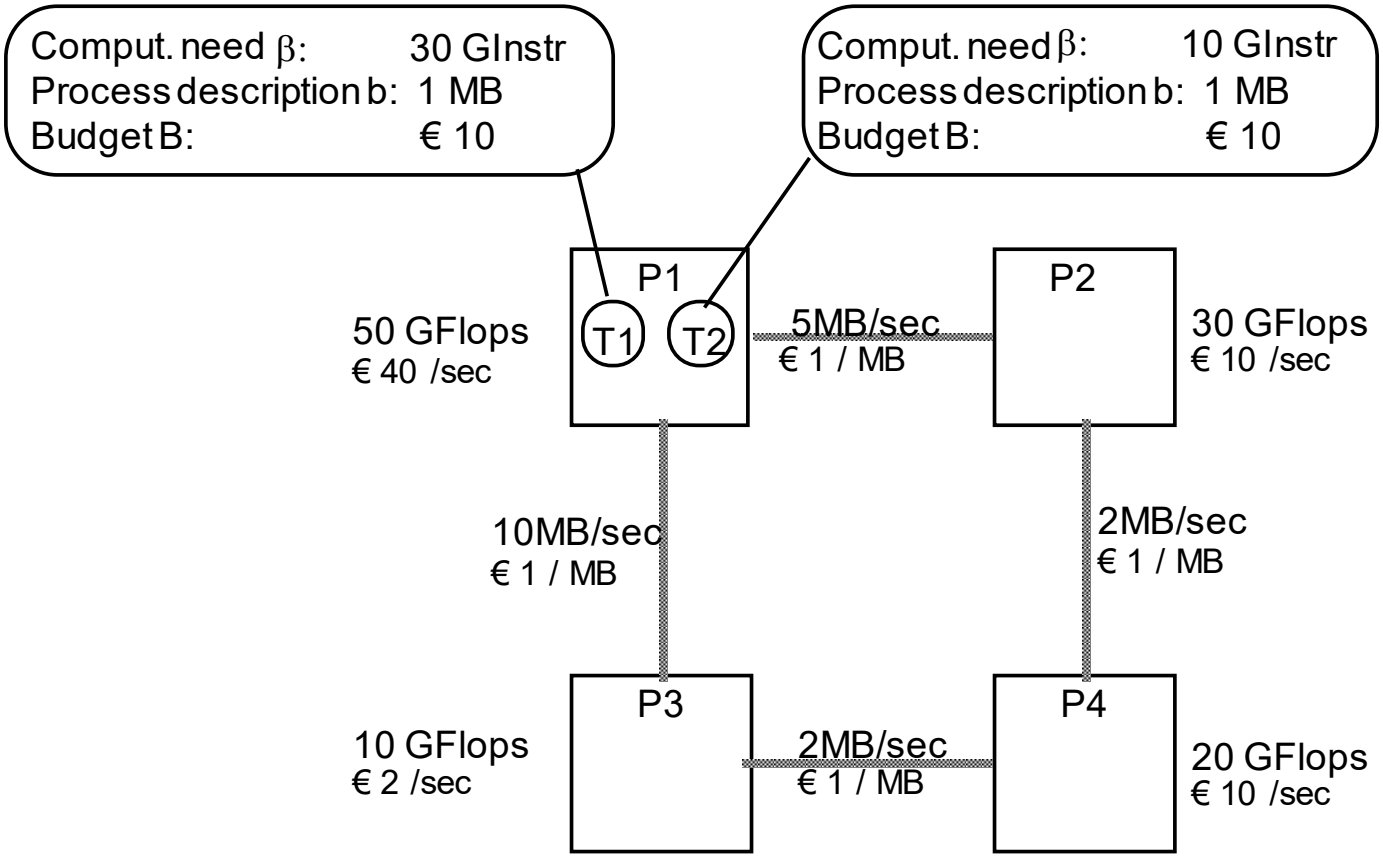
## (3) Submitting a bid

- A bid $G$ consists of the price charged plus a surcharge depending on the remaining budget:
- Let $(k, C_k)$ be the action selected by $T_i$:

  $G := \text{price} + (B - C_k)\, a_i$

  with $0 \leq a_i \leq 1$ outbid factor
- If there is more than one bid for a resource, the processor makes some appropriate choice.
- The process $T_j$ , that is awarded the contract, assumes that it might have bid too much and therefore sets: $a_j := a_j - d_j$ $(d_j > 0)$.
- Processes $T_i$ , that have been outbid conclude that they have bid too little and set: $a_i := a_i + d_i$ $(d_i > 0)$.

# Example:


Comput. need β: 30 GInstr
Process description b: 1 MB
Budget B: € 10
Comput. need β: 10 GInstr
Process description b: 1 MB
Budget B: € 10
P1
T1
T2
P2
50 GFlops
€ 40 /sec
5MB/sec
€ 1 / MB
30 GFlops
€ 10 /sec
10MB/sec
€ 1 / MB
2MB/sec
€ 1 / MB
P3
P4
10 GFlops
€ 2 /sec
2MB/sec
€ 1 / MB
20 GFlops
€ 10 /sec

# Example (continued)



- Calculation of action spaces (grey means „not feasible“, i.e. not within budget constraints)

| **Actions of T1** | **P1** | **P2** | **P3** |
|---|---|---|---|
| Price | | | |
| Performance | | | |
| Cost-effectiveness | | | |

| **Actions of T2** | **P1** | **P2** | **P3** |
|---|---|---|---|
| Price | | | |
| Performance | | | |
| Cost-effectiveness | | | |

# Example (continued)



- Calculation of action spaces (grey means „not feasible“, i.e. not within budget constraints)

| **Actions of T1** | **P1** | **P2** | **P3** |
|---|---|---|---|
| Price | 24 € | 11 € | 7 € |
| Performance | 0.6 sec | 1.2 sec | 3.1 sec |
| Cost-effectiveness | 14.4 € sec | 13.2 € sec | 21.7 € sec |

| **Actions of T2** | **P1** | **P2** | **P3** |
|---|---|---|---|
| Price | 8 € | 4.33 € | 3 € |
| Performance | 0.2 sec | 0.533 sec | 1.1 sec |
| Cost-effectiveness | 1.6 € sec | 2.31 € sec | 3.3 € sec |

# Behavior of Processors

**1 Perform Auction**

- Based on current prices the processes submit their bids.
- If there is more than one bid, the processor starts an auction aiming at selecting the most favorable bid.
- Two approaches for selection:
  - Sealed bid
  - Dutch auction

**2 Adapt Prices**

- After the auction the new price is determined as that at which the contract was signed.

**3 Advertise**

- The processors send their new pricelists to neighboring nodes.

## Maximum Unit price

- A bid consists of a demanded service $L_i$ and an offered amount of money $G_i$:
- The bid with max{$Gi / L_i$} wins.
- That leads to the following algorithm (sealed bid auction):

```
Price ← initial price
   while not (bid accepted) do
        notify prices
        collect bids
        if number of bids submitted > 0
              then contract for max{Gi/Li}
        else Price ← Price - deduction
   end while
```

# Dutch Auction

With the Dutch Auction the initial (high) price is decremented (and if necessary incremented again) until exactly one bid is remaining (Dutch auction):

```
Price ← initial price
while not (bid accepted) do
        notify prices
        collect bids
        case  (number of submitted bids)
                0 : price ← price - delta
                1:  accept bid
                otherwise:  price ← price + delta
        end case
end while
```

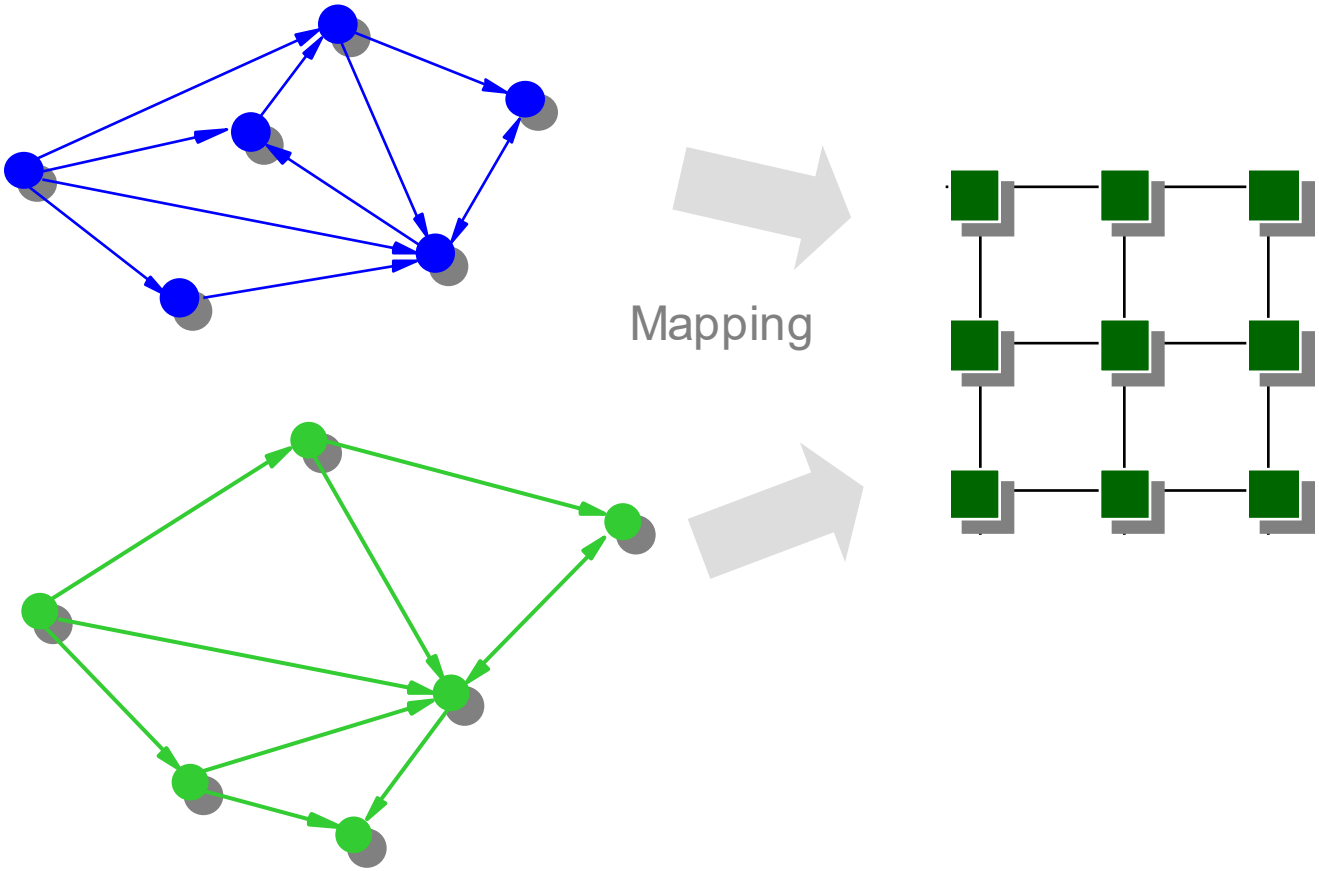
# Results

Simulations for a homogeneous 3x3-mesh-connected system and a dynamic arrival stream of processes with exponentially distributed service times yields the following results:

- Concerning response time, no significant differences for the preference- and auction variants.
- Migration behavior:
  - For price preference relatively high, for performance preference relatively low migration rate.
- Budget:
  - If all processes have the same budget then short processes are relatively „rich", long processes relatively „poor".
  - Short processes can outbid long processes and finish earlier.
  - This approximates the „shortest job next" strategy, that generally minimizes the average response time.
- Dutch Auction is slightly better than Sealed Bid Auction, but has a higher overhead due to additional communication rounds.

# 9.3 Physical Model

Many programs share the machine in space and time.
(Dynamic Multiprogramming)



# Model framework

Parallel computer given as a Processor connection graph $G^P = (P, E_P)$ with

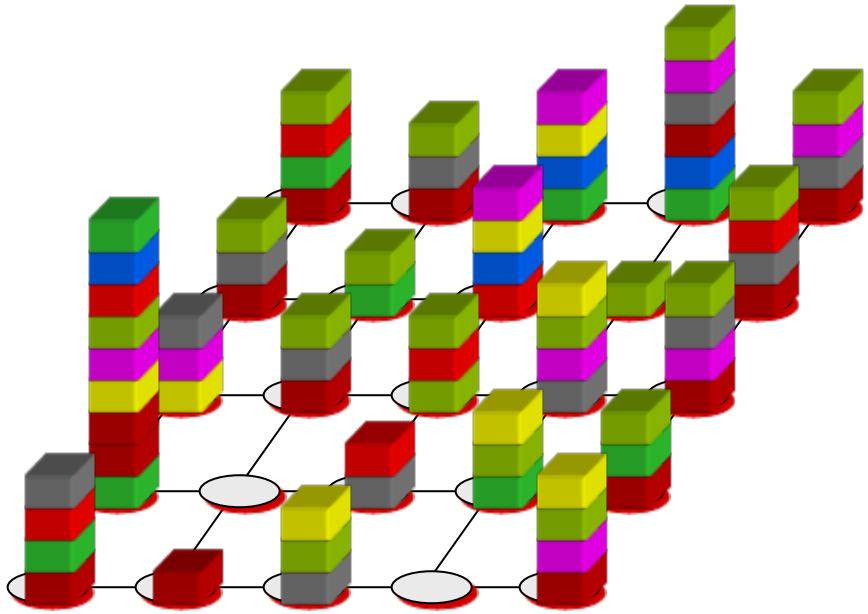
| | |
|---|---|
| $P$ | Processors (nodes) |
| $E^P \subseteq P \times P$ | Interprocessor links (edges) |
| $\delta_{kl}$ | Time to transmit a data packet over a link $(k,l)$ (edge weight) |
| $\mu_k$ | Processing speed of processor k *( node weight*) |

The set of the parallel threads is modeled as a task interaction graph $G^T = (T, E_T)$ with

| | |
|---|---|
| $T$ | Set of threads (nodes, vertices) |
| $E^T \subseteq T \times T$ | Set of communication relations (edges) |
| $\alpha_{ij}$ | Communication intensity |
| $\beta_i$ | Processing demand of thread $i$ (# Instructions) |
| $b_i$ | Size of thread description $i$ (Amount of data to be transferred when migrating) |

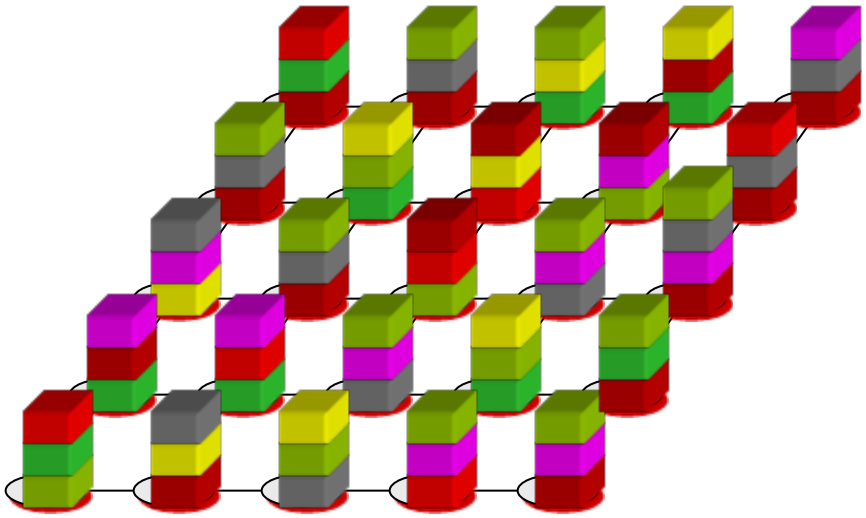
# Load situation

Spatial and temporal sharing among several programs.



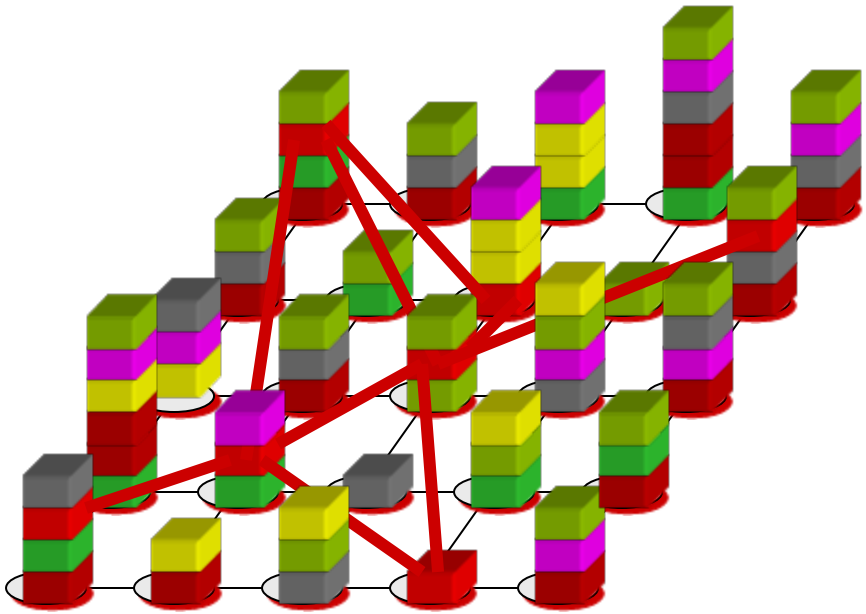
# Load balancing

All processors have roughly the same load.



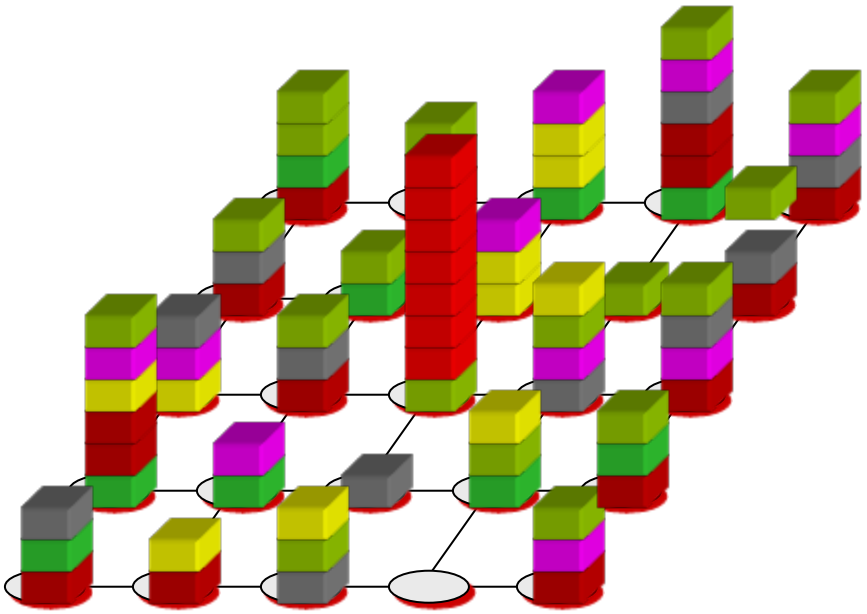
# Communication cost

Threads of the same program communicate with each other.



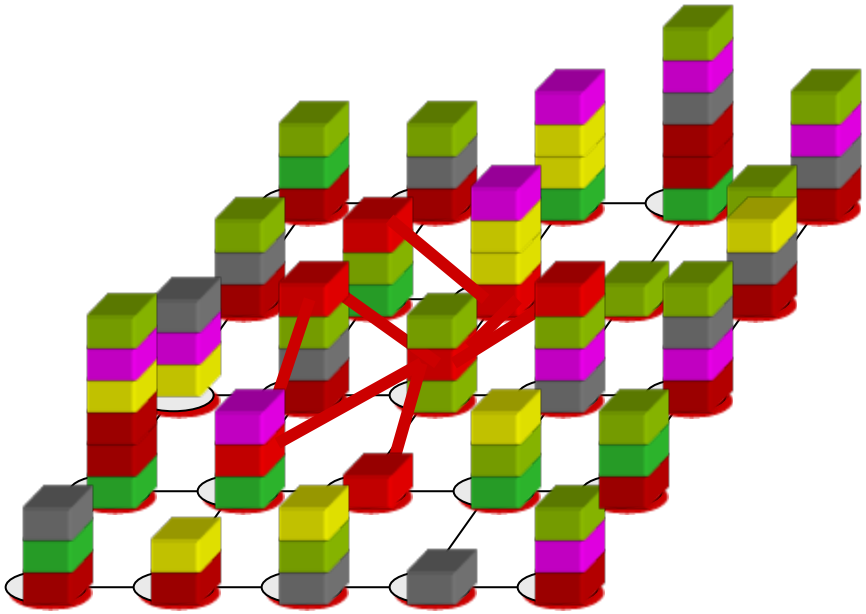
# Communication cost

Here the communication cost are 0, but the red program is no longer executed in parallel.



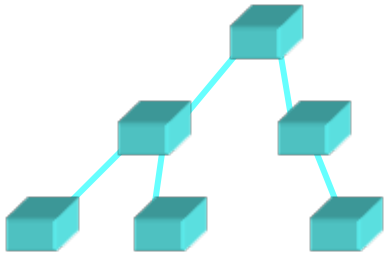
# Communication cost

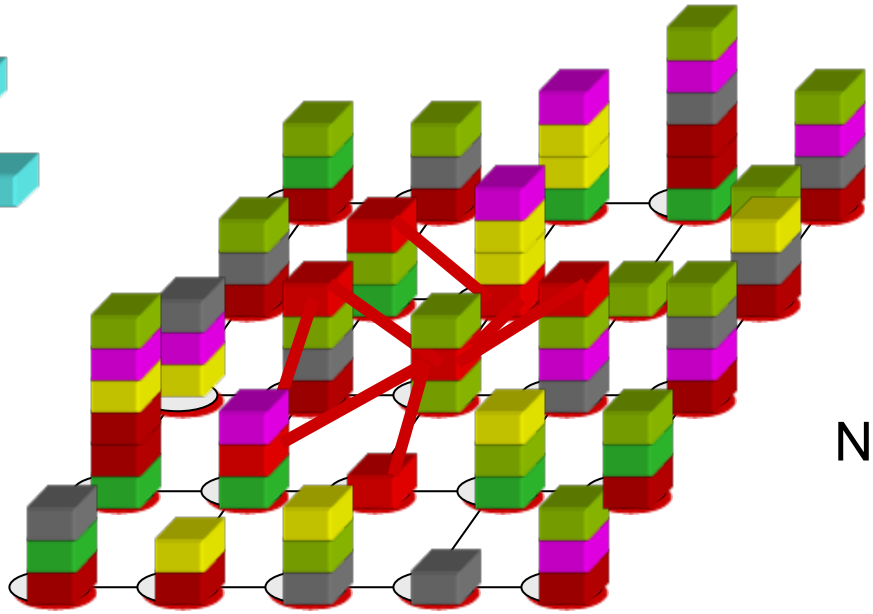
Communication cost better now than before, and nevertheless full parallelism.



# Dynamics

New program: where?





New threads: where?

Changing Communication behavior:
Remap? (=Migrate?)

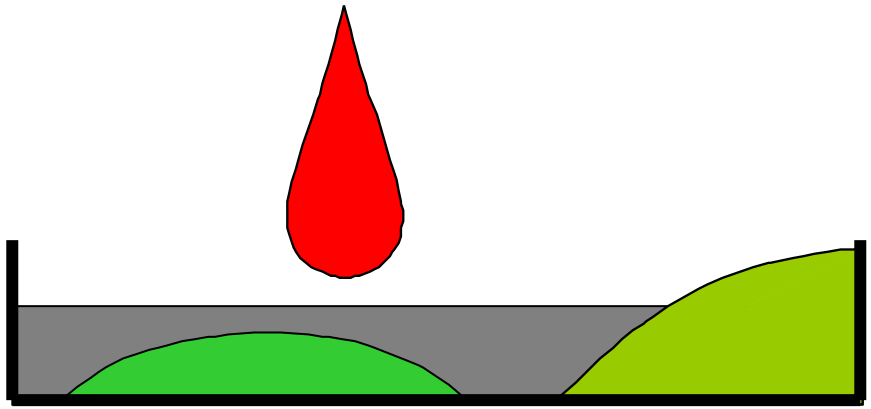
# Goals



- Balanced load
- Low communication cost
- Exploit potential parallelism
- Consider dynamic behavior
- Avoid unnecessary migrations
- Stability

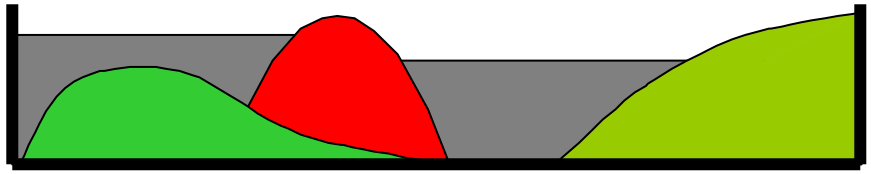
How to pursue contradictory goals simultaneously?

# Physical Analogy



Receptacle with nonmixable fluids of different viscosity.

New fluid is added.



After some time an energetic equilibrium is reached.

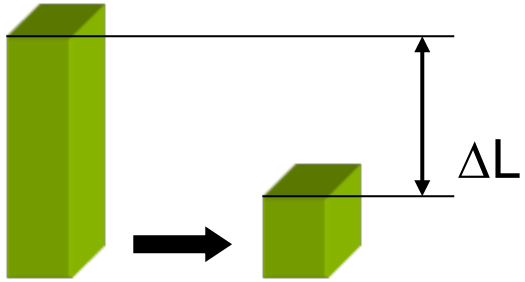
# Correspondence

| | |
|---|---|
| **thread** | **particle** |
| **program** | **viscous fluid** |
| **processor** | **coordinate** |
| **load balance** | **gravitation** |
| **communication** | **bonding forces** |
| **migration** | **friction** |

# Forces

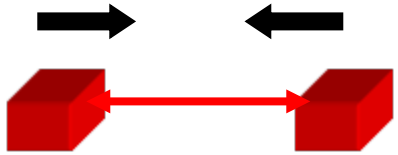
**Load balancing** ( gravitation)

Force proportional to load difference




**Communication cost** (bonding force)

Force proportional to communication intensity (rubber band)

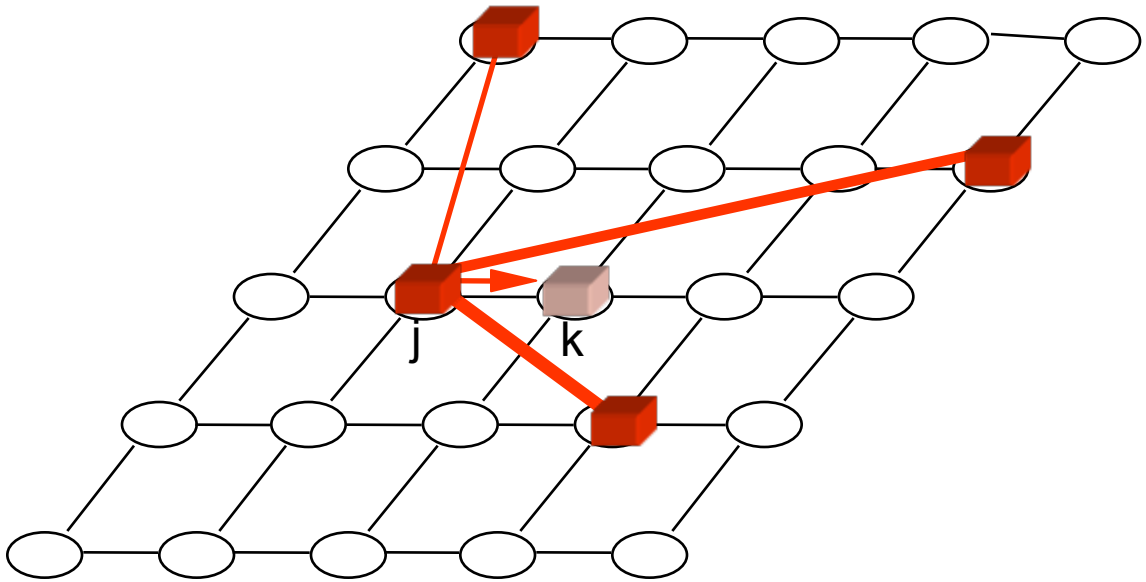


**Migration cost** (frictional resistance)

proportional to the amount of data to be transmitted

**Forces are weighted with coefficients („nature constants“) and added along the edges.**

# Example

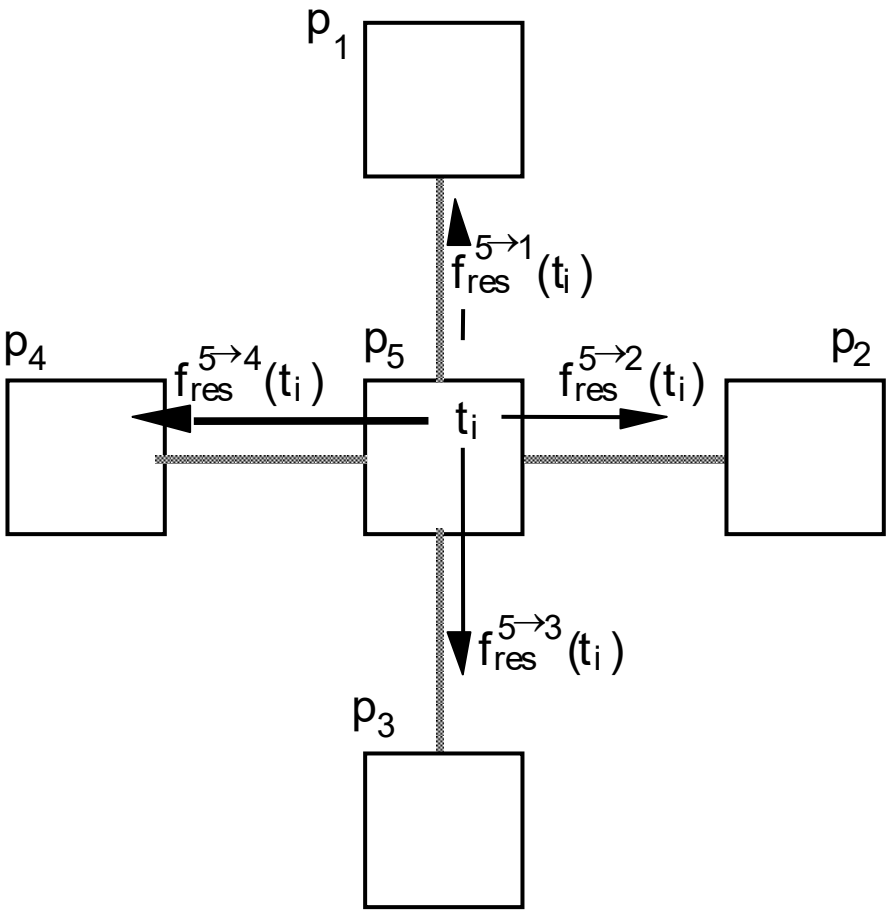



communication links act as rubber bands

# Total Force

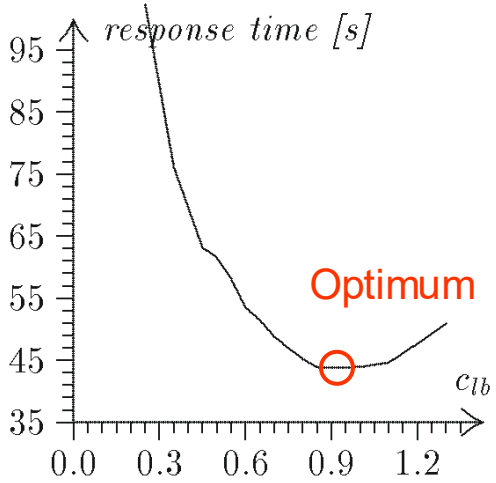
$$R_{res}^{j \to k}(t_i) := c_{lb} f_{lb}^{j \to k}(t_i) + c_{com} f_{com}^{j \to k}(t_i) + c_{mig} f_{mig}^{j \to k}(t_i)$$

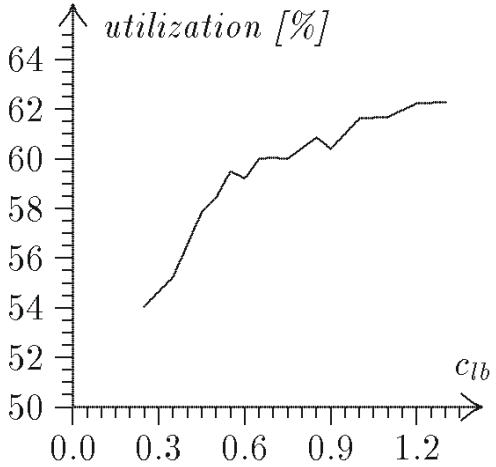
= Loadbalancing
+ Communication
+ Migration

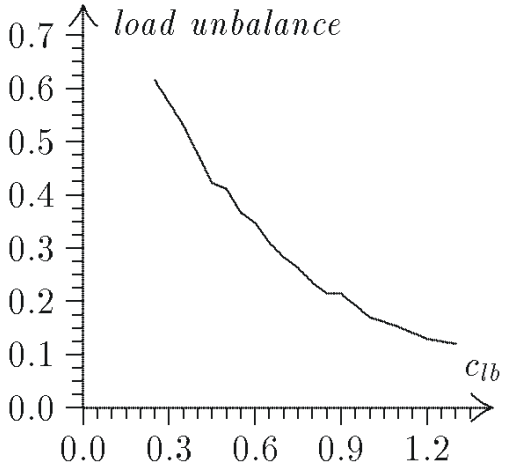


# Results (of simulation)

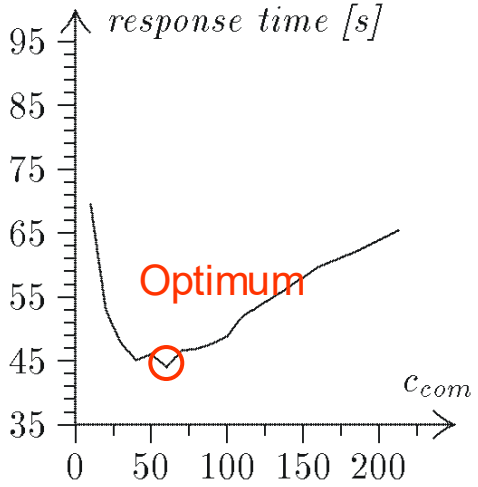
## Load balance

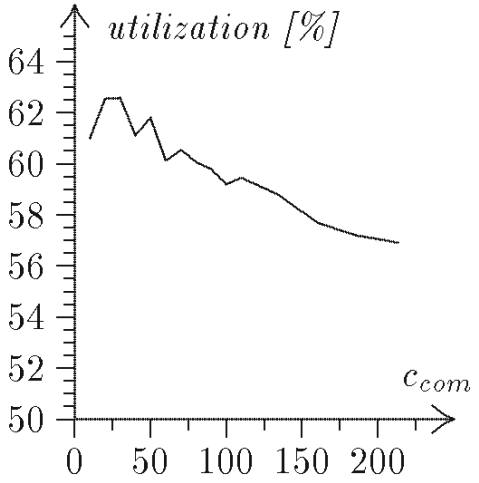


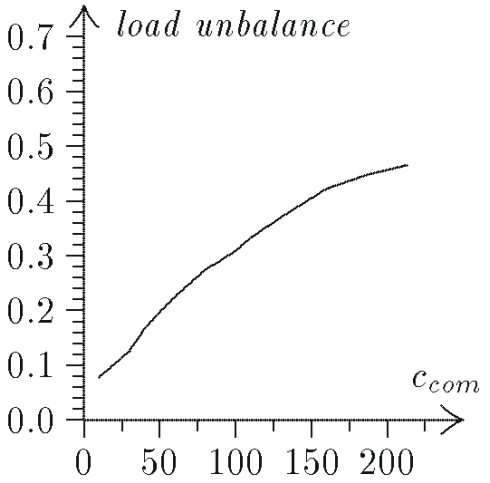



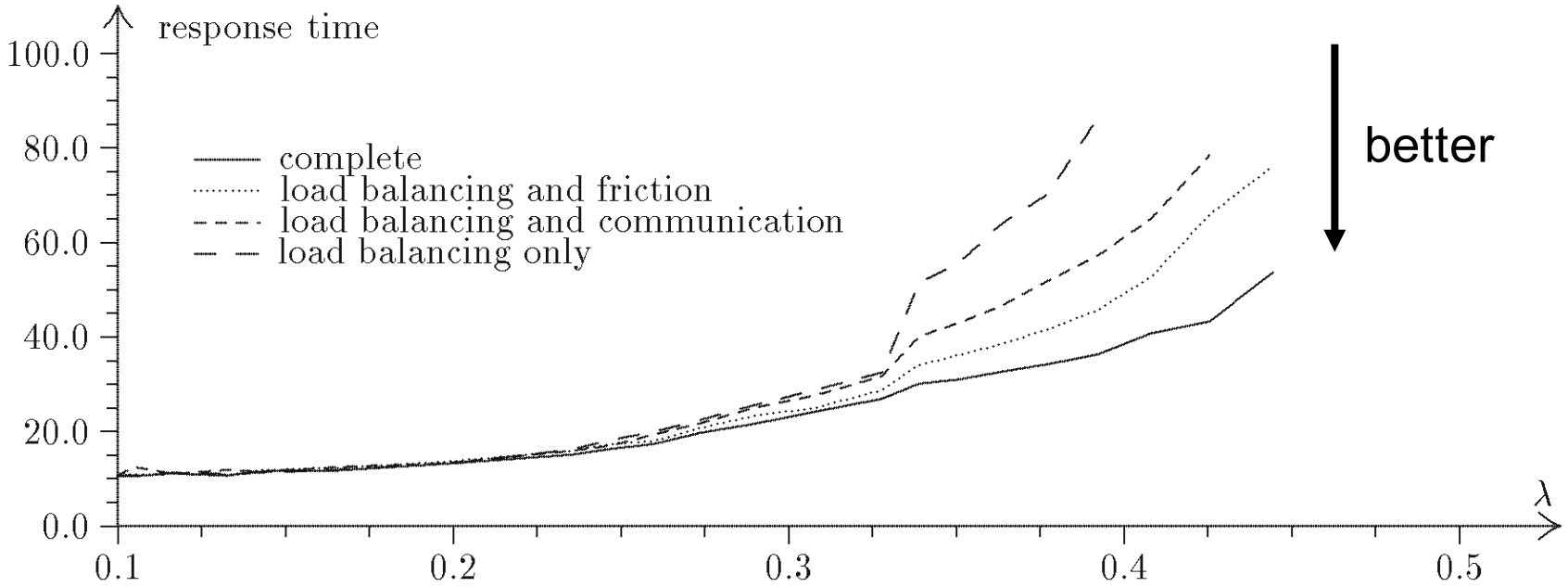

## Communication cost



# Results


response time
100.0
80.0
60.0
40.0
20.0
0.0
0.1
0.2
0.3
0.4
0.5
λ
complete
load balancing and friction
load balancing and communication
load balancing only
better

# References


- Barak,A.; Shiloh,A.: *A Distributed Load-balancing Policy for a Multicomputer.* Software - P&E 15,9 (Sept. 1985) pp. 901-913
- Casavant, T.L.; Kuhl,J.G. *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems.* IEEE TOSE 14,2 (Feb. 1988) pp. 141-154
- Cybenko,G.: *Dynamic Load Balancing for Distributed Memory Multiprocessors.* JPDC 7 (1989) : pp. 279-301
- Ferguson,D.; Yemini,Y.; Nikolaou,C.: *Microeconomic Algorithms for Load Balancing.* Proc. 8th Int. Conf. on Distr. Comp. Syst. San Jose (1988) pp.491-499
- Heiss, H.-U.; Schmitz, M.: *Decentralized Dynamic Load Balancing: The Particles Approach.* Information Sciences, Vol. 84, Issue 1-2 (May 1995) pp. 115 - 128.
- Hinz,D.Y.: *A run-time load balancing strategy for highly parallel systems.* Acta Informatica 29,1 (Feb. 1992) pp.63-94
- Lin,F.C.H.; Keller,R.M.: *The Gradient Model Load Balancing Method.* IEEE TOSE 13,1 (January 1987) pp.32-38
- Cesar A. F. De Rose, Hans-Ulrich Heiss, and Philippe O. A. Navaux. Distributed Processor Allocation in Multicomputers.. In *IEEE Intl. Conference on Cluster Computing (CLUSTER'2000), 2000, Chemnitz, Germany*, pages 397--398, 2000.
- Tanner, A.: *Distributed Resource Management with Monetary Incentives*. Dissertation, Technische Universtität Berlin, August 2005.