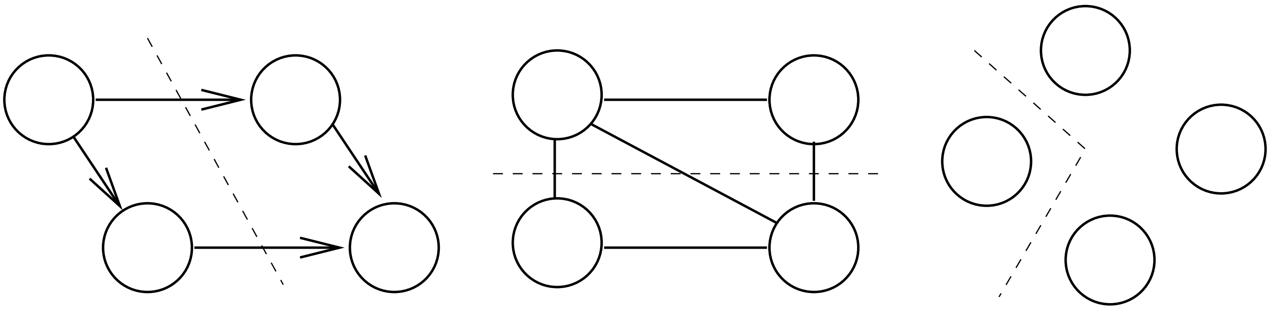
CHAPTER 5: DISTRIBUTED PROCESS SCHEDULING

Chapter outline

- Three process models:
  - precedence
  - communication
  - disjoint
- System performance model that illustrates the relationship among
  - algorithm, scheduling, and architecture
- Scheduling:
  - Static scheduling: precedence model, communication model
  - Dynamic scheduling: load sharing, load balancing
  - disjoint process model
  - interacting process model
- Implementation:
  - remote service and execution
  - process migration
- Real-time scheduling and synchronization
  - basic RTS
  - Priority Inversion
  - Blocking

Process models



(a) Precedence process model

(b) Communication process model

(c) Disjoint process model

Note: Dashed lines represent processor boundaries

- Precedence Process Model
  - Precedence relationship represented best by DAG
  - Suitable for Fork/Join or CoBegin/CoEnd code
  - Communication costs incurred if arc crosses processor boundary
  - Goal: Minimize makespan
- Interacting Process Model
  - Persistent processes that exhange messages asynchronously
  - Represented by graph showing processes and communication paths
  - Communication costs incurred if message crosses processor boundary
  - Goal: Minimize computation and communication costs
- Disjoint Process Model
  - Processes can run independently
  - Processes arrive independently
  - Queuing time and service time
  - Goal: Minimize turnaround time/processor idle time
  - If process migration, get load sharing/balancing at migration cost

A system performance model

Speed-up factor

$$S = F(Algorithm, System, Schedule)$$

$$S = \frac{OSPT}{CPT} = \frac{OSPT}{OCPT_{ideal}} \times \frac{OCPT_{ideal}}{CPT} = S_i \times S_d$$

- S = actual speedup on an n processor sytem
- $OSPT$ = Optimal Sequential Processing Time
- CPT = Actual Concurrent Processing Time
- $OCPT_{ideal}$ = Optimal Concurrent Processing Time
- S_i = ideal speedup
- S_d = degradation due to actual implementation

Ideal speed-up

$$S_i = \frac{RC}{RP} \times n$$

$$RP = \frac{\sum_{i=1}^{m} P_i}{OSPT}$$

$$RC = \frac{\sum_{i=1}^{m} P_i}{OCPT_{ideal} \times n}$$

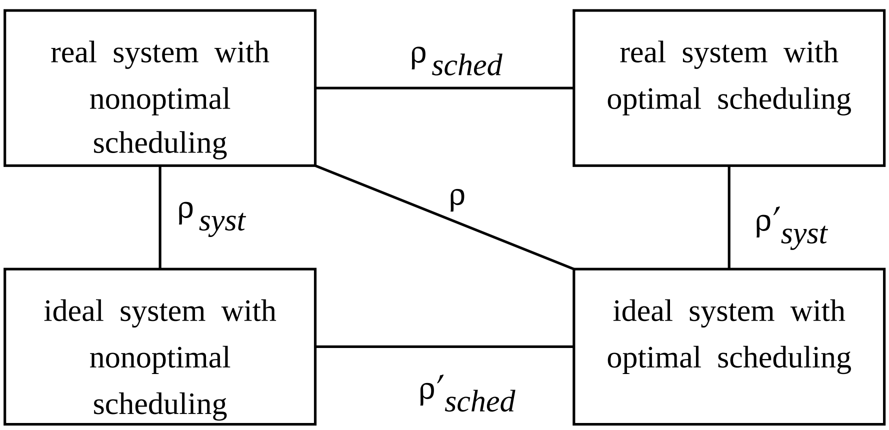
- S_i = ideal speedup on an n processor sytem
- RC = Relative Concurrency (processor utilization) ≤ 1
- RP = Relative Processing requirement ≥ 1
- n = number of processors
- m = number of tasks
- P_i = Computation time of task i

System degradation

$$S_d = \frac{1}{1+\rho}$$

$$\rho = \frac{CPT - OCPT_{ideal}}{OCPT_{ideal}}$$

- S_d = System degradation
- ρ = loss of parallelism on a real machine



Finally,

$$S = \frac{RC}{RP} \times \frac{1}{1+\rho} \times n$$

Amdahl's Law

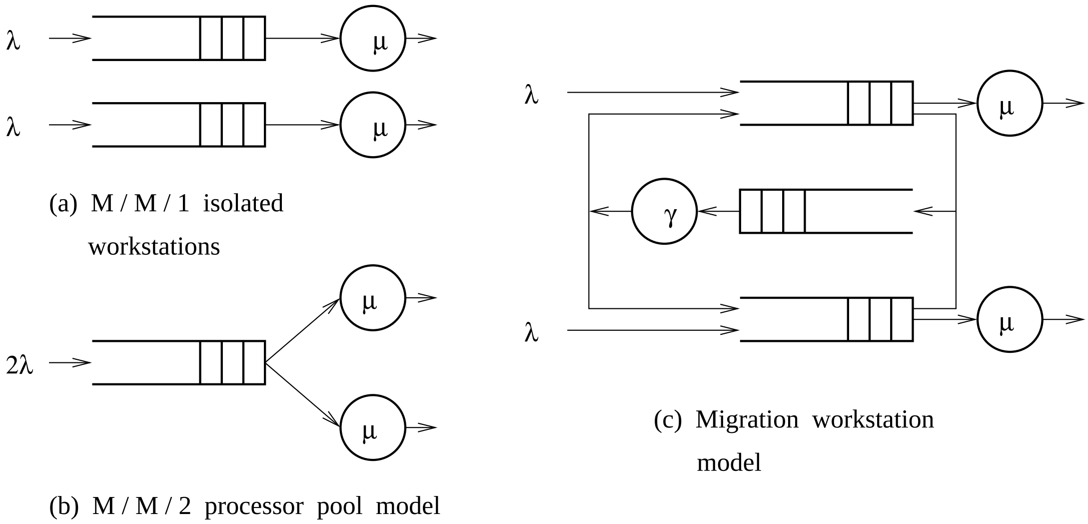
Speed-up is limited intrinsically by parallelizability of program.
If P is the parallelizable fraction of a program (in running time), and that part can be sped up a factor of S_p, then

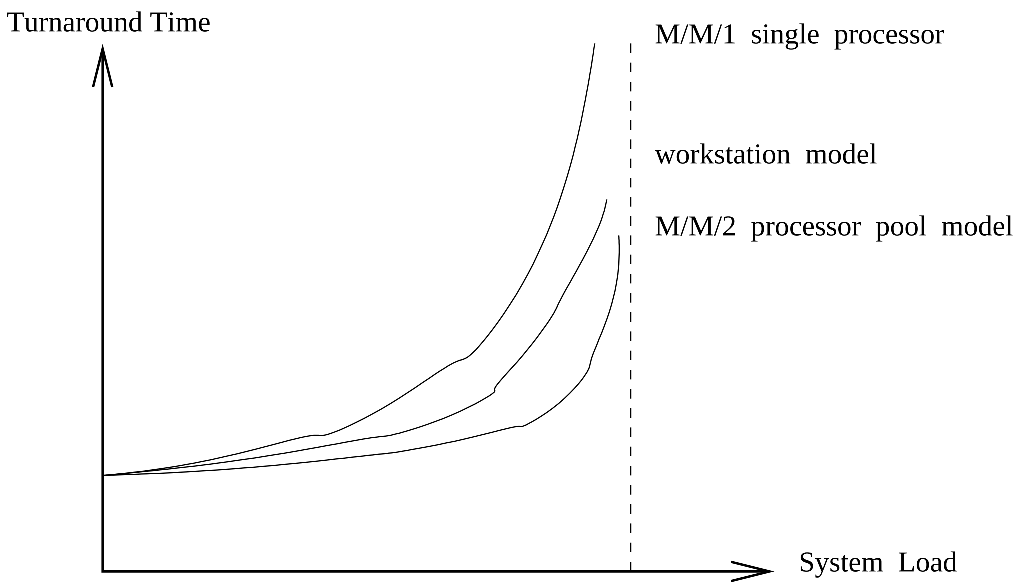
$$S_{tot} = \frac{1}{(1-P) + P/S}$$

Even if P can be sped up arbitrarily fast, depending only on the number N of processors used, the maximum speedup is still limited by

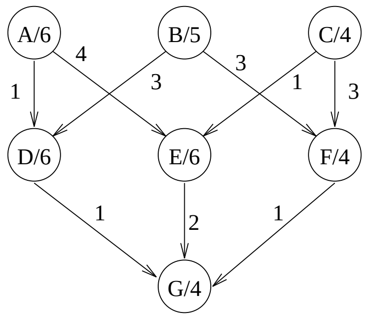
$$S_{tot} = \frac{1}{(1-P) + P/N}$$

Queuing System Comparisons

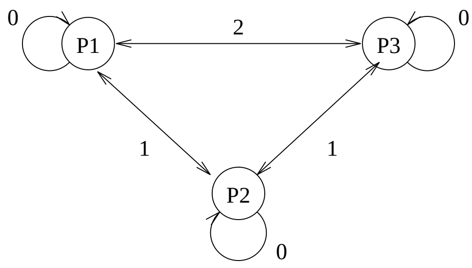




Static scheduling - Precedence process model

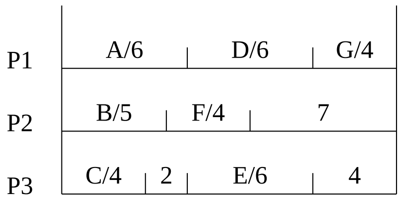


(a) Precedence process model



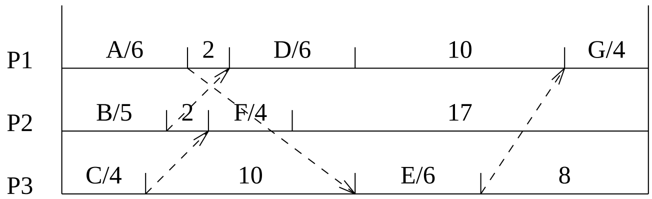
(b) Communication system model

(a) LS



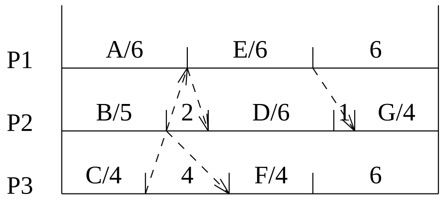
Makespan = 16

(b) ELS



Makespan = 28

(c) ETF



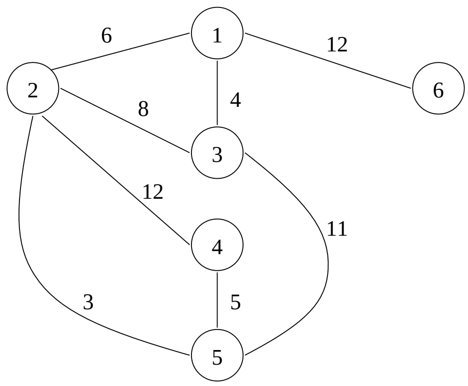
Makespan = 18

Note that List Scheduling has a lower makespan because it does not charge for communication costs at all (it represents a best case).
Also note that LS, Extended LS, and Earliest Task First are heuristics, and may not give an optimal schedule.

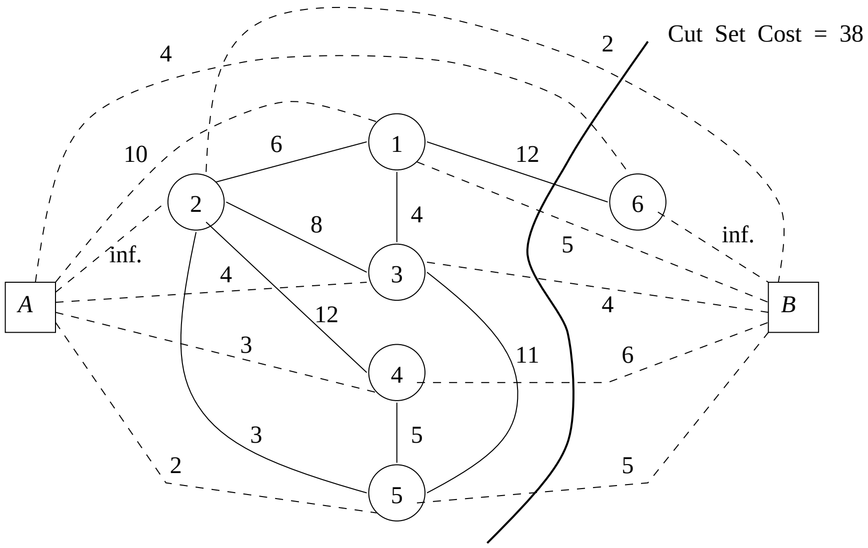
Communication process model

Process	Cost on A	Cost on B
1	5	10
2	2	infinity
3	4	4
4	6	3
5	5	2
6	infinity	4

(a) Computation cost



(b) Communication cost



For min-cut algorithms, see
http://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm, and
http://en.wikipedia.org/wiki/Edmonds-Karp_algorithm

Dashed line = cost to run task on *other* processor
(Task T in same set as A runs on A, so total cost must include cost of running T on A.)

Cluster tasks by Communication Threshold C
Limit # tasks (or computation cost) on processor by Task Threshold X

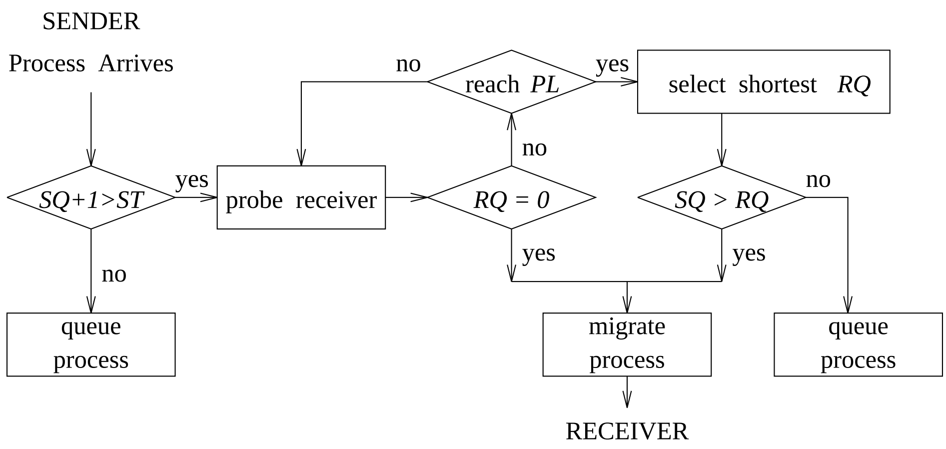
Objective: minimize cutset cost, representing aggregate cost (communication plus processing).
May lead to significant imbalance in processor loads.

Dynamic scheduling

Load sharing and balancing

Sender initiated algorithms (Push)

- **transfer policy**: When does a node become the sender?
- **selection policy**: How does the sender choose a process for transfer?
- **location policy**: Which node should be the target receiver?
  probe limit issues

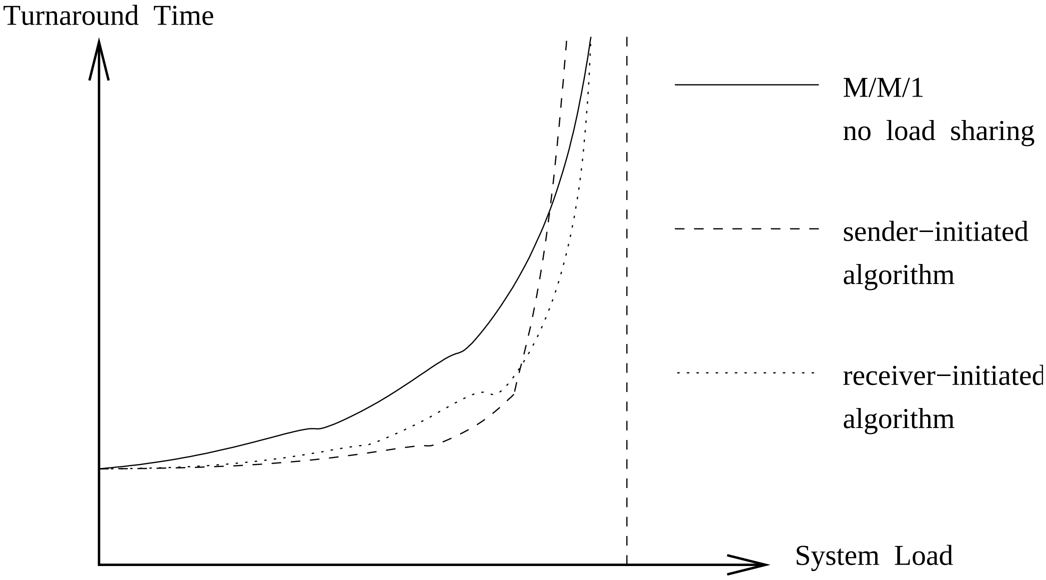


Receiver initiated algorithms (Pull)

- **transfer policy**: When does a node become a receiver?
- **location policy**: How does the receiver choose a heavily loaded node?
  probing
- **selection policy**: How does the source choose a process for transfer?
  requires preemption (processes have already started)

Hybrid algorithms and their performance

- **Push**: Best at low to moderate loads, unstable at high loads
- **Pull**: Best at higher loads, never unstable
- **Hybrid**: Sense system load, select algorithm accordingly
- **Brokered**: Server to hold load info, schedule jobs



Remote process implementation

Remote service

- As remote procedure calls at the language level
- As remote commands at the operating system level
- As interpretive messages at the application level

Remote execution

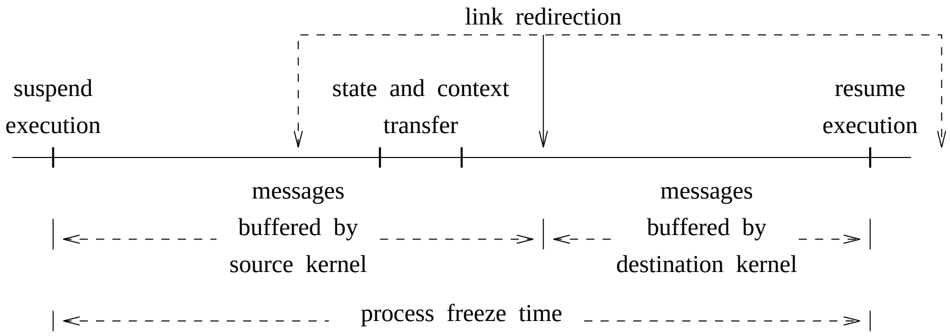
The remote operation initiated by a client is created by the client for resource or load sharing (processor-pool model).

- Load sharing algorithm
- Location independence
- System heterogeneity
- Protection and security

Process migration

Preemption and reconfiguration

link redirection and message forwarding



state and context transfer

freeze time and residual computation dependency

Pointer forwarding through DNS

soft (symbolic) and hard links

Real-time scheduling

Soft/hard deadlines, periodic/aperiodic, priority scheduling

Task τ_i described by

$$\tau_i \;=\; (S_i, C_i, D_i)$$

S_i = earliest start time
C_i = worst case execution time
D_i = deadline

(Aperiodic) Realtime Task Set V is

$$V \;=\; \{\tau_i \mid i = 1, 2, ..., n\}$$

Periodic Realtime Task Set V is

$$V \;=\; \{J_i = (C_i, T_i) \mid i = 1, 2, ..., n\}$$

C_i = worst case execution time
T_i = period

Schedule A is set of execution intervals

$$A \;=\; \{(s_i, f_i, t_i) \mid i = 1, 2, ..., m\}$$

s_i = start time of interval i
f_i = finish time of interval i
t_i = task executed during interval i

A is valid iff:
1. $\forall i = 1, 2, ..., m, \;\; s_i \leq f_i$
2. $\forall i = 1, 2, ..., m, \;\; f_i \leq s_{i+1}$
3. $\forall i = 1, 2, ..., m$, if $t_i = k$, then $S_k \leq s_i$ and $f_i \leq D_k$

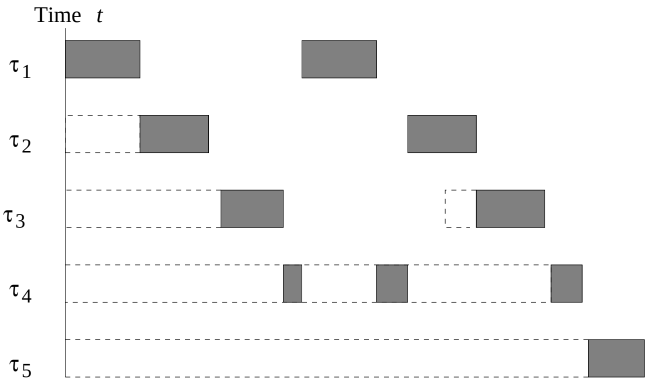
A schedule A is feasible iff every task receives at least C_k time in A
A RT Task Set V is feasible iff there is a feasible schedule for it.

Priority scheduling

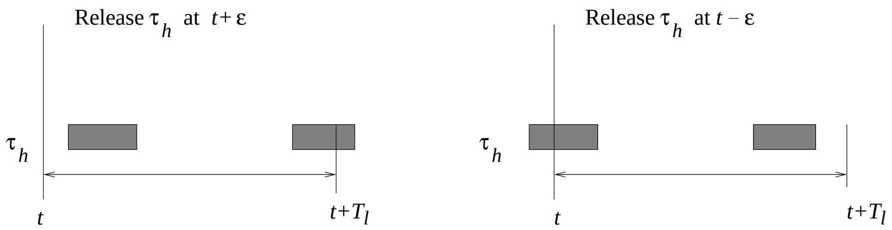
- Rate monotonic priority assignment: task period
- Deadline monotonic priority assignment: deadline
- Earliest deadline first: dynamic deadline

Static Priority Assignment

Critical Instant



Critical instant for τ_5

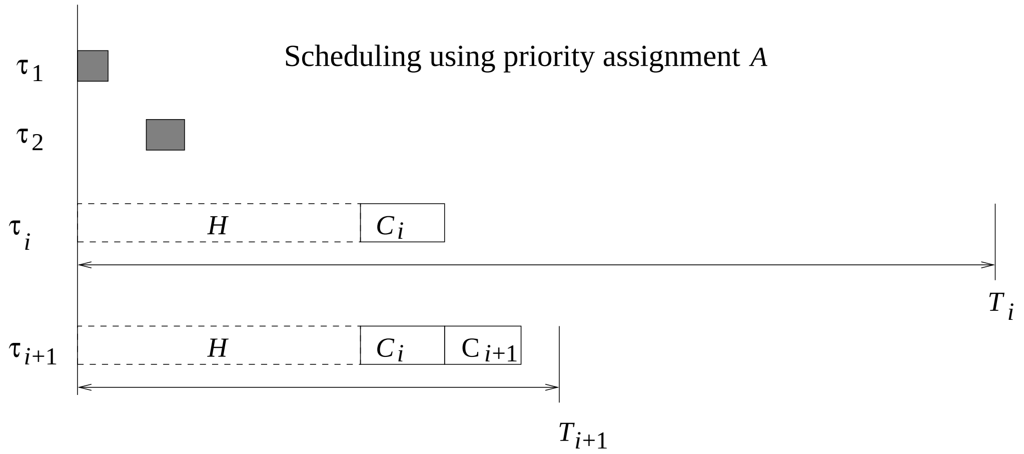


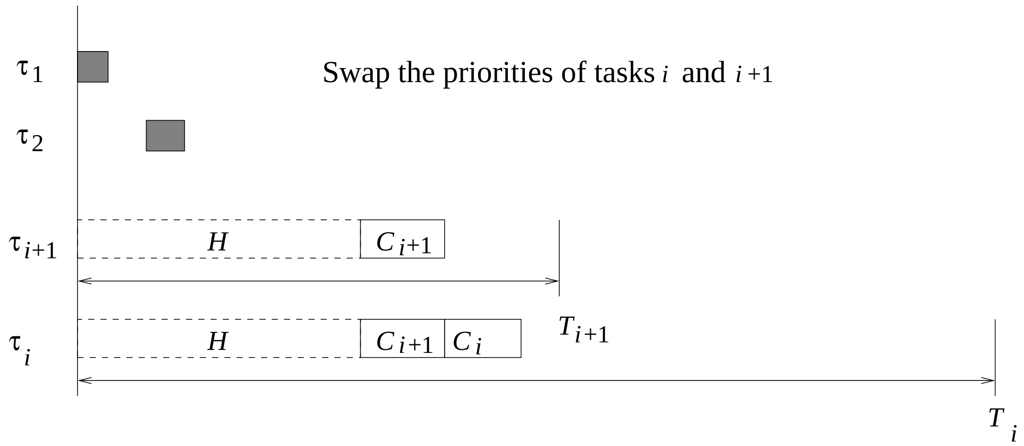
Scheduling a higher priority task earlier or later can only make its demand on the processor less.

Rate Monotonic (RM) priority assignment

Optimal static priority assignment

$$T_h < T_i \text{ then } Pr_h > Pr_i$$





If task τ_{i+1} could finish by time T_{i+1} in the first priority assignment, then it finishes by time $T_{i+1} - C_i$ in the Rate Monotonic assignment. Likewise, τ_i will finish by $T_{i+1} < T_i$ in the Rate Monotonic assignment, so both will still be satisfied.

A sufficient (but not necessary) condition for feasible RM schedule, Load L is:

$$L \;=\; \sum_{i=1}^{n} \frac{C_i}{T_i} \;\leq\; n(2^{1/n} - 1)$$

which approaches 0.69 as n becomes large. This is pessimistic.

Response time r_i of task i must satisfy:

$$r_i \;=\; C_i + \sum_{h=1}^{i-1} \lceil \frac{r_i}{T_h} \rceil C_h \;\leq\; T_i$$

Find r_i iteratively (start with $r_i(0) = C_i$), and if ever $r_i(k) > T_i$ for some iteration k, then fail.
If $r_i(k) = r_i(k-1) <= T_i$ for some iteration k, then we have reached a fixed point and the RM schedule is feasible.

Deadline Monotonic (DM) priority assignment

When periodic task deadlines are sooner than next start time

$$V \;=\; \{J_i = (C_i, T_i, D_i) \mid i = 1, 2, ..., n\}$$

C_i = worst case execution time
T_i = period
D_i = incremental deadline (if arrive at t, must finish by $t + D_i$)

$$D_h < D_i \text{ then } Pr_h > Pr_i$$

Optimal for static priority assignments for short deadline periodic tasks

Earliest Deadline First (EDF) priority assignment

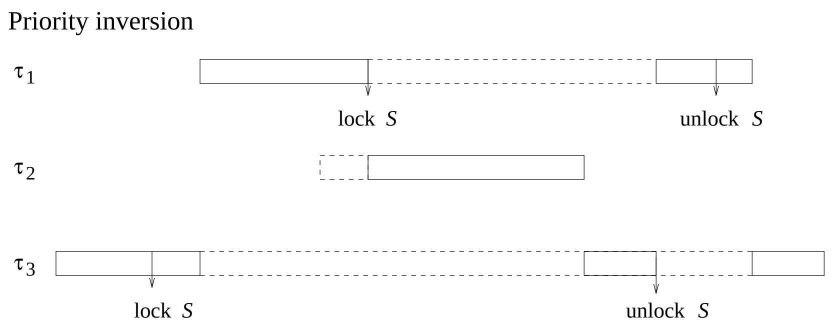
Optimal dynamic scheduling

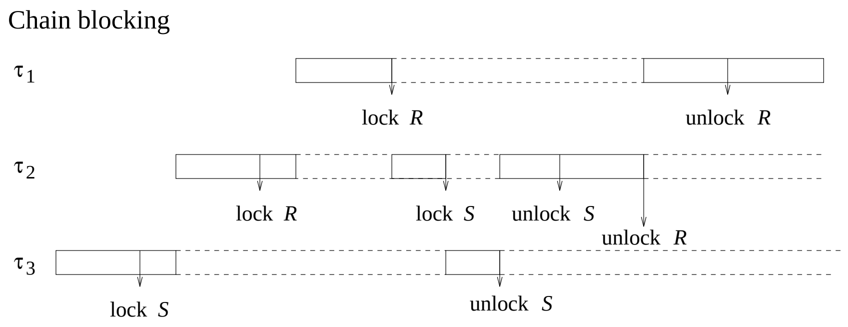
$$d_h(j) < d_i(k) \text{ then } Pr_h(j) > Pr_i(k)$$

$d_h(j)$ = deadline for j^{th} instance of task h
$Pr_h(j)$ = priority for j^{th} instance of task h

Real-time synchronization

Priority Inversion:
Direct:
τ_2 has higher priority than τ_3, so it effectively blocks τ_1 by running instead of τ_3, which holds lock S that τ_1 needs.

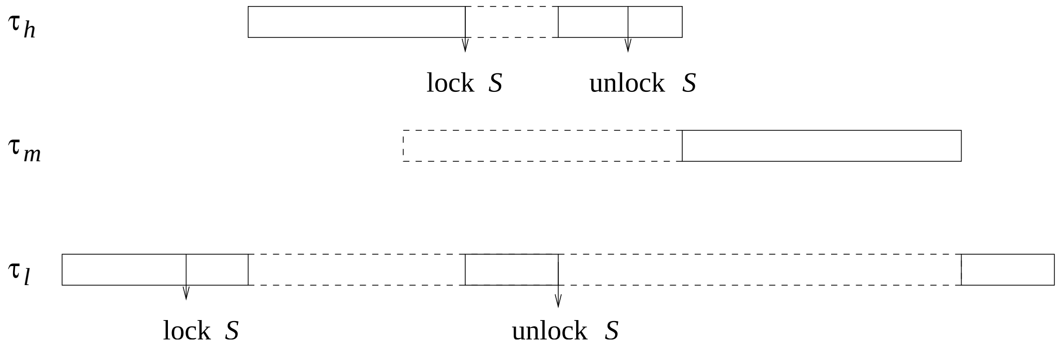




Indirect - Chain Blocking:
τ_3 blocks τ_1 through τ_2, even though it does not hold a lock needed by τ_1

PIP - Priority Inheritance Protocol:
When τ_h blocks on S (directly or indirectly), τ_l inherits the priority of τ_h (transitively).

Direct and Push-through Blocking:
τ_h suffers direct blocking;
τ_m suffers push-through blocking due to PIP (this is desired).



Let us find B_h = maximum time τ_h will be blocked

Task-oriented perspective:

$$B_h \leq \sum_{\{l|Pr_l<Pr_h\}} E_h(l)$$

$E_h(l)$ = maximum execution time of any critical section in $Bl_h(l)$
$Bl_h(l)$ = set of all critical sections of τ_l that can block τ_h (directly, indirectly, or push-through)

Lock-oriented perspective:
Since a higher priority task can only be blocked once by a lower priority task per semaphore S,

$$B_h \leq \sum_{\{S|ceil(S)\geq Pr_h\}} E_h(S)$$

$E_h(S)$ = maximum execution time of any critical section of a task with lower priority than τ_h that can block τ_h through S
Priority ceiling of S, $ceil(S) = max\{Pr_h|\tau_h$ can be blocked by $S\}$.
If no nested critical sections, then $ceil(S) = max\{Pr_h|\tau_h$ locks $S\}$.

So, for schedulability, take the smaller upper bound B_h.
A sufficient condition for a feasible RM assignement is then:

$$\sum_{j=1}^{h} \frac{C_j}{T_j} + \frac{B_h}{T_h} \ \leq \ h(2^{1/h} - 1)$$

More specifically, compute response time r_i for each task i iteratively:

$$r_i(0) \ = \ C_i + B_i$$

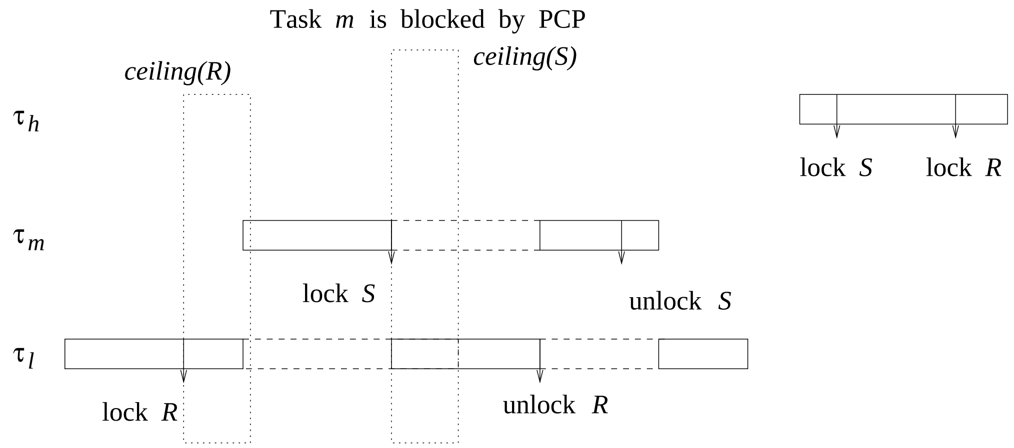
$$r_i(k+1) \ = \ C_i + B_i + \sum_{h=1}^{i-1} \lceil \frac{r_i(k)}{T_h} \rceil C_h \ \leq \ T_i$$

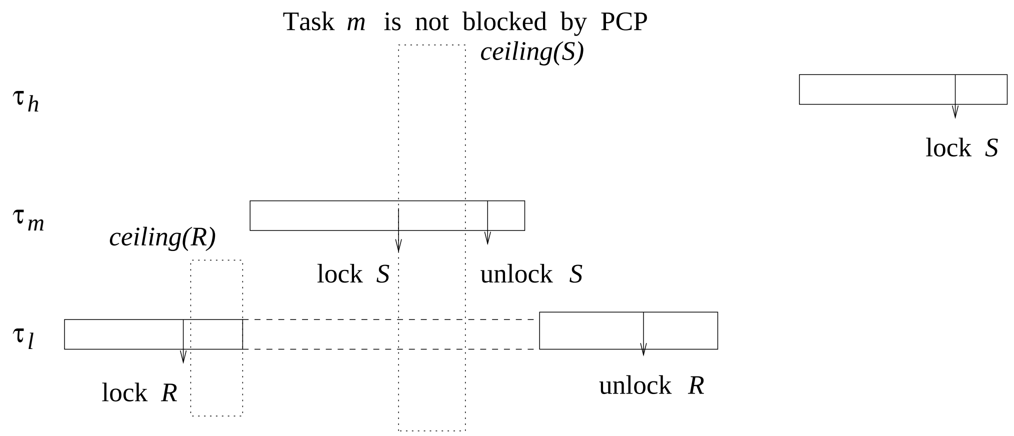
If ever $r_i(k) > T_i$ for some iteration k, then fail.
If $r_i(k) = r_i(k-1) <= T_i$ for some iteration k, then we have reached a fixed point and the RM schedule is feasible.

However, we are allowing a task to be blocked by more than one lower priority task or lock, making schedulability less certain....

PCP - Priority Ceiling Protocol:
Insure no process is blocked for more than one critical section by a lower priority process.

Add the following rule to PIP:
τ_i is granted lock S iff $Pr_i \geq ceil(R)$ for every lock R held by a different task.





So we get a (possibly) smaller value for B_h:

$$B_h \leq \max\{E_h(S)|ceil(S) \geq Pr_h\}$$