Coverage

- Introduction to real-time systems
- Performance measures
- Issues in real time computing
- Task allocation and scheduling techniques
- Power and energy issues
- Communication algorithms
- Fault tolerance and reliability evaluation
- Clock synchronization

Today's Topics

- What is a real-time system?
  - General characteristics
  - Hard and soft real-time systems
- Performance Measures
  - Why are they important?
  - For general-purpose systems
  - For real-time systems
- Uniprocessor task scheduling

What is a Real-Time System?

- Any system in which a deadline plays a central role in its perceived performance
  - But timely response is important for general-purpose systems, too!
  - There is no hard-and-fast demarcation between a real-time system and a general-purpose system
  - Systems in the control loop are always real-time

Introduction to Real-Time Systems

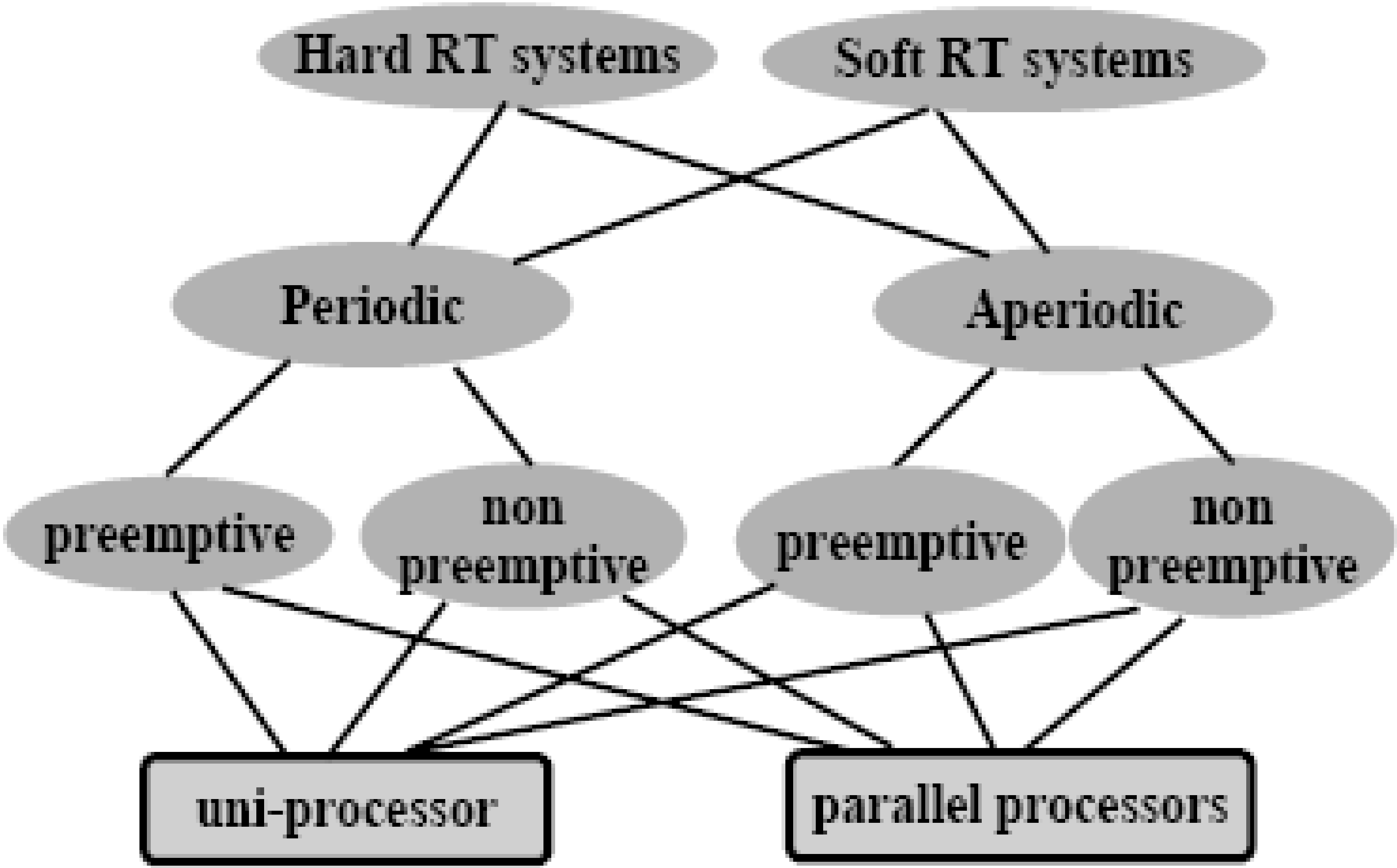
<u>What is a Real-Time System?</u>

Is defined as a system in which the time where the outputs are produced is significant (within specified bounds or deadlines)

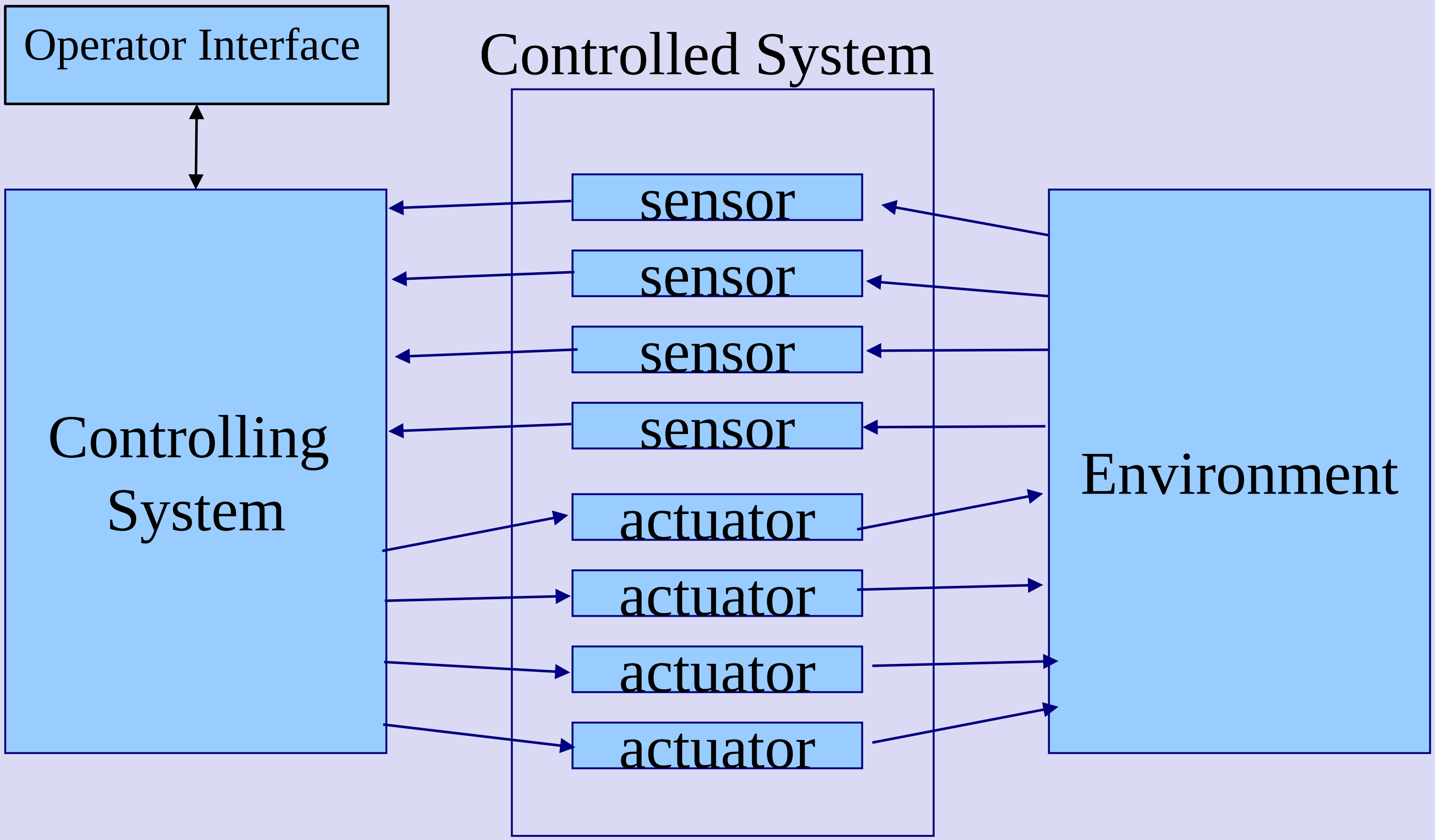
.

Correctness depends on output values and the time at which the inputs are processed and the outputs are produced

Introduction to Real Time systems



Typical Real-Time System



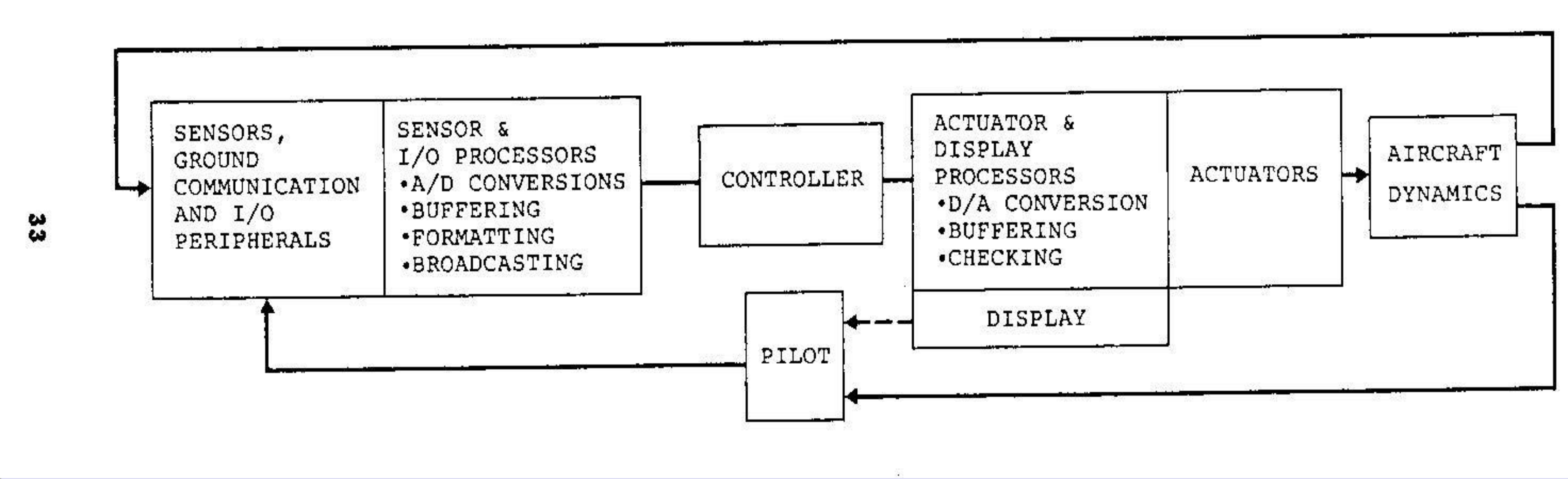
Types of RTS

- Hard Real-Time Systems
  - Missing a deadline (or series of deadlines) can cause a significant loss to the application.
  - Examples: Fly-by-wire, power-plant, and grid control
- Soft Real-Time Systems
  - Missing a deadline causes the quality of service to degrade, but nothing terrible happens
  - Examples: Video-on-demand, teleconferencing

Example: Fly-by-wire

- Used initially in military aircraft
  - » Dynamics time-constants are too small for humans to be effective controllers
  - » Philosophy:
    - Pilot sets policy
    - Computer carries out low-level actions to implement that policy
  - » If too many deadlines are missed in a row, the aircraft can crash

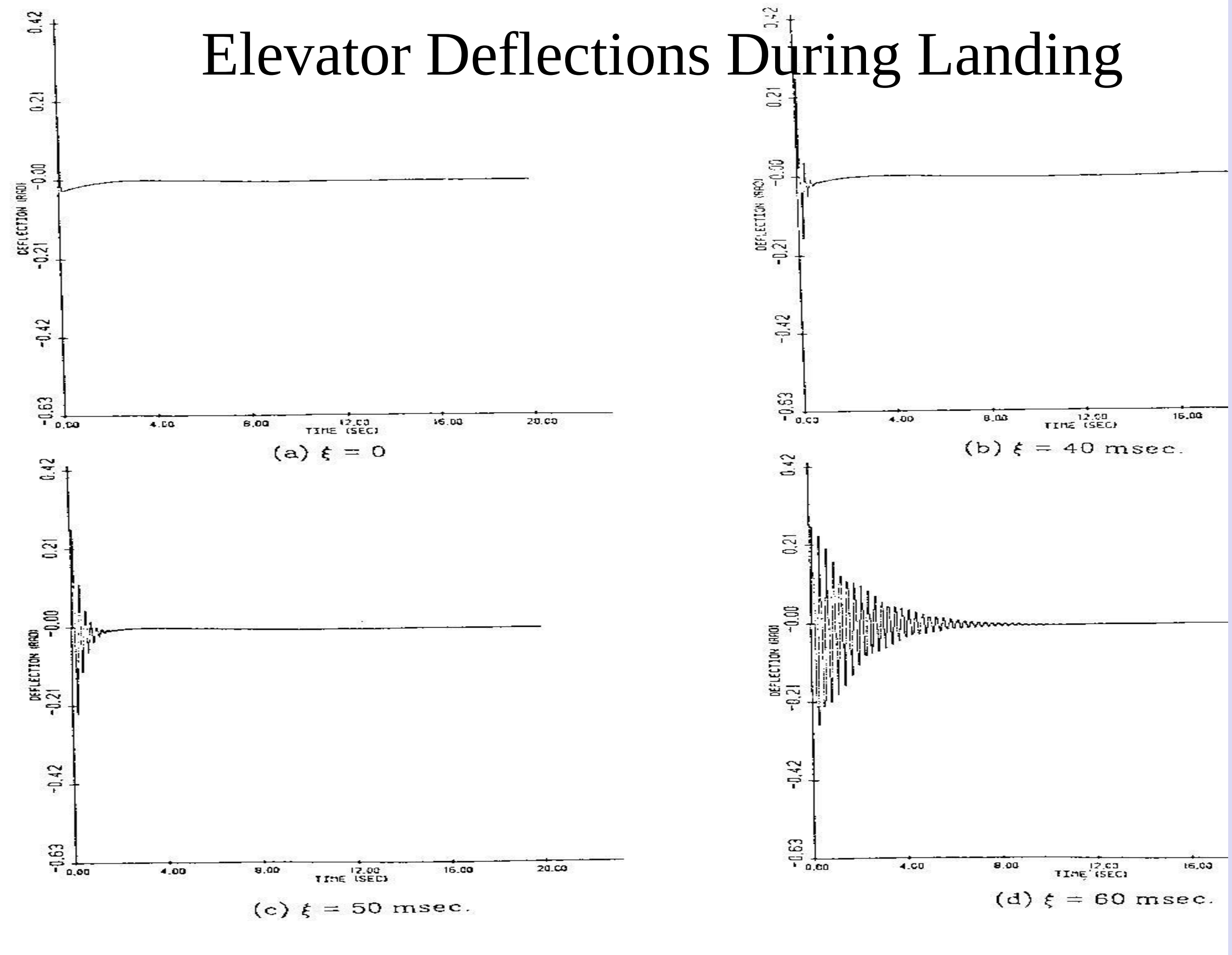
Feedback Loop



(From C. M. Krishna & K. G. Shin: NASA Con. Report 3807, 1984)

Impact of Feedback Delay (Simulation Example)

Elevator Deflections During Landing



Issues in real time computing

- Real time response
- Recovering from failures
- Working with distributed architecture
- Asynchronous communication
- Race conditions and timing

Scheduling RT Tasks with FT Requirement

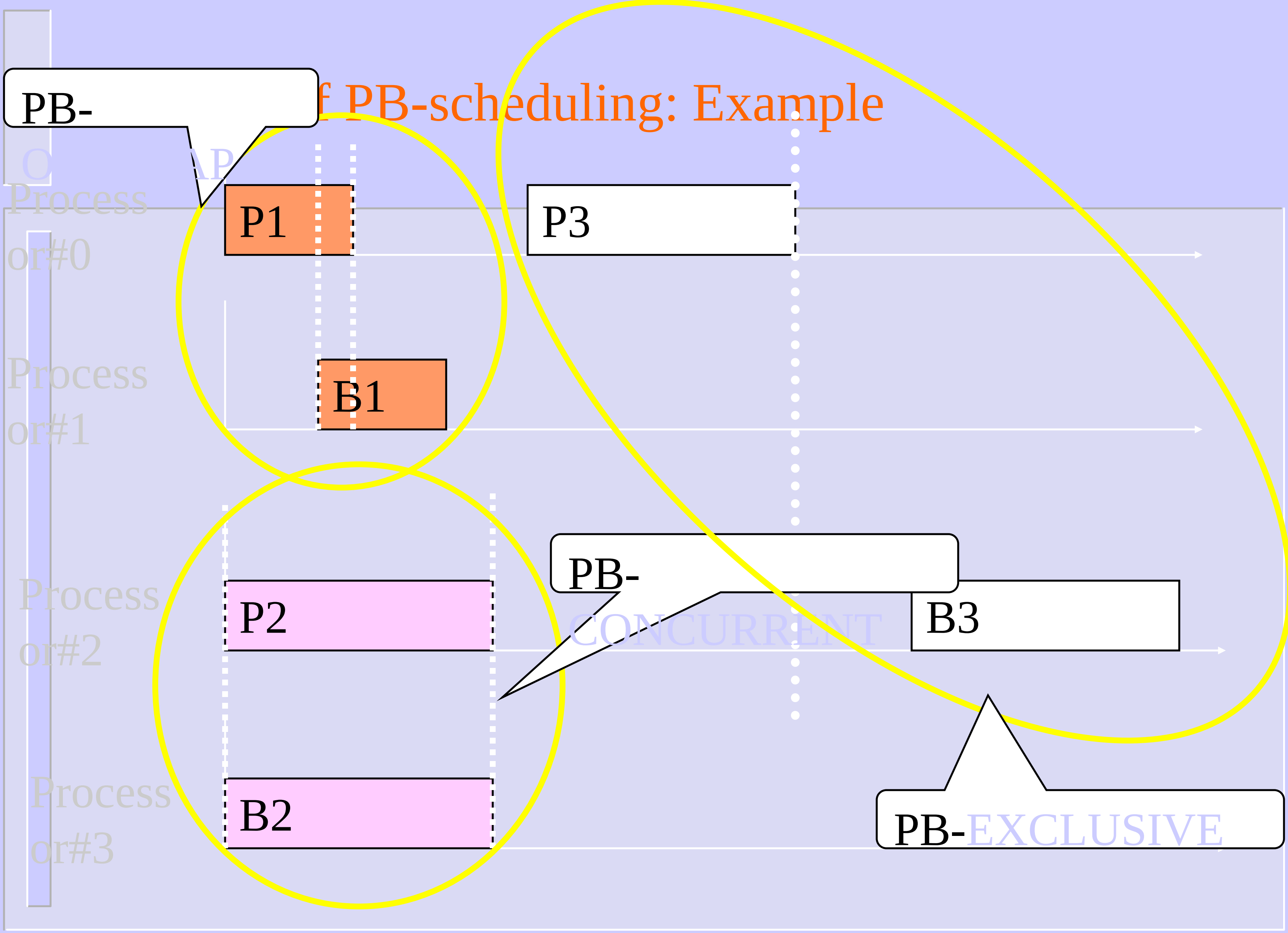
PB-based Fault-Tolerance

- Space exclusion – primary and backup scheduled on two different processors.
- Time exclusion – primary and backup should not overlap in execution.

Variants of PB-based Scheduling

- PB-Exclusive – Both time and space exclusion
- PB-Concurrent – Space exclusion, but concurrent execution
- PB-Overlap – Space exclusion, but overlap in execution

f PB-scheduling: Example



Scheduling RT Tasks with FT Req. (contd.)

- Each of the above three schemes has merits under certain workload and fault scenarios.
  - PB-Concurrent: at high fault rates, tight deadlines
  - PB-exclusive: at low fault rates, relaxed deadlines, high resource needs
- Generalized scheme
  - That adapts (estimating) the "primary-backup overlap interval" based on task parameters (e.g., deadline) and fault rate has the potential to offer the best schedulability under all scenarios.

Schedulability-Reliability Tradeoff

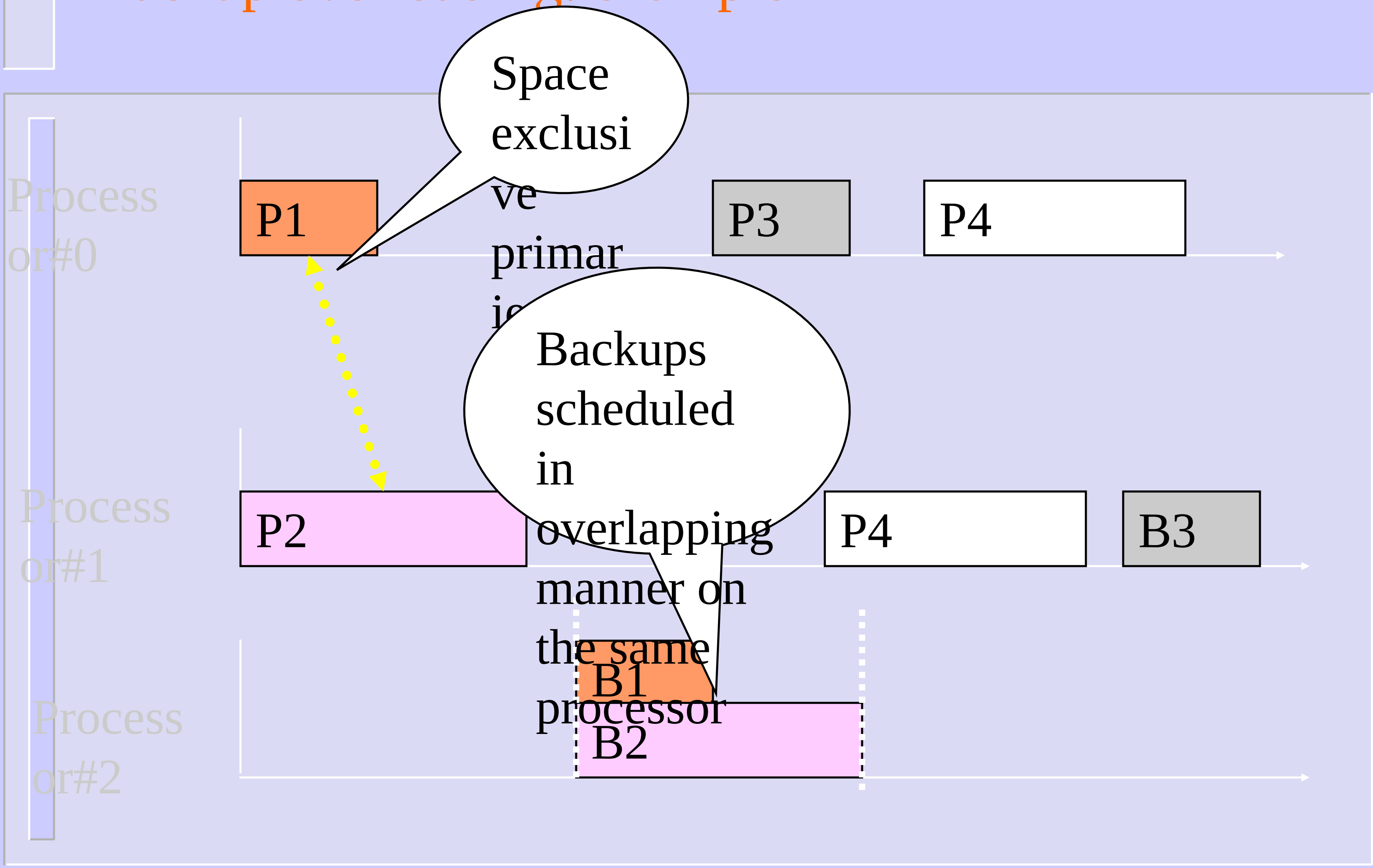
- Too much redundancy increases reliability, but it could potentially decrease the schedulability.
- Too little redundancy decreases reliability, but increases schedulability
- Also, designing and managing redundancy incurs additional cost, time, space, and power consumption
- Therefore, appropriate use of redundancy is important

Schedulability Enhancement Techniques in PB-based FT scheduling

- Backup overloading
  - Two backups can be scheduled in a overlapping manner if their primaries achieve space exclusion.
  - Assumes, at most only one fault at a given time, i.e., before the second fault, the first fault is recovered.

- Flexible overloading (static-grouping)
  - Partition the processors into groups
  - Schedule the primary and its backup in the same group
    - If primary is scheduled in group 1, its backup must also be scheduled in the group exploiting the backup overloading

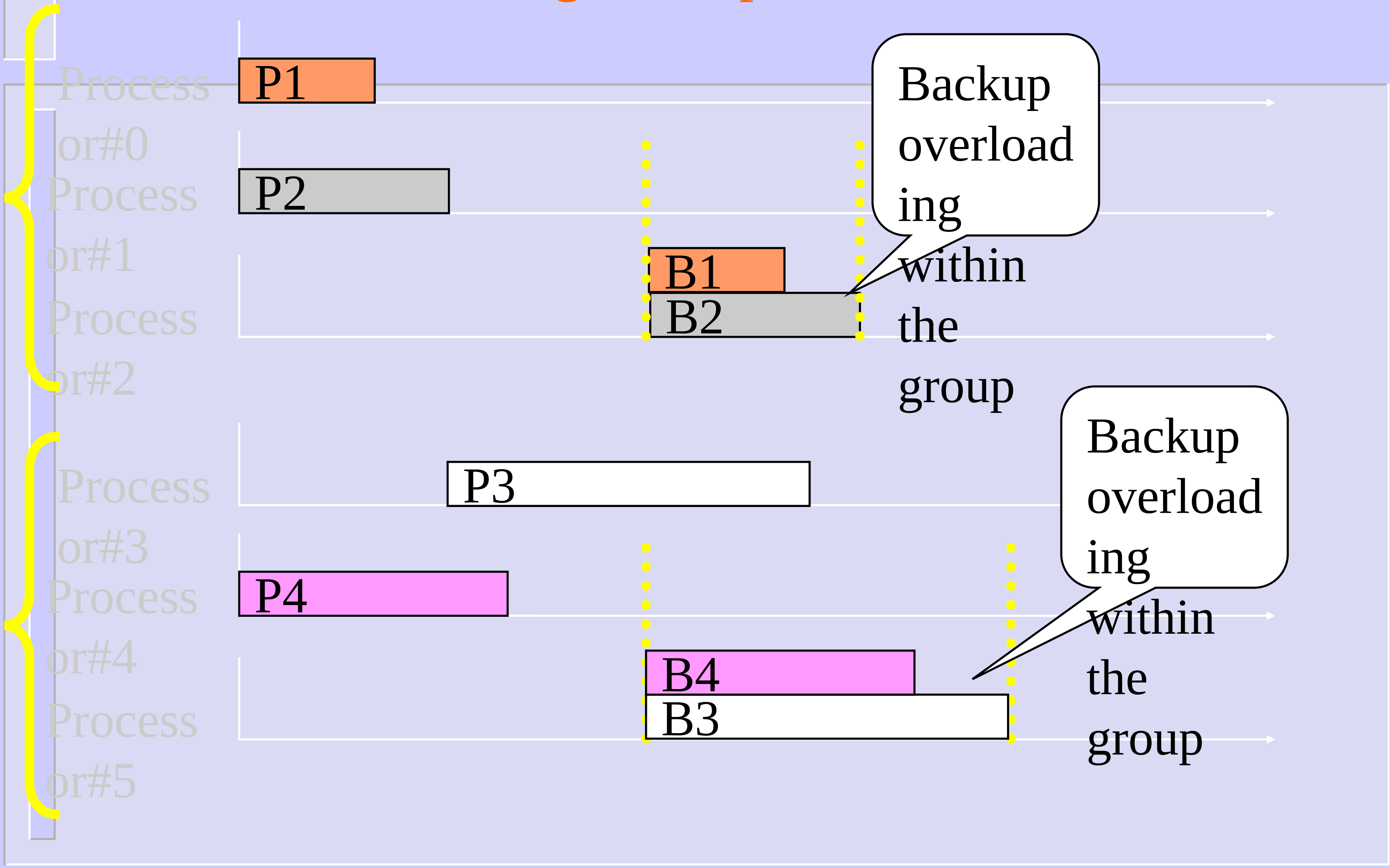
Backup overloading: example



Flexible overloading- details

- In flexible overloading, all “m” processors are partitioned into different groups
- Rules
  - Every processor is a member of exactly one group
  - For backup overloading to take place in a group, it must have at least three processors
  - The size of each group is the same (except for one group, when (m/gsize) is not an integer)

Flexible overloading: example



Distance concept: details

- Distance concept – the relative position of a primary task and its backup task in the task queue

- For a given set of "N" active tasks and a given distance of "d"

For all tasks, T_i

» Distance (Pr_i, Bk_i) is equal to

- d for the (N – (N mod d)) tasks
- N mod d for the (N mod d) tasks

Distance concept: example

- N= 4
- d = 3

P1	P2	P3	B1	B2	B3	P4	B4

- N= 4
- d = 2

P1	P2	B1	B2	P3	P4	B3	B4

The distance concept introduces a tradeoff between performance and fault tolerance in the myopic algorithm.

Distance should be appropriately chosen. The distance should be neither too low nor too high

Distance - some implications

- Backup postponement
  - If backup task is too closer (in queue position) relative to its primary, holes get created in the schedule, resulting in lower schedulability.
- Forced backtrack
  - If backup task is too far (in queue position) relative to its primary, missing the deadlines of backup could happen which would result in backtrack.

Performability measures

- Which is better? High schedulability or high relaibility
- Overall system metric is required
- Performability metrics combine schedulability and reliability into a single metric that captures the overall system goal
- Goal: Determining Redundancy level to maximize the performance index (PI)

Determining Redundancy Levels (contd.)

- Performance index (PI) is a measure that captures both performance and reliability requirements
- PI is defined as follows: For a task Ti,

$$PI_i = \begin{cases} V_i * R_i - P_i * F_i & \text{if } T_i \text{ is guaranteed} \\ -Q_i & \text{if } T_i \text{ is not guaranteed} \end{cases}$$

Where,

Vi = reward if Ti completes successfully
Ri = reliability of a task (1 – Fi)
Fi = Failure probability
Pi = penalty if Ti fails after being guaranteed
Qi = if Ti has not been guaranteed

Determining Redundancy Levels

- Goal:

  Given the relevant parameters for each of the "n" tasks to be scheduled on a set of "m" processors, the goal is to determine the appropriate redundancy levels for each task in order to maximize the total PI.

- Let Ri be the reliability of the task with one version, the reliability of the task with "n" versions is given by

  $1 - (1 - Ri)^{n}$

Determining Redundancy Levels: example

Given task-set

Task (Ti)	Task attributes	Penalty/reward
T1, T2, T3, T4	$R_i = 0$, $C_i = 10$, $D_i = 10$	V_i = … Q_i …

Calculations

U	$PI = \sum PI_i$
1	4 (10 * 0.9 – 100 * 0.1) = -4
2	2(10 * 0.99 – 100 * 0.01) - 2 = 16
3	1(10 * 0.999 – 100 * 0.001) – 3 = 7
4	1(10 *0.9999 – 100 * 0.0001) – 3 = 7

PI is maximum at u = 2. Therefore, a redundancy level of 2 is optimal

U: redundancy level

Fault-tolerance -- conclusions

- Dependability concepts
- Fault-tolerant design techniques

- Fault-tolerant scheduling
  - Primary-backup scheduling
  - Schedulability enhancement techniques
  - Redundancy level determination

Performance Measures

- Traditional Measures
  - Throughput: Average number of instructions processed per second
  - Availability: Fraction of time for which the system is up
  - Reliability: Probability that the system will remain up throughout a designated interval

Special-Purpose Measure

- Performability
  - Published by John Meyer in 1980
  - Identify accomplishment levels, {A0, A1, A2, …, An}, for the application
  - Determine the probability, P(Ai), that the real-time system will be able to perform in such a way that Ai will be accomplished
  - Performability is the vector (P(A0), P(A1), …, P(An))
  - Application-focused measure

Task Allocation and Scheduling

- How to assign tasks to processors and to schedule them in such a way that deadlines are met
- Our initial focus: uniprocessor task scheduling

Uniprocessor Task Scheduling

- Initial Assumptions:
  - Each task is periodic
  - Periods of different tasks may be different
  - Worst-case task execution times are known
  - Relative deadline of a task is equal to its period
  - No dependencies between tasks: they are independent
  - Only resource constraint considered is execution time
  - No critical sections
  - Preemption costs are negligible
  - Tasks must be completed for output to have any value

Standard Scheduling Algorithms

- Rate-Monotonic (RM) Algorithm:
  - Static priority
  - Higher-frequency tasks have higher priority
- Earliest-Deadline First (EDF) Algorithm:
  - Dynamic priority
  - Task with the earliest absolute deadline has highest priority

Rate Monotonic Algorithm

- Example
- Schedulability criteria:
  - Sufficiency condition (Liu & Layland, 1973)
  - Necessary & sufficient conditions (Joseph & Pandya, 1986; Lehoczky, Sha, Ding 1989)