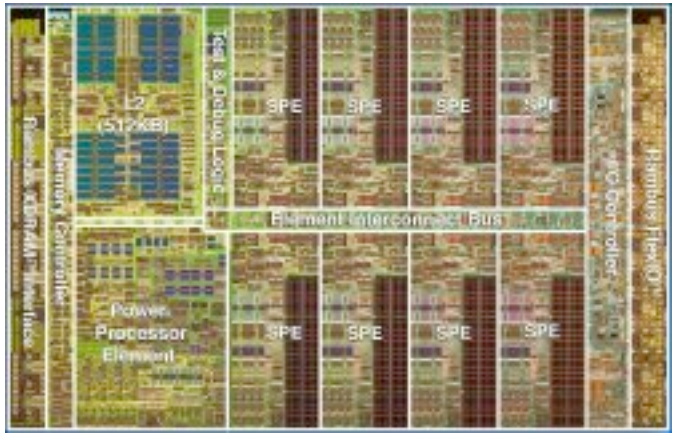
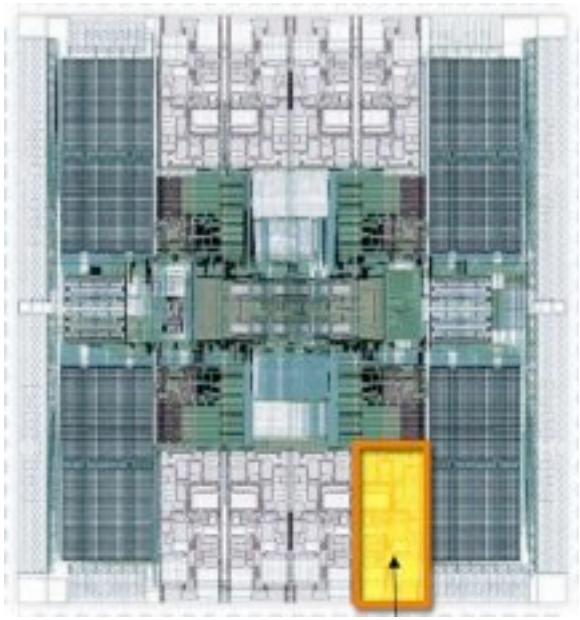
Multiprocessor platforms for real-time systems

Why?
Models of multiprocessor systems
Scheduling policies for multiprocessor systems
Schedulability tests

Multiprocessors (SMT)





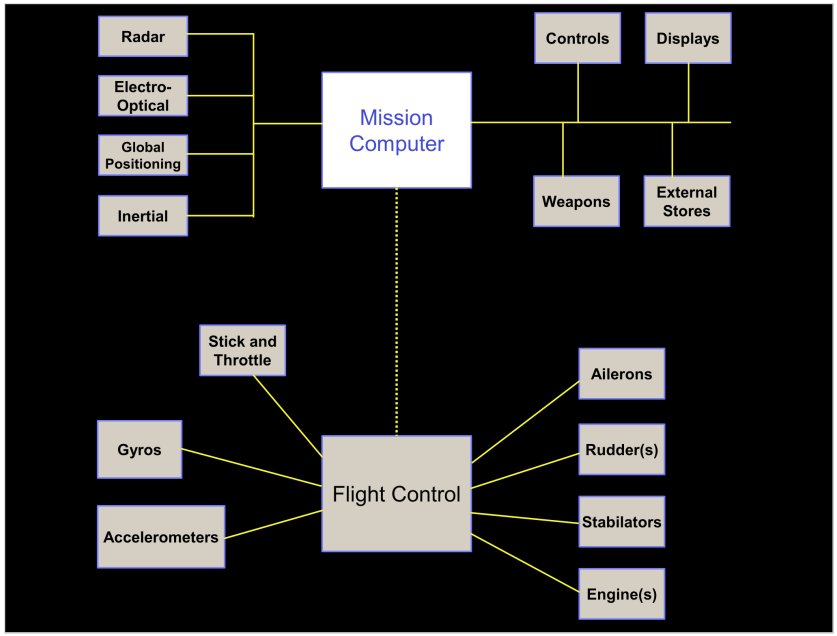
Sony/IBM/Toshiba cell processor

- One dual threaded, dual issue in-order PowerPC core @3.2 GH
- 8 Processing Elements, each with 256k local store, a vector Single Precision FPU and a conventional Double Precision FPU @3.2 GHz
- Bi-directional ring interconnect between all 9 PEs
- Rambus XDR memory controller

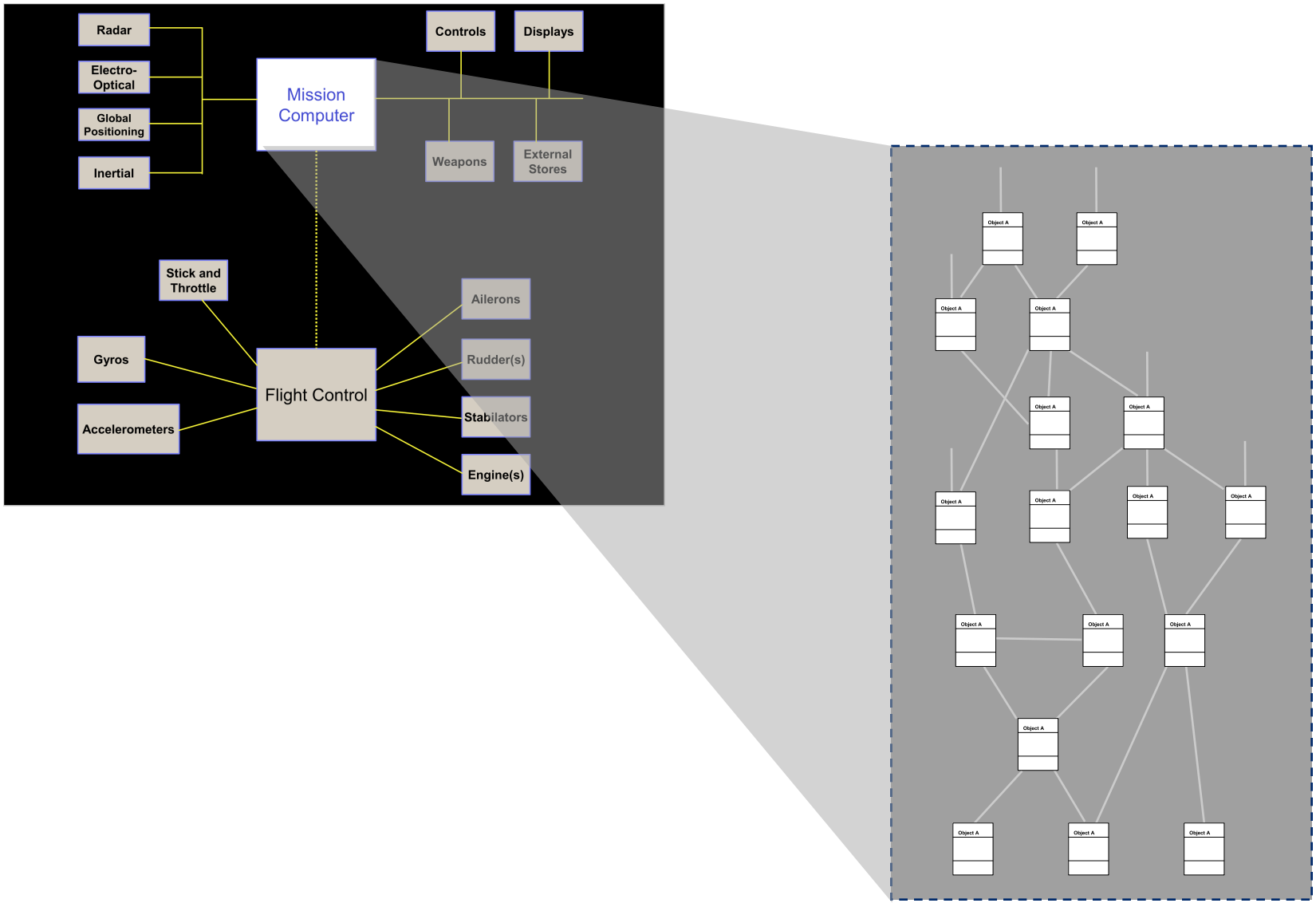
Sun UltraSparc T1 - Niagara

- Fine-grained chip multithreading
- Eight cores on one die (90nm technology)
- Each core has one pipeline that can support four threads in parallel with *zero context switch overhead*

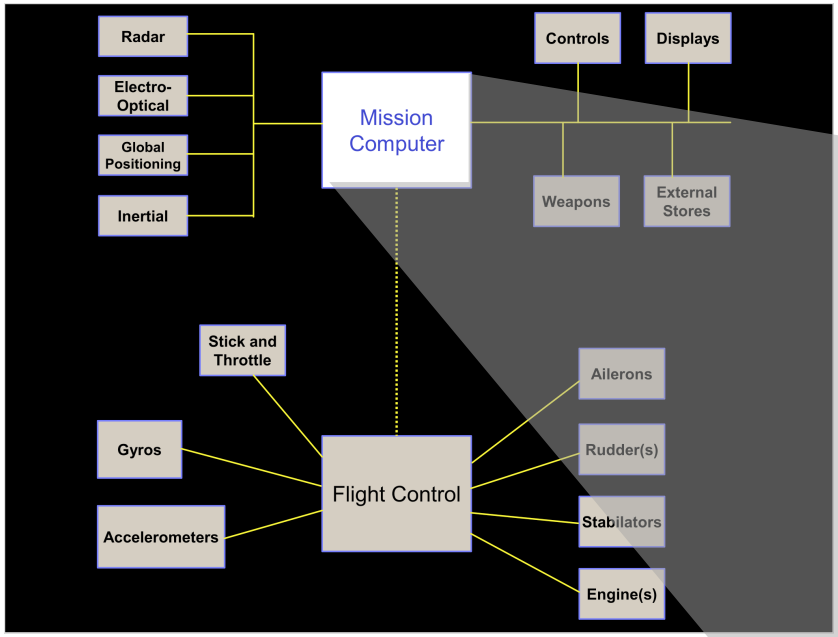
Multiprocessors in avionics systems

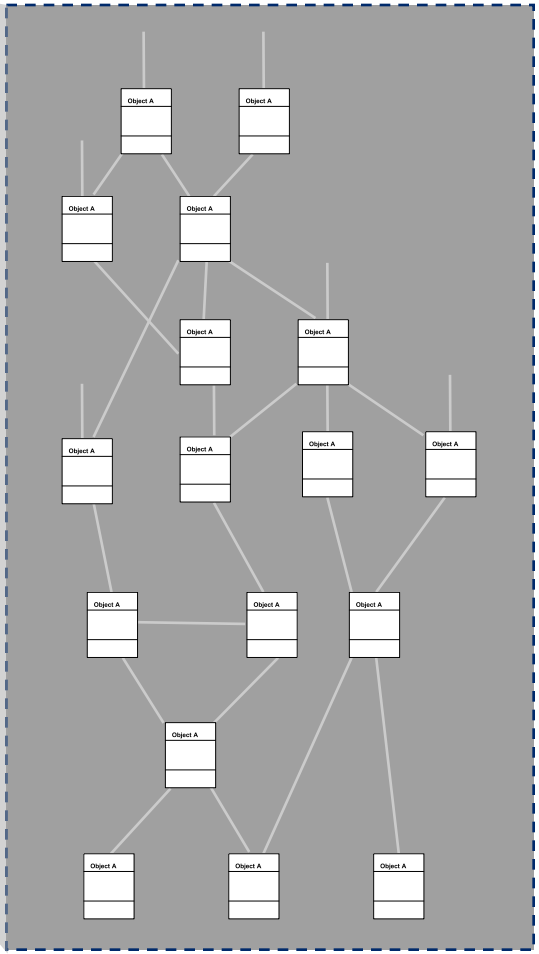


Multiprocessors in avionics systems



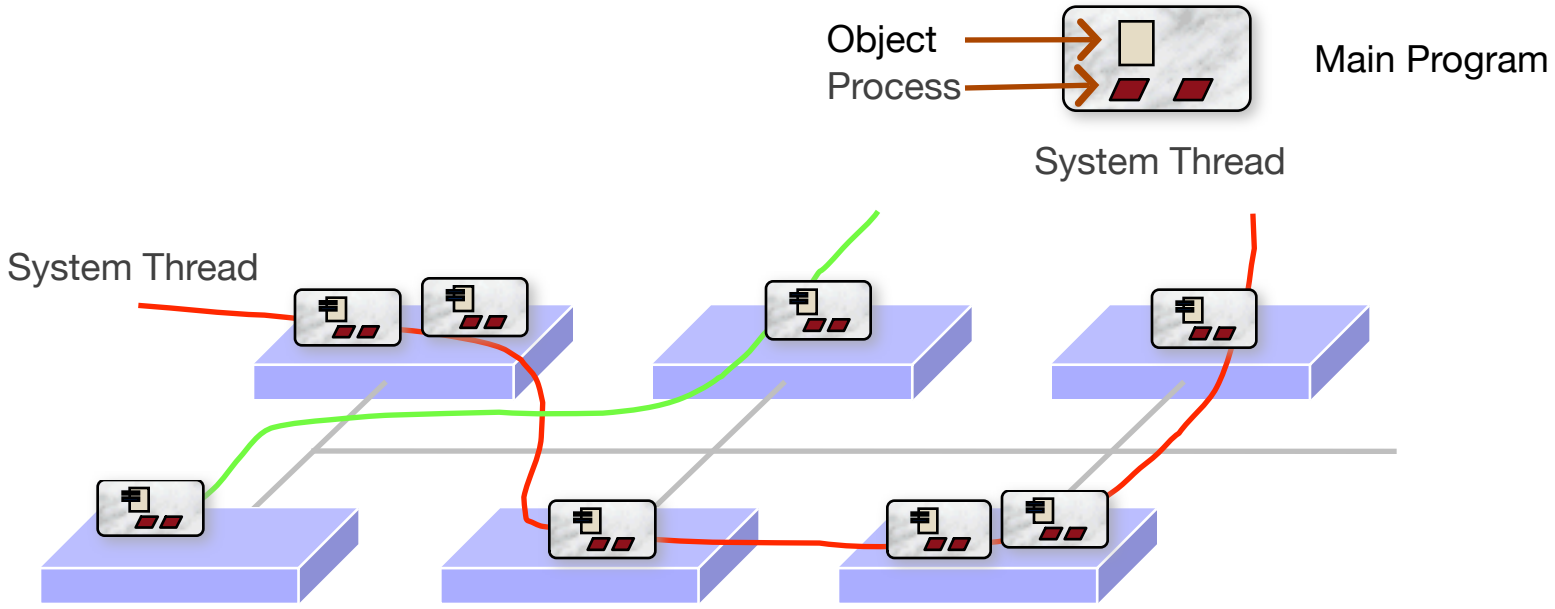
Multiprocessors in avionics systems





Application is designed as a set of interacting software objects (hundreds)

Multiprocessing platforms for large systems



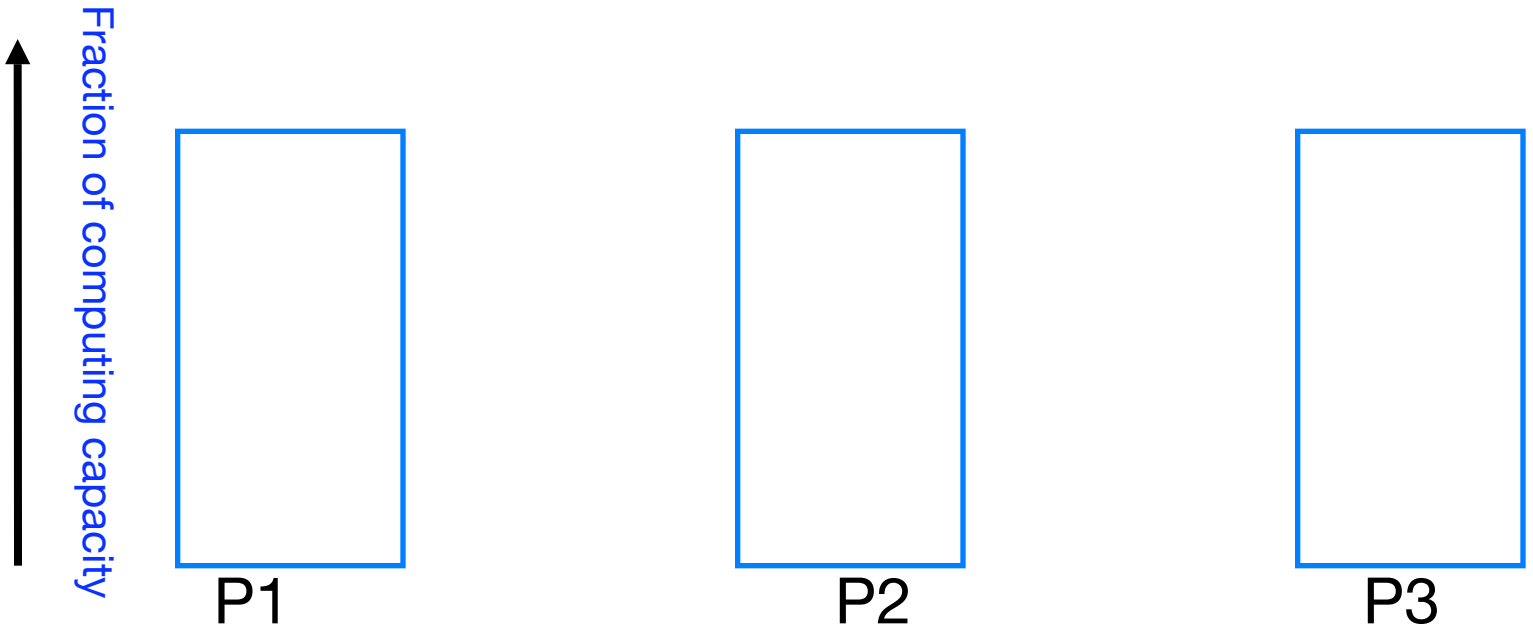
The advantages of multiprocessor systems

- Greater computational power (obviously!)
- Power savings
  - More slower processors when compared to a few fast (power-hungry) processors
  - Easier heat dissipation
- Reliability
  - Backups for critical tasks
  - Migrations when some processors fail
- Security and isolation
  - Critical tasks can be separated from non-critical tasks

Models of multiprocessor systems

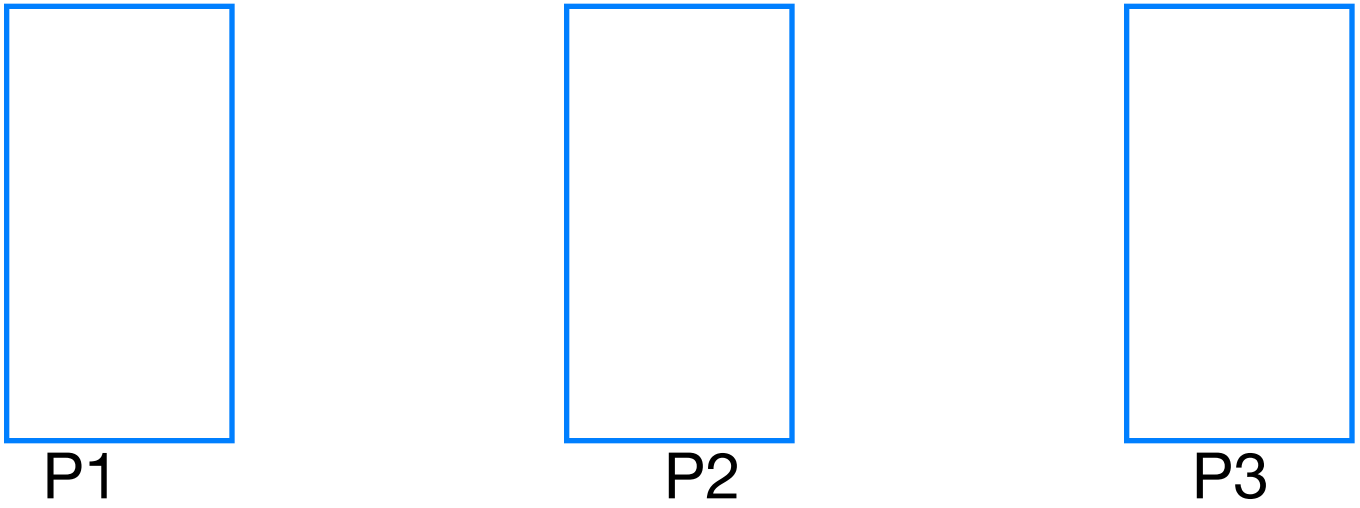
- Identical multiprocessors
  - Each processor has the same computing capacity
- Uniform multiprocessors
  - Different processors have different computing capacities
  - The faster a processor is, the lower the execution time of a task
- Heterogeneous multiprocessors
  - Each (task, processor) pair may have a different computational attribute
  - Execution times of a task may vary from processor to processor but there is no well-defined relationship

Multiprocessor models



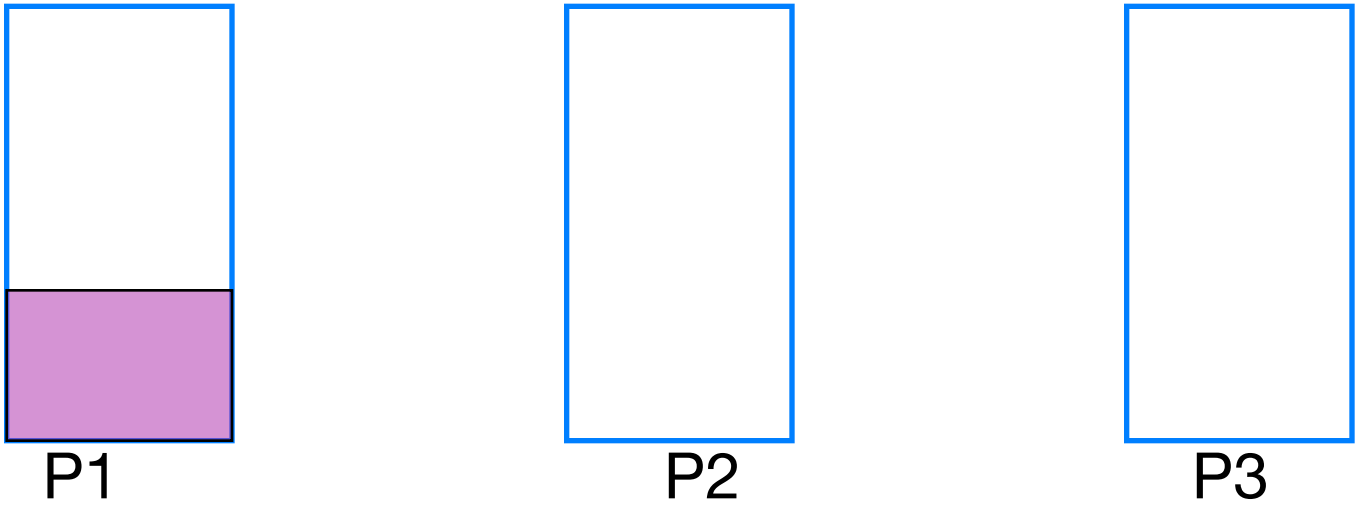
Identical multiprocessors

Task T1



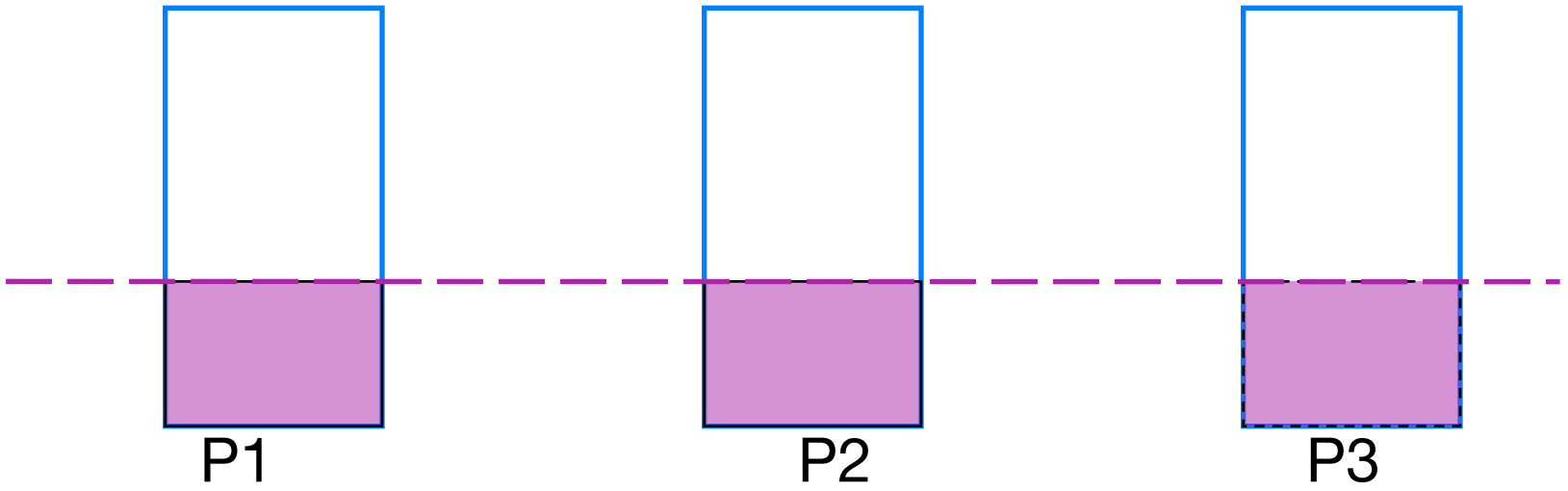
Identical multiprocessors

Task T1



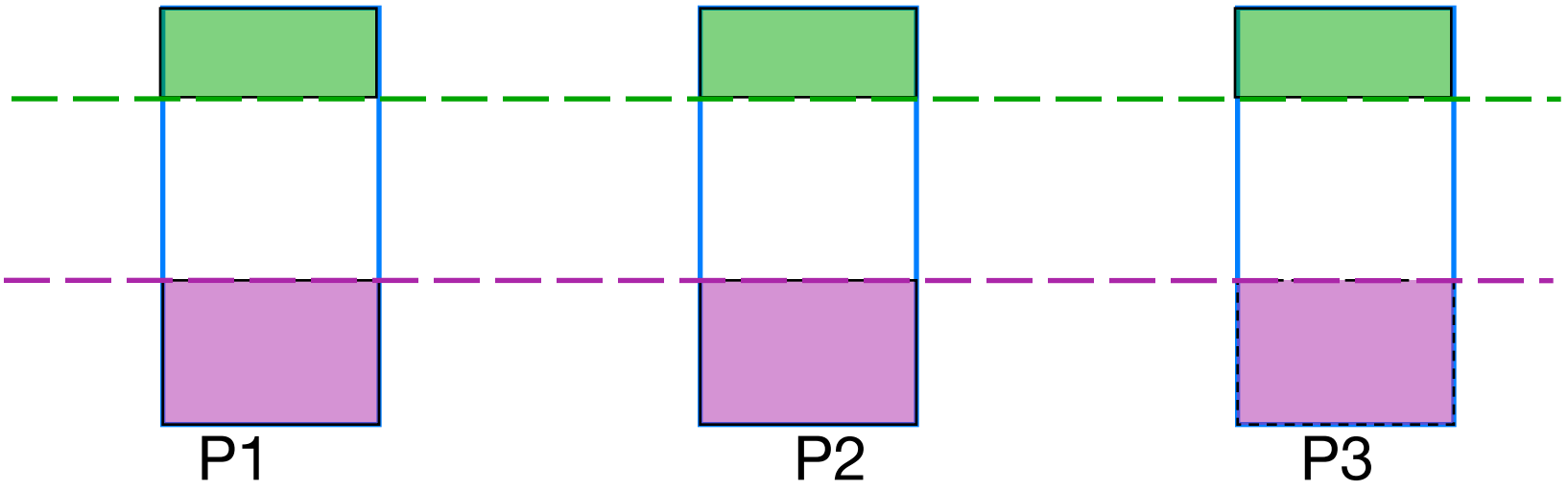
Identical multiprocessors

Task T1



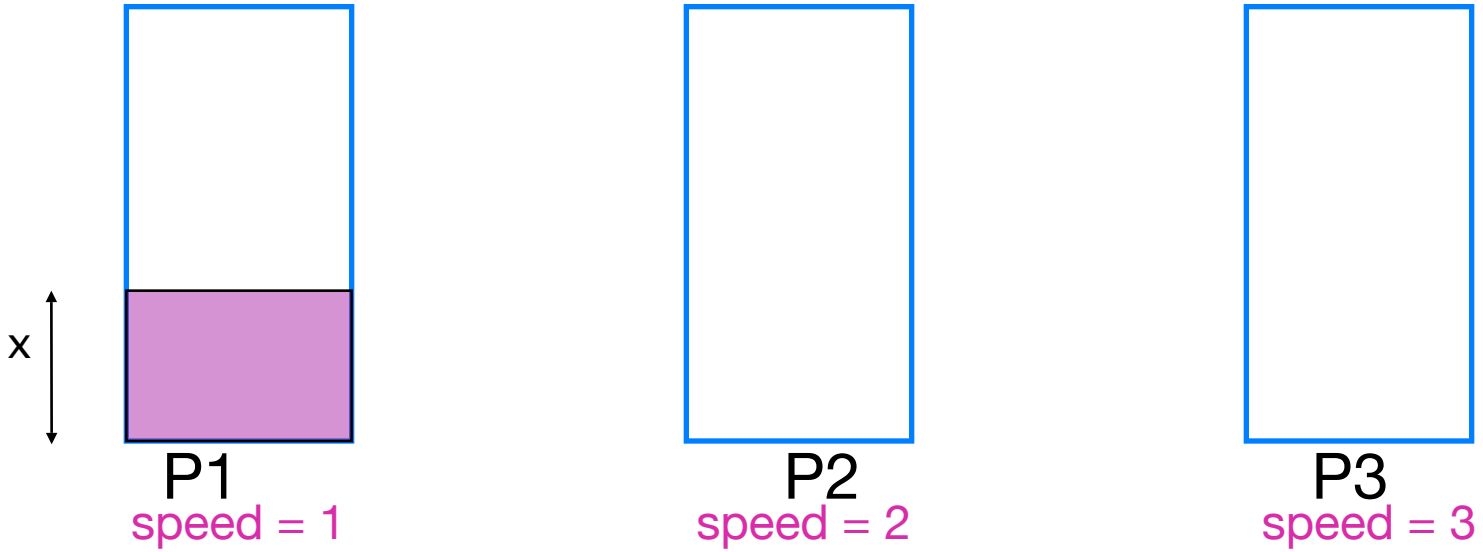
Identical multiprocessors

Task T1 Task T2



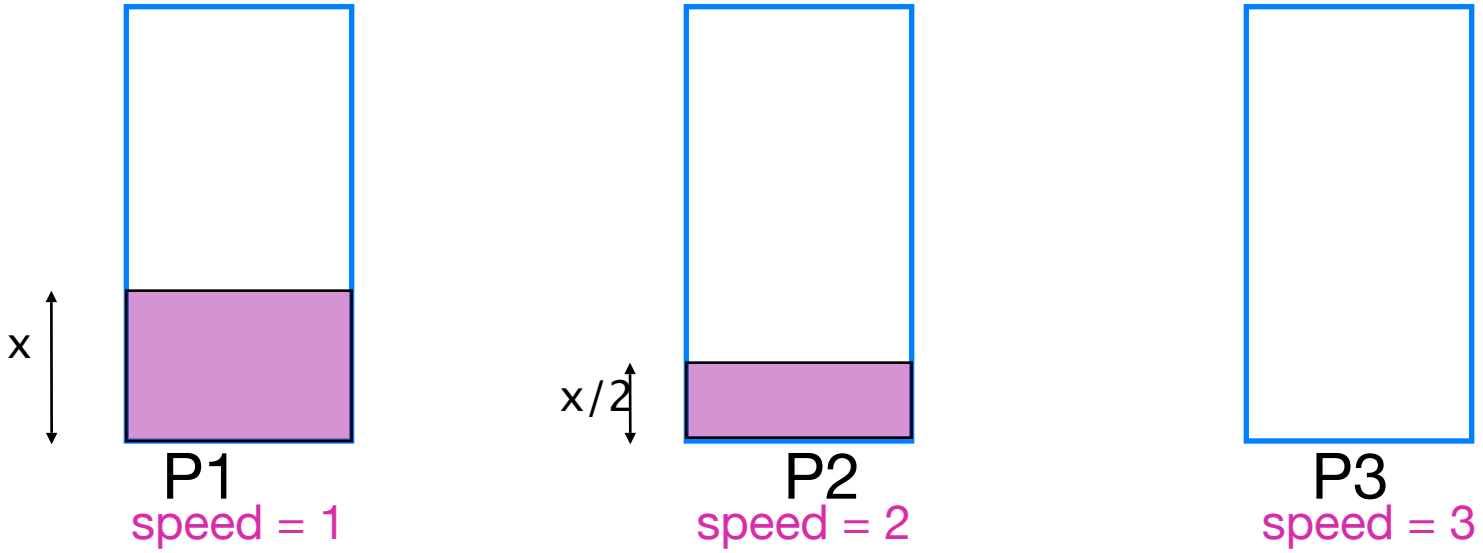
Uniform multiprocessors

Task T1



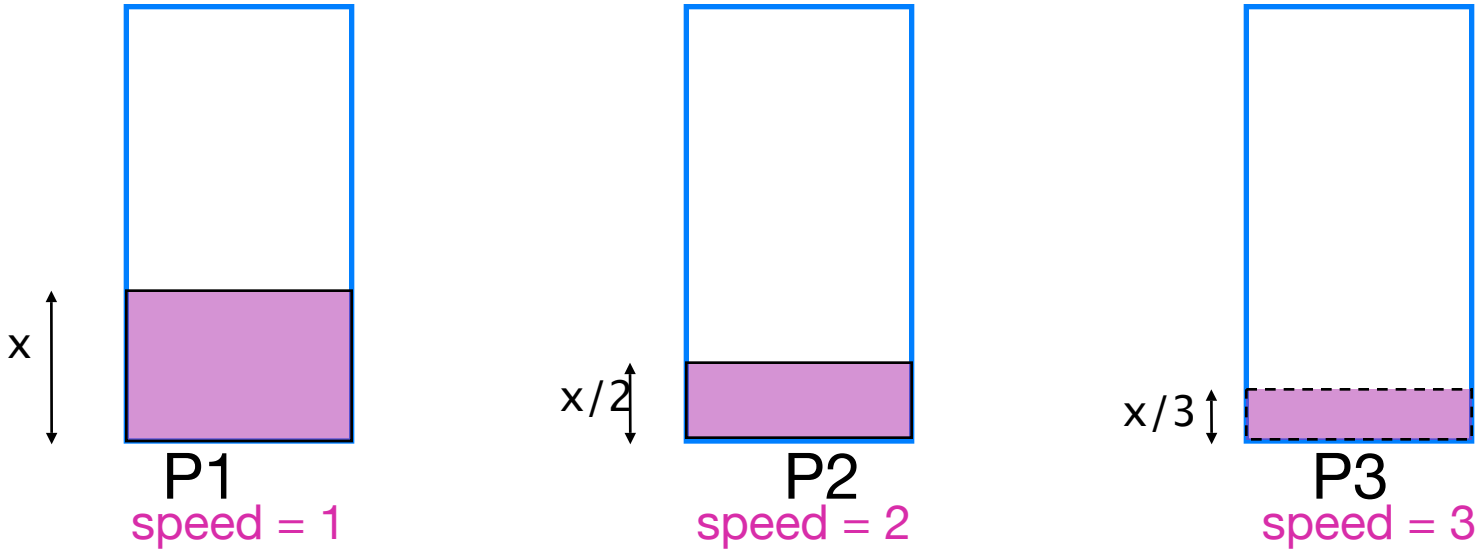
Uniform multiprocessors

Task T1



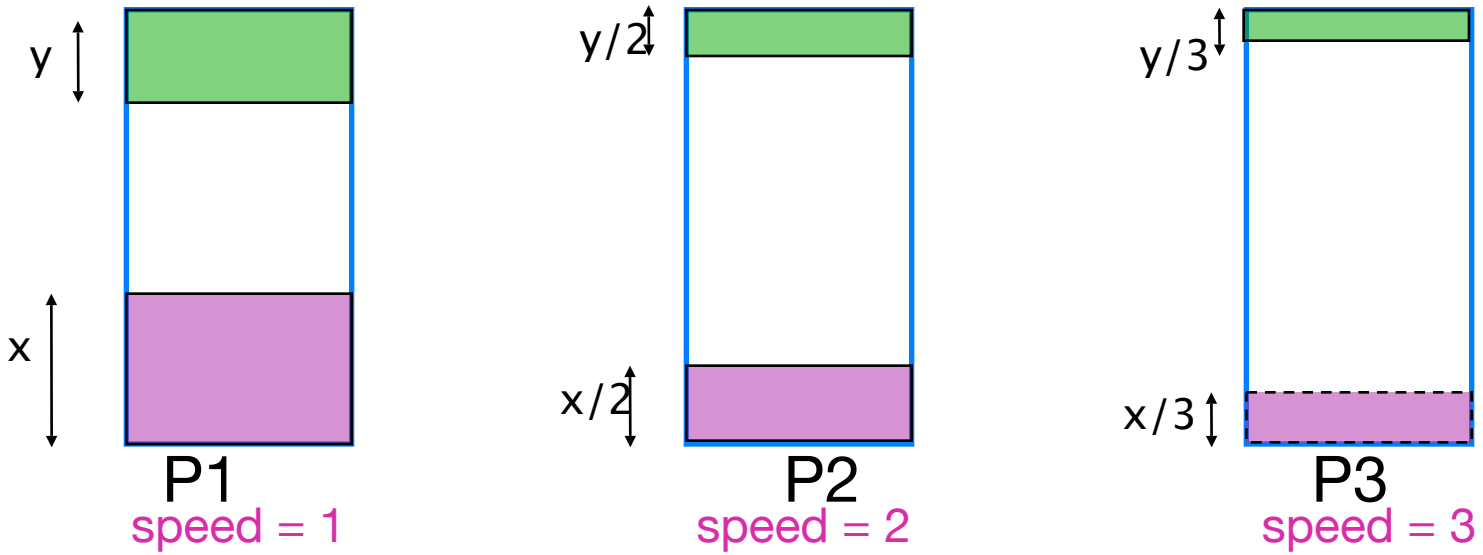
Uniform multiprocessors

Task T1



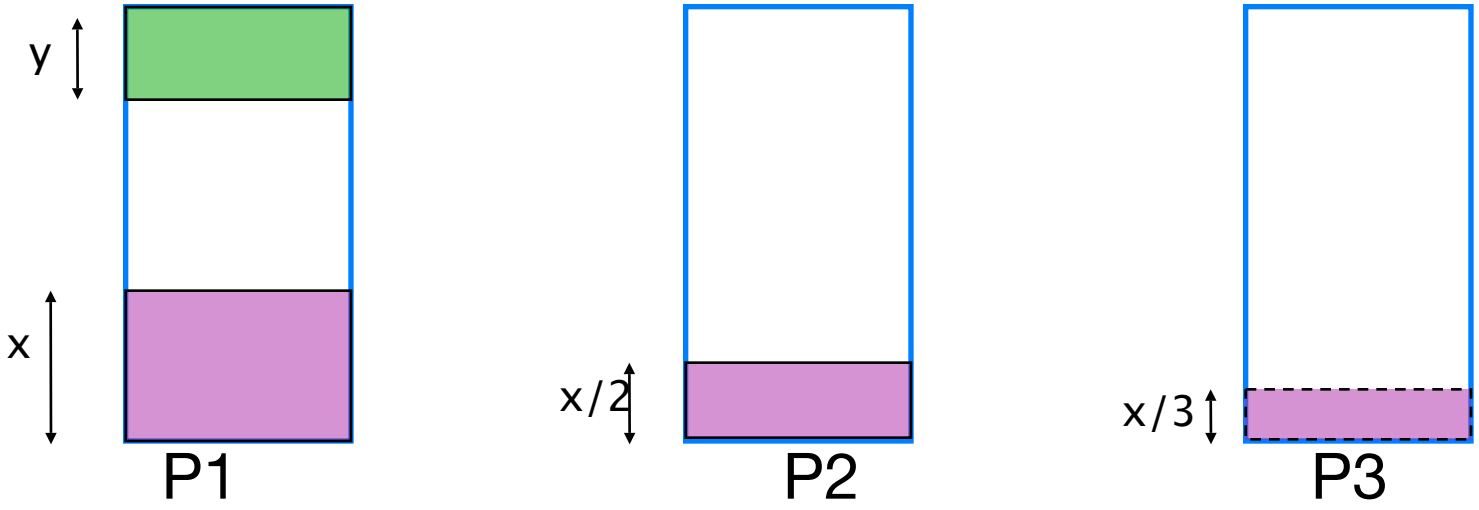
Uniform multiprocessors

Task T1 Task T2



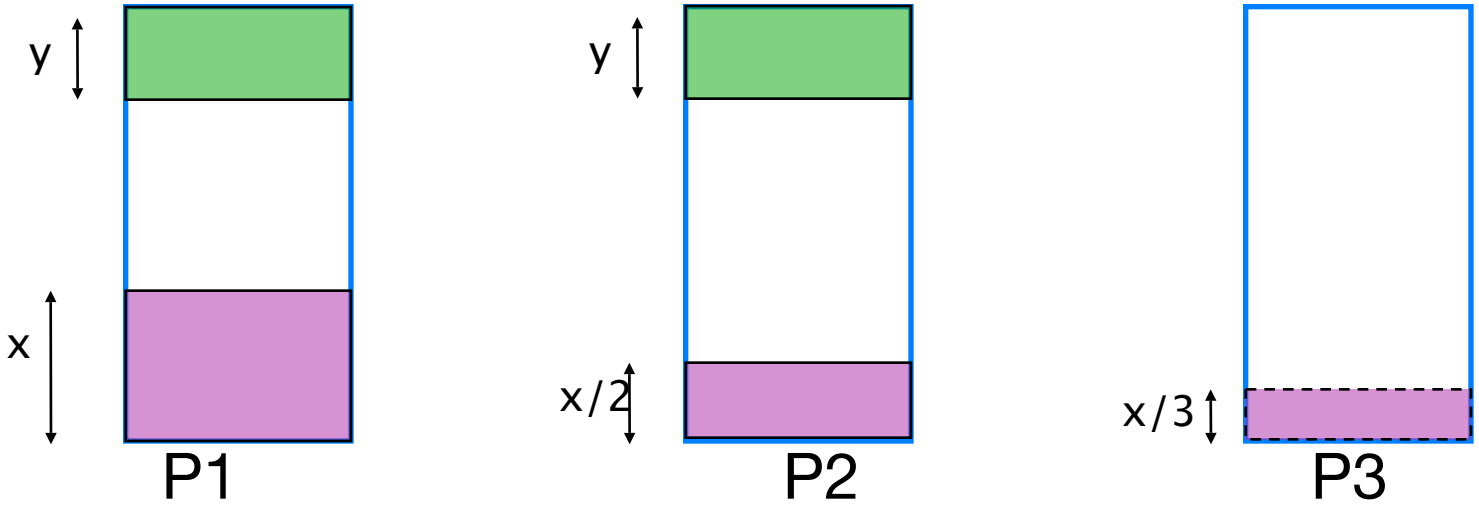
Heterogeneous multiprocessors

Task T1 Task T2



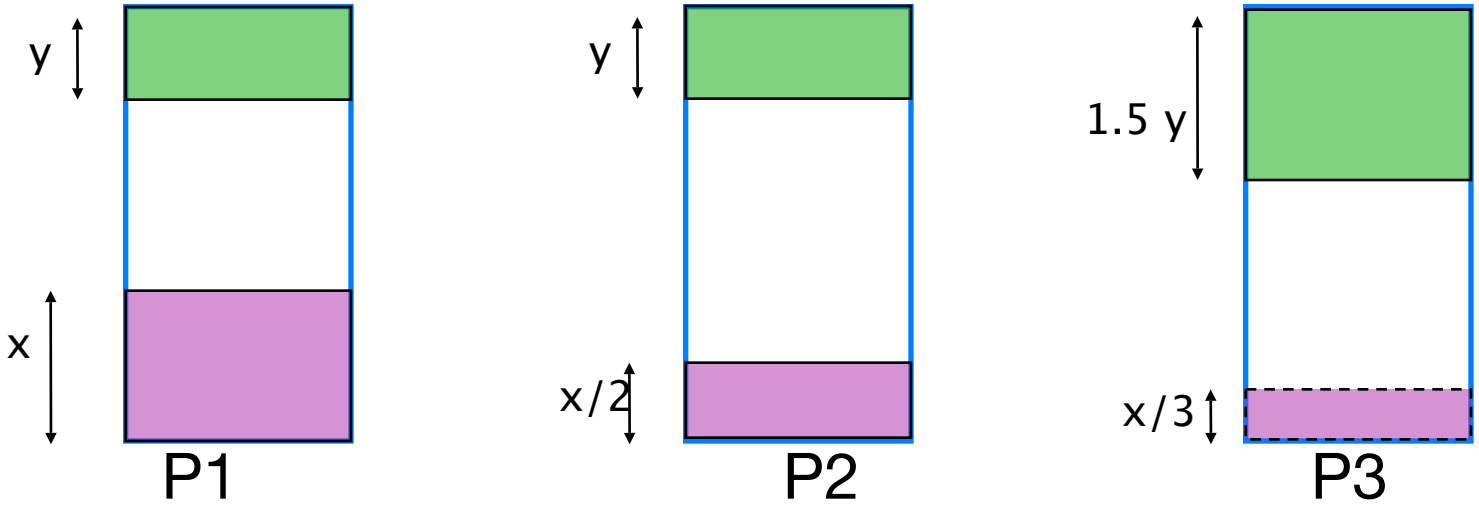
Heterogeneous multiprocessors

Task T1 Task T2

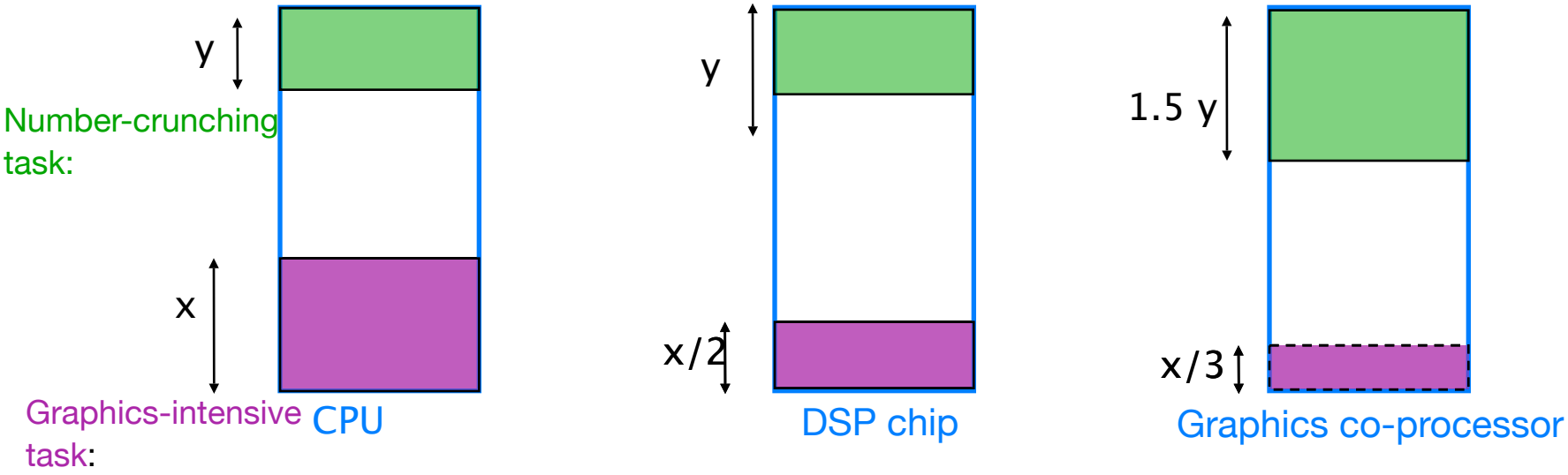


Heterogeneous multiprocessors

Task T1 Task T2



Heterogeneous multiprocessors



Resource management for real-time systems

- Given a multiprocessing platform and a set of recurring tasks with deadlines, can the tasks be scheduled to meet their deadlines on the platform?

- Standard recurring task model
  - Tasks $\{T_i\}$
  - Periodic tasks with periods $\{P_i\}$
  - Execution times of the tasks $\{e_i\}$
  - Known deadlines

Classes of scheduling policies

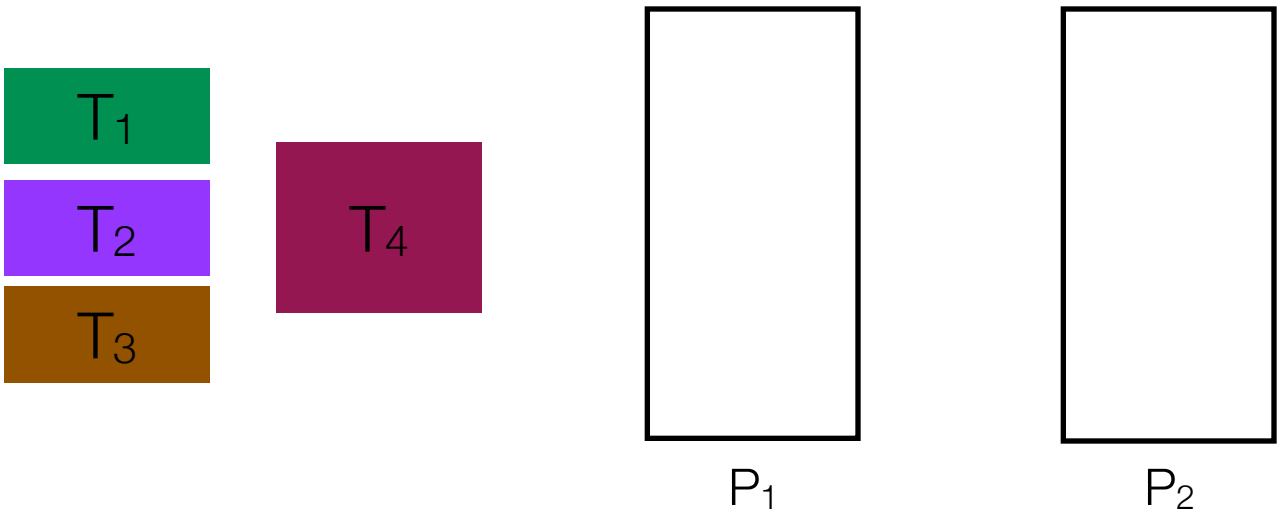
- Apart from the known classes (static and dynamic priorities), multiprocessing introduces further options
- Partitioned scheduling
  - Each task may execute only on one processor
  - No migration of jobs
- Global scheduling
  - Any instance of any task may execute on any processor
  - Jobs can migrate between processors

Partitioned scheduling vs. global scheduling

- Partitioned scheduling is easier to implement and reason about
  - Once tasks are assigned to a processor, we can apply known schedulability tests
  - Without migration it is easier to maintain context information
  - When processors are on different chips migration requires context transfer, cache problems, etc.
- Global scheduling, however, is more flexible
  - Allowing migration improves schedulability
  - On-chip multiprocessing minimizes some of the overhead of job migration

Partitioned scheduling

- We can use either fixed priority (rate monotonic) or dynamic priority (EDF) policies
- Need to assign tasks to processors such that the utilization bound (or other schedulability condition) is satisfied
  - For simplicity we will assume that any task can be allocated to any processor
  - This may not always be the case because of resource requirements and so on

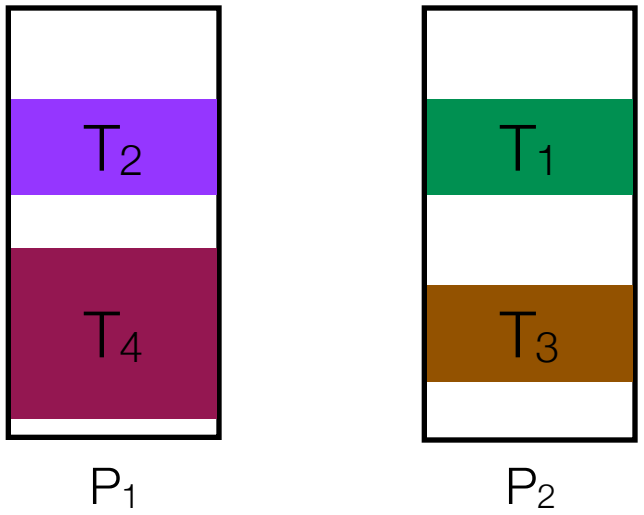


Partitioned scheduling

- We can use either fixed priority (rate monotonic) or dynamic priority (EDF) policies
- Need to assign tasks to processors such that the utilization bound (or other schedulability condition) is satisfied
  - For simplicity we will assume that any task can be allocated to any processor
  - This may not always be the case because of resource requirements and so on

Any assignment of tasks to processors is suitable as long as utilization bounds are not violated.

Is closely related to the **bin packing** problem, which is NP-Hard.



Partitioned scheduling

- Because allocating tasks to processors is NP-Hard, optimal allocation may be extremely hard -- especially if we want to allocate tasks to processors at run-time
  - Even when the processors are uniform
- We need heuristics for task allocation
  - A popular heuristic is first-fit decreasing (FFD)
  - FFD on uniform multiprocessors
    - Sort tasks by utilization and order the processors (any order is okay)
    - Starting with the task with highest utilization, assign tasks to the first possible processor

Utilization bounds for partitioned scheduling with EDF

- When tasks are allocated to processors using the FFD heuristic, we can derive a utilization upper-bound for a uniform multiprocessor system that guarantees schedulability
- Let *m* be the number of processors: then the maximum possible utilization of the system is *m* (each processor can have a utilization up to 1)

$$U_b = \frac{m+1}{2}$$

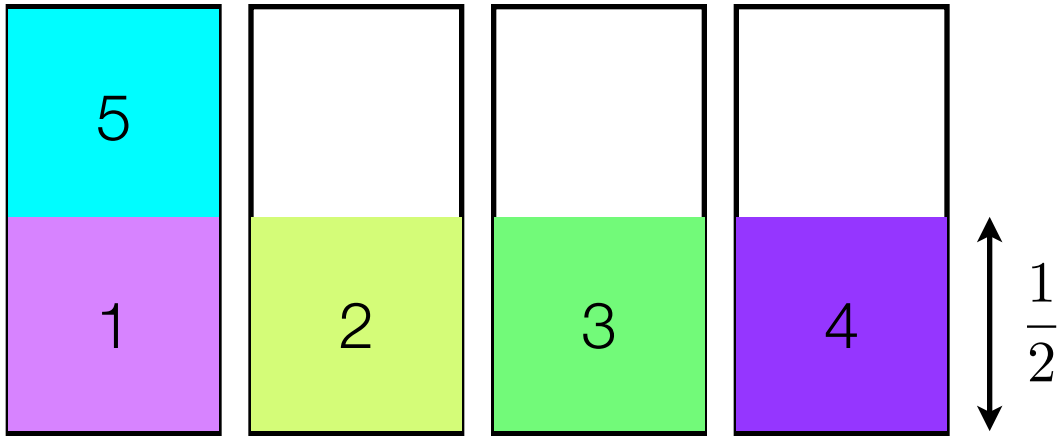
- Proof sketch
  - Consider a task set with m+1 tasks, each task having utilization 1/2; this task set is schedulable
  - If each task has utilization slightly greater than 1/2, the task set is not schedulable

Utilization bounds for partitioned scheduling with EDF

- When tasks are allocated to processors using the FFD heuristic, we can derive a utilization upper-bound for a uniform multiprocessor system that guarantees schedulability
- This is a sufficient condition but not necessary

$$U_b = \frac{m+1}{2}$$

Schedulable

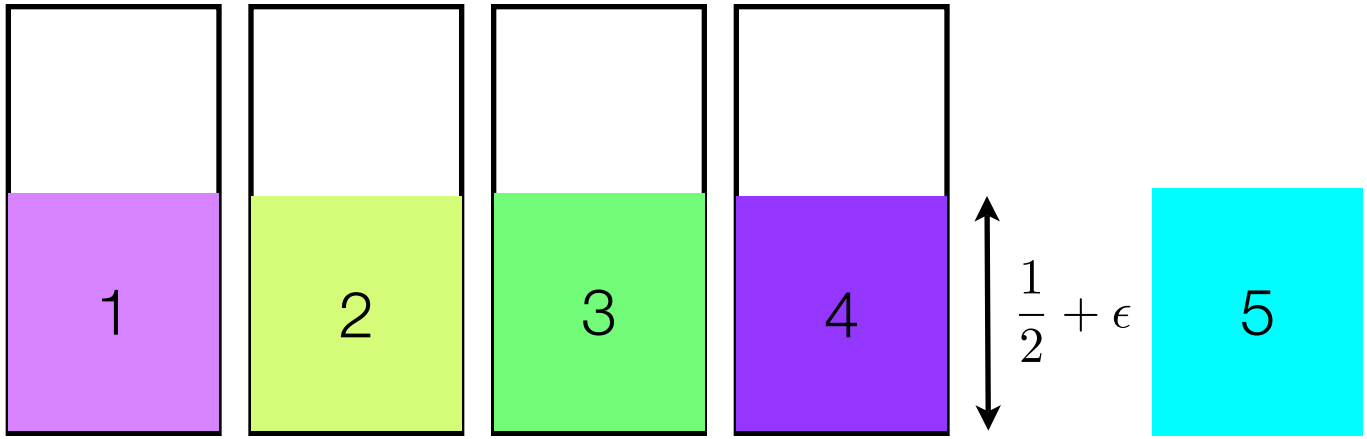


Utilization bounds for partitioned scheduling with EDF

- When tasks are allocated to processors using the FFD heuristic, we can derive a utilization upper-bound for a uniform multiprocessor system that guarantees schedulability
- This is a sufficient condition but not necessary

$$U_b = \frac{m+1}{2}$$

Not schedulable



Utilization bounds for partitioned scheduling with EDF

- When tasks are allocated to processors using the FFD heuristic, we can derive a utilization upper-bound for a uniform multiprocessor system that guarantees schedulability
- This is a sufficient condition but not necessary

$$U_b = \frac{m+1}{2}$$

- The utilization can be rather poor even though we have many processors
- The overall utilization is as low as 50%!

Utilization bounds for partitioned scheduling

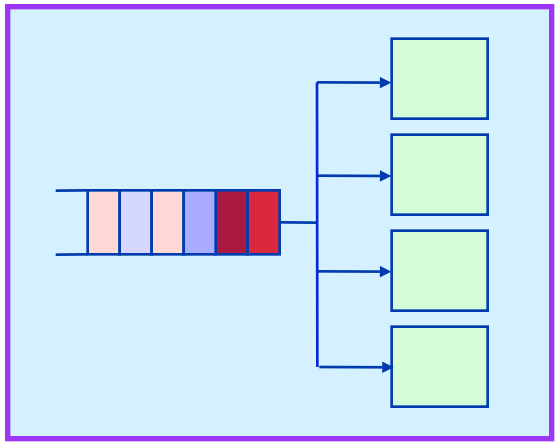
- The utilization bound can be improved if we know that the maximum individual task utilization is U_{max}

$$\frac{m\beta + 1}{\beta + 1},$$
$$\text{where } \beta = \lfloor \frac{1}{U_{max}} \rfloor$$

- With some extra information, the utilization bound is significantly better
- For rate monotonic scheduling of tasks on a multiprocessor, one approach is to assume that the utilization of any individual processor should be less than 69%
- We can do better but we will not deal with those results now
- Given the allocation scheme (such as FFD) it is not hard to verify schedulability

Global scheduling

- Is an alternative to partitioning
- Tasks may migrate among processors
- Appropriate for tightly-coupled systems



Global scheduling

- It is much harder to derive schedulability conditions for global scheduling on multiprocessors
- Still a very active research problem
- A sample result [Srinivasan & Baruah]: If the utilization of each task is less than $m/(2m-1)$ where m is the number of processors then $U \leq m^2/(2m-1)$ is a sufficient (not necessary) condition for schedulability

Multiprocessor scheduling

- There are more results concerning response times for scheduling on multiprocessors, but we will not discuss them in this course

- Fault tolerance in a multiprocessor system
  - We can tolerate a limited number of processor failures by replicating a task on more than one processor
  - Specific task allocation policies can be developed taking this constraint into account
  - We will discuss fault tolerance and reliability issues in greater detail in the next few lectures

Highlights

- This slide set provides a brief overview of multiprocessor platforms and real-time scheduling
  - Benefits of multiprocessor platforms
  - Models of multiprocessor systems
    - Identical processors, uniform processors, heterogeneous processors
  - Scheduling models
    - Partitioned scheduling, global scheduling
  - Utilization bounds
    - Specifically for partitioned scheduling
    - Tasks allocated to processors using the FFD heuristic