Multiprocessor Scheduling - I

Raj Rajkumar

Lecture #13

Mid-Term Grades

- Quizzes 1&2 (25% each)
- Lab #1 (50%)

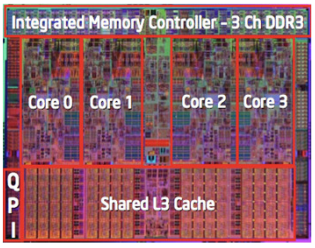
- Max: 100
- Min: 60
- Mean: 84
- Median: 85
- Std. Dev: 10.6

Outline

- Multi-core Trends
- Multiprocessor Scheduling Approaches
- Global Scheduling
- Anomalies in multiprocessor scheduling

System Trends

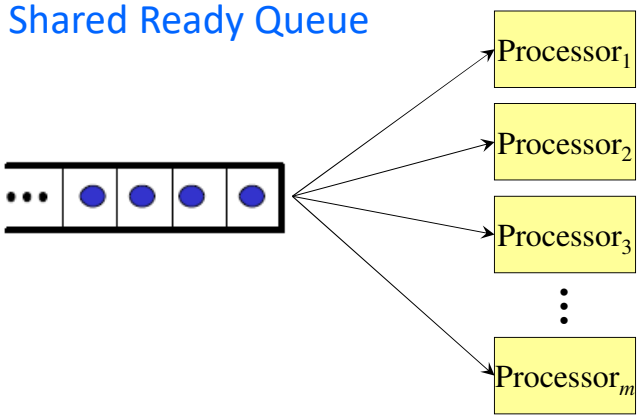
- **Processor trends**
  - Intel Xeon processor E5-2600 offers up to 22 cores
  - Intel Knights Landing: 72 cores!
  - Trend - numerous cores / chip
- Major driving forces
  - Cooling constraints
  - Fundamental limits on clock speeds



How do we schedule real-time tasks on multicore processors (or multiprocessors)?

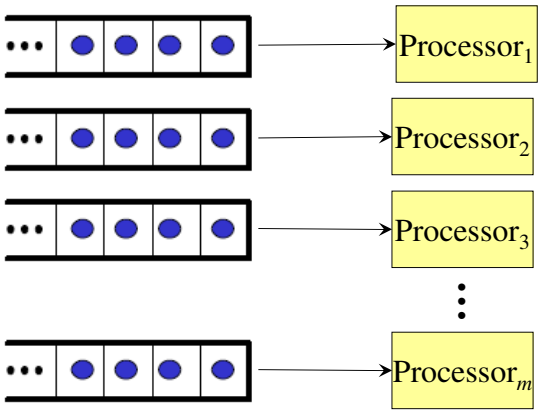
Global Scheduling for Multiprocessors

Single Shared Ready Queue



Partitioned Scheduling for Multiprocessors

One Ready Queue Per Processor



Global Multiprocessor Scheduling Characteristics

- All ready tasks are kept in a common (global) queue
- When selected for execution, a task can be dispatched to an arbitrary processor, even after being preempted
- Task execution is assumed to be "greedy":
  - If higher-priority tasks occupy all processors, a lower-priority task cannot execute until the execution of a higher-priority task is complete.

Global Scheduling

Advantages:

- Supported by most multiprocessor operating systems
  - Windows NT, Solaris, Linux, ...
- Effective utilization of processing resources
  - Unused processor time can easily be reclaimed

Disadvantages:

- Weak theoretical framework
  - Few results from the uni-processor case can be used
- Poor resource utilization for hard timing constraints
  - No more than 50% resource utilization can be guaranteed
- Suffers from several scheduling anomalies
  - Sensitive to period adjustments

Global Scheduling

Complexity of schedulability analysis for global scheduling: (Leung & Whitehead, 1982)

- *The problem of deciding if a task set is schedulable on m processors with respect to global scheduling is NP-complete in the strong sense.*

Consequence:

- There can only exist a pseudo-polynomial time algorithm for
  - (i) finding an optimal static priority assignment, or
  - (ii) feasibility testing
- But not both at the same time (unless P = NP)!

Global Scheduling

- The root-cause in global scheduling: (Liu, 1969)

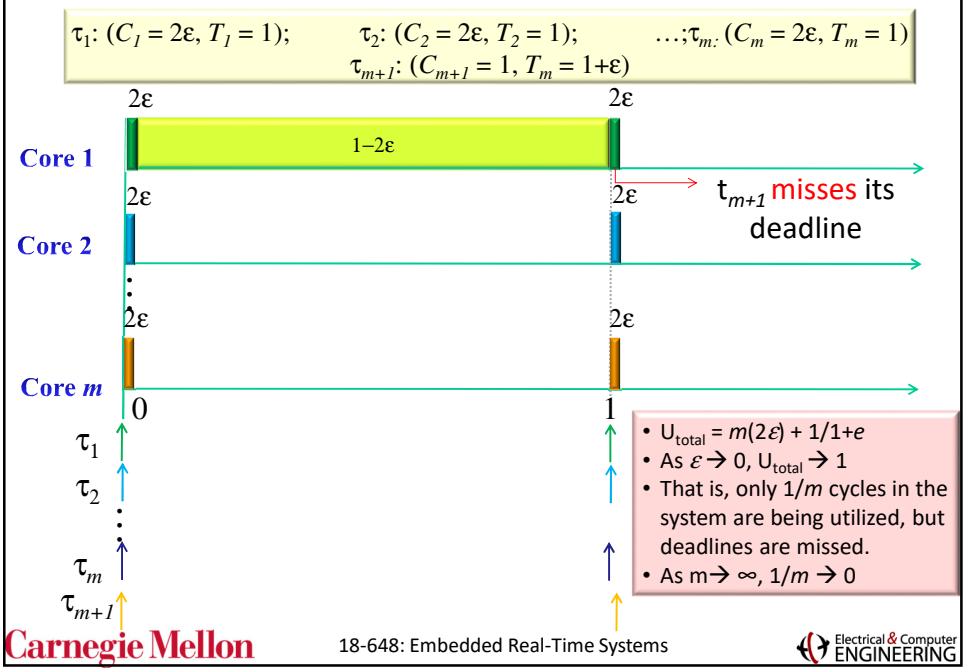
  Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.

Weak Theoretical Framework

Underlying causes:

- Dhall's effect:
  - With RM, DM and EDF, some low-utilization task sets can be unschedulable regardless of how many processors are used.
- Dependence on Relative Priority Ordering:
  - Changing the relative priority ordering among higher-priority tasks may affect the schedulability for a lower-priority task.
- Uncharacterized Critical Instant:
  - A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks.

Dhall Effect (Dhall & Liu, 1978)



Dhall's Effect (2 of 2)

- Also applies to (greedy) RM, DM and EDF scheduling
- The lowest utilization of unschedulable task sets can be ar :ter how many processo

$$U_{global} = m\frac{2\varepsilon}{1} + \frac{1}{1+\varepsilon} \rightarrow 1 \quad \text{when } \varepsilon \rightarrow 0$$

Consequence:

- New multiprocessor priority-assignment schemes are needed!

Impact of Relative Priority Ordering

- The response time of a task depends on the relative priority ordering of the higher-priority tasks
- This property does not exist for a uniprocessor system
- This means that well-known uniprocessor methods for finding optimal priority assignments cannot be applied

Consequence:

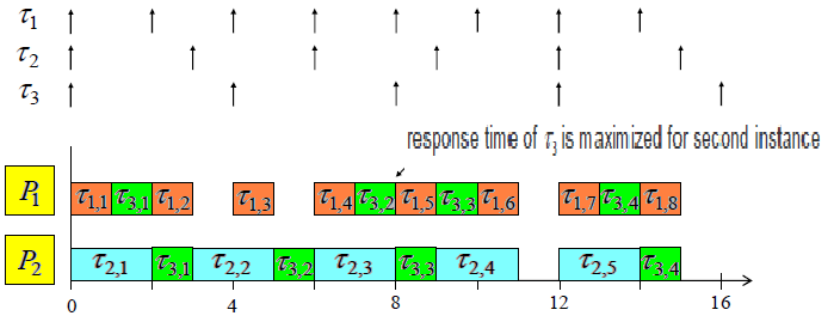
- New methods for constructing multiprocessor priority assignments are needed!

Uncharacterized Critical Instant

- Critical instant, the relative phasing at which a task has its maximum response time, cannot be easily characterized.

RM scheduling

$$\tau_1 = \{C_1 = 1, T_1 = 2\}$$
$$\tau_2 = \{C_2 = 2, T_2 = 3\}$$
$$\tau_3 = \{C_3 = 2, T_3 = 4\}$$



More on Critical Instant

- Critical Instant:
  - A critical instant does *not* always occur when a task arrives at the same time as all its higher-priority tasks.
  - Finding the critical instant is a very (NP-?) hard problem
  - **Note**: recall that knowledge about the critical instant is a fundamental property in uniprocessor feasibility tests.

Consequence:

- New methods for constructing effective multiprocessor feasibility tests are needed!

Underlying Causes

- Dhall's Effect:
  - With RM, DM and EDF, some low-utilization task sets can be unschedulable regardless of how many processors are used.
- Dependence on relative priority ordering:
  - Changing the relative priority ordering among higher-priority tasks may affect schedulability for a lower-priority task.
- Uncharacterized critical instant:
  - A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks.

New techniques for priority assignments and schedulability tests are needed!

Poor Resource Utilization

- The utilization guarantee bound for any static-priority multiprocessor scheduling algorithm cannot be higher than ½ of the capacity of the processors.
- This applies for all types of static-priority scheduling. That is, partitioned and global, greedy and p-fair scheduling.
- Hence, in the worst case, one cannot utilize more than half the processing capacity if hard timing constraints exist.

Scheduling Anomalies

Scheduling anomaly: A seemingly positive change in the system (reducing load or adding resources) causes a non-intuitive decrease in performance.

- Uniprocessor systems:
  - Anomalies only found for non-preemptive scheduling (Mok, 2000)
- Multiprocessor systems:
  - Richard's anomalies for non-preemptive scheduling
  - Execution-time-based anomalies for preemptive scheduling
  - Period-based anomalies for preemptive scheduling

Richard's Anomalies: (Graham, 1969)

- Assumptions:
  - Non-preemptive scheduling
  - Precedence constraints
  - Restricted migration (individual task instances cannot migrate)
  - Fixed execution times
- Task completion times may increase as a result of:
  - Changing the task priorities
  - Increasing the number of processors
  - Reducing task execution times
  - Weakening the precedence constraints
  - Having shared resources

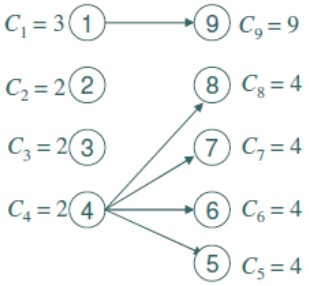
Execution-time-based anomalies: (Ha & Liu, 1994)

- Assumptions:
  - Preemptive scheduling
  - Independent tasks
  - Restricted migration (individual task instances cannot migrate)
  - Fixed execution times
- Task completion times may increase as a result of:
  - Reducing task execution times

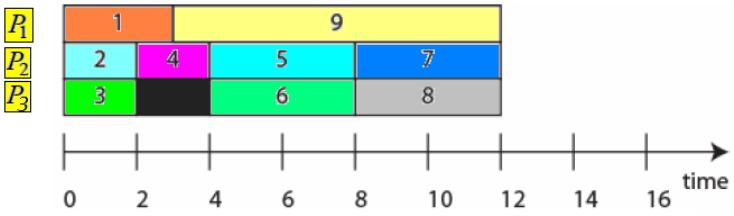
Period-based anomalies (Andersson & Jonsson, 2000)

- **Assumptions**:
  - Preemptive scheduling
  - Independent tasks
  - Full migration
  - Fixed execution times
- A task's completion time may increase as a result of:
  - Increasing the period of a higher-priority task
  - Increasing the period of the task itself
- **Note**: increasing the periods is commonly used to reduce the load in real-time systems!

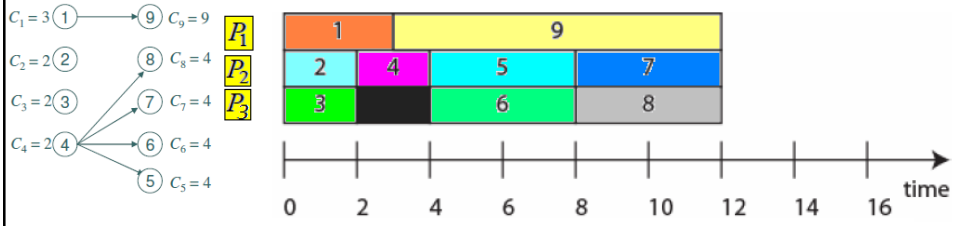
Precedence Constraints



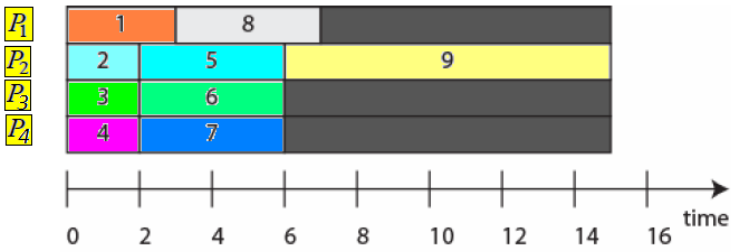
An edge in the ***precedence graph*** from job *i* to job *j* means that job *j* can execute only after job *i* has completed execution.



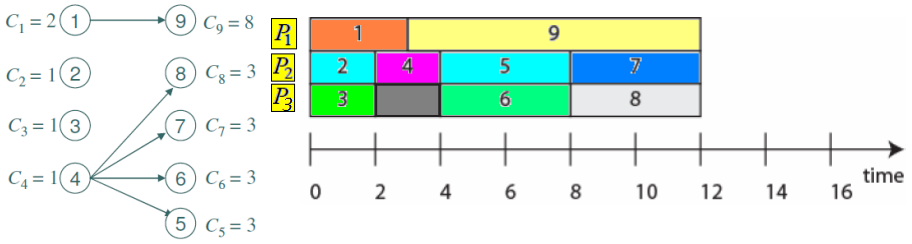
Anomaly with # of Processors



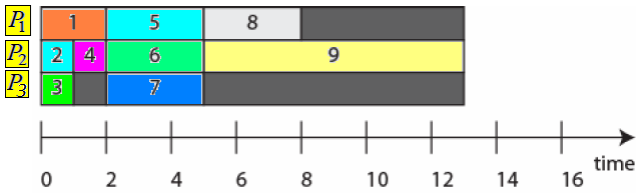
What happens if one more processor is introduced?



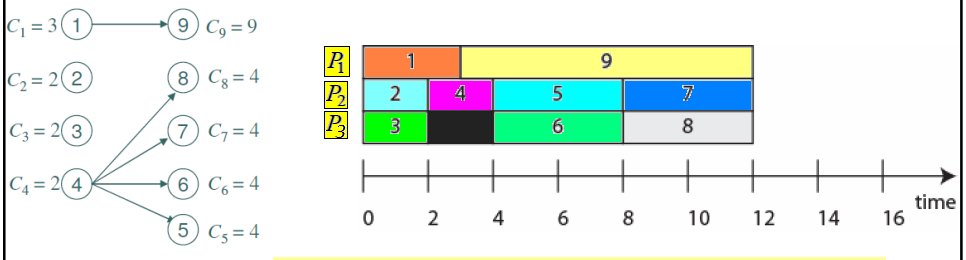
Anomaly with Reduction of Computation Time



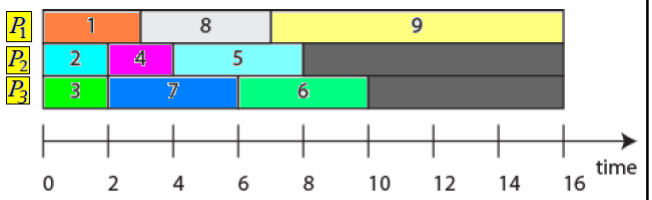
What happens if all computation times are reduced by 1?



Anomaly with Relaxation of Precedence Constraints

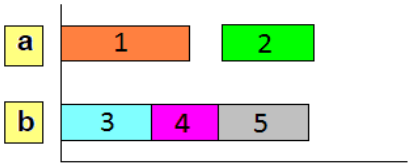


What happens if you remove the precedence constraints (4 → 8) and (4 → 7)?

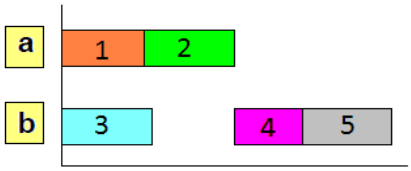


Anomaly with Resource Sharing

Jobs 2 and 4 share a resource: i.e. must execute mutually exclusively



What happens if you reduce C_1?



Global Scheduling

Some Older Basic Results in global scheduling:

- **Static priorities**:
  - The RM-US[m/(3m-2)] priority assignment scheme offers a way to circumvent Dhall's effect and a non-zero resource utilization guarantee bound of m/(3m-2) ≥ 33.3%.
- In 2003, Baker generalized the RM-US results to DM.
- **Dynamic priorities**:
  - In 2002, Srinivasan & Baruah proposed the EDF-US[m/(2m-1)] scheme with a corresponding non-zero resource utilization guarantee bound of m/(2m-1) ≥ 50%.
- Optimal multiprocessor scheduling:
  - Using p-fair scheduling and dynamic priorities, it is possible to achieve 100% resource utilization on a multiprocessor.
  - Impractical to use?

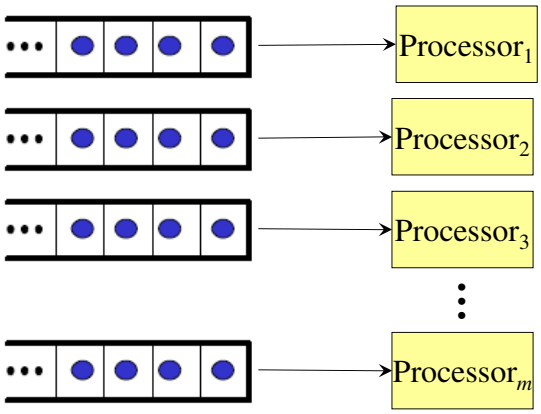
Conclusions

- Multi-core systems are a part of the future
- Global and partitioned scheduling techniques are two approaches to multiprocessor scheduling
- Timing behavior under task scheduling strategies can be brittle.
  - Small changes can have big (and unexpected) consequences.

PARTITIONED SCHEDULING

Partitioned Scheduling for Multiprocessors

One Ready Queue Per Processor



Partitioned Scheduling

- Given a set of tasks, how to assign each task to a queue (processor)?
- Notes:
  - Once assigned, a task only executes on the processor it is assigned to
  - Single-processor schedulability analysis can be carried out on each node
    - i.e. look at all the tasks that can be in the same ready queue
  - No anomalies to worry about
  - No Dhall's Effect
- Question:

The answer lies in the so-called Bin-Packing problem.

Bin-Packing

- The Bin-Packing Problem
  - *n* objects, each of size ≤ 1.0, must be packed into *m* identical "bins", each of which is of size 1.0 such that
    a. the sum of the objects allocated to any bin is ≤ 1.0
    b. the number of bins used is minimized.

Bin-Packing Heuristics

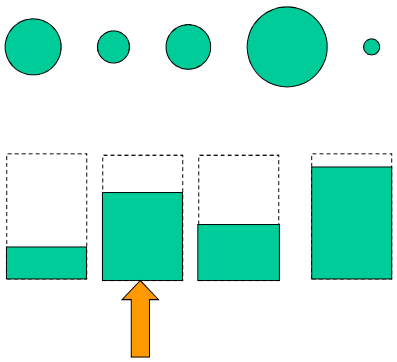
- First-Fit (FF)
- Best-Fit (BF)
- Next-Fit (NF)
- Worst-Fit (WF)
- First-Fit Decreasing (FFD)
- Best-Fit Decreasing (BFD)
- Worst-Fit Decreasing (WFD)

Questions to Ask:

- How many bins do you need?
  - What is the minimum number of bins you need?
  - What is the maximum number of bins you need?
- Can you find the optimum number of bins?

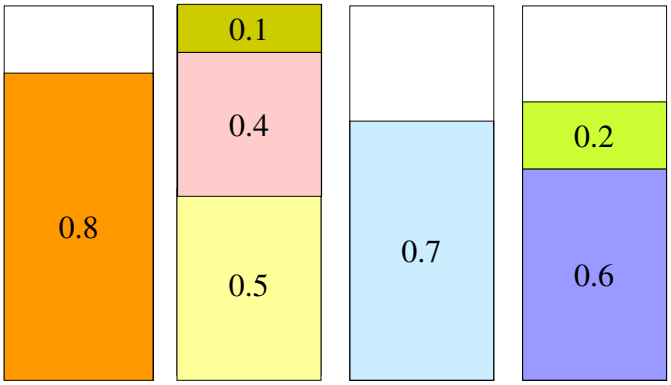
Next-Fit Heuristic

1. Fit next object into next bin that it can fit into.
   - If it does not fit into any bin, add a new bin to fit into.
2. If no object left, done. Go to Step 1.



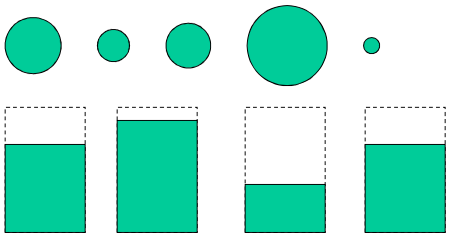
Next Fit Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



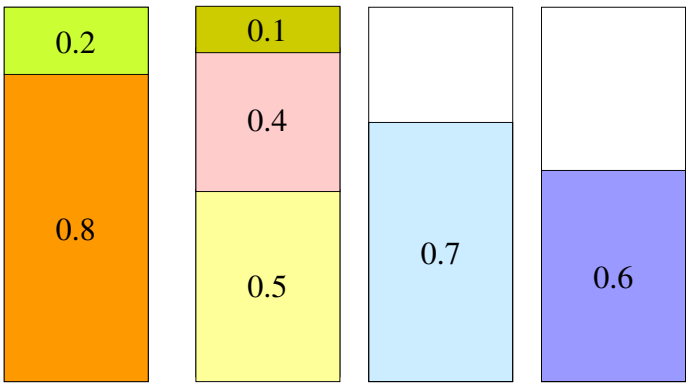
First-Fit (FF) Heuristic

1. Fit next object into the first bin that it can fit into
   - If it does not fit into any bin, add a new bin
2. If no object left, done. Else, go to Step 1.



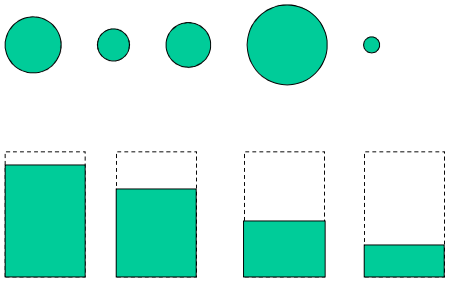
First Fit Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



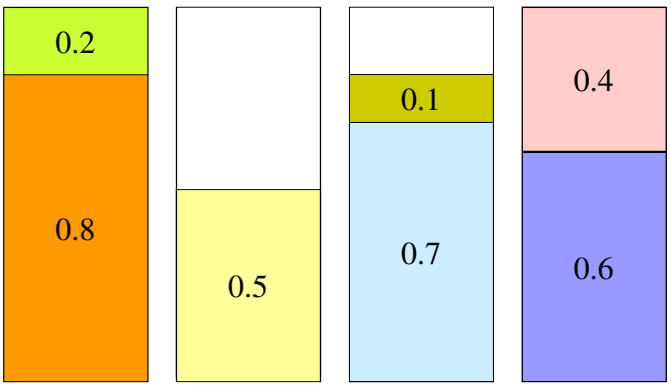
Best-Fit Heuristic

1. Sort the bins in descending order of consumed space
2. Fit next object into first sorted bin that it can fit into
   - If it does not fit into any bin, add a new bin to fit into
3. If no object left, done. Go to Step 1.



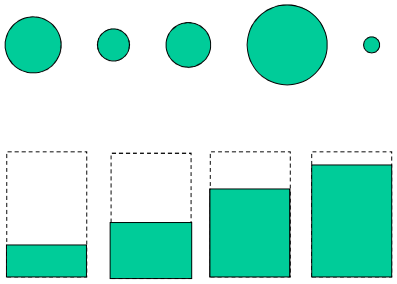
Best Fit Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



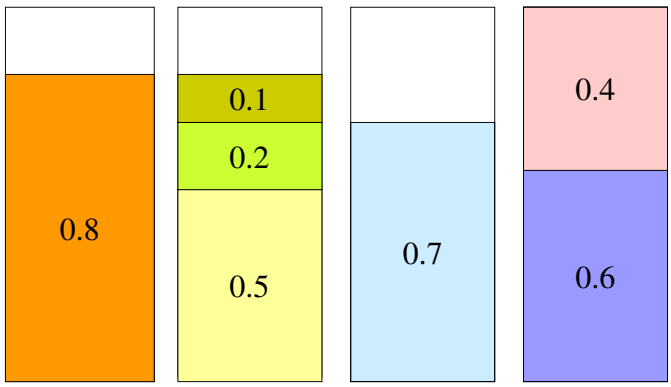
Worst-Fit Heuristic

1. Sort the bins in ascending order of consumed space
2. Fit next object into first sorted bin that it can fit into
   - If it does not fit into any bin, add a new bin to fit into
3. If no object left, done. Go to Step 1.



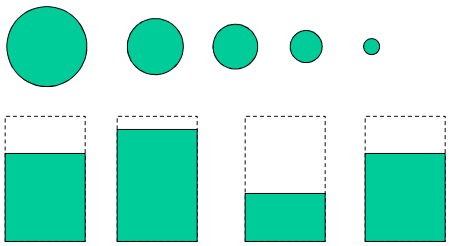
Worst Fit Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



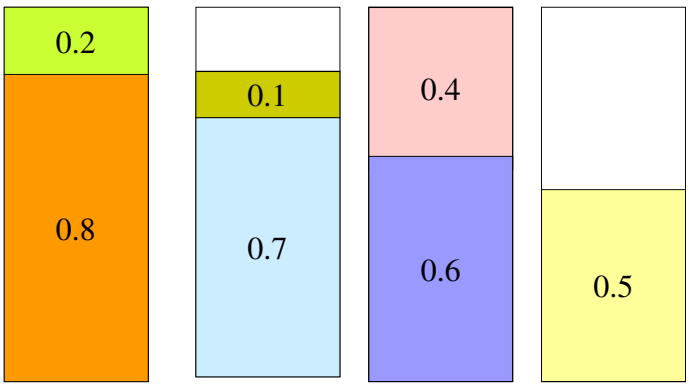
First-Fit Decreasing (FFD) Heuristic

1. Sort the objects in descending order of size
2. Fit first object into the first bin that it can fit into
   - If it does not fit into any bin, add a new bin
3. If no object left, done. Else, go to Step 2.



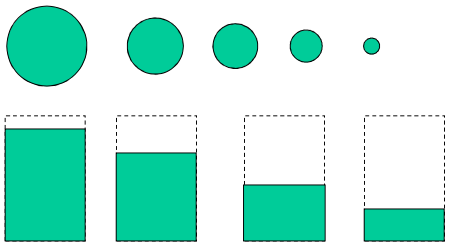
First Fit Decreasing Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



Best-Fit Decreasing (BFD) Heuristic

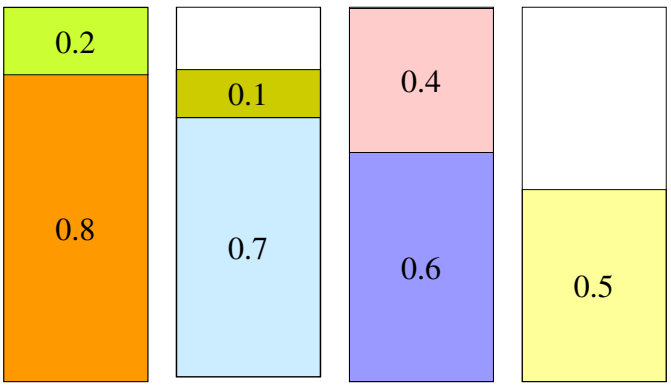
1. Sort the objects in descending order of size
2. Sort the bins in descending order of consumed space
3. Fit next object into first sorted bin that it can fit into
   - If it does not fit into any bin, add a new bin to fit into
4. If no object left, done. Go to Step 2.



"Tries to pack bins as much as possible with empty space in the 'last' bins"

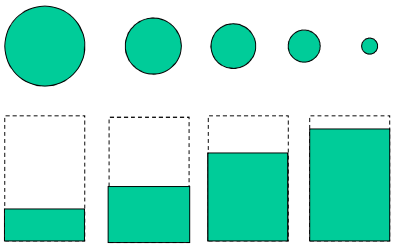
Best Fit Decreasing Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



Worst-Fit Decreasing (WFD) Heuristic

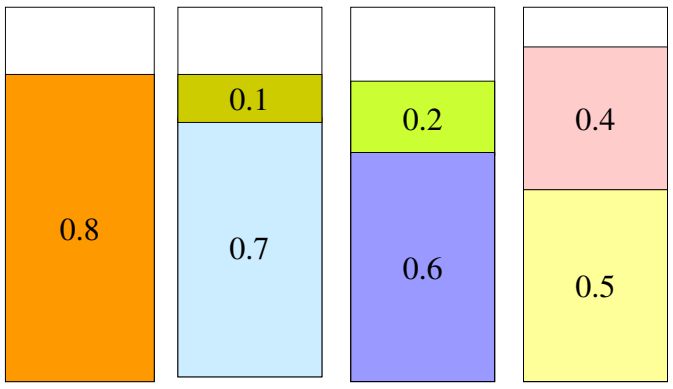
1. Sort the objects in descending order of size
2. Sort the bins in ascending order of consumed space
3. Fit next object into first sorted bin that it can fit into
   - If it does not fit into first bin, add a new bin to fit into
4. If no object left, done. Go to Step 2



"Balances load across available bins"

Worst Fit Decreasing Example

- Given a list of objects of size 0.8, 0.5, 0.7, 0.6, 0.2, 0.4, 0.1 in that order
- Pack these objects into bins of capacity 1.0



Complexity of Bin-Packing

- The bin-packing problem is known to be NP-complete:
  - It could take an exponential # of steps in the worst case to determine the optimal number of bins
- Very efficient near-optimal heuristics exist
  - *Best Fit Decreasing* and *First Fit Decreasing* heuristics use no more than **11/9 OPT + 2 bins** (where **OPT** is the number of bins given by the optimal solution).
    - Exception: there are simple cases which have worse performance, but can be detected and addressed as special cases.

The Bad Case of Bin Packing

- Given a list of objects of size 0.51, 0.51, 0.51, 0.51, 0.51
- Pack these objects into bins of capacity 1.0
- How many bins are required?
- With objects of size 0.50+ε, 0.50+ε, 0.50+ε, 0.50+ε, 0.50+ε,
  - What is the utilization of the system?
- Large objects can be a problem!
  - Think *rocks*…

The Good Case of Bin-Packing

- Consider a list of objects of size 0.01, 0.01, 0.01, 0.01, …. (1000 objects)
- Pack these objects into bins of size 1.0
- How many bins are required?
- Are small objects a problem?
  - Think sand…

Getting Back to Partitioned Scheduling

- Only having tasks with high utilization can lead to poor utilization of processors
  - We refer to these tasks as "heavyweight tasks"
- Only having tasks with low utilization leads to extremely good packing and high system utilization
  - We refer to these tasks as "lightweight tasks"
- What does having a hybrid of "lighweight" and "heavyweight" tasks mean??

Important Points to Remember

- When we perform bin-packing for partitioned scheduling, consider which scheduling policy is used
  - Under EDF, 100% schedulability is possible → bin-packing condition is straightforward (modulo overheads and worst-case execution time estimates).
  - Under RM, schedulability analysis must be carried out (exact or otherwise) → "empty space" test is less straightforward.
- All tasks are assumed to be independent, i.e. there is no resource sharing
  - In practice, this assumption is generally not true.

Conclusions

- Partitioned scheduling for multiprocessors requires the bin-packing problem to be solved
- Bin-packing is an NP-complete problem
- Many efficient but sub-optimal heuristics are available
  - First fit, next fit, best fit and worst fit
    - Usually applied to pack objects as they arrive
  - First-fit decreasing, next-fit decreasing, best-fit decreasing and worst-fit decreasing
    - Usually applied when all objects to be packed are readily available (and can be sorted before packing)