Implementation of Real-Time Scheduling Algorithms

Luca Abeni
luca.abeni@unitn.it

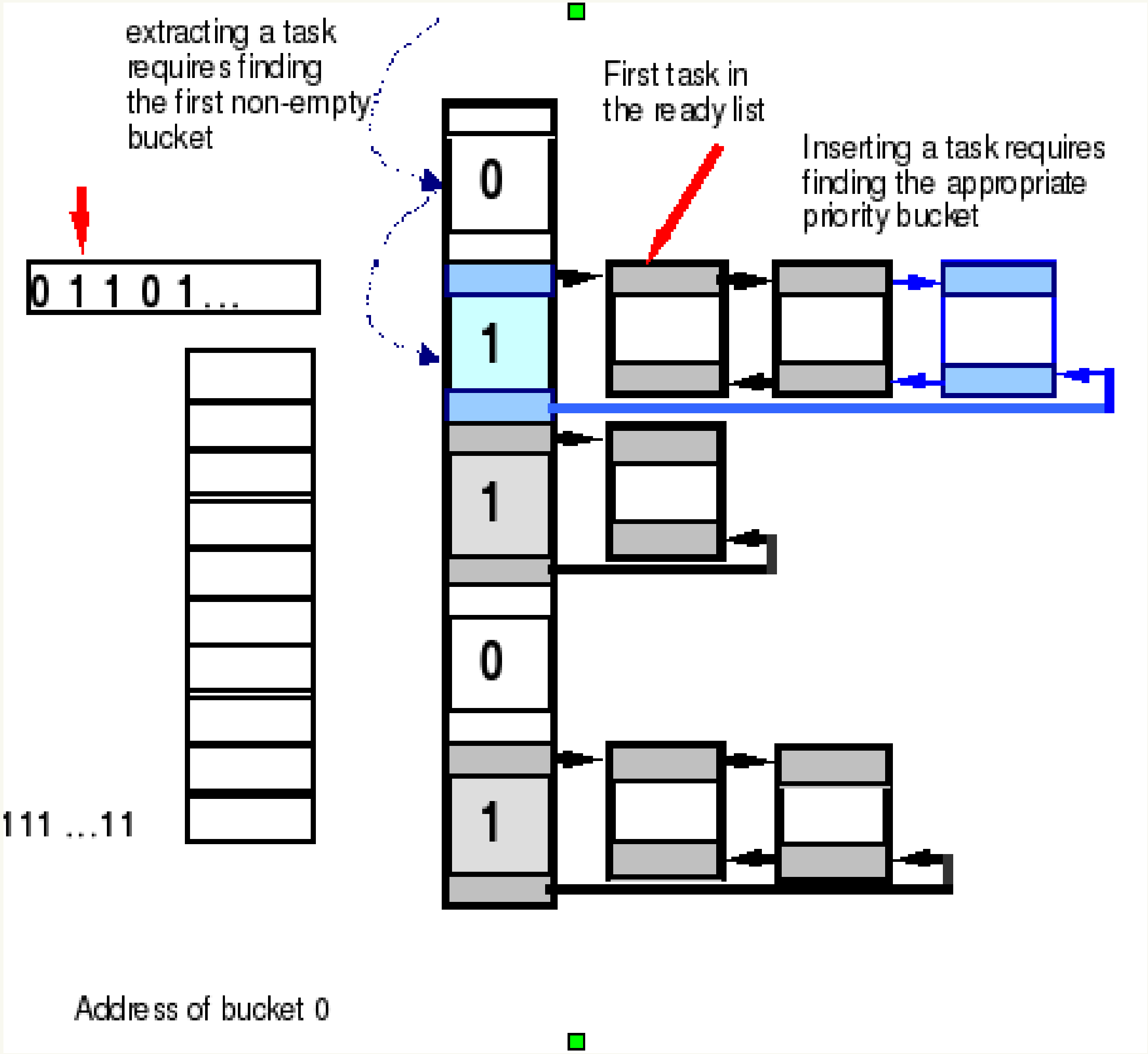
Implementation of Fixed Priorities

- Fixed priority schedulers can be implemented with an array of queues (one per priority level)
- Insertion into the queue (task wake-up) $\rightarrow O(1)$ operation
- Extraction of the highest priority task from the queue (scheduling decision)
  - Find the highest priority non-empty queue
  - $O(n)$ search!!! Too much overhead!!!
- Overhead due to naive implementation, not to an inherent problem

More Efficient Implementation

- The scheduler scalability can be improved by using a bitmap
  - Array of bits to mark the queues that are non-empty
- The highest priority queue can be found by finding the most significant bit in a word
  - Extraction becomes $O(1)$ if there is an Assembly instruction that returns the first 1 bit in a word (CLZ)
  - If not, table to implement the operation $\lceil \log w \rceil$

Implementation of fixed priority - I



Implementing EDF

- EDF queueing is more complex
  - Dynamic priorities $\rightarrow$ cannot use the “bitmask trick”
  - No $O(1)$ complexity
- Can EDF be implemented with something better than $O(n)$ complexity?
- Yes we can!
  - But an appropriate data structure is needed!
- Which data structure? Which are the requirements?

EDF Queues: Requirements

- Data structure storing ordered keys, with efficient:
  - Ordered Insertion (task wake-up)
  - Selection of the first entry (scheduling)
  - extraction of the first entry (dispatching)
- Efficient removal of non-first entries is not important
  - Why removing non-executing tasks from the queue?
- Efficient search of specific entries is not important
  - Why should we need this??

Red/Black Trees

- The deadline queue can be implemented using a Red/Black tree
  - Self-balancing tree, based on nodes colouring
  - $O(log(n))$ on all the operations
- Not too bad, if n is not too large!
- $\Rightarrow$ Red/Black trees make EDF implementable in practice (without too much overhead)!