Hashed and Hierarchical Timing Wheels

A paper by

George Varghese and *Tony Lauck*

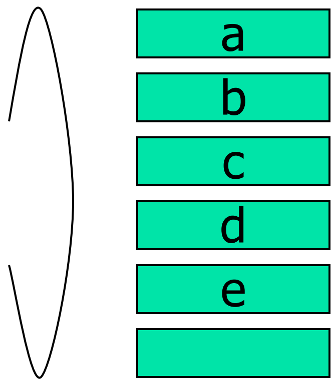
Motivation

- Timers are important for
  - Failure recovery, rate based flow control, scheduling algorithms, controlling packet lifetime in networks
- Timer maintenance high if
  - Processor interrupted every clock tick
  - Fine granularity timers are used
  - # outstanding timers is high
- Efficient timer algorithms are required to reduce the overall interrupt overhead

Model & Performance Measure

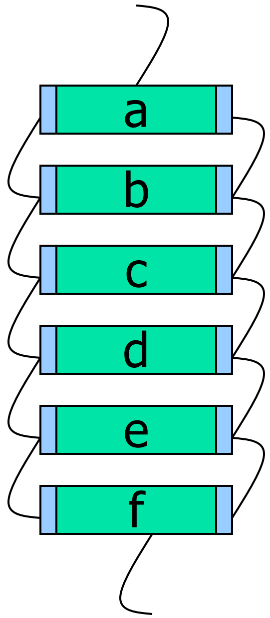
- Routines in the model
  - Client Invoked :
    - START_TIMER(Interval, Request_ID, Expiry_Action)
    - STOP_TIMER(Request_ID)
  - Timer tick invoked :
    - PER_TICK_BOOKKEEPING
    - EXPIRY_PROCESSING
- Performance Measure
  - Space : Memory used by the data structures
  - Latency : Time required to begin and end any of the routines mentioned above

Currently Used Timer Schemes



Can maintain absolute expiry time or the timer interval

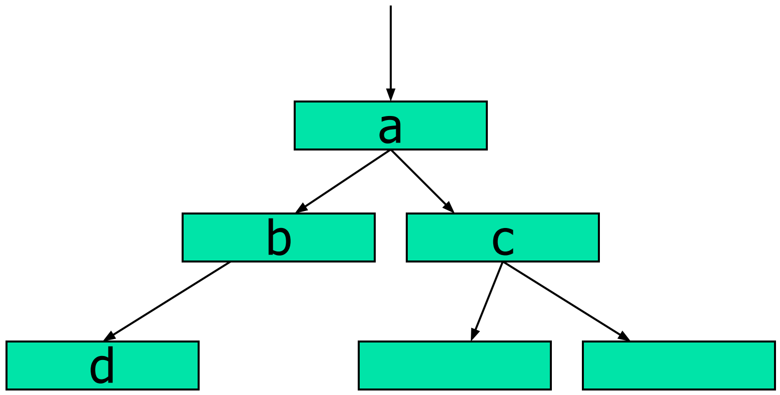
START_TIMER = O(1)
STOP_TIMER = O(1)
PER_TICK_BOOKKEEPING = O(n)

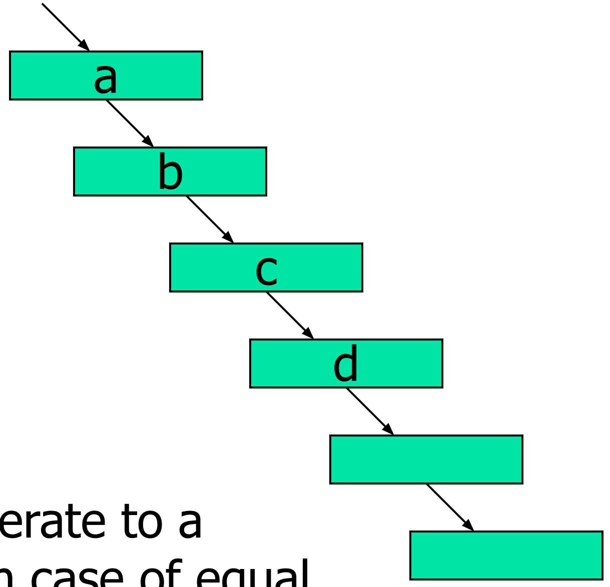


maintain absolute expiry time

START_TIMER = O(n)
STOP_TIMER = O(1)
PER_TICK_BOOKKEEPING = O(1)

Tree based timers





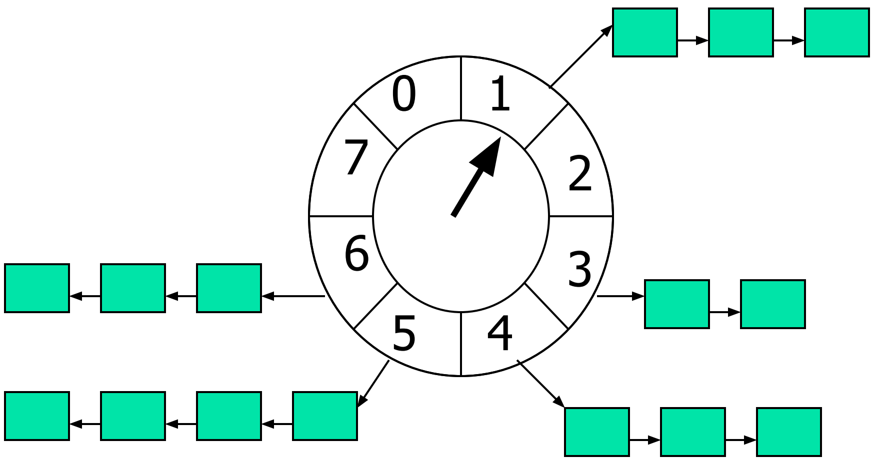
maintain absolute expiry time

START_TIMER = O(log(n))
STOP_TIMER = O(1)
PER_TICK_BOOKKEEPING = O(1)

Can degenerate to a linear list in case of equal Interval timers

START_TIMER = O(n)
STOP_TIMER = O(1)
PER_TICK_BOOKKEEPING = O(1)

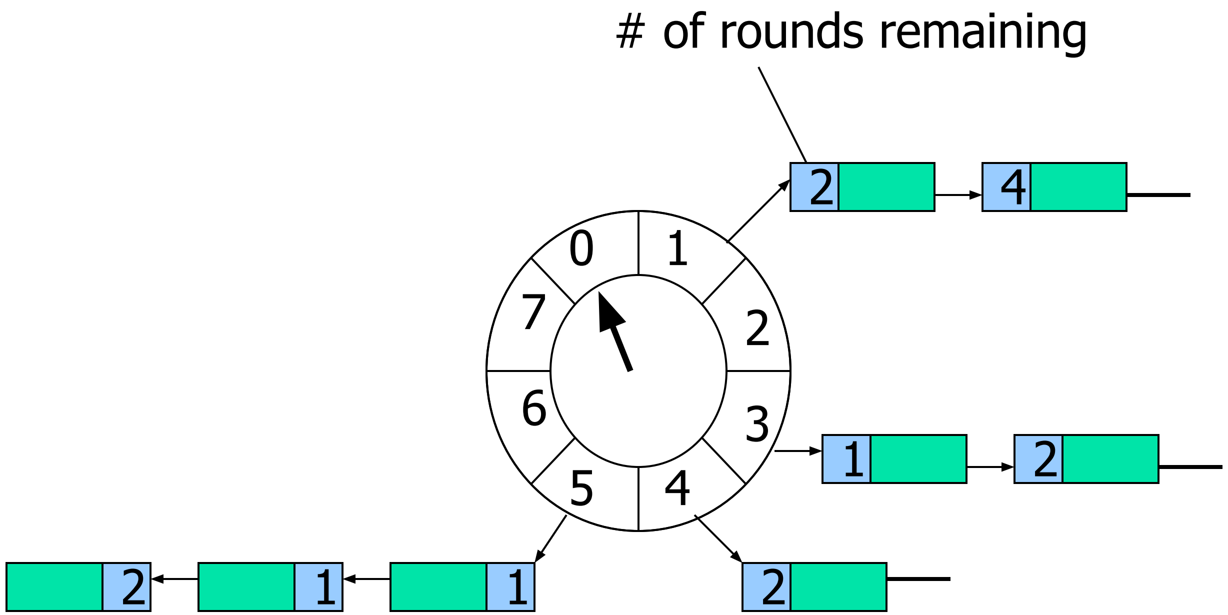
Simple Timing Wheel



START_TIMER = O(1)
STOP_TIMER = O(1)
PER_TICK_BOOKKEEPING = O(1)

- Keep a large timing wheel
- A curser in the timing wheel moves one location every time unit (just like a seconds hand in the clock)
- If the timer interval is within a rotation from the current curser position then put the timer in the corresponding location
- Requires exponential amount of memory

Hashed Timing Wheel

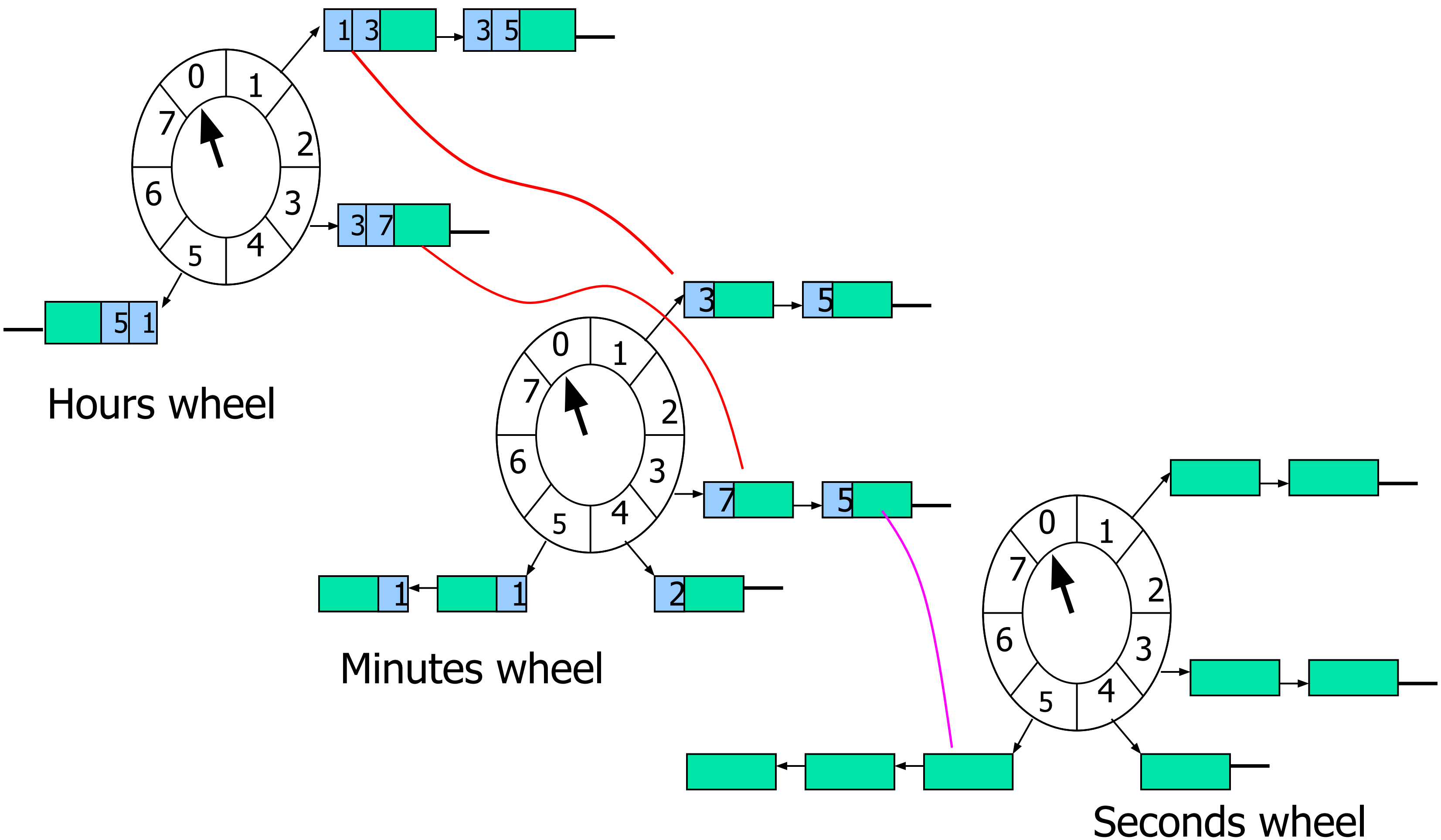


- Say wheel has 8 ticks
- Timer value = 17
- Make 2 rounds of wheel + 1 more tick
- Schedule the timer in the bucket "1"
- Keep the # rounds with the timer
- At the expiry processing if the # rounds > 0 then reinsert the timer

Hashed Timing Wheel

- Sorted Lists in each bucket
  - The list in each bucket can be insertion sorted
  - Hence START_TIMER takes O(n) time in the worst case
  - If n < WheelSize then average O(1)
- Unsorted list in each bucket
  - List can be kept unsorted to avoid worst case O(n) latency for START_TIMER
  - However worst case PER_TICK_BOOKKEEPING = O(n)
  - Again, if n < WheelSize then average O(1)

Hierarchical Timing Wheel



Hierarchical Timing Wheels

- START_TIMER = O(m) where m is the number of wheels
  - The bucket value on each wheel needs to be calculated
- STOP_TIMER = O(1)
- PER_TICK_BOOKKEEPING = O(1) on avg.

Comparison

	START_TIMER	STOP_TIMER	PER_TICK
Straight Fwd	O(1)	O(1)	O(n)
Sequential List	O(n)	O(1)	O(1)
Tree Based	O(log(n))	O(1)	O(1)
Simple Wheel	O(1)	O(1)	O(1)
Hashed Wheel (sorted)	O(n) worst case O(1) avg	O(1)	O(1)
Hashed Wheel (unsorted)	O(1)	O(1)	O(n) worst case O(1) avg
Hierarchical Wheels	O(m)	O(1)	O(1)

→ High memory requirement