Why care about timers?
- Operating system timers can be bottleneck
- Failure recovery based on timers
- Time integral part of some algorithms (rate based flow control, real time systems, etc.)
- Concrete uses of timers
  - Fine grained timers enable more accurate TCP timeouts (TCP Reno vs. TCP Vegas)
  - Multimedia systems – timers of 10-20 msec
  - Watchdog timer (one per disk access in FreeBSD)

Timer operation
- `StartTimer(Interval,Id,ExpiryAction)`
- `StopTimer(Id)`
- Per tick processing every $T$ seconds based on periodic hardware interrupt
  - `ExpiryAction` called when the timer expires
- Goal: minimize average and worst case time for all these operations
  - As a function of the number $n$ of timers

Naïve schemes
- Scheme 1: Store in array, decrement all on every tick – $O(1)$ start, $O(n)$ tick
- Scheme 2: Store in sorted list, check head of list on tick – $O(n)$ start, $O(1)^*$ tick

*Per tick processing must handle all timers that expire in that tick.

Obvious fix
- Scheme 3: use a heap or a (balanced) binary search tree – $O(\log n)$ start, $O(\log n)$ tick
- Better than sorted list but
  - Time to start timer a problem if we have many timers restarted often
  - More pointers per timer increase memory footprint

Solving a special case first
- Assumption: no timer is larger than `MaxInterval`
- Use circular array of size `MaxInterval`
- To start timer $j$ ticks past current time $i$, add to list at position $(i+j) \mod \text{MaxInterval}$
- On tick – `ExpiryAction()` for timers of next list
General solution – timing wheels

- Timers in same list might expire in different traversals of the array
- Solutions:
  - Sort the lists (slow insertion fast per tick)
  - Unsorted lists (slow per tick, fast insertion)
    - Amortized analysis shows that per tick cost not bad
  - Worst case* per tick work can be reduced using hash function to map timers to lists

Hierarchical wheels

- Many wheels each at different granularity
- Lists need no sorting (except coarsest wheel)

Microsecond timers

- Need timers with granularity of 10 μs
- Tremendous overhead if interrupt cost is 4 μs
- Solution 1 (P3(b) relax system requirements)
  - Check for timers on every system call, page fault, non-timer interrupt – soft timers (no “hard” guarantees, but would be accurate often)
- Solution 2 (P5 add hardware)
  - Have hardware clock interrupt processor only when a timer expires (instead of every 10 μs)
  - Low-cost solution for loading timers into clock?