Elevator synchronization and scheduling

COMPSCI210 Recitation

18th Mar 2013

Vamsi Thummala

Check point: Mission in progress

- Master synchronization techniques
- Develop best practices for writing synchronization code
- Write solid concurrent code

EventBarrier: Use case

- A complex computation can be divided and distributed among multiple tasks. Some parts of this computation can be I/O bound, the other parts are CPU intensive, and other are GPU operations that rely on specialized graphics chip. These partial results must be collected from various tasks for the final calculation. The result determines what other partial computations each task is to perform next.

EventBarrier: Another analogy (on piazza)

- Alice, Bob, and Charlie are three secret agents who are good in their respective domains: Math, Physics, and CS. They are given a jigsaw puzzle to solve, which demands the knowledge from all the three domains. However, due to the nature of operation involved there are certain constraints: they cannot talk to each other directly, and they cannot meet for more than 10 minutes at a time. There is an agent coordinator, who arranges rendezvous, whenever all the agents agrees to meet. They worked out a plan: all agents work independently on a certain task and notifies the coordinator when they are done with that task and want to meet (through arrive() call), and wait perpetually until the coordinator responds with details (through raise() call). Once all three agents notifies the coordinator, the coordinator send the details of rendezvous, and they all meet and synchronize on the tasks, and dissemble. With the collective new found knowledge, they start working independently again the next day, and this process continues until the puzzle is solved.

Testing EventBarrier

- Say you have n consumers with some local variable set to "phase1". On complete(), each consumer increments their count. For example, the second iteration their local variable will be set to "phase2". But the barrier does not return until all the consumers arrived. So if you have print() statement after the barrier, you should see all the consumers printing "phase2". If some consumer prints "phase1" that means that complete() did not happen but still passed through the barrier. Hence, indicative of a bug.

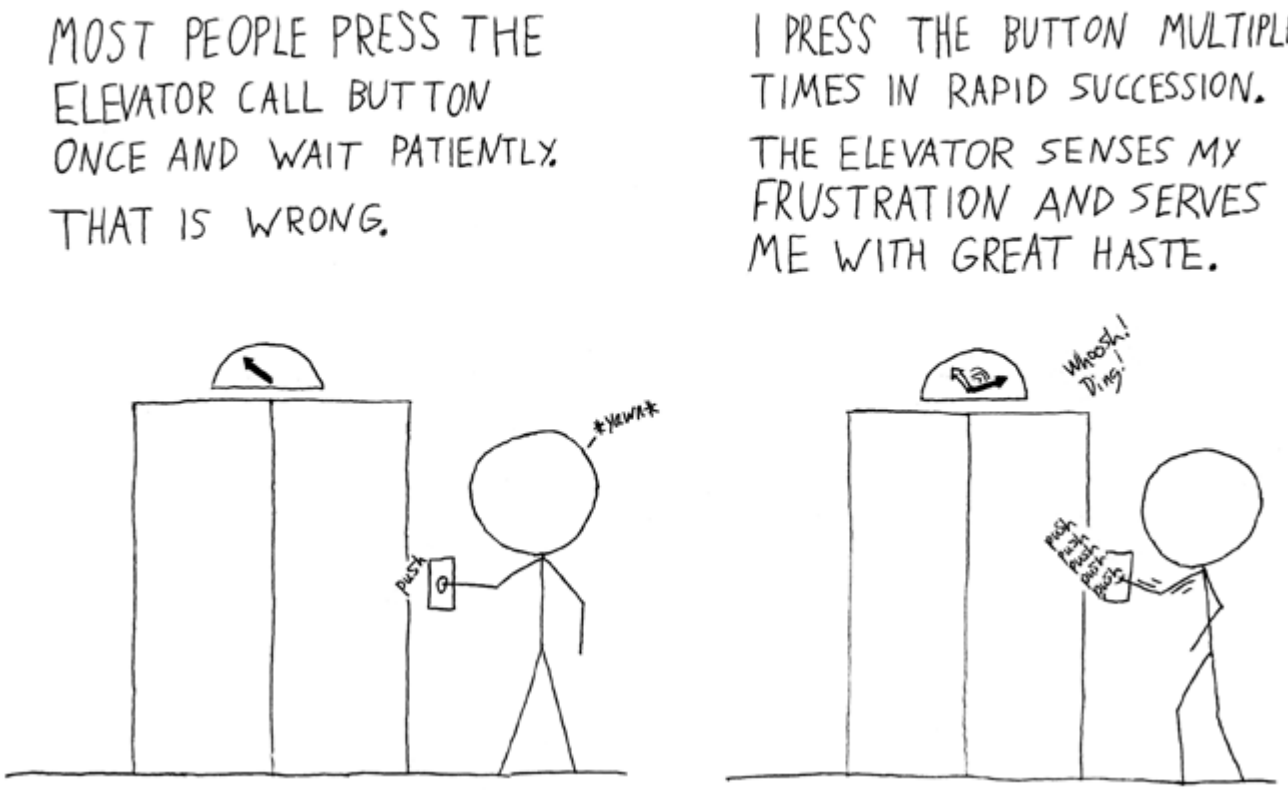
Using EventBarrier for Elevator

- Where to place the EventBarrier?
- How many are needed per building with F floors?
- How many are needed per building with F floors and E elevators?

Elevator data structure(s)

- ElevatorController
  - Pool/Queue of events
    - CallUp/CallDown
      - Can return an elevator
- Elevator
  - Pool/Queue of requests
    - Direction
    - Destination floor

What metrics do you consider for elevator scheduling?



Metrics for elevator scheduling

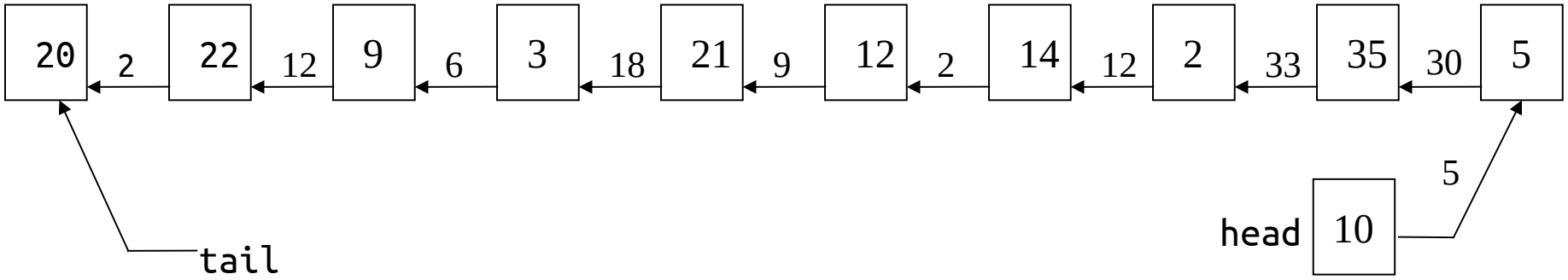
- Service time
  - Time between pushing the button and exit the elevator
  - Approximate
    - Wait time
- Fairness
  - Variation in the service time(s)
- Efficiency
  - Roughly defined as the amount of total work done (Energy)
  - Work done: Number of floors the elevators pass in total
- Objective: Minimize service time, Maximize fairness, Minimize work done

First Come First Served (FCFS)

- Service in the order in which the requests are made
  - Riders enter and press the destination floor
- Simple to implement
- No starvation
  - Every request is serviced
- Is FCFS a good policy?

FCFS

- The elevator is currently servicing the 10th floor
- Order of requests from riders at the 10th floor:
  5 (down), 35 (up), 2 (down), 14 (up), 12 (up), 21 (up),
  3 (down), 9 (down), 22 (up), 20 (up)
- To simplify, let us assume everyone gets in



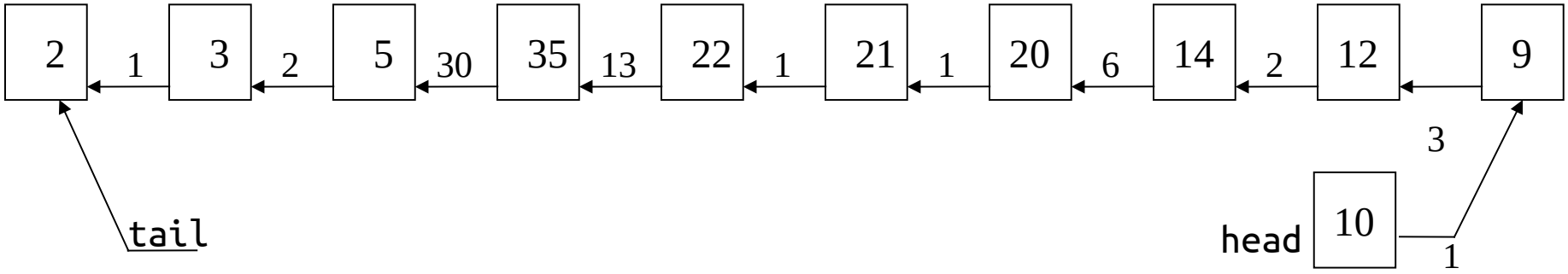
- Total service time (assuming 1 unit time per floor serviced):

5 + 30 + 33 + 12 + 2 + 9 + 18 + 6 + 12 + 2 = 129, Avg: 12.9

- Can we do better?
  - Service the closest floor

Shortest Seek Time First (SSTF)

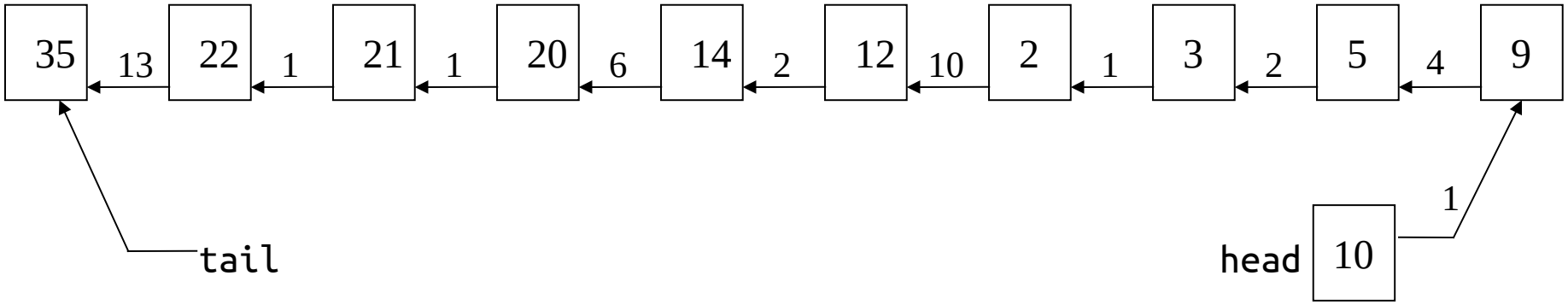
- Go to the closest floor in the work queue
- Reduces total seek time compared to FCFS
- Order of requests from riders at the 10th floor:
  5 (down), 35 (up), 2 (down), 14 (up), 12 (up), 21 (up),
  3 (down), 9 (down), 22 (up), 20 (up)



- Total service time (assuming 1 unit time per floor serviced):
  1 + 3 + 2 + 6 + 1 + 1+ 13 + 30 + 2 + 1 = 60, Avg: 6
- Disadvantages:
  - Starvation possible
  - Switching directions may slow down the actual service time
- Can we do better? Reorder the requests w.r.t direction

SCAN

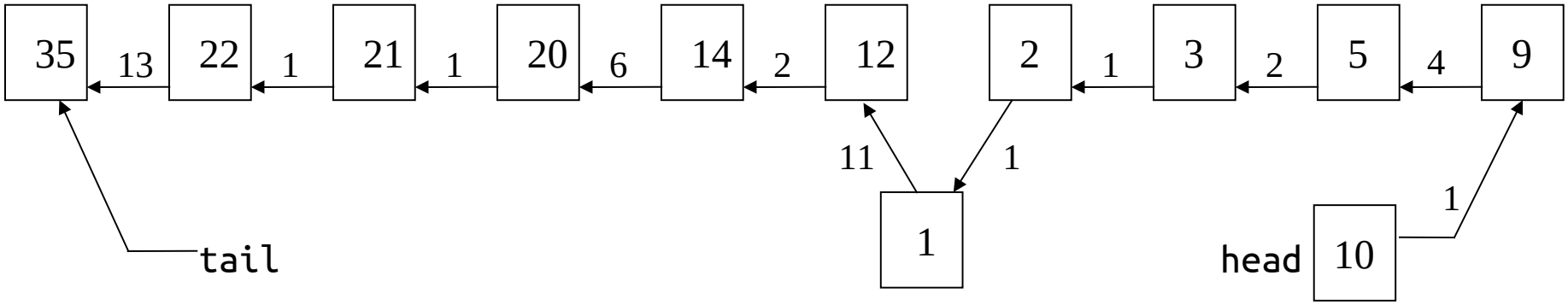
- Start servicing in a given direction to the end
  - Change direction and start servicing again
- Order of requests from riders at the 10th floor:
  5 (down), 35 (up), 2 (down), 14 (up), 12 (up), 21 (up),
  3 (down), 9 (down), 22 (up), 20 (up)



- Total service time (assuming 1 unit time per floor serviced):
  1 + 4 + 2 + 1 + 10 + 2 + 6 + 1 + 1 + 13 = 41, Avg: 4.1
- Advantages
  - Reduces variance in seek time
- Can we do better?

Circular SCAN (C-SCAN)

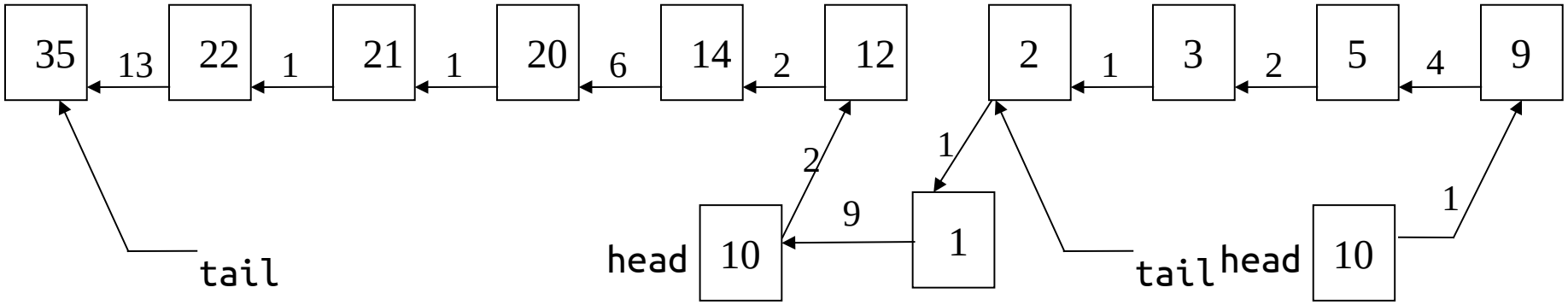
- Start servicing in a given direction to the end
  - Go to the first floor without servicing any requests;
  - Restart servicing
- Order of requests from riders at the 10th floor:
  5 (down), 35 (up), 2 (down), 14 (up), 12 (up), 21 (up),
  3 (down), 9 (down), 22 (up), 20 (up)



- Total service time (assuming 1 unit time per floor serviced):
  1 + 4 + 2 + 1 + 1 + 11 + 2 + 6 + 1 + 1 + 13 = 43, Avg: 4.3
- Advantages
  - More fair compared to SCAN
- Is this what you expect in a real-world elevator?

Elevator Scheduling

- At least one difference from C-SCAN
  - Direction of pick up
- Order of requests from riders at the 10th floor:
  5 (down), 35 (up), 2 (down), 14 (up), 12 (up), 21 (up),
  3 (down), 9 (down), 22 (up), 20 (up)



- Total service time (assuming 1 unit time per floor serviced):
  1 + 4 + 2 + 1 + 1 + 9 + 2 + 2 + 6 + 1 + 1 + 13 = 43, Av: 4.3
- Can you do better?
  - We look forward to your lab submissions

Disk Scheduling

- Similar to elevator scheduling
- Each disk has a queue of jobs waiting to access disk
  - read jobs
  - write jobs
- Each entry in queue contains the following
  - pointer to memory location to read/write from/to
  - sector number to access
  - pointer to next job in the queue
- OS usually maintains this queue