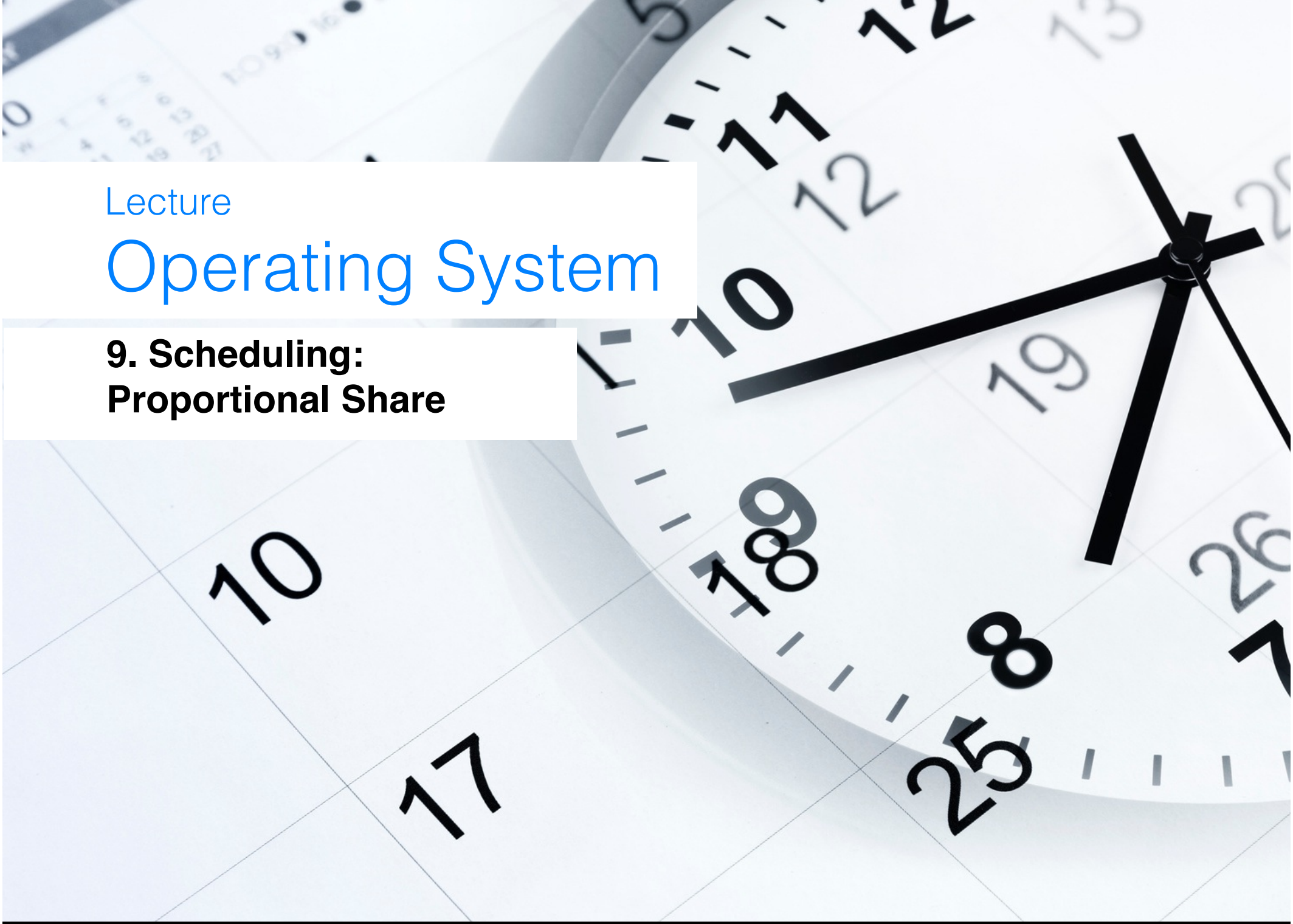


Lecture

Operating System

9. Scheduling: Proportional Share

Professor Dr. Michael Mächtel

9. Scheduling: Proportional Share

How to share CPU in a promotional manner?

What are the key mechanism for doing so?

How effective are they?

Proportional Share Scheduler

- **Fair-share** scheduler
  - Guarantee that **each job** obtain a certain **percentage** of CPU time.
  - **Not optimized** for turnaround or response time

Basic Concept

- **Tickets**
  - **Represent the share** of a resource that a process should receive
  - The percent of tickets represents its share **of the system resource** in question.
- Example
  - There are two processes, A and B.
    - Process A has 75 tickets ➙ receive 75% of the CPU
    - Process B has 25 tickets ➙ receive 25% of the CPU

9. Scheduling: Proportional Share

1. Lottery Scheduling

2. Stride Scheduling

9. Scheduling: Proportional Share

1. Lottery Scheduling

2. Stride Scheduling

Lottery scheduling

- The scheduler **randomly** picks a winning ticket.
  - Load the state of that winning process and runs it.
- Example
  - There are 100 tickets
    - Process A has 75 tickets: 0 ~ 74
    - Process B has 25 tickets: 75 ~ 99

Scheduler's winning tickets:	63	85	70	39	76	17	29	41	36	39	10	99	68	83	63
Resulting scheduler:	A	B	A	A	B	A	A	A	A	A	A	B	A	B	A

The longer these two jobs compete,
The more likely they are to achieve the desired percentages.

Ticket Mechanisms

- **Ticket currency**
  - A user allocates tickets among their own jobs in whatever currency they would like.
  - The system converts the currency into the correct global value.
  - Example
    - There are 200 tickets (Global currency): Process A has 100 tickets and Process B has 100 tickets

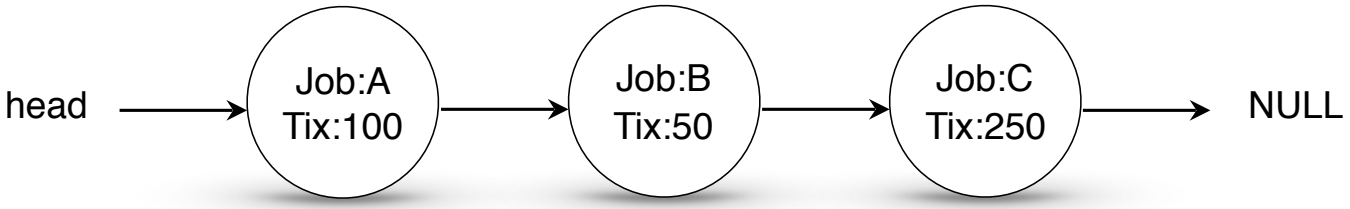
User A	→ *500* (A's currency) to A1 → *50* (global currency)
	→ *500* (A's currency) to A2 → *50* (global currency)
User B	→ *10* (B's currency) to B1 → *100* (global currency)

Ticket Mechanisms (Cont.)

- **Ticket transfer**
  - A process can temporarily hand off its tickets to another process.
- **Ticket inflation**
  - A process can temporarily raise or lower the number of tickets is owns.
  - If any one process needs more CPU time, it can boost its tickets.

Implementation

- Example: There are there processes, A, B, and C.
  - Keep the processes in a list:



- Algorithm
  - Pick winning ticket
  - walk through list and update a counter by sum up `Tix`
    - until sum is > winning ticket -> Job found

Implementation (Cont.)

```
1   // counter: used to track if we've found the winner yet
2   int counter = 0;
3
4   // winner: use some call to a random number generator to
5   // get a value, between 0 and the total # of tickets[1]
6   int winner = getrandom(0, totaltickets);
7
8   // current: use this to walk through the list of jobs
9   node_t *current = head;
10
11  // loop until the sum of ticket values is > the winner
12  while (current) {
13      counter = counter + current->tickets;
14      if (counter > winner)
15          break; // found the winner
16      current = current->next;
17  }
18  // 'current' is the winner: schedule it ...
```

1: http://stackoverflow.com/questions/2509679/how-to-generate-a-random-number-from-within-a-range

Lottery Fairness Study

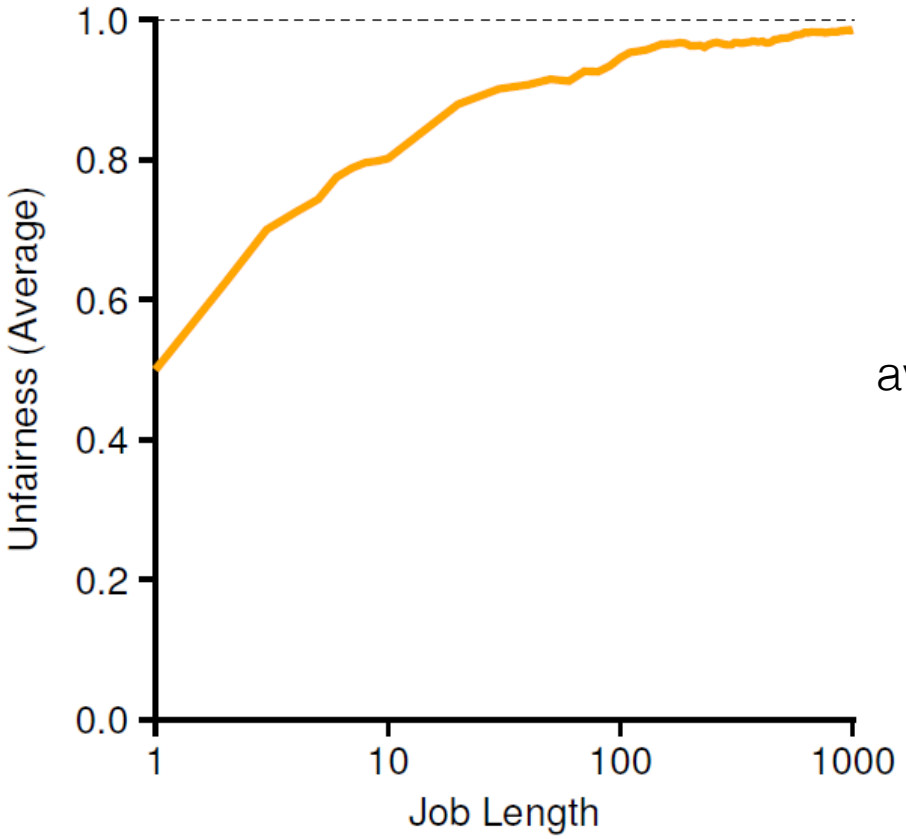
- ***U***: unfairness metric
  - The time the first job completes divided by the time that the second job completes.
- Example:
  - There are two jobs, each jobs has runtime 10.
    - First job finishes at time 10
    - Second job finishes at time 20

$$U = \frac{10}{20} = 0.5$$

- ***U*** will be close to 1 when both jobs finish at nearly the same time.

Lottery Fairness Study (Cont.)

- There are two jobs.
  - Each jobs has the same number of tickets (100).



When the job length is not very long, average unfairness can be quite severe.

9. Scheduling: Proportional Share

1. Lottery Scheduling

2. Stride Scheduling

Stride Scheduling

- Problem: How to assign tickets to jobs
  - Randomness (Lottery Scheduling)
    - but no fair if jobs are short
  - By the user?
- Another Solution: Stride of each process
  - (A large number) / (the number of tickets of the process)
  - Example: A large number = 10,000
    - Process A has 100 tickets ➙ stride of A is 100
    - Process B has 50 tickets ➙ stride of B is 200
- A process runs, **increment** a counter(=pass value) for it **by its stride**.
  - Pick the process to run that has the lowest pass value

A Pseudo Code implementation

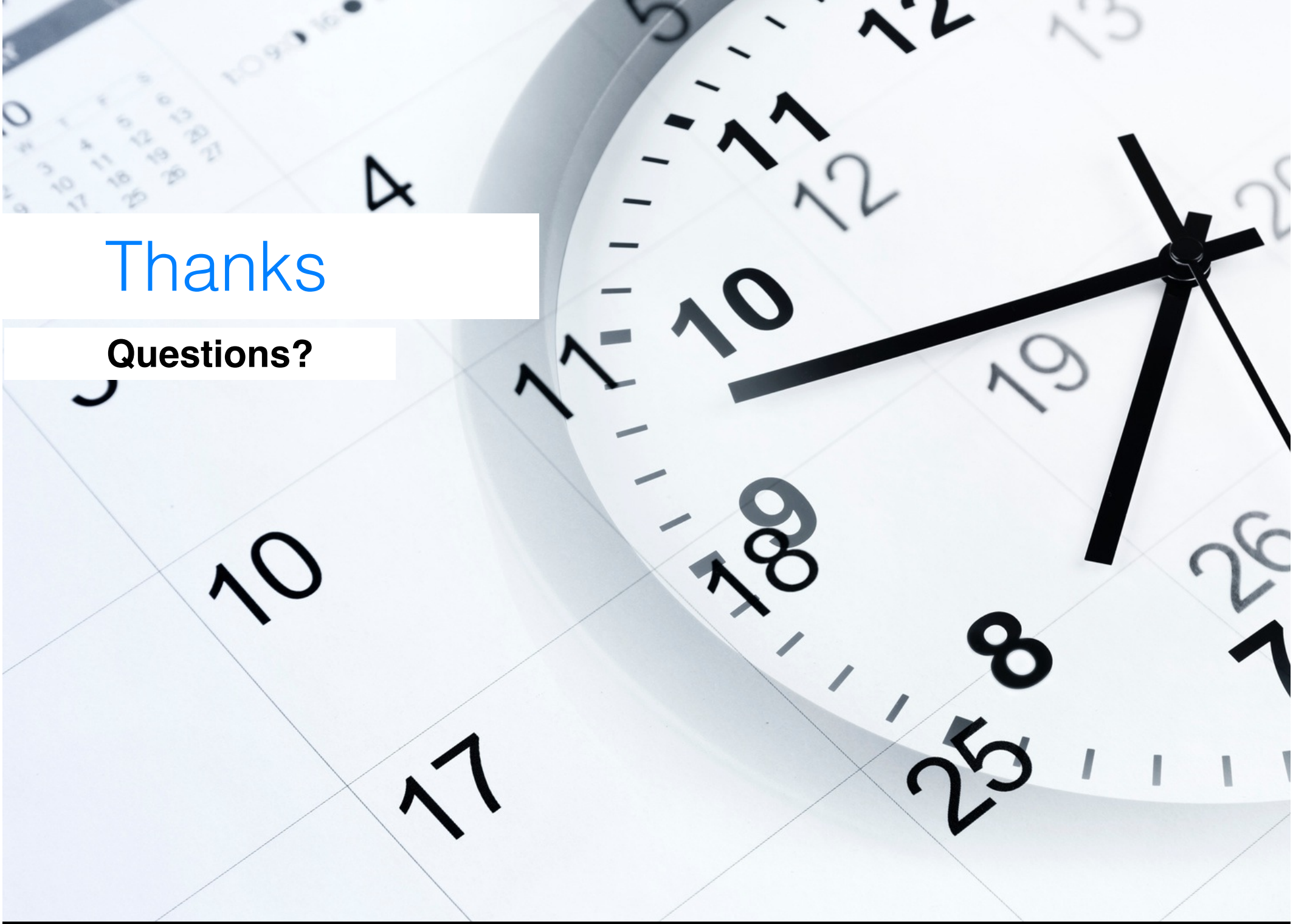
```
current = remove_min(queue);          // pick client with minimum pass
schedule(current);                    // use resource for quantum
current->pass += current->stride;     // compute next pass using stride
insert(queue, current);               // put back into the queue
```

Pass(A) *(stride=100)*	Pass(B) *(stride=200)*	Pass(C) *(stride=40)*	Who runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

If new job enters with pass value 0,
It will monopolize the CPU!

Summary

- Proptional-Share Scheduling
  - Lottery (using randomness)
  - Stride (deterministic)
- Conceptually interesting but not wide-spread adaption
  - does not work well with I/O
  - Problem: How to assign tickets?
- MLFQ and similar schedulers do so more gracefully



Thanks

Questions?