University of Pittsburgh

Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 5 is due **this Friday**
  - Project 1 is due **this Friday** at 11:59 pm
  - Lab 2 is due on Tuesday 2/28 at 11:59 pm
  - Project 2 will be posted this Friday

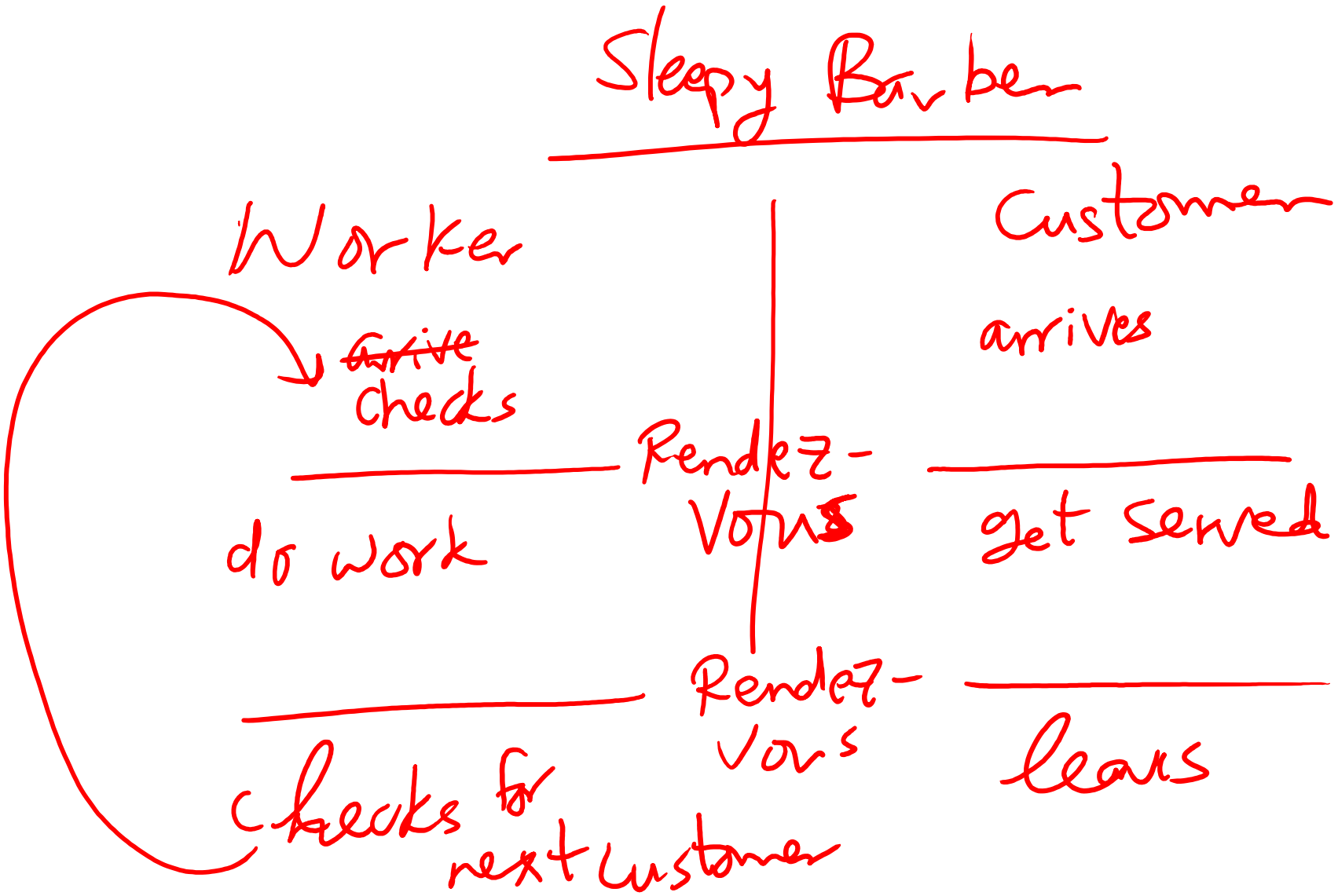
Previous lecture …

- Deadlock detection and avoidance using the Banker's algorithm
- Sleepy Barbers problem

Problem of the Day: Sleepy Barbers

- We have two sets of processes
  - Worker processes (e.g., barbers)
  - Customer processes
- Customer processes may arrive at anytime
- Worker processes check in when they are not serving any customers
- Each worker process must **wait** until it gets matched with a customer process
- Each customer process must **wait** until it gets matched with a worker process
- The customer process cannot leave until the matched worker process finishes the work
- The worker process cannot check in for the next customer until the matched customer process leaves

Rendezvous Pattern



Solution Using Semaphores: Take 1

- One pair of semaphores per rendezvous
  - RV1a and RV1b
  - RV2a and RV2b
- Notice the flipped order of the down and up calls in the two processes

Worker

Semaphores RV1a, RV1b (0) (0)

RV2a, RV2b (0) (0)

arrives/checks in

down(RV1a)

up(RV1b)

does work

up(RV2a)

down(RV2b)

Customer

arrives

up(RV1a)

down(RV1b)

gets served

down(RV2a)

up(RV2b)

Solution Using Semaphores: Take 1

- This solution doesn't work for multiple workers and multiple customers
  - In that case, a customer can leave before its associated worker finishes

Sleepy Barbers Solution: Take 2

```
struct mysems {
   Semaphore RV1a(0), RV1b(0), RV2a(0), RV2b(0);
};
SharedBuffer buff; //From producers-consumers problem
```

Worker Process

```
struct mysems sems = buff.consume();
up(sems.RV1a);
down(sems.RV1b);
//do work
down(sems.RV2a);
up(sems.RV2b);
//check-in for next customer
```

Customer Process

```
struct mysems sems = new struct mysems
buff.produce(sems);
down(sems.RV1a);
up(sems.RV1b);
//get work
up(sems.RV2a);
down(sems.RV2b);
//leave
```

Solution using Mutex and Condition Variables

- https://cs1550-2214.github.io/cs1550-code-handouts/ProcessSynchronization/Slides/