CSCI 460 Operating Systems

Processes & Threads

Professor Travis Peters
Fall 2019

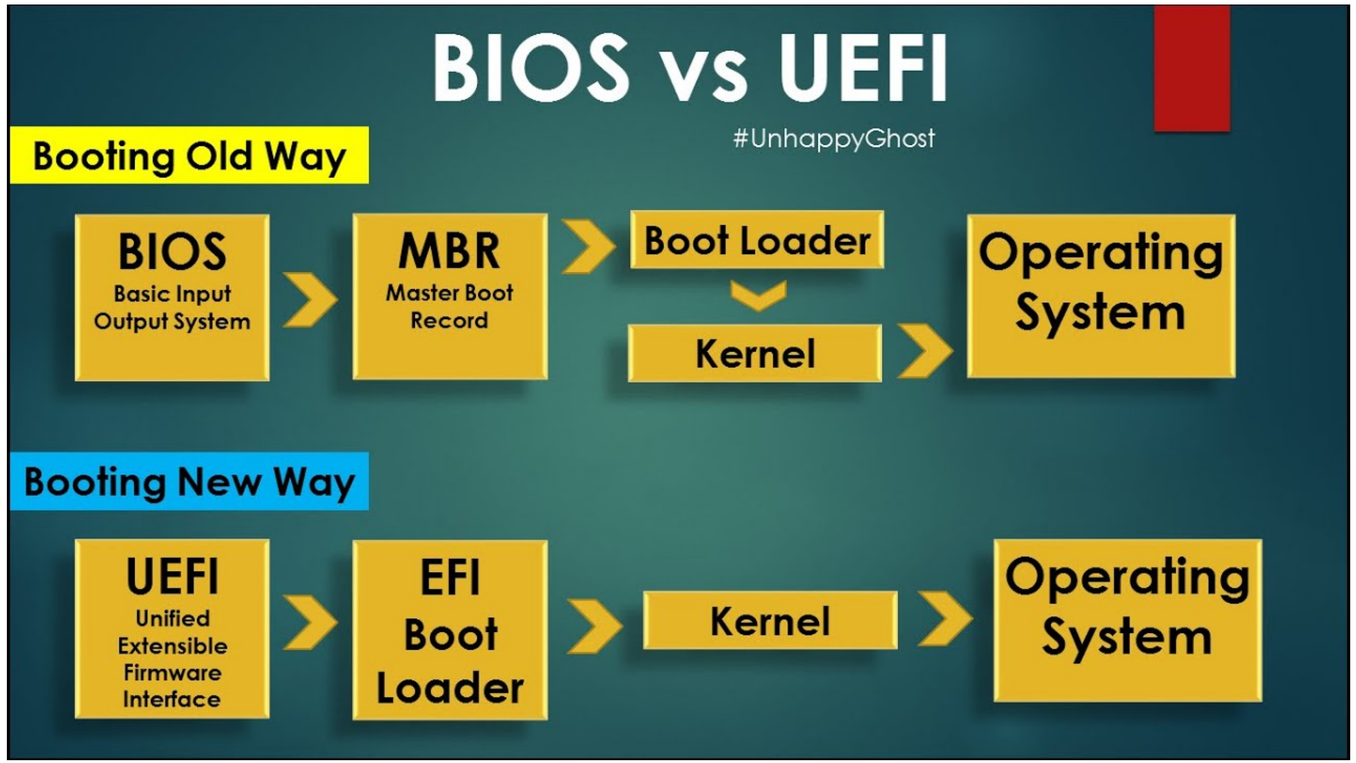
Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

Following Up on Questions About Boot Loaders

- BIOS is dead…
  - bootable-drive size limitations (MBR)
  - 16-bit mode / mem. limitations (1MB)
  - no networking pre-OS
  - + slow, etc.
- …long live UEFI! – *https://uefi.org*
  - boot from larger drives,
    run in 32-/64-bit mode,
    better UI, Secure Boot,
    networking/remote config.

  ➡ *UEFI is essentially a tiny OS!*
- Other relics…
  - CMOS vs. Flash/EEPROM, …



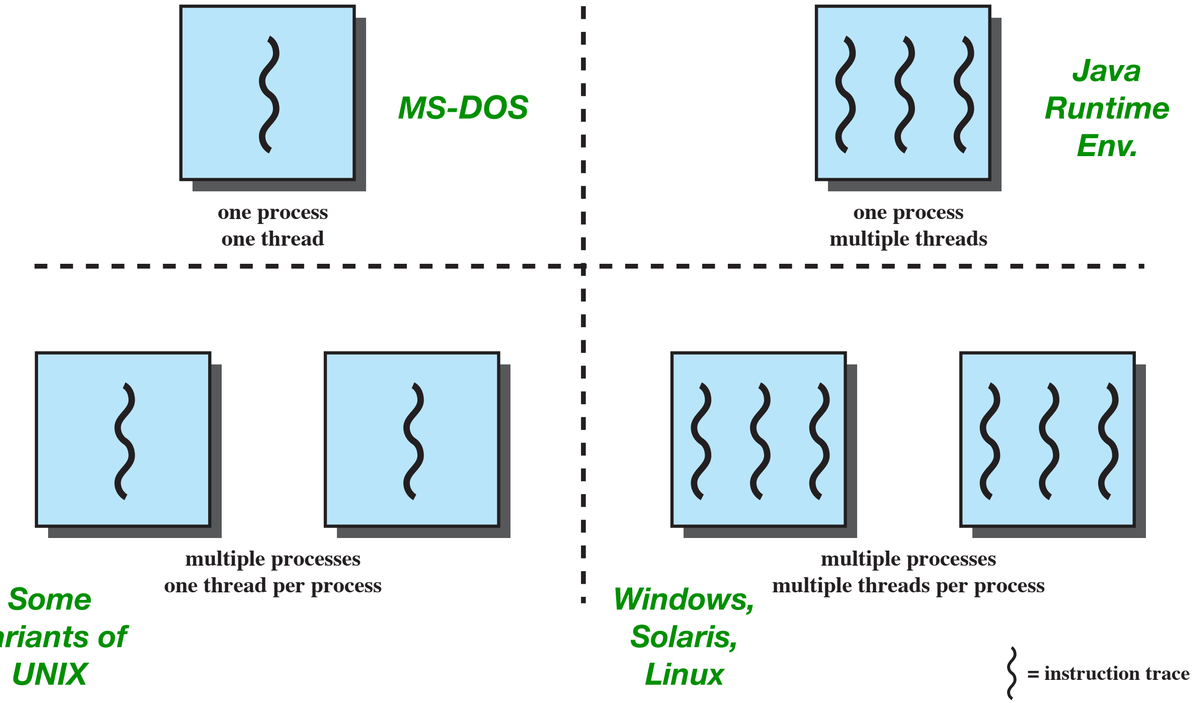
Potential Final Project? ;-)

Goals for Today

- **Learning Objectives**
  - Understand the basic concept of threads and how they relate to the concept of a process
- **Announcements**
  - zyBook for OS is now accessible (optional)
  - *Coming Soon…*
    - 1st programming assignment — on basic sys programming + concurrency

Processes vs. Threads

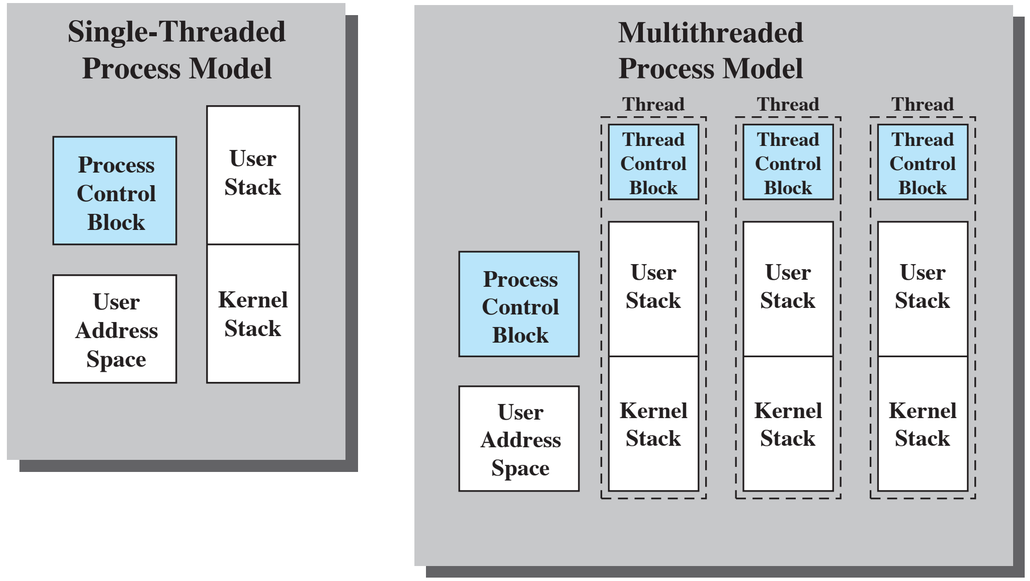
- What is the difference between a ***process*** and a ***thread***?
- Processes can be further divided in terms of their responsibilities:
  - 1st Part = **resource ownership** (*process* or *task*)
  - 2nd Part = **scheduling/execution** (*thread* or *lightweight process*)
- **Multithreadng**—*the ability of an OS to support mutliple, concurrent paths of execution within a single process.*



Processes vs. Threads

- Both provide independent execution sequences, but…
  - ***Processes***…
    - each have their own ***private*** memory space
    - each have their own ***resources***
      (protected access to processors, other processes (IPC), files, and I/O (devices and channels)
  - ***Threads***…
    - run in a ***shared*** memory space (the process)
    - have their own ***execution state*** (Running, Ready, etc.), ***execution context*** (think PC), ***execution stack***, some ***"thread local" storage***, etc.
  - ➡ Potentially many threads per process

NOTE: Thus far we have been assuming one thread per process



Processes vs. Threads — ***WHY***

➡ *Threads (can) prevent a process from blocking entirely*

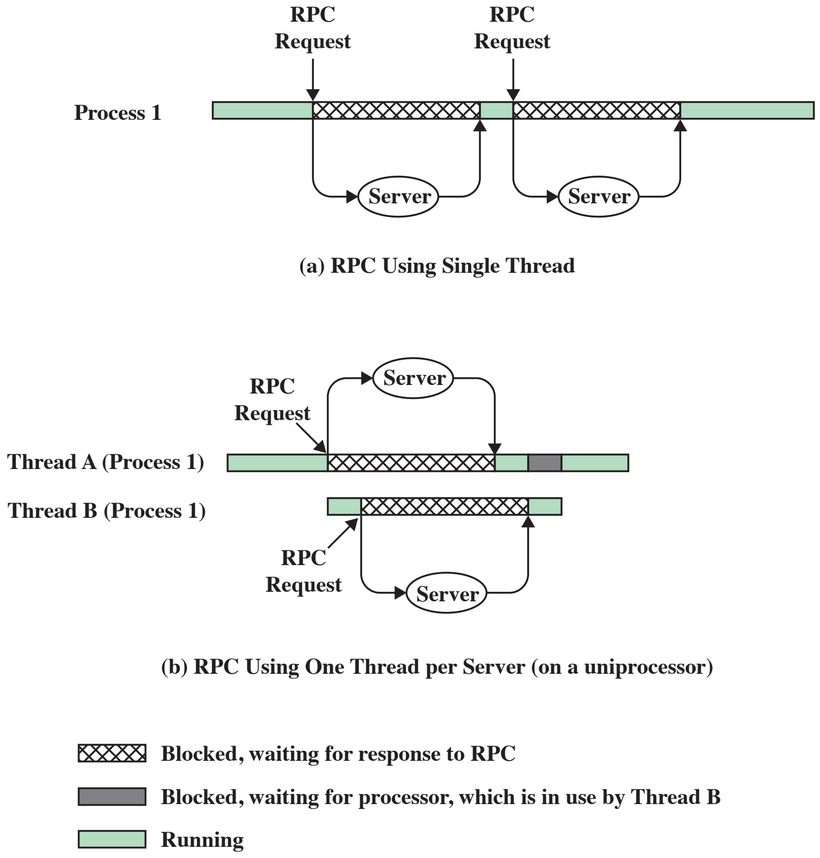


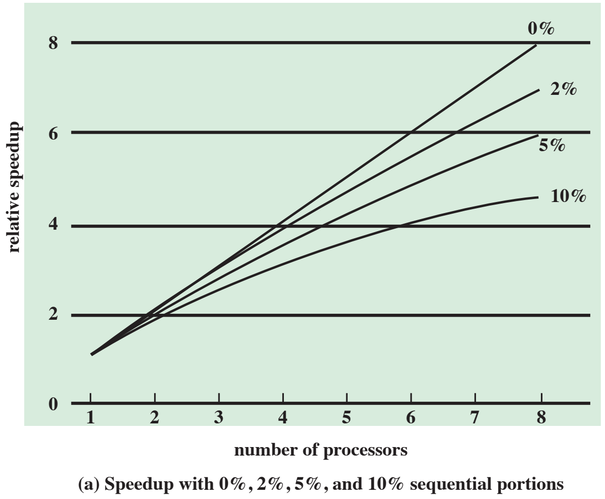
Figure 4.3 Remote Procedure Call (RPC) Using Threads

Processes vs. Threads — ***WHY***

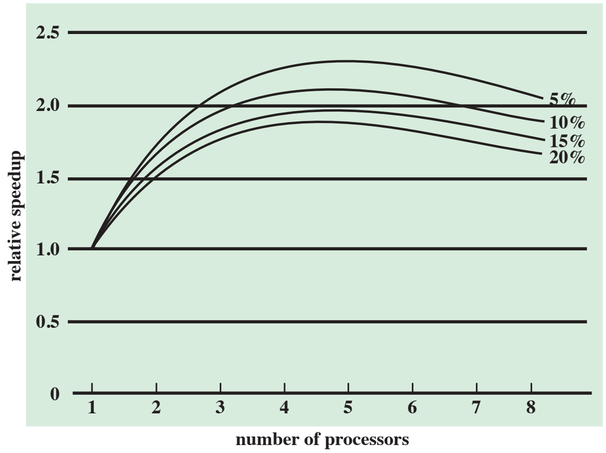
➡ *It takes far less time to create, terminate, switch between, and communicate among threads as compared to processes!*

Table 4.1 Thread and Process Operation Latencies (μs)

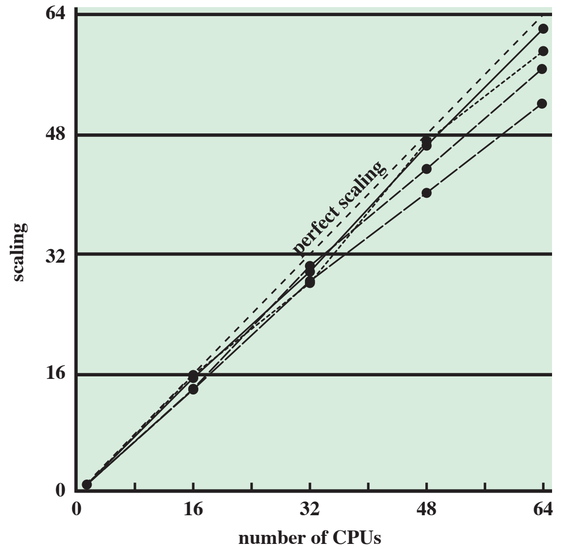
Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads



Processes vs. Threads — ***HOW***

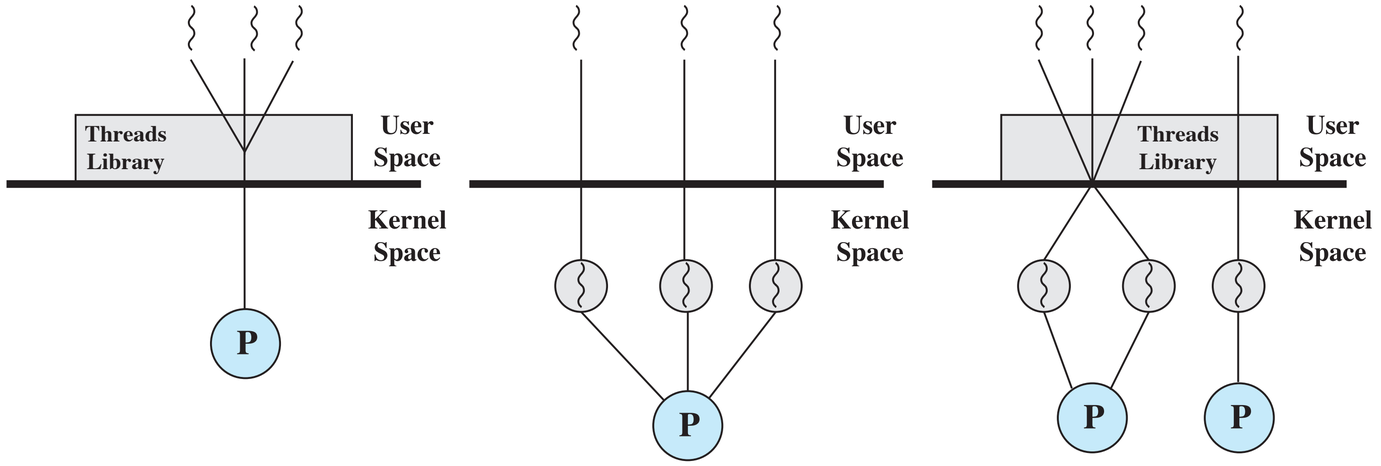
- What is the difference between **user threads** and **kernel threads**?
- pros and cons?

User Threads

- fast context switching
- customized scheduling

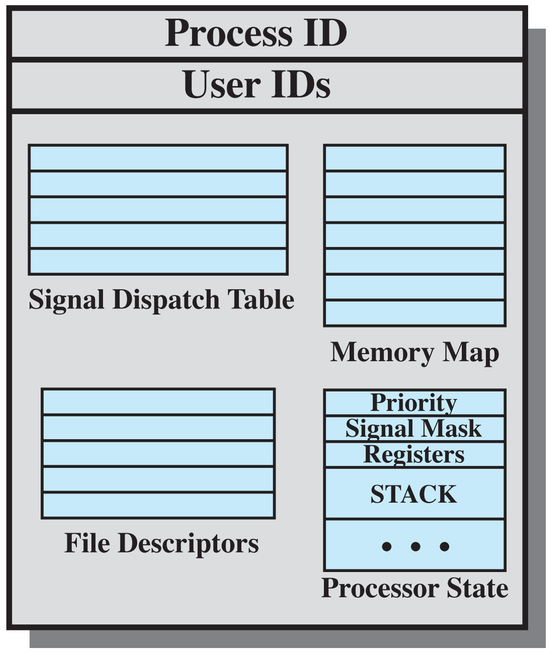
Kernel Threads

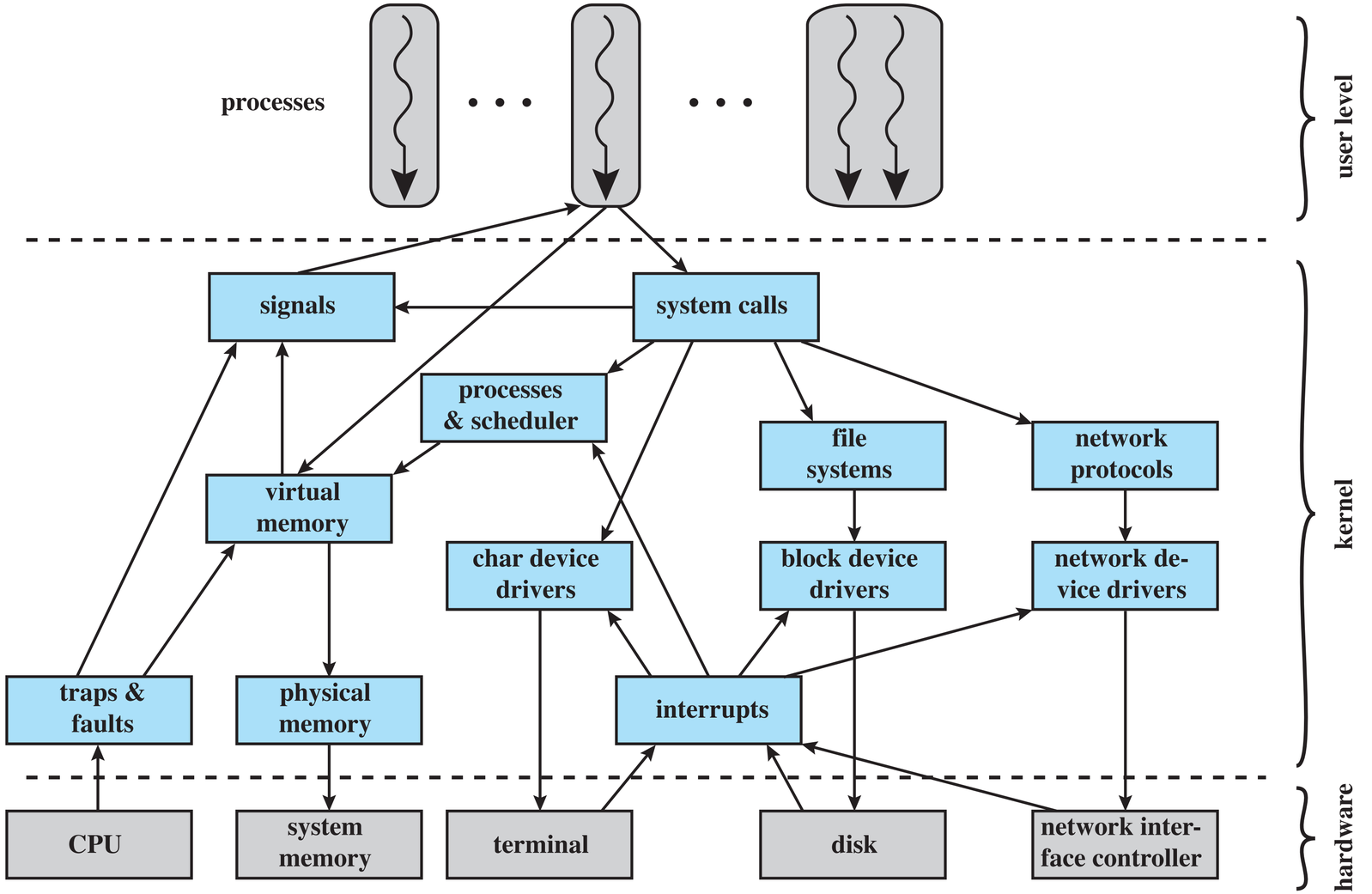
- each thread can make blocking I/O calls
- can run conncurrently on multiple processors



(a) Pure user-level (b) Pure kernel-level (c) Combined

After Today, Be Sure to Review
Processes & Threads in UNIX & The Linux Kernel





Activity: Think-Pair-Share Summary of Processes & Threads

1. Spend a few minutes alone summarizing everything you remember about processes, and the role of the OS in controlling them and managing resources.

Use whiteboards around the room to write on?!

2. Spend a few minutes reviewing your summary with a neighbor.

Did you miss anything?

3. Come back together and share.

What idea(s) seem to be the most critical to understand going forward?

What is still unclear?