CS 423
Operating System Design:
Final Review
May 5

Ram Kesavan

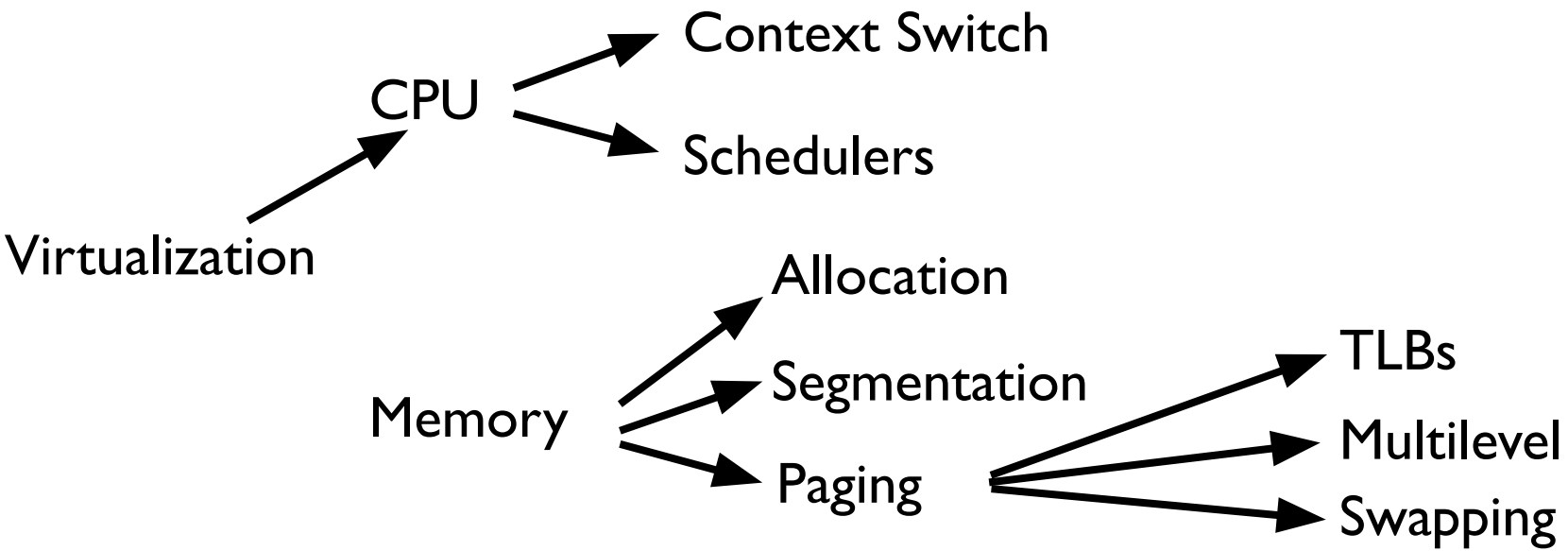
Logistics

Midterm papers: pickup from my office during OH
MP4 due by May 6, 11:59 CT

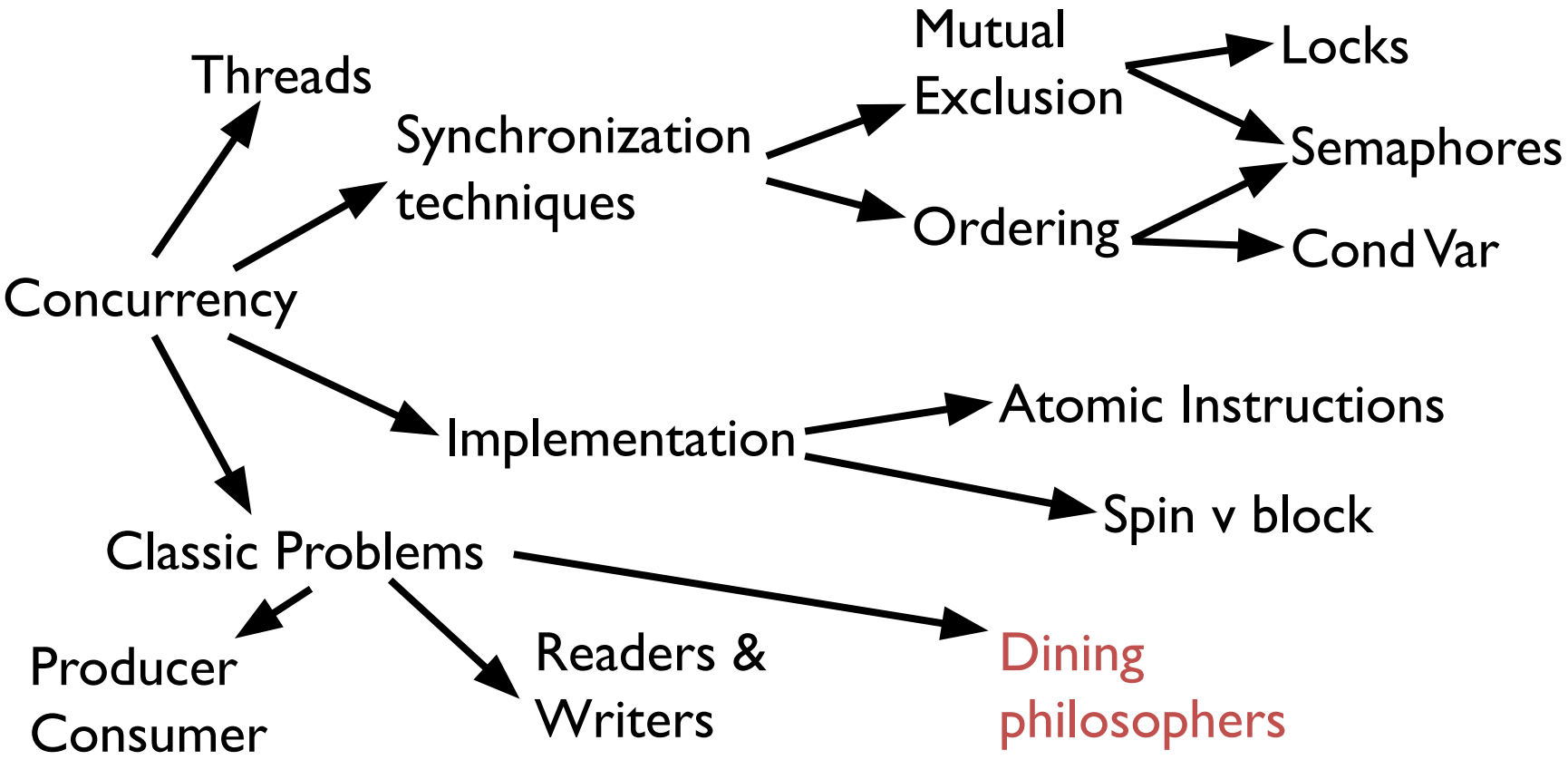
Will publish some practice questions this week
- Discuss them on Piazza
- Will post solutions on Piazza in the weekend

Finals: 5/13 at 7-10pm, DCL 1320 (next door)

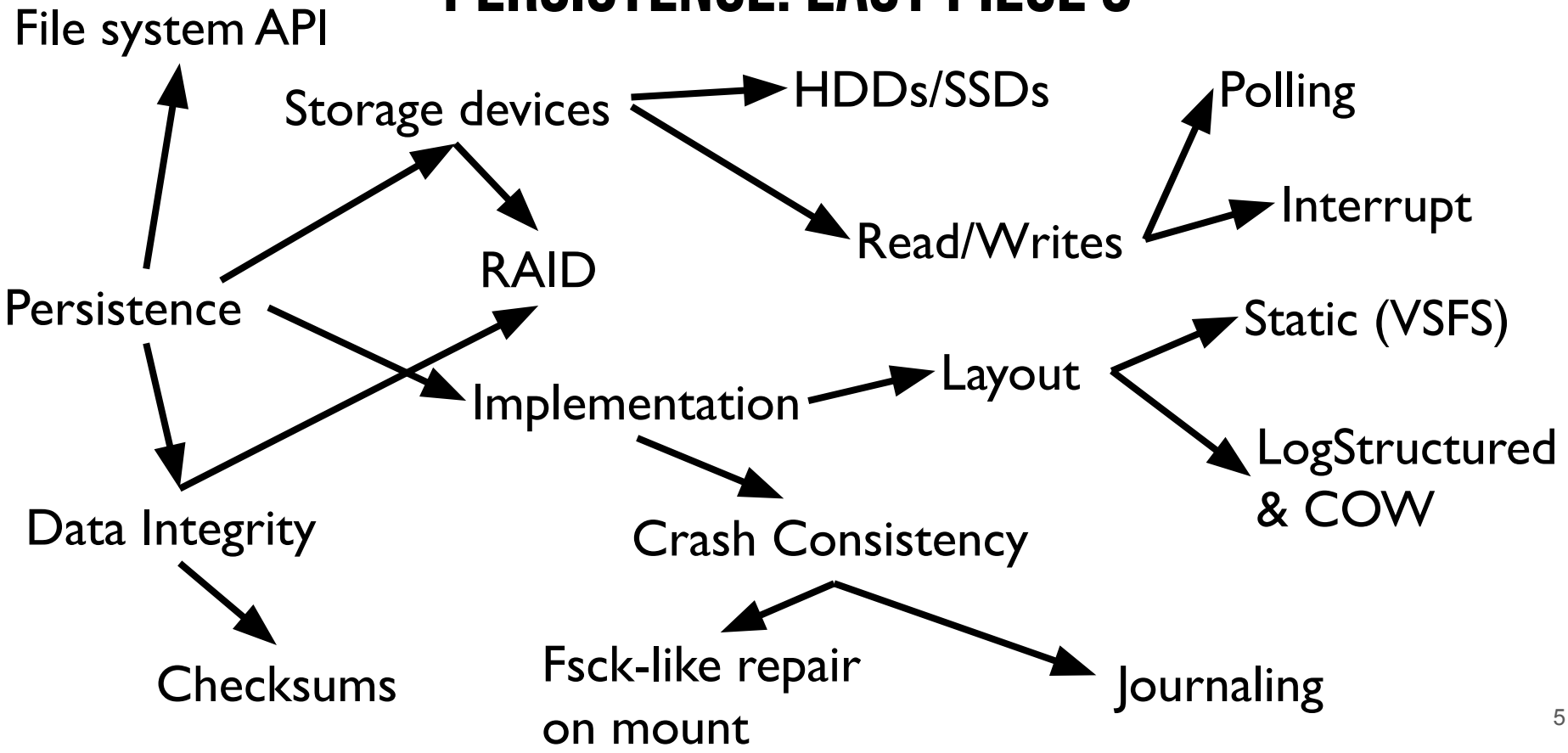
VIRTUALIZATION: EASY PIECE 1



CONCURRENCY: EASY PIECE 2



PERSISTENCE: EASY PIECE 3



Final Exam Format

Similar to midterm: MCQs & long-form questions

- MCQ: **only 1 correct answer**
- Expect to read & debug some code
- Questions on MP3 and MP4
- More questions on 2nd half of the sem
  - Semaphores, storage devices, RAID, file systems

Should not require 3 hours :-)

File Systems: in kernel or user-space?

OS needs at least 1 file system inside the kernel

In kernel: speed, avoid context-switches

In user-space: debugging, portability, safety/isolation

Popular API for building user-space file system:

- FUSE (file system in userspace)

Distributed file systems typically in user-space

- Context switch latency < network hop latency

SSD v HDD

SSDs: high performance with lower endurance
- High-end storage systems
- For low-latency workloads
- Petabyte (and less) scale storage

HDDs: still the workhorse for storage media-type today
- Large-scale distributed storage
- Exabyte+ scale storage

SSDs to HDDs:
- 16x for $/GB
- More expensive for sequential IOPS
- But much cheaper for random IOPS

Most storage systems use tiering strategies:
- Put "hot" data in SSDs & "cold" data in HDDs
- Within the same file system or SSDs as a separate cache layer

Sequence of NFS v3 messages

client % *sudo mount -t nfs <server-ip>:/home /mnt/fs1*
// remote file system has now been mounted at local dir /mnt/fs1
// client knows that fs1 maps to remote NFS server
// client has fetched and mapped file handle of ROOT from remote server to local "fs1"

User app: `open("/mnt/fs1/home/user/doc.txt",xxx);`

1. C→S: LOOKUP (dir_handle = ROOT, file = "home")
   S→C: fh_home // fh_home = <inum, generation> & attributes
2. C→S: LOOKUP (dir_handle = fh_home, file = "user")
   S→C: fh_user // and attributes
3. C→S: LOOKUP (dir_handle = fh_user, file = "doc.txt")
   S→C: fh_doc // and attributes
4. C→S: ACCESS (file_handle = fh_doc, access_requested = R/W)
   S→C: allowed permissions
5. C→S: GETATTR (file_handle = fh_doc) // get the latest so it can be cached
   S→C: latest file metadata (size, last modify time, owner, etc.)
6. C's kernel opens file, and returns fd to user application

Sequence of NFS v4 messages

client % *sudo mount -t nfs <server-ip>:/home /mnt/fs1*
// remote file system has now been mounted at local dir /mnt/fs1
// client knows that fs1 maps to remote NFS server
// client has fetched and mapped file handle of ROOT from remote server to local “fs1”

User app: `open(“/mnt/fs1/home/user/doc.txt”,xxx);`

1. C→S: Compound message
   a. PUTFH (dir_handle = ROOT)
   b. LOOKUP (“home”)
   c. LOOKUP (“user”)
   d. LOOKUP (“doc.txt”)
   e. OPEN
2. S→C: S executes all operations sequentially, and responds with fh of “doc.txt”, or appropriate error message

If client wants to cache intermediate file handles (“home”, “user”), then it must include a GETFH after each LOOKUP.

Metadata (v Full) Journaling

Metadata journaling: Data is written (checkpointed) directly to storage & only metadata is logged to journal

Ordering requirements:

An op can be logged to journal only after data has been written to storage

Aka Ordered Journaling (eg. ext3, ext4)

Pro:

We don't need to log the data, so higher performance than full journaling

Con:

Other forms of latent inconsistencies (not perceptible by fsck)

1. Data could get (over)written before metadata is logged
   Eg. last-modify time may remain inconsistent
2. Data could get overwritten partially–user data corruption!

To disable in ext3/ext4: set mount option **data=journal** (default **data=ordered**)

Redo & Undo Journaling

Recovery always starts from the persisted version of the file system
Redo: roll-forward to ensure committed operations are not lost
Undo: roll-back to reverse any operations that should not be committed

Undo needed only if checkpointing of op's dirty content can start BEFORE the op's entry has been committed to the journal

Early checkpointing allowed for performance reasons

3 pointers in the journal: (1) checkpointed, (2) committed, (3) current.
Redo: entries from (1) to (2)
Undo: entries from (2) to (3)

Undo logging is expensive: requires logging the prior versions (so can be undone)

Not popular in file systems: not a net positive WRT performance
Popular in databases: to undo committed transactions & MVCC

COW File Systems

- No in-place overwrites (except superblock)
  - Guaranteed crash consistency
  - The persisted file system is always intact
  - Each update to the file system is all-or-nothing/atomic
    - Committed only when the new superblock is overwritten
- Benefits:
  - Recovery is simple
    - Take the latest intact version of the persistent file system
    - Replay the committed entries in journal
    - Make another atomic update to the persistent file system
  - Lots of features enabled by snapshots

- Challenges:
  - *Write-amplification* (minor changes require many changes to file system)
    - Amortize that cost by making large updates to the file system
  - Fragmentation
    - Segment-cleaner (from LFS) style work to compact used blocks

Data Integrity: Rare events

Torn Write:
- Each storage device promises some granularity of write atomicity: 512b, 4KiB, etc.
- File system block-size is typically a larger multiple
- How to guard against a torn write?
  - Store a checksum with each block write
  - Recompute checksum on each read and compare
    - If mismatch, reconstruct/repair block using RAID

Lost/Redirected Write
- An entire write (block + checksum) is lost!
  - Subsequent read will not find a checksum mismatch
- How to guard against this?
  - Store some file-system context with each block (eg. inum+offset)
  - If checksum matches but context does not → lost write
    - Reconstruct/repair using RAID

GOOD LUCK!