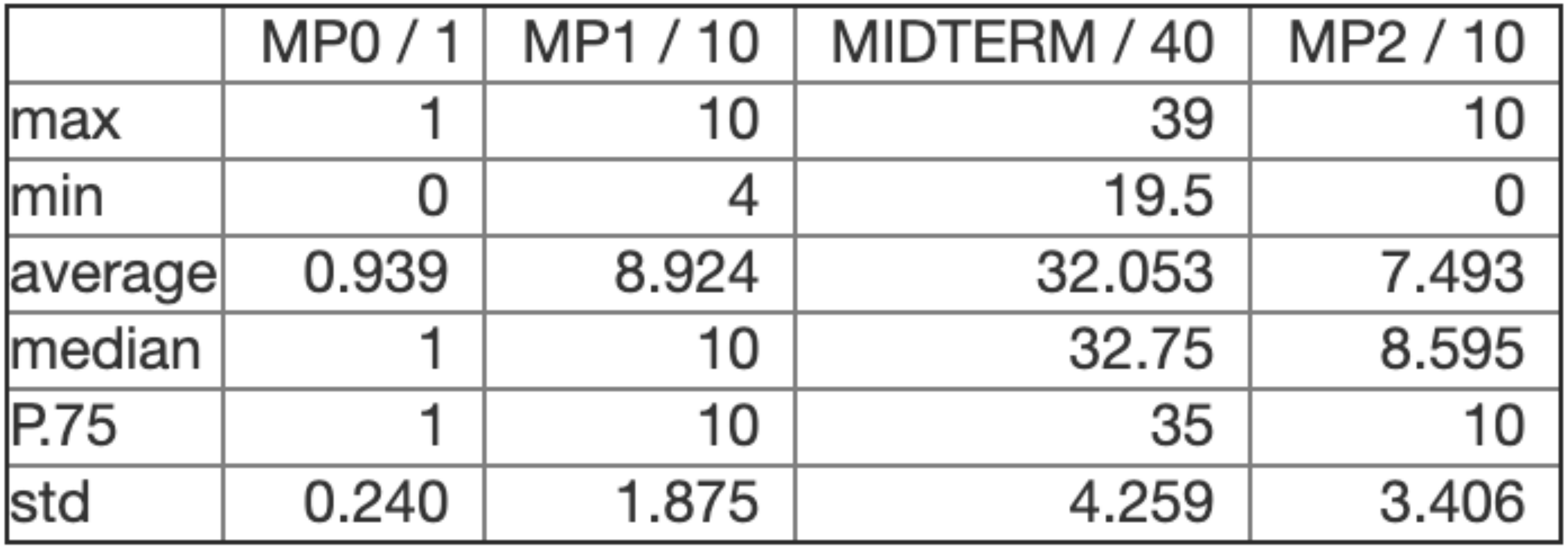
# CS 423
# Operating System Design: Log-Structured File Systems

Professor Tianyin Xu

# MP1/MP2/Midterm Stats



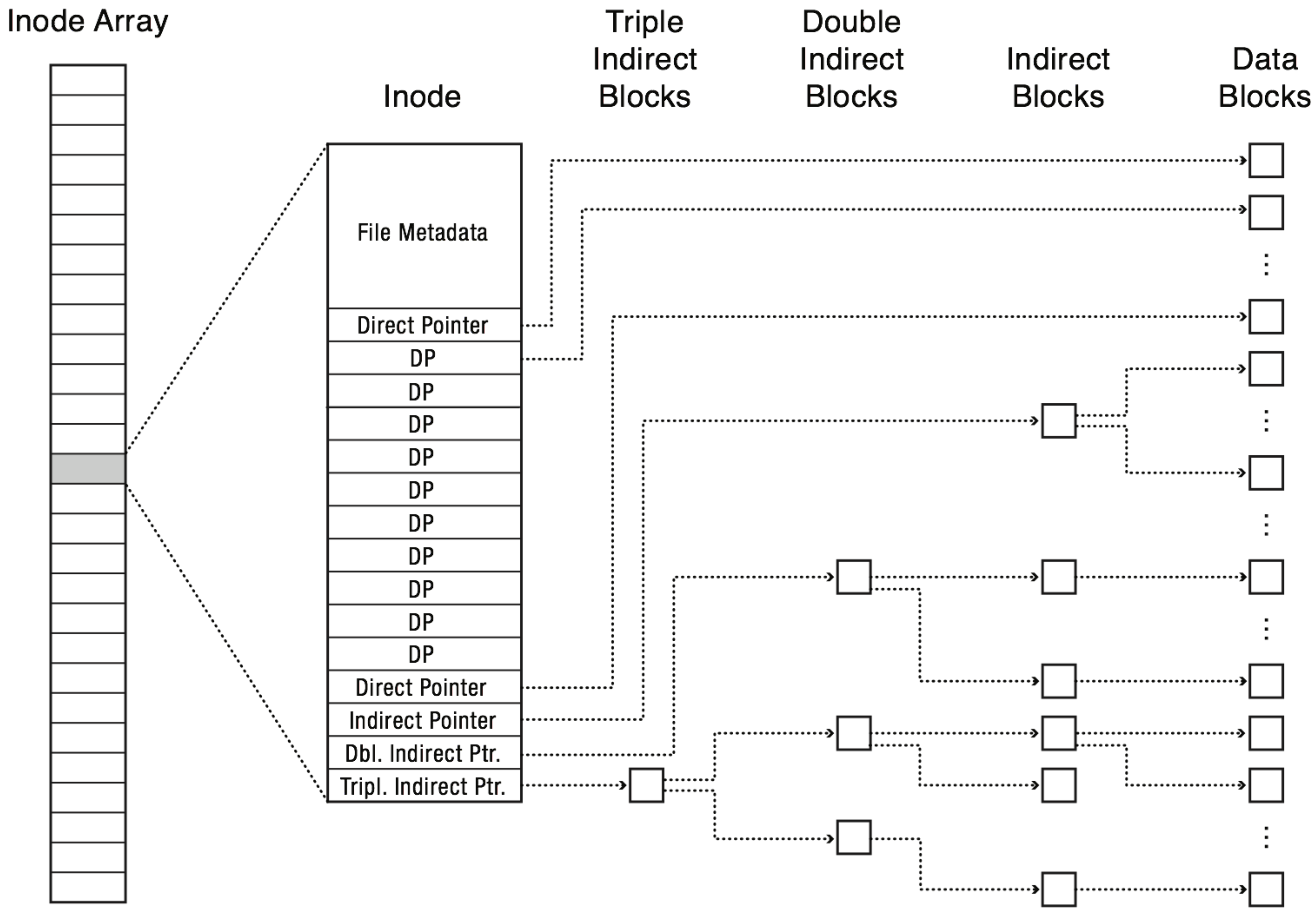
| | MP0 / 1 | MP1 / 10 | MIDTERM / 40 | MP2 / 10 |
|---|---:|---:|---:|---:|
| max | 1 | 10 | 39 | 10 |
| min | 0 | 4 | 19.5 | 0 |
| average | 0.939 | 8.924 | 32.053 | 7.493 |
| median | 1 | 10 | 32.75 | 8.595 |
| P.75 | 1 | 10 | 35 | 10 |
| std | 0.240 | 1.875 | 4.259 | 3.406 |

# Recap

- **File**
- **Disk block**
- **Inode**
  - **To read/write a file, we have to find the inode of the file first.**
- **Sequential reads/writes are MUCH faster than random reads/writes**
  - **Why?**

# Berkeley FFS / UNIX FS

## Alternate figure, same basic idea



# Computers Circa 1991

- Disk bandwidth is improving rapidly
- Computers have more memory (up to 128 MB)
- And, alas, disk seek times are … still dog slow!
  - The overhead was becoming larger now (as the bandwidth is higher..)

- What can we do to solve the problem?

- Why not we always do sequential I/O?

# Thought Experiments

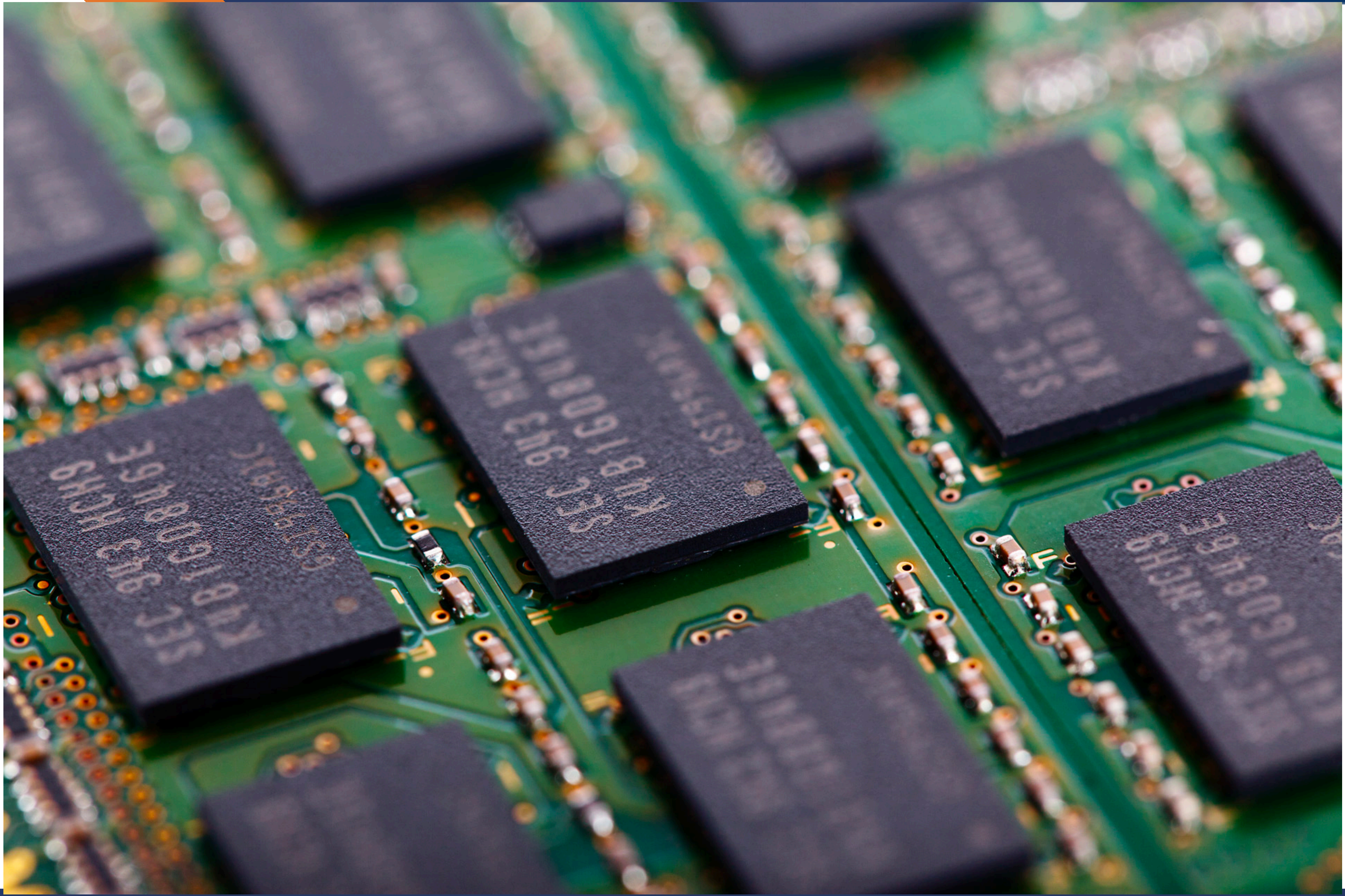
# Let’s do a design

# Debates

## The LFS Flame Wars of the 1990s

- In two papers (one in 1993, one in 1995) Seltzer compared FFS to a BSD port of Sprite's LFS, finding that:
  - LFS is great for workloads with:
    - frequent small writes
    - read patterns that are amenable to hitting in the buffer cache
    - enough idle time for cleaning to run without hurting foreground tasks
  - LFS is not great when:
    - the disk is full (because the cleaner must read many segments just to find a little free space)
    - writes are too random (because dead space will be spread evenly throughout the segments, forcing the cleaner to read many segments to free space)
- Ousterhout (who wrote Sprite LFS) claimed that:
  - BSD LFS was poorly implemented and had performance bugs
  - The benchmarks used to evaluated BSD LFS were unfair (e.g., the compilation benchmark was CPU bound and doesn't provide much insight into file system behavior; the transaction processing workload contains a pathological number of random writes)
  - FFS fragmentation can hurt performance just as much as LFS cleaning

# Flash memory



# Flash memory

- **No need for sequential writes**
  **- just need to find unused blocks**
- **Can do 1->0 rewrites**
  **- Maintain a bitmap of used blocks at fixed block**
- **Lots of complexity**
  **- Bits wear out, read disruption, etc**
  **- Who should deal with those complexity?**