UC San Diego

DSC 102
Systems for Scalable Analytics

Rod Albuyeh

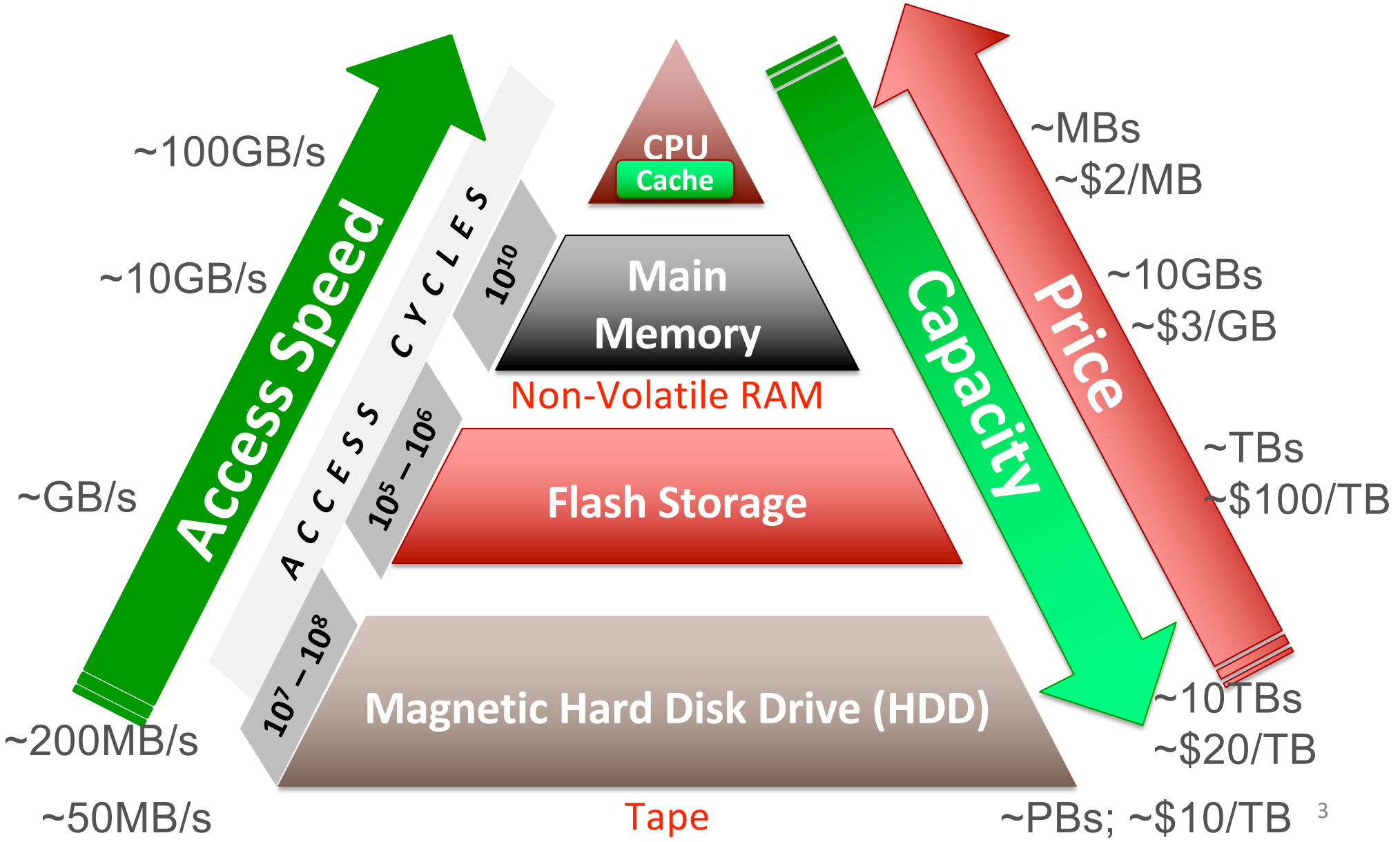
Topic 3: Parallel and Scalable Data Processing

Part 2: Scalable Data Access

Outline

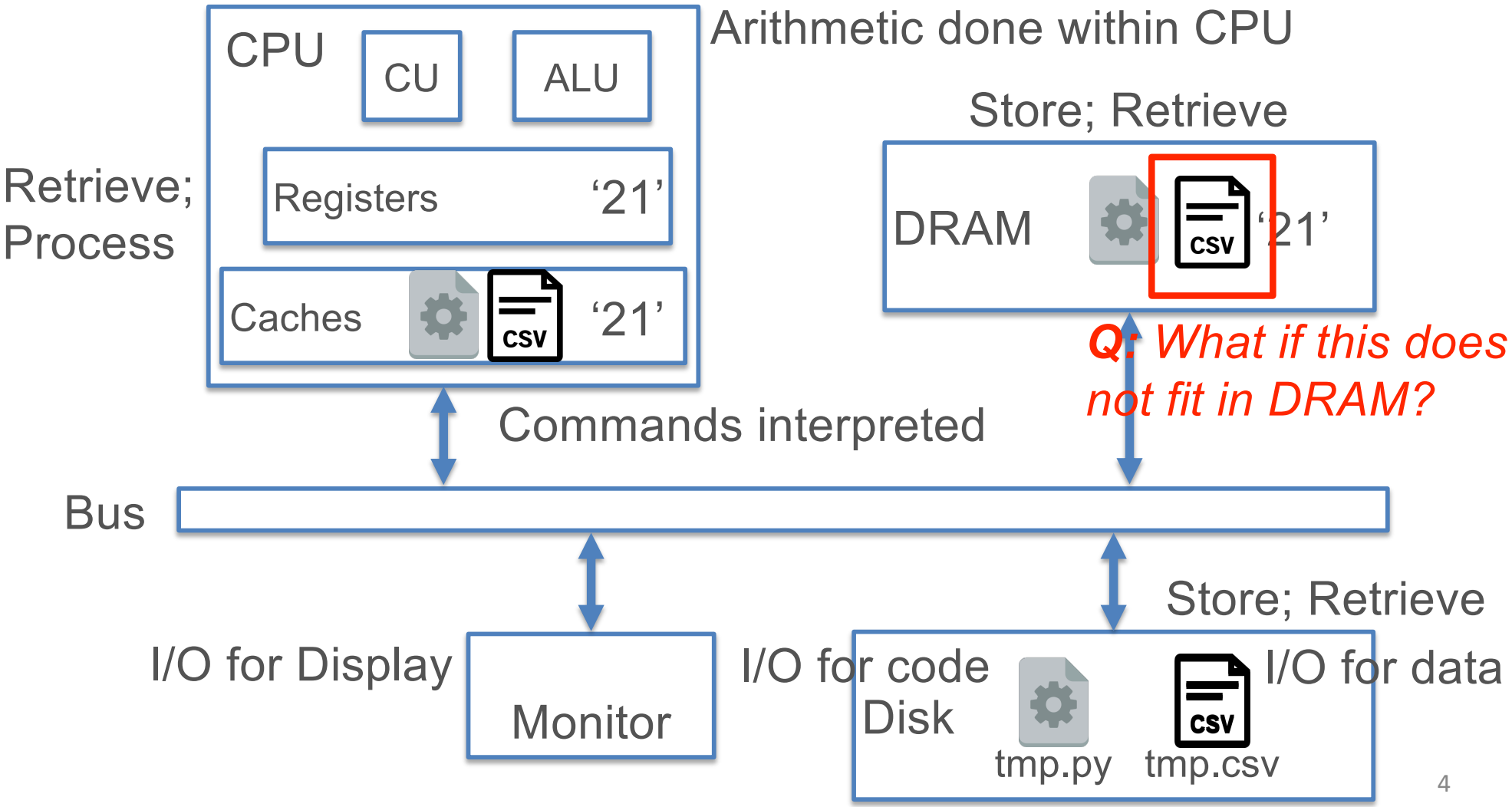
- Basics of Parallelism
  - Task Parallelism; Dask
  - Single-Node Multi-Core; SIMD; Accelerators
- Basics of Scalable Data Access
  - Paged Access; I/O Costs; Layouts/Access Patterns
  - Scaling Data Science Operations
- Data Parallelism: Parallelism + Scalability
  - Data-Parallel Data Science Operations
  - Optimizations and Hybrid Parallelism

Recap: Memory Hierarchy

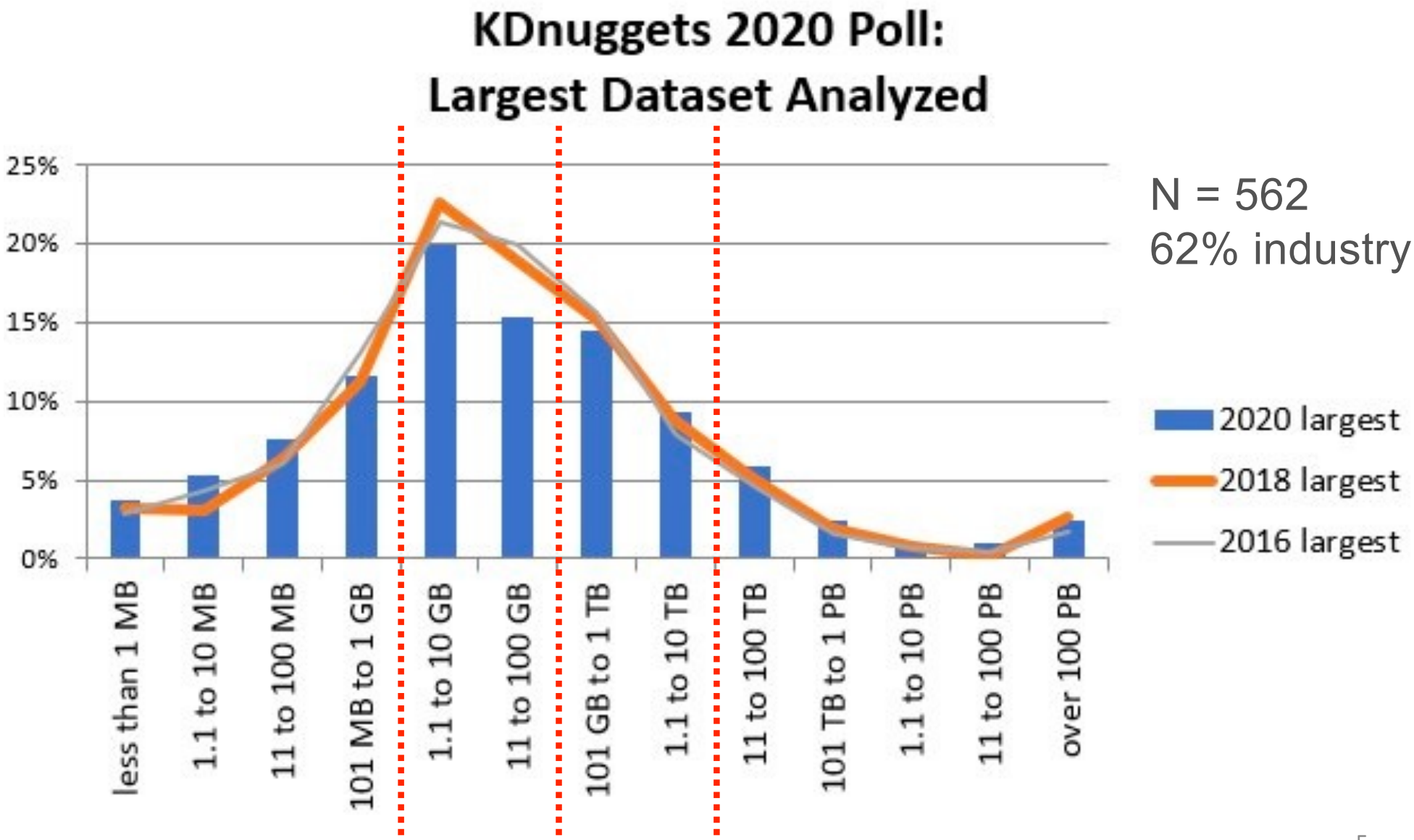


Memory Hierarchy in Action

Rough sequence of events when program is executed



Scale of Datasets in Practice



https://www.kdnuggets.com/2020/07/poll-largest-dataset-analyzed-results.html

How much DRAM might a machine have?

Common DRAM configs:

- Average Laptop: 16GB
- t2.xlarge EC2 instance: 16GB (at $0.19/hour)
- 2023 MacBook Pro: 32GB-96GB
- Consumer Deep Learning / Gaming PC: 128GB ($288 fixed)
- r7g.metal EC2 instance: 512GB (at $3.43/hour)
- hpc6id.32xlarge EC2 instance: 1024GB (at $5.70/hour)

Less common: u-24tb1.112xlarge: 24TB (at $218.40/hour)

Scalable Data Access

Central Issue: Large data file does not fit entirely in DRAM

Basic Idea: Divide-and-conquer again.
"Split" a data file (virtually or physically)
and *stage reads* of its pages from disk to DRAM;
vice versa for writes

4 key regimes of scalability / staging reads:

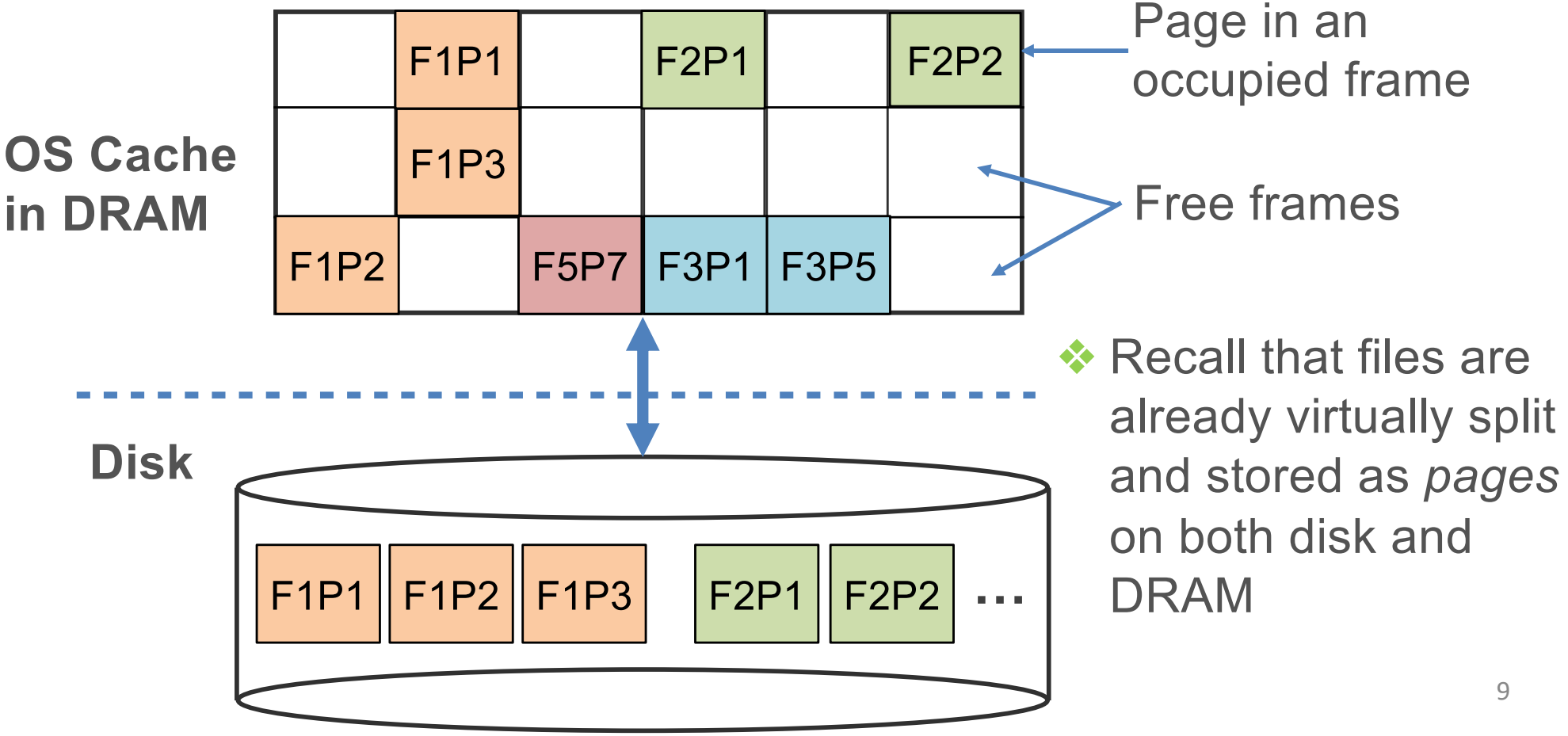
- **Single-node disk**: Paged access from file on local disk
- **Remote read**: Paged access from disk(s) over a network
- **Distributed memory**: Data fits on a cluster's total DRAM
- **Distributed disk**: Use entire memory hierarchy of cluster

Outline

- Basics of Parallelism
  - Task Parallelism; Dask
  - Single-Node Multi-Core; SIMD; Accelerators
- Basics of Scalable Data Access
  - Paged Access; I/O Costs; Layouts/Access Patterns
  - Scaling Data Science Operations
- Data Parallelism: Parallelism + Scalability
  - Data-Parallel Data Science Operations
  - Optimizations and Hybrid Parallelism

Paged Data Access to DRAM

Basic Idea: "Split" data file (virtually or physically)
and *stage reads* of its pages from disk to DRAM (vice versa for writes)



- Recall that files are already virtually split and stored as *pages* on both disk and DRAM

Page Management in DRAM Cache

- **Caching**: Retaining pages read from disk in DRAM
- **Eviction**: Removing a page frame's content in DRAM
- **Spilling**: Writing out pages from DRAM to disk
  - If a page in DRAM is "**dirty**" (i.e., some bytes were written but not backed up on disk), eviction requires a spill.
- The set of DRAM-resident pages typically changes over the lifetime of a process
- **Cache Replacement Policy**: The algorithm that chooses which page frame(s) to evict when a new page has to be cached but the OS cache in DRAM is full
  - Popular policies include Least Recently Used, Most Recently Used, etc. (more shortly)

Quantifying I/O: Disk, Network

- Page reads/writes to/from DRAM from/to disk incur latency
- **Disk I/O Cost**: Abstract counting of number of page I/Os; can map to bytes given page size
- Sometimes, programs read/write data over network
- **Communication/Network I/O Cost**: Abstract counting of number of pages/bytes sent/received over network
- I/O cost is *abstract*; mapping to latency is *hardware-specific*

Example: Suppose a data file is 40GB; page size is 4KB
I/O cost to read file = 10 million page I/Os

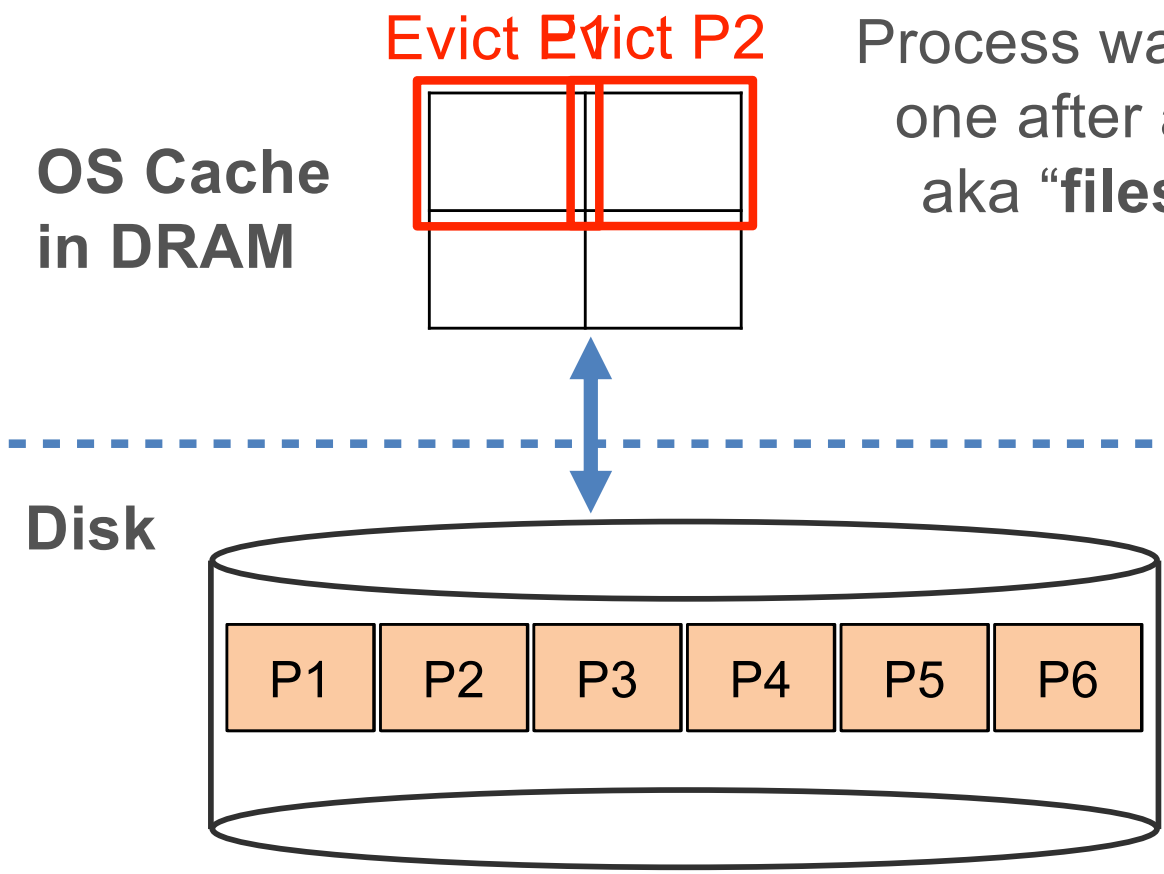
Disk with I/O throughput: 800 MB/s → 40GB/800MBps = 50s

Network with speed: 200 MB/s → 40GB/200MBps = 200s

Scaling to (Local) Disk

Basic Idea: Split data file (virtually or physically) and *stage reads* of its pages from disk to DRAM (vice versa for writes)

Suppose OS Cache has only 4 frames; initially empty



Process wants to read file's pages one after another, then discard: aka "**filescan**" access pattern

Read P1
Read P2
Read P3
Read P4
Read P5
Read P6

Cache is full! Cache Repl. needed

Total I/O cost: **6**

Scaling to (Local) Disk

- In general, *scalable programs stage access* to pages of file on disk and efficiently use available DRAM
  - Recall that typically DRAM size << Disk size
- Modern DRAM sizes can be 10s of GBs; so we read a "chunk"/"block" of file at a time (say, 1000s of pages)
  - On magnetic hard disks, such chunking leads to *more sequential I/Os*, raising throughput and lowering latency!
  - Similarly, write a chunk of dirtied pages at a time

Generic Cache Replacement Policies

***Q:** What to do if number of page frames is too few for file?*

- **Cache Replacement Policy**: Algorithm to decide which page frame(s) to evict to make space
  - Typical frame ranking criteria:
    - recency of use
    - frequency of use
    - number of processes reading it
  - Typical optimization goal: Reduce total page I/O costs
- A few well-known policies:
  - **Least Recently Used (LRU)**: Evict page that was used longest ago
  - **Most Recently Used (MRU)**: (Opposite of LRU)
  - ML-based caching policies are “hot” nowadays!

Ad: Take CSE 132C for more on cache replacement policies

Data Layouts and Access Patterns

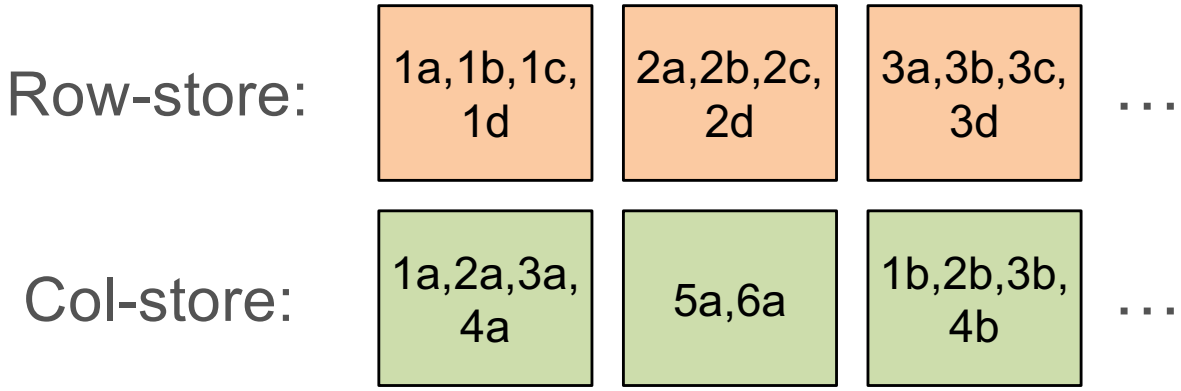
- Recall that *data layouts* and *data access patterns* affect what data subset gets cached in higher level of memory hierarchy
  - Recall matrix multiplication example and CPU caches
- **Key Principle:** Optimizing layout of data file on disk based on data access pattern can help reduce I/O costs
  - Applies to both magnetic hard disk and flash SSDs
    - But especially critical for magnetic hard disks due to vast differences in latency of random vs sequential access!

Row-store vs Column-store Layouts

- A common dichotomy when serializing 2-D structured data (relations, matrices, DataFrames) to file on disk

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Say, a page can fit only 4 cell values

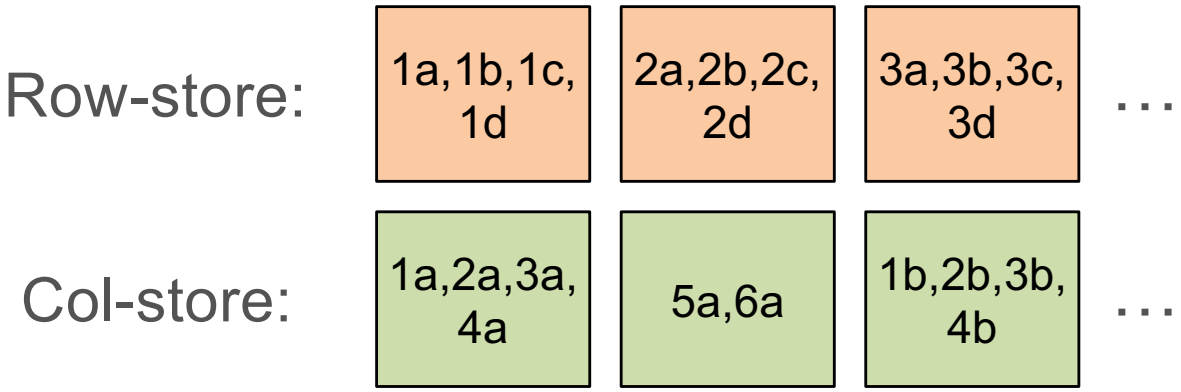


- Based on data access pattern of program, I/O costs with row- vs col-store can be orders of magnitude apart!

Row-store vs Column-store Layouts

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Say, a page can fit only 4 cell values



***Q:** What is the I/O cost with each to compute, say, a sum over B?*

- With row-store: need to fetch *all* pages; I/O cost: 6 pages
- With col-store: need to fetch only B's pages; I/O cost: 2 pages

This difference generalizes to higher dimensions for tensors

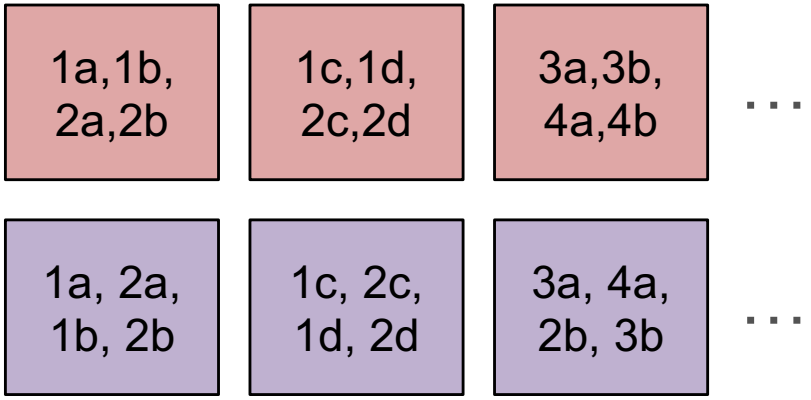
Hybrid/Tiled/"Blocked" Layouts

- Sometimes, it is beneficial to do a hybrid, especially for analytical RDBMSs and matrix/tensor processing systems

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Say, a page can fit only 4 cell values

Hybrid stores with 2x2 tiled layout:

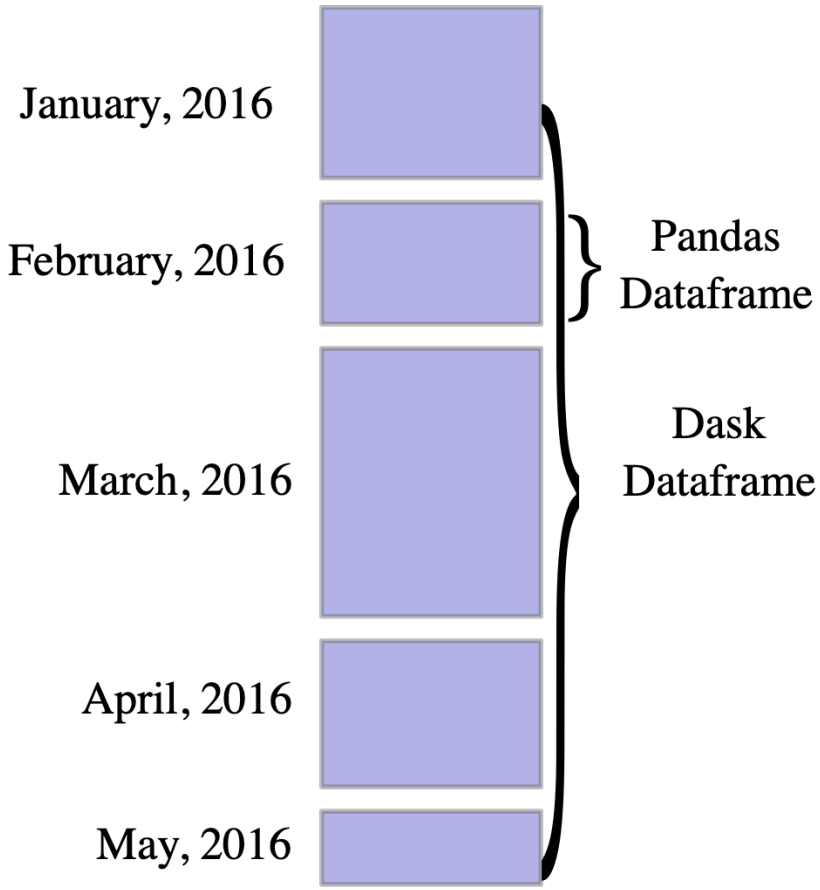


Key Principle: What data layout will yield lower I/O costs (row vs col vs tiled) depends on data access pattern of the program!

Example: Dask's DataFrame

Basic Idea: Split data file (virtually or physically) and *stage reads* of its pages from disk to DRAM (vice versa for writes)

- Dask DF scales to disk-resident data via a row-store
- "Virtual" split: each split is a Pandas DF under the hood
- Dask API is a "wrapper" around Pandas API to scale ops to splits and put all results together
- If file is too large for DRAM, need manual *repartition()* to get physically smaller splits (< ~1GB)

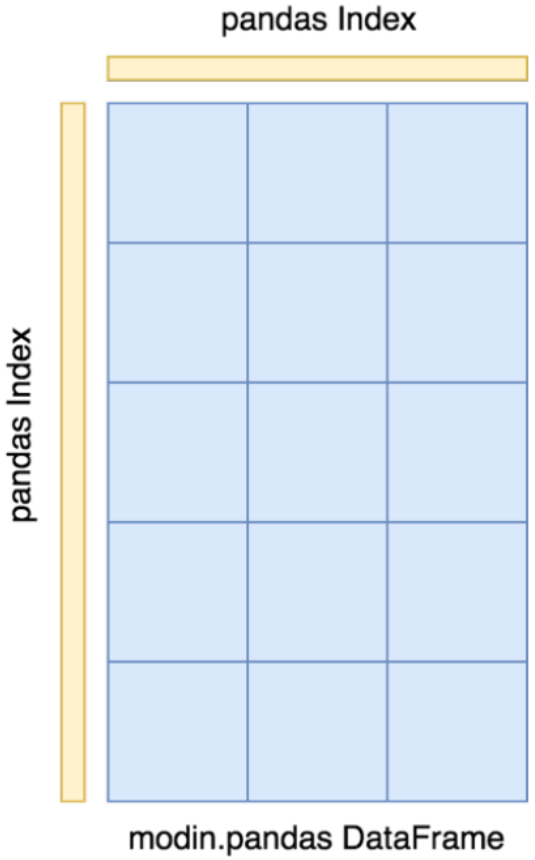


https://docs.dask.org/en/latest/dataframe-best-practices.html#repartition-to-reduce-overhead

Example: Modin's DataFrame

Basic Idea: Split data file (virtually or physically) and *stage reads* of its pages from disk to DRAM (vice versa for writes)

- Modin's DF aims to scale to disk-resident data via a tiled store
- Enables seamless scaling along *both* dimensions
- Easier use of multi-core parallelism
  - Many in-memory RDBMSs had this, e.g., SAP HANA, Oracle TimesTen
  - ScaLAPACK had this for matrices



https://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-191.pdf

Scaling with Remote Reads

Basic Idea: Split data file (virtually or physically) and *stage reads* of its pages from disk to DRAM (vice versa for writes)

- Similar to scaling to local disk but not "local":
  - Stage page reads from remote disk/disks over the network (e.g., from S3)
- *More restrictive* than scaling with local disk, since spilling is not possible or requires costly network I/Os
  - OK for a *one-shot* filescan access pattern
  - Use DRAM to cache; repl. policies
  - Can also use smaller local disk as cache (done in PA1)

Outline

- Basics of Parallelism
  - Task Parallelism; Dask
  - Single-Node Multi-Core; SIMD; Accelerators
- Basics of Scalable Data Access
  - Paged Access; I/O Costs; Layouts/Access Patterns
  - Scaling Data Science Operations
- Data Parallelism: Parallelism + Scalability
  - Data-Parallel Data Science Operations
  - Optimizations and Hybrid Parallelism