BOSTON UNIVERSITY

CS 561: **Data Systems Architectures**

class 14

Data Systems on Modern Hardware:
Multi-cores, Solid-State Drives, and Non-Volatile Memories

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/

Google's Spanner Database

4/3/2025

Ben Vandiver
Senior Staff Software Engineer
Google
website

OSDB: Exposing the Operating System's Inner Database
4/10/2025

George Neville-Neil
Researcher
Yale University
website

Cloud Data Lakes

4/17/2025

Andrew Lamb
Staff Engineer
InfluxData
website

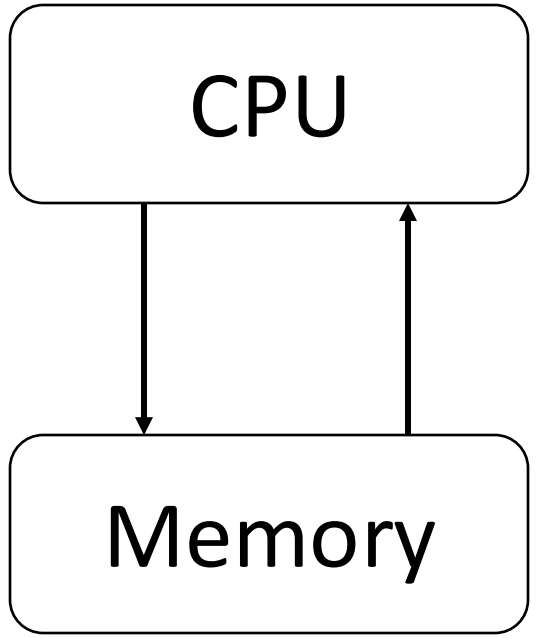
BOSTON UNIVERSITY

More Reminders

- Meet with your mentor by the end of this week (if not already).
  - This is a graded meeting
- For systems projects, meet with Teona
- For other projects with the designated mentor
- **Must** attend the discussion on Fri 3/21
  - Short Progress Updare
- Submit the midterm project report (due on 03/22)

Compute, Memory, and Storage Hierarchy

Traditional von-Neuman computer architecture



(i) assumes CPU is fast enough (for our applications)

(ii) assumes memory can keep-up with CPU and can hold all data

is this the case?

for (i): applications increasingly complex, higher CPU demand
is the CPU going to be always fast enough?

Moore's law

Often expressed as:

"*X* doubles every 18-24 months"

where X is:

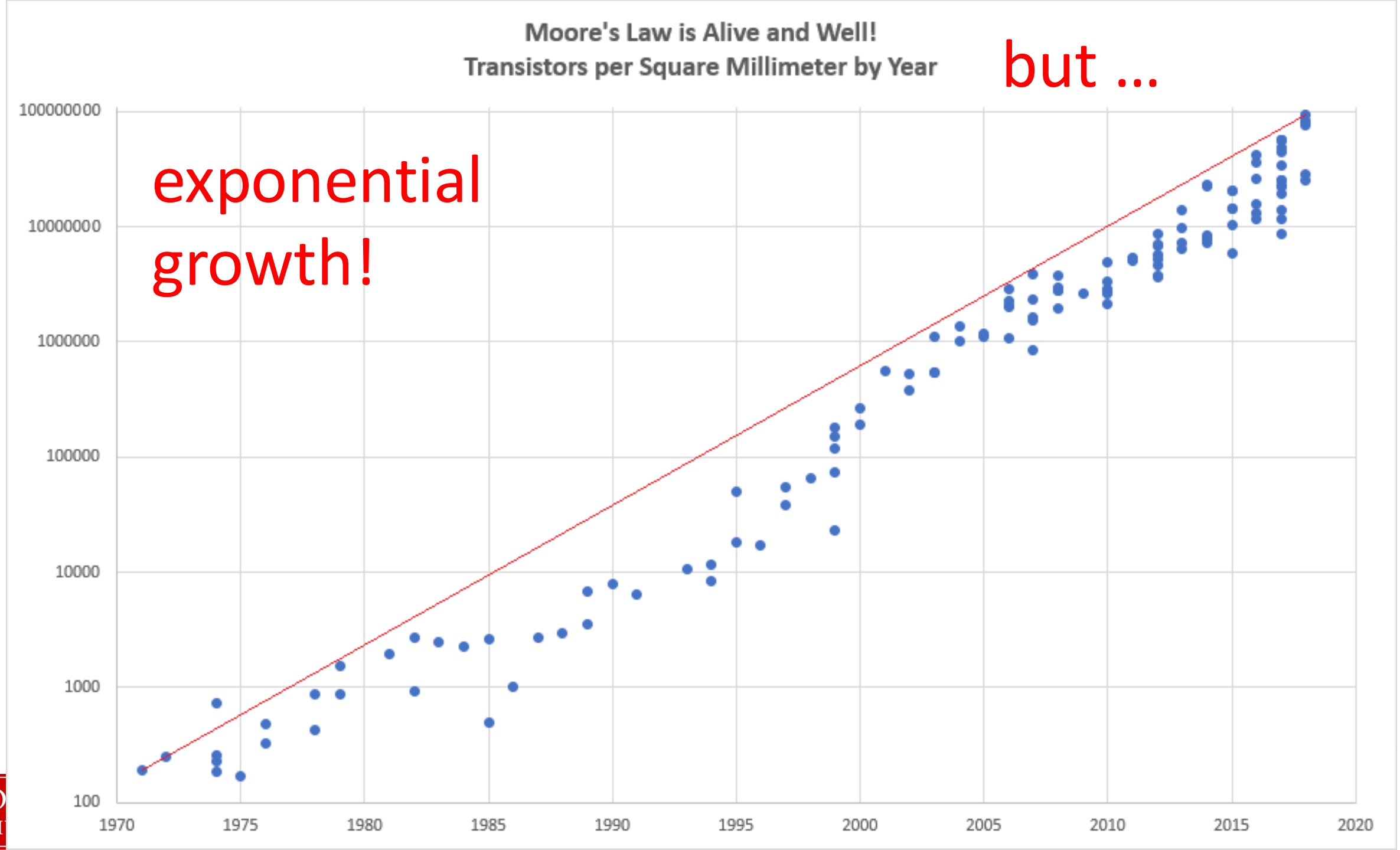
"performance"

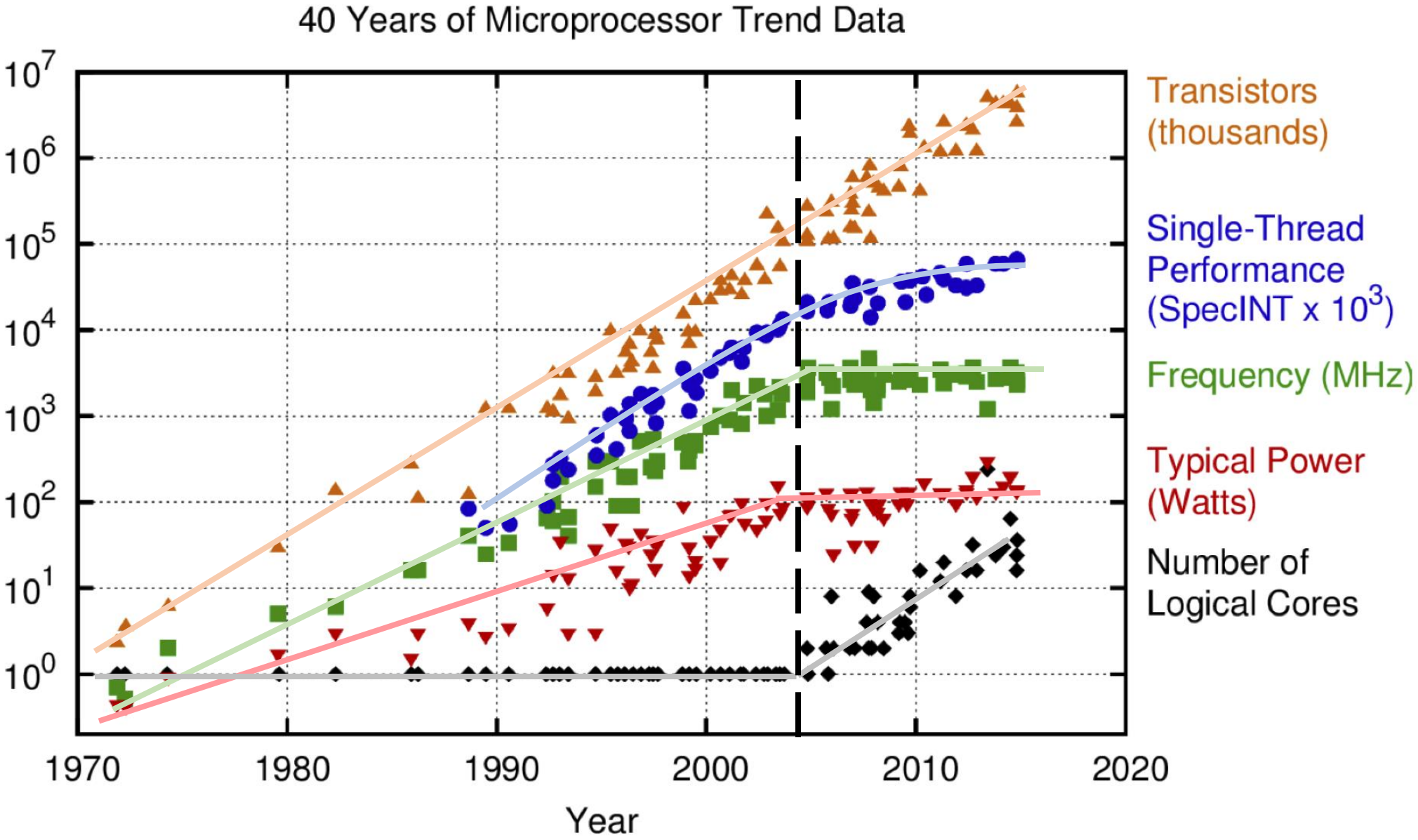
CPU clock speed

the number of transistors per chip

which one is it?

based on William Gropp's slides


Moore's Law is Alive and Well!
Transistors per Square Millimeter by Year
but …
exponential growth!
100000000
10000000
1000000
100000
10000
1000
100
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Can (a single) CPU cope with increasing application complexity?

No, because CPUs (cores) are **not** getting faster!!!

.. but they are getting more and more (**higher parallelism**)

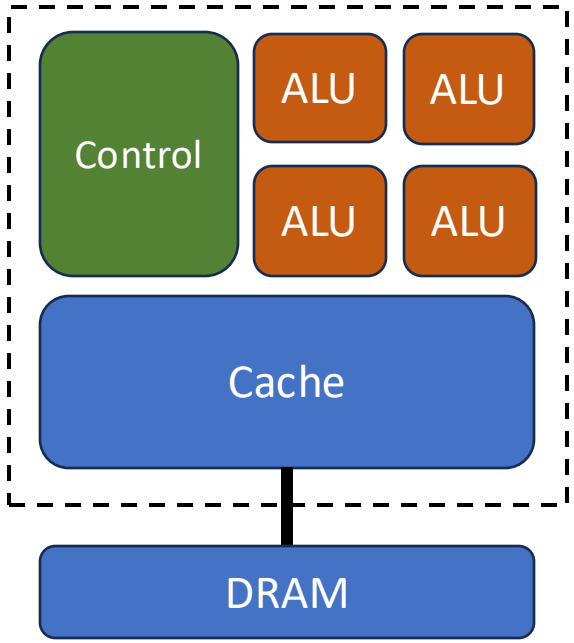
Research Challenges

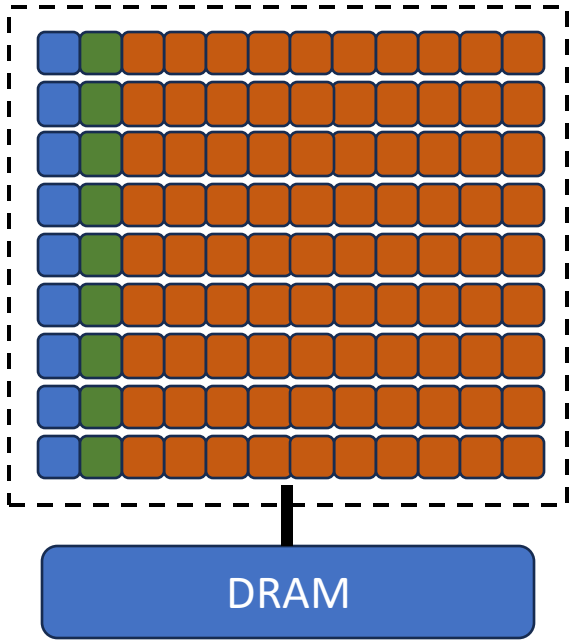
how to handle them?

how to parallel program?

CPU vs. GPU

Why did GPU evolve? What is their difference?

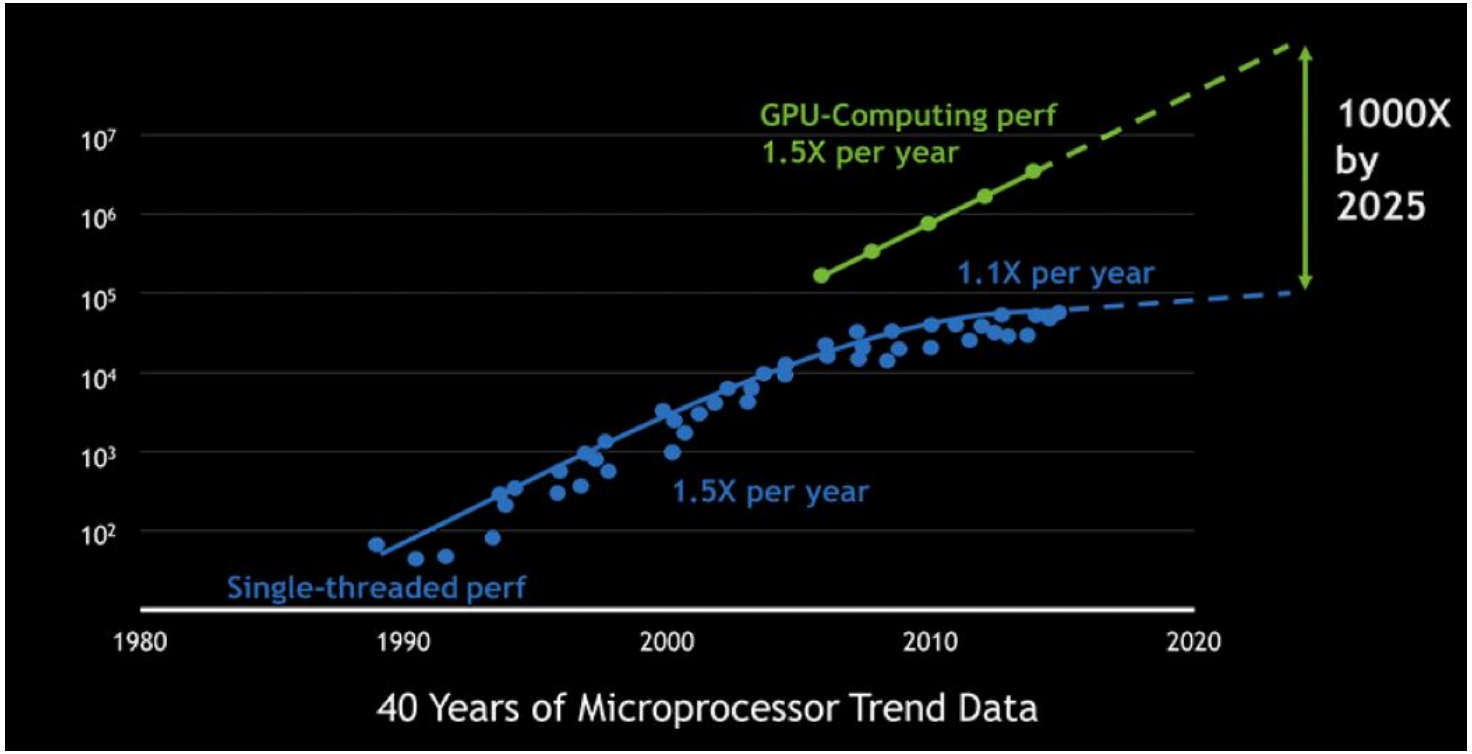




CPU: A few powerful cores with large caches!
Optimized for ***sequential*** computation.

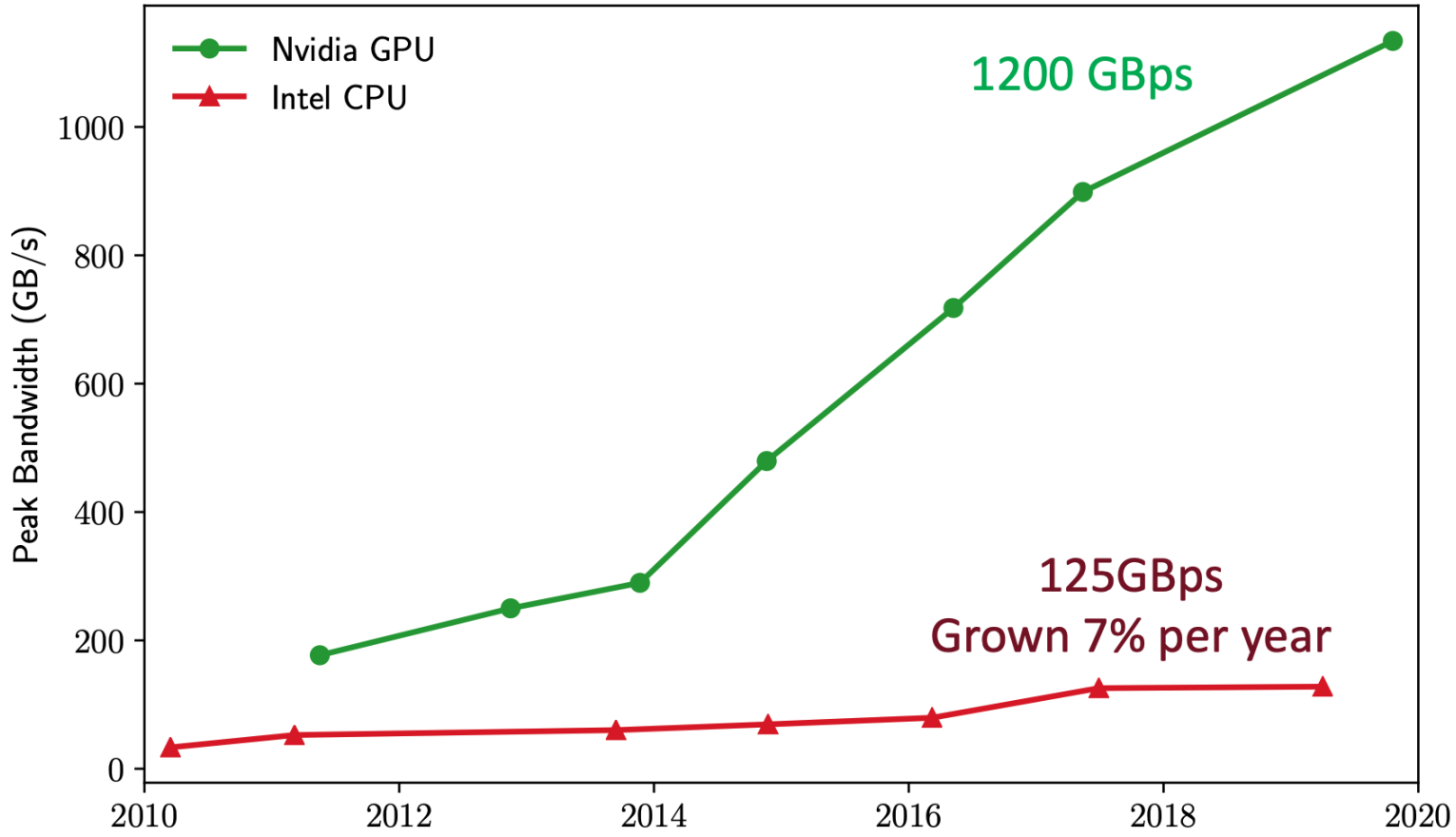
GPU: Many more small cores.
Optimized for ***parallel*** computation.

CPU vs. GPU



	Throughput	Power	Throughput/Power
Intel Skylake	128 GFLOPS/4 Cores	100+ Watts	~1 GFLOP/Watt
NVIDIA V100	15 TFLOPS	200+ Watts	~75 GFLOP/Watt

CPU vs. GPU: Peak Memory Bandwidth



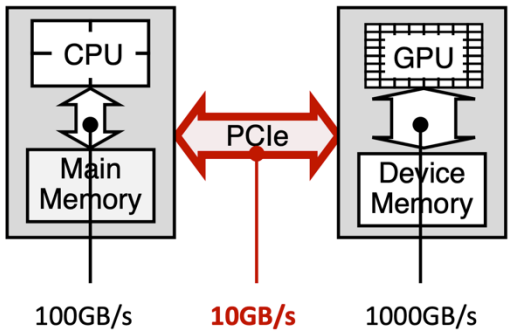
GPU has one order of magnitude higher memory bandwidth than CPU

Memory Bandwidth is the bottleneck for in-memory analytics

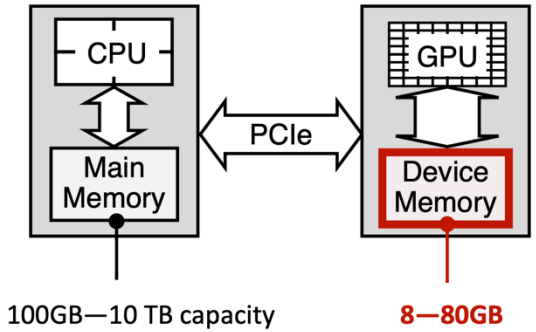
Research Path: **use GPUs for data analytics**

GPU Challenges

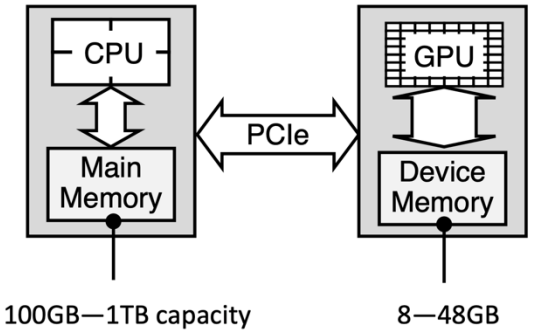
Limitation 1
Low interconnect bandwidth



Limitation 2
Small GPU memory capacity



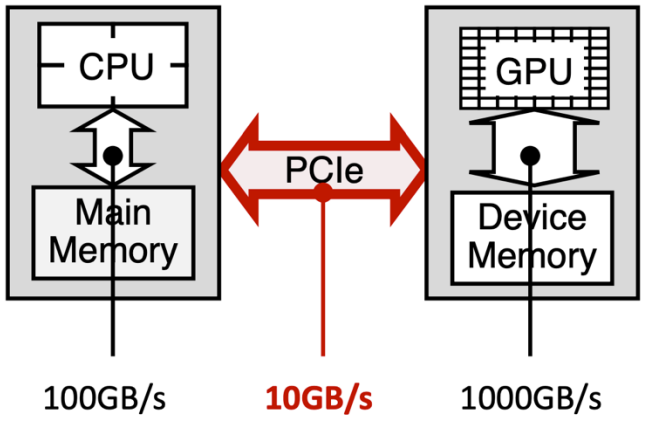
Limitation 3
Coarse-grained cooperation of CPU and GPU

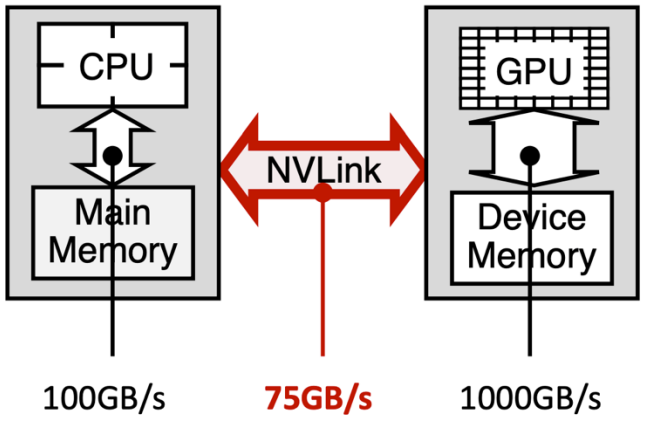


Coprocessor mode: Every query loads data from CPU memory to GPU
GPU-only mode: Store working set in GPU memory and run the entire query on GPU

[With info from Xiangyao Yu]

Emerging GPU Fast Interconnects





Fast Interconnect can solve the PCIe bottleneck

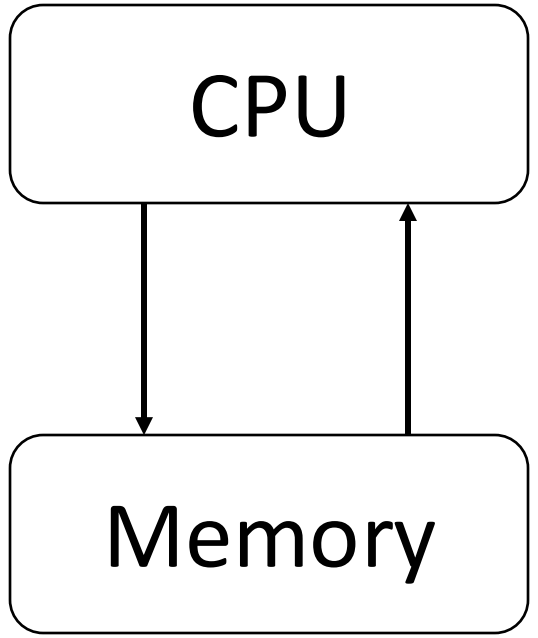
Research Challenges

how to use GPUs? how to program them?

which operations are amenable to high parallel execution?

Compute, Memory, and Storage Hierarchy

Traditional von-Neuman computer architecture



(i) assumes CPU is fast enough (for our applications)

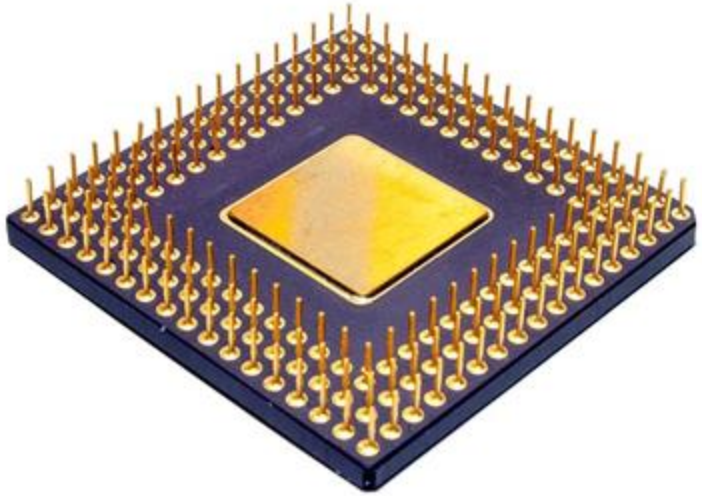
not always!

(ii) assumes memory can keep-up with CPU and can hold all data

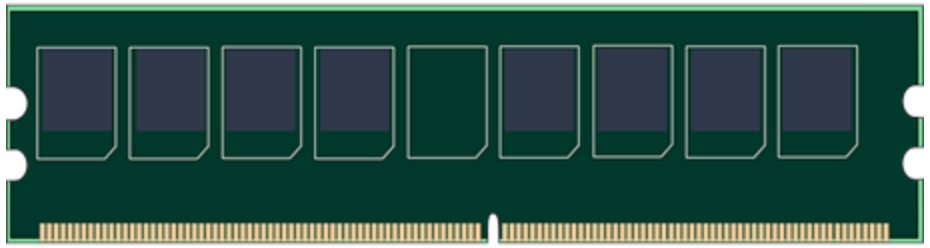
is this the case?

for (ii): is memory faster than CPU (to deliver data in time)?

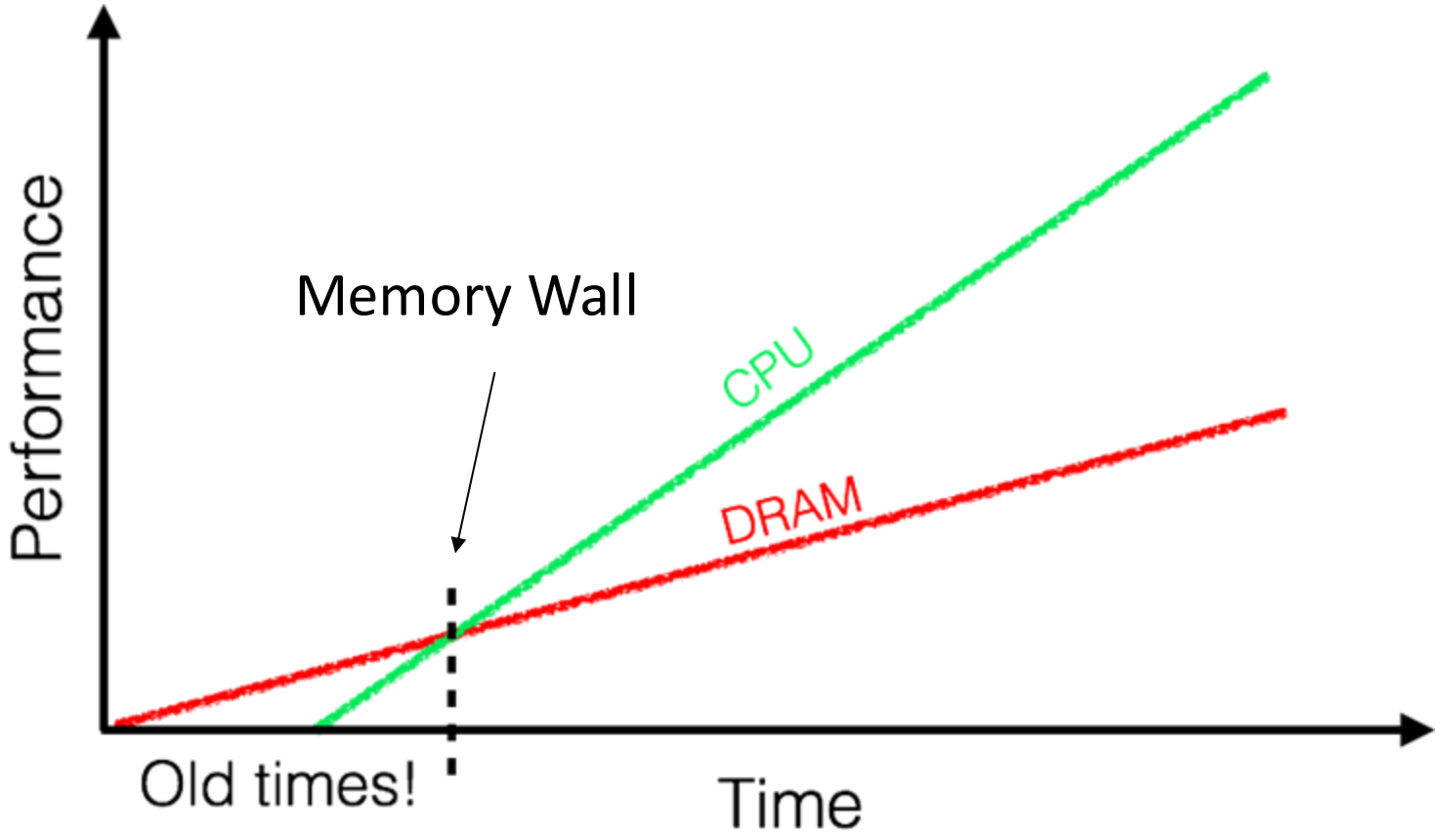
does it have enough capacity?



?



Which one is faster?

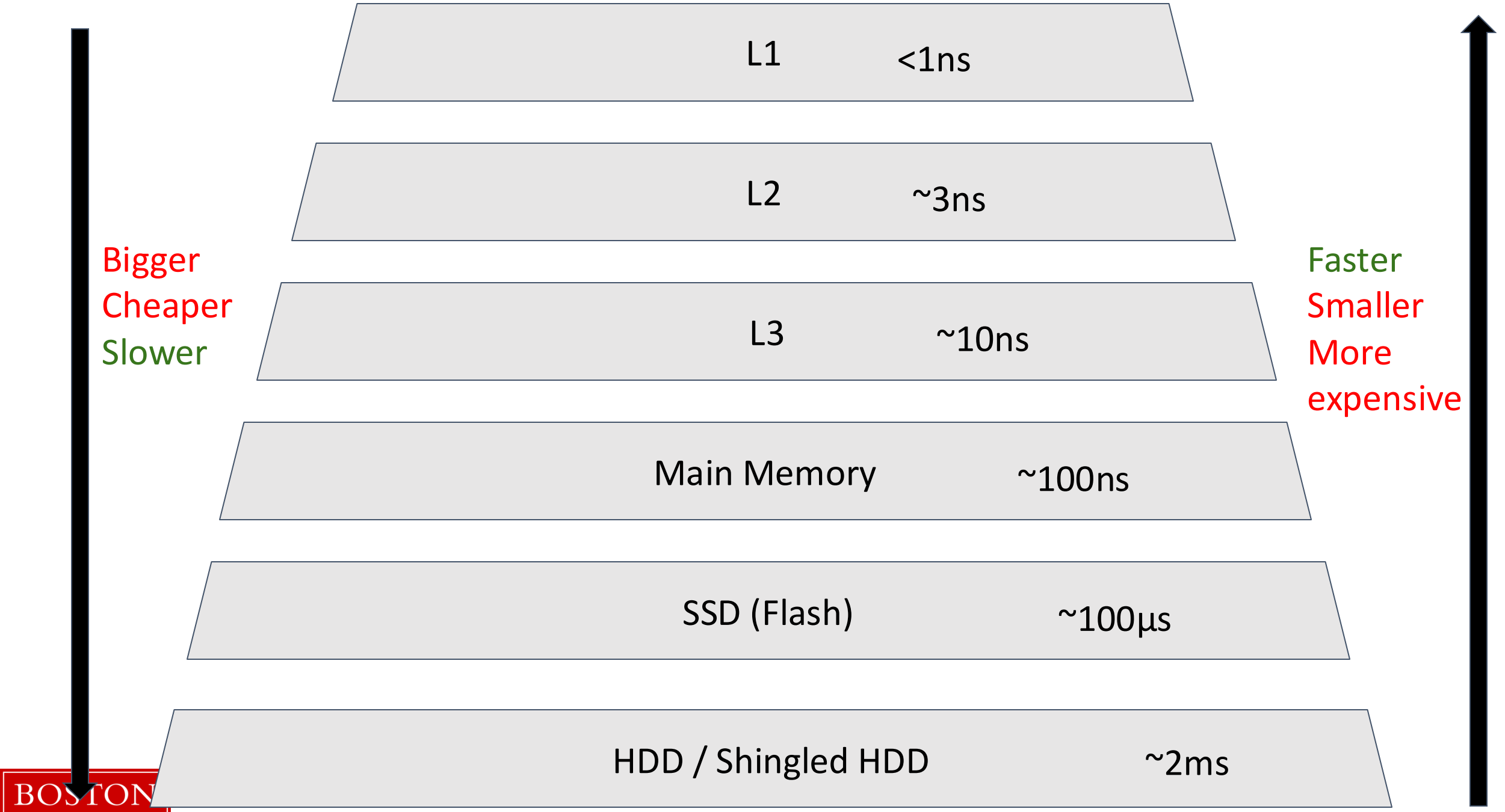


As the gap grows, we need a *deep* ***memory hierarchy***

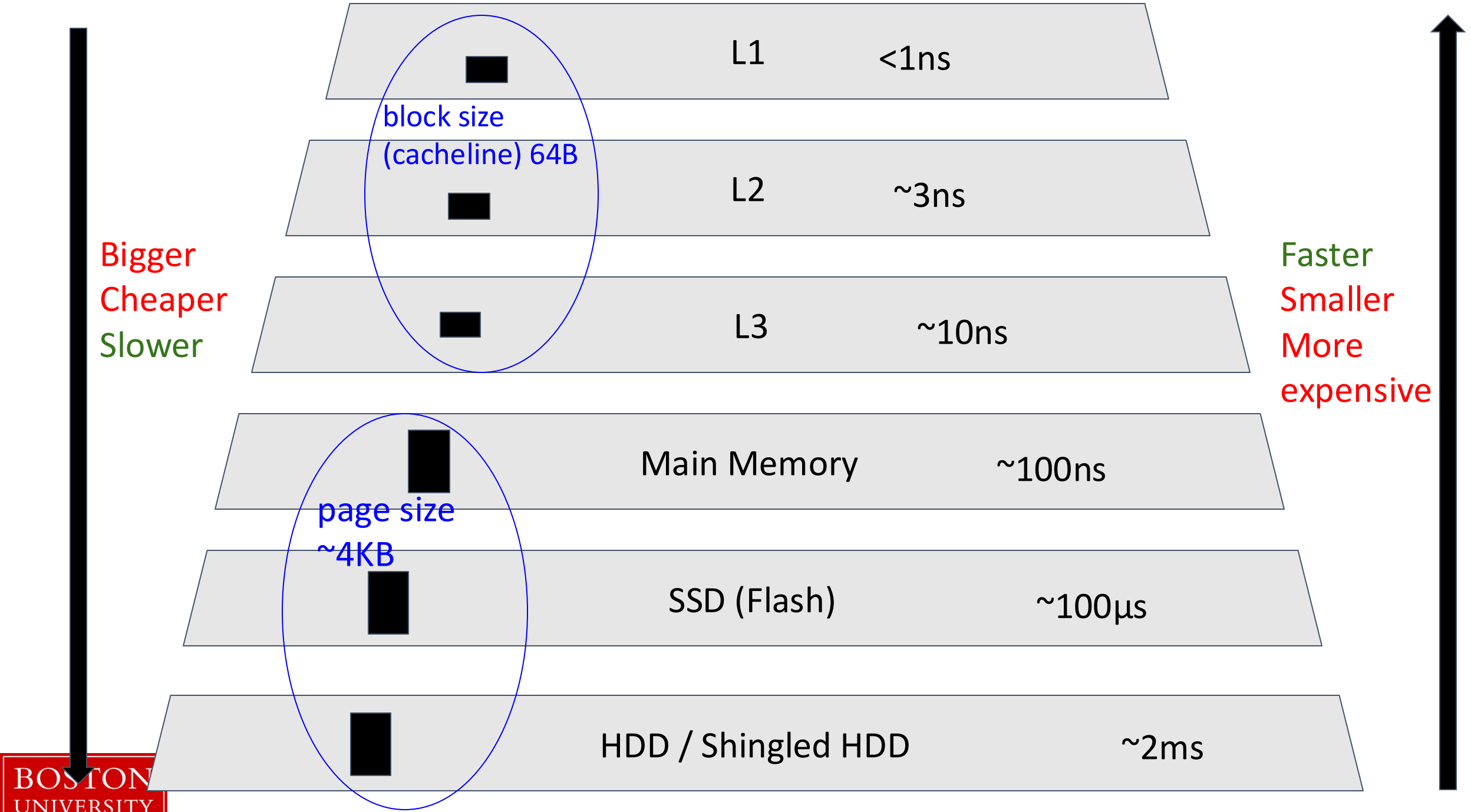
A single level of main memory is not enough

We need a ***memory hierarchy***

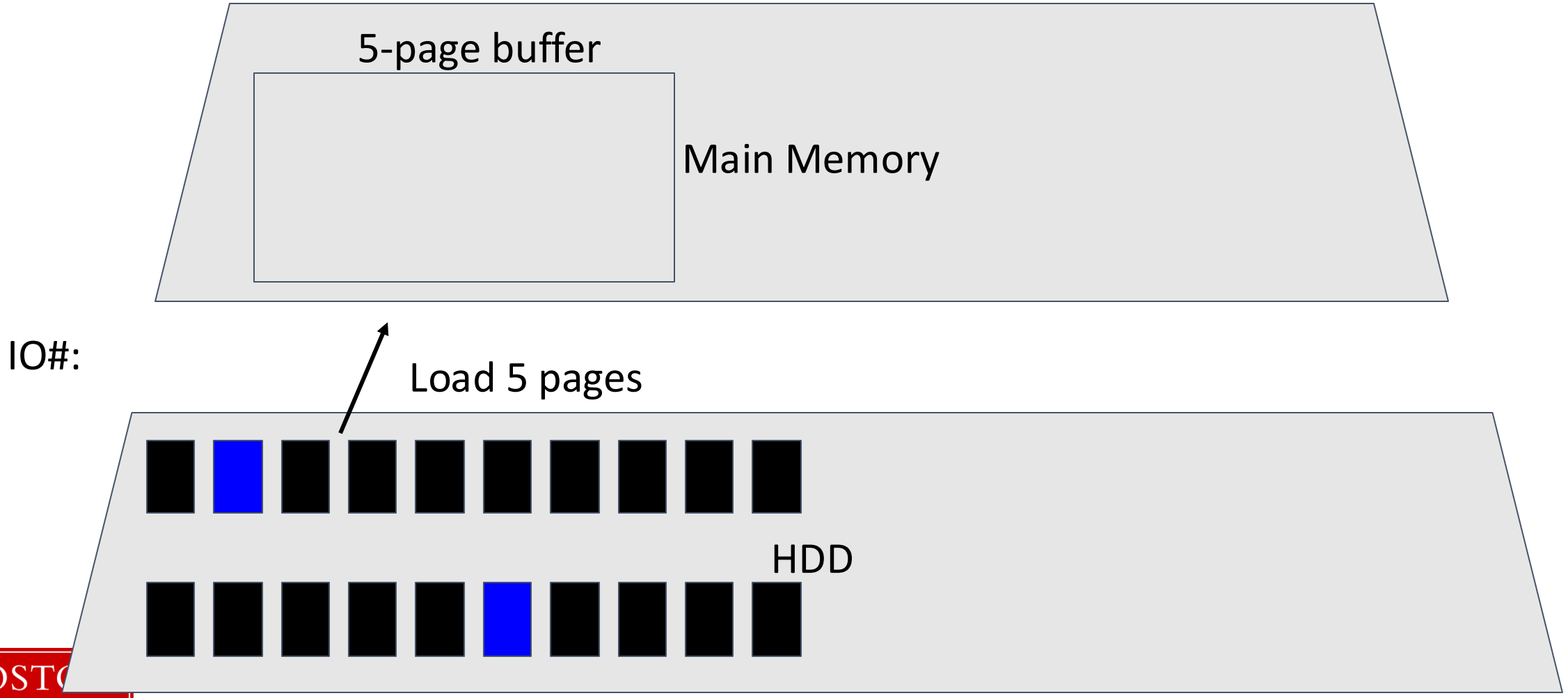
What is the memory hierarchy ?



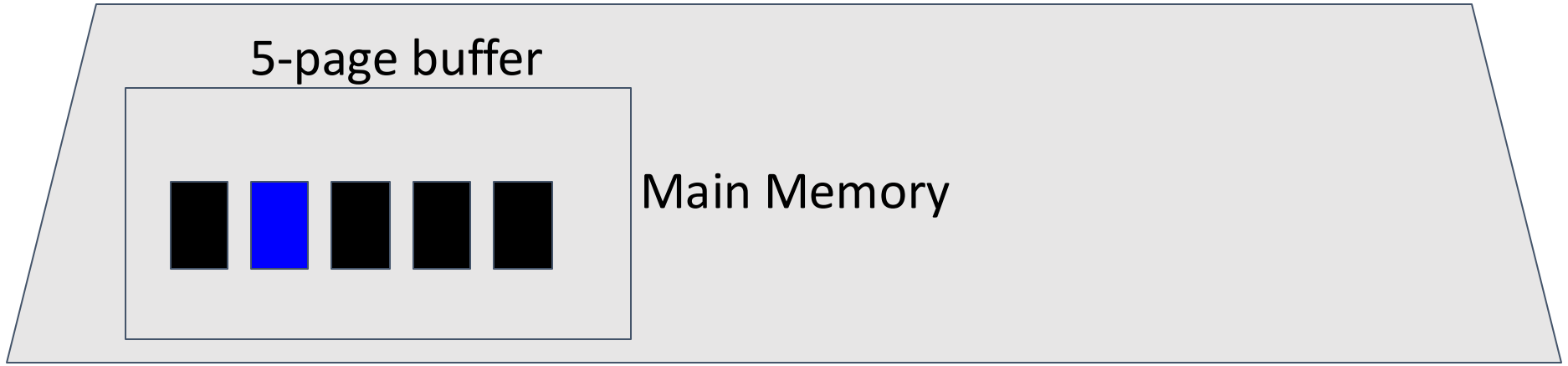
Access Granularity



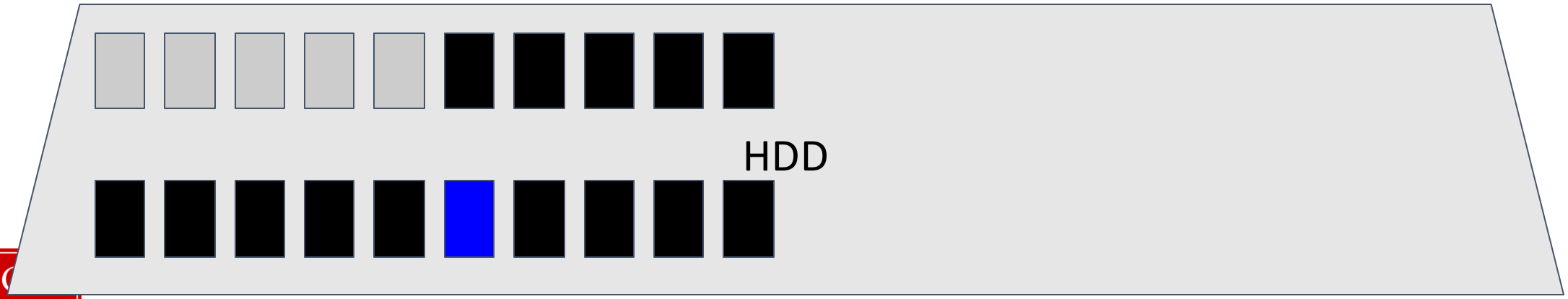
IO cost: Scanning a relation to select 10%



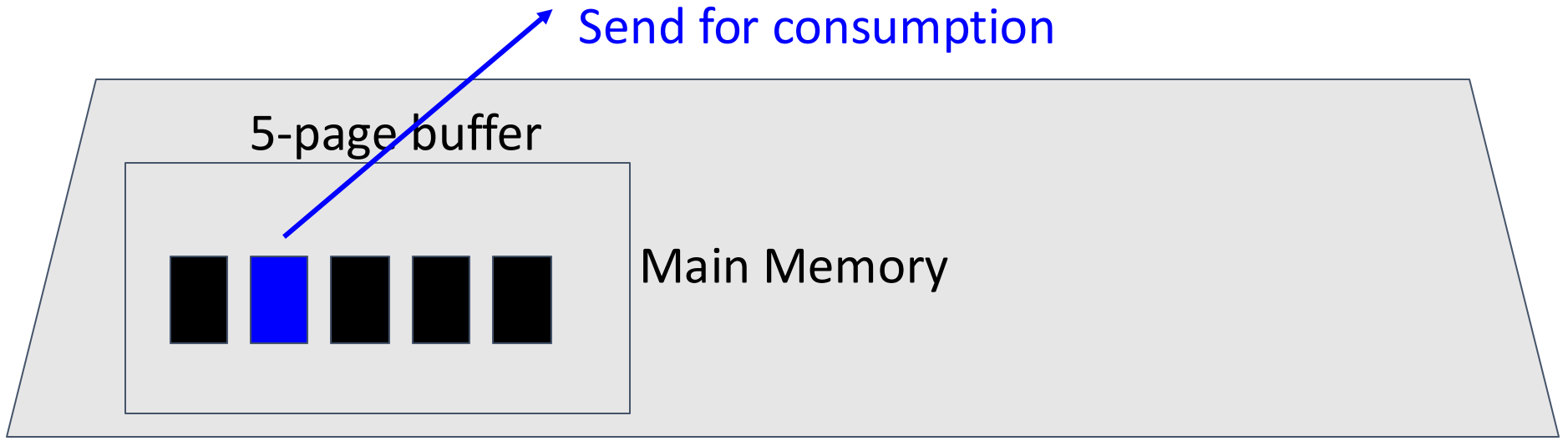
IO cost: Scanning a relation to select 10%



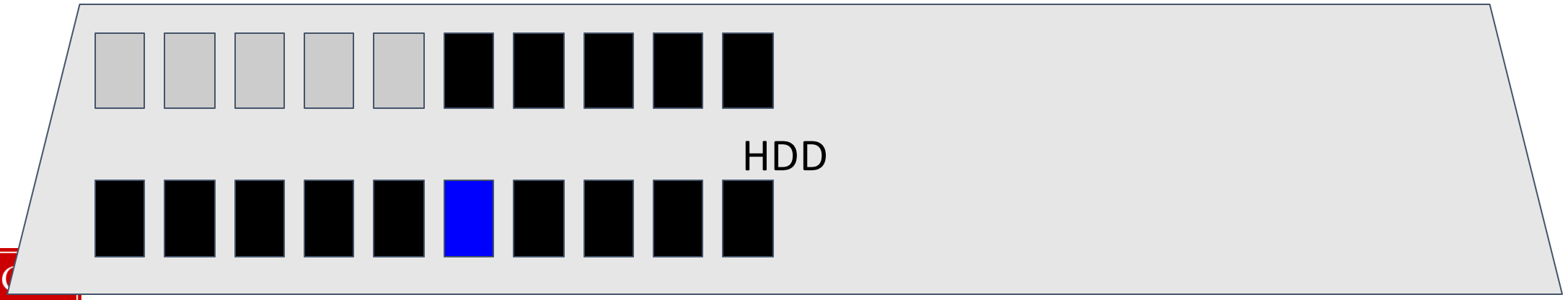
IO#: 5



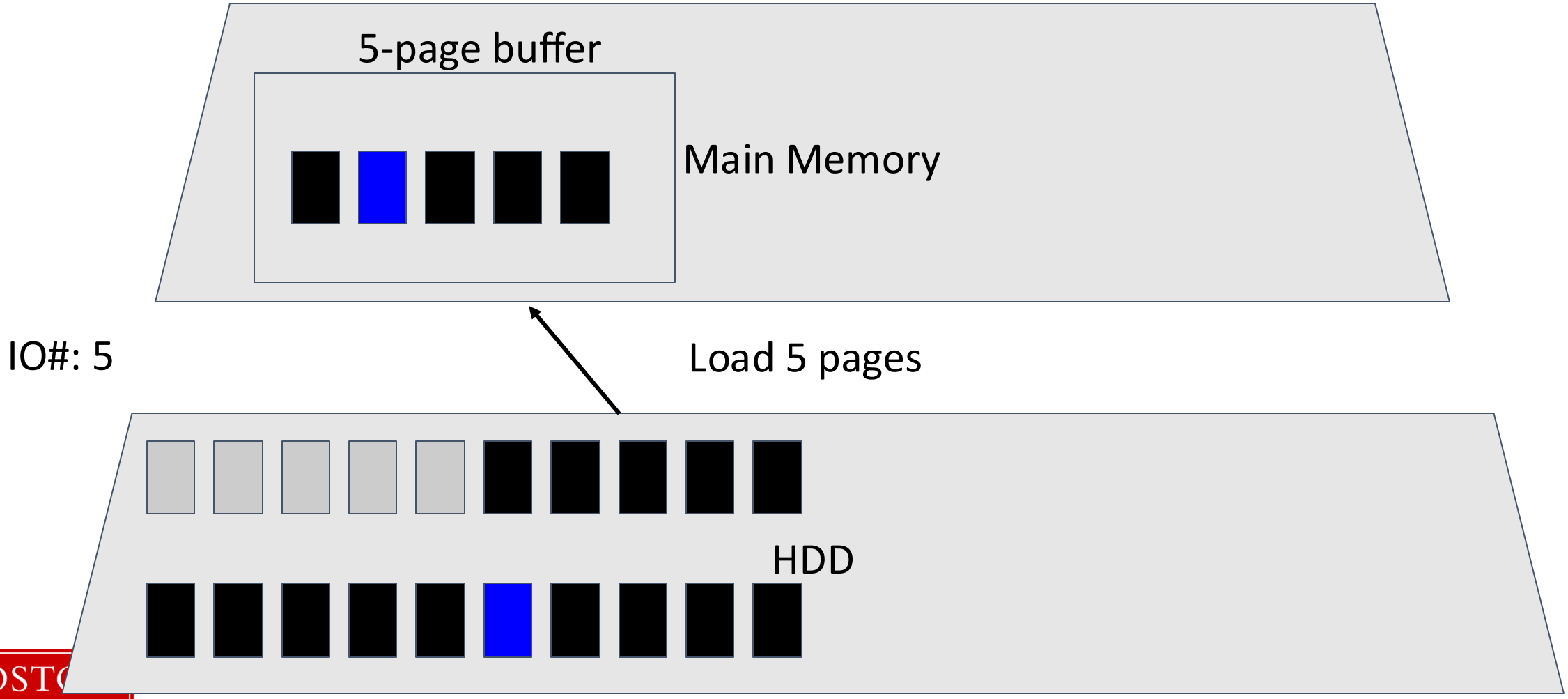
IO cost: Scanning a relation to select 10%



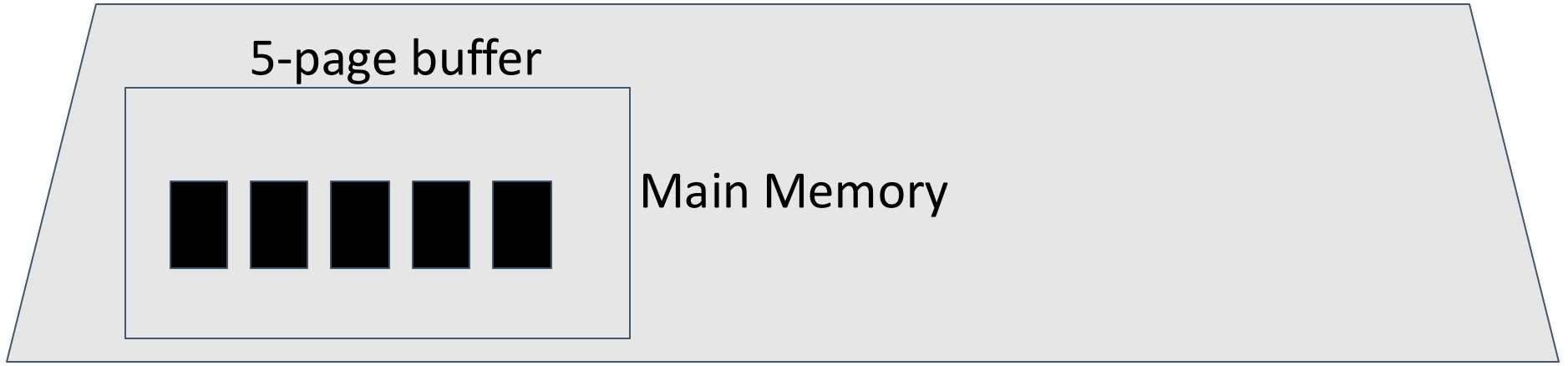
IO#: 5



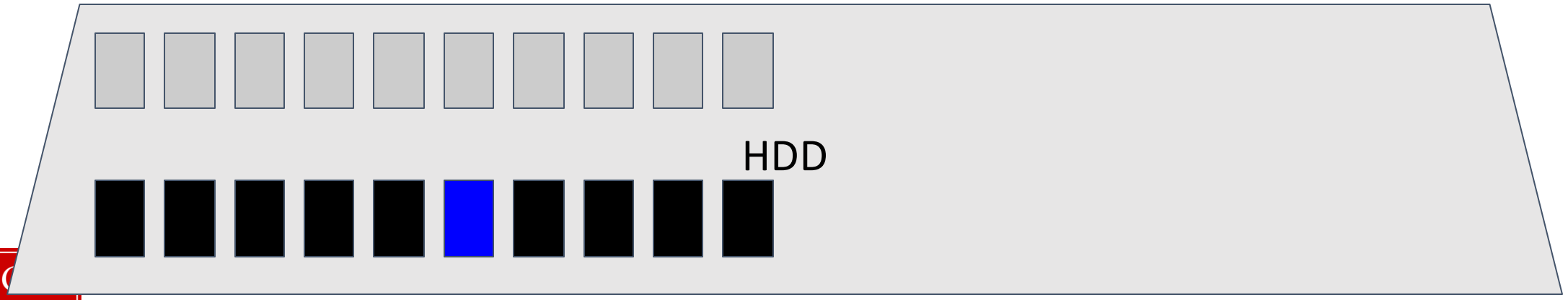
IO cost: Scanning a relation to select 10%



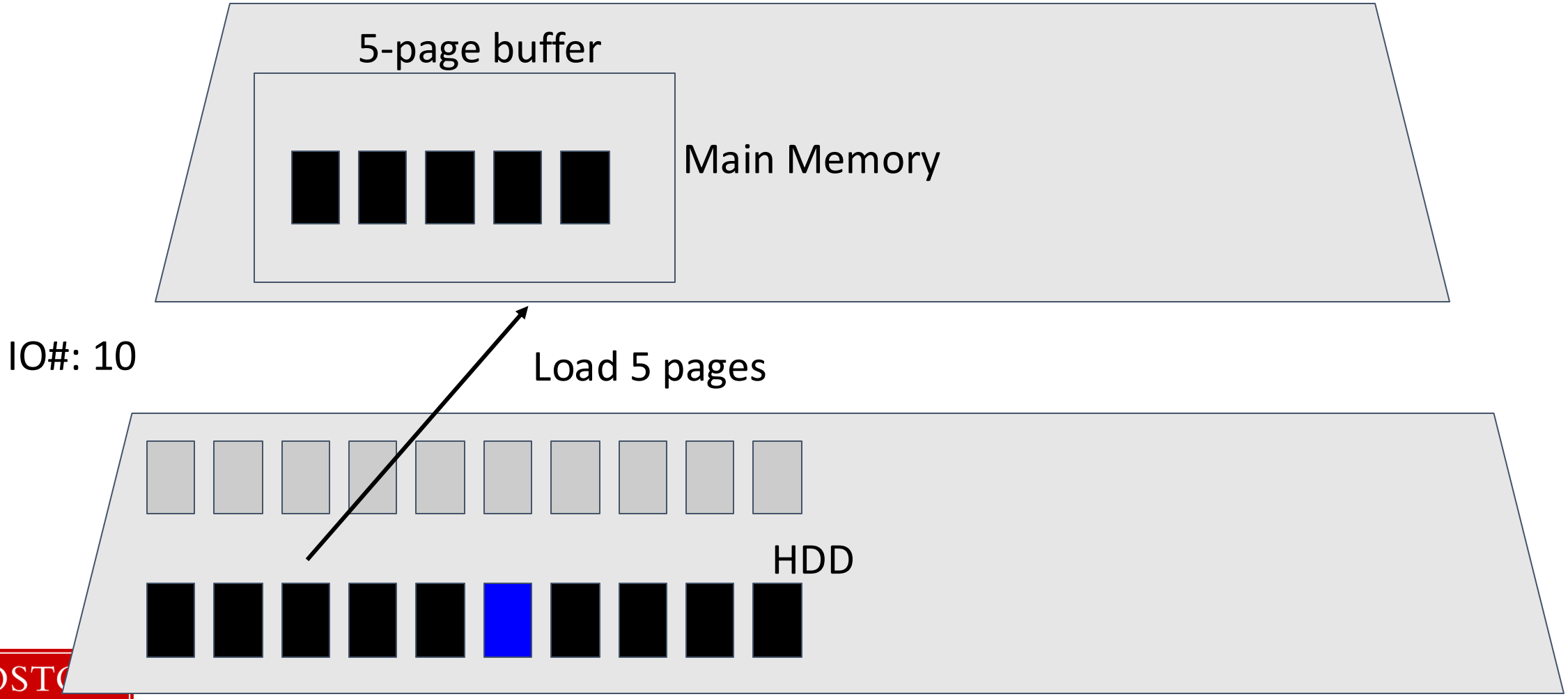
IO cost: Scanning a relation to select 10%



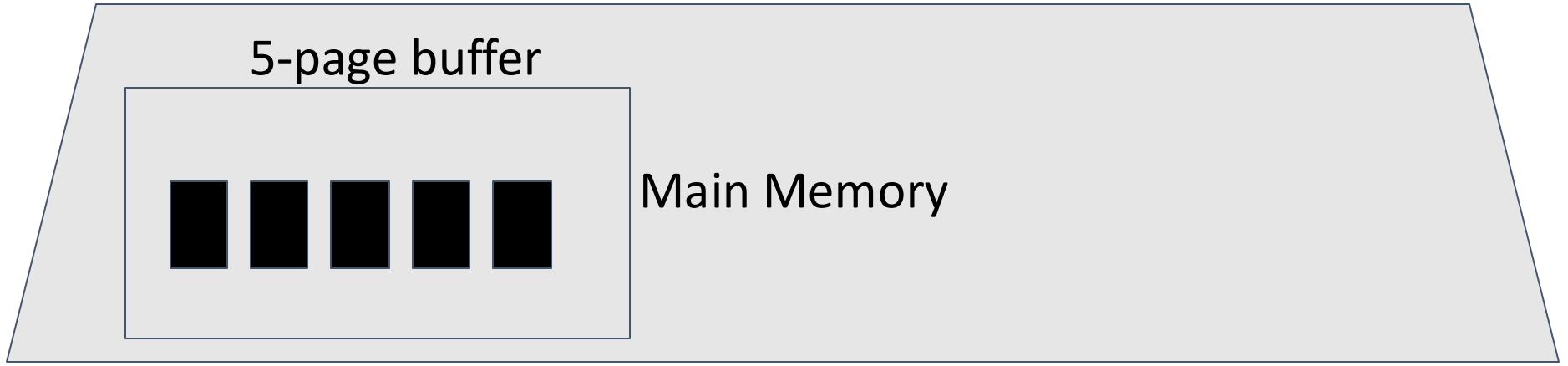
IO#: 10



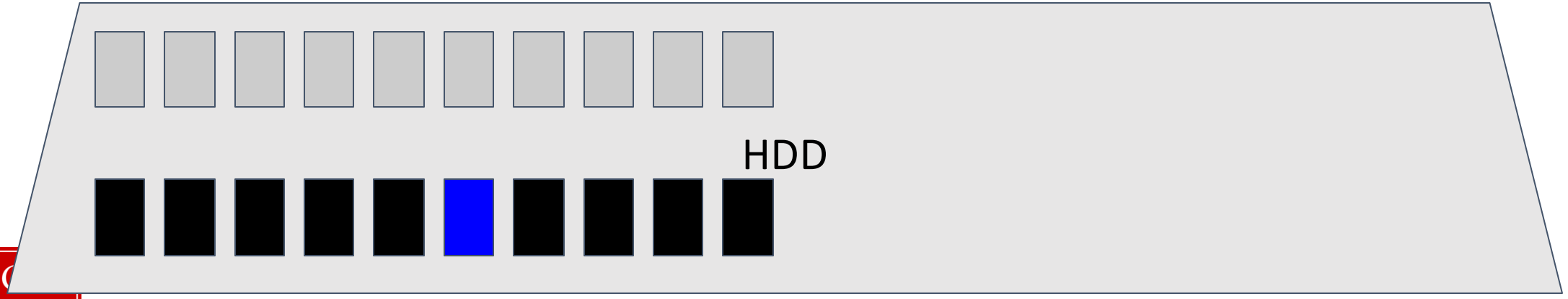
IO cost: Scanning a relation to select 10%



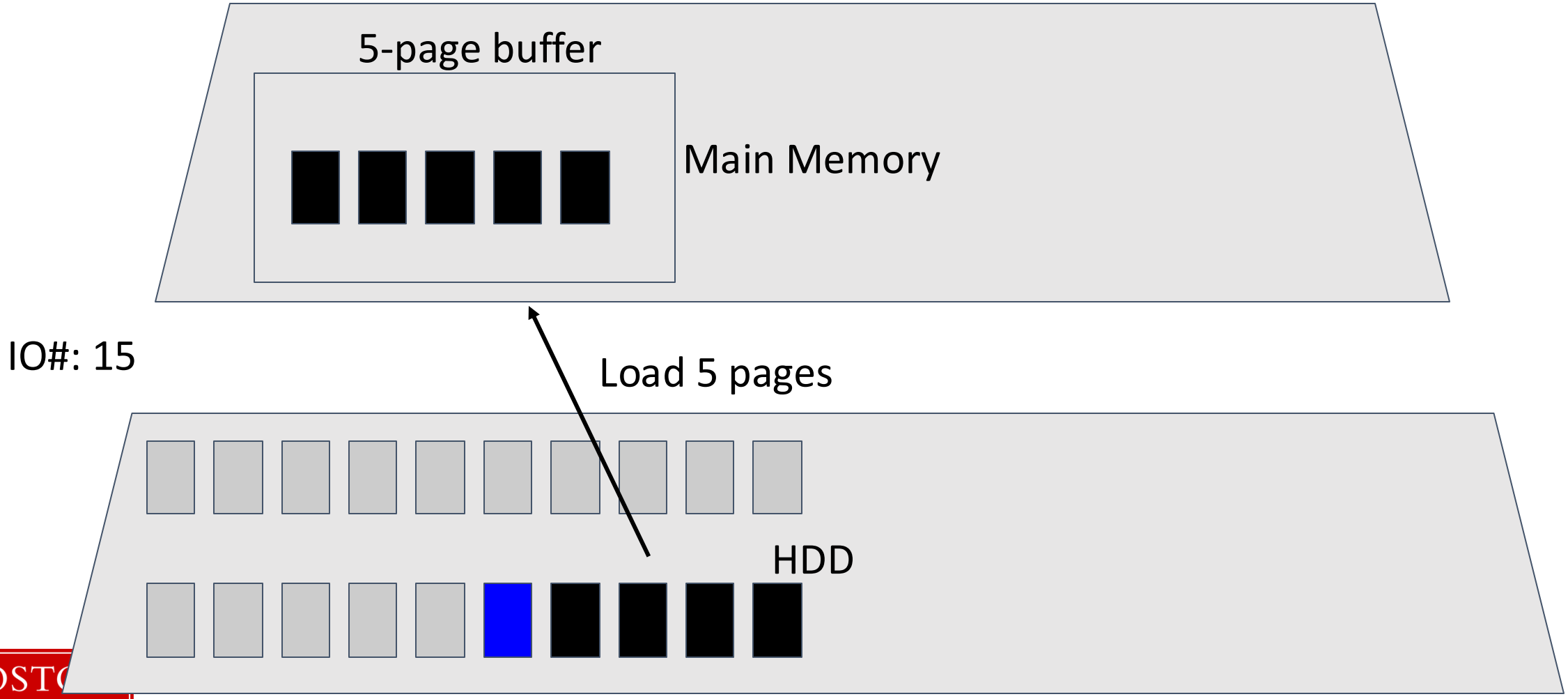
IO cost: Scanning a relation to select 10%



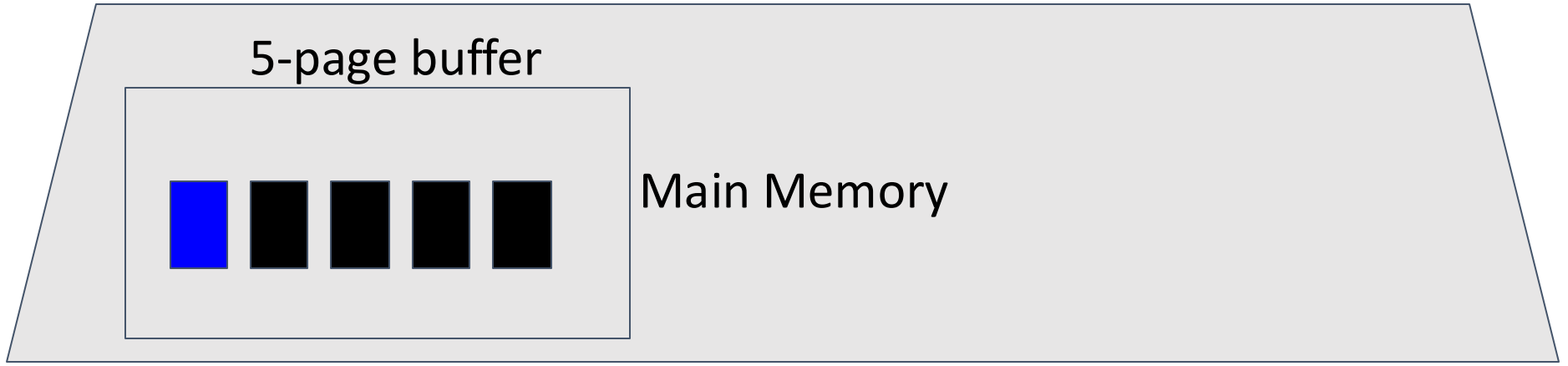
IO#: 15



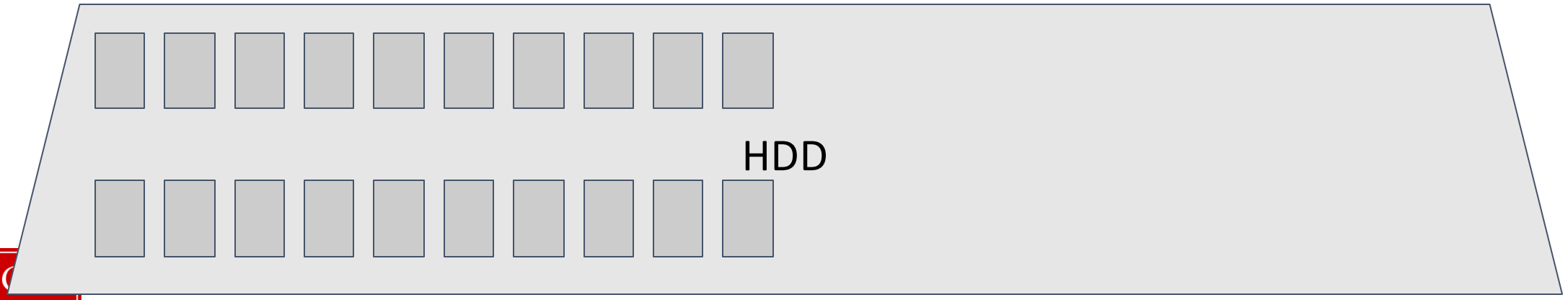
IO cost: Scanning a relation to select 10%



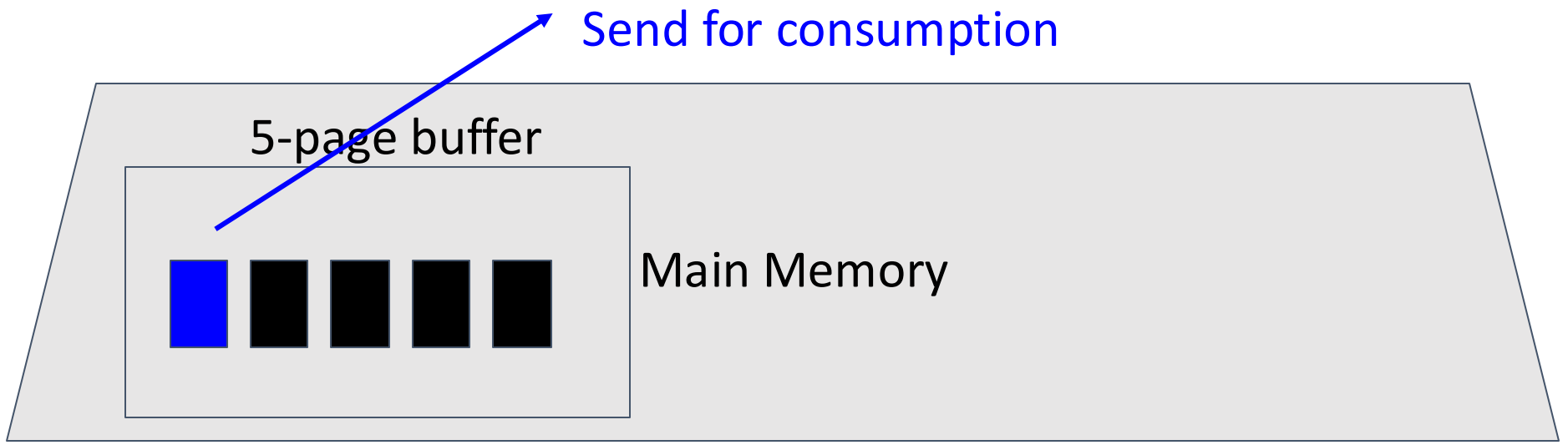
IO cost: Scanning a relation to select 10%



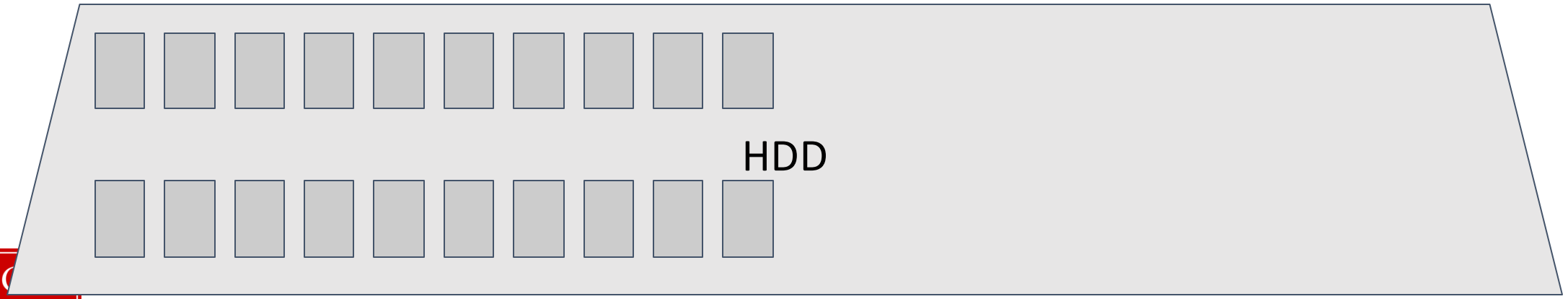
IO#:
20



IO cost: Scanning a relation to select 10%

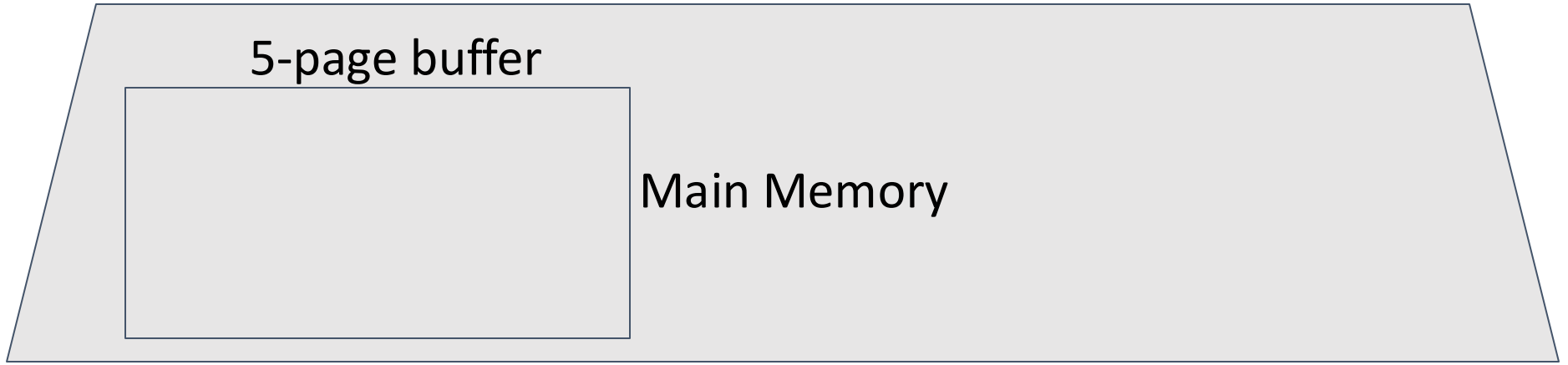


IO#:
20

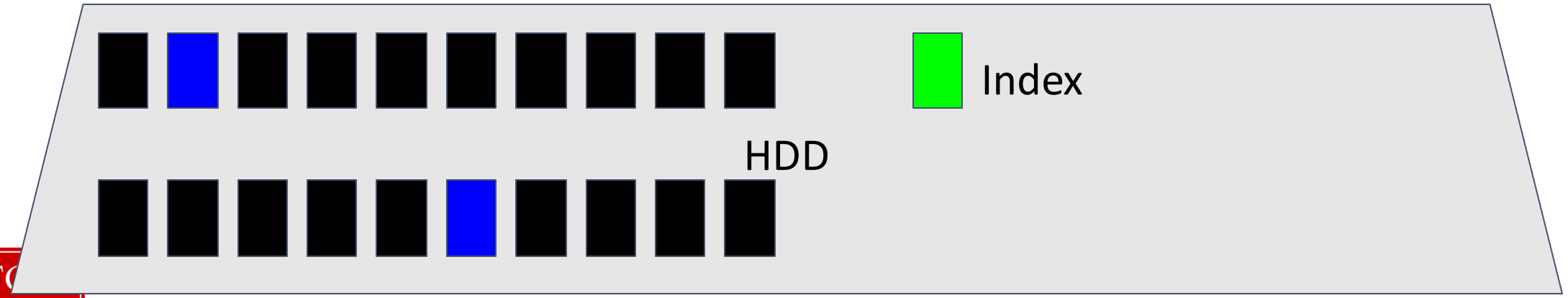


What if we had an oracle (index)?

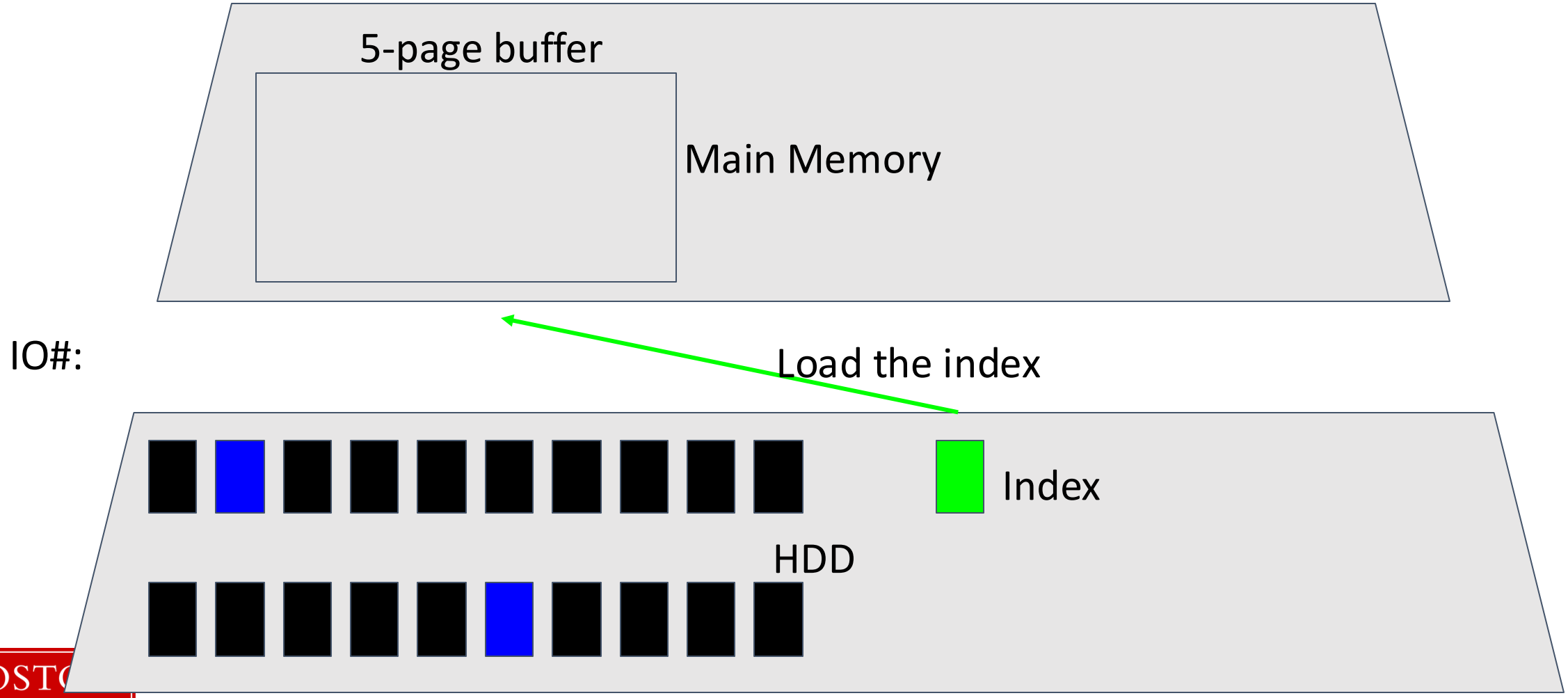
IO cost: Scanning a relation to select 10%



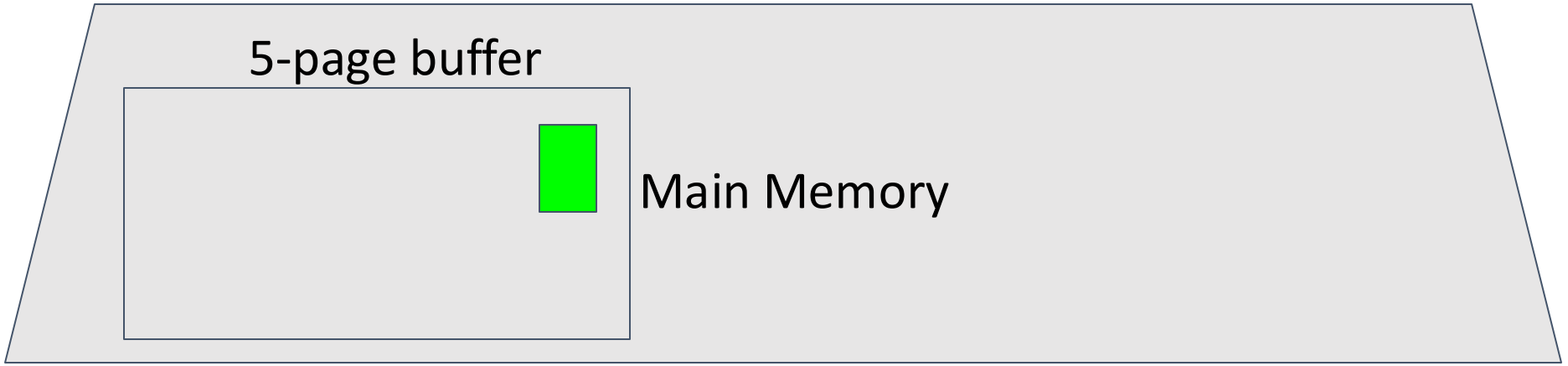
IO#:



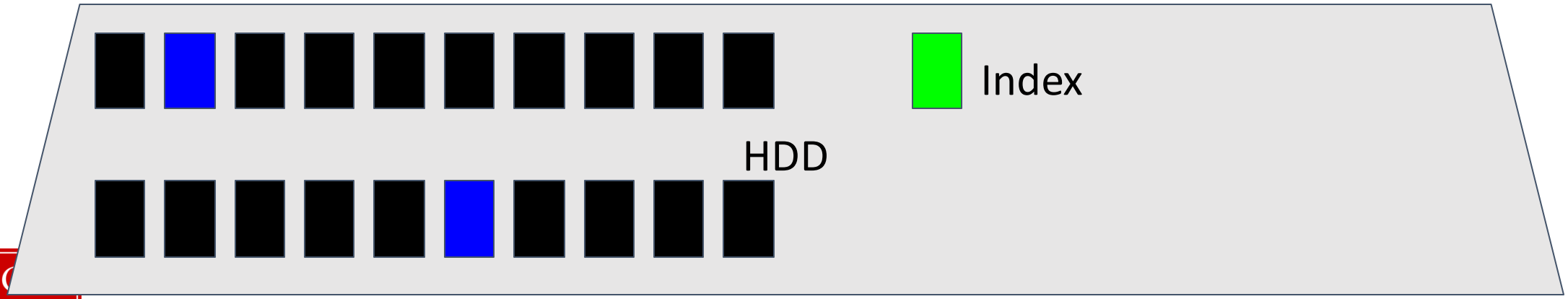
IO cost: Use an index to select 10%



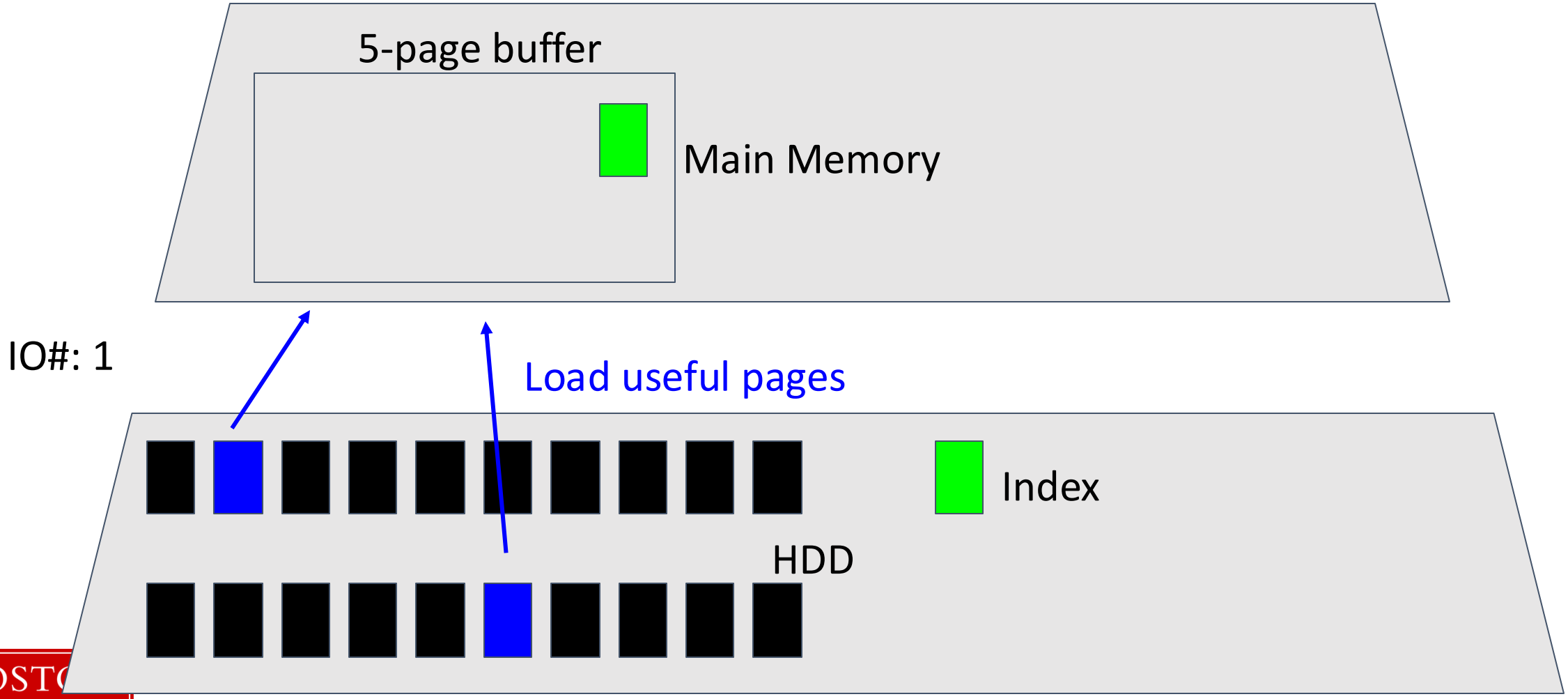
IO cost: Use an index to select 10%



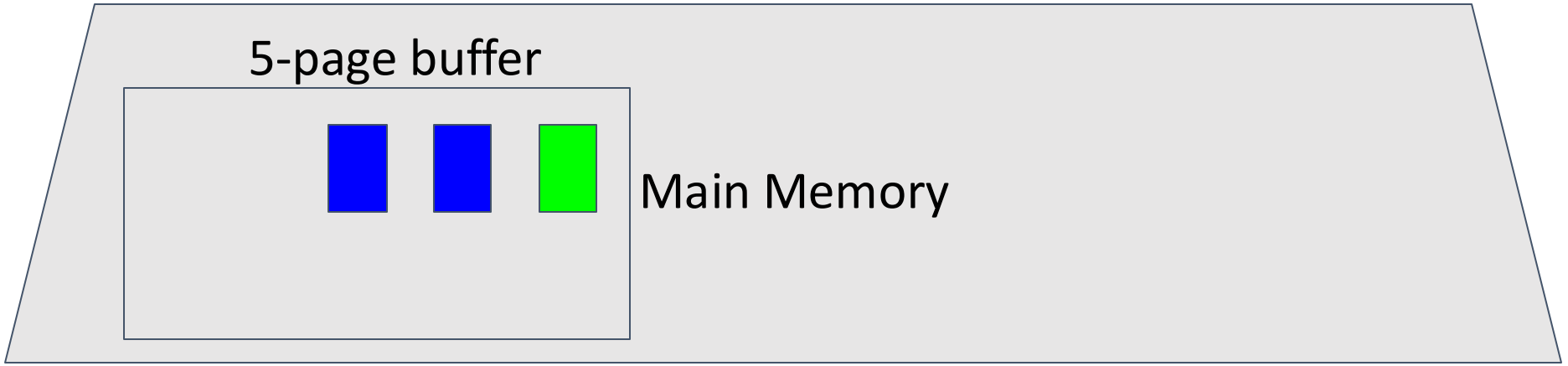
IO#: 1



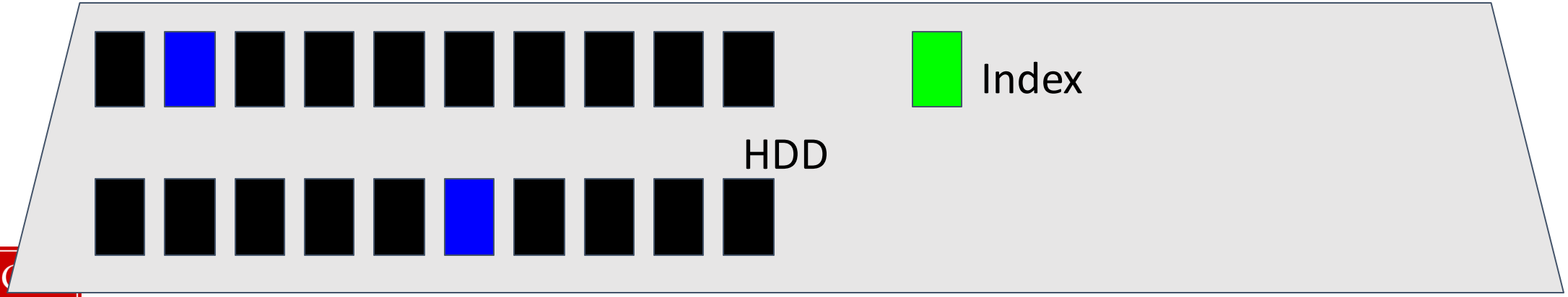
IO cost: Use an index to select 10%



IO cost: Use an index to select 10%

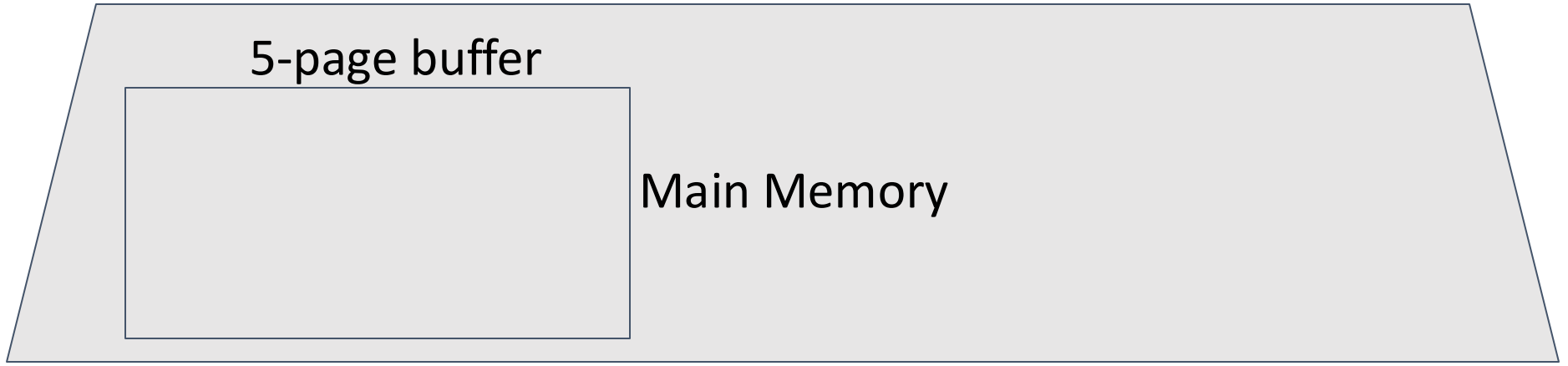


IO#: 3

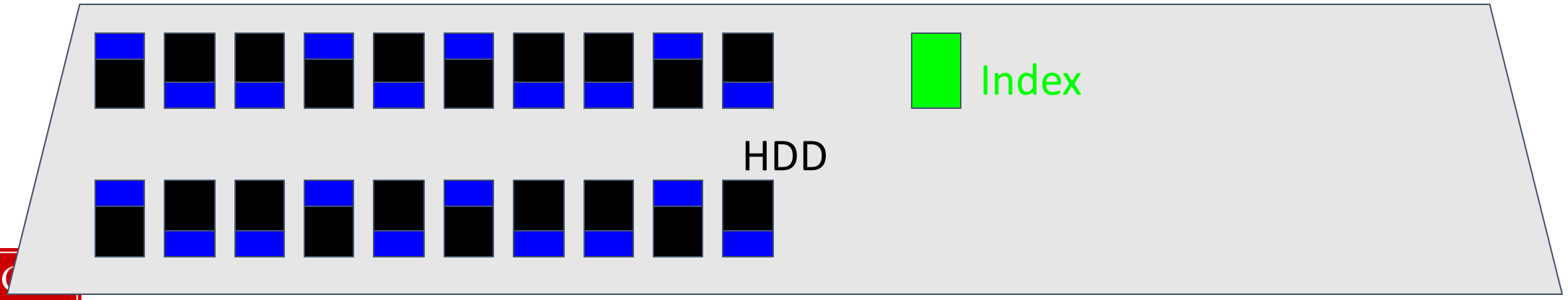


What if useful data is in all pages?

Scan or Index ?

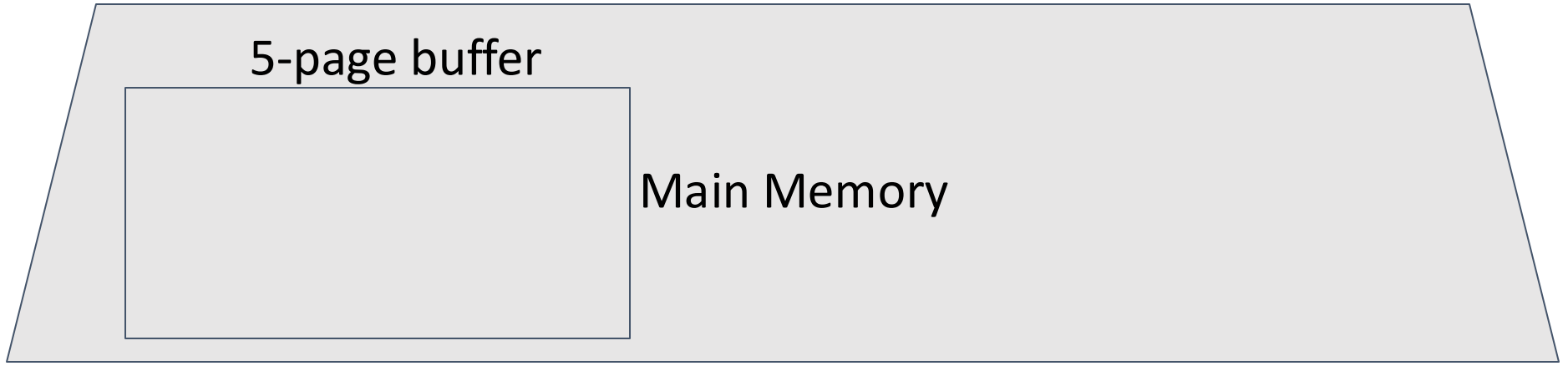


IO#:



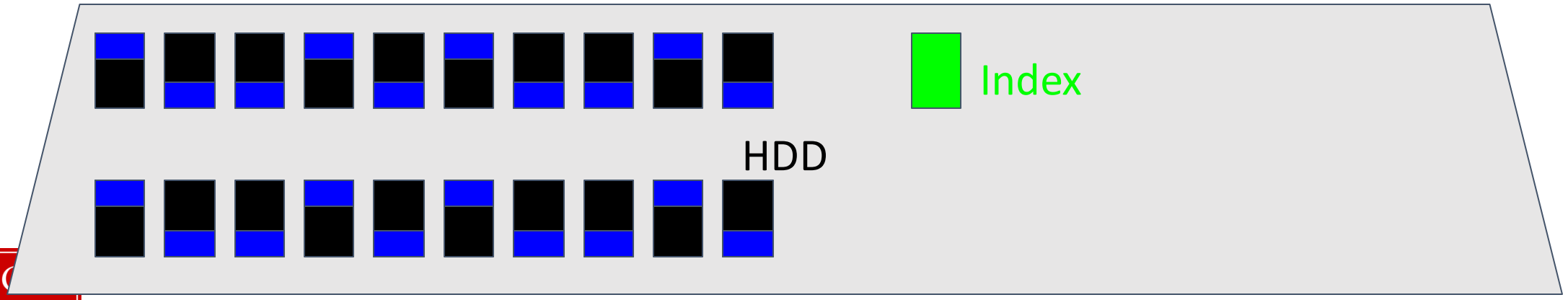
Scan or Index ?

depends on hardware, selectivity, and physical data storage

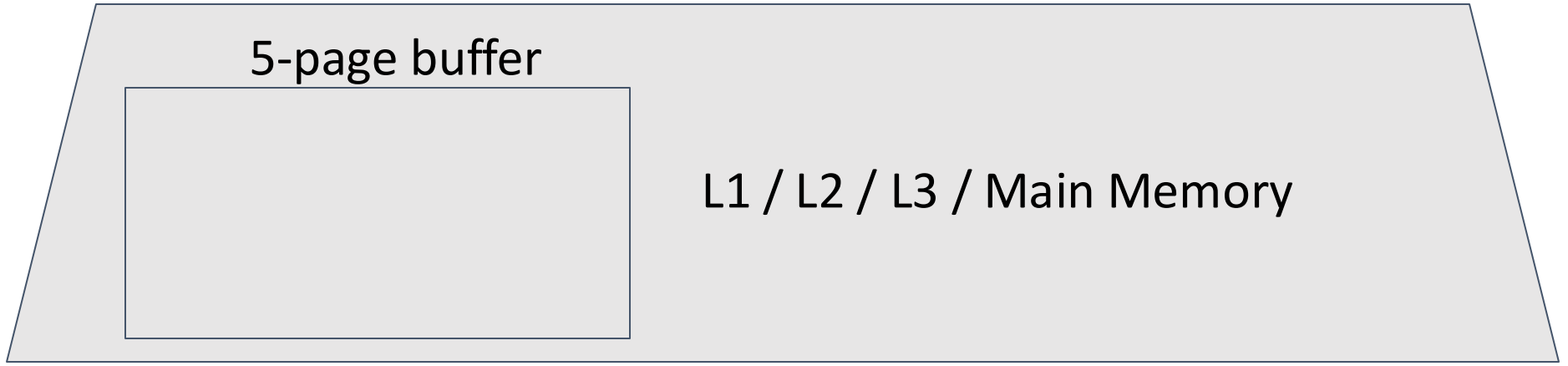


IO#: 20 with scan

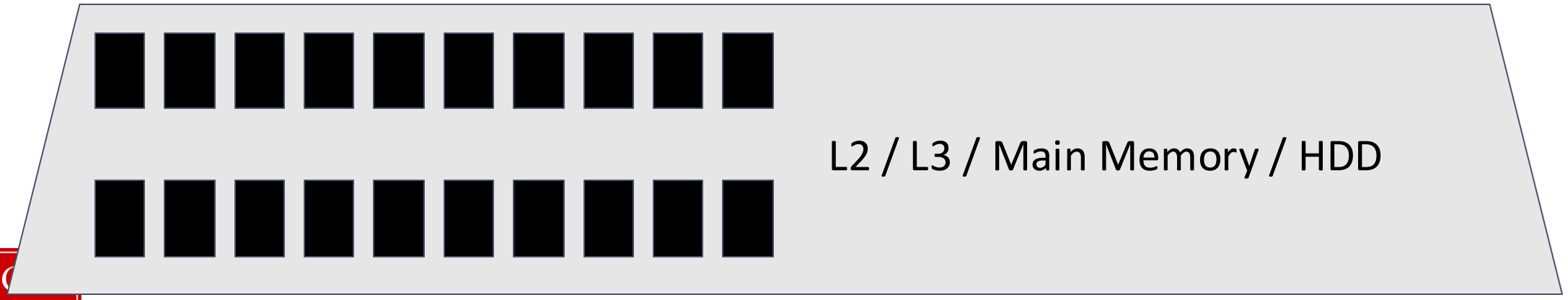
IO#: 21 with index



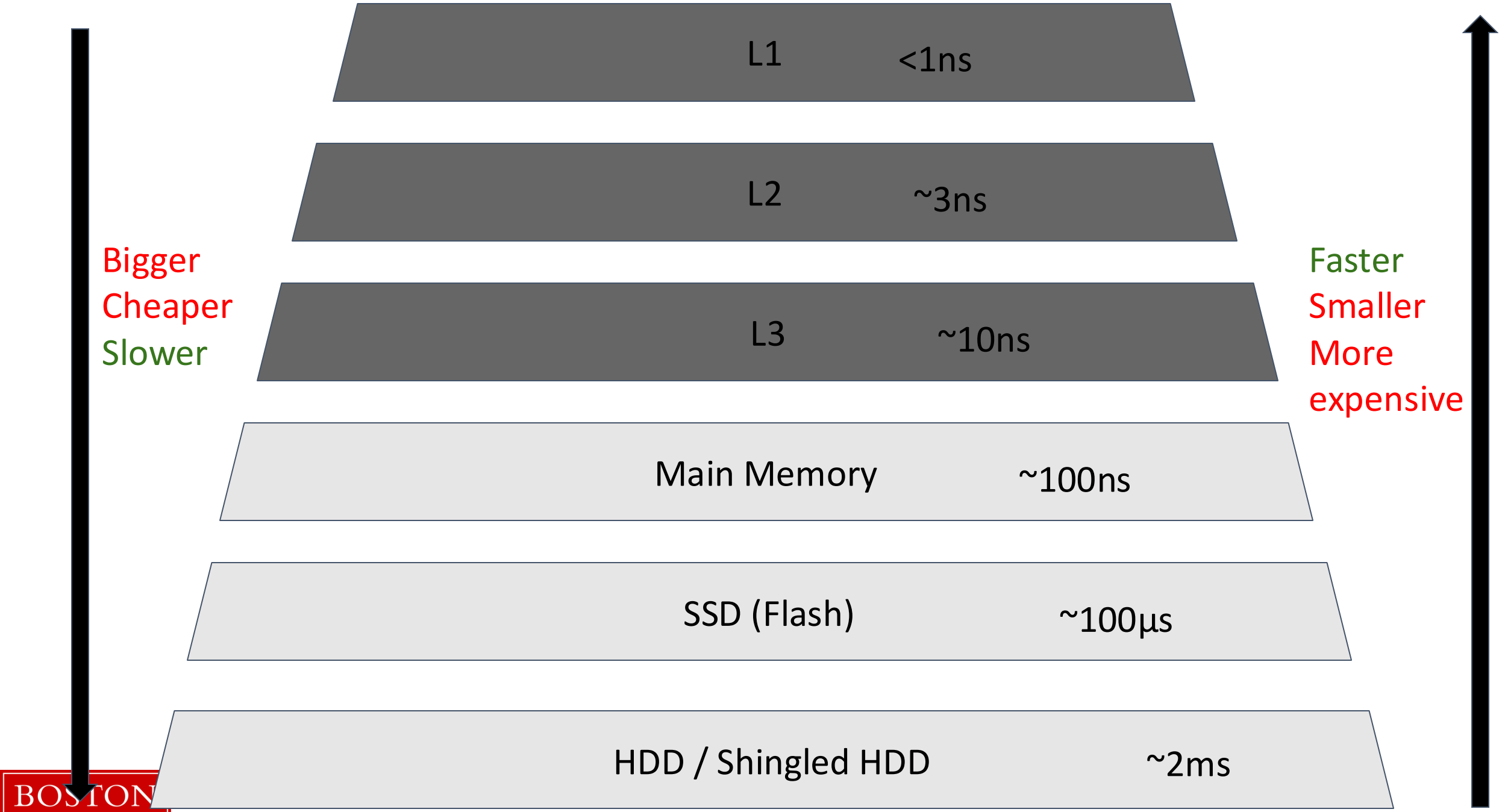
Same analysis for any two memory levels!!



IO#:



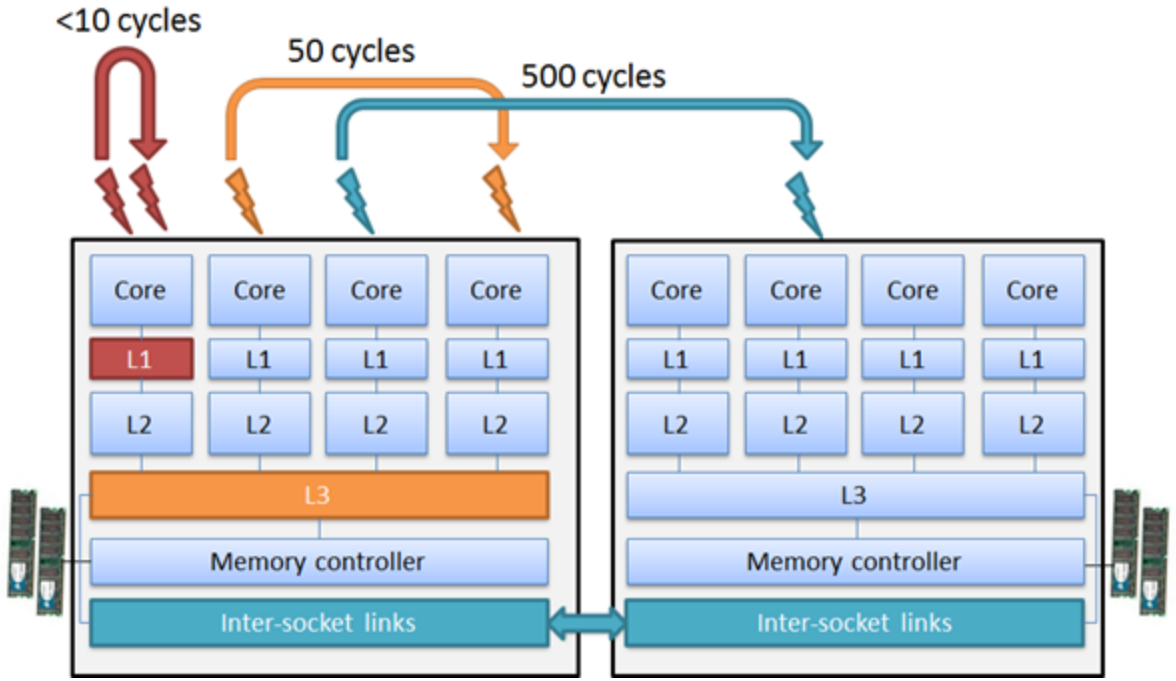
Cache Hierarchy

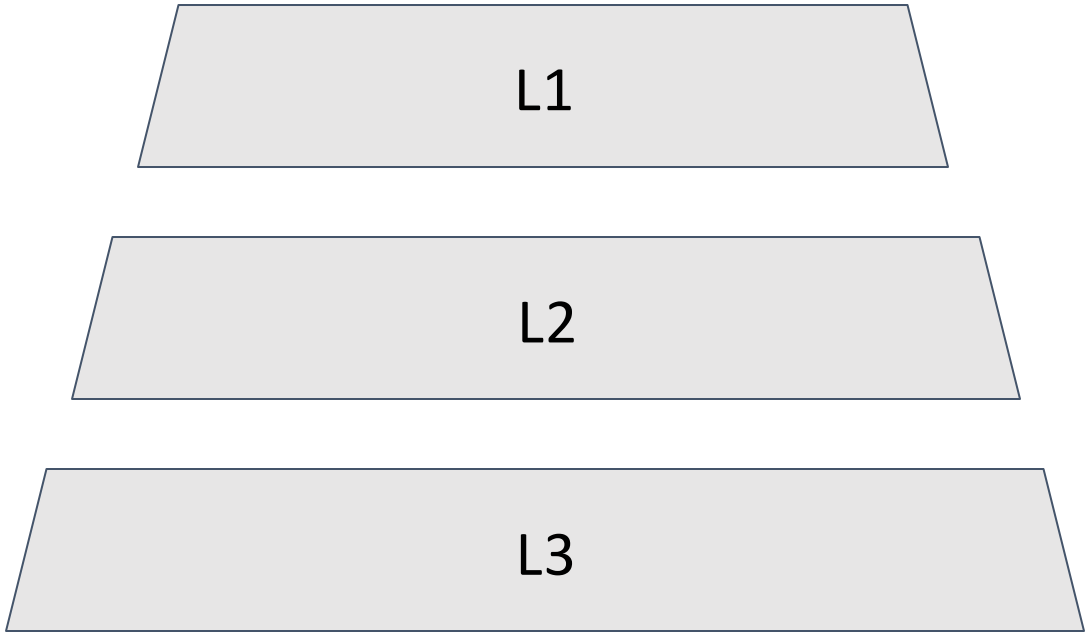


Cache Hierarchy

What is a core?

What is a socket?





Cache Hierarchy

Shared Cache: L3 (or LLC: Last Level Cache)

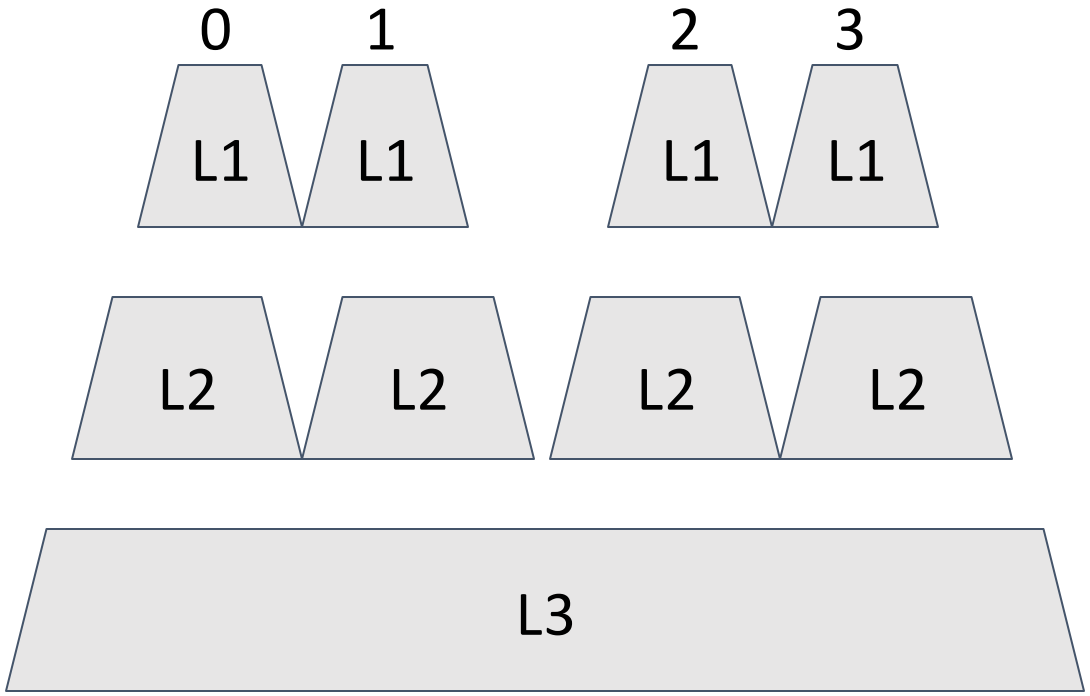
L3 is physically distributed in multiple sockets

L2 is physically distributed in every core of every socket

Each *core* has its own **private** L1 & L2 cache

All levels need to be ***coherent****

what does coherent mean?



Non Uniform Memory Access (NUMA)

A core reads faster when data are in its L1

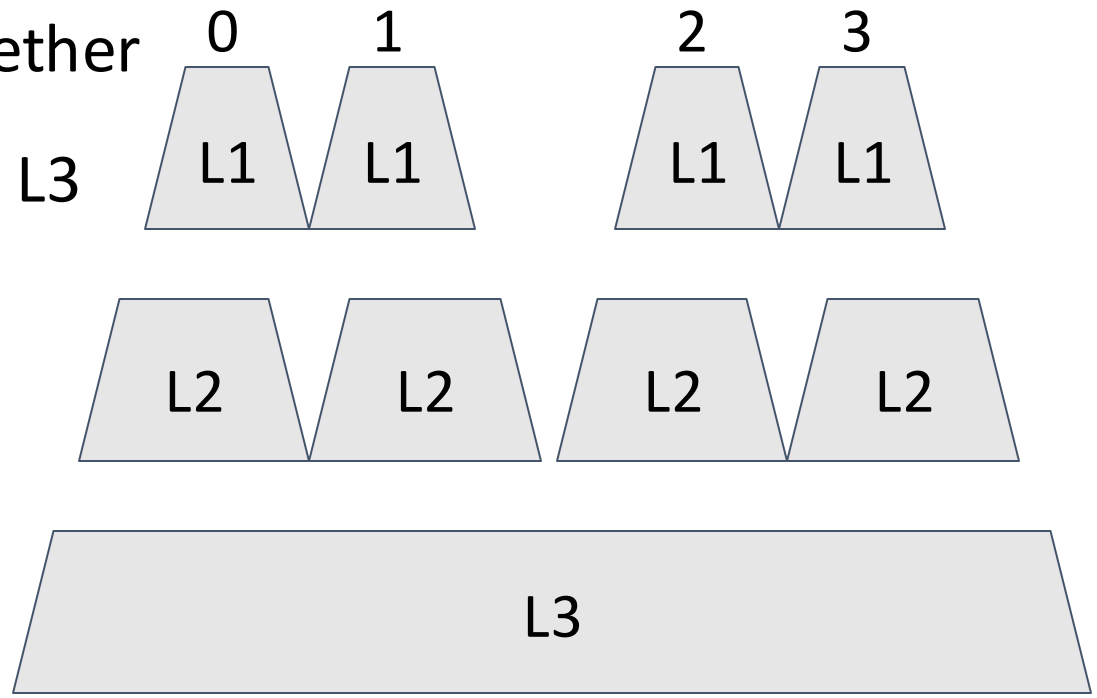
If it does not fit, it will go to L2, and then in L3

Can we control where data is placed?

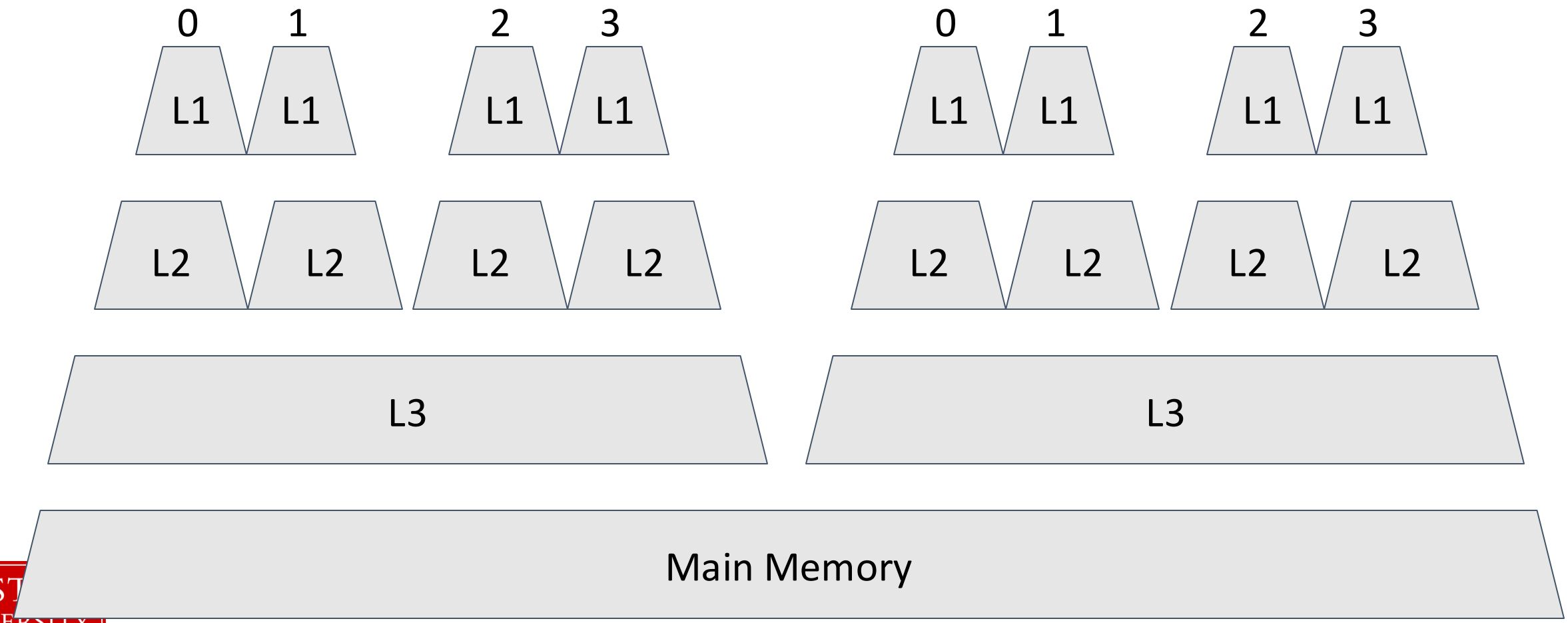
We would like to avoid going to L2 and L3 altogether

But, at least we want to avoid to remote L2 and L3

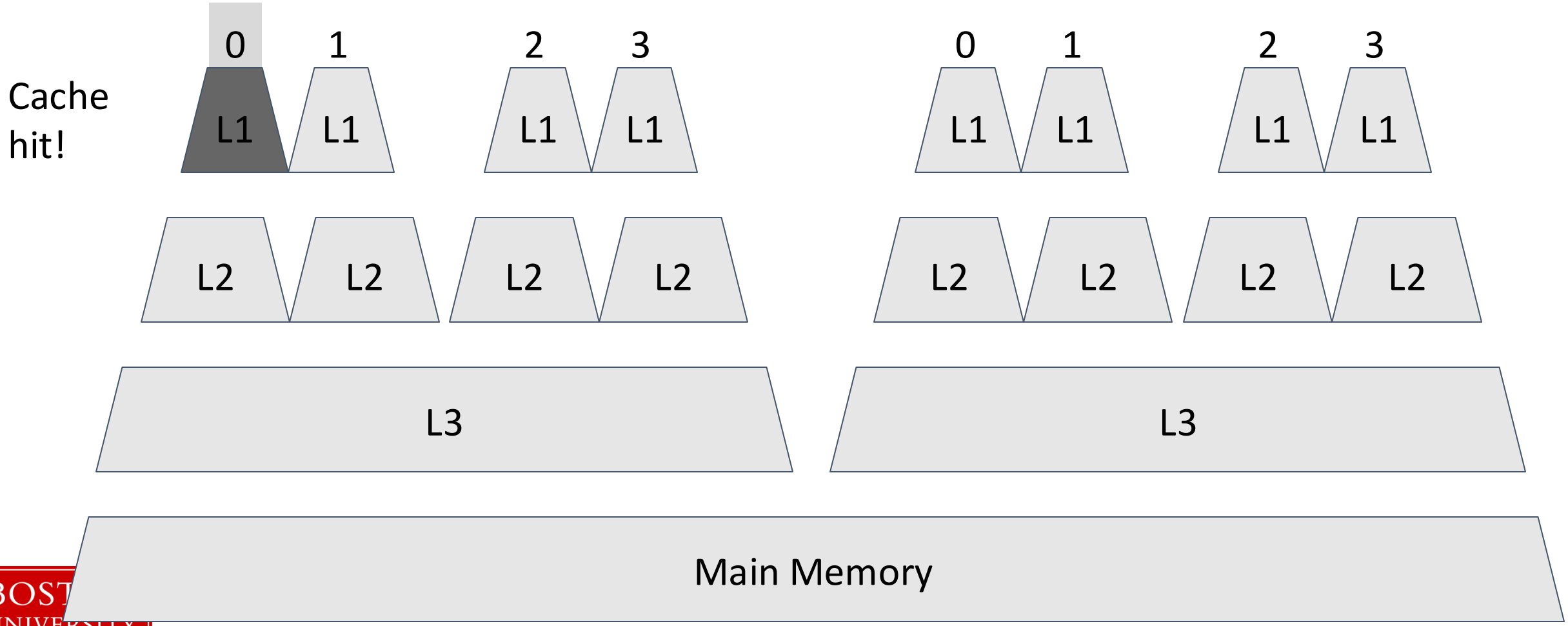
And remember: this is only one socket.
We have **multiple** sockets!



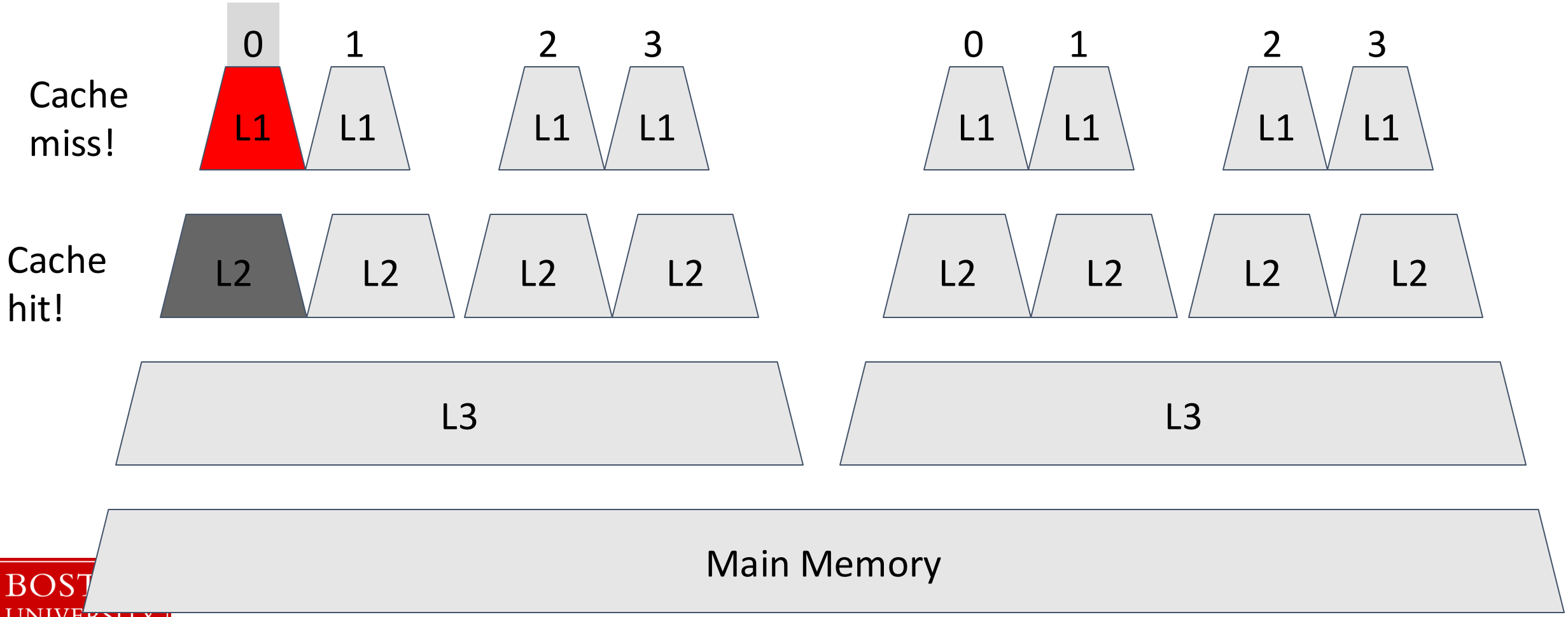
Non Uniform Memory Access (NUMA)



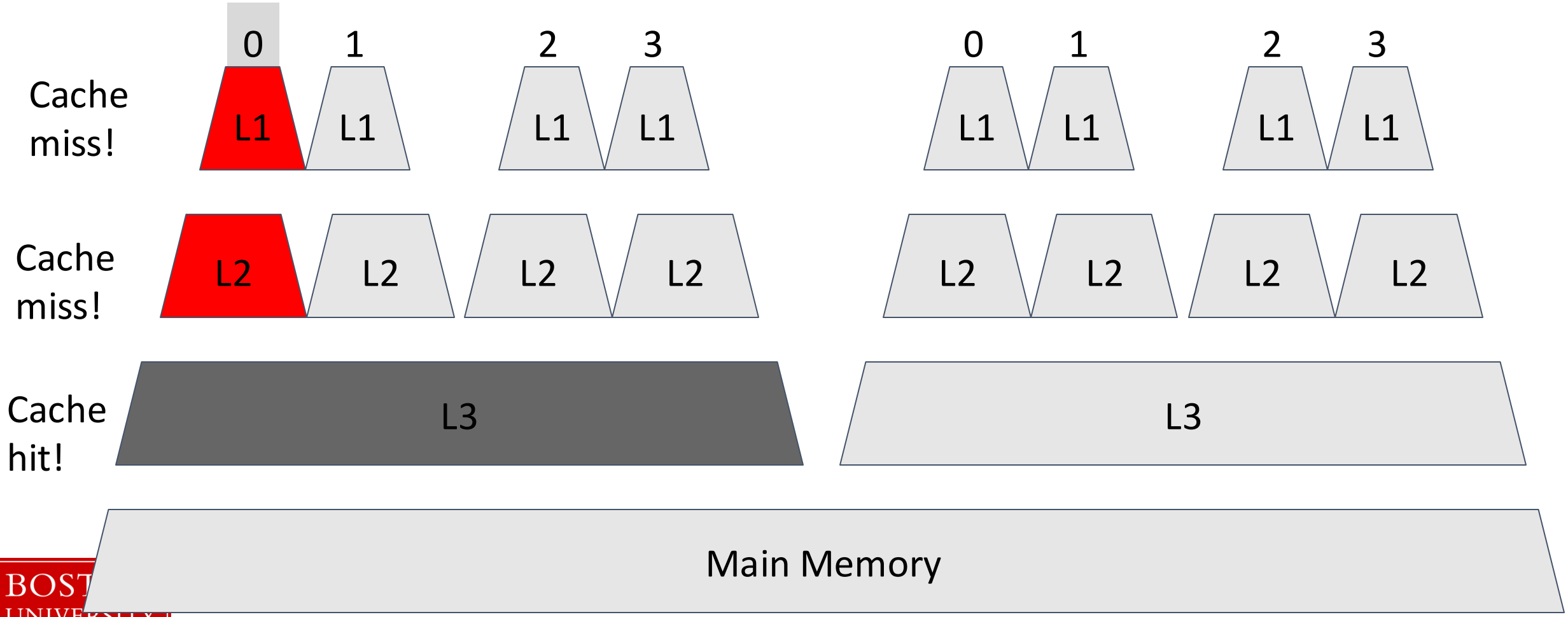
Non Uniform Memory Access (NUMA)



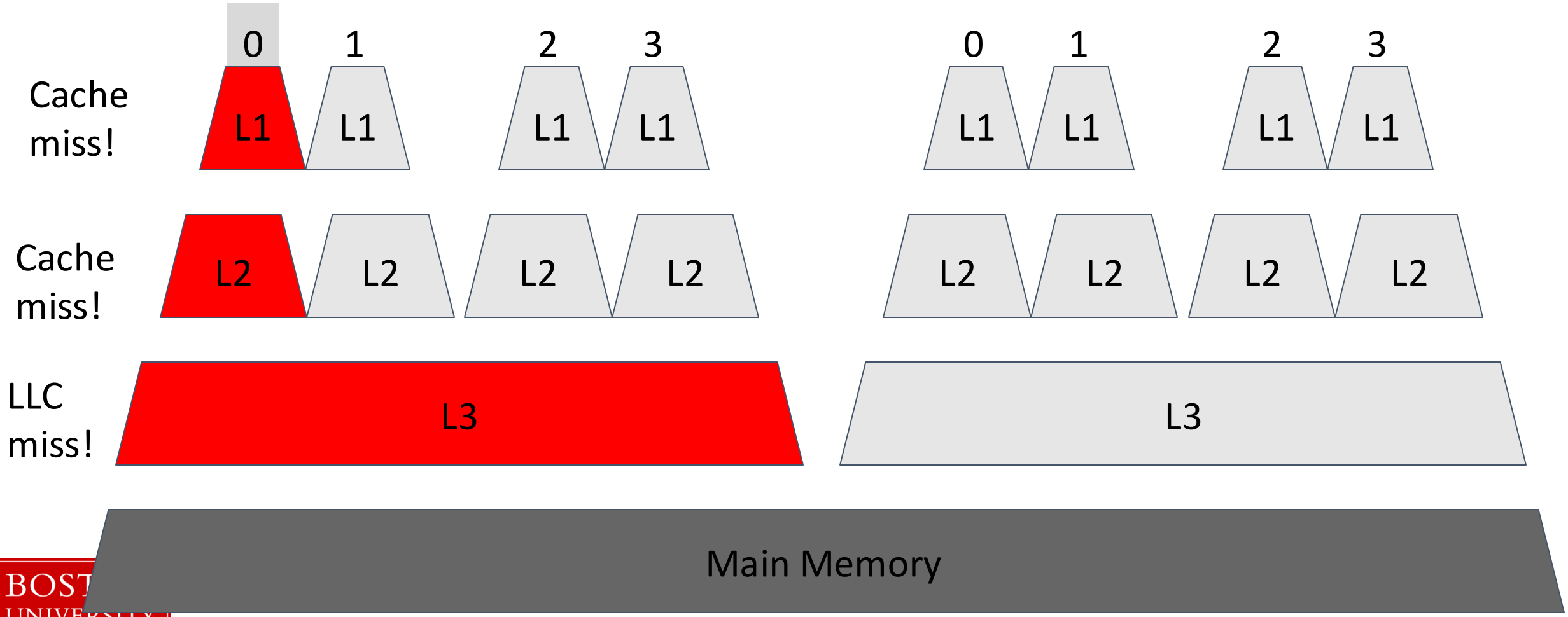
Non Uniform Memory Access (NUMA)



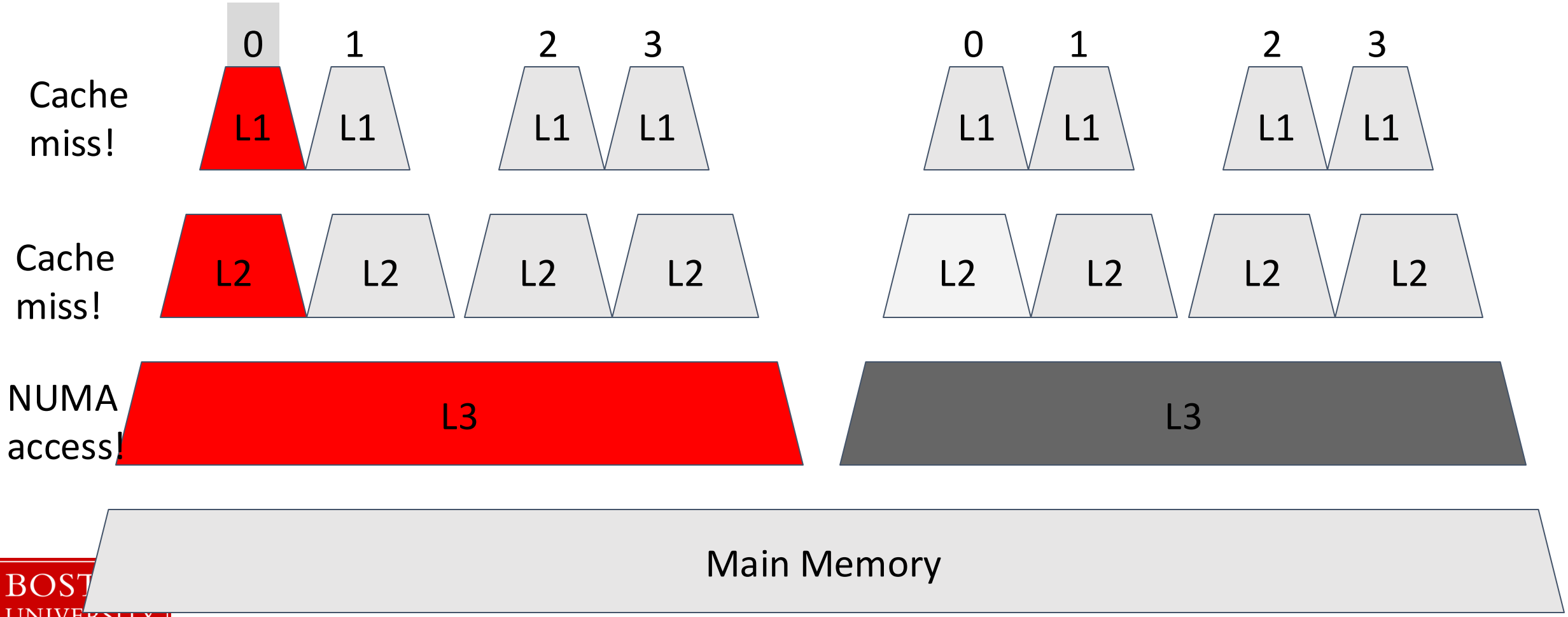
Non Uniform Memory Access (NUMA)



Non Uniform Memory Access (NUMA)



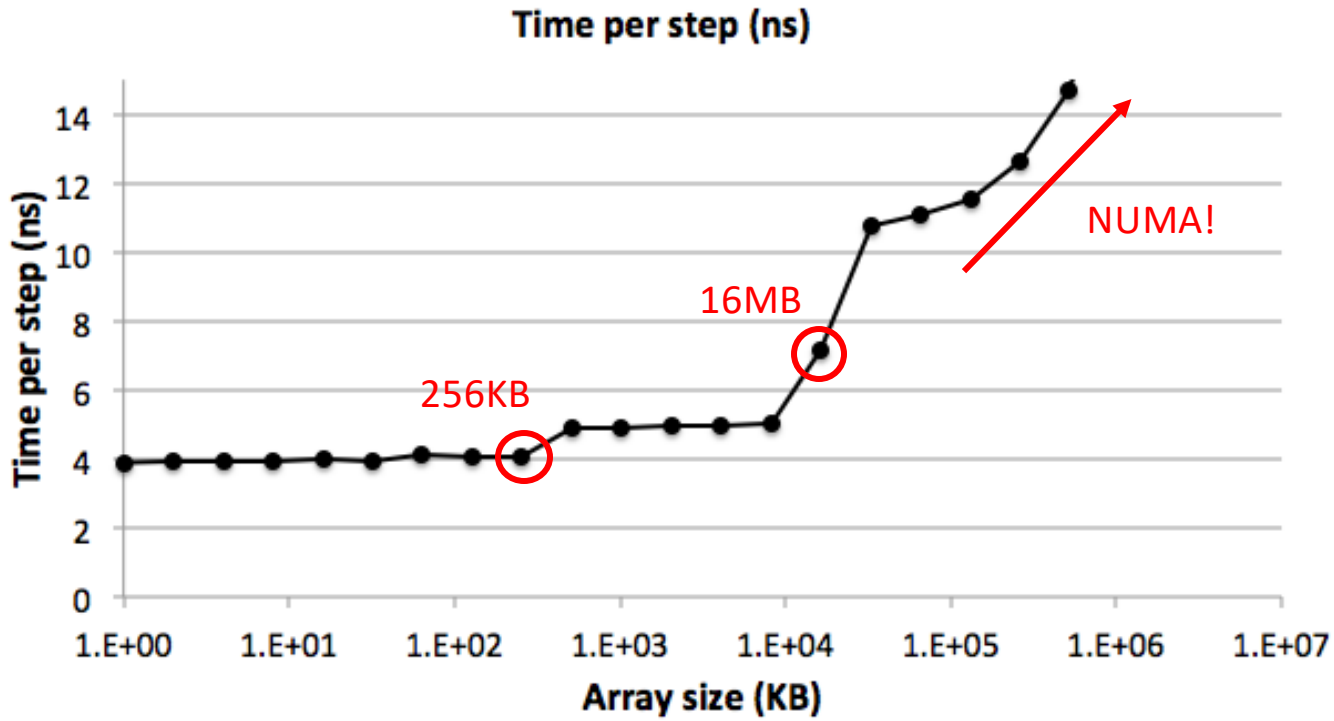
Non Uniform Memory Access (NUMA)



Why knowing the cache hierarchy matters

```
int arraySize;
for (arraySize = 1024/sizeof(int) ; arraySize <= 2*1024*1024*1024/sizeof(int) ; arraySize*=2)
// Create an array of size 1KB to 4GB and run a large arbitrary number of operations
{
    int steps = 64 * 1024 * 1024; // Arbitrary number of steps
    int* array = (int*) malloc(sizeof(int)*arraySize); // Allocate the array
    int lengthMod = arraySize - 1;

    // Time this loop for every arraySize
    int i;
    for (i = 0; i < steps; i++)
    {
        array[(i * 16) & lengthMod]++;
        // (x & lengthMod) is equal to (x % arraySize)
    }
}
```



This machine has:

256KB L2 per core

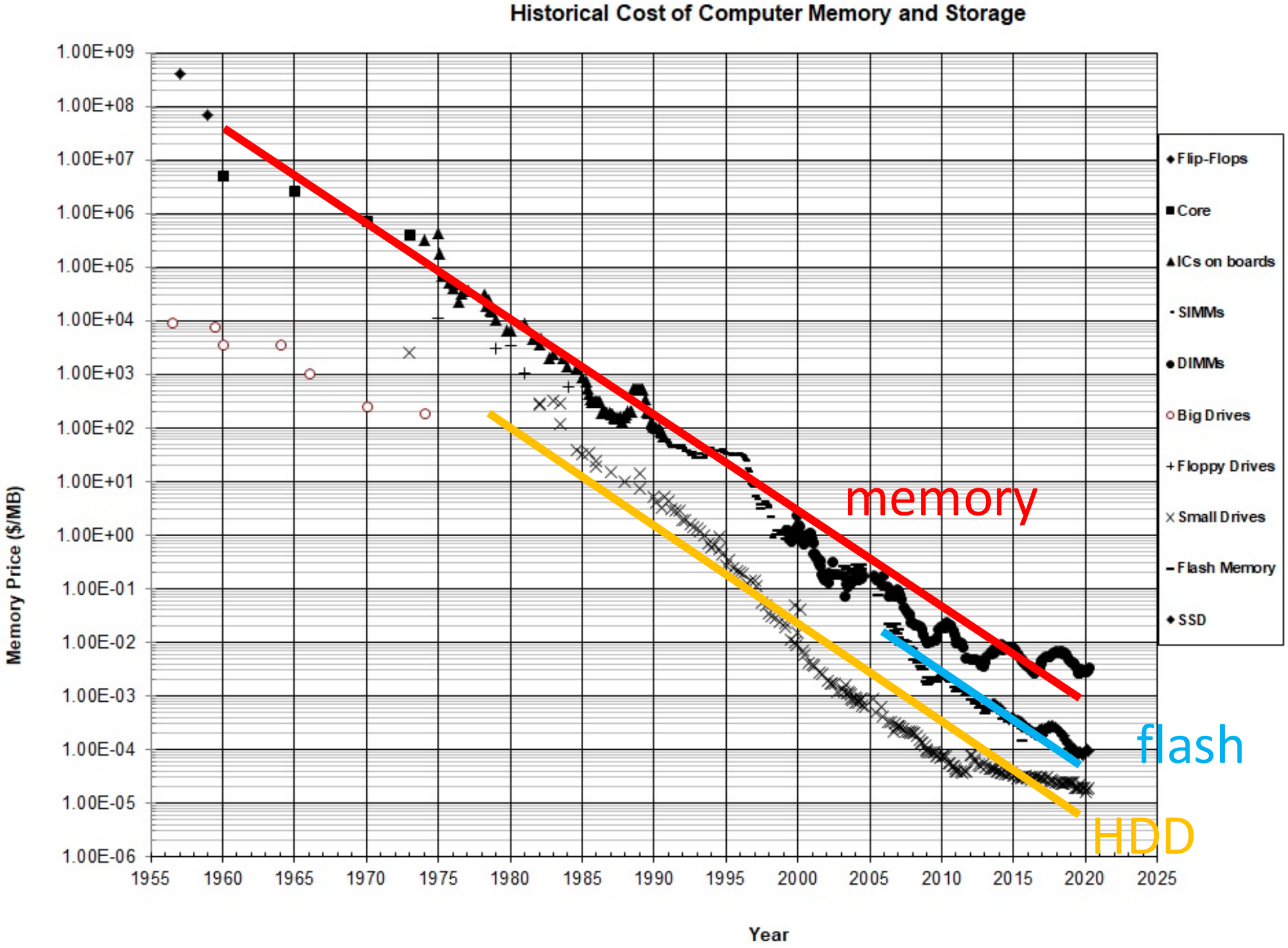
16MB L3 per socket

Storage Hierarchy

Why not just stay in memory?

Cost!

what else?



BOSTON UNIVERSITY

Storage Hierarchy

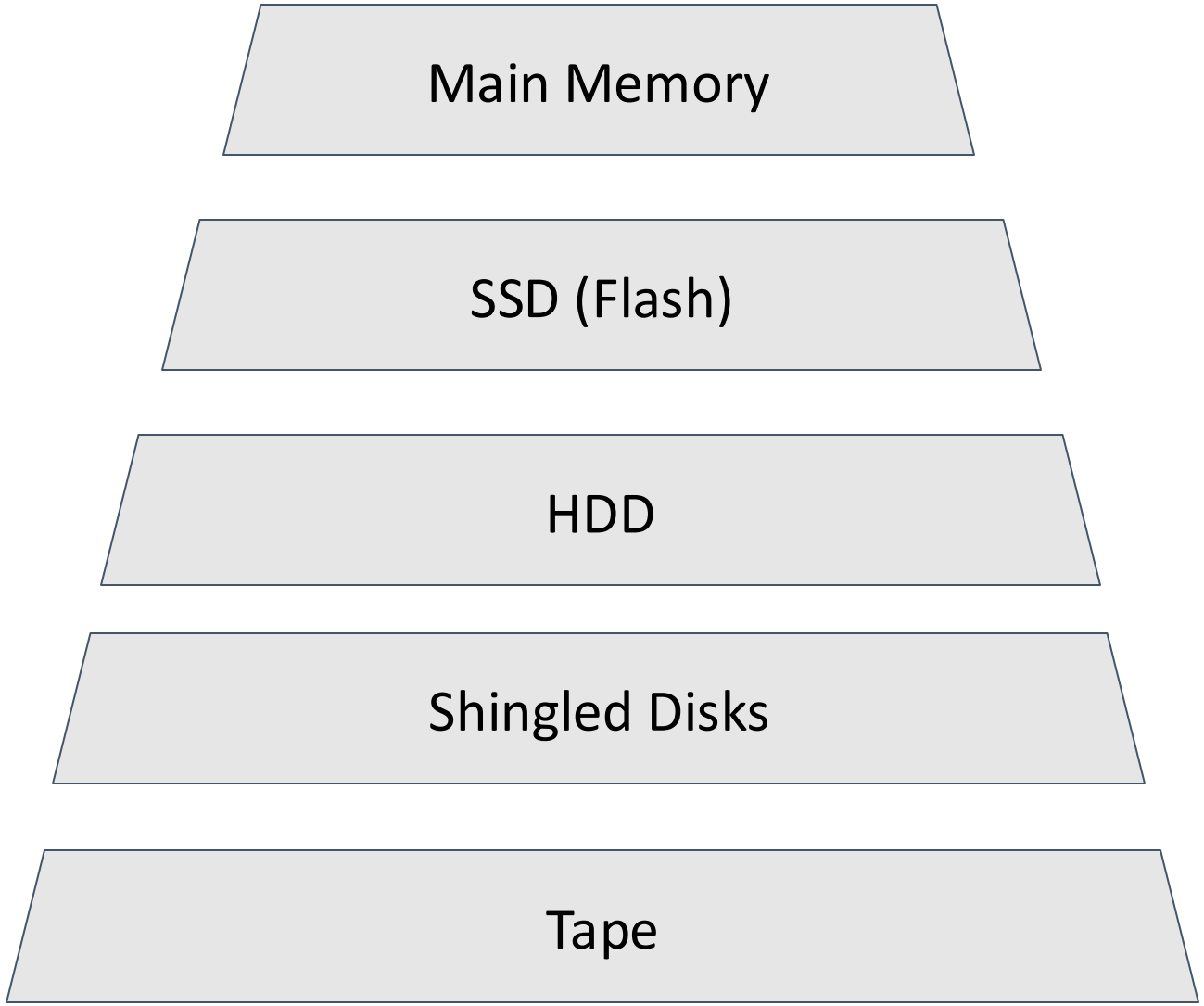
Why not stay in memory?

Rephrase: what is missing from memory hierarchy?

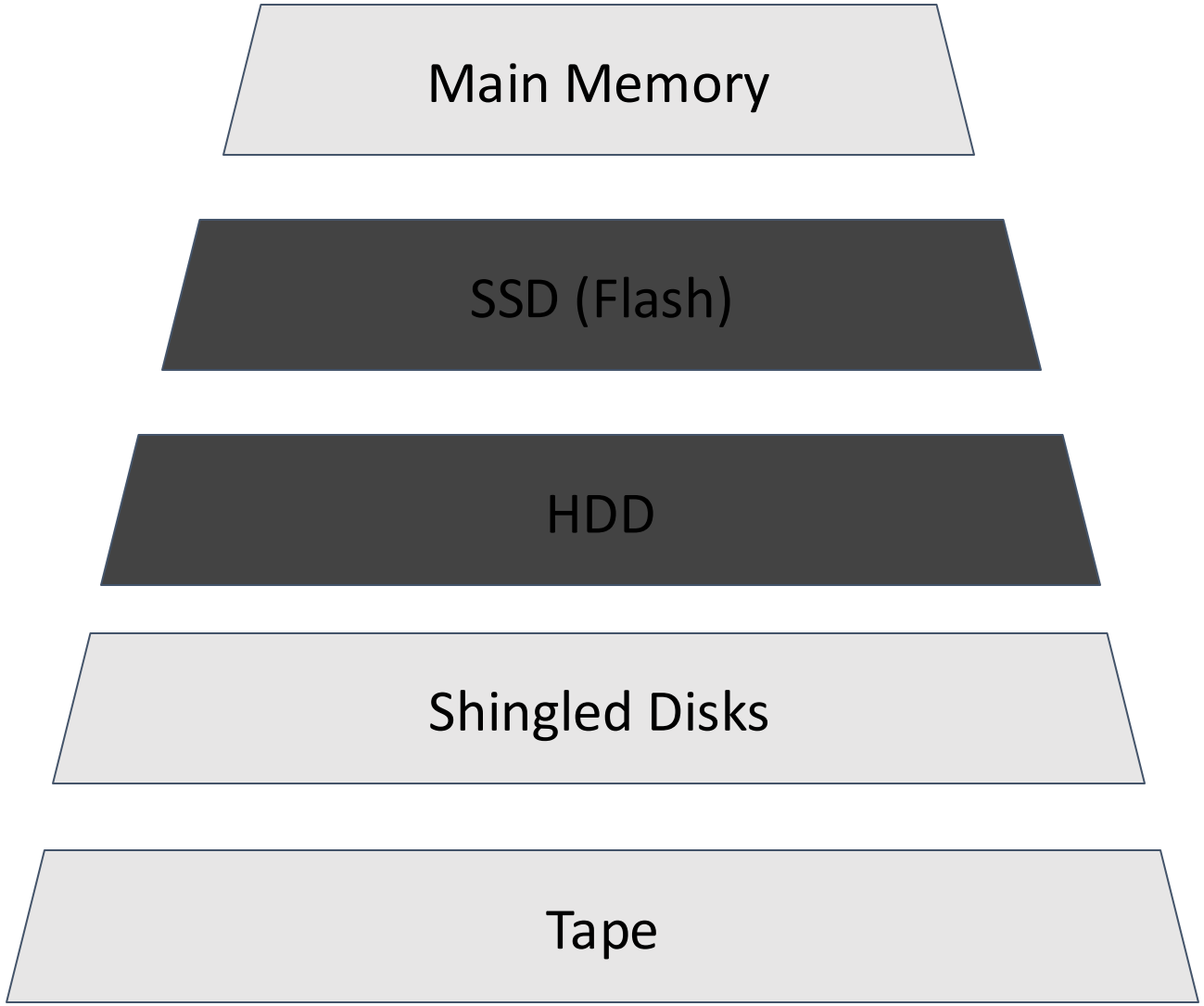
Durability (data survives between restarts)

Capacity (enough capacity for data-intensive applications)

Storage Hierarchy



Storage Hierarchy



Hard Disk Drives

Secondary durable storage that support both *random* and *sequential* access

Data organized on pages/blocks (across tracks)

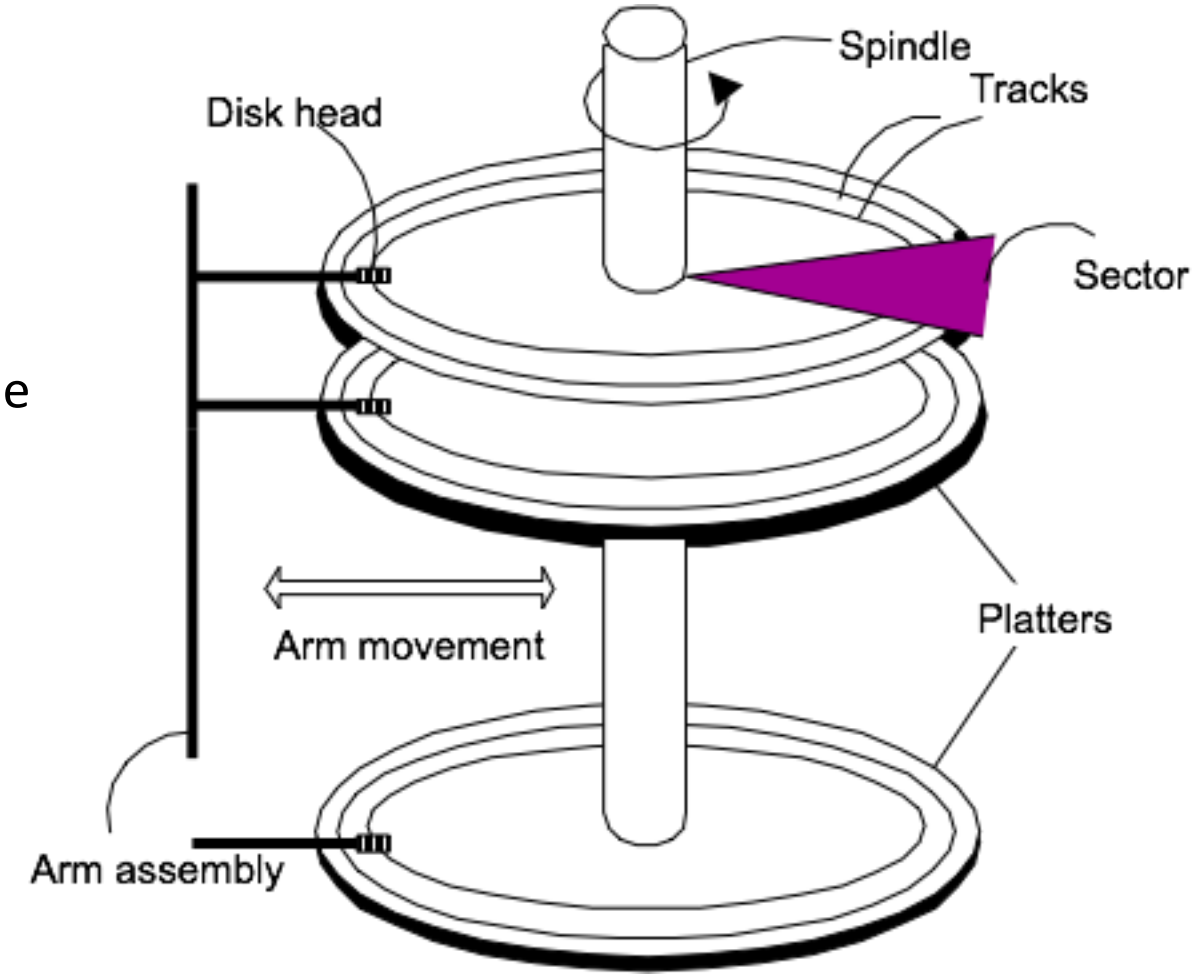
Multiple *tracks* create an (imaginary) *cylinder*

Disk access time:

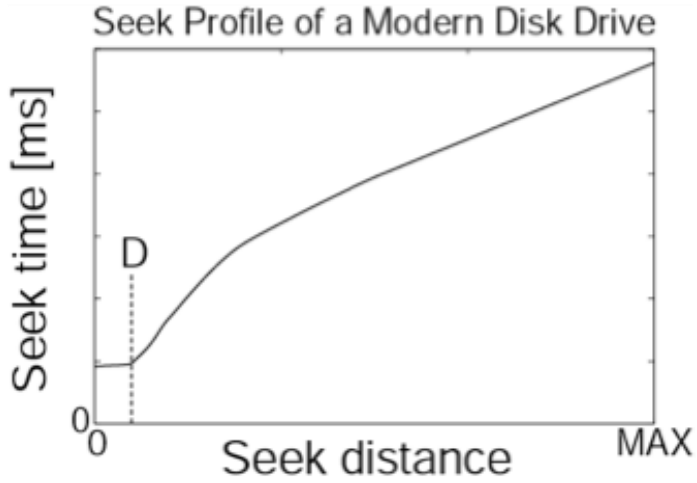
seek latency + rotational delay + transfer time
(0.5-2ms) + (0.5-3ms) + <0.1ms/4KB

Sequential >> random access (~10x)

Goal: *avoid random access*



Seek time + Rotational delay + Transfer time



Seek time: the **head** goes to the right **track**

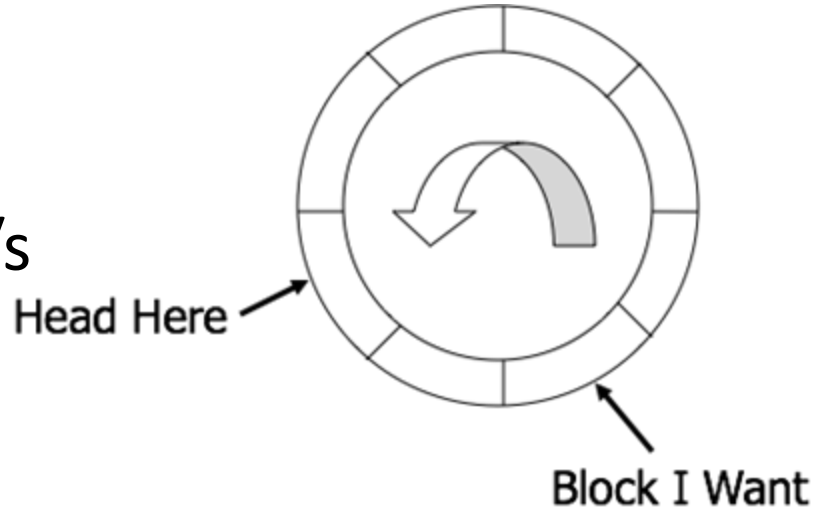
Short seeks are dominated by "settle" time
(D is on the order of hundreds or more)

Rotational delay: The **platter** rotates to the right **sector.**

What is the min/max/avg rotational delay for 10000RPM disk?

min: 0, max: 60s/10000=6ms, avg: 3ms

Transfer time: <0.1ms / page → more than 100MB/s



Sequential vs. Random Access

Bandwidth for Sequential Access (assuming 0.1ms/4KB):

0.04ms for 4KB → **100MB/s**

Bandwidth for Random Access (4KB):

0.5ms (seek time) + 3ms (rotational delay) + 0.04ms = 3.54ms

4KB/3.54ms → **1.16MB/s**

Flash

Secondary durable storage that support both *random* and *sequential* access

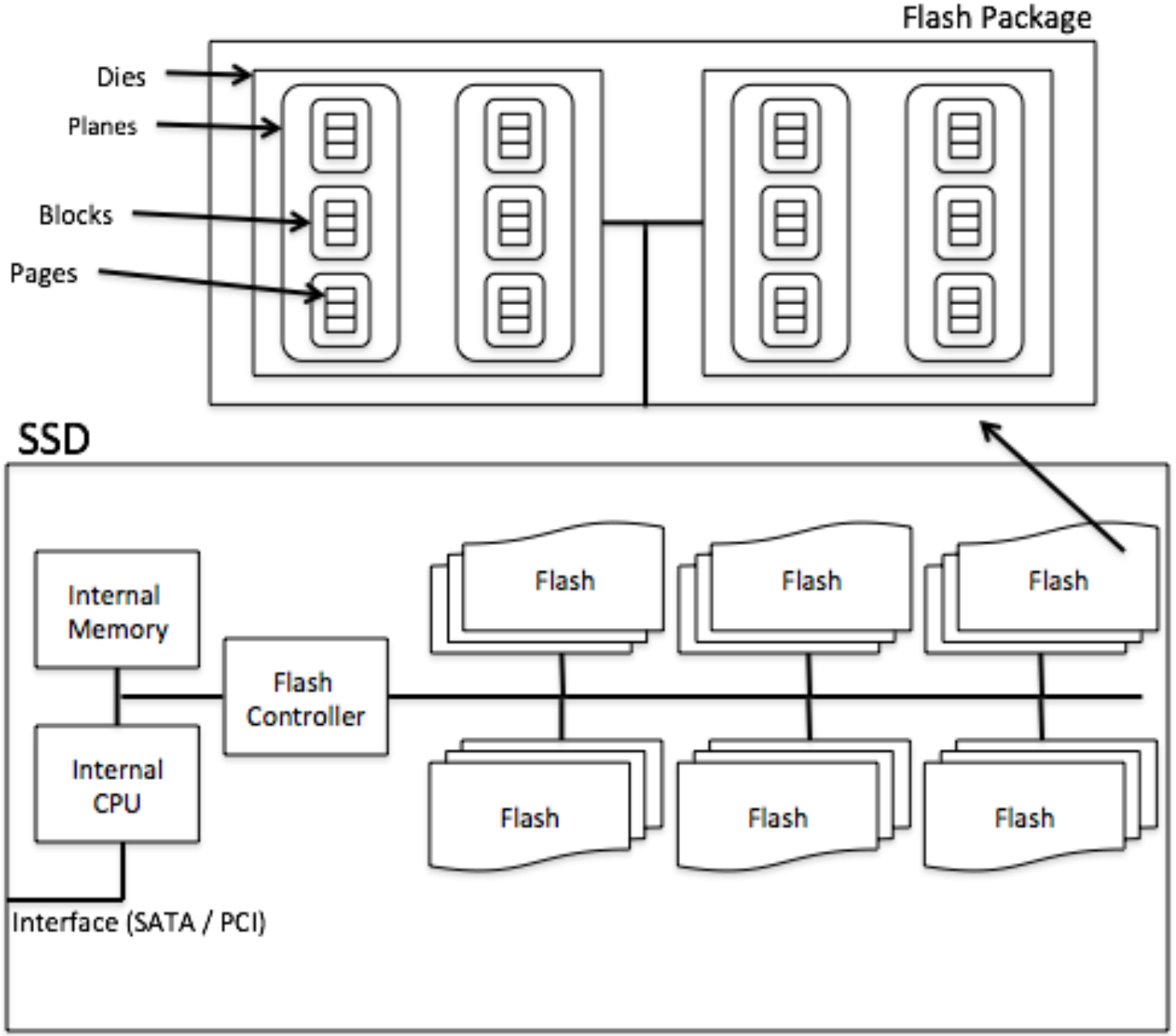
Data organized on pages (similar to disks) which are further grouped to erase blocks

Main advantage over disks: random read is now much more efficient

BUT: Not as fast random writes!

Goal: *avoid random writes*

The internals of flash



interconnected flash chips

no mechanical limitations

maintain the block API
compatible with disks layout

internal parallelism
for both read/write

complex software driver

Flash access time

… depends on:

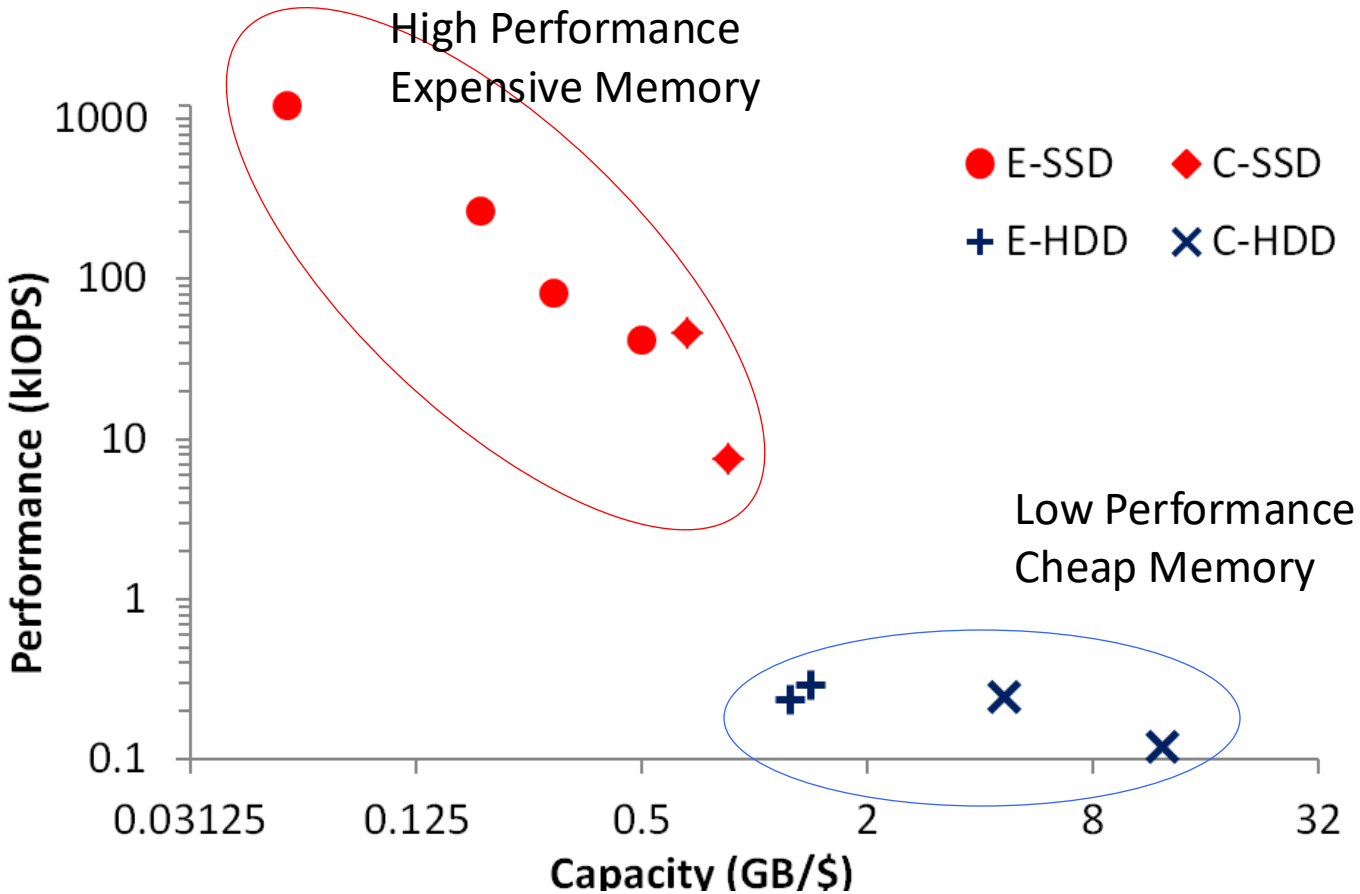
device organization (***internal parallelism***)

software efficiency (***driver***)

bandwidth of flash packages

the Flash Translation Layer (FTL), a complex device driver (firmware) which
tunes performance and device lifetime

Flash vs HDD



HDD

✓ Large - cheap capacity

✗ Inefficient random reads

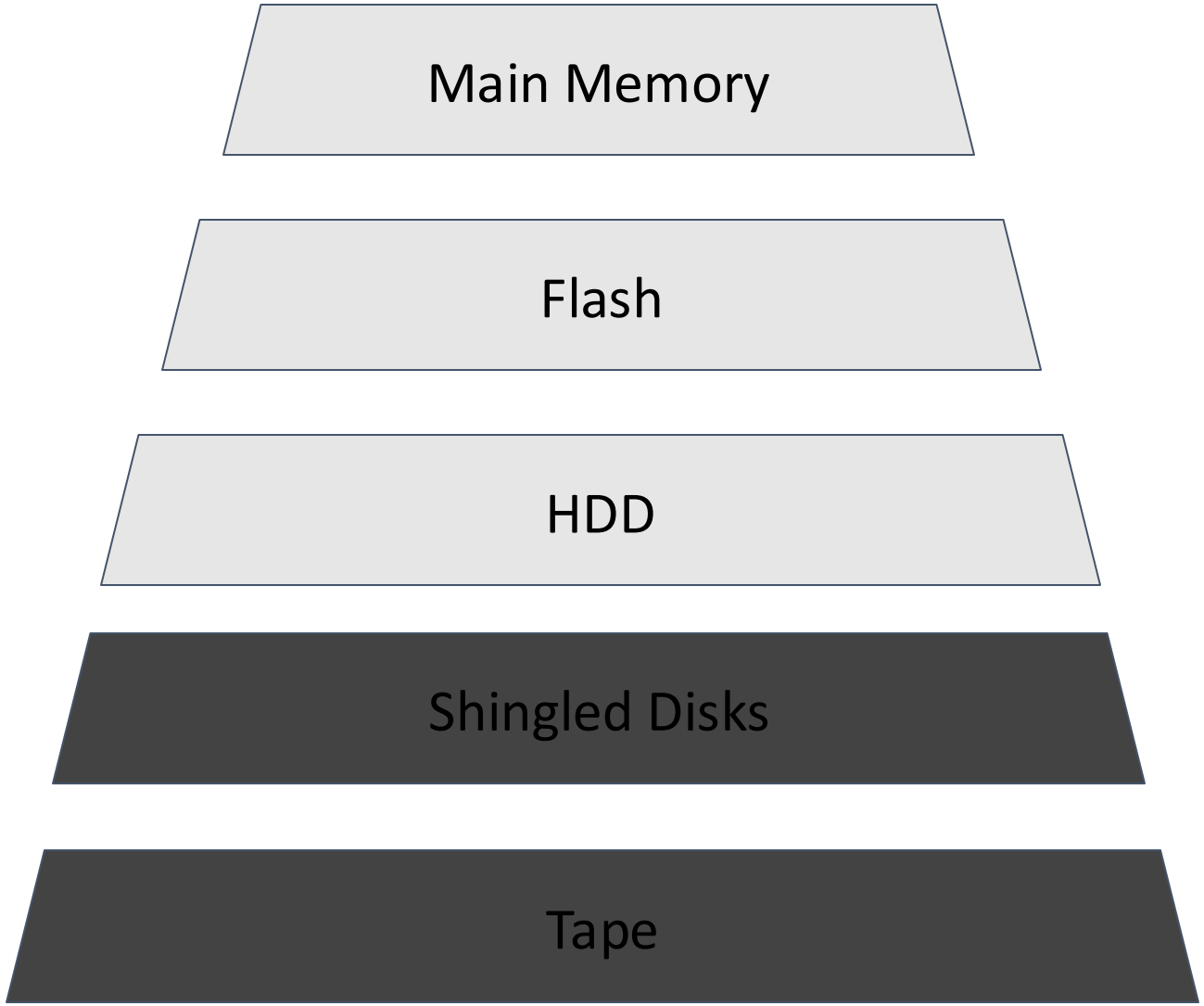
Flash

✗ Small - expensive capacity

✓ Very efficient random reads

✗ Read/Write Asymmetry

Storage Hierarchy



Tapes

Data size grows exponentially!

Cheaper capacity:

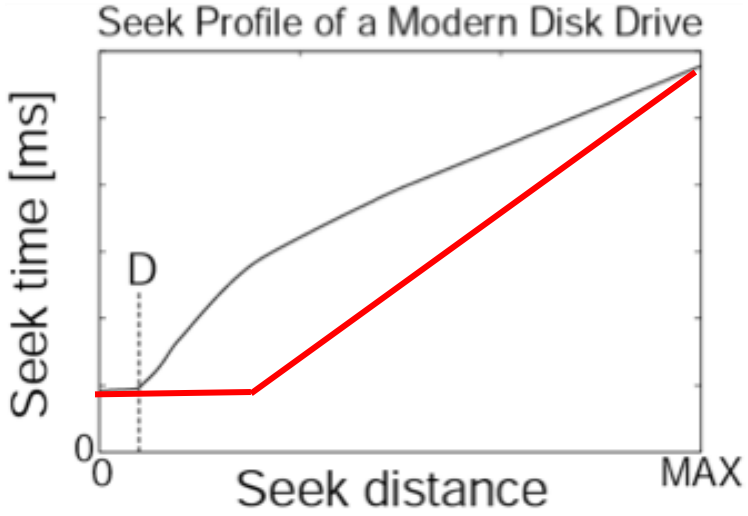
- Increase density (bits/in^2)
- Simpler devices

Tapes:

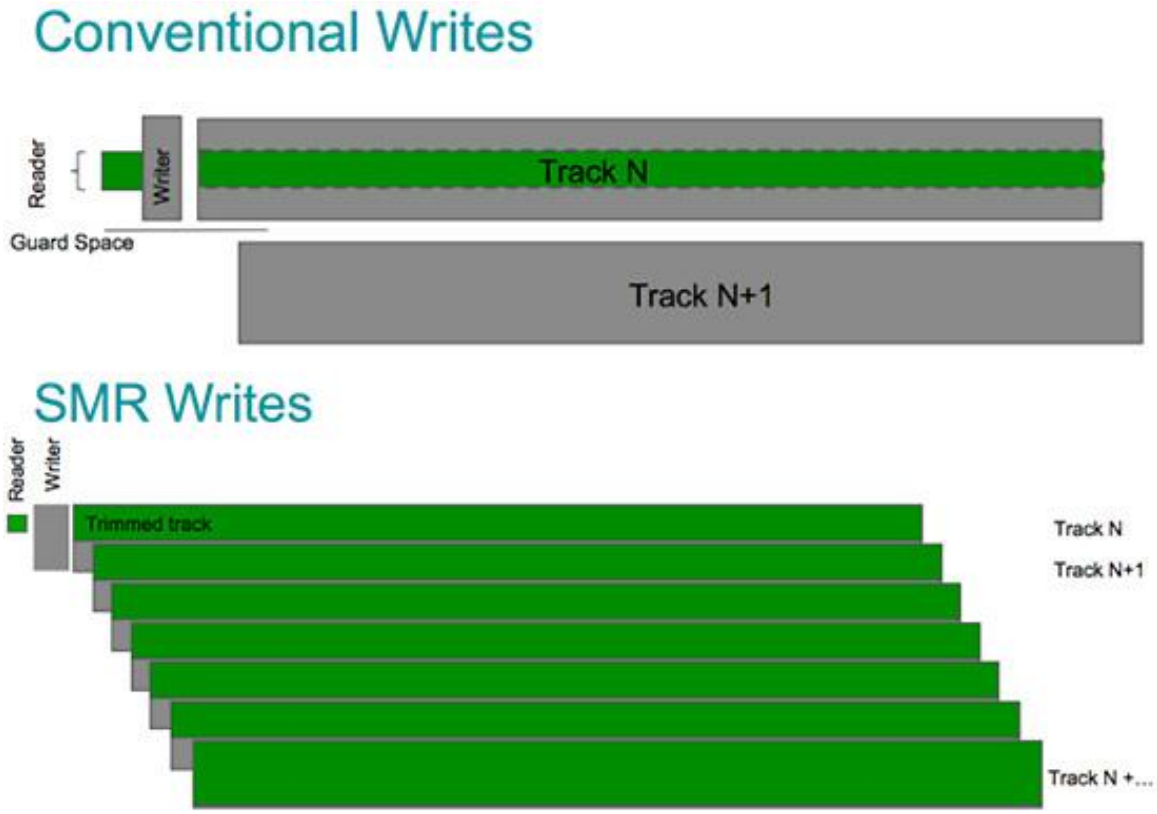
- Magnetic medium that allows only **sequential access** (yes like an old-school tape)!

Increasing disk density

Very difficult to differentiate between tracks
“settle” time becomes

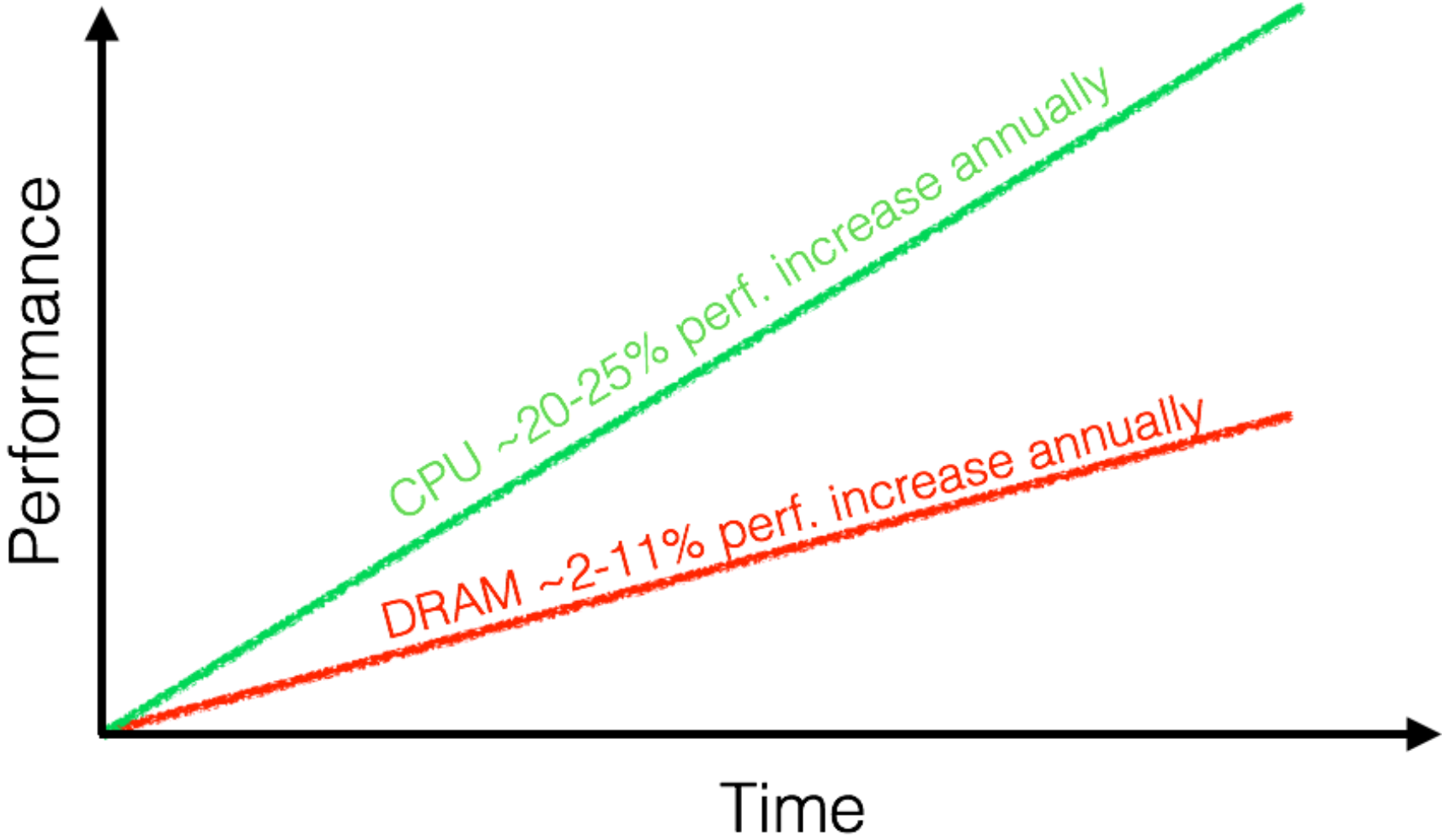


Writing a track affects neighboring tracks
Create different readers/writers
Interleave writes tracks

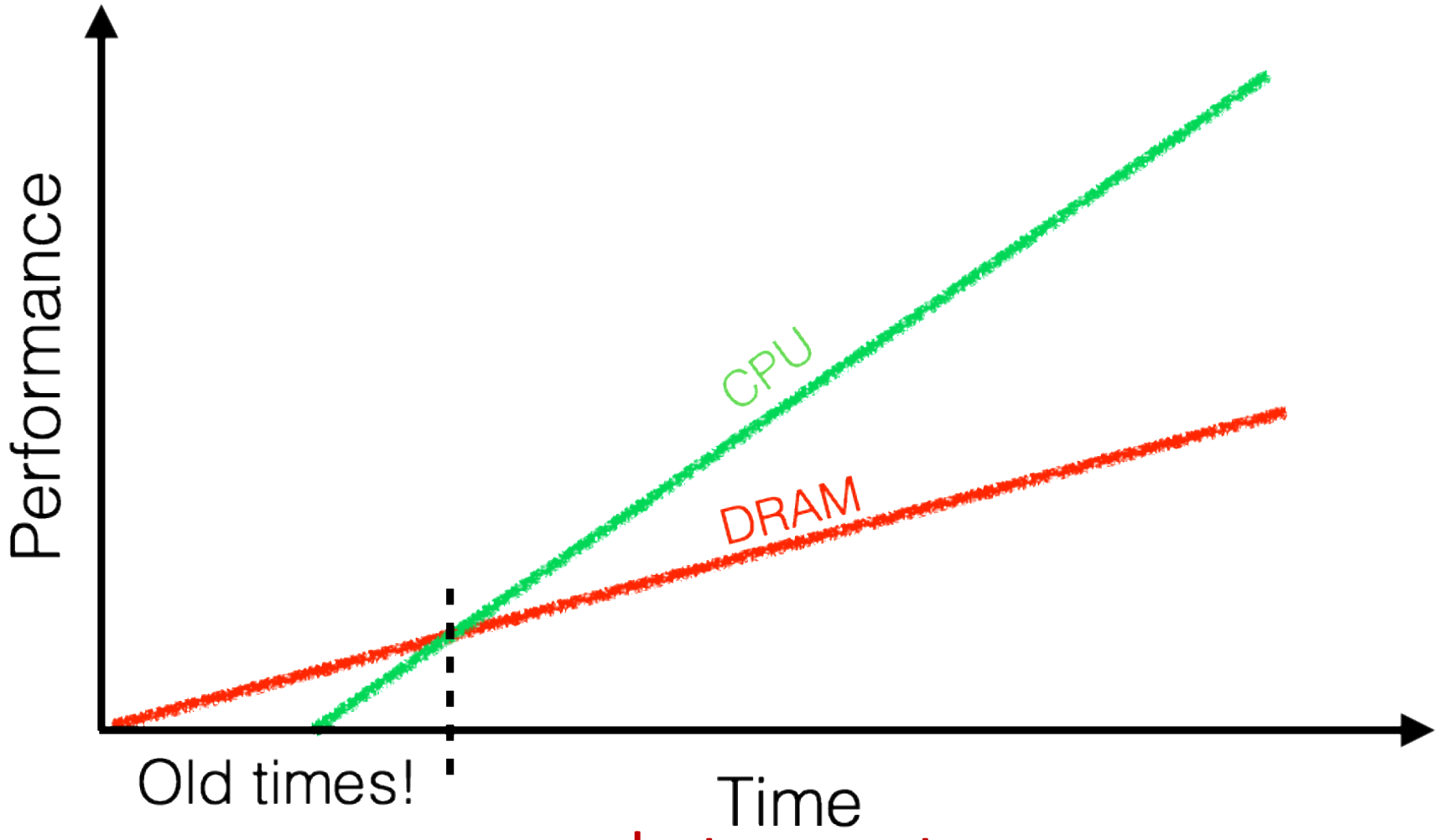


Memory & Storage Walls

Memory Wall


Performance
CPU ~20-25% perf. increase annually
DRAM ~2-11% perf. increase annually
Time

Memory Wall



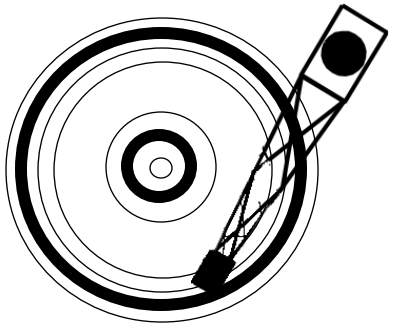
every byte counts

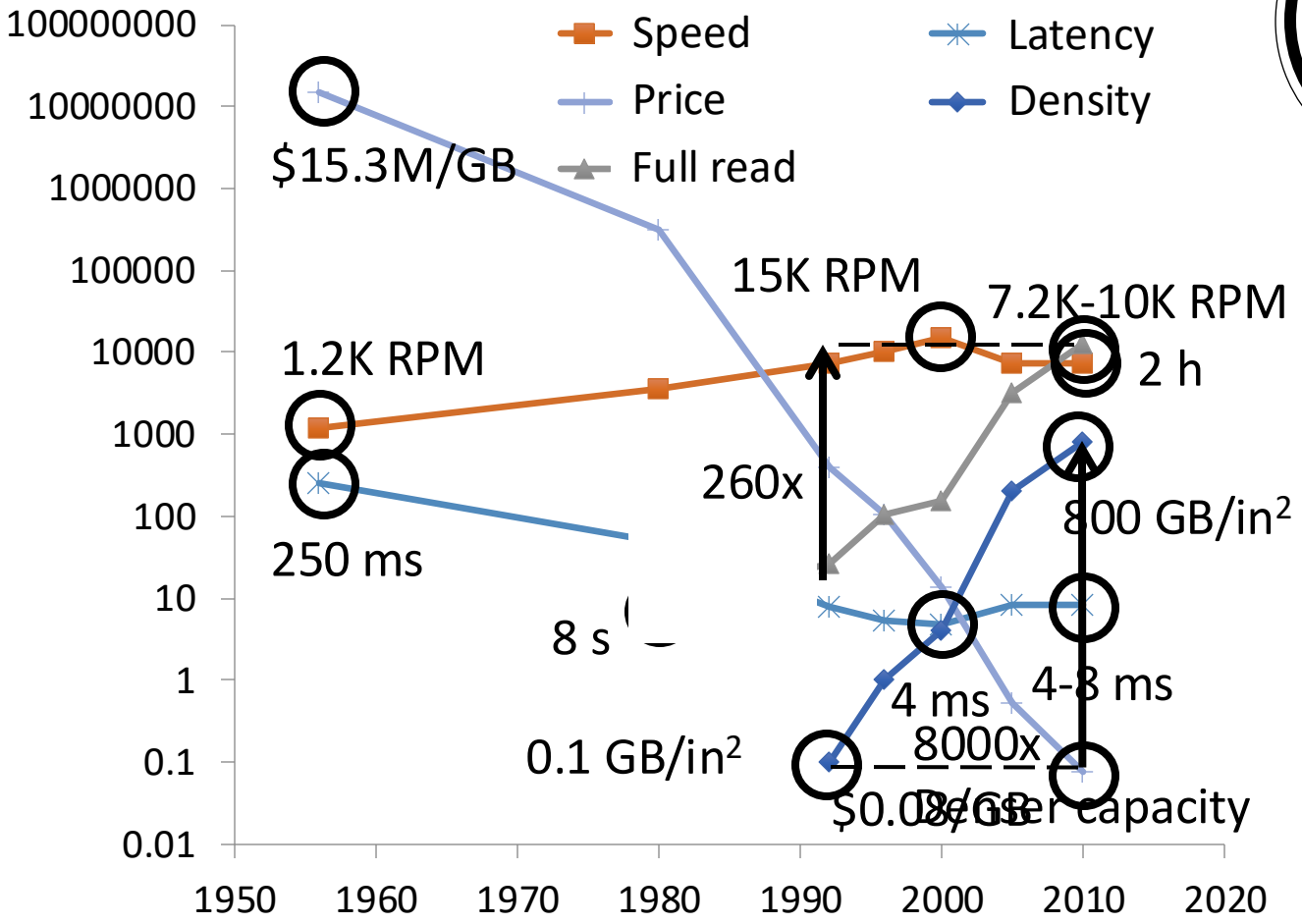
Storage Wall

HDD

- ✓ capacity
- ✓ sequential access
- × random access
- × latency plateaus

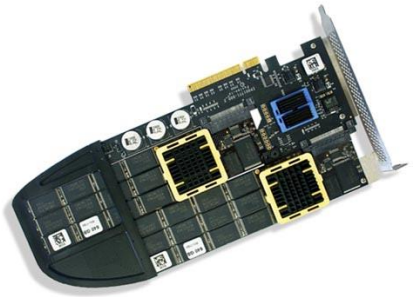
Evolution of hard disks





disks become larger but – relatively – slower

Storage Wall



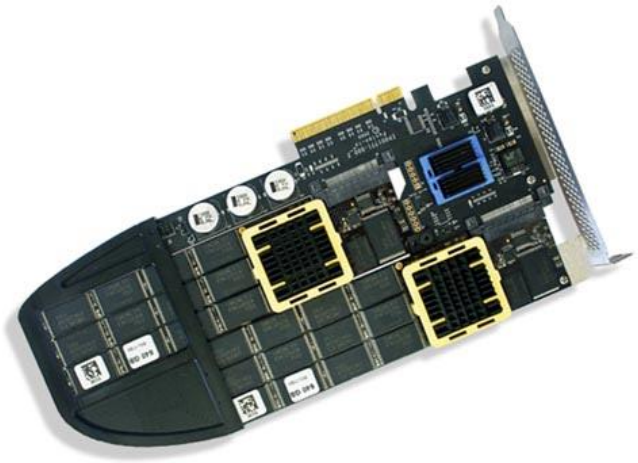
HDD
- ✓ capacity
- ✓ sequential access
- × random access
- × latency plateaus

SSD (Single Level Cell)
- ✓ random reads
- ✓ low latency
- × capacity
- × endurance
- × read/write asymmetry

SSD (Multi Level Cell)
- ✓ capacity
- × endurance (worse)

BOSTON UNIVERSITY

“Tape is Dead, Disk is Tape, Flash is Disk” [Jim Gray 2007]



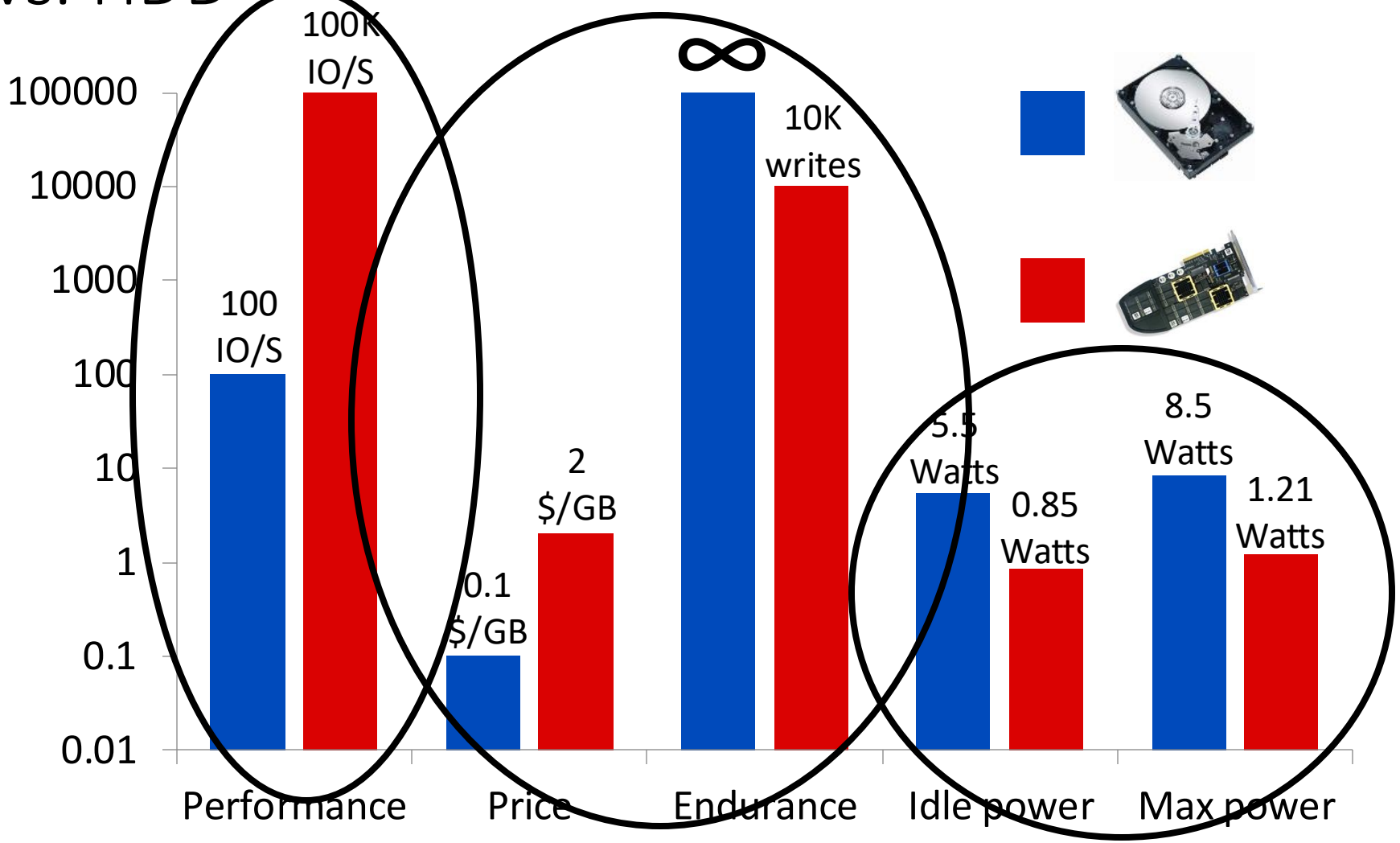
- Storage without mechanical limitations

Several technologies

- Flash
- Phase Change Memory (IBM)
- Memristor (HP)

flash vs. hard disks?

Flash vs. HDD

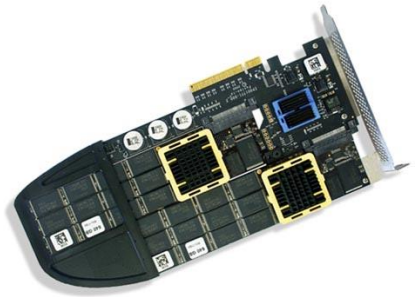


read, update, storage tradeoffs

Storage Wall

HDD
- ✓ capacity
- ✓ sequential access
- × random access
- × latency plateaus

SSD (Single Level Cell)
- ✓ random reads
- ✓ low latency
- × capacity
- × endurance
- × read/write asymmetry



SSD (Multi Level Cell)
- ✓ capacity
- × endurance (worse)

HDD (Shingled Magnetic Rec.)
- ✓ capacity
- × read/write asymmetry

every byte counts

Technology Trends & Research Challenges

(1) From fast **single cores** to **increased parallelism**

(2) From **slow storage** to **efficient random reads**

(3) From **infinite** endurance to **limited endurance**

(4) From **symmetric** to **asymmetric read/write performance**

Technology Trends & Research Challenges

How to exploit increasing parallelism (in compute and storage)?

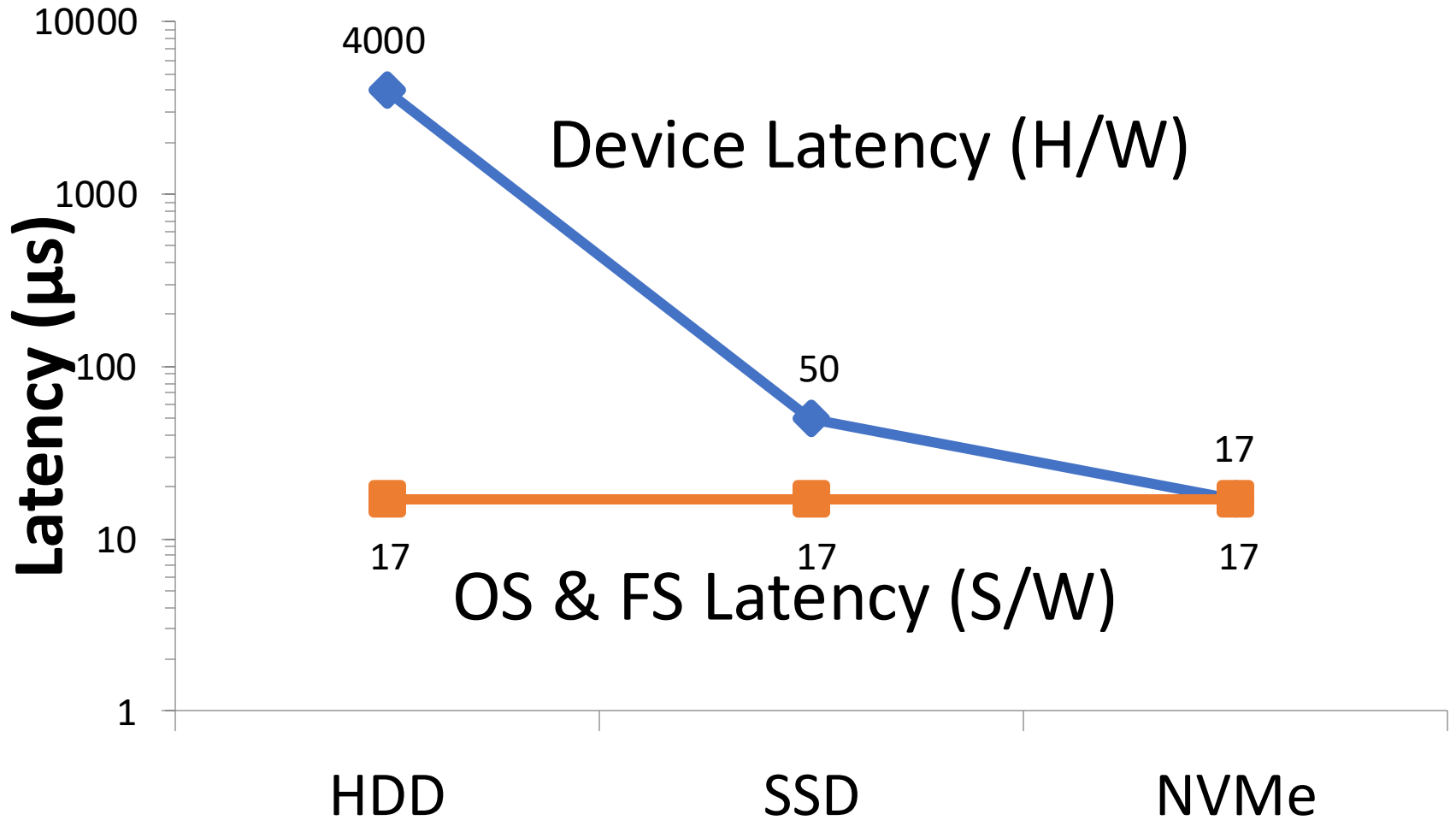
How to redesign systems for efficient random reads?

e.g., no need to aggressively minimize index height!

How to reduce ***write amplification*** (physical writes per logical write)?

How to write algorithms for asymmetric storage?

Even faster devices are available (NVMe)
How to use them when the software stack is too slow?



BOSTON UNIVERSITY

CS 561: **Data Systems Architectures**

class 14

Data Systems on Modern Hardware:
Multi-cores, Solid-State Drives, and Non-Volatile Memories

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/