CS 561: Data Systems Architectures

BOSTON UNIVERSITY

class 15

Data Systems on Modern Hardware:

Multi-cores, Solid-State Drives, and Non-Volatile Memories

Prof. Manos Athanassoulis

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

with slides developed at Harvard

Some class logistics in light of … reality

https://forms.gle/hWHhu6YVbr4ZDicq8

See also Zoom chat for the link!

Please respond now!

Let’s also try the raise-hand option in a different way!

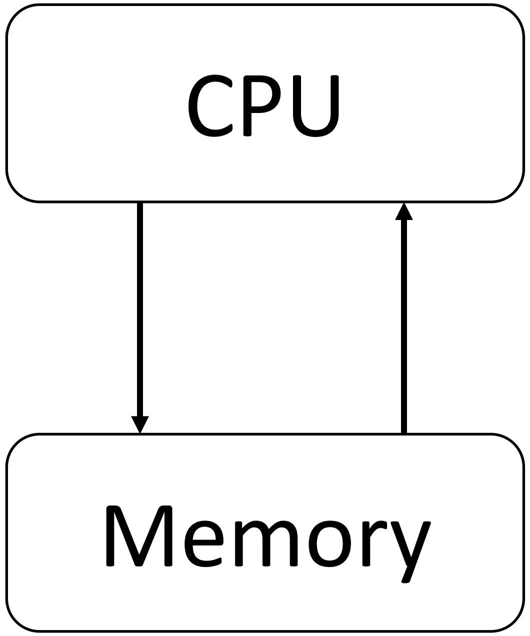
Everyone who is connected via Ethernet (not wifi) please raise your hand

Now, everyone who is connected via wifi (not Ethernet) please do so

BOSTON UNIVERSITY

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?

BOSTON UNIVERSITY

Moore’s law

Often expressed as:

“*X* doubles every 18-24 months”

where X is:

“performance”

CPU clock speed

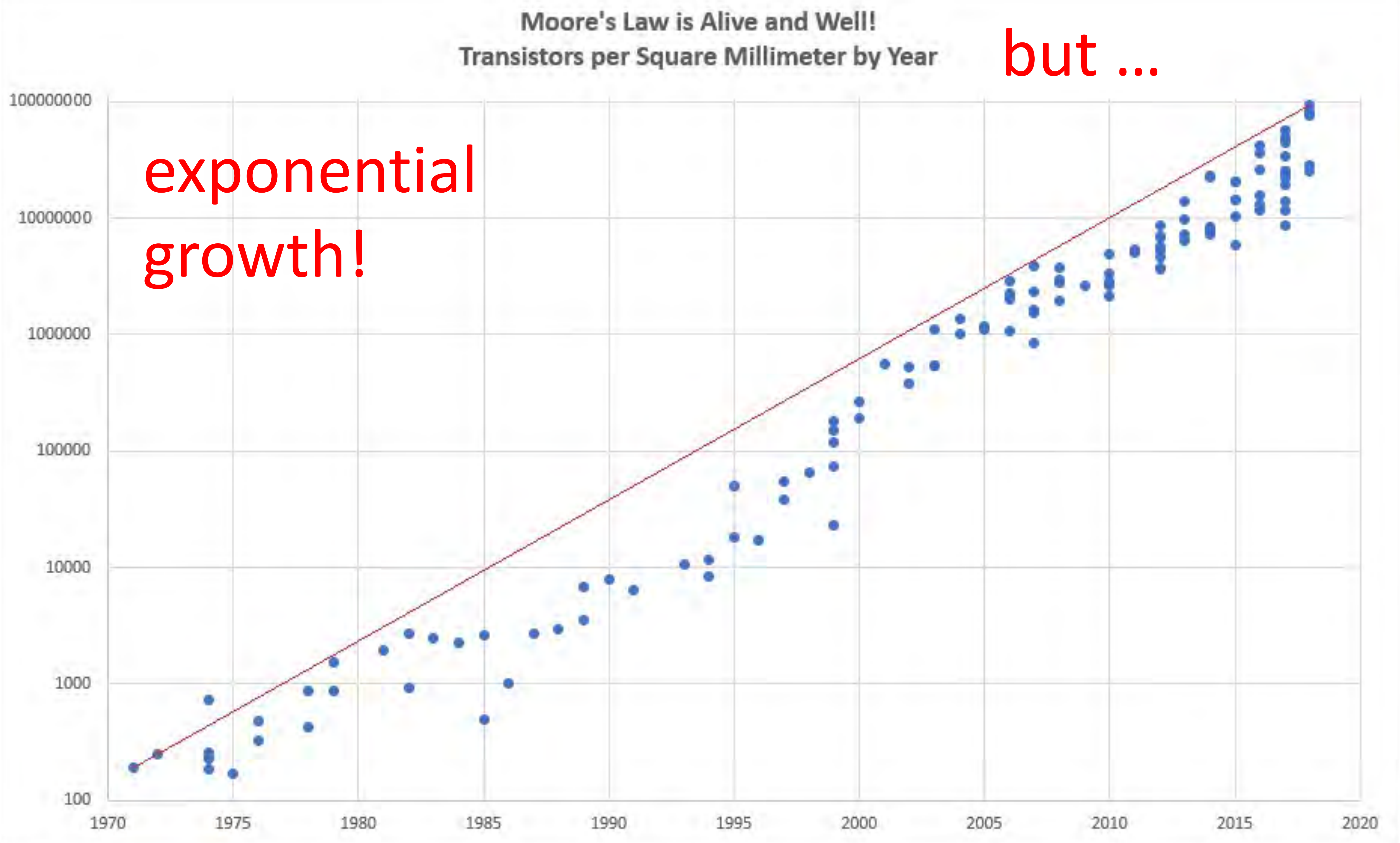
the number of transistors per chip

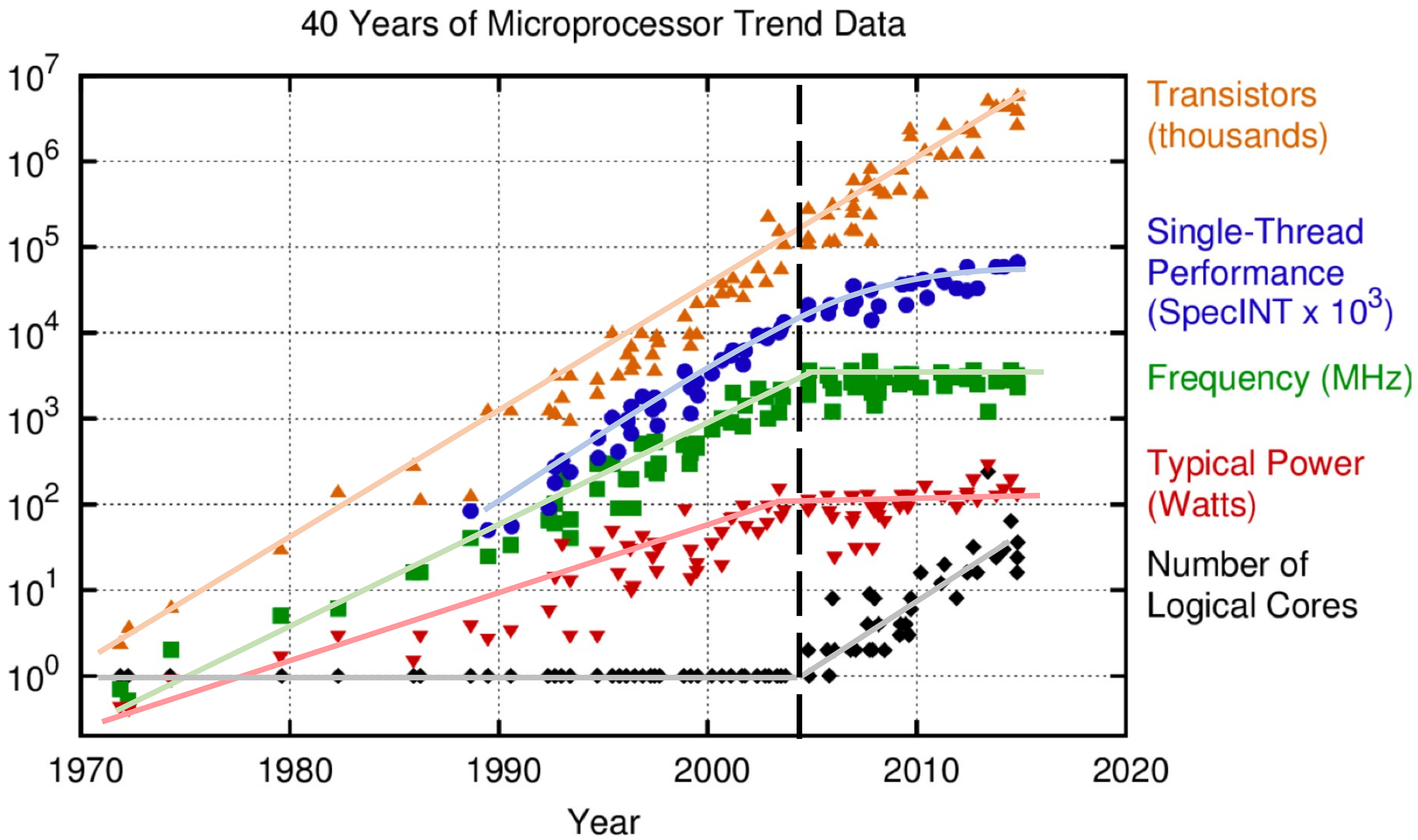
which one is it?

BOSTON UNIVERSITY

but …

exponential growth!





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

BOSTON UNIVERSITY

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**)

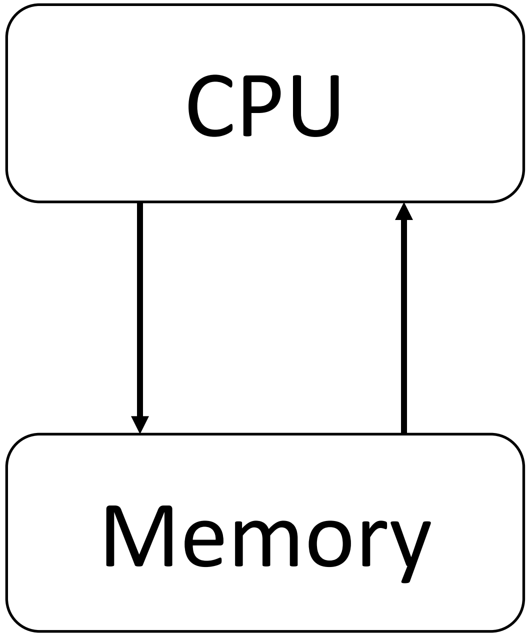
<u>Research Challenges</u>

how to handle them?

how to parallel program?

Compute, Memory, and Storage Hierarchy

Traditional von-Neuman computer architecture



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

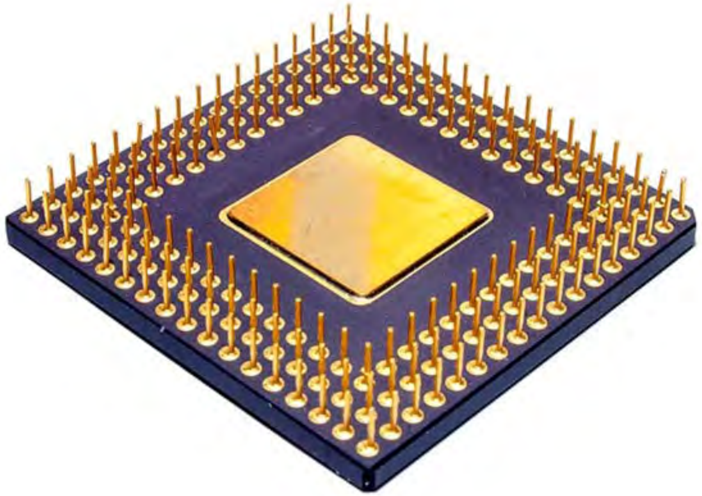
not always!

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

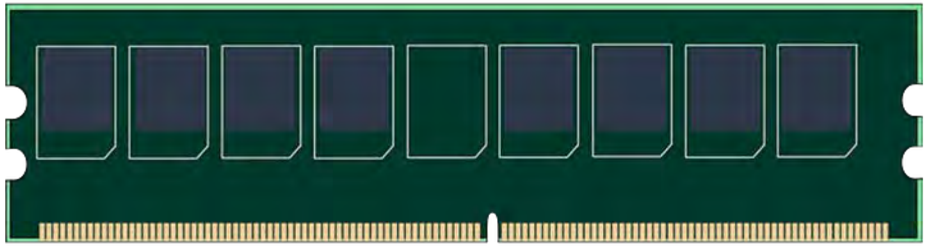
is this the case?

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

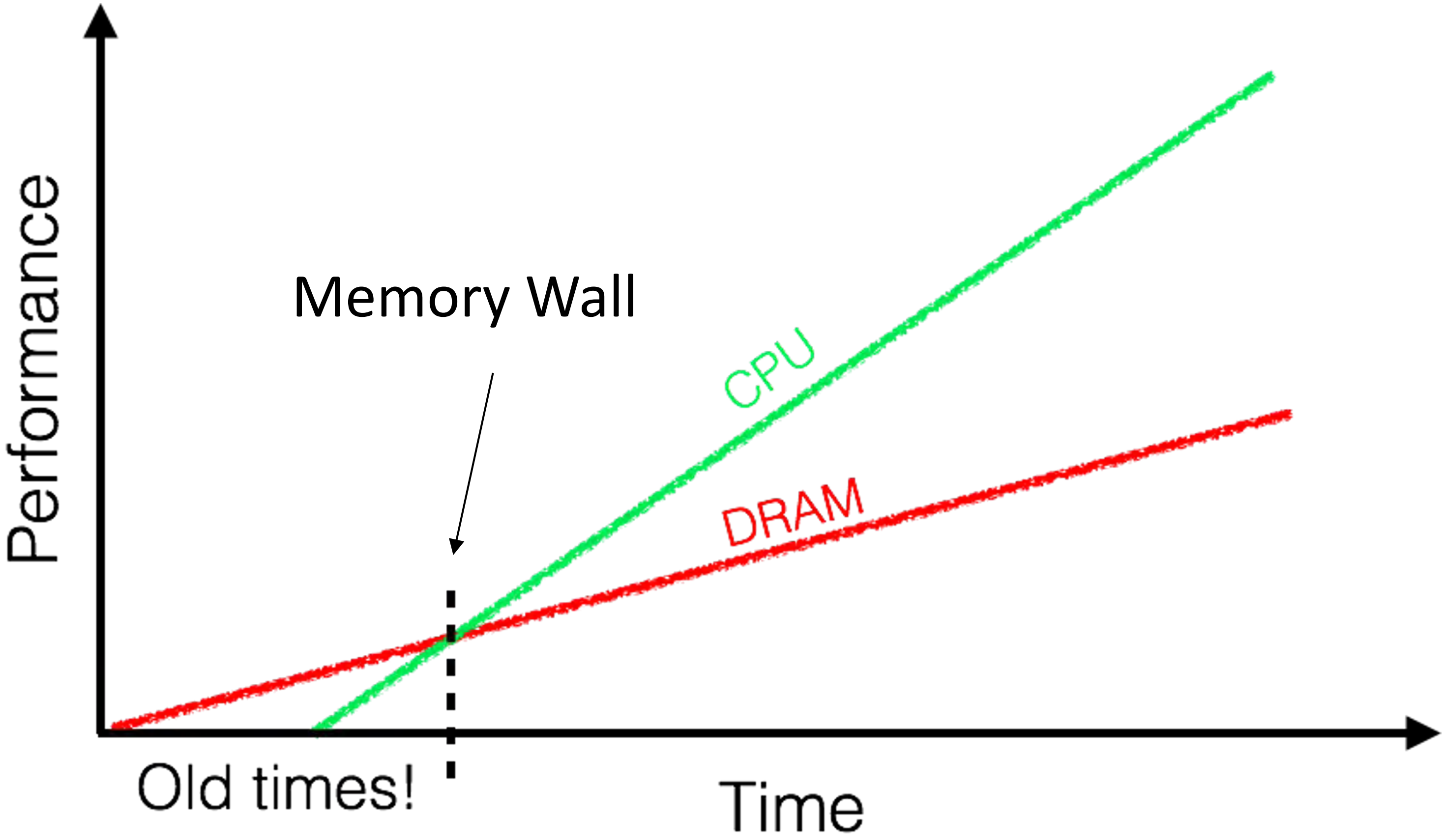
does it have enough capacity?



?



Which one is faster?

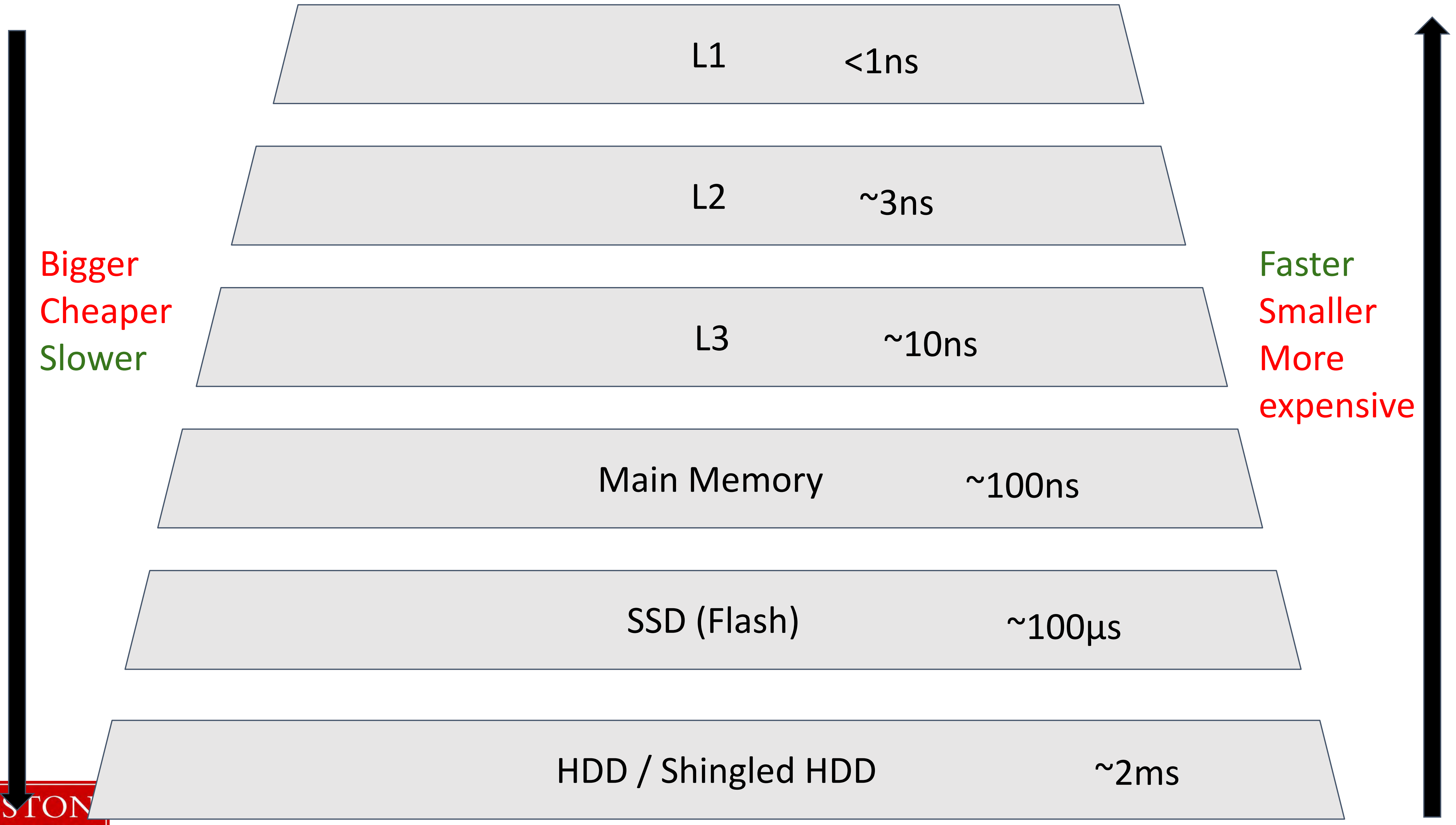


As the gap grows, we need a *deep* ***memory hierarchy***

A single level of main memory is not enough

We need a ***memory hierarchy***

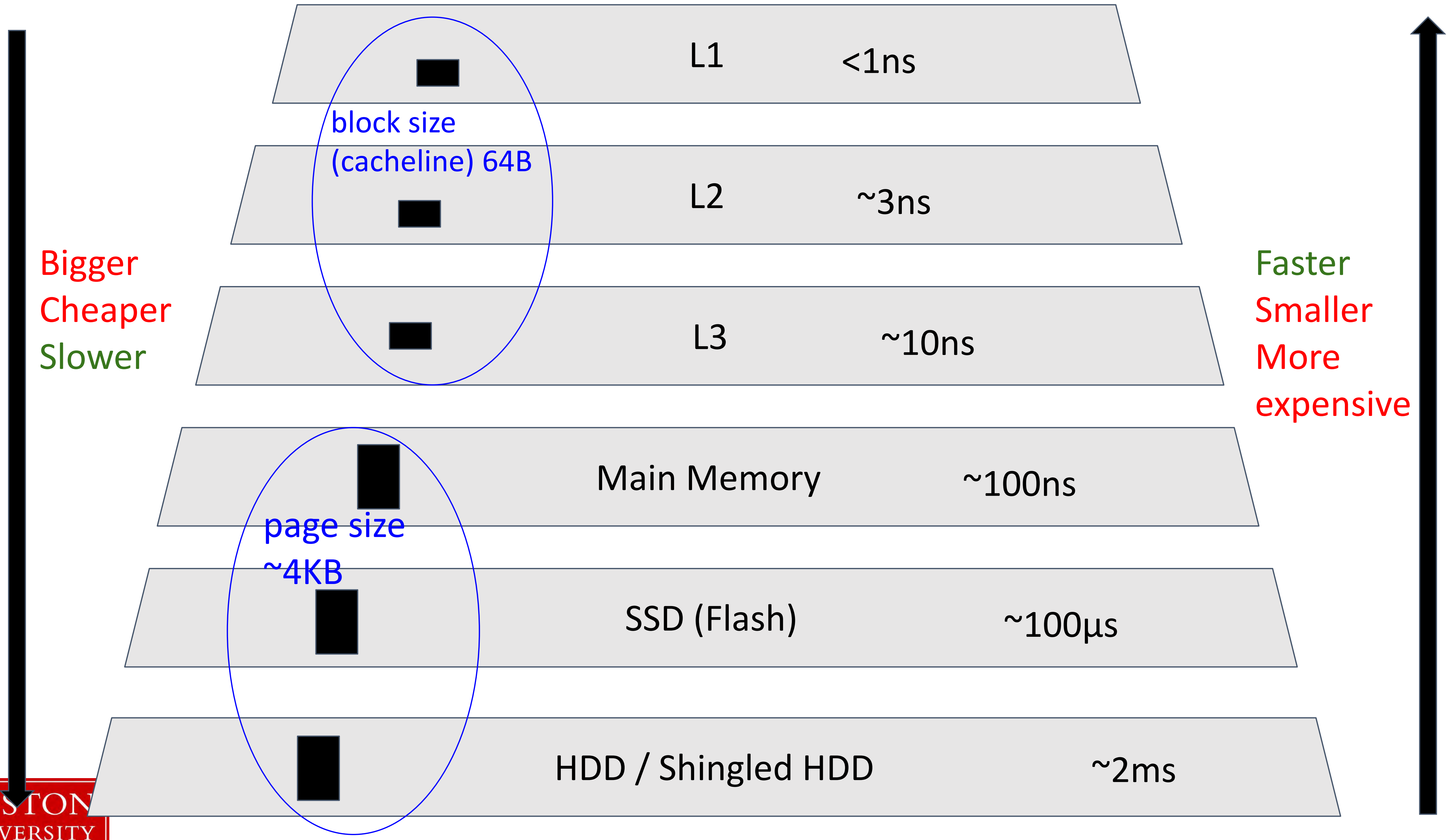
What is the memory hierarchy ?



L1
<1ns
L2
~3ns
L3
~10ns
Main Memory
~100ns
SSD (Flash)
~100µs
HDD / Shingled HDD
~2ms
Bigger
Cheaper
Slower
Faster
Smaller
More
expensive

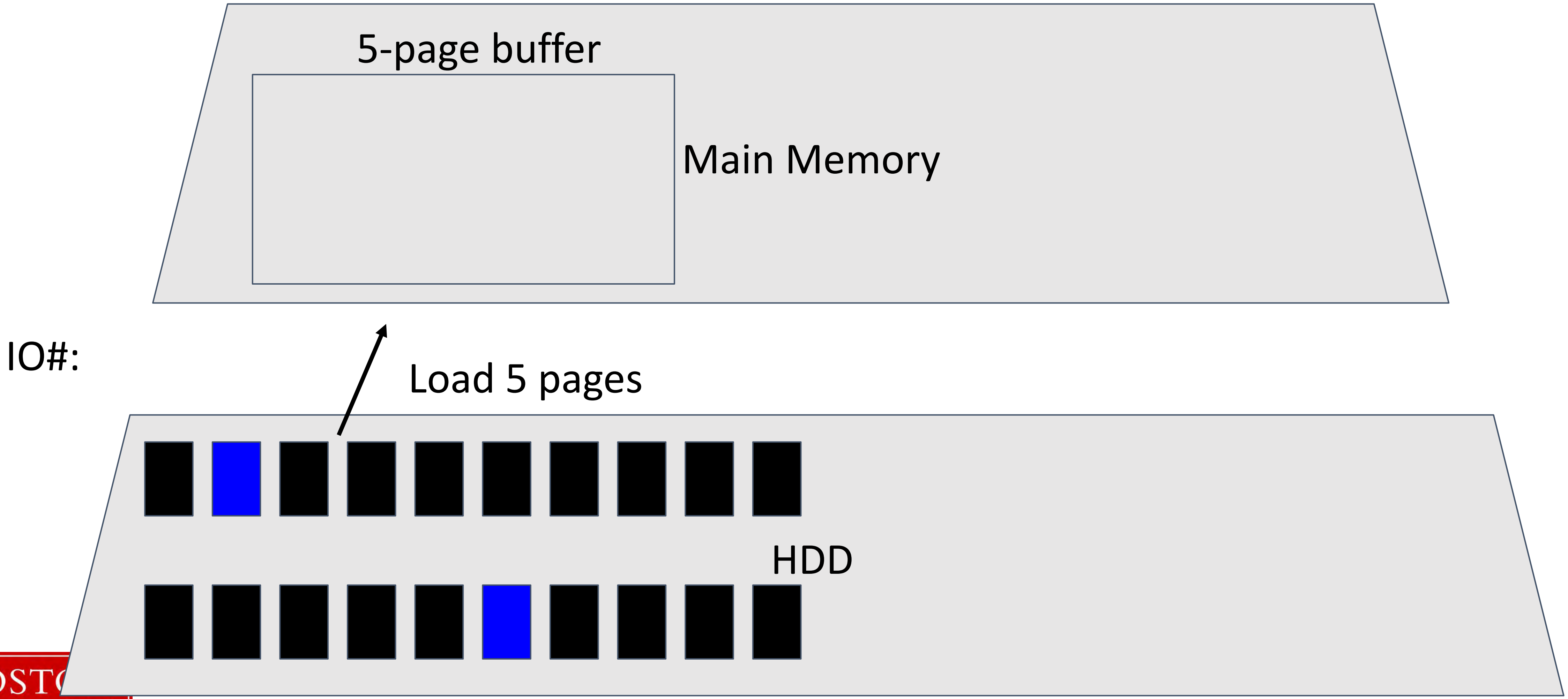
Access Granularity

BOSTON UNIVERSITY

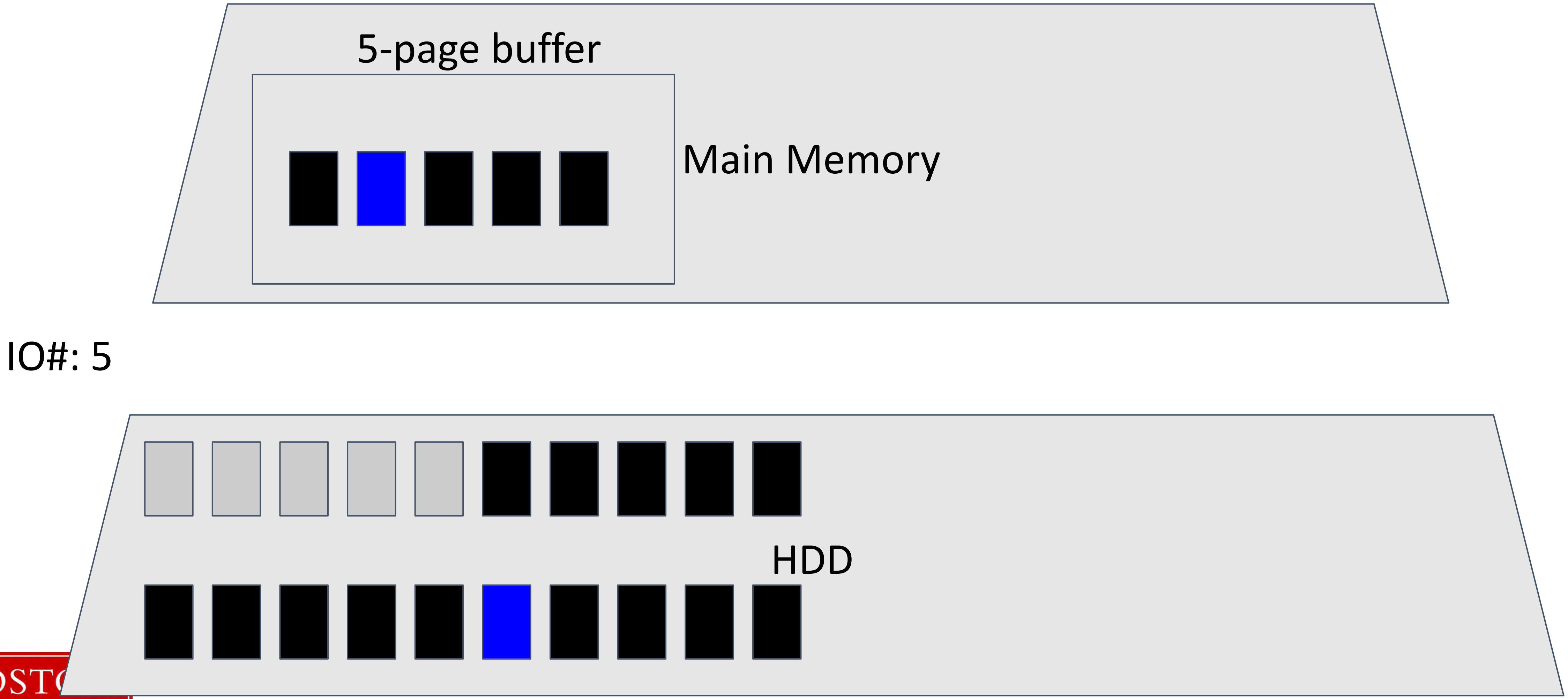


L1
<1ns
block size
(cacheline) 64B
L2
~3ns
L3
~10ns
Main Memory
~100ns
page size
~4KB
SSD (Flash)
~100µs
HDD / Shingled HDD
~2ms
Bigger
Cheaper
Slower
Faster
Smaller
More
expensive

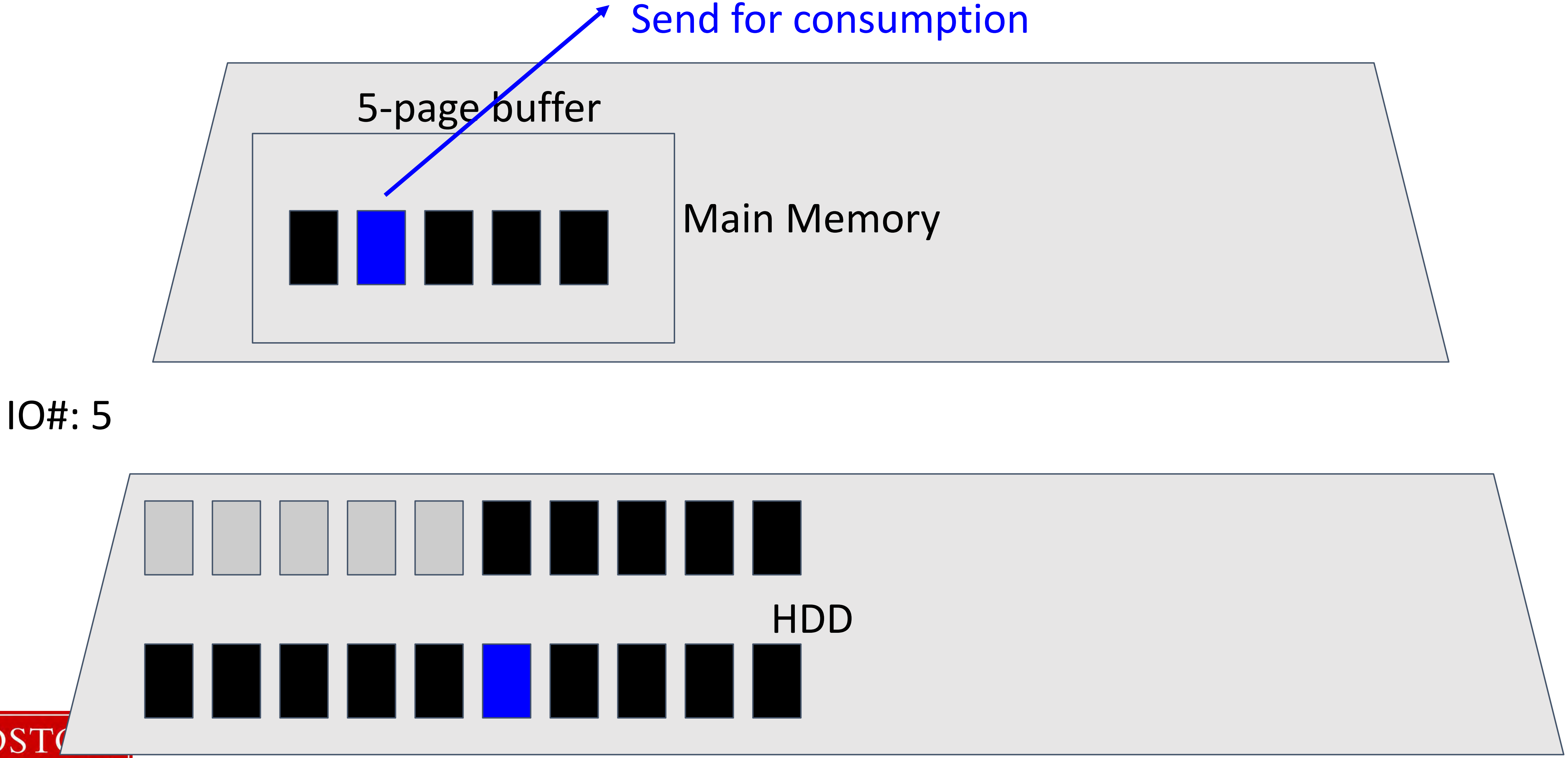
IO cost: Scanning a relation to select 10%



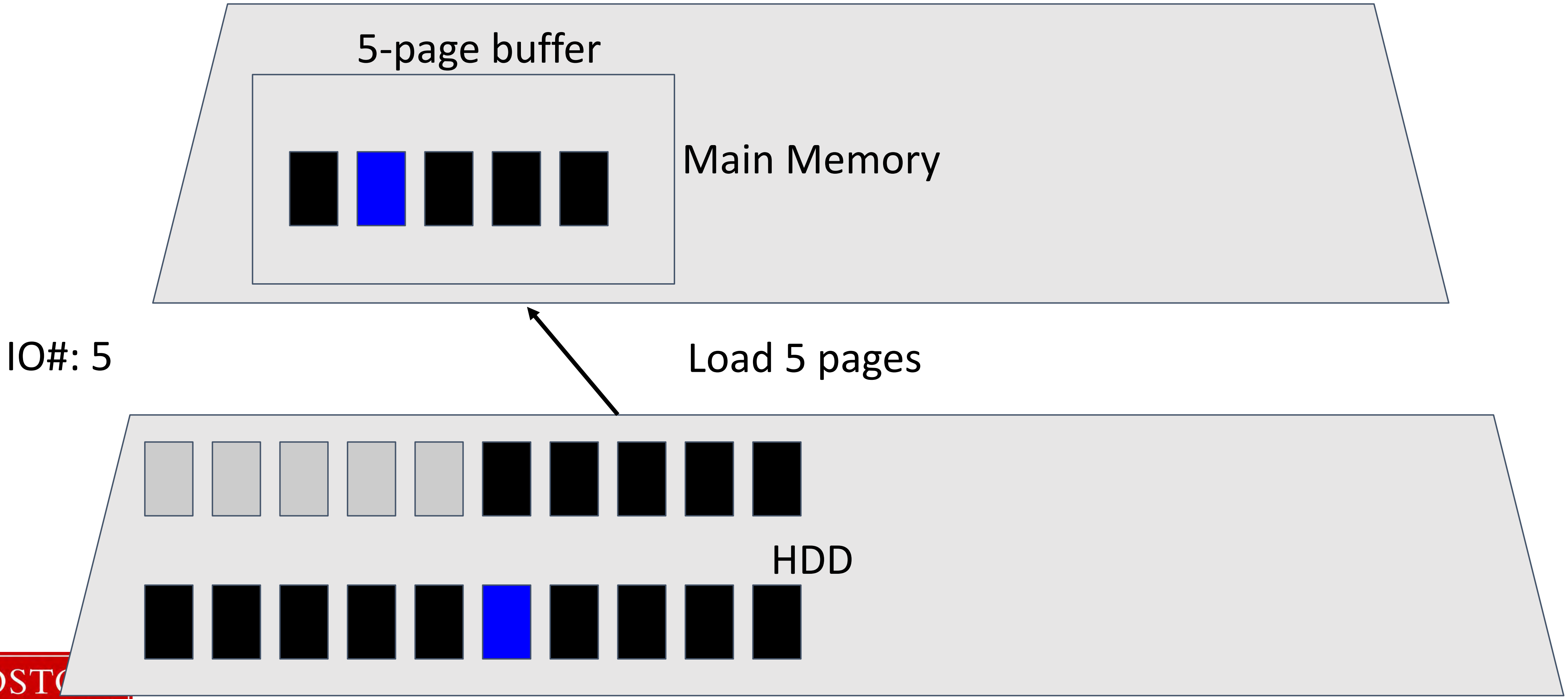
IO cost: Scanning a relation to select 10%



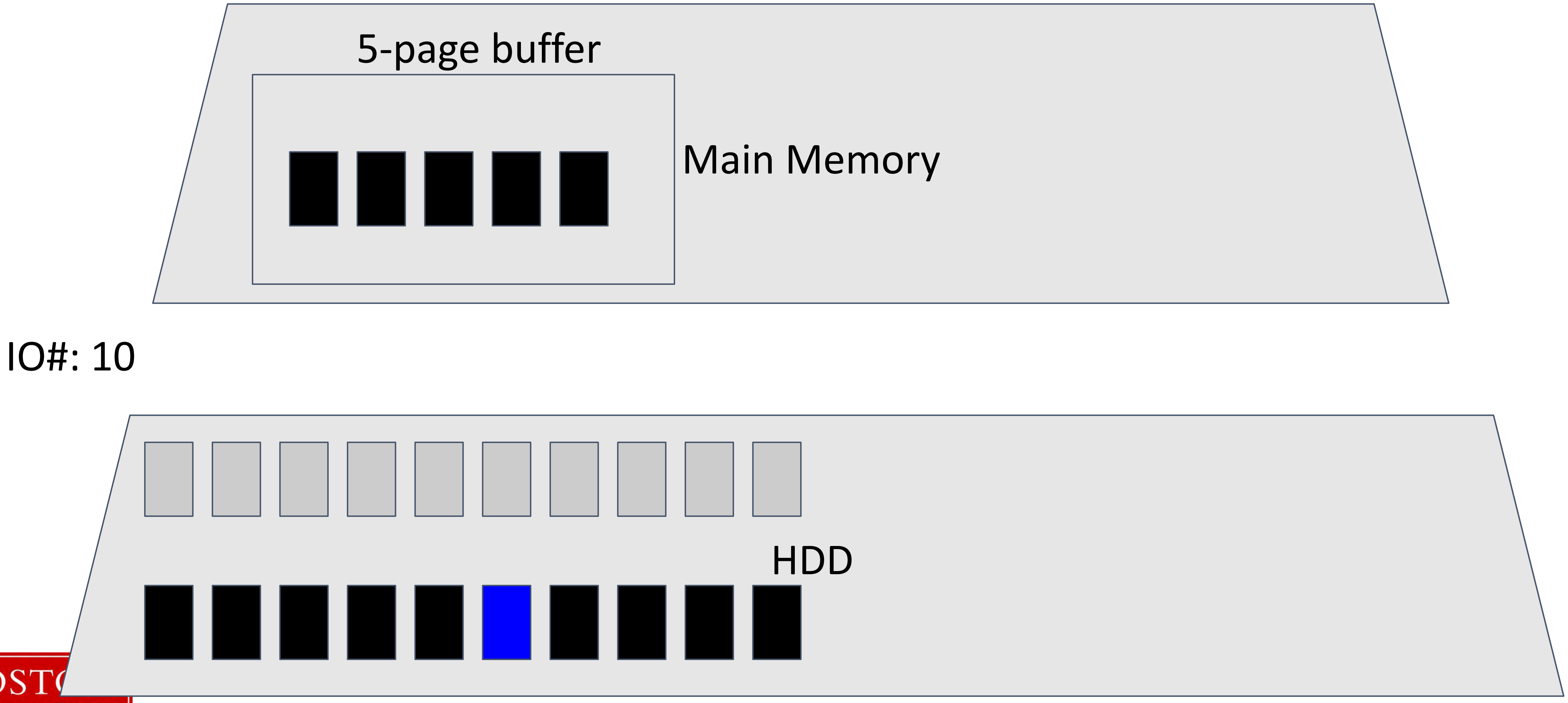
IO cost: Scanning a relation to select 10%



IO cost: Scanning a relation to select 10%

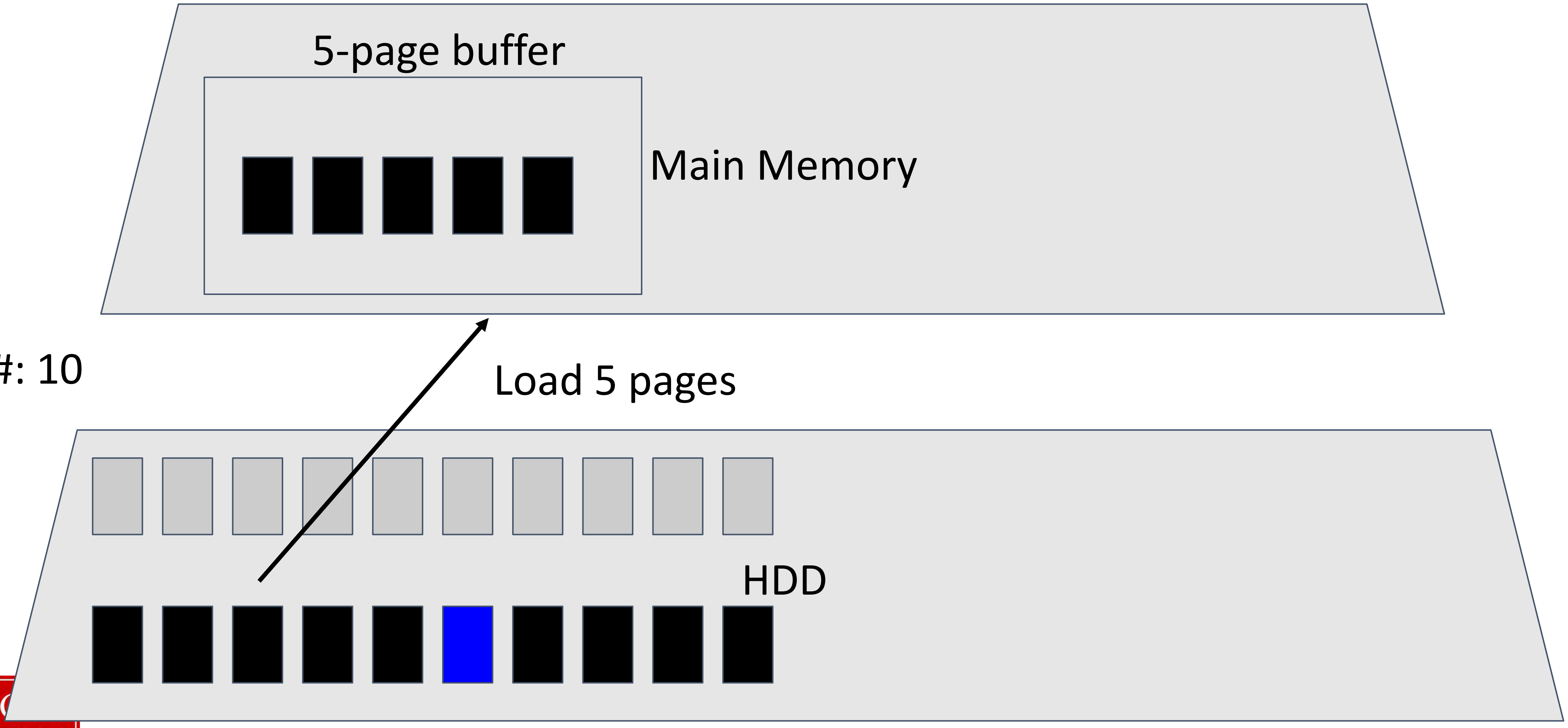


IO cost: Scanning a relation to select 10%

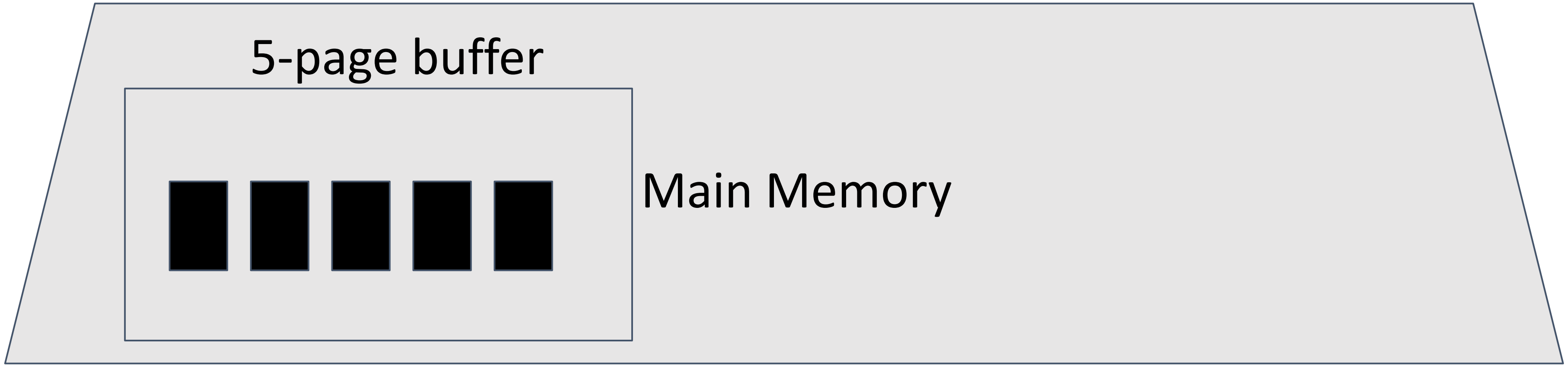


IO cost: Scanning a relation to select 10%

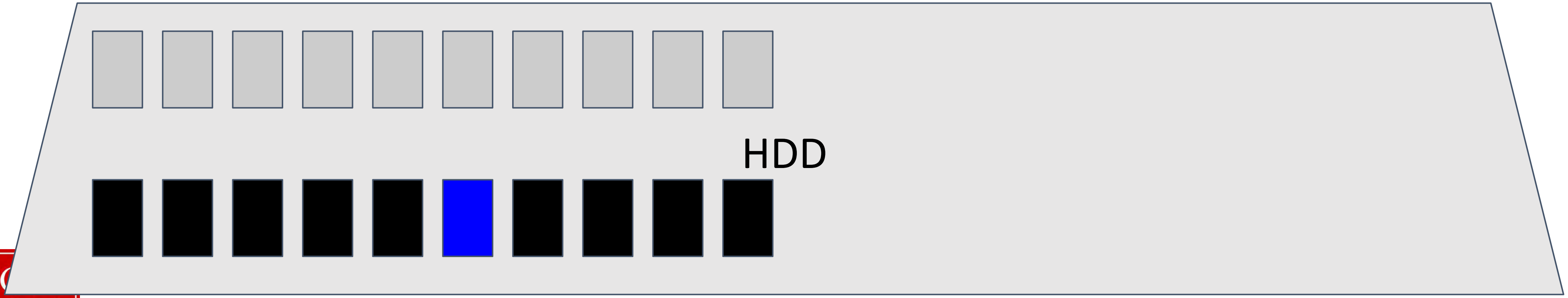
IO#: 10



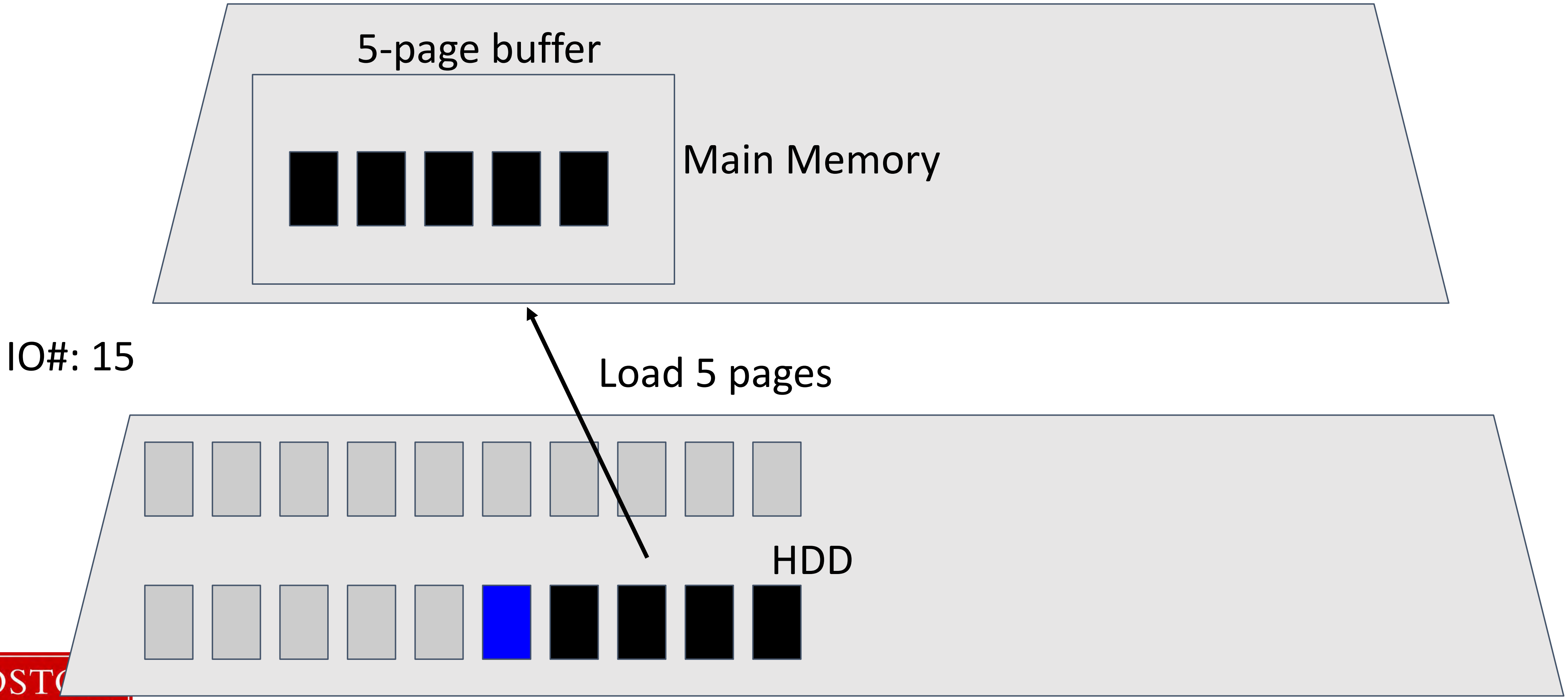
IO cost: Scanning a relation to select 10%



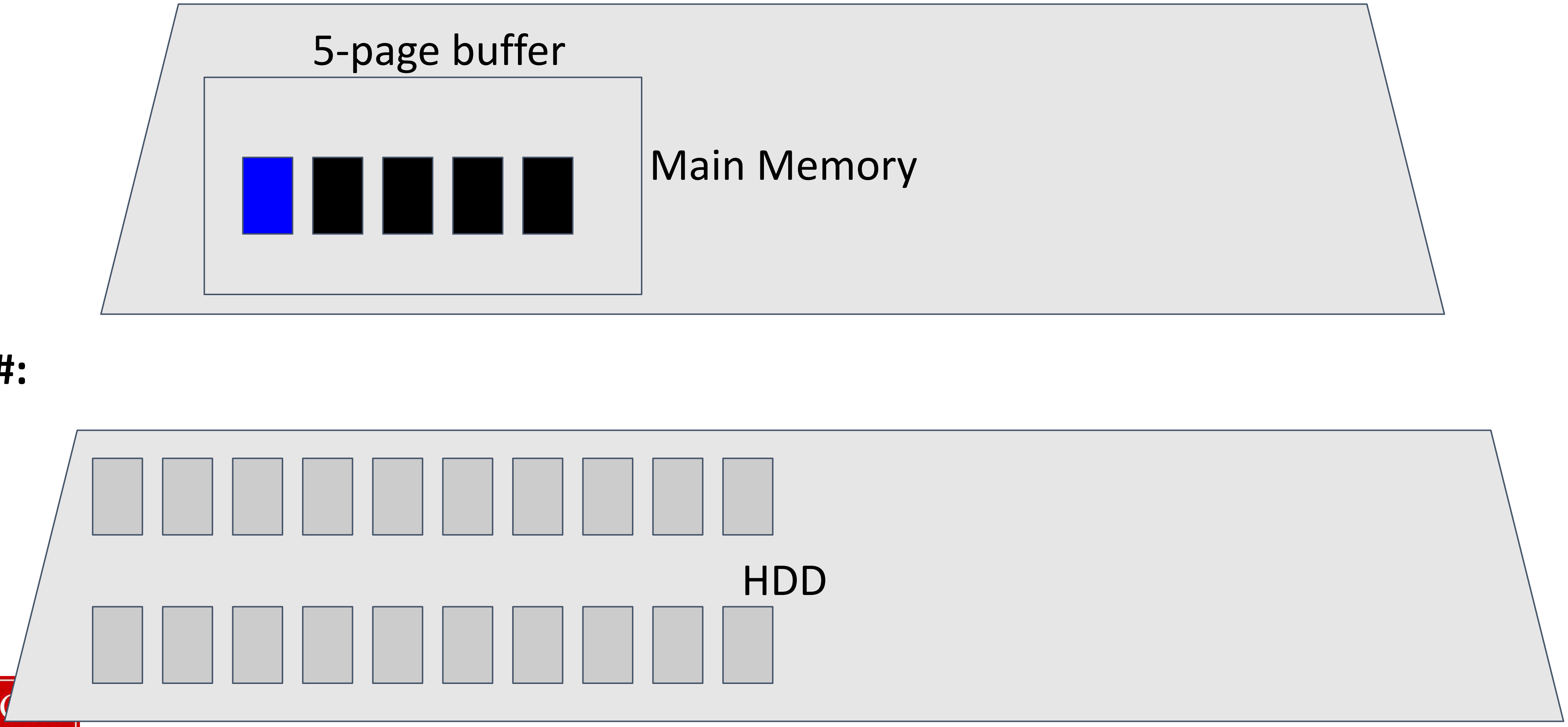
IO#: 15



IO cost: Scanning a relation to select 10%

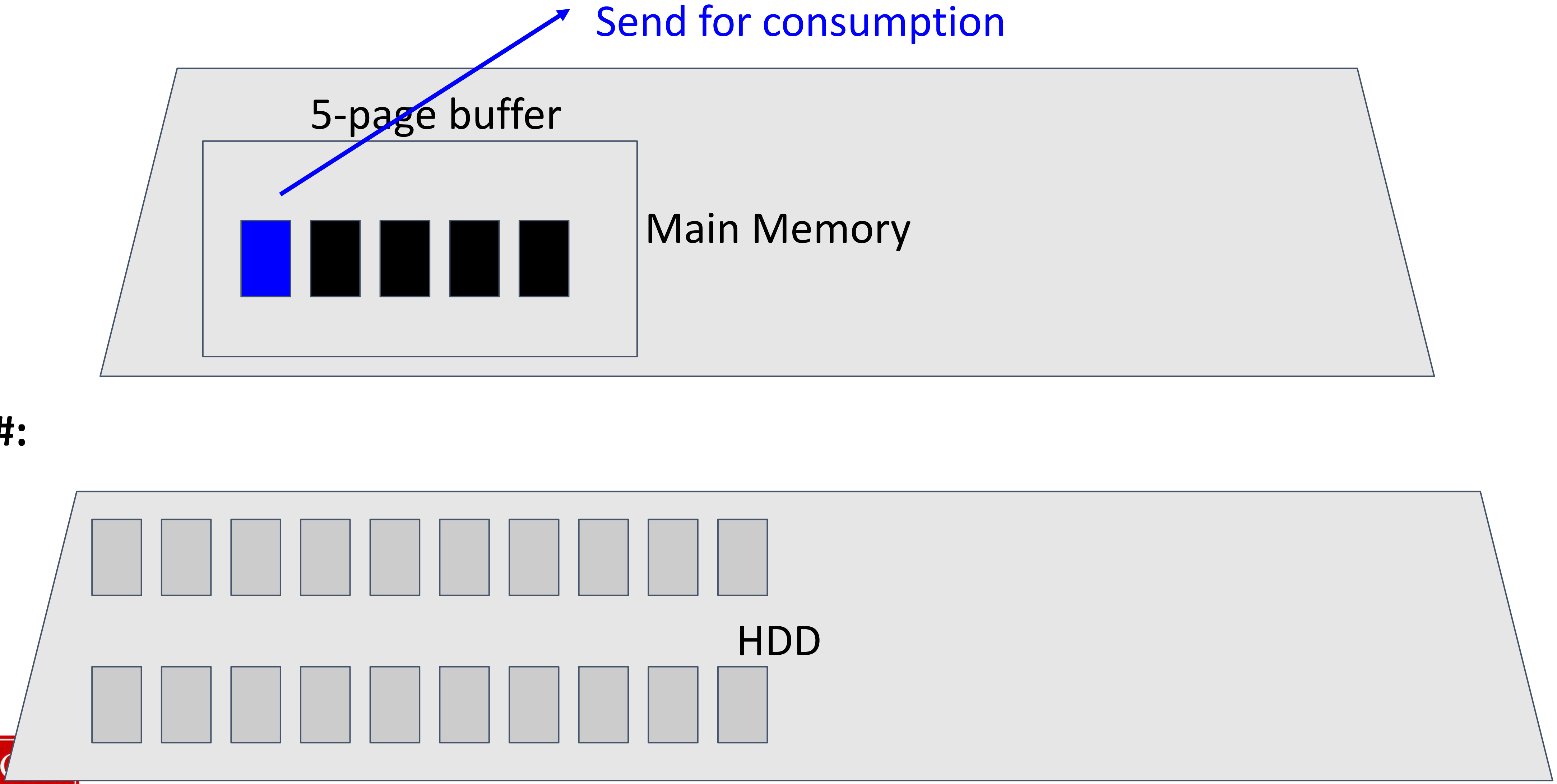


IO cost: Scanning a relation to select 10%



IO#: 20

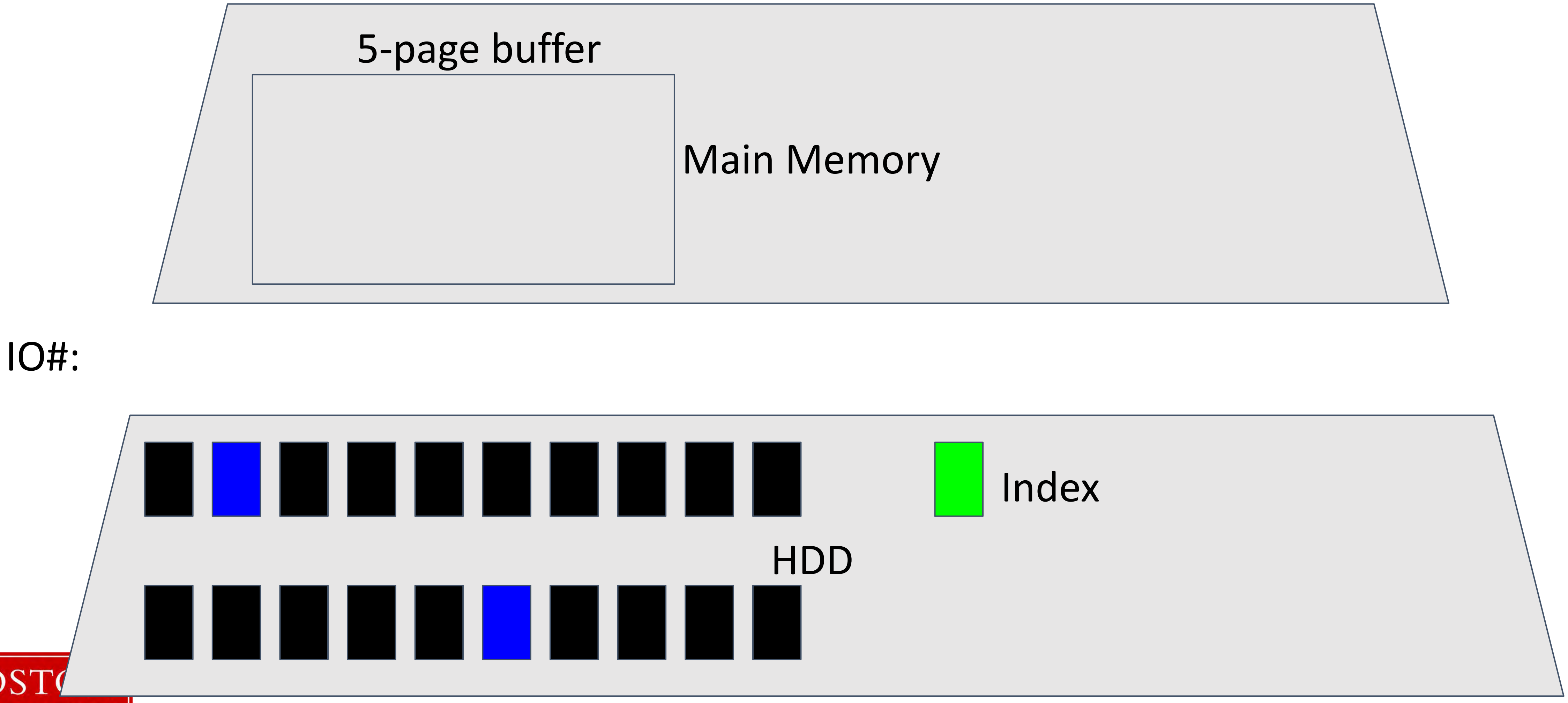
IO cost: Scanning a relation to select 10%



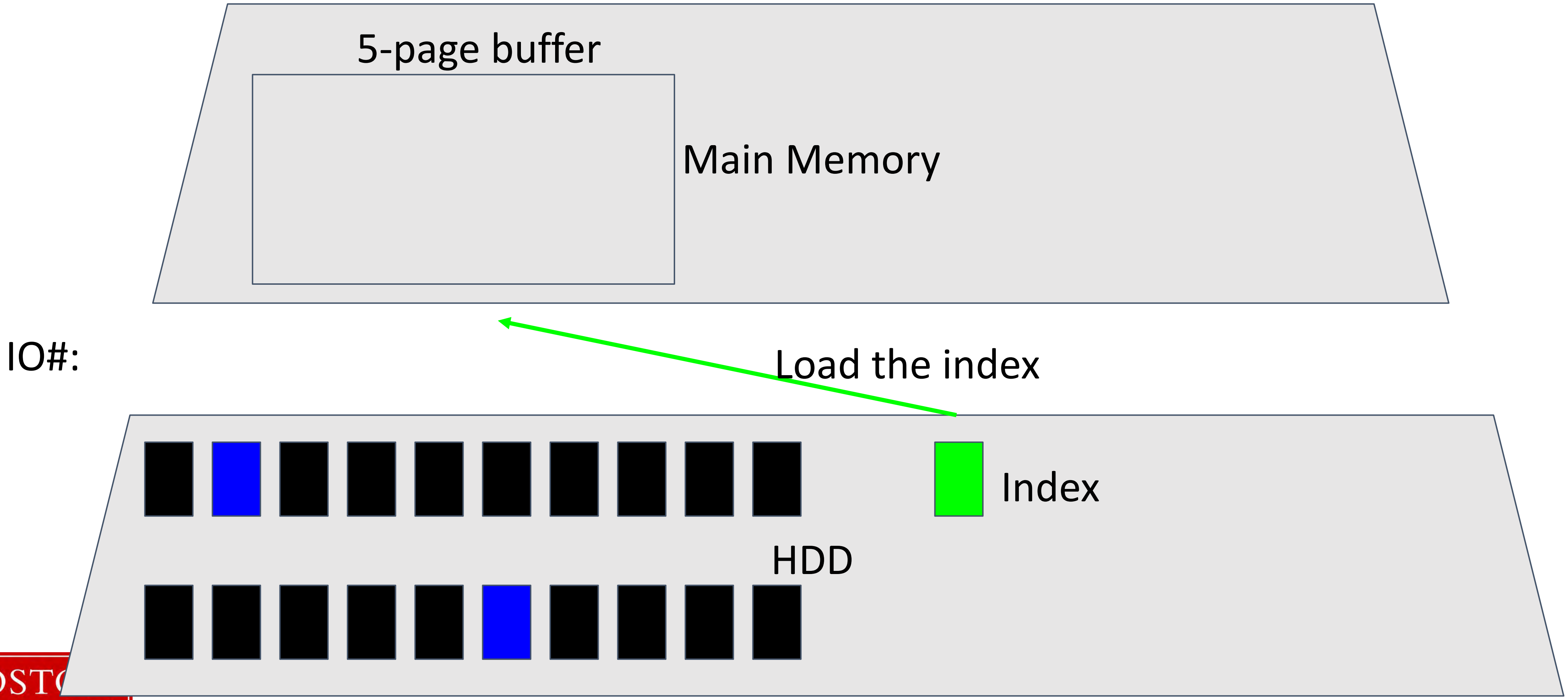
IO#: 20

What if we had an oracle (index)?

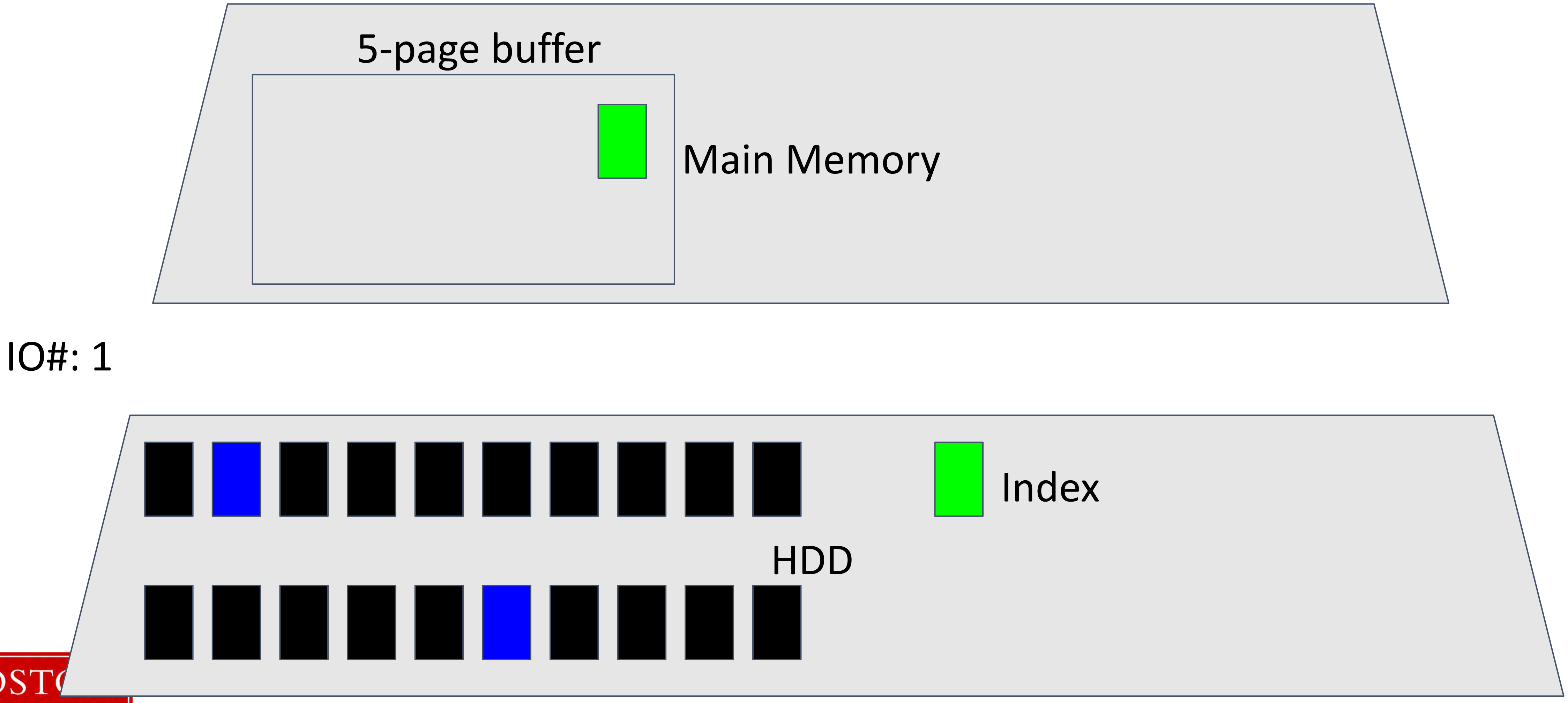
IO cost: Scanning a relation to select 10%



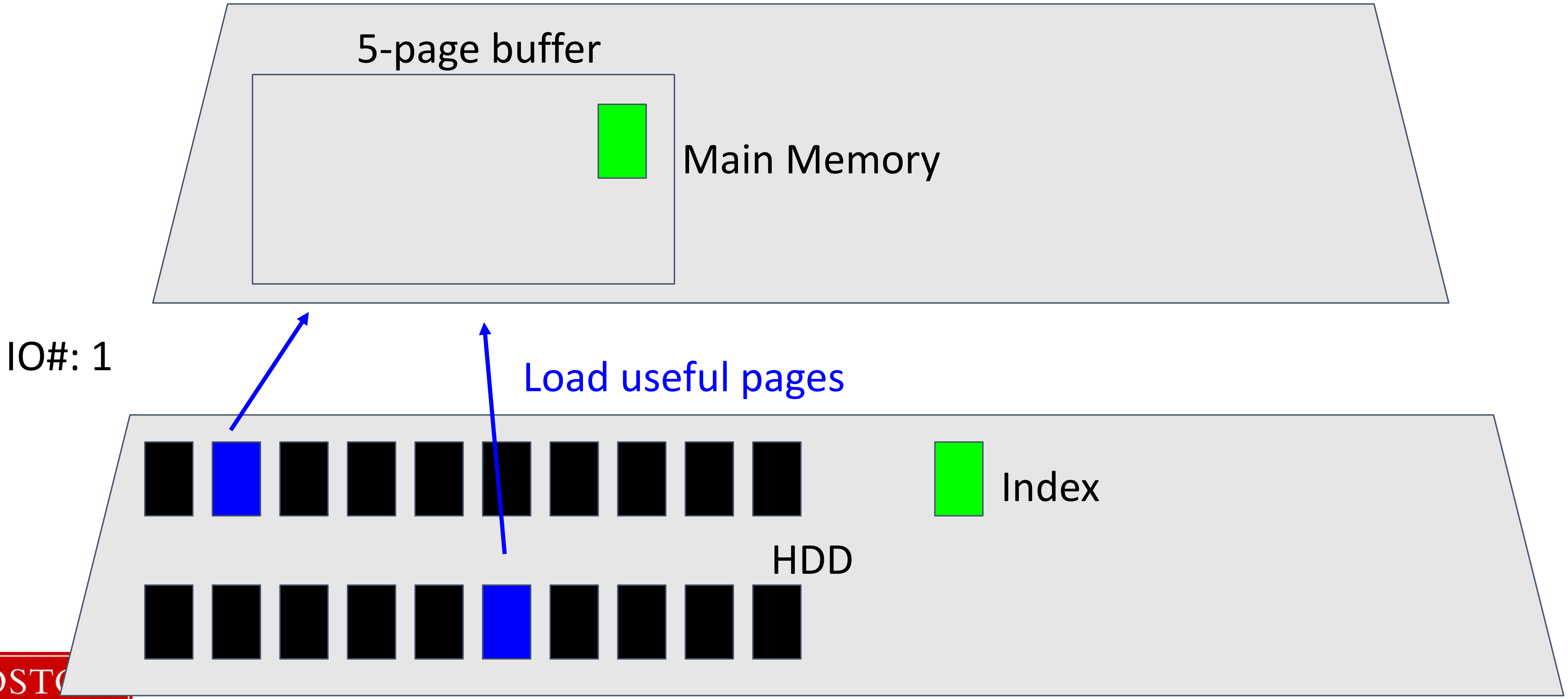
IO cost: Use an index to select 10%



IO cost: Use an index to select 10%

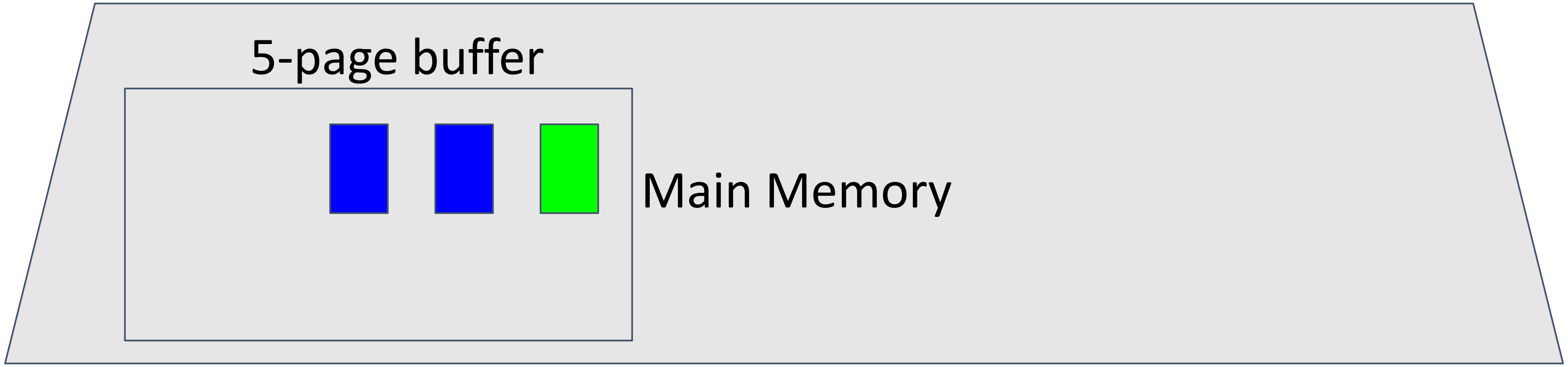


IO cost: Use an index to select 10%

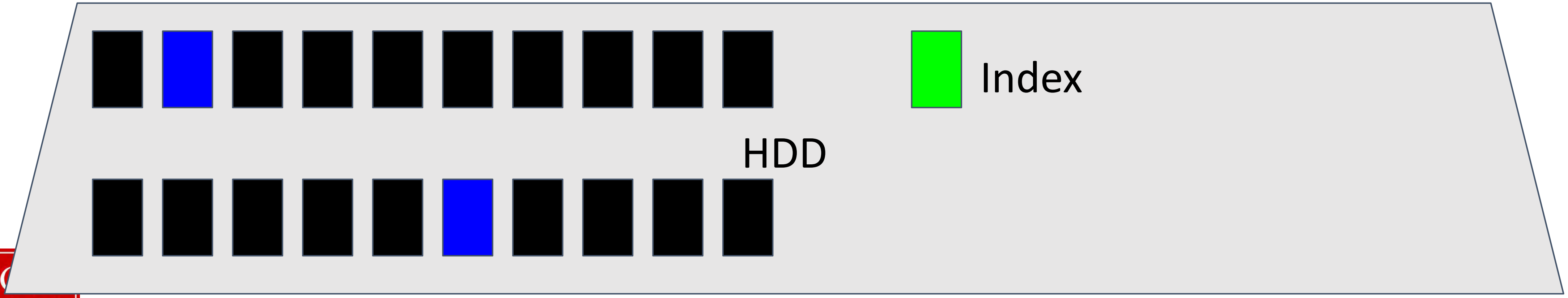


BOSTON UNIVERSITY

IO cost: Use an index to select 10%



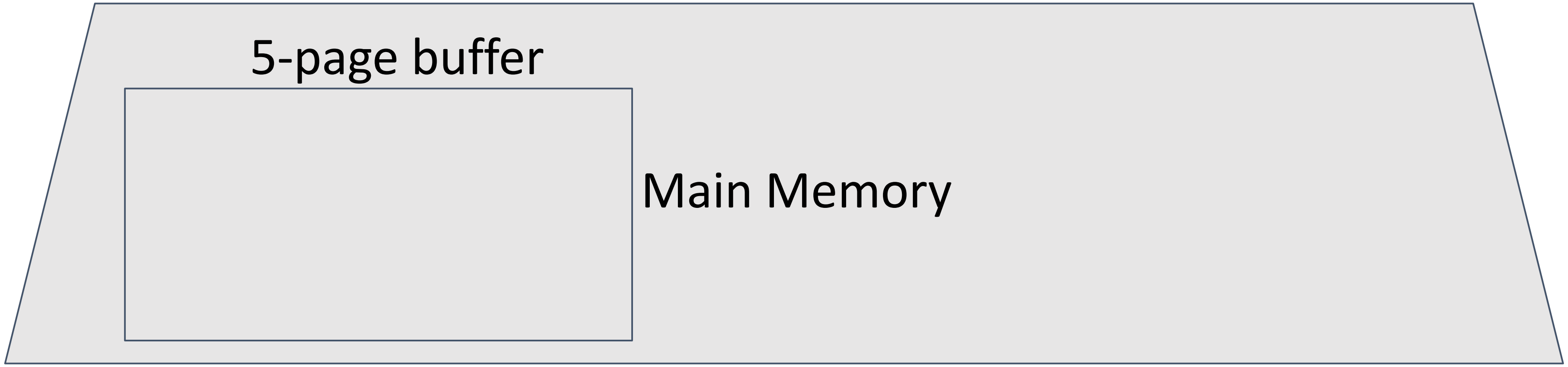
IO#: 3



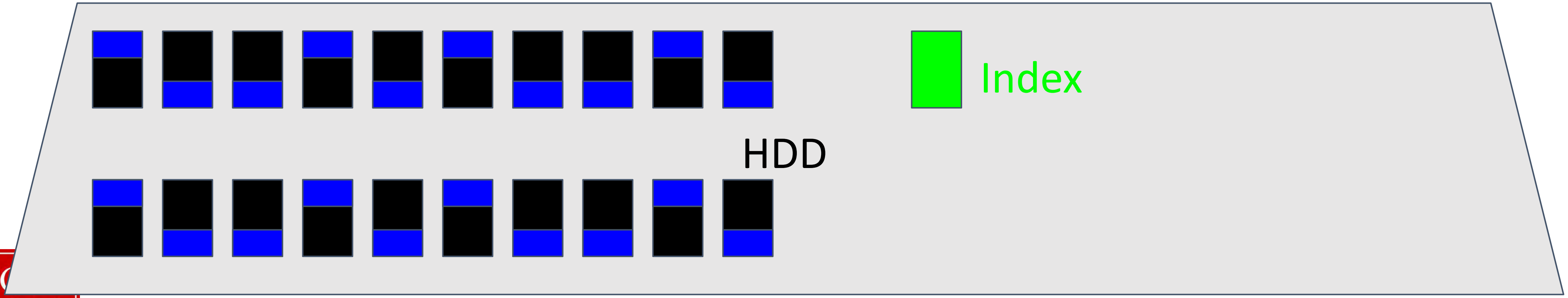
BOSTON UNIVERSITY

What if useful data is in all pages?

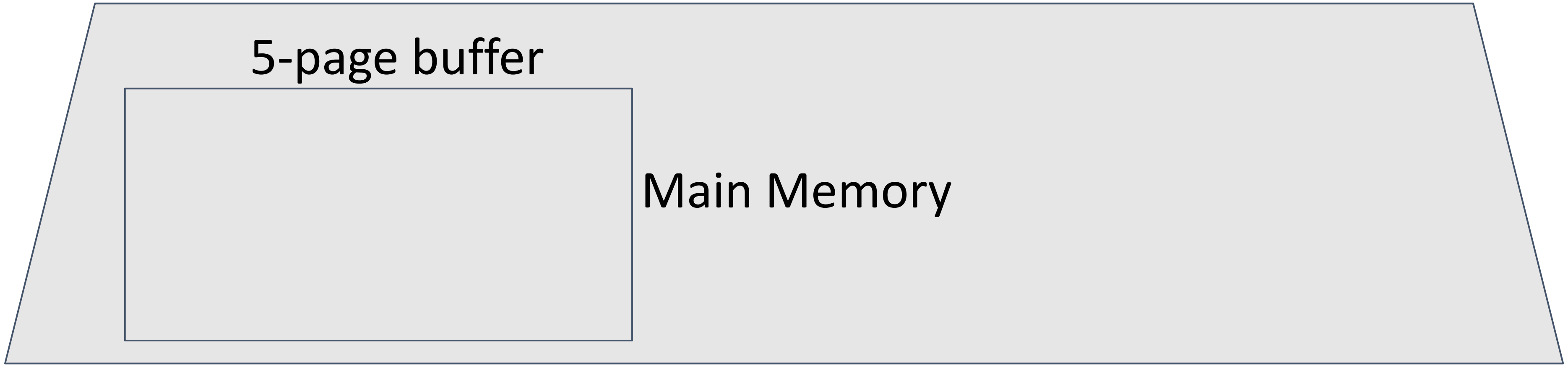
Scan or Index ?



IO#:

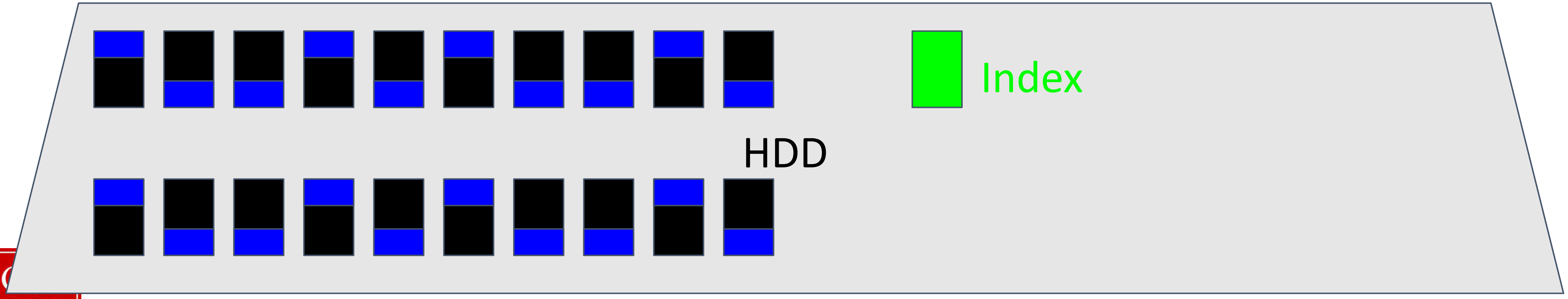


Scan or Index ?

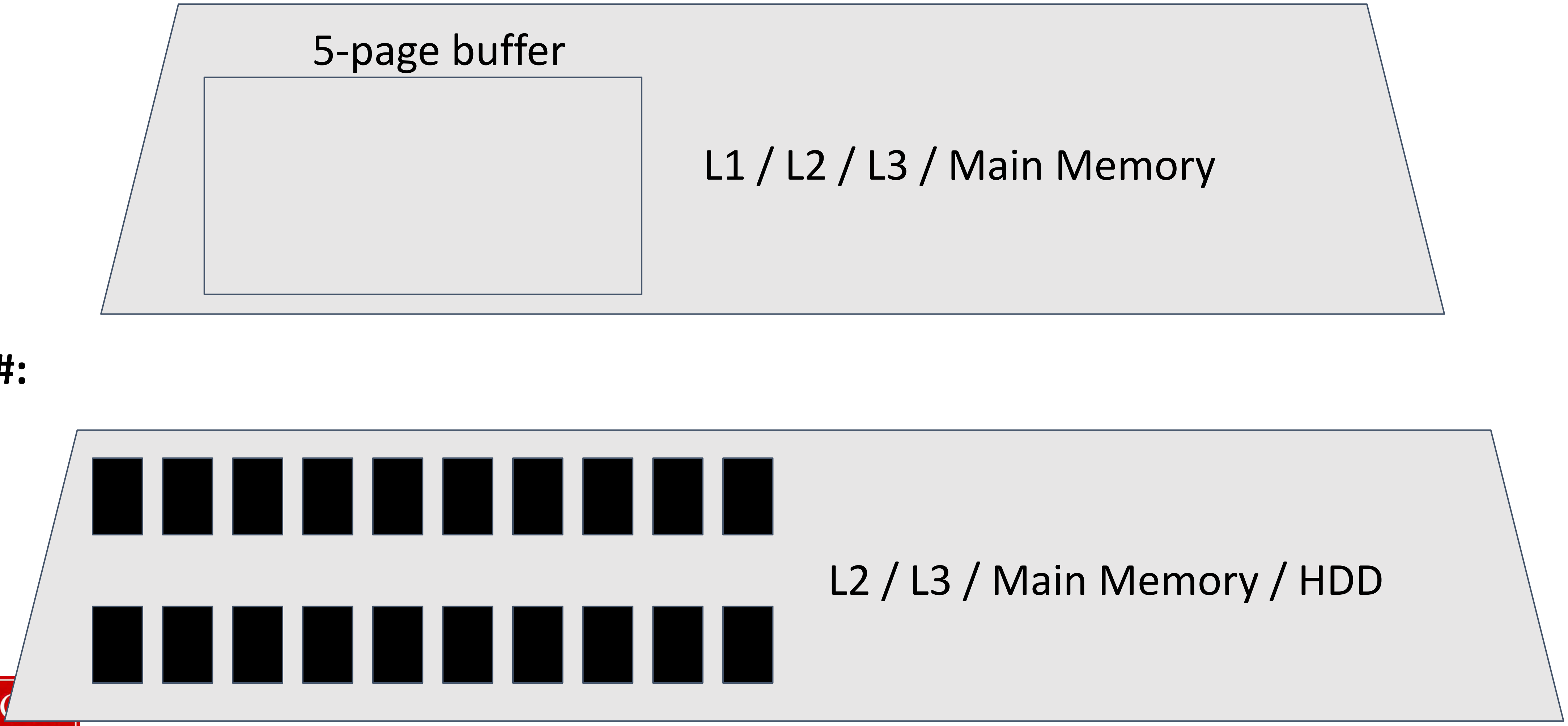


IO#: 20 with scan

IO#: 21 with index

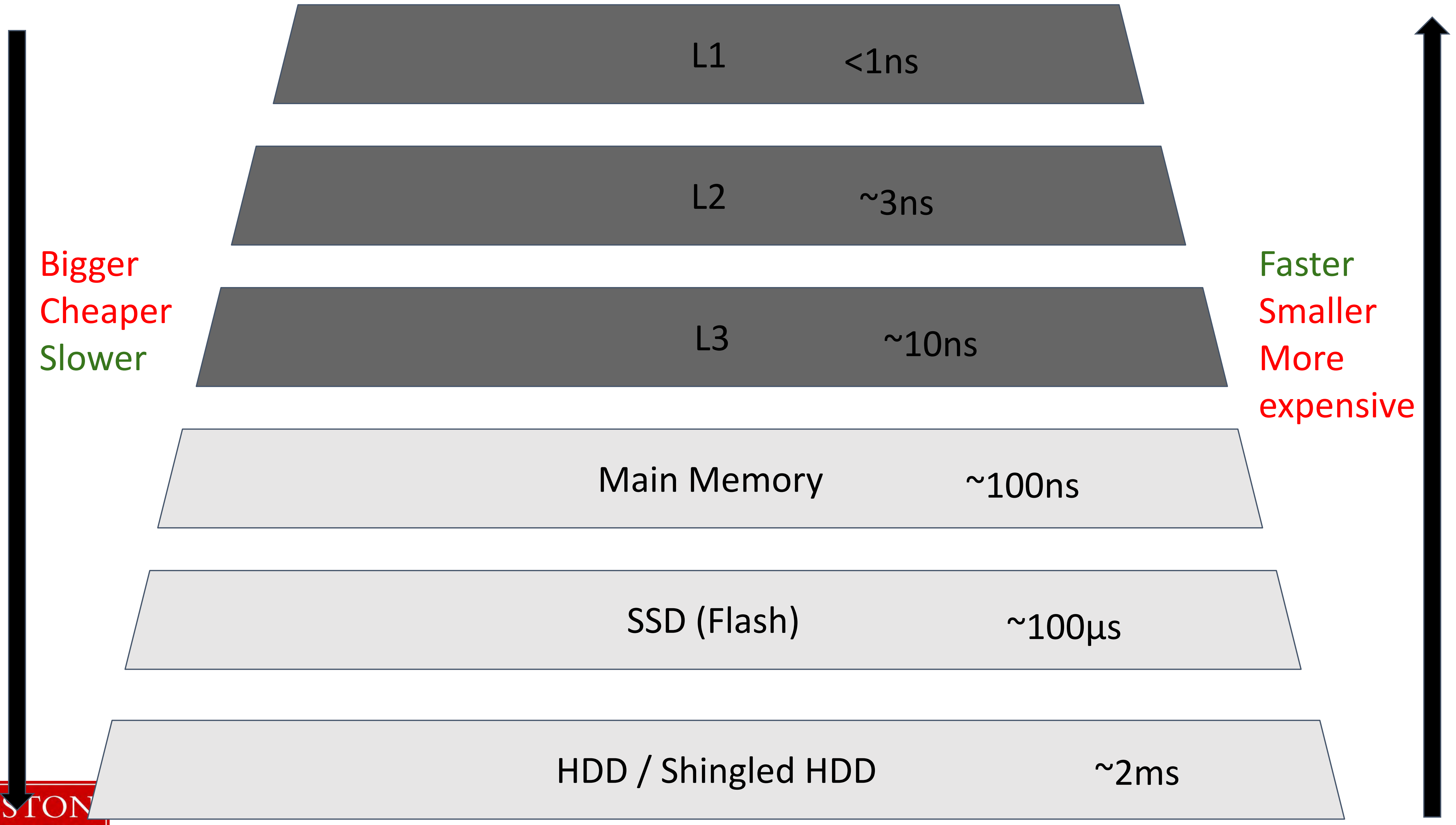


Same analysis for any two memory levels!!



IO#:

Cache Hierarchy

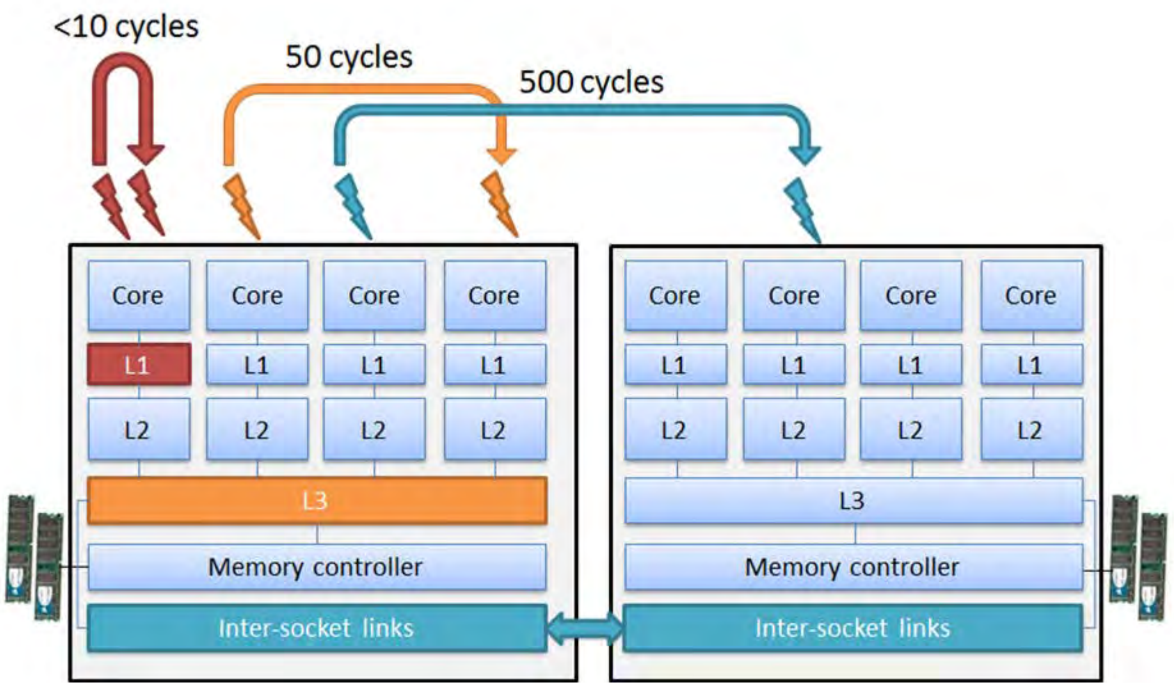


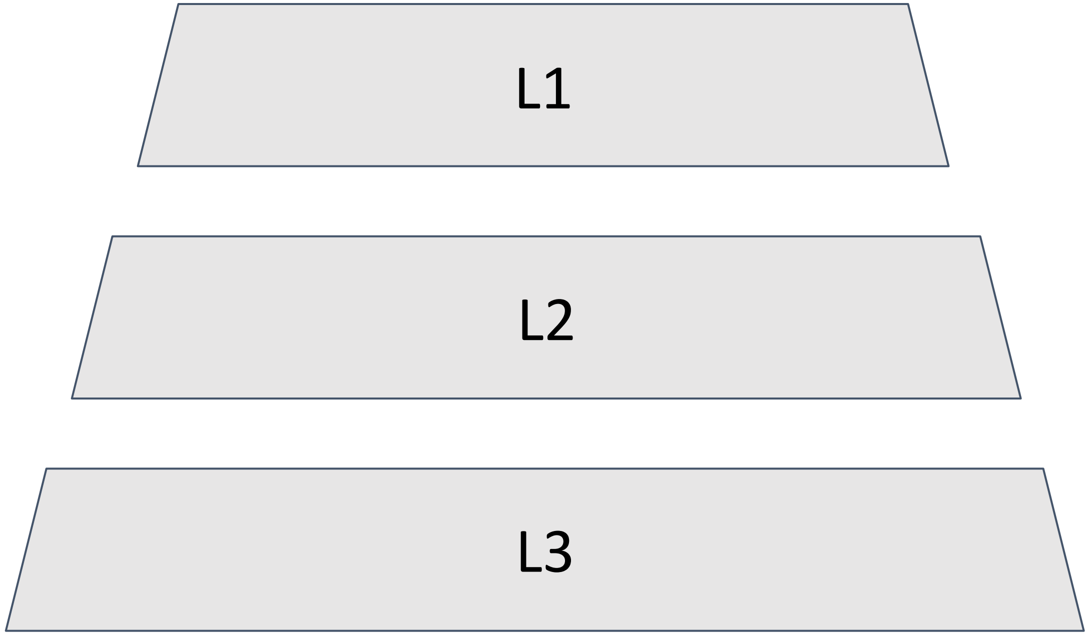
L1
<1ns
L2
~3ns
L3
~10ns
Main Memory
~100ns
SSD (Flash)
~100µs
HDD / Shingled HDD
~2ms
Bigger
Cheaper
Slower
Faster
Smaller
More expensive

Cache Hierarchy

What is a core?

What is a socket?





BOSTON UNIVERSITY

Cache Hierarchy

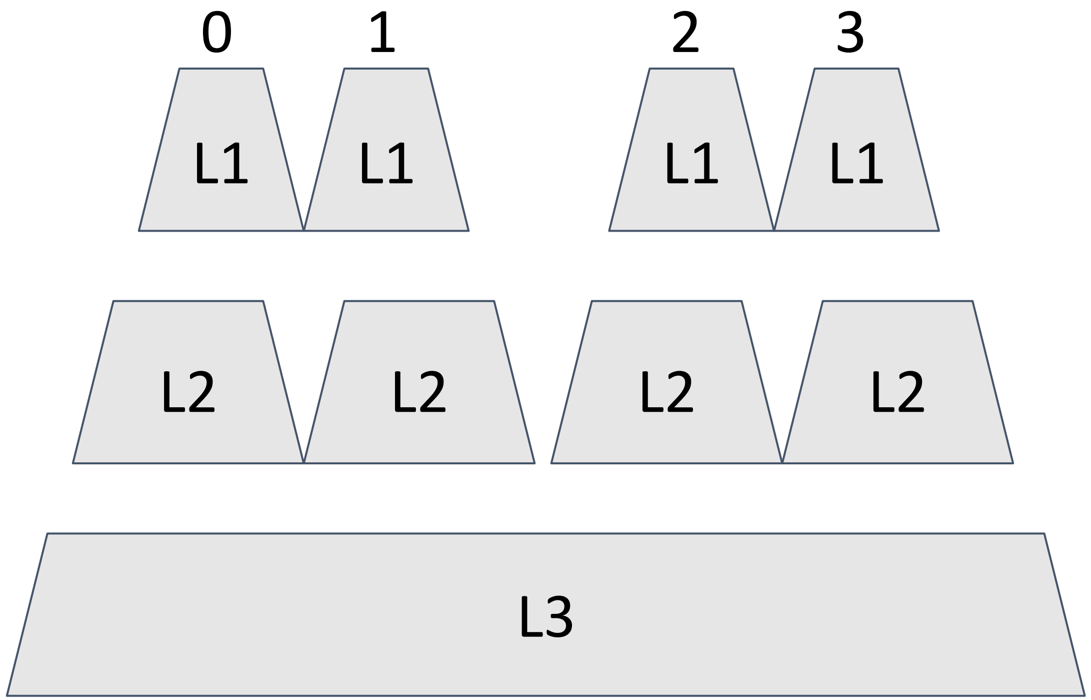
Shared Cache: L3 (or LLC: Last Level Cache)

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**



Non Uniform Memory Access (NUMA)

Core 0 reads faster when data are in its L1

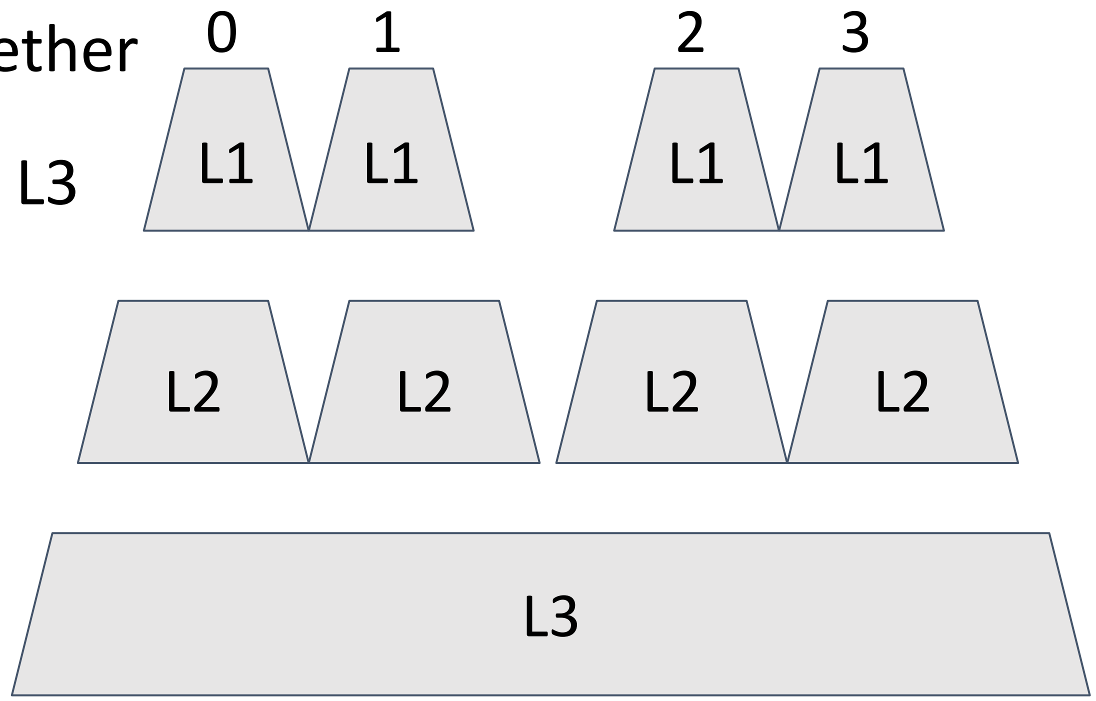
If it does not fit, it will go to L2, and then in L3

Can we control where data is placed?

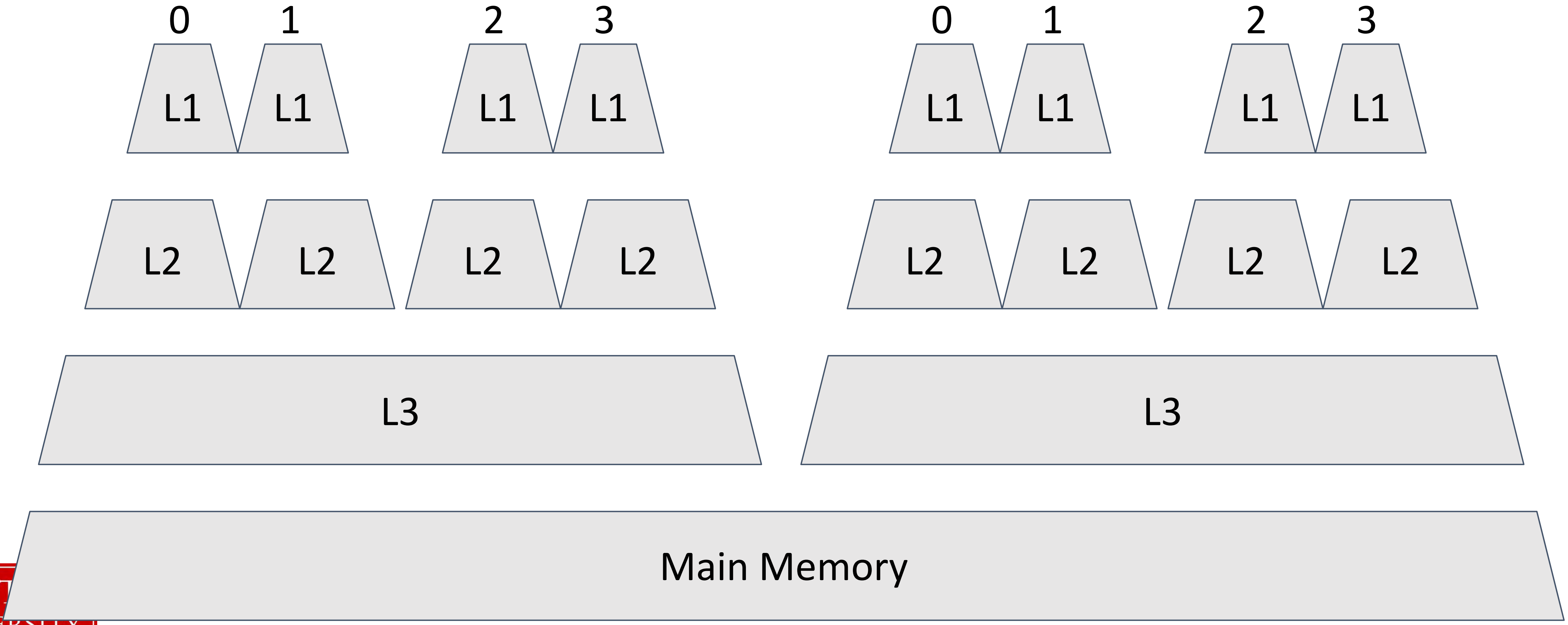
We would like to avoid going to L2 and L3 altogether

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

And remember: this is only one socket!
We have multiple of those!

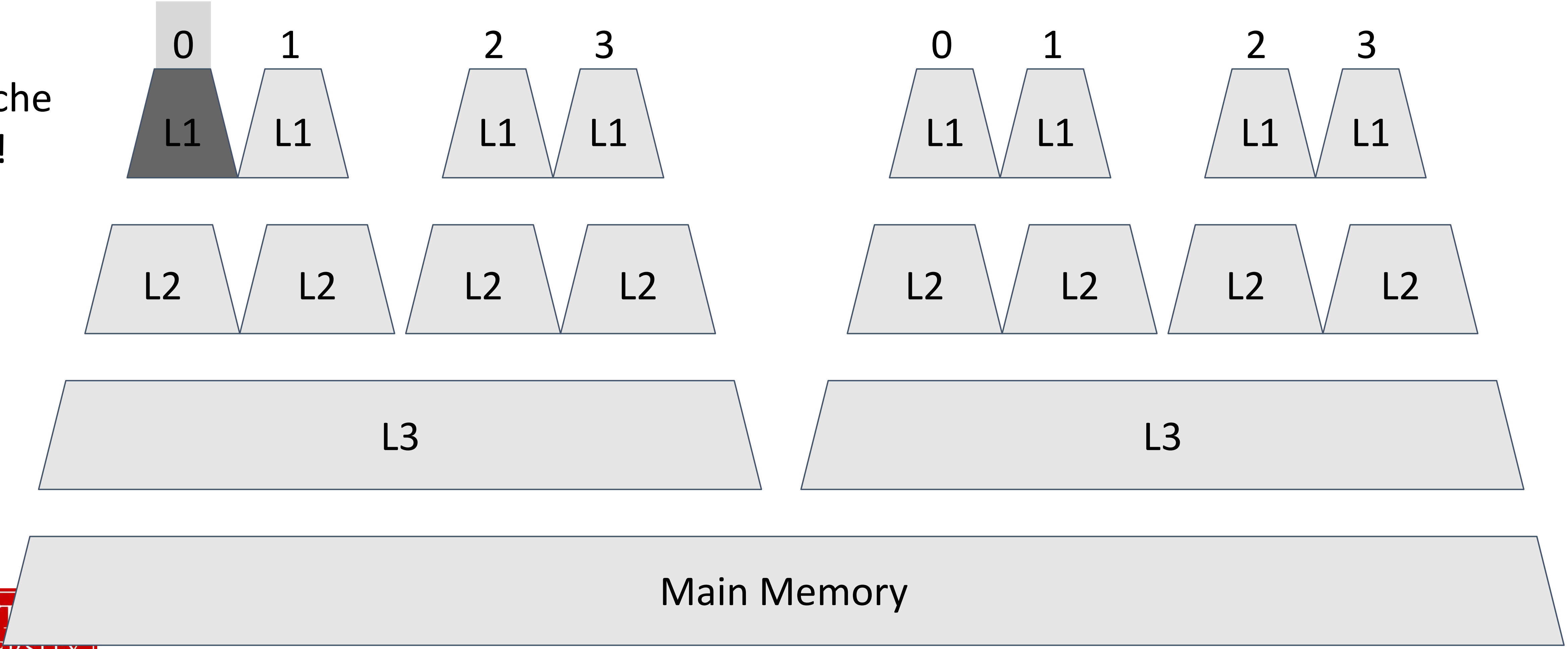


Non Uniform Memory Access (NUMA)

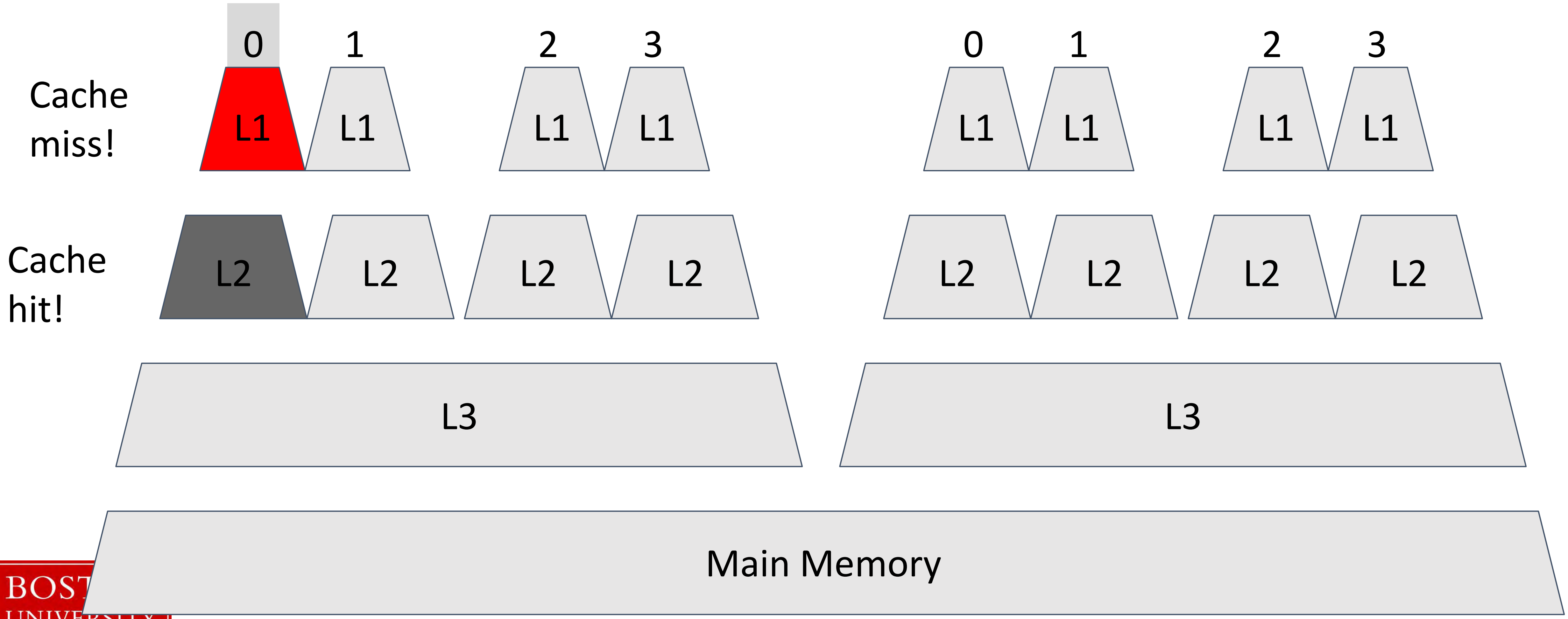


Non Uniform Memory Access (NUMA)

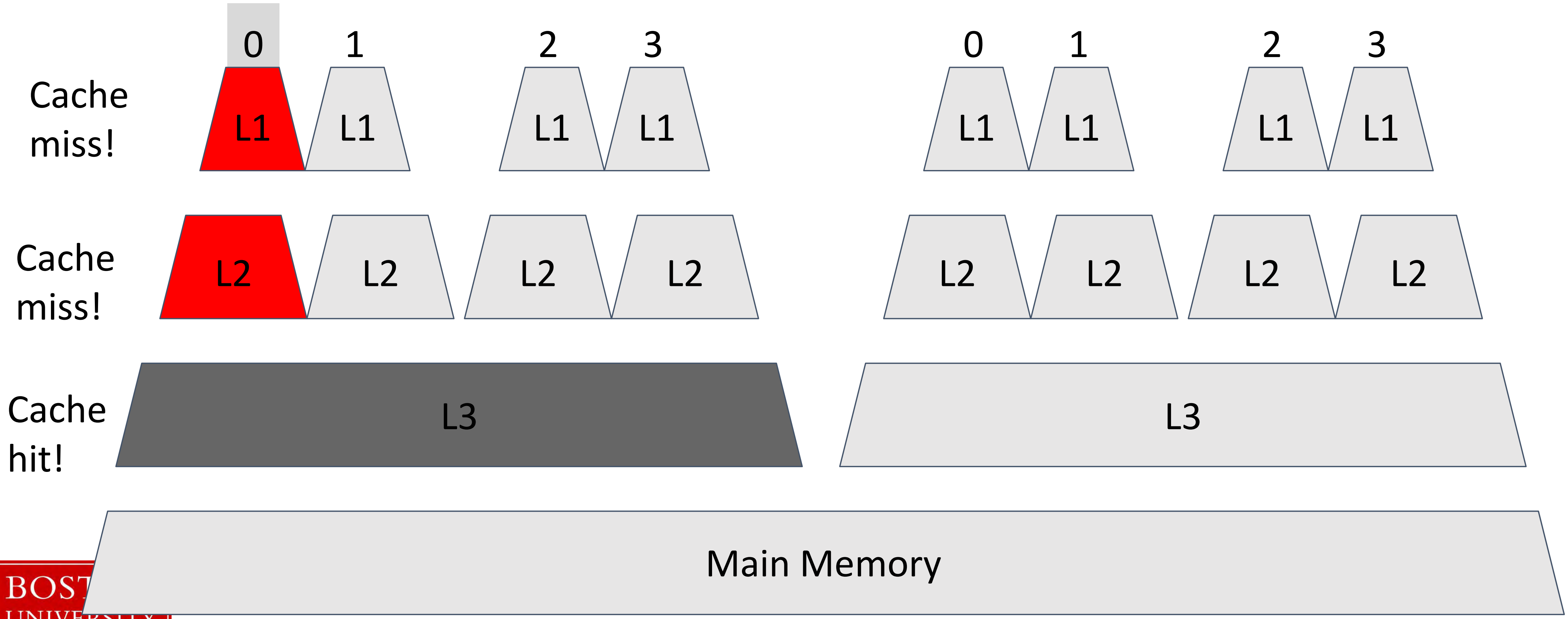
Cache hit!



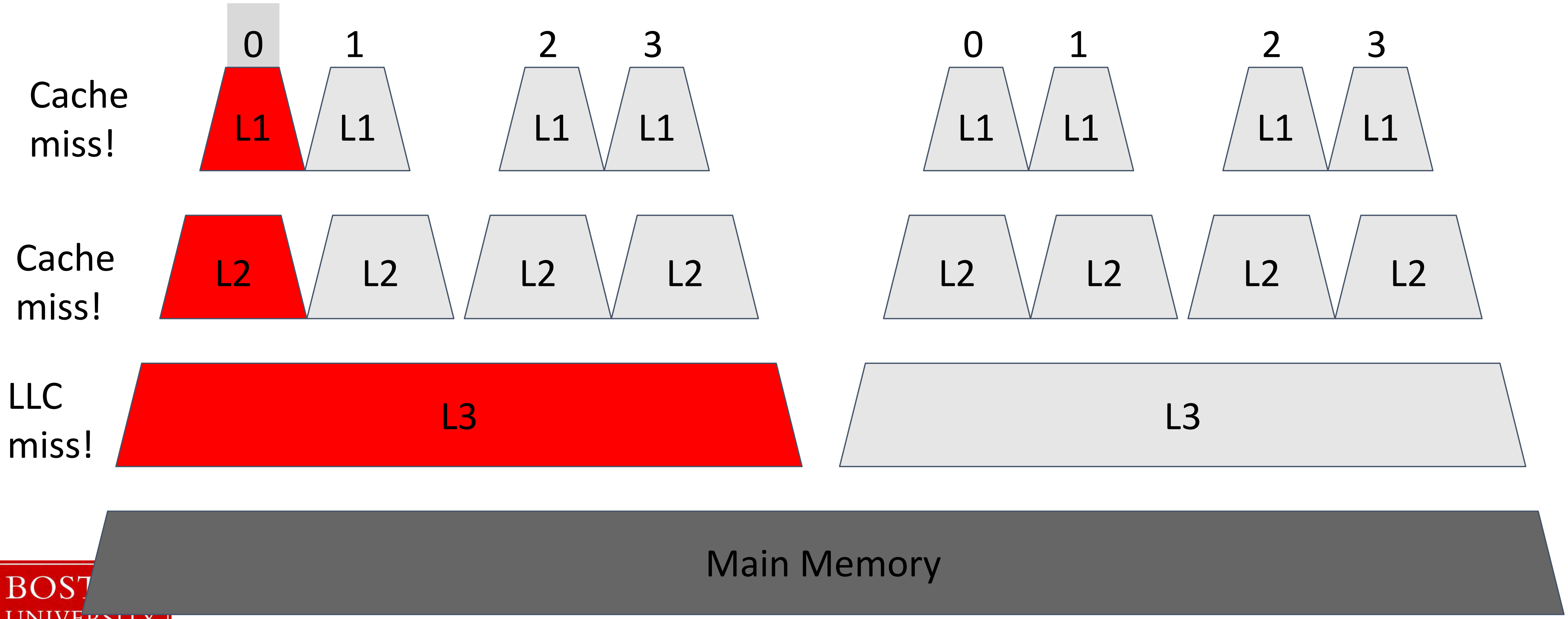
Non Uniform Memory Access (NUMA)



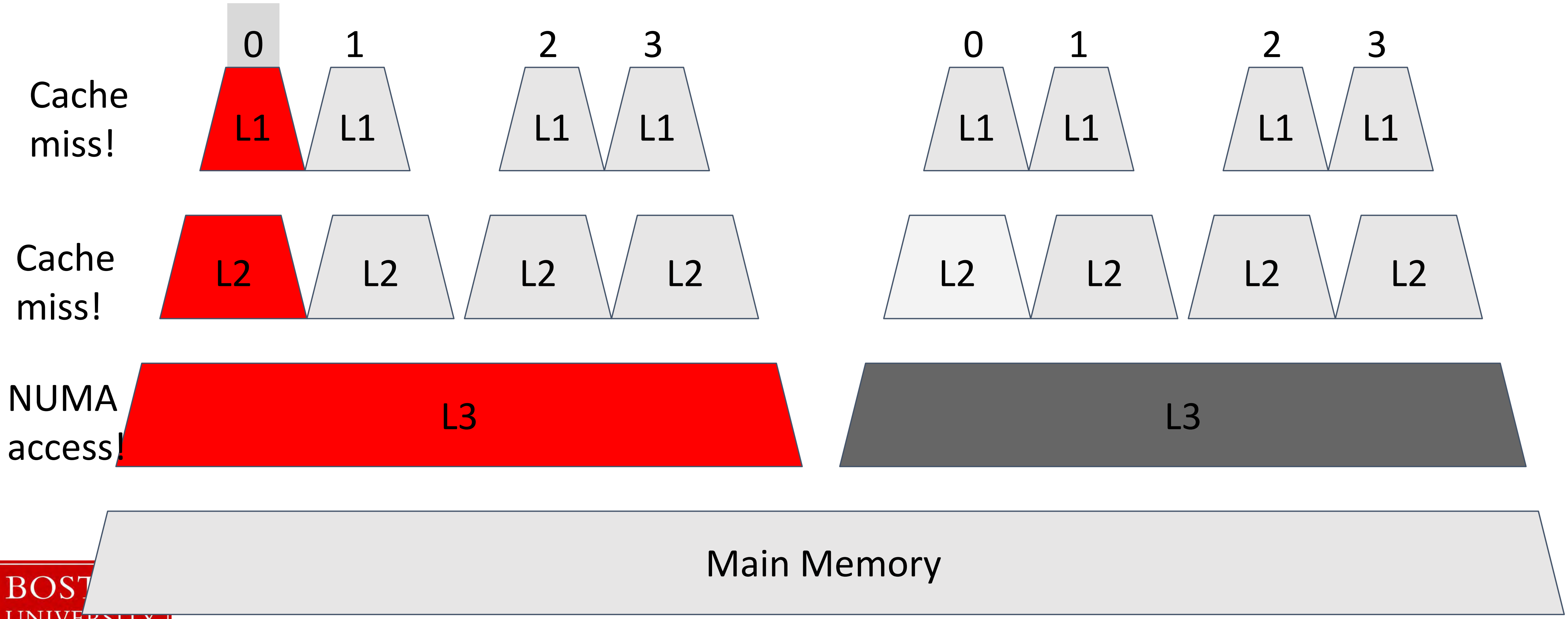
Non Uniform Memory Access (NUMA)



Non Uniform Memory Access (NUMA)



Non Uniform Memory Access (NUMA)



Why knowing the cache hierarchy matters

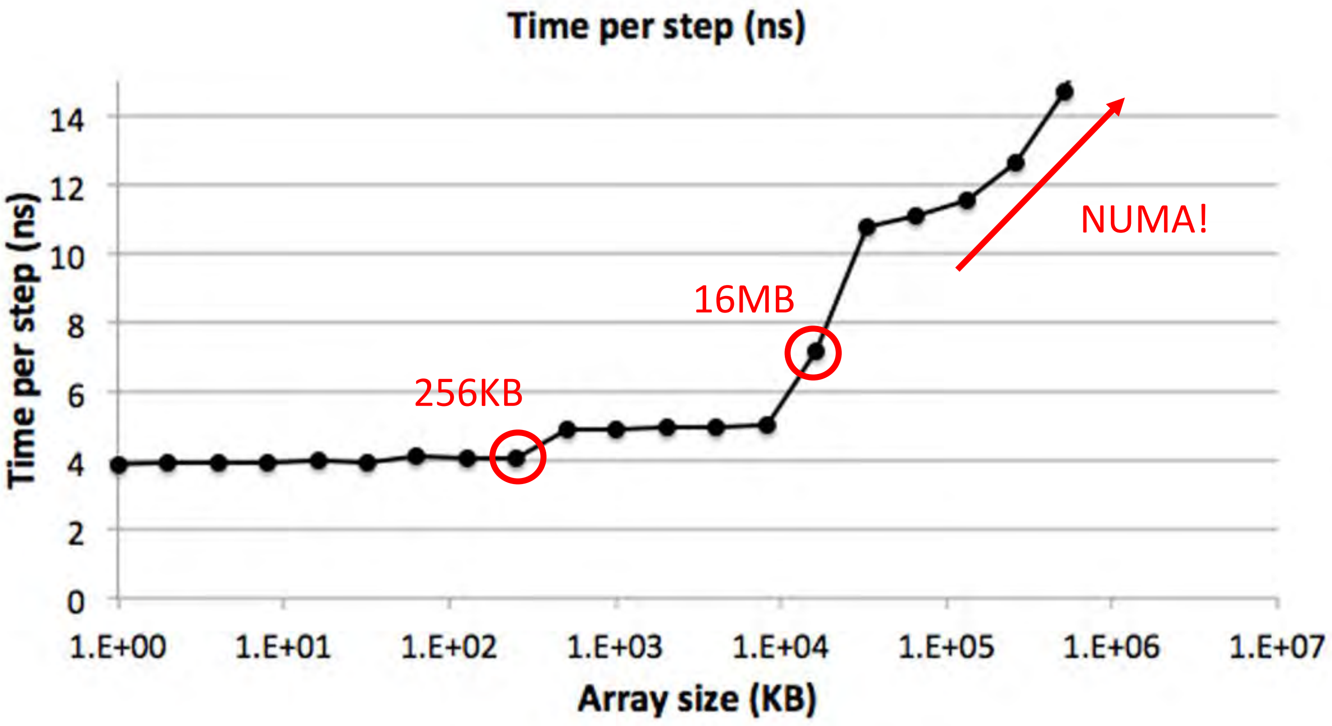
```
int arraySize;
for (arraySize = 1024/sizeof(int) ; arraySize <= 2*1024*1024*1024/sizeof(int) ; arraySize*=2)
// Create an array of size 1KB to 4GB and run a large arbitrary number of operations
{
    int steps = 64 * 1024 * 1024; // Arbitrary number of steps
    int* array = (int*) malloc(sizeof(int)*arraySize);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



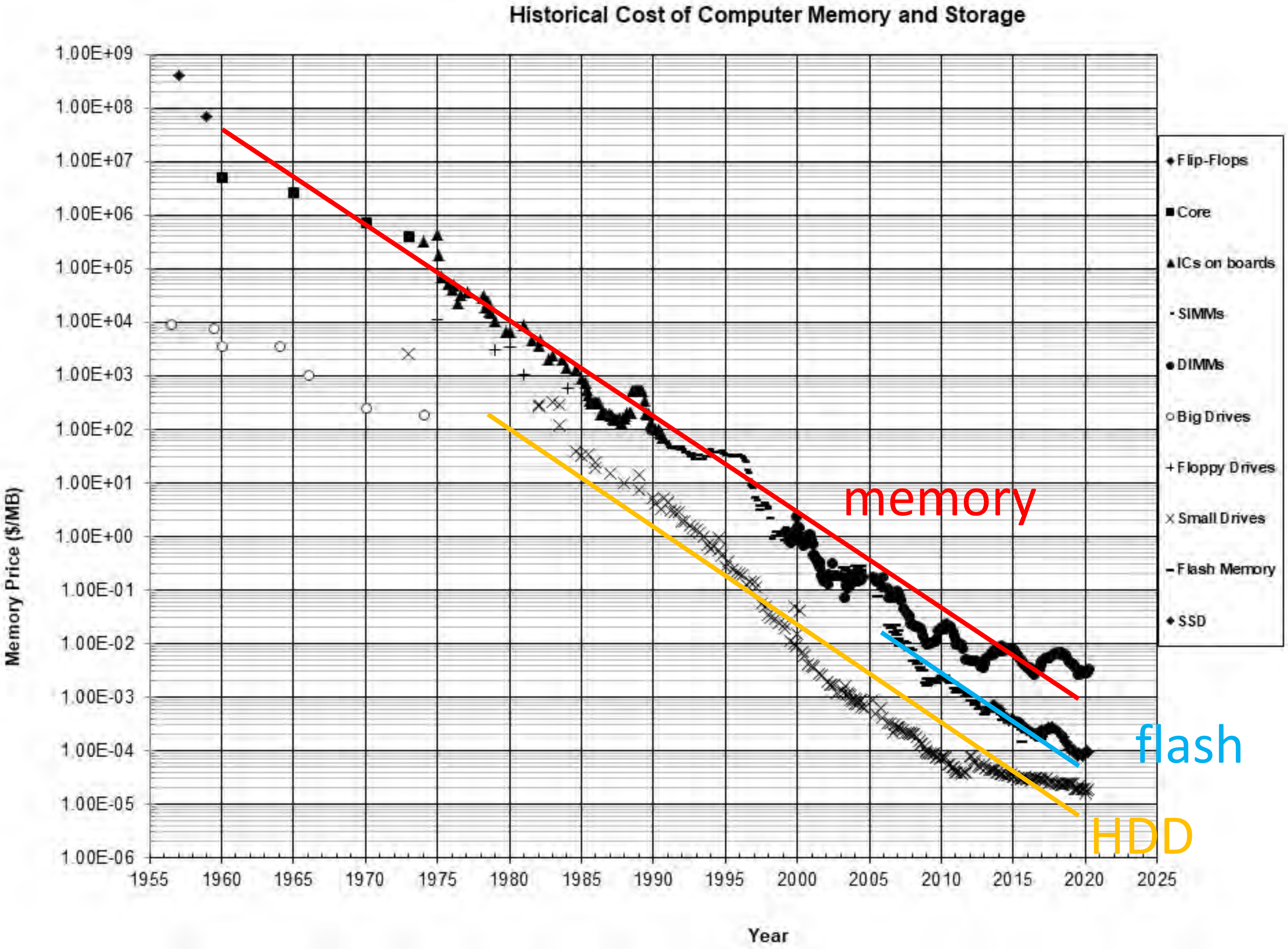
Storage Hierarchy

Why not just stay in memory?

BOSTON UNIVERSITY

Cost!

what else?



BOSTON UNIVERSITY

Storage Hierarchy

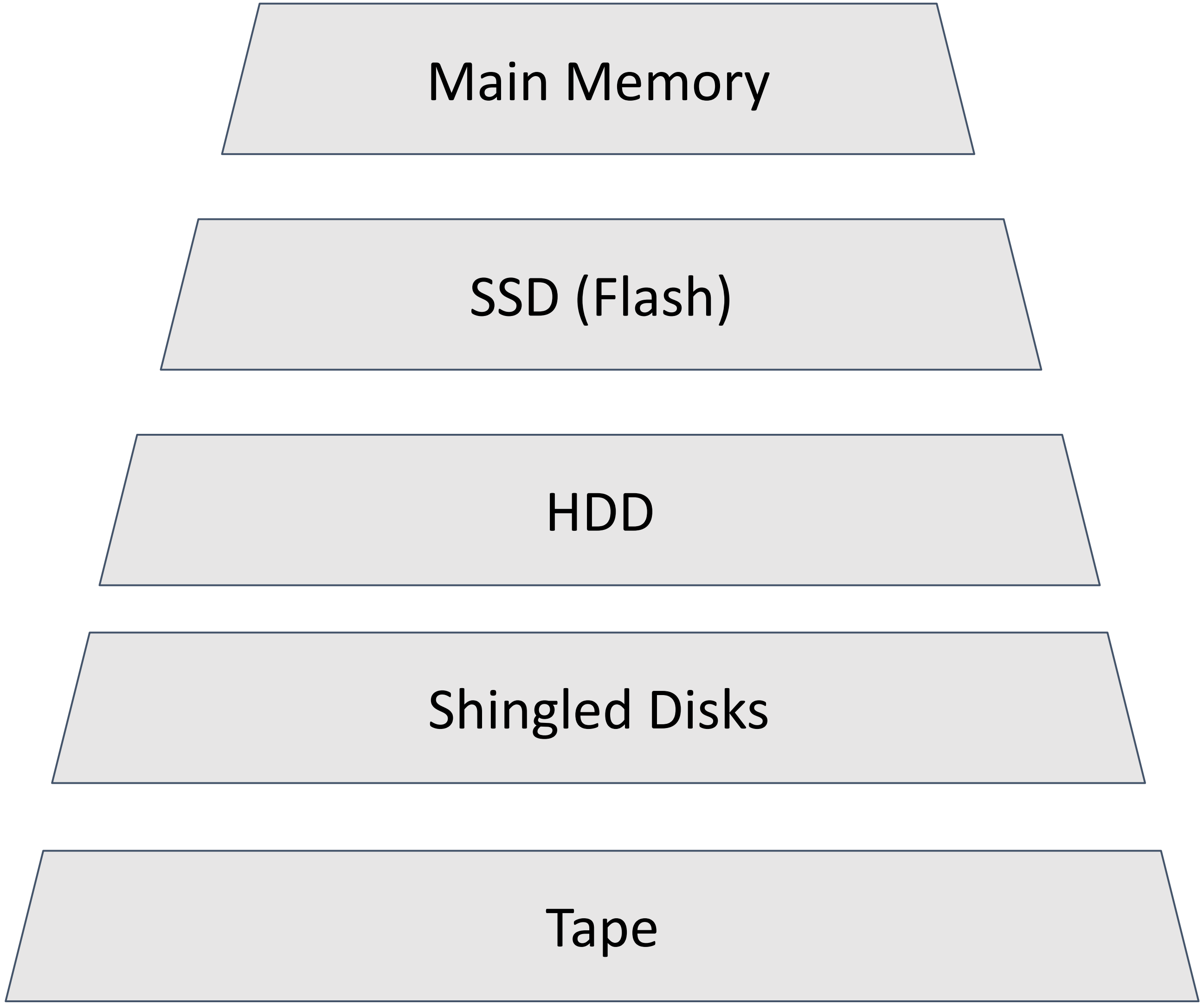
Why not stay in memory?

Rephrase: what is missing from memory hierarchy?

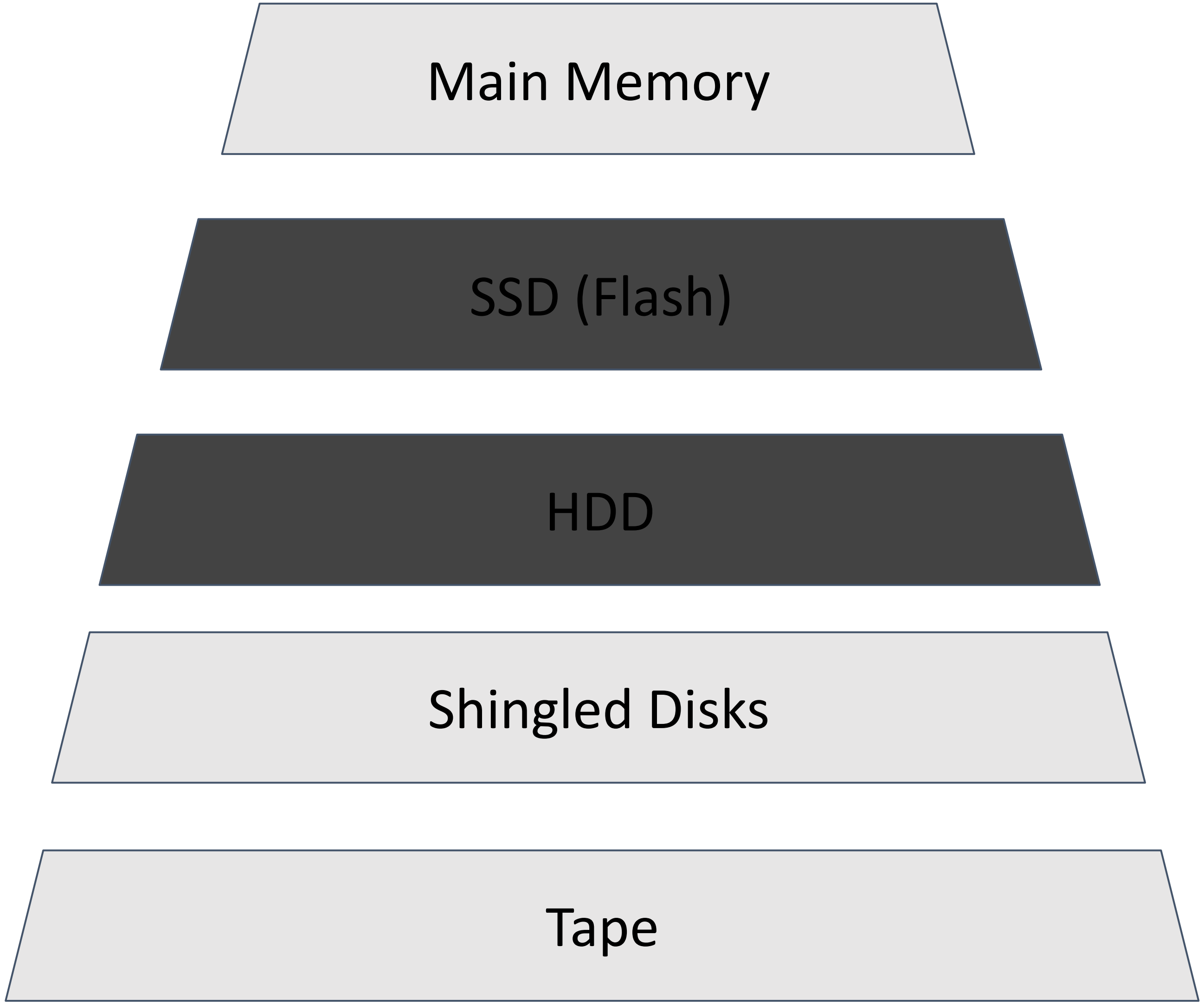
Durability (data survives between restarts)

Capacity (enough capacity for data-intensive applications)

Storage Hierarchy



Storage Hierarchy



Hard Disk Drives

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

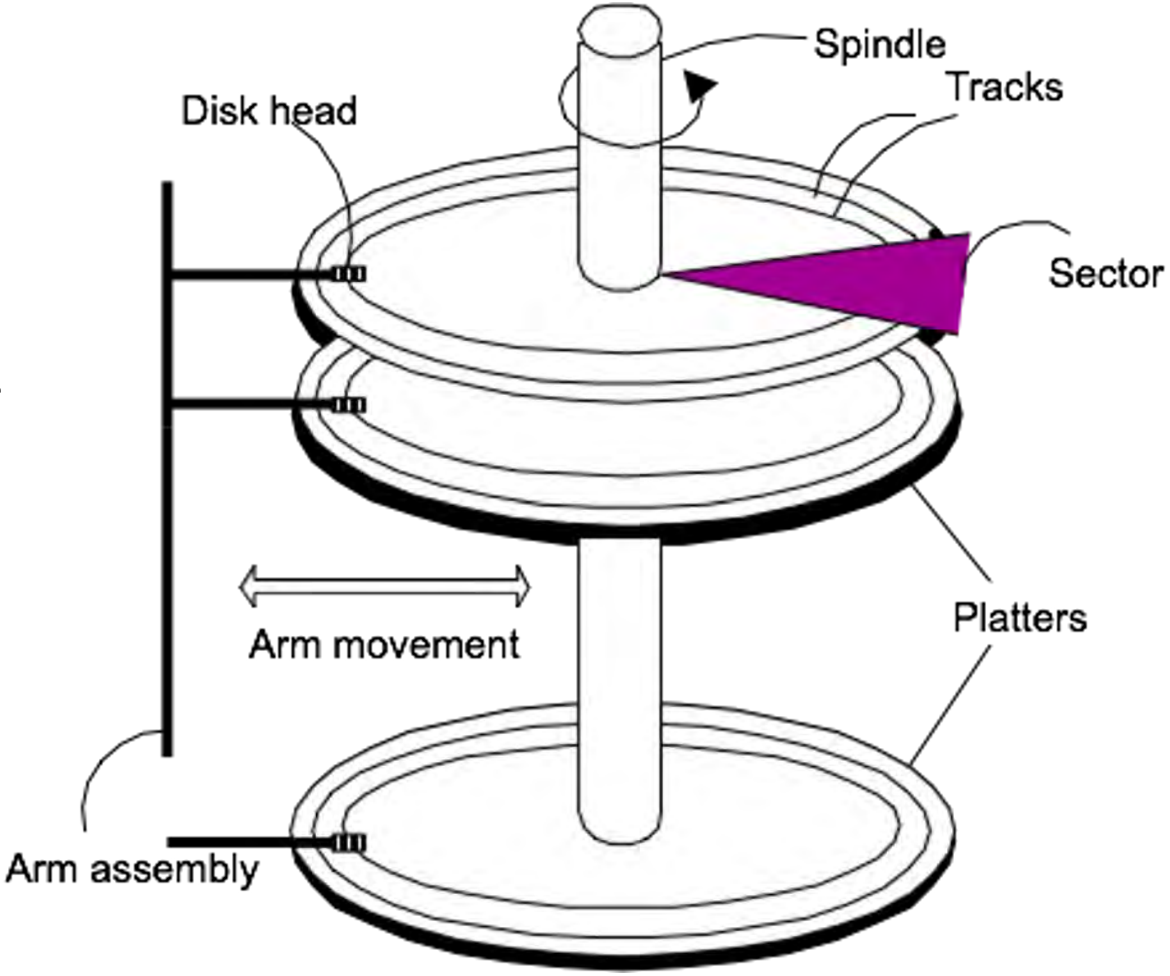
Data organized on pages/blocks (across tracks)

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

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

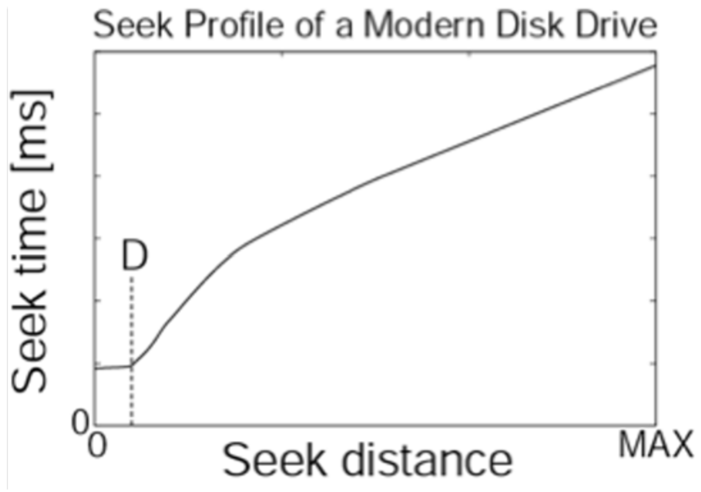
Sequential >> random access (~10x)

Goal: *avoid random access*



BOSTON UNIVERSITY

Seek time + Rotational delay + Transfer time

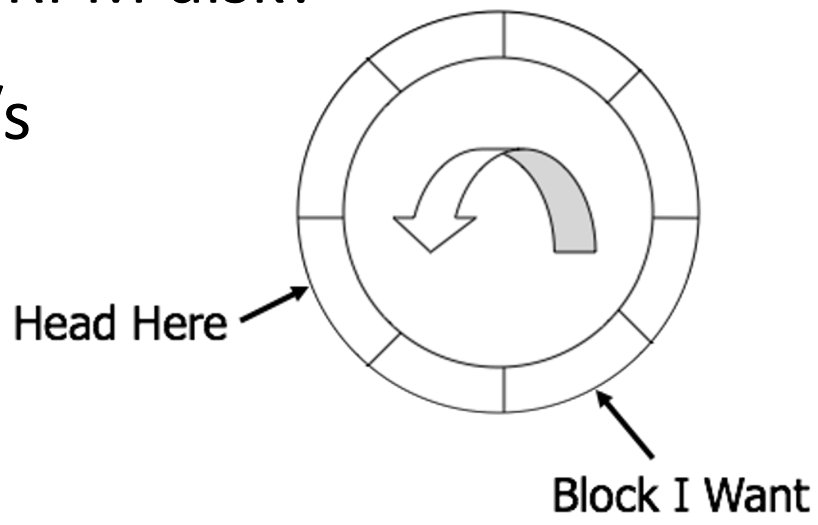


Seek time: the **head** goes to the right **track**

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?

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



BOSTON UNIVERSITY

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) + 1ms (rotational delay) + 0.04ms = 1.54ms

4KB/1.54ms → **2.5MB/s**

BOSTON UNIVERSITY

Flash

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

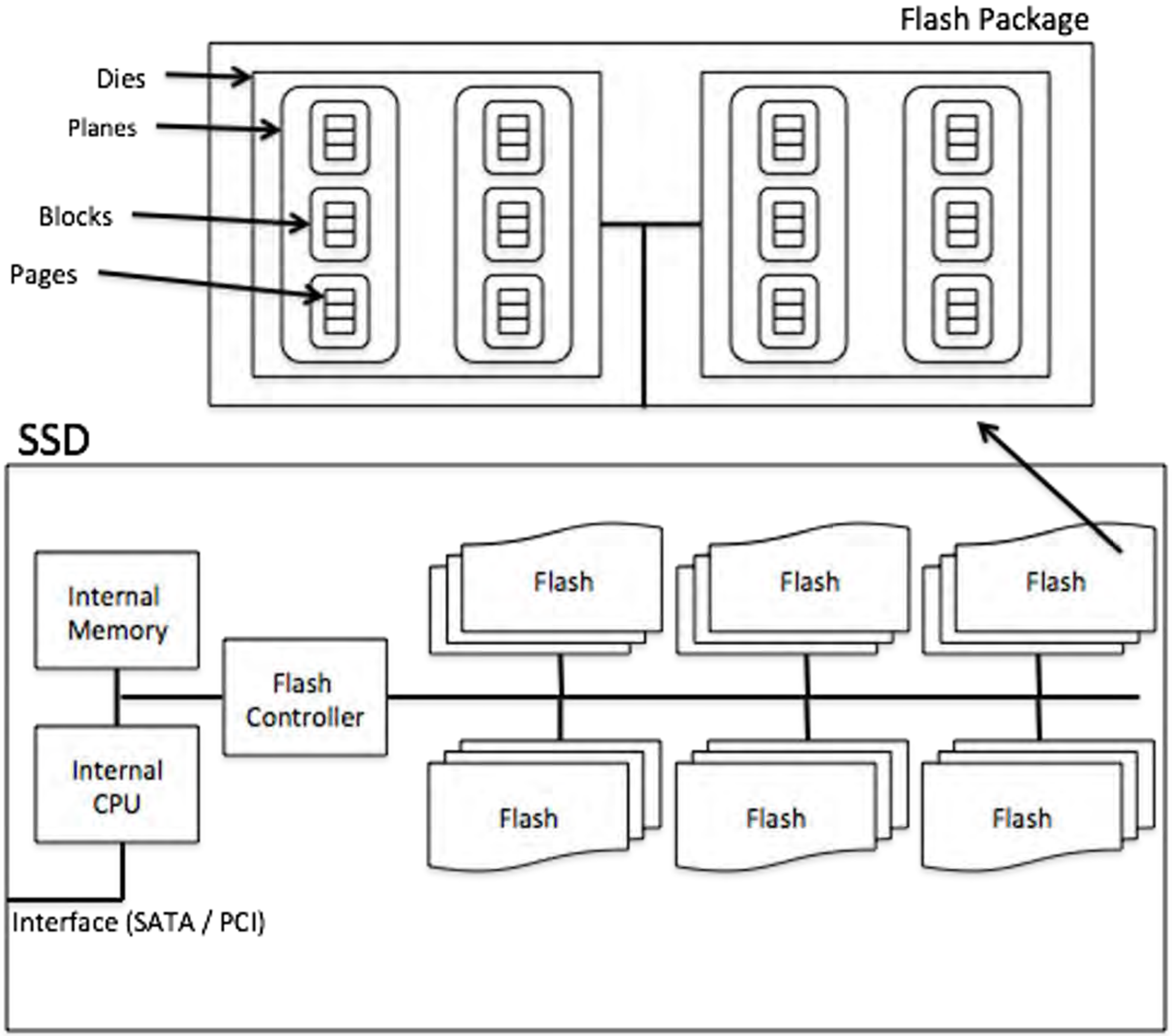
Data organized on pages (similar to disks) which are further grouped to erase blocks

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

BUT: Slow 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:

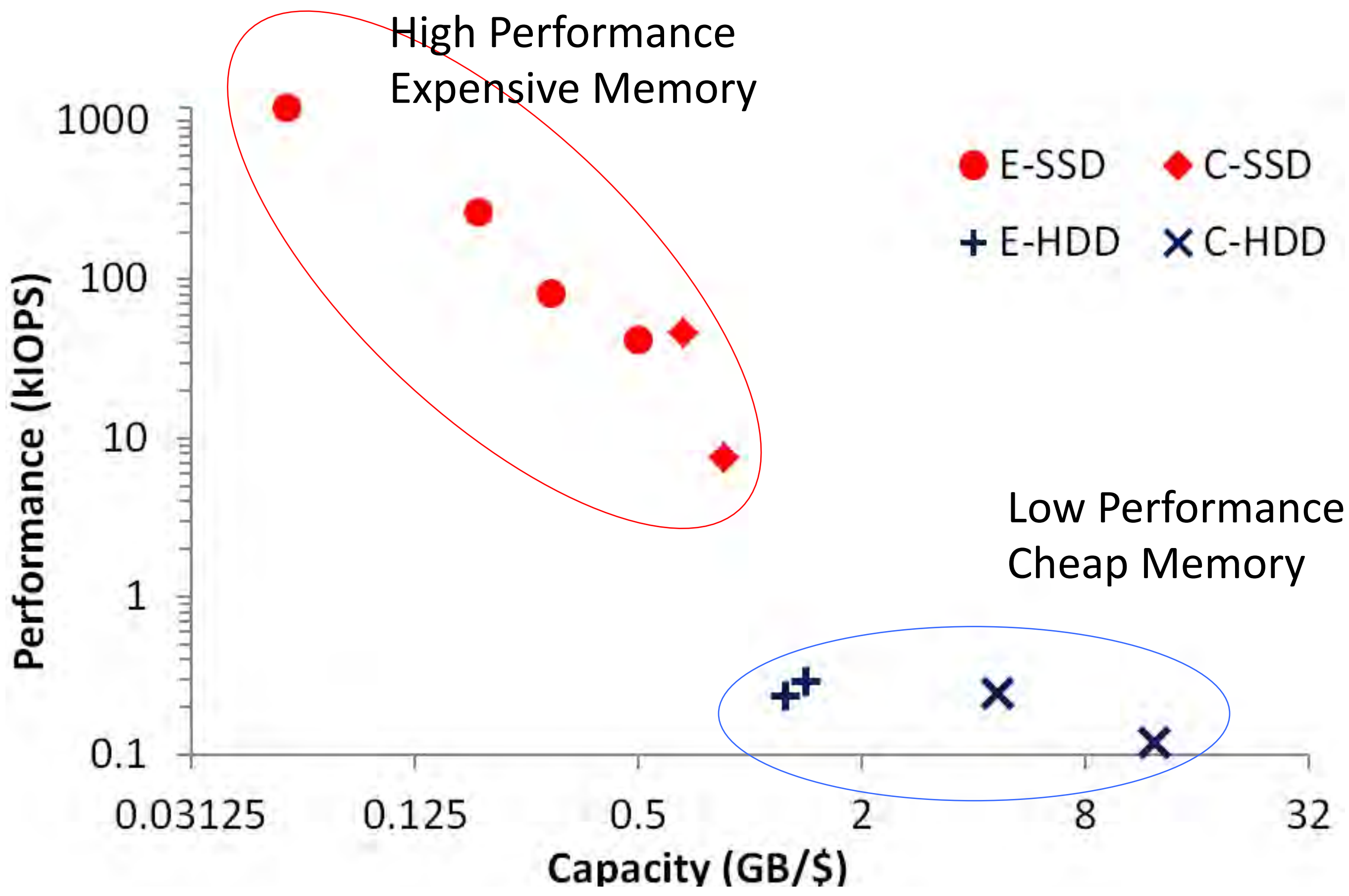
device organization (***internal parallelism***)

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

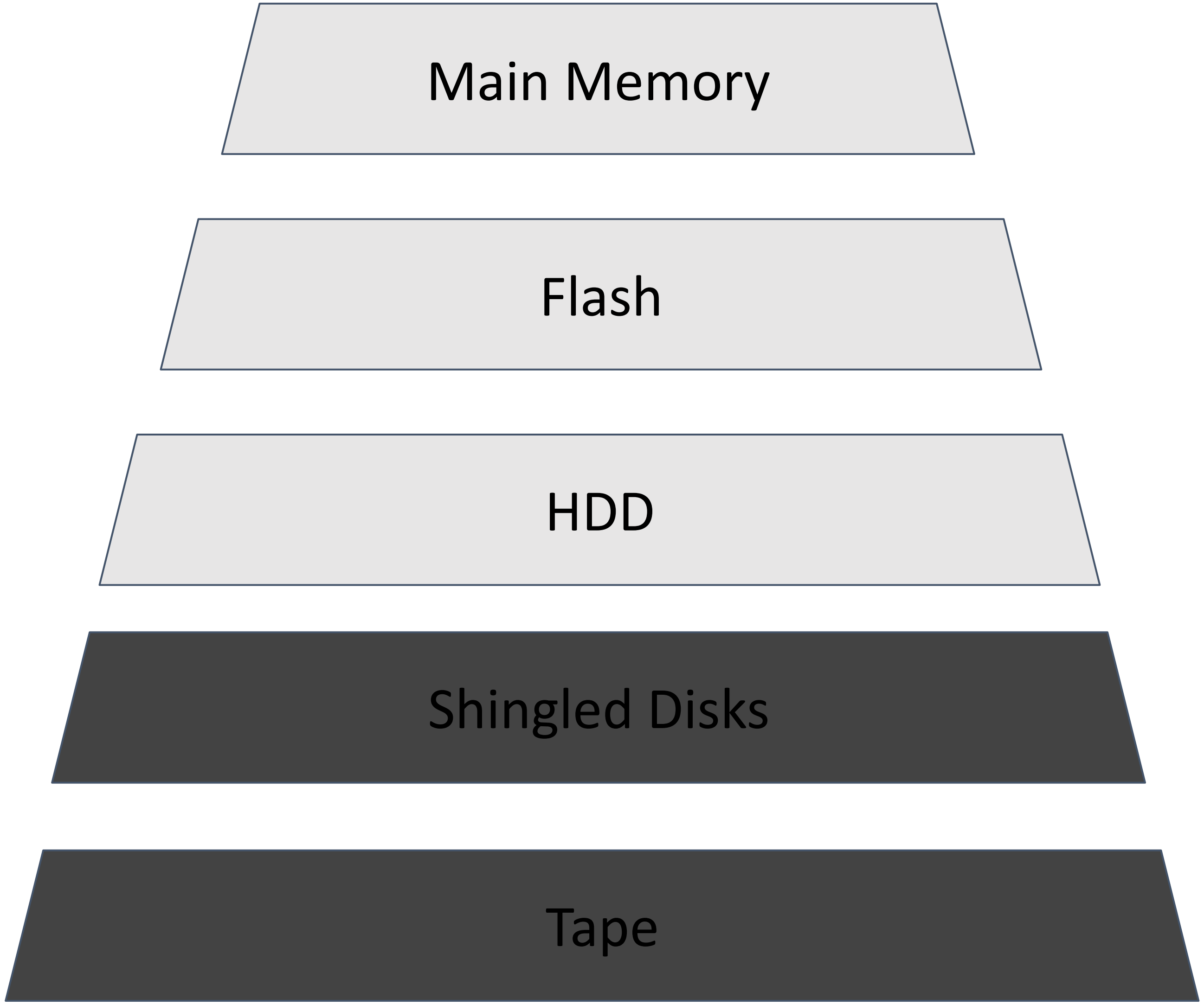
Flash

✗ Small - expensive capacity

✓ Very efficient random reads

✗ Read/Write Asymmetry

Storage Hierarchy



BOSTON UNIVERSITY

Tapes

Data size grows exponentially!

Cheaper capacity:

Increase density (bits/in^2)
Simpler devices

Tapes:

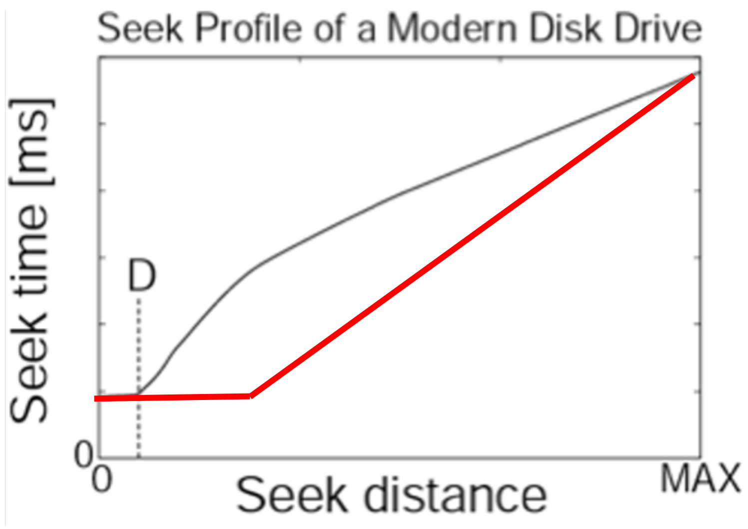
Magnetic medium that allows only **sequential access** (yes like an old-school tape)!

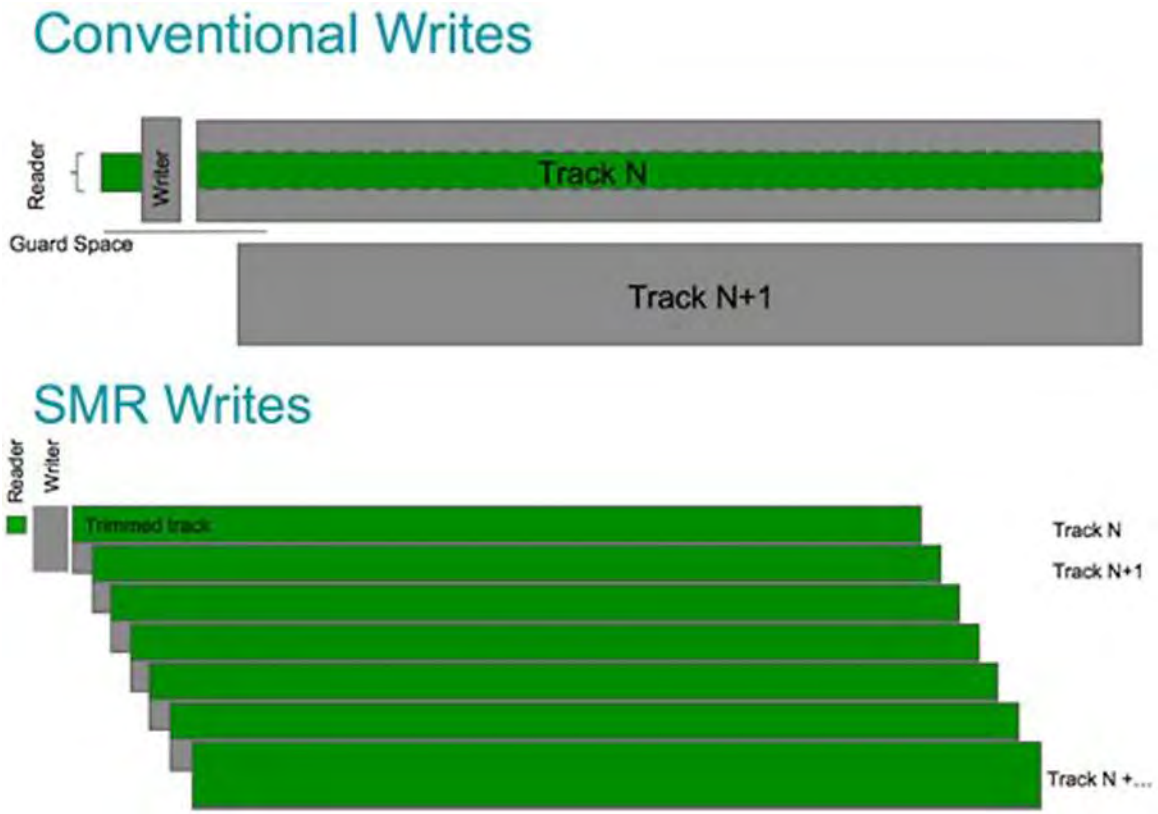
BOSTON UNIVERSITY

Increasing disk density

Very difficult to differentiate between tracks
“settle” time becomes

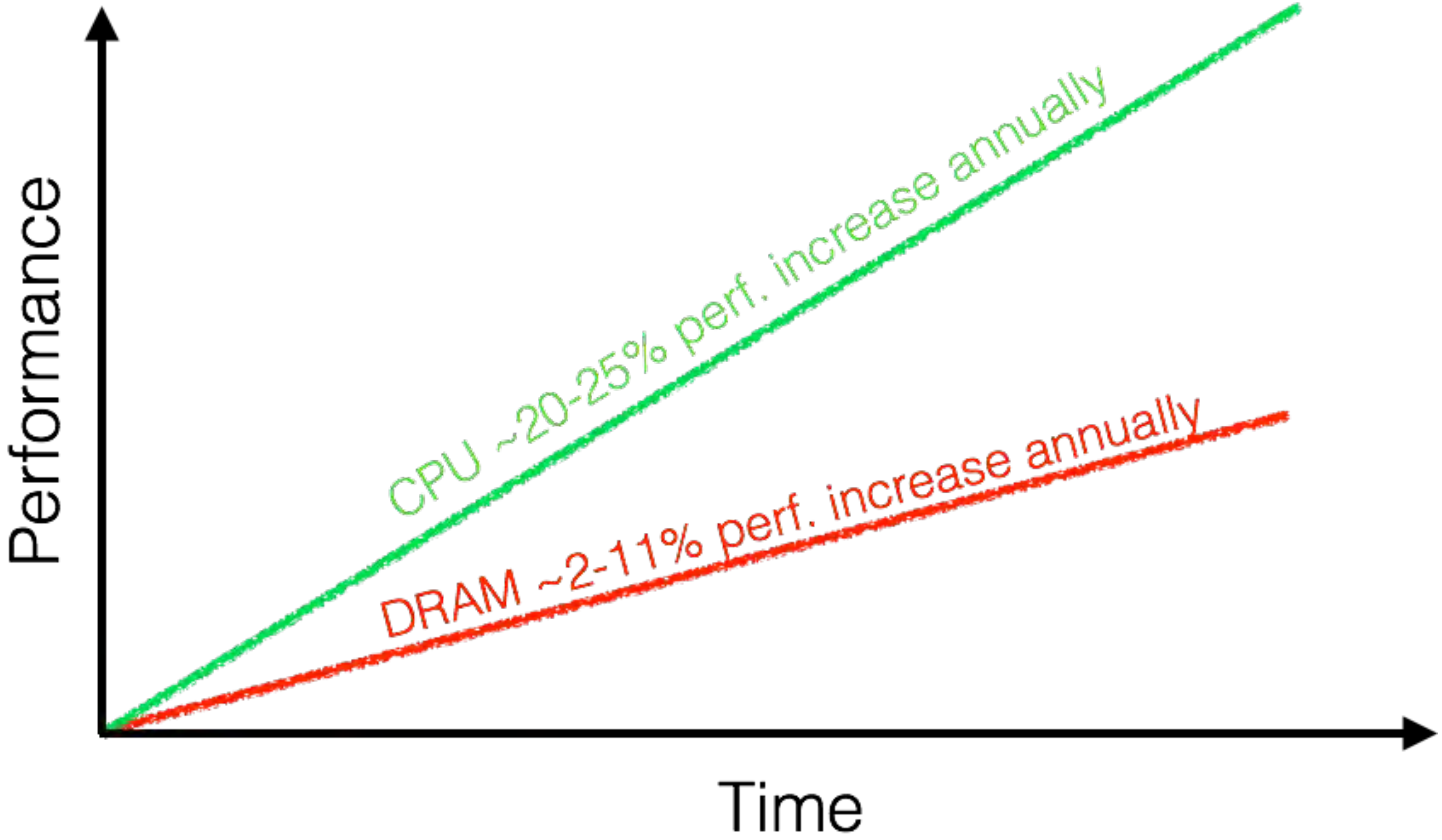
Writing a track affects neighboring tracks
Create different readers/writers
Interleave writes tracks





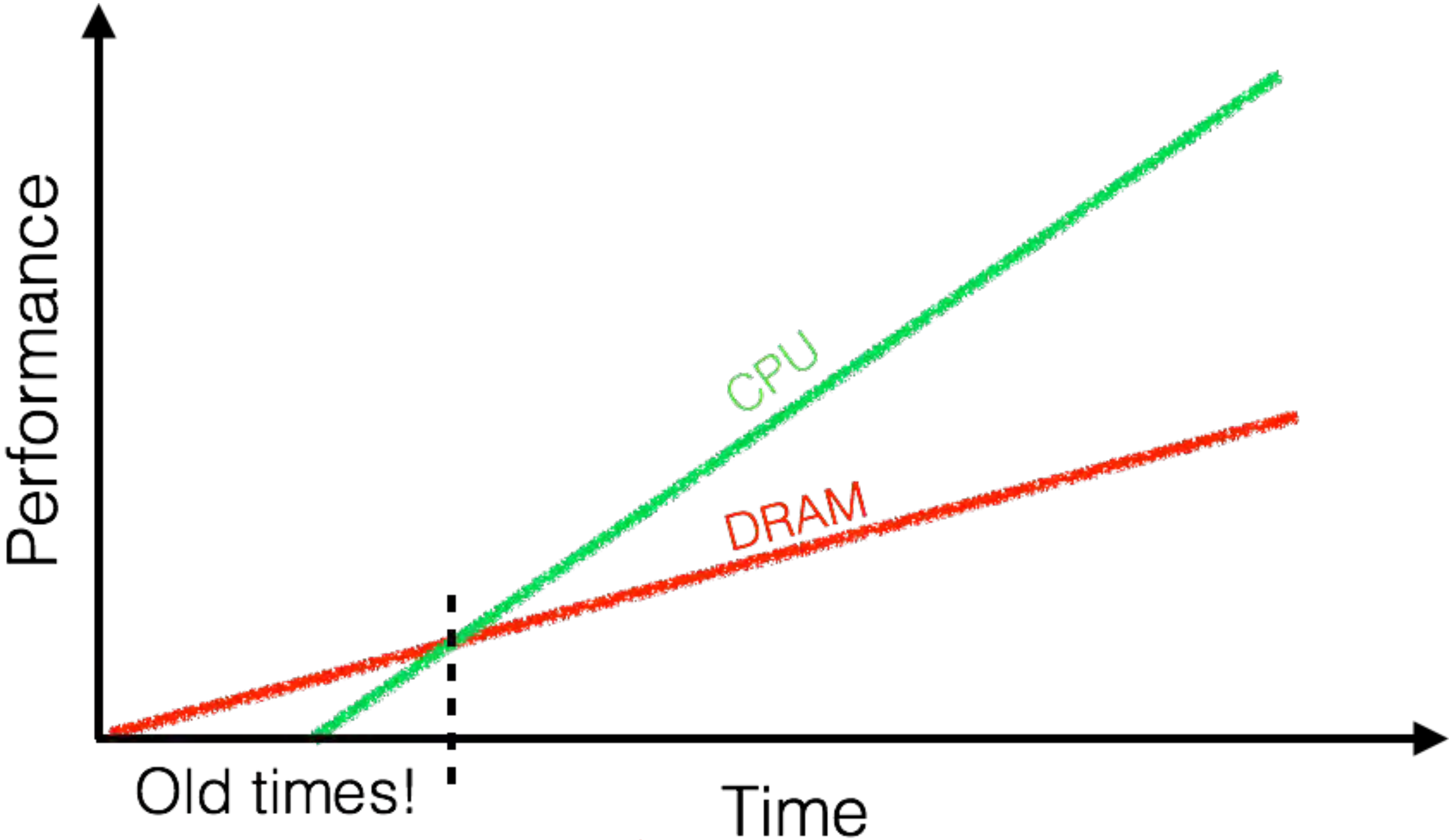
Memory & Storage Walls

Memory Wall



BOSTON UNIVERSITY

Memory Wall



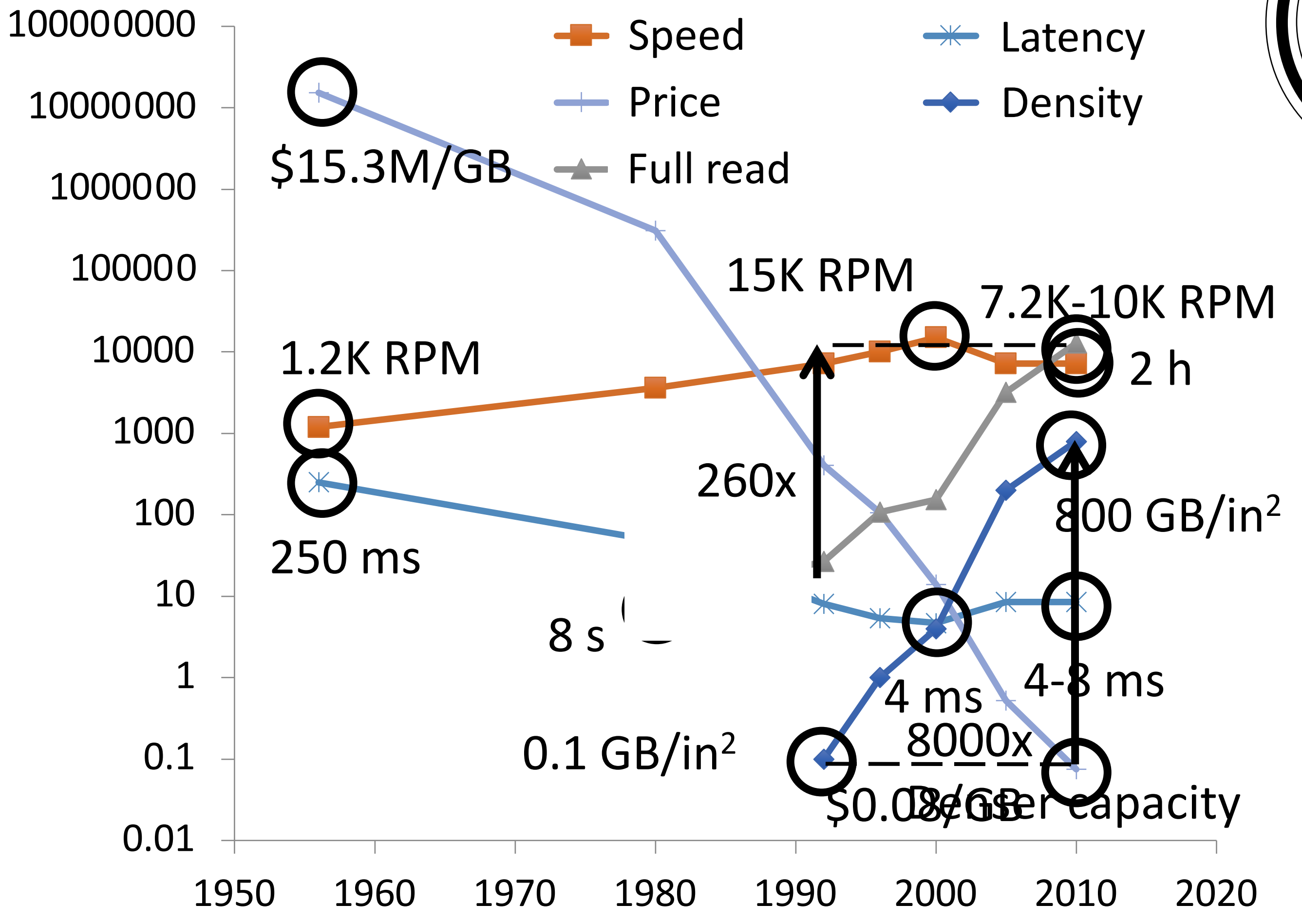
every byte counts

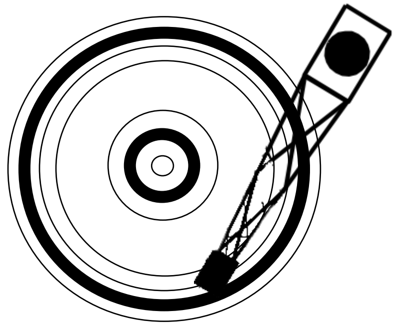
Storage Wall

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

BOSTON UNIVERSITY

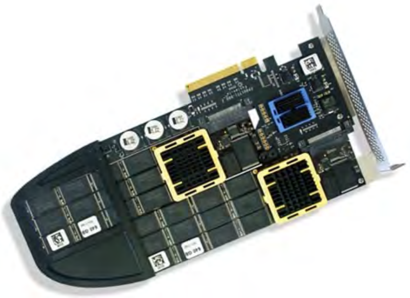
Evolution of hard disks





disks become larger but – relatively – slower

Storage Wall



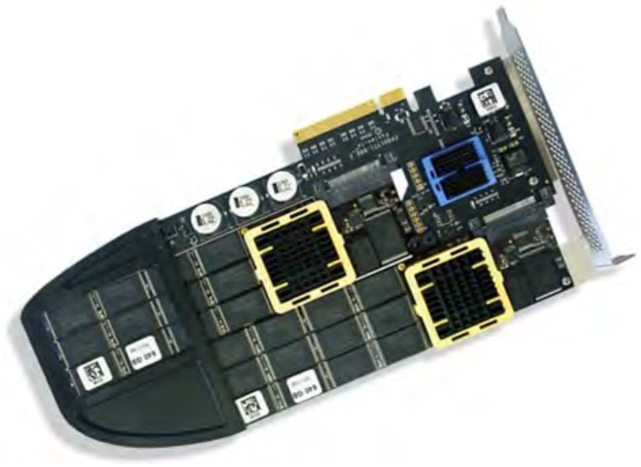
HDD
- ✓ capacity
- ✓ sequential access
- × random access
- × latency plateaus

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

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

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

- Storage without mechanical limitations

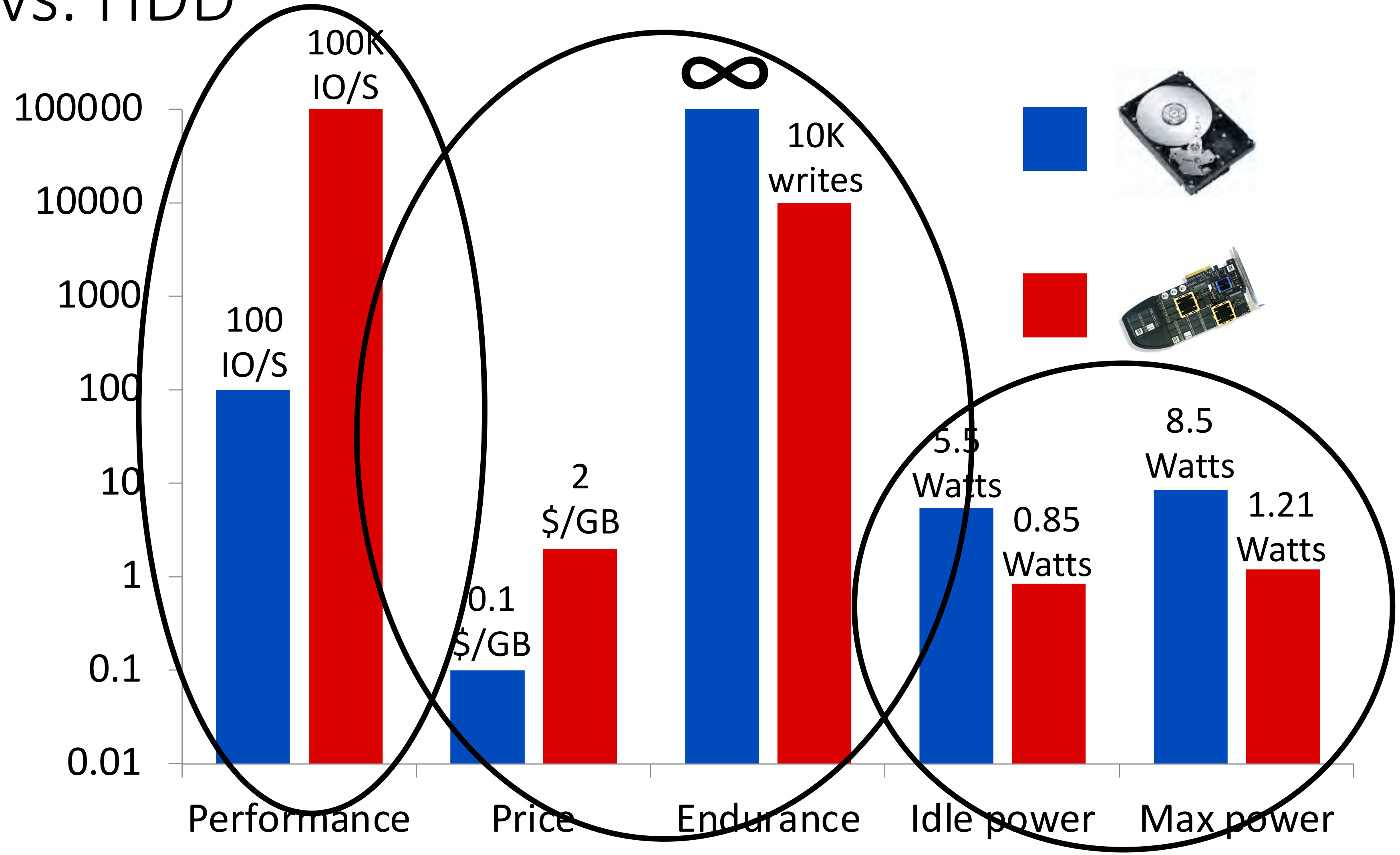


Several technologies

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

flash vs. hard disks?

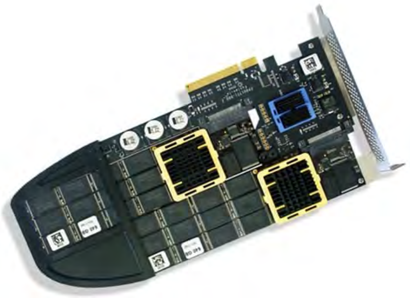
Flash vs. HDD



read, update, storage tradeoffs

BOSTON UNIVERSITY

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

BOSTON UNIVERSITY

Technology Trends & Research Challenges

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

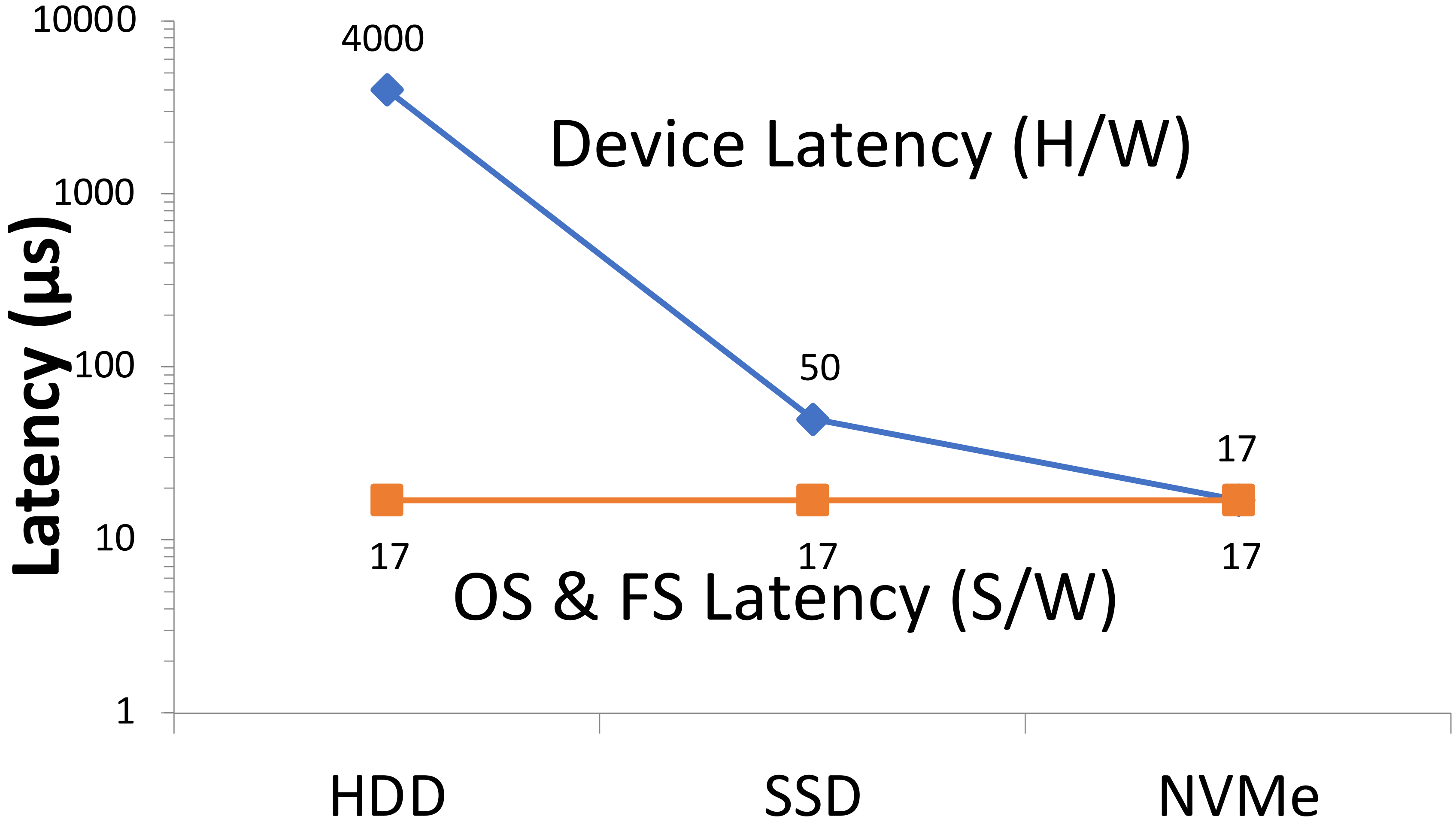
How to redesign systems for efficient random reads?

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

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 15

Data Systems on Modern Hardware:

Multi-cores, Solid-State Drives, and Non-Volatile Memories

Prof. Manos Athanassoulis

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

with slides developed at Harvard