CS 6530: Advanced Database Systems Fall 2022

# Lecture 20
# Databases on Modern Hardware

Prashant Pandey

prashant.pandey@utah.edu

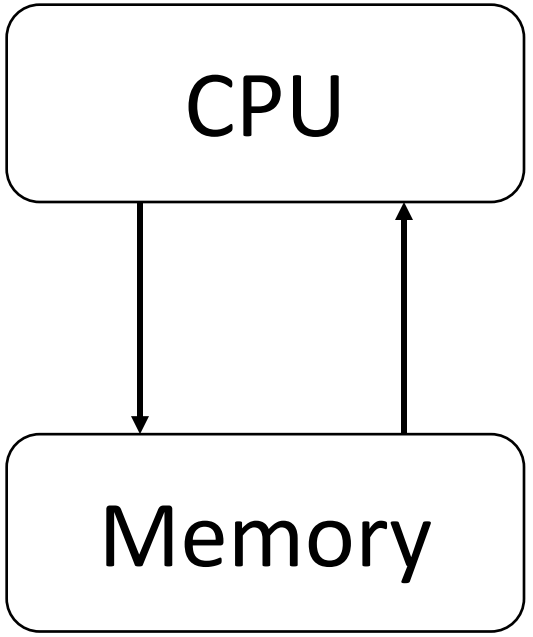
Acknowledgement: Slides taken from Prof. Manos Athanassoulis, BU

Prof. Xiangyao Yu, University of Wisconsin

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

# 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"

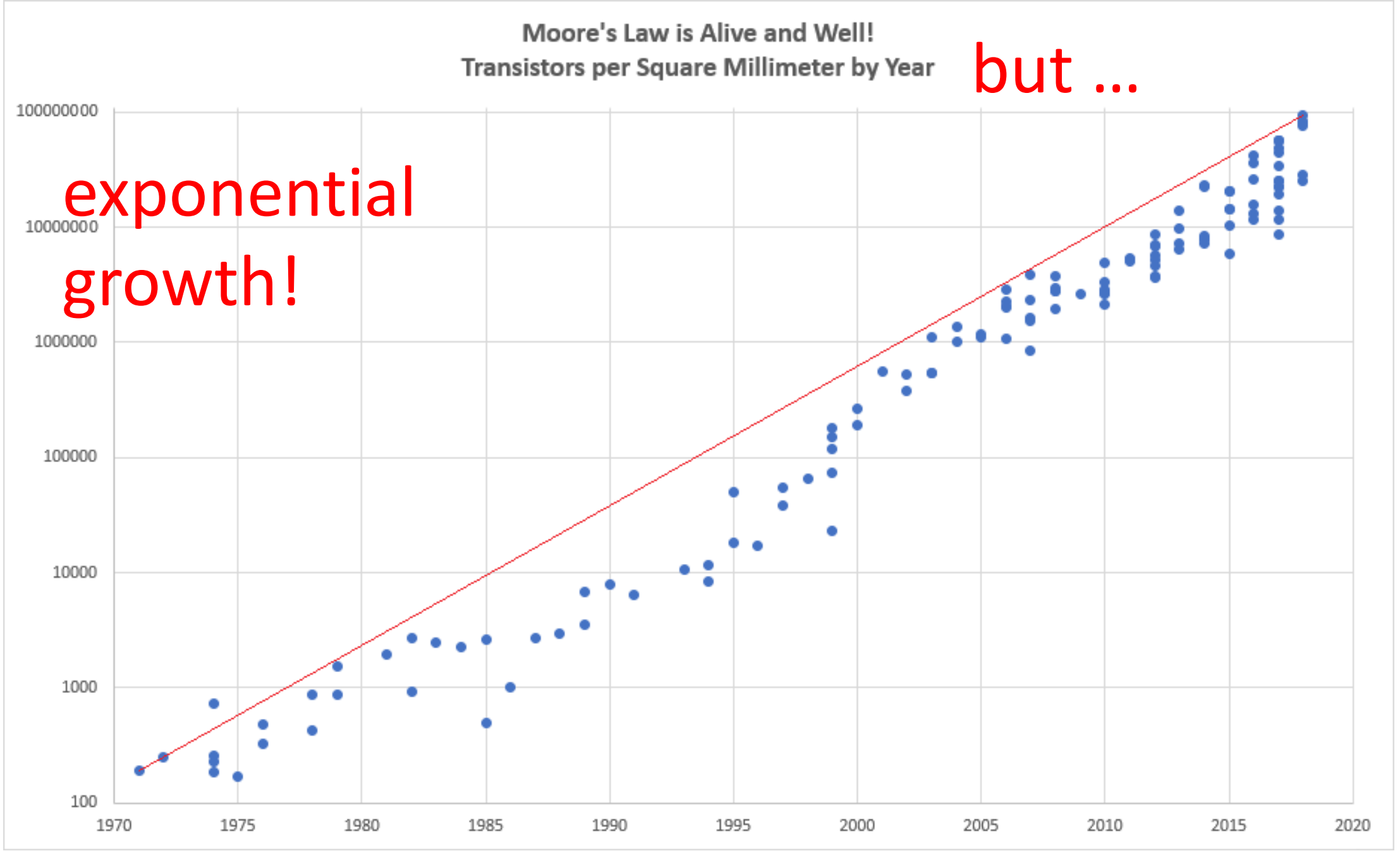
CPU clock speed

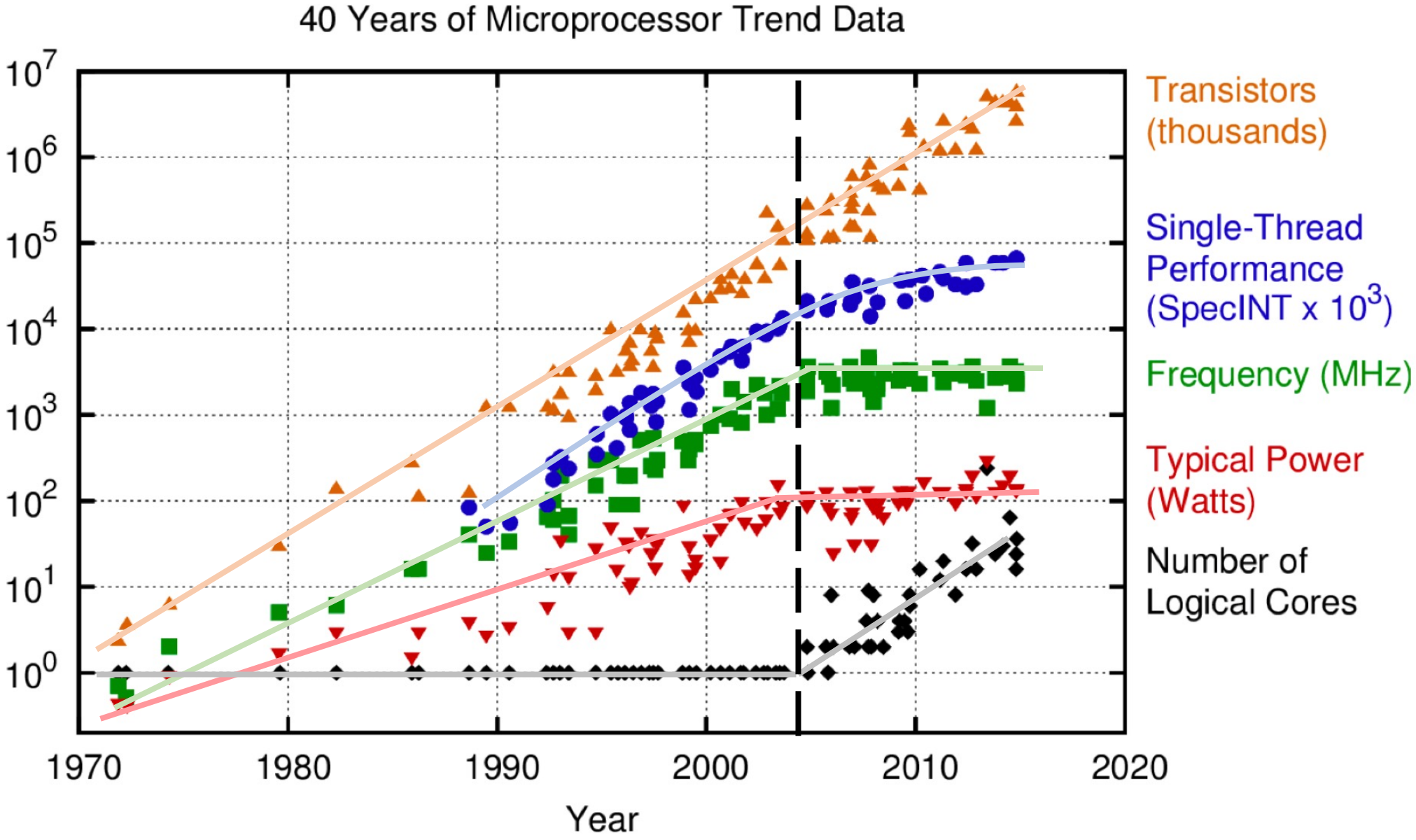
the number of transistors per chip

*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!!!

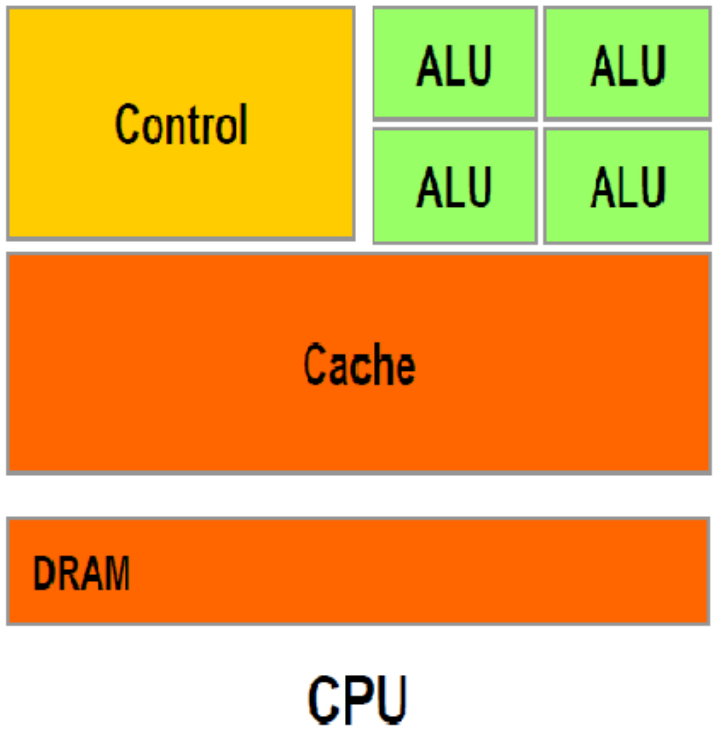
.. but they are getting more and more (**higher parallelism**)

***Research Challenges***

*how to handle them?*

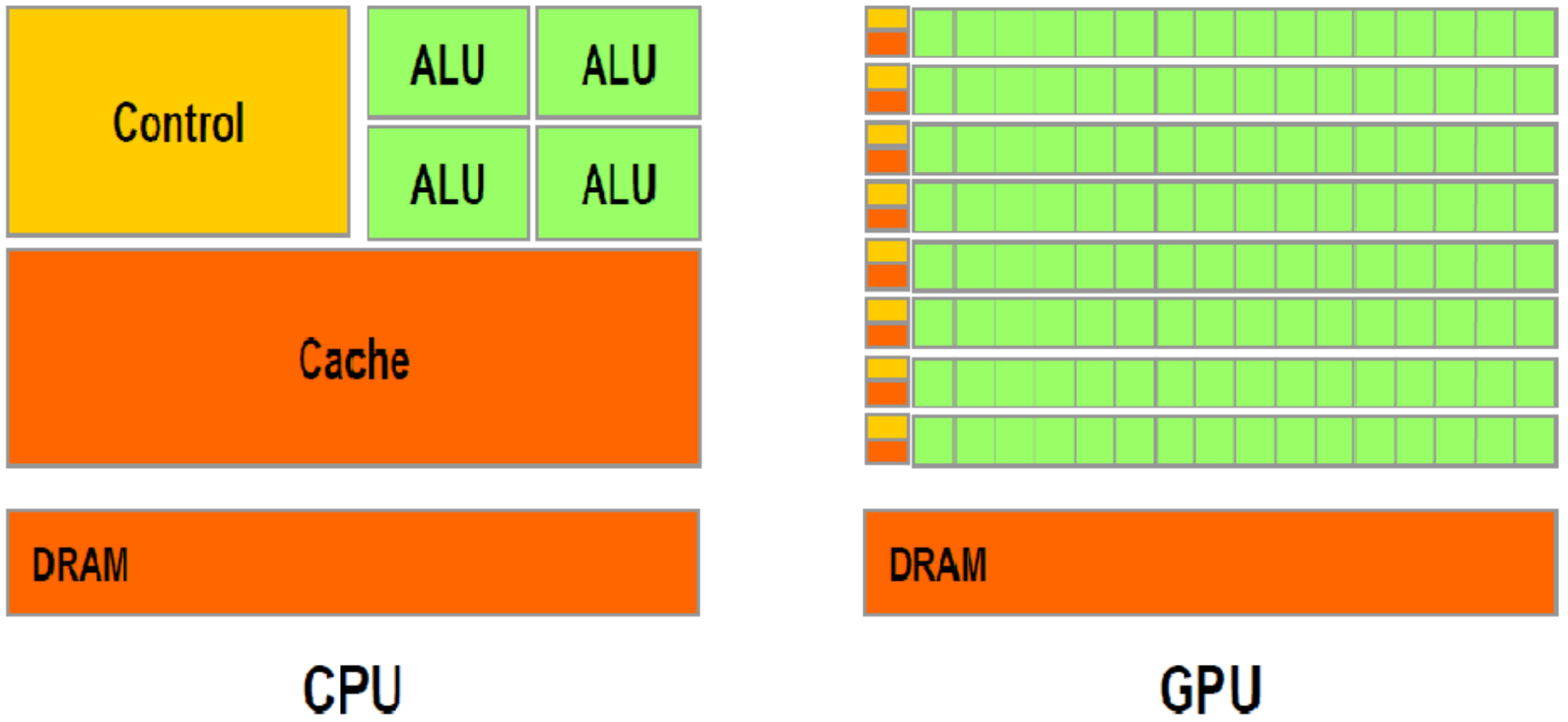
*how to parallel program?*

# CPU vs. GPU




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

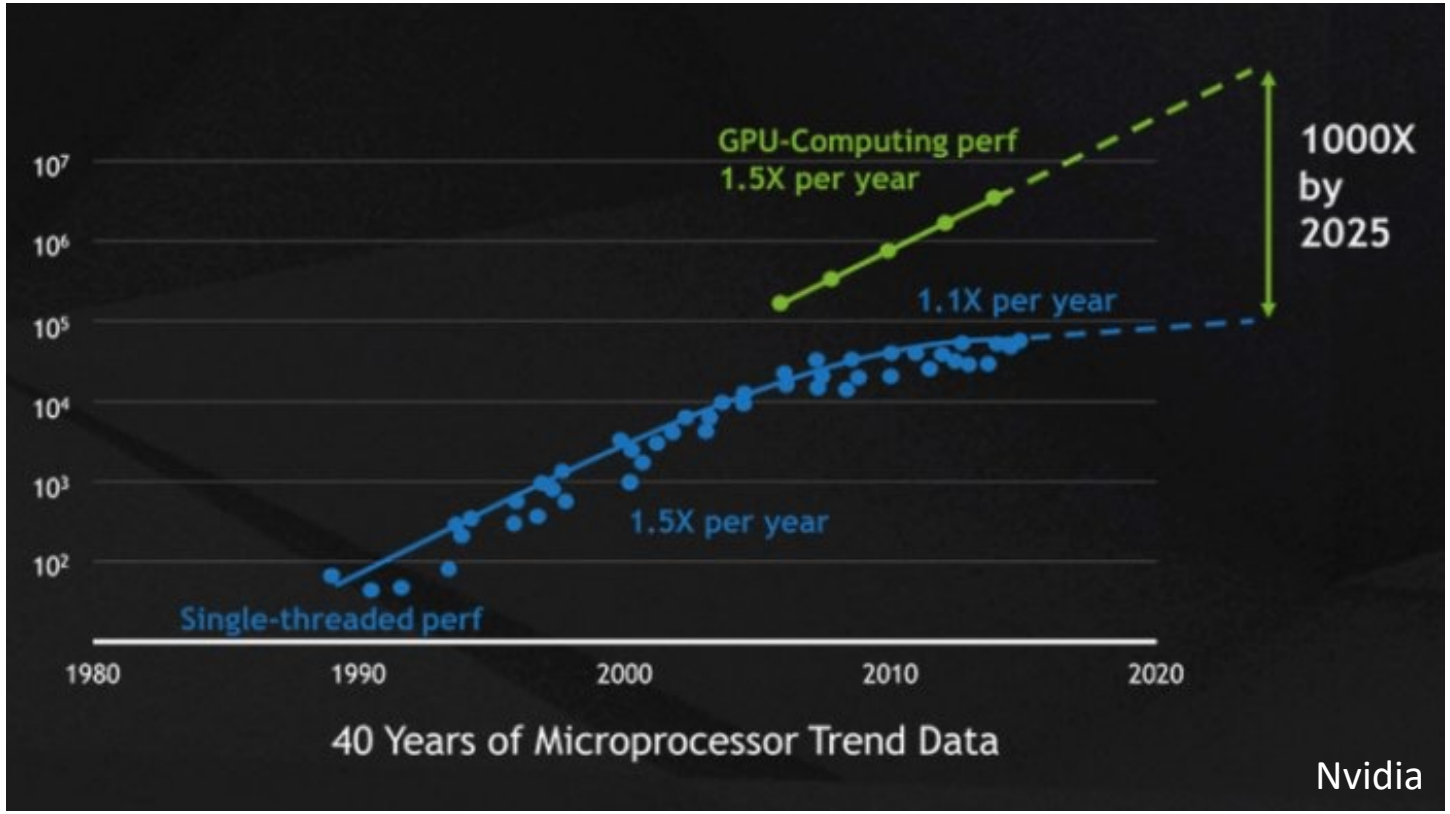
# CPU vs. GPU




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

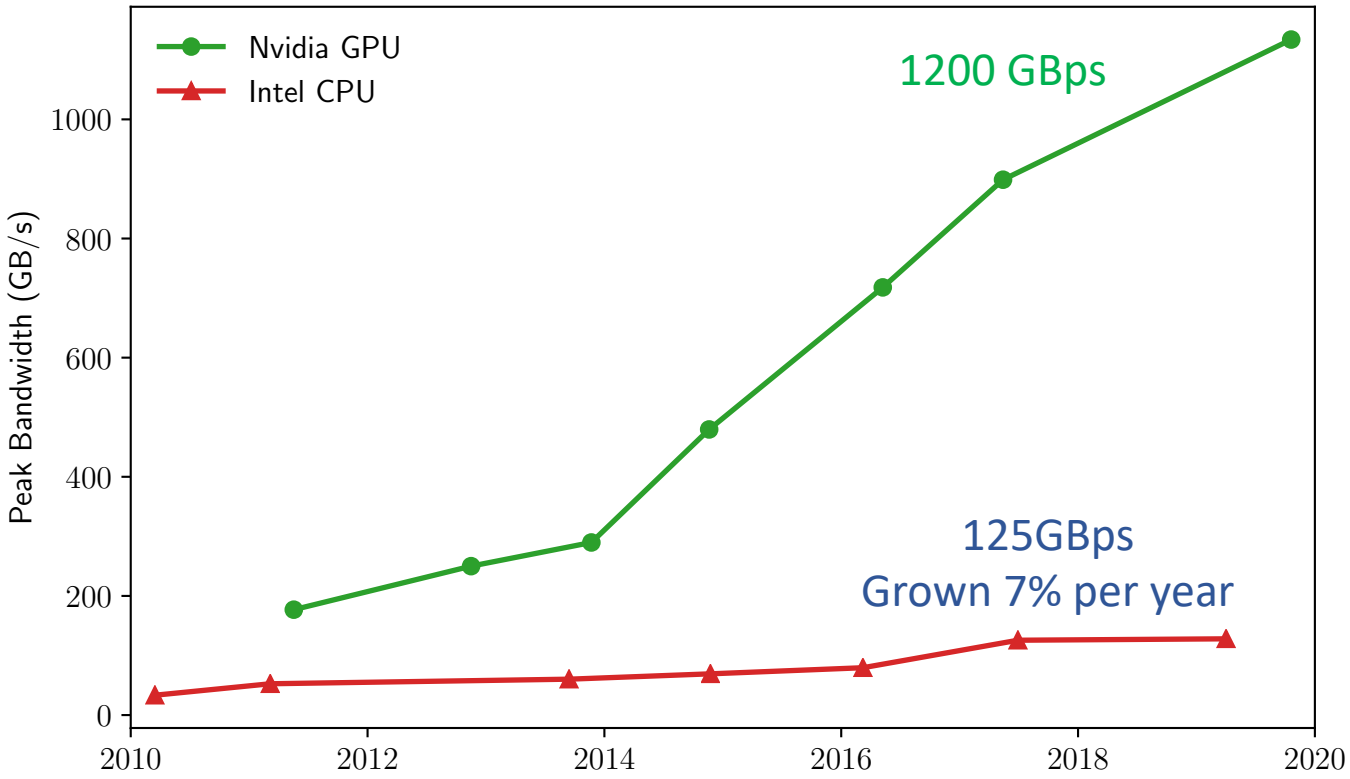
**GPU**: Many small cores. Optimized for parallel computation

# CPU vs. GPU – Processing Units




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

# CPU vs. GPU — Memory Bandwidth



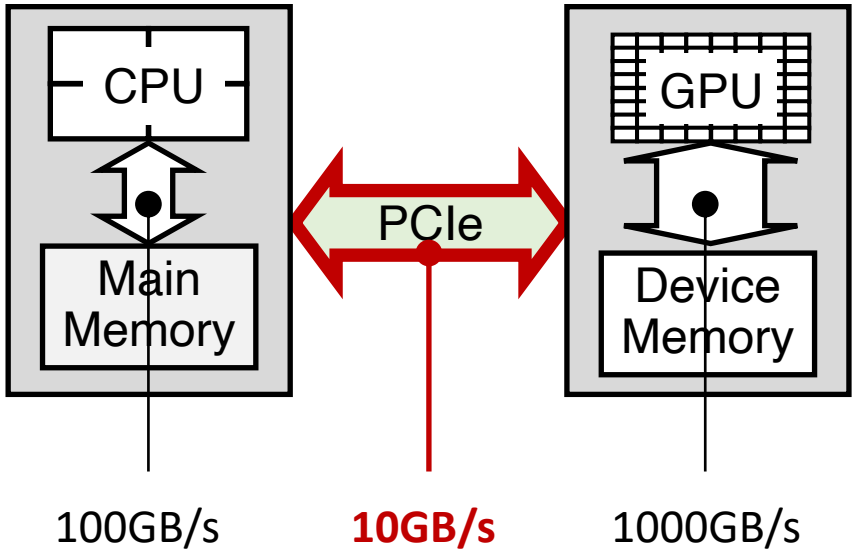

**GPU has one order of magnitude higher memory bandwidth than CPU**

Memory Bandwidth is the bottleneck for in-memory analytics

A natural idea: **use GPUs for data analytics**

# GPU-DB Limitations

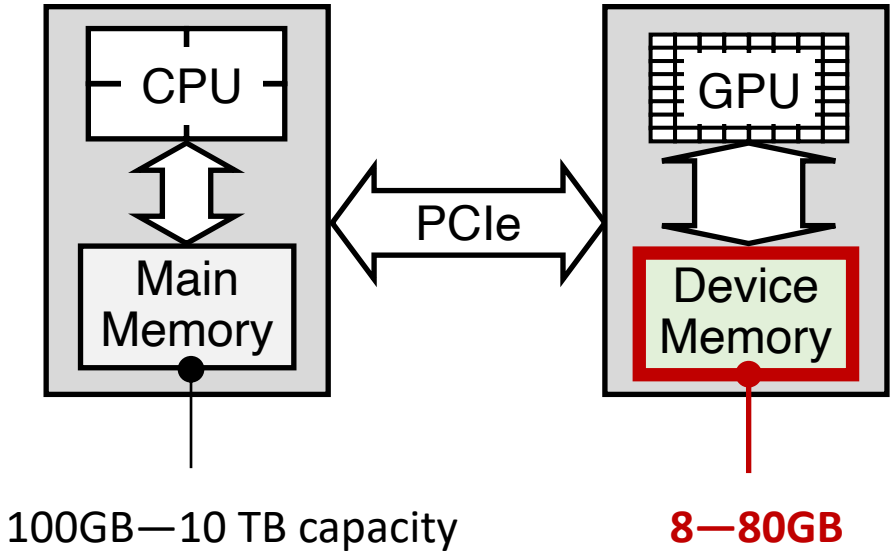
Limitation 1: Low interconnect bandwidth



# GPU-DB Limitations

Limitation 1: Low interconnect bandwidth

Limitation 2: Small GPU memory capacity

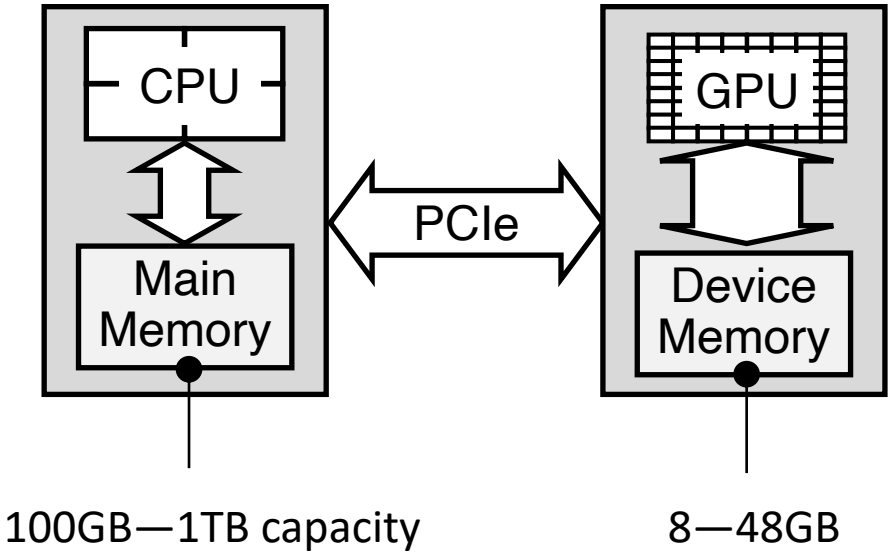


# GPU-DB Limitations

Limitation 1: Low interconnect bandwidth

Limitation 2: Small GPU memory capacity

Limitation 3: Coarse-grained cooperation of CPU and GPU

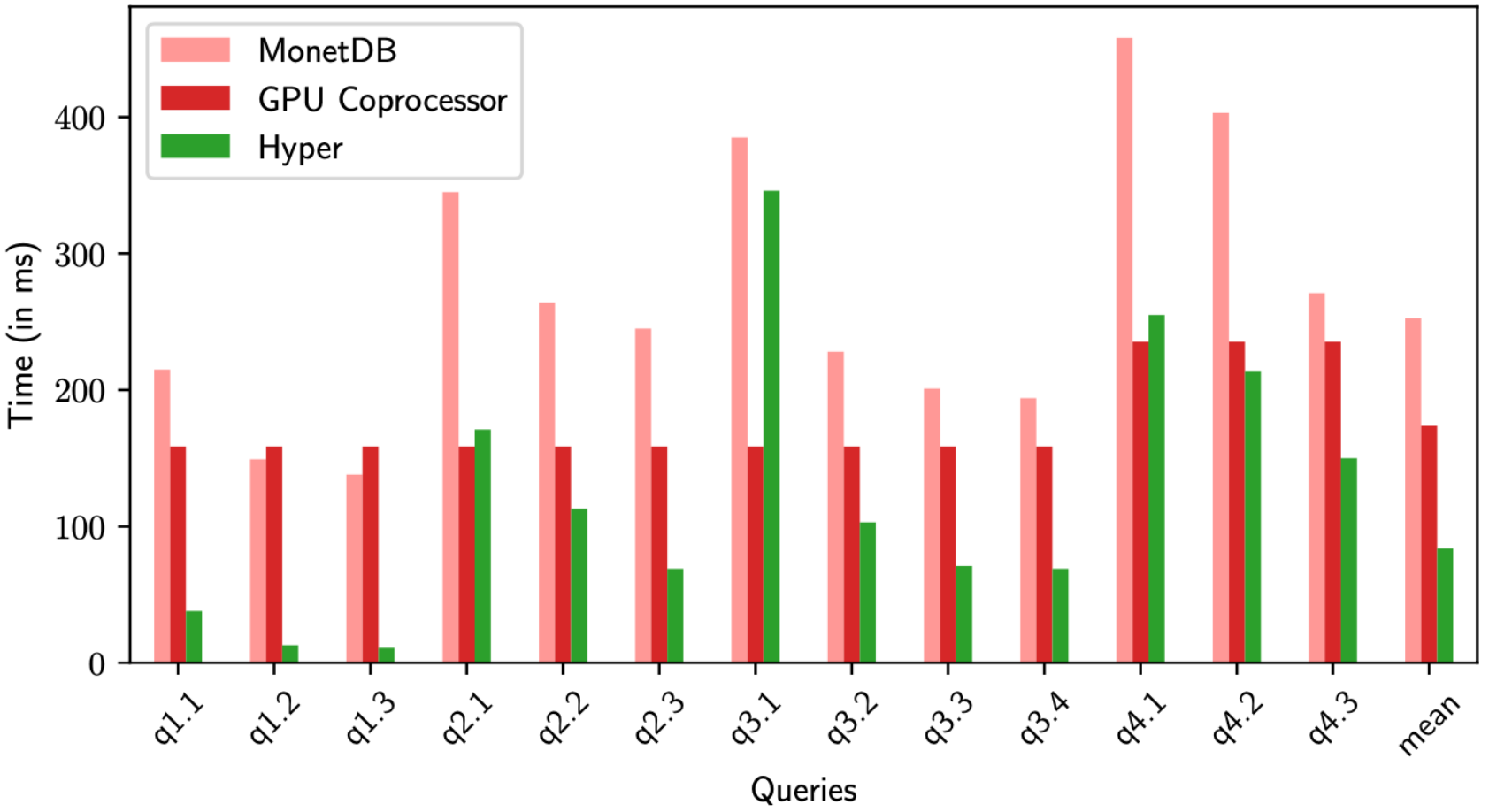


# GPU Database Operation Mode

**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

# CPU-only vs. Coprocessor




**Key observation**: With efficient implementations that can saturate memory bandwidth

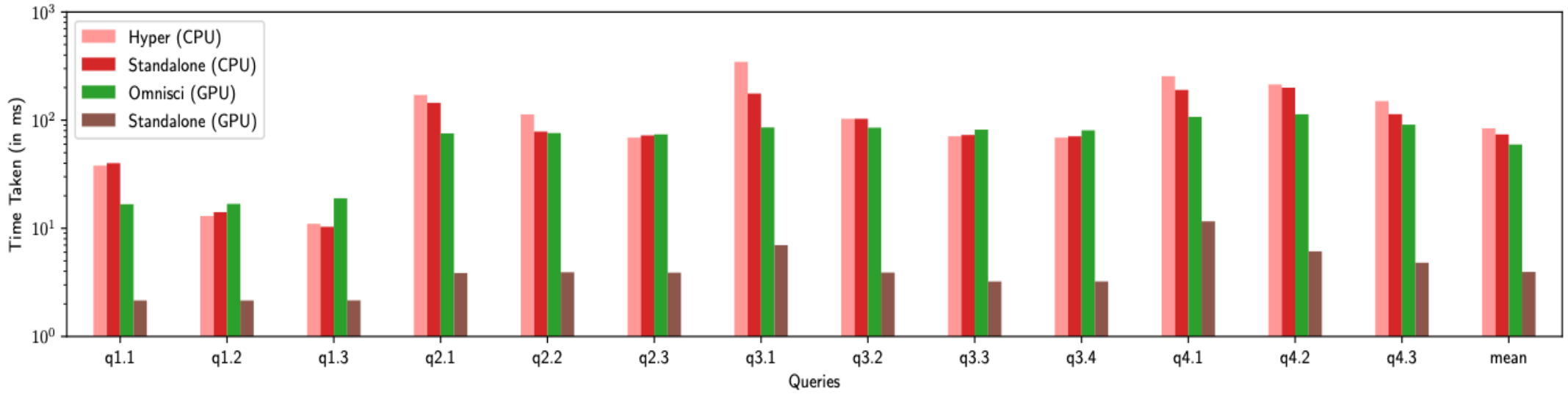
**GPU-only > CPU-only > coprocessor**

# Star-Schema Benchmark

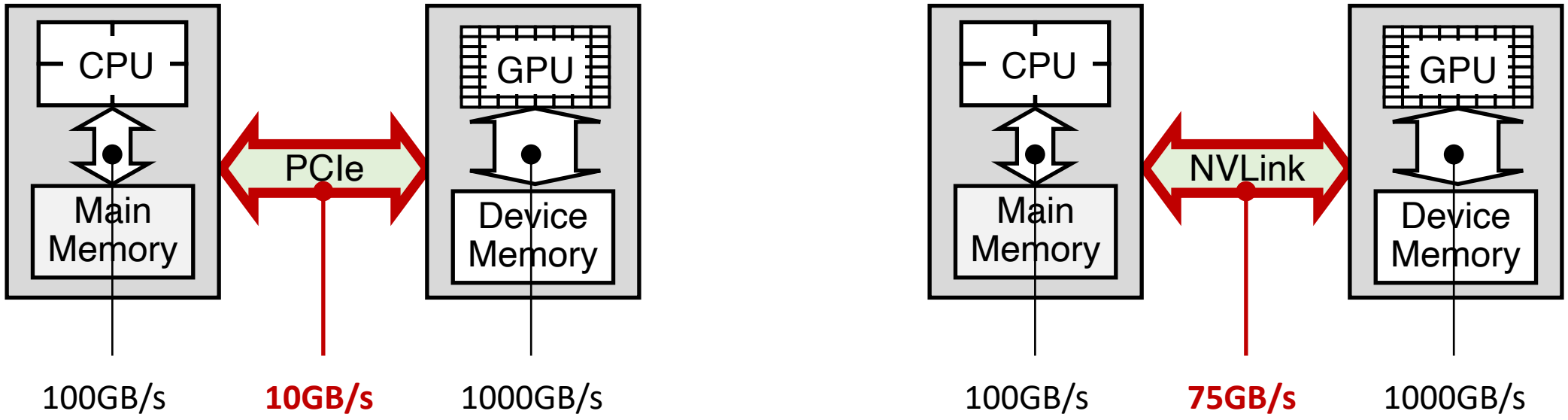
| Platform | CPU | GPU |
|---|---|---|
| Model | Intel i7-6900 | Nvidia V100 |
| Cores | 8 (16 with SMT) | 5000 |
| Memory Capacity | 64 GB | 32 GB |
| L1 Size | 32KB/Core | 16KB/SM |
| L2 Size | 256KB/Core | 6MB (Total) |
| L3 Size | 20MB (Total) | - |
| Read Bandwidth | 53GBps | 880GBps |
| Write Bandwidth | 55GBps | 880GBps |
| L1 Bandwidth | - | 10.7TBps |
| L2 Bandwidth | - | 2.2TBps |
| L3 Bandwidth | 157GBps | - |

Crystal-based implementations always saturate GPU memory bandwidth

GPU is on average **25X** faster than CPU



# Emerging Fast Interconnect



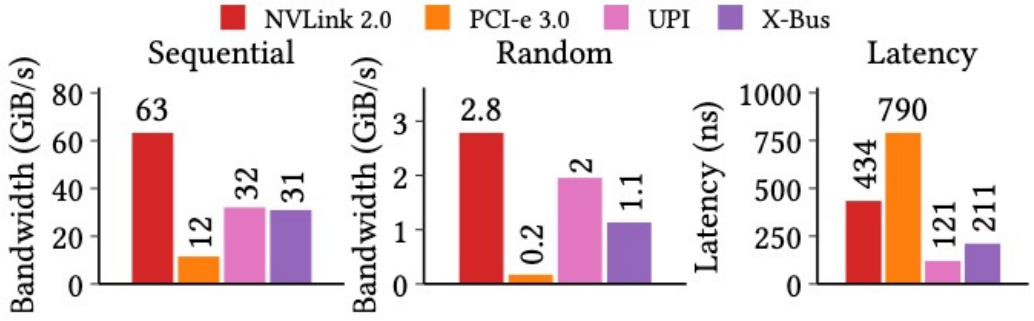

Fast Interconnect can solve the PCIe bottleneck

Emerging alternative interconnect technologies:

- NVLink
- Infinity Fabric
- Compute Express Link (CXL)

# NVLink Bandwidth and Latency

NVLink has much higher bandwidth than PCIe

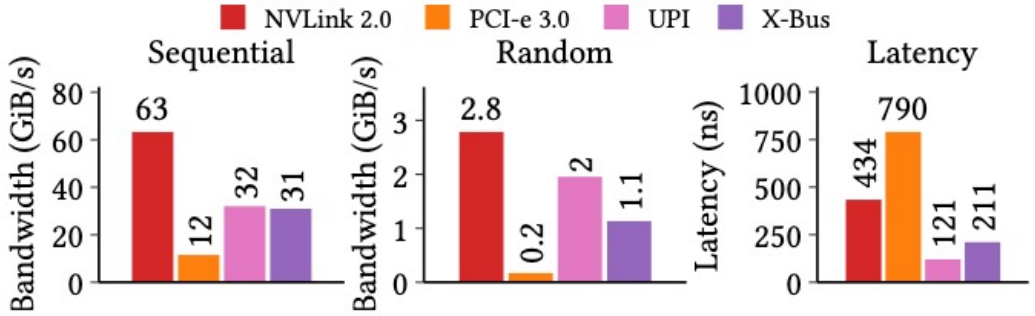



**(a) NVLink 2.0 vs. CPU & GPU Interconnects.**

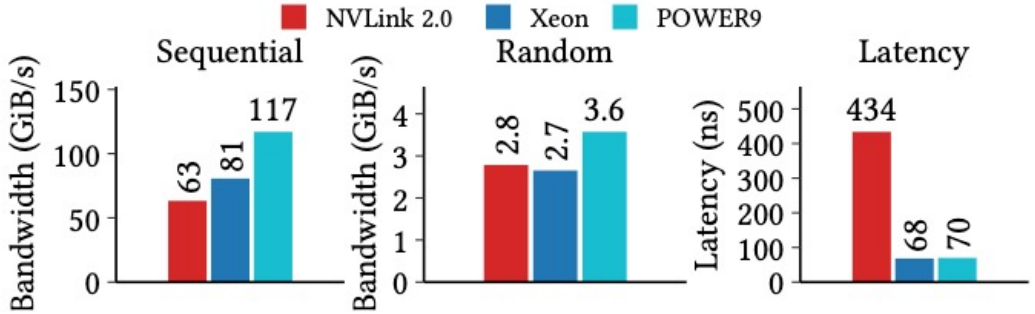
# NVLink Bandwidth and Latency

NVLink has much higher bandwidth than PCIe

NVLink has comparable bandwidth as CPU local memory




(a) NVLink 2.0 vs. CPU & GPU Interconnects.



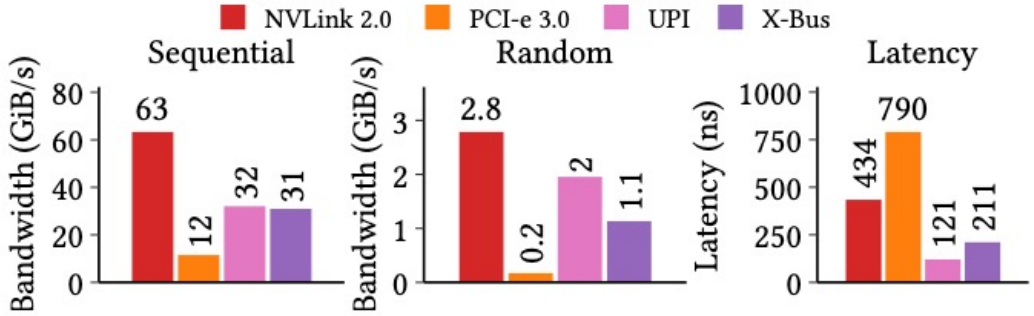

(b) NVLink 2.0 vs. CPU memory.

# NVLink Bandwidth and Latency

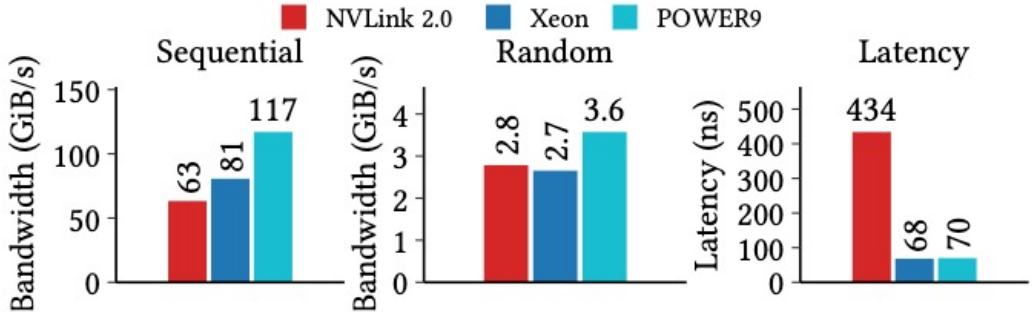
NVLink has much higher bandwidth than PCIe

NVLink has comparable bandwidth as CPU local memory

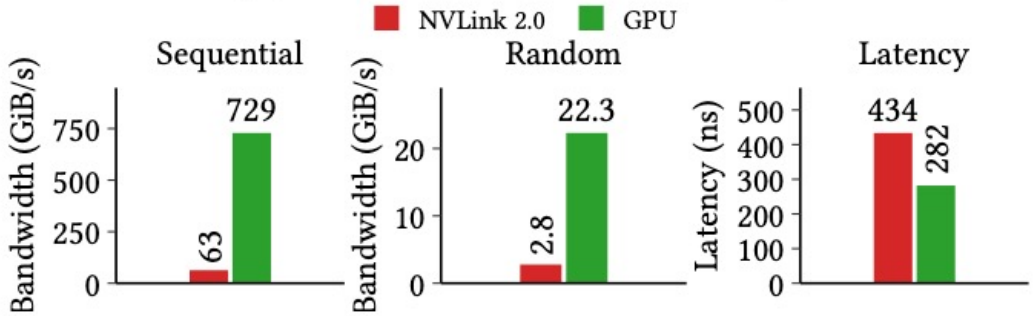
NVLink bandwidth has much lower bandwidth than GPU memory




(a) NVLink 2.0 vs. CPU & GPU Interconnects.




(b) NVLink 2.0 vs. CPU memory.




(c) NVLink 2.0 vs. GPU memory.

# GPU Transfer Methods

**Table 1: An overview of GPU transfer methods.**

| Method | Semantics | Level | Granularity | Memory |
|---|---|---|---|---|
| Pageable Copy<br>Staged Copy<br>Dynamic Pinning | Push | SW | Chunk | Pageable |
| Pinned Copy | | | | Pinned |
| UM Prefetch | | | | Unified |
| UM Migration | Pull | OS | Page | Unified |
| Zero-Copy | | HW | Byte | Pinned |
| Coherence | | | | Pageable |

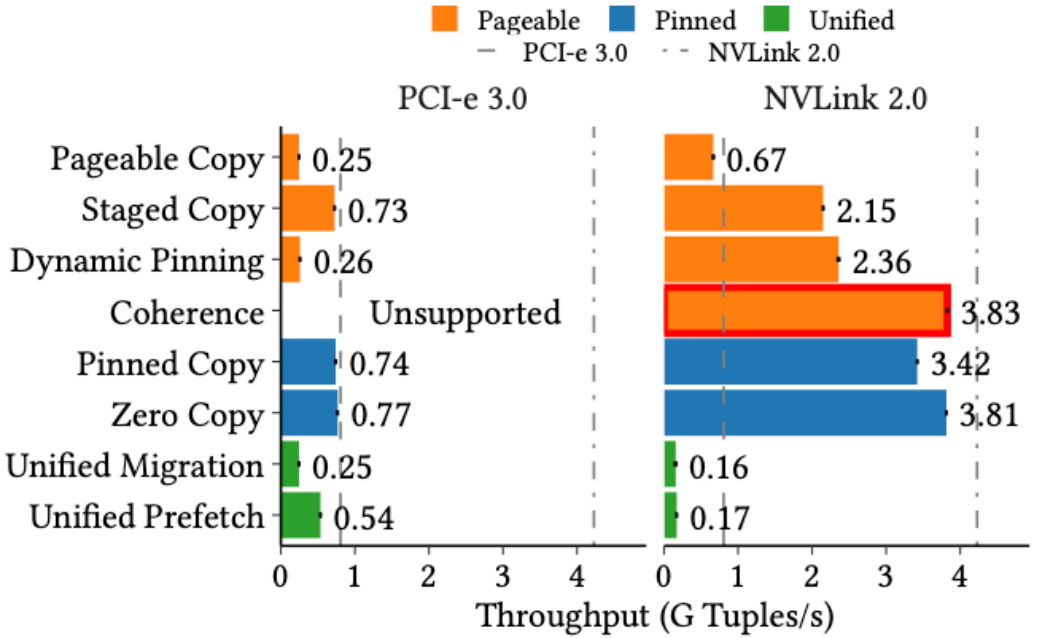



**Figure 12: No-partitioning hash join using different transfer methods for PCI-e 3.0 and NVLink 2.0.**

**Pinned copy** and **zero copy** can saturate PCIe bandwidth

**Coherence** can saturate NVLink bandwidth

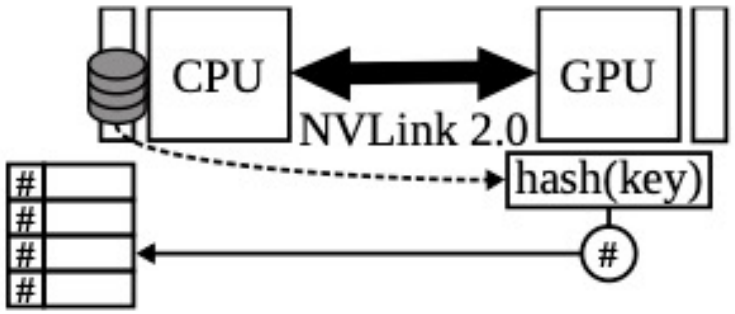
# Non-Partitioned Hash Join Methods

**Build phase**: build the hash table using inner relation R

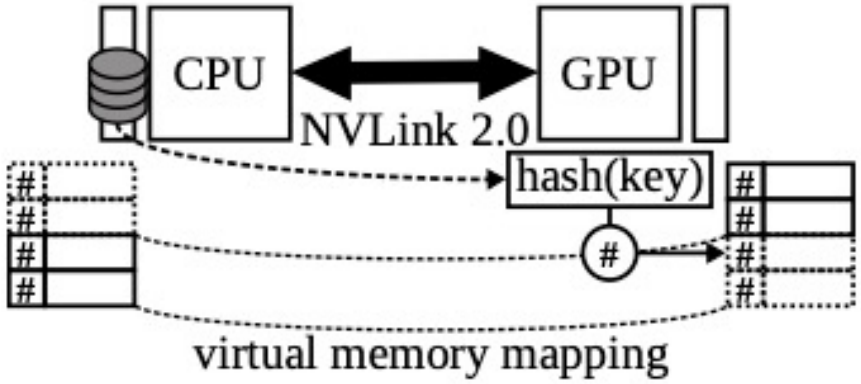
**Probe phase**: lookup hash table for each record in outer relation S

# Hash Join – Build Phase

Build phase: build the hash table using inner relation R



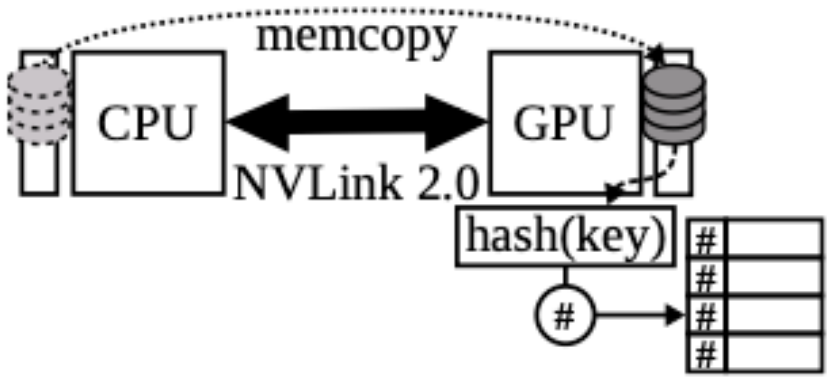

(a) Data and hash table in CPU memory.



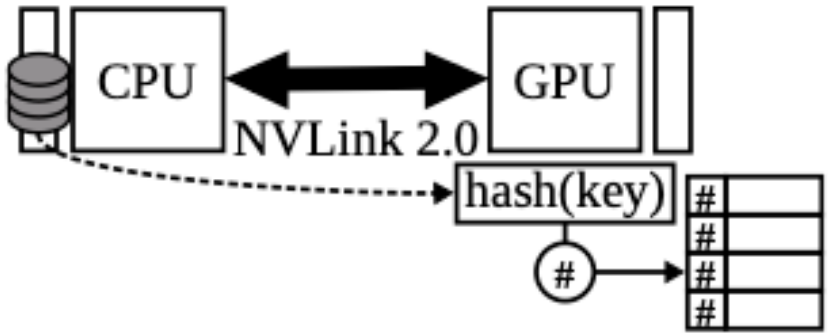

(b) Data in CPU memory and hash table spills from GPU memory into CPU memory.

# Hash Join – Probe Phase

Probe phase: lookup hash table for each record in outer relation S

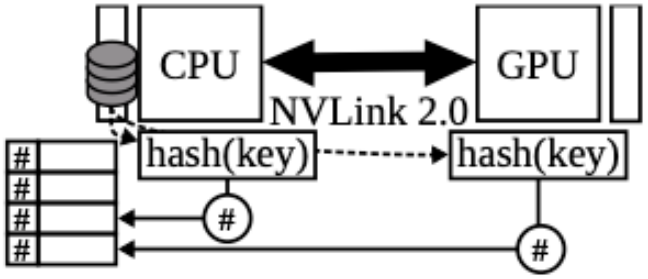



**(a) Data and hash table in GPU memory.**

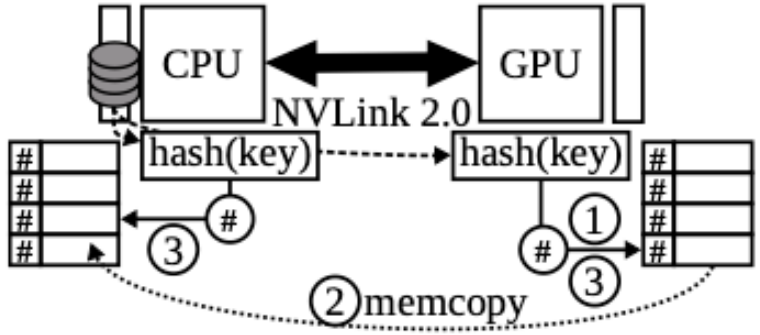



**(b) Data in CPU memory and hash table in GPU memory.**

# Hash Join




(a) Cooperatively process join on CPU and GPU with hash table in CPU memory.




(b) Build hash table on GPU, copy the hash table to processor-local memories, and then cooperatively probe on CPU and GPU.

This **hybrid design** subsumes the previous designs in the paper

- Dynamically schedule tasks to both CPU and GPU

# Hash Table Locality

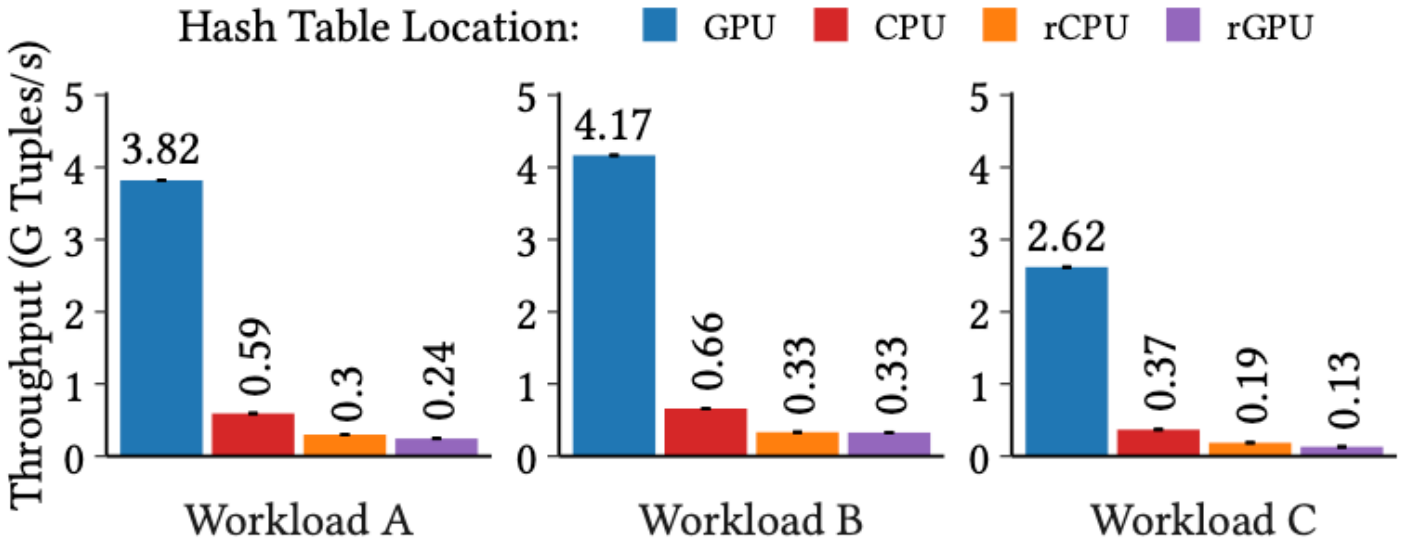



**Figure 14: Join performance of the GPU when the hash table is located on different processors, increasing the number of interconnect hops from 0 to 3.**

Best performance achieved when the hash table is in GPU memory

# Scaling Data Size in TPC-H Q6

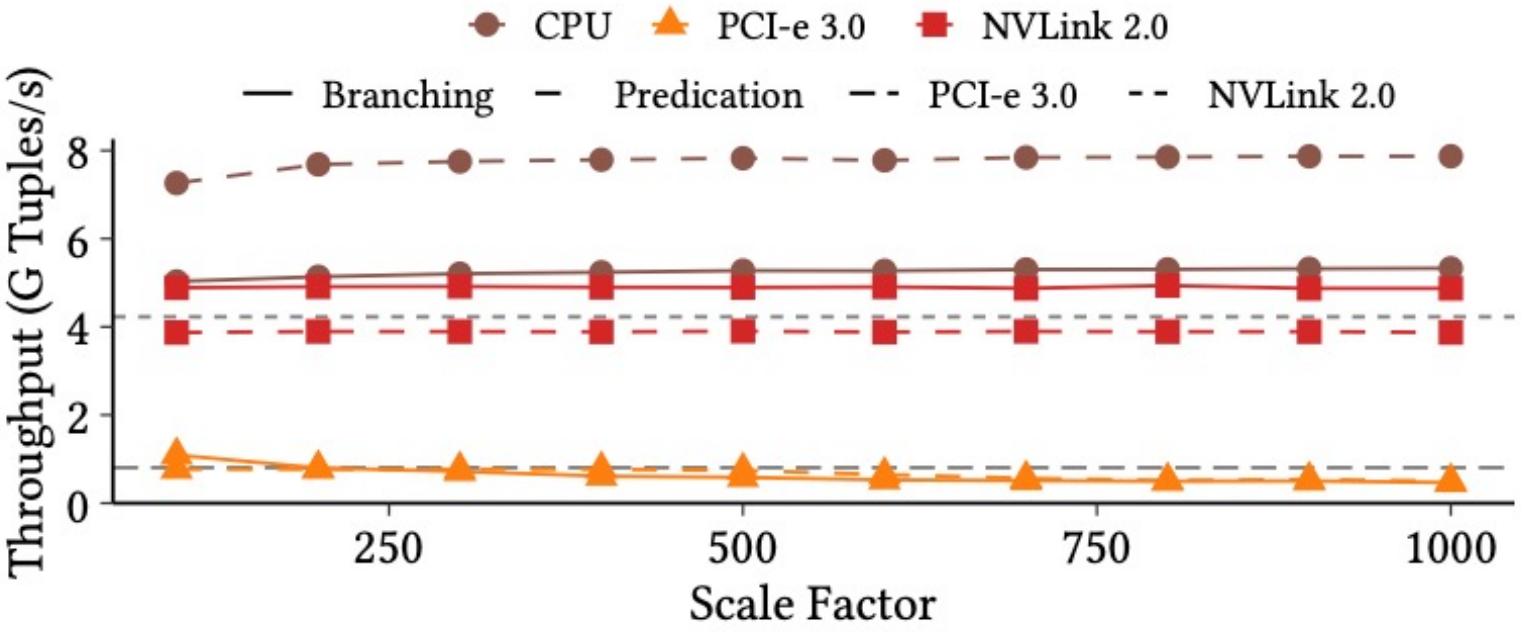



**Figure 15: Scaling the data size of TPC-H query 6.**

TPC-H Q6 contains a simple scan + aggregation with no join

Running the query on CPU leads to the highest performance

# Scaling the Probe Side Relation

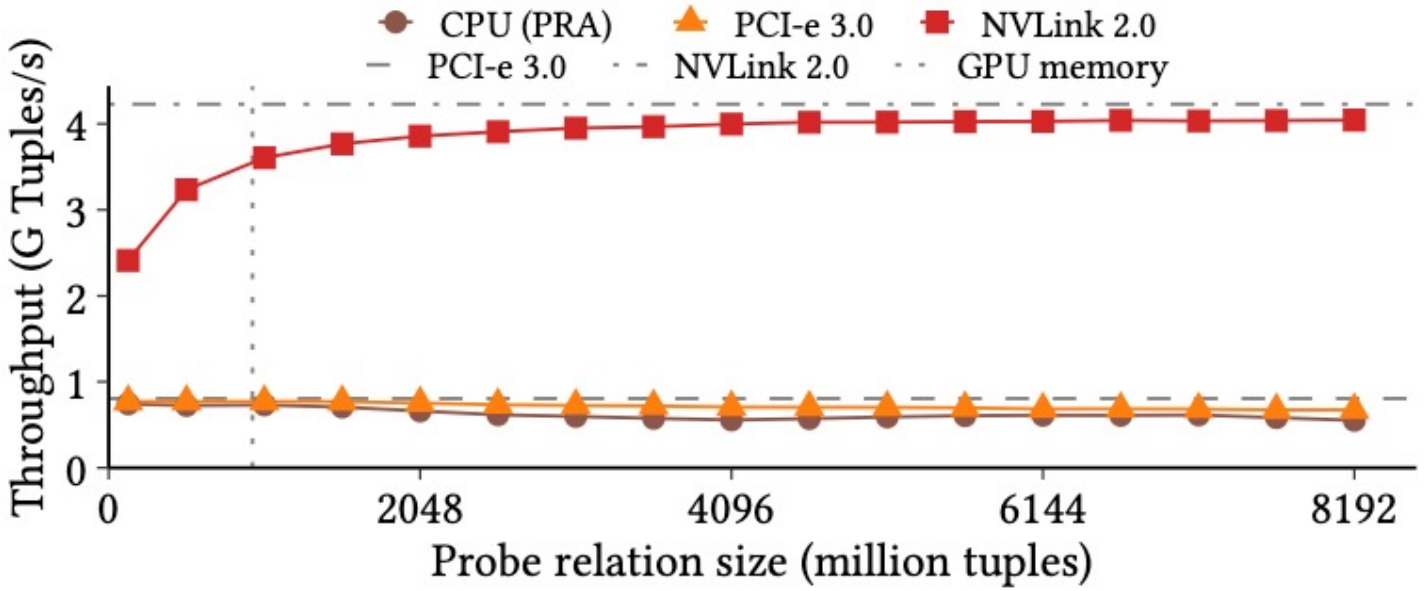



**Figure 16: Scaling the probe-side relation.**

NVLink is faster than both PCIe and CPU only

# Scaling the Build Side Relation

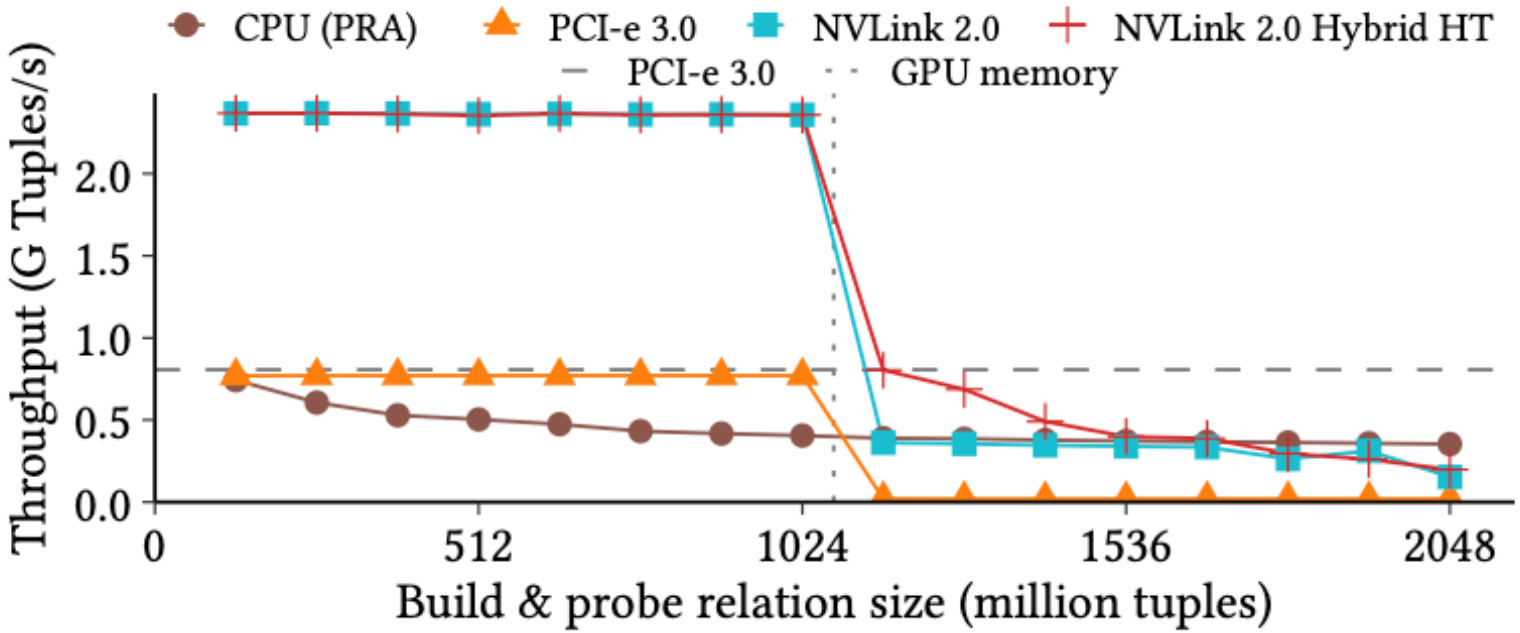



**Figure 17: Scaling the build-side relation.**

Performance drops when the hash table does not fit in GPU memory

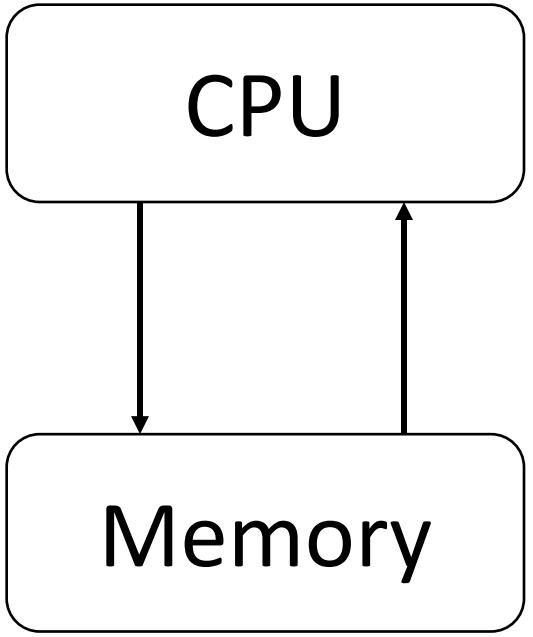
# Discussion

| | Crystal | NVLink |
|---|---|---|
| Query Type | SPJA analytical queries | Non-partitioned hash join |
| Execution Model | Data fits in GPU memory | Coprocessor |
| Interconnect | PCIe 3.0 | NVLink 2.0 |

**Research question**: How to maximize GPU database performance with different interconnect technology?

# Compute, Memory, and Storage Hierarchy

Traditional von-Neuman computer architecture




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

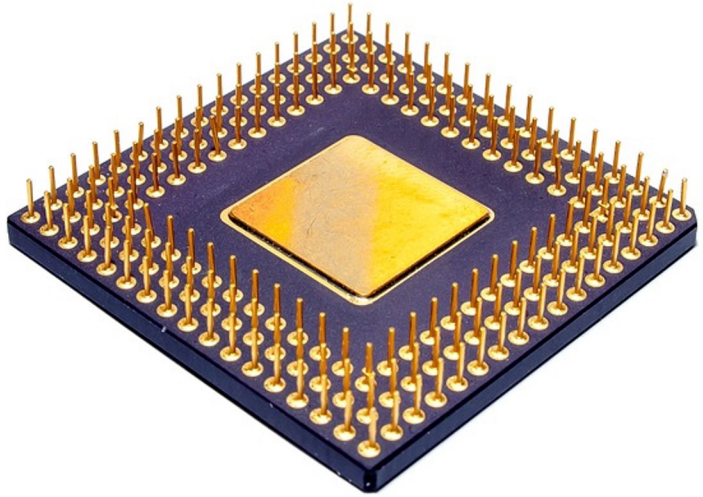
*not always!*

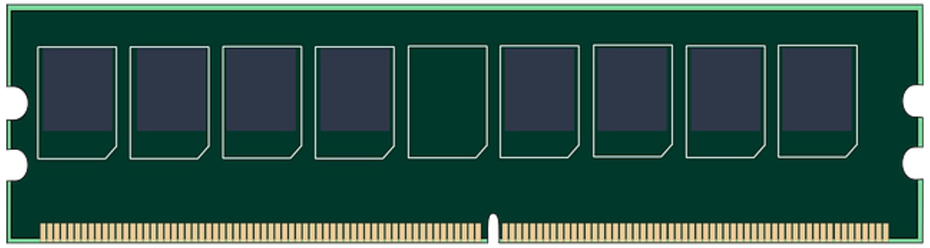
(ii) assumes memory can keep-up with CPU and can hold all data

*is this the case?*

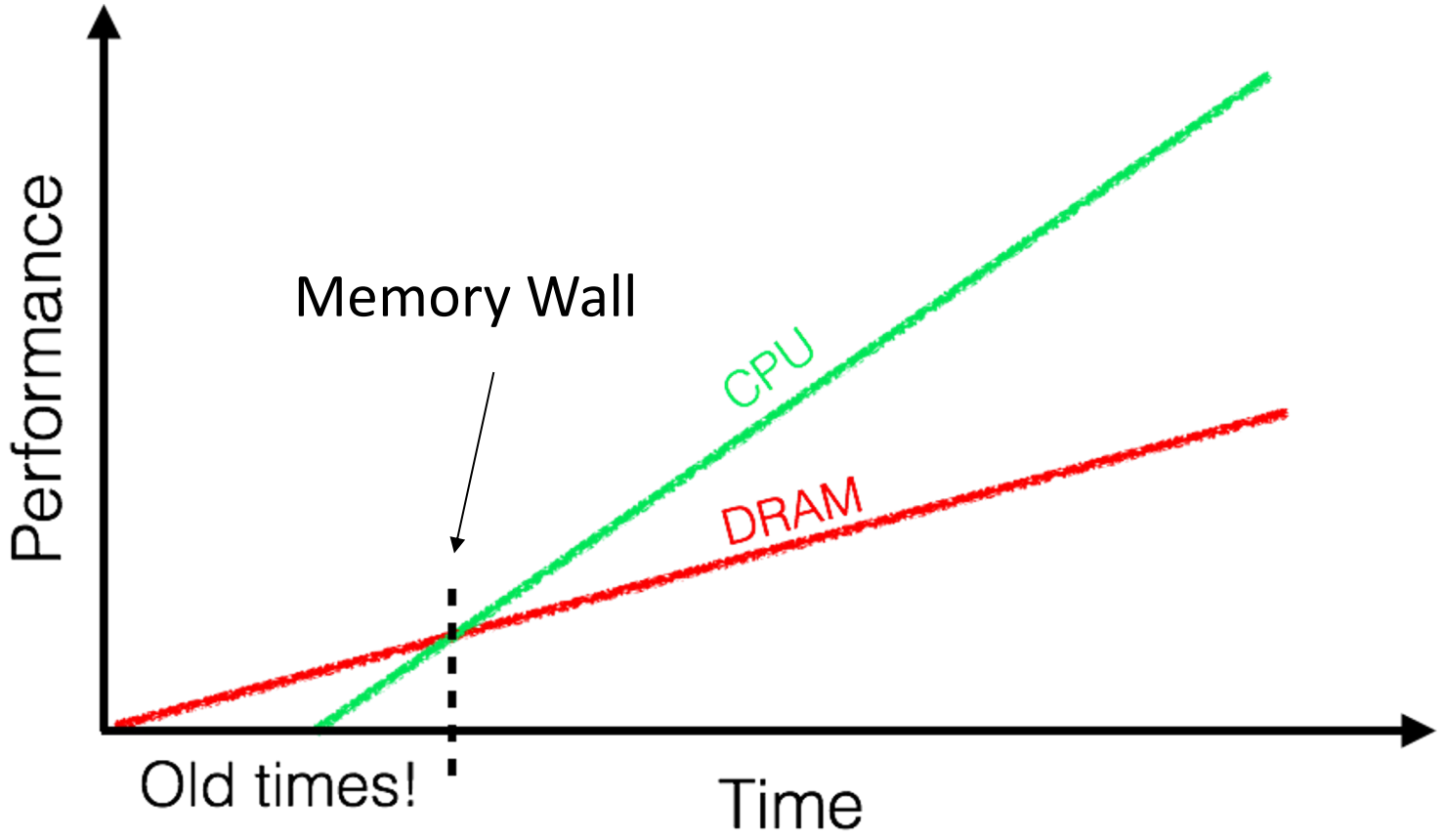
*for (ii): is memory faster than CPU (to deliver data in time)?*

*does it have enough capacity?*





## Which one is faster?

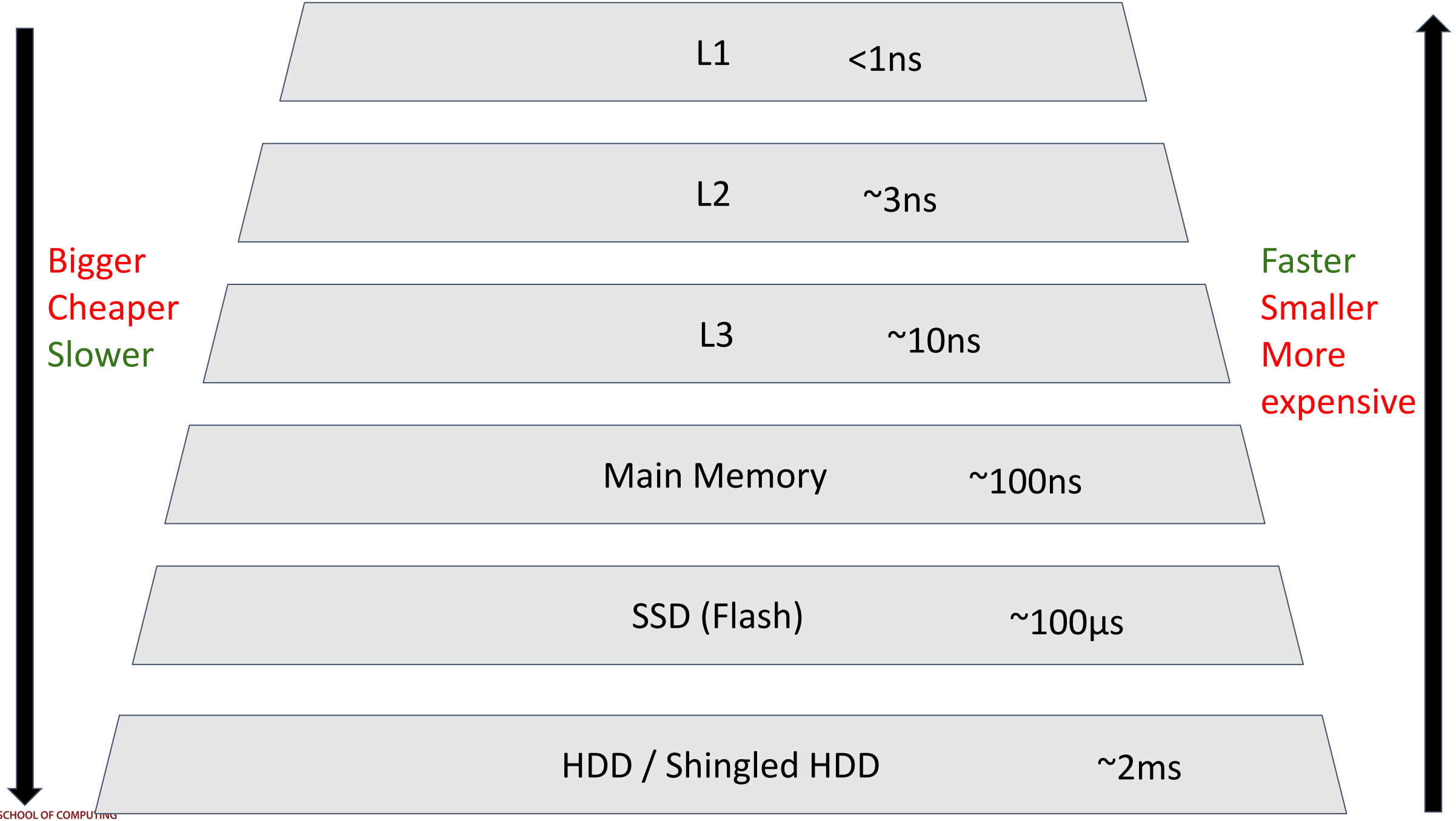



As the gap grows, we need a *deep* ***memory hierarchy***

A single level of main memory is not enough

We need a ***memory hierarchy***

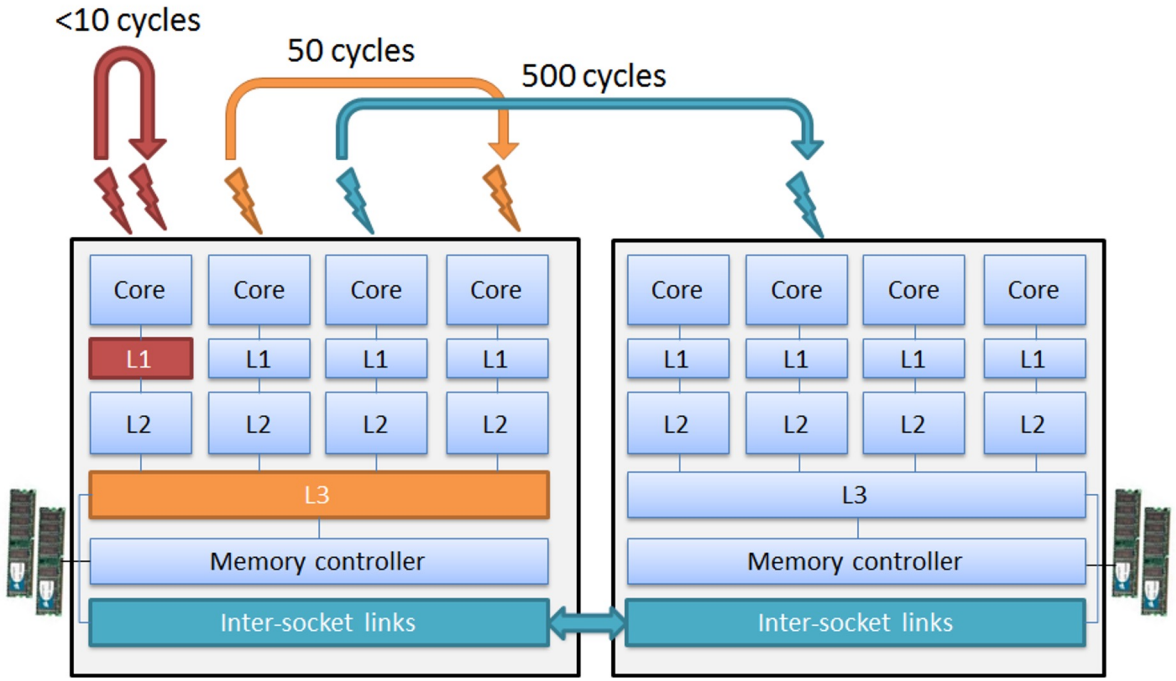
# What is the memory hierarchy ?

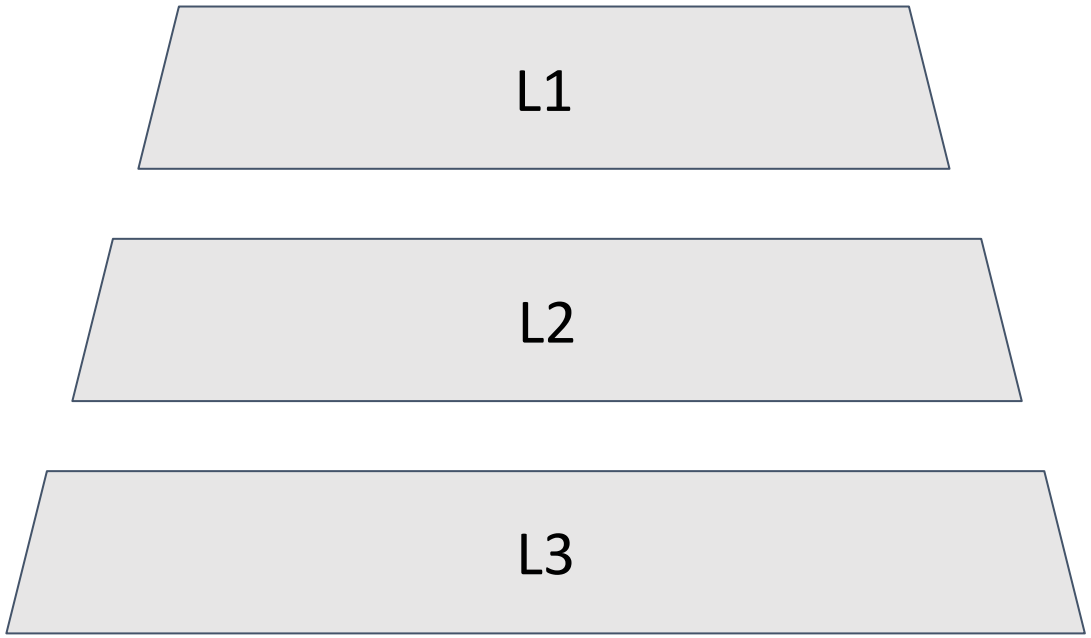

L1
<1ns
L2
~3ns
L3
~10ns
Main Memory
~100ns
SSD (Flash)
~100µs
HDD / Shingled HDD
~2ms
Bigger
Cheaper
Slower
Faster
Smaller
More
expensive

# Cache Hierarchy

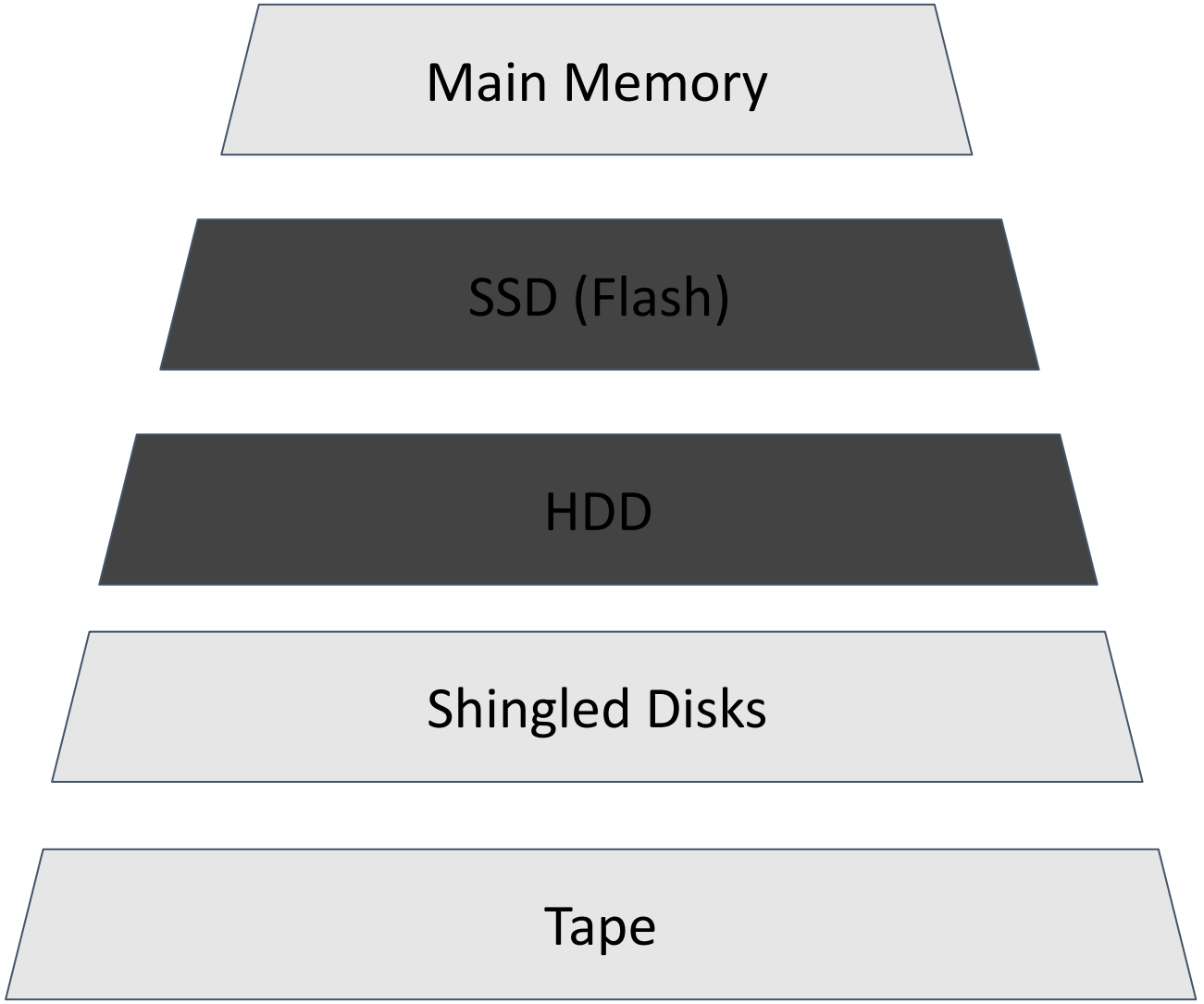
What is a core?

What is a socket?



# Storage Hierarchy



# Hard Disk Drives

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

Data organized on pages/blocks (across tracks)

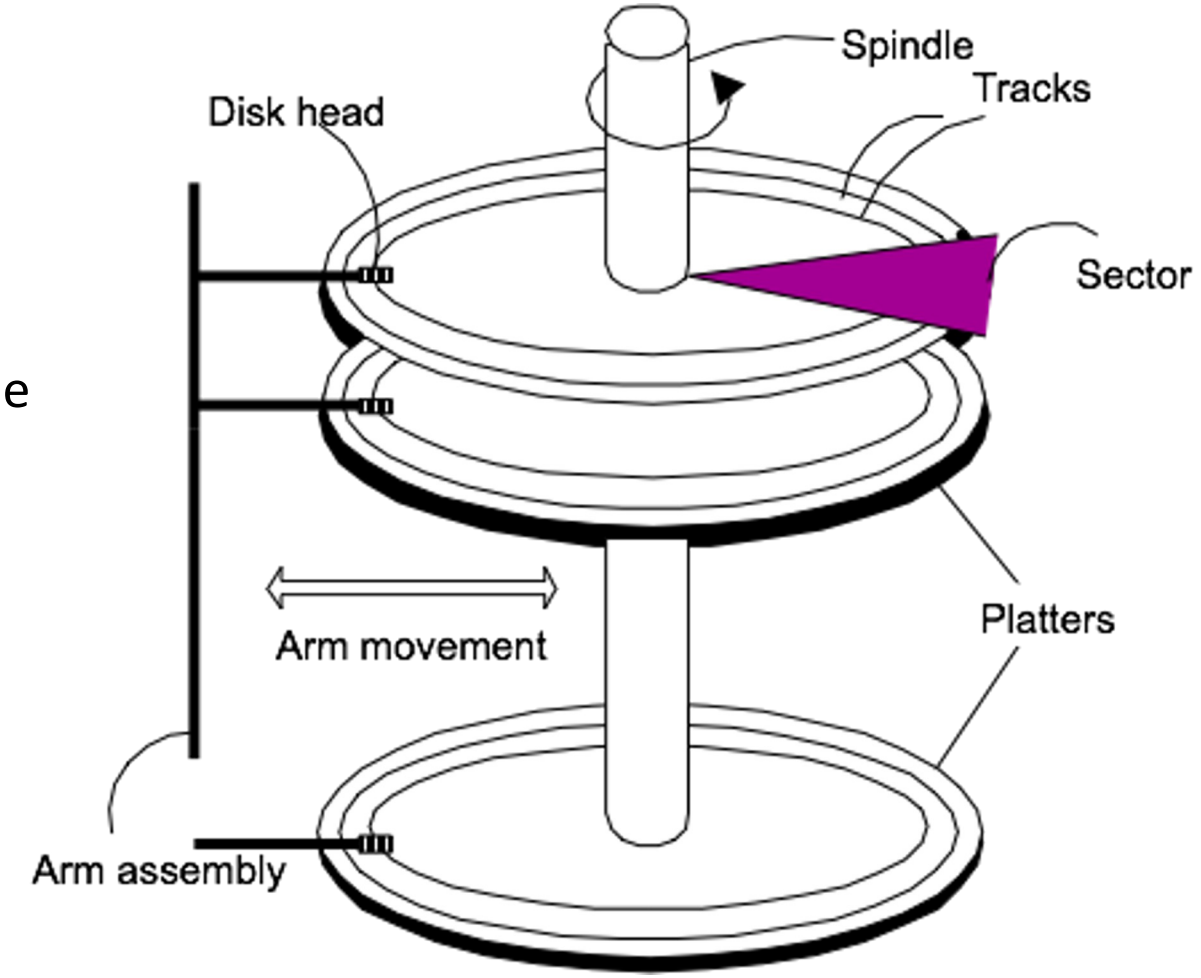
Multiple *tracks* create an (imaginary) *cylinder*

Disk access time:

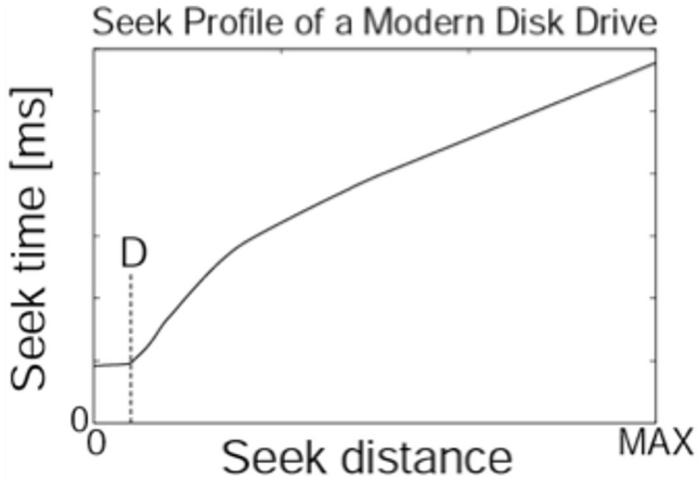
seek latency + rotational delay + transfer time
(0.5-2ms) + (0.5-3ms) + <0.1ms/4KB

Sequential >> random access (~10x)

**Goal:** *avoid random access*



# Seek time + Rotational delay + Transfer time




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

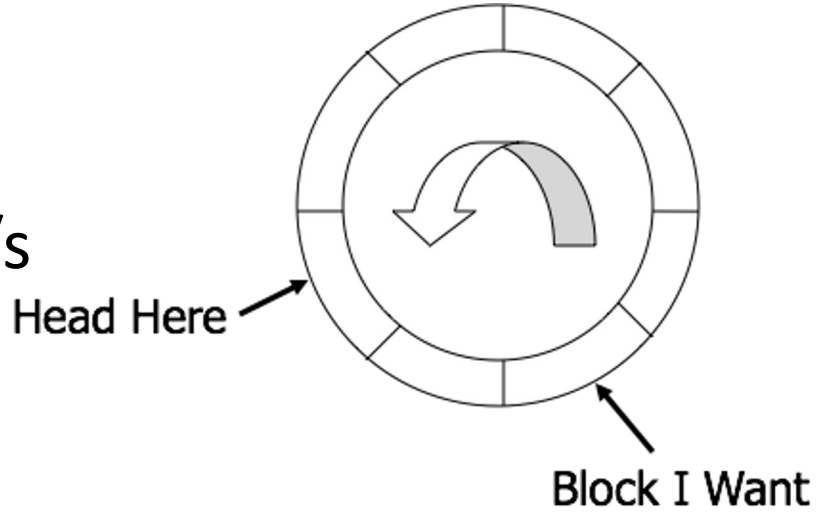
Short seeks are dominated by "settle" time
(D is on the order of hundreds or more)

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

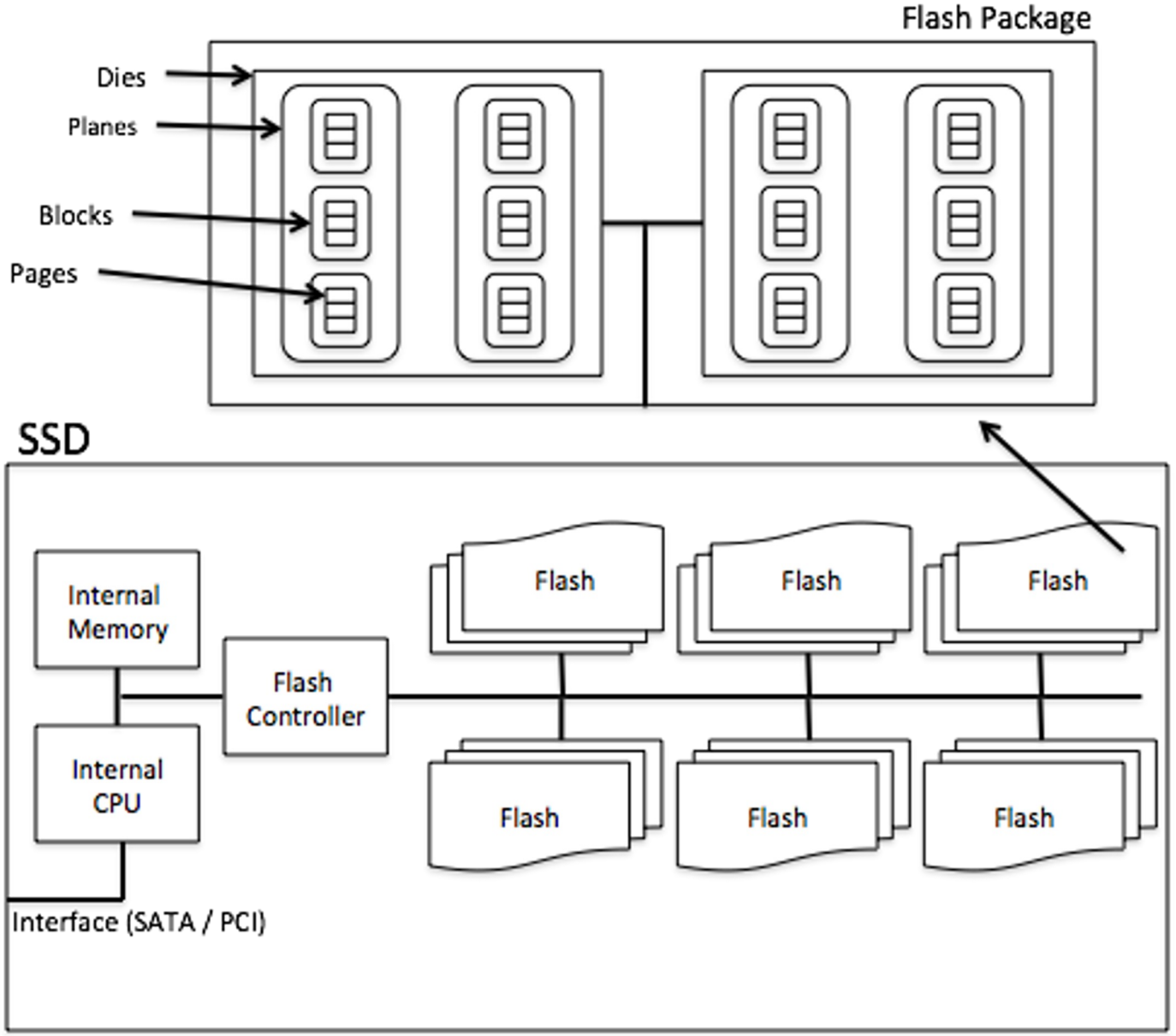
Data organized on pages (similar to disks) which are further grouped to erase blocks

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:

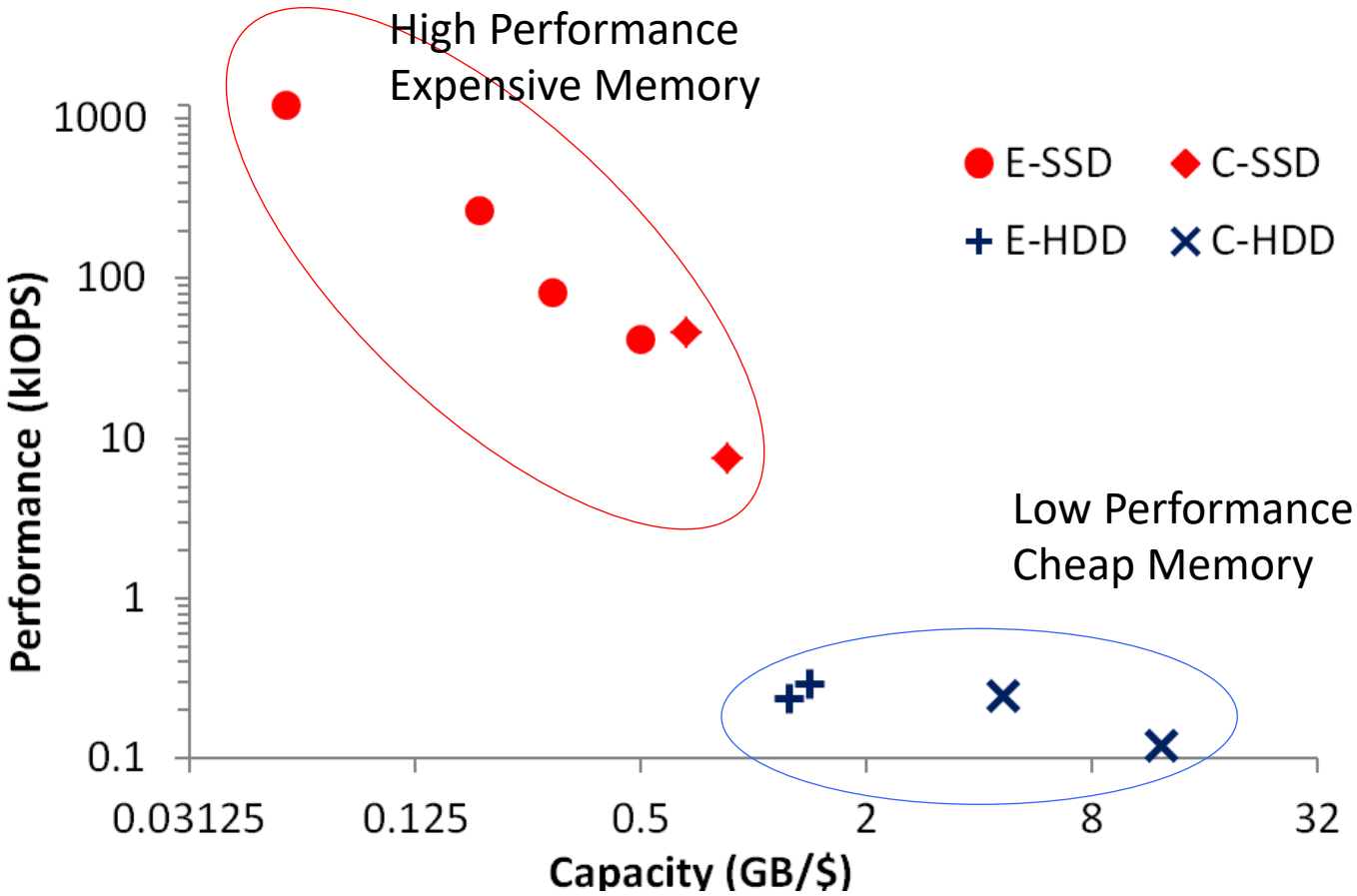
device organization (***internal parallelism***)

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

# 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?