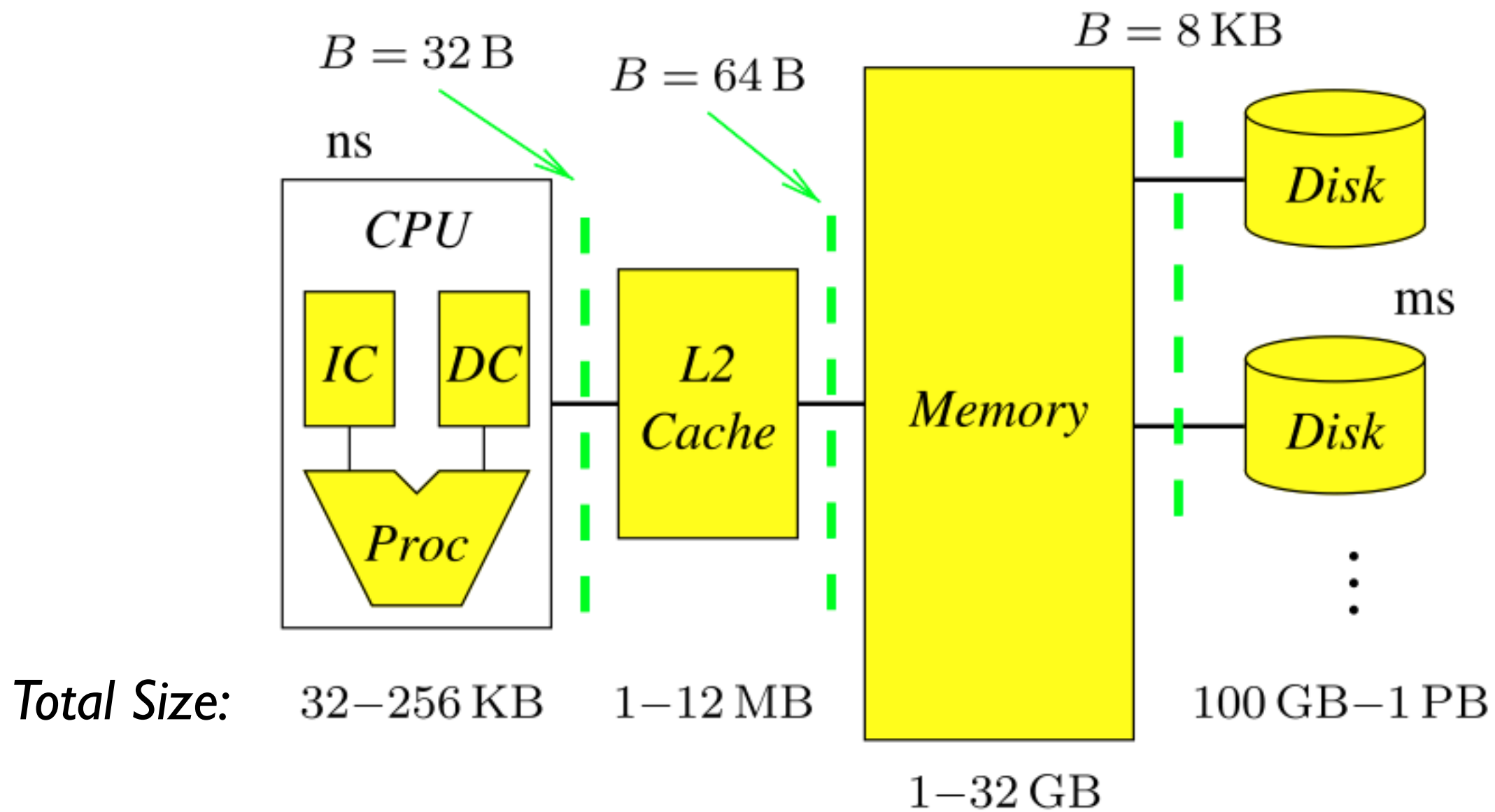


Streaming & Chunking

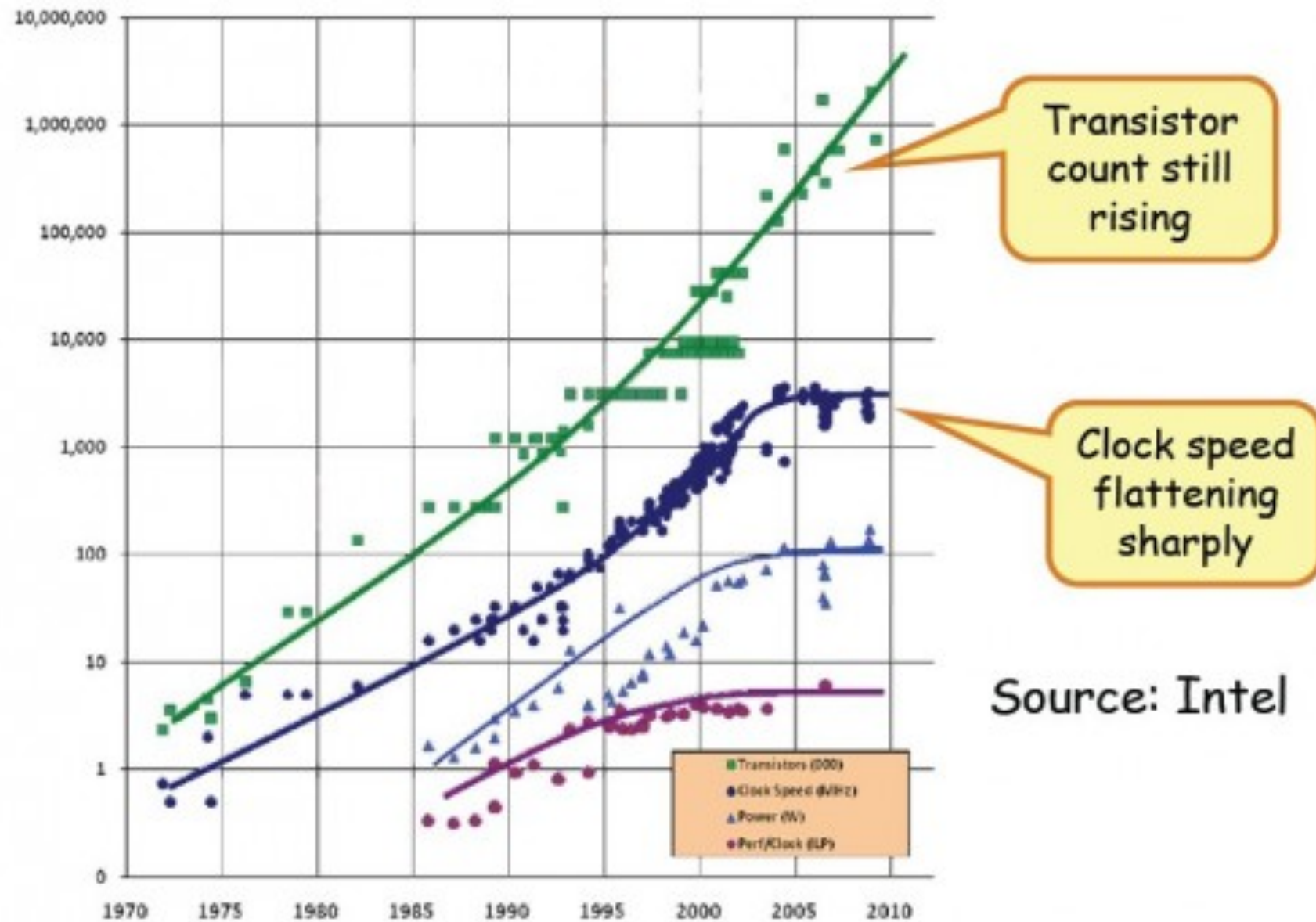
Big Data Analytics

The Memory Hierarchy

$B = \text{Block size}$

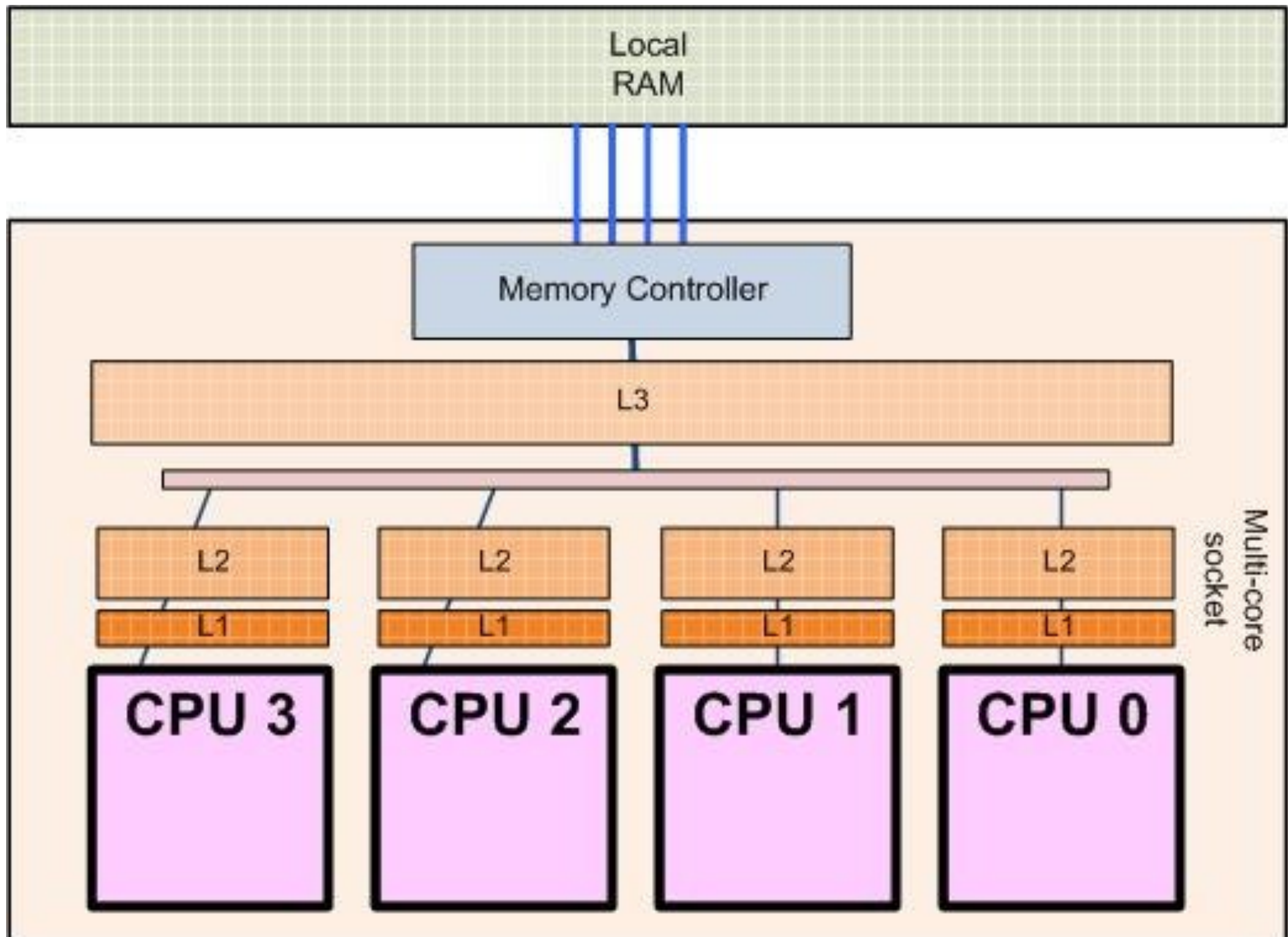


Moore's law



The only way to improve performance
is by going parallel

Multi-core computers



Data-centers

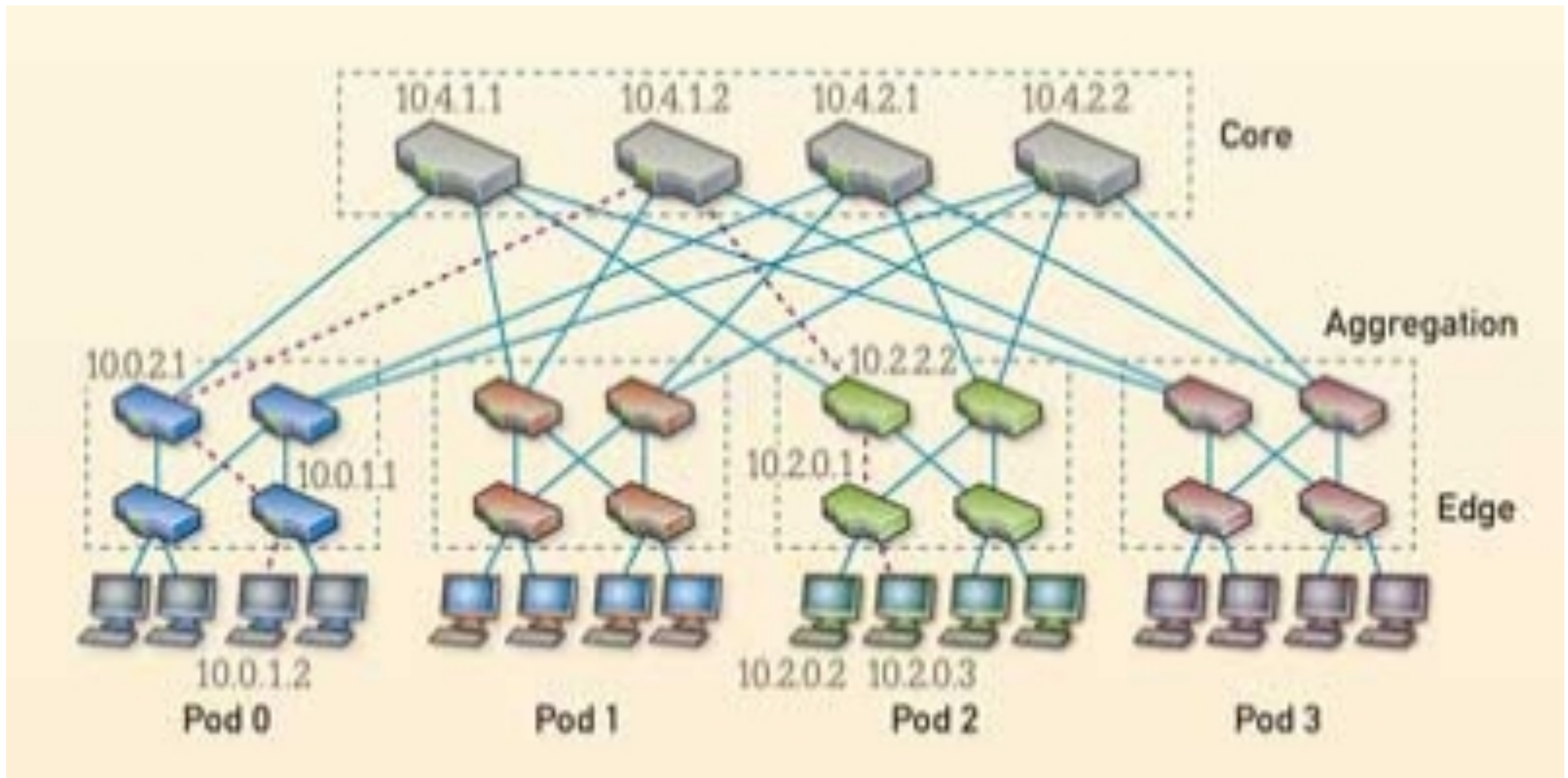


Google

google.com/datacenters

Data-Center networks

Taken from the academic paper authored by UCSD's Mohammad Al-Fares, Alexander Loukissas and Amin Vahdat, this illustration of a simple fat-tree architecture shows the path (dotted line) that packets would follow from one server to another. Source: A Scalable, Commodity Data Center Network Architecture.



Up to date Characteristics

	Cost	Power	Block Size	Bandwidth
L1-L2	On-Chip with CPU	low-end: iTouch: 1Watt high-end: Intel Core i7-950: 130Watt	L1:32-64 Bytes L2: 64-256 Bytes	100s of GB/ sec
DRAM	8\$-16\$/GB	1-2 Watts/GB	max throughput at: 8-16KB	50-70 GB/sec
SSD	High end, \$11/GB. Low end: \$1-4/GB About \$50,000 for I/O	high end: 0.15W/GB Low End: 0.05 W/GB	4KB	high end: 1-3GB/sec low end: .1-.2GB/sec
Disk	0.03-0.1\$/GB	0.01W/GB	4KB	100MB/sec sequential 0.4-2.0MB/sec random Getting to each block takes 2-10ms

The disk drive

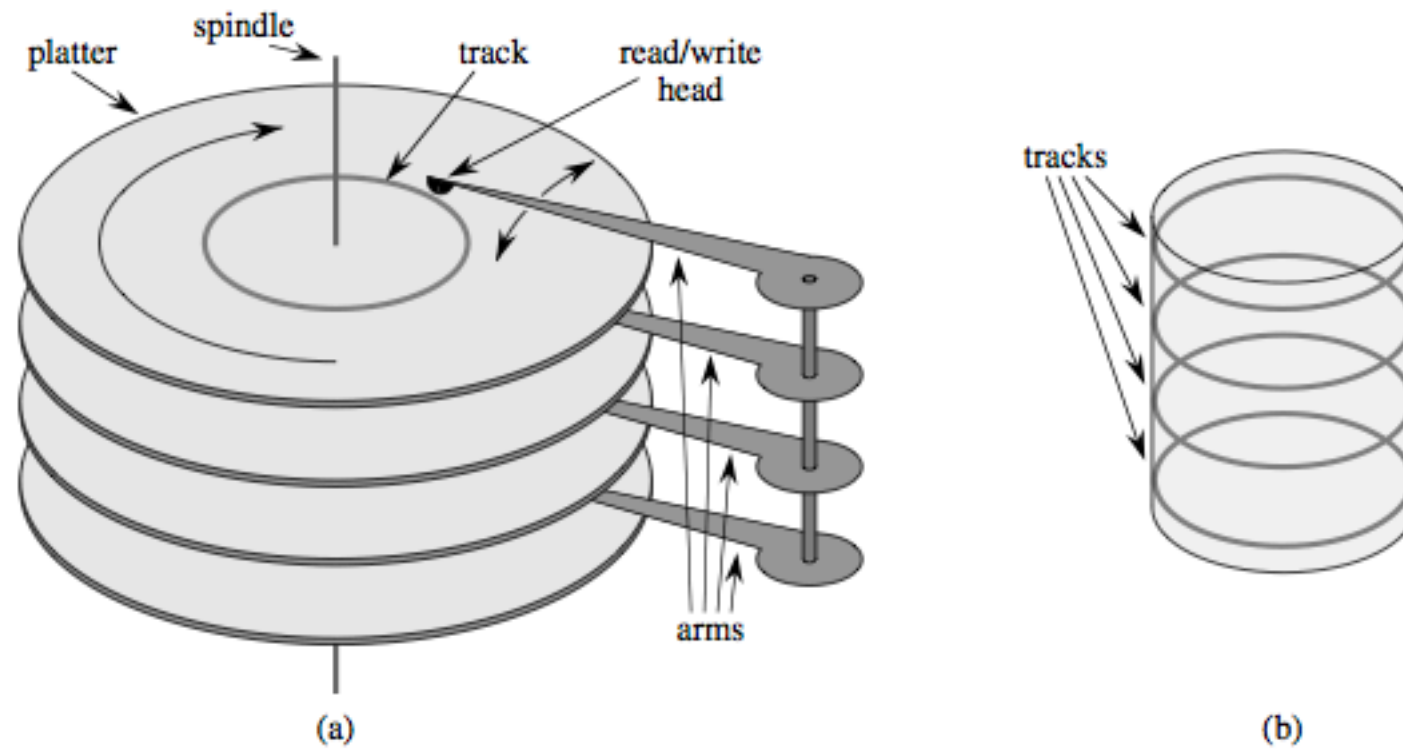


Fig. 2.1 Magnetic disk drive: (a) Data are stored on magnetized platters that rotate at a constant speed. Each platter surface is accessed by an arm that contains a read/write head, and data are stored on the platter in concentric circles called tracks. (b) The arms are physically connected so that they move in unison. The tracks (one per platter) that are addressable when the arms are in a fixed position are collectively referred to as a cylinder.

Parallel Disk model of computation

2.1 I/O MODEL AND PROGRAM PARAMETERS 19

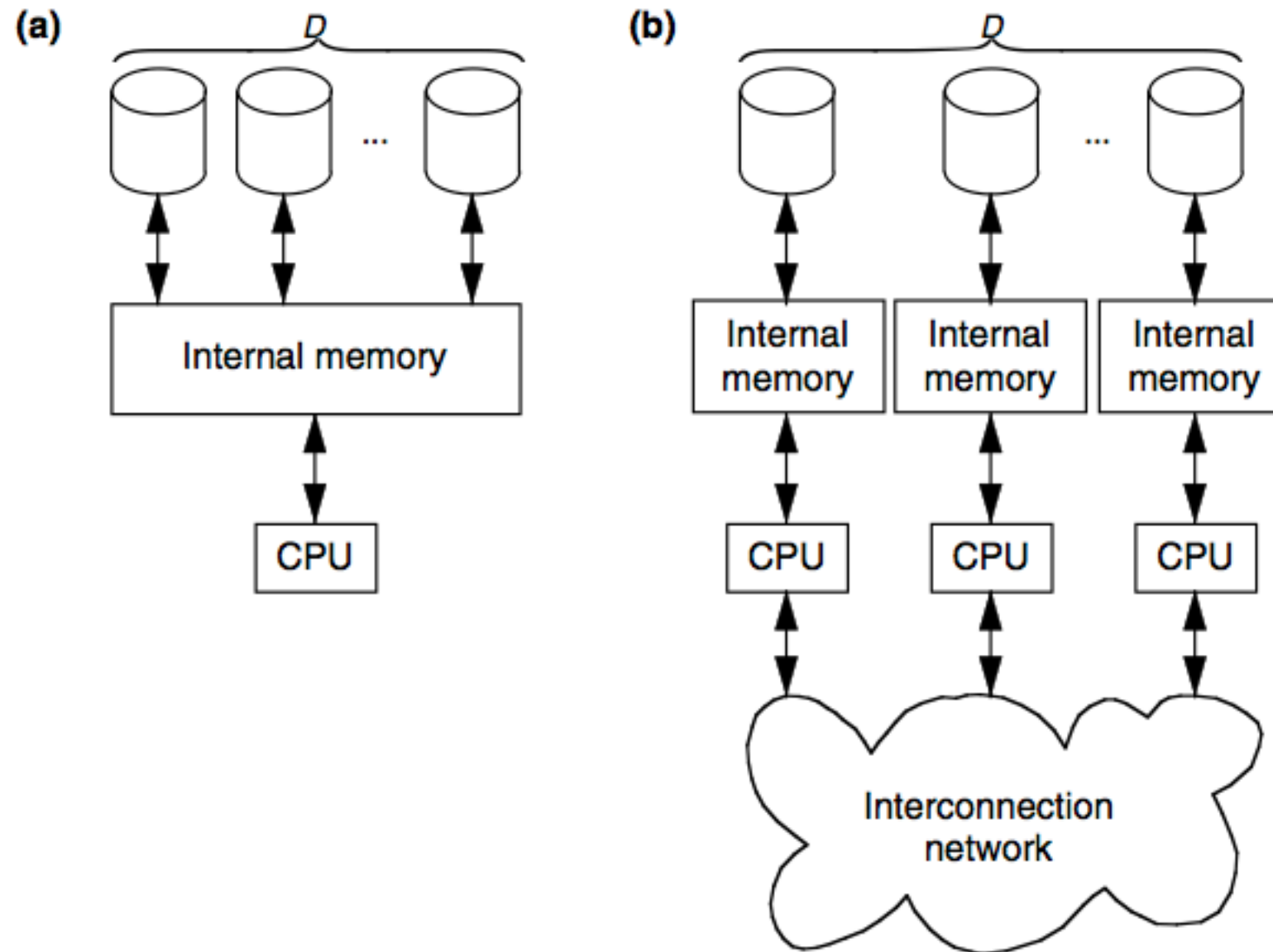


Fig. 2.2 Parallel disk model: (a) $P = 1$, in which the D disks are connected to a common CPU; (b) $P = D$, in which each of the D disks is connected to a separate processor.

Paging

- Cache fast & small. Memory slow & large.
- Complexity is hidden from software.
- Each level is divided into smaller **pages**.
- Software asks to read address X
 - Hardware checks if X is in L1 Cache.
 - If yes – use it
 - If not – cache miss, check L2 cache
 - Repeat for L2, L3, RAM, Local Disk, Networked disk...

Dealing with a Cache Miss

- Find page to remove (LRU, LRW,...)
- If page has been written to (dirty) then write page content back to slower memory before releasing it.
- Fetch requested page from slower memory.

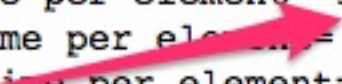
The price of caching

- Each level of the hierarchy is much slower than the one above it. The CPU is the fastest.
- A Cache miss causes a significant time delay.
- During the time the page is fetched, the CPU can, in principle, do something else. However, if the memory hierarchy is the bottleneck, the CPU is typically idle.
- Caching works when there is **data locality**: if prog. read/writes one byte in a page, it is likely that it will read/write many.

Sequential memory access

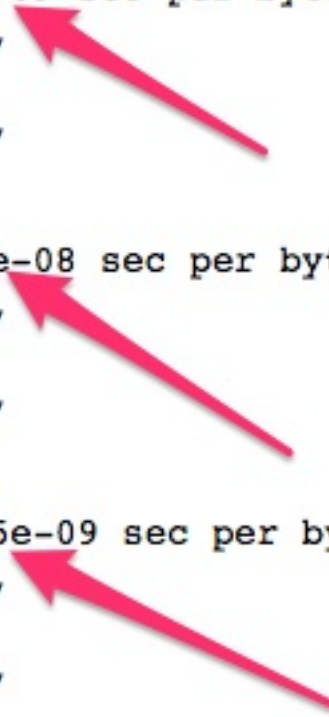
Consecutive Memory writes

```
array of length 1000 repeated 1 times. total size= 0.001 MB, Time per element= 1.5974e-08
array of length 1000 repeated 3 times. total size= 0.003 MB, Time per element= 1.13646e-08
array of length 1000 repeated 5 times. total size= 0.005 MB, Time per element= 1.02043e-08
array of length 1000 repeated 7 times. total size= 0.007 MB, Time per element= 9.70704e-09
array of length 1000 repeated 10 times. total size= 0.010 MB, Time per element= 1.06096e-08
array of length 1000 repeated 100 times. total size= 0.100 MB, Time per element= 5.76019e-09
array of length 1000 repeated 1000 times. total size= 1.000 MB, Time per element= 5.40495e-09
array of length 1000 repeated 10000 times. total size=10.000 MB, Time per element= 5.98199e-09
array of length 1000 repeated 100000 times. total size=100.000 MB, Time per element= 8.09567e-09
array of length 1000 repeated 1000000 times. total size=1000.000 MB, Time per element= 9.4426e-09
```



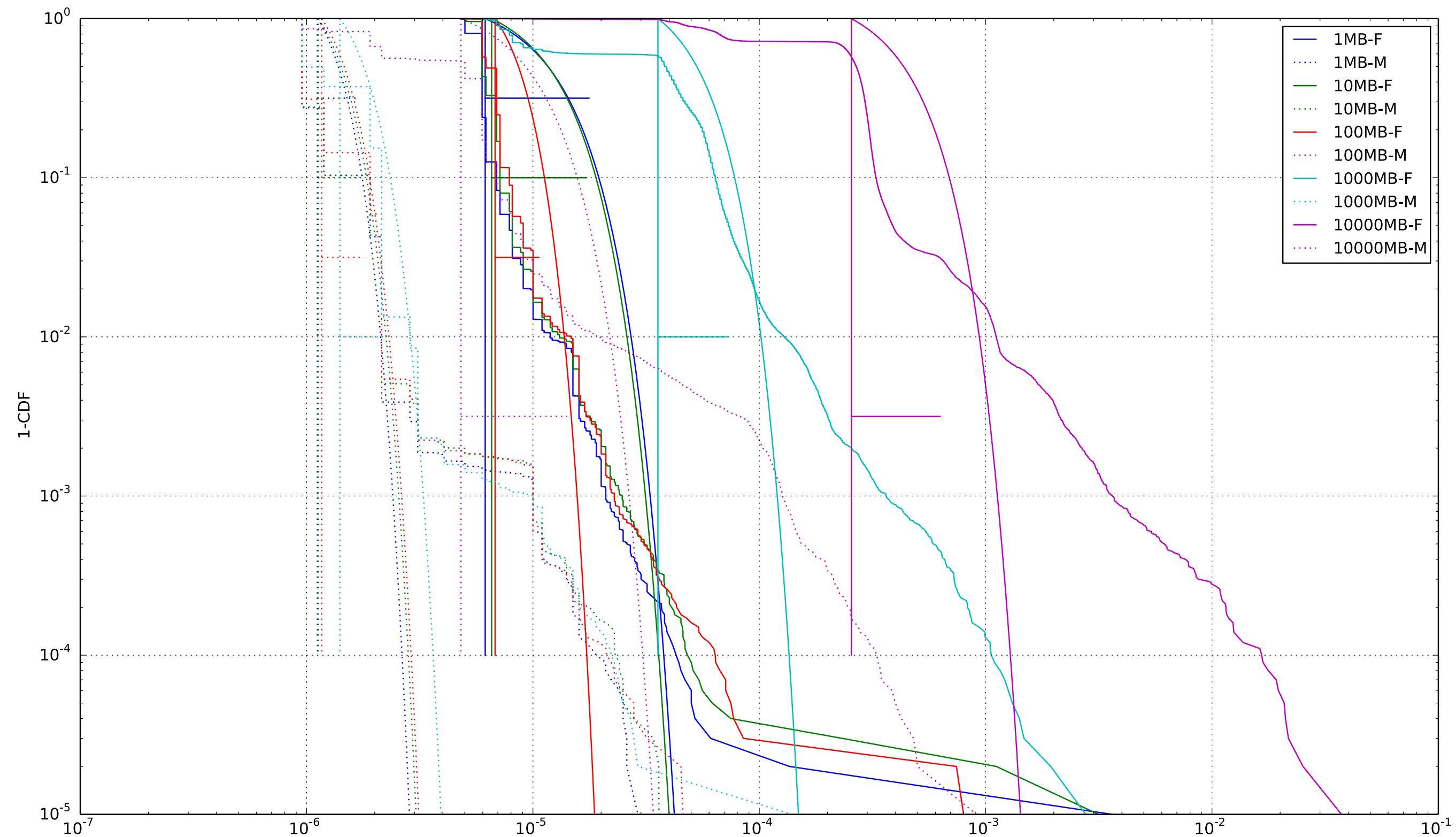
Sequential disk writes

```
creating 1000000 byte block: 0.000472, writing 1 blocks 0.003439, 3.43919e-09 sec per byte
0 , 10000 , 20000 , 30000 , 40000 , 50000 , 60000 , 70000 , 80000 , 90000 ,
File pokes mean=6.18668794632e-06, file std=1.0410197823e-05
0 , 10000 , 20000 , 30000 , 40000 , 50000 , 60000 , 70000 , 80000 , 90000 ,
Memory pokes mean=1.112408638e-06, file std=5.80482263226e-07
0 ,
creating 1000000 byte block: 0.000036, writing 10 blocks 0.153996, 1.53996e-08 sec per byte
0 , 10000 , 20000 , 30000 , 40000 , 50000 , 60000 , 70000 , 80000 , 90000 ,
File pokes mean=6.42552375793e-06, file std=1.04976719992e-05
0 , 10000 , 20000 , 30000 , 40000 , 50000 , 60000 , 70000 , 80000 , 90000 ,
Memory pokes mean=1.11732244492e-06, file std=5.33633764697e-07
0 ,
creating 1000000 byte block: 0.000037, writing 100 blocks 0.318675, 3.18675e-09 sec per byte
0 , 10000 , 20000 , 30000 , 40000 , 50000 , 60000 , 70000 , 80000 , 90000 ,
File pokes mean=6.78221702576e-06, file std=9.65892486086e-06
0 , 10000 , 20000 , 30000 , 40000 , 50000 , 60000 , 70000 , 80000 , 90000 ,
Memory pokes mean=1.12557649612e-06, file std=5.04215251283e-07
0 , 100 , 200 , 300 , 400 , 500 , 600 , 700 , 800 , 900 ,
```

Three red arrows point to the 'sec per byte' values in the benchmark output. The first arrow points to '3.43919e-09' in the first test. The second arrow points to '1.53996e-08' in the second test. The third arrow points to '3.18675e-09' in the third test.

Random Access

Produced using iPython Notebook “Measuring performance of memory hierarchy.ipynb”
running on Apple Powerbook with 16GB memory



Lets look at the measurements using some notebooks

- Runnable notebook available through github.
- We will look at “frozen” notebooks (nbviewer over github)
- Go to [class Web Page](#)
- For Thursday:
 - Install Ipython
 - Clone repository
 - Run notebooks for first class
 - Optional: measure the performance of your machine
 - Do optional work on mean+std page