

# **CS 7172**

# **Parallel and Distributed Computation**

## **Overview**

**Kun Suo**

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

# Outline

---

- Why we need ever-increasing performance.
- Why we're building parallel systems.
- Why we need to write parallel programs.
- How do we write parallel programs?
- What we'll be doing.
- Concurrent, parallel, distributed!



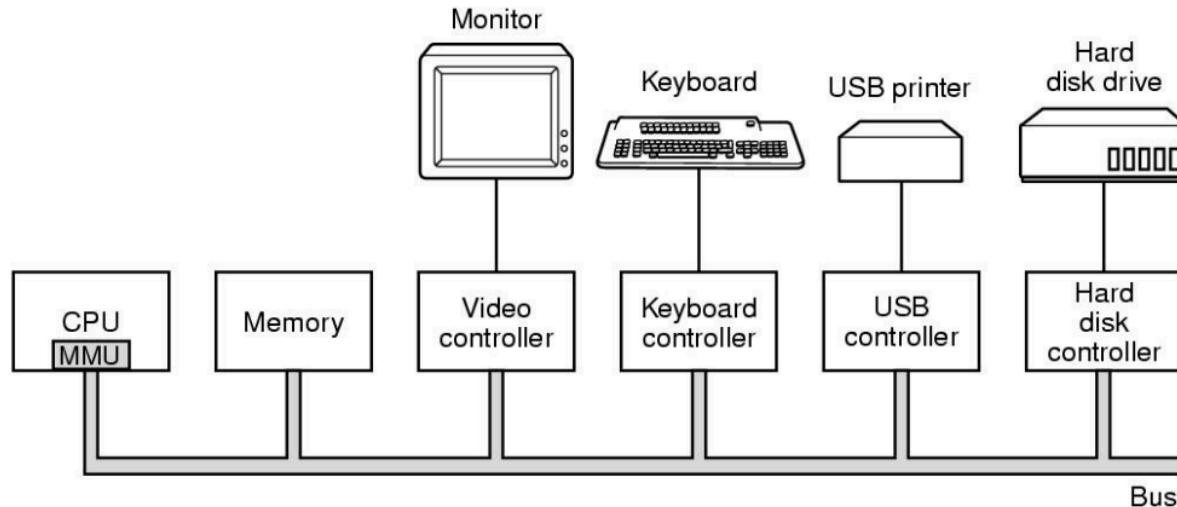
# Computer Hardware Review

---



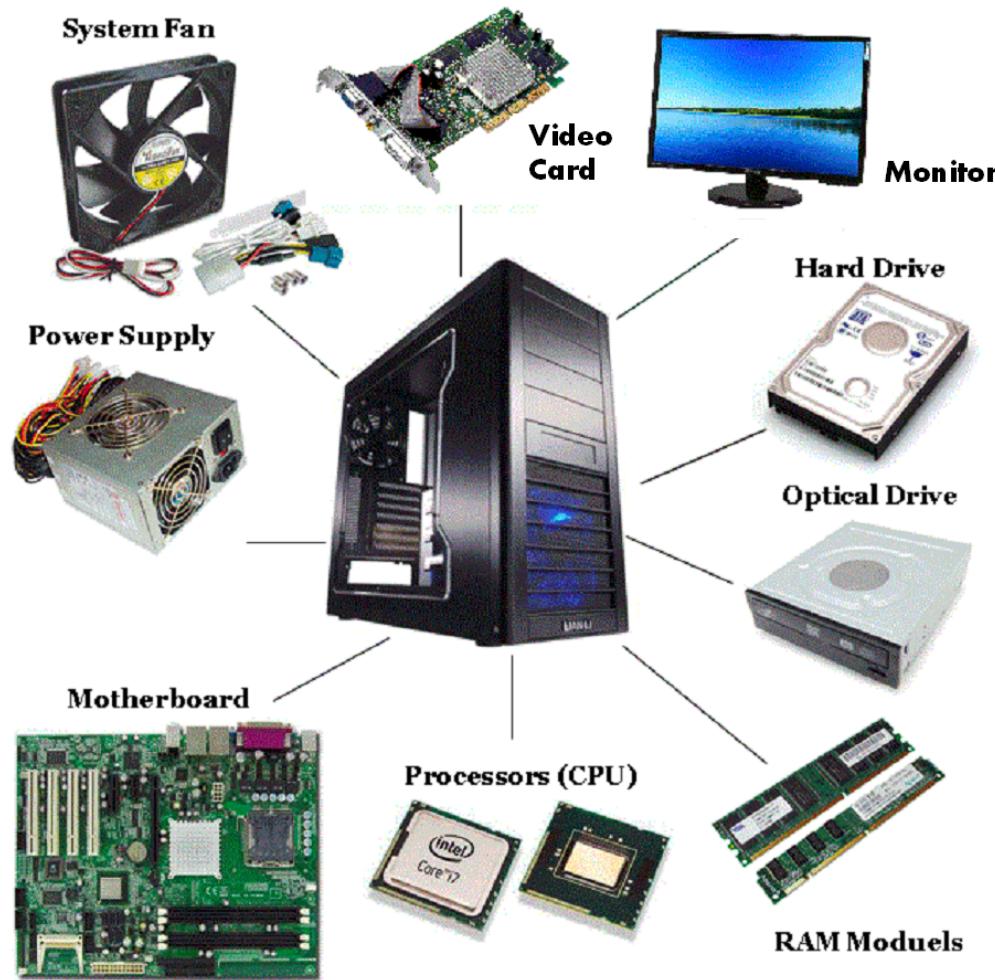
# Computer Hardware Review

- Basic components of a simple personal computer

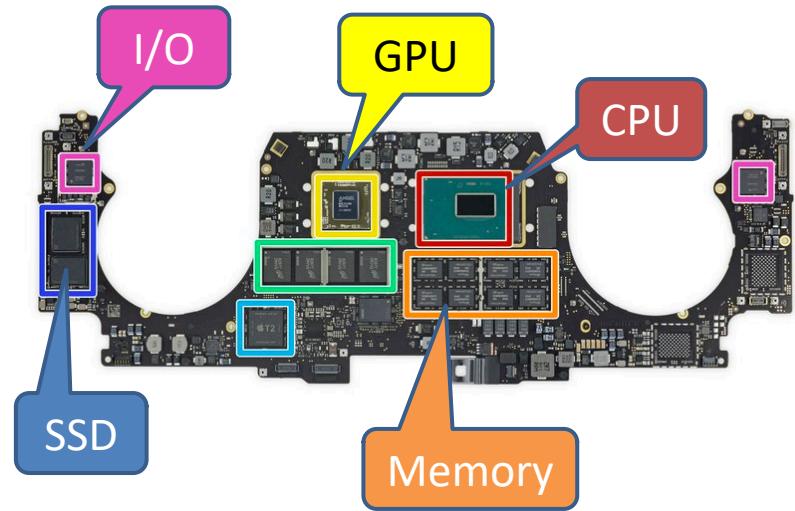


- **CPU**: data processing
- **Memory**: volatile data storage
- **Disk**: persistent data storage
- **NIC**: inter-machine communication
- **Bus**: intra-machine communication

# Computer Hardware Review

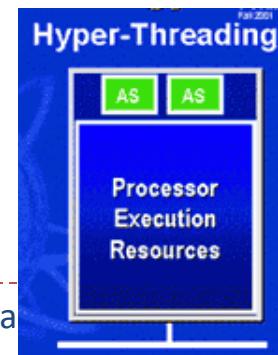
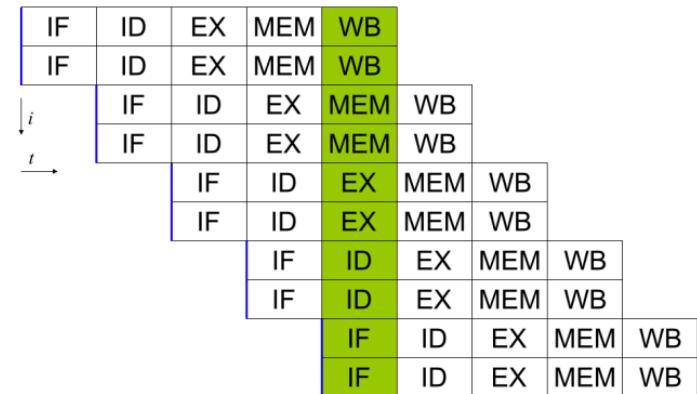


# Computer Hardware Review



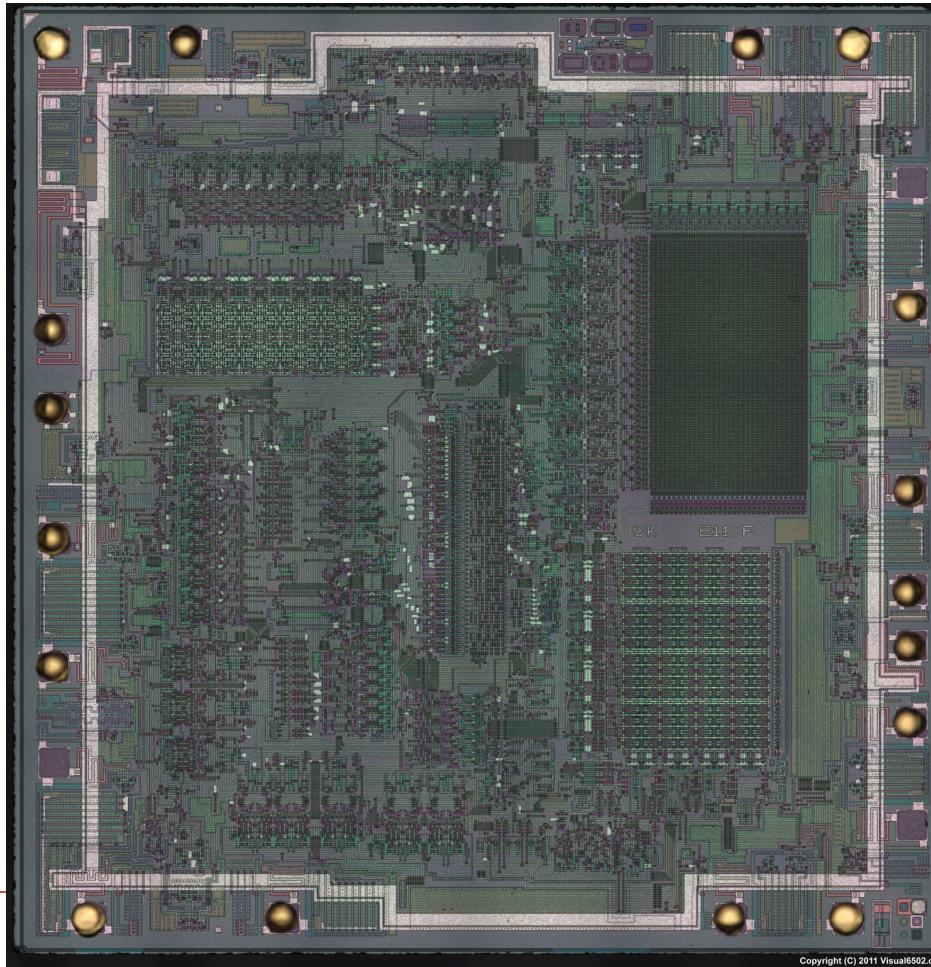
# Central Processing Unit (CPU)

- Components
  - Arithmetic Logic Unit (ALU) -> Compute and data
  - Control Unit (CU) -> control device and system
- Clock rate
  - The speed at which a CPU is running
- Data storage
  - General-purpose registers: EAX, EBX ...
  - Special-purpose registers: PC (program counter), SP (stack), IR (instruction register) ...
- Parallelism
  - Instruction-level parallelism
  - Thread-level parallelism
    - ▶ Hyper-threading: duplicate units that store architectural states
    - ▶ Replicated: registers. Partitioned: ROB, load buffer... Shared: reservation station, caches



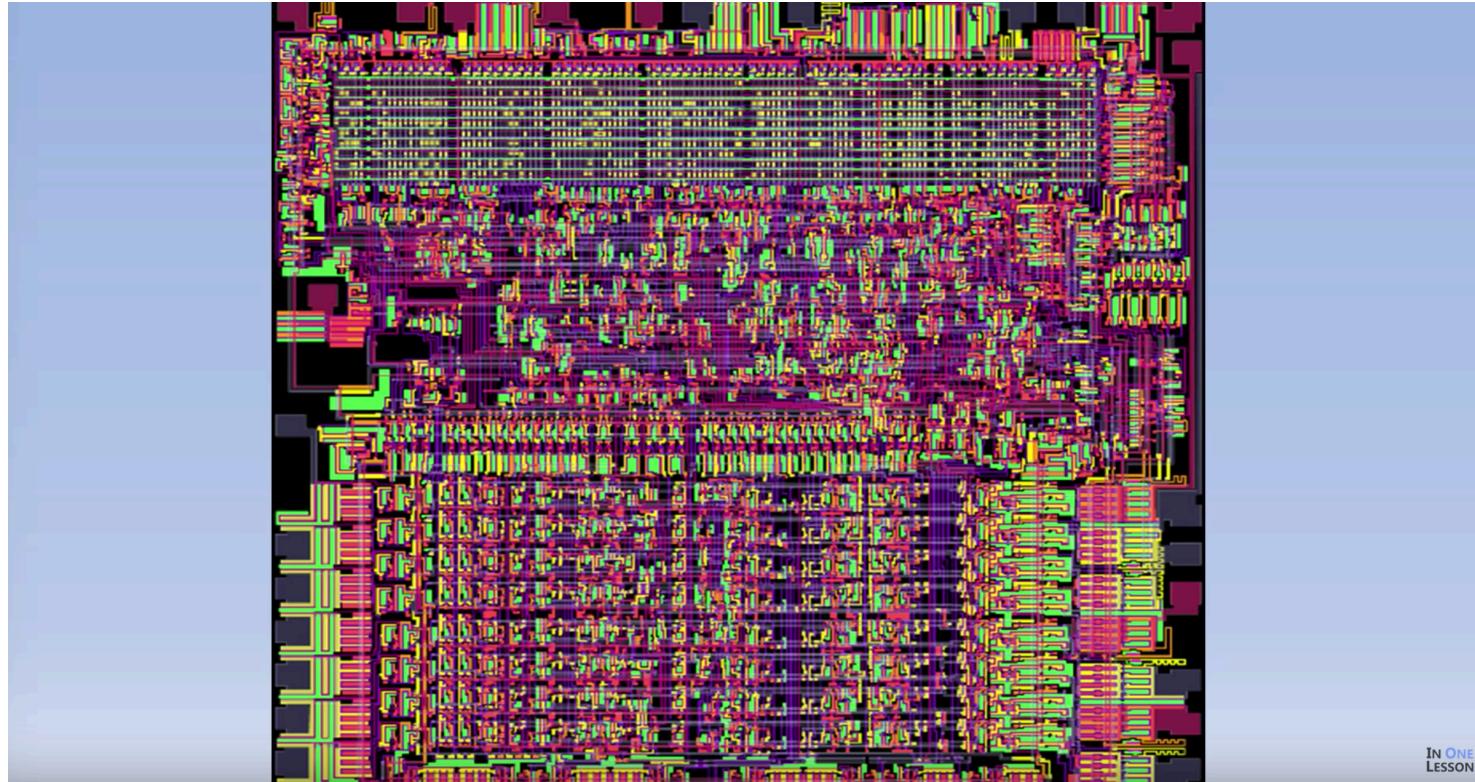
# What's inside of CPU?

- <http://www.visual6502.org/>



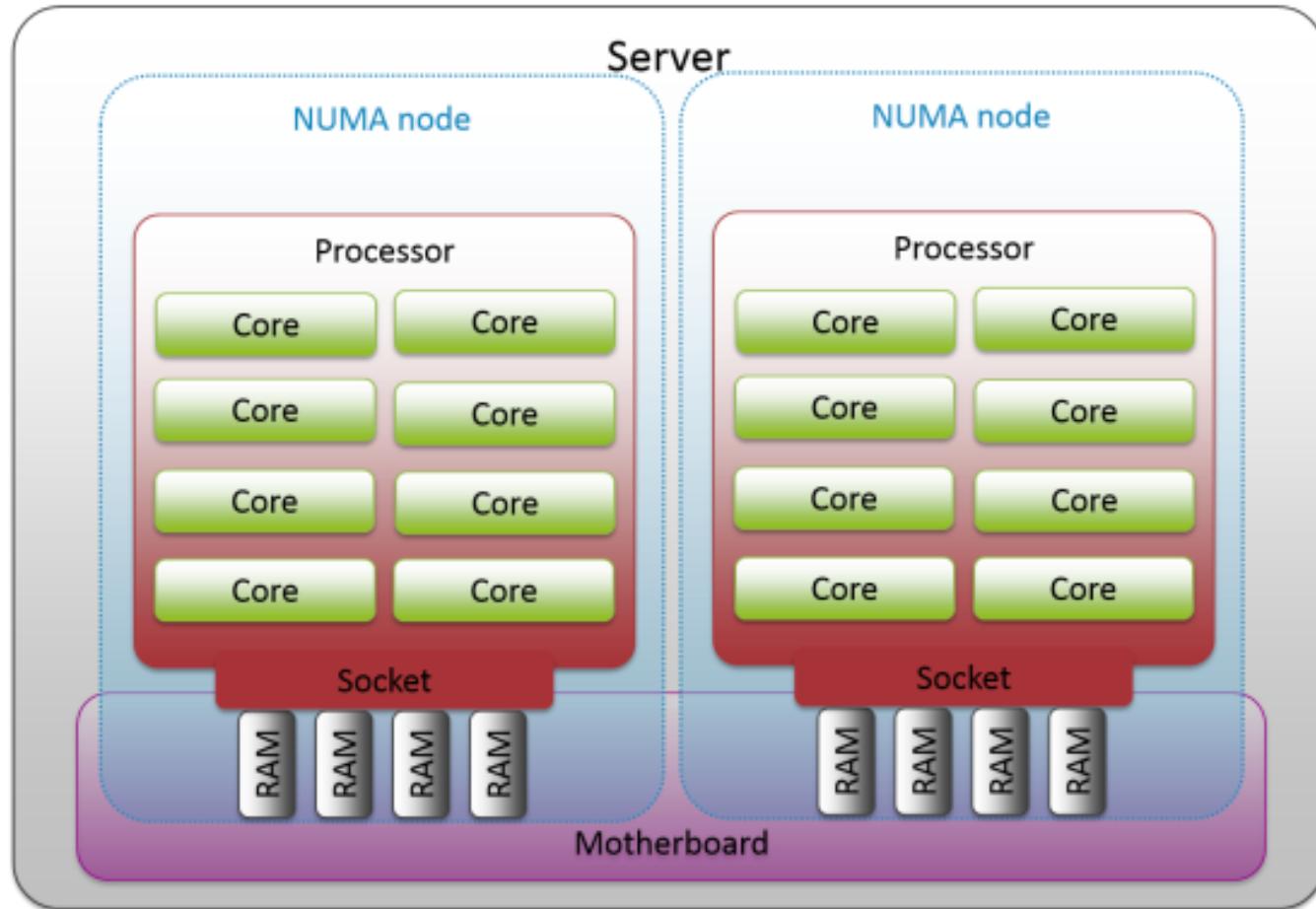
# How CPU works?

- [https://youtu.be/cNN\\_tTXABUA?t=494](https://youtu.be/cNN_tTXABUA?t=494)



IN ONE  
LESSON

# NUMA node vs Socket vs Core relationship



# CPU information

- lscpu



```
administrator@ubuntuvm-1604 ~> lscpu
Architecture:          x86_64
CPU op-mode(s):       32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:  0,1
Thread(s) per core:   1
Core(s) per socket:   1
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
Stepping:               1
CPU MHz:                2199.998
BogoMIPS:              4399.99
Hypervisor vendor:    VMware
Virtualization type:  full
L1d cache:              32K
L1i cache:              32K
L2 cache:                256K
L3 cache:                51200K
NUMA node0 CPU(s):     0,1
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep m
all nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology ts
id sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave av
lt invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1
flush_l1d arch_capabilities
```

# CPU information

- lscpu

```
pi@raspberrypi:~ $ lscpu
Architecture:          armv7l
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:  0-3
Thread(s) per core:   1
Core(s) per socket:   4
Socket(s):             1
Vendor ID:             ARM
Model:                 3
Model name:            Cortex-A72
Stepping:              r0p3
CPU max MHz:           1500.0000
CPU min MHz:           600.0000
BogoMIPS:              108.00
Flags:                 half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
```



# CPU information

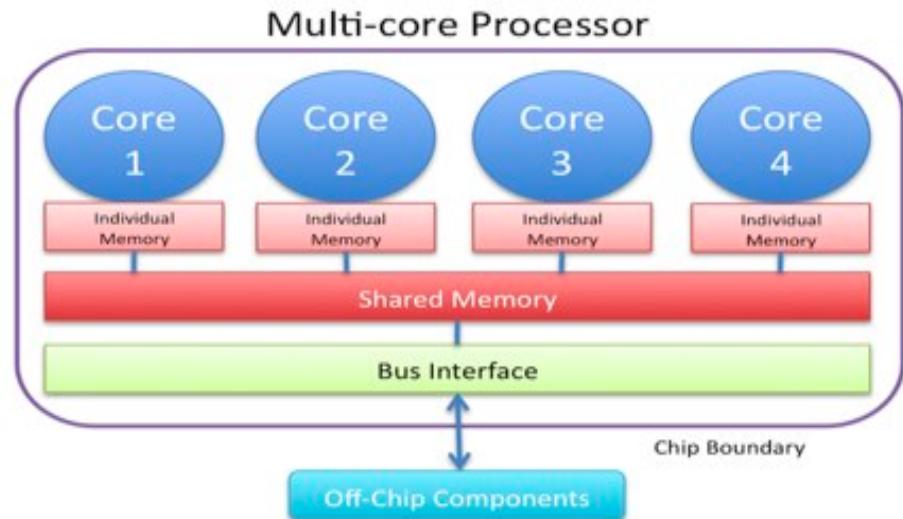
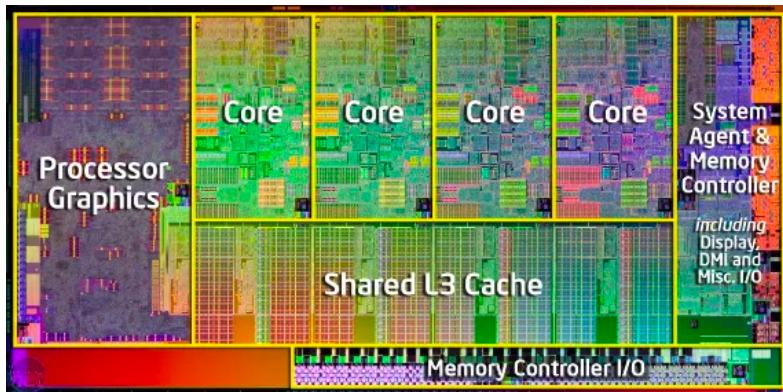
- \$ cat /proc/cpuinfo

```
administrator@ubuntuvm-1604 ~> cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
stepping        : 1
microcode      : 0xb000036
cpu MHz        : 2199.998
cache size     : 51200 KB
physical id    : 0
siblings        : 1
core id         : 0
cpu cores      : 1
apicid          : 0
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level    : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
x pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology t
e4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave a
vpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1
h_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_stor
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 42 bits physical, 48 bits virtual
power management:
```

```
processor       : 1
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
stepping        : 1
microcode      : 0xb000036
cpu MHz        : 2199.998
cache size     : 51200 KB
physical id    : 2
siblings        : 1
core id         : 0
cpu cores      : 1
apicid          : 2
initial apicid : 2
fpu             : yes
fpu_exception   : yes
cpuid level    : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
x pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology t
e4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave a
vpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1
h_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_stor
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 42 bits physical, 48 bits virtual
power management:
```

# Multi-Core Processors (SMP)

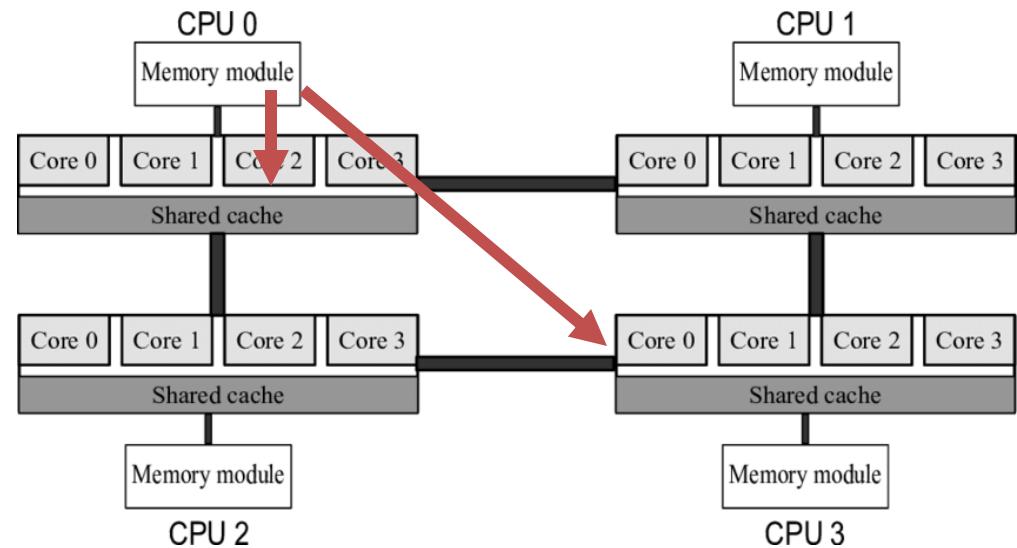
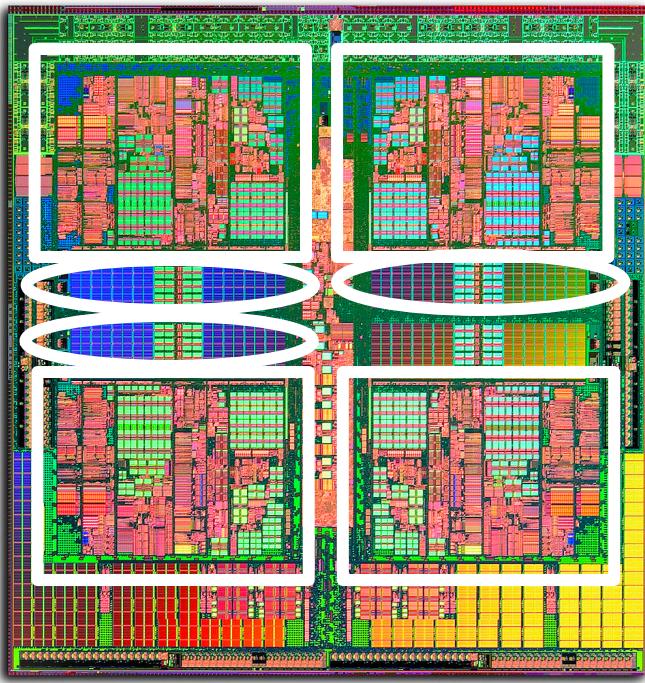
- Multiple CPUs on a single chip



Symmetric multiprocessing (**SMP**)

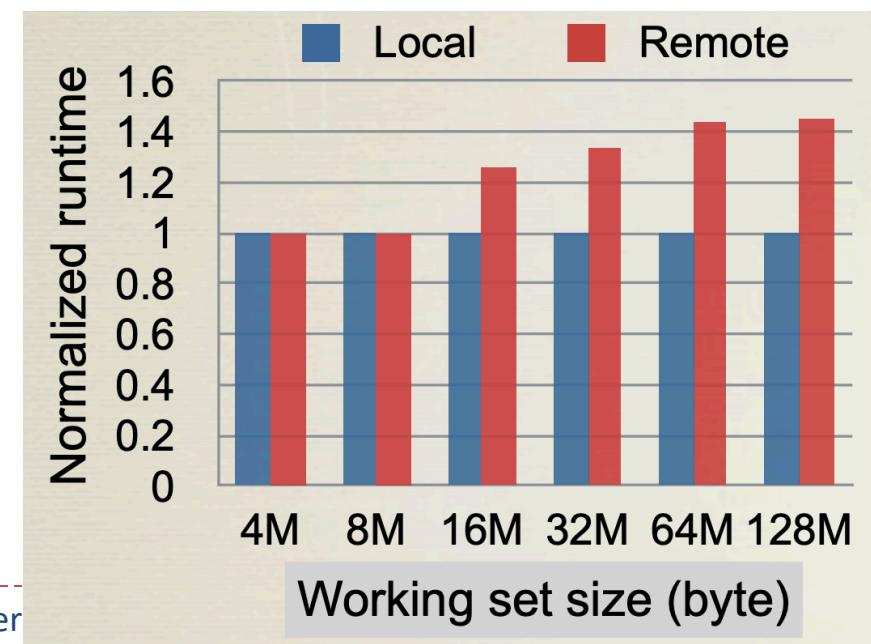
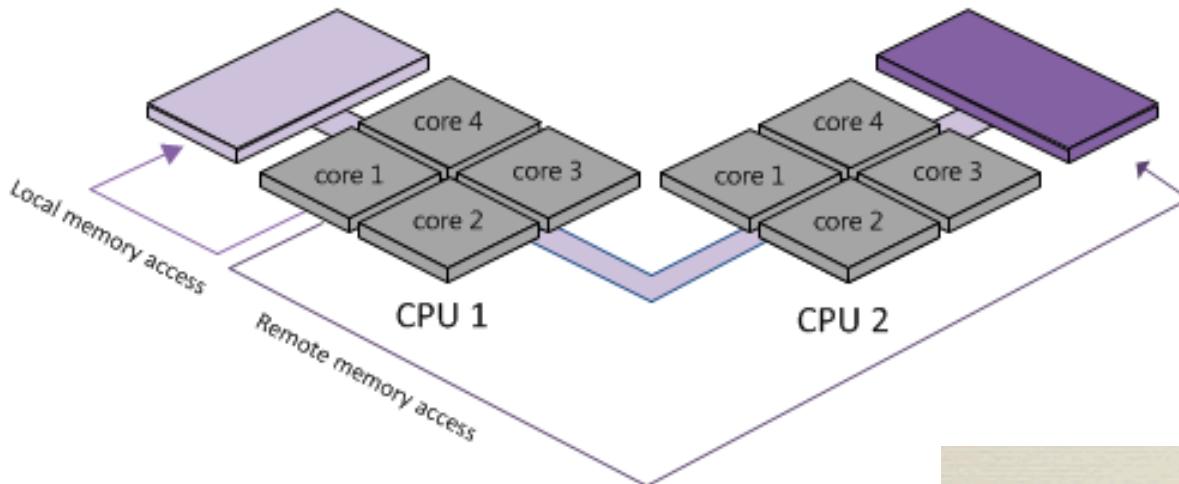
# Multi-Core Processors (NUMA)

- Multiple CPUs on a single chip



Non-uniform memory access (**NUMA**)

# Multi-Core Processors (NUMA)



# Check CPU topology

- \$ likwid-topology -g

```
ksuo@ksuo-VirtualBox ~/likwid-5.0.0> likwid-topology -g
-----
CPU name:      Intel(R) Core(TM) i9-9880H CPU @ 2.30GHz
CPU type:      Intel CoffeeLake processor
CPU stepping:  13
*****
Hardware Thread Topology
*****
Sockets:        1
Cores per socket: 4
Threads per core: 1
-----
HWThread   Thread   Core   Socket   Available
0          0        0       0        *
1          0        1       0        *
2          0        2       0        *
3          0        3       0        *
-----
Socket 0:      ( 0 1 2 3 )
-----
*****
Graphical Topology
*****
Socket 0:
+-----+
| +---+ +---+ +---+ +---+ |
| | 0 | | 1 | | 2 | | 3 | |
| +---+ +---+ +---+ +---+ |
| +---+ +---+ +---+ +---+ |
| | 32 kB | | 32 kB | | 32 kB | | 32 kB | |
| +---+ +---+ +---+ +---+ |
| +---+ +---+ +---+ +---+ |
| | 256 kB | | 256 kB | | 256 kB | | 256 kB | |
| +---+ +---+ +---+ +---+ |
| +---+ +---+ +---+ +---+ |
| | 16 MB | | 16 MB | | 16 MB | | 16 MB | |
| +---+ +---+ +---+ +---+ |
```

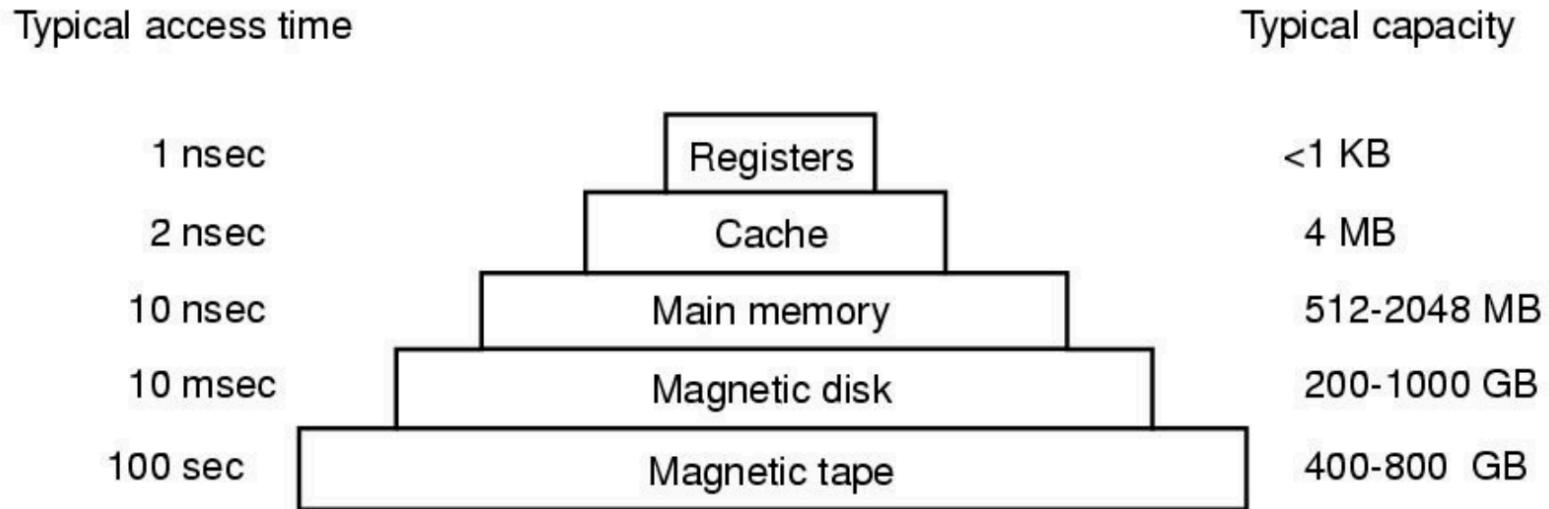
NUMA example:

<https://github.com/RRZE-HPC/likwid/wiki/TutorialNUMA>



# Memory

- A typical memory hierarchy

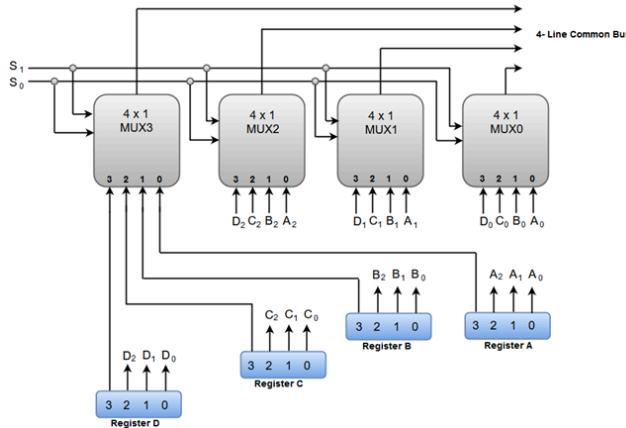


Minimize the access time vs. Cost

# Memory

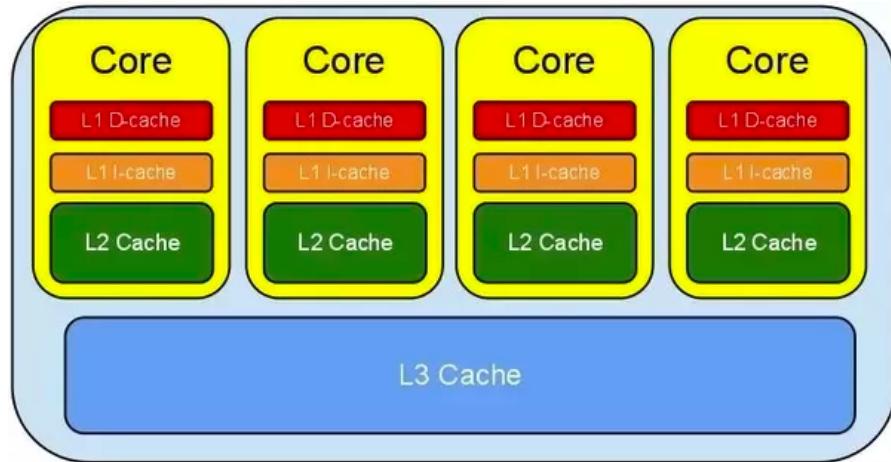
- A typical memory hierarchy

Bus System for 4 Registers:

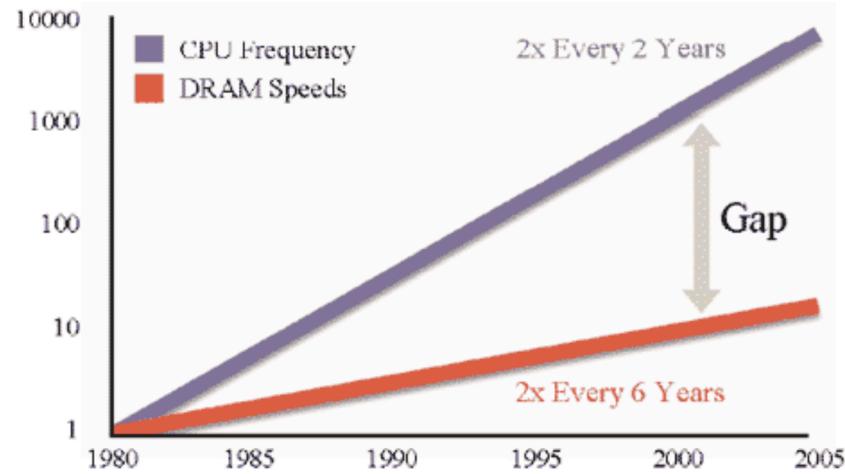


# Cache

- Why Cache is important?



A larger size than registers  
A much faster speed than memory  
Tradeoff between performance and cost



| MacBook Pro                 |  |
|-----------------------------|--|
| Hardware Overview:          |  |
| Model Name:                 | MacBook Pro                                |
| Model Identifier:           | MacBookPro15,1                             |
| Processor Name:             | Intel Core i9                              |
| Processor Speed:            | 2.3 GHz                                    |
| Number of Processors:       | 1  |
| Total Number of Cores:      | 8  |
| L2 Cache (per Core):        | 256 KB                                     |
| L3 Cache:                   | 16 MB                                      |
| Hyper-Threading Technology: | Enabled                                    |
| Memory:                     | 16 GB                                      |
| Boot ROM Version:           | 220.270.99.0.0 (iBridge: 16.16.6568.0.0,0) |
| Serial Number (system):     | C02YR4JHLVCJ                               |
| Hardware UUID:              | DCC2D30A-9630-57B5-89A2-5F2B85254DC1       |



# Check Cache info

- \$ likwid-topology -g
- \$ lscpu | grep cache

```
*****
Cache Topology
*****
Level:          1
Size:           32 kB
Cache groups:  ( 0 ) ( 1 ) ( 2 ) ( 3 )

Level:          2
Size:           256 kB
Cache groups:  ( 0 ) ( 1 ) ( 2 ) ( 3 )

Level:          3
Size:           16 MB
Cache groups:  ( 0 ) ( 1 ) ( 2 ) ( 3 )

*****
NUMA Topology
*****
NUMA domains:  1

Domain:        0
Processors:    ( 0 1 2 3 )
Distances:     10
Free memory:   1967.79 MB
Total memory:  3942.19 MB
```

```
ksuo@ksuo-VirtualBox ~/likwid-5.0.0> lscpu | grep cache
L1d cache:            32K
L1i cache:            32K
L2 cache:             256K
L3 cache:            16384K
```

# Memory information

- free

```
administrator@ubuntuvm-1604 ~> free
              total        used        free      shared  buff/cache   available
Mem:       8168756     1348160     3031888        97424    3788708     6383036
Swap:      998396          0     998396
```

- The free command displays:
  - ✓ Total amount of free and used physical memory
  - ✓ Total amount of swap memory in the system
  - ✓ Buffers and caches used by the kernel

# Memory information

- cat /proc/meminfo
  - MemTotal
  - MemFree
  - MemAvailable
  - Buffers
  - Cached
  - SwapCached
  - SwapTotal
  - SwapFree

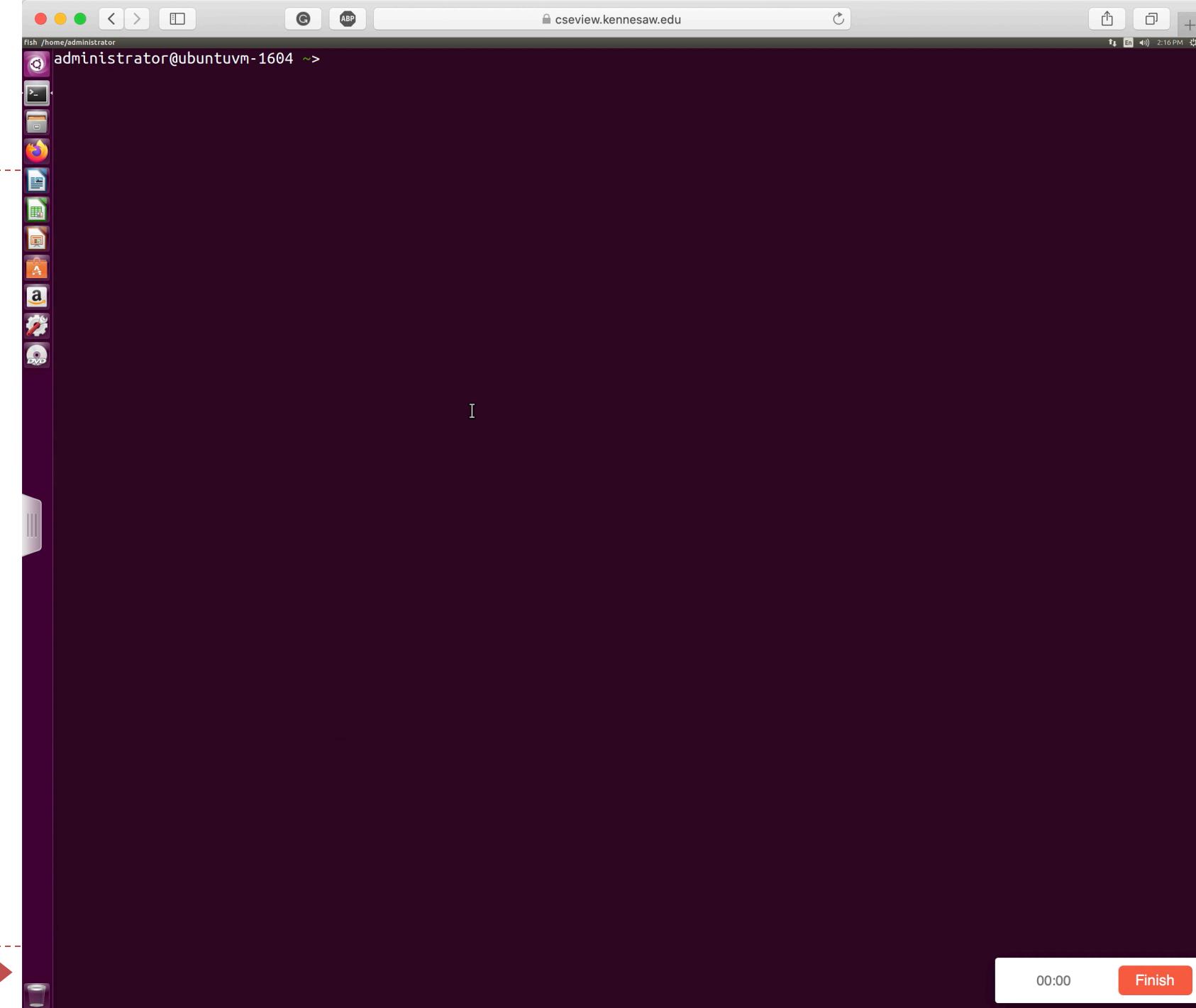
```
administrator@ubuntuvm-1604 ~> cat /proc/meminfo
MemTotal:           8168756 kB
MemFree:            3034596 kB
MemAvailable:       6385744 kB
Buffers:             287636 kB
Cached:              3188812 kB
SwapCached:          0 kB
Active:              2523764 kB
Inactive:            2177968 kB
Active(anon):        1226312 kB
Inactive(anon):      96392 kB
Active(file):        1297452 kB
Inactive(file):      2081576 kB
Unevictable:          48 kB
Mlocked:              48 kB
SwapTotal:            998396 kB
SwapFree:             998396 kB
Dirty:                  264 kB
Writeback:              0 kB
AnonPages:            1225344 kB
Mapped:                304212 kB
Shmem:                 97424 kB
Slab:                  312268 kB
SReclaimable:         278104 kB
SUnreclaim:            34164 kB
KernelStack:            8896 kB
PageTables:            31004 kB
NFS_Unstable:           0 kB
Bounce:                  0 kB
WritebackTmp:           0 kB
CommitLimit:           5082772 kB
Committed_AS:          5183160 kB
VmallocTotal:          34359738367 kB
VmallocUsed:             0 kB
VmallocChunk:             0 kB
HardwareCorrupted:      0 kB
AnonHugePages:            0 kB
ShmemHugePages:           0 kB
ShmemPmdMapped:          0 kB
CmaTotal:                  0 kB
CmaFree:                  0 kB
HugePages_Total:          0
HugePages_Free:            0
HugePages_Rsvd:            0
HugePages_Surp:             0
Hugepagesize:             2048 kB
DirectMap4k:              135040 kB
DirectMap2M:              5107712 kB
DirectMap1G:              5242880 kB
```



# Memory information

---

- lstopo: show the CPU cache and logical CPU layout
- Install: `$ sudo apt-get install hwloc`
- Run: `$ lstopo`



# Test: what is the speed of your memory?

---

- STREAM Benchmark:

it can measure the performance of the memory system,  
including bandwidth and latency.

<https://www.cs.virginia.edu/stream/>

| Function | Rate (MB/s) | Avg time | Min time | Max time |
|----------|-------------|----------|----------|----------|
| Copy:    | 16302.4084  | 0.0026   | 0.0020   | 0.0031   |
| Scale:   | 13411.0440  | 0.0029   | 0.0024   | 0.0040   |
| Add:     | 15529.6662  | 0.0041   | 0.0031   | 0.0050   |
| Triad:   | 12889.0264  | 0.0042   | 0.0037   | 0.0047   |

# Test: what is the speed of your memory?

---

Copy

```
void tuned_STREAM_Copy()
```

```
{
```

```
    int j;
```

```
    for (j=0; j<N; j++)
```

```
        c[j] = a[j];
```

*Read from one memory cell  
Write to another memory cell*

```
}
```

# Test: what is the speed of your memory?

---

Scale

```
void tuned_STREAM_Scale(double scalar)
```

```
{
```

```
    int j;
```

```
    for (j=0; j<N; j++)
```

```
        b[j] = scalar*c[j];
```

```
}
```

*Read from one memory cell  
Make a multiply operation  
Write to another memory cell*

# Test: what is the speed of your memory?

---

Sum

```
void tuned_STREAM_Add()
```

```
{
```

```
    int j;
```

```
    for (j=0; j<N; j++)
```

```
        c[j] = a[j]+b[j];
```

```
}
```

*Read from two memory cell  
Make an add operation  
Write to another memory cell*

# Test: what is the speed of your memory?

---

Triad

```
void tuned_STREAM_Triad(double scalar)
```

```
{
```

```
    int j;
```

```
    for (j=0; j<N; j++)
```

```
        a[j] = b[j]+scalar*c[j];
```

```
}
```

*Read from two memory cell*

*Make an add operation and a multiply operation*

*Write to another memory cell*



# Test: what is the speed of your memory?

---

- 1, Download STREAM Benchmark

```
$ wget https://www.nersc.gov/assets/Trinity--NERSC-8-RFP/Benchmarks/Jan9/stream.tar
```

- 2, Compile & run

```
$ tar xvf stream.tar
```

```
$ gcc stream.c -o stream
```

```
$ ./stream
```

# Test: what is the speed of your memory?

---

- 1, install memory bandwidth app

```
$ sudo apt-get install mbw
```

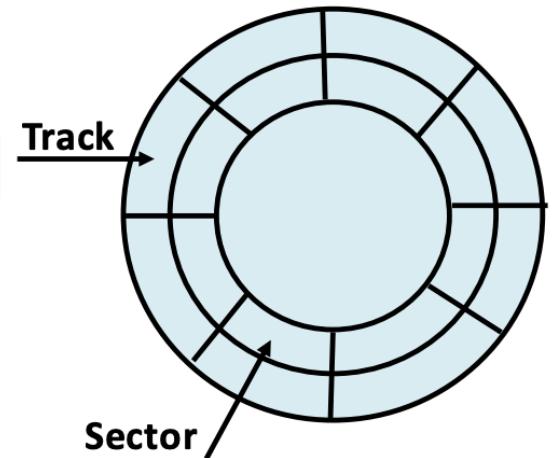
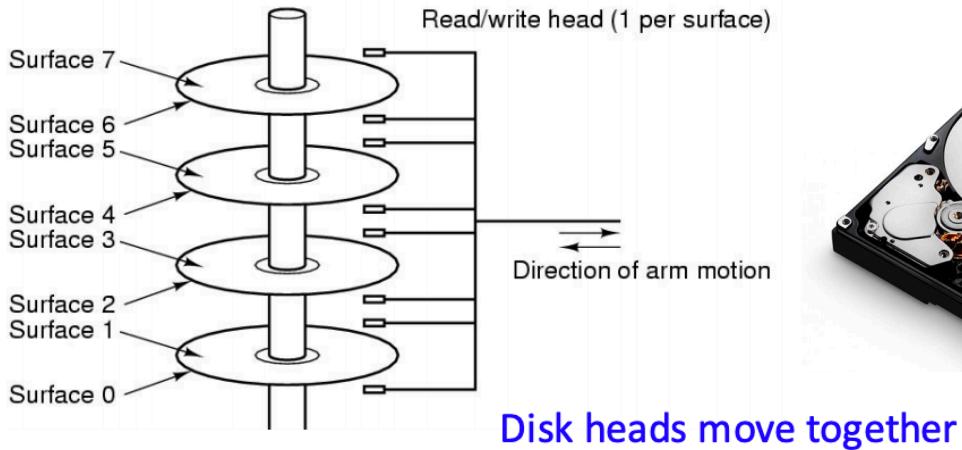
- 2, run

```
$ mbw -q -n 100 256
```

n: run 100 times

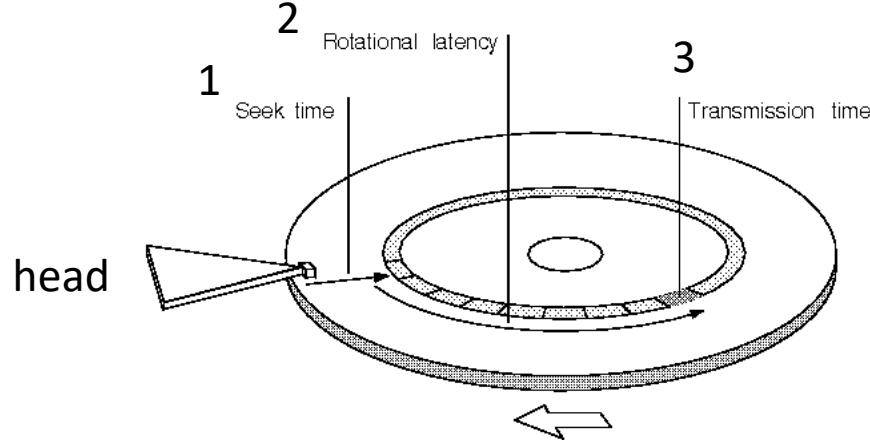
256: means test 256MB in memory

# Disk



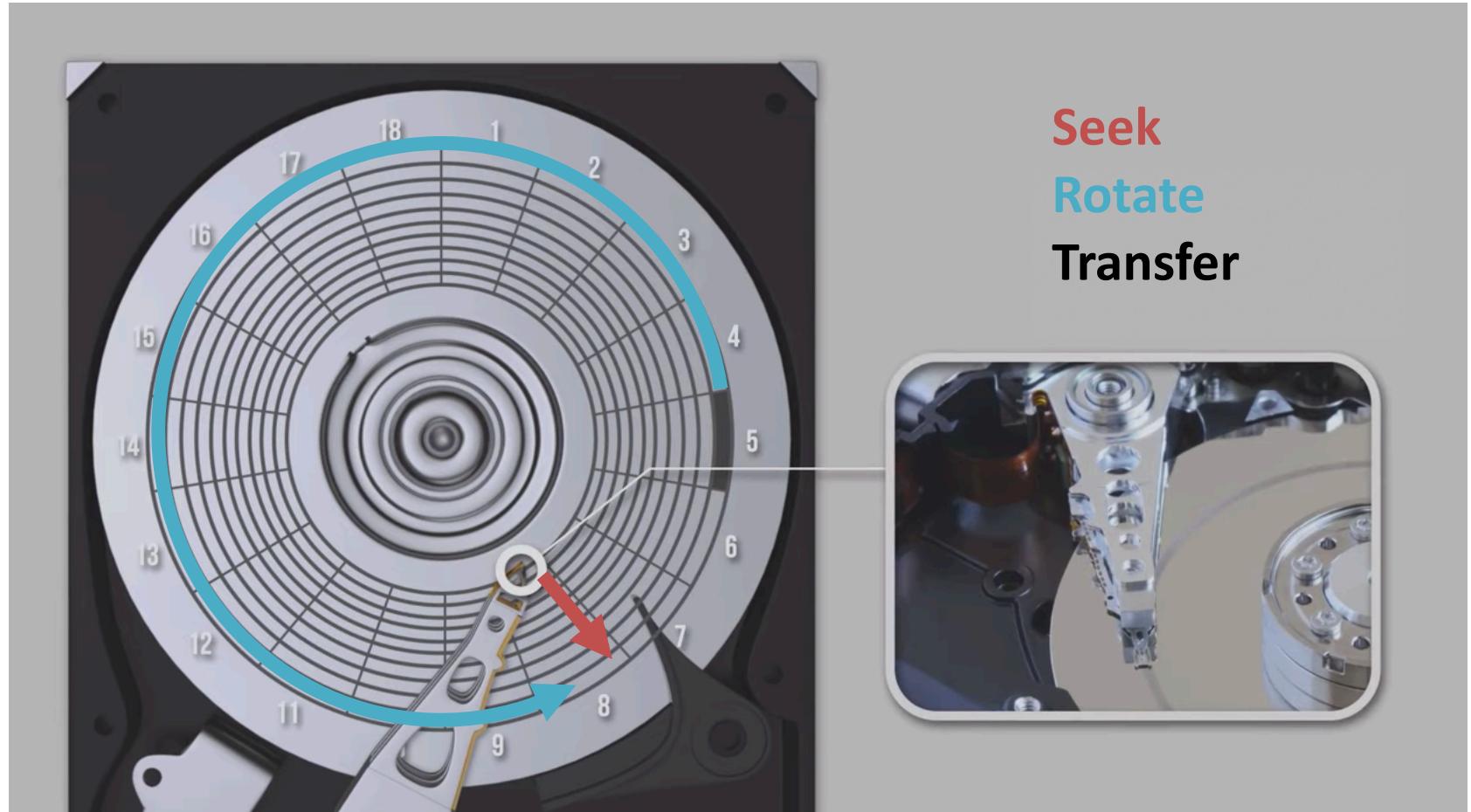
- A stack of platters, a surface with a magnetic coating
- Typical numbers (depending on the disk size):
  - 500 to 2,000 tracks per surface
  - 32 to 128 sectors per track
    - ▶ A sector is the smallest unit that can be read or written
- Originally, all tracks have the same number of sectors

# Disk



- Disk head: each side of a platter has separate disk head
- Read/write data is a three-stage process:
  - Seek time: position the arm over the proper track
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 15 ms

# Disk



# Disk information

---

- `lsblk`
  - Lists out all the storage blocks, which includes disk partitions and optical drives. Details include the total size of the partition/block and the mount point if any.

```
administrator@ubuntuvm-1604 ~> lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0     11:0    1  1.5G  0 rom  /media/administrator/Ubuntu 16.04.5 LTS amd64
sda      8:0    0   40G  0 disk 
└─sda2   8:2    0    1K  0 part 
└─sda5   8:5    0  975M  0 part [SWAP]
└─sda1   8:1    0   39G  0 part /
```

# Disk information

---

- `df`
  - prints out details about only mounted file systems. The list generated by `df` even includes file systems that are not real disk partitions.

```
administrator@ubuntuvm-1604 ~> df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev            4054632      0   4054632   0% /dev
tmpfs           816876   42148   774728   6% /run
/dev/sda1       40168028 7350596  30753972  20% /
tmpfs           4084376   1044   4083332   1% /dev/shm
tmpfs            5120      0     5120   0% /run/lock
tmpfs           4084376      0   4084376   0% /sys/fs/cgroup
/dev/sr0        1610928 1610928          0 100% /media/administrator/Ubuntu 16.04.5 LTS amd64
tmpfs           816876     56   816820   1% /run/user/1000
```

# Disk information

- **fdisk**
  - display the partitions and details like file system type

```
administrator@ubuntuvm-1604 ~> sudo fdisk -l
[sudo] password for administrator:
Disk /dev/sda: 40 GiB, 42949672960 bytes, 83886080 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6e6dc012

Device      Boot   Start     End   Sectors   Size Id Type
/dev/sda1    *      2048 81885183 81883136   39G 83 Linux
/dev/sda2          81887230 83884031 1996802 975M  5 Extended
/dev/sda5          81887232 83884031 1996800 975M 82 Linux swap / Solaris
```



# Disk R/W Process

---



# Disk R/W Speed

- \$ sudo hdparm -Tt /dev/sda

```
administrator@ubuntuvm-104 ~> sudo hdparm -Tt /dev/sda
/dev/sda:
Timing cached reads: 17132 MB in 1.99 seconds = 8599.49 MB/sec
Timing buffered disk reads: 592 MB in 3.04 seconds = 194.58 MB/sec
```

Read cache data size

Read cache speed

Read disk data size

Read disk speed

# dd: Disk Write Speed

- \$ dd if=/dev/zero of=/testw.dbf bs=4k count=100000 oflag=direct

/dev/zero: generate null string stream

Write into this file

```
sudo dd if=/dev/zero of=/testw.dbf bs=4k count=100000 oflag=direct
100000+0 records in
100000+0 records out
409600000 bytes (410 MB, 391 MiB) copied, 25.1254 , 16.3 MB/s
```

Block size 4k

Block number 100000

# dd: Disk Read Speed

- \$ dd if=/dev/sdb of=/dev/null bs=4k  
count=100000

/dev/zero: the source file

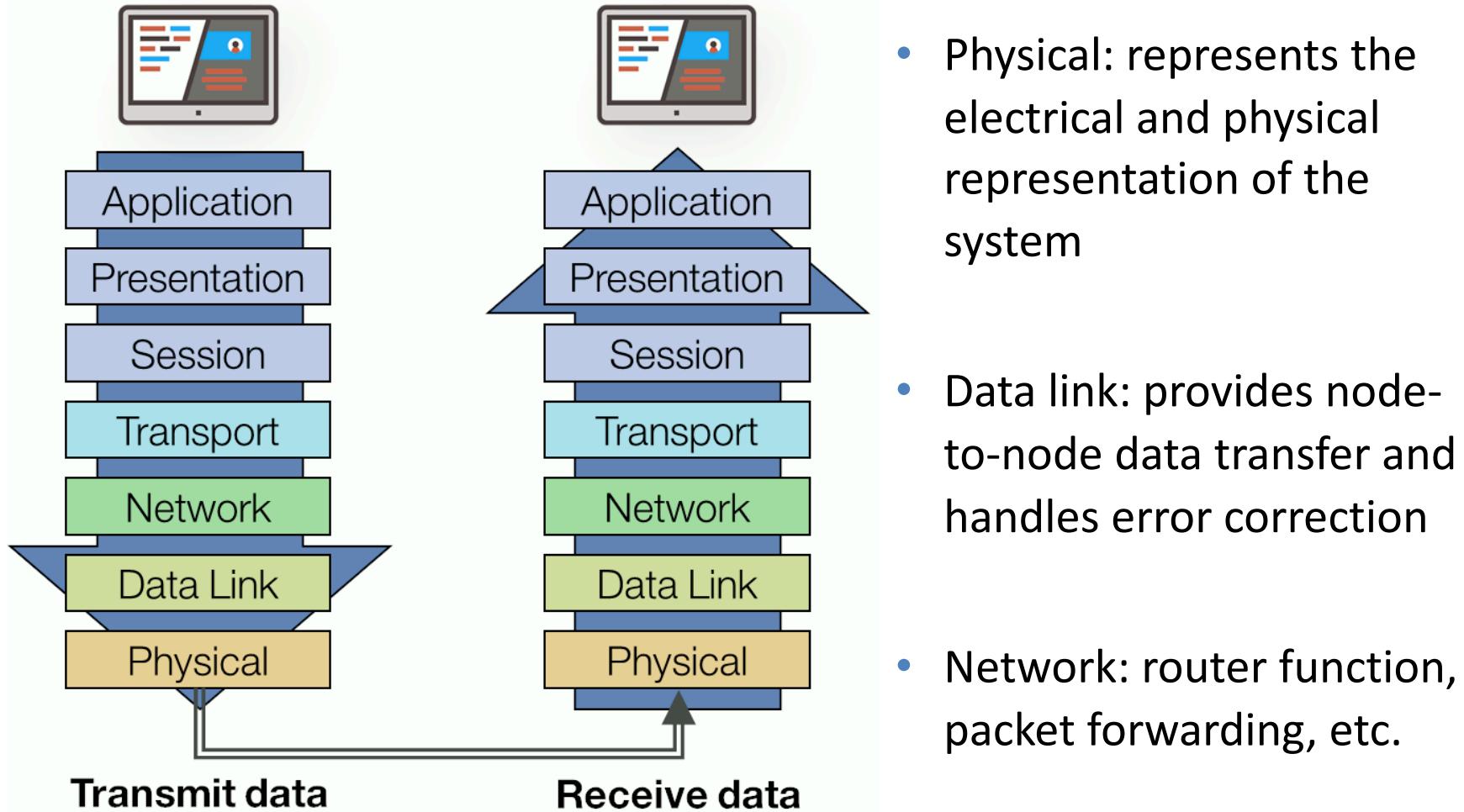
Write into this file (it is an empty file)

```
administrator@ubuntuvm-1604 ~>
sudo dd if=/dev/sda of=/dev/null bs=4k count=100000
100000+0 records in
100000+0 records out
4096000000 bytes (410 MB, 391 MiB) copied, 2.03887 s, 197 MB/s
```

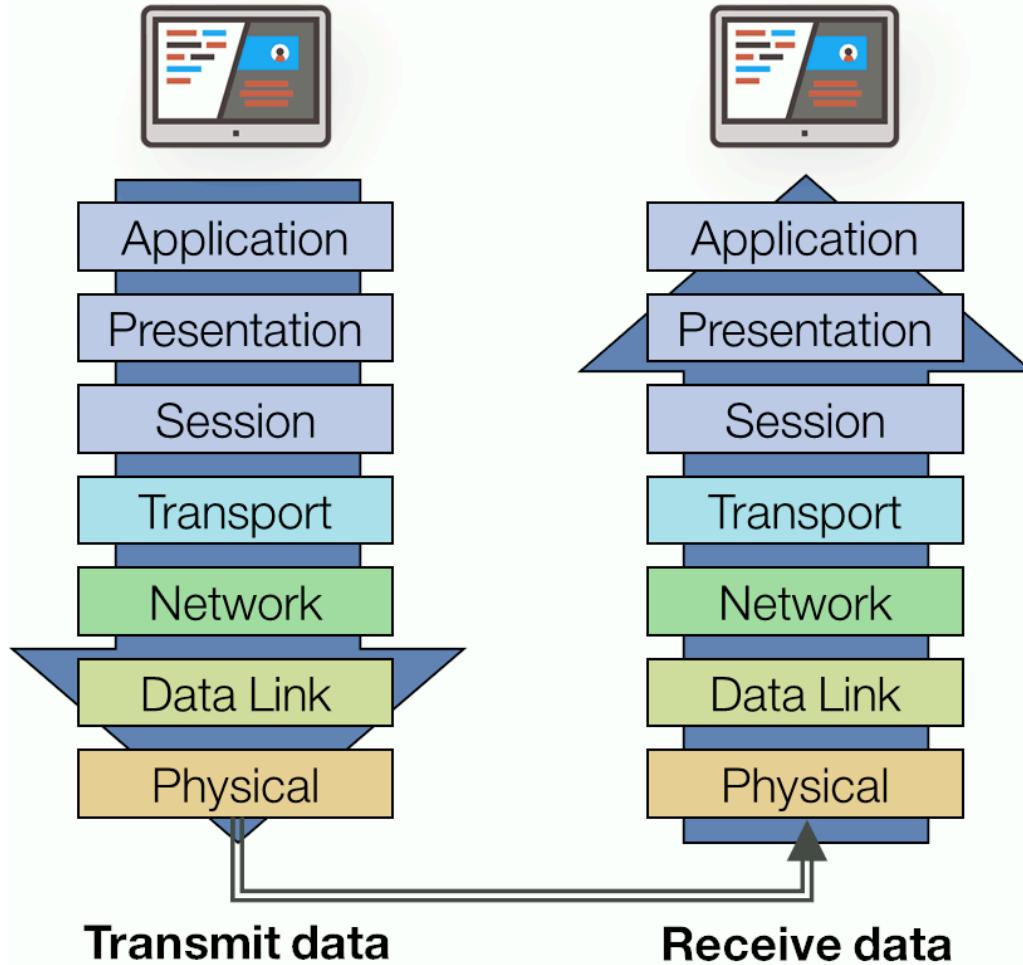
Block size 4k

Block number  
100000

# Networking

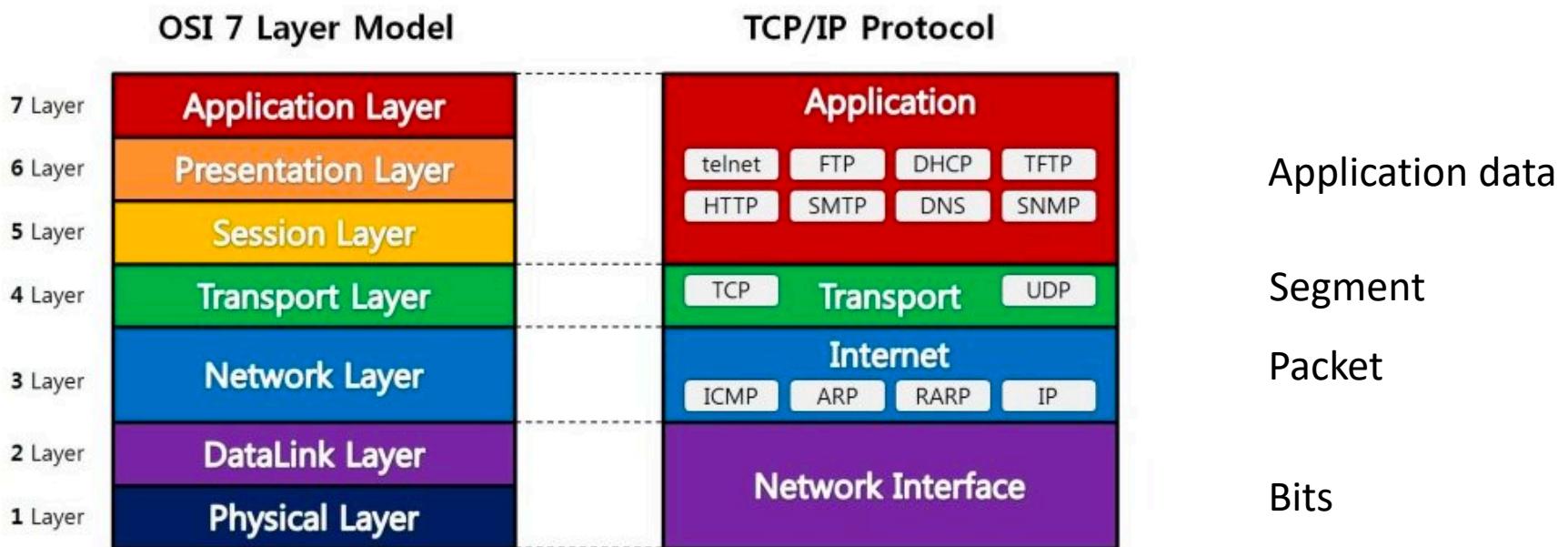


# Networking



- Transport: coordination of the data transfer between end systems and hosts. How much data to send, at what rate, where it goes, etc.
- Session/Presentation/Application: interaction with the user level application

# Networking



# Networking connection: ping

- Ping is a computer network administration software utility used to test the reachability of a host on an Internet Protocol network.

```
administrator@ubuntuvm-1604 ~> ping www.google.com
PING www.google.com (216.58.194.228) 56(84) bytes of data.
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=1 ttl=116 time=2.70 ms
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=2 ttl=116 time=3.11 ms
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=3 ttl=116 time=2.81 ms
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=4 ttl=116 time=3.14 ms
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=5 ttl=116 time=3.01 ms
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=6 ttl=116 time=3.07 ms
64 bytes from atl14s63-in-f4.1e100.net (216.58.194.228): icmp_seq=7 ttl=116 time=3.03 ms
^C
```



# Networking throughput: speedtest-cli

- \$ sudo apt install python-pip
- \$ pip install speedtest-cli
- \$ which speedtest-cli | bash -

```
administrator@ubuntuvm-1604 ~> which speedtest-cli | bash -
Retrieving speedtest.net configuration...
Testing from PeachNet (168.28.186.189)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by Windstream (Atlanta, GA) [37.21 km]: 5.168 ms
Testing download speed.....
.....
Download: 1163.29 Mbit/s
Testing upload speed.....
.....
Upload: 1465.92 Mbit/s
```

An interesting video introducing hardware

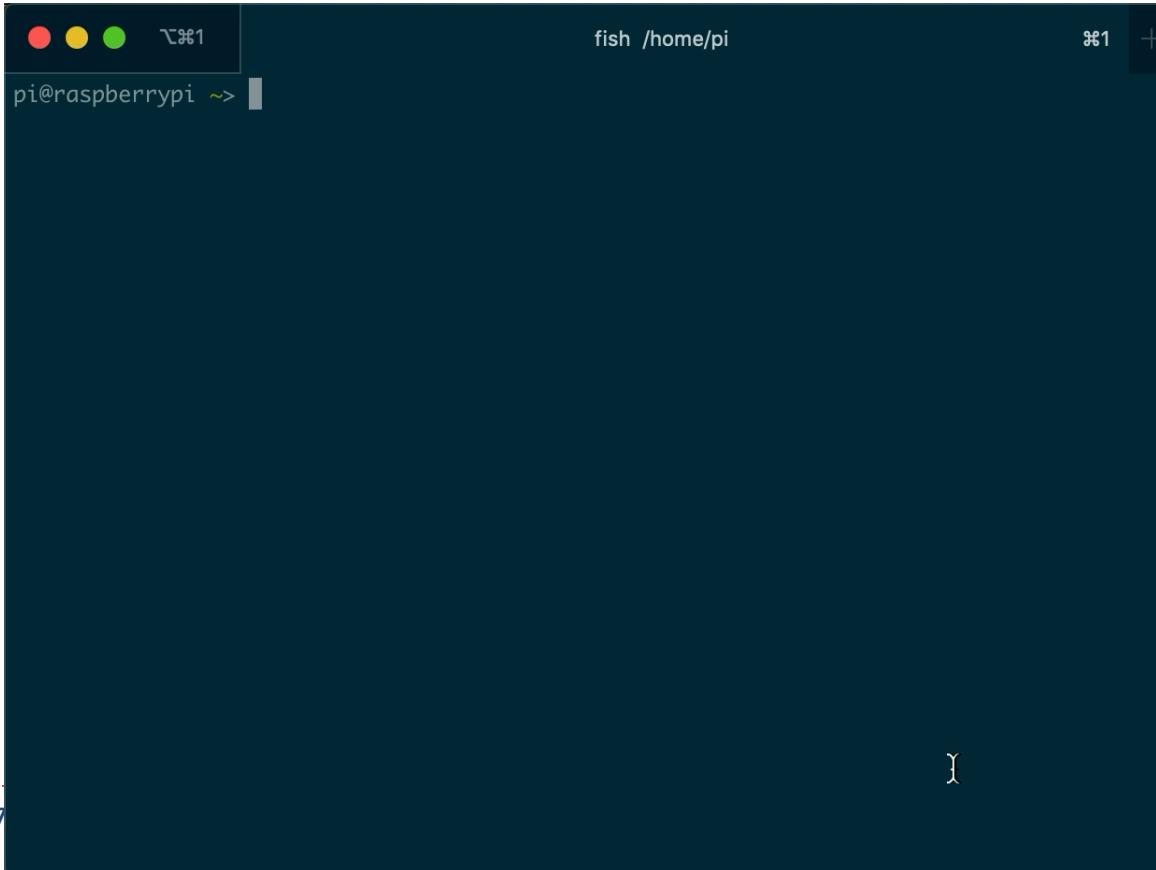
# Computer Parts!



<https://www.youtube.com/watch?v=ExxFxD4OSZ0>

# How to get my machine spec?

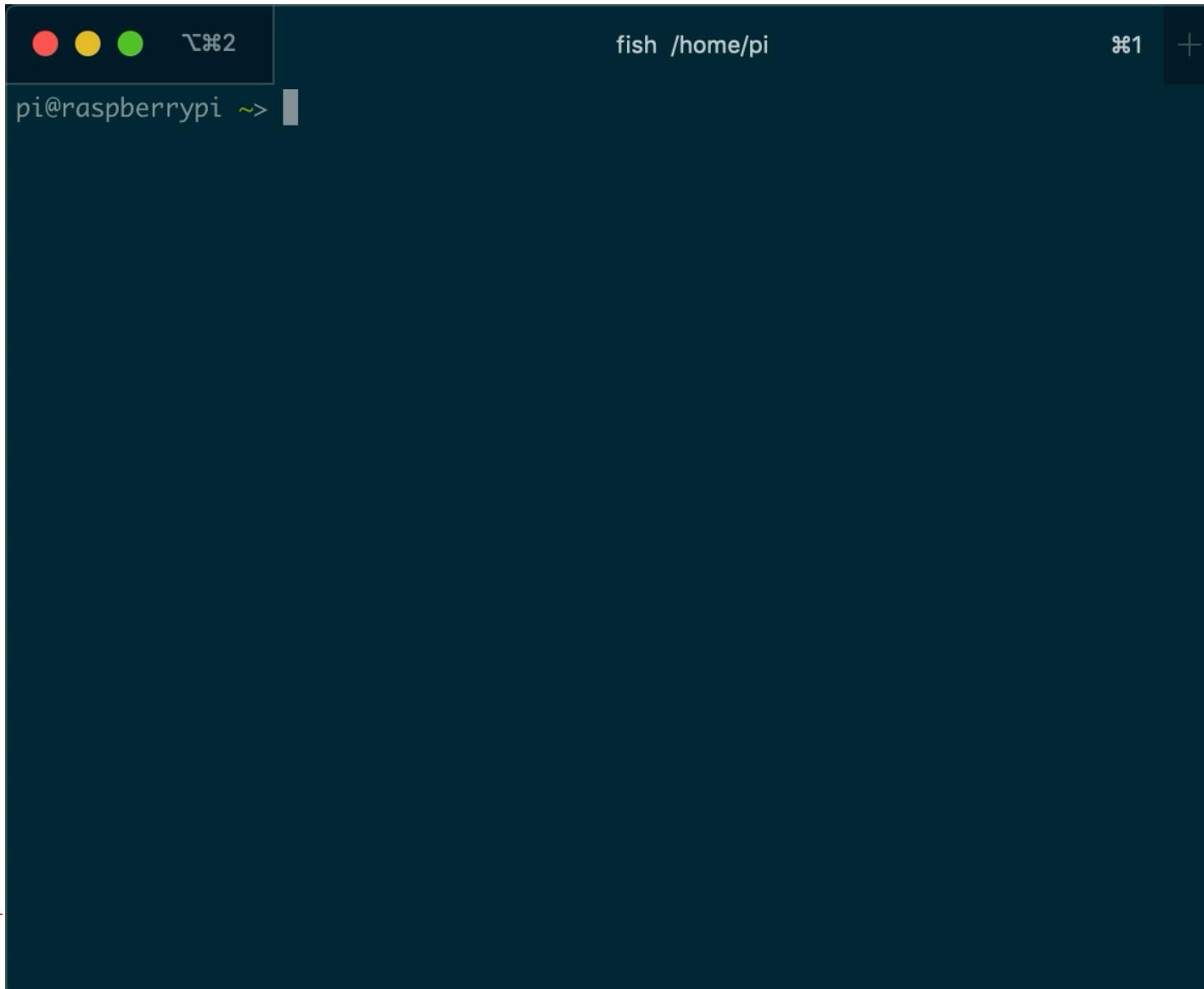
- Inxi: <https://www.tecmint.com/inxi-command-to-find-linux-system-information/>



A screenshot of a terminal window on a Raspberry Pi. The window title bar shows three colored dots (red, yellow, green) and the text 'fish /home/pi'. The status bar at the bottom right shows '⌘1 +'. The terminal prompt is 'pi@raspberrypi ~>'. The main area of the terminal is completely black, indicating no output from the command.



# How to get my machine spec? One command for All!



# Test 1: Amazon Web Service VM

- ssh -i "osclass.pem" ubuntu@ec2-3-133-133-128.us-east-2.compute.amazonaws.com

The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with links for New EC2 Experience, EC2 Dashboard, Events, Tags, Reports, Limits, and INSTANCES (with sub-links for Instances and Instance Types). The main area has tabs for Launch Instance, Connect, and Actions. A search bar at the top says "Filter by tags and attributes or search by keyword". Below it is a table with columns: Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, and Public DNS (IPv4). The table contains three rows:

| Name | Instance ID         | Instance Type | Availability Zone | Instance State | Status Checks  | Alarm Status | Public DNS (IPv4) |
|------|---------------------|---------------|-------------------|----------------|----------------|--------------|-------------------|
|      | i-07f06331098f60f9d | t2.micro      | us-east-2c        | terminated     | None           |              |                   |
|      | i-0cc0052f572ddf928 | t2.micro      | us-east-2a        | running        | 2/2 checks ... | None         |                   |
|      | i-0ed8bfe921659cd7e | t2.micro      | us-east-2c        | terminated     | None           |              |                   |



# Test 2: Microsoft Azure VM

- ssh ksuo@13.90.101.24

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar and a user profile for 'ksuo@kennesaw.edu KENNESAW STATE UNIVERSITY'. The main navigation bar includes 'Home', 'All resources', and the current resource group 'test'. On the left, a sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main content area displays details for a 'Virtual machine' named 'test'. It shows the following information:

| Resource group (change) : test |  | Azure Spot         |
|--------------------------------|--|--------------------|
| Status                         | : Running                              | Public IP address  |
| Location                       | : East US                              | Private IP address |
| Subscription (change)          | : Azure subscription 1                 | Public IP address  |
| Subscription ID                | : dd8ac868-cf63-4b5d-9abf-edcb82209bb5 | Private IP address |
| Computer name                  | : test                                 | Virtual network    |
| Operating system               | : Linux (ubuntu 16.04)                 | DNS name           |
| Size                           | : Standard B1s (1 vcpus, 1 GiB memory) | Scale Set          |
| Tags (change)                  | : Click here to add tags               |                    |

Below this, there are two charts: 'CPU (average)' and 'Network (total)'. The 'CPU (average)' chart shows usage over the last hour, with values ranging from 2% to 6%. The 'Network (total)' chart shows traffic over the last hour, with values ranging from 2MB to 4MB.

To the right, a 'Connect to virtual machine' panel is open, showing options for RDP, SSH, and BASTION. It prompts the user to enable just-in-time access for security. It also provides fields for 'IP address' (Public IP address: 13.90.101.24), 'Port number' (22), and 'Login using VM local account' (ssh ksuo@13.90.101.24). A link for troubleshooting connection issues is also provided.



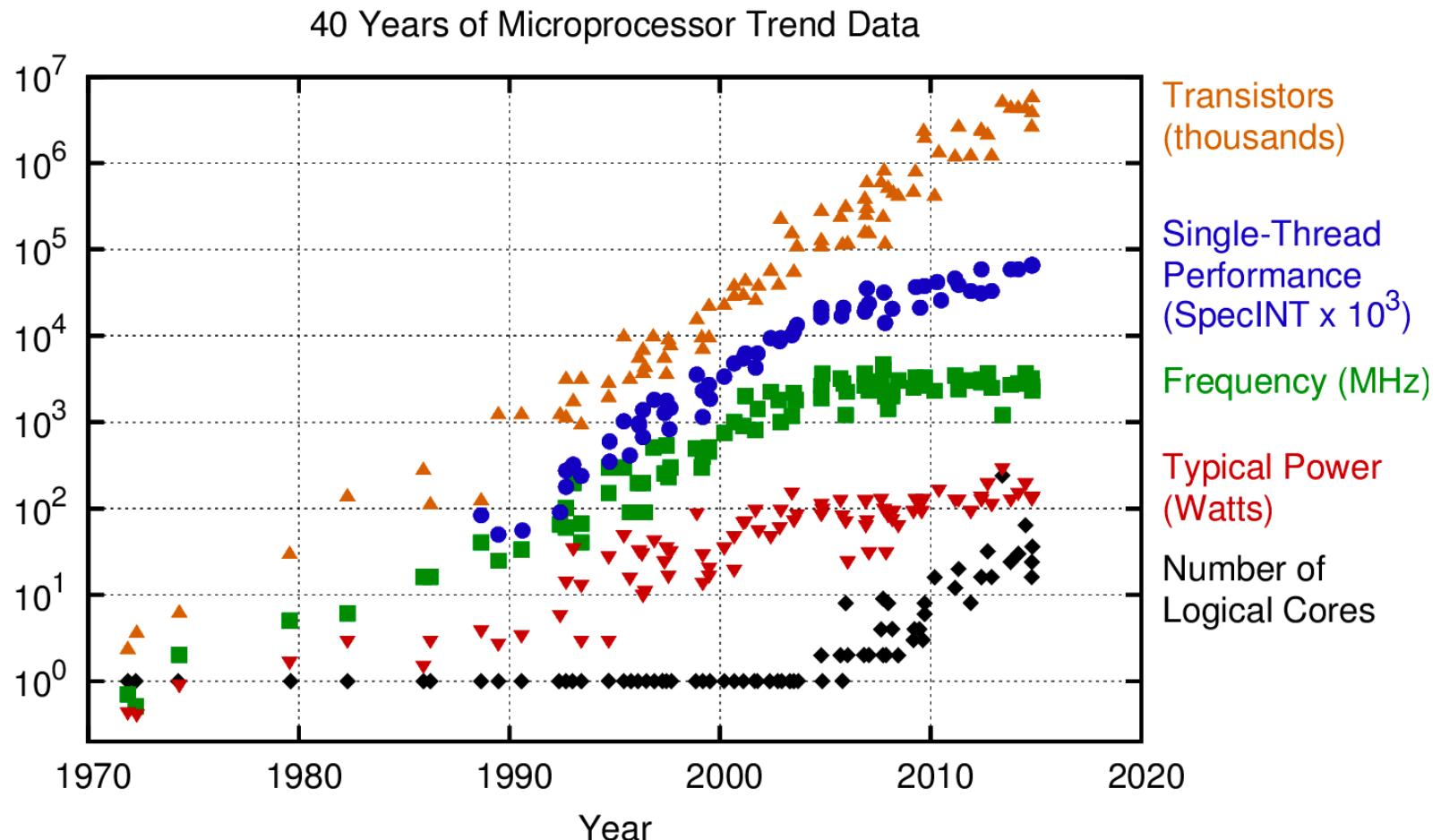
# Changing times

---

- From 1986 – 2002, microprocessors were speeding like a rocket, increasing in performance an average of 50% per year.
- Since then, it's dropped to about 20% increase per year.



# Changing times

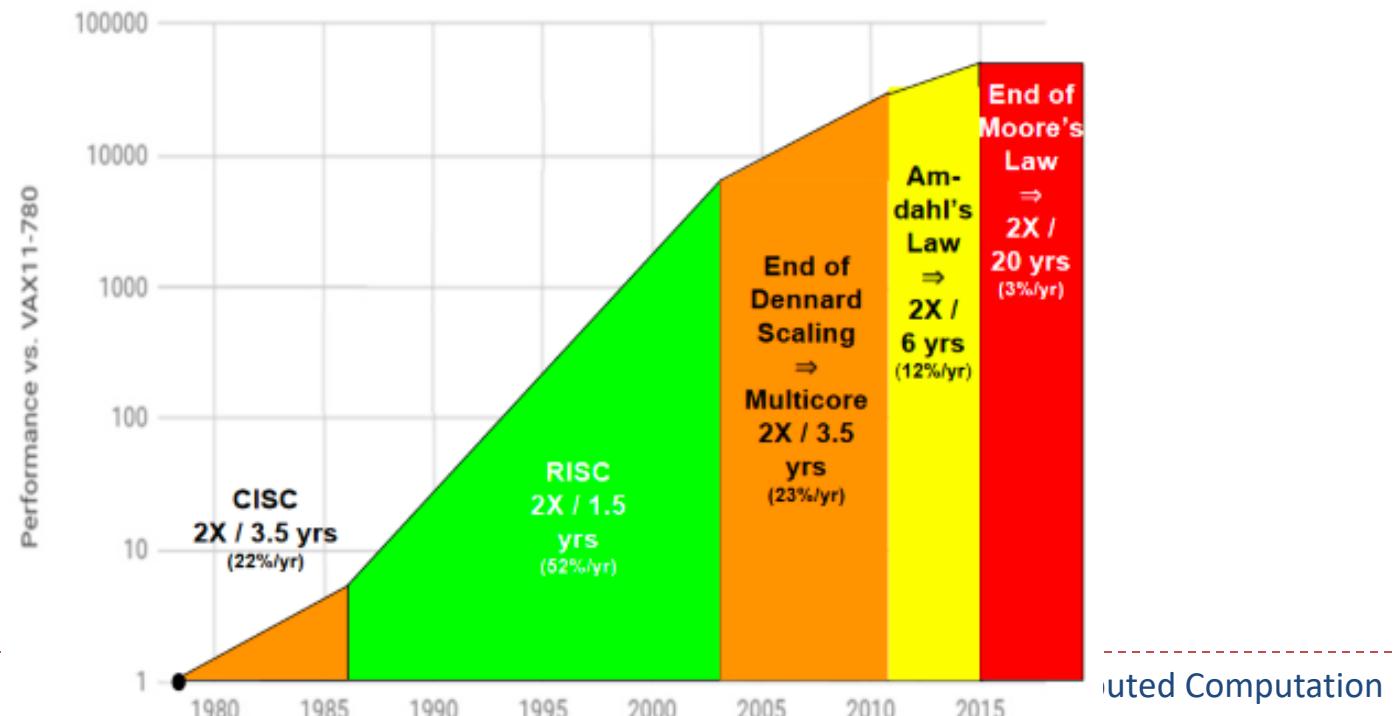


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2015 by K. Rupp



# Moore's Law

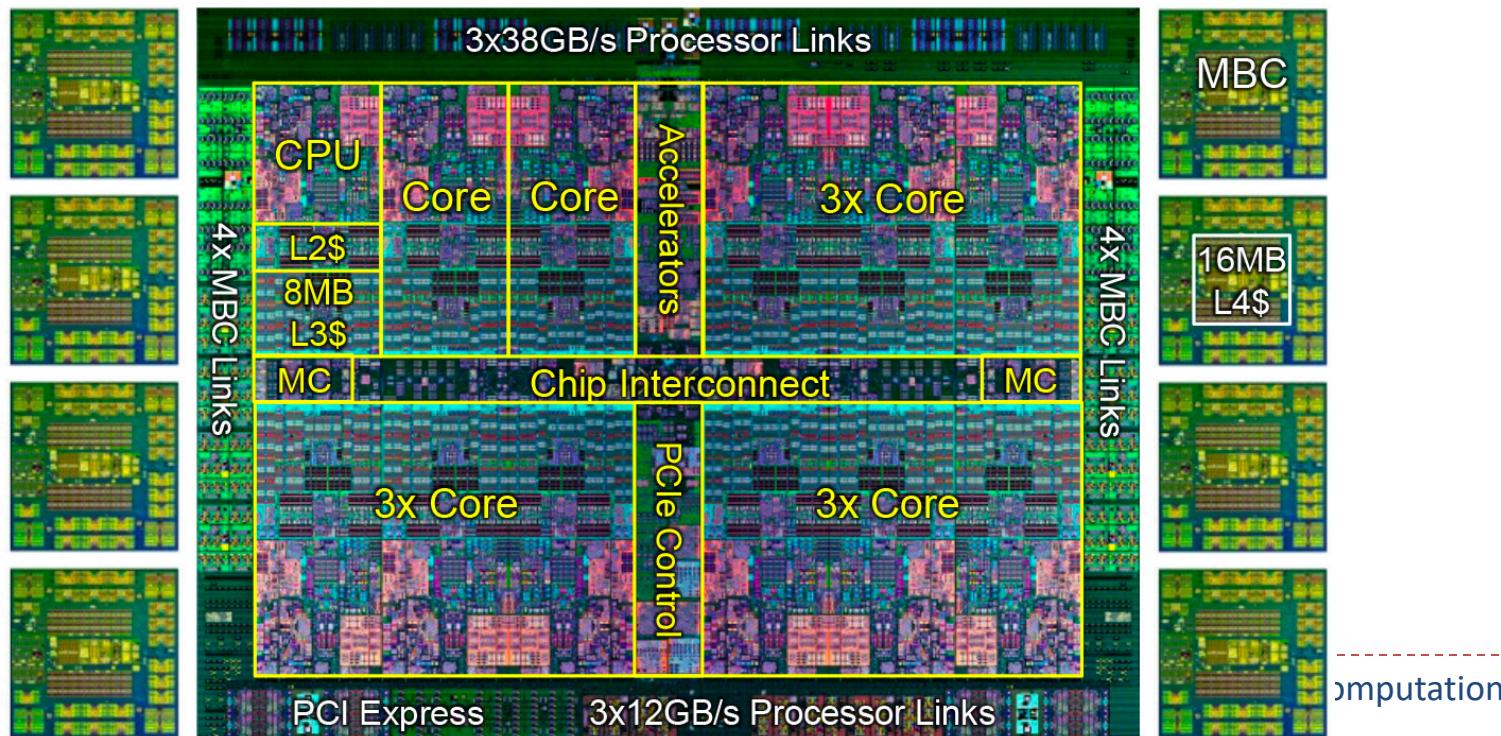
- The observation that the number of transistors in a dense integrated circuit doubles about every two years



# An intelligent solution

<https://www.techradar.com/news/ amd-ryzen-threadripper-3990x-is-a-ridiculous-64-core-cpu-and-its-coming-soon>

- Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit.



# Now it's up to the programmers

---

- Adding more processors doesn't help much if programmers aren't aware of them...
- ... or don't know how to use them.
- Serial programs don't benefit from this approach (in most cases).



# Why we need ever-increasing performance

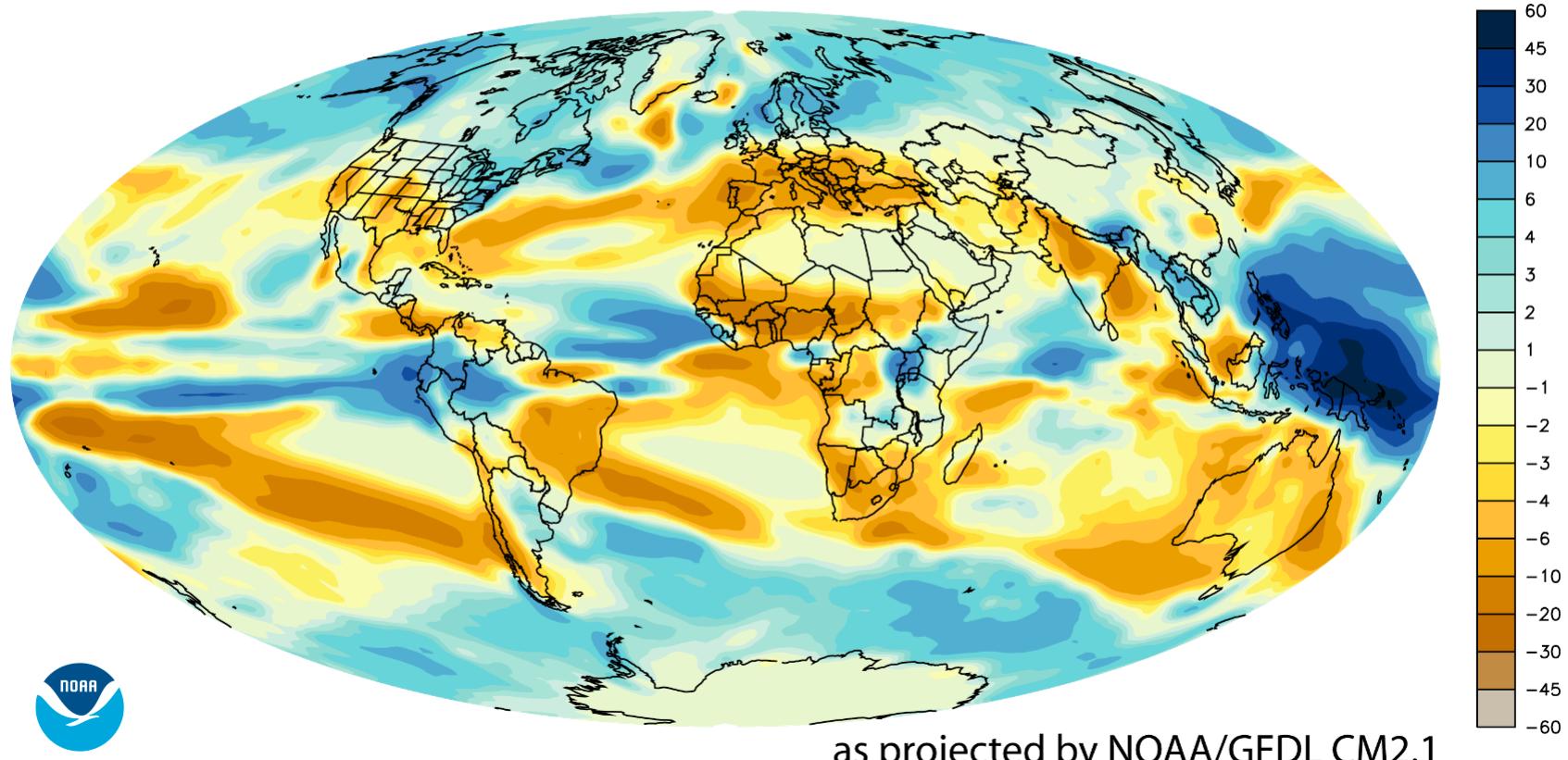
---

- Computational power is increasing, but so are our computation problems and needs.
- Problems we never dreamed of have been solved because of past increases, such as decoding the human genome.
- More complex problems are still waiting to be solved.

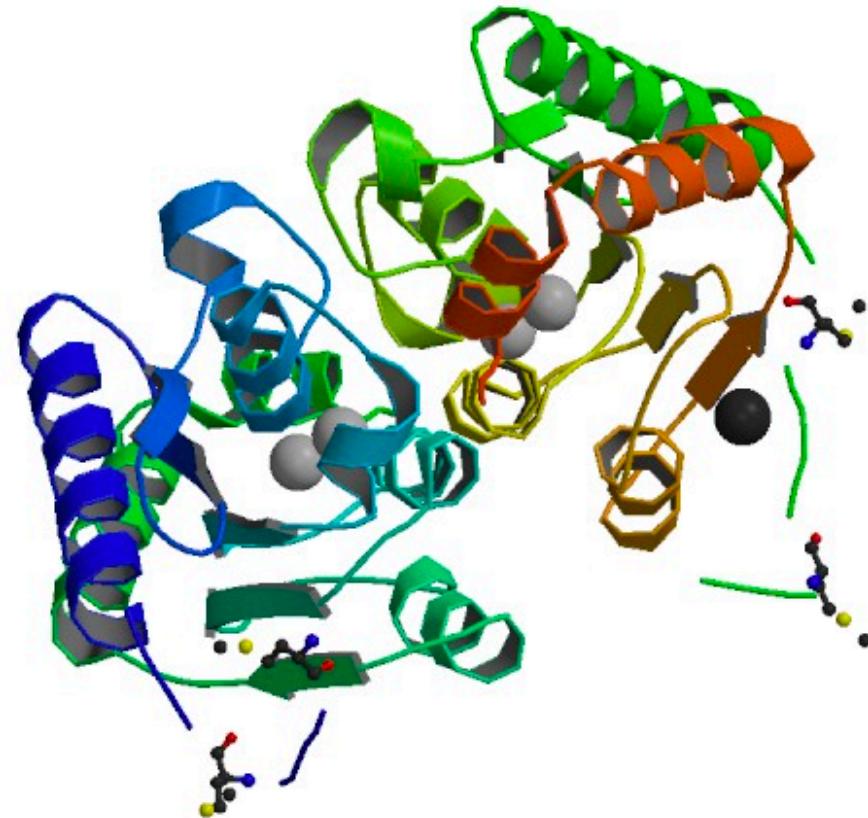
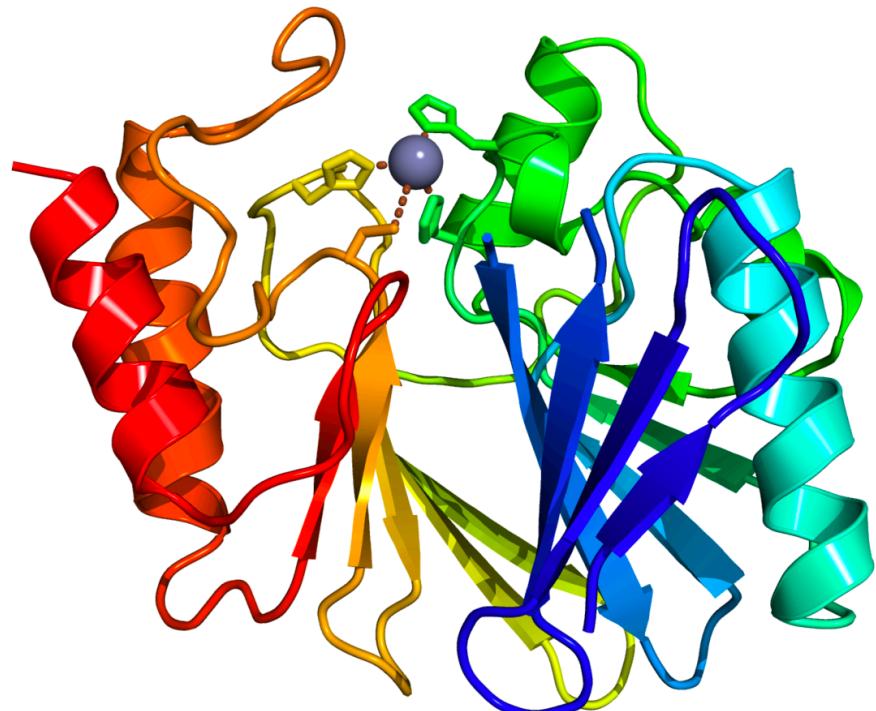


# Climate modeling

CHANGE IN PRECIPITATION BY END OF 21st CENTURY  
inches of liquid water per year



# Protein folding



# Drug discovery

---



# Energy research

---



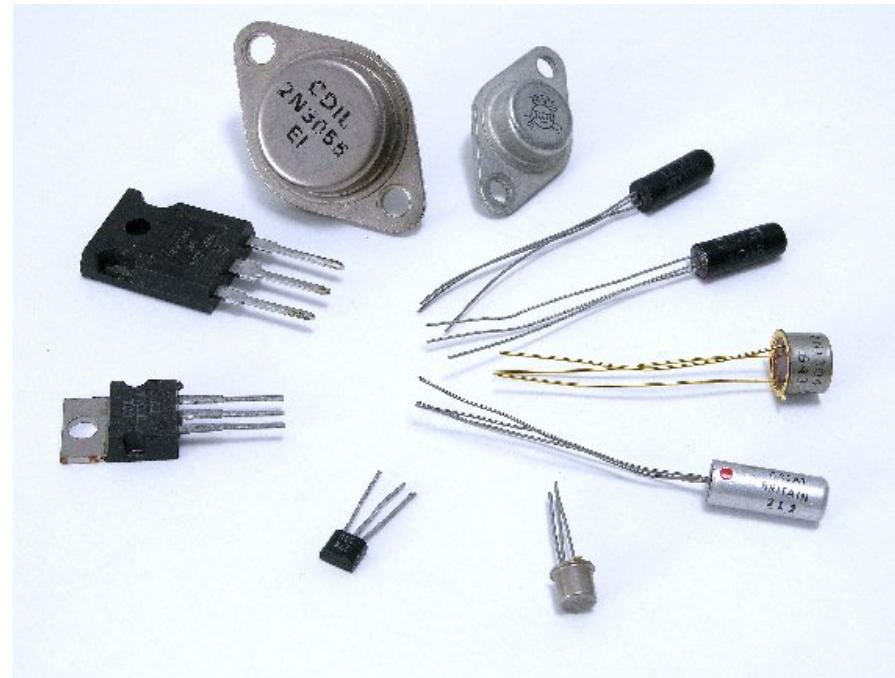
<https://www.ncbi.nlm.nih.gov>

# Data analysis

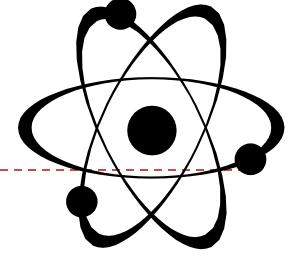


# Why we're building parallel systems

- Up to now, performance increases have been attributable to increasing density of transistors.
- But there are inherent problems.



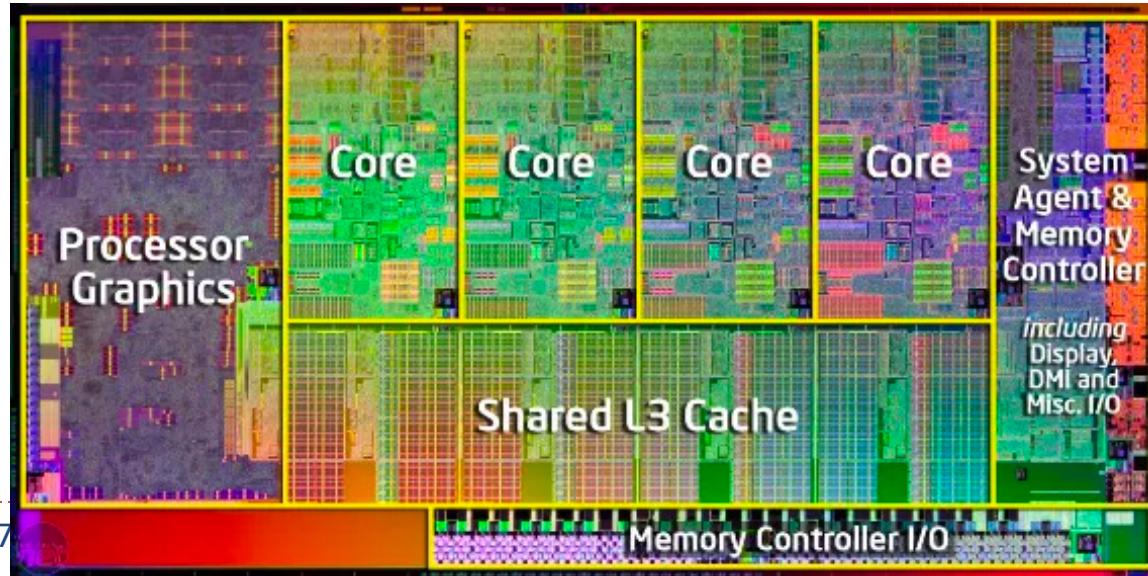
# A little physics lesson



- Smaller transistors = faster processors.
- Faster processors = increased power consumption.
- Increased power consumption = increased heat.
- Increased heat = unreliable processors.

# Solution

- Move away from single-core systems to multicore processors.
- “core” = central processing unit (CPU)
- Introducing parallelism!!!



# Why we need to write parallel programs

---

- Running multiple instances of a serial program often isn't very useful.
- Rewrite serial programs so that they're parallel.
- Write translation programs that automatically convert serial programs into parallel programs.
  - This is very difficult to do.
  - Success has been limited.



# More problems

---

- Some coding constructs can be recognized by an automatic program generator and converted to a parallel construct.
- However, it's likely that the result will be a very inefficient program.
- Sometimes the best parallel solution is to step back and devise an entirely new algorithm.



# Example

---

- Serial solution:

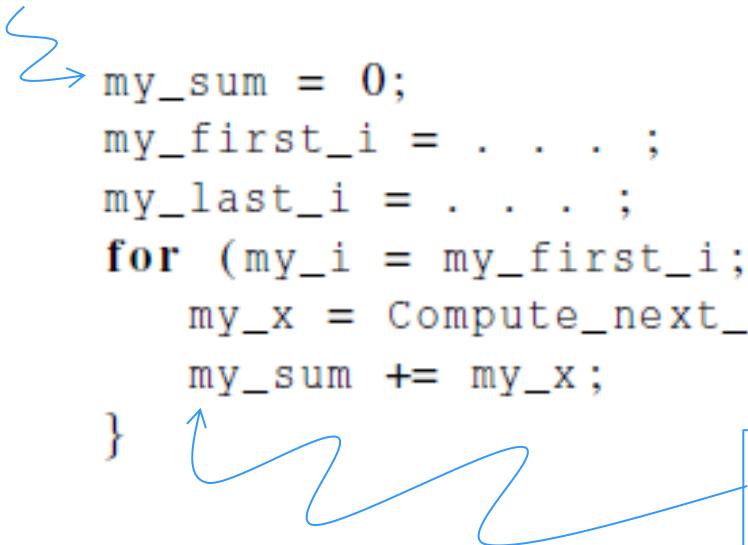
```
sum = 0;  
for (i = 0; i < n; i++) {  
    x = Compute_next_value(. . .);  
    sum += x;  
}
```

- What does this program do?
  - Compute  $n$  values and add them together.



# Example (cont.)

- We have  $p$  cores,  $p$  much smaller than  $n$ .
- Each core performs a partial sum of approximately  $n/p$  values.



```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
    my_x = Compute_next_value( . . . );
    my_sum += my_x;
}
```

Each core uses its own private variables and executes this block of code independently of the other cores.

# Example (cont.)

---

- After each core completes execution of the code, is a private variable `my_sum` contains the sum of the values computed by its calls to `Compute_next_value`.
- Ex., 8 cores,  $n = 24$ , then the calls to `Compute_next_value` return:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9



# Example (cont.)

---

- Once all the cores are done computing their private `my_sum`, they form a global sum by sending results to a designated “master” core which adds the final result.



# Example (cont.)

---

```
if (I'm the master core) {  
    sum = my_x;  
    for each core other than myself {  
        receive value from core;  
        sum += value;  
    }  
} else {  
    send my_x to the master;  
}
```



# Example (cont.)

---

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

| Core   | 0 | 1  | 2 | 3  | 4 | 5  | 6  | 7  |
|--------|---|----|---|----|---|----|----|----|
| my_sum | 8 | 19 | 7 | 15 | 7 | 13 | 12 | 14 |

## Global sum

$$8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95$$

| Core   | 0  | 1  | 2 | 3  | 4 | 5  | 6  | 7  |
|--------|----|----|---|----|---|----|----|----|
| my_sum | 95 | 19 | 7 | 15 | 7 | 13 | 12 | 14 |



# But wait!

---

- There's a much better way to compute the global sum.

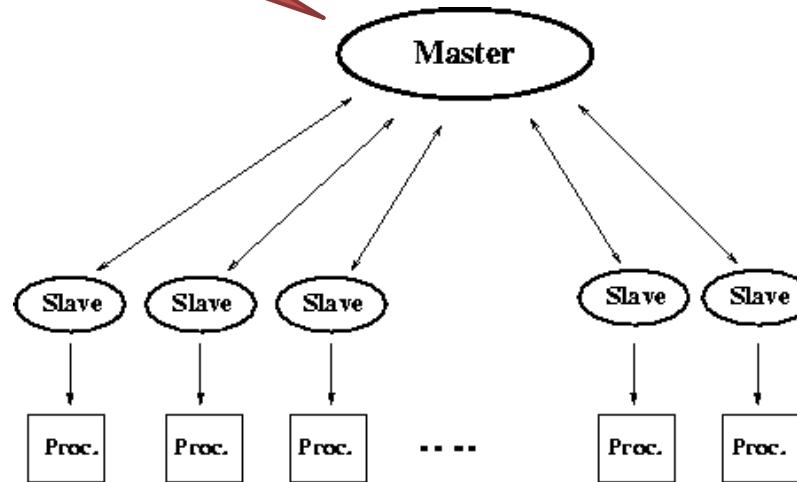


# Problems?

Potential bottleneck!

## Global sum

$$8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95$$



| Core   | 0 | 1  | 2 | 3  | 4 | 5  | 6  | 7  |
|--------|---|----|---|----|---|----|----|----|
| my_sum | 8 | 19 | 7 | 15 | 7 | 13 | 12 | 14 |

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9



# Better parallel algorithm

---

- Don't make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1's result.
- Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.



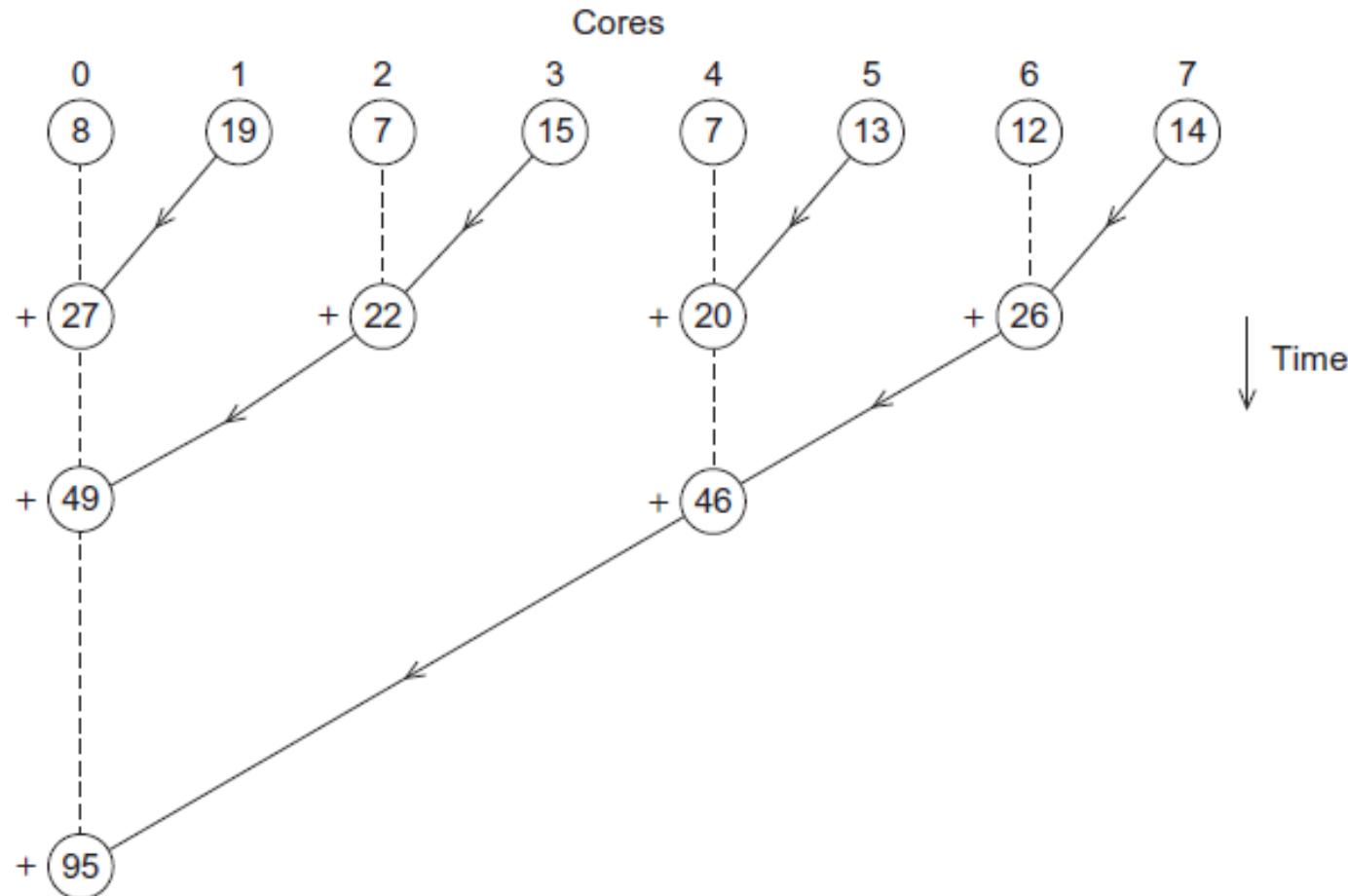
# Better parallel algorithm (cont.)

---

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.
- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.



# Multiple cores forming a global sum



# Analysis

---

- In the first example, the master core performs 7 receives and 7 additions.
- In the second example, the master core performs 3 receives and 3 additions.
- The improvement is more than a factor of 2!

# Analysis (cont.)

---

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
  - The first example would require the master to perform 999 receives and 999 additions.
  - The second example would only require 10 receives and 10 additions.  
 $(2^{10}=1024)$
- That's an improvement of almost a factor of 100!



# How do we write parallel programs?

---

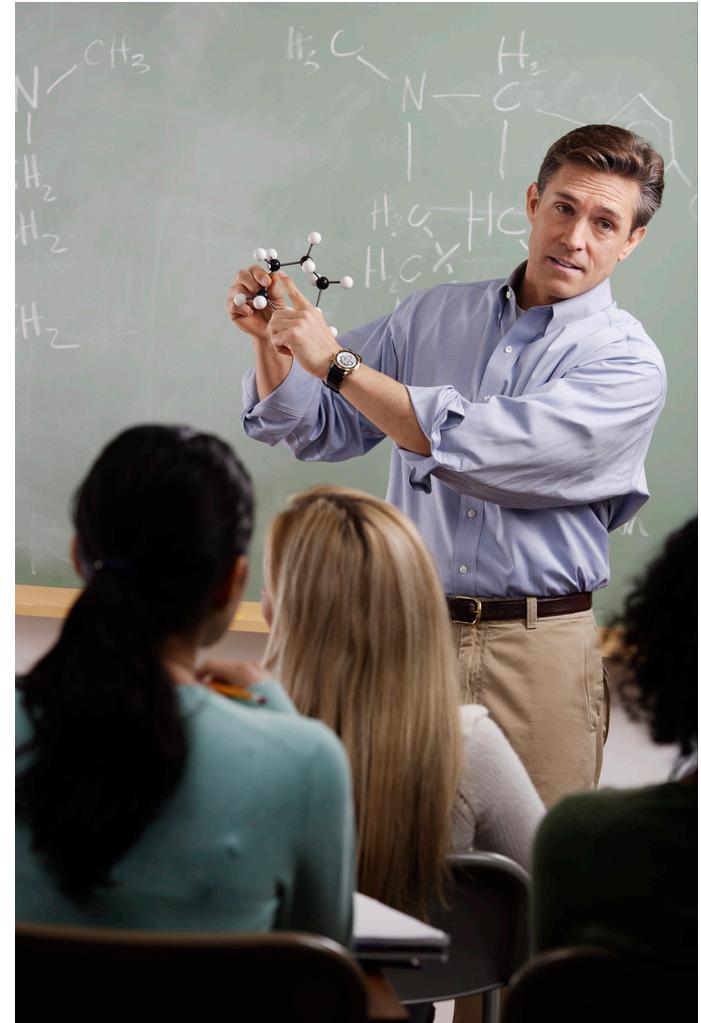
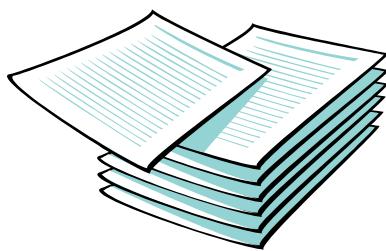
- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.
- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it's part of the data.



# Professor P

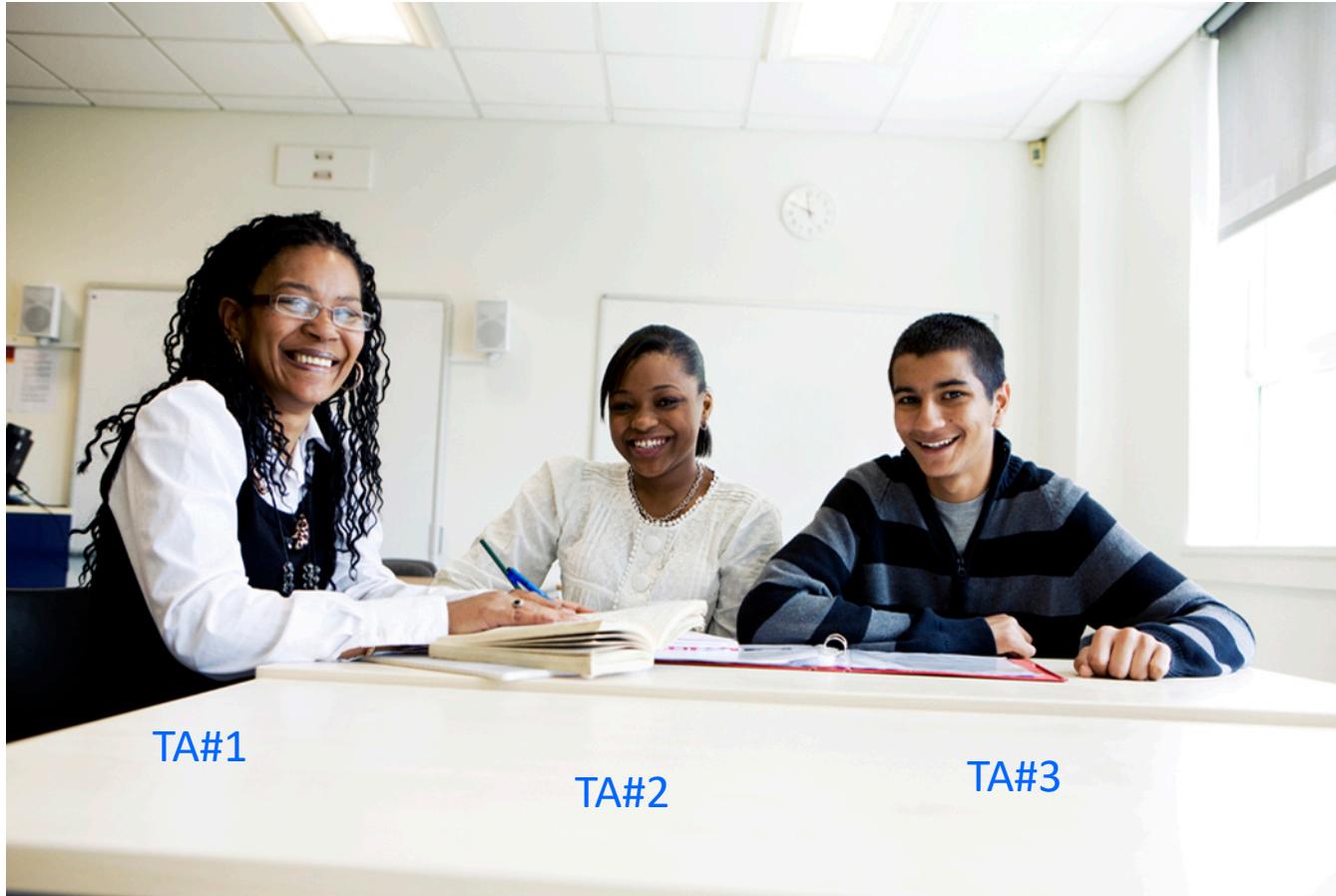
15 questions

300 exams



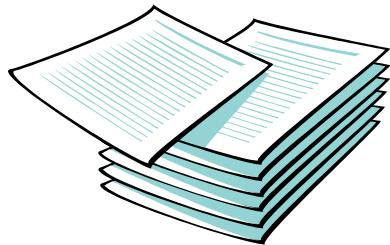
# Professor P's grading assistants

---



# Division of work – data parallelism

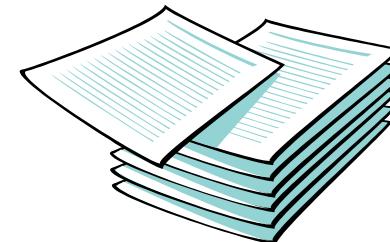
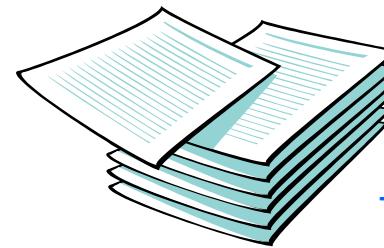
TA#1



100 exams

100 exams

TA#3



TA#2

100 exams



# Division of work – task parallelism

---

TA#1



Questions 1 - 5



TA#3

Questions 11 - 15



TA#2

Questions 6 - 10



# Division of work – data parallelism

---



```
sum = 0;  
for (i = 0; i < n; i++) {  
    x = Compute_next_value(. . .);  
    sum += x;  
}
```

Same task (program), different data

# Division of work – task parallelism

---

```
if (I'm the master core) {  
    sum = my_x;  
    for each core other than myself {  
        receive value from core;  
        sum += value;  
    }  
} else {  
    send my_x to the master;  
}
```

different tasks (program): program A and B could be totally different logic

different data

## Tasks

- 1) Receiving
- 2) Addition



# Coordination

---

- Cores usually need to coordinate their work.
- **Communication** – one or more cores send their current partial sums to another core.
- **Load balancing** – share the work evenly among the cores so that one is not heavily loaded.
- **Synchronization** – because each core works at its own pace, make sure cores do not get too far ahead of the rest.



# What we'll be doing

---

- Using the C language.
- Using three different extensions to C.
  - Message-Passing Interface (MPI)
  - Posix Threads (Pthreads)
  - OpenMP
- Write programs that are explicitly parallel.



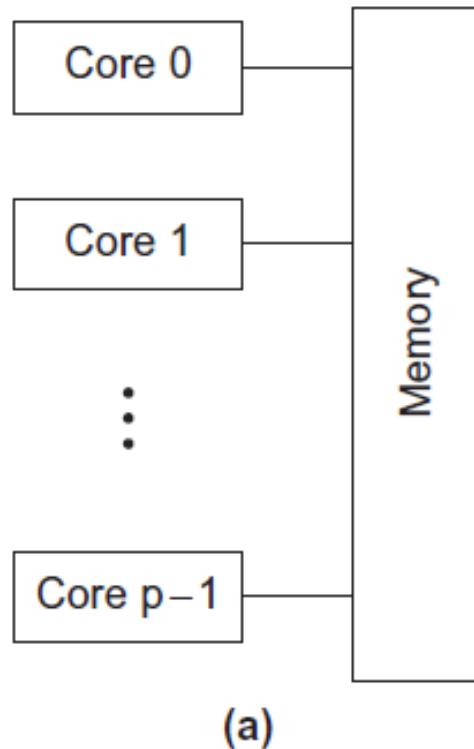
# Type of parallel systems

---

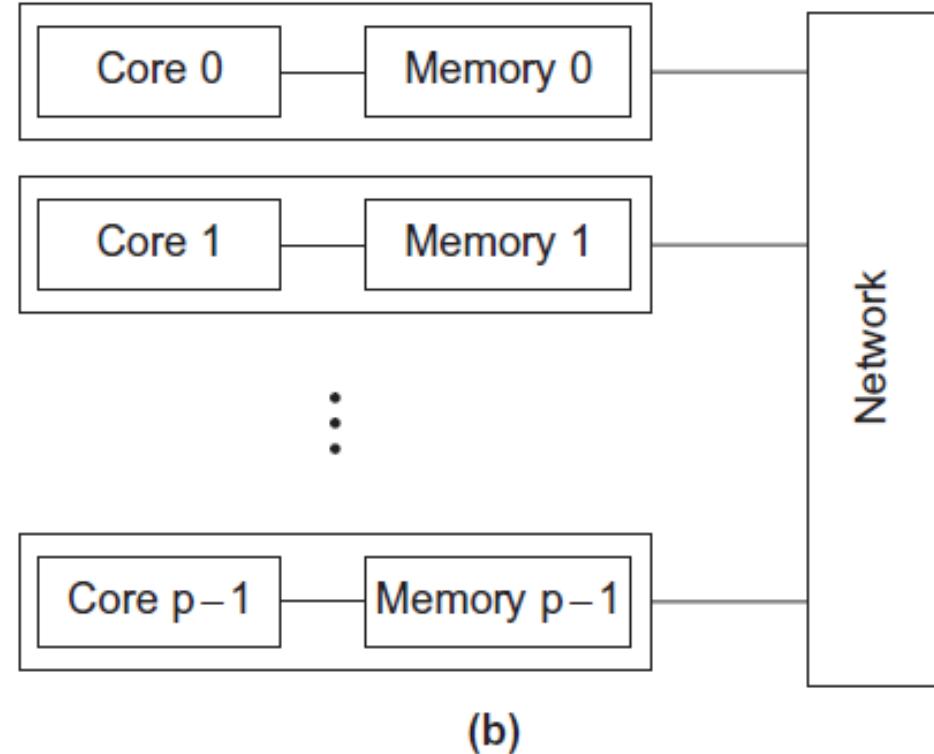
- Shared-memory
  - The cores can share access to the computer's memory.
  - Coordinate the cores by having them examine and update shared memory locations.
- Distributed-memory
  - Each core has its own, private memory.
  - The cores must communicate explicitly by sending messages across a network.



# Type of parallel systems



Shared-memory



Distributed-memory



# Terminology

---

- **Concurrent computing** – a program is one in which multiple tasks can be in progress at any instant.
- **Parallel computing** – a program is one in which multiple tasks cooperate closely to solve a problem
- **Distributed computing** – a program may need to cooperate with other programs to solve a problem.



# Conclusion

---

- The laws of physics have brought us to the doorstep of multicore technology.
- Serial programs typically don't benefit from multiple cores.
- Automatic parallel program generation from serial program code isn't the most efficient approach to get high performance from multicore computers.



# Conclusion (cont.)

---

- Learning to write parallel programs involves learning how to coordinate the cores.
- Parallel programs are usually very complex and therefore, require sound program techniques and development.

