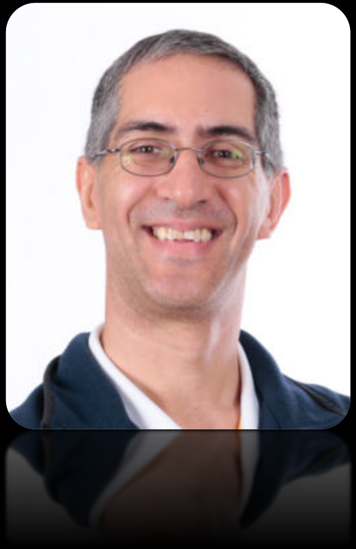
CS61C



UC Berkeley
Teaching Professor
Dan Garcia

Great Ideas
in
Computer Architecture
(a.k.a. Machine Structures)

UC Berkeley
Professor
Bora Nikolić

Thread-Level Parallelism I

Berkeley
UNIVERSITY OF CALIFORNIA

cs61c.org

Parallel Computer Architectures

Improving Performance

1. Increase clock rate f_s
   - Reached practical maximum for today's technology
   - < 5GHz for general purpose computers
2. Lower CPI (cycles per instruction)
   - SIMD, "instruction level parallelism"

Today's lecture

3. Perform multiple tasks simultaneously
   - Multiple CPUs, each executing different program
   - Tasks may be related
     - E.g. each CPU performs part of a big matrix multiplication
   - or unrelated
     - E.g. distribute different web http requests over different computers
     - E.g. run pptx (view lecture slides) and browser (youtube) simultaneously
4. Do all of the above:
   - High f_s, SIMD, multiple parallel tasks

New-School Machine Structures

Software

Parallel Requests

Assigned to computer

e.g., Search “Cats”

Parallel Threads

Assigned to core e.g., Lookup, Ads

Parallel Instructions

>1 instruction @ one time

e.g., 5 pipelined instructions

Parallel Data

>1 data item @ one time

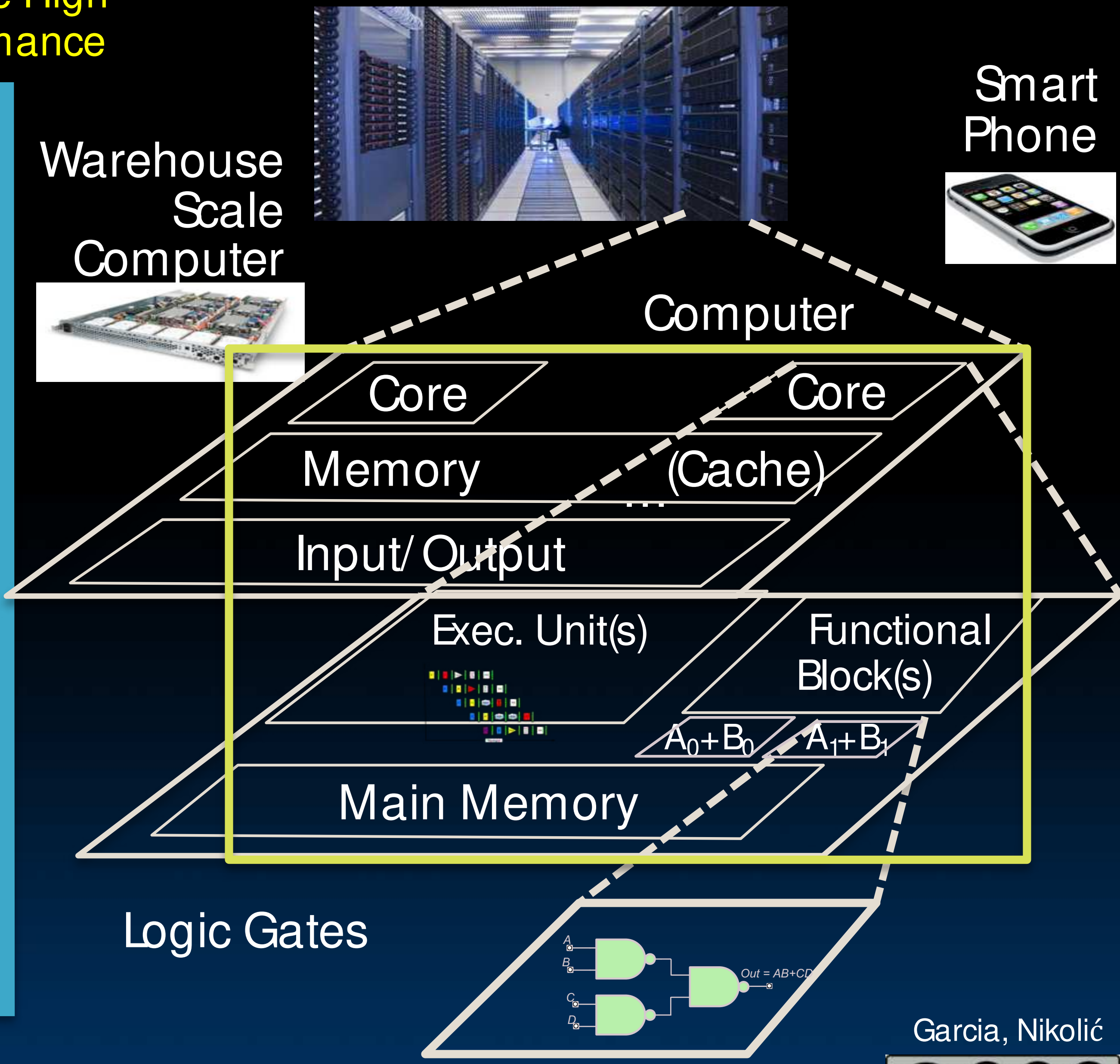
e.g., Add of 4 pairs of words

Hardware descriptions

All gates work in parallel at same time

Harness Parallelism & Achieve High Performance

Hardware



Warehouse Scale Computer

Smart Phone

Computer

Core

Core

Memory

(Cache)

...

Input/Output

Exec. Unit(s)

Functional Block(s)

A_0+B_0

A_1+B_1

Main Memory

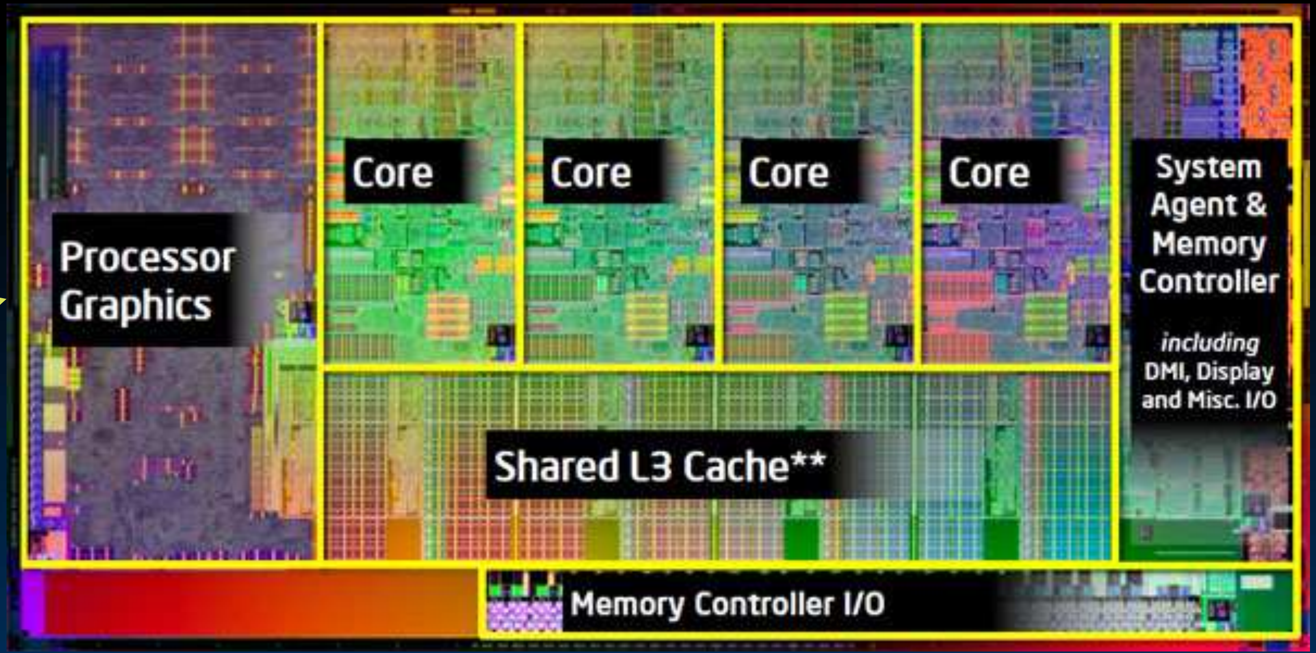
Logic Gates

Parallel Computer Architectures

Several separate computers,
some means for communication
(e.g., Ethernet)

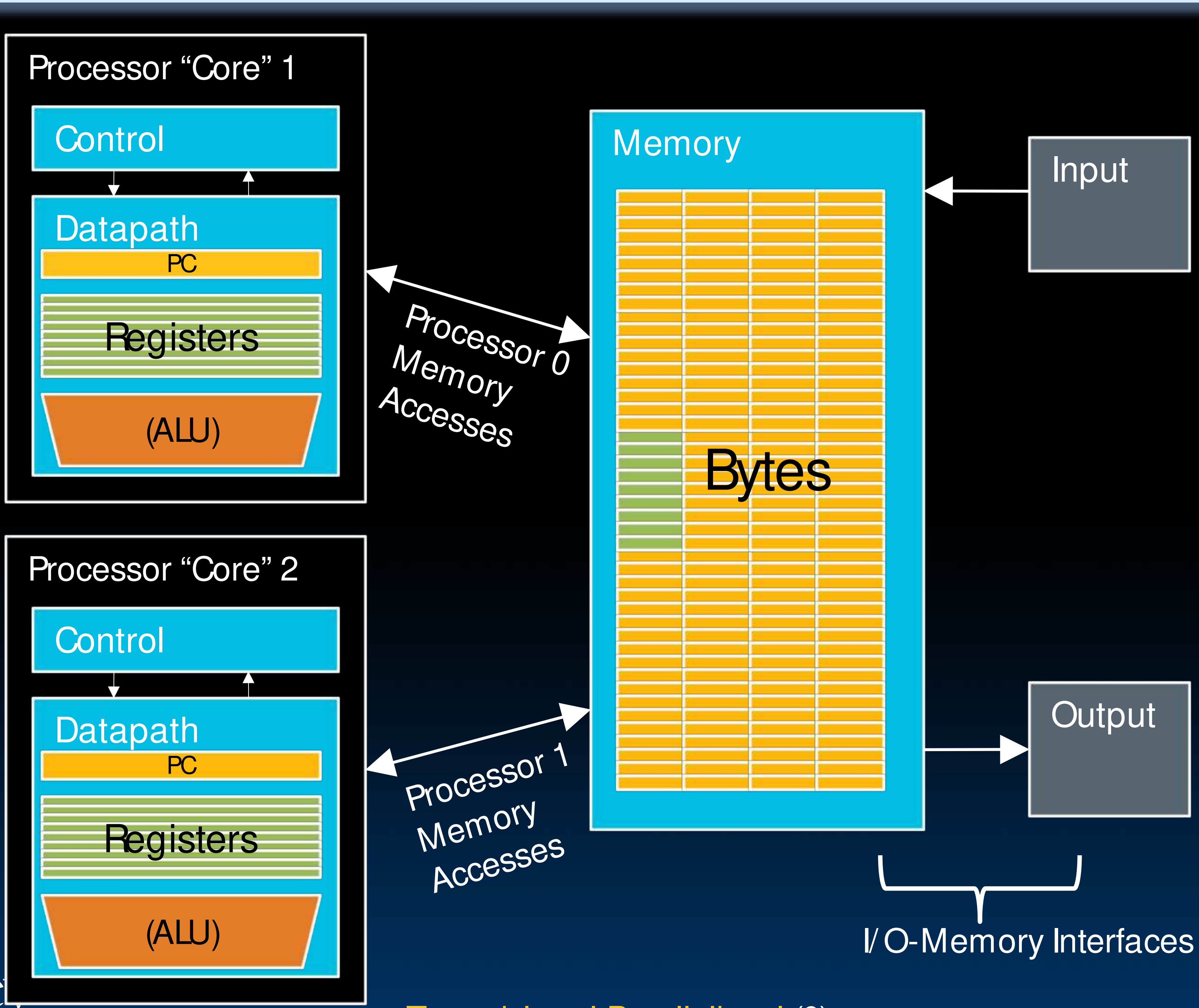
Massive array of computers,
fast communication between processors

GPU “graphics processing unit”



Multi-core CPU:
1 datapath in single chip
share L3 cache, memory, peripherals
Example: Hive machines

Example: CPU with Two Cores



Multiprocessor Execution Model

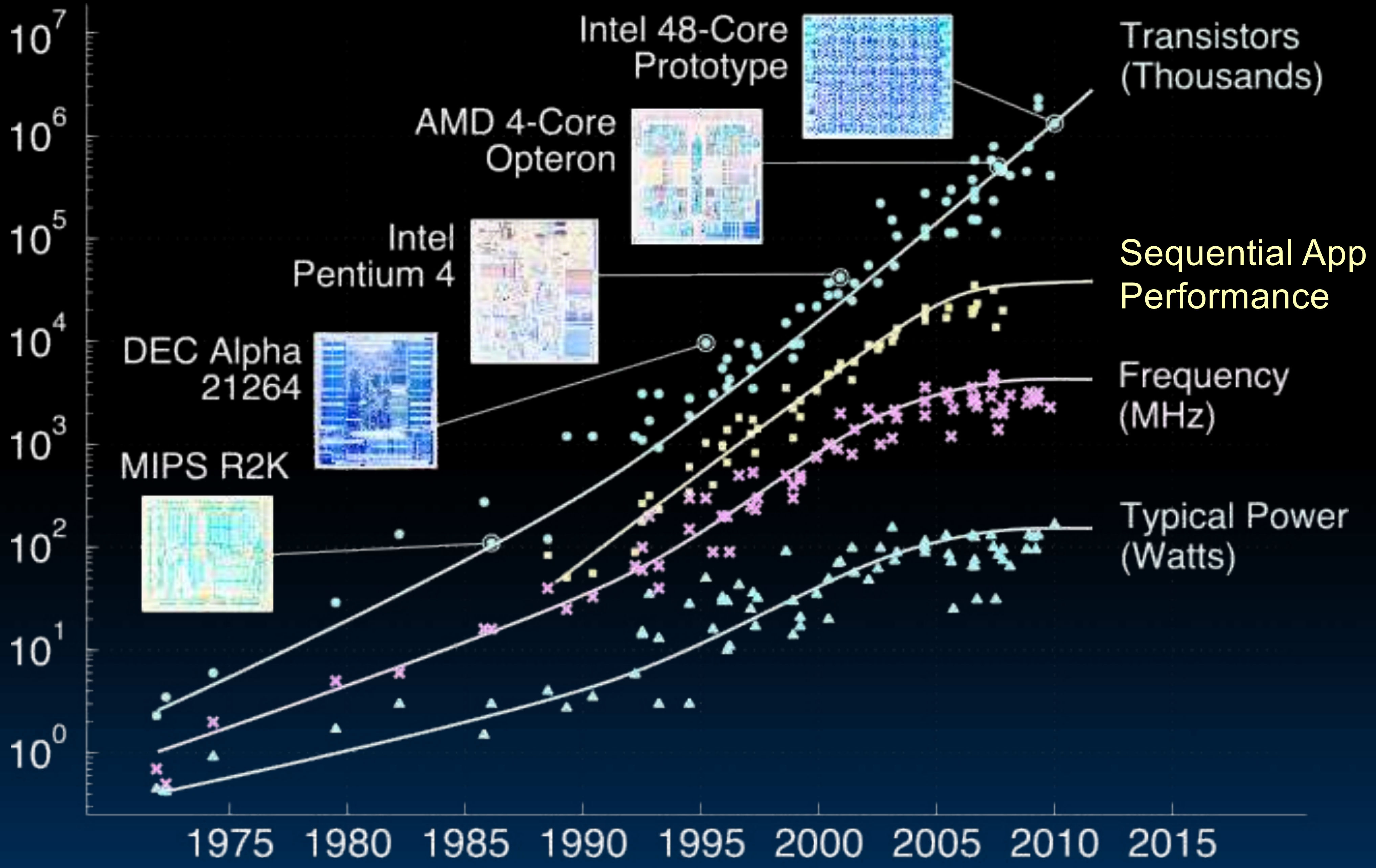
- Each processor (core) executes its own instructions
- *Separate* resources (not shared)
  - Datapath (PC, registers, ALU)
  - Highest level caches (e.g., 1st and 2nd)
- *Shared* resources
  - Memory (DRAM)
  - Often 3rd level cache
    - Often on same silicon chip
    - But not a requirement
- Nomenclature
  - "Multiprocessor Microprocessor"
  - Multicore processor
    - E.g., four core CPU (central processing unit)
    - Executes four different instruction streams simultaneously

☑

Garcia, Nikolić

Multicore

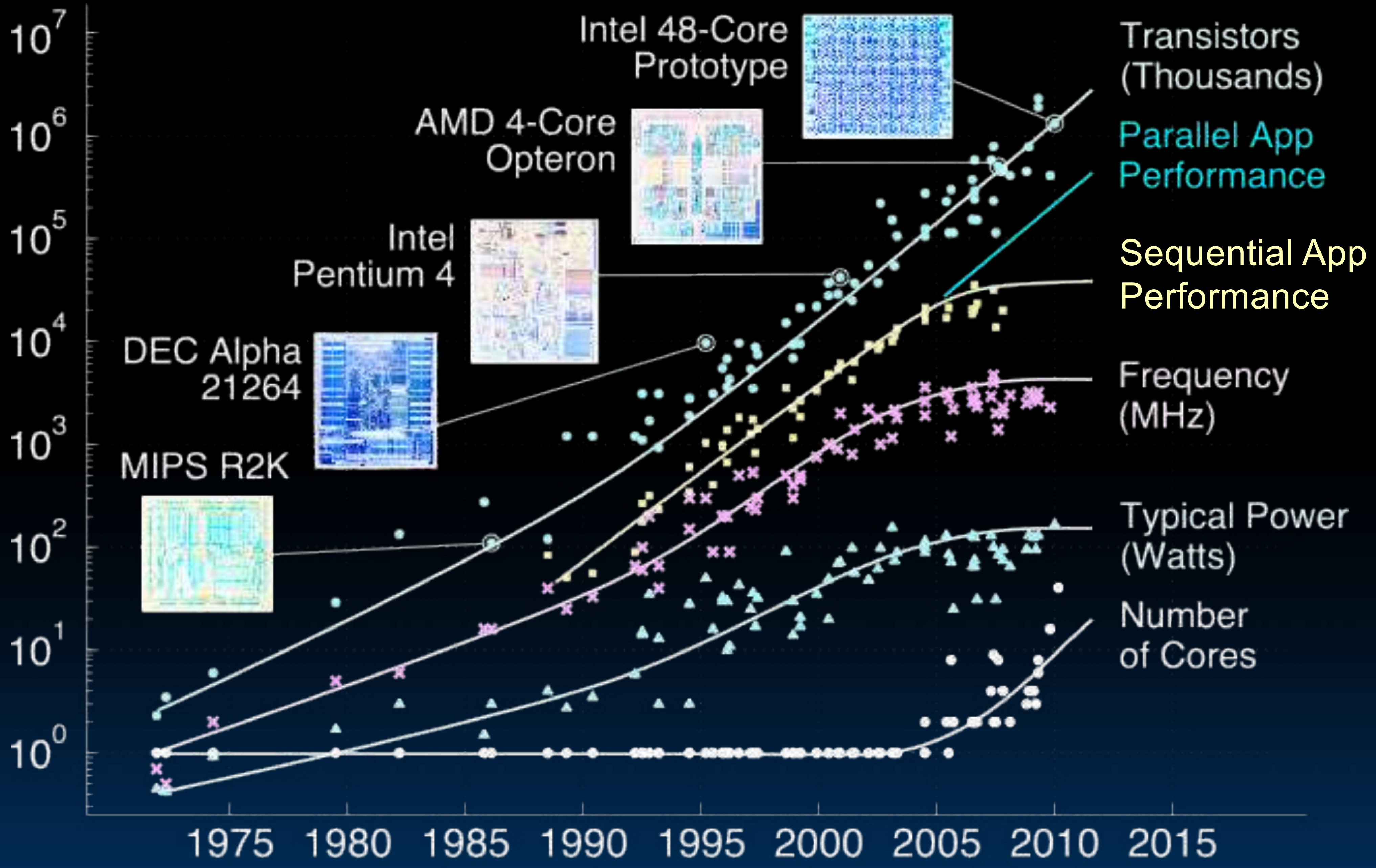
Transition to Multicore



Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

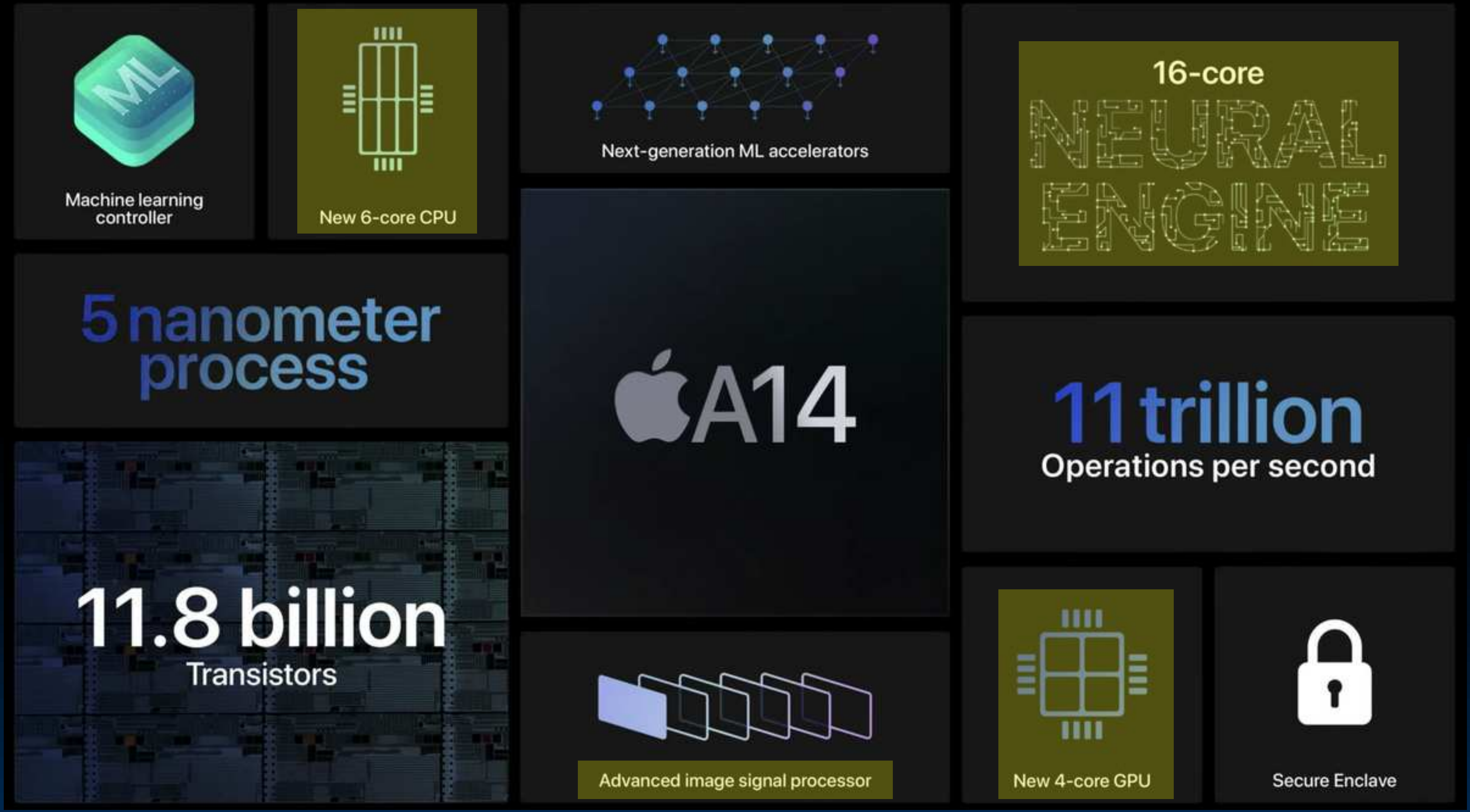
Garcia, Nikolić

Transition to Multicore



Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

Apple A14 Chip (in their latest phones)



Multiprocessor Execution Model

- Shared memory
  - Each "core" has access to the entire memory in the processor
  - Special hardware keeps caches consistent (next lecture!)
  - Advantages:
    - Simplifies communication in program via shared variables
  - Drawbacks:
    - Does not scale well:
      - "Slow" memory shared by many "customers" (cores)
      - May become bottleneck (Amdahl's Law)
- Two ways to use a multiprocessor:
  - Job-level parallelism
    - Processors work on unrelated problems
    - No communication between programs
  - Partition work of single task between several cores
    - E.g., each performs part of large matrix multiplication

Parallel Processing

- It's difficult!
- It's inevitable
  - Only path to increase performance
  - Only path to lower energy consumption (improve battery life)
- In mobile systems (e.g., smart phones, tablets)
  - Multiple cores
  - Dedicated processors, e.g.,
    - Motion processor, image processor, neural processor in iPhone 8 + X
    - GPU (graphics processing unit)
- Warehouse-scale computers (next week!)
  - Multiple "nodes"
    - "Boxes" with several CPUs, disks per box
  - MIMD (multi-core) and SIMD (e.g. AVX) in each node

Potential Parallel Performance

(assuming software can use it)

Year	Cores	SIMD bits / Core	Core * SIMD bits	Total, e.g. FLOPs/ Cycle
2003	2	128	256	4
2005	4	128	512	8
2007	6	128	768	12
2009	8	128	1024	16
2011	10	256	2560	40
2013	12	256	3072	48
2015	14	512	7168	112
2017	16	512	8192	128
2019	18	1024	18432	288
2021	20	1024	20480	320

MIMD +2/ 2yrs

SIMD 2X/ 4yrs

MIMD & SIMD

12 years

2.5X

8X

20X

20 x in 12 years

$20^{1/12}$ = 1.28 x → 28% per year or 2x every 3 years!

IF (!) we can use it

☑

Garcia, Nikolić

Threads

Programs Running on a typical Computer

unix% ps -x

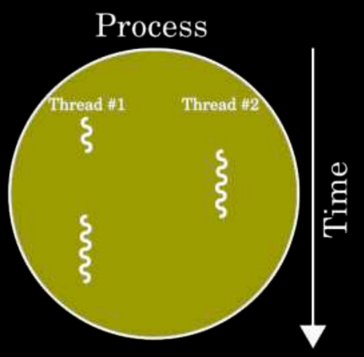
```
  PID TTY           TIME CMD
  220 ??         0:04.34 /usr/libexec/UserEventAgent (Aqua)
  222 ??         0:10.60 /usr/sbin/distnoted agent
  224 ??         0:09.11 /usr/sbin/cfprefsd agent
  229 ??         0:04.71 /usr/sbin/usernoted
  230 ??         0:02.35 /usr/libexec/nsurlsessiond
  232 ??         0:28.68 /System/Library/PrivateFrameworks/CalendarAgent.framework/Executables/CalendarAgent
  234 ??         0:04.36 /System/Library/PrivateFrameworks/GameCenterFoundation.framework/Versions/A/gamed
  235 ??         0:01.90 /System/Library/CoreServices/cloudphotosd.app/Contents/MacOS/cloudphotosd
  236 ??         0:49.72 /usr/libexec/secinitd
  239 ??         0:01.66 /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd
  240 ??         0:12.68 /System/Library/Frameworks/Accounts.framework/Versions/A/Support/accountsd
  241 ??         0:09.56 /usr/libexec/SafariCloudHistoryPushAgent
  242 ??         0:00.27 /System/Library/PrivateFrameworks/CallHistory.framework/Support/CallHistorySyncHelper
  243 ??         0:00.74 /System/Library/CoreServices/mapspushd
  244 ??         0:00.79 /usr/libexec/fmfd
  246 ??         0:00.09 /System/Library/PrivateFrameworks/AskPermission.framework/Versions/A/Resources/askpermissiond
  248 ??         0:01.03 /System/Library/PrivateFrameworks/CloudDocsDaemon.framework/Versions/A/Support/bird
  249 ??         0:02.50 /System/Library/PrivateFrameworks/IDS.framework/identityservicesd.app/Contents/MacOS/identityservicesd
  250 ??         0:04.81 /usr/libexec/secd
  254 ??         0:24.01 /System/Library/PrivateFrameworks/CloudKitDaemon.framework/Support/cloudd
  258 ??         0:04.73 /System/Library/PrivateFrameworks/TelephonyUtilities.framework/callservicesd
  267 ??         0:02.15 /System/Library/CoreServices/AirPlayUIAgent.app/Contents/MacOS/AirPlayUIAgent --launchd
  271 ??         0:03.91 /usr/libexec/nsurlstoraged
  274 ??         0:00.90 /System/Library/PrivateFrameworks/CommerceKit.framework/Versions/A/Resources/storeaccountd
  282 ??         0:00.09 /usr/sbin/pboard
  283 ??         0:00.90 /System/Library/PrivateFrameworks/InternetAccounts.framework/Versions/A/XPCServices/
com.apple.internetaccounts.xpc/Contents/MacOS/com.apple.internetaccounts
  285 ??         0:04.72 /System/Library/Frameworks/ApplicationServices.framework/Frameworks/ATS.framework/Support/fontd
  291 ??         0:00.25 /System/Library/Frameworks/Security.framework/Versions/A/Resources/CloudKeychainProxy.bundle/
Contents/MacOS/CloudKeychainProxy
  292 ??         0:09.54 /System/Library/CoreServices/CoreServicesUIAgent.app/Contents/MacOS/CoreServicesUIAgent
  293 ??         0:00.29 /System/Library/PrivateFrameworks/CloudPhotoServices.framework/Versions/A/Frameworks/
CloudPhotoServicesConfiguration.framework/Versions/A/XPCServices/com.apple.CloudPhotosConfiguration.xpc/Contents/MacOS/
com.apple.CloudPhotosConfiguration
  297 ??         0:00.84 /System/Library/PrivateFrameworks/CloudServices.framework/Resources/com.apple.sbd
  302 ??         0:26.11 /System/Library/CoreServices/Dock.app/Contents/MacOS/Dock
  303 ??         0:09.55 /System/Library/CoreServices/SystemUIServer.app/Contents/MacOS/SystemUIServer
```

…156 total at this moment… How does my laptop do this?
Imagine doing 156 assignments all at the same time!

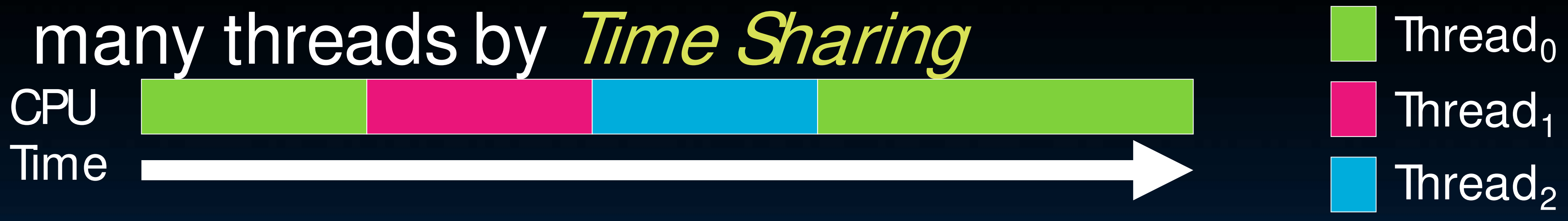
Garcia, Nikolić

Threads (1)

- A *Thread* stands for "thread of execution", is a single stream of instructions
  - A program / process can split, or fork itself into separate threads, which can (in theory) execute simultaneously.
  - An easy way to describe/think about parallelism



- With a single core, a single CPU can execute many threads by *Time Sharing*


CPU

Time

🟩 Thread$_0$

🟥 Thread$_1$

🟦 Thread$_2$

Threads (2)

- Sequential flow of instructions that performs some task
  - Up to now we just called this a "program"
- Each thread has:
  - Dedicated PC (program counter)
  - Separate registers
  - Accesses the shared memory
- Each physical core provides one (or more)
  - *Hardware* threads that actively execute instructions
  - Each executes one "*hardware* thread"
- Operating system multiplexes multiple
  - *Software* threads onto the available hardware threads
  - All threads except those mapped to hardware threads are waiting

Thoughts about Threads

“Although threads seem to be a small step from sequential computation, in fact, they represent a huge step. They discard the most essential and appealing properties of sequential computation: understandability, predictability, and determinism. Threads, as a model of computation, are wildly non-deterministic, and the job of the programmer becomes one of pruning that nondeterminism.”

— The Problem with Threads,
Edward A. Lee, UC Berkeley, 2006

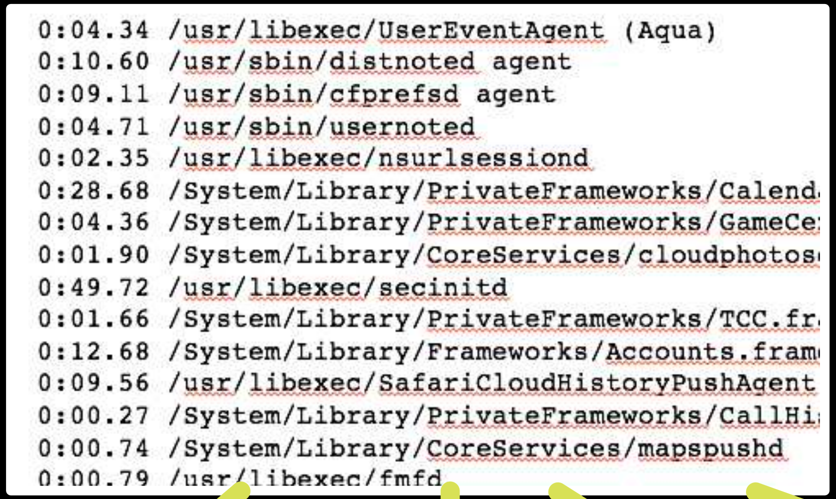
Garcia, Nikolić

Operating System Threads

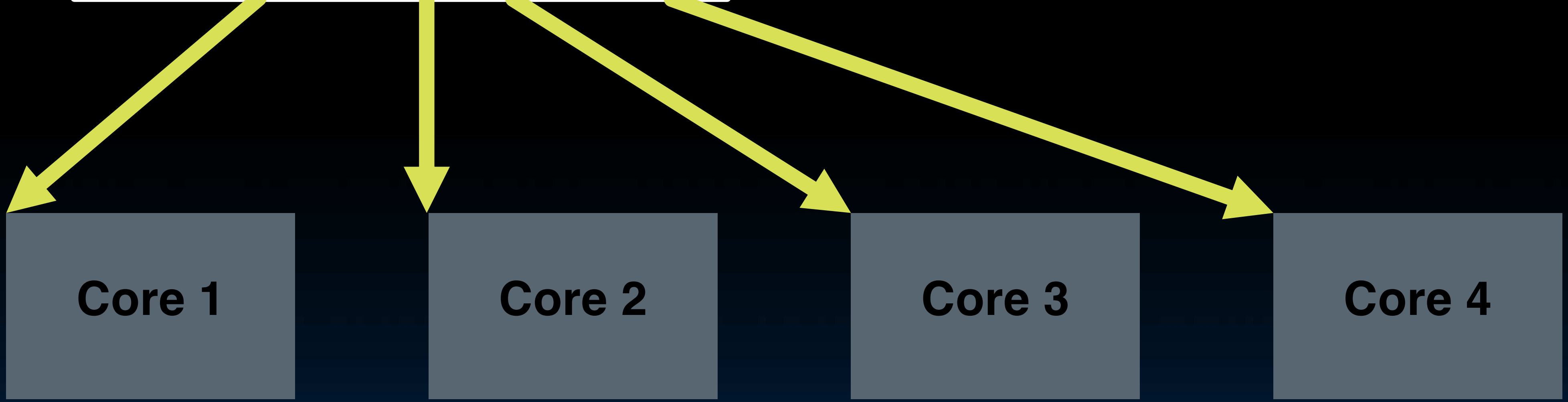
Give illusion of many “simultaneously” active threads

1. Multiplex software threads onto hardware threads:
   a) Switch out blocked threads (e.g., cache miss, user input, network access)
   b) Timer (e.g., switch active thread every 1 ms)
2. Remove a software thread from a hardware thread by
   a) Interrupting its execution
   b) Saving its registers and PC to memory
3. Start executing a different software thread by
   a) Loading its previously saved registers into a hardware thread’s registers
   b) Jumping to its saved PC

Example: Four Cores



```
0:04.34 /usr/libexec/UserEventAgent (Aqua)
0:10.60 /usr/sbin/distnoted agent
0:09.11 /usr/sbin/cfprefsd agent
0:04.71 /usr/sbin/usernoted
0:02.35 /usr/libexec/nsurlsessiond
0:28.68 /System/Library/PrivateFrameworks/Calend
0:04.36 /System/Library/PrivateFrameworks/GameCe
0:01.90 /System/Library/CoreServices/cloudphotos
0:49.72 /usr/libexec/secinitd
0:01.66 /System/Library/PrivateFrameworks/TCC.fr
0:12.68 /System/Library/Frameworks/Accounts.fram
0:09.56 /usr/libexec/SafariCloudHistoryPushAgent
0:00.27 /System/Library/PrivateFrameworks/CallHi
0:00.74 /System/Library/CoreServices/mapspushd
0:00.79 /usr/libexec/fmfd
```



Thread pool:
List of threads competing for processor

OS maps threads to cores and schedules logical (software) threads

Each "Core" actively runs one instruction stream at a time

☑

Garcia, Nikolić

Multithreading

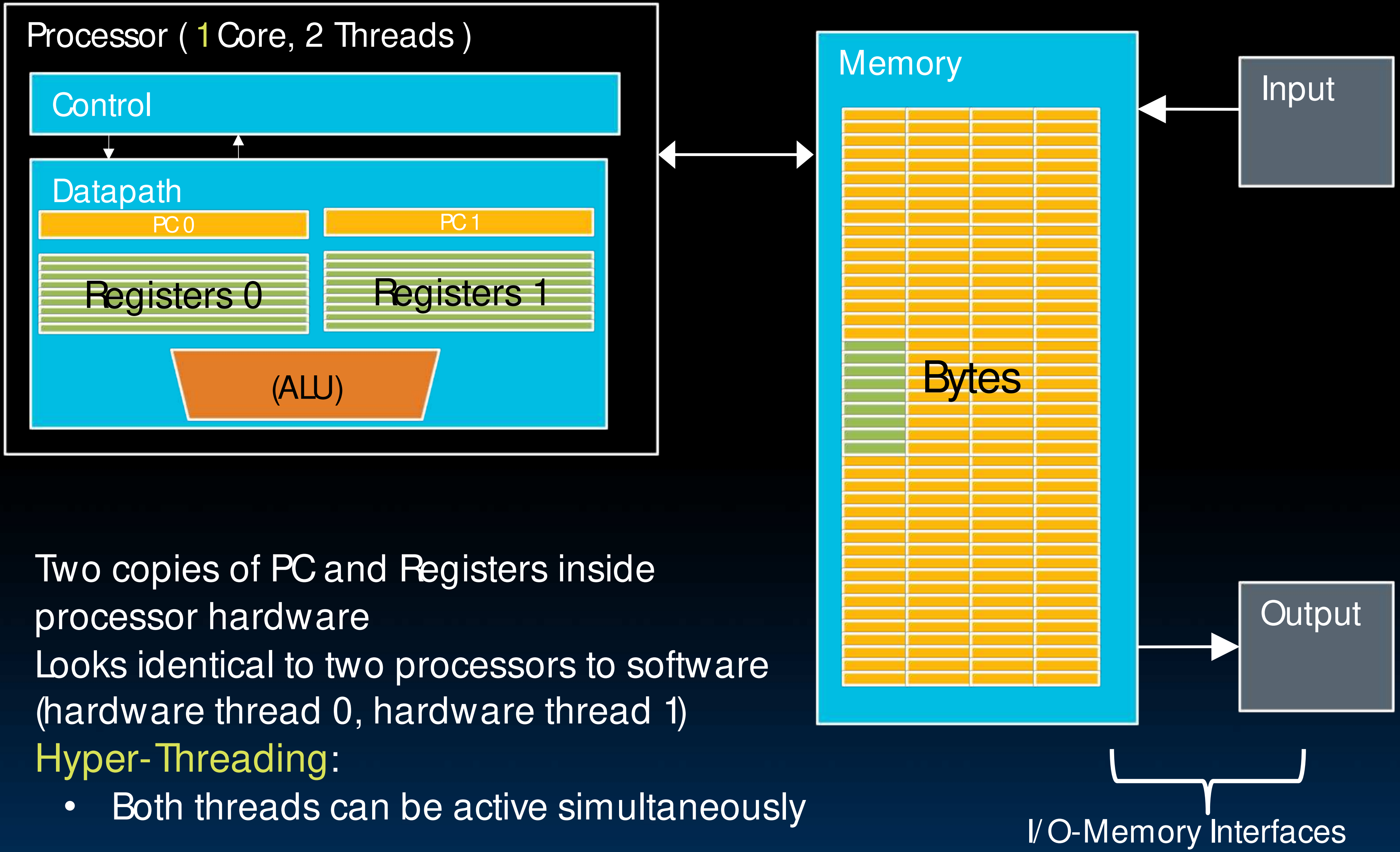
Multithreading

- Typical scenario:
  - Active thread encounters cache miss
  - Active thread waits ~ 1000 cycles for data from DRAM

  → switch out and run different thread until data available
- Problem
  - Must save current thread state and load new thread state
    - PC, all registers (could be many, e.g. AVX)

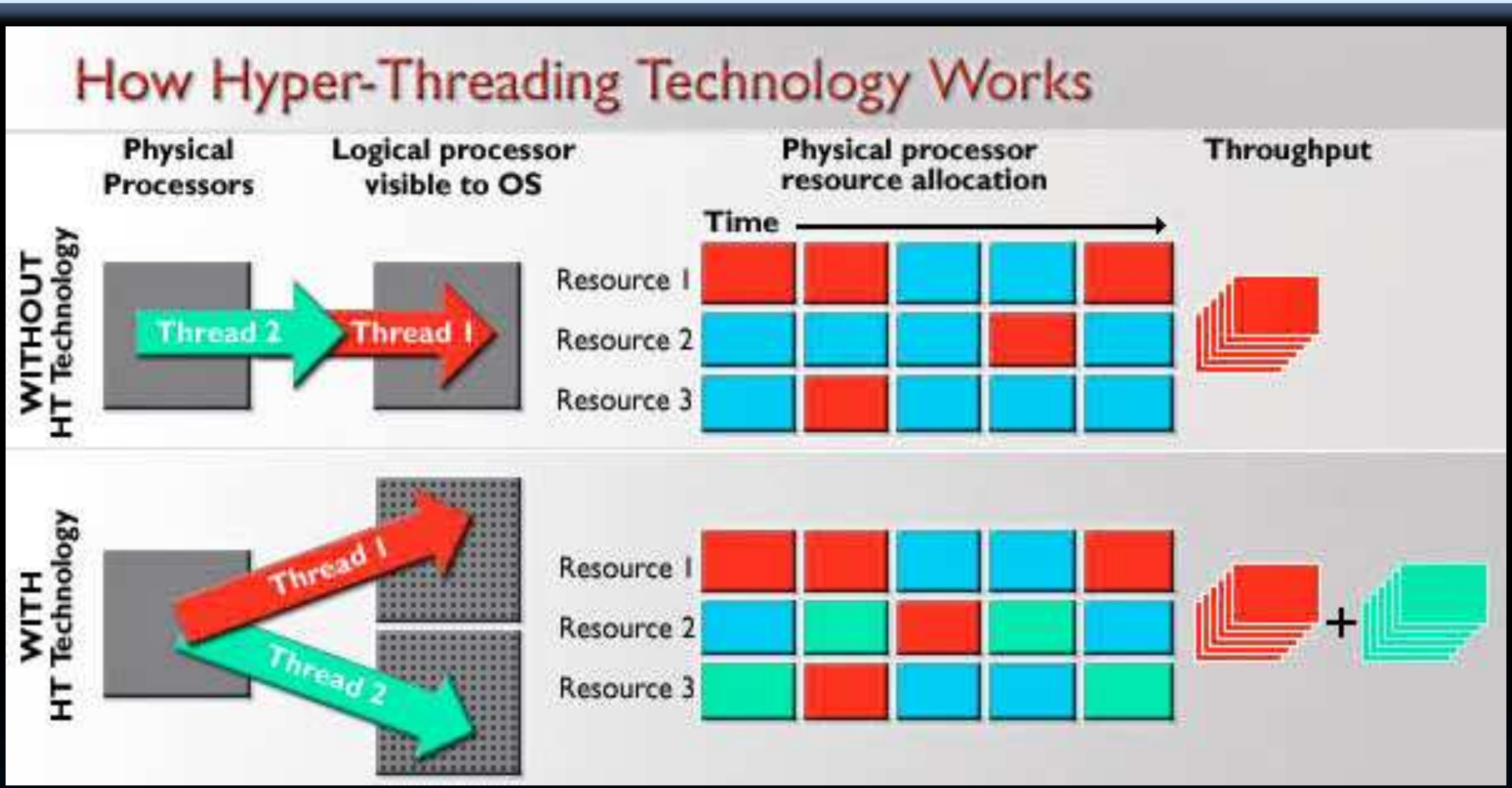
  → must perform switch in ≪ 1000 cycles
- Can hardware help?
  - Moore's Law: transistors are plenty

Hardware Assisted Software Multithreading



- Two copies of PC and Registers inside processor hardware
- Looks identical to two processors to software (hardware thread 0, hardware thread 1)
- Hyper-Threading:
  - Both threads can be active simultaneously

Hyper-Threading



- Simultaneous Multithreading (HT): Logical CPUs > Physical CPUs
  - Run multiple threads at the same time per core
  - Each thread has own architectural state (PC, Registers, etc.)
  - Share resources (cache, instruction unit, execution units)
  - See http://dada.cs.washington.edu/smt/

Multithreading

- Logical threads
  - ≈ 1% more hardware
  - ≈ 10% (?) better performance
    - Separate registers
    - Share datapath, ALU(s), caches
- Multicore
  - => Duplicate Processors
  - ≈ 50% more hardware
  - ≈ 2X better performance?
- Modern machines do both
  - Multiple cores with multiple threads per core

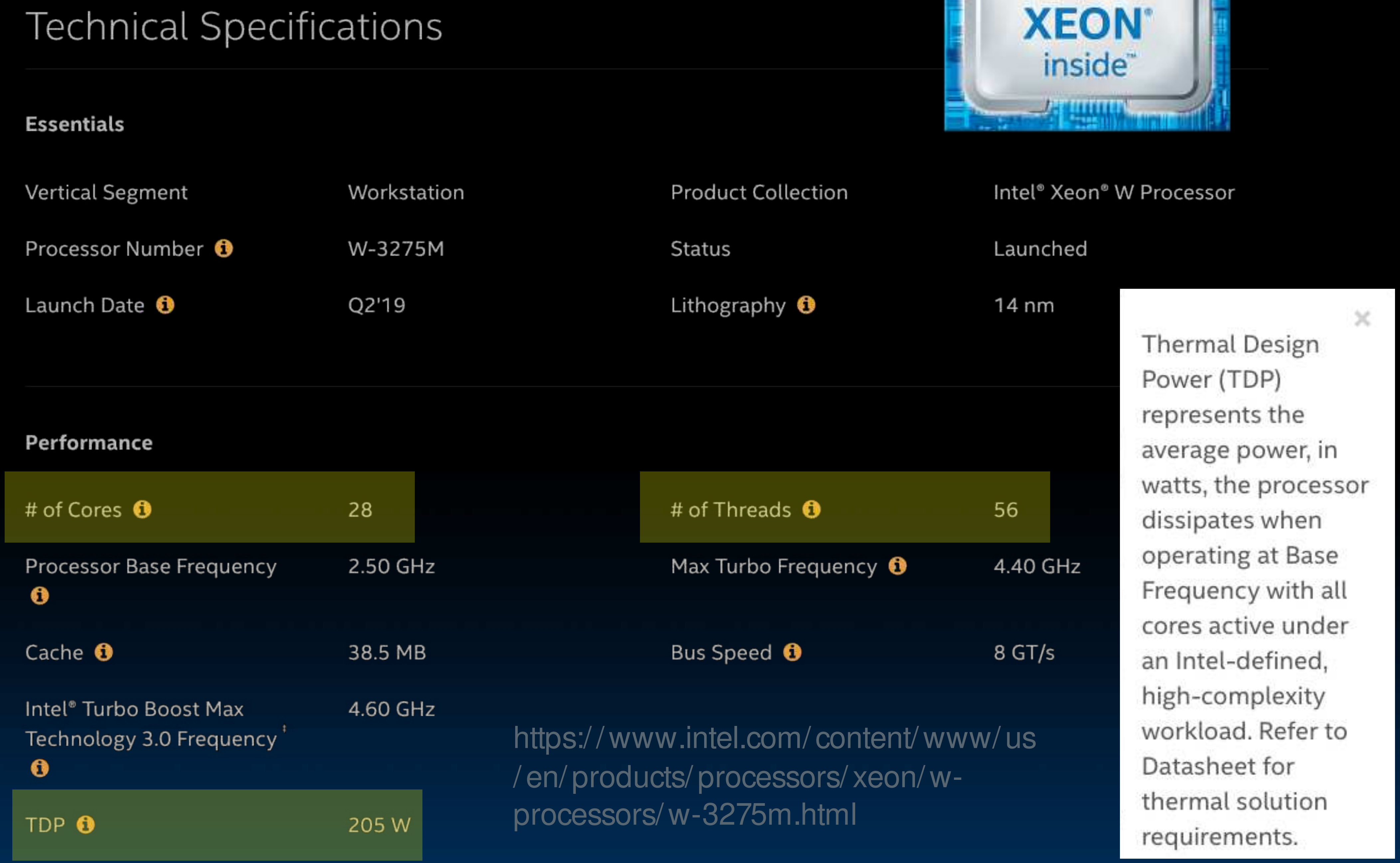
Dan’s Laptop (cf Activity Monitor)

```
$ sysctl hw

hw.physicalcpu:   4
hw.logicalcpu:    8
```

- 4 Cores
- 8 Threads total

Intel® Xeon® W-3275M Processor



Technical Specifications

Essentials

Vertical Segment	Workstation	Product Collection	Intel® Xeon® W Processor
Processor Number	W-3275M	Status	Launched
Launch Date	Q2'19	Lithography	14 nm

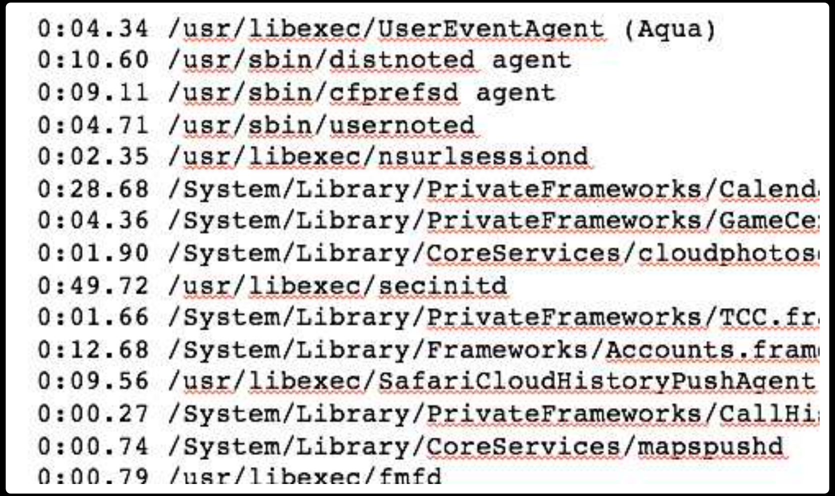
Performance

# of Cores	28	# of Threads	56
Processor Base Frequency	2.50 GHz	Max Turbo Frequency	4.40 GHz
Cache	38.5 MB	Bus Speed	8 GT/s
Intel® Turbo Boost Max Technology 3.0 Frequency ‡	4.60 GHz		
TDP	205 W		

Thermal Design Power (TDP) represents the average power, in watts, the processor dissipates when operating at Base Frequency with all cores active under an Intel-defined, high-complexity workload. Refer to Datasheet for thermal solution requirements.

https://www.intel.com/content/www/us/en/products/processors/xeon/w-processors/w-3275m.html

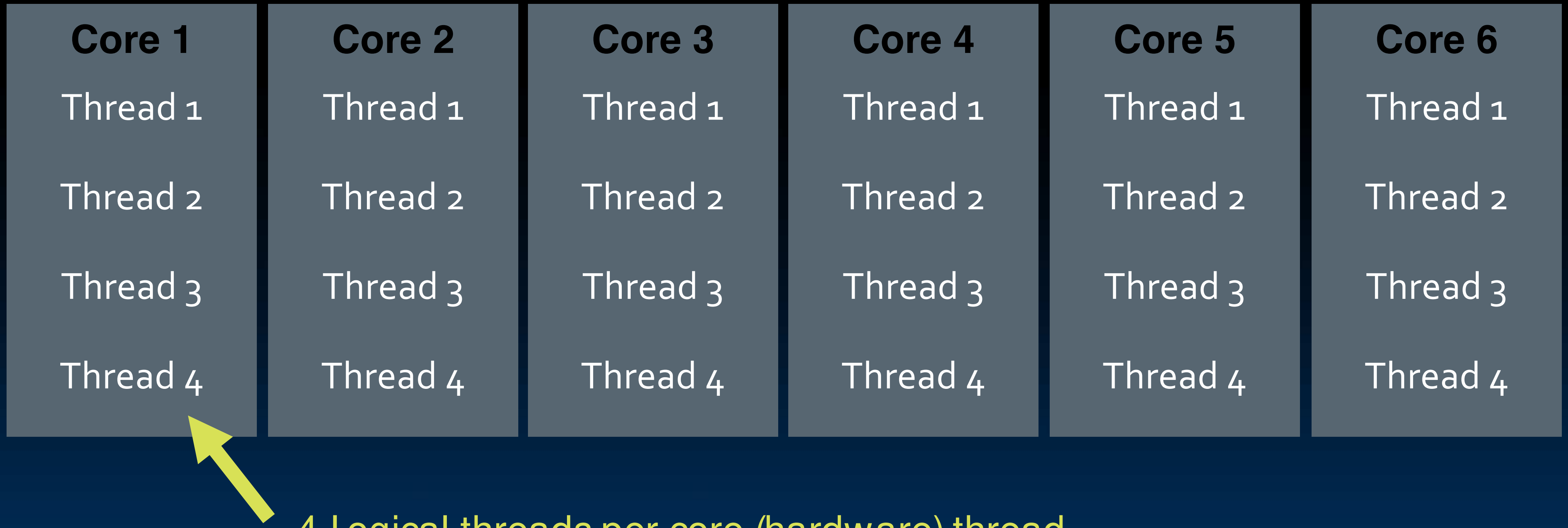
Example: 6 Cores, 24 Logical Threads



```
0:04.34 /usr/libexec/UserEventAgent (Aqua)
0:10.60 /usr/sbin/distnoted agent
0:09.11 /usr/sbin/cfprefsd agent
0:04.71 /usr/sbin/usernoted
0:02.35 /usr/libexec/nsurlsessiond
0:28.68 /System/Library/PrivateFrameworks/Calend
0:04.36 /System/Library/PrivateFrameworks/GameCe
0:01.90 /System/Library/CoreServices/cloudphotos
0:49.72 /usr/libexec/secinitd
0:01.66 /System/Library/PrivateFrameworks/TCC.fr
0:12.68 /System/Library/Frameworks/Accounts.fram
0:09.56 /usr/libexec/SafariCloudHistoryPushAgent
0:00.27 /System/Library/PrivateFrameworks/CallHi
0:00.74 /System/Library/CoreServices/mapspushd
0:00.79 /usr/libexec/fmfd
```

Thread pool:
List of threads competing for processor

OS maps threads to cores and schedules logical (software) threads



Core 1	Core 2	Core 3	Core 4	Core 5	Core 6
Thread 1	Thread 1	Thread 1	Thread 1	Thread 1	Thread 1
Thread 2	Thread 2	Thread 2	Thread 2	Thread 2	Thread 2
Thread 3	Thread 3	Thread 3	Thread 3	Thread 3	Thread 3
Thread 4	Thread 4	Thread 4	Thread 4	Thread 4	Thread 4

4 Logical threads per core (hardware) thread

Review: Definitions

- Thread Level Parallelism
  - *Thread:* sequence of instructions, with own program counter and processor state (e.g., register file)
  - *Multicore:*
    - Physical CPU: One thread (at a time) per CPU, in software OS switches threads typically in response to I/O events like disk read/write
    - Logical CPU: Fine-grain thread switching, in hardware, when thread blocks due to cache miss/memory access
    - Hyper-Threading aka Simultaneous Multithreading (SMT): Exploit superscalar architecture to launch instructions from different threads at the same time!

Garcia, Nikolić

And, in Conclusion, …

- Sequential software execution speed is limited
  - Clock rates flat or declining
- Parallelism the only path to higher performance
  - SIMD: instruction level parallelism
    - Implemented in all high perf. CPUs today (x86, ARM, …)
    - Partially supported by compilers
    - 2X width every 3-4 years
  - MIMD: thread level parallelism
    - Multicore processors
    - Supported by Operating Systems (OS)
    - Requires programmer intervention to exploit at single program level (we see later)
    - Add 2 cores every 2 years (2, 4, 6, 8, 10, …)
      - Intel Xeon W-3275: 28 Cores, 56 Threads
  - SIMD & MIMD for maximum performance
- Key challenge: craft parallel programs with high performance on multiprocessors as # of processors increase – i.e., that scale
  - Scheduling, load balancing, time for synchronization, overhead communication

☑

Berkeley
UNIVERSITY OF CALIFORNIA

Garcia, Nikolić