Threads

Programs Running on a typical Computer

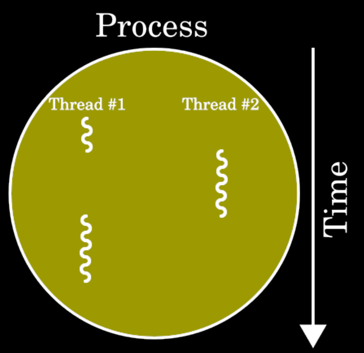
unix% ps -x

```
  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
```

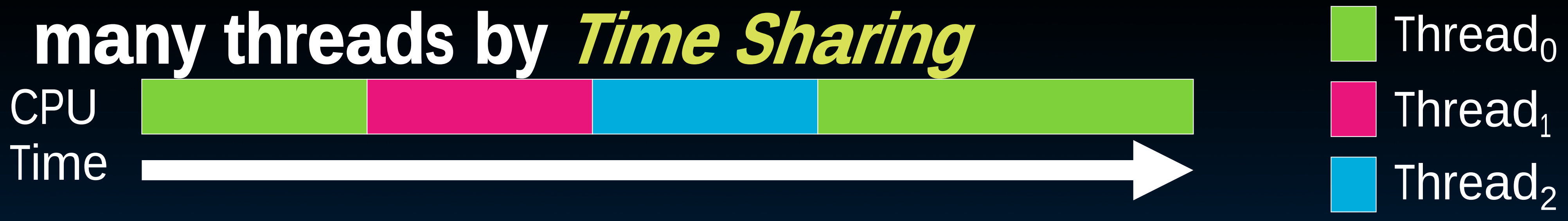
…156 total at this moment… How does my laptop do this?
Imagine doing 156 assignments all at the same time!

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*



Garcia, Nikolić

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

Garcia, Nikolić

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

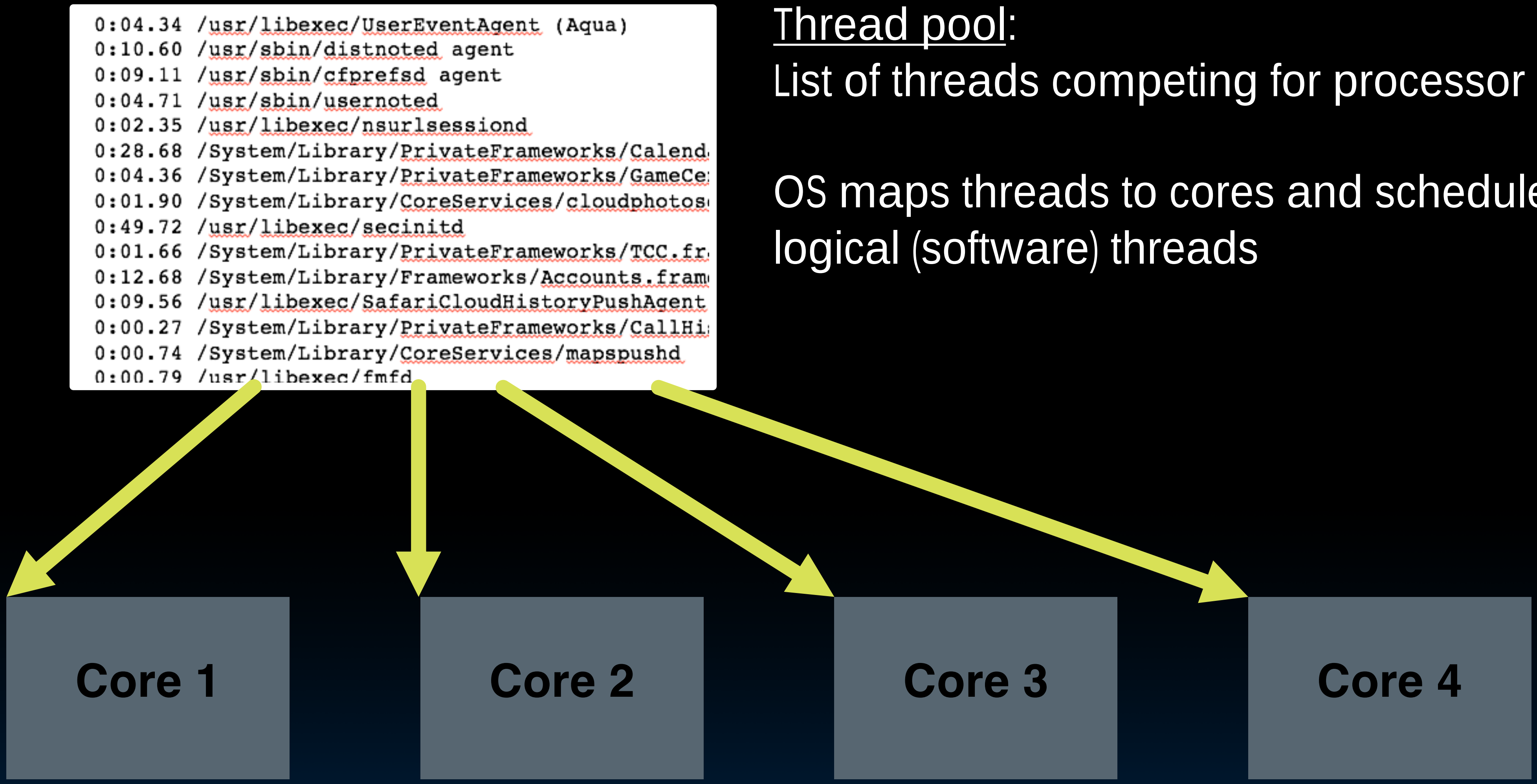
Operating System Threads

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

Garcia, Nikolić

Example: Four Cores



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

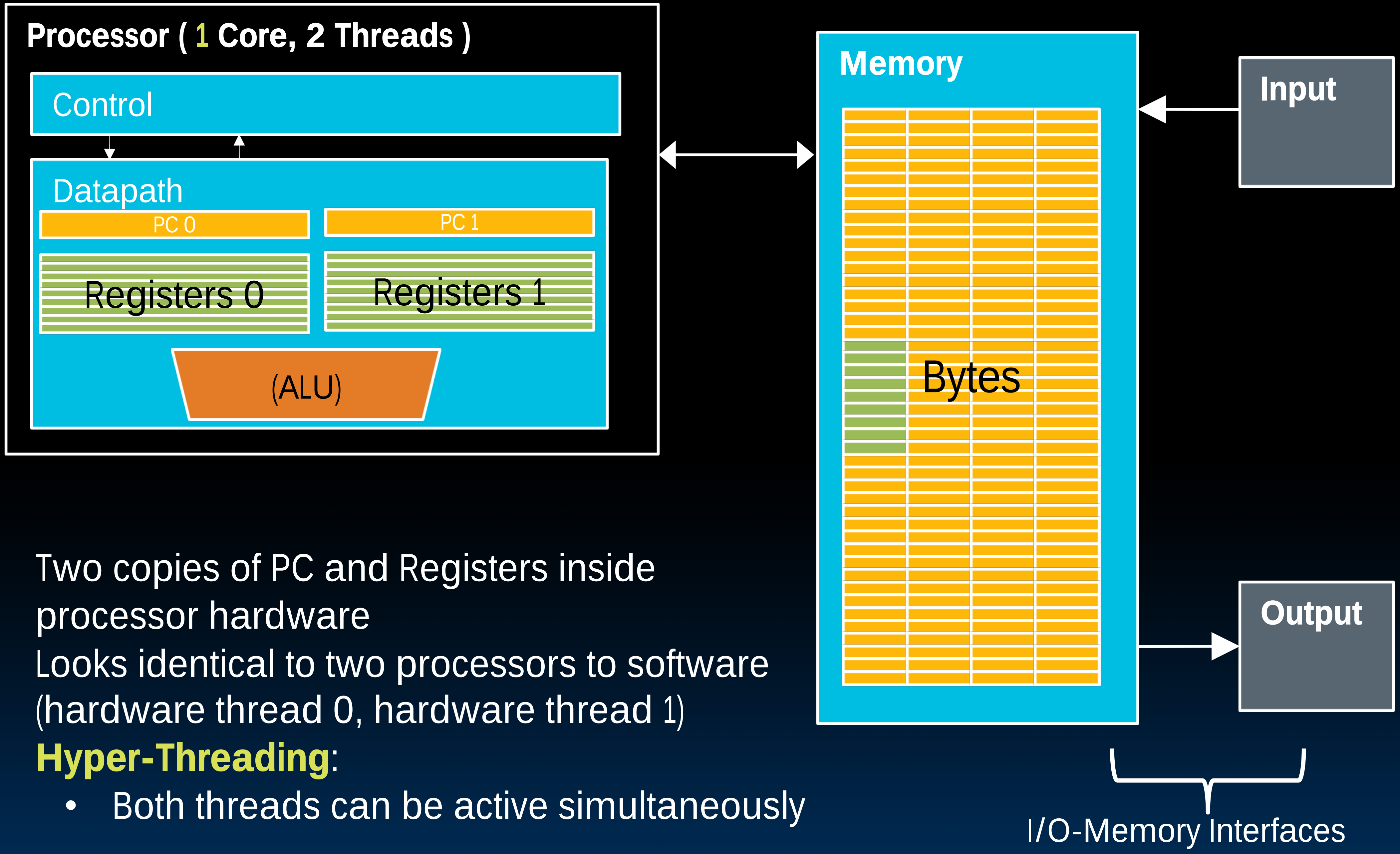
Multithreading

- **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)

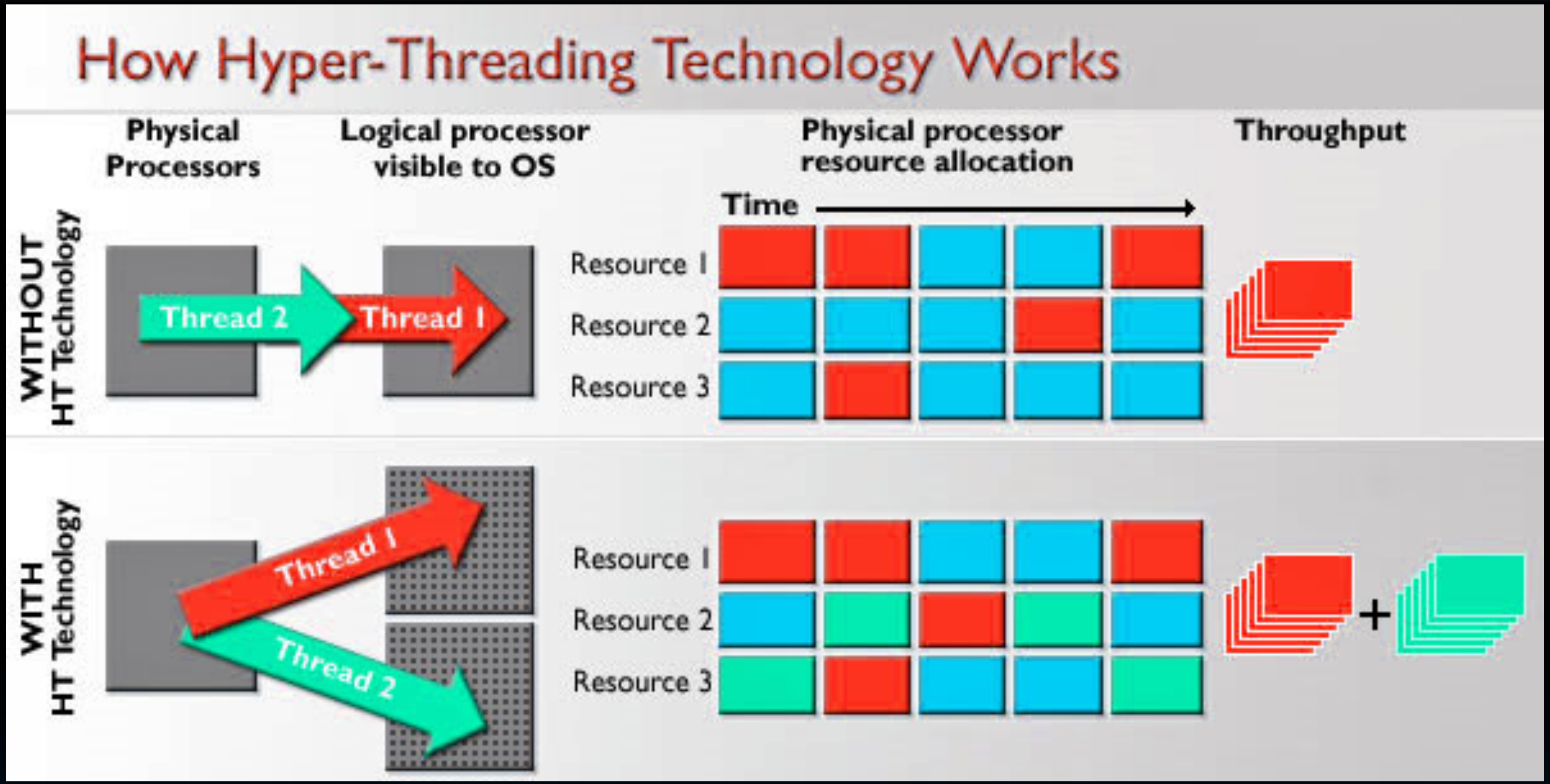
  → must perform switch in ≪ 1000 cycles
- **Can hardware help?**
  - Moore's Law: transistors are plenty

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**

Garcia, Nikolić

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

☑

Garcia, Nikolić

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

UC Berkeley
Teaching Professor
Dan Garcia

UC Berkeley
Professor
Bora Nikolić

Thread-Level Parallelism II

Berkeley
UNIVERSITY OF CALIFORNIA

cs61c.org

Parallel Programming Languages

Languages Supporting Parallel Programming

ActorScript	Concurrent Pascal	JoCaml	Orc
Ada	Concurrent ML	Join	Oz
Afnix	Concurrent Haskell	Java	Pict
Alef	Curry	Joule	Reia
Alice	CUDA	Joyce	SALSA
APL	E	LabVIEW	Scala
Axum	Eiffel	Limbo	SISAL
Chapel	Erlang	Linda	SR
Cilk	Fortan 90	MultiLisp	Stackless Python
Clean	Go	Modula-3	SuperPascal
Clojure	Io	Occam	VHDL
Concurrent C	Janus	occam-π	XC

Which one to pick?

Why So Many Parallel Programming Languages?

- **Why "intrinsics"?**
  - TO Intel: fix your #()&$! compiler, thanks…
- **It's happening ... but**
  - SIMD features are continually added to compilers (Intel, gcc)
  - Intense area of research
  - Research progress:
    - 20+ years to translate C into good (fast!) assembly
    - How long to translate C into good (fast!) parallel code?
      - General problem is very hard to solve
      - Present state: specialized solutions for specific cases
      - Your opportunity to become famous!

Garcia, Nikolić

Parallel Programming Languages

- **Number of choices is indication of**
  - No universal solution
    - Needs are very problem specific
  - E.g.,
    - Scientific computing/machine learning (matrix multiply)
    - Webserver: handle many unrelated requests simultaneously
    - Input / output: it's all happening simultaneously!
- **Specialized languages for different tasks**
  - Some are easier to use (for some problems)
  - None is particularly "easy" to use
- **61C**
  - Parallel language examples for high-performance computing
  - OpenMP

Garcia, Nikolić

OpenMP

Parallel Loops

- Serial execution:

```
for (int i=0; i<100; i++) {
        …
}
```

- Parallel Execution:

`for (int i=0; i<25; i++) { … }`	`for (int i=25; i<50; i++) { … }`	`for (int i=50; i<75; i++) { … }`	`for (int i=75; i<100; i++) { … }`

Garcia, Nikolić

Parallel `for` in OpenMP

```
#include <omp.h>

#pragma omp parallel for
for (int i=0; i<100; i++) {
        …
}
```


Garcia, Nikolić

OpenMP Example

```c
/* clang -Xpreprocessor -fopenmp -lomp -o for for.c */

#include <stdio.h>
#include <omp.h>
int main()
{
    omp_set_num_threads(4);
    int a[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int N = sizeof(a)/sizeof(int);

    #pragma omp parallel for
    for (int i=0; i<N; i++) {
        printf("thread %d, i = %2d\n",
            omp_get_thread_num(), i);
        a[i] = a[i] + 10 * omp_get_thread_num();
    }

    for (int i=0; i<N; i++) printf("%02d ", a[i]);
    printf("\n");
}
```

```
$ gcc-5 -fopenmp for.c;./a.out
% gcc -Xpreprocessor -fopenmp -
lomp -o for for.c; ./for
thread 0, i =  0
thread 1, i =  3
thread 2, i =  6
thread 3, i =  8
thread 0, i =  1
thread 1, i =  4
thread 2, i =  7
thread 3, i =  9
thread 0, i =  2
thread 1, i =  5
00 01 02 13 14 15 26 27 38 39
```

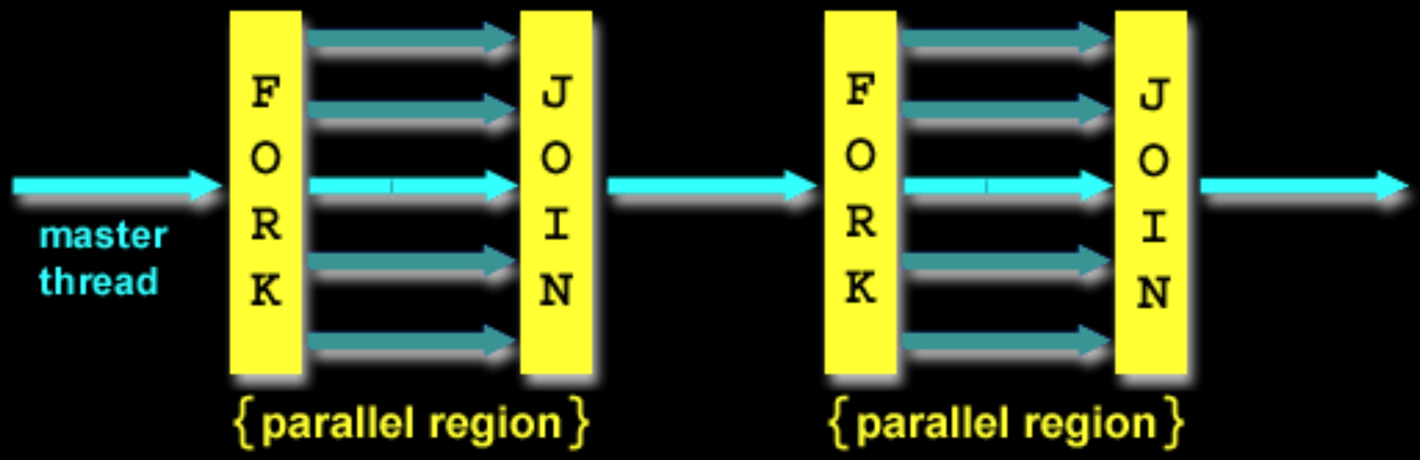
The call to find the maximum number of threads that are available to do work is **`omp_get_max_threads()`** (from **`omp.h`**).

OpenMP

- **C extension: no new language to learn**
- **Multi-threaded, shared-memory parallelism**
  - Compiler Directives, **`#pragma`**
  - Runtime Library Routines, **`#include <omp.h>`**
- **`#pragma`**
  - Ignored by compilers unaware of OpenMP
  - Same source for multiple architectures
    - E.g., same program for 1 & 16 cores
- **Only works with shared memory**

OpenMP Programming Model

- **Fork - Join Model:**



- **OpenMP programs begin as single process *(main thread)***
  - Sequential execution
- **When parallel region is encountered**
  - Master thread "forks" into team of parallel threads
  - Executed simultaneously
  - At end of parallel region, parallel threads "join", leaving only master thread
- **Process repeats for each parallel region**
  - Amdahl's Law?

Garcia, Nikolić

What Kind of Threads?

- OpenMP threads are operating system (software) threads
- OS will multiplex requested OpenMP threads onto available hardware threads
- Hopefully each gets a real hardware thread to run on, so no OS-level time-multiplexing
- But other tasks on machine compete for hardware threads!
- Be "careful" (?) when timing results for Projects!
  - 5AM?
  - Job queue?

☑

Garcia, Nikolić