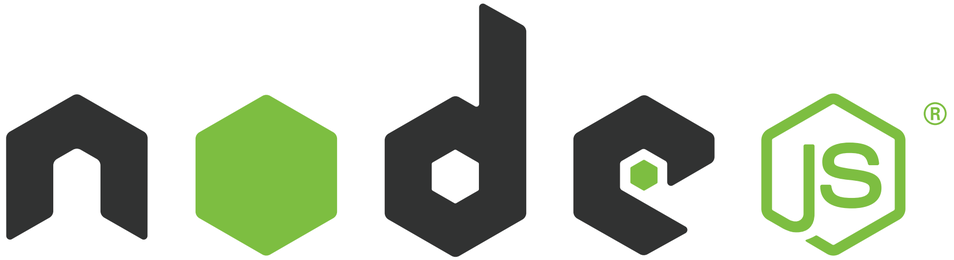
Intro to



or: Kernals, Processes, Threads — Oh My!
or: Fun with Modules
or: The Kitchen Sink & Async
or: Six Degrees from Ryan Dahl

What is Node?

Node.js: the Essentials

- Node is a program (written in C, C++, and JS)
- It can run JavaScript via the Chrome V8 engine (C++)
- It provides APIs for your OS's file / network system (slow I/O)
- In other words, Node lets you run JS on a computer *outside of a browser* and interact with stuff on that computer
- Why?

Upcoming

- How computers multi-task
  - CPUs / processes / threads
  - Scheduling system
- History / context
- Node as process
- Node modules
- Node asynchronicity


Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking
Multi-tasking

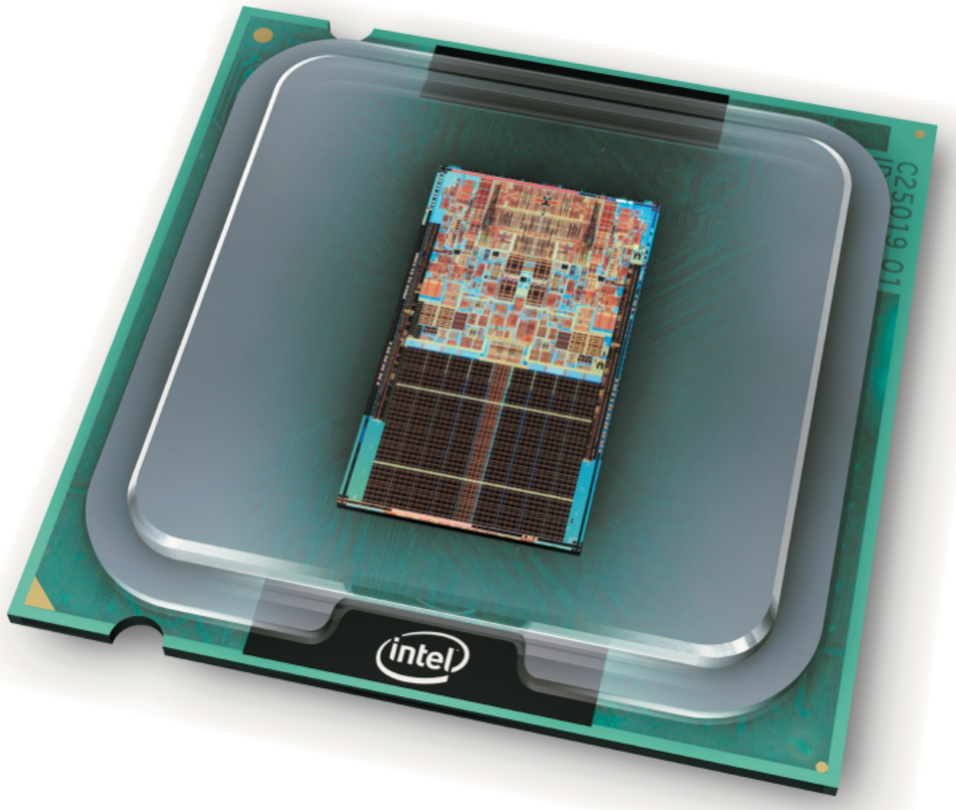
Program vs. Process

"recipe"

- **Program is data**
  - machine code (pre-compiled)
  - bytecode
  - text file (can be interpreted)
- **Inert — not doing anything**
- **Ready to be run as a process**

- **Process is execution** "cooking"
  - memory allocated
  - CPU performing steps
- **"Live"**
- **Produces results**
- **Interactive**
- **Can be started/stopped**
- **Multiple processes from one program…**

CPU Core



"chef" — only speaks machine code

Grab File Edit Capture Window Help

56° 100% Mon 12:04 PM

Skype

Fullstack Academy
$9.52

Search

Home
Contacts
Recent

Kyle Burke
Offline

en.wikipedia.org/wiki

LearnDot Fs-Interview onAir Feedly Facebook R/T.n R/S NMB NYT Pocket K-r

1506-Library/g... sandy-bridge-... apcnewschip-... i7-art-2.jpg 50... Intel-Sandy-Bri... core2duo_illus.... Preemption (co...

"busywait" while waiting for requested input (such as disk, keyboard or network input). During this time, the process was not performing useful work, but still maintained complete control of the CPU. With the advent of interrupts and preemptive multitasking, these I/O bound processes could be "blocked", or put on hold, pending the arrival of the necessary data, allowing other processes to utilize the CPU. As the arrival of the requested data would generate an interrupt, blocked processes could be guaranteed a timely return to execution.

Although multitasking techniques were originally developed to allow multiple users to share a single machine, it soon became apparent that multitasking was useful regardless of the number of users. Many operating systems, from mainframes down to single-user personal computers and no-user control systems (like those in robotic spacecraft), have recognized the usefulness of multitasking support for a variety of reasons. Multitasking makes it possible for a single user to run multiple applications at the same time, or to run "background" processes while retaining control of the computer.

Time slice [edit]

The period of time for which a process is allowed to run in a preemptive multitasking system is generally called the *time slice*, or *quantum*. The scheduler is run once every time slice to choose the next process to run. The length of each time slice can be critical to balancing system performance vs process responsiveness - if the time slice is too short then the scheduler will consume too much processing time, but if the time slice is too long, processes will take longer to respond to input.

An interrupt is scheduled to allow the operating system kernel to switch between processes when their time slices expire, effectively allowing the processor's time to be shared between a number of tasks, giving the illusion that it is dealing with these tasks simultaneously, or concurrently. The operating system which controls such a design is called a multi-tasking system.

Systems supporting preemptive multitasking [edit]

Today, nearly all operating systems support preemptive multitasking, including the current versions of Windows, Mac OS,

Slack

Fullstack Academy
Gabriel Lebec

general
random
help-ticket-test
+51 More...

DIRECT MESSAGES
slackbot
Alex Castillo
Alexander Holman
Ben Neiswander
Chelsea Du
Chris Carrick
Cristina Colon
Jay Lee
Jenny Cui
Jimmy Farrell
Michael Abraham
Mike Lee
Roger Beaman
Ryan Dwyer

#random 287

July 30th, 2015

Stephen Colacrucio 10:28 PM
left #random, along with @tmoreton

July 31st, 2015

Christian Cueto 1:11 PM
Slack Update

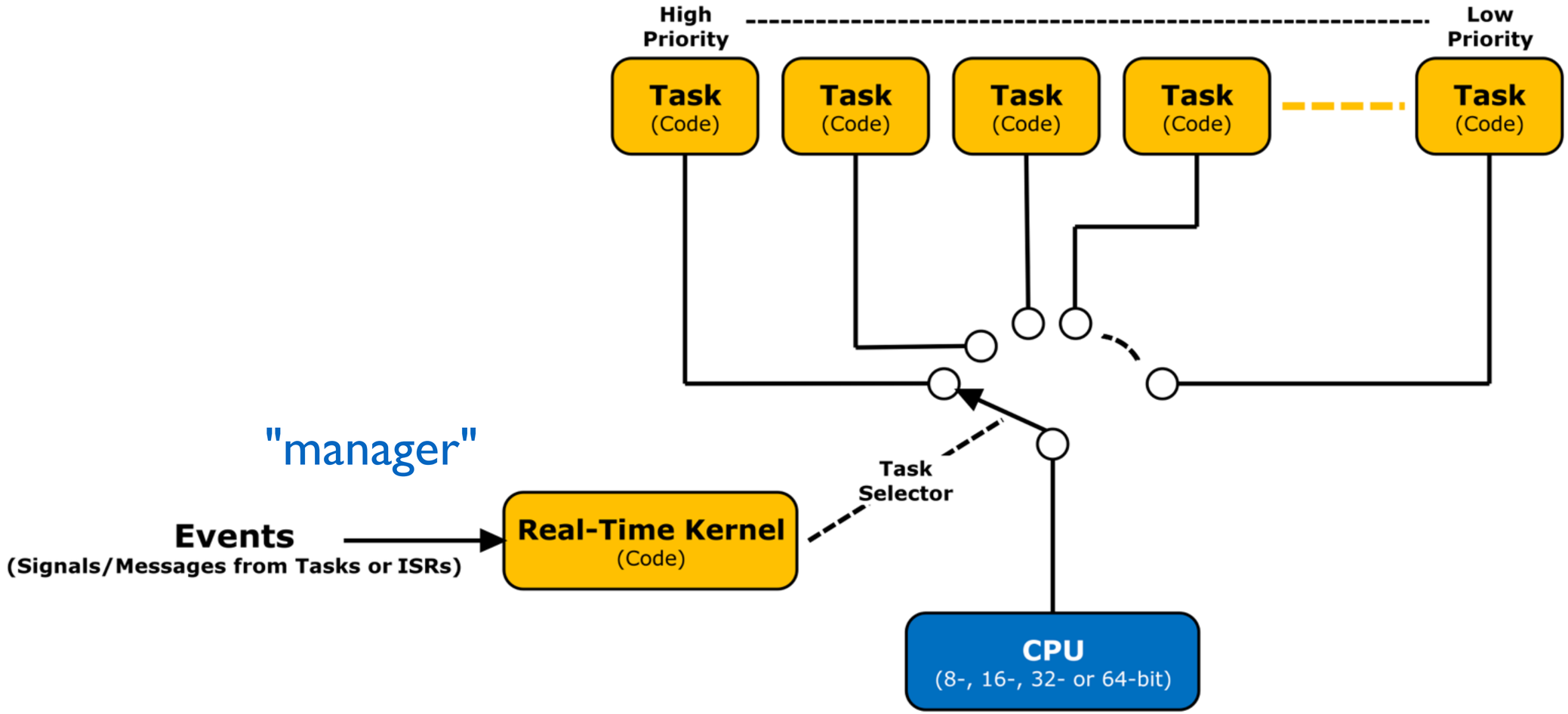
2

40KB PNG in #random · Add comment · Open original

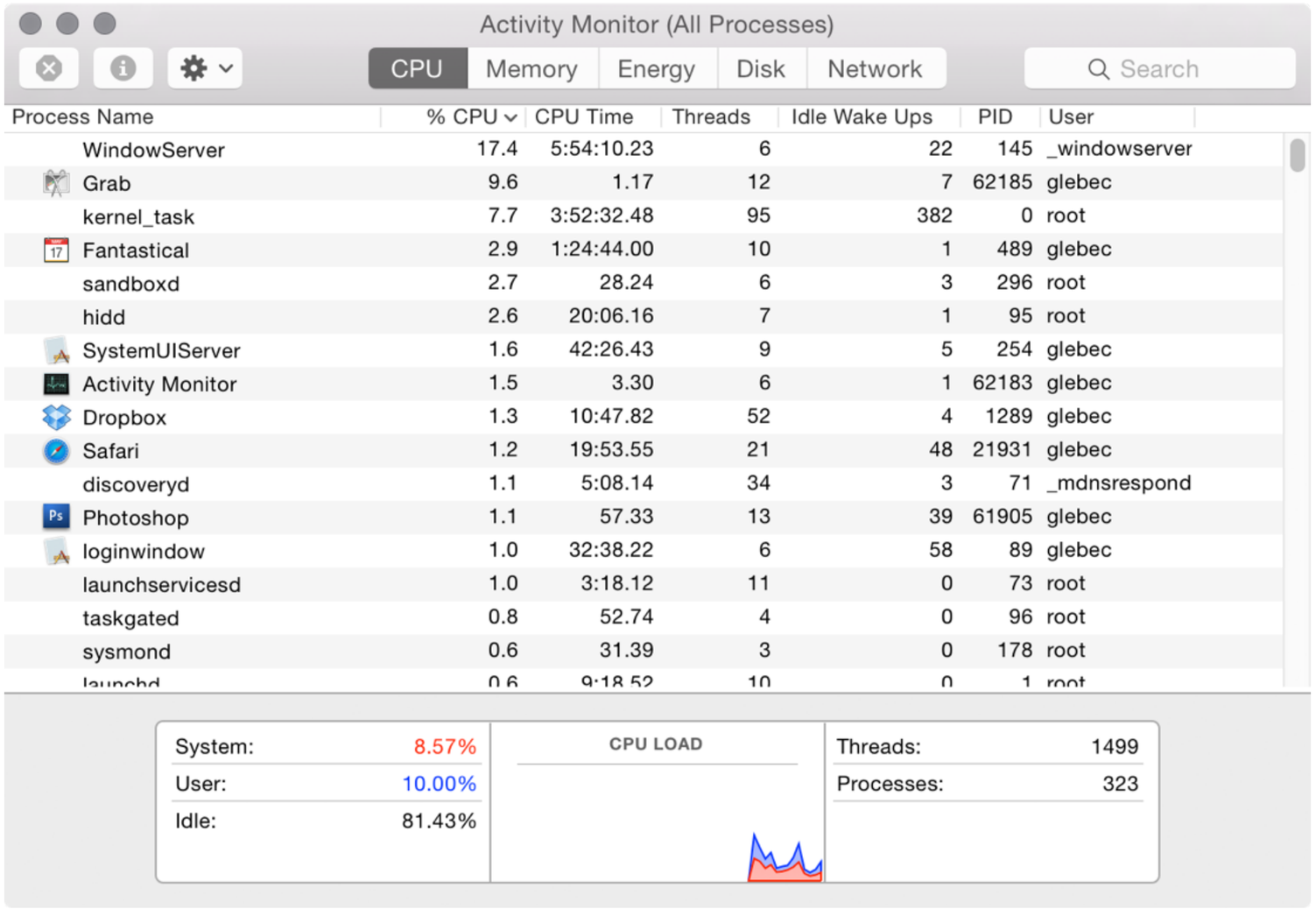
Highscores Help

NEXT
SCORE 2
LINES 0
LEVEL 1
LPM 0.00

Time Slice


High Priority
Low Priority
Task (Code)
Task (Code)
Task (Code)
Task (Code)
Task (Code)
"manager"
Task Selector
Events
(Signals/Messages from Tasks or ISRs)
Real-Time Kernel (Code)
CPU
(8-, 16-, 32- or 64-bit)

The OS Manages Processes

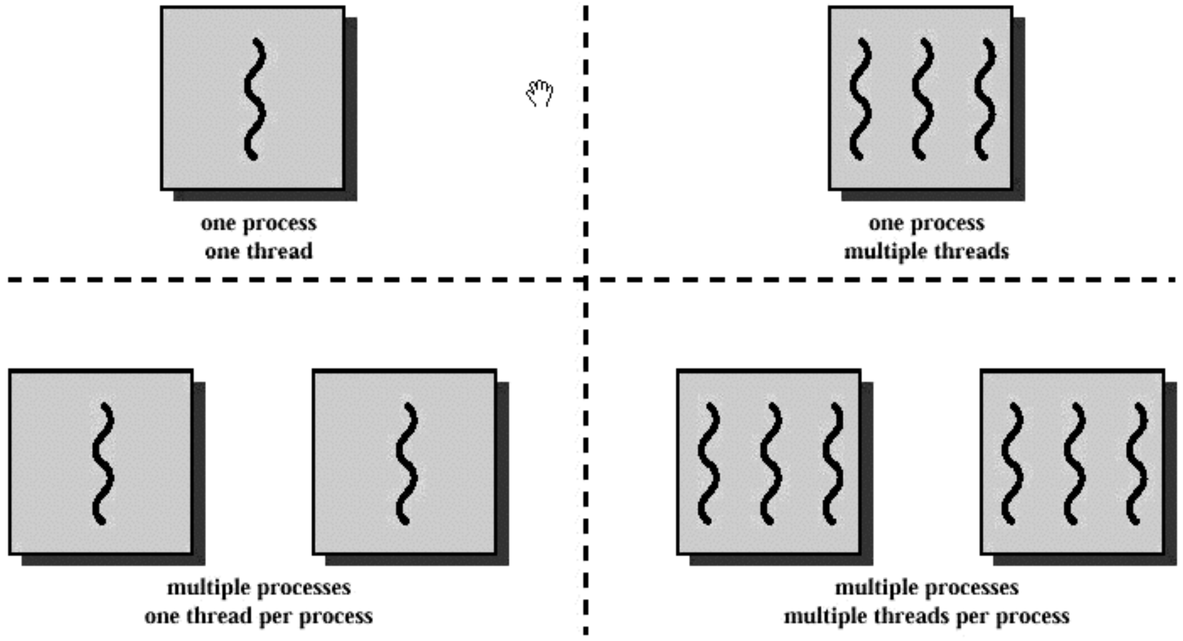


What if a process itself needs to multi-task?

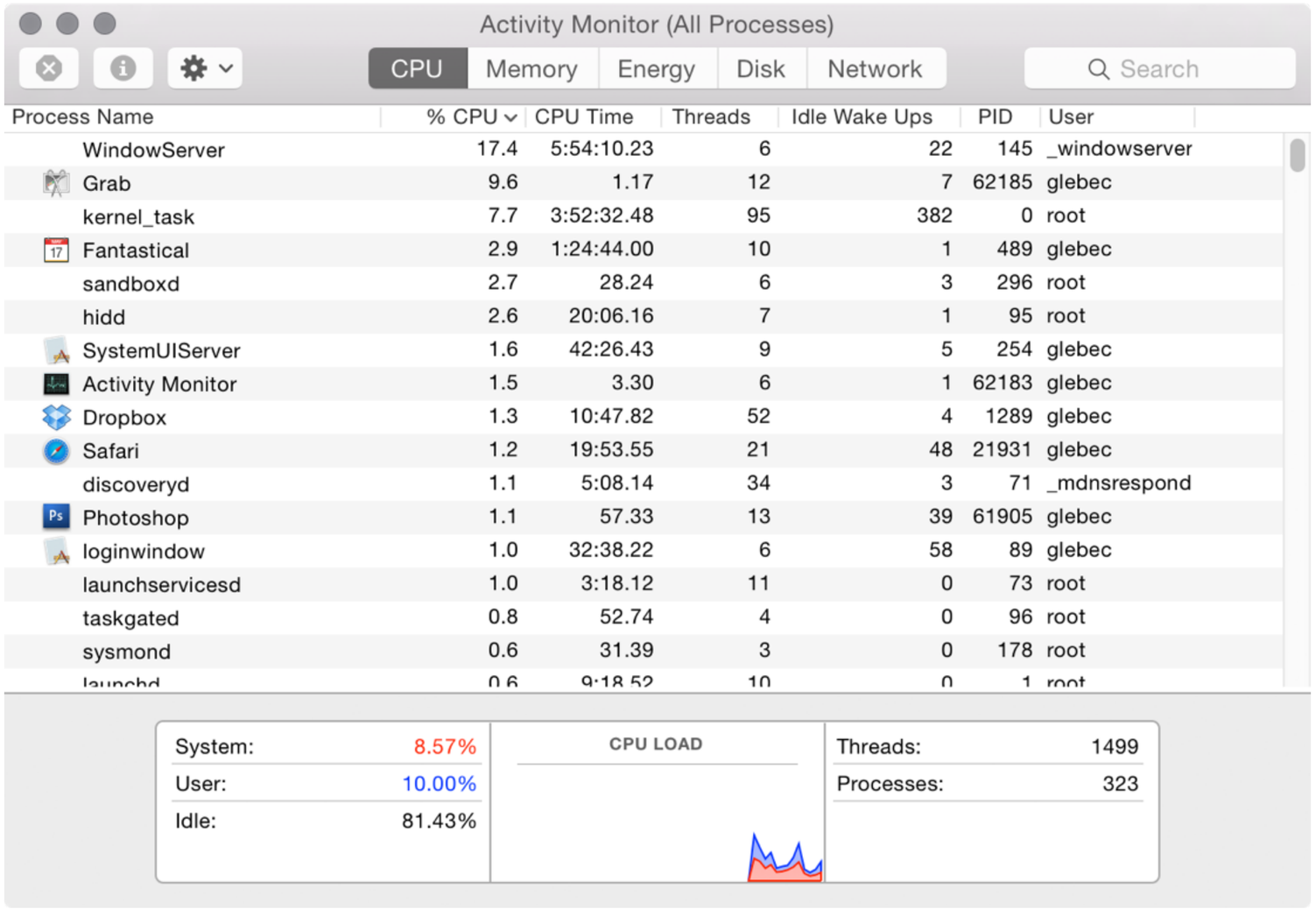
Break It Down Further: Threads

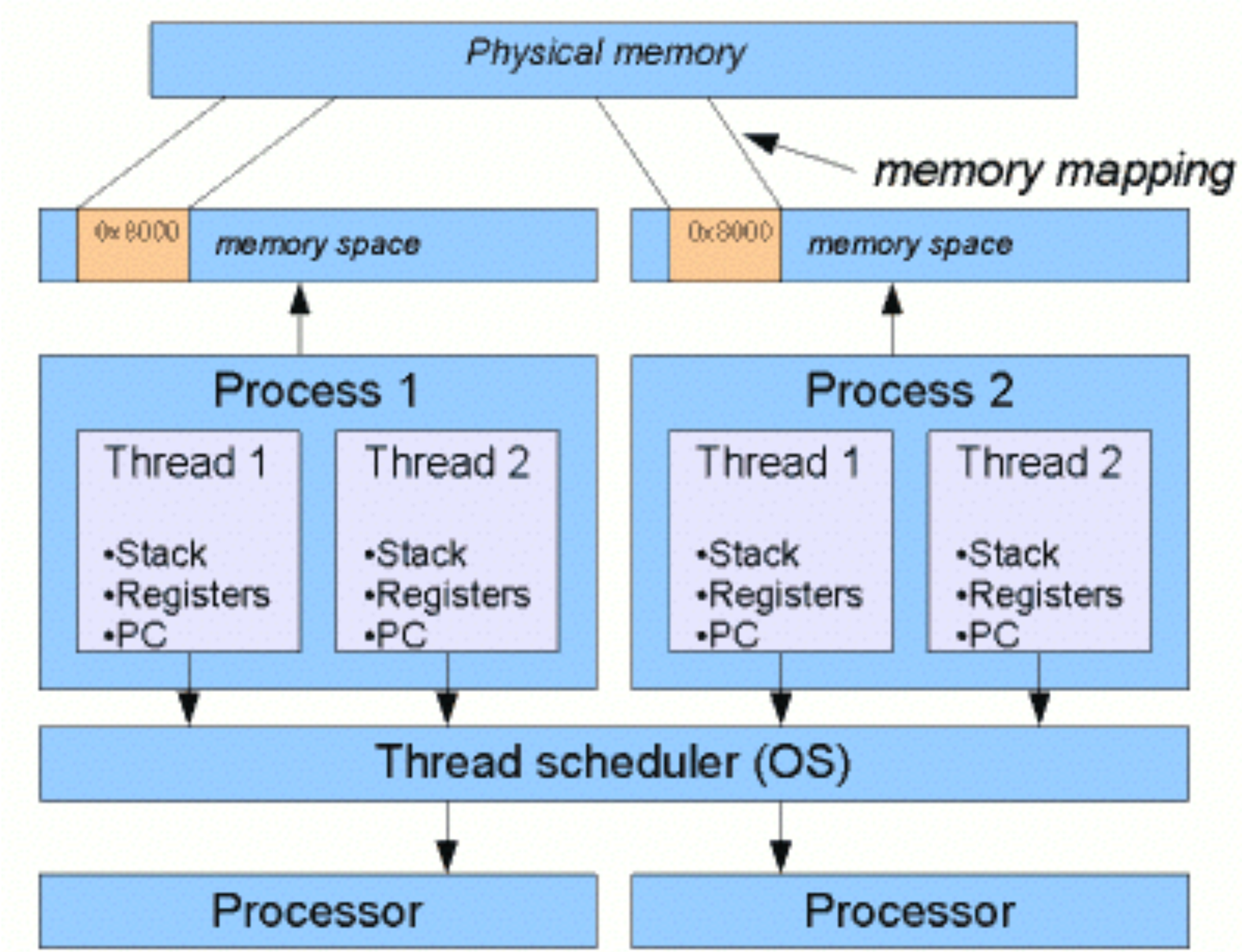
- A thread is a sequence of steps that the CPU can execute
- One *or more* threads per process — share memory
- OS *schedules* jumps between threads using a fair algorithm

"activity", perhaps

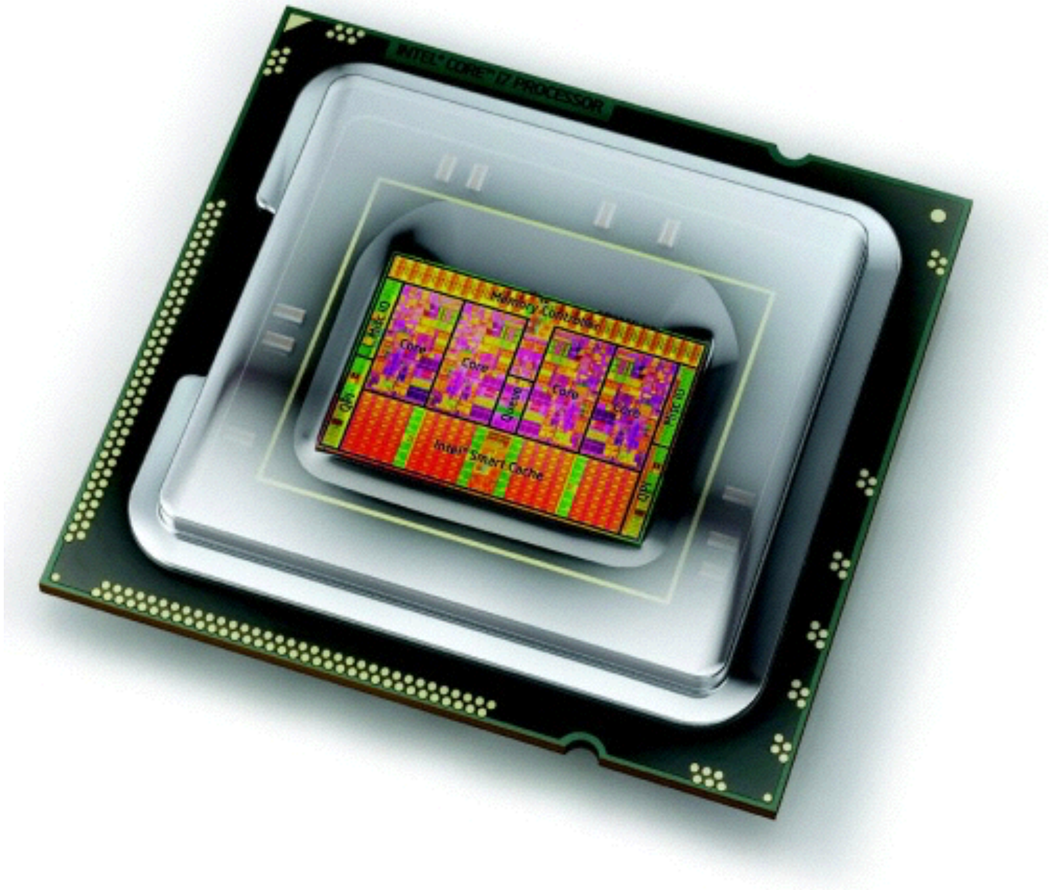


The OS Schedules Threads



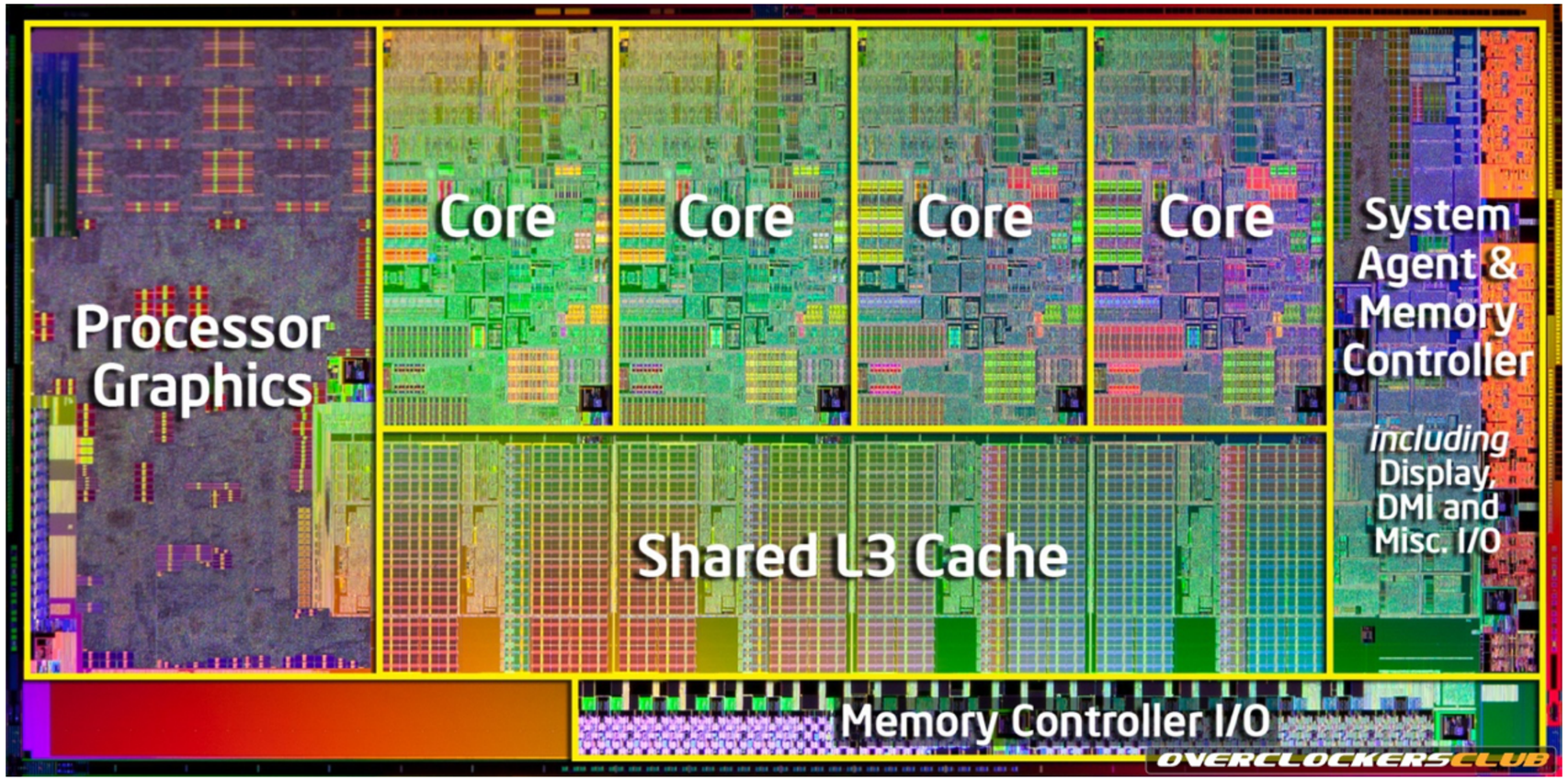

Physical memory
memory mapping
0x8000
memory space
0x8000
memory space
Process 1
Thread 1
•Stack
•Registers
•PC
Thread 2
•Stack
•Registers
•PC
Process 2
Thread 1
•Stack
•Registers
•PC
Thread 2
•Stack
•Registers
•PC
Thread scheduler (OS)
Processor
Processor

...Wait — Multiple Processors?



(Aside:) Multi-Core Processors

"chefs"

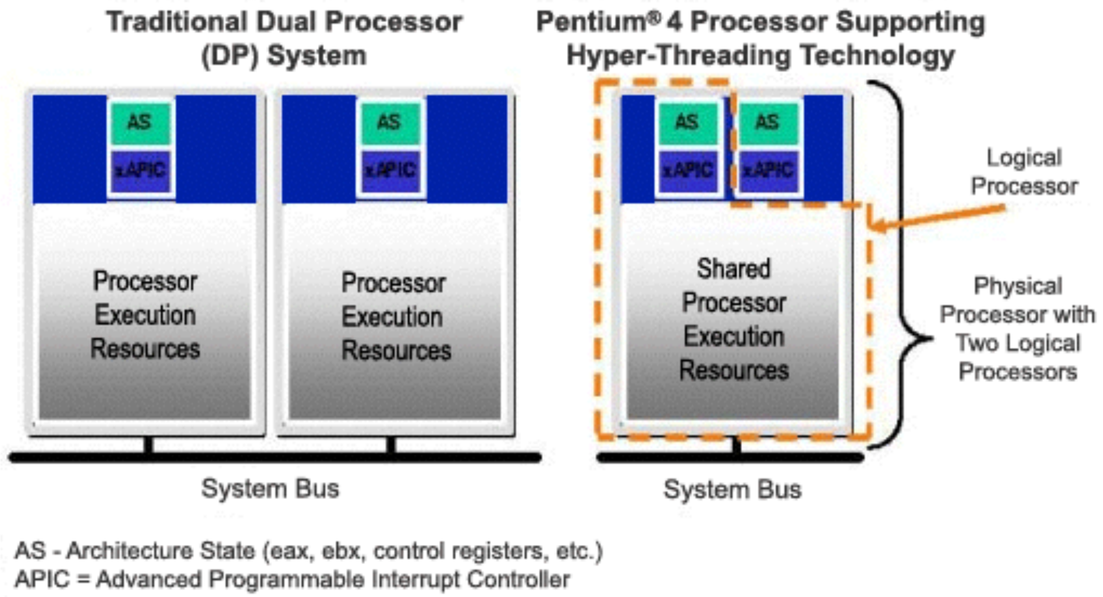


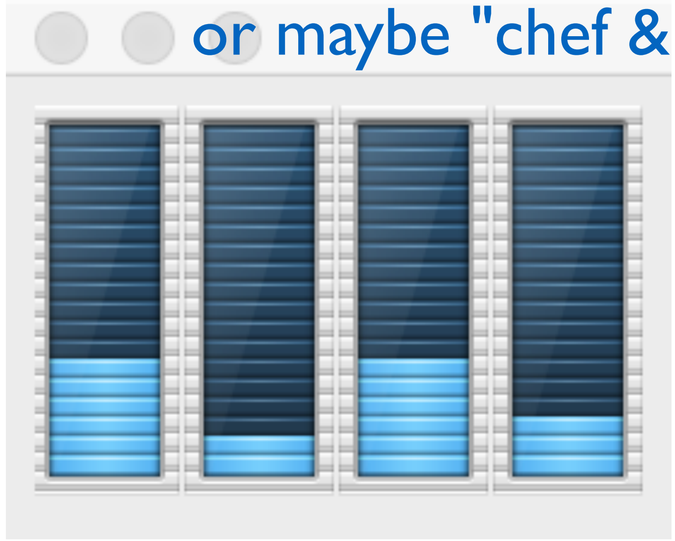
But wait, there's more — HyperThreading

- Each physical core can run *two* threads, doubling the number of "logical" cores! These black magic extra threads aren't as fast as the main threads, but they still help a little bit.

"left hand & right hand,"

or maybe "chef & sous-chef"





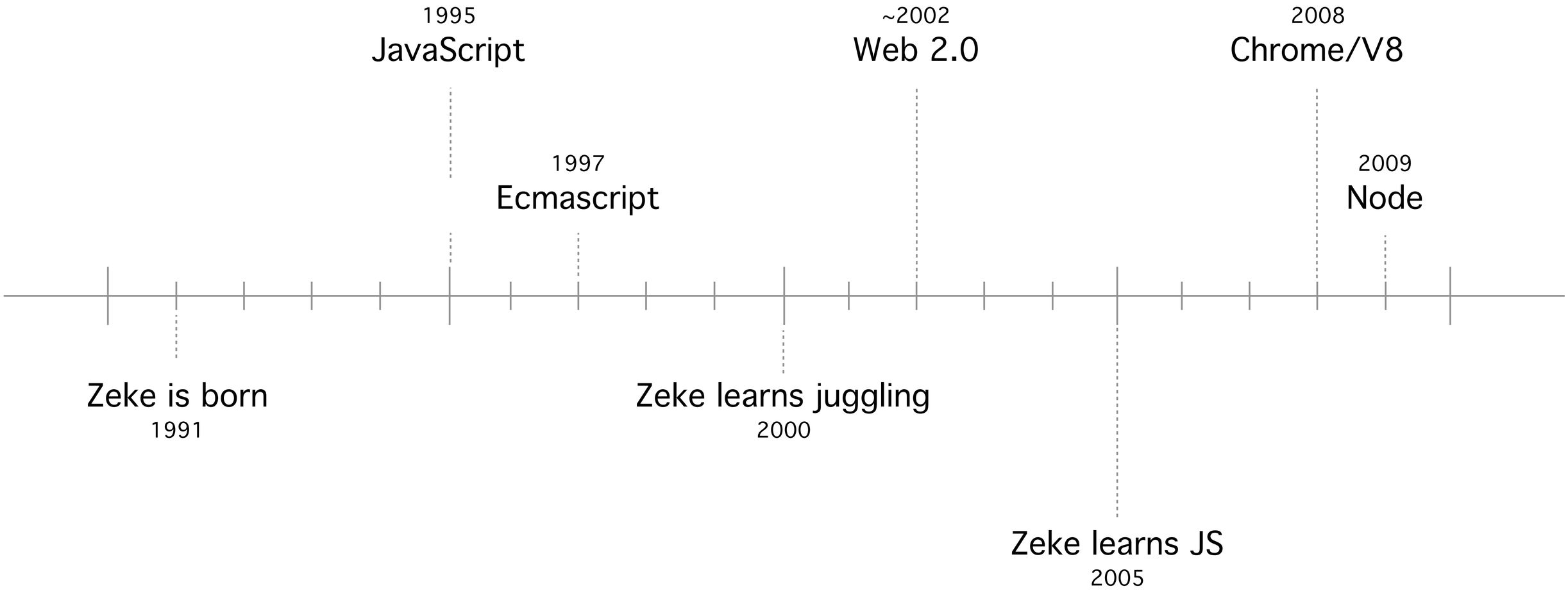
Takeaway:

- **Program** is a file (data) that can be loaded/executed as a **process**
- **Process** is state (memory/variables) and one or more **threads**
- **Thread** is a sequence of operations that the CPU performs
- **Operating system** schedules jumps between threads (loading and unloading state), as there are many more threads than logical **cores**
- **CPU cores** are the machines that actually perform steps; most processors have 2–4 physical cores / 4–8 logical cores

Node as Process

(Demo)

A Brief History of Everything



JS in Node vs Browser

"kitchen" with tools (APIs)

"interpreter" — translates JS to machine code, live

fs | process | net

window | history | document

Browser Tool Examples

- **Synchronous stuff:**
  - console
  - window
  - document
- **Asynchronous stuff:**
  - setTimeout(callback, delay) (note: not in ECMAScript, "not JS" — an API!)
  - setInterval(callback, delay) — ditto above
  - new XMLHttpRequest() / .open, etc. — making HTTP requests
  - element.onclick(callback) // slightly different example

Node Tool Examples

- **Synchronous stuff:**
  - console (same name, but a wrapper for process.stdout)
  - global (no window!)
  - module (we'll get to this)
- **Asynchronous stuff:**
  - setTimeout(callback, delay) — same name for convenience's sake
  - setInterval(callback, delay) — ditto
  - http.request(options, callback) — uses built-in `http` JS module
  - etc.

(Loose) Metaphor

Example	Term	Metaphor
OS X 10.10 Yosemite	operating system	manager
physical or logical core	CPU / core / processor	chef / sous-chef
checking network response	thread	specific activity sequence in cooking
node or *chrome* in activity monitor	process	cooking (chef doing work)
Node.js / Chrome	runtime environment (has APIs)	kitchen (with tools)
V8	engine / VM / interpreter	translator / interpreter
JavaScript	programming language	recipe language
`log('hi');`	program	recipe

Big Use Case: Server

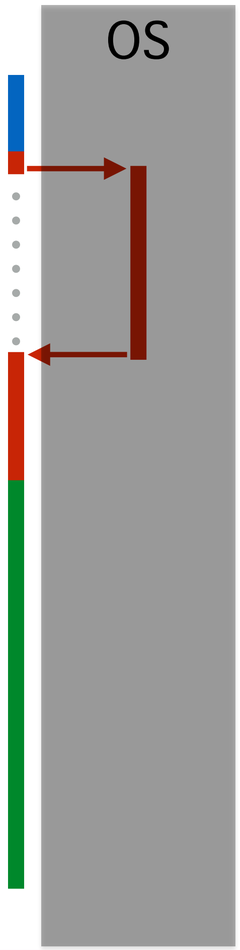
- A program running on a computer
- Listens on a *port* for *requests*
- Sends *responses* back
- Communication must follow agreed-upon *protocol* (e.g. HTTP)
- [DEMO]
- Problem: communication over network is super-slow! Should be able to field other requests / do other stuff in meantime.

Slow *blocking* I/O is handed off to a thread

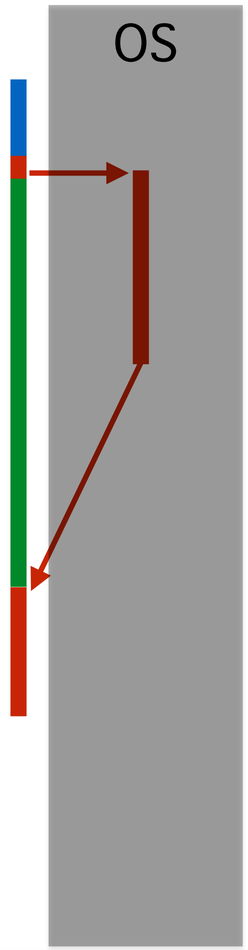
- **CPU clock cycles per access level (*Dahl 2010):**
  - Dynamic memory: "non-blocking"
    - L1 cache: 3
    - L2 cache: 14
    - RAM: 250
  - "Blocking" I/O:
    - Disk: 41,000,000
    - Network: 240,000,000

Concurrency

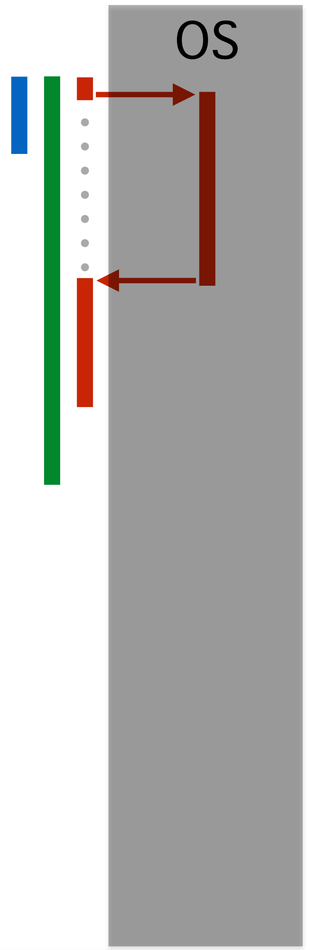
blocking



non-blocking



parallel (blocking)



"Threads are Hard"

- Every thread comes with performance overhead — memory, CPU, etc.
- Switching threads within a process consumes clock cycles (have to schedule jumps)
- Switching threads between processes consumes clock cycles AND waits for memory (have to load process state)
- Communicating between threads in the same process is very tricky — synchronizing results, being efficient
- Thread programming best left to experts

Server Strategies

- Apache: every new connection gets its own thread (wasteful of resources, doesn't scale well, lots of work to synchronize)

- Nginx: an event loop which queues *callbacks* back into the main execution stack, once thread managing blocking code finishes

"Browsers got it right — abstract that to callbacks with the DOM API."

RYAN DAHL, CREATOR OF NODE

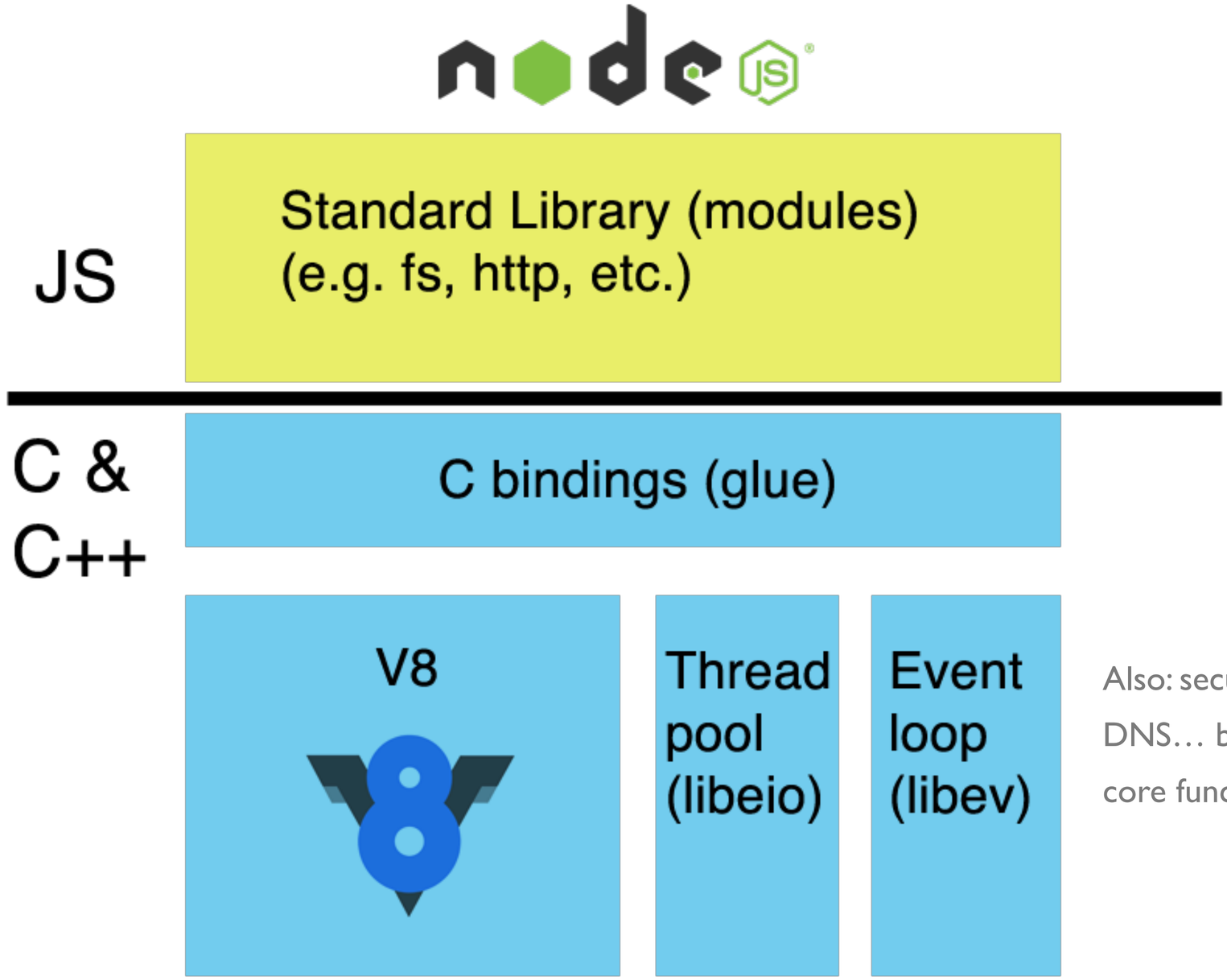
Er, not exactly

"Node.js is a ~~single-threaded~~, event-driven, non-blocking I/O platform"

– SOME PEOPLE ON THE INTERNET

"JavaScript is single-threaded" …arguably yes

– OTHER PEOPLE ON THE INTERNET

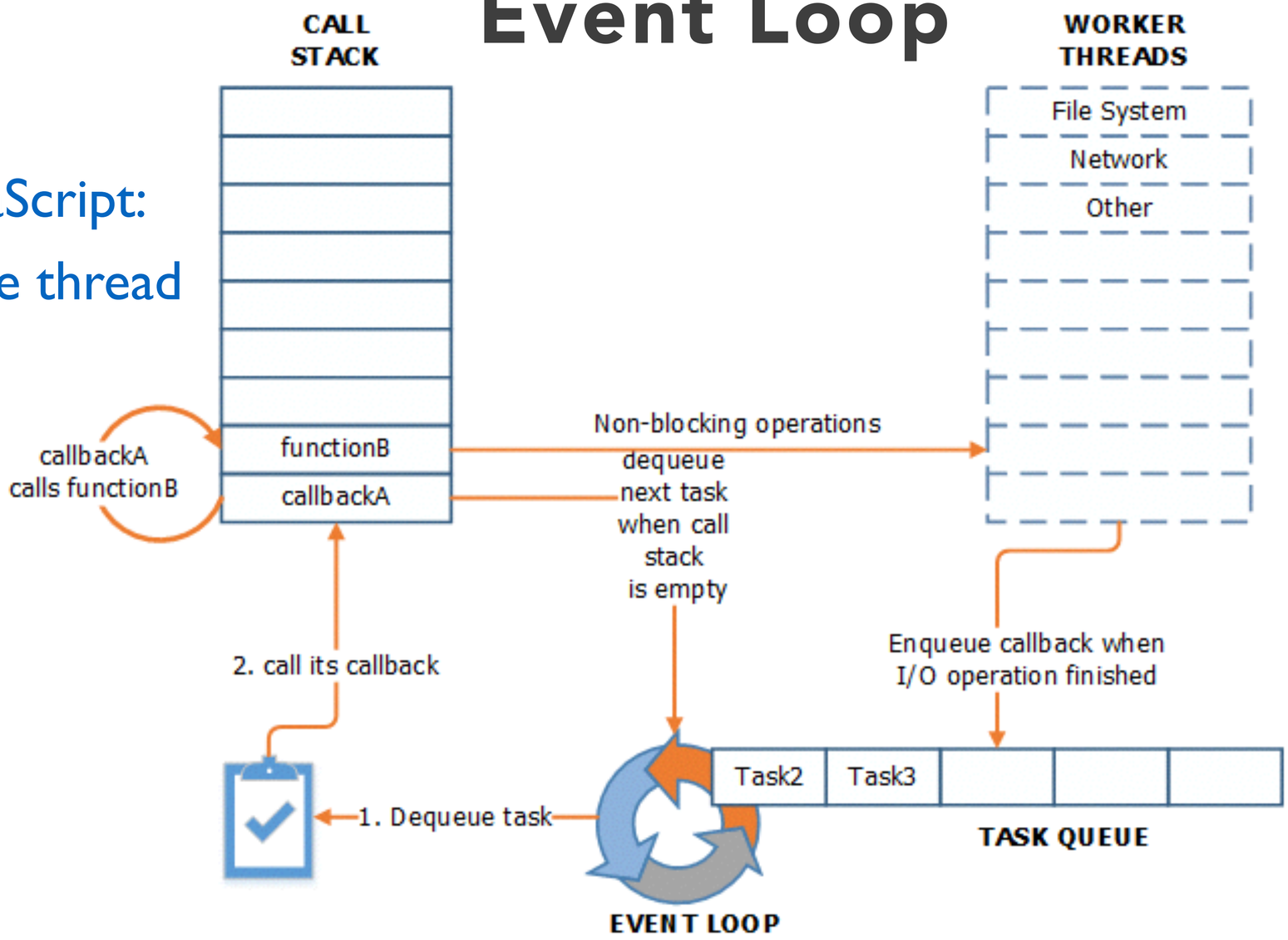


Also: secure crypto, DNS… but this is the core functionality

Event Loop

JavaScript:

One thread



Thread pool (libeio):

Slow stuff, multiple threads

Event loop (libev):

One thread