Introduction to OS

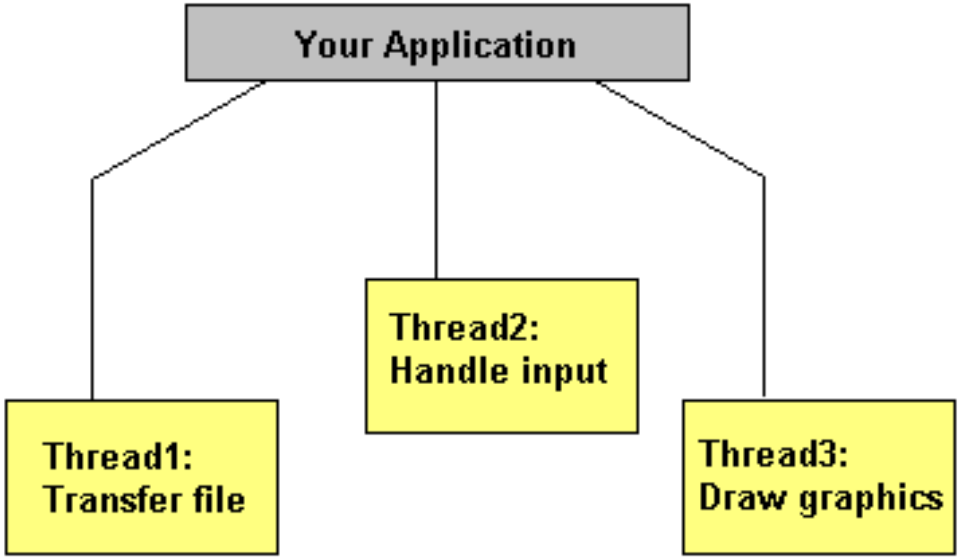
Introduction to Concurrency

(Processes, Threads, Interrupts, etc.)

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

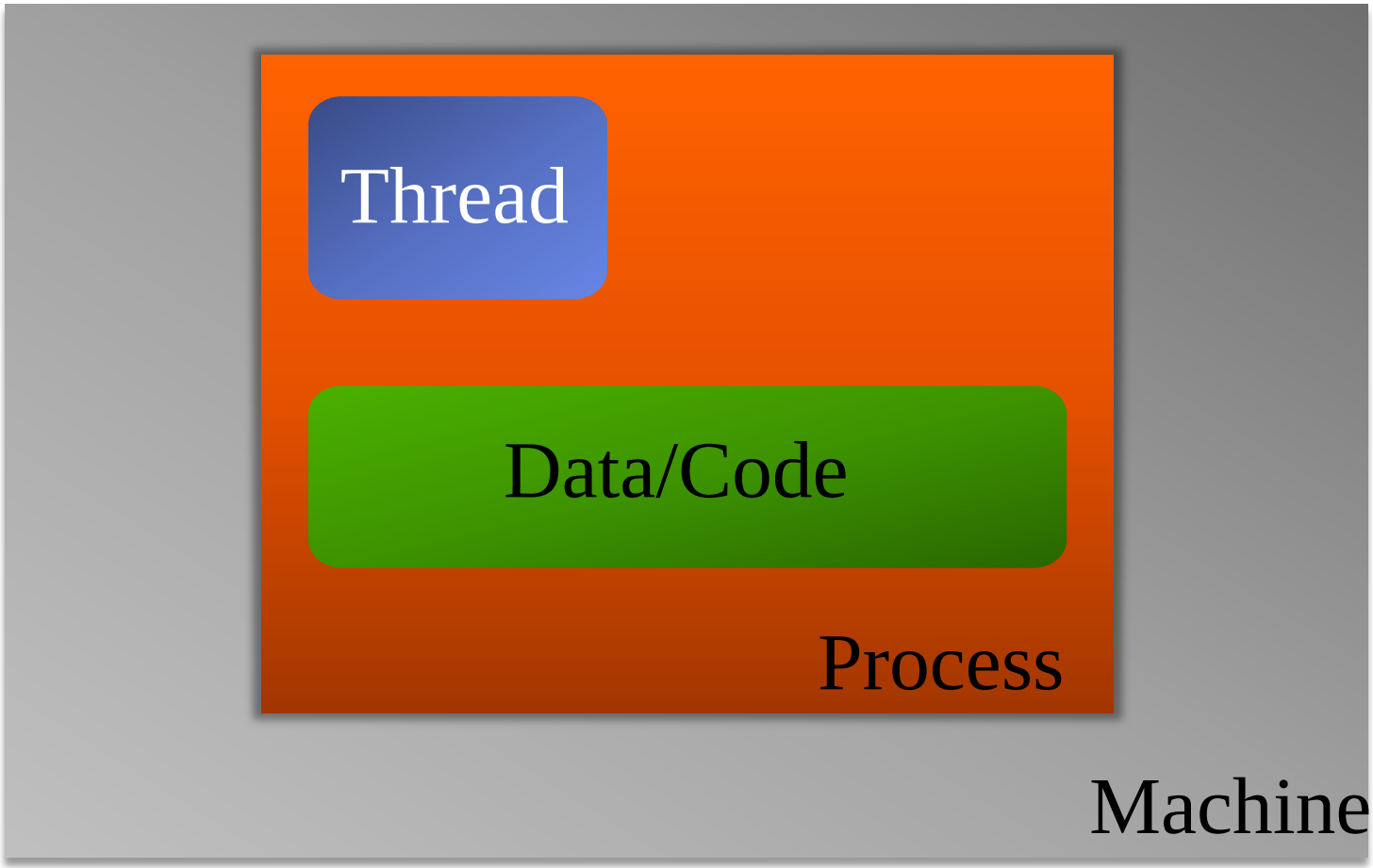
Why Concurrency?



Why Worry?

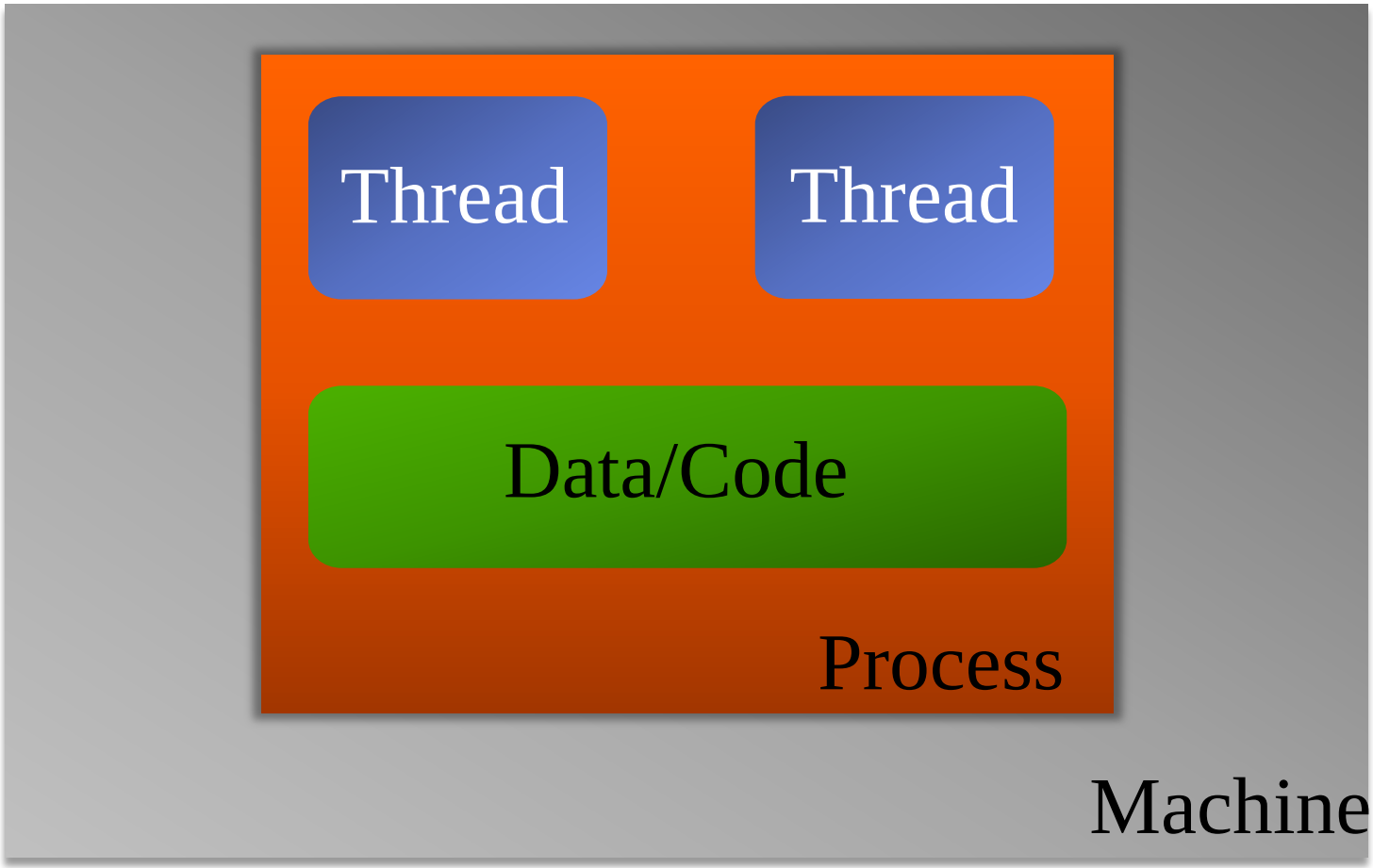
- Concurrency is hard
  - and I've only ever needed single-threaded programs: Why should I care about it?"
- ***Answer:***
  - ***multi-core*** computers (not faster chips), increasing use of ***clusters***
  - lots of other domains in which concurrency is the norm
    - ✓ Robotics, high performance computing (e.g. clusters, grids, clouds)
  - Web browsers: examples of multi-threaded GUI applications
    - ✓ without threads the UI would block as information is downloaded

Basics: Single Thread, Single Process, Single Machine



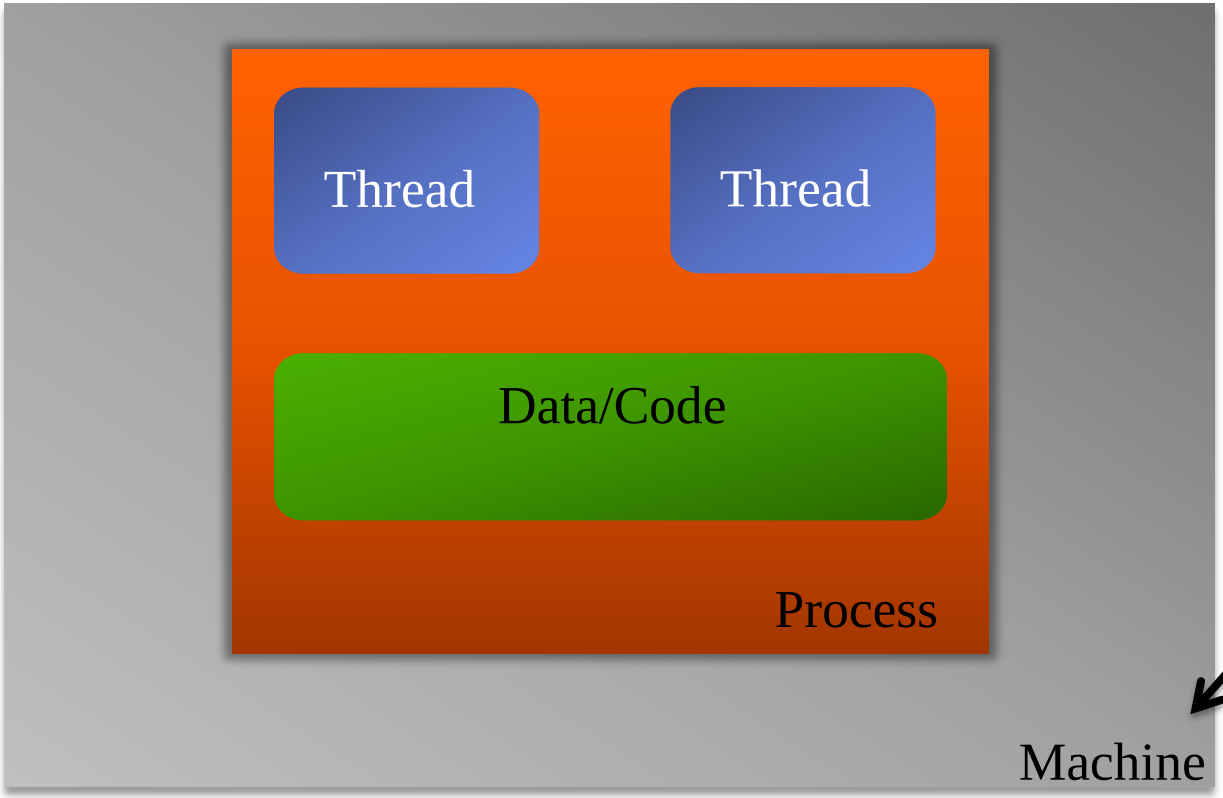
Sequential Program == Single Thread of Control

Basics: Multiple Thread, Single Process, Single Machine



Concurrent Program == Multiple Threads of Control

Multi-Thread: But is it truly parallel?



We may have multiple threads in this process, but we may not have events truly occurring in parallel. Why not?

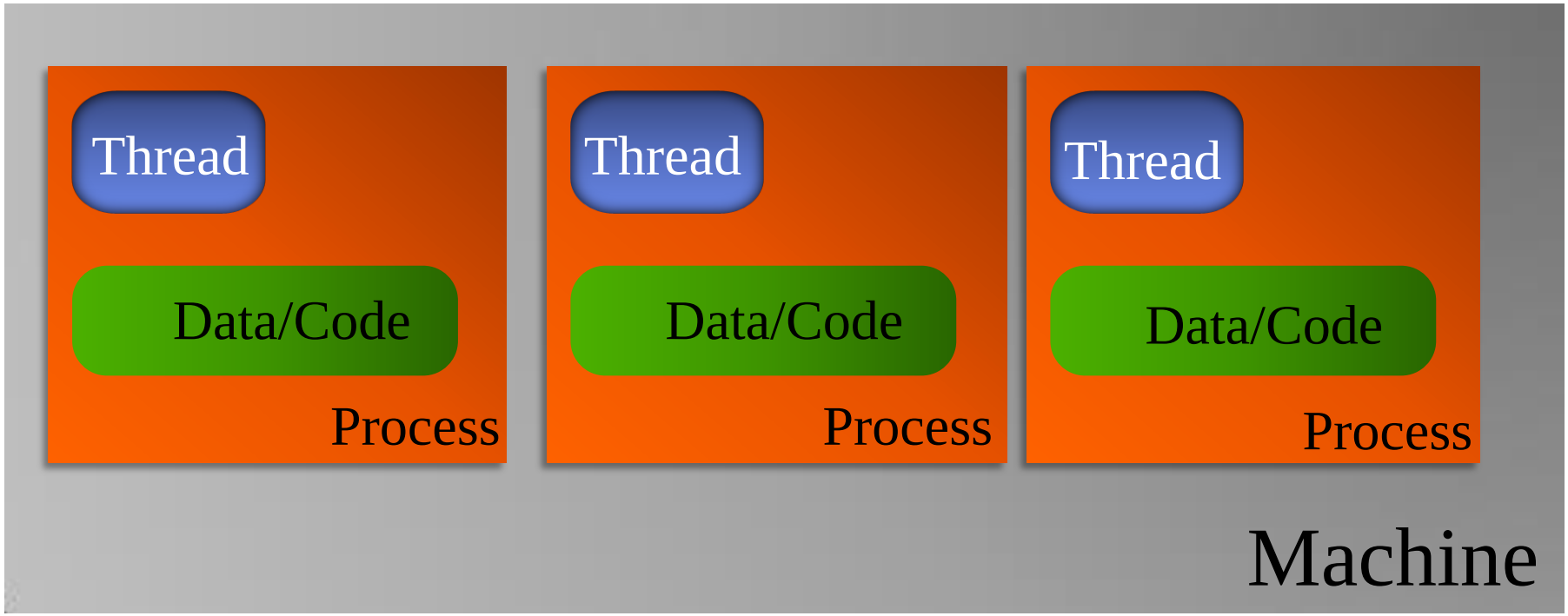
It depends on….

Concurrent Program == Multiple Threads of Control

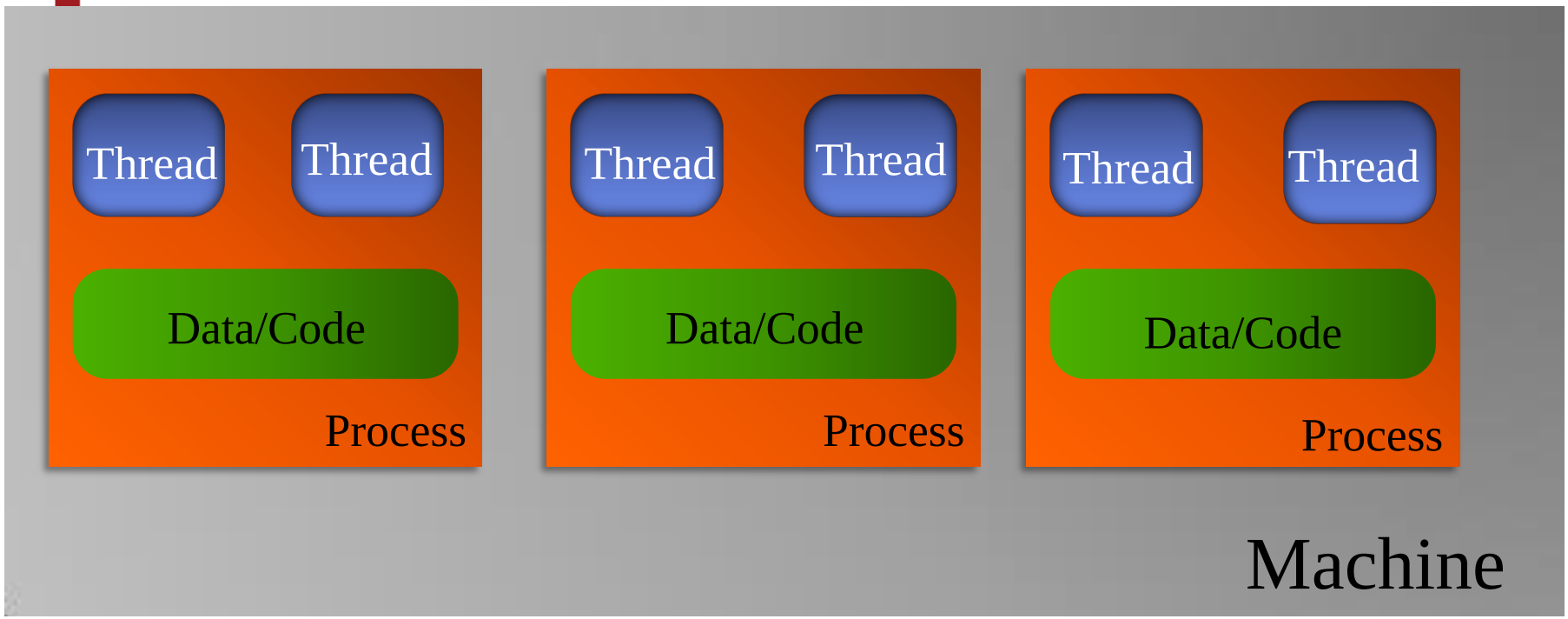
multiple processors,
true parallelism

Otherwise,
parallelism is simulated

Basics: Single Thread, Multiple Process, Single Machine

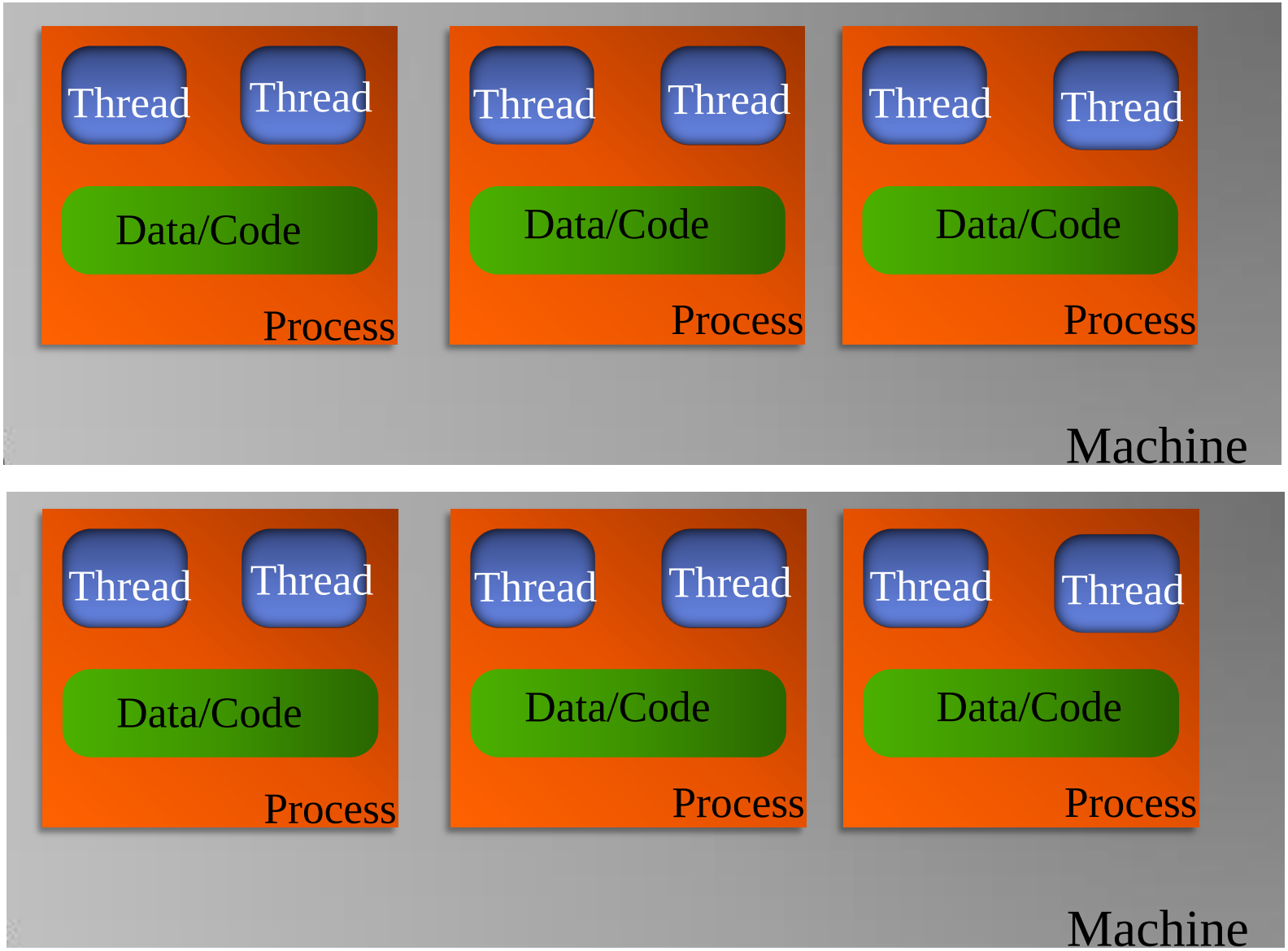


Basics: Multiple Thread, Multiple Process, Single Machine

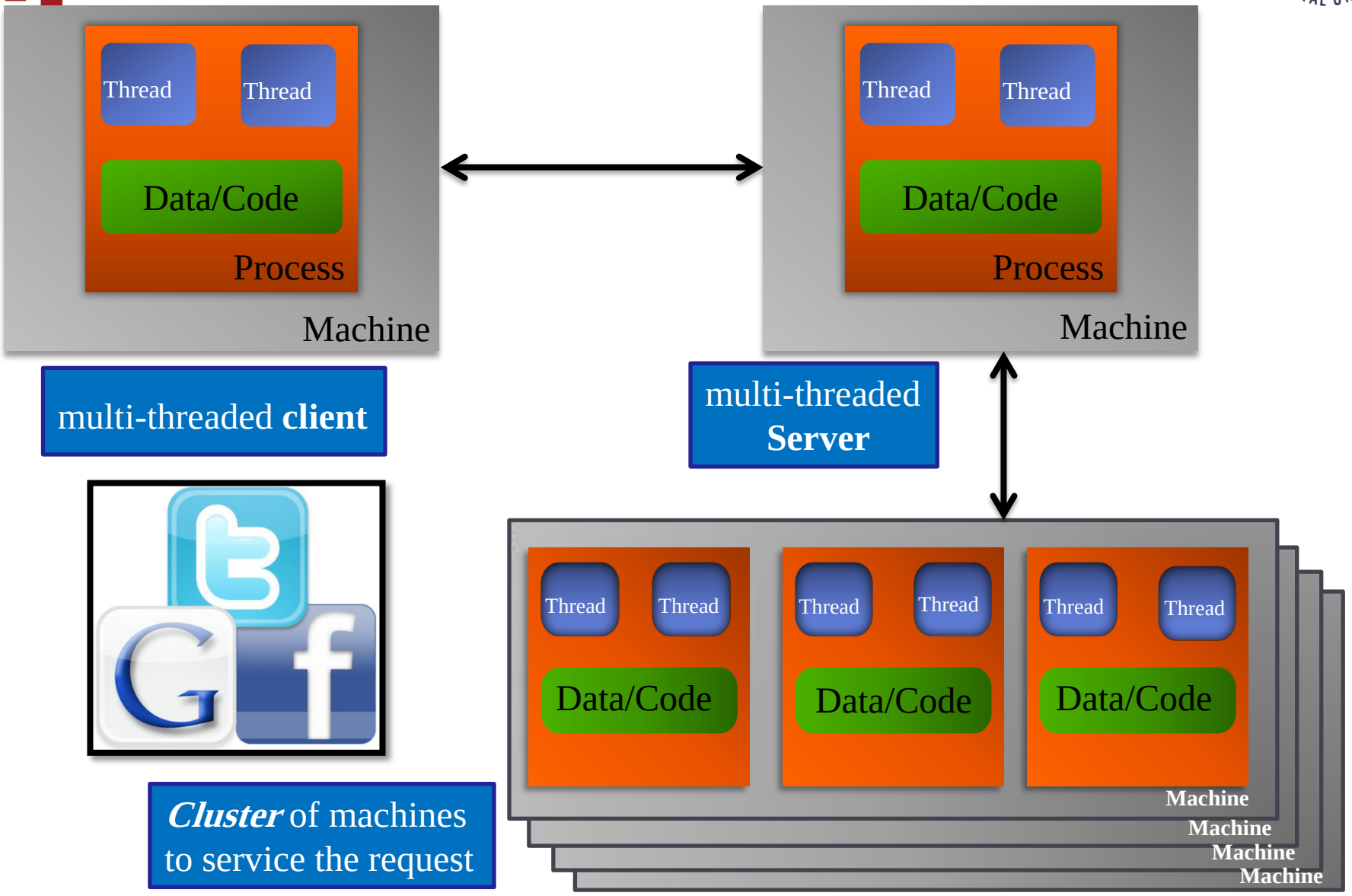


Note: You can have way more than just two threads per process.

Basics: Multi- Everything



Now… we might refer to this as "an application



Consider Chrome

- Google browser:
  - multi-process (one process per tab) and
  - multi-threaded (multiple threads handle loading of content within each process)
- Some of the advantages they cite for this design
  - **stability**
    - single-process, multi-threaded browsers are vulnerable to having a crash in one tab bring down the entire browser
  - **speed**
    - multi-process browsers can be more responsive due to OS support
  - **security**
    - browsers are easier if malware loaded in one tab can grab information contained in another tab; much harder to grab information across processes

Fundamental Abstraction

- Process
- … *aka* Task
- … *aka* Thread
- … *aka* Job
- … *aka* [other terms]

Definition

- *Process* (generic): A *particular* execution of a *particular* program.
  - Requires time, space, and (perhaps) other resources
- Separate from all other executions of the same program
  - Even those at the same time!
- Separate from executions of other programs

Process (continued)

- Can be
  - Interrupted
  - Suspended
  - Blocked
  - Unblocked
  - Started or continued
- Fundamental *abstraction* of all modern operating systems

Background – *Interrupts*

- A mechanism in (nearly) all computers by which a running program can be suspended in order to cause processor to do something else
- Two kinds:–
  - *Traps* – synchronous, caused by running program
    - Intended: e.g., system call
    - Error/Exception: divide by zero
  - *Interrupts* – asynchronous, produced by some other concurrent activity or device.

Hardware Interrupt Mechanism

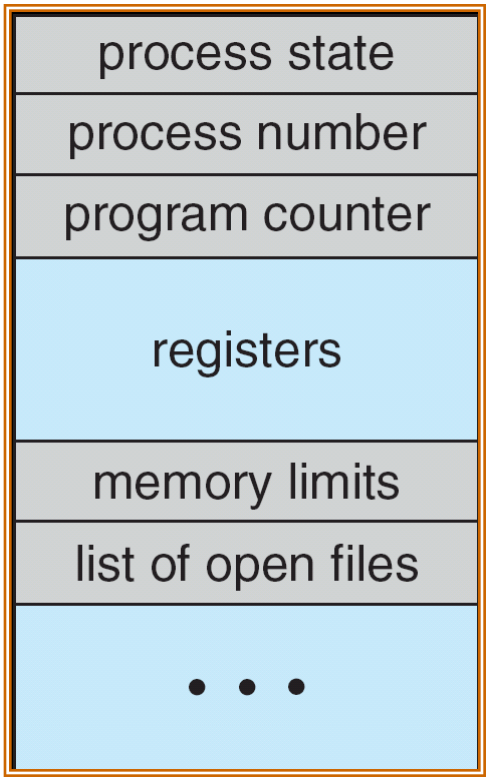
- Upon receipt of electronic signal, the processor
  - Saves current PSW to a fixed location
  - Loads new PSW from another fixed location
- PSW — *Program Status Word*
  - *Program counter*
  - Condition code bits (comparison results)
  - Interrupt enable/disable bits
  - Other control and mode information
    - E.g., privilege level, access to special instructions, etc.

Information the system needs to implement a process

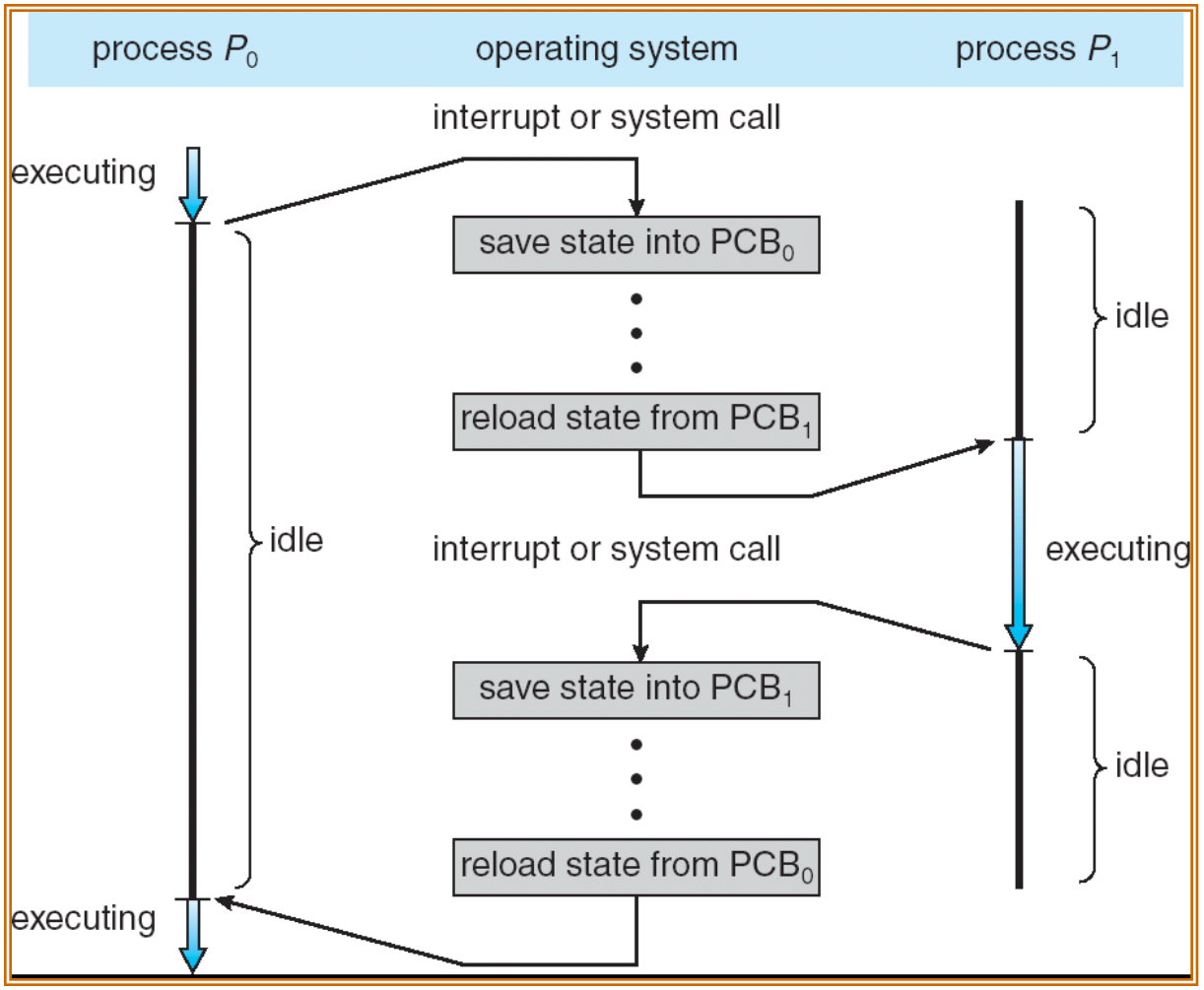
- PSW (program status word)
  - Program counter
  - Condition codes
  - Control information – e.g., privilege level, priority, etc
- Registers, stack pointer, etc.
  - Whatever hardware resources needed to compute
- Administrative information for OS
  - Owner, restrictions, resources, etc.
- Other stuff …

Process Control Block (PCB)

(example data structure in an OS)



Switching from Process to Process



Definition – *Context Switch*

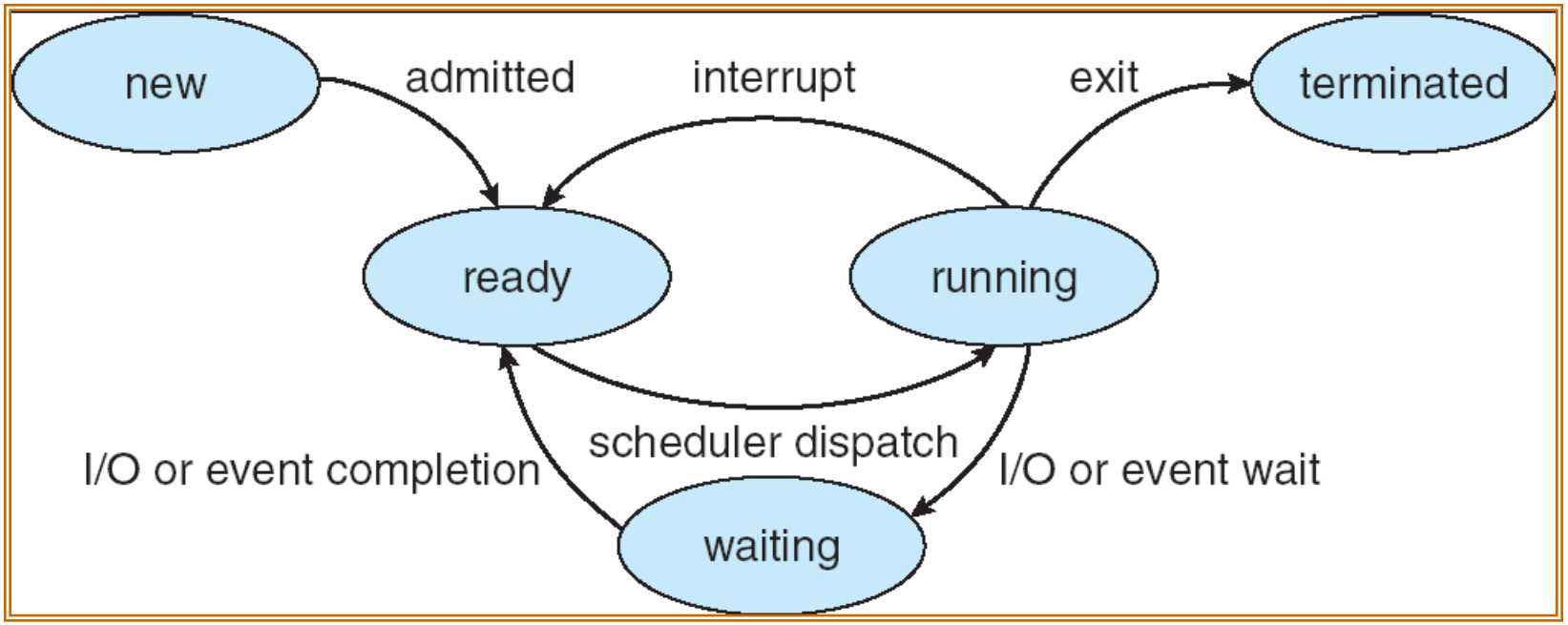
- The act of switching from one process to another
  - E.g., upon interrupt or some kind of wait for event
- Not a big deal in simple systems and processors
- *Very big deal* in large systems such
  - Linux and Windows
  - Pentium 4, etc.

Many microseconds!

Result

- A very clean way of thinking about separate computations
- Processes can *appear* be executing in parallel
  - Even on a single processor machine
- Processes really *can* execute in parallel
  - Multi-processor, multi-core, or multi-threaded hardware

Process States



Timer Interrupts

- Can be used to enforce “fair sharing”
- Current process goes back to *ReadyQueue*
  - *After* other processes of equal or higher priority
- Simulates concurrent execution of multiple processes on same processor

Definition — *Scheduling*

- The art and science of deciding *which* process to dispatch next …
- … and for how long …
- … and on which processor

Topic for later in this course

The Fundamental Abstraction of the OS

- Each process has its “virtual” processor
- Each process can be thought of as an independent computation
- On a fast enough physical processor, processes can look like they are really running concurrently


Questions?