Parallel and concurrent programming

2. Programming Model

Michelle Kuttel

parallel concurrent

Amdahl's Law thread of control

PROTECTION data race synchronization

THREAD SAFETY correctness MUTUAL EXCLUSION

locks

liveness

DEADLOCK starvation

HIGH PERFORMANCE COMPUTING

EXECUTORS thread pools timing

Overview: How to write parallel programs

To write a parallel program, you (the programmer) need new **primitives** from a programming language or library, that enable you to:

- run multiple operations at once
- share data between operations
- coordinate *(a.k.a. synchronize)* operations

Overview: How to write parallel programs

To write a parallel program, you (the programmer) need new **primitives** from a programming language or library, that enable you to:

- run multiple operations at once
- share data between operations
- coordinate *(a.k.a. synchronize)* operations

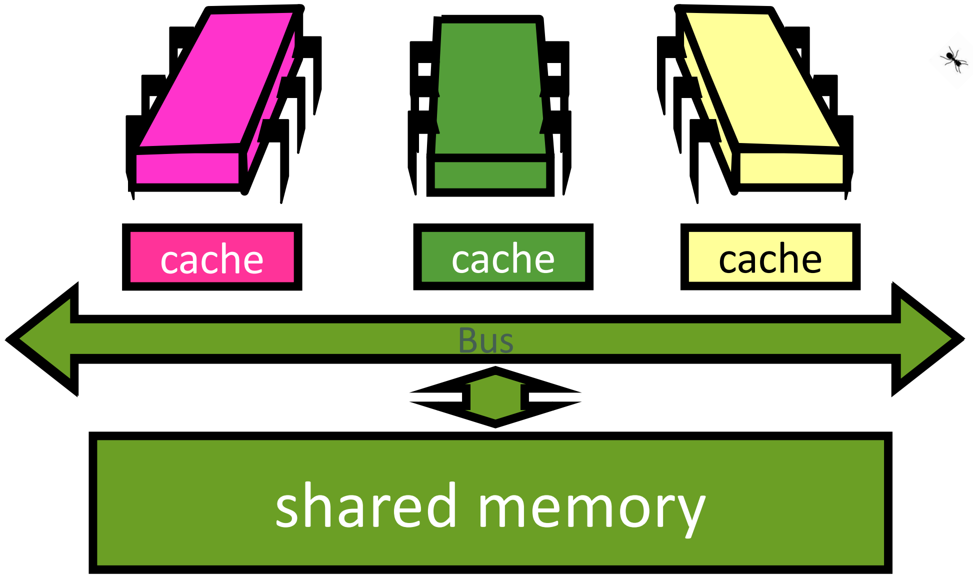
How this works/is done depends on the **parallel programming model** used in the language/library.

Java uses the *Shared Memory* parallel programming model

The Shared Memory Model

All memory is placed into a single (physical) address space.

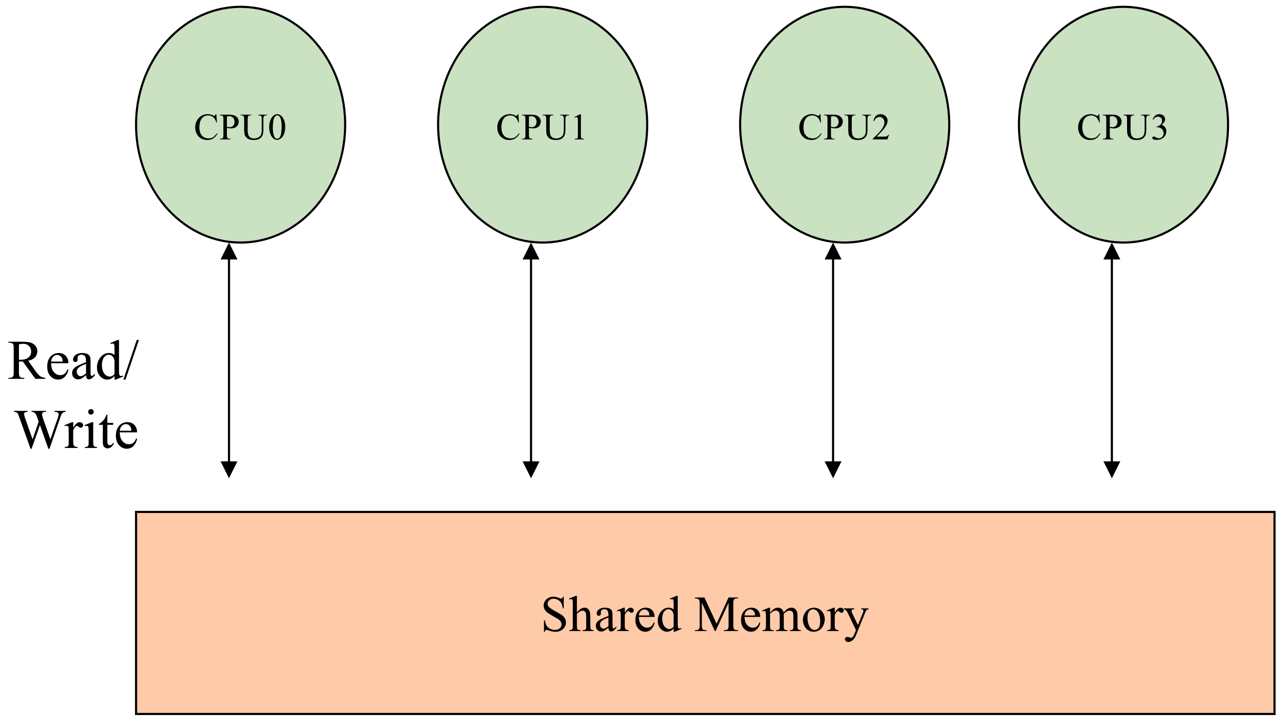
- Processors connected by some form of **interconnection network**
- **Single virtual address space** across **all of memory**. Each processor can access all locations in memory.



from: Art of Multiprocessor Programming

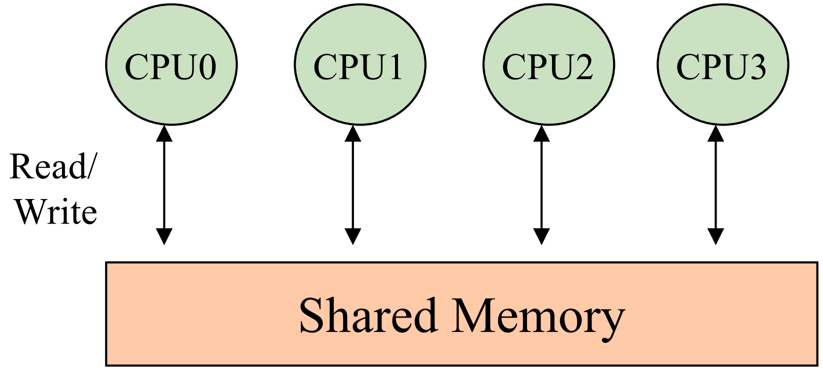
Shared Memory

The **ideal picture** of **shared memory**:



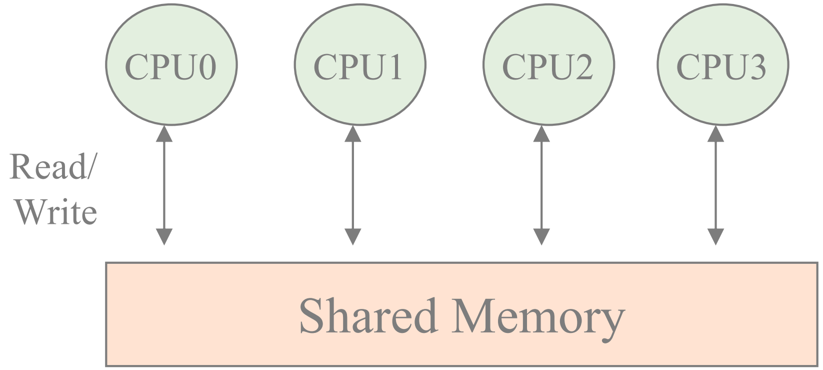
Shared Memory

The **ideal picture** of **shared memory**:



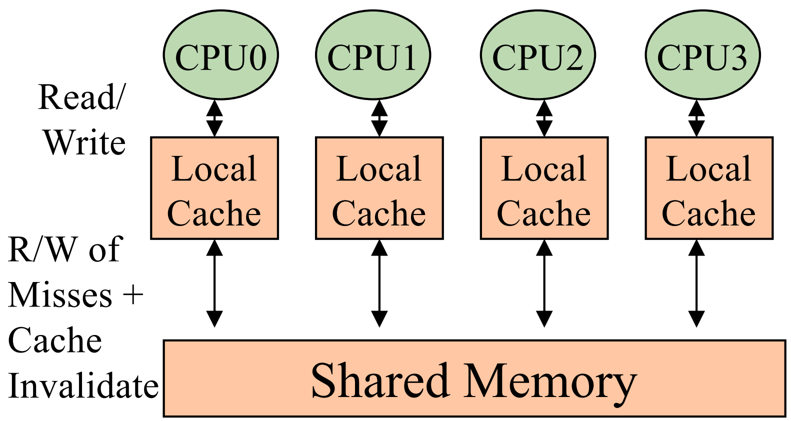
Shared Memory

The **ideal picture** of **shared memory**:

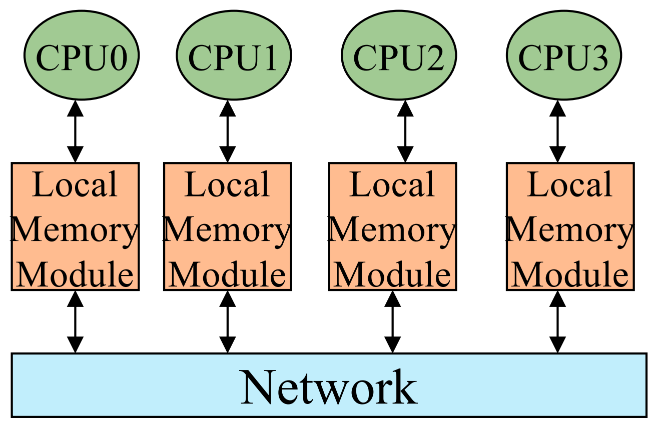


The **actual architecture** of shared memory systems:

Symmetric Multi-Processor (SMP):

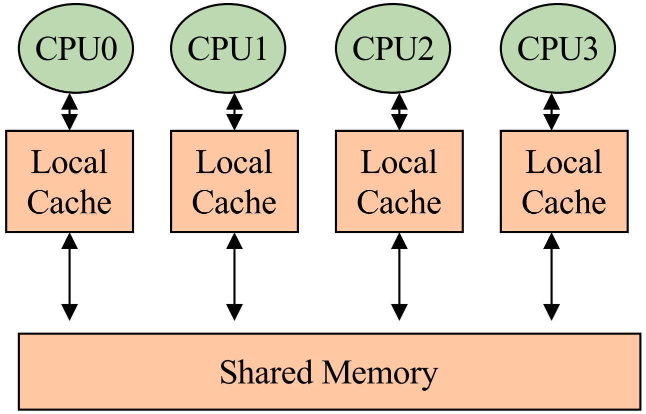


Distributed Shared Memory (DSM):



[Also have Non-uniform Memory Access Symmetric Multi-Processor (NUMA-SMP)]

An aside: cache (Architecture)



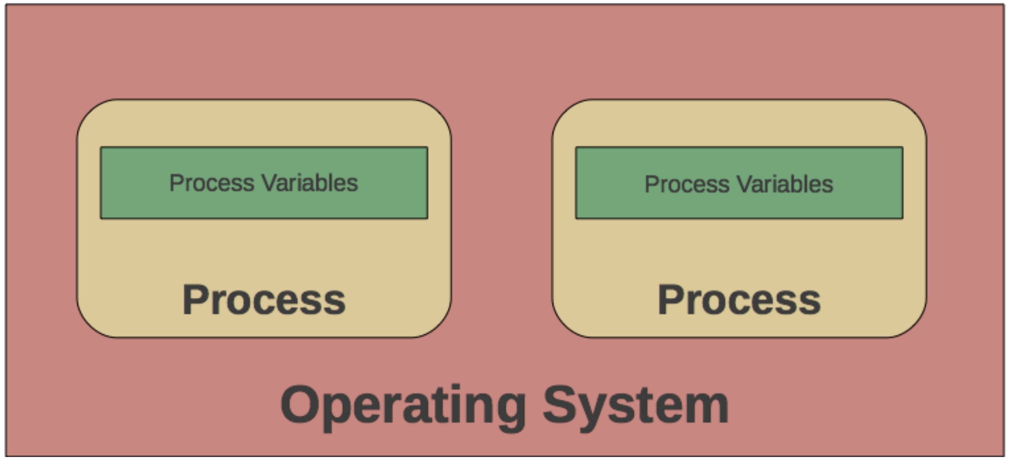
A memory cache, also called a "CPU cache," is a memory bank that bridges main memory and the processor.

- It has faster static RAM (SRAM) chips than the dynamic RAM (DRAM) used for main memory.
- The cache allows instructions to be executed and data to be read and written at **higher speed**.

Can have multiple caches (L1, L2, L3) in modern chips

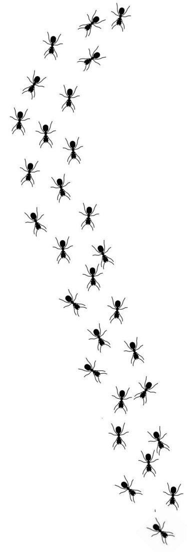
- L1 is the fastest; each subsequent cache is slower and larger than L1, and instructions and data are staged from main memory to L3 to L2 to L1 to the processor.
- On multicore chips, the L3 cache is generally shared among all the processing cores.

A Process (operating system)



Operating system unit of resource allocation both for CPU time and for memory.

- A process is represented by its **code**, **data** and the **state of the machine registers**.
- data of the process divided into
  - global variables and local variables
  - organized as a **stack**.
- Generally, each process in an operating system has its own address space
  - entirely separate entities



It is hard to obtain parallelism or concurrency with separate *processes*… (why?)

… so operating systems created *threads*.

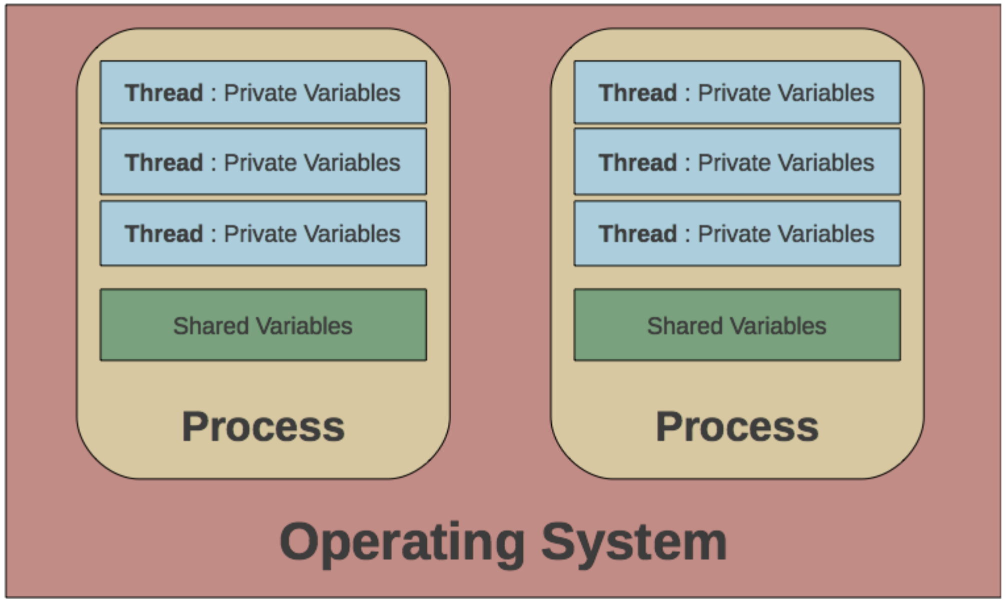
Thread

Process given internal concurrency with multiple *lightweight processes* or ***threads***.

- multiple ***threads of control***

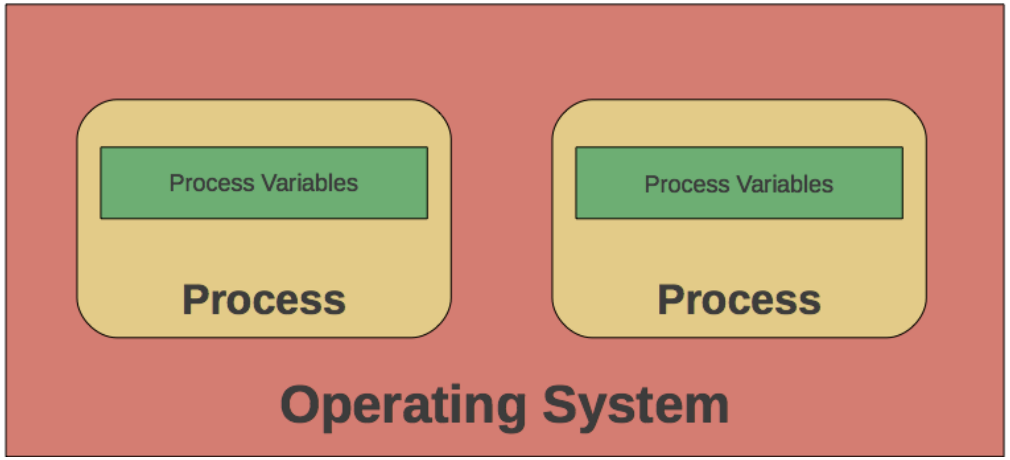
- a process with multiple (lightweight) threads of control has multiple stacks
  - one for each thread.

but access to **shared memory** too.

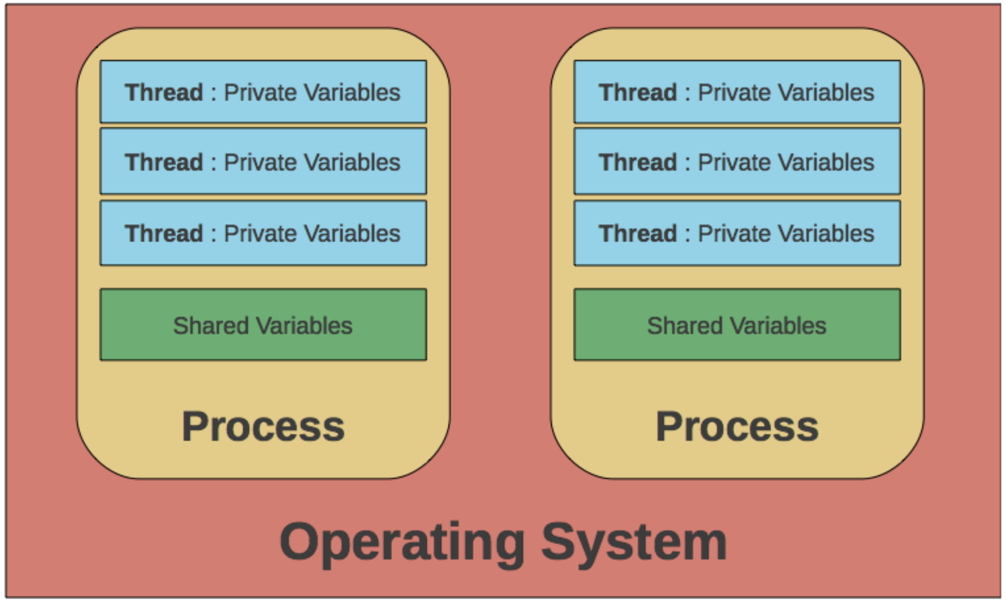


Processes versus threads

Process memory model



Thread memory model



Graphic adapted slides obtained from : www.Intel-Software-Academic-Program.com

What is a parallel program?

The **shared memory model** has multiple **explicit threads** running concurrently.

Threads can:

- perform **multiple computations** in parallel;
- perform separate **simultaneous activities;**
- **communicate easily** and **implicitly** with each other through **shared memory.**

  (but this is dangerous if you don't protect your variables correctly)

* This is true for the shared memory model of parallel computing discussed in this module.

Programming Model: Sequential program state

A running serial program has

- One *program counter* (current statement executing)
- One *call stack*
  - each stack frame holds the local variables for a method call that has started but not yet finished.
  - Calling a method pushes a new frame and returning from a method pops a frame.
  - Call stacks are why recursion is not “magic.”
- *Objects.* Object are created by calling **`new`**. We call the memory that holds all the objects the *heap.*

  (nothing to do with data structure called a heap)
- *Static fields* of classes.

Programming Model: Shared memory

- Each thread has its own program counter, call stack and local variables
- All threads share one collection of objects and static fields
- Static fields of classes are also shared by all threads.
- Threads communicate through shared objects (implicit communication)
  - To *communicate*, **write** somewhere another thread **reads**

Slide adapted from: Sophomoric Parallelism and Concurrency, Lecture 1

Sequential Computation

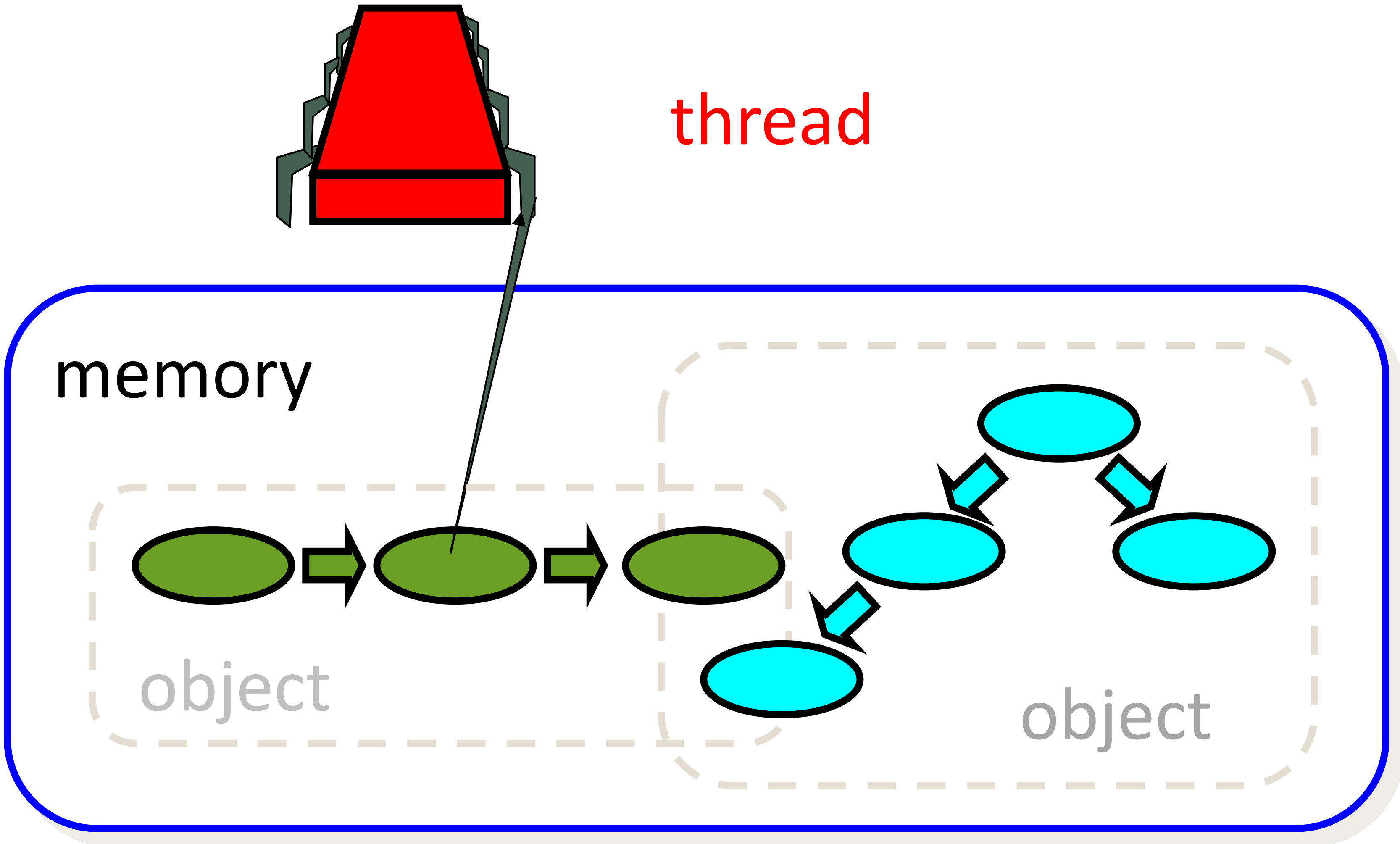
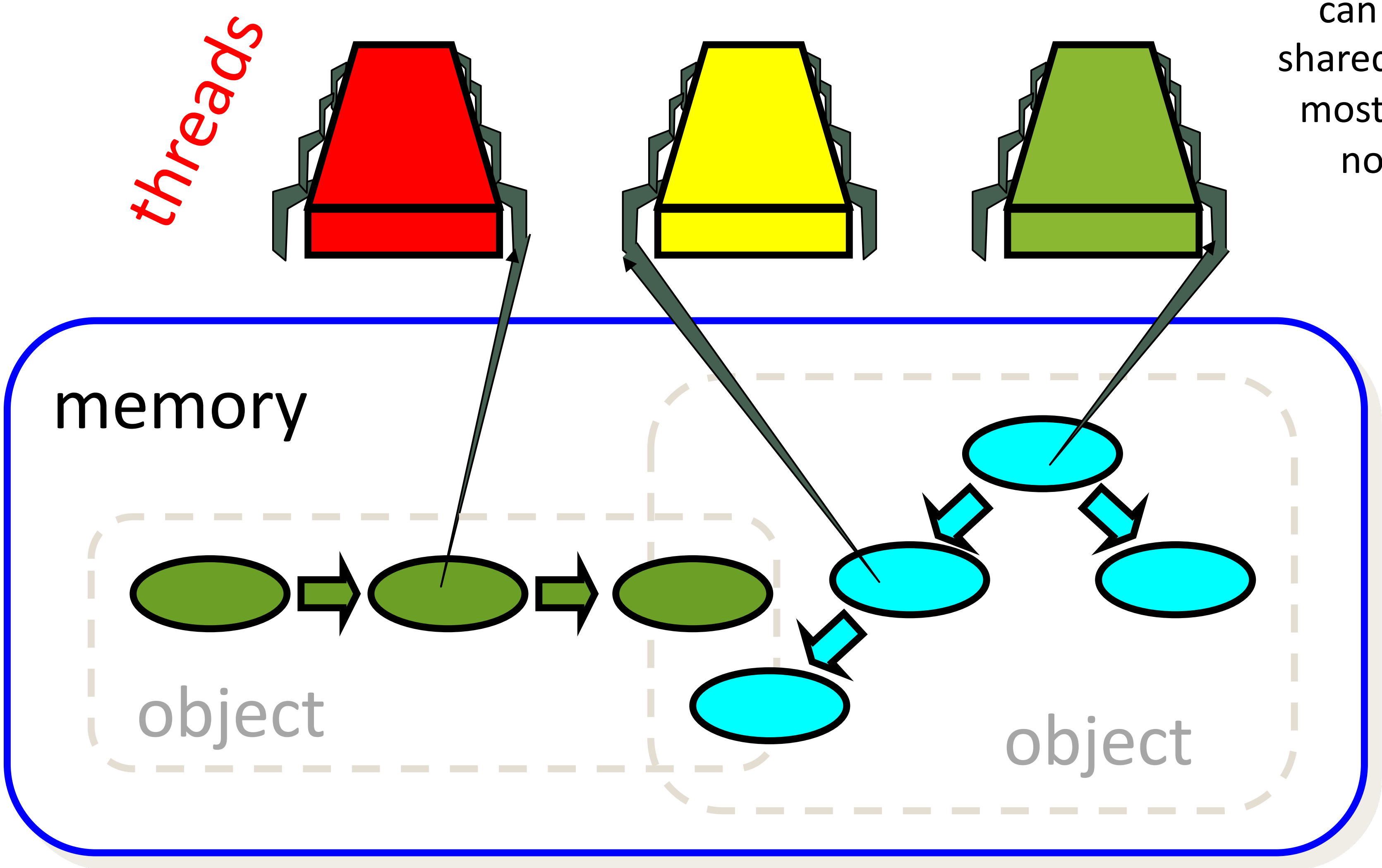


image from slide set: Art of Multiprocessor Programming, Maurice Herlihy

Concurrent Computation



Any object can be shared, but most are not.

image from slide set: Art of Multiprocessor Programming, Maurice Herlihy

How threads run

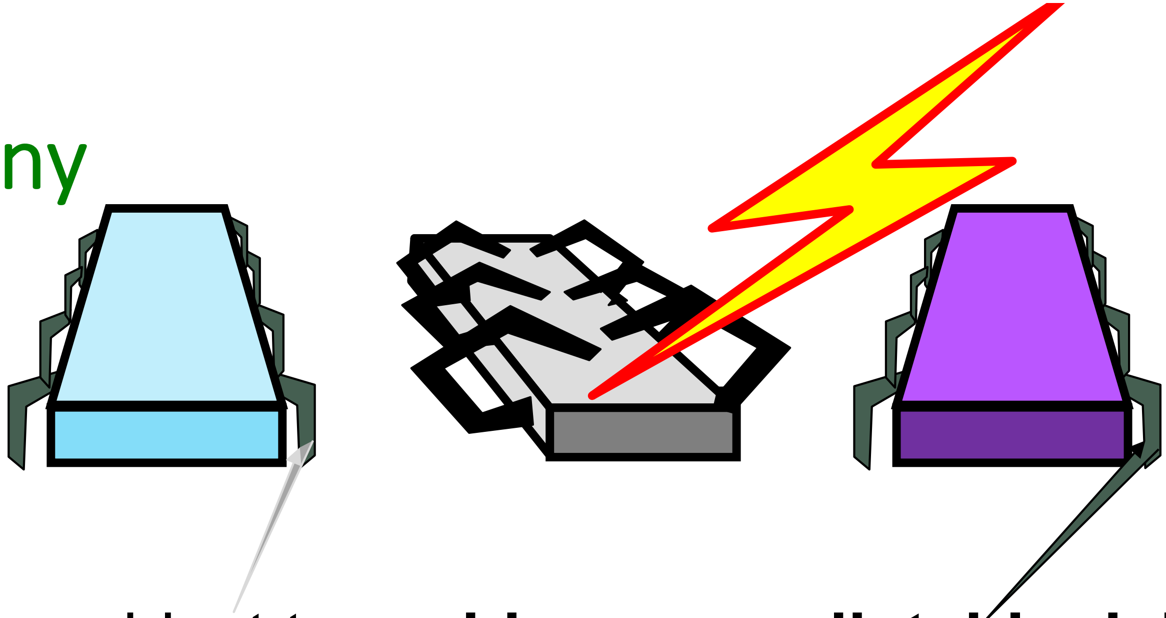
As the programmer, **you create threads**.

- The **Operating System scheduler** determines how and when those threads are run on the available processors
  - Unless you are writing a scheduler, **you don't have control over this**
  - You don't know how many **processors/cores** your program will use

    (though you can guess).
  - You don't know **the order** in which threads will execute

    (you can't even guess).

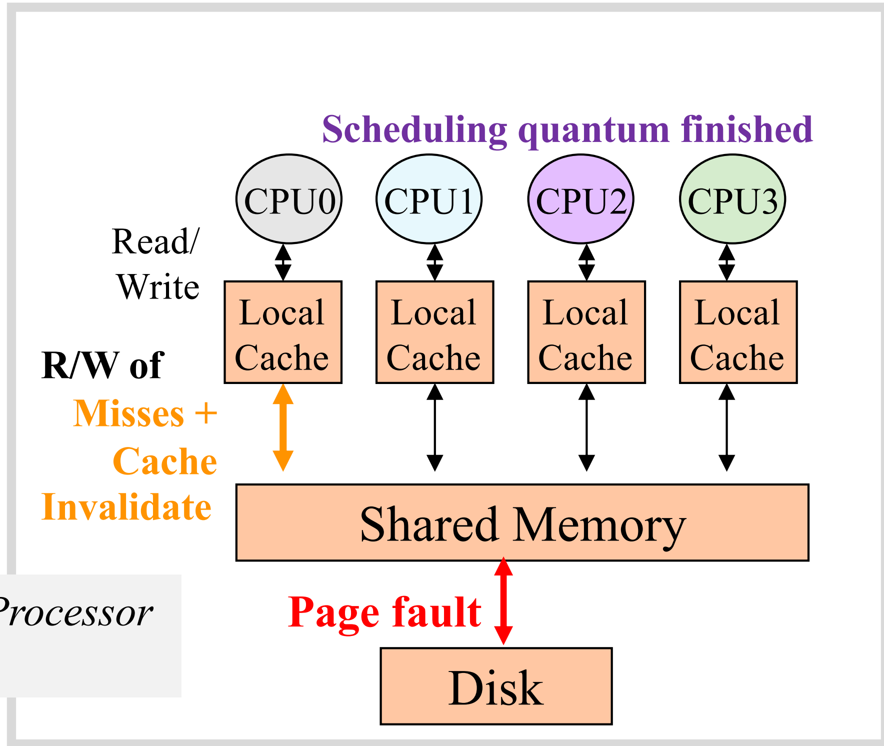
Asynchrony



Threads are subject to **sudden unpredictable delays:**

- **Cache misses** (*short*)
- Page faults (*long*)
- Scheduling quantum finished

  (*really long*)

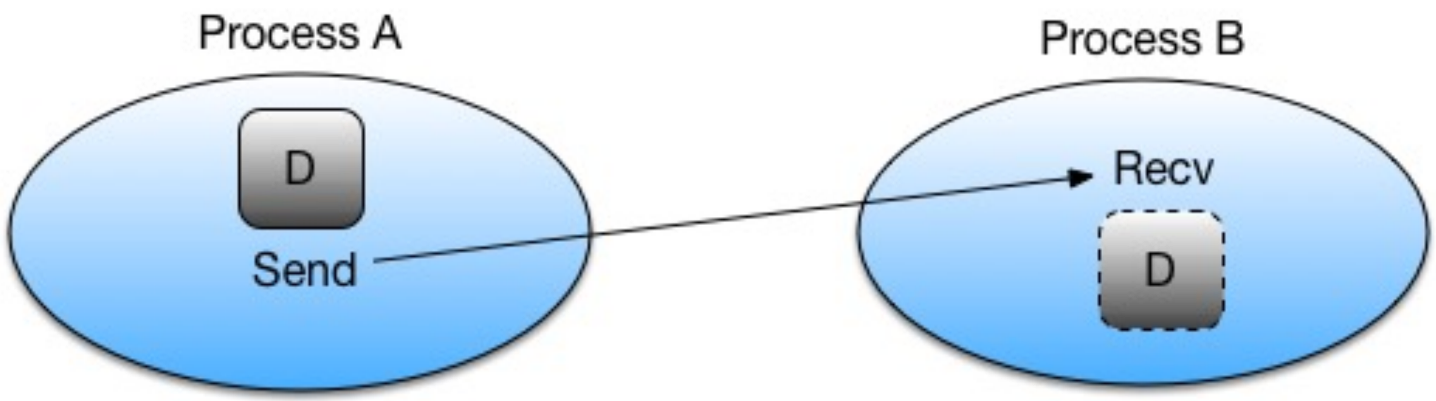


Symmetric Multi-Processor (SMP)

Other parallel programming models

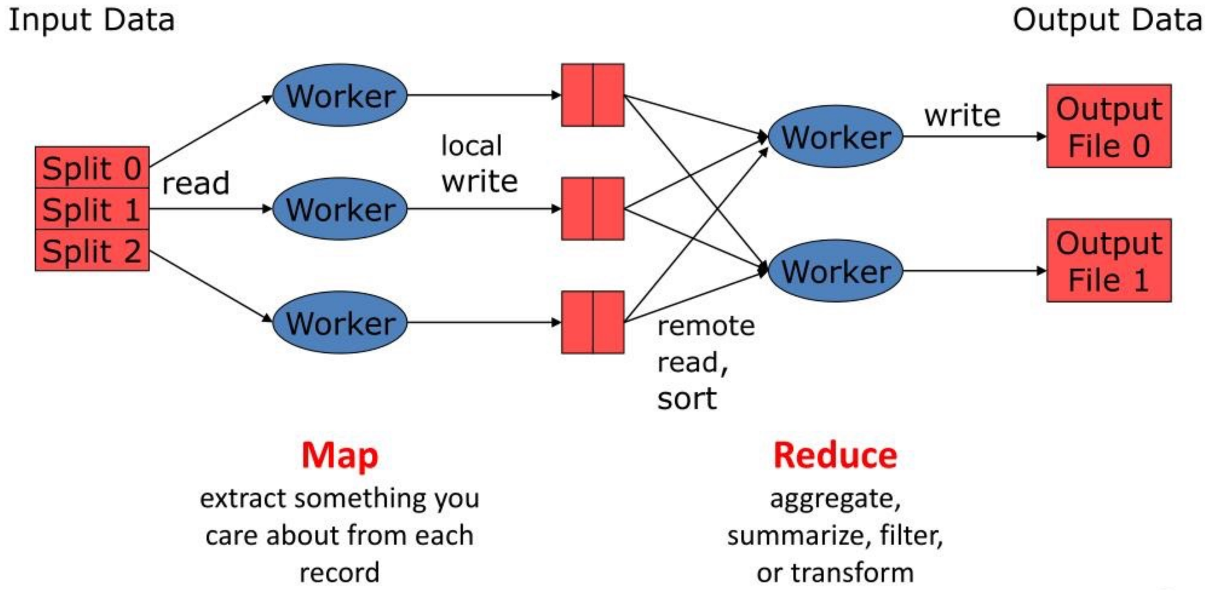
We focus on **shared memory**, but several **other models** exist. Common alternatives are:

- **Message-passing:**
  - Explicit **threads/processes**, each with their own objects/data.
  - Communication is via explicitly **sending/receiving messages**, containing **copies** of the data (share nothing)
  - Most common model on HPC systems (though usually in a hybrid with shared memory)



Other parallel programming models

- **Map reduce:**
  - Data parallelism concept from functional programming languages like LISP
  - Have **primitives** for things like “apply function to every element of an array in parallel”.
    - details of the **underlying parallelization** are **hidden** from the programmer, provided you can express your program using the available primitives
  - MapReduce was developed by Google and the programming model has since been adopted by many software frameworks, e.g. Apache’s open-source Hadoop



Other parallel programming models

Golang

- Go-routines,
  - Go runtime maps these onto operating system threads
- Channels – used for communication between Go-routines

```
package main

import "fmt"

func main() {
   fmt.Println("Hello, World!")
}
```