Lancaster University

SCC.211 Operating Systems

Lecture 3 – Programs, Processes, Processors, and OS

Amit Chopra
School of Computing & Communications, Lancaster University, UK
amit.chopra@lancaster.ac.uk

Lancaster University

The three central concepts in concurrent programming

Processor

Hardware device that executes machine instructions

Program

Instruction sequence defining potential execution path

Passive description of what you would like to happen

Stored on disk/secondary memory

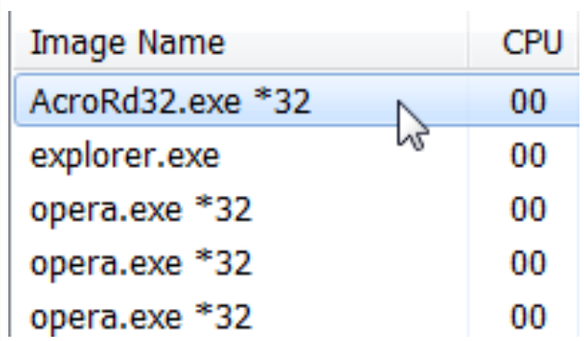
Process

Active system entity executing associate program(s)

or; Program *in execution* on a processor

Resides in primary memory, removed on reboot

(We'll assume process, thread, and task are equivalent for now)

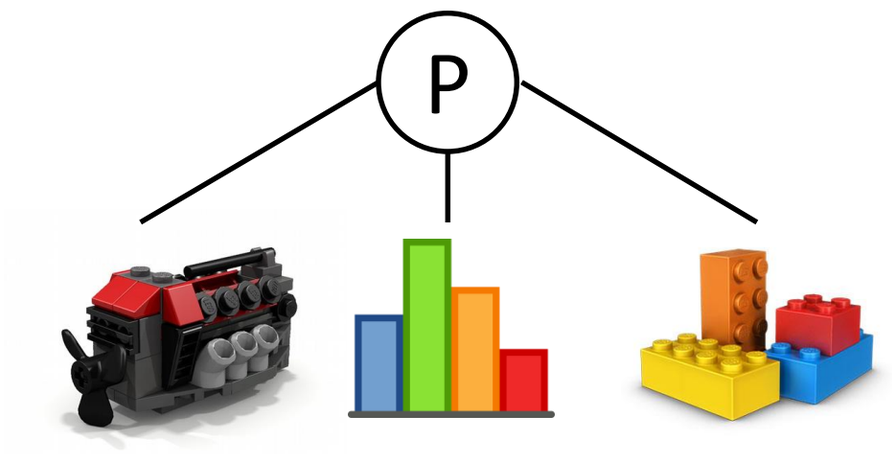


Processes and Programs

Why is a process not a program?

1) **Program may be executed by multiple processes at the same time**
   - Wordpad opened twice as separate processes of the same program
   - 1 program, multiple processes

2) **Process might run one program, and then another**
   - **gcc** pre-processor is a different program (executable file) from the compilers syntax analyser and code generator
   - The pre-processor, syntax and analyser, and code generator are executed one after the other by the same process
   - 1 process, multiple programs



Lancaster University

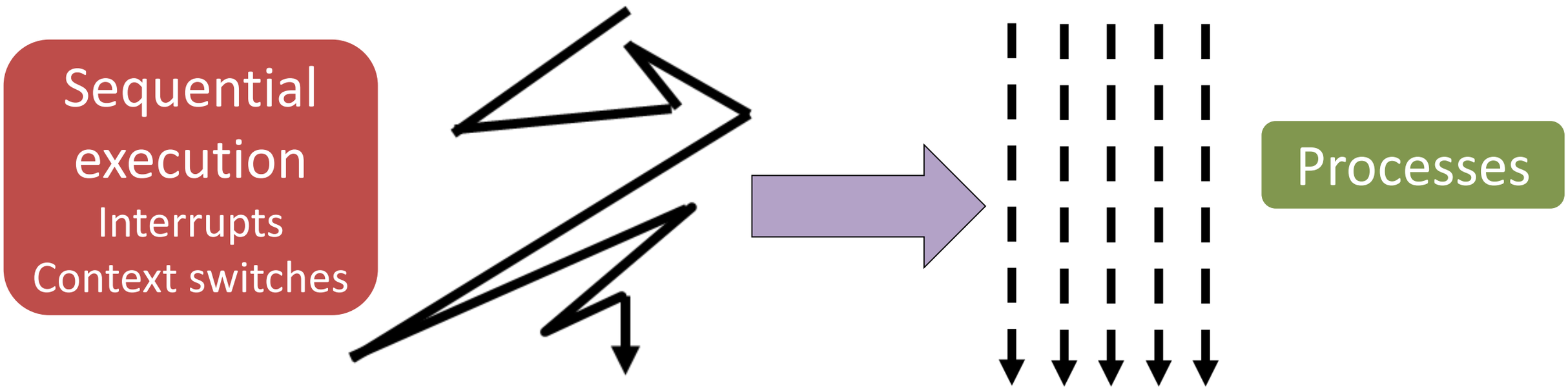
Processes abstract over…

1) Processors

- Can have multiple processes regardless number of processors

2) "Disturbed" sequential execution

- Interrupt processing: Handling external events (keypresses, clock ticks)
- **Context switch**: Execution jumps to another part of memory



Context Switching

Lancaster University

Need to store context information for a switched-out thread so can later resume exactly where it left off

A thread's context (minimally) comprises

- **Program Counter (PC):** Instruction in the program (or function) that the thread will execute next
- **Stack Pointer (SP):** Address at the top of the thread's call stack. The call stack remembers the sequence of function calls the thread is making as it executes program (each thread needs own stack)

Context may also include other info

(Three-step context-switching sequence)

Lancaster University

1

De-schedule currently-running thread

–Save PC and SP CPU registers of current running thread

–Required so that thread resumes execution exactly where left off

2

Scheduler selects 'best' ready thread to run next

- Time-slicing, priority, starvation
- Hardware architecture, etc.

3

Resume newly-selected thread

- Restore register contents back to PC & SP registers
- Thread resumes where it last left off (PC is loaded last)

Concurrency vs. Parallelism

- Processes are always concurrent
- …but are not always parallel

Parallel

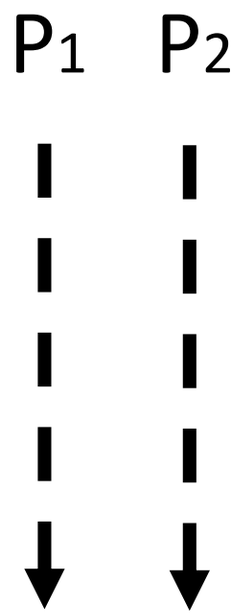
Multiple ($n > 1$) processes executing simultaneously.

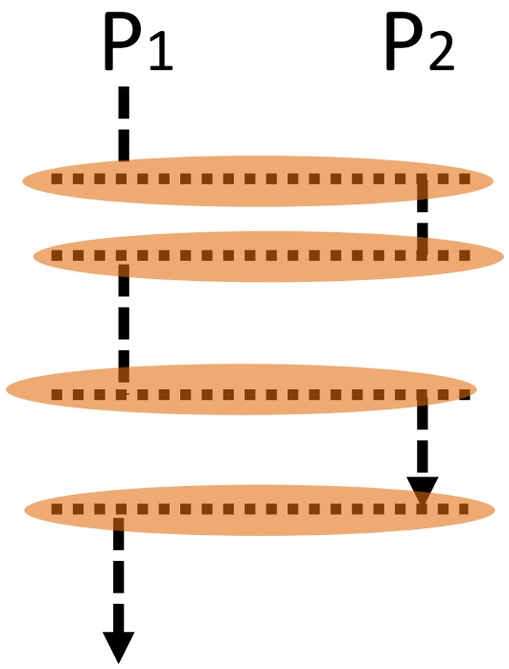
All executing at a given instant.

Concurrent

$n > 1$ processes are underway simultaneously.

< n may execute at given instant





At each context switch, a **scheduler** decides which process is executed next

Question

Lancaster University

Which processes finishes first?

Process 1

```
while(i < 10)
{
  i = i + 1;
  println(i);
}
println("Process 1 completed.");
```

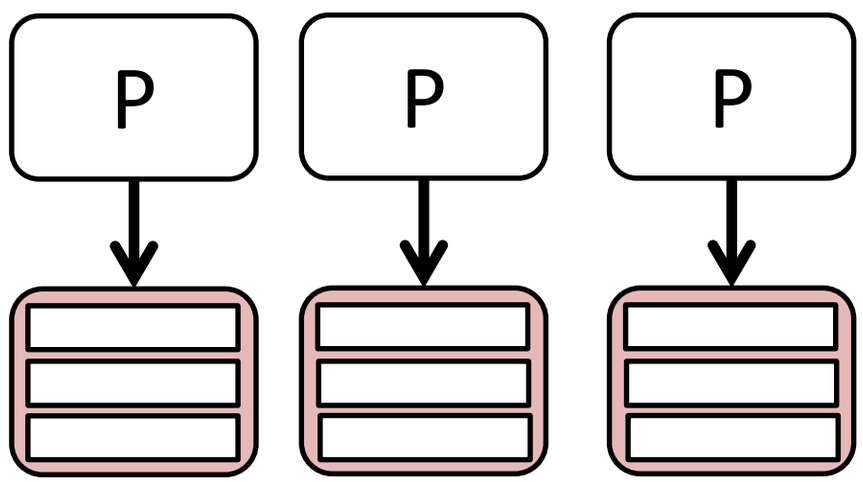
Process 2

```
while(i < 10)
{
  i = i + 1;
  println(i);
}
println("Process 2 completed.");
```

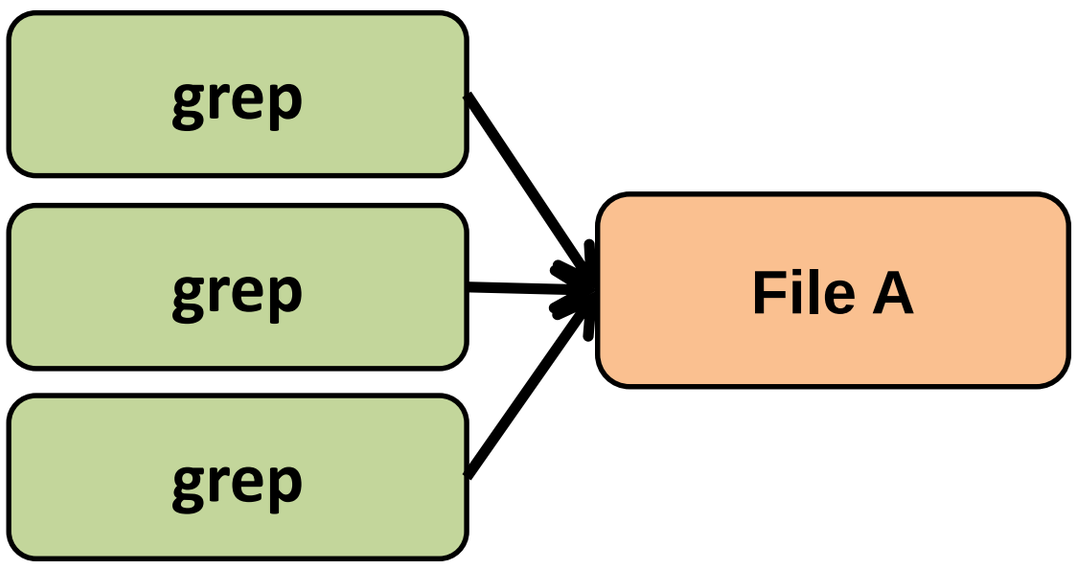
Lancaster University

When do we not have to worry about concurrency?

1) No shared data or communication



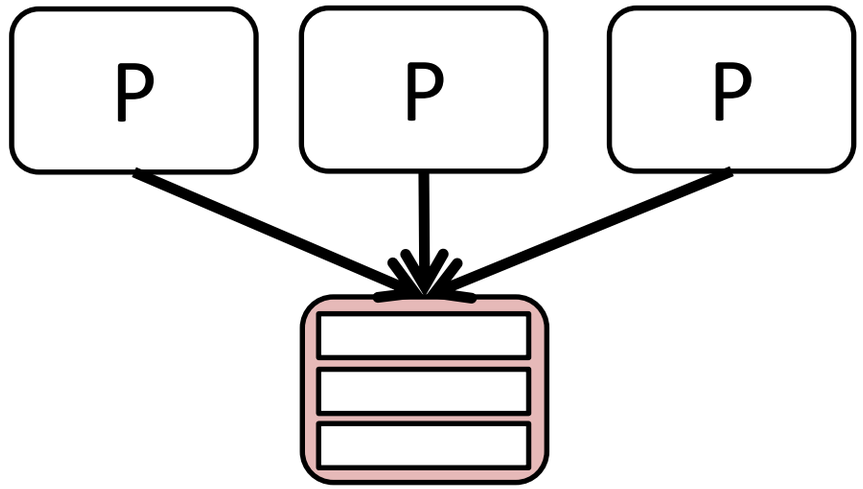
2) Read only data (constant)



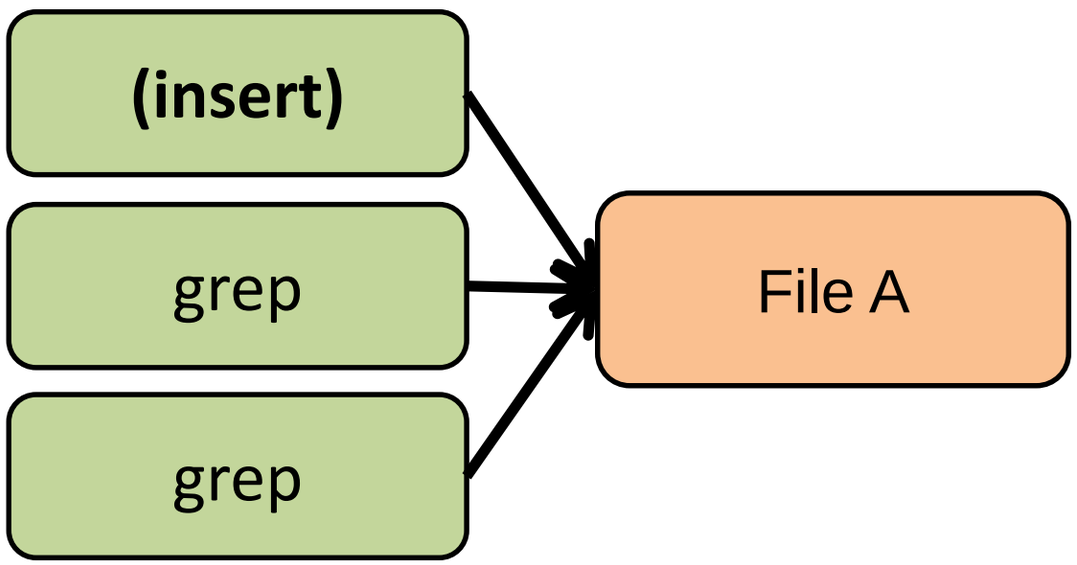
Lancaster University

When should we worry about concurrency?

Threads use a **shared resource** without **synchronization**



One or more threads **modify** the **shared resource**



Concurrency Context: Leveraging Hardware and Software

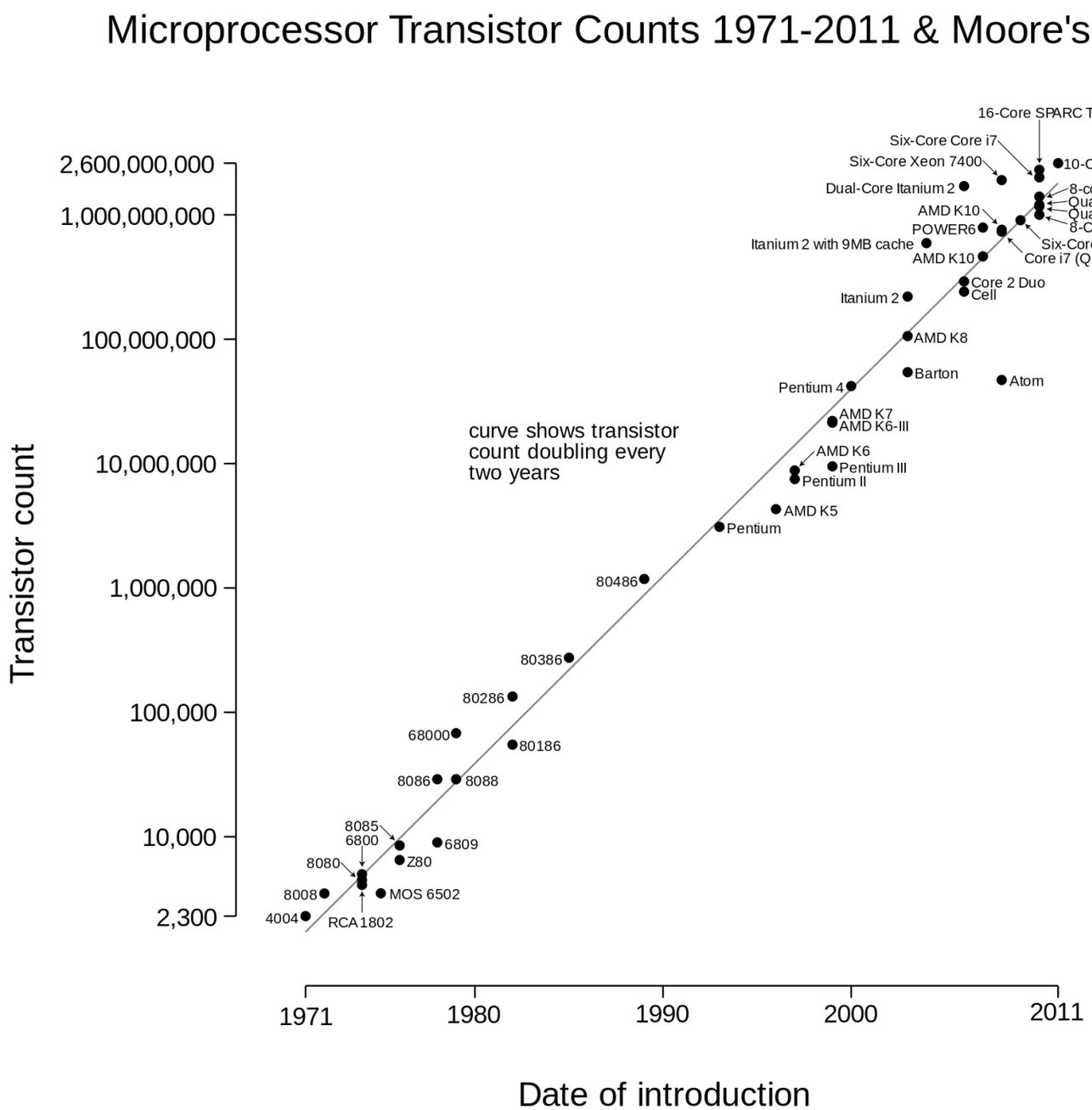
Lancaster University

Commoditization

Massive improvement in technology

Lancaster University

Multi-core processors: Each core can run a thread concurrently with other cores



Number of transistors in dense integrated circuits doubles every 2 years

…Not for much longer!

en.wikipedia.org/wiki/Moore's_law

Leveraging Hardware and Software

Lancaster University

Multi-threaded Operating Systems (OS) and Virtual Machines (VM)

Windows

IEEE POSIX

Multi-threaded middleware

Microsoft .NET

Leveraging Hardware and Software

Lancaster University

Multi-threaded Middleware

Multi-threaded OS & VM

Multi-core Processors

Commoditization

Knowing concurrency is a requirement for efficient multi-thread and multi-core systems

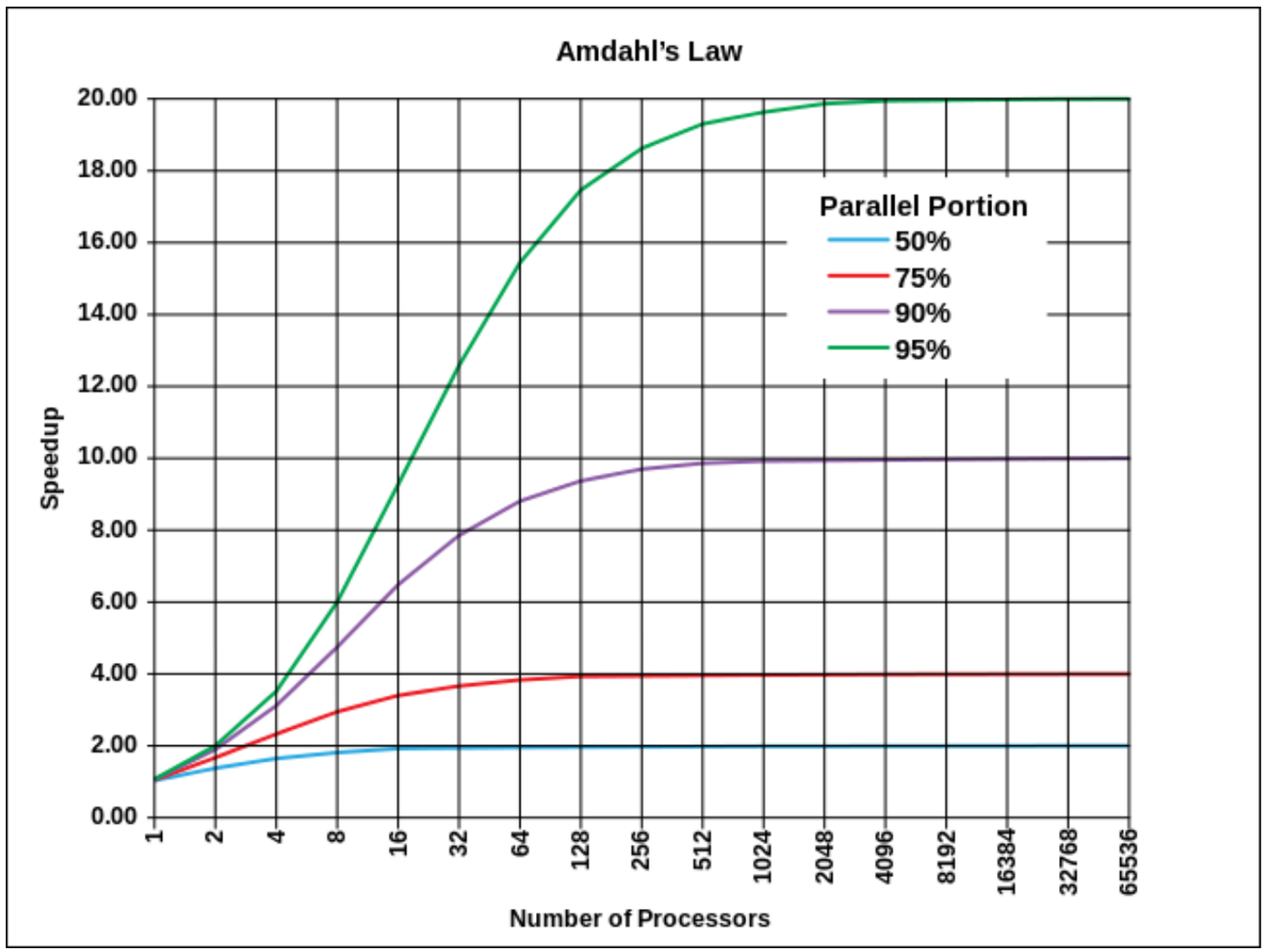
Limit to Performance: Amdahl's law

Lancaster University

Program speedup by adding more processors is limited by the serial part of the program

e.g. If 95% of program (its runtime) can be parallelized, theoretical maximum speedup is x20

1/(1-p) where p is proportion of the program that can be parallelized.



en.wikipedia.org/wiki/Amdahl's_law

Complexities of Concurrency

Lancaster University

- **Accidental complexity**
  - Low-level APIs
  - Tedious, error-print, time consuming
  - Non-portable
- **Limited debugging tools**
  - Actual behaviour vs. debug environment
  - Lack of tools to identify and rectify *race conditions*

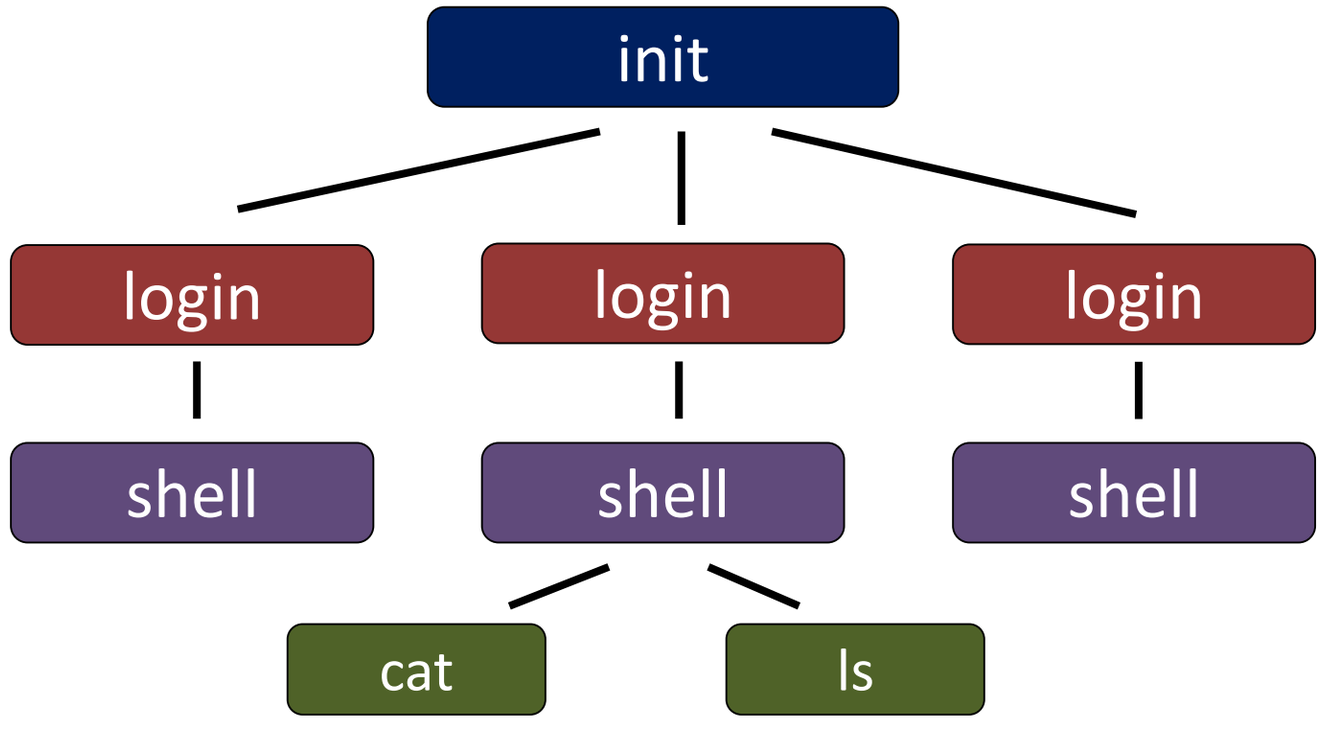
Some System Calls for Unix Processes

Lancaster University

fork()	Create a ‘child’ process
wait()	Block until child process dies
execve()	Make a calling process run a new program
kill()	Send a ‘signal’ to a process
signal()	Set a function to be called on arrival of ‘signal’
pause()	Make calling process block until a ‘signal’ arrives
exit()	Commit suicide
getpid()	Get a process identifier of calling process

Unix Process Hierarchy

- Every Unix process (except one!) has a parent
- Processes may create multiple children (via fork())
- Example: The (traditional) Unix boot procedure
  - Single parentless process runs (init)
  - init reads a file which lists all connected terminals
  - init forks a login process for each terminal
  - If a login process validates a user, it forks a shell process…



Unix-style Processes

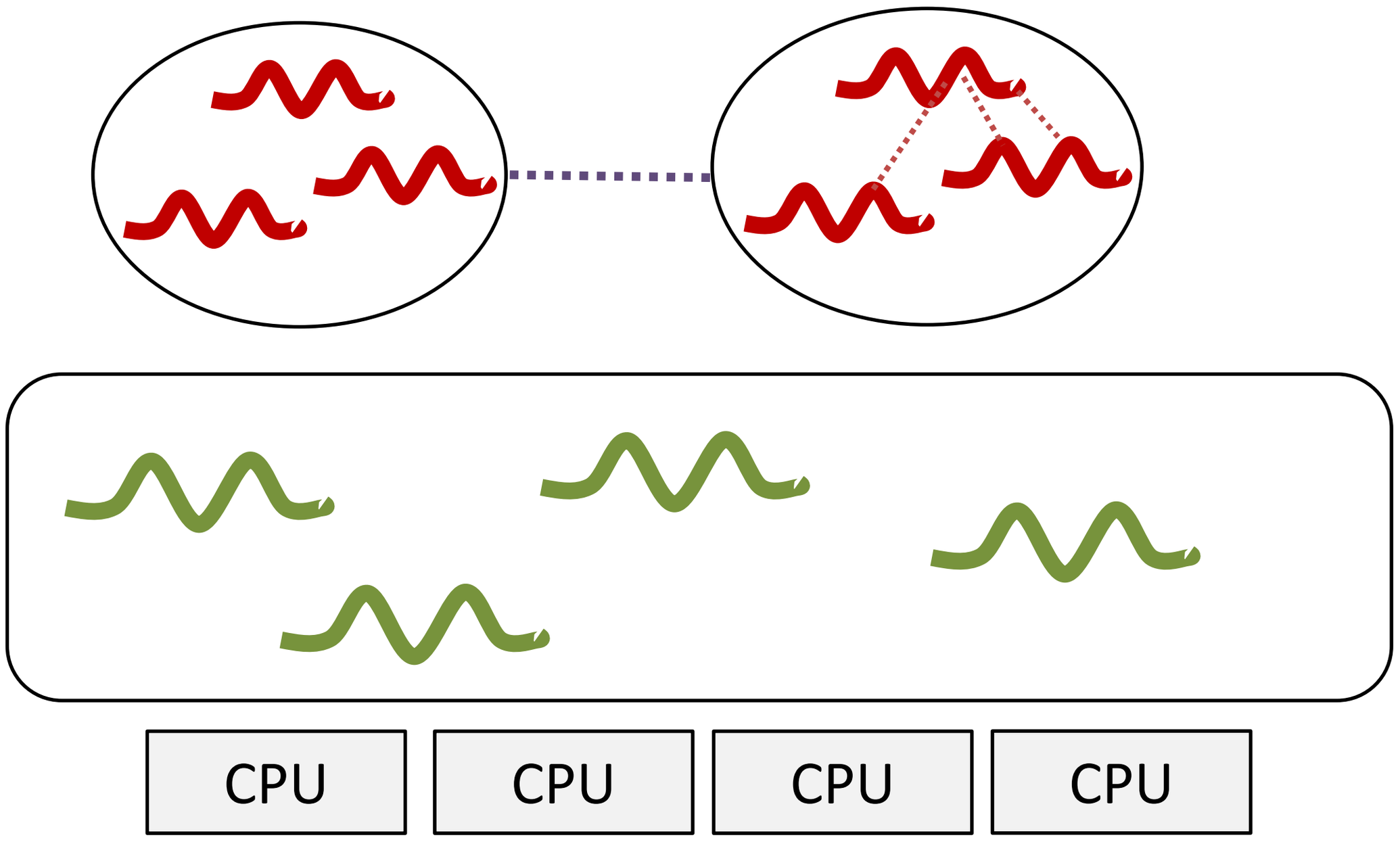
Unix processes combine a thread of execution with a 'private virtual address space'

- Each process is like a 'virtual machine': CPU and memory combined in single abstraction
- User processes are protected from another
  - Good to avoid corruption and security problems
  - Cannot therefore straightforwardly share memory
  - Processes are heavyweight and costly to switch - as well as thread context (PC, SP, flags etc)

Lancaster University

Threads share a common address space

- Can easily cooperate on common tasks
- Lighter weight switch; support fine-grained concurrency



Threads: User vs Kernel vs. Language

Lancaster University

User Threads

- Supported in user-level library
- Kernel only knows about processes
- Scheduled by a per-process, user-level scheduler

Kernel threads

- Supported by OS
  - Scheduled by OS scheduler
- Threads are preemptible by the OS
  - Potentially more concurrent, especially on multicore
  - More predictability useful for real-time applications
  - OS itself is multithreaded
- Supported in almost all modern OS

Threads in language runtime environments (Java)

- Available only in particular language environments
- Might be implemented either as user or kernel threads

User vs. Kernel Threads

Lancaster University

User thread Advantages

- No OS support required
- Cheap context switch
  - Typically **order of magnitude faster** than kernel thread switch
  - (Which itself is an **order of magnitude faster** than process switch…)
  - Easy to offer per-application scheduling policies

User thread disadvantages

- If thread makes a system call that blocks, all other threads in that process also block (all reduce to the one underlying process)
- User threads in same process can't execute on separate CPUs on multicore – as the OS is not user-thread aware

User threads most useful for structuring conceptually concurrent applications (GUIs, RPC servers, *some* web servers)

Each Java App. Launches a JVM

Lancaster University

Inside JVM

Java Stack (created per thread)

Contains local variables, parameters and return values

Heap (one per JVM)

Contains objects, arrays (which are objects!)

Summary

Lancaster University

- **Concurrency occurs in the physical world**
  - However computers need to be told explicitly what to do…
- **Programs**, **processes** and **processors** are different things
- Concurrency **does not** equal parallelism
- Concurrent programming is programming with >1 processes
- Threads can be implemented in user library or supported by OS
  - If library, then not visible to OS