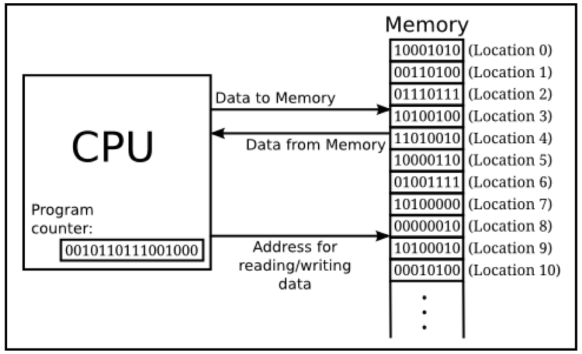
Introduction to shared memory programming

Damian Podareanu
SURFsara
damian.podareanu@surfsara.nl

Questions for audience

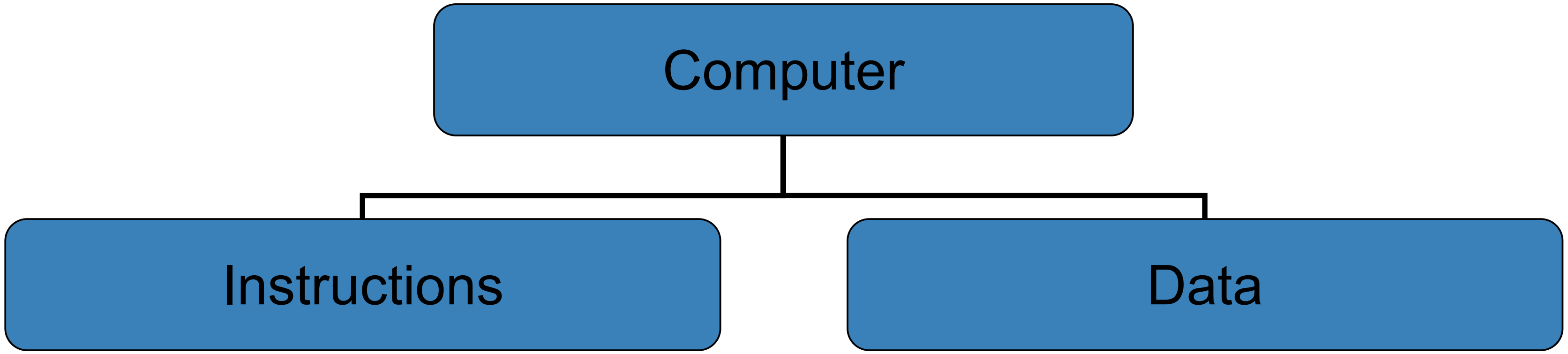
- What is a computer program?
- Does this



- mean something to you?

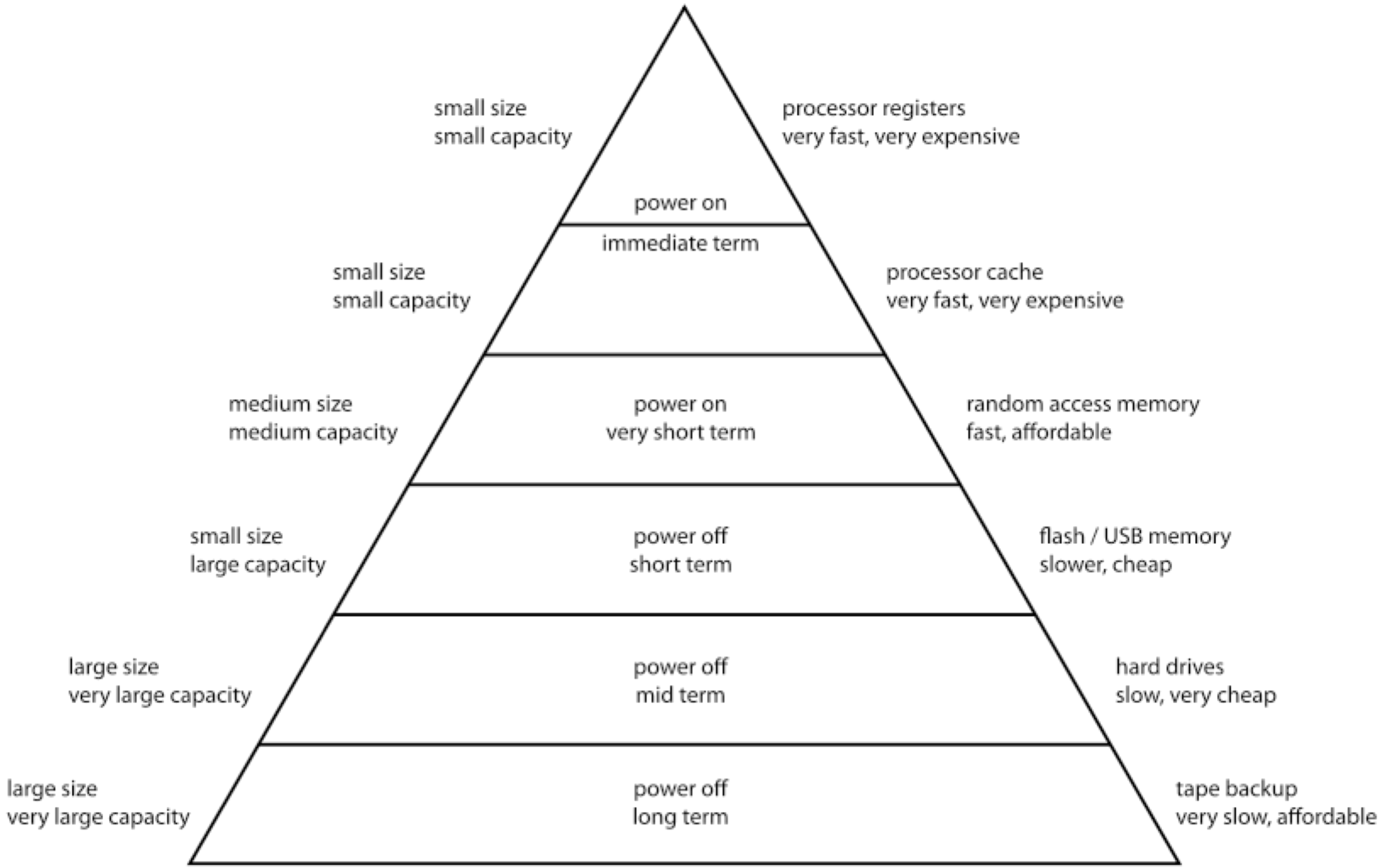
- What are instructions per second?

- Computing: execute instructions that operate on data.



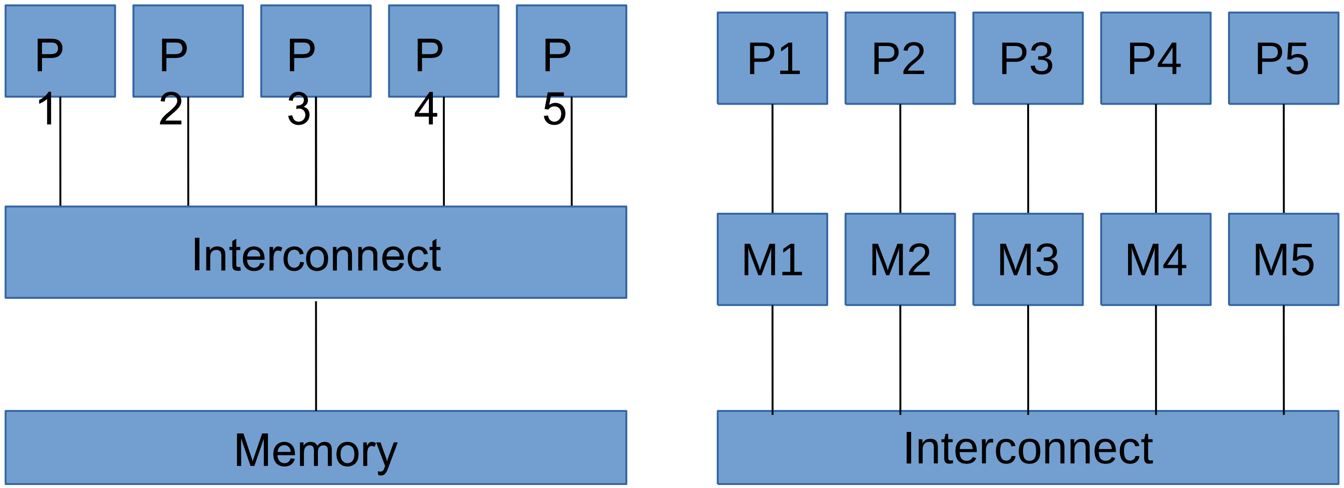
- Flynn's taxonomy (Michael Flynn, 1967) classifies computer architectures based on the number of instructions that can be executed and how they operate on data.

Computer Memory Hierarchy



https://en.wikipedia.org/wiki/Memory_hierarchy

	Parallel Systems	Distributed Systems
Memory	Tightly coupled shared	Distributed memory
Control	Global clock	Explicit synchronization needed
Main focus	Scientific computing	Reliability, scaling, sharing



(De)Motivation for Parallelism and Concurrency

“

[…] give up on parallelism already. It's not going to happen. End users are fine with roughly on the order of four cores, and you can't fit any more anyway without using too much energy to be practical in that space. And nobody sane would make the cores smaller and weaker in order to fit more of them - the only reason to make them smaller and weaker is because you want to go even further down in power use, so you'd *still* not have lots of those weak cores.

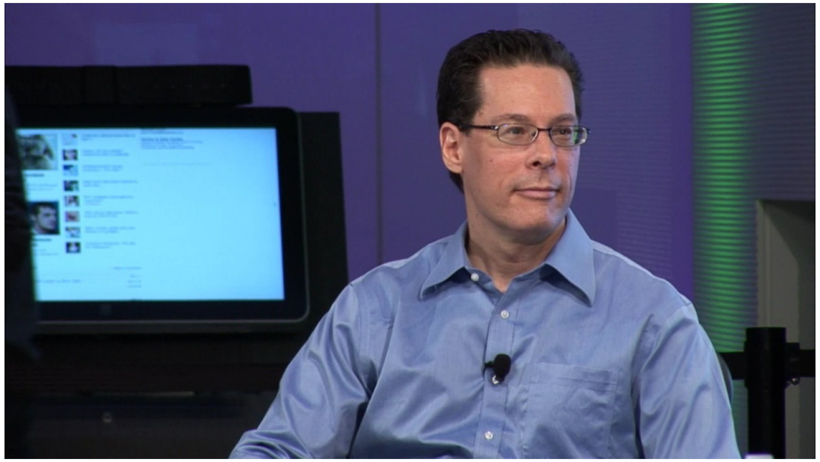
Linus Torvalds
(2014)

If you want to do low-power ubiquotous computer vision etc, I can pretty much guarantee that you're not going to do it with code on a GP CPU. You're likely not even going to do it on a GPU because even that is too expensive (power wise), but with specialized hardware, probably based on some neural network model.

Give it up. The whole "parallel computing is the future" is a bunch of crock.

“

Everybody who learns concurrency thinks they understand it, ends up finding mysterious races they thought weren’t possible, and discovers that they didn’t actually understand it yet after all.



Herb Sutter

chair of the ISO C++ standards committee, Microsoft

“Free lunch is over” (Dennard scaling)

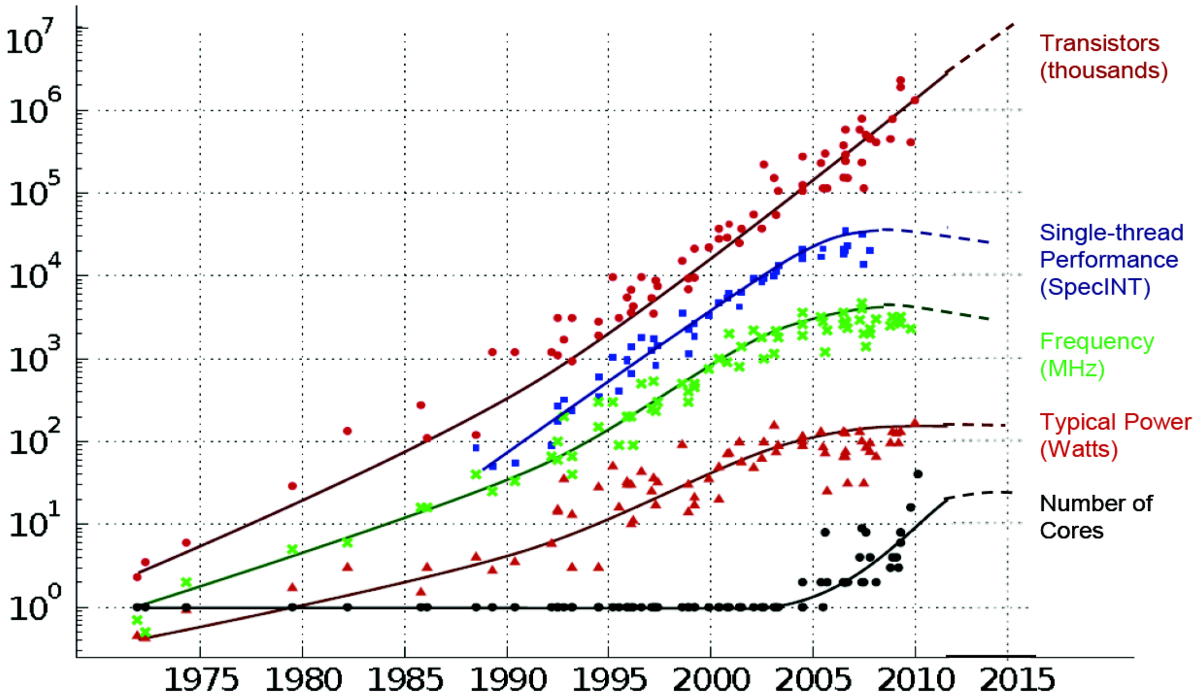
Before

1. Clock speed (dead)
2. Execution optimization (dead)
3. *Cache*

After

1. (Hyper)threading
2. Multicore
3. *Cache*

35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Parallel programming is hard

- Need to optimize for performance
- Understand management of resources
- Identify bottlenecks
- No one technology fits all needs
- Zoo of programming models, languages, run-times
- Hardware architecture is a moving target
- Parallel thinking is not intuitive
- Parallel debugging is not fun

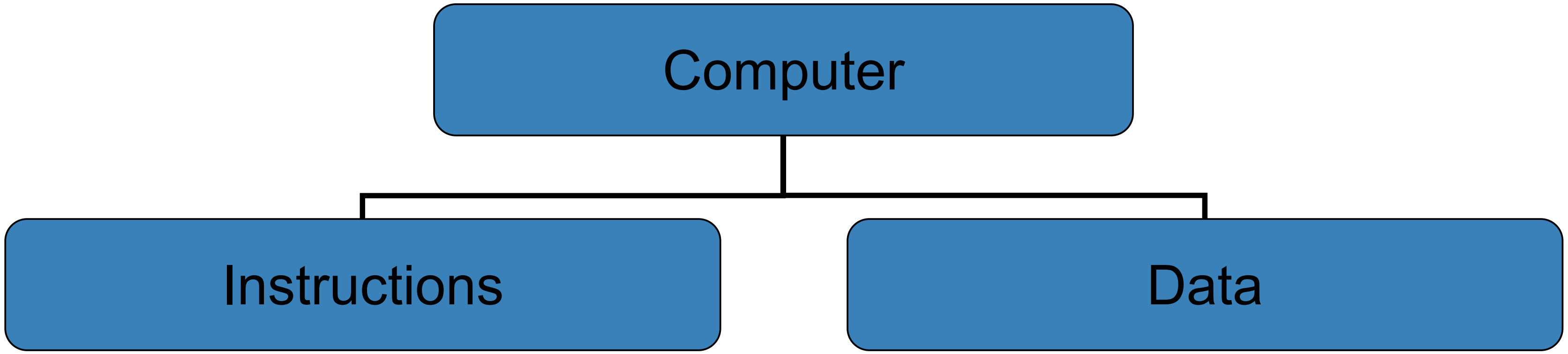
But there is no better alternative!!!

General Concepts in Parallelism and Concurrency

Concurrency vs Parallelism

Concurrency is the execution of multiple tasks at the same time, regardless of the number of processors.

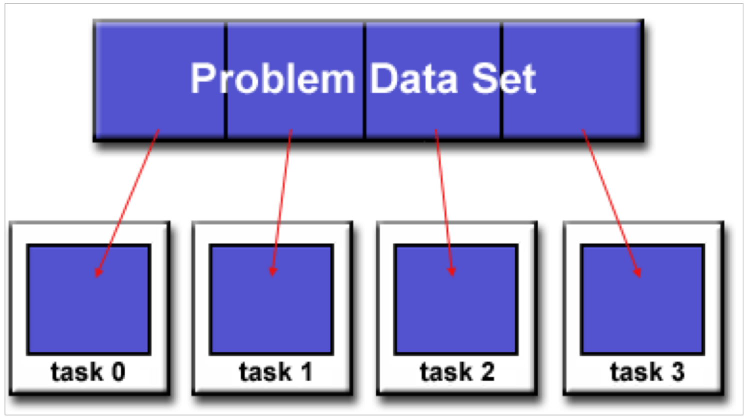
Parallelism is the execution on multiple processors of the same task.


Computer
Instructions
Data

Problem partitioning

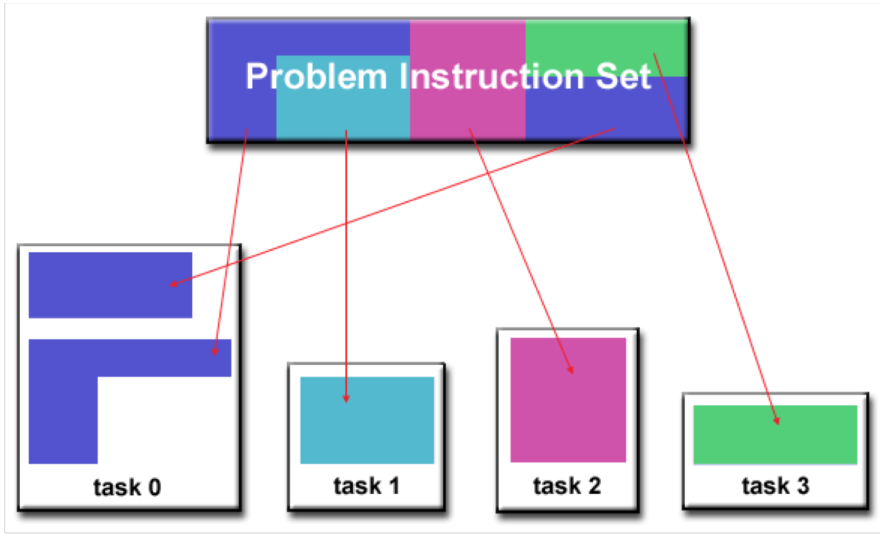
Domain decomposition

- Single Program, Multiple Data
- Input domain
- Output domain
- Both



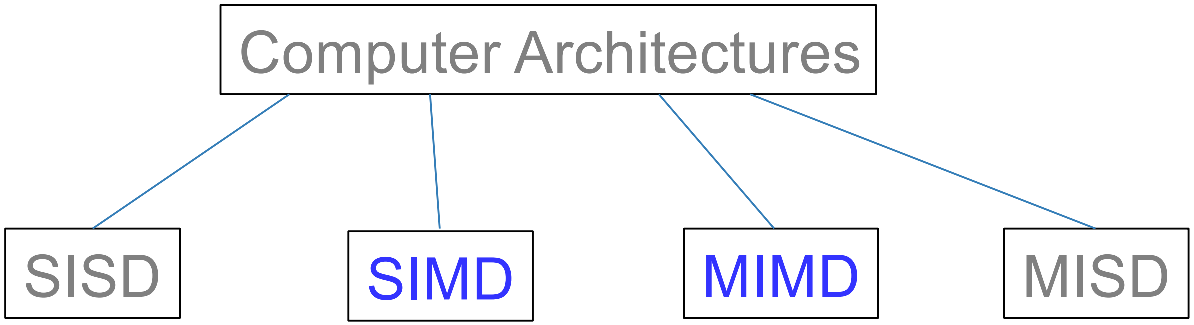
Functional decomposition

- Multiple Programs, Multiple Data
- Independent tasks
- Pipelining



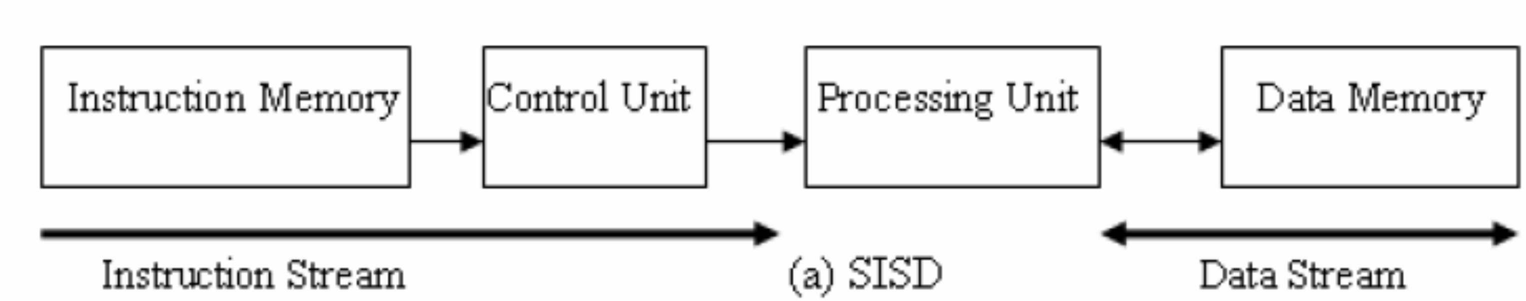
Flynn’s taxonomy

- Single Instruction Single Data (SISD)
  - Traditional sequential computing systems
- Single Instruction Multiple Data (SIMD)
- Multiple Instructions Multiple Data (MIMD)
- Multiple Instructions Single Data (MISD)



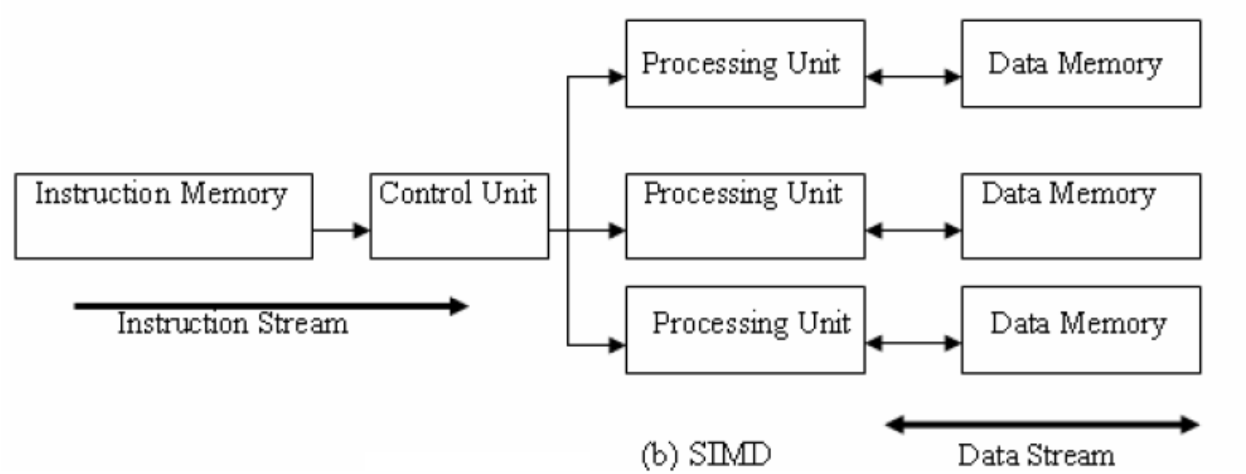
SISD

- At one time, one instruction operates on one data
- Traditional sequential architecture



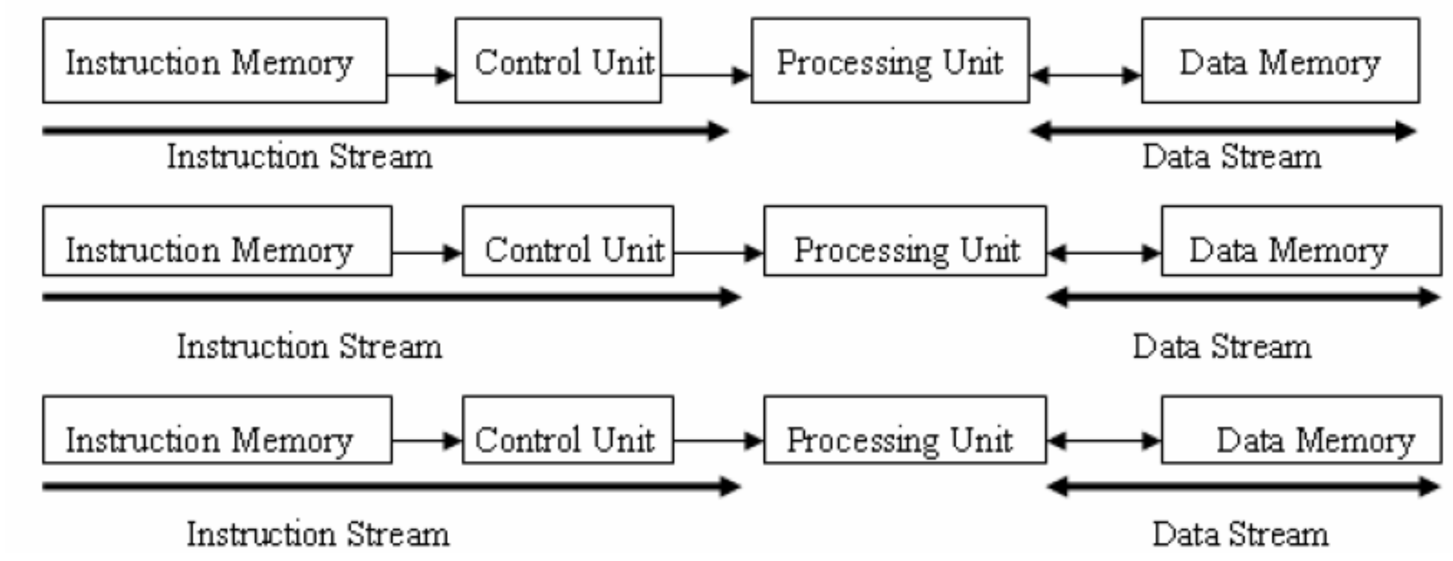
SIMD

- At one time, one instruction operates on many data
  - Data parallel architecture
  - Vector architecture has similar characteristics, but achieves parallelism via pipelining.
- Array processors



MIMD

- Multiple instruction streams operating on multiple data streams
- Classical distributed memory or symmetric multiprocessing (SMP) architectures



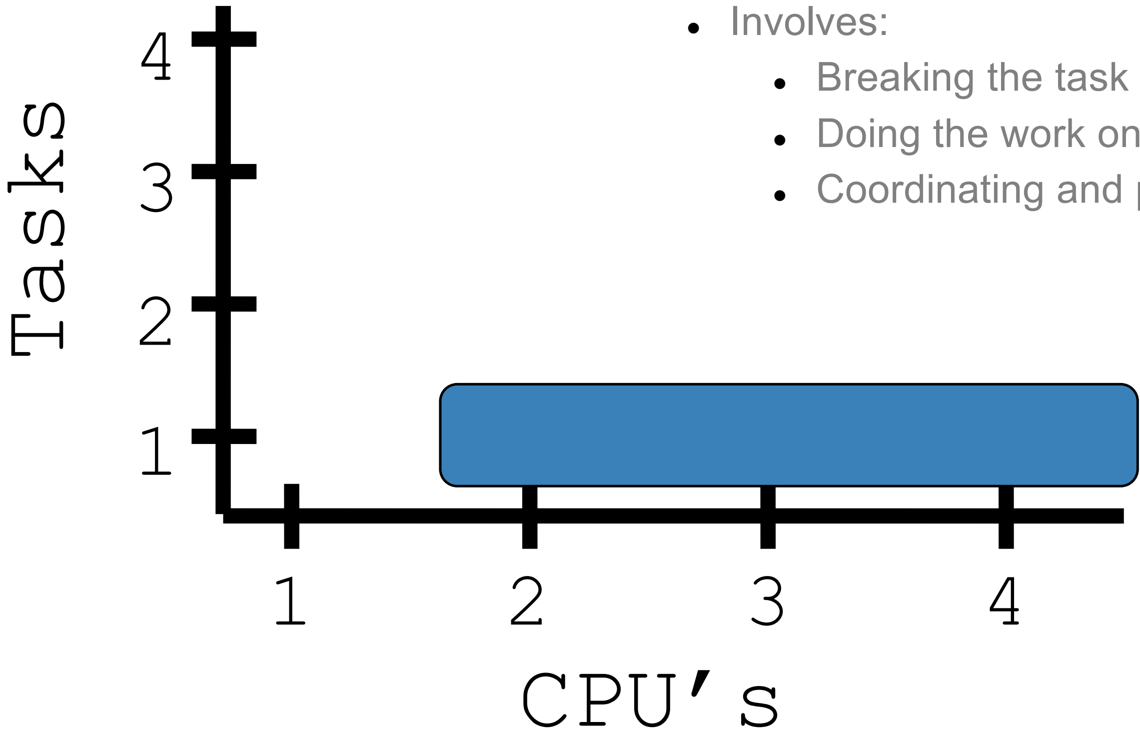
Multiprocessor Execution Model

- Each processor has its own PC and executes an independent stream of instructions (MIMD)
- Different processors can access the same memory space
- Processors can communicate via shared memory by storing/loading to/from common locations
- Two ways to use a multiprocessor:
  1. Deliver high throughput for independent jobs via job-level parallelism
  2. Improve the run time of a single program that has been specially crafted to run on a multiprocessor - a parallel-processing program

We use the term *core* for processor ("Multicore") because "Multiprocessor Microprocessor" is too redundant

Parallelism

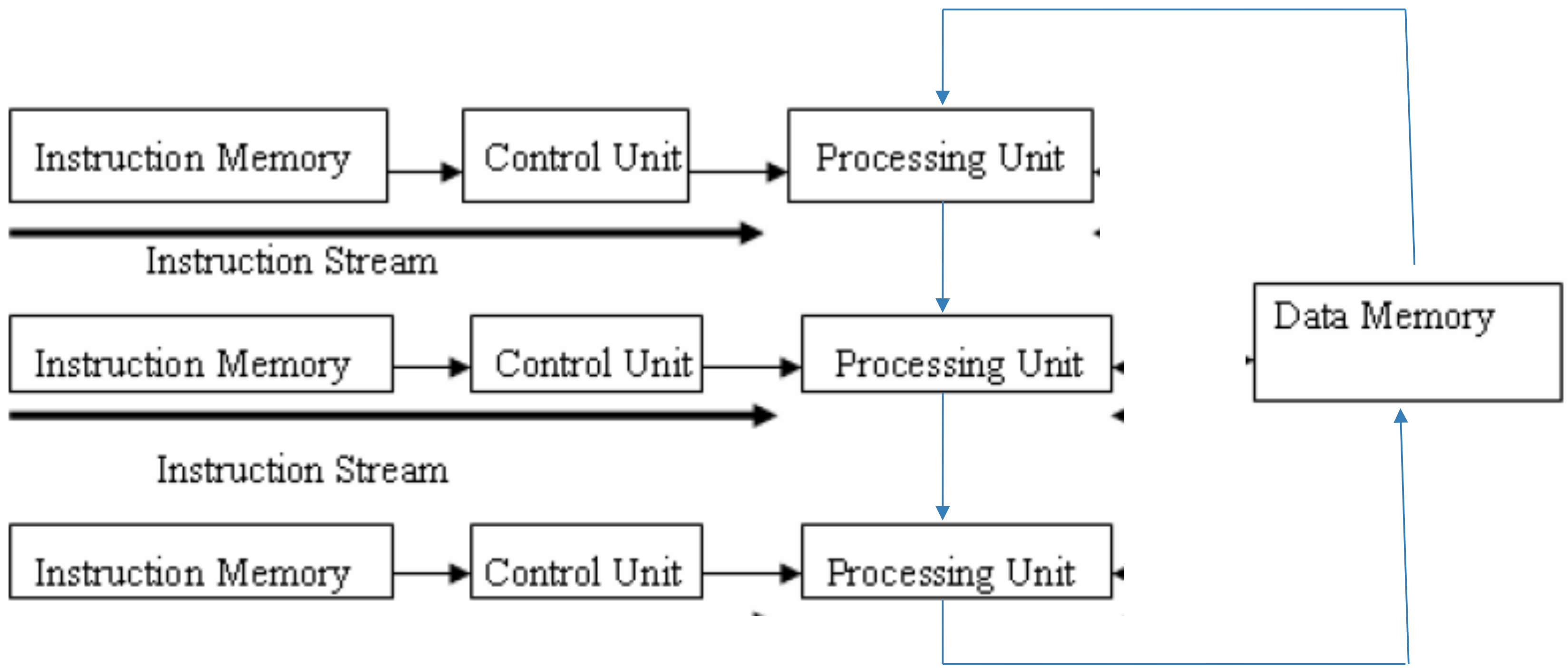
- Using multiple processors to solve a single problem.



- Involves:
  - Breaking the task into meaningful pieces
  - Doing the work on many processors
  - Coordinating and putting the pieces back together.

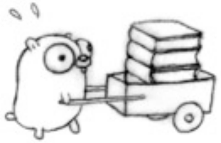
MISD

- Not commonly seen.
- Systolic array is one example of an MISD architecture.



Sequential processing

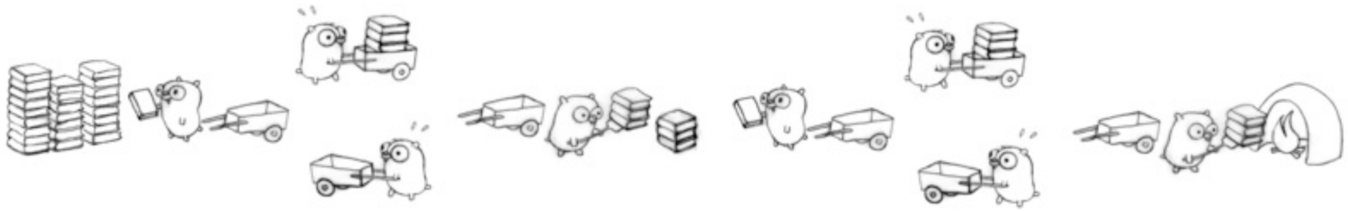
- We have one “thread” of execution
- One step follows another in a sequence
- One processor is all that is needed to run the algorithm



Rob Pike's GOphers

Concurrent Systems

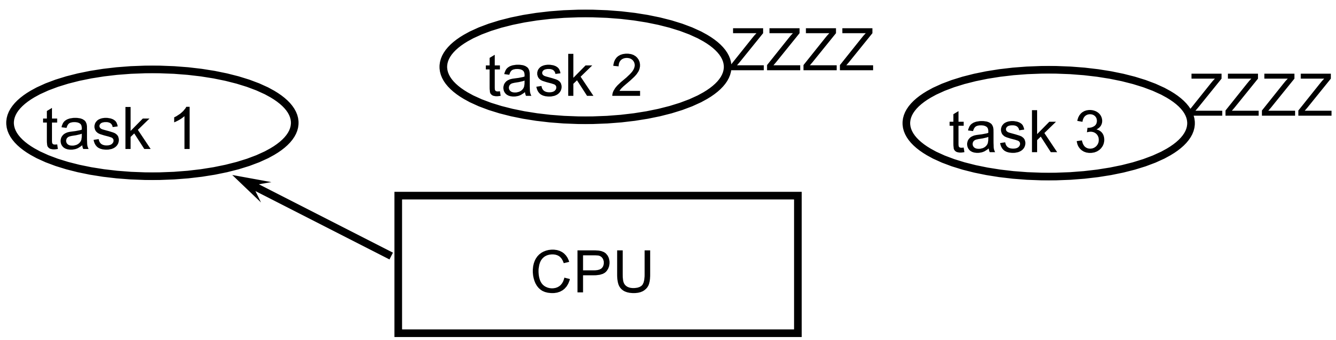
- Multiple tasks can be executed at the same time
- The tasks may be duplicates of each other, or distinct tasks
- The overall time to perform the series of tasks is reduced



Achieving concurrency

Concurrency can also be achieved on a computer with only one processor:

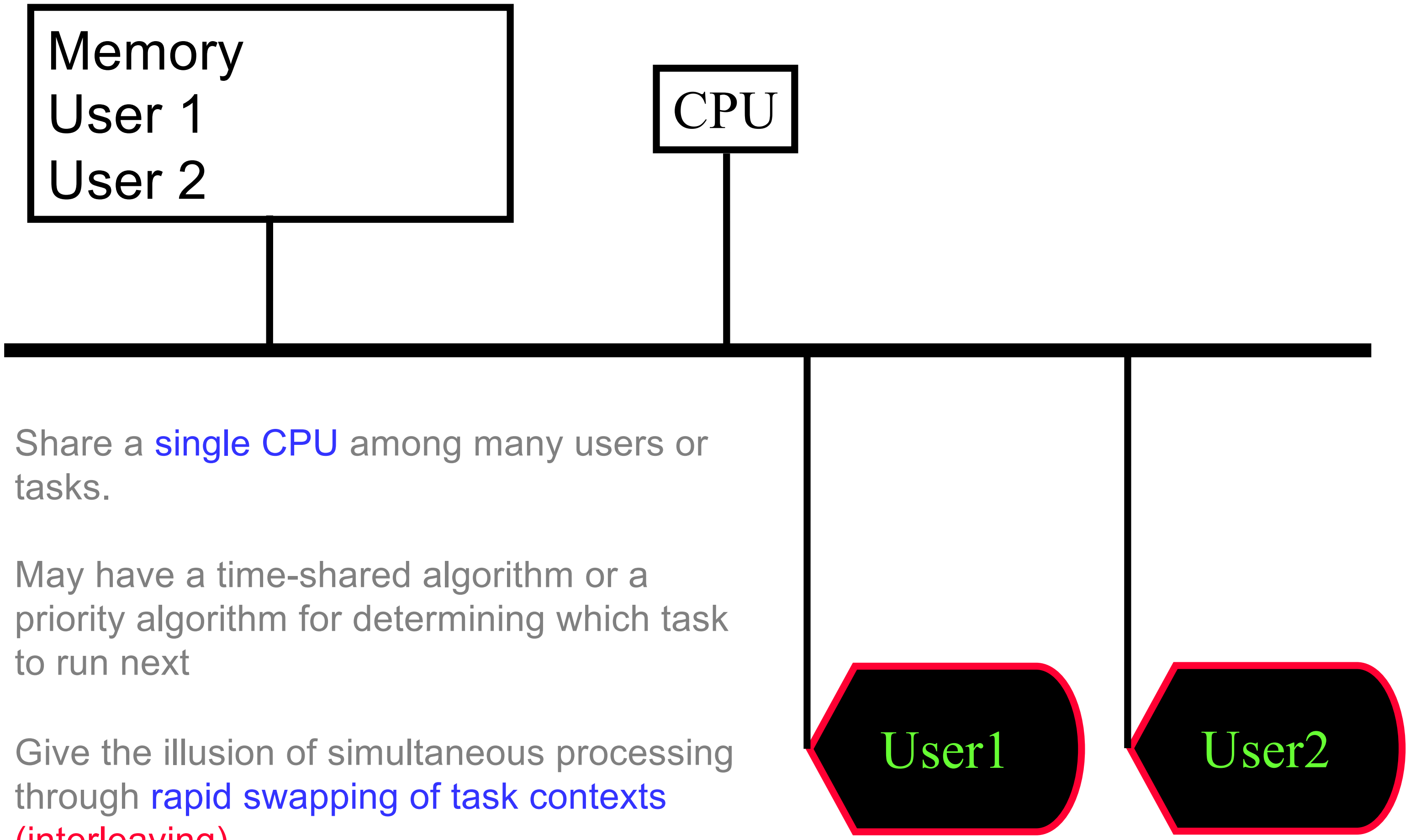
- The computer “juggles” jobs, swapping its attention to each in turn
- “Time slicing” allows many users to get CPU resources
- Tasks may be suspended while they wait for something, such as device I/O



Types of Concurrent Systems

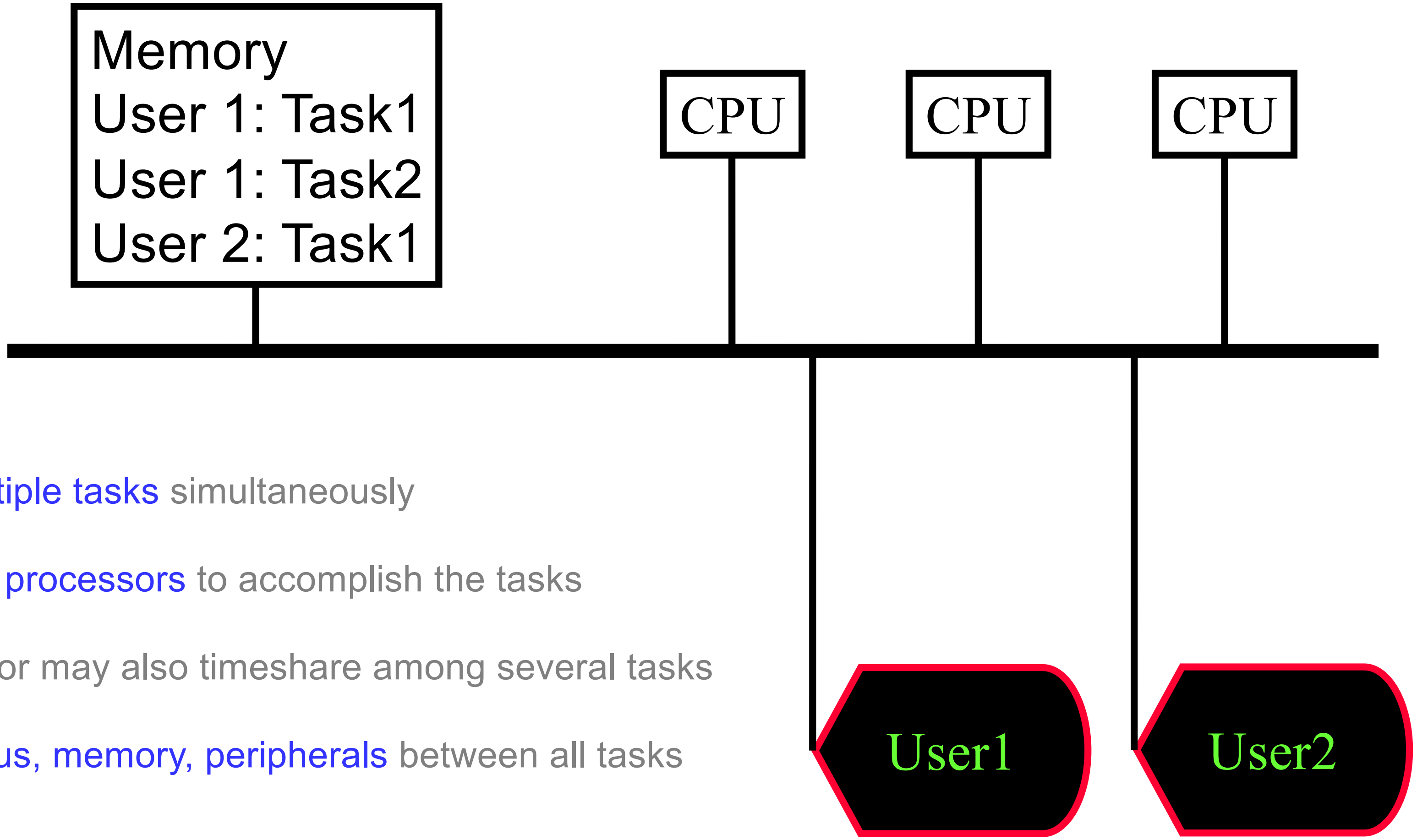
- Multiprogramming
- Multiprocessing
- Multitasking
- Distributed Systems

Multiprogramming – software concurrent execution



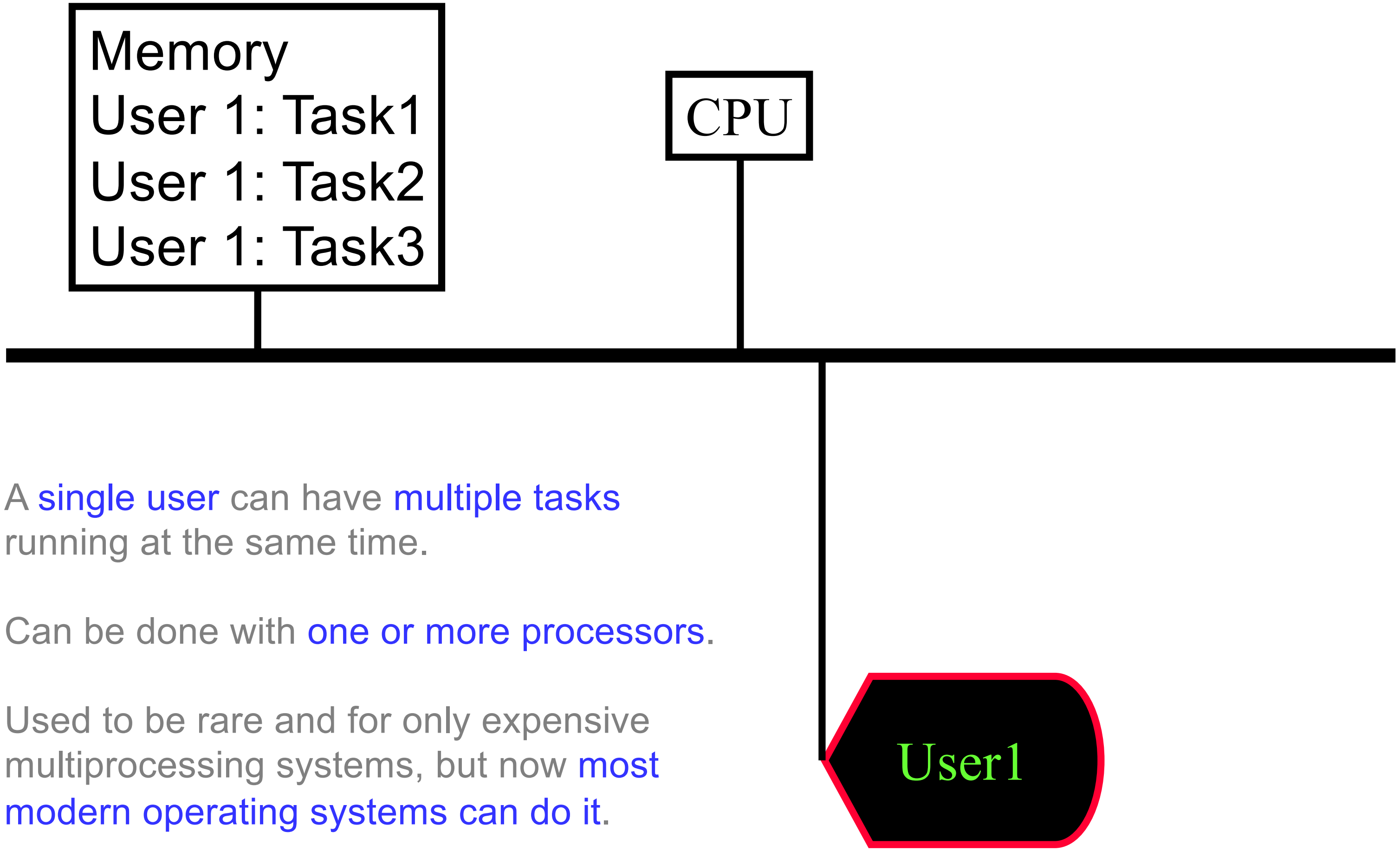
- Share a single CPU among many users or tasks.
- May have a time-shared algorithm or a priority algorithm for determining which task to run next
- Give the illusion of simultaneous processing through rapid swapping of task contexts (interleaving).

Multiprocessing – hardware parallel execution



- Executes multiple tasks simultaneously
- Uses multiple processors to accomplish the tasks
- Each processor may also timeshare among several tasks
- Has shared bus, memory, peripherals between all tasks

Multitasking – logical extension of multiprogramming



- A single user can have multiple tasks running at the same time.
- Can be done with one or more processors.
- Used to be rare and for only expensive multiprocessing systems, but now most modern operating systems can do it.

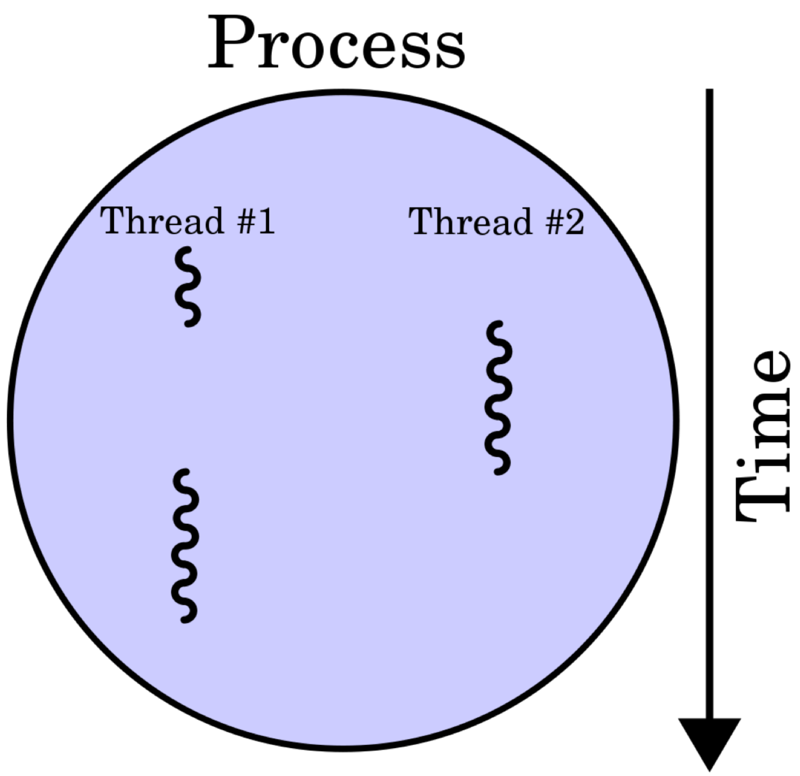
Threads

Thread: a sequential flow of instructions that performs some task

- Each thread has a PC + processor registers and accesses the shared memory
- Each processor provides one or more *hardware* threads, sometimes called *harts,* that actively execute instructions
- The operating system (OS) maps *software* threads to hardware threads

Multithreading

- Execution model that allows a single process to have multiple threads running concurrently within the context of that process



https://en.wikipedia.org/wiki/Thread_(computing)

Advantages of concurrency

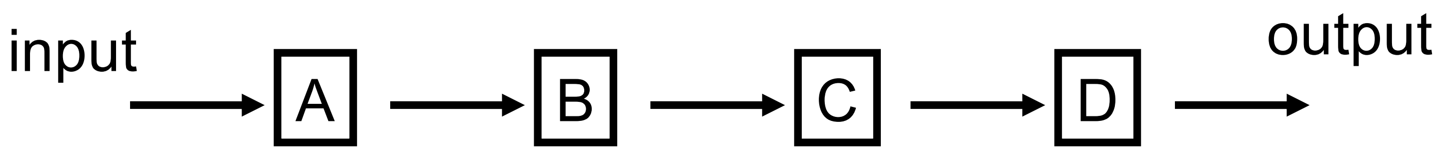
- Concurrent processes can reduce duplication.
- The overall runtime of the algorithm can be significantly reduced.
- More real-world problems can be solved than with sequential algorithms alone.

Disadvantages of concurrency

- Runtime is not always reduced, so careful planning is required
- Concurrent algorithms can be more complex than sequential algorithms
- Shared data can be corrupted
- Communication between tasks is needed

Pipeline processing

- Repeating a sequence of operations or pieces of a task.
- Allocating each piece to a separate processor and chaining them together produces a pipeline, completing tasks faster.

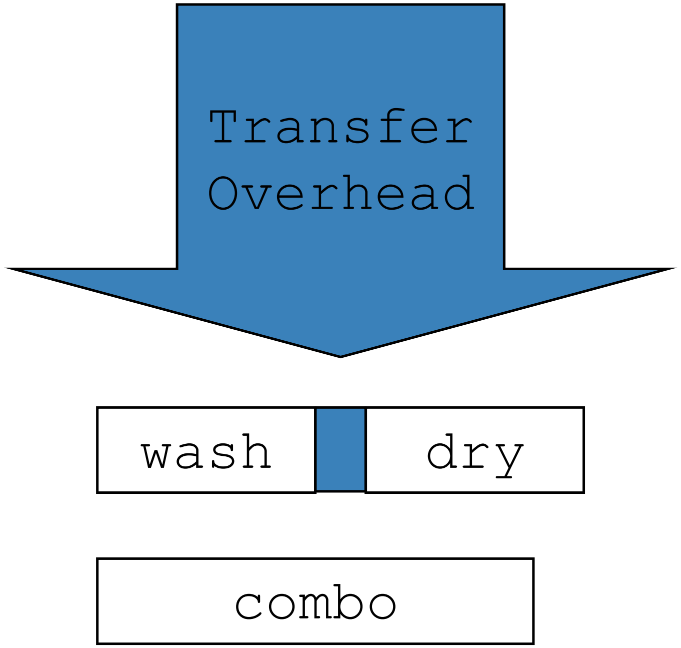


Example

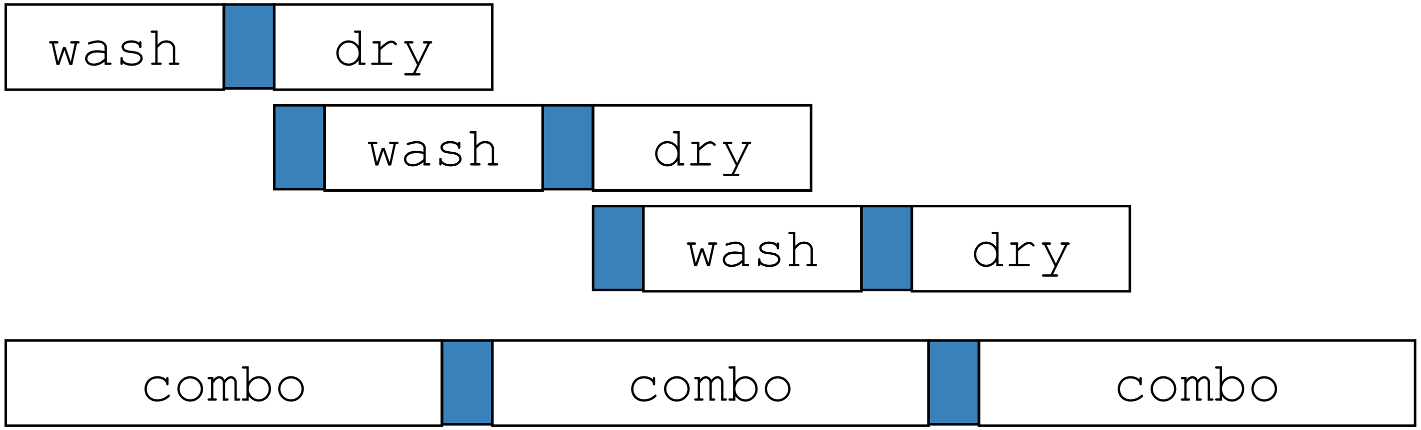
Suppose you have a choice between a washer and a dryer each having a 30 minutes cycle or
A washer/dryer with a one hour cycle

The correct answer depends on how much work you have to do.

One Load

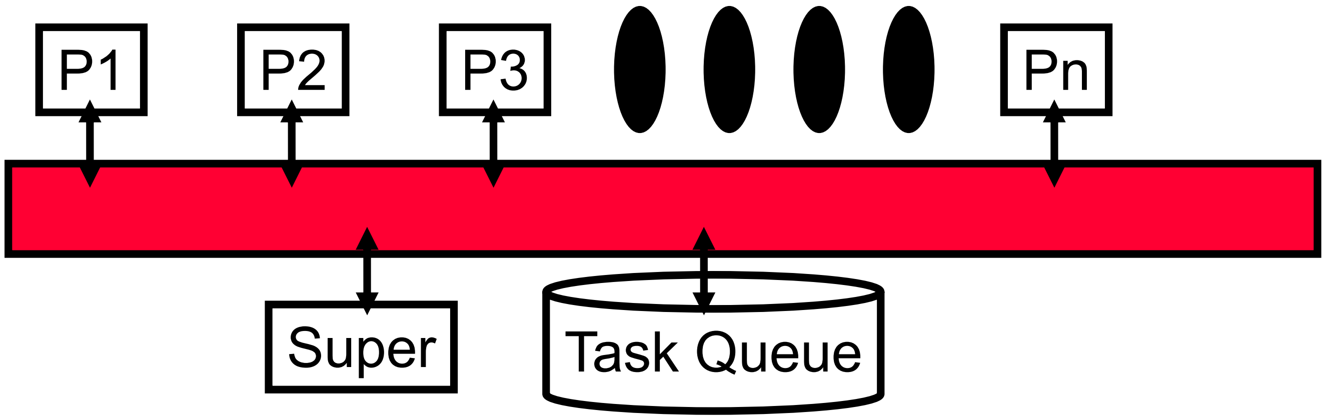


Three Loads

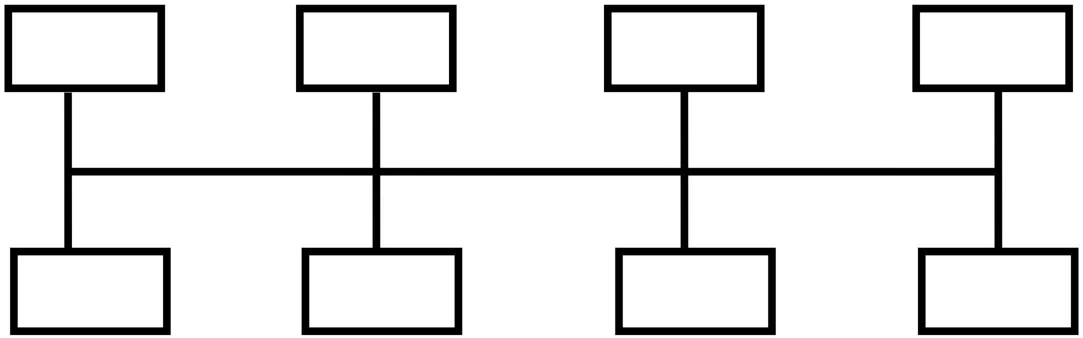


Task Queues

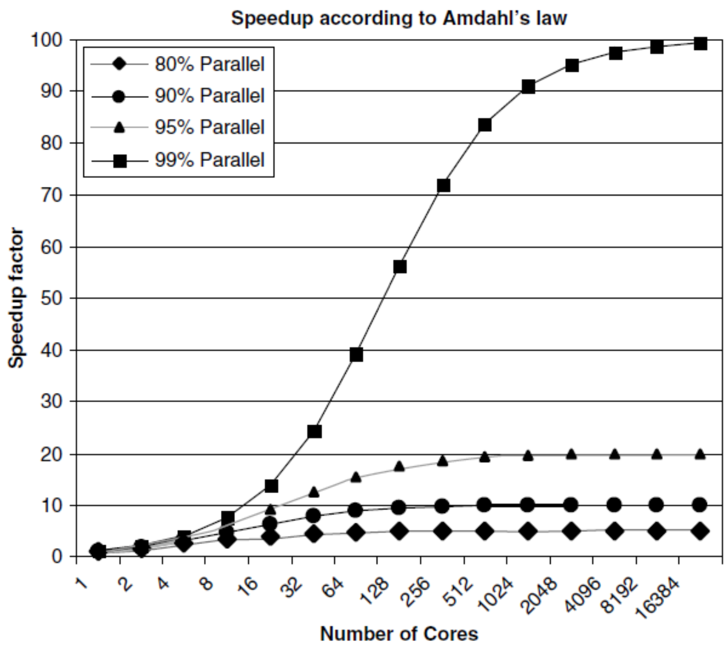
- A supervisor processor maintains a queue of tasks to be performed in shared memory.
- Each processor queries the queue, dequeues the next task and performs it.
- Task execution may involve adding more tasks to the task queue.



How much gain can we get from parallelizing an algorithm?



Amdahl's law



P is the proportion that can be made parallel
1-P is the proportion that remains serial

Then the maximum speedup that can be obtained is:

$$\frac{1}{1 - P + \frac{P}{N}}$$

If N tends to infinity then the maximum speedup tends to 1/(1-P).

Product Complexity

- Got done in O(N) time, better than $O(N^2)$
- Each time “chunk” does O(N) work
- There are N time chunks.
- Thus, the amount of work is still $O(N^2)$

Product complexity is the amount of work per “time chunk” multiplied by the number of “time chunks” – the total work done.

Ceiling of Improvement

Parallelization can reduce time, but it cannot reduce work. The product complexity cannot change or improve.
How much improvement can parallelization provide?

Given an O(NLogN) algorithm and Log N processors, the algorithm will take at least O(?) time.

O(N) time.

Given an O(N^3) algorithm and N processors, the algorithm will take at least O(?) time.

O(N^2) time.

Number of Processors

- Processors are limited by hardware.
- Typically, the number of processors is a power of 2
- Usually: The number of processors is a constant factor, 2^K
- Conceivably: Networked computers joined as needed.

Adding Processors

- A program on one processor
  - Runs in X time
- Adding another processor
  - Runs in no more than X/2 time
  - Realistically, it will run in $X/2 + \varepsilon$ time because of overhead
- At some point, adding processors will not help and could degrade performance.

Overhead of Parallelization

- Parallelization is not free.
- Processors must be controlled and coordinated.
- We need a way to govern which processor does what work; this involves extra work.
- Often the program must be written in a special programming language for parallel systems.
- Often, a parallelized program for one machine (with, say, 2^K processors) is not optimal on other machines (with, say, 2^L processors).

Python – The Global Interpreter Lock

- The Global Interpreter Lock refers to the fact that the Python interpreter is not thread safe.
  - Thread safe = shared data is manipulated so that threads don't have unintended interactions

- There is a global lock that the current thread holds to safely access Python objects.

- Because only one thread can acquire Python Objects/C API, the interpreter regularly releases and reacquires the lock every 100 bytecode of instructions. The frequency at which the interpreter checks for thread switching is controlled by the sys.setcheckinterval() function.

- In addition, the lock is released and reacquired around potentially blocking I/O operations.

- It is important to note that, because of the GIL, the CPU-bound applications won't be helped by threads. In Python, it is recommended to either use processes, or create a mixture of processes and threads.

http://www.laurentluce.com/posts/python-threads-synchronization-locks-rlocks-semaphores-conditions-events-and-queues/

Concurrency in Python

- With Python, there is no shortage of options for concurrency, the standard library includes support for threading, processes, and asynchronous I/O.
-
- In many cases Python has removed much of the difficulty in using these various methods of concurrency by creating high-level modules such as asynchronous, threading, and subprocess.
-
- Outside of the standard library, there are 3rd party solutions such as twisted, stackless, and the processing module, to name a few.

Process and threads in Python

- It is important to first define the differences between processes and threads.
-
- Threads are different than processes in that they share state, memory, and resources.
- This simple difference is both a strength and a weakness for threads.
- On one hand, threads are lightweight and easy to communicate with, but on the other hand, they bring up a whole host of problems including deadlocks, race conditions, and sheer complexity.
- Fortunately, due to both the GIL and the queuing module, threading in Python is much less complex to implement than in other languages.

THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu

RESERVE SLIDES

SURFsara

History:
1971: Founded by the VU, UvA, and CWI
2013: SARA (Stichting Academisch Rekencentrum A'dam) becomes part of SURF

Cartesius (Bull supercomputer):
40.960 Ivy Bridge / Haswell cores: 1327 TFLOPS
56GBit/s Infiniband
64 nodes with 2 GPUs each: 210 TFLOPS
NVIDIA Tesla K40m GPU
Broadwell & KNL extension (Nov 2016)
177 BDW and 18 KNL nodes: 284TFLOPS
7.7 PB Lustre parallel file-system

Top500 position
#45 2014/11
#97 2016/11