Operating Systems

Lecture 3a

Dr Ronald Grau School of Engineering and Informatics Spring term 2018

Previously

- Processes
- Threads
  - Distinction to processes (lightweight sub-processes)
  - Single- and multi-threaded processes
  - Implementation
    - User-level threads
    - Kernel-level threads
    - Hybrid threads

Today

Programming with Threads

- The relationship between processes and threads
- Threads & Multiprogramming
- Parallel vs concurrent execution
- Data vs. Task vs. Pipeline parallelism
- Single- and multi-core execution
- Hyperthreading
- Java thread library
- Thread Safety

Recap: Processes and threads

Revisiting the key abstractions that a **process** provides

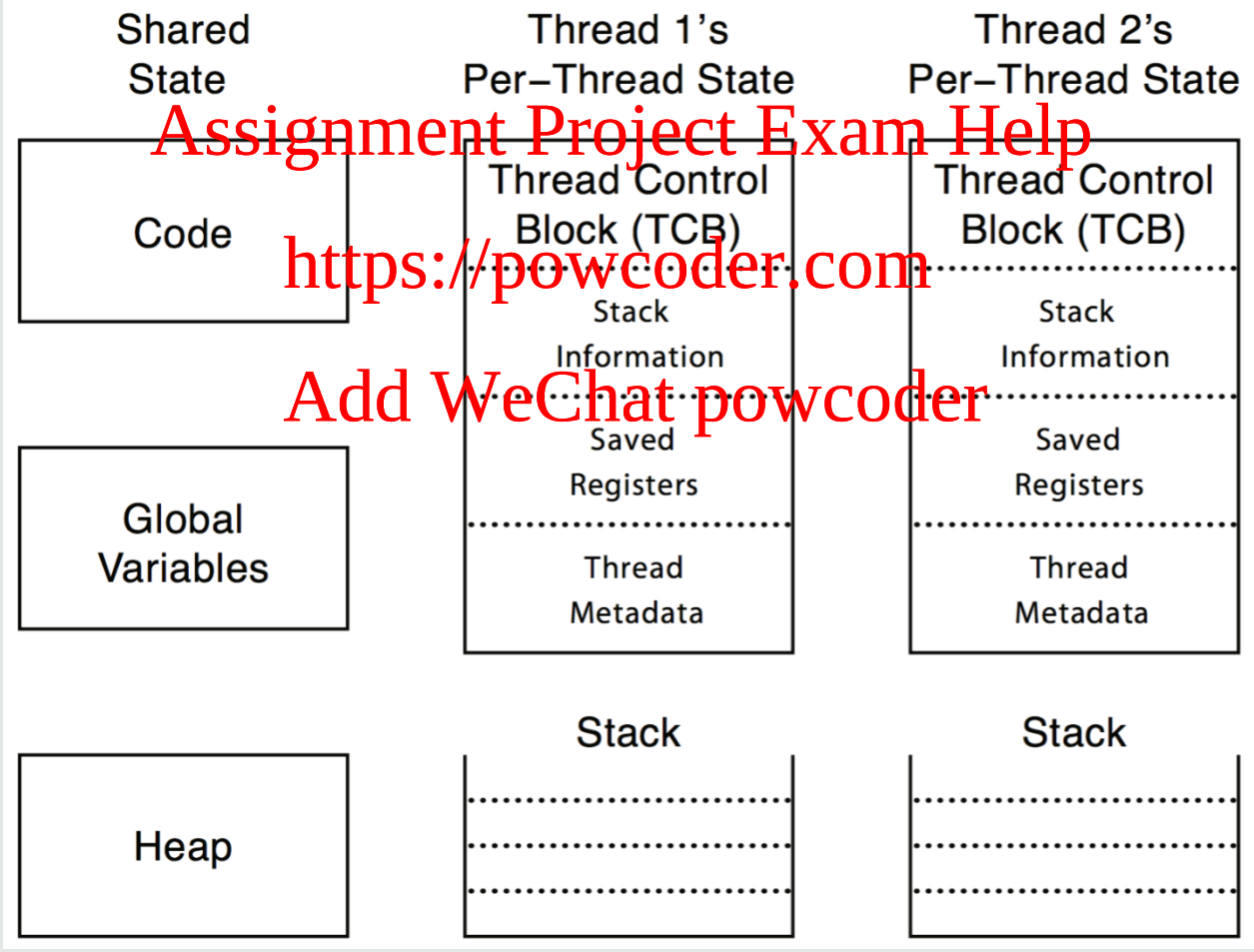
- A **private address space** that provides the illusion that our program has exclusive use of the memory system. This provides a way for the operating system to **group related resources** together. Thus, a process provides a ***unit of resource ownership*** abstraction to the running program
- An **independent logical thread of control** that provides the illusion that our program has exclusive use of the processor. Thus, process provides a ***unit of scheduling (or dispatch)*** abstraction that runs on the processor.

The concept of a **thread** comes from the realisation that the two characteristics above are independent and can be treated independently by the OS.

A thread is a unit of scheduling / dispatch that comes without the burden of extensive resource ownership. Threads are therefore also called *light-weight processes*.

Recap: Processes and threads

Threads



Terminology

- Process
  - Unit of resource ownership and protection
  - Consists of one or more threads
- Thread
  - Unit for scheduling
  - Sequential thread of execution
  - Owned by a process

Terminology

- Job
  - Generally, “work to be done" on the application level
  - Usually, groups of processes to be managed as a unit
  - Term originating from batch processing (e.g. IBM OS/360)
- Task
  - Meaning depends on context
  - Generally, a “piece of work" that is part of a job
  - In the sense of multitasking: processes or threads executing concurrently / in parallel

Parallelism vs. Concurrency

Parallelism

- N tasks execute at the same time
- Requires >= N computing resources

Building a house…

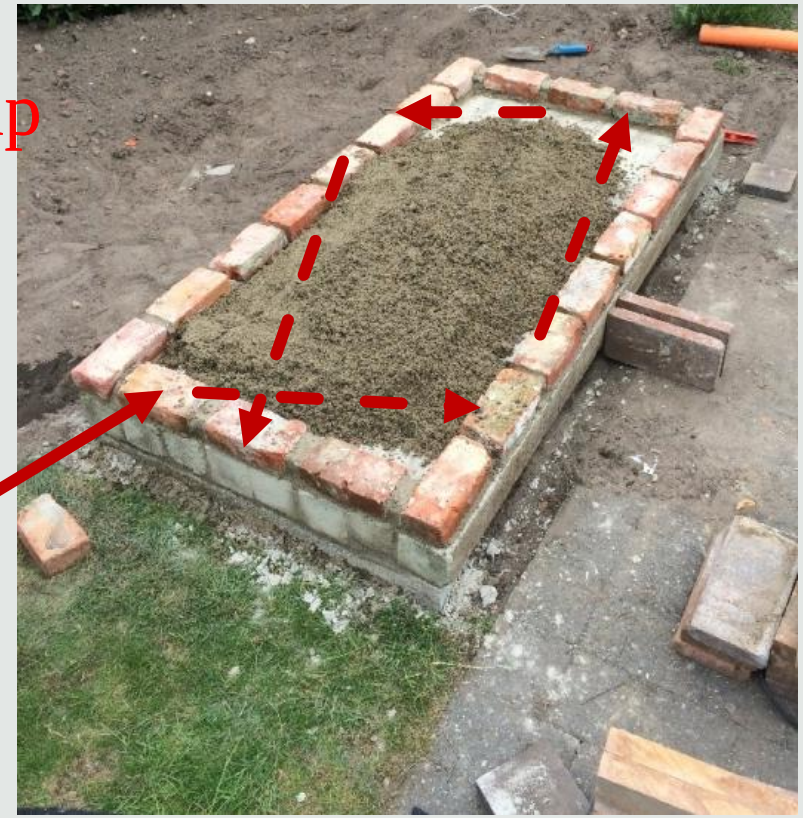
Parallelism vs. Concurrency

Parallelism

- N tasks execute at the same time
- Requires >= N computing resources

Concurrency

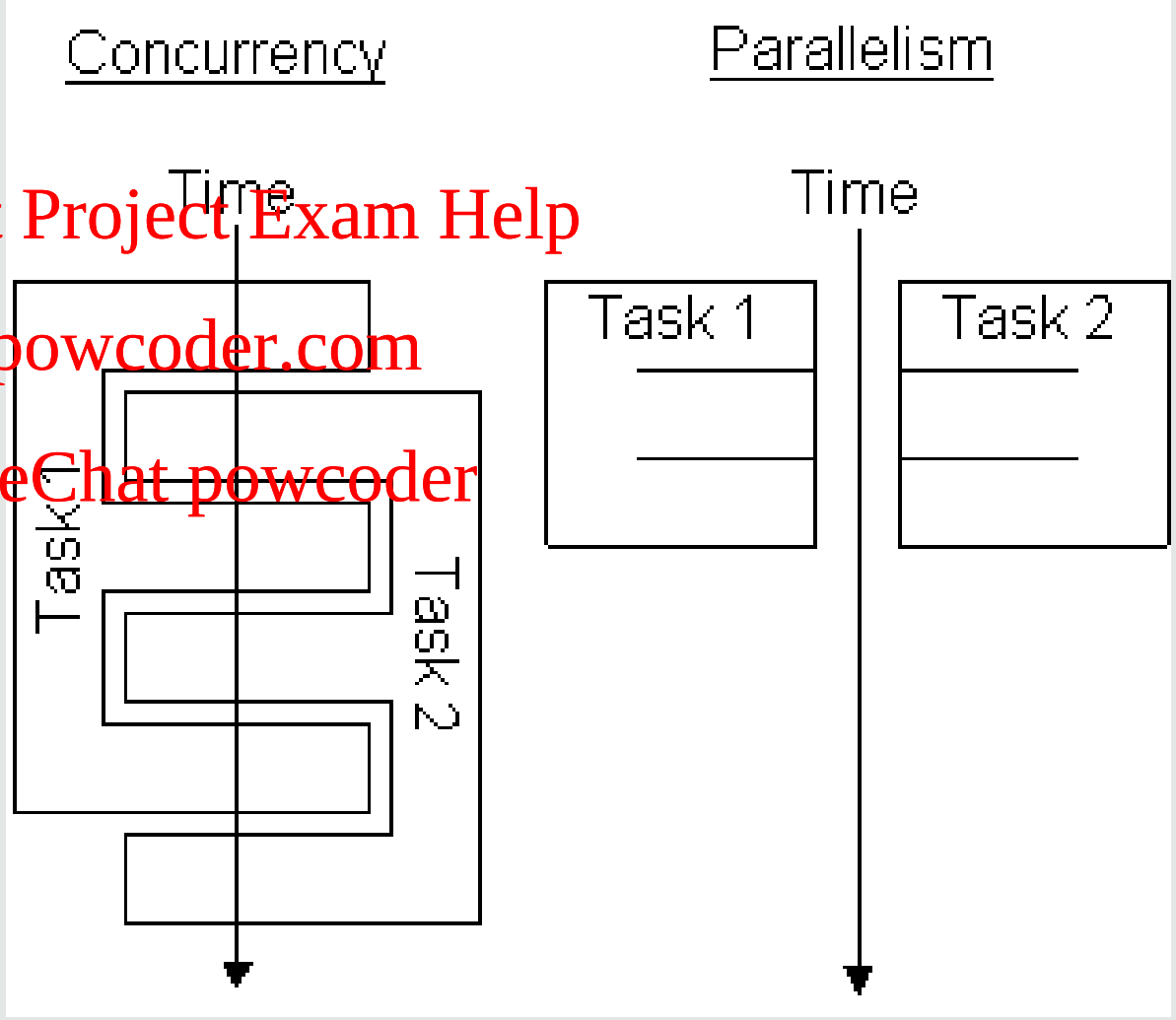
- N tasks progress at the same time
- Competition for < N resources



Building a house…

Parallelism vs. Concurrency

- Multiprogramming corresponds to concurrency
- Multiprocessing corresponds to parallelism



Parallelising Applications

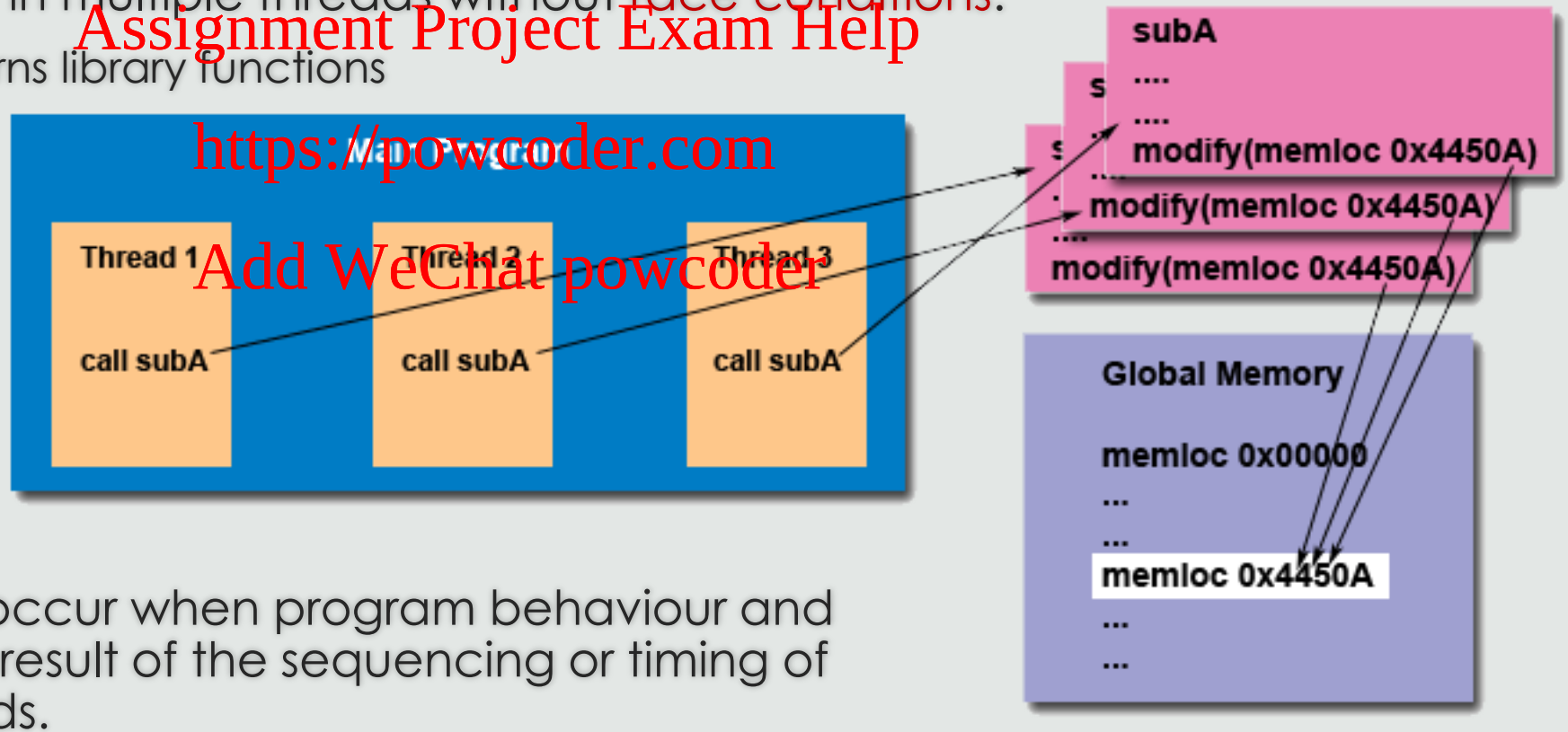
Challenges

- Identifying tasks:
  What can be run concurrently?
- Balance:
  Do tasks perform equal work of equal value?
- Data splitting:
  How best divide the data for parallel processing?
- Data dependency:
  Need consider dependencies?
- Testing and debugging:
  Many different execution paths are possible

Parallelising Applications

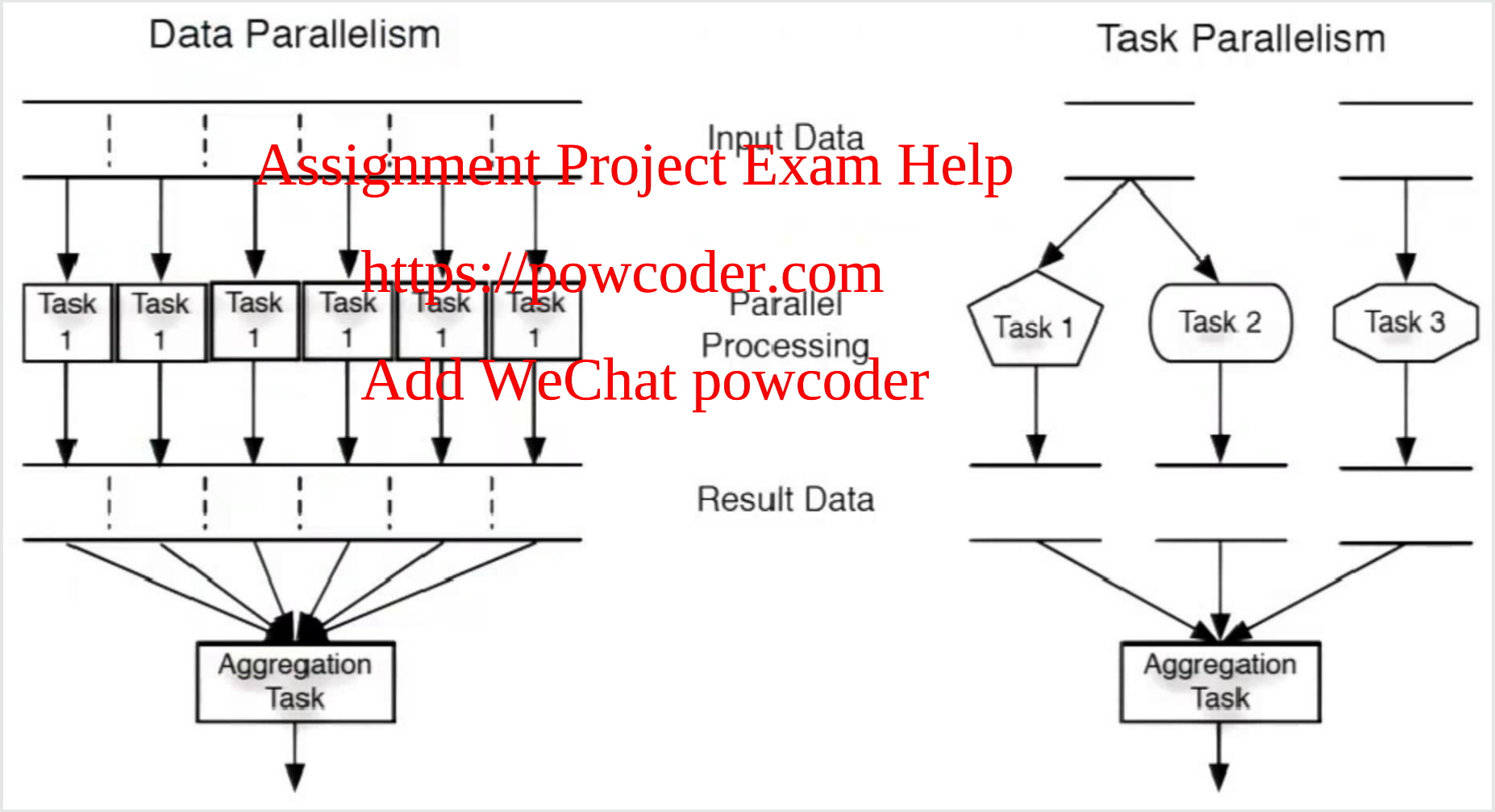
Thread safety

- Ability to execute in multiple threads without race conditions.
  - Typically concerns library functions

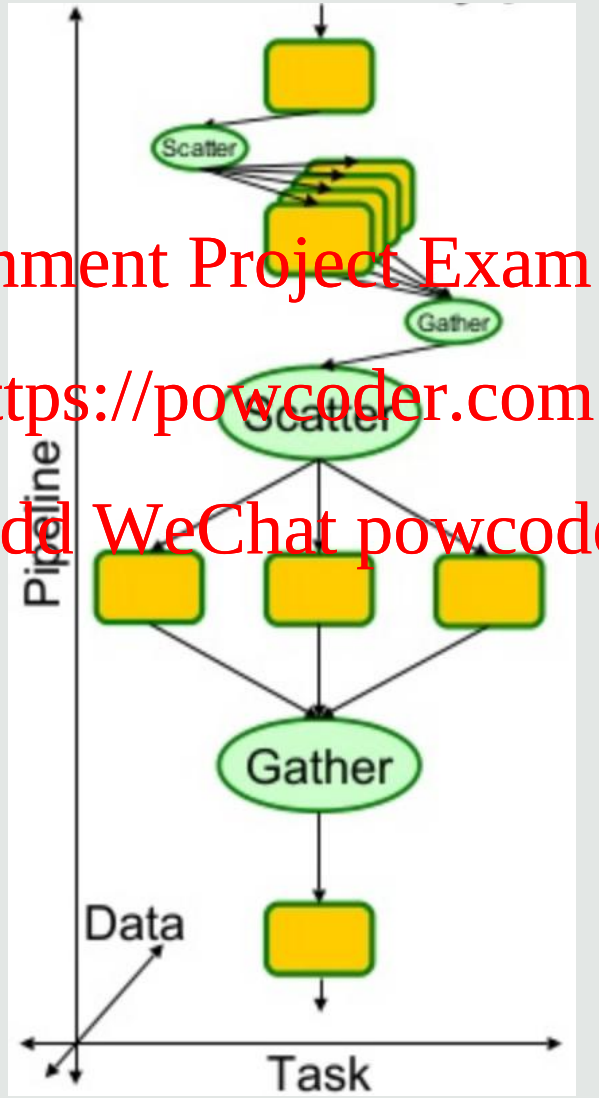


- Race conditions occur when program behaviour and outputs vary as a result of the sequencing or timing of concurrent threads.

Data vs Task Parallelism



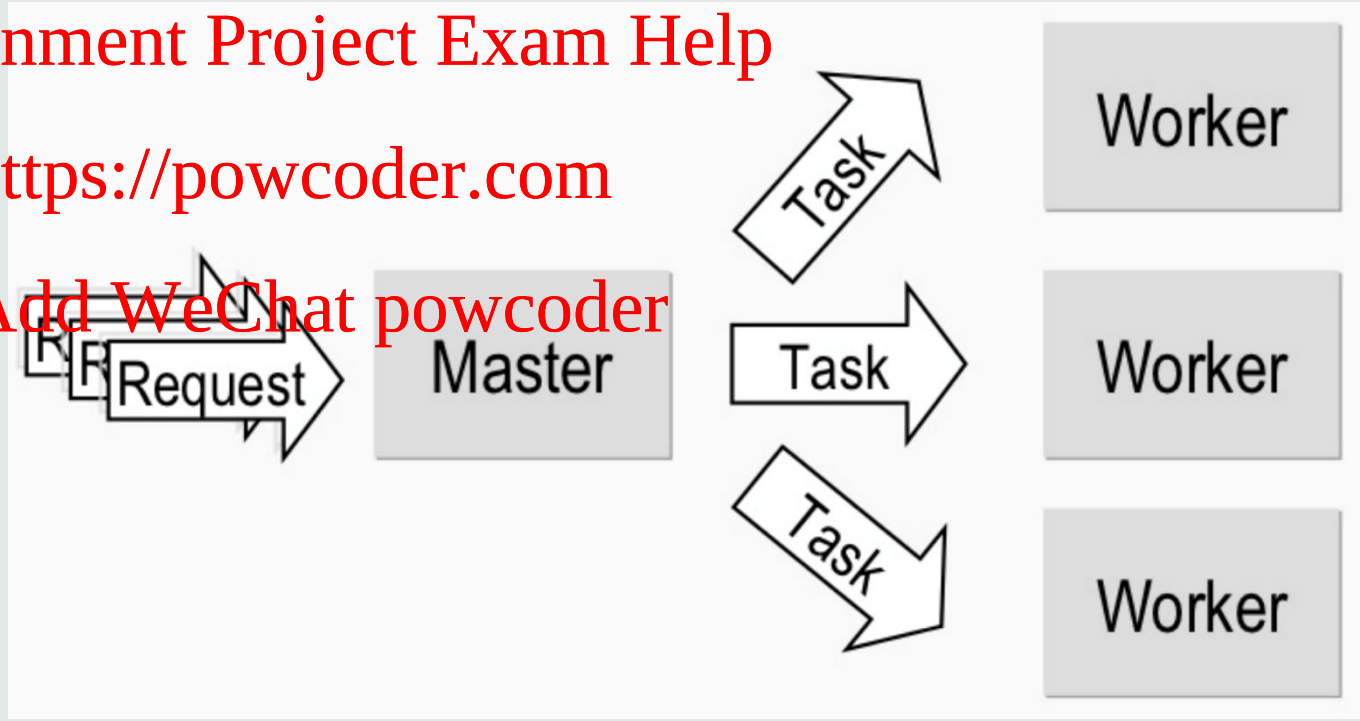
… vs Pipeline Parallelism



Example

Worker model

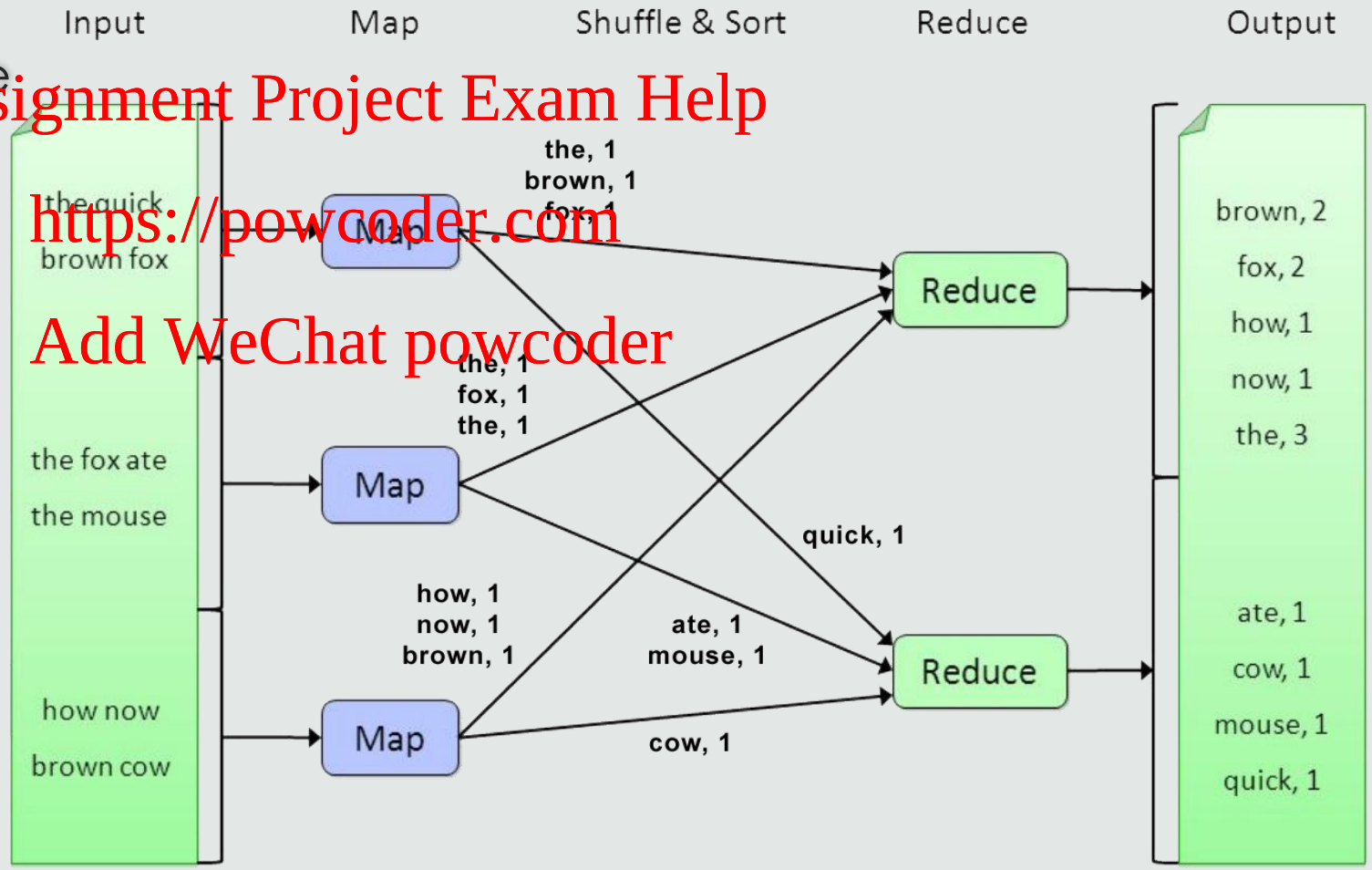
- Similar to the previous webserver example



Example

Map-Reduce Model

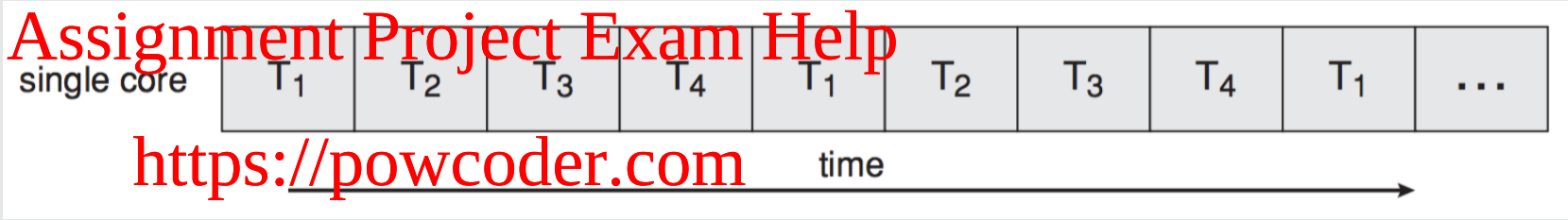
- Different functions for the distribution (mapping), distributed processing and aggregation of results (shuffle & sort), and re-combination (reducing) of data
- E.g., counting word occurrence in many different data sources



Single-core CPUs

Single-core

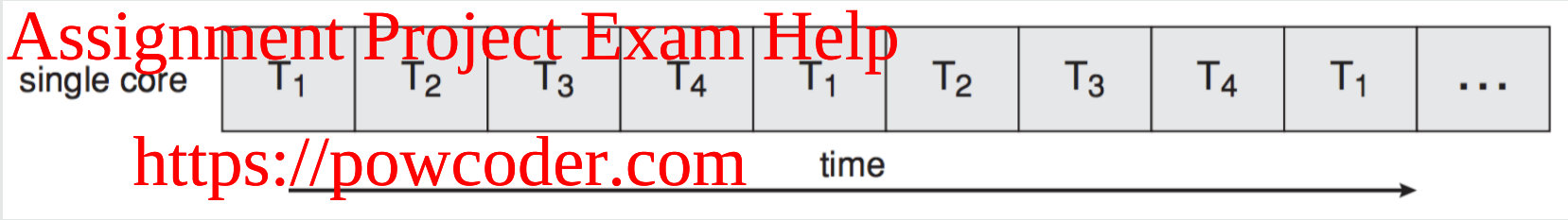
- Concurrent thread execution



Single-core vs Multi-core CPUs

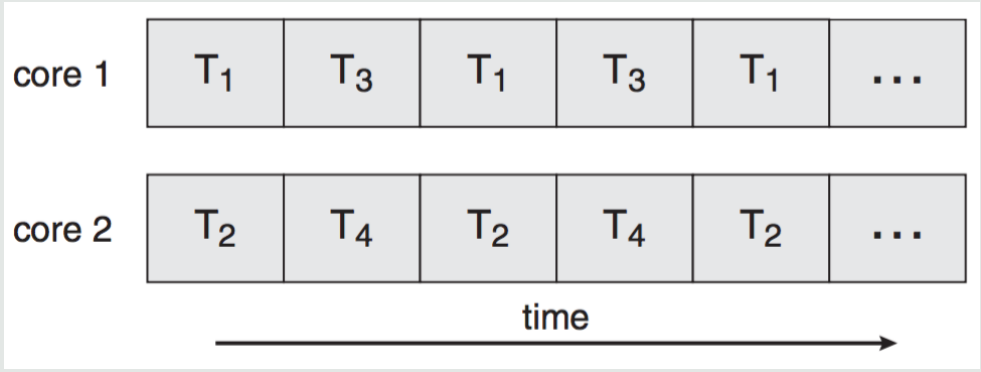
Single-core

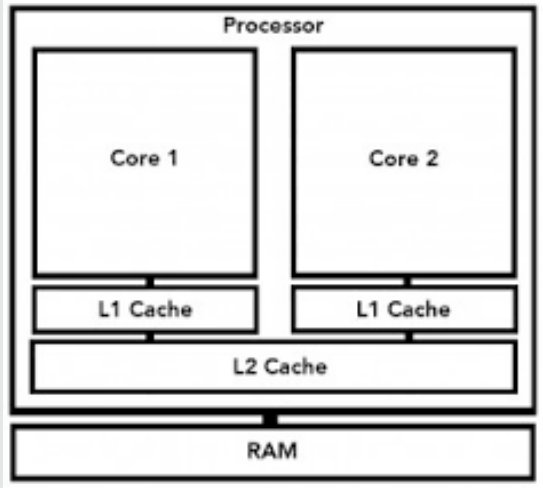
- Concurrent thread execution



Multi-core

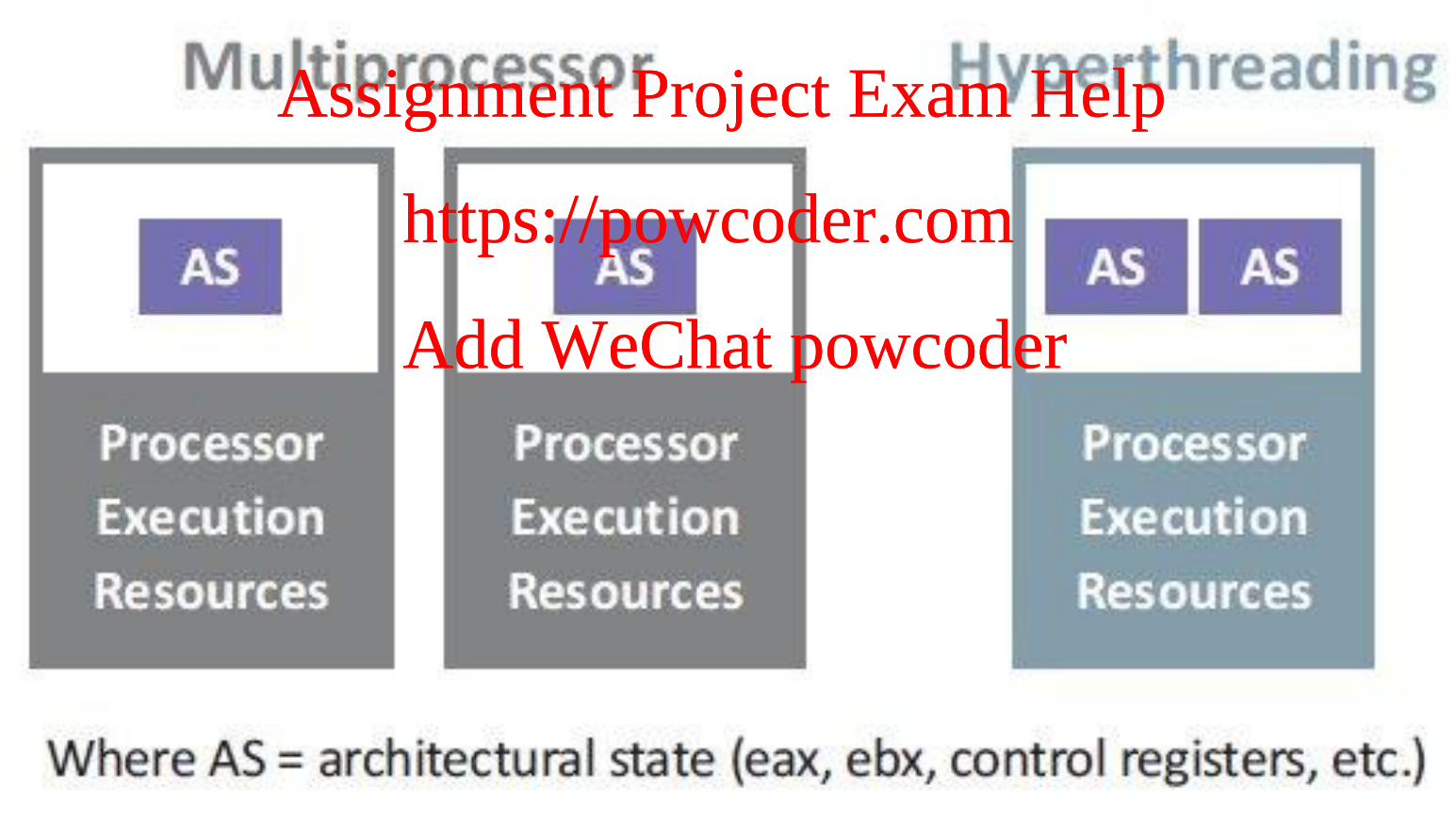
- Parallel and concurrent thread execution





Hyperthreading

Multi-threaded virtual cores



Limits of parallelisation

How much speed can we gain from parallelisation?

- Consider we have a program running on a single-core machine.
- We are now running the same program on 2 / 4 / 8 cores.
- Are we going to get a 2x / 4x / 8x speed increase as a result?

- What are the factors that impact on parallelisation performance?

Limits of parallelisation

Amdahl's Law

- Upper Bound on performance gains from parallelisation:

$$speed\_up = \frac{1}{(1 - P) + \frac{P}{N}}$$

- P: the portion of the program that can be executed in parallel
- N: number of processing cores

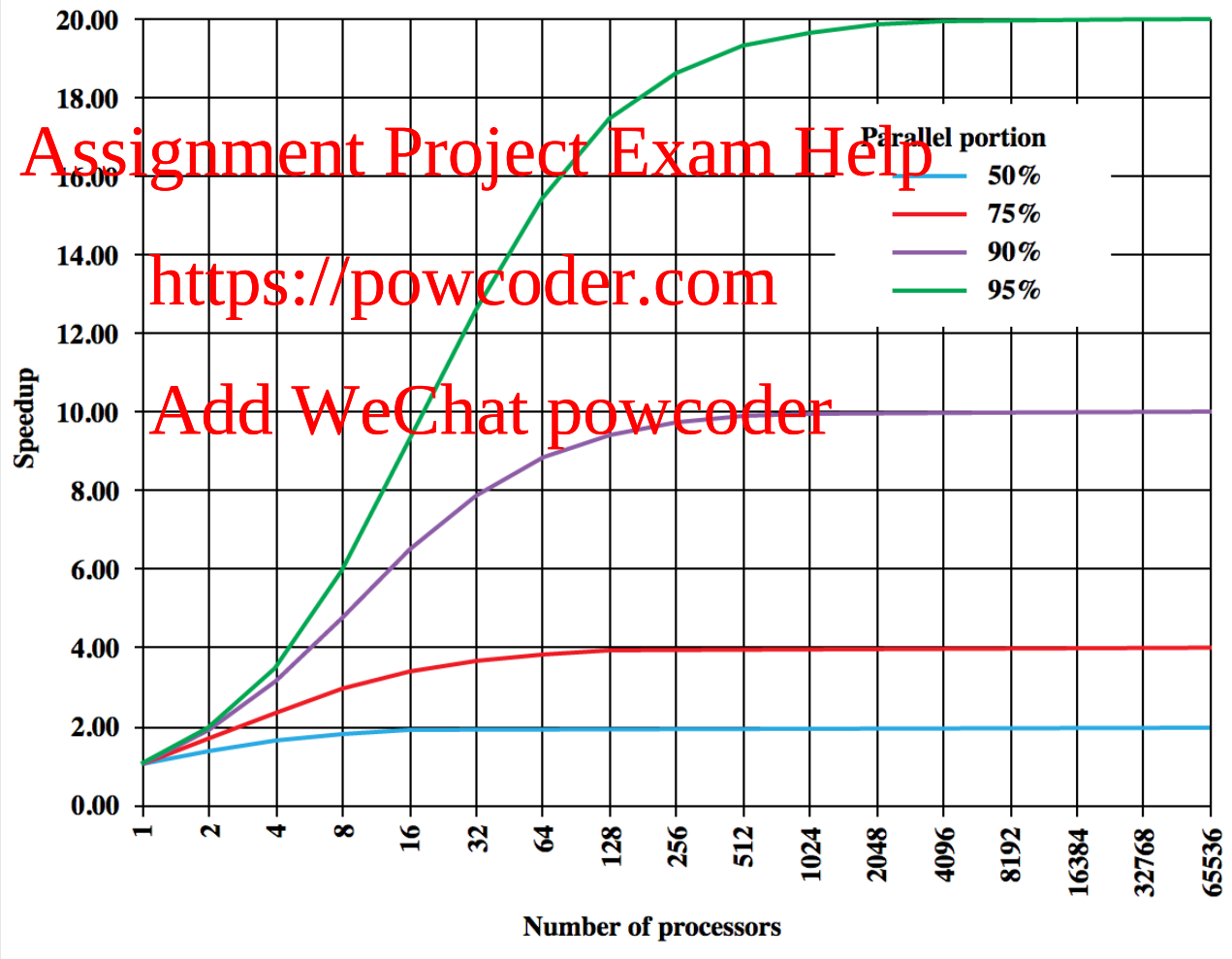
- Example: P = 0.75, N = 2.

$$speed\_up = \frac{1}{(1 - 0.75) + \frac{0.75}{2}} = 1.6$$

Limits of parallelisation

Amdahl's Law

- Trends



Java threads

How do threads work in Java?

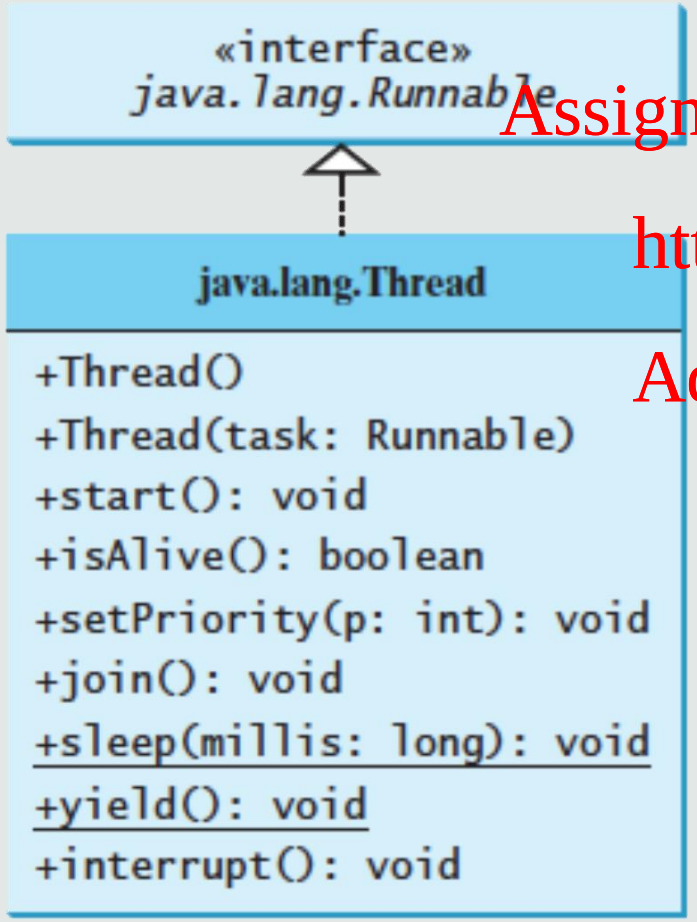
- Thread implementation dependent on the JVM implementation
- Nowadays, 1:1 mapping on threads provided by kernel
- E.g. in Linux, JVM calls POSIX thread functions

- Implemented by a class that defines a task
- run() will be executed as a thread
- Task passed to constructor of Thread class

```
java.lang.Runnable

public interface Runnable {

      public void run () ;

}
```

Java threads

Thread class



Creates an empty thread.	
Creates a thread for a specified task.	
Starts the thread that causes the `run()` method to be invoked by the JVM.	
Tests whether the thread is currently running.	
Sets priority `p` (ranging from 1 to 10) for this thread.	
Waits for this thread to finish.	
Puts a thread to sleep for a specified time in milliseconds.	
Causes a thread to pause temporarily and allow other threads to execute.	
Interrupts this thread.	

Java threads

Implementing Runnable vs Extending Thread

- `java.lang.Thread` implements `java.lang.Runnable`
- Why not extend `java.lang.Thread`?

Java threads

Implementing Runnable vs Extending Thread

- `java.lang.Thread` implements `java.lang.Runnable`
- Why not extend `java.lang.Thread`?

- Multiple inheritance not supported in Java
- Task and Runner (i.e. Thread) should not be tightly coupled
- Task object can be created without being a thread

- When is it a good idea to extend Thread?

Java threads

Example 1

```
public class RowSums {
    static final int N = 10;
    static int[][] matrix = new int[N][N];
    static int[] rowSums = new int[N];
    static class RowSumsTask implements Runnable {
        // ... see Example (2) ...
    }
    public static void main(String[] args) {
        // ... initialise matrix ...
        Thread[] threads = new Thread[N];
        for(int i=0; i<N; i++) {
        threads[i] = new Thread(new RowSumsTask(i));
        threads[i].start();
    }
    for(int i=0; i<N; i++)
        threads[i].join();
    }
}
```

Java threads

Example 2

```
public class RowSums {
    static final int N = 10;
    static int[][] matrix = new int[N][N];
    static int[] rowSums = new int[N];
    static class RowSumsTask implements Runnable {
        private int row;
        RowSumsTask(int row) { this.row = row; }
        public void run() {
            rowSums[row] = 0;
            for(int i=0; i<N; i++)
            rowSums[row] += matrix[row][i];
            }
        }
        public static void main(String[] args) {
                // ... see Example (1) ...
    }
}
```

Summary

Multi-programming / Multi-processing

- Process, thread, job, task
- Concurrency, parallelism
- Task, data and pipeline parallelism

Threads

- Java Thread Library
- `java.lang.Runnable`
- `java.lang.Thread`

Read

- Tanenbaum & Bos., Modern Operating Systems
  - Chapter 2
- Silberschatz et al., Operating System Concepts
  - Chapter 4

Next Lecture

- Introduction
- Operating System Architectures
- Processes
- Threads - Programming
- **Process Scheduling**
- Process Synchronisation
- Deadlocks
- Memory Management
- File Systems
- Input / Output
- Security and Virtualisation