Sistemi Operativi I

Corso di Laurea in Informatica

2023-2024

SAPIENZA
UNIVERSITÀ DI ROMA

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

Today: Beyond Single-Threaded Processes

- The process model assumes that a process is an executing program with a **single thread** of control

Today: Beyond Single-Threaded Processes

- The process model assumes that a process is an executing program with a **single thread** of control
- All modern operating systems provide features enabling a process to contain **multiple threads** of control

Today: Beyond Single-Threaded Processes

- The process model assumes that a process is an executing program with a **single thread** of control
- All modern operating systems provide features enabling a process to contain **multiple threads** of control
- We introduce many concepts associated with multi-threaded computer systems

Today: Beyond Single-Threaded Processes

- The process model assumes that a process is an executing program with a **single thread** of control
- All modern operating systems provide features enabling a process to contain **multiple threads** of control
- We introduce many concepts associated with multi-threaded computer systems
- We look at a number of issues related to multi-threaded programming and its effect on the design of operating systems

Threads: Overview

- A **thread** is a basic unit of CPU utilization, consisting of a program counter, a stack, a set of registers and a thread ID

Threads: Overview

- A **thread** is a basic unit of CPU utilization, consisting of a program counter, a stack, a set of registers and a thread ID
- Traditional (heavyweight) processes have a single thread of control
  - There is only one program counter, and one sequence of instructions that can be carried out at any given time

Threads: Overview

- A **thread** is a basic unit of CPU utilization, consisting of a program counter, a stack, a set of registers and a thread ID
- Traditional (heavyweight) processes have a single thread of control
  - There is only one program counter, and one sequence of instructions that can be carried out at any given time
- Multi-threaded applications have multiple threads within a single process, each having their own program counter, stack, and set of registers
  - But sharing common code, data, and certain structures, such as open files

Process vs. Thread

- A **process** defines the address space, text (code), data, resources, etc.

Process vs. Thread

- A **process** defines the address space, text (code), data, resources, etc.
- A **thread** defines a single sequential execution stream *within* a process (i.e., program counter, stack, registers)

Process vs. Thread

- A **process** defines the address space, text (code), data, resources, etc.
- A **thread** defines a single sequential execution stream *within* a process (i.e., program counter, stack, registers)
- A thread is bound to a specific process

Process vs. Thread

- Each process may have several threads of control within it

Process vs. Thread

- Each process may have several threads of control within it
- The process' address space is shared among all its threads

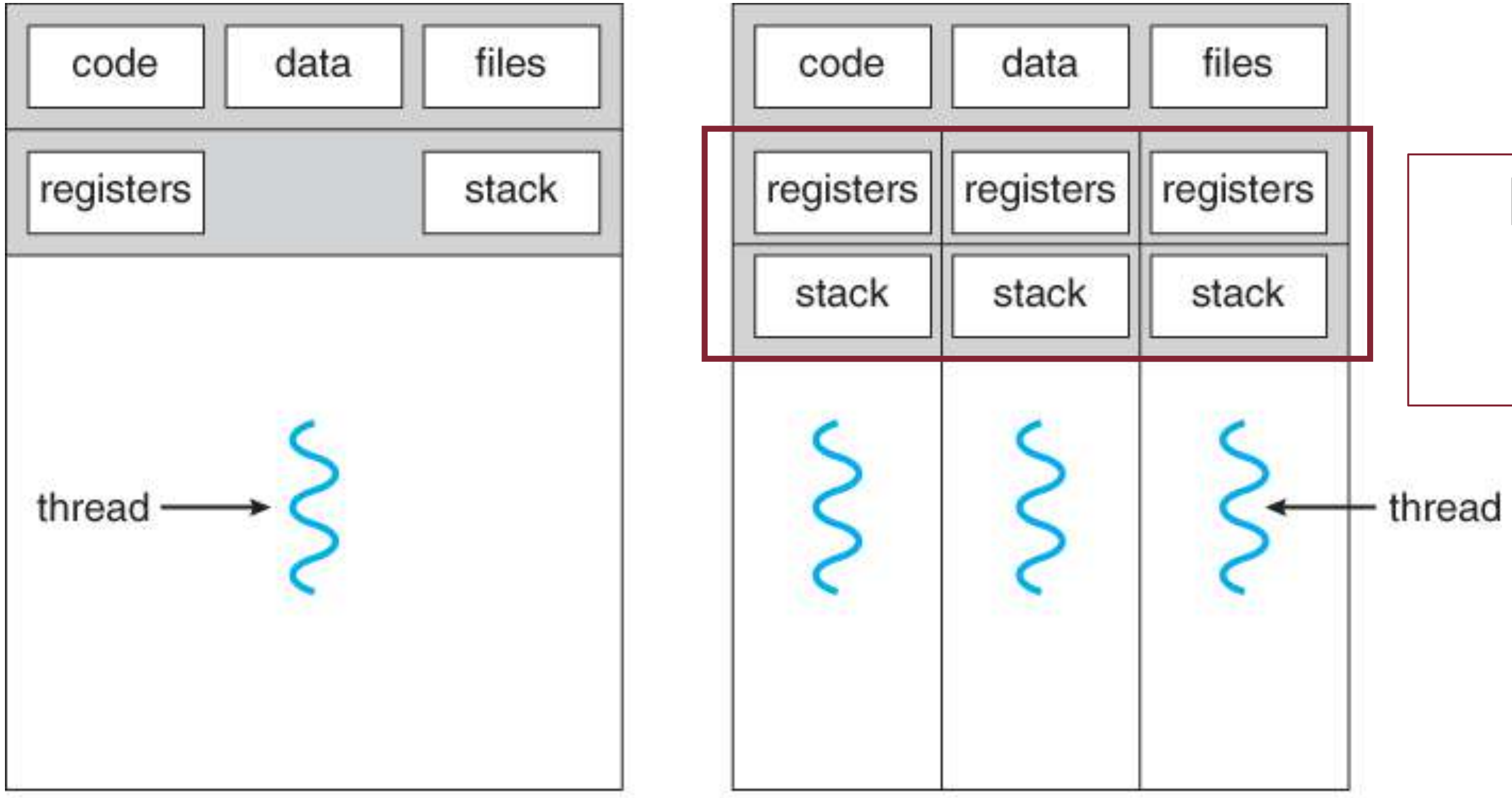
Process vs. Thread

- Each process may have several threads of control within it
- The process' address space is shared among all its threads
- No system calls are required for threads to cooperate with each other

Process vs. Thread

- Each process may have several threads of control within it
- The process' address space is shared among all its threads
- No system calls are required for threads to cooperate with each other
- Simpler than message passing and shared memory

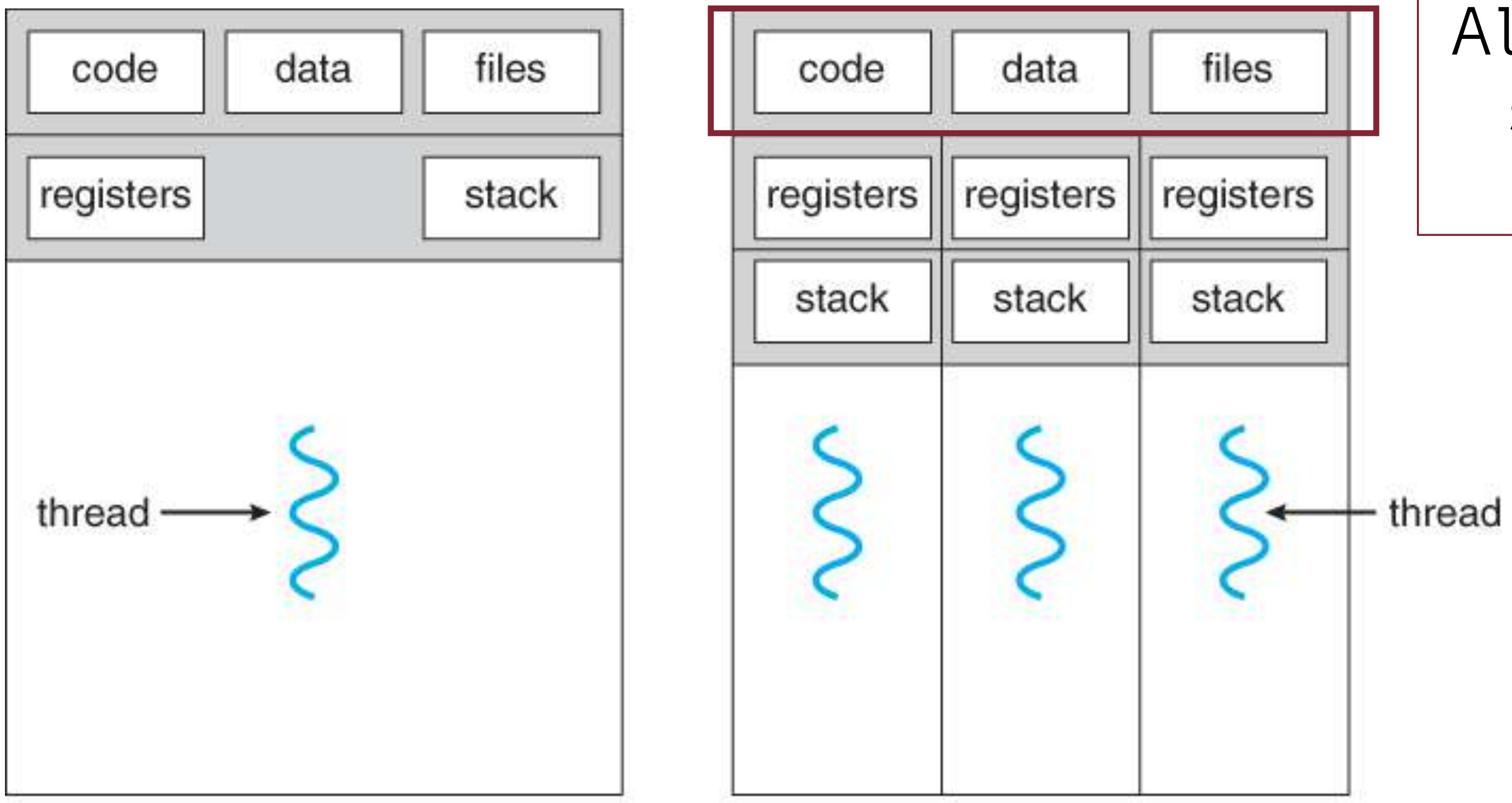
Single- vs. Multi-Threaded Process



single-threaded process
multithreaded process

Each thread has its own independent set of registers and "state"

Single- vs. Multi-Threaded Process

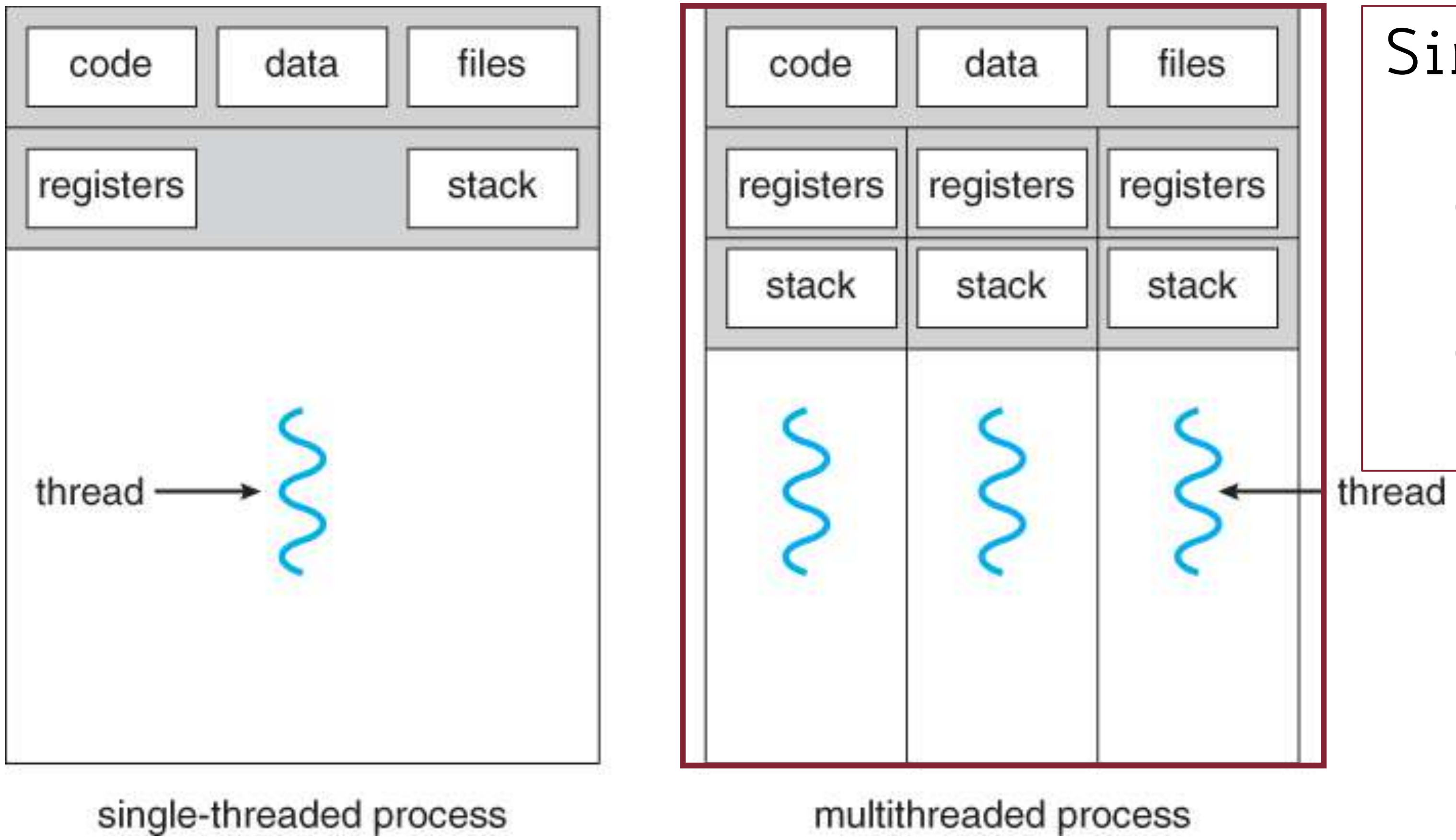


single-threaded process

multithreaded process

All the threads of a process share the same code and "global" resources

Single- vs. Multi-Threaded Process



Since all the threads live in the same address space, communication between them is easier than communication between processes

Threads: Motivation

- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others

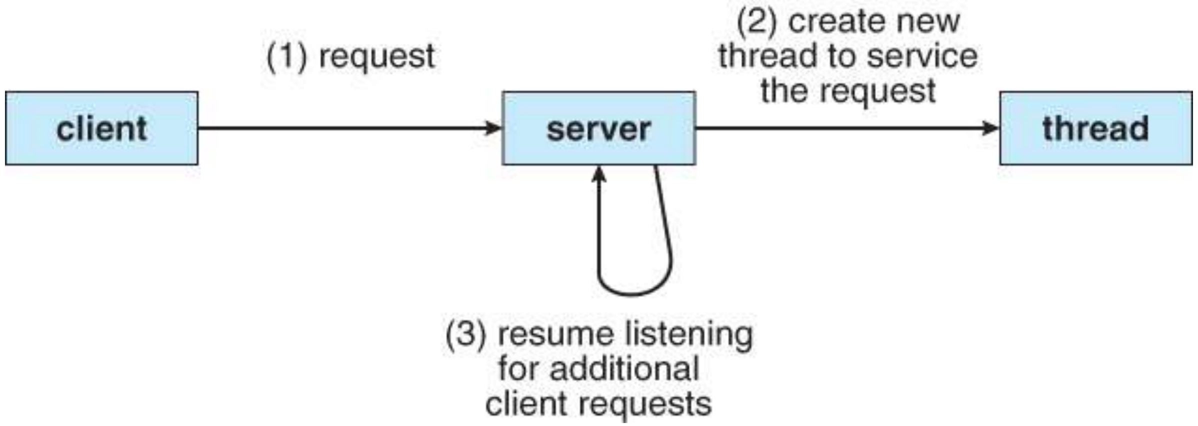
Threads: Motivation

- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others
- This is particularly true when one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking

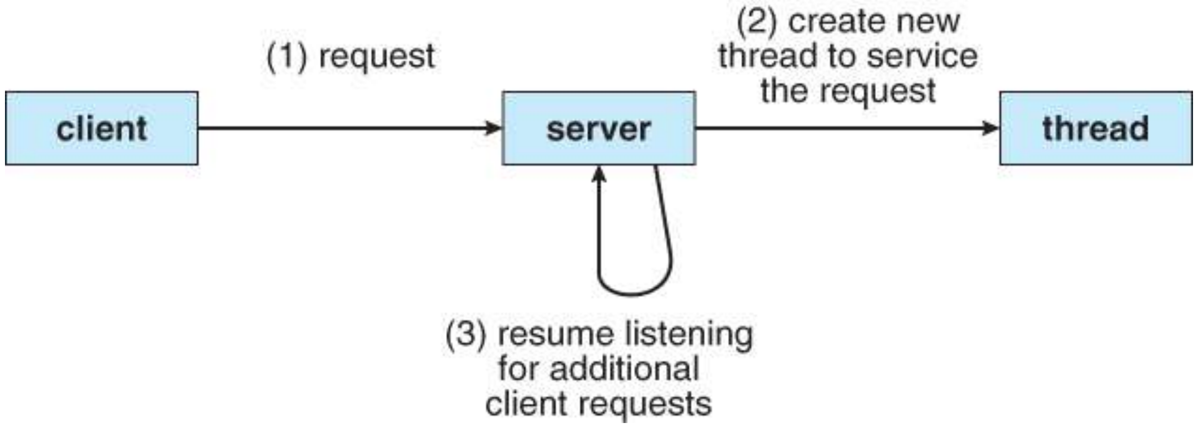
Threads: Motivation

- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others
- This is particularly true when one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking
- Example: word processor
    - a thread may check grammar while another thread handles user input (keystrokes), and a third does periodic backups of the file being edited

Multi-threaded Web Server

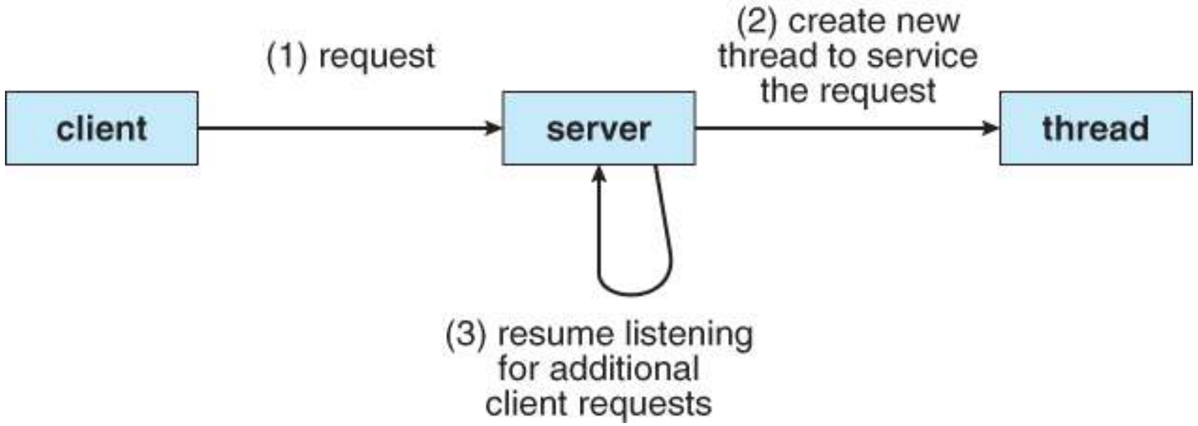


Multi-threaded Web Server



Multiple threads allow for multiple requests to be satisfied simultaneously, without having to serve requests sequentially or to fork off separate processes for every incoming request

Multi-threaded Web Server



What if the server process spawns off a new process for each incoming request rather than a thread?

Multiple Processes vs. Multiple Threads

- Theoretically, each sub-task of an application could be implemented as a new single-threaded process rather than a multi-threaded process

Multiple Processes vs. Multiple Threads

- Theoretically, each sub-task of an application could be implemented as a new single-threaded process rather than a multi-threaded process
- There are at least **2 reasons** why this is not the best choice:

Multiple Processes vs. Multiple Threads

- Theoretically, each sub-task of an application could be implemented as a new single-threaded process rather than a multi-threaded process
- There are at least **2 reasons** why this is not the best choice:
  - Inter-thread communication is significantly quicker than inter-process one

Multiple Processes vs. Multiple Threads

- Theoretically, each sub-task of an application could be implemented as a new single-threaded process rather than a multi-threaded process
- There are at least **2 reasons** why this is not the best choice:
  - Inter-thread communication is significantly quicker than inter-process one
  - Context-switches between threads is a lot faster than between processes

Threads: Benefits

- 4 main benefits:

Threads: Benefits

- 4 main benefits:
  - Responsiveness → one thread may provide rapid response while other threads are blocked or slowed down doing intensive computations

Threads: Benefits

- 4 main benefits:
  - Responsiveness → one thread may provide rapid response while other threads are blocked or slowed down doing intensive computations
  - Resource sharing → threads share common code, data, and address space

Threads: Benefits

- 4 main benefits:
  - Responsiveness → one thread may provide rapid response while other threads are blocked or slowed down doing intensive computations
  - Resource sharing → threads share common code, data, and address space
  - Economy → creating and managing threads (and context switches between them) is much faster than performing the same tasks for processes

Threads: Benefits

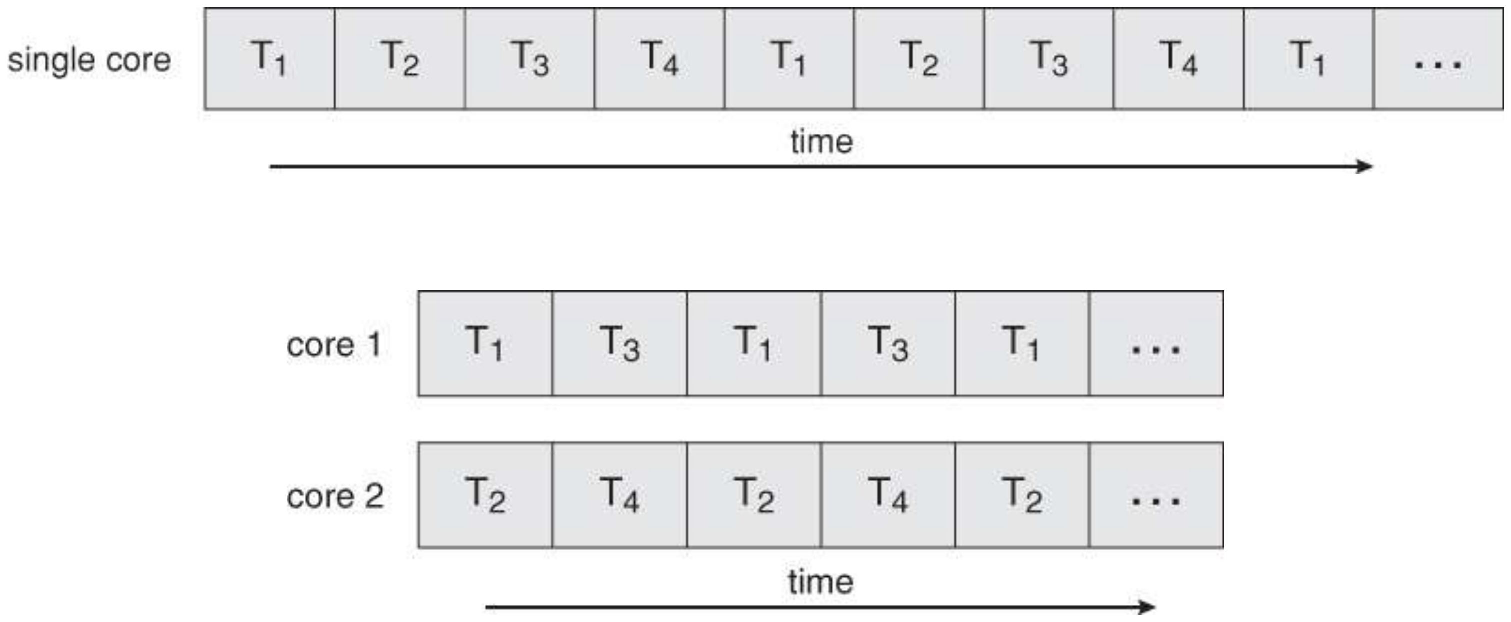
- 4 main benefits:
  - Responsiveness → one thread may provide rapid response while other threads are blocked or slowed down doing intensive computations
  - Resource sharing → threads share common code, data, and address space
  - Economy → creating and managing threads (and context switches between them) is much faster than performing the same tasks for processes
  - Scalability (multi-processor architectures) → A single threaded process can only run on one CPU, whereas a multi-threaded process may be split amongst all available processors/cores

Multi-core Programming

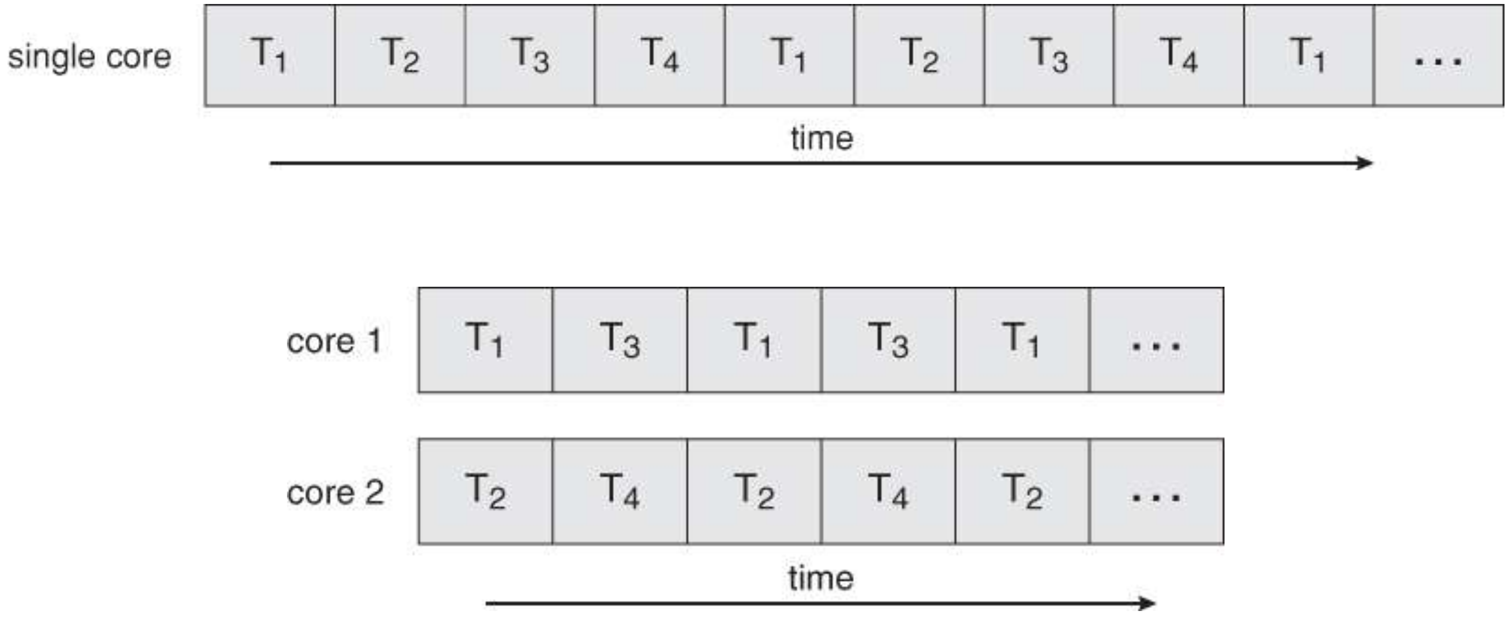
- A recent trend in computer architecture is to produce chips with multiple cores, or CPUs on a single chip
- A multi-threaded application running on a traditional single-core chip would have to interleave the threads
- On a multi-core chip, however, threads could be spread across the available cores, allowing **true parallel processing**!

Single- vs. Multi-core Programming

Single- vs. Multi-core Programming

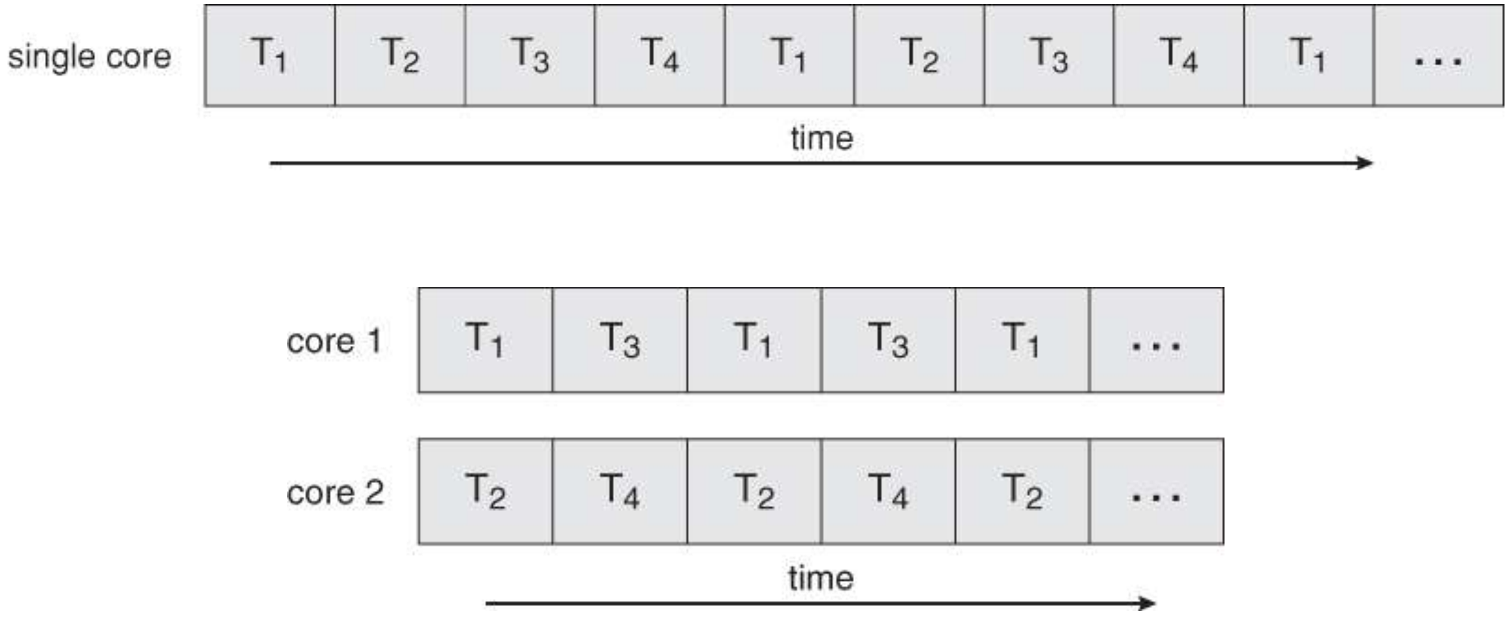


Single- vs. Multi-core Programming



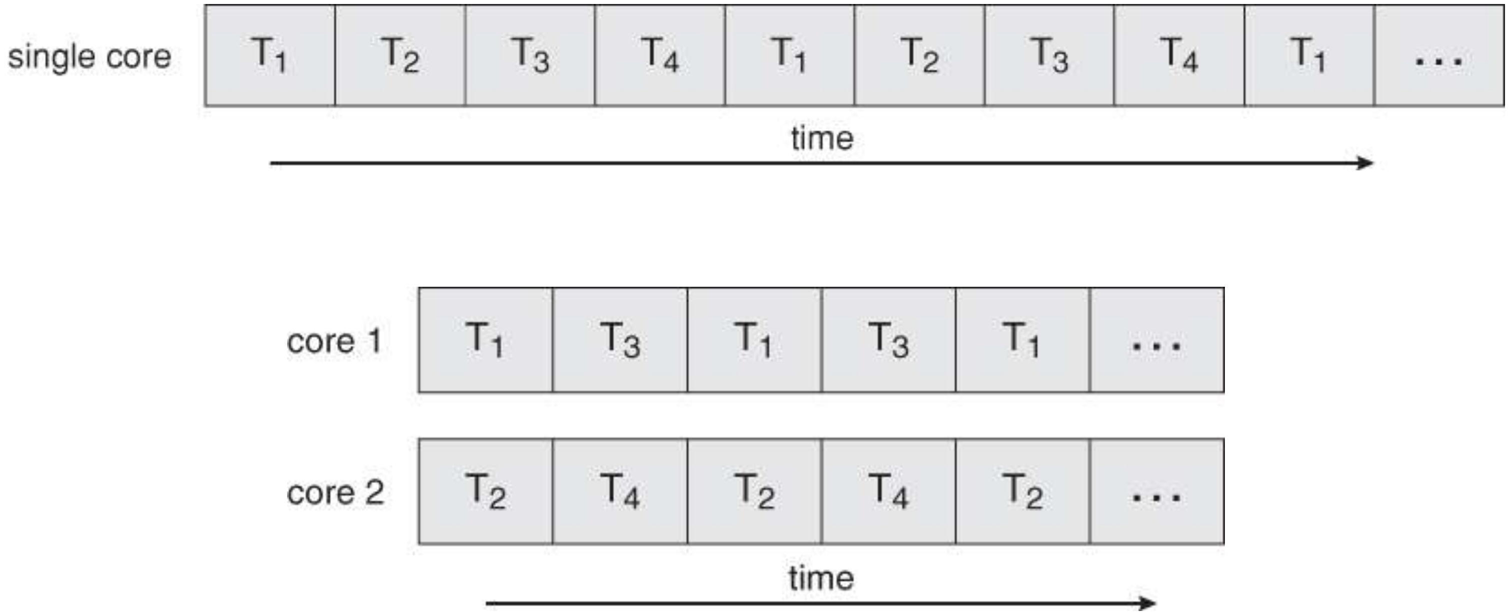
Multi-core chips require new OS scheduling algorithms to make better use of the multiple cores available

Single- vs. Multi-core Programming



CPUs have been developed to support more simultaneous threads per core in hardware (e.g., Intel's hyper-threading)

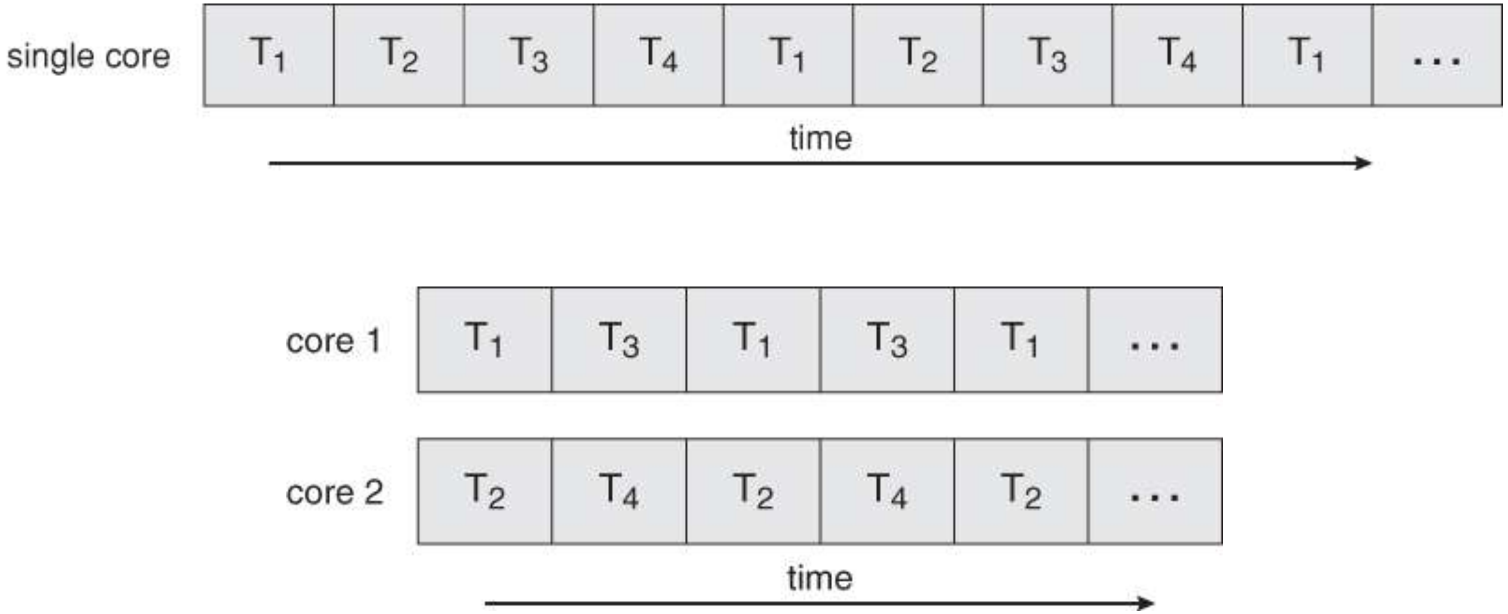
Single- vs. Multi-core Programming



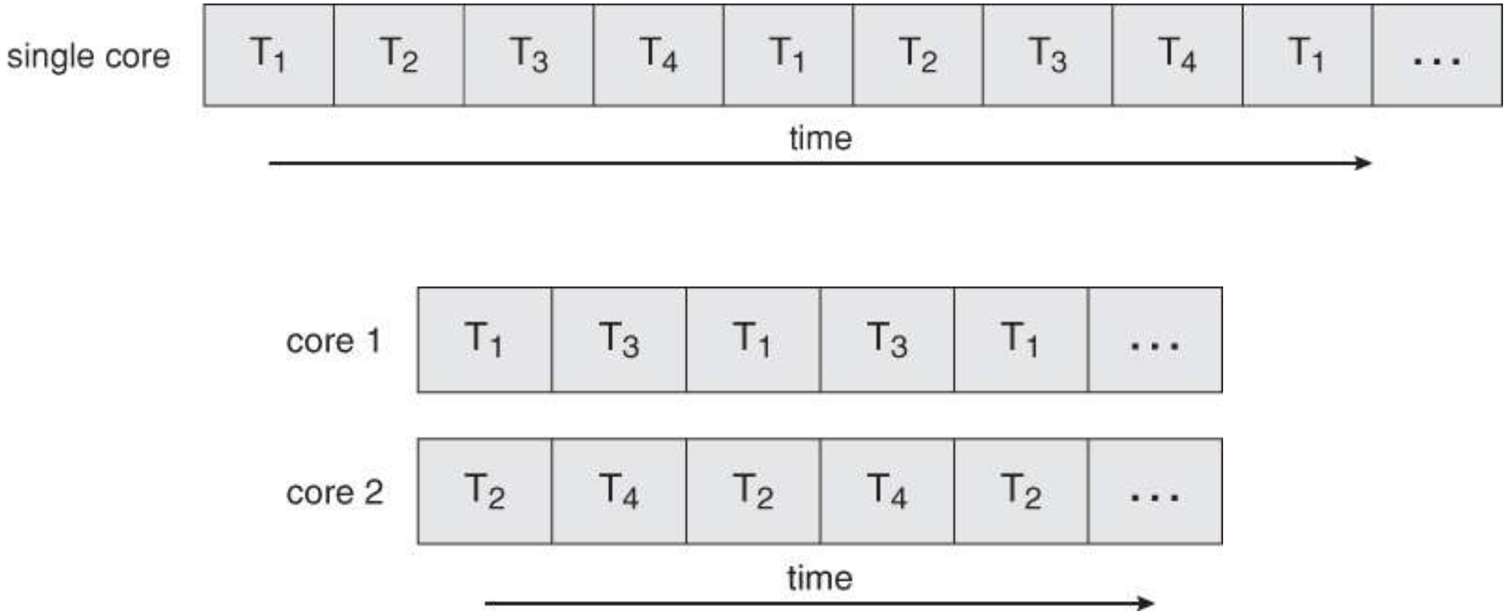
Hyper-threading

Each physical core appears as two processors to the OS, allowing concurrent scheduling of two threads per core

Single- vs. Multi-core Programming



Single- vs. Multi-core Programming



Concurrency

vs.

Parallelism

Types of Parallelism

- In theory, there are **2 ways** to parallelize the workload:

Types of Parallelism

- In theory, there are **2 ways** to parallelize the workload:
  - **Data parallelism:** divides the data up amongst multiple cores (threads), and performs the same task on each chunk of the data

Types of Parallelism

- In theory, there are **2 ways** to parallelize the workload:
  - **Data parallelism:** divides the data up amongst multiple cores (threads), and performs the same task on each chunk of the data
  - **Task parallelism:** divides the different tasks to be performed among the different cores and performs them simultaneously

Example: A Pure CPU-bound Task

- Suppose you are asked to implement a simple program that:
  - Takes as input a positive integer N
  - Produces as output the total sum from 1 to N

Example: A Pure CPU-bound Task

- Suppose you are asked to implement a simple program that:
  - Takes as input a positive integer N
  - Produces as output the total sum from 1 to N
- The easiest solution is something as follows:

```
int sum = 0;
for (int i=1; i <= N; ++i) {
    sum += i;
}
return sum;
```

Example: A Pure CPU-bound Task

- Suppose you are asked to implement a simple program that:
  - Takes as input a positive integer N
  - Produces as output the total sum from 1 to N
- The easiest solution is something as follows:

```
int sum = 0;
for (int i=1; i <= N; ++i) {
    sum += i;
}
return sum;
```

CPU-bound

Example: A Pure CPU-bound Task

- If N grows large it may take a while…

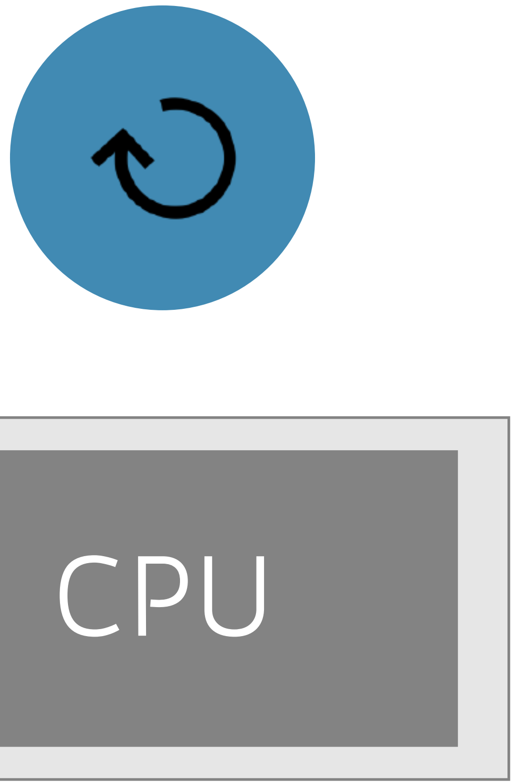
Example: A Pure CPU-bound Task

- If N grows large it may take a while…
- Based on the underlying HW, can we improve the performance of the previously single-threaded process?

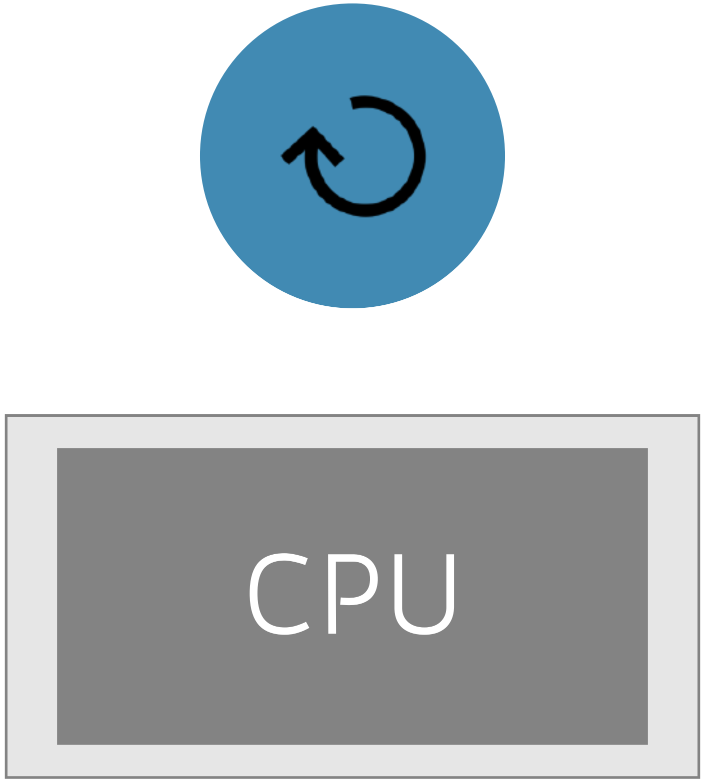
Example: A Pure CPU-bound Task

- If N grows large it may take a while…
- Based on the underlying HW, can we improve the performance of the previously single-threaded process?
- We will consider the following setups:
  - Number of CPU cores: 1 vs. M
  - Processes/Threads: 1/1 vs. M/1 vs. 1/M

1 CPU Core, 1 Process, 1 Thread



1 CPU Core, 1 Process, 1 Thread

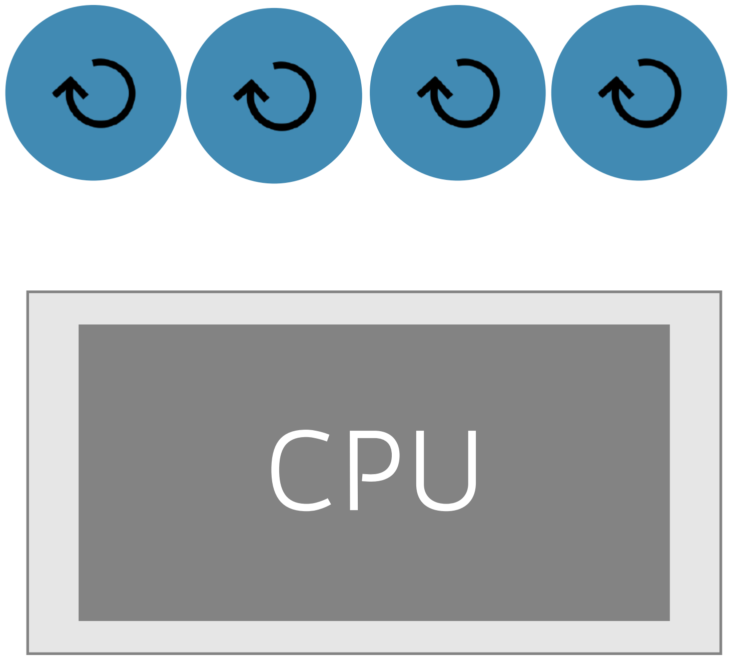


No Parallelism

No Concurrency

1 CPU Core, M Processes, 1 Thread

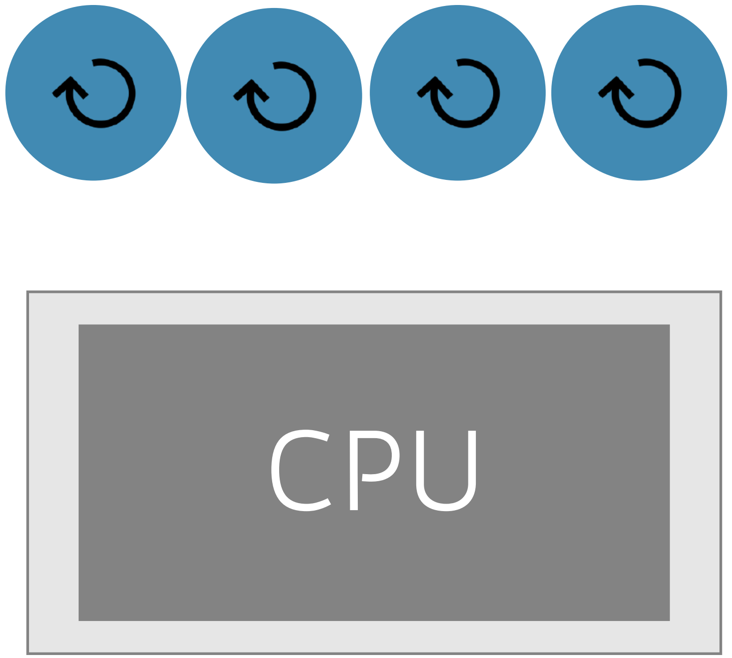
Divide N into M chunks: {[1, …, N/M], [(N/M)+1, … , 2N/M], …, [(M-1)(N/M)+1, …, N/M]}



1 CPU Core, M Processes, 1 Thread

Divide N into M chunks: {[1, …, N/M], [(N/M)+1, … , 2N/M], …, [(M-1)(N/M)+1, …, N/M]}

e.g., N = 1000; M=8: {[1, …, 125], [126, …, 250], …, [876, …, 1000]}

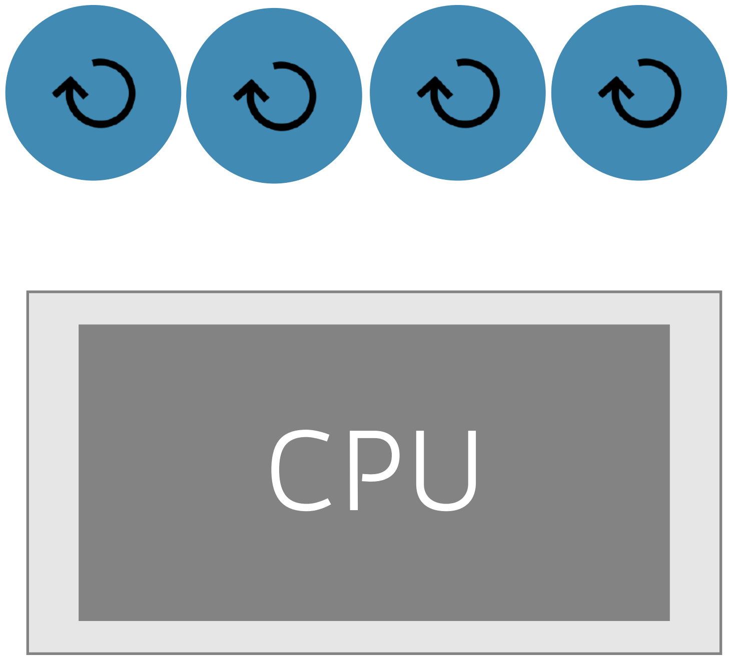


1 CPU Core, M Processes, 1 Thread

Divide N into M chunks: {[1, …, N/M], [(N/M)+1, … , 2N/M], …, [(M-1)(N/M)+1, …, N/M]}

e.g., N = 1000; M=8: {[1, …, 125], [126, …, 250], …, [876, …, 1000]}

The i-th process computes the sum of the numbers contained in the i-th chunk

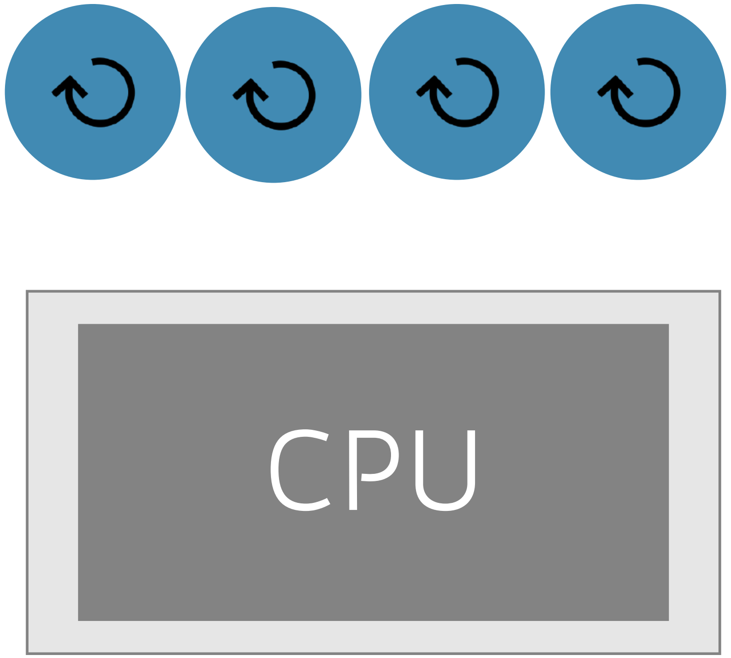


1 CPU Core, M Processes, 1 Thread

Divide N into M chunks: {[1, …, N/M], [(N/M)+1, … , 2N/M], …, [(M-1)(N/M)+1, …, N/M]}

e.g., N = 1000; M=8: {[1, …, 125], [126, …, 250], …, [876, …, 1000]}

The i-th process computes the sum of the numbers contained in the i-th chunk

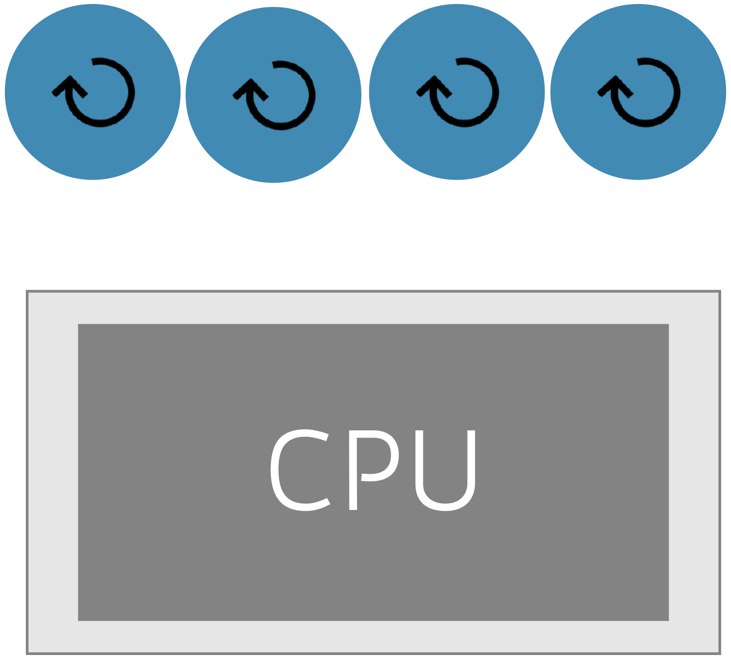


No Parallelism

Concurrency (among processes)

1 CPU Core, M Processes, 1 Thread

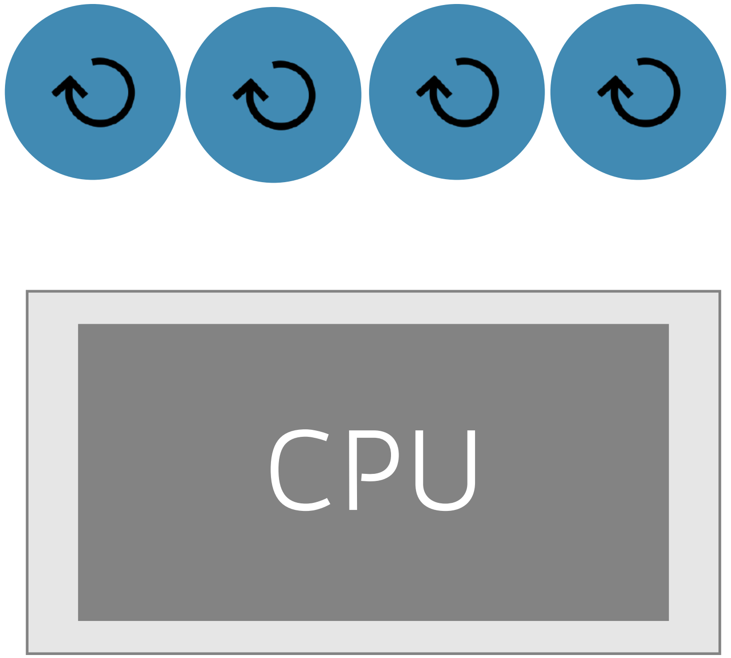
Will this solution get any speedup to the whole computation?



1 CPU Core, M Processes, 1 Thread

Will this solution get any speedup to the whole computation?

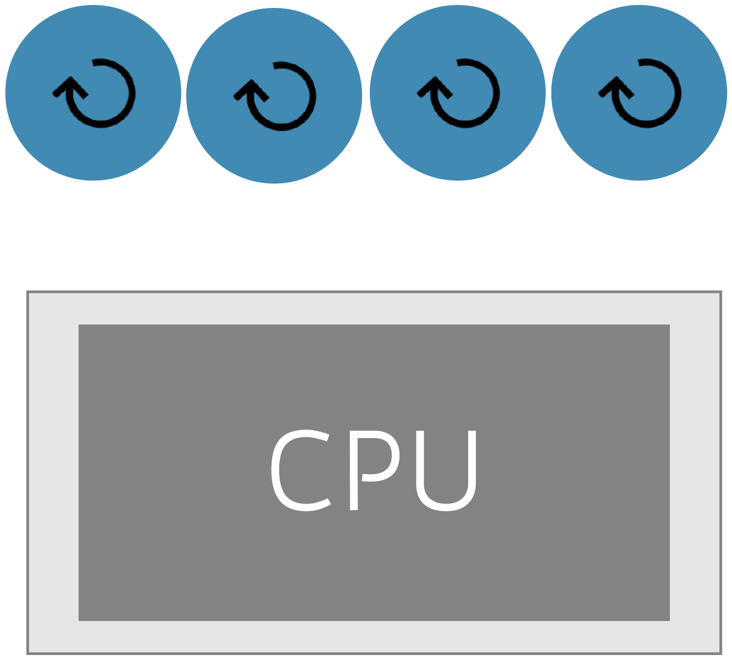
NO!



1 CPU Core, M Processes, 1 Thread

Will this solution get any speedup to the whole computation?

Only one process is running on a single CPU core

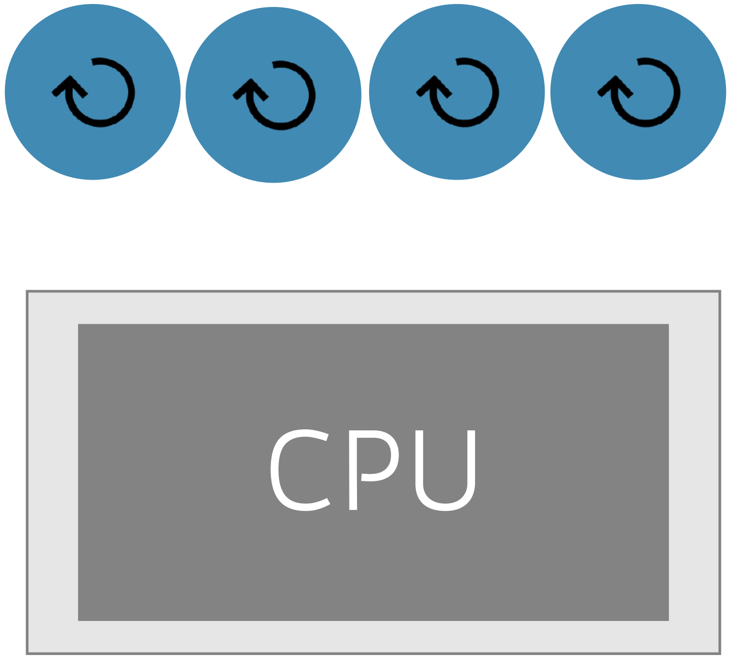


1 CPU Core, M Processes, 1 Thread

Will this solution get any speedup to the whole computation?

Only one process is running on a single CPU core

All the M processes must finish to get the final result

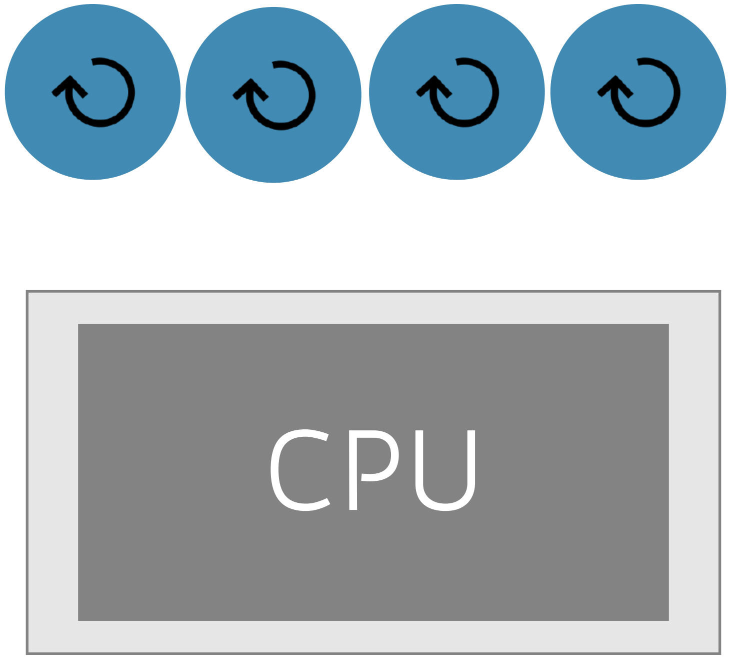


1 CPU Core, M Processes, 1 Thread

Will this solution get any speedup to the whole computation?

Only one process is running on a single CPU core

All the M processes must finish to get the final result

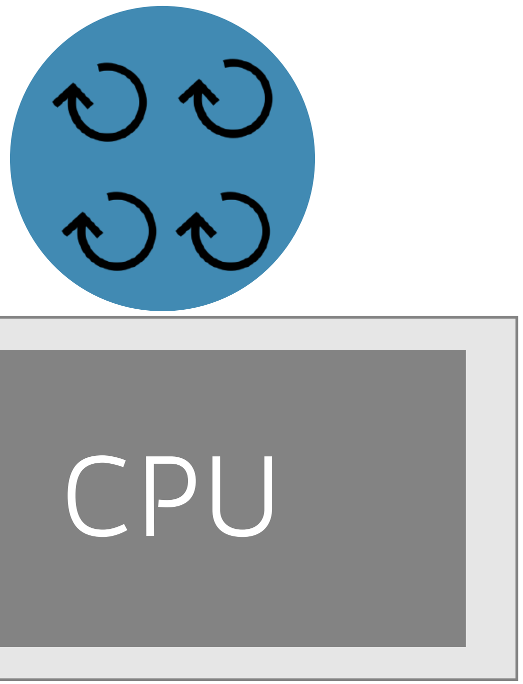


Eventually, each process must communicate its partial sum to the others

Inter-Process Communication

1 CPU Core, 1 Process, M Threads

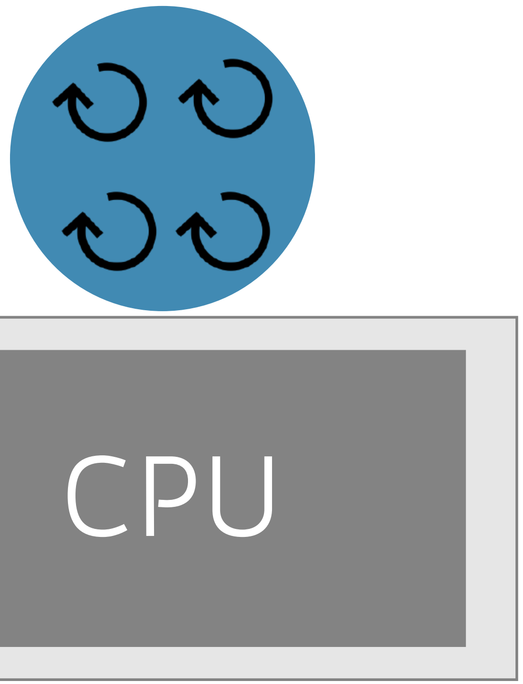
Will this solution get any speedup to the whole computation?



1 CPU Core, 1 Process, M Threads

Will this solution get any speedup to the whole computation?

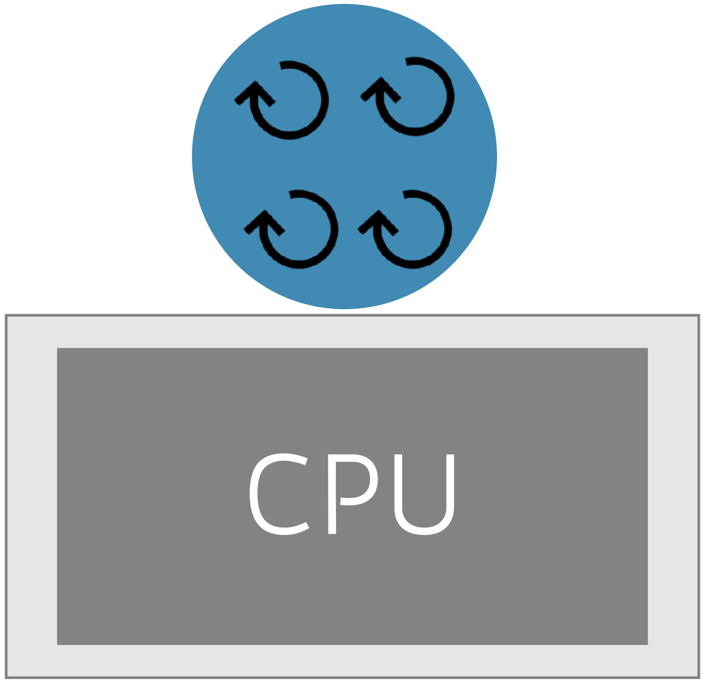
NO!



1 CPU Core, 1 Process, M Threads

Will this solution get any speedup to the whole computation?

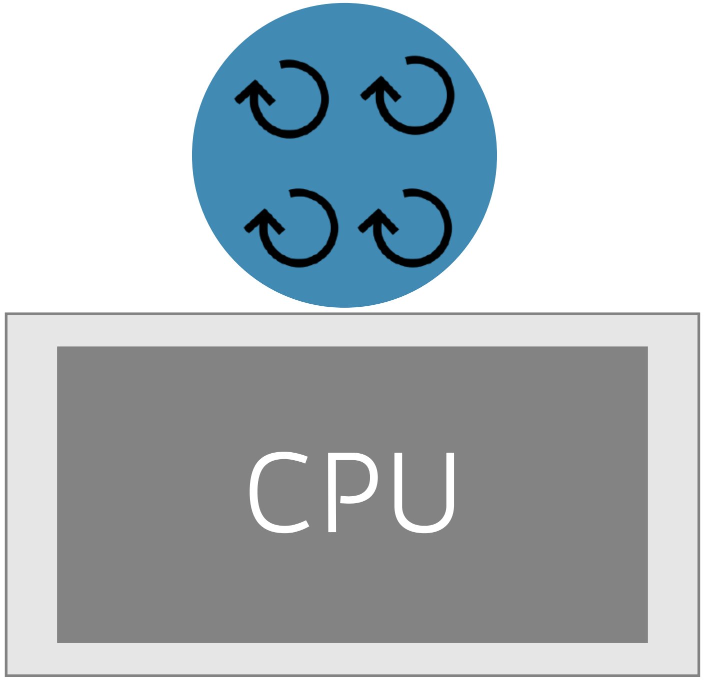
Only one thread is running on a single CPU core



1 CPU Core, 1 Process, M Threads

Will this solution get any speedup to the whole computation?

Only one thread is running on a single CPU core

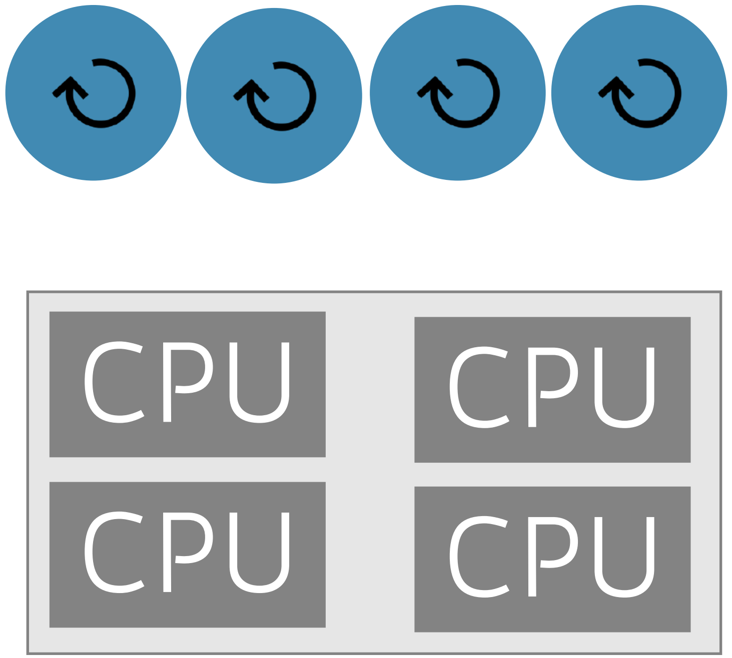


The only advantage is that each thread can easily share its partial sum with the others!

No Inter-Process Communication

M CPU Cores, M Processes, 1 Thread

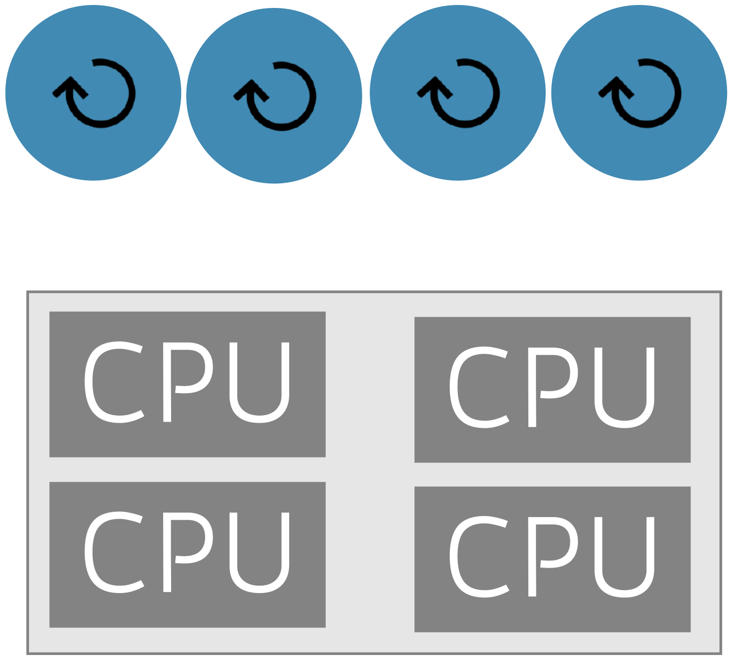
Will this solution get any speedup to the whole computation?



M CPU Cores, M Processes, 1 Thread

Will this solution get any speedup to the whole computation?

YES!

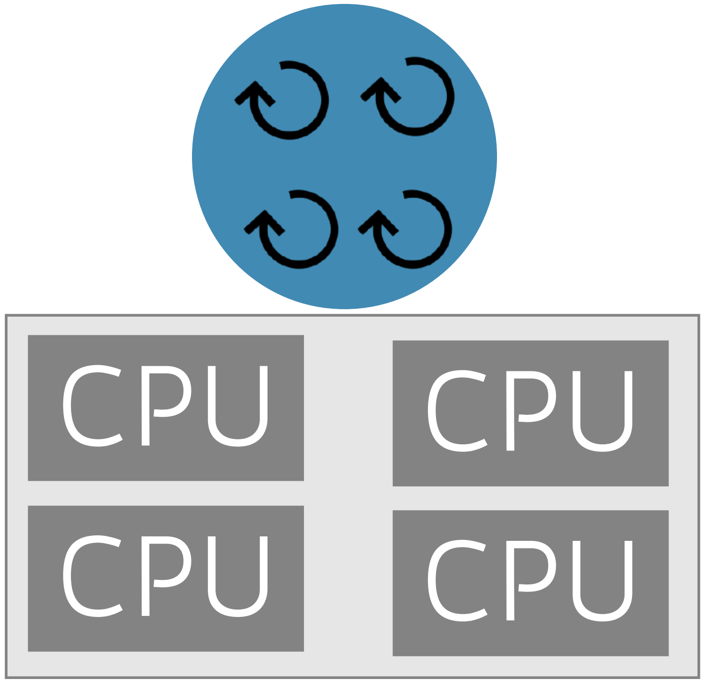


True Parallelism

Still, each process must communicate its partial sum to the others

M CPU Cores, 1 Process, M Threads

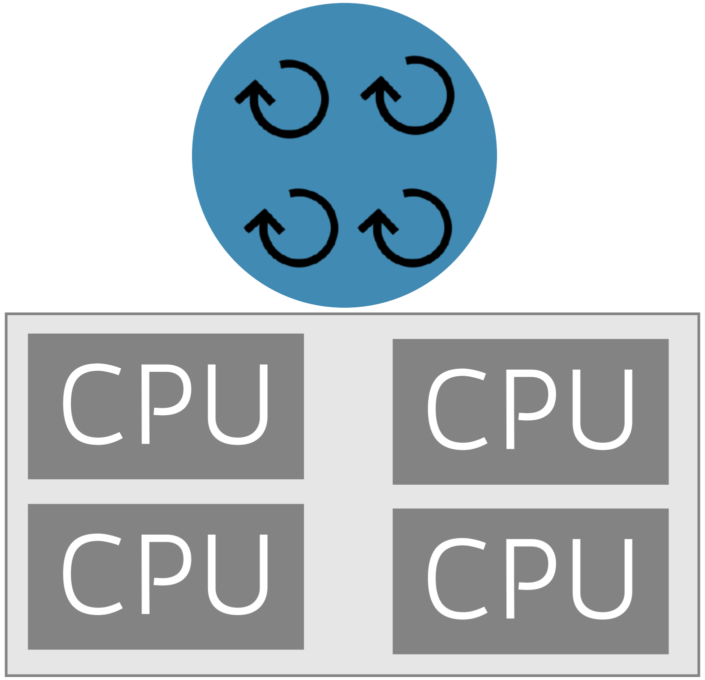
Will this solution get any speedup to the whole computation?



M CPU Cores, 1 Process, M Threads

Will this solution get any speedup to the whole computation?

YES!



True Parallelism

No Inter-Process Communication

A Mixed CPU- and IO-bound Task

- There are a lot of complex problems that are **both** CPU and IO intensive **in phases**

A Mixed CPU- and IO-bound Task

- There are a lot of complex problems that are **both** CPU and IO intensive **in phases**
- IO-intensive when CPU is free, CPU-intensive when IO finished or side-by-side

A Mixed CPU- and IO-bound Task

- There are a lot of complex problems that are **both** CPU and IO intensive **in phases**
- IO-intensive when CPU is free, CPU-intensive when IO finished or side-by-side
- For instance:
    - Distributed processing of high volumes of data

A Mixed CPU- and IO-bound Task

- There are a lot of complex problems that are **both** CPU and IO intensive **in phases**
- IO-intensive when CPU is free, CPU-intensive when IO finished or side-by-side
- For instance:
  - Distributed processing of high volumes of data
  - Disk defragmentation

A Mixed CPU- and IO-bound Task

- There are a lot of complex problems that are **both** CPU and IO intensive **in phases**
- IO-intensive when CPU is free, CPU-intensive when IO finished or side-by-side
- For instance:
  - Distributed processing of high volumes of data
  - Disk defragmentation
  - Compression/Decompression algorithms (side-by-side)

A Mixed CPU- and I/O-bound Task

- In all these cases, multi-threading can be useful **even on a single-core CPU**

A Mixed CPU- and I/O-bound Task

- In all these cases, multi-threading can be useful **even on a single-core CPU**
- Indeed, it might pay to split CPU- and I/O-intensive tasks of an application into separate threads

A Mixed CPU- and I/O-bound Task

- In all these cases, multi-threading can be useful **even on a single-core CPU**
- Indeed, it might pay to split CPU- and I/O-intensive tasks of an application into separate threads
- This way the CPU- and I/O-bound threads can alternate on the CPU

A Mixed CPU- and I/O-bound Task

- In all these cases, multi-threading can be useful **even on a single-core CPU**
- Indeed, it might pay to split CPU- and I/O-intensive tasks of an application into separate threads
- This way the CPU- and I/O-bound threads can alternate on the CPU
- This slows down the CPU-bound thread a little, but reduces or eliminates the I/O-bound gap

Classifying OSs

address space

thread

Classifying OSs

single thread

address space

thread

multiple threads

Classifying OSs

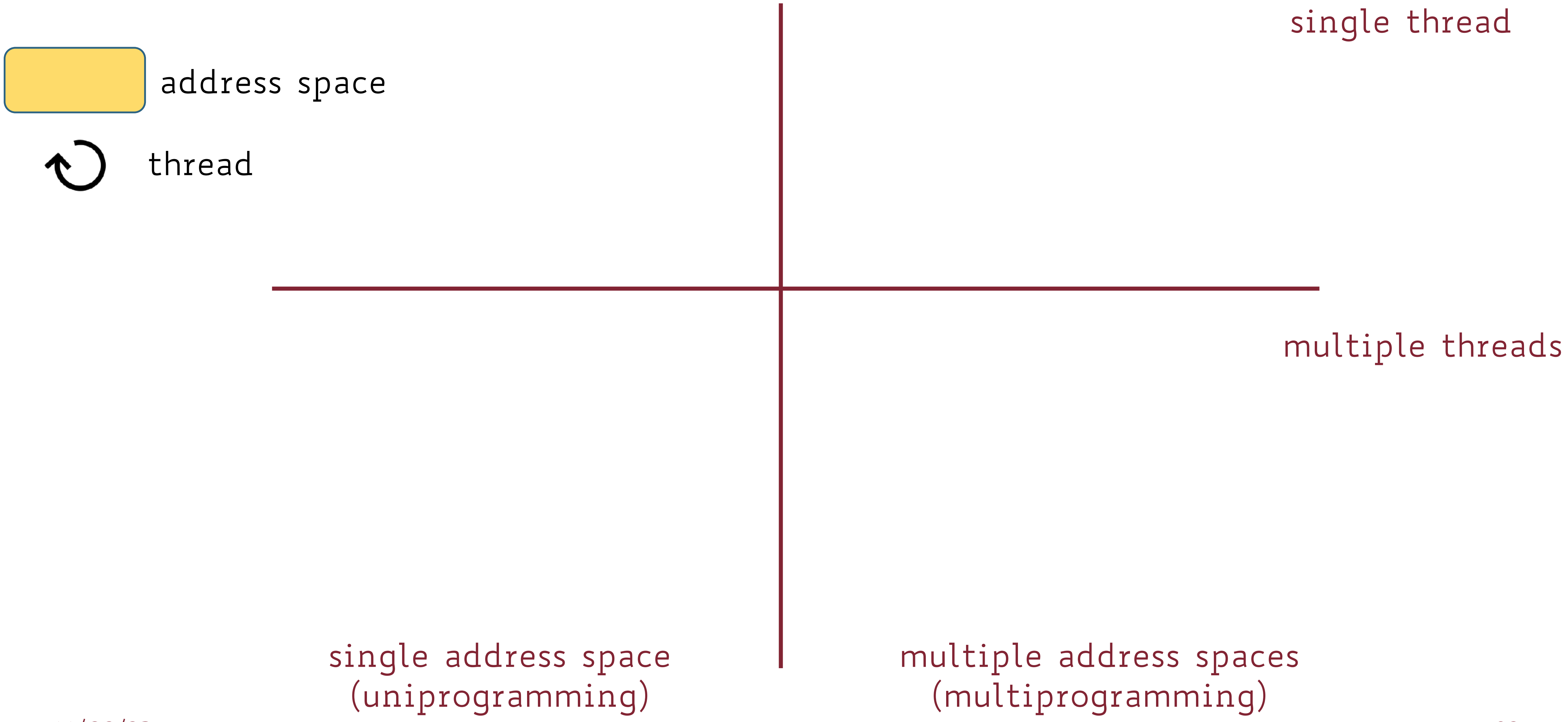
address space

thread

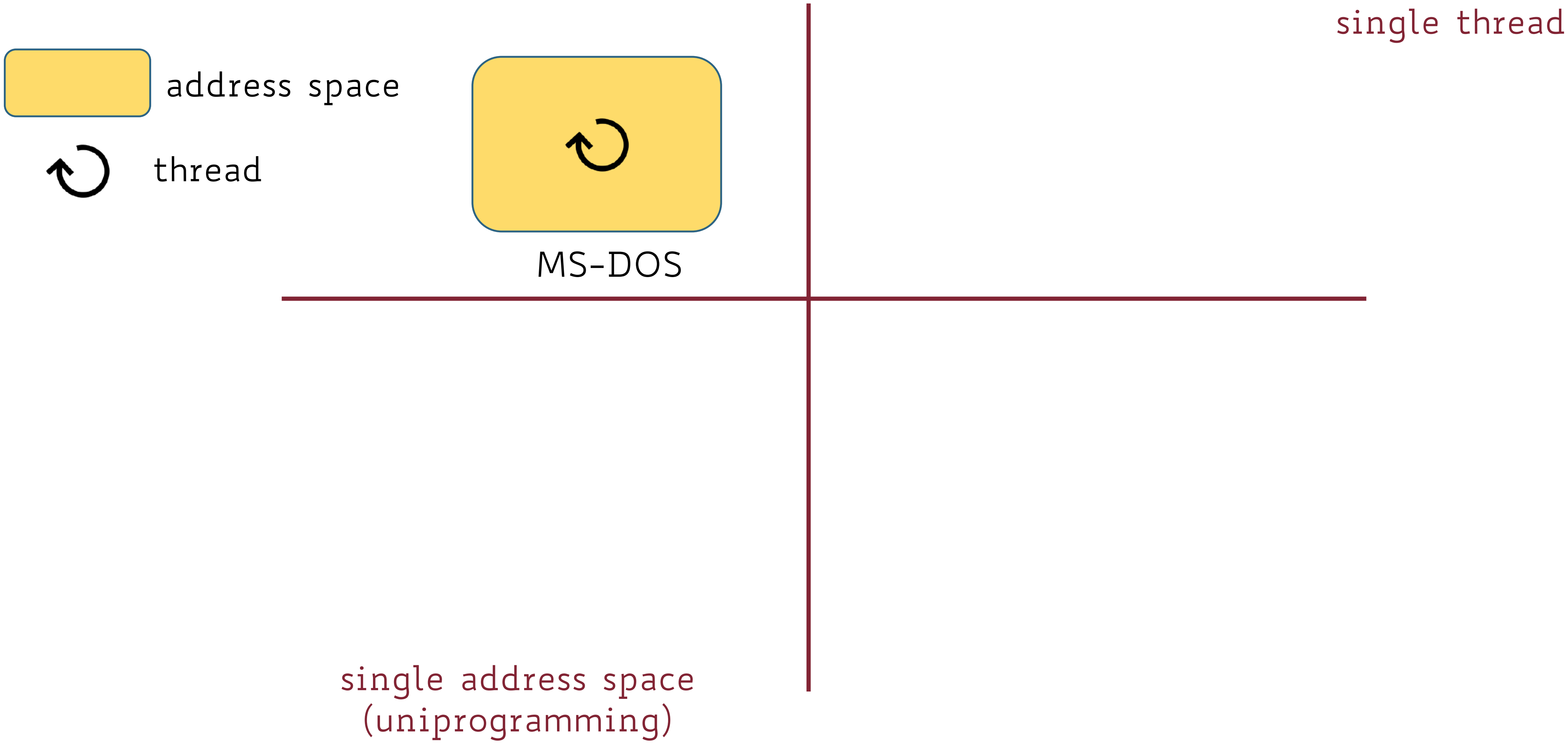
single address space
(uniprogramming)

multiple address spaces
(multiprogramming)

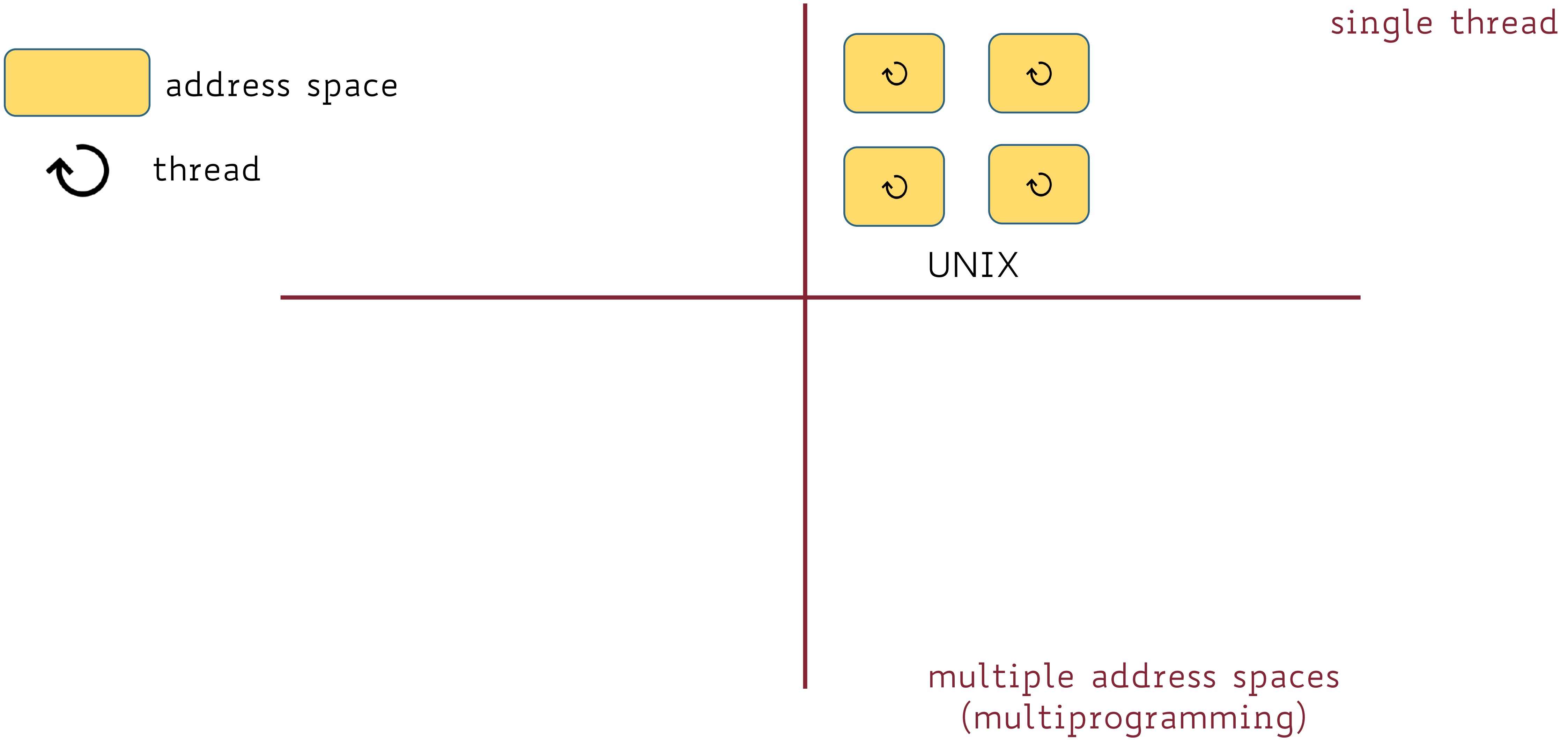
Classifying OSs



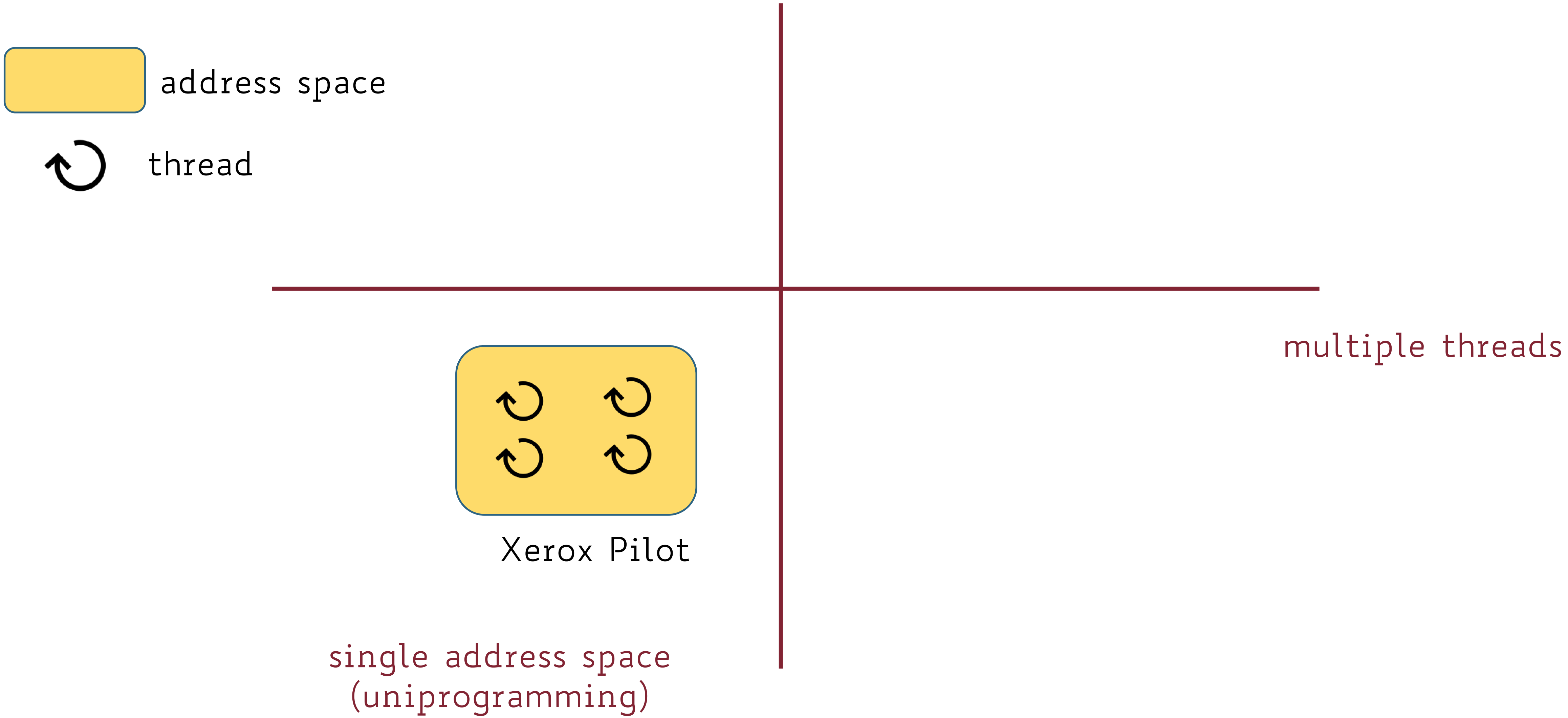
Classifying OSs



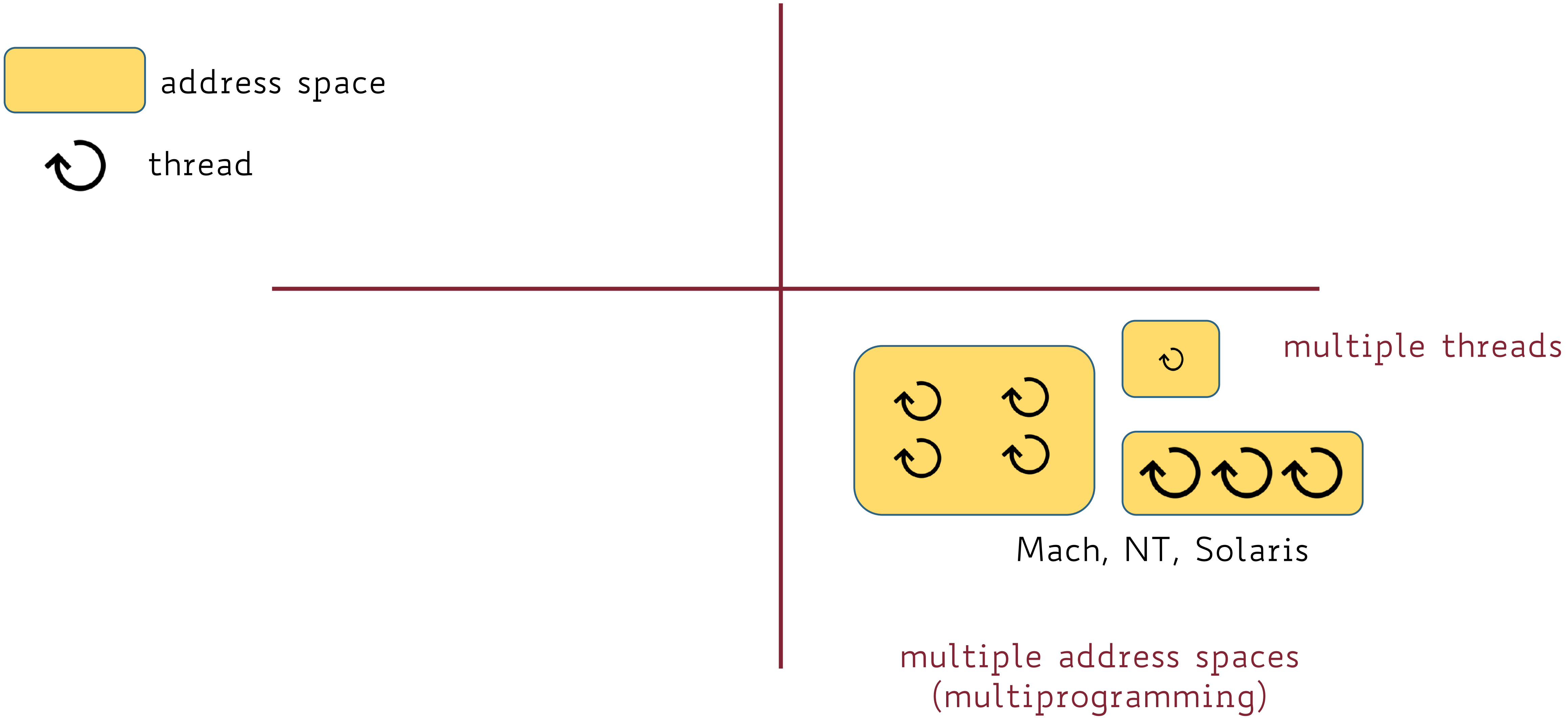
Classifying OSs



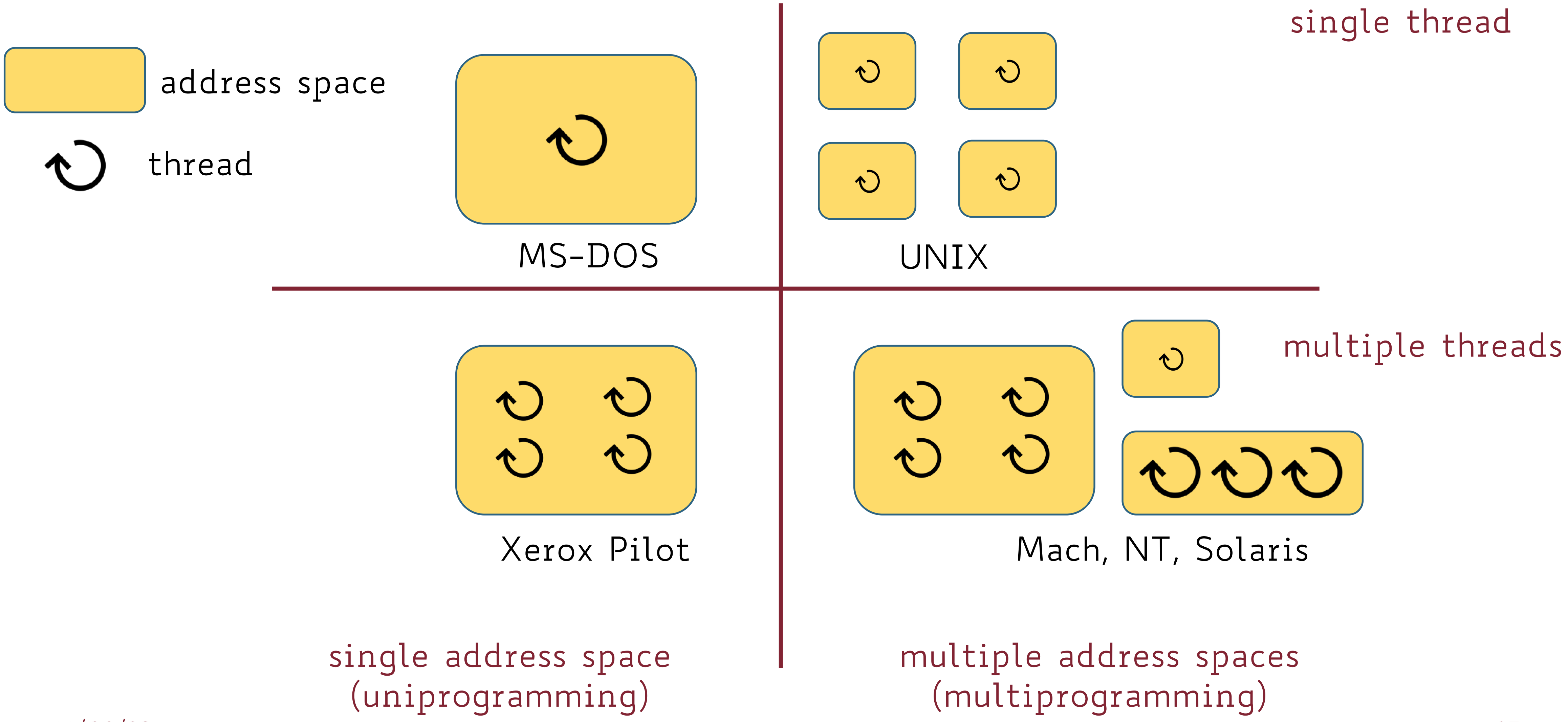
Classifying OSs



Classifying OSs



Classifying OSs



Summary

- A **thread** is a single execution stream within a process

Summary

- A thread is a single execution stream within a process
- Thread vs. Process:
  - common vs. separate address spaces → quicker communication
  - lightweight vs. heavyweight → faster context switching

Summary

- A **thread** is a single execution stream within a process
- Thread vs. Process:
  - common vs. separate address spaces → **quicker communication**
  - lightweight vs. heavyweight → **faster context switching**
- On a single core:
  - Fully CPU-bound processes do not take advantage of multi-threading
  - Concurrency between threads in mixed CPU- and I/O-bound processes