Chapter 1: Parallel computing introduction

Zhiming Zhao

University of Amsterdam

Outline

1. Basic terminologies
2. From single core to multi and many cores
3. From sequential to concurrency
4. Data parallelization and Task parallelization
5. Speed up and efficiency

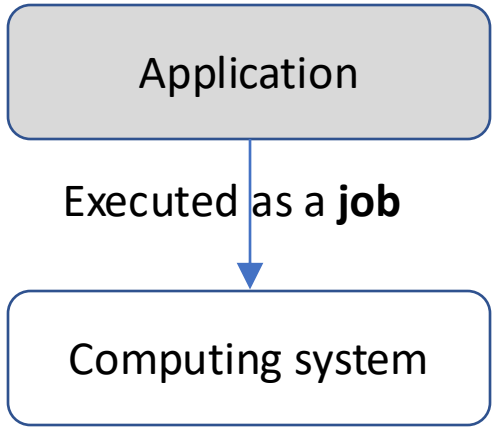
1. Basic terminologies

Terminology: *computing systems*

A Computing system is one or more networked computers (processing units) with necessary support software;

Computing system

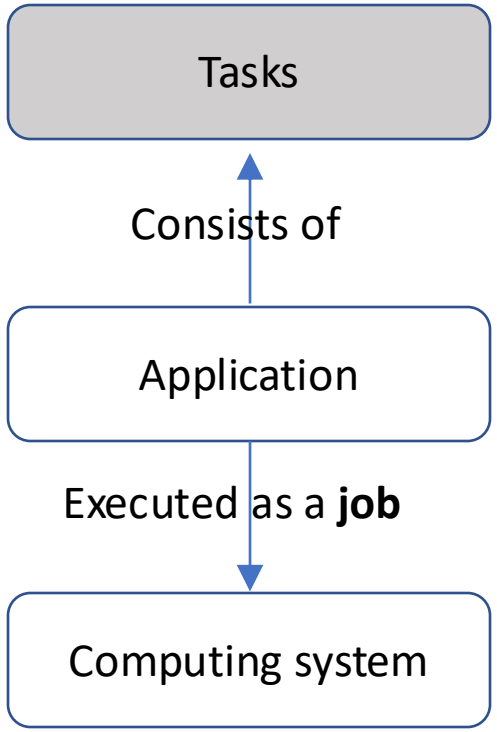
Terminology: *application*



An application is a program executed on one or more computing systems for a specific purpose.

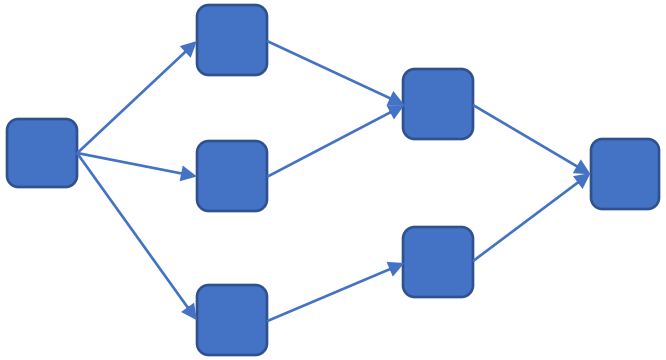
When an application is executed, it becomes a ***job*** on the computing system.

Terminology: *task*

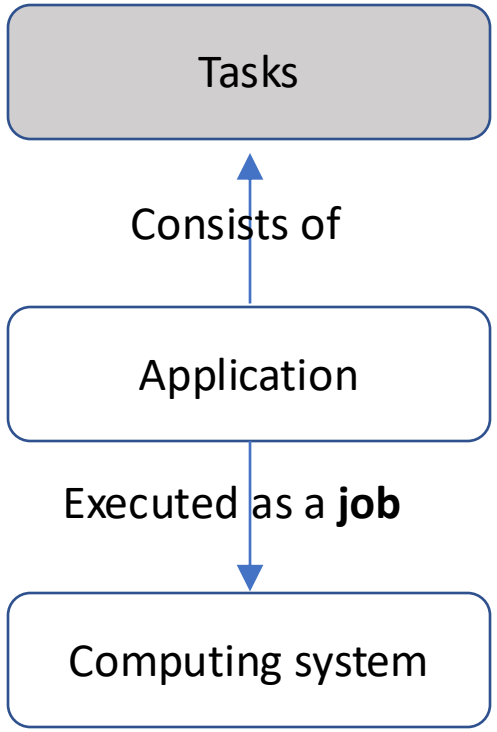


Tasks are work units in an application.

The dependencies among Tasks are often represented as **Directed Acyclic Graph** (DAG)

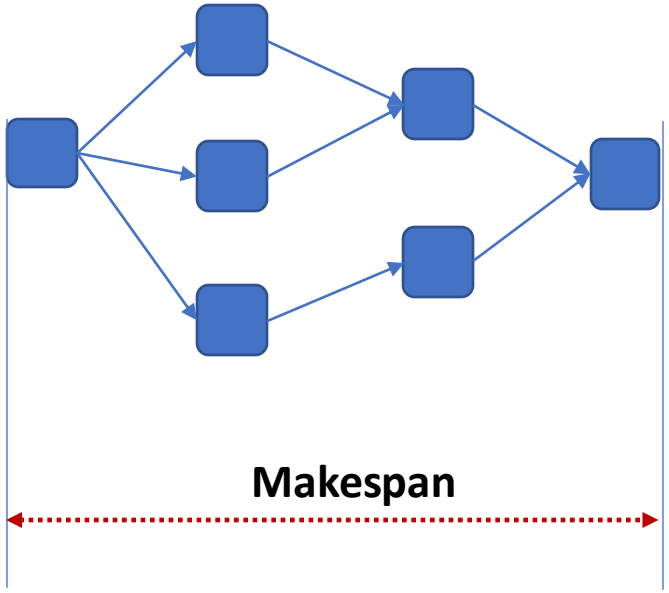


Terminology: *makespan*

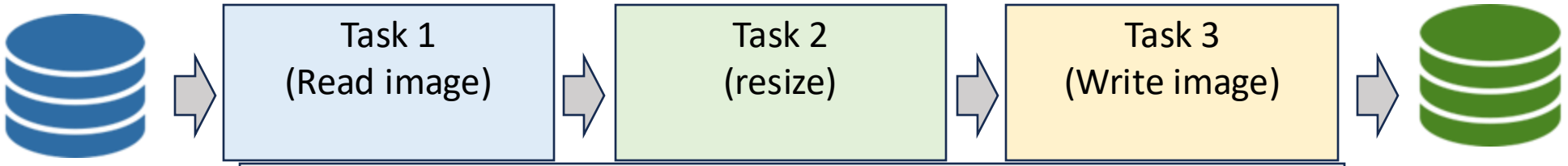


Makespan is *the total time* required to complete a set of jobs or tasks from start to finish.

The dependencies among Tasks are often represented as **Directed Acyclic Graph** (DAG)



Example: sequential program

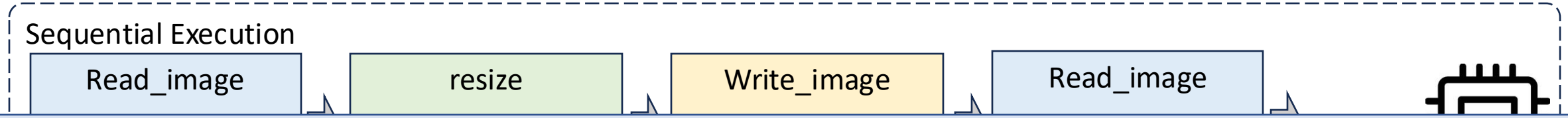



```
while (image set not empty)
 {
     image_s=read_image ();
     image_t=resize (image_s);
     write_image(image_t);
 }
```

Example: sequential program

$$Makespan_{sequential} = \sum_{i=1}^{N} (Time_{read_{(i)}} + Time_{resize_{(i)}} + Time_{write_{(i)}})$$

```
while (image set not empty)
 {
     image_s=read_image ();
     image_t=resize (image_s);
     write_image(image_t);
 }
```



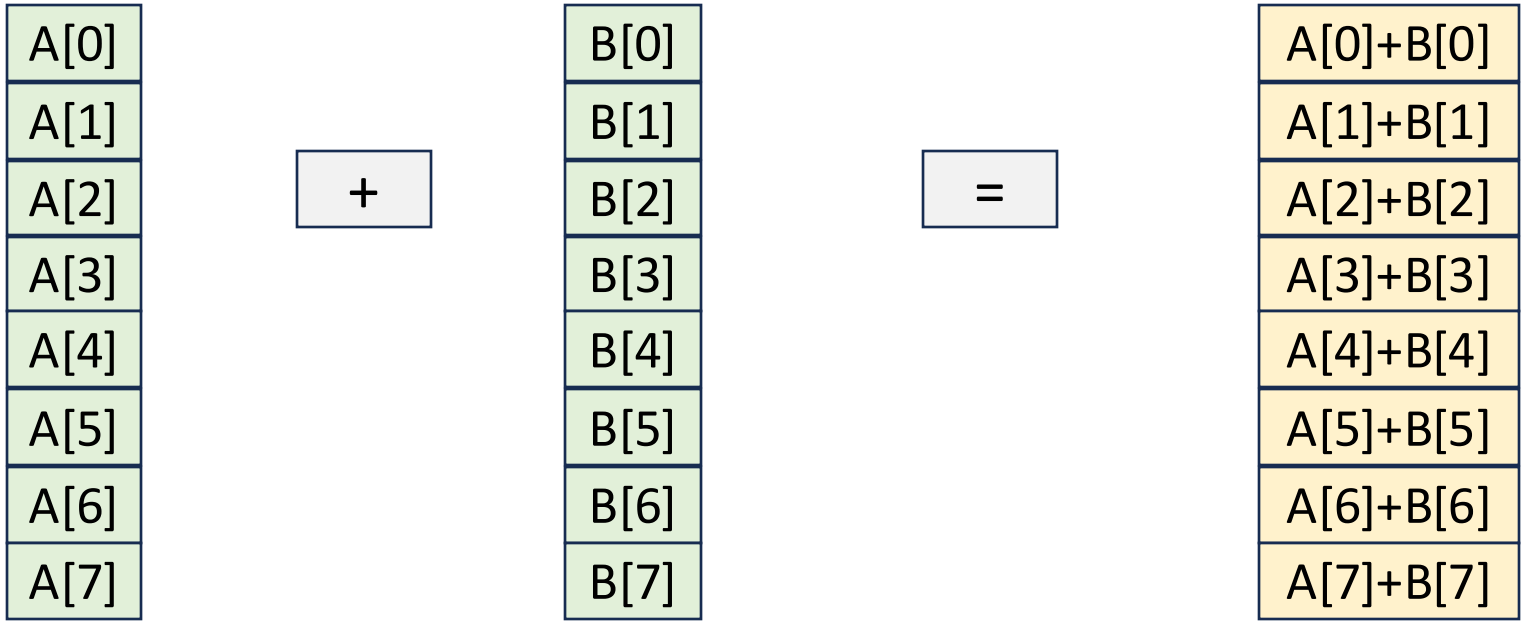
Discussion:
Can we reduce the makespan of an application?

2. From single core to multi and many cores

Make the system more *powerful*

- Increase the number of **clock cycles** per second
- increase the word size of the processor (process more **bits** simultaneously)
- Increase the number of **instructions** being processed simultaneously (e.g., superscale processors)
- Increase the number of **data units** being processed simultaneously
- …

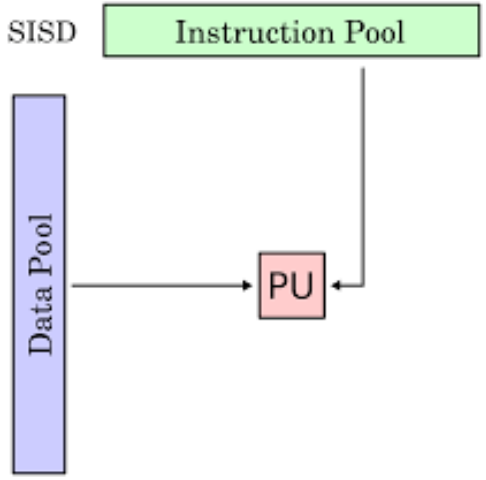
Single instruction single data (SISD)
Single instruction multi-data (SIMD)

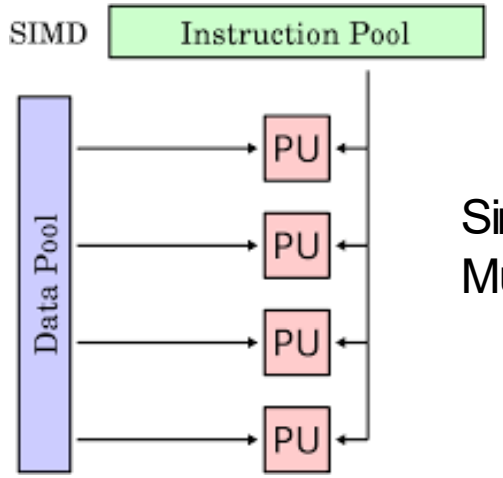


1. In **SISD**, the instruction will be executed eight times, with **each time one data unit**;
2. In **SIMD**, the instruction will be executed once with **eight data units together**.

First taxonomy: Michael Flynn (1966)

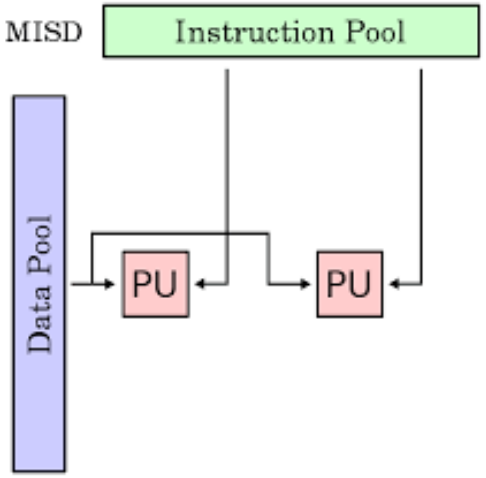
Single Instruction
Single Data

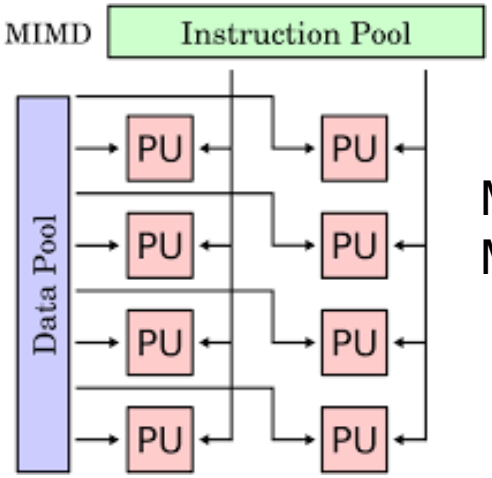




Single Instruction
Multiple Data

Multiple Instructions
Single Data

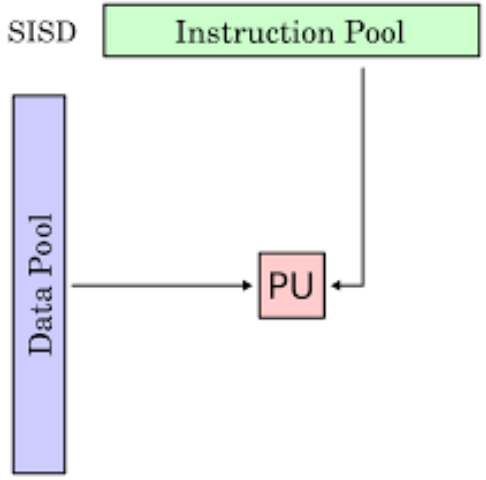


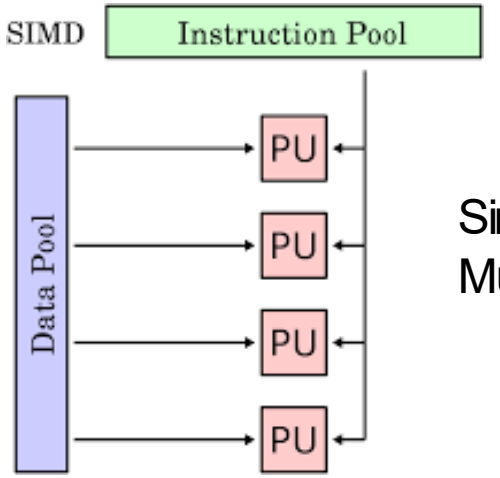


Multiple Instructions
Multiple Data

Flynn's taxonomy

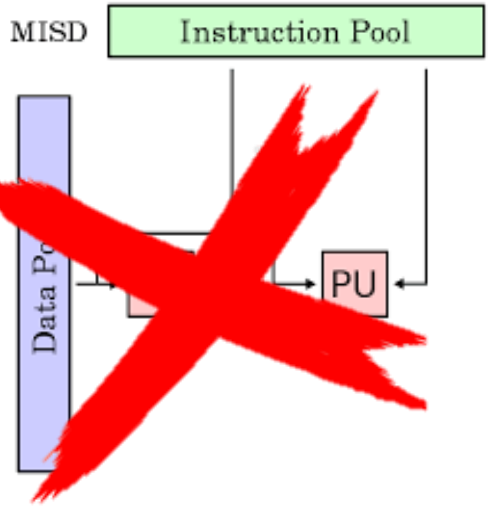
Single Instruction
Single Data

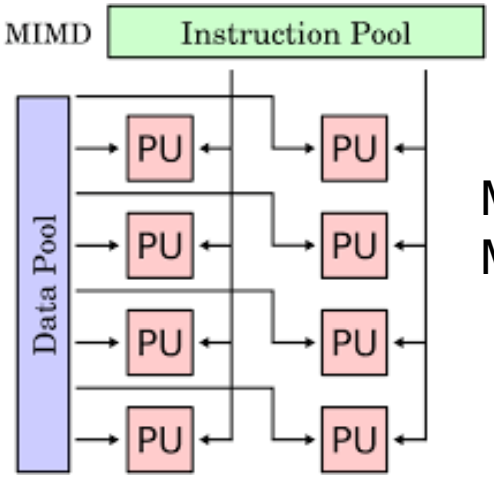




Single Instruction
Multiple Data

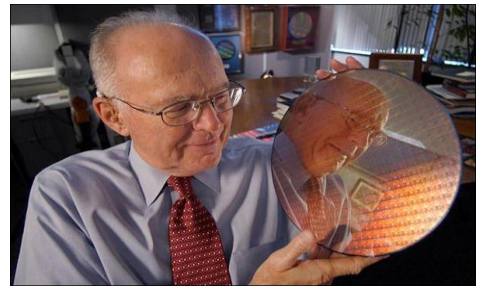
Multiple Instructions
Single Data





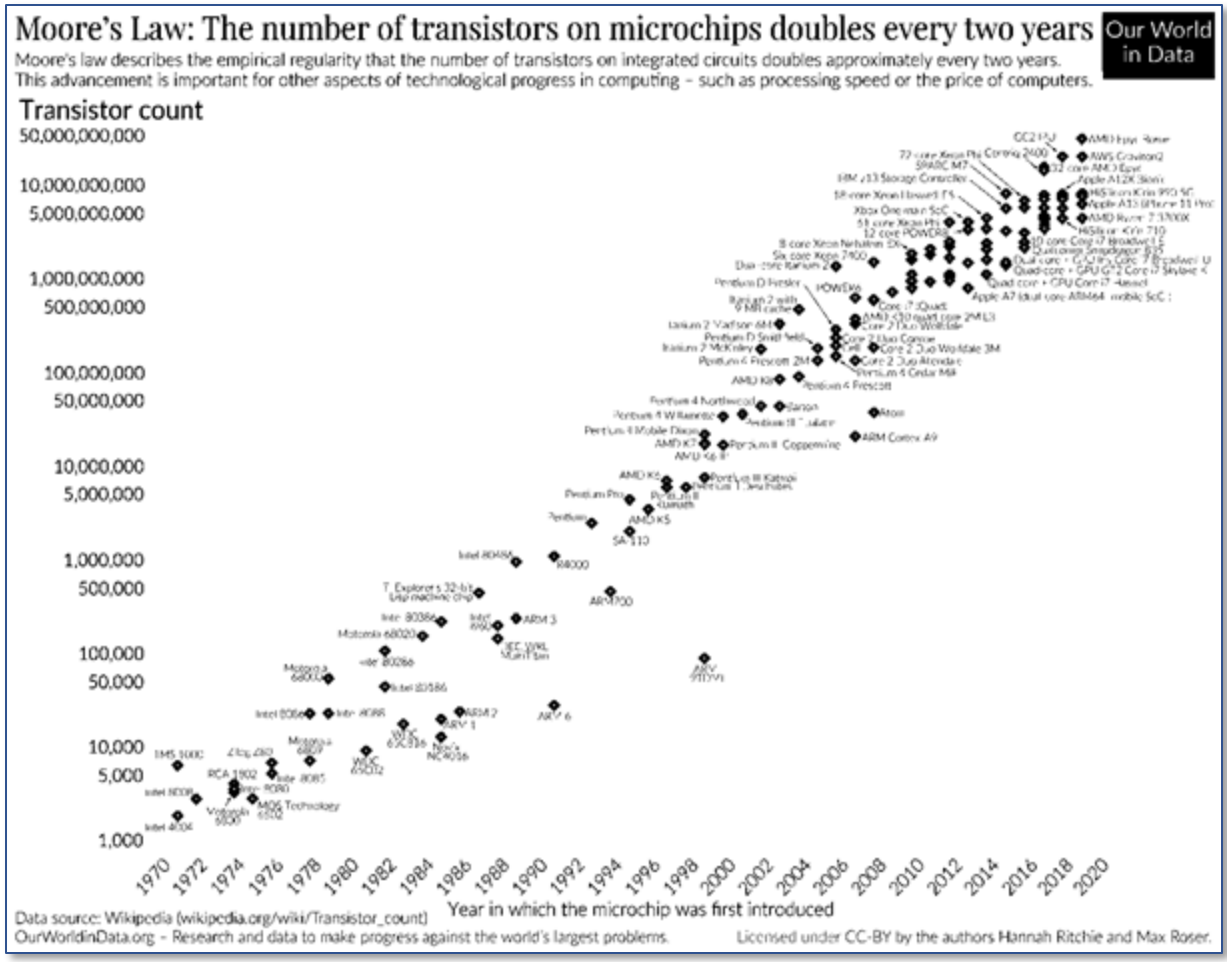
Multiple Instructions
Multiple Data

Moore's Law



Gordon Moore (co-founder of Intel) predicted in 1965 that the **transistor density of semiconductor chips would double roughly every 24 months.**

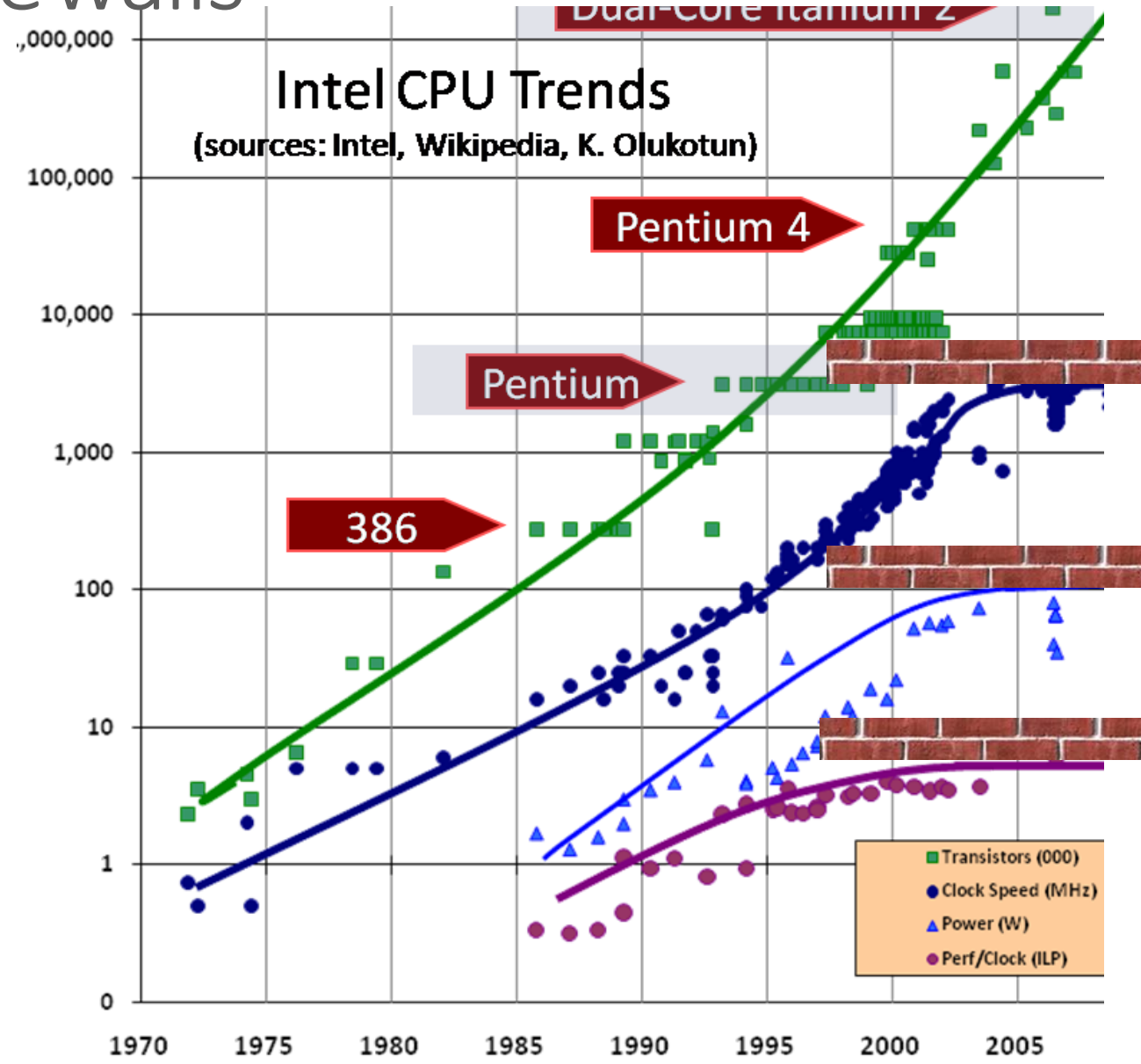
- More **transistors/gates** in a chip
- Higher clock **frequency**
- More advanced **design**



https://ourworldindata.org/uploads/2020/11/Transistor-Count-over-time.png

Around 2005: hitting the walls

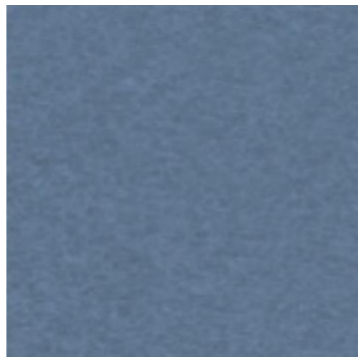
- Power wall
- Instruction level parallelism wall
- Design complexity wall



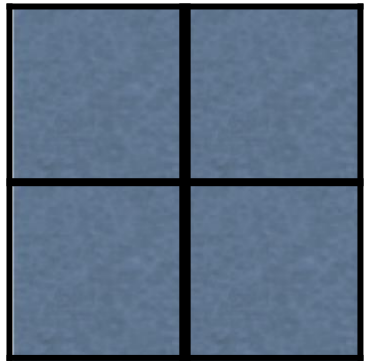
The shift to multi-core

$$\text{Power}_{\text{density}} \sim \frac{C \cdot V^2 \cdot f}{A}$$

Performance 1
Power 1

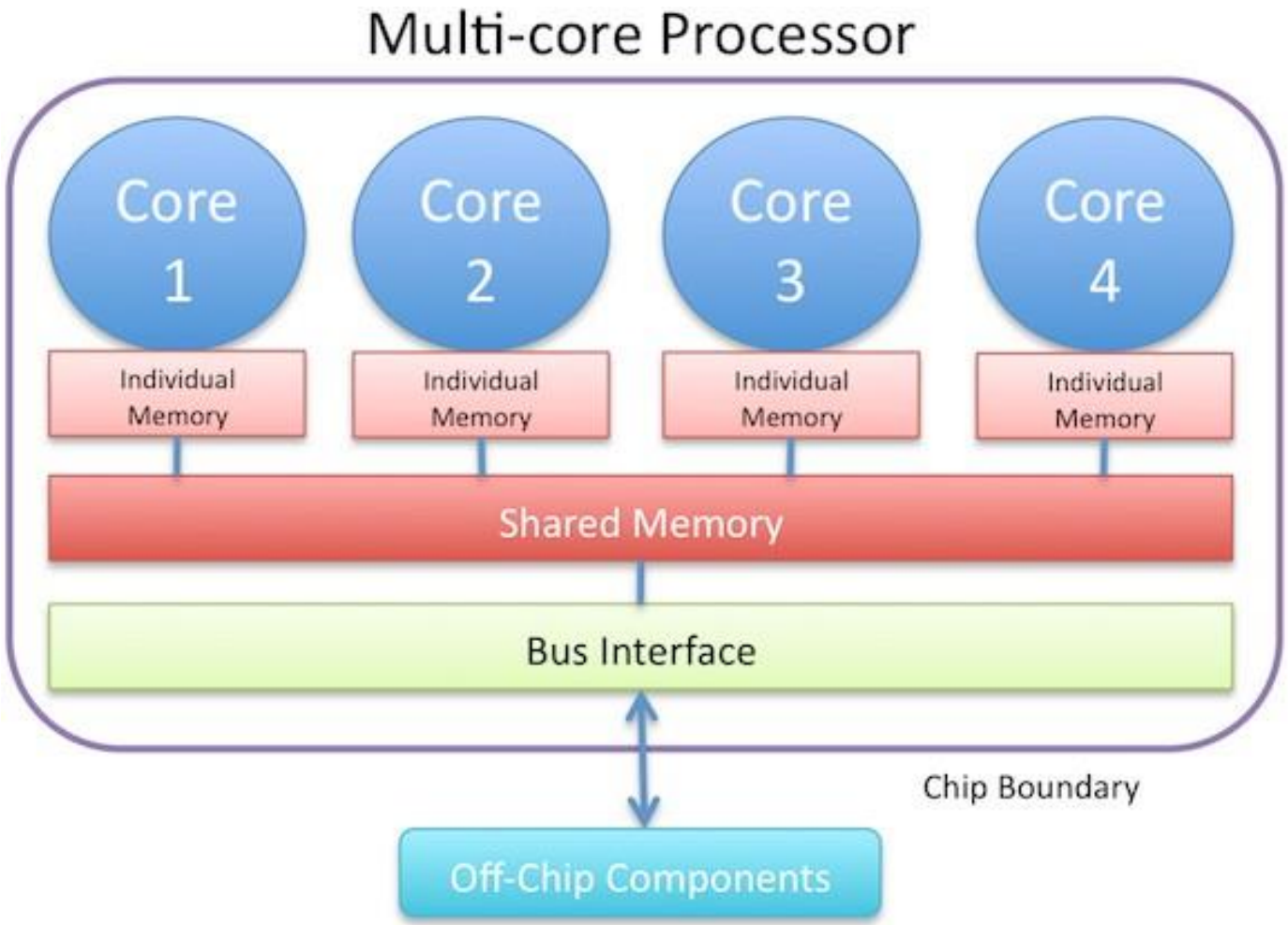


Performance: X **2** (2*F)
Power: X **4**

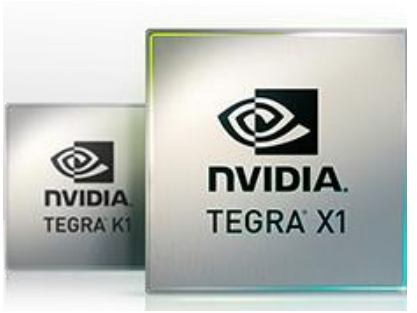


F remains same
Performance = ½ * 4 = 2
Power = ¼ * 4 =1

Generic multi-core CPU

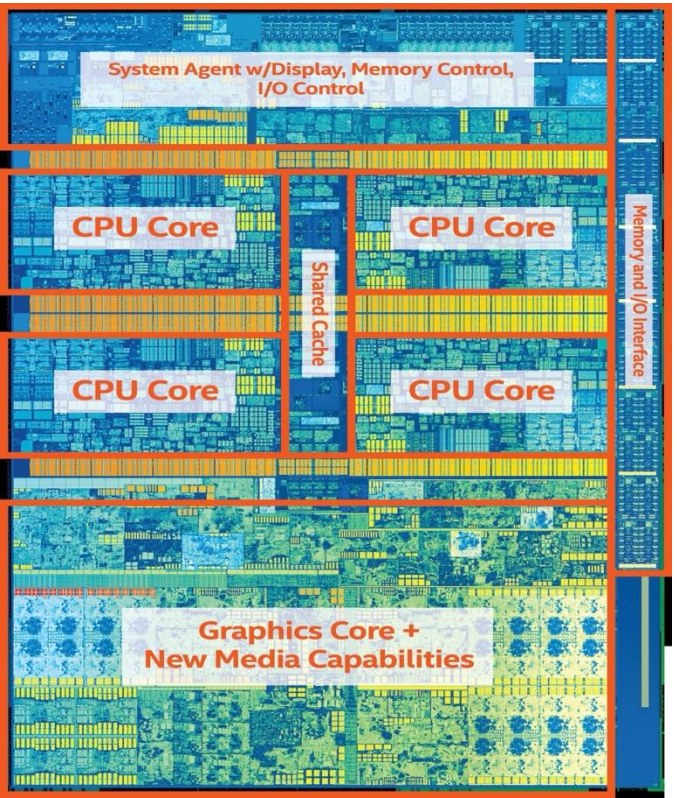


Has been widely used

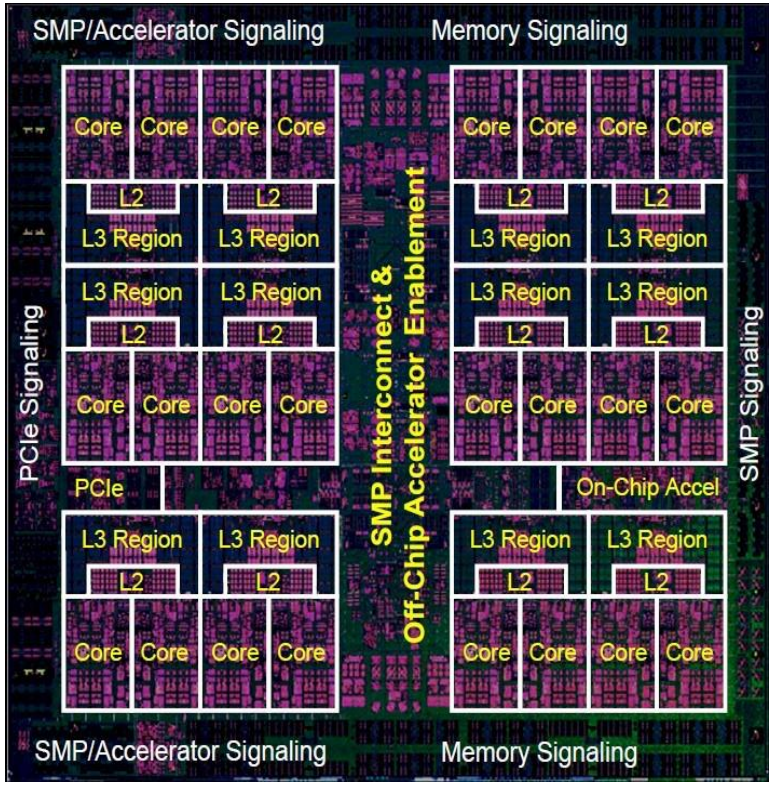


Nvidia Tegra:
Quad-core CPU, 256-core GPU

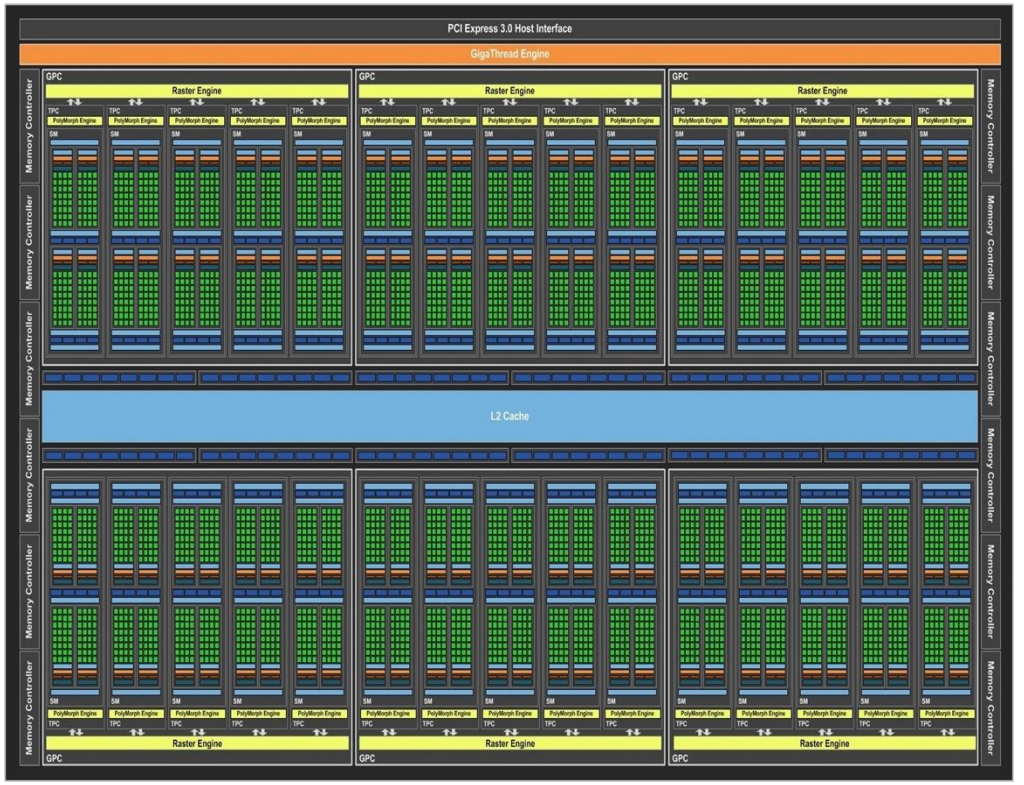
Intel Kaby Lake(2017)



IBM Power9(2016)

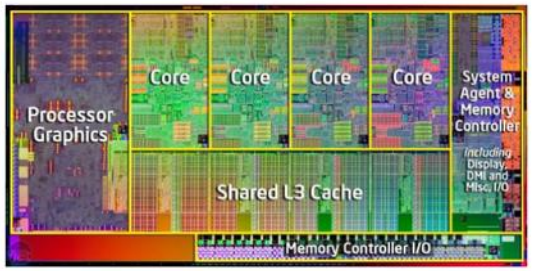


Nvidia Titan Xp(2017)



Changes in computing systems

- From single core to multi-core and many cores
- From shared memory to distributed memory
- From sequential execution to concurrent, parallel and distributed executions



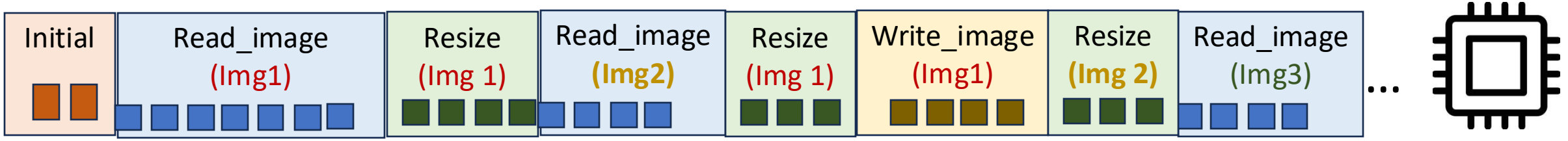
Quad-core processors such as this Intel Sandy Bridge introduce programming complexities not present in a single-core CPU.

3. From sequential to concurrency

How to reduce the makespan of sequential tasks?

- The **Resize** task requires more CPU clocks than the other two tasks.
- The system often wastes time waiting for I/O tasks.

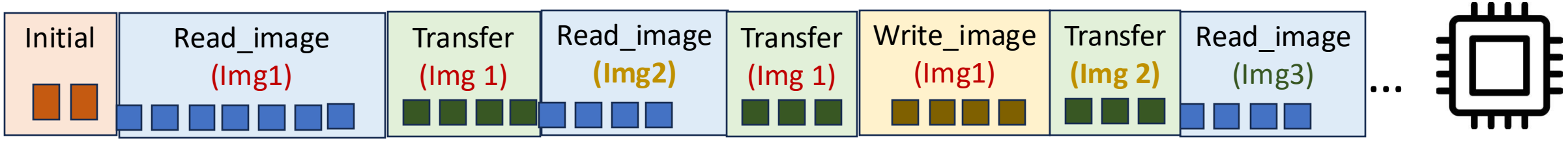
Concurrent execution



Concurrency

- **Concurrent execution**: multiple, possibly interacting tasks are executed potentially at the same time.
- **Concurrency:** multiple, possibly interacting tasks that may potentially be executed at the same time. (tasks can be executed with overlapping time.)
- **Concurrent applications:** applications that make use of concurrency.

Concurrent execution



Concurrency

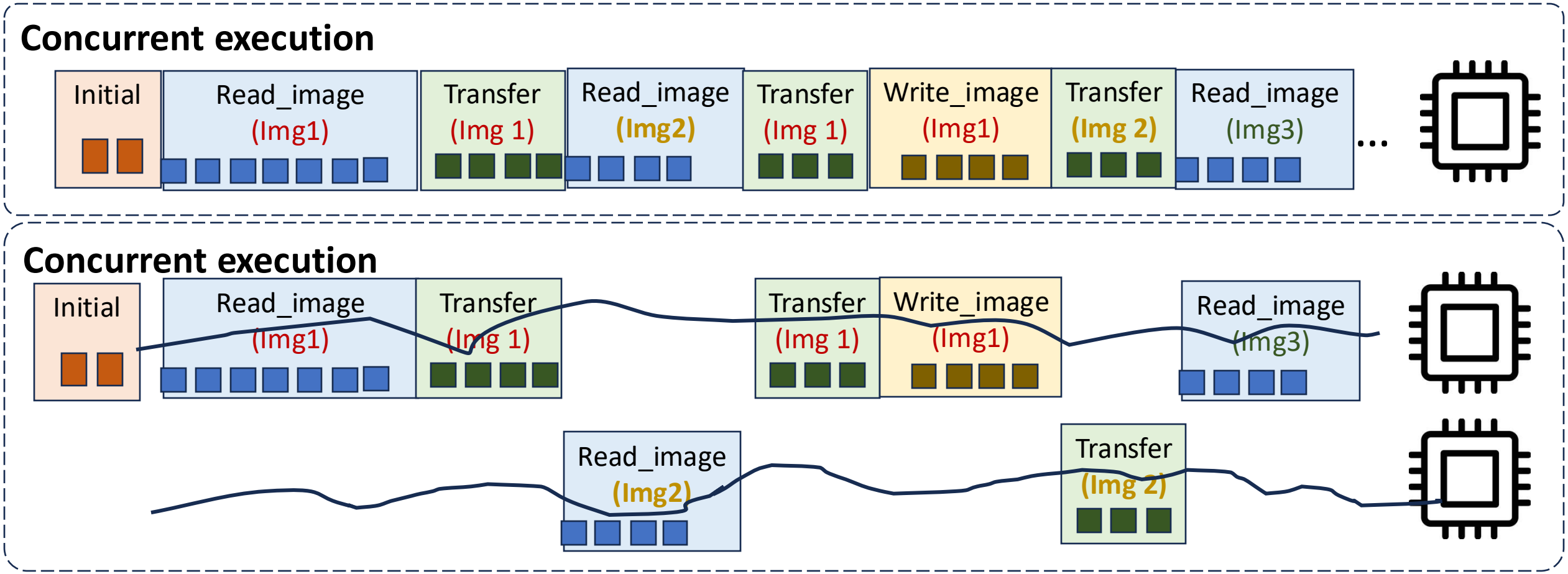
- Application **tasks** must be designed to support **concurrency**.
- Concurrent **tasks** can run on a uniprocessor machine (**single core**) if the Operating System (OS) supports it.
- Concurrent tasks can potentially execute **at the same time**, but they do **not** have to run **simultaneously**.
- The **execution order** of concurrent tasks is often **nondeterministic**.

The makespan of concurrent tasks is often not deterministic.
It Depends on the **available processors**, the **number** of concurrent tasks, the **characteristics** of the tasks and their **execution order.**

Discussion:

1. What is the makespan of conurrent tasks? Can we reduce it?

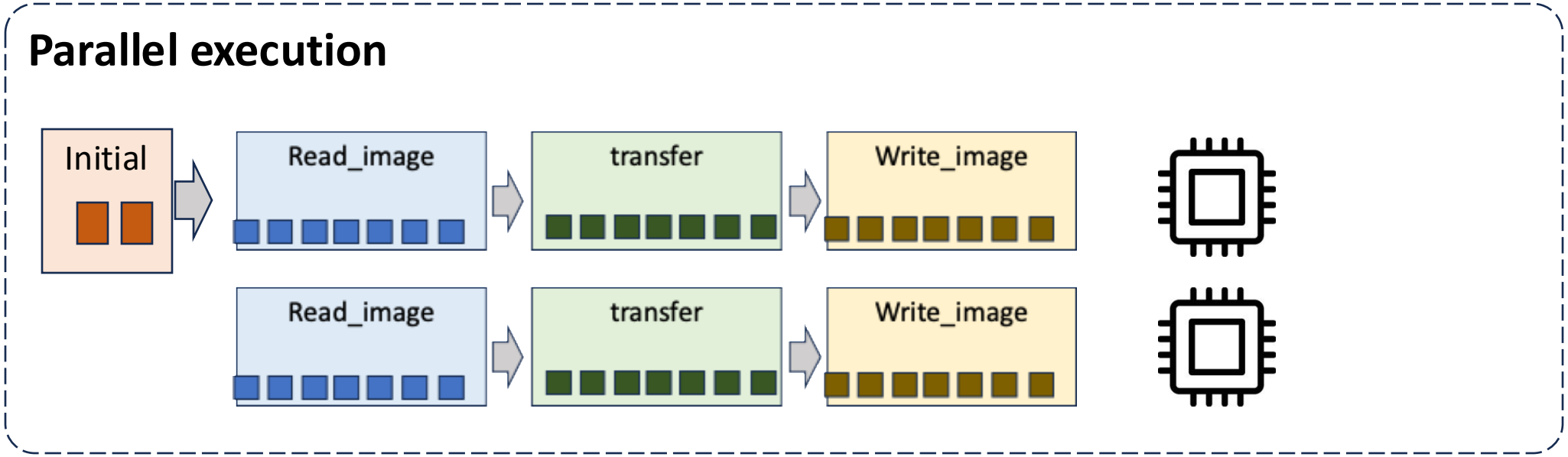
Concurrency cont.



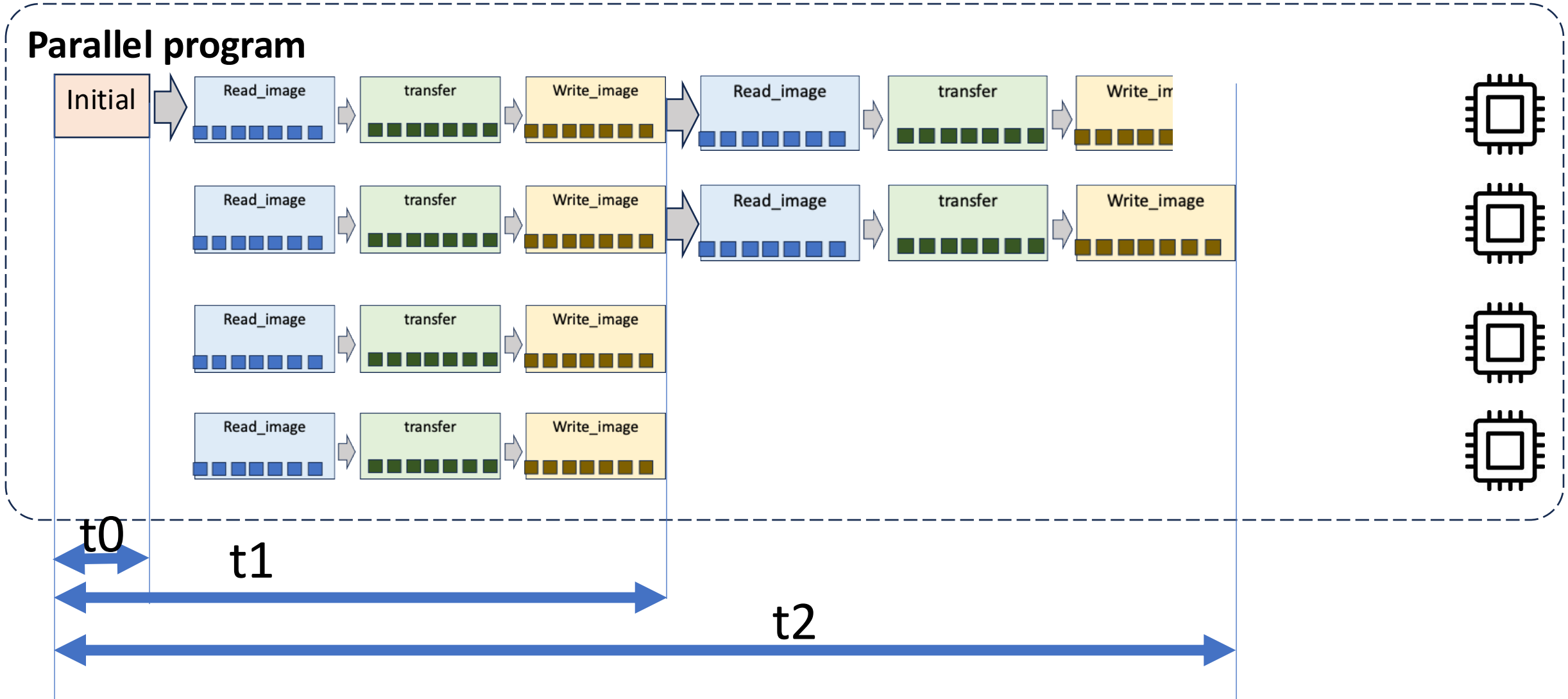
The **number of threads (processes)** can be more than the number of **available cores/processors**.

Parallelism

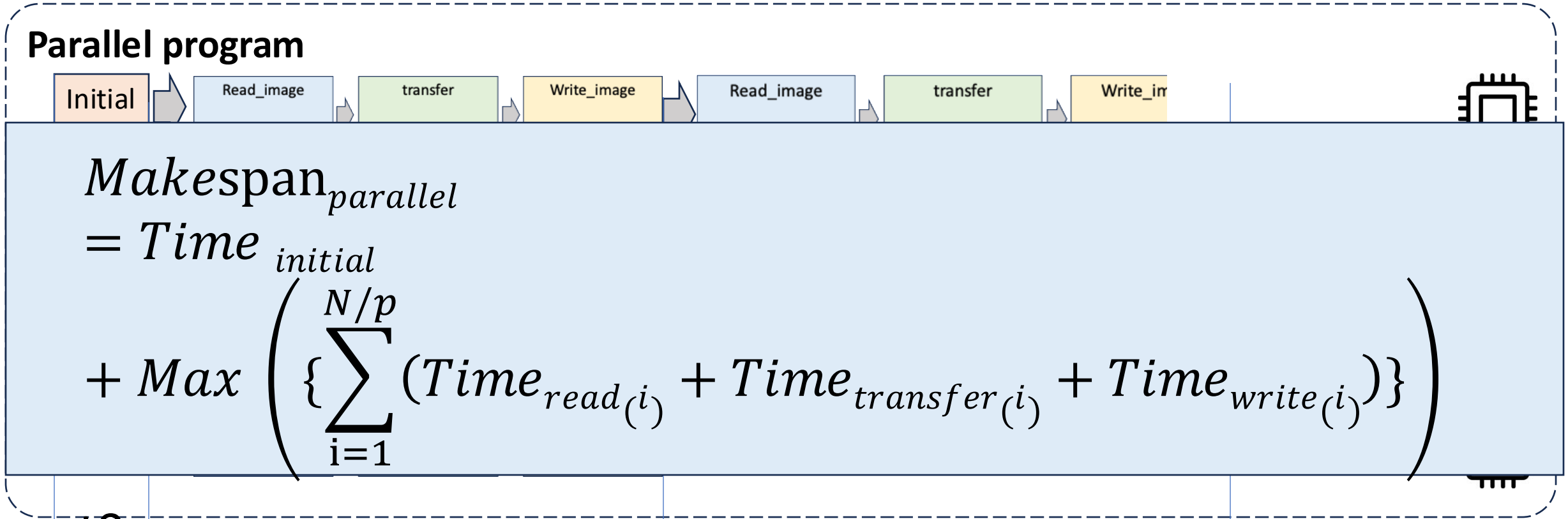
- **Parallelism**: execution of concurrent tasks on computing systems capable of executing more than one task **simultaneously**.
- **Parallel applications:** applications that make use of **parallelism**.



Makespan of parallel execution



Makespan of parallel execution



What is the difference between concurrency and parallelism?

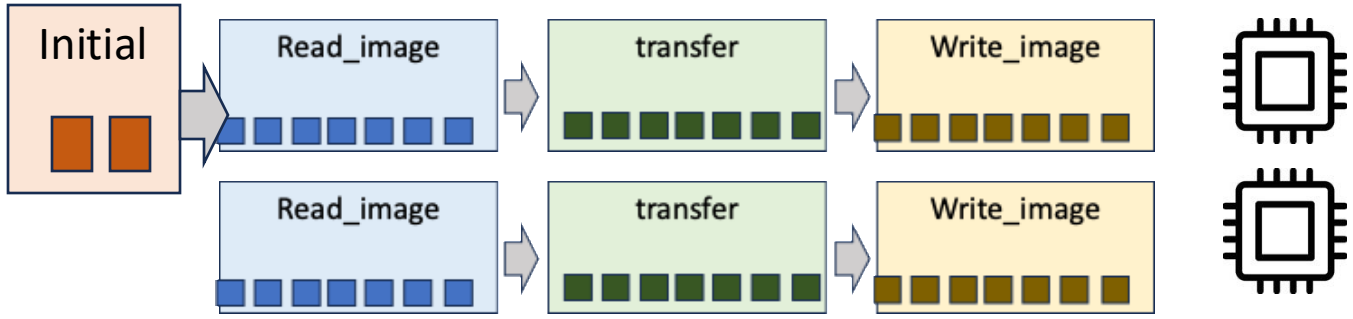
What is the difference between concurrency and parallelism?

- **Concurrency:** multiple, possibly interacting tasks that may potentially be executed **at the same time**.
- Do **not** have to run on multiple processors (cores)
- Might be better than sequential execution

- **Parallelism**: execution of **concurrent** tasks on computing systems capable of executing more than one task **simultaneously**.
- Require multiple processors (cores)
- Better performance than sequential and concurrent execution

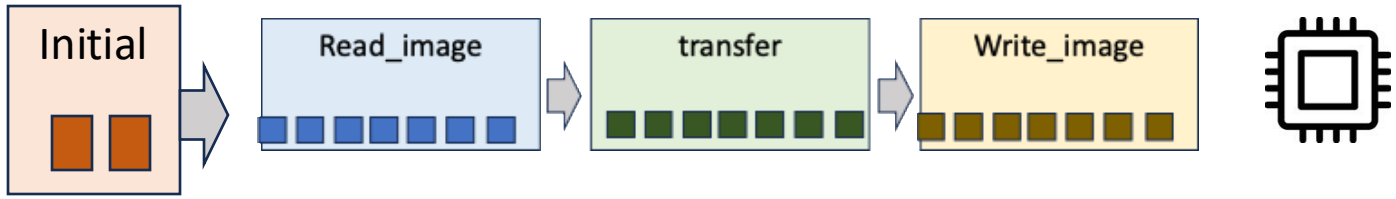
Parallel and distributed execution

Parallel tasks and execution



Can be in the **same machine**, or different machines which are connected with **high speed network**.

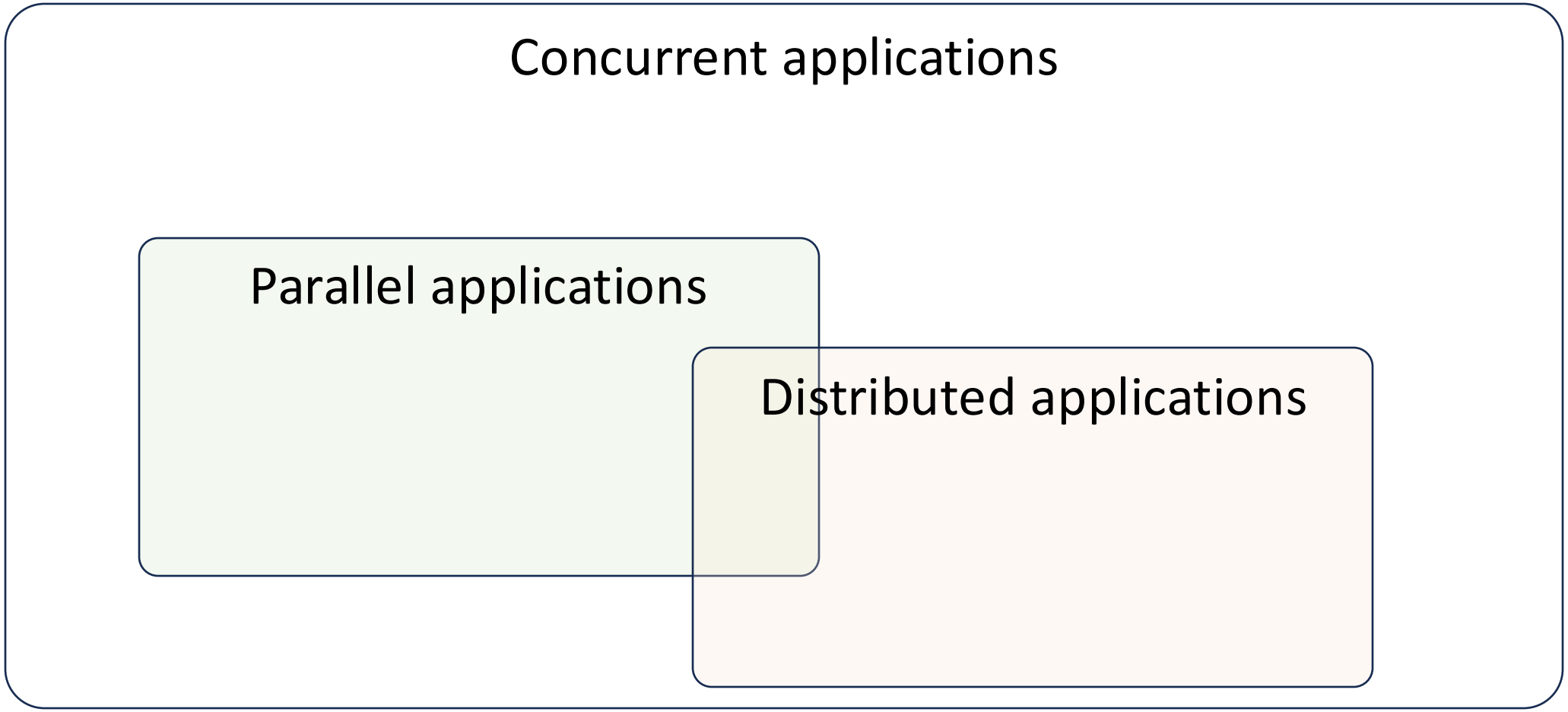
Distributed tasks and execution



Computer 1

Computer 2

Concepts


Concurrent applications
Parallel applications
Distributed applications

A short summary

	Sequential	Concurrent	Parallel	Distributed
Tasks				
Application				
Computing system				

Makespan of the application in different paradigms

	Sequential	Concurrent	Parallel	Distributed
Task				
Application	**Ms=Makespan (sequential)**	**Mc=Makespan (concurrent)**	**Mp=Makespan (Parallel)**	**Md=Makespan (Distribute)**
Computing system				

$$Mp < Ms,\ Mp <= Md,\ Mp <= Mc$$

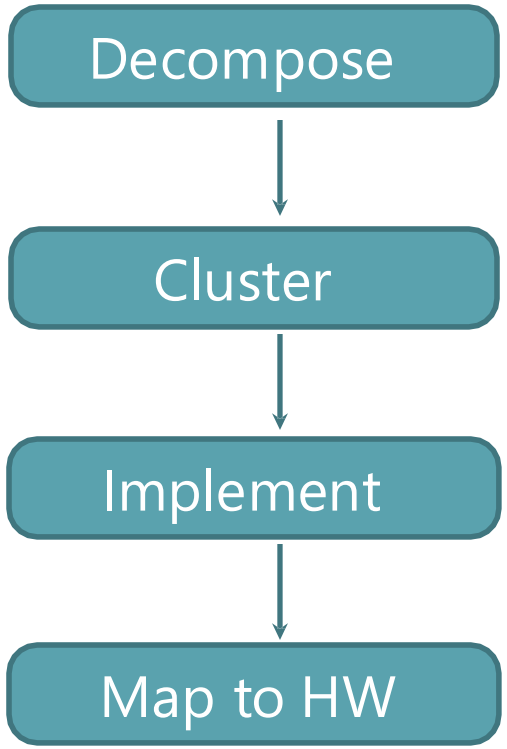
Discussion:

1. How can we compare those makespans if for the same amount of work?

4. Programming model

A Method for Parallel Application Design*

- Decompose (partition)
  - What is the computation? Which data?
- Cluster (communicate & agglomerate)
  - What granularity?
- Implement in the programming model
  - Which model? (Multithreading, MPI, CUDA etc.)
  - Implement tasks as processes or threads
  - Add communication and synchronization constructs
- Map units of work to processors
  - Might be transparent.



*inspired by
”Designing and Building Parallel Programs”, by Ian Foster
http://www.mcs.anl.gov/dbpp

Data parallization

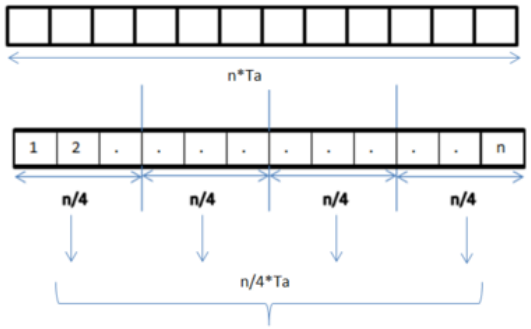
Often used when

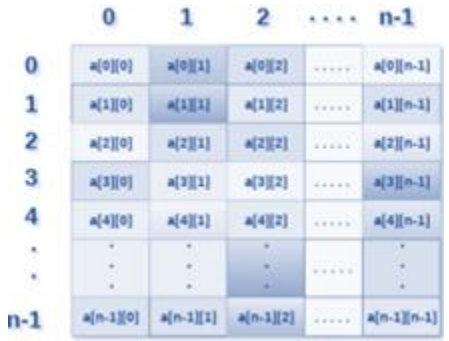
- The application has to handle **a large data input**
- Each unit should be processed consistently using the same program

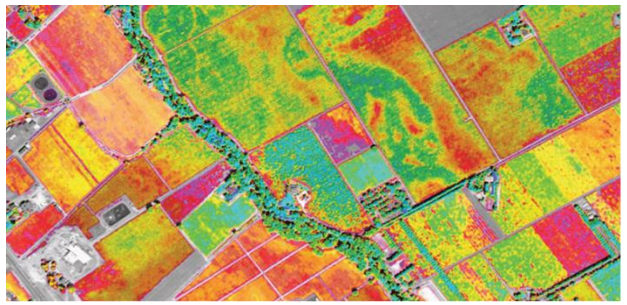
The basic idea

- Partition the data into **multiple chunks**
- Those data chunks will be processed **in parallel**
- Also called **Single Program Multiple Data (SPMD)**

Examples







Task parallization

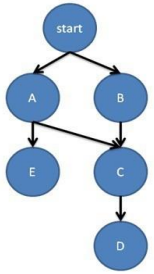
Often used when

- The application has to handle **a large number of tasks**
- Those tasks have **directed acyclic dependencies**

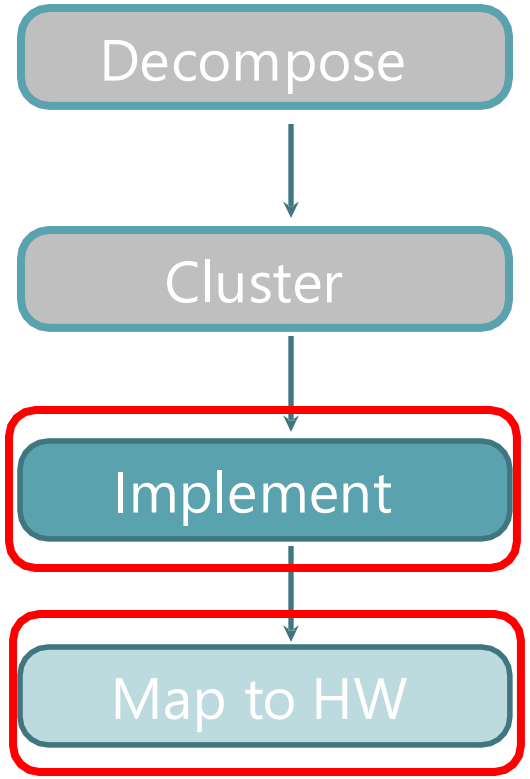
The basic idea

- Identify the dependencies
- Run those *independent tasks* in **concurrent** or **in parallel**

Examples: project management, interactive GUI,

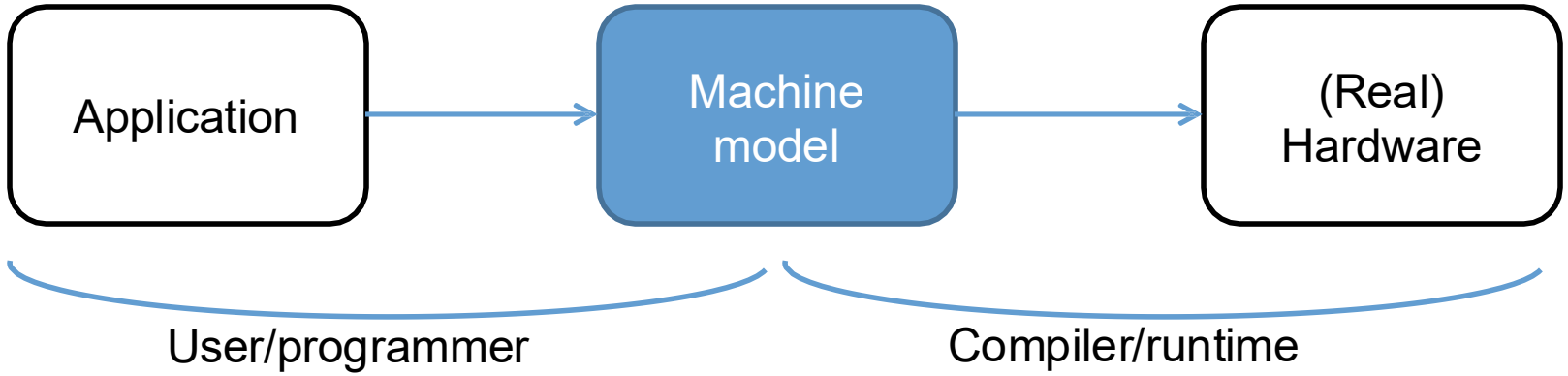


Programming models



Programming model* = machine abstraction + set of rules to use the machine

- User programs machine model
- Compiler/run-time provides the execution



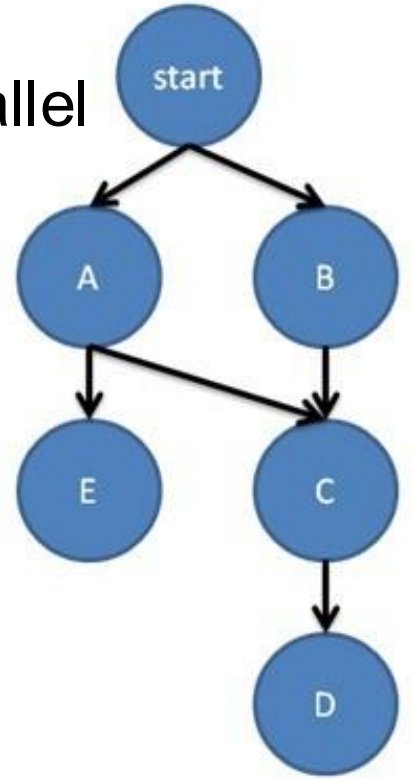
*simplified definition

5. Task parallelization and the degree of concurrency

Terminology

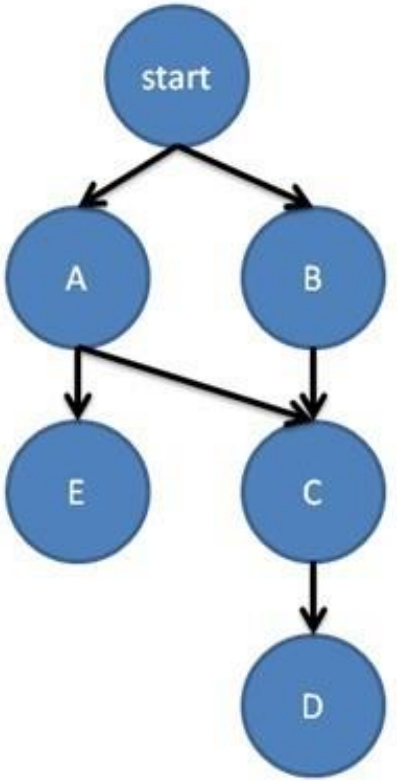
- **Task**
  - A logically discrete section of computational work.
  - A parallel application consists of multiple tasks running in parallel

- Application = **DAG of tasks/operations**
  - Operations = nodes
  - Dependencies = edges
    - Dependency = child-after-parent



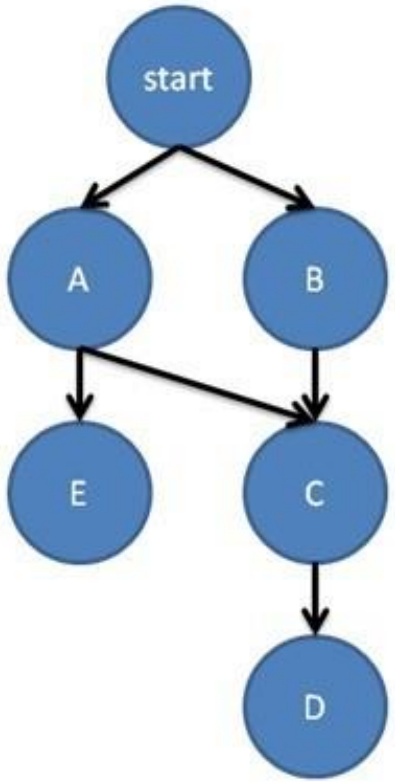
Terminology

- **Computation**
  - The actual **work** to be executed by the tasks
- **Communication**
  - Tasks must **exchange** information
  - Different mechanisms depend on the machine model
    - Shared variables
    - Message Passing
- **Synchronisation**
  - A task execution **depends** on another task's progress
  - Different mechanisms
    - Barriers
    - Locks, mutexes, semaphores
    - Message Passing



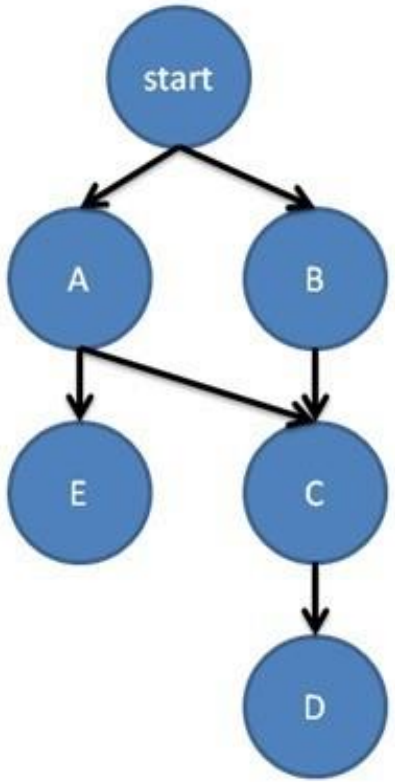
Terminology

- **Two metrics**:
  - The **amount of work** to be executed
  - The **makespan** of the application
  - **Minimal** makespan: ???
  - **Maximal** makespan: ???



Terminology

- **Two metrics**:
  - The **amount of work** to be executed
  - The **makespan** of the application
  - **Minimal** makespan: the **critical-path length** is of a DAG
  - **Maximal** makespan: the **sequential execution** of all tasks



Terminology

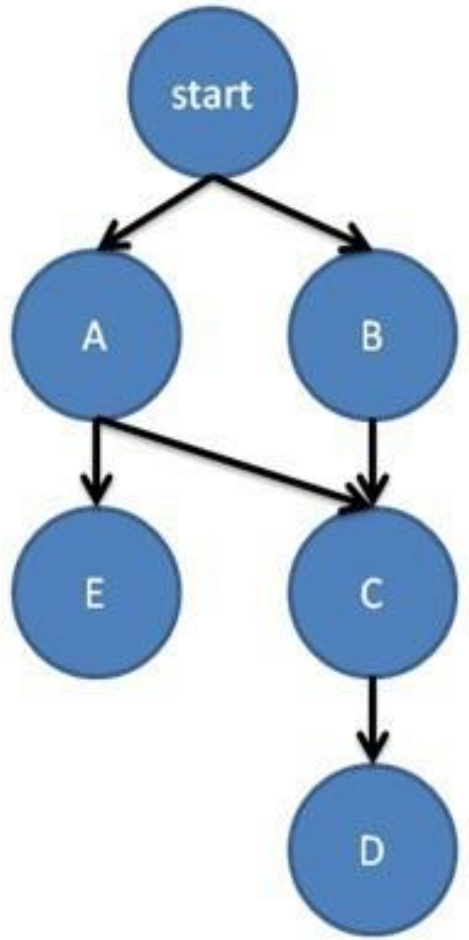
- **Maximum degree of concurrency**
  - The maximum number of tasks that can be potentially executed simultaneously

- **Average degree of concurrency**
  - **T**he average number of tasks that can be executed in parallel

$$Average\ degree\ of\ concurrency = \frac{Work}{Critical\ path\ length}$$

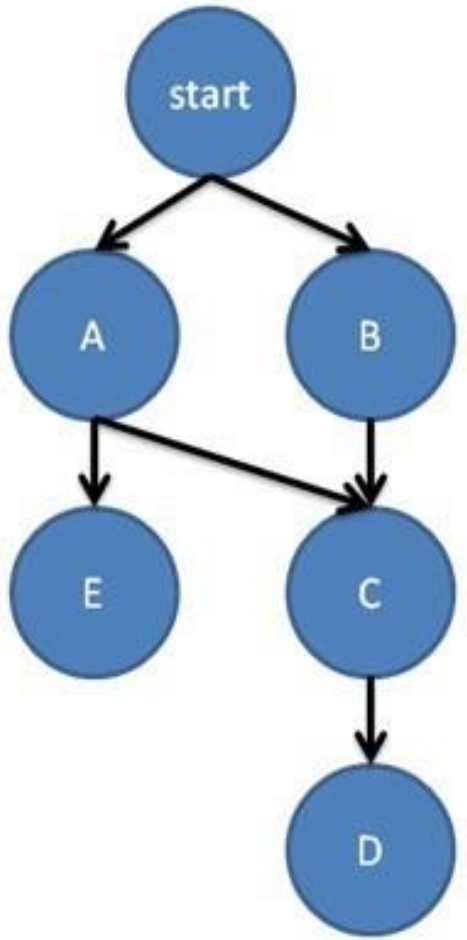
Example

- Amount of Work=6
- Critical path=4
- Maximum degree concurrency=
- The average degree of concurrency=



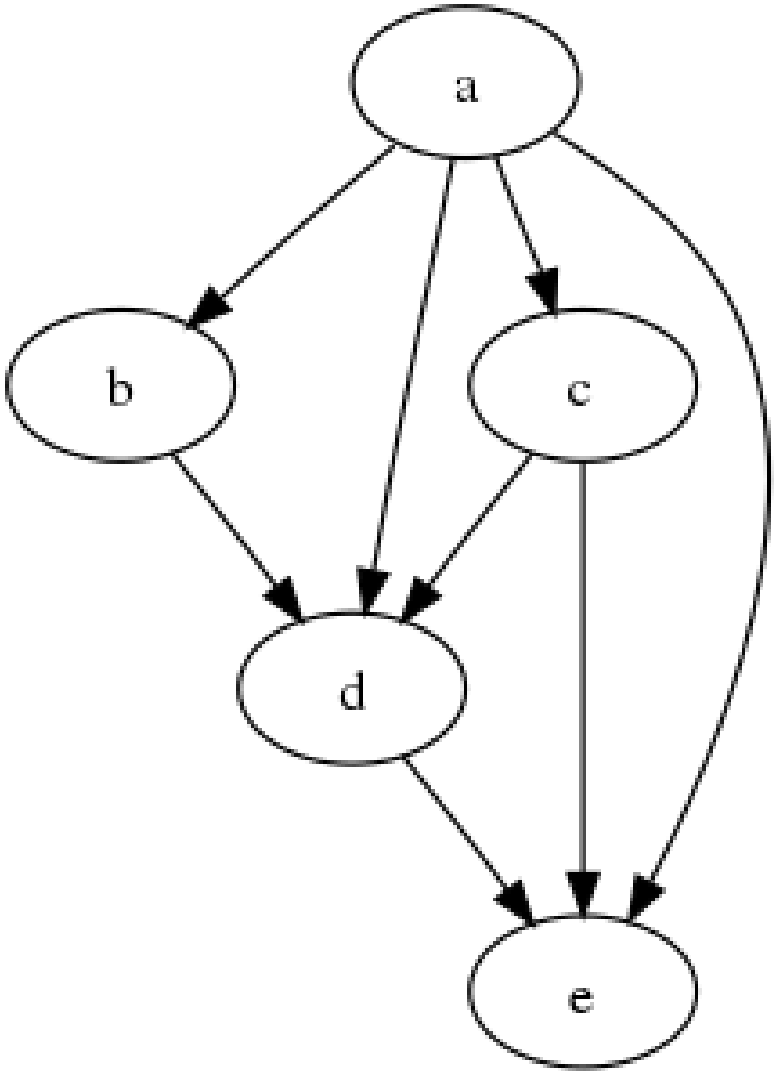
Example

- Amount of Work=6
- Critical path=4
- Maximum degree concurrency=2
- The average degree of concurrency=1.5



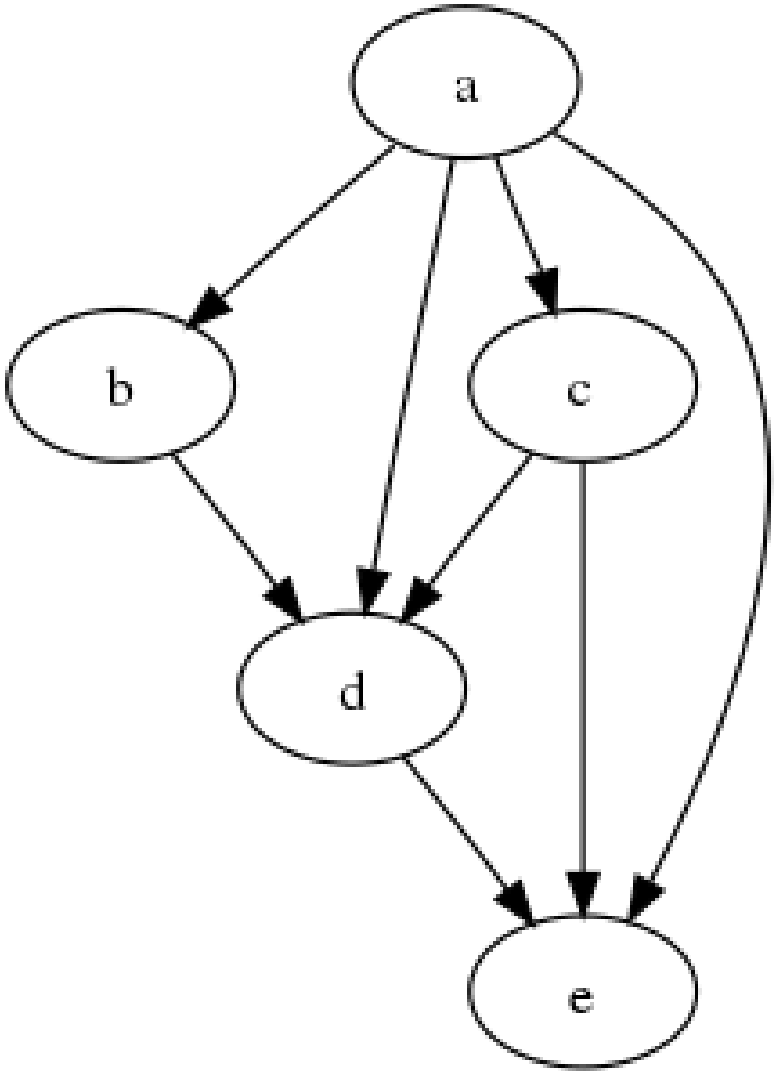
Example

- Amount of Work=
- Critical path=
- Maximum degree concurrency=
- The average degree of concurrency=



Example

- Amount of Work=5
- Critical path=4
- Maximum degree concurrency=2
- The average degree of concurrency=1.25



Terminology

- Task **scheduling** = the order in which the threads are executed on the machine
  - **User-based**: programmer decides
  - **OS-based**: OS decides (e.g., Linux, Windows)
  - **Hardware-based**: hardware decides (e.g., GPUs)

Terminology

- **Granularity** = trade-off between the size of the task (including communication tasks) and the number of tasks
  - **Granularity of work**
    - Fine: little work per task
    - Coarse: a lot of work per task
  - **Granularity of interaction**
    - Fine: very frequent communication
    - Coarse: infrequent communication
- **Locality** = keep the data close to the computation
  - **Relevant** when processing in multiple tasks/processors
  - **Ideal**: keep data local to the task
    - Implies less communication => less dependences => less performance penalty

Summary