

Parallel Processing in Python

DS 5110: Big Data Systems (Spring 2023)

Lecture 5

Yue Cheng



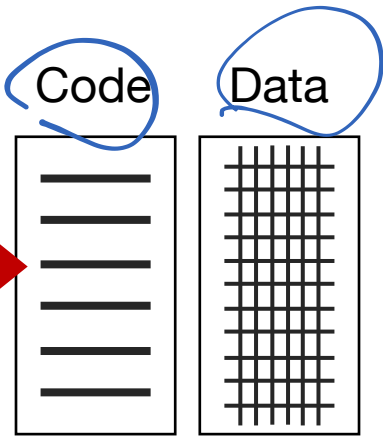
Some material taken/derived from:

- Wisconsin CS301 by Tyler Harter and UC San Diego DSC102 by Arun Kumar.

@ 2023 released for use under a [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.

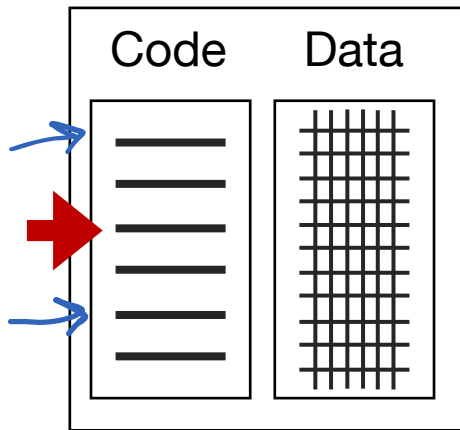
Parallelism: Doing multiple things at once

- Mental models
- Two problems
- Parallelism
 - Thread
 - Process
 - Task

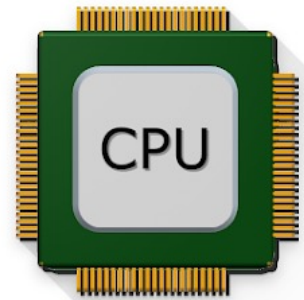
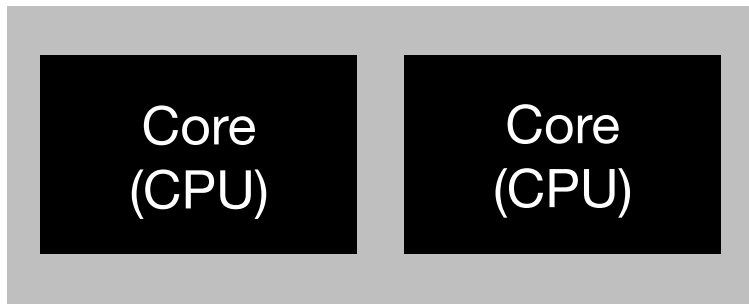
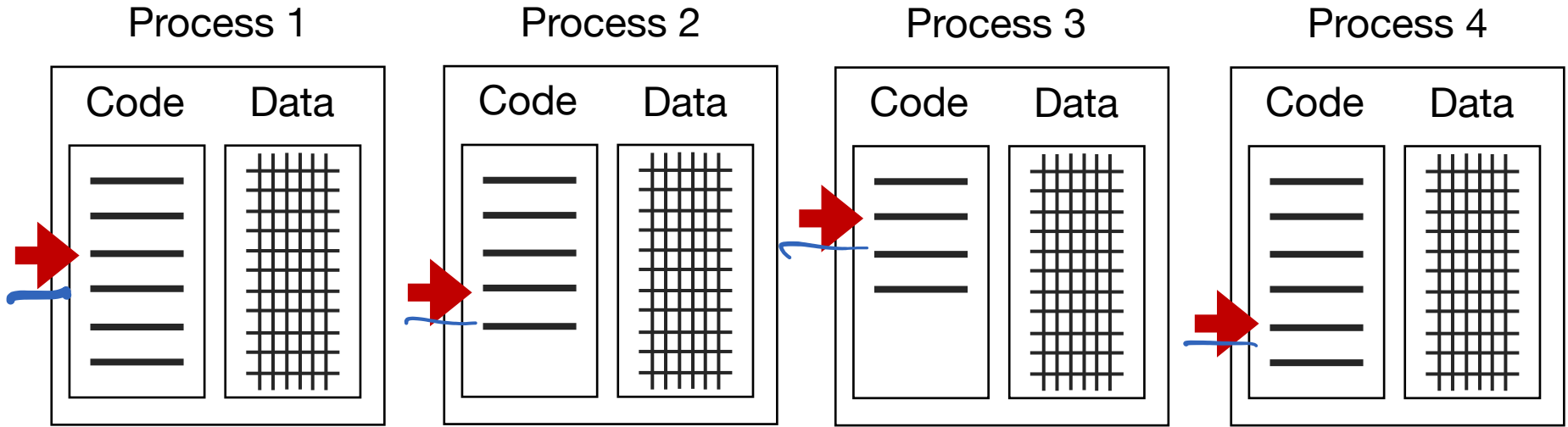


Instruction pointer
(also called “program counter”)

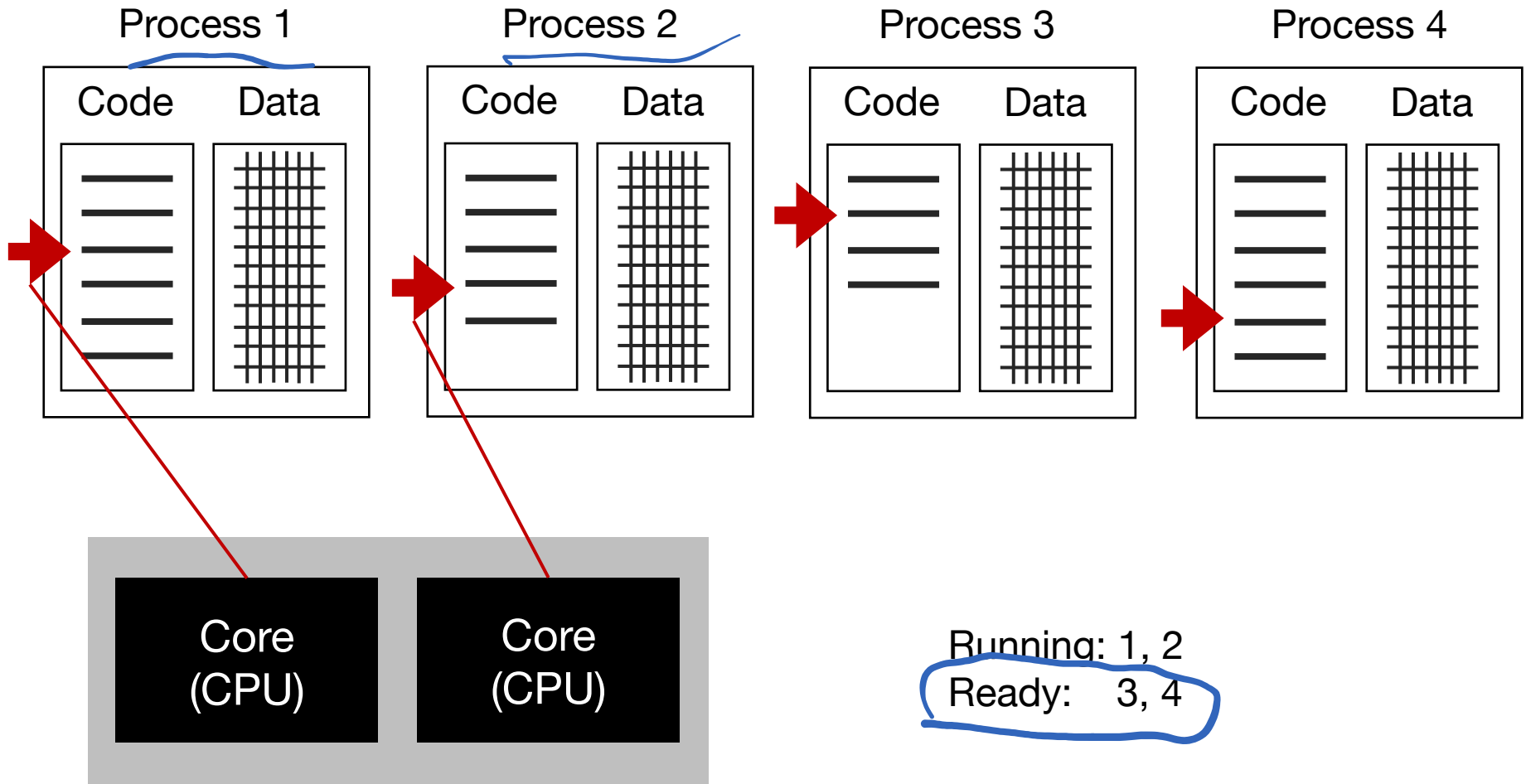
Process



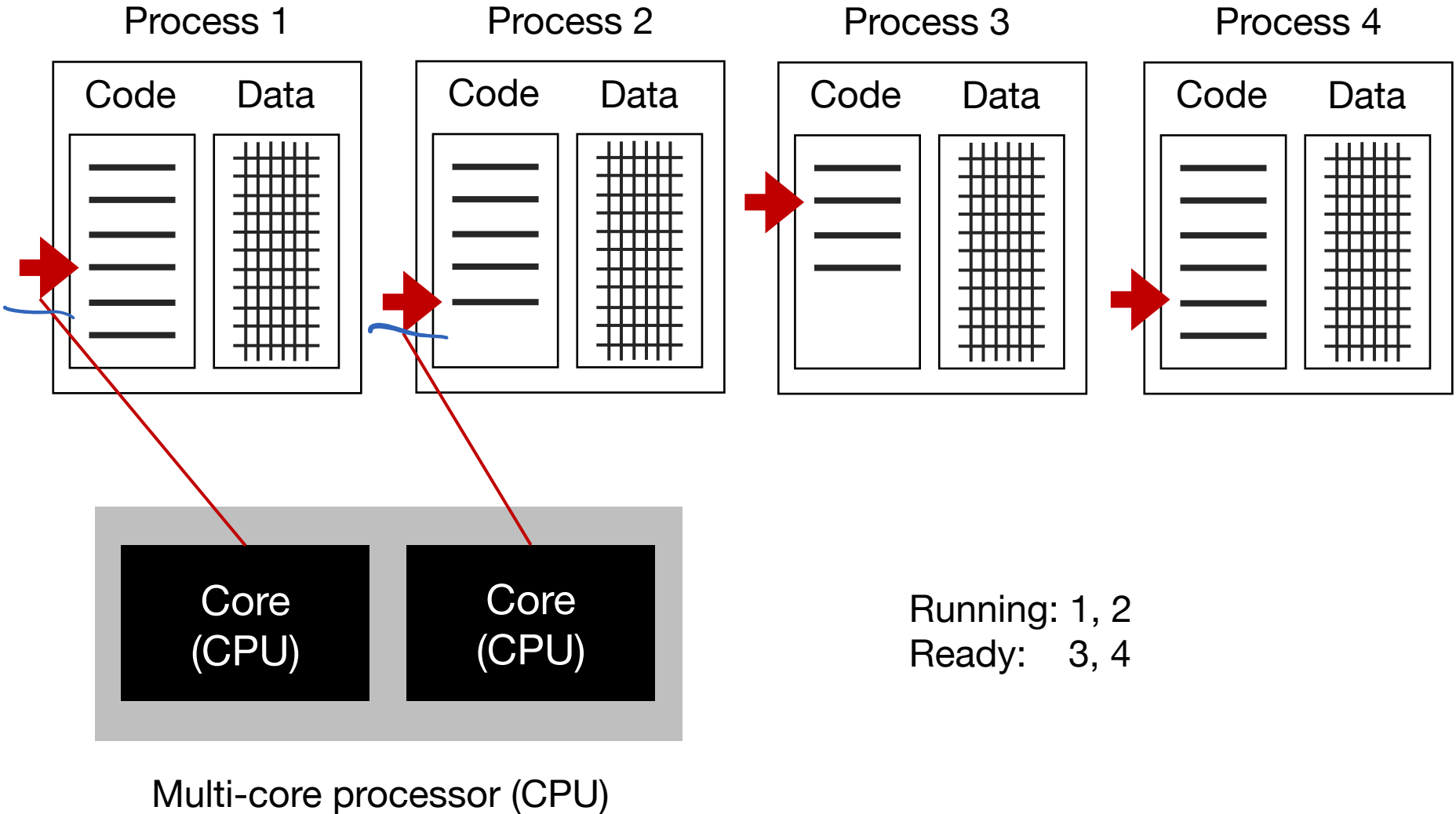
Instruction pointer belongs to a thread within the process

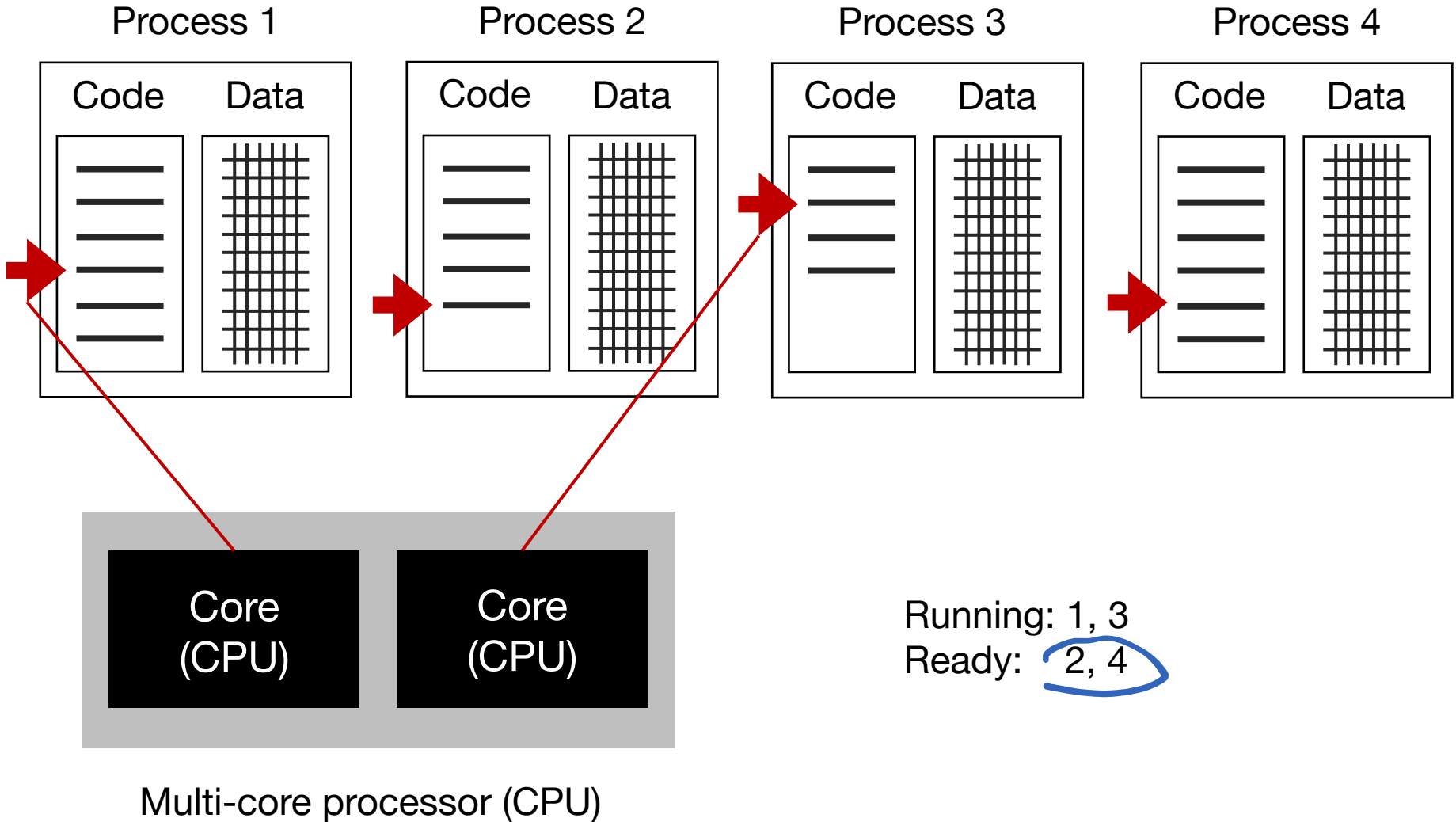


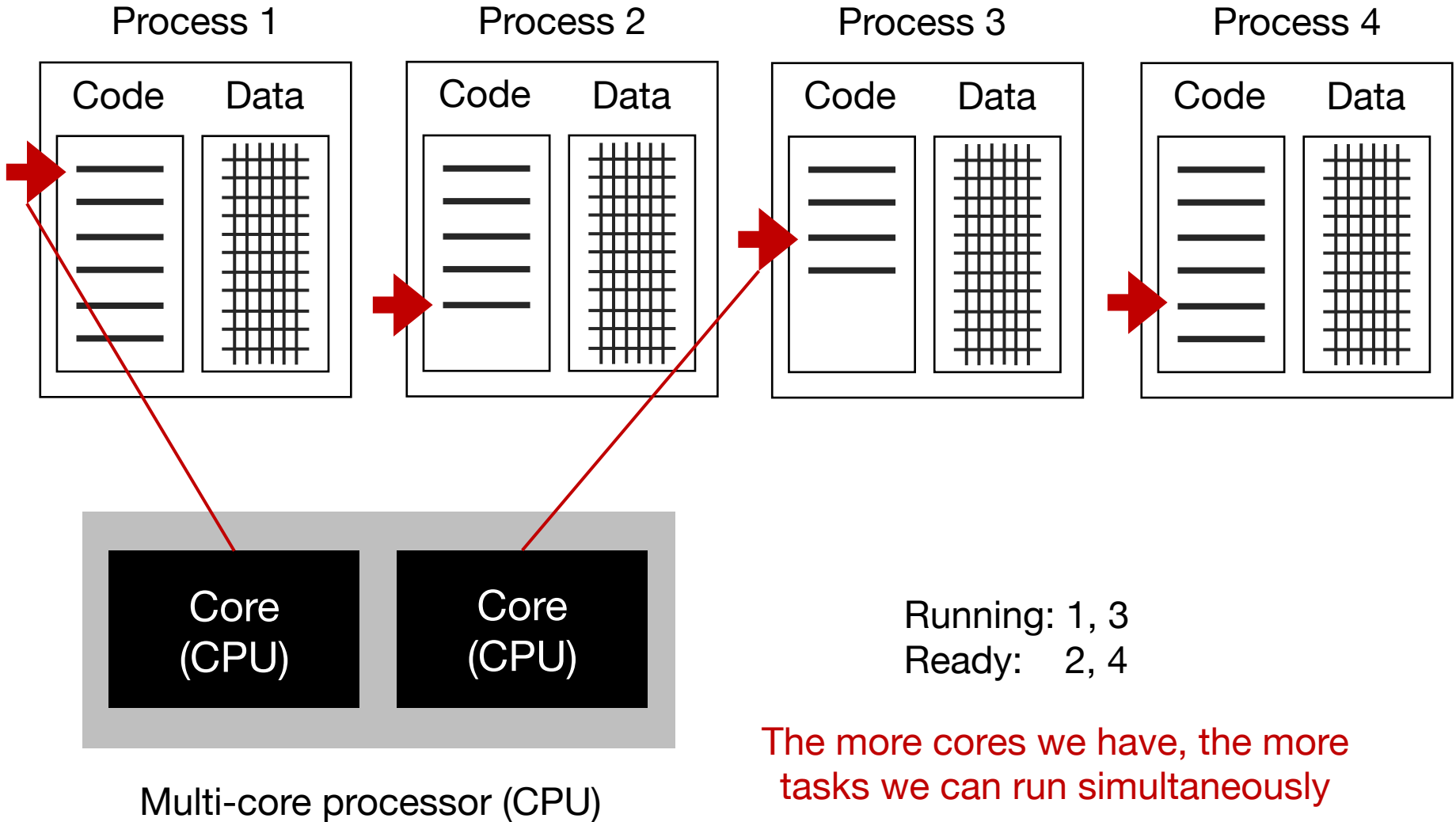
Multi-core processor (CPU)



Multi-core processor (CPU)







Wasted compute resources

Two problems

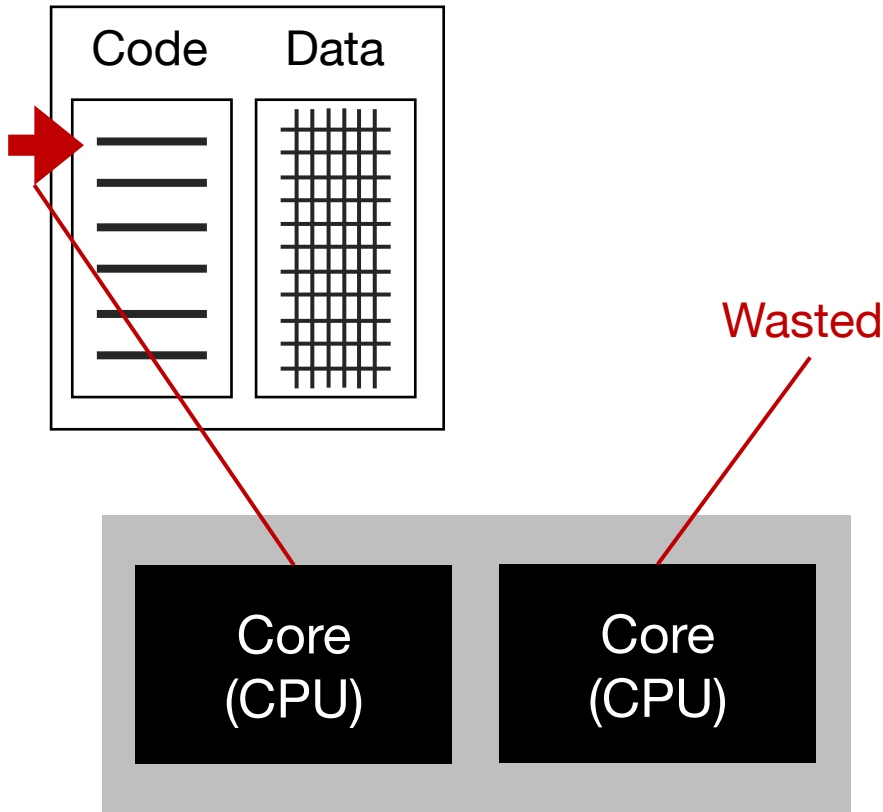
- • Not enough distinct (parallelizable) tasks to utilize all cores
- • Some operations require waiting (task is “blocked”)

Problem 1

Not enough distinct (parallelizable) tasks to utilize all cores

Single-thread.

Process 1

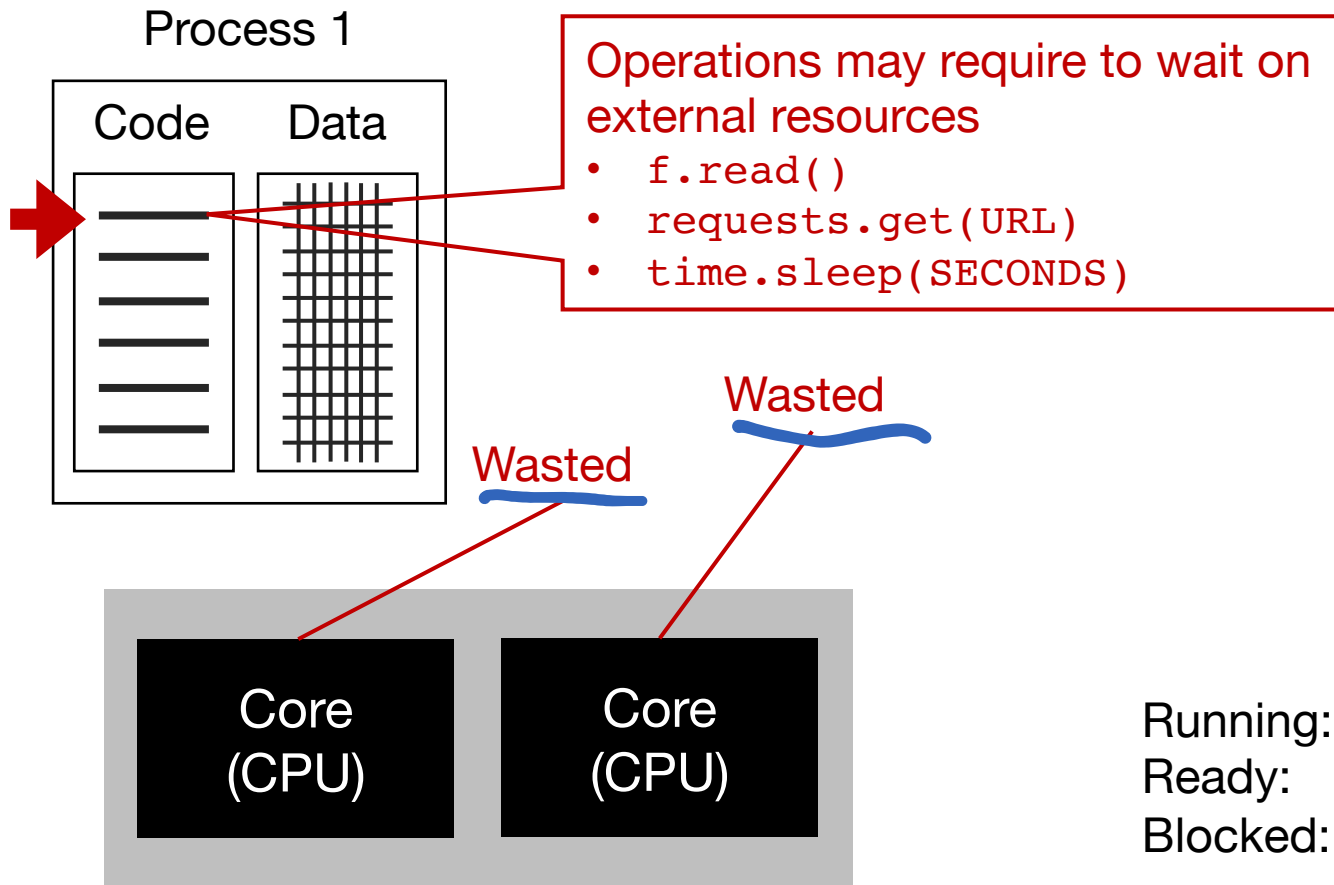


Multi-core processor (CPU)

Running: 1
Ready:

Problem 2

Some operations require waiting (task is “blocked”)



Running:
Ready:
Blocked: 1

Multi-core processor (CPU)

Solution

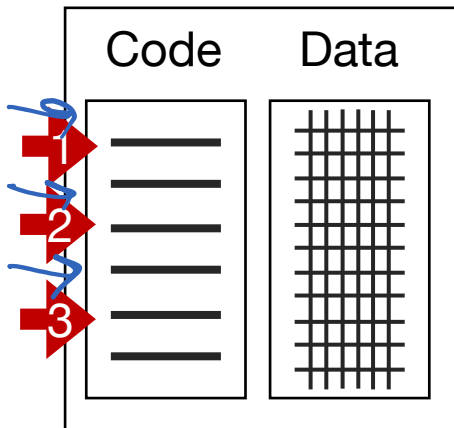
- Thread-level parallelism
- Process-level parallelism
- Task-level parallelism

Solution

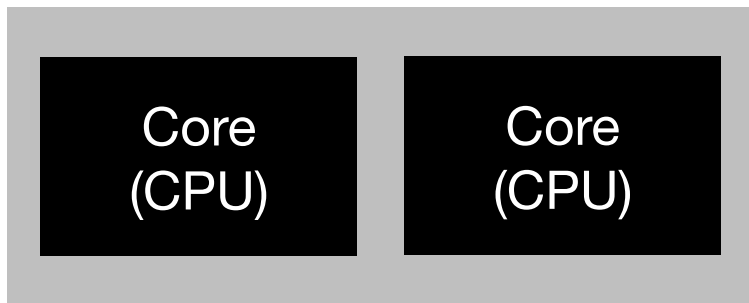
- **Thread-level parallelism**
- Process-level parallelism
- Task-level parallelism

Thread-level parallelism

Process 1

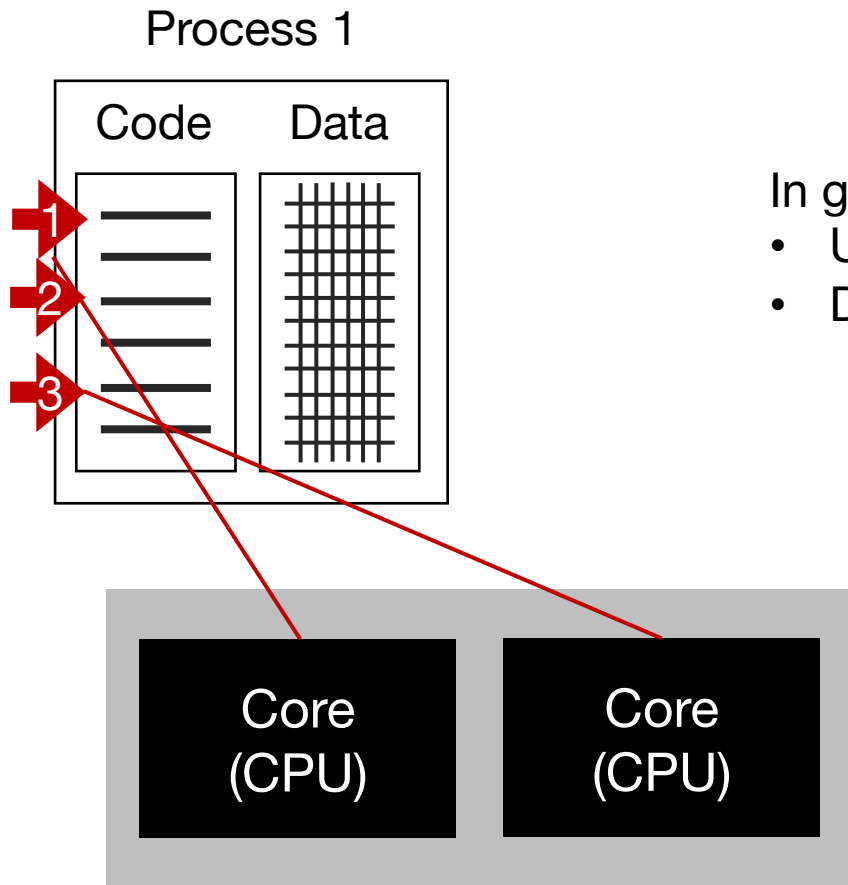


Threads give us multiple instruction pointers in a process, allowing us to execute multiple parts of the code at the same time!



Multi-core processor (CPU)

Thread-level parallelism



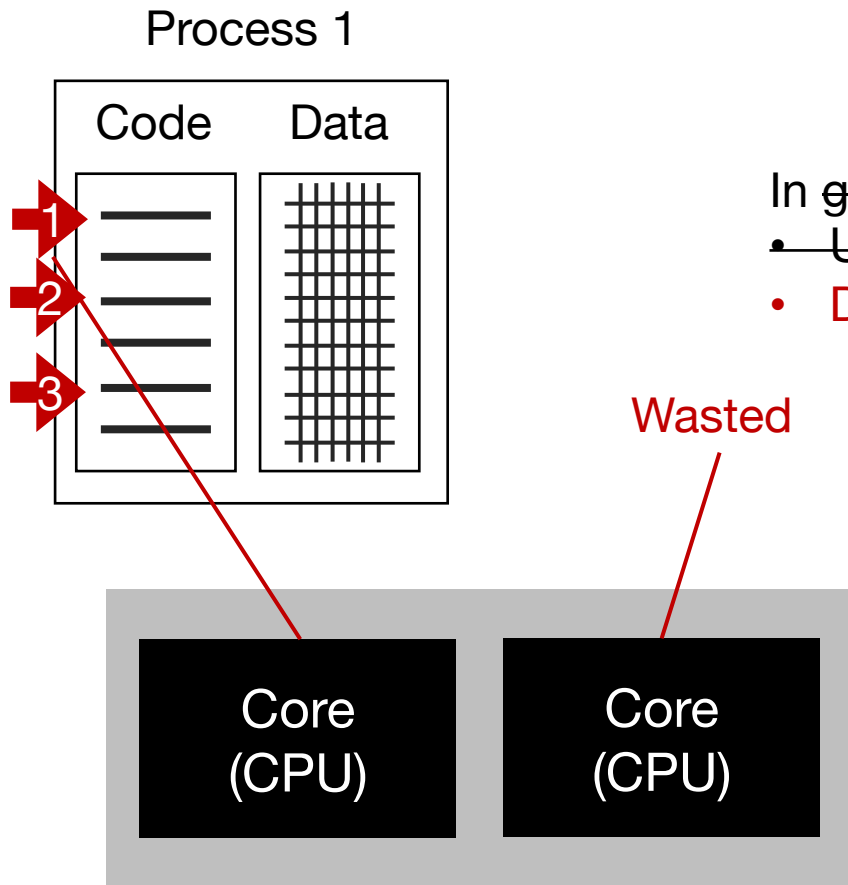
In general, threads help:

- Use multiple cores
- Do useful work when threads are blocking

Running: 1, 3

Ready: 2

Thread-level parallelism in Python



In general Python, threads help:

- ~~Use multiple cores~~ (b/c of the **GIL**)
- Do useful work when threads are blocking

<https://wiki.python.org/moin/GlobalInterpreterLock>

Running: 1
Ready: 3
Blocked: 2

Multi-core processor (CPU)

Thread-level parallelism in Python

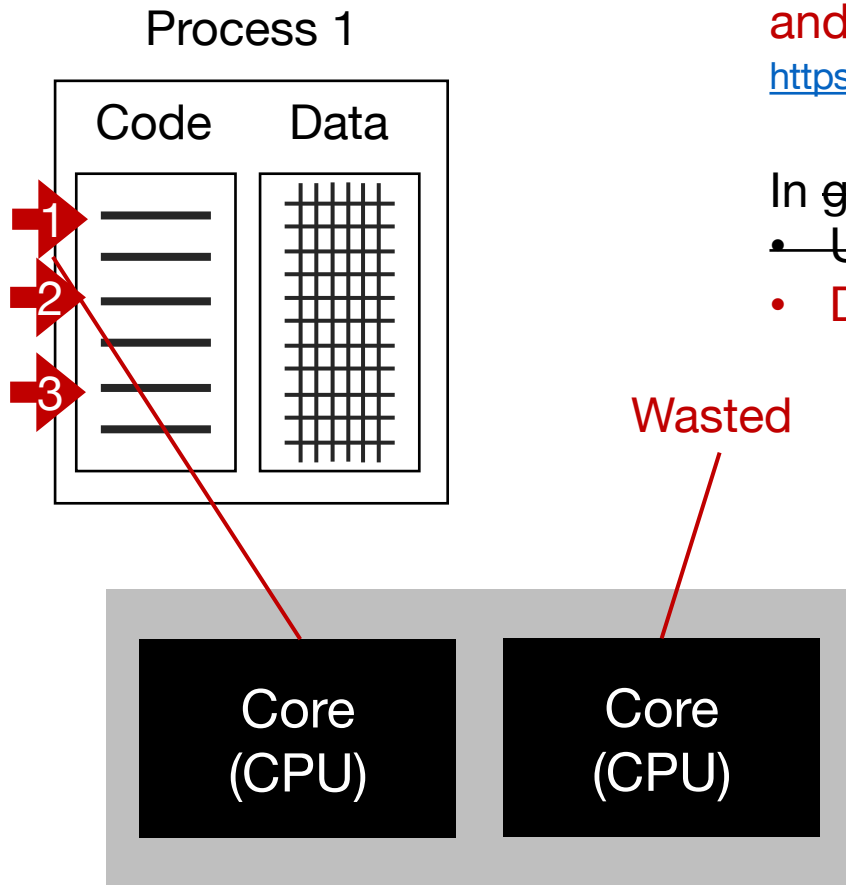
Recommendation: Don't use threads unless you learn a lot on asynchronous processing and/or coroutines

<https://docs.python.org/3/library/asyncio-task.html>

In general Python, threads help:

- ~~Use multiple cores~~ (b/c of the GIL)
- Do useful work when threads are blocking

<https://wiki.python.org/moin/GlobalInterpreterLock>



Multi-core processor (CPU)

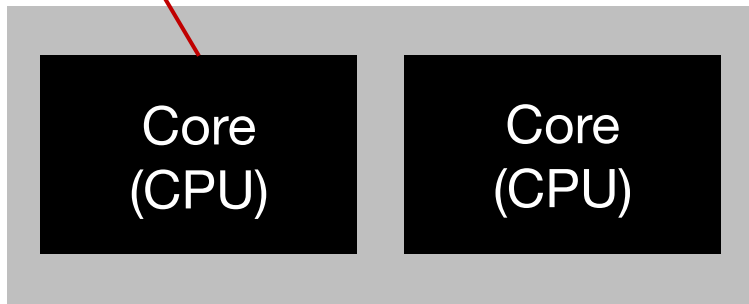
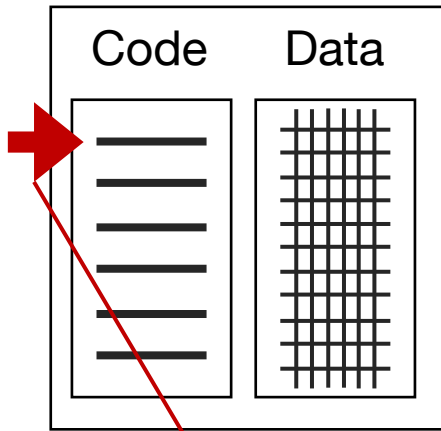
Running: 1
Ready: 3
Blocked: 2

Solution

- Thread-level parallelism
- **Process-level parallelism**
- Task-level parallelism

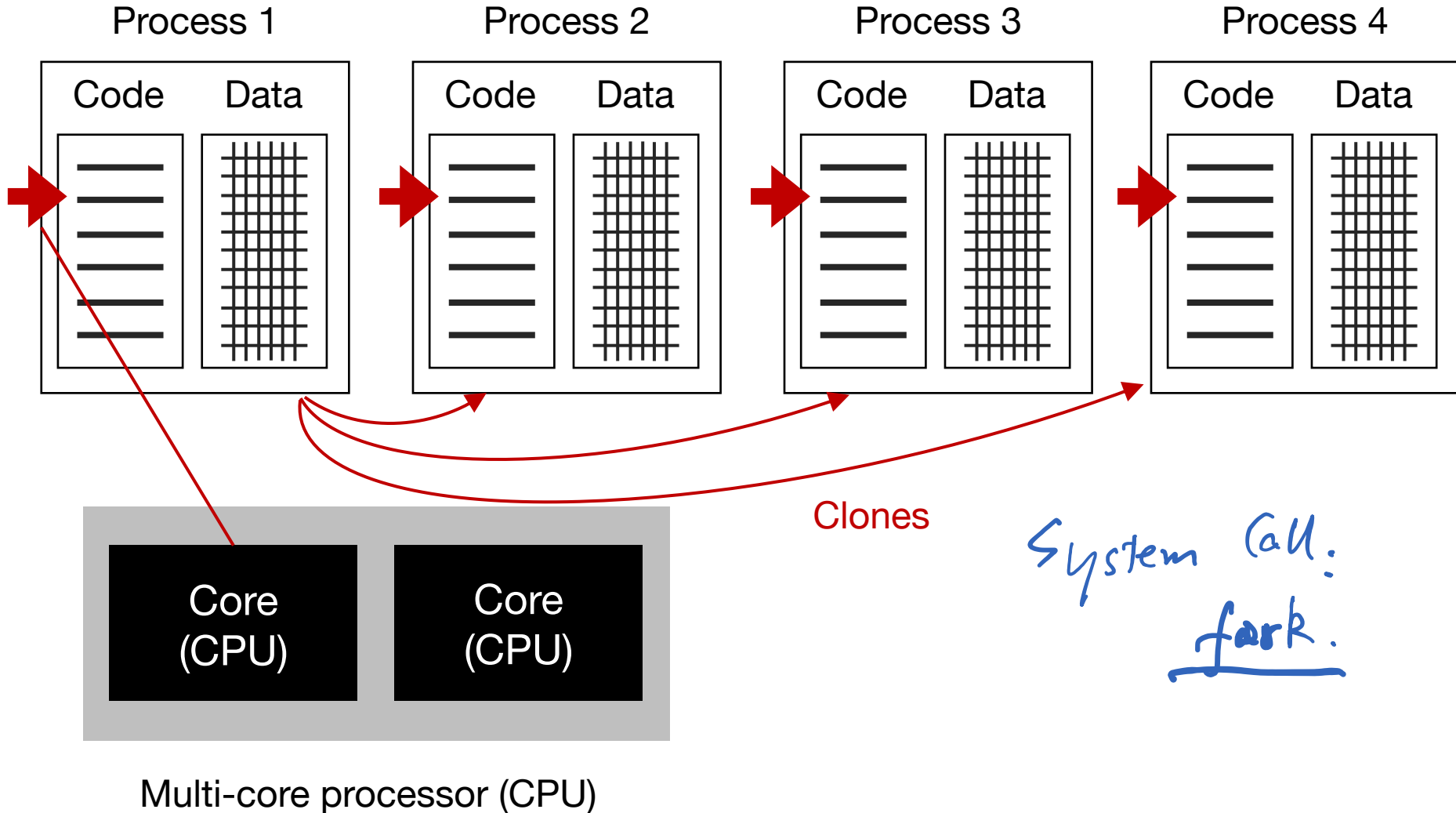
Process-level parallelism

Process 1

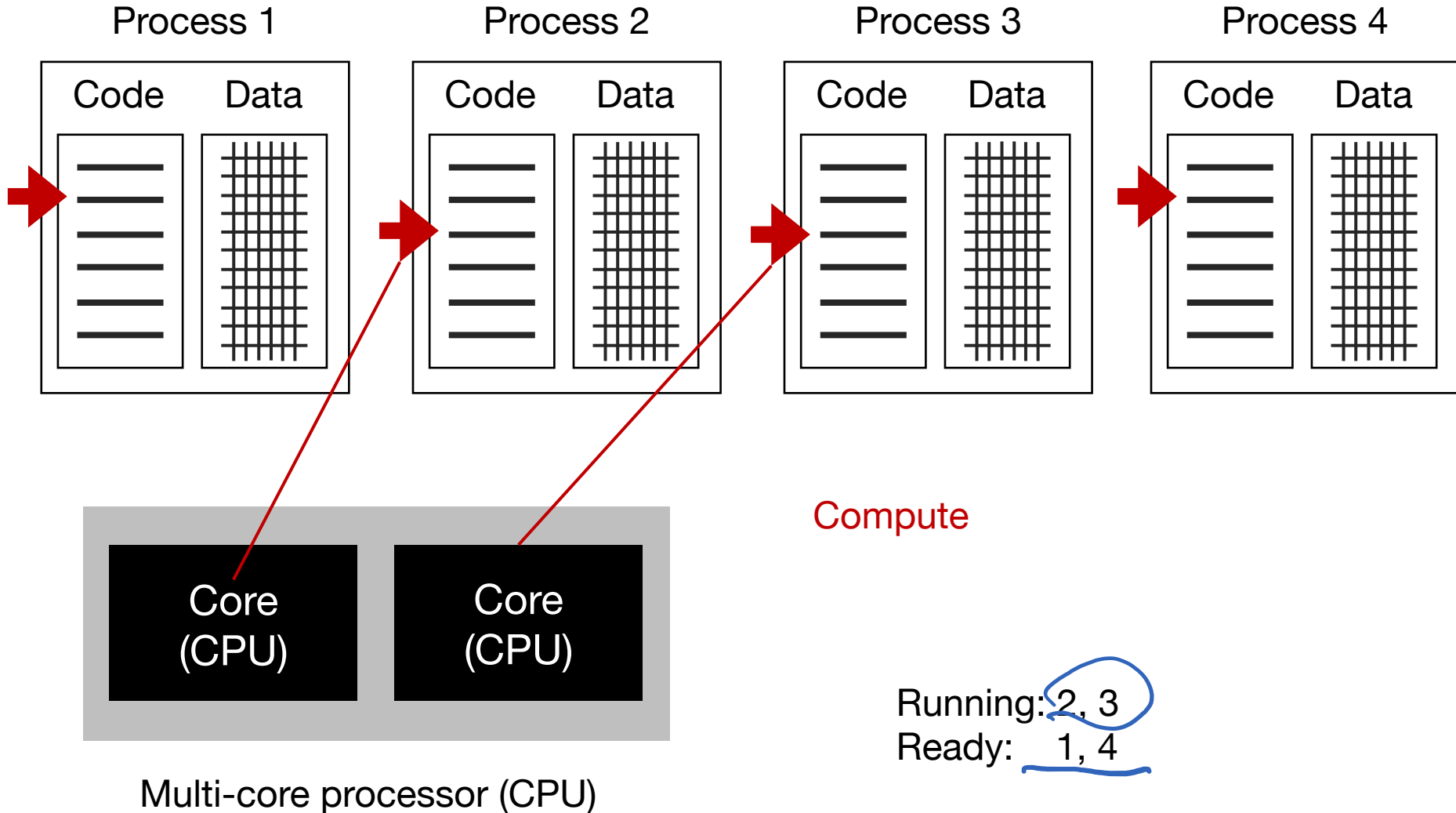


Multi-core processor (CPU)

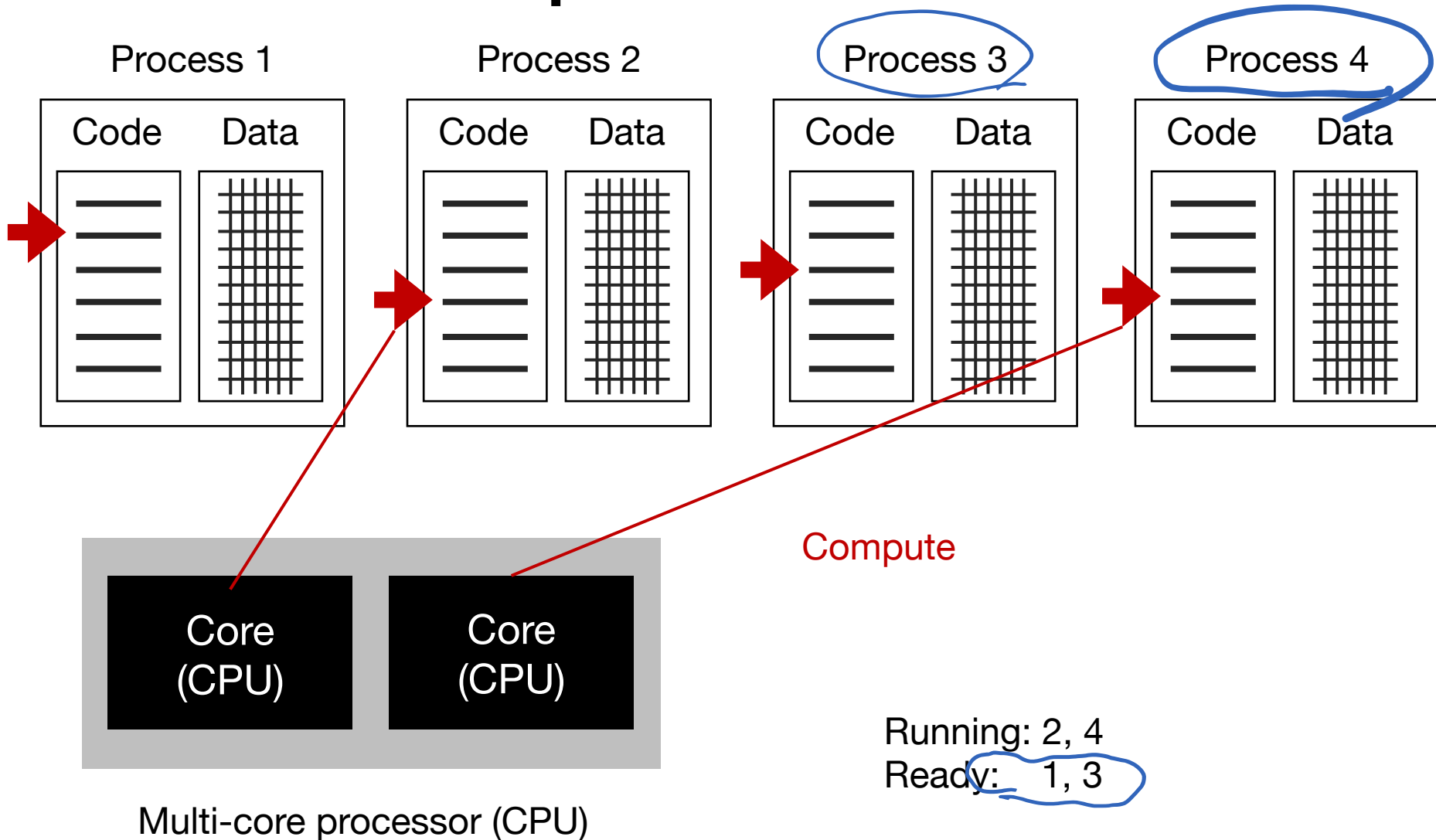
Process-level parallelism



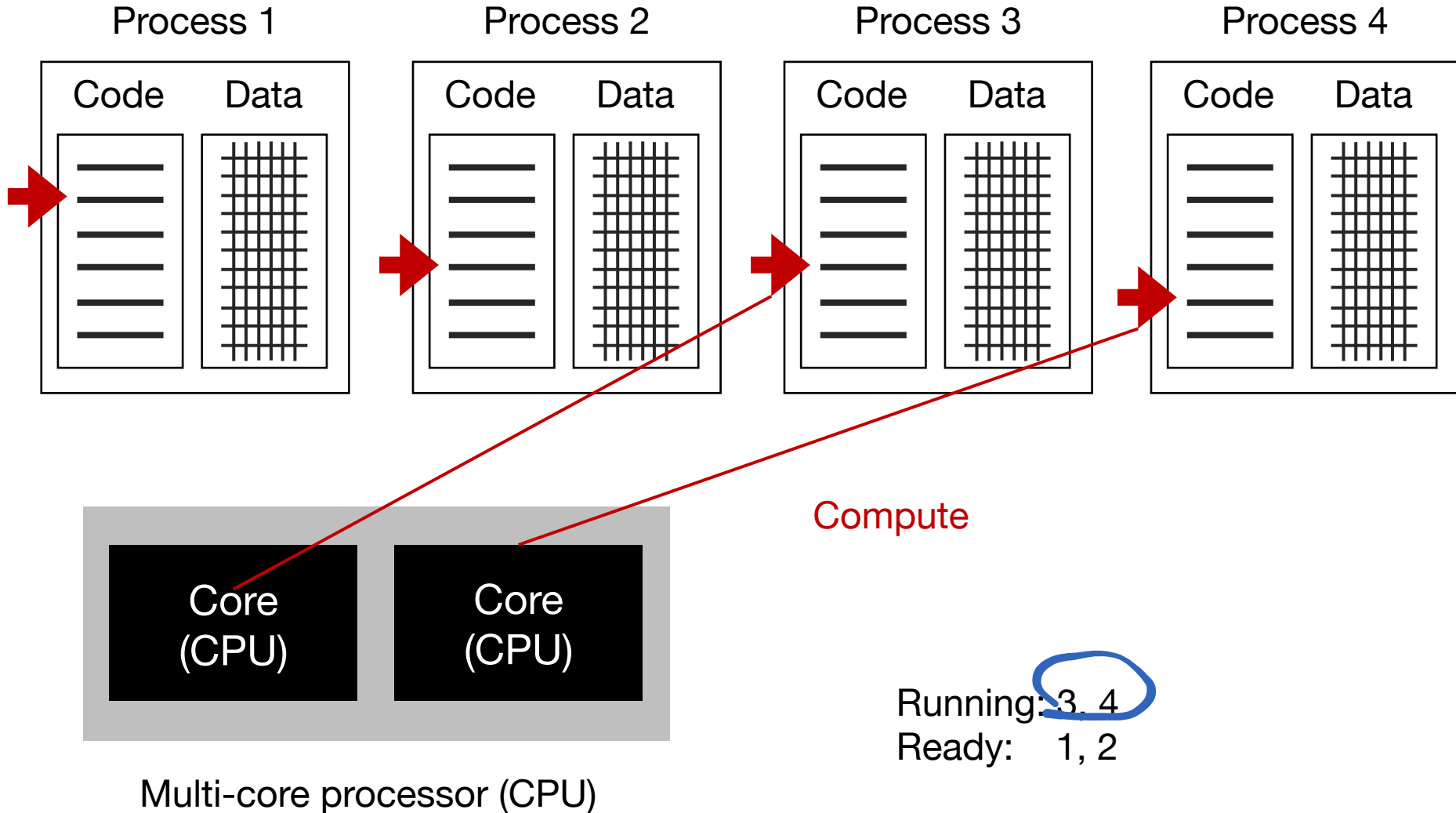
Process-level parallelism



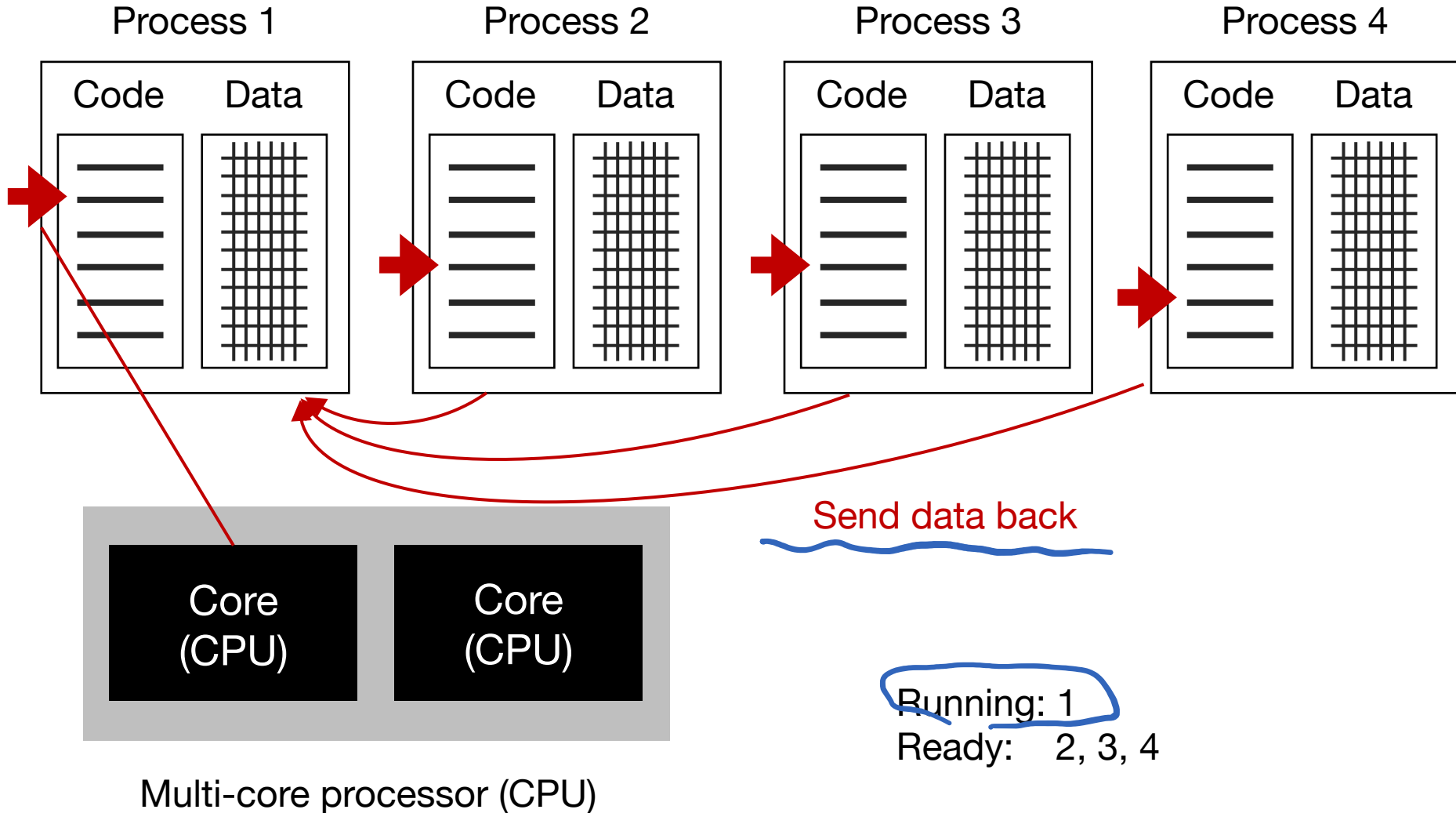
Process-level parallelism



Process-level parallelism

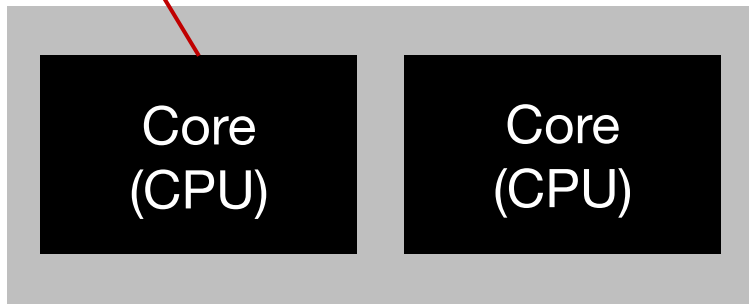
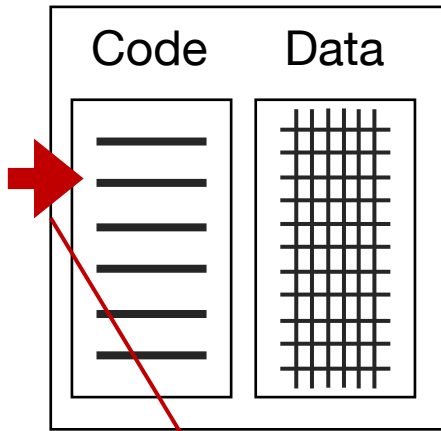


Process-level parallelism



Process-level parallelism

Process 1

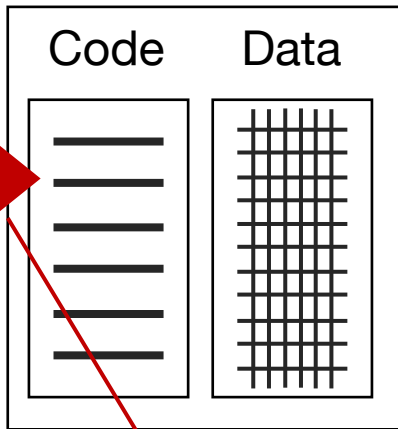


Multi-core processor (CPU)

Process-level parallelism in Python

Process 1

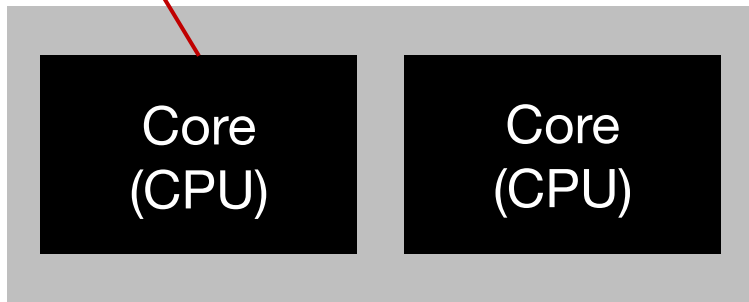
<https://docs.python.org/3/library/multiprocessing.html>



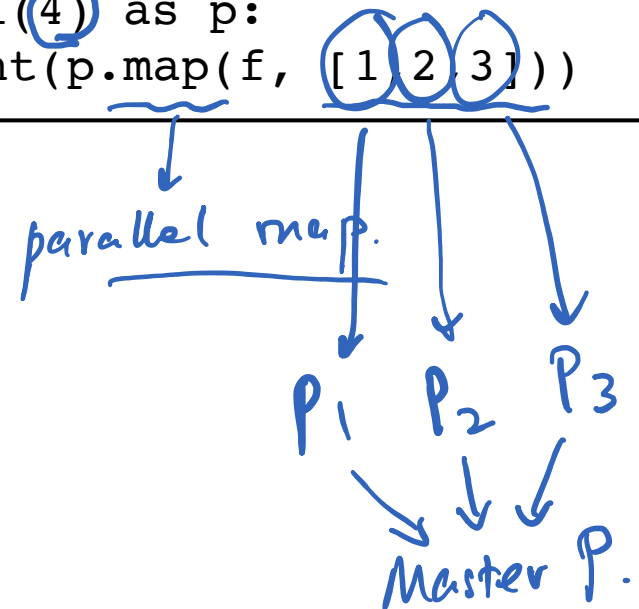
```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(4) as p:
        print(p.map(f, [1, 2, 3]))
```



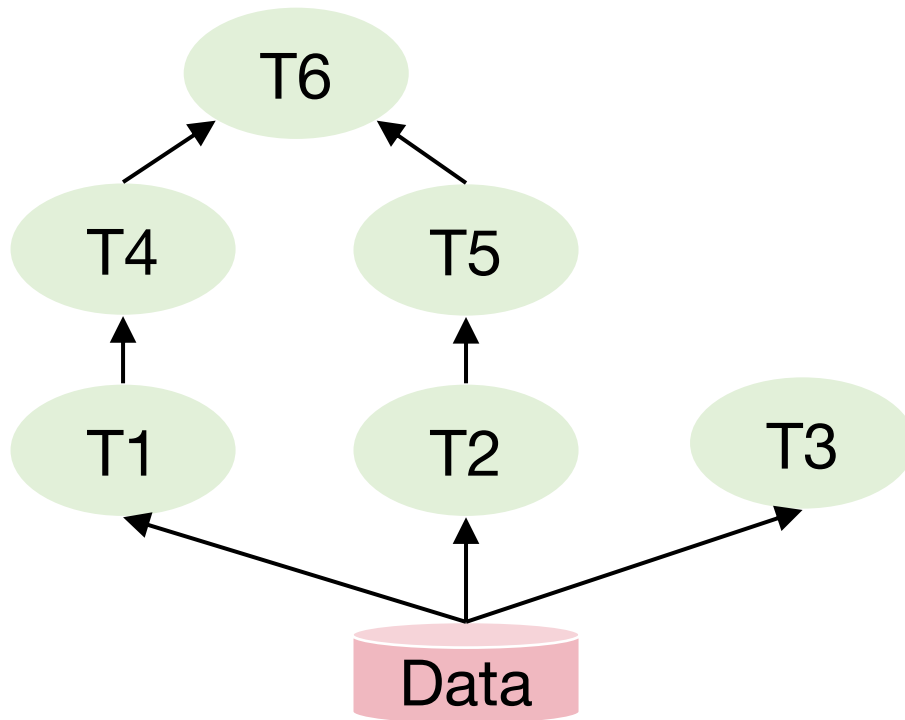
Multi-core processor (CPU)



Solution

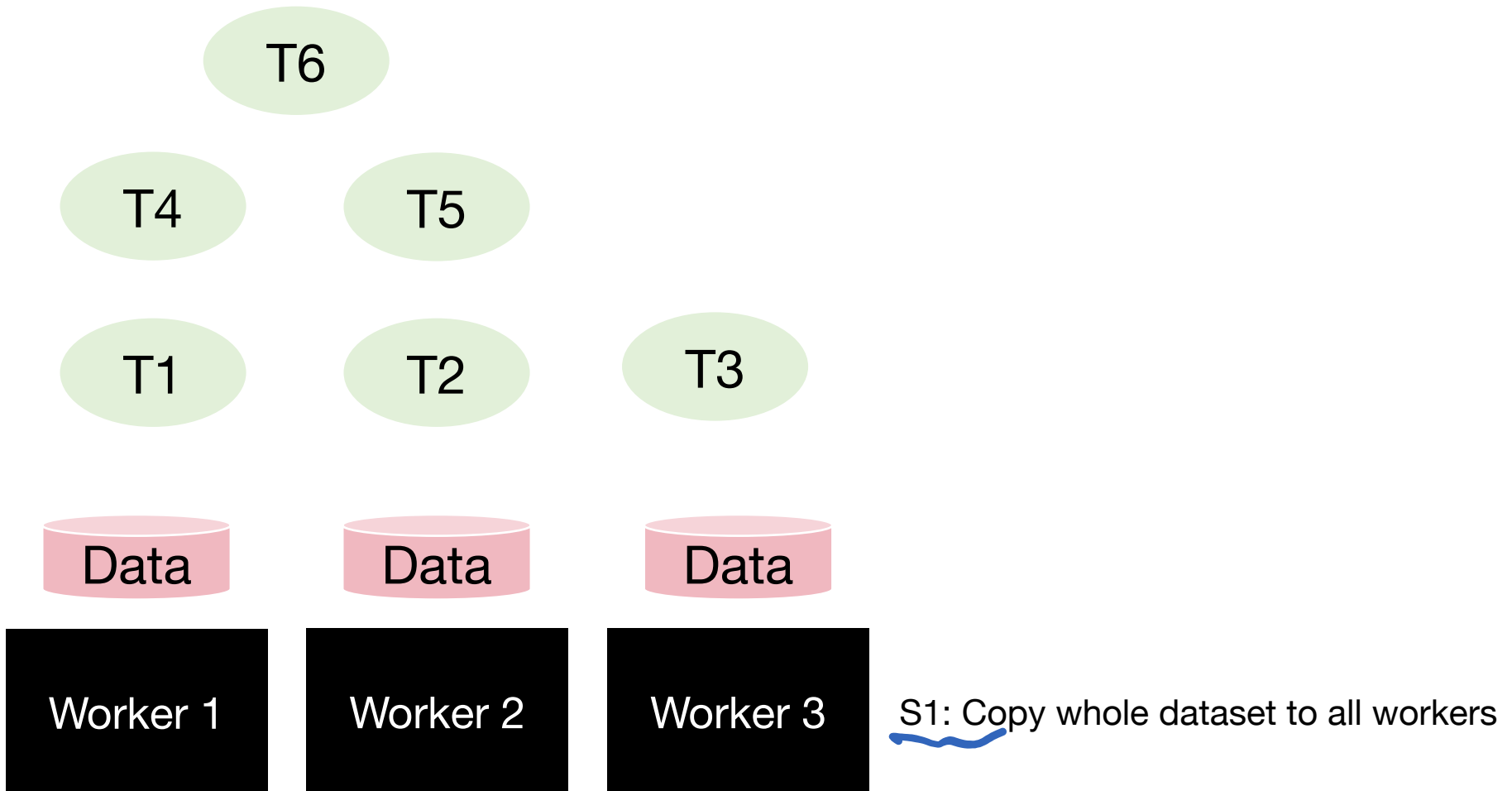
- Thread-level parallelism
- Process-level parallelism
- **Task-level parallelism**

Task-level parallelism

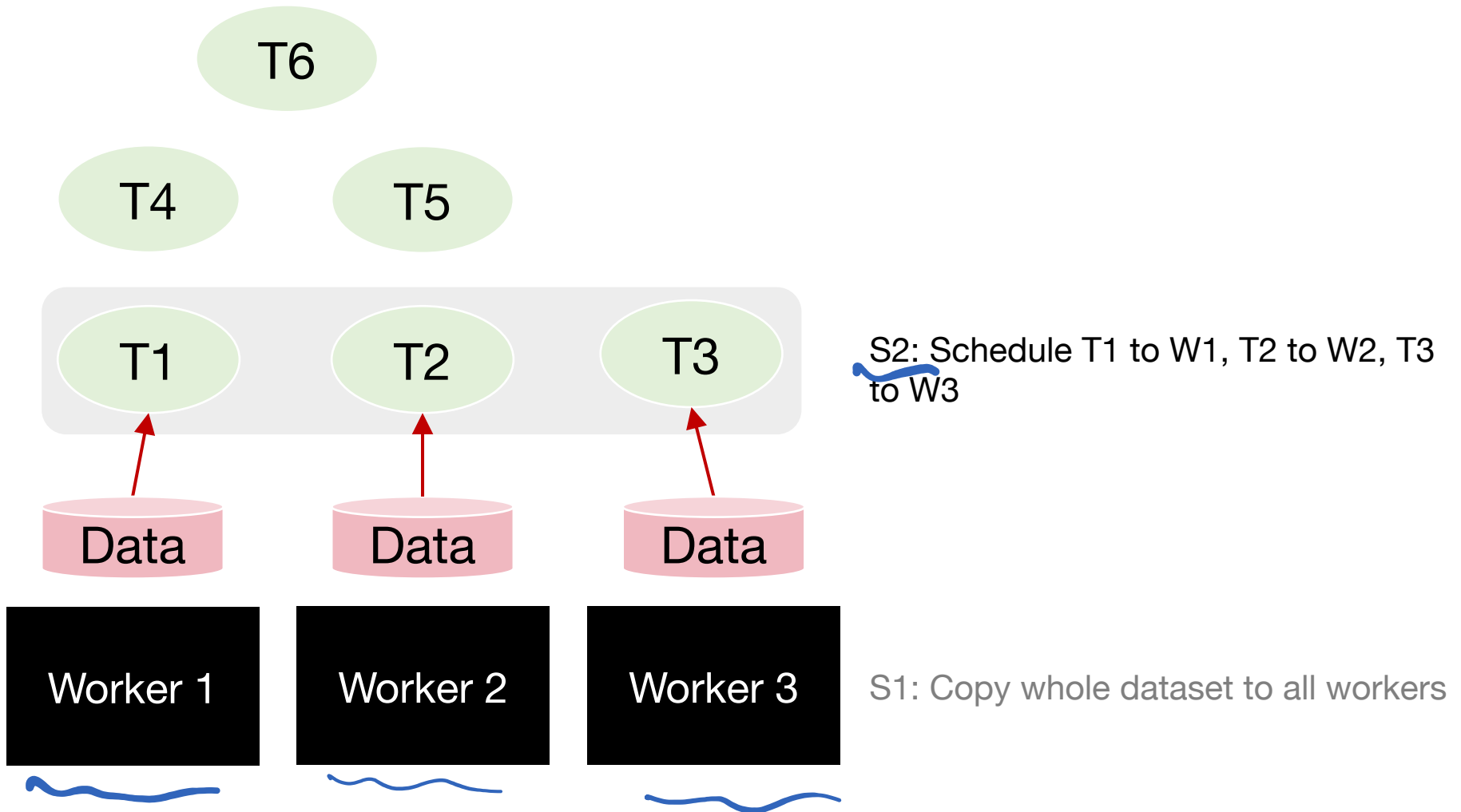


Task DAG
(Directed Acyclic Graph)

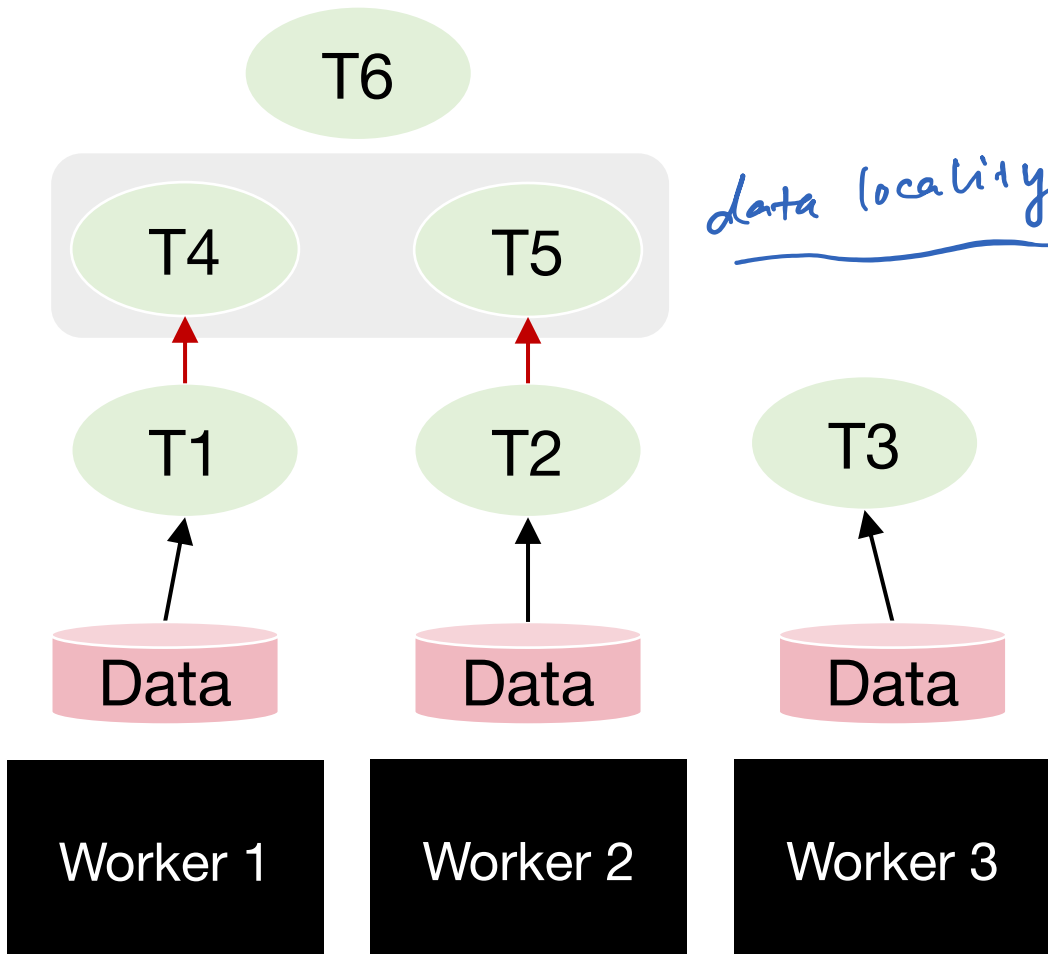
Task-level parallelism



Task-level parallelism



Task-level parallelism

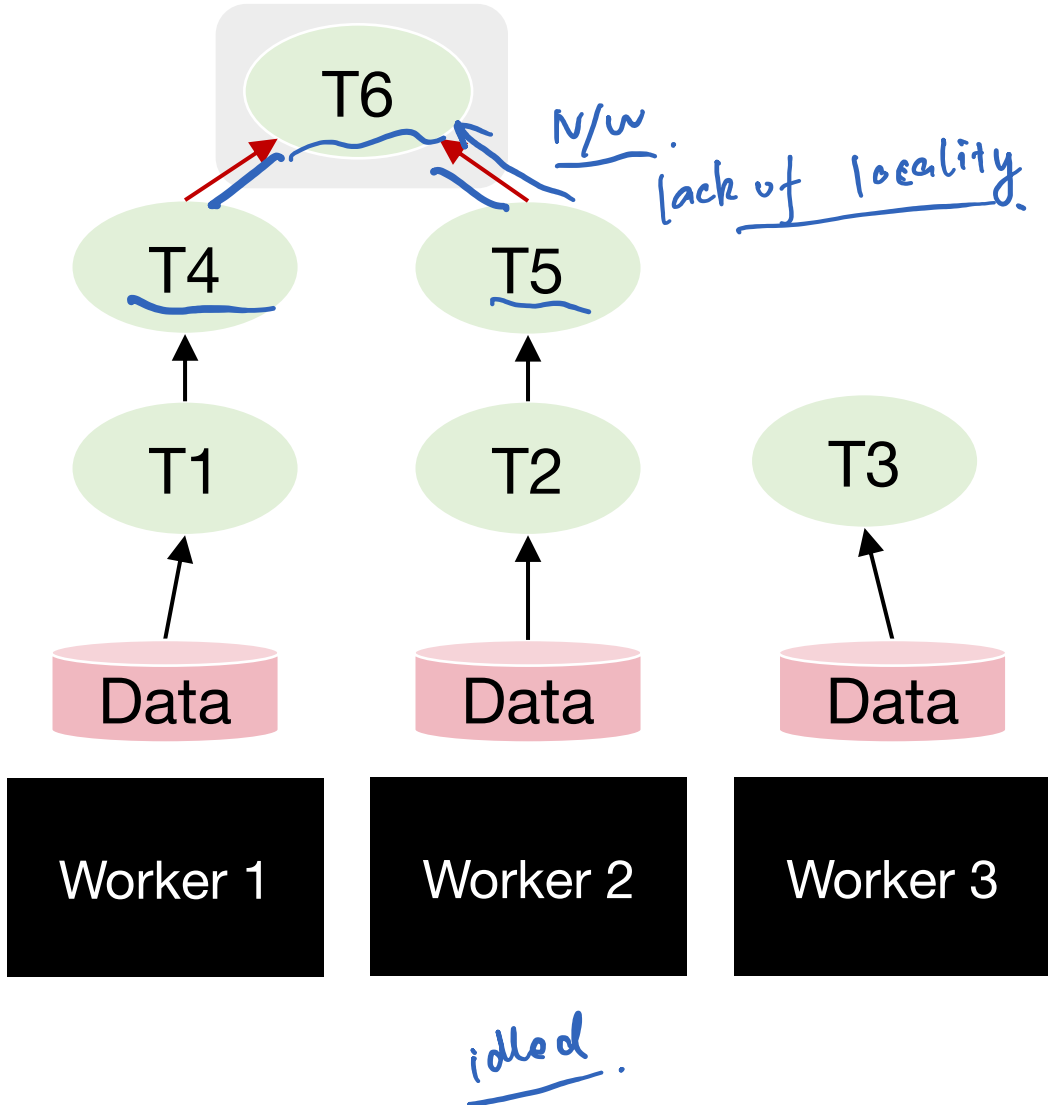


S3: Run T4 after T1 on W1, run T5 after T2 on W2; after T3, W3 is idle

S2: Schedule T1 to W1, T2 to W2, T3 to W3

S1: Copy whole dataset to all workers

Task-level parallelism



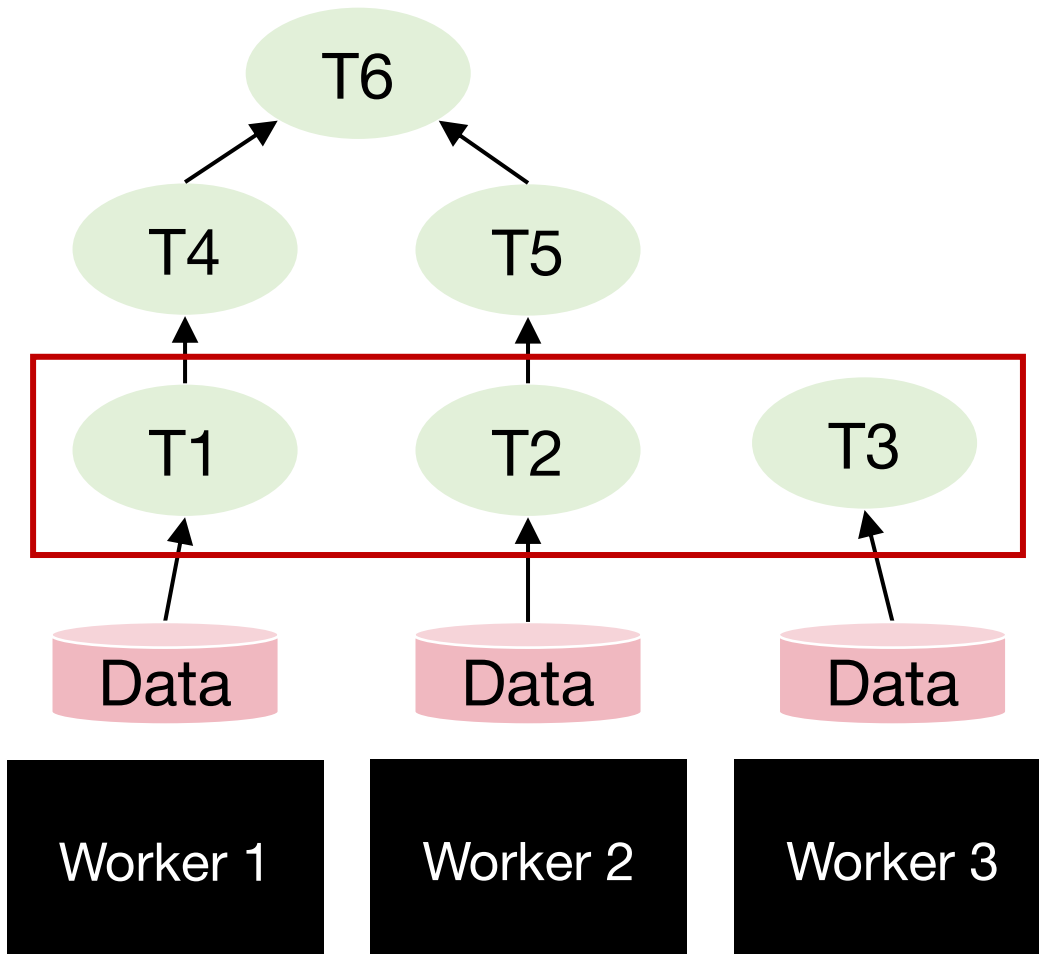
S4: After T4 and T4 ends, run T6 on W1; after T5, W2 is idle

S3: Run T4 after T1 on W1, run T5 after T2 on W2; after T3, W3 is idle

S2: Schedule T1 to W1, T2 to W2, T3 to W3

S1: Copy whole dataset to all workers

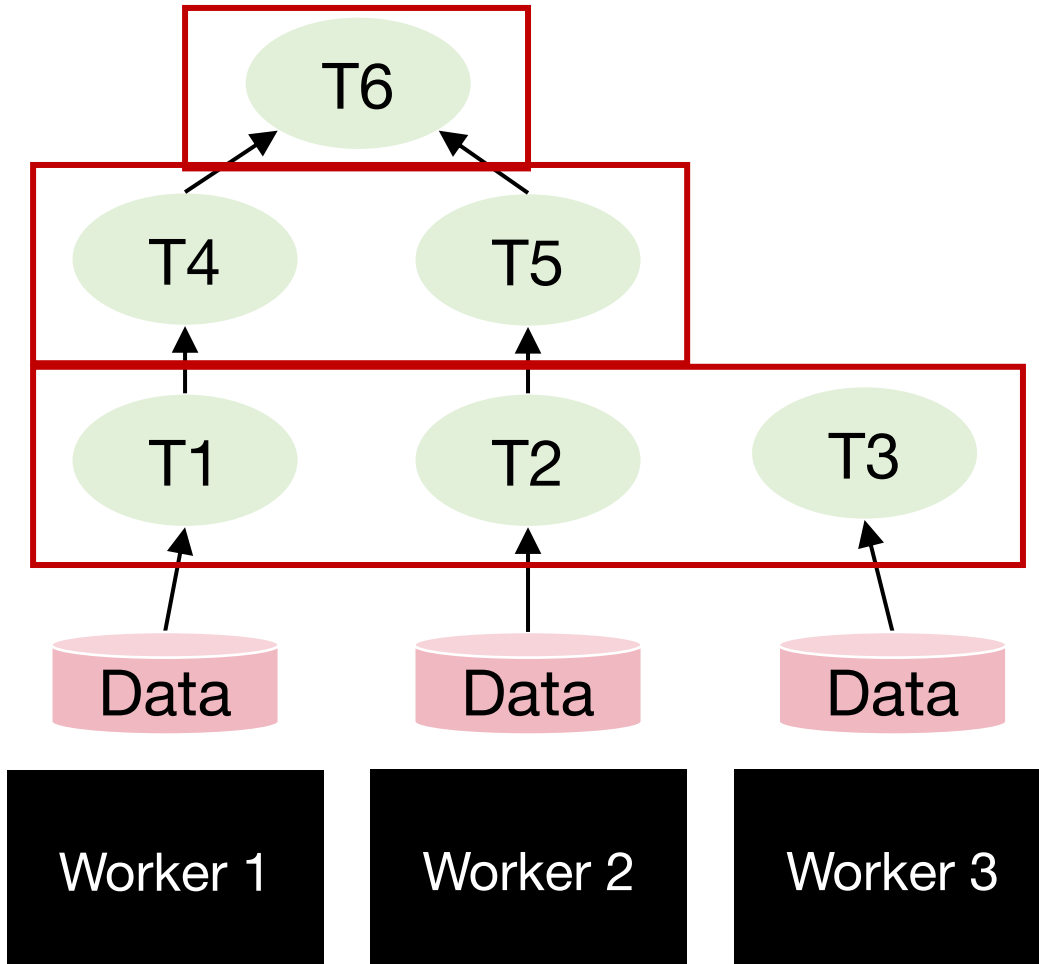
Task-level parallelism



Degree of parallelism is the largest amount of parallelism possible in the DAG:

- How many tasks can be run in parallel at most

Task-level parallelism



Resource wastage on idle workers

Overtime degree of parallelism drops!

Degree of parallelism is the largest amount of parallelism possible in the DAG:

- How many tasks can be run in parallel at most

Quantify benefit of parallelism: Speedup

$$\text{Speedup} = \frac{\text{Completion time given 1 worker}}{\text{Completion time given } \underline{N} \text{ worker}}$$

Quantify benefit of parallelism: Speedup

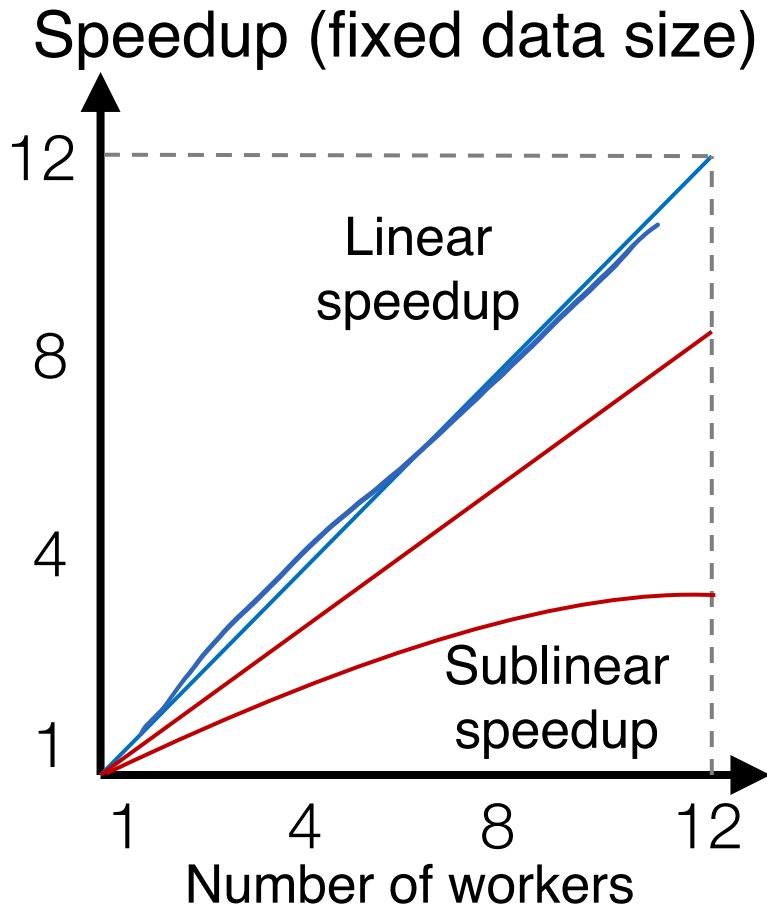
$$\text{Speedup} = \frac{\text{Completion time given 1 worker}}{\text{Completion time given N worker}}$$

Q: Given N workers, can we get a speedup of N?

It depends.

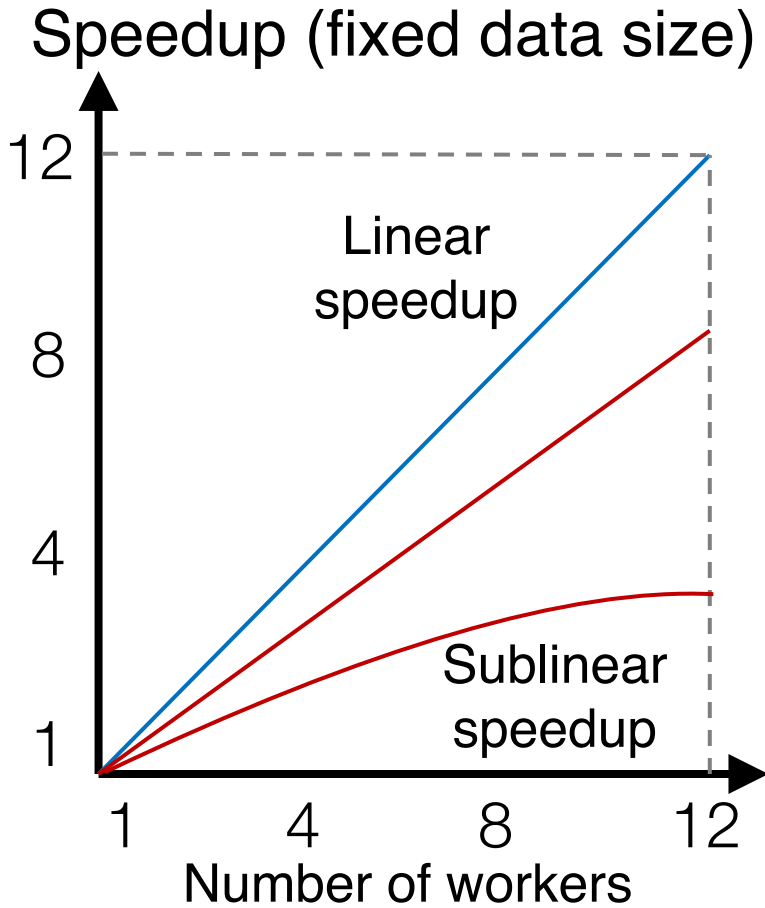
Map Reduce

Quantify speedup

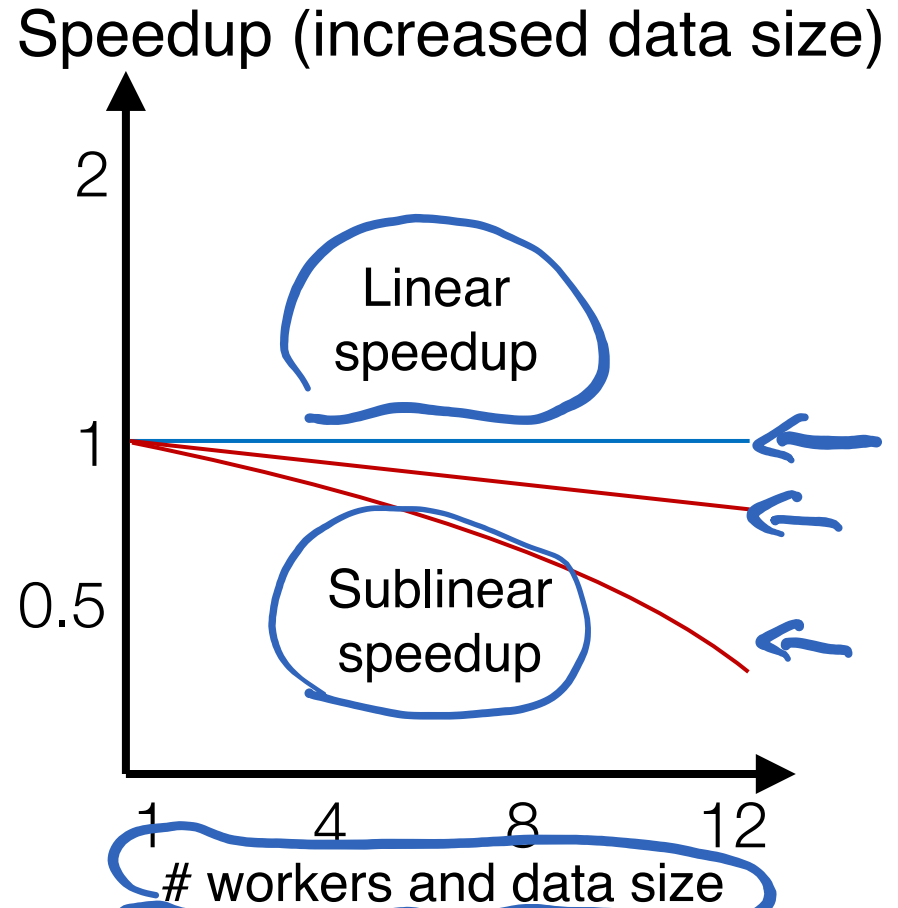


Strong scaling

Quantify speedup



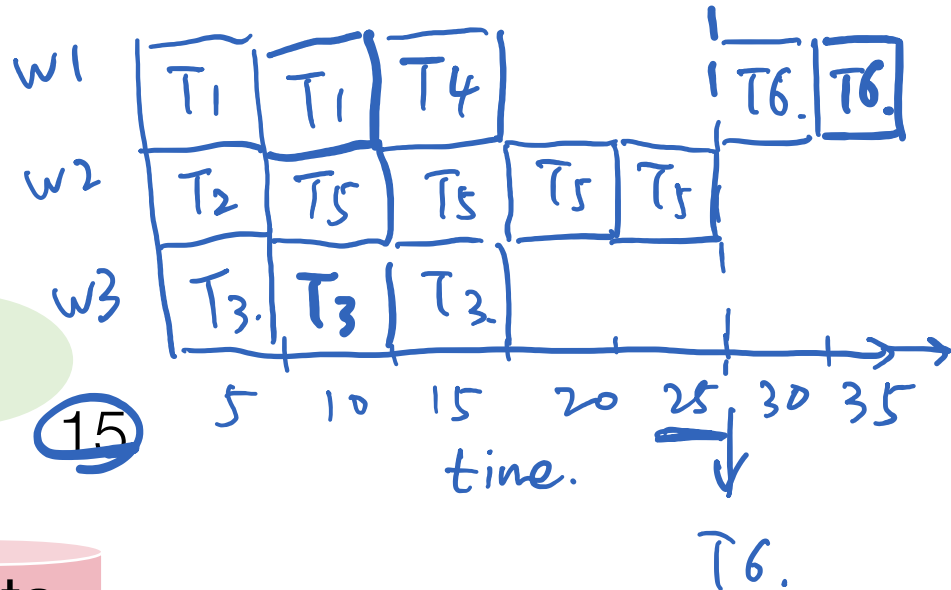
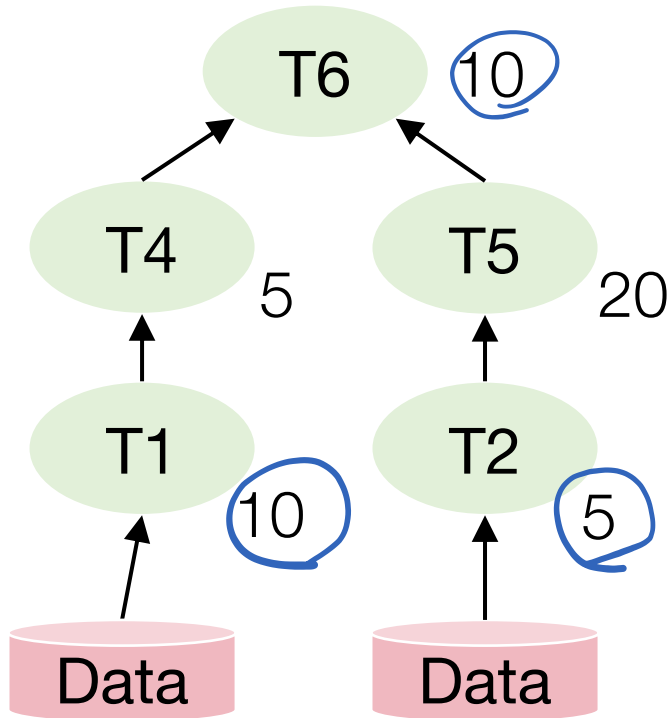
Strong scaling



Weak scaling

Idle resources in task-level parallelism

Q: How long would job completed?



Worker 1

Worker 2

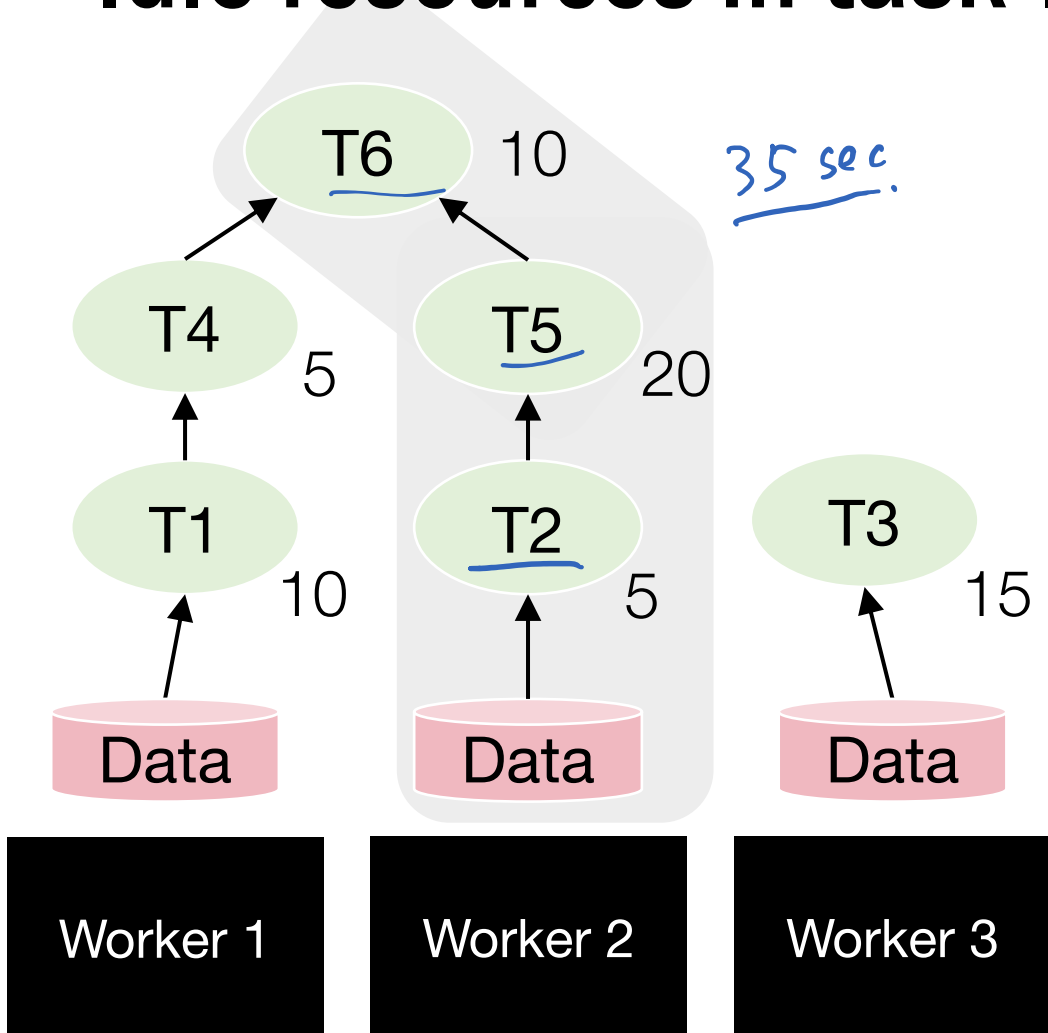
Worker 3

Task completion time varies

35 sec

Takeaway?

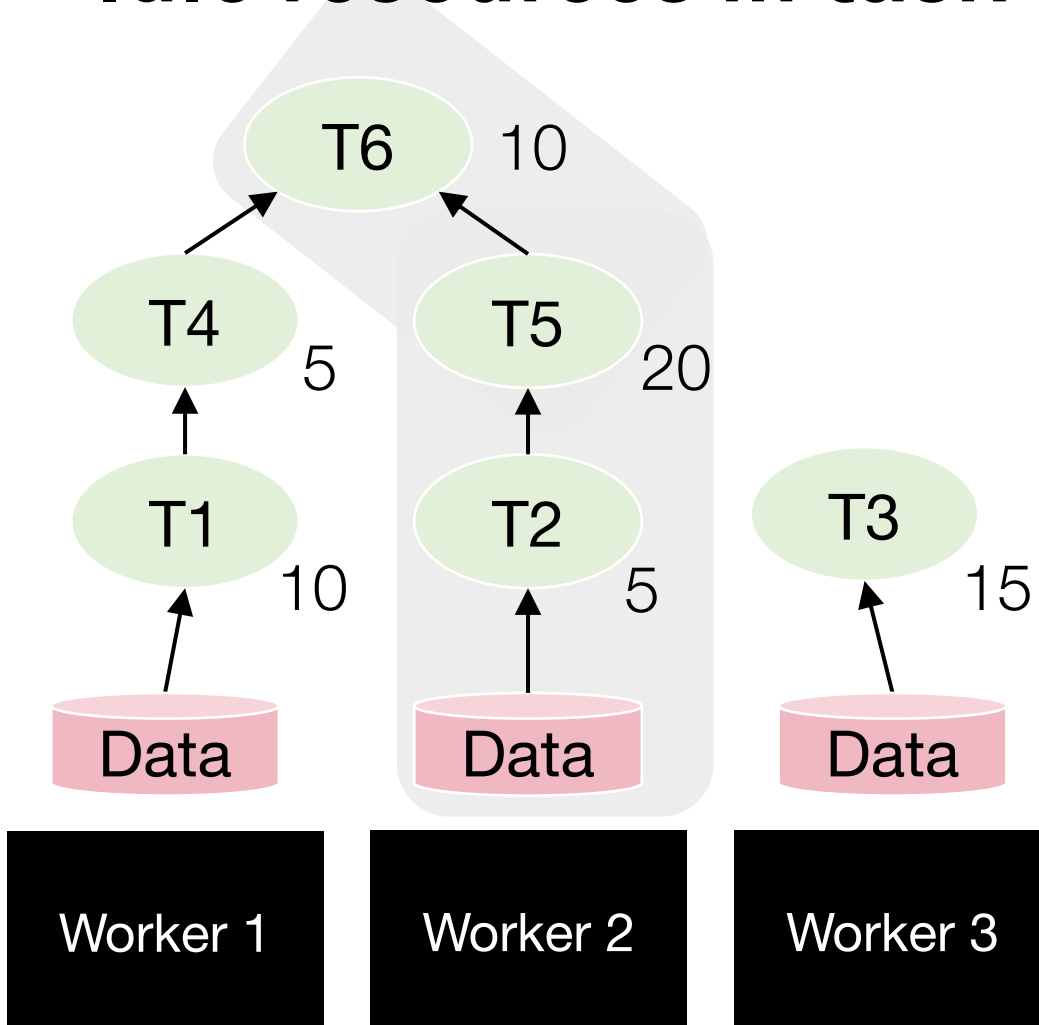
Idle resources in task-level parallelism



- Job completion time is always bounded by the **longest path** in the DAG

Task completion time varies

Idle resources in task-level parallelism



Task completion time varies

- Job completion time is always bounded by the **longest path** in the DAG
- **Potential optimization:** The scheduler can elastically release a worker if it knows the worker will be idle till the end
 - Can **save \$ cost** in cloud

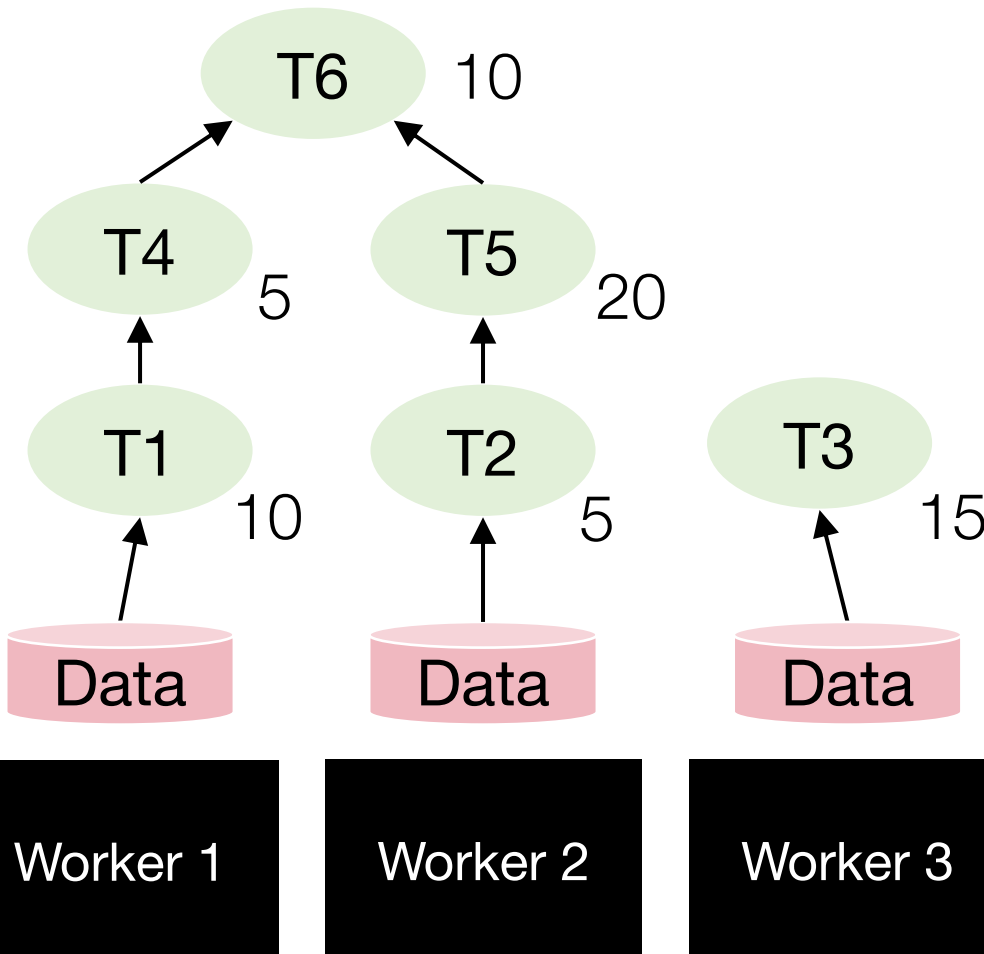
Serverless Computing

Idle resources in task-level parallelism

Q: What's the job completion time with 1 worker?

Q: What's the job completion time with 3 worker?

Q: What's the speedup?



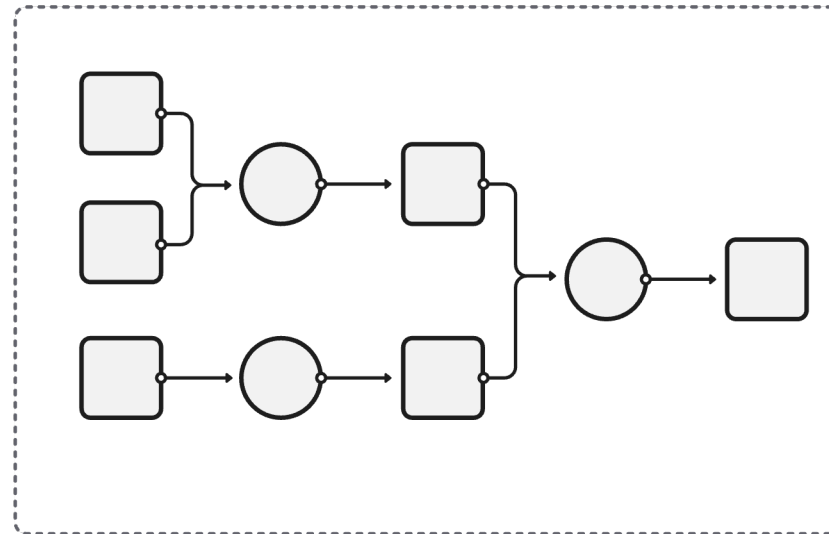
Task parallelism in Dask

Collections

(create task graphs)

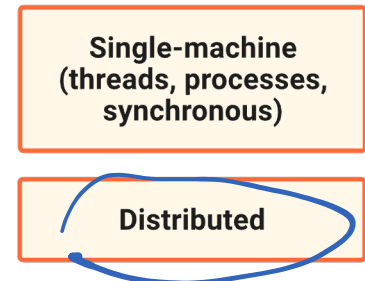


Task Graph



Schedulers

(execute task graphs)



* <https://docs.dask.org/en/stable/>

* <https://docs.dask.org/en/stable/scheduling.html>

Dask's task graph and workflow

```
import dask
import dask.array as da
→ x = da.random.normal(size=1_000_000, chunks=100_000)
```

Dask's task graph and workflow

```
import dask
import dask.array as da
x = da.random.normal(size=1_000_000, chunks=100_000)
```



```
data = x.compute()
```

Lazy evaluation: Dask computation can be triggered manually, e.g., `.compute()`

- only when the result is needed

Dask's task graph and workflow

```
import dask
import dask.array as da
x = da.random.normal(size=1_000_000, chunks=100_000)
```

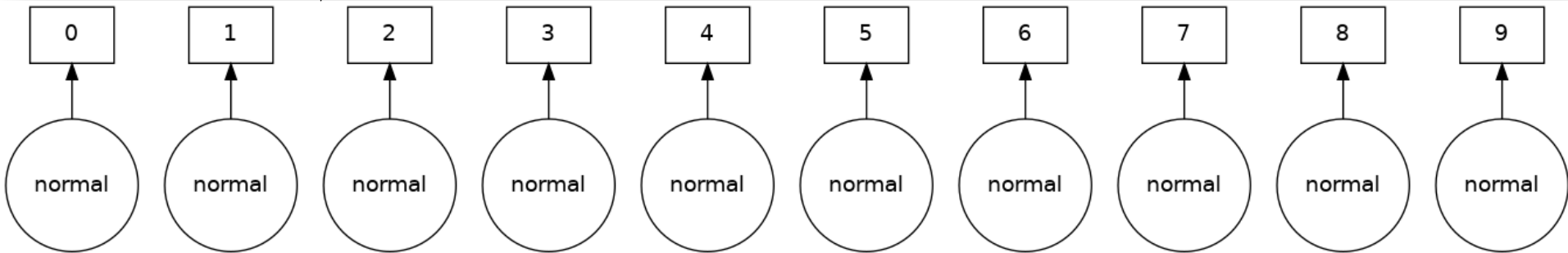
```
data = x.compute()
```

Lazy evaluation: Dask computation can be triggered manually, e.g., `.compute()`

- only when the result is needed

```
dask.visualize(x)
```

Draw the task graph using `.visualize()`



Dask task graph

Next steps

- Assignment 2 is out
 - Due on Wednesday, 03/15, 11am ET
- Project bidding is due this Friday, 02/24
- Next Monday, 02/27
 - Midterm review

Dask demo