# Introduction to Parallel Computing

## Week 1 – Part 1

**Amittai Aviram – 2020-09-08 – Boston University**

CS 591 A1 – Parallel Computing and Programming

# Welcome to Parallel Computing and Programming!

## About the Course

- We meet once a week on Tuesday night, 6:30 PM to 9:15 PM, in B06A.
- The syllabus and course materials are on Blackboard.
- Office hours are Wednesday, 3:00 PM to 6:00 PM, over Zoom.  (Please see the Information tab or the syllabus in Blackboard for the link.)
- This course is for advanced undergraduate and master's students.
- This course requires knowledge of Python 3 and modern C++.  You can learn the C++ in the early weeks of the course.
- This is a brand-new course.

# More Course Details

- Students will complete 8 weekly programming assignments—first in Python 3 and then in C++.
- Students will work in pairs (groups of two) for the weekly programming assignments.  It is up to you to distribute the work evenly and fairly.
- The plan is to pair more with less experienced students, so more experienced students will have to educate and explain.
- Students will prepare a final project—tentatively, a program to simulate the Covid-19 pandemic using parallel processing techniques.
- Students will work on the final project in larger teams.

# About the Instructor

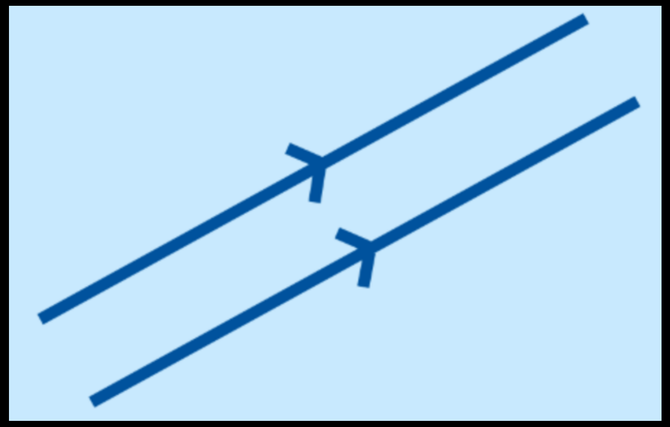
## Amittai Aviram

- Pronounced “ah-mit-TYE,” rhymes with “guy”; “ah-vee-RAHM”
- PhD in Computer Science and PhD in English
- Used to be a professor of English and comparative literature
- Dissertation on parallel programming (with OpenMP)
- Has been a software engineer for 8 years

# Terms

## Parallelism and Concurrency

- Parallelism
- Concurrency
- Doing more than one thing “at the same time”
- In this course, I will treat these as interchangeable.



# People Are Parallel Processors

# The Human Body is a Complex of Parallel Systems

# The World is a Parallel System

More precisely, we usually mean an organization of agents (workers) to accomplish a *single task* by working together on it at the same time.

The task may be variously broken up into sub-tasks.

This is essentially *division of labor*.

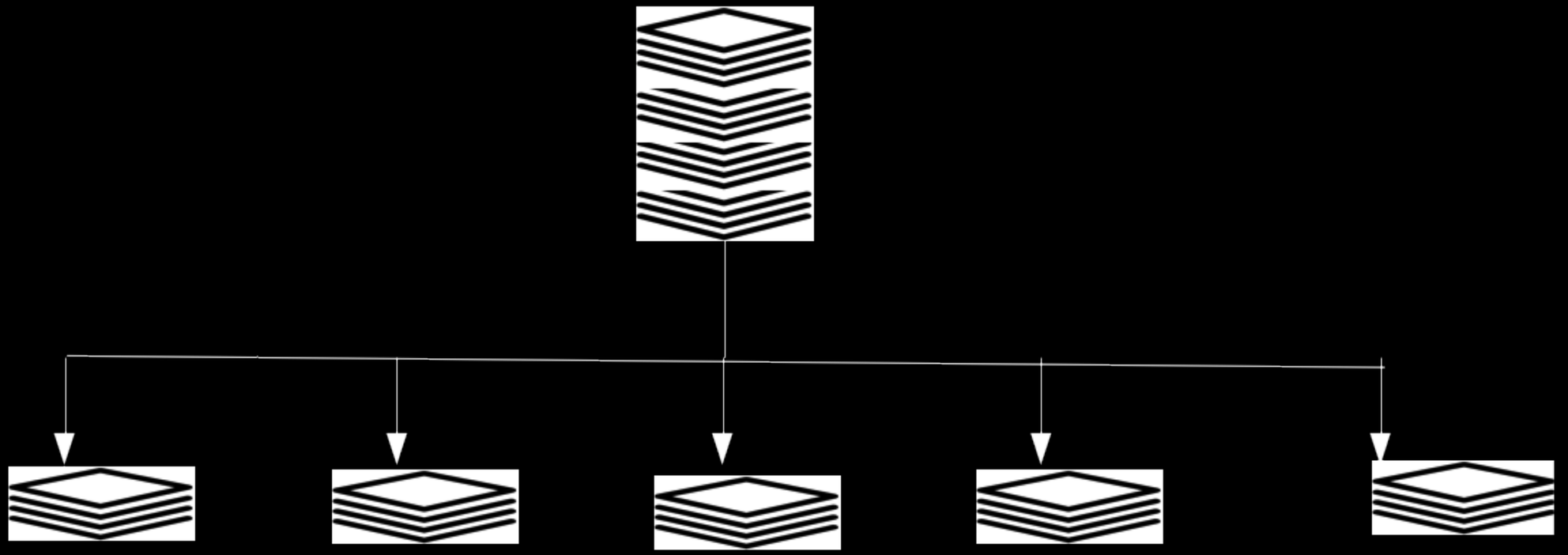
# Scenario

## Grading Homework

- The professor has a team of eager TAs to help grade assignments.
- To simplify, we assume that all TAs and the professor are fair graders.
- We assume that they all would grade the same paper identically.

# Distributed Batches

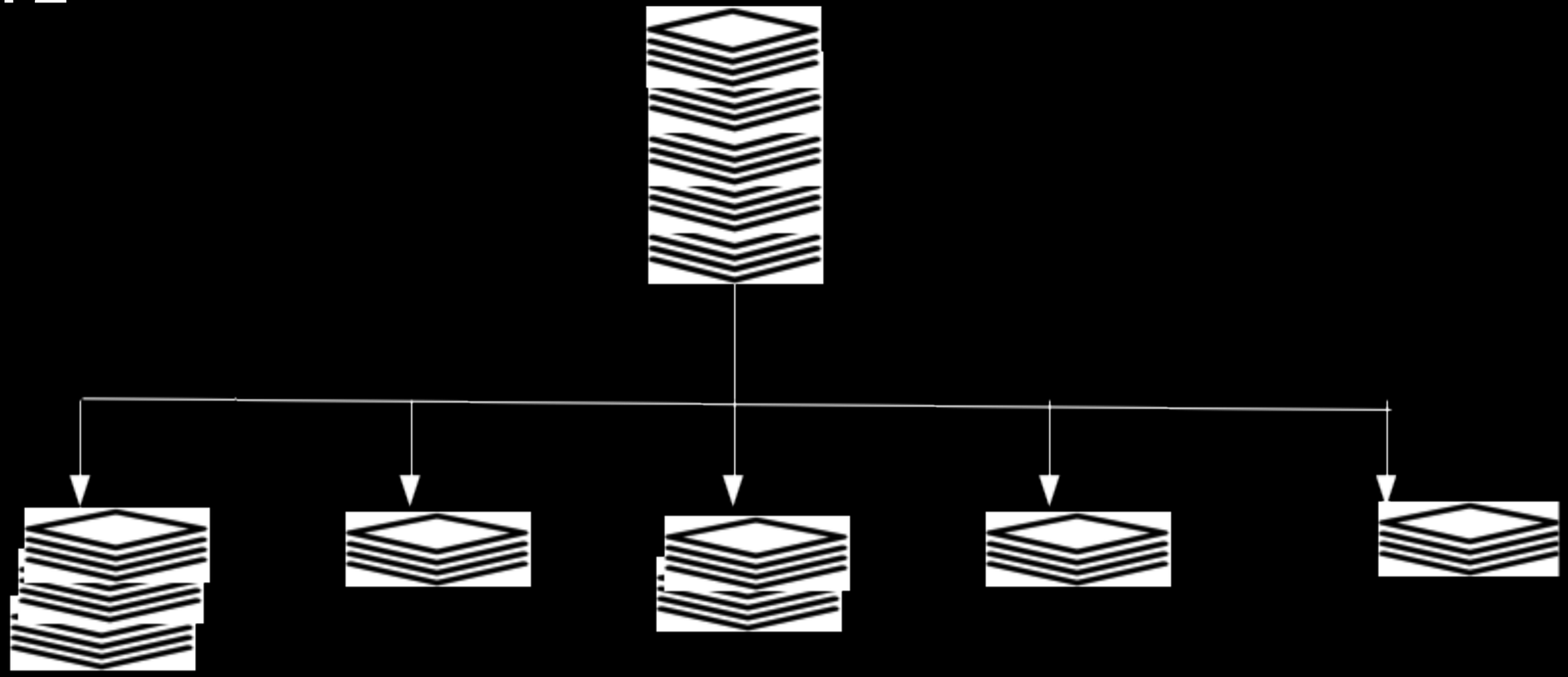
## Solution 1



- What are the advantages?
- What could go wrong?

# Balanced Loads

## Solution 2

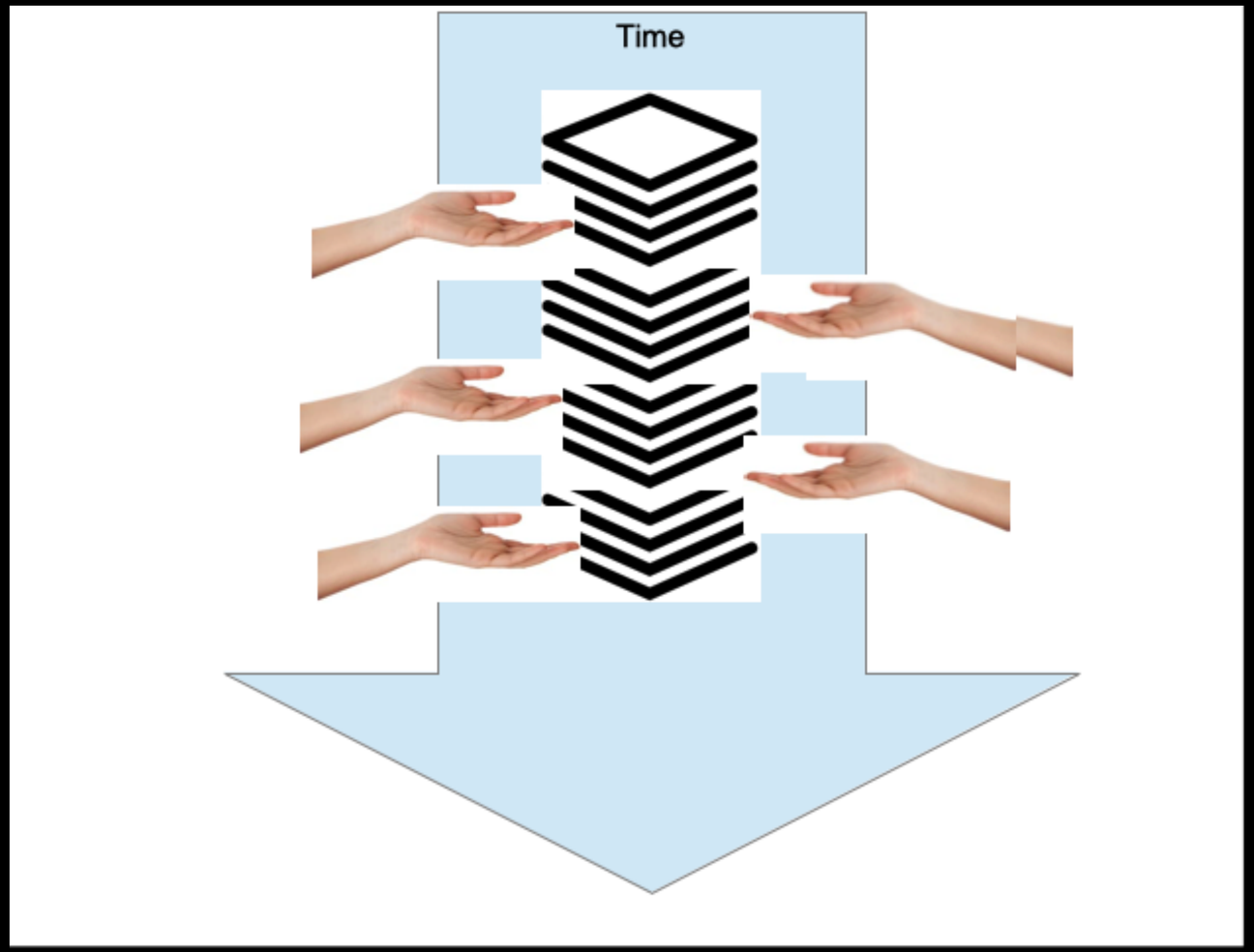


This worker can manage 3 times the average workload in a unit of time.

This worker can manage twice the average workload in a unit of time.

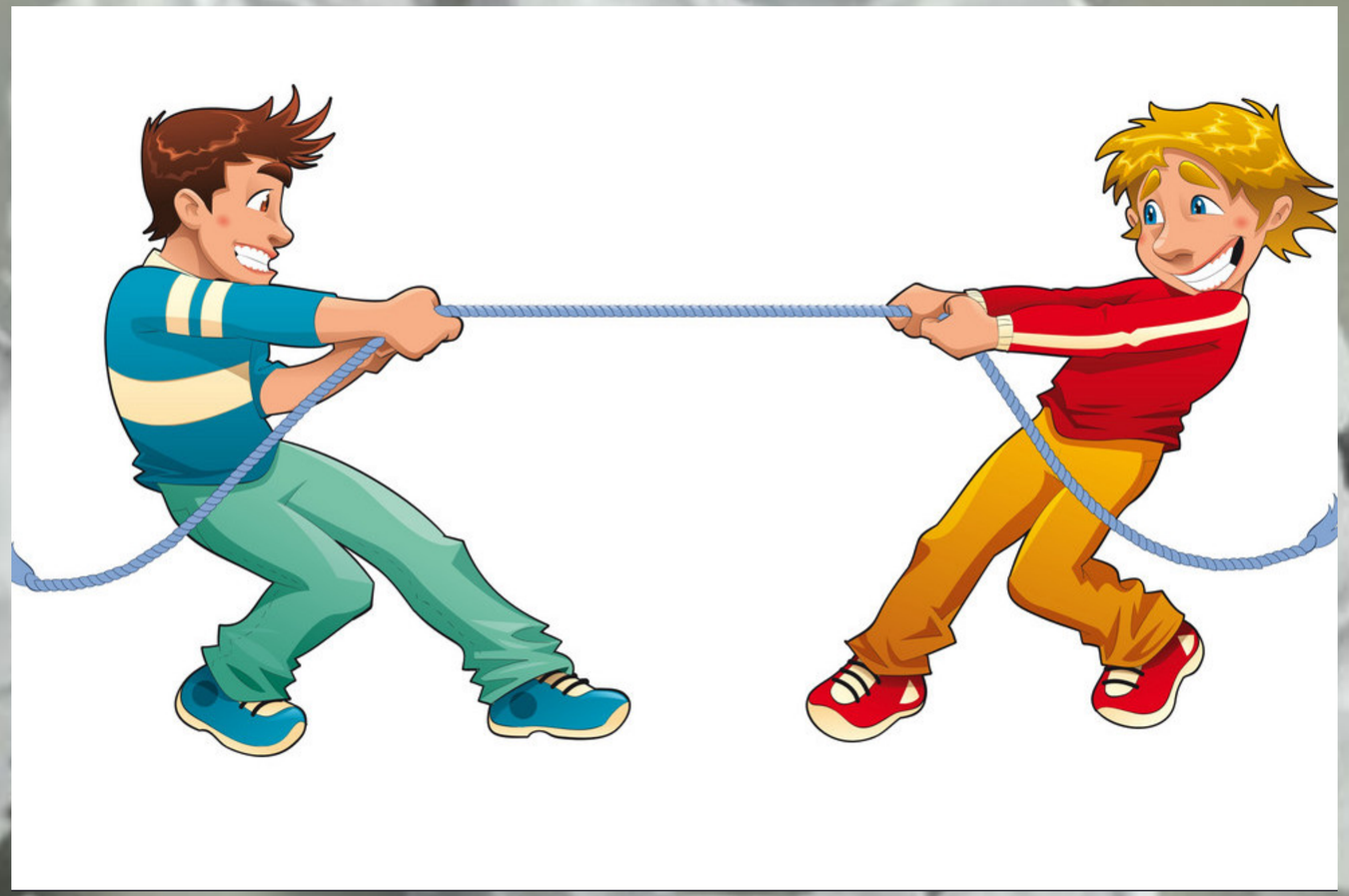
- What are the advantages?
- What are the costs?

# Worker Pool
## Solution 3



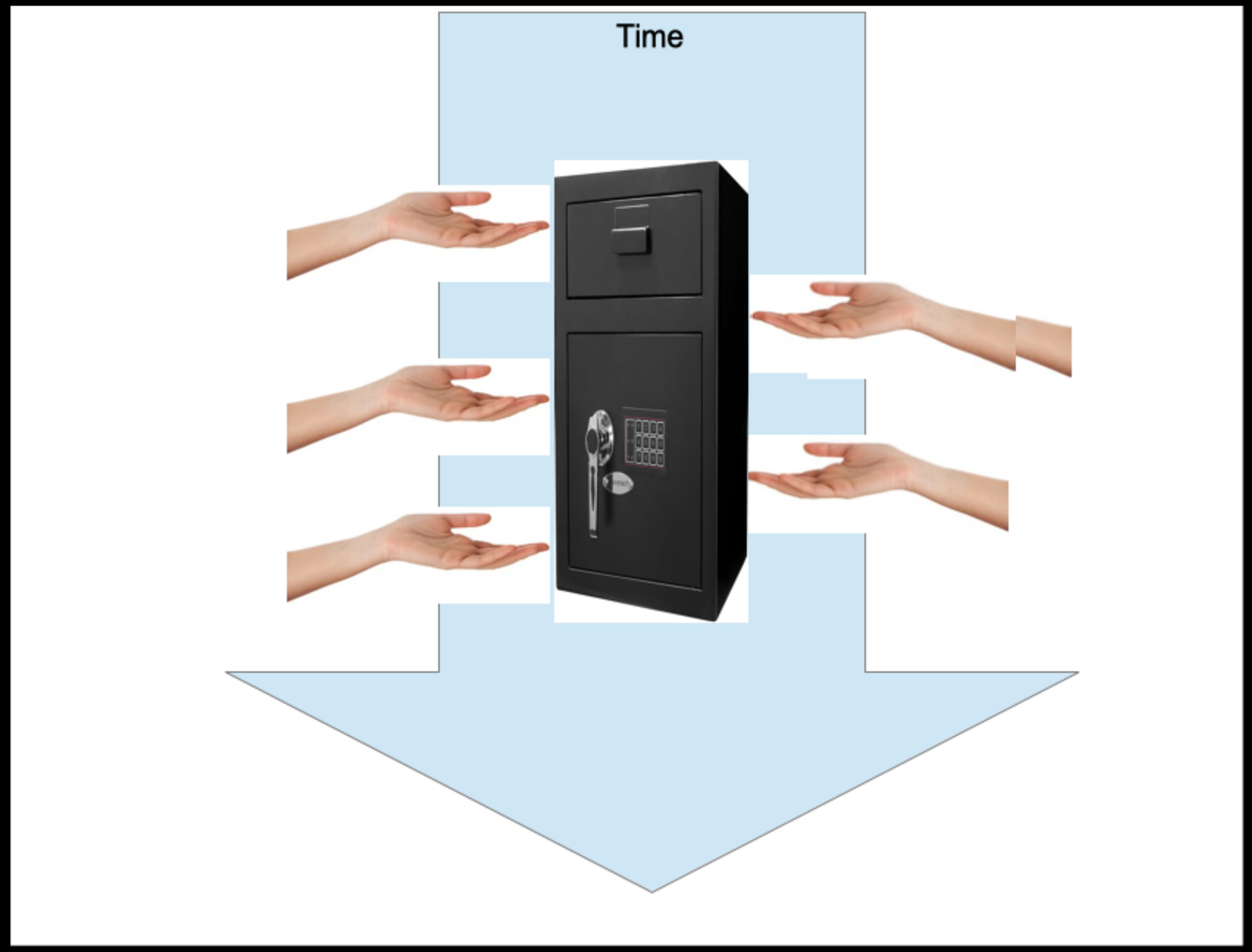

- What are the advantages?
- What could go wrong?

# Contention



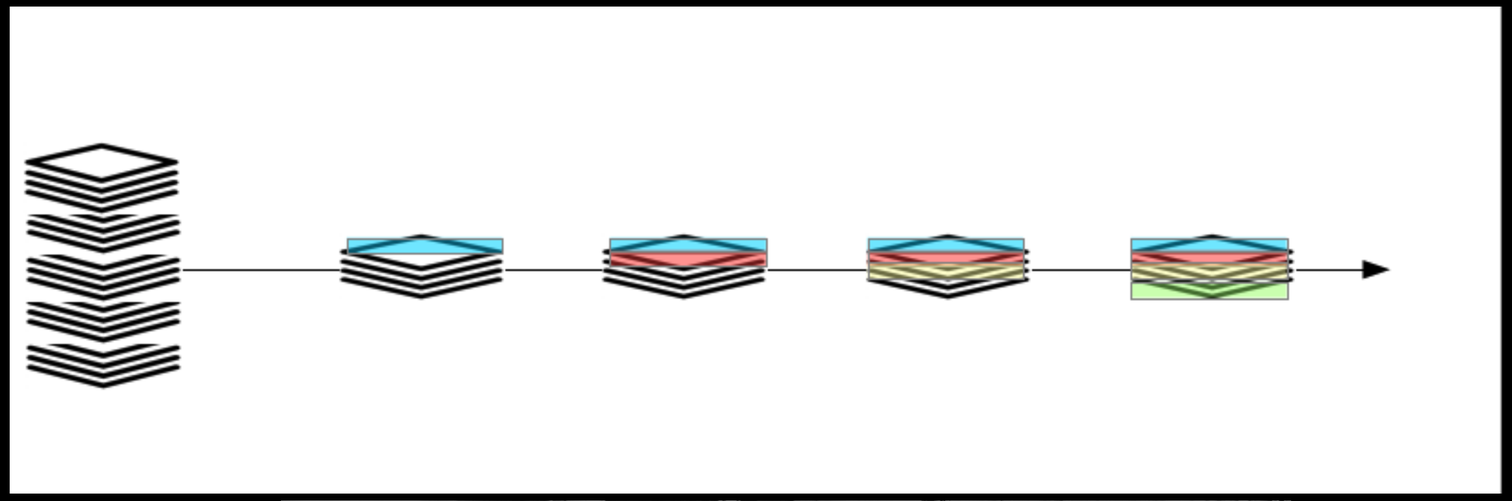
# Worker Pool with Locked Queue

## Solution 4




- What are the advantages?
- What are the costs?

# Pipeline
## Solution 5



- What are the advantages?
- What could go wrong?

# Problem

## Merging *k* Sorted Lists

- Input:
  - $n$ integers $i_0, i_1, \ldots, i_{n-1}$—for simplicity, $i_a \geq 0 \ \forall a \in [0,n)$.
  - These are distributed across $k$ *sorted* arrays or lists of size $n_0, n_1, \ldots, n_{k-1}$, so that $n = \sum_{j=0}^{k-1} n_j$, and each array or list holds integers $i_{j,0}, i_{j,1}, \ldots, i_{j,(n_j-1)}$.
- Output: One sorted list holding all $n$ integers.
- How would you solve this problem in serial (sequential, non-parallel) code?

# Example

- Input $(k = 4)$
  $l1 = [1,21,27,49,64]$
  $l2 = [7,20,47,56]$
  $l3 = [19,26,70]$
  $l4 = [1,29,50,77,87,96]$
- Output
  $out = [1,1,7,19,20,21,26,27,29,47,49,50,56,64,70,77,87,96]$
- What is the simplest (brute-force) solution that comes to mind?
  - Can we do better?

# Brute Force

```python
from typing import List


def merge_lists(inputs: List[List[int]]) -> List[int]:
    """
    Merges an arbitrary number of sorted lists of integers
    into a single sorted list of integers.

    :type inputs: list of sorted lists of integers
    :return sorted list of integers
    """
    output = []
    while len(inputs):
        next_item = inputs[0][0]
        min_index = 0
        for i in range(1, len(inputs)):
            if inputs[i][0] < next_item:
                next_item = inputs[i][0]
                min_index = i
        output.append(next_item)
        inputs[min_index].pop(0)
        if not len(inputs[min_index]):
            inputs.pop(min_index)
    return output
```

Time complexity?

# Brute Force

```python
from typing import List


def merge_lists(inputs: List[List[int]]) -> List[int]:
    """
    Merges an arbitrary number of sorted lists of integers
    into a single sorted list of integers.

    :type inputs: list of sorted lists of integers
    :return sorted list of integers
    """
    output = []
    while len(inputs):
        next_item = inputs[0][0]
        min_index = 0
        for i in range(1, len(inputs)):
            if inputs[i][0] < next_item:
                next_item = inputs[i][0]
                min_index = i
        output.append(next_item)
        inputs[min_index].pop(0)
        if not len(inputs[min_index]):
            inputs.pop(min_index)
    return output
```

Time complexity $c(n) \in O(nk)$

# Using a Priority Queue

```python
import heapq
from typing import List


def merge_lists(inputs: List[List[int]]) -> List[int]:
    """
    Merges an arbitrary number of sorted lists of integers
    into a single sorted list of integers.

    :type inputs: list of sorted lists of integers
    :return sorted list of integers
    """
    output = []
    first_items = []
    for index, input_l in enumerate(inputs):
        first_items.append((input_l[0], index))
    heapq.heapify(first_items)
    num_lists = len(inputs)
    while num_lists:
        next_item, min_index = heapq.heappop(first_items)
        output.append(next_item)
        inputs[min_index].pop(0)
        if len(inputs[min_index]):
            heapq.heappush(first_items, (inputs[min_index][0], min_index))
        else:
            num_lists -= 1
    return output
```

Time complexity?

# Using a Priority Queue

```python
import heapq
from typing import List


def merge_lists(inputs: List[List[int]]) -> List[int]:
    """
    Merges an arbitrary number of sorted lists of integers
    into a single sorted list of integers.

    :type inputs: list of sorted lists of integers
    :return sorted list of integers
    """
    output = []
    first_items = []
    for index, input_l in enumerate(inputs):
        first_items.append((input_l[0], index))
    heapq.heapify(first_items)
    num_lists = len(inputs)
    while num_lists:
        next_item, min_index = heapq.heappop(first_items)
        output.append(next_item)
        inputs[min_index].pop(0)
        if len(inputs[min_index]):
            heapq.heappush(first_items, (inputs[min_index][0], min_index))
        else:
            num_lists -= 1
    return output
```

Time complexity $c(n) \in O(n \log k)$

# Dividing and Conquering

## Helper Function

```
def merge_two_lists(list_a: List[int], list_b: List[int]) ->
List[int]:
    output = []
    while len(list_a) or len(list_b):
        if len(list_b) and (not len(list_a) or list_b[0] <
list_a[0]):
            output.append(list_b[0])
            list_b.pop(0)
        else:
            output.append(list_a[0])
            list_a.pop(0)
    return output
```

Time complexity?

# Dividing and Conquering

## Main Function

```python
def merge_lists(inputs: List[List[int]]) -> List[int]:
    """
    Merges an arbitrary number of sorted lists of integers
    into a single sorted list of integers.

    :type inputs: list of sorted lists of integers
    :return sorted list of integers
    """
    num_lists = len(inputs)
    leap = 1
    while leap < num_lists:
        i = 0
        while i < num_lists:
            if i + leap < num_lists:
                inputs[i] = merge_two_lists(inputs[i], inputs[i + leap])
            i += leap * 2
        leap *= 2
    return inputs[0]
```

Time complexity?

# Dividing and Conquering

## Main Function

```python
def merge_lists(inputs: List[List[int]]) -> List[int]:
    """
    Merges an arbitrary number of sorted lists of integers
    into a single sorted list of integers.

    :type inputs: list of sorted lists of integers
    :return sorted list of integers
    """
    num_lists = len(inputs)
    leap = 1
    while leap < num_lists:
        i = 0
        while i < num_lists:
            if i + leap < num_lists:
                inputs[i] = merge_two_lists(inputs[i], inputs[i + leap])
            i += leap * 2
        leap *= 2
    return inputs[0]
```

Time complexity $c(n) \in O(n \log k)$

# Parallel Helper Function

```
from multiprocessing import Process, Value, Array

def merge_two_lists(list_a: List[int], list_b: List[int], result: Array):
    i = j = k = 0
    end_a = end_b = False
    while i < len(list_a) or j < len(list_b):
        if end_b or (not end_a and list_a[i] < list_b[j]):
            result[k] = list_a[i]
            i += 1
            if i == len(list_a):
                end_a = True
        else:
            result[k] =list_b[j]
            j += 1
            if j == len(list_b):
                end_b = True
        k += 1
```

# Parallel Merge Function

```
def merge_lists(inputs: List[List[int]]) -> List[int]:
   num_lists = len(inputs)
    leap = 1
    while leap < num_lists:
        i = 0
        arrays = [None] * num_lists
        processes = [None] * num_lists
        while i < num_lists:
            if i + leap < num_lists:
                arrays[i] = Array('i', len(inputs[i]) + len(inputs[i + leap]))
                processes[i] = Process(target=merge_two_lists, args=(inputs[i], inputs[i + leap], arrays[i]))
            i += leap * 2
        for process in processes:
            if process is not None:
                process.start()
        for i in range(len(processes)):
            if processes[i] is not None:
                processes[i].join()
                inputs[i] = arrays[i][:]
            processes[i] = None
            arrays[i] = None
        leap *= 2
    return inputs[0]
```