High Performance & Scalable Analytics

HPSA – 2019

Master in Big Data Analytics & Social Mining

nicola.tonellotto@isti.cnr.it

Learning Objectives

- **Knowledge**
  - High performance computing
  - Systems & storage for Big Data
  - Programming for Big Data
- **Skills**
  - Interacting with distributed resources
  - Use of distributed storage systems
  - Use of distributed data processing tools
  - How to prepare big data for HPSA
  - How to process big data with HPSA
  - How to make inference from big data through HPSA
  - How to assess results from HPSA

Tentative Timetable

- **Wednesday 17/4/19 (9-13)**
  - Course presentation
  - Project discussion
  - Parallel/distributed computing
  - Map/Reduce
  - Remote access to resources
- **Friday 19/4/19 (9-13)**
  - Distributed filesystem
  - HDFS management
  - Data analytics with Spark I
- **Thursday 2/5/19 (9-13)**
  - Data analytics with Spark II
  - Machine learning with Spark I
- **Saturday 4/5/19 (9-13)**
  - Data analytics with Spark III
  - Machine learning with Spark II
- **Thursday 9/5/19 (9-12)**
  - Use case walkthrough
- **Thursday 16/5/19 (14-15)**
  - Final exam

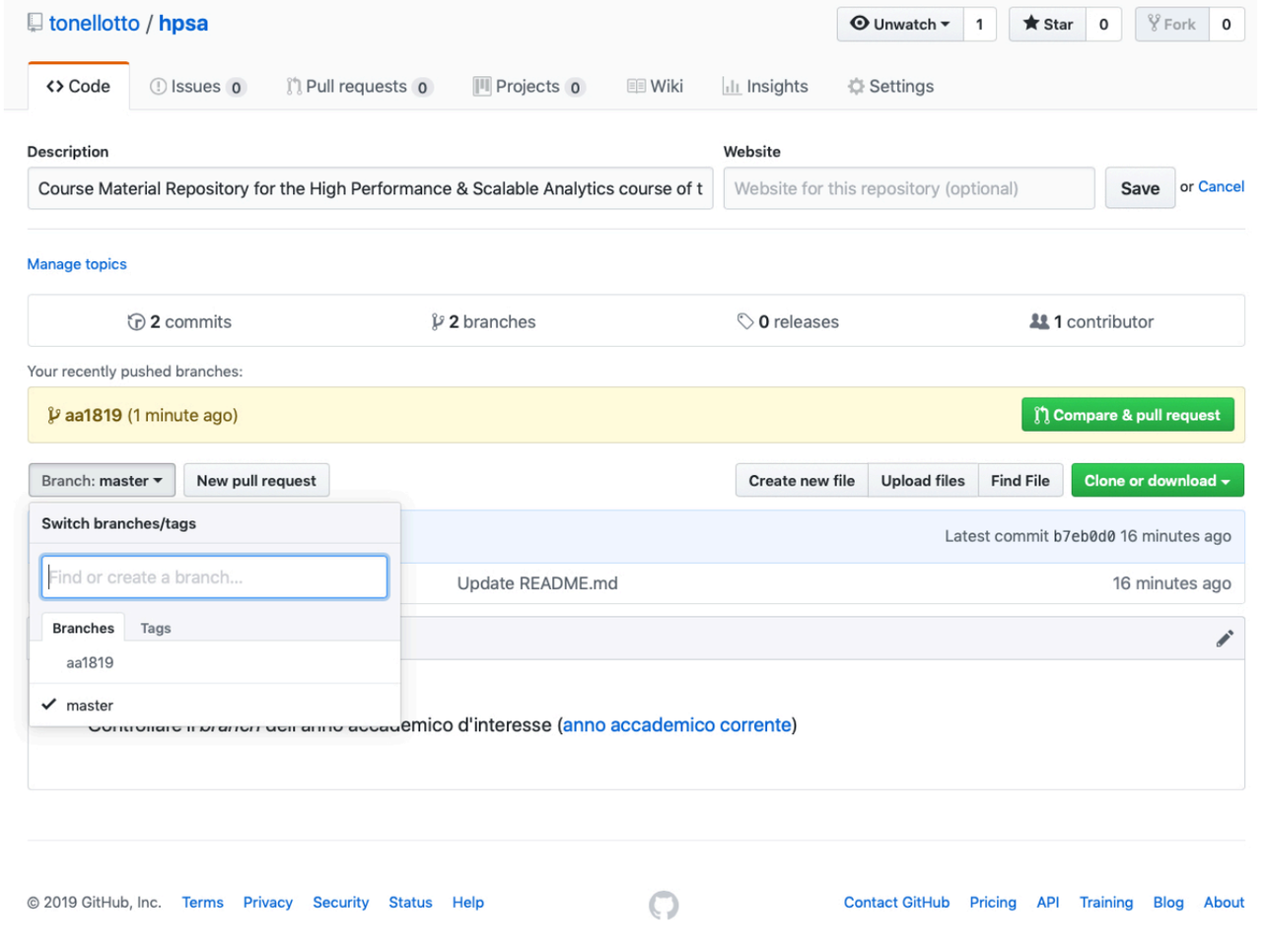
Admin I

- **Web site hosted on Github**

https://github.com/tonellotto/hpsa

- **Select the current year branch**

https://github.com/tonellotto/hpsa/tree/aa1819



Admin II

```
$> # sudo apt-get install git
$> cd $HOME
$> git clone https://github.com/tonellotto/hpsa
$> git checkout aa1819
$> ls -ltrh
```

Admin III

- **Teacher:** Nicola Tonellotto (the bad cop)
- **Tutor:** Roberto Trani (the good cop)
- **Tools:** SSH/SCP, HDFS, Spark, MLLib
- **Credits:** 2
- **Hours:** 20

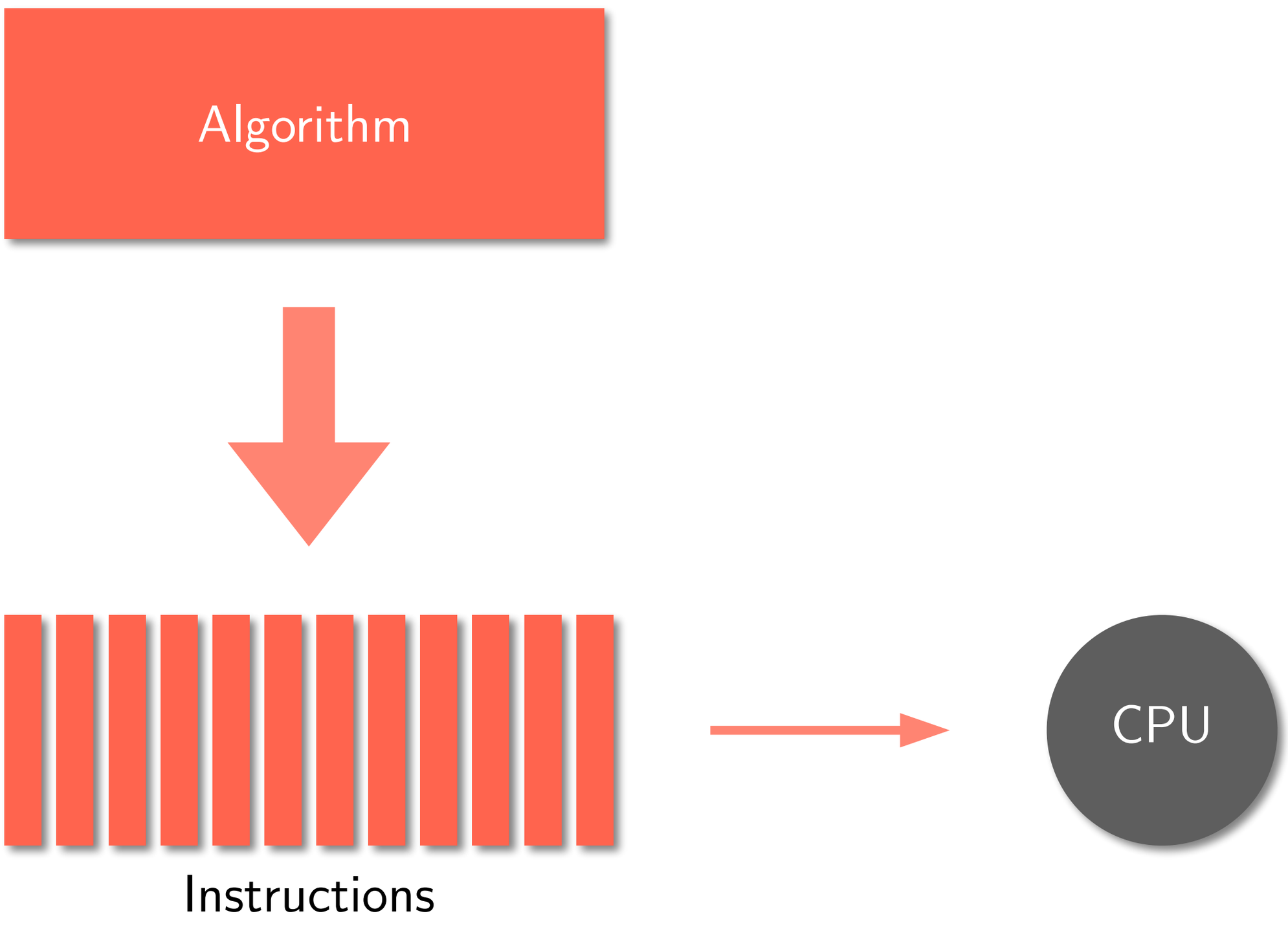
Projects

- All projects are composed by 4 phases:
  1. Data preparation
  2. Feature extraction
  3. Machine learning
  4. Results assessment
- Project topics
  - Spam classification of SMS
  - Sentiment analysis of movie reviews
  - Category identification of beers
  - Stock prediction using news
  - Customer segmentation of bank accounts
  - Customer segmentation from e-commerce transactions

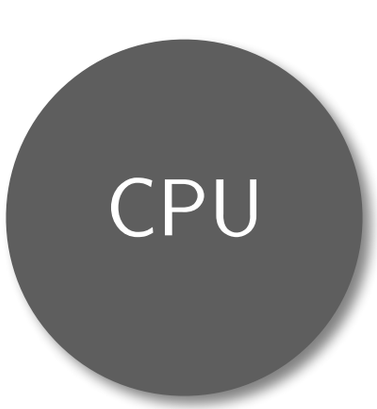
Final Grading

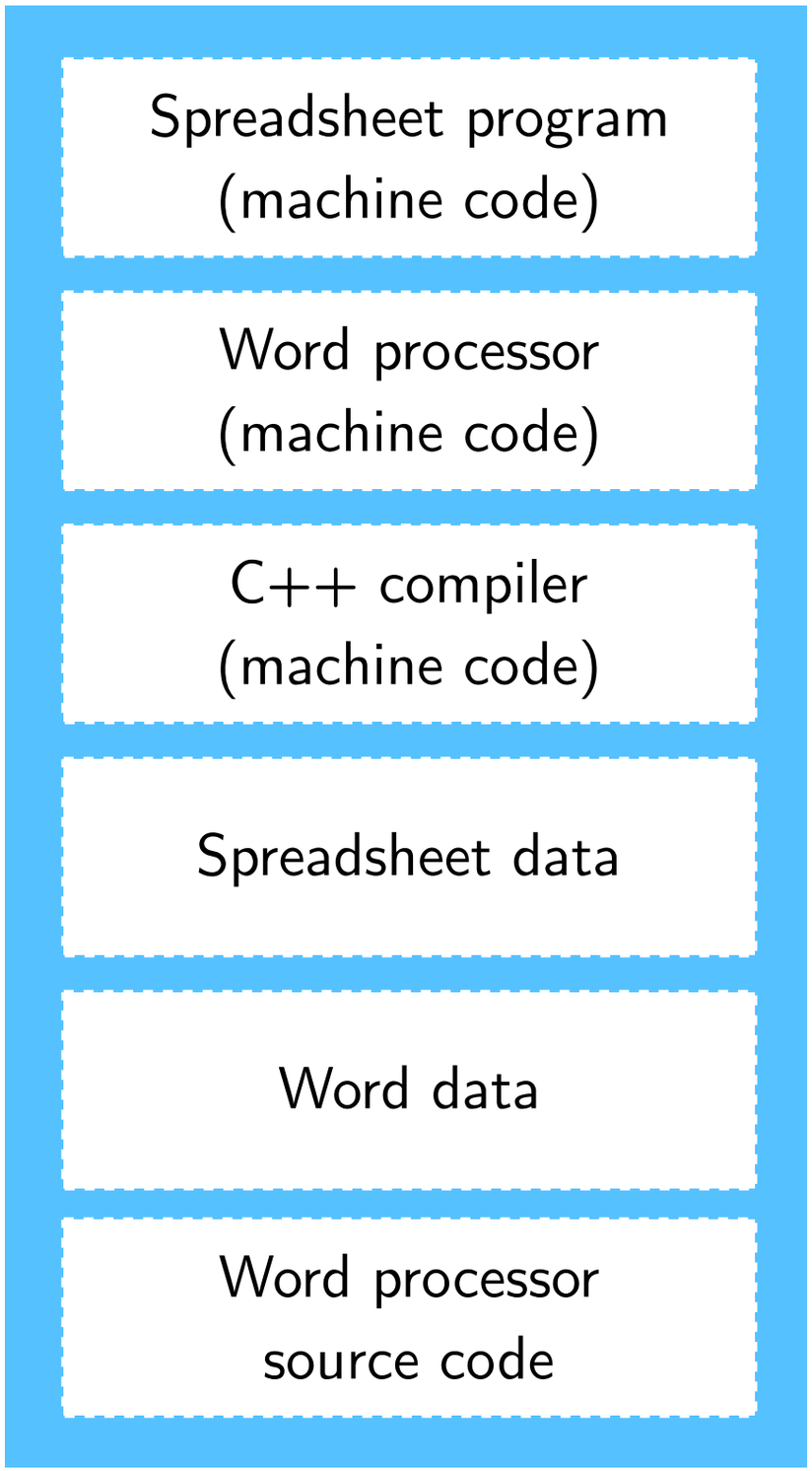
- **50% project evaluation**
  - Carried out during the open lab hours
  - 2 pages report (font 11 pt, single interline, A4 standard margins)
  - 1 self-contained python notebook
  - Group activity (max 6 per project)
  - Final version to be submitted by email to the tutor no later than 13/5/19, 11:59 AM CET
  - *Group grade*
- **50% written exam**
  - Thursday 16/5/19, 13:45 in the classroom
  - Multiple choice questions, open-ended questions, programming questions
  - *Individual grade*

Classical Computation



Stored Program Computer



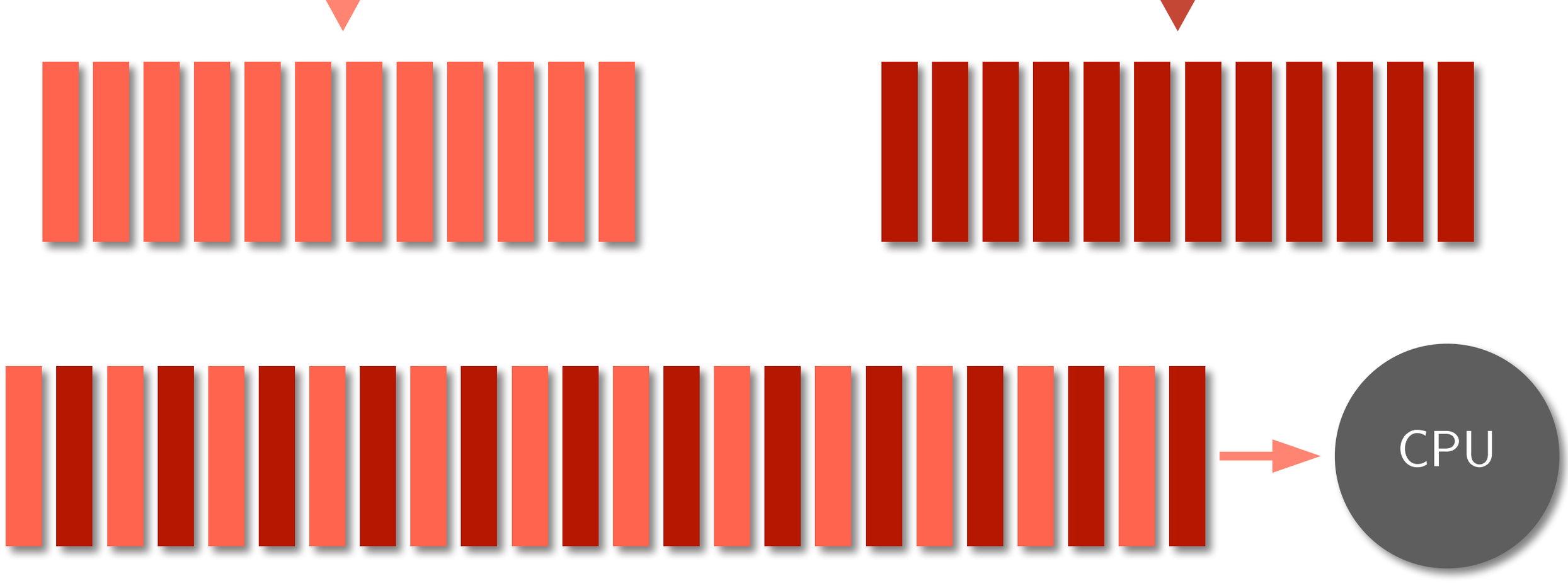


- Instructions represented as binary numbers (like data) (0010010101111001...)
- Data and instructions stored in main memory
- A CPU:
  - **FETCH**: get instructions from memory
  - **DECODE**: identify what to do
  - **EXECUTE**: perform actions

Time Sharing

Word processor

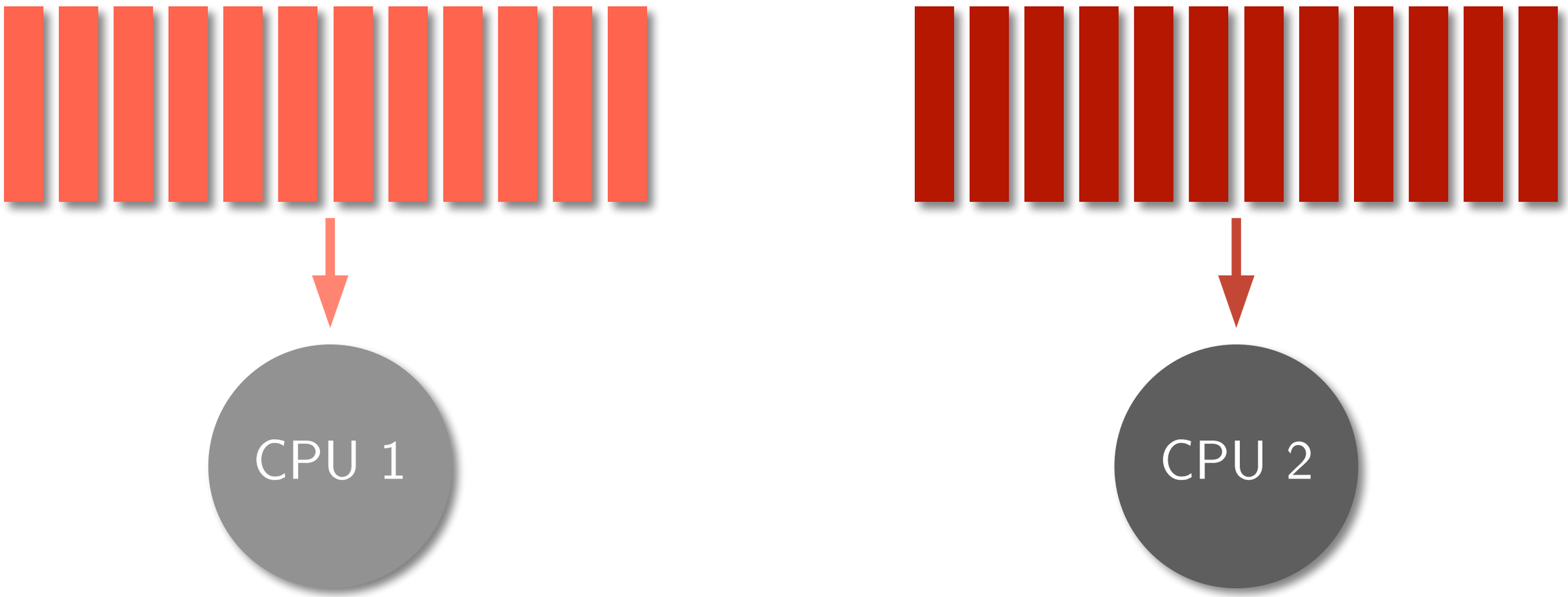
Spreadsheet



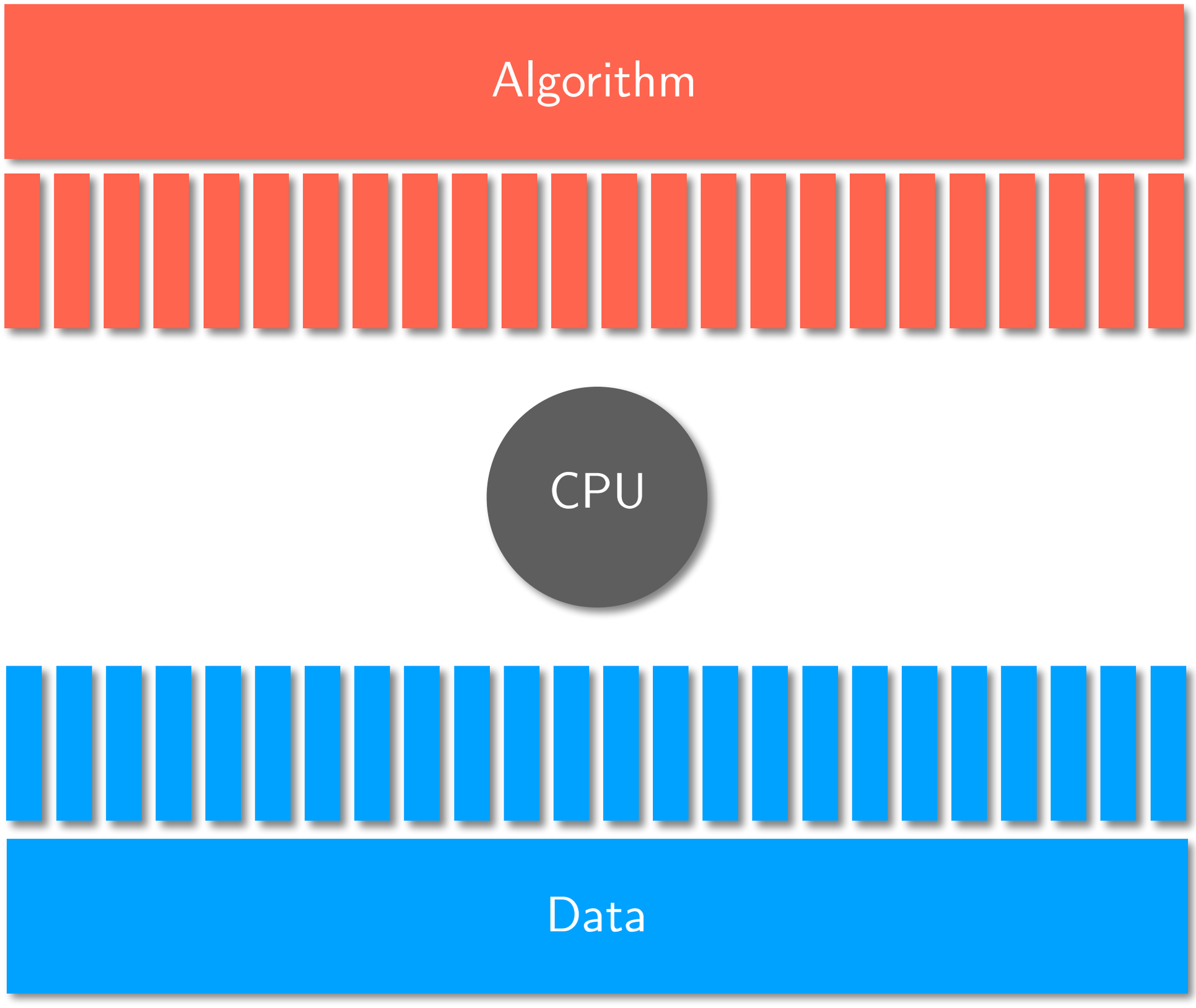
Parallel Computation

Program 1

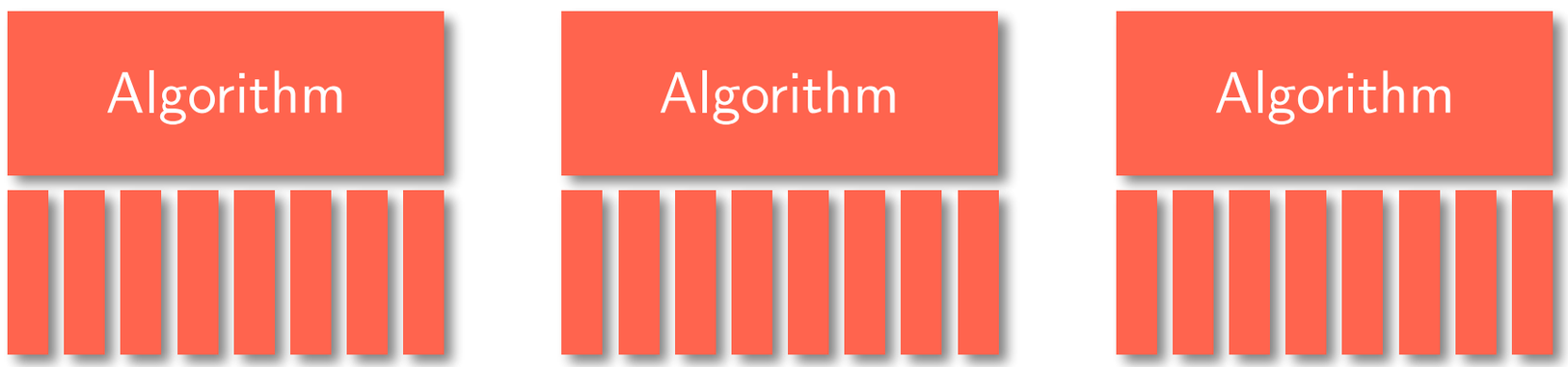
Program 2



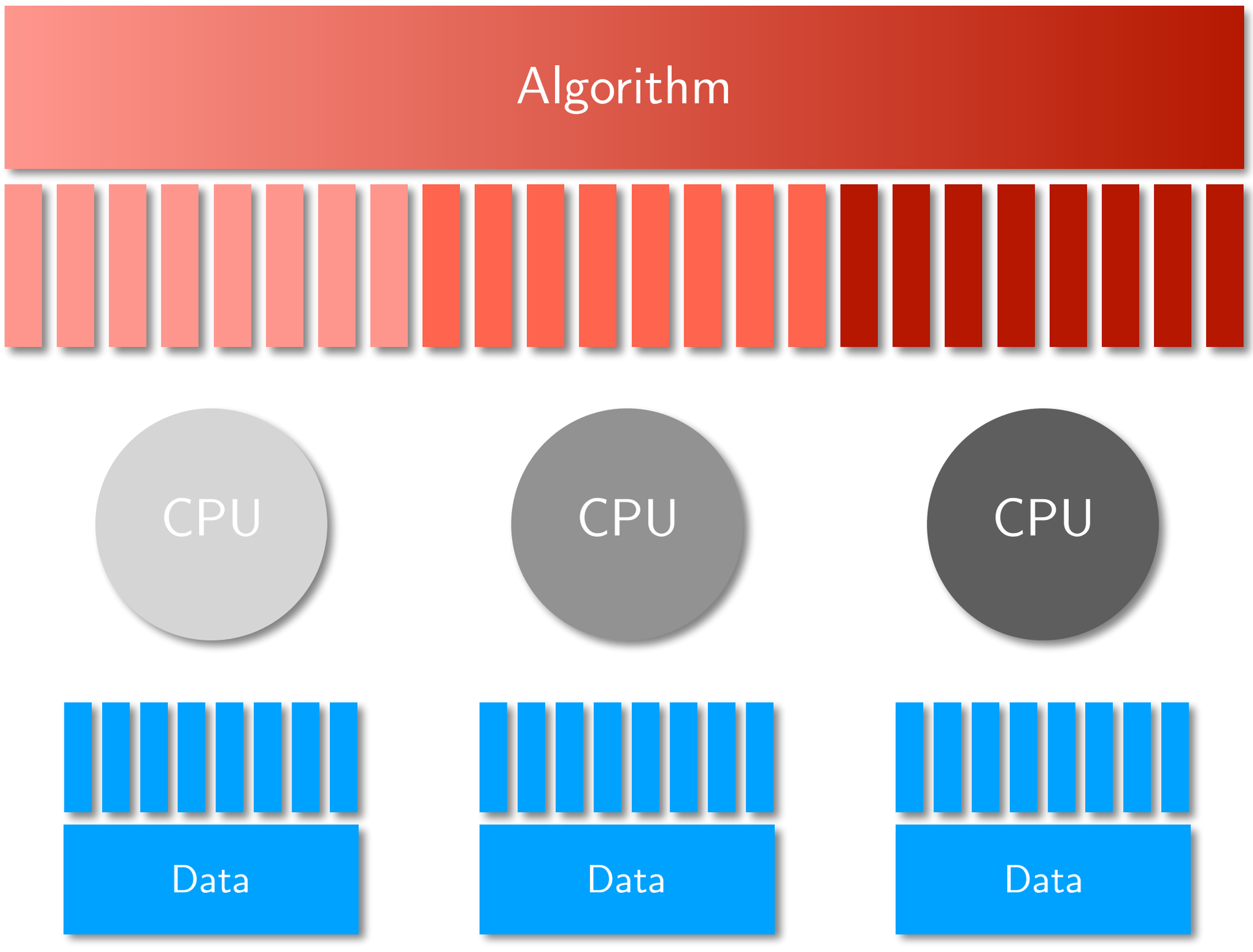
Single Instruction Single Data



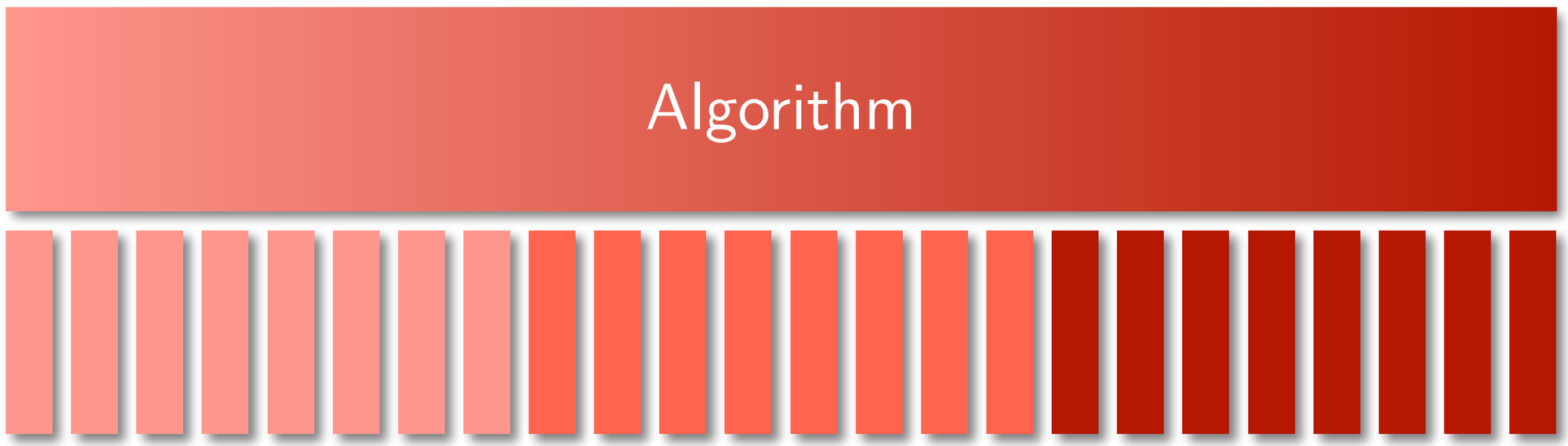
Single Instruction Multiple Data



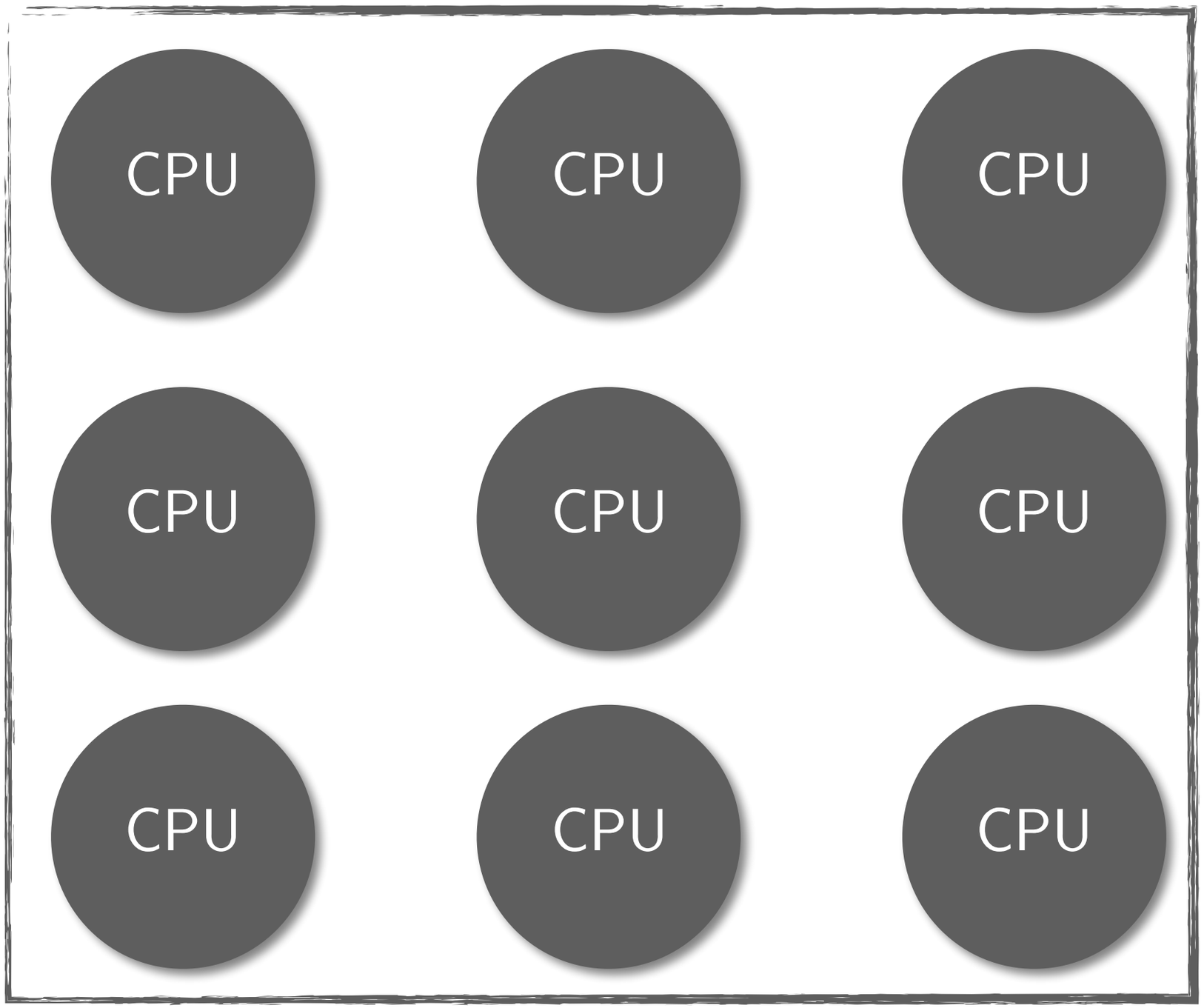
Multiple Instructions Single Data



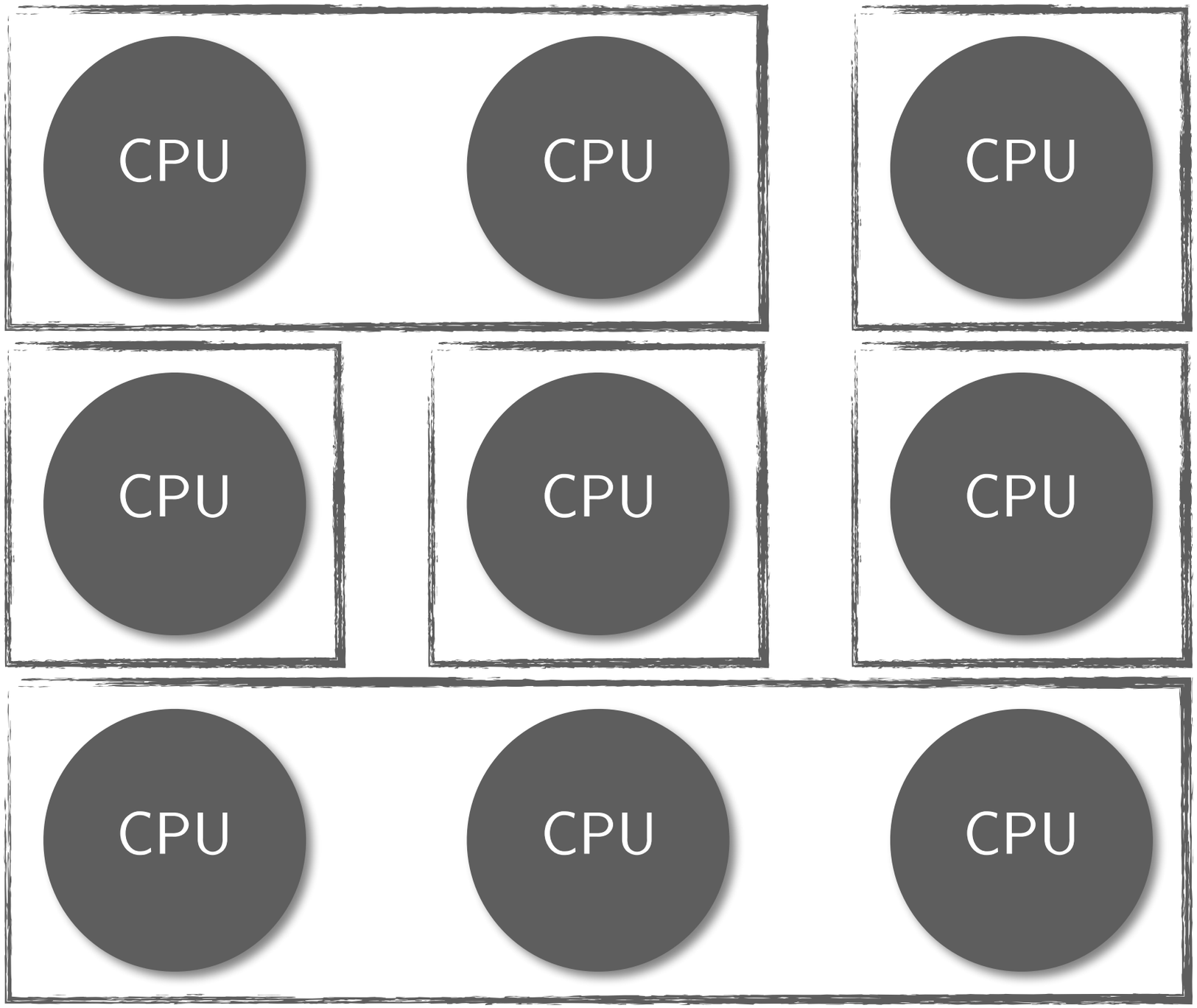
Multiple Instructions Multiple Data



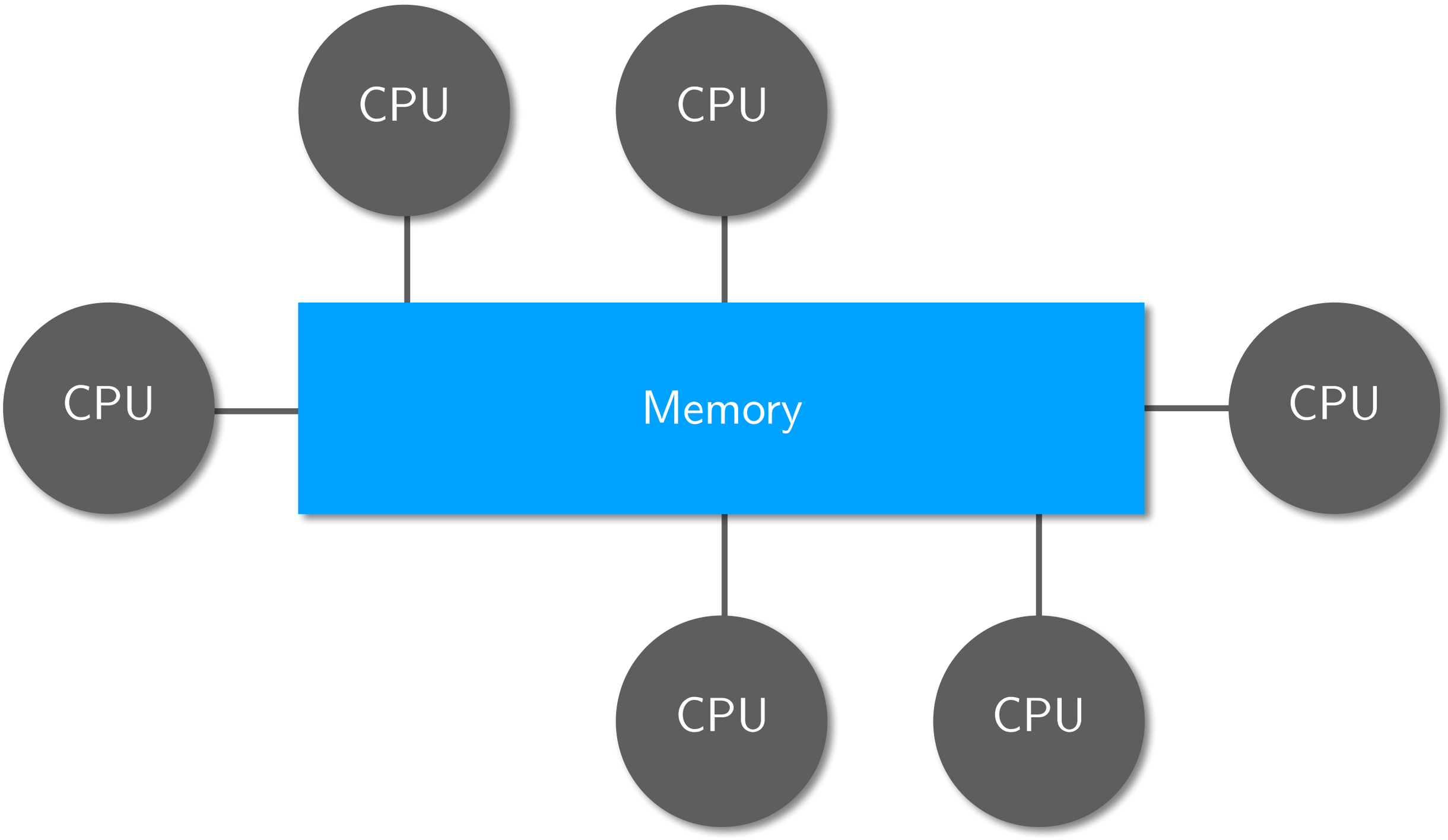
Parallel CPUs



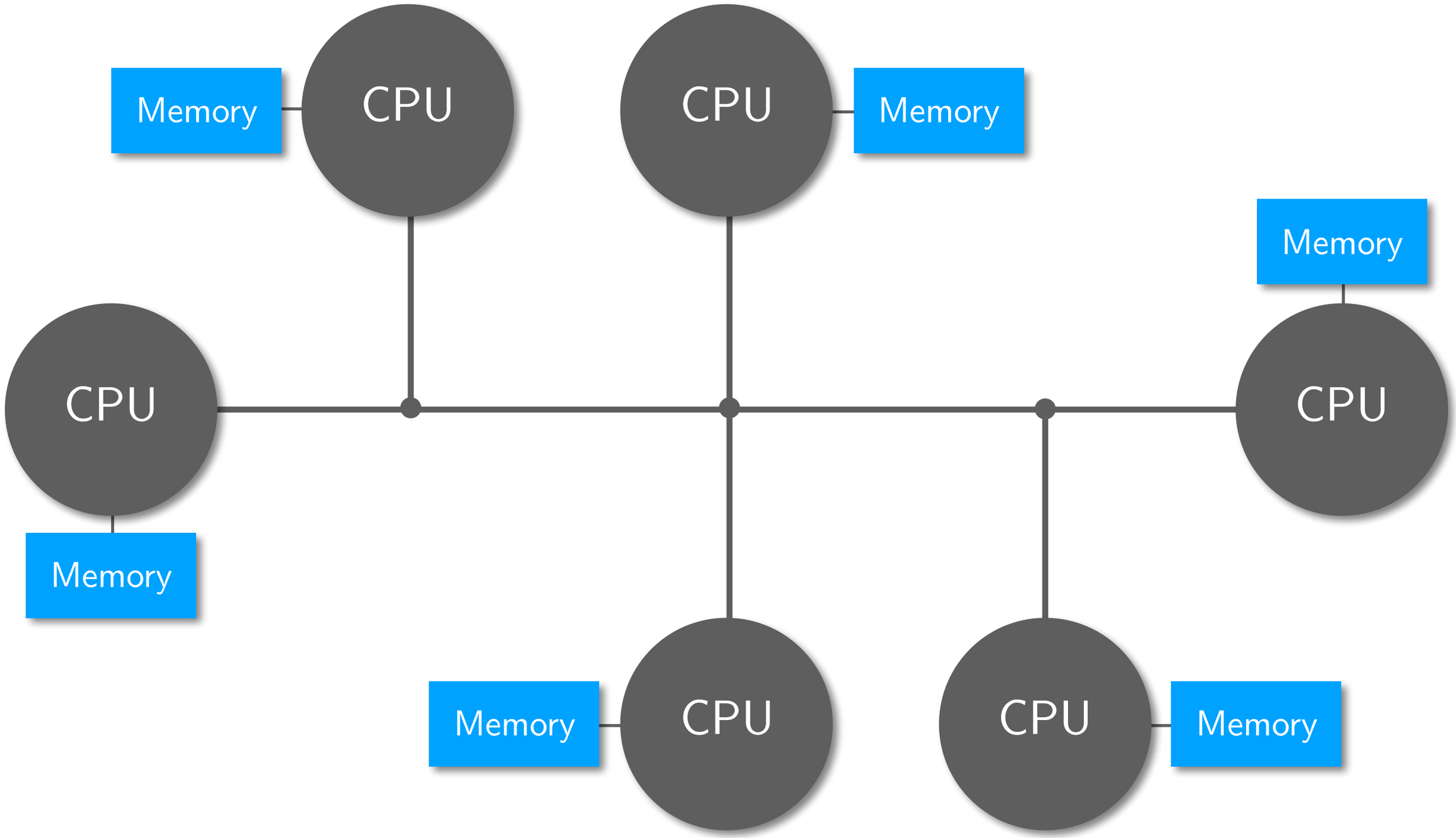
Distributed CPUs



Shared Memory



Distributed Memory



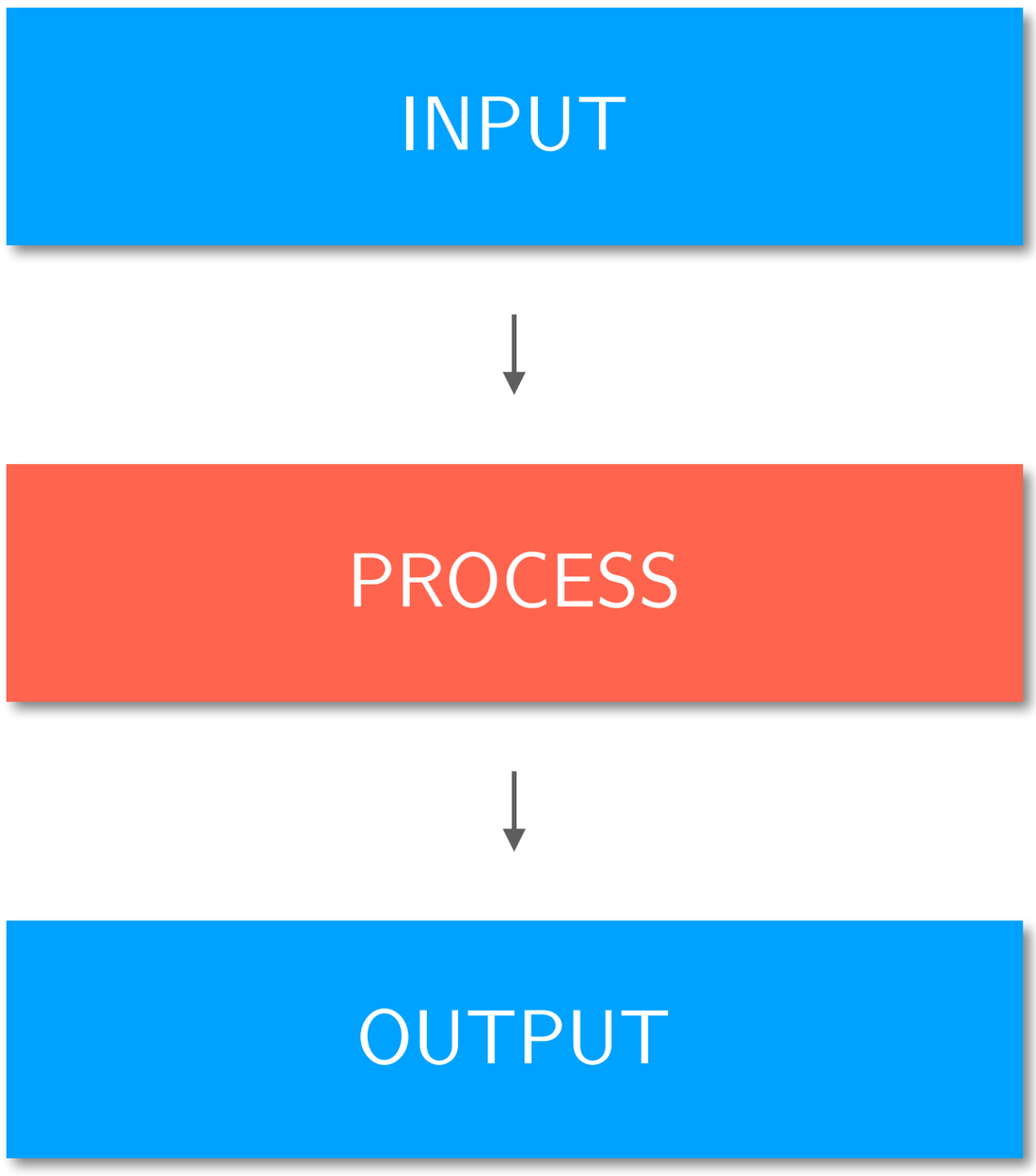
Splitting Code

- How to divide code into parallel tasks?
- How to distribute the code?
- How to coordinate the execution?
- How to load the data?
- How to store the data?
- What if more tasks than CPUs?
- What if a CPU crashes?
- What if a CPU is taking too long?
- What if the CPUs are different?
- What if we have a new (serial) code?

Splitting Data

- How to split the data?
- How to distribute the data?
- How to collect the data?
- How to merge the data?
- How to coordinate the access to the data?
- What if more data splits than tasks?
- What if tasks need to share data splits?
- What if a data split becomes unavailable?
- What if we have a new input?
- What if the data is big?

Typical Application



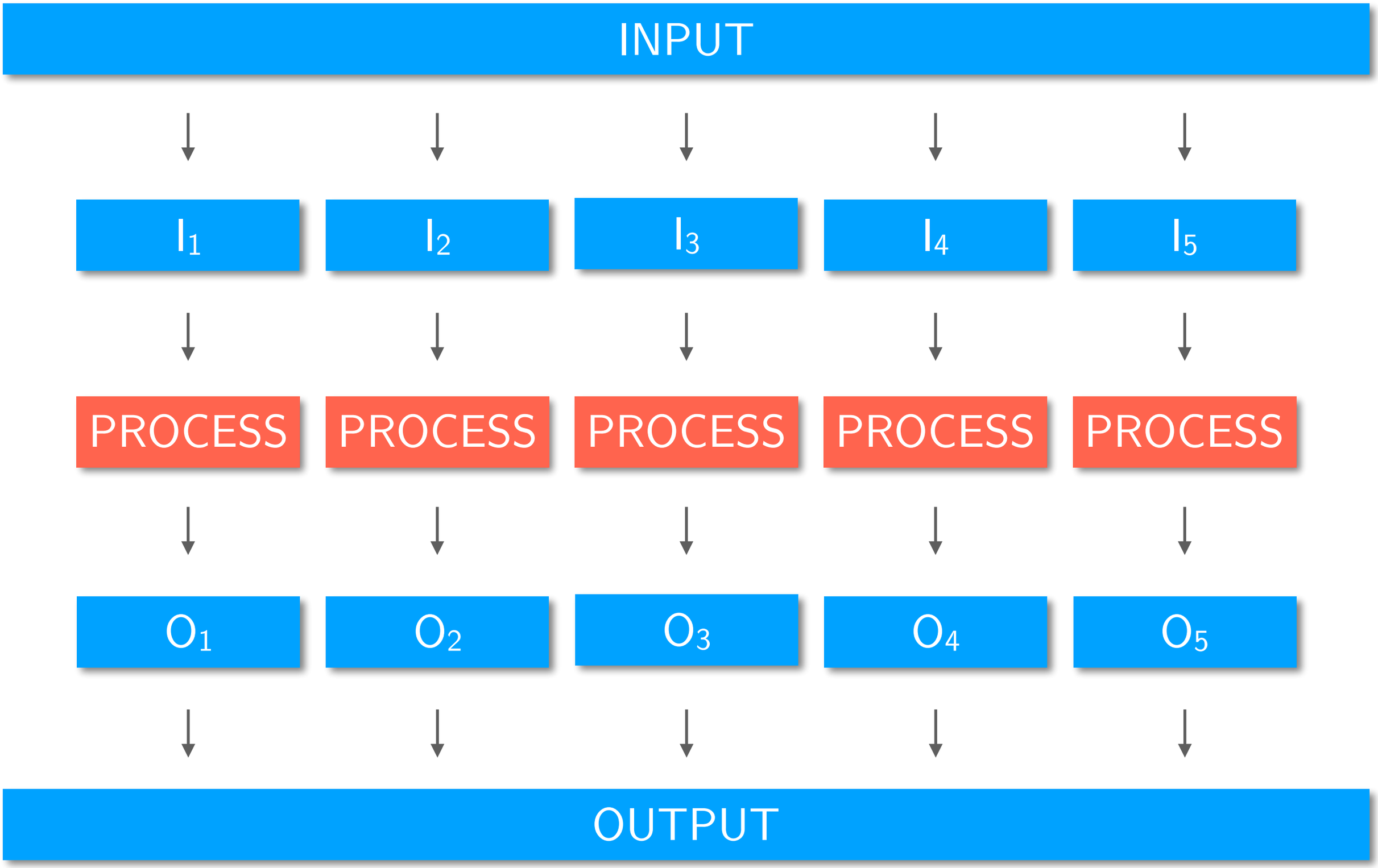
What if?

INPUT

PROCESS

OUTPUT

Divide and Conquer



Challenges

- How do we split the input?
- How do we distribute the input splits?
- How do we collect the output splits?
- How do we aggregate the output?
- How do we coordinate the work?
- What if input splits > num workers?
- What if workers need to share input/output splits?
- What if a worker dies?
- What if we have a new input?

Computing Platform

Source: https://www.pexels.com/photo/datacenter-server-449401/

Computing Platform

Source: https://www.pexels.com/photo/datacenter-server-449401/

Design Ideas

- Scale "out", not "up"
  - Avoid supercomputer, too costly
  - Use commodity machines, low costs
  - Drawback: many failures
- Move processing to the data
  - Same code, runs everywhere
  - Reduce data over the network
  - Drawback: code must be portable
- Process data sequentially, avoid random access
  - Huge data files (terabytes), not small files (megabytes)
  - Write once, read many
  - Drawback: poor support for standard file APIs
- Right level of abstraction
  - Hide implementation details from applications development
  - Write very few lines of code
  - Drawback: everything needs to fit into the abstraction

Typical Application

1. Iterate over a large number of records

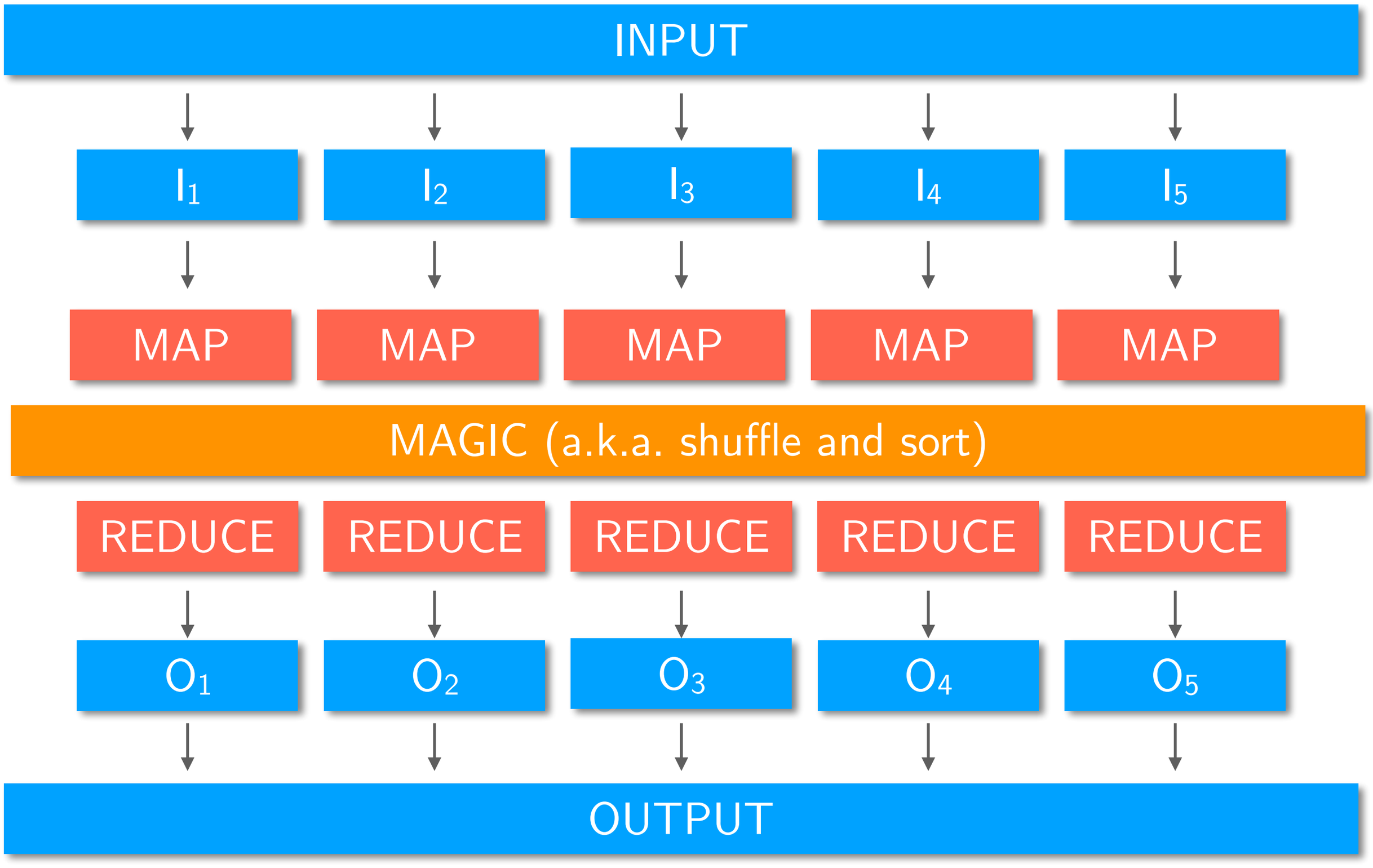
2. Extract something of interest from each

3. Shuffle and sort intermediate results

4. Aggregate intermediate results

5. Generate final output

Map Reduce Application



Programming Model

- Programmers specify **two functions**
  - *map function*: from [key, value] (1) to [key, value] (0 or more)
  - *reduce function*: from [key, list of values] (1) to [key, values] (0 or more)
- **Map** function
  - Receives as input a key-value pair
  - Produces as output a list of key-value pairs (typically 1 or more per input)
- **Reduce** function
  - Receives as input a key-list of values pair
  - Produces as output a list of key-value pairs (typically none or 1 per input)
- Both functions are **STATELESS**
- The **runtime support** handles everything else...

Wordcount Example (I)

```
class MAPPER
    method MAP(docid a, doc d)
        for all term t in doc d do
            EMIT(term t, count 1)
```

```
class REDUCER
    method REDUCE(term t, counts [c1, c2,...])
        sum ← 0
        for all count c in counts [c1, c2,...] do
            sum ← sum + c
        EMIT(term t, count sum)
```

Wordcount Example

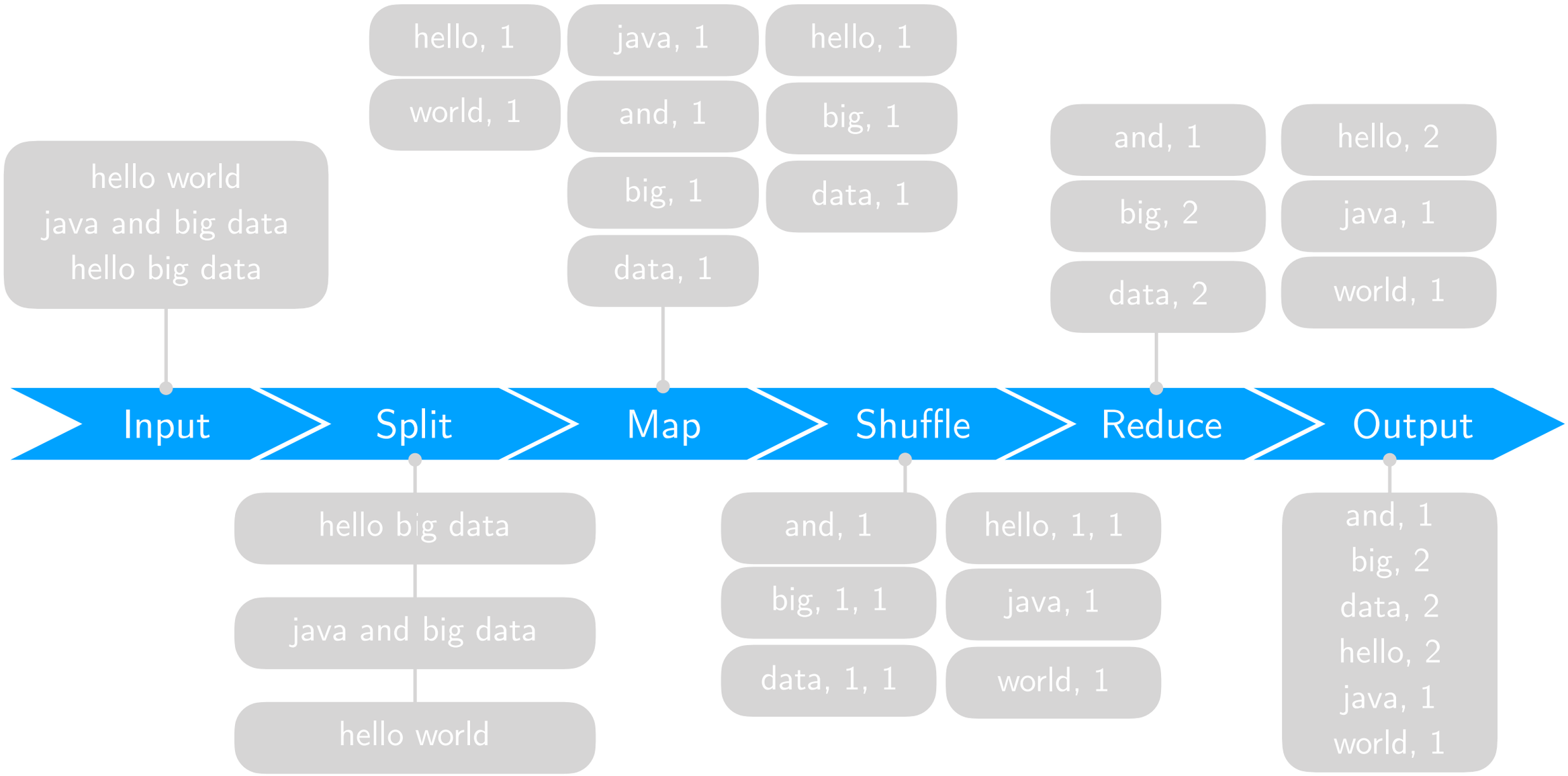


Image Data Example (I)

- Image data from **different** content providers
  - Different formats
  - Different coverages
  - Different timestamps
  - Different resolutions
  - Different exposures/tones
- **Large amount** to data to be processed
- Goal: produce data to serve a **"satellite" view** to users

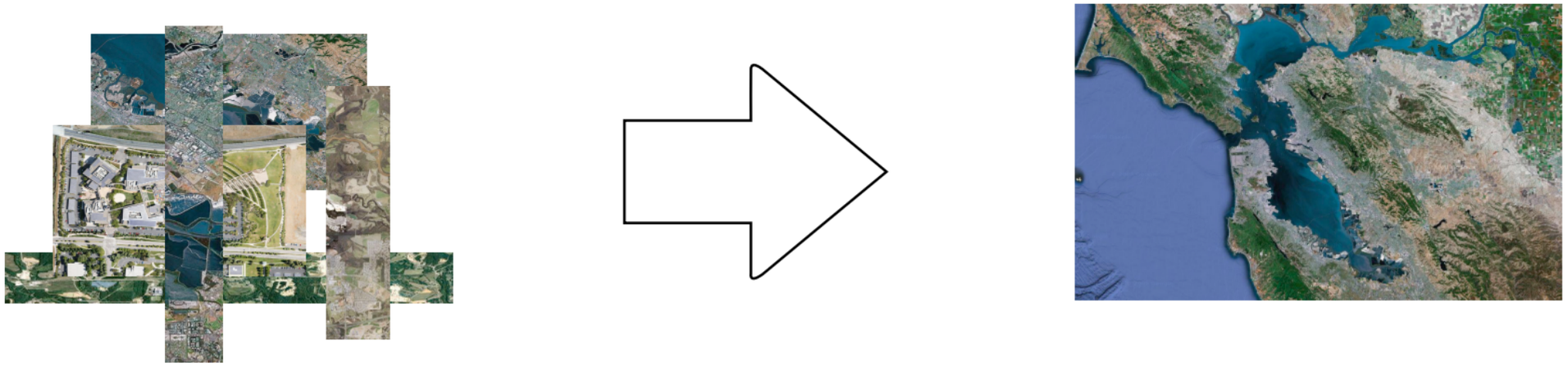
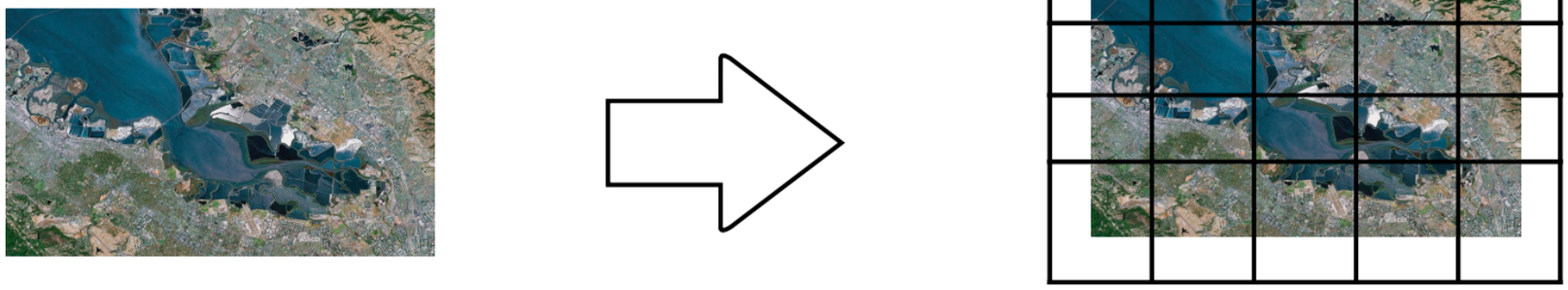
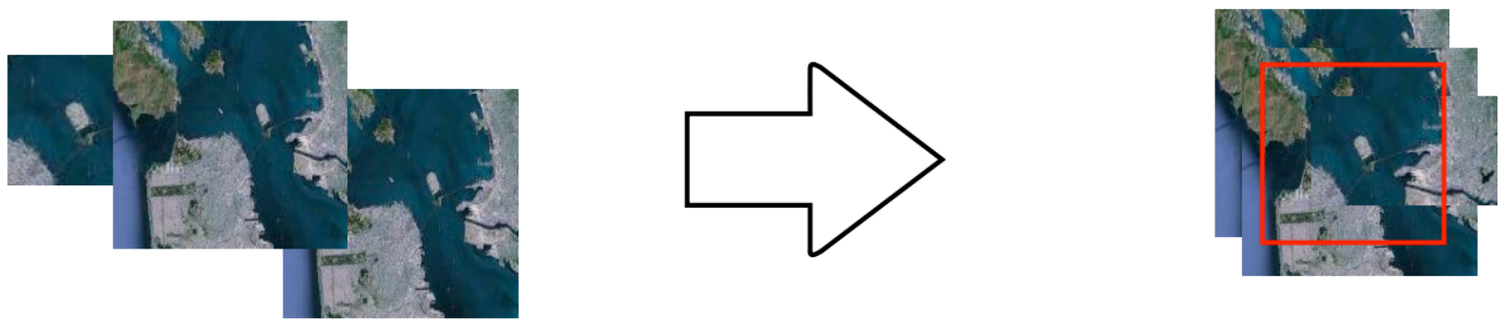


Image Data Example (II)

- Split the whole territory into "tiles" with fixed location IDs
- Split each source image according to the tiles it covers



- For a given tile, stitch contributions from different sources, based on its freshness and resolution, or other preference



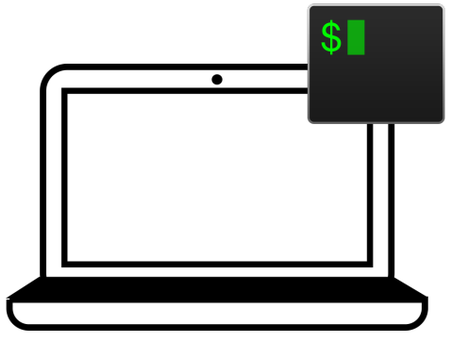
- Serve the merged imagery data for each tile, so they can be loaded into and served from an image server farm.

SSH Crash Course

- SSH = Secure Shell
- Replacement for RSH, the Unix remote shell application
  - RSH allowed one to connect to a shell on a remote machine
    - First issue: all traffic in cleartext
    - Second issue: access to a machine grants access to all machines
- Command-line connection tool
- All traffic encrypted
- Both ends authenticate themselves to the other end

SSH Picture

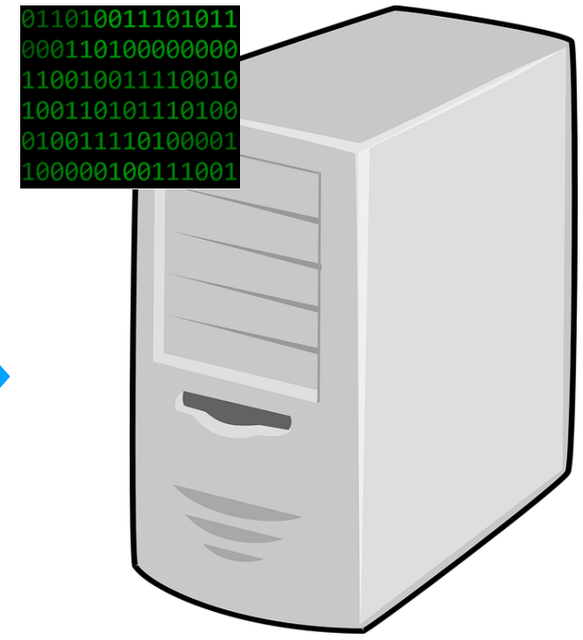
SSH client
(command line)



client host

network

SSH server
(service)



server host

SSH Basic Use

- `ssh <server_name>`
- `ssh -l <user_name> <server_name>`
- `ssh <user_name>@<server_name>`
- `ssh <server_name> <command>`
- `ssh -v <server_name>`

Secure File Copy

- SCP = Secure Copy

  `scp localfile remotemachine:/remotepath/file`

- The local and remote file specifications are of the form

  `user@host:/path/to/file`

  - If omitted, `user`, `host` and `path/to/file` directory default to local username, local hostname, and current directory, respectively.

- Prompts for authentication if needed
- All traffic encrypted
- If including wildcards (*), include the block in quotes (")