Queue sytems

Maciej Marchwiany, PhD

https://www.linkedin.com/pulse/ai-powered-physics-how-ai-revolutionizing-scientific-discovery-cain-ddbwc

Plan

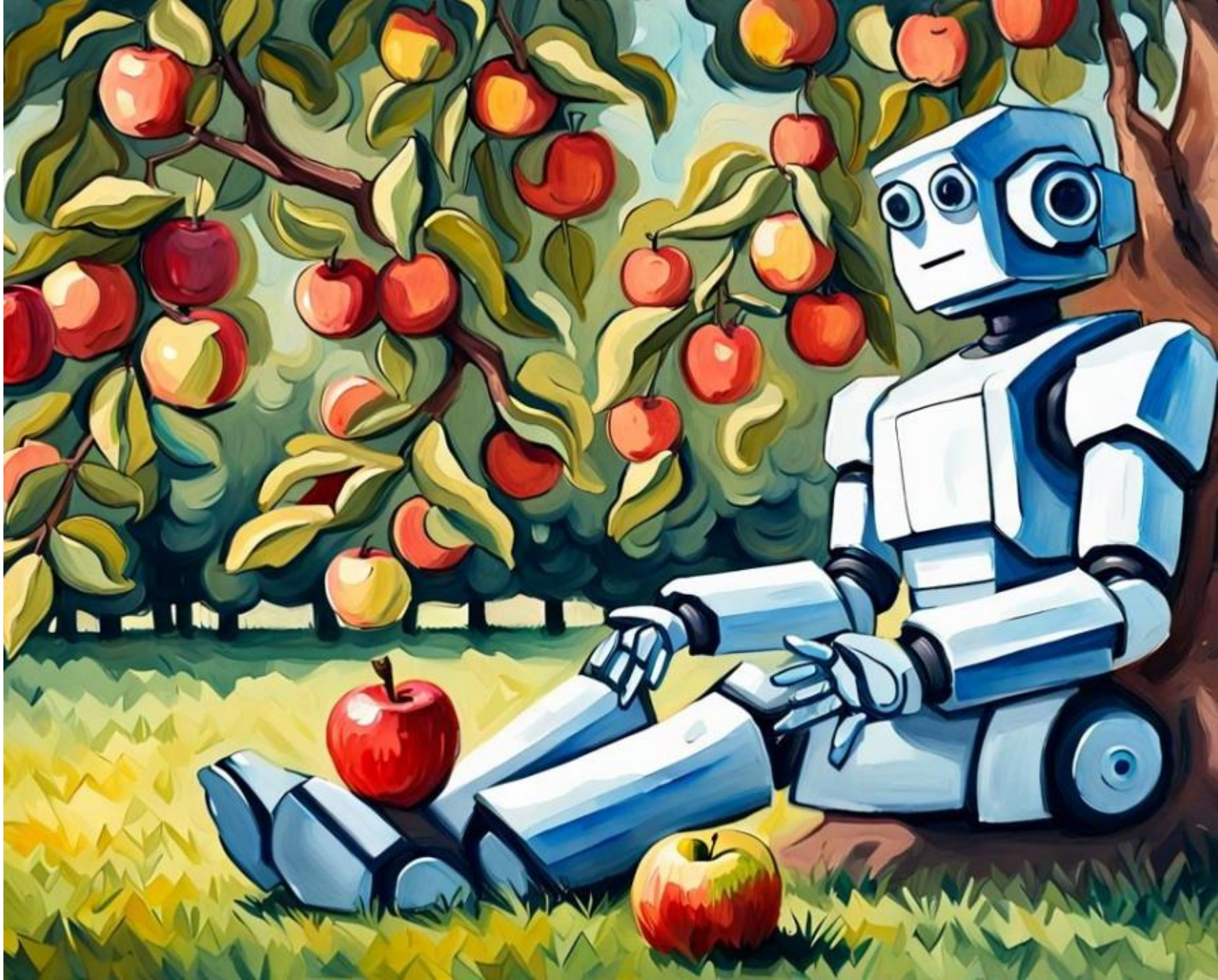
Supercomputers

Parallel compiuting

Memory models

Queue systems

Super – computers



Supercomputer

Supercomputer - a computer with a **high** level of **performance** as compared to a general-purpose computer.

Supercomputers play an important role in the field of computational science, and are used for a wide range of **computationally intensive** tasks in various fields, including:

- quantum mechanics,
- weather forecasting,
- oil and gas exploration,
- molecular modeling,
- physical simulations,
- cryptanalysis

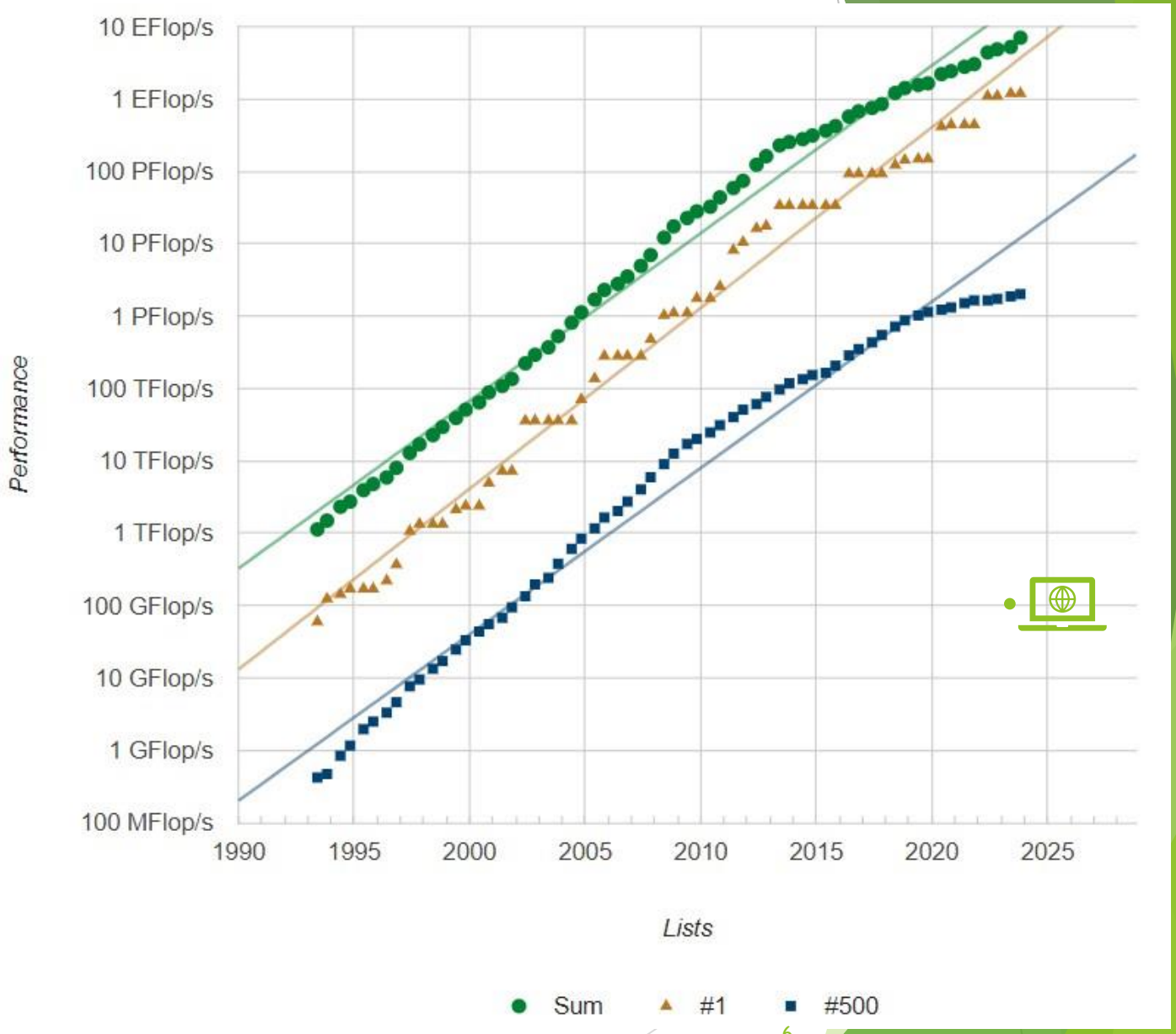
Flops

- FLOPS (Floating Point Operations Per Second), is a measure of a computer's processing speed for performing floating-point arithmetic, often used to gauge the performance of high-performance computing systems.
- GFLOPS: 109 FLOPS (Giga)
- TFLOPS: 1012 FLOPS (Tera)
- PFLOPS: 1015 FLOPS (Peta)
- EFLOPS: 1018 FLOPS (Exa)

- $FLOPS = \frac{flops}{cycle} \times \frac{cycle}{second} \times \frac{cores}{socket} \times \frac{socekets}{node} \times nodes$

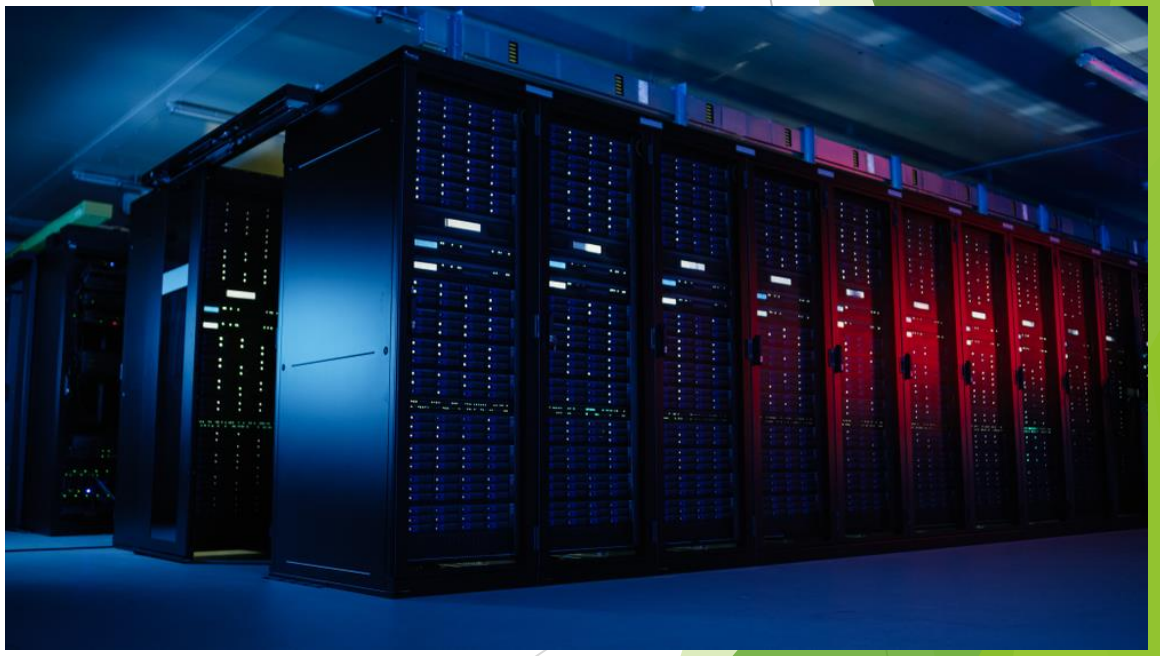
Top500

https://www.nextplatform.com/2023/11/13/top500-supercomputers-who-gets-the-most-out-of-peak-performance/



Computing cluster

- Computing cluster - a group or collection of **interconnected computers** that work **together** as a single system to perform computational tasks or provide services.



https://www.itpro.com/hardware/360706/7-most-powerful-computers-of-all-

Cluster architecture

A computer cluster is built of:

- Nodes:

Cluster nodes

- A cluster consists of multiple **individual computers** called nodes or cluster nodes.
- Each node is a **self-contained** computer with its own **CPU**, **memory**, storage, and operating system.
- Nodes are **connected** through a network to facilitate communication and data transfer.

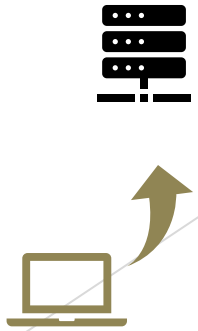
Cluster architecture

A computer cluster is built of:

- Nodes:
  - Head nodes
  - Compute nodes
  - Service nodes
- Interconection
- Shared storage

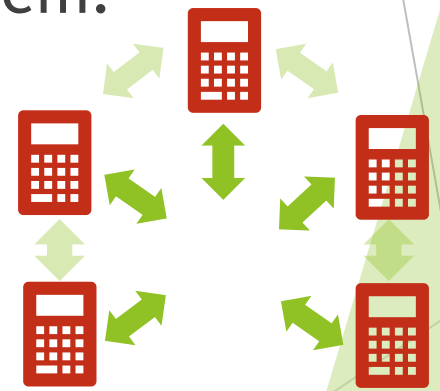
Head node

- The head node, also known as the master node, login node or frontend node, serves as the central control point for the cluster.
- It manages the cluster's overall operation, including **job scheduling**, **resource allocation**, and coordination of tasks among the nodes.
- The head node typically runs **cluster management software** and provides a user interface for cluster administration.



Compute node

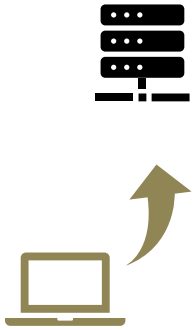
- Compute nodes are the **primary processing** units in the cluster.
- These nodes **execute computational** tasks, simulations, data analysis, or other workload-specific operations.
- Compute nodes are responsible for executing the **tasks assigned** to them by the head node or cluster management system.



Nodes

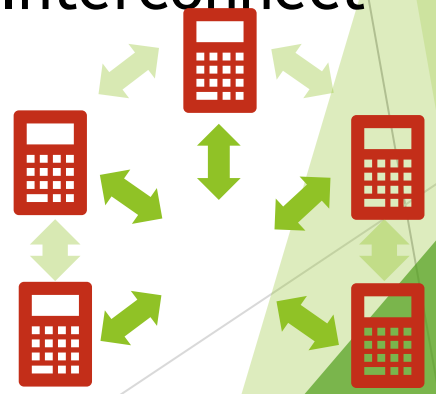
Login node:

- User Access Point
- Job Management
- Data Pre-/Post-Processing (small)
- Data Transfer (sometimes better is a bether place)
- Limited Computing Power

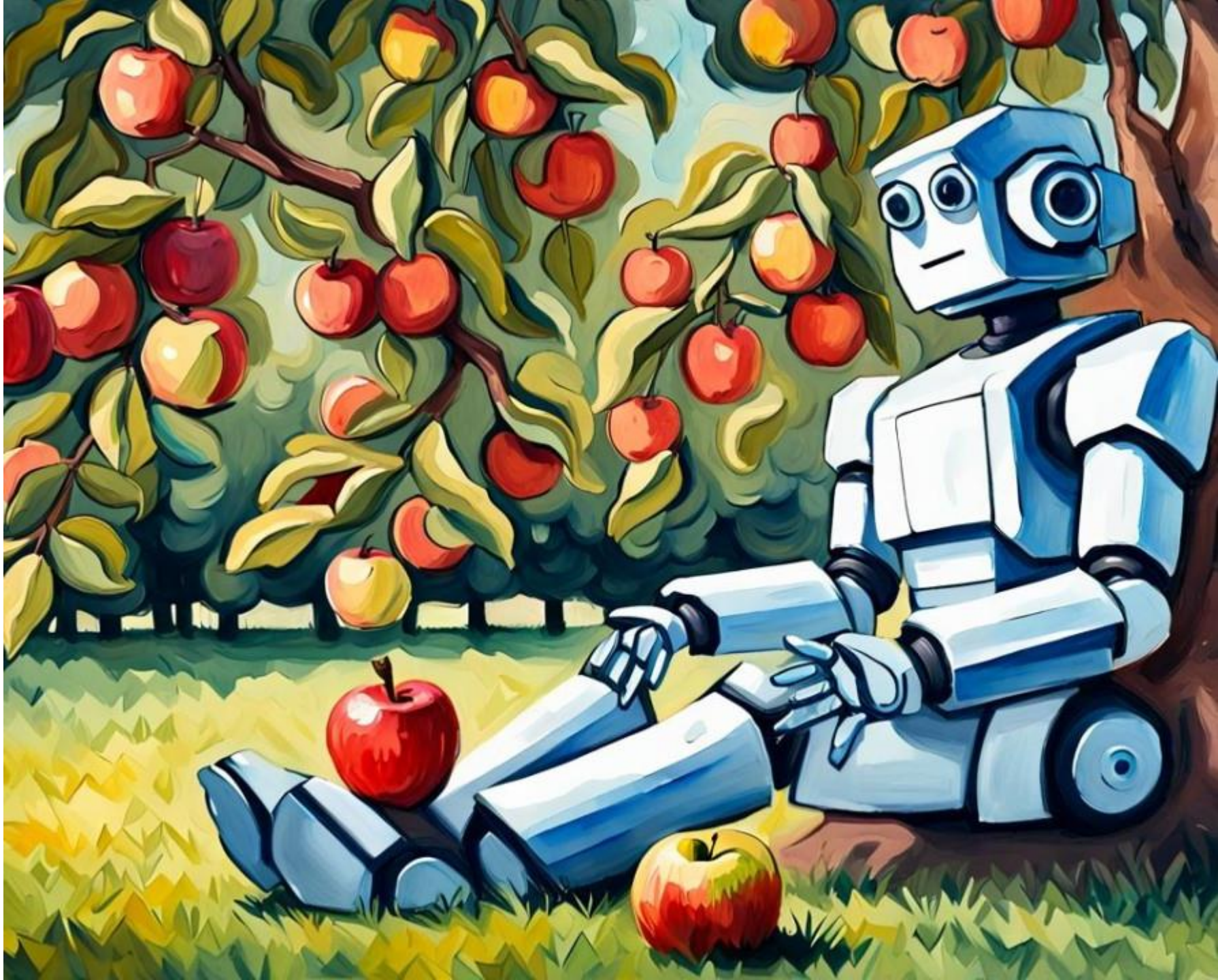


Compute node:

- Computations are performed on these
- **High** Computing Power
- No Direct Access (typically)
- **High-Speed** Interconnect



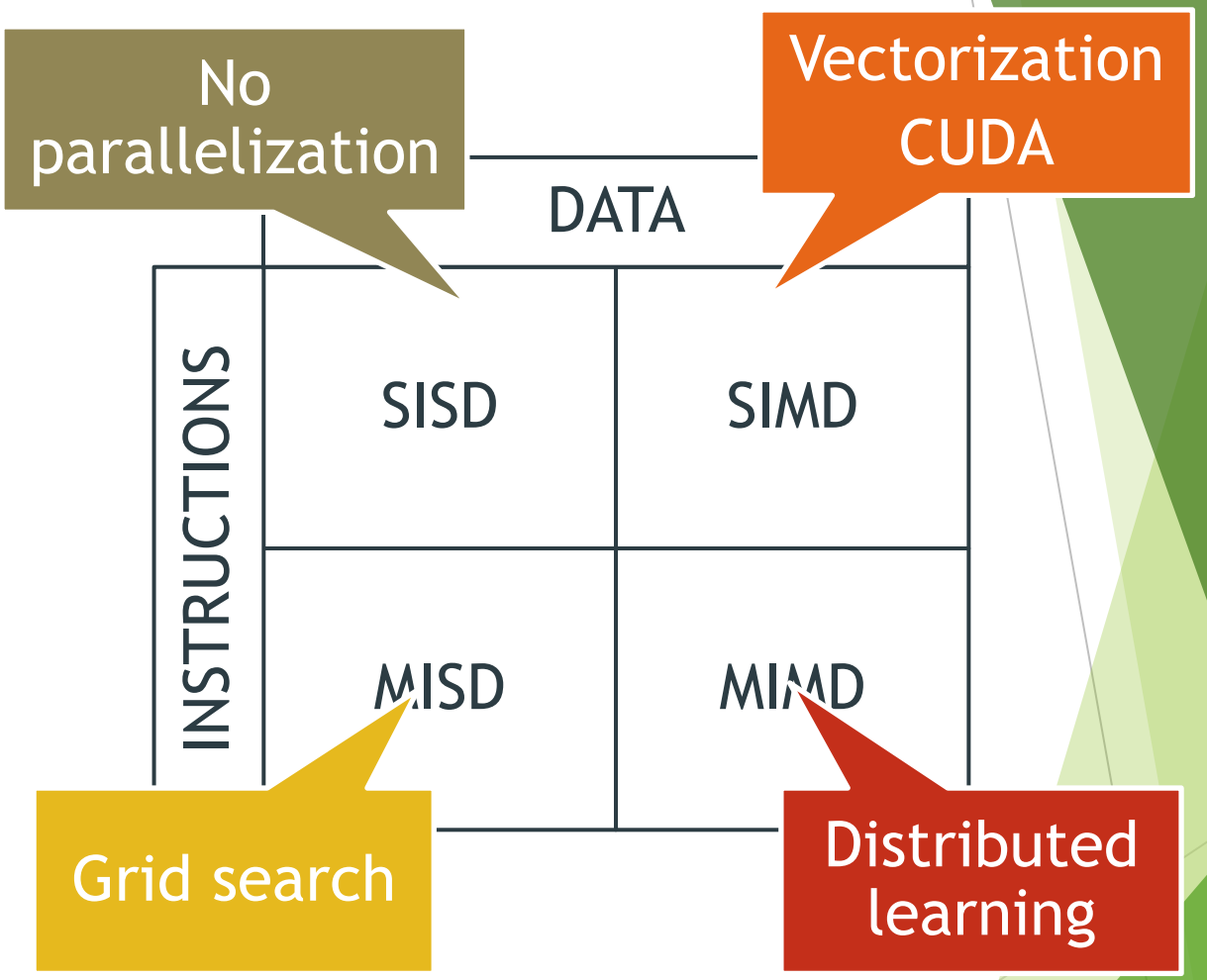
Parallel compiuting



Parallel integral

t =T/9

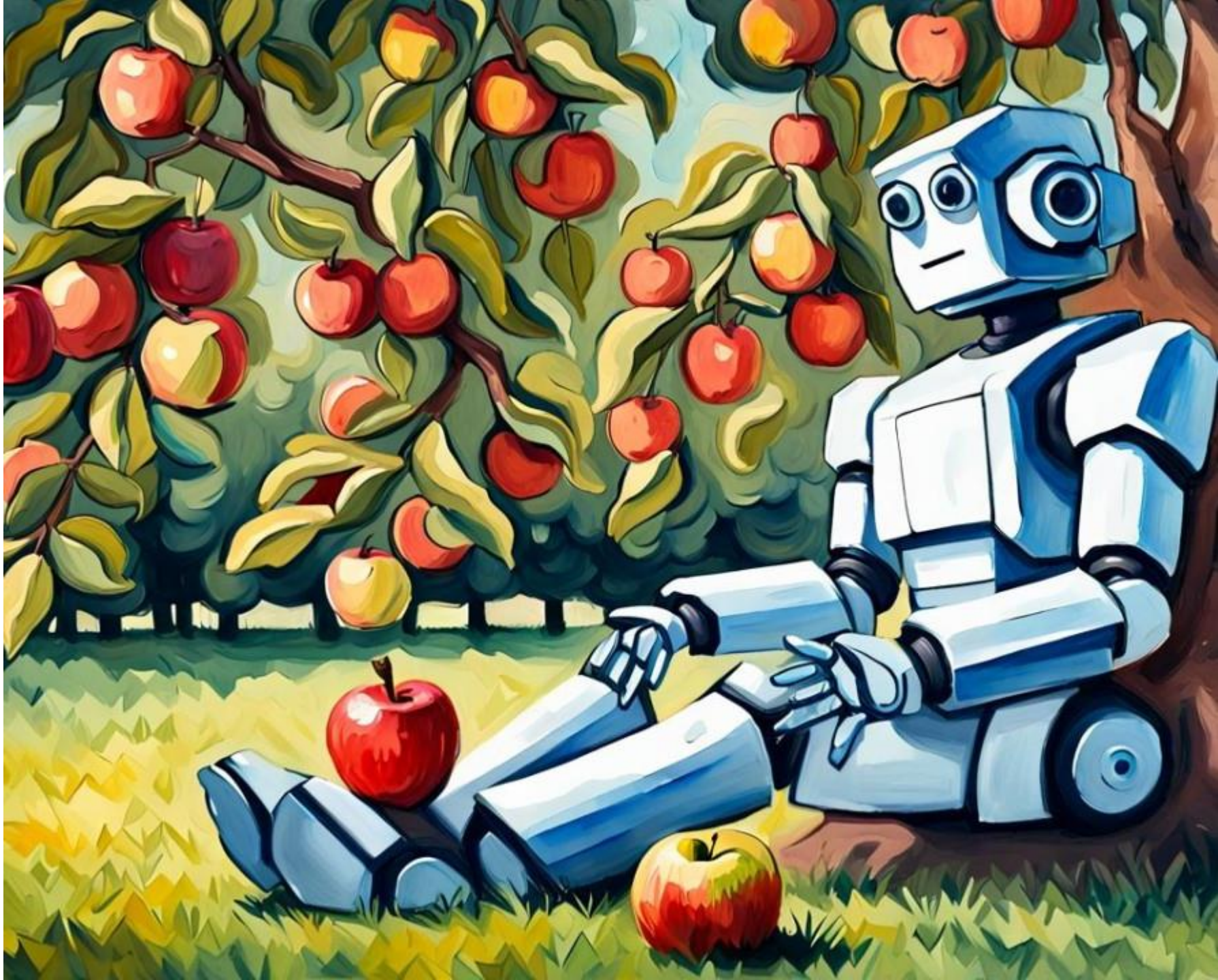
Parallelism models



Parallelization

- **Trivial Parallelization** - Problem that can be parallelized without communication
- **Vectorization** – Code use SIMD on single CPU
- **OpenMP** – parallization with threads – single node
- **MPI** – parallization with processes over multiple nodes

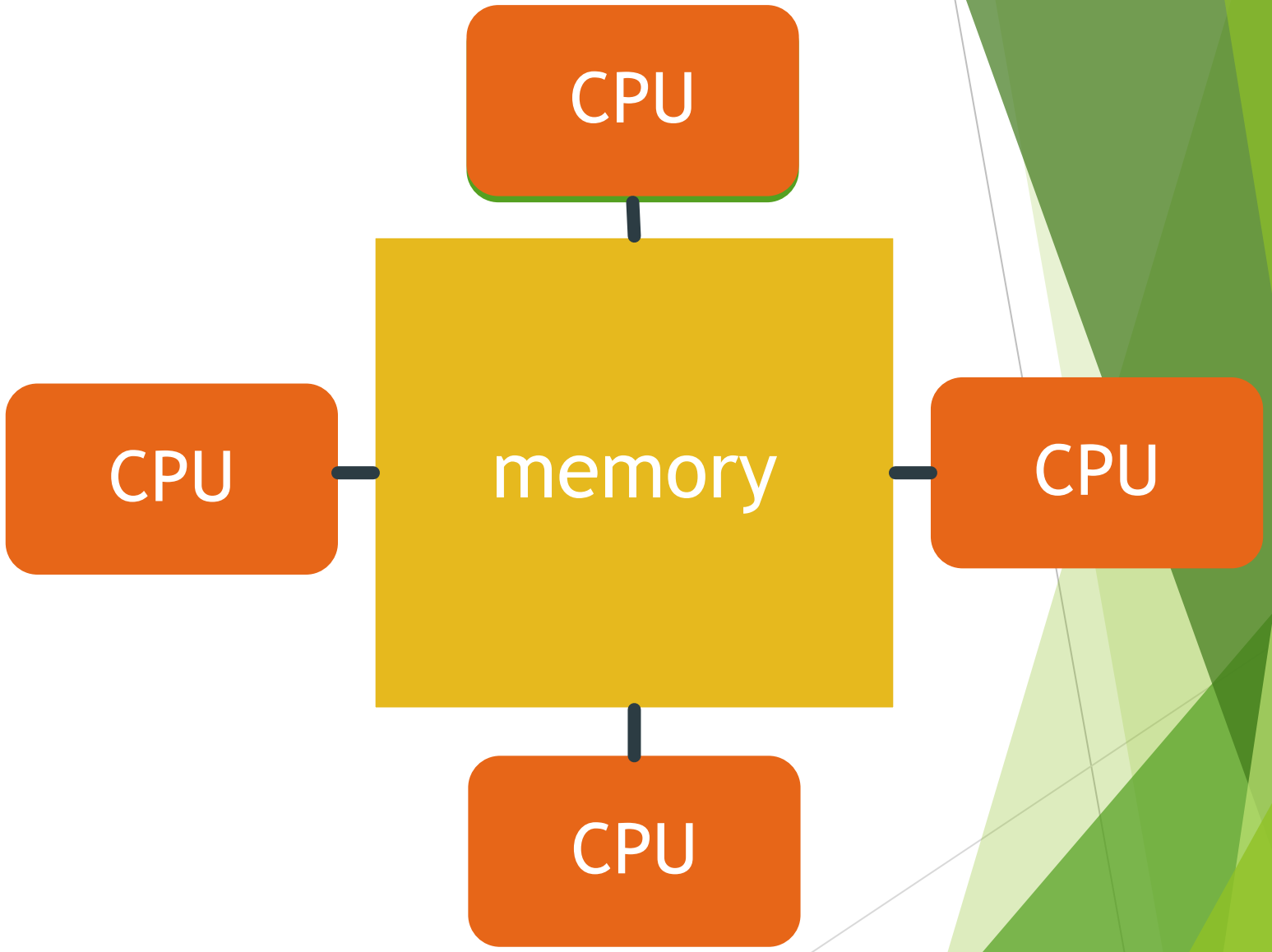
Memory models



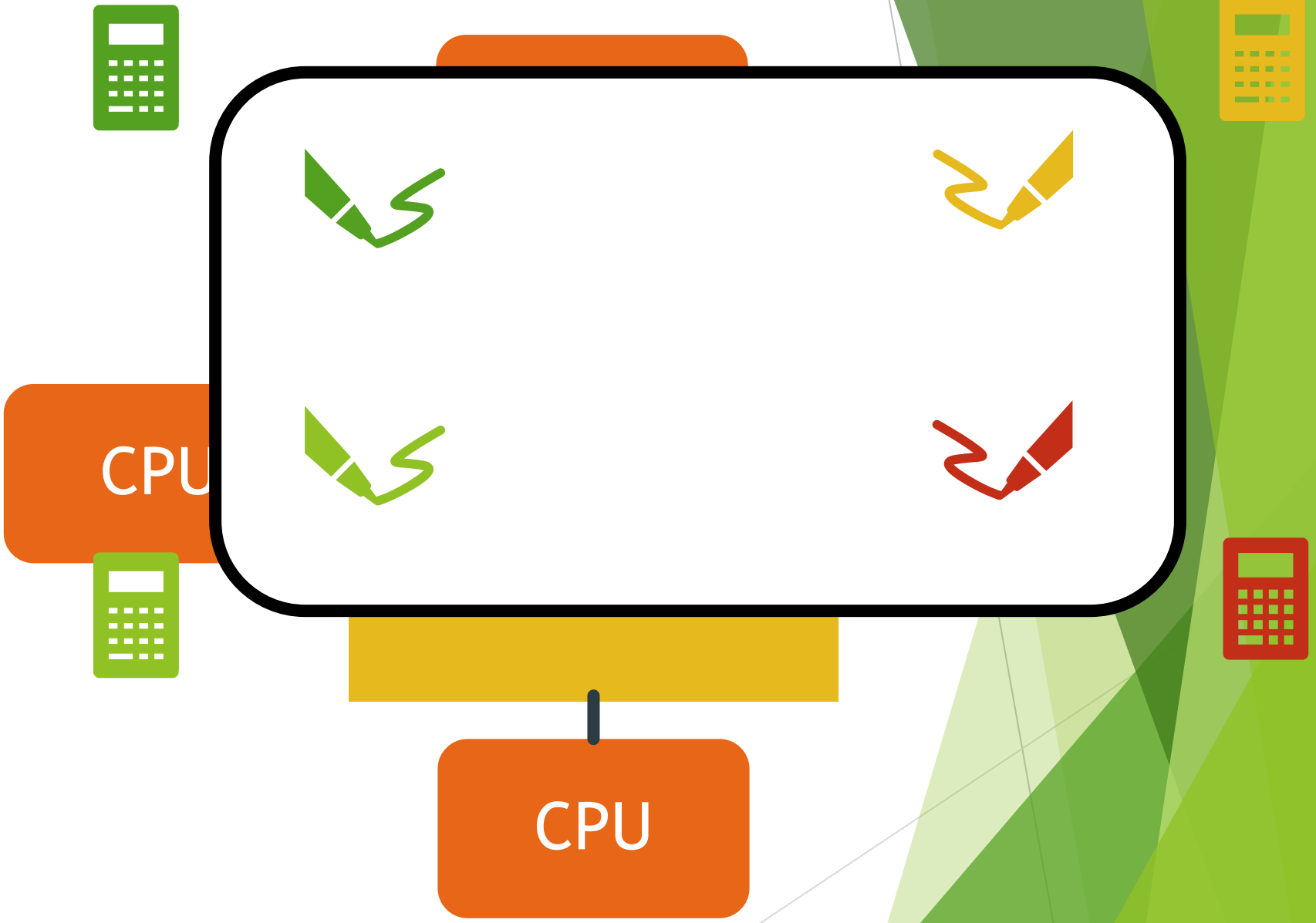
Memory models

- Distributed memory
- Shared memory
- Mixed memory

Shared memory

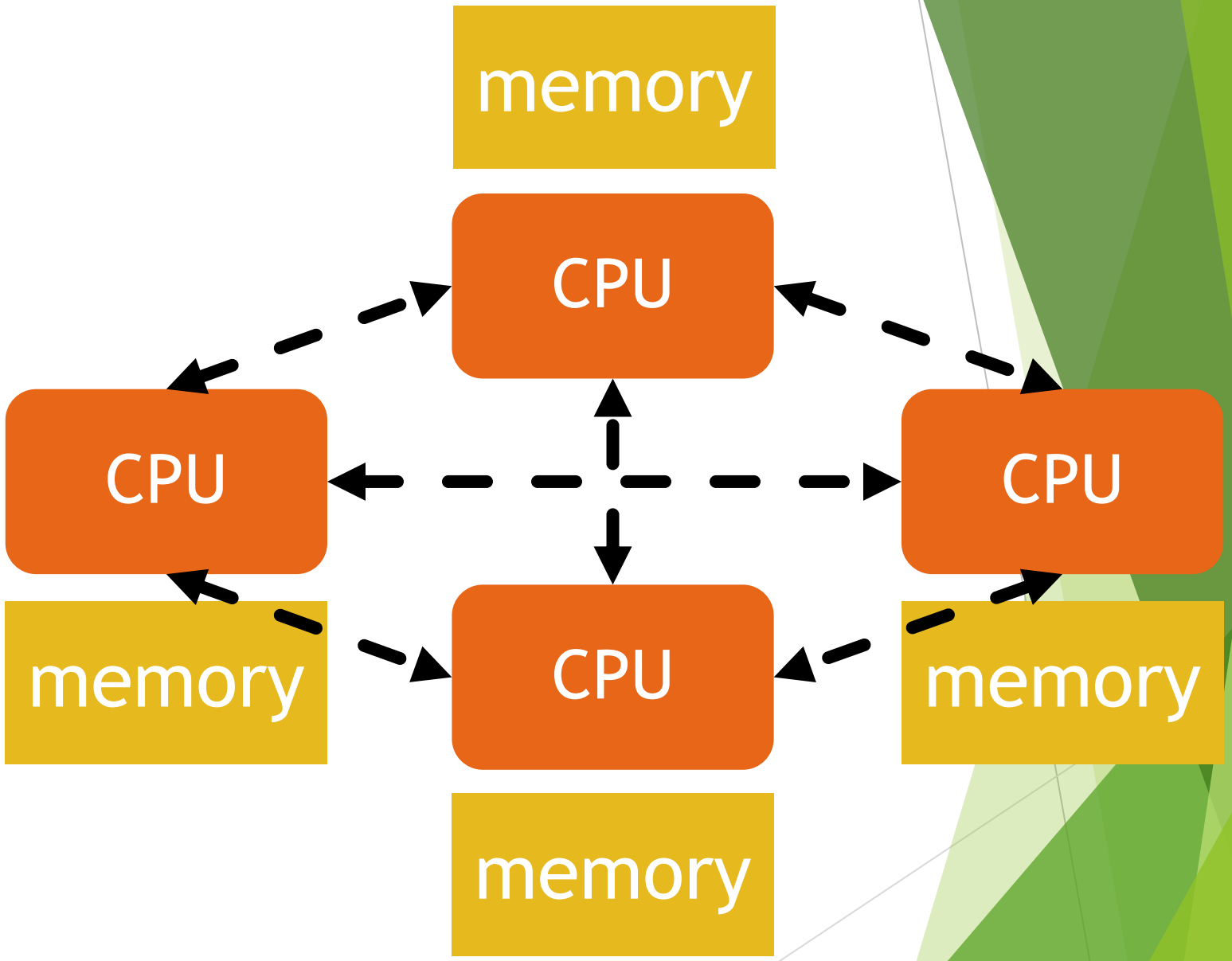


Shared memory

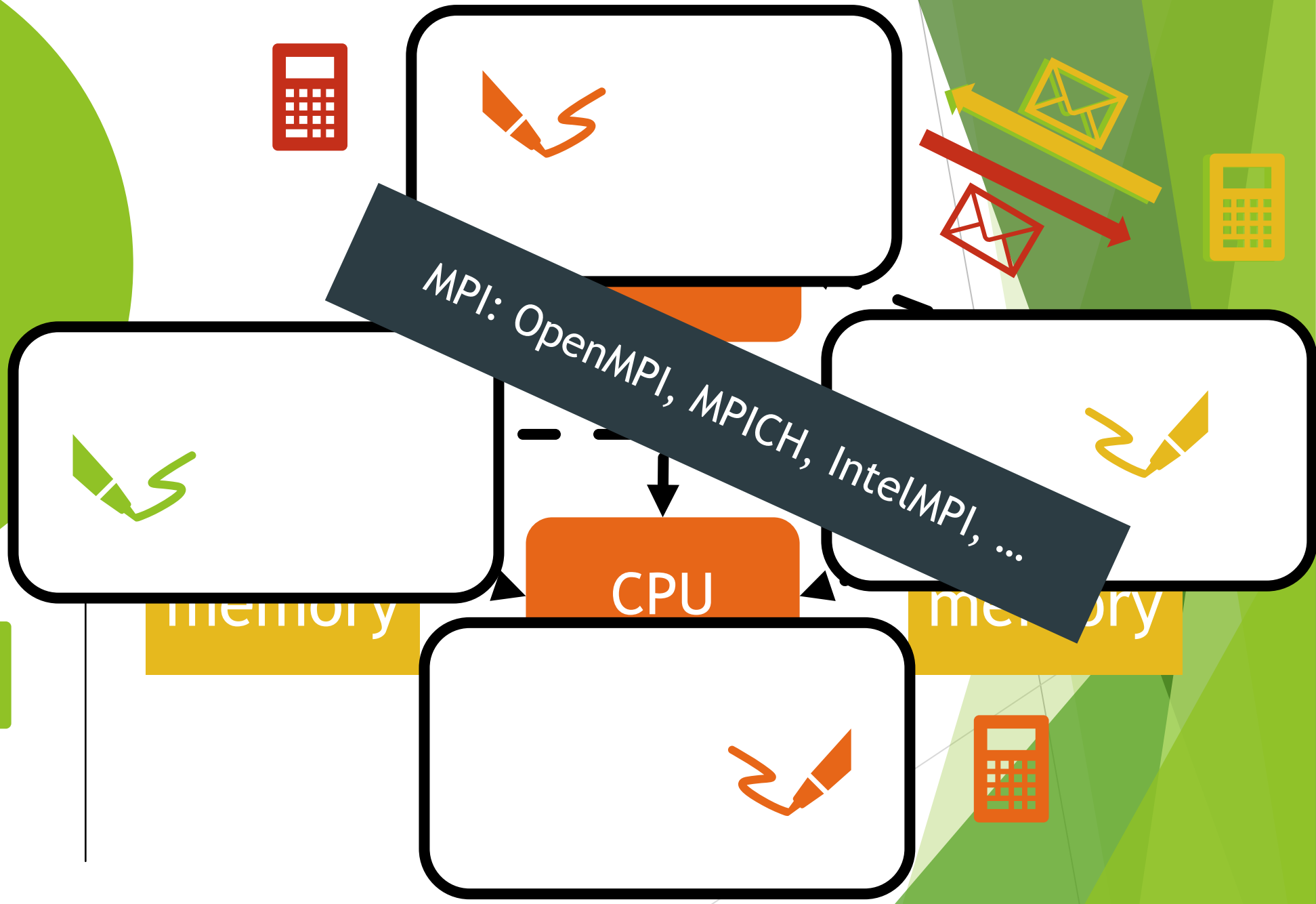


Shared memory

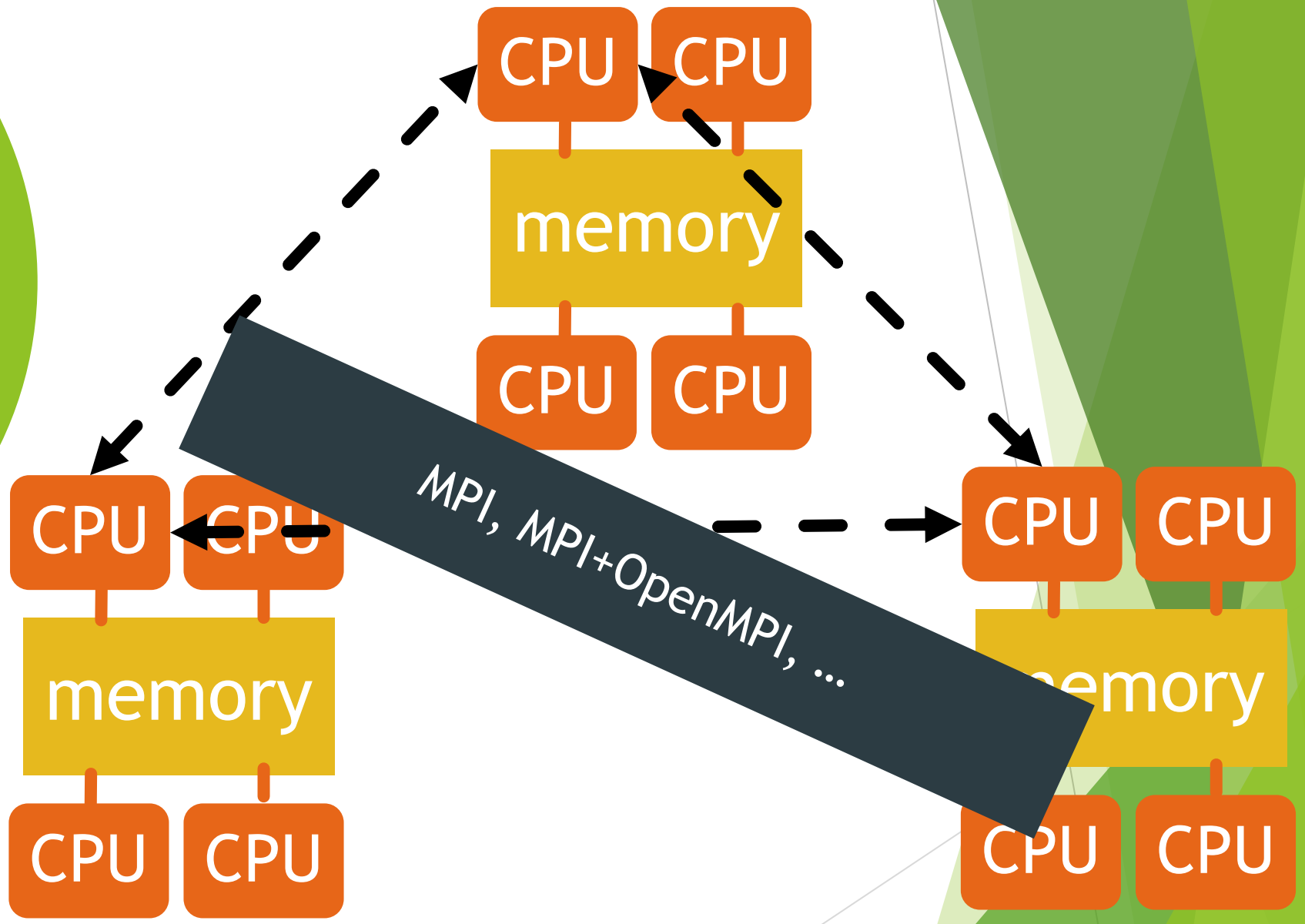
Distributed memory



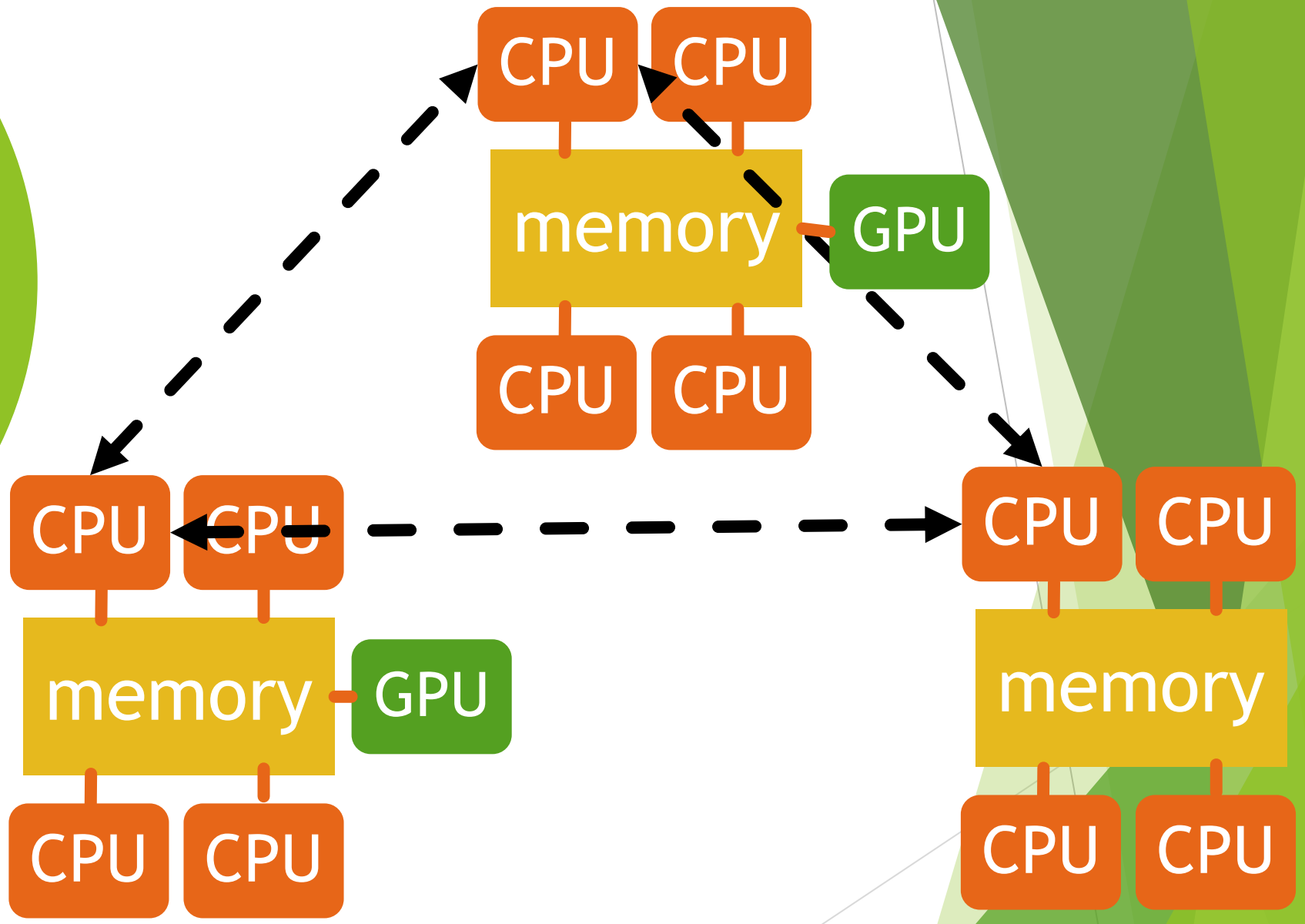
Distributed memory



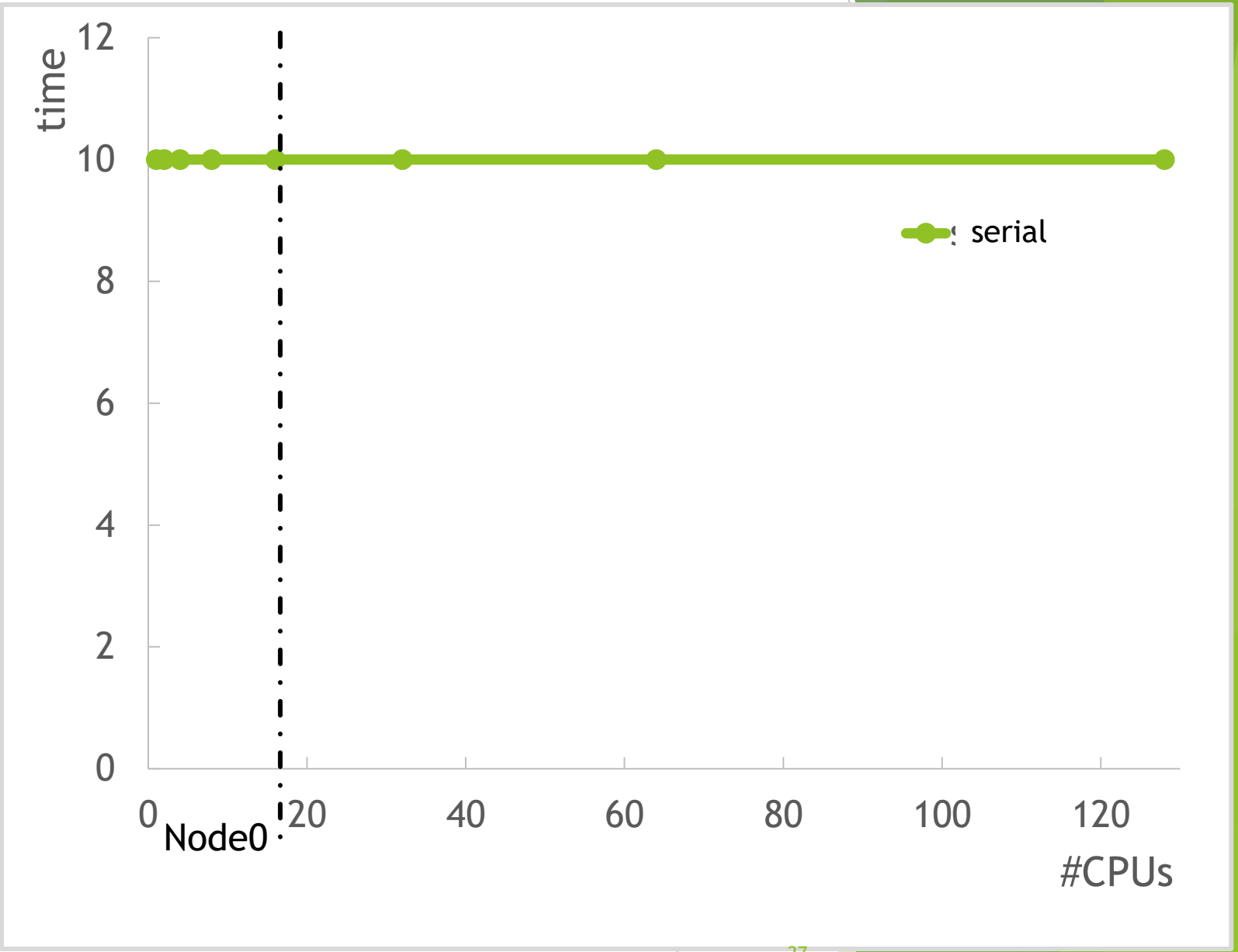
Mixed memory



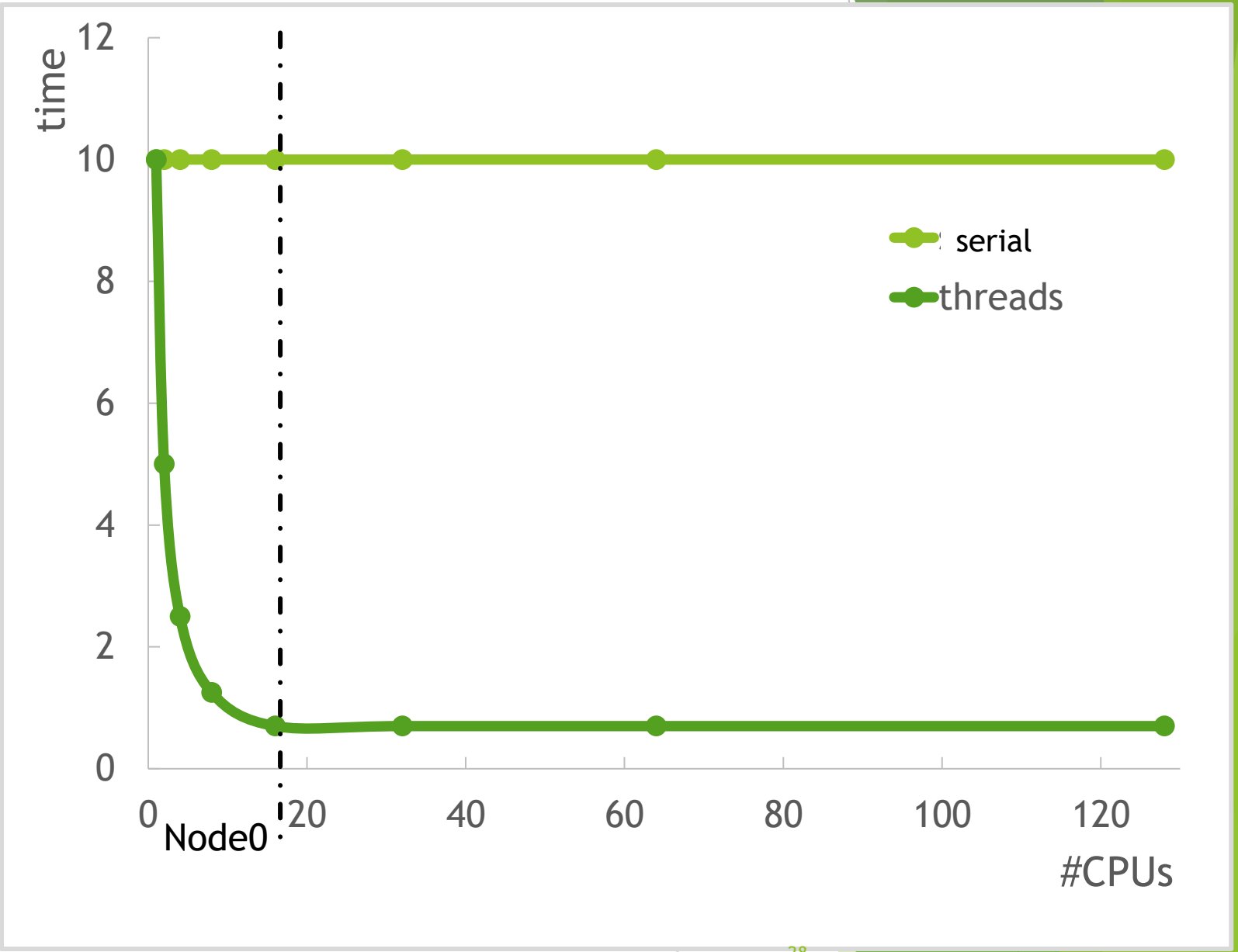
Mixed memory



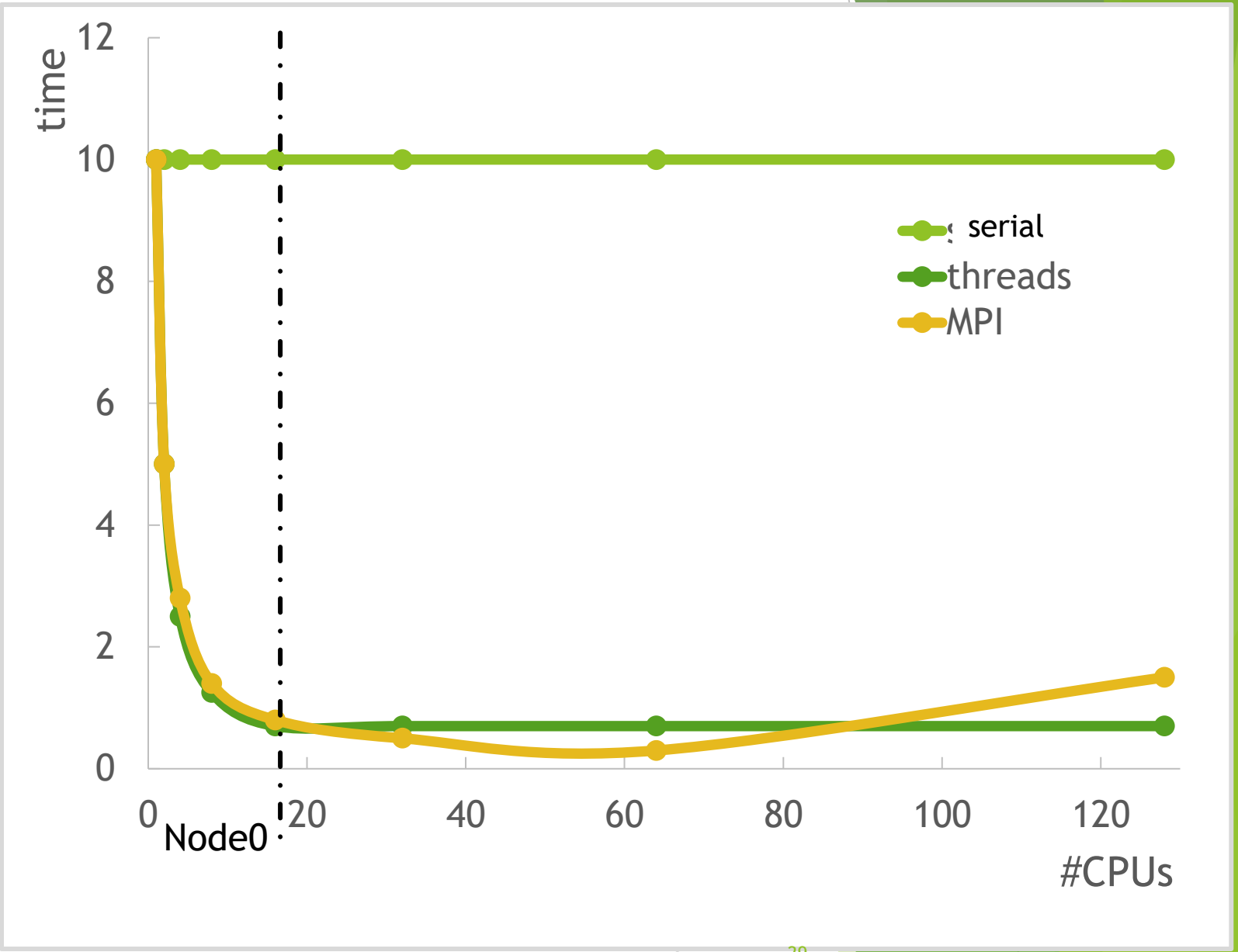
Suboptimal calculations



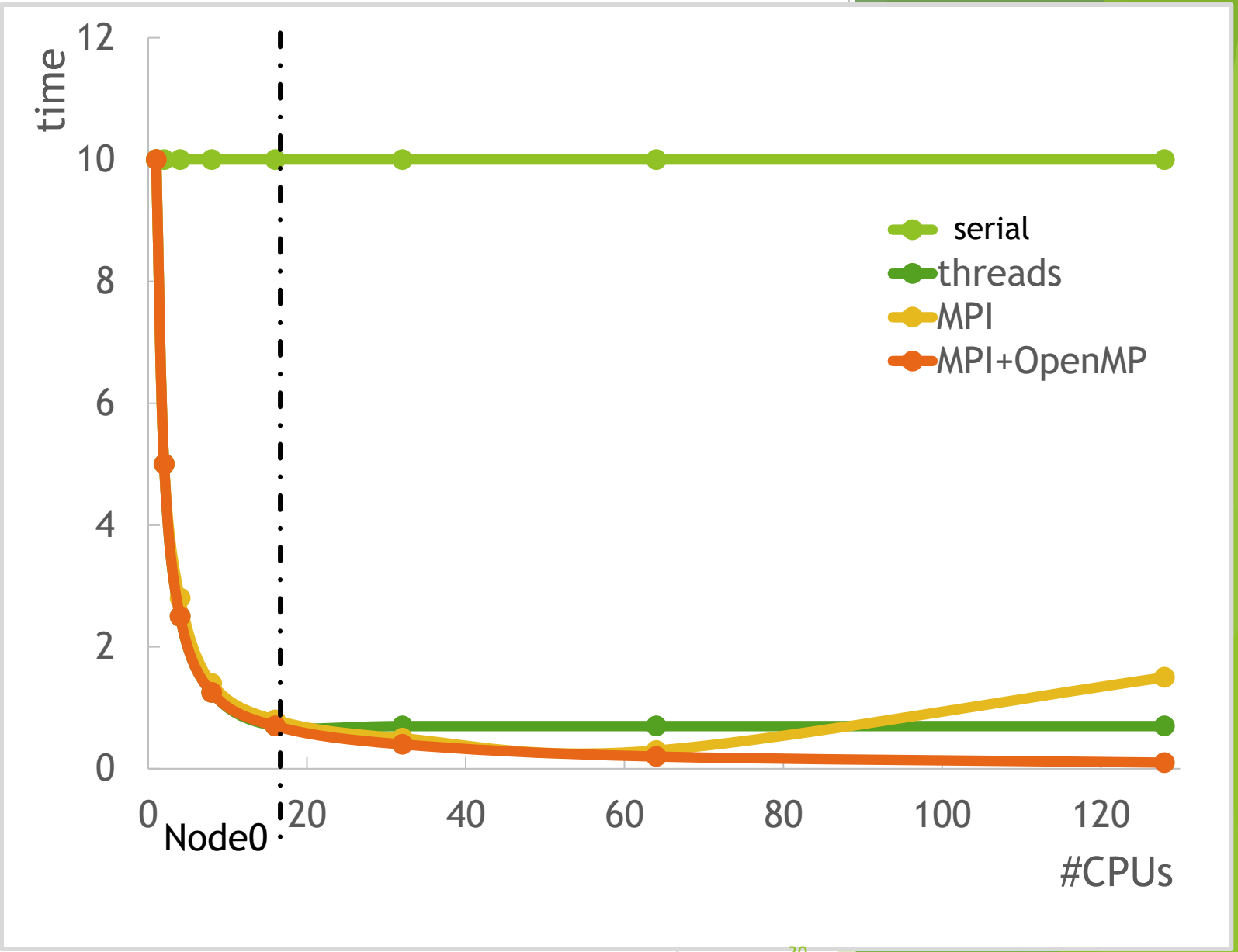
Suboptimal calculations



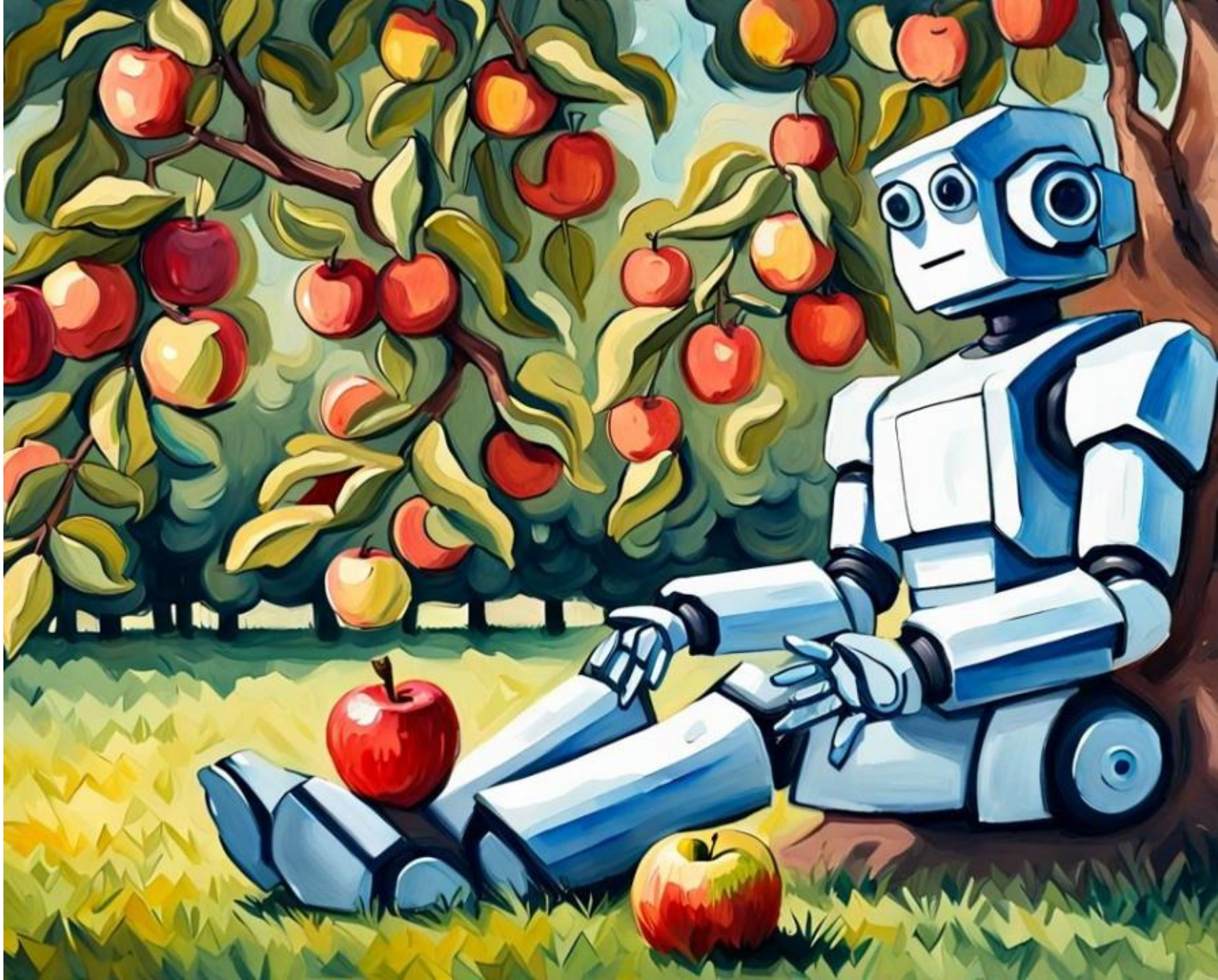
Suboptimal calculations



Suboptimal calculations



Queue system



Queue system

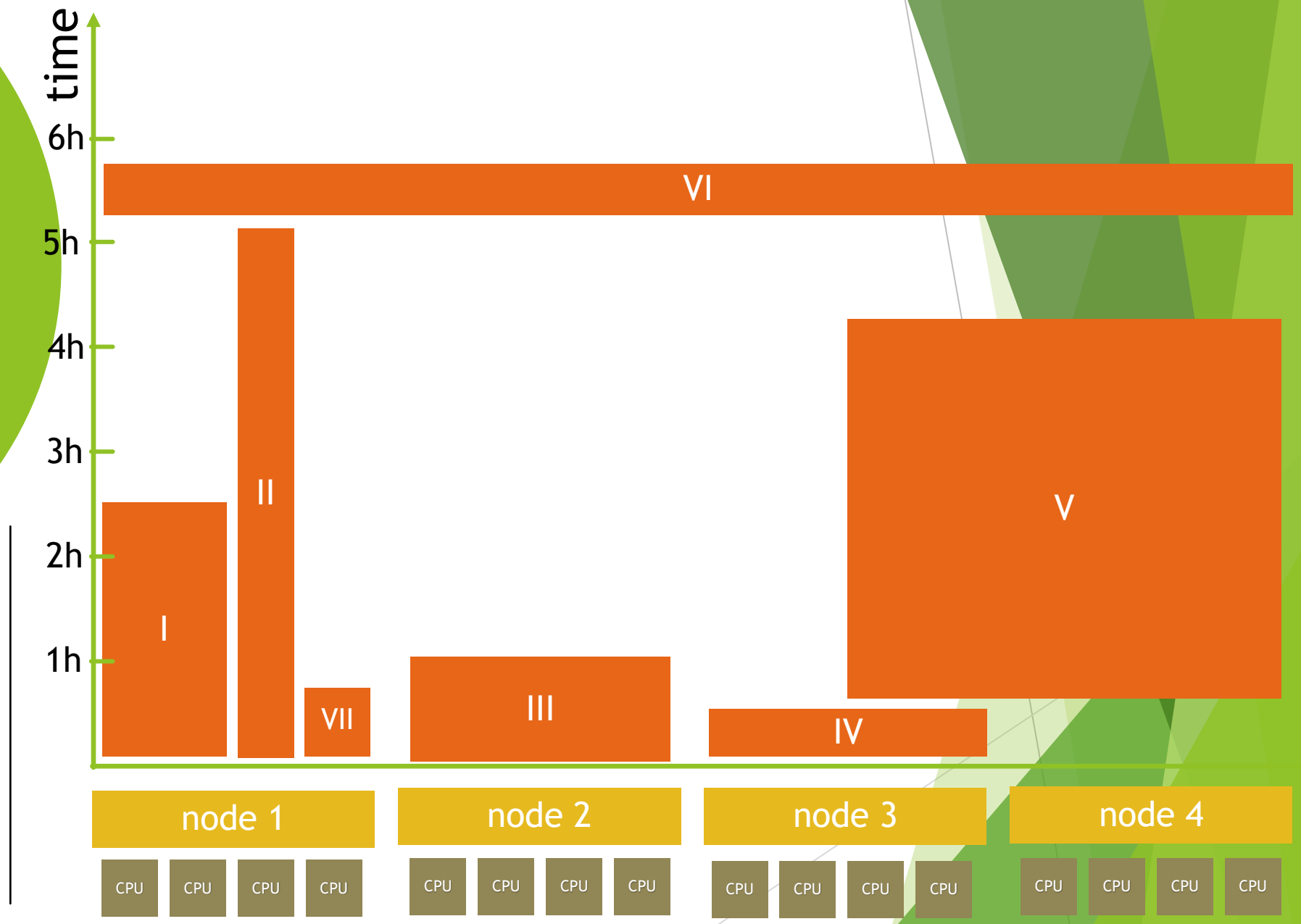
Queue system - a software component responsible for managing the **allocation** and **execution** of computational tasks or **jobs** on the available computing resources within a cluster or **HPC infrastructure**. It ensures fair and **efficient utilization** of the resources while providing a controlled and organized environment for job submission and execution.

Key aspects of a queue system:

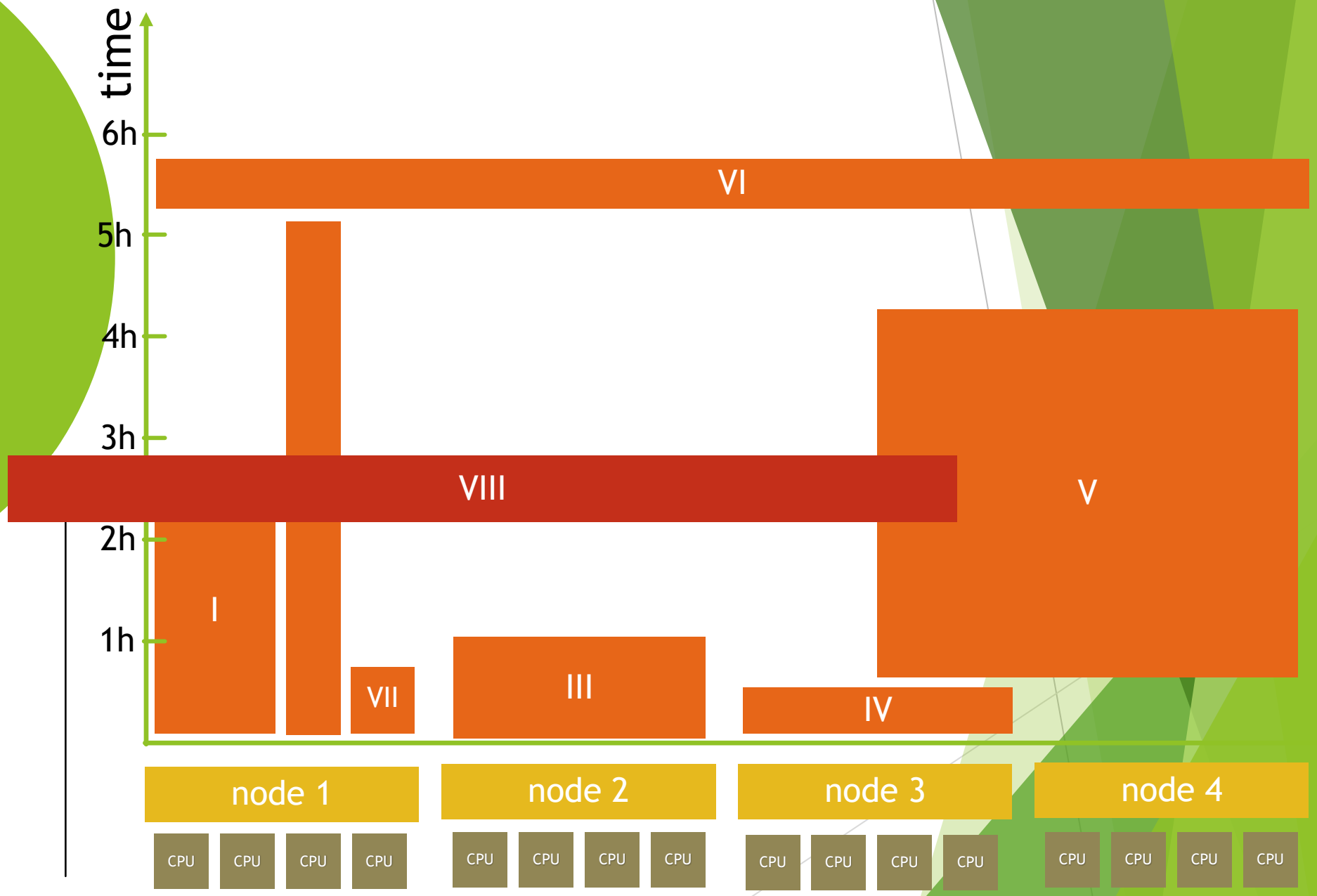
- Resource Allocation.
- Job Prioritization.
- Job Scheduling.
- Job Monitoring and control.
- Fairness and quotas.

Waiting in queue

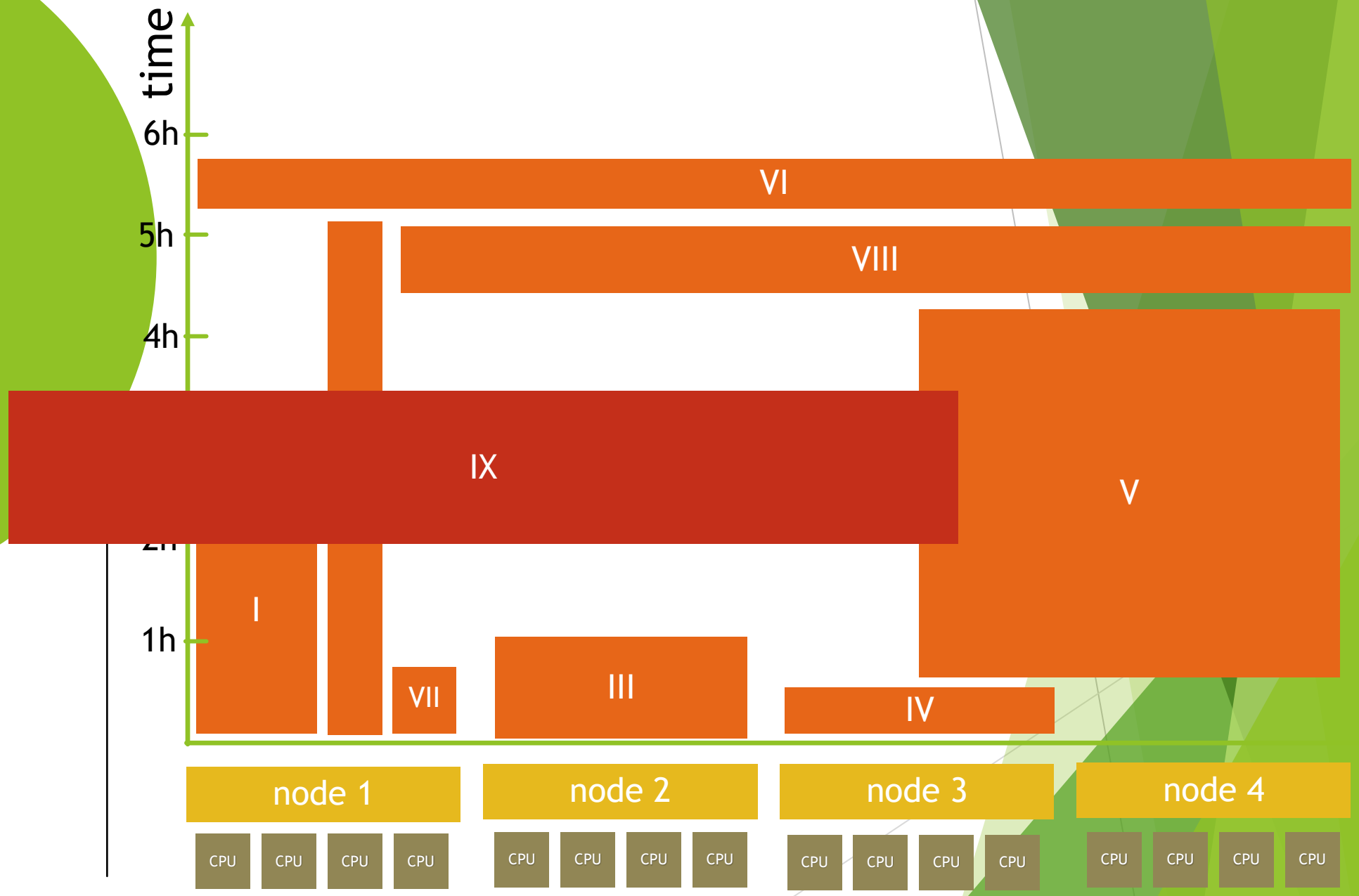
Waiting in queue



Waiting in queue



Waiting in queue

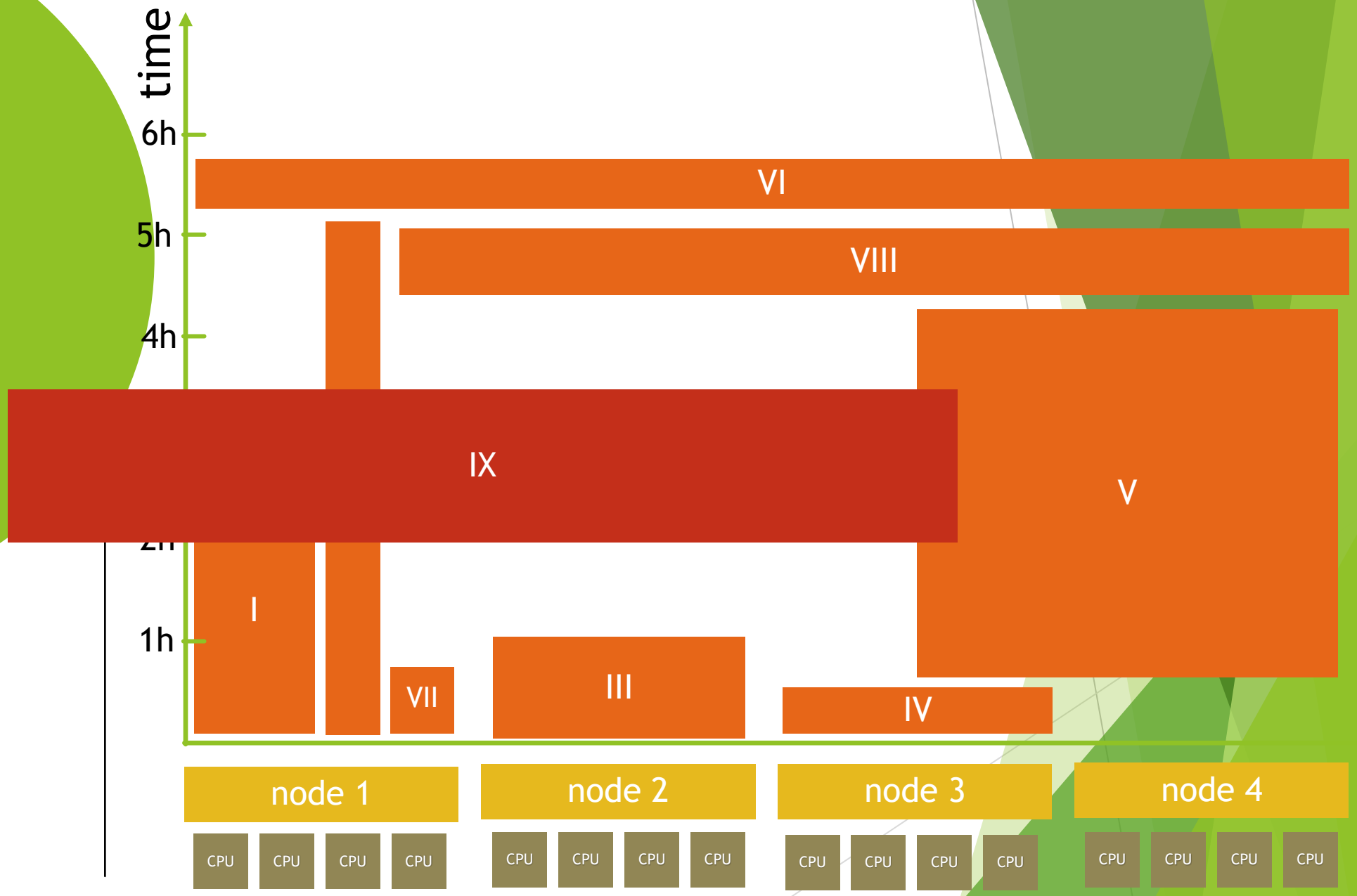


Resource in SLURM

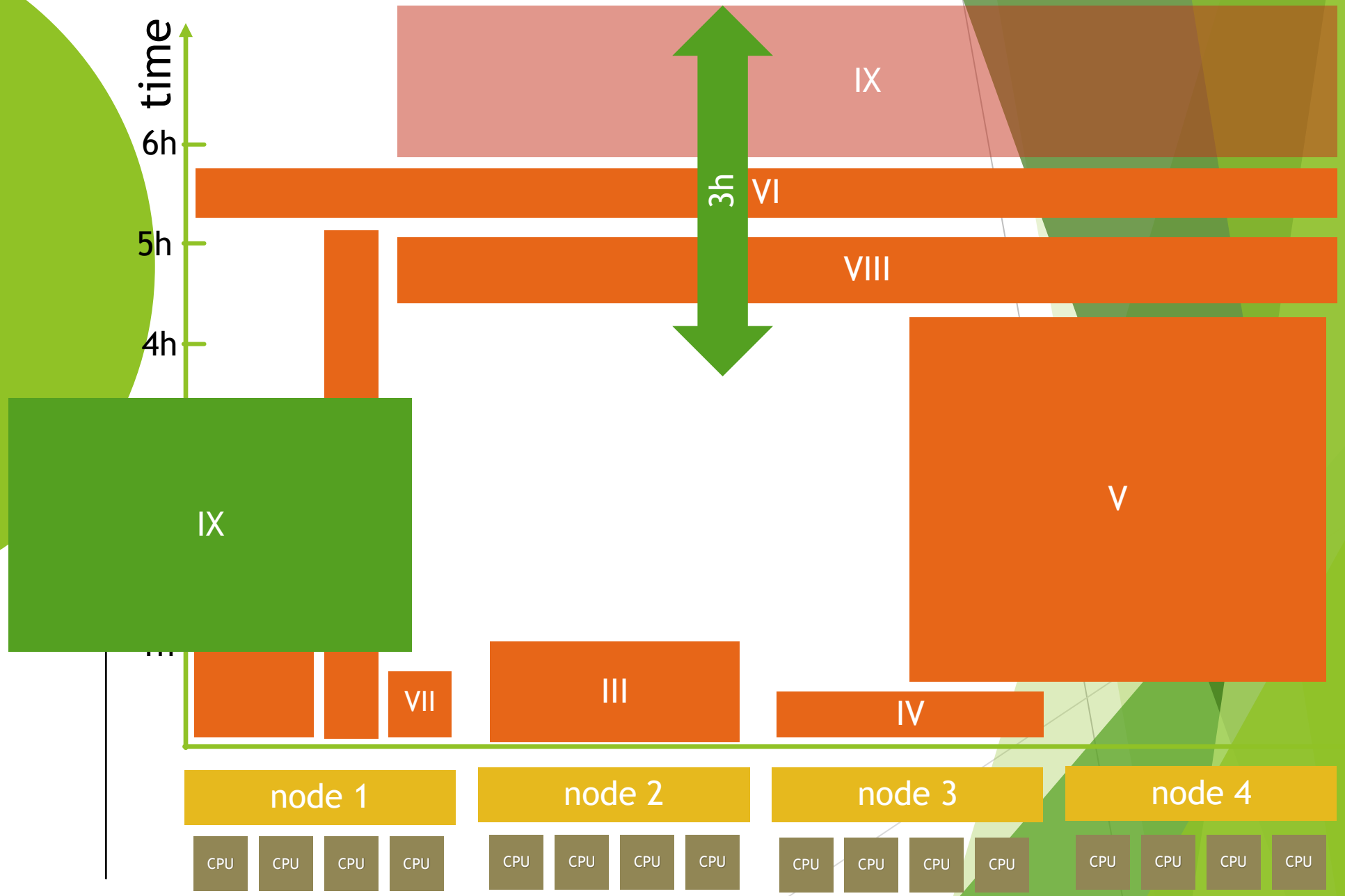
- CPU cores
  - Processes (tasks) --ntasks=<number>
  - Threads (CPUs) --cpus-per-task=<number>
- Memory --mem=<size>[units]
- Time --time=d-HH:MM:SS
- $T_{solution} = T_{queue} + T_{calculations}$
- Nodes --nodes=<number>
- Licenses --licenses=<license>

- Partition --partition=<partition_name>
- Constraints --constraint=<list>

Waiting in queue

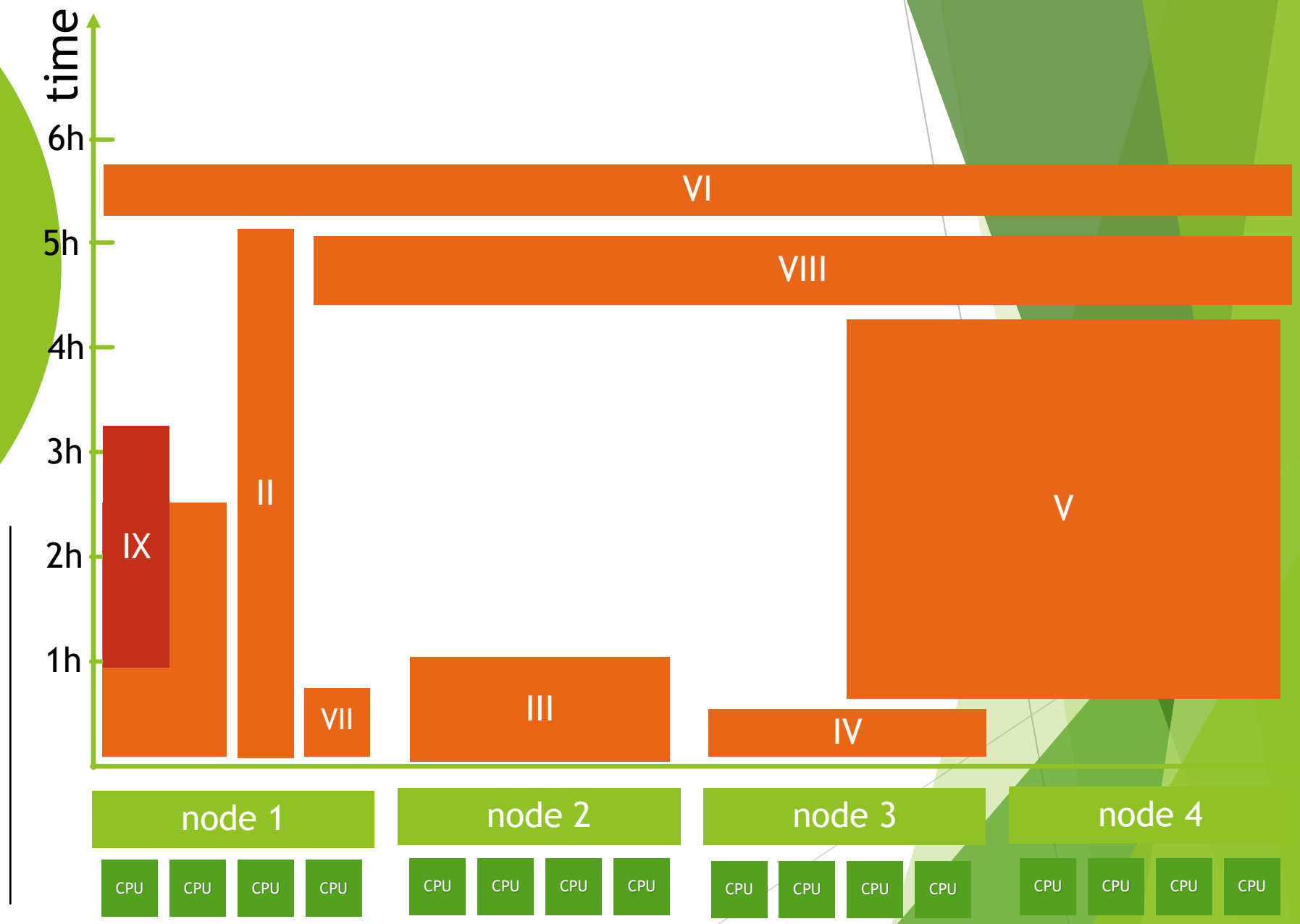


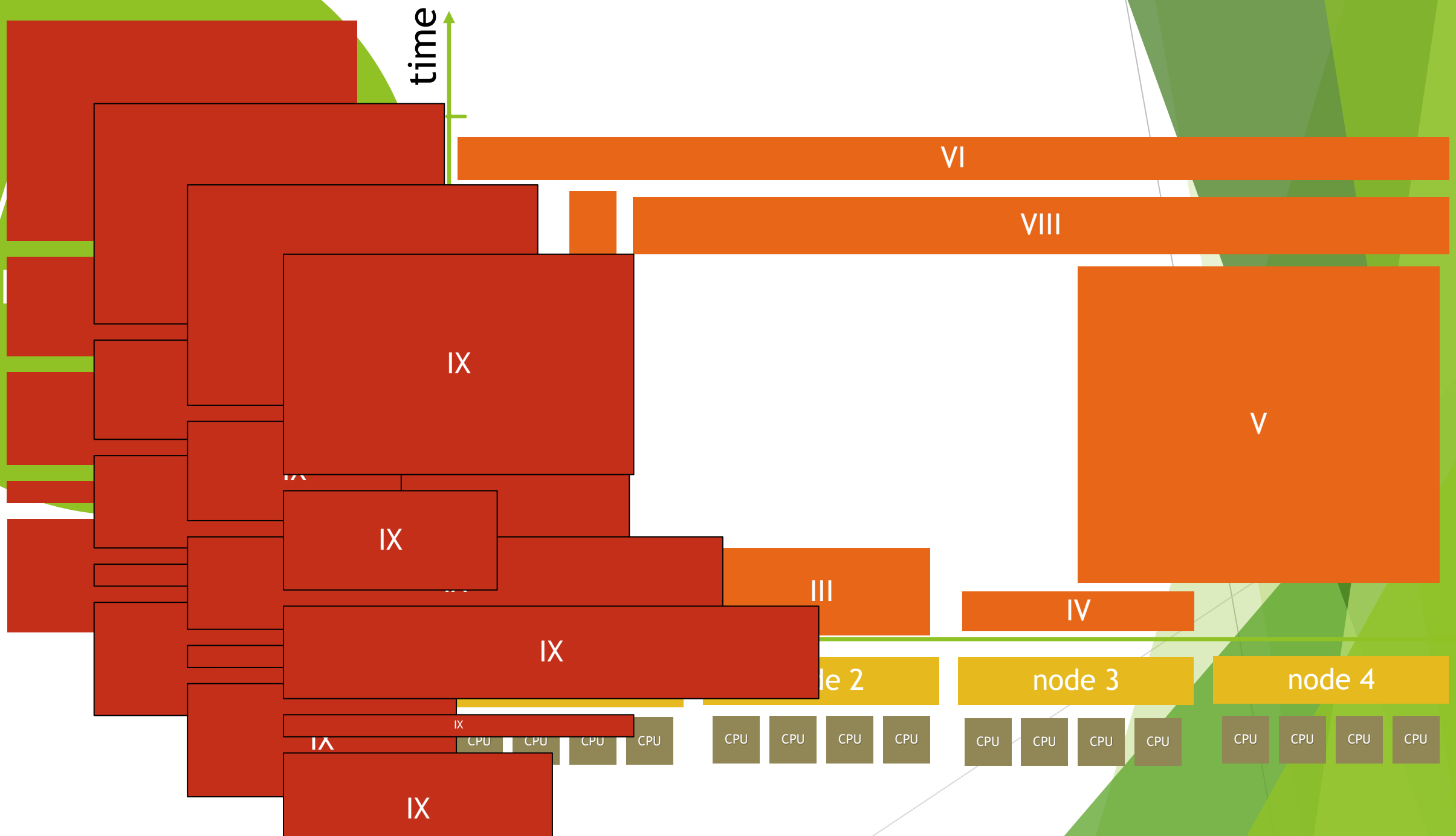
Waiting in queue



Waiting in queue

Waiting in queue

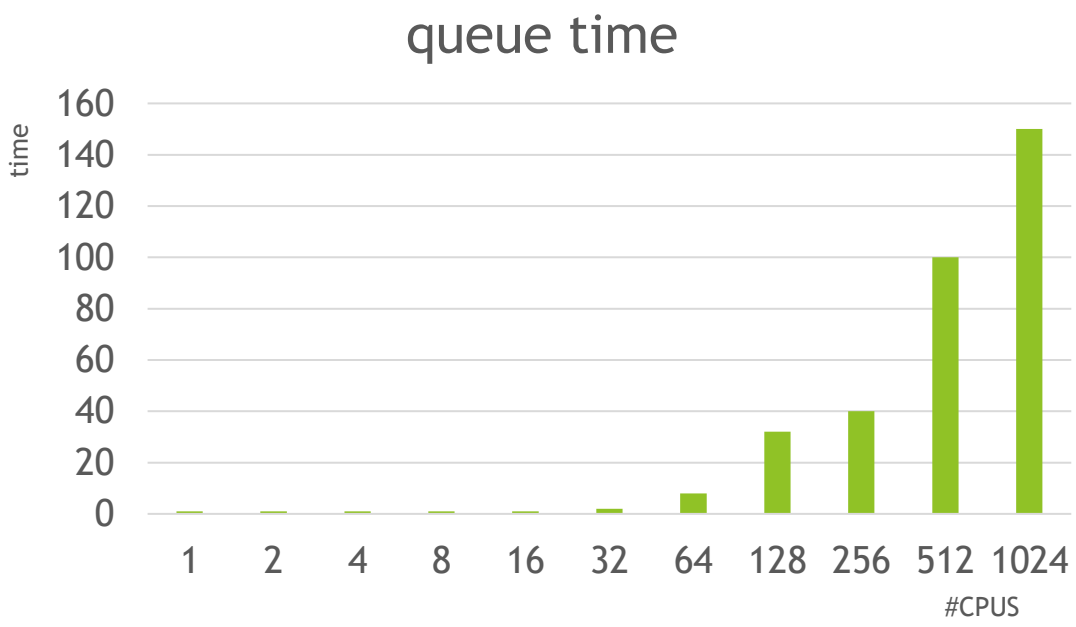



time
VI
VIII
IX
IX
IX
IX
IX
V
III
IV
le 2
node 3
node 4
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU

SLURM job priority factors

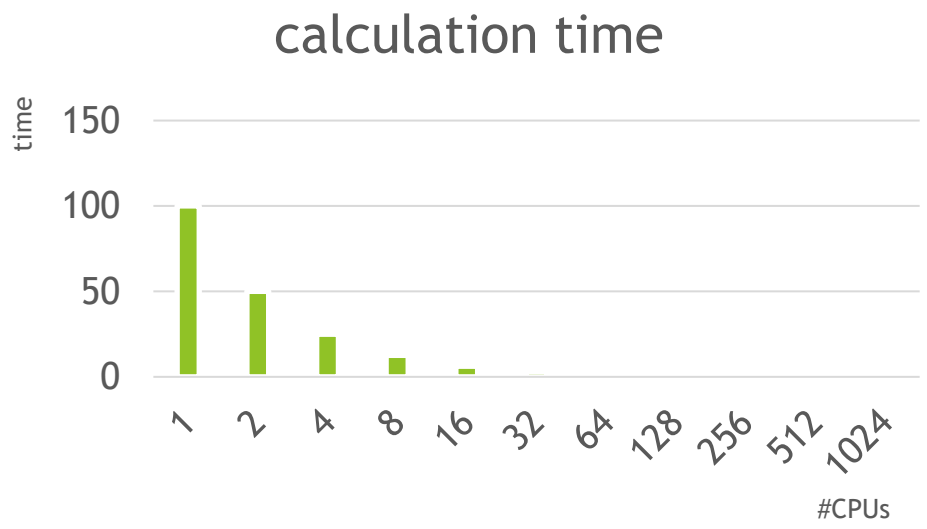
- **Age**
- Association
- **Fair-share**
- **Job size**
- Nice
- Partition
- **QOS**
- Site
- **TRES**
  - Licences
  - CPUs
  - GPUs
  - Memory

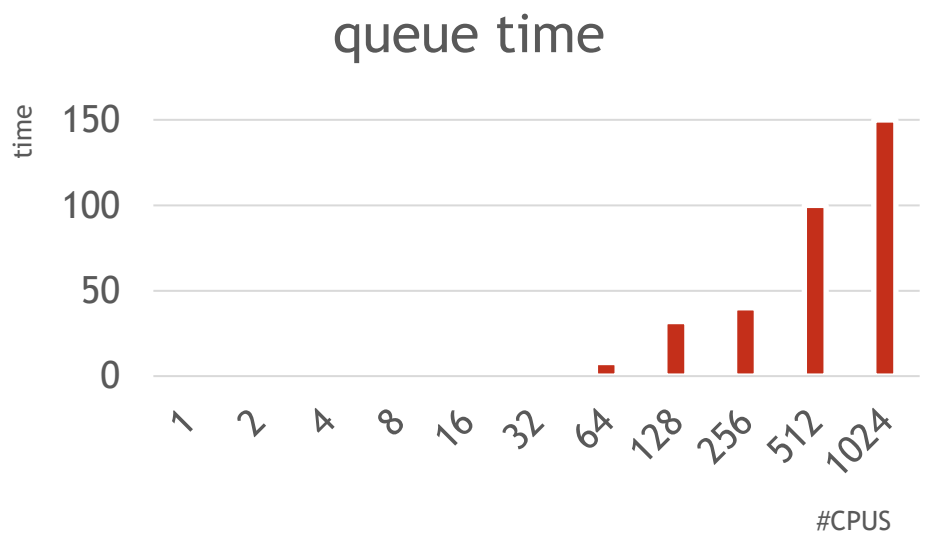
$$P = F_{size} + W_{age}F_{age} + W_{f-s}F_{f-s} + \\ + W_{size}F_{size}W_{part}F_{part} + \\ + \sum W_{TRES_i}F_{TRES_i} - \\ - F_{nice}$$

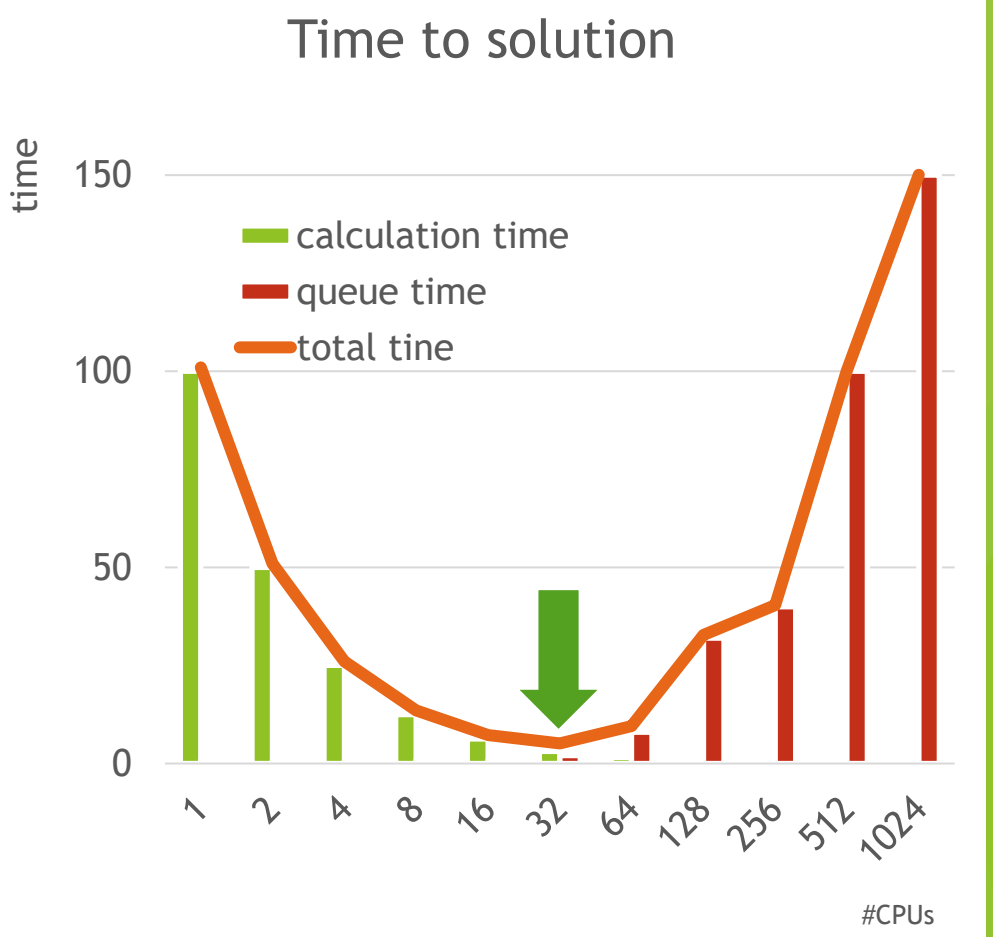


Time to solution

$$T_{solution} = T_{queue} + T_{calculations}$$







Working in SLURM

- Sabmitting job
  `sbatch job.sh`
- Check queue
  `squeu`
- Check your jobs
  `squeue -u $USER`
- Cancel a job
  `scancel JOB_ID`

- Run interactive job
  `srun [OPTIONS] --pty bash -l`

```
JOBID   PARTITION     NAME    USER  ST       TIME NODES NODELIST(REASON)
12345     compute    myjob   alice   R   01:23:45     2      node[01-02]
12346         gpu   gpujob     bob  PD       0:00     1      (Resources)
12347     compute  testjob   carol   R   00:10:15     1           node03
12348       debug   debug1    dave  PD       0:00     1       (Priority)
```

SLURM script

```
#!/bin/bash

#SBATCH --job-name=my_python_job    # Job name
#SBATCH --output=output_%j.log      # Output log file
#SBATCH --error=error_%j.log        # Error log file
#SBATCH --time=01:00:00             # Time limit (hh:mm:ss)
#SBATCH --nodes=1                   # Number of nodes
#SBATCH --ntasks=1                  # Number of rocesses
#SBATCH --cpus-per-task=4           # Number of CPU per task
#SBATCH --mem=16GB                  # Memory allocation
#SBATCH --partition=compute         # Partition/queue name


# Load necessary modules
module load python


# Activate a virtual environment (optional)
source ~/my_env/bin/activate

# Run your Python script
srun python my_script.py
```

SLURM scripts

`--job-name`	a name for your job.
`--output`	the file to save standard output.
	%j is replaced with the job ID.
`--error`	the file to save error messages.
`--time`	a time limit for the job (s-HH:MM:SS).
`--nodes`	number of nodes required.
`--ntasks`	number of tasks or processes to run.
`--cpus-per-task`	cumber of CPU cores for each task.
`--mem`	memory required per node.
`--partition`	the partition to submit the job to.

Interactive jobs

Interactive jobs

An **interactive job in SLURM** is a job where you get a **live interactive session** on a compute node, allowing you to run commands interactively instead of submitting a batch script. It is commonly used for **debugging, testing code, running interactive applications, or performing exploratory analysis.**

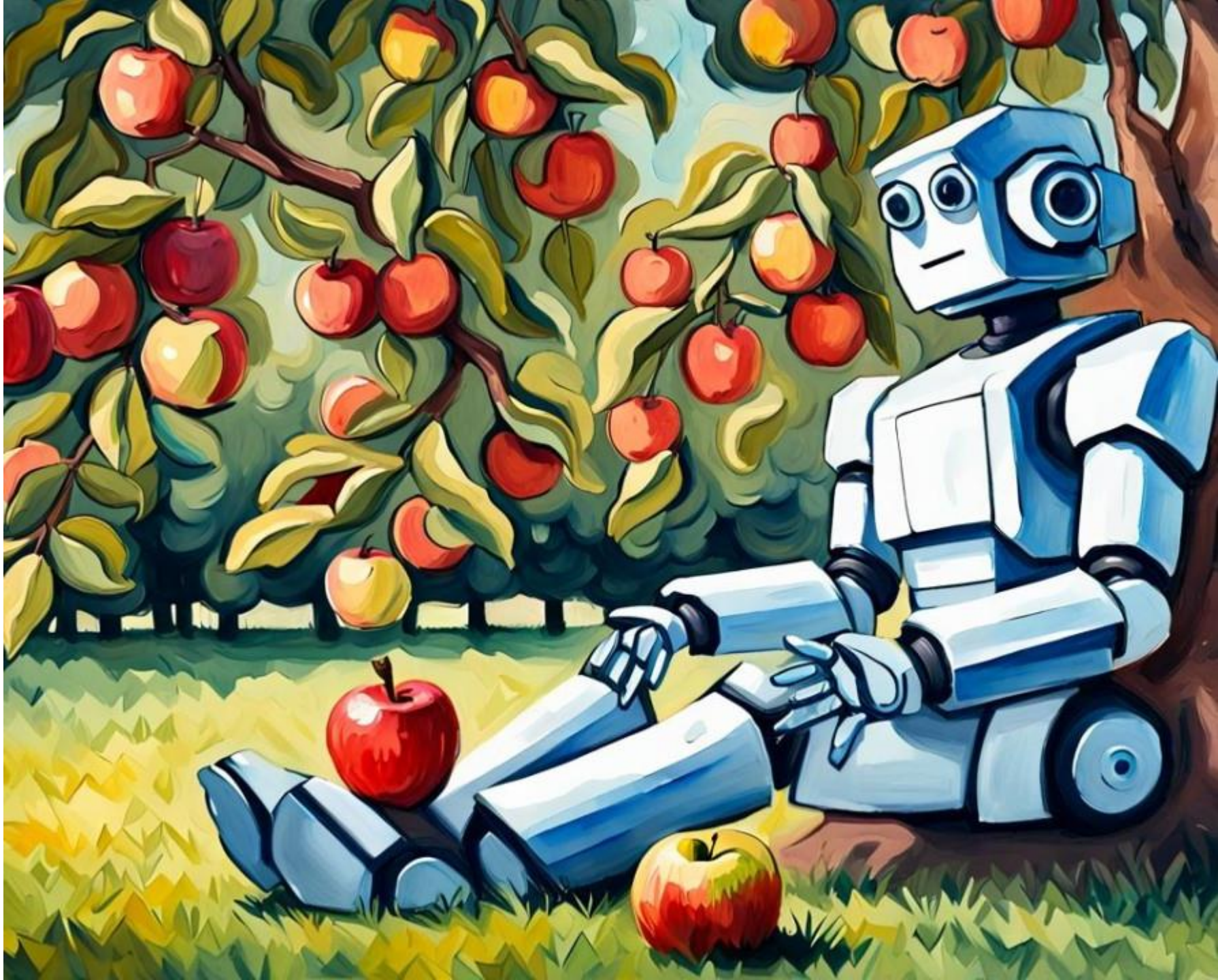
Why Use Interactive Jobs?

- **Test code interactively** before submitting a long-running batch job.
- **Debug issues** on a compute node with live feedback.
- **Run Jupyter Notebooks** or interactive Python sessions.
- **Access GPUs interactively** to test deep learning models.
- **Compile and optimize software** on the compute node.

```
srun [options] –pty bash -l
```

```
salloc [options]
srun <command>
```

Advance SLURM



Job steps

preprocessing

calculations

postprocessing

Job steps

```
#!/bin/bash
#SBATCH -t 10:00:00
#SBATCH -N 1
#SBATCH -n 4
#SBATCH --mem=30g

module load Python

srun -n 1 python preprocessing.py

srun -n 4 python calulations.py

srun -n 1 python postprocessing.py
```

Reuse resources

Job steps

```
#!/bin/bash
#SBATCH -t 10:00:00
#SBATCH -N 1
#SBATCH --gres=gpu:1
#SBATCH --mem=30g

module load Python

srun python preprocessing.py

srun python calulationsGPU.py

srun python postprocessing.py
```

Reuse resources

Multistep jobs

Multistep jobs

```
#!/bin/bash
#SBATCH -t 10:00:00
#SBATCH -N 1
#SBATCH -n 4
#SBATCH --mem=30g

module [illegible]hon

srun -l -n 1 python calculations.py input1 &
srun -l -n 1 python calculations.py input2 &
srun -l -n 1 python calculations.py input3 &
srun -l -n 1 python calculations.py input4 &

wait
```

Process ID in outputs

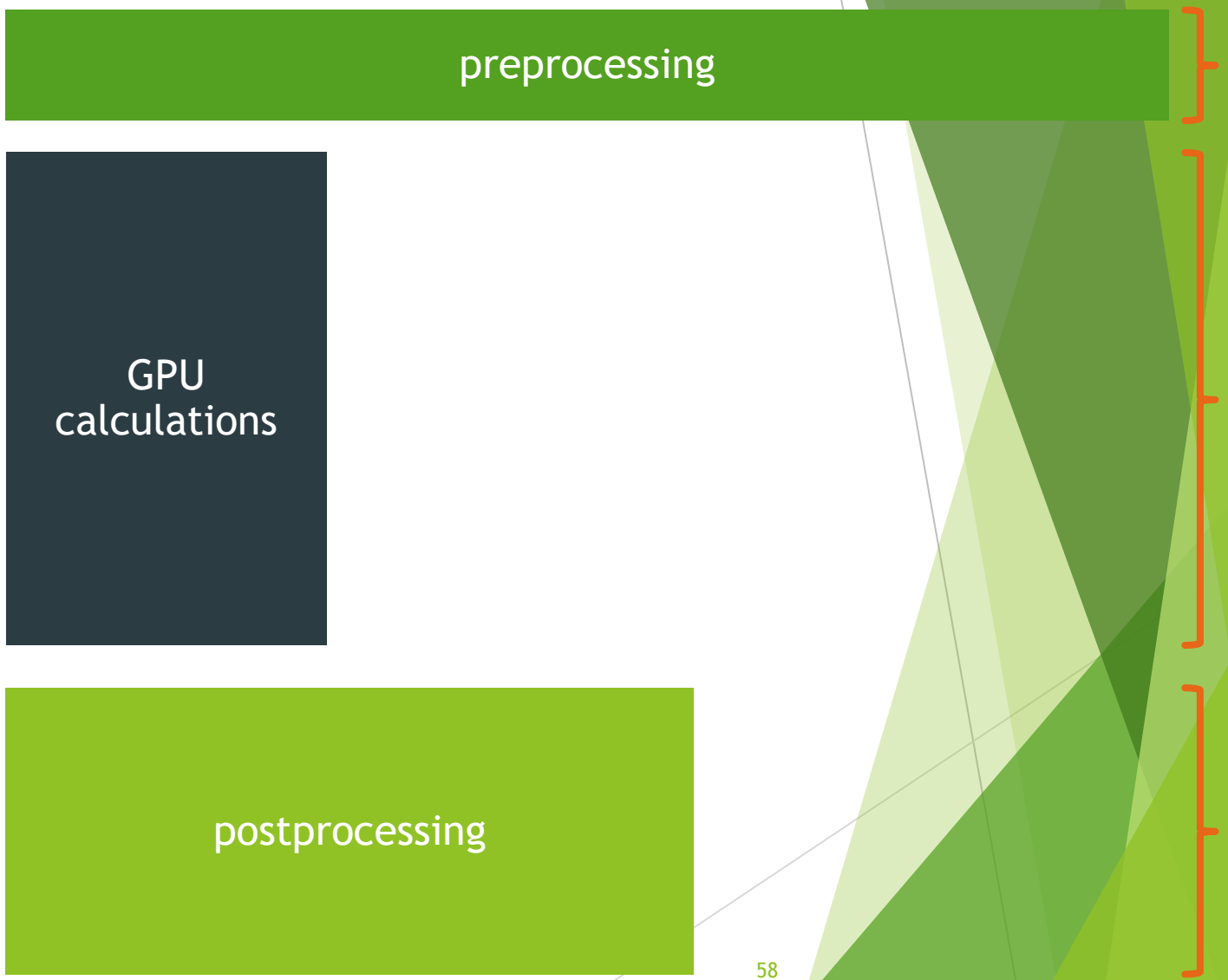
Run parallel calculations

Multistep jobs

```
#!/bin/bash
#SBATCH -t 10:00:00
#SBATCH -N 1
#SBATCH -n 4
#SBATCH -gres=gpu:1
#SBATCH --mem=30g

module load Python

nvidia-cuda-mps-control -d

srun -l -n 1 python calculationsGPU.py input1 &
srun -l -n 1 python calculationsGPU.py input2 &
srun -l -n 1 python calculationsGPU.py input3 &
srun -l -n 1 python calculationsGPU.py input4 &

wait
```

Run parallel on a GPU

Job dependencies



Types of dependencies

The types of dependencies:

- `after:jobid[:jobid...]` job can begin after the specified jobs have started
- `afterany:jobid[:jobid...]` job can begin after the specified jobs have terminated
- `afternotok:jobid[:jobid...]` job can begin after the specified jobs have failed
- `afterok:jobid[:jobid...]` job can begin after the specified jobs have run to completion successful
- `singleton` jobs can begin execution after all previously launched jobs with the same name and user have ended.

```
--dependency=<type:job_id[:job_id][,type:job_id[:job_id]
```

Job dependencies

```
$sbatch ./prerun.sh
Submitted batch job 112211
$sbatch --dependency=afterok:112211 ./run.sh
…
$squeue -me
```

Job array

calculations input1	calculations input2	calculations input3	calculations input4
calculations input5	calculations input6	calculations input7	calculations input8
calculations input9	calculations input10	calculations input11	calculations input12
calculations input13	calculations input14	calculations input15	calculations input16

Job array

```
#!/bin/bash -l
#SBATCH --ntasks=4
#SBATCH --error=output_%A_%a.err
#SBATCH --output=output_%A_%a.out
#SBATCH --time=00:30:00
#SBATCH --mem=10g
#SBATCH -N 1
#SBATCH --array=10-25

module load Python

srun -l -n 1 python calculations.py \
         input_$SLURM_ARRAY_TASK_ID
```

Job ID

Array index

TASK ID

Run parallel processing

Job array

```
#!/bin/bash -l
#SBATCH --ntasks=4
#SBATCH --time=00:30:00
#SBATCH --mem=10g
#SBATCH -N 1

#SLURM_ARRAY_TASK_ID=10
```

```
-l
asks=4
me=00:30:00
m=10g

#SLURM_ARRAY_TASK_ID=11
```

```
-l
tasks=4
me=00:30:00
em=10g

#SLURM_ARRAY_TASK_ID=12
```

```
#!/bin/bash -l
#SBATCH --ntasks=4
#SBATCH --time=00:30:00
#SBATCH --mem=10g
#SBATCH -N 1
```

```
#!/bin/bash -l
#SBATCH --ntasks=4
#SBATCH --time=00:30:00
#SBATCH --mem=10g
#SBATCH -N 1

#SLURM_ARRAY_TASK_ID=25
```

Job array

Array ranges:

`--array=1-10`	from 1 to 10 (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
`--array=1,4-6,10`	1, 4 to 6, then 10 (1, 4, 5, 6, 10)
`--array=0-16:5`	Every 5th value from 0 to 16 (0, 5, 10, 15)
`--array=0-1999%100`	No more than 100 running at any one time

Bash arrays:

```
params=(input1a input4b in_2024)
${params[${SLURM_ARRAY_TASK_ID}]}
```

Multistep job array

Run parallel chunks

- ```
  #SBATCH --array=0-999
  myjob ${SLURM_ARRAY_TASK_ID}
  ```
- ```
  #SBATCH --array=0-999:4
  myjob $(( SLURM_ARRAY_TASK_ID ))
  myjob $(( SLURM_ARRAY_TASK_ID + 1 ))
  myjob $(( SLURM_ARRAY_TASK_ID + 2 ))
  myjob $(( SLURM_ARRAY_TASK_ID + 3 ))
  ```
- ```
  #SBATCH --array=0-999:4
  for i in {0..3}; do
   myjob $(( SLURM_ARRAY_TASK_ID + i ))
  done
  ```
- ```
  #SBATCH --array=0-1000:4
  for i in {0..3}; do
   index=$(( SLURM_ARRAY_TASK_ID + i ))
   if [$index -le $SLURM_ARRAY_TASK_MAX]; then
      myjob $index
   fi
  done
  ```