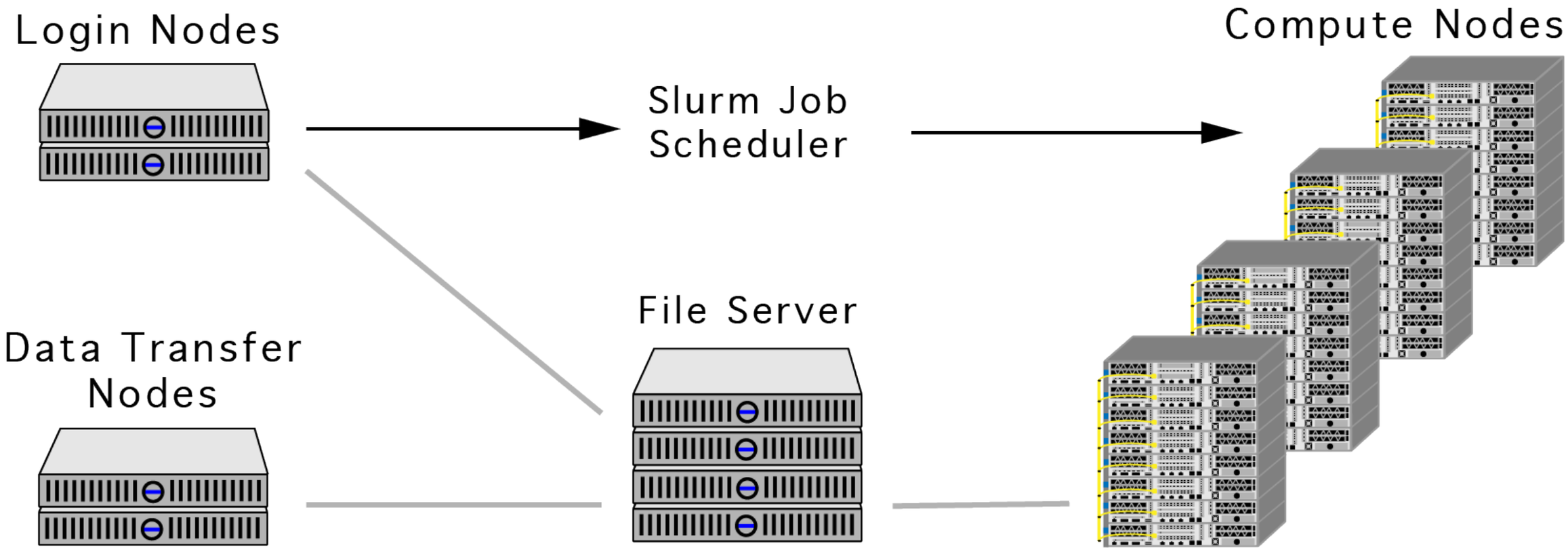
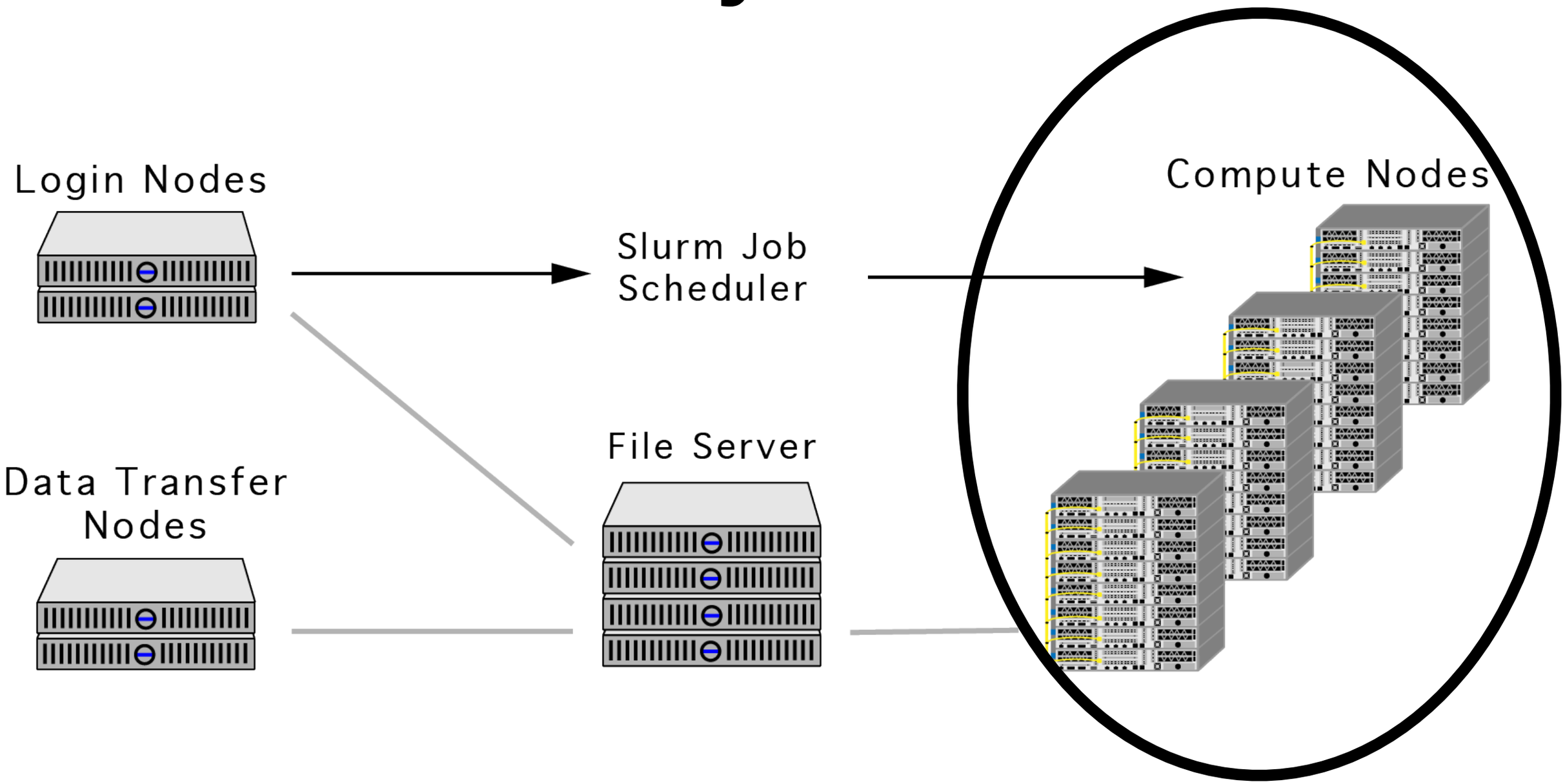
Lecture 9: Compute nodes and the scheduler

CEE 690

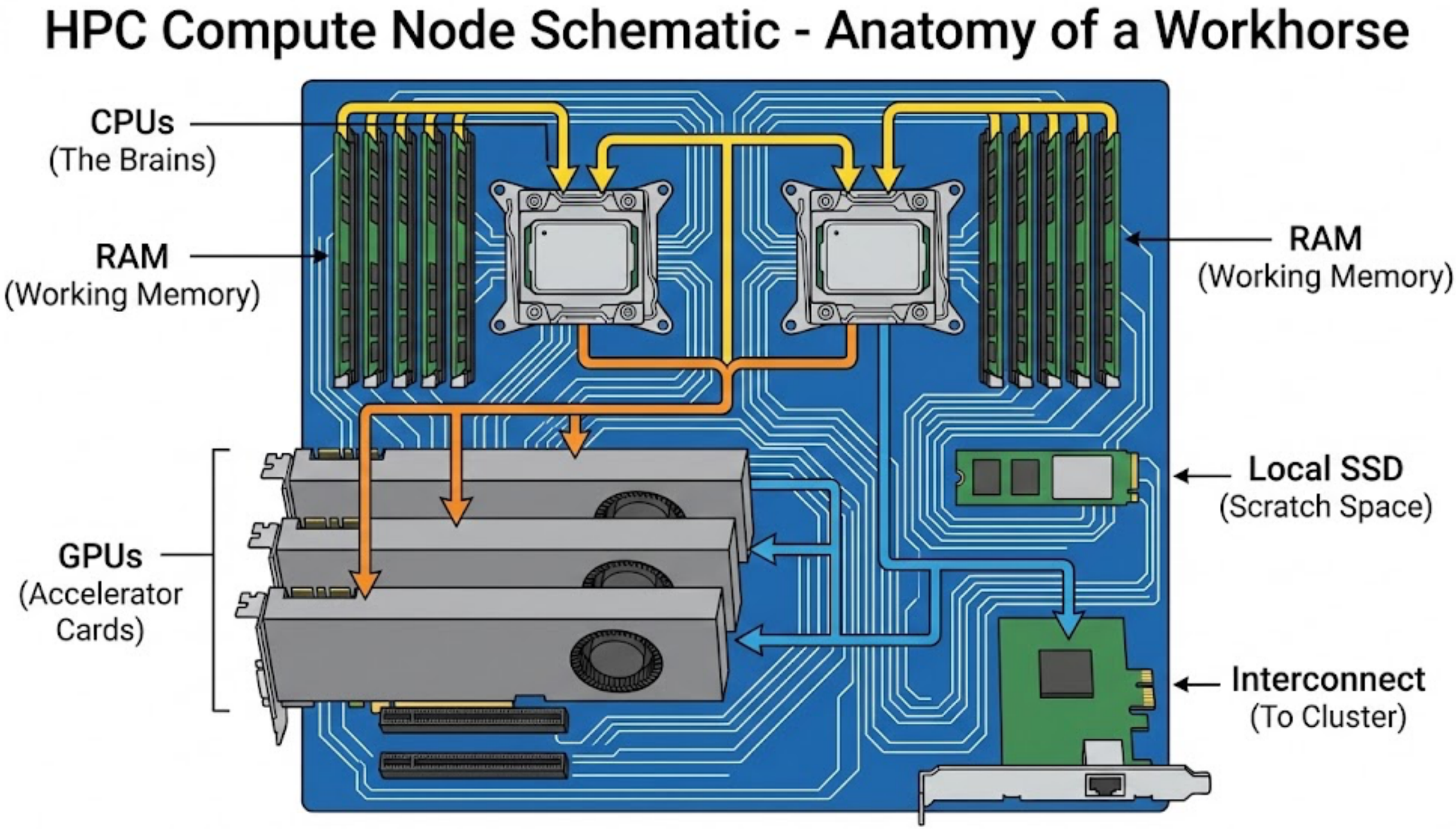
How are they structured?



How are they structured?

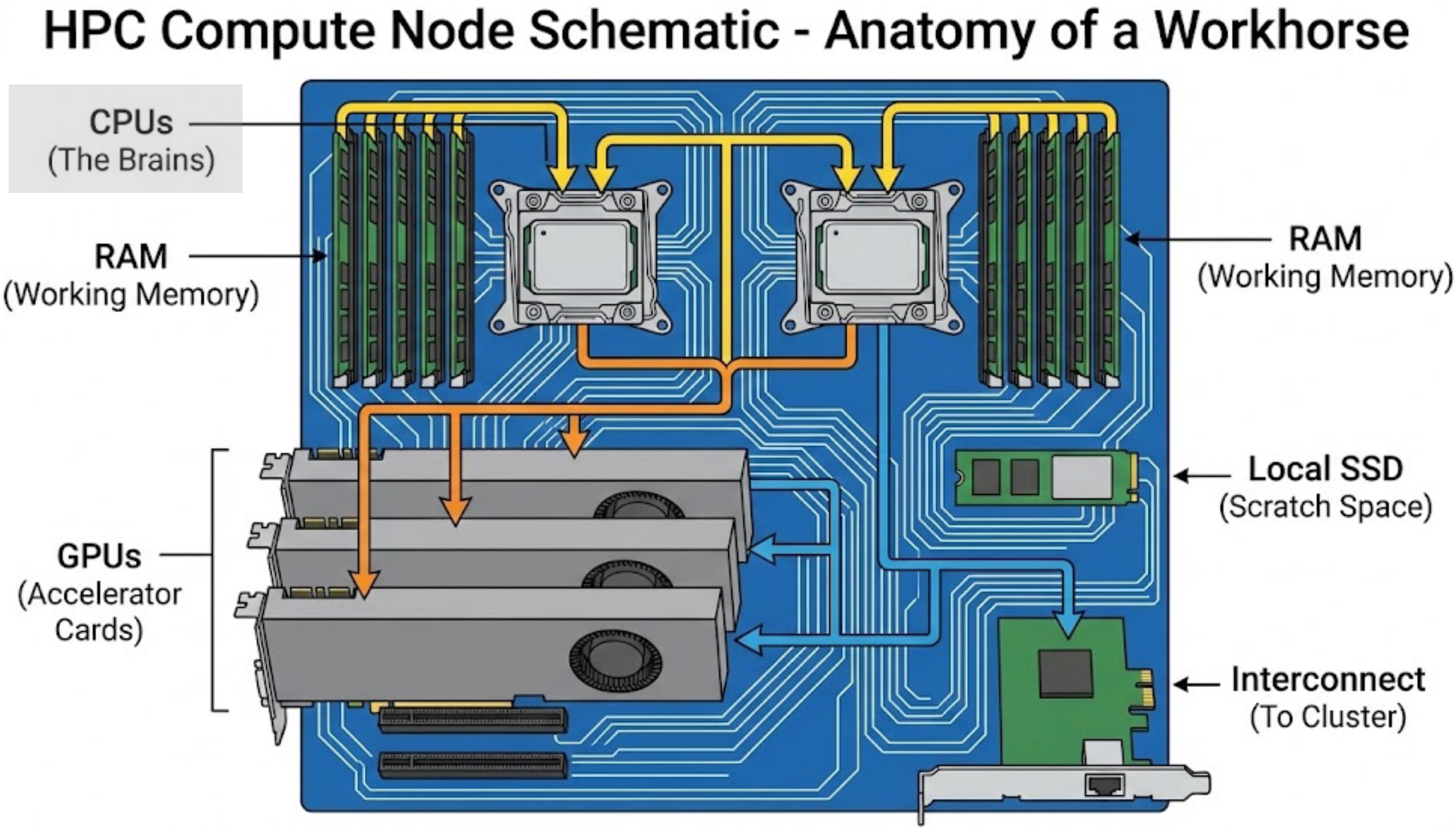


Compute nodes



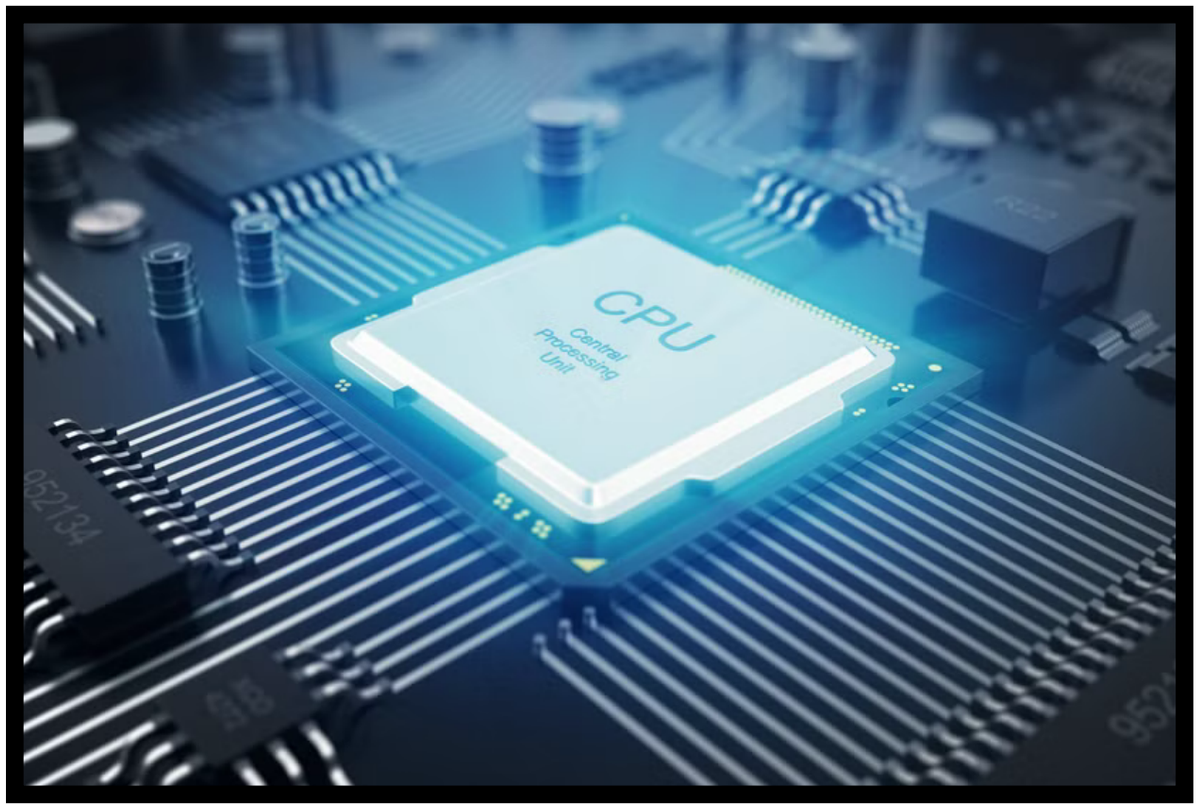
This is where your code is actually being executed.

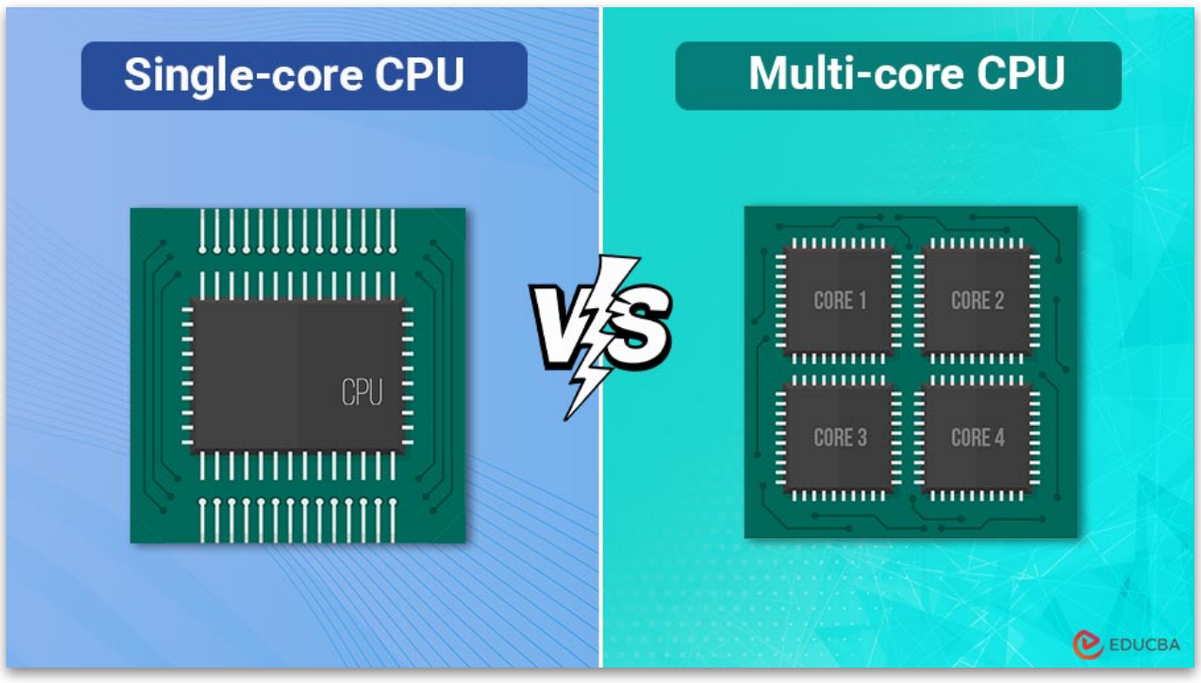
Compute nodes



This is where your code is actually being executed.

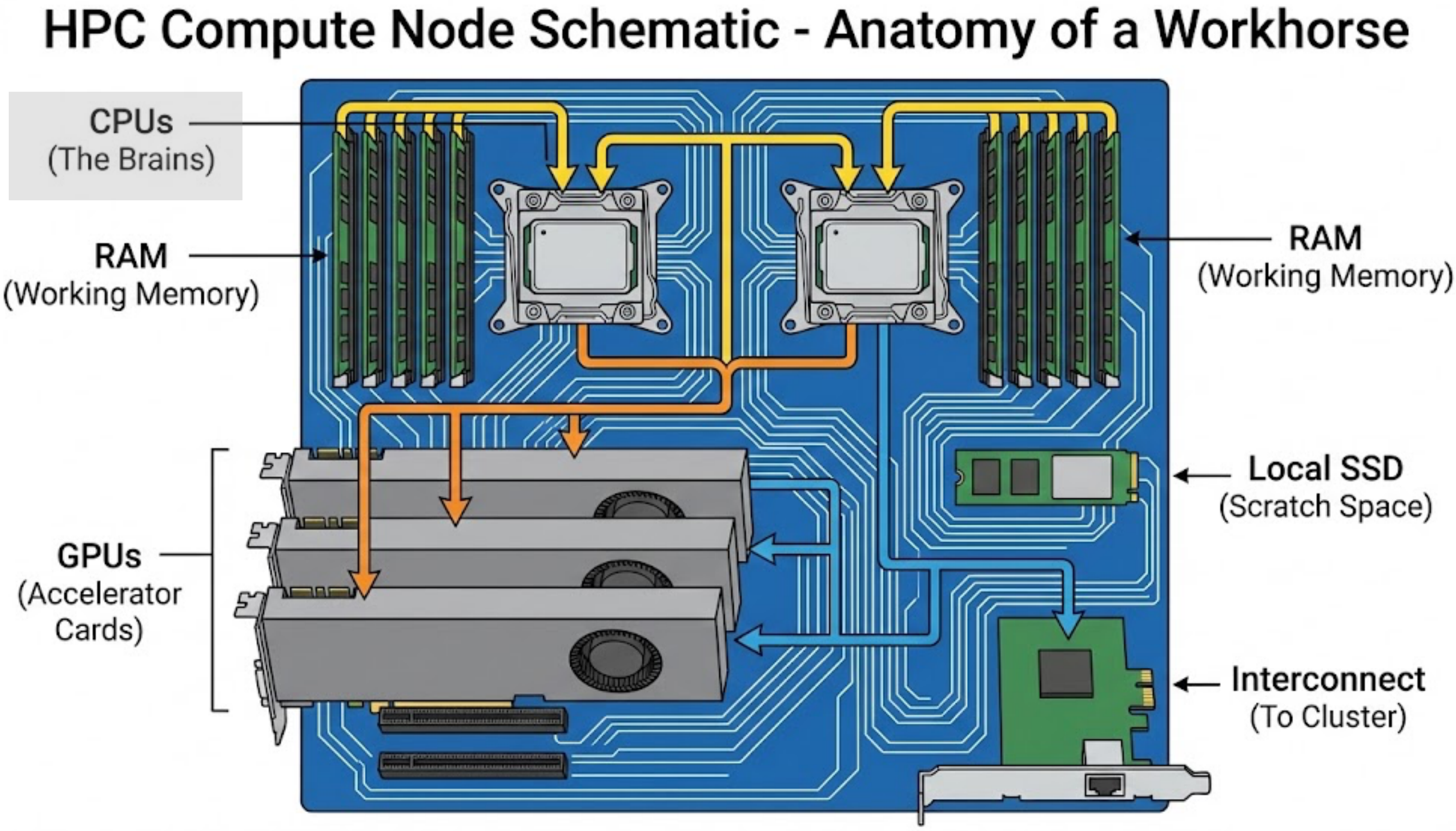
CPU (Central Processing Unit)





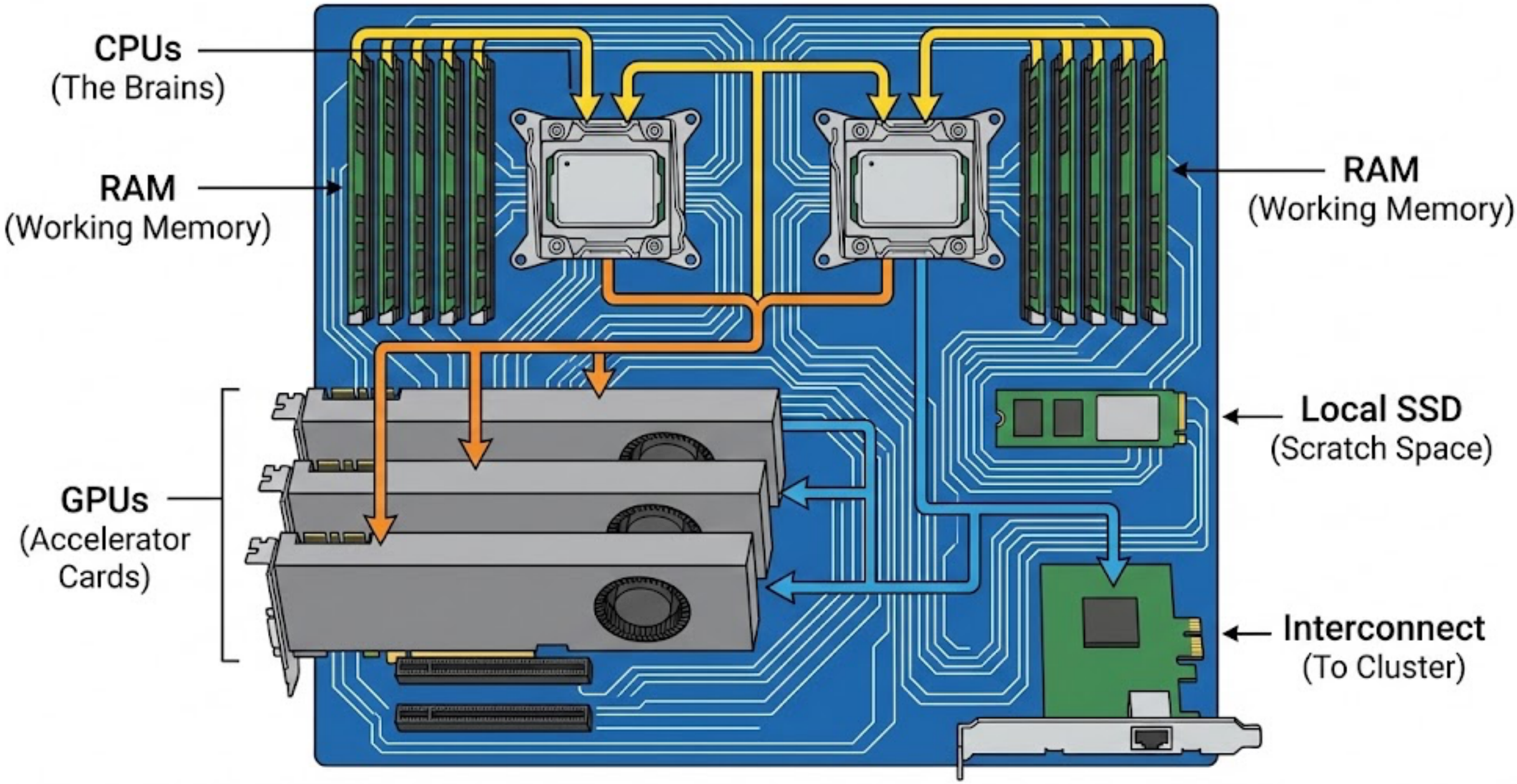
- Purpose: General of the army (tells everyone what to do)
- CPUs are not meant to do the repetitive tasks (nowadays)
- Each CPU can have many cores (and each core usually has two threads)

Multiple sockets = Multiple CPUs = Lots of cores = Lots and lots of threads

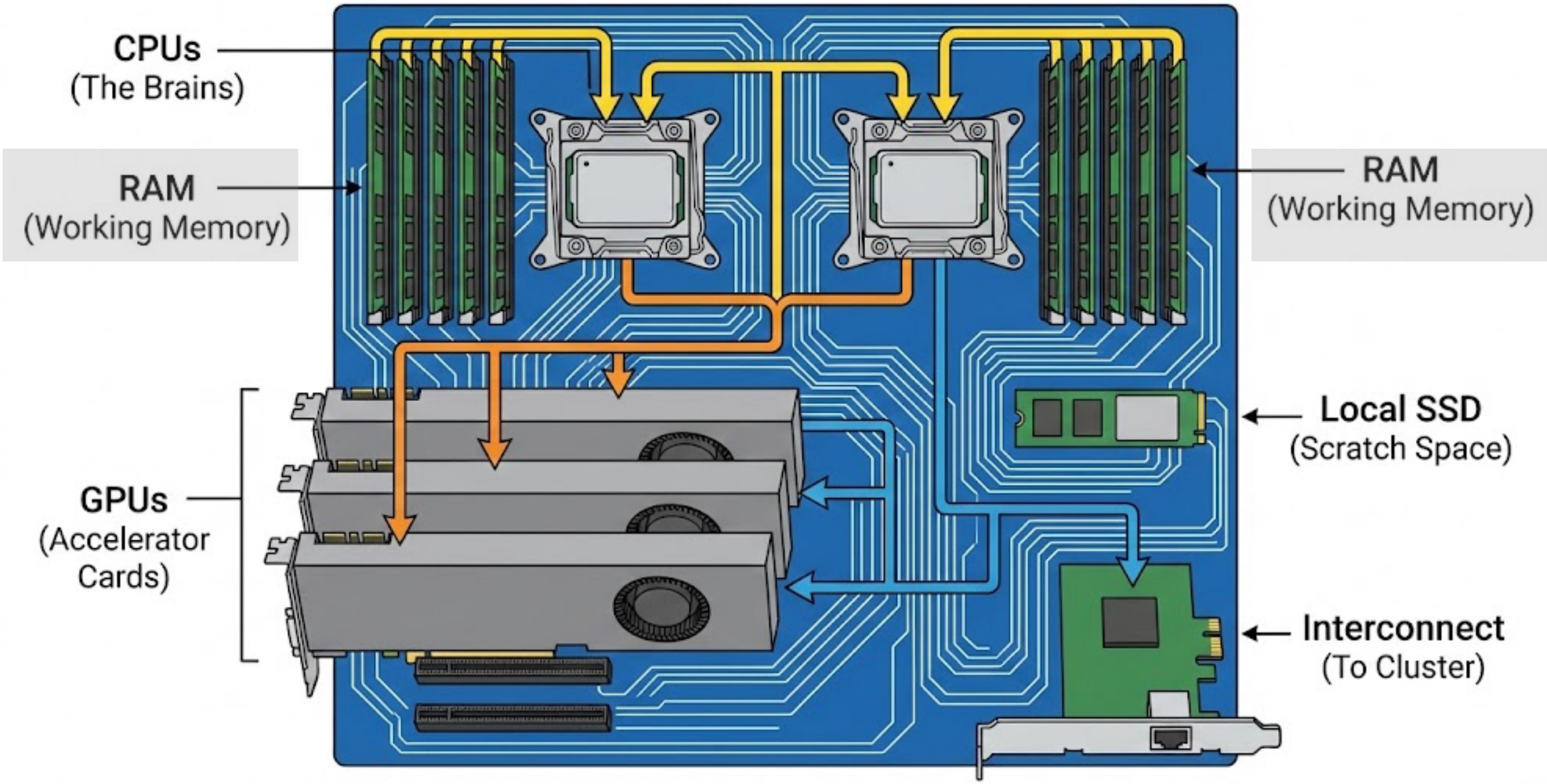


This is how you end up with 32-128 threads on a node

HPC Compute Node Schematic - Anatomy of a Workhorse

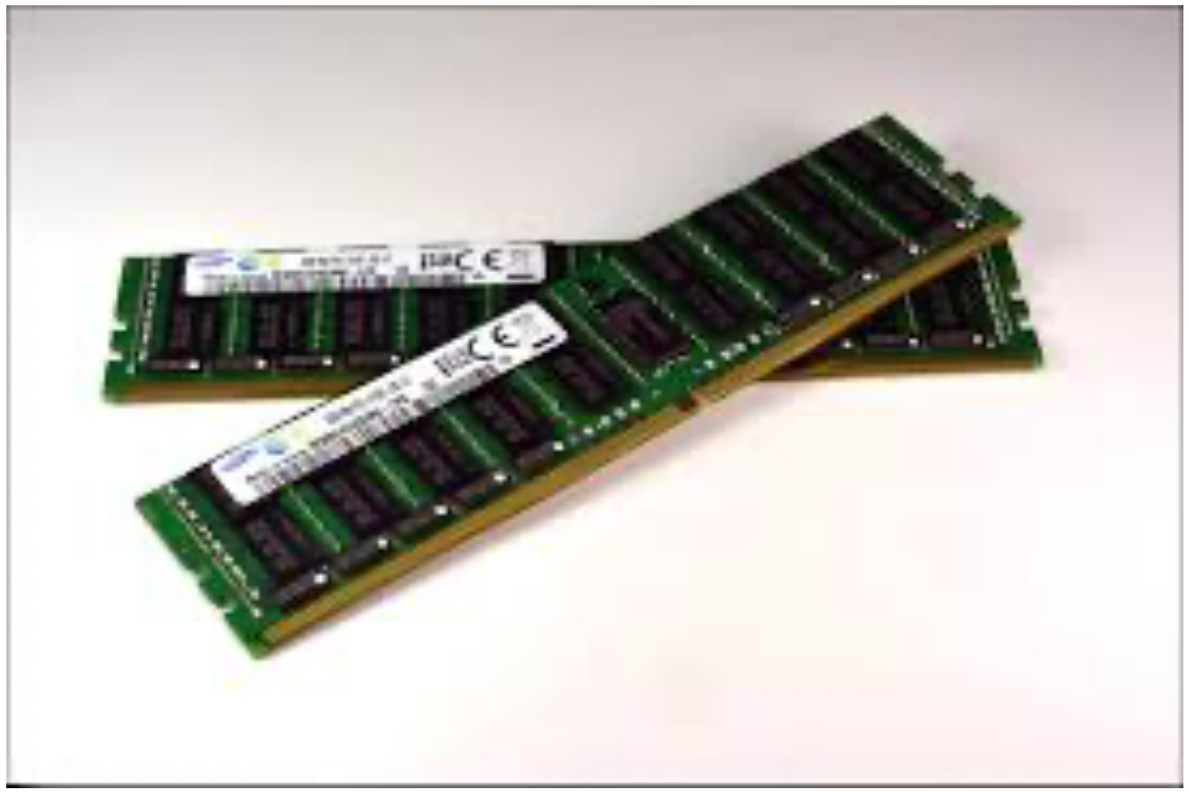


HPC Compute Node Schematic - Anatomy of a Workhorse



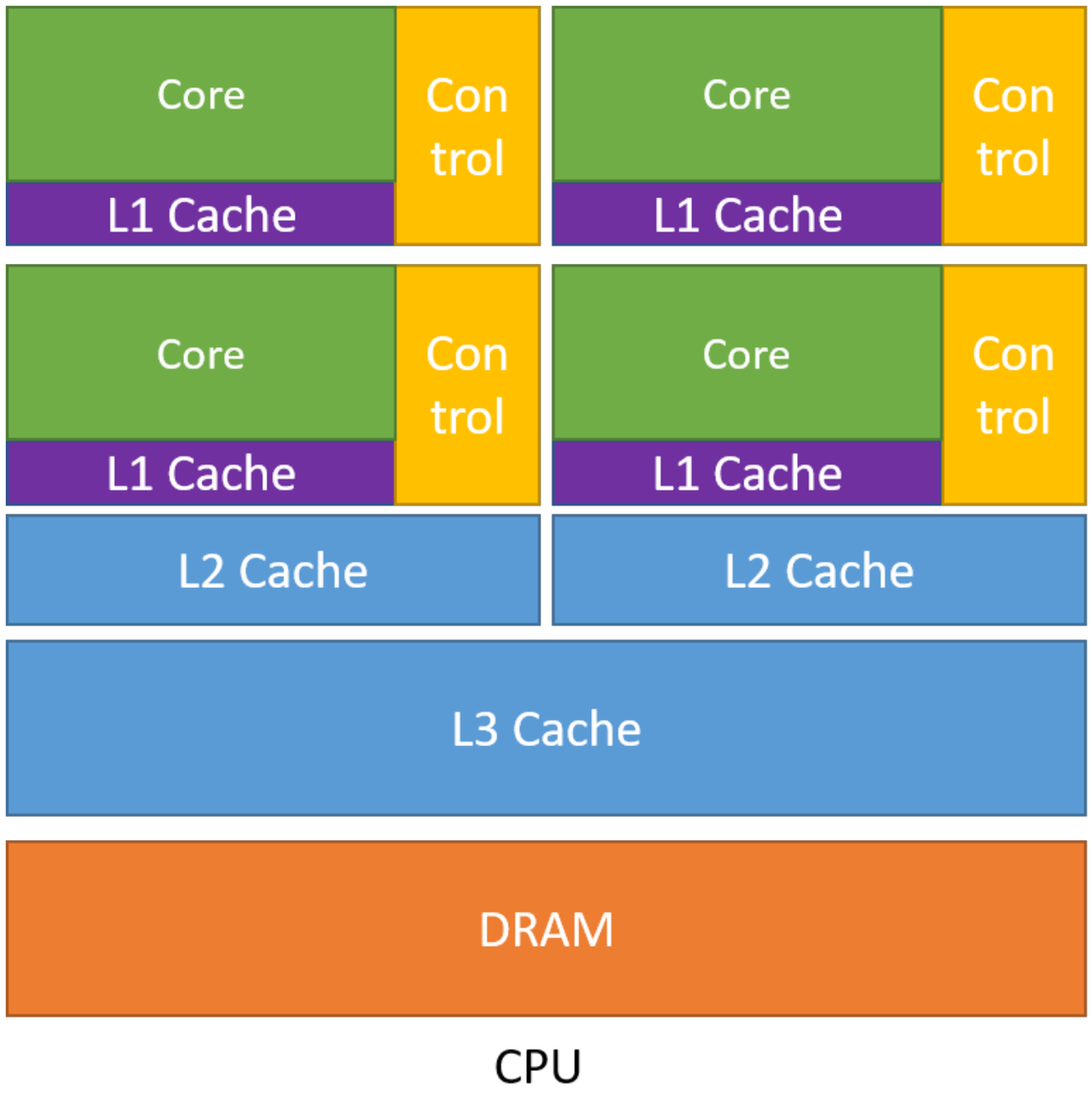
RAM (Random Access Memory)

- **Purpose** - “The filesystem is your library and the RAM is your desk”
- Generally 2-4 GB of ram per CPU core
- I/O from RAM is much faster than storage.
- You can’t store anything permanently on RAM (requires power).



Memory Type	Actual Latency	Human Scale Time
Main RAM	~100 ns	**3.3 minutes**
SSD (NVMe)	~10,000 ns	**5.5 hours**
Hard Drive	~10,000,000 ns	**4 months**

L1/L2/L3 cache



- L1 - 32-80 KB/core
- L2 - 1-2 MB/core
- L3 - 256-512 MB/socket

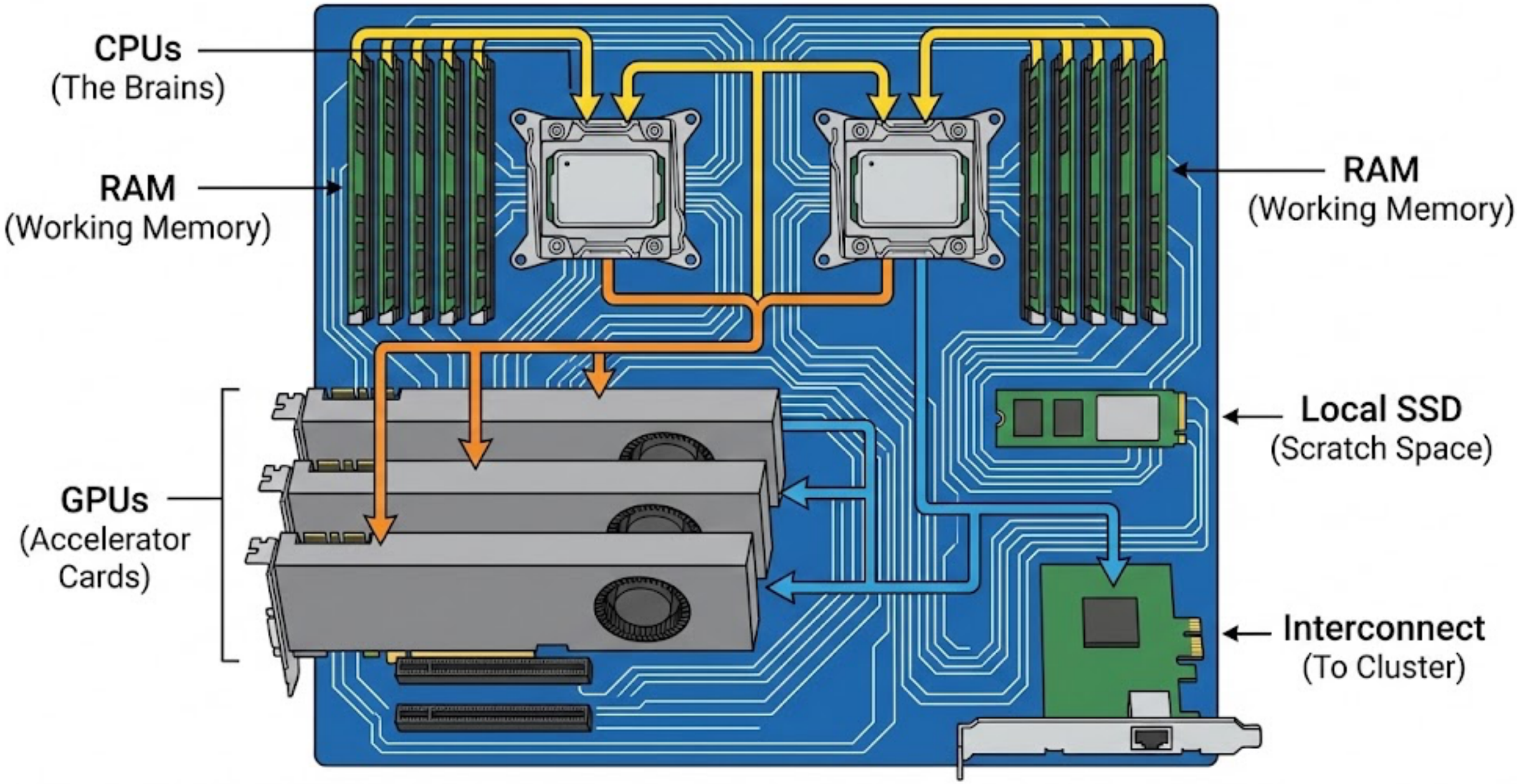
Memory Type	Actual Latency
L1 Cache	~0.5 ns
L2 Cache	~7 ns
L3 Cache	~20 ns
Main RAM	~100 ns
SSD (NVMe)	~10,000 ns
Hard Drive	~10,000,000 ns

How to leverage RAM and cache?

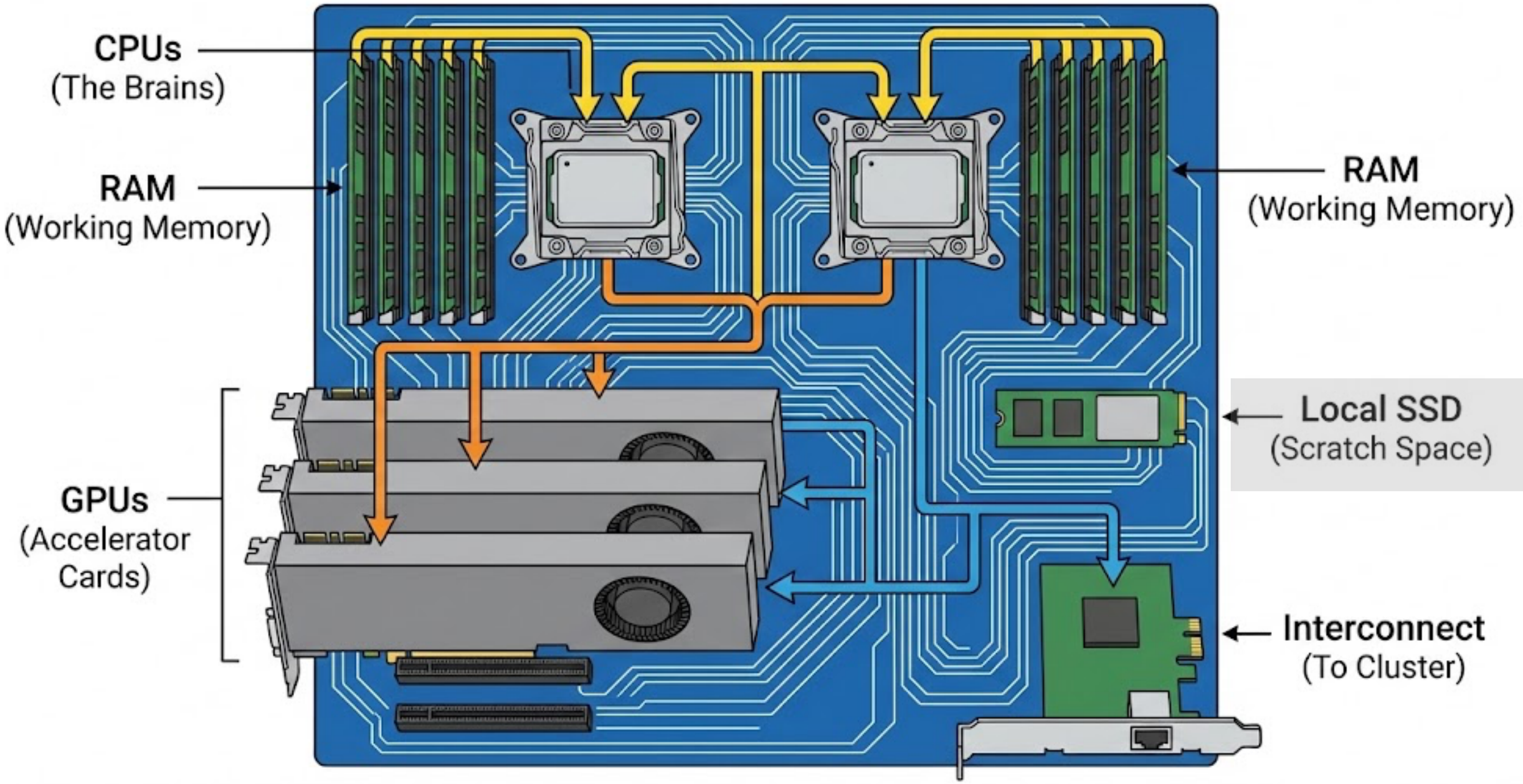
1. Minimize accessing the file system. Load as much as makes sense into RAM.
2. Break up into smaller chunks that fit into the cache (focusing on L3 cache is sufficient).
3. Ensure that the data you are working with is focused only on what you need for your problem.
4. Access data as it was stored (the last dimension is the fastest in Python).

Caution: If you get to this level of optimization, you are probably overthinking your code speed-up.

HPC Compute Node Schematic - Anatomy of a Workhorse

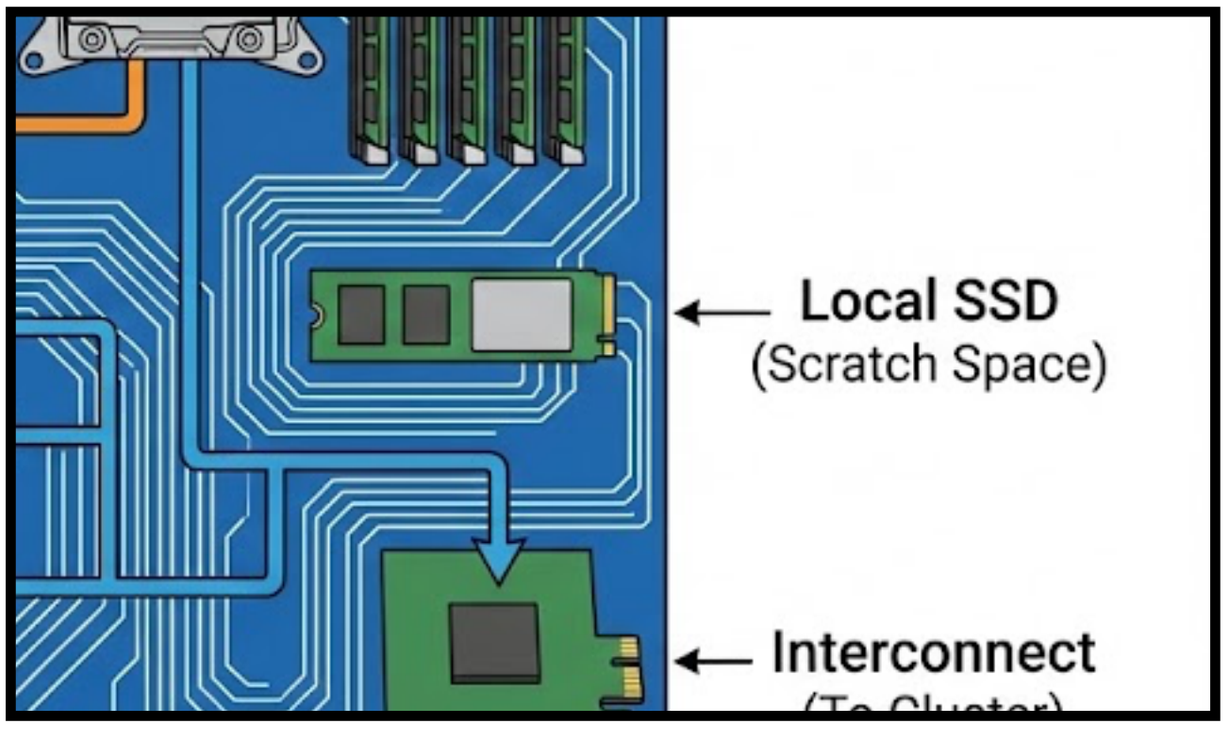


HPC Compute Node Schematic - Anatomy of a Workhorse

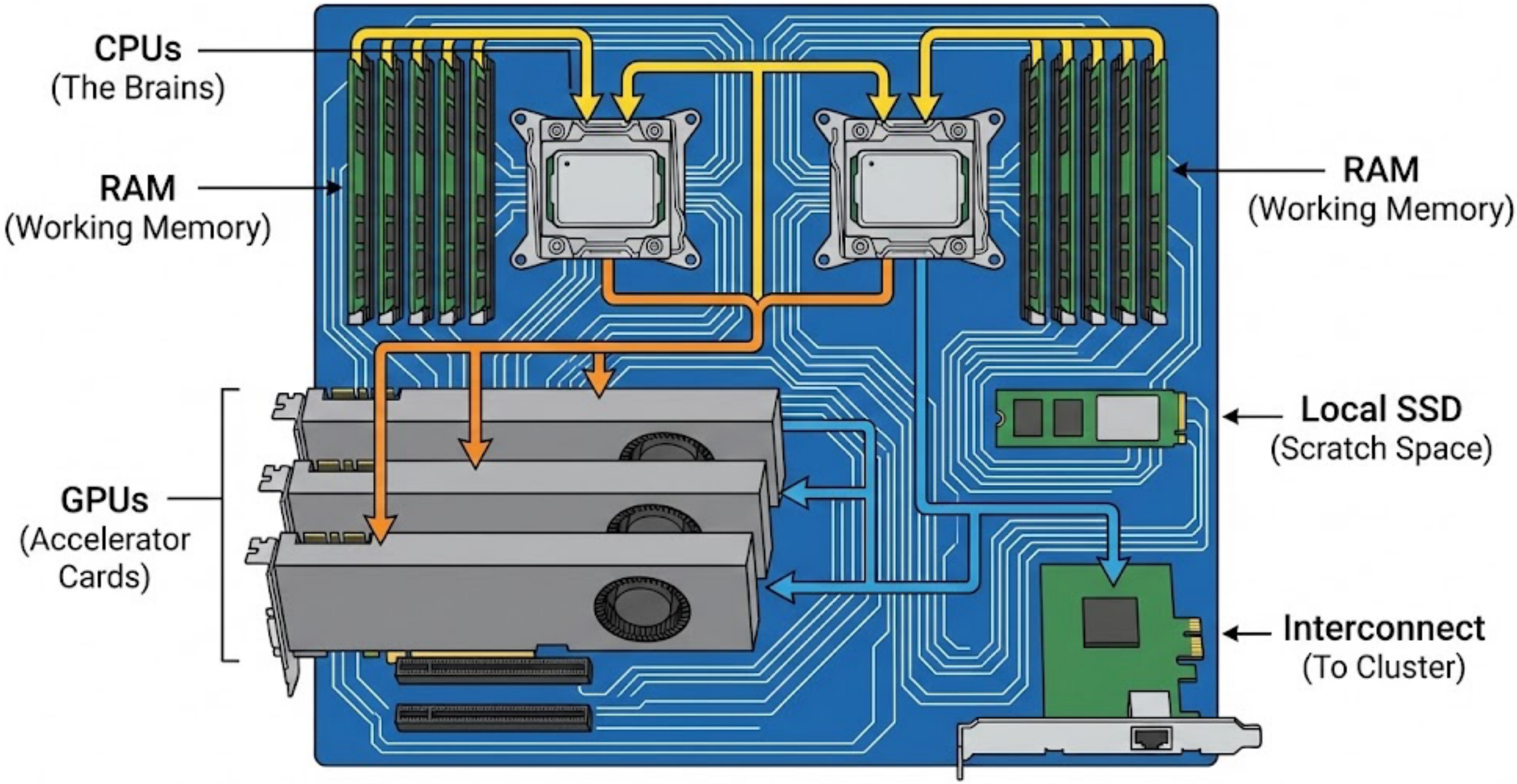


Using node SSD as storage

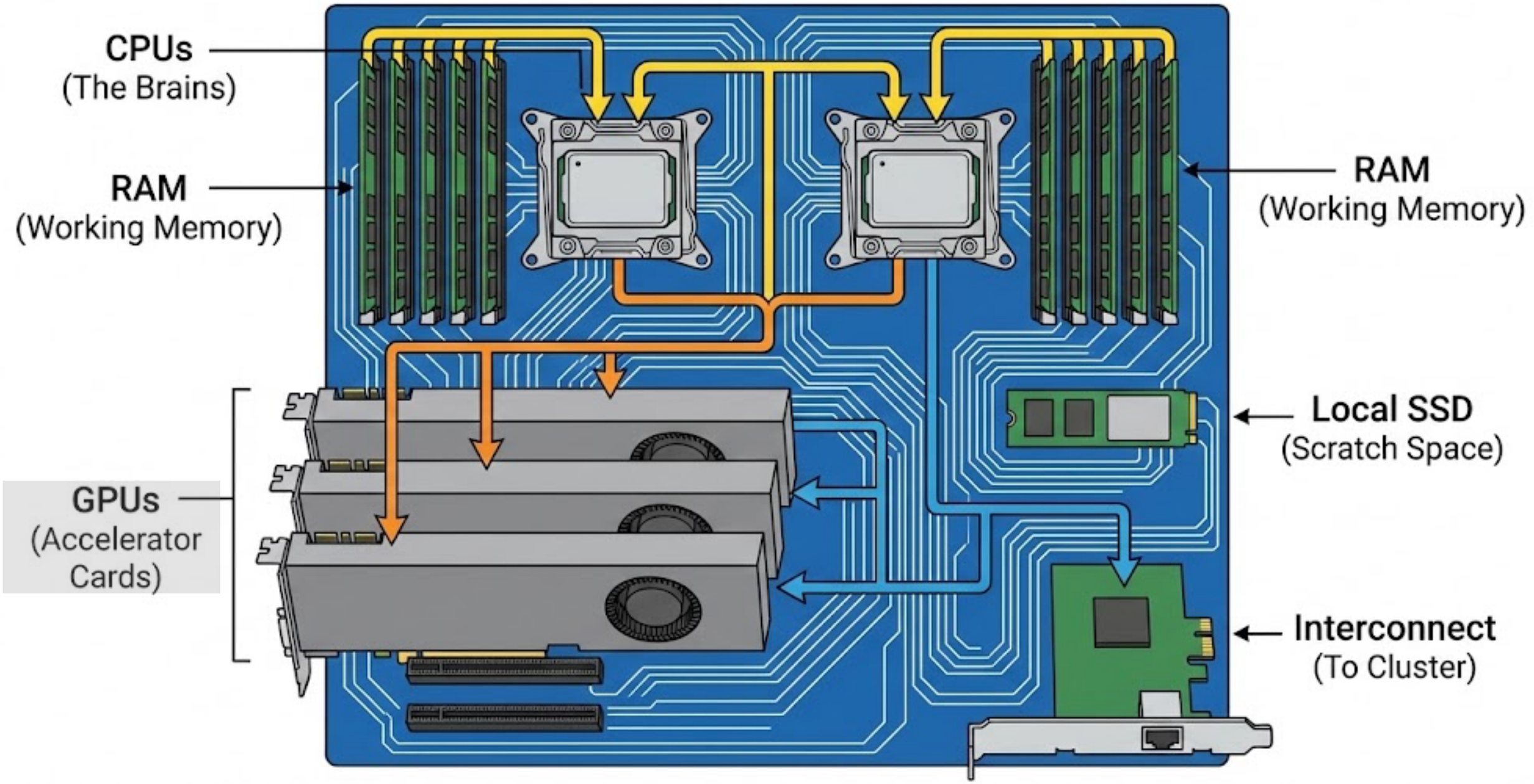
- Each node will have it own SSD. This is generally used almost exclusively for the node’s operating system.
- Sometimes, there will be a scratch directory on the node SSD as well (1 TB?). If you can find out a way to use it as temporary storage, it can help speed things up (best of all worlds).



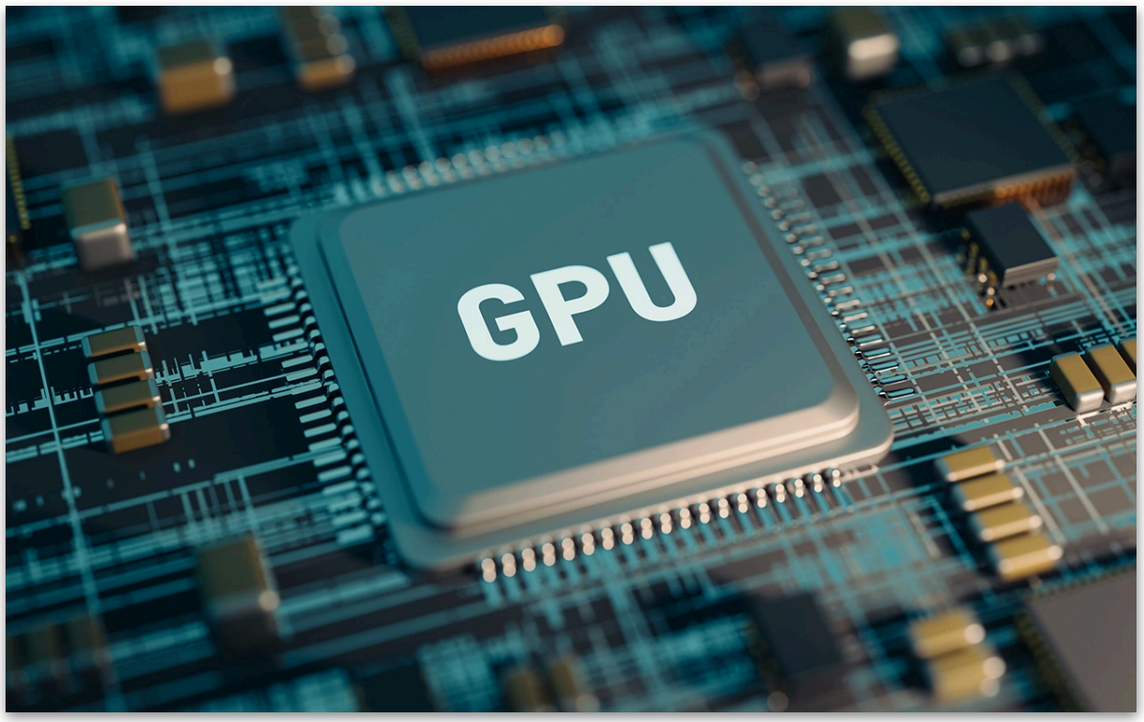
HPC Compute Node Schematic - Anatomy of a Workhorse

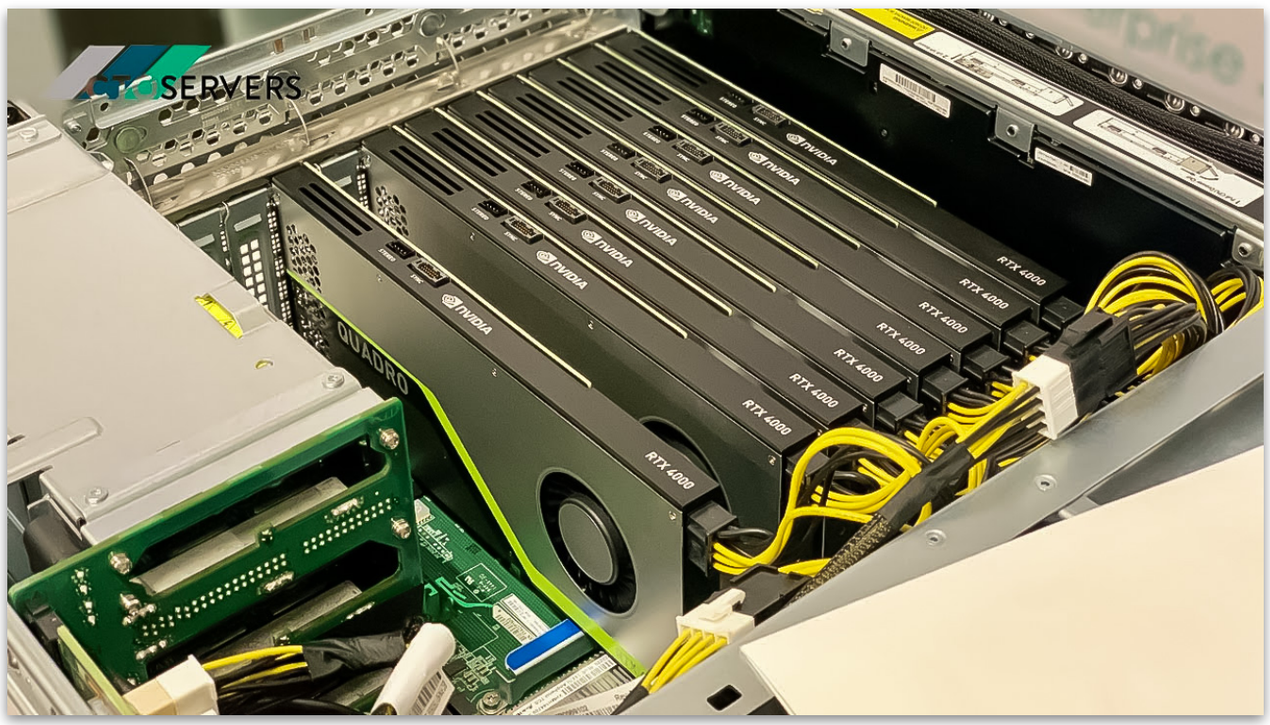


HPC Compute Node Schematic - Anatomy of a Workhorse



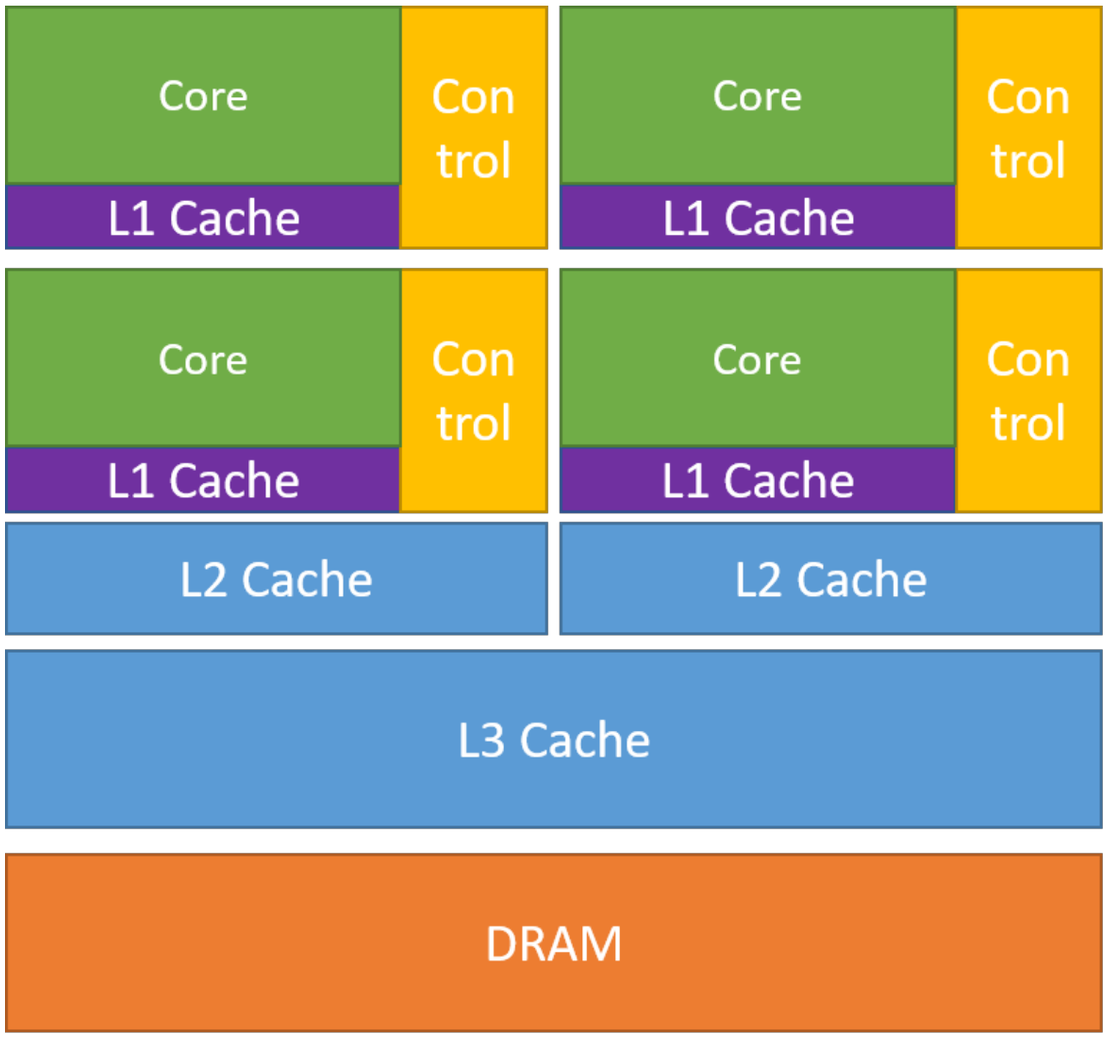
GPU (Graphical Processing Unit)



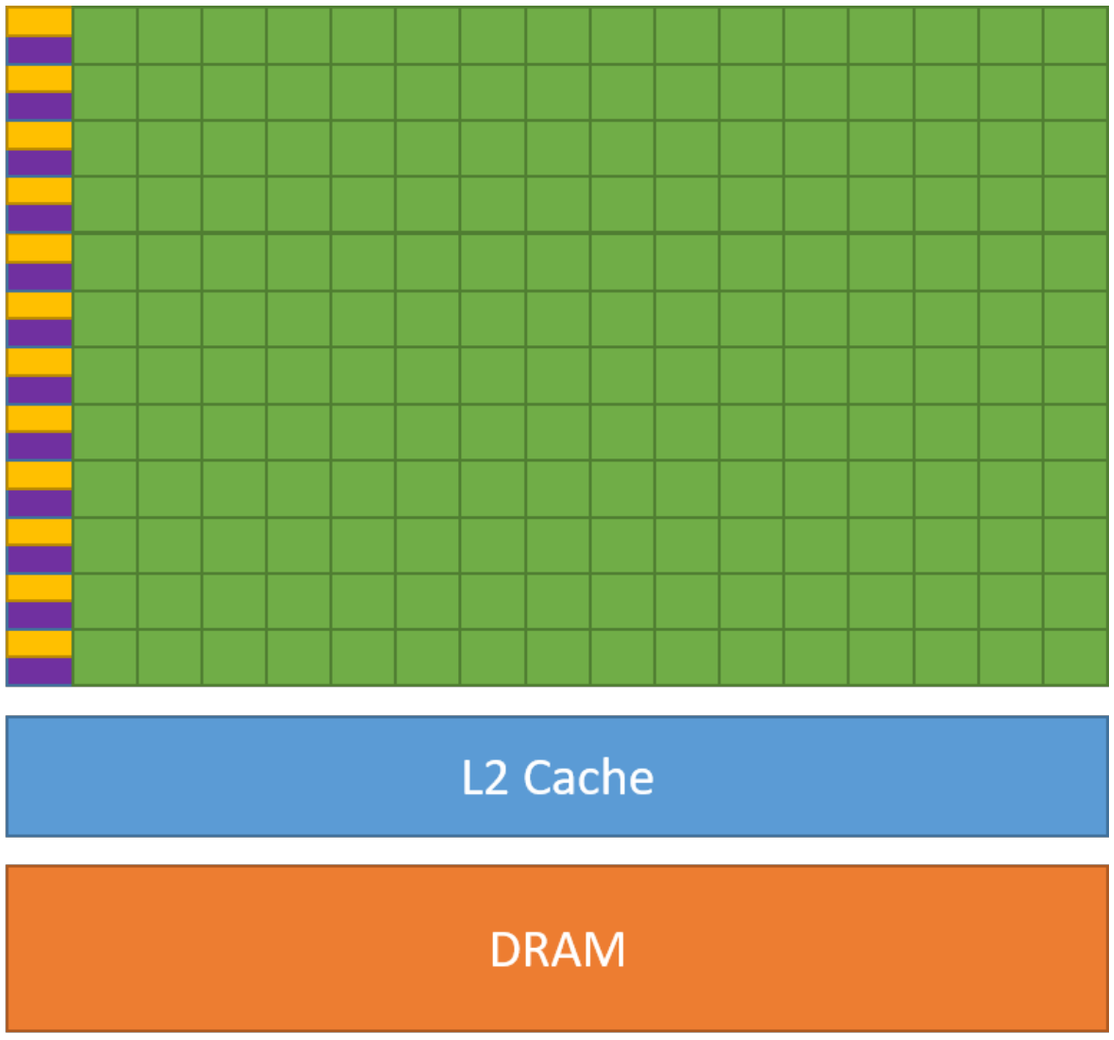


- Purpose: Soldiers in the army.
- Very good at single instruction/multiple data.
- Getting data onto the GPU's VRAM is the largest bottleneck.

CPU vs GPU



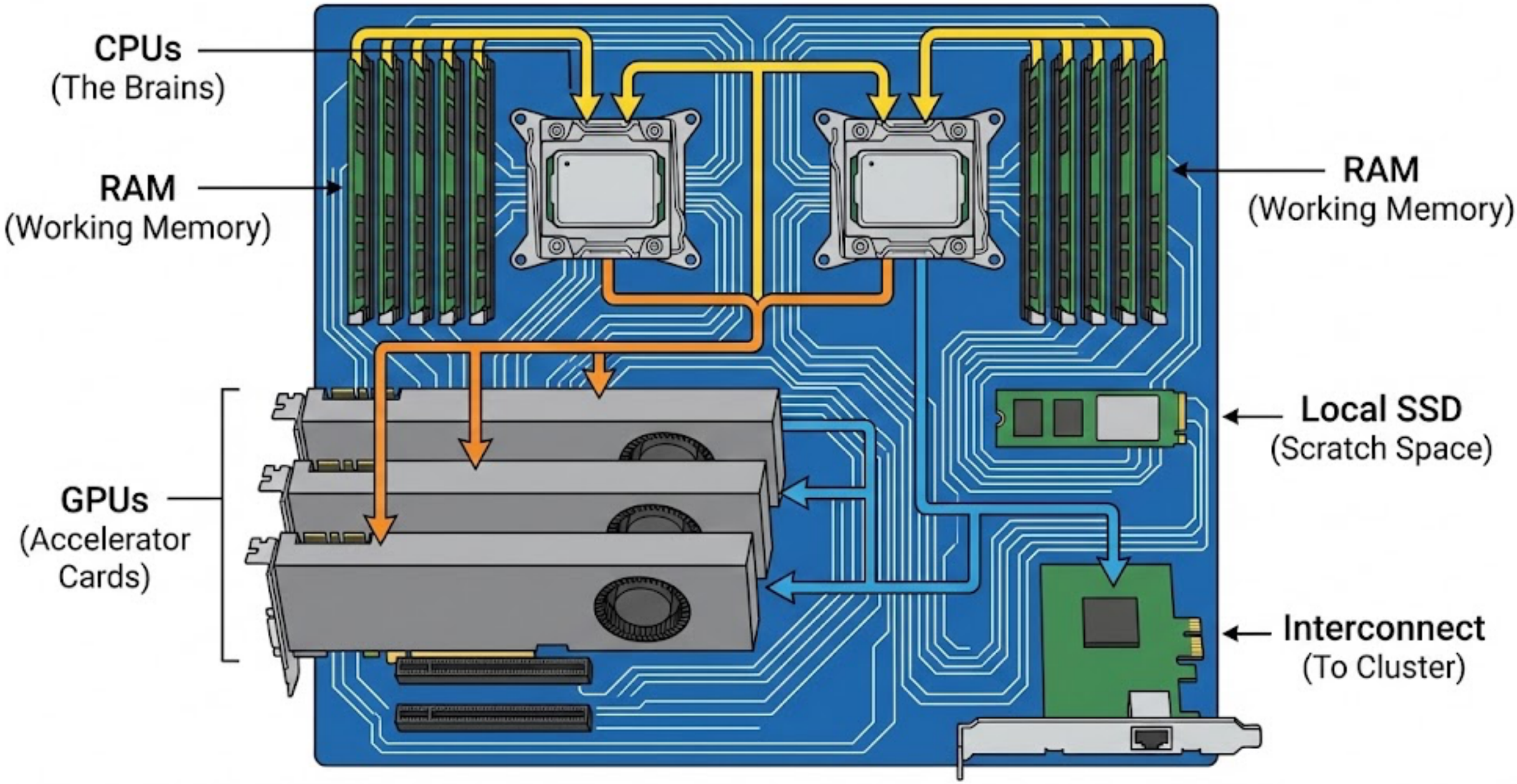
CPU



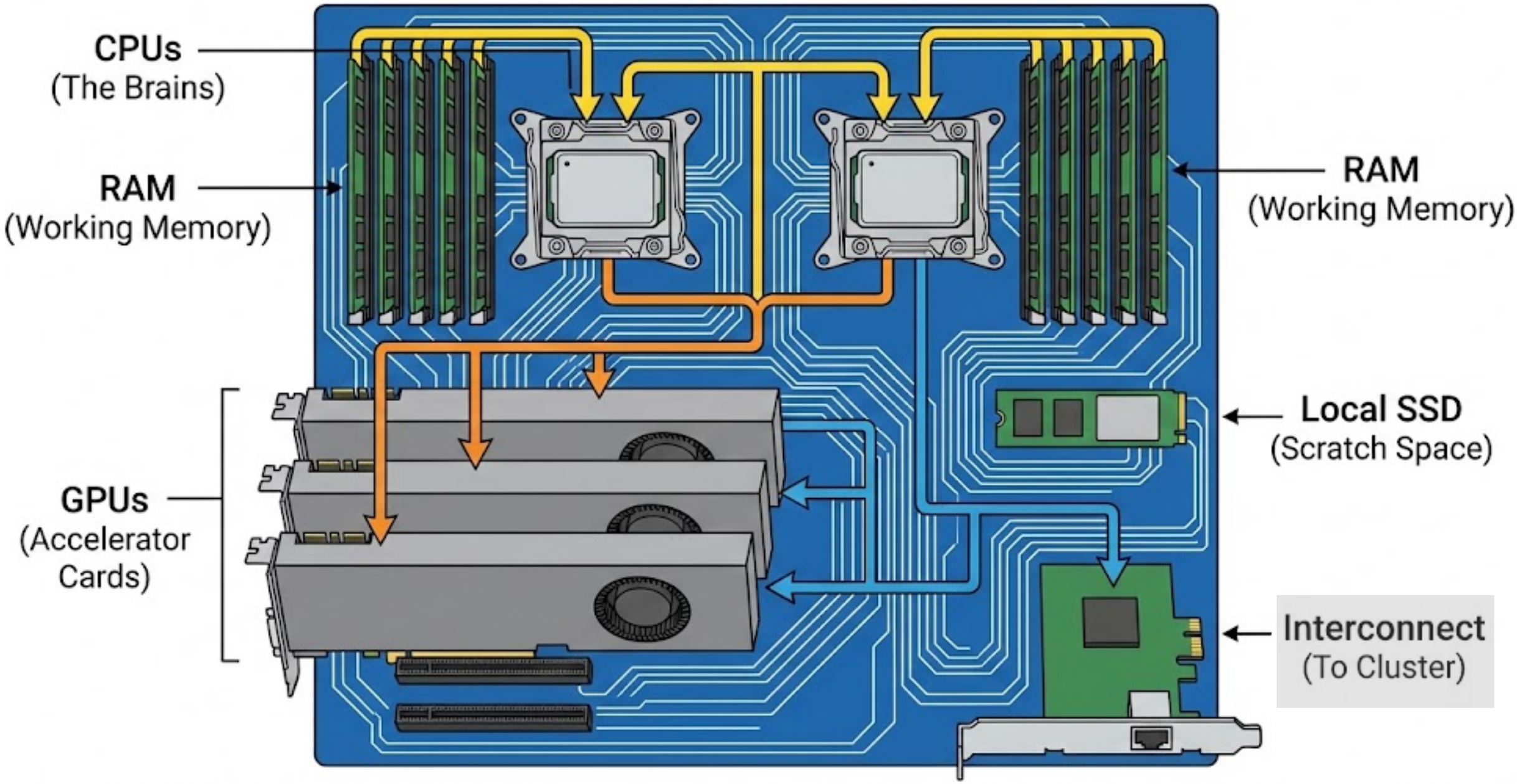
GPU

- The control manages the logistics. The cores do the math.
- CPUs have 10-100 of cores. GPUs have 1000s of cores.

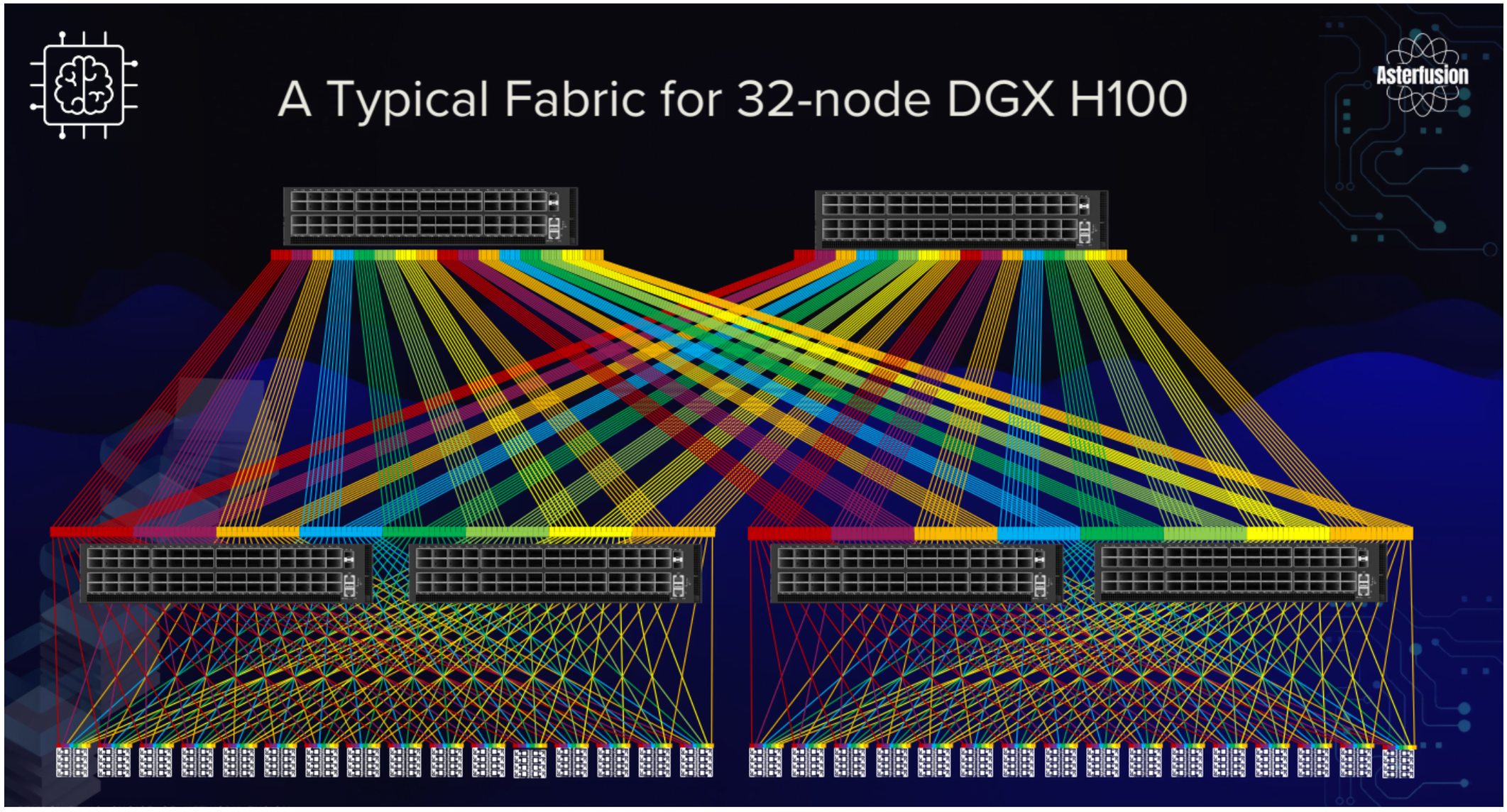
HPC Compute Node Schematic - Anatomy of a Workhorse



HPC Compute Node Schematic - Anatomy of a Workhorse



Interconnect



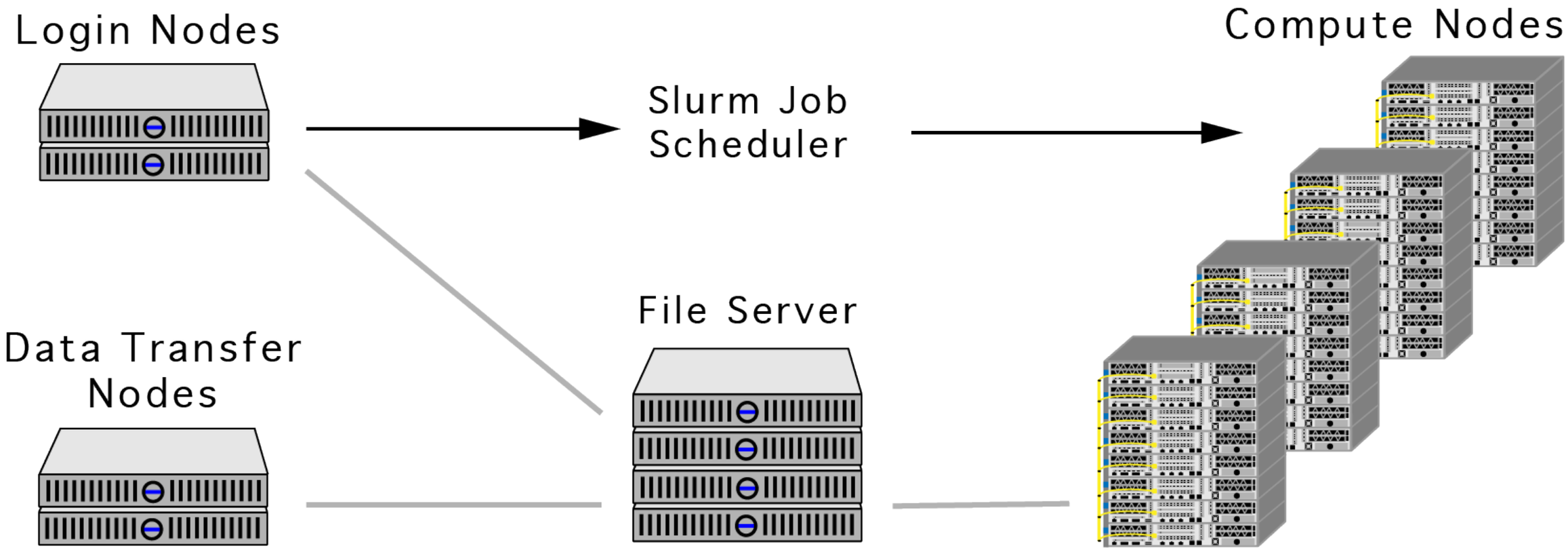
To read/write to/from filesystem and to have compute nodes interact, we need a very fast interconnect system

Does the interconnect matter?

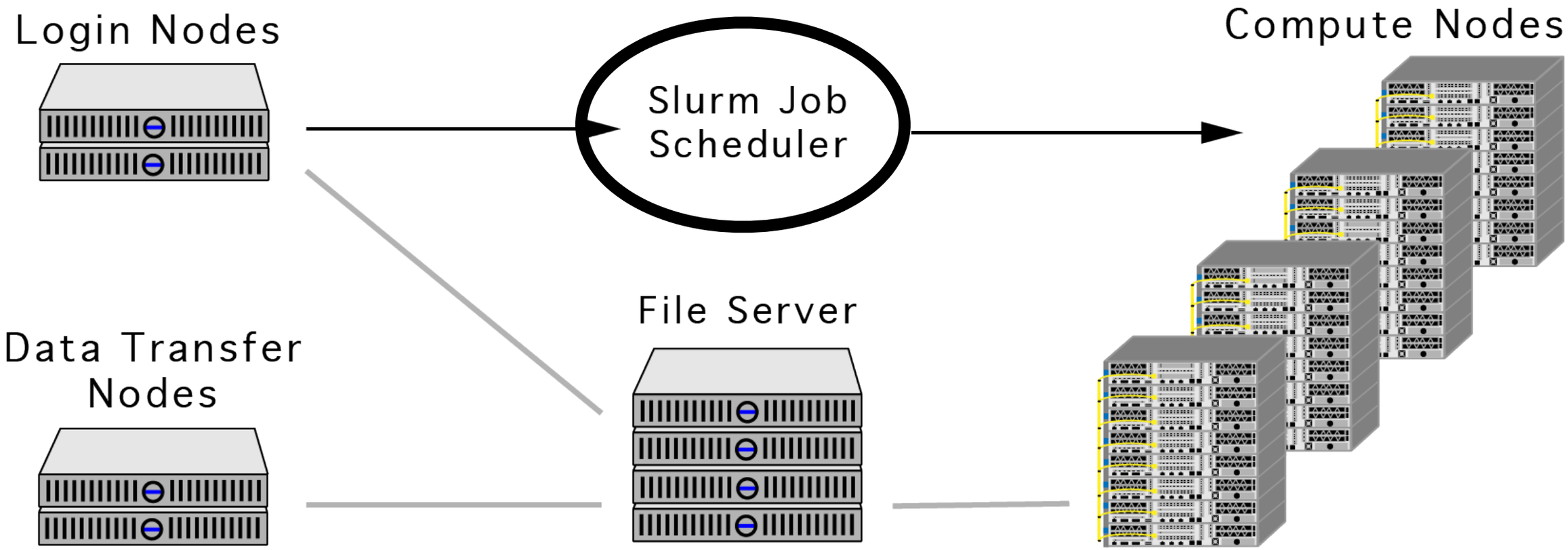
Feature	Ethernet	InfiniBand	Omni-Path
Throughput	Moderate	**Extreme**	High
Latency	Poor (High)	**Excellent (Lowest)**	Excellent (Low)
CPU Load	**High**	**Very Low**	**Medium**
Best Use	Moving data to/ from the internet.	Large CPU/GPU clusters	Large CPU-only clusters.

HPC will use Infiniband (or maybe Omni-Path)

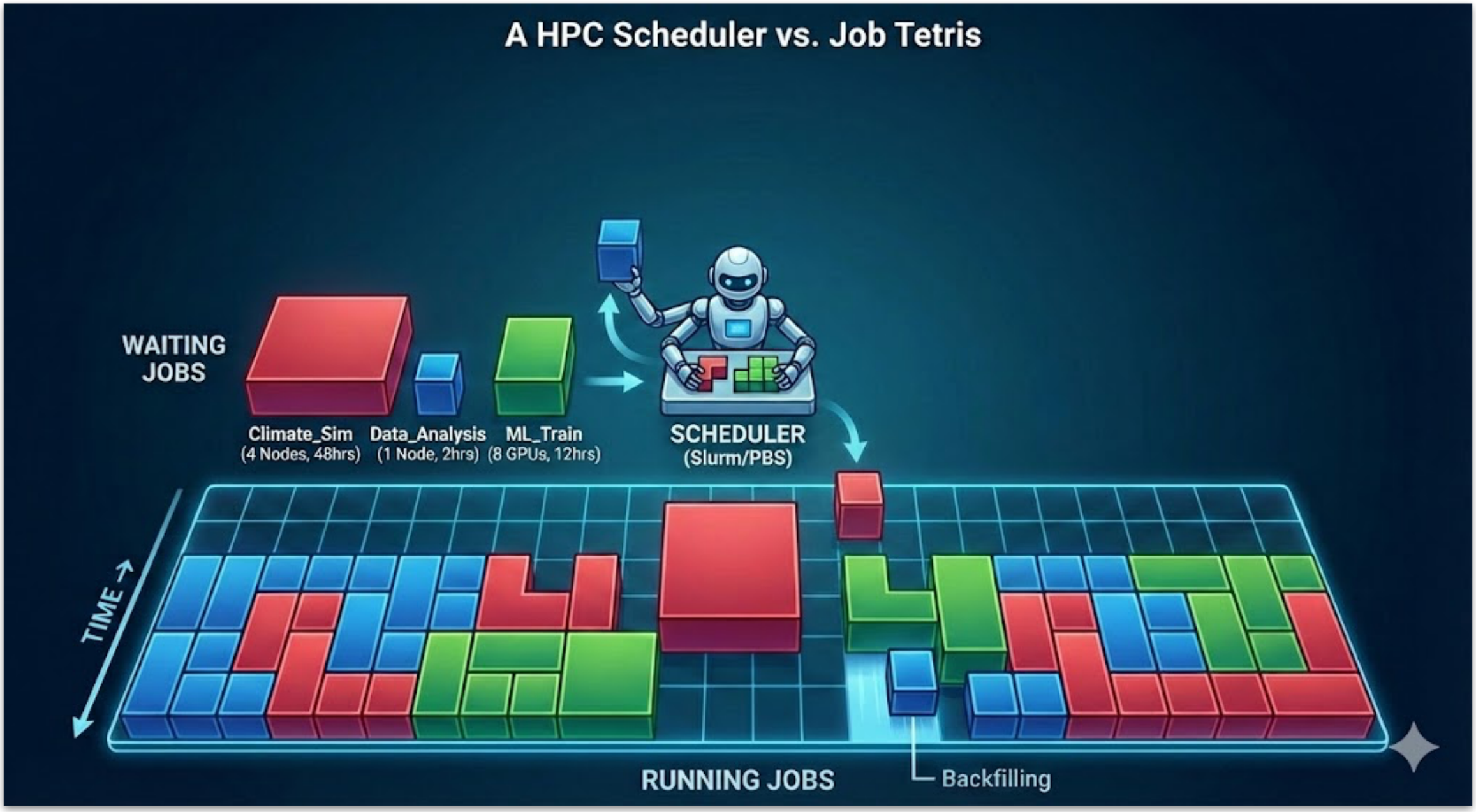
How are they structured?



How are they structured?



The Scheduler



Why do we need a scheduler?

There are 100 researchers on the HPC system and 20 compute nodes. Everyone tries to use multiple compute node.
Result… The entire system crashes.

The scheduler is designed to avoid this from ever happening.

Life on the scheduler: Submit, Queue, Run

1. **Submit**: The user submits a reservation (script) specifying what is needed and how to run it.
2. **Queue**: The scheduler looks at the priority (who has waited the longest) and the request and backfills accordingly.
3. **Execution**: Once the compute nodes are ready, the scheduler logs the user on, runs the code, and then logs back out.

Scheduler: SLURM

- The most common scheduler nowadays
- Slurm won the “scheduler race” because it was free, faster, and smarter than its competition

The SBATCH script

```
#!/bin/bash

# 1. Identity
#SBATCH --job-name=spatialstats_test     # The name that appears in 'squeue'
#SBATCH --output=/hpc/home/nc153/CEE690/Miscellanous/slurm/%x_%j.out     # Save normal output
#SBATCH --error=/hpc/home/nc153/CEE690/Miscellanous/slurm/%x_%j.err      # Save error messages

# 2. Partition (The Queue)
#SBATCH --partition=common         # Which line to stand in (ask your admin for names)

# 3. Resources (The Hardware)
#SBATCH --nodes=1                   # Request 1 physical compute node
#SBATCH --ntasks-per-node=4         # Request 4 CPU cores on that node
#SBATCH --mem=16G                   # Request 16GB of RAM total
#SBATCH --time=01:00:00             # Hard time limit: 1 Hour (HH:MM:SS)

# 4. Notifications (Optional but recommended)
#SBATCH --mail-type=END,FAIL        # Email me when it finishes or crashes
#SBATCH --mail-user=nc153@duke.edu # Where to send the email

# --- PART 2: THE EXECUTION (The Body) ---
# This part runs exactly like a normal shell script, but it runs ON the compute node.

echo "Job started on $(hostname) at $(date)"
# 1. Change directory
cd /hpc/home/nc153/CEE690/Miscellanous/slurm

# 2. Activate Virtual Environment (if using one)
source $HOME/.bashrc
conda activate cee690

# 3. Run the Python Code
echo "Starting simulation..."
spatialstats --JSON_FILE example.json

echo "Job finished at $(date)"
```

Submitting and tracking

Submit (sbatch)

```
(cee690) nc153 $ sbatch spatialstats_submit.sh
Submitted batch job 42974586
```

Check queue (squeue)

```
(cee690) nc153 $ squeue -u nc153
             JOBID PARTITION     NAME     USER ST       TIME
          42974586    common spatials   nc153 PD       0:00
```

Reading logs

```
(cee690) nc153 $ tree -L 1
.
├── example.json
├── output_plot.png
├── output_temporal_stats.nc
├── spatialstats_submit.sh
├── spatialstats_test_42974586.err
└── spatialstats_test_42974586.out
```

```
(cee690) nc153 $ cat spatialstats_test_42974586.out
Job started on dcc-core-07 at Wed Feb  4 11:06:40 PM EST 2026
Starting simulation...
Configuration loaded from example.json
Computing the statistics
Visualizing the data
Plot saved to output_plot.png
Saving statistics to output_temporal_stats.nc
Processing complete.
Job finished at Wed Feb  4 11:06:45 PM EST 2026
```

Debugging Batch jobs is clunky, frustrating, and time consuming

Interactive jobs:

The secret sandbox of HPC

Interactive I: srun —pty

```
(base) nc153 $ hostname
dcc-login-03
(base) nc153 $ srun --pty --nodes=1 --ntasks=4 --mem=8G --time=00:30:00 /bin/bash
nc153 $ hostname
dcc-core-07
nc153 $ squeue -u nc153
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          42976024    common     bash    nc153  R       0:09      1 dcc-core-07
```

```
(cee690) nc153 $ spatialstats --JSON_FILE example.json
Configuration loaded from example.json
Computing the statistics
Visualizing the data
Plot saved to output_plot.png
Saving statistics to output_temporal_stats.nc
Processing complete.
```

Anything that you run will be on the compute node

Interactive II: salloc

```
(base) nc153 $ hostname
dcc-login-03
(base) nc153 $ salloc --nodes=1 --ntasks=4
salloc: Granted job allocation 42976465
salloc: Nodes dcc-core-07 are ready for job
nc153 $ hostname
dcc-login-03
nc153 $ squeue -u nc153
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          42976465    common interact    nc153  R       0:33      1 dcc-core-07
```

```
(cee690) nc153 $ srun -n 1 spatialstats --JSON_FILE example.json
Configuration loaded from example.json
Computing the statistics
Visualizing the data
Plot saved to output_plot.png
Saving statistics to output_temporal_stats.nc
Processing complete.
```

You will remain on the login node and need to use “srun” to launch your process on the allocated compute node(s)

Finally: Please, please remember