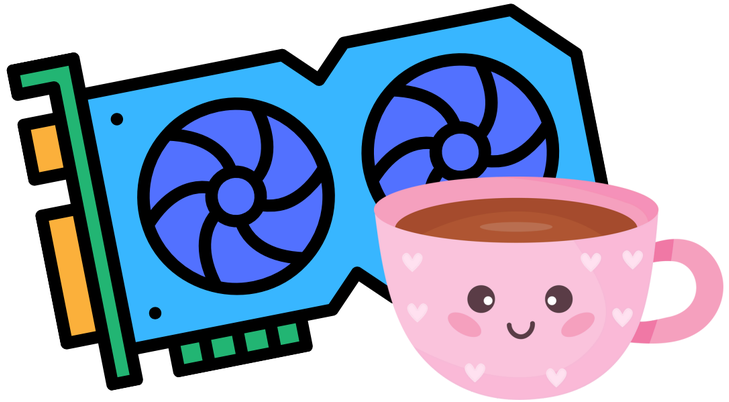
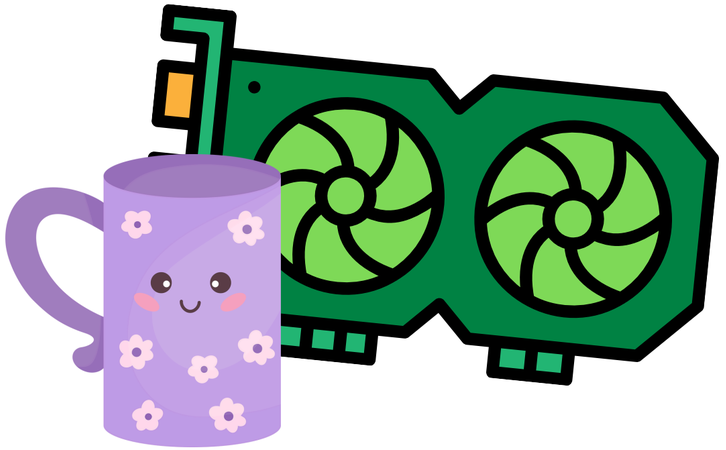
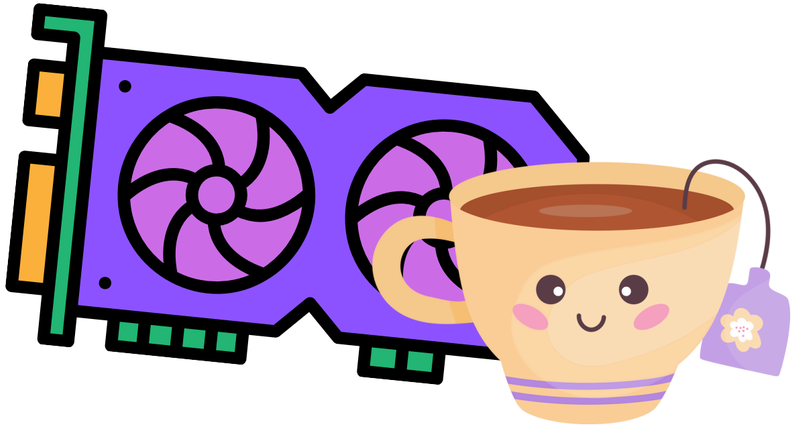
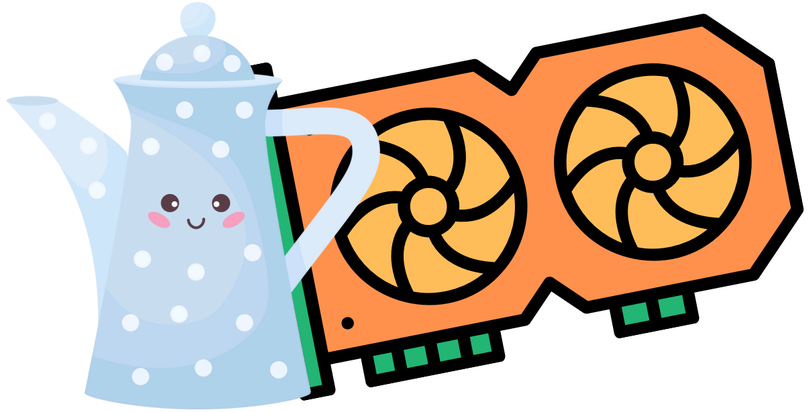




GPU 101





CAVEATS:

- I am not an expert
- This is a pretty basic explanation
- This will not be comprehensive - just a starter to encourage you to go and explore!
- Offload heavy explanation vs GPU-only approach

WHAT IS A GPU?

Graphic Processing Unit

Multi-cored accelerator used for parallel programming

Can perform many calculations simultaneously across different cores

- Better FLOPs per second (plus cooling, size, energy usage arguments)
- They can also allow for memory transfers and calculations to happen simultaneously
- Transferring work to GPU frees up the CPU to do other tasks

GETTING STARTED WITH GPUS

What kind of GPU do you have access to?

What do you want to do on your GPU?

What limitations does your code/device have?

GETTING STARTED WITH GPUS

What kind of GPU do you have access to?

- Nvidia vs AMD vs Intel

What do you want to do on your GPU?

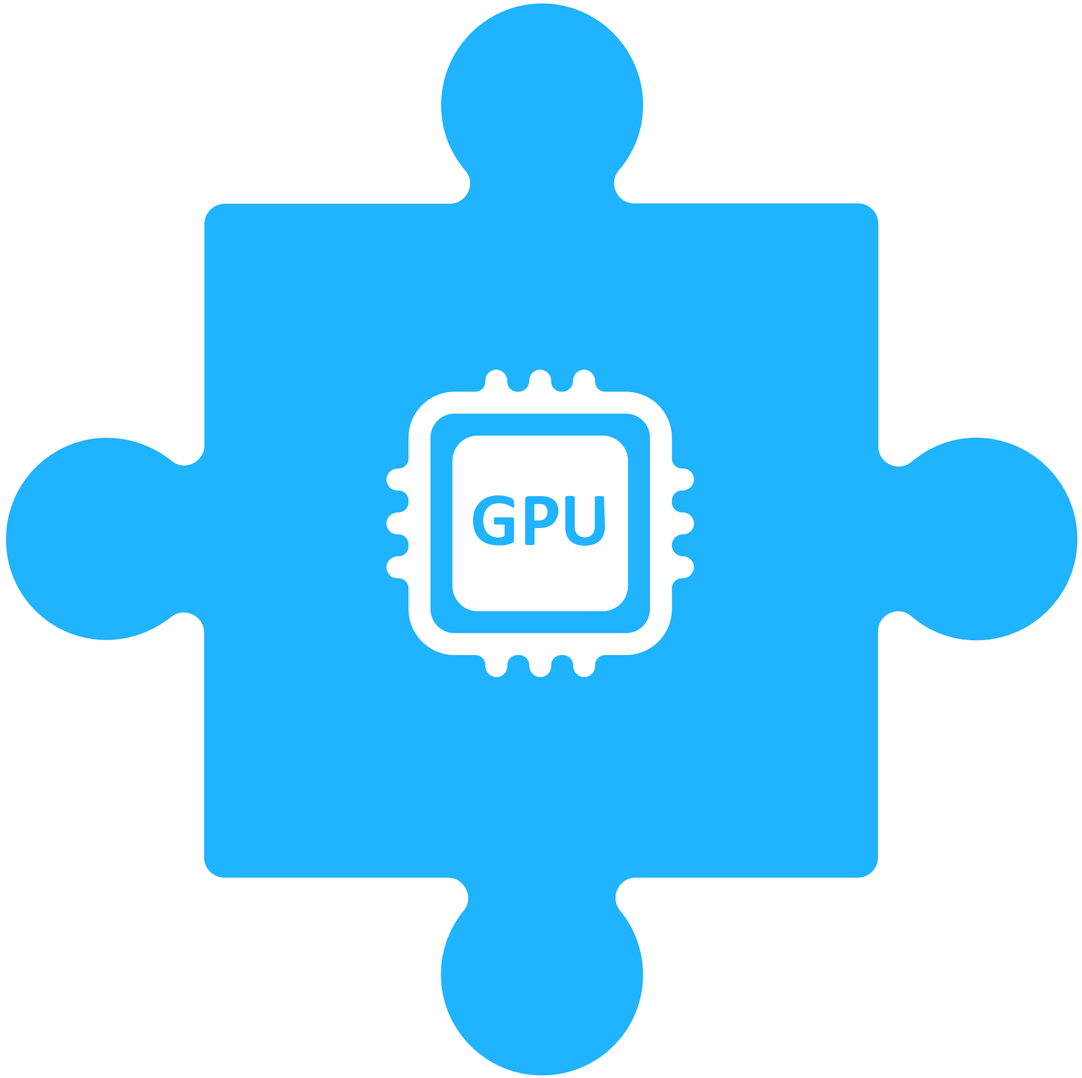
- GPU only vs offloads

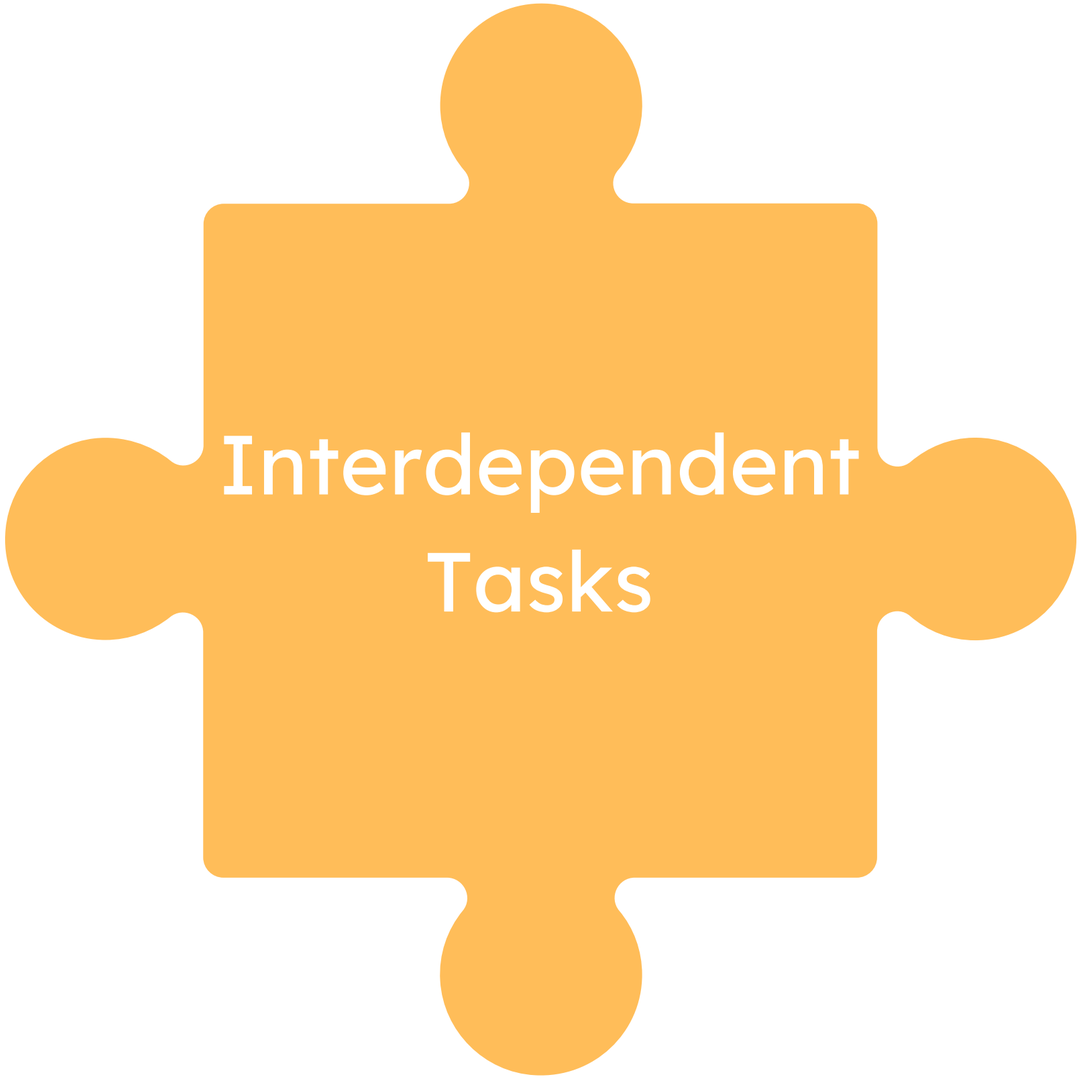
What limitations does your code/device have?

- memory and memory transfer
- is there a good area to target in your code?

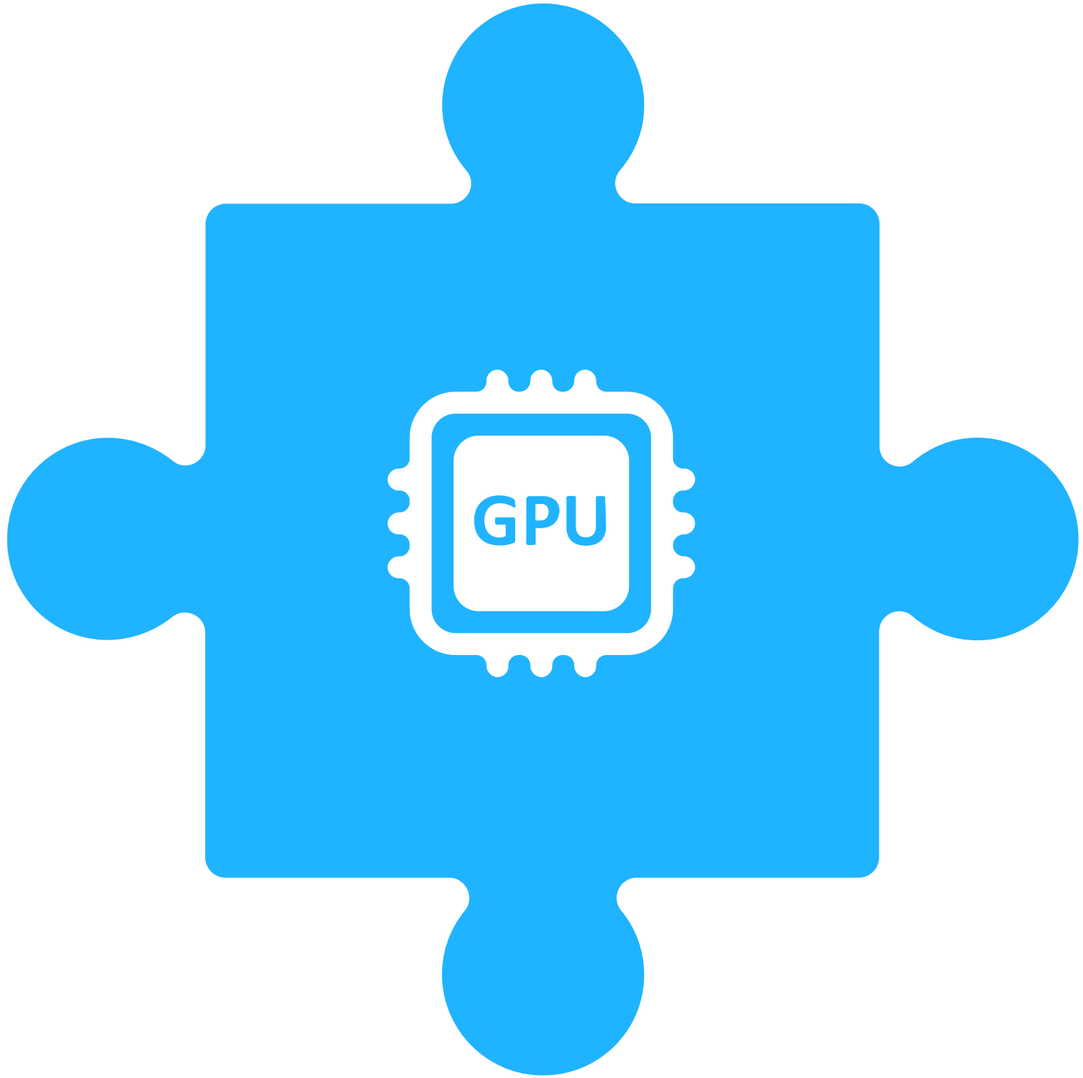
Some processes are not suitable for GPUs

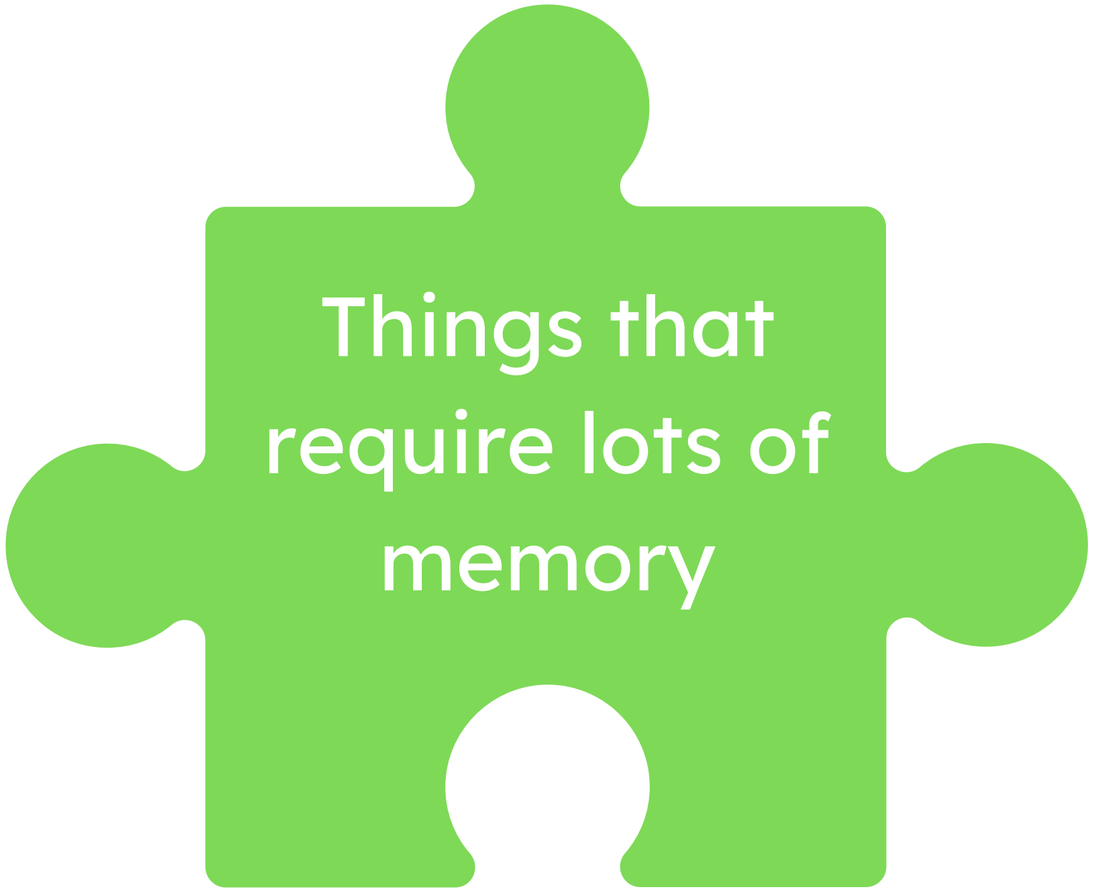
Some things can’t work



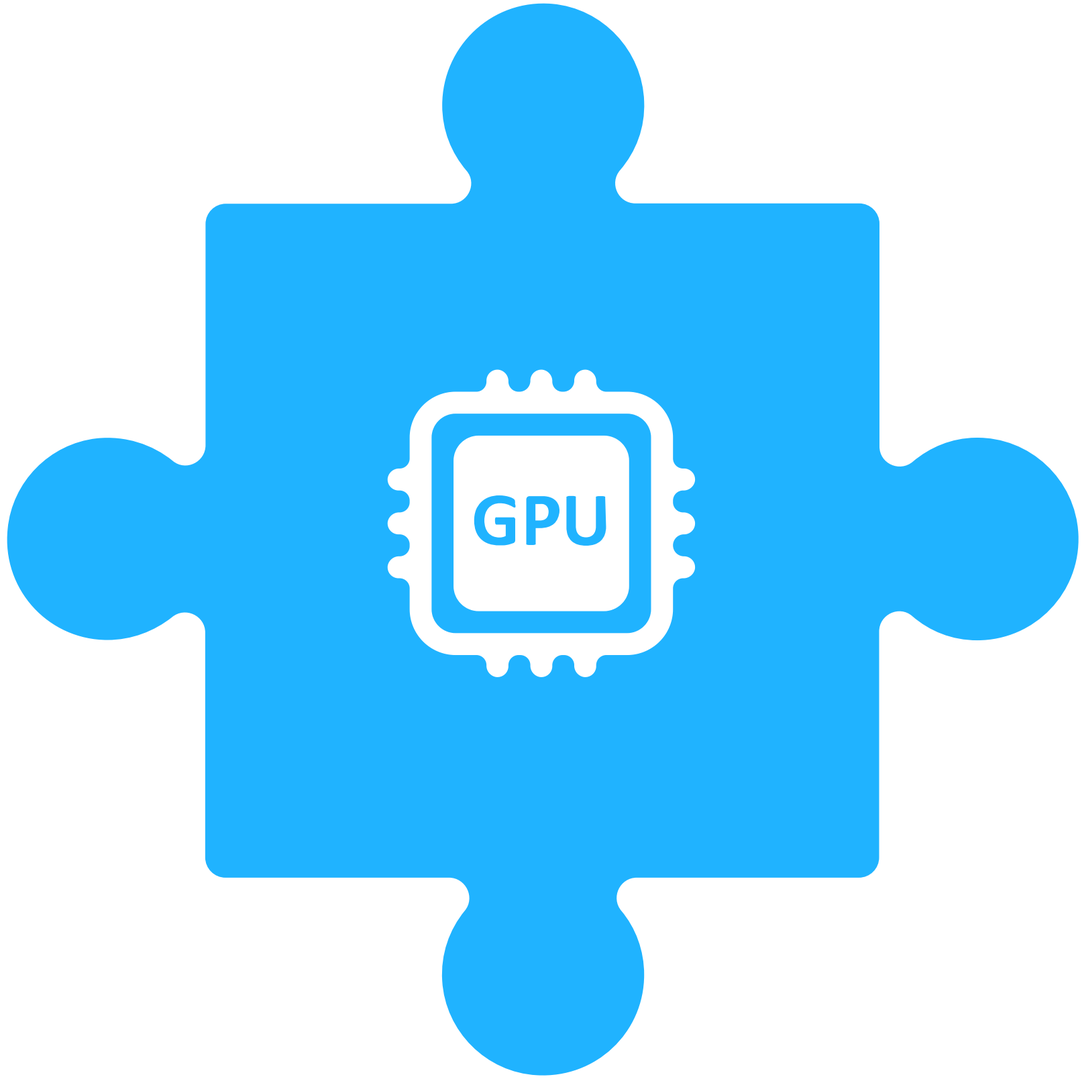


Some things aren’t ideal





Some things fit perfectly

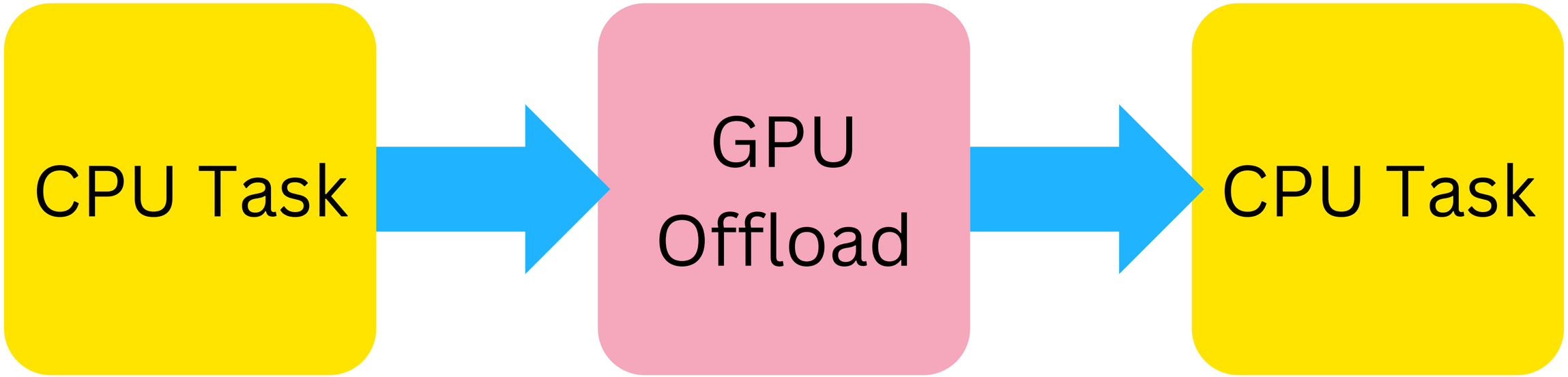


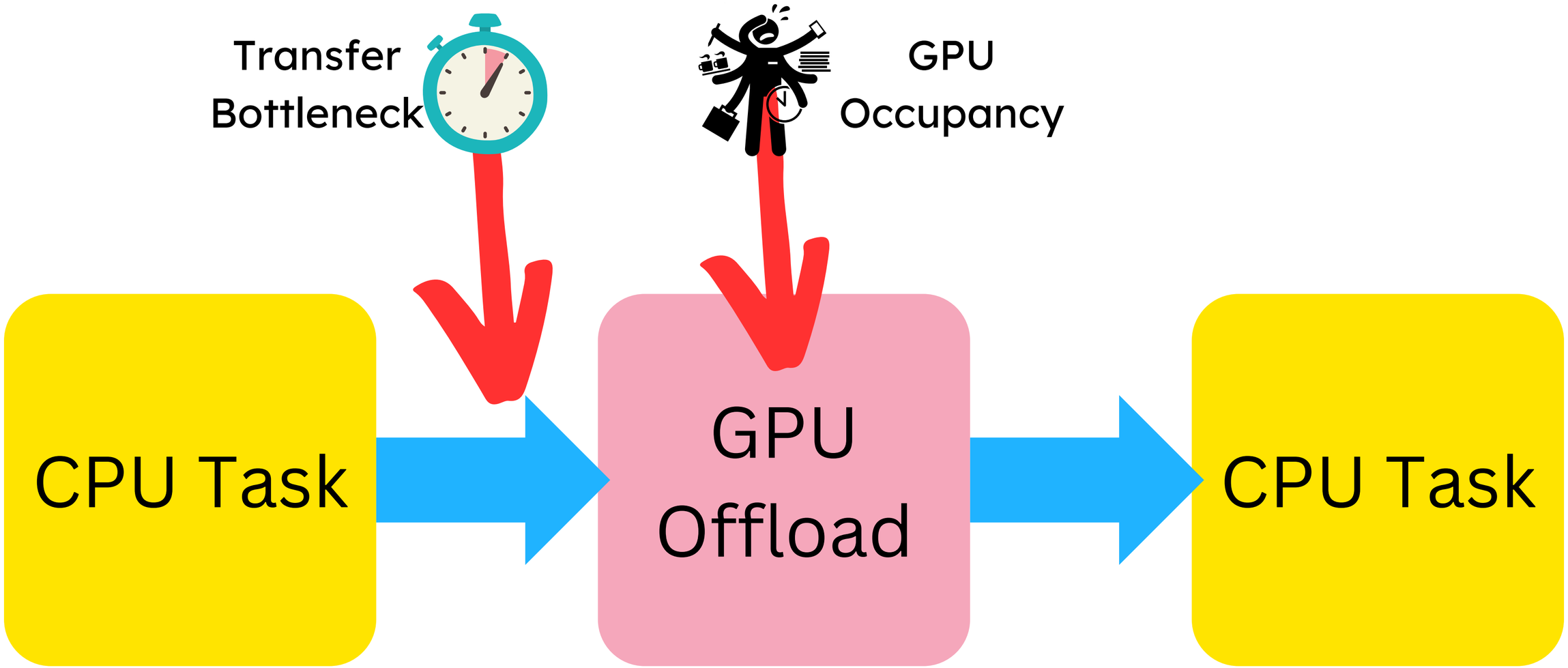
TRANSFORMS - every element of an array has the same operation applied

REDUCTIONS - returns sum of every element in vector

GATHERS - retrieves elements by index and maps them to a new array

INTERACTIONS - sums interactions between elements i and j


CPU Task
GPU
Offload
CPU Task


Transfer
Bottleneck
GPU
Occupancy
CPU Task
GPU
Offload
CPU Task

PROGRAMMING LANGUAGES

Vendor specific:

- CUDA (Nvidia)
- HIP (AMD - can turn CUDA into HIP/use CUDA on AMD devices now!)

Portable:

- SYCL/Kokkos/Raja

GPU parallisation options:

- OpenMP

PROGRAMMING LANGUAGES

Vendor specific:

- CUDA (Nvidia)
- HIP (AMD - can turn CUDA into HIP/use CUDA on AMD devices now!)

Portable:

- SYCL/Kokkos/Raja

GPU parallisation options:

- OpenMP

GPUS ON COSMA

SSH to node with GPU
(or submit job to queue at end)

module load nvhpc
(for CUDA)

We can find out about our device properties (example)

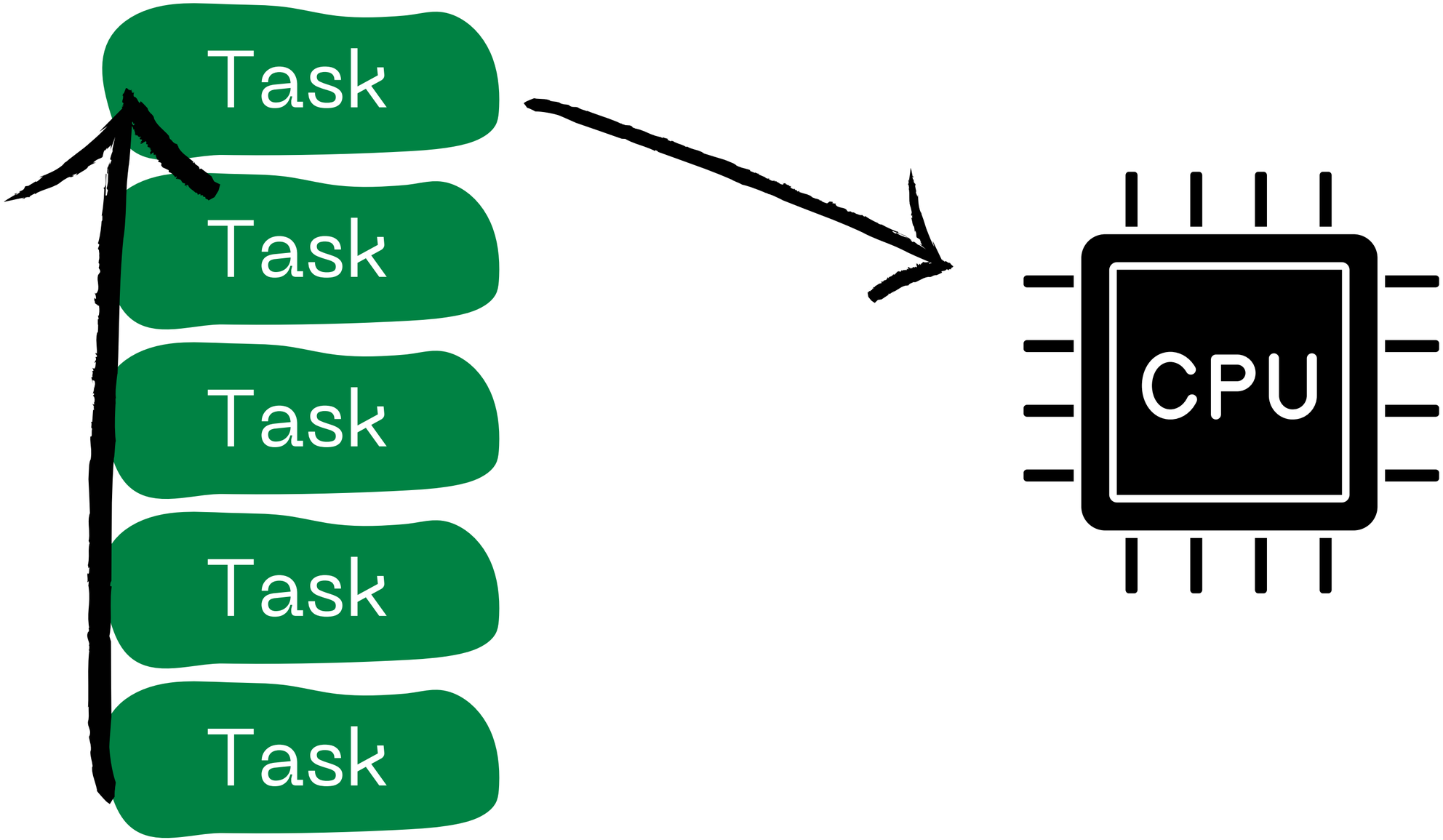
GPUs

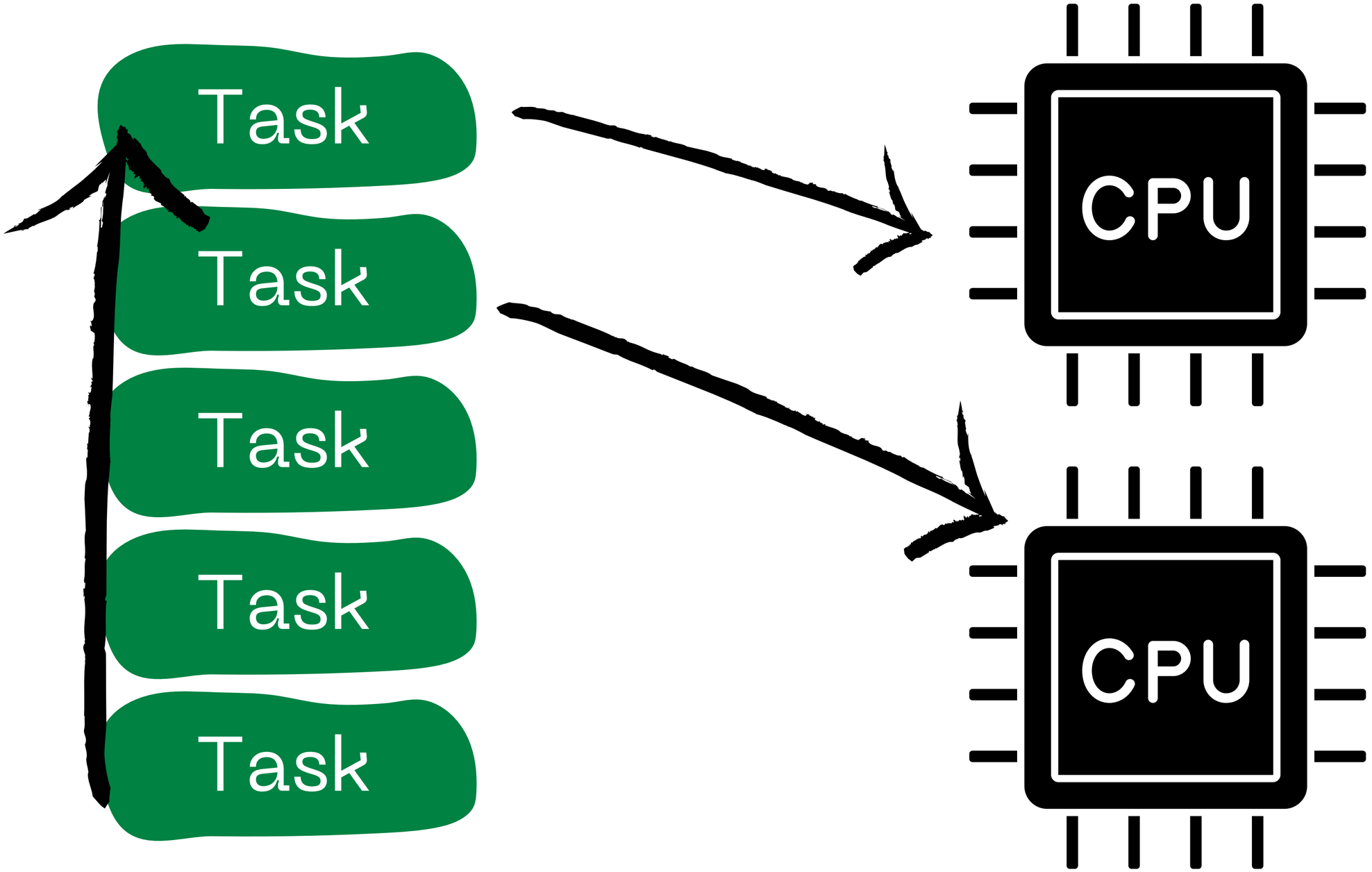
COSMA has a number of GPU systems, which are available for use. These are:

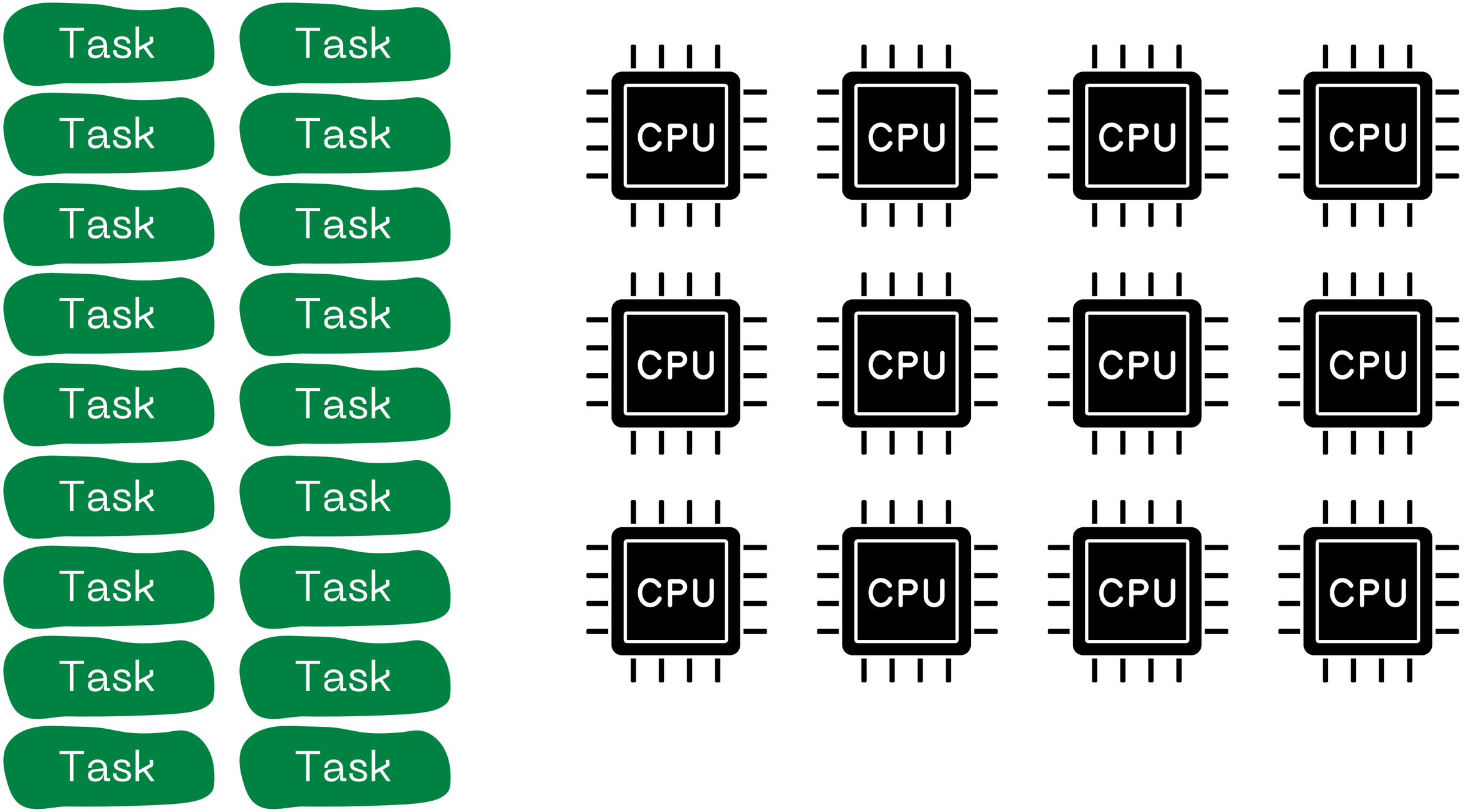
- Direct login (from a login node)
  - gn001: 10x NVIDIA V100 GPUs
  - gn002: NVIDIA Grace-Hopper (ARM) system
  - gi001: 2x Intel Ponte Vecchio GPUs
  - mad06: 0-3x NVIDIA A100 GPUs (1TB RAM)
- cosma8-shm Slurm partition
  - mad04: 0-3x NVIDIA A100 GPUs (4TB RAM)
  - mad05: 0-3x NVIDIA A100 GPUs (4TB RAM)
- cosma8-shm2 Slurm partition
  - ga004: 1x AMD MI100 GPU
  - ga005: 2x AMD MI200 GPUs
  - ga006: 2x AMD MI200 GPUs
- dine2 Slurm partition
  - gc[001-008]: 0-8x NVIDIA A30 GPUs
- gracehopper Slurm partition
  - gn003: NVIDIA Grace-Hopper (ARM) system
- mi300x, mi300x-prince partition: AMD MI300X system
  - ga007: 8x AMD MI300 GPUs

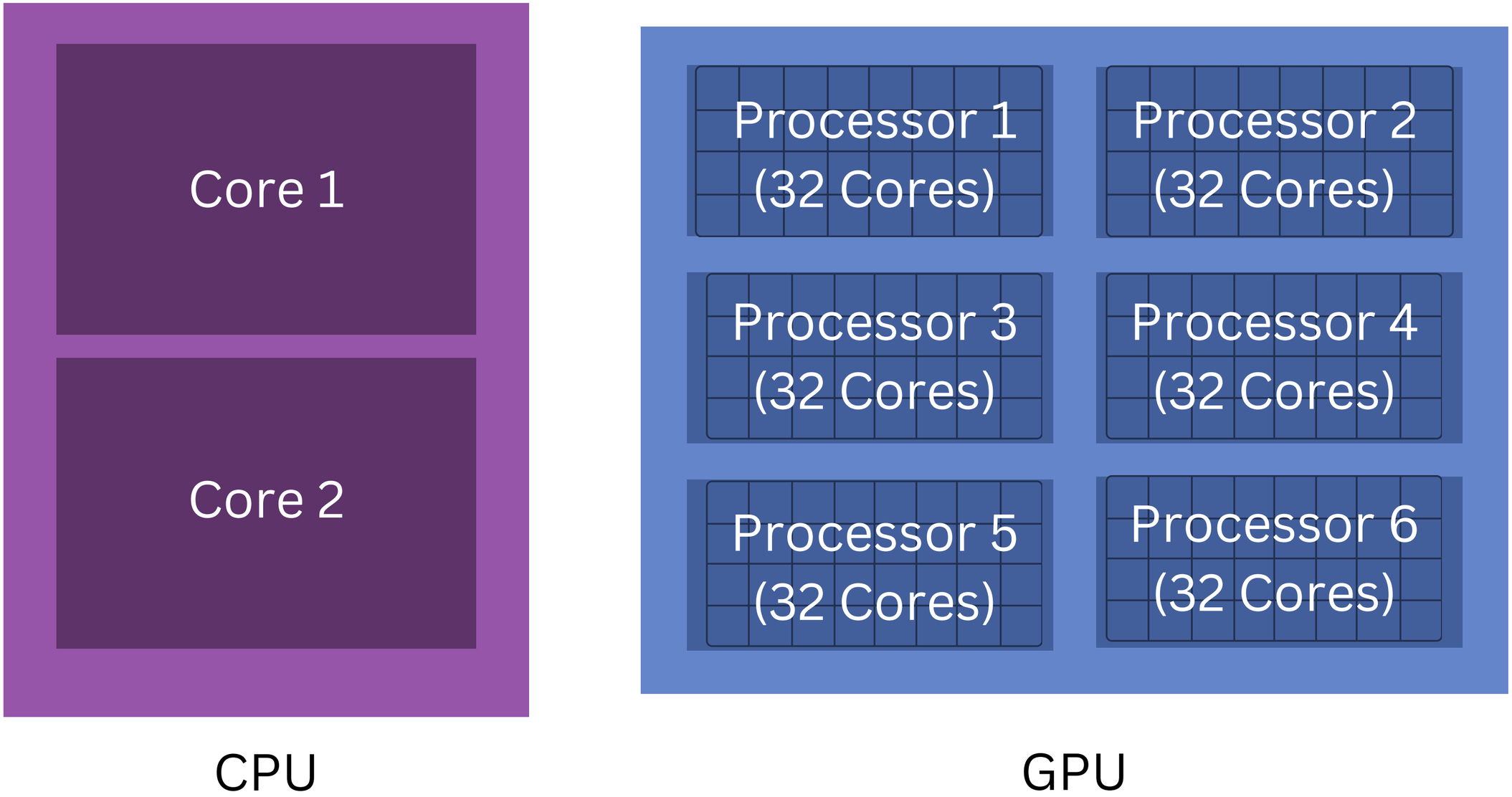
PARALLELISM AND WORKLOAD MANAGEMENT

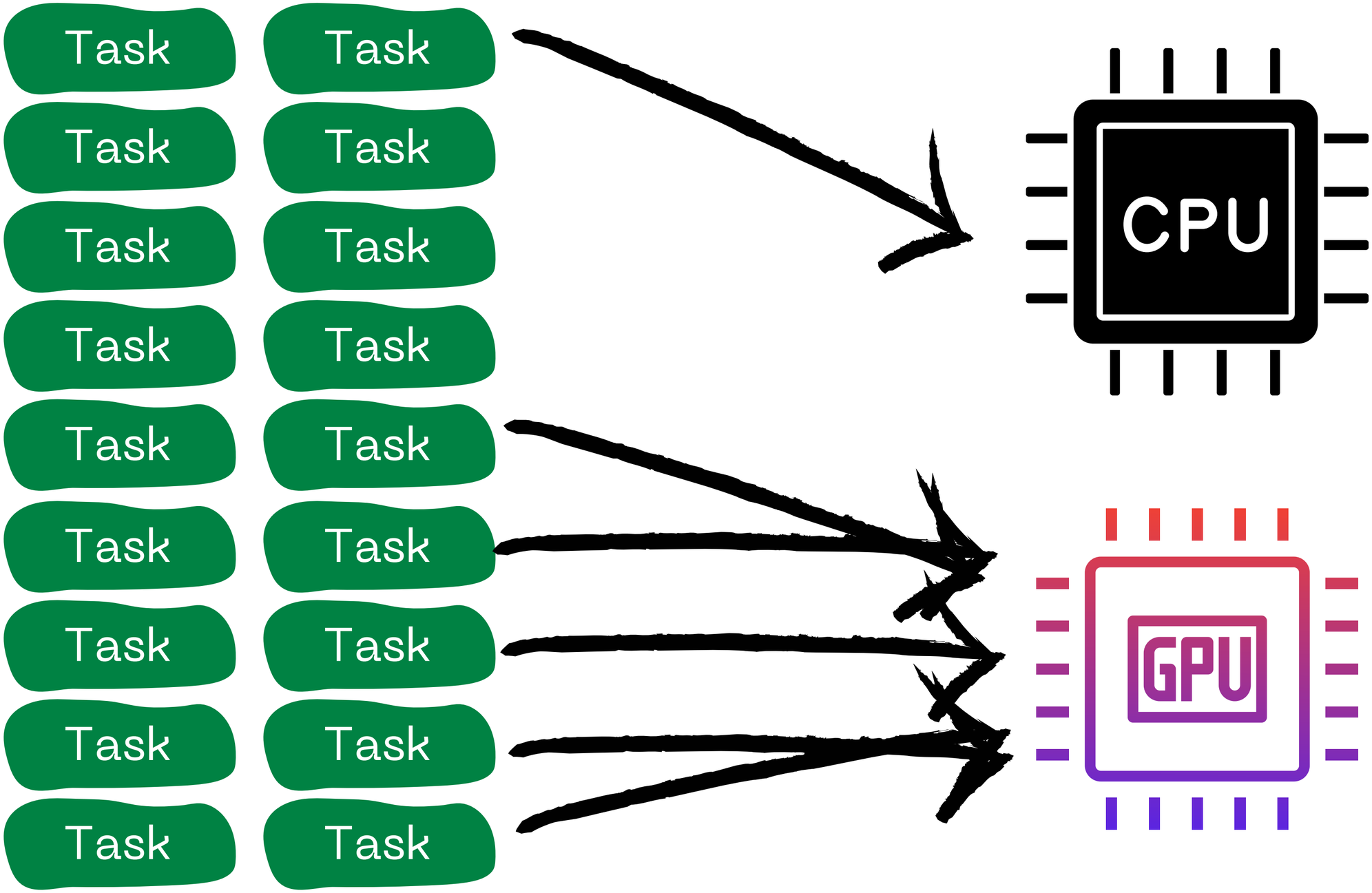

Task
CPU

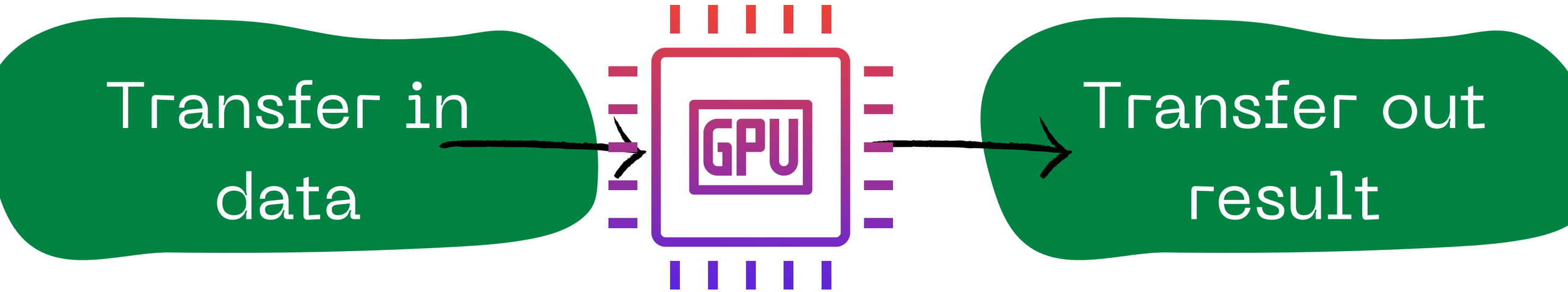

Task
Task
Task
Task
Task
CPU

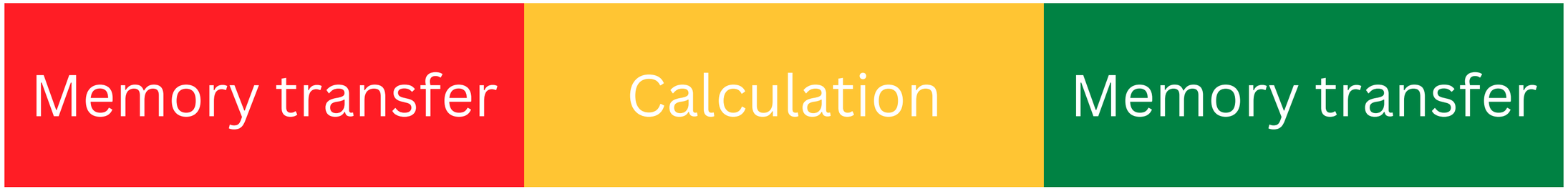

Task
Task
Task
Task
Task
CPU
CPU

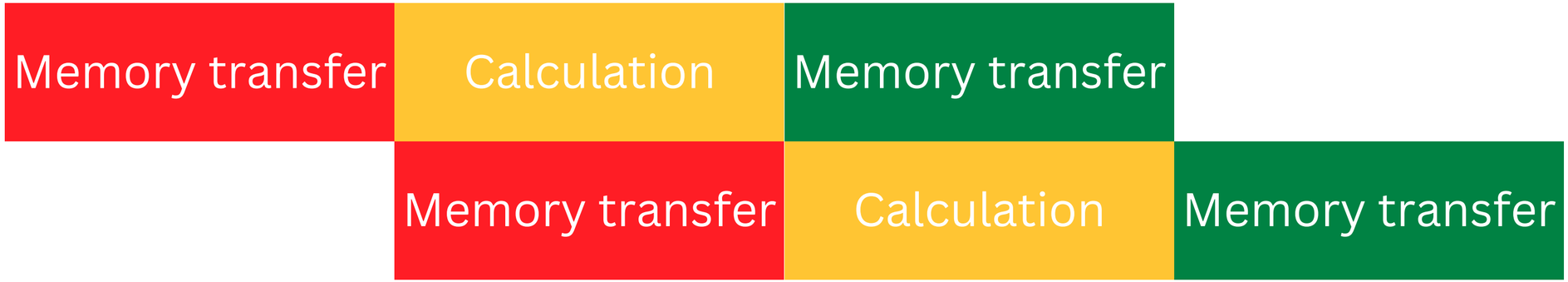

Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU

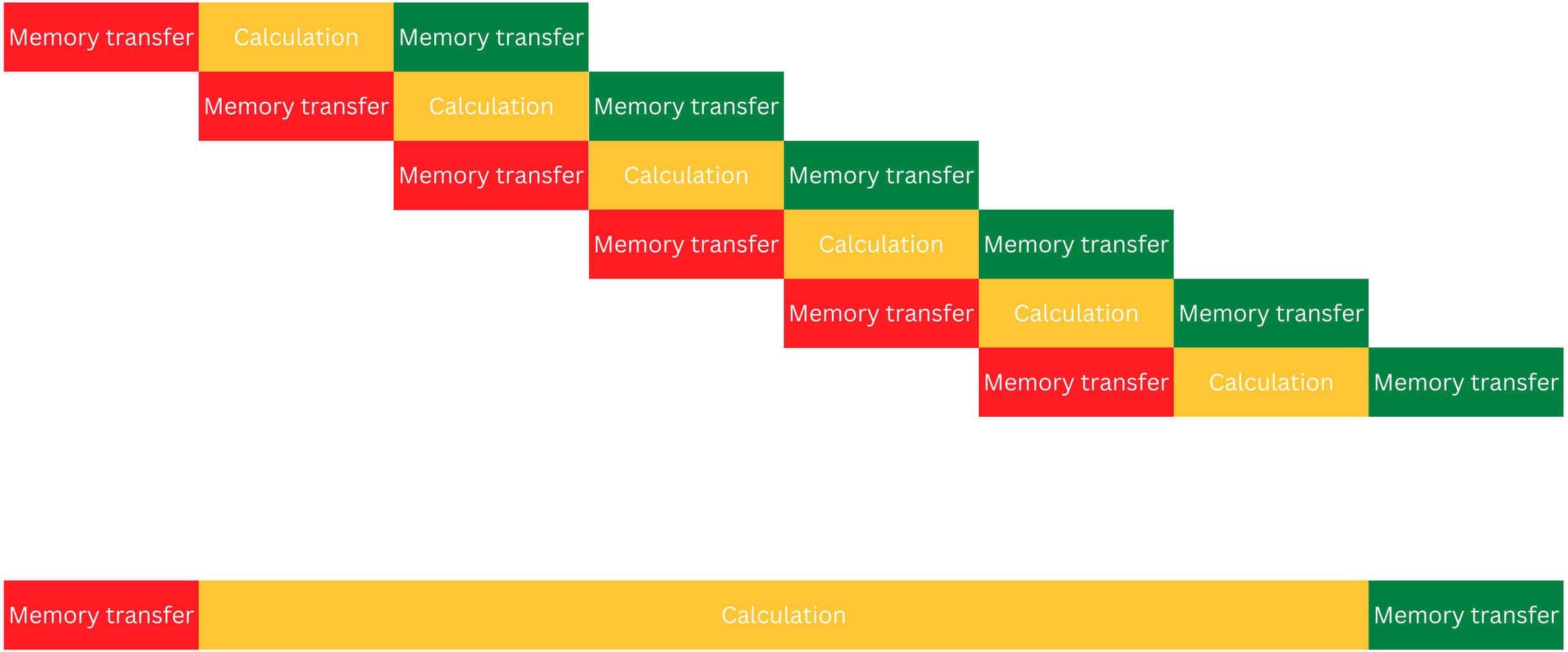

Core 1
Core 2
CPU
Processor 1
(32 Cores)
Processor 2
(32 Cores)
Processor 3
(32 Cores)
Processor 4
(32 Cores)
Processor 5
(32 Cores)
Processor 6
(32 Cores)
GPU


Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
Task
CPU
GPU


Transfer in data
GPU
Transfer out result


Memory transfer
Calculation
Memory transfer


Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer


Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer

SETTING UP CUDA CODE

We need to:

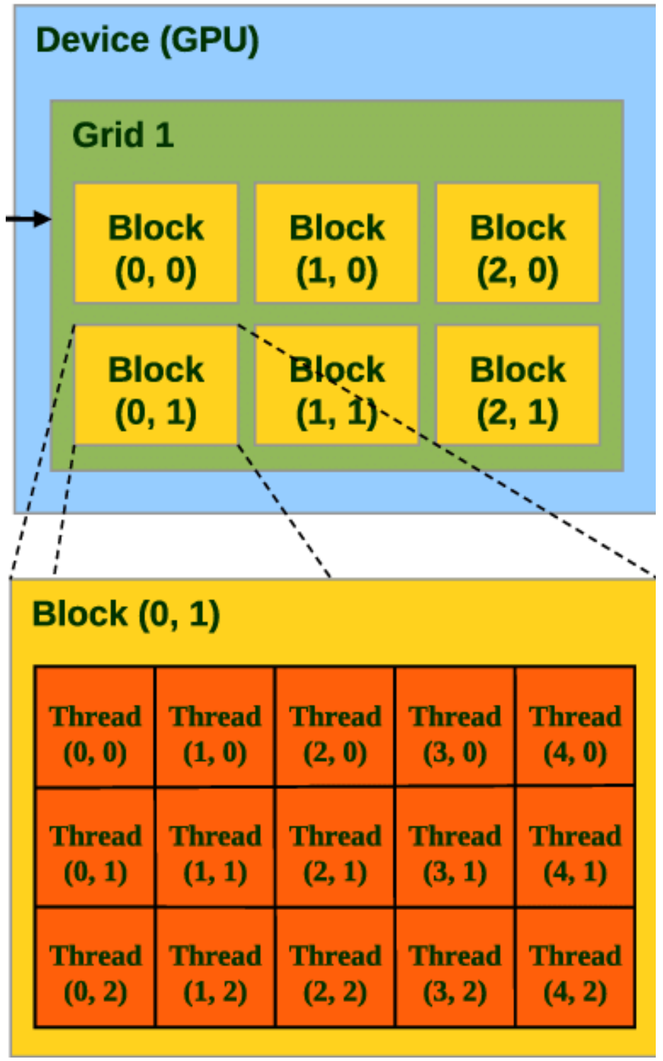
- allocate memory on the GPU
- fill memory on the GPU
- write a device kernel (bit that does the calculation)
- copy results back from GPU

SETTING UP CUDA CODE

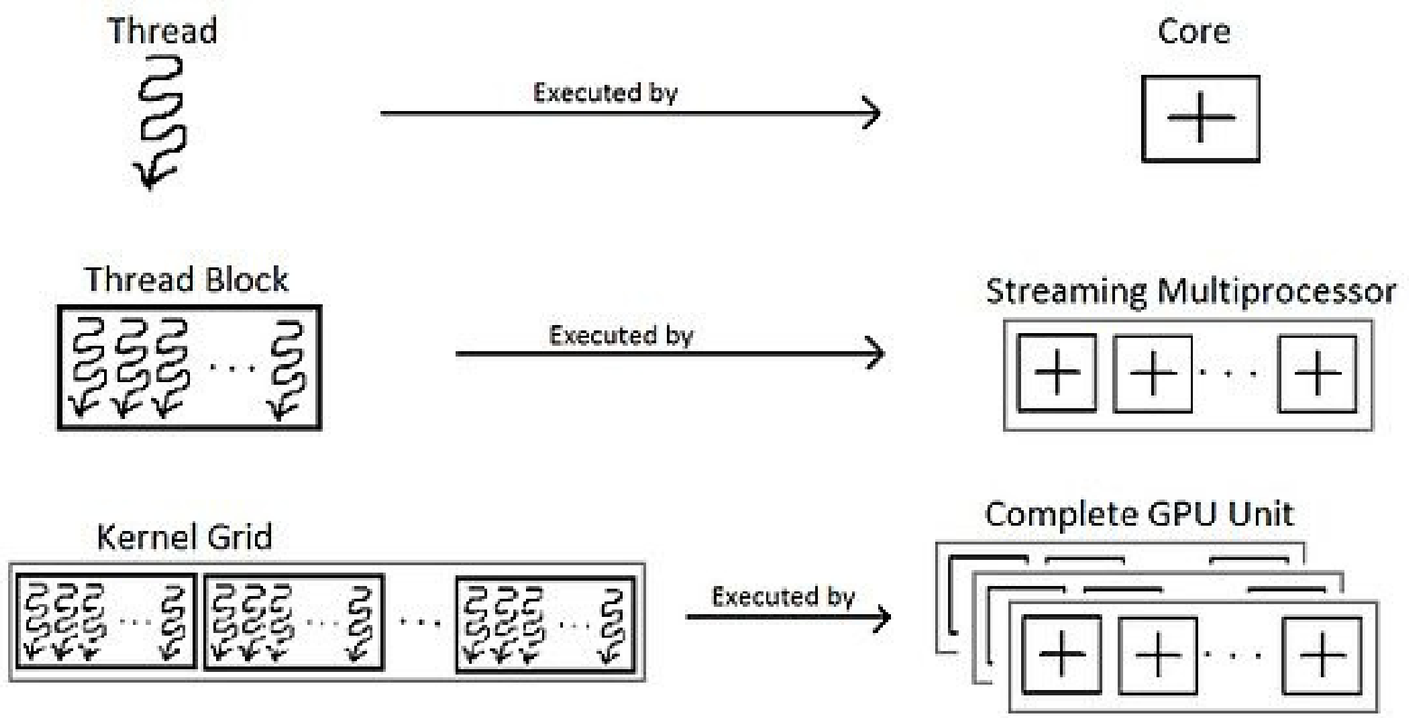
We need to:

- allocate memory on the GPU
  - cudaMalloc
- fill memory on the GPU
  - cudaMemcpy (device_array, host_array, cudaMemcpyHostToDevice)
- write a device kernel (bit that does the calculation)
  - kernel << blocks, threads >>
- copy results back from GPU
  - cudaMemcpy (host_array, device_array, cudaMemcpyDeviceToHost)

ASIDE: THREADS AND BLOCKS



Scalabrin et al. 2014

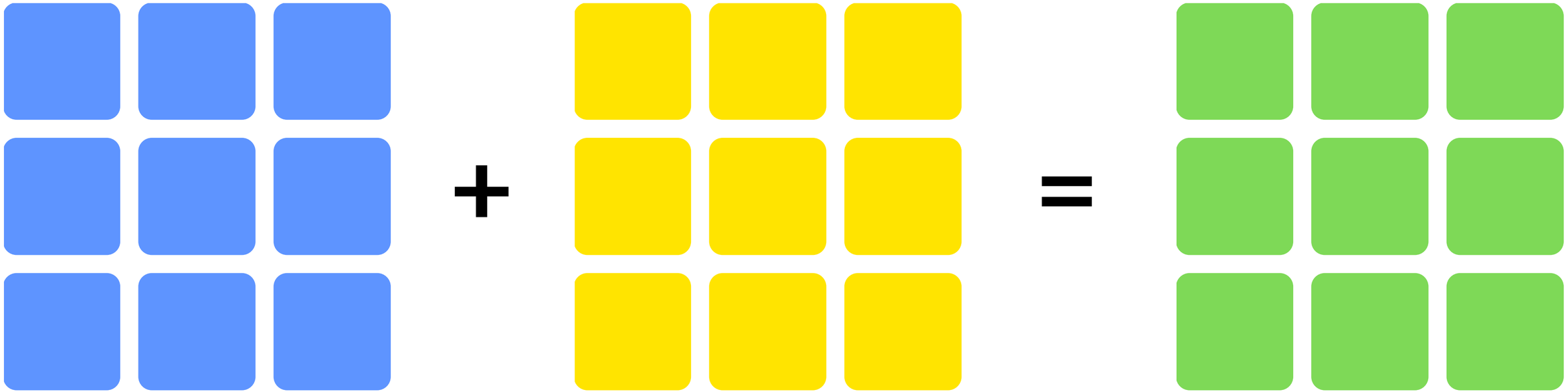


Hwu Wen-Mei 2009

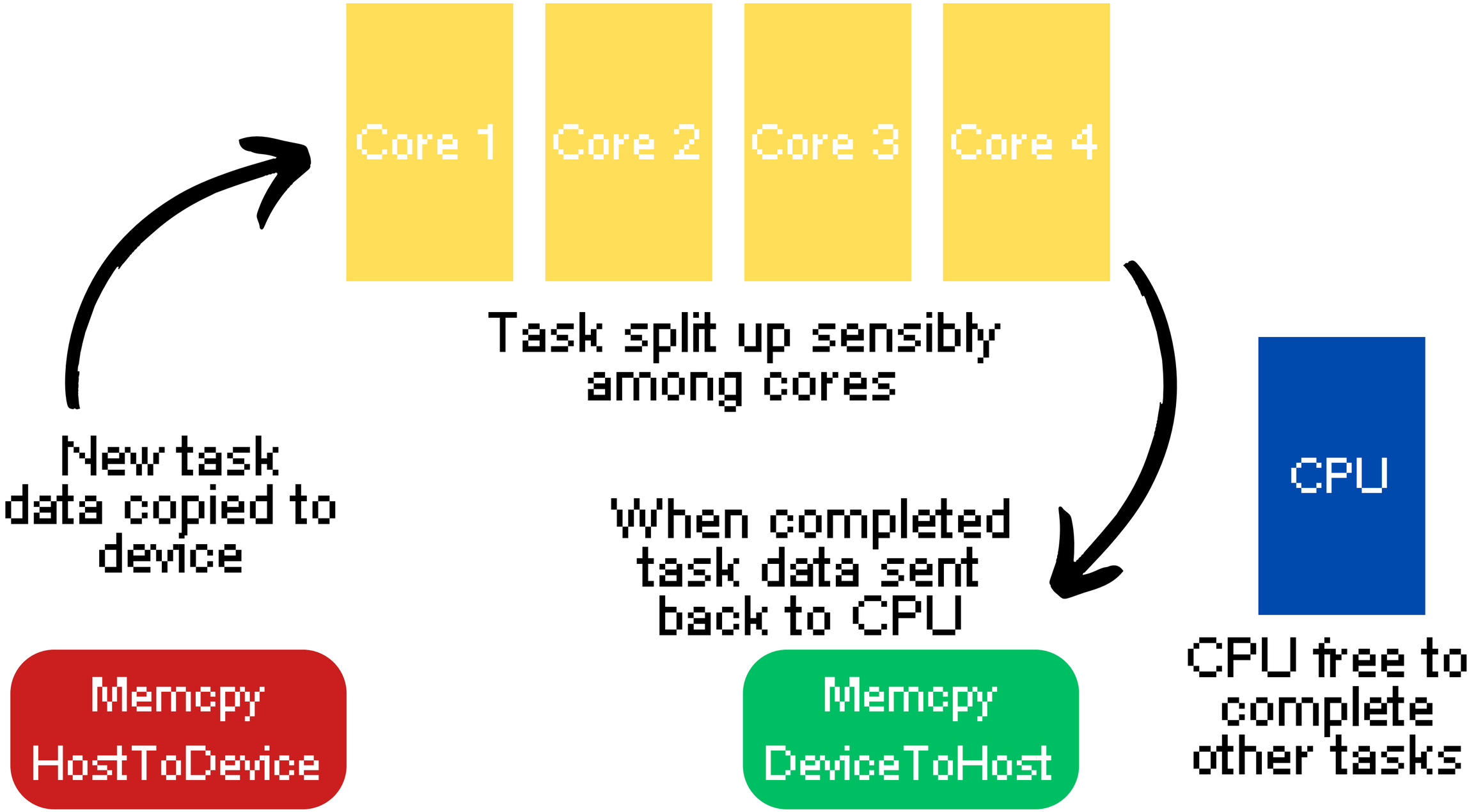
LET'S DO A 'HELLO WORLD'

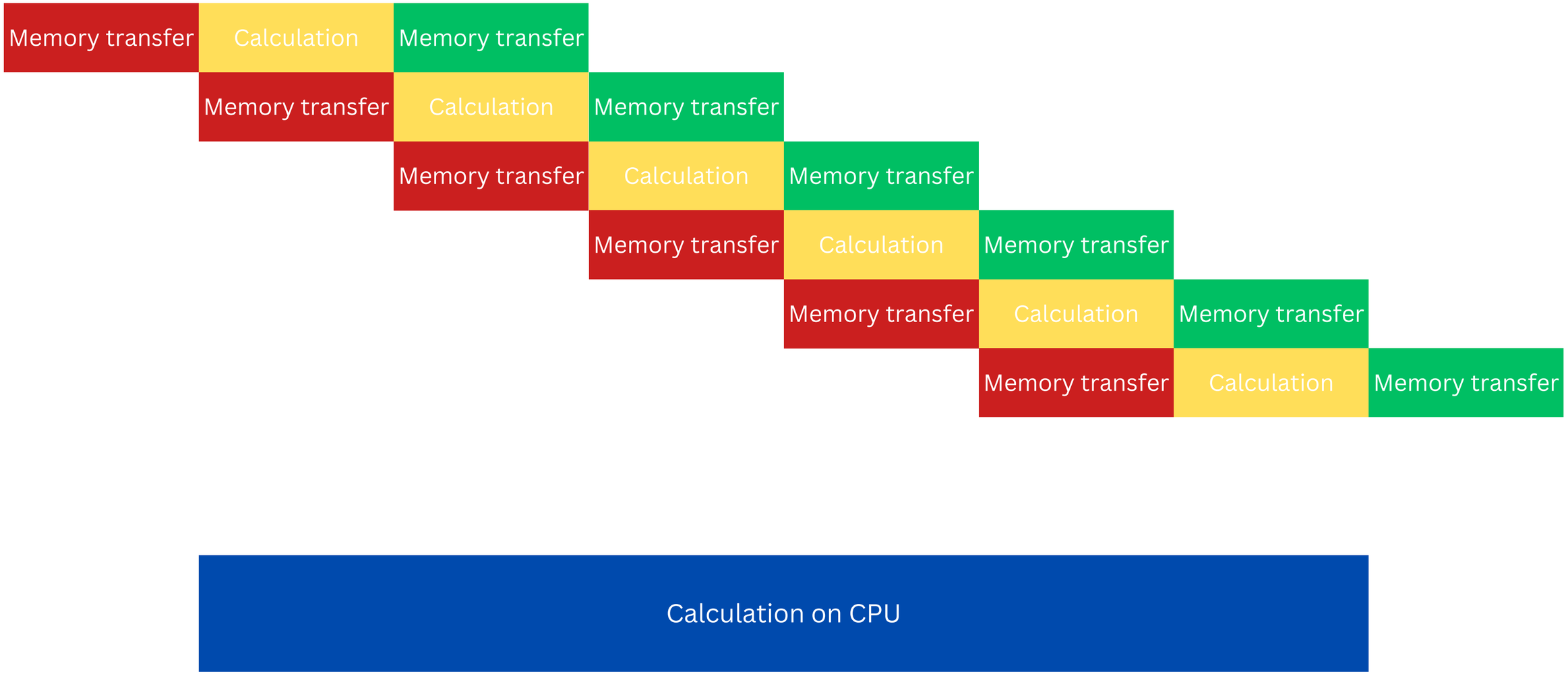
CPU: Hello World

GPU: Vector addition



GPU OFFLOADS AND CPU USAGE

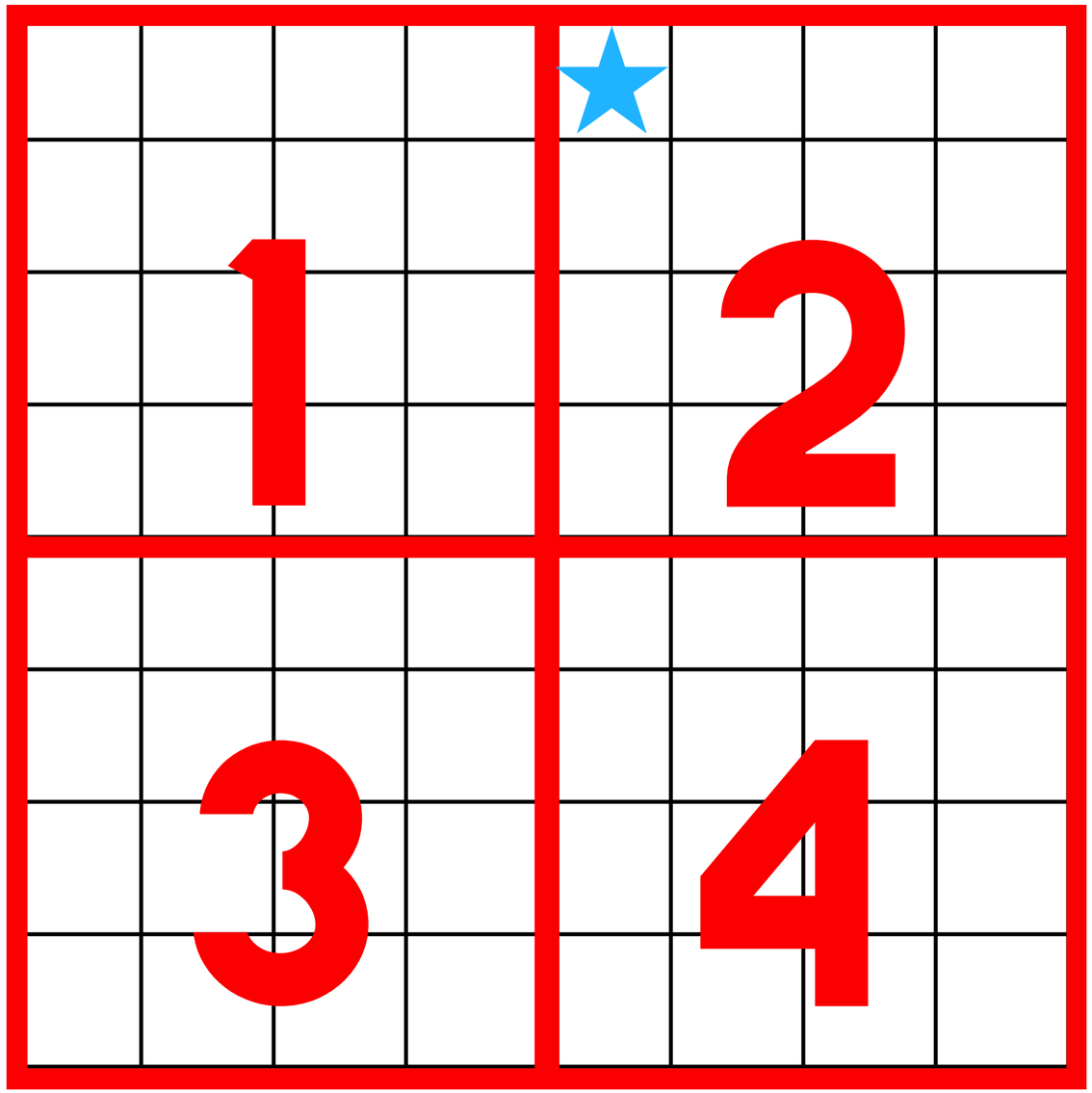

Core 1
Core 2
Core 3
Core 4
Task split up sensibly among cores
New task data copied to device
When completed task data sent back to CPU
CPU
CPU free to complete other tasks
Memcpy HostToDevice
Memcpy DeviceToHost


Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Memory transfer
Calculation
Memory transfer
Calculation on CPU

OFFLOAD EXAMPLE

N-body GPU example

GRID-STEPPING



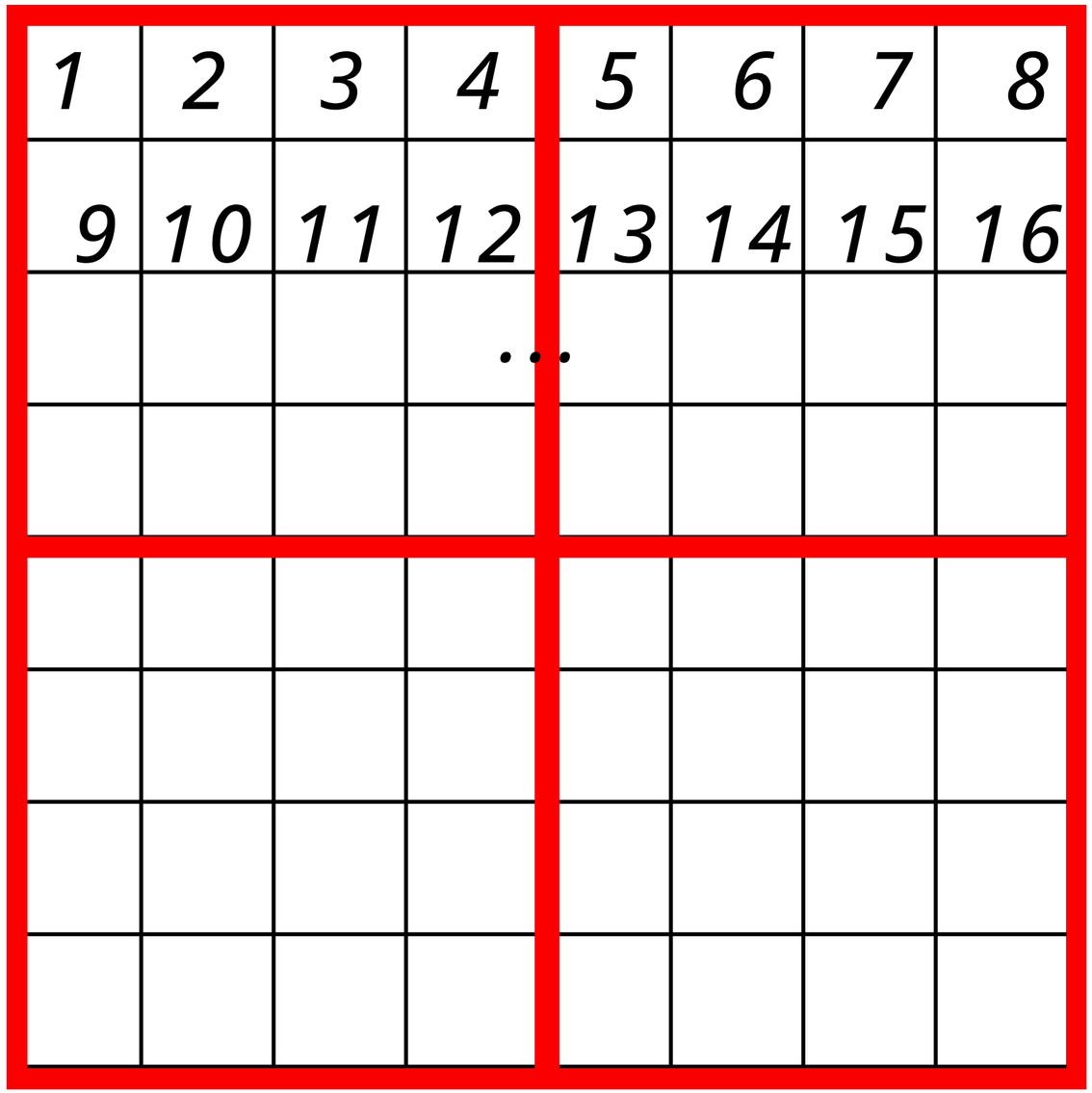
for (int i = t; i < n; i+=T)

t: starting position in your loop
T: jumps in loop

int t = blockIdx.x*blockDim.x +threadIdx.x;
int T = blockDim.x*gridDim.x;

int t = 2*4 + 1;
int T = 4*1;

GRID-STEPPING



for (int i = t; i < n; i+=T)

t: starting position in your loop
T: jumps in loop

int t = blockIdx.x*blockDim.x +threadIdx.x;
int T = blockDim.x*gridDim.x;

ERRORS ON GPUS

Segmentation faults - memory problems

- use CUDA error checkers to identify specific problems
- synchronise the device (cudaDeviceSynchronize)
- check mallocs and memcpys

Incorrect results:

- rounding
- check naming and copies of arrays

OPTIMISATION

Where would I benefit from a kernel?

- run a profiler on your code
- Valgrind/dmalloc for memory checks

When is my kernel being called?

- run a profiler on your code that has features for GPUs (nvprof/rocprof)

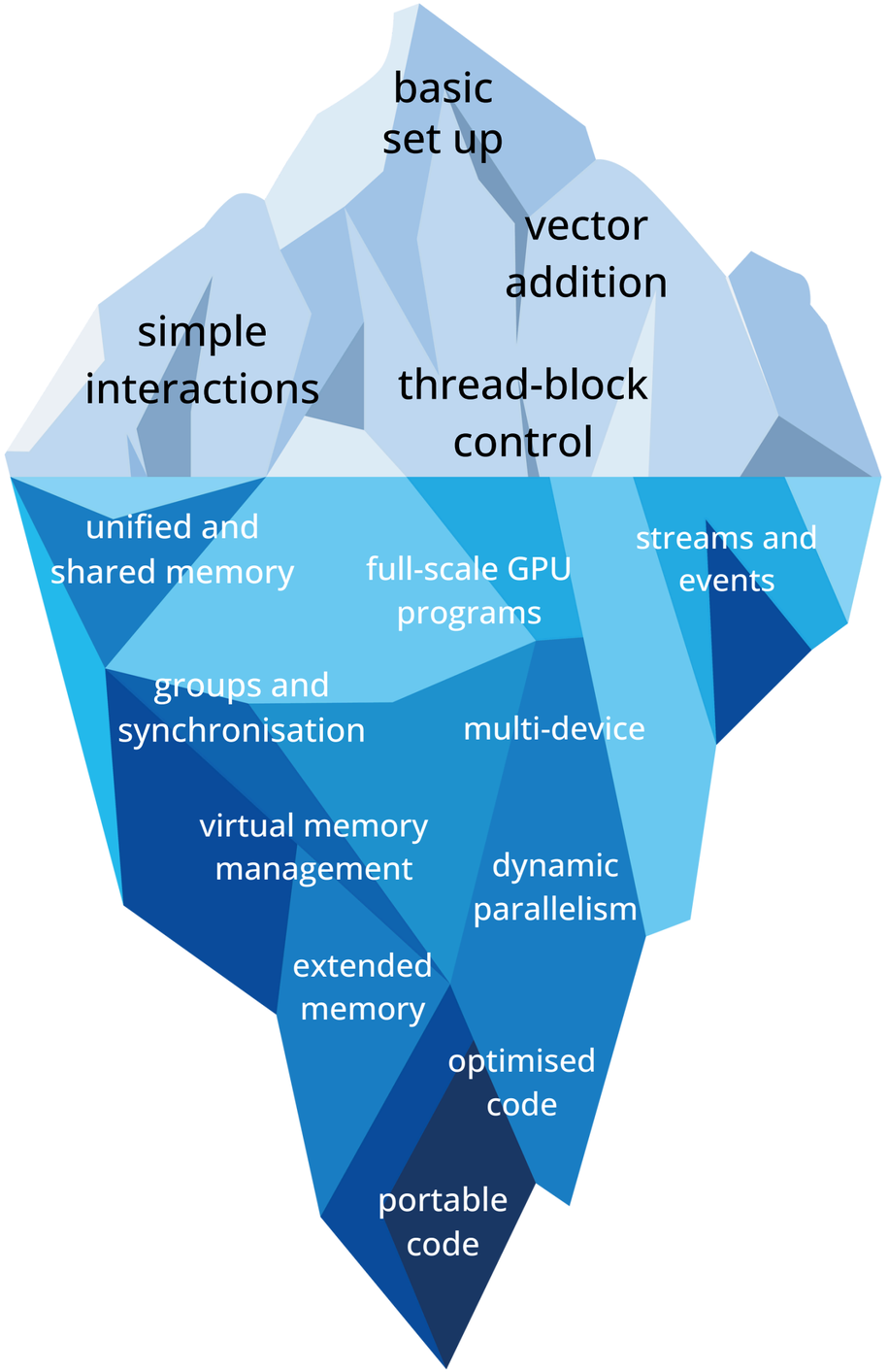
How is my kernel behaving?

- run a kernel profiler (nsight/omniperf)

I have a bug :(

- have a crisis
- use a debugger (compute sanitiser, rocgdb)
- in-code checks and print statements


CUDA HW (0000:97:00.0 - NVIDIA
Kernel
Memory
[All Streams]
25.1% Stream 14
25.0% Stream 15
25.0% Stream 16
24.9% Stream 13
pair_grav_...
Threads (133)
0 to 100%
[3622797] swift
OS runtime libraries
CUDA API
Profiler overhead
ioctl
pthr...
cudaGet...
cud...
cudaMem...
cudaGetDeviceProperties_v2
[3622794] swift
pthread_mutex_lock
cudaMemcpyAsync
[3622795] swift
pt...
pth...
cu...
cudaMem...
[3622792] swift
pthread_mut...
pthread_...
cudaMemcp...
cudaMem...
cudaMe...


basic
set up
vector
addition
simple
interactions
thread-block
control
unified and
shared memory
full-scale GPU
programs
streams and
events
groups and
synchronisation
multi-device
virtual memory
management
dynamic
parallelism
extended
memory
optimised
code
portable
code

FURTHER RESOURCES

TRAINING:

- DiRAC training (1-3 times a year)
- Nvidia CUDA Programming Course (online, recorded)
- Archer2/CompSci Courses
- Udemy/CodeAcademy

GUIDES:

- CUDA C/C++ Programming Guides
- “Parallel and High Performance Computing” textbook

QUESTIONS AND ISSUES:

- google is your friend
- Nvidia LLM/AI programming assistants (use with caution)

GO AND HAVE A PLAY AROUND WITH GPU THINGS!