GPU Computing at ACCRE

CPU vs GPU Computing Overview

Toy Model of Single Core CPU

ACCRE
Advanced Computing Center for Research & Education

Instructions

```
c = b + 10
if (c < 0):
    a = b * a
else:
    a = a + 4
```

Data

a	3
b	-8
c	0

CPU Core 1

Clock

T = 0

Toy Model of Single Core CPU

Instructions

```
c = b + 10
if (c < 0):
    a = b * a
else:
    a = a + 4
```

Data

a	3
b	-8
c	2

CPU Core 1

Clock

T = 1

Toy Model of Single Core CPU

Instructions

```
c = b + 10
if (c < 0):
    a = b * a
else:
    a = a + 4
```

Data

a	-24
b	-8
c	2

CPU Core 1

Clock

T = 2

Toy Model of 2-Core CPU

ACCRE Advanced Computing Center for Research & Education

Instructions

```
c = b + 10
if (c < 0):
    a = b * a
else:
    a = a + 4
```

Data

a	3
b	-8
c	0

CPU Core 1

Instructions

```
if (b < 0):
    c = b * a
else:
    c = a + 4
c = c * 2
```

Data

a	2
b	3
c	1

CPU Core 2

Clock

T = 0

Notice: Each core is completely independent, different data, different instructions

Toy Model of 2-Core CPU

Instructions

```
c = b + 10
if (c < 0):
    a = b * a
else:
    a = a + 4
```

Data

a	3
b	-8
c	2

CPU Core 1

Instructions

```
if (b < 0):
    c = b * a
else:
    c = a + 4
c = c * 2
```

Data

a	2
b	3
c	6

CPU Core 2

Clock

T = 1

Notice: Each core is completely independent, different data, different instructions

Toy Model of 2-Core CPU

ACCRE
Advanced Computing Center for Research & Education

Instructions

```
c = b + 10
if (c < 0):
    a = b * a
else:
    a = a + 4
```

Data

a	-24
b	-8
c	2

CPU Core 1

Instructions

```
if (b < 0):
    c = b * a
else:
    c = a + 4
c = c * 2
```

Data

a	2
b	3
c	12

CPU Core 2

Clock

T = 2

Notice: Each core is completely independent, different data, different instructions

Modern Multi-Core x86 CPU Characteristics

- MIMD – Multiple instructions, multiple data
- Very smart independent cores
    - Cache multiple levels of data
    - Predict the outcome of if statements
    - Execute instructions out of order
    - BIG in terms of transistor count and power draw
- Mass produced by AMD and Intel, used for home and business PCs, servers
    - Inexpensive relative to development cost

Toy Model of a SIMD Processor

	CPU Core 1	CPU Core 2	CPU Core 3
a	3	2	0
b	-8	10	-8

	CPU Core 1	CPU Core 2	CPU Core 3
a	1	4	1
b	1	7	1

Shared Instructions

```
a = b + 10
a = a + 4
b = b * 8
```

Clock

T = 0

SIMD means single instruction, multiple data

Smaller to represent on the slide, also smaller in transistor count

Toy Model of a SIMD Processor

Data
a 2
b -8
CPU Core 1

Data
a 12
b 10
CPU Core 2

Data
a 2
b -8
CPU Core 3

Data
a 11
b 1
CPU Core 1

Data
a 17
b 7
CPU Core 2

Data
a 11
b 1
CPU Core 3

Shared Instructions

```
a = b + 10
a = a + 4
b = b * 8
```

Clock

T = 1

SIMD means single instruction, multiple data

Smaller to represent on the slide, also smaller in transistor count

Toy Model of a SIMD Processor

ACCRE Advanced Computing Center for Research & Education

Data
a 6
b -8
CPU Core 1

Data
a 16
b 10
CPU Core 2

Data
a 6
b -8
CPU Core 3

Data
a 15
b 1
CPU Core 1

Data
a 21
b 7
CPU Core 2

Data
a 15
b 1
CPU Core 3

Shared Instructions

```
a = b + 10
a = a + 4
b = b * 8
```

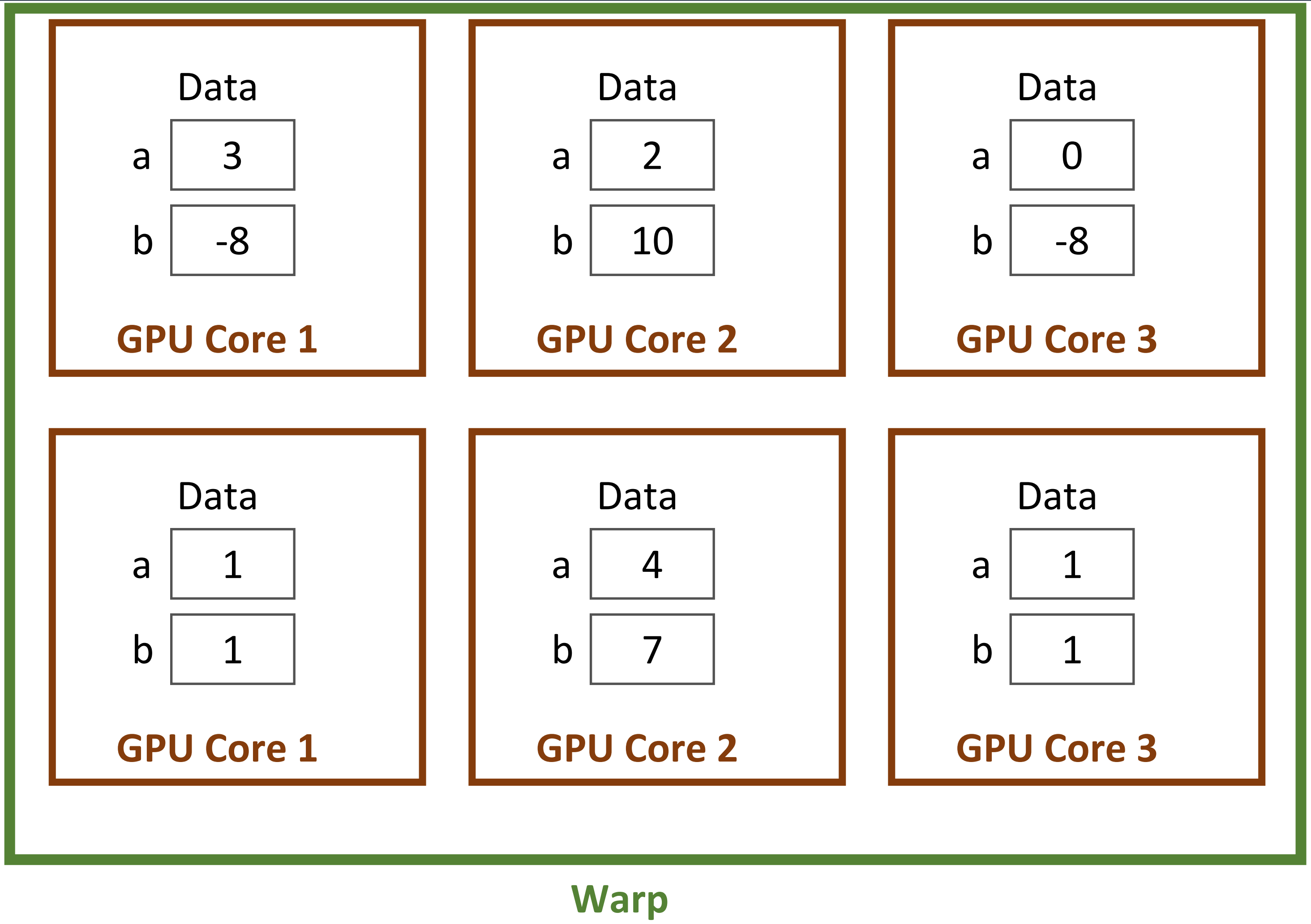
Clock

T = 2

SIMD means single instruction, multiple data

Smaller to represent on the slide, also smaller in transistor count

Toy Model of a SIMT Processor



Shared Instructions

```
b = b * 2
if b < 0:
    a = a * b
else:
    a = 0
```

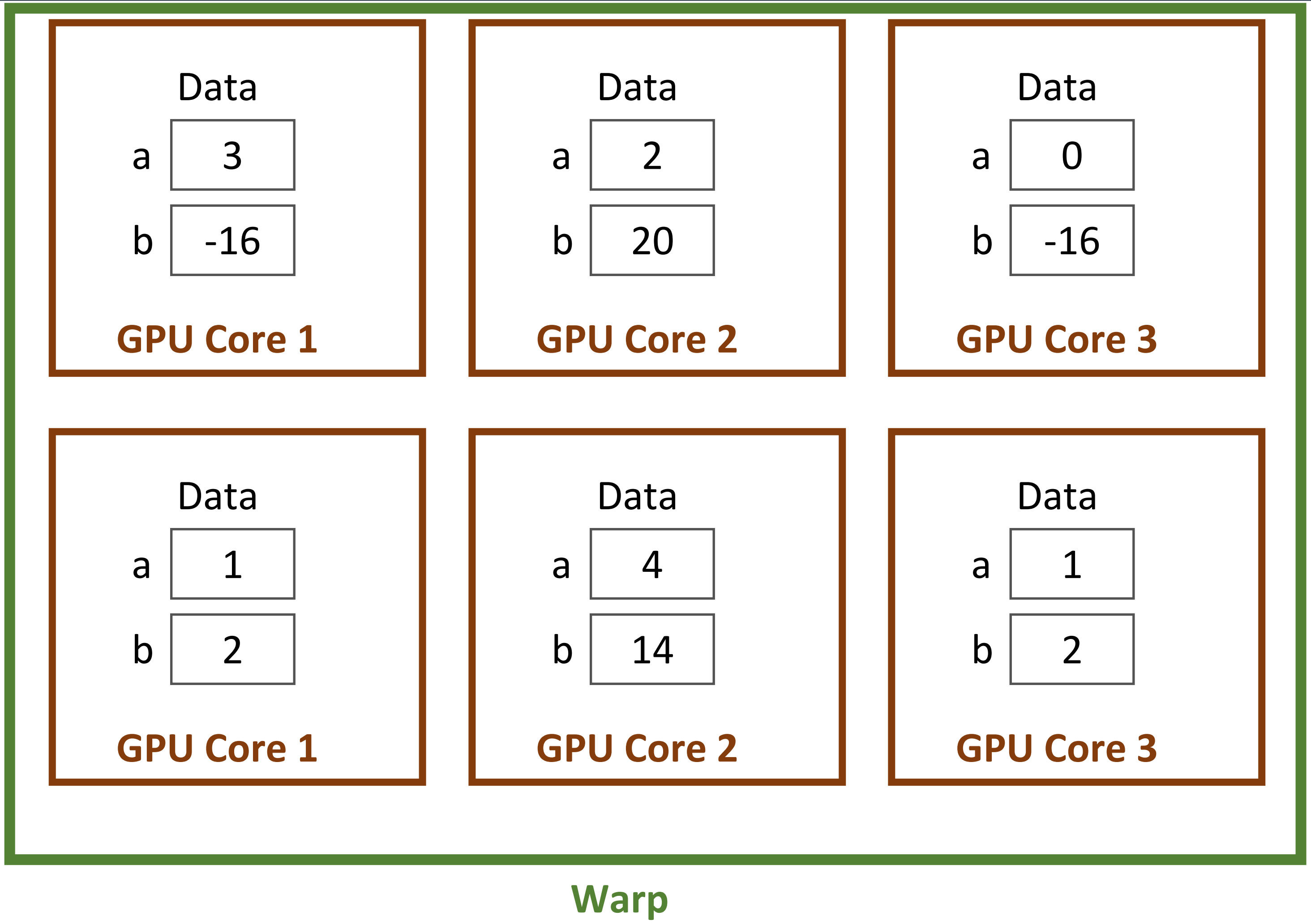
Clock

T = 0

SIMD means single instruction, multiple thread

Implemented in modern GPUs, confusing terminology, not the same as multithreading. Instructions implemented in lock-step, some cores can be masked

Toy Model of a SIMT Processor



Shared Instructions

```
b = b * 2
if b < 0:
    a = a * b
else:
    a = 0
```

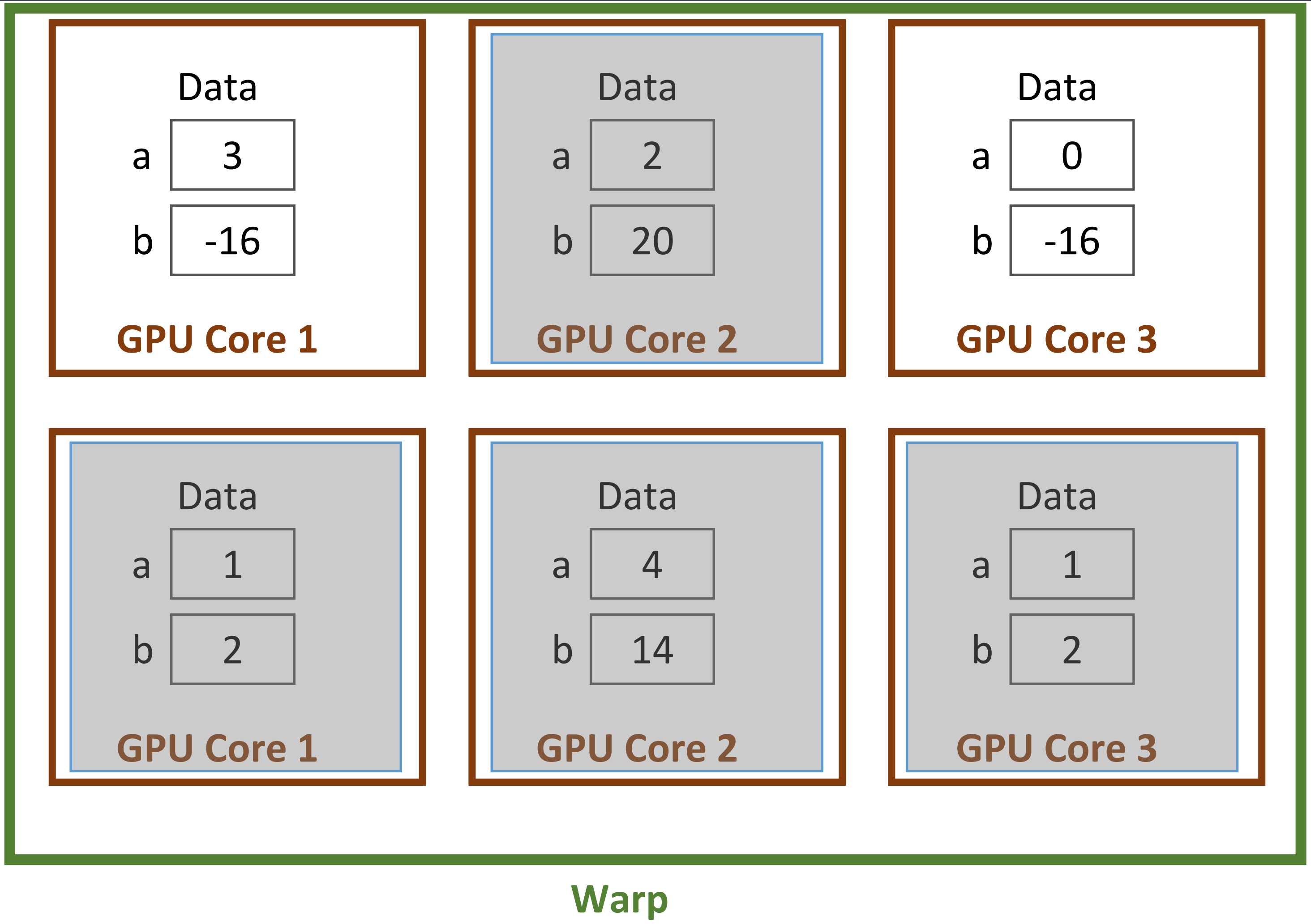
Clock

T = 1

SIMD means single instruction, multiple thread

Implemented in modern GPUs, confusing terminology, not the same as multithreading. Instructions implemented in lock-step, some cores can be masked

Toy Model of a SIMT Processor



Shared Instructions

```
b = b * 2
if b < 0:
    a = a * b
else:
    a = 0
```

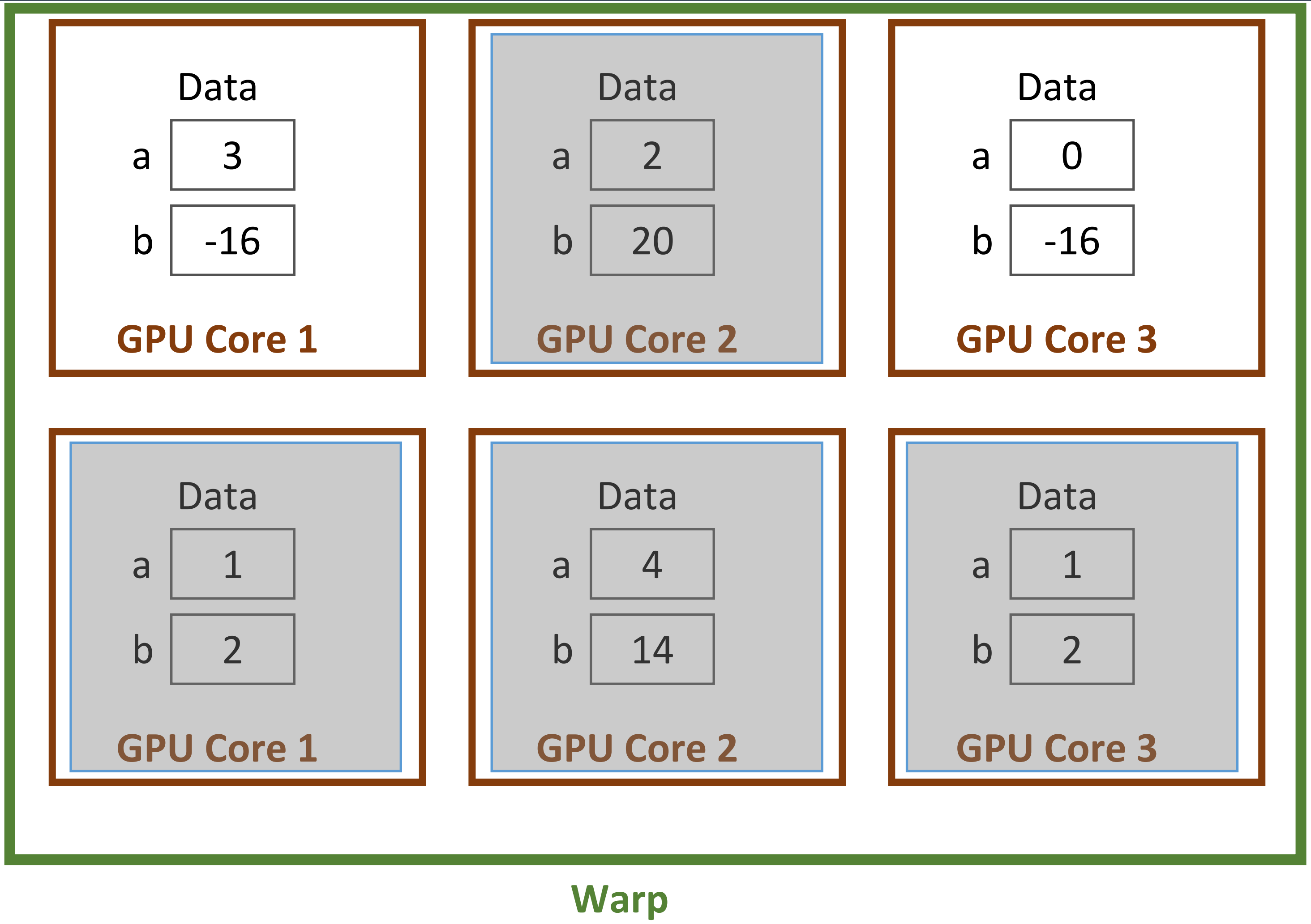
Clock

T = 2

SIMD means single instruction, multiple thread

Implemented in modern GPUs, confusing terminology, not the same as multithreading. Instructions implemented in lock-step, some cores can be masked

Toy Model of a SIMT Processor



Shared Instructions

```
b = b * 2
if b < 0:
    a = a * b
else:
    a = 0
```

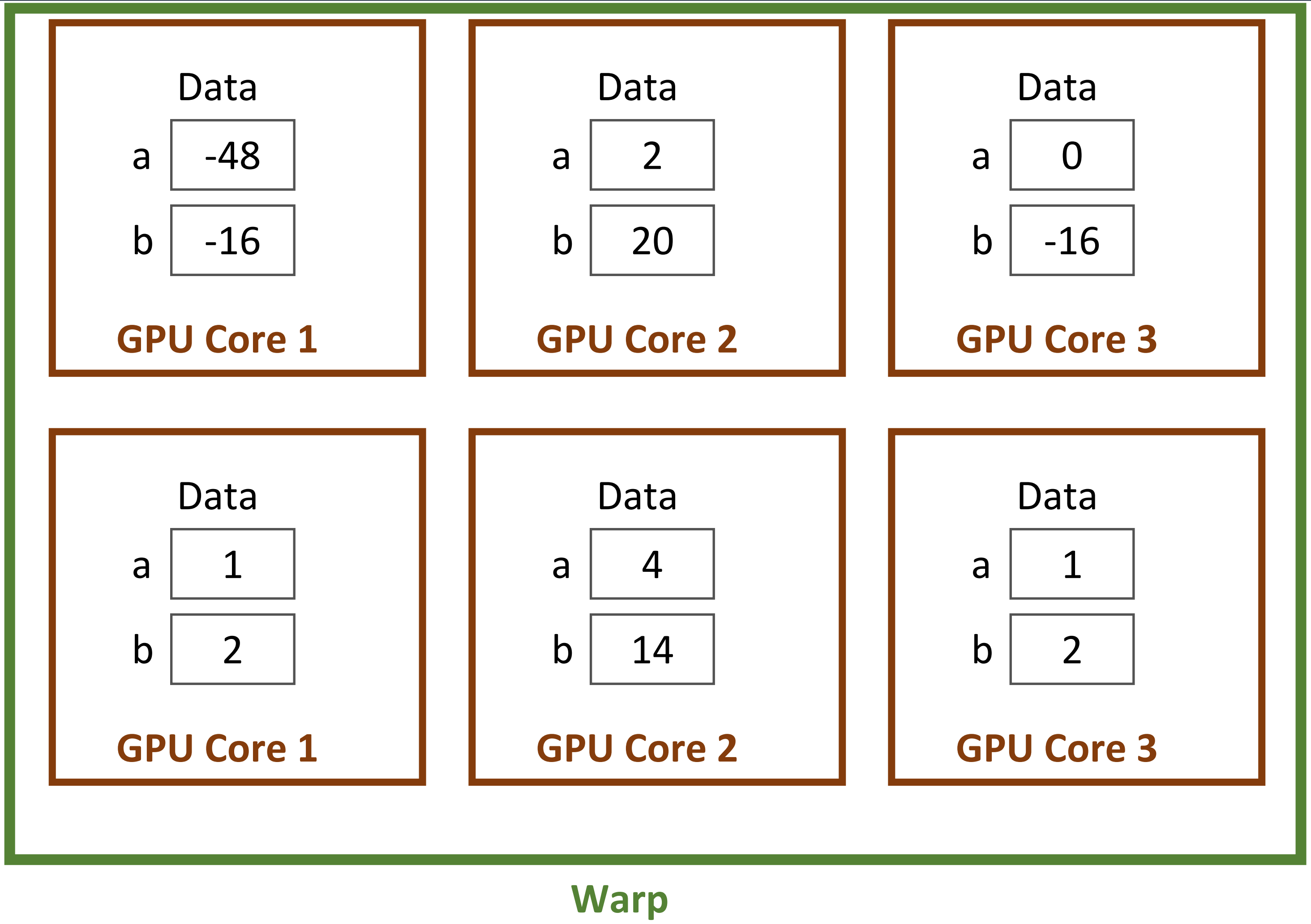
Clock

T = 3

SIMD means single instruction, multiple thread

Implemented in modern GPUs, confusing terminology, not the same as multithreading. Instructions implemented in lock-step, some cores can be masked

Toy Model of a SIMT Processor



Shared Instructions

```
b = b * 2
if b < 0:
    a = a * b
else:
    a = 0
```

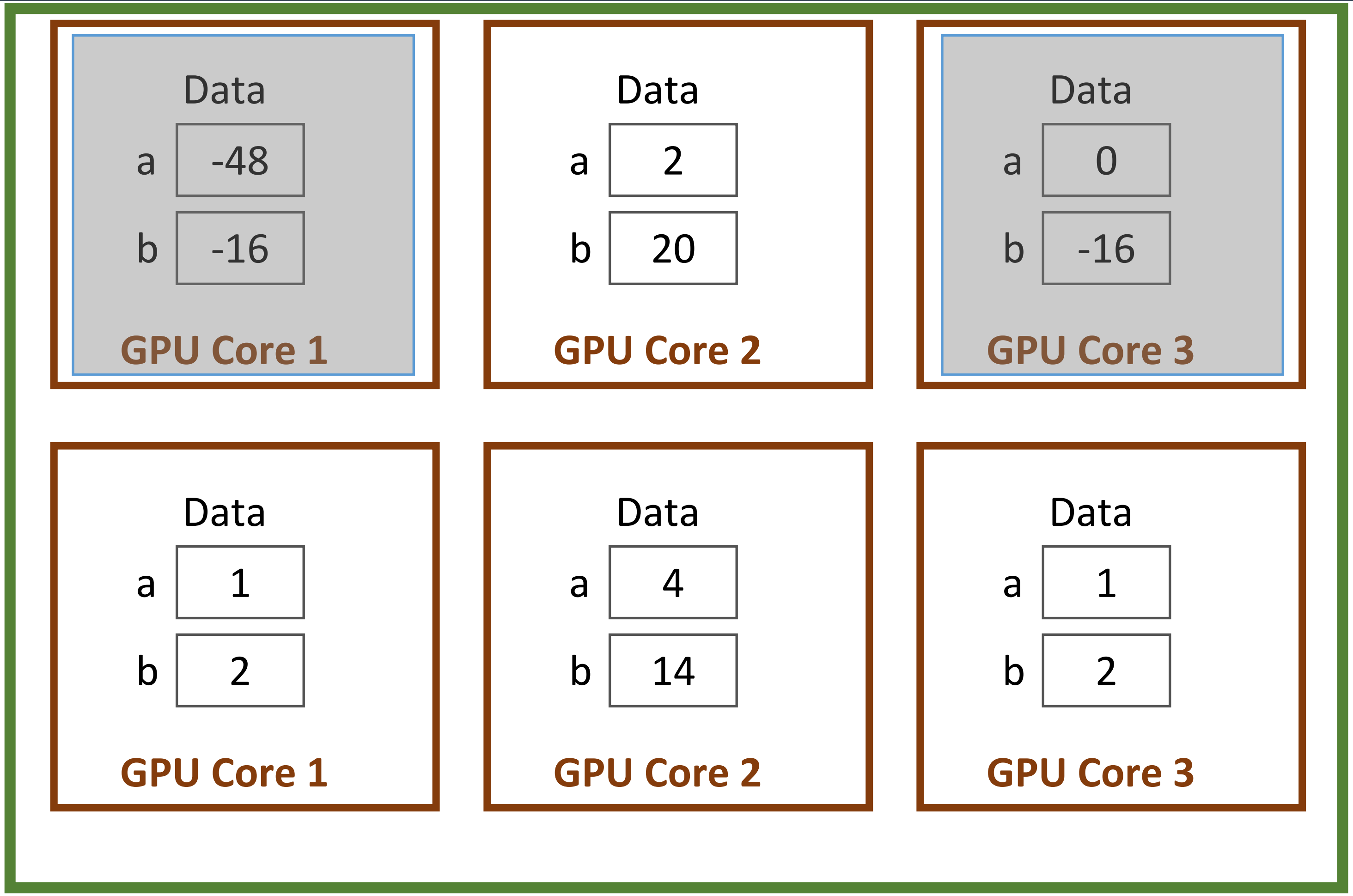
Clock

T = 5

SIMD means single instruction, multiple thread

Implemented in modern GPUs, confusing terminology, not the same as multithreading. Instructions implemented in lock-step, some cores can be masked

Toy Model of a SIMT Processor



Warp

Shared Instructions

```
b = b * 2
if b < 0:
    a = a * b
else:
    a = 0
```

Clock

T = 4

SIMD means single instruction, multiple thread

Implemented in modern GPUs, confusing terminology, not the same as multithreading. Instructions implemented in lock-step, some cores can be masked

Modern Nvidia GPU Characteristics

- SIMT – Single instructions, multiple threads
- Partially-independent simple cores sharing instructions across a “warp”
  - No fancy branch prediction (I think)
  - Each core small in terms of transistor count, power draw
- Mass produced by Nvidia as people love 3D video games
  - Inexpensive relative to development cost

Why do scientists use commodity x86 CPUs?

- x86 CPUs are inexpensive since so many are mass produced
- Relatively easy to program
- Up until recently, GPUs were difficult to code for

Why are scientists switching to GPUs

- More massive parallel computations more suited to many scientific computing tasks
- Becoming easier to program for, tools are maturing
- Can result in over 50x speedup for many tasks over traditional CPU computing
- Deep learning (image recognition, etc.) increasingly used in scientific practice and are very suited to GPUs.

Why not make non-video game processors?

- Unit cost can extremely expensive due to low demand and high development cost.
- Some facilities do use more specialized “manycore” architectures
- ACCRE tried out Xeon Phi manycore machines, but adoption was low

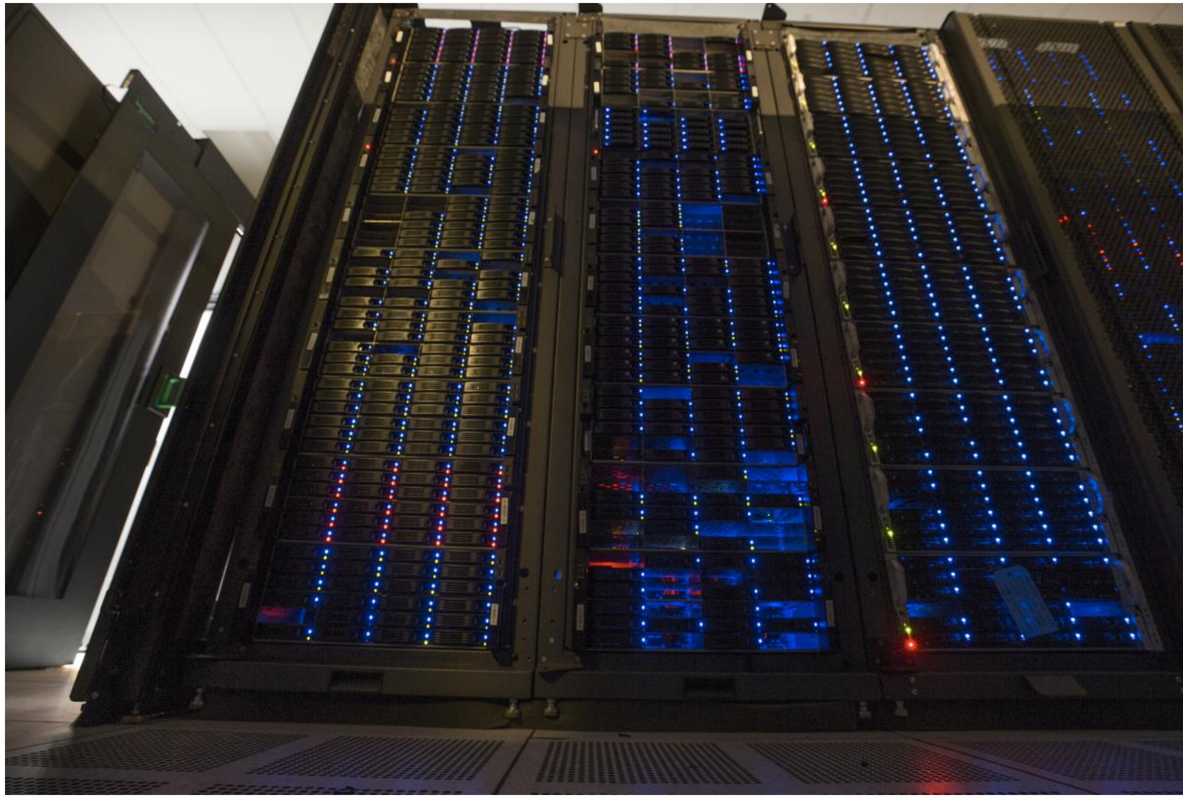
Tianhe-2, Guangzhou, China

Intel Xeon Phi manycore processor

ACCRE OVERVIEW

ACCRE Overview

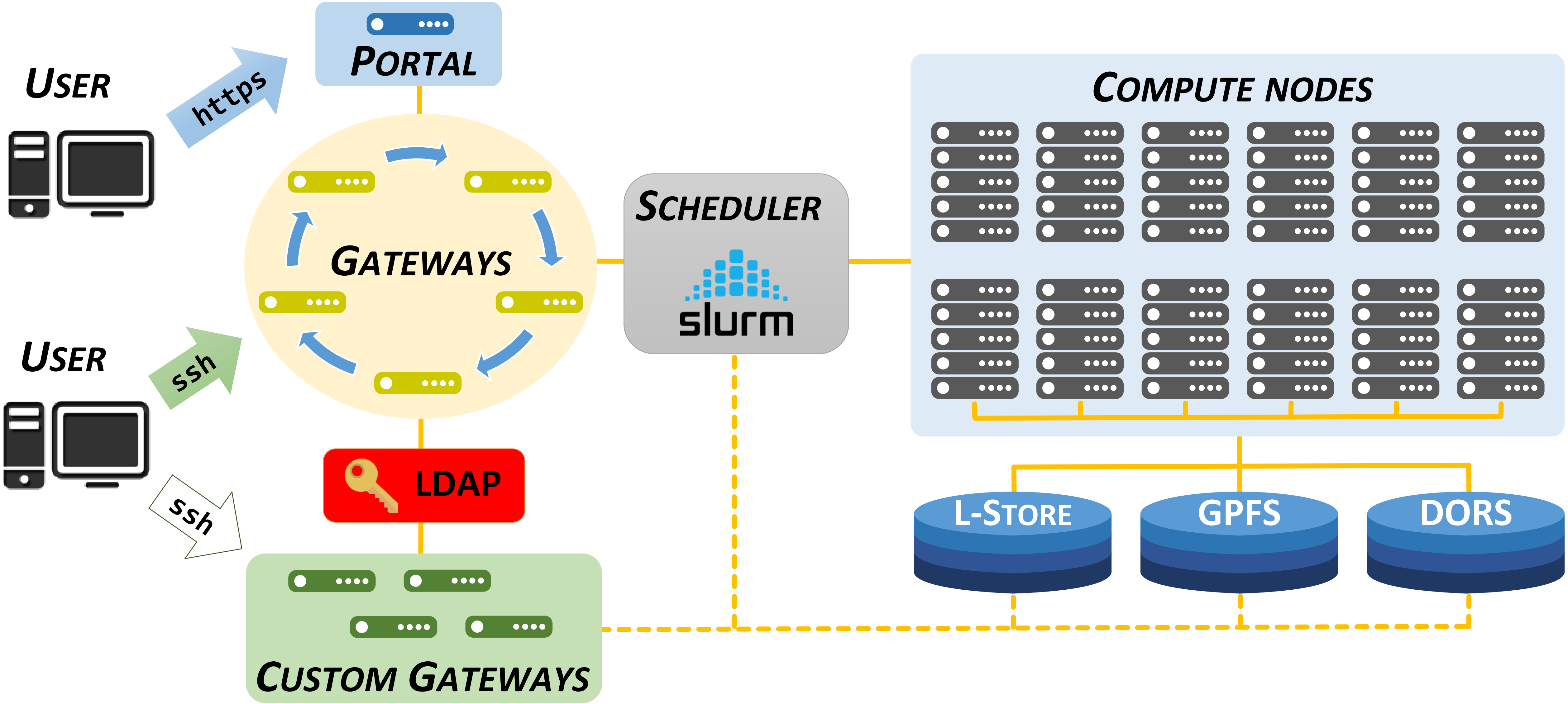
- ACCRE - Advanced Computing Center For Research and Education
- Centralized computing infrastructure for Vanderbilt researchers
- Operates as a co-op in which researchers share hardware
- ~10k CPU cores
- ~200 GPUs
- ~10PB disk storage + tape backups
- Optimized Scientific Software Stack
- Batch Job Scheduler
- Interactive resources (Jupyter, etc.)
- Staff of ~10



Using ACCRE vs Using Your Own Hardware

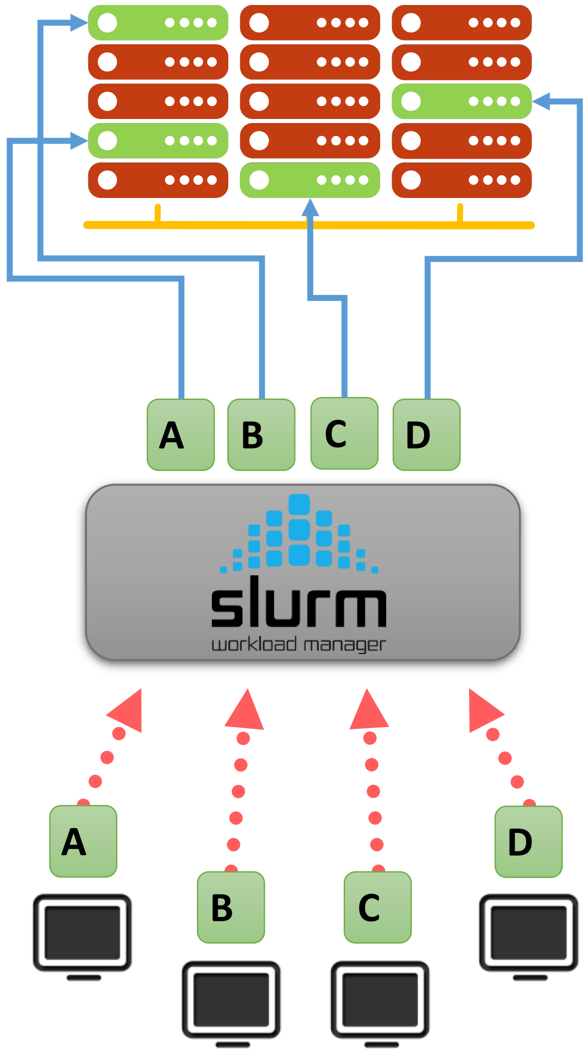
- Using your own hardware:
  - can use all resources immediately
  - have to set up software, system, and networking yourself
  - full administrative access (root)
- Using ACCRE:
  - must schedule resource requirements
  - can "burst" to use more resources than you own
  - dedicated staff maintain system and software stack
  - no administrative access (regular user)

ACCRE ARCHITECTURE



THE SCHEDULER

1. Execute user's workloads in the right priority order
2. Provide requested resources on compute nodes
3. Optimize cluster utilization



ACCRE is a Heterogeneous Cluster

- Different Memory Configurations and CPU Core Counts
  - Nodes with 64GB, 128GB, 192GB, 256GB, and 384GB
  - Between 8 and 128 CPU-cores per node
- Different Intel and AMD CPU Architecture Families
  - Variable clock speed, L1/2/3 Cache Memory
  - Additional Instruction Sets on Newer CPUs
- Specialized Accelerated Nodes
  - Nvidia 4x GPU Nodes (Maxwell, Pascal, Turing)

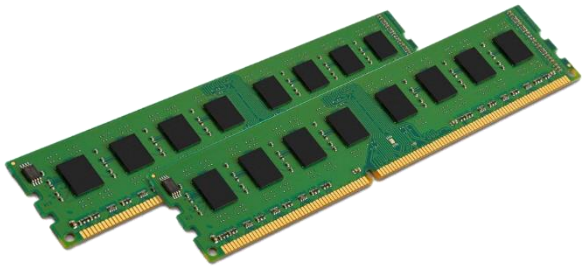
ACCRE Cluster Compute Nodes (out of date!!!)

ACCRE Advanced Computing Center for Research & Education

Regular nodes

Dual multicore CPUs

Random Access Memory



Newer

Older

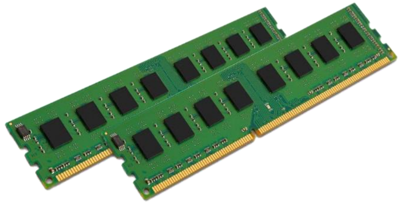
Family	*No. of cores*	*RAM* / GB	*No. of nodes*
Skylake	**16**	*256*	41
	24	*128*	52
Haswell	**12**	*128*	41
	16	*128*	120
		256	50
Sandy Bridge	**12**	*64*	31
		96	2
		128	193
		256	4
	16	*128*	3
Westmere	**8**	*128*	22
	12	*48*	16
Total	**8,292**	**82,432**	**575**

The GPU Nodes (Out Of Date!!!)

Accelerated nodes

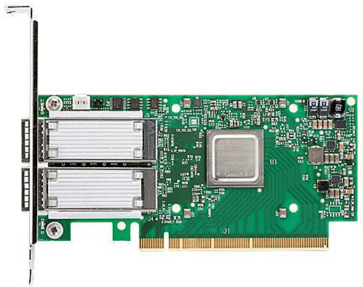
Dual multicore CPUs

Random Access Memory



4 x Nvidia GPU

25/40 Gbit/s RoCE Network



Newer

Older

Family	*No. of cores*	*RAM / GB*	*No. of nodes (GPUs)*
Nvidia Turing ***Intel Skylake***	**24**	*384*	21 (84)
Nvidia Pascal ***Intel Broadwell***	**8**	*256*	24 (96)
Nvidia Maxwell ***Intel Haswell***	**12**	*128*	10 (40)
Total	**816**	**15,488**	**55 (220)**

For those of you who like video games

Maxwell - NVIDIA GeForce GTX TITAN X (9xx series, 2015)

Pascal - NVIDIA TITAN Xp (10xx series, 2017)

Turing - NVIDIA GeForce RTX 2080 Ti (20xx series, 2018)

Ampere – (coming soon) NVIDIA A6000 (2022)

(workstation card, not for video games, similar to 30xx series)

Submitting Batch Jobs

Determining Requirements

- # of tasks
- # of cpu cores per task
- Memory (GB) per node or core
- # of GPUs
- Time allowed to complete job

Optimizing requirements results in jobs being scheduled sooner

For quad-GPU nodes, it is best to request ¼ of the total memory per GPU requested

Example Batch Job Script

A **batch job** consists of a sequence of commands listed in a file with the purpose of being interpreted as a single program.

SHEBANG

- Specify the script interpreter (Bash)
- Must be the first line!

SLURM DIRECTIVES

- Start with "#SBATCH":
  Parsed by Slurm but ignored by Bash.
- Can be separated by spaces.
- Comments between and after directives are allowed.
- Must be before actual commands!

SCRIPT COMMANDS

- Commands you want to execute on the compute nodes.

`myjob.slurm`

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --mem=1G
#SBATCH --time=1-06:30:00
#SBATCH --job-name=myjob
#SBATCH --output=myjob.out


# Just a comment
module load GCC Python
python myscript.py
```

Communicating with the scheduler

- Jobs are submitted with the “sbatch” command
  - i.e. “sbatch myjob.slurm”
- Similar jobs can be submitted with a single script in an arbitrarily large “job array”
- Automated systems such as the Open Science Grid are constantly submitting jobs
- “Standard” CPU nodes are combined into a single production partition
- Accelerated GPU nodes are separated into partitions by Nvidia Architecture generations – maxwell, pascal, turing, etc…
- As GPU nodes become more in demand, we want to understand their usage in more detail

Jobs Data for Analysis

Taken from the scheduler job database and put into CSV format

Job Record Format

```
JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED
```

- Job records are given as a CSV file with one row for each job

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- Each job gets a unique ID
- For a large group of similar jobs, a job-array may be used
- Array tasks have numbers after an underscore
- Array tasks treated as individual jobs

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- The account is the research group at Vanderbilt
- These are anonymized, the strings you see are from grocery store PLU code names for vegetables
- ACCRE has the mapping to actual research groups

Job Record Format

```
JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED
```

- Each account may have multiple users, these are individuals at Vanderbilt or robot users from automated submission systems
- These are anonymized, the strings you see are from a list of baby names
- ACCRE has the mapping to actual people

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- Used Memory is in MB
- Trailing "M" may be missing if memory usage is zero
- Note: the scheduler checks memory usage once per minute, very short running jobs may report 0, these may be excluded from memory usage analysis

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- Requested time is in d-hh:mm:ss or just hh:mm:ss

- Used time is in d-hh:mm:ss or just hh:mm:ss
- Watch out for really short jobs! Bad use of cluster resources or failure

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- Nodes is number of servers used for this job
- Multi-node jobs are uncommon at ACCRE, indicate usage of the RoCE hardware on these GPU jobs

- CPUs is the total number of CPU-cores allocated to the job
- For multi-node jobs this includes all nodes
- GPU jobs can automatically use ¼ of all CPUs on a machine per-GPU requested, so this field is not particularly meaningful in this analysis

- GPUs is the total number of GPU cards allocated to the job
- For multi-node jobs this includes all nodes
- Each node has 4 GPUs, over 4 GPUs indicates a job using RoCE

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- Each partition should be generally analyzed separately

Job Record Format

JOBID,ACCOUNT,USER,USEDMEM,REQTIME,USEDTIME,NODES,CPUS,GPUS,PARTITION,EXITCODE,STATE

32880657,malanga,arline,18.66M,2-00:00:00,00:13:13,1,2,1,pascal,0:0,COMPLETED
32880701,glasshouse,brady,0,05:00:00,00:00:22,1,3,1,maxwell,0:0,COMPLETED

- Exit code should be "0:0" for successful jobs
- Lots of long-running failed jobs are a waste of resources
- Any job with a state of CANCELLED, PENDING, or RUNNING may generally be ignored in this analysis

Analysis Questions

1. What is the distribution of per-GPU main memory usage over all runtime-weighed jobs in each partition?

   Knowing this will help ACCRE to understand our users memory needs for future hardware purchases.

2. What is the distribution of the number of GPUs in each job (runtime-weighted) for each partition?

   What fraction of runtime-weighted and GPU-weighted jobs are using more than 4 GPUs and thus probably using the RoCE networking?

   Is this fraction different for each partition?

3. What is the total runtime usage per-gpu (i.e. multiply runtime by the number of gpus) in each of the 3 partitions over the last year?

4. What is the distribution of different groups and users accessing each partition?

   In each partition, who are the top users, and do they represent a majority of the runtime-weighted jobs on the partition?

5. Currently there is a 5 day limit on runtime for GPU jobs, although some users have been asking for extensions.

   What is the distribution of requested runtime and actual runtime on jobs on each partition?

   Do users really need more time, or are they simply always requesting the maximum?