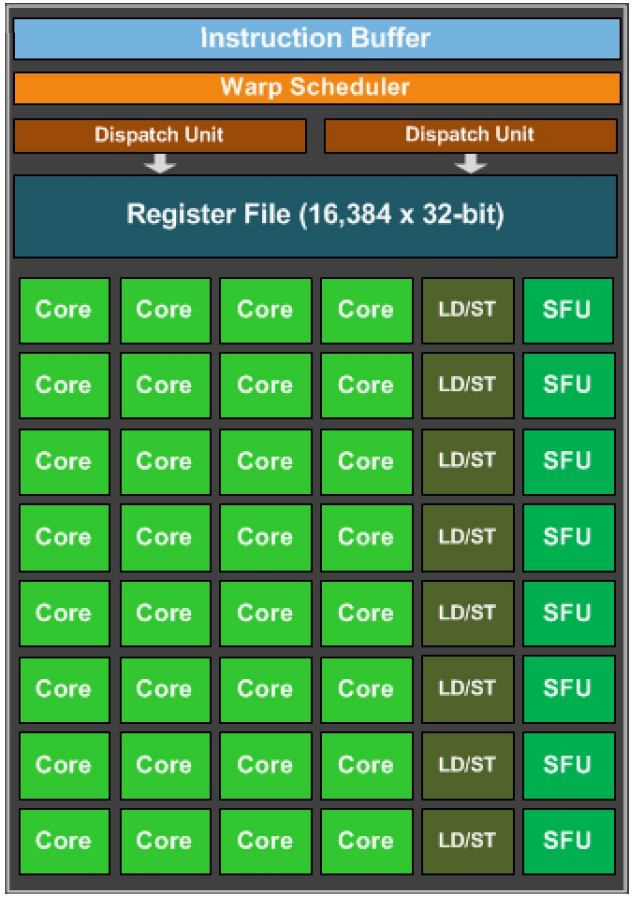
CSE113: Parallel Programming

March 4, 2022

- **Topics**:
  - Discuss HW 5
  - GPU programming



Announcements

- HW 4 is due today!
  - Sanya has office hours
  - Piazza will be monitored until 5 pm
- HW 5 is released today by midnight
  - Due the day before the final
- HW 3 grades are coming either by midnight tonight or by Monday
  - Wrapping a few things up

Today’s Quiz

- We will cancel quizzes for the rest of the quarter;
  - It’s a busy time for everyone and I want to make sure we can support you in HW 5 as much as possible.

Previous quiz

Review

Livelock vs. Starvation

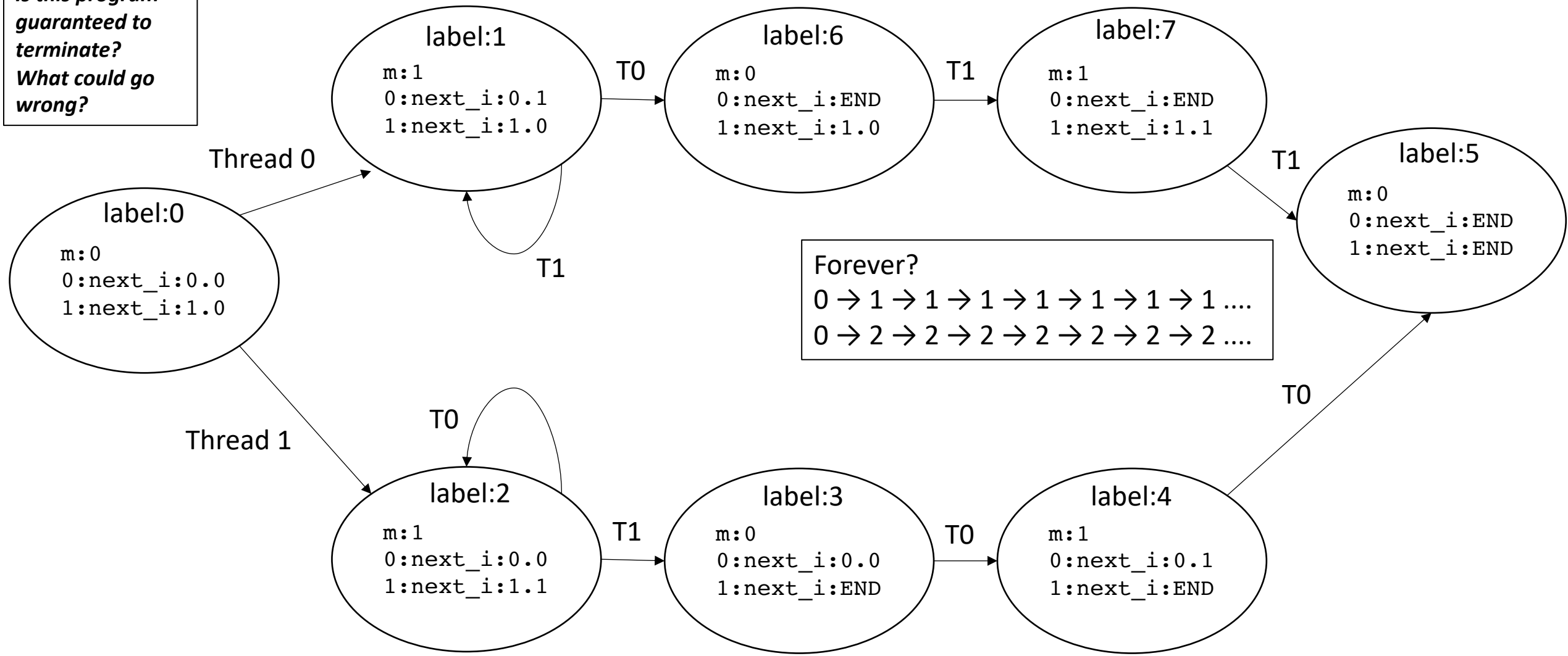
Thread 0:

```
0.0: while(CAS(&m,0,1) == false); //lock
     // critical section
0.1: m.store(0); //unlock
```

Thread 1:

```
1.0: while(CAS(&m,0,1) == false); //lock
     // critical section
1.1: m.store(0); //unlock
```

Is this program guaranteed to terminate? What could go wrong?



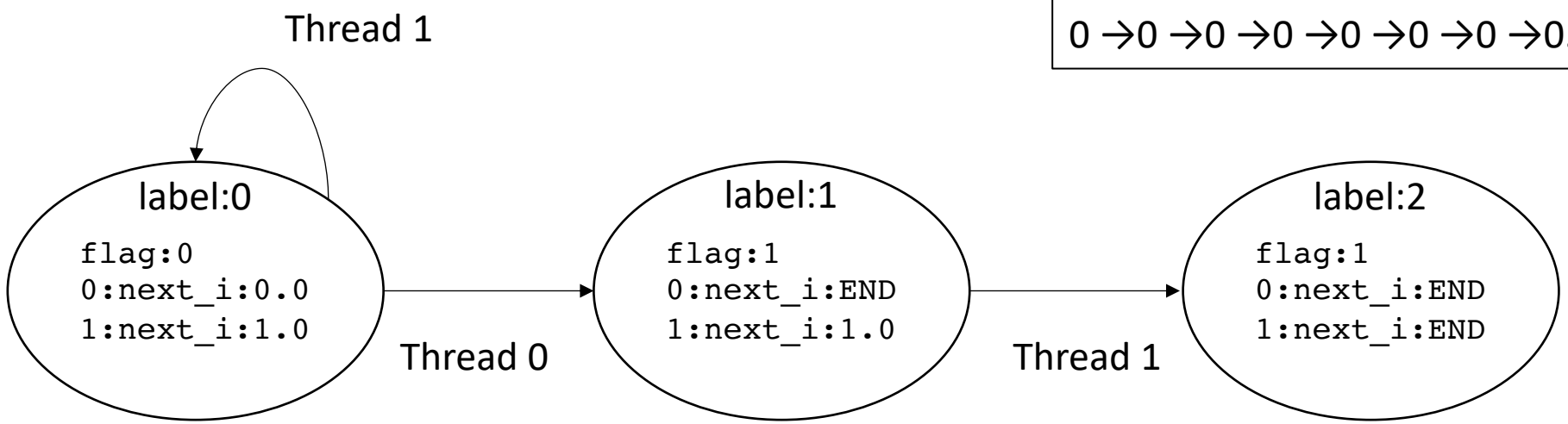
Thread 0:

```
0.0: flag.store(1);
```

Thread 1:

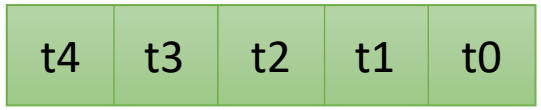
```
1.0: while(flag.load() == 0);
```

Forever?
0 →0 →0 →0 →0 →0 →0 →0….

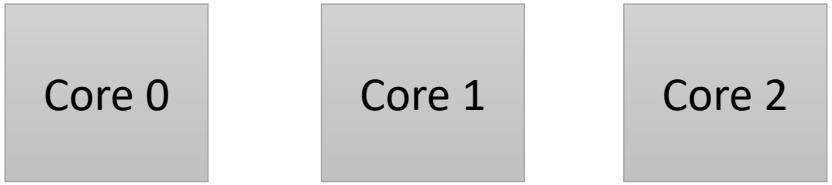


A power-saving scheduler

Program with 5 threads



thread pool



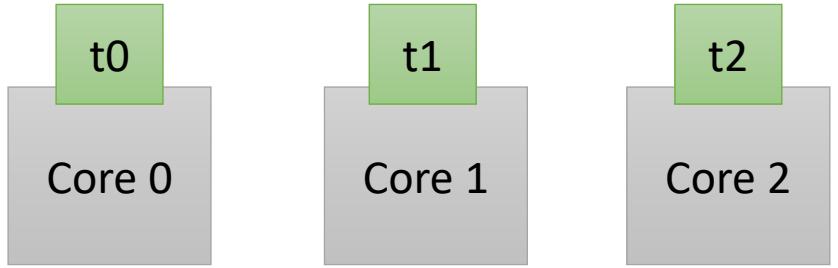
Device with 3 Cores

finished threads

A power-saving scheduler

Program with 5 threads

thread pool



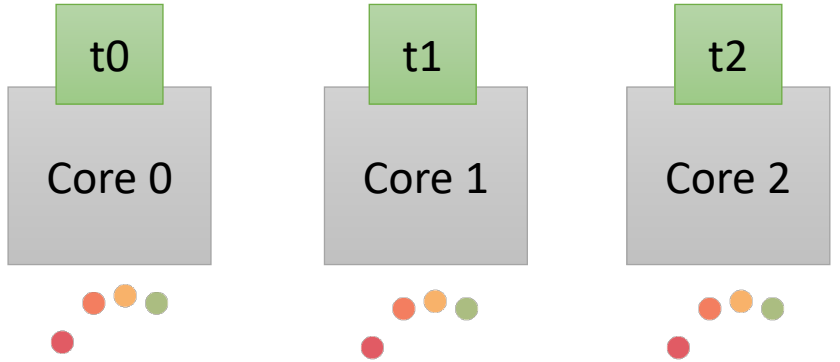
Device with 3 Cores

finished threads

A power-saving scheduler

Program with 5 threads

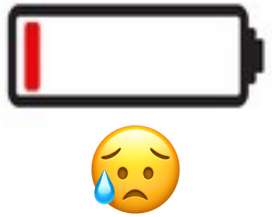
thread pool



Device with 3 Cores

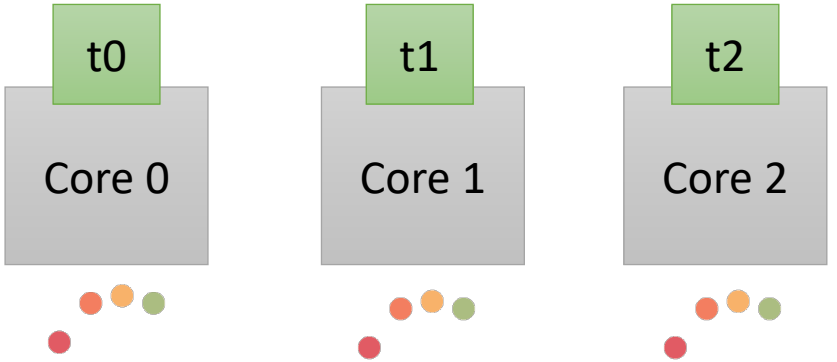
finished threads

A power-saving scheduler



Program with 5 threads

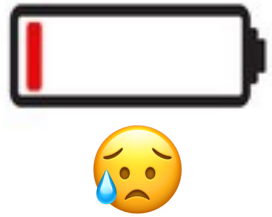
thread pool



Device with 3 Cores

finished threads

A power-saving scheduler

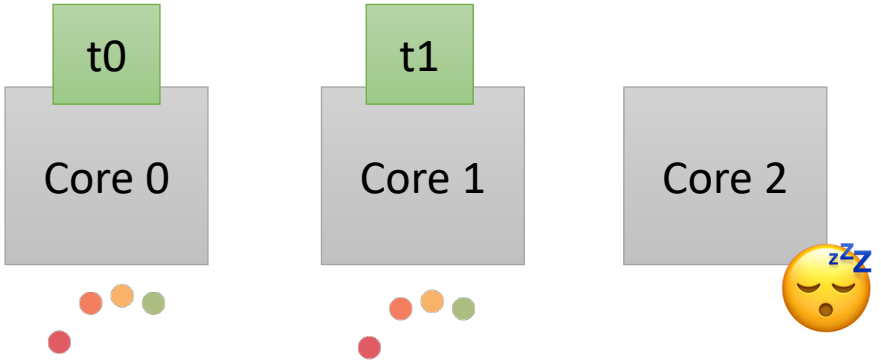


Program with 5 threads

thread pool

t2

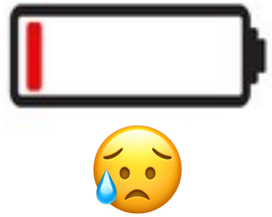
preempted



Device with 3 Cores

finished threads

A power-saving scheduler

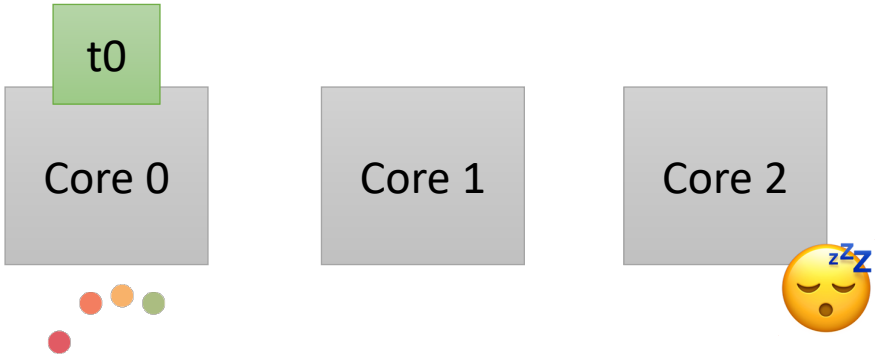


Program with 5 threads

thread pool

t2

preempted

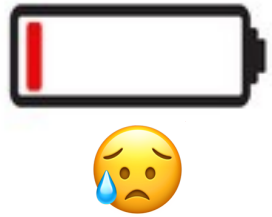


t1

finished threads

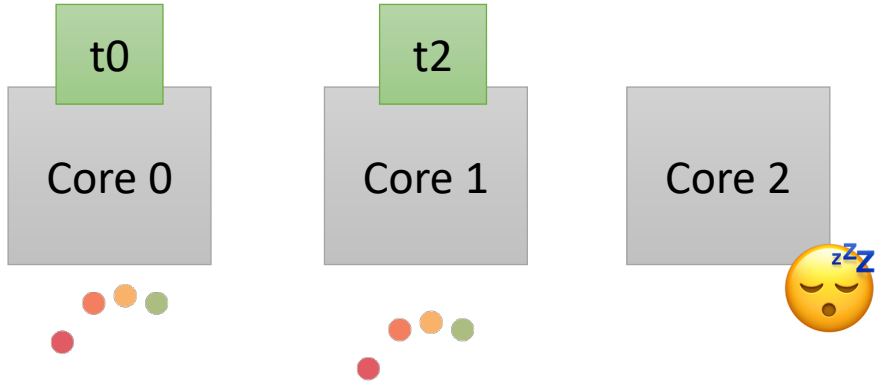
Device with 3 Cores

A power-saving scheduler

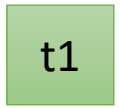


Program with 5 threads

thread pool



Device with 3 Cores



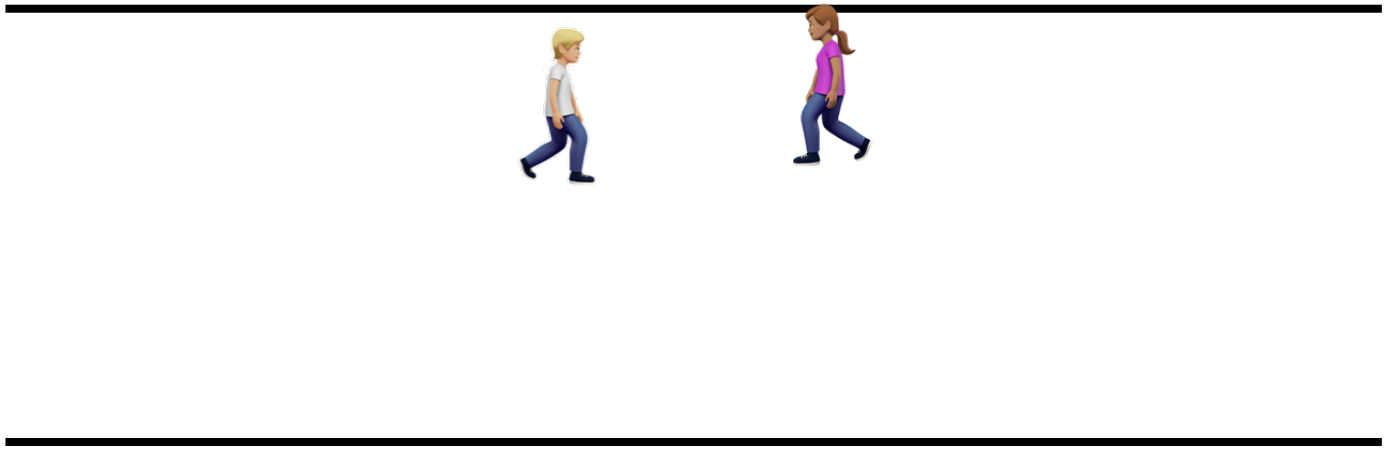
finished threads

A different type of non-termination

Hallway problem

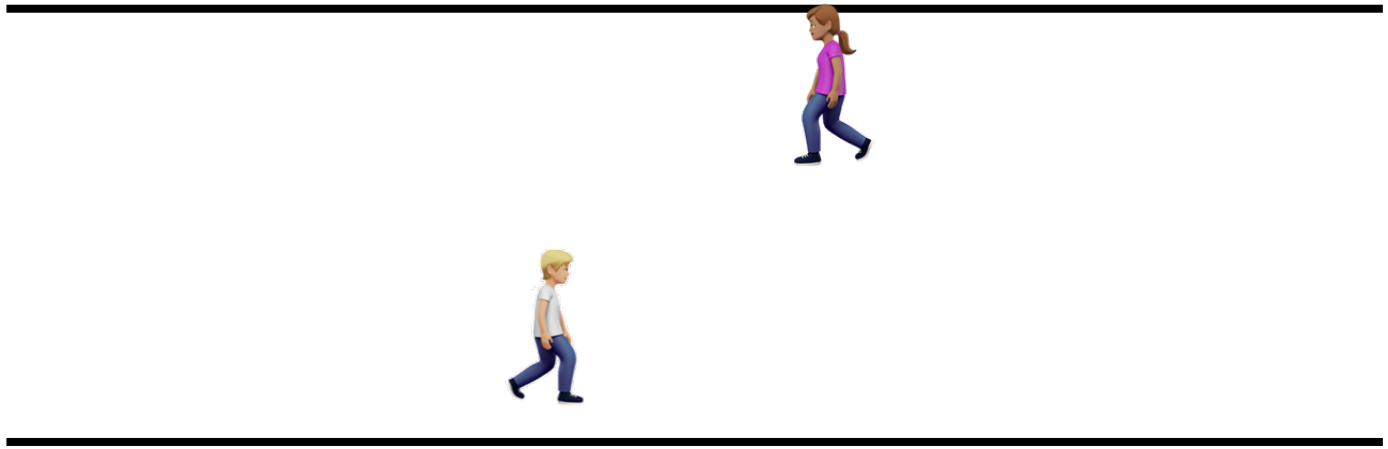
A different type of non-termination

Hallway problem



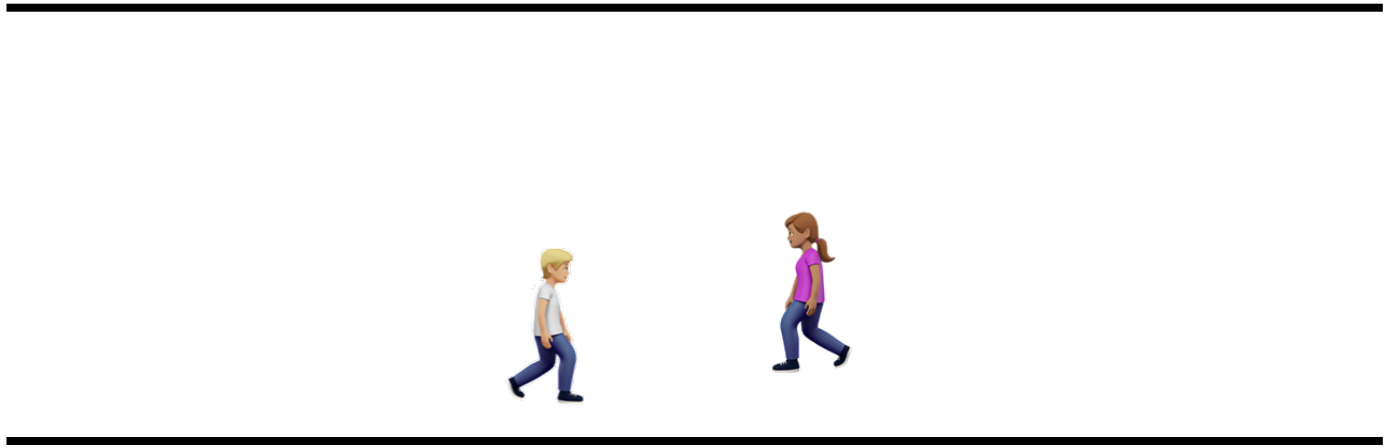
A different type of non-termination

Hallway problem



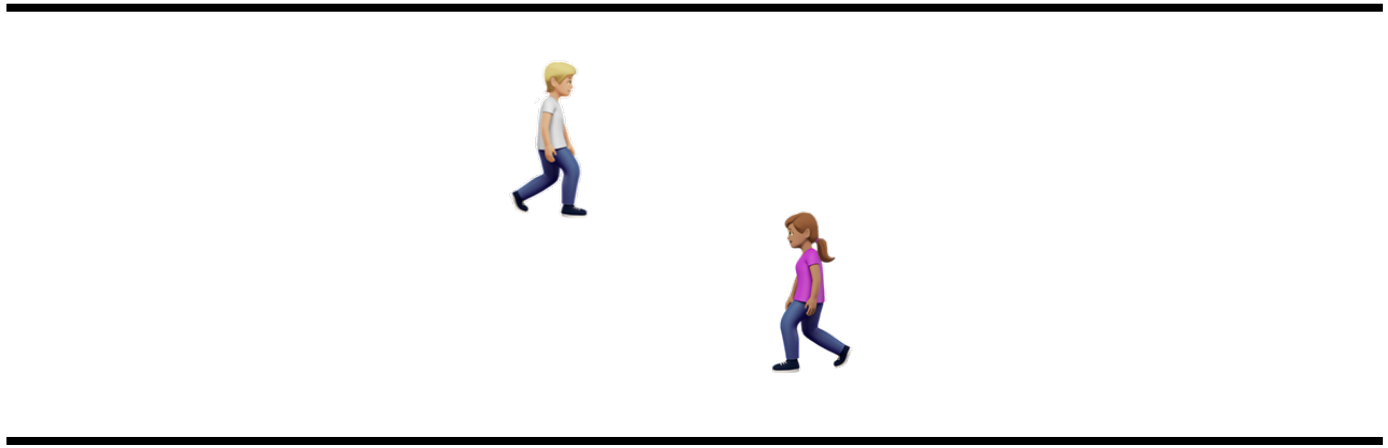
A different type of non-termination

Hallway problem



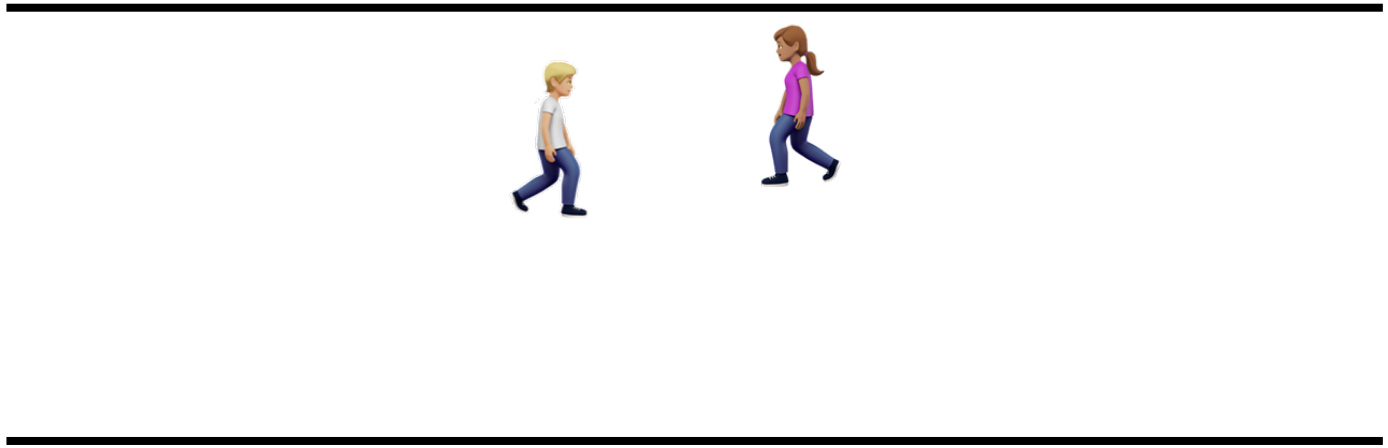
A different type of non-termination

Hallway problem



A different type of non-termination

Hallway problem



Can they dance around each other forever?

Thread 0:

```
... do {
0.0   x.store(0);
0.1 } while (x.load() != 0)
```

Thread 1:

```
... do {
1.0   x.store(1);
1.1 } while (x.load() != 1)
```

Each thread stores their thread id,
and then loads the thread id. It loops while
it doesn't see its id

Each thread gets a chance to execute, but they
get in each others way.

This is called a livelock

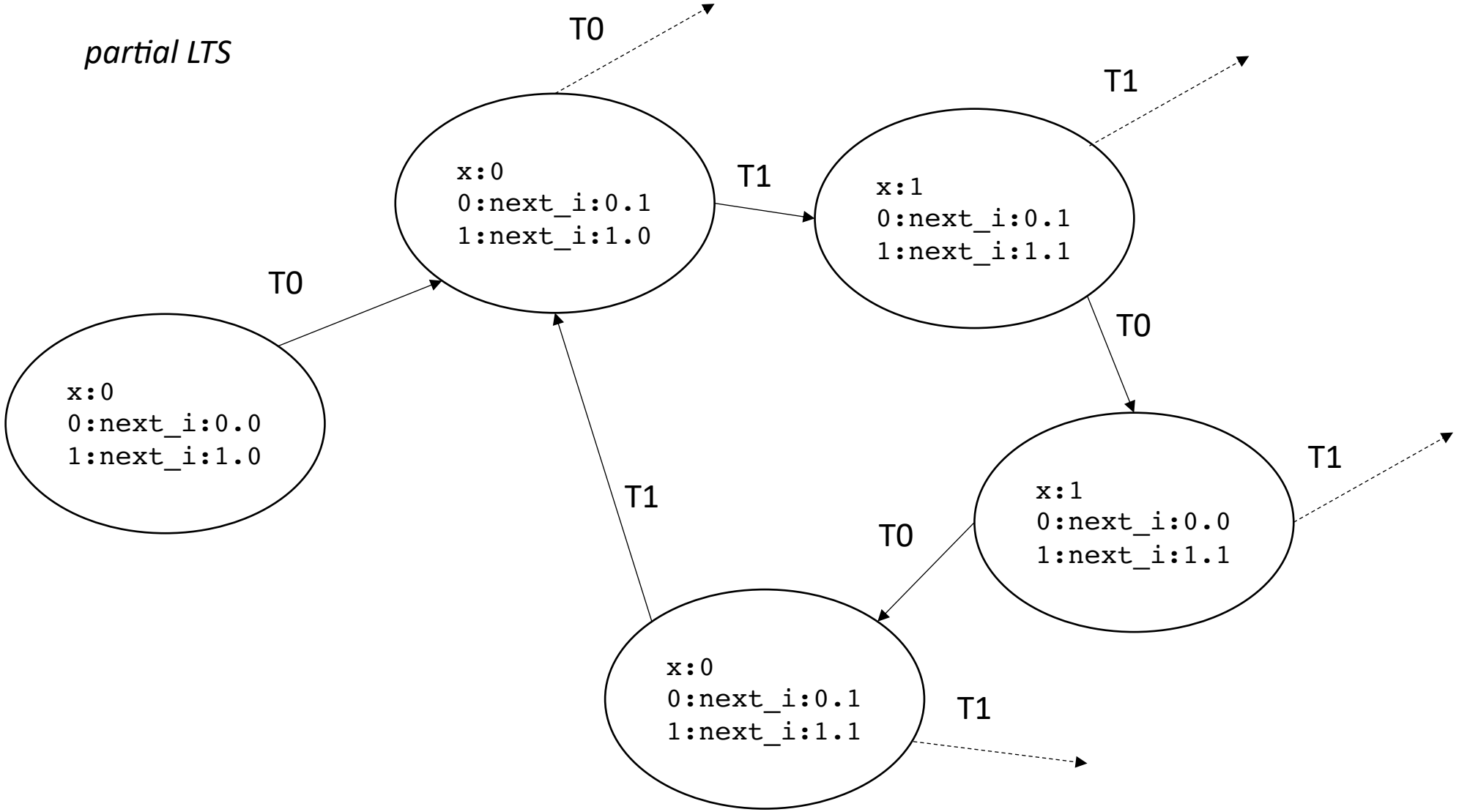
Thread 0:

```
... do {
0.0   x.store(0);
0.1 } while (x.load() != 0)
```

Thread 1:

```
... do {
1.0   x.store(1);
1.1 } while (x.load() != 1)
```

partial LTS



Livelock

- All threads are getting a turn, but they are constantly getting in each others way

- Requires a different type of fairness
  - Strong fairness
  - All threads get a turn, and for a variable amount of time
  - Tends to work on CPU threads due to natural variance of processors and preemption
  - Can actually hang on GPUs - much more regular scheduler

GPU history

- Very intertwined with video games and graphics, even if they aren't on the front page of Nvidia anymore!

- Early designs were very specialized.
  - Next came coarse-grained APIs.
  - Then came programable shaders
  - Then came general programming languages for one vendor (CUDA)
  - Then came general programming languages for many vendors (OpenCL, Vulkan)

- Used to ship in video game cartridges

- They are now in nearly every single mainstream device

GPU Shortages?

- Cryptocurrency:
  - 2018 reported tripling of GPU prices and shortages due to increase demand from miners.
  - Still happening will lots of market fluctuations and supply chain issues
  - Still plenty of GPUs in your phone, laptop, etc. ☺

Teaching GPU programming

- This is difficult!
- Nvidia GPUs have the most straightforward programming model (CUDA). They also have great PR.
- It is extremely difficult to get a class of 60 students access to Nvidia GPUs these days.
  - AWS? Expensive and often oversubscribed w.r.t. GPUs
  - Department? ML folks get priority and super computing clusters are painful

Going forward

- The GPU programming lectures will use CUDA
  - It is widely used
  - The programming model is straightforward

- Homework will use WebGPU, because it is widely supported
  - *There are more non-Nvidia GPUs in this room than Nvidia GPUs*
  - *There are more non-Nvidia GPUs in the world than Nvidia GPUs*

Going forward

- The homework uses Javascript as its ”CPU” language, and webGPU as its ”GPU” language.
  - We should be able to adopt to new language!

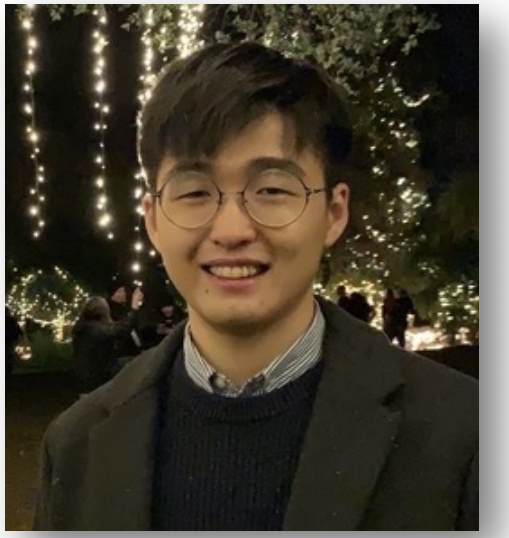
- We have provided generous skeletons for the homework. We can go over some javascript, but it is a high-level language and should not be hard to figure out what you need to do.

- The WebGPU portion is straight forward and I will provide a mapping directly from what we talked about to what you need.

Homework 5 - first look

- It is the first time offering this homework, so feedback is very welcome and we will be generous with support.

- Thanks to Mingun Cho who basically did all the work setting up the assignment!



Homework 5- first look

- Prerequisits
  - Google Chrome Canary
  - (if you have linux, Google Chrome Dev might work)

- Why do we need the Canary?
  - WebGPU is new and support is inconsistent on main (Although it is officially supported)

  - Perhaps more interesting is the shared array buffer.

Homework 5- first look

- Javascript shared array buffer:
  - How javascript threads can actually share memory
  - Similar to memory in C++

Shared memory and high-resolution timers were effectively disabled at the start of 2018 in light of Spectre. In 2020, a new, secure approach has been standardized to re-enable shared memory. With a few security measures, `postMessage()` will no longer throw for `SharedArrayBuffer` objects and shared memory across threads will be available:

As a baseline requirement, your document needs to be in a secure context.

Your application will be in a secure context (you are writing and running locally!)

Homework 5- first look

- You will also need Node.js to run a local web server.

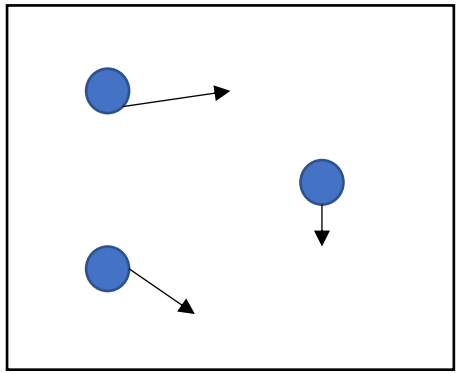
Homework 5- first look

- Let’s have a look!

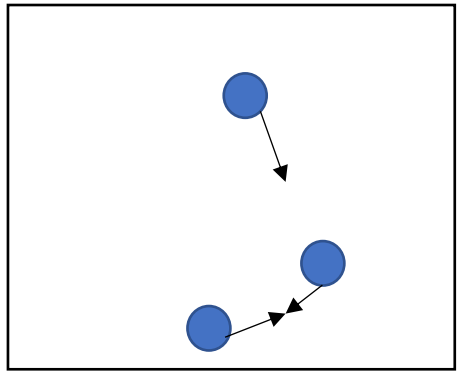
Homework 5- first look

- Your assignment:
  - N-body simulation

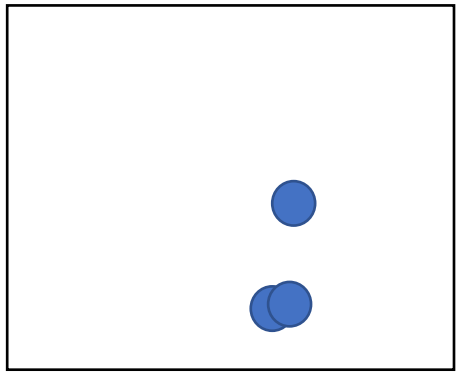
- Each particle interacts with every other particle



time = 0



time = 1

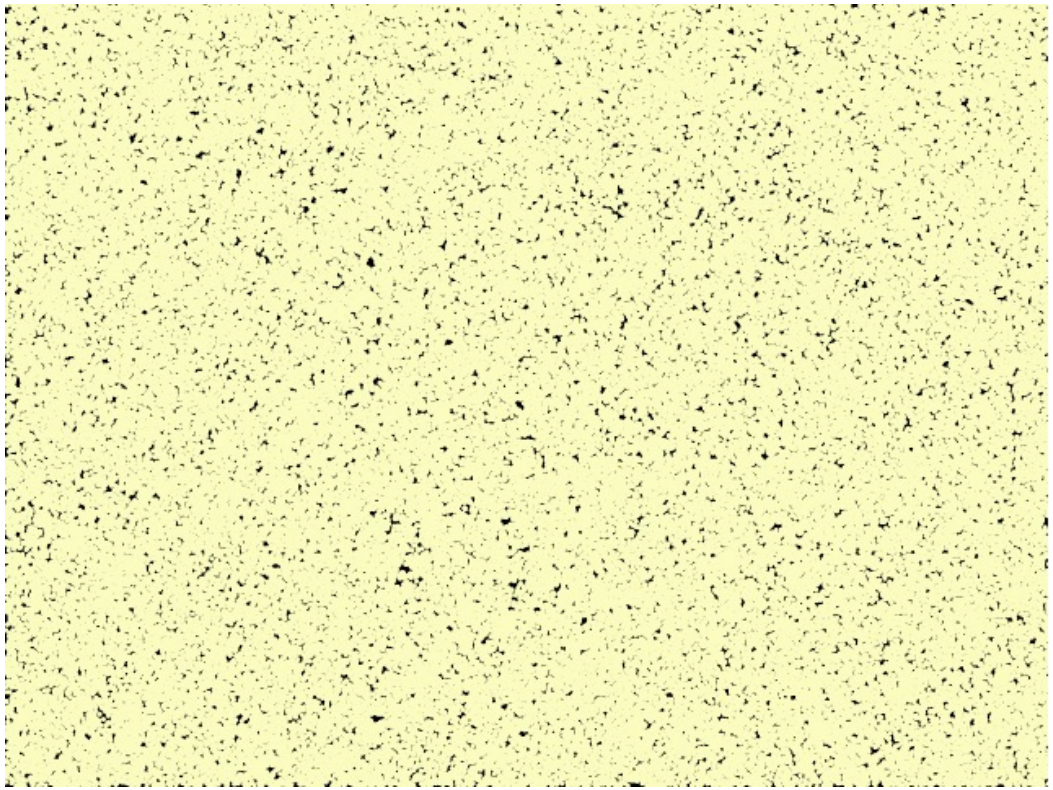


time = 2

Examples

- Gravity:

- Boids:
  - https://en.wikipedia.org/wiki/Boids



Your homework

- white board example

Your homework

- Part 1 of your homework will do this on a single javascript thread

- Demo

Your homework

- Looks good, but with more particles, things start to go slower...

Your homework

- Looks good, but with more particles, things start to go slower...
- Part 2 of the homework is to implement with multiple CPU threads using javascript webworkers
  - Should get around a linear speedup
- Part 3 is to implement with webGPU
  - Should get a BIG speedup!
- You need to explore how many particles you can simulate while keeping a 60 FPS framerate.

Let’s look at the code and see some javascript

- look at HTML
- how to print to the console (with interpolation).
  - Syntax errors
- how to interact with HTML
  - overwrite elements
  - modify elements

Shared Array Buffer

- Like Malloc, allocates a ”pointer” to a contagious array of bytes
- Can pass the “pointer” to different threads (webworkers)
- Need to instantiate a typed array to access the values
- Example

More Javascript

- Objects

Web Workers

- How to do multi-threading in javascript

- Async
  - Concurrent (executes on the same thread)
  - Good for I/O and user interactions

- Web Workers are guaranteed to execute on multiple cores
  - Better for compute intensive applications
  - Better performance

How to use?

- Create a new worker with a file
  - Doesn’t do anything yet
- File contains a function: “on message”
- Main file calls ”post message” to start the thread along with arguments
- Worker sends a message back to the main file, it can catch the data

Web Workers

Example with data and arrays

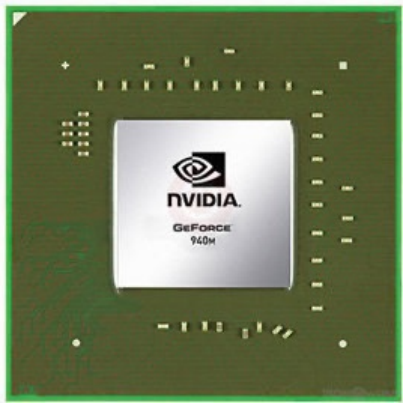
Continue with code

- Let’s keep going through the code

Start on GPU lectures

Programming a GPU

The GPU in
my PhD laptop



Nvidia 940m
1.8 Billion transistors
33 TDP
Est. $130

Fight!

The CPU in
my professor
workstation

Intel i7-9700K
2.16 Billion transistors
95 TDP
Est. $316

https://www.techpowerup.com/gpu-specs/geforce-940m.c2648
https://www.alibaba.com/product-detail/Intel-Core-i7-9700K-8-Cores_62512430487.html
https://www.prolast.com/prolast-elevated-boxing-rings-22-x-22/

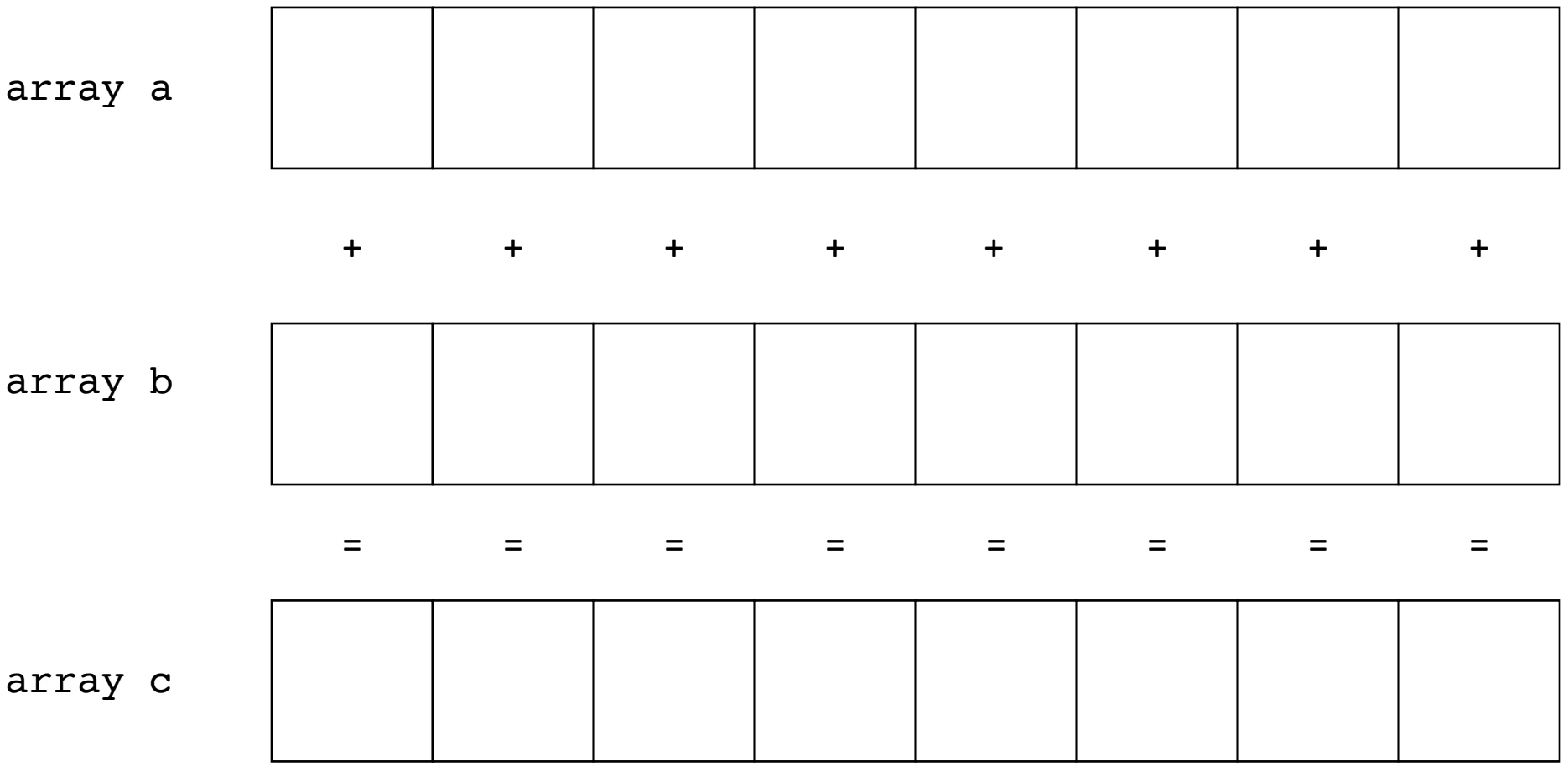
Programming a GPU

- The problem: Vector addition

Embarrassingly parallel

Computation can easily be divided into threads

Thread 0 - Blue
Thread 1 - Yellow
Thread 2 - Green
Thread 3 - Orange



Programming a GPU

- The problem: Vector addition

- Who can do it faster?

Lets set up the CPU

- CPU code

Now for the GPU

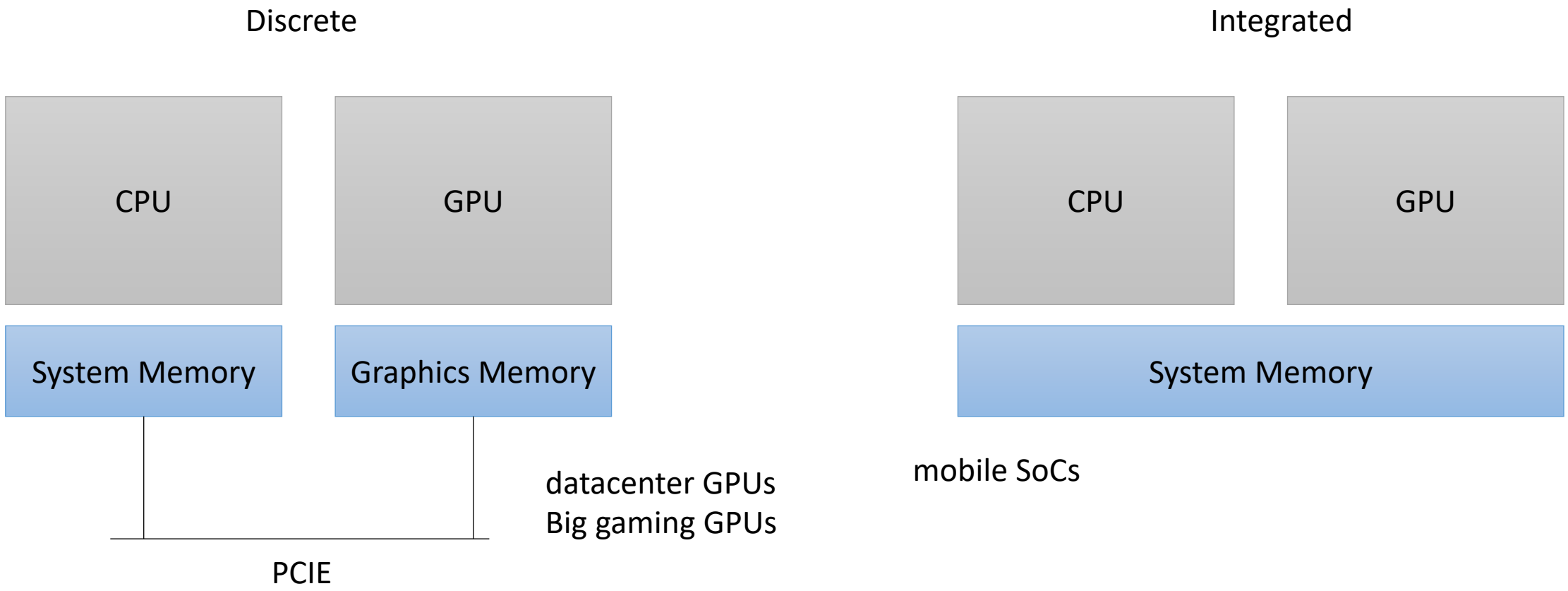
- Its going to take a bit of work....

GPU set up

- We need to allocate and initialize memory

GPU set up

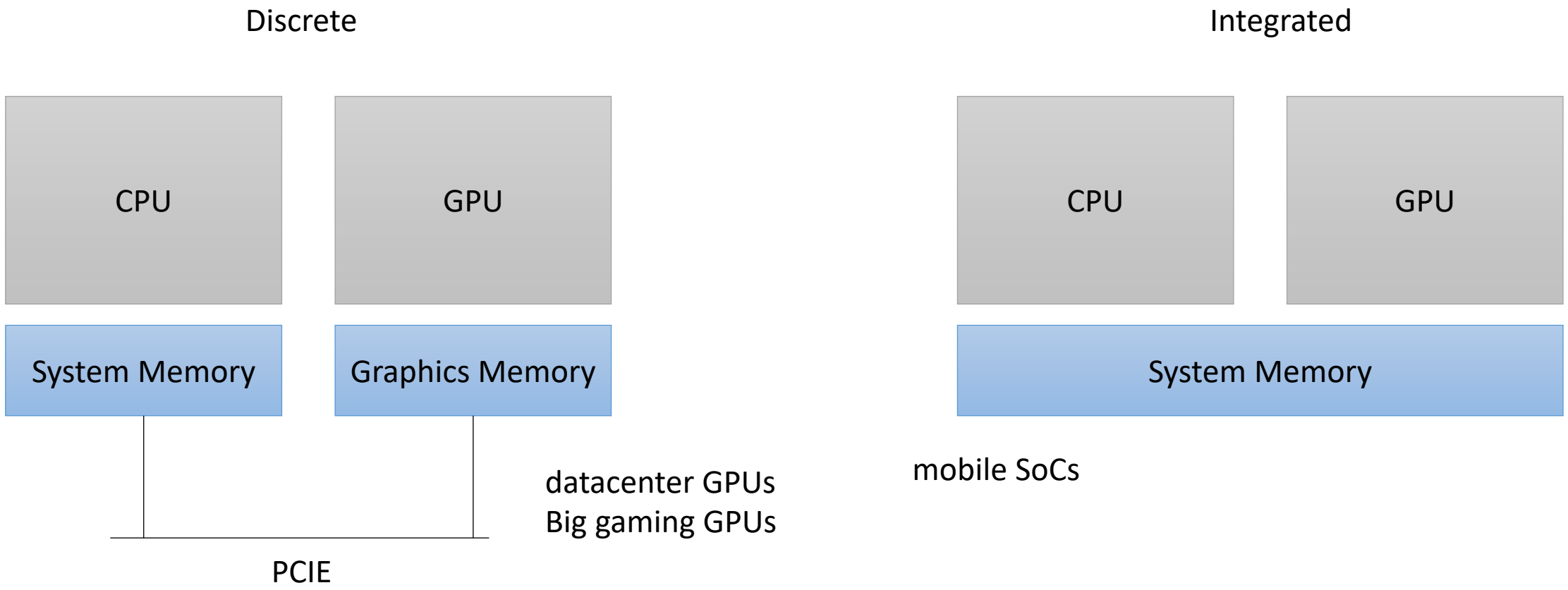
- GPUs come in two flavors



GPU set up

Pros and cons of each?

- GPUs come in two flavors

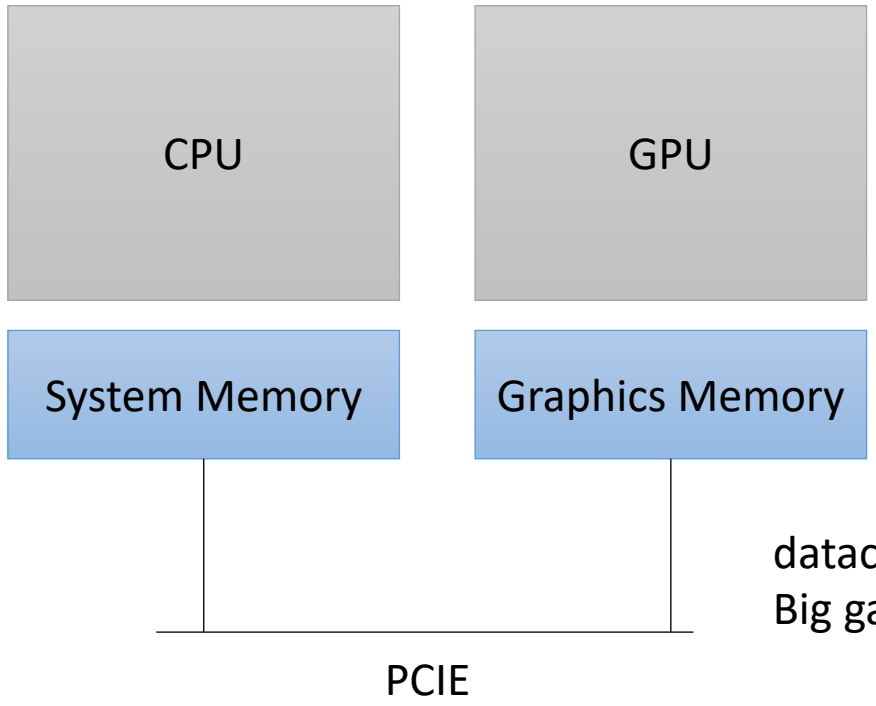


GPU set up

- GPUs come in two flavors

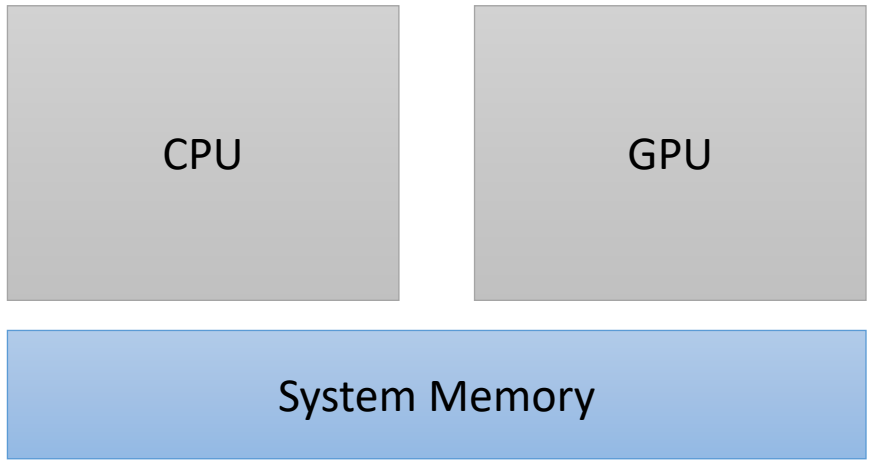
Pros and cons of each?
*Different types of memory for discrete
*Swappable for discrete
*More energy efficient for integrated
*Better memory utilization for integrated

Discrete



datacenter GPUs
Big gaming GPUs

Integrated



mobile SoCs

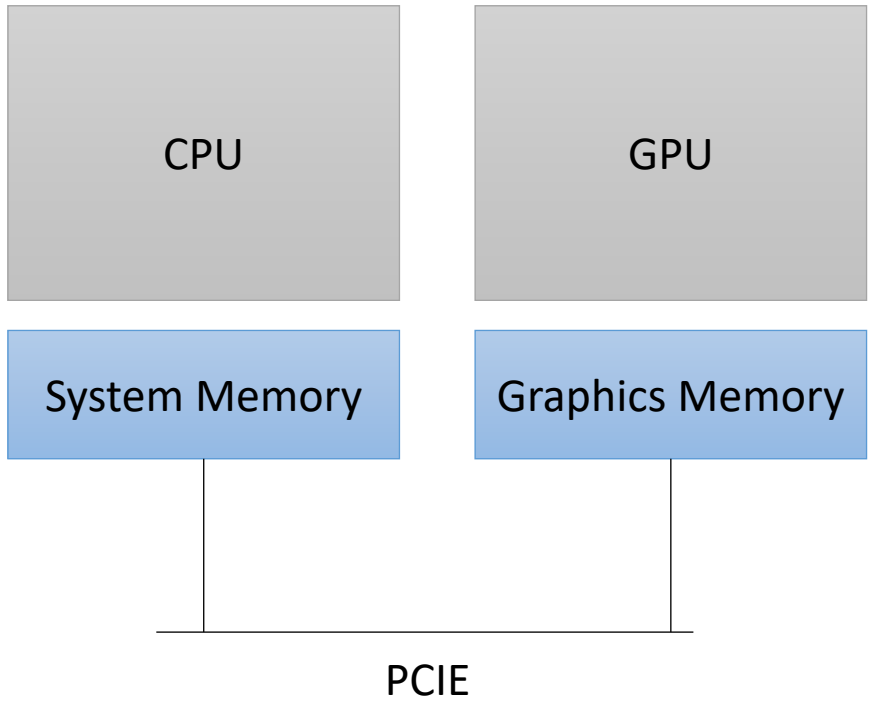
GPU set up

- GPUs come in two flavors

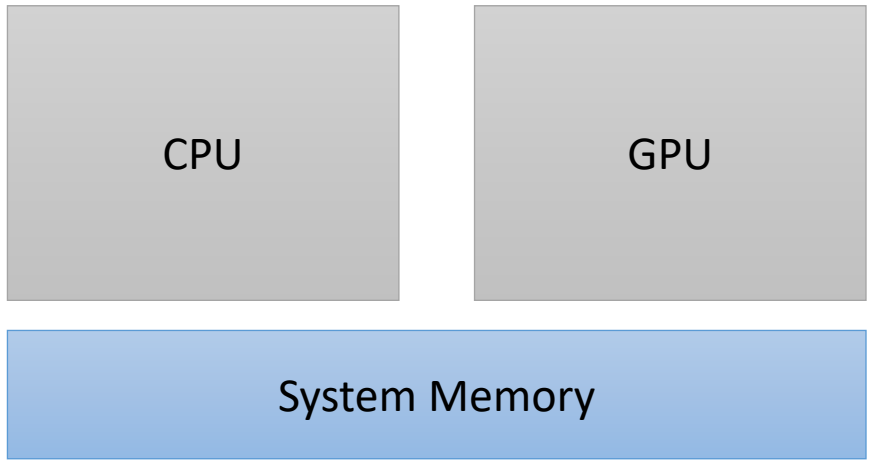
Although mobile GPUs share the system memory, Most still require you to program as if they didn't have shared memory.

Why?

Discrete



Integrated



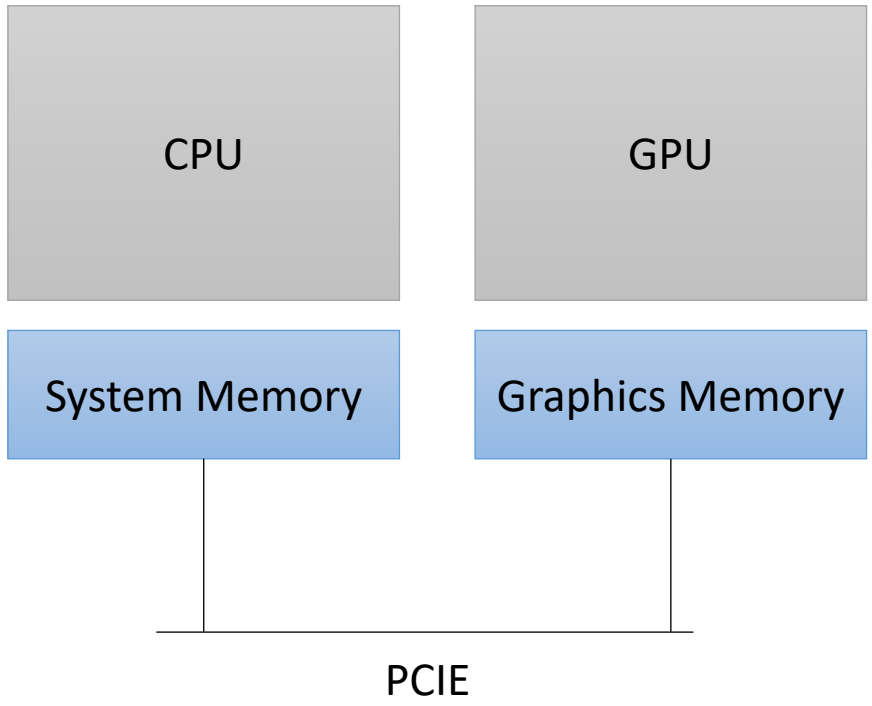
GPU set up

- GPUs come in two flavors

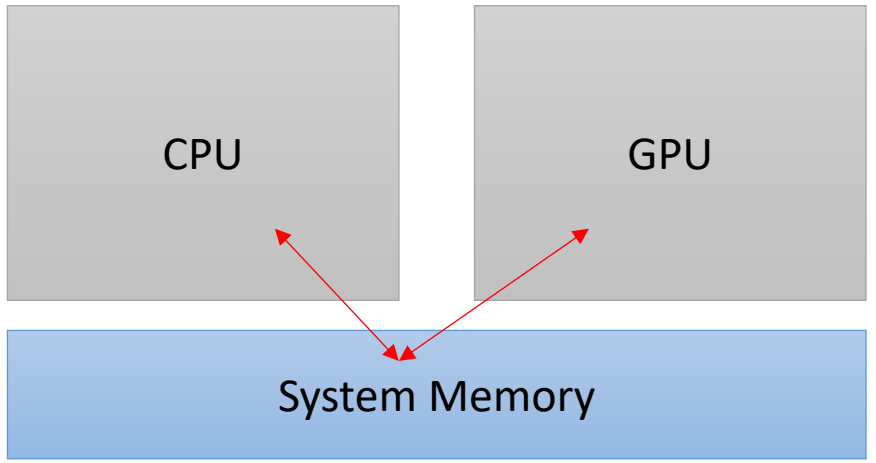
Although mobile GPUs share the system memory, Most still require you to program as if they didn't have shared memory.

Why?

Discrete



Integrated



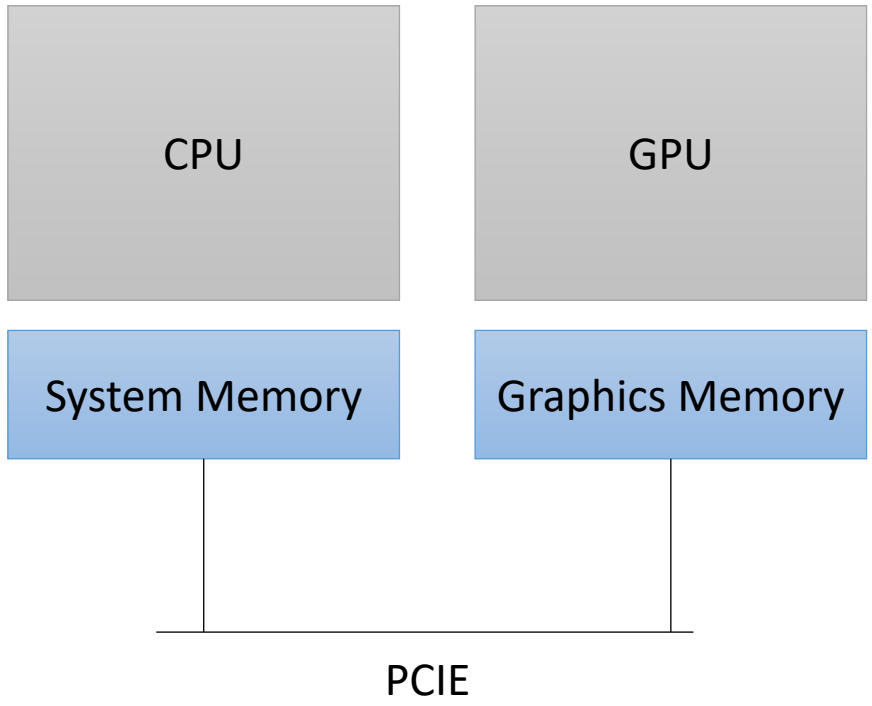
GPU set up

- GPUs come in two flavors

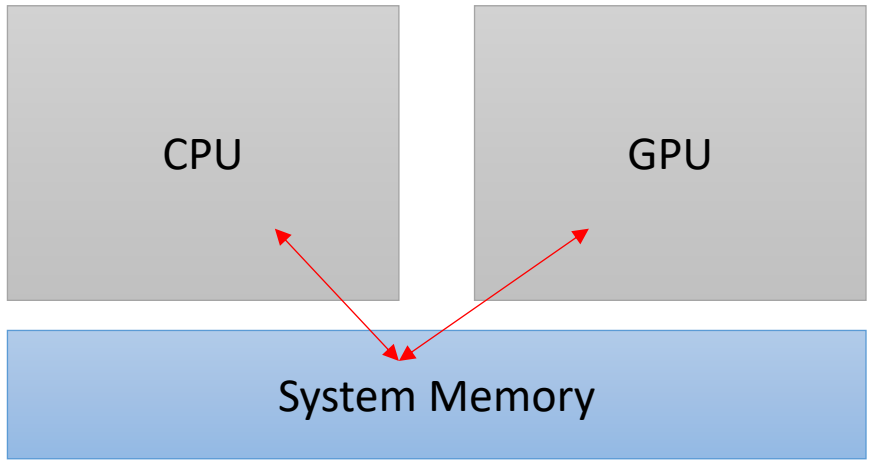
Although mobile GPUs share the system memory, Most still require you to program as if they didn't have shared memory.

Why?

Discrete



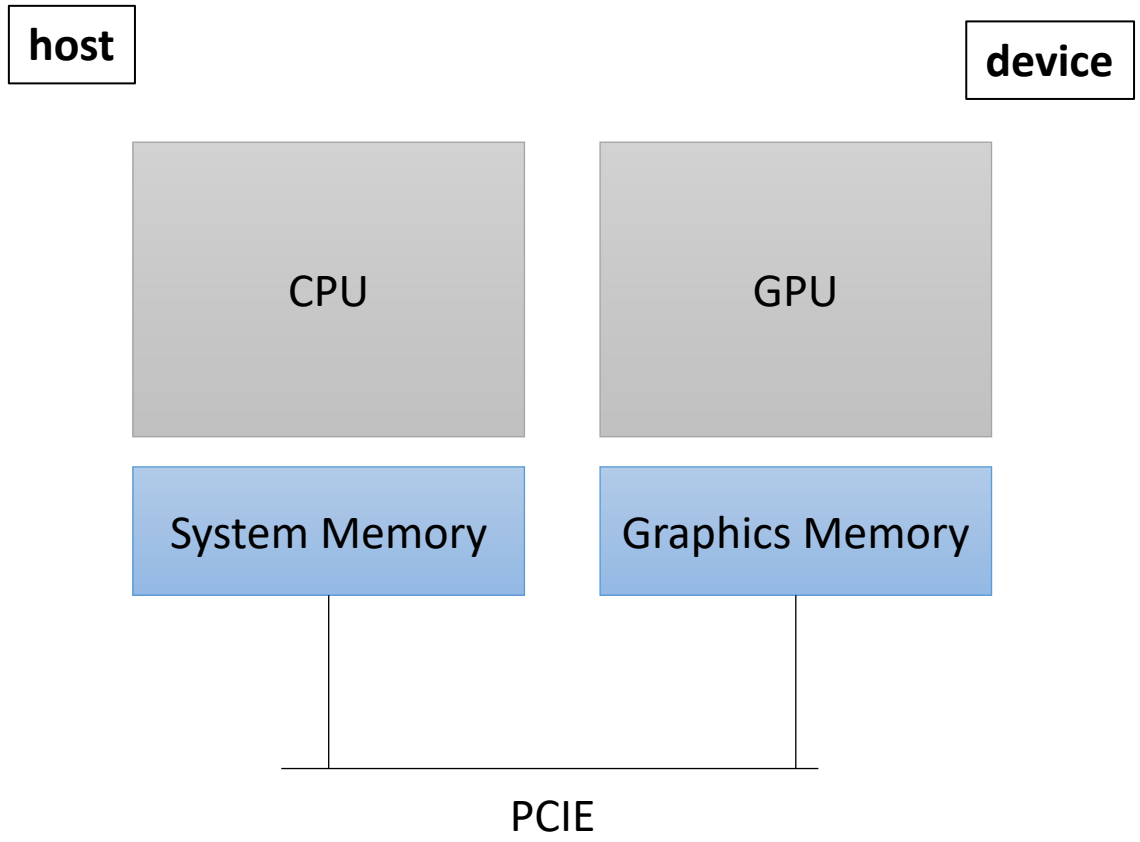
Integrated



In many cases, CPU-GPU communication is not fully supported coherence, fences, and RMWs might now be supported.

GPU set up

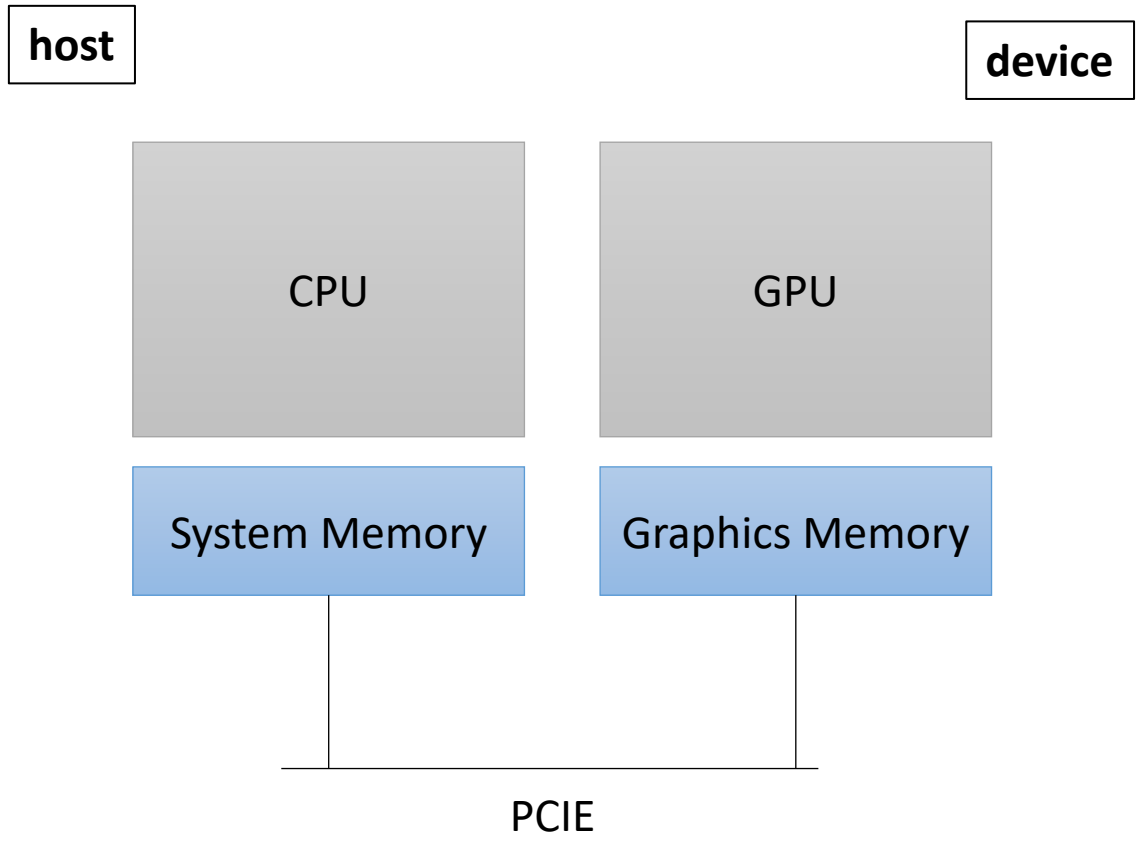
- Our heterogeneous, parallel, programming model



GPU set up

- Our heterogeneous, parallel, programming model

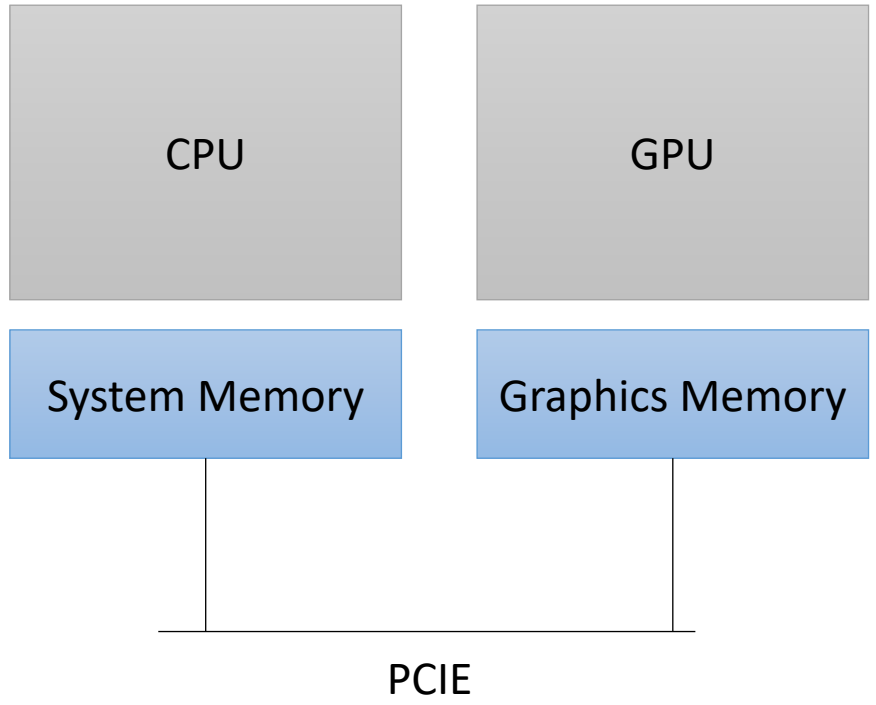
The host (CPU) will write a C++-like program that allocates and sets up memory on the GPU. The host will then call a GPU program called a kernel.



GPU set up

How do we allocate memory on a CPU?

- Our heterogeneous, parallel, programming model

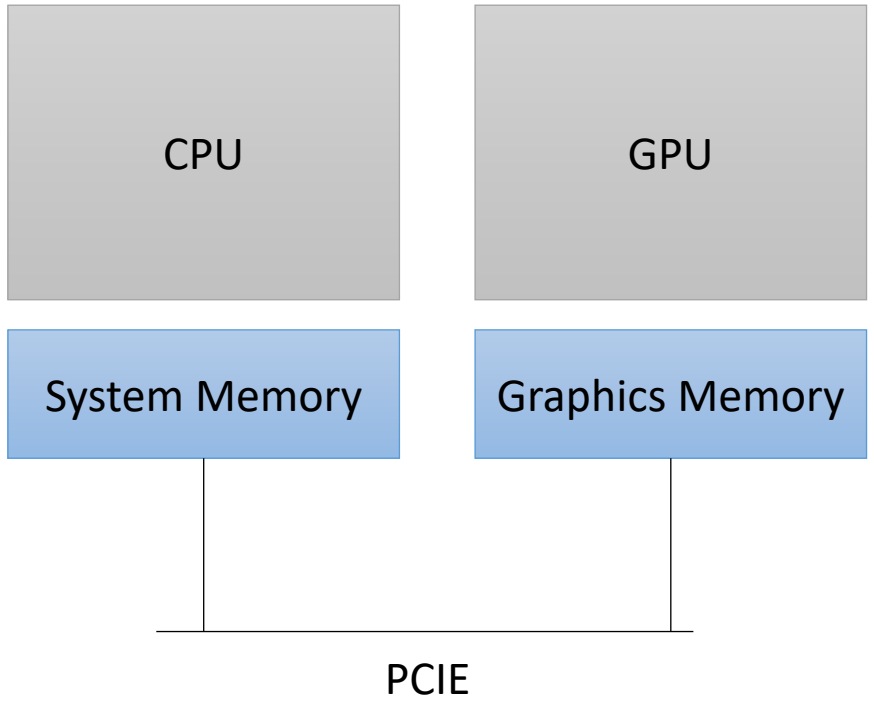


GPU set up

How do we allocate CPU memory on the host?

- Our heterogeneous, parallel, programming model

```
int *x = (int*) malloc(sizeof(int)*SIZE);
```

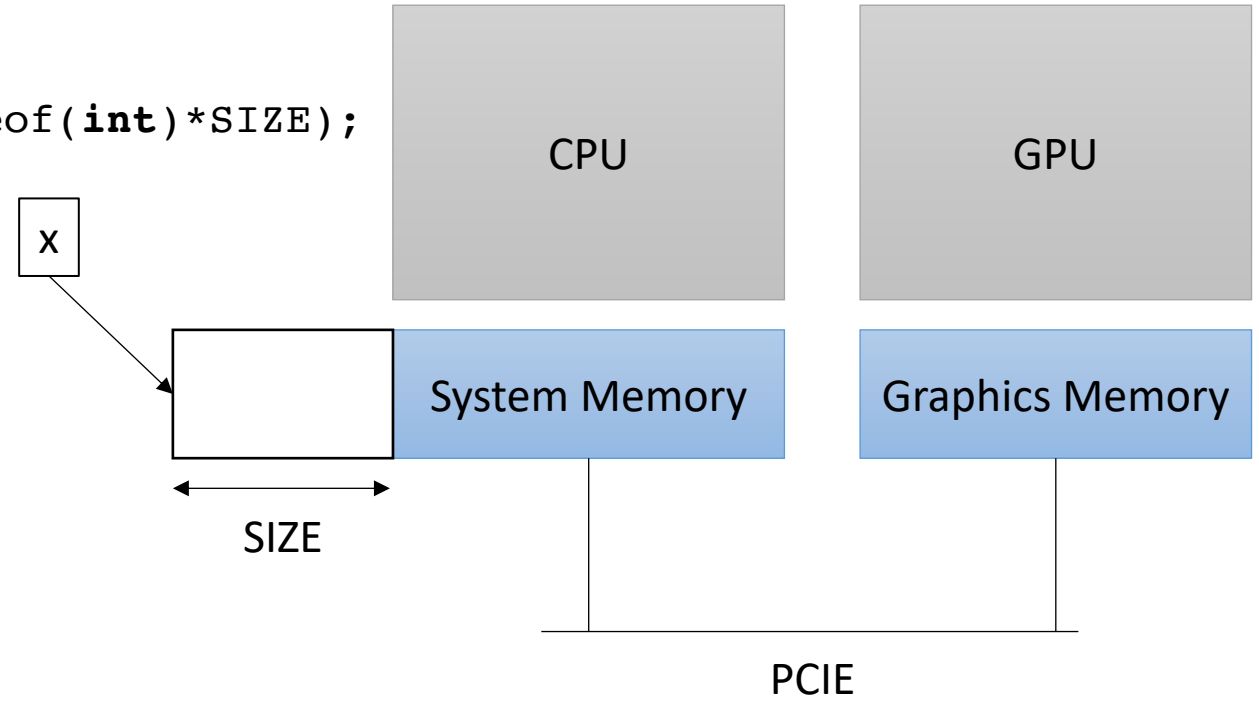


GPU set up

How do we allocate CPU memory on the host?

- Our heterogeneous, parallel, programming model

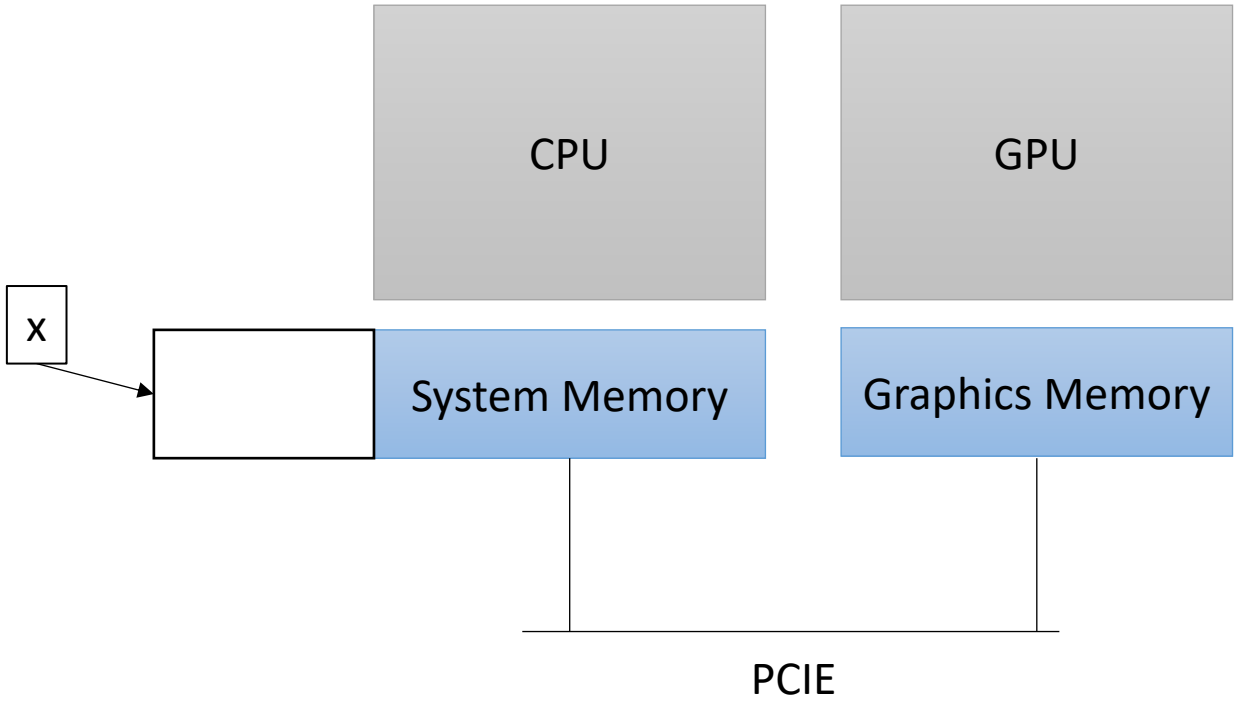
```
int *x = (int*) malloc(sizeof(int)*SIZE);
```



GPU set up

We need to allocate GPU memory on the host

- Our heterogeneous, parallel, programming model

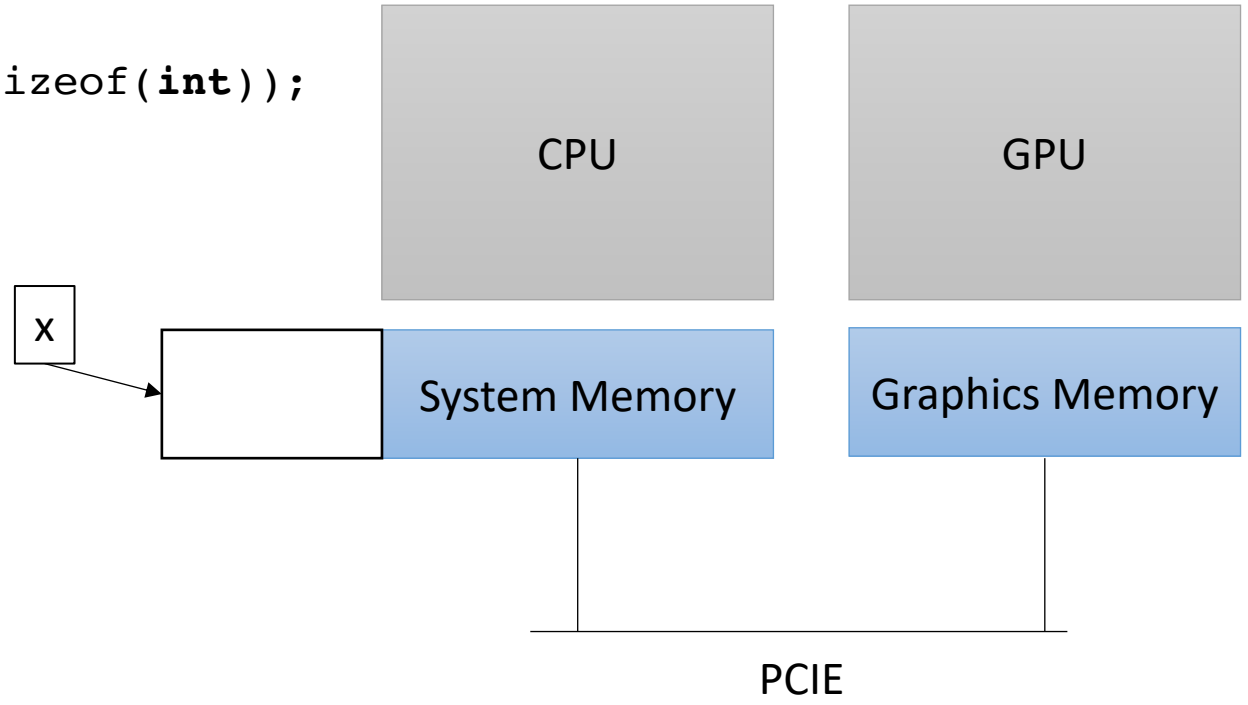


GPU set up

We need to allocate GPU memory on the host

- Our heterogeneous, parallel, programming model

```
int *d_x;
cudaMalloc(&d_x, SIZE*sizeof(int));
```

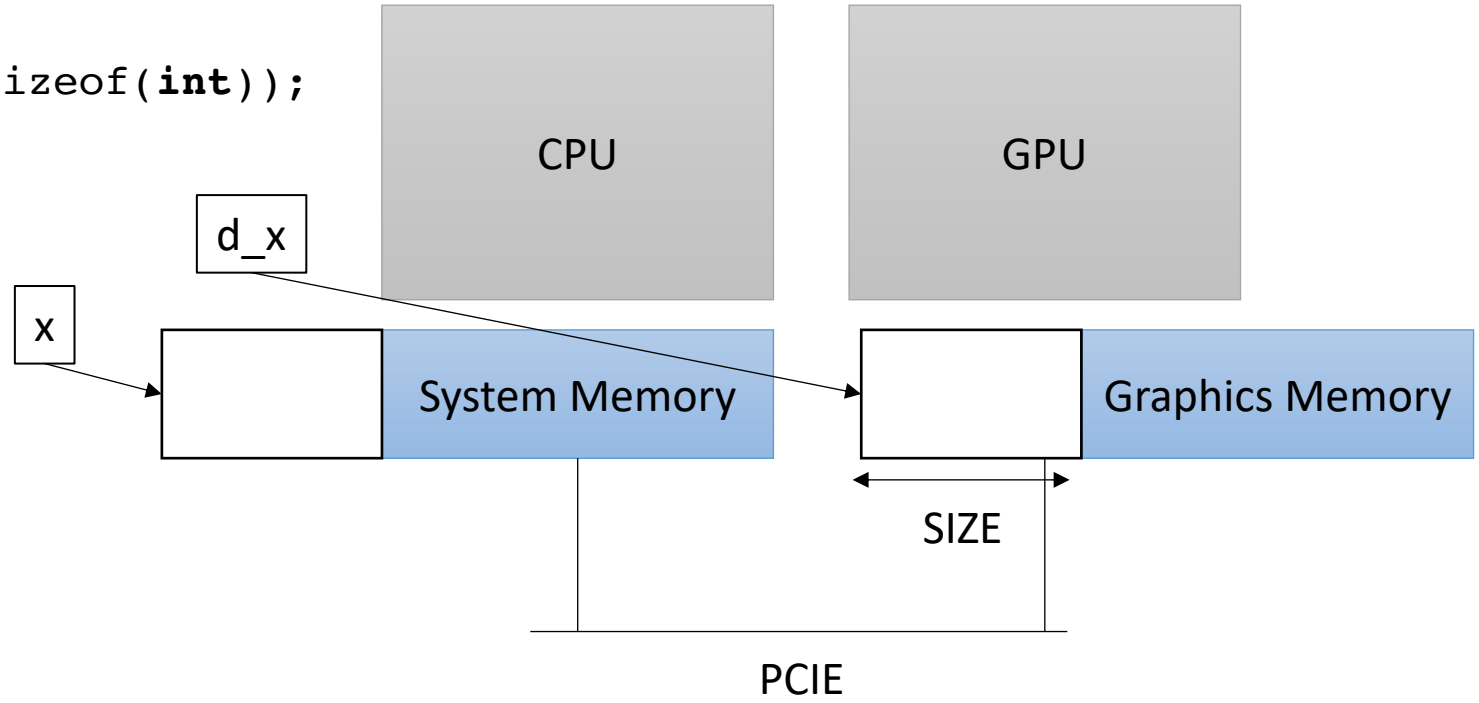


GPU set up

We need to allocate GPU memory on the host

- Our heterogeneous, parallel, programming model

```
int *d_x;
cudaMalloc(&d_x, SIZE*sizeof(int));
```

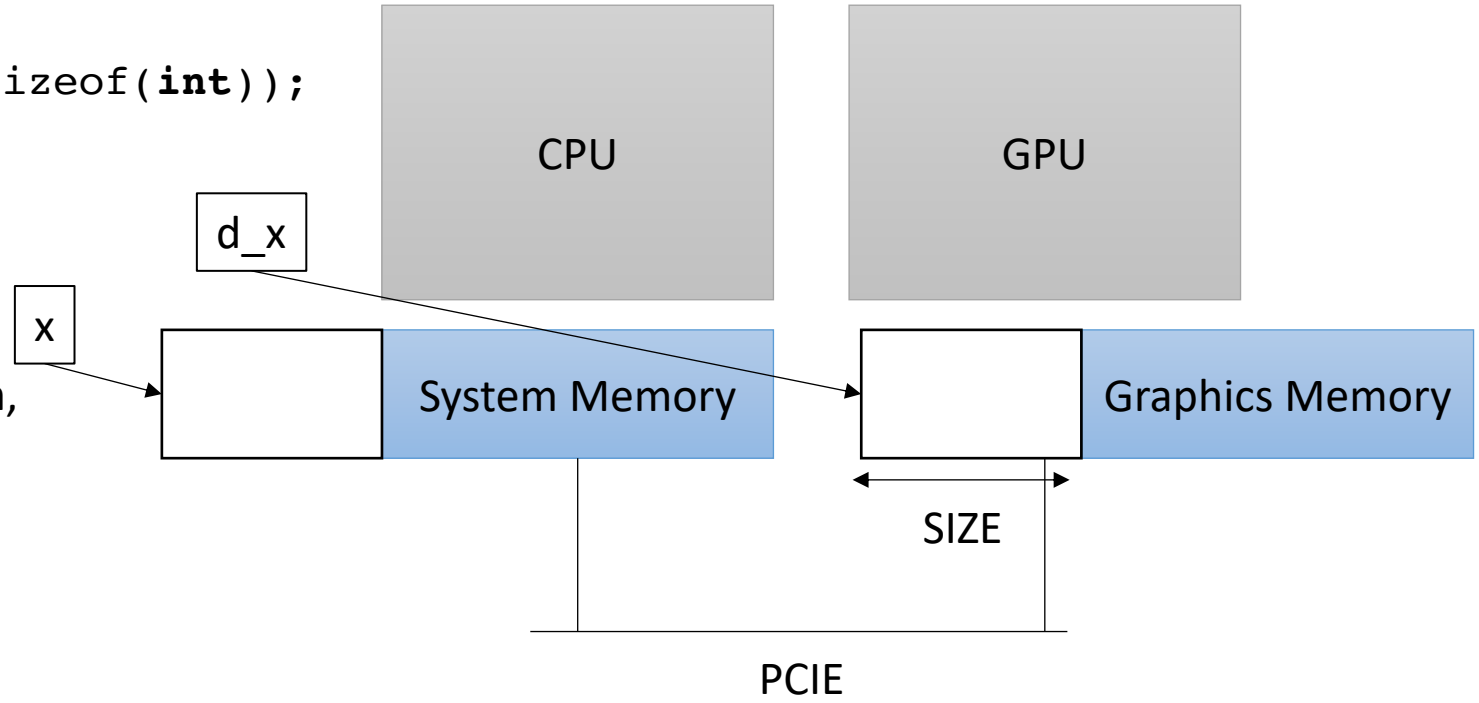


GPU set up

We need to allocate GPU memory on the host

- Our heterogeneous, parallel, programming model

```
int *d_x;
cudaMalloc(&d_x, SIZE*sizeof(int));
```

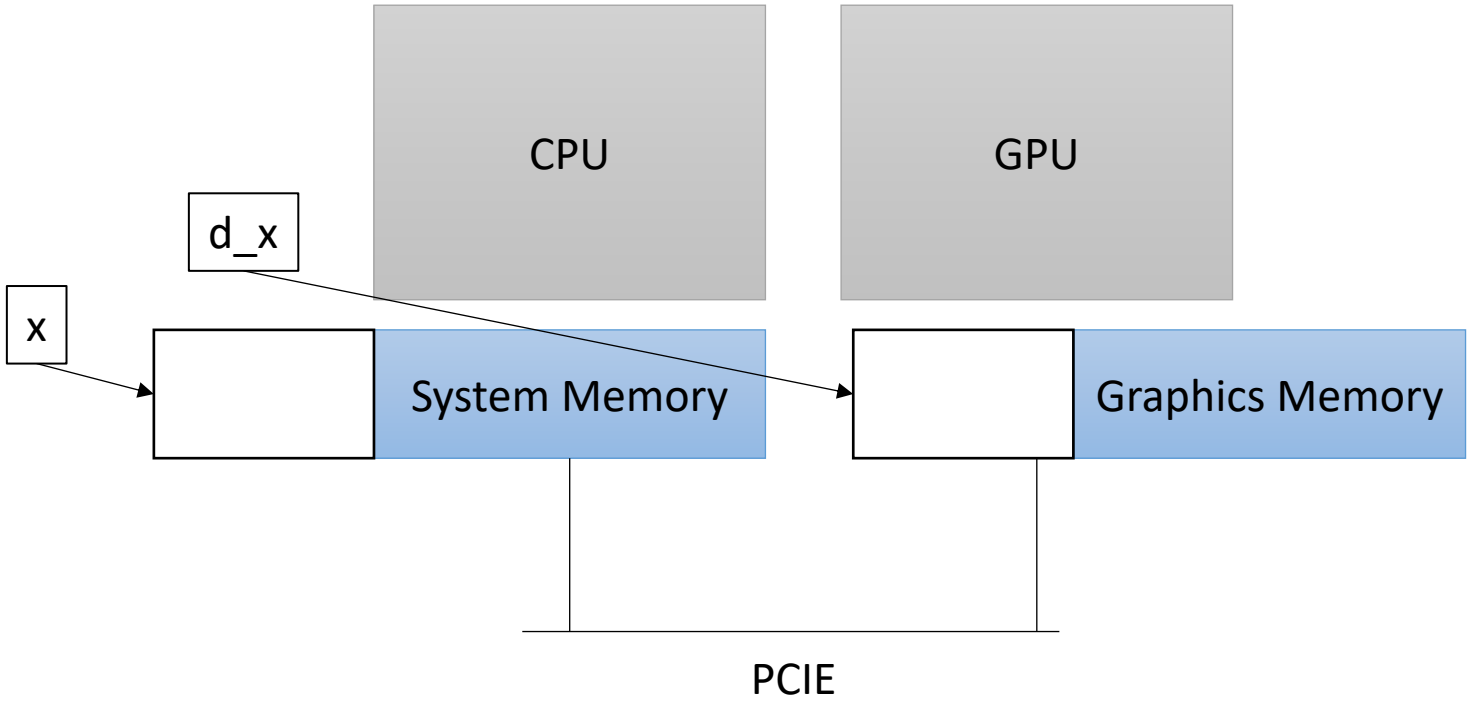


`d_x` is a pointer, in the CPU program, that points to memory on the GPU.

We can pass the pointer around, but the CPU cannot access the data
i.e. `d_x[0]` gives an error!

GPU set up

- Our heterogeneous, parallel, programming model

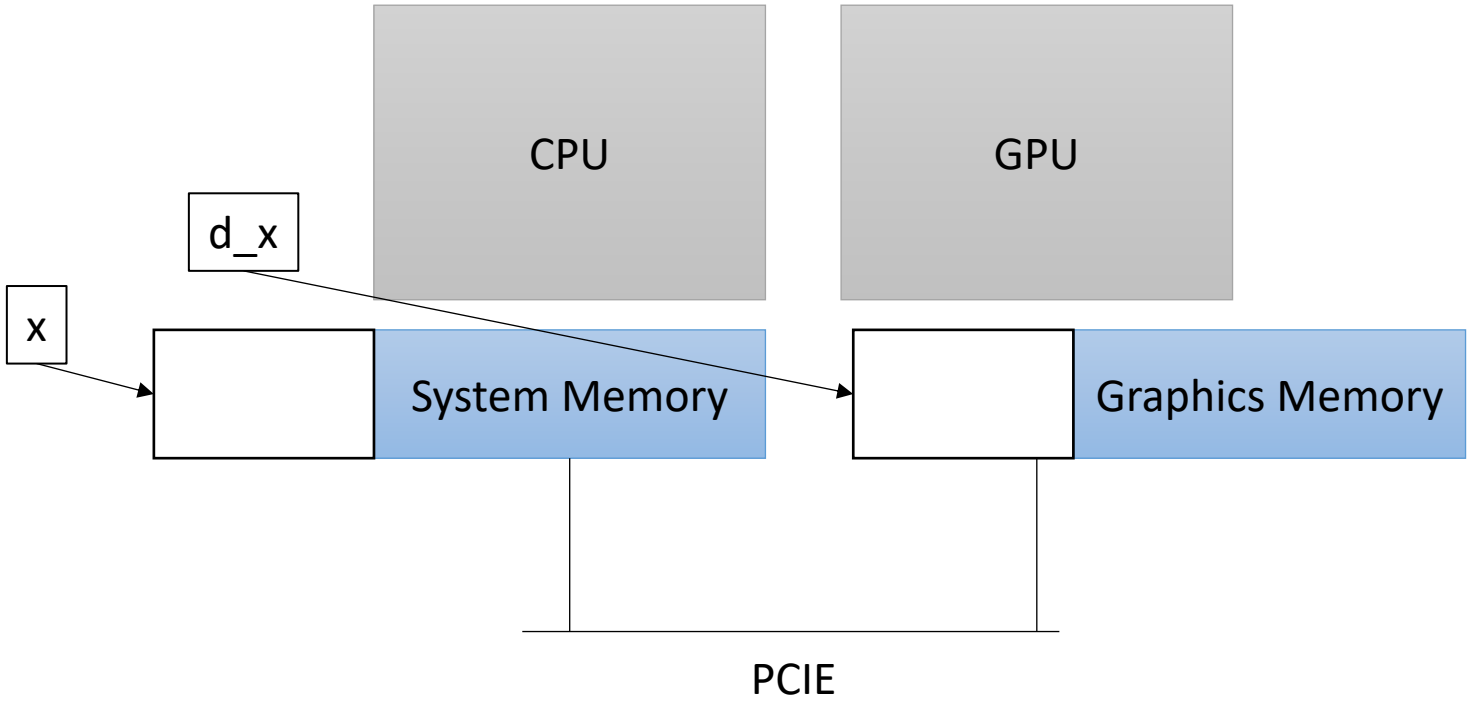


GPU set up

- Our heterogeneous, parallel, programming model

If we can't access d_x on the CPU, how do we initialize the memory?

GPU has no access to input devices e.g. disk



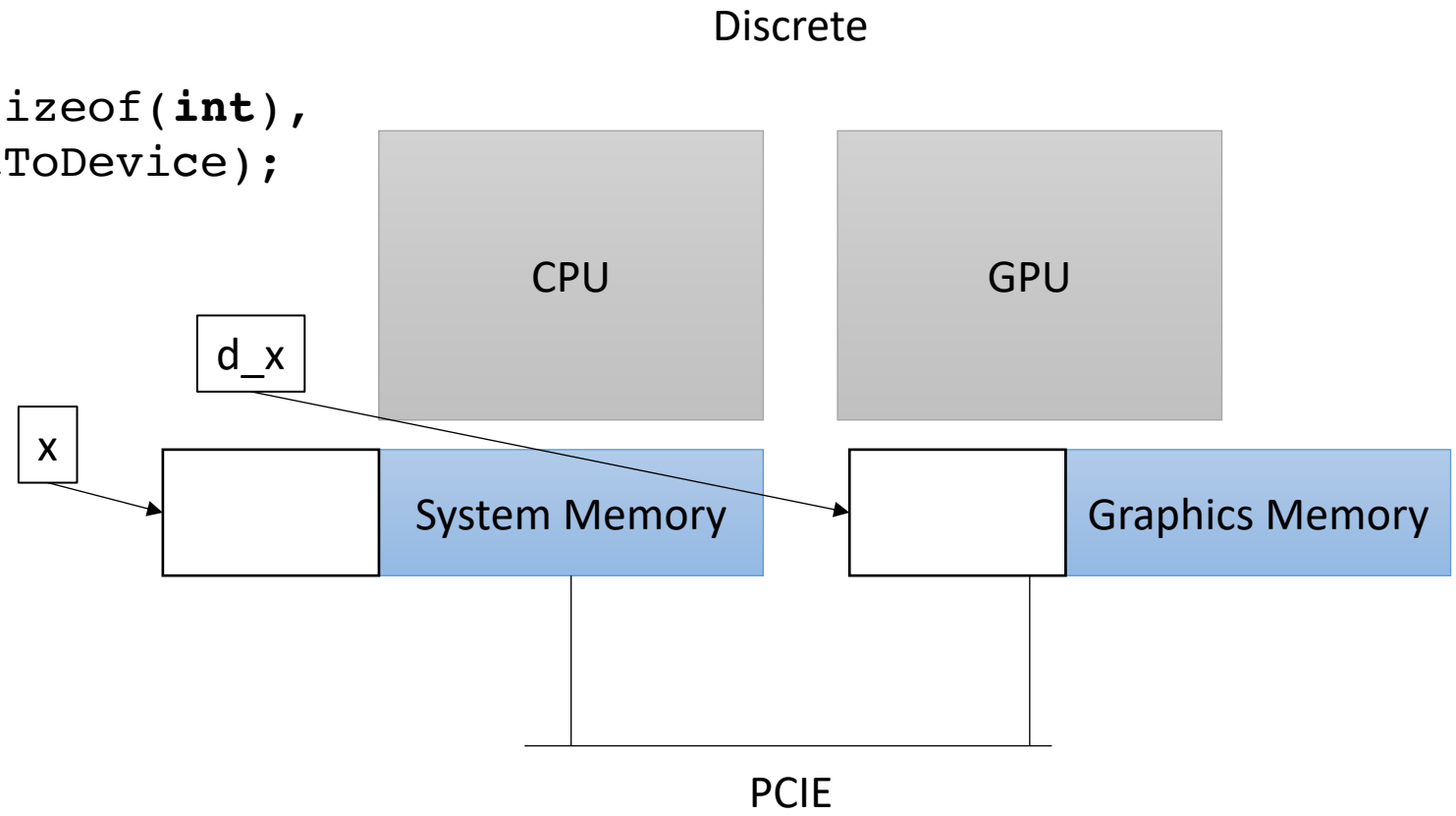
GPU set up

- Our heterogeneous, parallel, programming model

If we can't access d_x on the CPU, how do we initialize the memory?

GPU has no access to input devices e.g. disk

```
//initialize x on host
cudaMemcpy(d_x, x, SIZE*sizeof(int),
           cudaMemcpyHostToDevice);
```



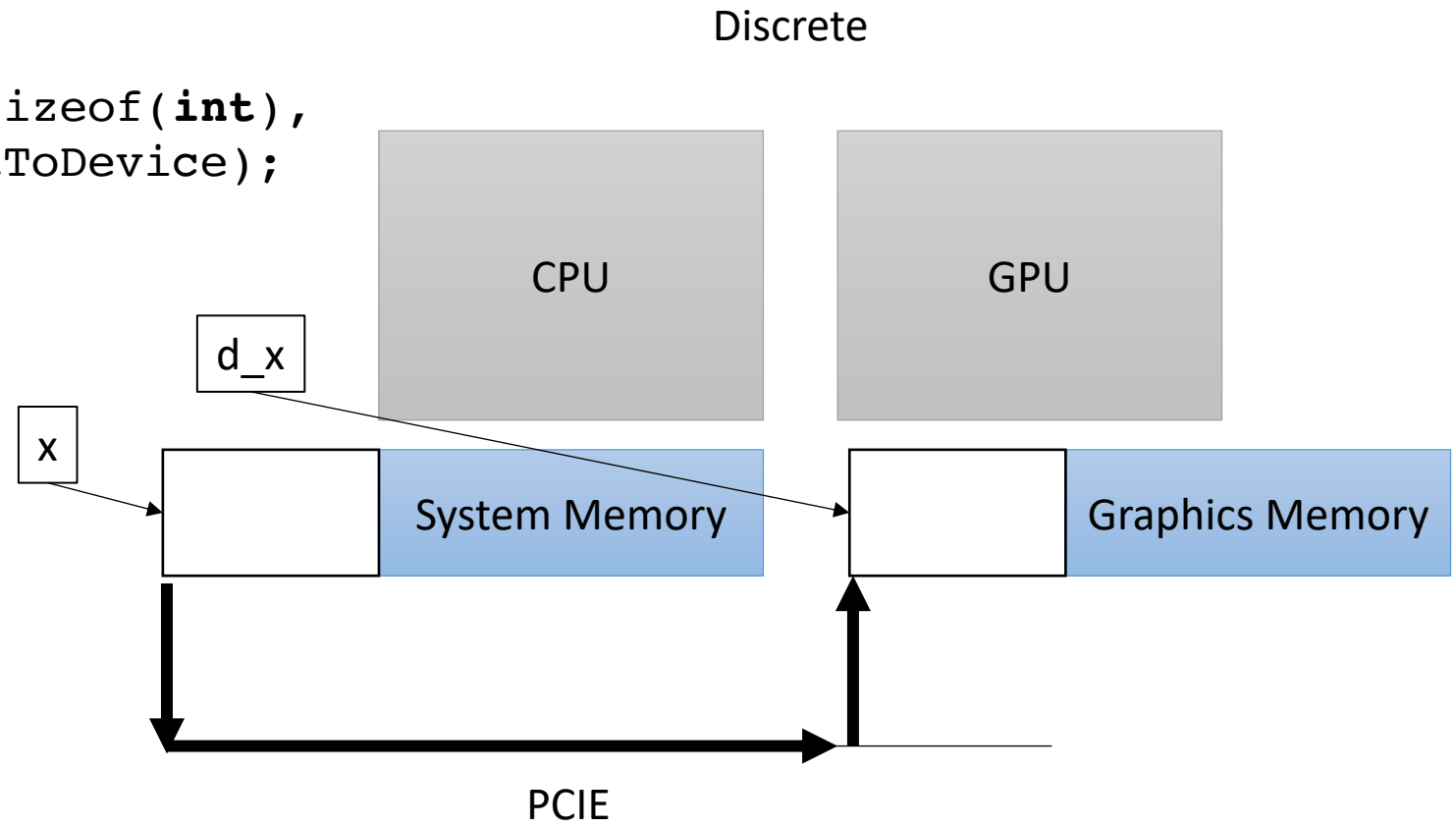
GPU set up

- Our heterogeneous, parallel, programming model

If we can't access d_x on the CPU, how do we initialize the memory?

GPU has no access to input devices e.g. disk

```
//initialize x on host
cudaMemcpy(d_x, x, SIZE*sizeof(int),
           cudaMemcpyHostToDevice);
```



How does this look in code?

How does this look in code?

Nothing too exciting yet.

The GPU Program

- Write a special function in your C++ code.
  - Called a Kernel
  - Use the new keyword `__global__`
  - Keywords in
    - OpenCL `__kernel`
    - Metal `kernel`

- Write it how you'd write any other function

The GPU Program

```
__global__ void vector_add(int * a, int * b, int * c, int size) {
   for (int i = 0; i < size; i++) {
     a[i] = b[i] + c[i];
  }
}
```

The GPU Program

```
__global__ void vector_add(int * a, int * b, int * c, int size) {
    for (int i = 0; i < size; i++) {
      a[i] = b[i] + c[i];
   }
}
```

calling the function

```
vector_add<<<1,1>>>(d_a, d_b, d_c, size);
```

The GPU Program

```
__global__ void vector_add(int * a, int * b, int * c, int size) {
    for (int i = 0; i < size; i++) {
      a[i] = b[i] + c[i];
   }
}
```

What in the world?
special new CUDA syntax. We will talk more soon

calling the function

```
vector_add<<<1,1>>>(d_a, d_b, d_c, size);
```

The GPU Program

```
__global__ void vector_add(int * d_a, int * d_b, int * d_c, int size) {
    for (int i = 0; i < size; i++) {
      d_a[i] = d_b[i] + d_c[i];
   }
}
```

Pass in pointers to memory on the device

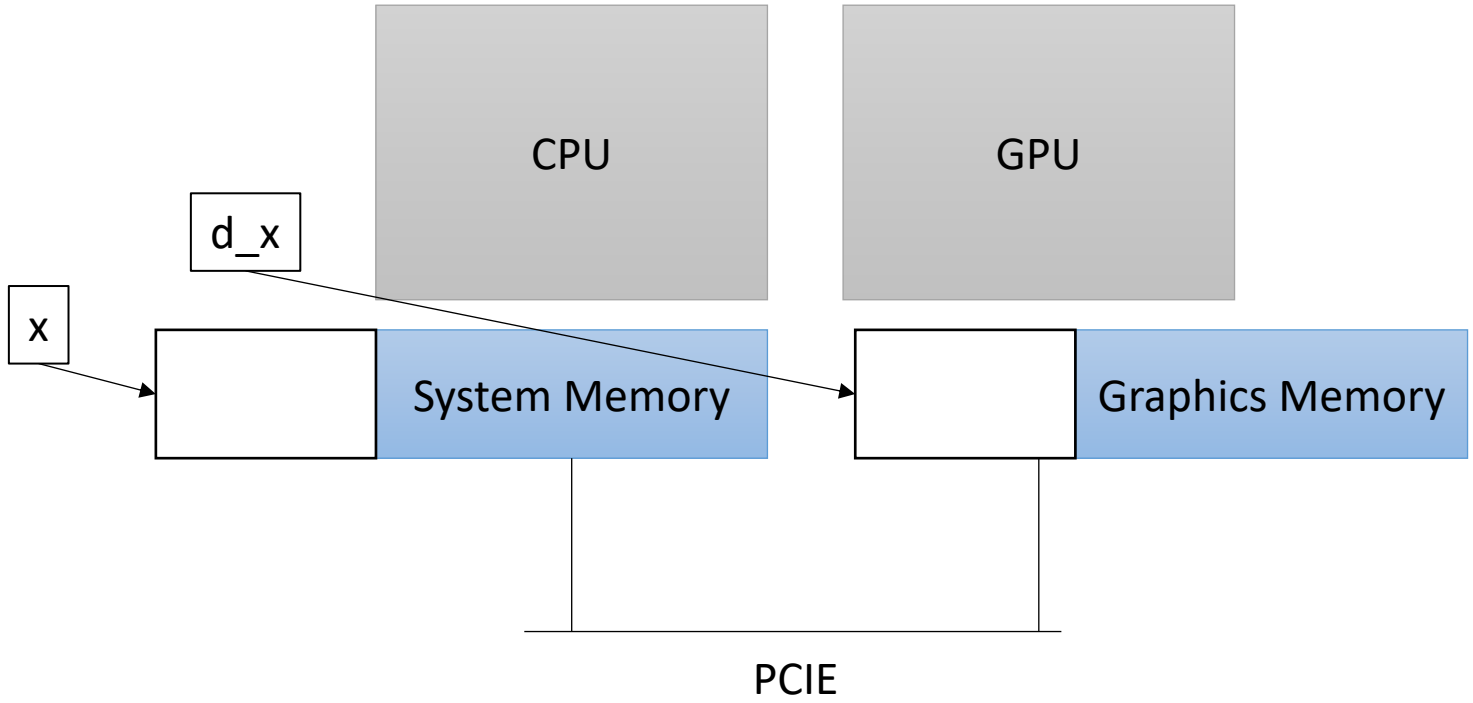
calling the function

```
vector_add<<<1,1>>>(d_a, d_b, d_c, size);
```

GPU set up

- Our heterogeneous, parallel, programming model

Remember, GPU needs to access its own memory



The GPU Program

```
__global__ void vector_add(int * d_a, int * d_b, int * d_c, int size) {
    for (int i = 0; i < size; i++) {
      d_a[i] = d_b[i] + d_c[i];
   }
}
```

Constants can be passed in regularly

calling the function

```
vector_add<<<1,1>>>(d_a, d_b, d_c, size);
```

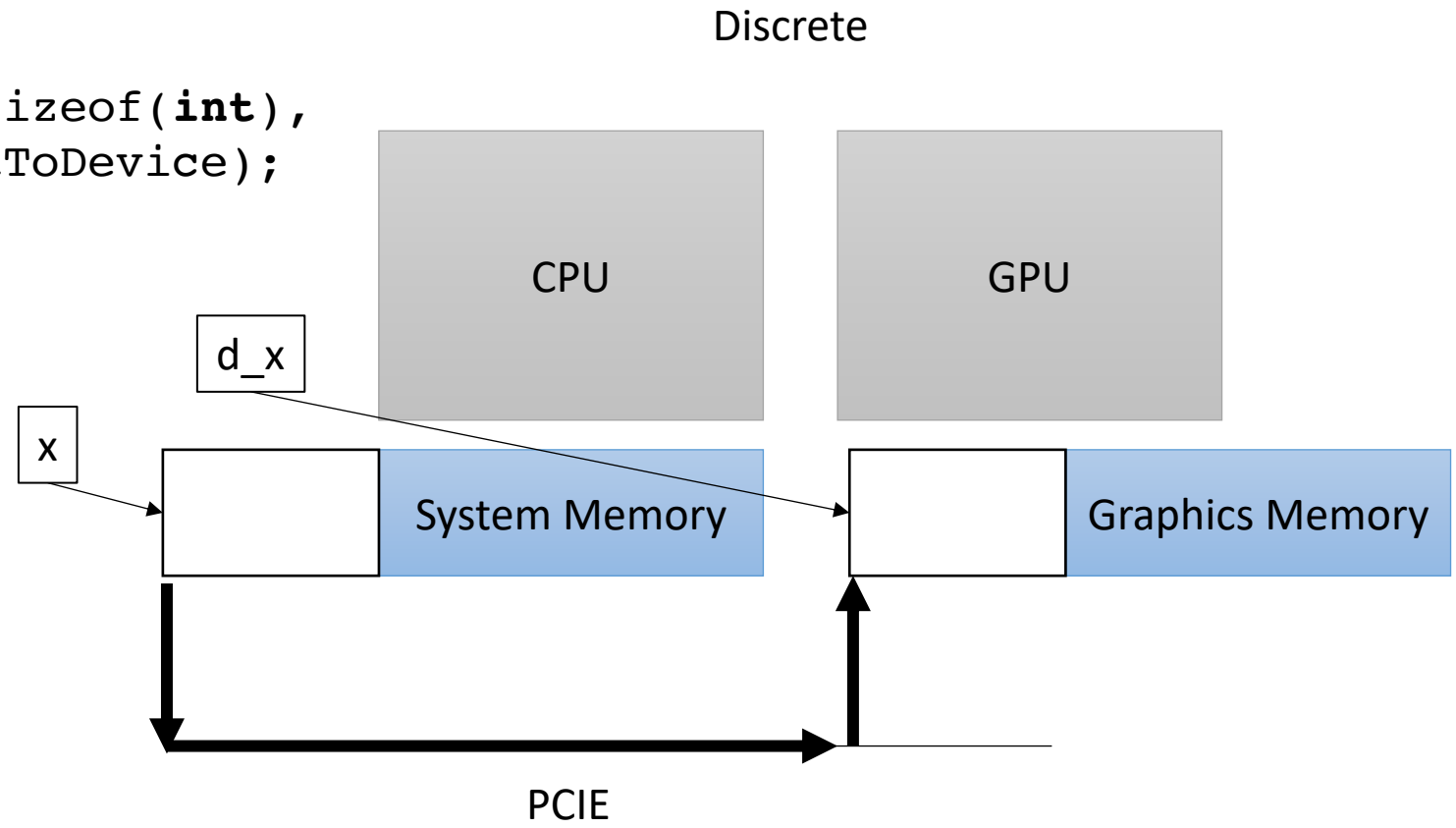
The GPU Program

Are we ready to run the program? What are we missing?

GPU set up

- Our heterogeneous, parallel, programming model

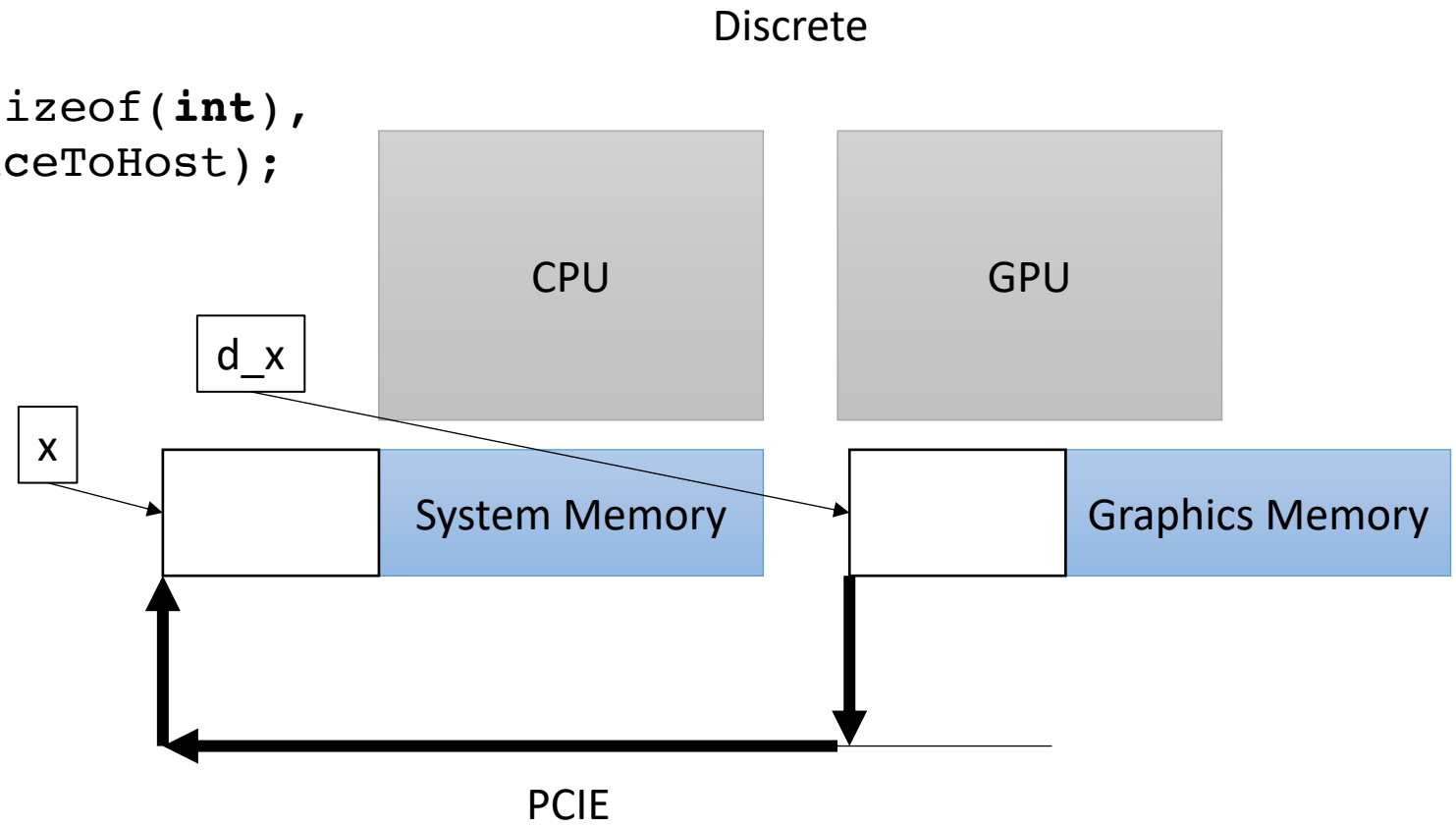
```
//initialize x on host
cudaMemcpy(d_x, x, SIZE*sizeof(int),
           cudaMemcpyHostToDevice);
```



GPU set up

- Our heterogeneous, parallel, programming model

```
//initialize x on host
cudaMemcpy(x, d_x, SIZE*sizeof(int),
           cudaMemcpyDeviceToHost);
```



The GPU Program

Finally, we can run the GPU program!

Lets see what all the hype is about

The GPU Program

It didn’t do so well...

Next lecture:

- Optimizing the GPU program!

See you on Monday!

- Homework 4 due on today

- Homework 5 assigned my midnight today