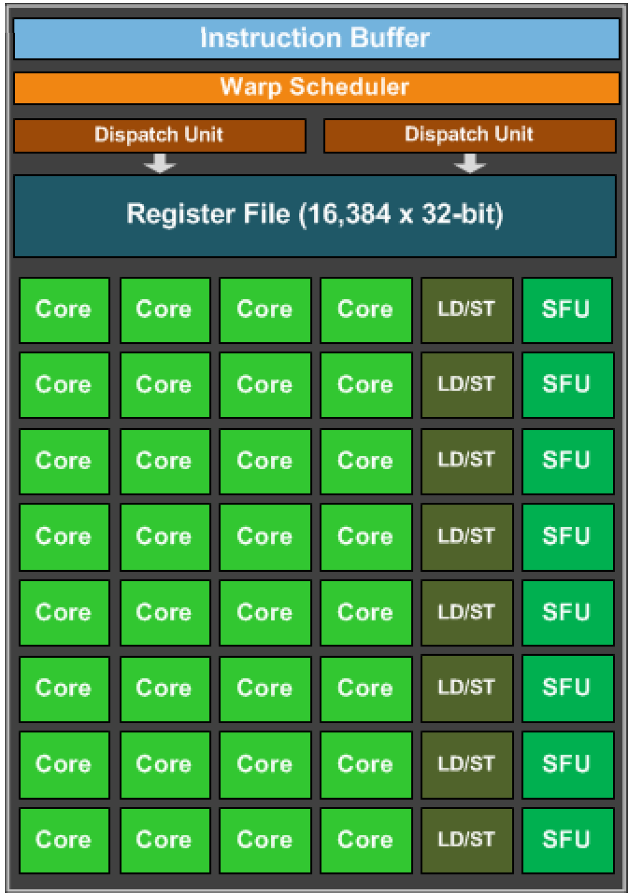
CSE113: Parallel Programming
March 10, 2023

- **Topics**:
  - Intro to GPUs



Announcements

- HW 2 grades are out, let us know if there are any issues
  - Especially let us know if there are issues with throughput
- Work on Homework 4 (Due today, you have until March 14)
- Planning on HW 5 to be released today
- Last module of the class!

Previous Quiz

Previous Quiz

In terms of memory models, the compiler needs to ensure the following property:

- ○ Any weak behavior allowed in the language is also allowed in the ISA
- ○ Any weak behaviors that are disallowed in the language need to be disallowed in the ISA
- ○ The compilation ensures that the program has sequentially consistent behavior at the ISA level
- ○ The compiler does not need to reason about relaxed memory

Previous Quiz

The C++ relaxed memory order provides

- ○ no orderings at all
- ○ orderings only between accesses of the same address
- ○ TSO memory behaviors when run on an x86 system
- ○ an easy way to accidentally introduce horrible bugs into your program

Previous Quiz

A program that uses mutexes and has no data conflicts does not have weak memory behaviors for which of the following reasons?

- ○ Mutexes prevent memory accesses from happening close enough in time for weak behaviors to occur
- ○ The OS has built in support for Mutexes that disable architecture features, such as the store buffer
- ○ A correct mutex implementation uses fences in lock and unlock to disallow weak behaviors

Previous Quiz

Assuming you had a sequentially consistent processor, any C/++ program you ran on it would also be sequentially consistent, regardless of if there are data-conflicts or not.

- ○ True
- ○ False

Previous Quiz

If you put a fence after every memory instruction, would that be sufficient to disallow all weak behaviors on a weak architecture? Please write a few sentences explaining your answer.

Review: Progress and schedules

Liveness property

- Something good will eventually happen

- Examples:
  - The mutex program *will eventually terminate*
  - The self driving car *will eventually reach its destination*

- More difficult to reason about that safety properties

Scheduler specifications

- What is a scheduler specification?
  - A programming guide should give you a scheduler specification
  - As a programmer, you need to make sure that your program is safe to run under the scheduler
  - This is similar to the memory model, however, there are no “fences” in the scheduler.
  - For example mutexes can starve under the system scheduler, then you simply can’t use mutexes on that system.
  - C++ let’s you query the threading library to see what scheduler they support.

Schedulers

- A fair scheduler typically requires preemption

Schedulers

- A fair scheduler typically requires preemption

Schedulers

- A fair scheduler typically requires preemption

Schedulers

- A fair scheduler typically requires preemption

OS does a good job giving all threads a chance

The fair scheduler

- every thread that has not terminated will “eventually” get a chance to execute.
  - “concurrent forward progress”: defined by C++
    not guaranteed, but encouraged (and likely what you will observe)
  - “weakly fair scheduler”: defined by classic concurrency textbooks

Schedulers

- A fair scheduler typically requires preemption

Schedulers

- A fair scheduler typically requires preemption

peak into a thread object:

Thread 1:
program data
local variables

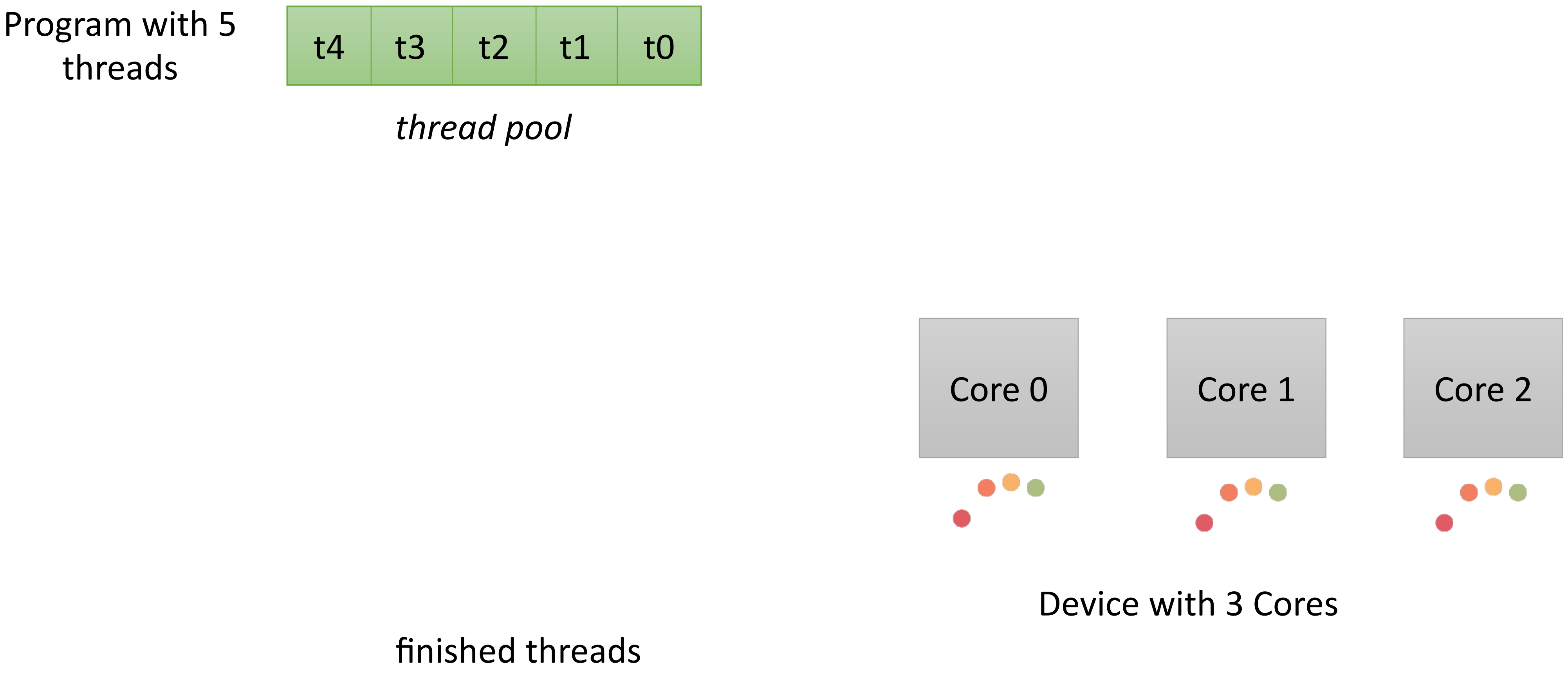
Estimated to be ~30K cycles to context switch between threads

Parallel Forward Progress

- “Any thread that has executed at least 1 instruction, is guaranteed to continue to be fairly executed”

- Also called:
  - “Parallel Forward Progress”: by C++
  - “Persistent Thread Model”: by GPU programmers
  - ”Occupancy Bound Execution Model”: in some of my papers

simplified execution model



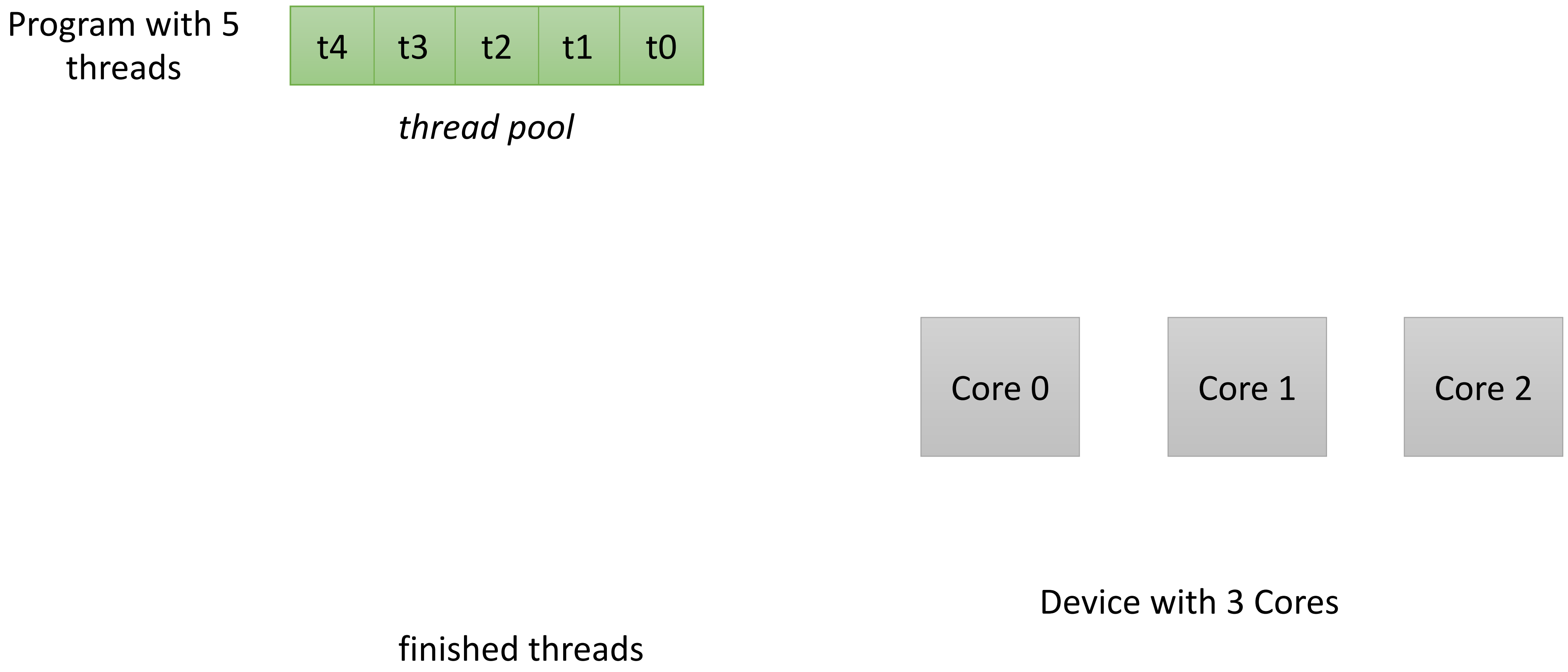
The HSA scheduler (power saving scheduler)

- The thread with the lowest ID that hasn’t terminated is guaranteed to eventually be executed.

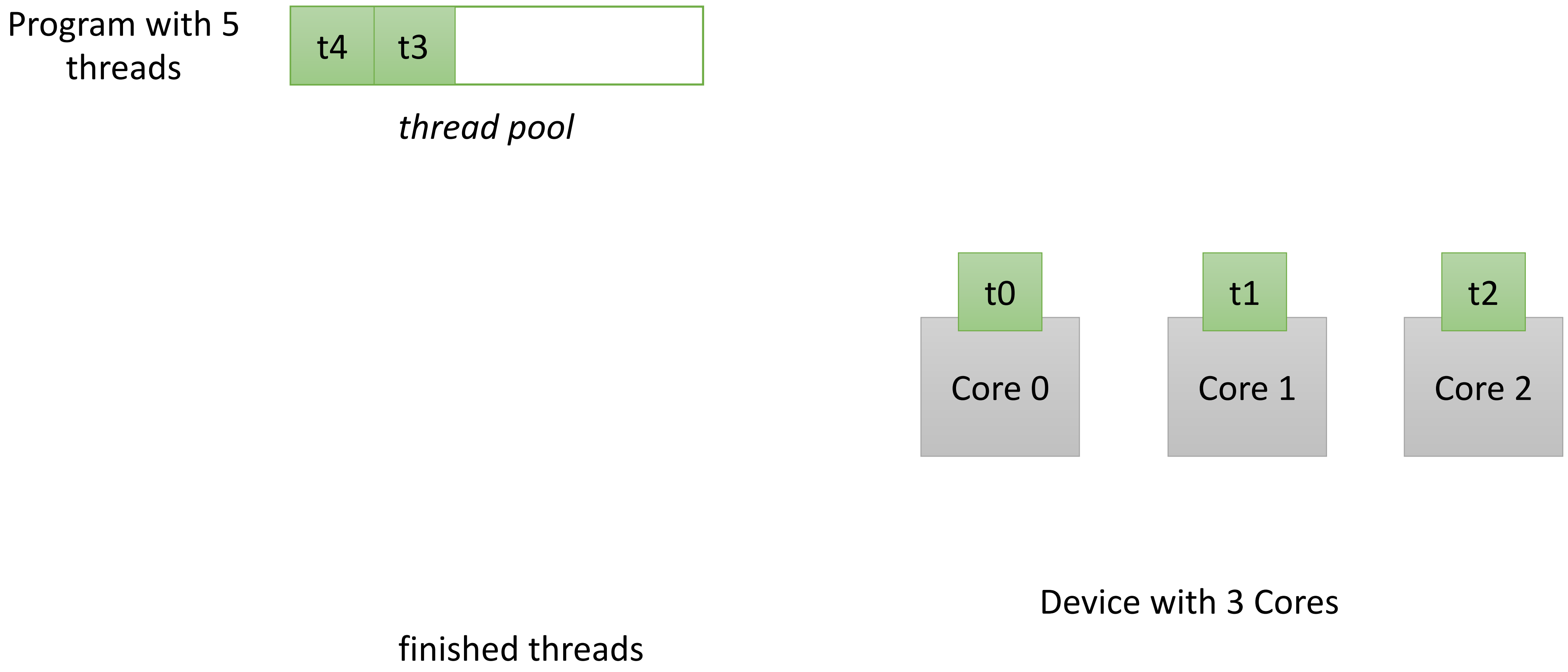
- Called:
  - “HSA” - Heterogeneous System Architecture, programming language proposed by AMD for new systems.

  - The HSA language appears to be defunct now, but the scheduler is a good fit for mobile devices (esp. mobile GPUs).

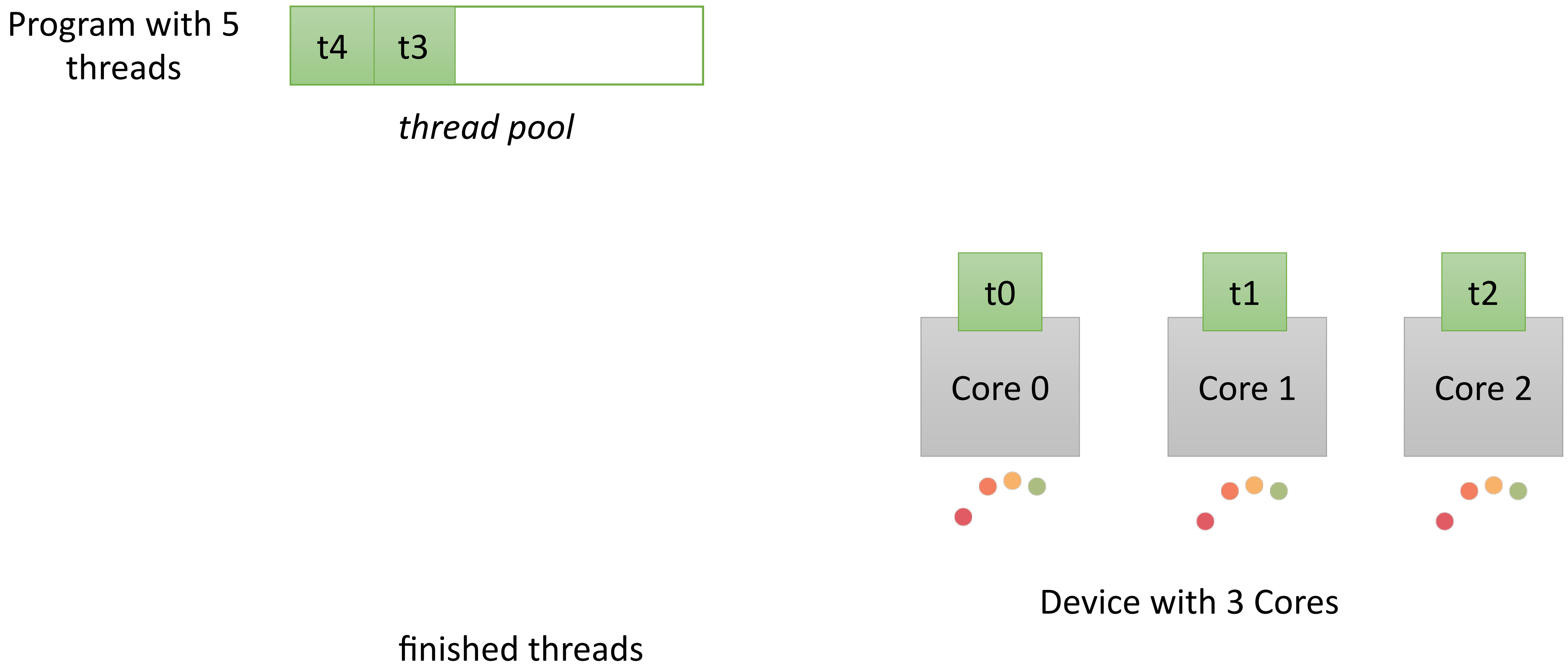
A power-saving scheduler



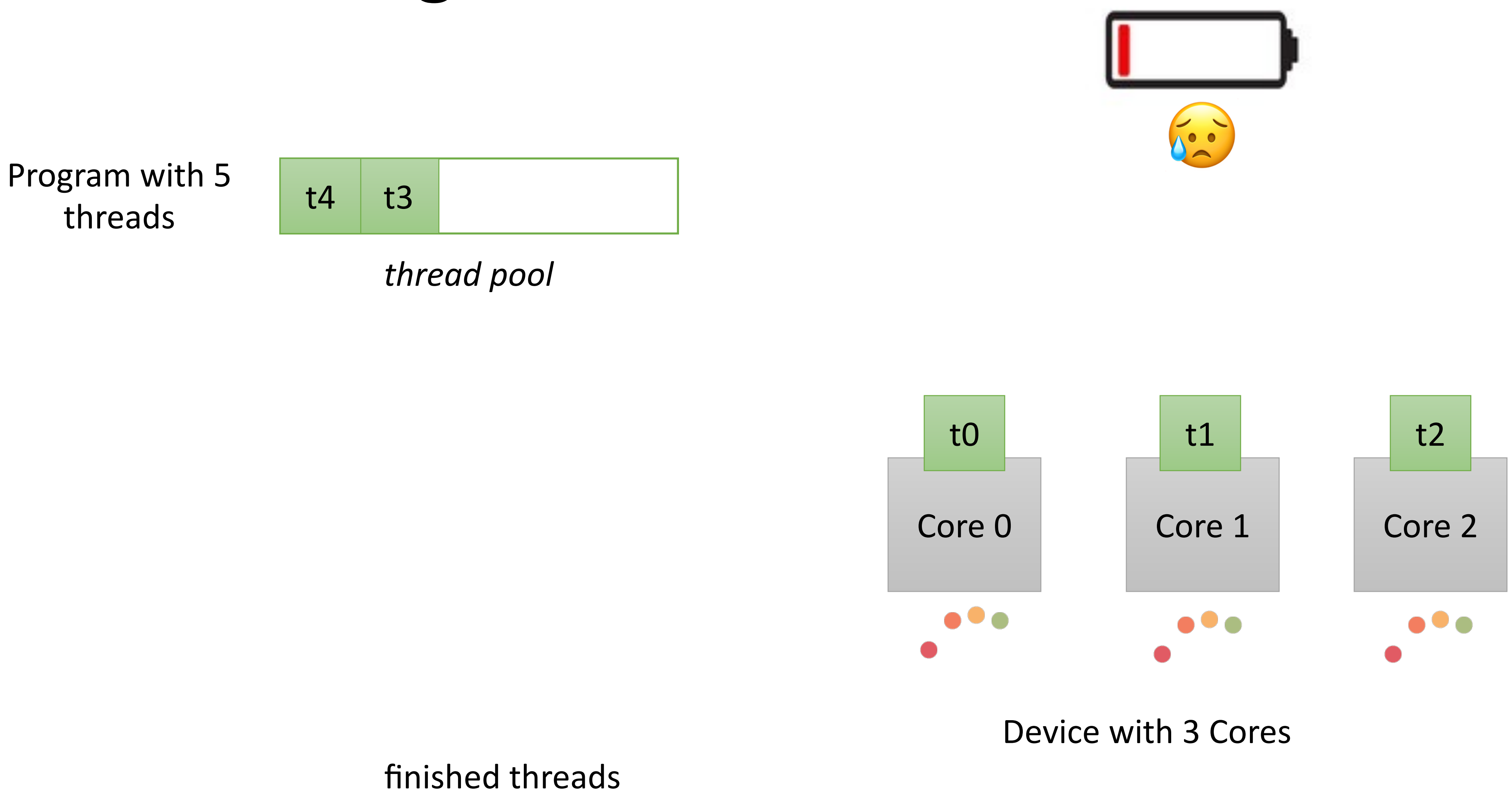
A power-saving scheduler



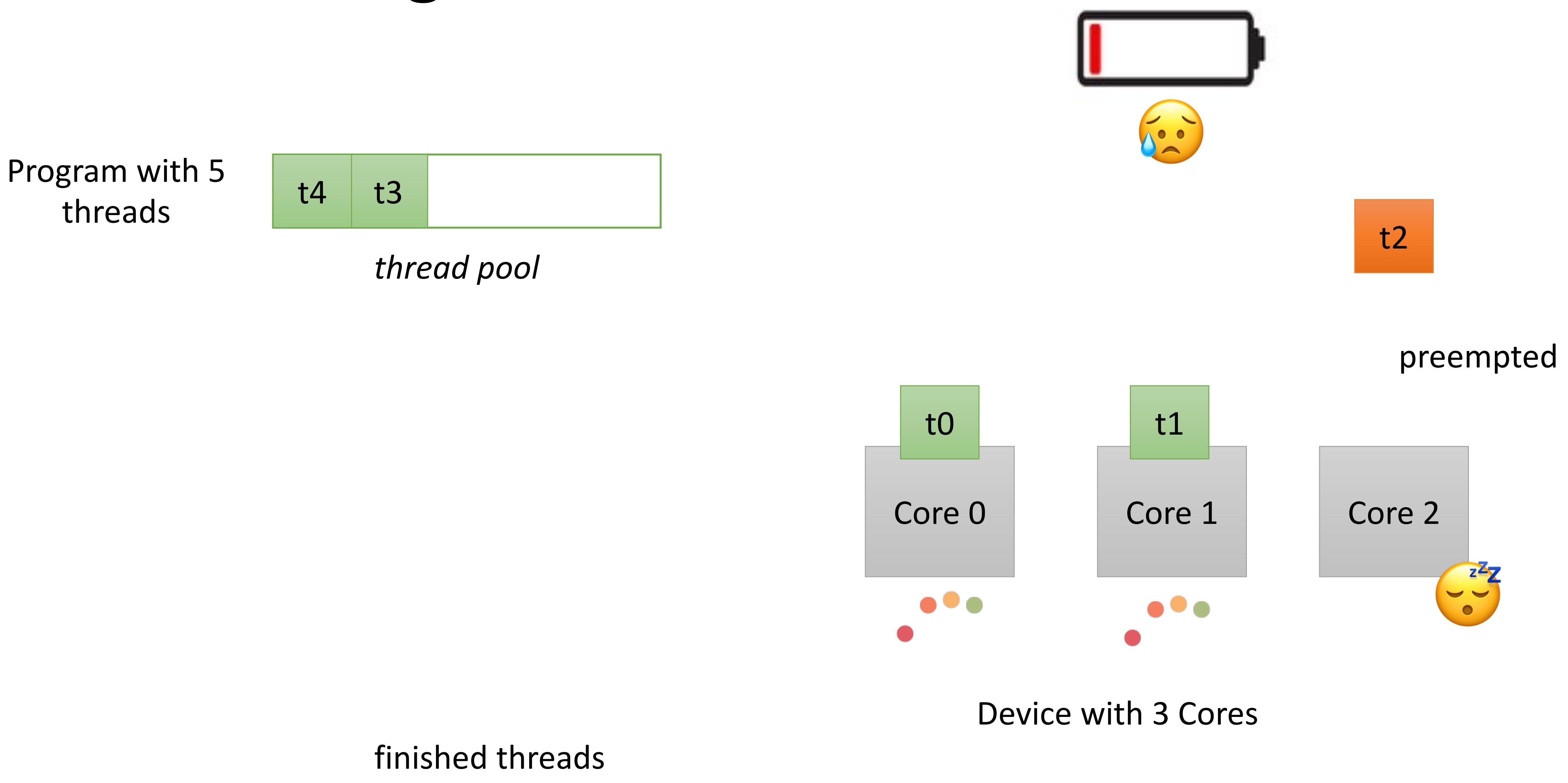
A power-saving scheduler



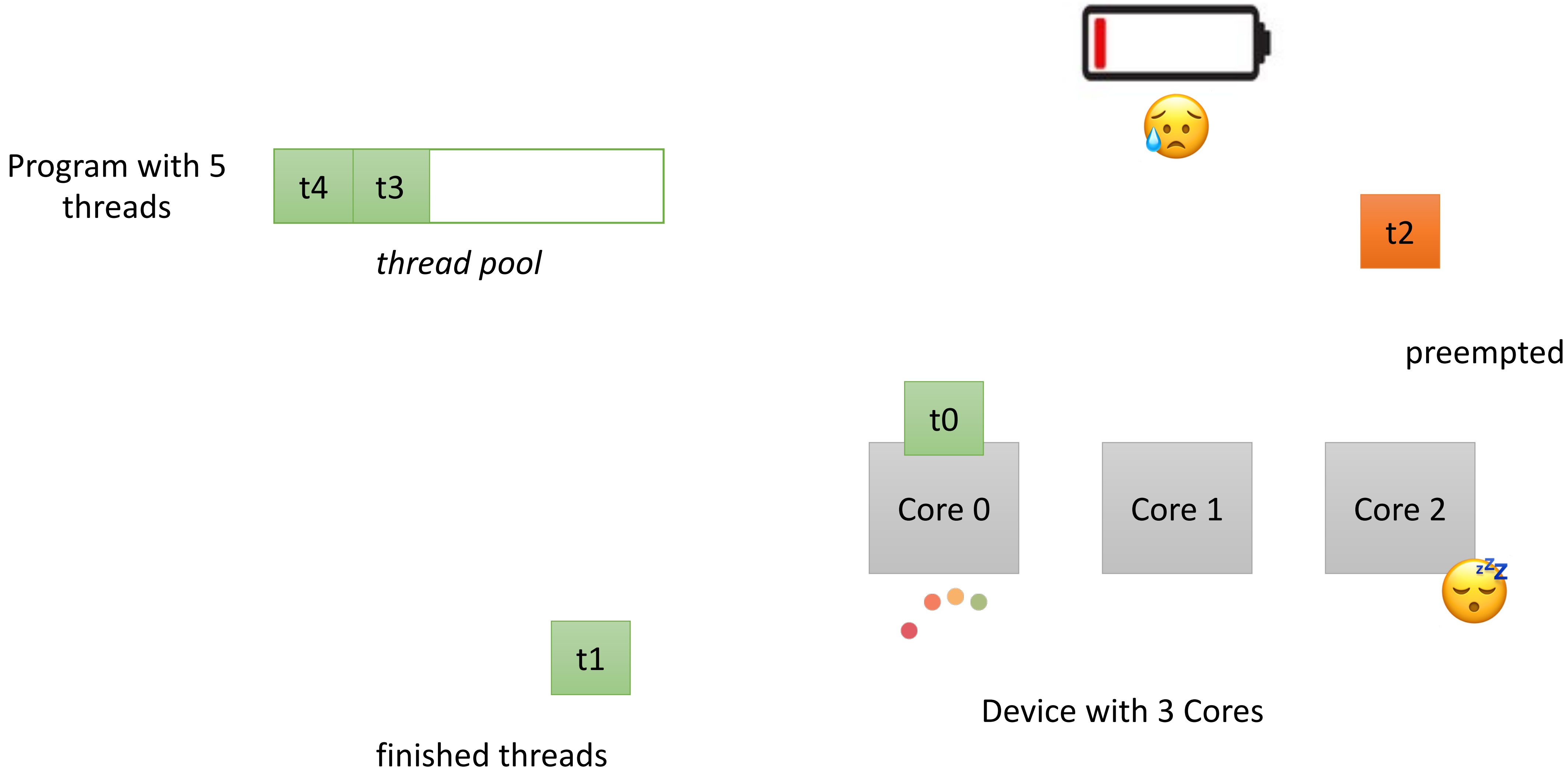
A power-saving scheduler



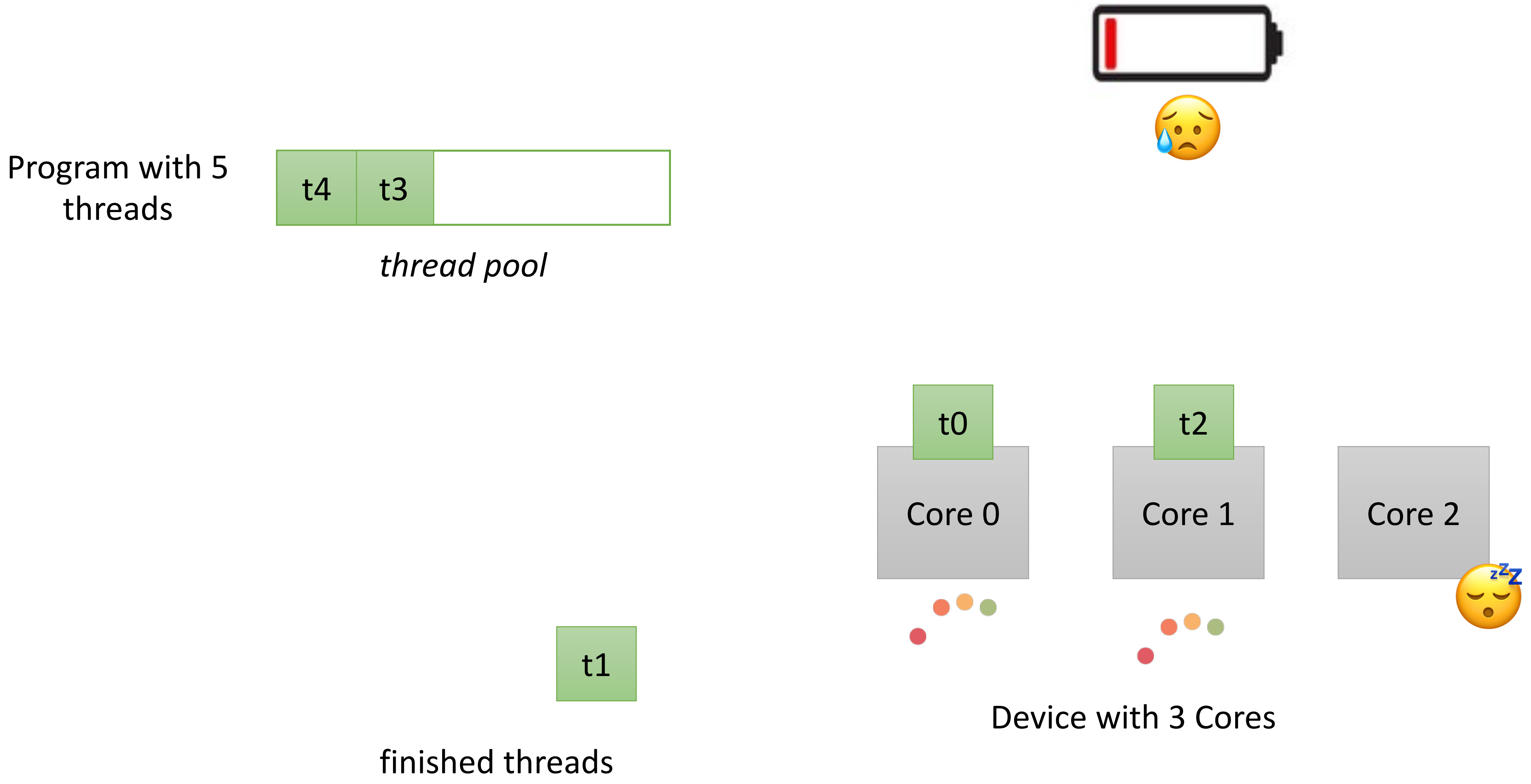
A power-saving scheduler



A power-saving scheduler



A power-saving scheduler



Thread 0:

```
0.0: m.lock();
0.1: m.unlock();
```

Thread 1:

```
1.0: m.lock();
1.1: m.unlock();
```

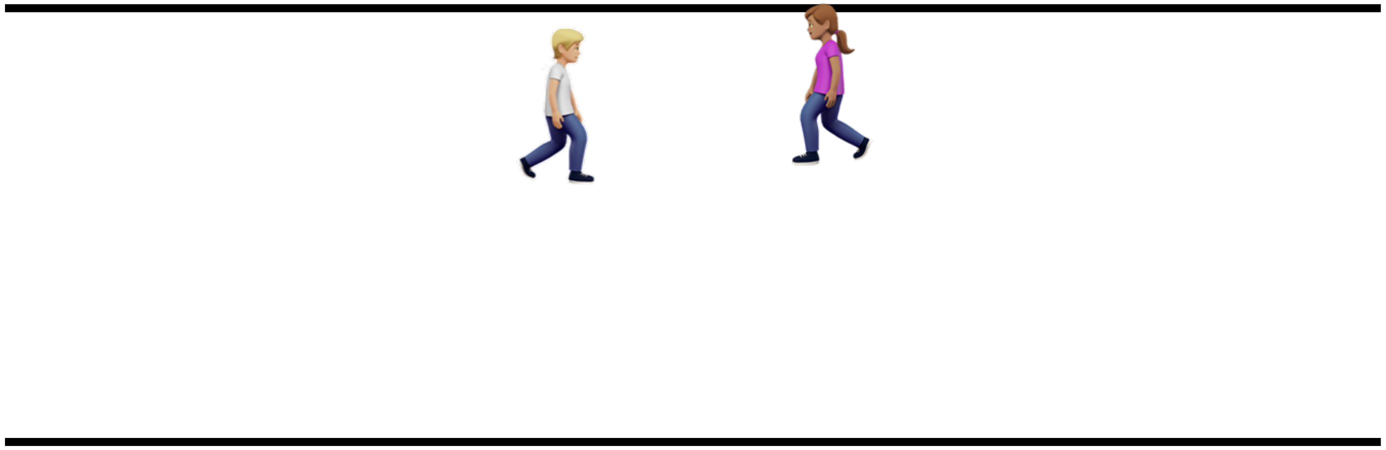
What about a mutex? Which scheduler is it guaranteed to work with?

A different type of non-termination

Hallway problem

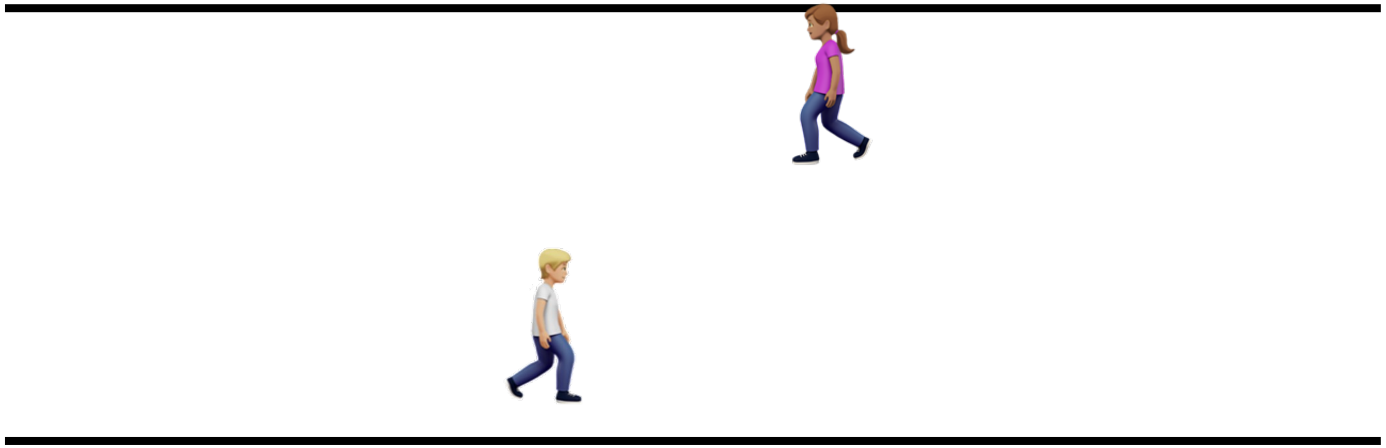
A different type of non-termination

Hallway problem



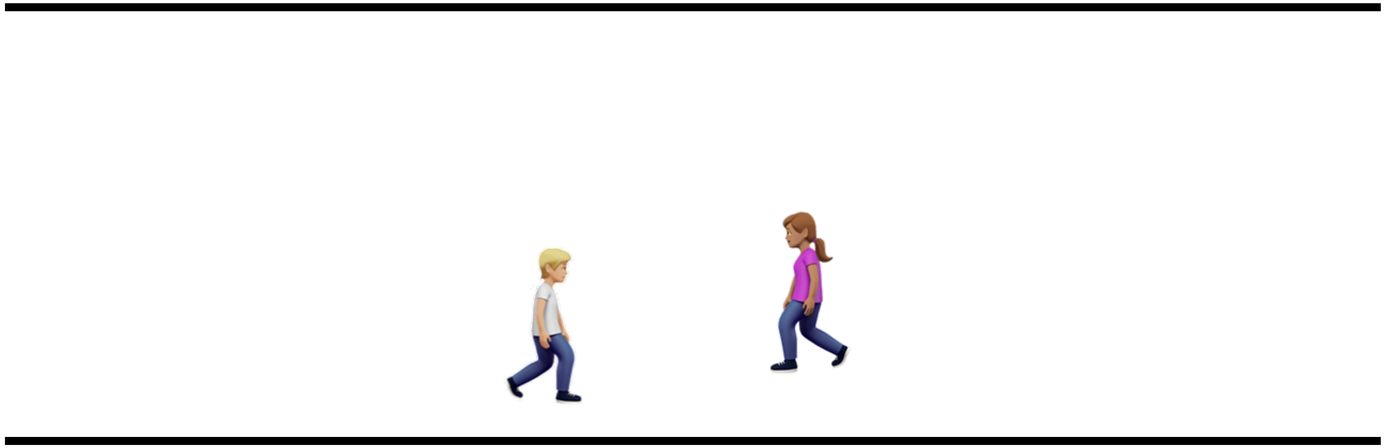
A different type of non-termination

Hallway problem



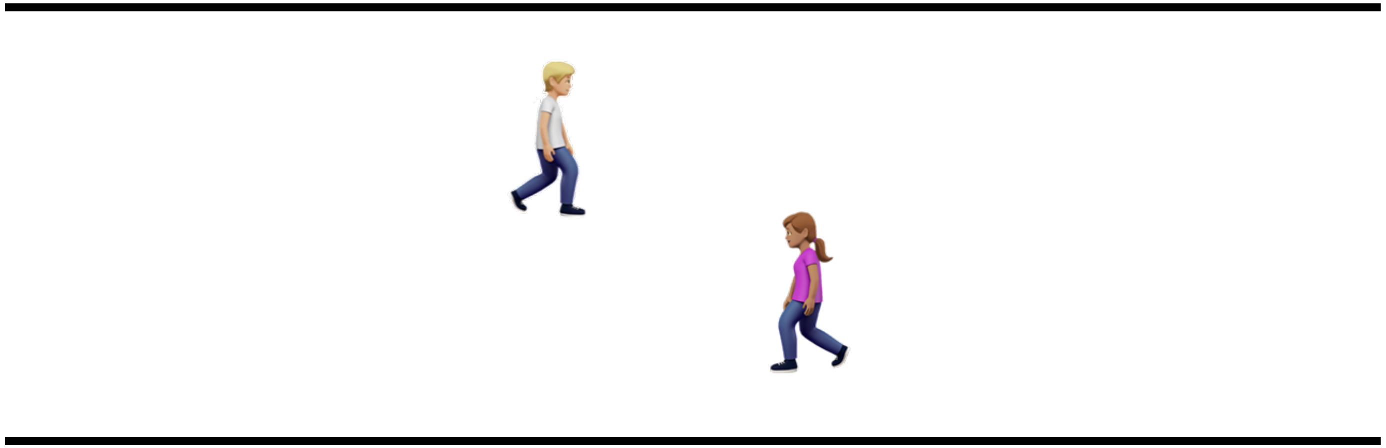
A different type of non-termination

Hallway problem



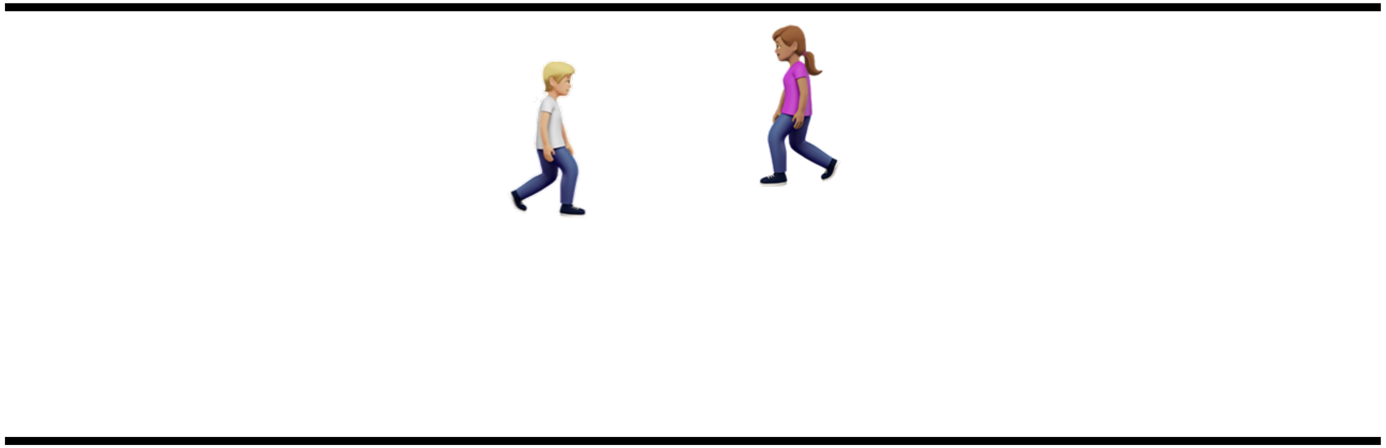
A different type of non-termination

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

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

GPUs: a brief history

- Hard to track everything down
  - First chapter of CUDA by Example
  - https://www.techspot.com/article/650-history-of-the-gpu/
- Please send me any other references you might find!

The very beginning

- Specialized hardware to accelerate graphics rendering
- One of the first real-time computers: Whirlwind 1 at MIT (1951)
  - Flight simulator for bombers
  - vector graphics

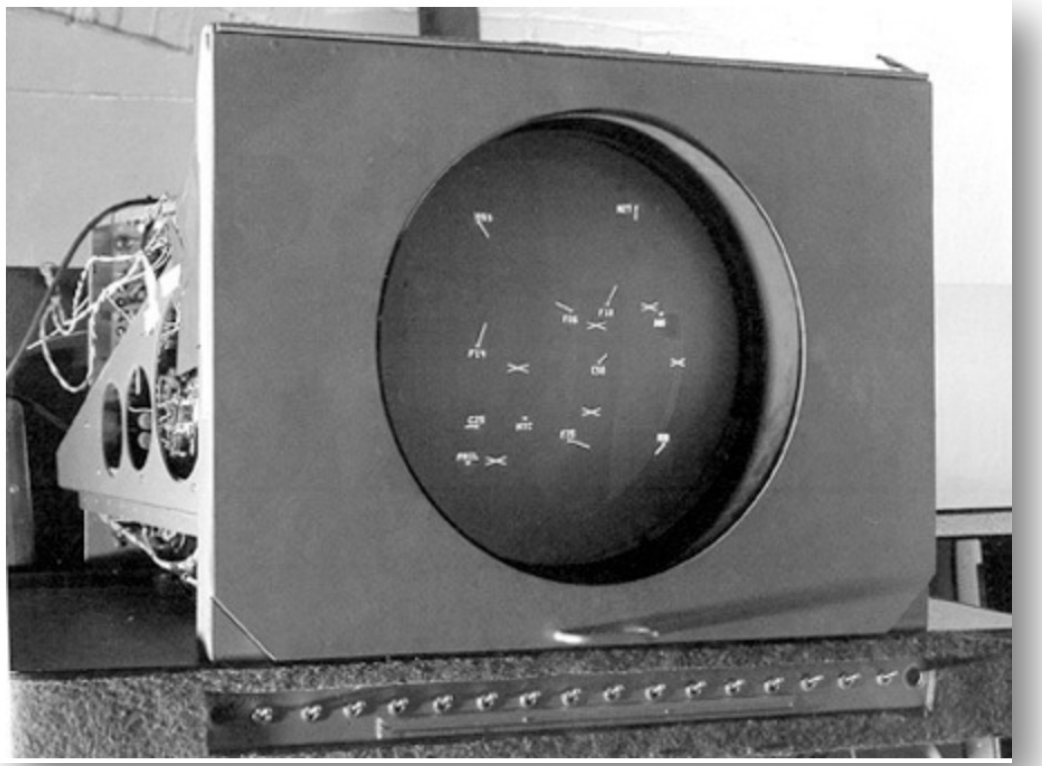


Image from: https://ohiostate.pressbooks.pub/graphicshistory/chapter/2-1-whirlwind-and-sage/

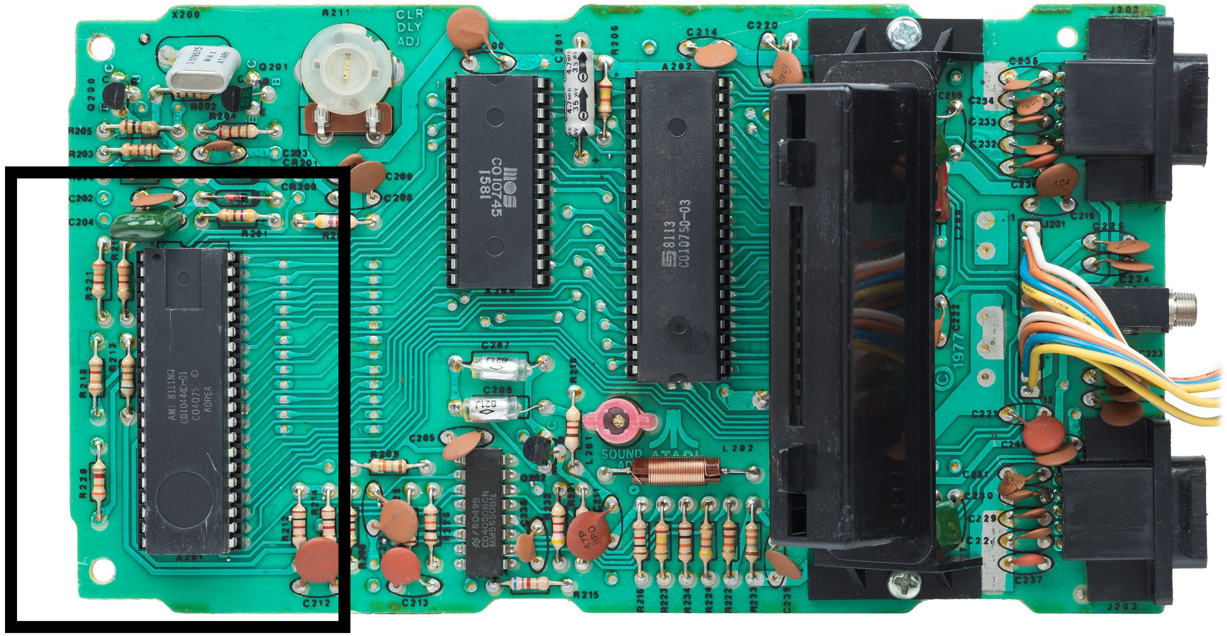
Specialization

- Next 30 years, specialized hardware for specialized software to display 2D graphics

- Specialized
  - Typically ran specific programs
  - portability was not a top priority
  - Even the idea of portable ISAs were not mainstream

Multi-program devices

- 1977: Television Interface Adapter
  - One of the first (and widely produced) portable (i.e. multiple program) GPUs



from: https://en.wikipedia.org/wiki/Television_Interface_Adaptor https://en.wikipedia.org/wiki/DirectX

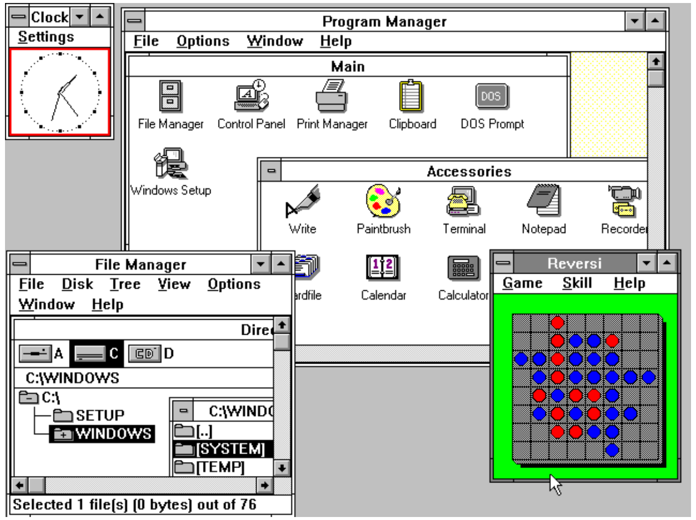
https://en.wikipedia.org/wiki/Microsoft_Windows

https://en.wikipedia.org/wiki/OpenGL

OS integration

- 1990s: Windows: a graphical operating systems, required chips to support 2D graphics.

- New APIs starting appearing, to enable GUI programs



Windows 3 (1990)

1992

1995

3D graphics in consoles (1993)

- Super Nintendo was not powerful enough to draw 3D graphics
- Shigeru Miyamoto really wanted a 3D flight simulator though
- Worked with a British software company to develop...

3D graphics in consoles (1993)

- Super Nintendo was not powerful enough to draw 3D graphics
- Shigeru Miyamoto really wanted a 3D flight simulator though
- Worked with a British software company to develop...

https://en.wikipedia.org/wiki/Star_Fox_(1993_video_game)

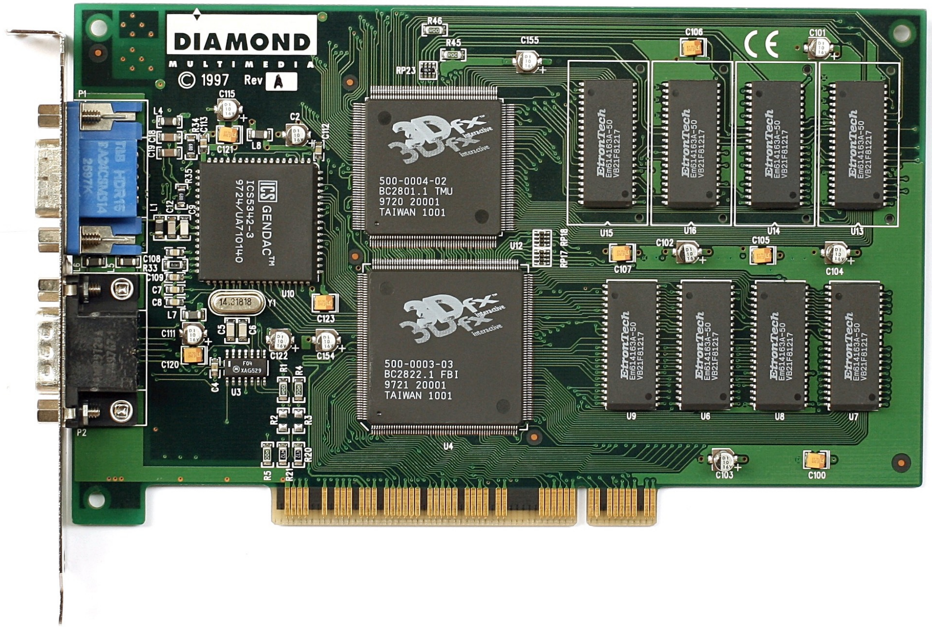
3D graphics in consoles (1993)

- Game cartridges shipped with a “mini GPU” on them:
  - the Super FX

https://twitter.com/gameminesocials/status/1322946537077526528?s=20

3D graphics acceleration

- 1996 : First 3D graphics accelerator: 3Dfx Vodoo
  - Discrete GPU
  - Early 3D games: e.g. tomb raider
  - Acquired by Nvidia in 2002



https://en.wikipedia.org/wiki/3dfx_Interactive

3D graphics acceleration

- 3D accelerators continued, many companies competing:
  - Nvidia
  - ATI
  - 3Dfx
  - and more...

- Next milestone in 1999:
  - Nvidia coins the term “GPU”
  - Compare with modern website

https://web.archive.org/web/20030814003456/www.nvidia.com/object/gpu.html

Programmable 3D accelerators

- 2001: Microsoft DirectX 8 required programmable vertex and pixel shaders.
- 2001: First GPU to satisfy the requirement was Nvidia GeForce 3
  - we are now on 17
  - Used on the original Xbox
- Programmers started writing general programs for these GPUs:
  - Present your data as a graphical input (e.g. Textures and Triangles)
  - Read the output after a series of “graphics” API calls

GPGPU Programming

- 2006: Nvidia releases CUDA: programming language for their GPUs
  - Supported by 8th generation CUDA devices.
  - Integrated vertex and pixel cores into “shader cores”
  - Support for IEEE floating point

- Soon after...

GPGPU Programming

- 2006: Nvidia releases CUDA: programming language for their GPUs
  - Supported by 8th generation CUDA devices.
  - Integrated vertex and pixel cores into "shader cores"
  - Support for IEEE floating point
- Soon after...
- 2008: The Khronos Group launches OpenCL for cross vendor GPGPU:
  - including AMD, Intel, Qualcomm

Khronos Group

- Started in 2000 by Apple as a standards body for graphics API:
  - A way to unify APIs across many different vendors
  - at the time: ATI, Nvidia, Intel, Sun Microsystems (and a few others)
  - now: Many companies, including AMD, Nvidia, Intel, Qualcomm, ARM, Google

  - OpenGL is maybe the biggest standard they maintain (for graphics)
  - OpenCL is biggest for compute
  - Vulkan is their new standard (will it catch on??)
  - (disclosure: I am an individual contributor ☺)

- Apple deprecated Khronos group standards to support Metal in 2018

https://en.wikipedia.org/wiki/Khronos_Group

Where are we now?

- Nvidia CUDA is widely used, driving many HPC and ML applications
- OpenCL is used to program other GPUs (although it is not as widely used)
- Metal is used for Apple devices
- Vulkan has momentum

- New GPGPU programming languages are on the horizon:
  - WebGPU - a javascript interface to unite Metal, Vulkan and DirectX
  - Its ambitious! Will it work?!
  - Available in canary builds of Chrome

GPU Shortages?

- Cryptocurrency:
  - 2018 reported tripling of GPU prices and shortages due to increase demand from miners.
  - Still happening will lots of market fluctuations.
  - 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 120 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*

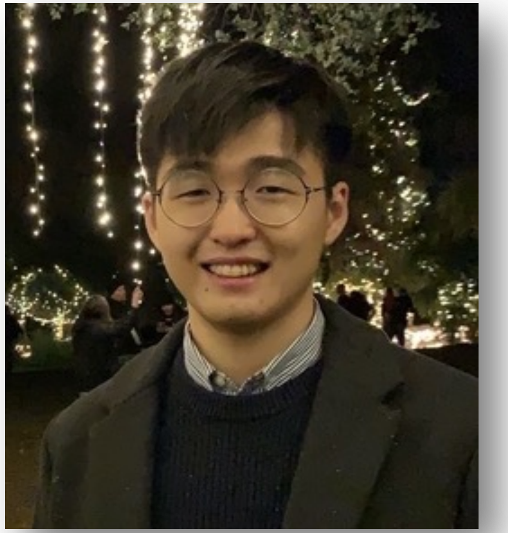
Going forward

- The homework uses Javascript as its ”CPU” language, and webGPU as its ”GPU” 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 (possibly we need canary version)
  - should be stable on Windows and Mac
  - if you are running linux, please try things out ASAP

- 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 to be able to run a local web server.

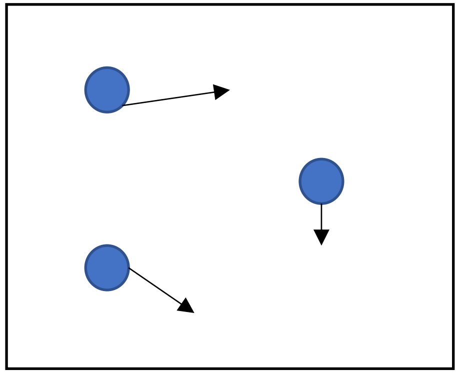
Homework 5- first look

- Let’s have a look!

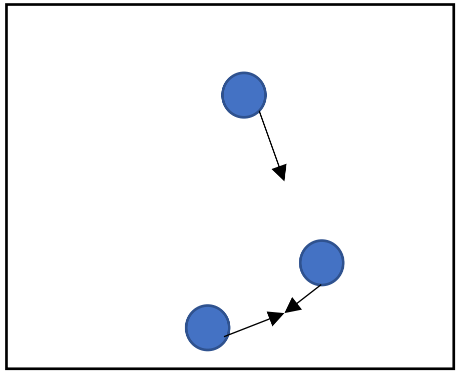
Homework 5- first look

- Your assignment:
  - N-body simulation

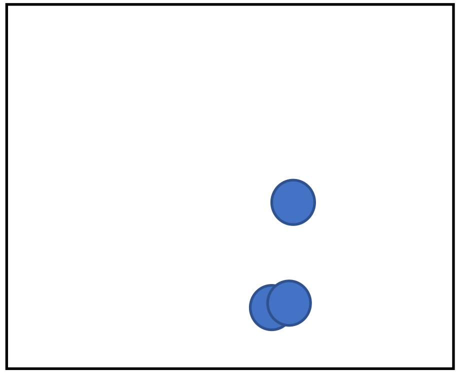
- Each particle interacts with every other particle



time = 0



time = 1

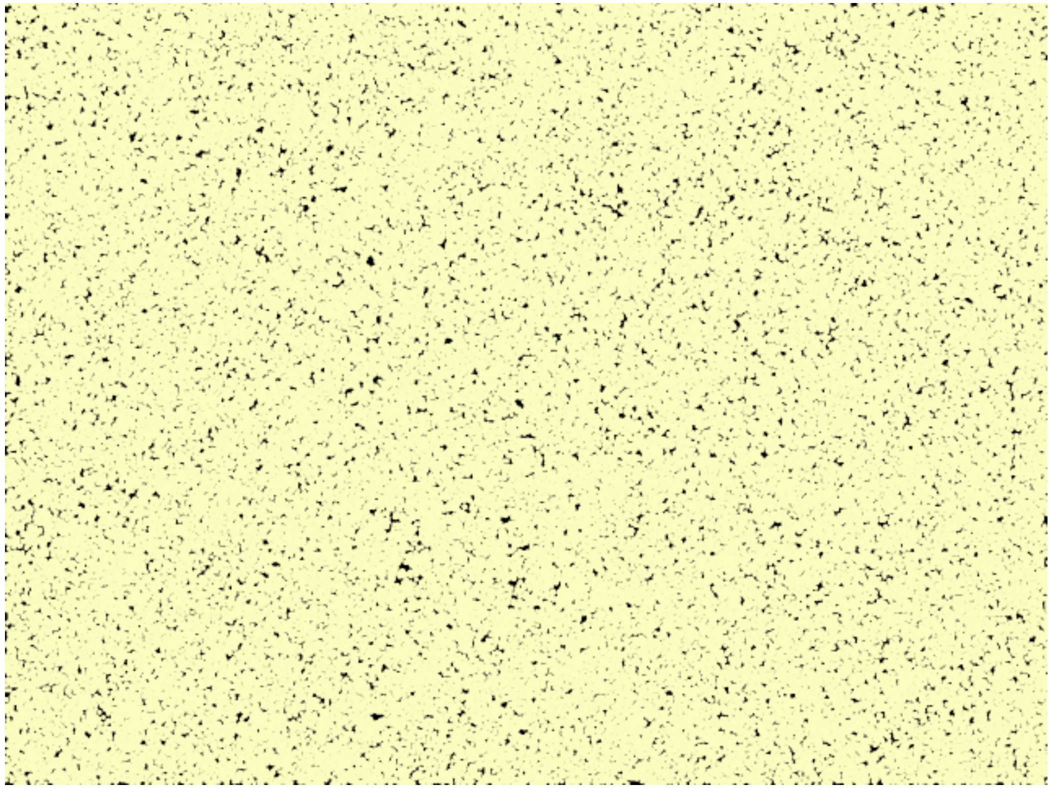


time = 2

Examples

- Gravity:

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



Your homework

- Boids and N-body require a little bit of physics background so we will do something simpler.
  - If you want to explore with physics please feel free

- Local attraction clustering:
  - For each particle: find your closest neighbor
  - You can take one step in the x direction and one step in the y direction towards your closest neighbor.

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 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

Shared Array Buffer

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

See you on Monday

- Turn in HW 4 if you haven't already
- Working on GPU programming!