PSAM 5600 Section E CRN 4598
Spring Semester 2017
Wednesdays 7:00PM – 9:40PM

Currents: Coding with Cinder

Week 2: C++ fundamentals recap / surface, texture and shader

Instructors
Luobin Wang (luobin@newschool.edu)
Weili Shi (weili@newschool.edu)

```
/* This is a comment */

/* C++ comments can  also
 * span multiple lines
 */

#include <iostream>

int main() { // the main function where program execution begins.
   std::cout << "Hello World” << std::endl;
   return 0;
}
```

C++ Fundamentals Recap

```
#include <iostream>

int main() {
   if(1 == 2)
       std::cout << “I know this line won’t execute.” << std::endl;
       std::cout << “I didn’t know this line will.” << std::endl;

   while(true) { // infinite loop
        std::cout << “Can’t Stop Won’t Stop”;
    }
   return 0;
}
```

Flow control

if else do while for switch case break default

true false == != > >= < <= && || !

Primitive Built-in Types

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Data Types

Definition vs. Declaration

A **declaration** provides basic attributes of a symbol: its type and its name.

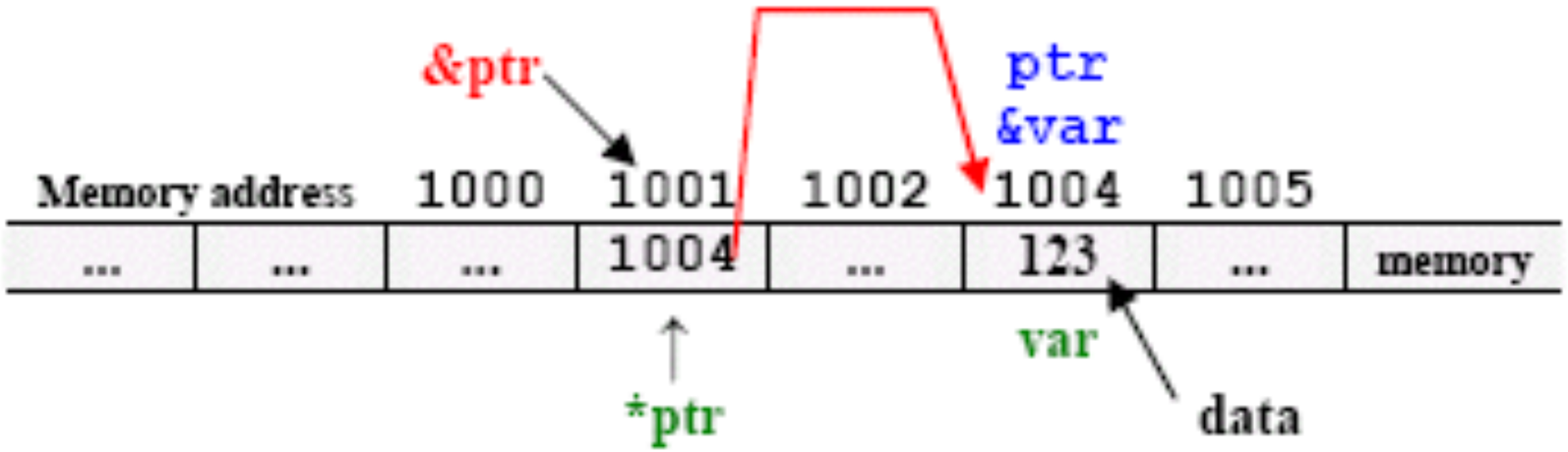
A **definition** provides all of the details of that symbol—if it's a function, what it does; if it's a class, what fields and methods it has; if it's a variable, where that variable is stored.

http://www.cprogramming.com/declare_vs_define.html

```
bool IsHandsome(std::string name) {
    return true;
}
```

Functions & Operators

A C++ function definition consists of **return type**, **function name**, **parameter list**, and **function body**. In C++, you can overload operators in the way you overload functions.



Pointers & Arrays

http://www.tenouk.com/Module8_files/cpluspointer032.png

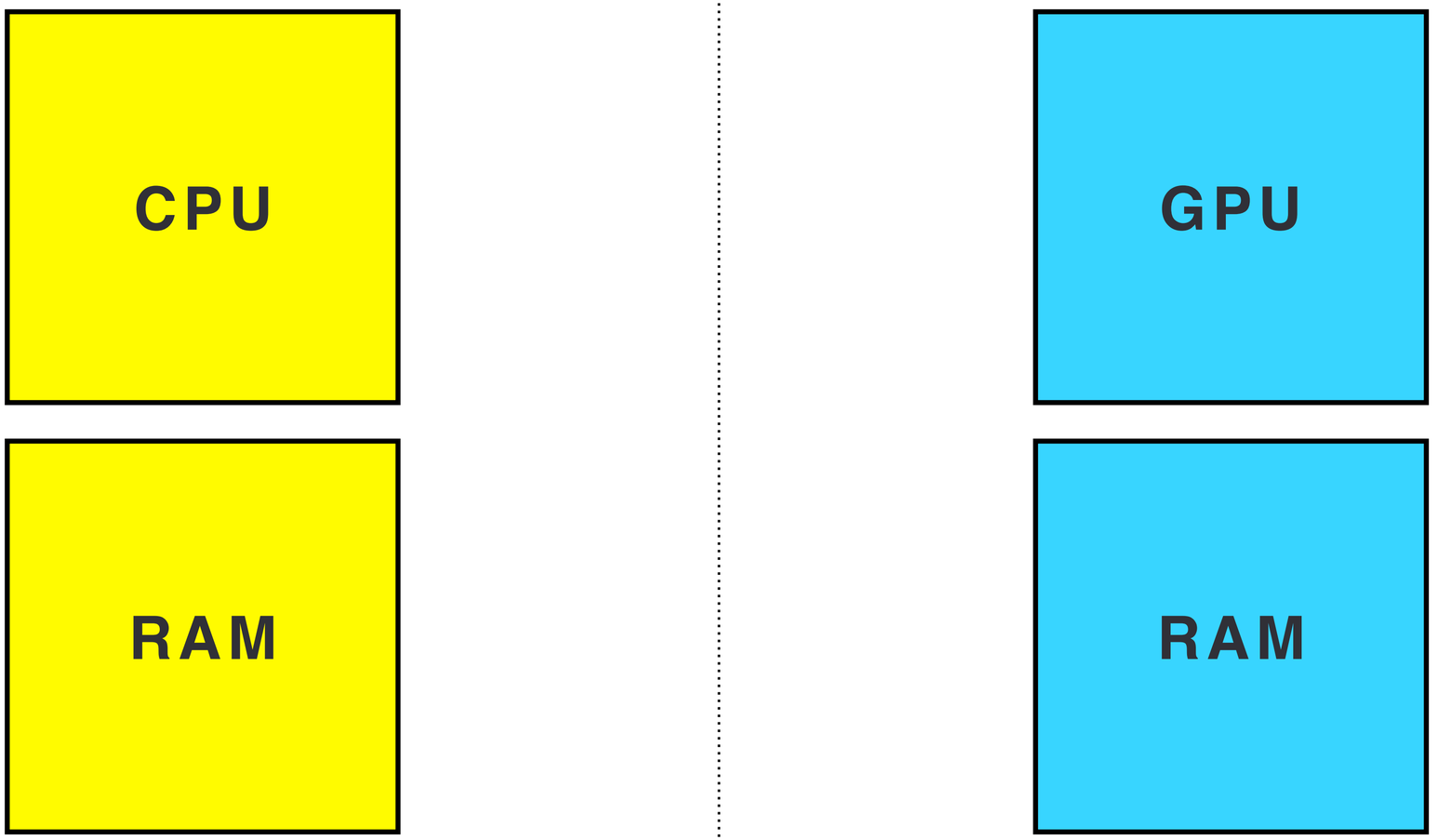
```
#include <iostream>
```

The Preprocessor

The preprocessor modifies a source code file before handing it over to the compiler. You're most likely used to using the preprocessor to **include files** directly into other files, or **#define constants**, but the preprocessor can also be used to **create "inlined" code using macros** expanded at compile time and to **prevent code from being compiled twice**.

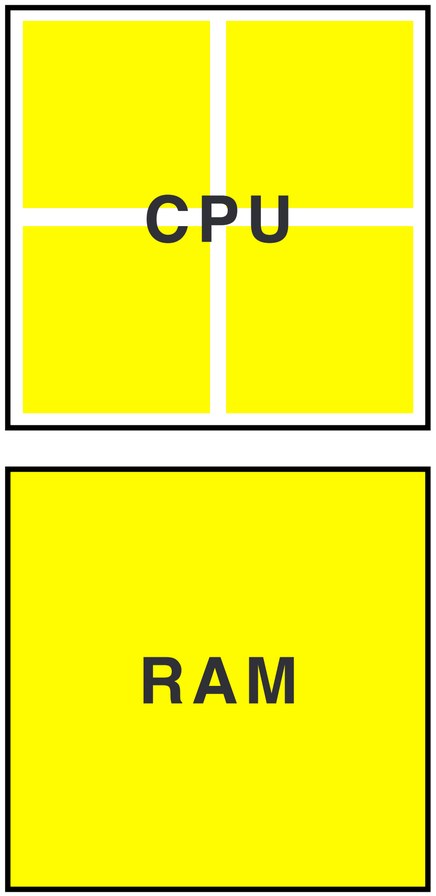
http://www.cprogramming.com/tutorial/cpreprocessor.html

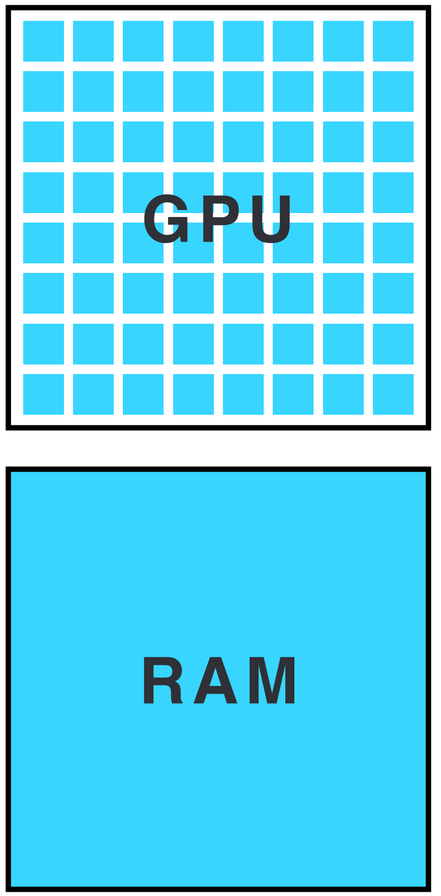
CPU vs GPU



Credit: Ben Bojko, Bluecadet

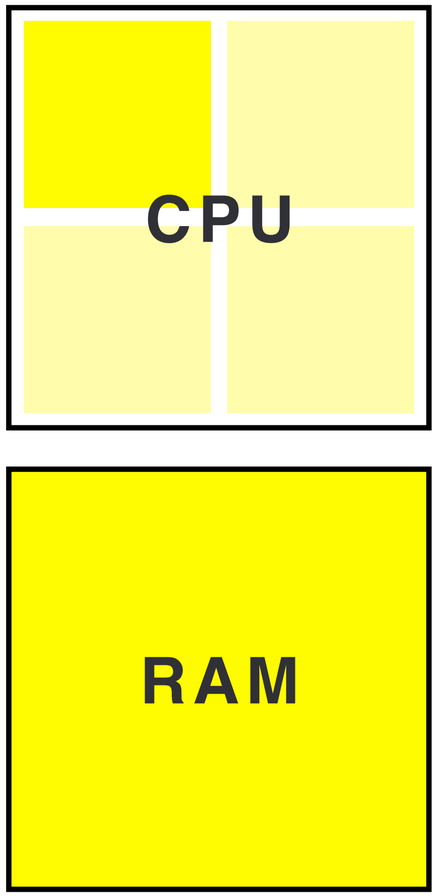
CPU vs GPU

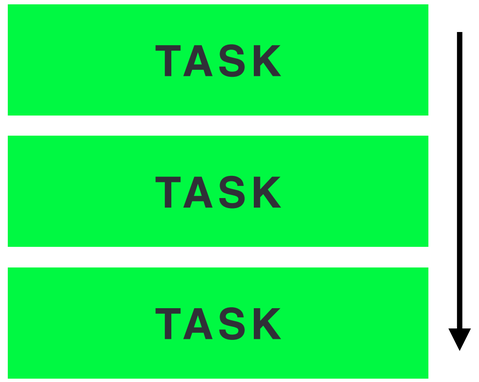




Credit: Ben Bojko, Bluecadet

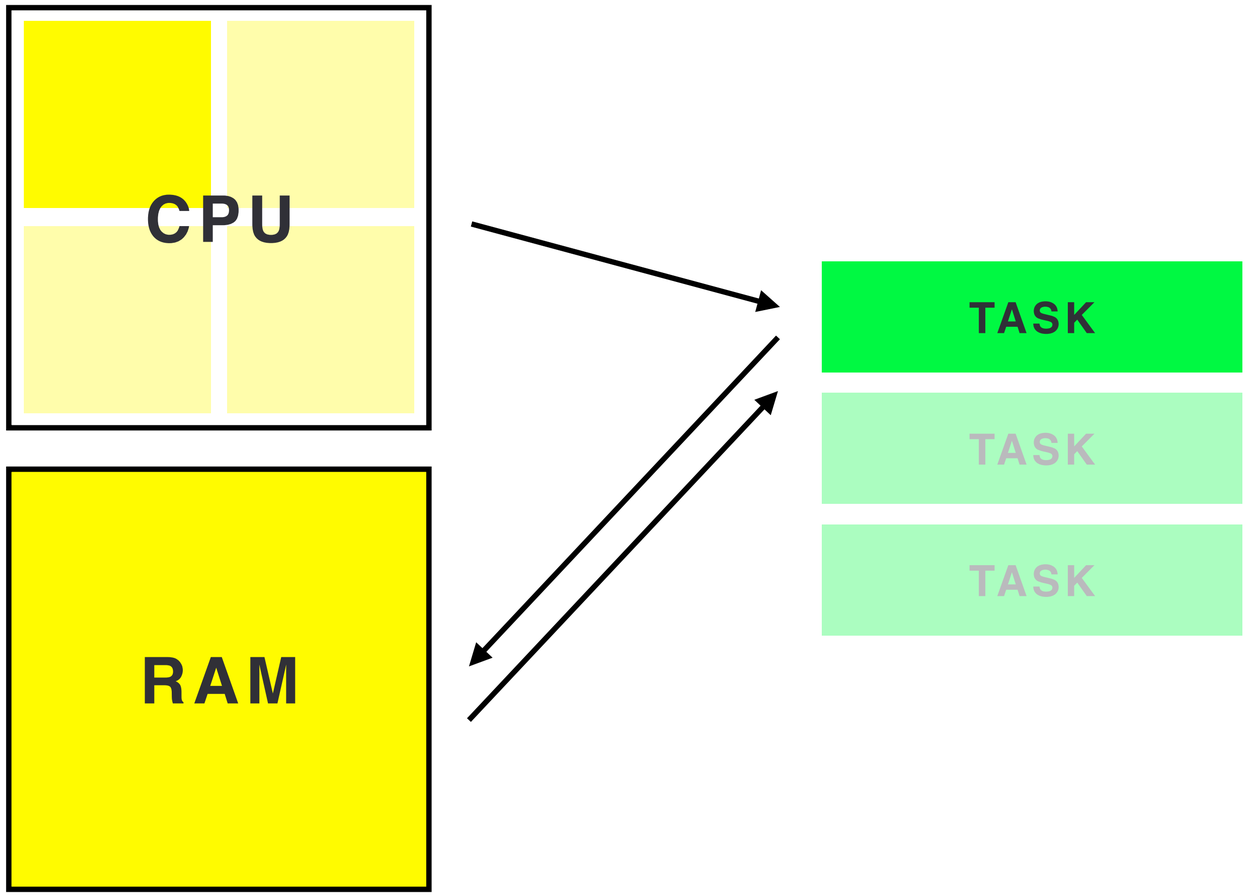
CPU Execution





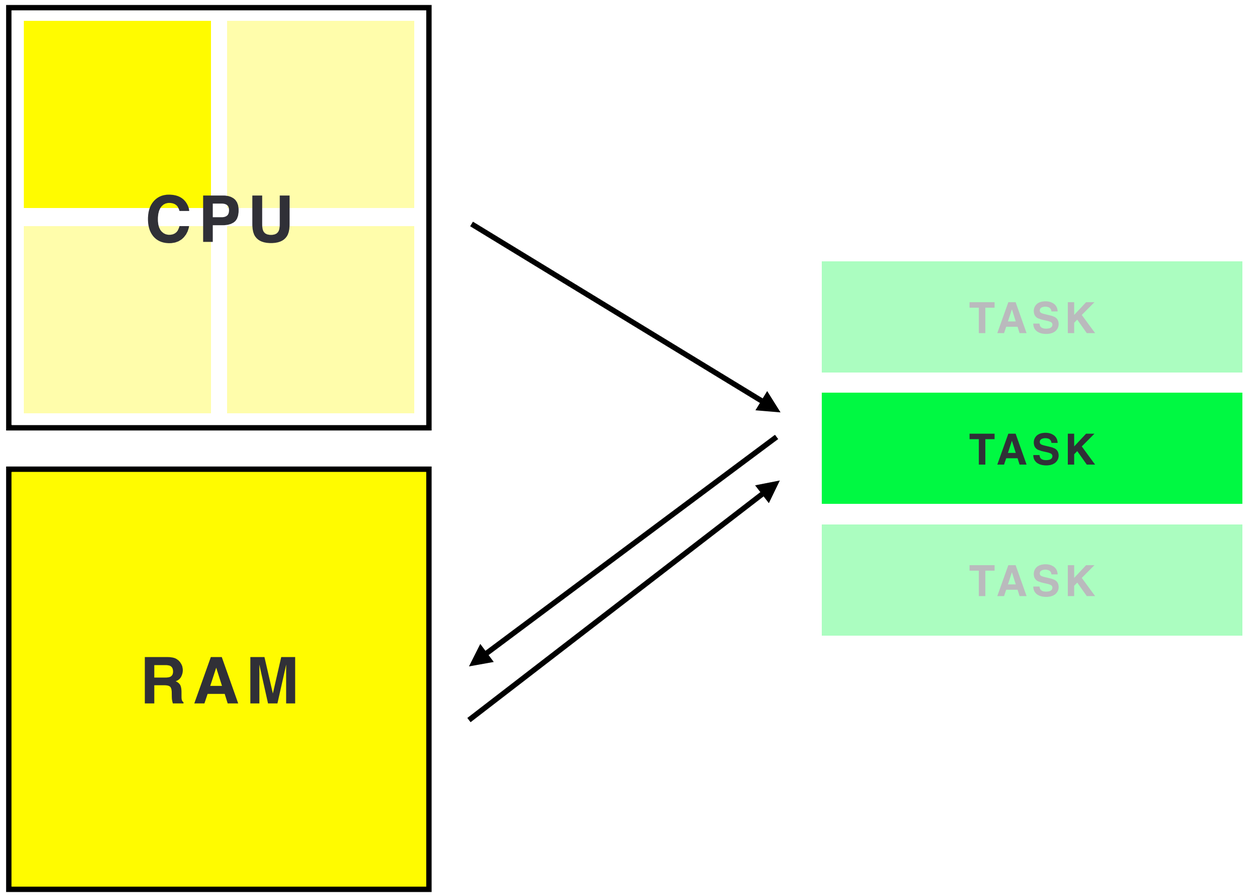
Credit: Ben Bojko, Bluecadet

CPU Execution



Credit: Ben Bojko, Bluecadet

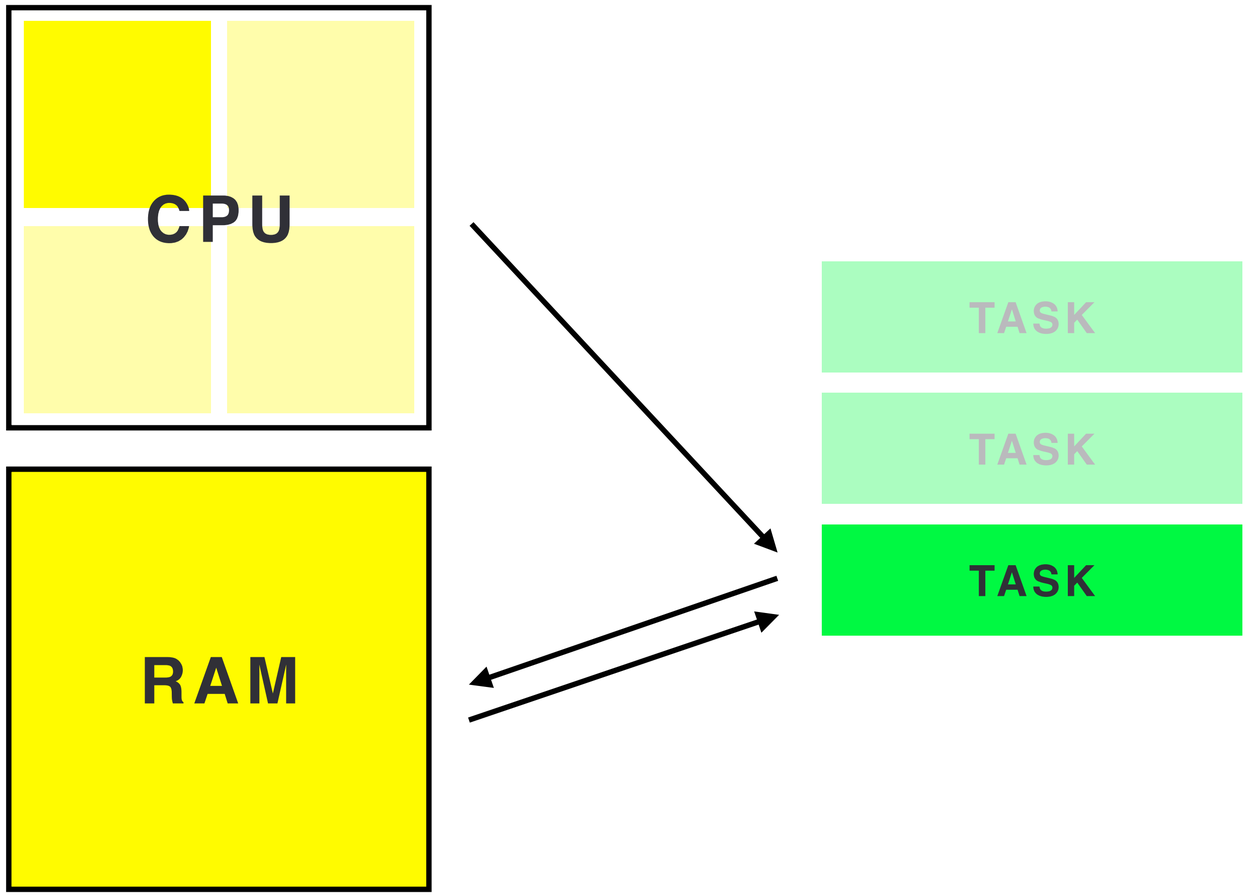
CPU Execution



Credit: Ben Bojko, Bluecadet

WHEN TO USE SHADERS

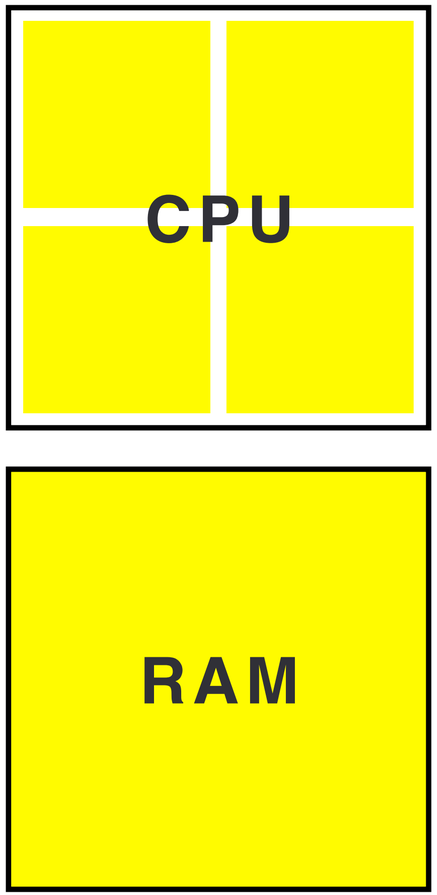
CPU Execution



Credit: Ben Bojko, Bluecadet

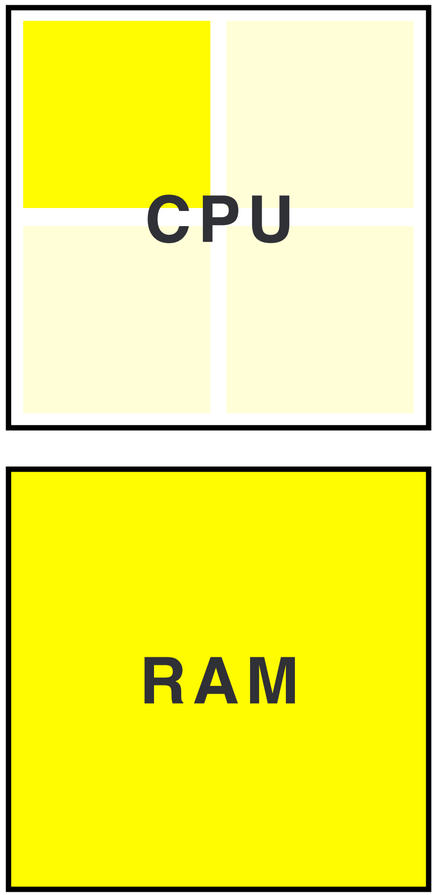
WHEN TO USE SHADERS

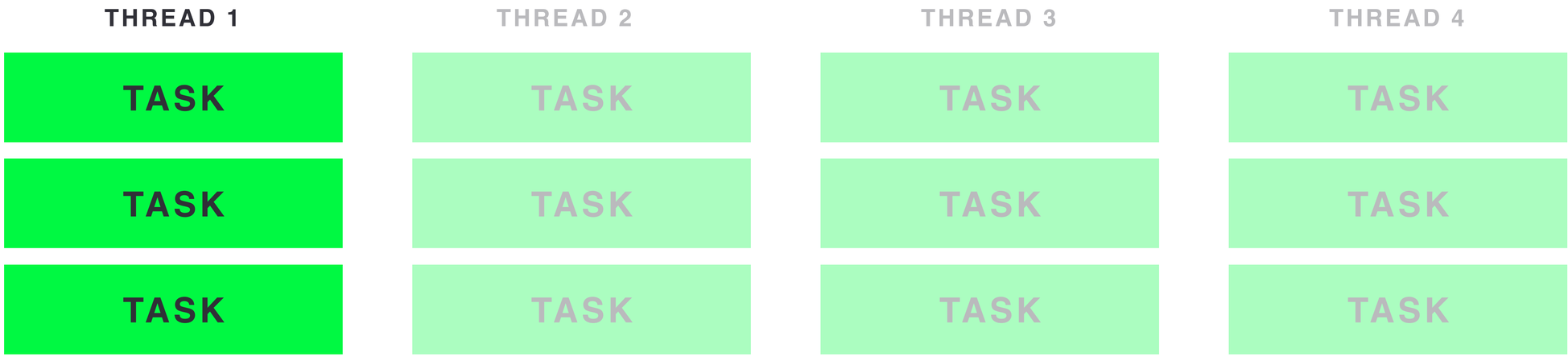
CPU Multithreading



Credit: Ben Bojko, Bluecadet

CPU Multithreading

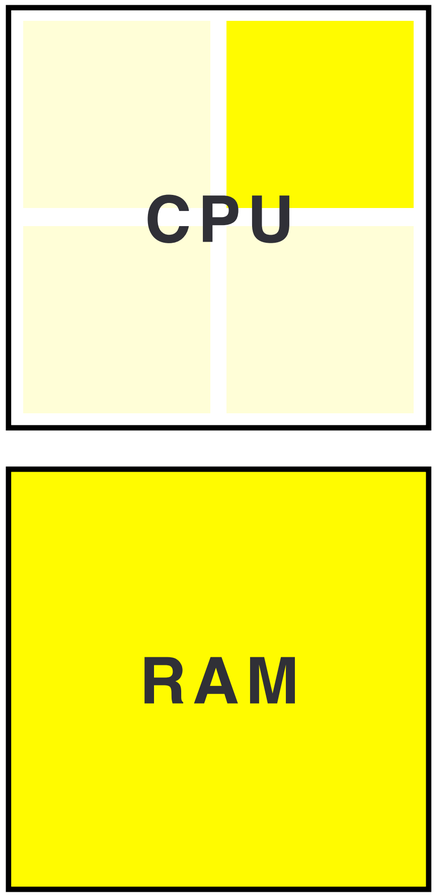




Credit: Ben Bojko, Bluecadet

WHEN TO USE SHADERS

CPU Multithreading

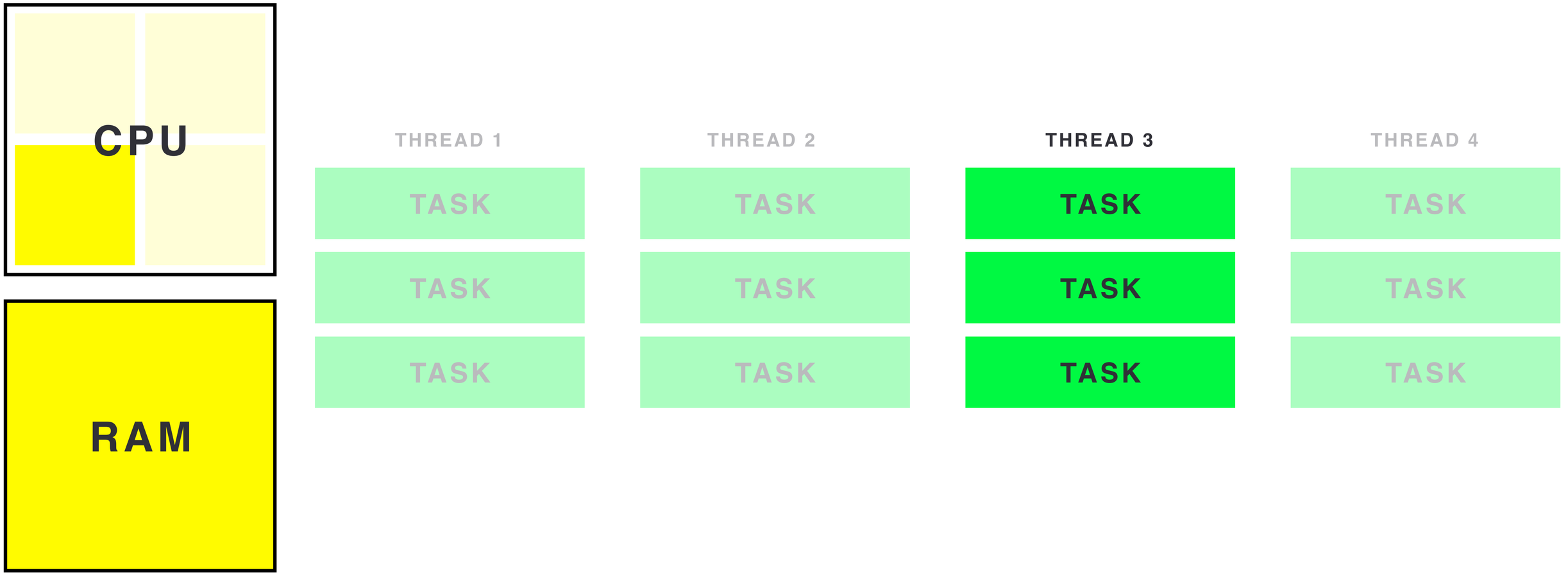


THREAD 1	THREAD 2	THREAD 3	THREAD 4
TASK	TASK	TASK	TASK
TASK	TASK	TASK	TASK
TASK	TASK	TASK	TASK

Credit: Ben Bojko, Bluecadet

WHEN TO USE SHADERS

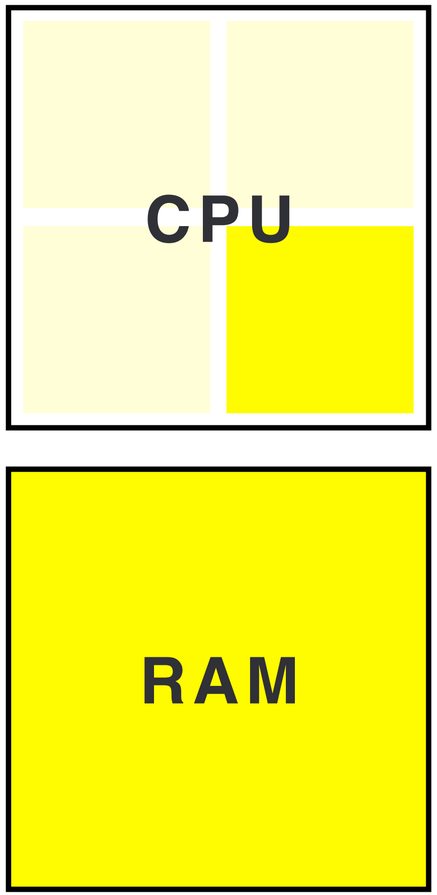
CPU Multithreading

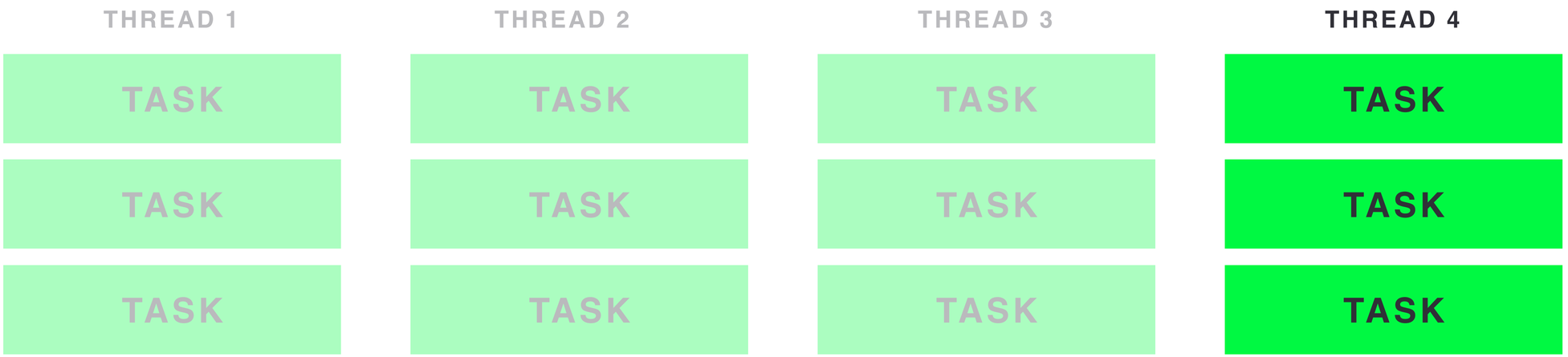


Credit: Ben Bojko, Bluecadet

WHEN TO USE SHADERS

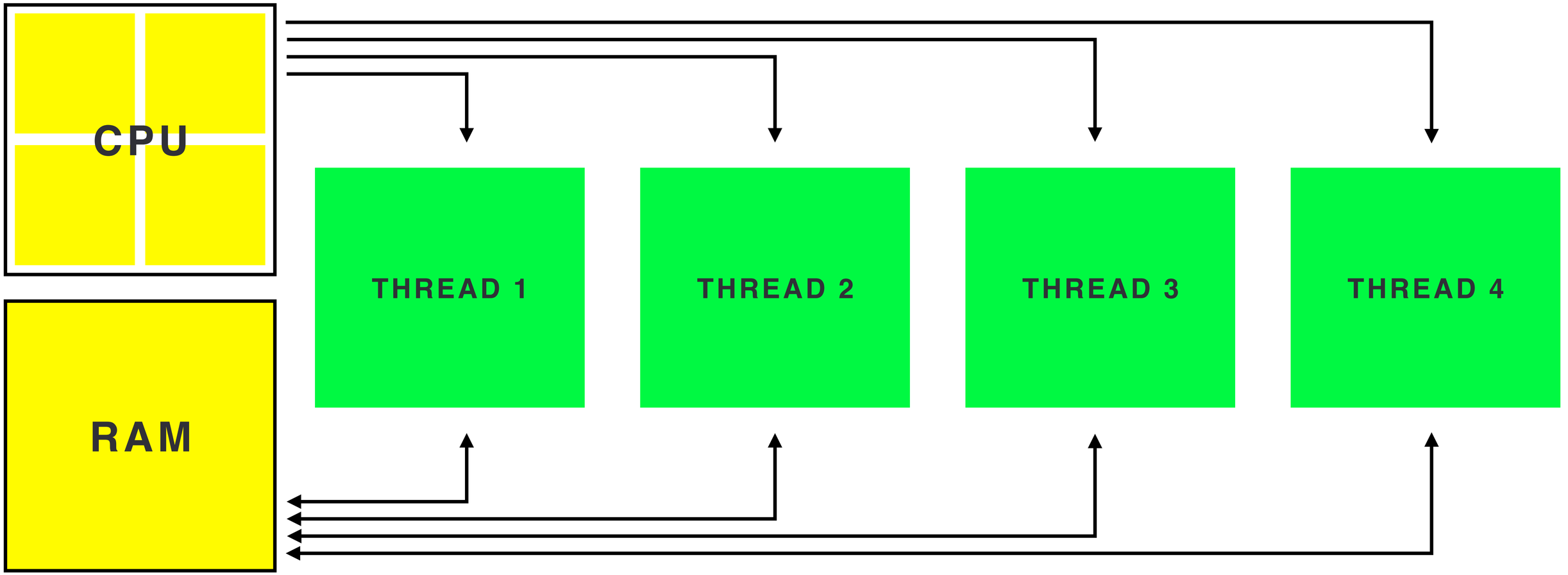
CPU Multithreading





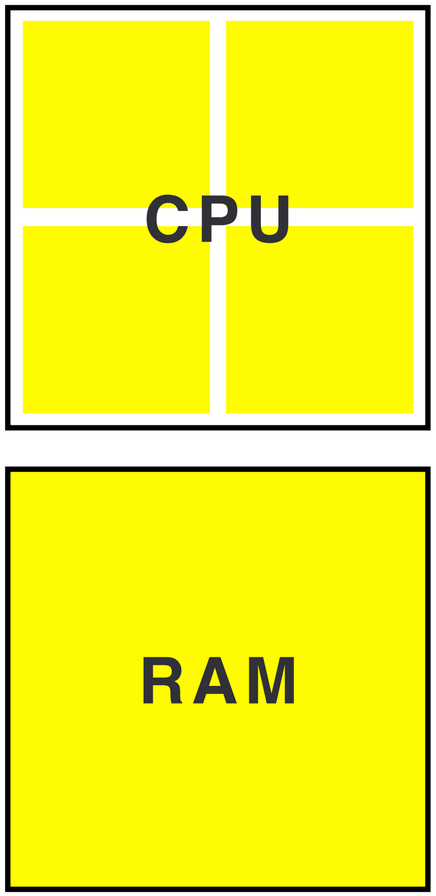
Credit: Ben Bojko, Bluecadet

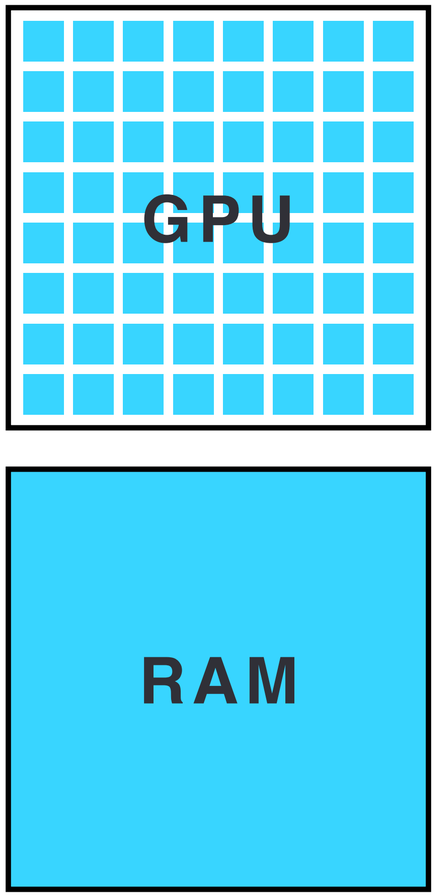
CPU Multithreading



Credit: Ben Bojko, Bluecadet

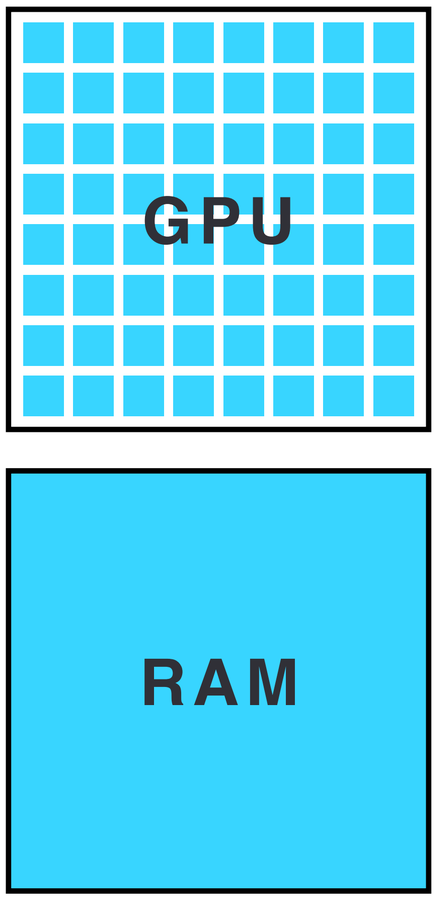
CPU vs GPU

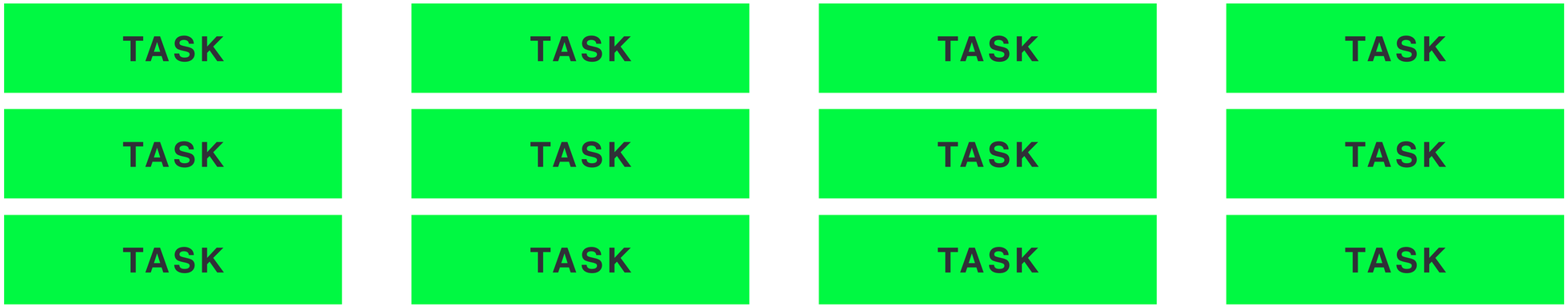




Credit: Ben Bojko, Bluecadet

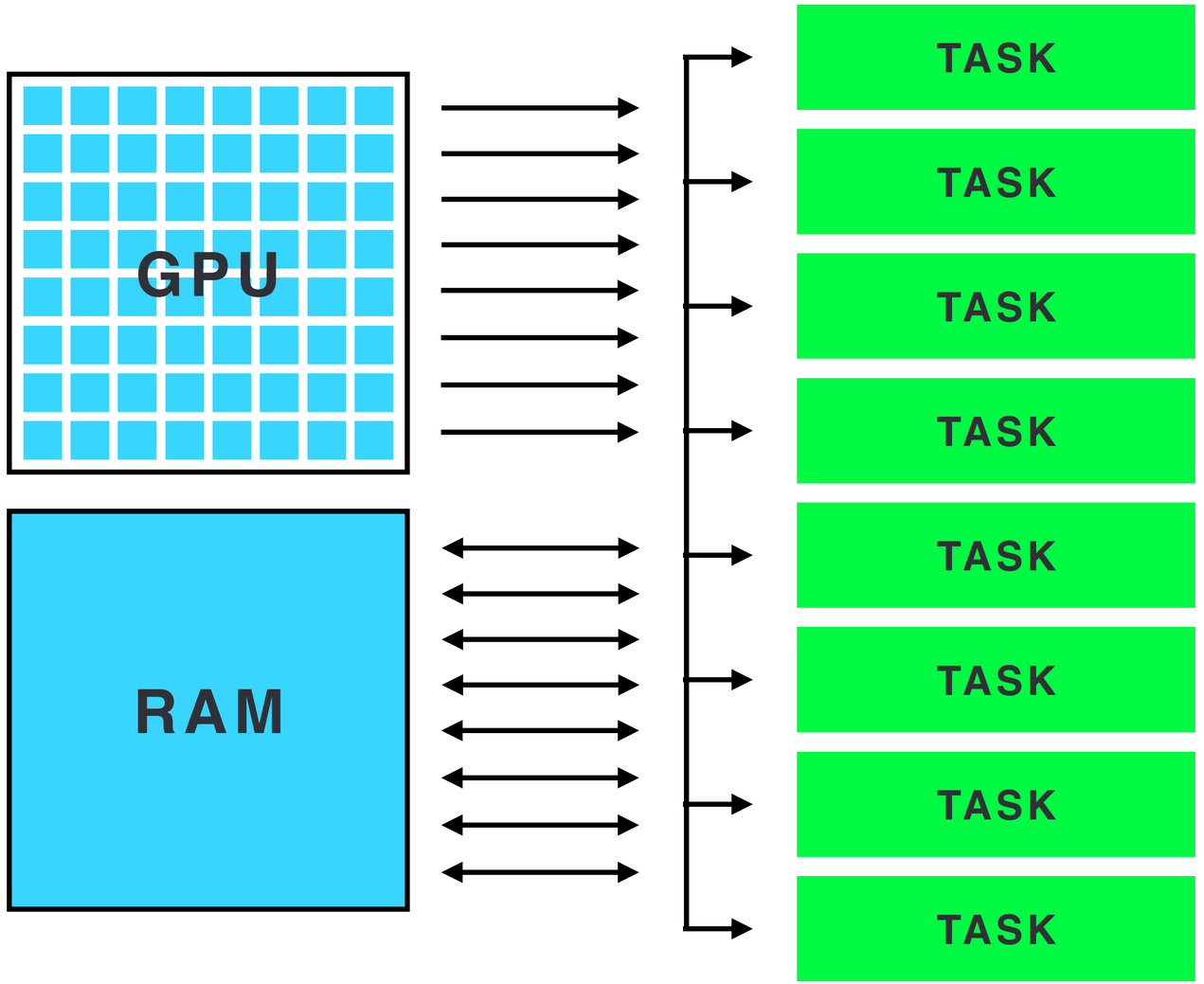
GPU Execution





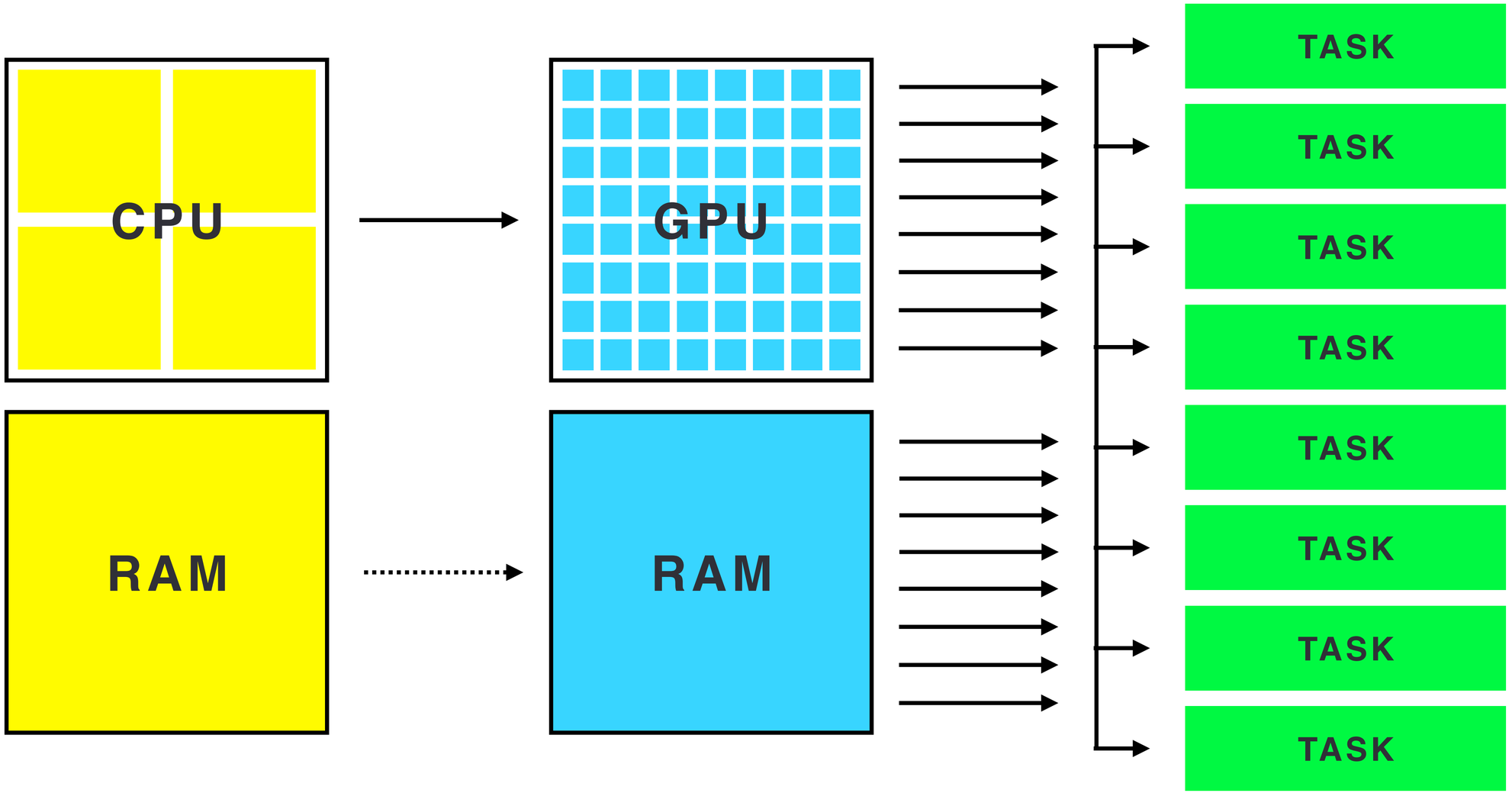
Credit: Ben Bojko, Bluecadet

GPU Execution



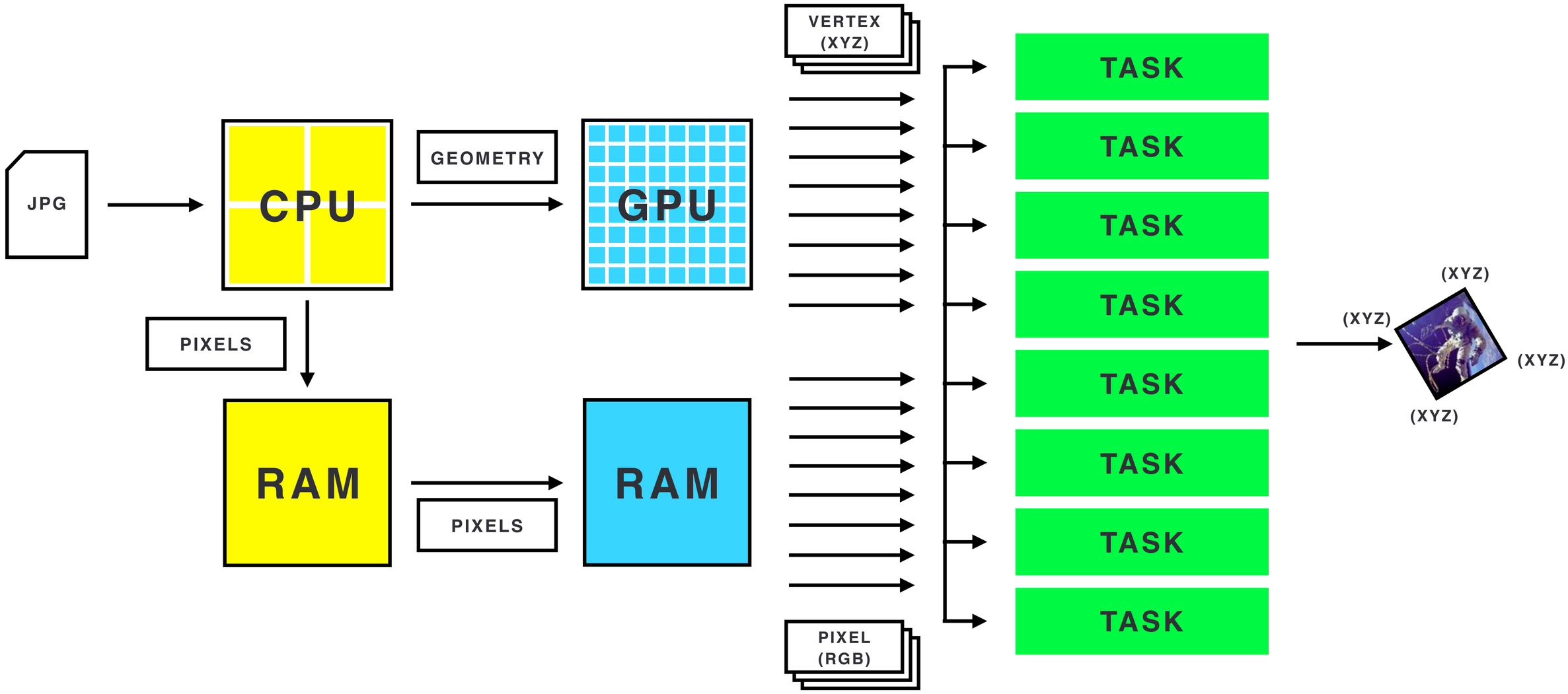
Credit: Ben Bojko, Bluecadet

GPU Execution

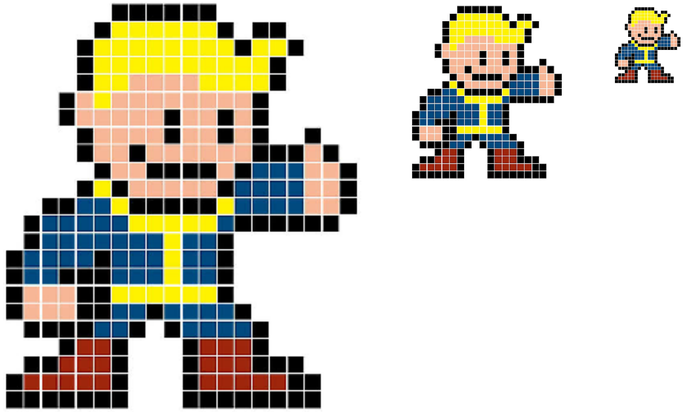


Credit: Ben Bojko, Bluecadet

Example: Rendering an Image

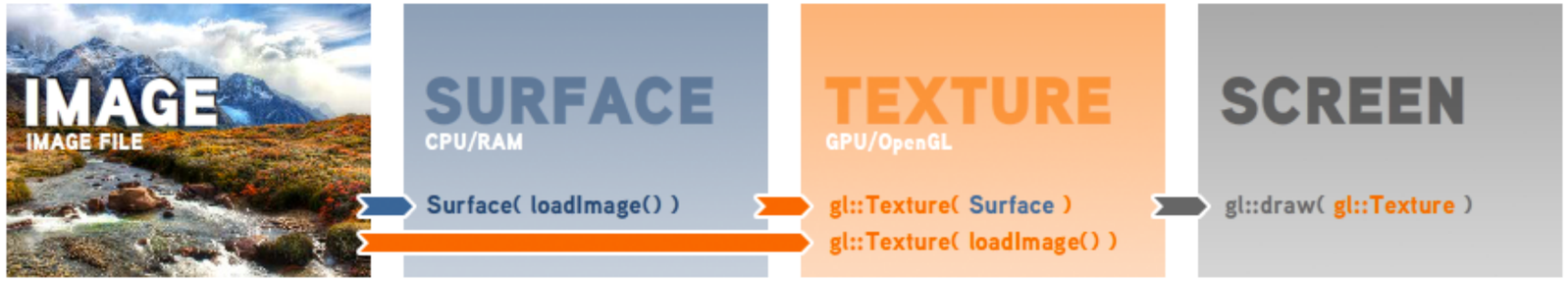


Credit: Ben Bojko, Bluecadet



What is an image made of?

An image is made of individual pixels



How an image file is being drawn on screen

Image will be stored as surface (that includes all the pixels).
Then passed to GPU as a texture and draw to the screen.

```
void addCircle(glm::vec2 pos, float radius = 3.f);
```

A declaration of a function can have a lot of flexibilities. In here, our function has one arg that is not given a value, and another arg that has been given a value. This type usage is called default arguments, which gives you a lot of convenience since this function might be used in two or more types of scenario.

https://www.programiz.com/cpp-programming/default-argument

gl::Surface

Image data. An in-memory representation of an image. Implicitly shared object. A Surface always contains red, green and blue data, along with an optional alpha channel.

https://libcinder.org/docs/classcinder_1_1_surface_t.html

gl::Texture

A texture is an OpenGL Object that contains one or more images that all have the same image format. A texture can be used in two ways. It can be the source of a texture access from a Shader, or it can be used as a render target.

http://en.cppreference.com/w/cpp/language/namespace

gl::draw() gl::draw(mTexture)

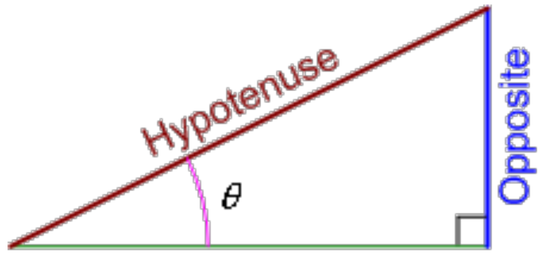
Drawing something to the screen

http://en.cppreference.com/w/cpp/language/namespace

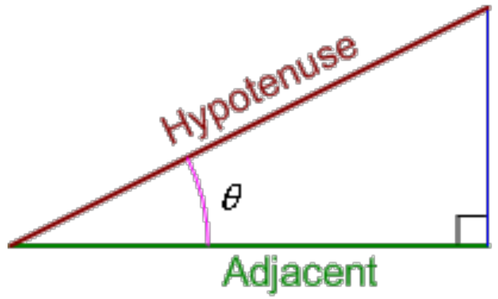
Dealing with images

Trigonometry

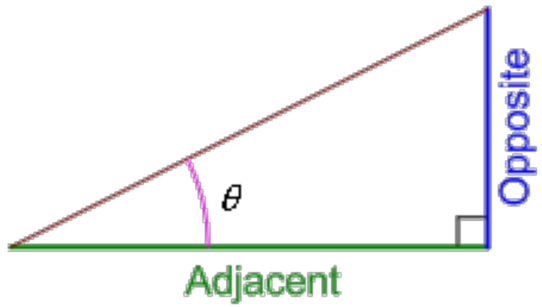
Triangles! Sin! Cos! Tan! Atan2?!



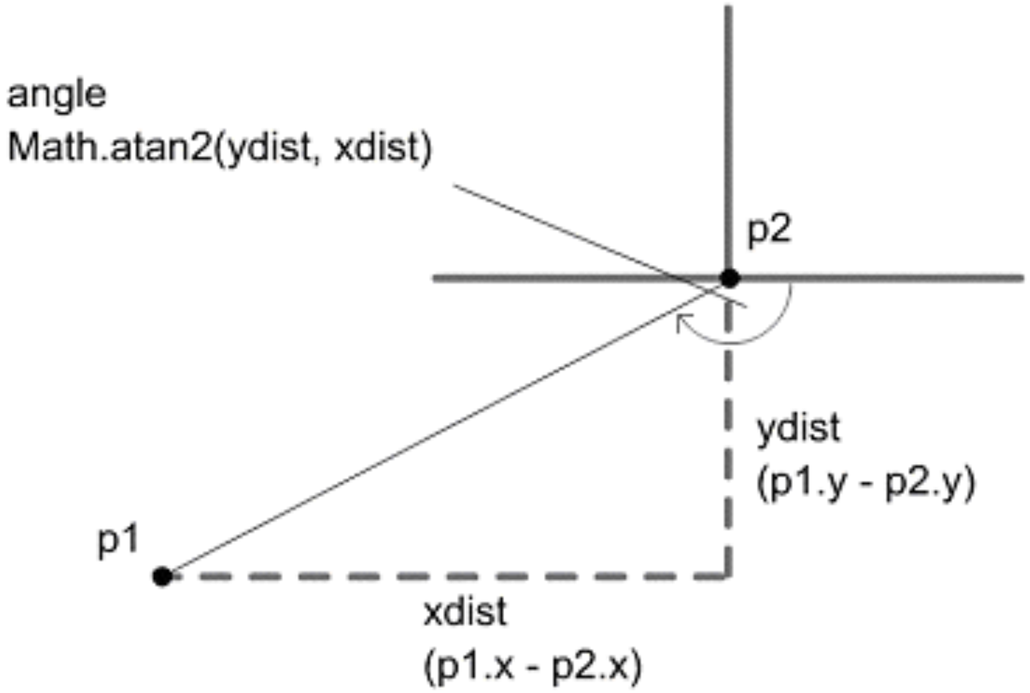
$$\text{Sin}\,\theta = \frac{\text{Opposite}}{\text{Hypotenuse}}$$



$$\text{Cos}\,\theta = \frac{\text{Adjacent}}{\text{Hypotenuse}}$$



$$\text{Tan}\,\theta = \frac{\text{Opposite}}{\text{Adjacent}}$$



Atan2?!

In a variety of computer languages, the function atan2 is the arctangent function with two arguments. For any real number (e.g., floating point) arguments x and y not both equal to zero, atan2(y, x) is the angle in radians between the positive x-axis of a plane and the point given by the coordinates (x, y) on it.

https://en.wikipedia.org/wiki/Atan2

```
gl::Texture::create( loadImage
( loadAsset( "sample.jpg" ) ) );
```

Cinder namespace. Check Cinder documentation for namespaces and functions: https://libcinder.org/docs/

gl::Texture::create( );

Create a texture;

loadImage()

Returns a image source.

loadAsset("sample.jpg");

Load an asset in the /asset folder of your project.

```
try{
    //do something
} catch(exception& e){
    //tell me what's wrong
}
```

Try catch is a great method to keep your app from crashing if it could not successfully perform a task like loading a texture. The program will not break if your //do something has failed.

```
while(iter.line()){
    while(iter.pixel()){
        //do something
    }}
```

Nested for loop to iterate through every pixels.

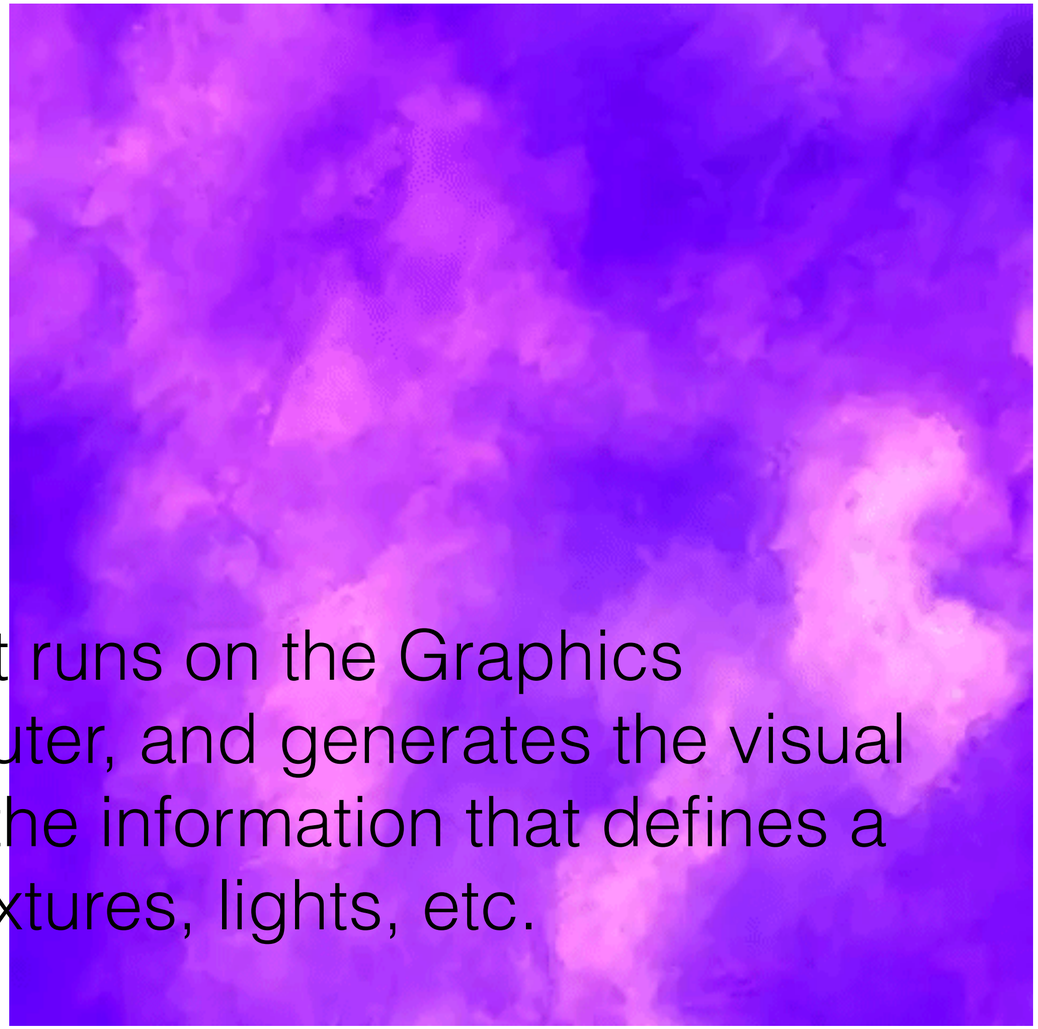
Shader

a shader is basically a program that runs on the Graphics Processing Unit (GPU) of the computer, and generates the visual output we see on the screen given the information that defines a 2D or 3D scene: vertices, colors, textures, lights, etc.

https://processing.org/tutorials/pshader/

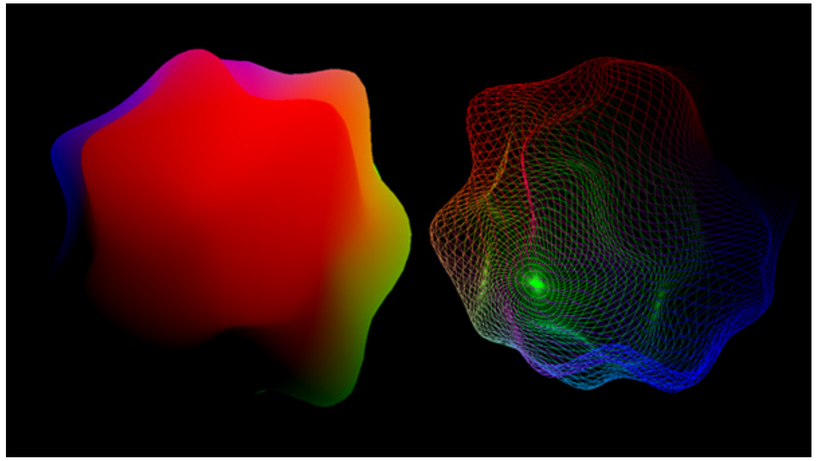
Shader

a shader is basically a program that runs on the Graphics Processing Unit (GPU) of the computer, and generates the visual output we see on the screen given the information that defines a 2D or 3D scene: vertices, colors, textures, lights, etc.



https://processing.org/tutorials/pshader/
http://jamesaproctor.com/

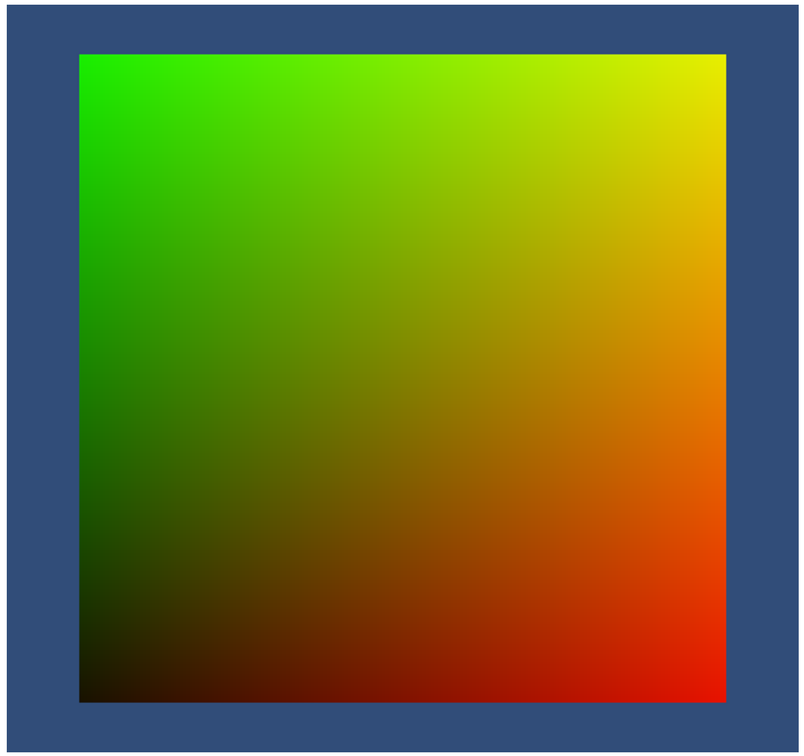
Vertex Shader



The Vertex Shader is the programmable Shader stage in the rendering pipeline that handles the processing of individual vertices.

https://processing.org/tutorials/pshader/

Fragment Shader



A Fragment Shader is the Shader stage that will processes a Fragment generated by the Rasterization into a set of colors and a single depth value.

https://processing.org/tutorials/pshader/

Output color is r,g,b,a;

```
gl_FragColor = vec4(1.0,1.0,1.0,1.0);
```

At early stage, you will deal more with fragment shaders.

https://processing.org/tutorials/pshader/

Go to

https://thebookofshaders.com/

If you want to learn more about shader stuff.

Homework:

1. Manipulate an image Interactively (color alternation, distortion, displacement)

2. (Bonus) Read Book of Shaders, try to manipulate images using Shaders.

3. Make sure all your codes have custom functions

Due: Feb 7 Tue
Upload a video demo of each project to our slack channel.
Keep the code for later integration into your GitHub repo.