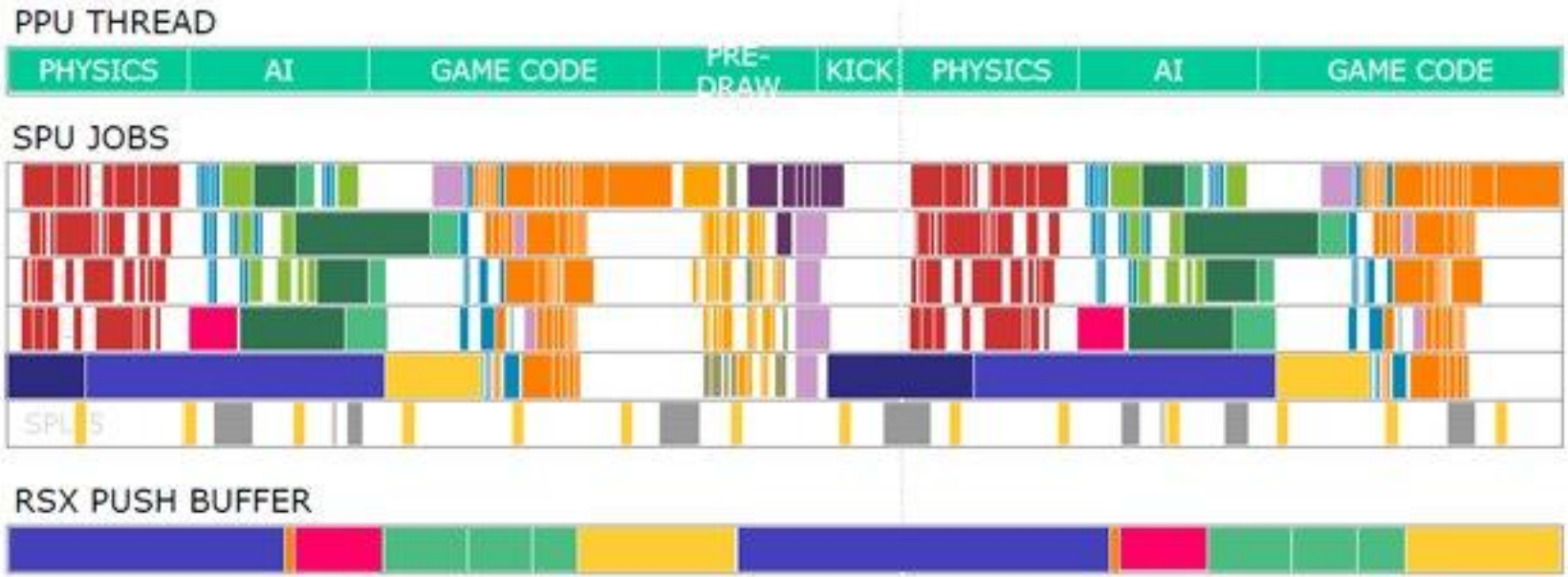
Advanced Games

Parallelism 2
Where's the data?

PS3 Engine


PPU THREAD
PHYSICS
AI
GAME CODE
PRE-DRAW
KICK
PHYSICS
AI
GAME CODE
SPU JOBS
RSX PUSH BUFFER

Interleaved Frames

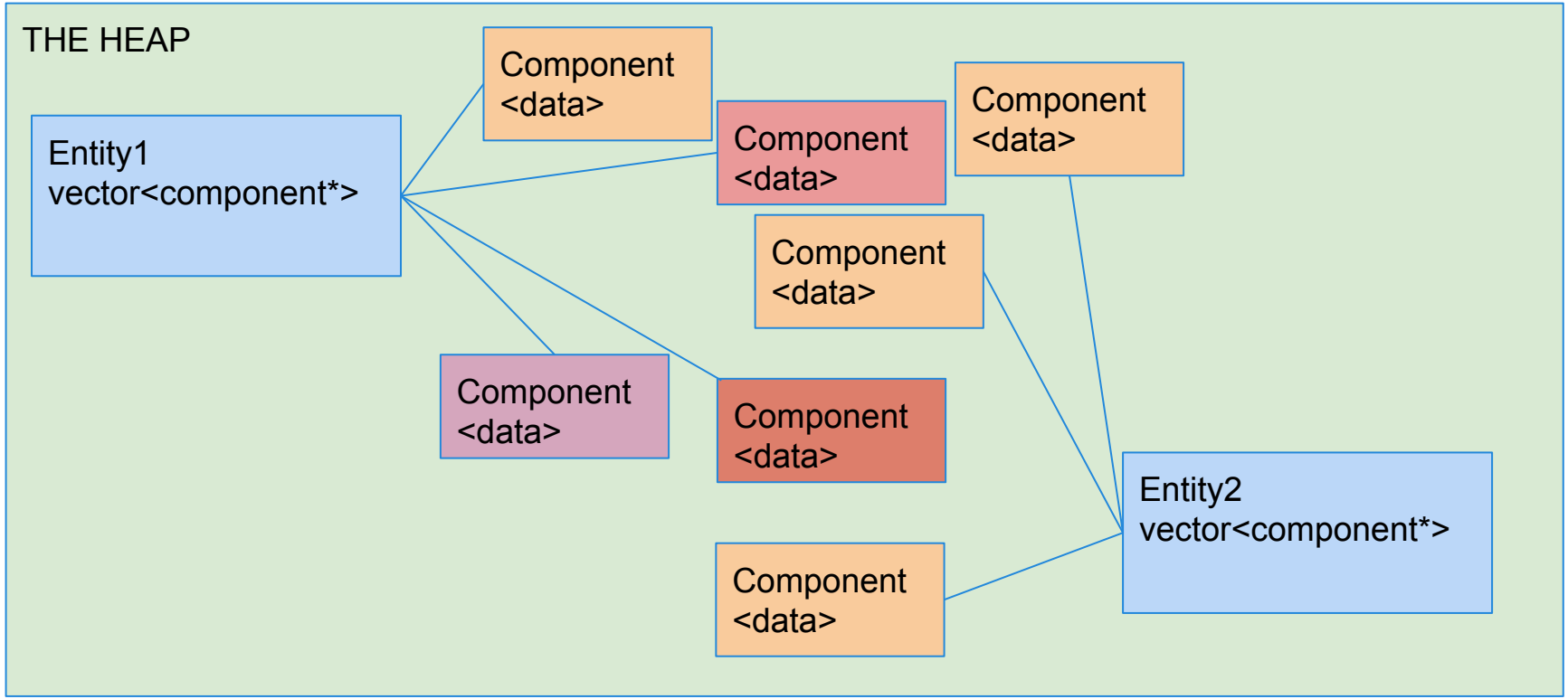
PPU	Update		Network	Update		Network	Update		Network	Update	
SPU		Physics	Sound	Lighting	Physics	Sound	Light	Physics	Sound	Lighting	Physics
SPU		Physics		Lighting	Physics		Light	Physics		Lighting	Physics
RSX			Render		Output	Render		Output	Render		Output
					Frame 1 Lag: 5			Frame 2 Lag: 3			Frame 3 Lag: 3

Awesomenauts

The threading scheme in Awesomenauts. While one thread updates the game, the other thread renders the *previous* frame. At the end of every frame there is a moment where all objects are copied from the gameplay thread to the render thread. The question is how to make that copying period very short.

Where's the Data?

ent.components.push(new Component());

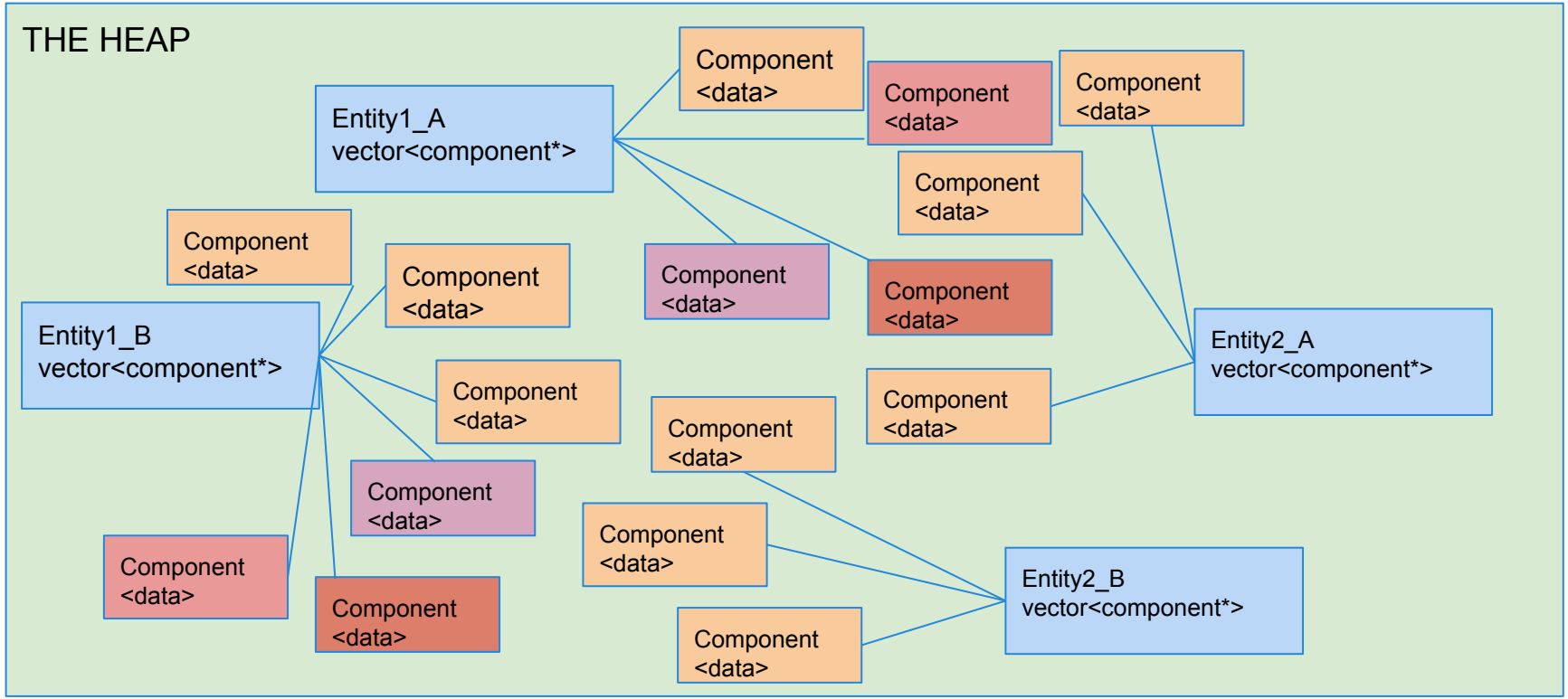


Where’s the Data?

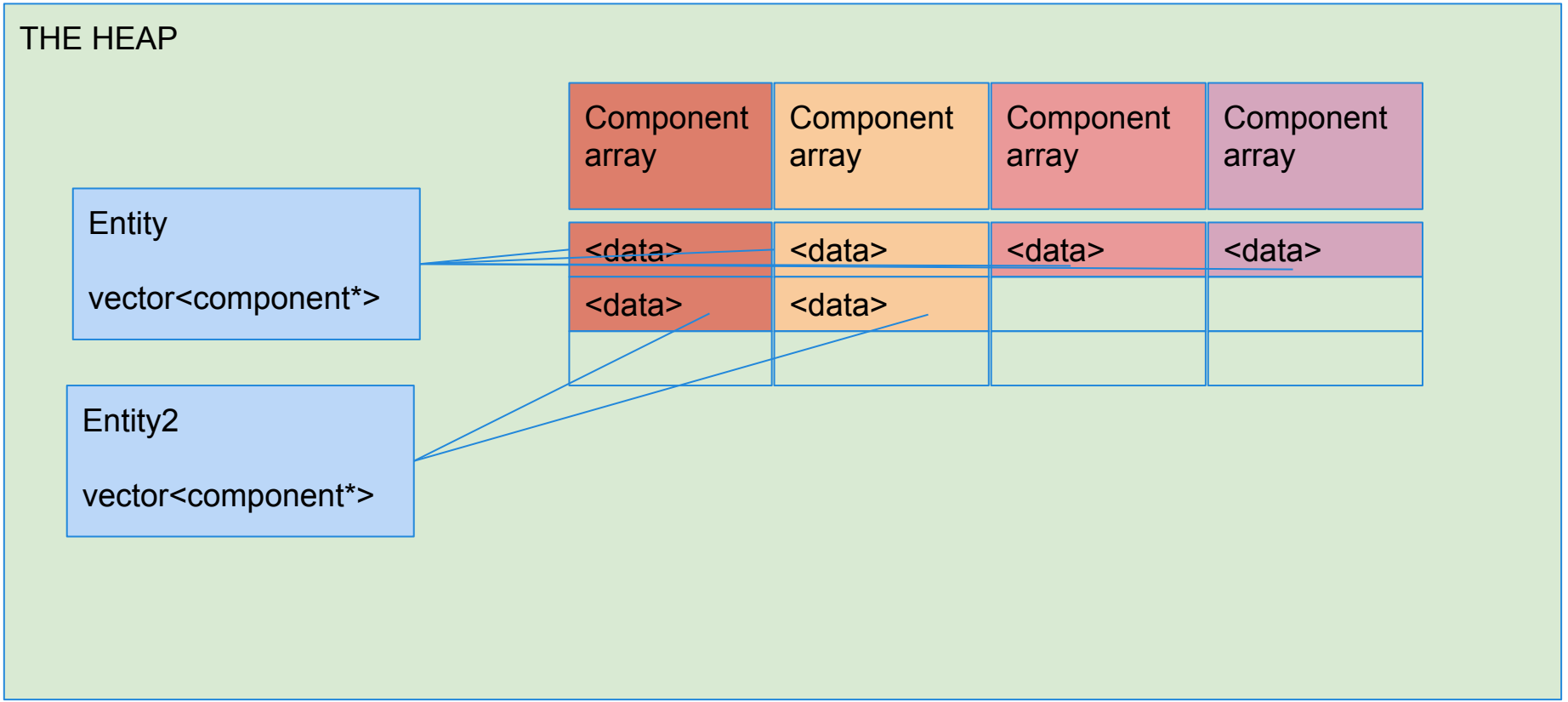
```
entities_A;
entities_B;
bool frame_count;
void Update(){
  if(frame_count){
    entities_A = deepCopy(entities_B);
    for(auto e : entities_A);
            e.update();
  }else{
     entities_B = deepCopy(entities_A);
     for(auto e : entities_B);
            e.update();
  }
}
```

Where's the Data?

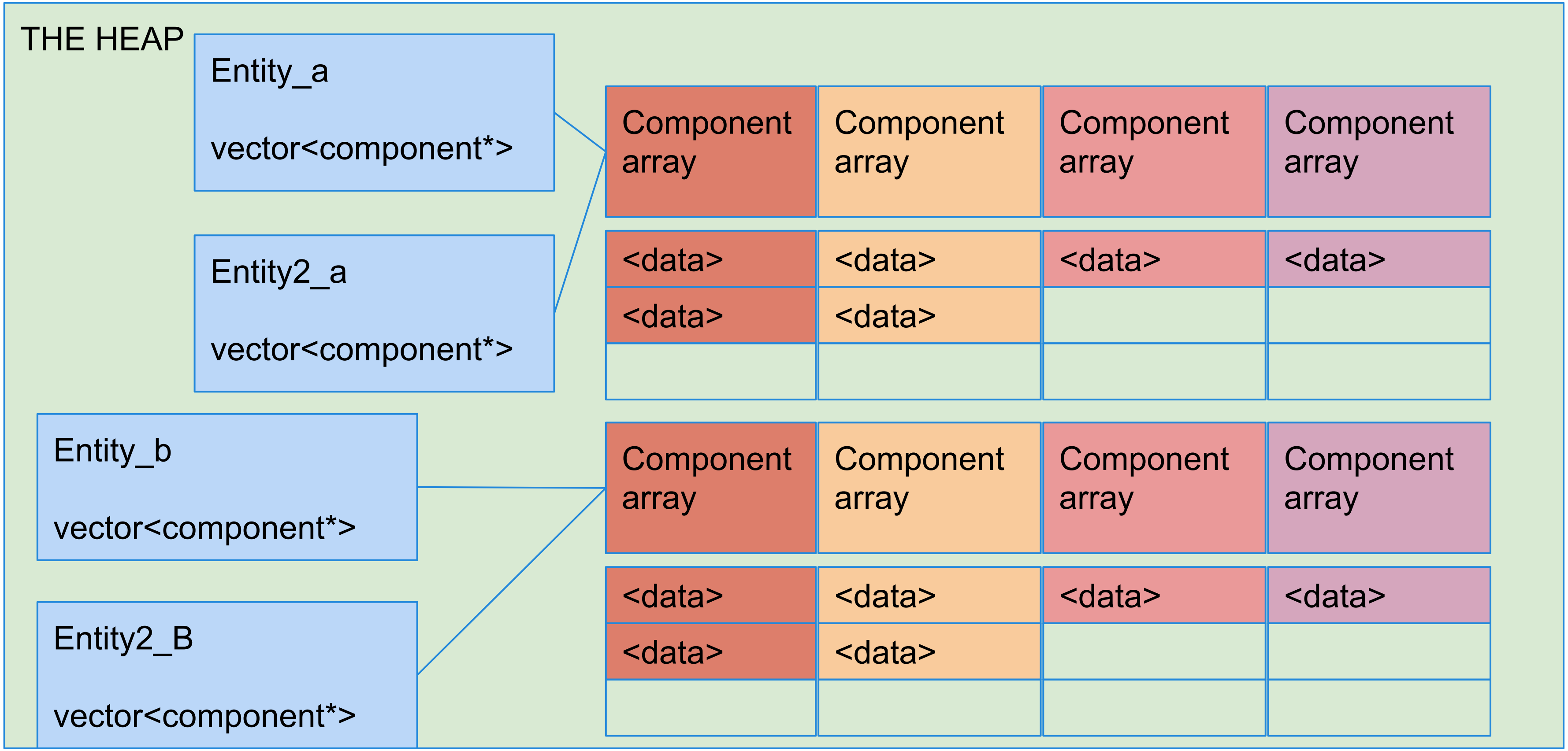
ent.components.push(new Component());



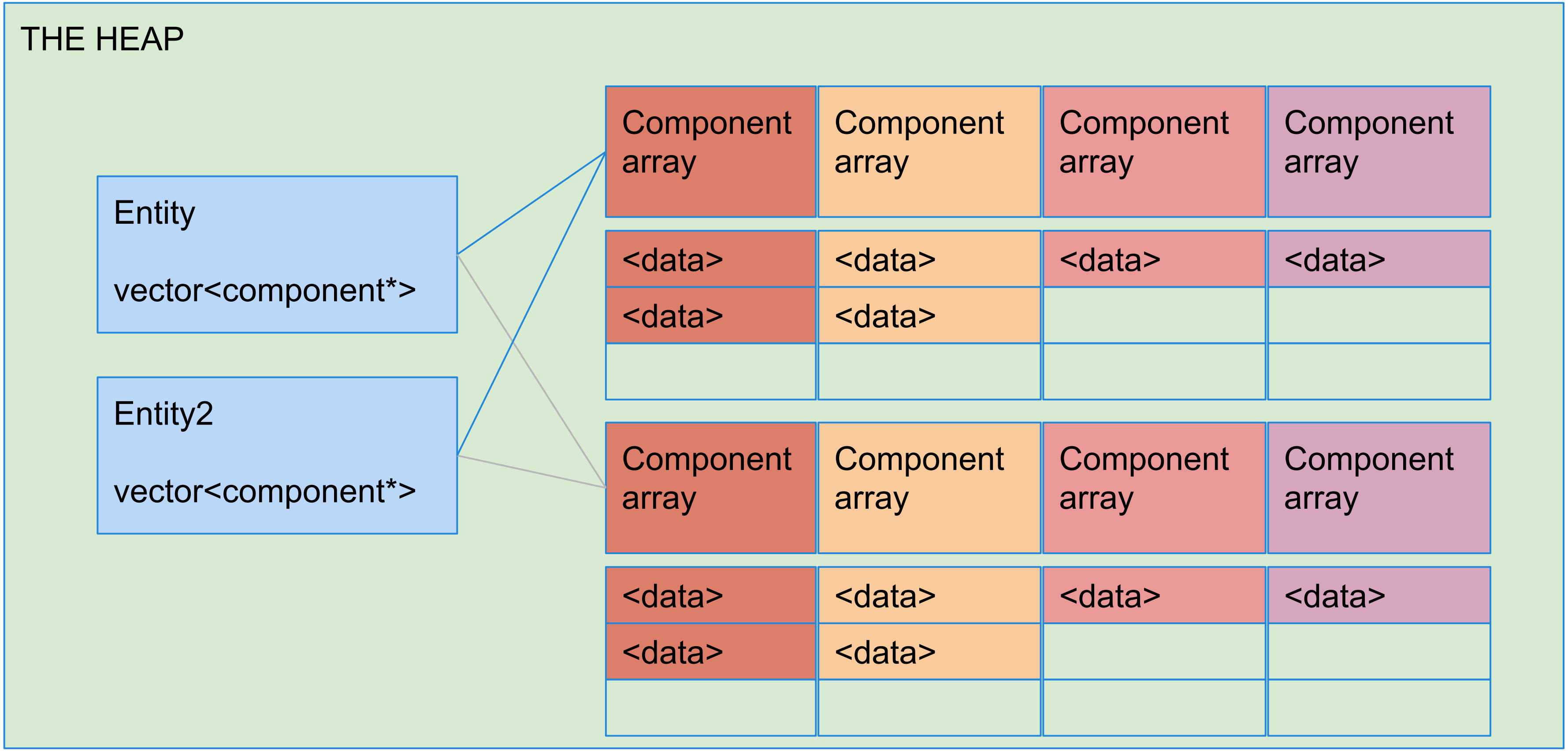
Where's the Data?



Where’s the Data?



Where's the Data?



Making OpenGL Multithreaded

Render Jobs

OpenGL is single threaded.

How do we Render as a job?

How do we feed the “Render worker?”

Rendering all the things

How do *you* render with ECM?

```
class Entity{
  std::vector<components*> cmpts;

  vec3 position;
  vec3 rotation;
}


class Model{
 GlUint vao;
 GlUint index_buffer;
 ...
}
```

```
class render_component : public component{
  Model* model;
  string colour;
  bool isVisible;
  GLuint shader;

  void Render(){
    glBind(shader)
    glBind(model->vao, textures, index_buffer)
    generateMVP();
    glSetUniforms(...)
    glEnable(...)
    glDrawElements(...)
    glDisable(...)
  }
}
```

Isolating Render Data

What do you need to render something?

- Shader
- Uniform data
  - MVP
  - Misc
- VAO
- Index Buffer
- Draw Mode
- Textures

```
struct render_data{
        int shader;
        int vao;
        int index_buffer;
        int draw_type;
        vector<Uniform_data> uniforms;
        vector<Texture_data> textures;
}
```

Ordering Render Data

All this assumes your data is already on the GPU!

```
struct render_data{
        int shader;
        int vao;
        int index_buffer;
        int draw_type;
        vector<Uniform_data> uniforms;
        vector<Texture_data> textures;
}
```

Cost of Changing: Arbitrary $ Scale

1. Shader = $10000
2. Textures = $100
3. Uniforms = $80
4. VAO / IB / DrawType* = $5

*Instanced rendering can make this even cheaper