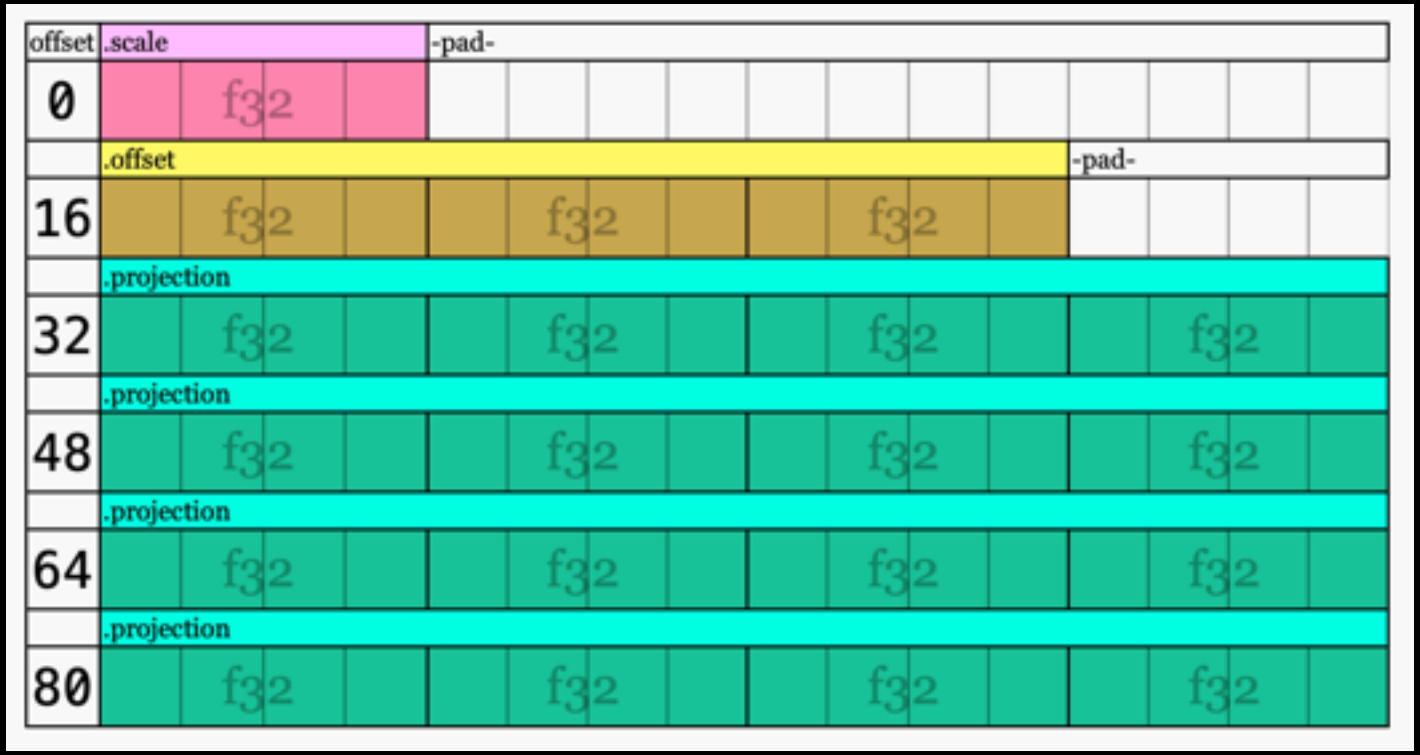
GPU Programming and Visualization

Graduate Course
Fall 2024

WebGPU

Alignment

```
struct Ex2 {
  scale: f32,
  offset: vec3f,
  projection: mat4x4f,
};
```

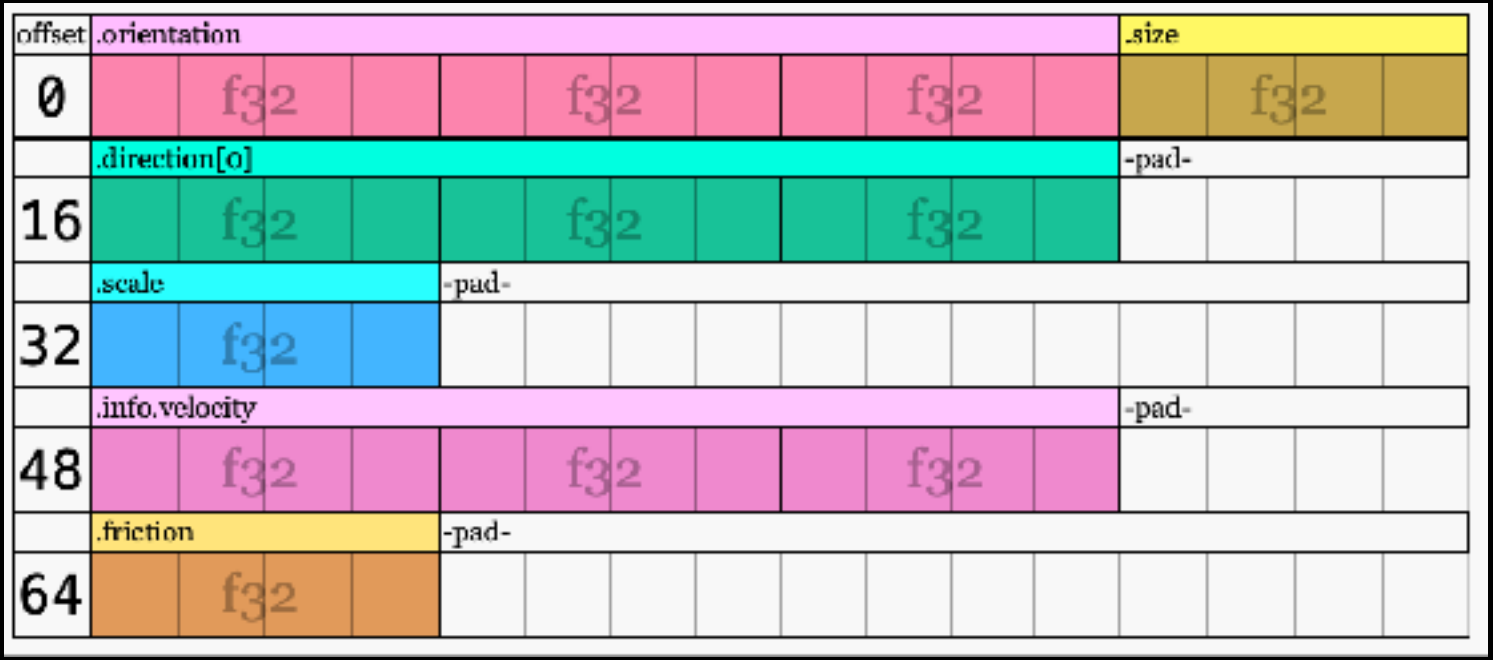


WebGPU

Alignment

```
struct Ex4a {
  velocity: vec3f,
};

struct Ex4 {
  orientation: vec3f,
  size: f32,
  direction: array<vec3f, 1>,
  scale: f32,
  info: Ex4a,
  friction: f32,
};
```



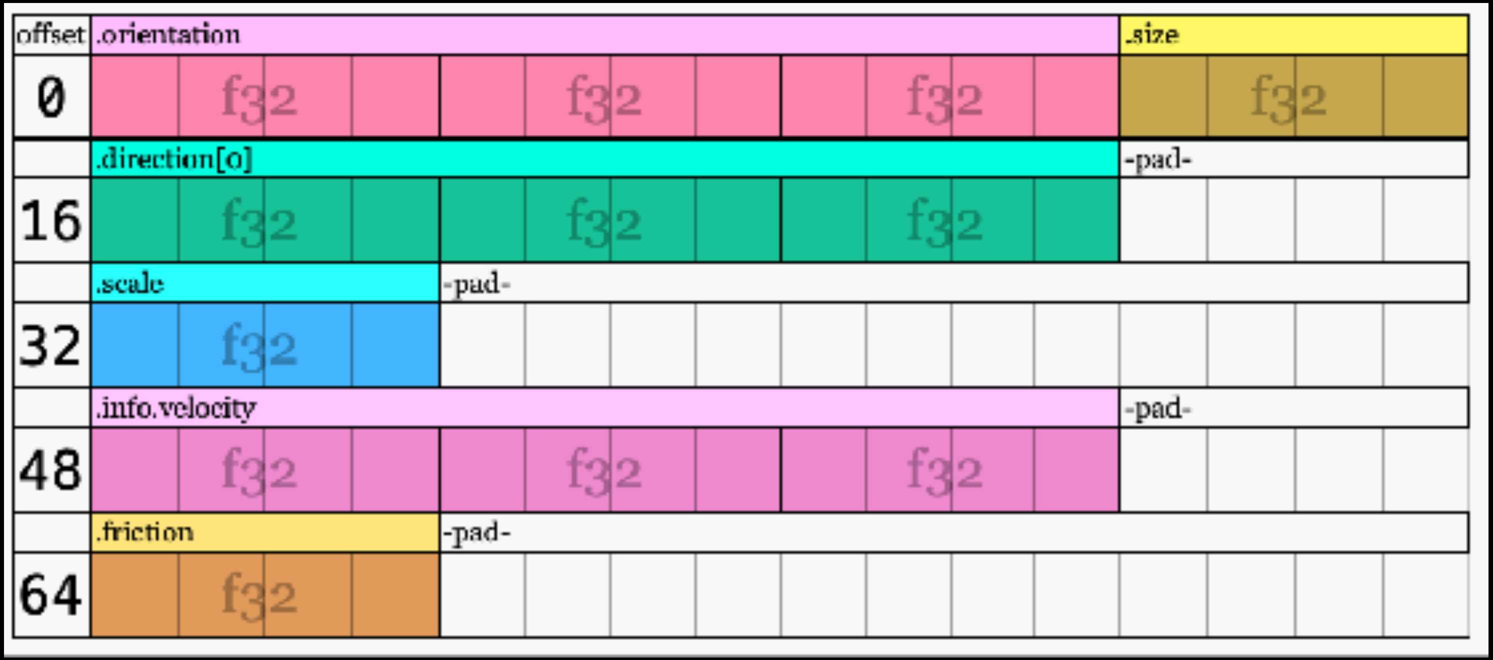
offset	.orientation			.size
0	f32	f32	f32	f32
	.direction[0]			-pad-
16	f32	f32	f32	
	.scale	-pad-		
32	f32			
	.info.velocity			-pad-
48	f32	f32	f32	
	.friction	-pad-		
64	f32			

WebGPU

Alignment

```
struct Ex4a {
  velocity: vec3f,
};

struct Ex4 {
  orientation: vec3f,
  size: f32,
  direction: array<vec3f, 1>,
  scale: f32,
  info: Ex4a,
  friction: f32,
};
```



WebGPU

Task 1: Add compute pass to render()

```
const computeEncoder = device.createCommandEncoder();
const computePass = computeEncoder.beginComputePass();
computePass.setBindGroup(0, computeBindGroup);
computePass.setPipeline(computePipeline);
computePass.dispatchWorkgroups(1);
computePass.end();
const computeCommandBuffer = computeEncoder.finish();
device.queue.submit([computeCommandBuffer]);
```

WebGPU

Task 2: Complete leapfrog code

```
 for (var pi = 0; pi<${Nparticles}; pi++){
  for (var pj = 0; pj<${Nparticles}; pj++){
    if (pi!=pj){
      let rel_pos = particles[pj].pos-particles[pi].pos;
      let softening = 0.01;
      let d =
         sqrt(rel_pos.x*rel_pos.x+rel_pos.y*rel_pos.y+rel_pos.z*rel_pos.z)
         +softening;
      particles[pi].vel += ${mass} * ${dt} * rel_pos/(d*d*d) ;
    }
  }
}
for (var pi = 0; pi<${Nparticles}; pi++){
  particles[pi].pos += 0.5*${dt} * particles[pi].vel;
}
```

WebGPU

Task 2: Complete leapfrog code

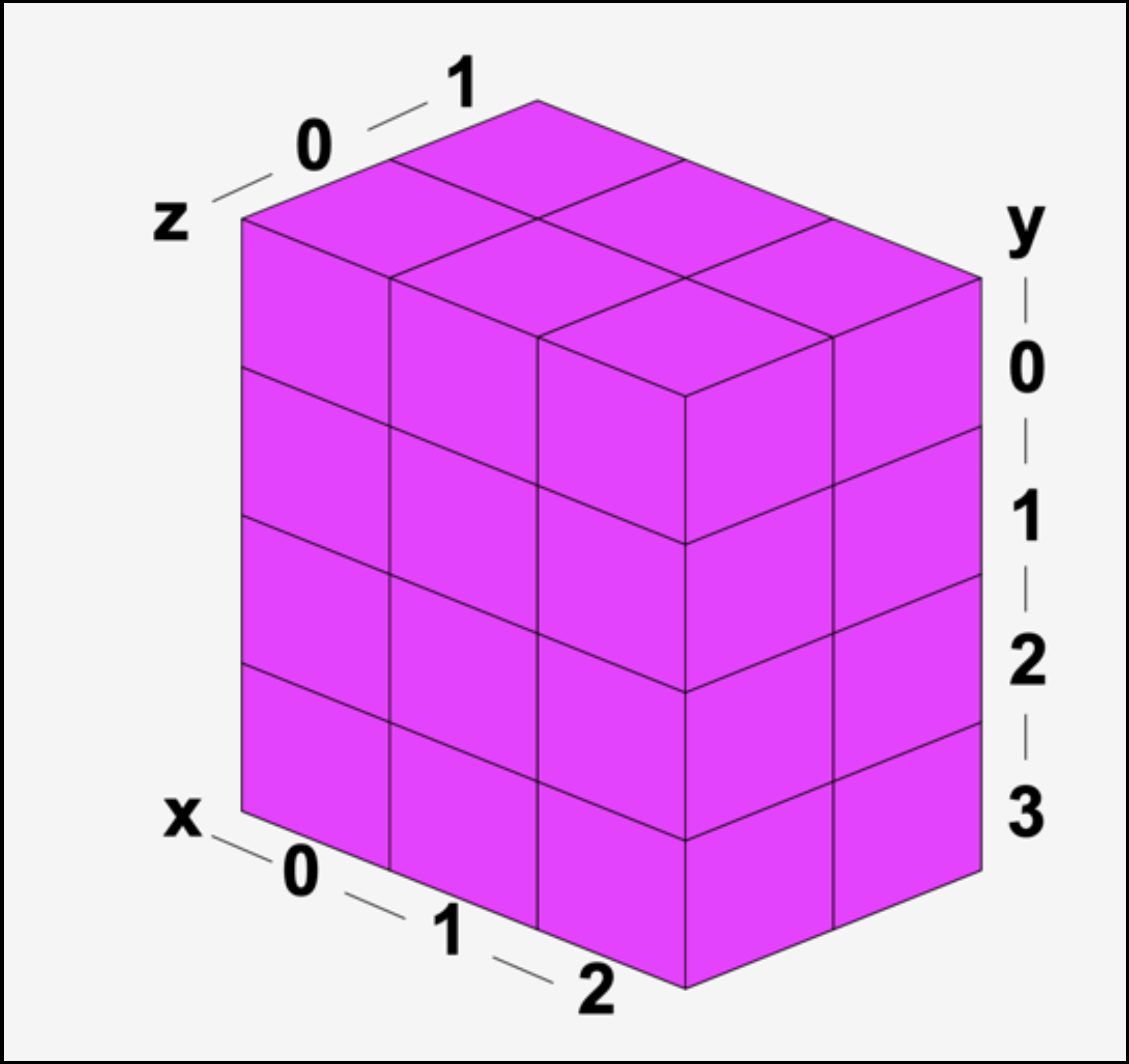
```
for (var pi = 0; pi<${Nparticles}; pi++){
 for (var pj = 0; pj<${Nparticles}; pj++){
   if (pi!=pj){
     let rel_pos = particles[pj].pos-particles[pi].pos;
     let softening = 0.01;
     let d =
        sqrt(rel_pos.x*rel_pos.x+rel_pos.y*rel_pos.y+rel_pos.z*rel_pos.z)
        +softening;
     particles[pi].vel += ${mass} * ${dt} * rel_pos/(d*d*d) ;
   }
 }
}
for (var pi = 0; pi<${Nparticles}; pi++){
 particles[pi].pos += 0.5*${dt} * particles[pi].vel;
}
```

WebGPU

Workgroups

local_invocation_id (3D)

local_invocation_index (1D)



WebGPU

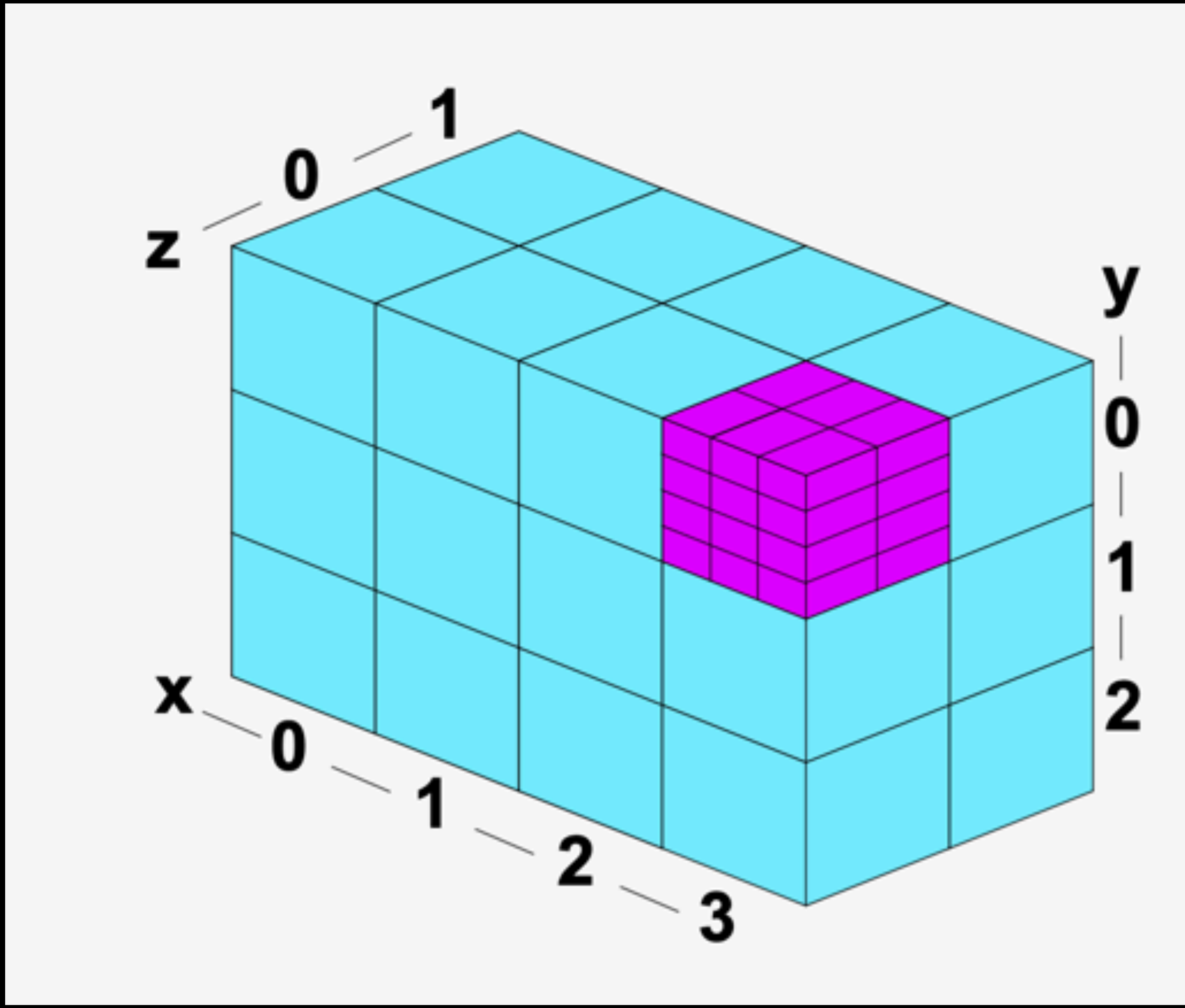
Workgroups

pass.dispatchWorkgroups(4, 3, 2)

workgroup_id

also:

global_invocation_id = workgroup_id * workgroup_size + local_invocation_id



WebGPU

Task 3: Use work groups

```
let workgroup_size = 256;
let Nworkgroups = Nparticles/workgroup_size;

@compute @workgroup_size(${workgroup_size}) fn step(

let pi = workgroup_id.x * ${workgroup_size} + local_invocation_index;
// ^^ global_invocation_index

// Remove loops
// 0 - > 0u
// Up particles to  2**13

computePass.dispatchWorkgroups(Nworkgroups);
```

WebGPU

WebGPU provides no guarantees about:

- Whether invocations from different workgroups execute concurrently. That is, you cannot assume more than one workgroup executes at a time.
- Whether, once invocations from a workgroup begin executing, that other workgroups are blocked from execution. That is, you cannot assume that only one workgroup executes at a time. While a workgroup is executing, the implementation may choose to concurrently execute other workgroups as well, or other queued but unblocked work.
- Whether invocations from one particular workgroup begin executing before the invocations of another workgroup. That is, you cannot assume that workgroups are launched in a particular order.

WebGPU

Task 4: Split pipeline in kick + drift

See extra code

WebGPU

Task 5: SIMD Optimizations

```
let pj = select(j+1, j, j<pi);


   vs


if (j!=pi) {

}
```

WebGPU

Task 6: Copy results to CPU

```
const resultBuffer = device.createBuffer({
  label: 'result buffer',
  size: initialConditions.byteLength,
  usage: GPUBufferUsage.MAP_READ | GPUBufferUsage.COPY_DST,
});

const encoder = device.createCommandEncoder();
encoder.copyBufferToBuffer(workBuffer, 0, resultBuffer, 0, resultBuffer.siz
const commandBuffer = encoder.finish();
device.queue.submit([commandBuffer]);

await resultBuffer.mapAsync(GPUMapMode.READ);
const result = new Float32Array(resultBuffer.getMappedRange().slice());
resultBuffer.unmap();
console.log('result', result);
```

WebGPU

Task 6: Draw faster stars red