INSTANCING AND OIT IN TOTAL WAR:THREE KINGDOMS

Tamas Rabel
Rendering Technical Director
Creative Assembly


Campaign is turn-based


DEVELOPMENT IN PROGRESS
LÜ BU PERISHES AT XIAPI
22
1728
Spring
35:52
Battle is RTS
@IntelSoftware
@IntelGraphics
@totalwar
@CAGames

AGENDA

Instancing pipeline

Worker threads

Sequential part

Results

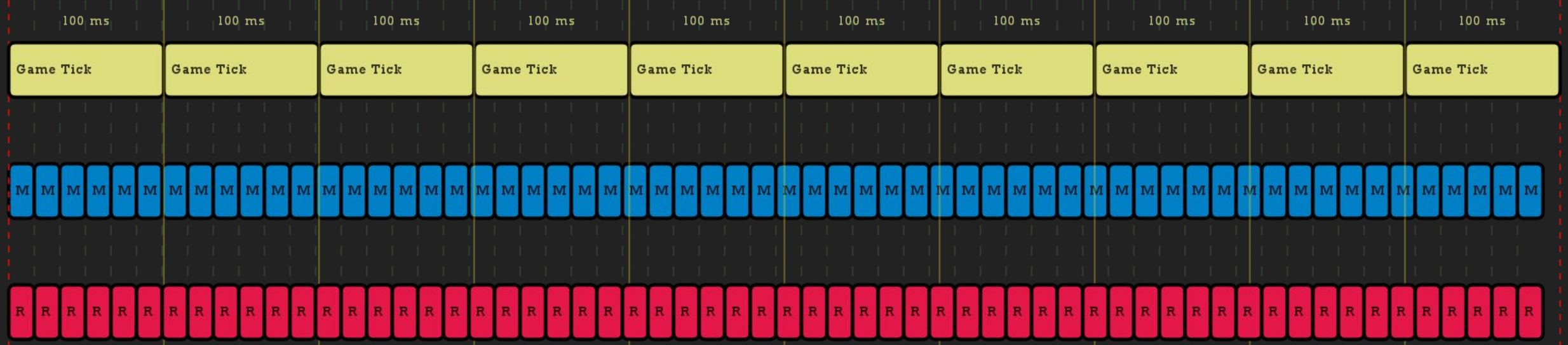
Order independent transparency for VFX

Different methods

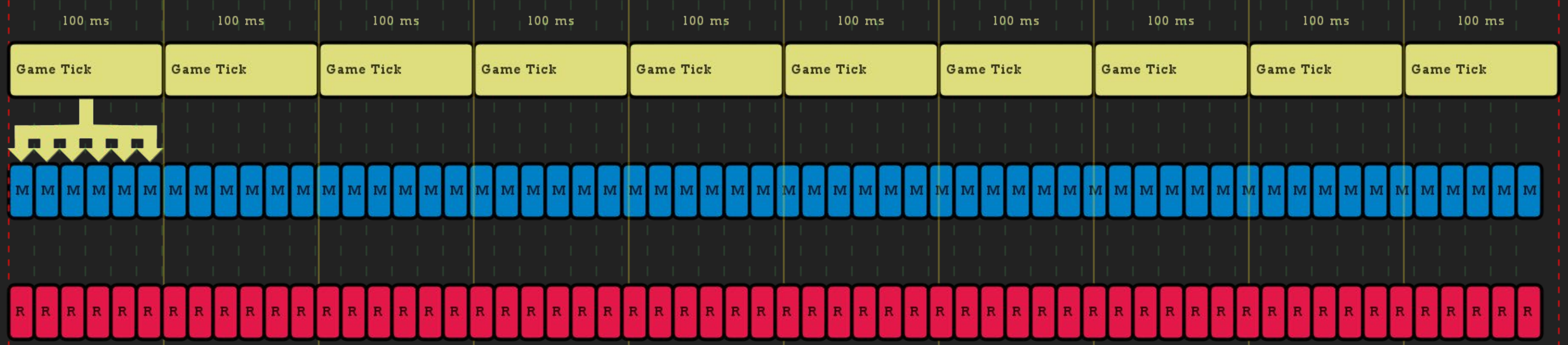
Results

100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms
Game Tick	Game Tick	Game Tick	Game Tick	Game Tick	Game Tick	Game Tick	Game Tick	Game Tick	Game Tick

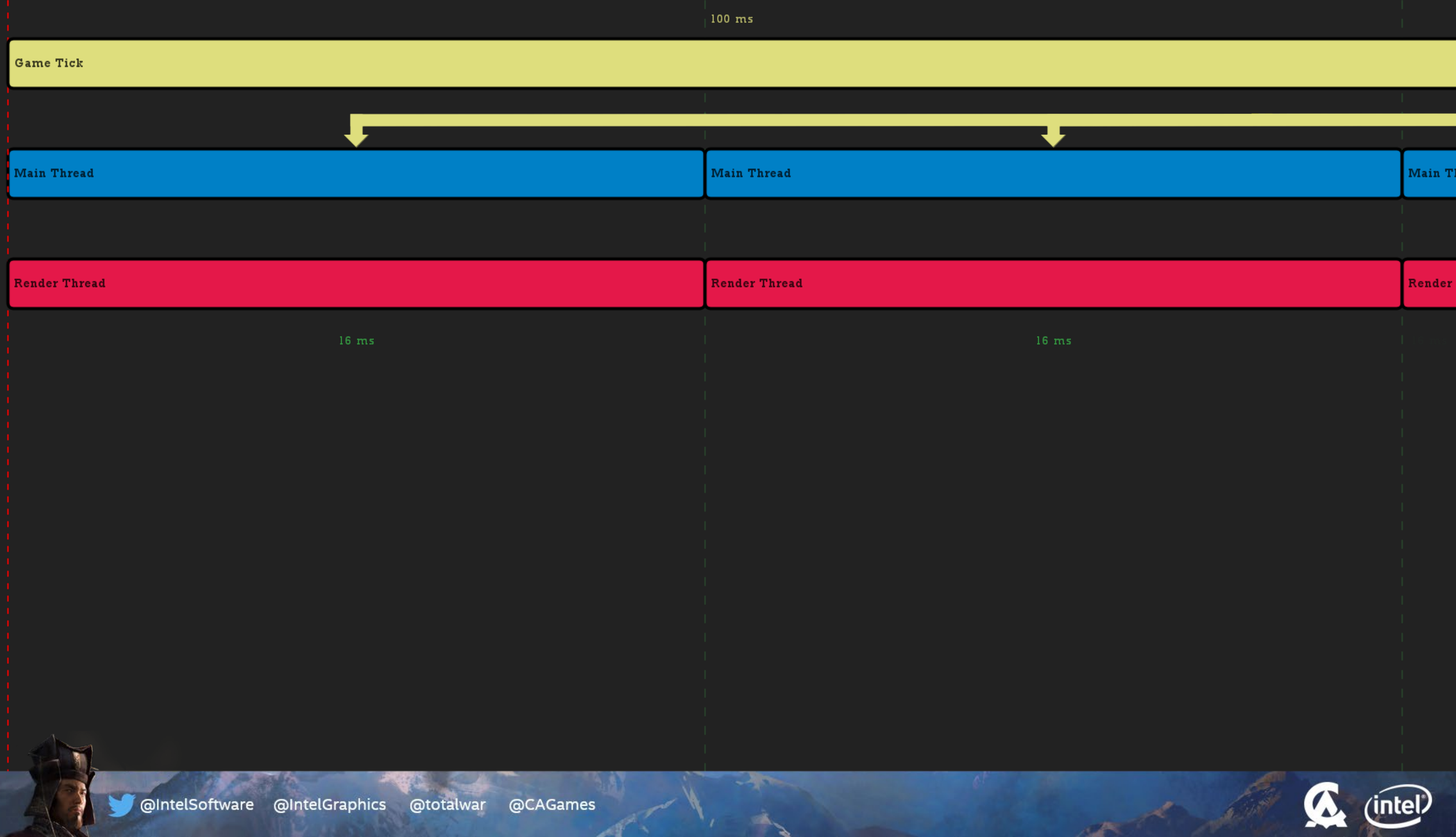
Game Tick thread is running ~10 fps



Main Thread and Render Thread are running in sync

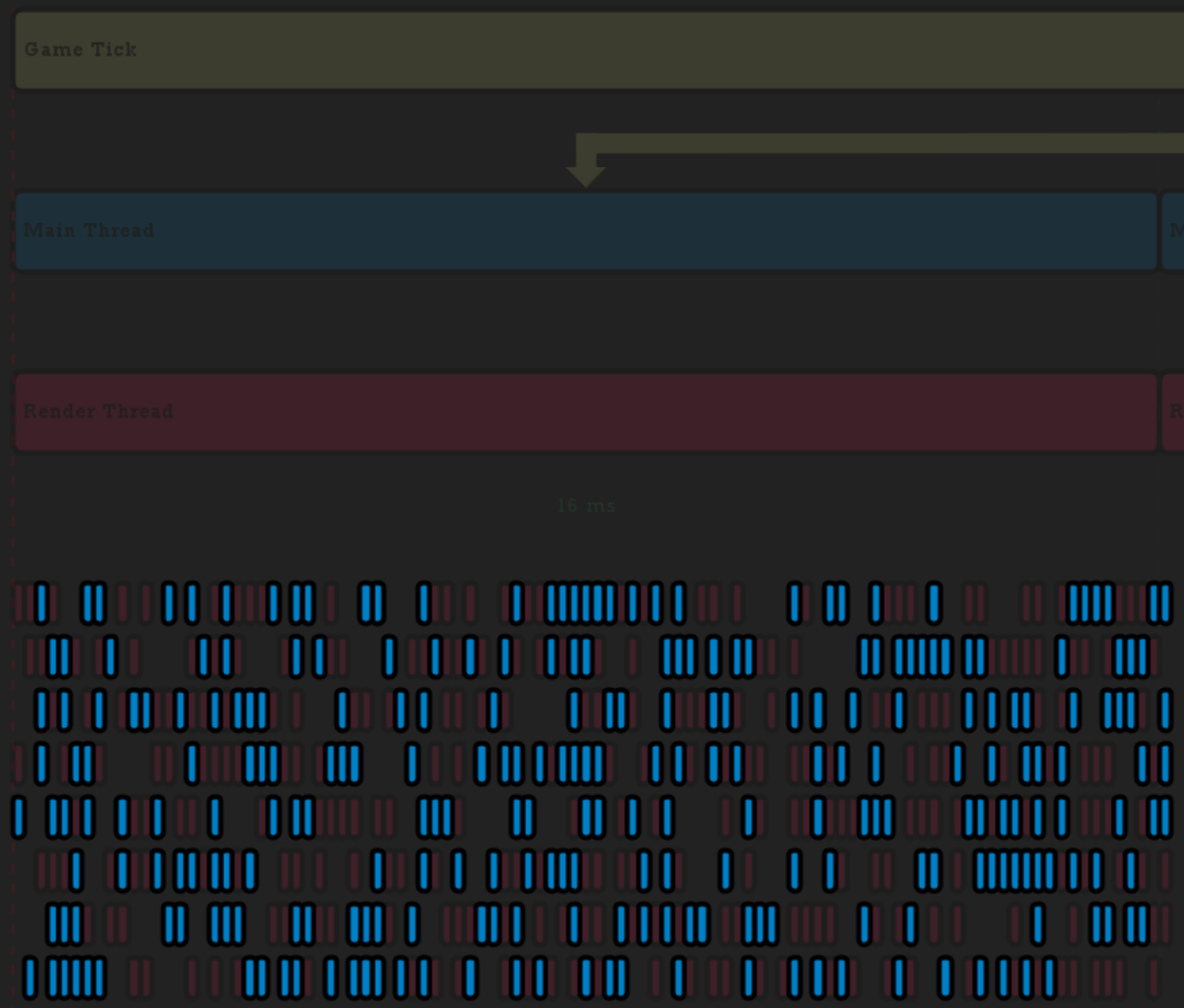


Game Tick builds game state and passes it to main thread

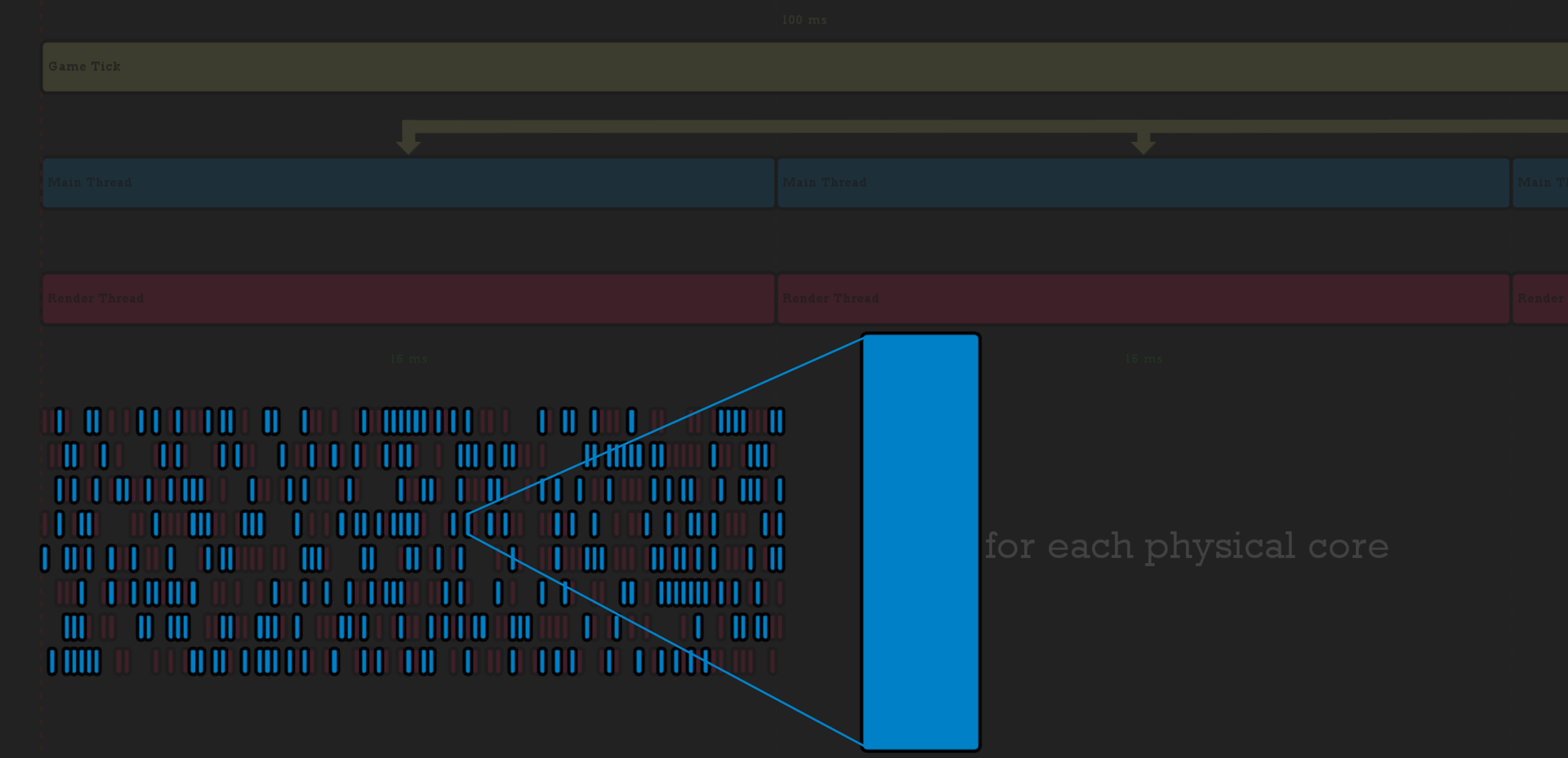

100 ms
Game Tick
Main Thread
Main Thread
Main Th
Render Thread
Render Thread
Render
16 ms
16 ms

Worker Threads

One for each physical core

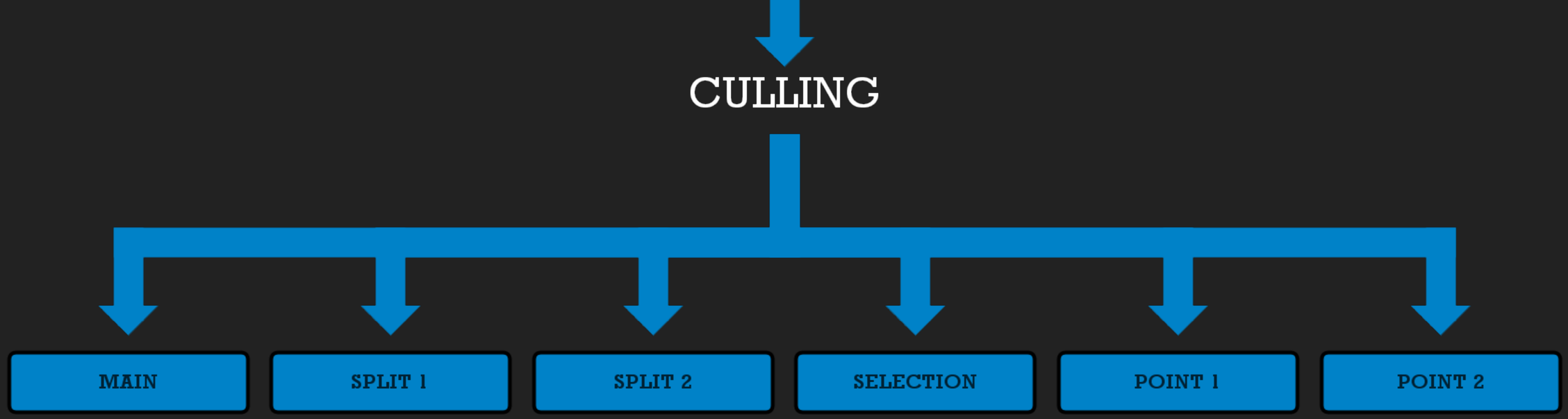


One for each physical core

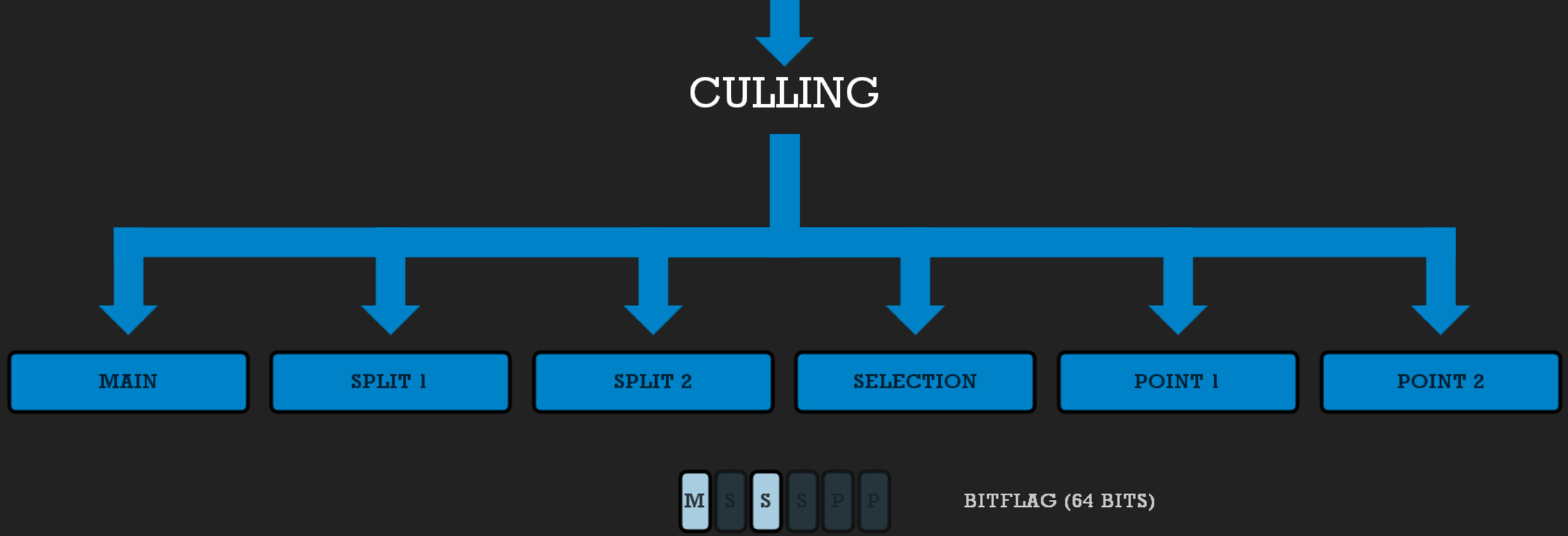

100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms
for each physical core

WORKER THREAD FOR MT


CULLING



We calculate intersection with all the frustums at once

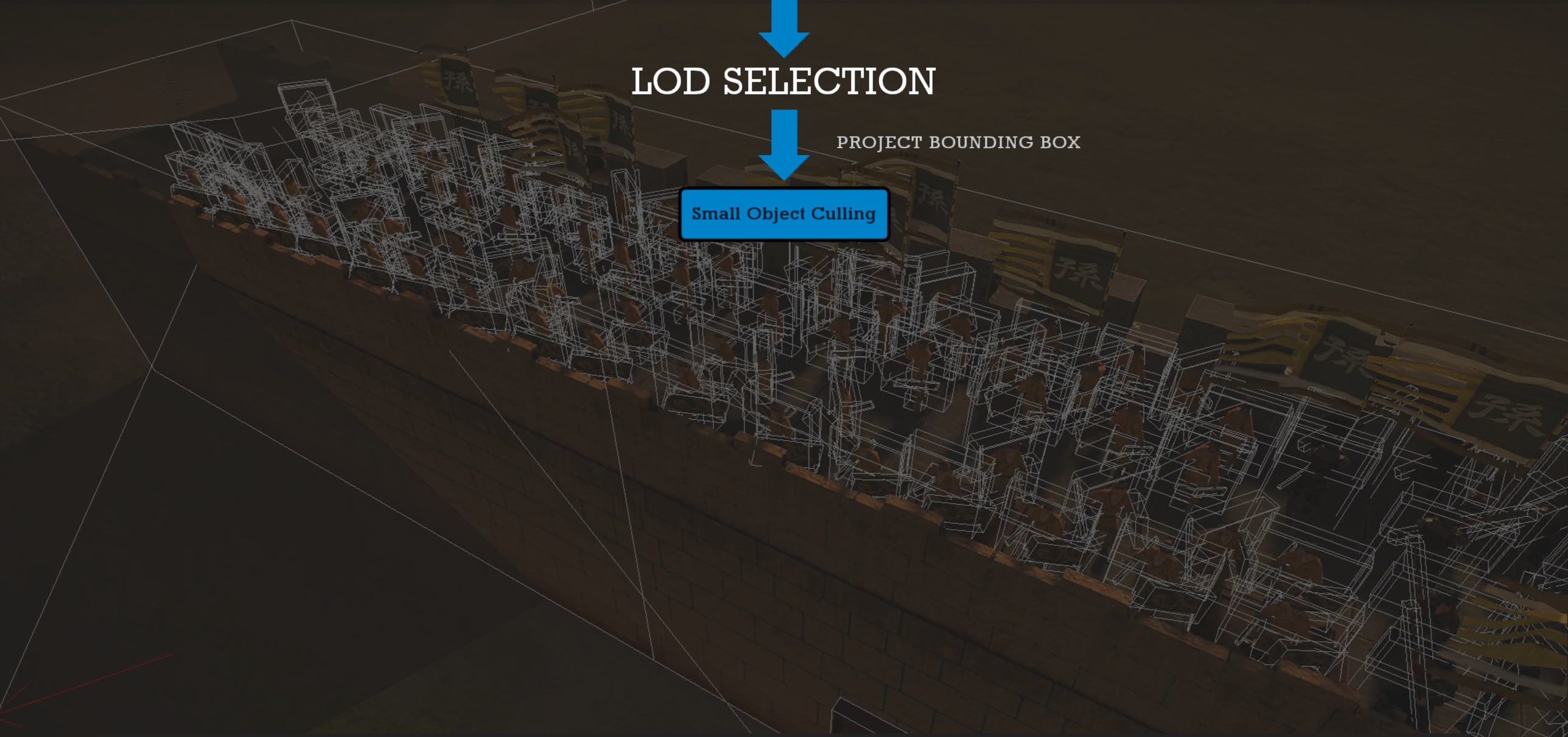


Intersection results are stored in a bitfield

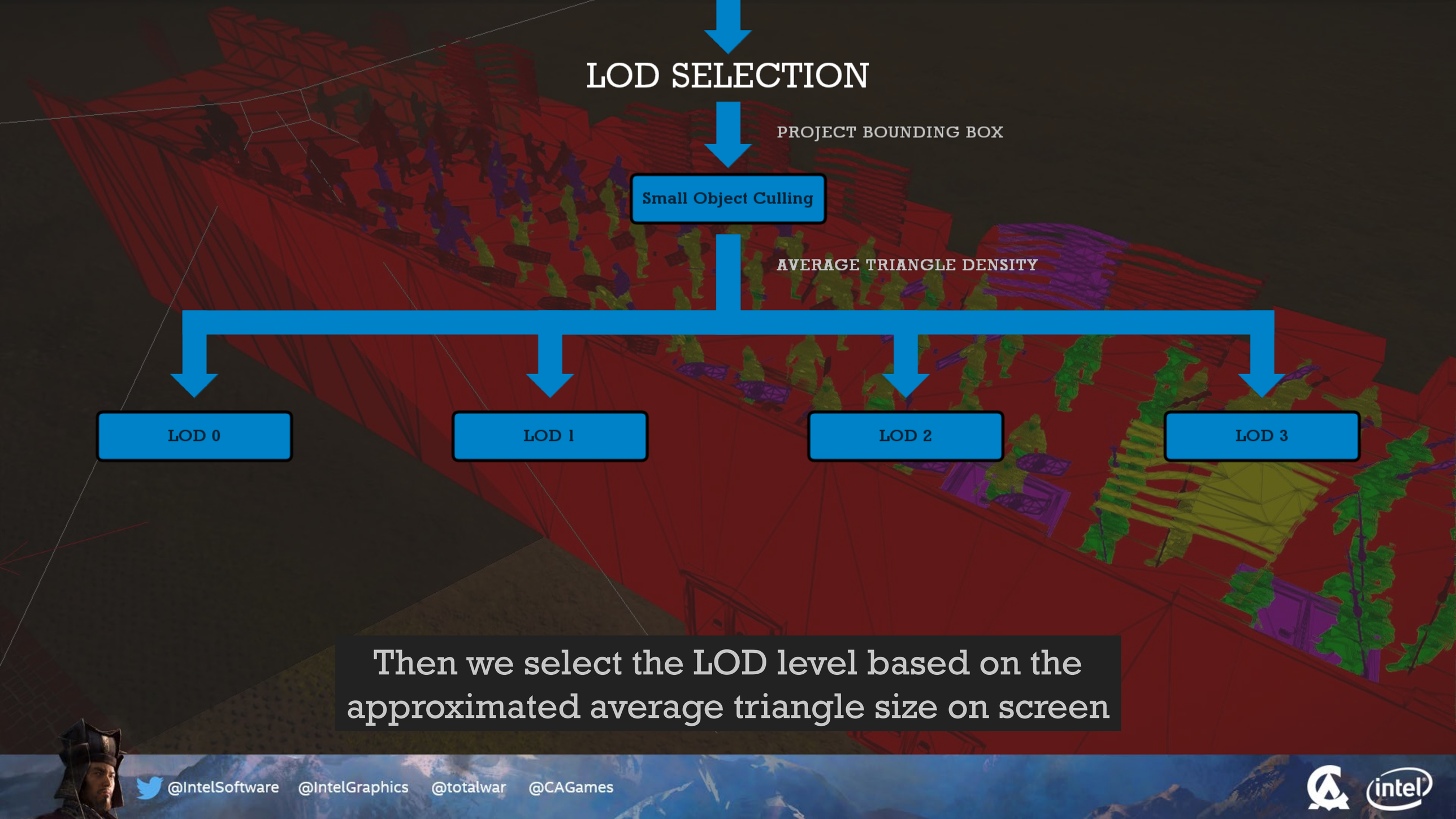
LOD SELECTION

PROJECT BOUNDING BOX

Small Object Culling

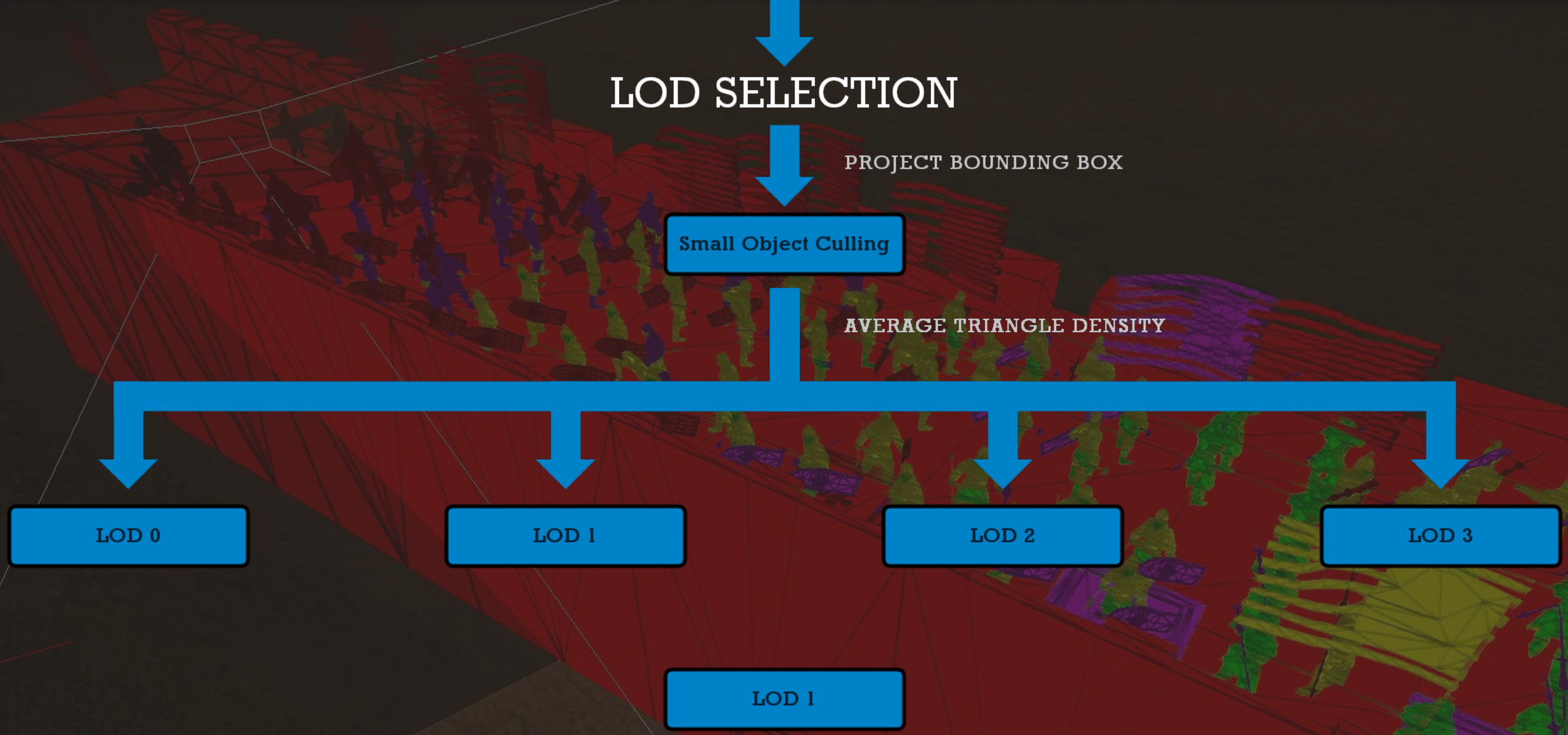


We calculate the area of the projected bounding box and cull small models



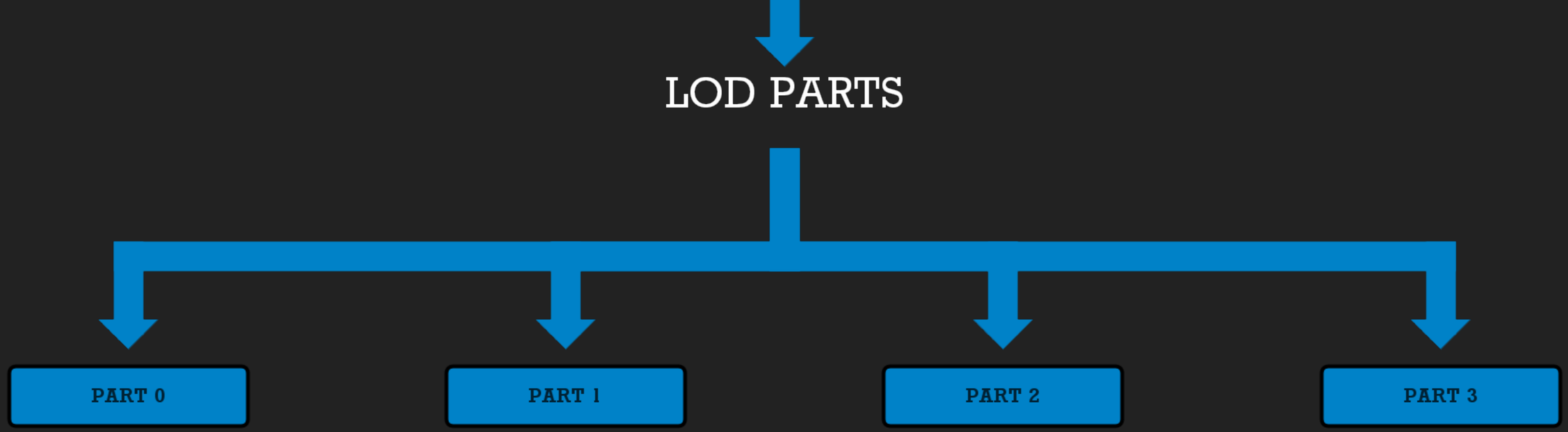
Then we select the LOD level based on the approximated average triangle size on screen

LOD SELECTION



Then we select the LOD level based on the approximated average triangle size on screen


LOD PARTS
@IntelSoftware
@IntelGraphics
@totalwar
@CAGames



Triangles with different materials are different parts

Part is what can be rendered with a single draw call

In our terminology a part is a MESH

LOD PARTS

3-5 parts on average plus weapons/shield

3 different variations on average

5*3 = 15 draw calls per unit

(Unit is a group of soldiers, players control units)

LOD PARTS

3-5 parts on average plus weapons/shield

3 different variations on average

Heads are using more variations

5*3 + 5 = 20 draw calls per unit

(Unit is a group of soldiers, players control units)

LOD PARTS

3-5 parts on average plus weapons/shield

3 different variations on average

Heads are using more variations

Each unit is using 2 lods on average

(5*3 + 5) * 2 = 40 draw calls per unit

(Unit is a group of soldiers, players control units)

LOD PARTS

3-5 parts on average plus weapons/shield

3 different variations on average

Heads are using more variations

Each unit is using 2 lods on average

Everything is rendered into the shadow map too

(5*3 + 5) * 2 * 2 = 80 draw calls per unit

(Unit is a group of soldiers, players control units)

LOD PARTS

3-5 parts on average plus weapons/shield

3 different variations on average

Heads are using more variations

Each unit is using 2 lods on average

Everything is rendered into the shadow map too

(5*3 + 5) * 2 * 2 = 80 draw calls per unit

(5*3 + 5) * 2 * 2 * 42 = 3360 draw calls (for ~5k soldiers)

(Unit is a group of soldiers, players control units)

LOD PARTS

Imposters are always 1 draw call

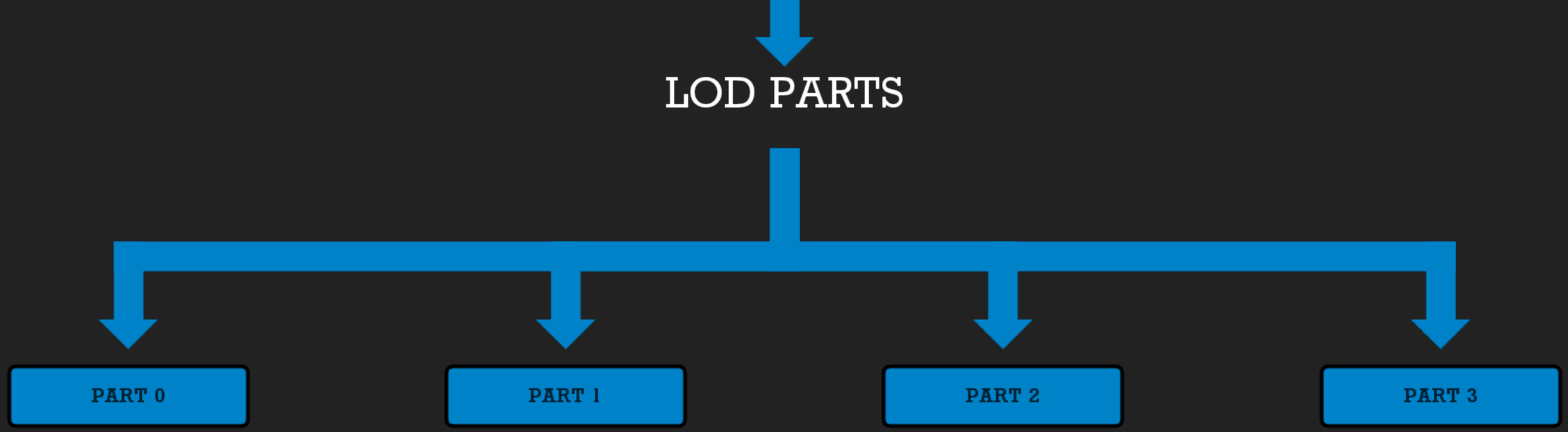
Imposters are super low poly (<200 tris)

LOD PARTS

Imposters are always 1 draw call

Imposters are super low poly (<200 tris)

Imposters also have two lod levels (<200 tris and <50 tris)



Triangles with different materials are different parts

Part is what can be rendered with a single draw call

In our terminology a parts is a MESH

PART 1

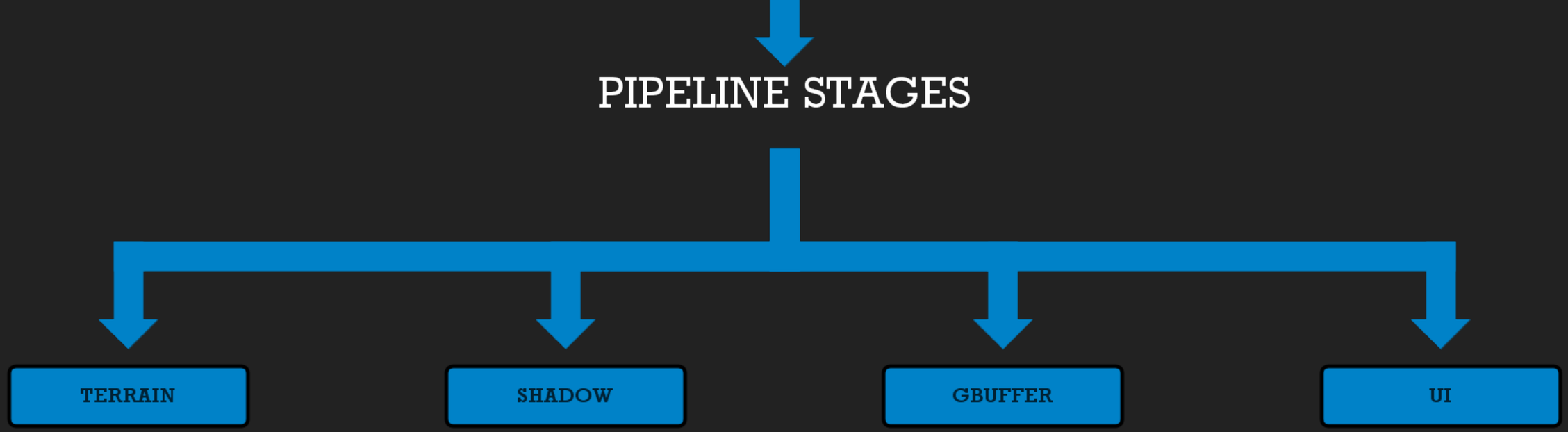
Next step is building instance data for the individual parts

INSTANCE DATA

Every mesh type has its own instance data format

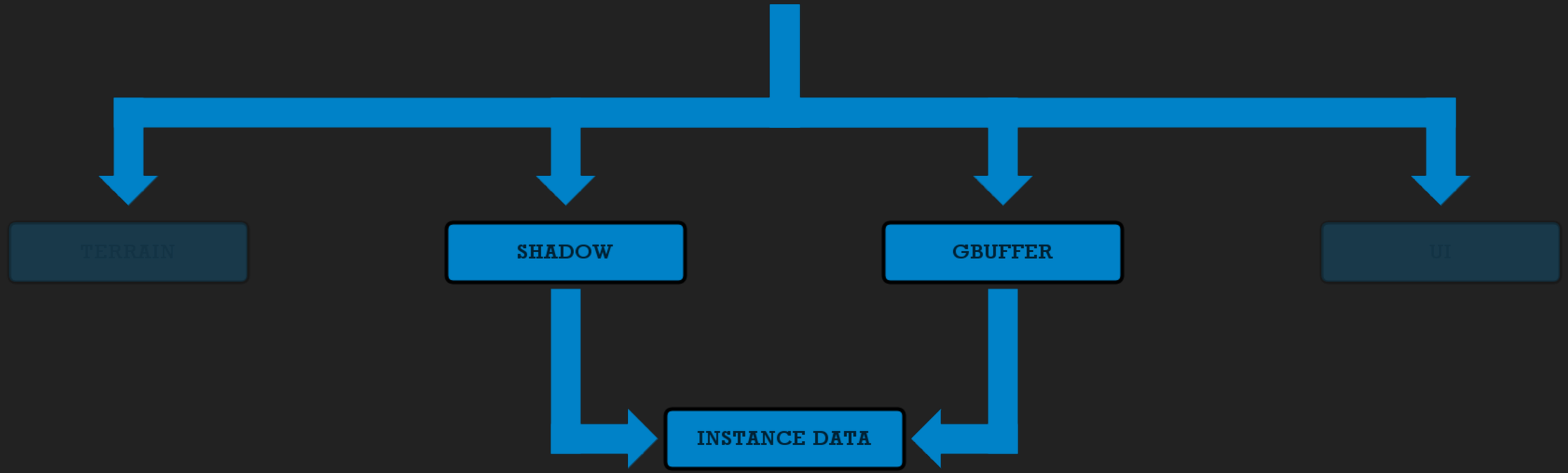
Different materials use the same instance data

This is what will be uploaded to the GPU



Based on the mesh and instance data and the material
we figure out which parts of the pipeline the mesh needs to be rendered to.

PIPELINE STAGES



Each pipeline stage uses the same instance data.

INSTANCE LISTS

TERRAIN

SHADOW

GBUFFER

UI

Every single mesh has a number of instance lists.
One instance list per pipeline stage.
The instance is added to all the relevant instance lists.

INSTANCE LISTS

TERRAIN	SHADOW	GBUFFER	UI
TERRAIN	SHADOW	GBUFFER	UI
TERRAIN	SHADOW	GBUFFER	UI
TERRAIN	SHADOW	GBUFFER	UI
TERRAIN	SHADOW	GBUFFER	UI

The instance lists are duplicated to every worker thread to ensure lockless access.

TERRAIN SHADOW GBUFFER UI

INSTANCE LISTS

TERRAIN	ERRAIN
TERRAIN	ERRAIN
TERRAIN	ERRAIN
TERRA	
TERRA	
TERRA	
TERRAIN	

SHADOW	IADOW
SHADOW	IADOW
SHADOW	IADOW

GBUFFER	BUFFER
GBUFFER	BUFFER
GBUFFER	BUFFER

UI	UI
UI	UI
UI	UI
UI	UI
UI	UI
UI	UI
UI	UI

And double buffered because while the render thread
is rendering the current frame, the main thread (and workers)
Is adding instances to the next frame's instance lists.

MESH LISTS

TERRAIN

RIGID

WEIGHTED

BILLBOARD

We have one mesh list per mesh type.
Every mesh with at least a single instance in the frame
will get added to exactly one mesh list matching its mesh type.

MESH LISTS

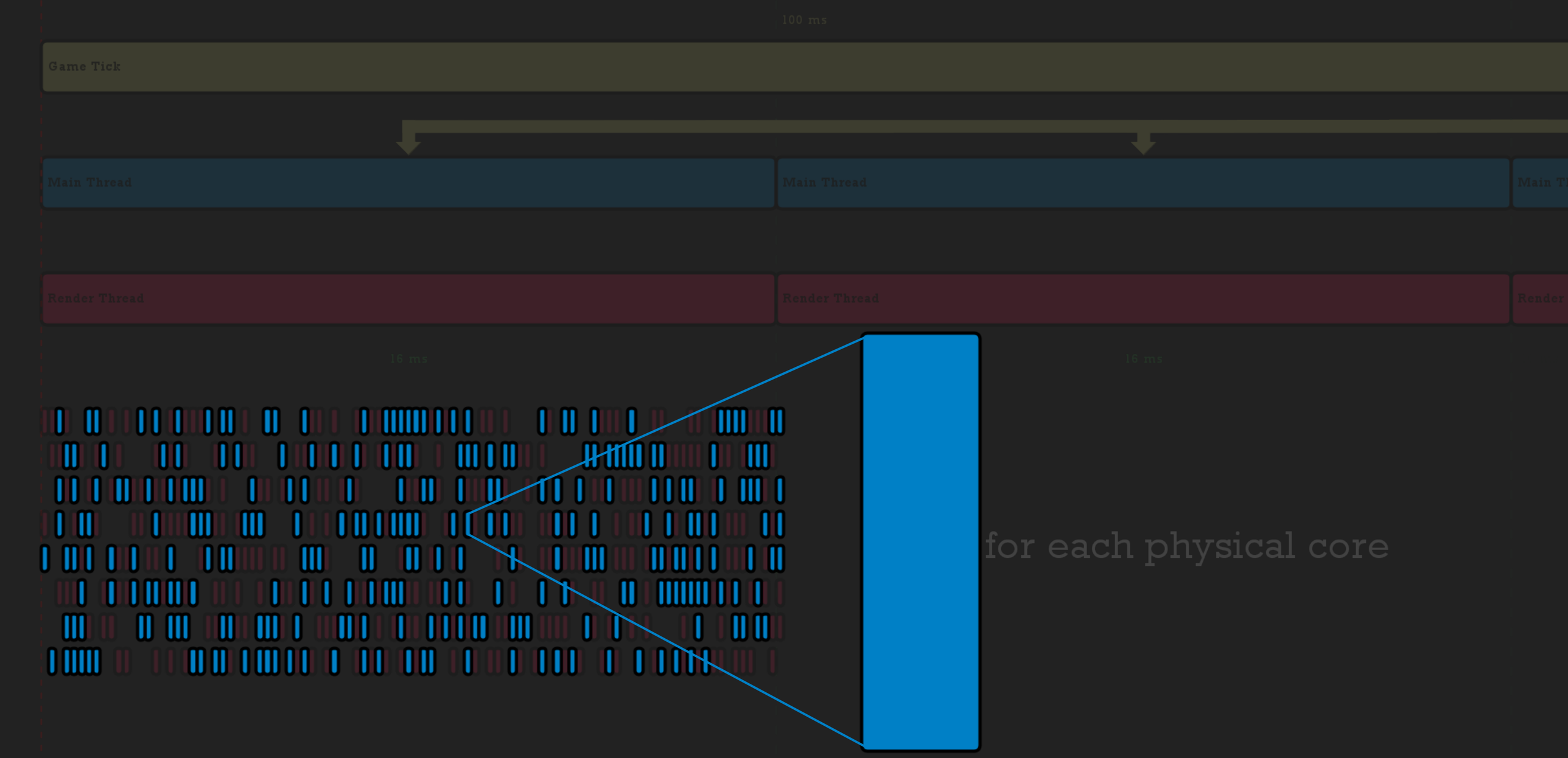
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD

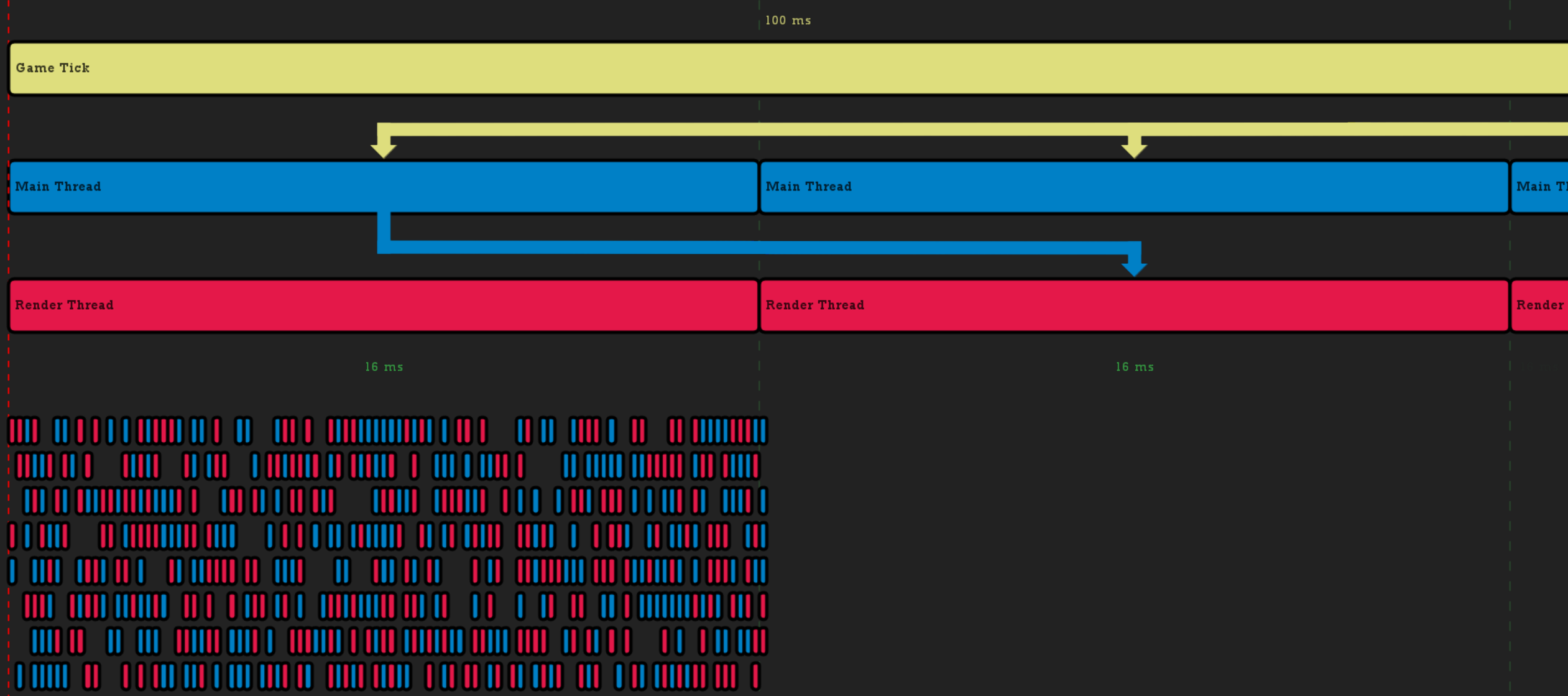
Just like the instance lists we have one mesh list for every single worker thread.

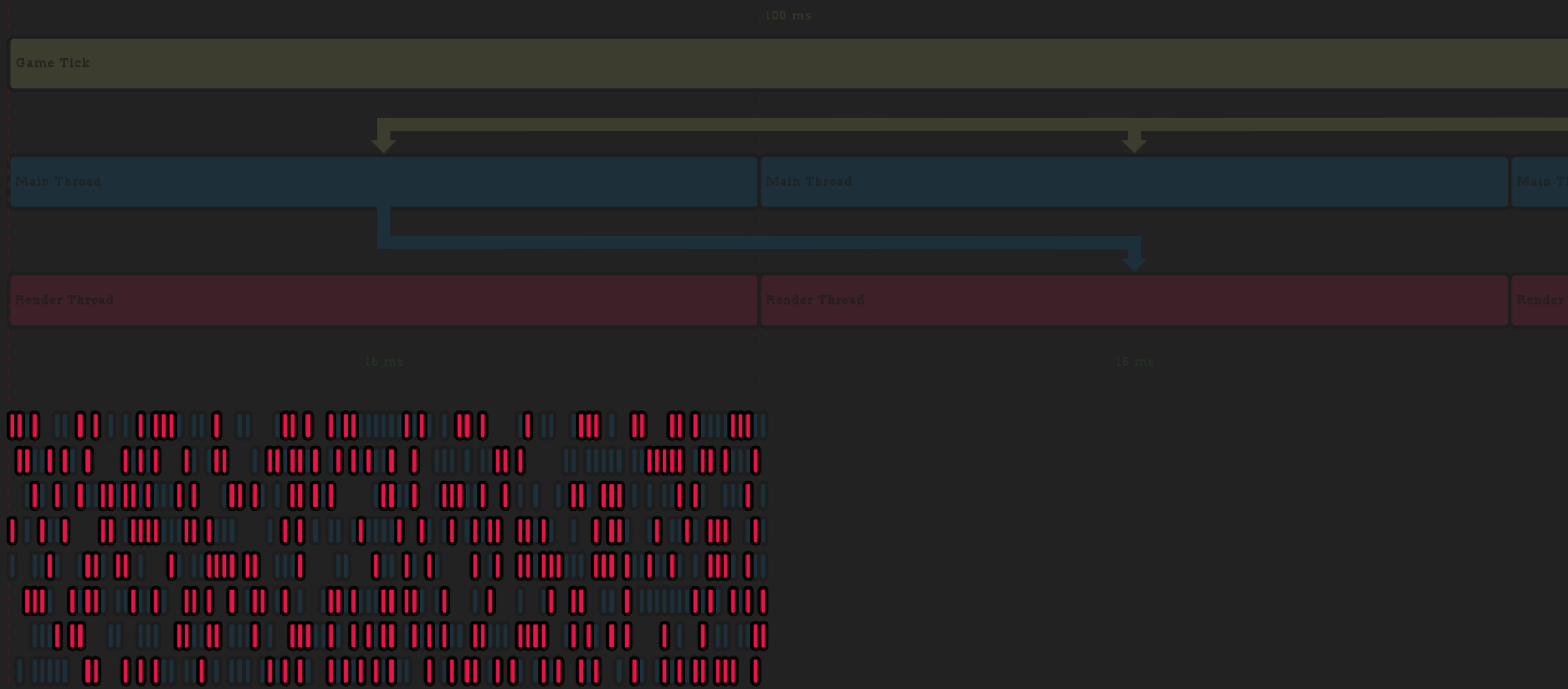
MESH LISTS

TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD
TERRAIN	RIGID	WEIGHTED	BILLBOARD

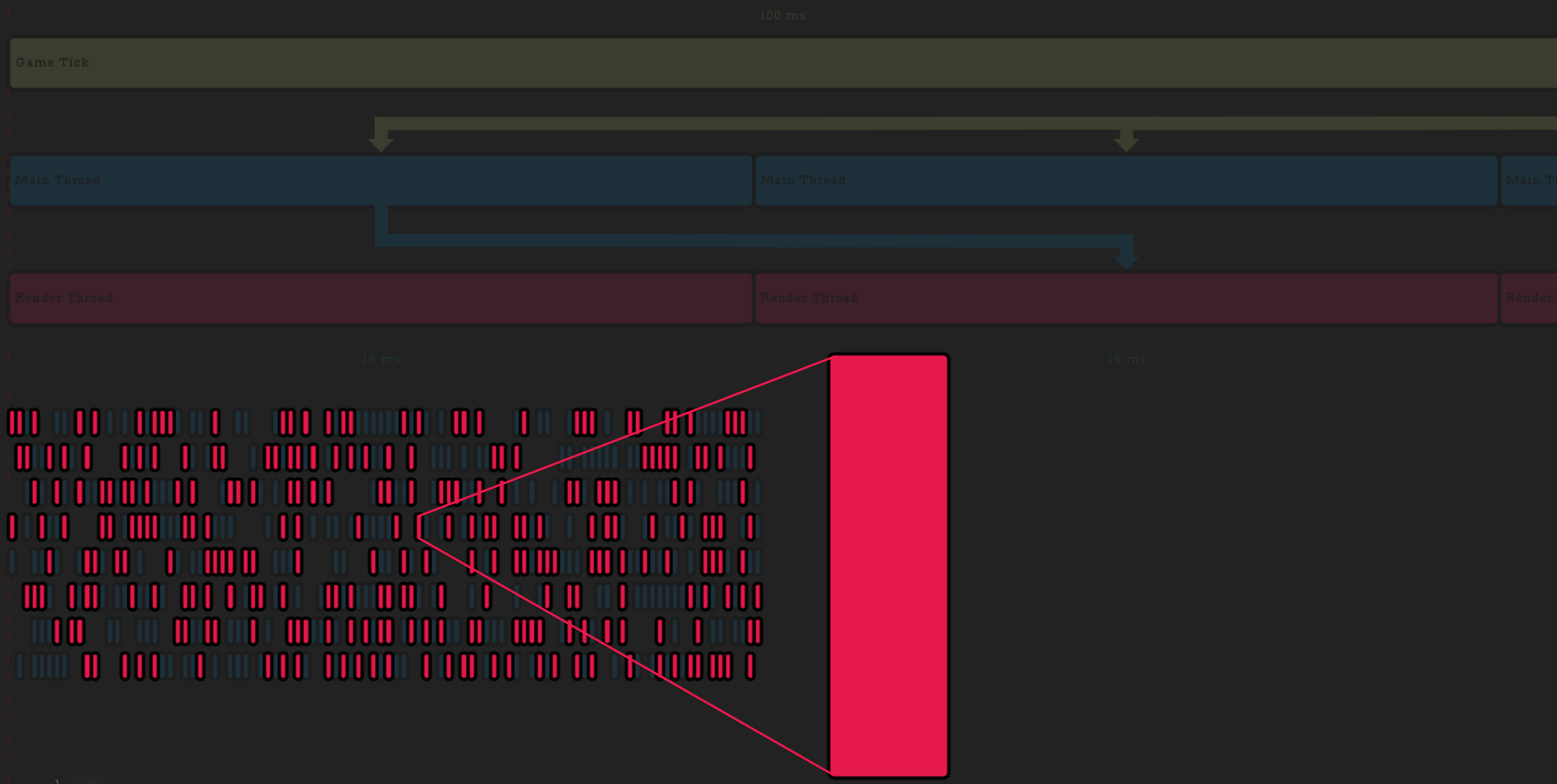
Double buffered too.


100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms
for each physical core


100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms

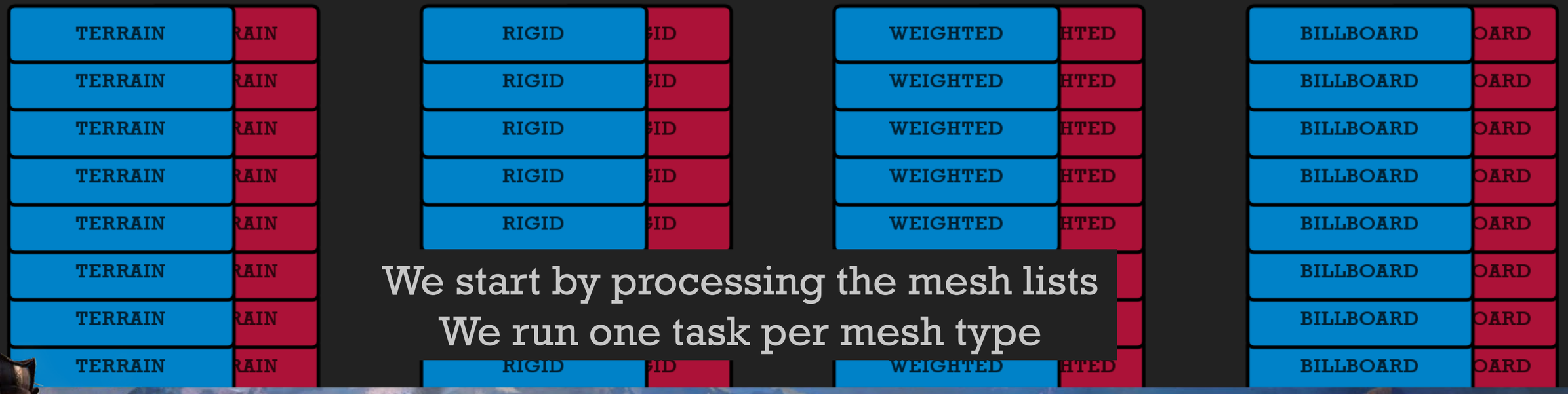

100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms


@IntelSoftware
@IntelGraphics
@totalwar
@CAGames


100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms

WORKER THREAD FOR RT

MESH LIST OF ONE MESH TYPE



We start by processing the mesh lists
We run one task per mesh type

MESH LIST OF ONE MESH TYPE

Per-thread mesh lists are combined to a single list of meshes

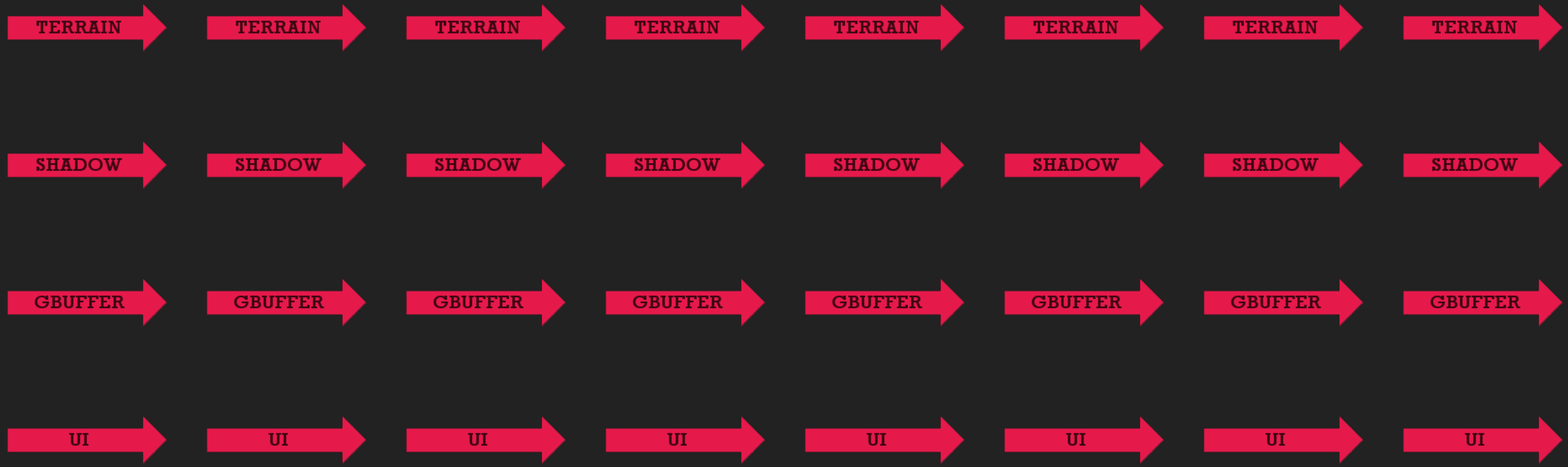
MESH LIST OF ONE MESH TYPE

Per-thread mesh lists are combined to a single list of meshes

MESH LIST OF ONE MESH TYPE

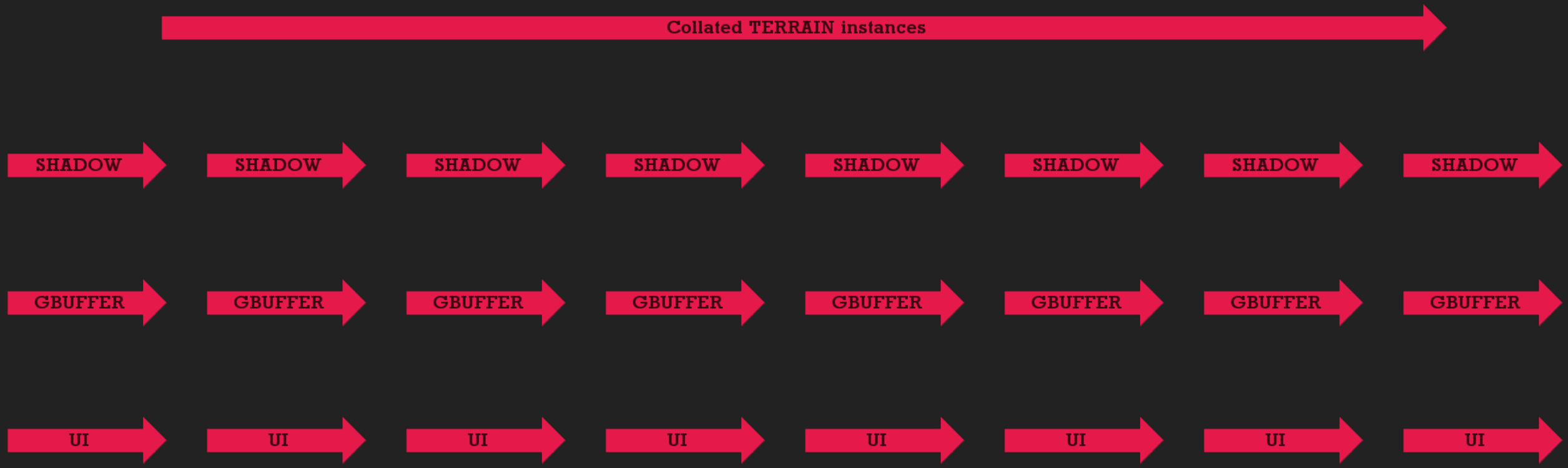
Then we process all the meshes in the list one by one

INSTANCE LISTS



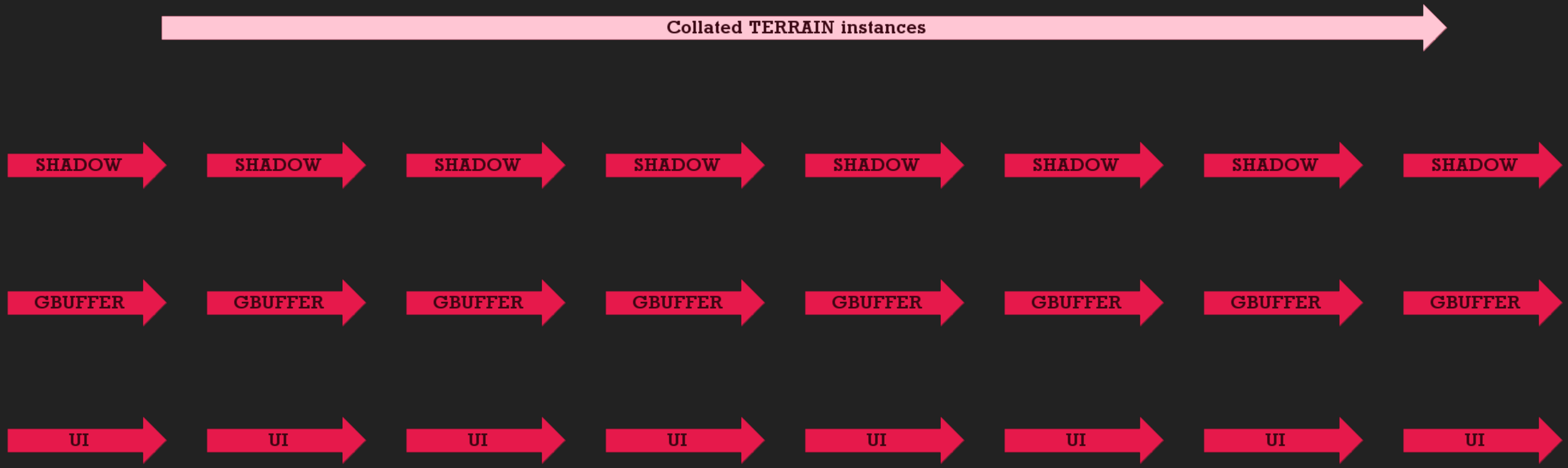
Each mesh has instance lists per pipeline stage

INSTANCE LISTS

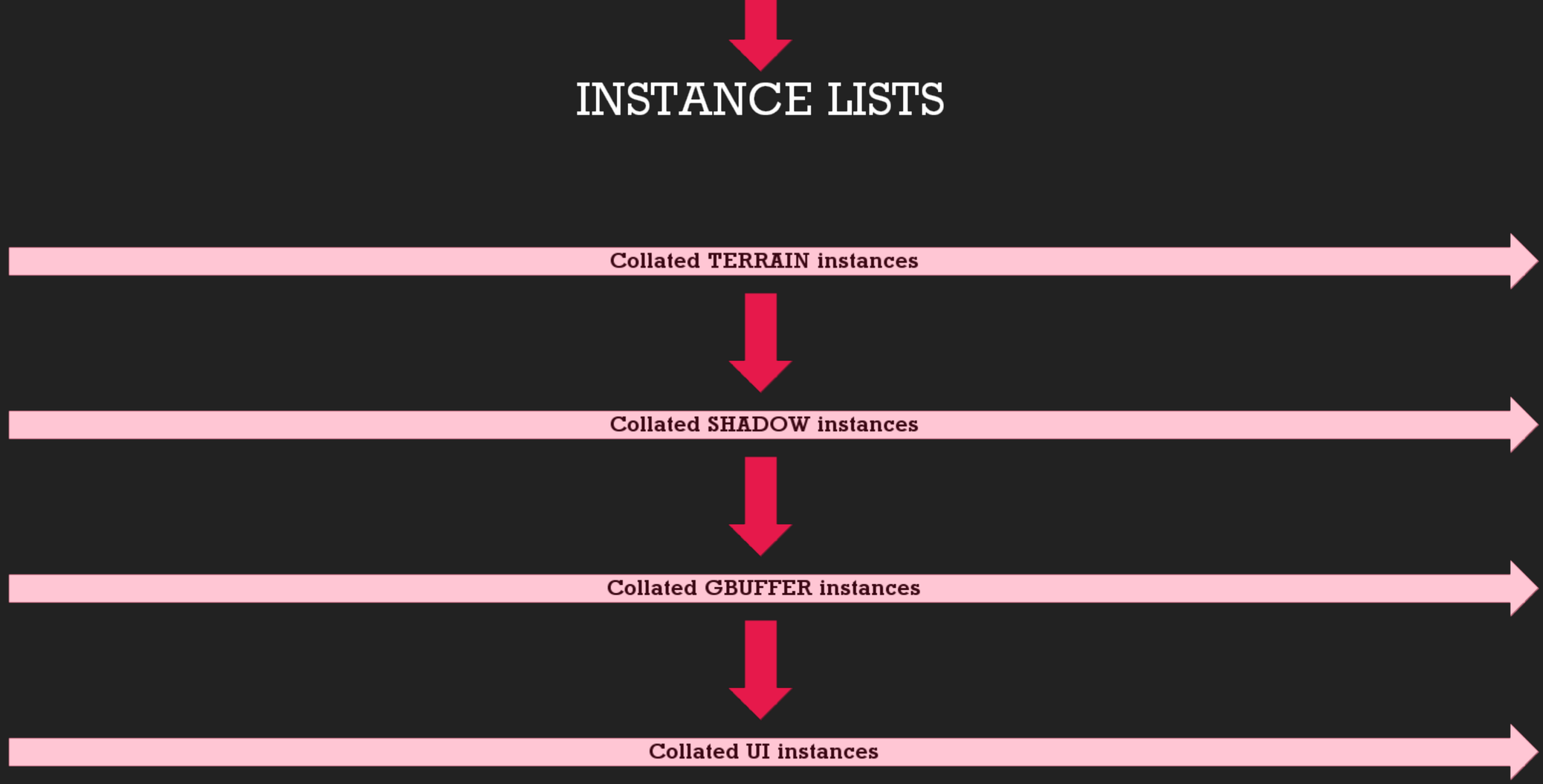


We start with the first pipeline stage and
combine the instances there into a single list

INSTANCE LISTS



Then we process the instances



Then process all subsequent stages sequentially

INSTANCE LIST PROCESSING

Collated TERRAIN instances

INSTANCE LIST PROCESSING

Every instance has a 64 bit sort key

Includes bits from distance from camera

Pieces of 'instance' data map to bits in the sort key

Material flags (blood, burning, dead), LUT index, etc.

Collated TERRAIN instances

INSTANCE LIST PROCESSING

Every instance has a 64 bit sort key

Includes bits from distance from camera

Pieces of 'instance' data map to bits in the sort key

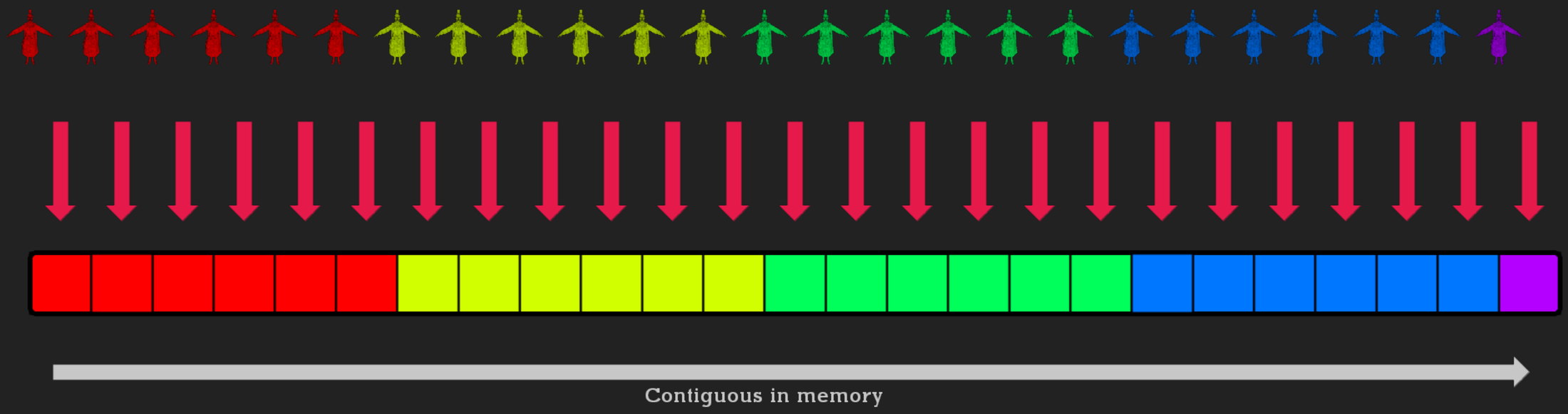
Material flags (blood, burning, dead), LUT index, etc.

Sorting moves similar instances next to each other

Similar instances can be rendered with a single draw call

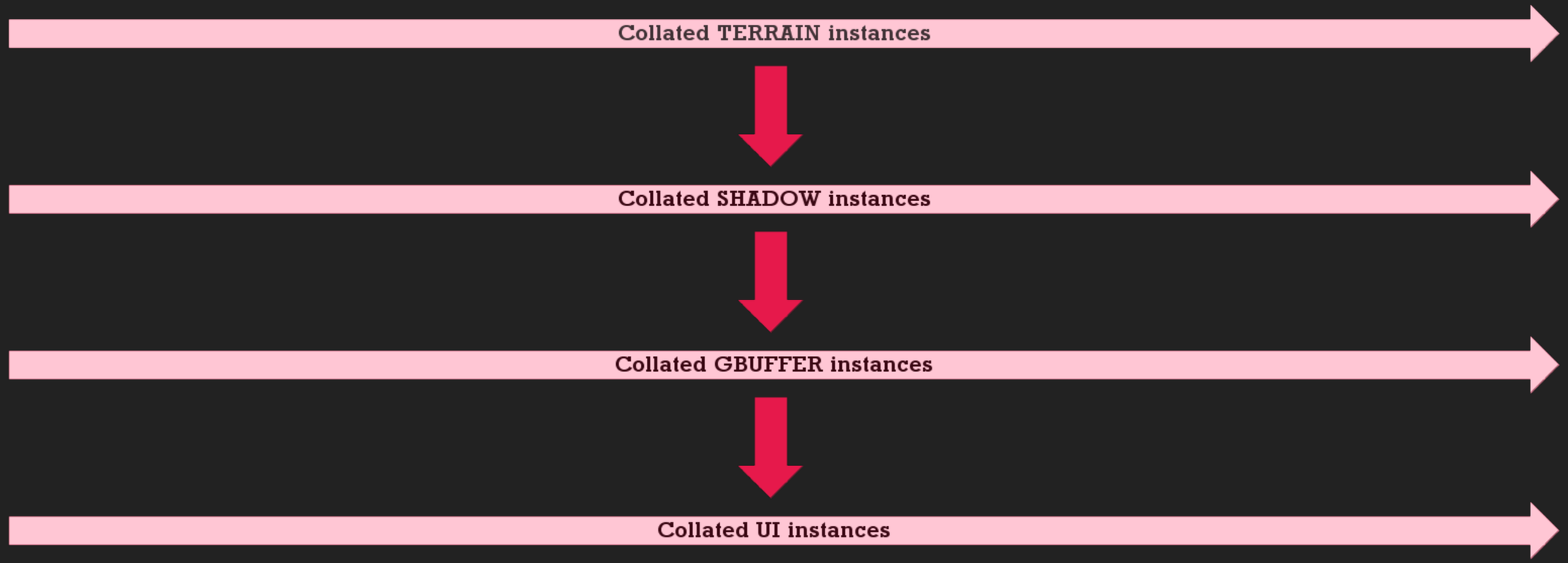
Collated TERRAIN instances

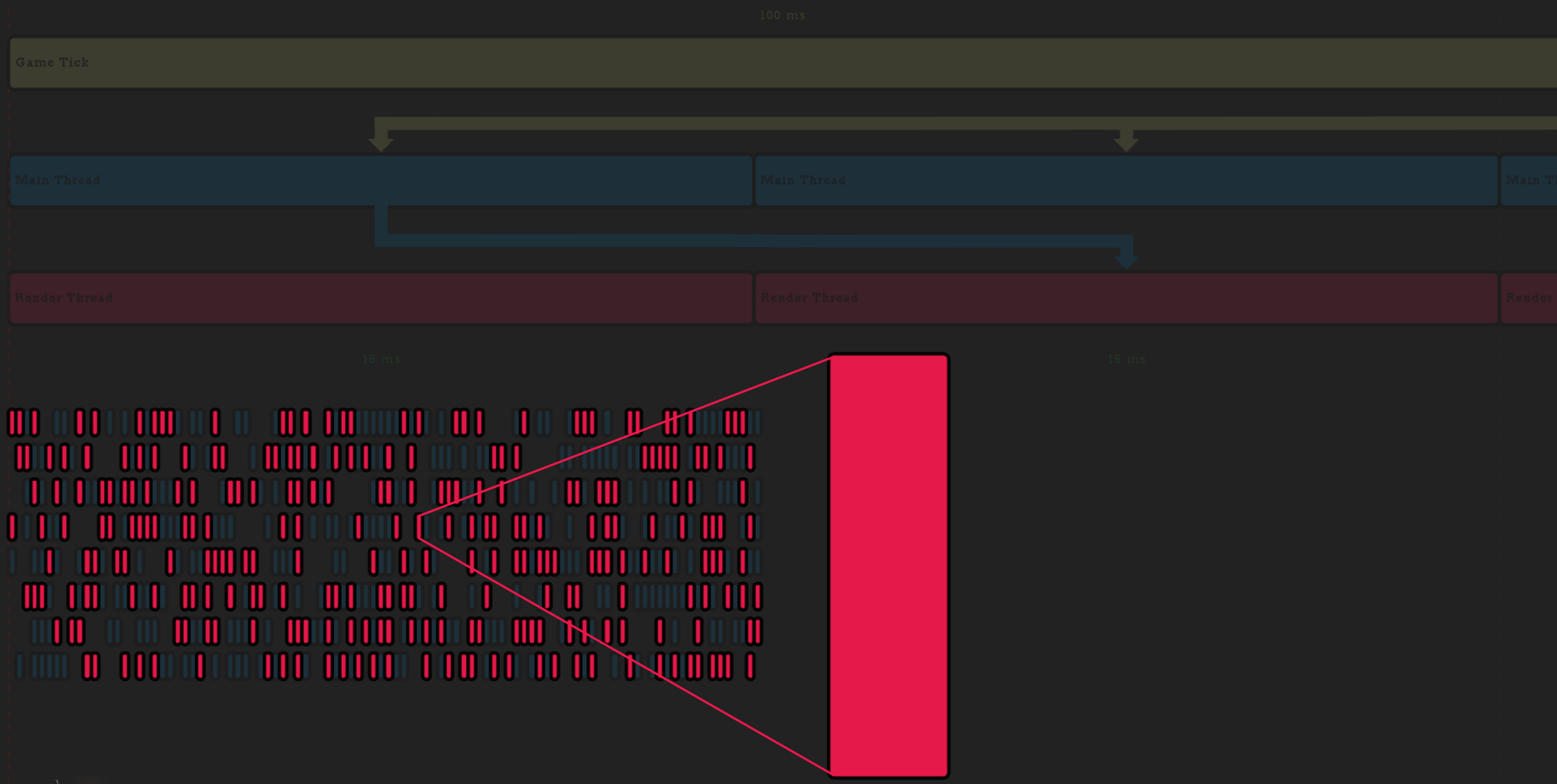
INSTANCE LIST PROCESSING

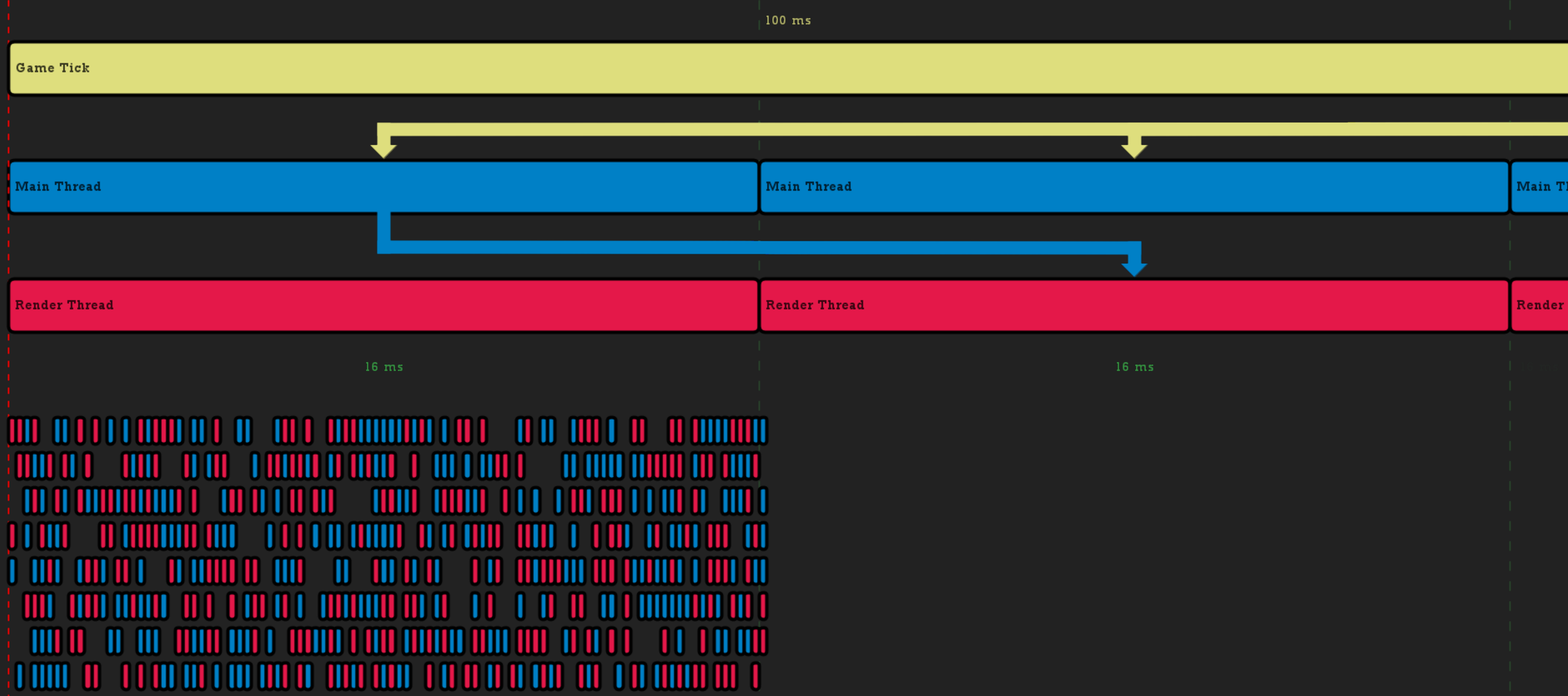


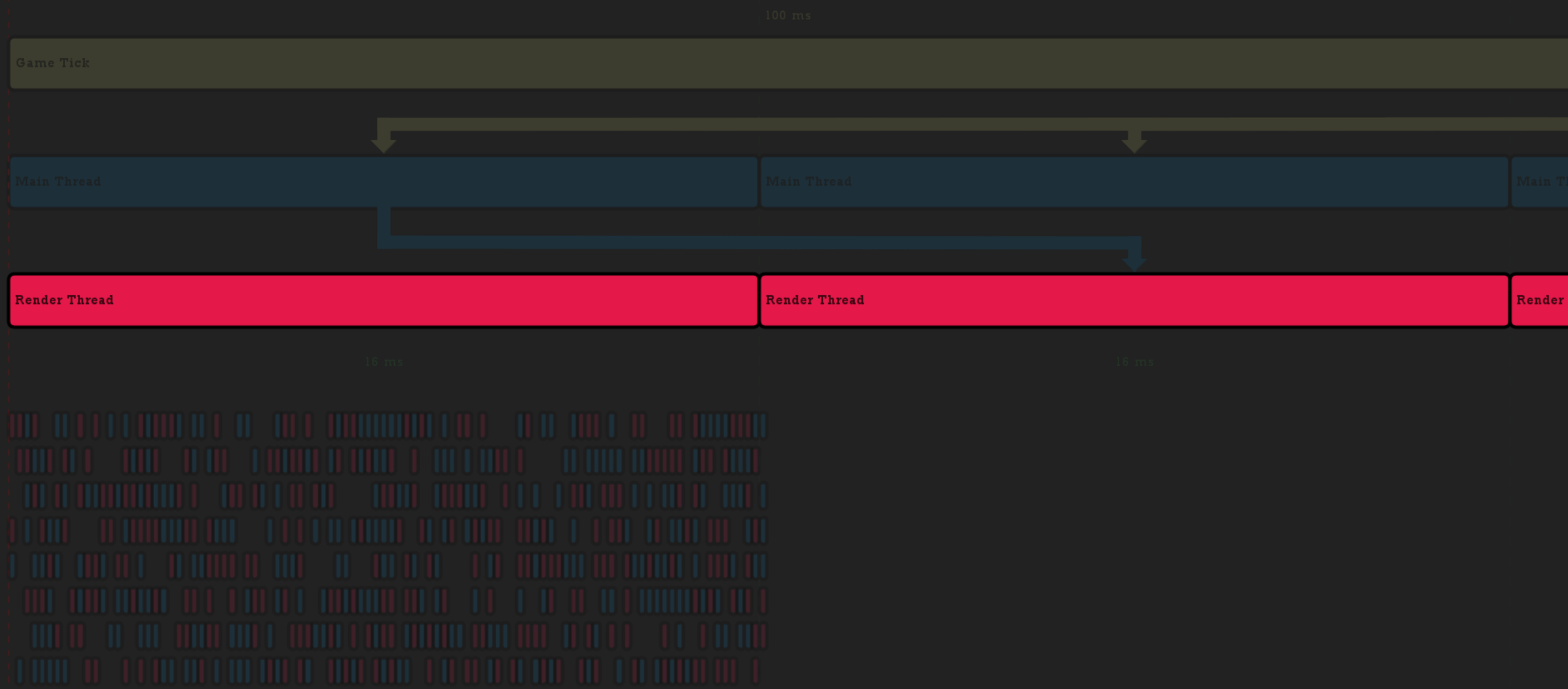
Instance data is now uploaded to the GPU and prepared for batching

INSTANCE LISTS




100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms


100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms


100 ms
Game Tick
Main Thread
Main Thread
Main T
Render Thread
Render Thread
Render
16 ms
16 ms

RENDERING

Render Thread

The actual rendering follows the different pipeline stages

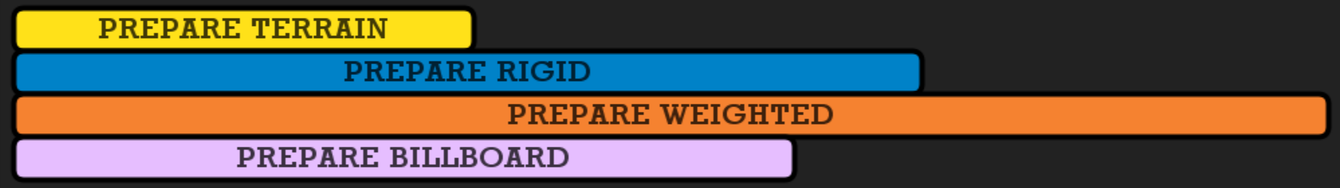
RENDERING

Render Thread

Each stage renders a selection of mesh types

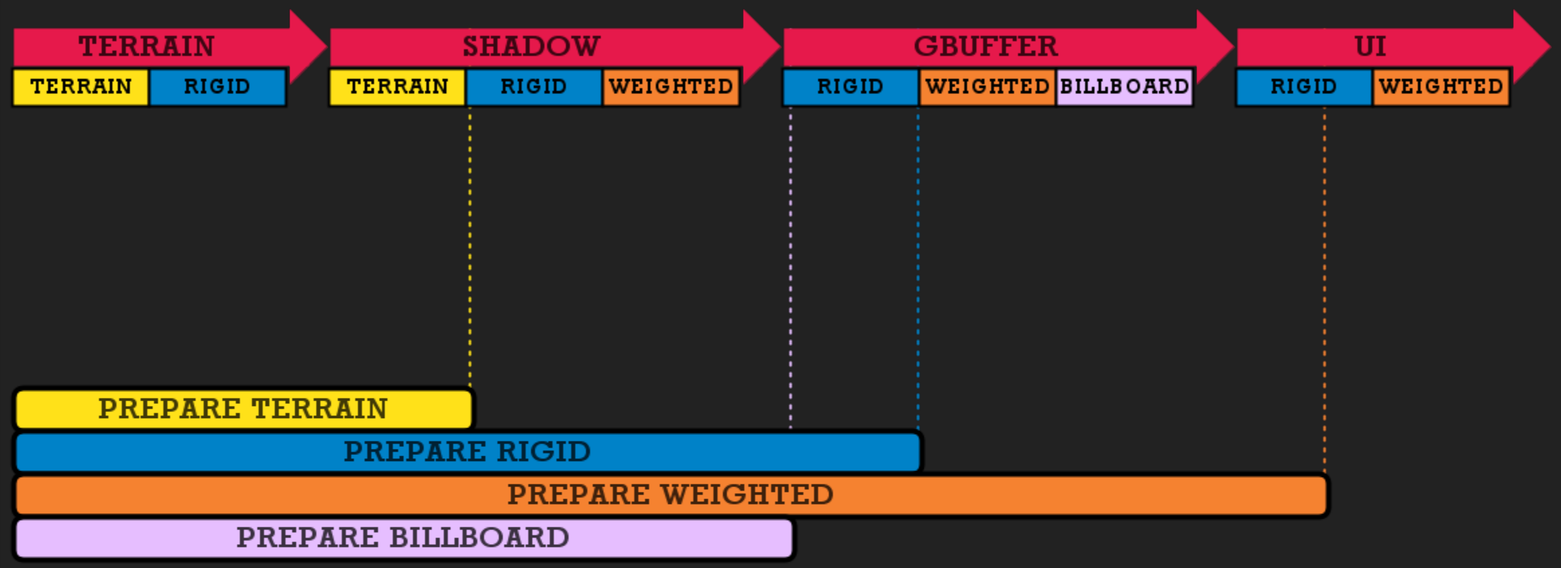
RENDERING

Render Thread



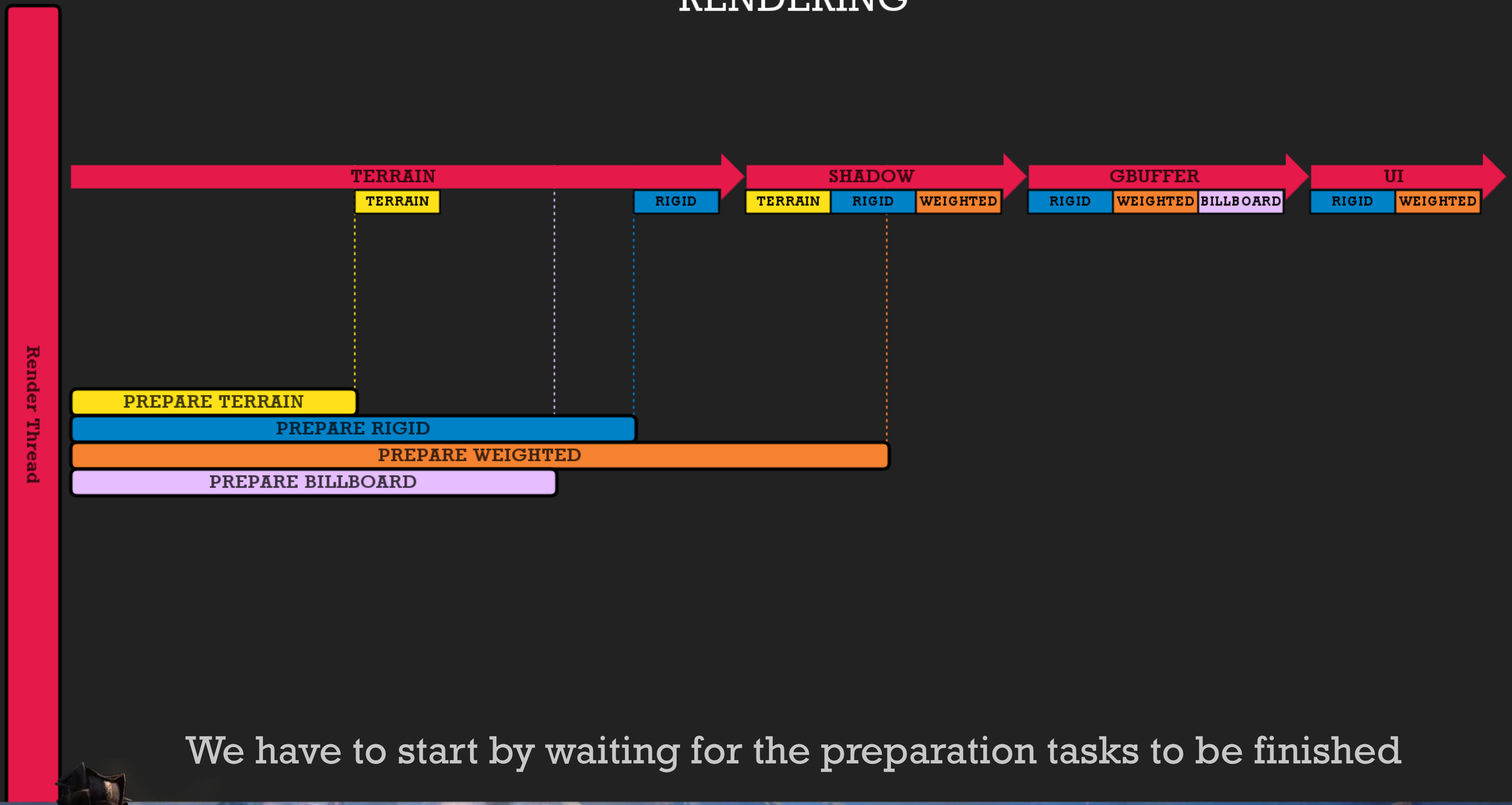
The meshes are prepared by the render thread workers

RENDERING



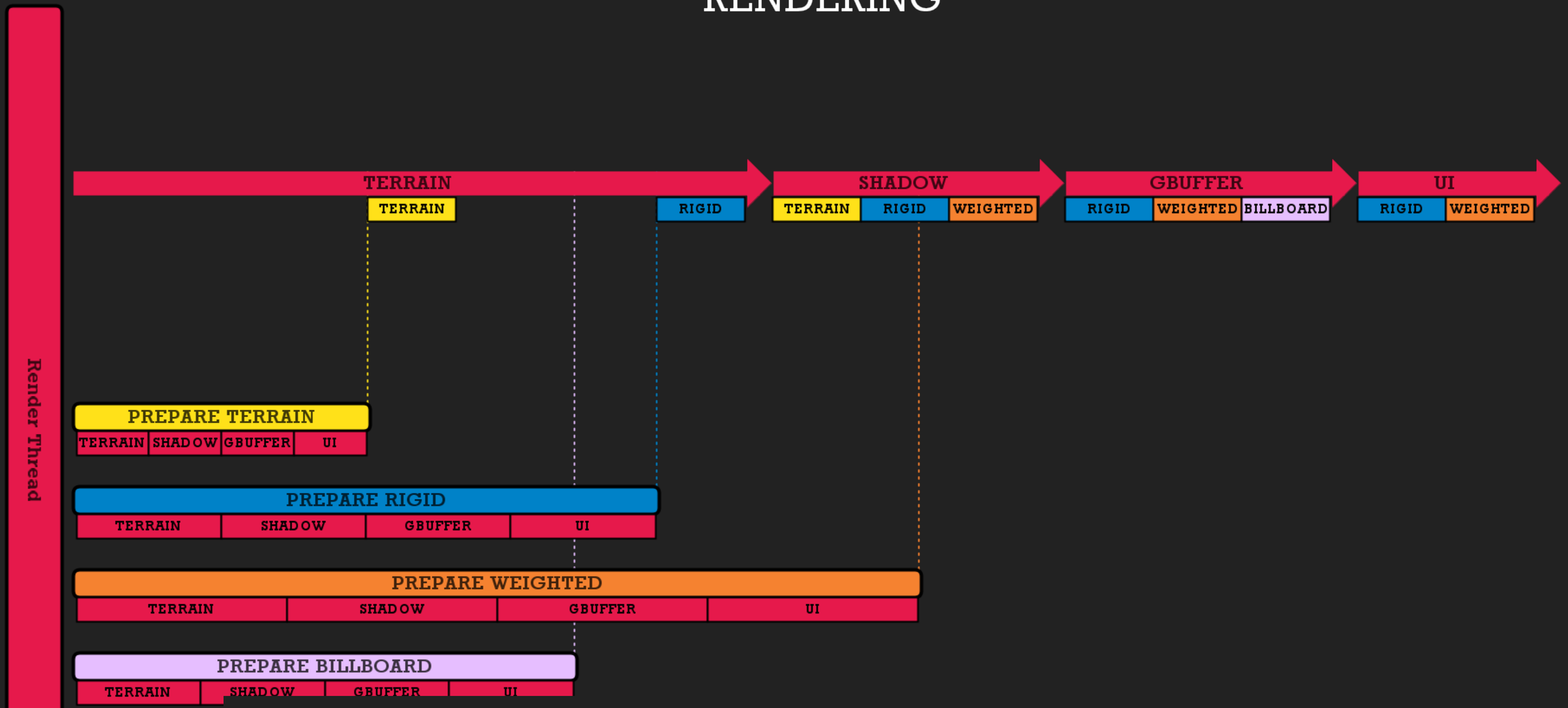
We have to start by waiting for the preparation tasks to be finished

RENDERING



We have to start by waiting for the preparation tasks to be finished

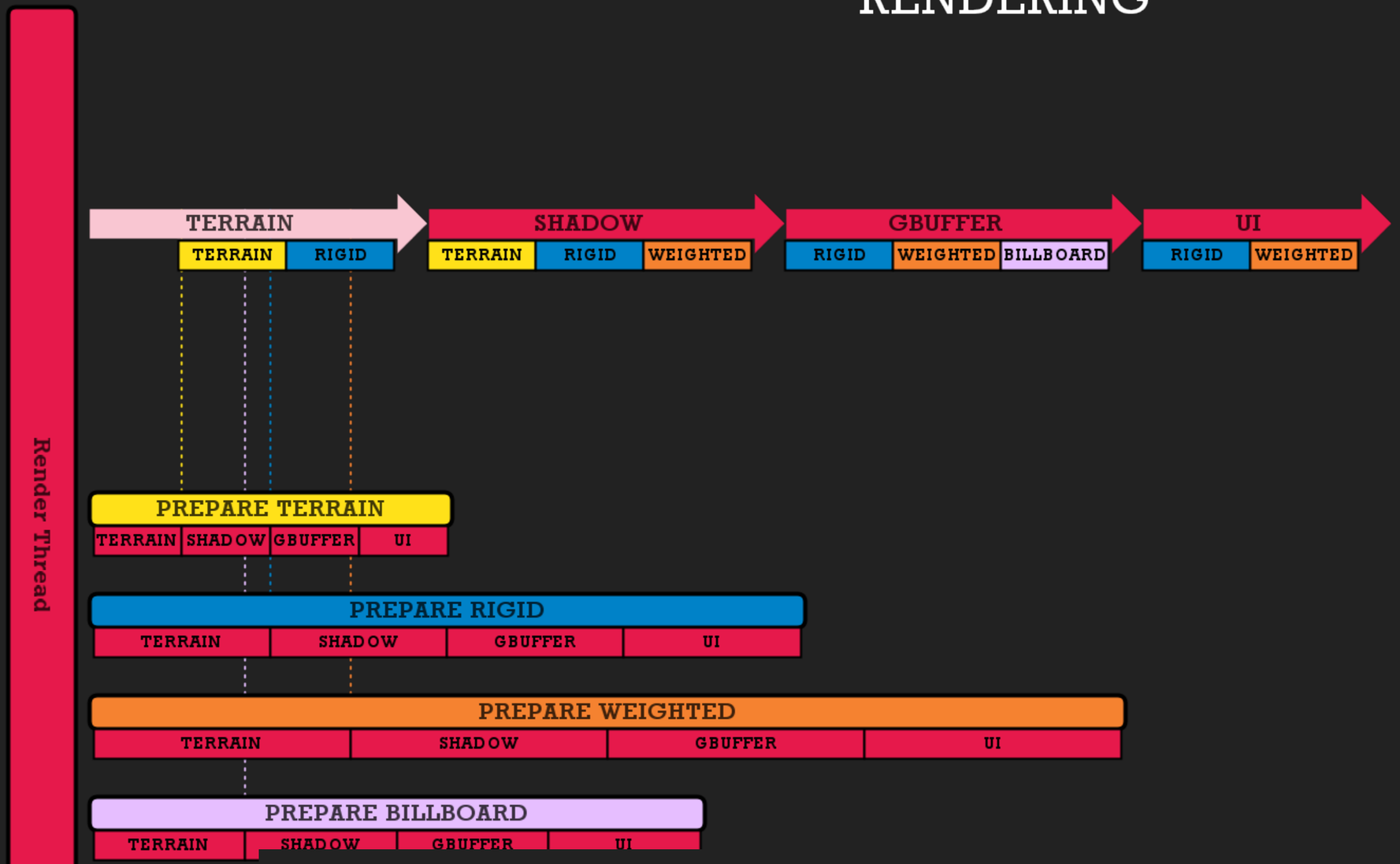
RENDERING



The tasks process all meshes/instances for all pipeline stages

We can use an atomic counter per pipeline stage

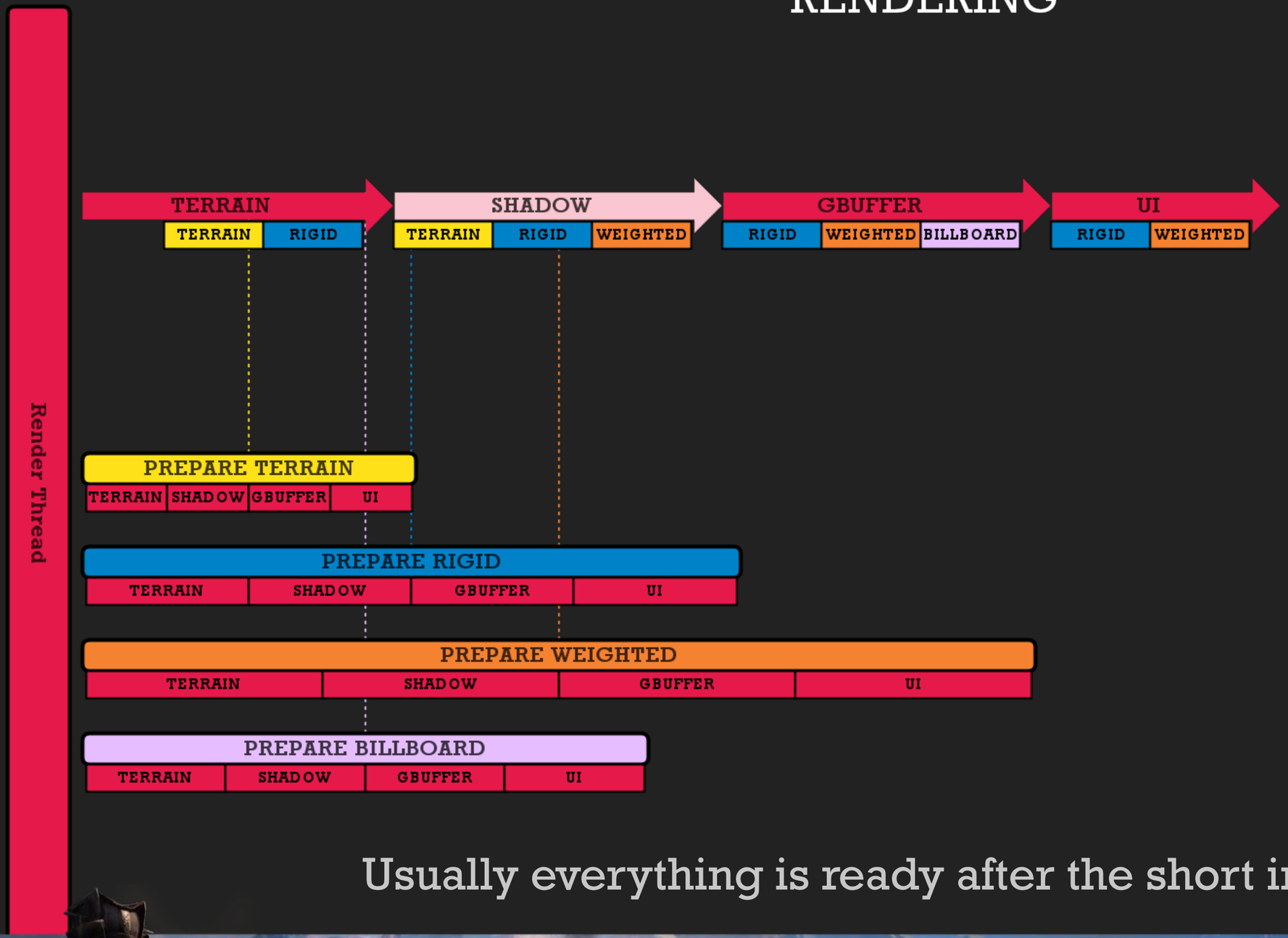
RENDERING



No need to further split the tasks

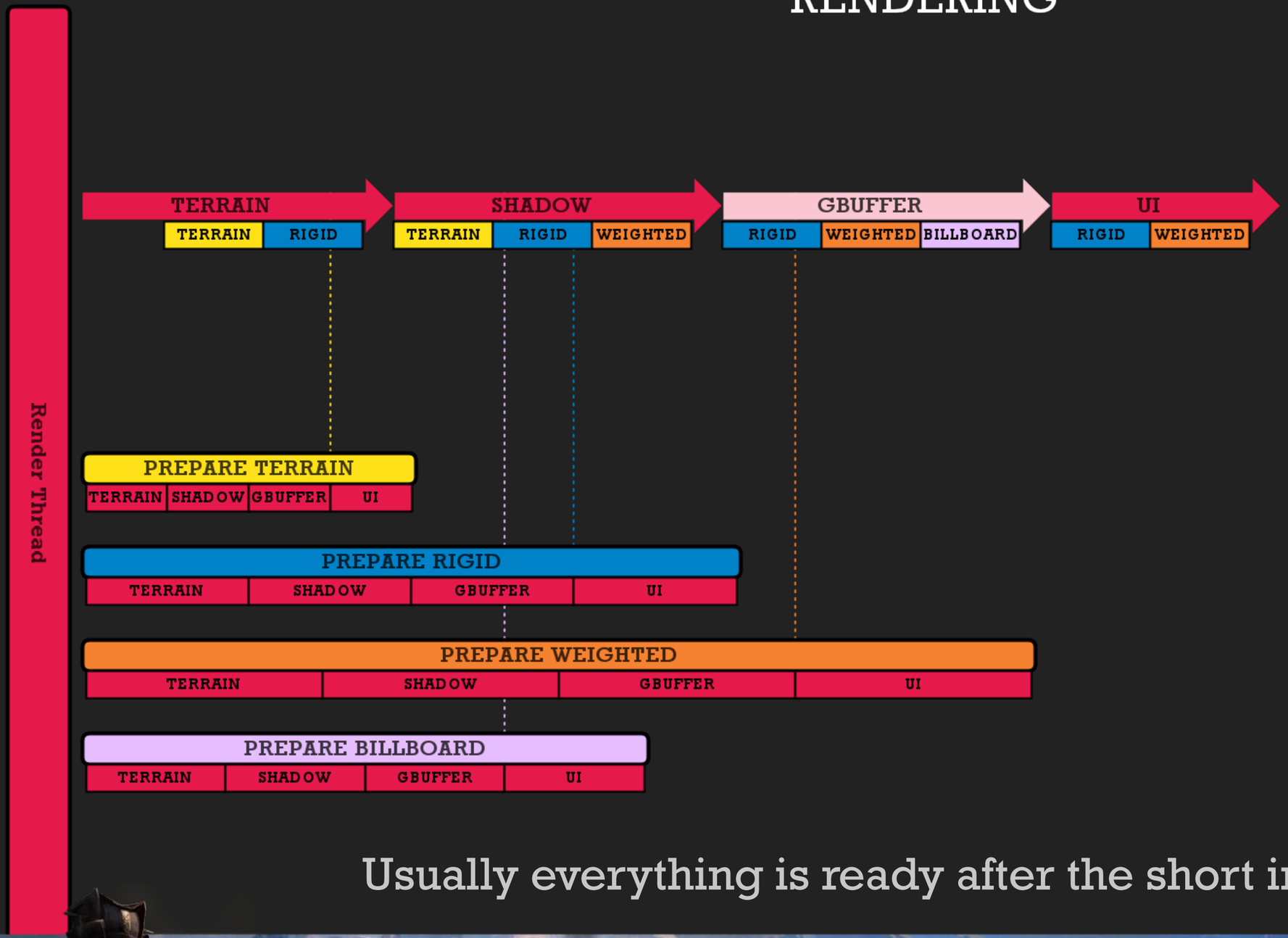
We just increased the granularity of waiting for other sub-tasks

RENDERING



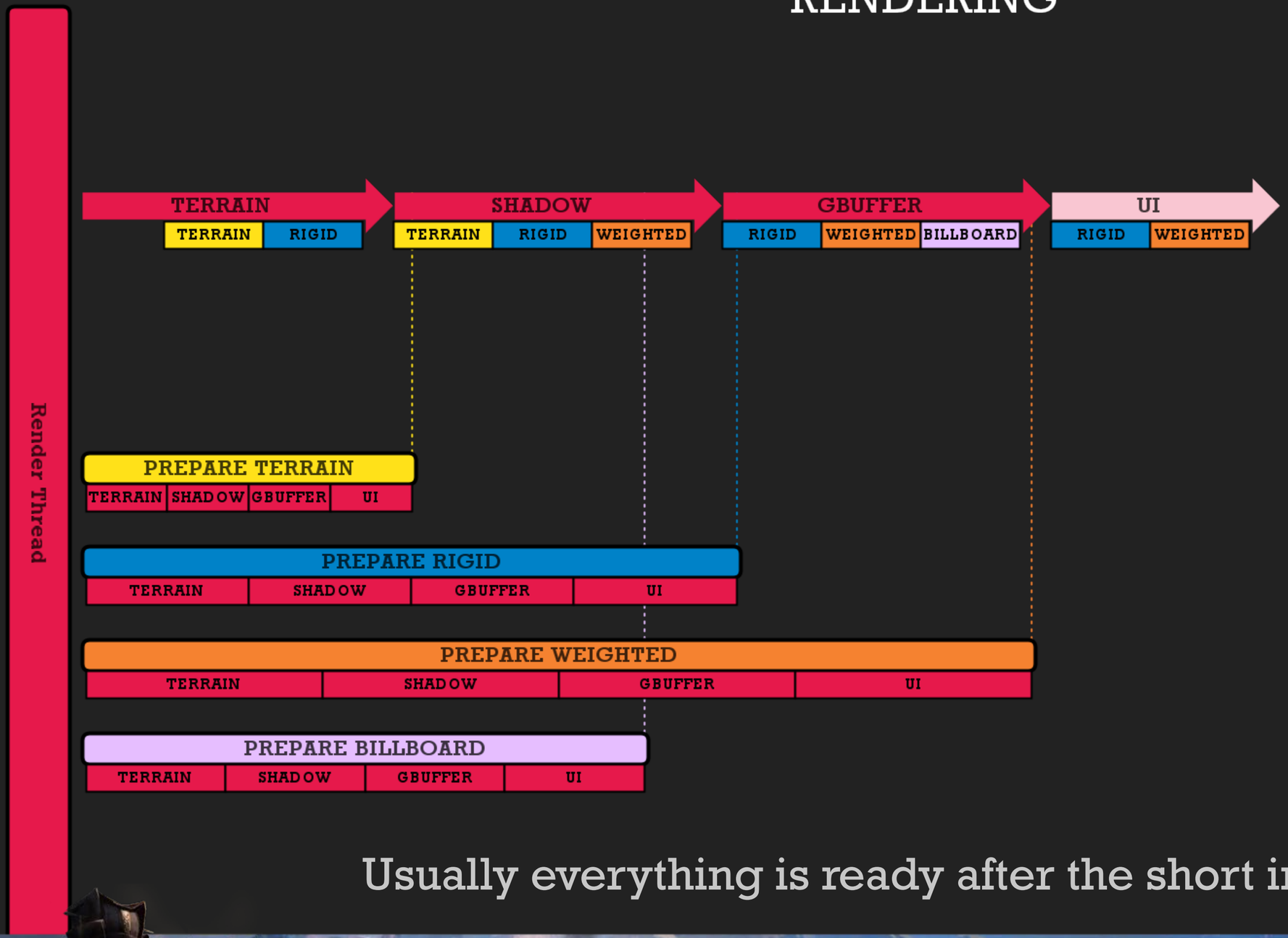
Usually everything is ready after the short initial wait

RENDERING

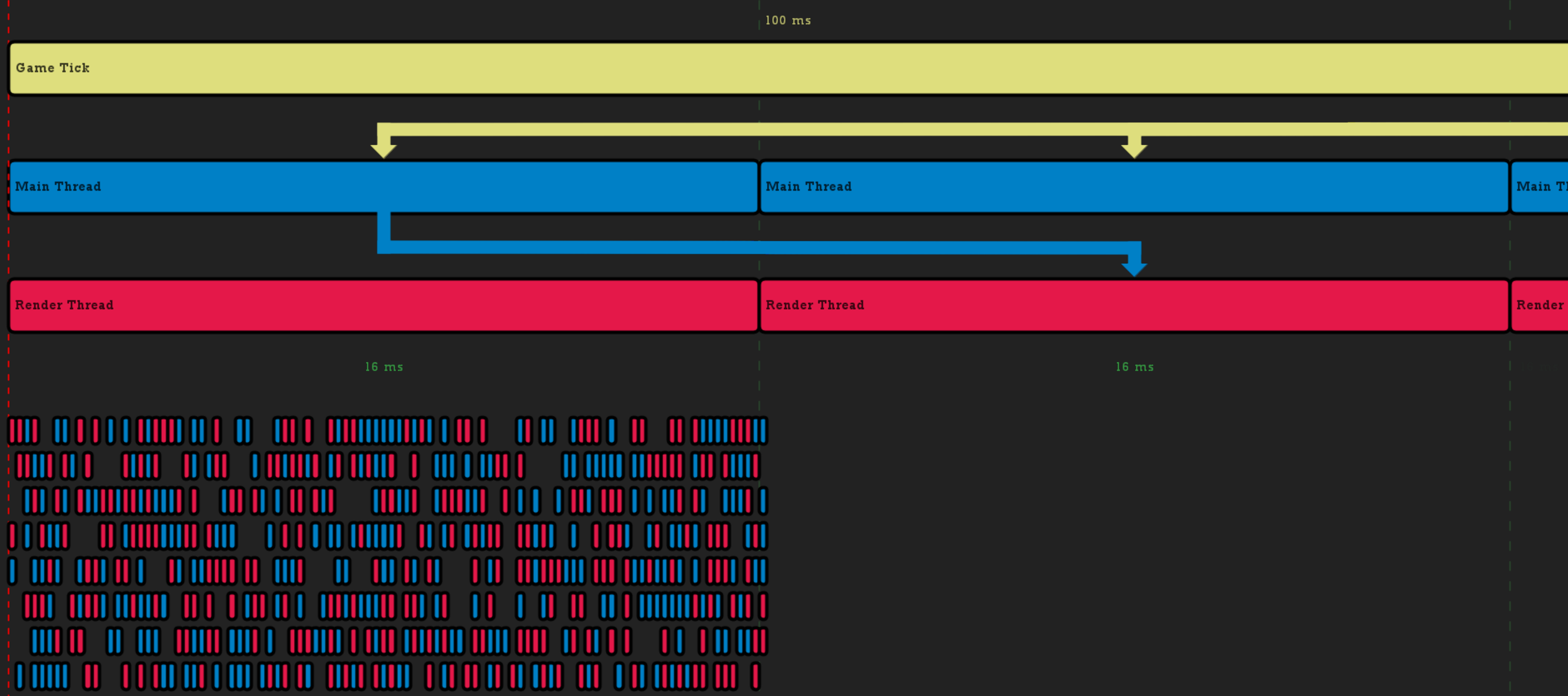


Usually everything is ready after the short initial wait

RENDERING

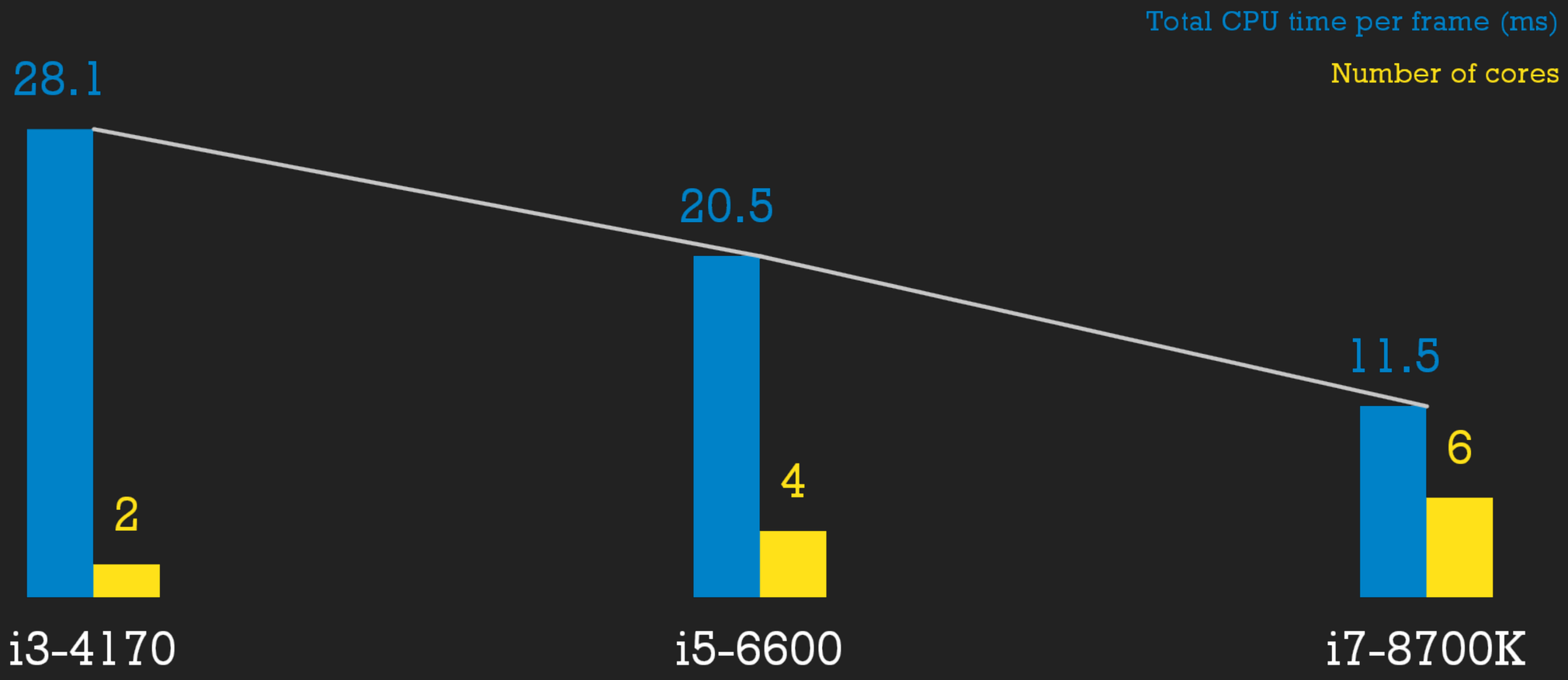


Usually everything is ready after the short initial wait


100 ms
Game Tick
Main Thread
Main Thread
Render Thread
Render Thread
16 ms
16 ms

RESULTS

70-80% scaling



INSIGHTS

Per-thread lists are still our go-to way doing batch processing

Try splitting batch processing tasks to 2 * #cores parts

Memory layout can highly influence performance

You can’t do better than your longest serial part

You need to plan your systems with multithreading in mind from the start

More cores, stronger CPU can mean more detail and better visuals

INTEGRATED CARDS

Main test machine was Skull Canyon

Almost every change we made was to relieve pressure on memory

A quick early out in our DoF pass resulted in a 3.6ms uplift

Shader register usage can also cause problems

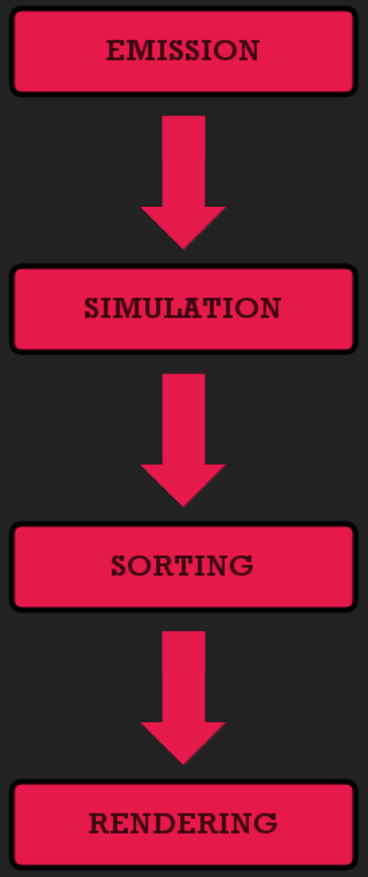
Reducing the number of raymarching steps in fog shader was a 3ms uplift

Copying resources can also take up significant time

Highest individual copy was 0.9ms

ORDER INDEPENDENT TRANSPARENCY FOR VFX

ORDER INDEPENDENT TRANSPARENCY FOR VFX



ORDER INDEPENDENT TRANSPARENCY FOR VFX

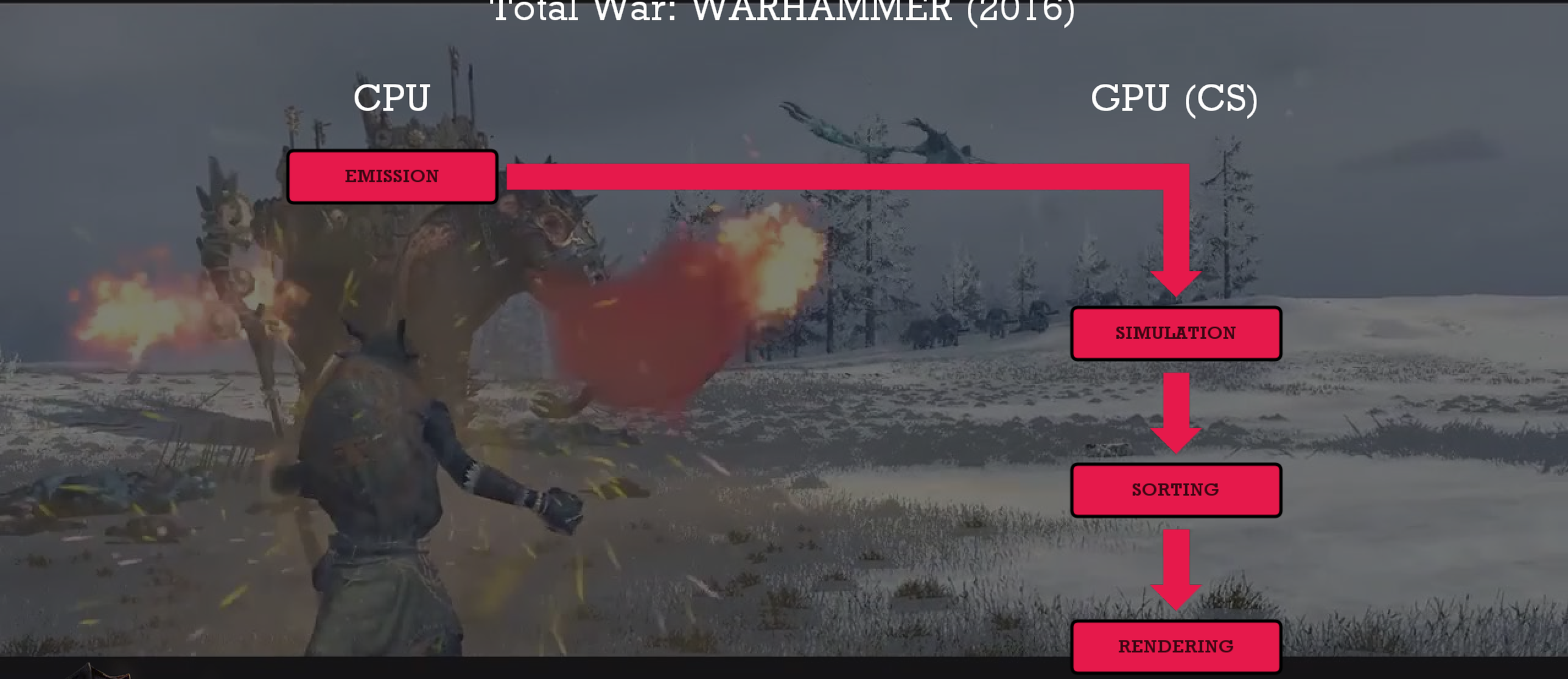
Total War: ROME 2 (2013)

ORDER INDEPENDENT TRANSPARENCY FOR VFX

Total War: ATTILA (2015)

ORDER INDEPENDENT TRANSPARENCY FOR VFX

Total War: WARHAMMER (2016)

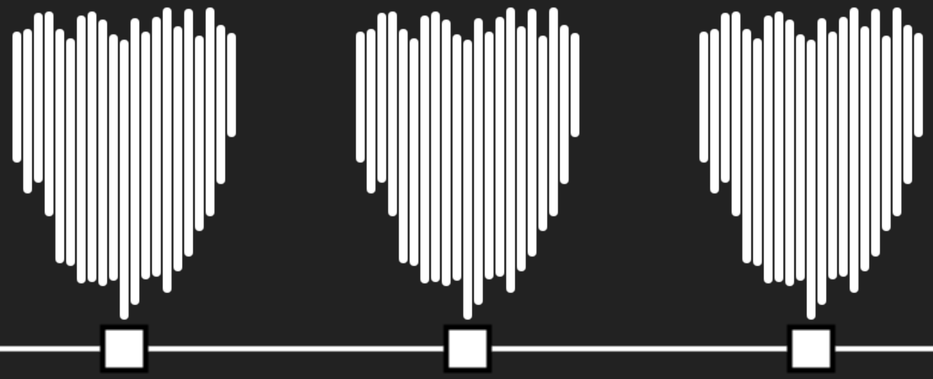


SORTING

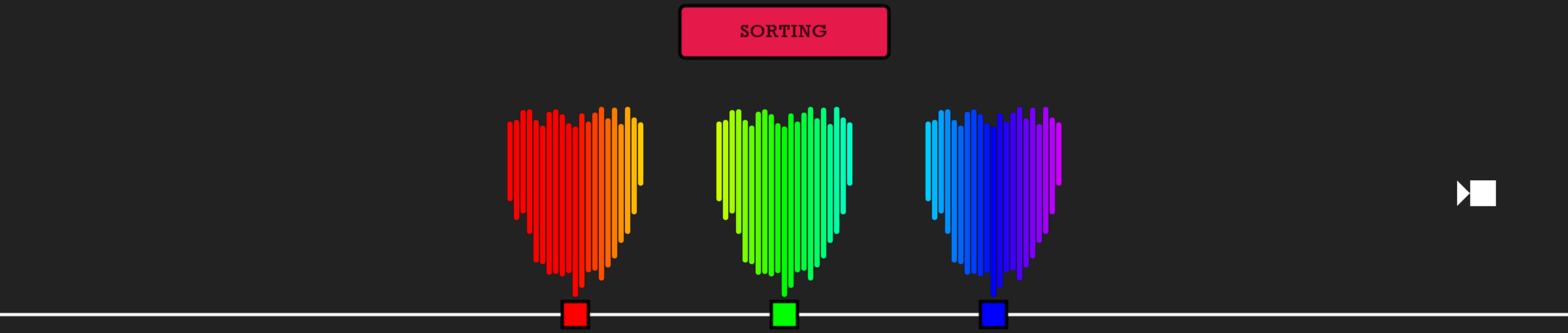
SORTING

The basic building blocks of the system are emitters

SORTING

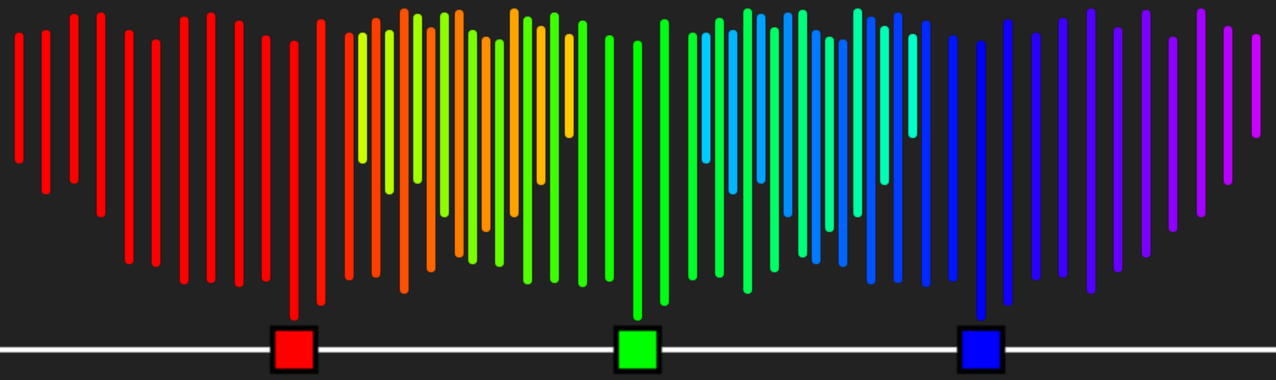


Emitters are emitting a number of particles over time



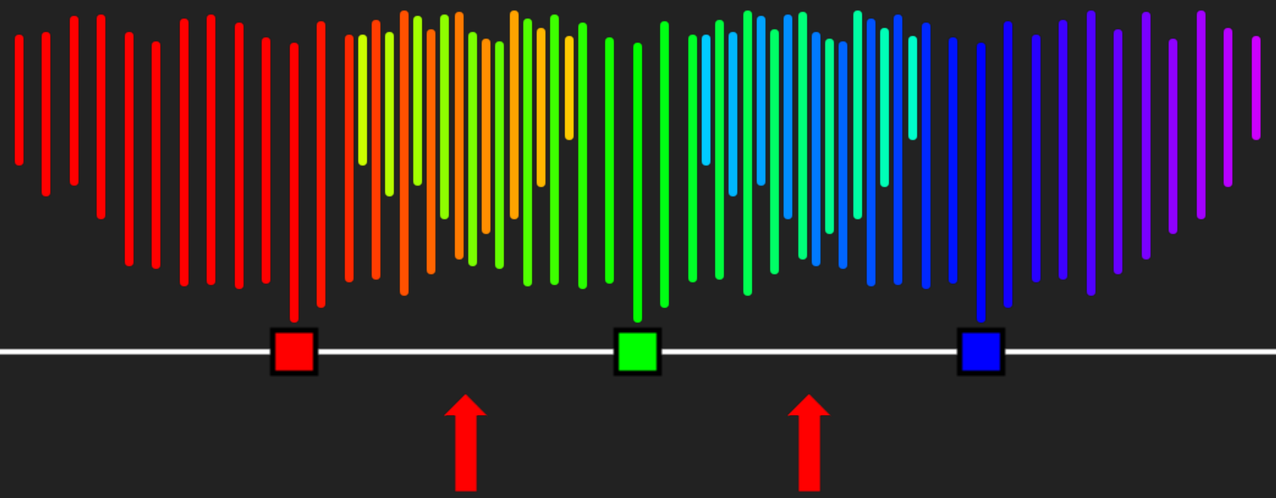
Particles are often just sorted per emitter

SORTING



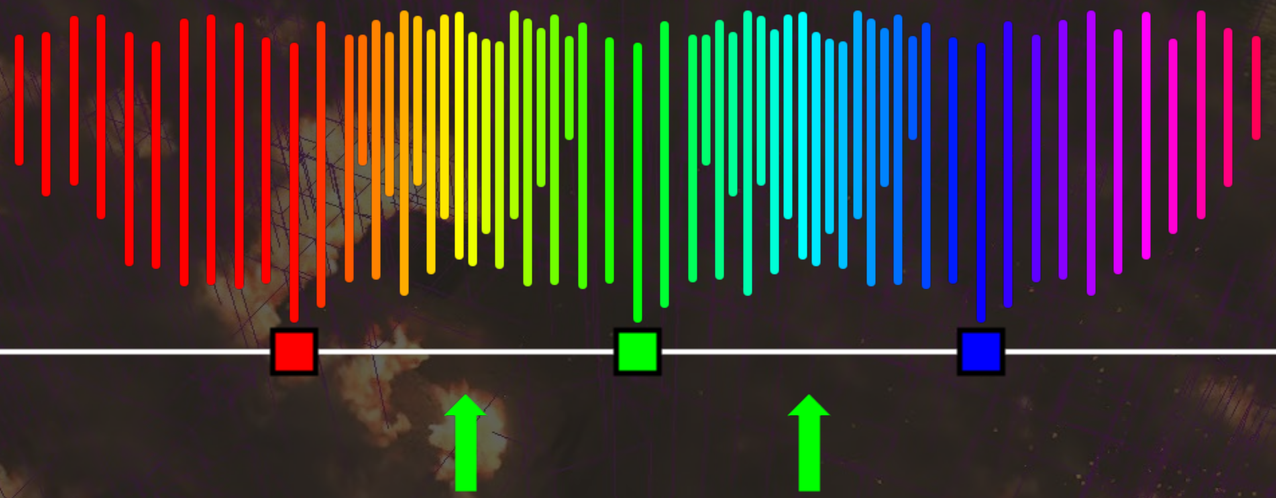
Problems start happening when the emitters overlap

SORTING



Problems start happening when the emitters overlap

SORTING



No artifacts when emitters are overlapping

Have to use übershader

Sorting time increases exponentially with number of particles

We sort all the particles together

EMISSION

EMISSION

NEW PARTICLES (CPU)

Emission takes place on the CPU

EMISSION

NEW PARTICLES (CPU)

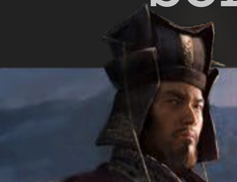
DISTANCE FROM CAMERA

ALIVE PARTICLES

DEAD PARTICLES

PARTICLE BUFFER (GPU)

Sorting is responsible for moving dead particles to one end of the GPU buffer



EMISSION

NEW PARTICLES (CPU)

DISTANCE FROM CAMERA

ALIVE PARTICLES

DEAD PARTICLES

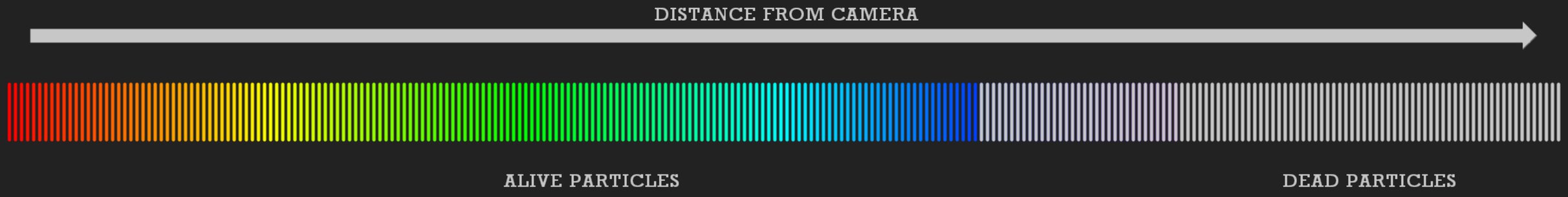
PARTICLE BUFFER (GPU)

This make uploading new particles trivial

EMISSION

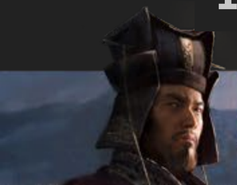
NEW PARTICLES (CPU)

If overrun, new particles stomp over the farthest live particles



PARTICLE BUFFER (GPU)

Running out of particle space stomps over particles farthest from camera



GOALS
(for Total War: THREE KINGDOMS)

Freely mix half and full resolution particles

Get rid of (some of) the sorting cost

Get rid of übershader and unlock artist driven shaders for particles

We were confident that order-independent transparency is the solution

PHENOMENOLOGICAL TRANSPARENCY (WBOIT) [McGuire17]

Lightweight

Already used in production
(Rendering Technology in 'Agents of Mayhem' [Kircher18])

Doesn't handle additive blending

Lots of magic numbers

Doesn't work well with close to opaque particles

First try was WBOIT

MOMENT BASED TRANSPARENCY (MBOIT) [Muenstermann2018]

Supports additive and multiplicative blending

Works well with near opaque particles

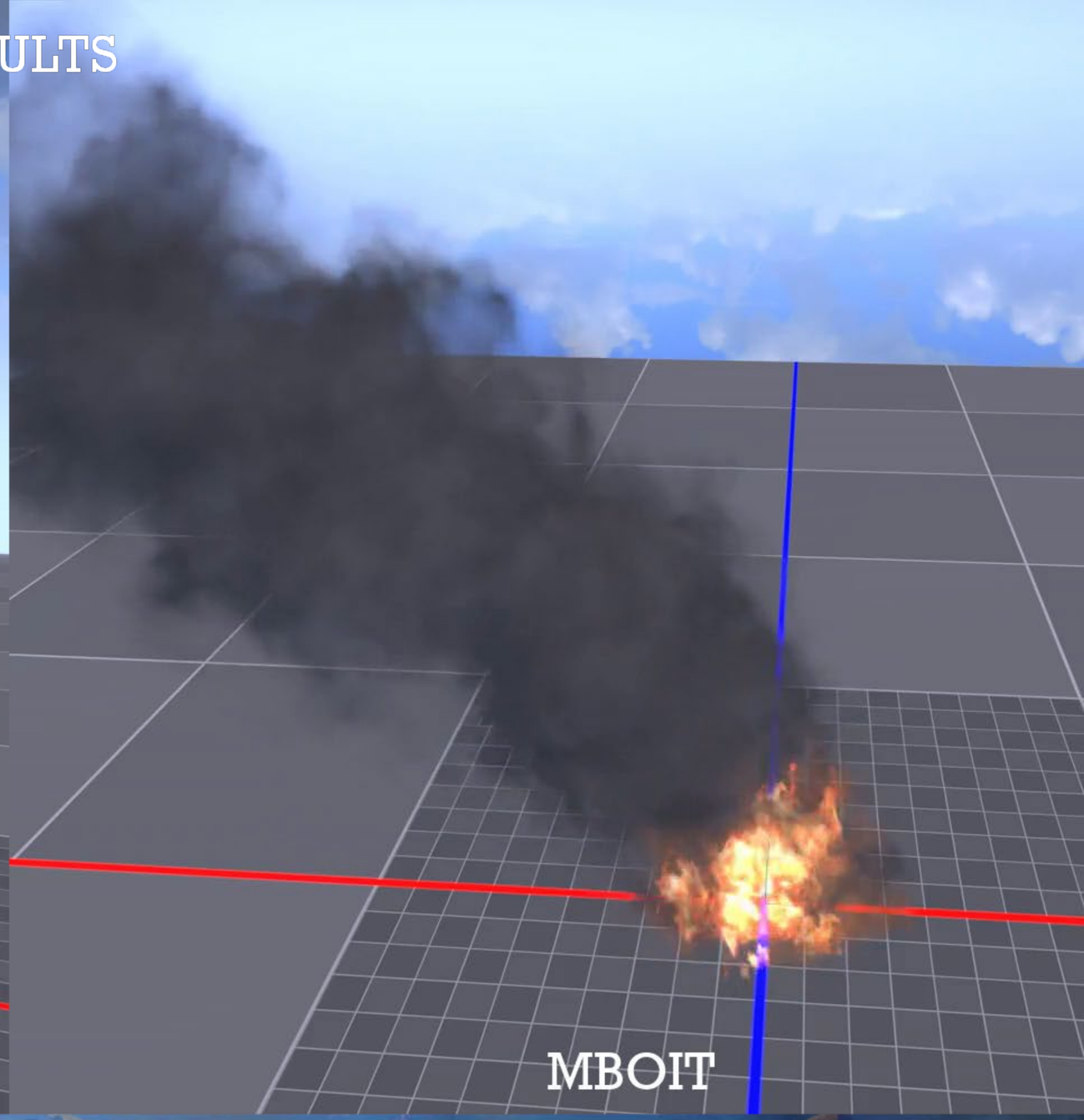
Only a few parameters to tweak

More expensive (extra pass and more render targets)

Second try was MBOIT

RESULTS

WBOIT



MBOIT

RESULTS

No sorting, only bucketing for dead particles

Same performance as before, less fixed cost

Mix and match shaders

Mix and match resolutions

PARTICLE BUFFER (GPU)

TW:3K was released with MBOIT

THANKS

Wittawat Keawcharoen (Aik) for his OIT work

Steve Hughes for his support from Intel

Rendering Team @ CA

We are hiring!

Intel team


QUESTIONS?


intel®