Multiprocessor Game Loops:

Lessons from

UNCHARTED 2

AMONG THIEVES

Jason Gregory

Naughty Dog, Inc.

NAUGHTY DOG

Agenda

Agenda

- Goal:

Agenda

- Goal:
  - Learn about **modern multiprocessor game engine update loops**…

Agenda

- Goal:
  - Learn about **modern multiprocessor game engine update loops**…
  - … by investigating Naughty Dog's **train level**.

Agenda

Agenda

- During our investigation, we'll answer the following questions:

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?
  - How does the train **move**?

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?
  - How does the train **move**?
  - How does it tie into the **update of other engine systems**?

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?
  - How does the train **move**?
  - How does it tie into the **update of other engine systems**?
  - How do **player mechanics** and **animation** work on the train?

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?
  - How does the train **move**?
  - How does it tie into the **update of other engine systems**?
  - How do **player mechanics** and **animation** work on the train?
  - How do game object **attachment hierarchies** work?

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?
  - How does the train **move**?
  - How does it tie into the **update of other engine systems**?
  - How do **player mechanics** and **animation** work on the train?
  - How do game object **attachment hierarchies** work?
  - How are **ray and sphere casts** used on the train?

Agenda

- During our investigation, we'll answer the following questions:
  - Why did we make the train itself **dynamic** (not static)?
  - How does the train **move**?
  - How does it tie into the **update of other engine systems**?
  - How do **player mechanics** and **animation** work on the train?
  - How do game object **attachment hierarchies** work?
  - How are **ray and sphere casts** used on the train?
  - How do we utilize the PS3's **parallel computing resources**?

Static or Dynamic?

Could the Train be Static?

- In the past, games with trains in them have used a **static train** approach.
  - Train is actually **stationary**.
  - **Background scrolls by** to produce illusion of movement.
- This solves a lot of problems.
  - Player mechanics, NPC locomotion, weapon mechanics, etc. are all the same as in a “regular” non-moving game level.

Was That Good Enough for Us?

Was That Good Enough for Us?

- No way!

Was That Good Enough for Us?

- No way!
- Stationary train design imposes undue limitations:

Was That Good Enough for Us?

- No way!
- Stationary train design imposes undue limitations:
  - Train must move in a **perfectly straight line**...

Was That Good Enough for Us?

- **No way!**
- Stationary train design imposes undue limitations:
  - Train must move in a **perfectly straight line**...
  - ... or along a broad **circular arc** (constant curvature).

Was That Good Enough for Us?

- No way!
- Stationary train design imposes undue limitations:
  - Train must move in a **perfectly straight line**...
  - ... or along a broad **circular arc** (constant curvature).
  - No **ups and downs** allowed in the track either.

Was That Good Enough for Us?

- **No way!**
- Stationary train design imposes undue limitations:
  - Train must move in a **perfectly straight line**…
  - … or along a broad **circular arc** (constant curvature).
  - No **ups and downs** allowed in the track either.
  - Not nearly as much **fun** to jump and shoot **between** trains.

Was That Good Enough for Us?

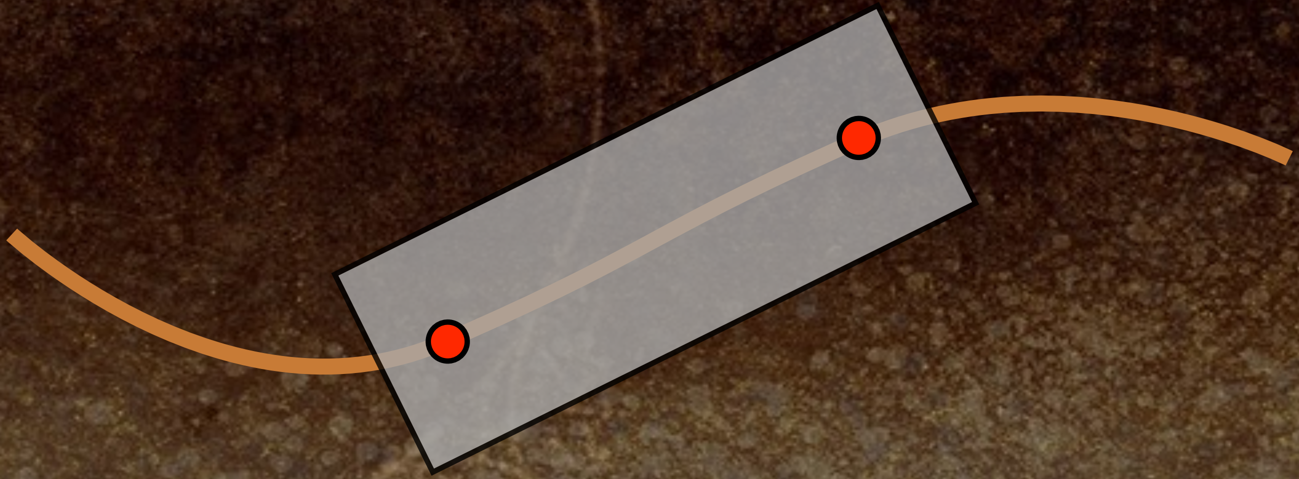
- **No way!**
- Stationary train design imposes undue limitations:
  - Train must move in a **perfectly straight line**...
  - ... or along a broad **circular arc** (constant curvature).
  - No **ups and downs** allowed in the track either.
  - Not nearly as much **fun** to jump and shoot **between** trains.

∴ We decided to go for a fully **dynamic** train.

Train Movement

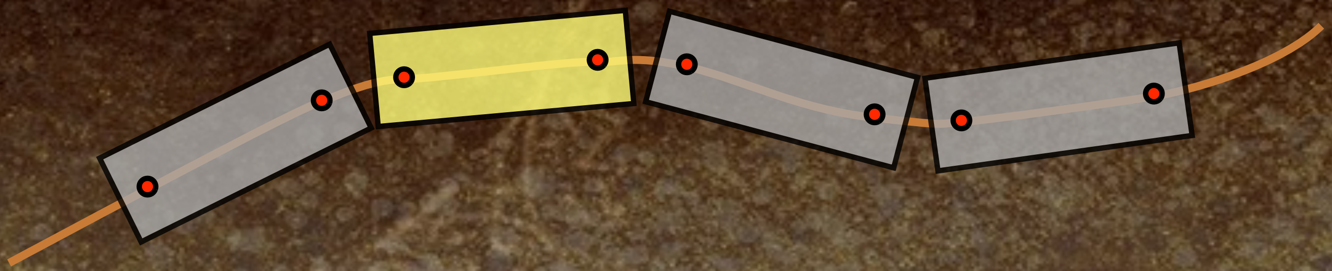
Spline Tracking

- The *Uncharted 2* train follows a **spline**.
  - Catmull-Rom.
  - 2 trackers for realistic movement.



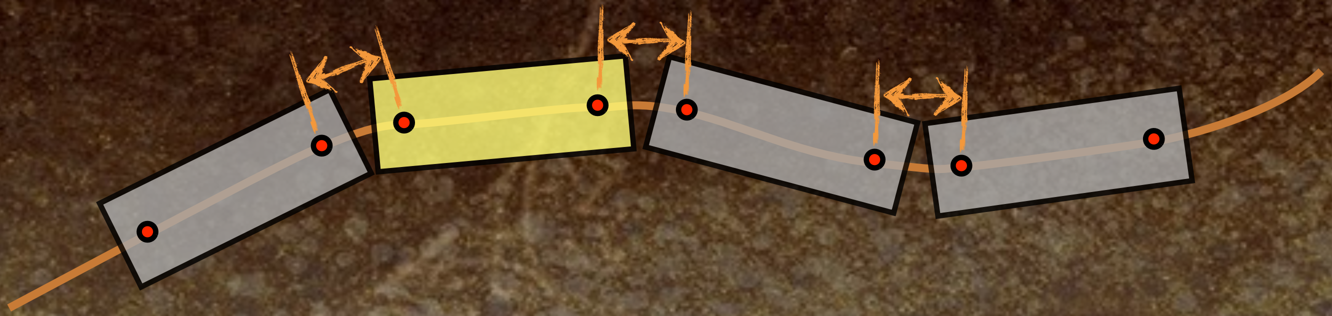
The Master Car

- Each train car is an **independent game object** (GO).
- One car is designated as the **master**.
    - It moves **without regard** for the other cars.
    - Every other car is a **slave**: it simply **maintains proper spacing** with the car(s) in front of and/or behind it.



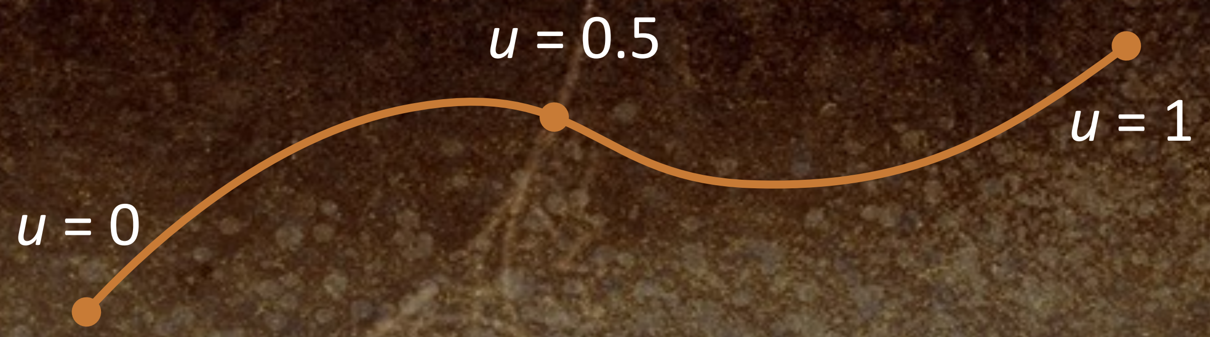
The Master Car

- Each train car is an **independent game object** (GO).
- One car is designated as the **master**.
  - It moves **without regard** for the other cars.
  - Every other car is a **slave**: it simply **maintains proper spacing** with the car(s) in front of and/or behind it.



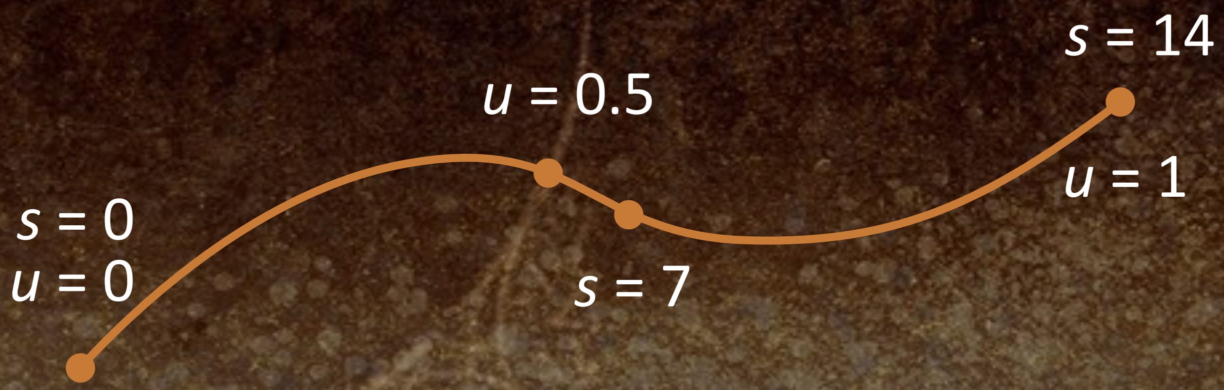
Spacing and Speed

- To maintain proper **spacing**, we must work in terms of **arc length**.
  - Catmull-Roms are parameterized by a unitless quantity u.
  - **Arc length** (s) is not the same thing as u.
  - Careful—must use s for spacing and ds/dt for speed, not u, du/dt.



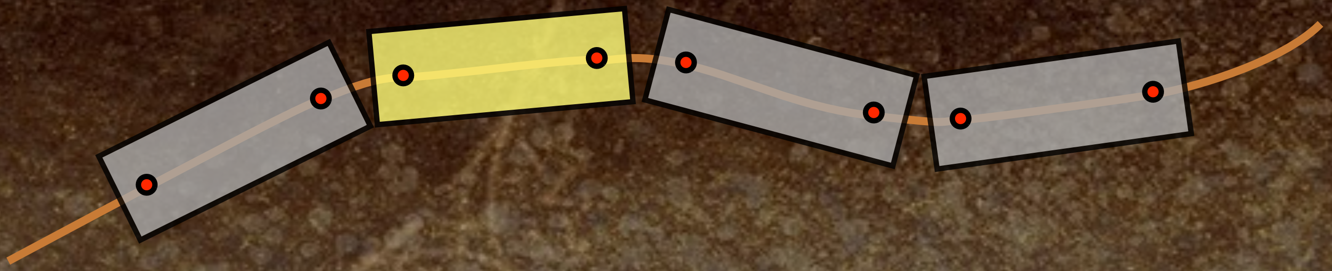
Spacing and Speed

- To maintain proper **spacing**, we must work in terms of **arc length**.
  - Catmull-Roms are parameterized by a unitless quantity *u*.
  - **Arc length** (*s*) is not the same thing as *u*.
  - Careful—must use *s* for spacing and ds/dt for speed, not *u*, du/dt.



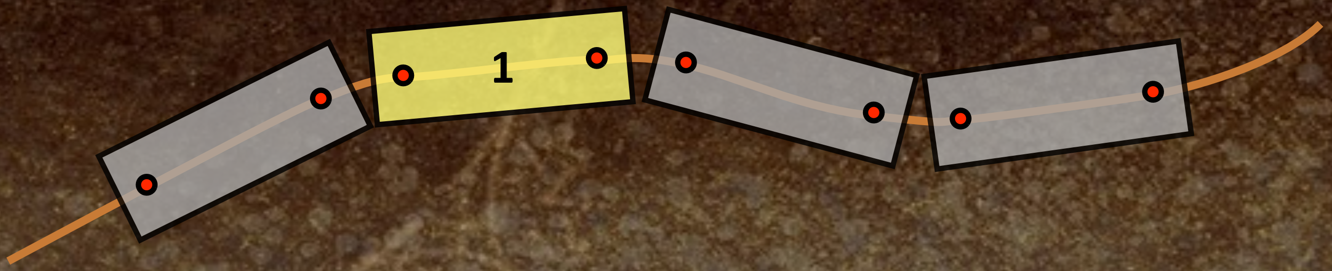
Updating the Cars

- Train car game objects need to update in a **specific order**:
  - **Master** first.
  - Then cars **in front** of master, from master to locomotive.
  - Then cars **behind** master, from master to caboose.



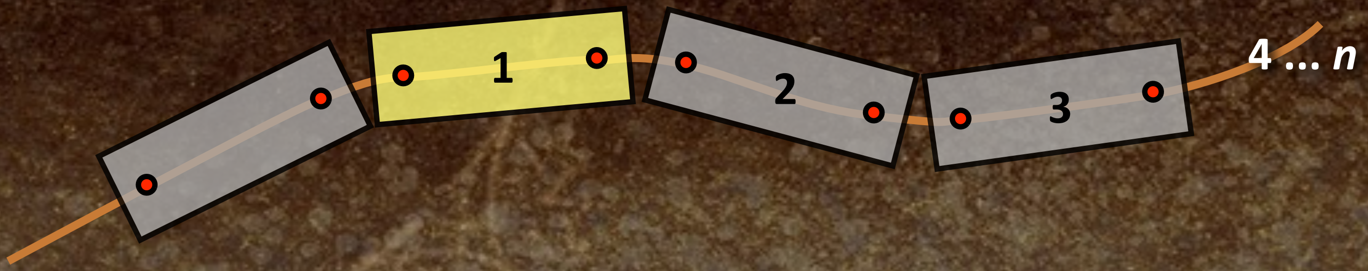
Updating the Cars

- Train car game objects need to update in a **specific order**:
    - **Master** first.
    - Then cars **in front** of master, from master to locomotive.
    - Then cars **behind** master, from master to caboose.



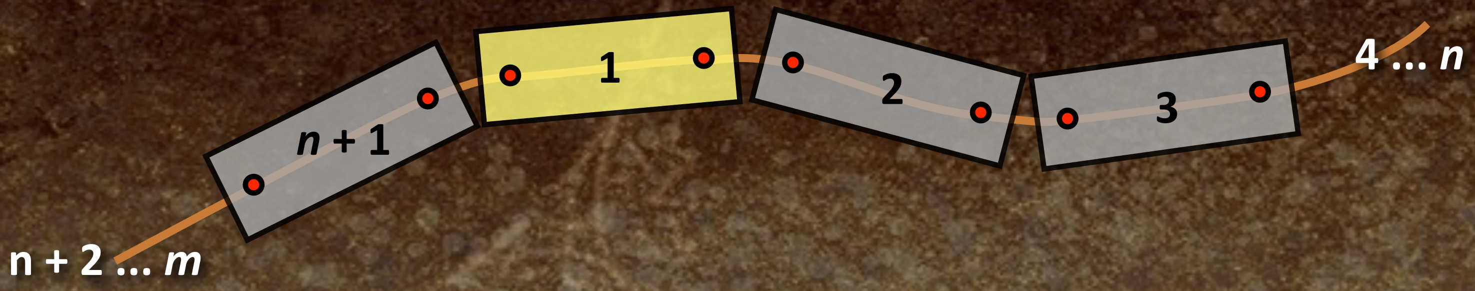
Updating the Cars

- Train car game objects need to update in a **specific order**:
  - **Master** first.
  - Then cars **in front** of master, from master to locomotive.
  - Then cars **behind** master, from master to caboose.



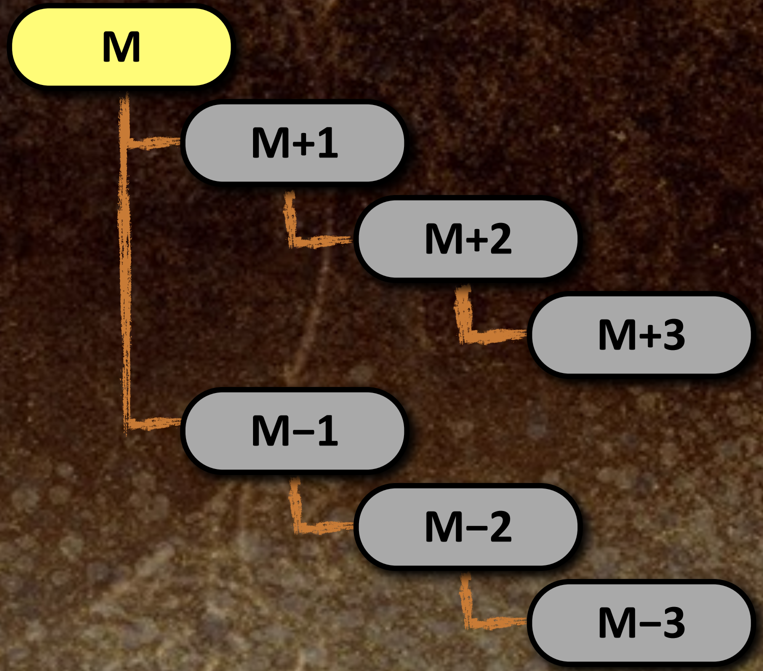
Updating the Cars

- Train car game objects need to update in a **specific order**:
  - **Master** first.
  - Then cars **in front** of master, from master to locomotive.
  - Then cars **behind** master, from master to caboose.



Updating the Cars

- In the *Uncharted 2* engine, game objects are managed as a **tree**.
  - **Children** update **after their parent**.
- For the train…



Teleporting the Train

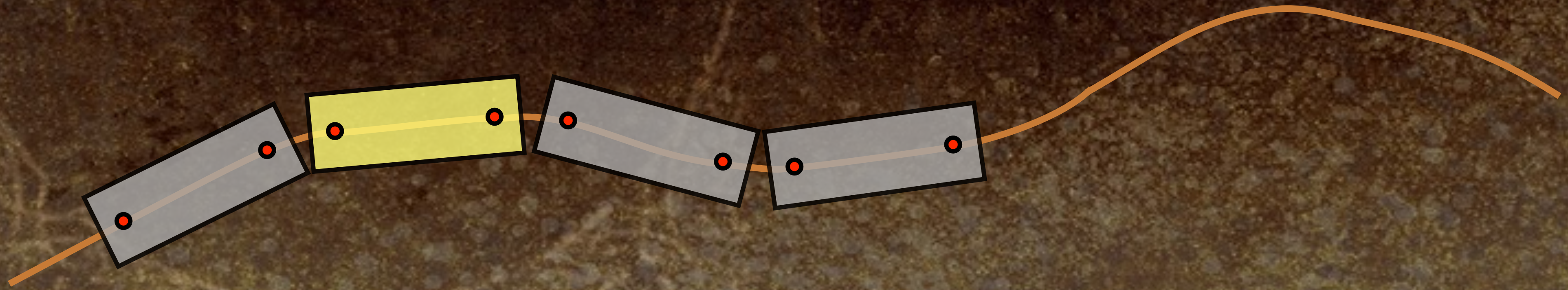
- The train sometimes **teleports** in the game.
  - e.g. To transition from a **looped section** to a **straight-away**.
  - Typically hidden by a **camera cut**.

Teleporting the Train

- The train sometimes **teleports** in the game.
  - e.g. To transition from a **looped section** to a **straight-away**.
  - Typically hidden by a **camera cut**.

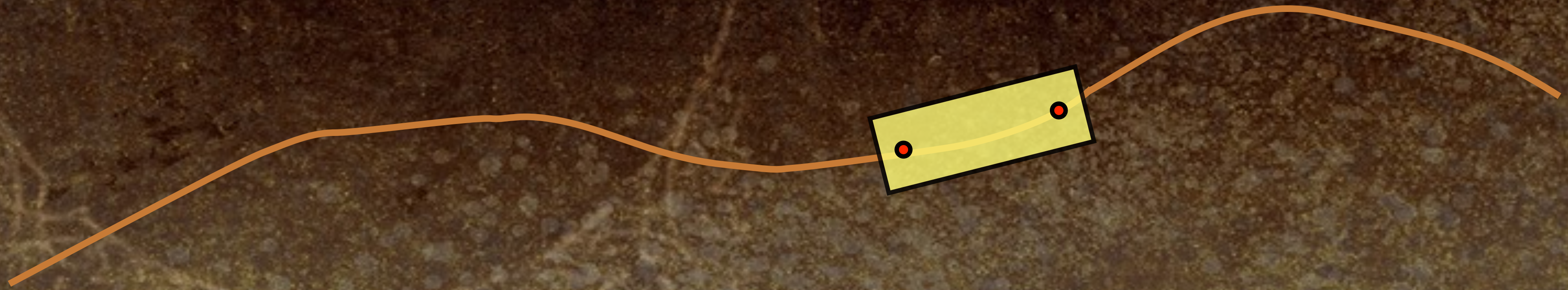
Teleporting the Train

- Problem: Teleport the train such that **whichever car the player is on** moves to a **predefined location** on the track.
  - To implement, **change the master** to be the player's car...
  - ... and **teleport it** to the desired location.
  - All other cars **follow automatically**.



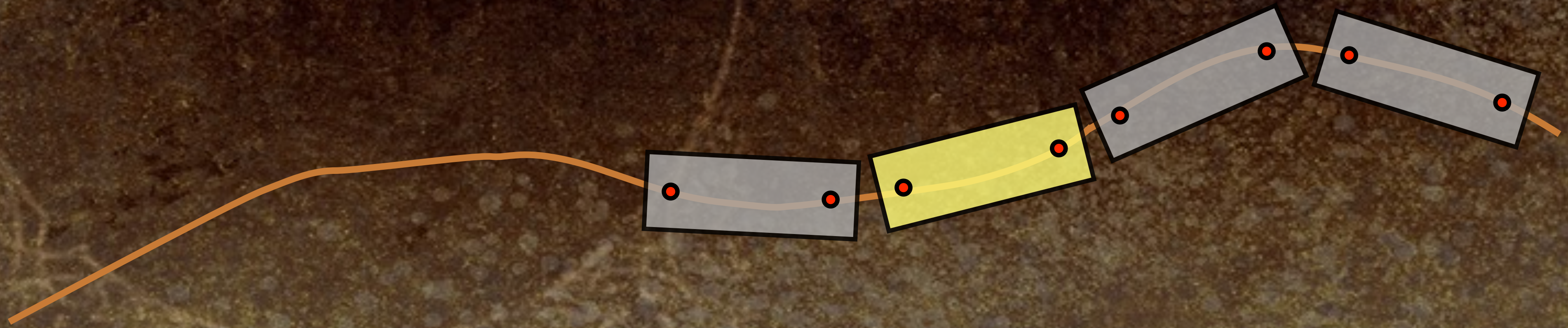
Teleporting the Train

- Problem: Teleport the train such that **whichever car the player is on** moves to a **predefined location** on the track.
  - To implement, **change the master** to be the player's car...
  - ... and **teleport it** to the desired location.
  - All other cars **follow automatically**.



Teleporting the Train

- Problem: Teleport the train such that **whichever car the player is on** moves to a **predefined location** on the track.
  - To implement, **change the master** to be the player's car…
  - … and **teleport it** to the desired location.
  - All other cars **follow automatically**.



Updating Large-Scale Engine Systems

Large-Scale Engine Systems

- Let's define **large-scale engine systems** as:
  - Engine components that operate on **lots of data**...
  - ... and require careful **performance optimization**.
- Examples include:
  - skeletal **animation**,
  - **collision** detection,
  - rigid body **dynamics**,
  - **rendering**, ...

A Simple Approach (That Doesn't Work)

- Most game programmers first learn about the **game loop** from **rendering tutorials**.

A Simple Approach (That Doesn't Work)

- Most game programmers first learn about the **game loop** from **rendering tutorials**.

```
while (!quit)
{
   ReadJoypad();
   UpdateScene();
   DrawScene();
   FlipBuffers();
}
```

A Simple Approach (That Doesn’t Work)

- Since we need to update our game objects anyway…

  UpdateScene() becomes **UpdateGameObjects()**

- In the spirit of good **object-oriented** design, we should let the **game objects** drive the **large-scale engine systems**.

  (Right???)

A Simple Approach (That Doesn't Work)

```
while (!quit)
{
   ReadJoypad();
   UpdateGameObjects();
   // DrawScene();
   FlipBuffers();
}
```

A Simple Approach (That Doesn’t Work)

```
while (!quit)
{
   ReadJoypad();
   UpdateGameObjects();
   // DrawScene();
   FlipBuffers();
}
```

```
void Tank::Update ()
{
   MoveTank();
   AimTurret();
   FireIfNecessary();

   Animate();
   DetectCollisions();
   SimulatePhysics();
   UpdateAudio();
   Draw();
}
```

A Simple Approach (That Doesn’t Work)

- The only problem with this is…

it doesn’t work!

- For one thing, it’s just **not feasible** for some engine systems.
  - e.g. **Collision detection** cannot be done (properly) one object at a time.
    - Need to **solve iteratively**, account for **time of impact** (TOI),
    - optimize collision detection via **spatial subdivision** (e.g. broadphase AABB prune and sweep),
    - optimize dynamics by grouping into **islands**, etc.

A Simple Approach (That Doesn’t Work)

A Simple Approach (That Doesn't Work)

- It's also terribly **inefficient**:

A Simple Approach (That Doesn't Work)

- It's also terribly **inefficient**:
  - potential for **duplicated computations**,

A Simple Approach (That Doesn't Work)

- It's also terribly **inefficient**:
  - potential for **duplicated computations**,
  - possible **reallocation of resources**,

A Simple Approach (That Doesn't Work)

- It's also terribly **inefficient**:
    - potential for **duplicated computations**,
    - possible **reallocation of resources**,
    - poor data and instruction **cache coherence**,

A Simple Approach (That Doesn’t Work)

- It’s also terribly **inefficient**:
  - potential for **duplicated computations**,
  - possible **reallocation of resources**,
  - poor data and instruction **cache coherence**,
  - not conducive to **parallel computation**.

Hardware in the Old Days

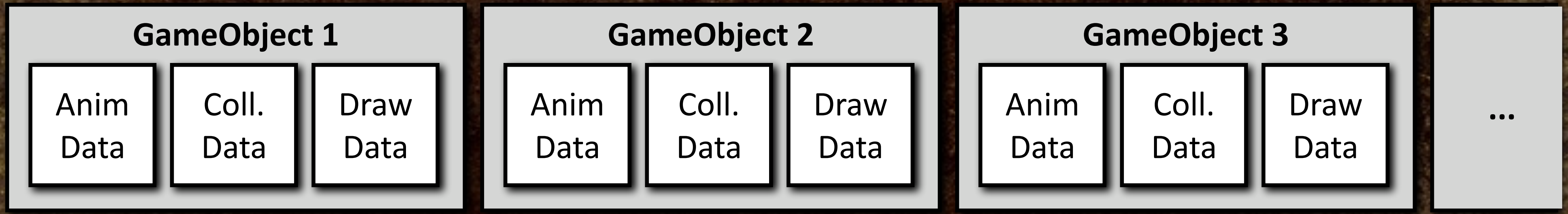
- On an Intel Pentium 286 (cira 1982):
    - **CPU speed** was the bottleneck (6 MHz).
        - ∴ Use **if() tests** to **avoid unnecessary computations**.
    - **Integer instructions** faster than **floating-point**.
        - ∴ Use **fixed-point** numbers, or FPU.
    - More **compute power** provided by **adding transistors**.

Modern Hardware

- On a Cell (PS3) or Xenon (Xbox 360):
  - **Memory access time** is the bottleneck.
    - L1 cache miss = ~60 cycles.
    - L2 cache miss = ~500 cycles.
    - ∴ Organize data in **compact**, **contiguous** blocks.
    - Use **struct of arrays** rather than array of structs.
  - **Pipelined** architecture = **branching** & **data conversion** are **slow**.
    - ∴ Use **duplicated computations** to **avoid if() tests**.
  - **Compute power** is provided via **parallelism**.

Optimizing Large-Scale Updates

Inefficient



GameObject 1

Anim Data

Coll. Data

Draw Data

GameObject 2

Anim Data

Coll. Data

Draw Data

GameObject 3

Anim Data

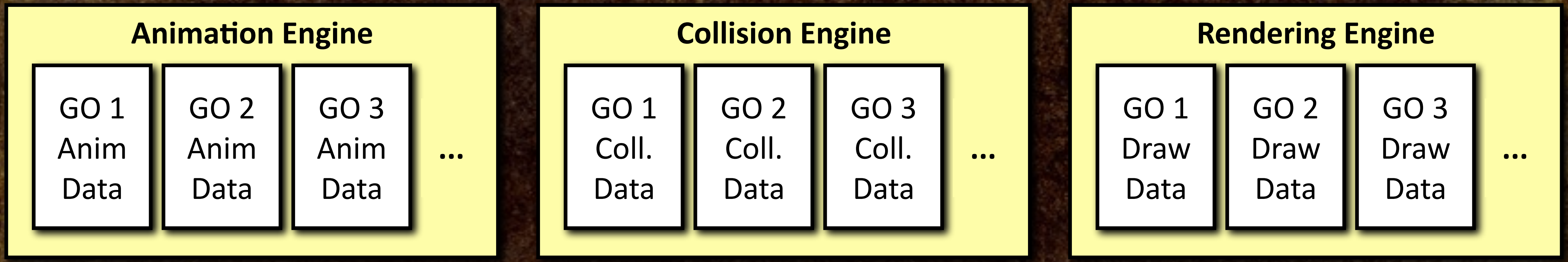
Coll. Data

Draw Data

...

Optimizing Large-Scale Updates

Much better!



Animation Engine

GO 1 Anim Data | GO 2 Anim Data | GO 3 Anim Data | …

Collision Engine

GO 1 Coll. Data | GO 2 Coll. Data | GO 3 Coll. Data | …

Rendering Engine

GO 1 Draw Data | GO 2 Draw Data | GO 3 Draw Data | …

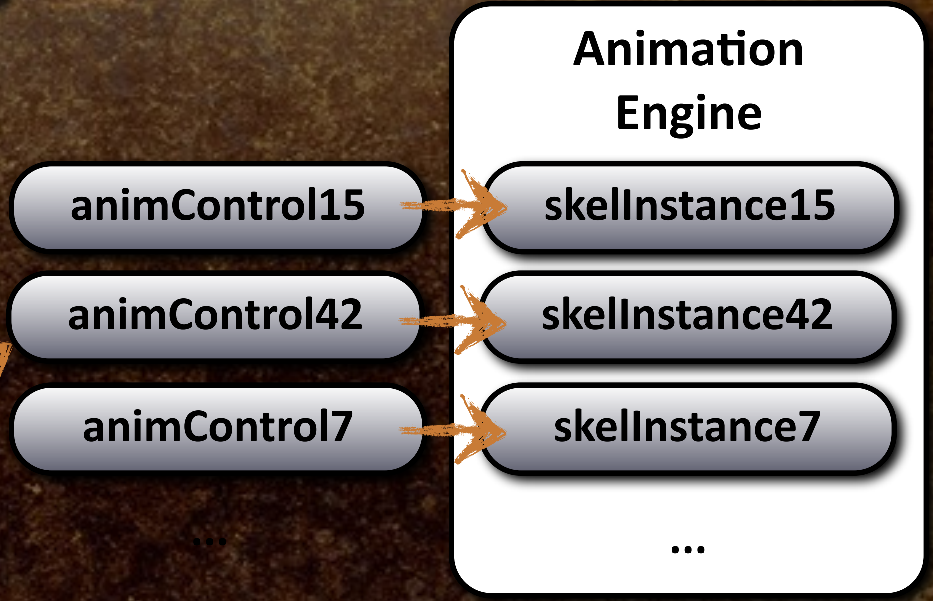
Optimizing Large-Scale Updates

- Game objects **do not own** their large-scale system data…
  - … they **point to** data that is **owned** by the large-scale systems.
- Game objects should not mutate their large-scale system data…
  - … they should only **request** changes.
- That way, each large-scale system can:
  - **apply any requested changes** as part of its **batch update**,
  - organize its data in **whatever manner is most efficient**.

Optimizing Large-Scale Updates

```
class Tank : public GameObject
{
public:
   // interface...

private:
   // components -- owned by engine subsystems
   AnimControl*    m_pAnimControl;
   RigidBody*      m_pRigidBody;
   MeshInstance* m_pRenderable;
};
```

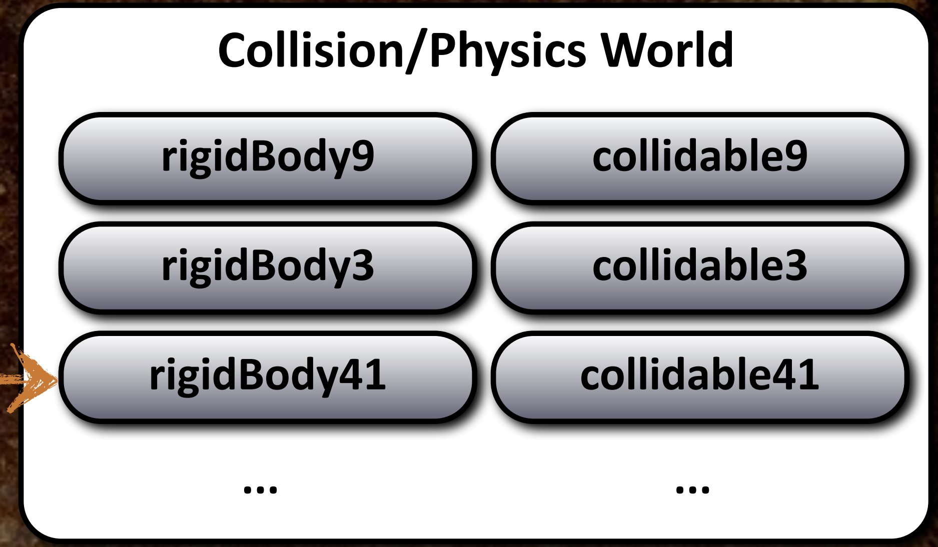
...



Optimizing Large-Scale Updates

```
class Tank : public GameObject
{
public:
   // interface...

private:
   // components -- owned by engine subsystems
   AnimControl*    m_pAnimControl;
   RigidBody*      m_pRigidBody;
   MeshInstance* m_pRenderable;
};
```

...



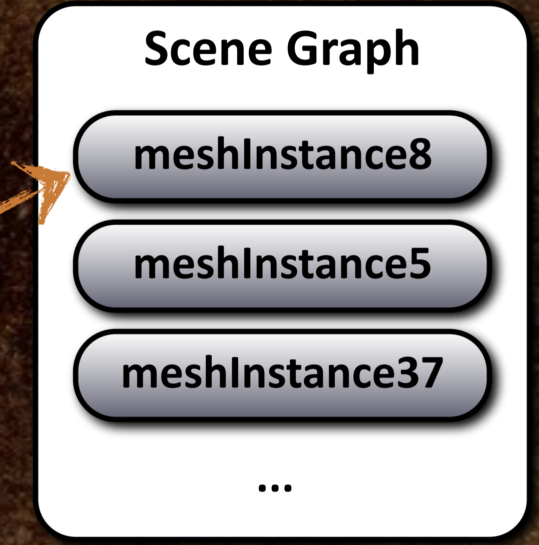
Collision/Physics World

rigidBody9	collidable9
rigidBody3	collidable3
rigidBody41	collidable41
...	...

Optimizing Large-Scale Updates

```
class Tank : public GameObject
{
public:
   // interface...

private:
   // components -- owned by engine subsystems
   AnimControl*    m_pAnimControl;
   RigidBody*      m_pRigidBody;
   MeshInstance* m_pRenderable;
};
```

...



```
while (!quit)
{
   ReadJoypad();

   g_gameWorld.UpdateAllGameObjects();
   //for (each GameObject* pGo)
   //    pGo->Update();

   g_animationEngine.Update();
   g_collisionWorld.DetectCollisions();
   g_physicsWorld.Simulate();
   g_audioEngine.Update();
   g_renderingEngine.DrawAndFlipBuffers();
}
```

Player Mechanics and Animation

Player Mechanics on a Moving Object

- Player's position and orientation represented as an **attached frame of reference**:
  - reference to **parent game object**,
  - **local transform** (relative to parent),
  - cached **world-space transform** (with dirty flag).
- Not quite as simple as it may sound!
  - Just **switching to attached frames** got us ~60% of the way there.
  - Lots of **bugs** to fix.
  - Special-case handling of **transitions** between attached frames.

Animation Pipeline Review

1. Animation Update
   - Update **clocks**, trigger **keyframed events**, detect **end-of-animation** conditions, take **transitions** to new animation states.
2. Pose Blending Phase
   - **Extract poses** from anim clips, **blend** individual joint poses.
     - Generates **local joint transforms** (parent-relative).

Animation Pipeline Review

3. Global Pose Phase
    - Walk hierarchy, calculate **global joint transforms** (model-space or world-space) and **matrix palette** (input to renderer).
4. Post-Processing
    - Apply **IK**, **procedural animation**, etc.
        - Generates new **local** joint transforms.
        - Recalculate global poses if necessary (re-run phase 3).

Game Object Hooks

- It's convenient to provide **game objects** with **hooks** into the various phases of the animation update.
- What we do at Naughty Dog:
  - GameObject::**Update**()
    - Regular GO update; runs before animation.
  - GameObject::**PostAnimUpdate**()
    - Runs after Animation Phase 1.
    - Game objects may respond to animation events, force transitions to new animation states, etc.

Game Object Hooks

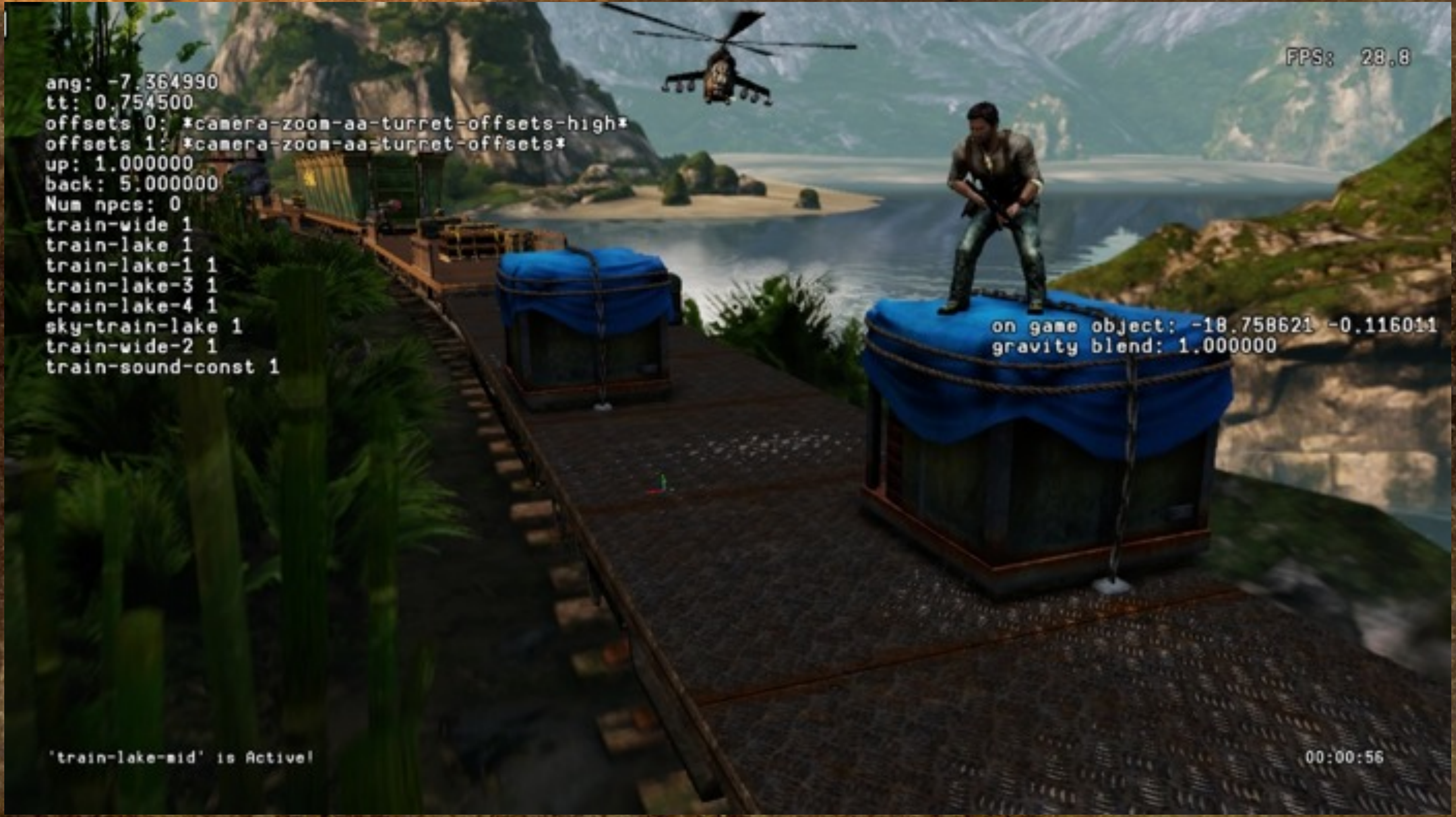
- GameObject::**PostAnimBlending**()
  - Runs after Animation Phase 2.
  - Game objects can apply procedural animation in local space.
- GameObject::**PostJointUpdate**()
  - Runs after Animation Phase 3.
  - This is the first time during the frame that the **global transform** of every joint is known.
  - Game objects can **apply IK**, or use the global space joint transforms in other ways.

```
while (!quit)
{
    // ...
    g_gameWorld.UpdateAllGameObjects();

    g_animationEngine.RunPhase1();
    g_gameWorld.CallPostAnimUpdateHooks();

    g_animationEngine.RunPhase2();
    g_gameWorld.CallPostAnimBlendingHooks();

    g_animationEngine.RunPhase3();
    g_gameWorld.CallPostJointUpdateHooks();
    // ...
```

Object Attachment Hierarchy

Bucketed Updates

Game Object Attachment Hierarchy



```
ang: -7.364990
tt: 0.754500
offsets 0: *camera-zoom-aa-turret-offsets-high*
offsets 1: *camera-zoom-aa-turret-offsets*
up: 1.000000
back: 5.000000
Num npcs: 0
train-wide 1
train-lake 1
train-lake-1 1
train-lake-3 1
train-lake-4 1
sky-train-lake 1
train-wide-2 1
train-sound-const 1
```

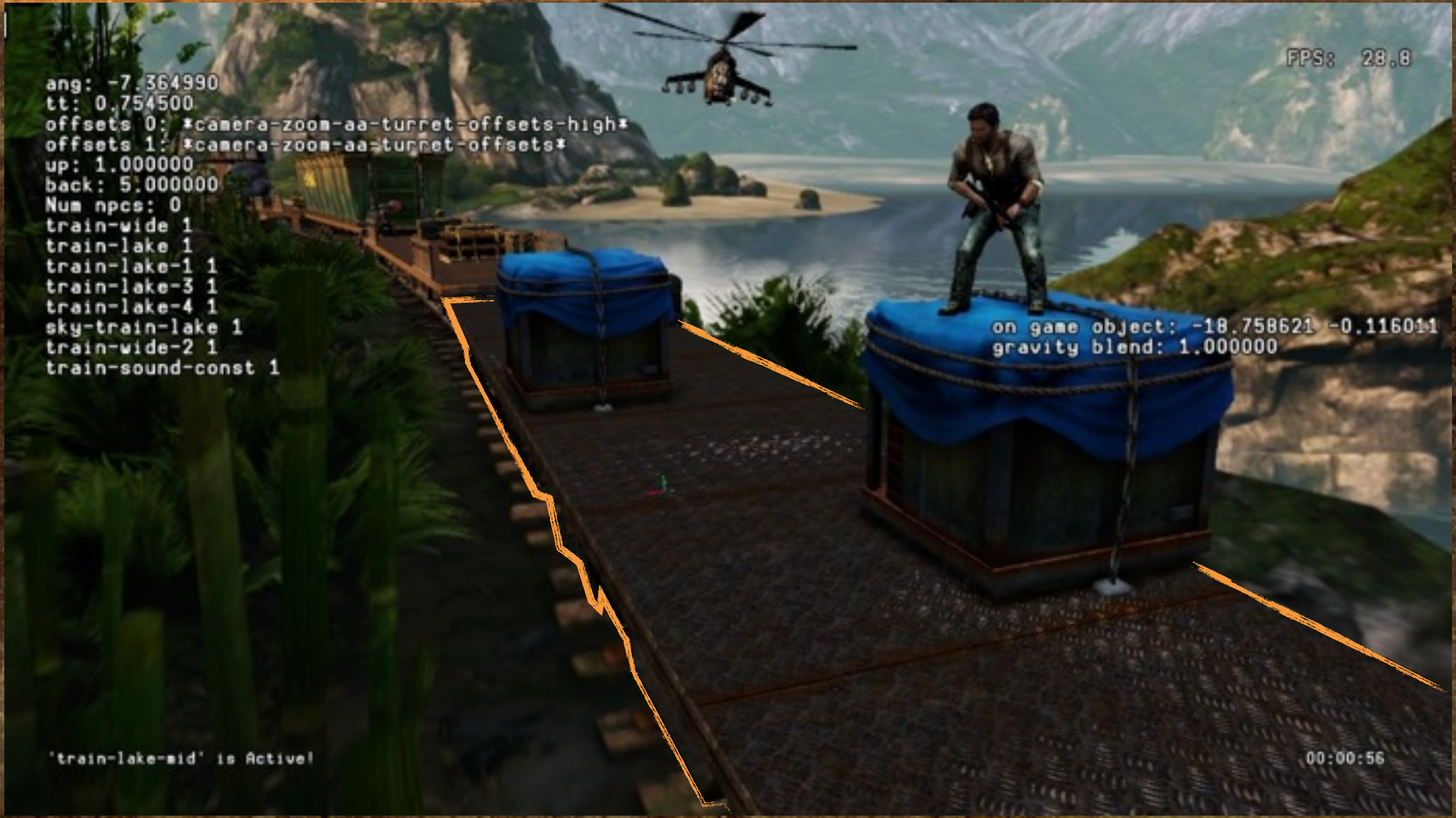
```
FPS:  28.8
```

```
on game object: -18.758621 -0.116011
gravity blend: 1.000000
```

```
'train-lake-mid' is Active!
```

```
00:00:56
```

Game Object Attachment Hierarchy



```
ang: -7.364990
tt: 0.754500
offsets 0: *camera-zoom-aa-turret-offsets-high*
offsets 1: *camera-zoom-aa-turret-offsets*
up: 1.000000
back: 5.000000
Num npcs: 0
train-wide 1
train-lake 1
train-lake-1 1
train-lake-3 1
train-lake-4 1
sky-train-lake 1
train-wide-2 1
train-sound-const 1
```

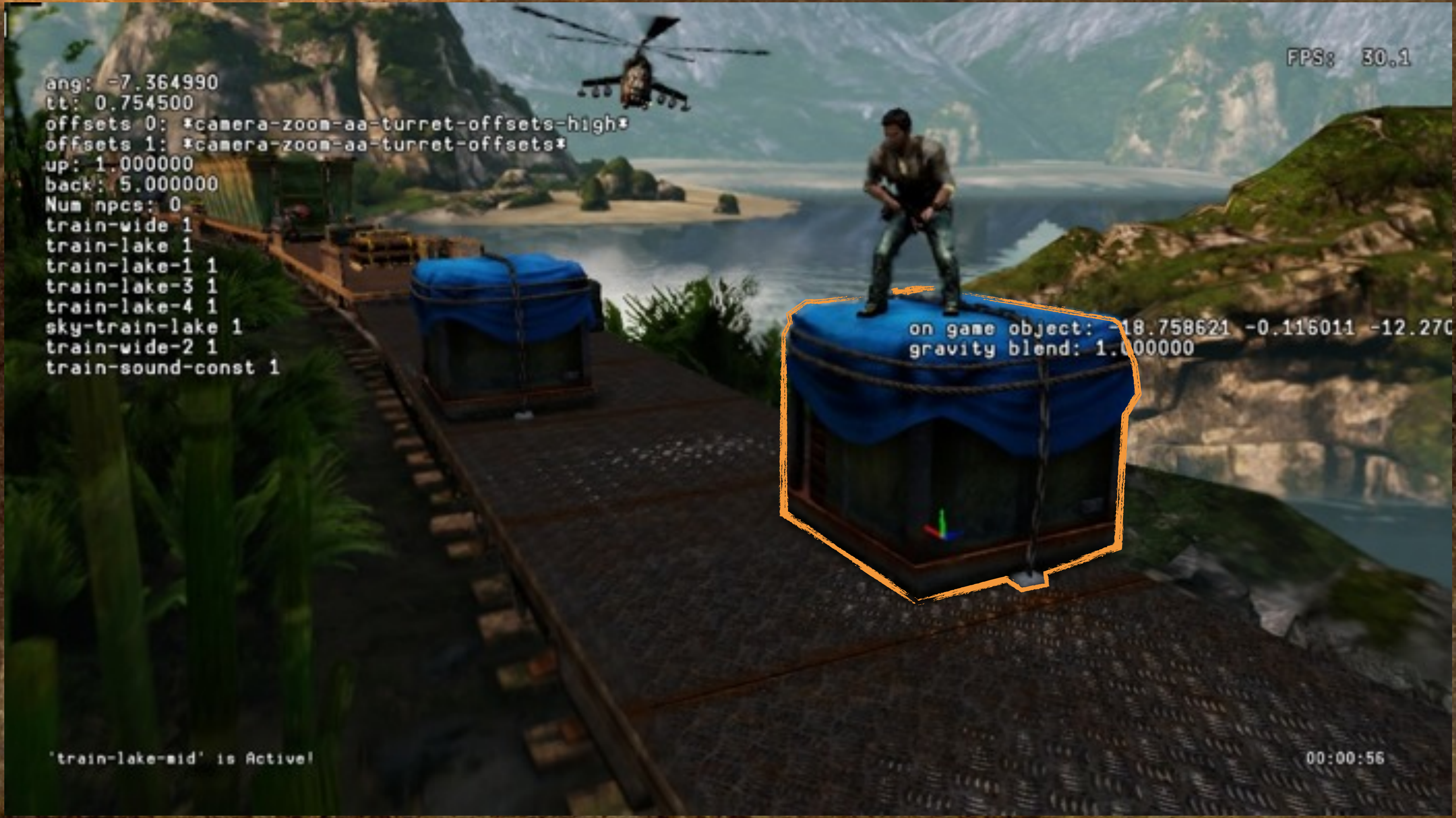
```
FPS:  28.8
```

```
on game object: -18.758621 -0.116011
gravity blend: 1.000000
```

```
'train-lake-mid' is Active!
```

```
00:00:56
```

Game Object Attachment Hierarchy



```
ang: -7.364990
tt: 0.754500
offsets 0: *camera-zoom-aa-turret-offsets-high*
offsets 1: *camera-zoom-aa-turret-offsets*
up: 1.000000
back: 5.000000
Num npcs: 0
train-wide 1
train-lake 1
train-lake-1 1
train-lake-3 1
train-lake-4 1
sky-train-lake 1
train-wide-2 1
train-sound-const 1
```

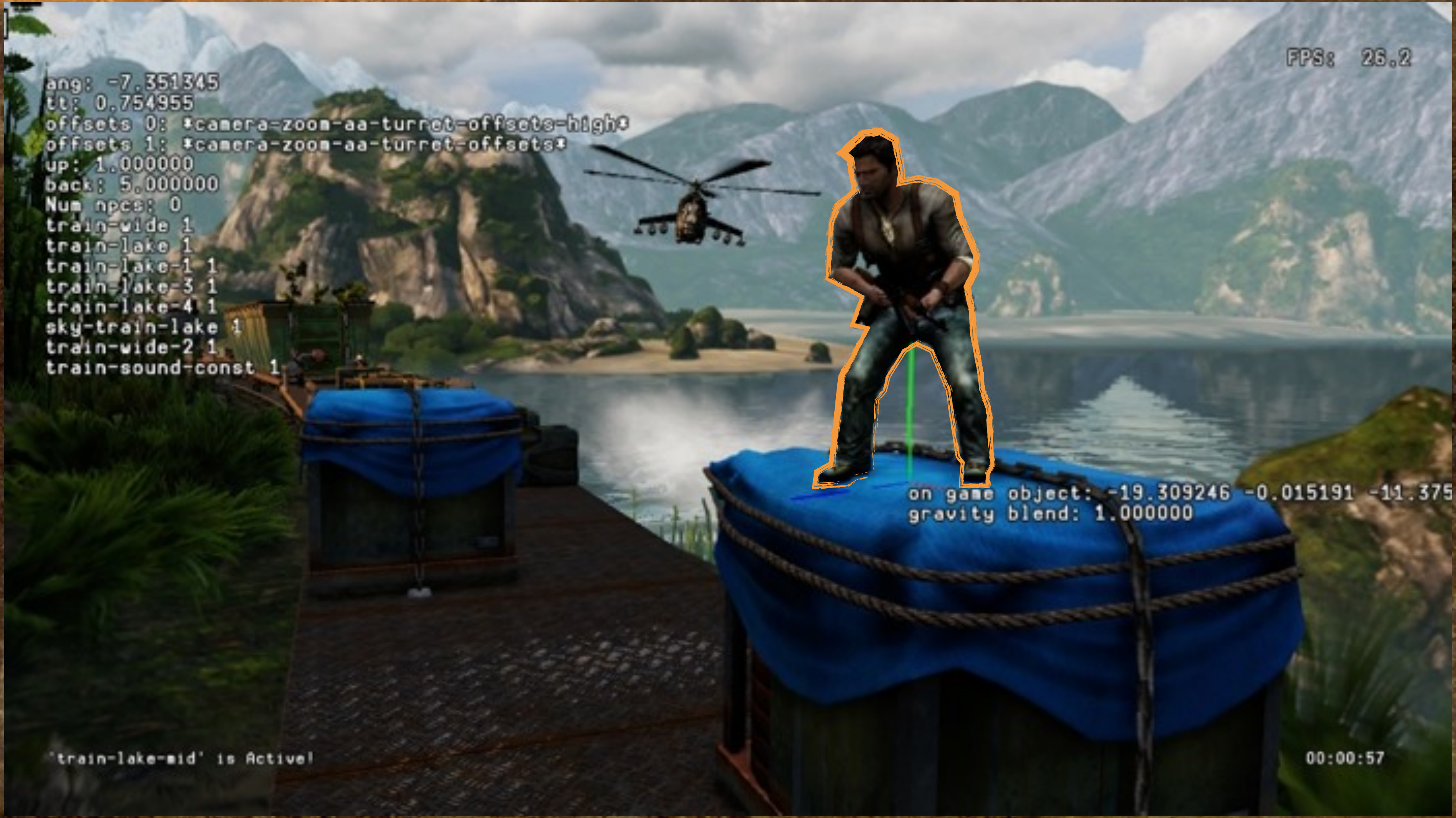
```
FPS:  30.1
```

```
on game object: -18.758621 -0.116011 -12.270
gravity blend: 1.000000
```

```
'train-lake-mid' is Active!
```

```
00:00:56
```

Game Object Attachment Hierarchy



```
ang: -7.351345
tt: 0.754955
offsets 0: *camera-zoom-aa-turret-offsets-high*
offsets 1: *camera-zoom-aa-turret-offsets*
up: 1.000000
back: 5.000000
Num npcs: 0
train-wide 1
train-lake 1
train-lake-1 1
train-lake-3 1
train-lake-4 1
sky-train-lake 1
train-wide-2 1
train-sound-const 1
```

```
FPS: 26.2
```

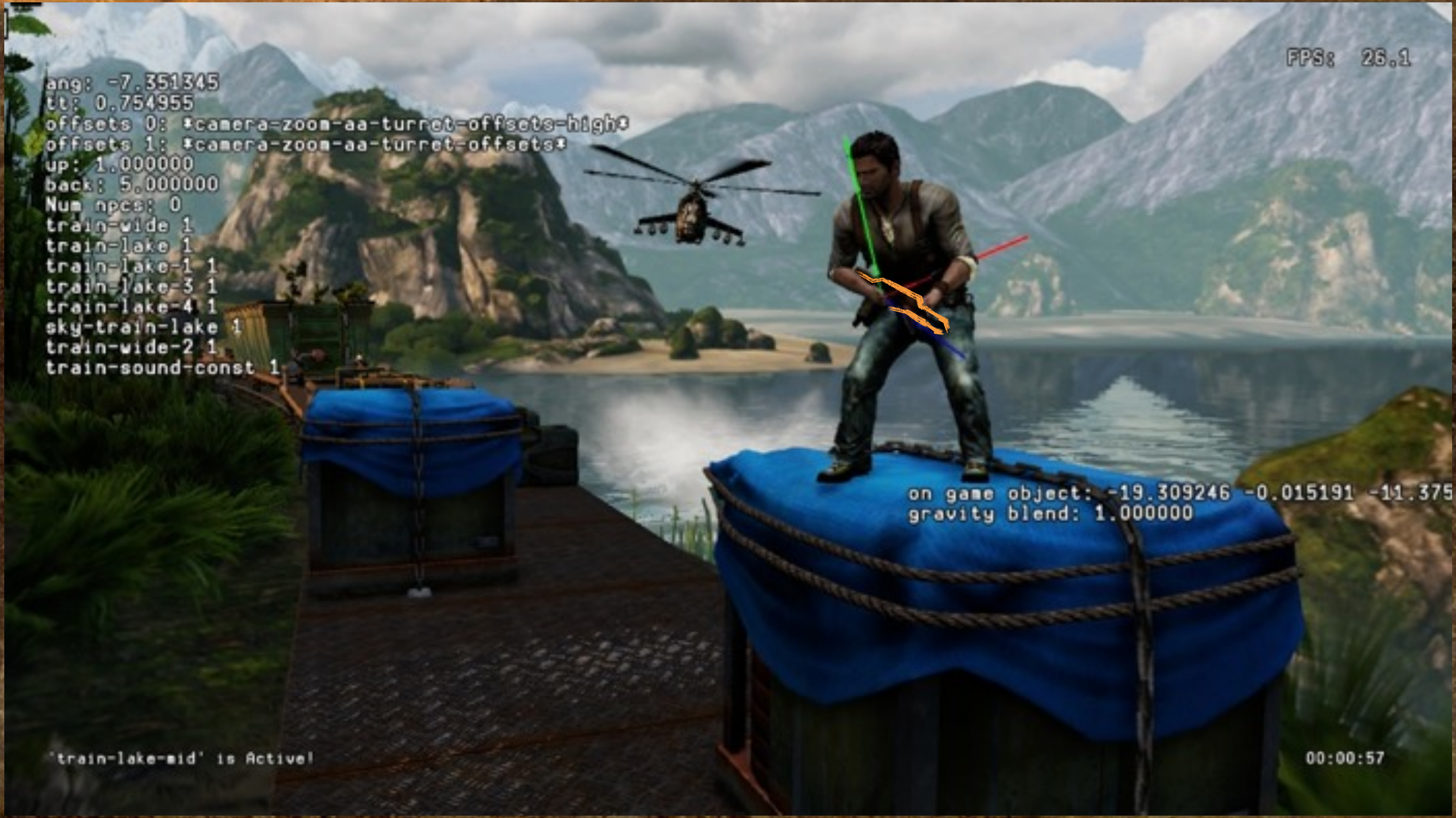
```
on game object: -19.309246 -0.015191 -11.375
gravity blend: 1.000000
```

```
'train-lake-mid' is Active!
```

```
00:00:57
```

Wednesday, January 27, 2010

Game Object Attachment Hierarchy



```
ang: -7.351345
tt: 0.754955
offsets 0: *camera-zoom-aa-turret-offsets-high*
offsets 1: *camera-zoom-aa-turret-offsets*
up: 1.000000
back: 5.000000
Num npcs: 0
train-wide 1
train-lake 1
train-lake-1 1
train-lake-3 1
train-lake-4 1
sky-train-lake 1
train-wide-2 1
train-sound-const 1
```

```
FPS: 26.1
```

```
on game object: -19.309246 -0.015191 -11.375
gravity blend: 1.000000
```

```
'train-lake-mid' is Active!
```

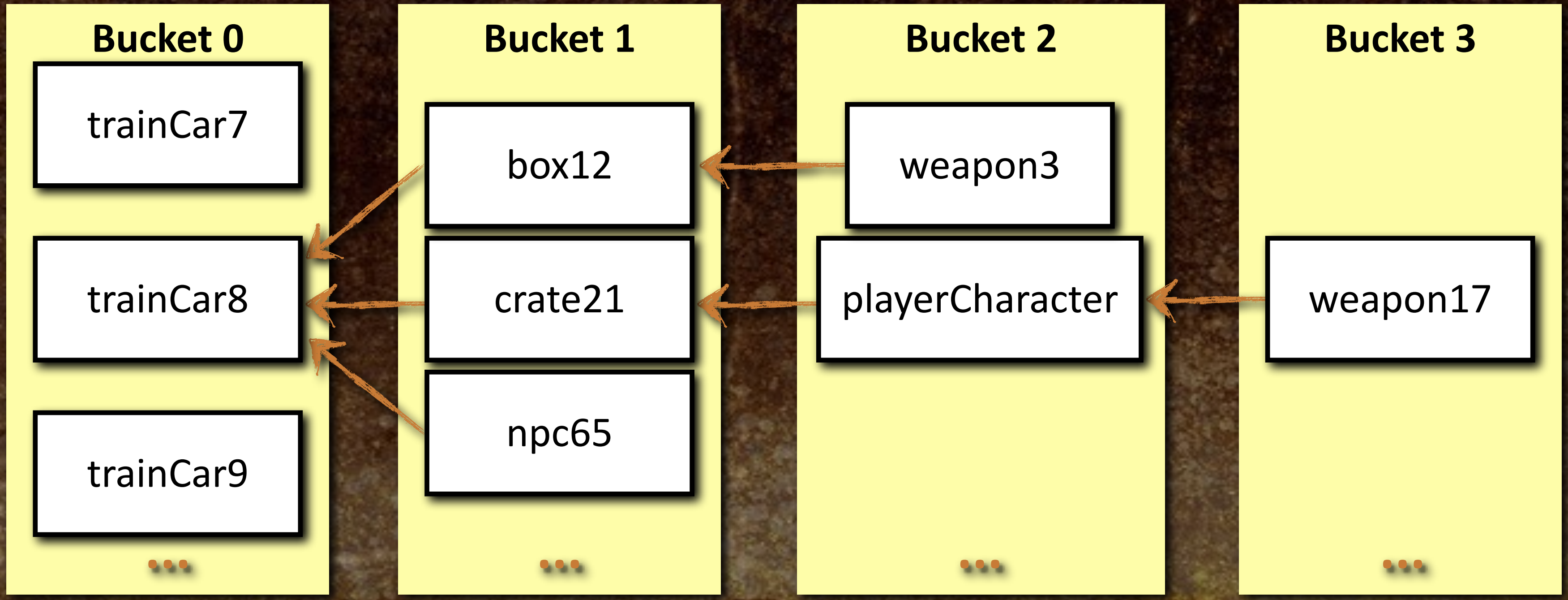
```
00:00:57
```

Bucketed Game Object Updates

- Can implement this by updating game objects in **buckets**.
  - Game object dependencies represented by a **dependency tree**.

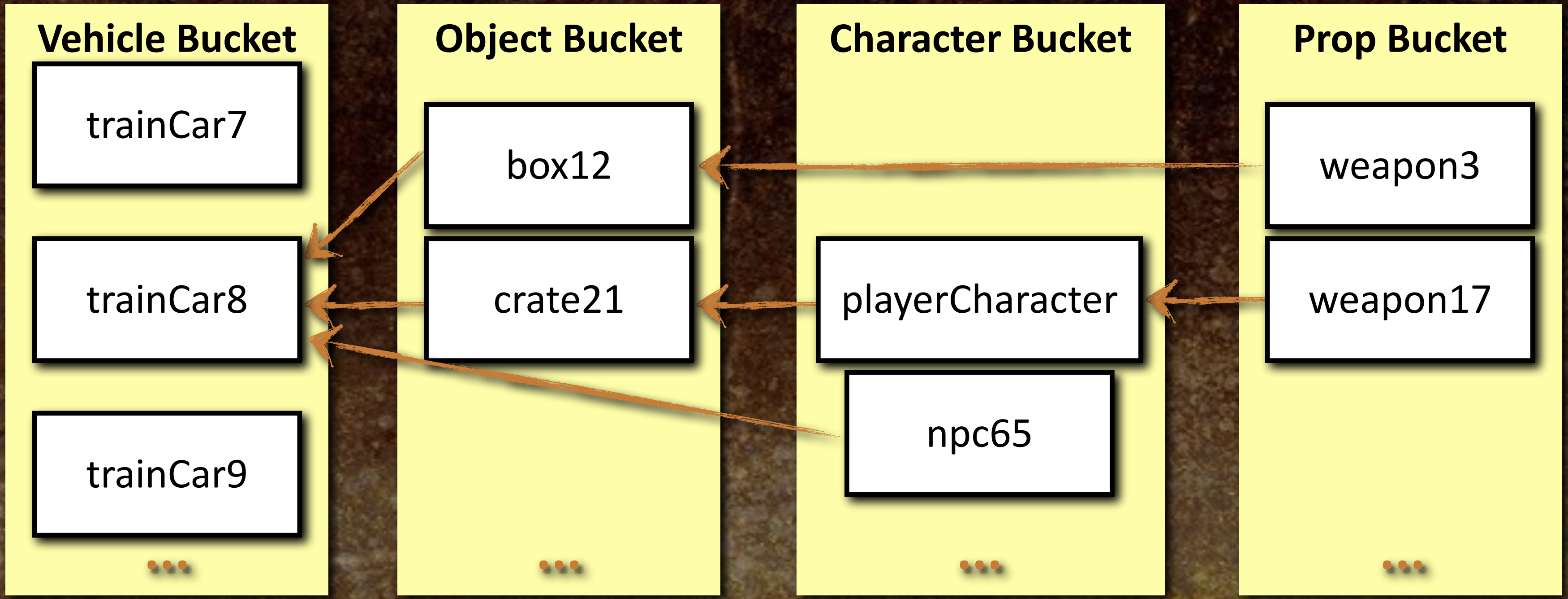
Bucketed Game Object Updates

- Can implement this by updating game objects in **buckets**.
  - Game object dependencies represented by a **dependency tree**.
  - Each **bucket** corresponds to one **level** of the tree.



Bucketed Game Object Updates

- Simpler to group objects into **pre-determined** buckets.



Bucketed Game Object Updates

```
while (!quit)
{
   // ...
   for (each bucket)
   {
      g_gameObjectMgr.UpdateObjects(bucket);
      AnimateBucket(bucket);
   }
   g_collphysWorld.CollideAndSimulate(dt);
   g_audioEngine.Update(dt);
   g_renderingEngine.DrawAndFlipBuffers(dt);
}
```

```
void AnimateBucket(bucket)
{
   g_animationEngine.UpdateClocks(bucket);
   for (each GameObject* pGo in bucket)
      pGo->PostAnimUpdate();

   g_animationEngine.CalcLocalPoses(bucket);
   for (each GameObject* pGo in bucket)
      pGo->PostAnimBlending();

   g_animationEngine.CalcGlobalPoses(bucket);
   for (each GameObject* pGo in bucket)
      pGo->PostJointUpdate();
}
```

Ray and Sphere Casts

Ray and Sphere Casts

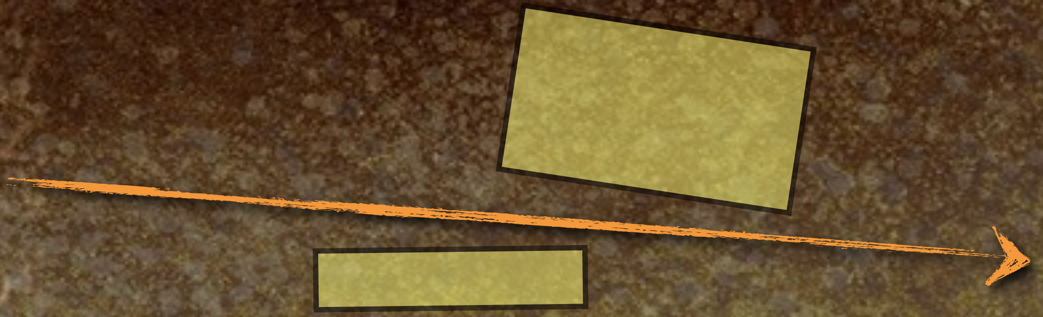
- A **collision cast** is an **instantaneous collision query**.
  - Input:
    - **Snapshot** of collision geometry in game world **at time *t***.
    - One or more **rays / moving spheres** (capsules) to cast.
  - Output:
    - Would any of the rays/spheres **strike anything**?
    - If so, **what**?

Ray and Sphere Casts

- A **collision cast** is an **instantaneous collision query**.
  - Input:
    - **Snapshot** of collision geometry in game world **at time *t***.
    - One or more **rays / moving spheres** (capsules) to cast.
  - Output:
    - Would any of the rays/spheres **strike anything**?
    - If so, **what**?

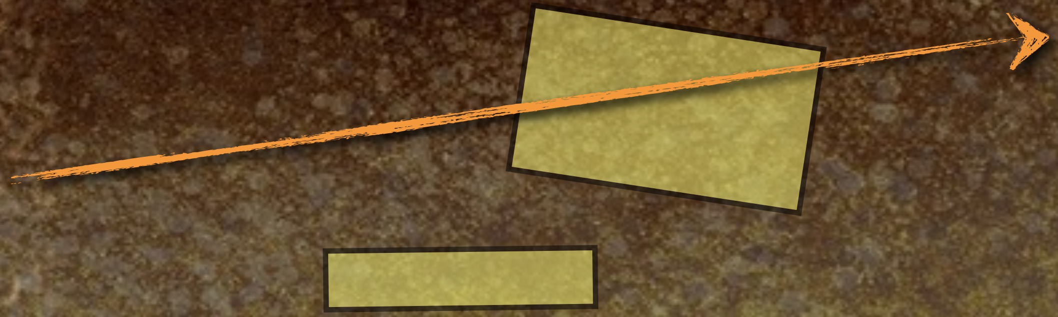
Ray and Sphere Casts

- A **collision cast** is an **instantaneous collision query**.
  - Input:
    - **Snapshot** of collision geometry in game world **at time *t***.
    - One or more **rays / moving spheres** (capsules) to cast.
  - Output:
    - Would any of the rays/spheres **strike anything**?
    - If so, **what**?



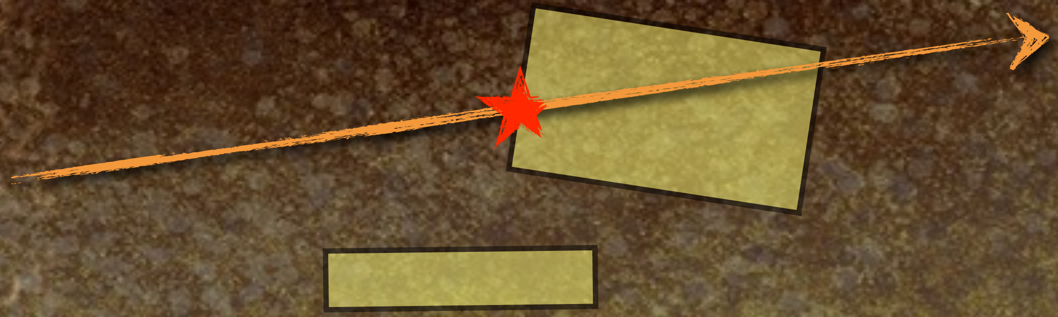
Ray and Sphere Casts

- A **collision cast** is an **instantaneous collision query**.
  - Input:
    - **Snapshot** of collision geometry in game world **at time *t***.
    - One or more **rays / moving spheres** (capsules) to cast.
  - Output:
    - Would any of the rays/spheres **strike anything**?
    - If so, **what**?



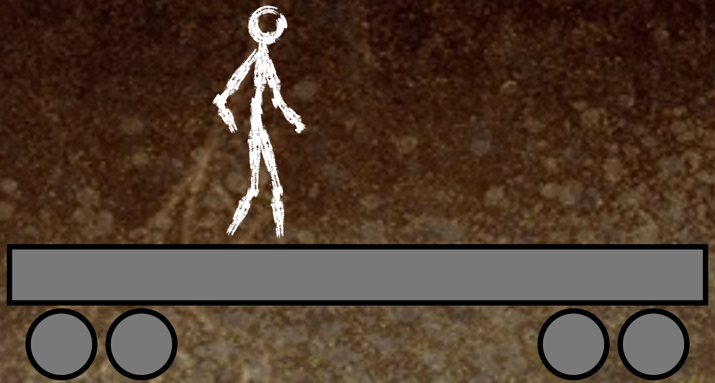
Ray and Sphere Casts

- A **collision cast** is an **instantaneous collision query**.
  - Input:
    - **Snapshot** of collision geometry in game world **at time *t***.
    - One or more **rays / moving spheres** (capsules) to cast.
  - Output:
    - Would any of the rays/spheres **strike anything**?
    - If so, **what**?



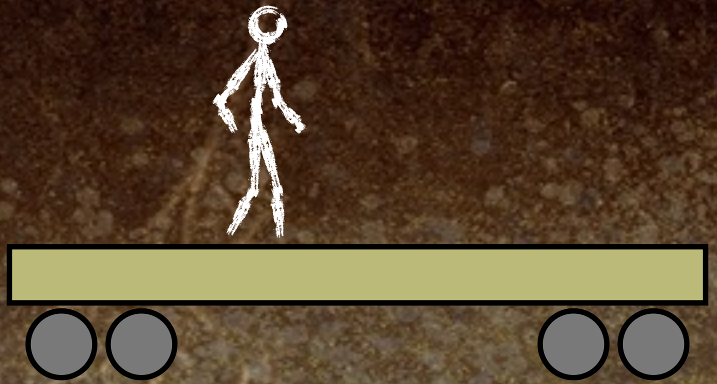
Uses for Collision Casting

- **Player** and **NPCs** use downward casts to determine what **surface** they are standing on.
- **NPCs** use ray casts to answer **line of sight** questions.
- **Weapons** use ray casts to determine **bullet impacts**.
- and the list goes on...



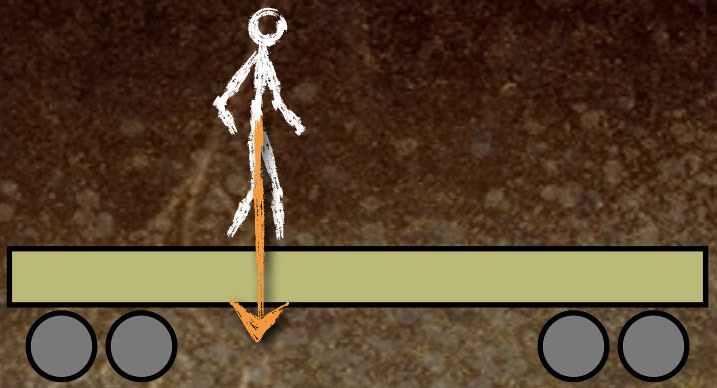
Uses for Collision Casting

- **Player** and **NPCs** use downward casts to determine what **surface** they are standing on.
- **NPCs** use ray casts to answer **line of sight** questions.
- **Weapons** use ray casts to determine **bullet impacts**.
- and the list goes on...



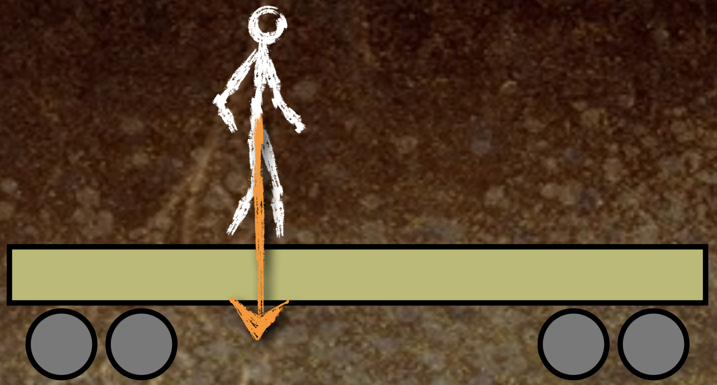
Uses for Collision Casting

- **Player** and **NPCs** use downward casts to determine what **surface** they are standing on.
- **NPCs** use ray casts to answer **line of sight** questions.
- **Weapons** use ray casts to determine **bullet impacts**.
- and the list goes on…



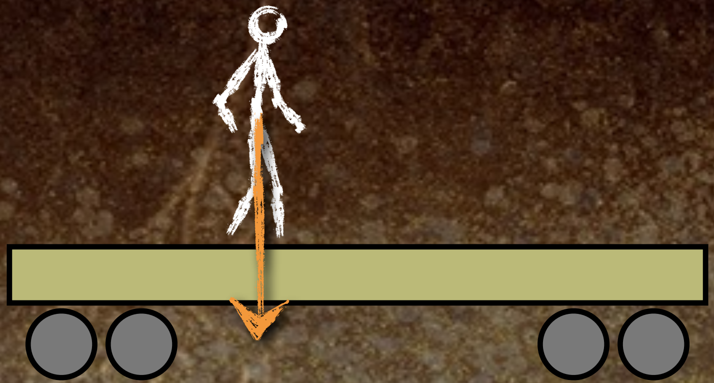
Uses for Collision Casting

- **Player** and **NPCs** use downward casts to determine what **surface** they are standing on.
- **NPCs** use ray casts to answer **line of sight** questions.



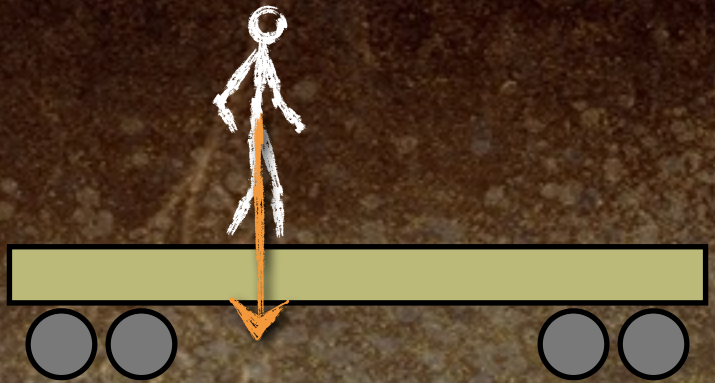
Uses for Collision Casting

- **Player** and **NPCs** use downward casts to determine what **surface** they are standing on.
- **NPCs** use ray casts to answer **line of sight** questions.
- **Weapons** use ray casts to determine **bullet impacts**.



Uses for Collision Casting

- **Player** and **NPCs** use downward casts to determine what **surface** they are standing on.
- **NPCs** use ray casts to answer **line of sight** questions.
- **Weapons** use ray casts to determine **bullet impacts**.
- and the list goes on...



Inter-Game-Object Queries and Synchronization

- **Synchronization problems** can arise whenever:
  - game object A…
  - … **queries** game object B.
- Not just limited to **ray and sphere casts**.
  - Any kind of **inter-object query** can be affected.

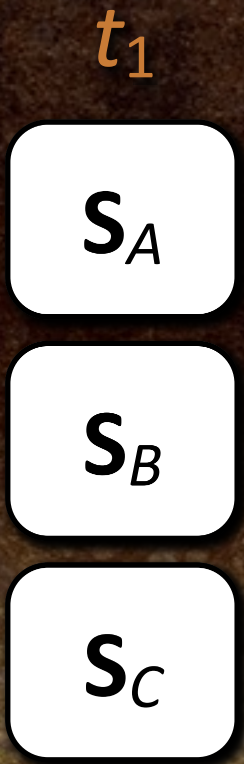
Game Object State Vectors

- Can think of a game object as a heterogeneous "**state vector**."
- The **state** of the i^{th} game object is a **vector function** of **time** t.

$$\begin{aligned} \mathbf{S}_i(t) = [\ & \mathbf{r}_i(t), \\ & \mathbf{v}_i(t), \\ & m_i, \ldots, \\ & health_i(t), \\ & ammo_i(t), \ldots\ ] \end{aligned}$$

Bucketing and Game Object State Vectors

- Theoretically, the state vectors of all game objects are updated **instantaneously** and **in parallel**.



t_1

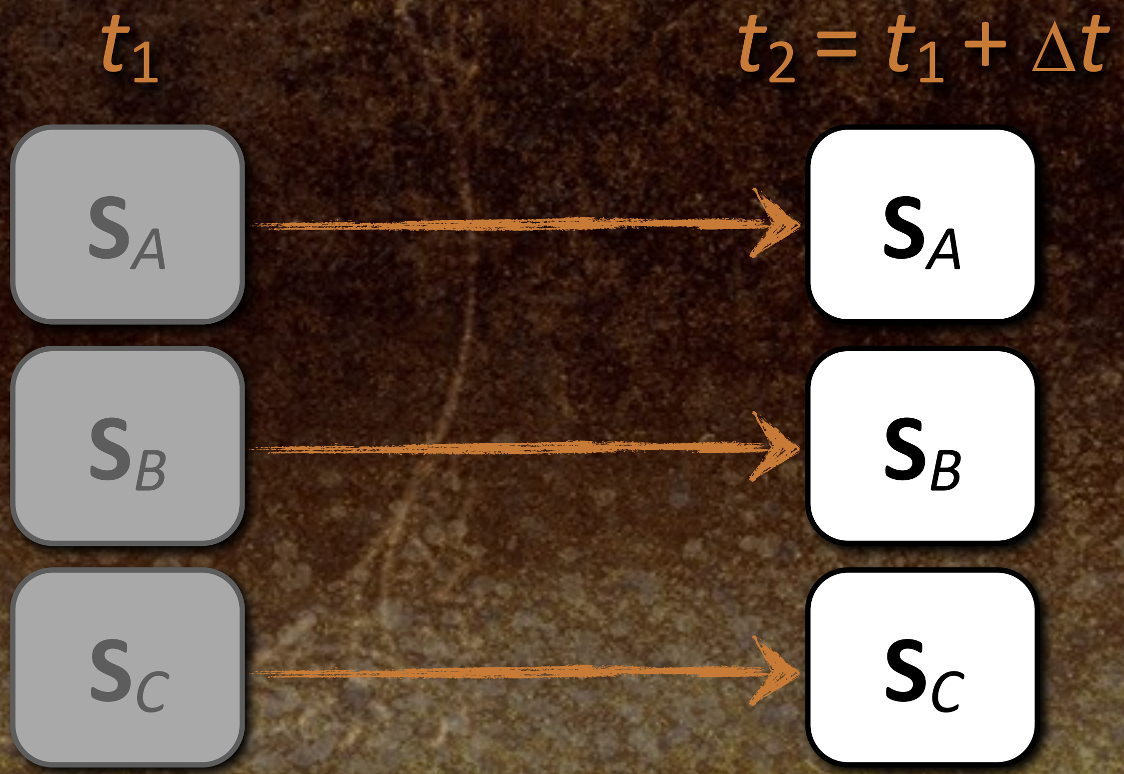
S_A

S_B

S_C

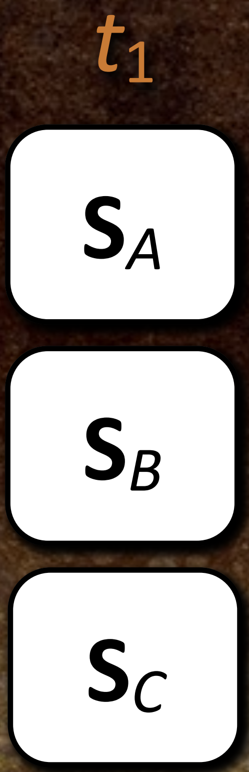
Bucketing and Game Object State Vectors

- Theoretically, the state vectors of all game objects are updated **instantaneously** and **in parallel**.



Bucketing and Game Object State Vectors

- In practice, updates **take time**—object states can be **inconsistent** during the update.



t_1

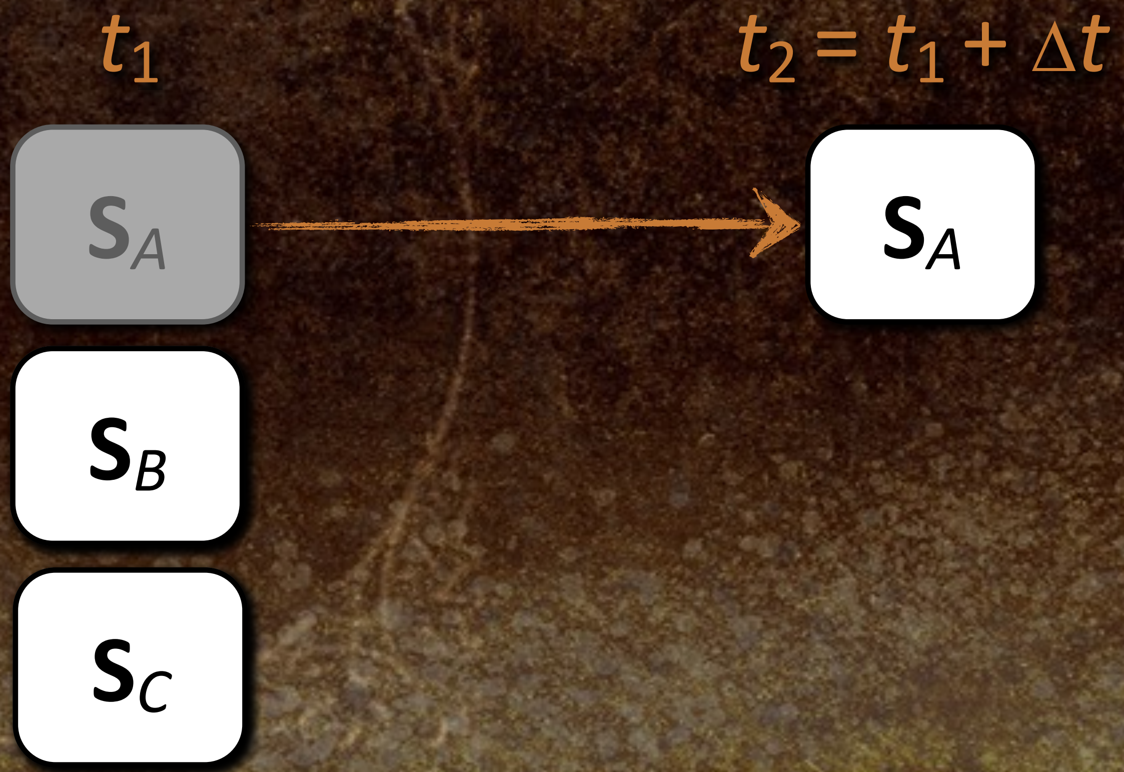
S_A

S_B

S_C

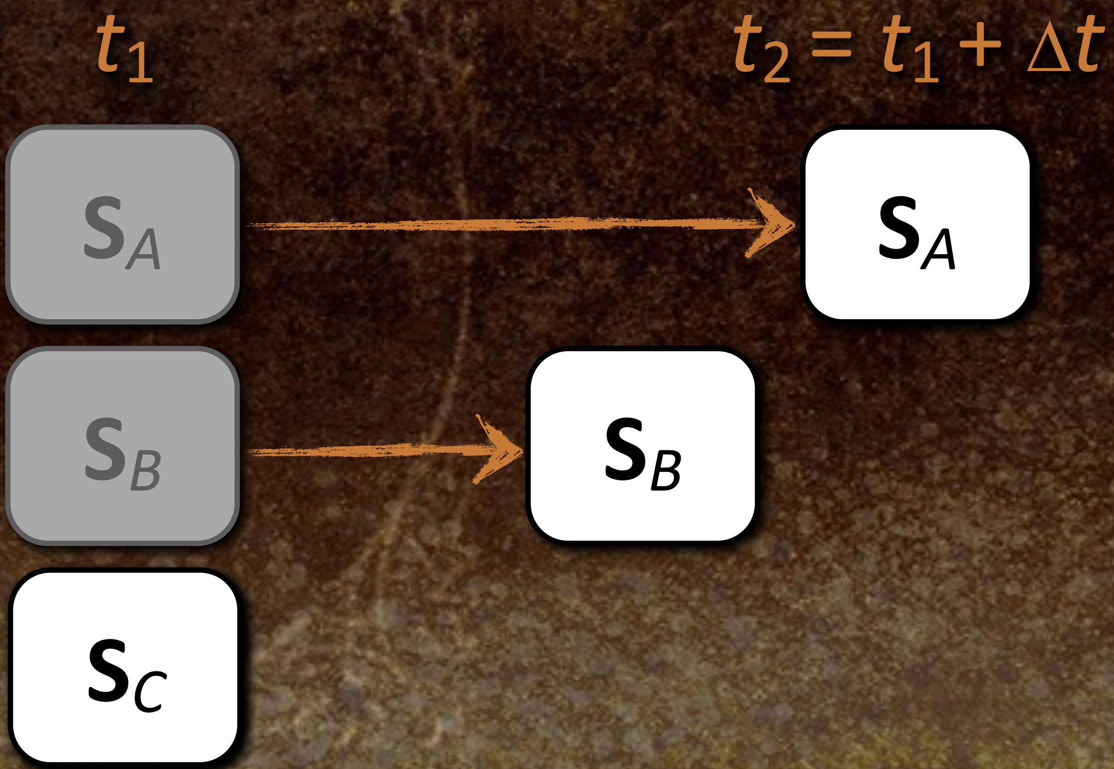
Bucketing and Game Object State Vectors

- In practice, updates **take time**—object states can be **inconsistent** during the update.



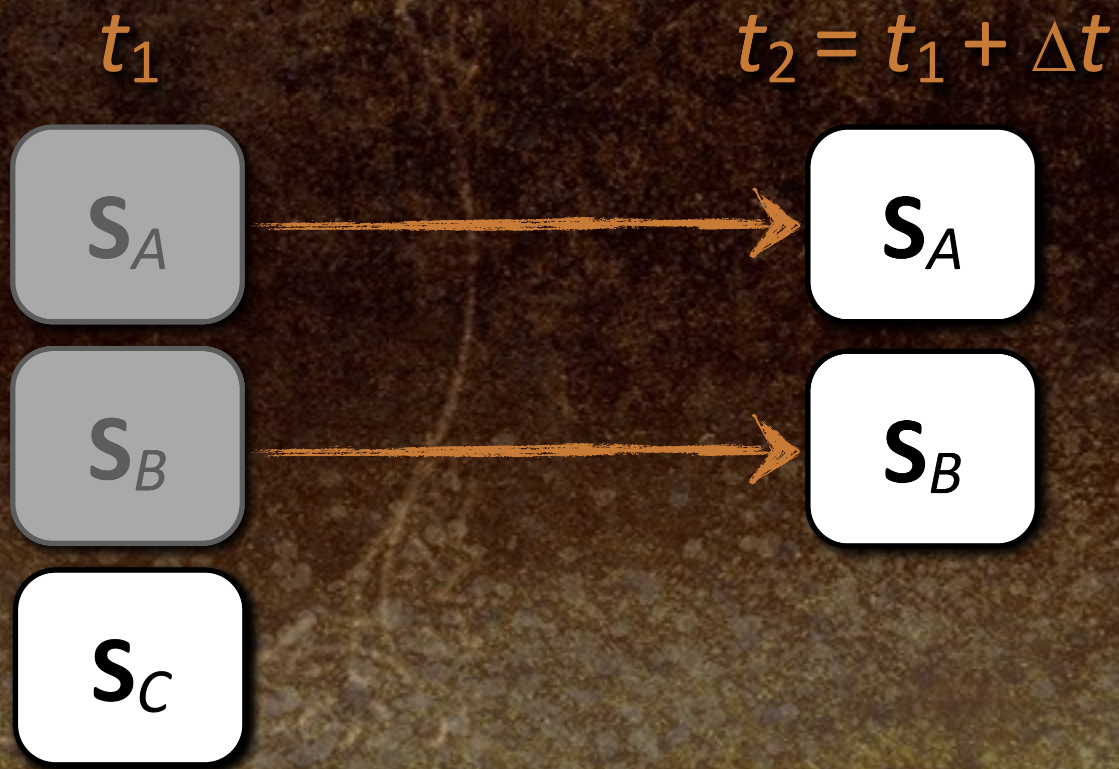
Bucketing and Game Object State Vectors

- In practice, updates **take time**—object states can be **inconsistent** during the update.



Bucketing and Game Object State Vectors

- In practice, updates **take time**—object states can be **inconsistent** during the update.



t_1

$t_2 = t_1 + \Delta t$

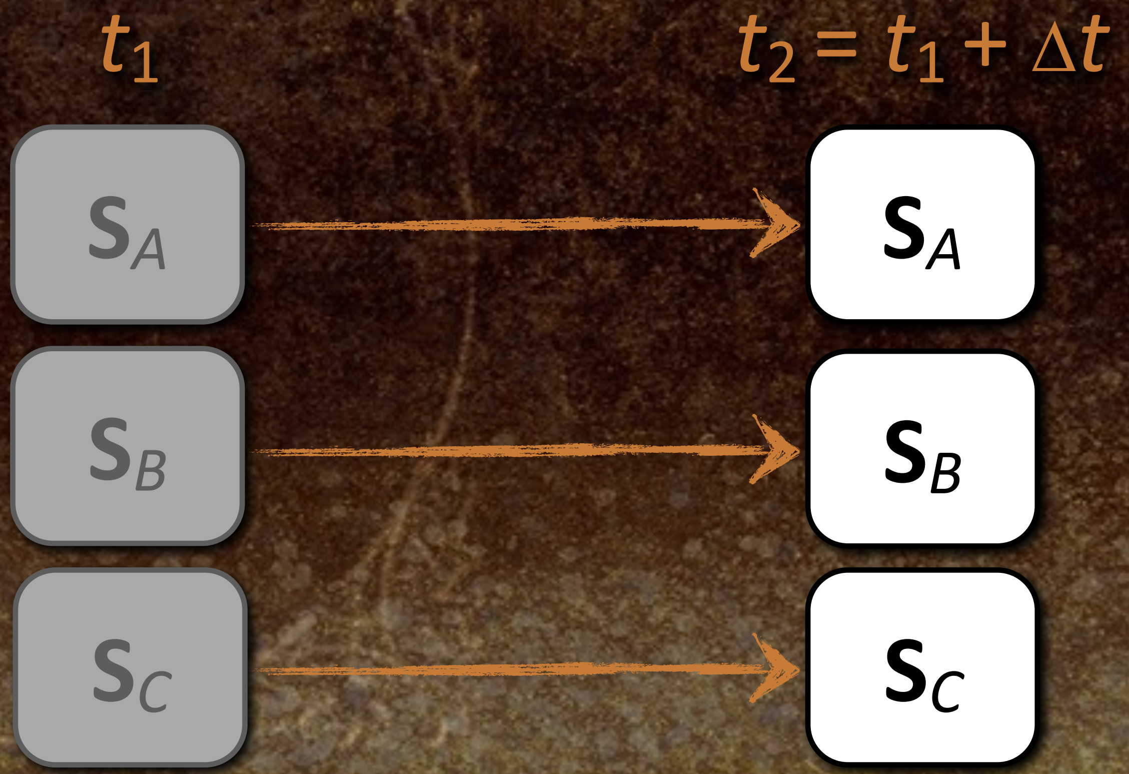
$\mathbf{S}_A$ → $\mathbf{S}_A$

$\mathbf{S}_B$ → $\mathbf{S}_B$

$\mathbf{S}_C$

Bucketing and Game Object State Vectors

- In practice, updates **take time**—object states can be **inconsistent** during the update.



t_1

$t_2 = t_1 + \Delta t$

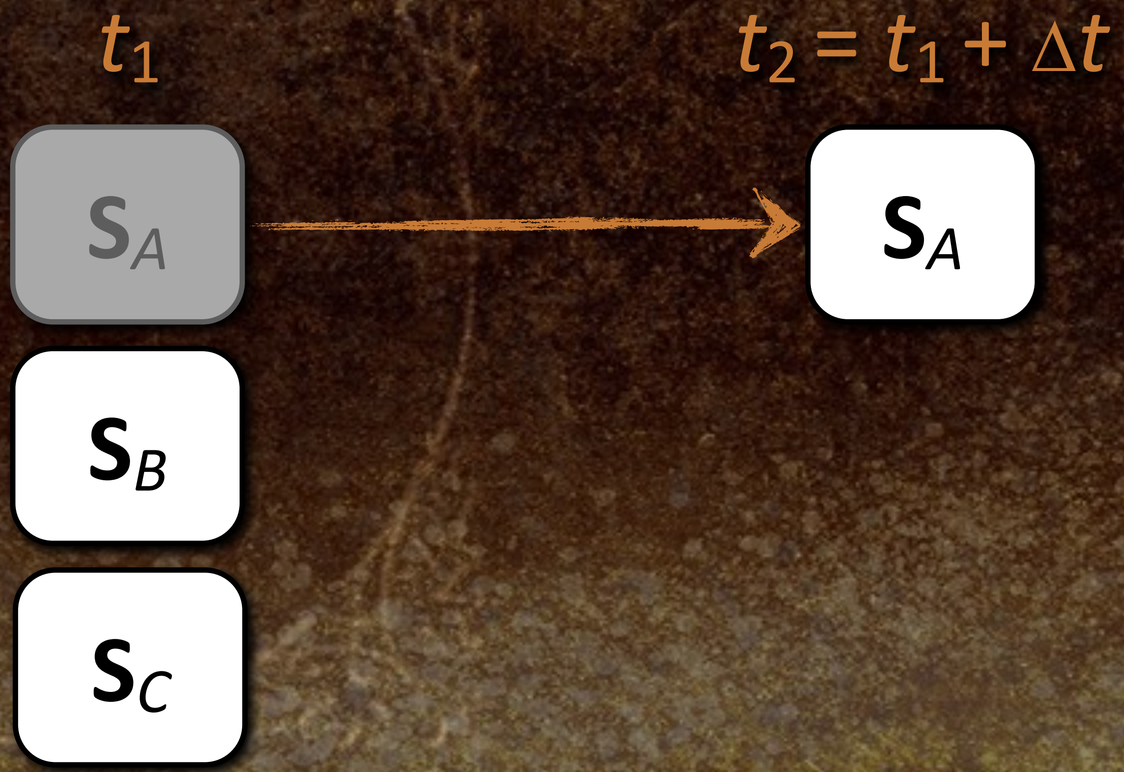
S_A → S_A

S_B → S_B

S_C → S_C

Bucketing and Game Object State Vectors

- Problems arise when we **query the state** of object A during the update of object B or C.



t_1

$t_2 = t_1 + \Delta t$

$\mathbf{S}_A$

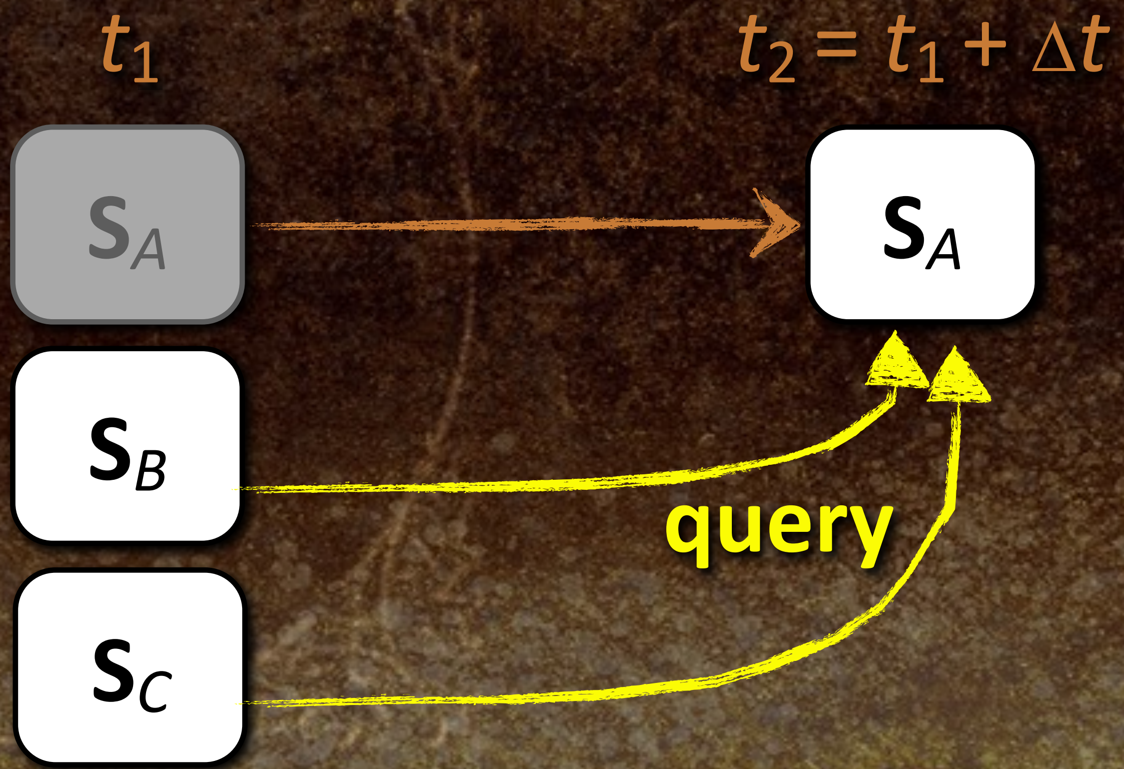
$\mathbf{S}_B$

$\mathbf{S}_C$

$\mathbf{S}_A$

Bucketing and Game Object State Vectors

- Problems arise when we **query the state** of object A during the update of object B or C.



Bucketing and Game Object State Vectors

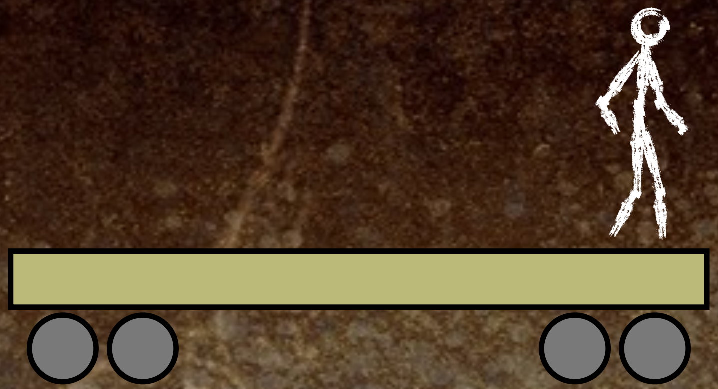
- Major contributor to the ubiquitous "**one frame off** bug."
- Update ordering via bucketing helps, but only when the following rule is adhered to:

An object in bucket *b* may only read the state of objects in buckets (*b* - 1), (*b* - 2), ...

- You can't reliably read the state of objects in your **own bucket**!

One Frame Off Bugs in Collision Queries

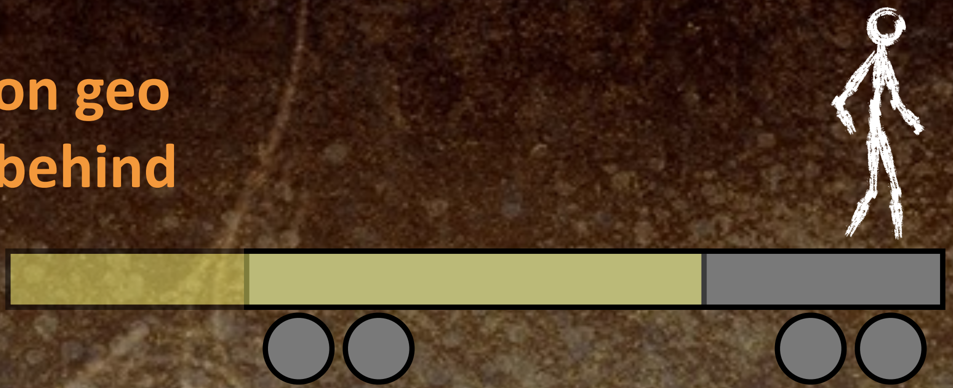
- Another complication arises because:
  - **Collision and physics** run **after** the game objects have **updated**.
  - So, the location of all collision geometry is **one frame old** when we cast our rays/spheres.



One Frame Off Bugs in Collision Queries

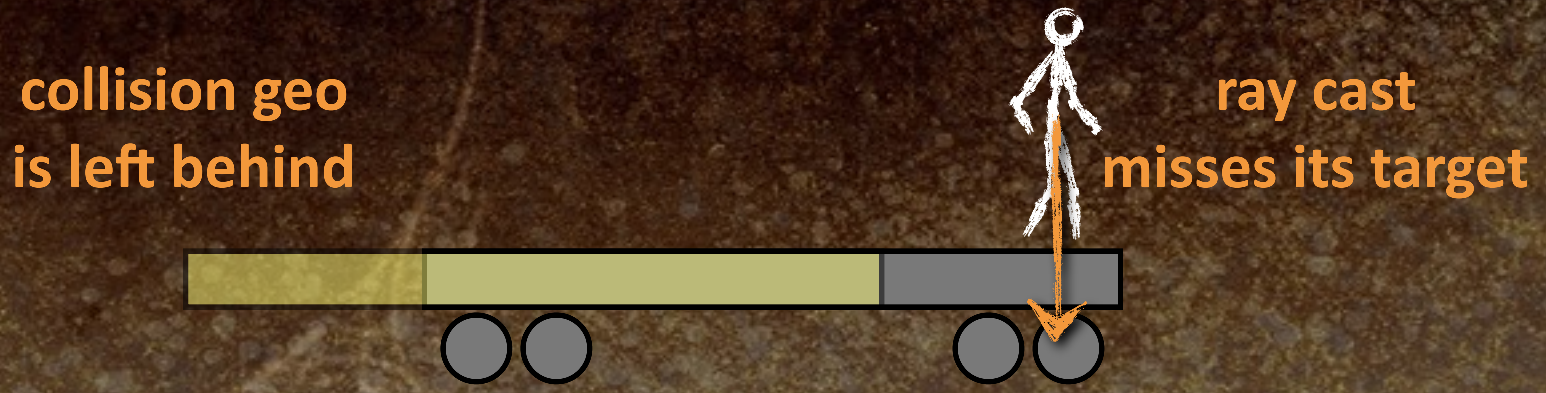
- Another complication arises because:
  - **Collision and physics** run **after** the game objects have **updated**.
  - So, the location of all collision geometry is **one frame old** when we cast our rays/spheres.

collision geo
is left behind



One Frame Off Bugs in Collision Queries

- Another complication arises because:
  - **Collision and physics** run **after** the game objects have **updated**.
  - So, the location of all collision geometry is **one frame old** when we cast our rays/spheres.



One Frame Off Bugs in Collision Queries

- Problem:
  - What we really want is to update each game object's collision geometry **with its bucket**.
- **Impossible**, because coll/phys update is **monolithic**—happens after all game objects have been updated.

One Frame Off Bugs in Collision Queries

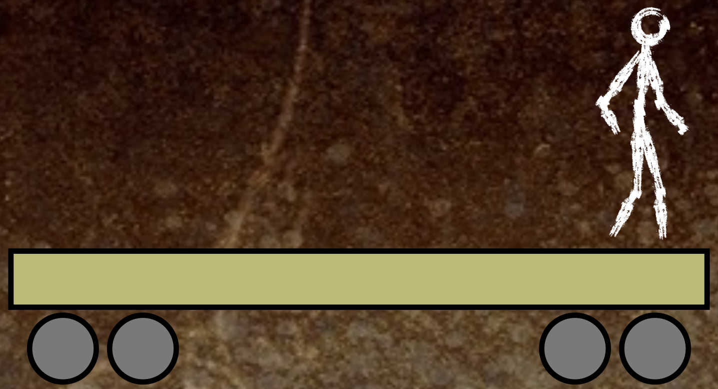
- Problem:
  - What we really want is to update each game object's collision geometry **with its bucket**.
- **Impossible**, because coll/phys update is **monolithic**—happens after all game objects have been updated.
  - **… or is it?**

One Frame Off Bugs in Collision Queries

- Observation:
  - Ray and sphere casts **don't care** about the full collision/physics update.
    - Don't need contact information, velocity, etc.
    - Only need to know where the collision geo **will be**.
- Solution:
  - All we need to do is **update the broadphase AABBs** between buckets.

Solving One Frame Off Bugs in Collision Queries

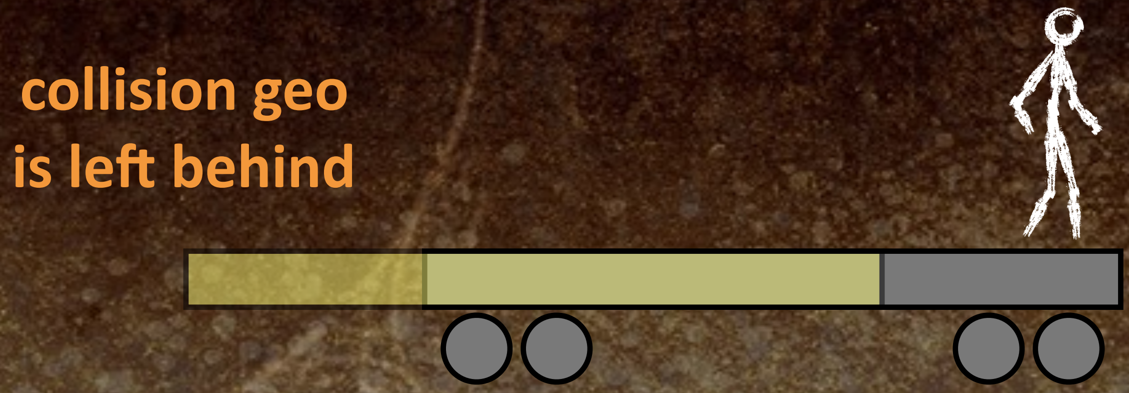
- Instead, we'll update the AABBs early, in between game object bucket updates.
  - This allows our collision queries to return **correct results**.



Game Forum Germany 2010

Solving One Frame Off Bugs in Collision Queries

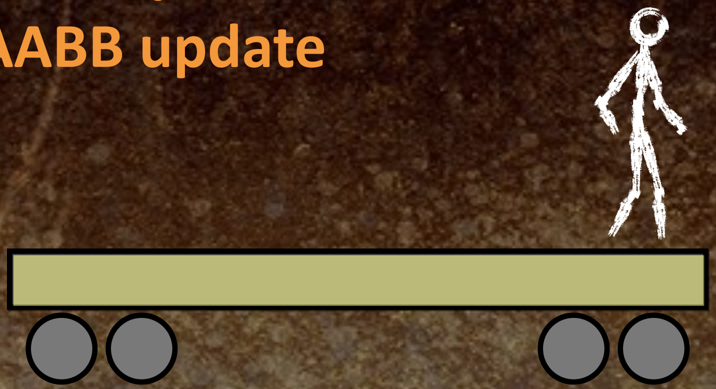
- Instead, we'll update the AABBs early, in between game object bucket updates.
  - This allows our collision queries to return **correct results**.



Solving One Frame Off Bugs in Collision Queries

- Instead, we'll update the AABBs early, in between game object bucket updates.
  - This allows our collision queries to return **correct results**.

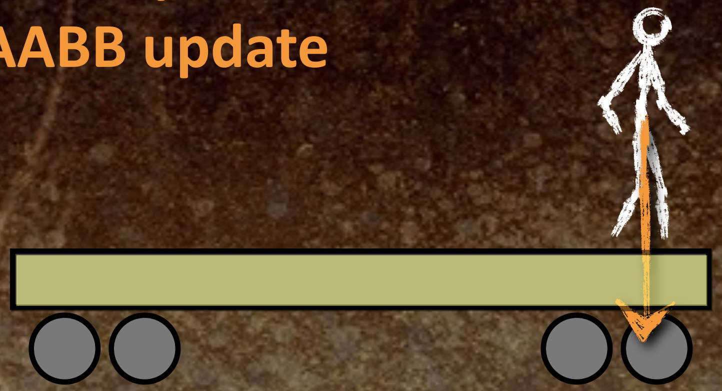
broadphase
AABB update



Solving One Frame Off Bugs in Collision Queries

- Instead, we'll update the AABBs early, in between game object bucket updates.
  - This allows our collision queries to return **correct results**.

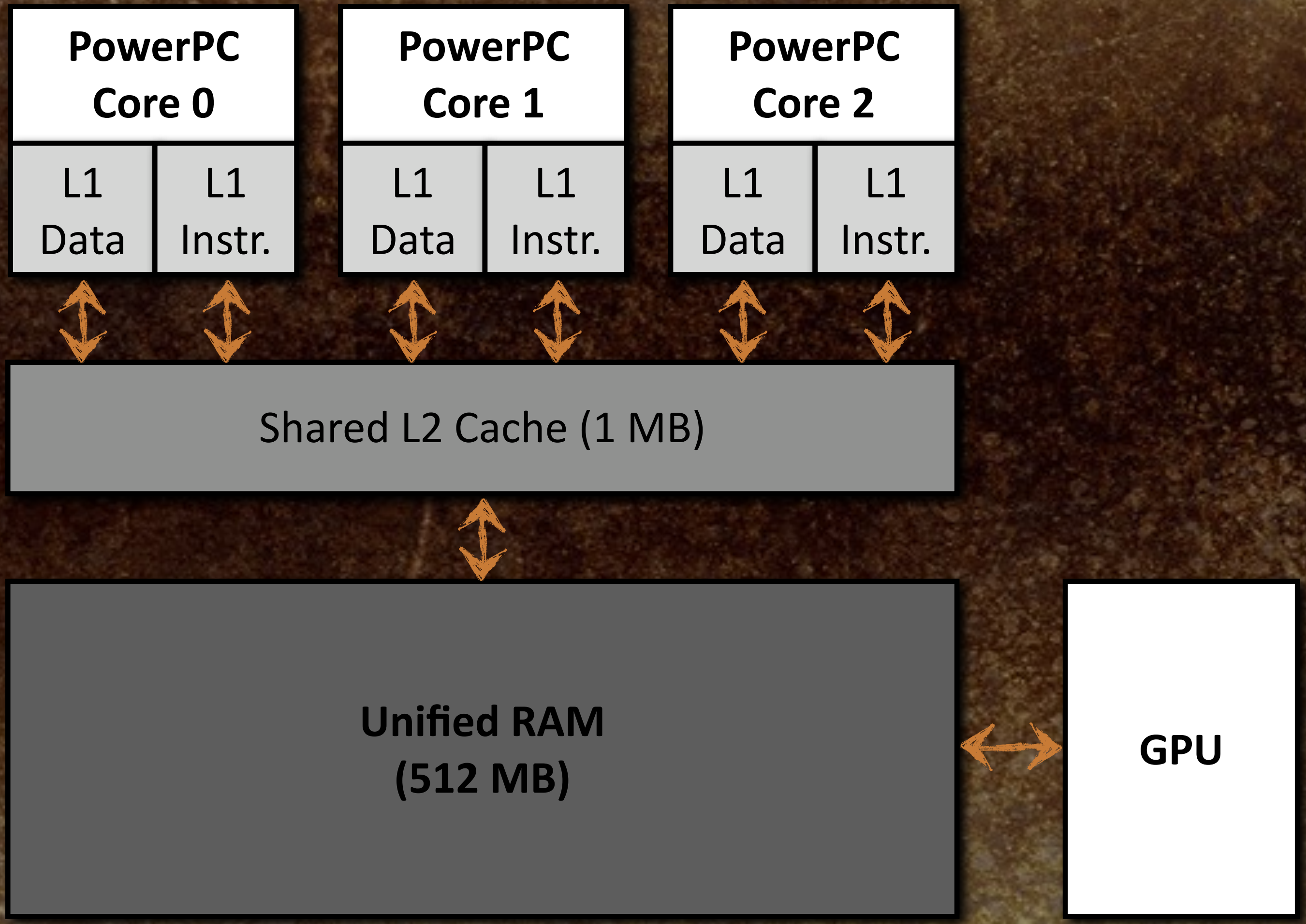
broadphase
AABB update



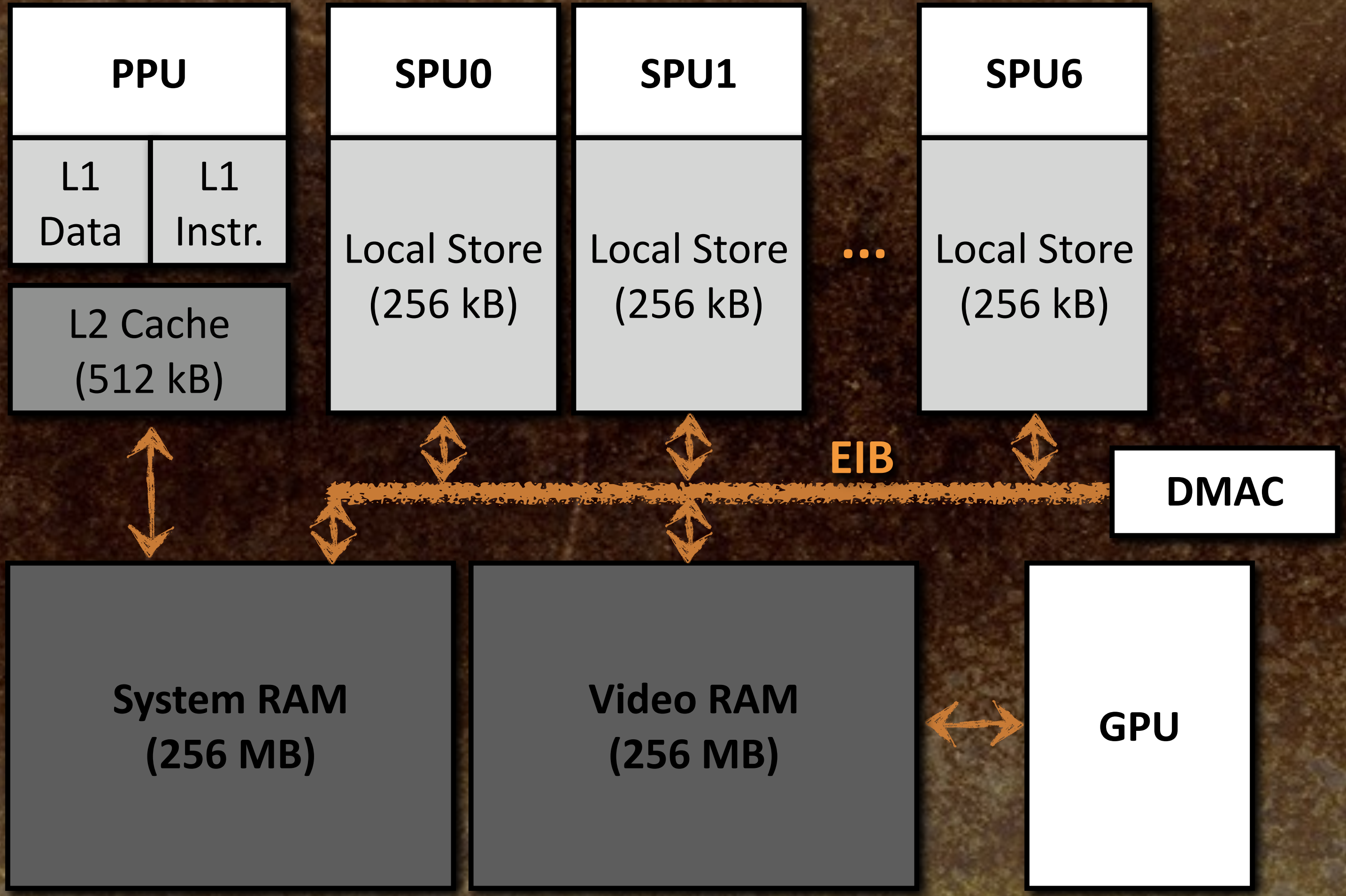
ray cast
hits

Achieving Parallelism

Game Console Architecture: Xbox 360



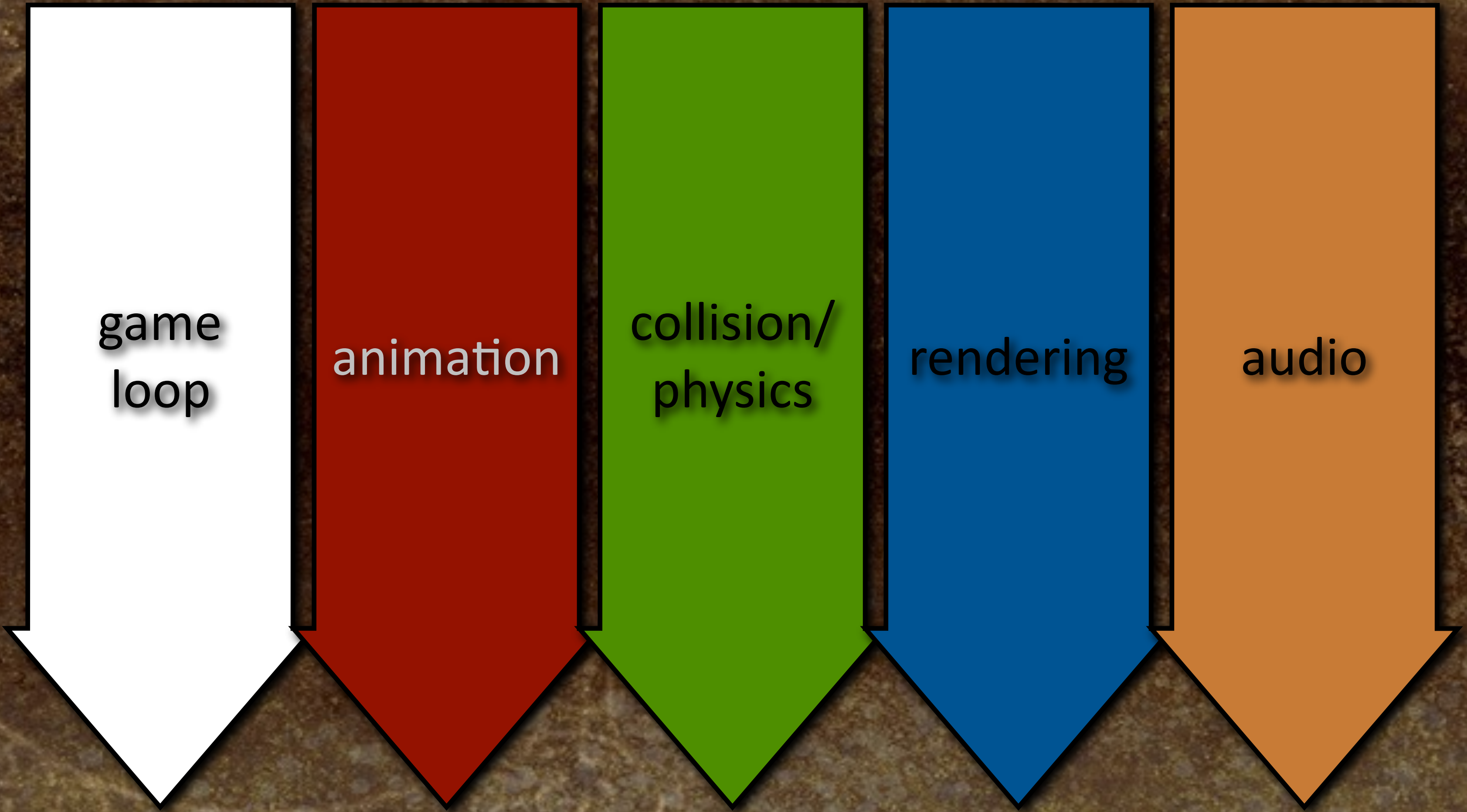
Game Console Architecture: PLAYSTATION 3



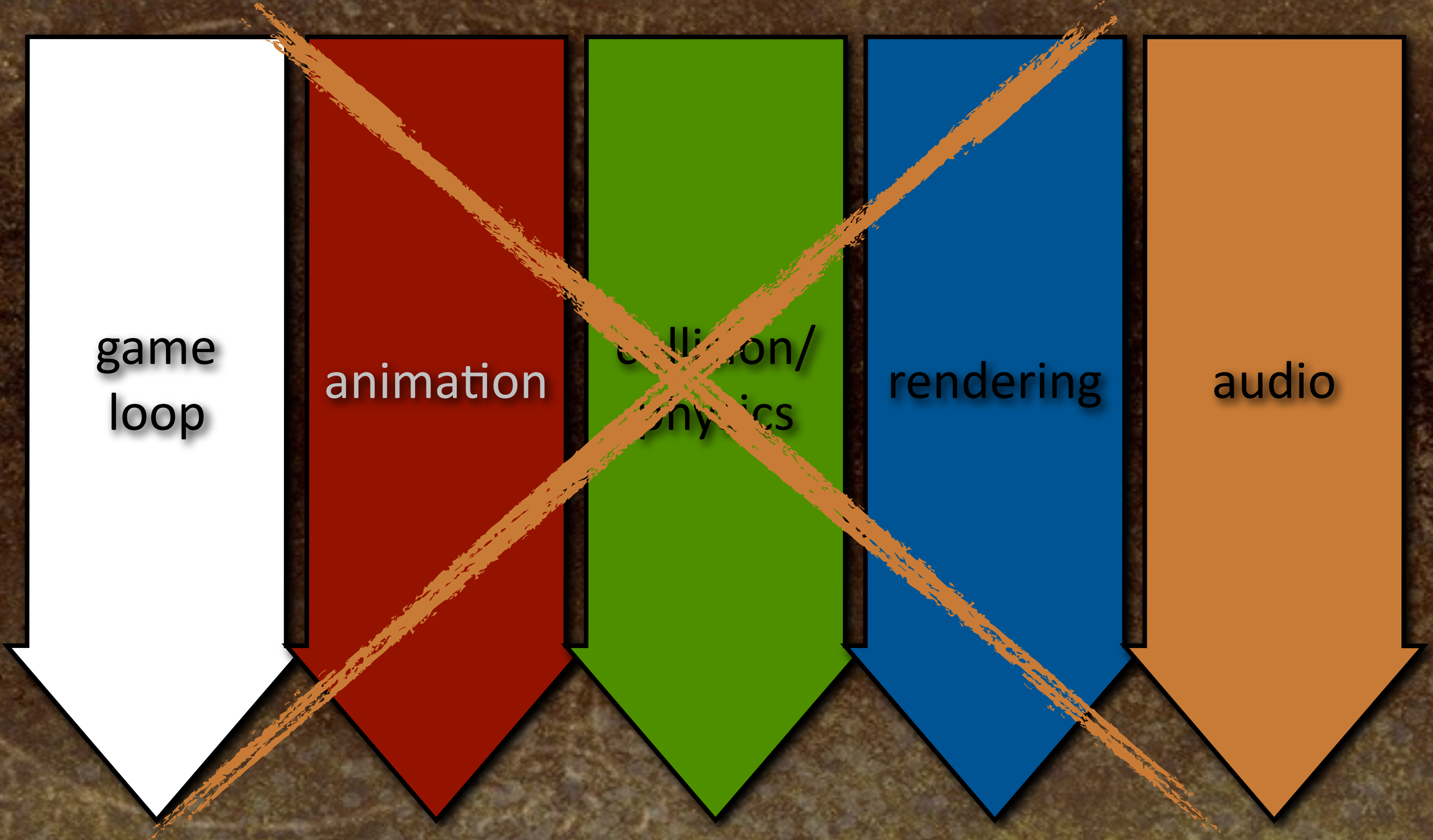
Thinking About Modern Multiprocessor Hardware

- We'll think in terms of multiple **hardware threads**.
- These could be provided by:
  - **hyperthreaded CPU**,
  - **multiple cores**,
  - **SPUs** on PS3/Cell.

Ways to Achieve Parallelism: Thread per Subsystem



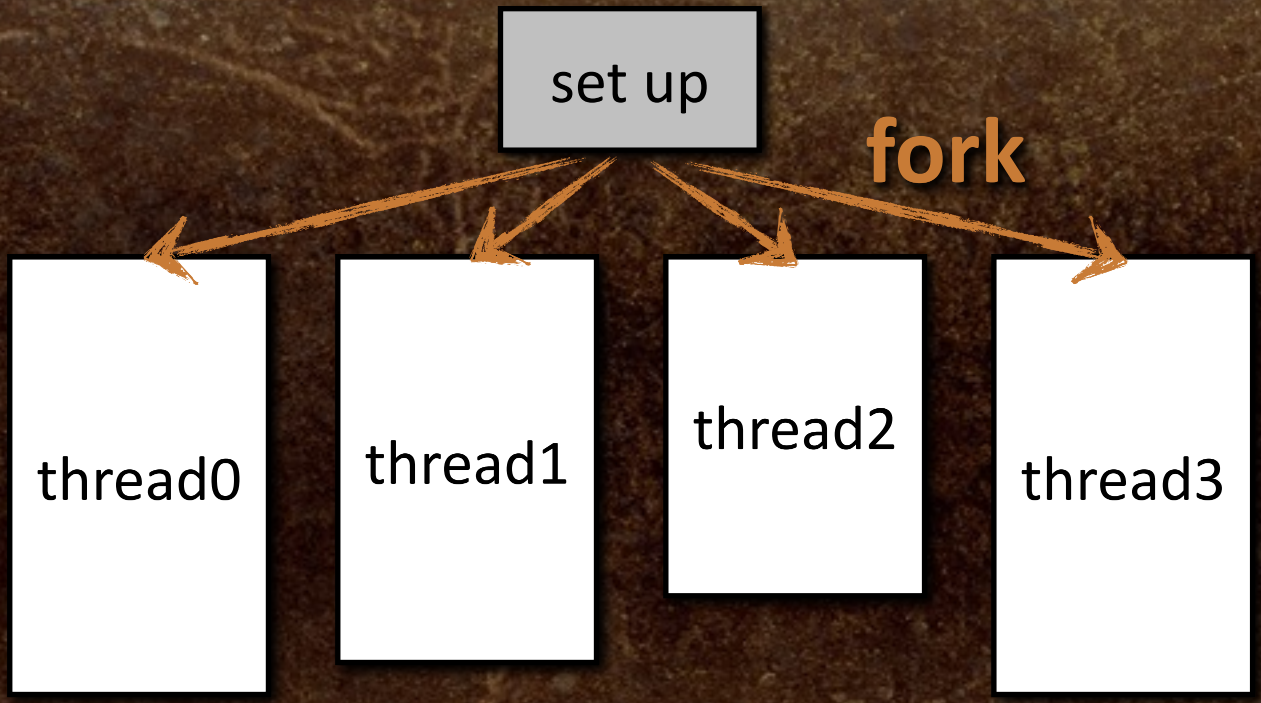
Ways to Achieve Parallelism: Thread per Subsystem



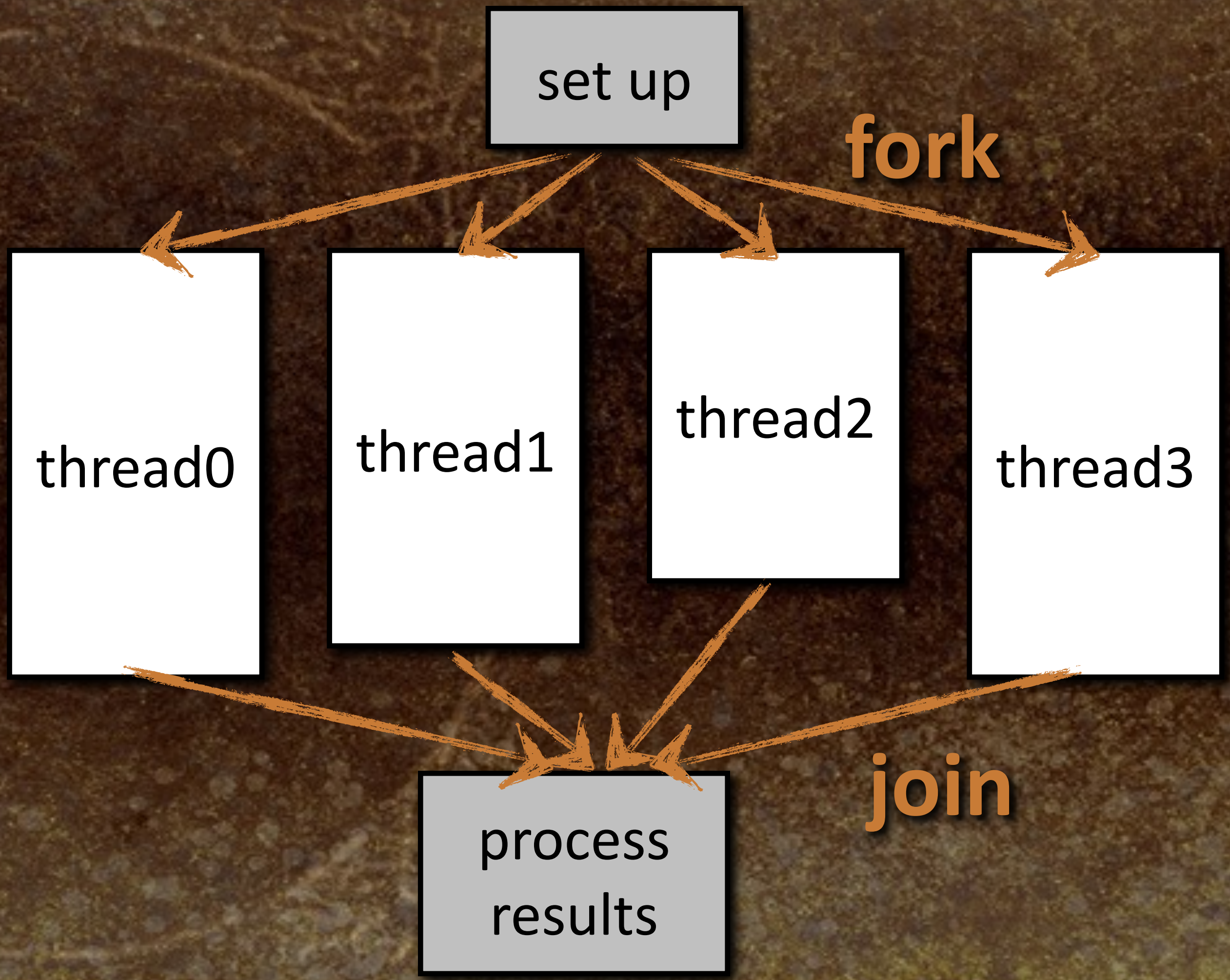
Ways to Achieve Parallelism: Fork/Join

set up

Ways to Achieve Parallelism: Fork/Join



Ways to Achieve Parallelism: Fork/Join



set up

fork

thread0

thread1

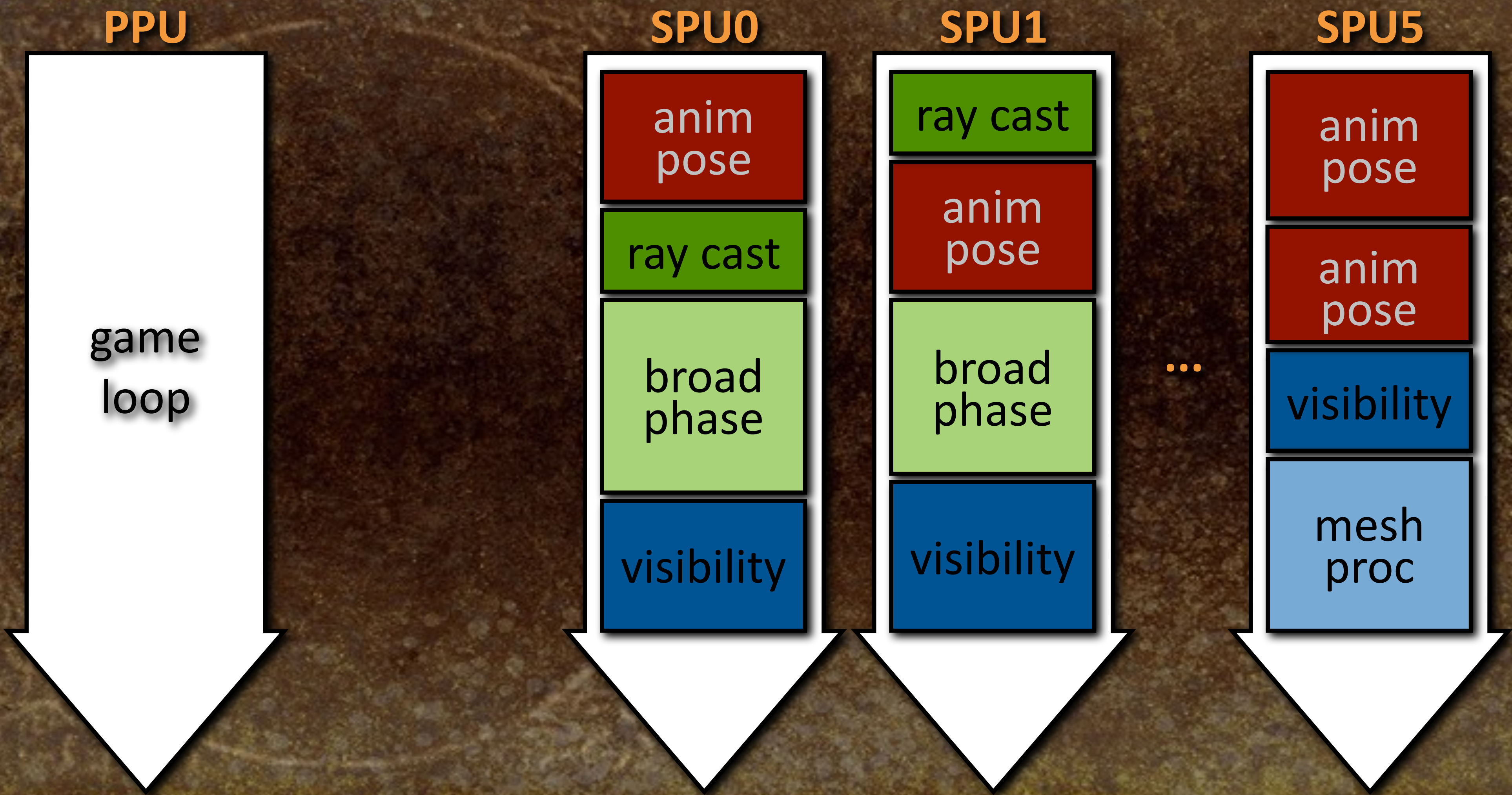
thread2

thread3

join

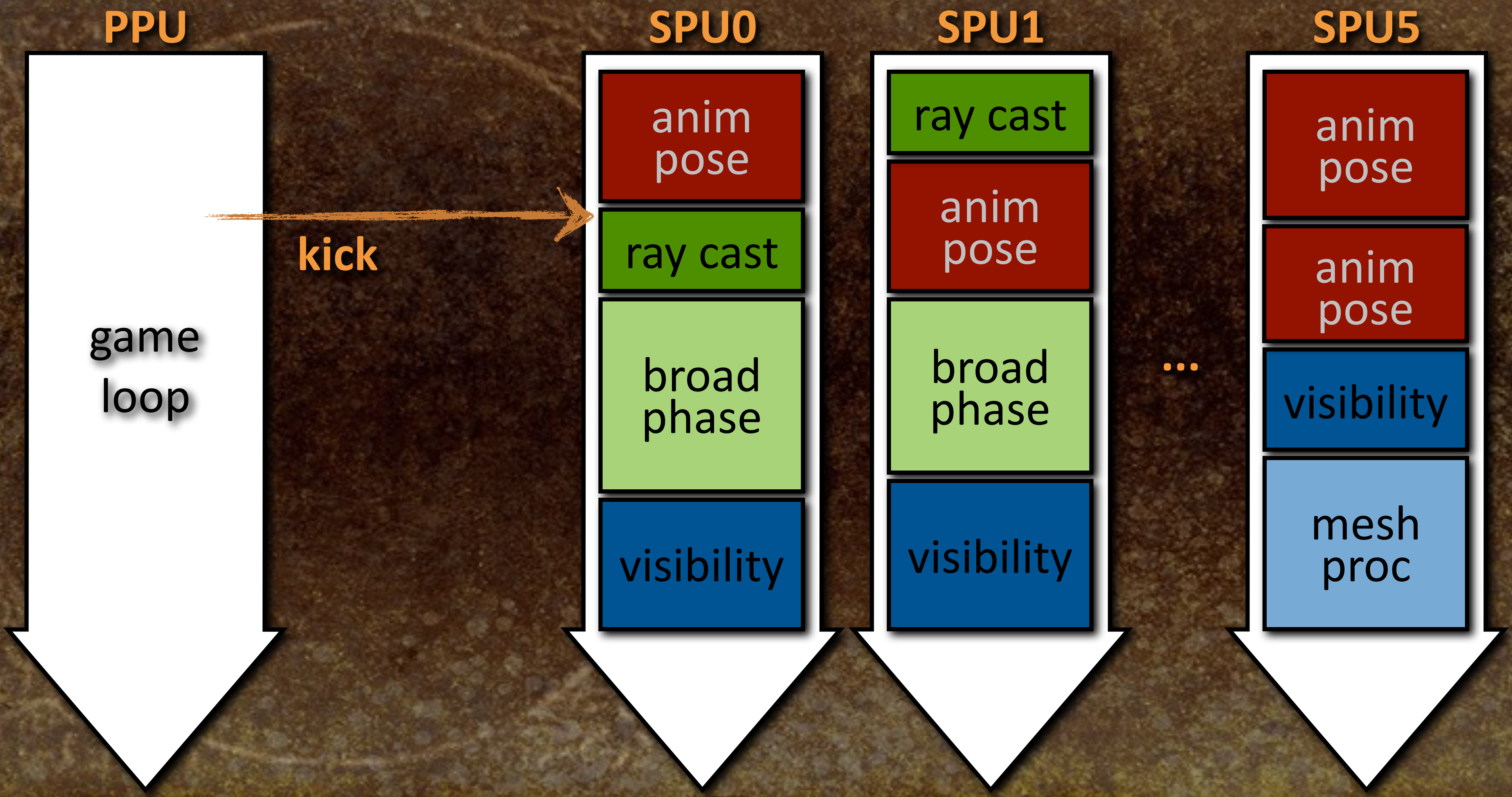
process results

Ways to Achieve Parallelism: Jobs

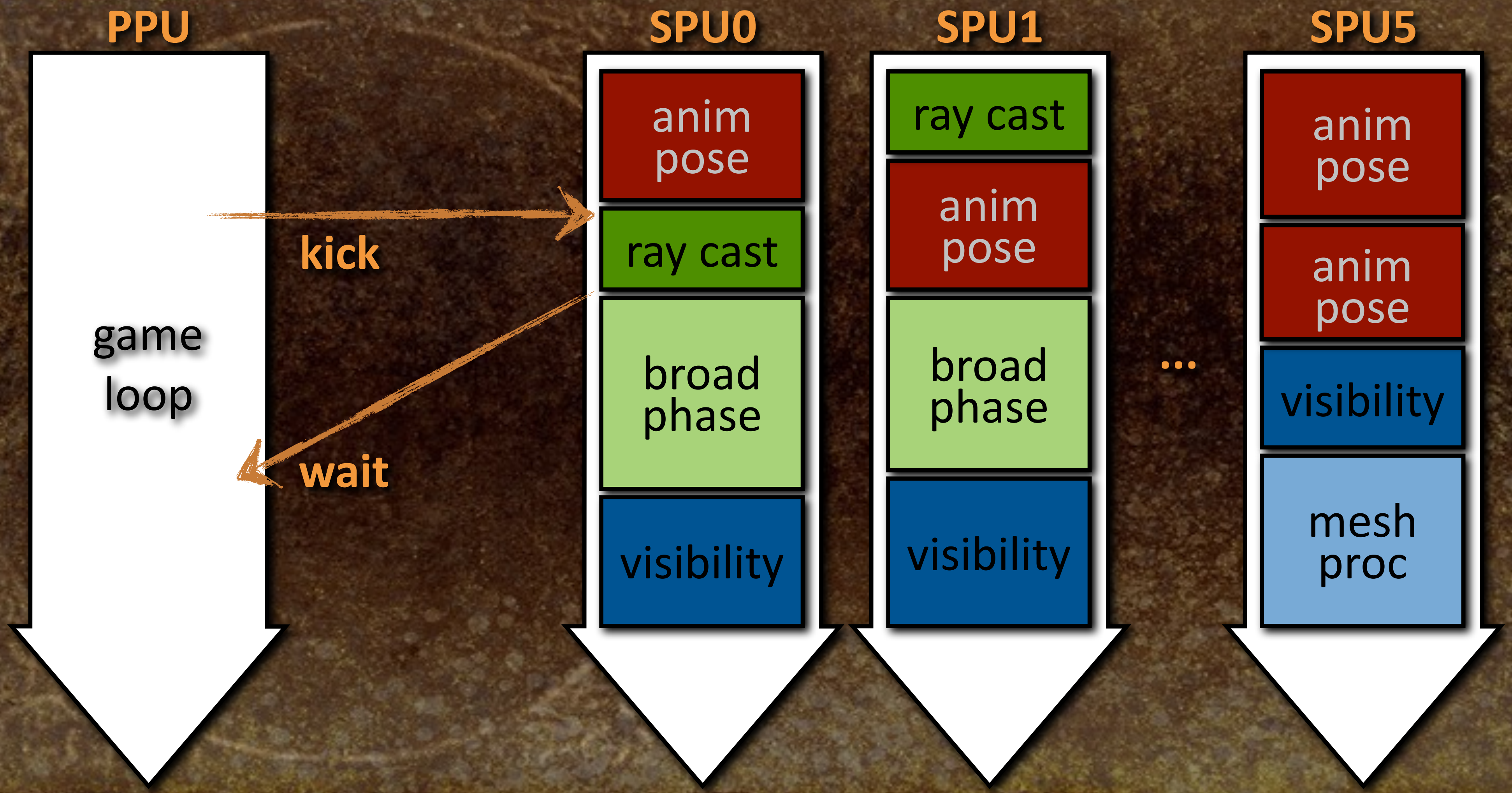


NAUGHTY DOG

Ways to Achieve Parallelism: Jobs



Ways to Achieve Parallelism: Jobs



Ways to Achieve Parallelism: Jobs

- **Job** = [(**code** + **input data**) → **output data**]
  - **Kick** job = request job to be scheduled on a HW thread.
  - Must **wait** for job before processing its output data.
    - If job is **done**, wait takes close to **zero time**.
    - If job is **not done**, main thread (PPU) **blocks** until it is done.

- **Job manager** handles **scheduling**, **allocates buffers** in local store, and **coordinates DMAs**.
  - Programmer specifies data sources & destinations via a **DMA list**.

Parallelism on the Train

```
while (!quit)
{
   // ...

   RayCastHandle hRayCast
    = kickRayCast(...);

   // do other useful work on PPU
   // ...

   waitRayCast(hRayCast);
   processRayCastResults(hRayCast);

   // ...
}
```

Parallelism on the Train

```
while (!quit)
{
   // ...

   RayCastHandle hRayCast
    = kickRayCast(...);

   // do other useful work on PPU
   // ...

   waitRayCast(hRayCast);
   processRayCastResults(hRayCast);

   // ...
}
```

```
RayCastHandle hRayCast = INVALID;

while (!quit)
{
   // ...

   // wait and re-kick immediately
   if (hRayCast.IsValid())
   {
      waitRayCast(hRayCast);
      processRayCastResults(hRayCast);
   }
   hRayCast = kickRayCast(...);

   // ...
}
```

Parallelism on the Train

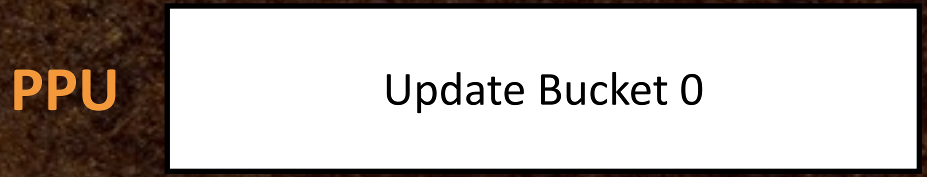
- How does parallelism with jobs affect our train design?
  - **Large-scale engine system updates** and **ray/sphere casts** are largely **asynchronous** in *U2:AT*.
    - Main game loop (PPU) **kicks jobs** on SPU.
    - **Other work can proceed** on the PPU while jobs are running.
    - **Results** picked up **later** this frame… or **next frame**.

Parallelism on the Train

- Data must be **compact** and **contiguous** so it can be DMA'd to the SPUs.
  - We're doing this already, to maintain good **cache coherency**.
- The collision world must be **locked** in order to update broadphase AABBs:
  - Wait until all **outstanding ray/sphere jobs** are **done**.
  - **Lock**.
  - **Update AABBs**.
  - **Unlock**.

Parallelism on the Train

- We can even do our broadphase AABB updates asynchronously.

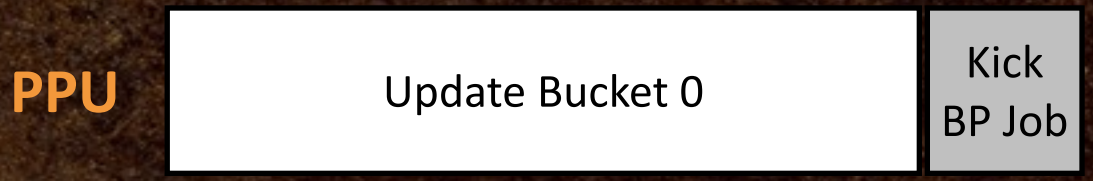


PPU

Update Bucket 0

Parallelism on the Train

- We can even do our broadphase AABB updates asynchronously.



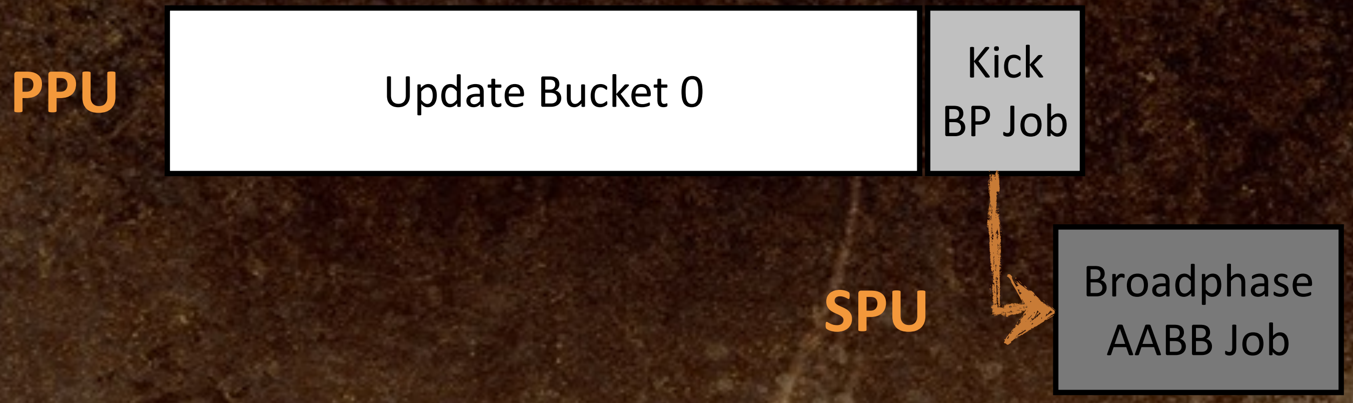
PPU

Update Bucket 0

Kick BP Job

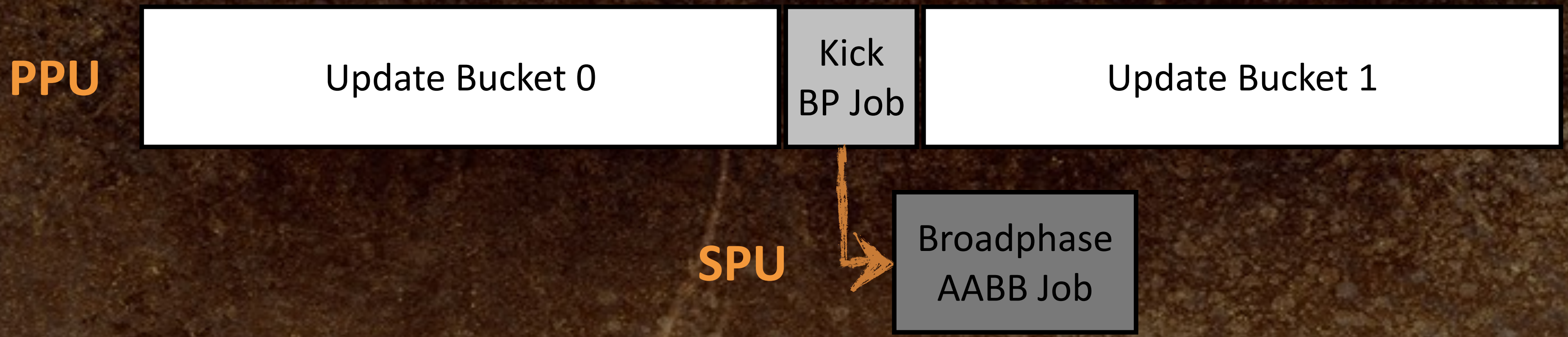
Parallelism on the Train

- We can even do our broadphase AABB updates asynchronously.



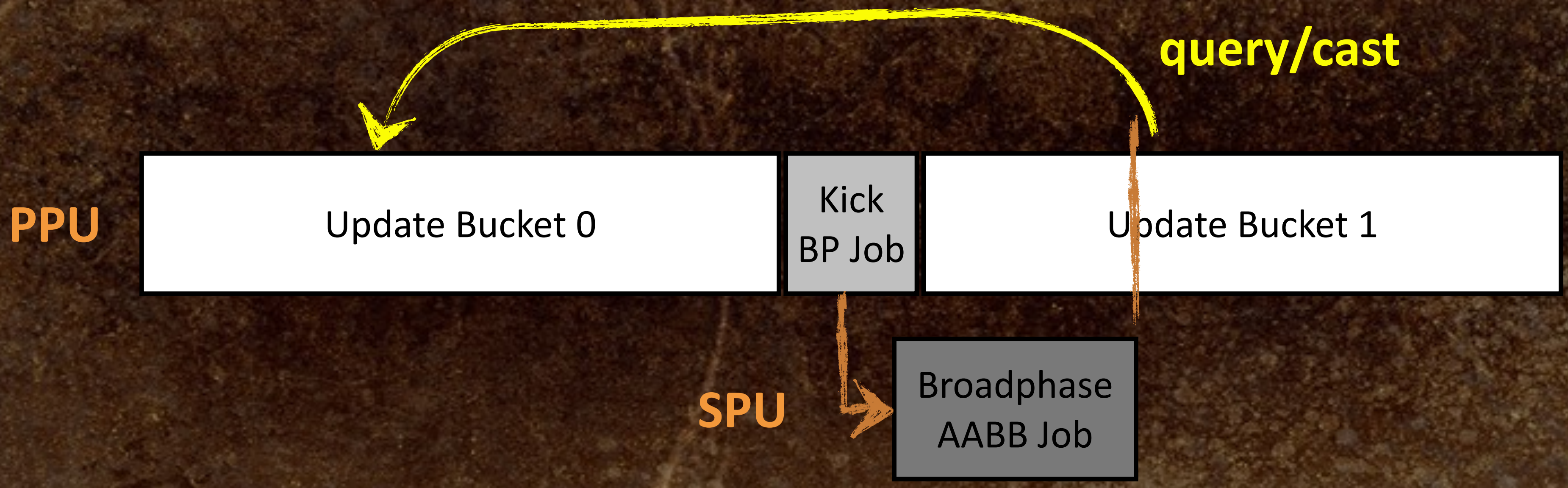
Parallelism on the Train

- We can even do our broadphase AABB updates asynchronously.



Parallelism on the Train

- We can even do our broadphase AABB updates asynchronously.



Conclusions

Conclusions

- The combination of **modern hardware restrictions** and the problems generated by the **train level**...
  - forced us to design our engine in a **robust** and **efficient** manner.
- Results:
  - *U2:AT* boasts near **100% hardware utilization** on PS3 (PPU + all 6 SPUs).
  - Way-cool **train level** as pay-off for all the hard work.
  - Technology was the primary enabler for the **convoy level** as well.

Thanks For Listening!

- Free free to send questions to me at:

jason_gregory@naughtydog.com

NAUGHTY DOG