



Insomniac Physics

Eric Christensen
GDC 2009

Overview

- Go over the evolution of IG physics system
- Shaders
- Library Shaders
- Custom event shaders

Original Design

Resistance: Fall of Man

- Ported From PC to PS3
- PPU Heavy
- SPU Processes Blocked
- Two Jobs (Collision, Simulation)
- Simulation Jobs too memory heavy dispatched to PPU version.
- Expensive

Original Design

Resistance: Fall of Man

Physics Update



Gather Potentially
Colliding Objects

Original Design

Resistance: Fall of Man

Physics Update



Cache Collision
Geometry

Original Design

Resistance: Fall of Man

Physics Update



Run
SPU Collision Jobs

Original Design

Resistance: Fall of Man

Physics Update



Sync

Original Design

Resistance: Fall of Man

Physics Update

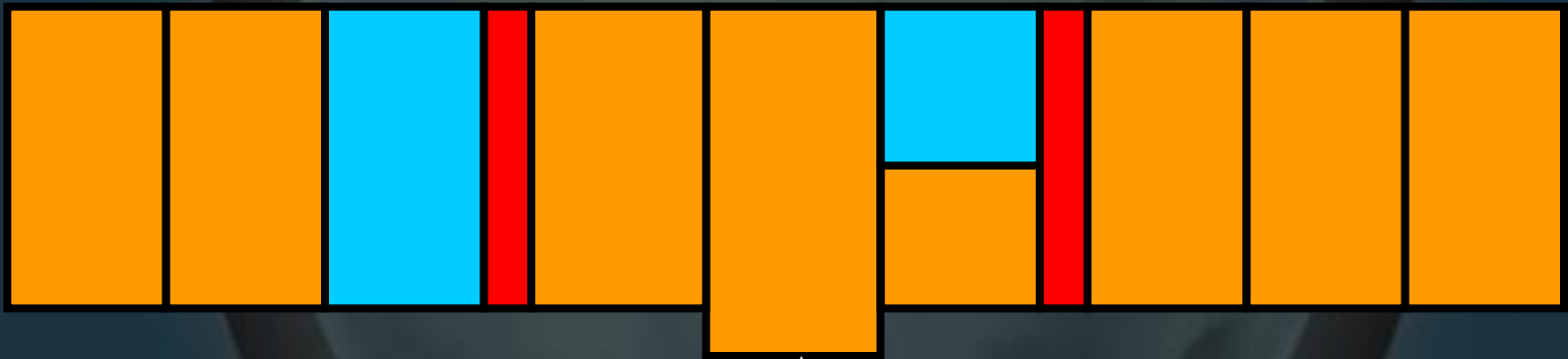


Process Contact
Constraints

Original Design

Resistance: Fall of Man

Physics Update



Create Simulation
Pools

Original Design

Resistance: Fall of Man

Physics Update

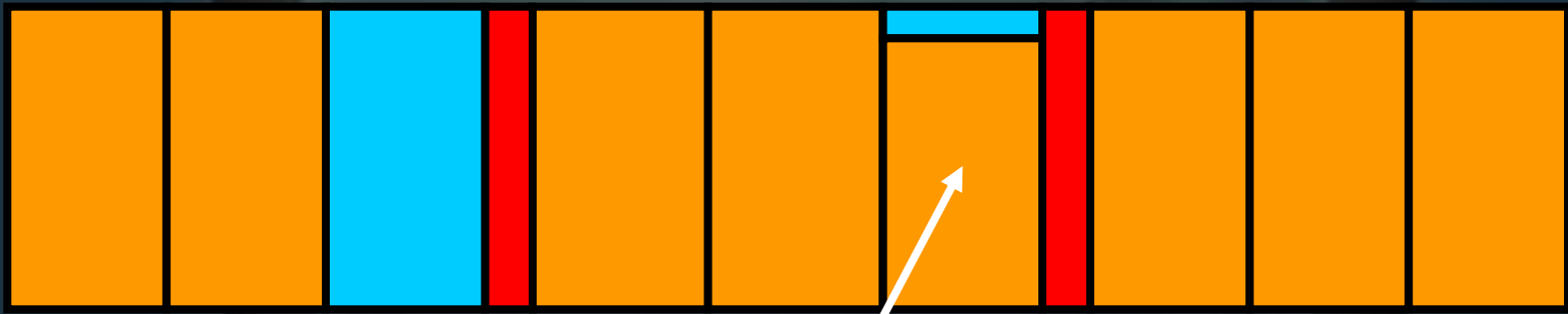


Run
SPU Simulation Jobs

Original Design

Resistance: Fall of Man

Physics Update



Run Simulations
Too Big For SPU!

Original Design

Resistance: Fall of Man

Physics Update



Sync

Original Design

Resistance: Fall of Man

Physics Update



Process Results

Original Design

Resistance: Fall of Man

Physics Update



Call Events

Original Design

Resistance: Fall of Man

Physics Update



Update Joints

Original Design

Resistance: Fall of Man

- Simulation Jobs Ran as Pools were generated.
- PPU Simulation Jobs ran concurrently with the SPU Simulation Jobs
- This was the **ONLY** asynchronous benefit!
- Not much!

Original Design

Resistance: Fall of Man

- Physics had the largest impact on frame rate
- Pipeline design made it difficult to reliably optimize
- There was A LOT to learn

Phase 2

Ratchet & Clank Future

- Collision and Simulation run in a single SPU Job
- Single sync-point
- Large PPU window from start of Job to End of Job
- Use of Physics Shaders

Phase 2

Ratchet & Clank Future

Physics Update



Gather Potentially Colliding Objects

Phase 2

Ratchet & Clank Future

Physics Update



Cache Collision Geometry

Phase 2

Ratchet & Clank Future

Physics Update



Start Physics SPU Jobs

Phase 2

Ratchet & Clank Future

Physics Update

PPU Work

Do Collision



Phase 2

Ratchet & Clank Future

Physics Update



Generate Simulation Pools

Phase 2

Ratchet & Clank Future

Physics Update



Simulate

Phase 2

Ratchet & Clank Future

Physics Update



Update Joints

Phase 2

Ratchet & Clank Future

Physics Update

PPU Work

DMA Results



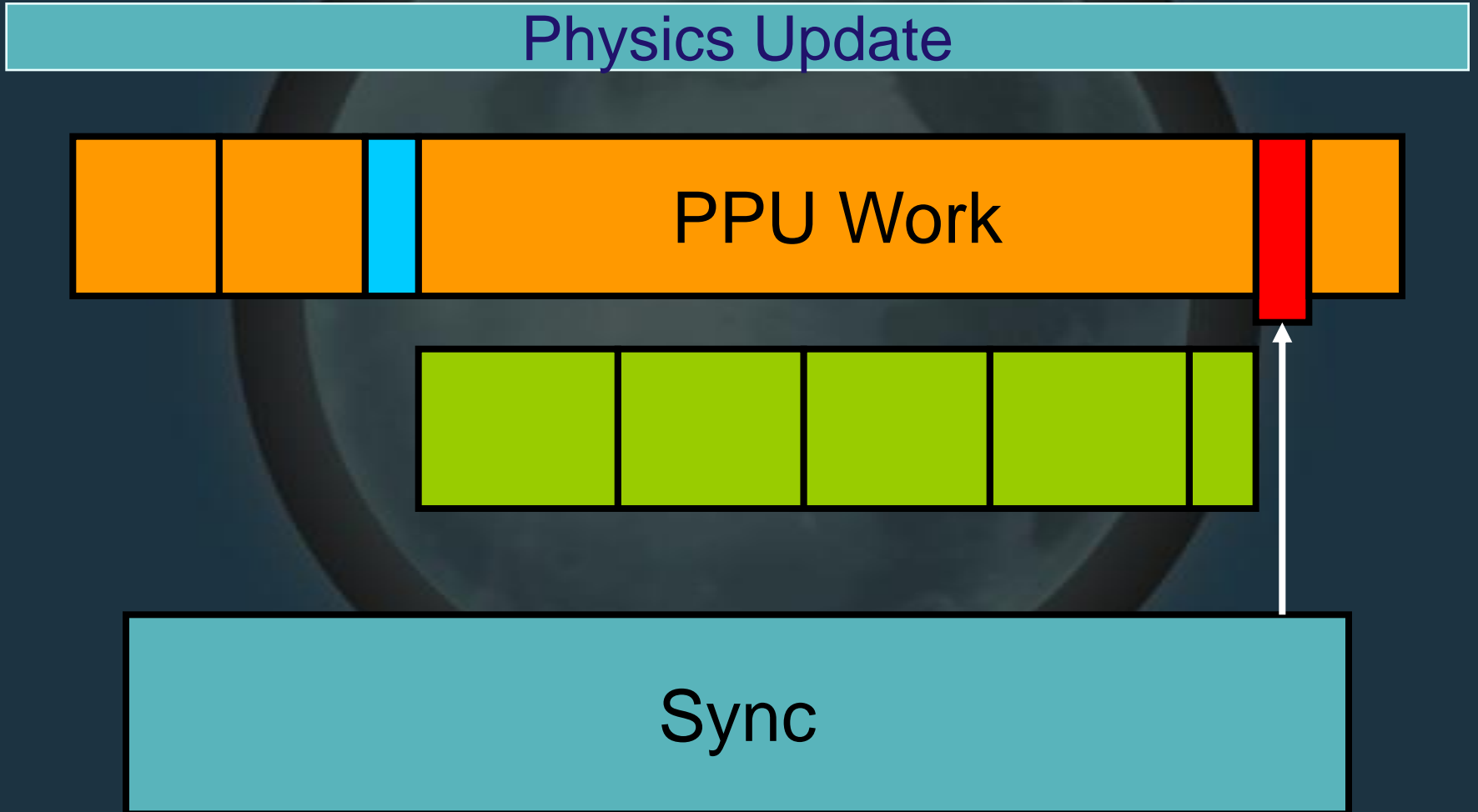
Phase 2

Ratchet & Clank Future

Physics Update

PPU Work

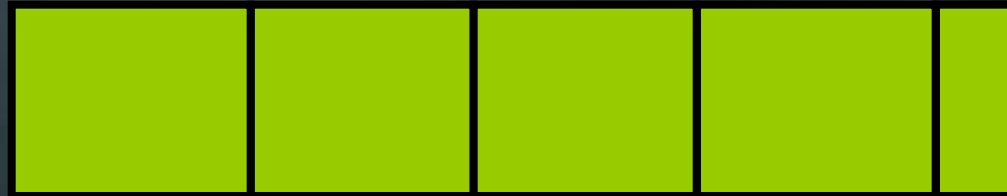
Sync



Phase 2

Ratchet & Clank Future

Physics Update



Update Events

Phase 2

Ratchet & Clank Future

- Shaders helped free up local store
- Each big component had it's own set of shaders
- Constraints
- Solvers
- User customized data transformation

Physics Intersection Shaders

Example Function Prototype

```
unsigned int SphereOBB(const CollPrim &a, const CollPrim &b, CollResult *results)
```

- Shaders are loaded into local store during the collision process and called via a function table using a mask created by geometry ID
- Rollback local store when done
- Savings of up to 70k of local store usage

Physics Jacobian Shaders

Example Function Prototype

```
unsigned int BuildJDBall(Constraint *c, Manifold *m, RigidBody *rblist, float fps,  
                        float error, float dscale, Jacobian *jlist,  
                        CommonTrig *trig_funcs, CommonFunc *common_funcs,  
                        ConstraintFunc *constraint_util);
```

- An example of a shader being called from another shader
- Constraints are sorted by type, then the corresponding shader is loaded to process a group of like constraints
- Saves us roughly 100k!
- We can add more constraint types without worrying about impact on kernel size

Physics Jacobian Shaders

Example Function Prototype

```
unsigned int BuildJDBall(Constraint *c, Manifold *m, RigidBody *rblist, float fps,  
                        float error, float dscale, Jacobian *jlist,  
                        CommonTrig *trig_funcs, CommonFunc *common_funcs,  
                        ConstraintFunc *constraint_util);
```

- CommonTrig contains pointers to trigonometry functions that live in the main physics kernel
- Sin, Cos, ACos, Atan, etc...
- Any optimizations will benefit the shaders without having to re-build them

Physics Jacobian Shaders

Example Function Prototype

```
unsigned int BuildJDBall(Constraint *c, Manifold *m, RigidBody *rblist, float fps,  
                        float error, float dscale, Jacobian *jlist,  
                        CommonTrig *trig_funcs, CommonFunc *common_funcs,  
                        ConstraintFunc *constraint_util);
```

- CommonFunc contains pointers to standard functions stored in the physics kernel
- Printf, Dma(get,put), etc...

Physics Jacobian Shaders

Example Function Prototype

```
unsigned int BuildJDBall(Constraint *c, Manifold *m, RigidBody *rblist, float fps,  
                        float error, float dscale, Jacobian *jlist,  
                        CommonTrig *trig_funcs, CommonFunc *common_funcs,  
                        ConstraintFunc *constraint_util);
```

- ConstraintFunc contains pointers to constraint utility functions that live in the physics kernel
- Generating test vectors for limits
- Constraint smoothing
- Shared between all constraints that have limits so optimization is a great benefit

Physics Solver Shaders

Example Function Prototype

```
void SolverSim(SimPool *sim_pool, Manifold *m, char *dimensions, int *jd_build_ea,  
               int *jd_build_size, ManagedLS *allocator, CommonFunc *common_funcs,  
               CommonTrig *trig_funcs, ConstraintFunc *constraint_util);
```

- One of many solver shaders that get loaded by the main physics kernel
- Full Simulation, IK, or “cheap” objects
- jd_build_ea/size tells us about our Jacobian functions (where they live / size)
- Local store allocator provided for scratch

Custom Event Shaders

- Anyone can author their own custom event shader for physics
- Currently we have two custom event shaders.
- The physics kernel passes common functions and a list of DMA tags

Custom Event Shaders

- Work memory is passed from to kernel to accommodate any temporary data. Currently this is 2k
- Shader author can DMA new data to a PPU buffer of choice

Phase 3

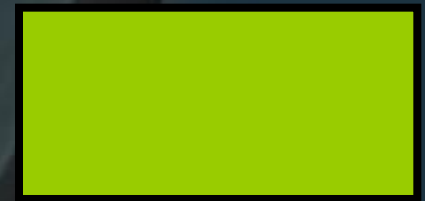
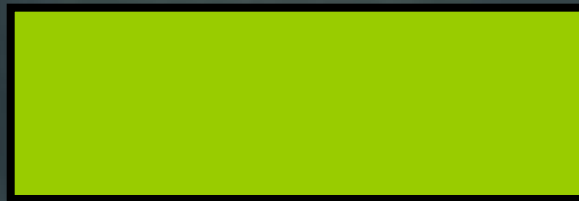
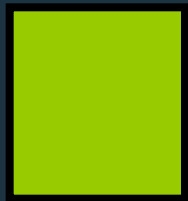
Resistance 2

- Immediate and Deferred Modes
- Constraint Data Streaming
- Using library shaders for collision

Phase 3

Resistance 2

Physics Update

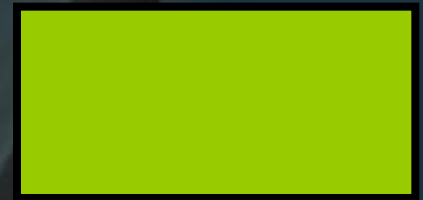
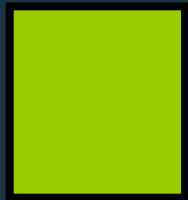


Create Entity (moby) Lists
Cache Collision Geometry
[Immediate Jobs]

Phase 3

Resistance 2

Physics Update

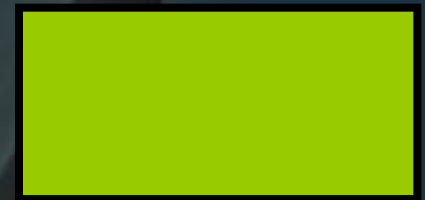
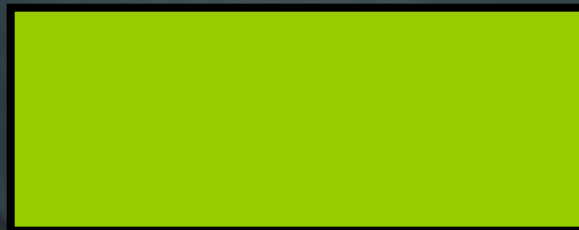
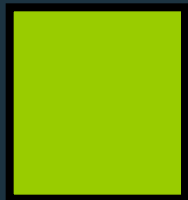


Start Immediate Jobs

Phase 3

Resistance 2

Physics Update

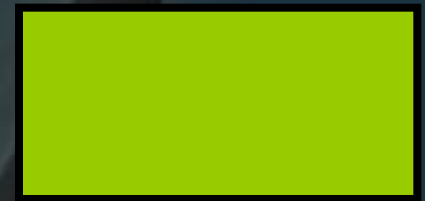
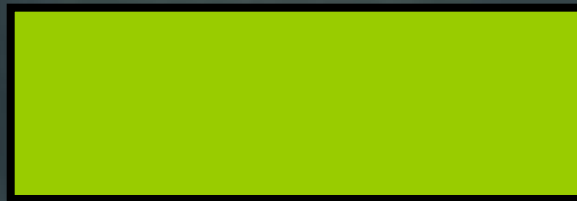
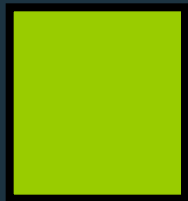


Update Immediate Physics Jobs

Phase 3

Resistance 2

Physics Update



Sync Immediate Physics Jobs

Phase 3

Resistance 2

Physics Update

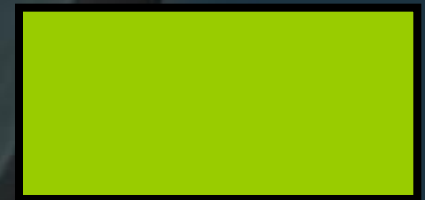
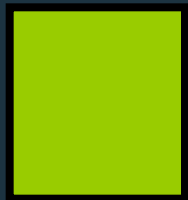


Call Events [immediate]

Phase 3

Resistance 2

Physics Update

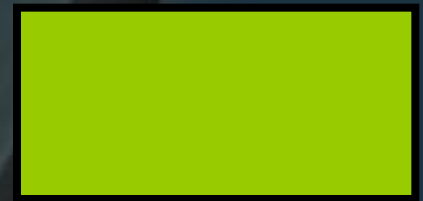
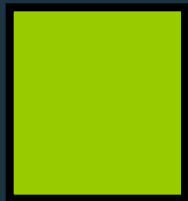


Create Entity (moby) Lists
Cache Collision Geometry
[Deferred Jobs]

Phase 3

Resistance 2

Physics Update

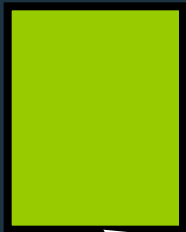


Start Deferred Physics Jobs

Phase 3

Resistance 2

Physics Update

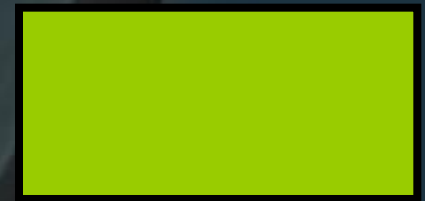
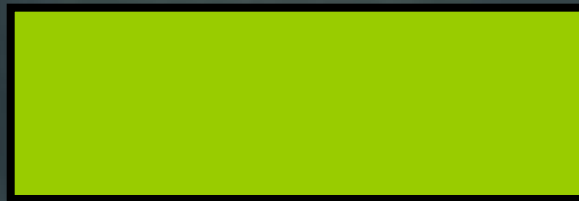
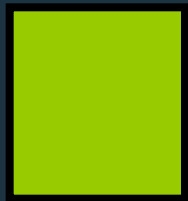
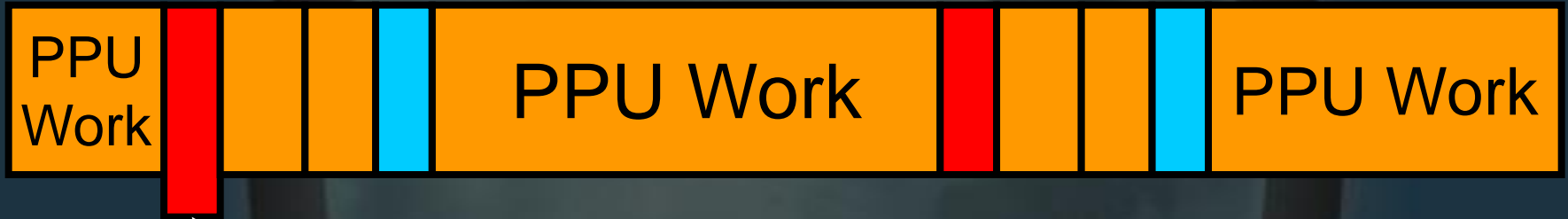


Update Deferred Jobs

Phase 3

Resistance 2

Physics Update

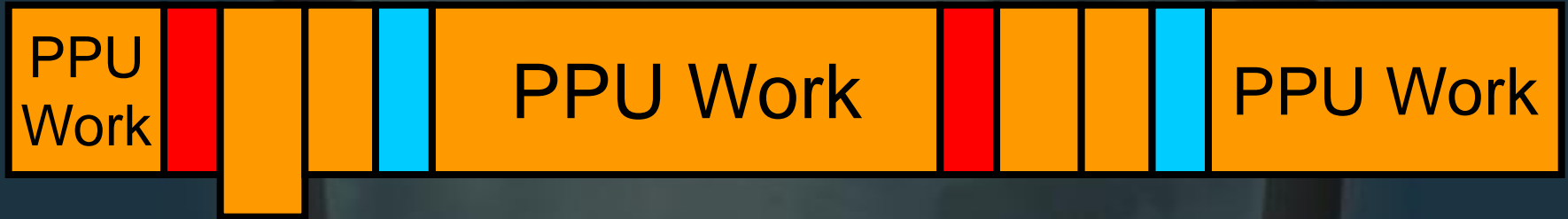


Sync Deferred Physics Jobs

Phase 3

Resistance 2

Physics Update



Call Events [deferred]

Immediate and Deferred Modes

- Physics objects that had no other gameplay or animation based dependencies didn't need to finish in one frame
- Ragdolls had a one frame immediate update and then defaulted to deferred so they could reflect one frame of simulation without “popping”

Immediate and Deferred Modes

- IK is run in immediate mode because it is being constantly being tweaked by gameplay. Lag is not an option
- Having a deferred process improved our frame rate immensely since the majority of the high volume environments had “fire-and-forget” physics objects

Constraint Data Streaming

- Even with shaders, solver could run out of local store
- Changed the solver update so that only 8 chunks of constraint data were allocated
- Solver chews on data while DMAing next list of constraints

Collision Shader Library

- Having multiple versions of the same type of thing adds more work and you have to optimize more than once.
- Not practical
- Physics native collision routines made available to all
- Great re-use and optimization benefit
- Resistance 2 successfully shipped with this model in place

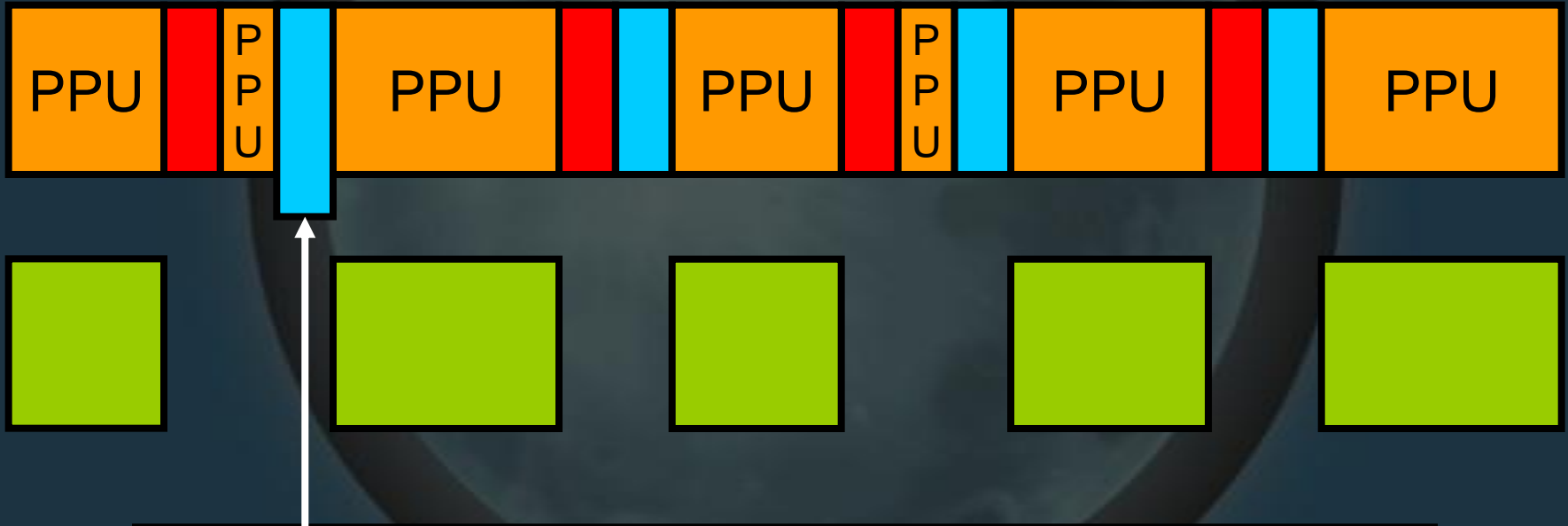
Current Phase

- Building of physics object lists as an SPU job
- Atomic allocation of PPU memory for heavily used data types as well as physics scratch memory
- Use of library shaders for broad phase collision caching

Phase 3

Resistance 2

Physics Update

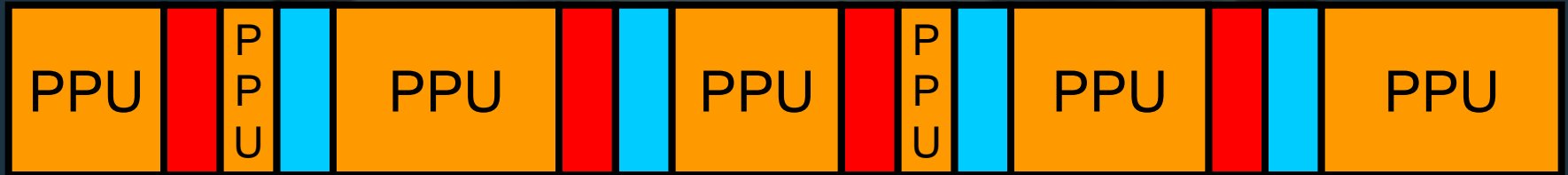


Start Entity Gathering & Collision Caching
SPU Job [for immediate jobs]

Phase 3

Resistance 2

Physics Update

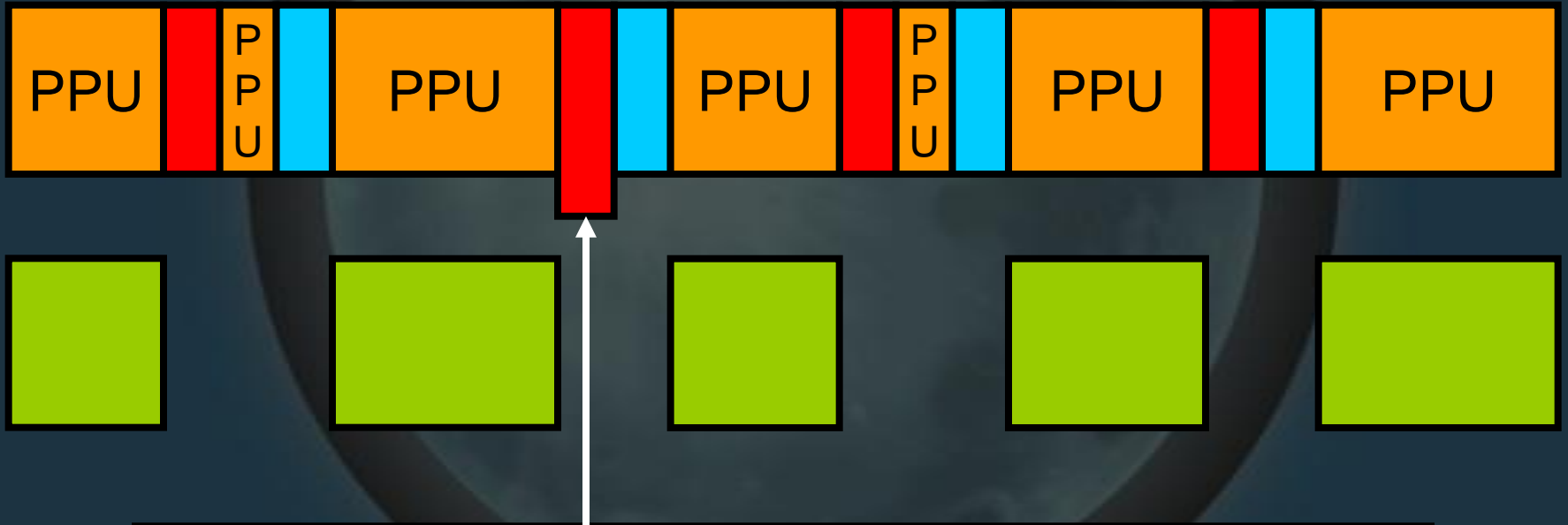


Gather Entities
Cache Collision, Pre-culling

Phase 3

Resistance 2

Physics Update

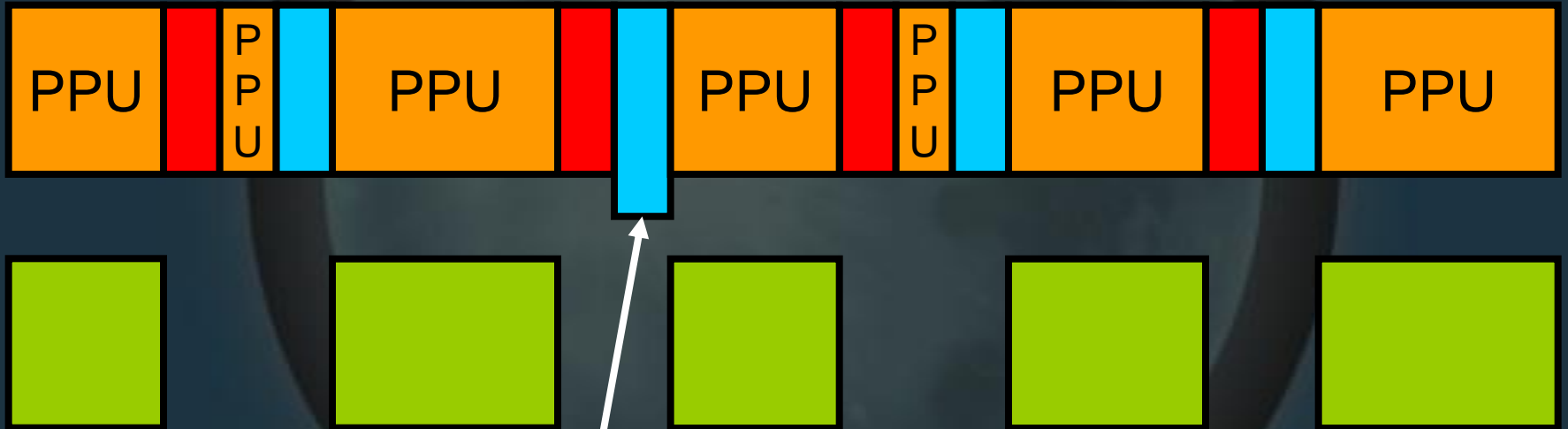


Sync Gathering Jobs [for immediate]

Phase 3

Resistance 2

Physics Update

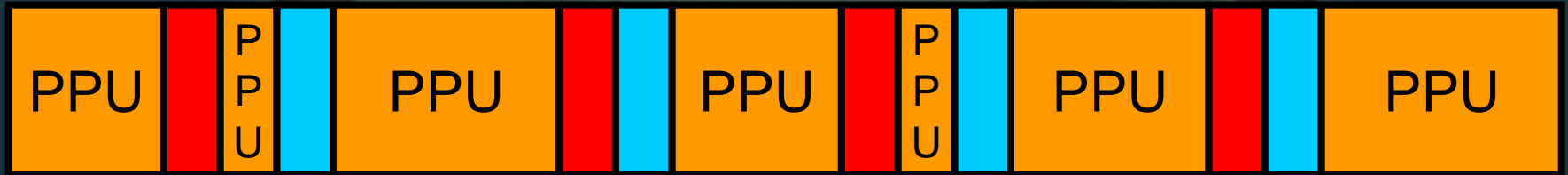


Start Immediate Physics Jobs

Phase 3

Resistance 2

Physics Update



Update Immediate Physics Jobs

Phase 3

Resistance 2

Physics Update

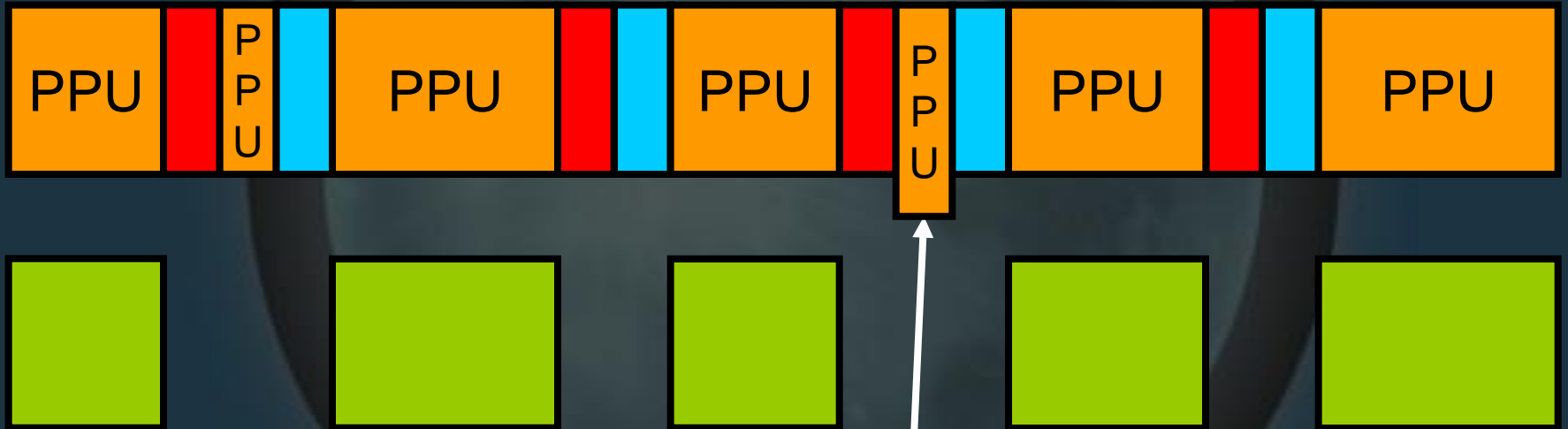


Sync Immediate Jobs

Phase 3

Resistance 2

Physics Update

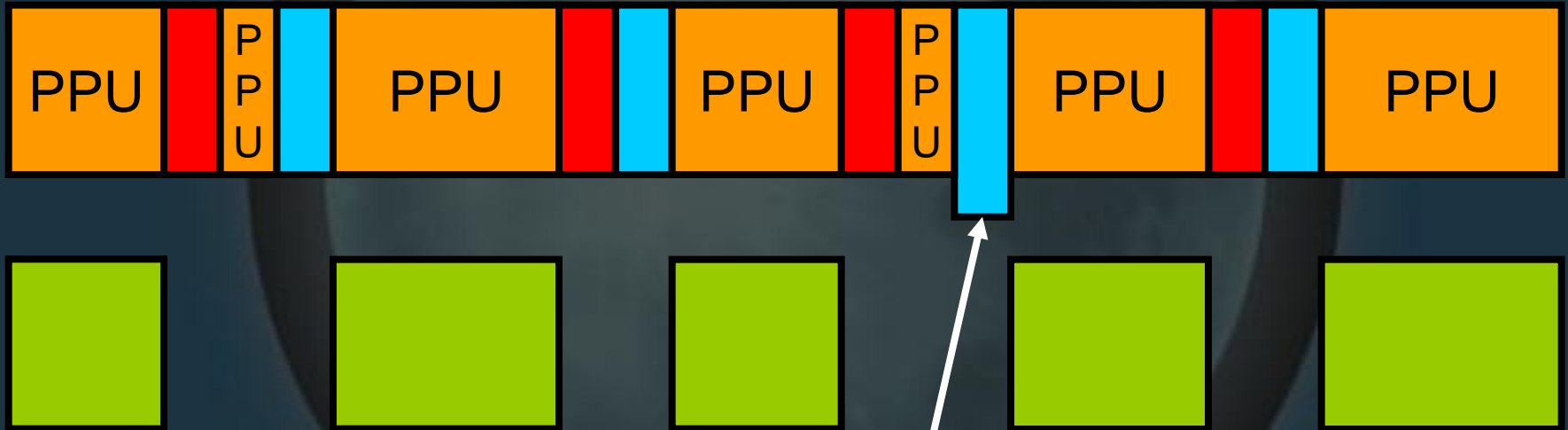


Call Events [immediate]

Phase 3

Resistance 2

Physics Update

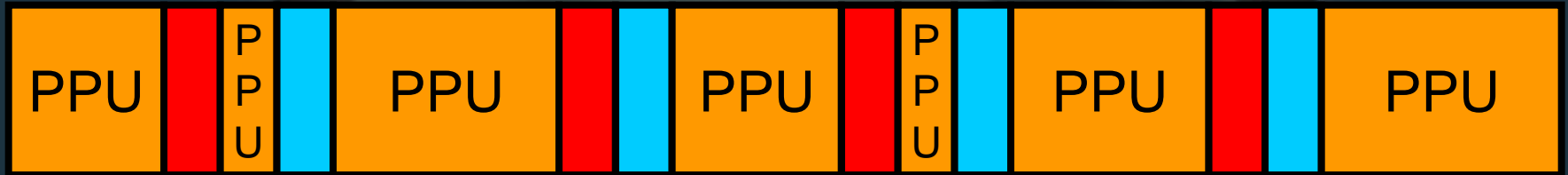


Start Entity Gathering & Collision Caching
SPU Job [for deferred jobs]

Phase 3

Resistance 2

Physics Update



Gather Entities
Cache Collision, Pre-culling

Phase 3

Resistance 2

Physics Update

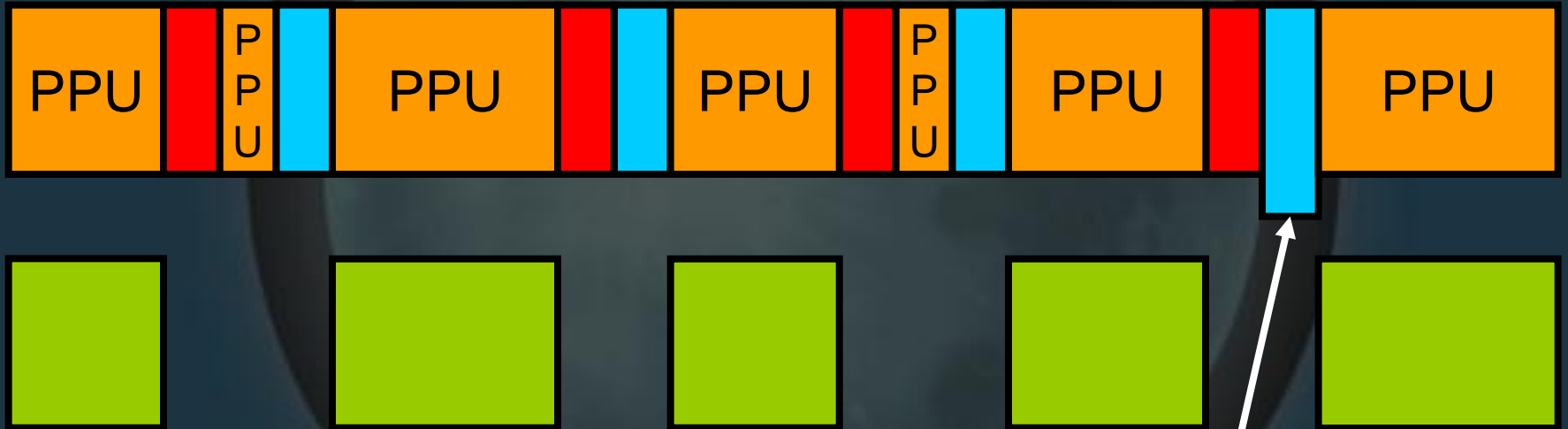


Sync Gathering Jobs [for deferred]

Phase 3

Resistance 2

Physics Update

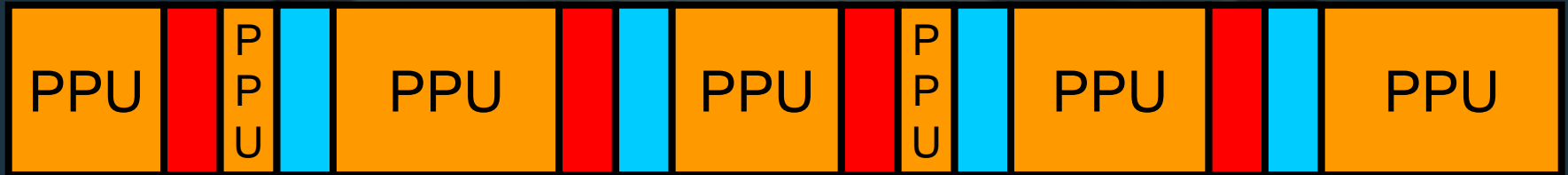


Start Deferred Jobs

Phase 3

Resistance 2

Physics Update



Update Deferred Jobs

Phase 3

Resistance 2

Physics Update



Sync Deferred Jobs

Phase 3

Resistance 2

Physics Update



Call Events [deferred]

Building Object Lists

- Object list building was taking up valuable time
- Caching geometry was a blocked process on the PPU
- Very expensive
- Now all object lists and geometry caches are generated on the SPU

Building Object Lists

- Larger physics data types organized for streaming
- Generating object lists requires allocation of data structures from the PPU
- This includes allocating scratch space for joint re-ordering and packed rigid body data

Atomic Allocation


- Converted PPU fixed block allocations to atomic allocations
- Physics scratch buffer allocation had to be atomic as well
- Rather straight-forward but...
- Exposed a lot of pre-existing problems with the way data was allocated on the PPU

Broad Phase Collision Shaders

- Previously, was only possible to gather game collision geometry on the PPU
- Insomniac Collision System ran on its own SPU
- Now the functions are in a shader library
- We can build physics collision data on the SPU through the use of the shader library interface
- Saved valuable time!

Looking Forward

- Optimize DMAs
- Better data organization
- Convert more of the physics kernel into Shaders
- Find more opportunities for interleaving SPU update with PPU



Eric Christensen
Insomniac Games
Principal Engine Programmer
ec@insomniacgames.com