ARCHITECTING JOLT PHYSICS FOR HORIZON FORBIDDEN WEST

JORRIT ROUWÉ – GUERRILLA

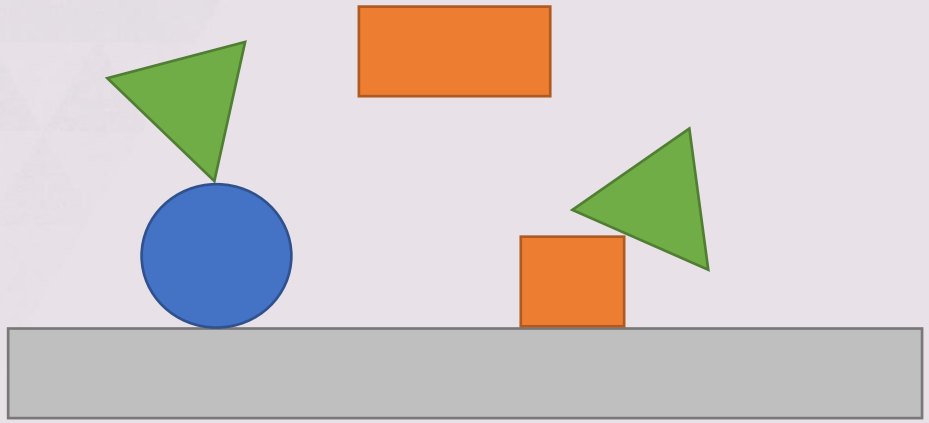
GDC 2022

CONTENTS

- Typical Physics Update
- Waits
  - Streaming Open World
  - Game Object Update
  - Background Queries
- Solutions
  - Lock free broad phase
  - Lock free island building
- Conclusion

CONTENTS

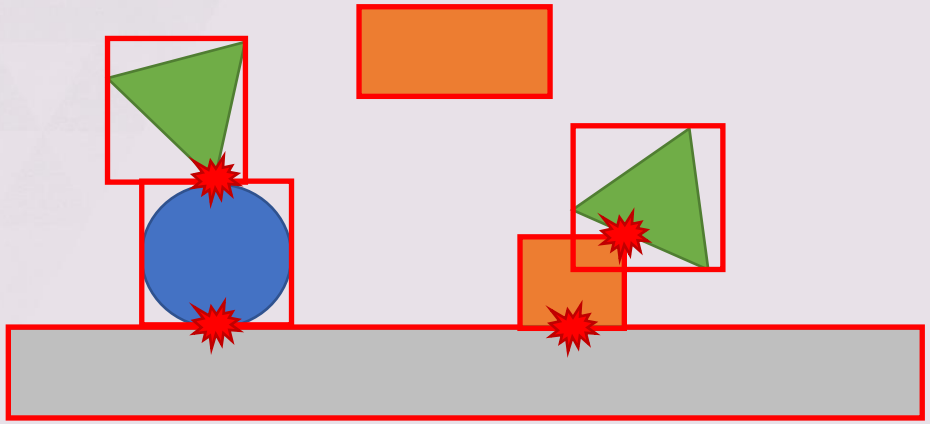
THE SCENE



Dynamic rigid body
(non-deformable shape)

Static floor

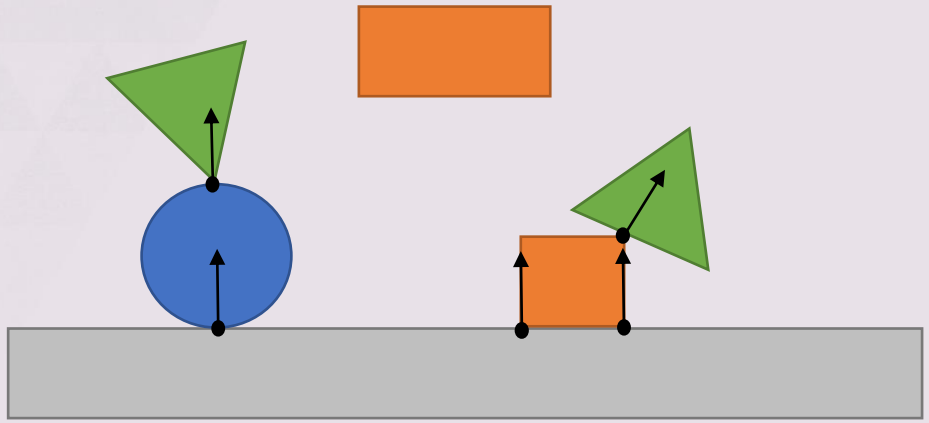
BROAD PHASE COLLISION DETECTION



Axis aligned bounding box

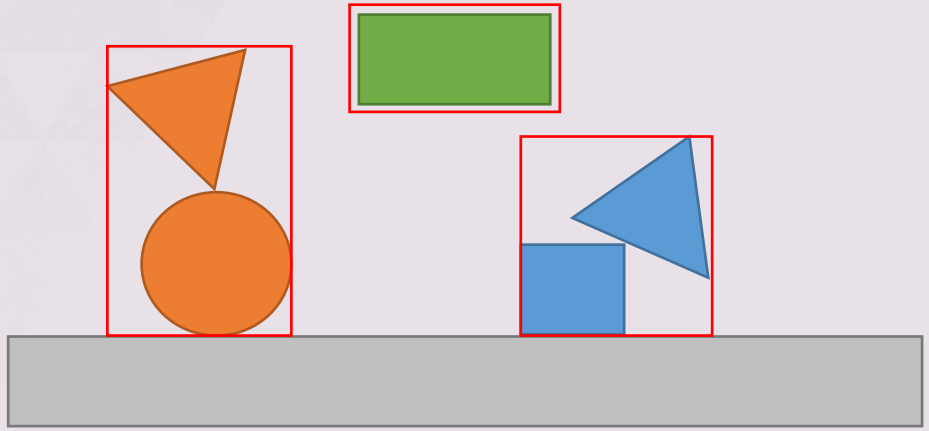
Overlap detected between body pairs

NARROW PHASE COLLISION DETECTION



Point/face and direction of least penetration.

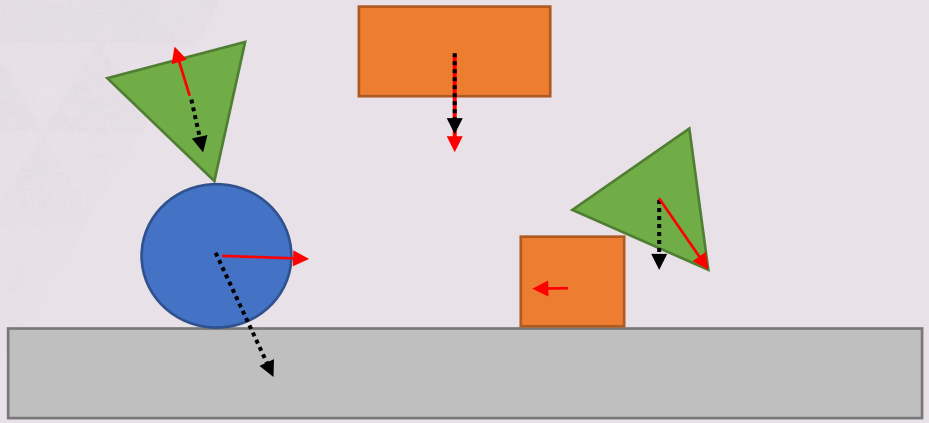
GENERATE ISLANDS



Simulation island of connected bodies.

Floor cannot move so not part of island.

SOLVE (CONTACT) CONSTRAINTS



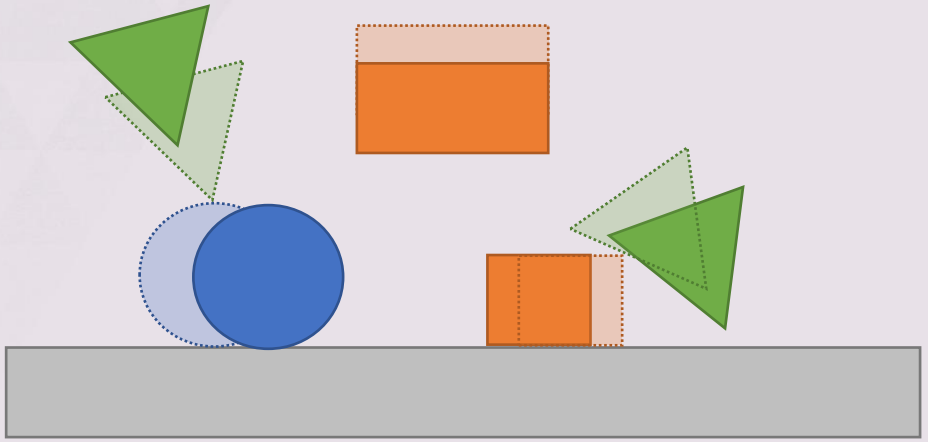
Velocity at beginning of step

Velocity after gravity and constraints solved

- Jolt uses Sequential Impulse Solver
- See Erin Catto at GDC 2009

Jolt Physics = Box 3D

INTEGRATE POSITION: VELOCITY · DELTA TIME



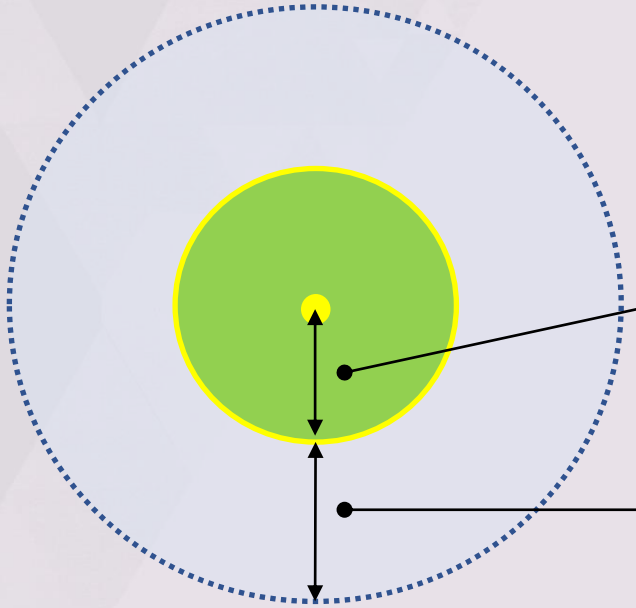
Old position

New position

- Jolt uses Symplectic Euler integration

CONTENTS

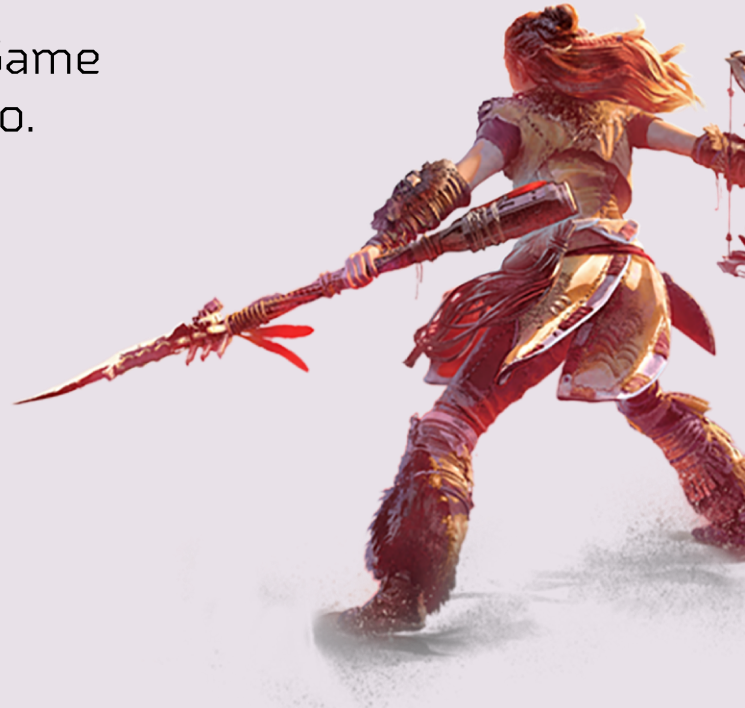
OPEN WORLD - SCENES



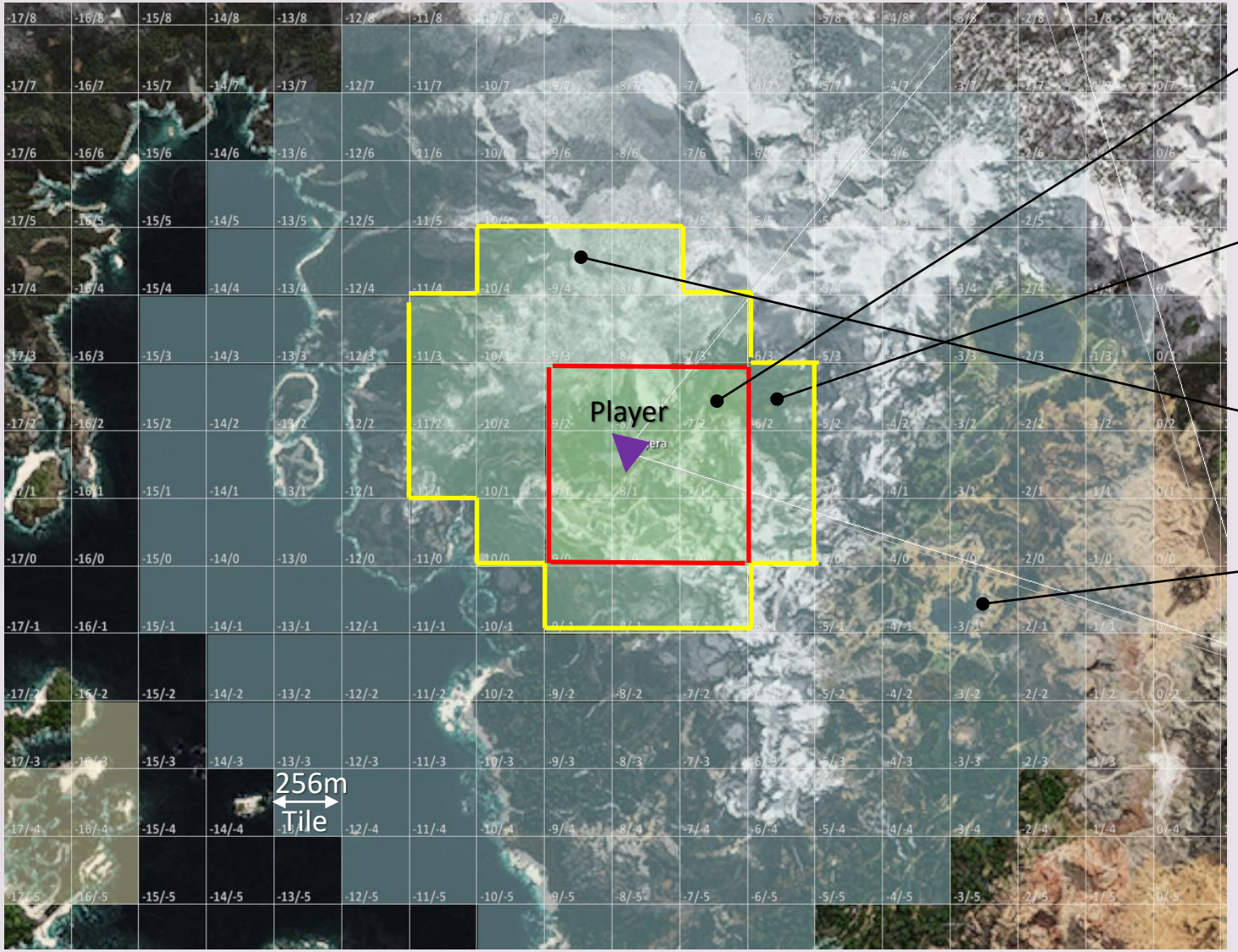
Scene = Script + Objects

Max 125m 'encounter': Game objects spawn / move to.

Max 200m 'activation': Player can interact.



OPEN WORLD - TILES



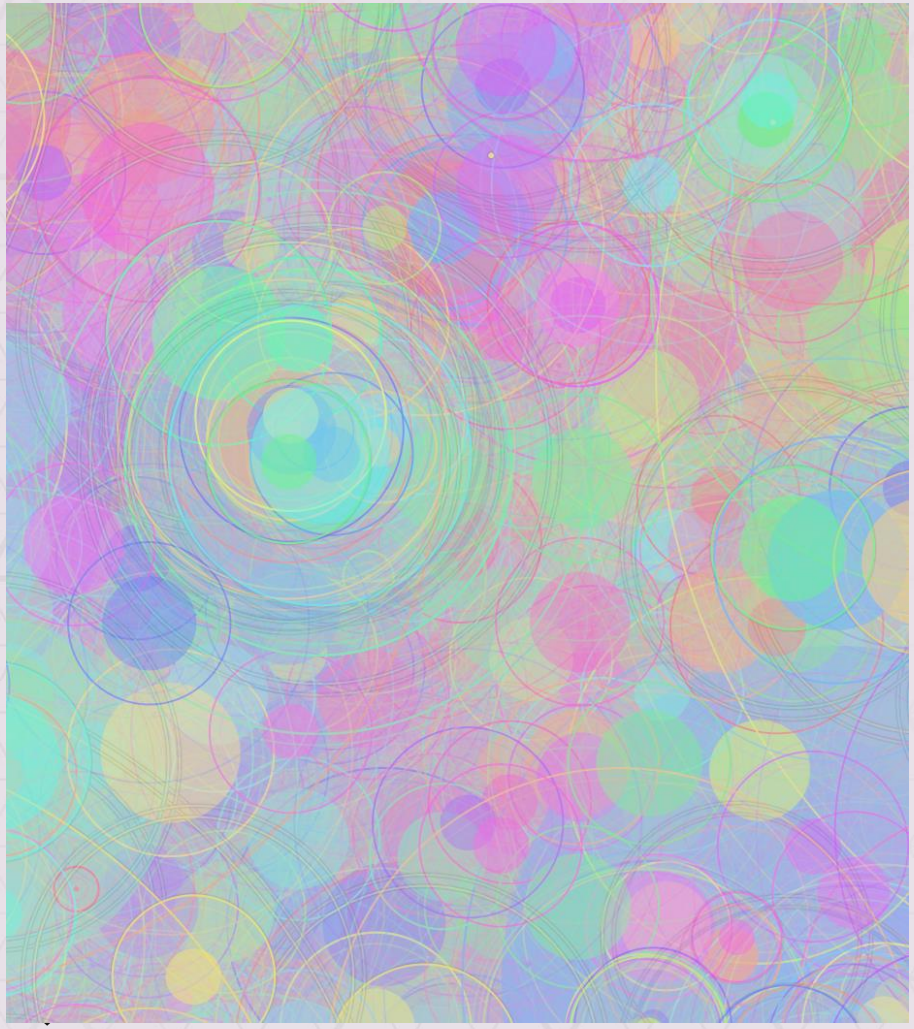
9 high LOD:
- Full graphics
- Full collision

19 medium LOD:
- Medium graphics
- Full collision

Extra row to cover loading time

Low LOD:
- Low graphics
- No collision

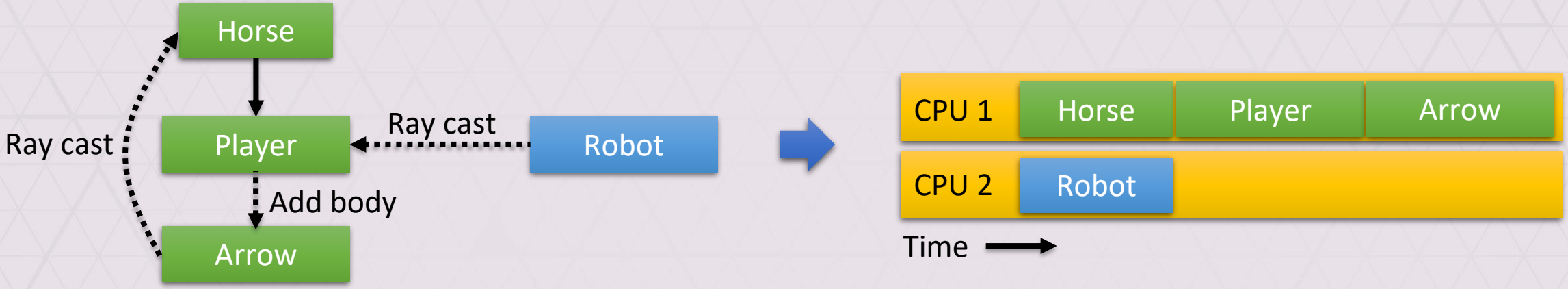
OPEN WORLD - REQUIREMENTS



- 10s of Scenes
- 28 Tiles
- 20K bodies
- Constantly changing
- Situation: Block main thread 5-10 ms
- Want: Insert on background thread!

CONTENTS

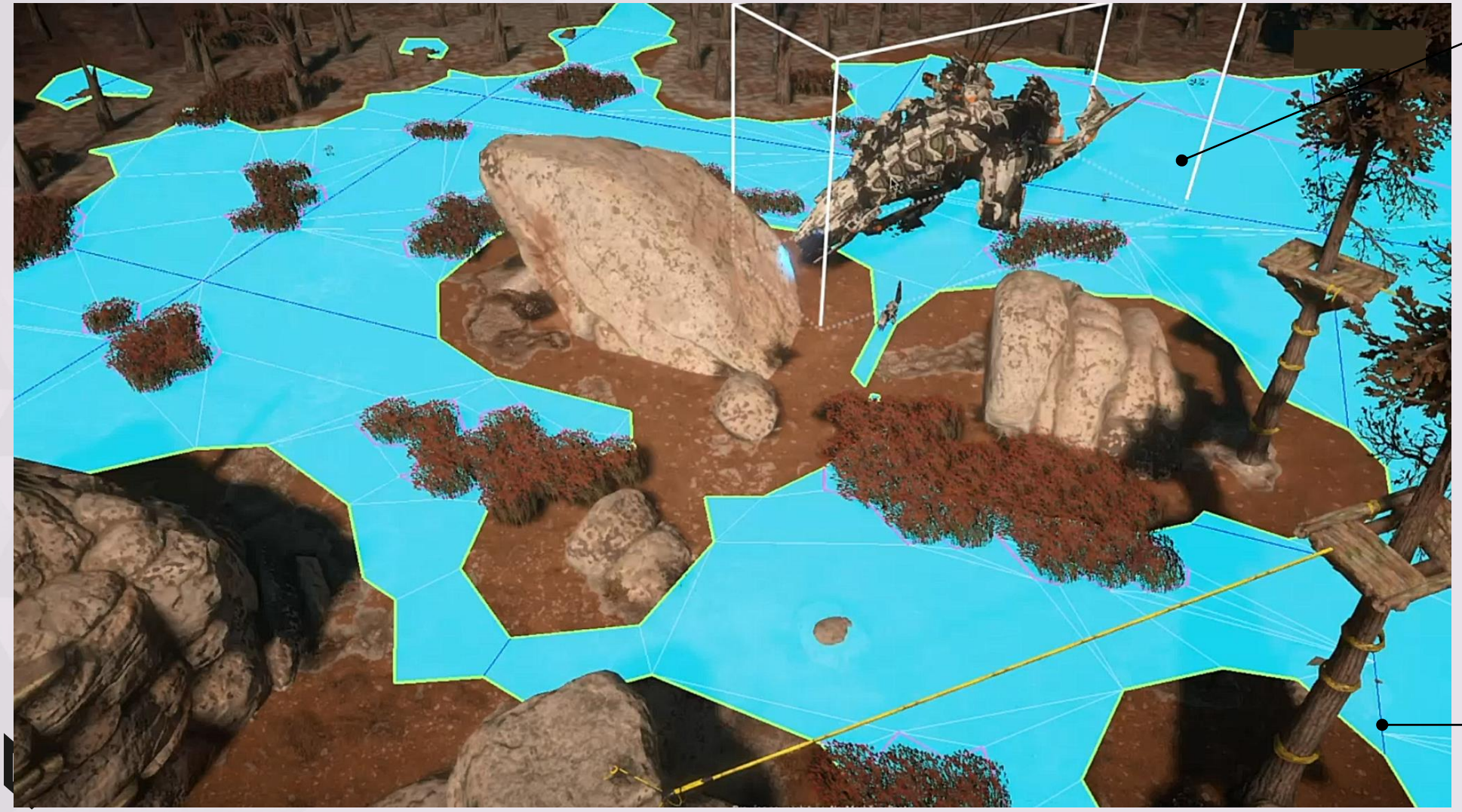
PARALLEL GAME OBJECT UPDATE



- Independent game objects on different threads
- Global systems = Lock contention
- PS5 CPUs > PS4 CPUs
- See GDCE 2014 presentation

CONTENTS

BACKGROUND NAVMESH JOB



Walkable area.

- Query 12.8x12.8m area.
- Recast on background thread.
- > 1 frame.

Other threads continue = contention!

Tile border

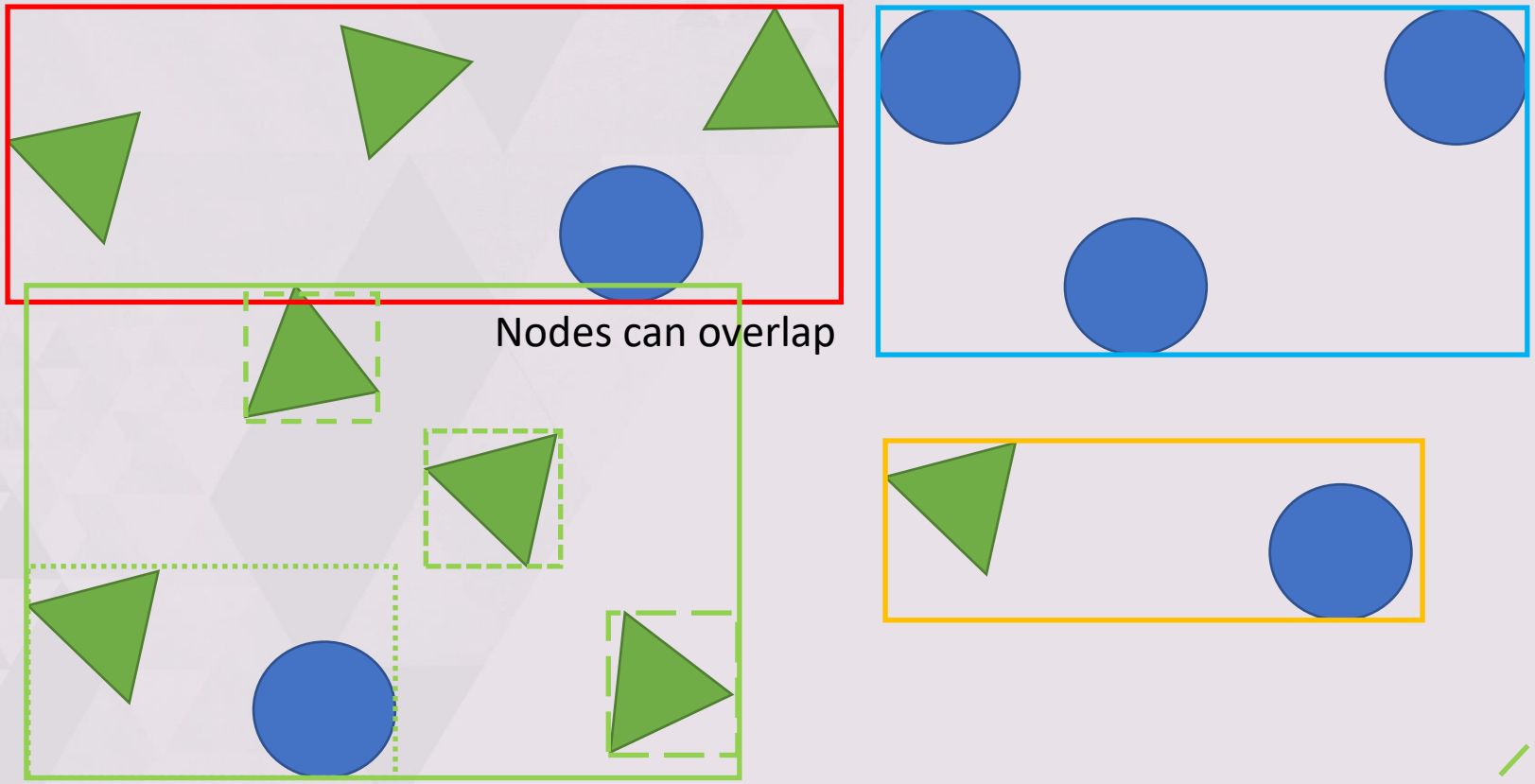
CONTENTS

BROAD PHASE OVERVIEW

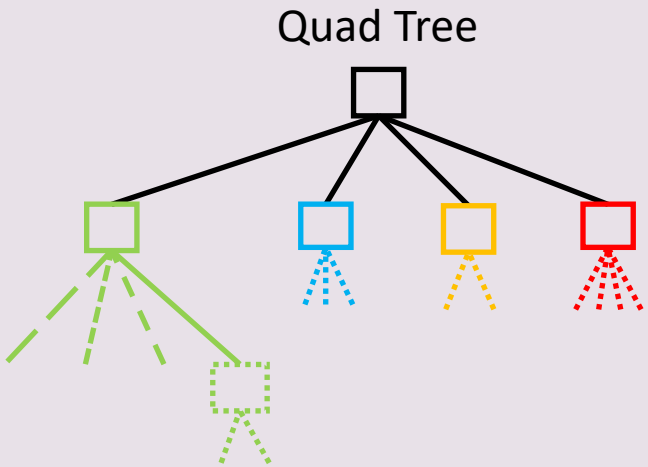
- Waiting on concurrent
  - Add, modify, remove
  - (Background) query

- Solution
  - A Quad Tree
  - Lock free (mostly)

QUAD TREE



- 4 children per node
- SIMD friendly

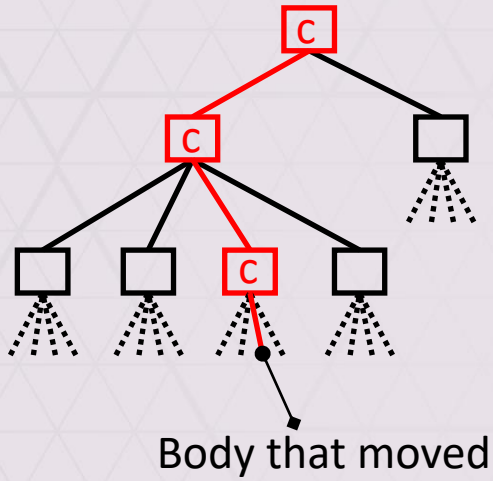


QUAD TREE - NODE

Type	Name
AtomicFloat	BoundsMinX[4]
AtomicFloat	BoundsMinY[4]
AtomicFloat	BoundsMinZ[4]
AtomicFloat	BoundsMaxX[4]
AtomicFloat	BoundsMaxY[4]
AtomicFloat	BoundsMaxZ[4]
AtomicUInt32	ChildNodeIdx[4]
AtomicUInt32	ParentNodeIdx
AtomicBool	Changed

- Child bounding boxes in SIMD format
- Initialized to [LargeFloat, -LargeFloat]
- ChildNodeIndex refers to node or body
- ParentNodeIndex refers to parent
- Changed flag set if bounds too large
- Using atomic operations

QUAD TREE – MOVING BODY

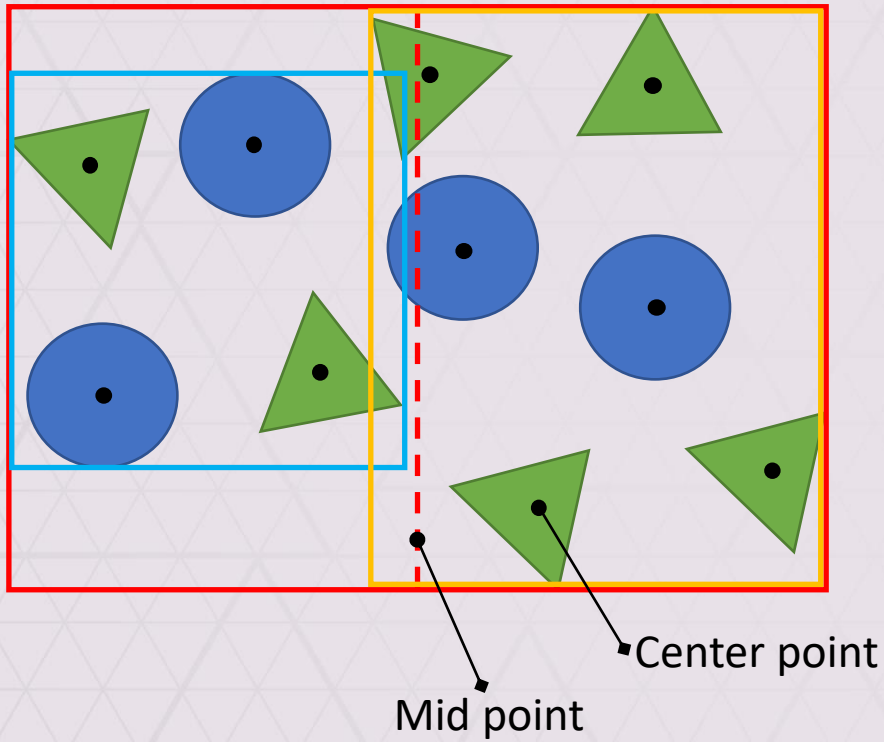


C Node flagged as ‘changed’

| Widened bounding box

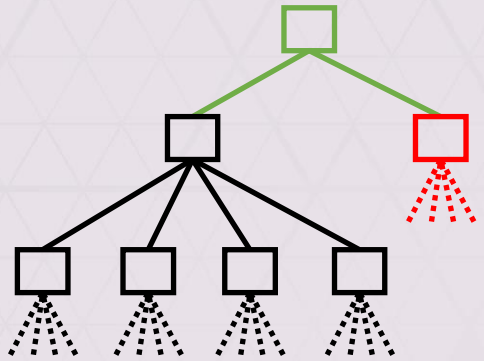
- Body stores (NodeIndex, ChildIndex)
- Body moves
  - Calculate bounding box
  - AtomicMin / AtomicMax to update node
  - Mark ‘Changed’
  - Go to ParentNodeIndex, repeat
- Query: No collisions missed!
- Performance degradation, need tree rebuild!

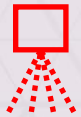
QUAD TREE – ADDING BODIES



- Loading thread: Sub-tree of objects to add
- Fast tree construction
  - Calculate bounds all objects
  - Split at mid point of longest axis
  - Classify ‘Center Point’ of AABB
  - Repeat to get 4 leaves
  - Recurse to create tree

QUAD TREE – ADDING BODIES – CASE 1





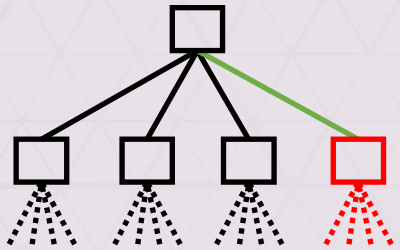
New objects tree

Existing node

Created node
/ connection

- Insertion in tree in O(1)
- Case 1: Root Full?
  - Create new root
  - Mark as 'Changed'
  - Child 1 is old root, AABB [-LargeFloat, LargeFloat]
  - Child 2 is 'new objects' & set parent
  - Compare exchange old root with new root
    - On failure, delete new root and go to case 2
  - Set old root parent to new root

QUAD TREE – ADDING BODIES – CASE 2



New objects tree

Existing node

Created node
/ connection

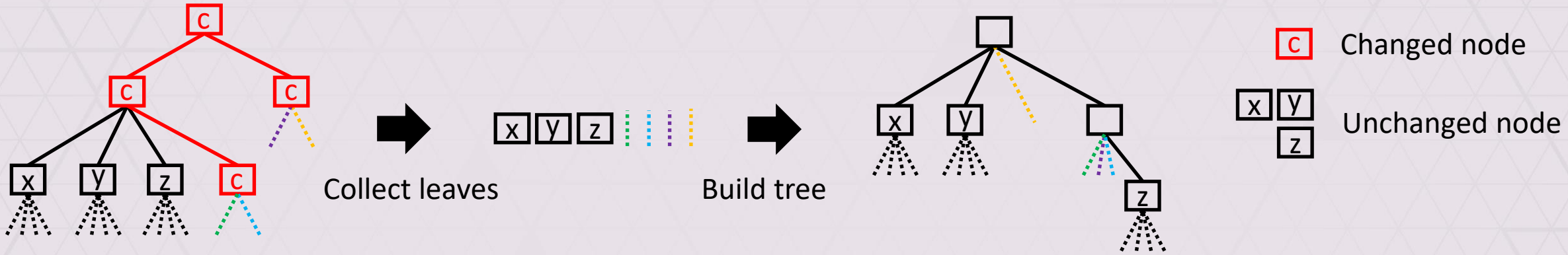
- Case 2: Root has slot?
  - Set parent 'new objects' to root
  - Compare exchange on root: Invalid Index → Node Index
    - On failure go to case 1 or 2
  - Update child bounds in root (Max then Min).
  - Mark root 'Changed'
- Inserting 1 body is bad!

QUAD TREE – REMOVING BODIES

- Invalidate bounding box
  - Set Min to LargeFloat
  - Set Max to -LargeFloat
- Set ChildNodeIdx to Invalid
- Mark node and parents as 'Changed'
- Don't remove entire sub-tree, rebuild mixes bodies
- Remove 1 body in 0.5 µs on PS4

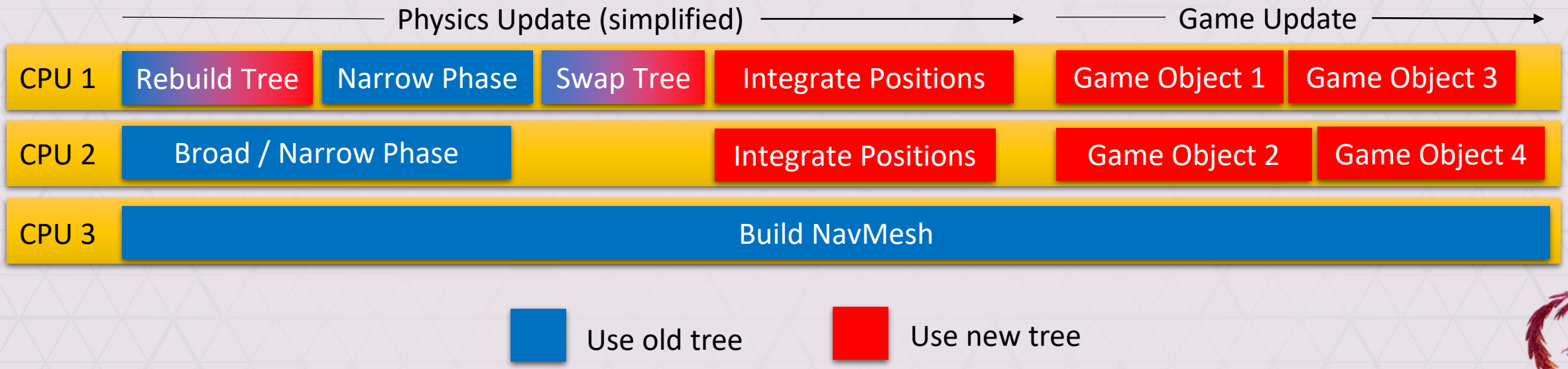
QUAD TREE – REBUILDING

- Not 'Changed' = Single Object



- Use multiple trees (static vs dynamic)

QUAD TREE - REBUILDING



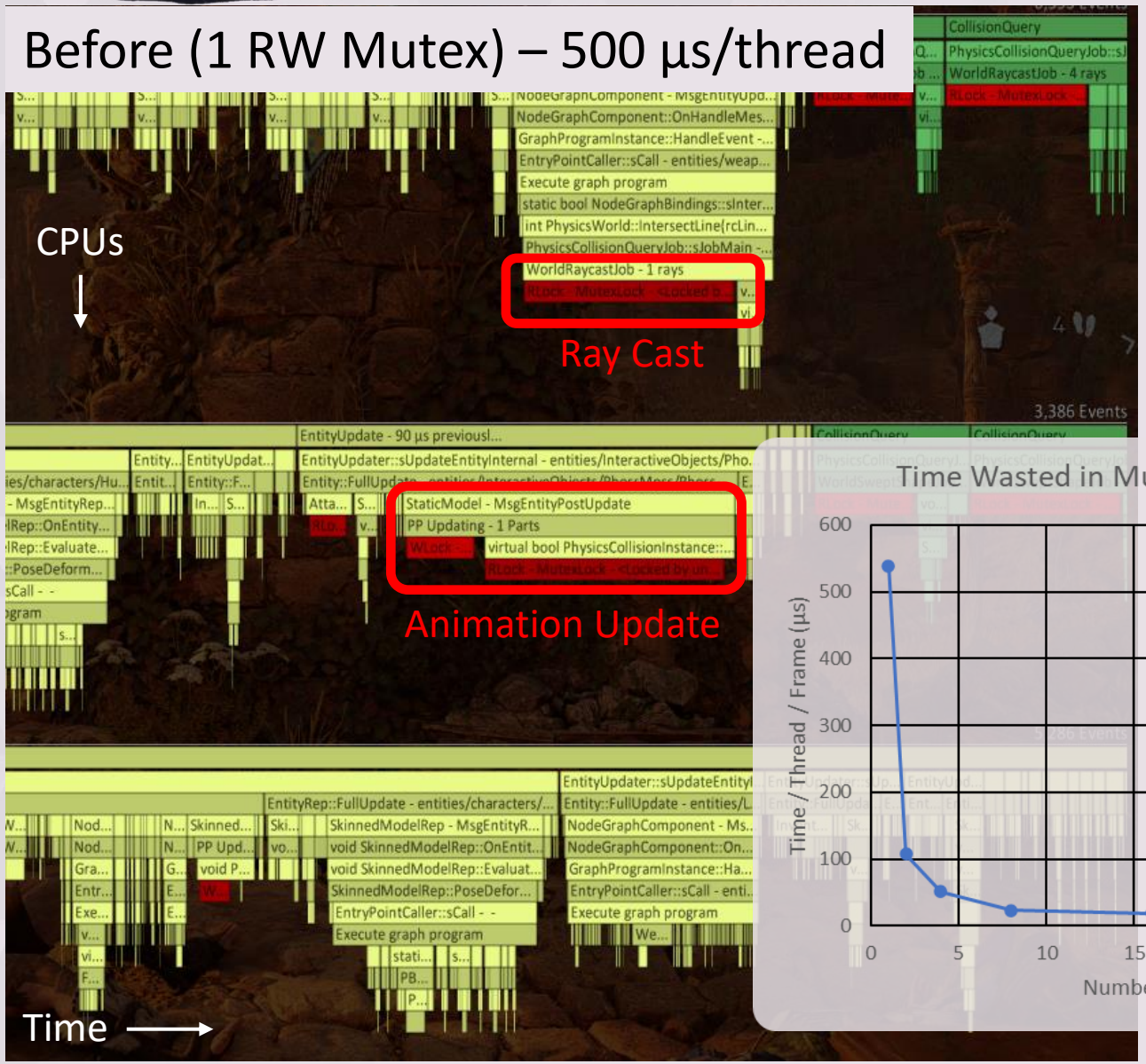
- Keep old tree until next physics update

QUAD TREE – COLLISION QUERY

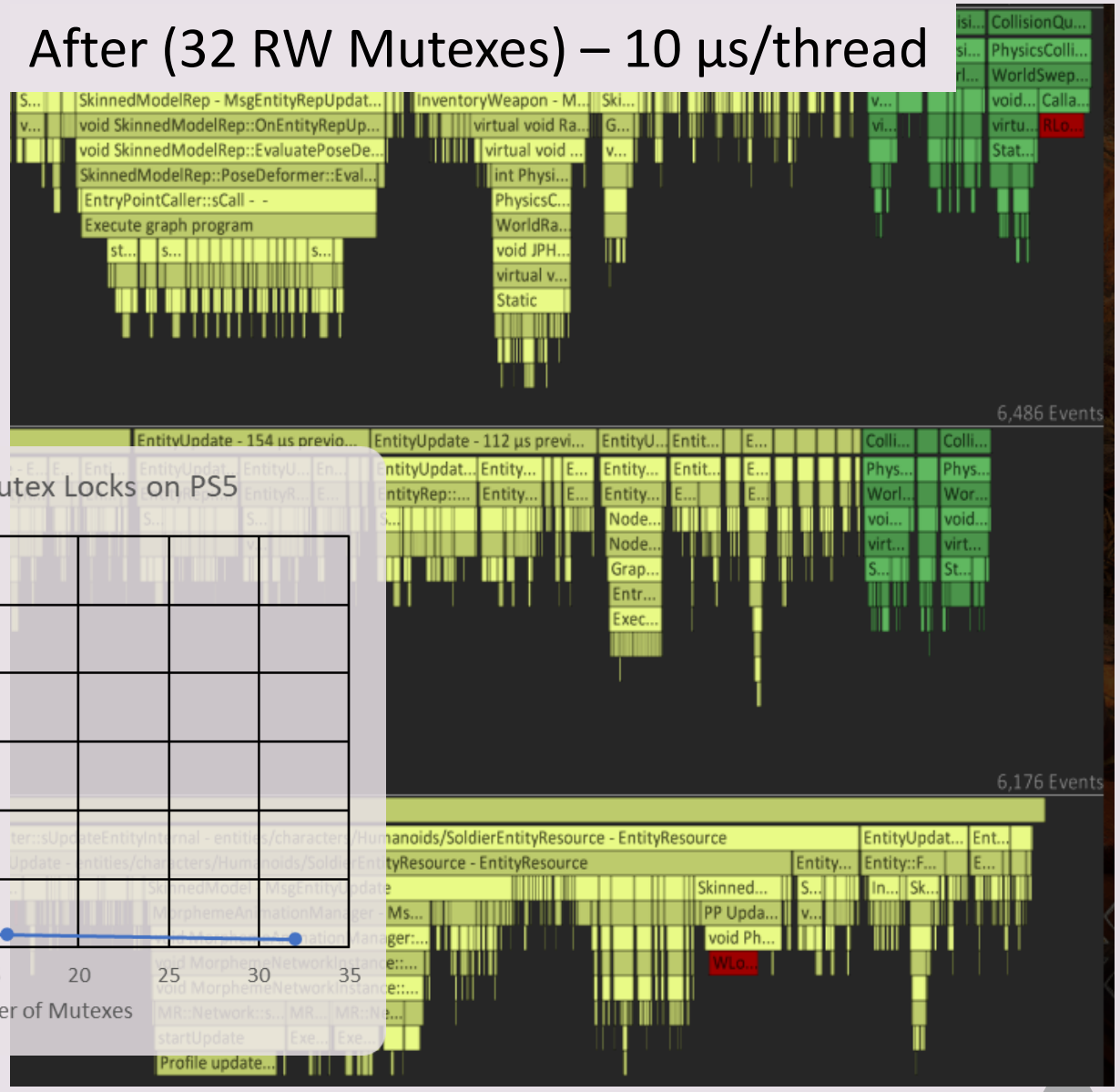
- Lock tree for read (for background jobs)
- Recurse children until leaf (body)
- Lock body mutex for read
  - Get shape (collision volume)
  - Get transform
- Unlock body mutex
- Narrow phase work
- Overhead locks 10-20%, only for Game Object Update

BROAD PHASE - RESULTS

Before (1 RW Mutex) – 500 µs/thread



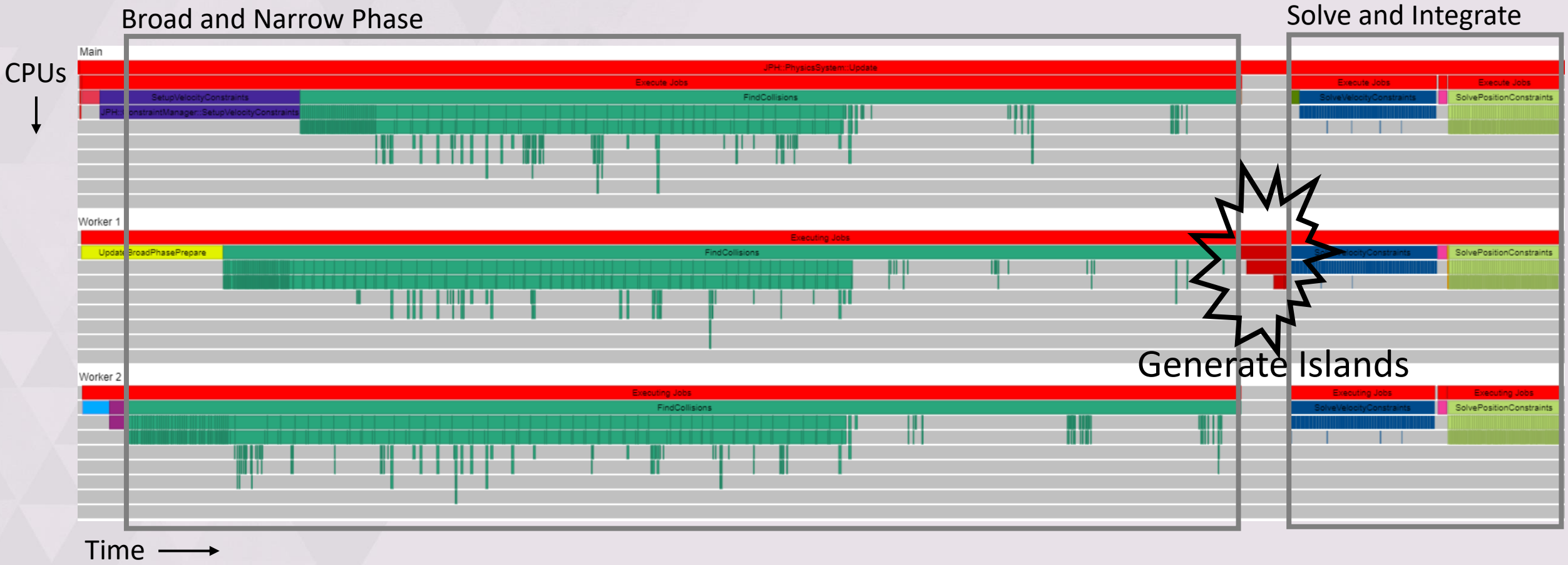
After (32 RW Mutexes) – 10 µs/thread



CONTENTS

- The Physics Update
- Waits
  - Streaming Open World
  - Game Object Update
  - Background Queries
- Solutions
  - Lock free broad phase
  - Lock free island building
- Conclusion

GENERATING ISLANDS - HOW TO SCHEDULE WORK



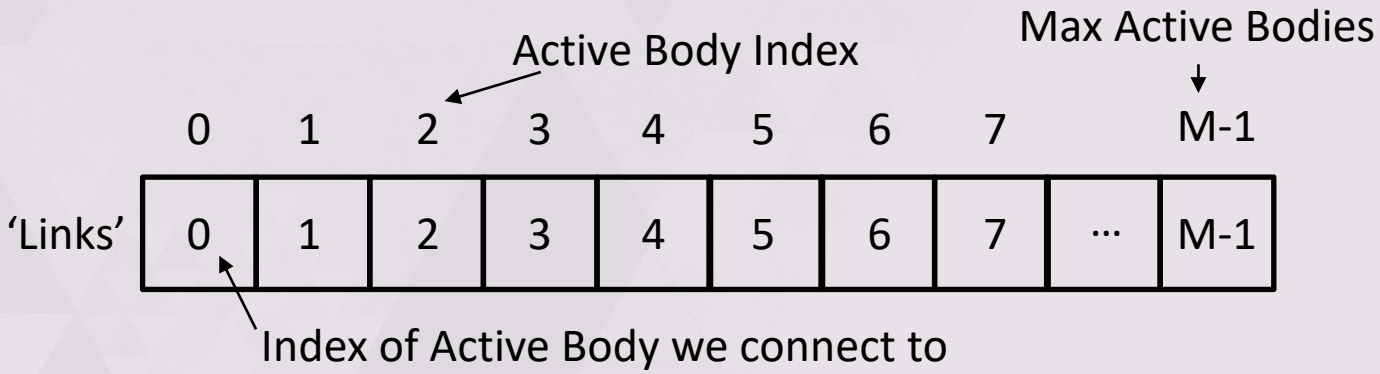
Generating islands single threaded!

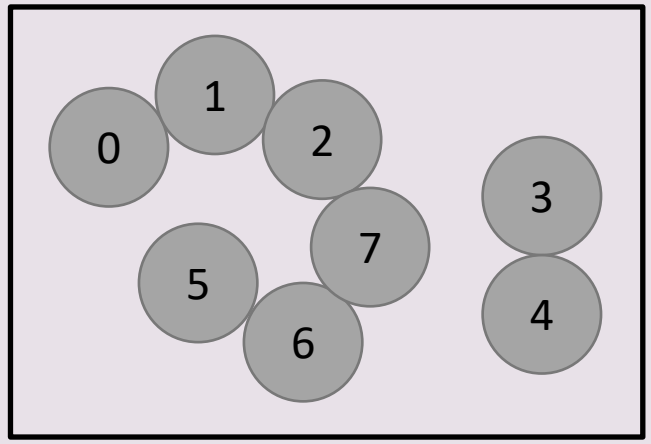
GENERATING ISLANDS - OPTIONS

- Build iteratively
  - Problem: Requires locking during update.
- Build from scratch
  - Problem: Needs connectivity = locking.

- Our Solution
  - Like Union Find with Path Compression.
  - Most work during Narrow Phase.
  - Lock free.
  - Finalize islands in O(N).

ACTIVE BODY LINKS – INITIAL STATE

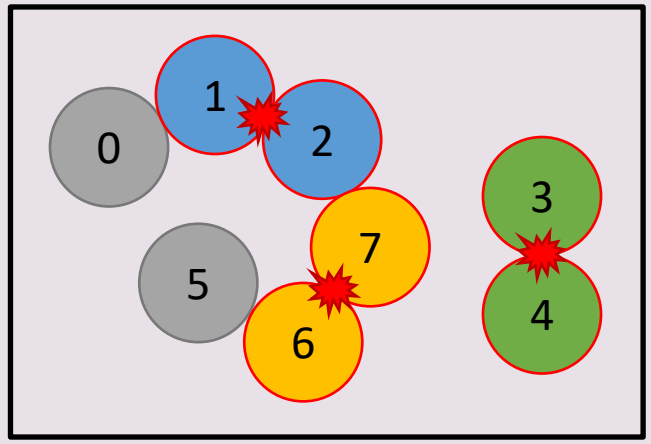


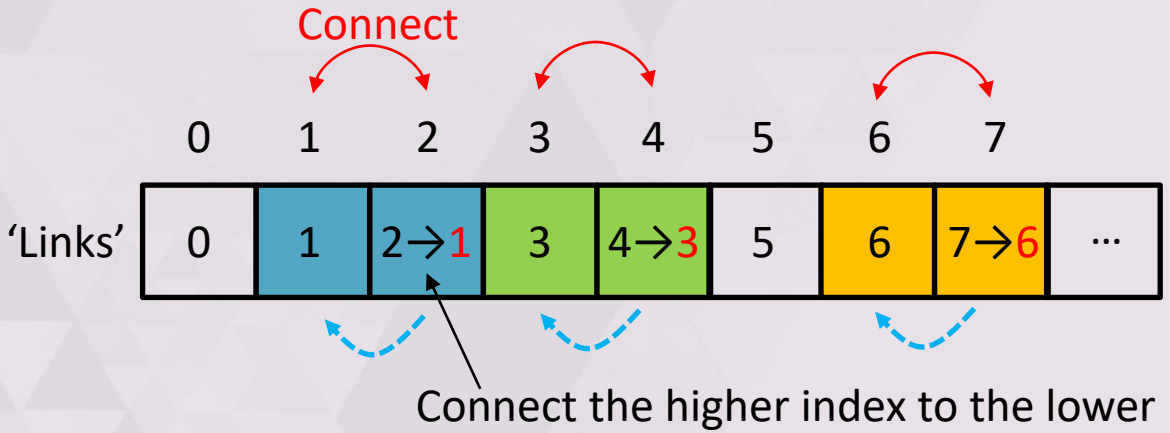


Example

- Initially connect to self.

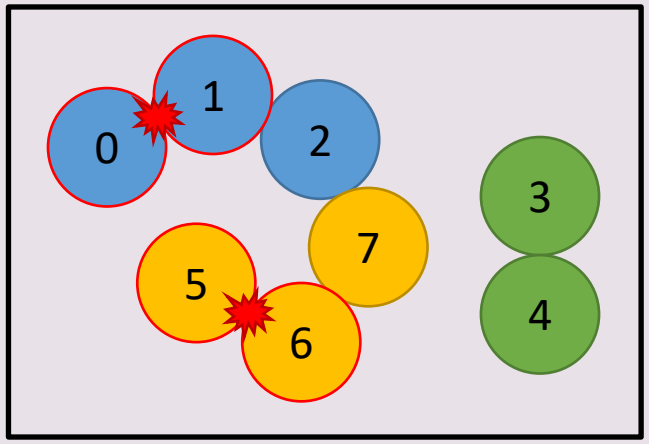
SIMPLE CASE: CONNECT 1 & 2, 3 & 4 AND 6 & 7

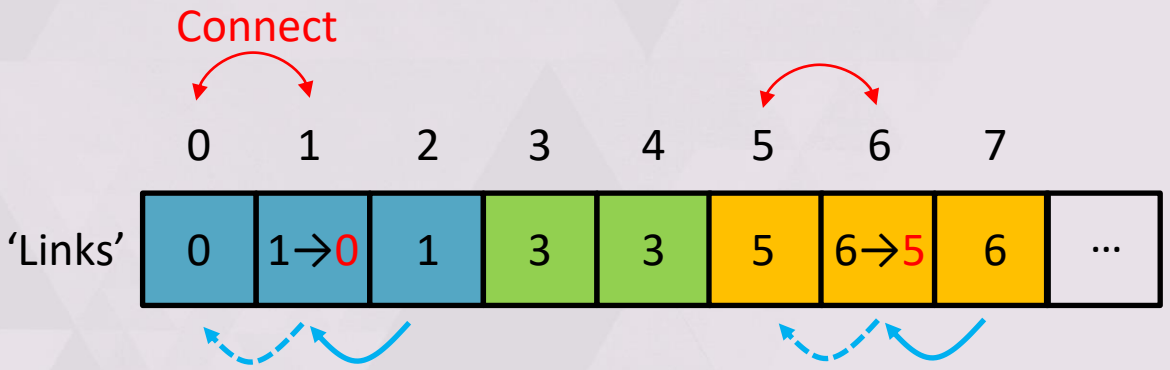




- Connect contacting bodies

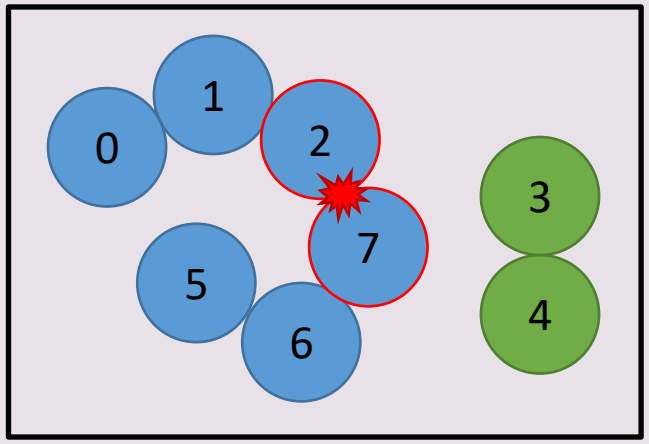
SIMPLE CASE: CONNECT 0 & 1 AND 5 & 6

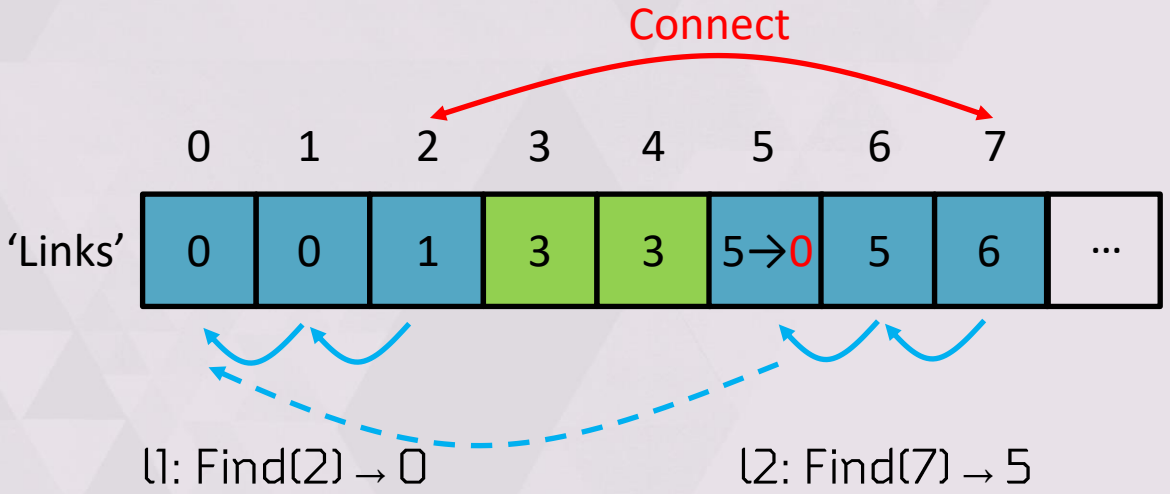




- Forming linked lists
- 'Find' function:
  - Find(7) → 5
  - Find(0) → 0

COMPLEX CASE: CONNECT 2 & 7

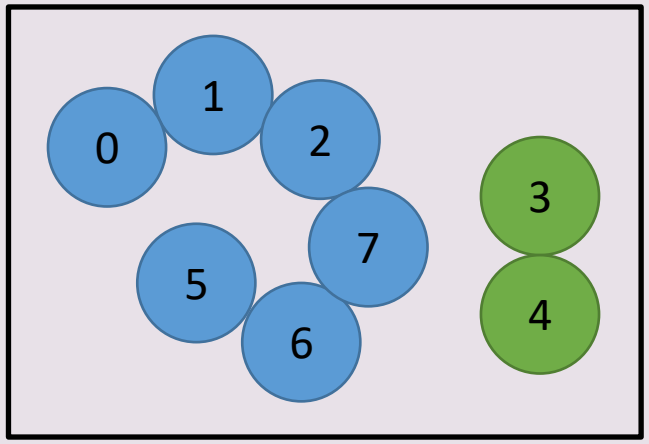


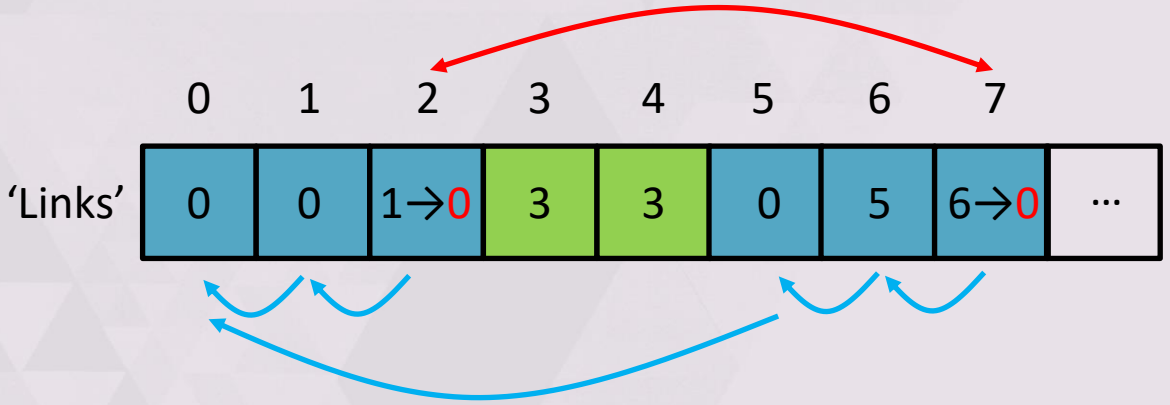


Connect higher index to lower index:
l1 < l2 → 5 connects to 0

```
Union(b1, b2):
  // Find lowest connected body
  l1 = Find(b1)
  l2 = Find(b2)

  // Ensure l1 < l2
  if l1 > l2: swap(l1, l2)

  // Link the bodies
  Links[l2] = l1
```

WORST CASE: FIND IS O(N)

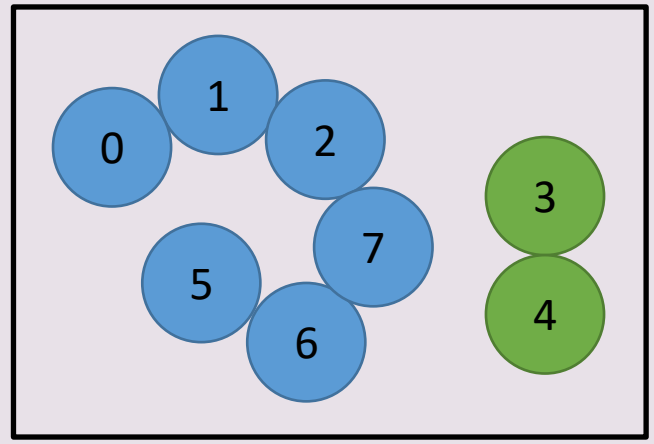


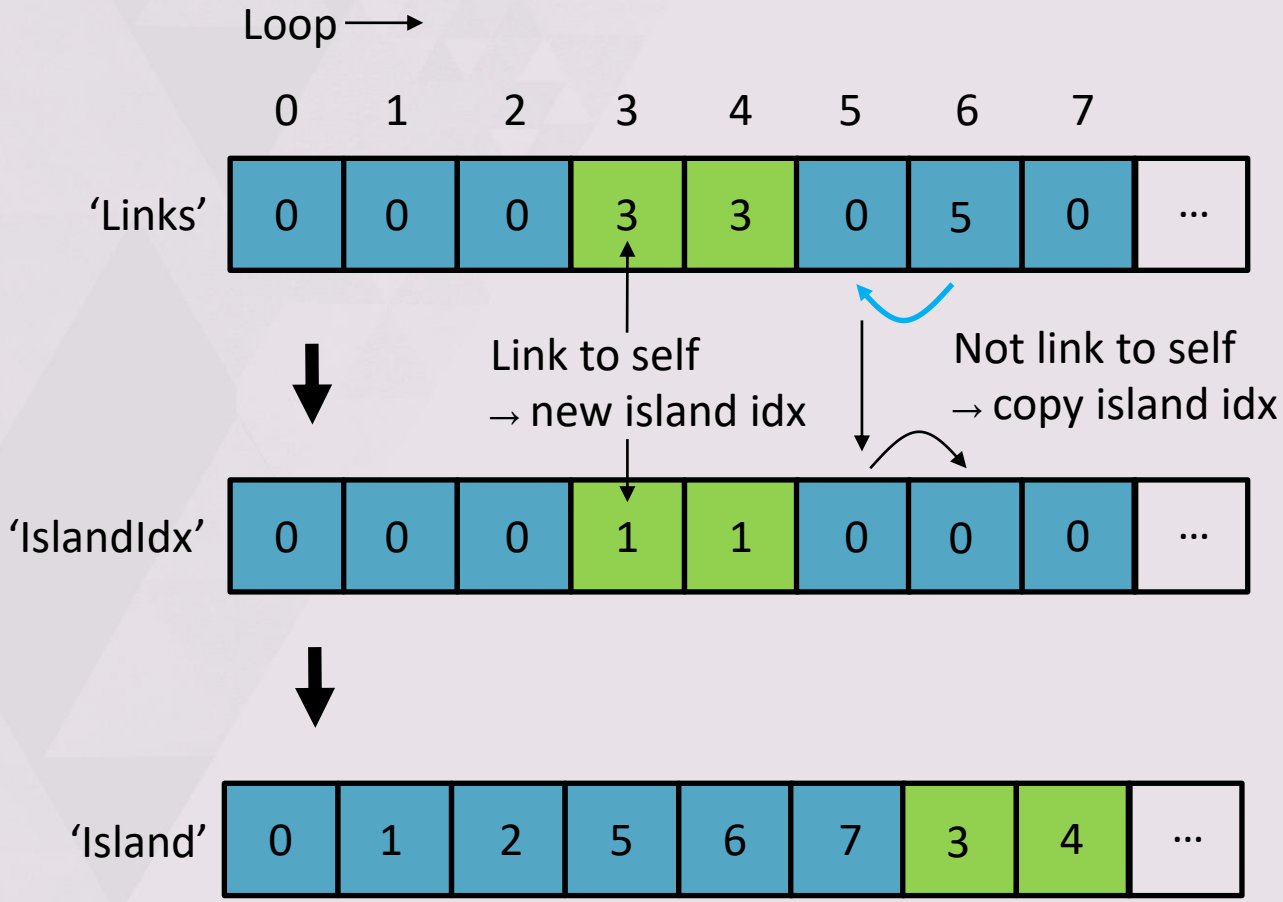


Find = O(N) → Path Compression.

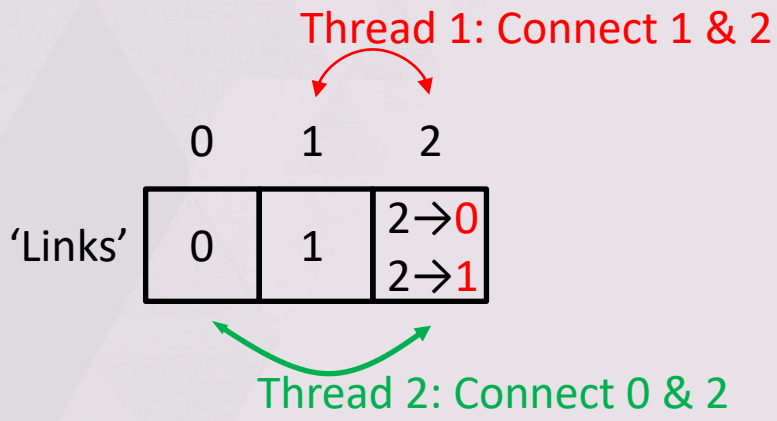
```
Union(b1, b2):
  … as before …

  // Link both bodies to minimum
  Links[b1] = l1
  Links[b2] = l1
```

FINISHING UP (ON 1 THREAD)





MAKING IT THREAD SAFE

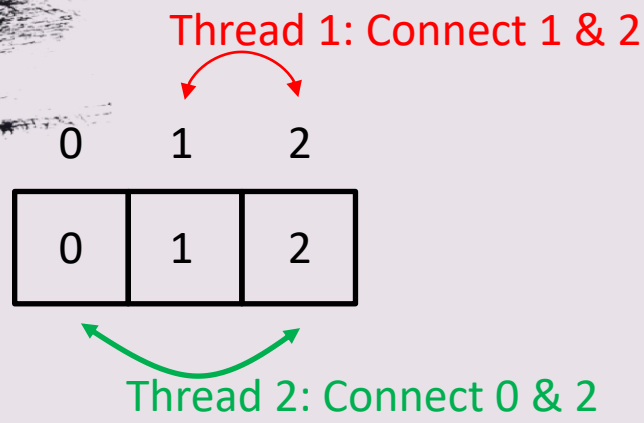


- Race condition:
  - T1: Read l1 = 1, l2 = 2
  - T2: Read l1 = 0, l2 = 2
  - T2: Write Links[2] = 0
  - T1: Write Links[2] = 1

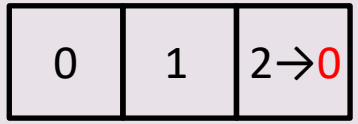
- 2 is **not** connected to 0!

LOCK FREE SOLUTION

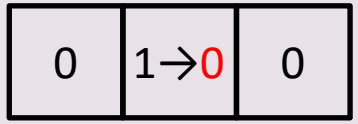
```
Union(b1, b2):
  l1 = b1
  l2 = b2

  while:
    // Search from previous lowest index
    l1 = Find(l1)
    l2 = Find(l2)

    // Ensure l1 < l2
    if l1 > l2: swap(l1, l2)

  // l1 == l2: Already connected
  until l1 == l2 or Links[l2].Exchange(l2→l1)

  // Link both bodies to the minimum
  AtomicMin(Links[b1], l1)
  AtomicMin(Links[b2], l1)
```



- T1: Read l1 = 1, l2 = 2
- T2: Read l1 = 0, l2 = 2
- T2: Links[2].Exchange(2→0)



- T1: Links[2].Exchange(2→1) fail!
- T1: Read l1 = 1, l2 = 0
- T1: Links[1].Exchange(1→0)

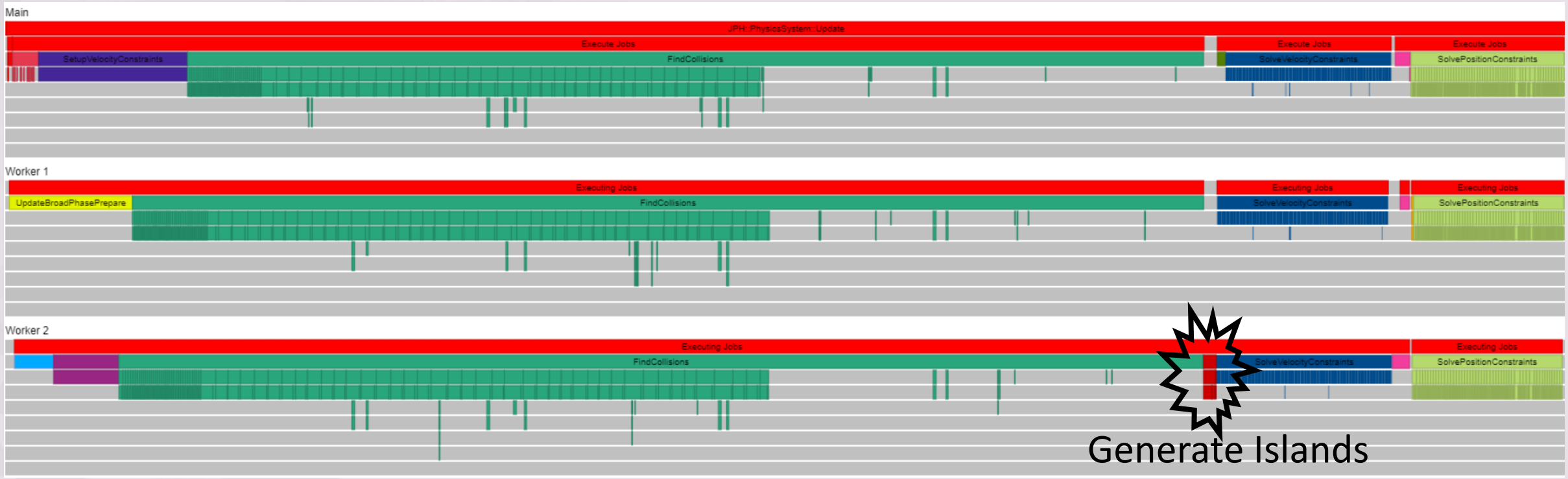


All done!

ISLAND BUILDING - BEFORE

ISLAND BUILDING - RESULT

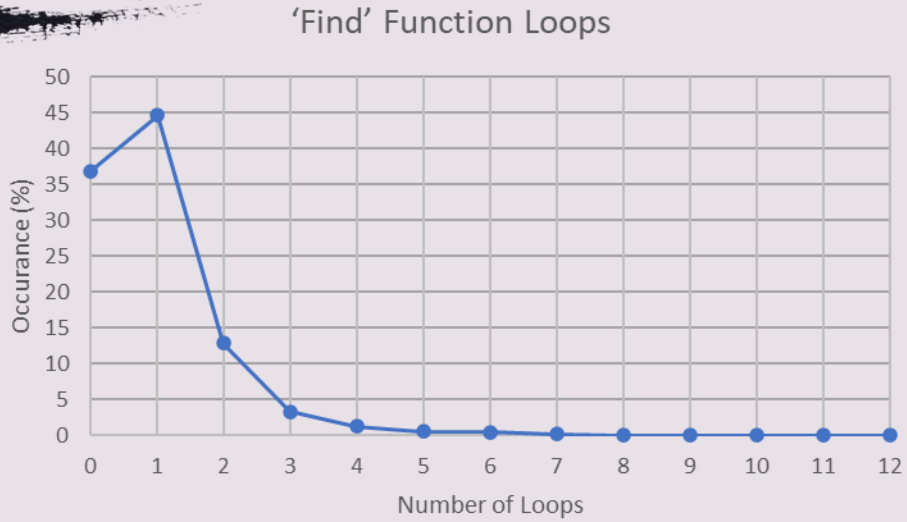
CPUs

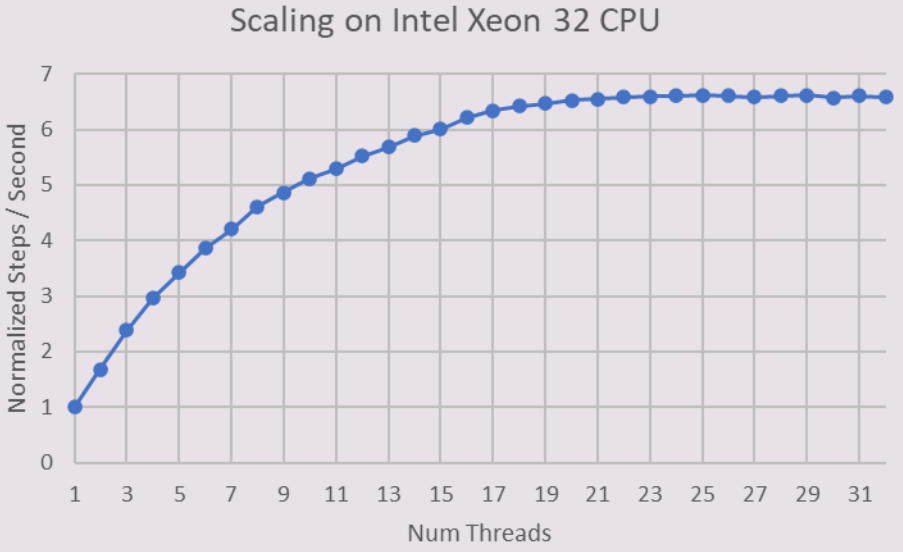


Time ⟶

ISLAND BUILDING - RESULT

- We connect bodies as contacts are detected
- Body unaware of contacts – no locks
- Finishing up O(N)
- Single threaded part 80% less time
- Total 70% less time (4 CPU)
- Find average 0.9 loops
- Union average 0.07 µs
- Not a bottleneck at 16 threads





CONTENTS

CONCLUSION

- Simulation frequency 30 Hz → 2 x 60 Hz in less time
- 60 fps mode halves update time
- Increased visual fidelity
- Reduced heap memory 25%, asset memory 30%, executable size 12%
- Eliminated global lock contention in game update, no more waits!


BEFORE (AVG 3.3 MS, SPIKE 21 MS)
13/13
12
Aloy: It's down!
2

AFTER (AVG 2.1 MS, SPIKE 10 MS)

QUESTIONS?

jorrit@guerrilla-games.com

https://github.com/jrouwe/JoltPhysics (MIT License)

References:

- 'Modeling and Solving Constraints' – Erin Catto – GDC 2009
- Beyond 'Killzone': Creating New AI Systems for 'Horizon Zero Dawn' – Julian Berteling – GDC 2018
- 'Killzone Shadow Fall: Threading the Entity Update on PS4' – Jorrit Rouwé – GDCE 2014