GDC 14

Multi-Actor Behavior Sequencing in The Sims 4

Brian Bell
Sr. Software Engineer, Maxis/EA
bmbell@ea.com

GAME DEVELOPERS CONFERENCE®
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21
2014

Introduction

- The Sims franchise is the best-selling PC game franchise of all time
- We have a really broad, diverse audience
- We work hard to provide a good experience on a really broad range of PC hardware
  - We run on less recent PCs by most standards
  - We pay a lot of attention to laptops w/ integrated graphics

Delivering The Sims

- We must deliver a quality experience on all hardware that we support
- We have to look “like a Sims game”
  - Precise synchronization and registration between Sims/objects
  - Sequence integrity
    - No skipping important steps
    - No harsh interruptions
- The simulation is king
    - Always in charge of what happens and how

Challenges

- Running our deep simulations at consistently interactive frame rates is difficult
  - Inherently spike-prone
  - Huge set of options for each Sim at any given time
- We’re constantly adding new complexity
  - The Sims 4 has taken on some performance-sensitive problems like multi-tasking
  - Every expansion adds even MORE options for the Sims to consider

Implementation Goals

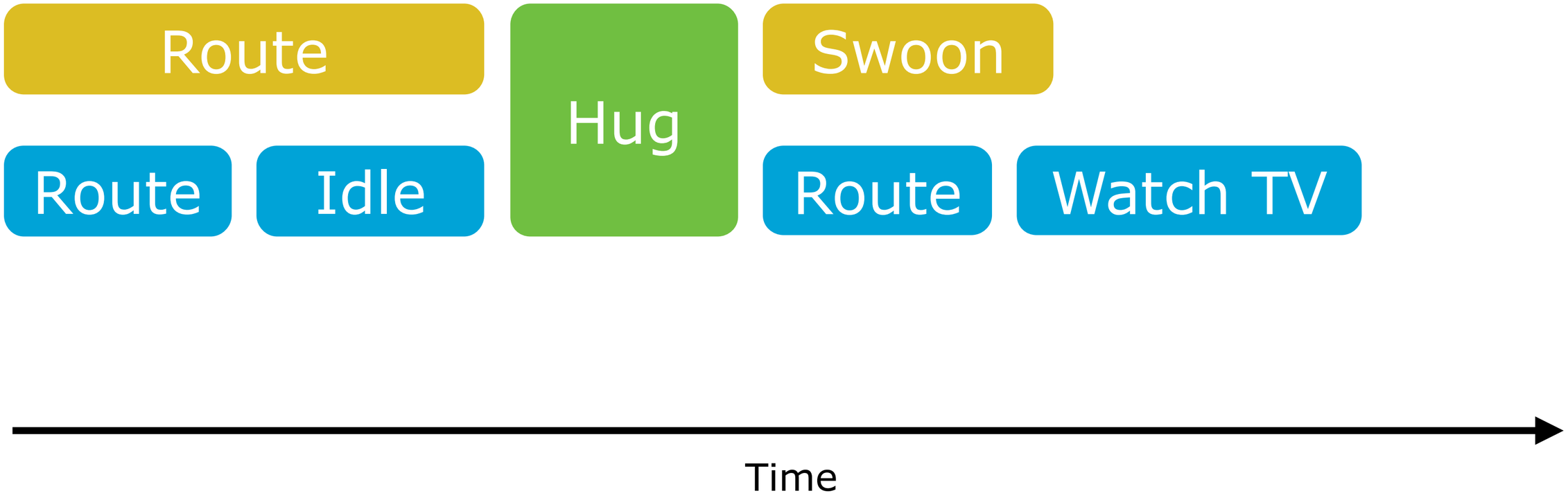
- Decouple the simulation from the renderer entirely
  - Protect the renderer from simulation spikes
  - Protect the simulation from difficult performance requirements
- Keep all impactful decision making in the simulation
  - Path planning, animation selection, etc.
- Move all high-frequency behavior to the renderer
  - Path following, animation playback, etc.

Variable Time Shifting

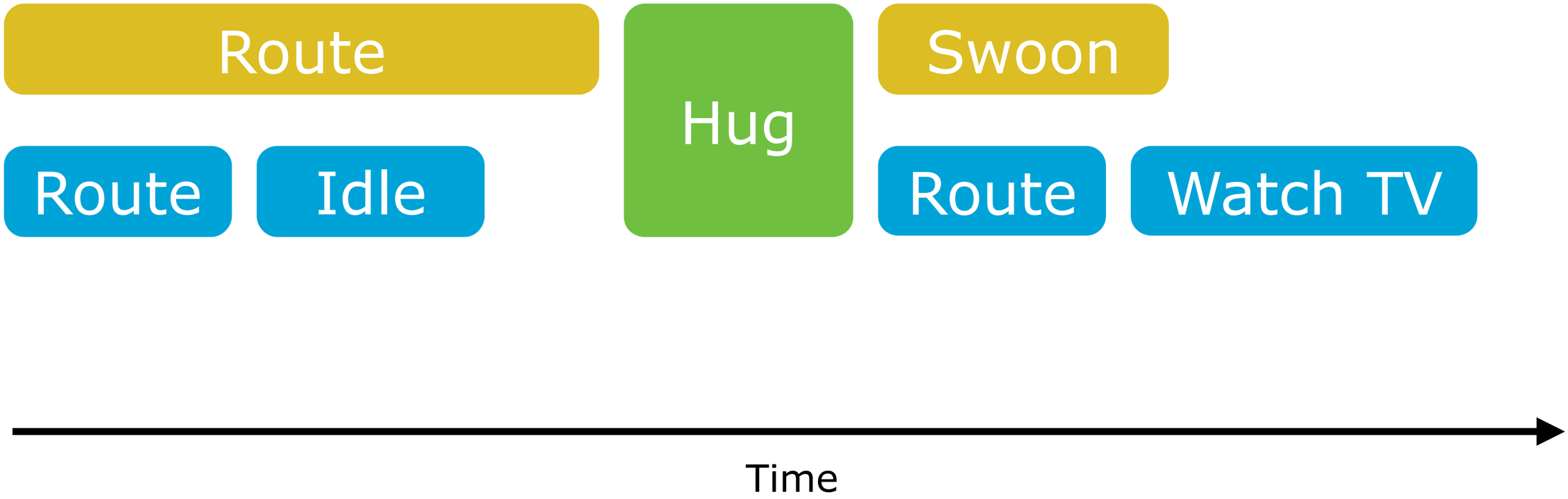
- We allow for variable time shifting on the renderer
  - Perceived delta from the simulation timeline may be different depending on which object(s) you are observing
- Especially important when dealing with transient divergences from the simulation timeline on the renderer, due to…
  - Animation branch restrictions
  - Dynamic avoidance
  - Etc.


ALPHA SOFTWARE
ALPHA SOFTWARE

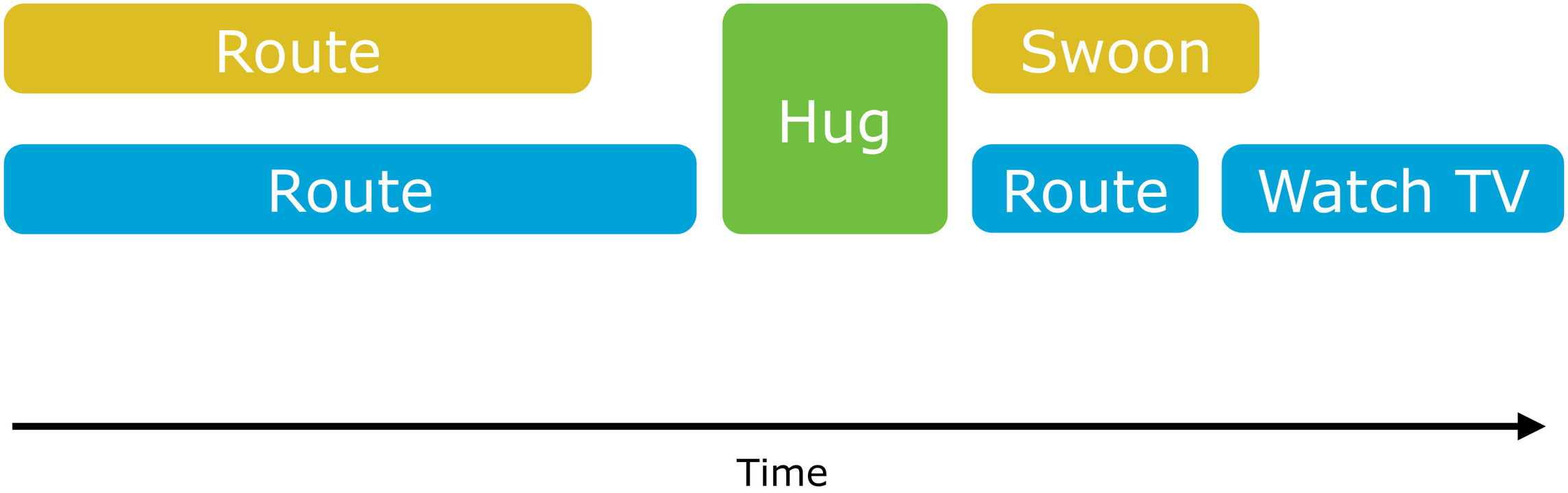
Dependent Timelines



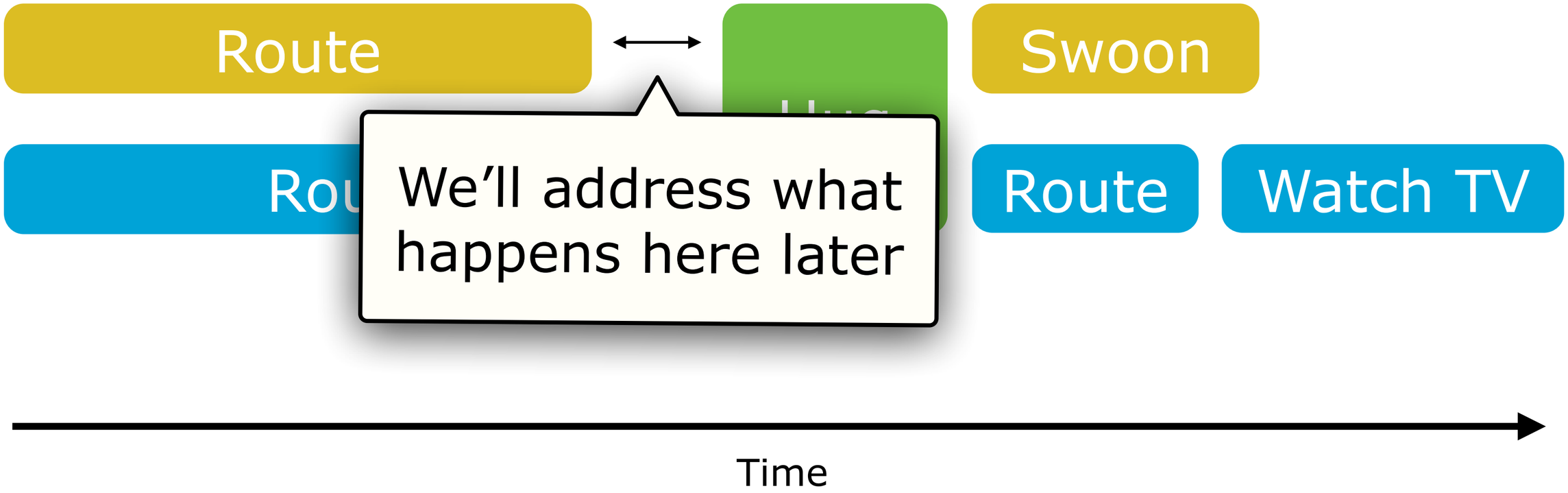
Dependent Timelines



Dependent Timelines

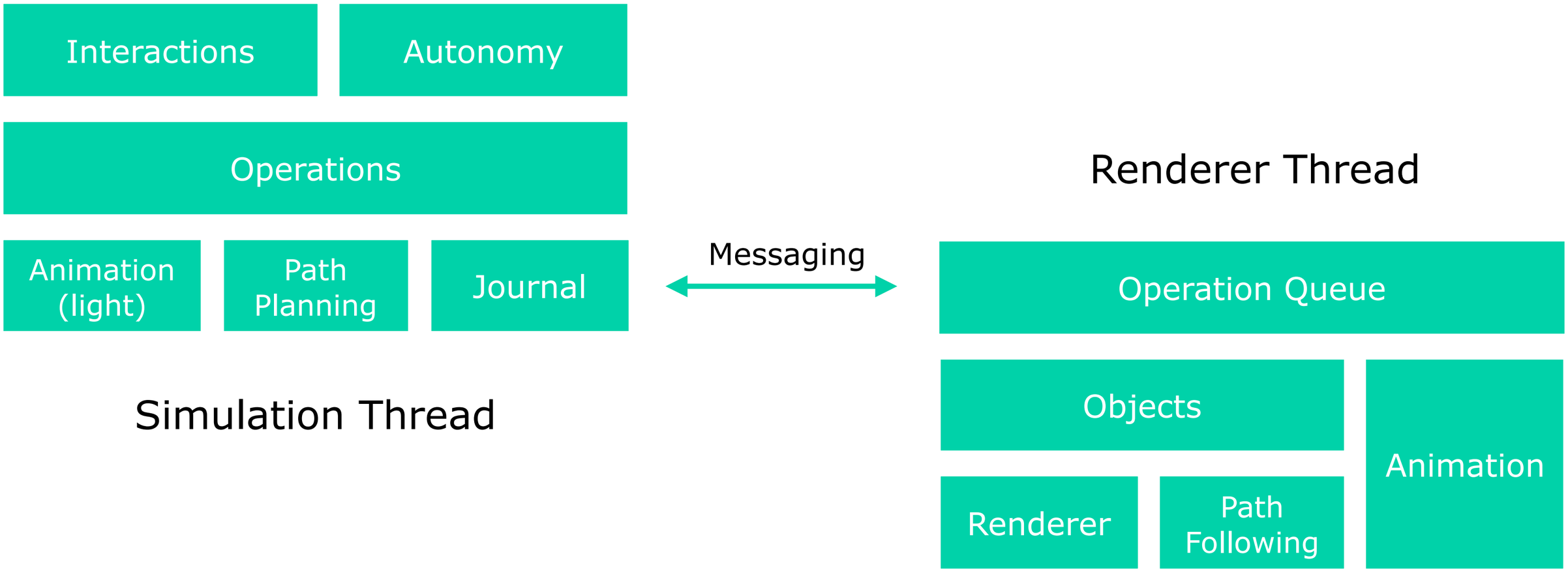


Dependent Timelines

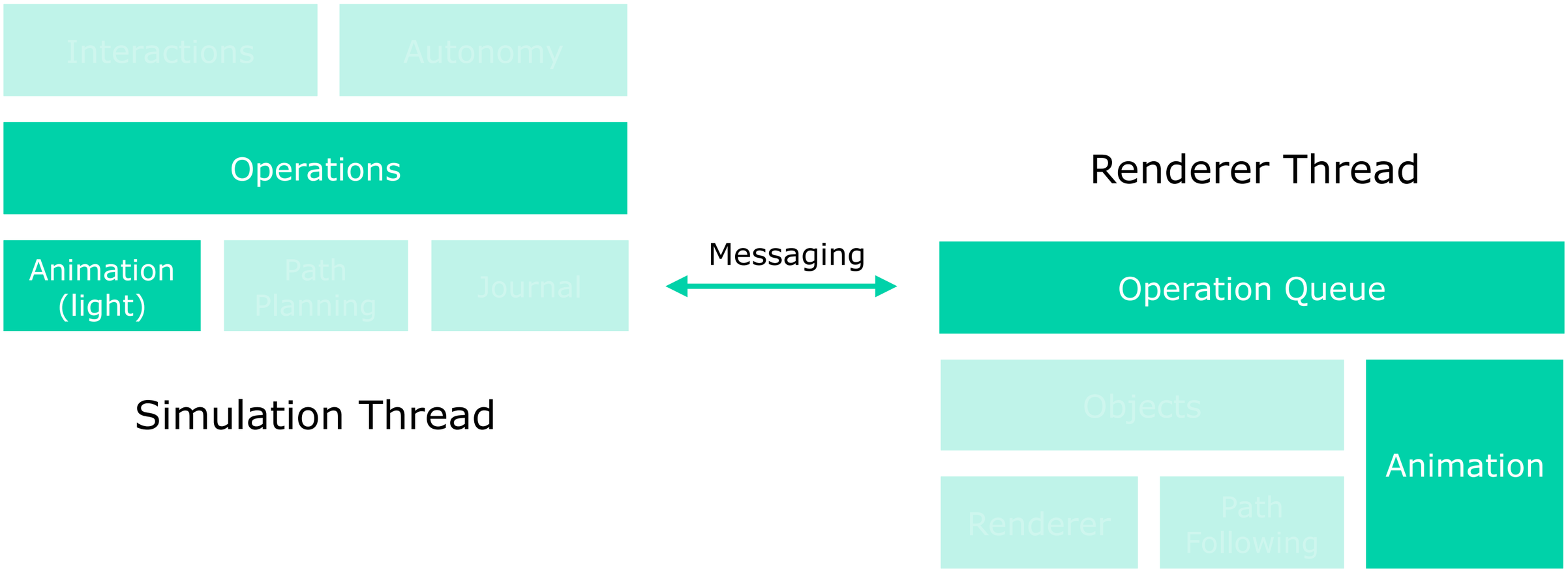


Implementation

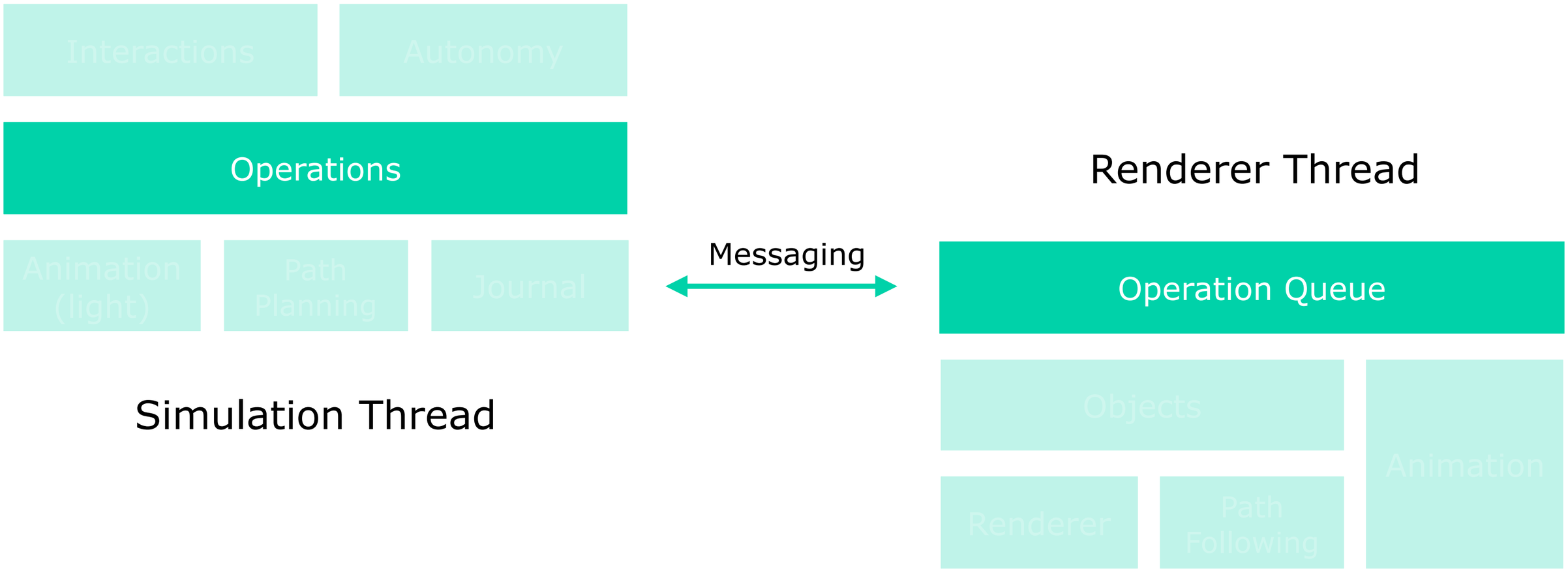
The Sims 4 Simulation Architecture



The Sims 4 Simulation Architecture



The Sims 4 Simulation Architecture



Operations and Channels

- Operations are things the renderer can do…
  - Set a transform
  - Play some animation
  - Follow a path
  - Play a sound
  - Start VFX
  - Trigger some UI
  - Etc.
- An Operation consists of a payload and a set of channels

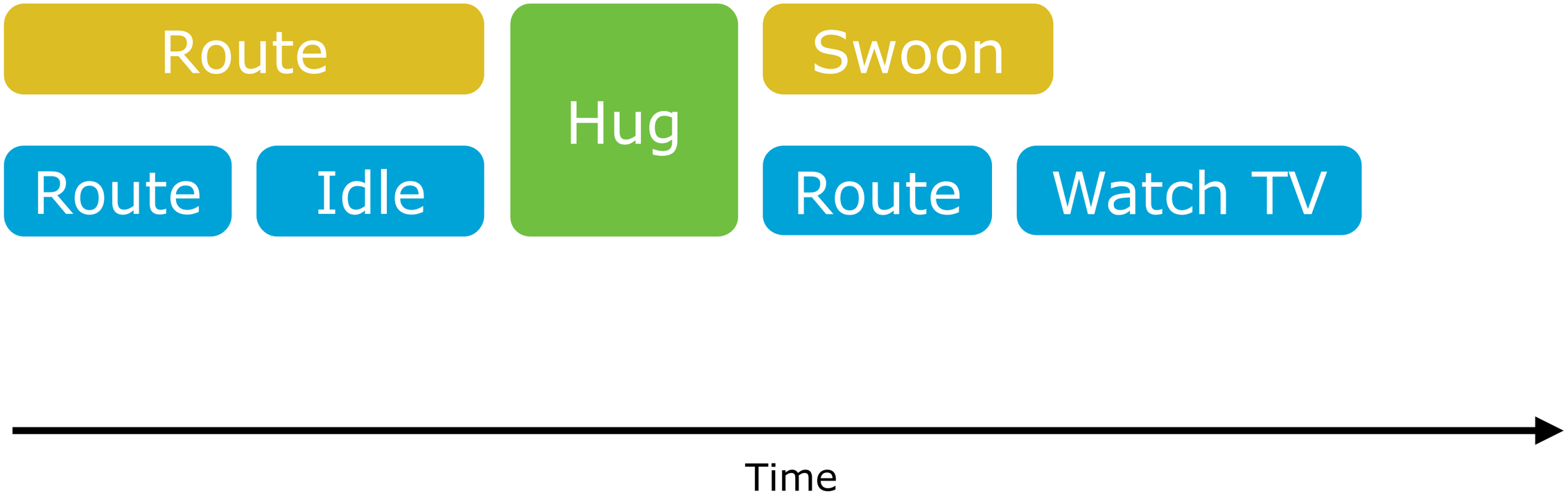
Operations and Channels

- Channels define timelines
  - Each Sim/object in the world is a channel
  - We create other channels too
- A Channel (for us) consists of…
  - An Object ID
  - A 32-bit mask (used for “multi-part” objects)

Operations and Channels

- Channels determine what operations must be executed serially, and which can be run in parallel
- There may be multiple, unrelated chains of serial operations (timelines) at any given time, which can merge and split as necessary
- Operations are inserted into the Operation Queue, which is responsible for maintaining these timelines

Dependent Timelines



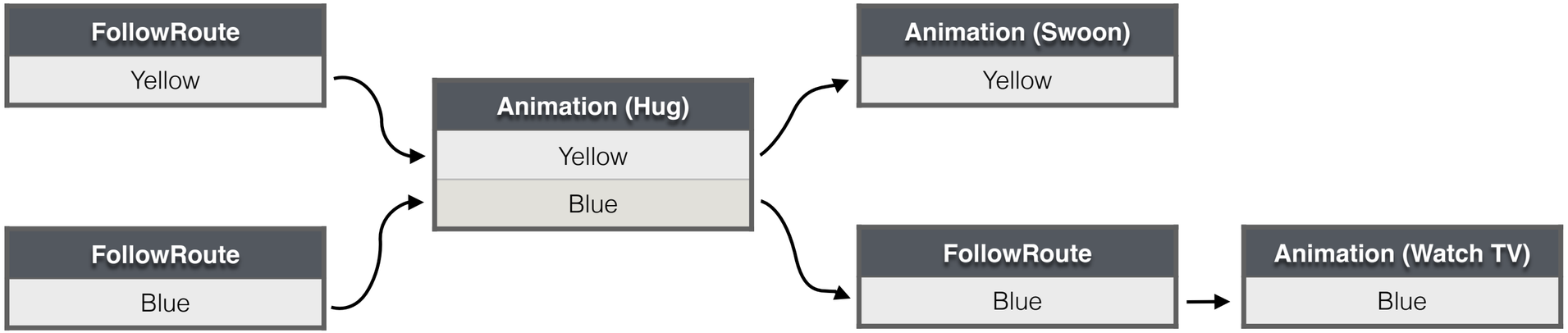
The Operation Queue

- Entirely passive - it is not actively ticked
- Only does work via one of the three public functions
  - Enqueue, AddBarrier, RemoveBarrier
  - All work is performed synchronously
- Enqueue and RemoveBarrier can invoke operation handlers
- When an operation is handled, it is removed from the queue immediately afterward

Enqueue

- On Enqueue, an operation is blocked if any of its channels overlap with the channels of any operation ahead of it in the queue
  - The overlap test is (`a.id == b.id && a.mask & b.mask`)
- If the new operation is blocked, it is pushed onto the back of the queue
- If not, its handler is invoked immediately

Operations and Channels



Operations and Channels

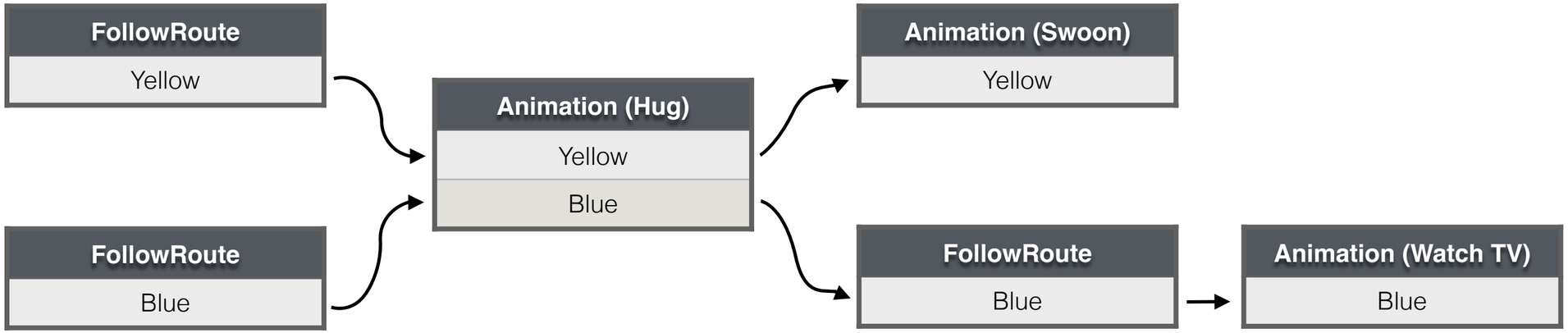
Barriers & Taking Time

- We use barriers as a way to let external systems gate progress through the queue
- Barriers are simply a special type of Operation which are not processed upon becoming unblocked. RemoveBarrier must be called to remove them
- When barriers are removed, we check to see whether any operations located behind the barrier in the queue are now unblocked, and if so, their handlers are run immediately

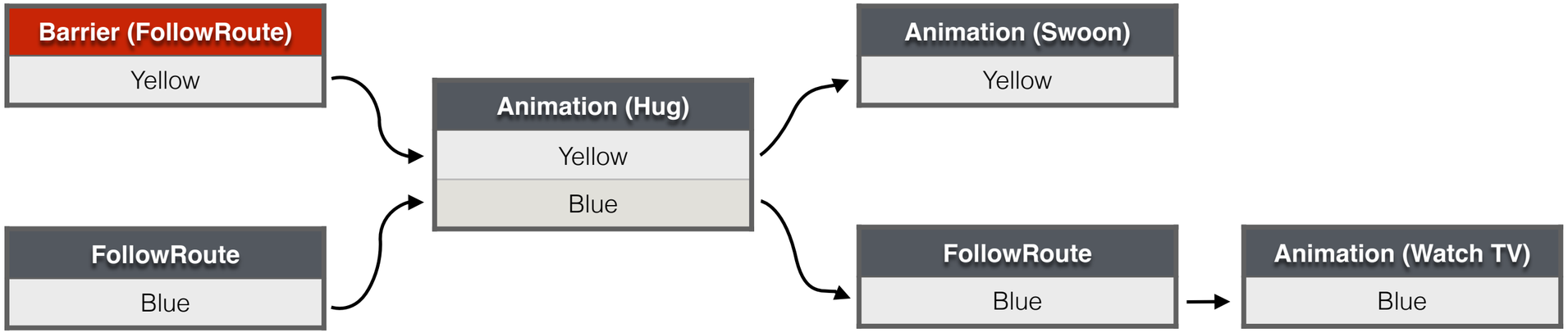
Barriers & Taking Time

- Barriers are added from an operation handler, function, and are added immediately behind the operation being handled
- Barriers are generally removed from separate systems (e.g. animation) when they have reached a logical place to clear them
  - *For animation, we use "eligible for branch out"… More on this later*
  - When we remove a barrier, we re-evaluate all operations behind the barrier in the queue to see if they are unblocked

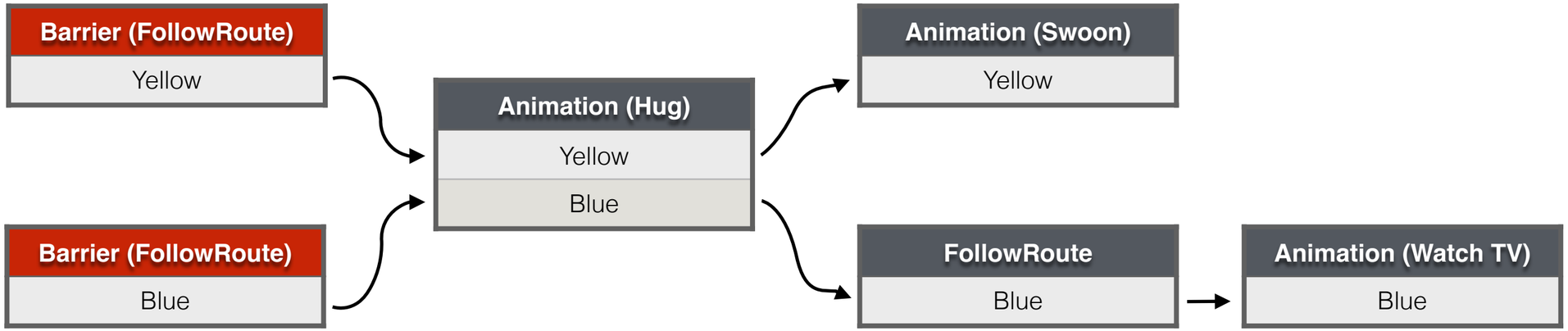
Barriers In Practice



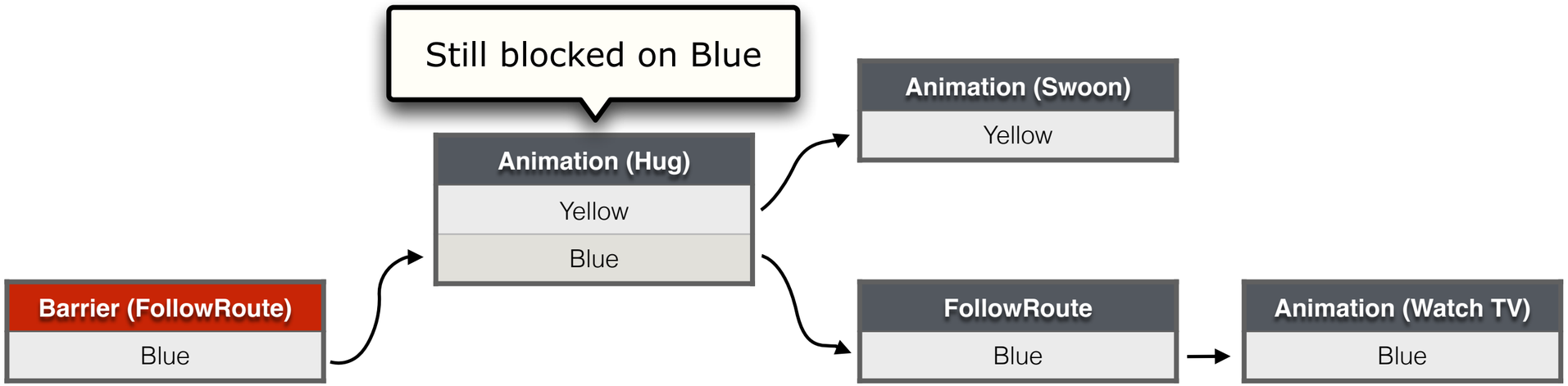
Barriers In Practice



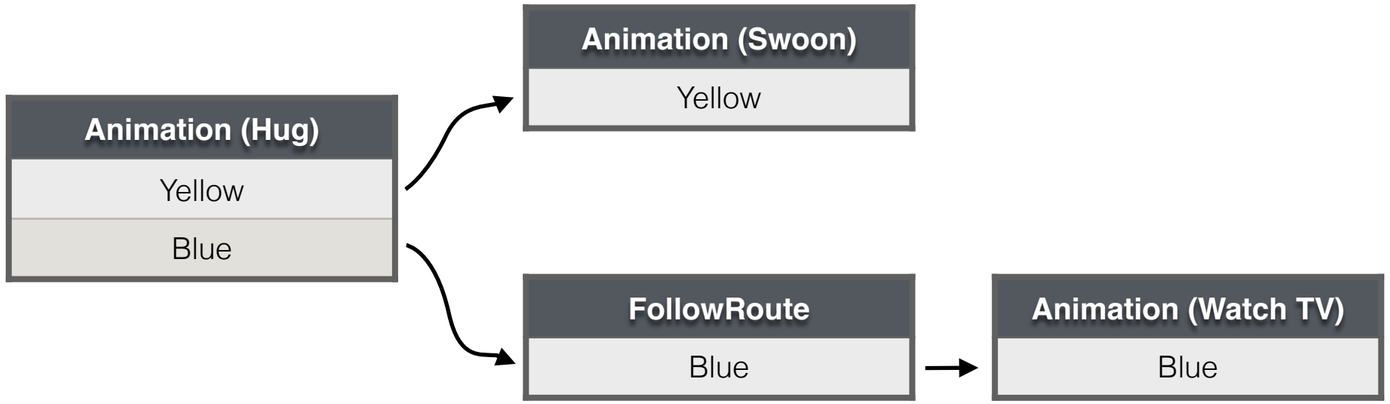
Barriers In Practice



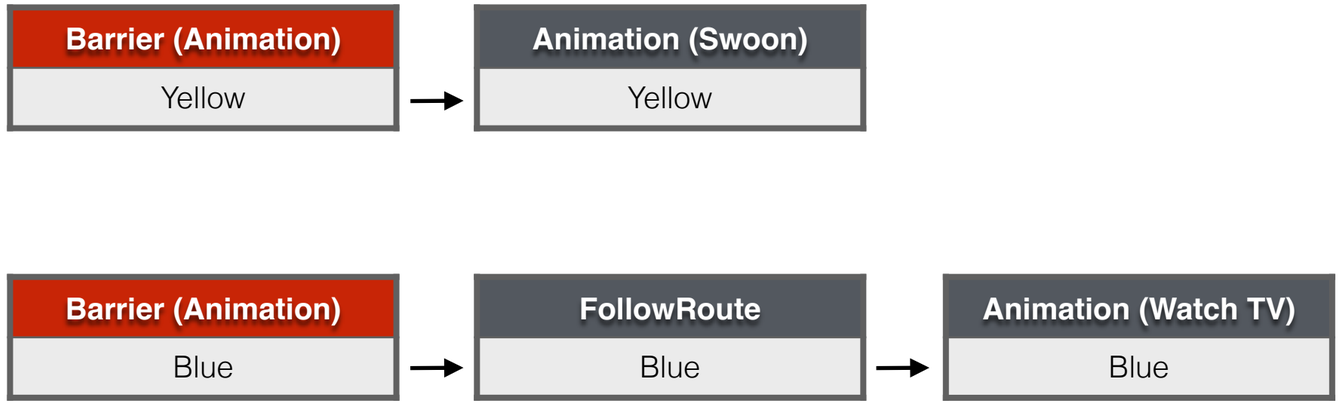
Barriers In Practice



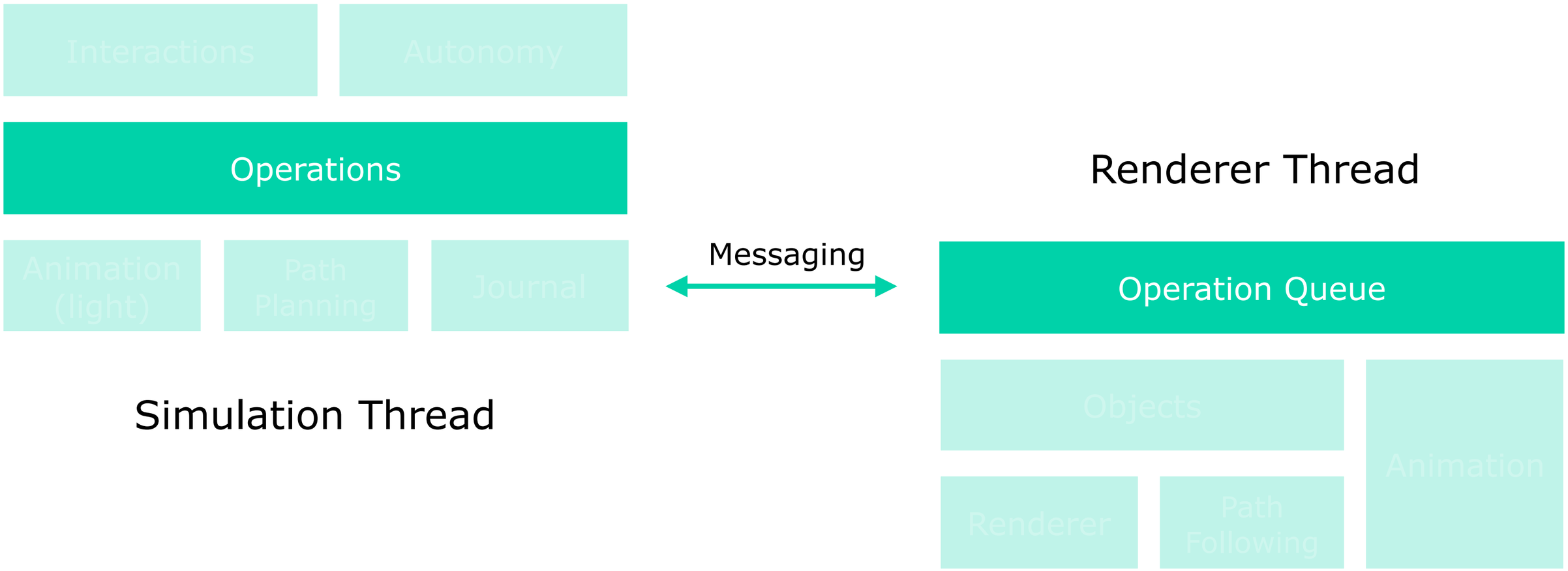
Barriers In Practice



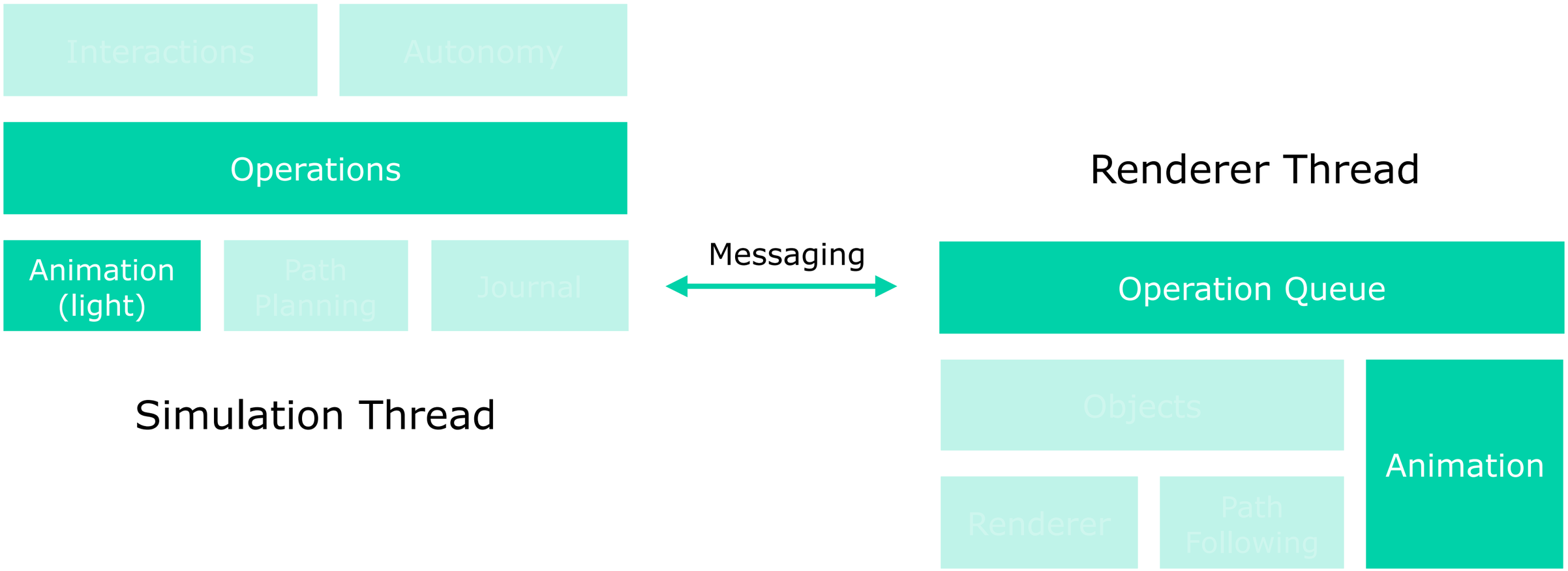
Barriers In Practice



Multi-Actor Animation



Multi-Actor Animation



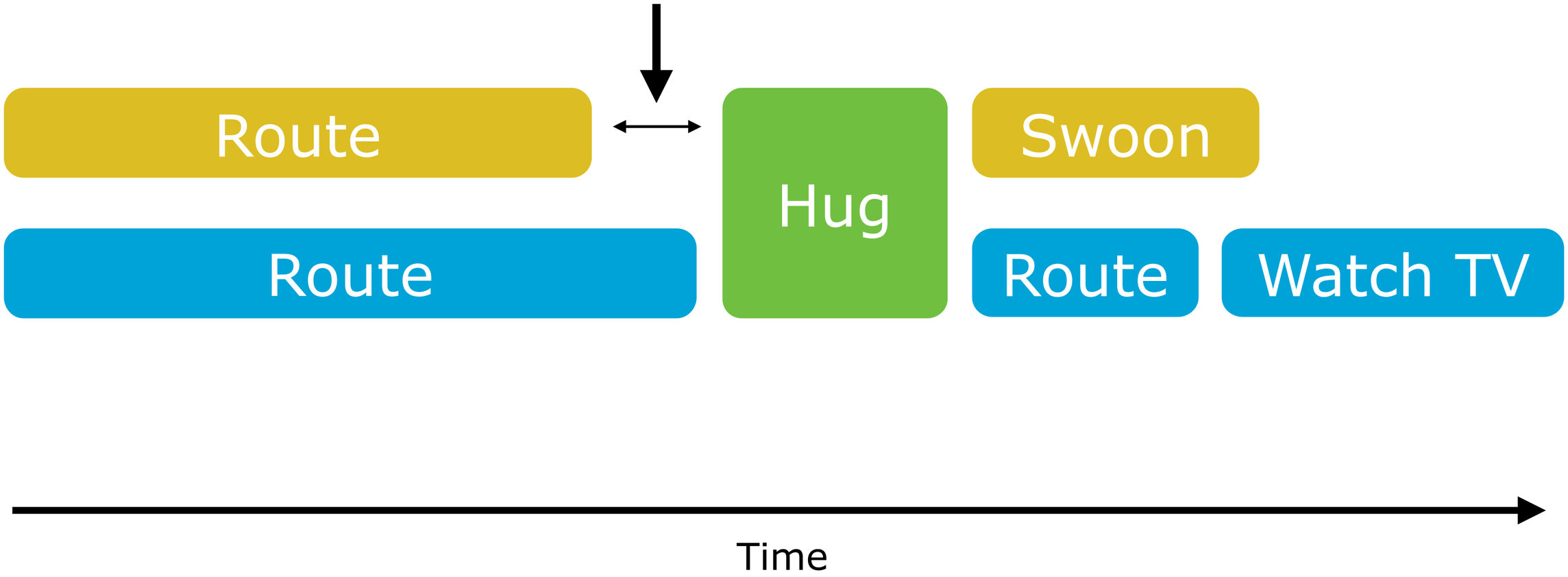
Animation Request Block (ARB)

- We need a generic way to specify multi-actor animation for serialization, including support for frame-accurate synchronization
  - Because we may be time-shifting things on the renderer, simply starting the animations at the same time is not an option
- We need to support sequences with reliable internal animation blending

ARB Structure

- ARBs contain a sequence of controller records, each of which contains…
  - Controller
    - Target Actor
    - Animation Clip
    - IK Information
    - Etc.
  - Blending Info
  - Synchronization Group ID
  - Flags

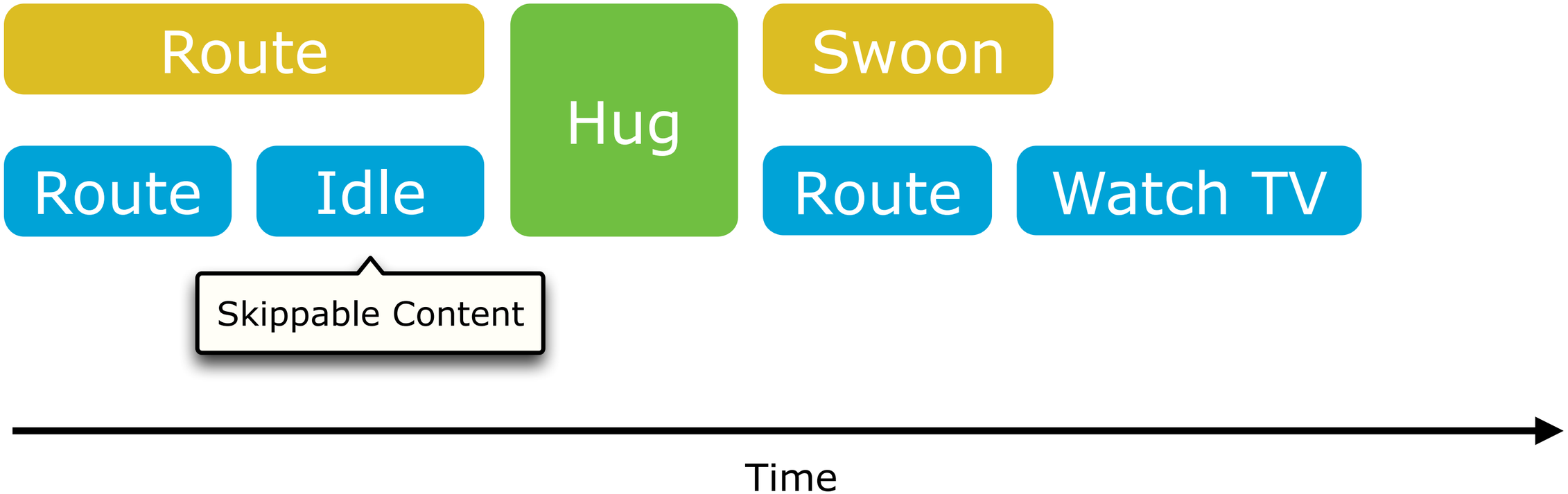
Gap Coverage?



Skippable Content

- Animation scheduled as “skippable” is key to ensuring good gap coverage
- Each ARB generally ends with some interruptible/skippable content
  - Designed to be unobtrusive and easy to blend quickly out of, but just active enough not to look dead
- If we’ve received another ARB affecting the actor in question before the skippable content would start, the skippable content is discarded entirely

Dependent Timelines



ARB Handling (Simulation)

- We do not actually run the animation system on the simulation thread
- We load just enough to support the construction of ARBs
- When the simulation executes an ARB, we compute a rough estimated duration for the ARB which related tasklets then sleep for
  - This duration is generally fairly accurate, and takes into account blend times, but is not perfect

ARB Handling (Renderer)

- When the ARB operation is handled, we…
  - Add barriers to the queue - one for each affected actor/event
  - These barriers are removed when the last non-skippable clip is eligible for blend out
    - For non-interruptible clips, this is (duration - blend out time)
    - For interruptible clips, this is when the clip is started
  - This allows us to unblock each actor individually as soon as possible, allowing for smoother transitions when timelines split

Animation Events

- We make heavy use of animation events to support frame-accurate changes to state, like…
  - Reparenting objects
  - Playing sounds
  - Triggering UI
  - Etc.
- Many of these events require the simulation to make decisions about how to handle them
  - Handling them on the renderer would require replicating a tremendous amount of state and logic

ARB Structure - Event Records

- ARBs also contain a sequence of event records, each of which contains…
  - Event Channel
  - Event Source Record
    - Which of the controller records generates the event
  - Event Type/ID

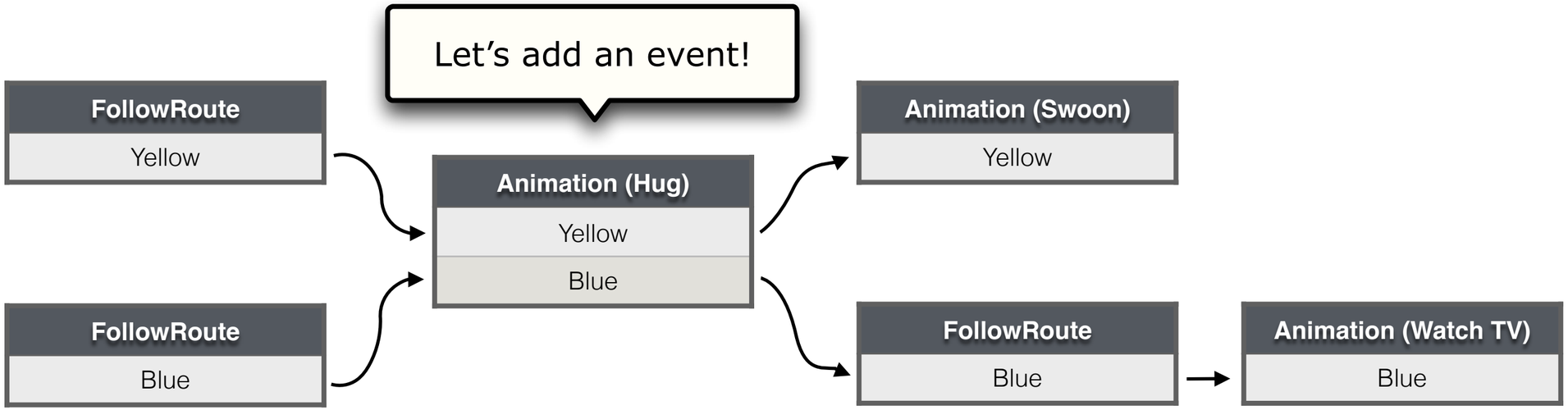
Animation Event Handling (Simulation)

- On the simulation, event handlers are run synchronously when the ARB is started
  - For each event we run handlers for, we allocate a unique channel that doesn’t map to a real object
  - Event handlers can do most supported operations
- Any operations that get enqueued during this time are automatically blocked ONLY on the event channel

Animation Event Handling (Renderer)

- We add barriers for each event channel, along with the per-actor barriers
- We remove these barriers when the event is actually fired in the animation system
  - All event-blocked operations are handled synchronously
- The renderer does handle SOME events natively
  - Simple sound events, one-shot FX, etc.
  - The simulation dictates everything else

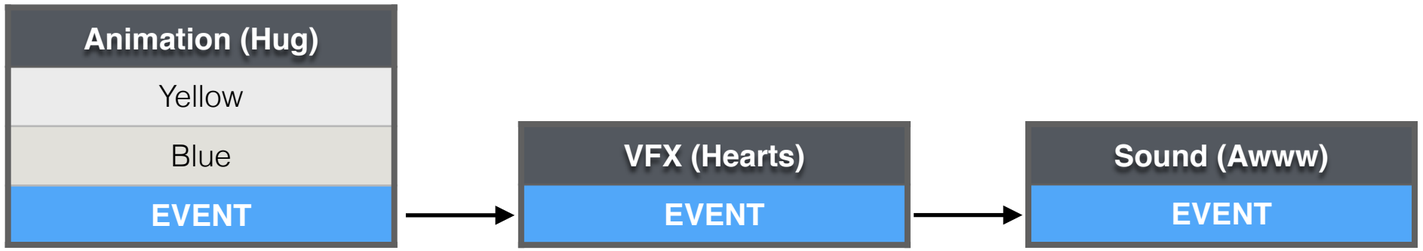
Event Handling



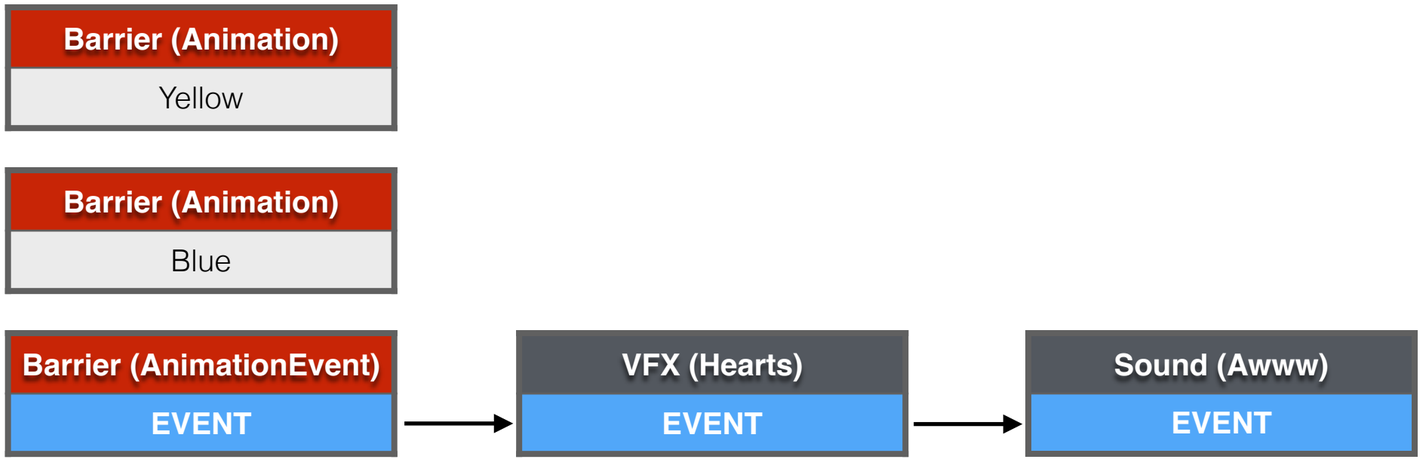
Event Handling

Animation (Hug)
Yellow
Blue

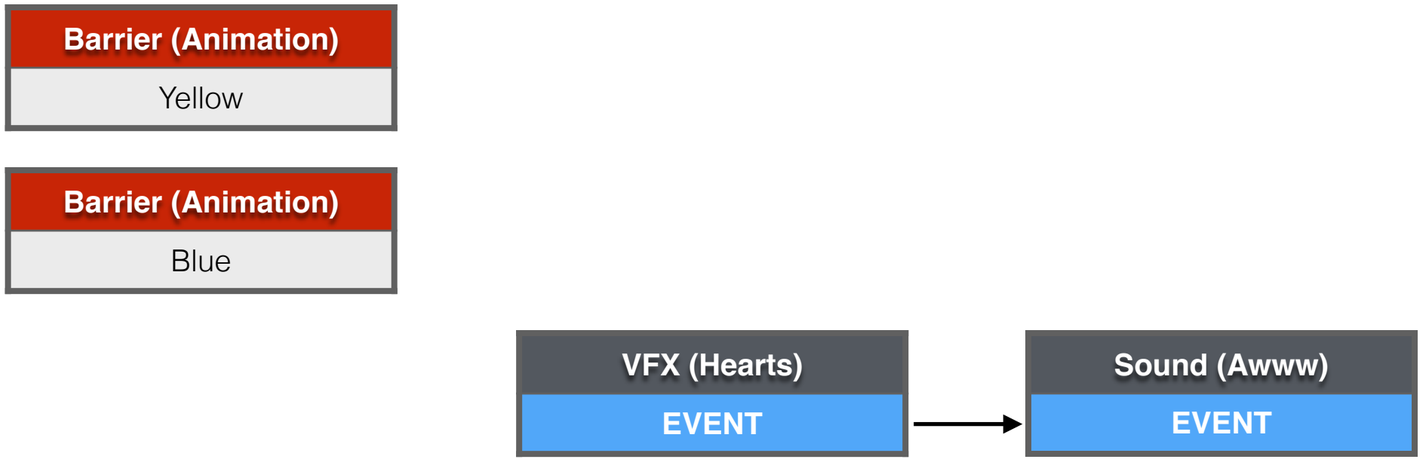
Event Handling



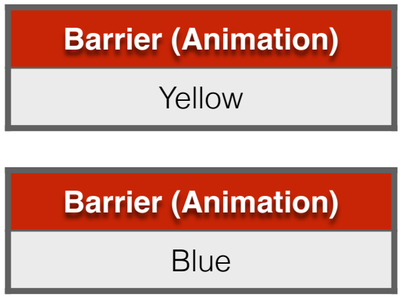
Event Handling



Event Handling



Event Handling



Issues

Nothing's perfect

Complexity

- Decoupling adds a fair amount of complexity to feature implementation
- Requires implementing simulation-side and renderer-side functionality to support a feature
  - This may sound familiar to some of you…
- Careful design of which Operations you support can mitigate the issue
  - The more re-usable/composable the better

Managing Desync

- Transient desync is okay (and expected)
- Excessive, persistent desync is not
- For ops that take time/create barriers...
  - Timestamp
  - Scale
  - Interrupt/Skip
    - Work with your content partners to come up with visually appealing “filler”

Managing Desync

- Many of our interactions involve interruptible looping content that runs on the simulation for a tunable amount of time, which is time we can get back when necessary
- We do the vast majority of our “catch up” during route following, where we have a variety of good looking options for recovering the drift
- In hard to manage cases, the renderer can talk back to the simulator to limit desync
  - Effectively implements a “re-sync” on certain operations

Debugging

- The most common trap that catches people when working with this system is inadvertently creating operations whose blocking channel sets do not match the actual behavior in the simulator
- This manifests as…
  - Operations happening too early, or “out of order”
  - MASSIVE desynchronization due to operations being serialized on the renderer, but not on the simulator

Debugging

- Having good debugging tools is a big help here
- Our toolset consists of…
  - A timestamped simulation side log of all operations and their channels sent to the renderer
  - A cheat command which outputs the current contents of the queue (all pending operations/barriers and their associated channels)
  - Between these two, we can pretty easily reconcile discrepancies and problems without having to touch the debugger

Other Applications

Dynamic Cutscene Sequencing

- Allow for in-gameplay cutscenes
- No need to interrupt in-progress behaviors
- No need to warp actors at synchronization points
- Could run in parallel to regular gameplay

Latency-Tolerant Online

- Our approach could be used to build online games where sequences are more important than sync
  - Probably excludes MOST competitive games
- Can scale down to very tight synchronization, so long as operations know how to scale/sync themselves
  - Relatively straightforward, with some visual degradation

Summary

Summary

- Allowing for variable time shifting has given us the flexibility to decouple completely, while retaining the “look” of The Sims
  - Sequences are preserved
  - Individual steps are shortened or skipped, but only when we’ve specifically allowed it

Summary

- Decoupling our simulation and renderer has allowed both systems to deliver a higher quality experience
  - Rendering frame rates are more predictable
  - The simulation is allowed to do the work it needs to, when it needs to

Summary

- Having a robust multi-actor sequencer that supports dynamic behaviors is broadly applicable
- Applicable to any scenario where…
  - Sequentiality is important
  - Multiple actors are involved and are sometimes synchronized
  - Simulation runs separately from rendering

Q&A

bmbell@ea.com