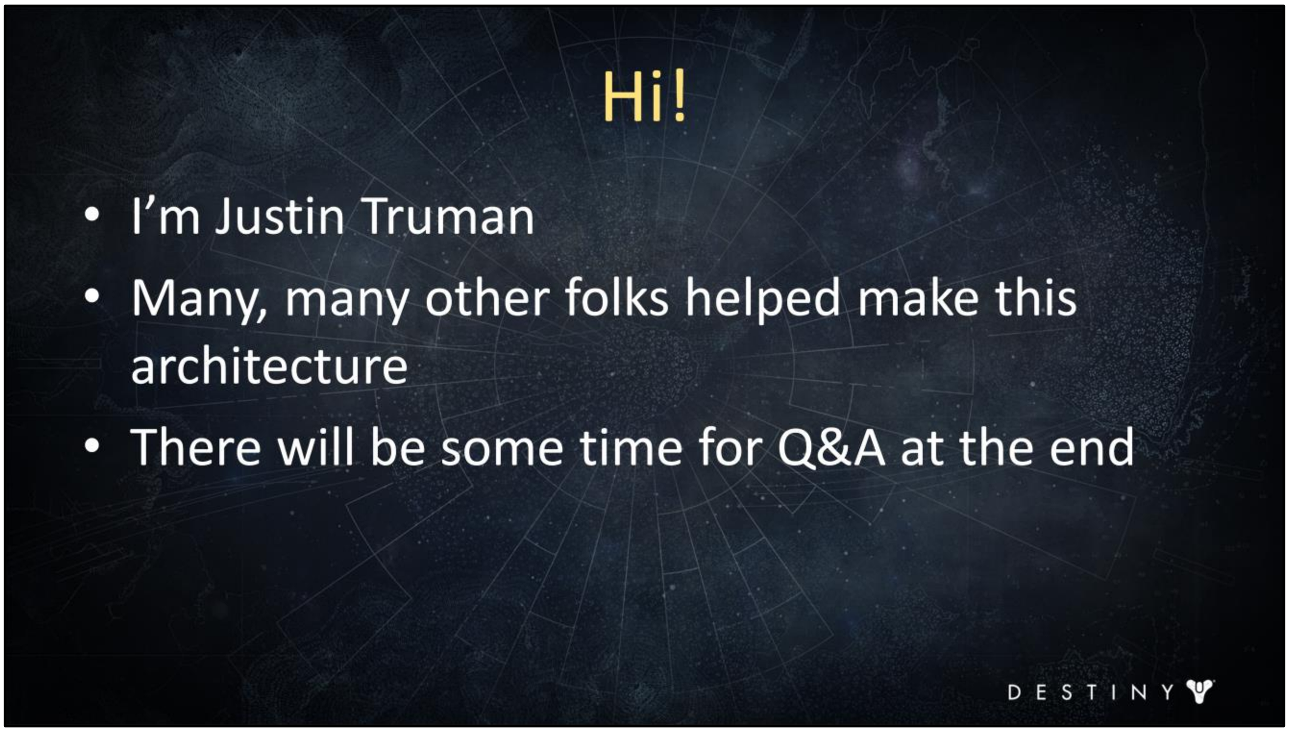
Shared World Shooter

Destiny's Networked Mission Architecture

BUNGIE

DESTINY

Alright, let's get started.



Hi!

- I’m Justin Truman
- Many, many other folks helped make this architecture
- There will be some time for Q&A at the end

I’m Justin Truman, I’m one of the Engineering Leads at Bungie.

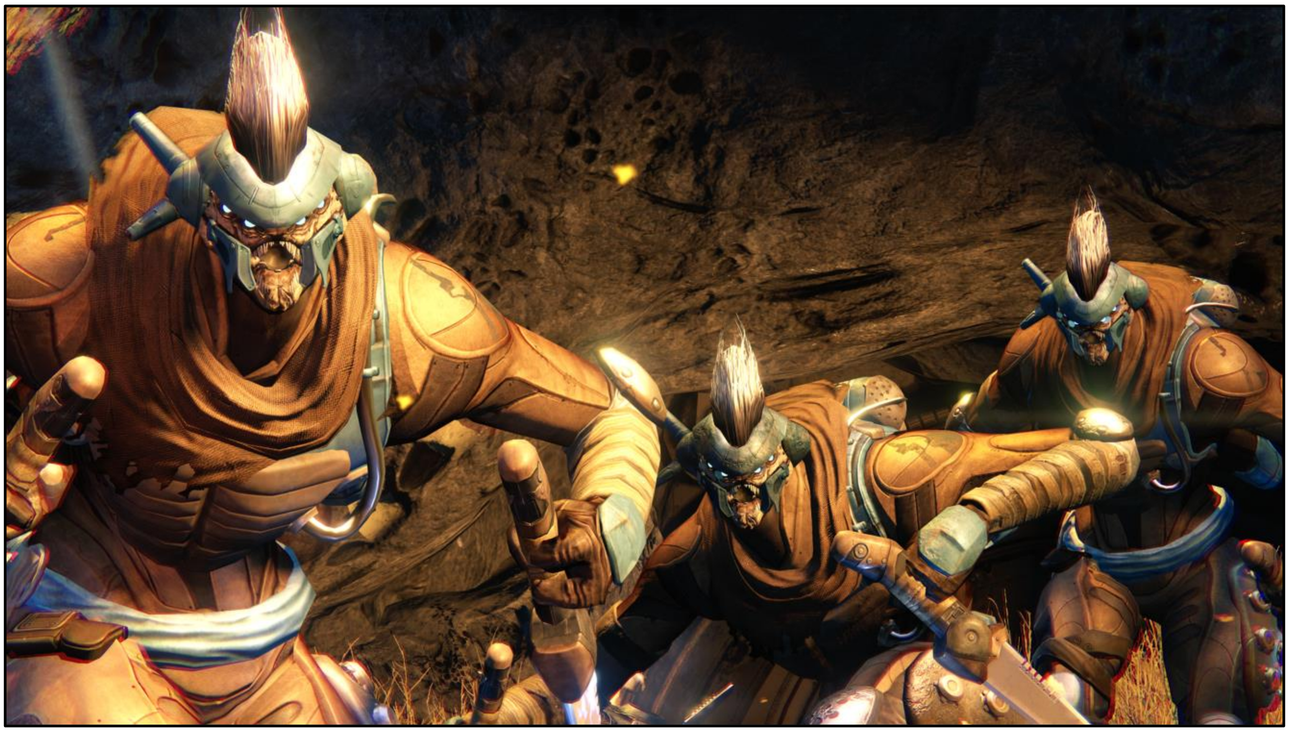
And before I even start, I want to make sure that I make clear that I’m not even close to the sole contributor of the architecture that we’ll be talking about today. There was a good 50 man years that went into our Activity Networking systems for Destiny.

It was a very large group effort, and that’s why I’ll be saying “we” a lot during this talk.

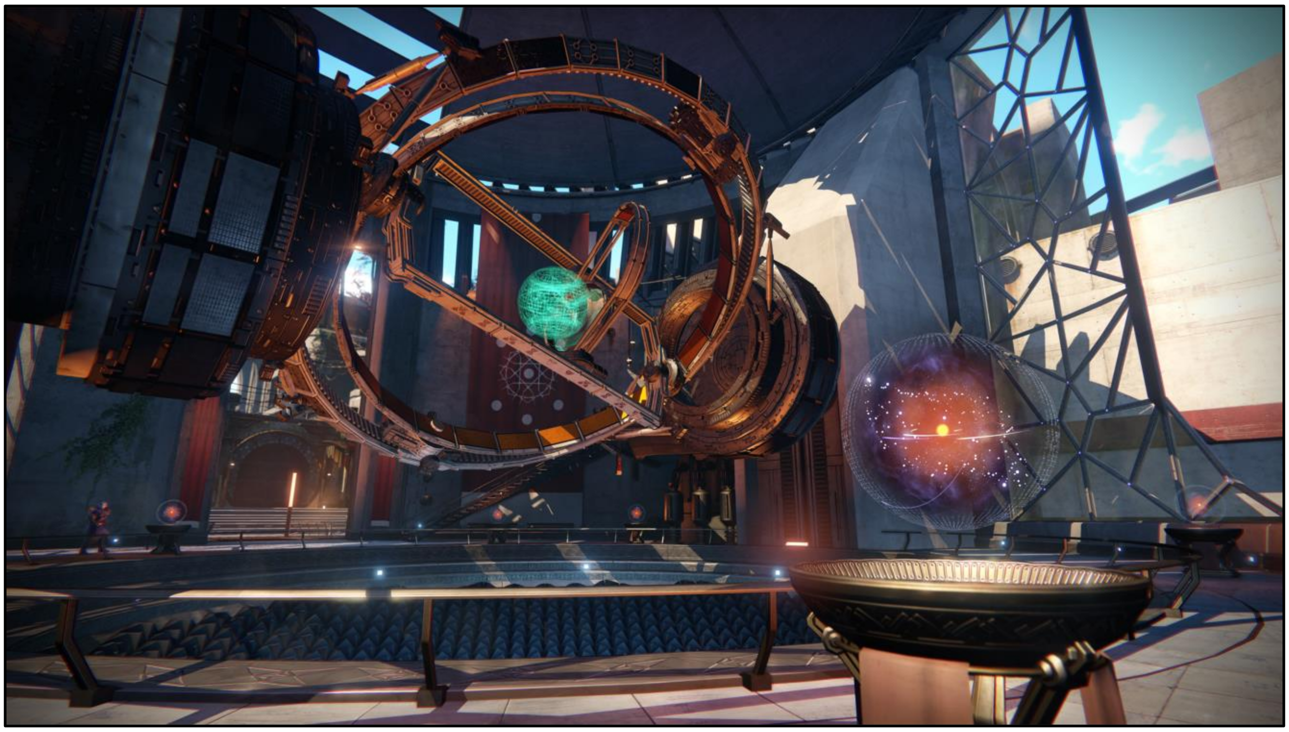
Also, I really want to answer any questions you guys have at the end, assuming I can stay on-time.

Now’s a great time to silence your cell phones, too!

So, Destiny was a lot of firsts for Bungie.

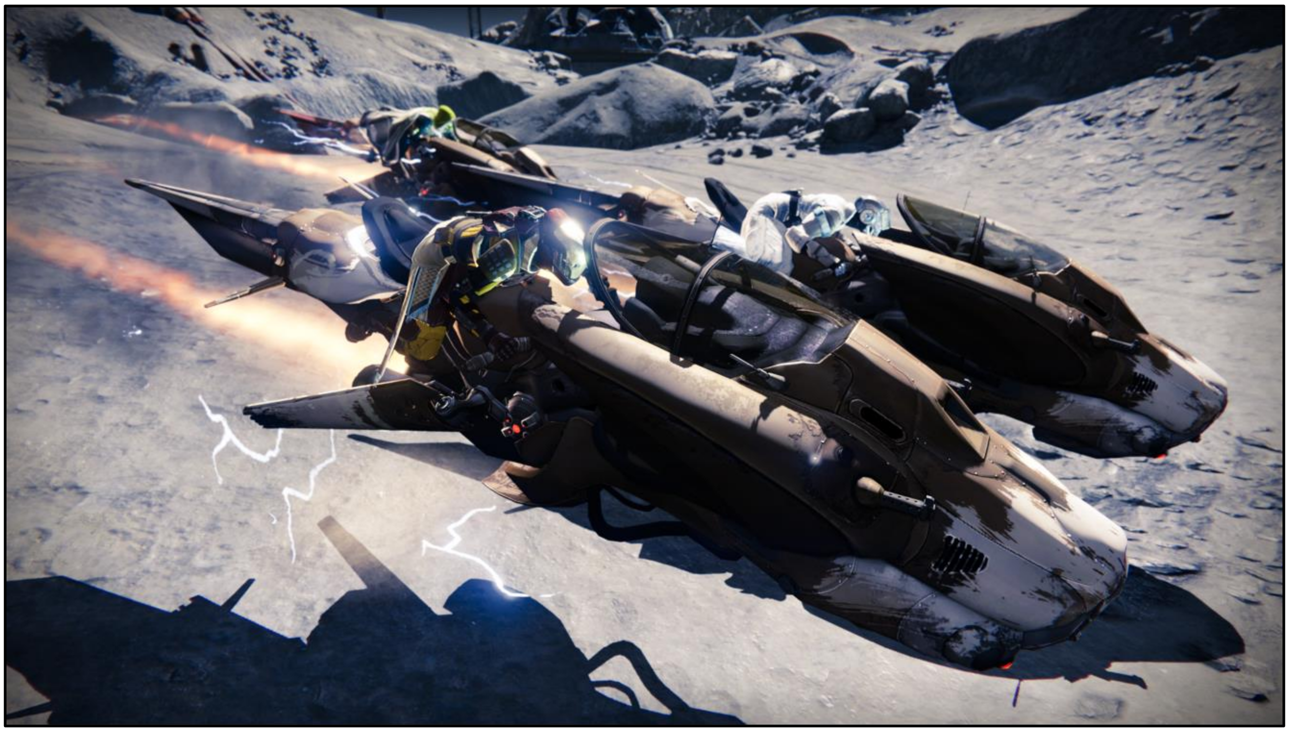


It was our first 4-platform game



It was our first always-online game.

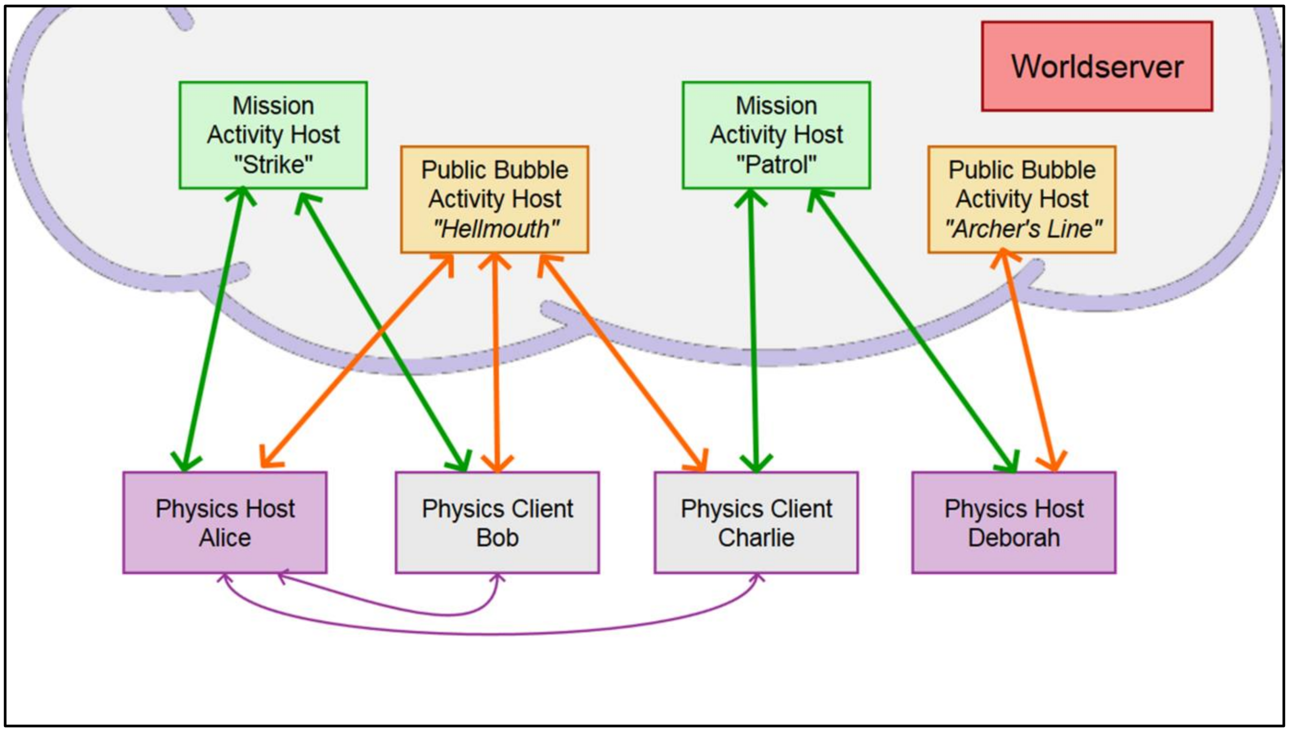
It was pretty scary getting in a room in 2010 (before the Xbox One had even been announced), and planning a console shooter that requires an always-online internet connection.



It was also our first time we tried seamless background matchmaking.

Halo had matchmaking lobbies and menus – we wanted to get rid of all of that, and just let strangers wander into your game, automatically and organically. We wanted to get rid of the “Single-Player” vs. “multi-player” menu options, and just have “Play Destiny”.

To accomplish that goal we built a unique …


Worldserver
Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Mission Activity Host "Patrol"
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah

… and uniquely complicated networking topology

Over the next hour, I'm going to talk through the decisions that led us to this unique architecture, and both the advantages and disadvantages that have resulted from our choices.

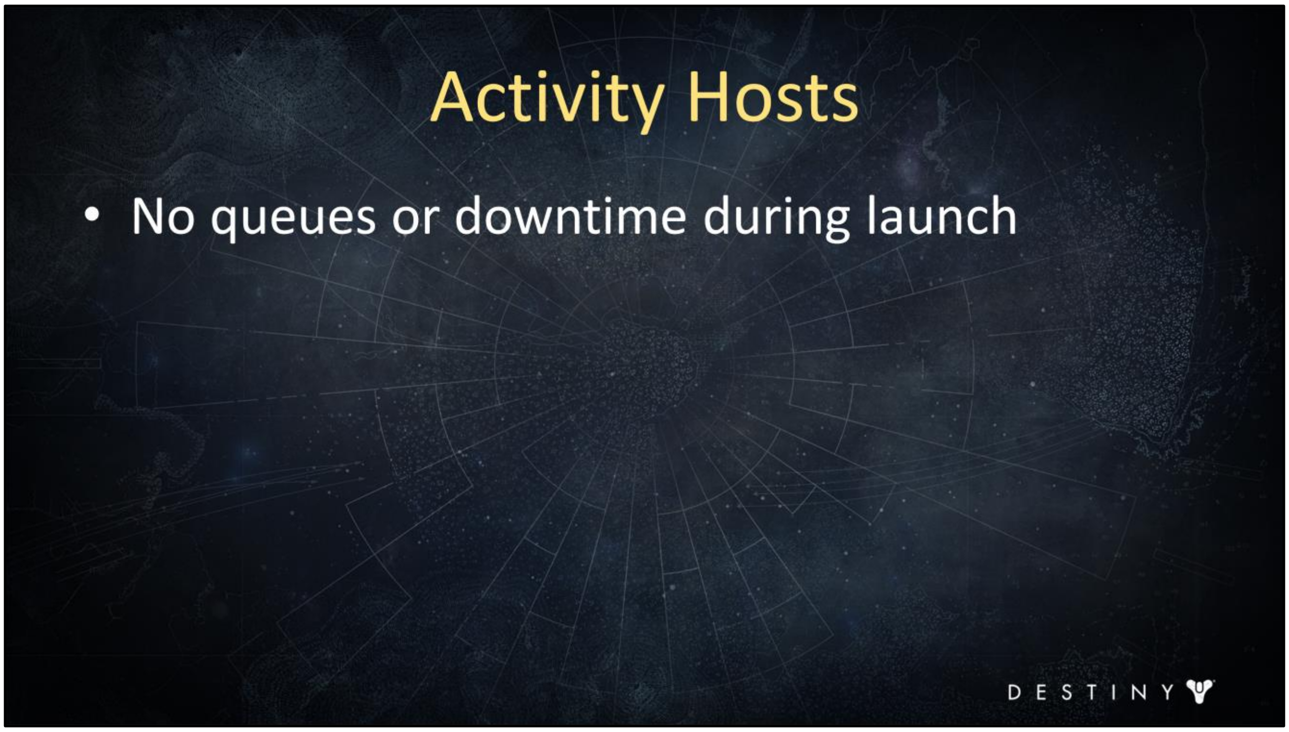
Instead of just using a Peer-to-peer architecture like our previous Halo games, or implementing traditional dedicated servers, we built a hybrid approach that we call "Activity Hosts."

Activity Hosts

DESTINY

Activity Hosts are cloud-hosted machines that run a stripped down simulation of just our Mission Script logic.

With our “Activity Hosts”, which I’ll be describing in detail, we were able to successfully scale at launch,

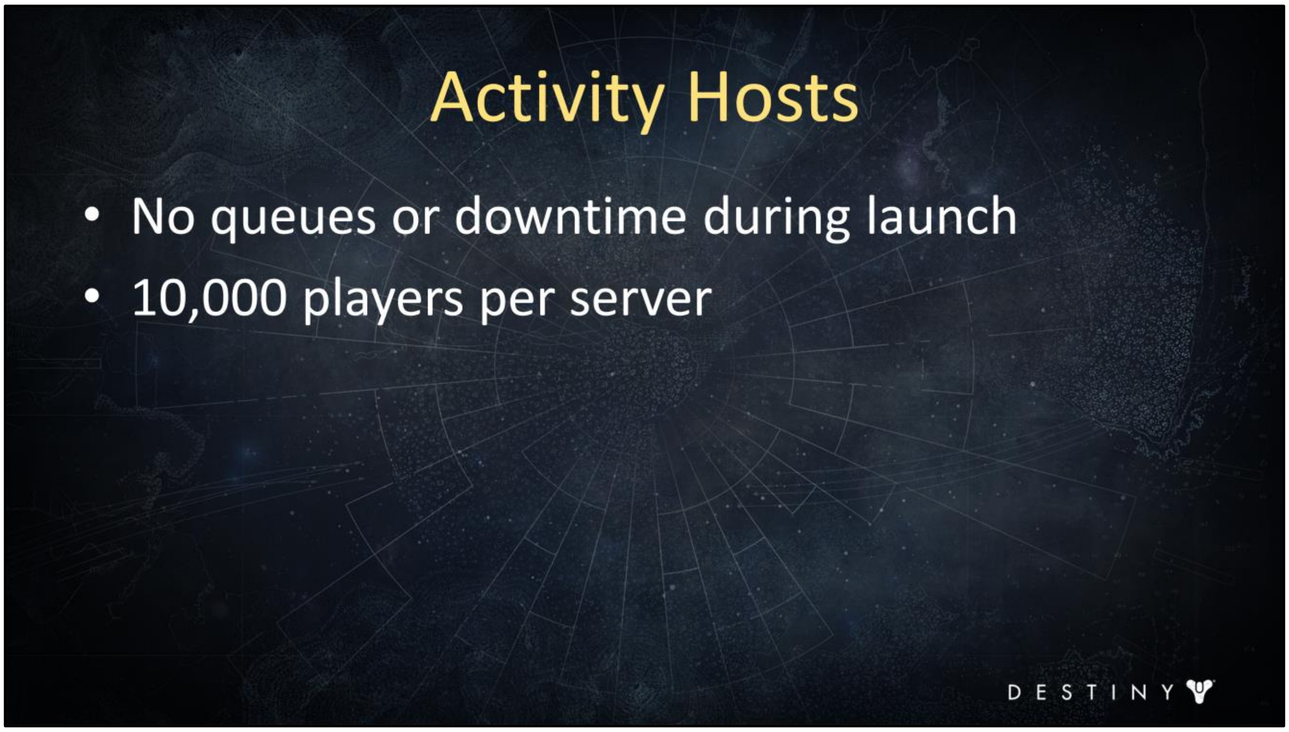


Activity Hosts

- No queues or downtime during launch

DESTINY

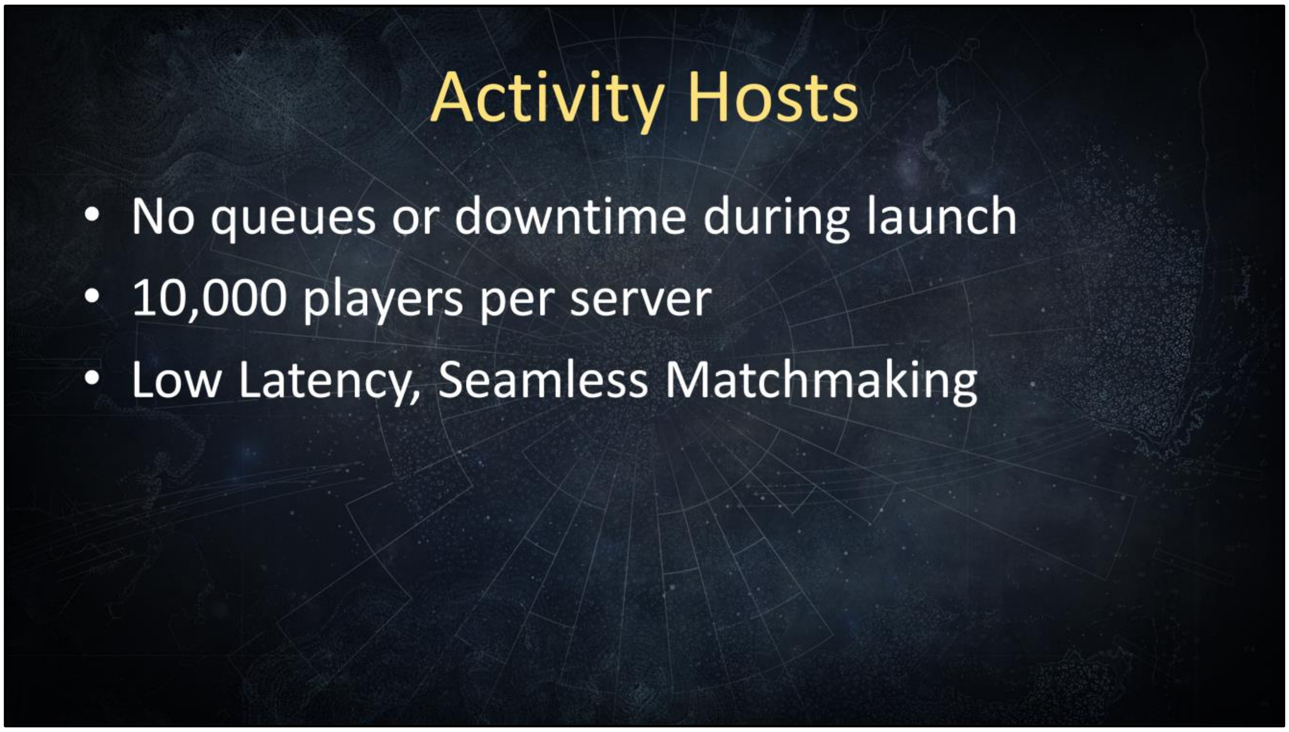
without any queues or downtime



Activity Hosts

- No queues or downtime during launch
- 10,000 players per server

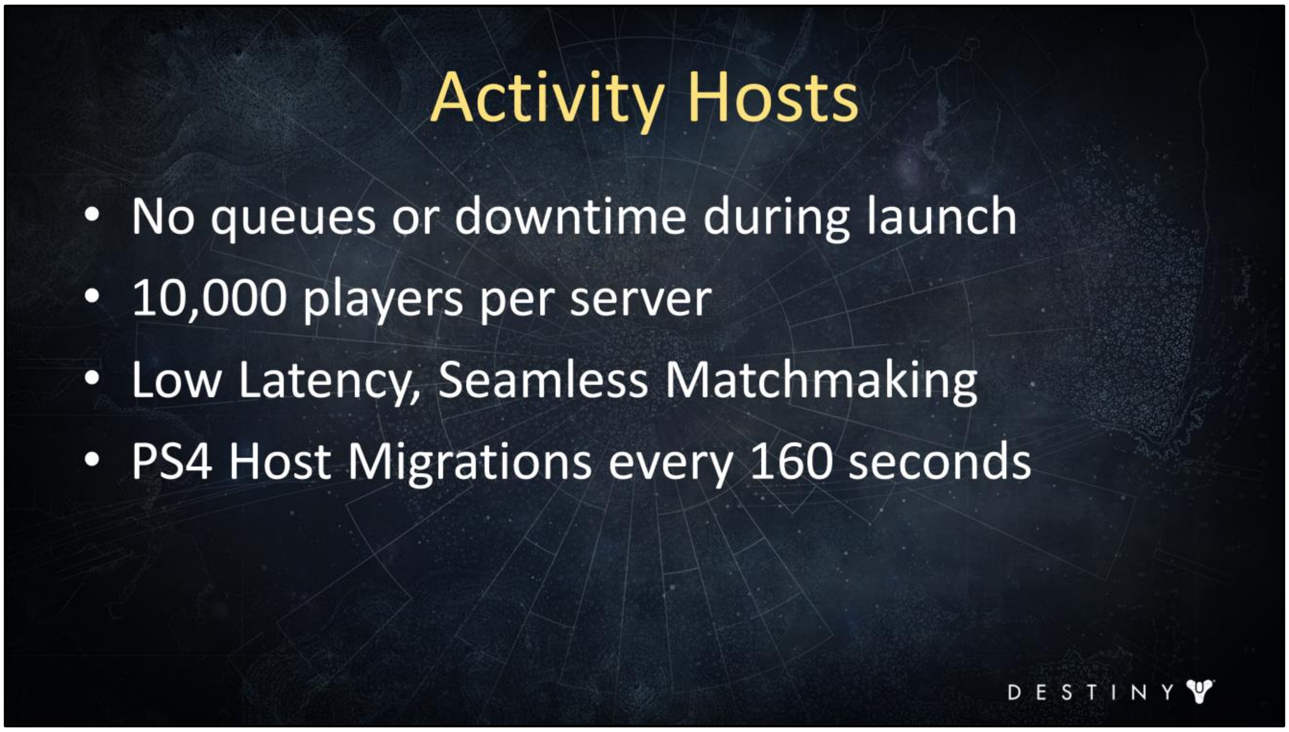
And we were able to support that load with just a few hundred servers in our datacenter, because we can handle loads of 10,000 players per server.



Activity Hosts

- No queues or downtime during launch
- 10,000 players per server
- Low Latency, Seamless Matchmaking

By combining our Activity Hosts with traditional Peer-to-peer networking, we get the low latency action gameplay of a Call of Duty or Halo, while constantly seamlessly matchmaking you to new strangers.



Activity Hosts

- No queues or downtime during launch
- 10,000 players per server
- Low Latency, Seamless Matchmaking
- PS4 Host Migrations every 160 seconds

A typical Destiny player is Host Migrating between different PS4's once every 160 seconds, without noticing any discontinuity in their simulation.

So, that's the player experience we ended up with today, but when I joined Bungie 5 years ago to work on Destiny, we didn't have any of that.

Design Pillars

All we really had to start with was some key Design Pillars, that informed all the early architectural planning.

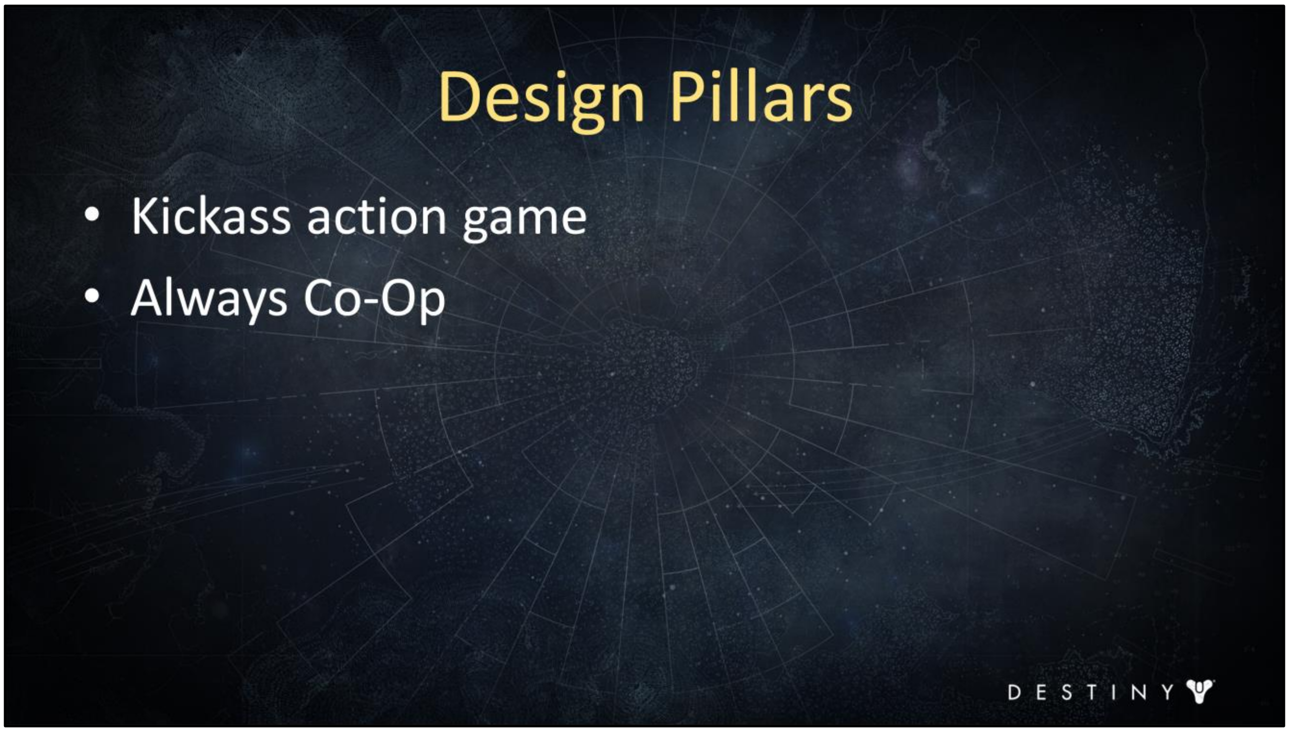
These design pillars were:

Design Pillars

- Kickass action game

DESTINY

Making a Kickass Action Game

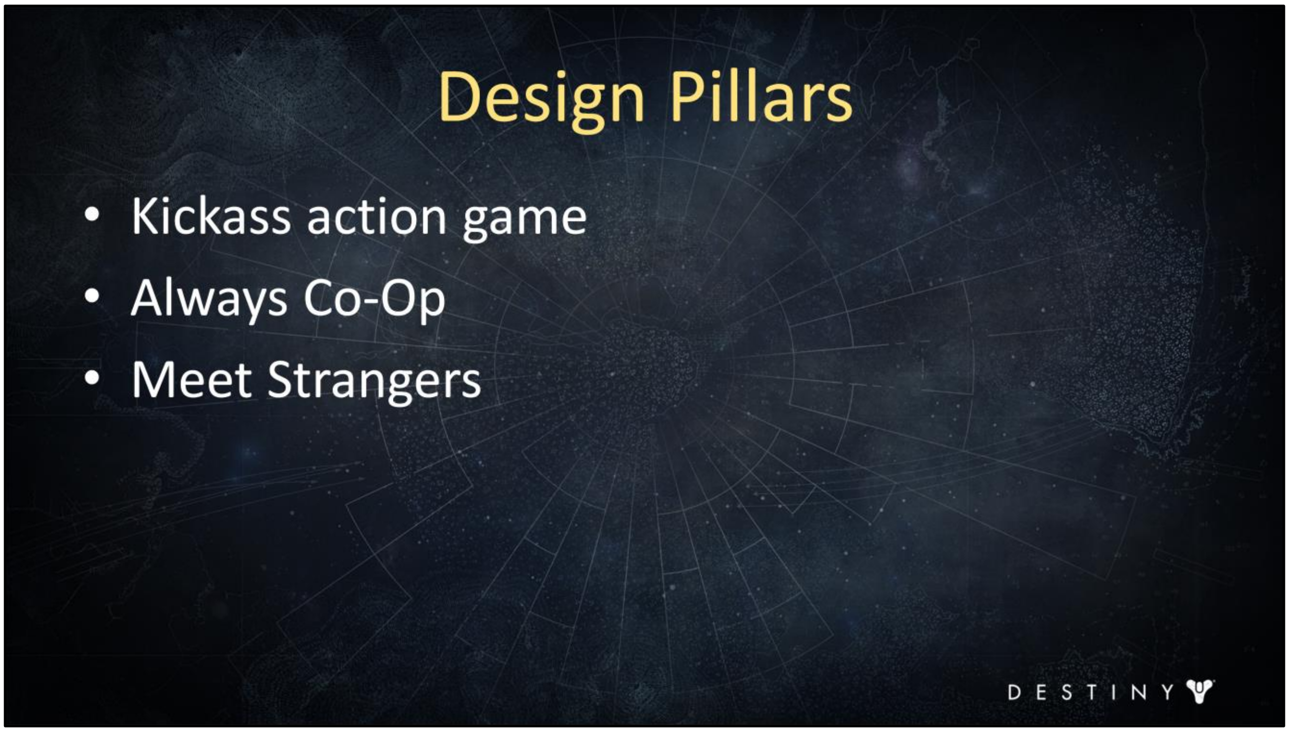


Design Pillars

- Kickass action game
- Always Co-Op

DESTINY

Making sure it always supported Co-Op

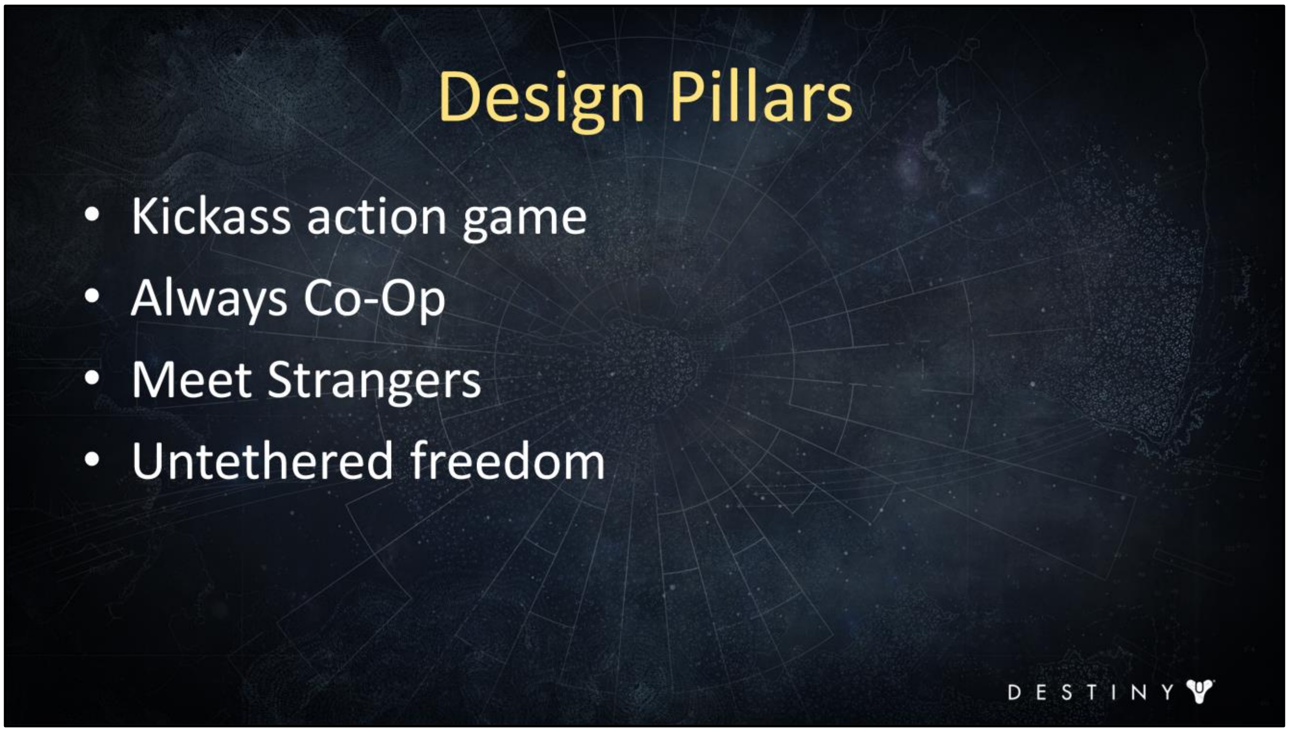


Design Pillars

- Kickass action game
- Always Co-Op
- Meet Strangers

DESTINY

Allowing you to Meet Strangers



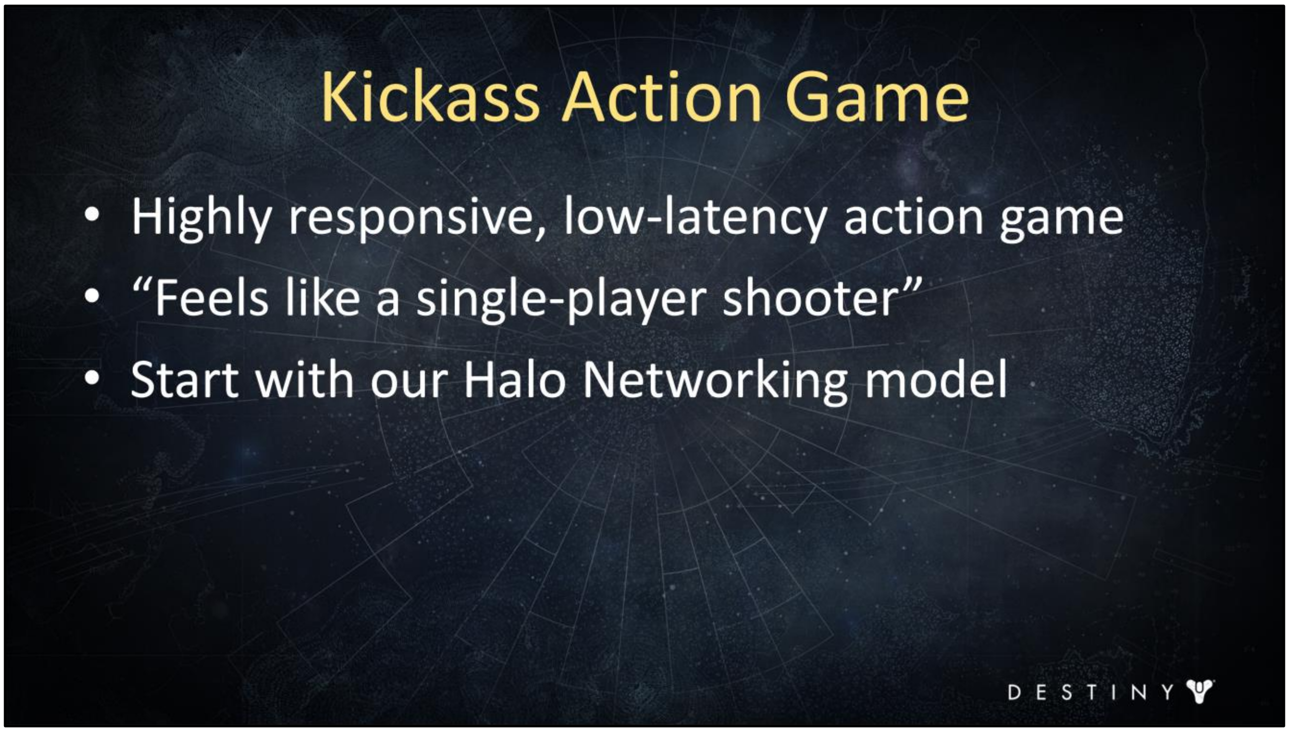
Design Pillars

- Kickass action game
- Always Co-Op
- Meet Strangers
- Untethered freedom

DESTINY

And Untethered Freedom to Explore

Before talking about the tech, it’s worth diving into each one of these in a bit of detail



Kickass Action Game

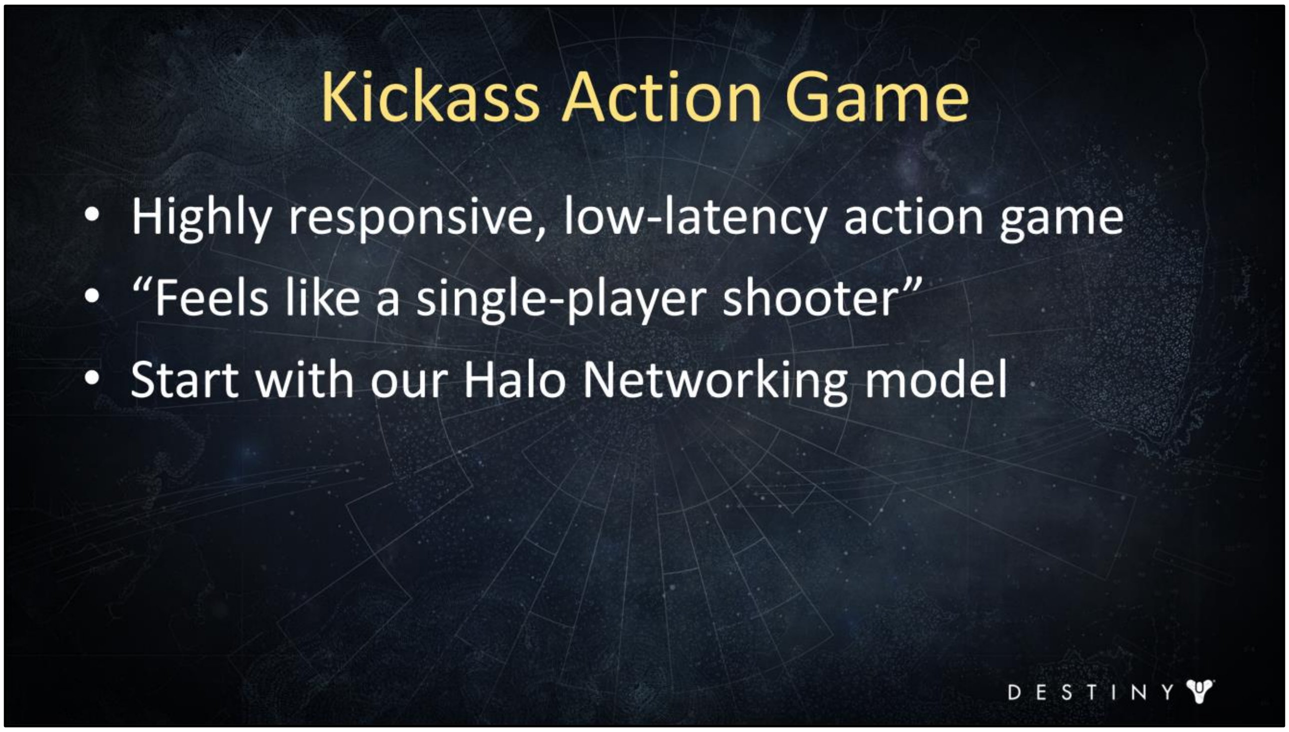
- Highly responsive, low-latency action game
- “Feels like a single-player shooter”
- Start with our Halo Networking model

DESTINY

“Kickass Action Game” means we needed Halo Parity. FPS genre parity. We needed to make a highly responsive, low latency action game, that’s instantly familiar and competitive with all the other great FPSes out there.

The internal bar we used to great effect was

“it needs to feel like a single player shooter”.

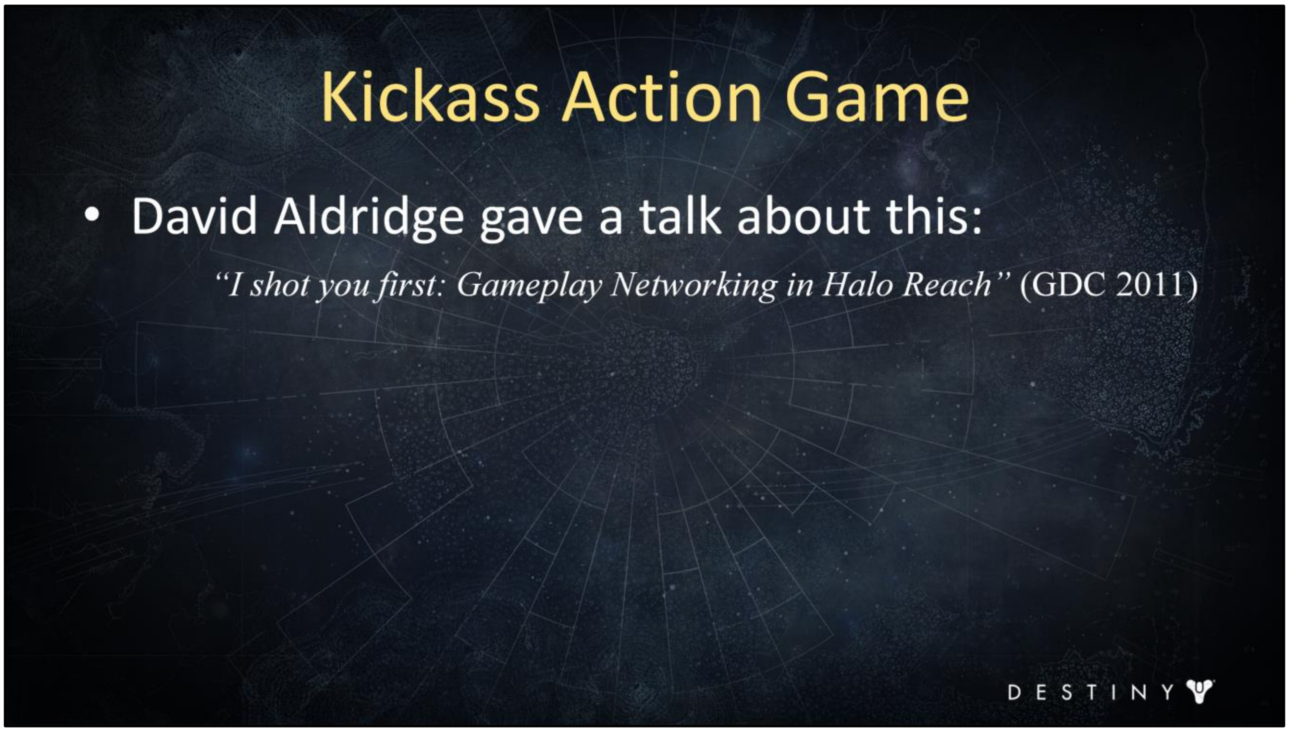


Kickass Action Game

- Highly responsive, low-latency action game
- “Feels like a single-player shooter”
- Start with our Halo Networking model

This was important to us because the online experience is not opt-in. We force you to matchmake with other players, even if all you want to do is play a solo campaign. Therefore we need to make sure that our matchmaking and networking goals never hurt that solo campaign experience.

So this means we started with the Halo codebase and networking model.



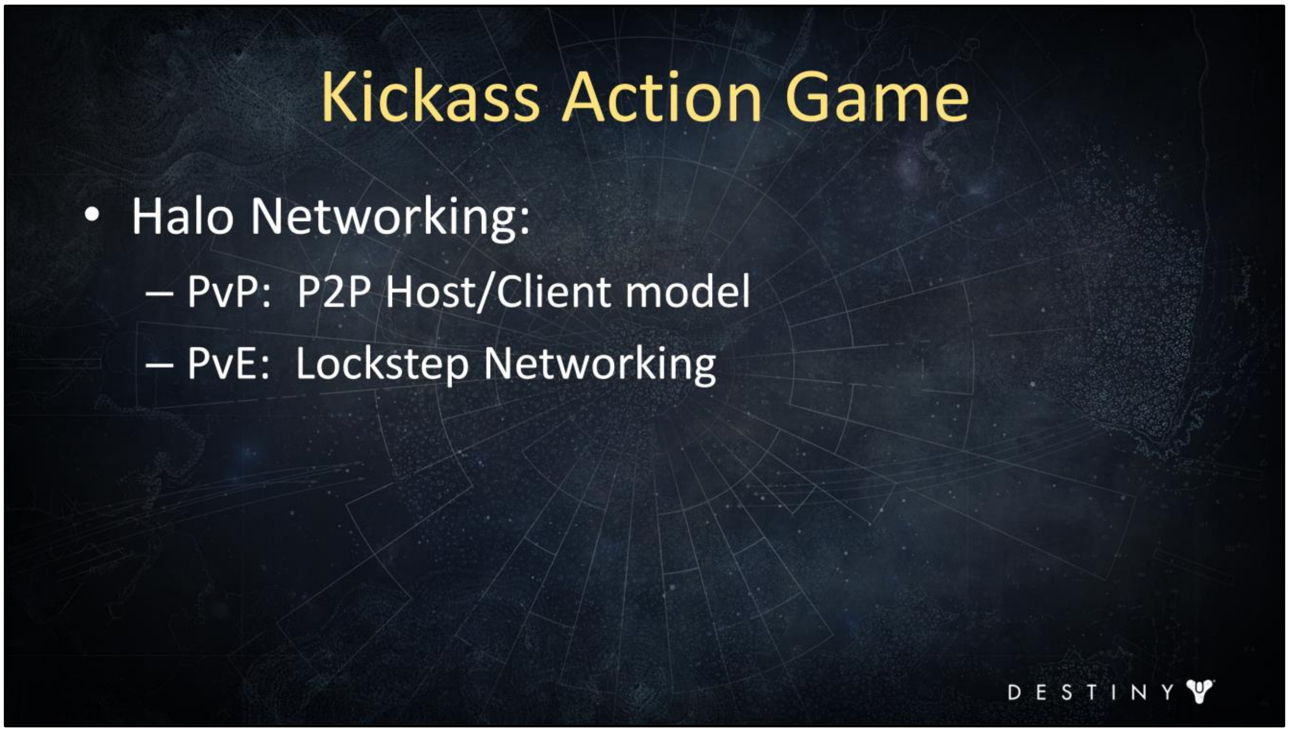
Kickass Action Game

- David Aldridge gave a talk about this:

  "I shot you first: Gameplay Networking in Halo Reach" (GDC 2011)

DESTINY

David Aldridge did a GDC talk 4 years ago about the Halo Reach Networking Model, which describes it in detail, I'm going to skip over most of that. You should really watch his talk, though, if you're interested in the details of our action game networking.

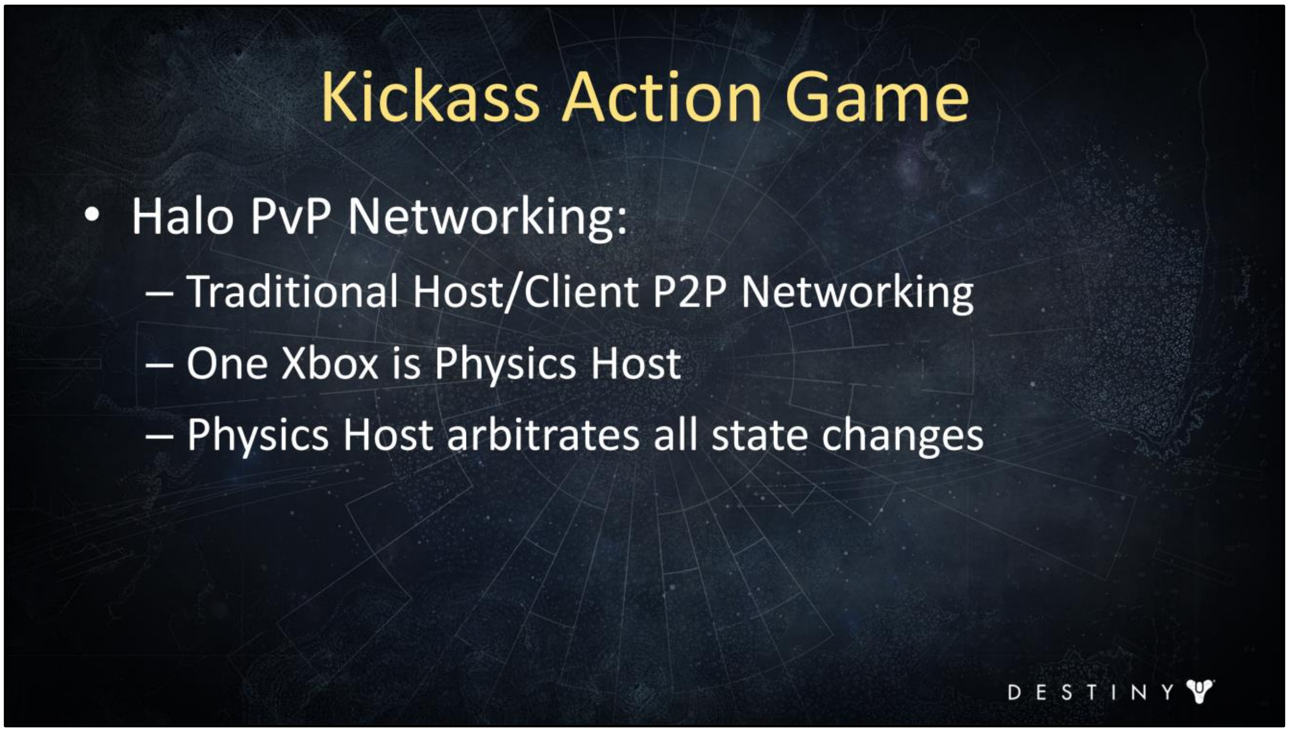


Kickass Action Game

- Halo Networking:
  - PvP: P2P Host/Client model
  - PvE: Lockstep Networking

But I will quickly touch on some Halo Networking terminology, which I'll be relying on later in this talk.

The most important thing to know about Halo: Reach was that it had 2 networking models – one for PvP and one for PvE.

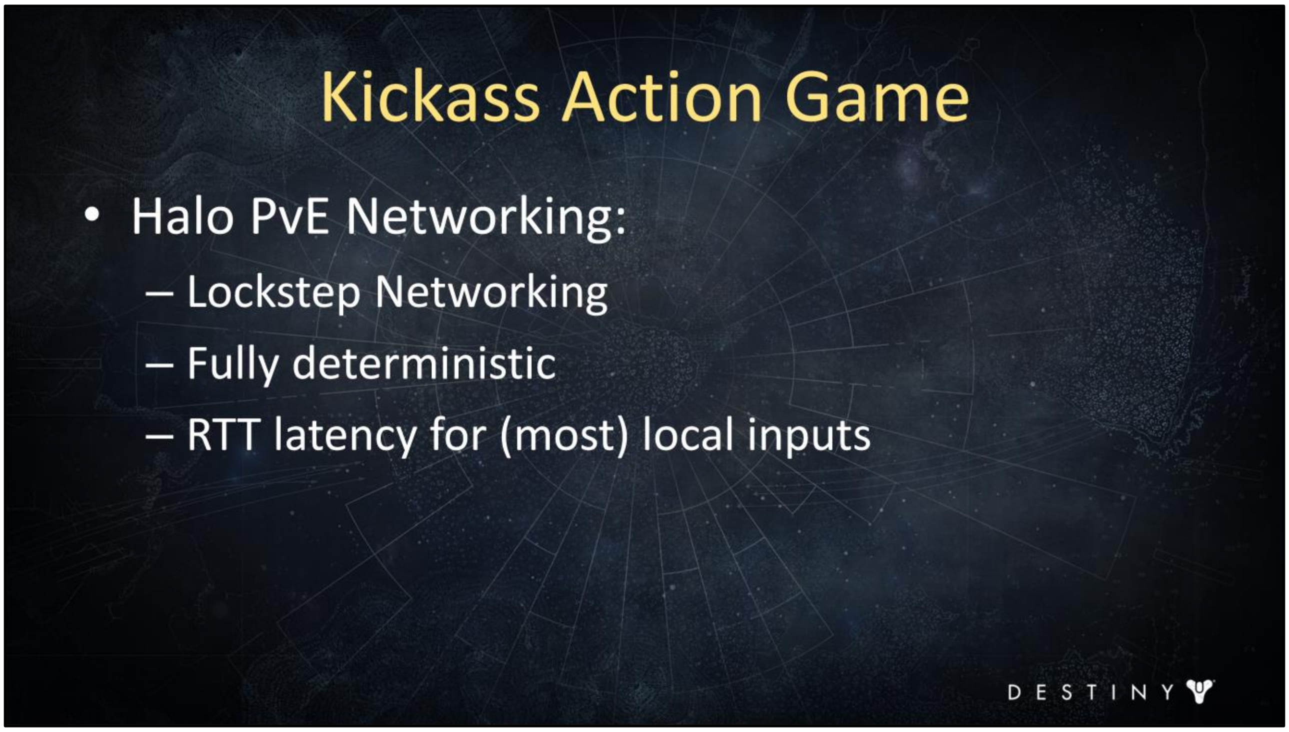


Kickass Action Game

- Halo PvP Networking:
  - Traditional Host/Client P2P Networking
  - One Xbox is Physics Host
  - Physics Host arbitrates all state changes

DESTINY

PvP used standard Peer-to-peer Host/Client networking. One Xbox was the Host of a game (let’s call that the “Physics Host” for reasons that are useful later), the other Xbox’s were all clients communicating with that host, but the Host arbitrated all state changes.



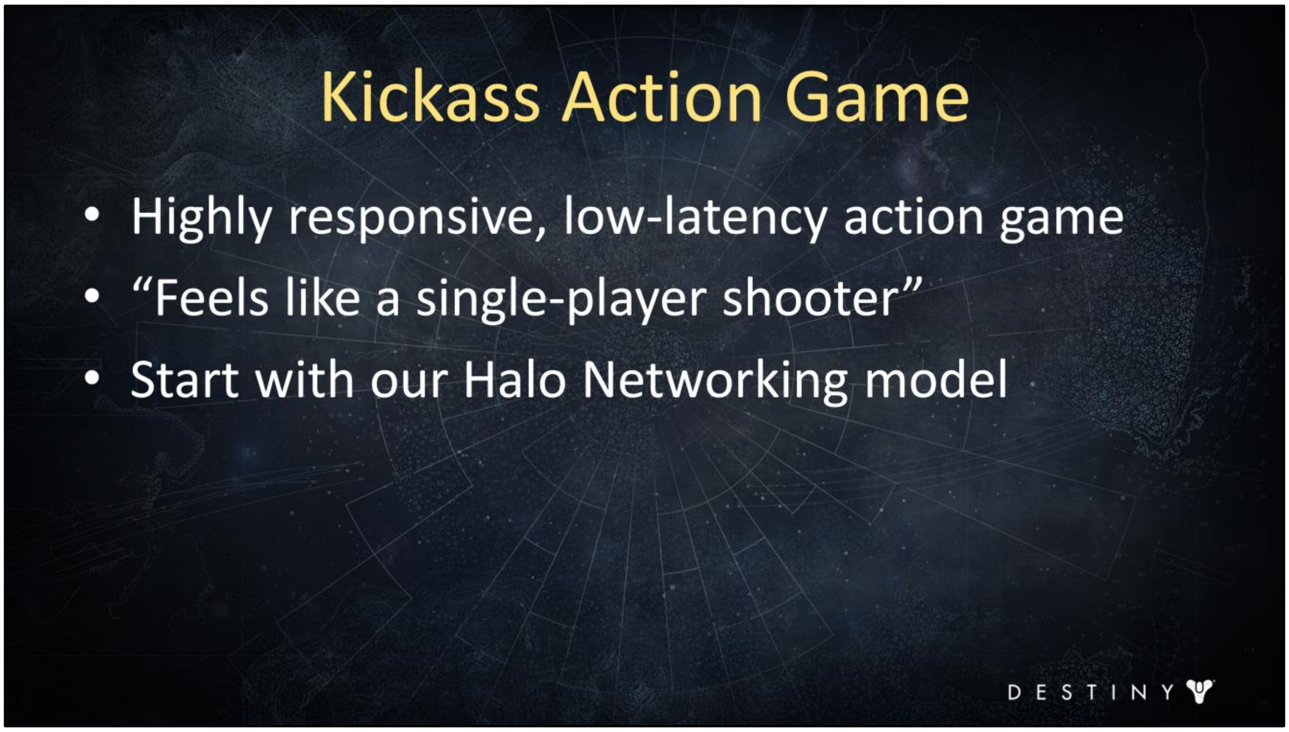
Kickass Action Game

- Halo PvE Networking:
  - Lockstep Networking
  - Fully deterministic
  - RTT latency for (most) local inputs

For PvE we used Lockstep Networking – this is the networking model most commonly used for stuff like a real-time-strategy game. If your game is fully deterministic (the same set of inputs always produce the same set of simulation outputs), then you can just network the controller inputs from each machine, and not simulate the next tick until you've received the inputs from everyone else about what should happen.

We do some tricks in our Lockstep Networking to hide that latency, but ultimately you have to pay full RTT networking time between when you pull the trigger and when your gun fires.

So that meant that while we had a networking model in Halo: Reach for campaign/PvE games (you could play through the whole Halo: Reach campaign with 4 players), it was a noticably more latent experience than our PvP game.



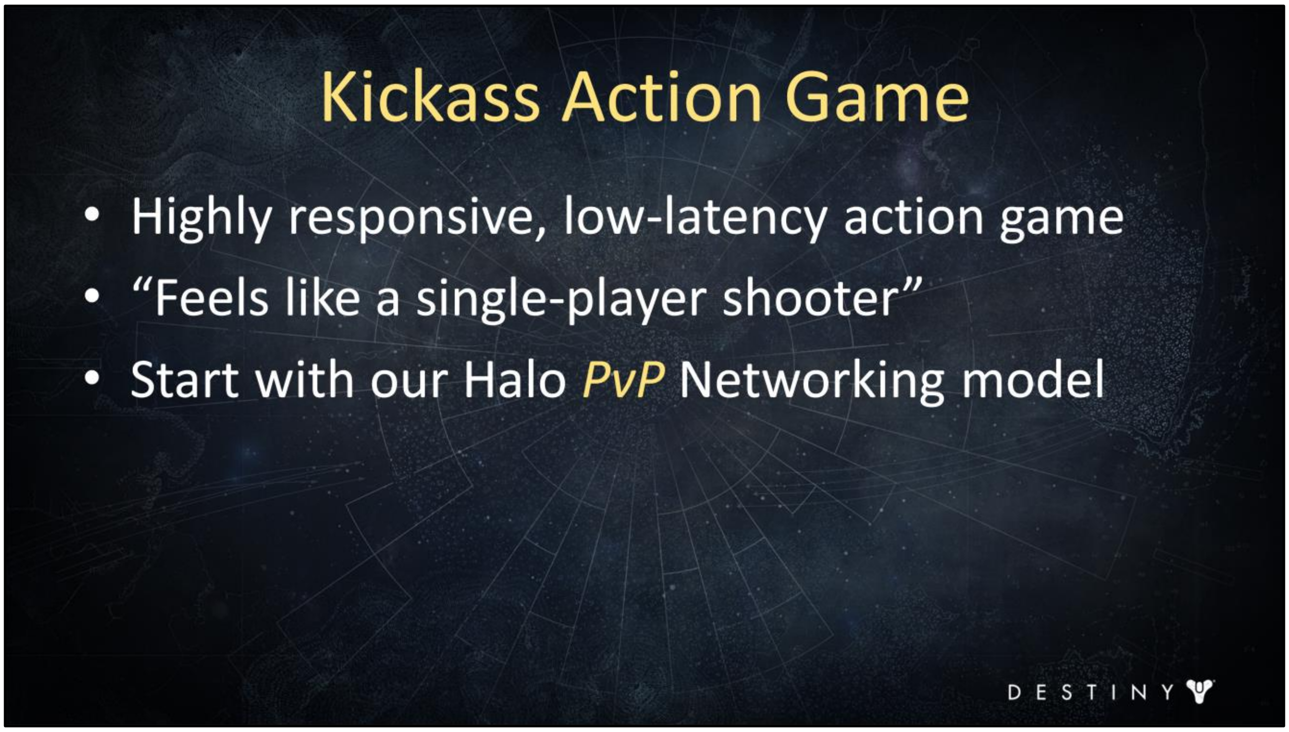
Kickass Action Game

- Highly responsive, low-latency action game
- “Feels like a single-player shooter”
- Start with our Halo Networking model

DESTINY

So, going back to the original design goals – this lockstep networking latency was unacceptable for the “Feels like a single-player shooter” goal I mentioned earlier.

So that meant that we chose to instead start with our Halo-era … [PvP]

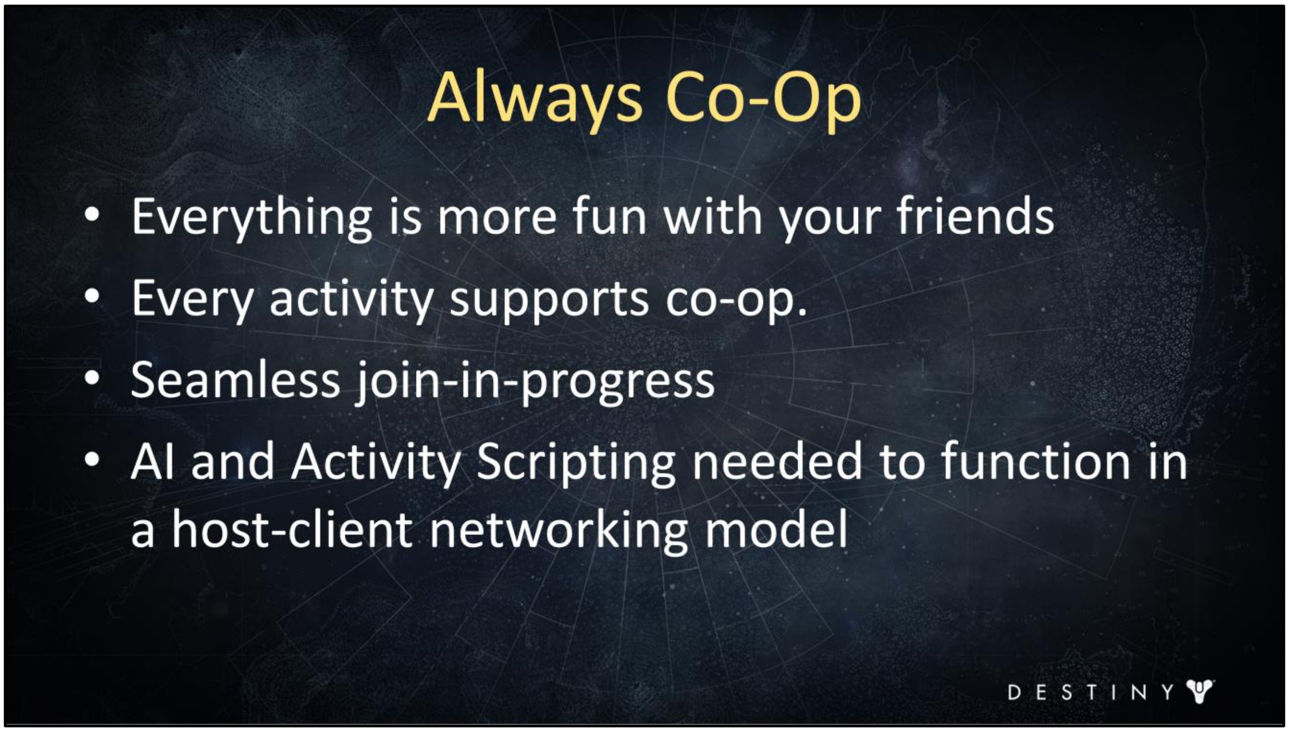


Kickass Action Game

- Highly responsive, low-latency action game
- "Feels like a single-player shooter"
- Start with our Halo *PvP* Networking model

PvP Networking model, but had to extend it to support the full story campaign.

Which leads well into the next pillar



Always Co-Op

- Everything is more fun with your friends
- Every activity supports co-op.
- Seamless join-in-progress
- AI and Activity Scripting needed to function in a host-client networking model

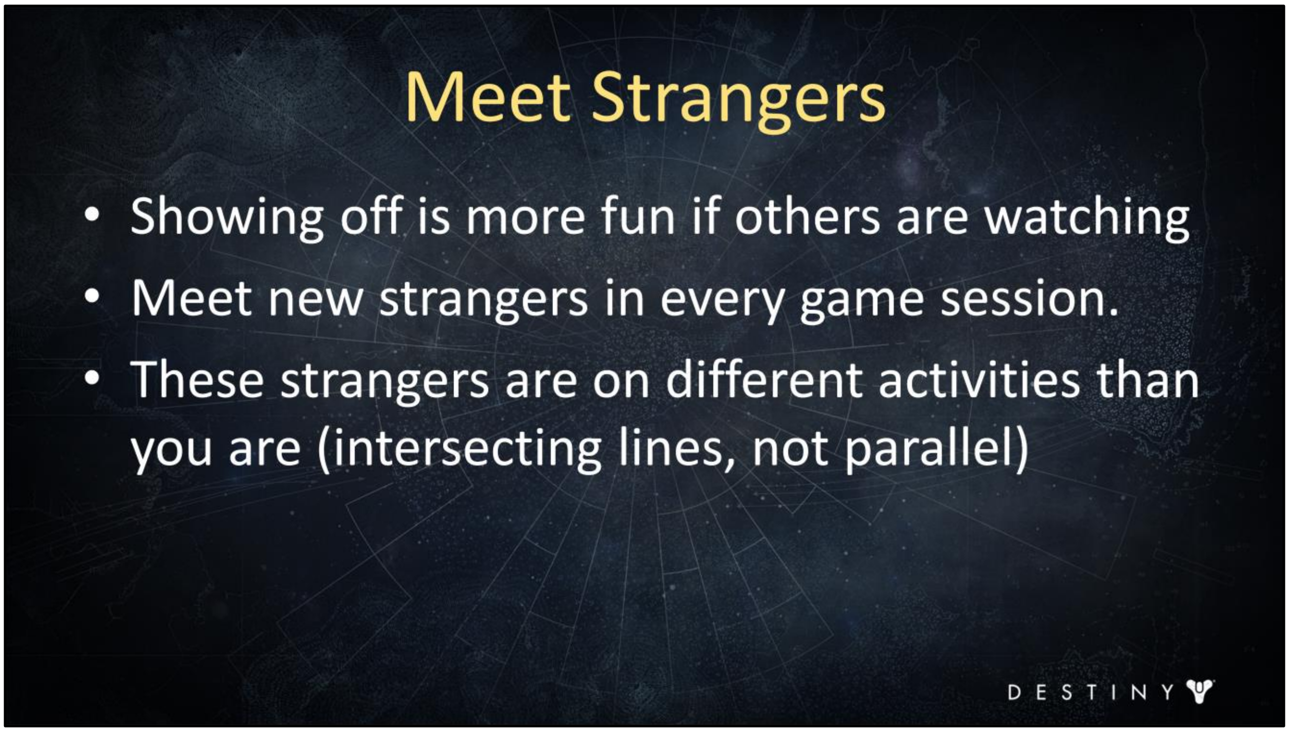
"Everything is more fun with your friends" – this meant to us that every activity supports co-op gameplay. Always.

This also meant that you can always hook up with your friends – every activity supports join-in-progress, and we endeavor to make that available at all times.

For our cooperative campaign, that meant that while we could start from our PvP Networking model, we had to build it out to support AI and complex Activity Scripting for the first time.

Lots of cool stuff was done to accomplish this networking goal for AI, but I'm not giving that talk either.

I am going to talk about Activity Scripting, though, later on.



Meet Strangers

- Showing off is more fun if others are watching
- Meet new strangers in every game session.
- These strangers are on different activities than you are (intersecting lines, not parallel)

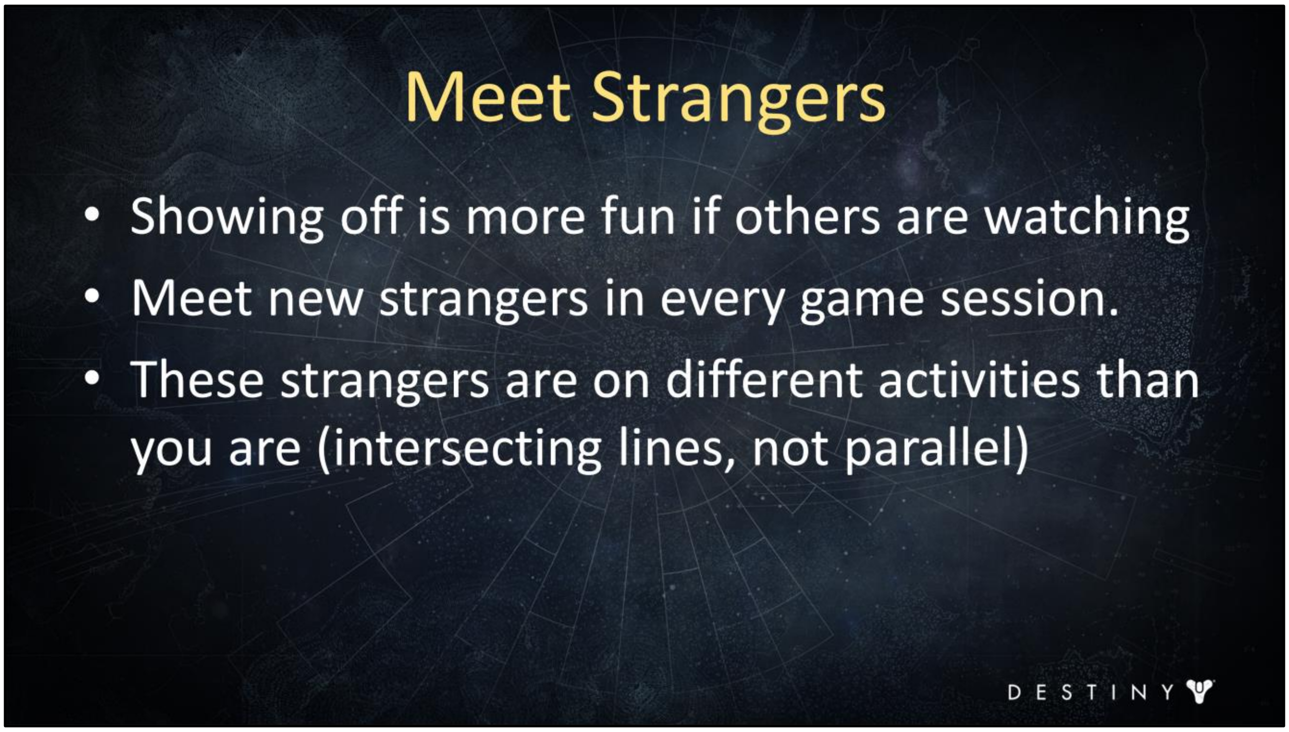
DESTINY

Moving on to our 3rd pillar, which I like to state as “Showing Off is more fun if others are watching”

At Bungie, we believe really strongly that, even with the minimal social interaction verbs we provided in Destiny, the mere existence of other players, perceiving you and your avatar, gives value to your actions.

Xbox360 Achievements were intrinsically more valuable than in-game rewards, because you could easily show them off to your friends.

As another example, as soon as you build something awesome in minecraft, you immediately want to show it to your other friends who play minecraft



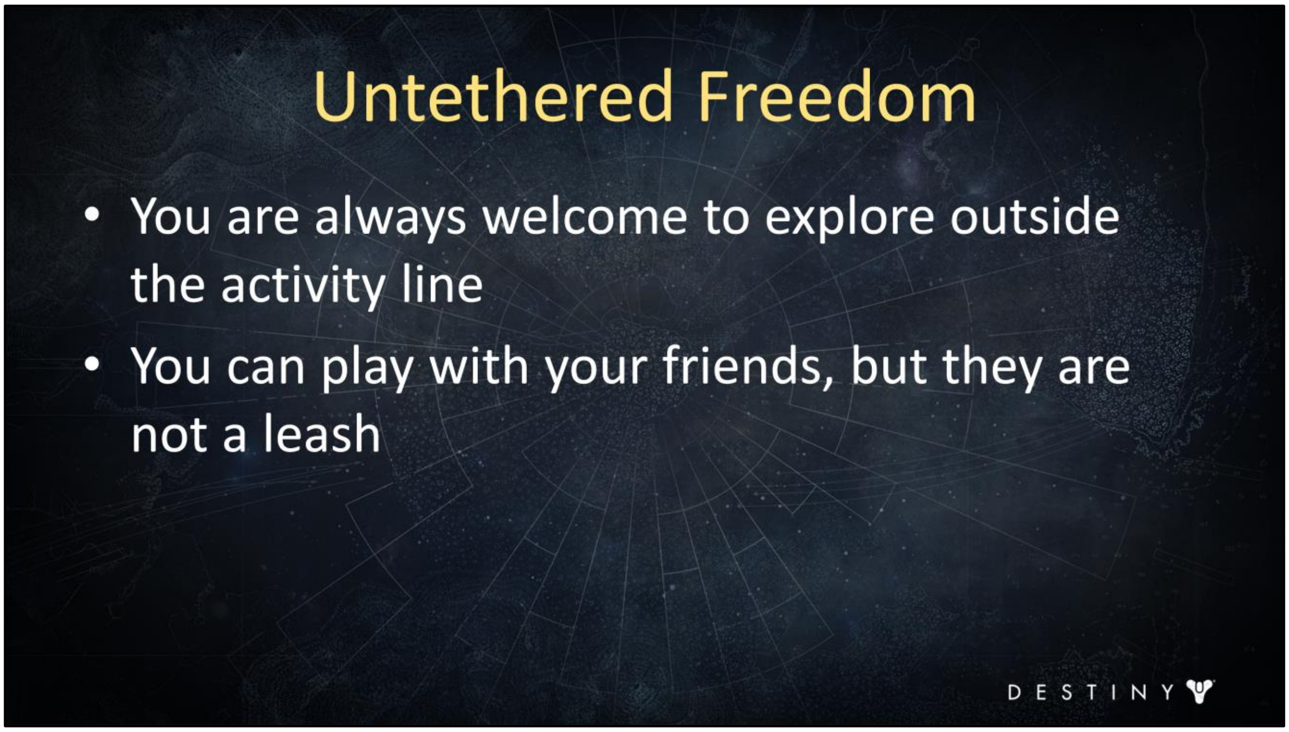
Meet Strangers

- Showing off is more fun if others are watching
- Meet new strangers in every game session.
- These strangers are on different activities than you are (intersecting lines, not parallel)

DESTINY

So we're convinced that if we can take even a solo player with no friends on Xbox Live or PSN, and regularly put them in rooms with other people, they will care more about their fancy hat and their attack power and the level number over their head.

We also wanted these strangers to not be pursuing the same goals as you. We think of it as intersecting, not parallel lines – we don't want these strangers competing with you for resources, or pushing you forward at a faster pace than you'd like. Instead, intersecting with strangers that are not directly competing for any of your goals minimizes friction and potential resentment.



Untethered Freedom

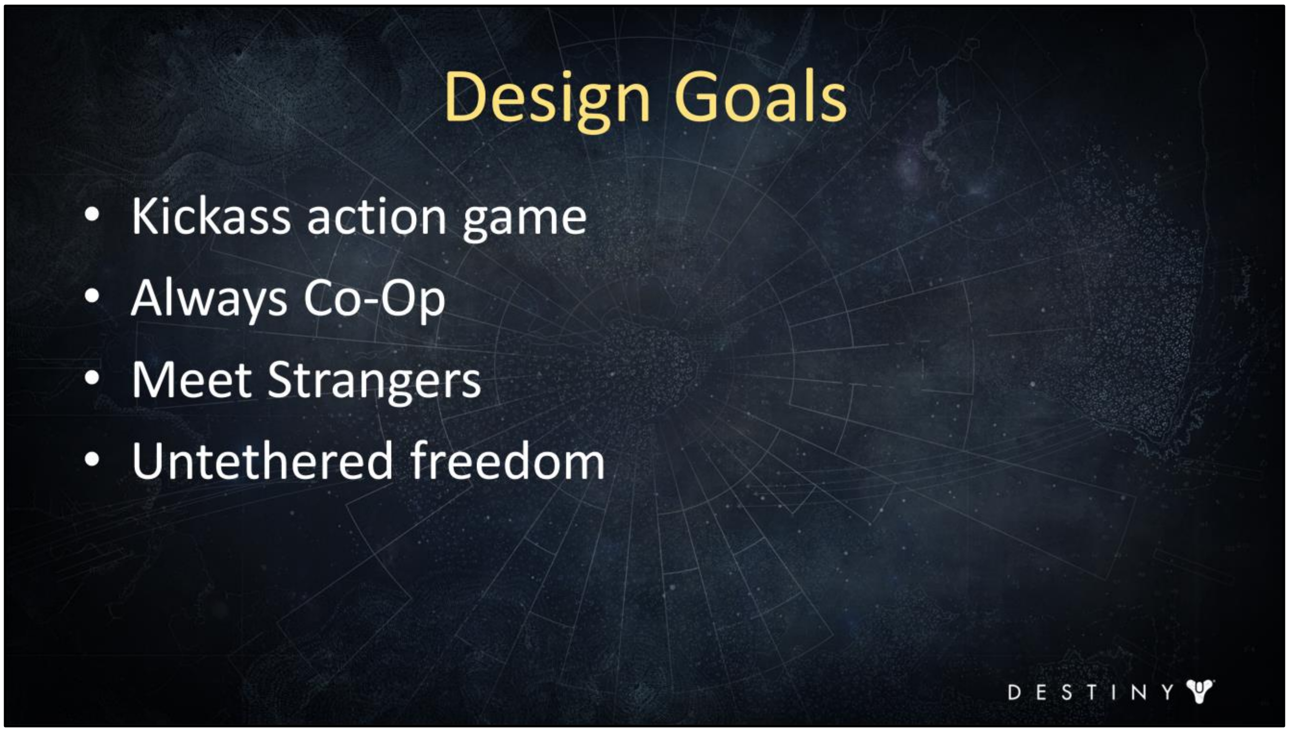
- You are always welcome to explore outside the activity line
- You can play with your friends, but they are not a leash

Our last pillar I'll be talking about was "Untethered Freedom to Explore"

This means that our Activity Designs shouldn't stop you from leaving the activity line and just exploring.

In Halo co-op, you were always tethered to a maximum radius from the Physics Host. If you fell behind, or tried to run too far away, we'd teleport you back to the Host.

In Destiny, however, we wanted to let you meet up with your friends, but have parallel play in different areas of the same destination. You can run alongside them for a mission, or stop to harvest some spinmetal while your friend rushes ahead.



Design Goals

- Kickass action game
- Always Co-Op
- Meet Strangers
- Untethered freedom

DESTINY

So, these are the 4 design pillars our architecture was trying to satisfy.

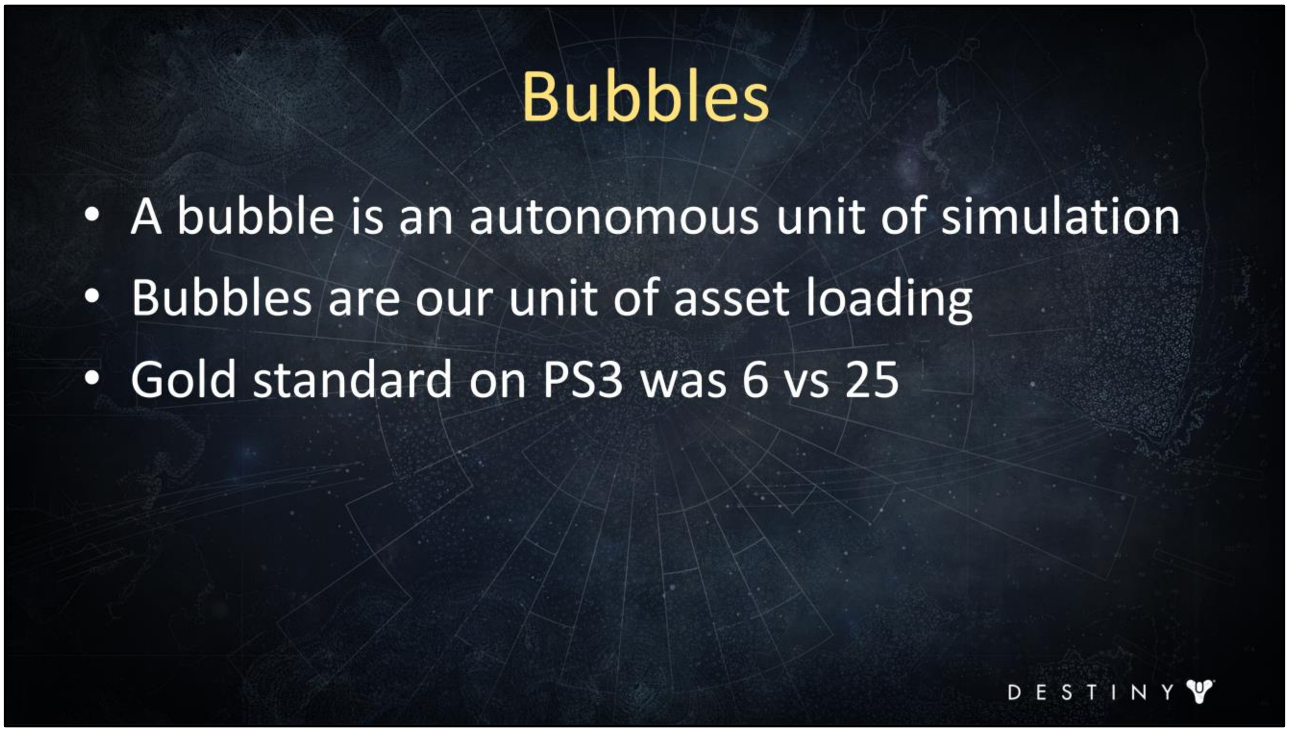
And looking at them, I feel like they're pretty scary from a risk perspective, especially when stacked up with "New IP", "1st Multiplatform title", and "New Engine".

Therefore, while we were setting ambitious design goals, we also set some early scoping constraints on our architecture. One of the strongest early constraints is what we call "Bubbles".

Bubbles

And it’s important to think about them as a constraint – Bubbles were not an accomplishment of a design goal or vision, they were a scoping decision. I could certainly imagine a version of Destiny without these, but I couldn’t imagine us shipping in 2014 without these.

So what is a bubble?

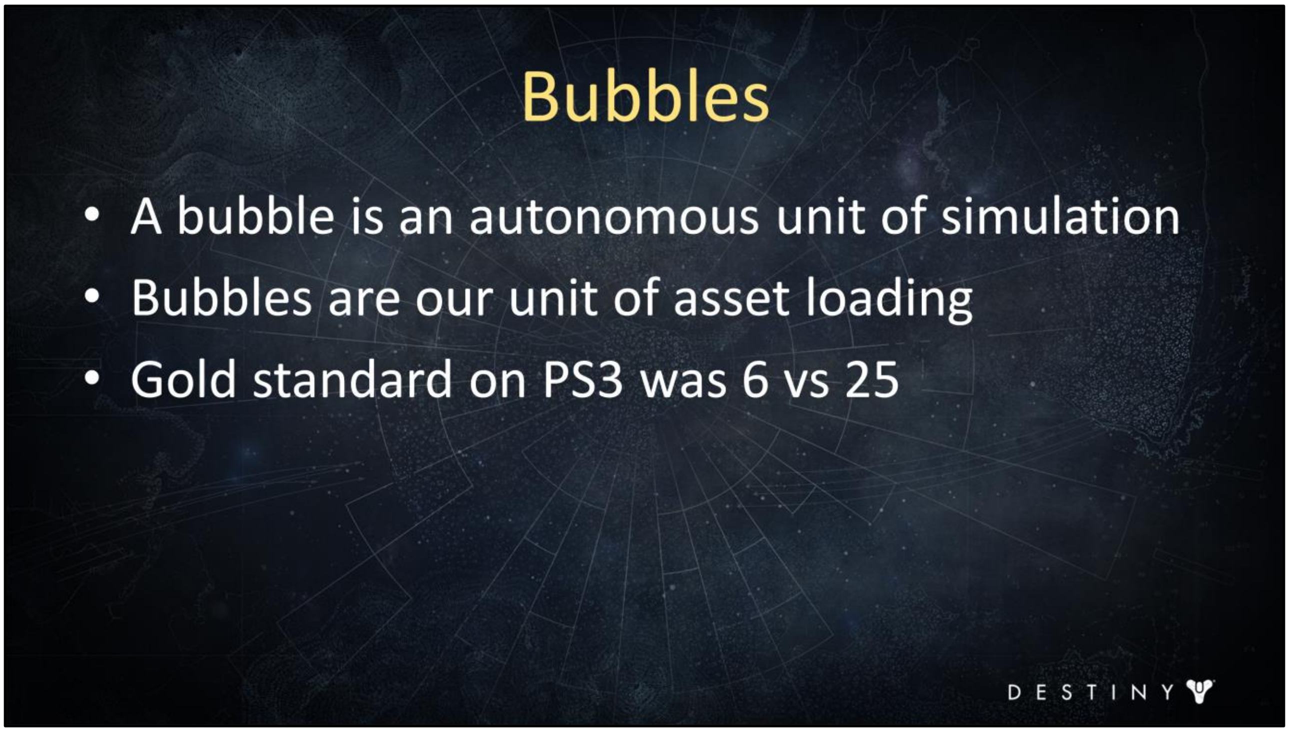


Bubbles

- A bubble is an autonomous unit of simulation
- Bubbles are our unit of asset loading
- Gold standard on PS3 was 6 vs 25

A bubble is our unit of simulation – a player is only ever simulating a single bubble's worth of combat and physics at any time.
A bubble is also our unit of asset streaming – you have at most 2 bubble's asset memory loaded at any time – the one you're currently simulating, and the one you're precaching.

For Simulation perf, we had a bunch of different perf rules depending on max player count, vehicles, etc., but our standard bubble was 6 vs 25 – that meant 6 players, and 25 AI in a single bubble active at any one time.



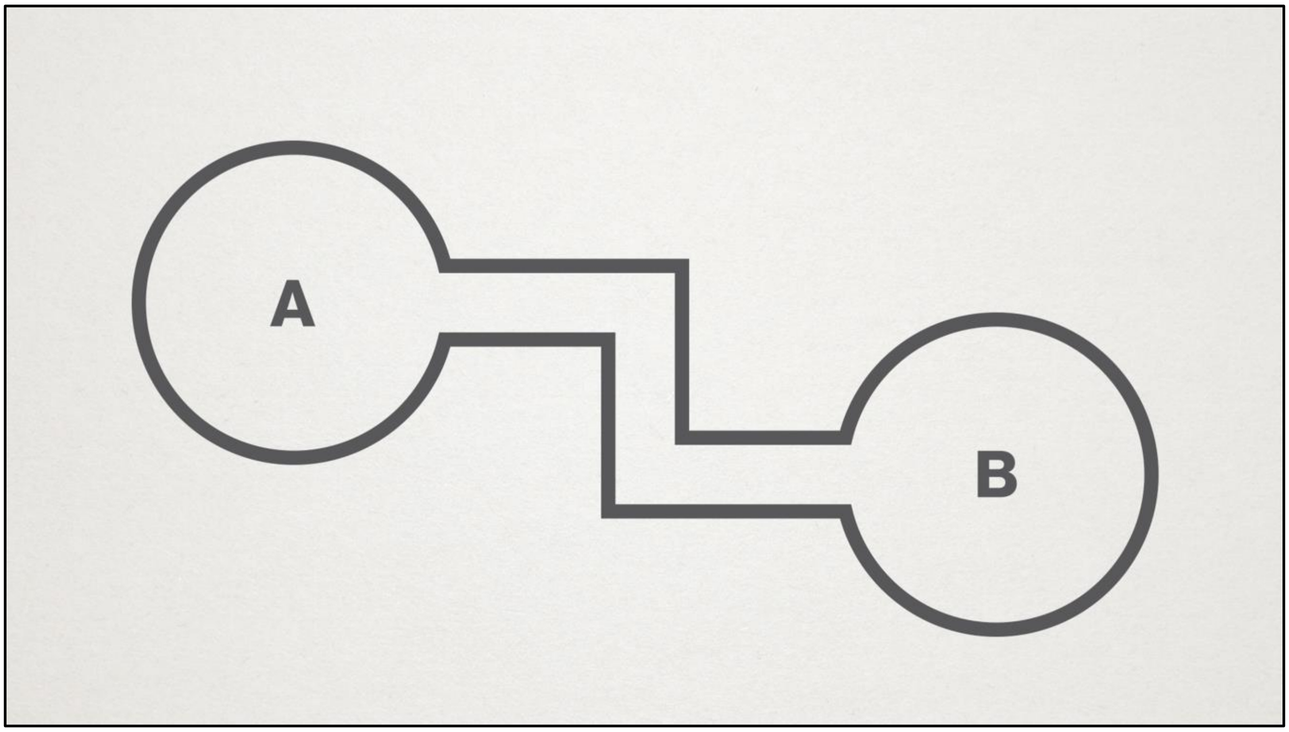
Bubbles

- A bubble is an autonomous unit of simulation
- Bubbles are our unit of asset loading
- Gold standard on PS3 was 6 vs 25

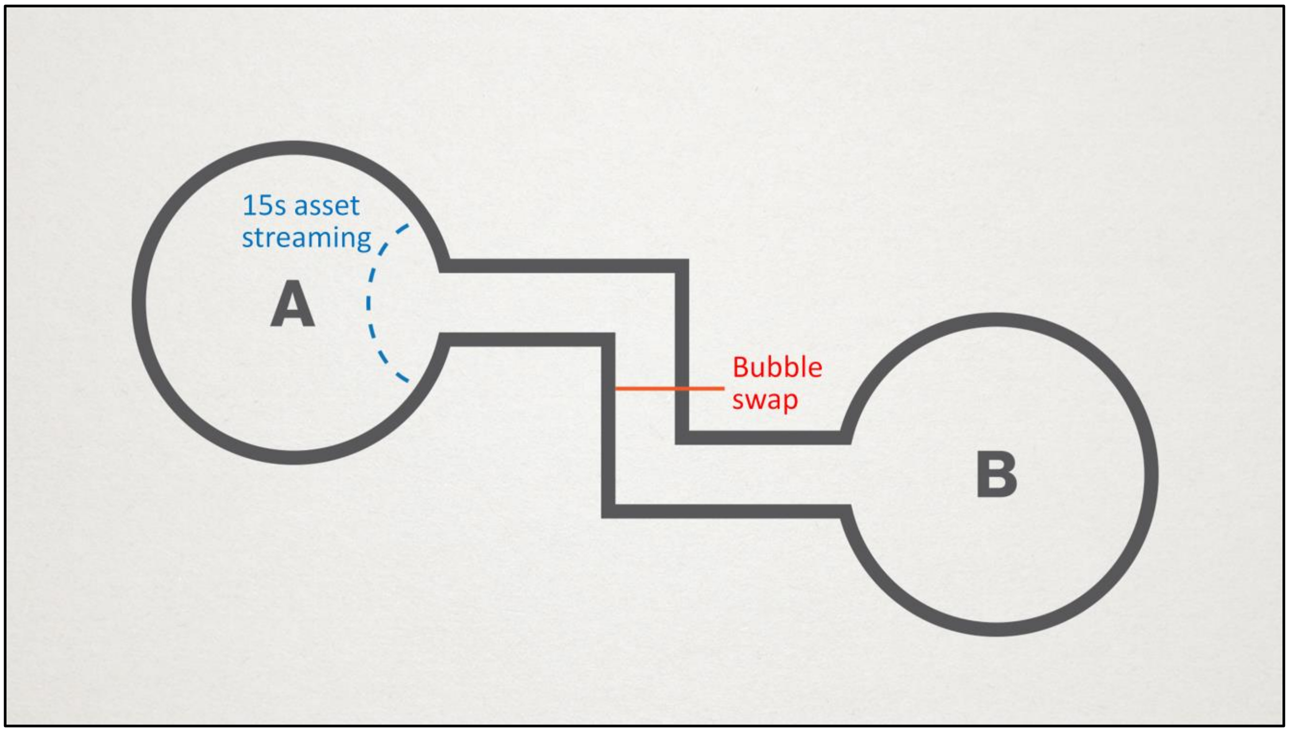
DESTINY

25 AI ought to be enough to give you the kickass action game, and with 3-player fireteams, a population of 6 gives you the ability to play with friends and have 3 strangers to interact with. But over a one-hour experience in Destiny, we need a lot more than 3 strangers.

That’s why bubbles are also our unit of matchmaking.

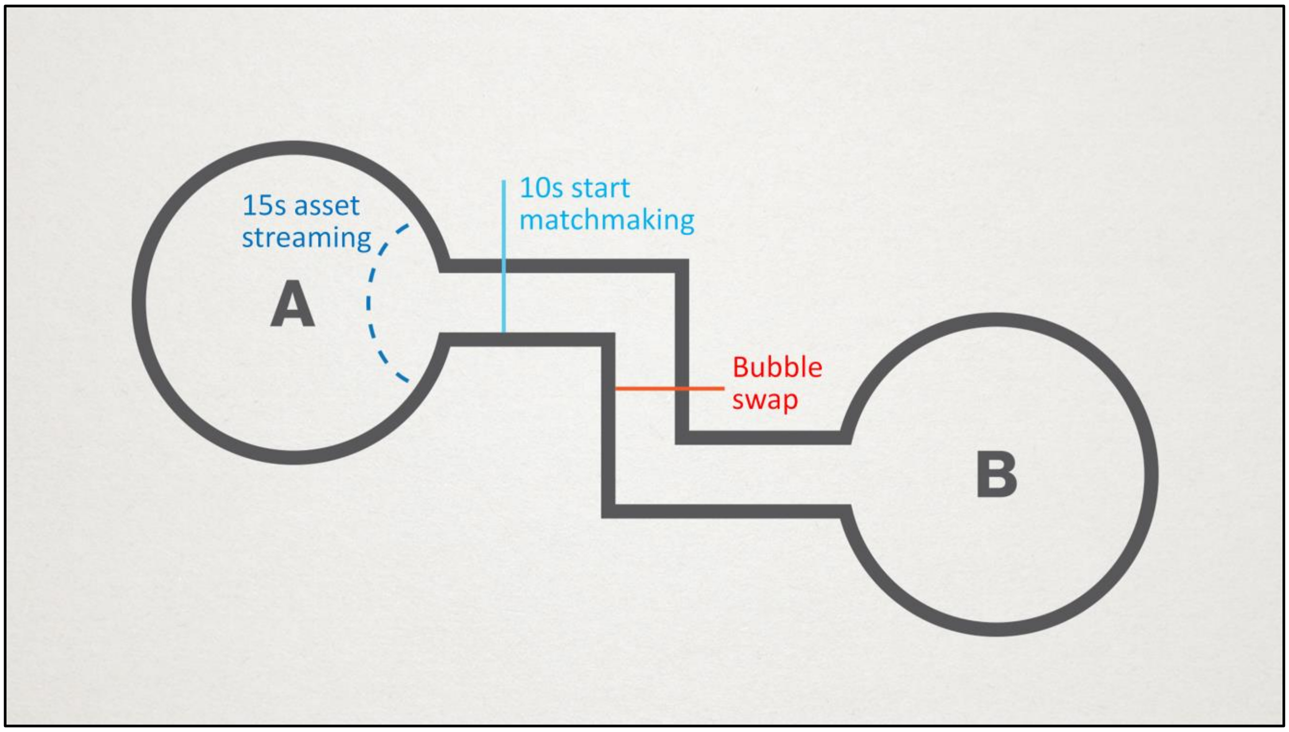


So here’s an example of 2 bubbles, with a z-leg transition between them. From an asset-loading perspective, this is pretty standard zone-based streaming.

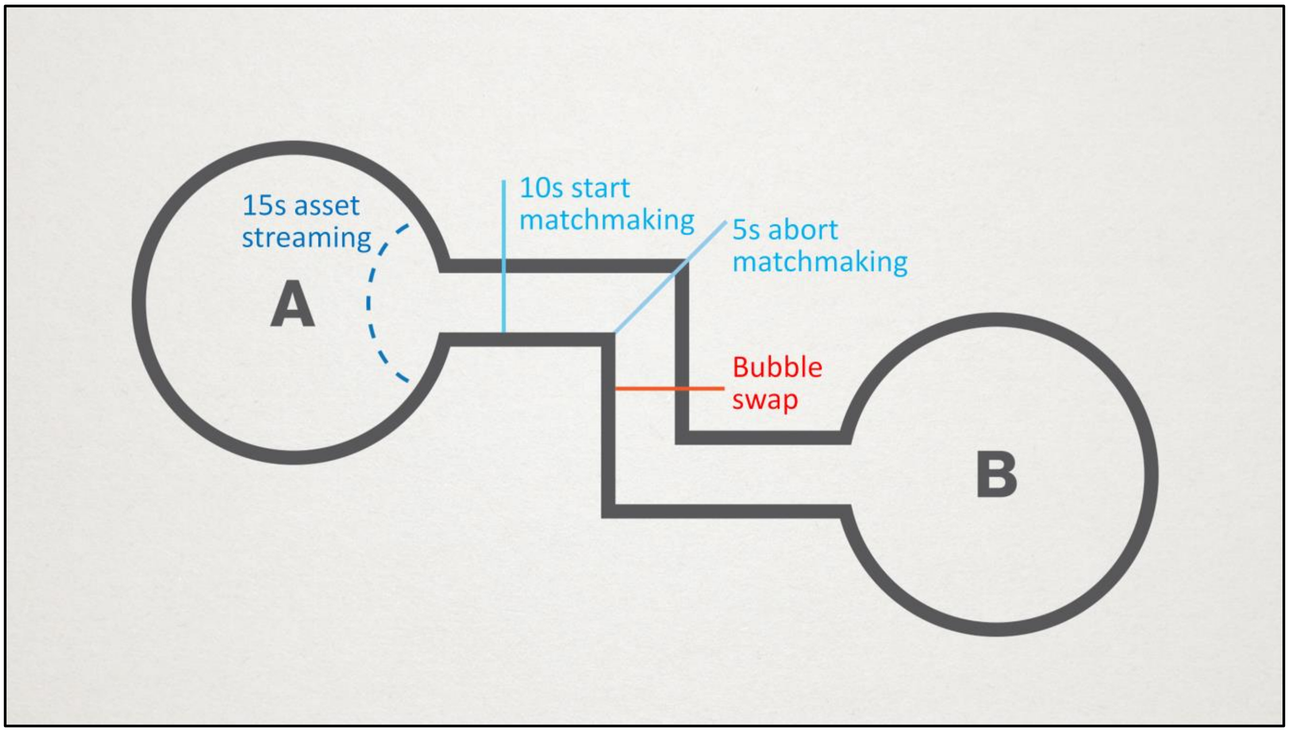


15s out from the bubble swap, you start precaching all resources for the new bubble.

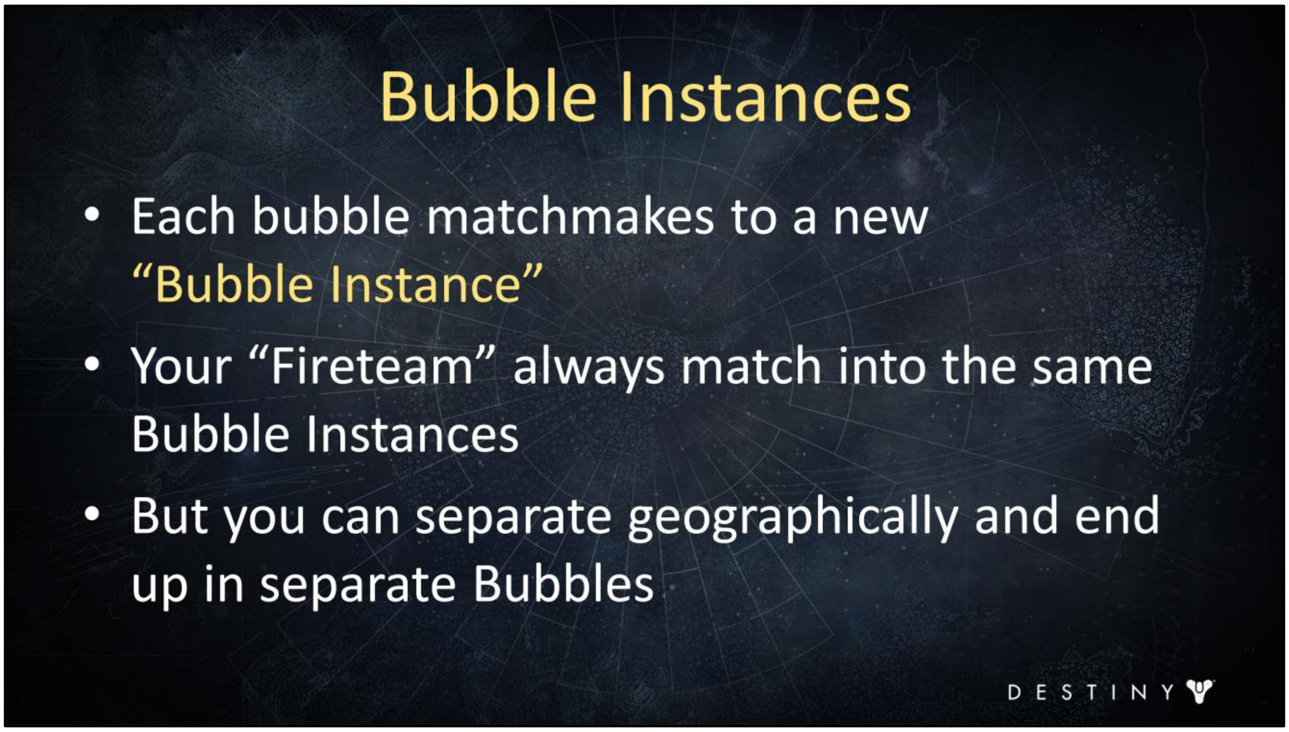
When you get to the “bubble swap” part at the end, you’ve got all your resources loaded for the new bubble, and can instantly deinstantiate everything in Bubble A, and instantiate everything in Bubble B.



The neat thing we also do during these transitions is matchmake you to a new networked game. In order to meet new strangers in bubble B, we start matchmaking for a new host at the 10s mark, and are ready to seamlessly swap you to the new game when we perform our bubble swap.



If we don't find a suitable bubble to match you with by the 5s mark, we abort matchmaking, which gives you enough time spin up your own standalone bubble (that other strangers should eventually join).

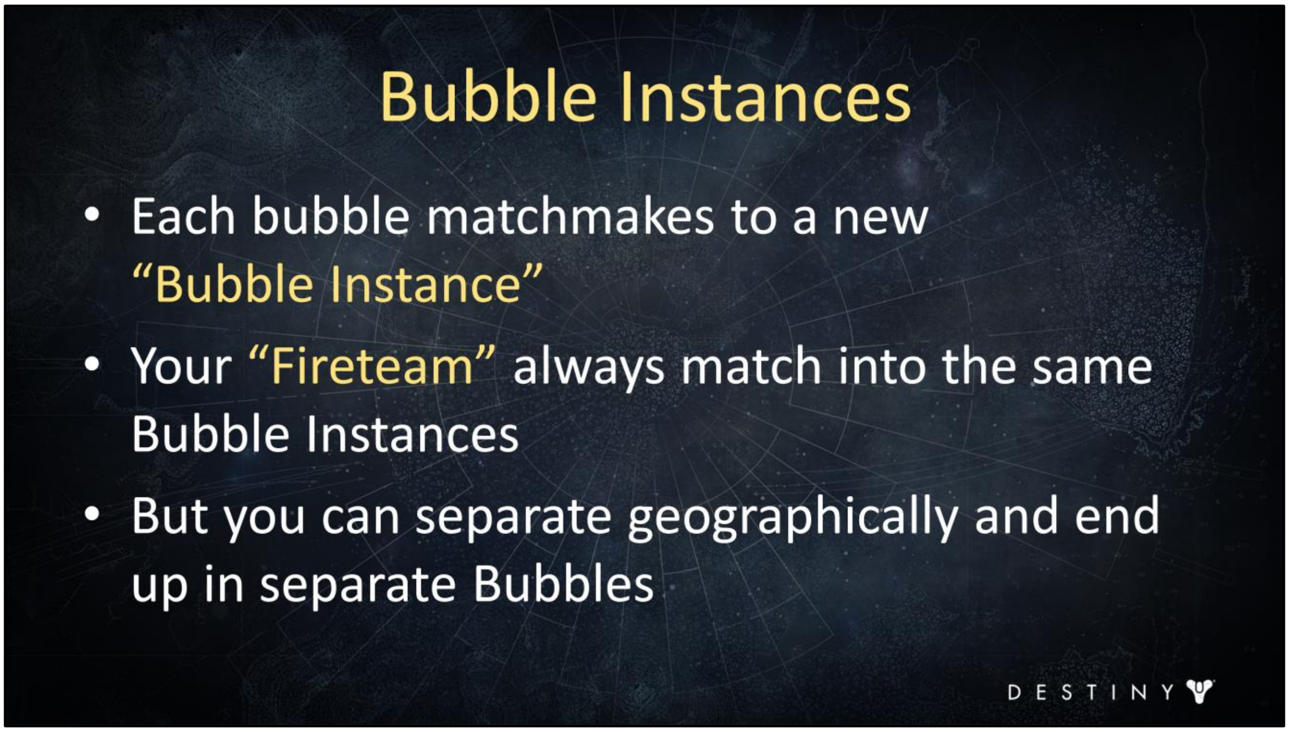


Bubble Instances

- Each bubble matchmakes to a new "Bubble Instance"
- Your "Fireteam" always match into the same Bubble Instances
- But you can separate geographically and end up in separate Bubbles

DESTINY

So, as you're running through the world, you're continuously matchmaking to new "Bubble Instances".



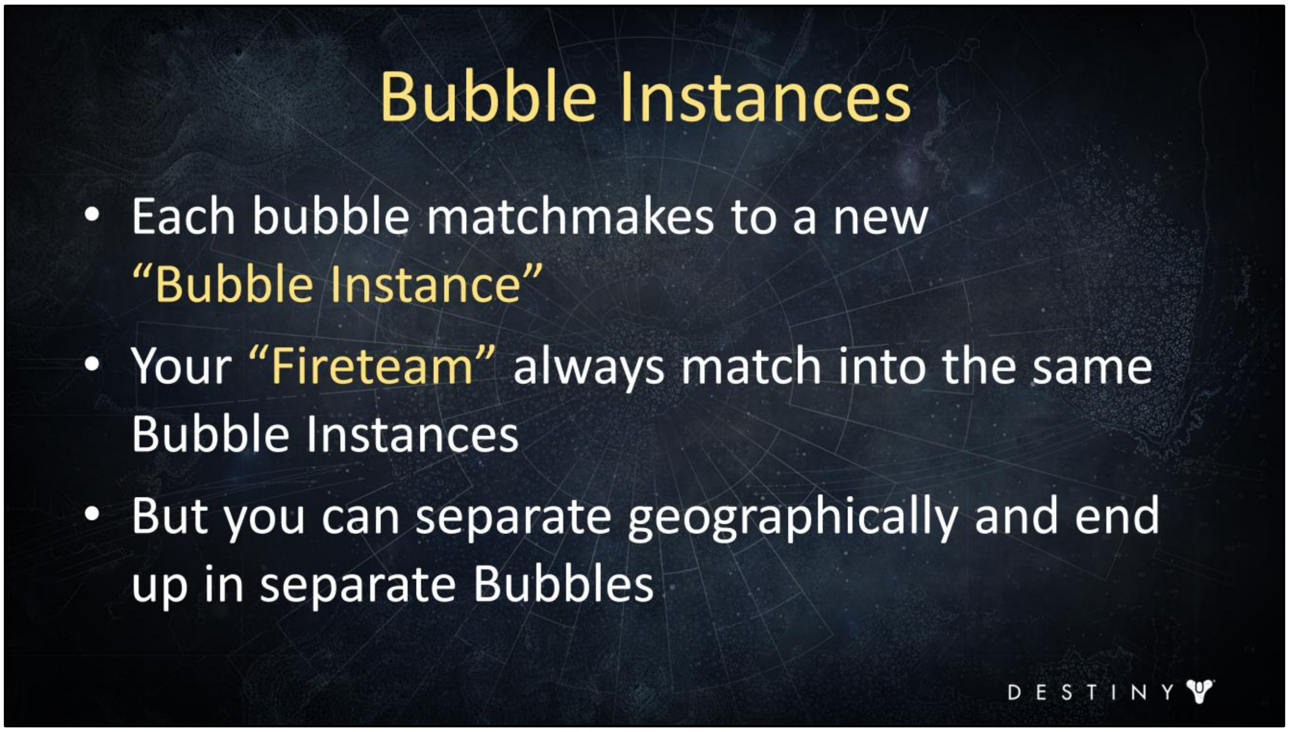
Bubble Instances

- Each bubble matchmakes to a new "Bubble Instance"
- Your "Fireteam" always match into the same Bubble Instances
- But you can separate geographically and end up in separate Bubbles

DESTINY

"Fireteam" is the term we use for a "party" – an intentionally formed group of friends, that all go on the same activity together. For most activities, the fireteam size limit is 3 (a notable exception is 6 fireteammates for Raids).

So you and your fireteammates are always guaranteed to matchmake into the same bubble instance if you go to the same geographic location.



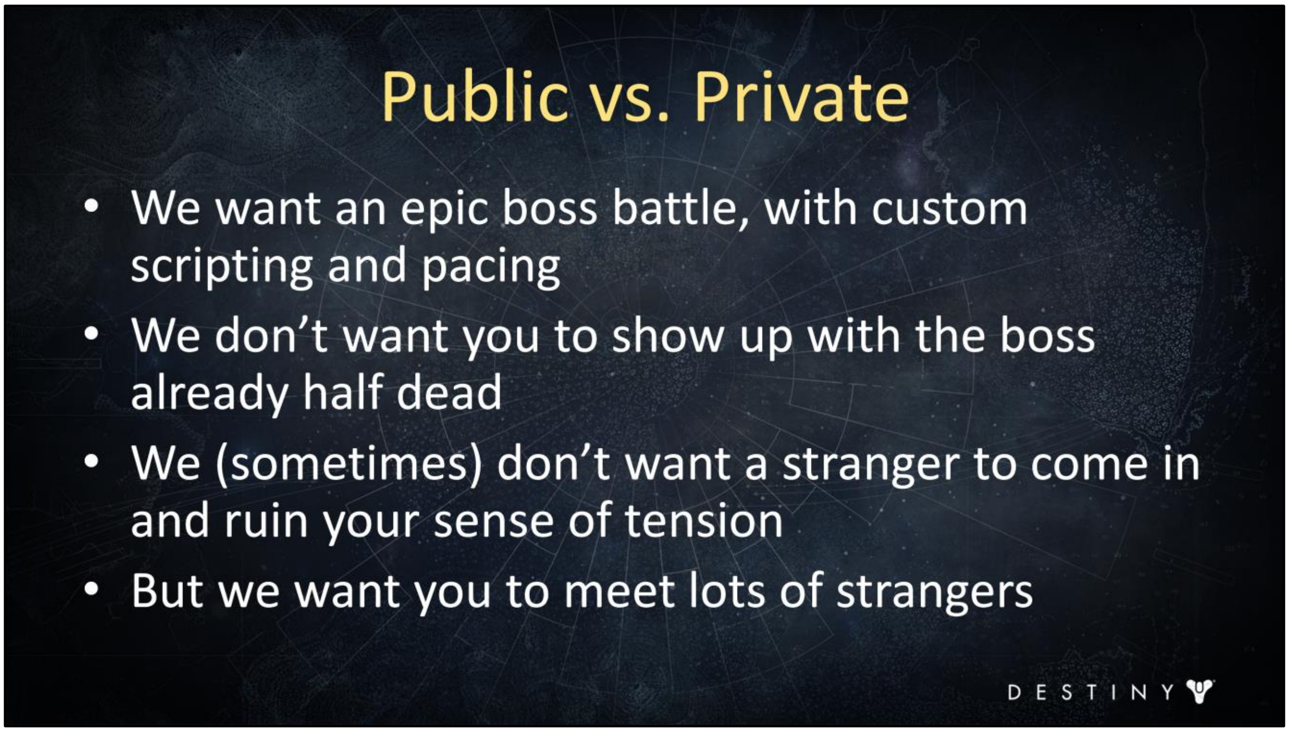
Bubble Instances

- Each bubble matchmakes to a new "Bubble Instance"
- Your "Fireteam" always match into the same Bubble Instances
- But you can separate geographically and end up in separate Bubbles

DESTINY

But while your fireteam is guaranteed to matchmake to the same bubble instances, you're not actually required to stay together – you're untethered, which means you could be in an activity with some fireteammates, but each of you is matched and connected to a completely different bubble instance in a different part of the world.

Strangers, on the other hand, will match with you in one bubble, but your connection to them is not guaranteed if you leave that bubble. So you and a stranger could matchmake together in one bubble, then run through a bubble transition side-by-side, and end up matching into separate bubble instances on the other side. On your screen he'll just phase out midway through the z-leg and disappear.


Public vs. Private

- We want an epic boss battle, with custom scripting and pacing
- We don't want you to show up with the boss already half dead
- We (sometimes) don't want a stranger to come in and ruin your sense of tension
- But we want you to meet lots of strangers

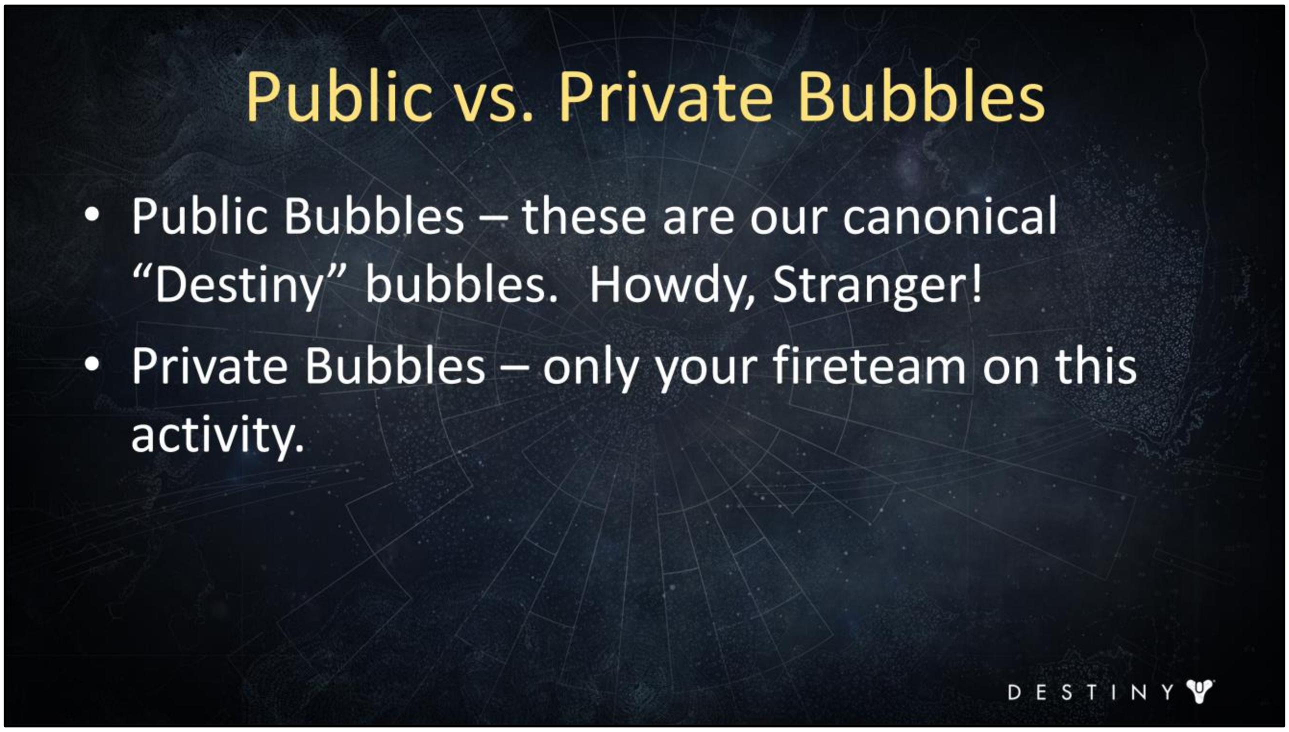
DESTINY

So, bubbles are our unit of autonomous simulation and matchmaking, but not all bubbles are the same.

If we talk about design goals again – we definitely want to achieve the pacing, spectacle, and designer-curated experiences of a Halo or other single-player-FPS. So we want you to be able to go to a boss battle, and not have some random stranger ruin the experience for you (or potentially show up and the boss is already half-dead).

On the other hand, we also really want you to meet strangers, and these goals are fundamentally at odds if we try to satisfy them simultaneously.

So we settled on the notion of “Public” vs. “Private” bubbles.



Public vs. Private Bubbles

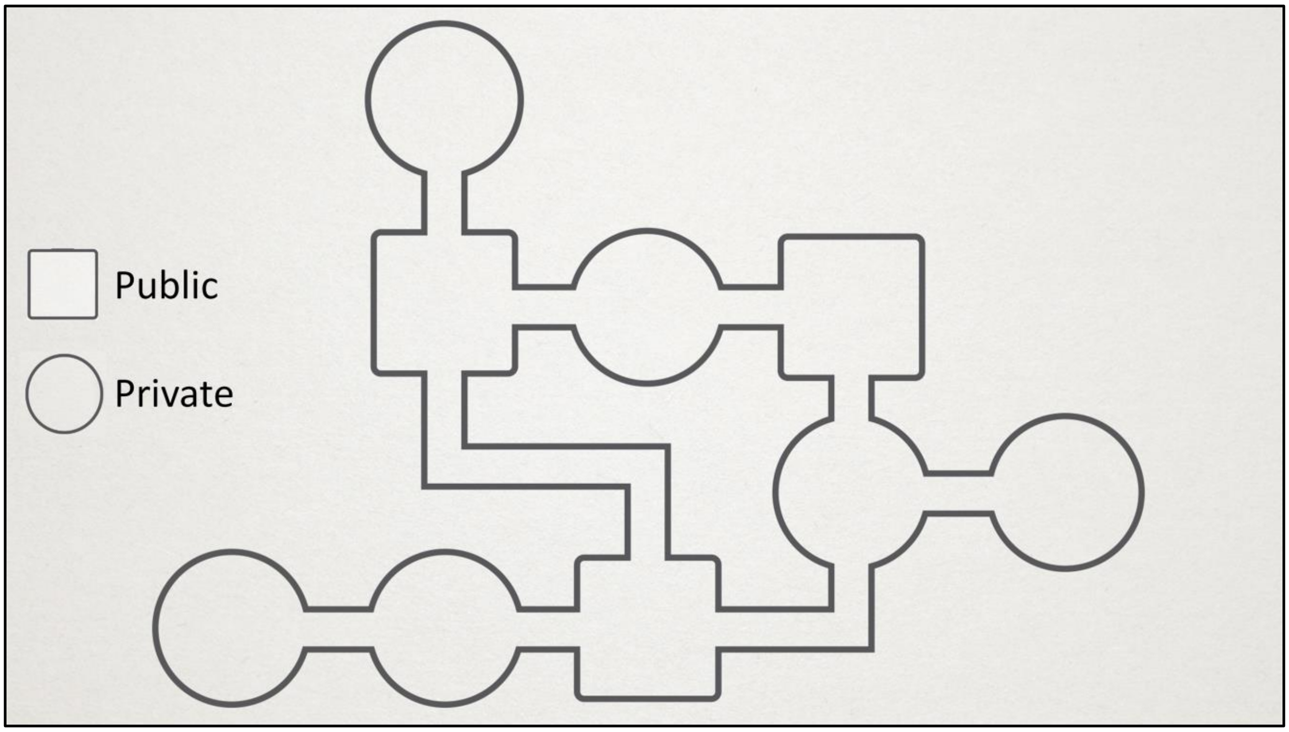
- Public Bubbles – these are our canonical "Destiny" bubbles. Howdy, Stranger!
- Private Bubbles – only your fireteam on this activity.

DESTINY

Public Bubbles are the canonical Destiny experience – lots of strangers interacting on intersecting activity lines.

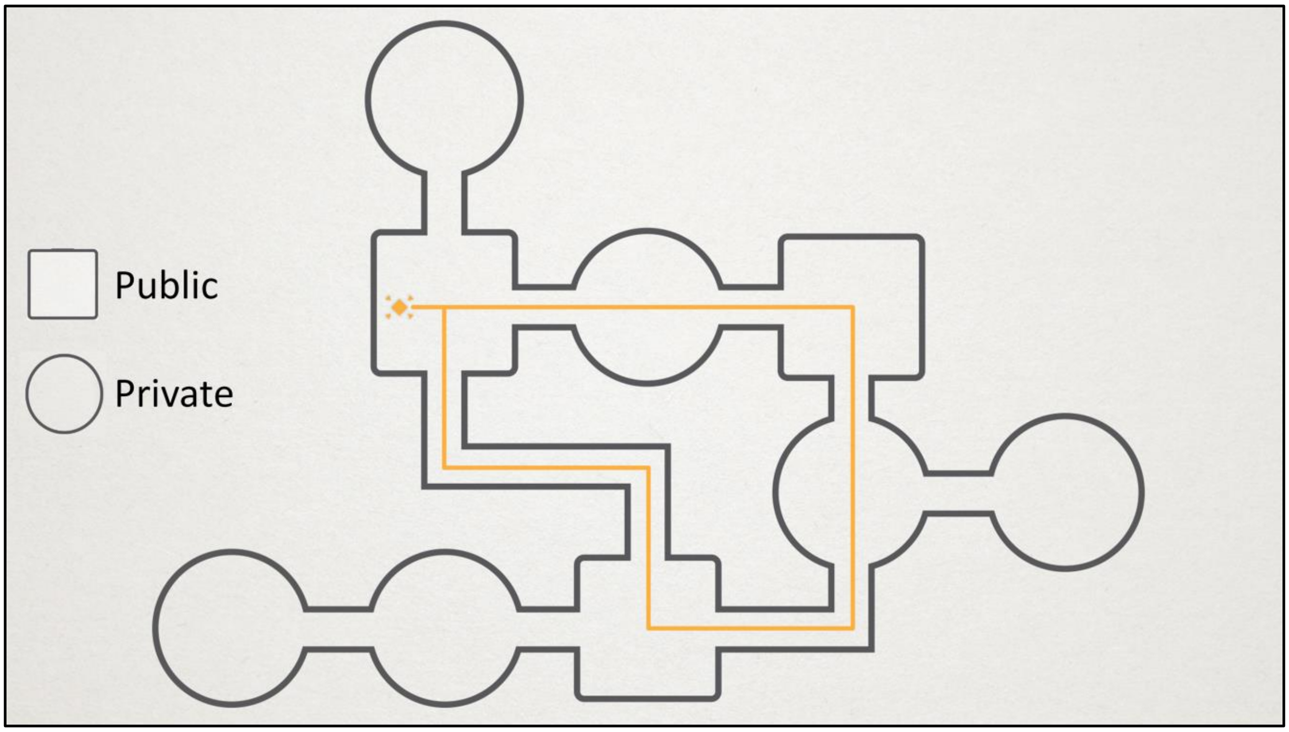
Private Bubbles are all reserved exclusively for your fireteam. No strangers ever show up there.

In MMO terminology, the private bubbles would be "instances", but that terminology isn't a perfect fit, since *all* of our bubbles are instances (there could be thousands copies of a single public bubble, each connecting up to 6 strangers).

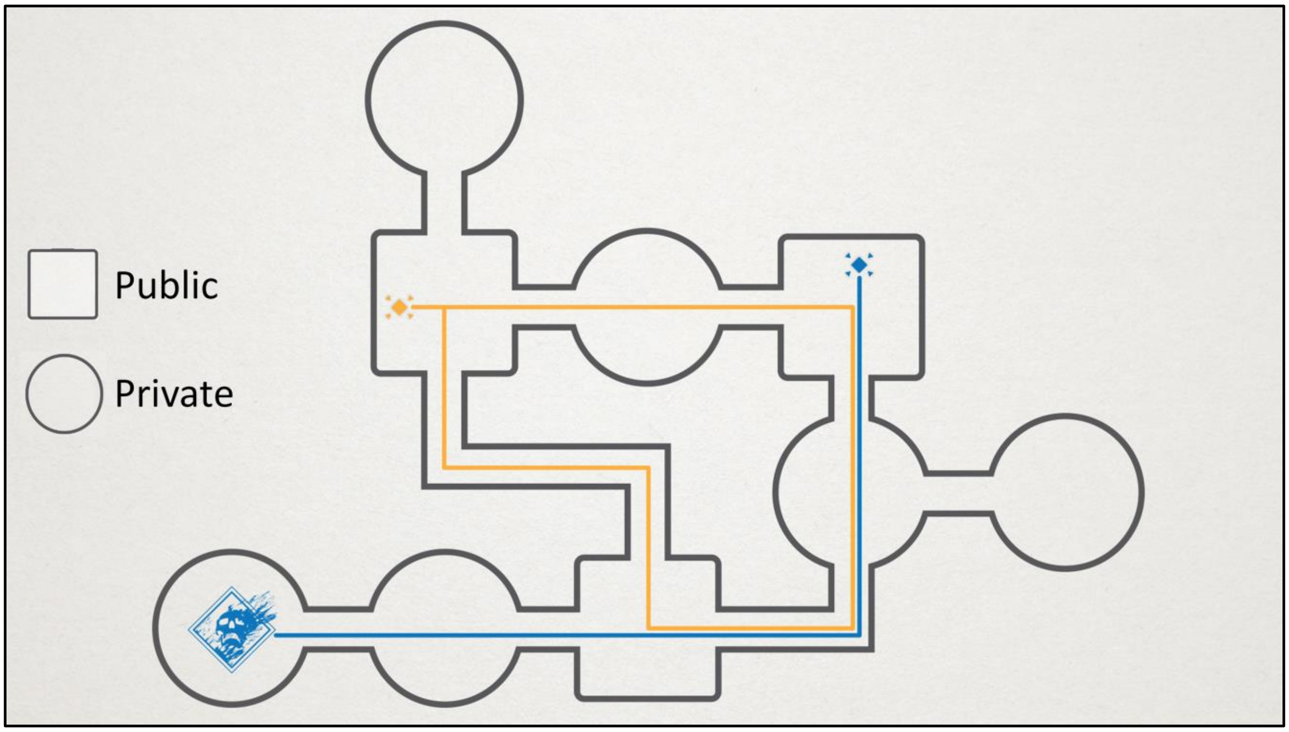


So here's an idea of how we layout one of our planets in Destiny (we call them "Destinations").

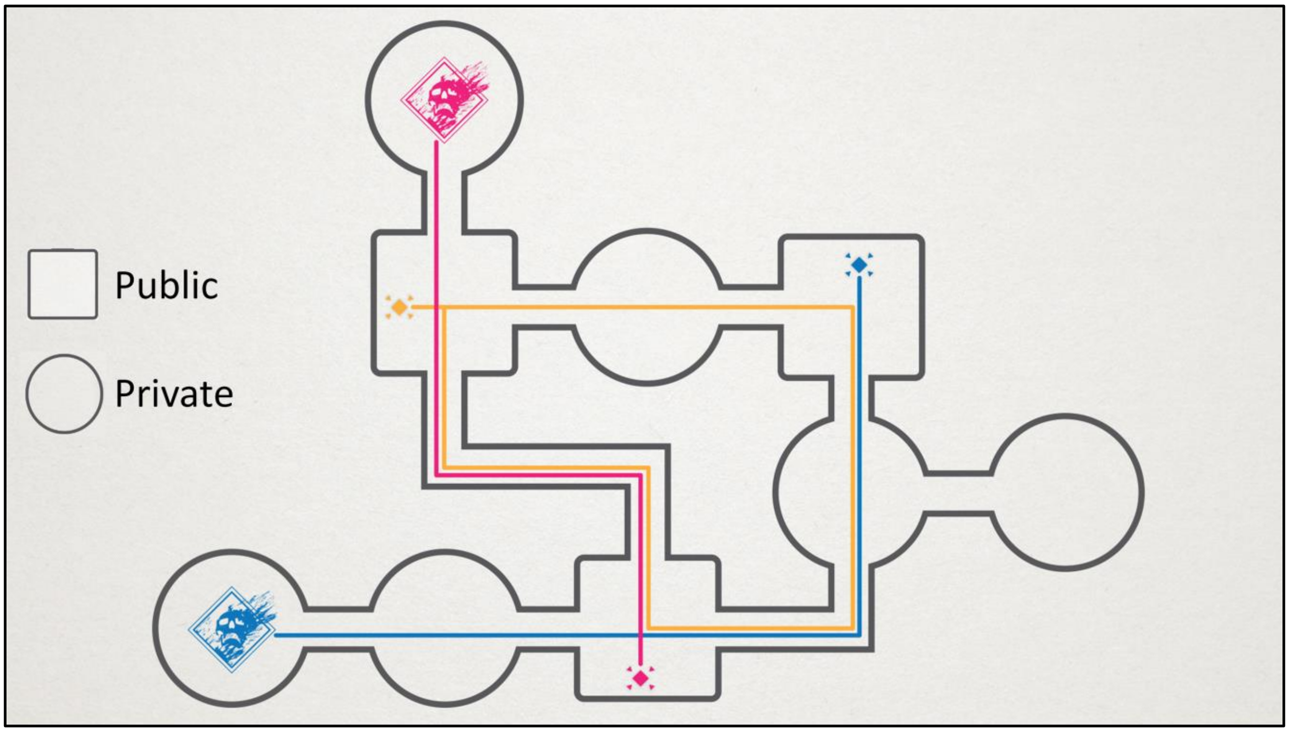
The Squares in this diagram are public bubbles, Circles are private bubbles.



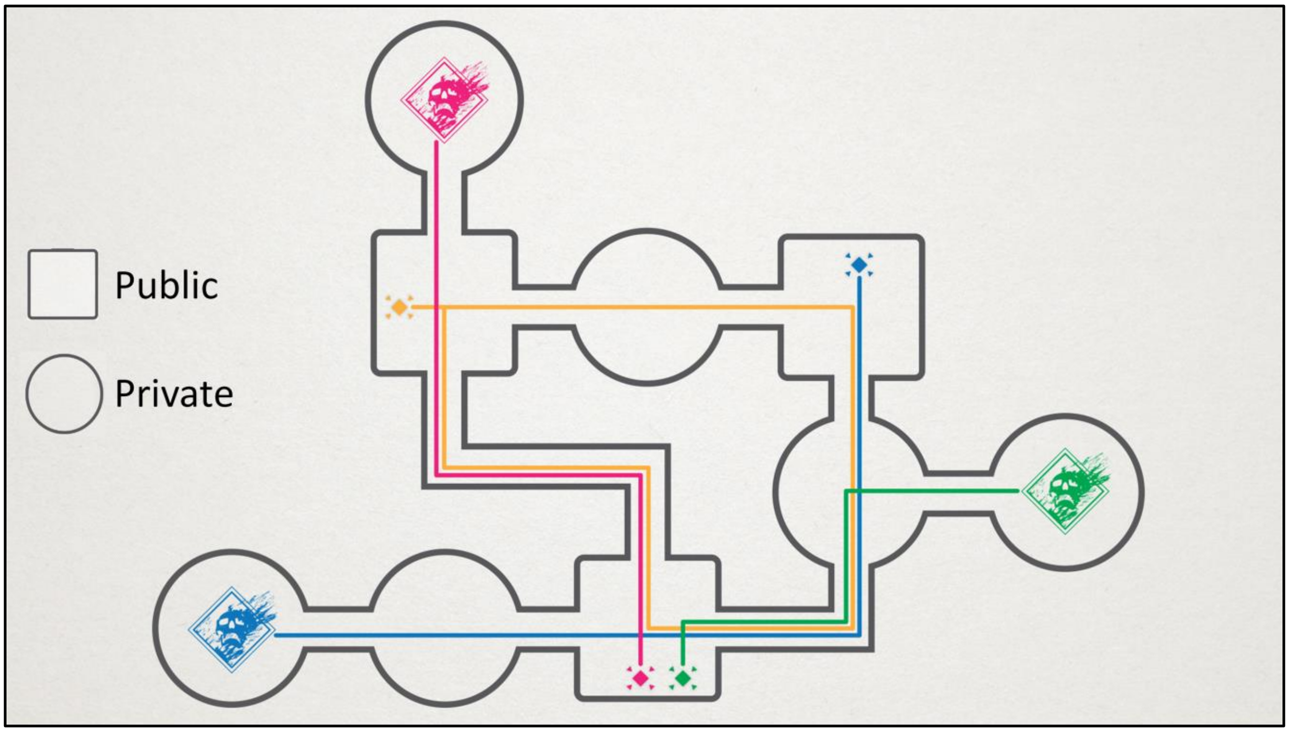
You can see how there's a "loop" with the public bubbles – a fairly easy way for you to roll around in a Patrol activity, moving predominately between Public Bubbles, opening treasure chests while interacting with strangers.



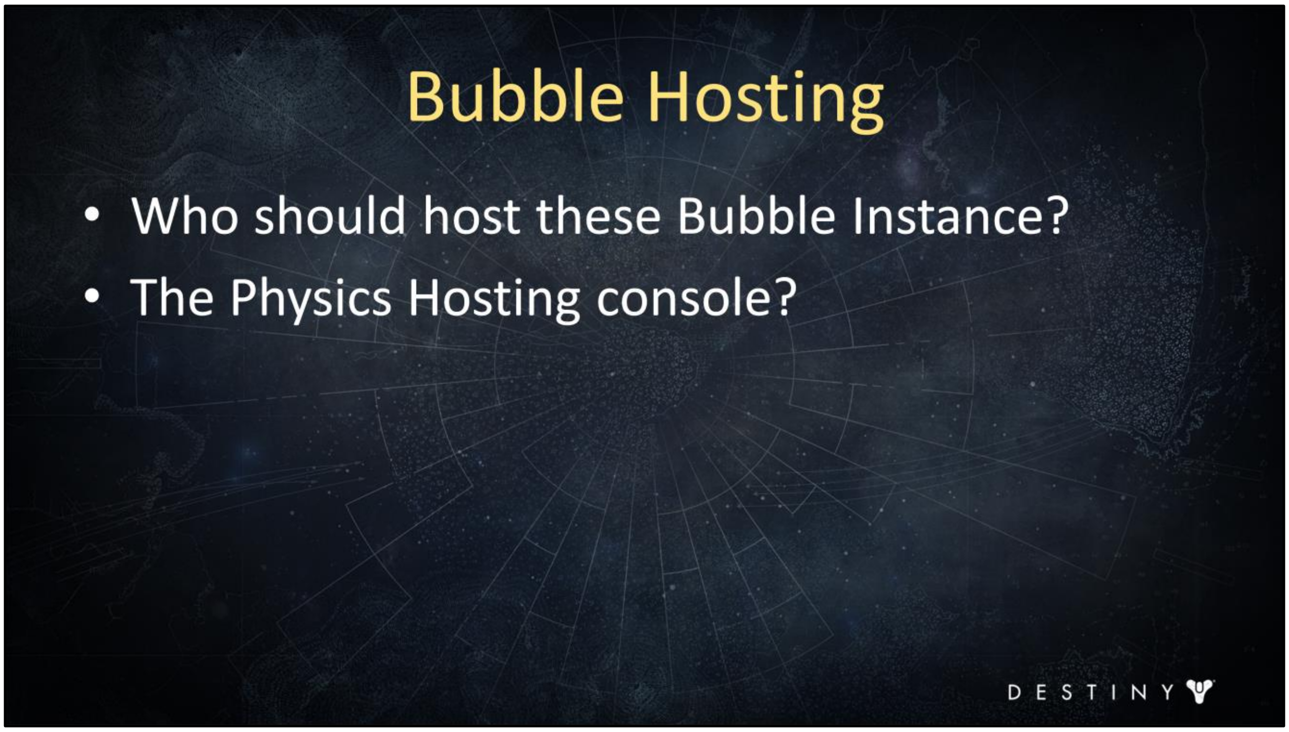
Now if I add some other activity lines – most of our activities start in a public bubble, and they all take you on a deliberate path through many bubbles. You typically get some exposure to 1-2 public bubbles during an activity line, and then dive into a private-bubble chain where we can have a hand-authored mission climax.



Now, as I continue to layer on the various activities that strangers can be on in the same bubble,



the intent is that when you encounter folks in a public bubble, they’re likely to be “intersecting lines”, passing by but having different goals and direction of travel.



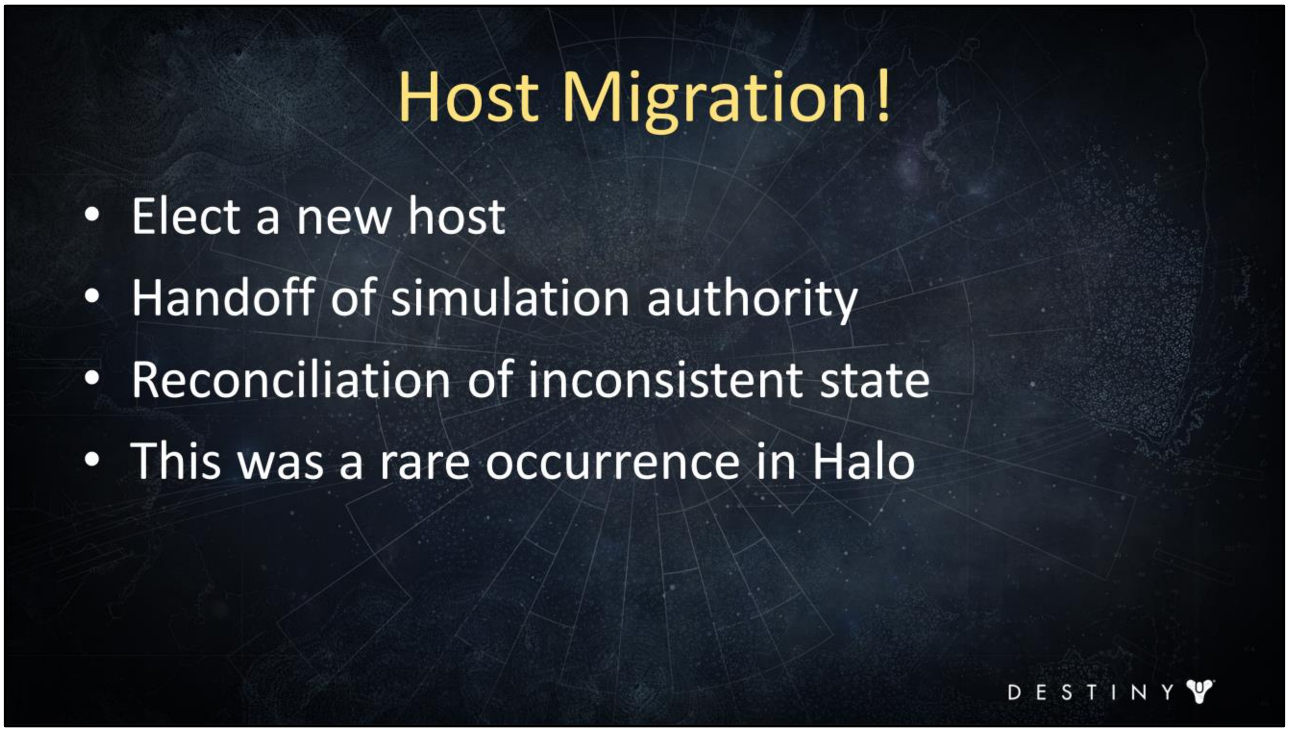
Bubble Hosting

- Who should host these Bubble Instance?
- The Physics Hosting console?

So, I mentioned that every time you enter a new public bubble, we matchmake you to a new Host. So who should be responsible for hosting each of these bubbles?

We could make it the traditional Halo-Reach era “Physics Host” – whichever console was elected to be the authoritative arbiter of events.

We decided *not* to simply use the Physics Hosting console. And to help explain why, we need to talk about Host Migrations

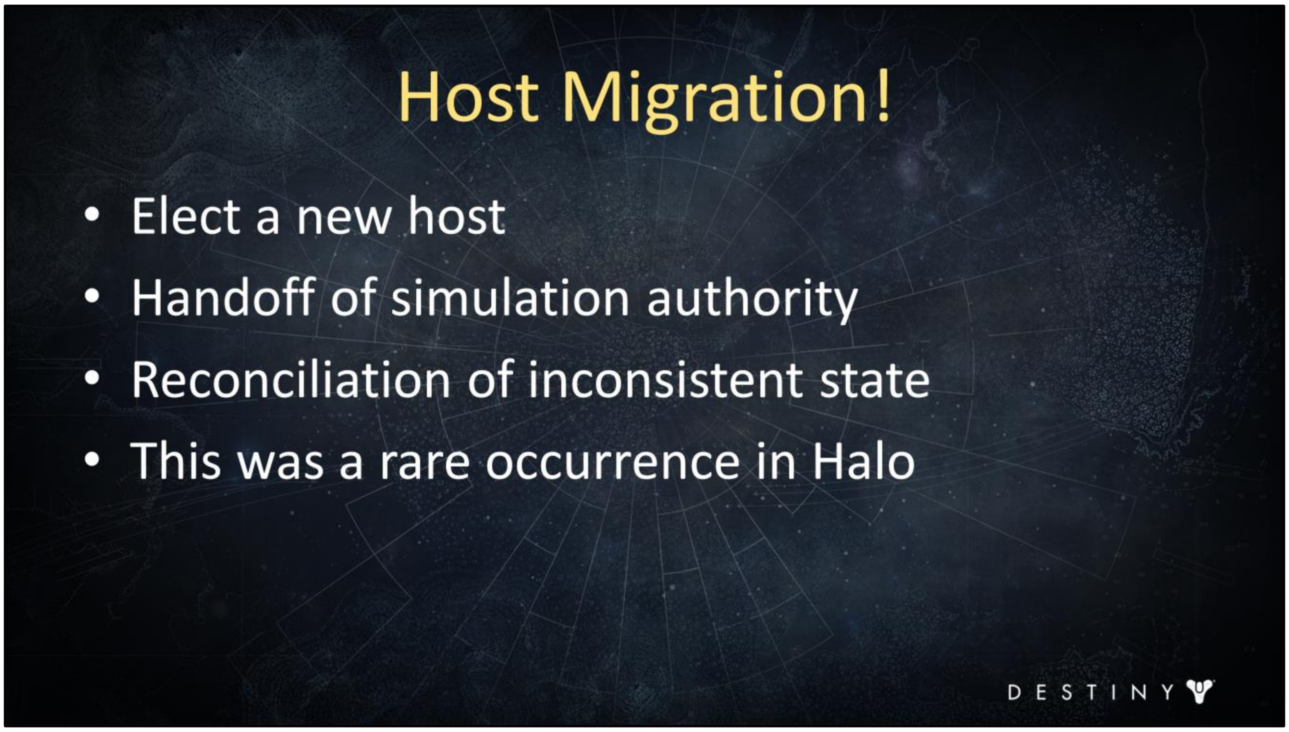


Host Migration!

- Elect a new host
- Handoff of simulation authority
- Reconciliation of inconsistent state
- This was a rare occurrence in Halo

A Host Migration occurs whenever the Physics Hosting console disconnects from the game. In order to keep the game running, we need to elect a new Physics Host from one of the remaining players.

We pick a new host, and then need to get a new authoritative simulation state from the new host.

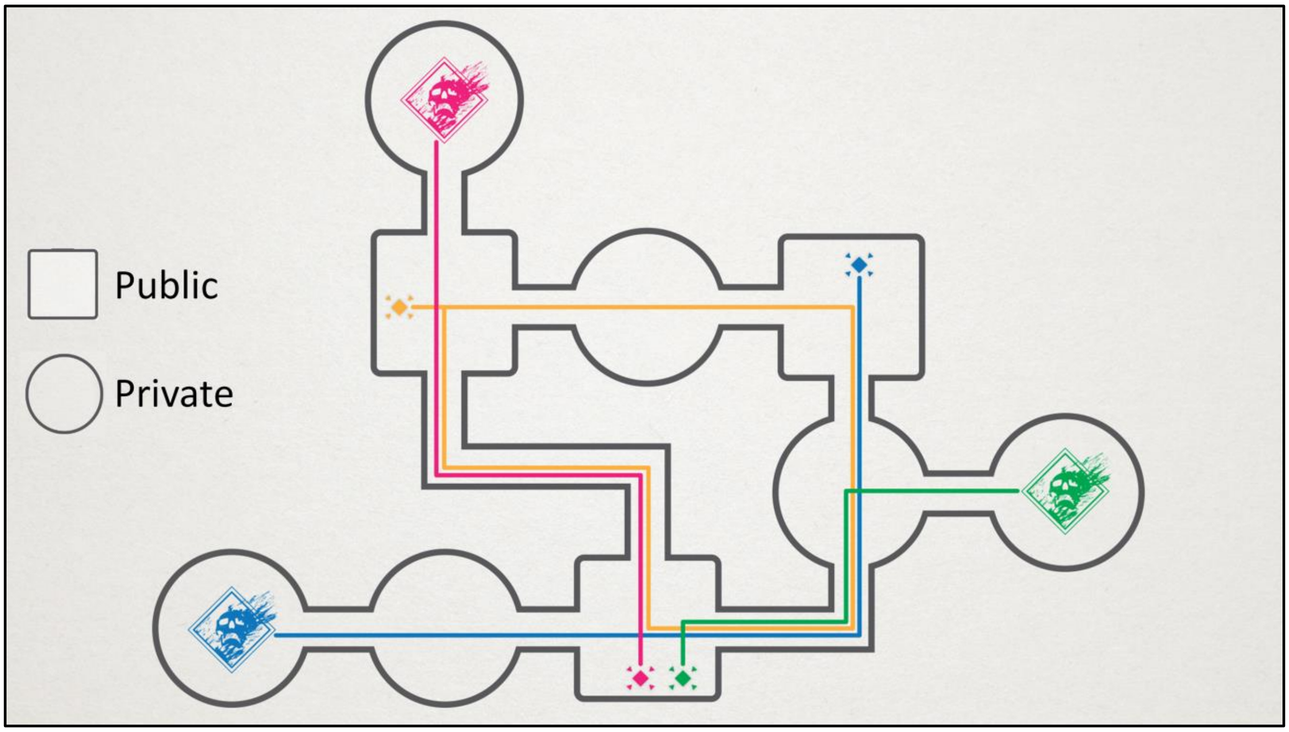


Host Migration!

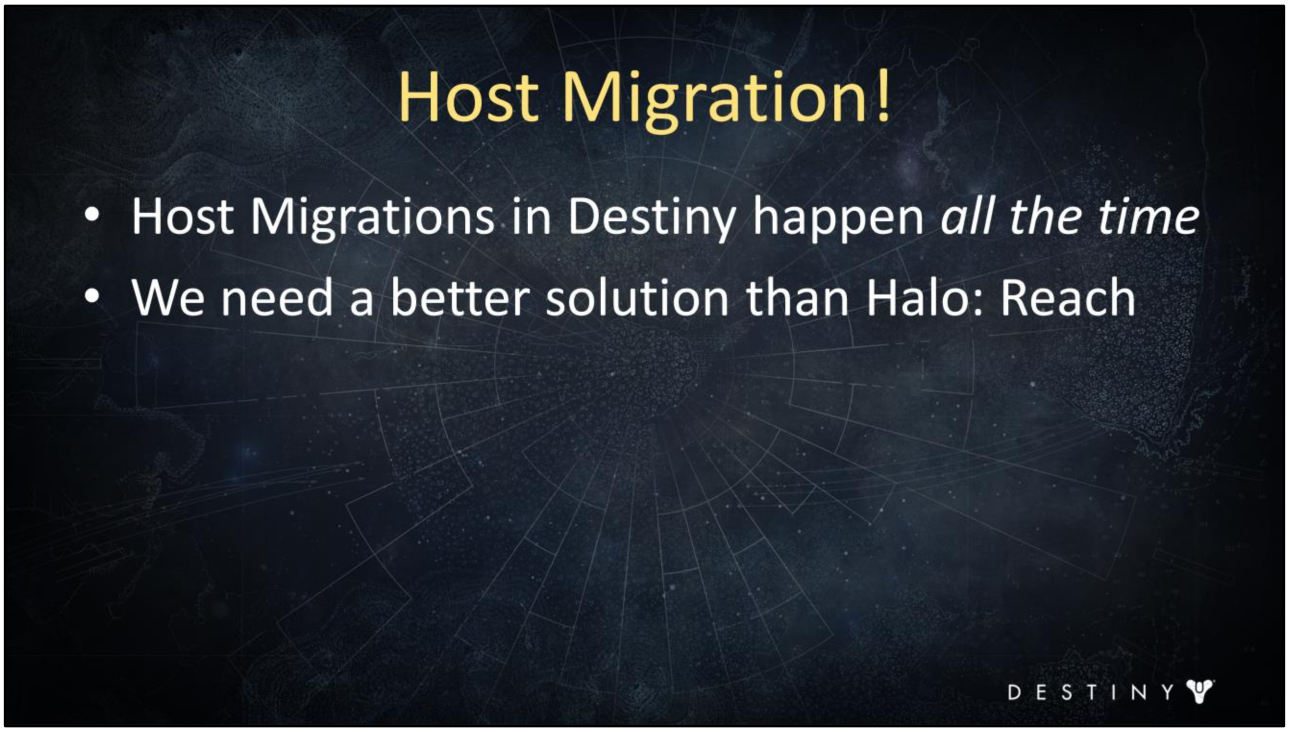
- Elect a new host
- Handoff of simulation authority
- Reconciliation of inconsistent state
- This was a rare occurrence in Halo

In Halo: Reach, This was a pretty abrupt experience for players, always caused a black screen for a few seconds, and could result in inconsistent state.

It was ok that this experience was a little sucky, because it was reasonably rare – not only did we tie all PvP rewards to finishing a match, we even temporarily banned players who quit PvP too often. So host migrations did not typically interrupt the player experience.



But Destiny is a world of intersecting, not parallel lines. Most of the time, players are just passing through public bubbles. Any one of these players could be the current Physics Host, and as soon as she leaves and deinstantiates the bubble, we need to Host Migrate to a new physics host.

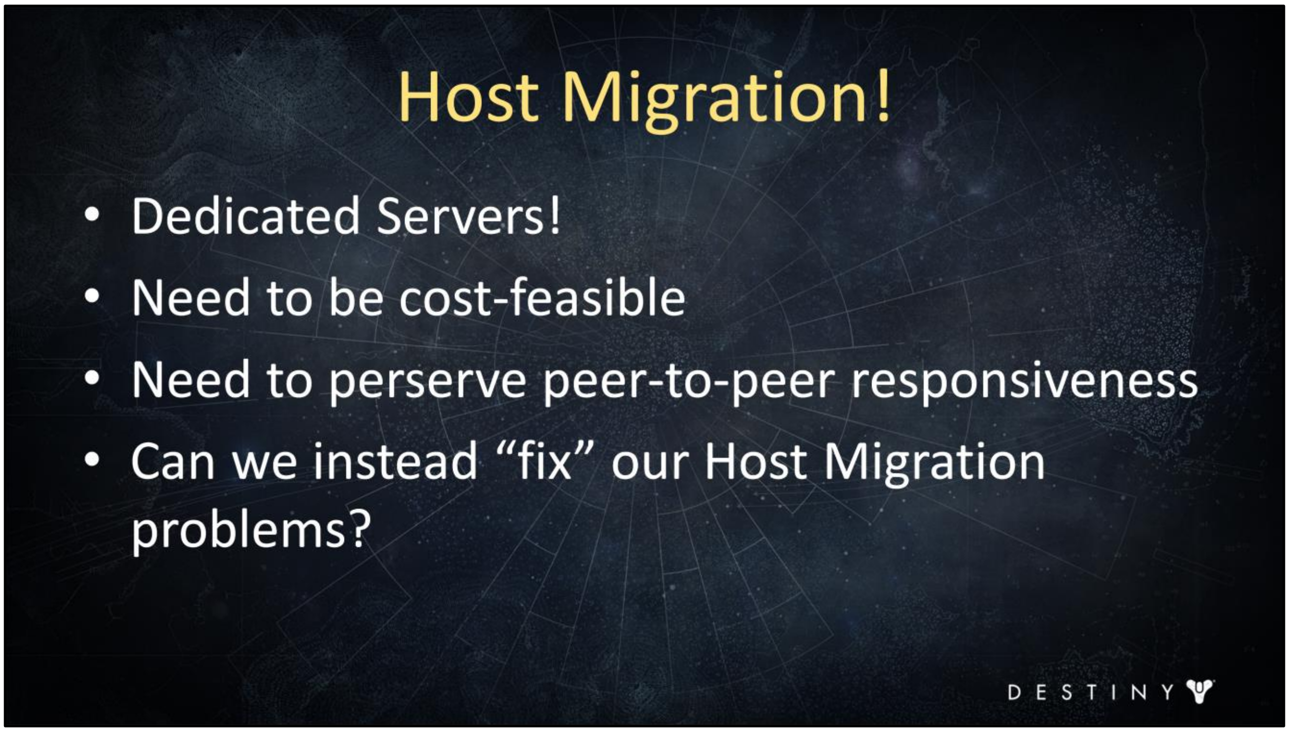


Host Migration!

- Host Migrations in Destiny happen *all the time*
- We need a better solution than Halo: Reach

So in most Destiny public bubbles, Physics Host Migrations happen *all the time*. The average player in a public bubble experiences a Host Migration every *160 seconds* - one every 2 and a half minutes.

We knew we needed a better solution, that didn't create a black screen load, and hopefully didn't cause obvious player-facing artifacts.



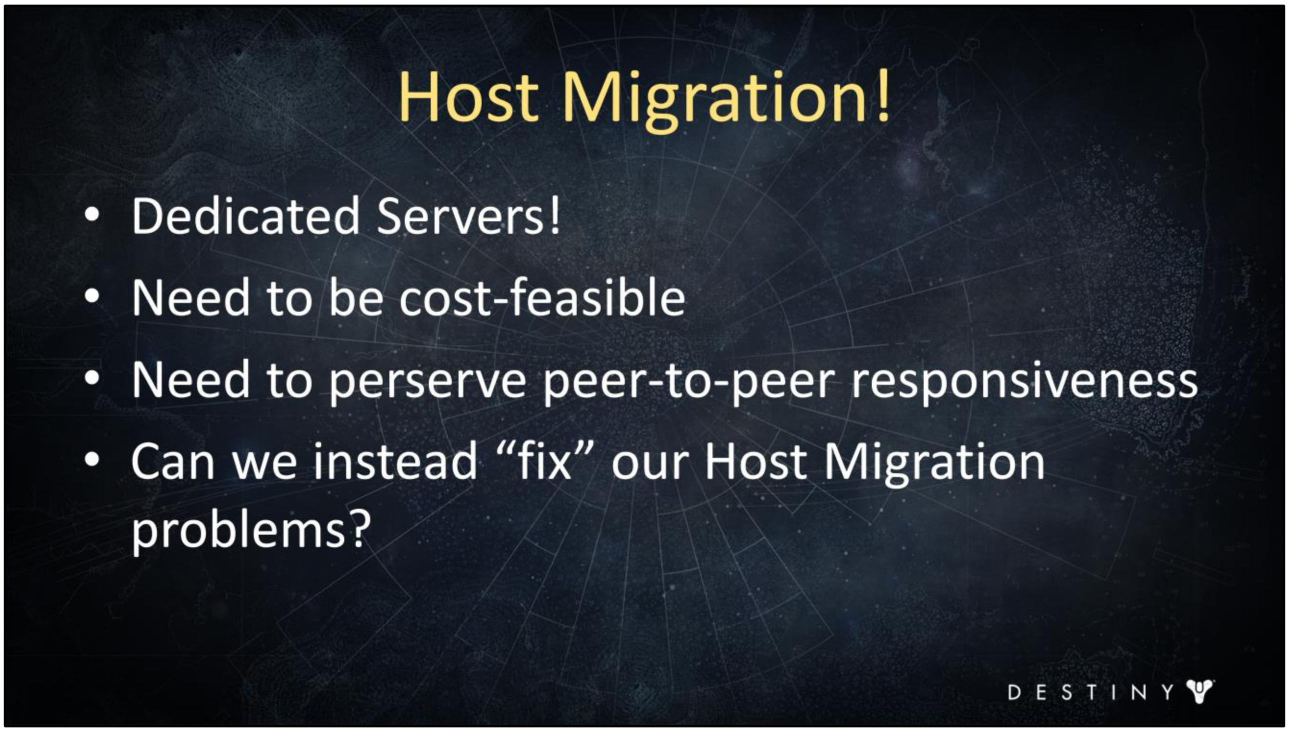
Host Migration!

- Dedicated Servers!
- Need to be cost-feasible
- Need to perserve peer-to-peer responsiveness
- Can we instead “fix” our Host Migration problems?

At this point, I’m sure some of you are thinking – just use dedicated servers! If we just never Host Migrate, because we put all our Physics Hosts in the cloud, we never have to solve all these pesky problems.

There’s a couple strong reasons we didn’t simply run dedicate servers that were traditional Physics Hosts.

For one thing, they need to be cost-feasible. To support our launch, we’d have needed hundreds of thousands of headless PS3-Parity executables in the cloud, and that becomes a significant continuous cost to maintain, especially if our player-retention continues to stay as strong as it has.



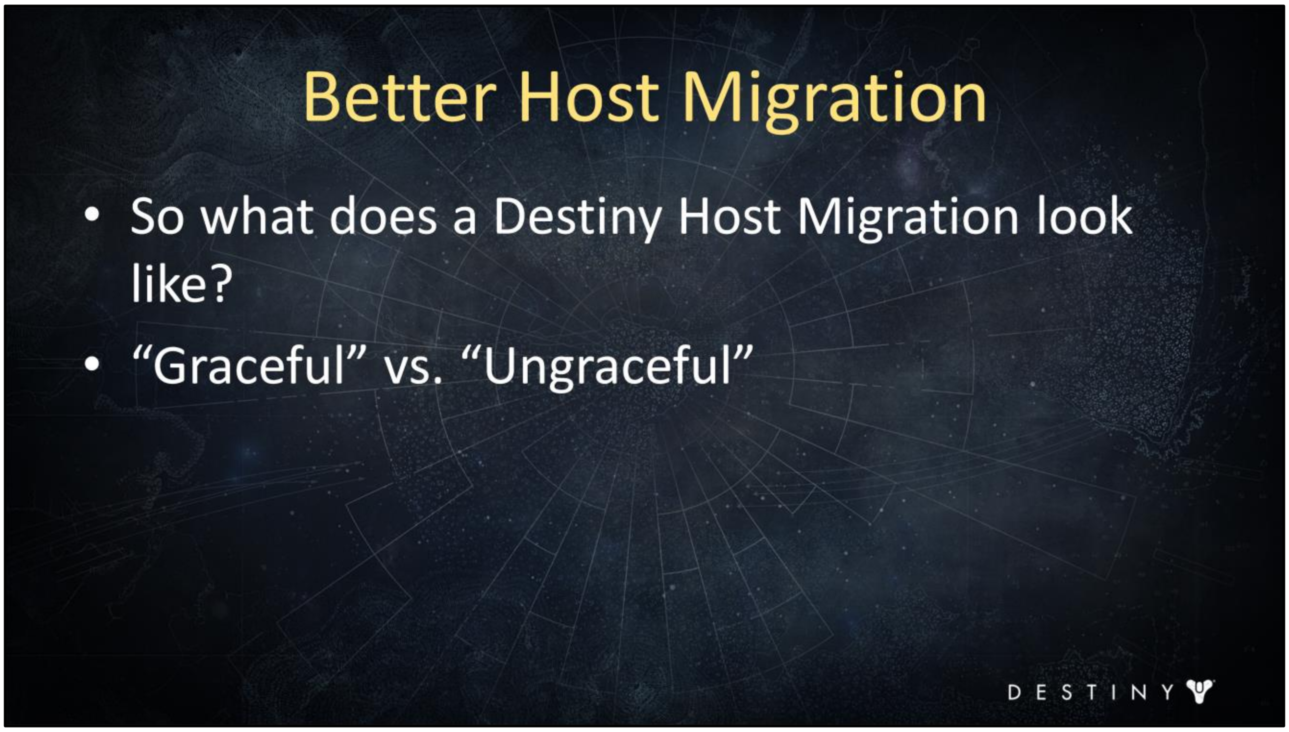
Host Migration!

- Dedicated Servers!
- Need to be cost-feasible
- Need to perserve peer-to-peer responsiveness
- Can we instead “fix” our Host Migration problems?

DESTINY

Additionally, peer-to-peer networking supports maximally responsive action gameplay. In many cases we can match you with players that are in the same city as you, and you get extremely low latency with your Physics Host – much better than what we could do with Dedicated Servers. We don’t want to increase our latency for firing bullets and doing damage – that violates our “Feels like a Single-Player Shooter” goal.

So – can we keep our traditional Physics Hosts around, but “fix” the Host Migration problems that Halo had?



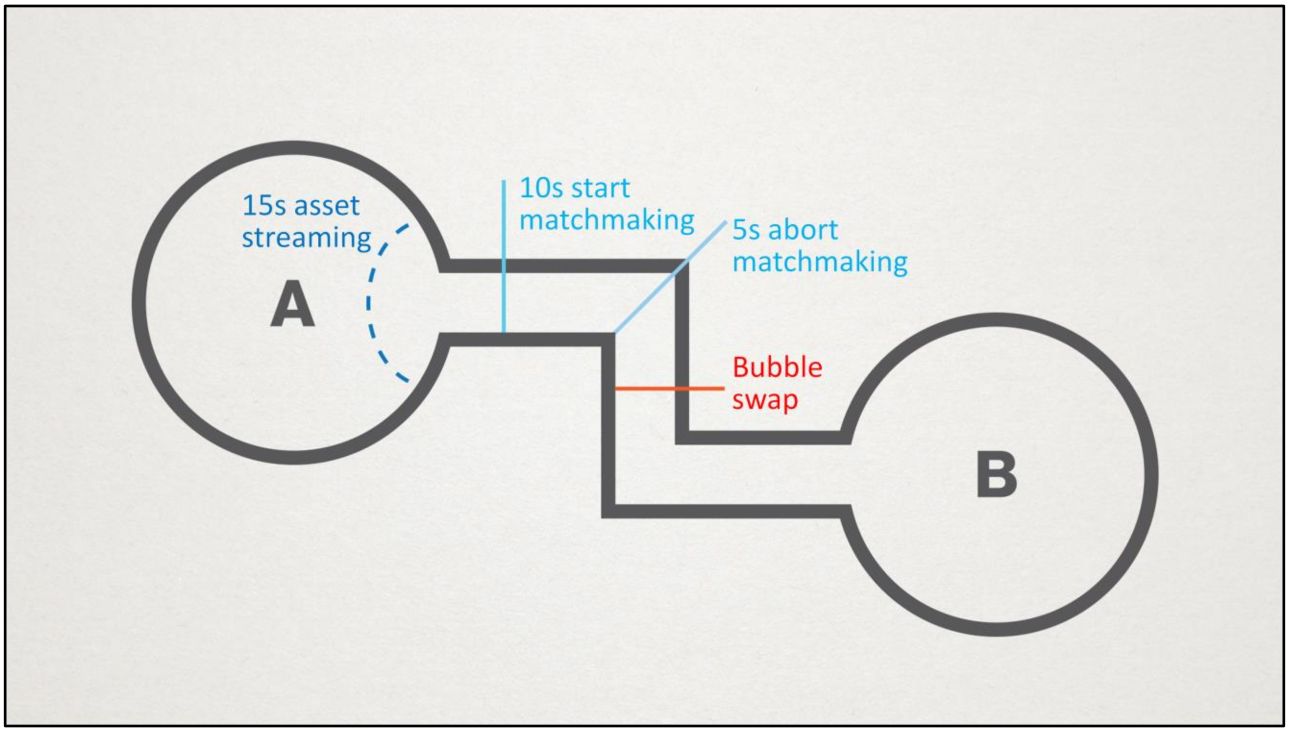
Better Host Migration

- So what does a Destiny Host Migration look like?
- "Graceful" vs. "Ungraceful"

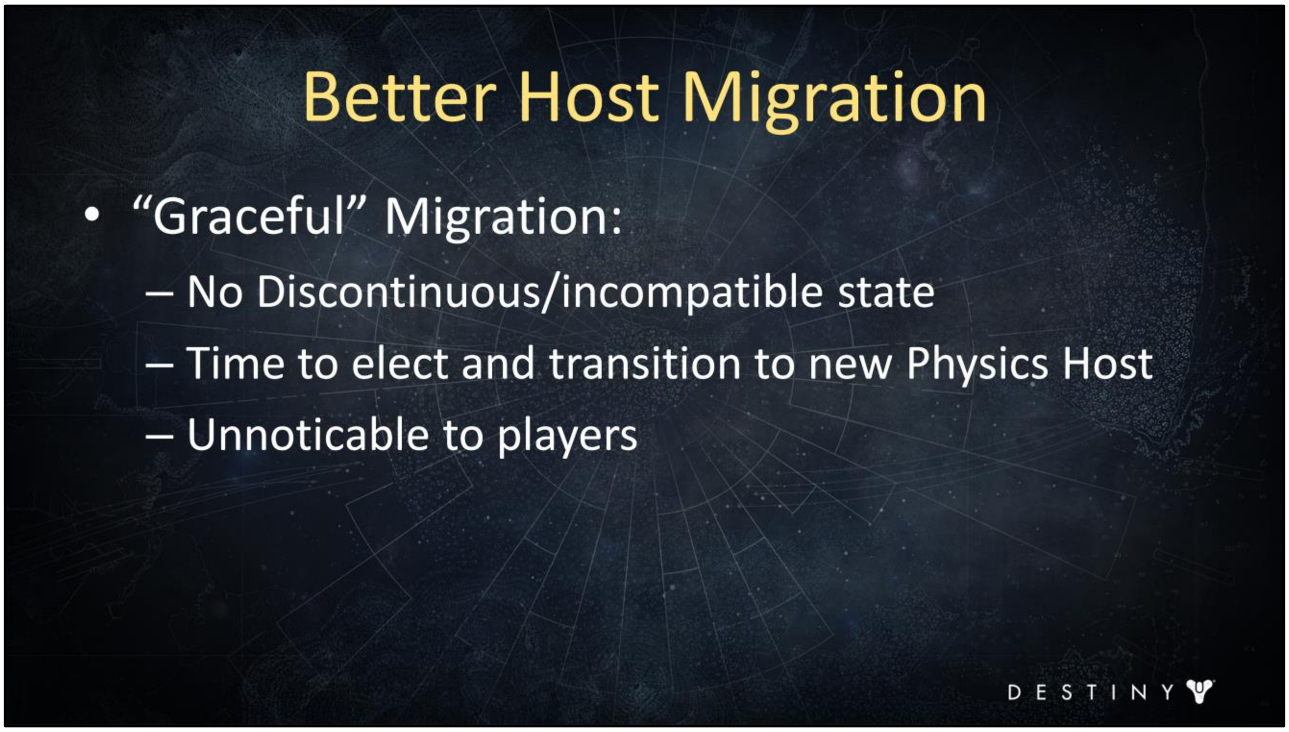
So, Halo: Reach host migrations were typically abrupt black screens - what does a Destiny Host Migration look like to players?

It depends. We can talk about 2 examples to begin with, what I'll call "Fully Graceful" and "Fully Ungraceful." This isn't a binary switch, but two poles on a continuum.

"Fully Graceful" would be our best case.



So, the current host knows he's about to transition to another bubble, as soon as he enters the transition interior. That gives us ~10s anticipation, during which we can elect a new host, and transfer ownership to them seamlessly.

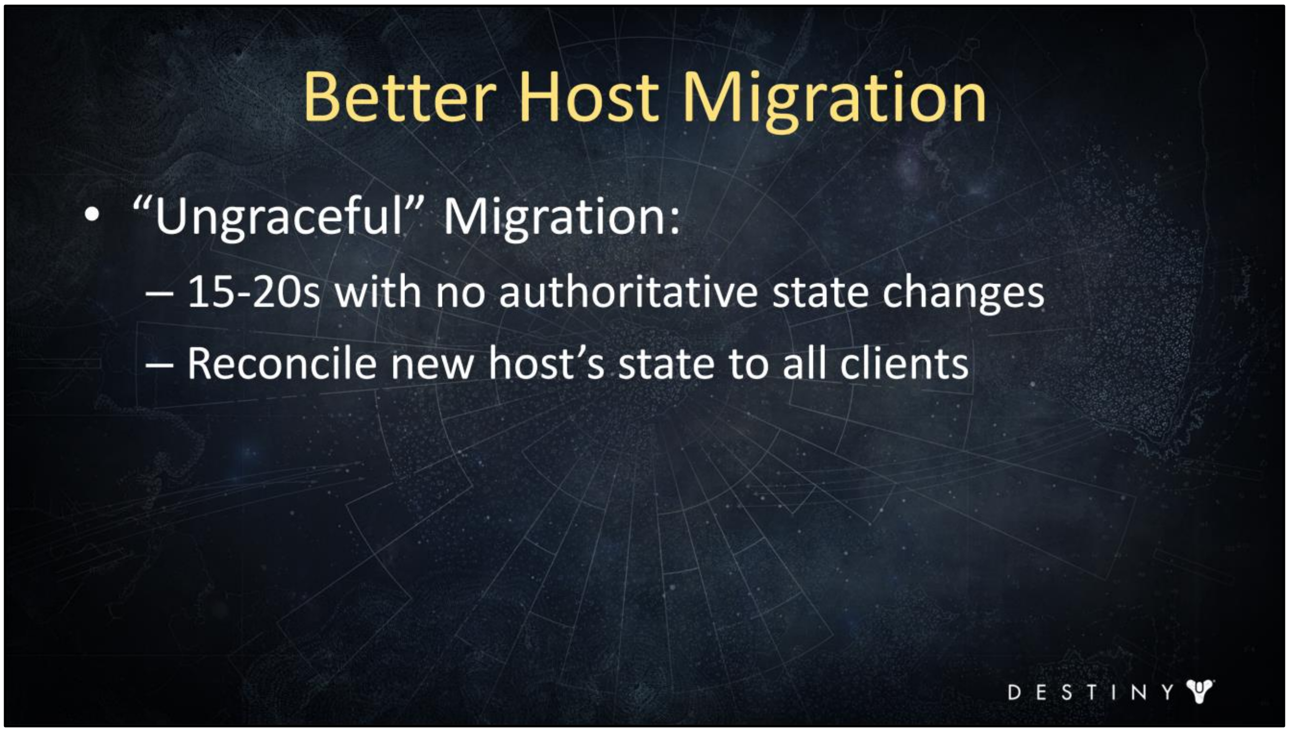


Better Host Migration

- "Graceful" Migration:
  - No Discontinuous/incompatible state
  - Time to elect and transition to new Physics Host
  - Unnoticable to players

In the best case, their current physics state is also fully compatible with the old host's state, so there's no need for discontinuous state, and players never notice that we switched physics hosts midway through combat.

This sort of host migration happens a lot during a typical player experience, and most of the time noone notices a thing.

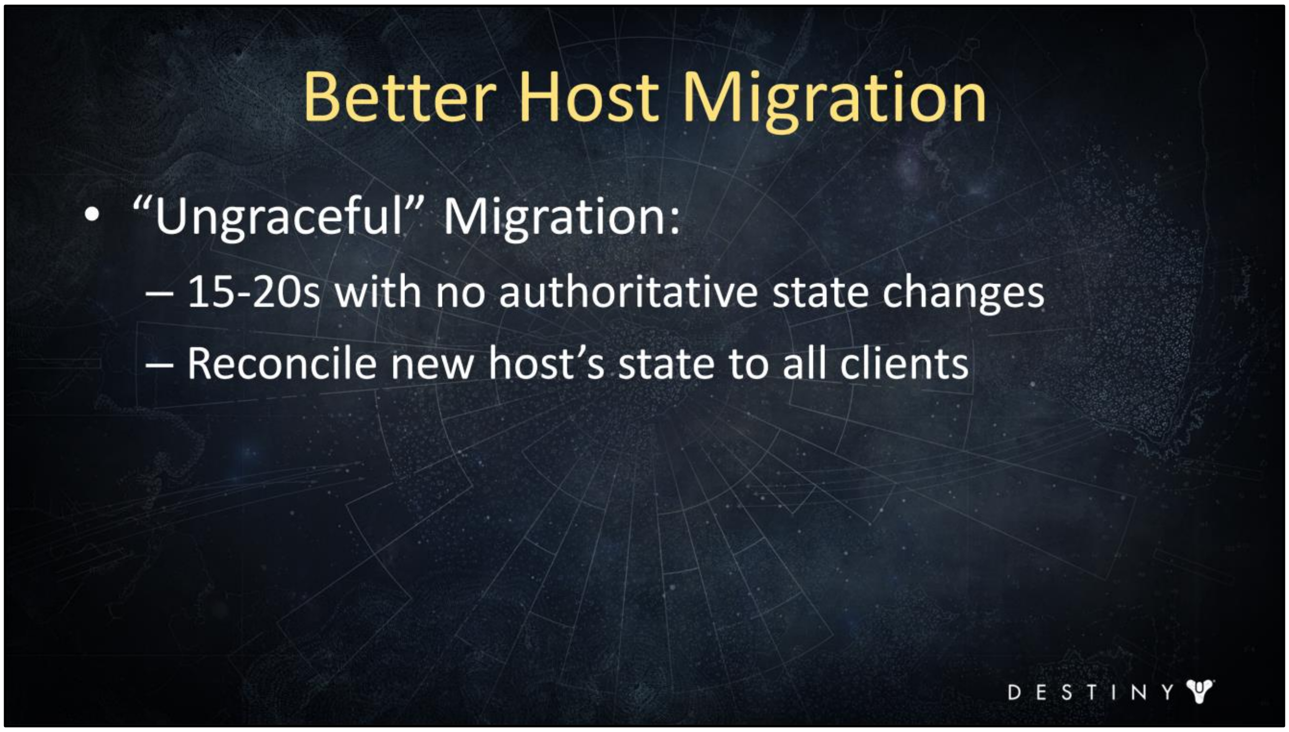


Better Host Migration

- "Ungraceful" Migration:
  - 15-20s with no authoritative state changes
  - Reconcile new host's state to all clients

On the other end, you could imagine our worst-case ungraceful host migration. The physics host pulls their ethernet cable, and we stop receiving any packets from them.

We have pretty lenient timeouts (because we're always online, and don't want to be constantly kicking people out of the gameworld if they have a bad connection). So it could be 15-20s before we fully timeout the old host.

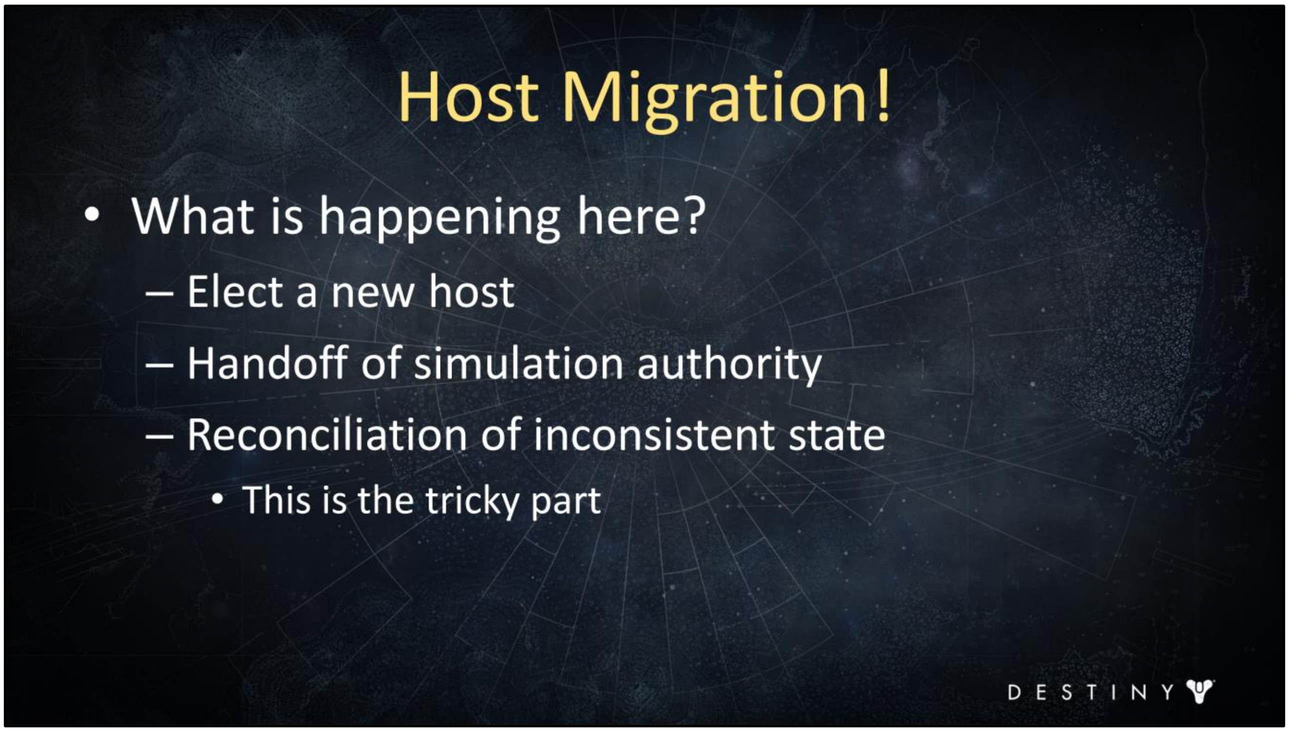

Better Host Migration

- "Ungraceful" Migration:
  - 15-20s with no authoritative state changes
  - Reconcile new host's state to all clients

DESTINY

During that time, you'll be locally predicting state changes, but they won't ever apply anything authoritative. So you'll see predictive health bar damage, but be unable to kill any AI, and your public events won't advance in any significant way.

Eventually, we'll elect a new physics host, and public events and AI death etc. will start functioning again.



Host Migration!

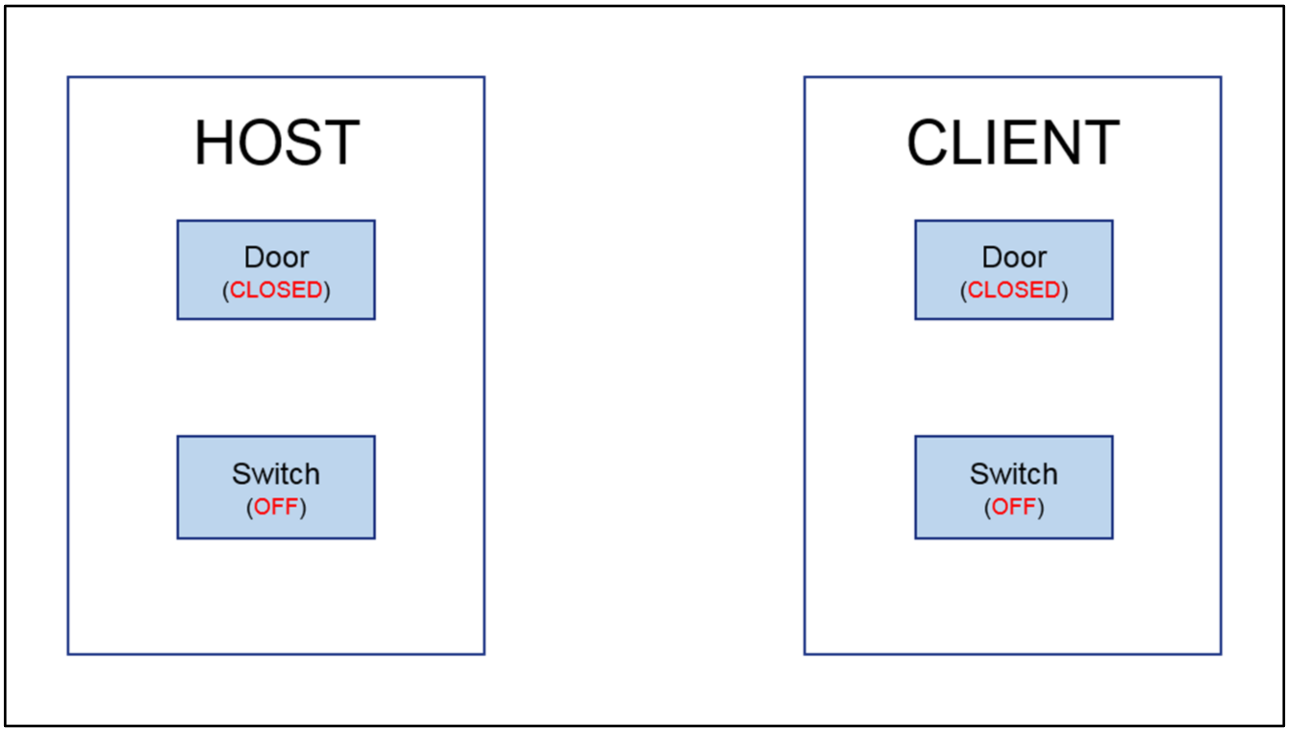
- What is happening here?
  - Elect a new host
  - Handoff of simulation authority
  - Reconciliation of inconsistent state
    - This is the tricky part

DESTINY

So, that’s the player experience, but what’s going on under the hood? We obviously need to elect a new host to hand off all of our simulation authority.

But that new host’s simulation state will not be identical to the old host. Not only may objects be in slightly different states, she might not have all the same objects instantiated as the old host. She may have prematurely deleted some objects.

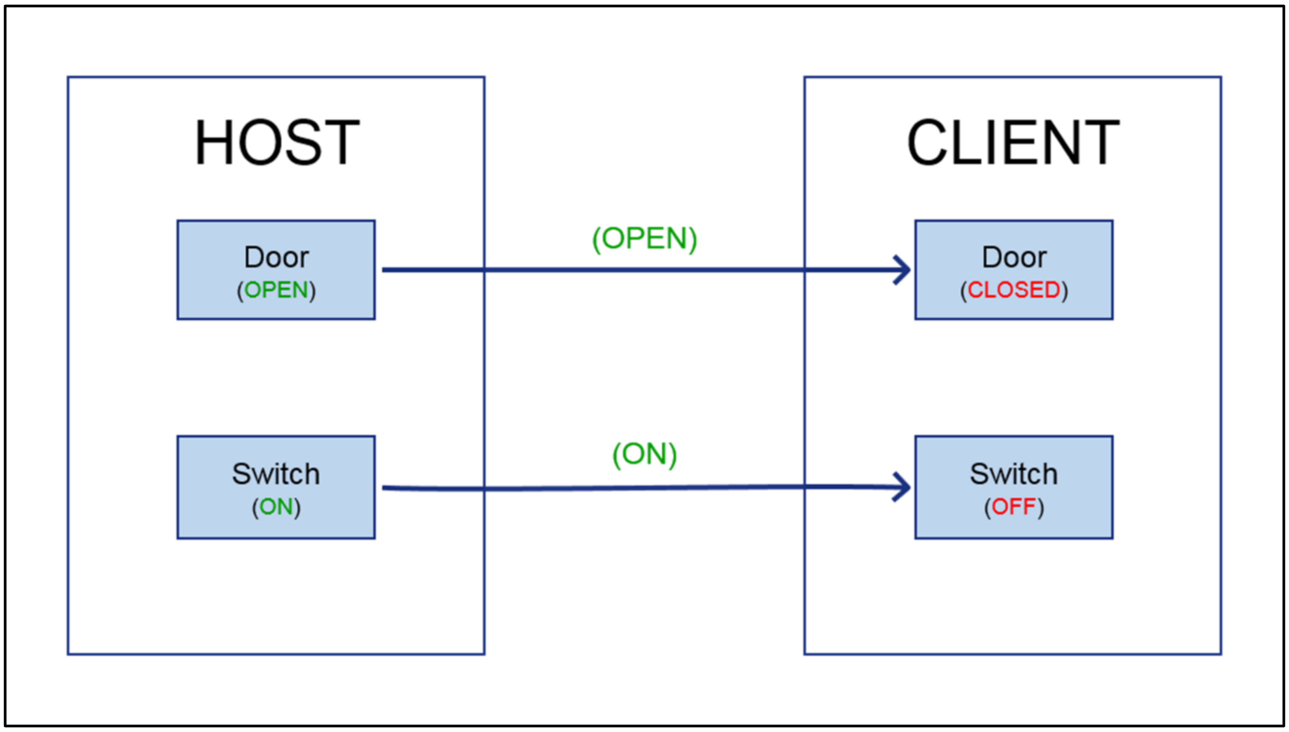
And this can cause some pretty severe experiential bugs – in the worst case, it can completely break activity script progression.



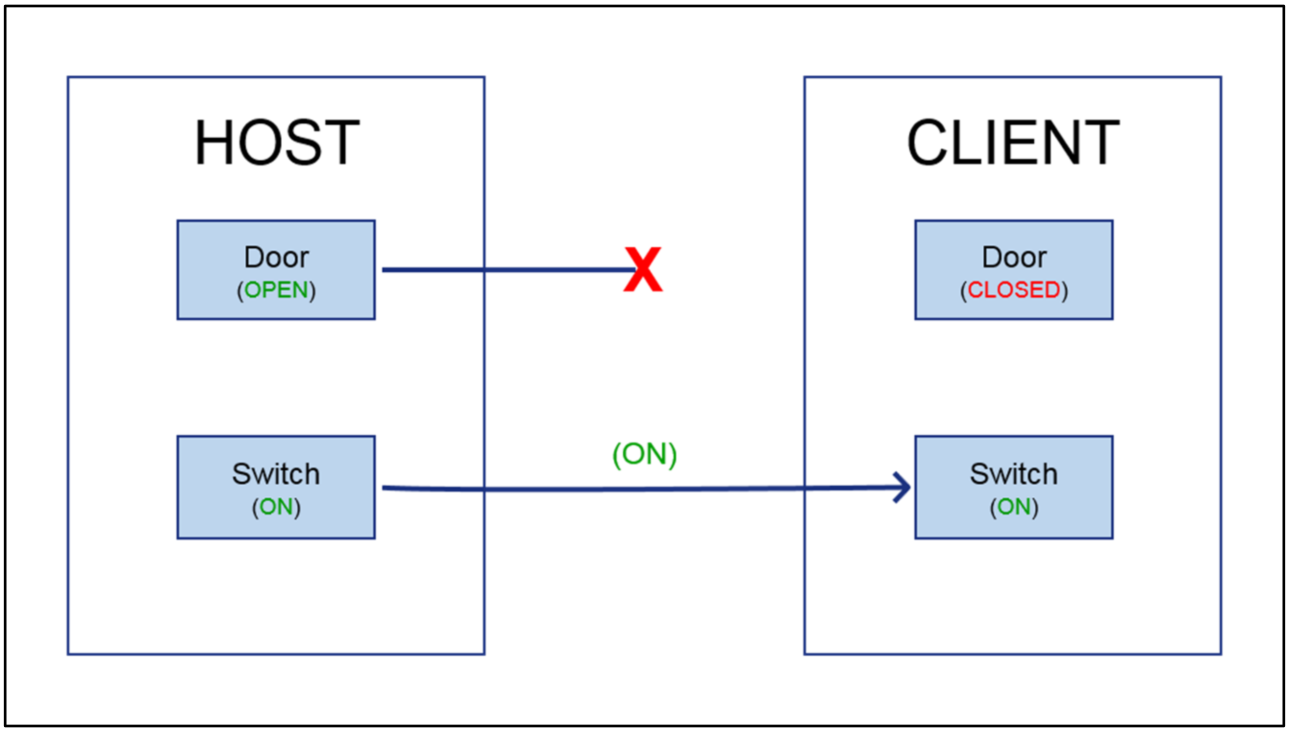
So let's imagine a really simple case from PvE scripting. You have a door that is scripted to "Open" when a player presses a switch.

These are both separate objects, that each track a separate boolean state (Open/Closed for the door, On/Off for the switch).

Crucially, because these objects are separate and unattached (outside of the script logic itself), they are *networked separately*. Destiny Bubbles have many dozens of complex objects all being simultaneously networked, competing for traffic. By networking each object separately, we can heavily prioritize our networking traffic to emphasize the objects that are most important to the local player. We've been doing that since the Halo days.

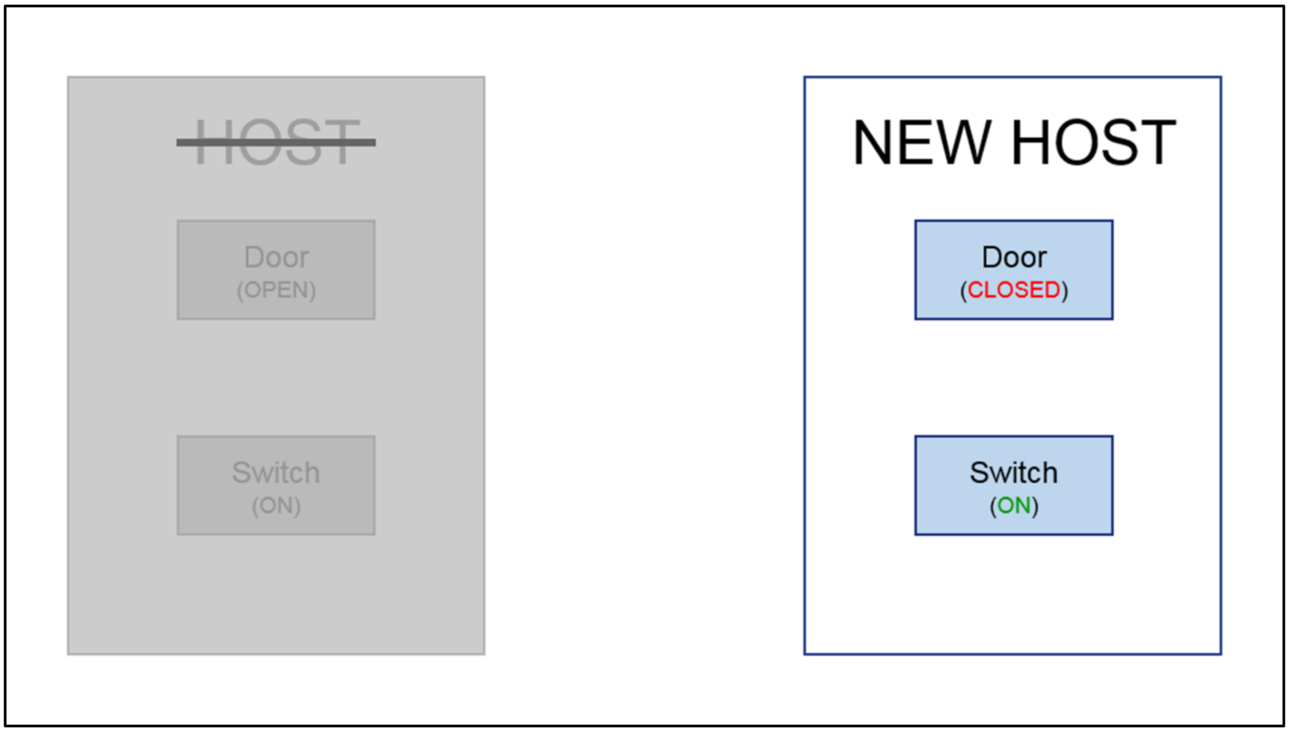


So, let's say the Physics Host goes over to the door, presses the switch, which opens the door. He then immediately pulls out his ethernet cable.

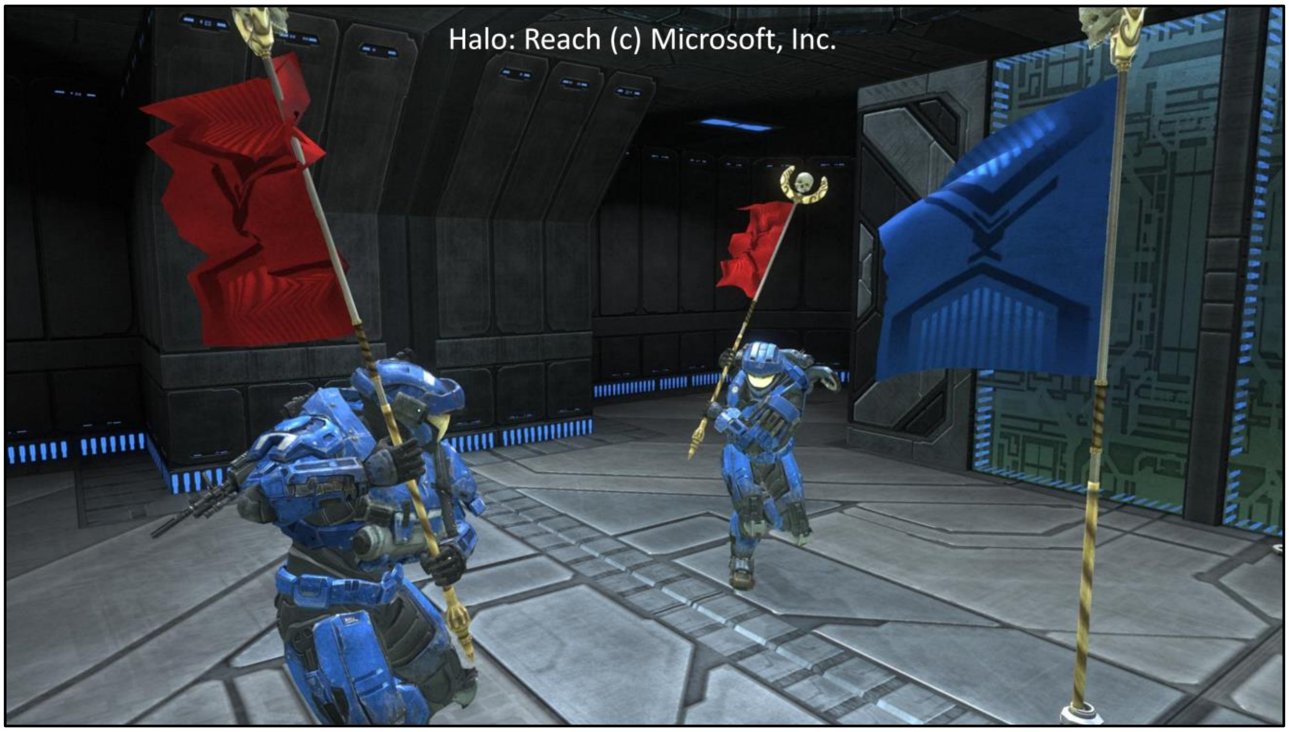


It’s entirely possible that all Clients (and whomever we elect as the new host) could receive the packet saying “Set Switch state to “ON”, but never receive the packet saying “Set Door state to “OPEN”

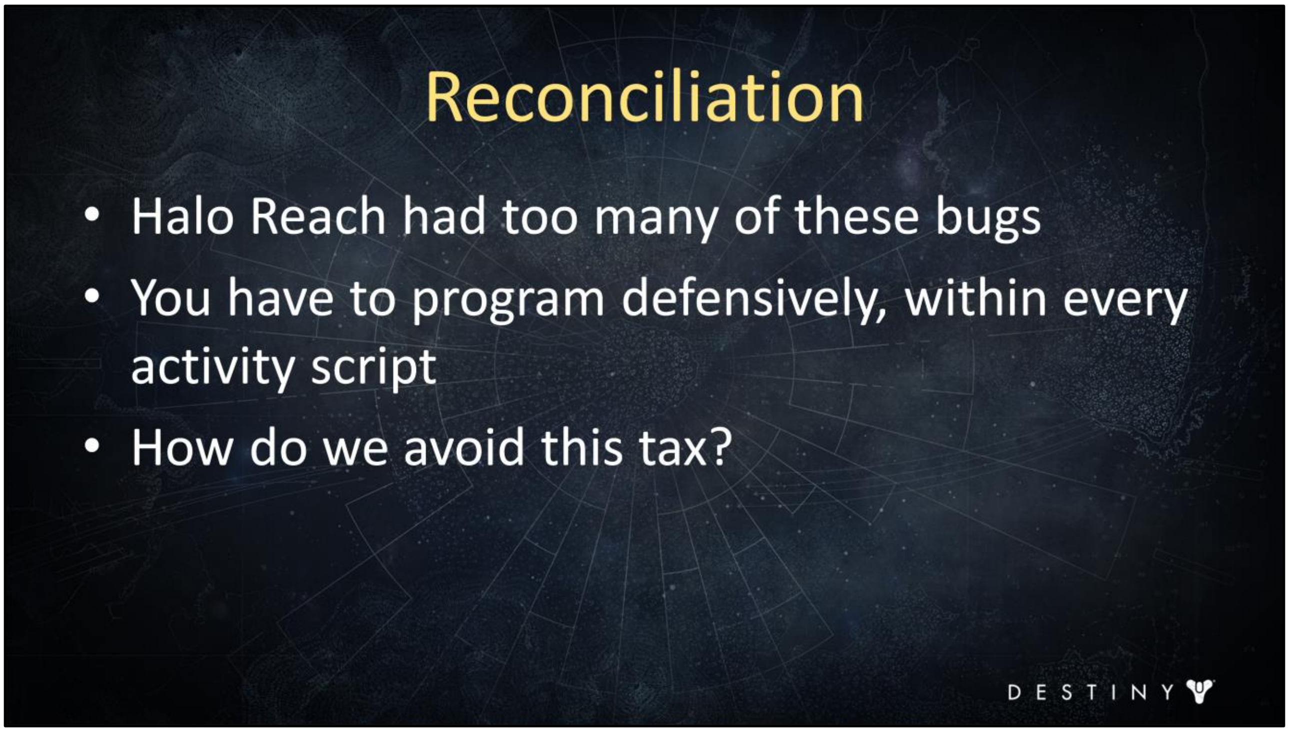
That packet could just get lost through ordinary packet loss, and the host is no longer available to resend it.



In that case, you now have a new host, with a switch that has already been pressed, but a door that is still closed, and you've now got a critical path progression blocker.



And for what it’s worth, this case is very real – Halo Reach had to deal with lots of inconsistent state like this during host migrations, it could easily break games like Capture the Flag. This is a bug screenshot I grabbed from Halo Reach – a Host Migration caused the script to spawn duplicate flags and break the score.



Reconciliation

- Halo Reach had too many of these bugs
- You have to program defensively, within every activity script
- How do we avoid this tax?

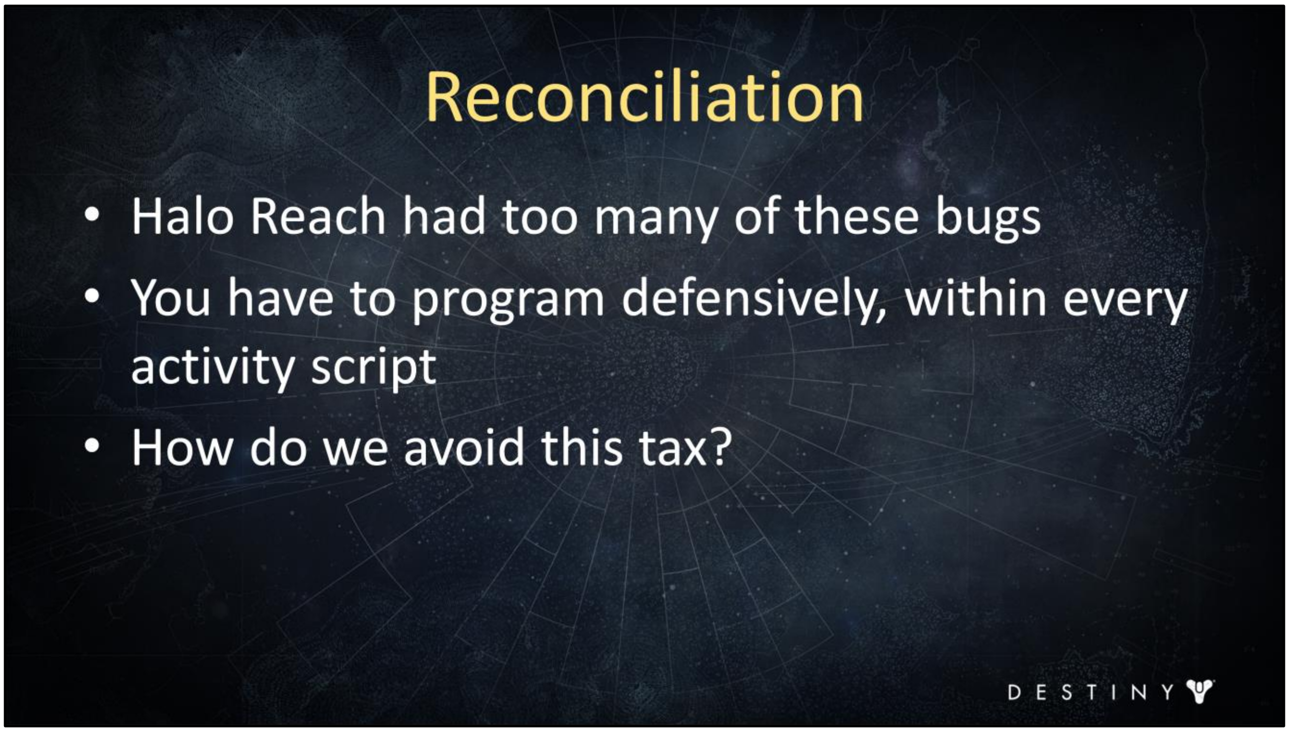
DESTINY

So, there's a couple ways you could imagine fixing this problem.

One is to program defensively – the CTF game ensures there's never more than one flag of a given team color. The door switch resets itself off after a few seconds, so you can reuse it if necessary.

But you have to think about each of those cases as you're writing each activity script, and it's hard to catch them all.

Plus, Host Migrations are frequently not covered in the standard development testing and iteration, so you could get very close to shipping before you find all these weird edge cases.

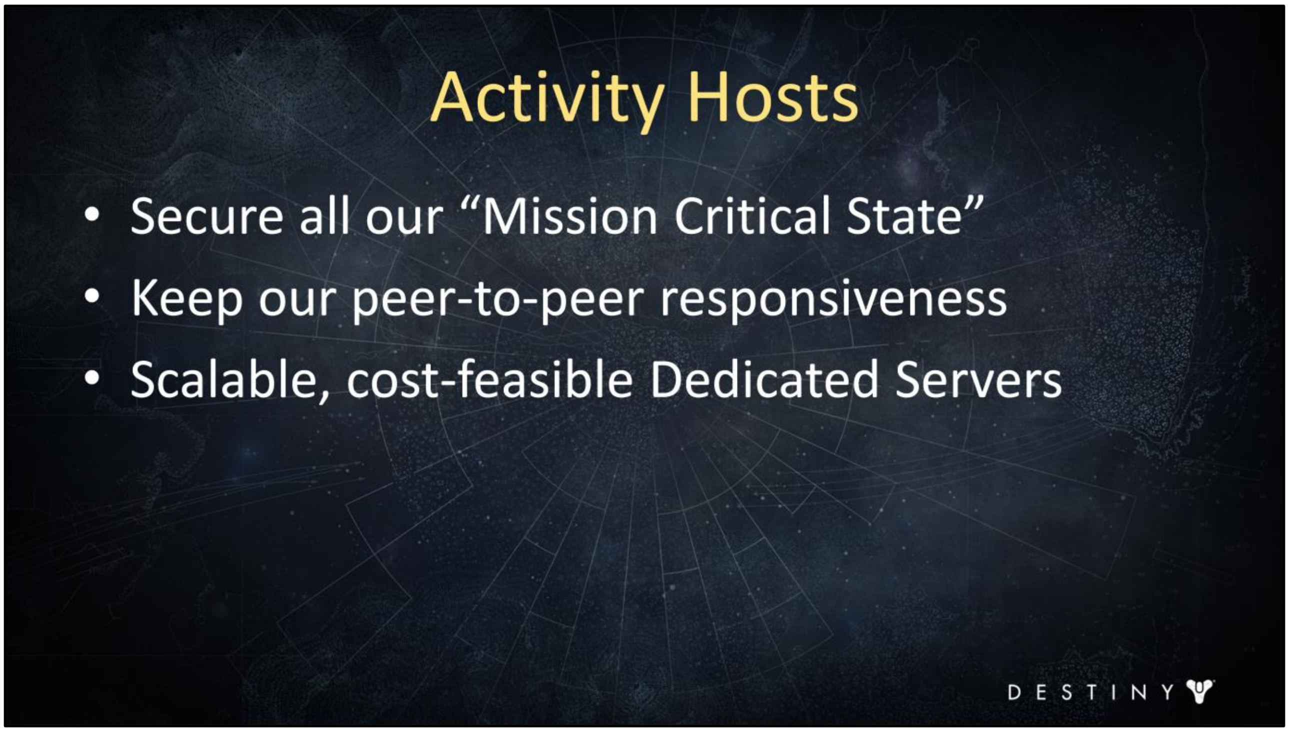


Reconciliation

- Halo Reach had too many of these bugs
- You have to program defensively, within every activity script
- How do we avoid this tax?

Many of them you won't even find – they're timing sensitive (our simple diagram that we just used required that 2 packets get sent, but you manage to drop just one of them, which is going to be hard to repro).

Halo Reach shipped with way more Host Migration bugs than we'd like, and that was just PvP. How do we avoid this tax on every single PvE script, each of which is much more complicated than a PvP gametype?



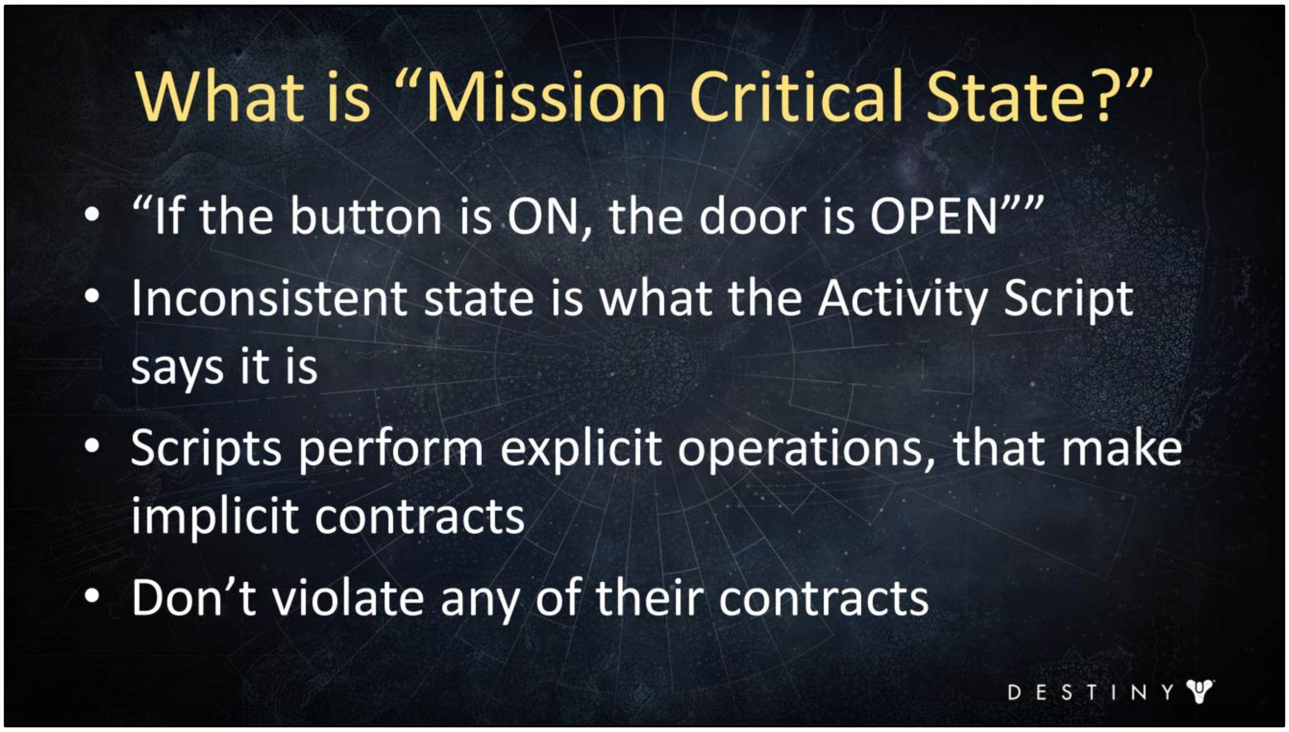
Activity Hosts

- Secure all our "Mission Critical State"
- Keep our peer-to-peer responsiveness
- Scalable, cost-feasible Dedicated Servers

So, what if we go back to that Dedicated Host idea, but only host the "Mission Critical State" that could break Activity Scripts?

We'd keep all the combat and physics peer-to-peer, so we have a responsive action game with low latency.

But we keep a minimal, cost-feasible set of state up in the cloud, so that it never Host Migrates.



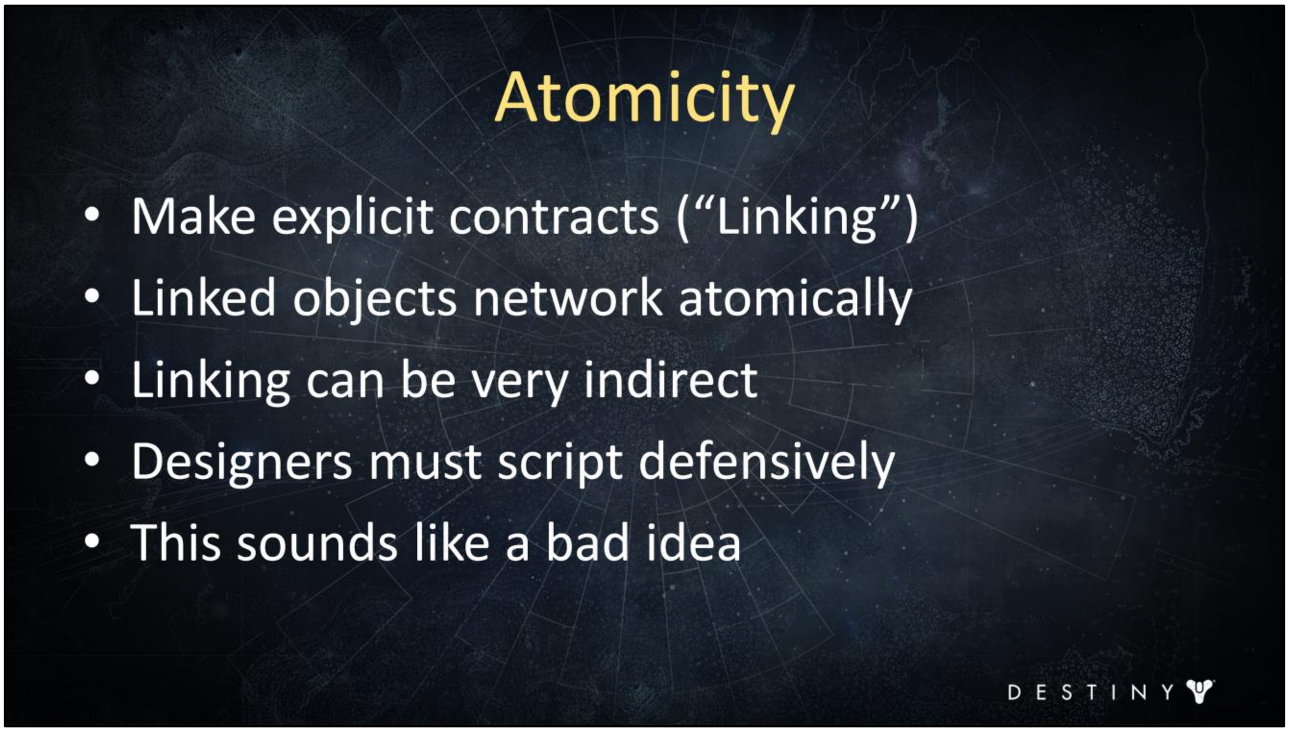
What is "Mission Critical State?"

- "If the button is ON, the door is OPEN""
- Inconsistent state is what the Activity Script says it is
- Scripts perform explicit operations, that make implicit contracts
- Don't violate any of their contracts

So what do we mean by "Mission Critical State"? Mission-Critical State is any contract that an activity script explicitly or implicitly requires.

If an activity script says a button opens a door, it's implicitly linking the two together, and saying "If the button is ON, the door is open, and vice versa".

And the really important, and tricky thing here, is that these contracts are usually implicit, not explicit. You have to first figure out how to discover all the Activity Script contracts, before you can find a way to enforce them all.

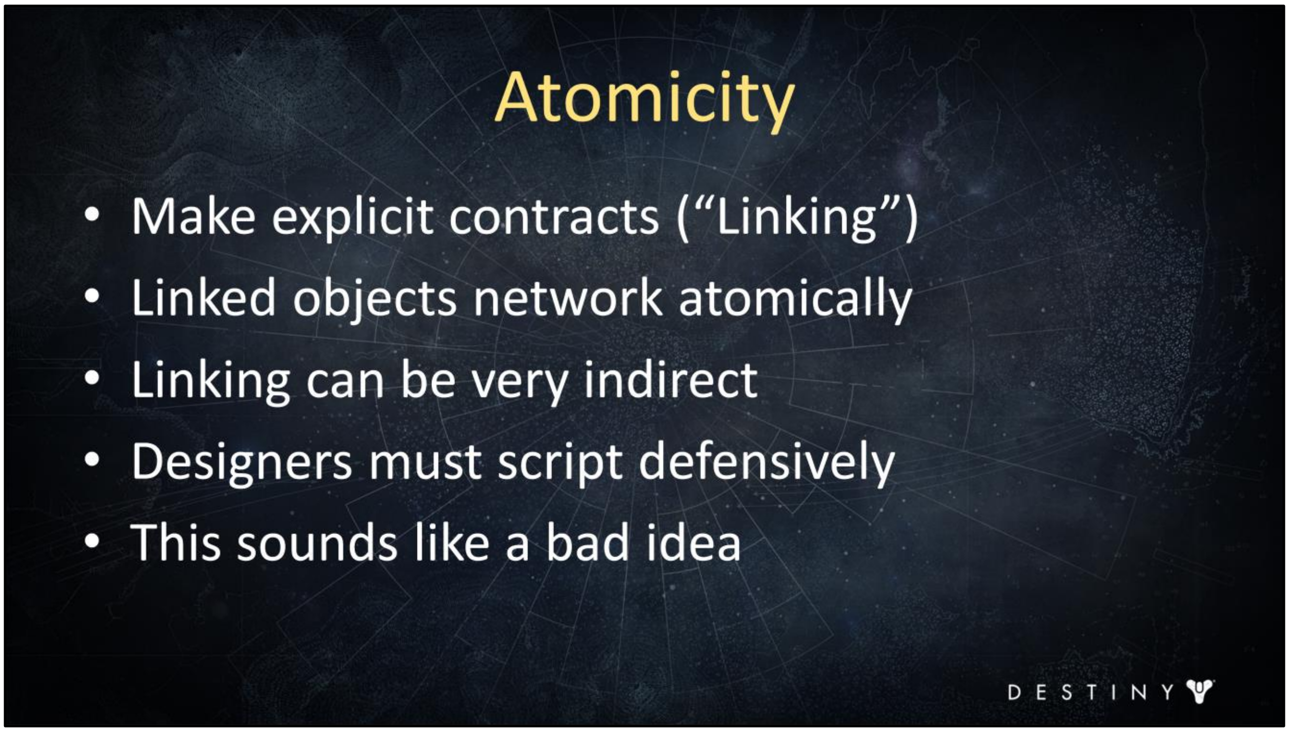


Atomicity

- Make explicit contracts ("Linking")
- Linked objects network atomically
- Linking can be very indirect
- Designers must script defensively
- This sounds like a bad idea

So some of our early attempts here were to try to create atomicity guarantees between any "linked" state. So if the button is connected to the door, just make sure all networking updates for both of them are atomic, and you're all good.

This approach has 2 major problems:

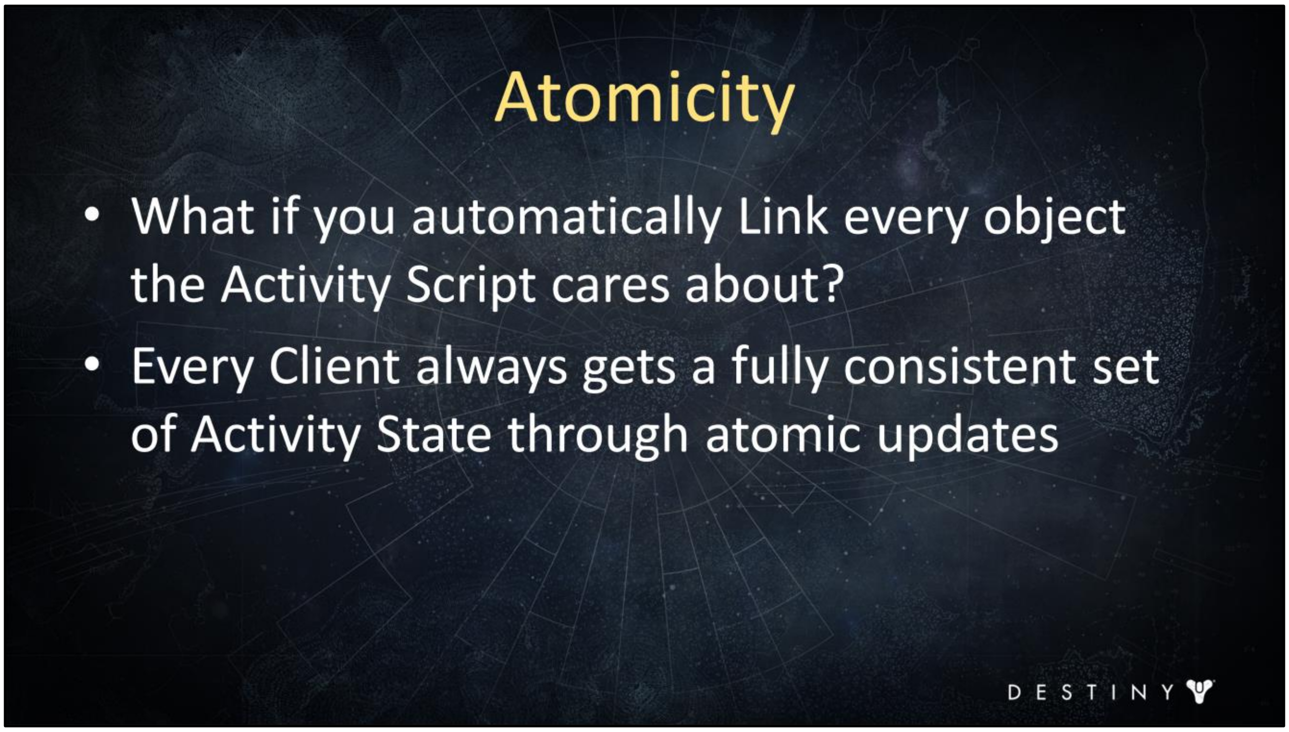


Atomicity

- Make explicit contracts (“Linking”)
- Linked objects network atomically
- Linking can be very indirect
- Designers must script defensively
- This sounds like a bad idea

A lot of contracts within real scripts are a lot more indirect -
(Kill 2 AI, which spawns a 3rd guy, who when he reaches a door, he opens it. Is the door “linked” to the original 2 AI?)

also, it requires that someone (if we’re talking activity script, we mean the designer) actively think about host migrations and atomic linkage, which puts us back in the defensive programming trap that we want to avoid



Atomicity

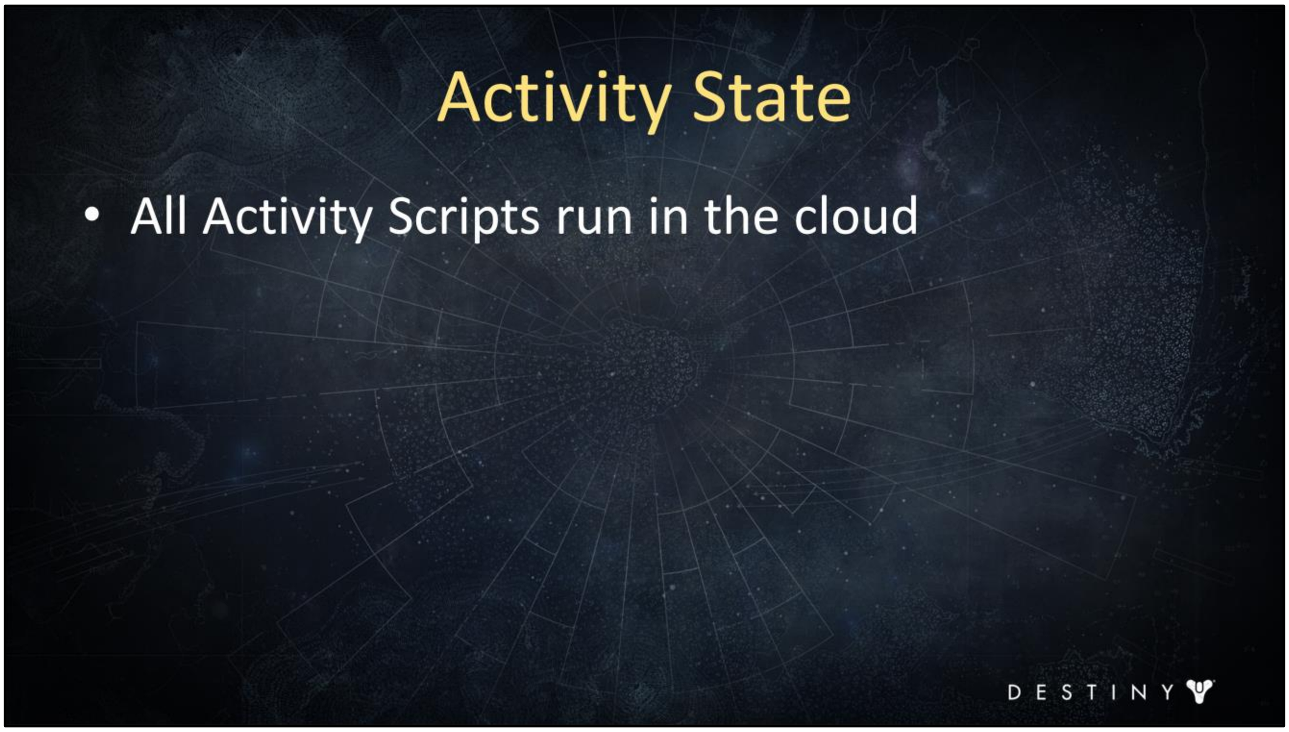
- What if you automatically Link every object the Activity Script cares about?
- Every Client always gets a fully consistent set of Activity State through atomic updates

So instead, what if we just compile the list of every object that the Activity Script ever cares about (all 3 AI, the door, the CTF flag), and make them all atomic with each other?

Then we only update atomically, and every client always gets a fully consistent set of activity state that satisfies all contracts

And it's important to note here – we are trying to get the *minimal* set of necessary state. That way as much as possible is still hosted by your low-latency Physics Host.

We call this minimal subset of gamestate that we “care about” and want to atomically reconcile, “Activity State”. “Activity State” is all authoritative on the “Activity Host”, which runs up in the cloud.

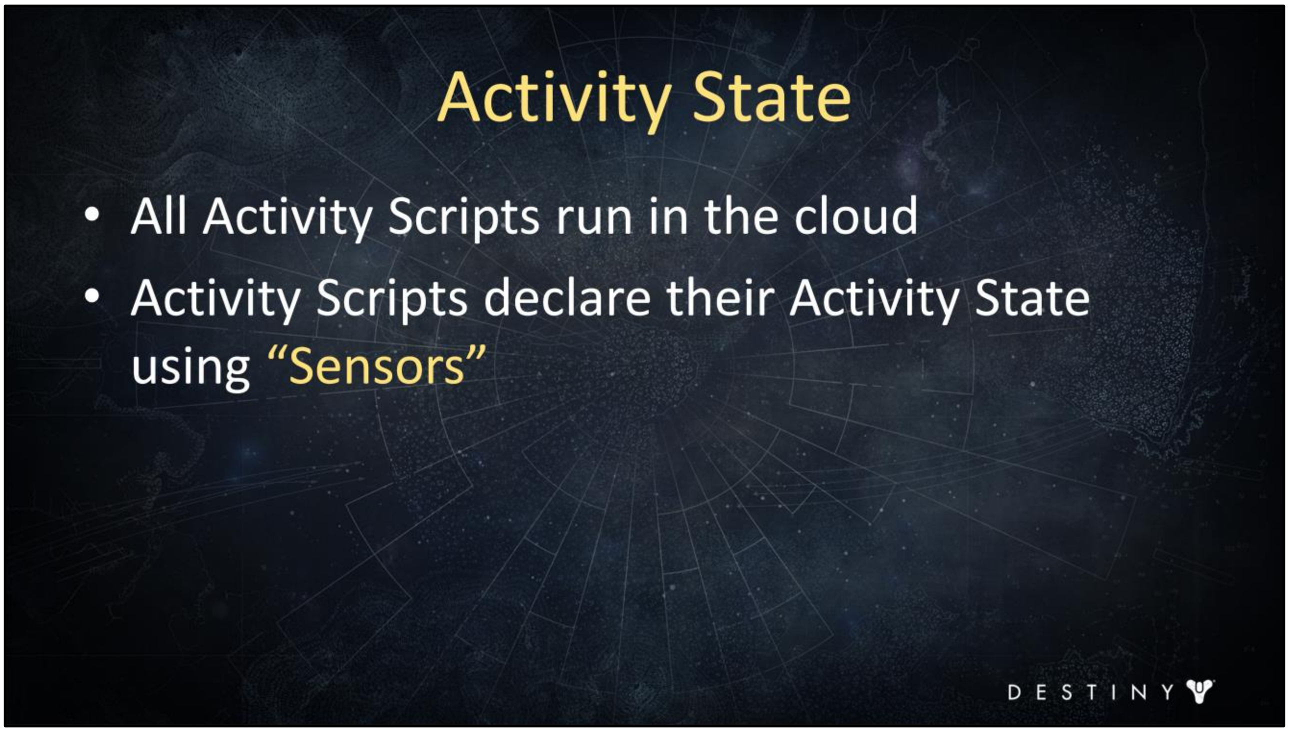


Activity State

- All Activity Scripts run in the cloud

DESTINY

So, all of our Activity Scripts – like a Story Mission Script, run in the cloud. And these scripts all declare, up front, what state they care about.



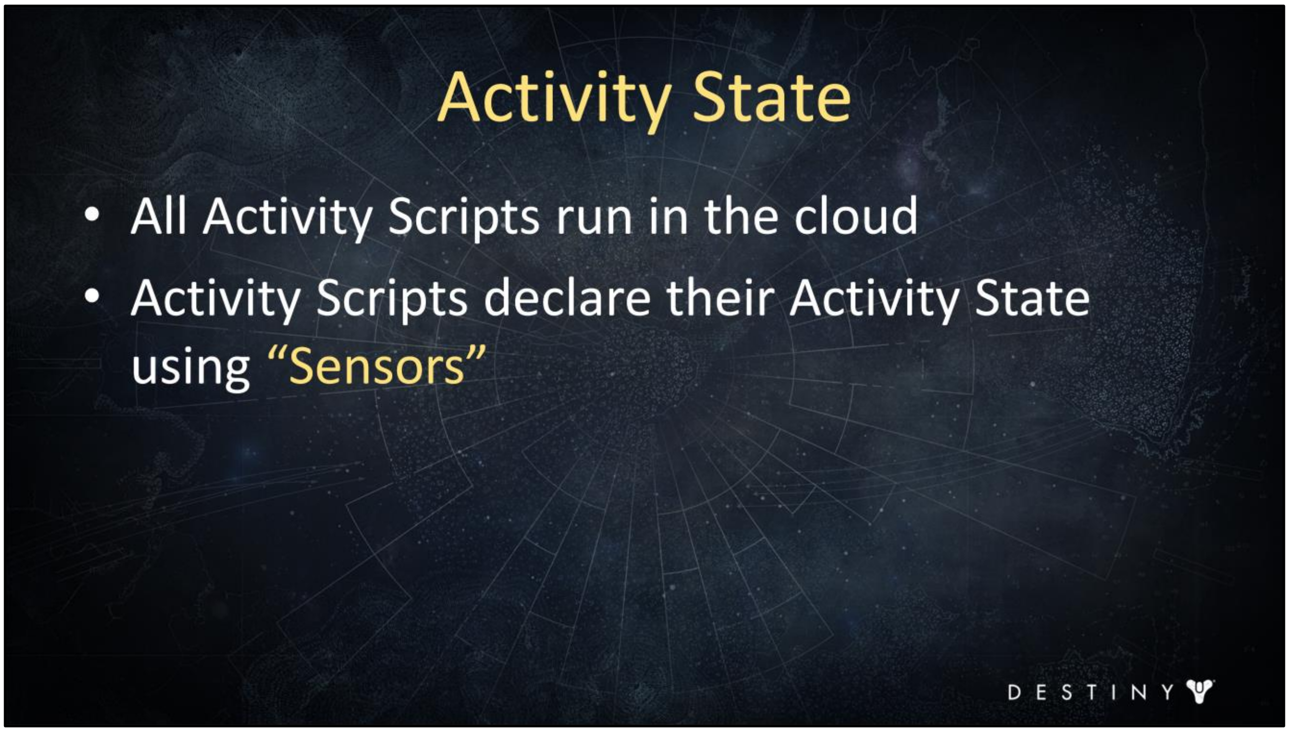
Activity State

- All Activity Scripts run in the cloud
- Activity Scripts declare their Activity State using "Sensors"

Activity Scripts operate on some set of objects – those are by definition the objects they care about. You can't reference an object in Activity Script without including it in Activity State.

There's tons of other objects in the simulation they never care about – bullets, crates, etc.

But we can do even better than that – Activity Scripts also declare *what* they care about each of these objects.

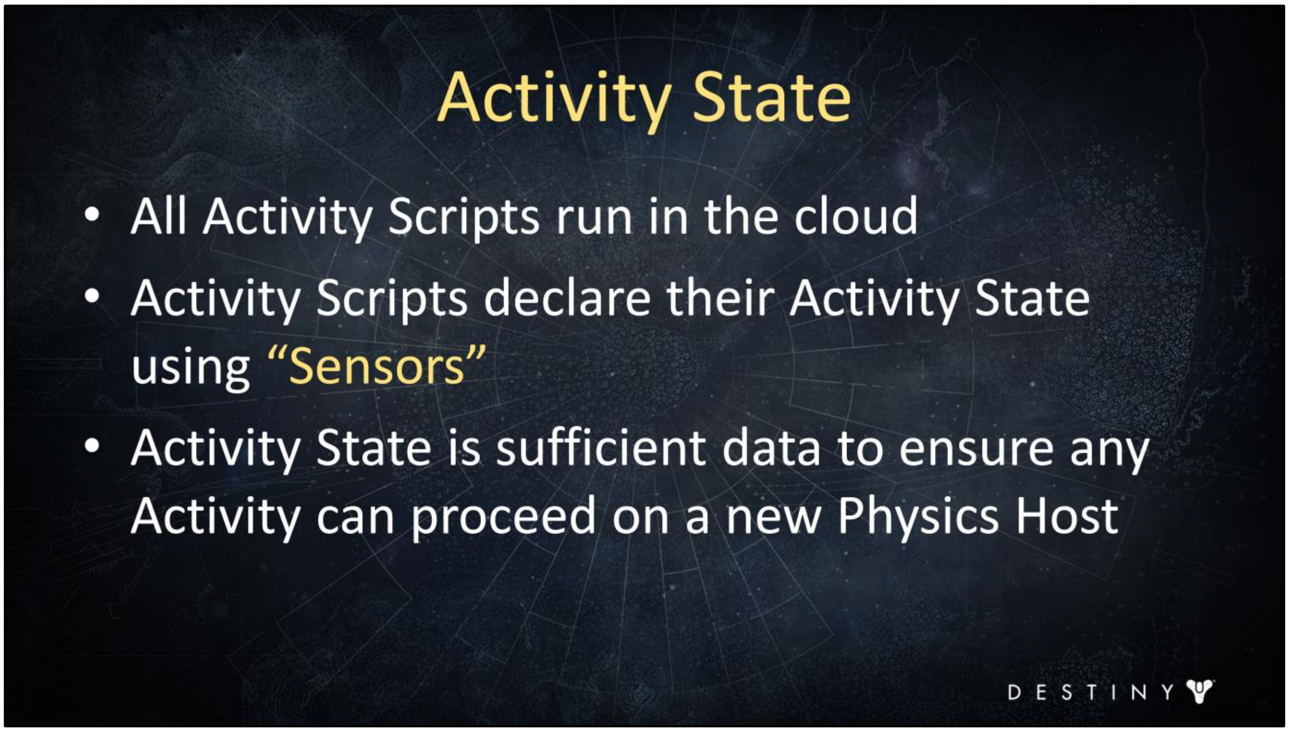


Activity State

- All Activity Scripts run in the cloud
- Activity Scripts declare their Activity State using “Sensors”

Let’s take a Squad for example (a group of coordinated AI in Destiny). An activity script might care – has it spawned? How many AI are alive or dead? But they probably don’t care about the individual health or worldspace positions of those AI.

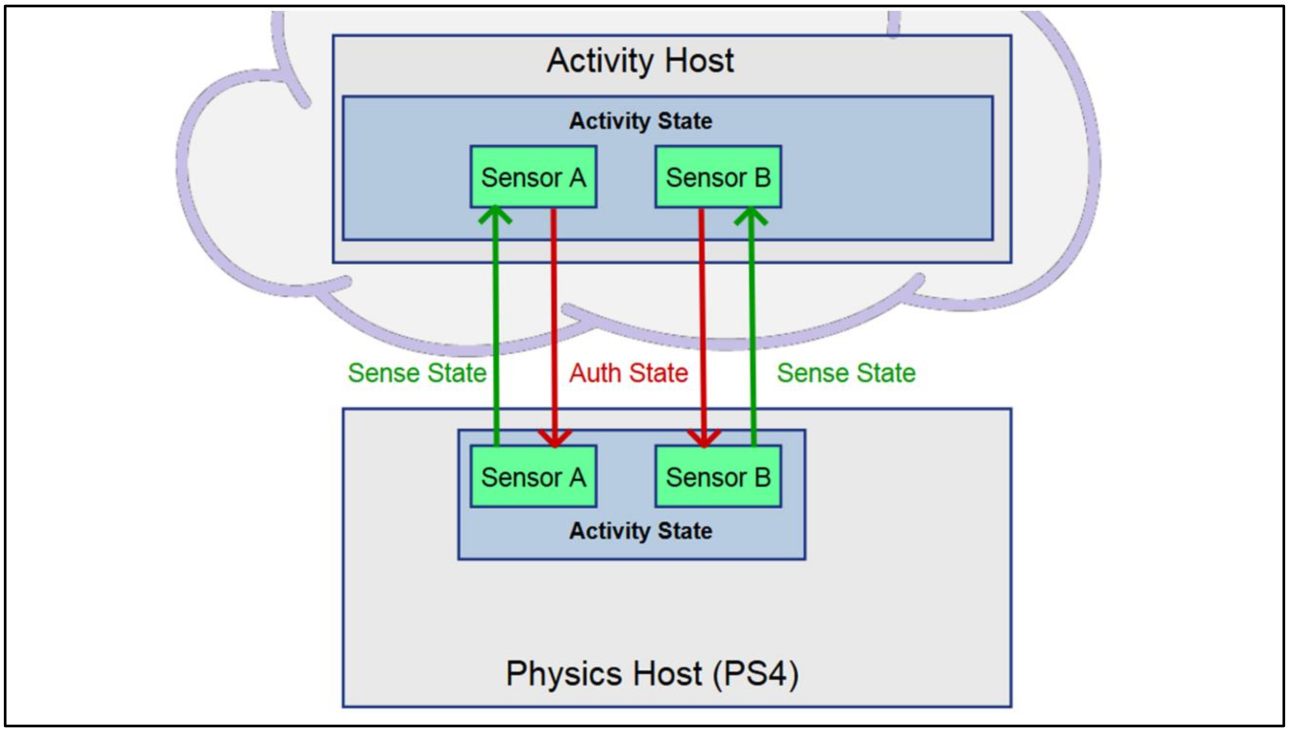
We call these bits of discrete, mission-critical state “Sensors”



Activity State

- All Activity Scripts run in the cloud
- Activity Scripts declare their Activity State using “Sensors”
- Activity State is sufficient data to ensure any Activity can proceed on a new Physics Host

We can take all this specified state (which isn’t very much) and make it all atomically reconcilable and persisted in the cloud. This way, at any time, a new Physics Host can take over, and can set itself into a fully consistent state that will allow the Activity script to proceed.

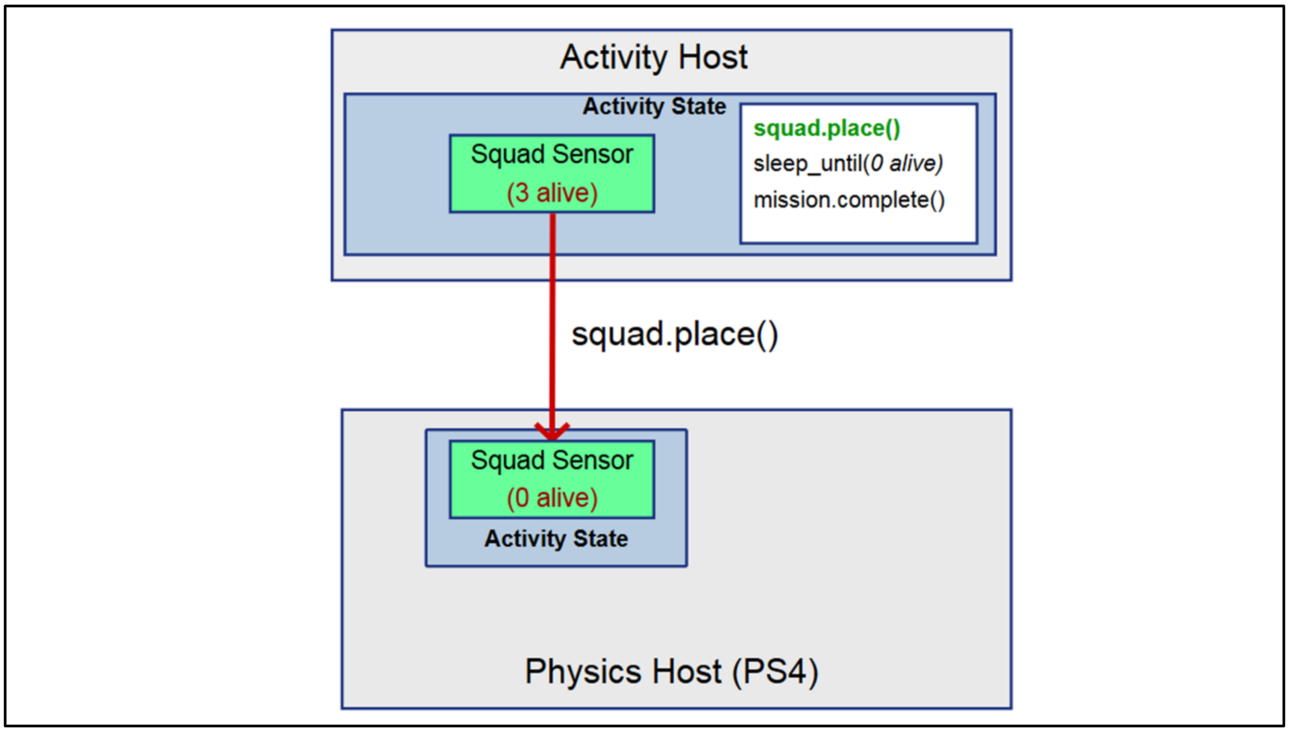


So here's an example of Activity State, on both
the Physics Host (that's a PS4 or 360 console in someone's house),
and on the Activity Host, which lives up in the cloud.

There's a full duplication of sensor state on both machines.

Auth State is what we call sensor communication from the Activity Host to the Client (Because the Activity Host is always the Authority over Activity State). Those are the red arrows.

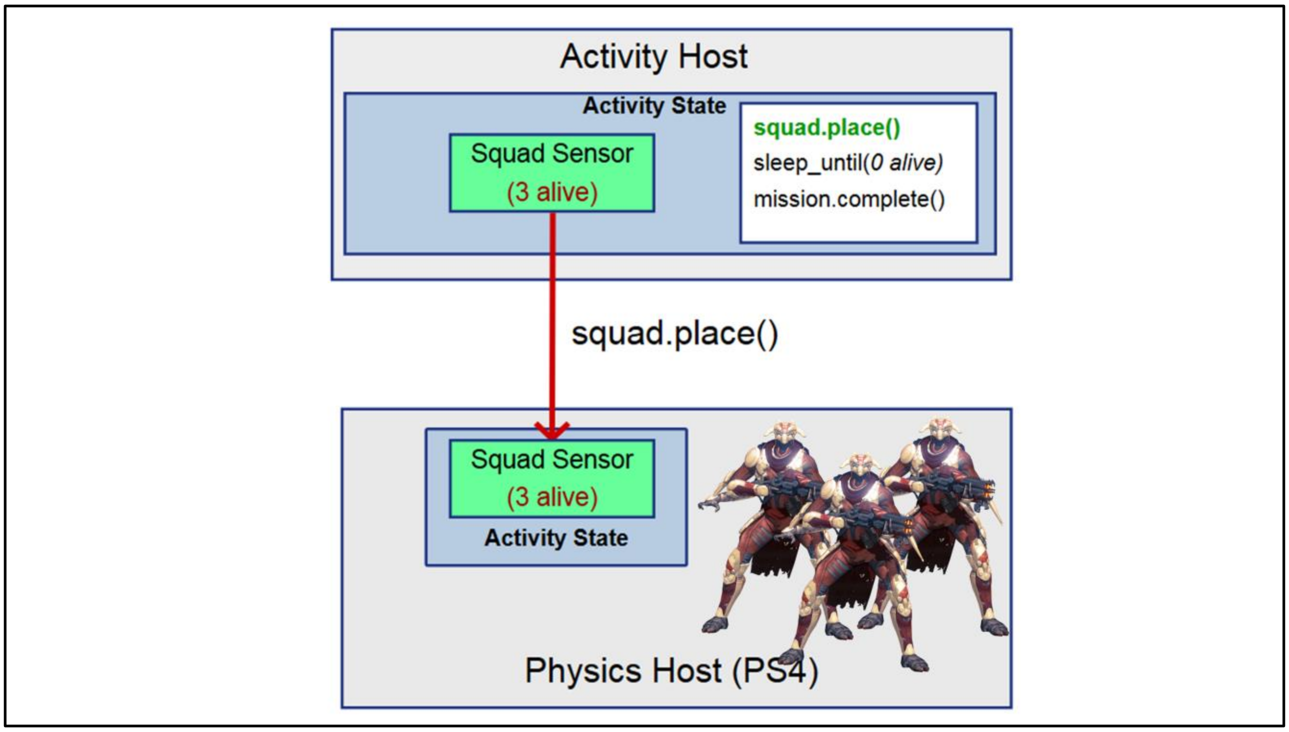
and Sense State is communication in the other direction – the green arrows



As a simple example, let's suppose you have a squad, you want to spawn it, and trigger an activity complete banner when everyone in the squad is dead.

I've written a sample script up there on the right – place the squad, wait for them all to die, then play activity complete.

So first the AH script calls "place()". This sends down Auth State to the Physics Host, which spawns 3 AI.

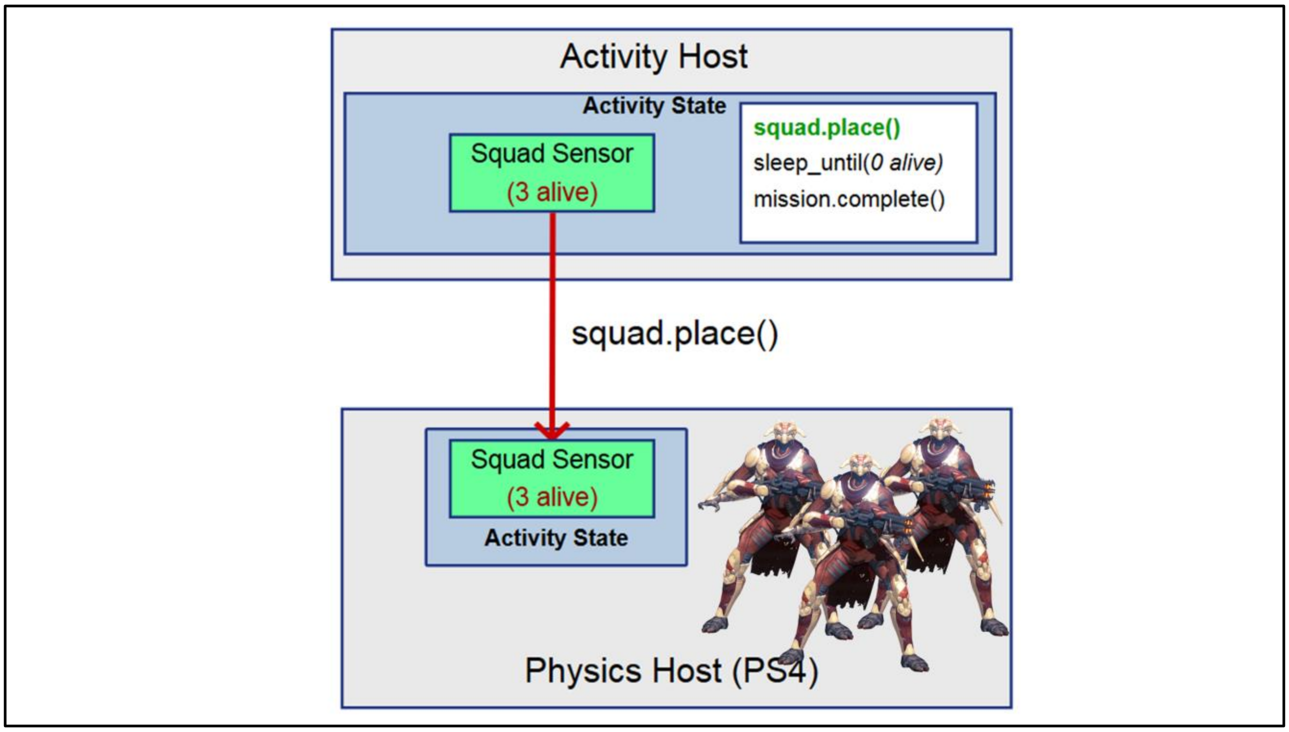


At this point the 3 AI are simulated on the Physics Host, and P2P networked to other clients.

Now there's already a couple interesting things to note here.

First, the Activity Script itself is running in the cloud – none of our Lua logic for Activity Scripts are executing on the PS4 client.

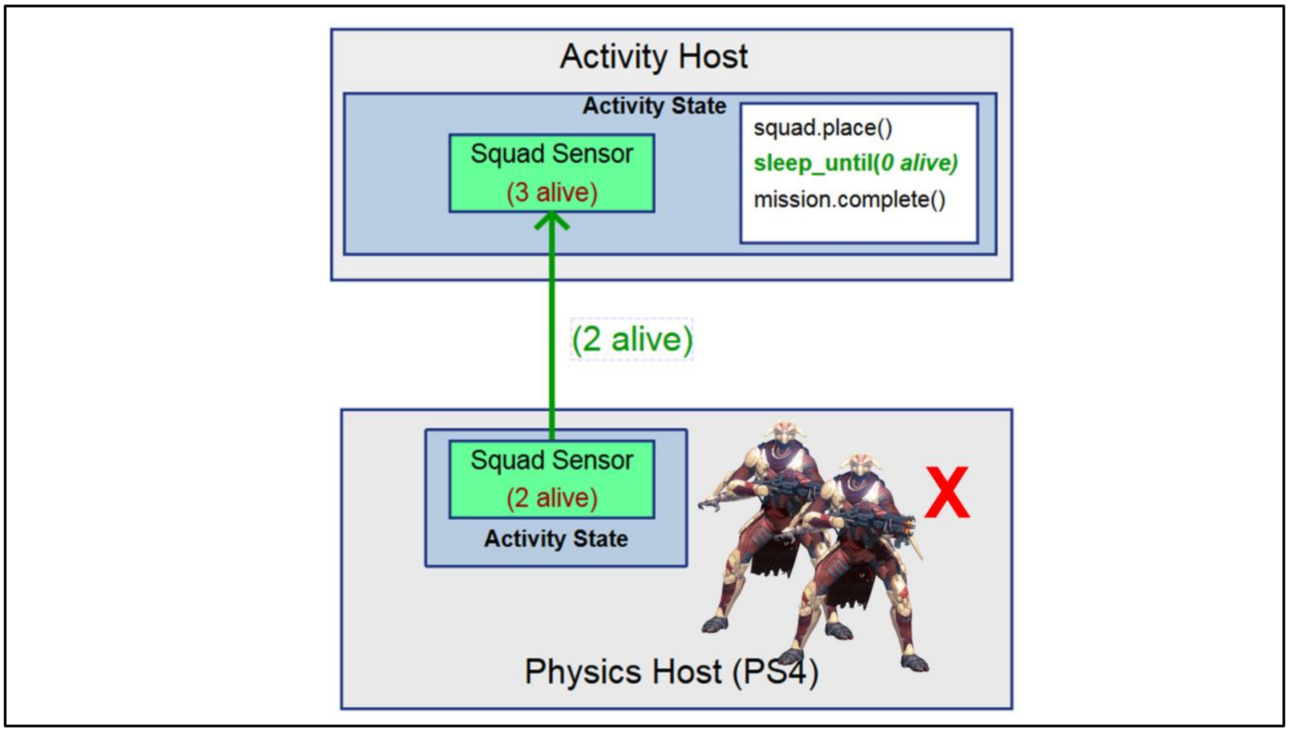
Second, it's worth pointing out that these 3 AI are not in "Activity State". There's a Squad Sensor inside activity state, but it's tracking very minimal state (there are 3 AI alive, and they're using this specific firing area).



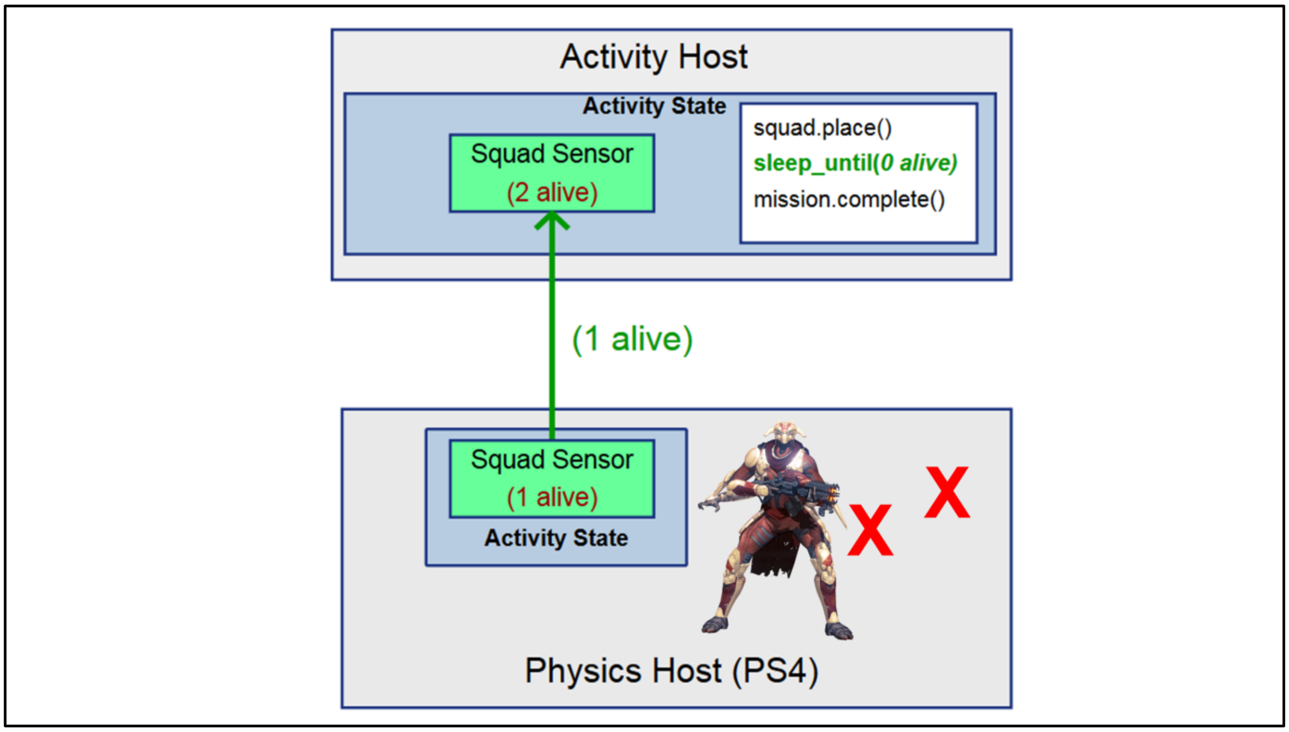
Outside of Activity State there are 3 linked heavyweight objects – actual bipeds with worldspace positions and skeletons with specific animation state. None of this state lives on the Activity Host.

The networking protocols are also different – those 3 AI are networked just like any other P2P object – they're each separately networked to all Physics Clients in the bubble, independent of the sensors, using traditional "Halo PvP Networking".

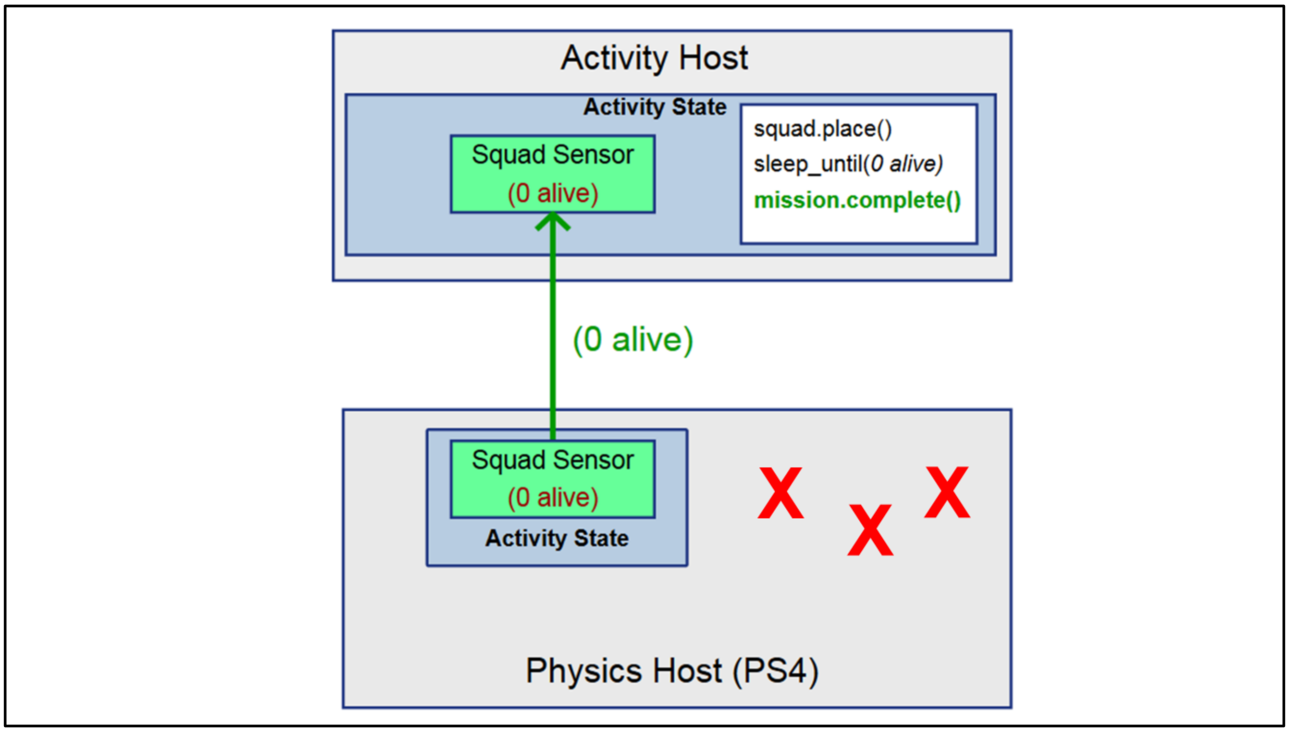
So, now let's start killing aliens.



As the Physics Host detects each kill, it sends Sense State up to the Activity Host decrementing the alive_count() on the sensor.

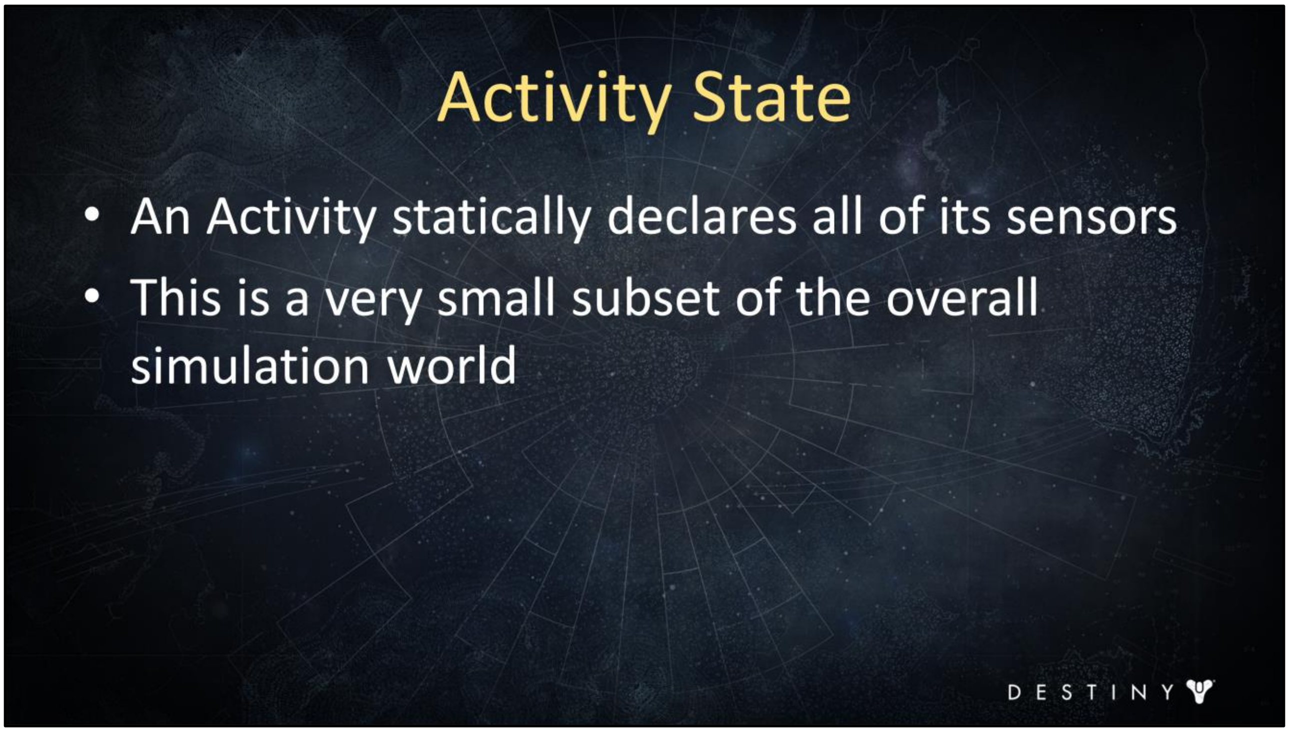


These updates are timesliced to be relatively infrequent (we use 10hz for both CPU and Bandwidth reasons). And it's atomic with all other sense state changes. We send all coherent sensor changes up simultaneously on a given sensor update.



After the alive_count hits 0, during the next script update, the script coroutine continues on the Activity Host.

At this point it would communicate down via an objective sensor, which would then update the HUD to display the activity complete banner.

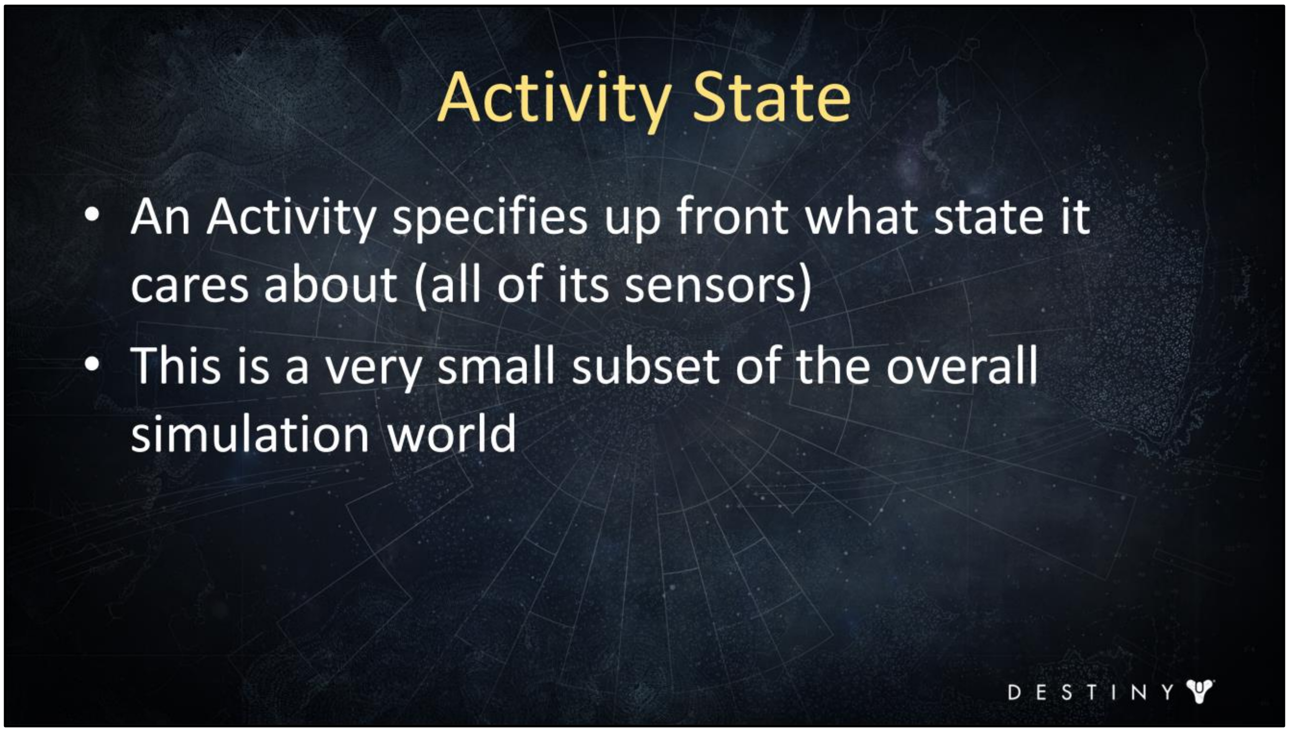


Activity State

- An Activity statically declares all of its sensors
- This is a very small subset of the overall simulation world

So, for a given activity, we specify all of its sensors up front – all the objects it might care about during the activity.

Sensors can be related to game objects, but they aren't 1:1. You can have multiple objects tracked by a single sensor (like a squad), or multiple sensors on a single object (for discrete unrelated components of state)



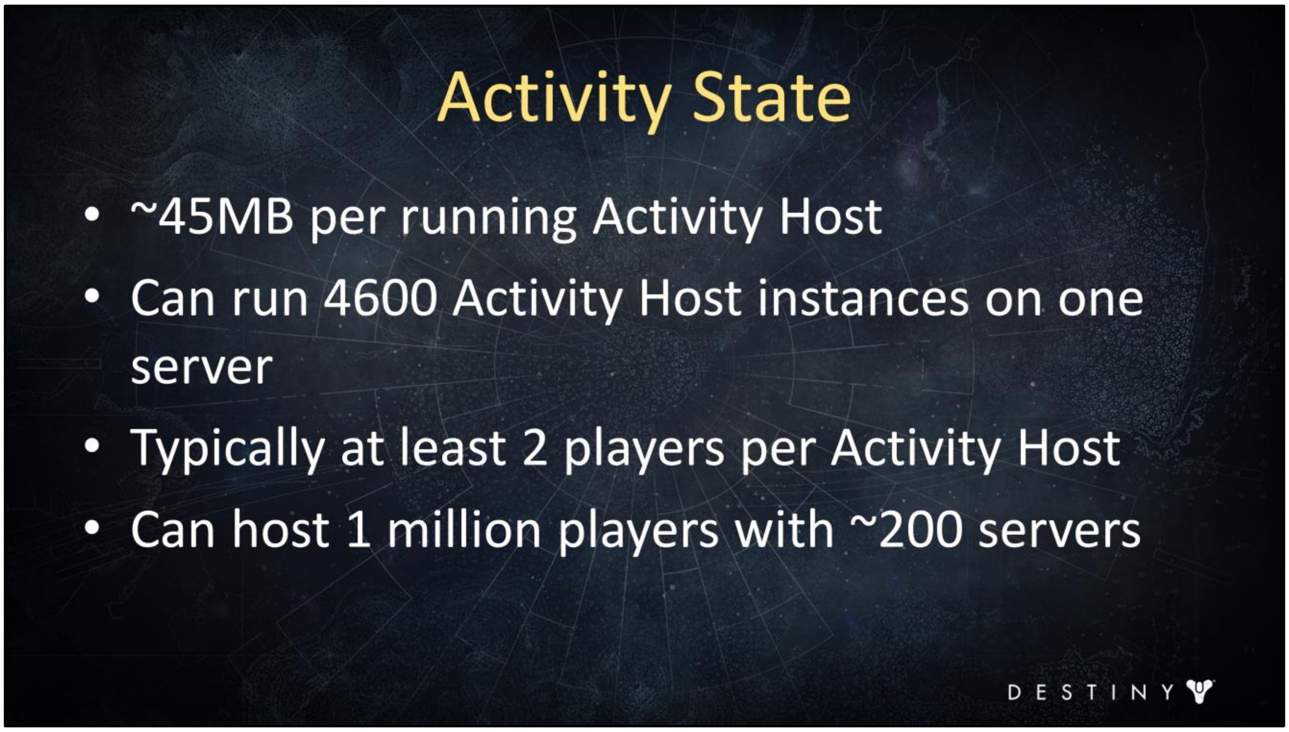
Activity State

- An Activity specifies up front what state it cares about (all of its sensors)
- This is a very small subset of the overall simulation world

DESTINY

Basically, on the Physics Host, any sensor can read anything it wants about gamestate. But whatever internal memory it decides to store gets communicated up to the Activity Host as sense state.

This sensor internal memory is a very small subset of the overall gamestate, so our Activity Hosts are much cheaper than a traditional Physics Host Dedicated Server. We only need to pay for our datacenter simulating and networking Activity State, not the entirety of the physical simulation.

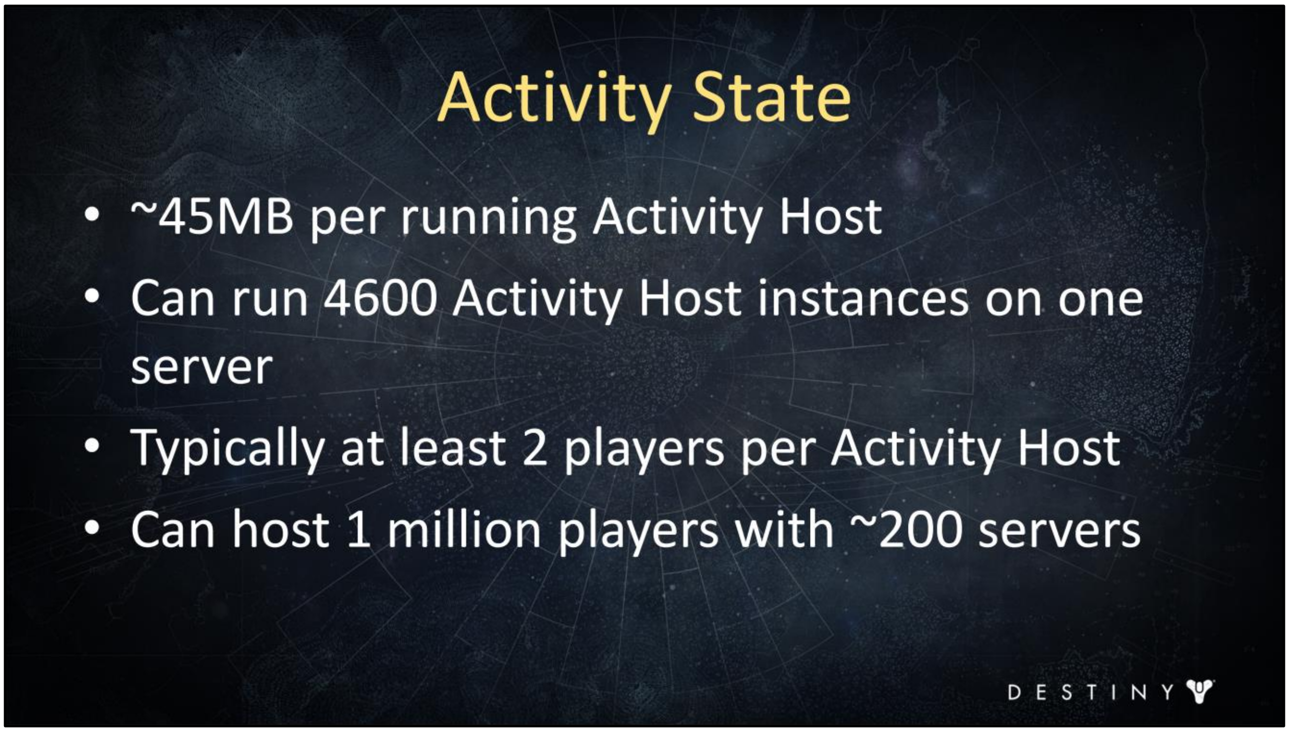


Activity State

- ~45MB per running Activity Host
- Can run 4600 Activity Host instances on one server
- Typically at least 2 players per Activity Host
- Can host 1 million players with ~200 servers

By pruning Activity State down to what's absolutely necessary, we're able to get significant scale on our datacenter.

Each of our Activity Host executables is around 45MB. We could make this even smaller - it's a stripped down version of the Destiny executable.



Activity State

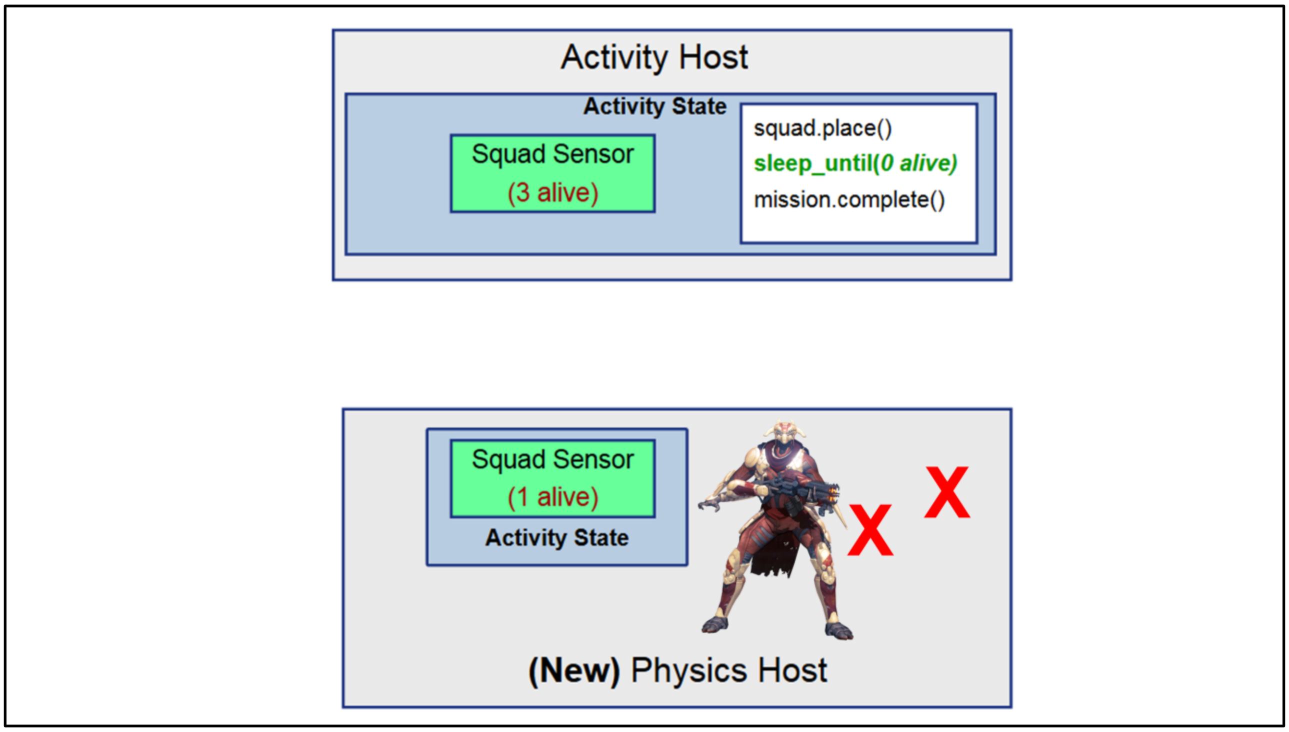
- ~45MB per running Activity Host
- Can run 4600 Activity Host instances on one server
- Typically at least 2 players per Activity Host
- Can host 1 million players with ~200 servers

DESTINY

We tick our Activity Hosts at 10Hz, which allows us to run almost 5000 per server [40 core, 256GB]

Given that we typically have a bit over 2 players per Activity Host in real-world conditions, that means our datacenter can handle a hypothetical 1 million concurrent users with only a couple hundred servers, and that's with plenty of safety headroom on each machine.

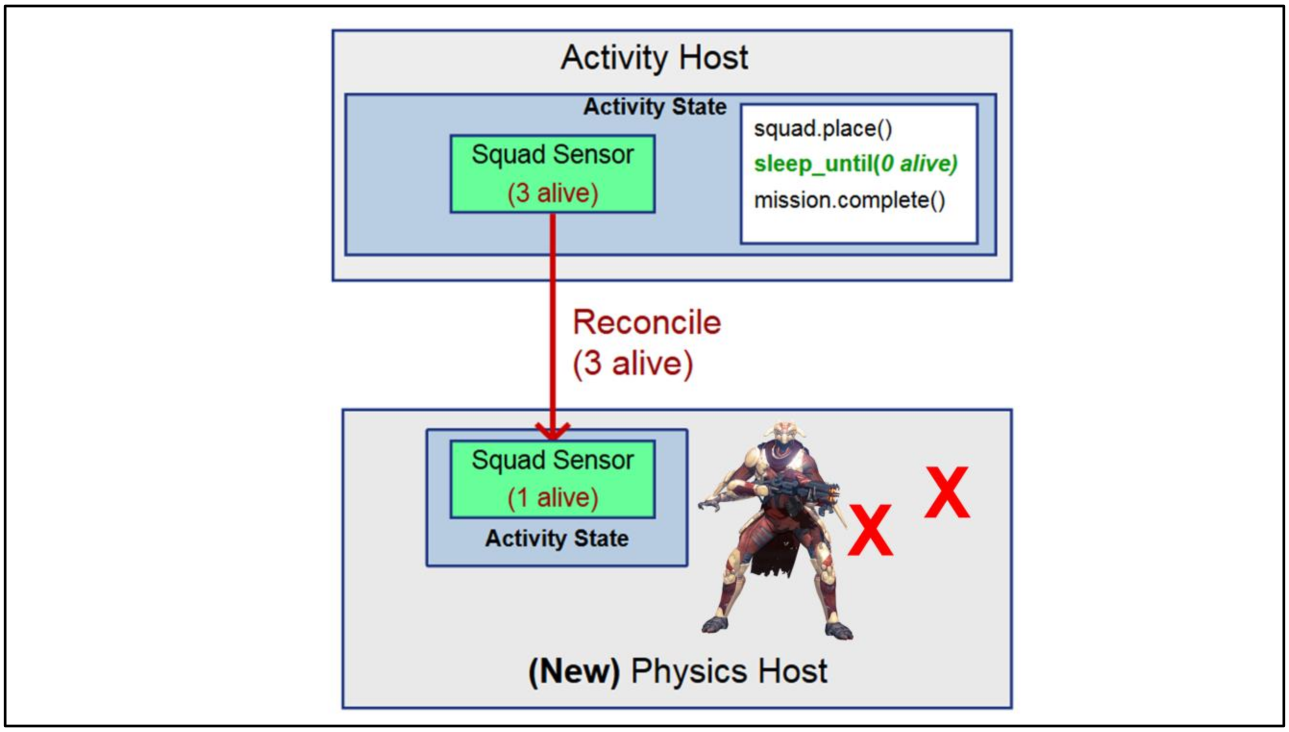
That's dramatically better scale than trying to use a full dedicated server. With full dedicated servers, that same hypothetical 1 million players would require half a million headless PS3 processes, each running our full game simulation.



So now, what happens if you have a Host Migration?

Let’s suppose the AH has an alive count of 3 – it never got sense state saying that any of the AI died. But the new Physics Host, for whatever reason, has inconsistent state that gives it an alive count of 1

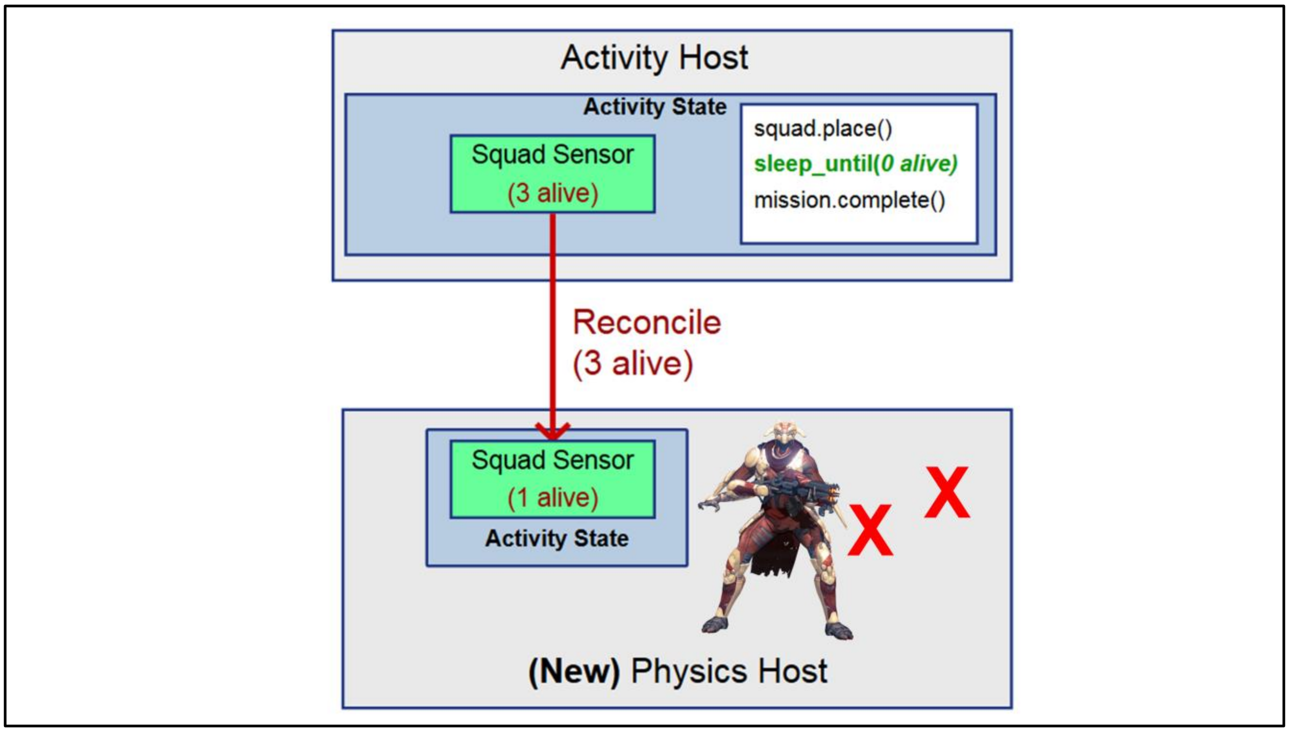
So the new Physics Host has only 1 AI left alive (and it’s only networking 1 AI to Physics Clients, using Halo P2P Networking), whereas the Activity Host authoritatively states that there should be 3 AI still alive.



The Activity Host is the session authority, who arbitrates election of new Physics Hosts.

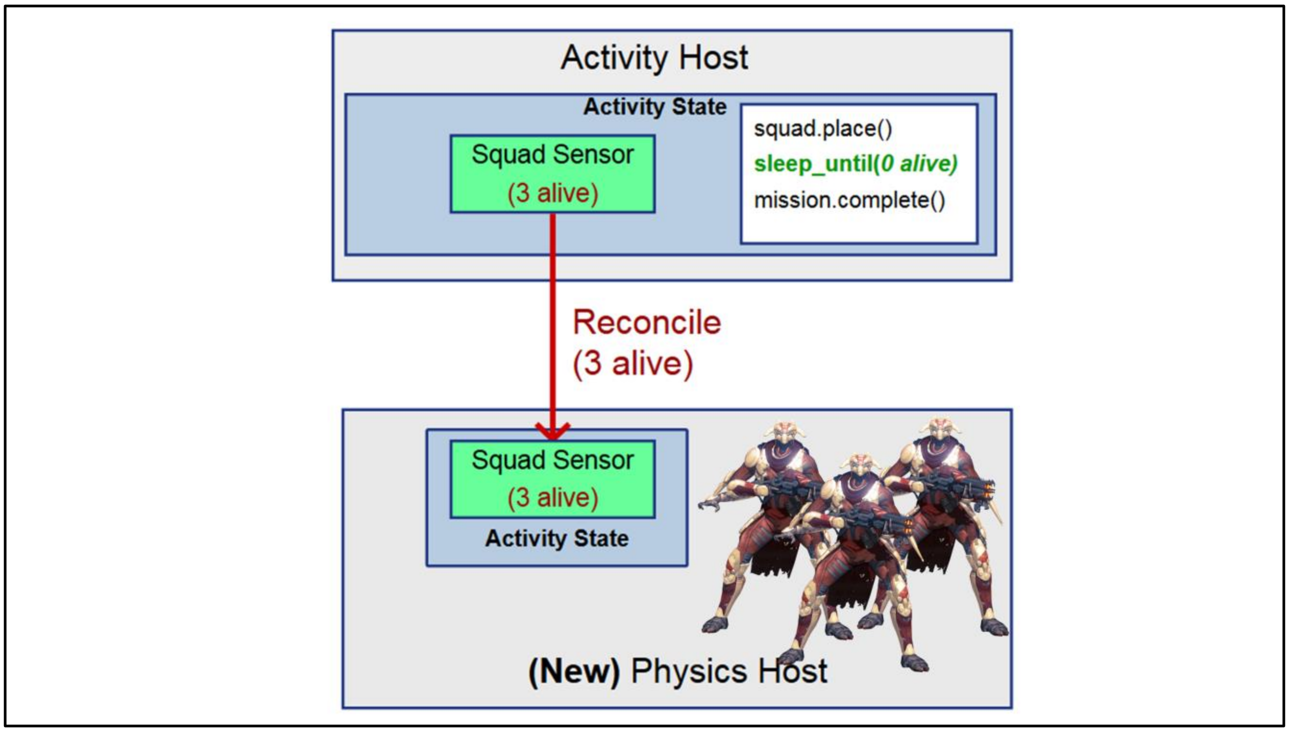
So any host migration necessarily goes through the Activity Host, and it calls a special "Reconcile" function on each sensor.

So the Squad sensor (along with all other sensors) send down auth state in a special "Reconcile" command, which requires that the new Physics Host modify his simulation to match the Auth State.



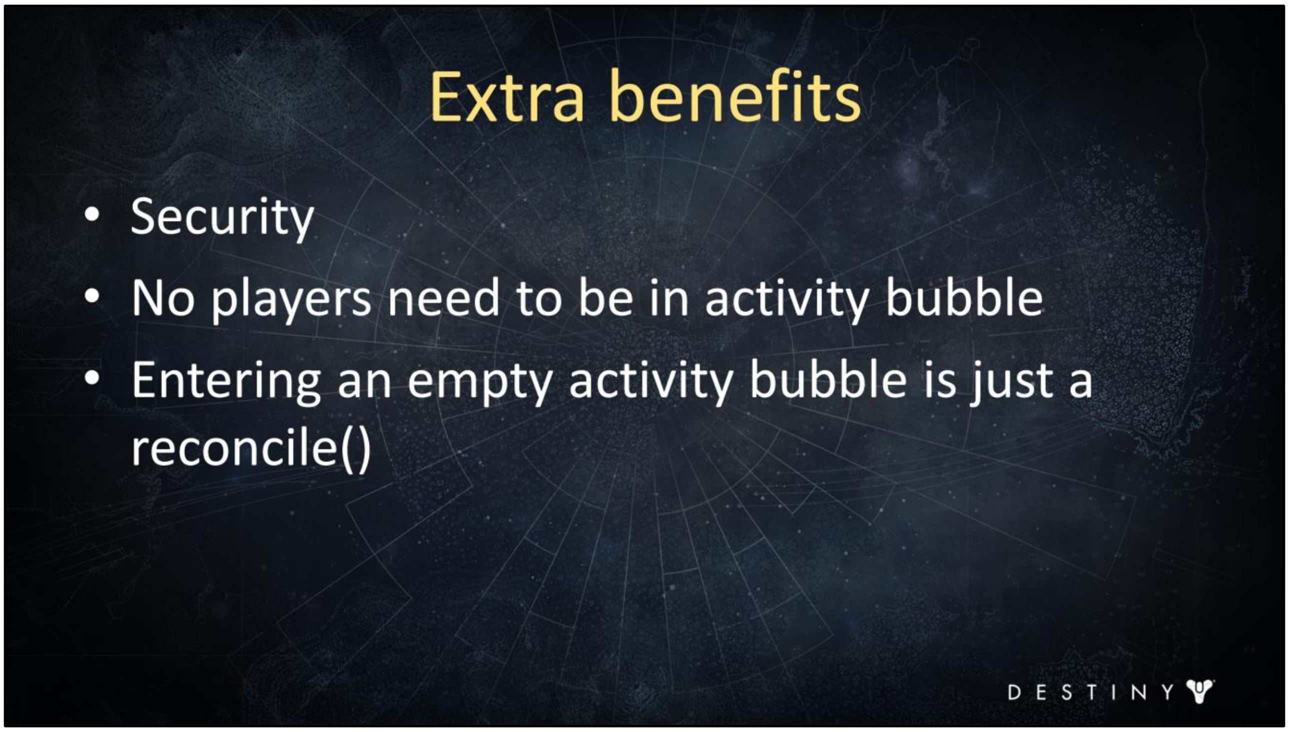
In the common, graceful migration case we talked about earlier, this often results in zero or imperceptible changes, if the new Physics Host is up to date with Activity State.

In this case, the Physics Host has to spawn 2 new AI, to match the Auth State requirement of 3.



And one cool thing here is *how* we spawn the 3 vandals – we already have “squad_place” logic in the sensor that knows how to tell the game to instantiate new AI bipeds, we can typically reuse those exact code pathways to handle Reconcile – it’s just like any other Auth State update.

Similarly, if the Auth State had said there were fewer AI than the Physics Host was simulating, code in the sensor would choose AI to instantly kill, in order to get it down to the correct count.



Extra benefits

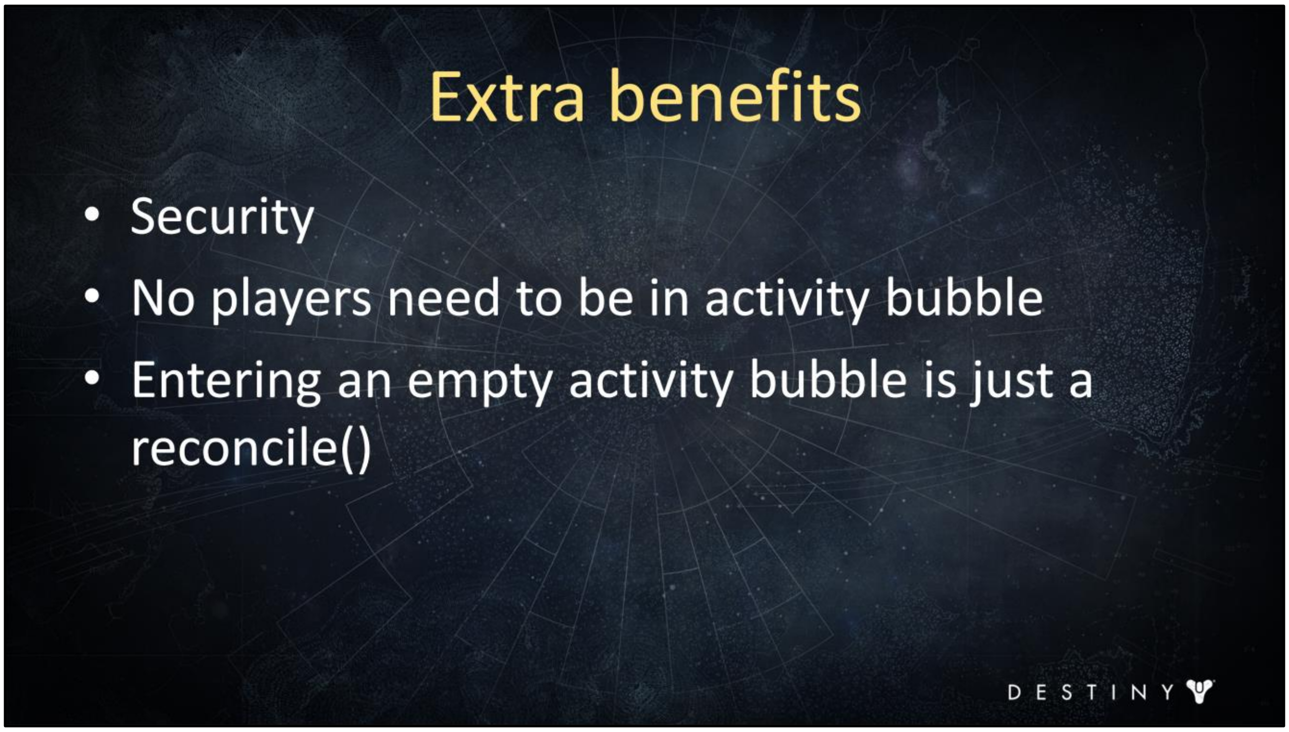
- Security
- No players need to be in activity bubble
- Entering an empty activity bubble is just a reconcile()

DESTINY

There's several other powerful benefits that come from having these dedicated Activity Hosts.

One is Security – for every Destiny Activity being played right now, there's a machine we can trust that has fairly complex understanding of the minute-to-minute gameplay. This helps greatly in combatting piracy, and is a strong avenue for investigating suspected cheaters and exploits.

Additionally, Activity Bubbles can execute scripts without any players being present – we don't have to leash you to a boss room, and can maintain an activity script even when every player has left the bubble and locally deleted every object from their console's physical memory.



Extra benefits

- Security
- No players need to be in activity bubble
- Entering an empty activity bubble is just a reconcile()

And what happens if a player reenters one of these empty activity bubbles? The player herself will spin up the simulation in the default state when she crosses the z-leg,

and she will be elected as the new physics host (since it was previously empty)

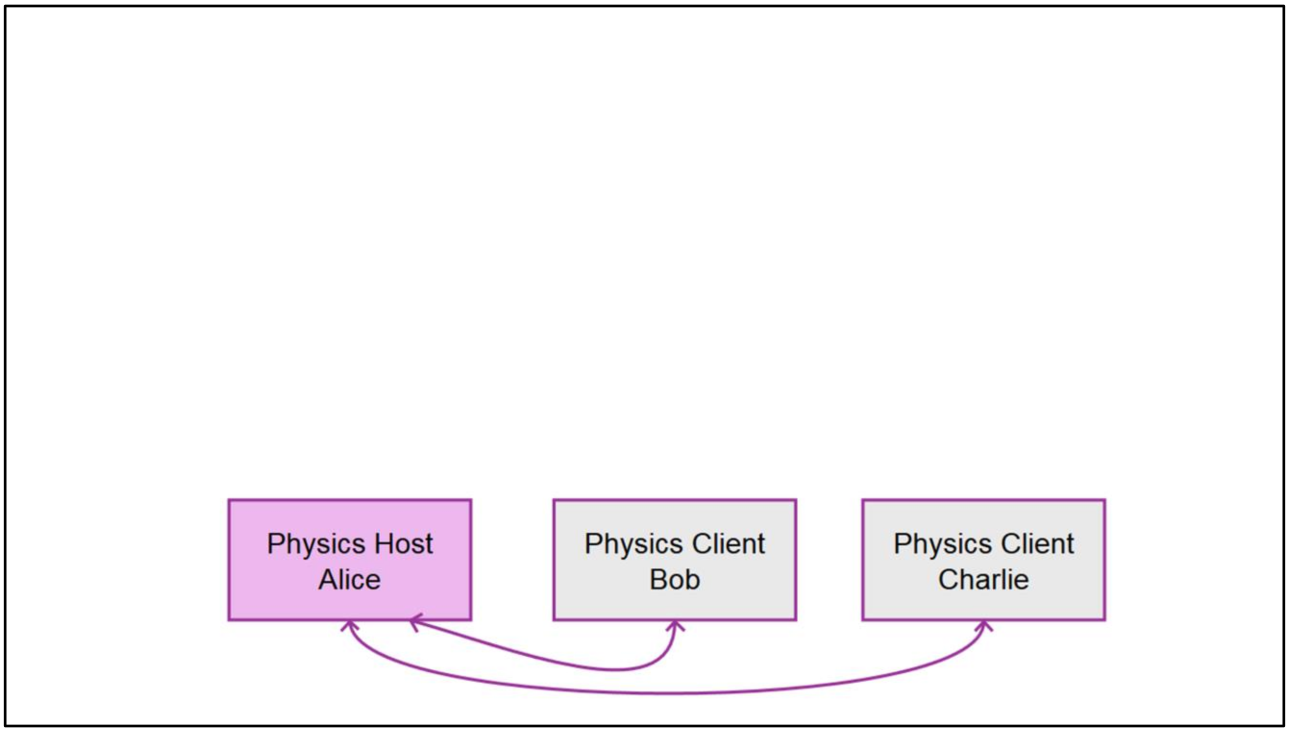
and then she will get a “reconcile()” call where she can fixup all of her simulation state to match the authoritative sensor state.

This is actually pretty cool – we automatically get a minimal-state load functionality when entering an empty activity bubble, because we only fixup the declared sensors and leave everything else in its default state.

Public Bubbles

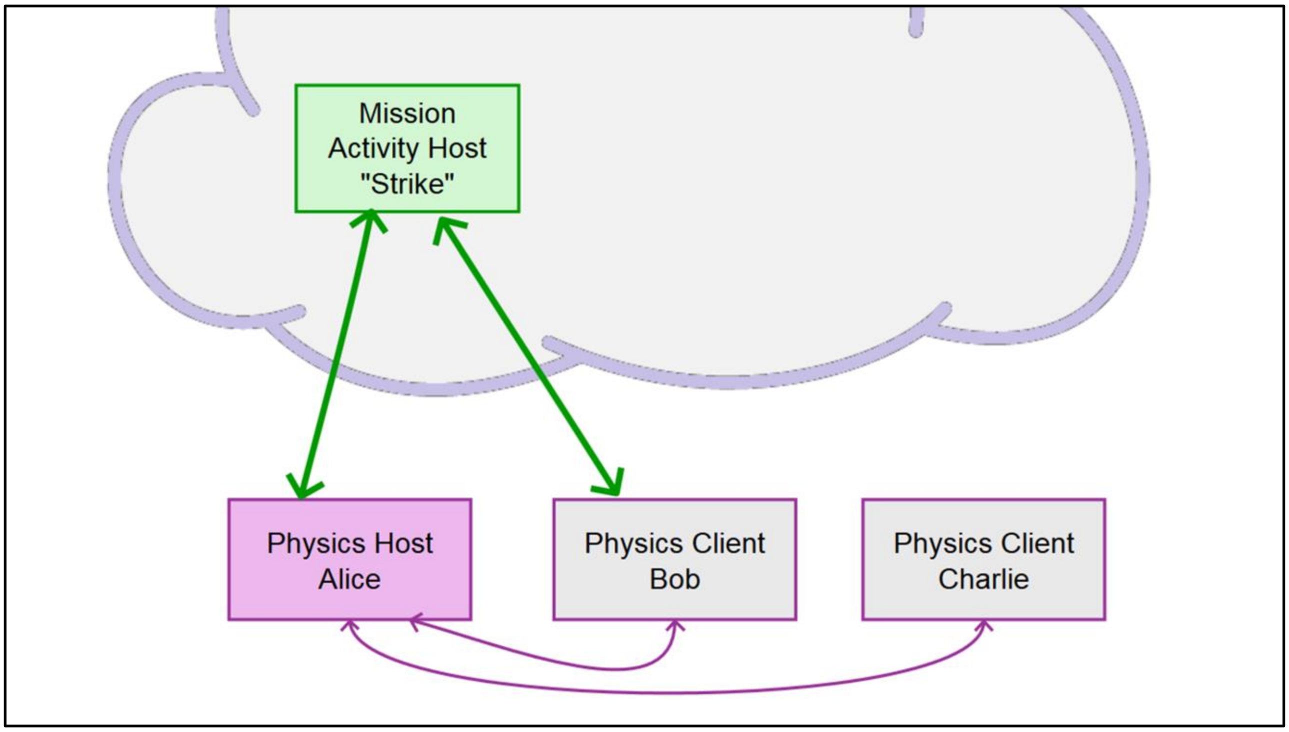
Ok, so all the examples we've talked about so far are just for co-op scripting – 2 or 3 players in a single fireteam, all on the same activity in a networked environment.

But Destiny is also about meeting strangers, on intersecting paths. What happens when two players meet who are on two completely different activities? This is where it starts to get even more complicated.

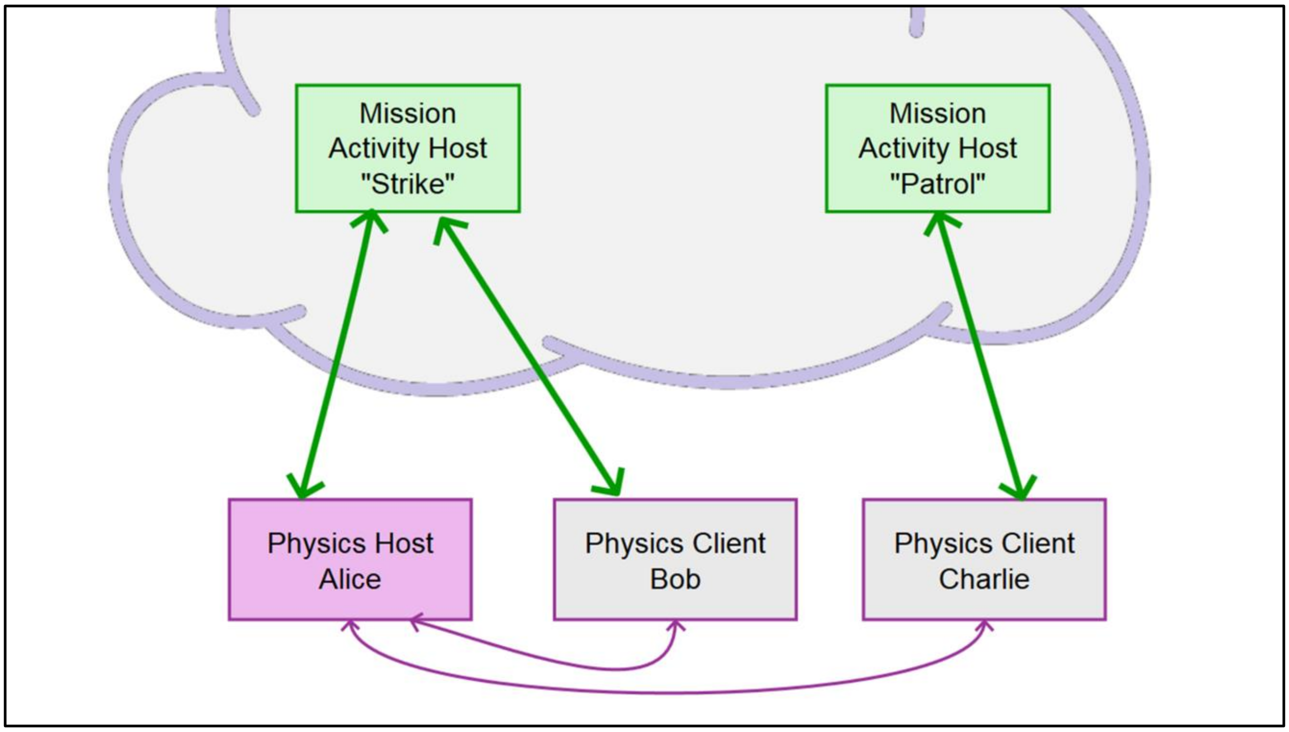


So, let's supposed we've got 3 players in this public bubble – Alice, Bob, and Charlie.

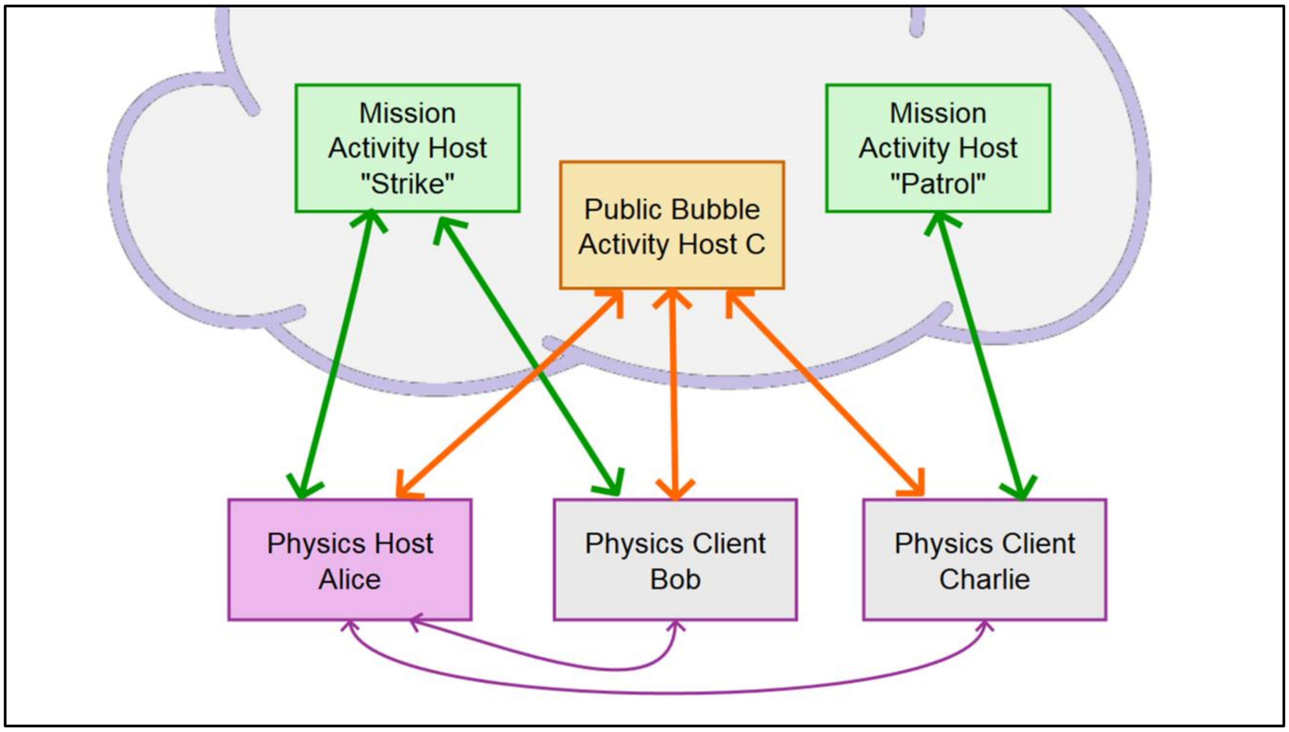
They're Peer-to-peer connected to each other using traditional Reach networking, and Alice is the Physics Host.



2 of them, Alice and Bob, are doing a Strike together. That means they're connected to a "Mission Activity Host" for their mission. The Mission Activity Host is where all the script logic for their chosen Activity lives (firing off objectives, spawning bosses in their private bubbles, etc.)


Mission Activity Host "Strike"
Mission Activity Host "Patrol"
Physics Host Alice
Physics Client Bob
Physics Client Charlie

Charlie is playing a Patrol, so while he's in the same bubble, he has a separate "Mission Activity Host", giving him local Patrol objectives.

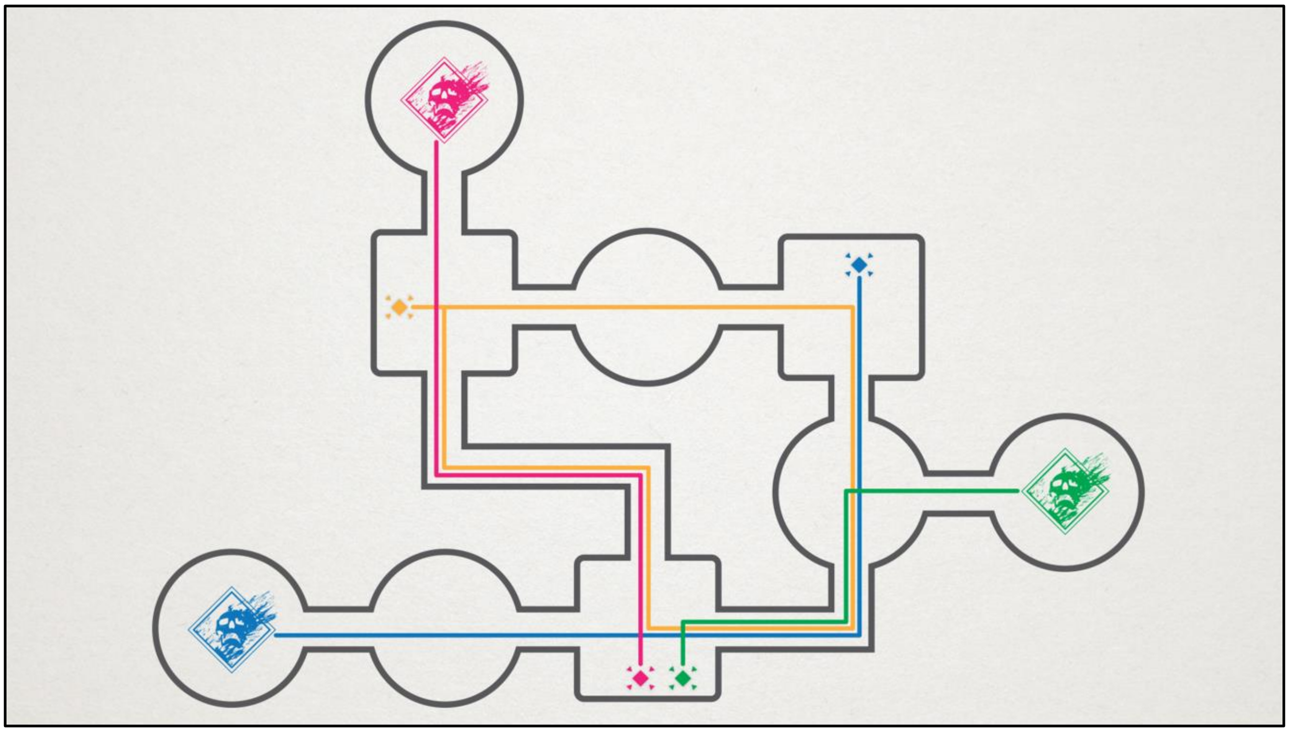

Mission Activity Host "Strike"
Public Bubble Activity Host C
Mission Activity Host "Patrol"
Physics Host Alice
Physics Client Bob
Physics Client Charlie

Finally, they're also simultaneously all connected to the same "Bubble Activity Host".

The Bubble Host is responsible for all the ambient scripting in the public bubble
it spawns and scripts all the ambient AI that you fight
It handles the respawn timing and placement of all the resource nodes and treasure chests
and it runs all the logic for all the public events.

Those are the main things that we do in our Destiny public bubbles – but technologically a Bubble Host is almost identical to a Mission Host. So you could write an arbitrarily complex script in there for a crazy public boss encounter.

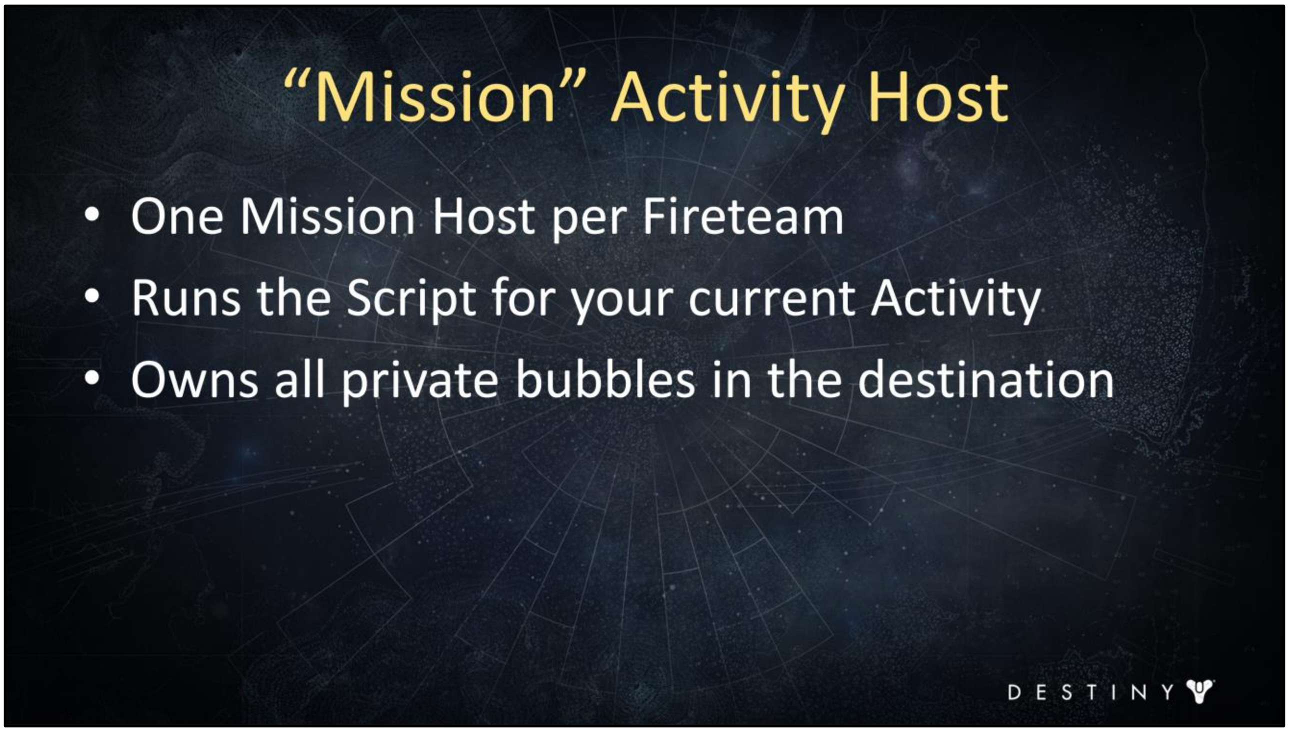
I said "Almost Identical," because there is one key difference between Mission Hosts and Bubble Hosts worth mentioning



If you go back to look at our activity line, with all these public and private bubbles - we could make them interchangable, and have an activity host for every single public bubble, and one for every single private bubble.

But there’s no need to split up the private bubbles for a given fireteam – one fireteam has their own instance of all the private bubbles in the destination, so we can simulate them all simultaneously on a single activity Host.

So you have one Bubble Host for each of the public bubbles, and then one Activity Host that manages every other bubble on the Destination (all the private bubbles).



“Mission” Activity Host

- One Mission Host per Fireteam
- Runs the Script for your current Activity
- Owns all private bubbles in the destination

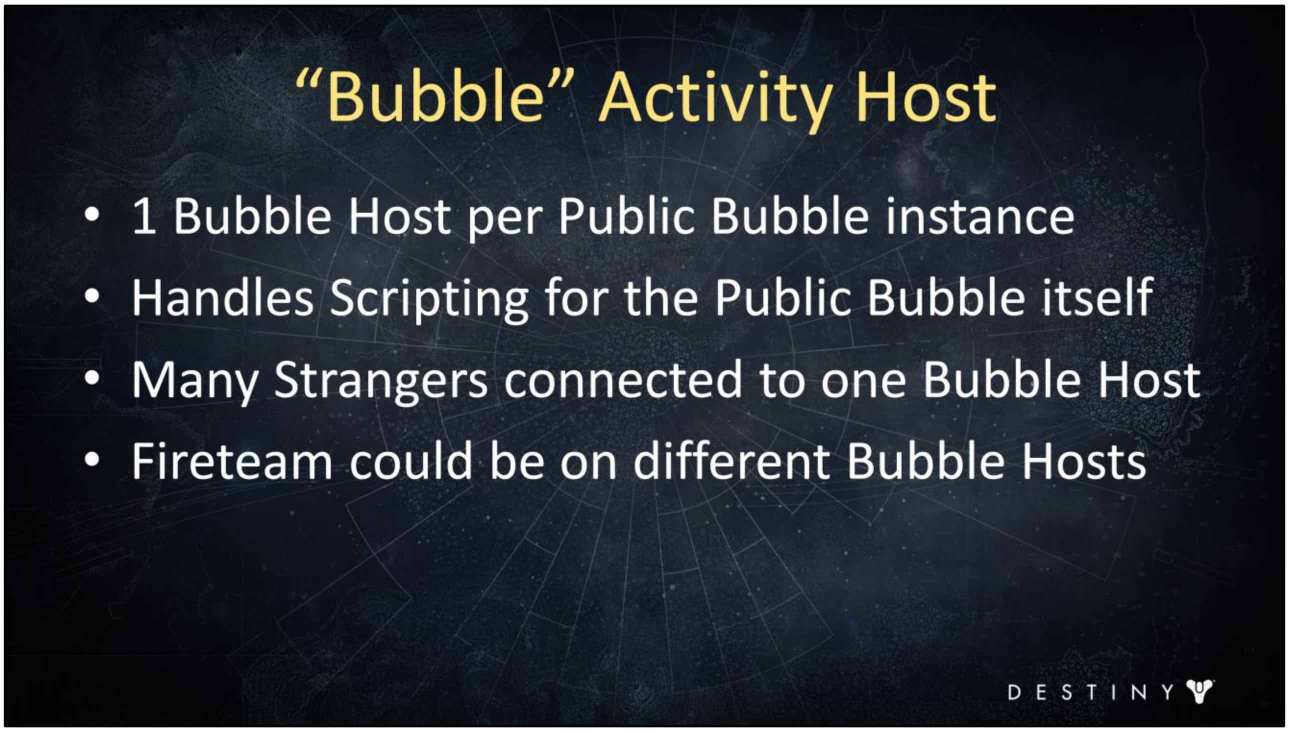
DESTINY

So, to reiterate:

Each Fireteam gets their own Mission Host. All fireteammates are always connected to the same mission host (and this may be the only connection they share, if they’re not in the same bubbles)

The Mission Host runs the Activity Script specific to the activity you’re on – like a given Story Mission Script.

The Mission Host also owns all the Private Bubbles in the destination – that way your fireteam can meet up in any of these private bubbles, but no strangers will ever show up there.



"Bubble" Activity Host

- 1 Bubble Host per Public Bubble instance
- Handles Scripting for the Public Bubble itself
- Many Strangers connected to one Bubble Host
- Fireteam could be on different Bubble Hosts

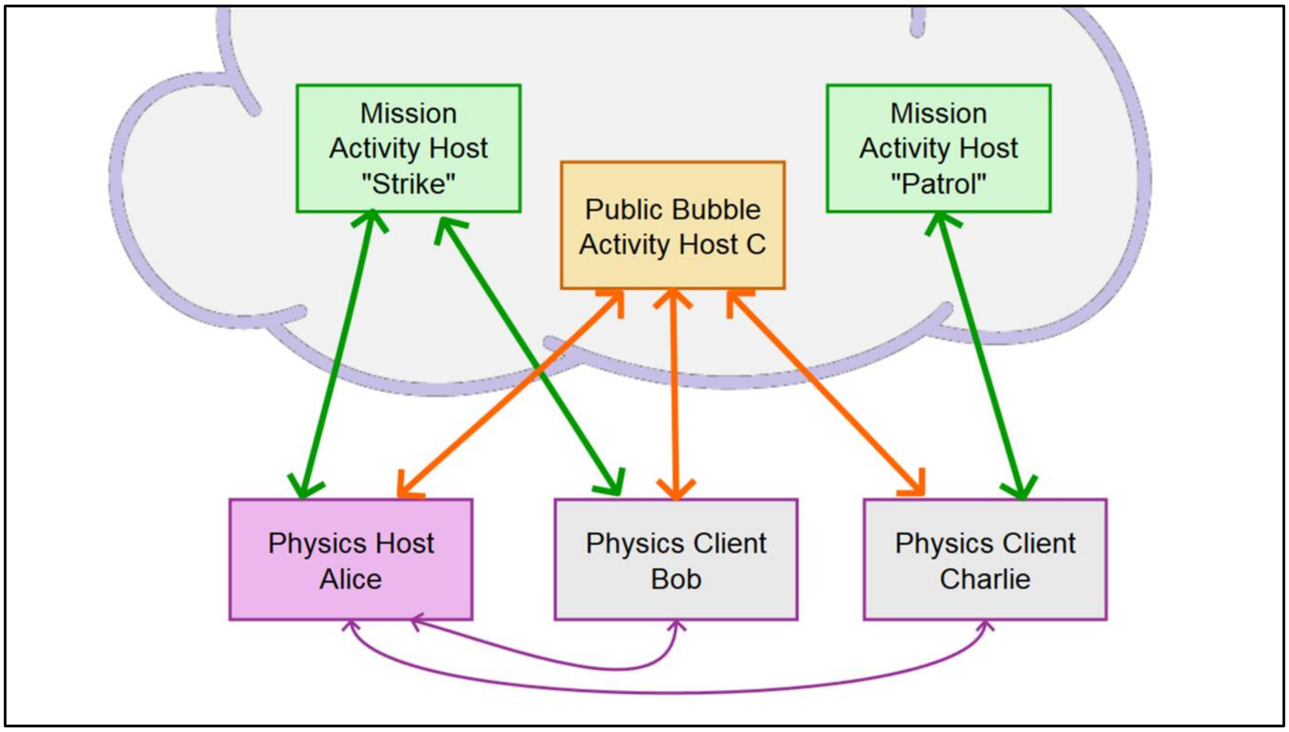
DESTINY

Then you also have "Bubble" Activity Hosts, which control public bubbles.

For every Public Bubble Instance we have running in the Destiny Universe, there's exactly one Bubble Activity Host in the cloud.

It does all the Scripting for the Public Bubble itself (Public Events, spawning treasure chests, ambient encounters)

And at any given time, a Bubble Host will likely have several strangers in it, who each have separate Mission Hosts. Additionally, every member of a fireteam could be connected to different Bubble Hosts (if they're all in different public bubbles)


Mission Activity Host "Strike"
Public Bubble Activity Host C
Mission Activity Host "Patrol"
Physics Host Alice
Physics Client Bob
Physics Client Charlie

So, going back to our Public Bubble case – you've got 3 different "Activity Hosts" all running scripts simultaneously and injecting state onto these Consoles down at the bottom. How do we keep the scripts from stomping all over each other?

As an example, you've got Charlie on the right who's on a Patrol – lots of the Patrol jobs take place in Public Bubbles, how is that ownership distributed between the Patrol Mission Host and the Public Bubble Host?

Activity Host Separation

DESTINY

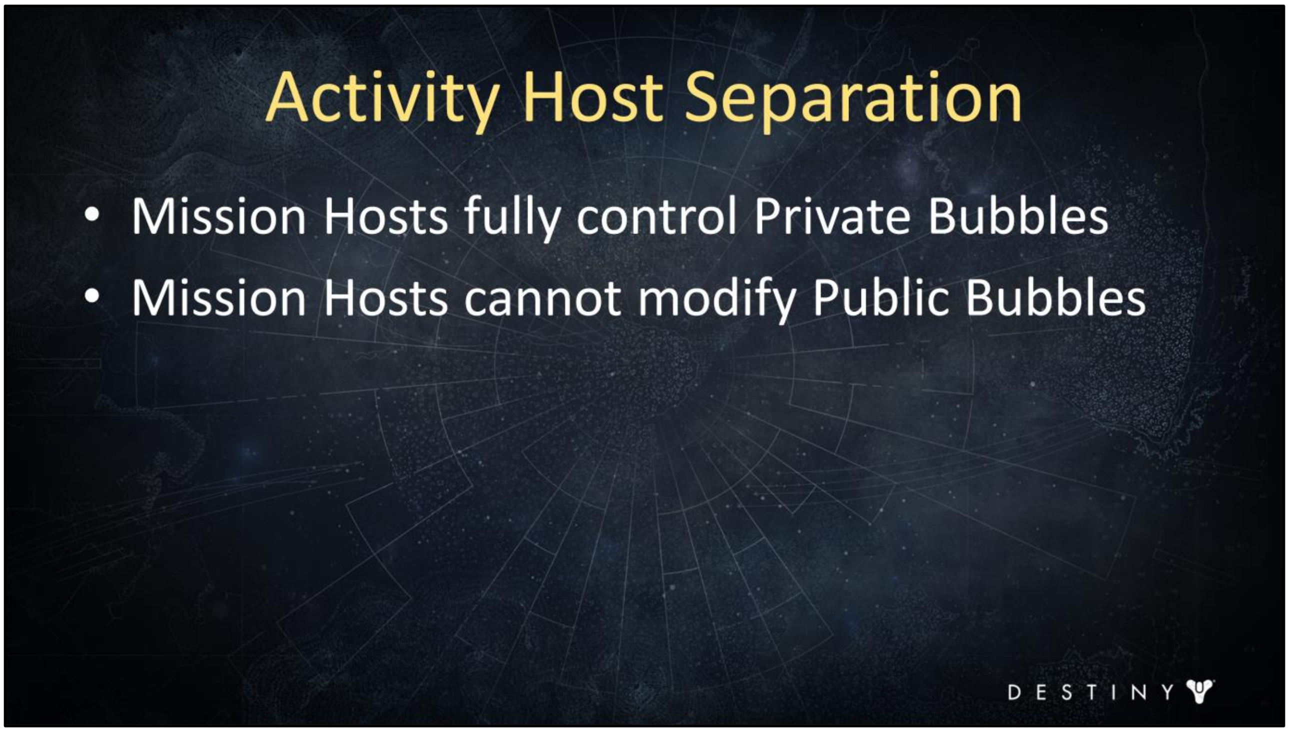
In order to keep these scripts from fighting each other, we have some policies we enforce on Mission Hosts.

Activity Host Separation

- Mission Hosts fully control Private Bubbles

DESTINY

They’re allowed to do whatever they want inside all those private bubbles that they own,



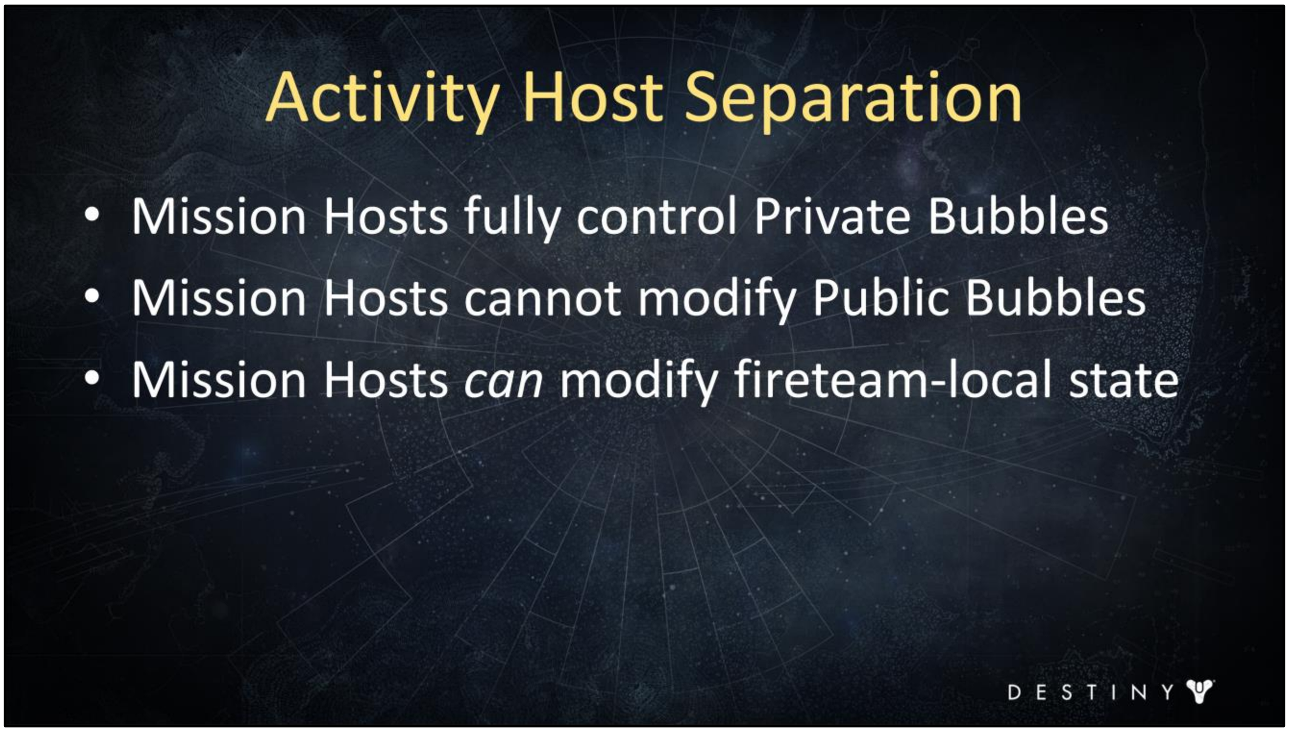
Activity Host Separation

- Mission Hosts fully control Private Bubbles
- Mission Hosts cannot modify Public Bubbles

DESTINY

but they're not allowed to instantiate or modify any networked objects inside a Public Bubble.

This way, there's only ever one Activity Host per bubble who can create or modify the shared simulation.

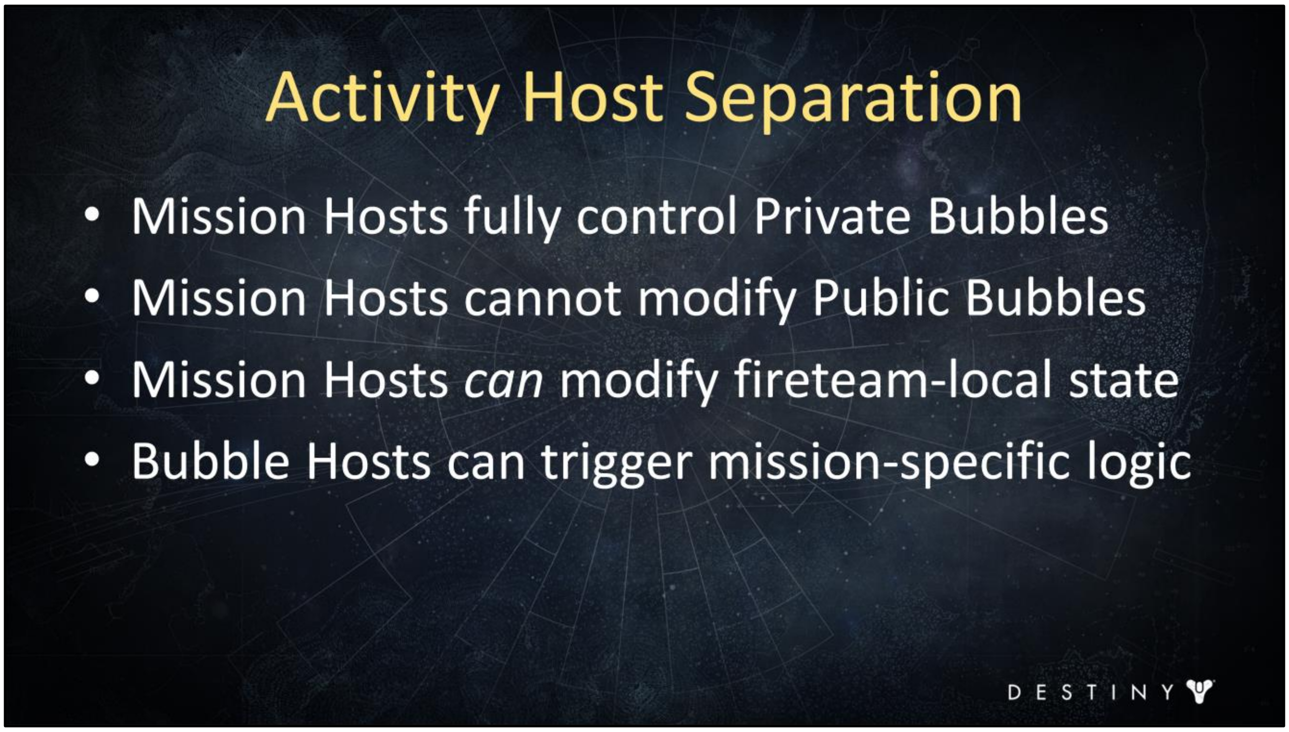


Activity Host Separation

- Mission Hosts fully control Private Bubbles
- Mission Hosts cannot modify Public Bubbles
- Mission Hosts *can* modify fireteam-local state

DESTINY

However, they are allowed to do some stuff in Public Bubbles, as long as it doesn’t affect the shared simulation. For example, they can play lines of dialog, they can set objectives, play fullscreen FX – all stuff that is local to just that fireteam.



Activity Host Separation

- Mission Hosts fully control Private Bubbles
- Mission Hosts cannot modify Public Bubbles
- Mission Hosts *can* modify fireteam-local state
- Bubble Hosts can trigger mission-specific logic

Additionally, it’s a common occurrance for the Public Bubble script to have some activity-specific logic in it. For example, it can say “If any Players in my bubble are on this specific Story Activity”, allow this interactable to be used by that player.

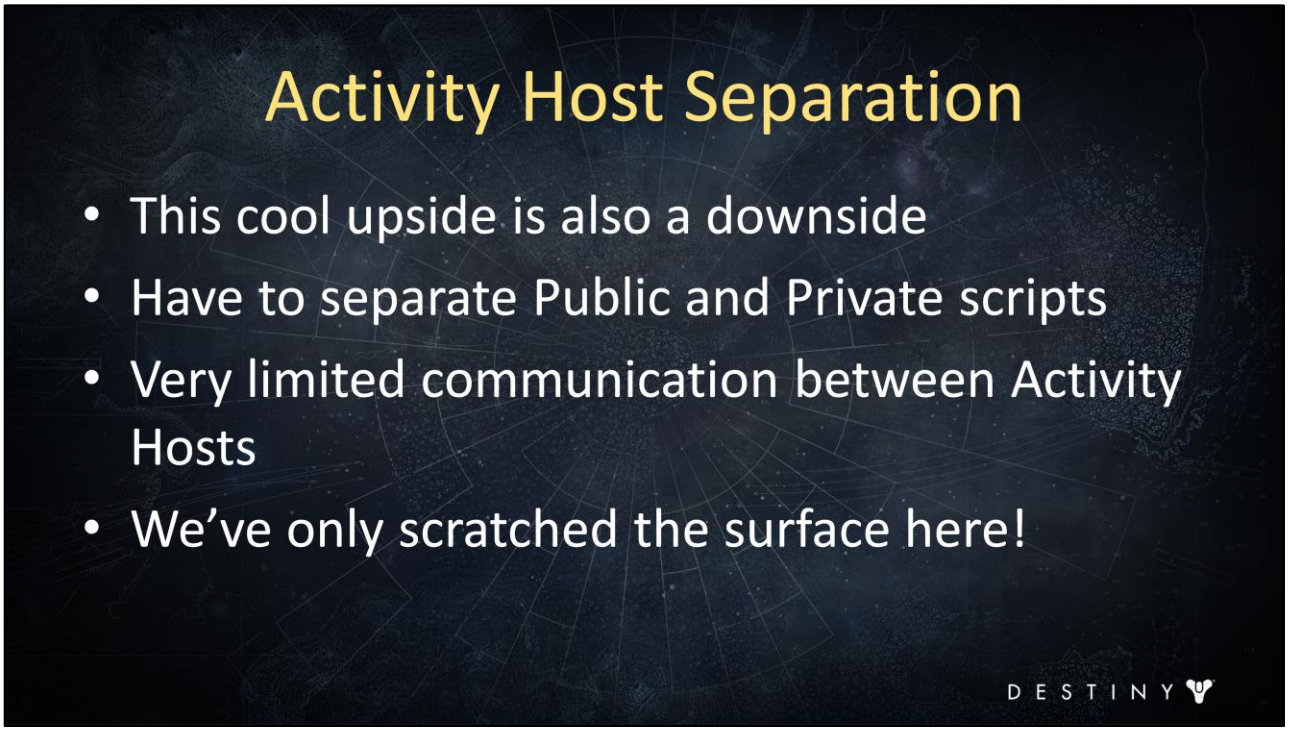
A great example of “Public Bubble Hosts Triggering Mission-Specific Logic” is the Raid entrance bubble for Vault of Glass on Venus. If you wander through that Bubble, you might see a Raid party show up, and their presence in the bubble causes the Bubble Host to trigger the first encounter of the Raid.

Special enemies from the Raid will spawn, and players in the bubble have to stand on 3 switches long enough to open a giant door.

All this logic occurs on the Public Bubble Host, *not* the Mission Host for the Raid.

The cool thing here is not only can the stranger wandering through this bubble see all of this happening (because it’s all happening on the shared Bubble Host they’re all connected to), but she can also participate in that first encounter and help out, without having to be on the Raid activity.

If she helps them complete this encounter, they’ll run off inside the door (which is a private bubble), and phase out of her game, even if she tries to follow them.



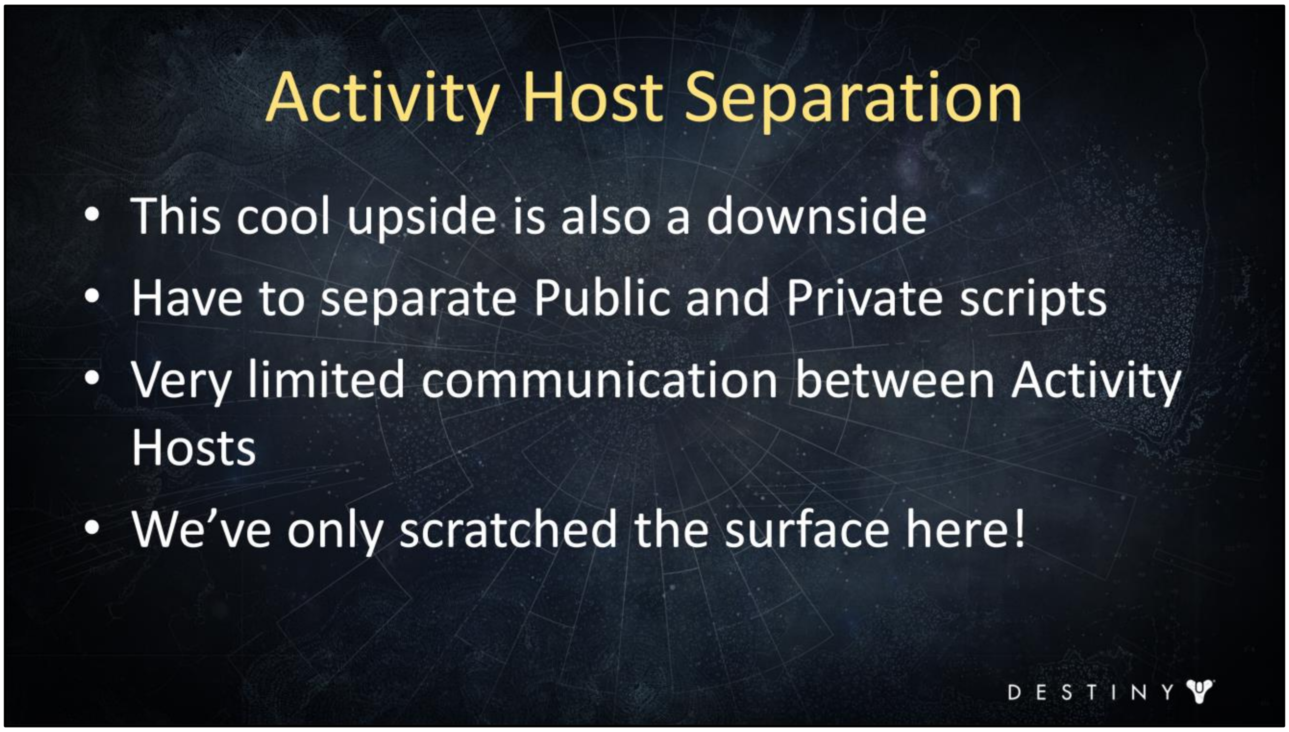
Activity Host Separation

- This cool upside is also a downside
- Have to separate Public and Private scripts
- Very limited communication between Activity Hosts
- We've only scratched the surface here!

This cool upside carries with it additional complexity – in order to do something like this, you have to split up your activity into Public and Private portions.

The Public Portions live on their respective Public Bubble Hosts, the private portions live on the Mission Host.

And for reasonably complex interactions between the two, you have to pass Global Flags back and forth between them, because there's very limited communication between Activity Hosts



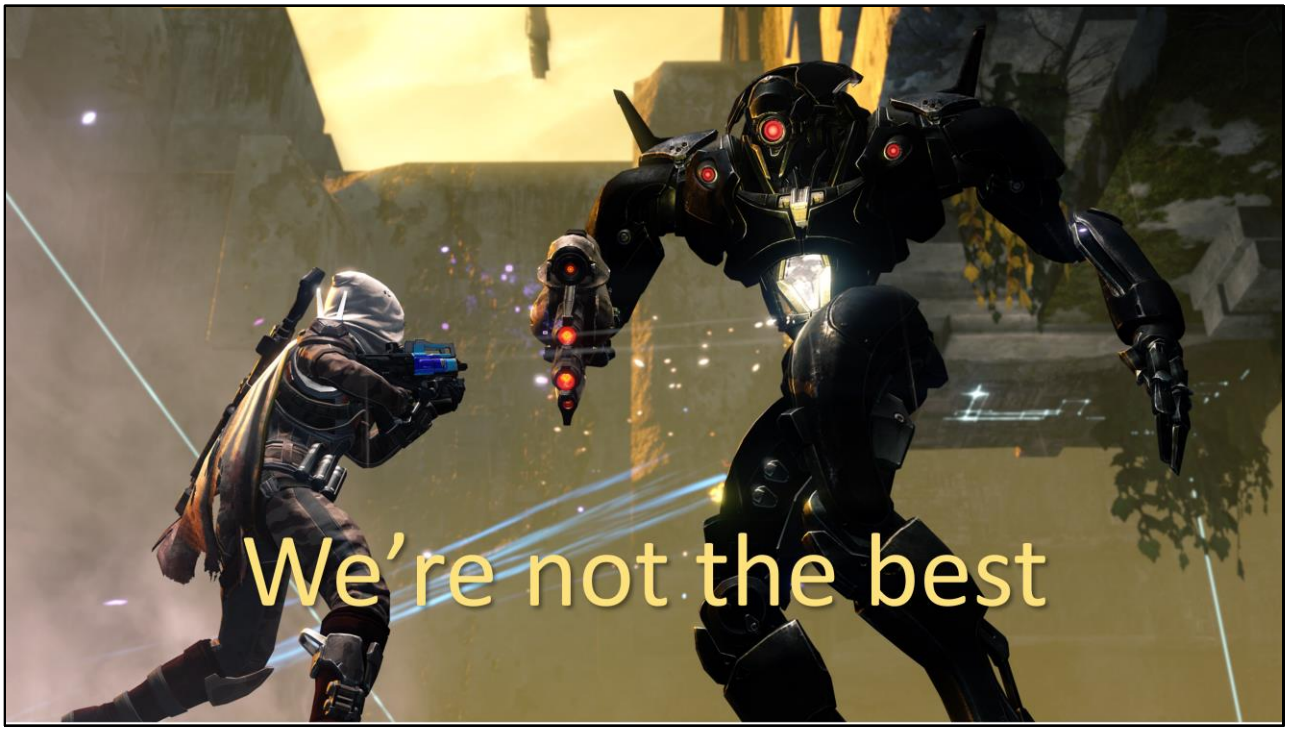
Activity Host Separation

- This cool upside is also a downside
- Have to separate Public and Private scripts
- Very limited communication between Activity Hosts
- We've only scratched the surface here!

DESTINY

This constraint (plus the fact that a single Public Bubble Script needs to know how to arbitrate all the different possible events that could happen inside it simultaneously) kept us pretty conservative with what we do in Public Bubbles for Destiny so far.

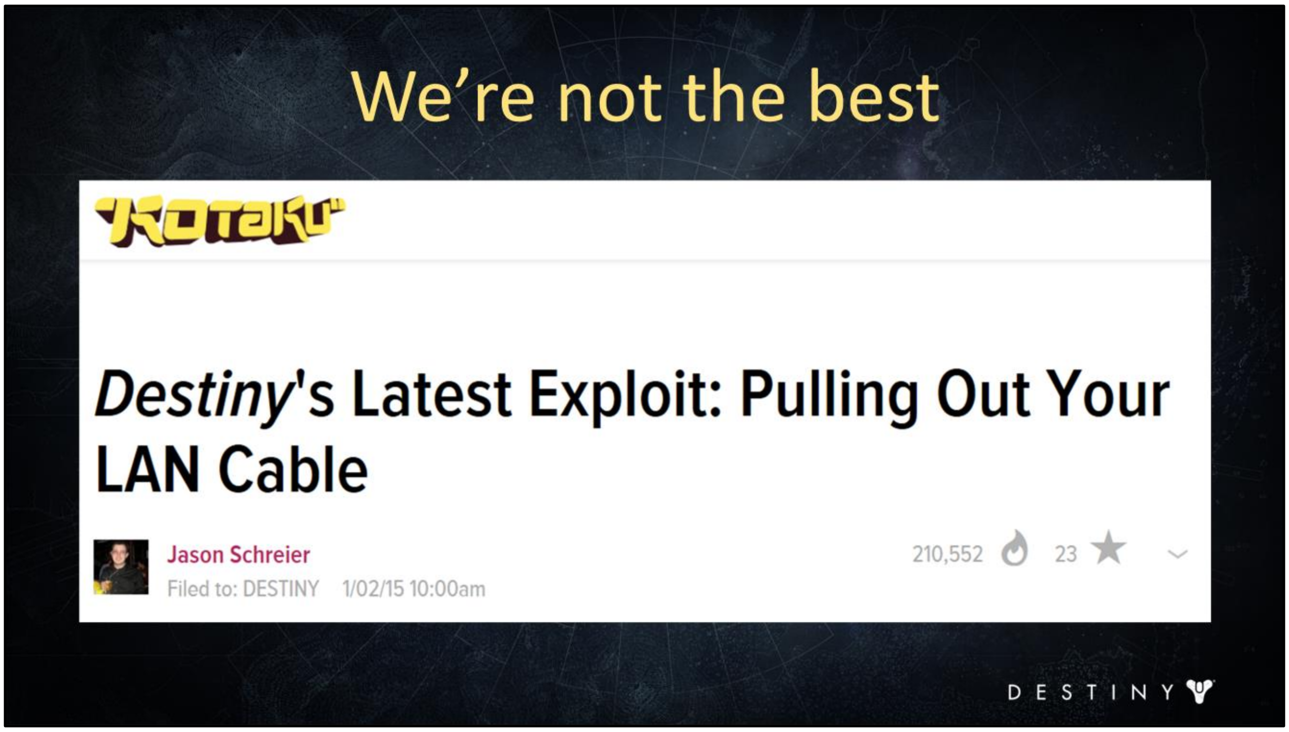
I believe we've only just scratched the surface here, and there's a lot of really ambitious stuff we can start doing in Public Bubbles for future releases, now that we're starting to get our sealegs.



And so while I do think we've managed to create some pretty cool tech, and have even more ambitious designs coming down the pipe, it's also worth spending a little time talking about the downsides and challenges we've encountered.

A phrase we like to use internally a lot at Bungie for this is "We're not the Best."

We’re not the best


KOTAKU

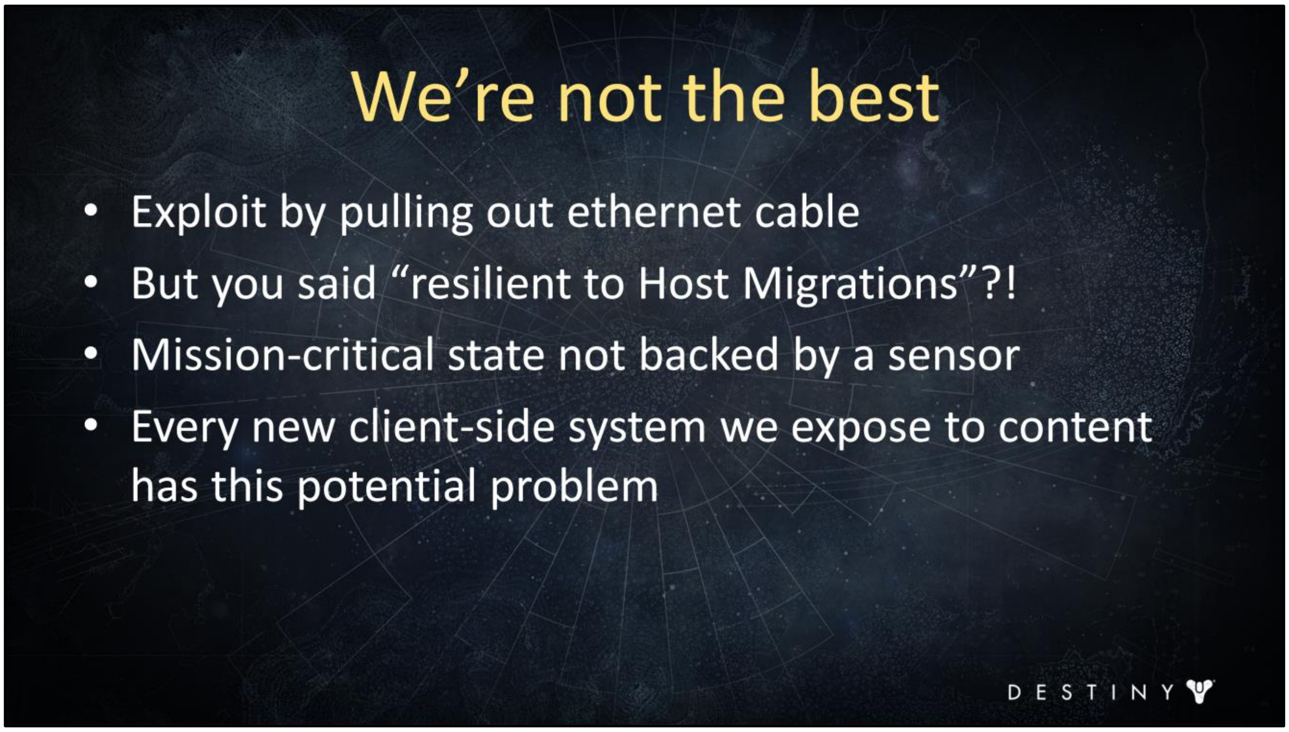
Destiny's Latest Exploit: Pulling Out Your LAN Cable

Jason Schreier
Filed to: DESTINY 1/02/15 10:00am
210,552 23

Yeah, so….

Back in January players found the first repeatable Host Migration exploit I’m aware of – they could cheese the final boss fight in our first DLC Raid by pulling out their network cable at a key moment. That rapidly turned into the defacto way to defeat the Raid.

If this makes you think “So isn’t all of this talk’s “resilience to host migration” stuff bullshit?”, I don’t blame you.



We’re not the best

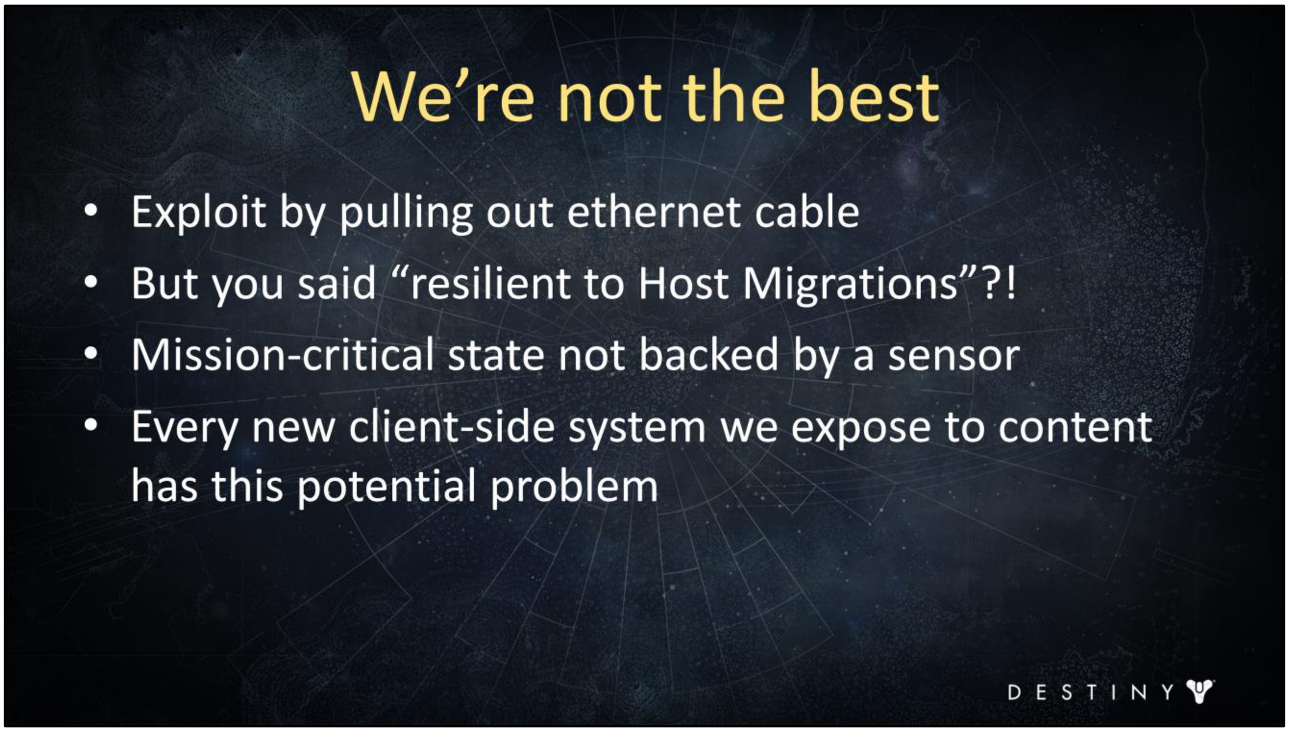
- Exploit by pulling out ethernet cable
- But you said “resilient to Host Migrations”?!
- Mission-critical state not backed by a sensor
- Every new client-side system we expose to content has this potential problem

DESTINY

This case was an interesting bug – we have several client-side systems that don’t run on the Activity Host for handling Damage State and AI Behavior transitions.

In this case a mission-critical portion of the Raid boss logic was setup entirely using those client side systems, so it never got persisted or secured by our Activity Host model, even though we had sensors built to secure this sort of state transition.

This is a general problem we have – any time we expose new complexity that is client-side only, you run the risk of anyone in the studio unintentionally rigging up a Rube Goldberg machine that runs entirely on the client, and then making it mission-critical.



We’re not the best

- Exploit by pulling out ethernet cable
- But you said “resilient to Host Migrations”?!
- Mission-critical state not backed by a sensor
- Every new client-side system we expose to content has this potential problem

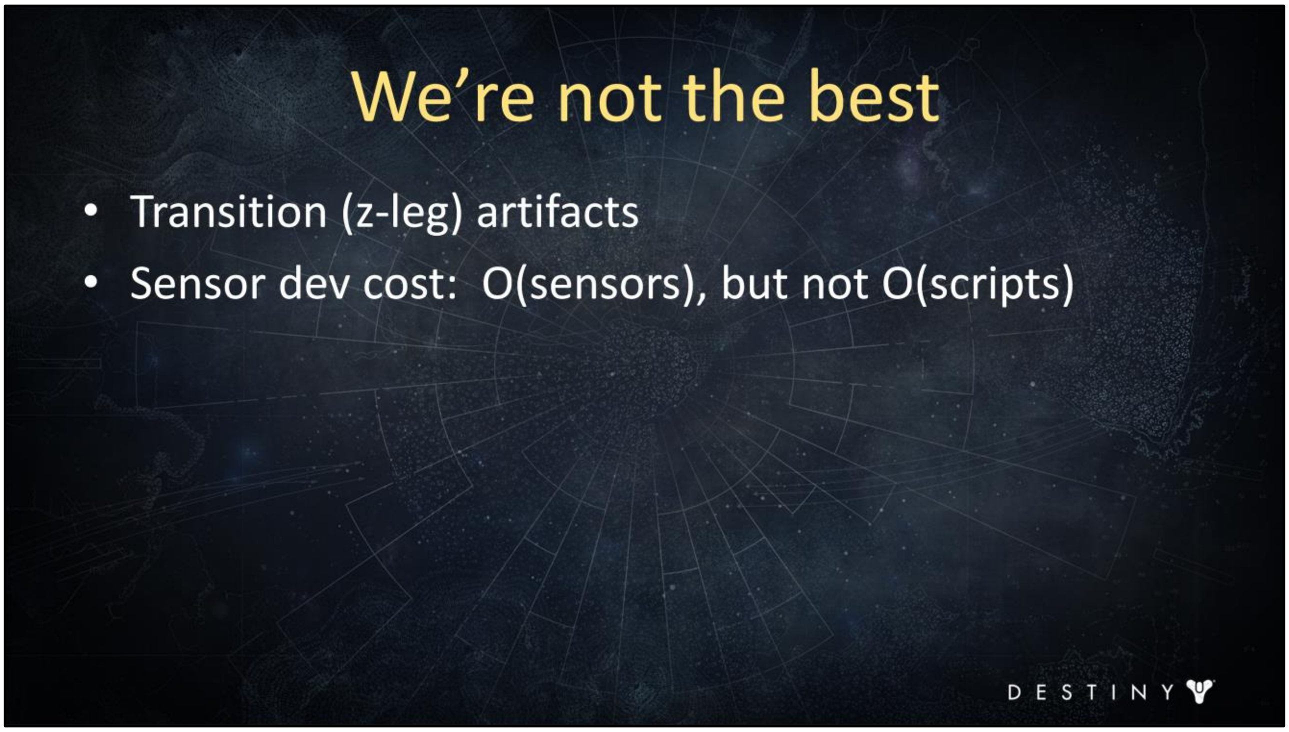
And to be clear – I’m not selling out the content creators here. It’s our job as the engineers to create intuitive systems and communicate their usage patterns well to the whole design team. I view this case as an engineering failing, for not paying close enough attention to and auditing the more complex Raid Scripts and damage setups.

Ideally, we all want to live in a world where Designers don’t ever have to think about Host Migrations, or about which machine their logic is executing on, and it should all “just work.”

We’re not the best

- Transition (z-leg) artifacts

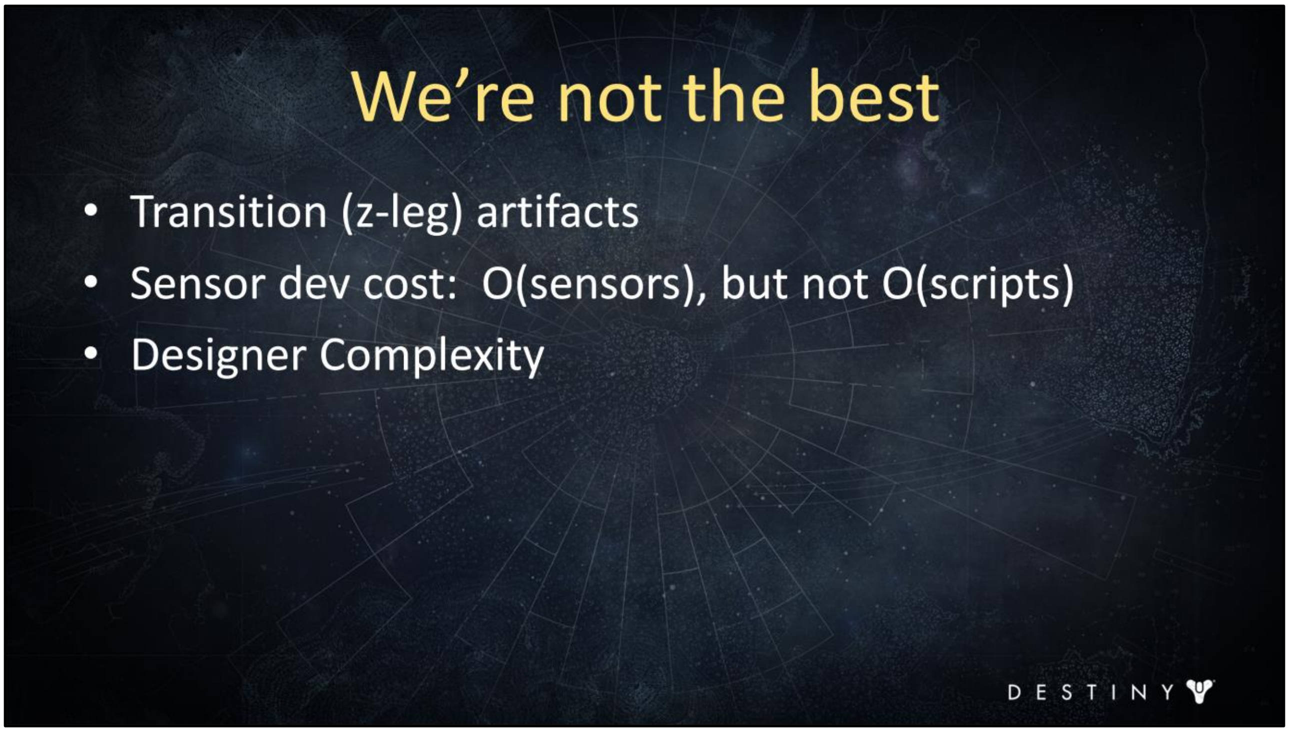
You also get some weird player artifacts – if you run through a public-bubble transition with another player in your fireteam, you’ll see them disappear and then respawn – that’s because when you bubble swap you’re fully disconnecting with that player, leaving their game, then respawning them in the new game that you connect to, on the other side of the transition.



We’re not the best

- Transition (z-leg) artifacts
- Sensor dev cost: O(sensors), but not O(scripts)

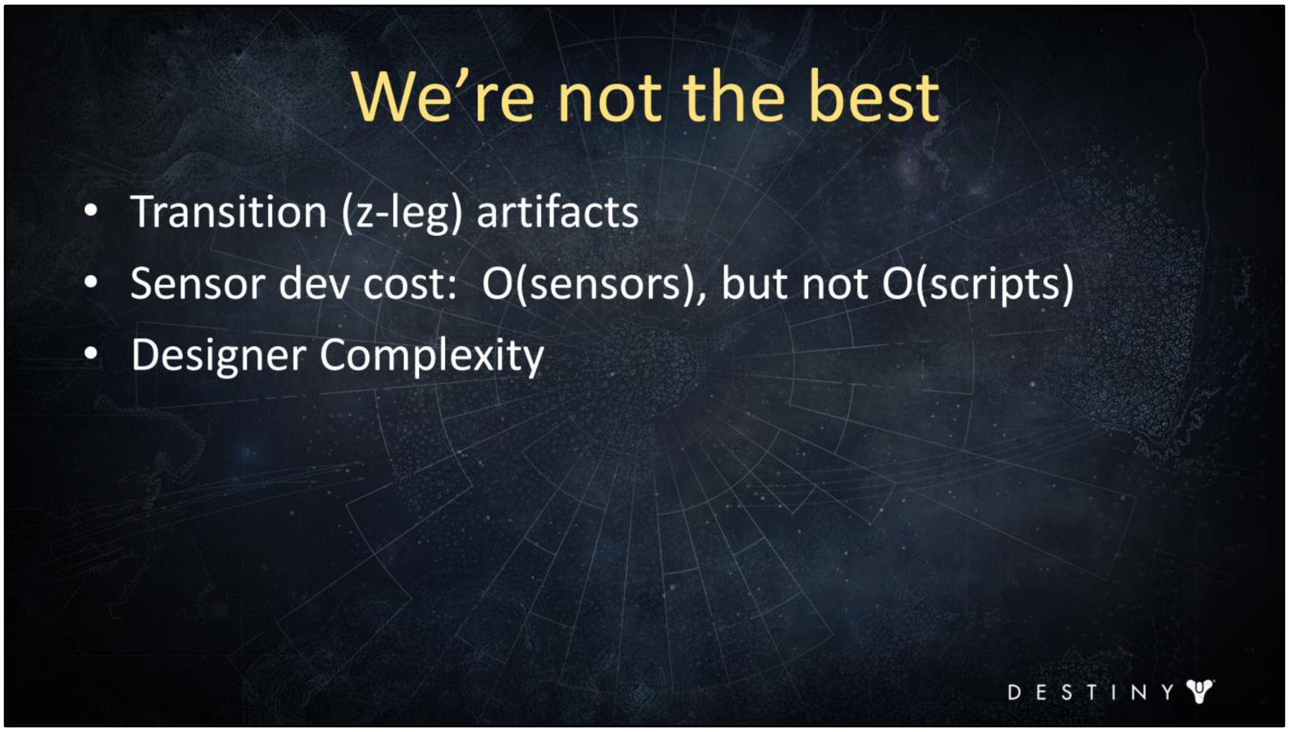
DESTINY



We’re not the best

- Transition (z-leg) artifacts
- Sensor dev cost: O(sensors), but not O(scripts)
- Designer Complexity

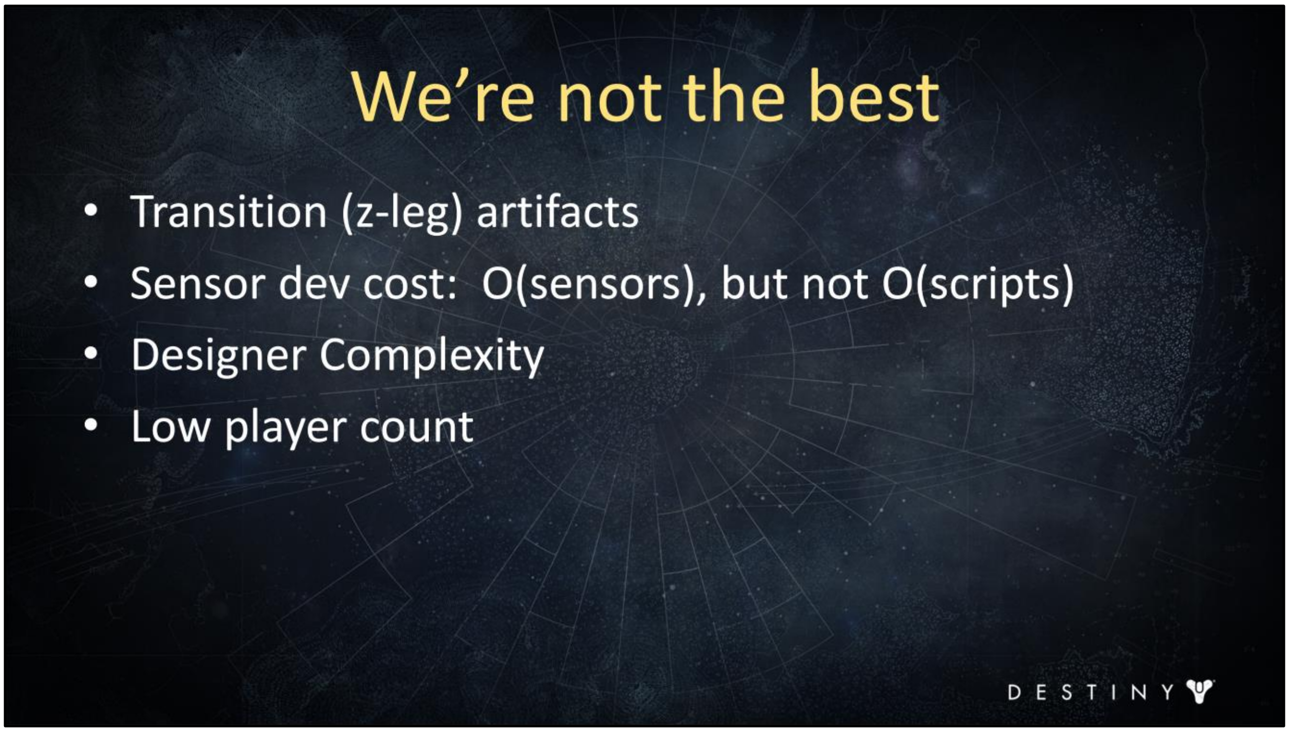
DESTINY



We’re not the best

- Transition (z-leg) artifacts
- Sensor dev cost: O(sensors), but not O(scripts)
- Designer Complexity

DESTINY



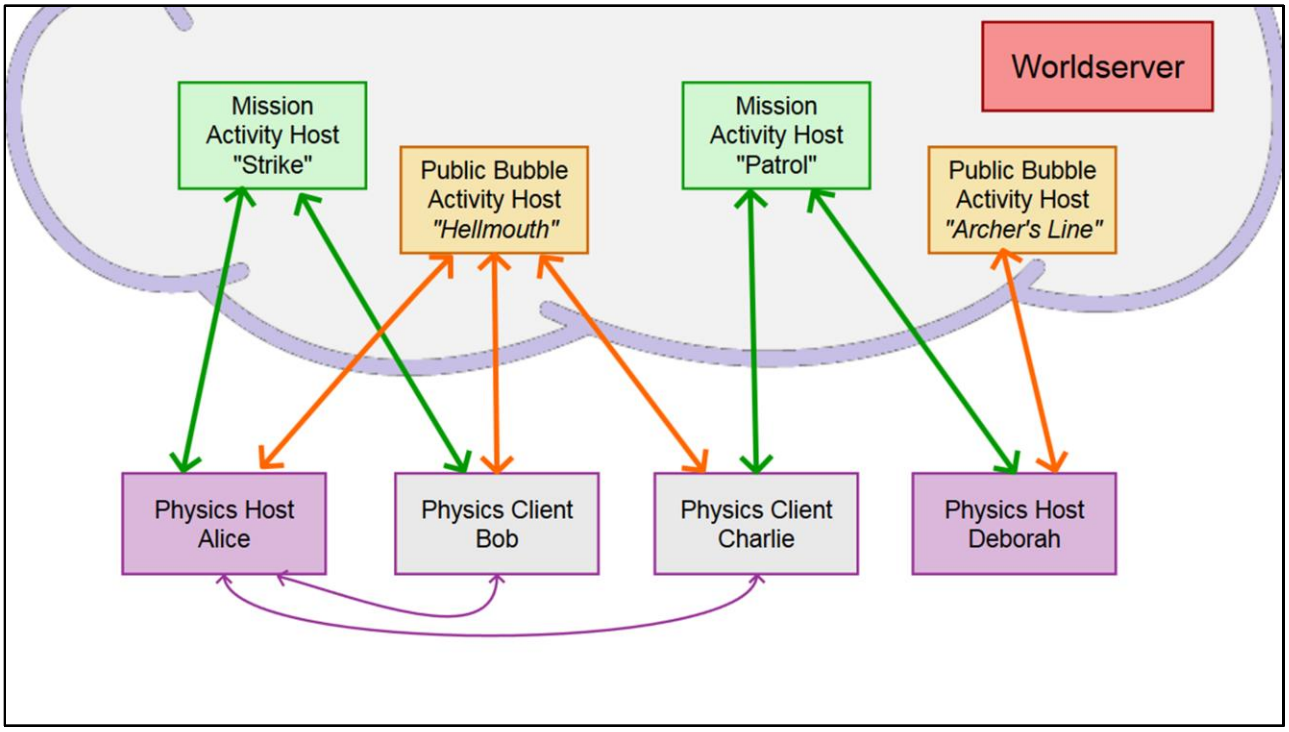
We’re not the best

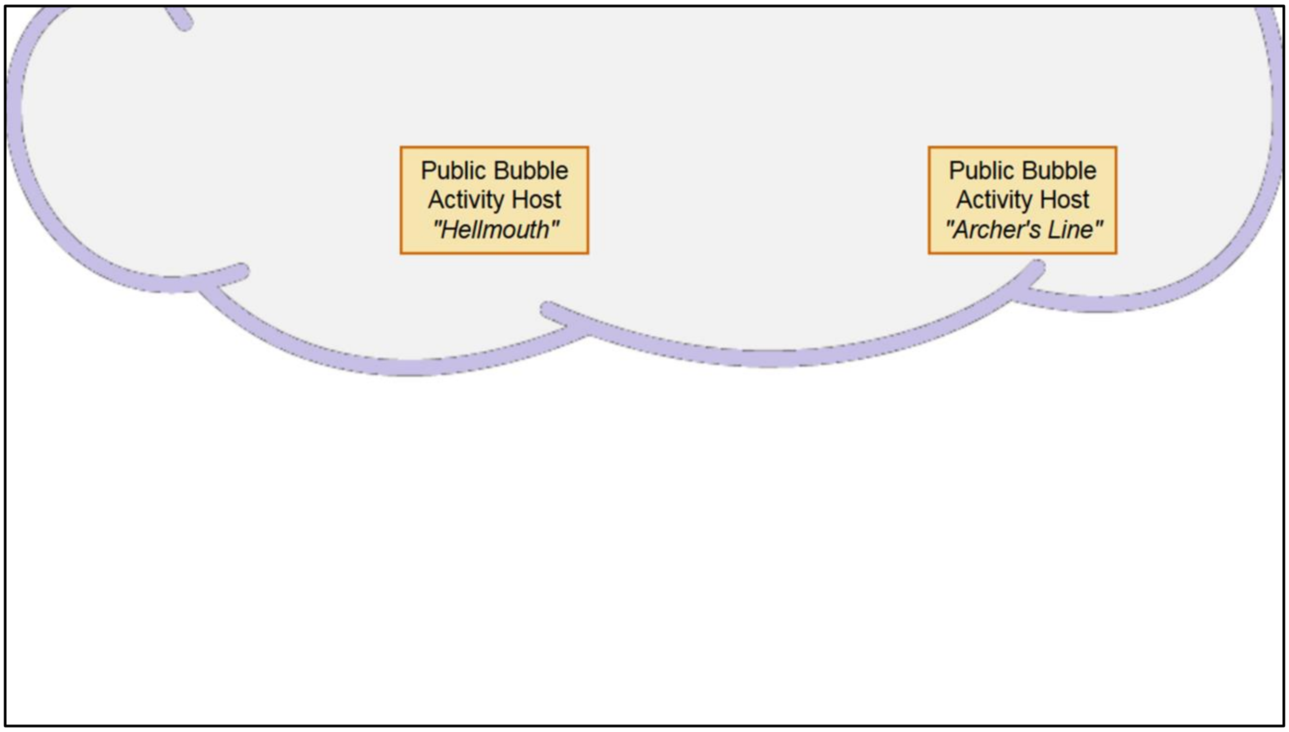
- Transition (z-leg) artifacts
- Sensor dev cost: O(sensors), but not O(scripts)
- Designer Complexity
- Low player count

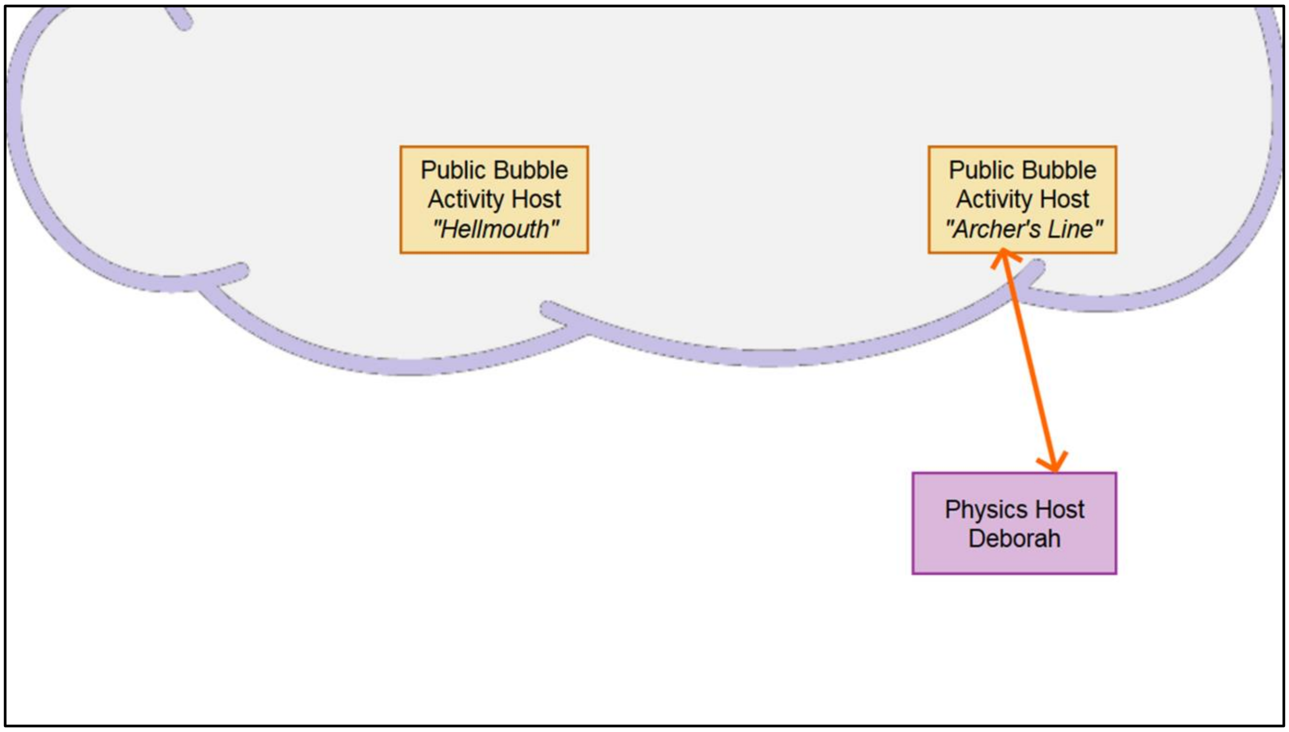
DESTINY

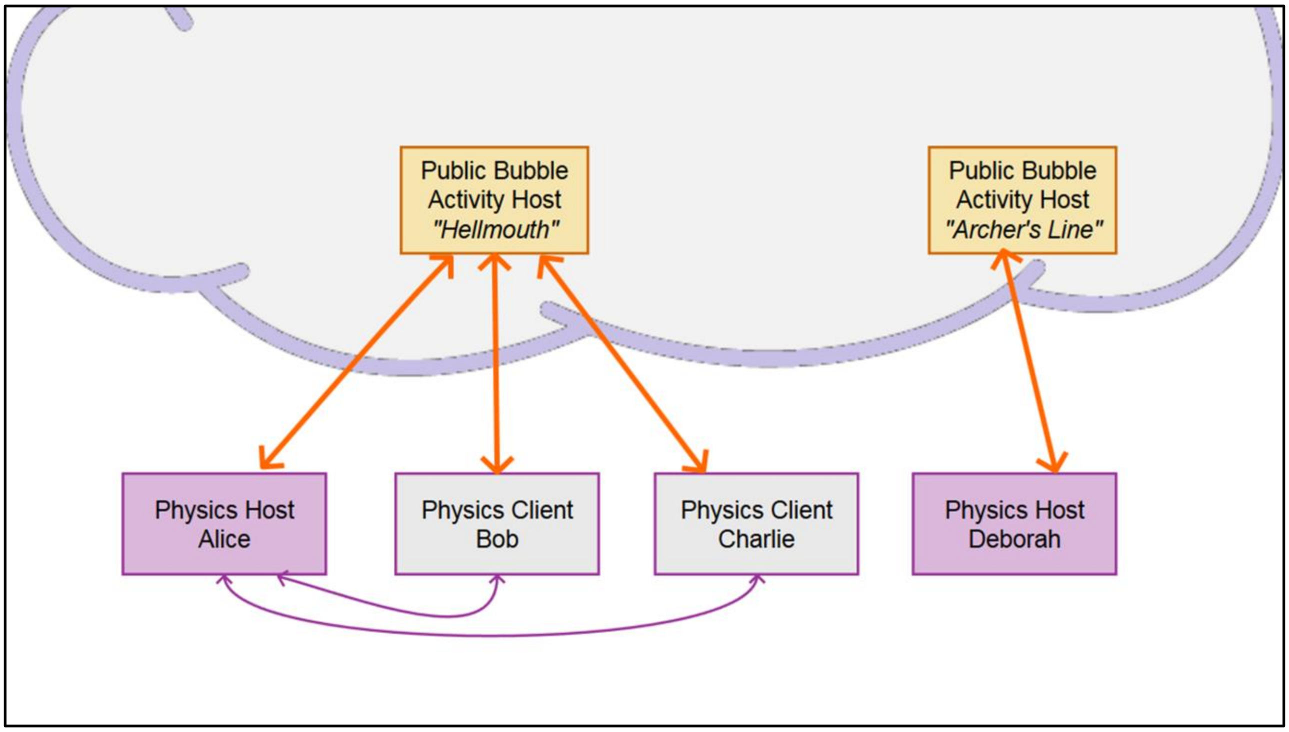
Conclusion

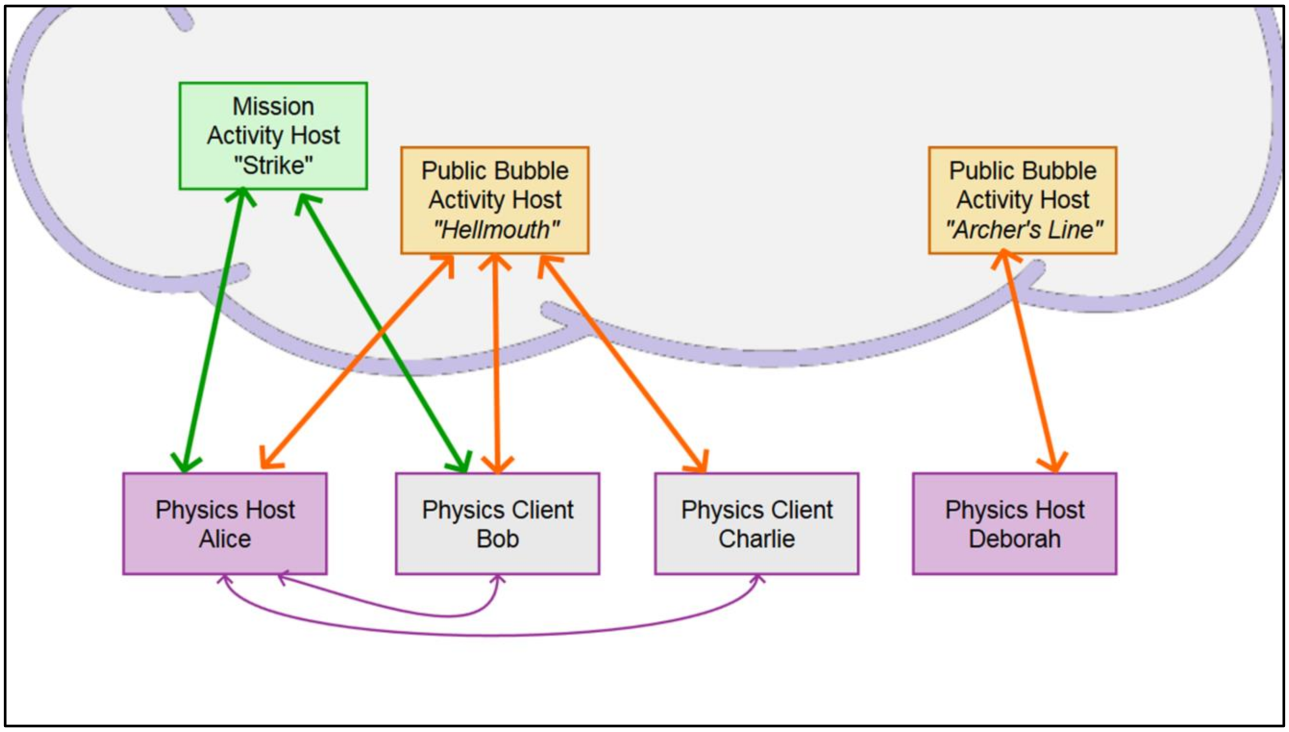
So, to conclude, let's walk through a summary example of the entire ecosystem

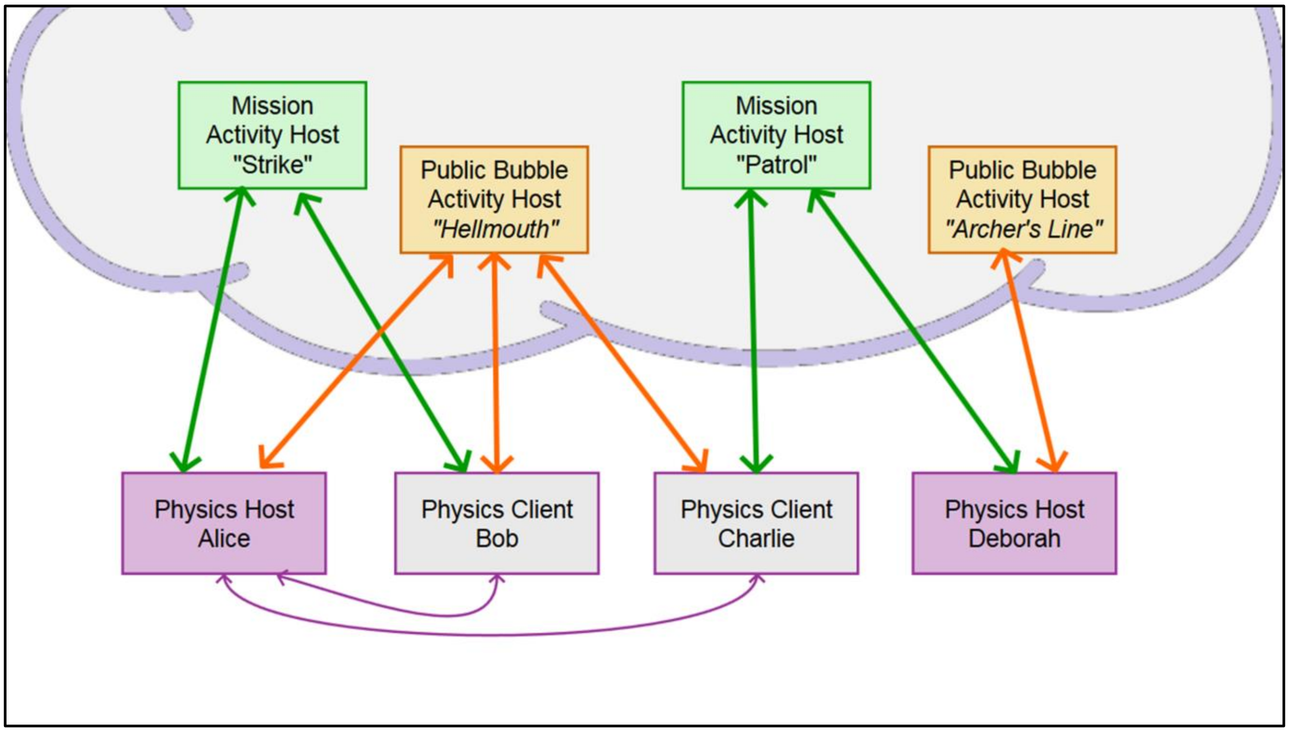

Worldserver
Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Mission Activity Host "Patrol"
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah

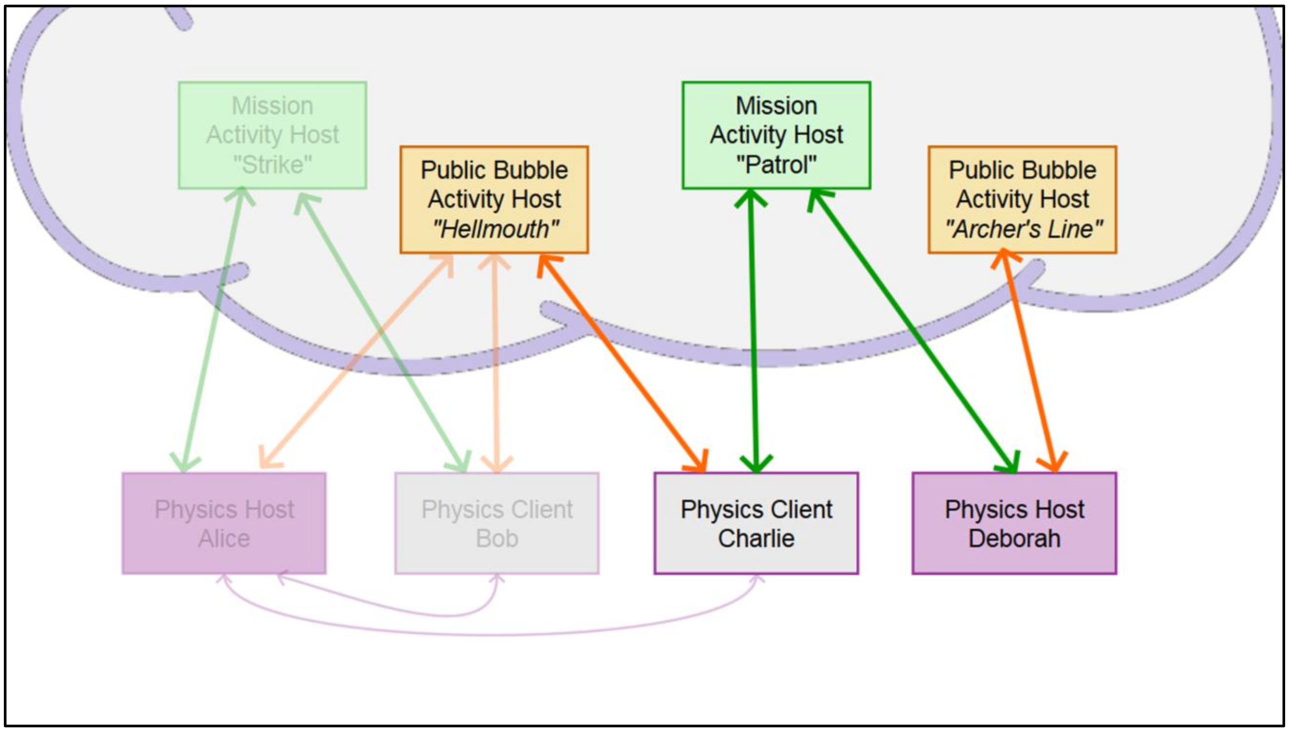

Public Bubble
Activity Host
"Hellmouth"
Public Bubble
Activity Host
"Archer's Line"

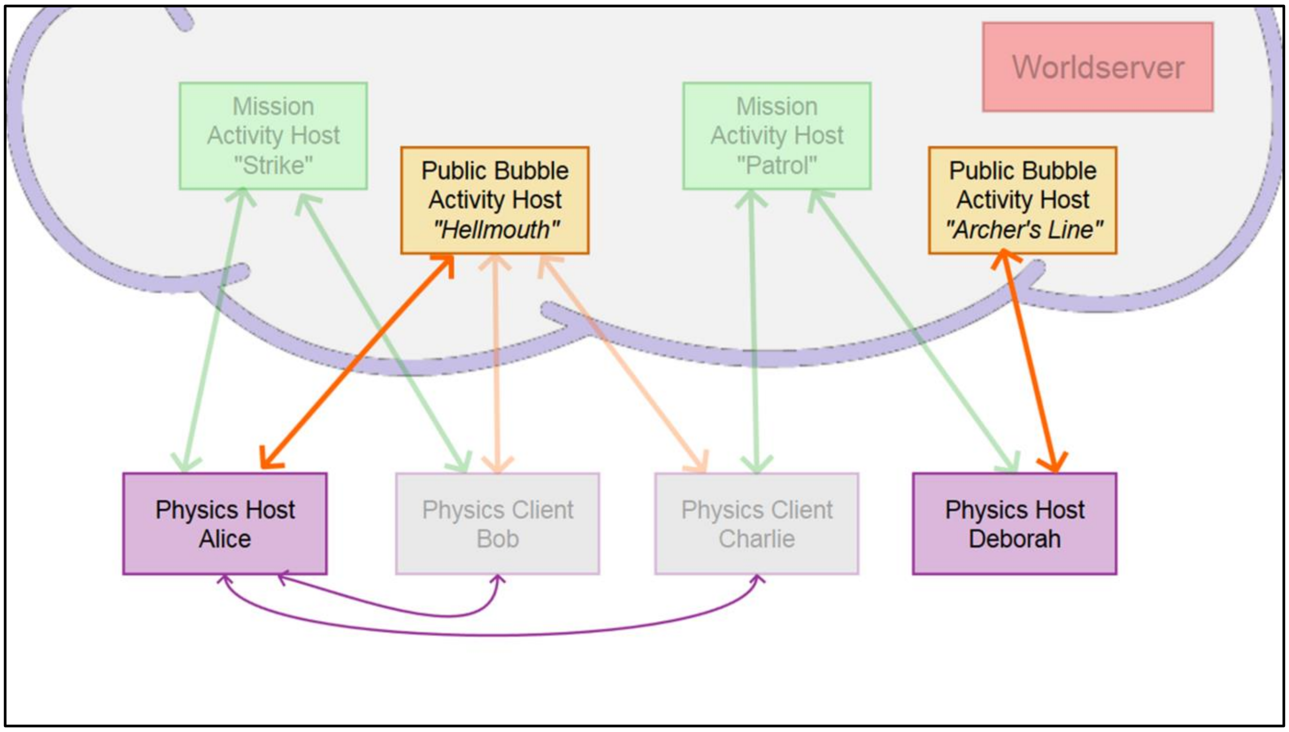

Public Bubble Activity Host *"Hellmouth"*
Public Bubble Activity Host *"Archer's Line"*
Physics Host Deborah

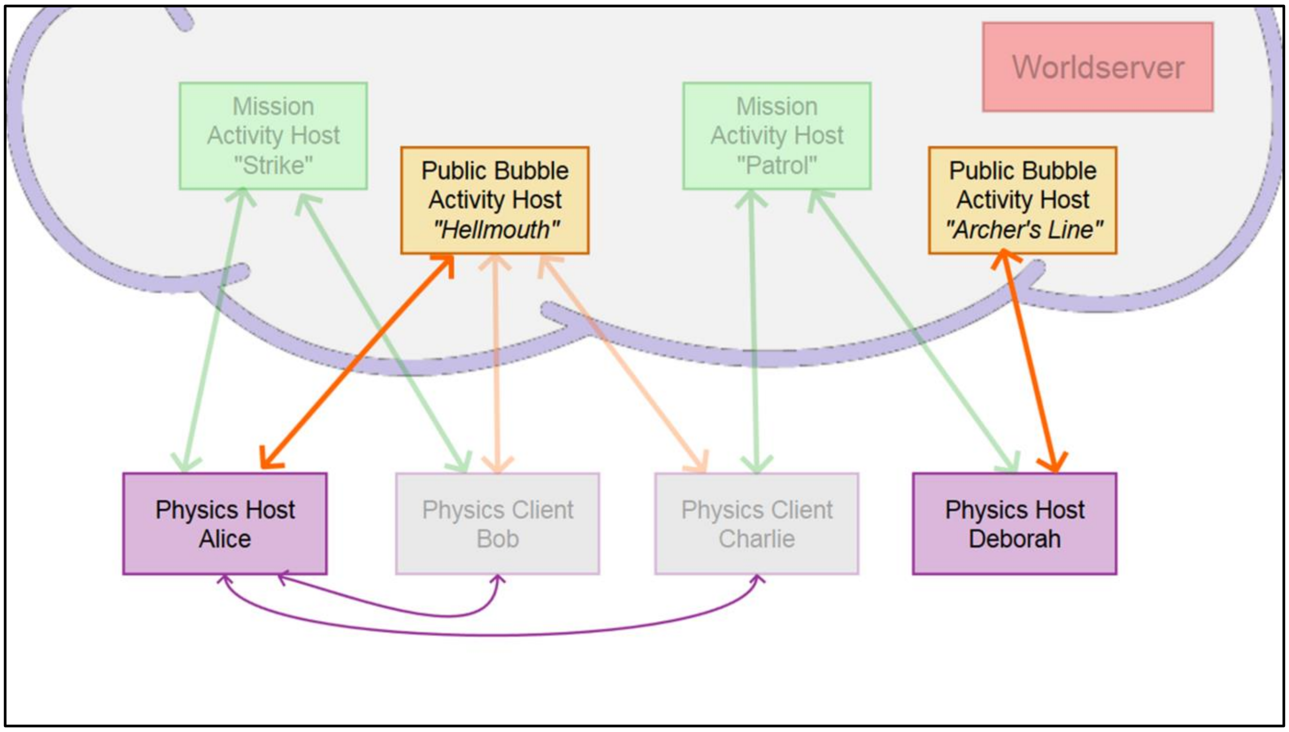

Public Bubble
Activity Host
"Hellmouth"
Public Bubble
Activity Host
"Archer's Line"
Physics Host
Alice
Physics Client
Bob
Physics Client
Charlie
Physics Host
Deborah

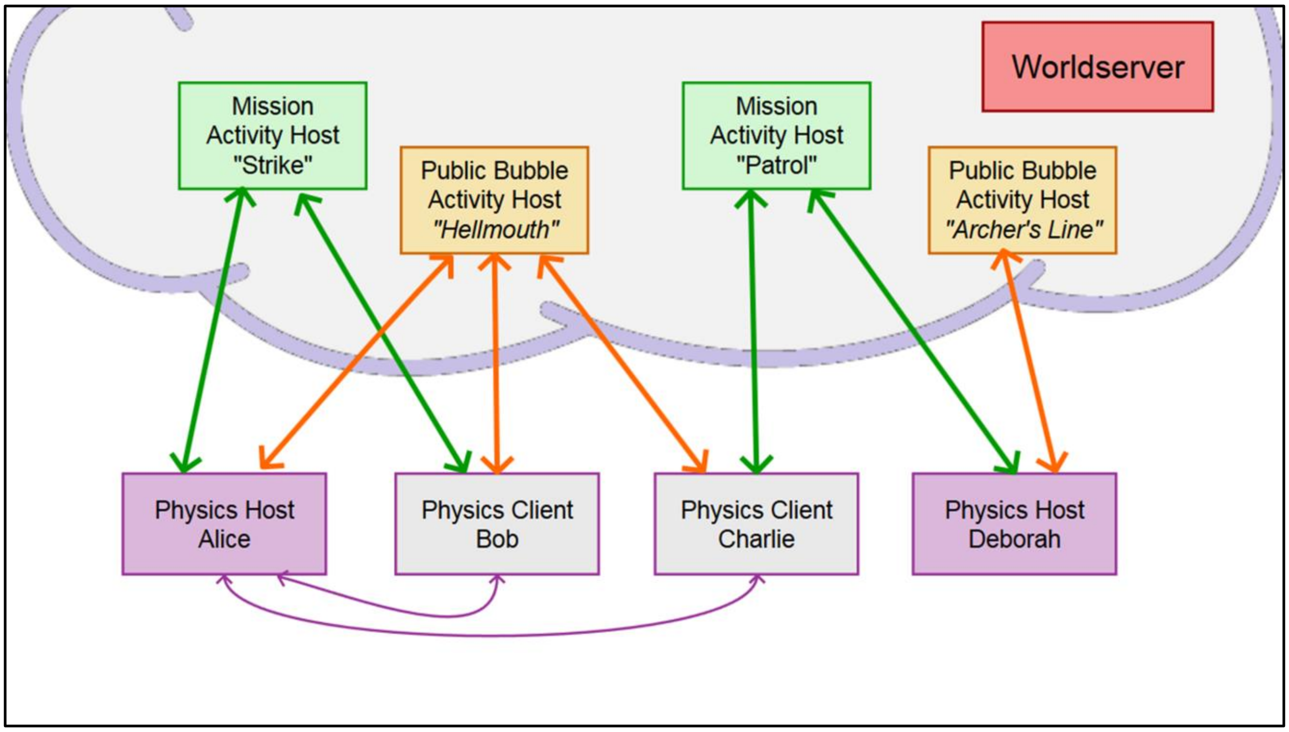

Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah


Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Mission Activity Host "Patrol"
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah


Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Mission Activity Host "Patrol"
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah


Worldserver
Mission Activity Host "Strike"
Public Bubble Activity Host "Hellmouth"
Mission Activity Host "Patrol"
Public Bubble Activity Host "Archer's Line"
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah


Worldserver
Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Mission Activity Host "Patrol"
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah


Worldserver
Mission Activity Host "Strike"
Public Bubble Activity Host *"Hellmouth"*
Mission Activity Host "Patrol"
Public Bubble Activity Host *"Archer's Line"*
Physics Host Alice
Physics Client Bob
Physics Client Charlie
Physics Host Deborah

W E ' R E   H I R I N G

WWW.BUNGIE.NET/CAREERS

CAREERS@BUNGIE.COM