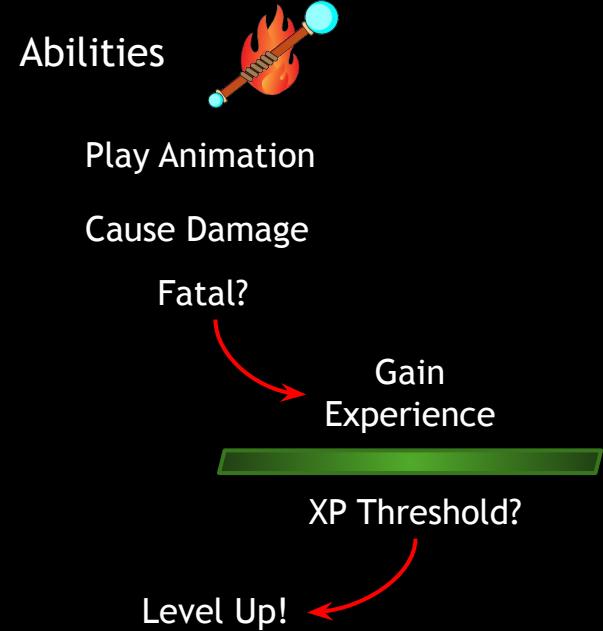
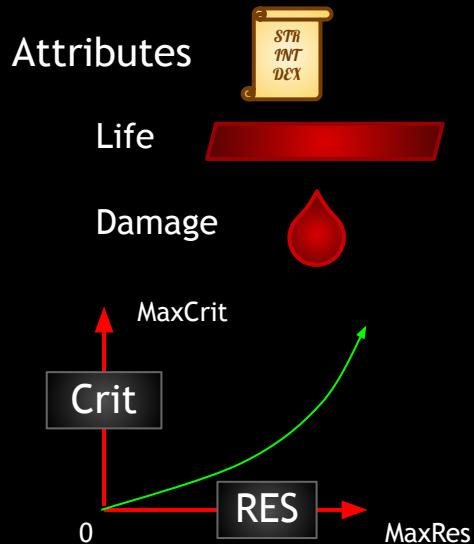


# Gameplay Ability System

**Unreal Engine Documentation:** The **Gameplay Ability System** is a highly flexible framework for building the types of abilities and attributes that you might find in an RPG or MOBA title. You can build actions or passive aggressive abilities for the characters in your games to use, and status effects that can build up or wear down various attributes as a result of these actions, additionally you can implement "cooldown" timers or resource costs to regulate the usage of these actions, change the level of the ability and its effects at each level, activate particle, sound effects, and more. The **Gameplay Ability System** can help you to design, implement, and efficiently network in-game abilities from as simple as jumping to as complex as your favorite character's ability set in any modern RPG or MOBA title.

# Gameplay Ability System



# Gameplay Ability System



Effects



Change Attribute Values



Cues



Particles

Sounds



(MOBA)



# Gameplay Ability System



Ability System Component



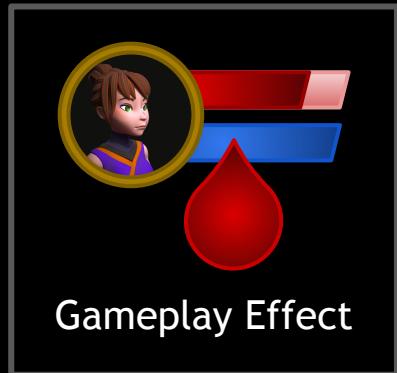
Attribute Set



Gameplay Ability



Ability Task



Gameplay Effect

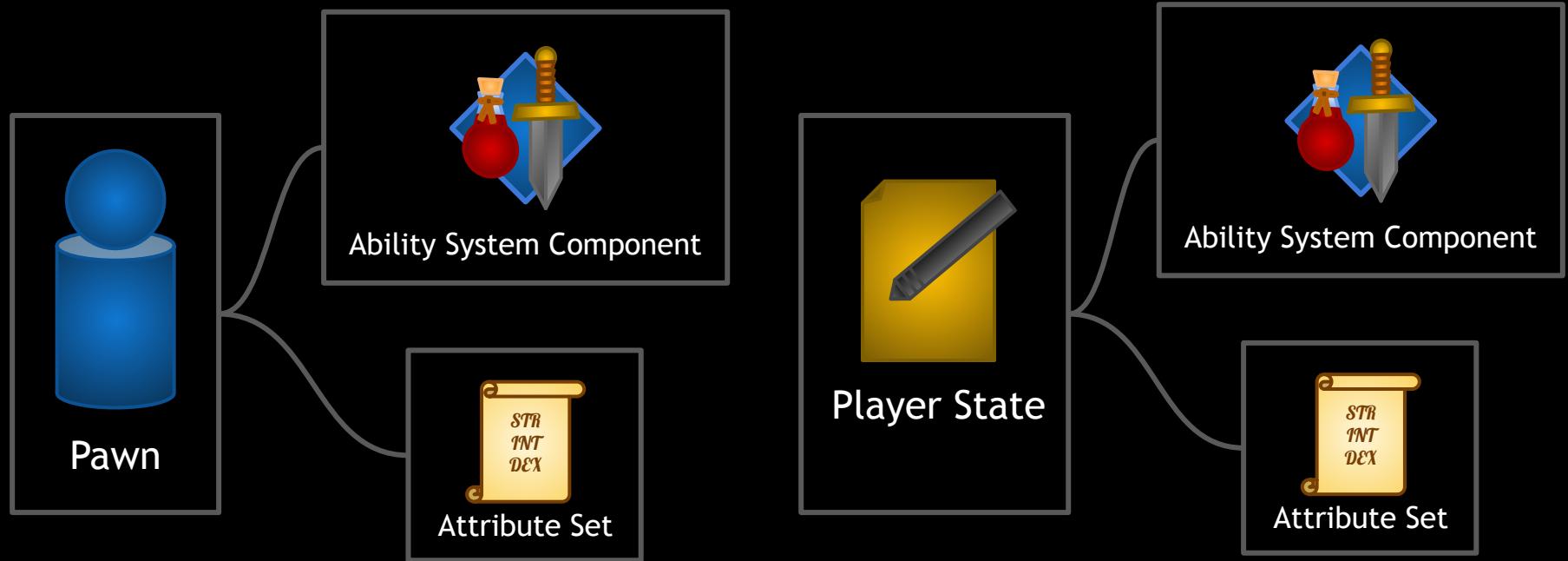


Gameplay Cue

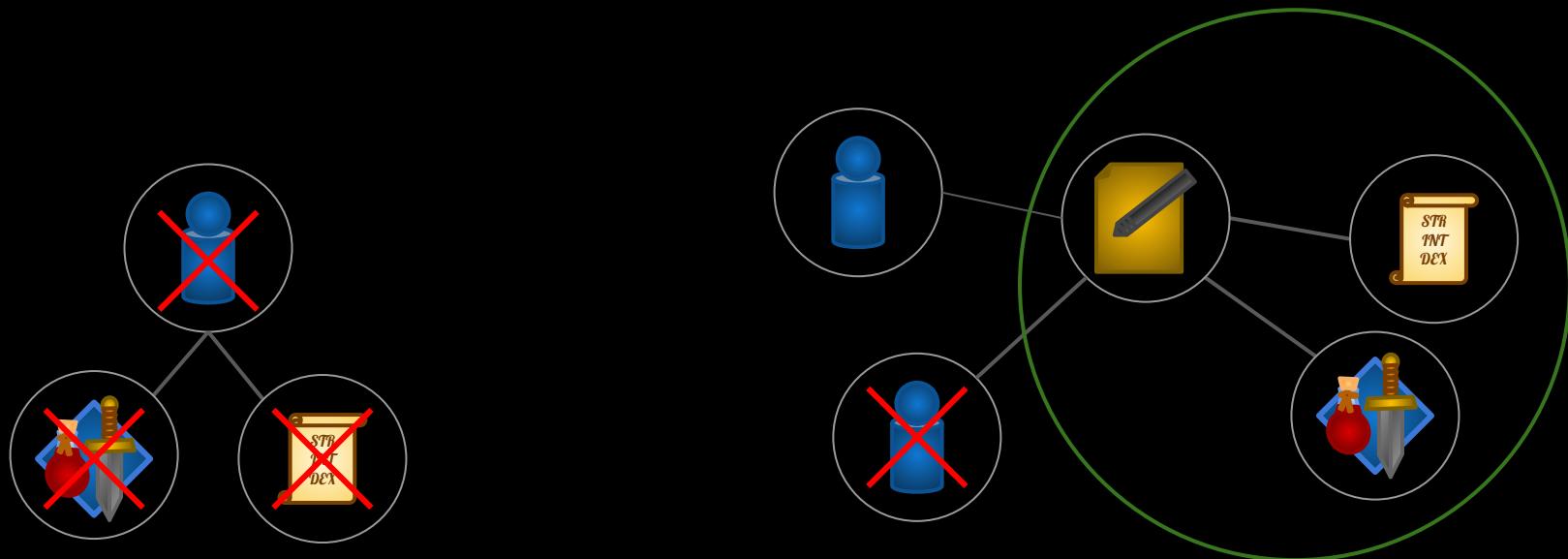


Gameplay Tag

# Gameplay Ability System

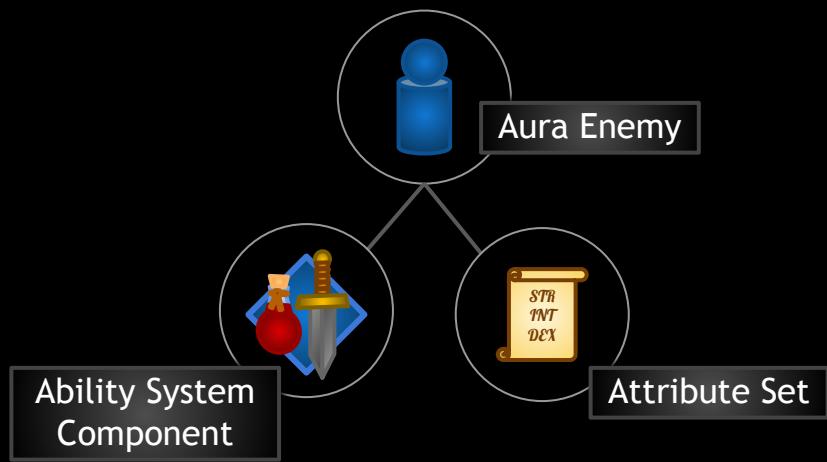


# Gameplay Ability System

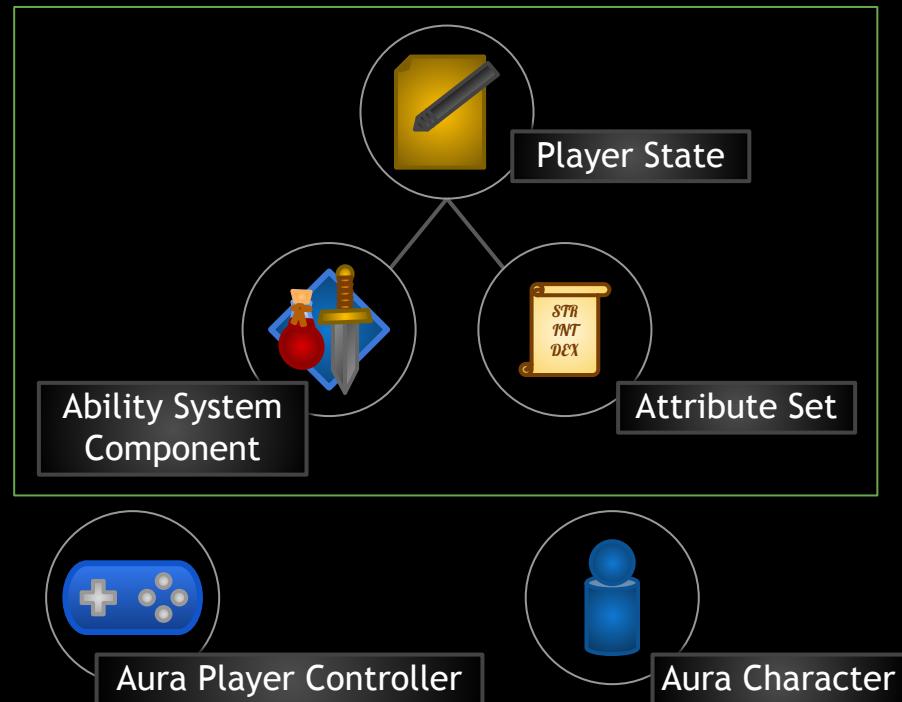


# Gameplay Ability System

Enemy Character



Player Controlled Character

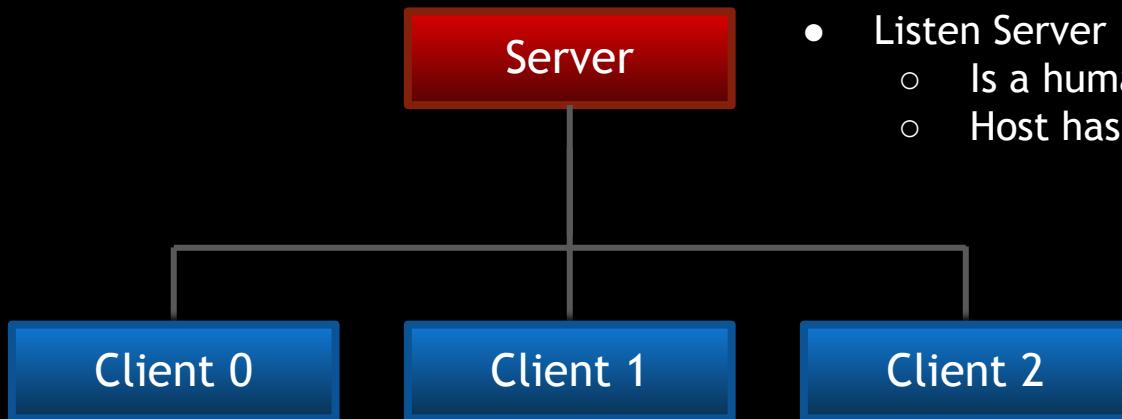


# Gameplay Ability System

## First Steps

1. Player State class
2. Ability System Component
3. Attribute Set

# GAS in Multiplayer

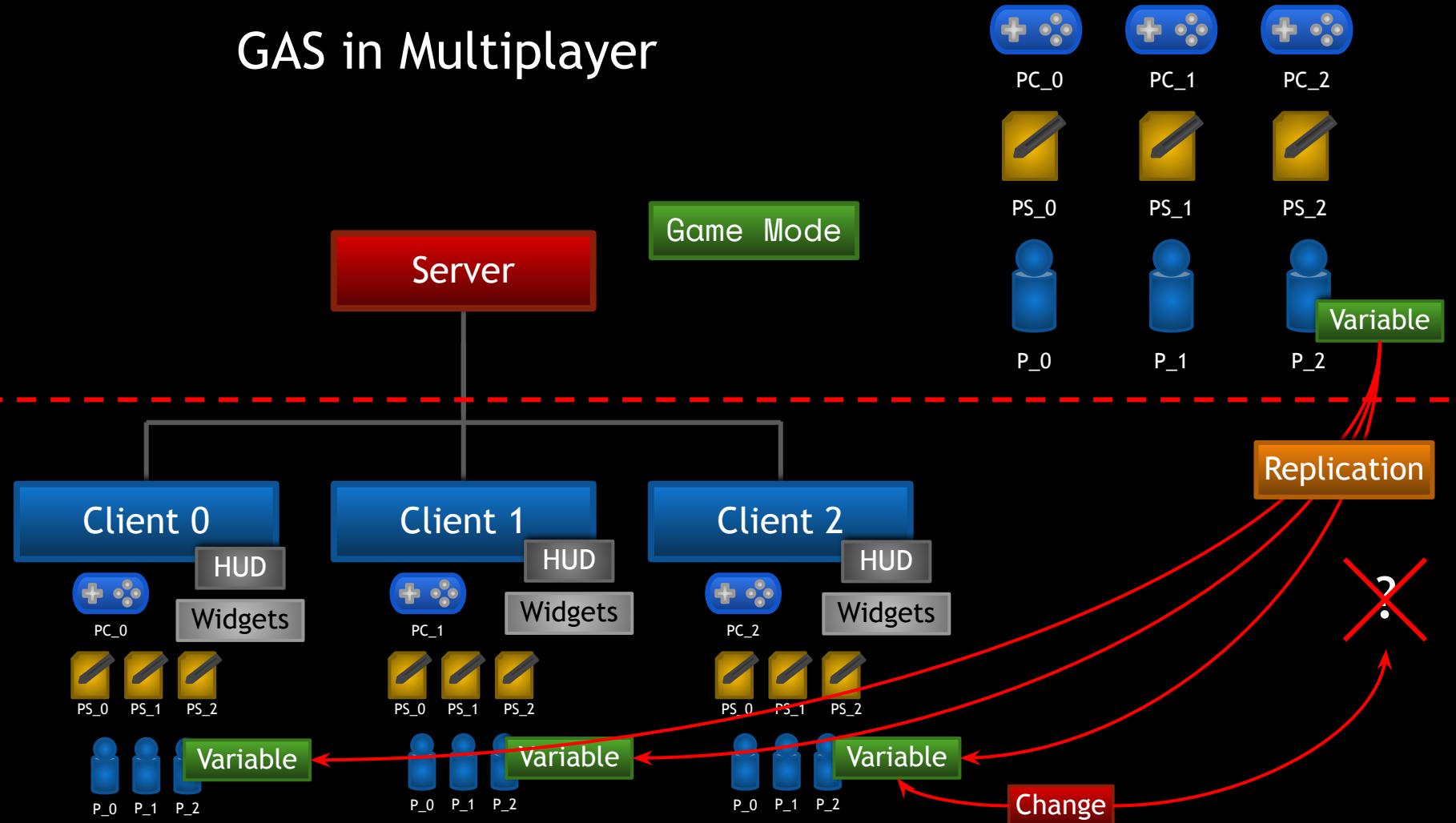


- Dedicated Server
  - No human player
  - No rendering to a screen
- Listen Server
  - Is a human player
  - Host has the advantage - no lag!

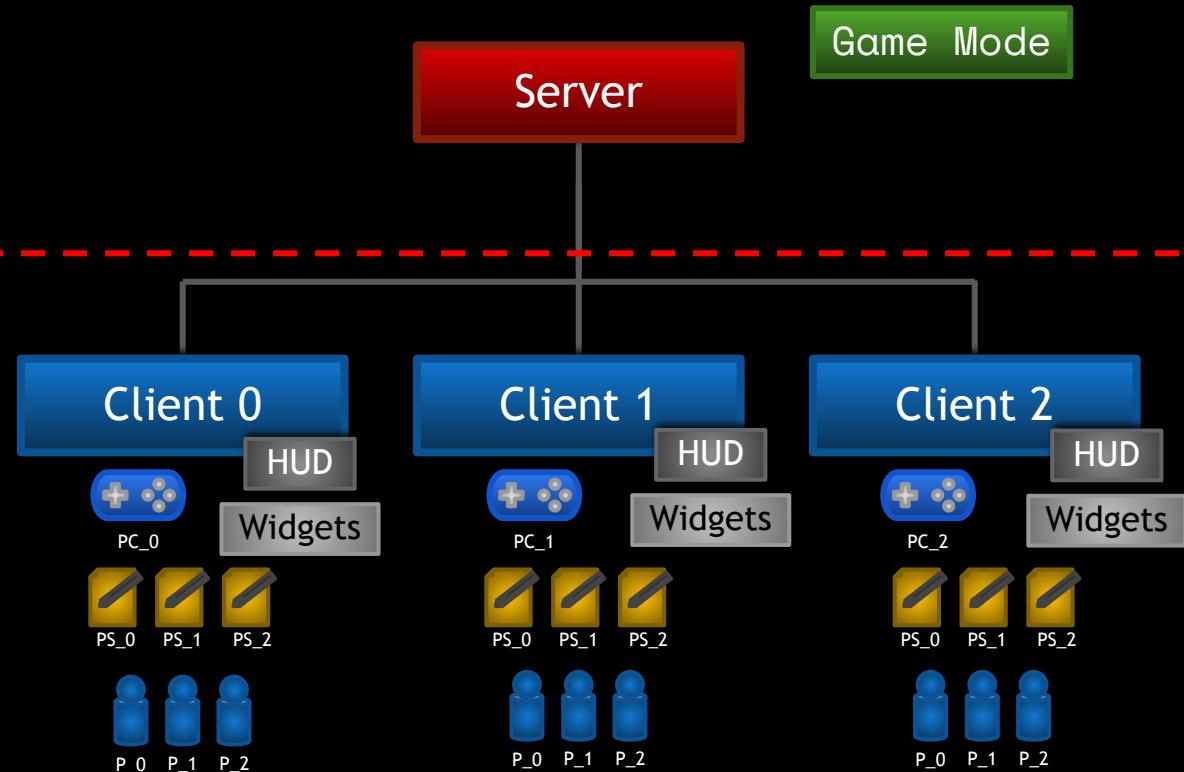
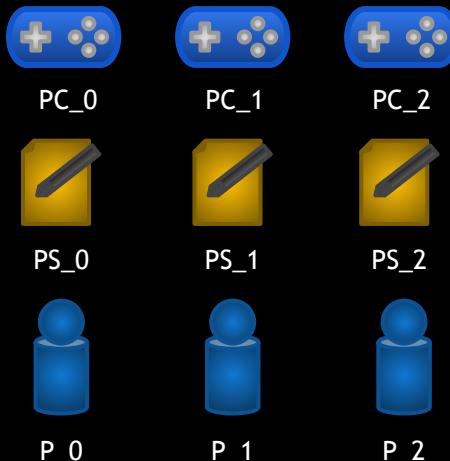
Server is the Authority

- “Correct” version of the game
- We do “important” things on the server
- We’ll learn what those are

# GAS in Multiplayer



# GAS in Multiplayer



# Init Ability Actor Info

# Replication Mode

```
UAbilitySystemComponent::SetReplicationMode(EGameplayEffectReplicationMode::Mixed);
```

Replication Mode	Use Case	Description
Full	Single Player	Gameplay Effects are replicated to all clients
Mixed	Multiplayer, Player-Controlled	Gameplay Effects are replicated to the owning client only. Gameplay Cues and Gameplay Tags replicated to all clients.
Minimal	Multiplayer, AI-Controlled	Gameplay Effects are <b>not</b> replicated. Gameplay Cues and Gameplay Tags replicated to all clients.

# Init Ability Actor Info

```
UAbilitySystemComponent::InitAbilityActorInfo(AActor* InOwnerActor, AActor* InAvatarActor);
```

Ability System Component



AActor



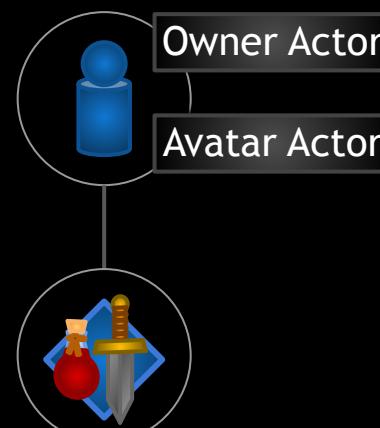
Owner Actor

AActor

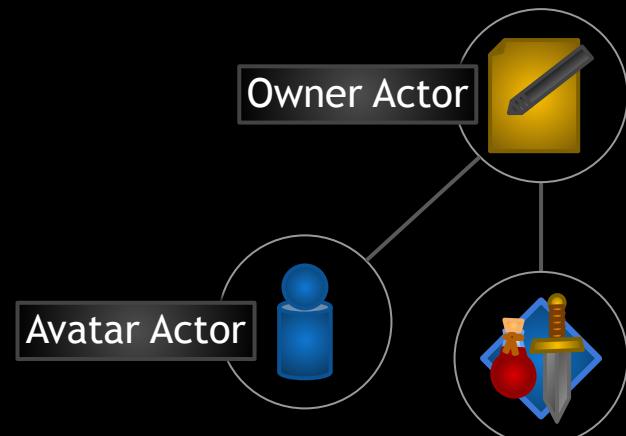


Avatar Actor

Enemy Character



Player Controlled Character



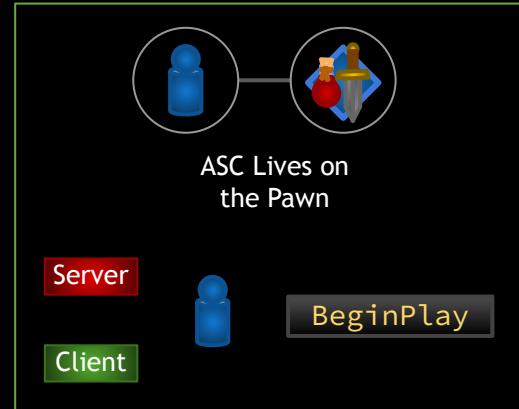
# Init Ability Actor Info

- Must be done after possession (the Controller has been set for the Pawn)

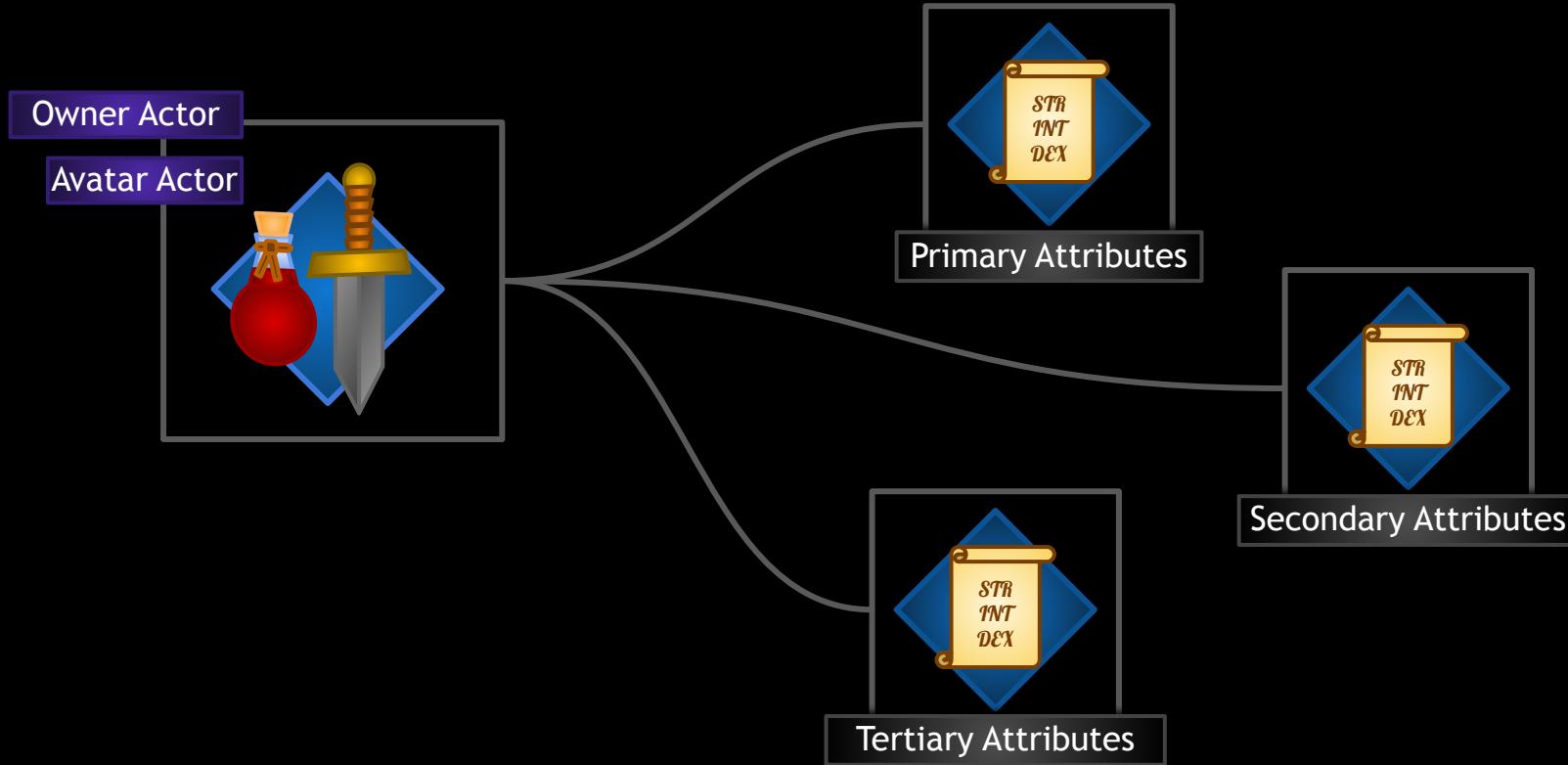
Player-Controlled Character



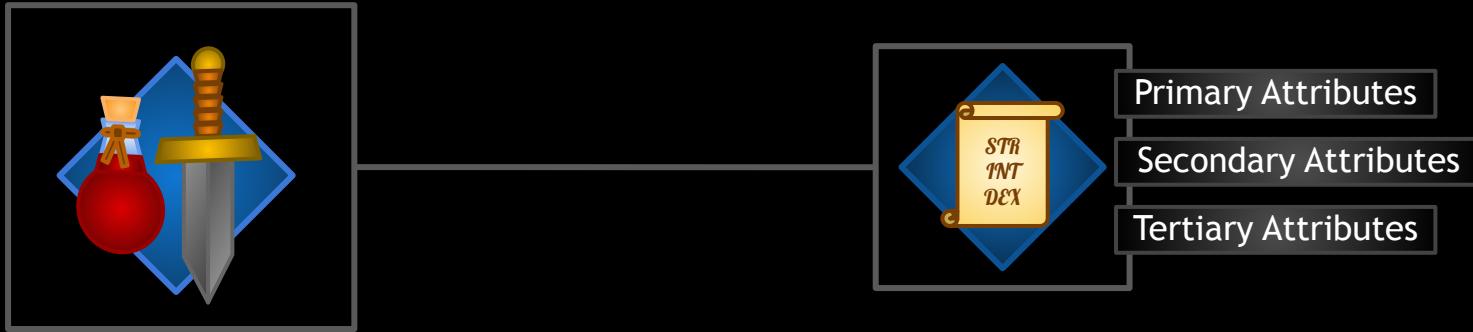
AI-Controlled Character



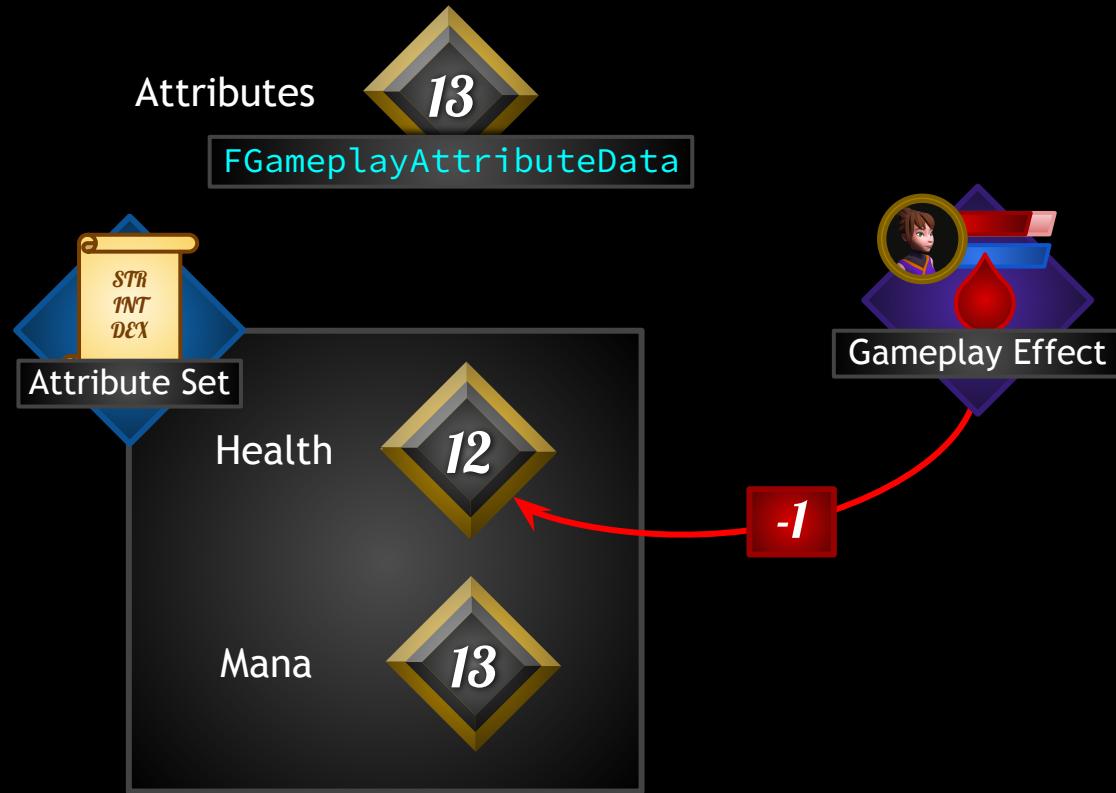
# Attributes



# Attributes



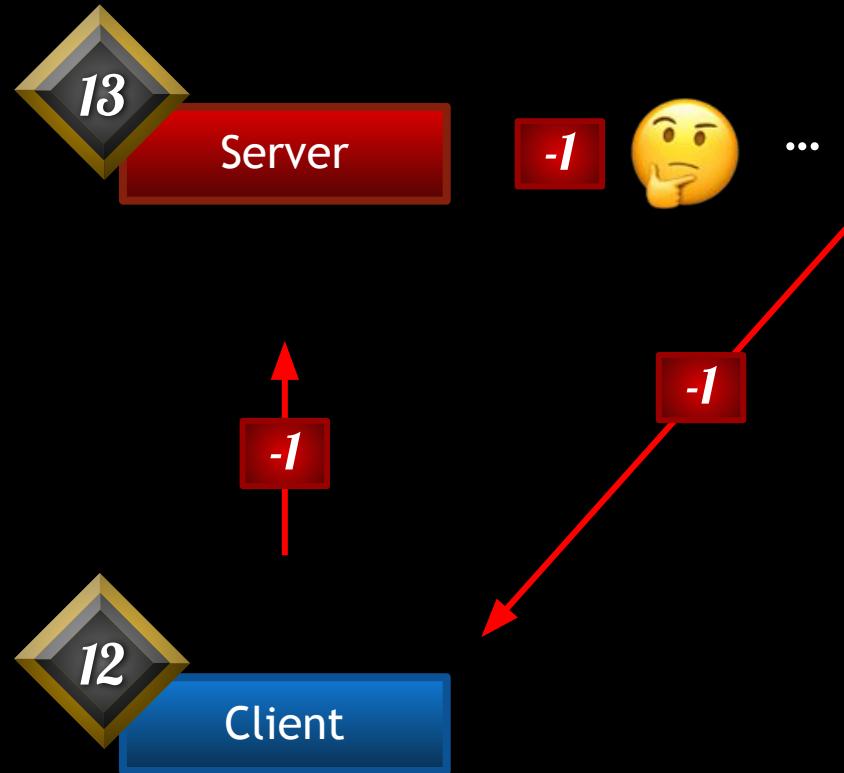
# Attributes



**Prediction:** The client doesn't need to wait for the server's permission to change a value. The value can change immediately client-side and the server is informed of the change. The server can roll back changes that are invalid.

# Attributes

Without Prediction:

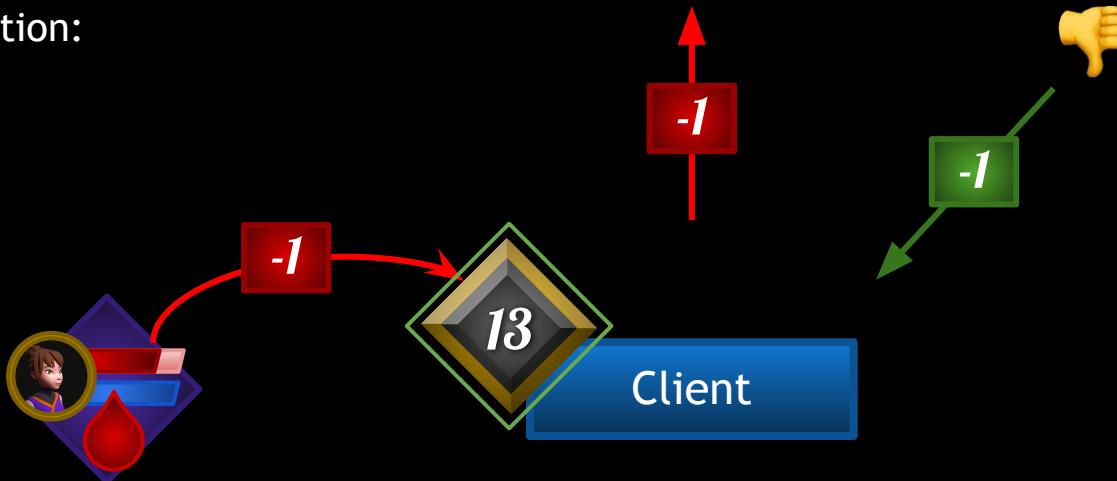


# Attributes

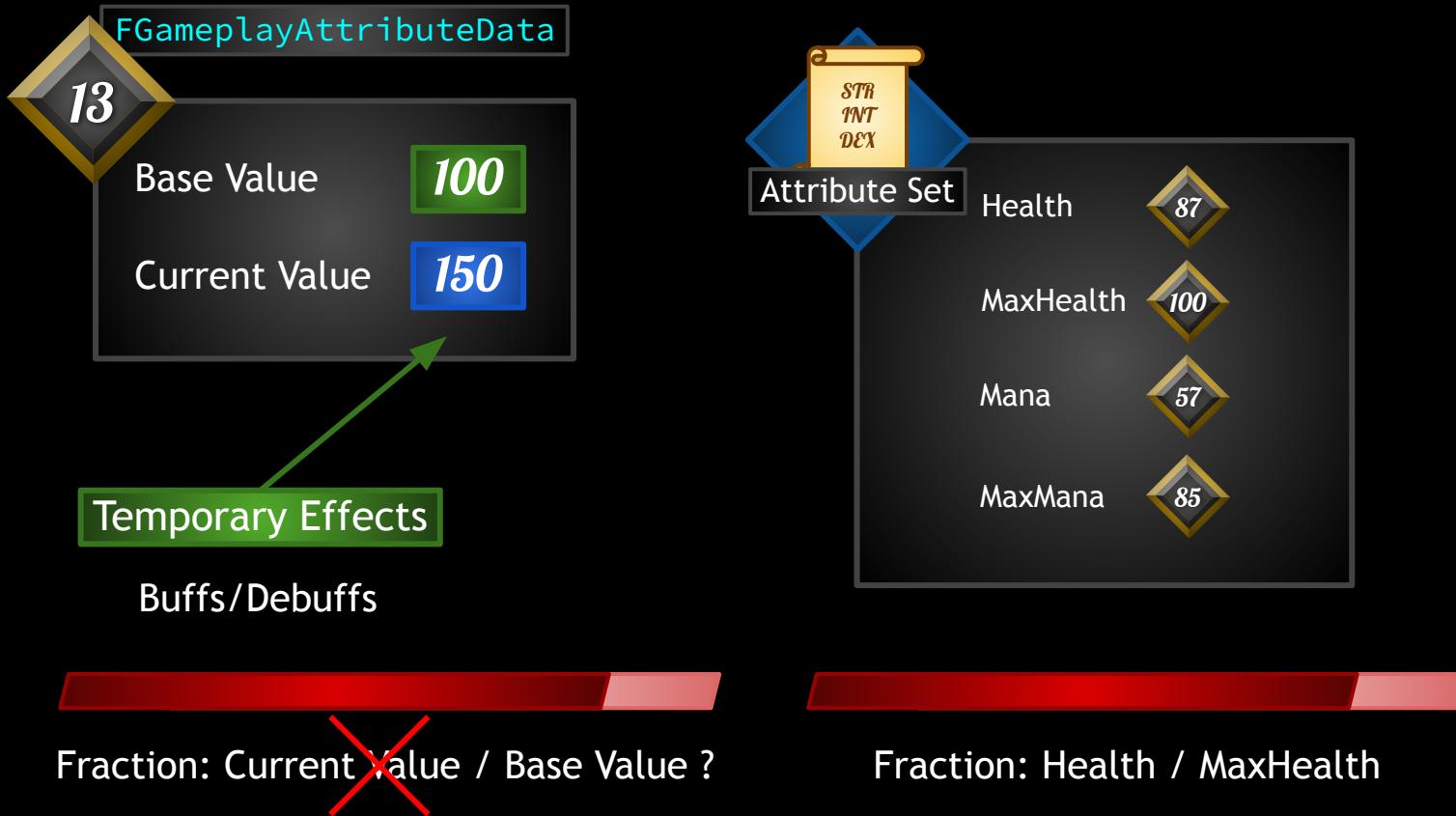


*or...*

With Prediction:



# Attributes



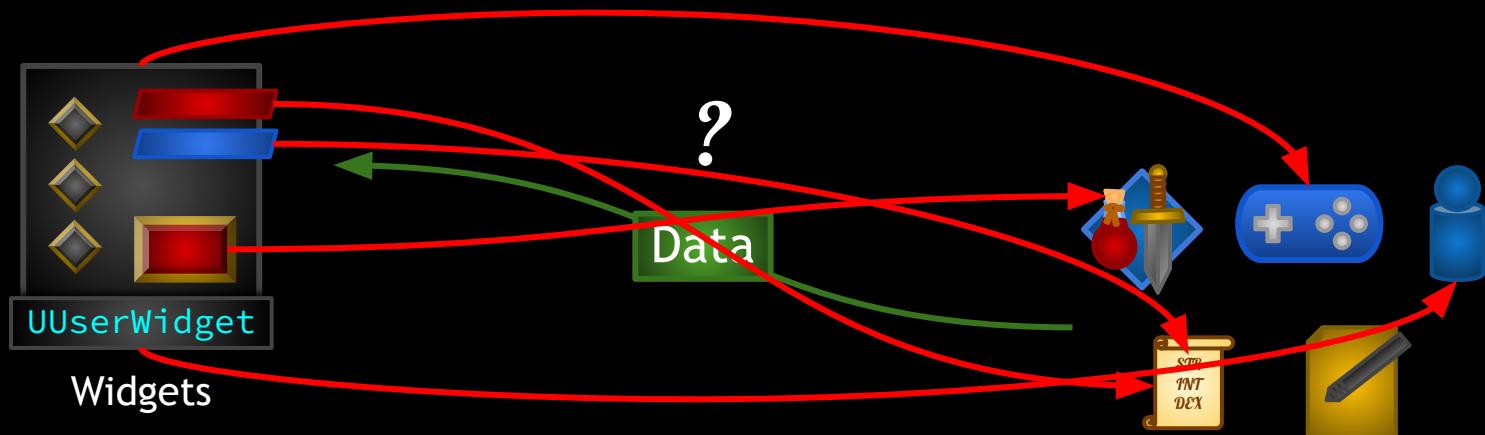
# Mixed Replication Mode

**For Mixed Replication Mode:** The `OwnerActor`'s Owner must be the `Controller`. For Pawns, this is set automatically in `PossessedBy()`.

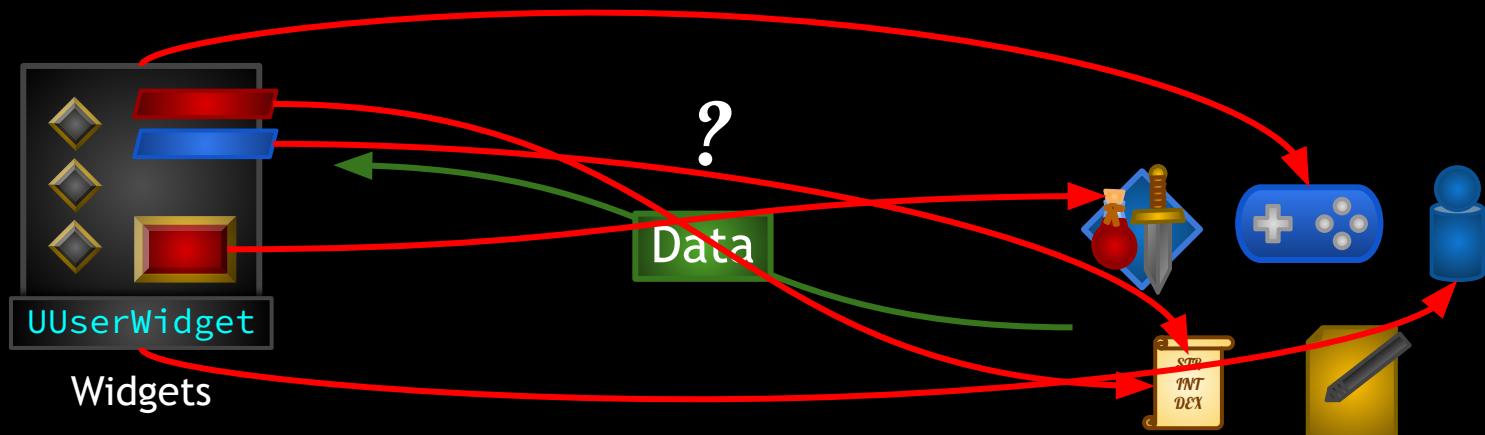
The `PlayerState`'s Owner is automatically set to the `Controller`.

Therefore, if your `OwnerActor` is not the `PlayerState`, and you use Mixed Replication Mode, you must call `SetOwner()` on the `OwnerActor` to set its owner to the `Controller`.

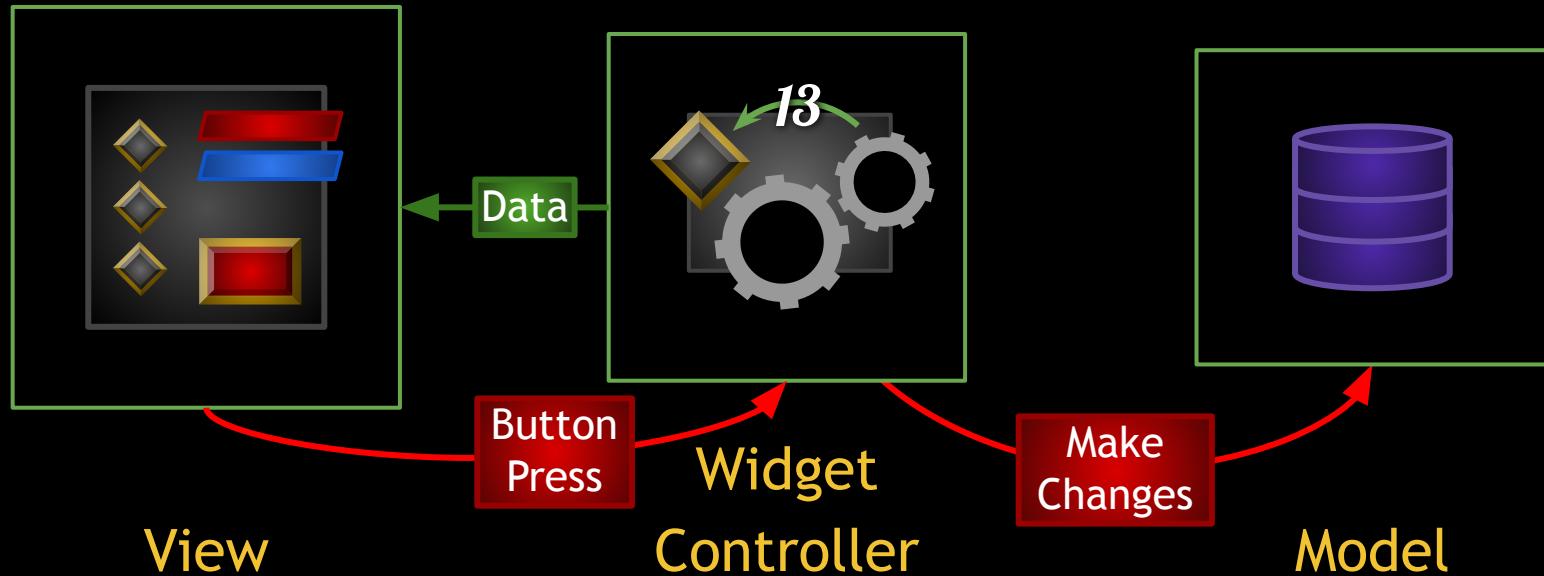
# RPG Game UI



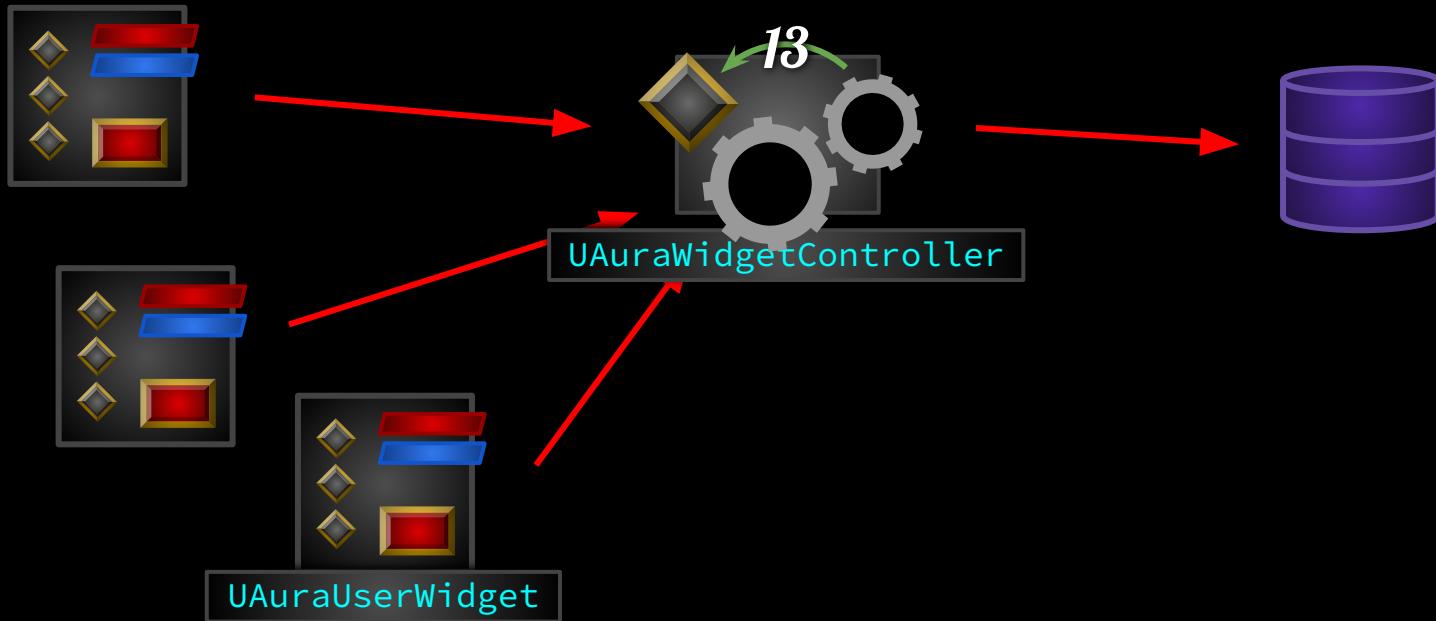
# RPG Game UI



# RPG Game UI



# RPG Game UI

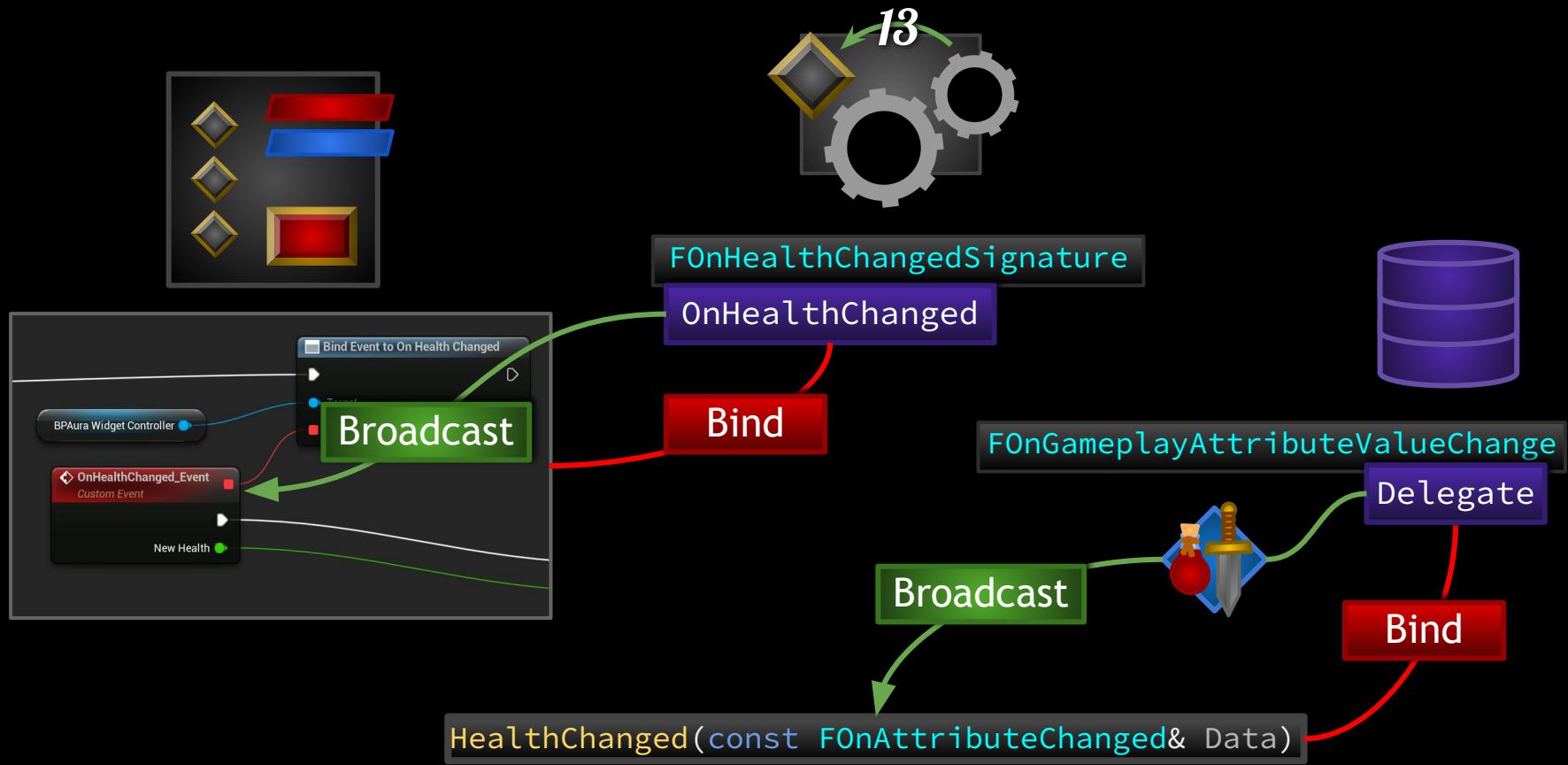


# RPG Game UI

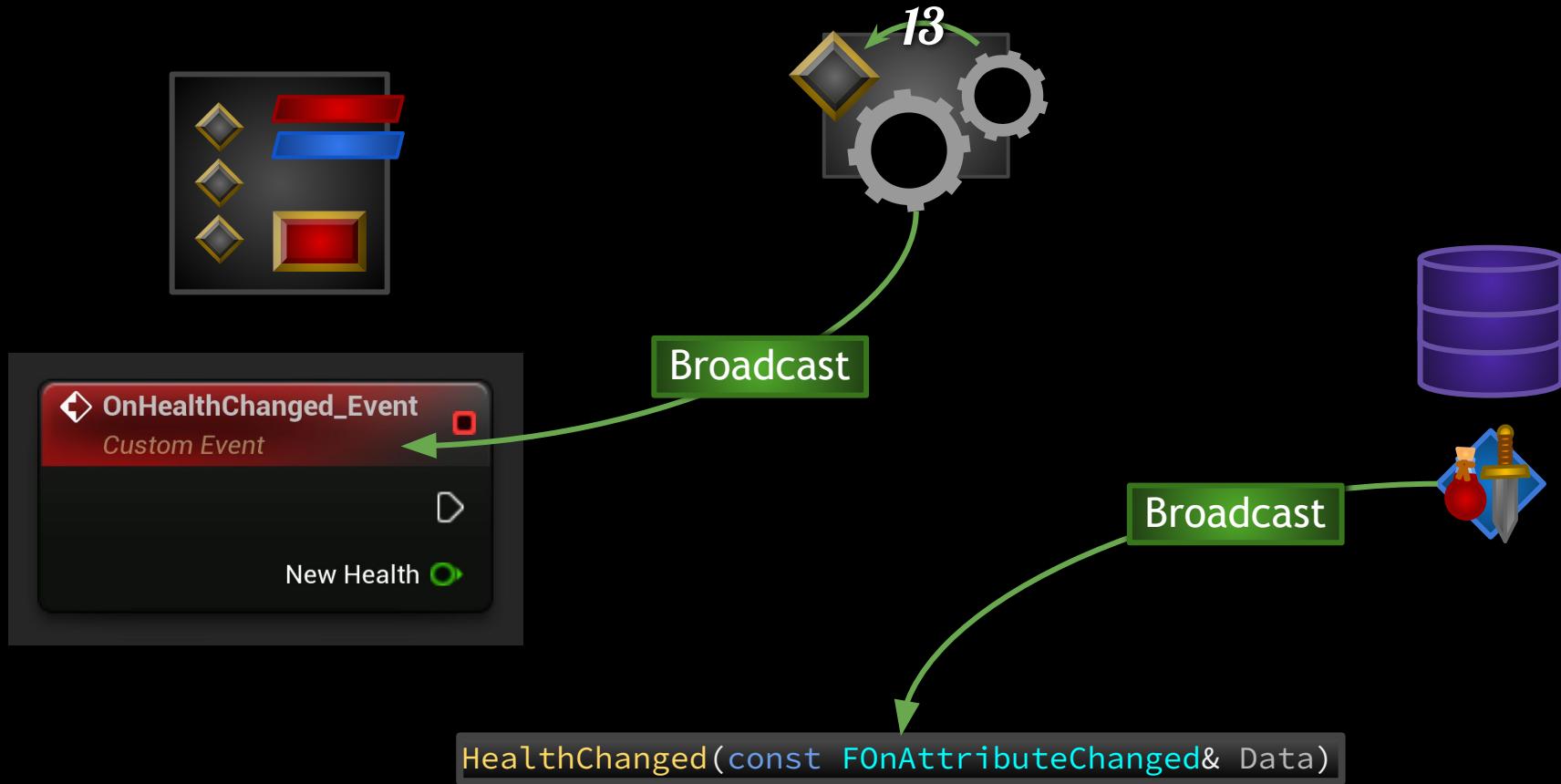
## Next Steps

1. UAuraUserWidget
2. UAuraWidgetController
3. AAuraHUD
4. Display Health and Mana

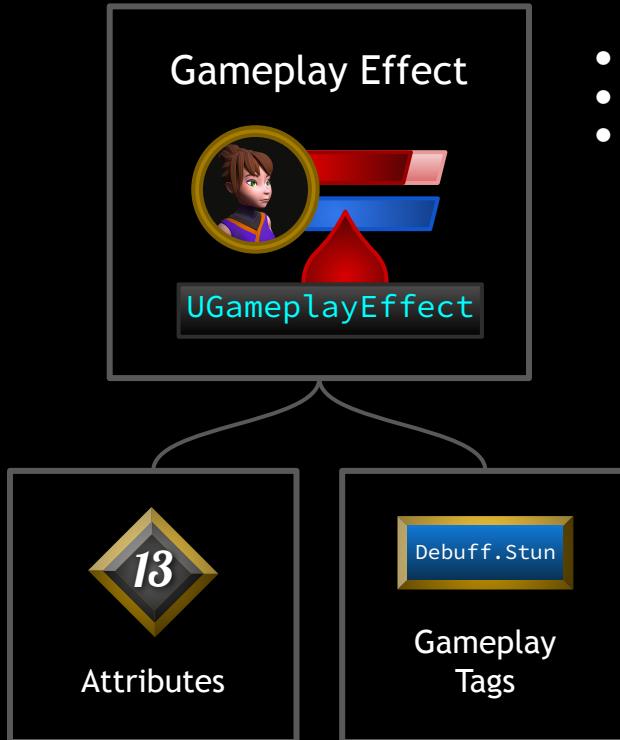
# RPG Game UI



# RPG Game UI



# Gameplay Effects



- Data only
- Don't subclass `UGameplayEffect`
- Change Attributes through:
  - Modifiers
  - Executions

## Modifier Op

- Add
- Multiply
- Divide
- Override

## Magnitude Calculation Type

- Scalable Float
- Attribute Based
- Custom Calculation Class (MMC)
- Set by Caller

## Executions

- Gameplay Effect Execution Calculation

# Gameplay Effects

## Gameplay Effect



## Duration Policy

- Instant
- Has Duration
- Infinite

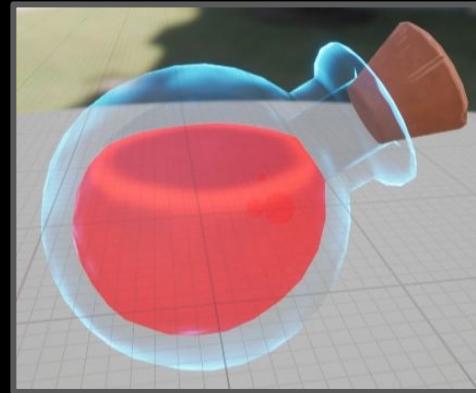


## Gameplay Effect Spec

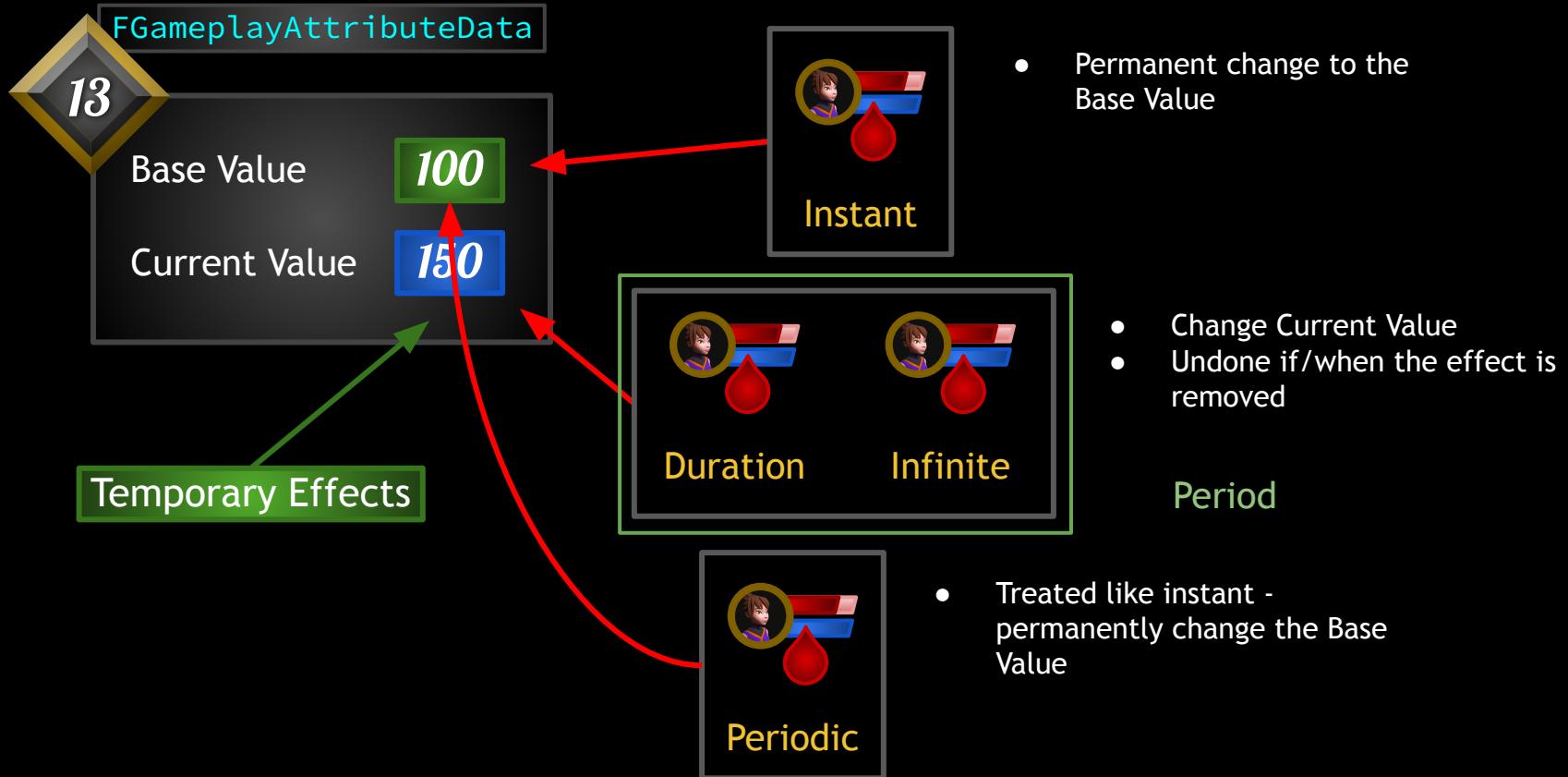
## Stacking

## Add Gameplay Tags

## Grant Abilities



# Periodic Gameplay Effects



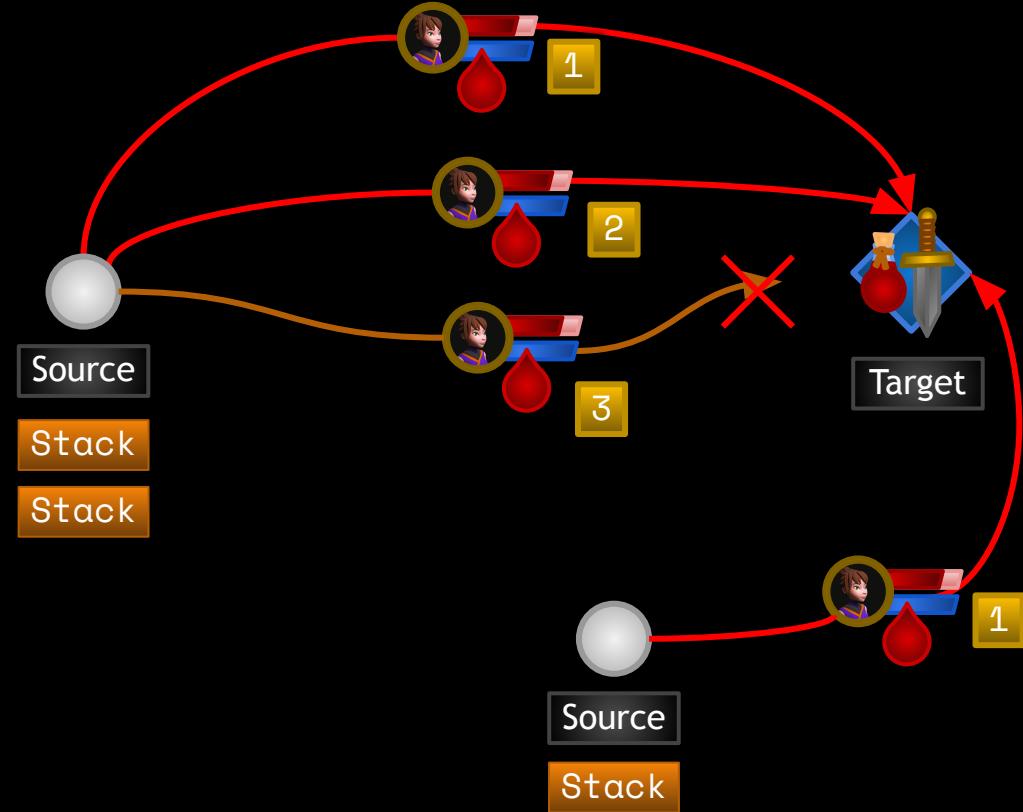
# Stacking

## Stacking Type

- Aggregate by Source

Stack Limit Count

2



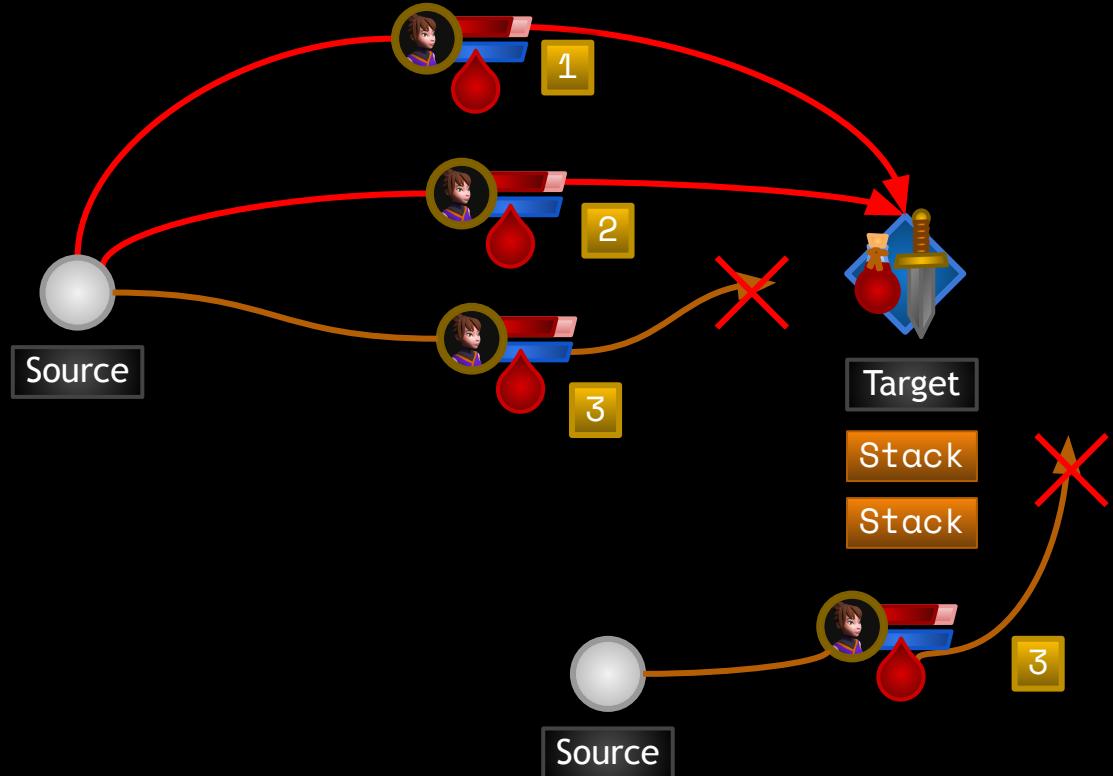
# Stacking

## Stacking Type

- Aggregate by Target

Stack Limit Count

2



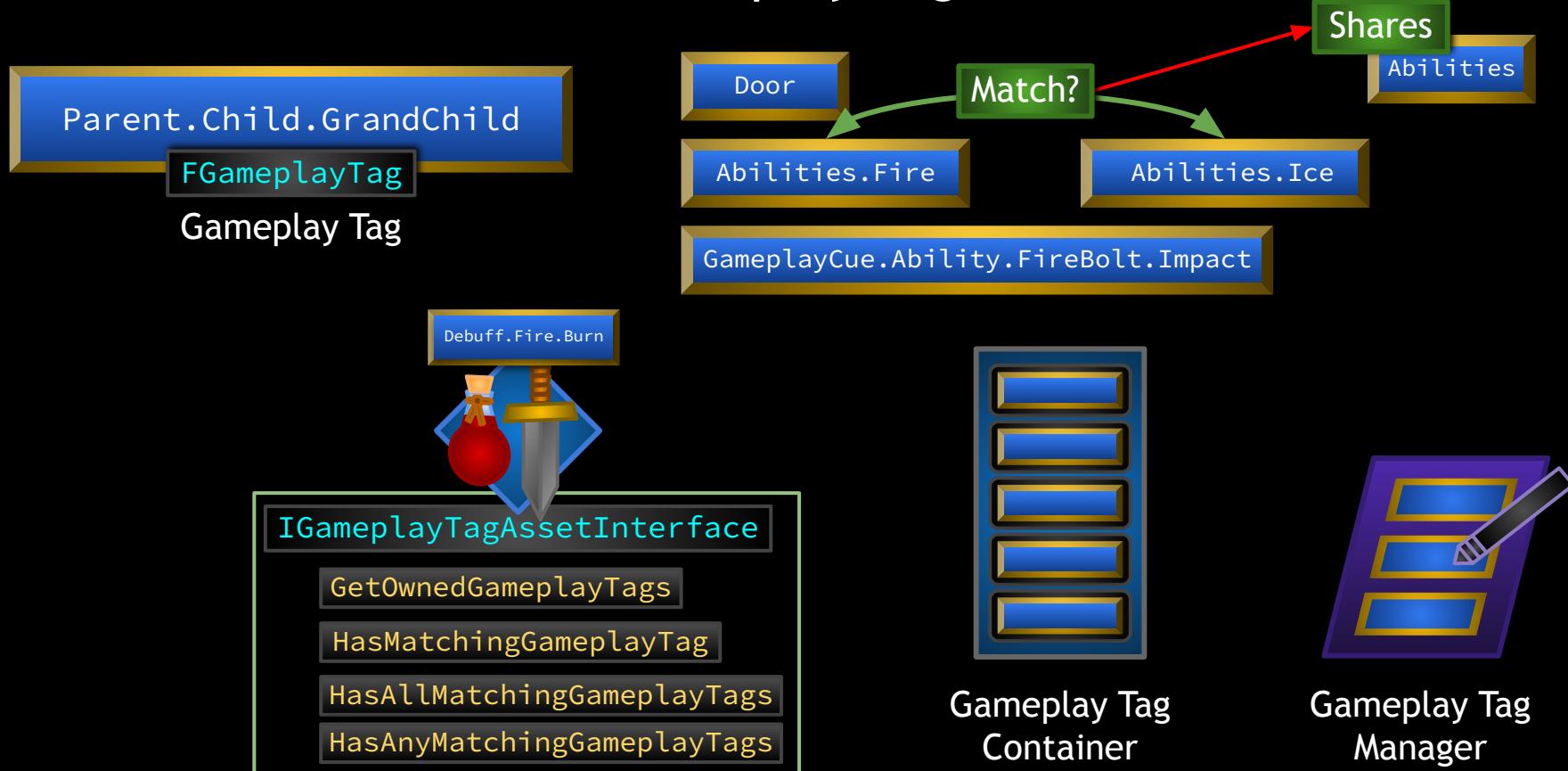
# PreAttributeChange

## PreAttributeChange

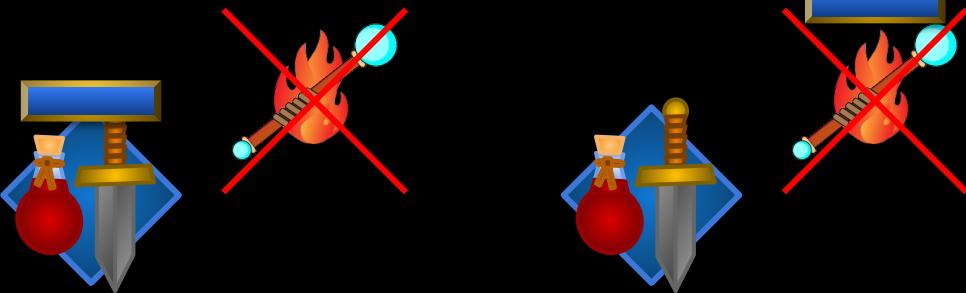
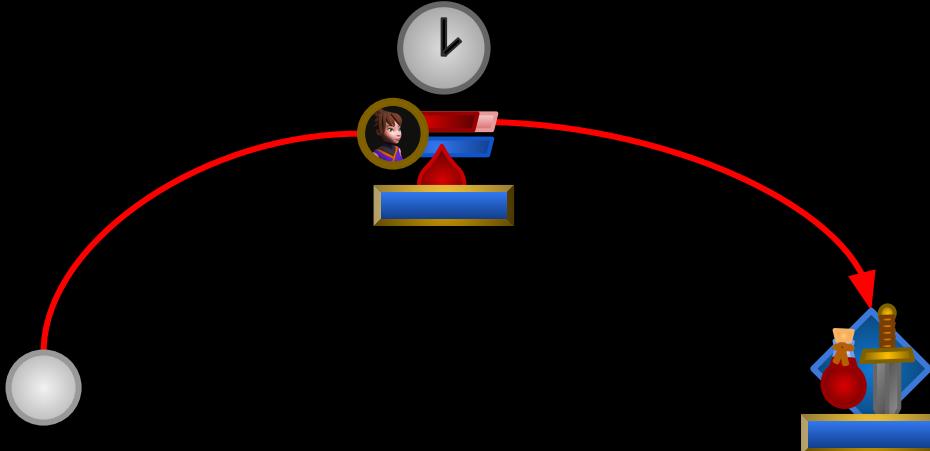
- Changes to CurrentValue
  - before the change happens
- Triggered by changes to Attributes
  - Attribute Accessors
  - Gameplay Effects
- Does not permanently change the modifier, just the value returned from querying the modifier
- Later operations recalculate the Current Value from all modifiers
  - We need to clamp again

## PostGameplayEffectExecute

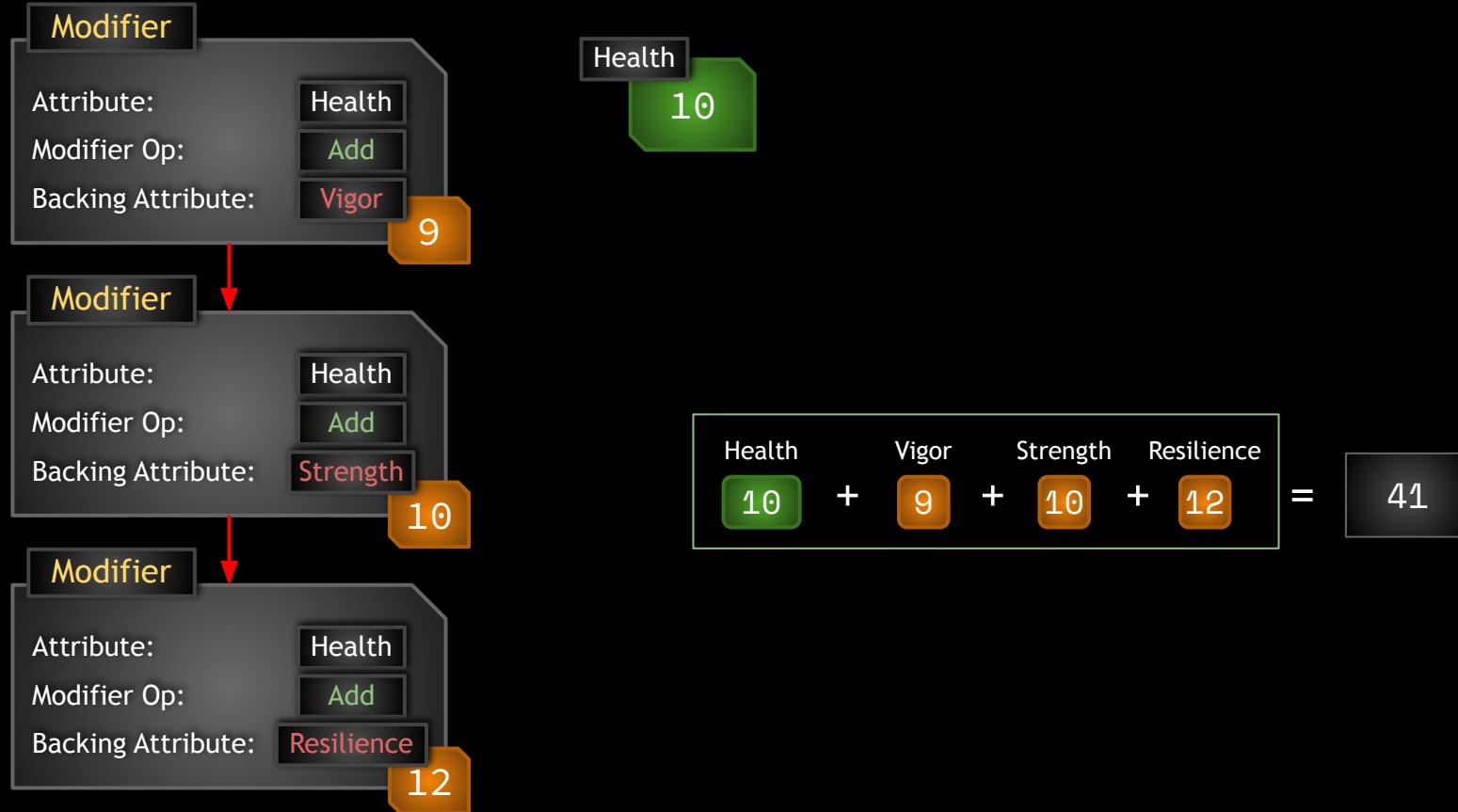
# Gameplay Tags



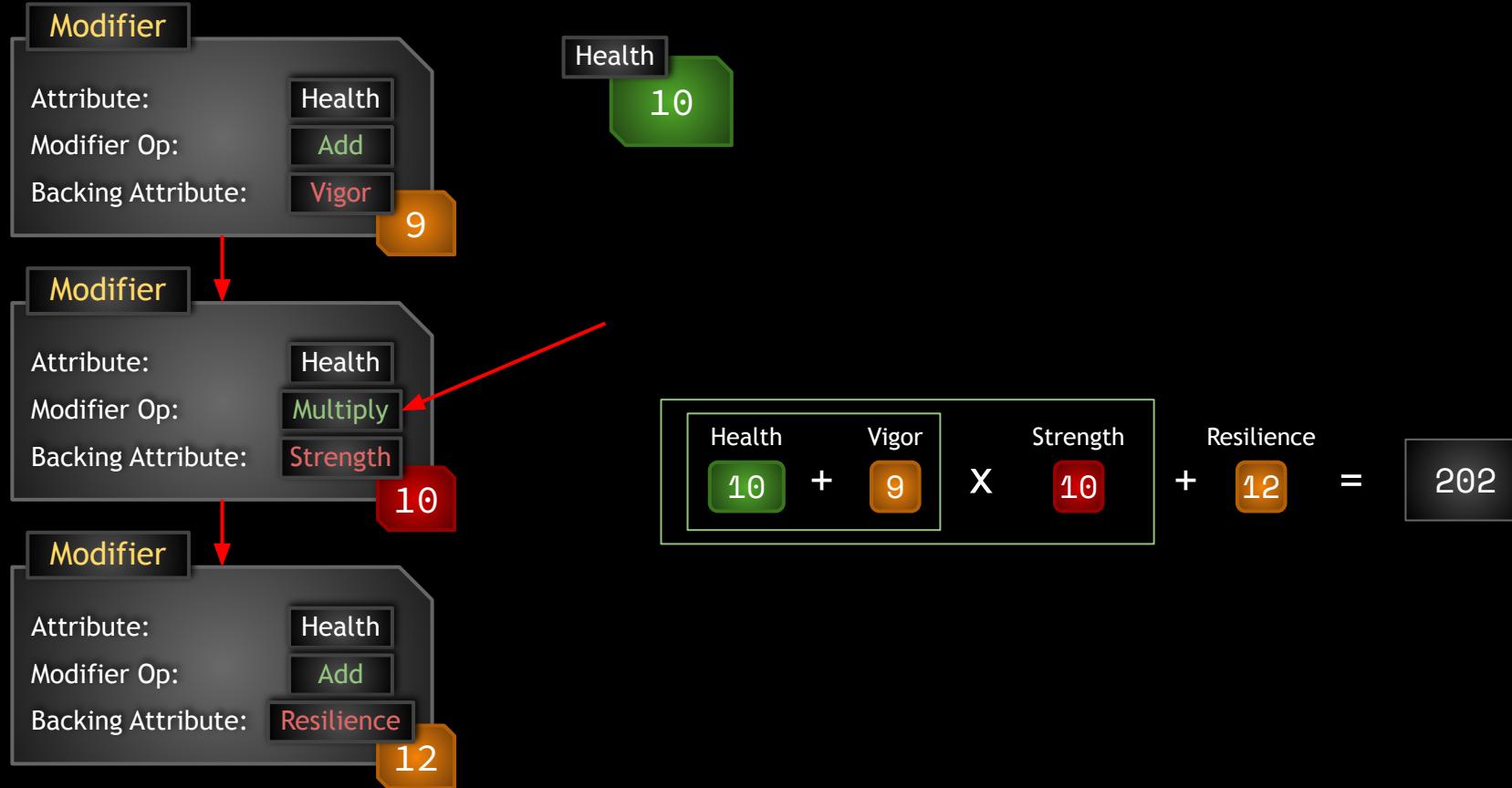
- Inputs
- Abilities
- Attributes
- Damage Types
- Buffs/Debuffs
- Messages
- Data
- Anything you want!



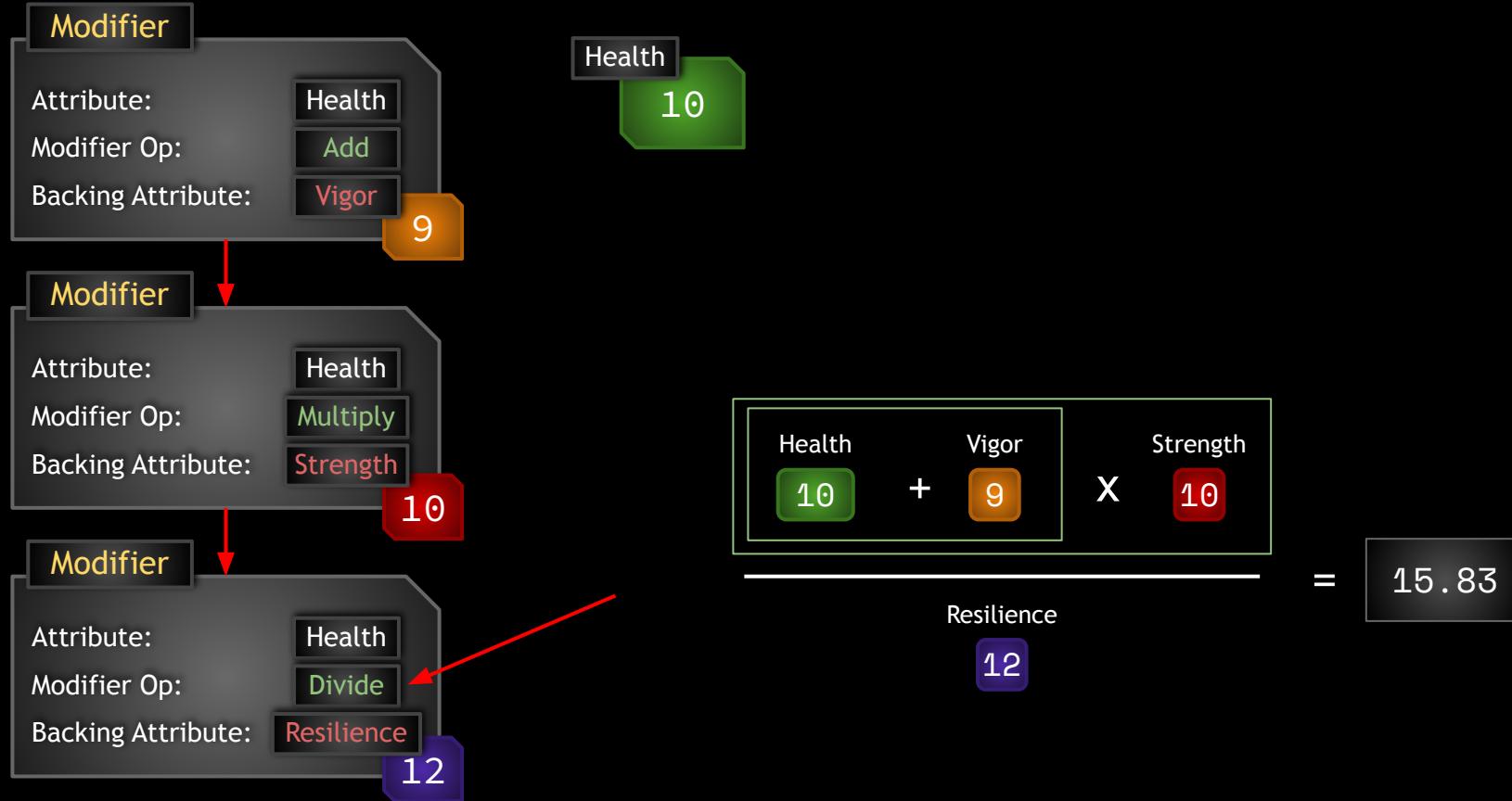
# Attribute Based Modifiers



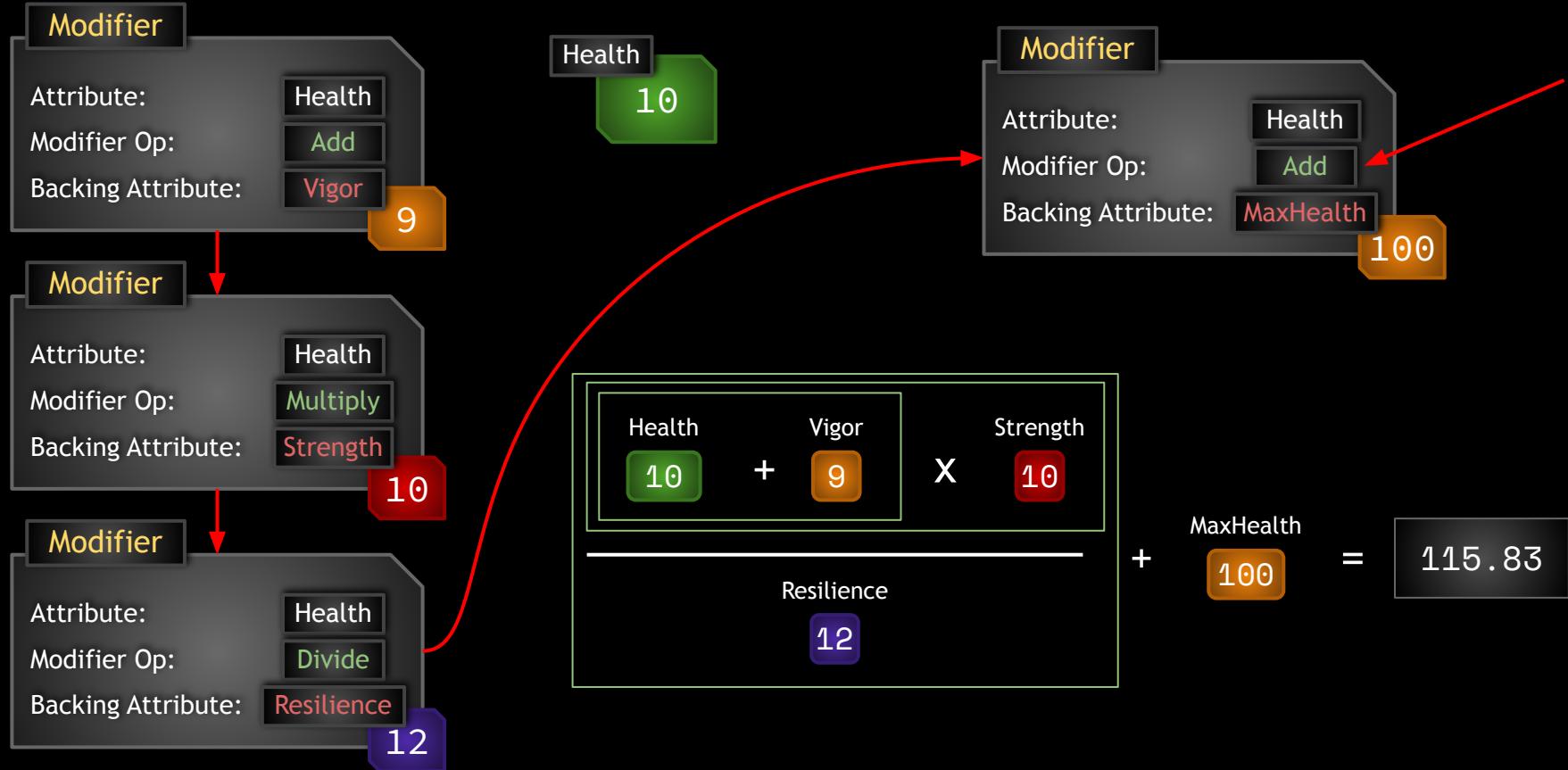
# Attribute Based Modifiers



# Attribute Based Modifiers



# Attribute Based Modifiers



# Attribute Based Modifiers

Coefficient

C

Pre Multiply Additive Value

Pre

Post Multiply Additive Value

Post



# Attribute Based Modifiers

Coefficient

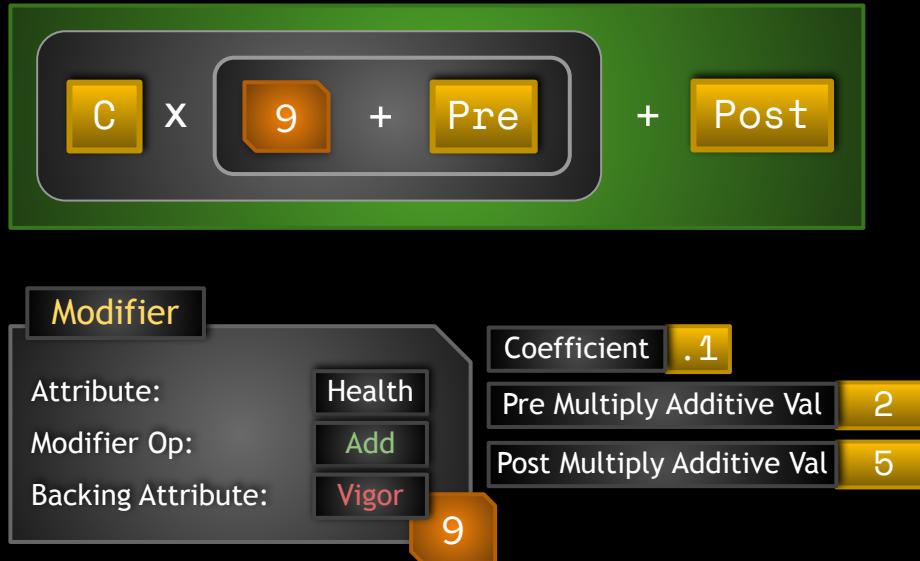
C

Pre Multiply Additive Value

Pre

Post Multiply Additive Value

Post

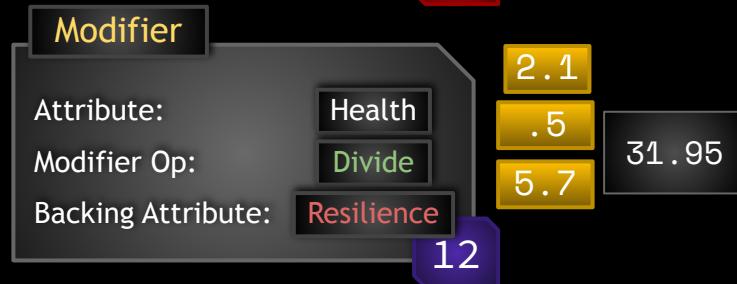
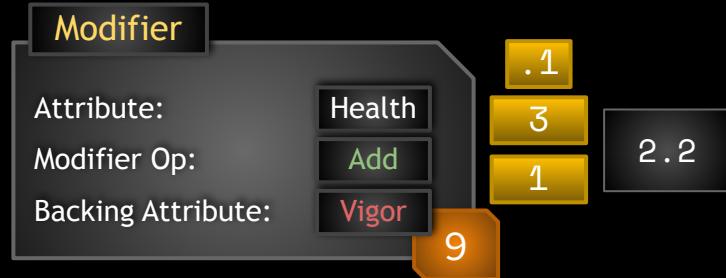


$$.1 \times 9 + 2 + 5 = 6.1$$

# Attribute Based Modifiers

Modifier	Coefficient	.1		
Attribute:	Health	Pre Multiply Additive Val	3	
Modifier Op:	Add	Post Multiply Additive Val	1	
Backing Attribute:	Vigor			
	9	$.1 \times 9 + 3 + 1 = 2.2$		2.2
Modifier	Coefficient	.5		
Attribute:	Health	Pre Multiply Additive Val	6	
Modifier Op:	Multiply	Post Multiply Additive Val	11	
Backing Attribute:	Strength			
	10	$.5 \times 10 + 6 + 11 = 19$		19
Modifier	Coefficient	2.1		
Attribute:	Health	Pre Multiply Additive Val	.5	
Modifier Op:	Divide	Post Multiply Additive Val	5.7	
Backing Attribute:	Resilience			
	12	$2.1 \times 12 + .5 + 5.7 = 31.95$		31.95

# Attribute Based Modifiers



$$\frac{10 + 2.2 \times 19}{31.95} = 7.26$$

Health + Vigor X Strength

---

Resilience

# Derived Attributes

## Primary Attributes

**Str** Increases physical damage

**Int** Increases magical damage

**Res** Increases Armor and Armor Penetration

**Vig** Increases Health

## Secondary Attributes

Resilience **Armor** Reduces damage taken, improves Block Chance

Resilience **Armor Penetration** Ignores percentage of enemy Armor, increases Crit Hit Chance

Armor **Block Chance** Chance to cut incoming damage in half

Armor Penetration **Critical Hit Chance** Chance to double damage plus critical hit bonus

Armor Penetration **Critical Hit Damage** Bonus damage added when a critical hit is scored

Armor **Critical Hit Resistance** Reduces critical hit chance of attacking enemies

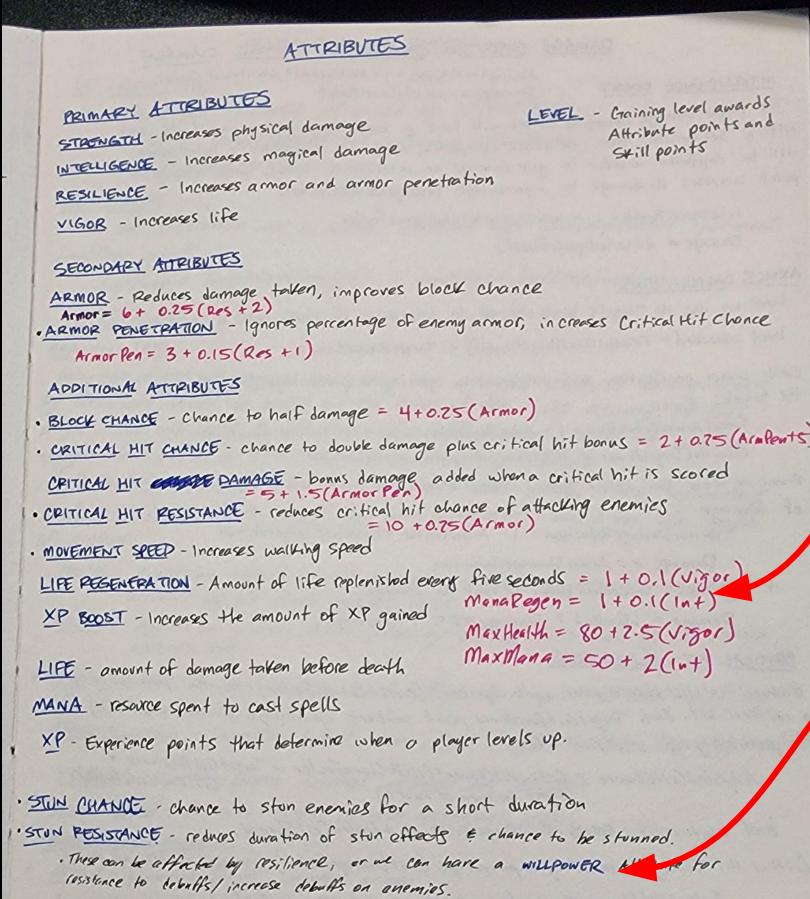
Vigor **Health Regeneration** Amount of Health regenerated every 1 second

Intelligence **Mana Regeneration** Amount of Mana regenerated every 1 second

Vigor **Max Health** Maximum amount of Health obtainable

Intelligence **Max Mana** Maximum amount of Mana obtainable

# Derived Attributes



- Brainstorming
  - Messy
  - Estimations
- 
- Speculation/ideas

# Derived Attributes

## Primary Attributes

**Str** Increases physical damage

**Int** Increases magical damage

**Res** Increases Armor and Armor Penetration

**Vig** Increases Health

## Secondary Attributes

Resilience

Armor

Resilience

Armor Penetration

Armor

Block Chance

Armor Penetration

Critical Hit Chance

Armor Penetration

Critical Hit Damage

Armor

Critical Hit Resistance

Vigor

Health Regeneration

Intelligence

Mana Regeneration

Vigor

Level

Max Health

Intelligence

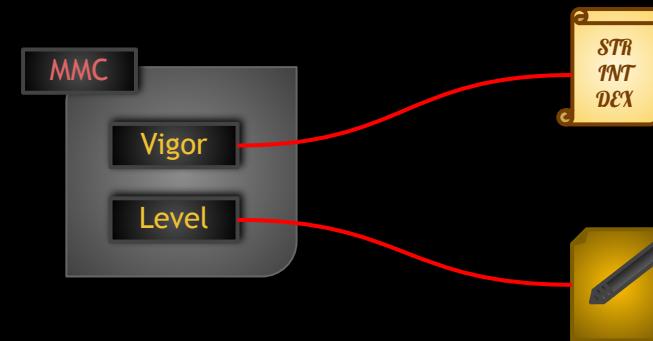
Level

Max Mana

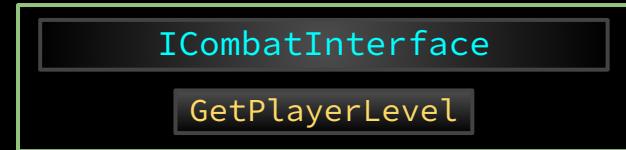
# Custom Calculation



Modifier Magnitude Calculation (MMC)



$$\text{MaxHealth} = 80 + 2.5 \times \text{Vigor} + 10 \times \text{Level}$$



# Custom Calculation

## Next Steps

1. Add a **Level** variable to the Player State
2. Create a **Combat Interface**
3. Create MMCs for MaxHealth and MaxMana
  - a. Make these dependent on **Level** as well as their backing Attributes

# Attribute Menu

WBP\_AttributeMenu

**ATTRIBUTES**

PRIMARY ATTRIBUTES

Attribute Points 0

Strength 6 +

Intelligence 13 +

Resilience 8 +

Vigor 7 +

SECONDARY ATTRIBUTES

Armor Penetration 10.35

Armor 17

Block Chance 0

Critical Hit Chance 23.52

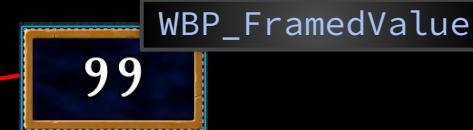
Critical Hit Damage 14.44

Health 89 / 89

Mana 164 / 164

X

This screenshot shows the Attribute Menu interface. It includes sections for Primary Attributes (Strength, Intelligence, Resilience, Vigor) and Secondary Attributes (Armor Penetration, Armor, Block Chance, Critical Hit Chance, Critical Hit Damage). Each attribute has a current value, a plus sign button for increasing it, and a minus sign button for decreasing it. Below these are Health and Mana bars, each showing a current value and a maximum value separated by a slash. A red 'X' button is located at the bottom right.



Framed Value



Text Value Row with Button



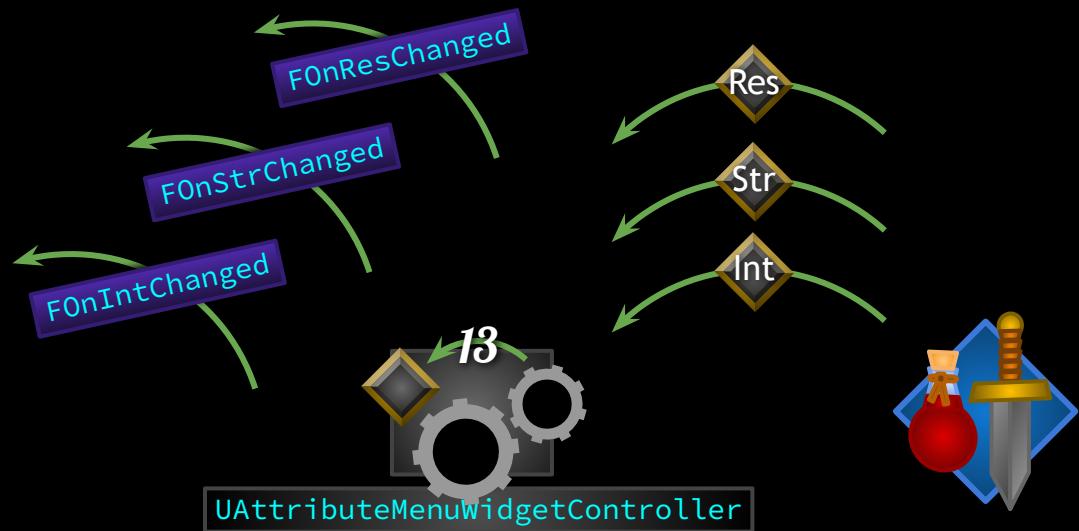
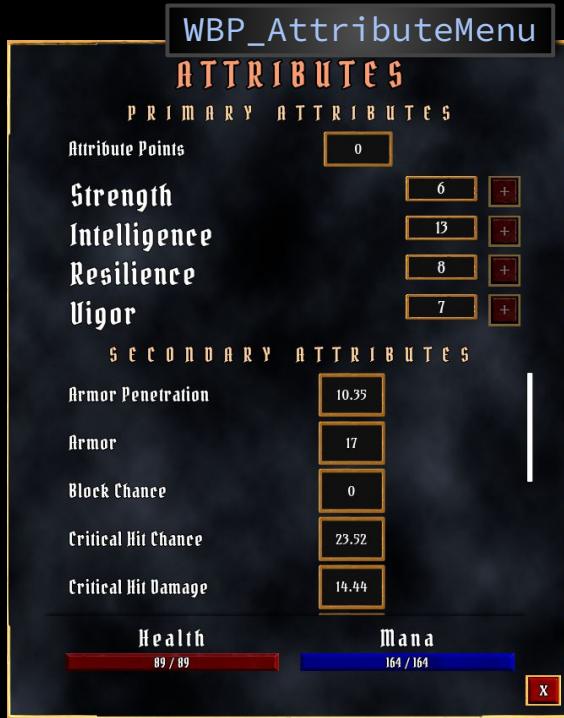
Attribute Row

# Attribute Menu

## Next Steps

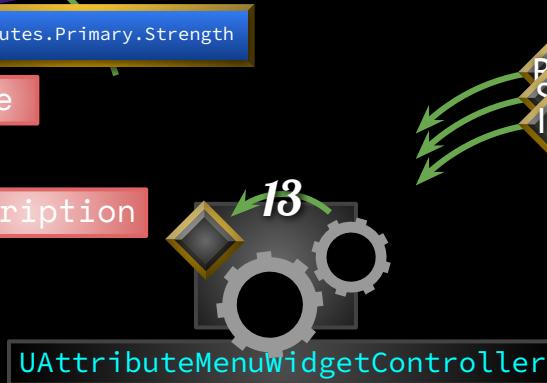
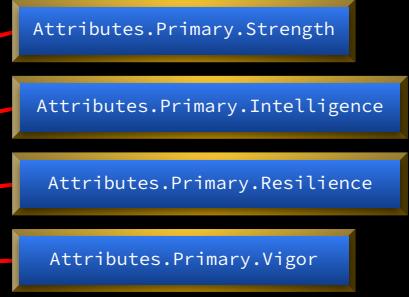
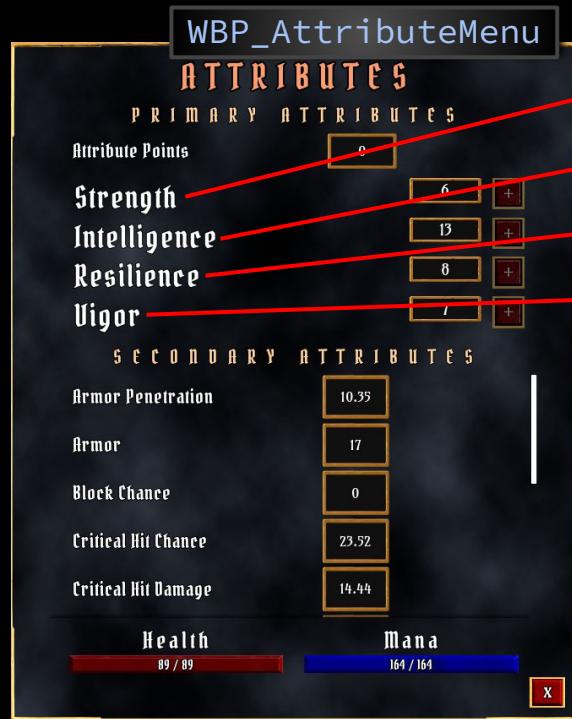
1. Create `WBP_FramedValue` widget
2. Create `WBP_TextValueRow` widget
3. Create `WBP_TextValueButtonRow` widget
4. Create `WBP_AttributeMenu` widget

# Attribute Menu

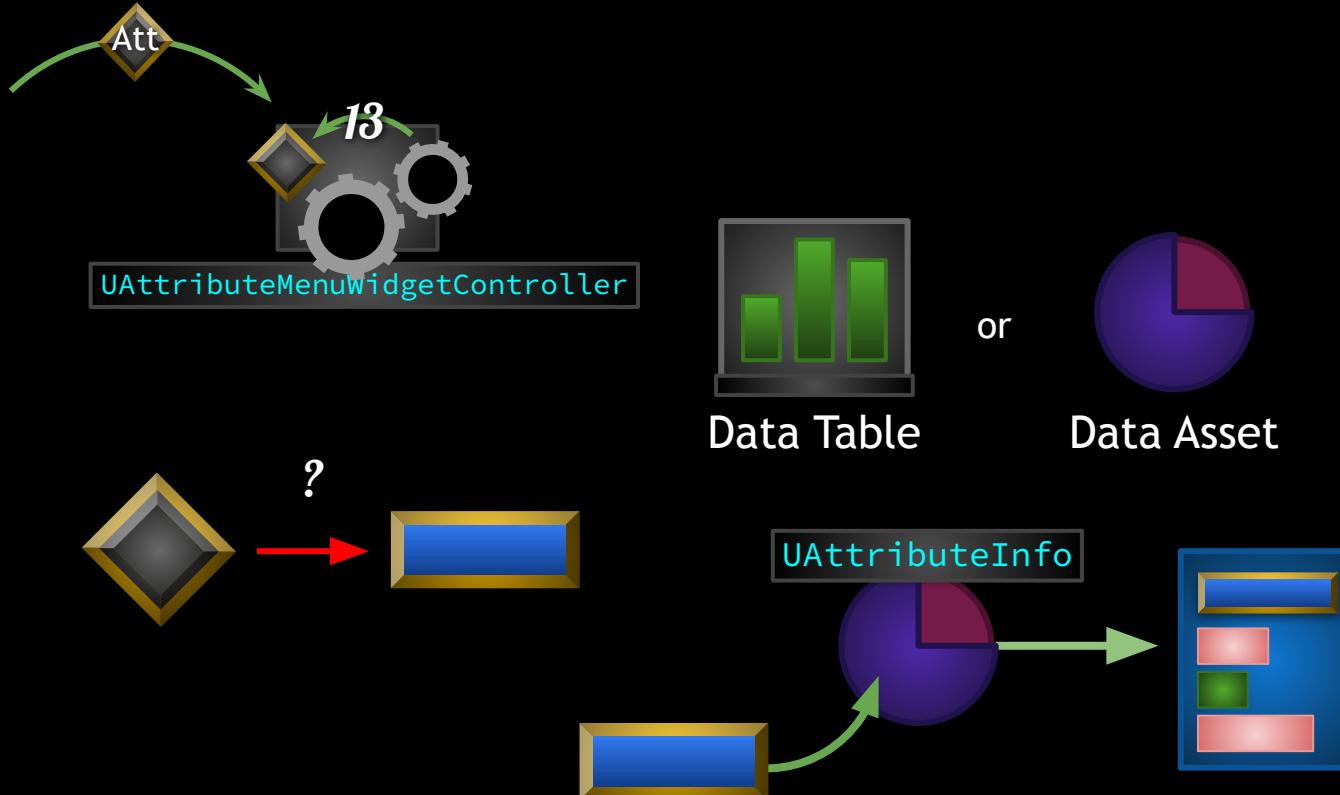


UAttributeMenuItemWidgetController

# Attribute Menu



# Attribute Menu

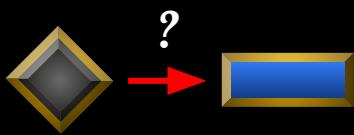


# Attribute Menu

1



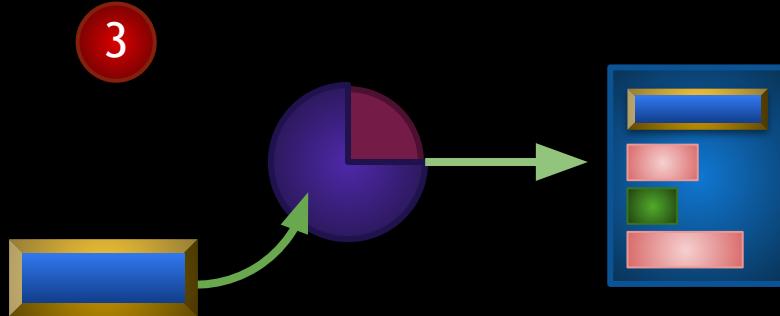
2



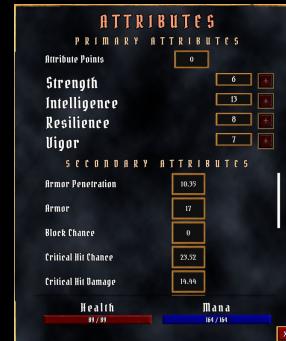
In C++?

```
RequestGameplayTag(FName("Attributes.Primary.Strength"))
```

3



4



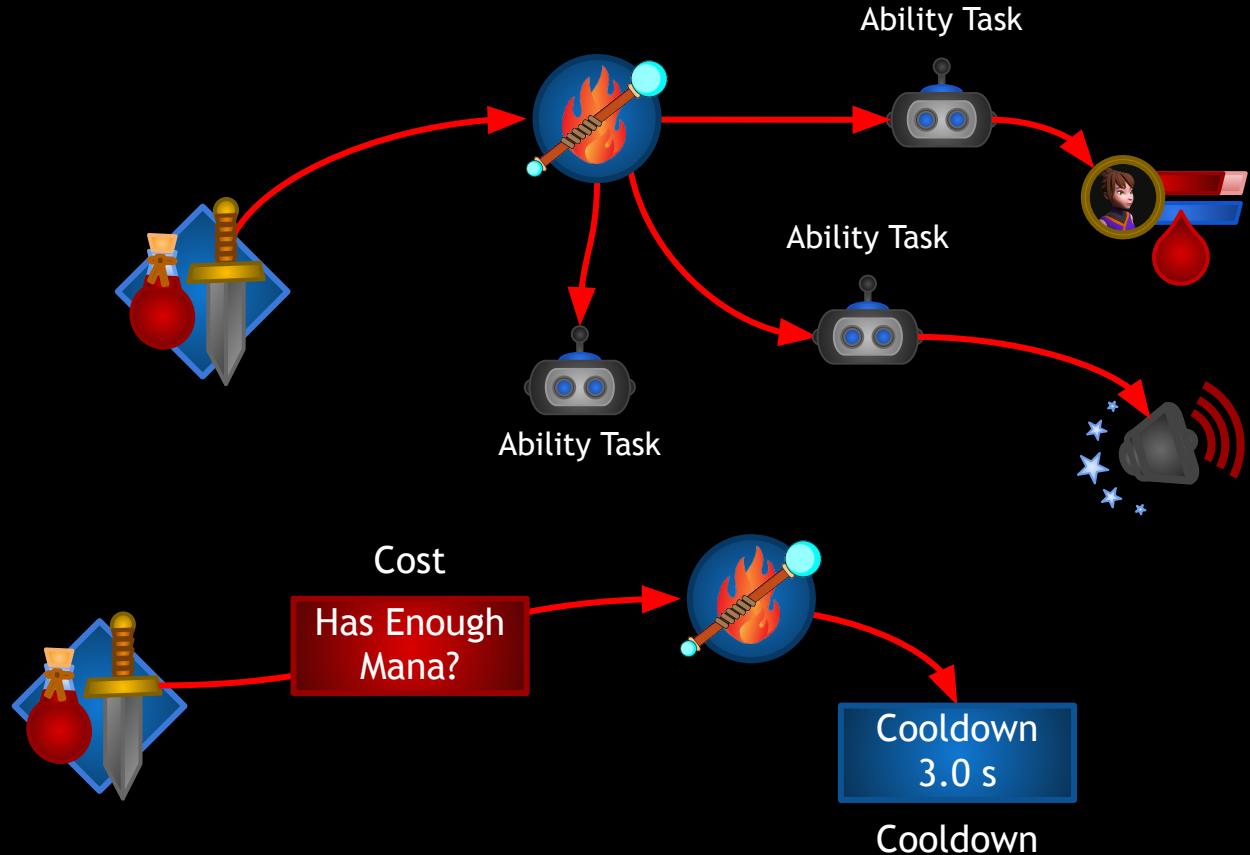
# Attribute Menu

## Next Steps

1. Create Secondary Attribute Gameplay Tags  
(handle tags better in C++)
2. Create UAttributeInfo Data Asset
3. Create FAuraAttributeInfo struct
4. Fill in the Data Asset for each Attribute
5. Create UAttributeMenuWidgetController

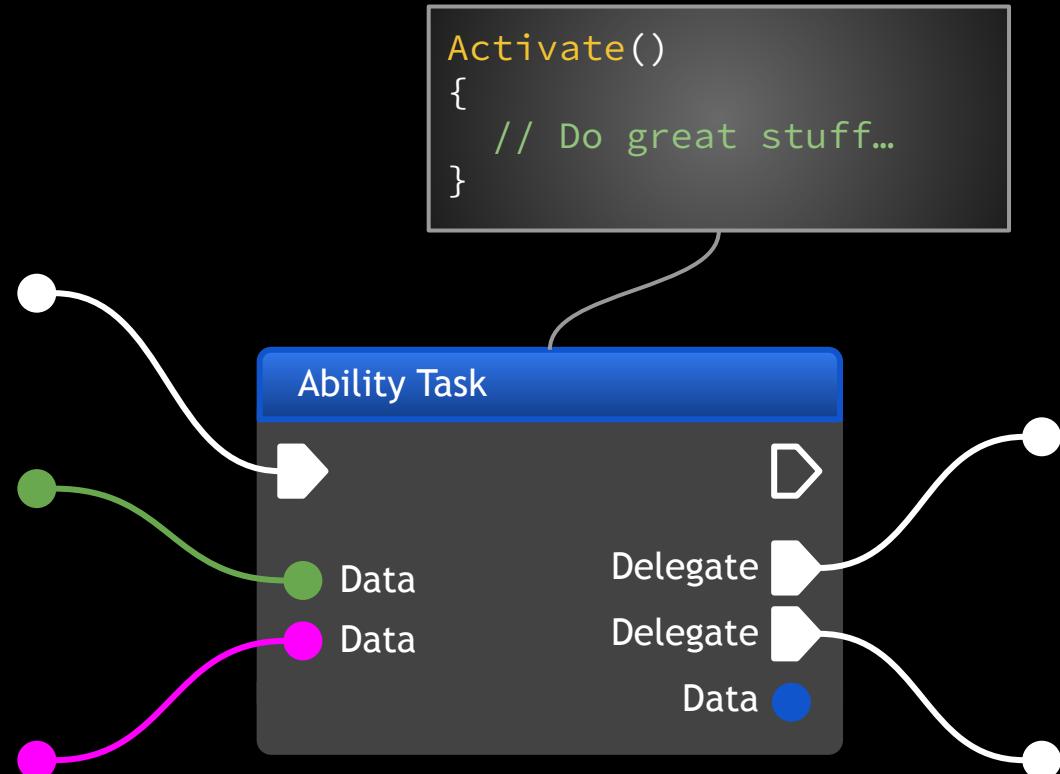
# Gameplay Abilities

Gameplay Ability

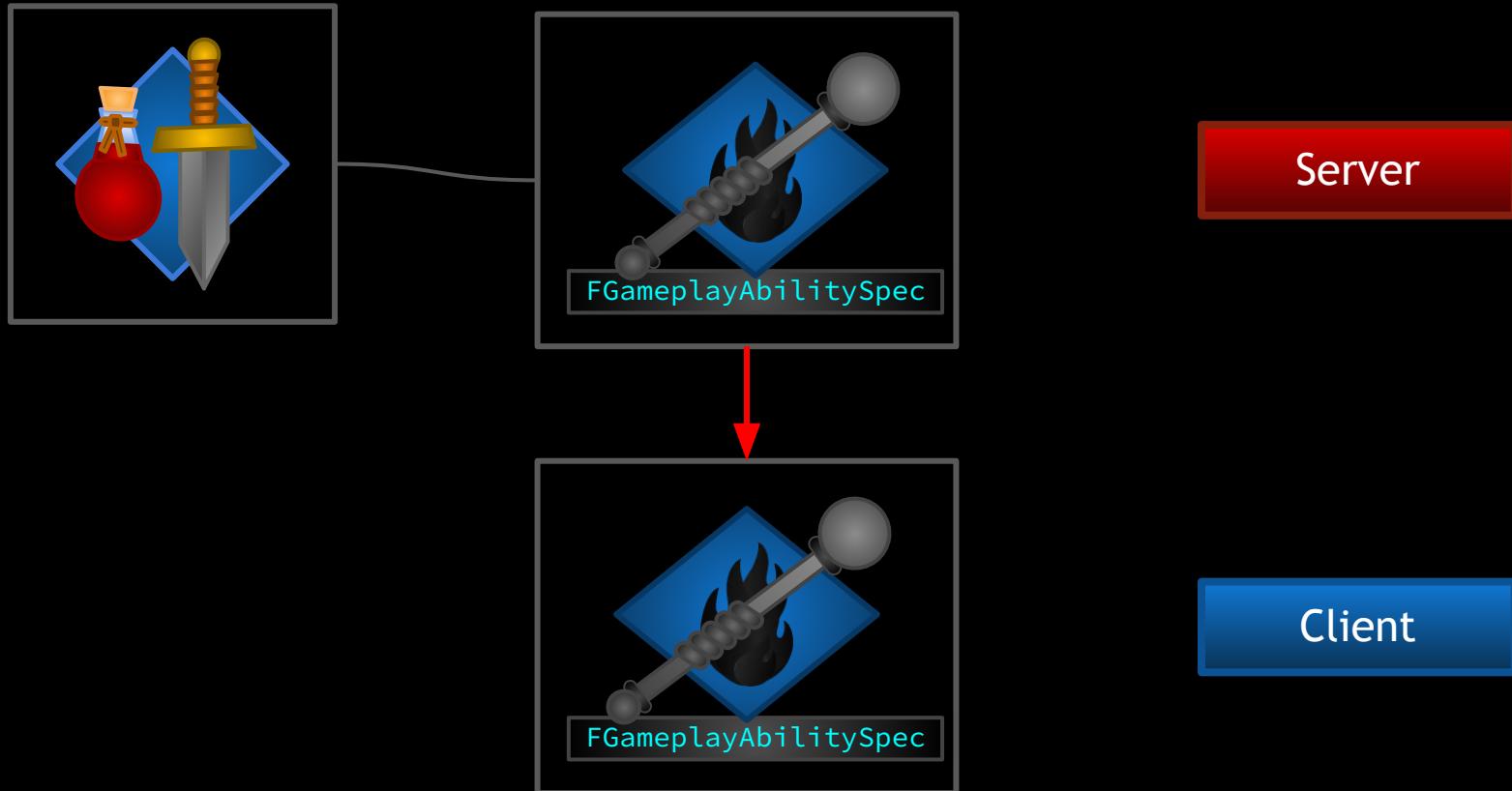


# Ability Tasks

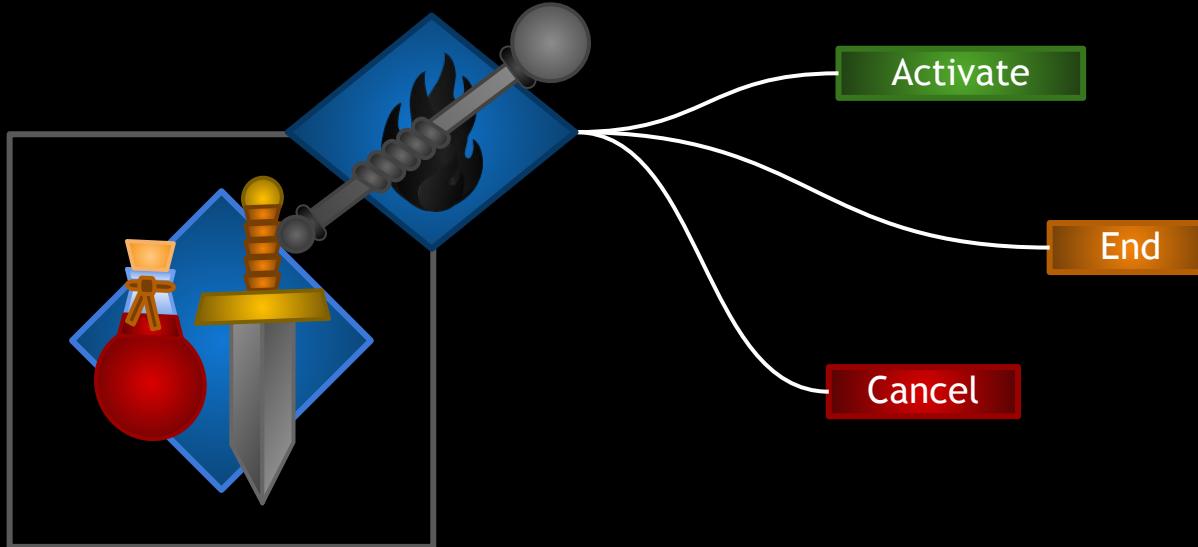
## Ability Tasks



# Granting Abilities



# Granting Abilities



# Summary

## Gameplay Abilities

- Classes that define a skill or ability
- Must be granted
  - Granted on server
  - Spec replicates to owning client
- Must be Activated to be used
- Cost and Cooldown
- Abilities run asynchronously
  - Multiple active at a time
- Ability Tasks
  - Asynchronously perform operations

# Tags

Gameplay Tag	Description
Ability Tags	<i>This ability has these tags</i>
Cancel Abilities with Tag	<i>Abilities with these tags are cancelled when this ability is executed</i>
Block Abilities with Tag	<i>Abilities with these tags are blocked while this ability is active</i>
Activation Owned Tags	<i>Tags to apply to activating owner while this ability is active. These are replicated if ReplicateActivationOwnedTags is enabled in AbilitySystemGlobals</i>
Activation Required Tags	<i>This ability can only be activated if the activating actor/component has all of these tags</i>
Activation Blocked Tags	<i>This ability is blocked if the activating actor/component has any of these tags</i>
Source Required Tags	<i>This ability can only be activated if the source actor/component has all of these tags</i>
Source Blocked Tags	<i>This ability is blocked if the source actor/component has any of these tags</i>
Target Required Tags	<i>This ability can only be activated if the target actor/component has all of these tags</i>
Target Blocked Tags	<i>This ability is blocked if the target actor/component has any of these tags</i>

# Instancing Policy

Instancing Policy	Description	Details
Instanced Per Actor	A single instance is created for the ability. It is reused with each activation.	Can store persistent data. Variables must be manually reset each time.
Instanced Per Execution	New instance created with each activation	Does not store persistent data between activations. Less performant than Instanced per Actor
Non-Instanced	Only the Class Default Object is used, no instances are created.	Cannot store state, cannot bind to delegates on Ability Tasks. Best performance of the three options.

# Net Execution Policy

Net Execution Policy	Description
Local Only	Only run on the local client. Server does not run the ability.
Local Predicted	Activate on the local client, and then on the Server. Makes use of prediction. Server can roll back invalid changes.
Server Only	Only run on the Server.
Server Initiated	Run on the Server first, then on the owning local client.

# Things Not to Use:

## Replication Policy

- Useless. Don't use it. Refer to Epic's Ability System Questions for an explanation from Epic.
- Gameplay Abilities are replicated from Server to owning Client already.
  - Note: Gameplay Abilities don't run on Simulated Proxies (use GEs and GCs)

## Server Respects Remote Ability Cancellation

- Means when the local Client's ability ends, the server's will end
  - Not typically a good idea; it's the Server's version that matters

## Replicate Input Directly

- Always replicates input press/release events to the Server.
  - Epic discourages it

```
/** Direct Input state replication. These will be called if bReplicateInputDirectly is true on the ability and  
is generally not a good thing to use. (Instead, prefer to use Generic Replicated Events). */
```

```
UAbilitySystemComponent::ServerSetInputPressed()
```

# Input

## Binding Input to the Ability System Component

- This was an option before Enhanced Input
- Inputs were bound directly to Abilities
- Enum with Ability Input constants
  - Rigid

## Enhanced Input

- Input Actions are bound to inputs via the Input Mapping Context.
- We can decide how to activate abilities in response to inputs.
  - Lyra provides one example
  - We'll use a similar approach (though less complicated)
- Data Driven
  - Change Input-to-Ability mappings at runtime

# Click To Move

## Top Down Template

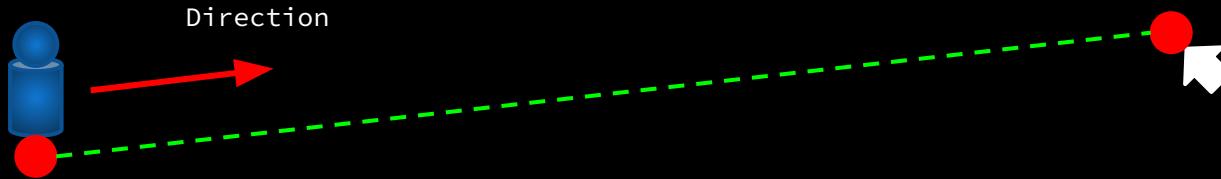
- Uses `SimpleMoveToLocation` if it was a short press
  - Does not work in multiplayer (only AI controlled on server)
- Uses `AddMovementInput` if input is held down
  - Works in multiplayer
  - Requires constant input (movement direction)

## Our GAS Project

- We must use `AddMovementInput`
- Need a direction each frame

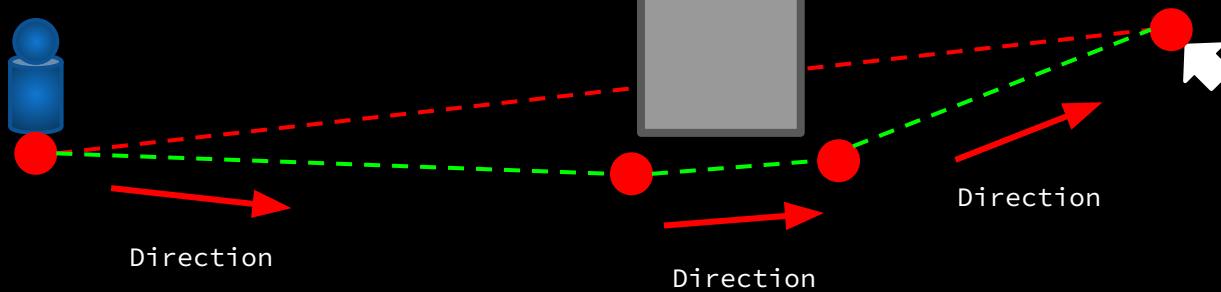
# Click To Move

AddMovementInput(Direction)

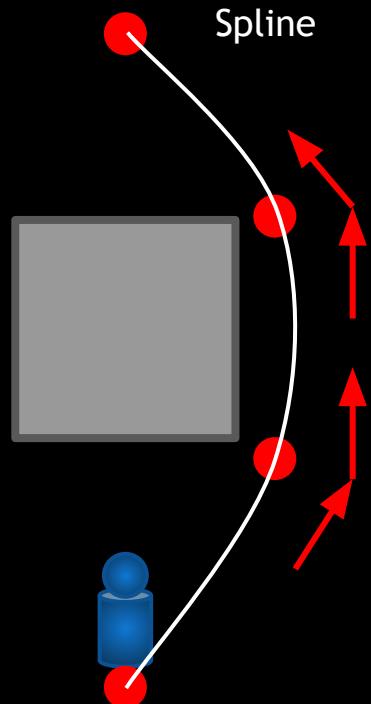


# Click To Move

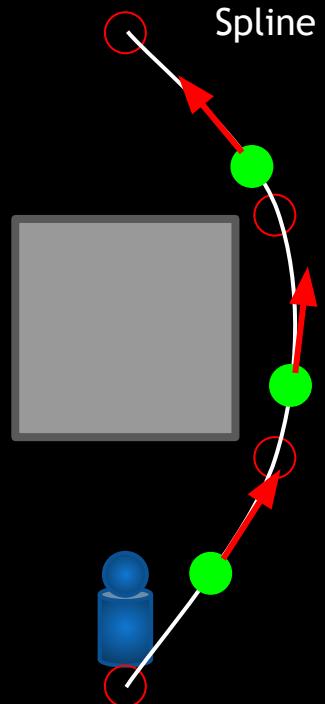
AddMovementInput(Direction)



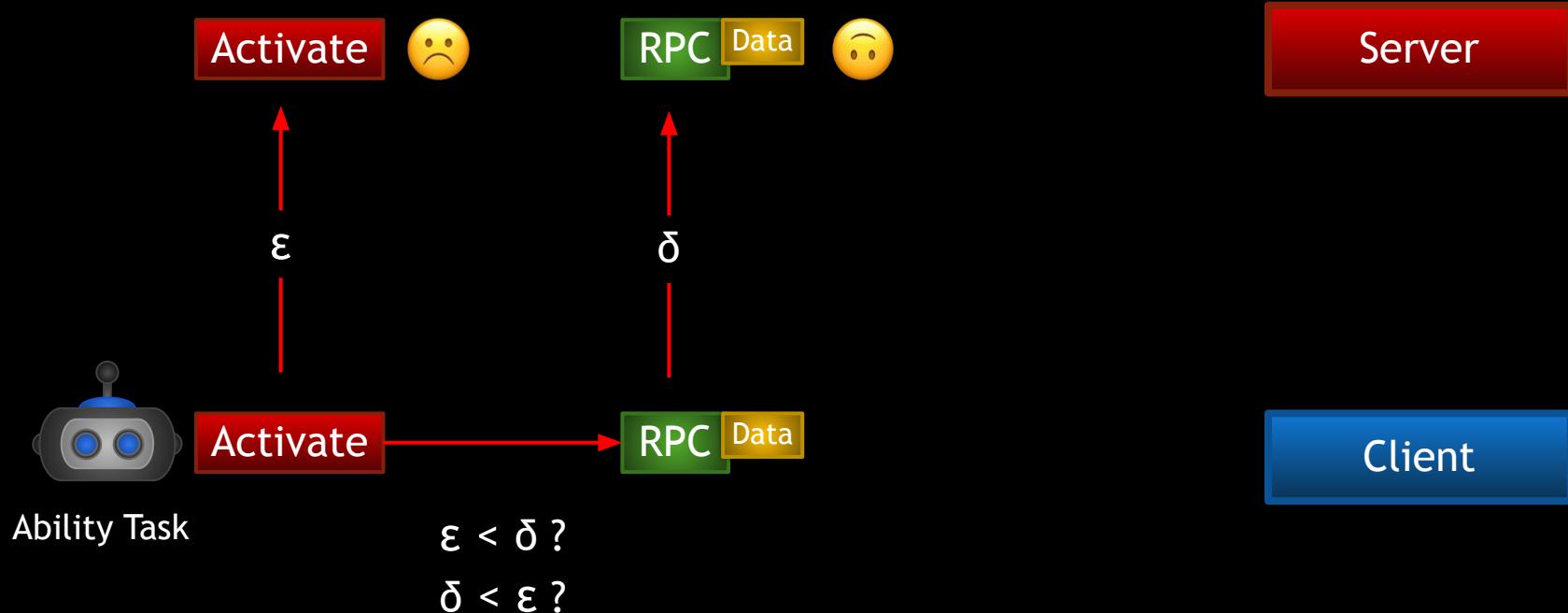
# Click To Move

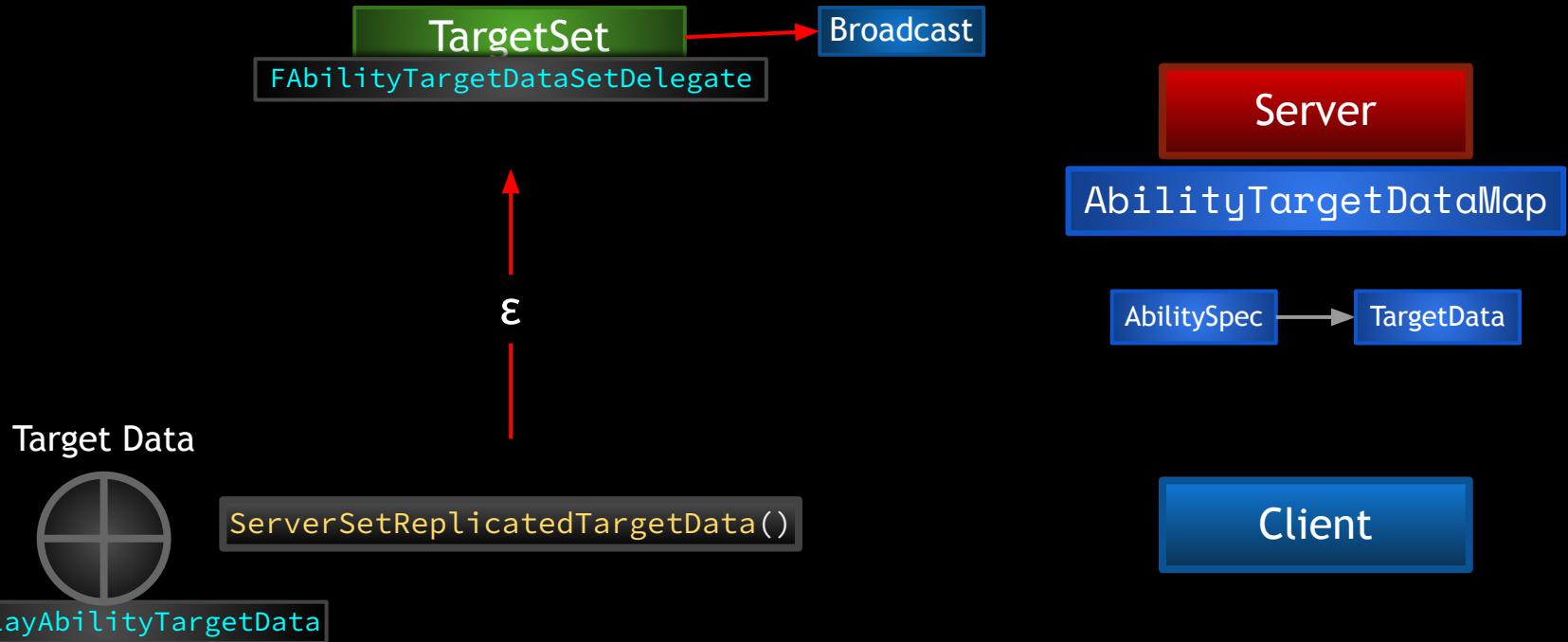


# Click To Move

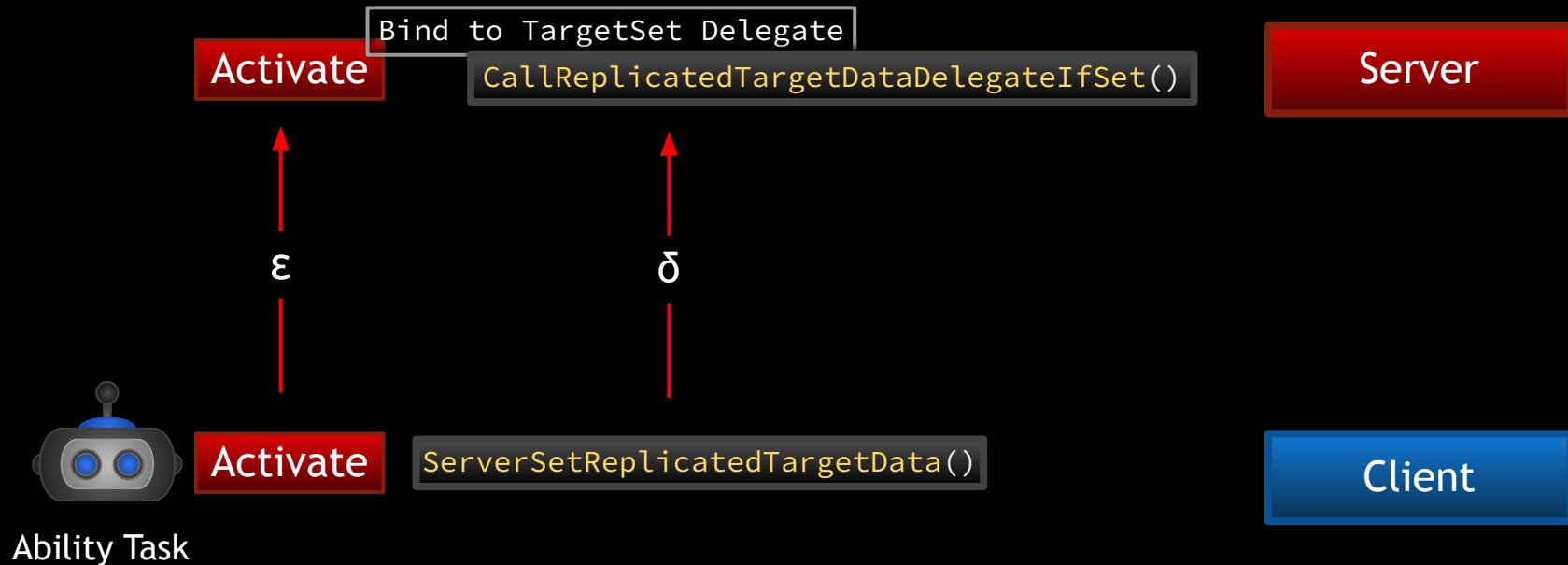


# Target Data

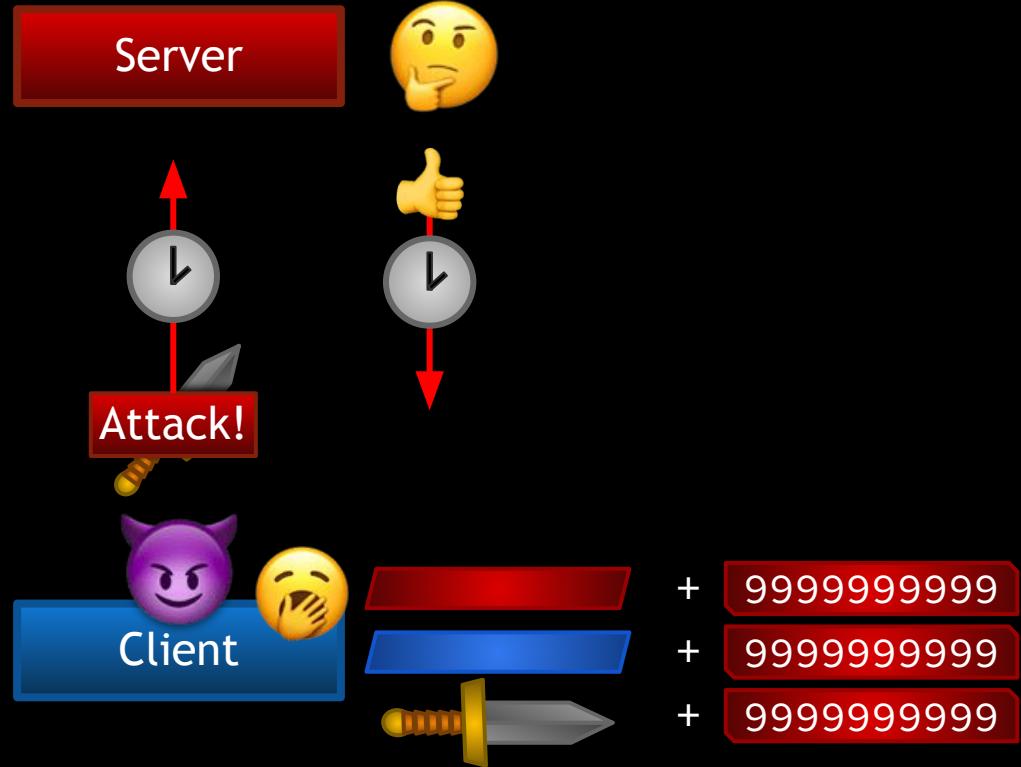




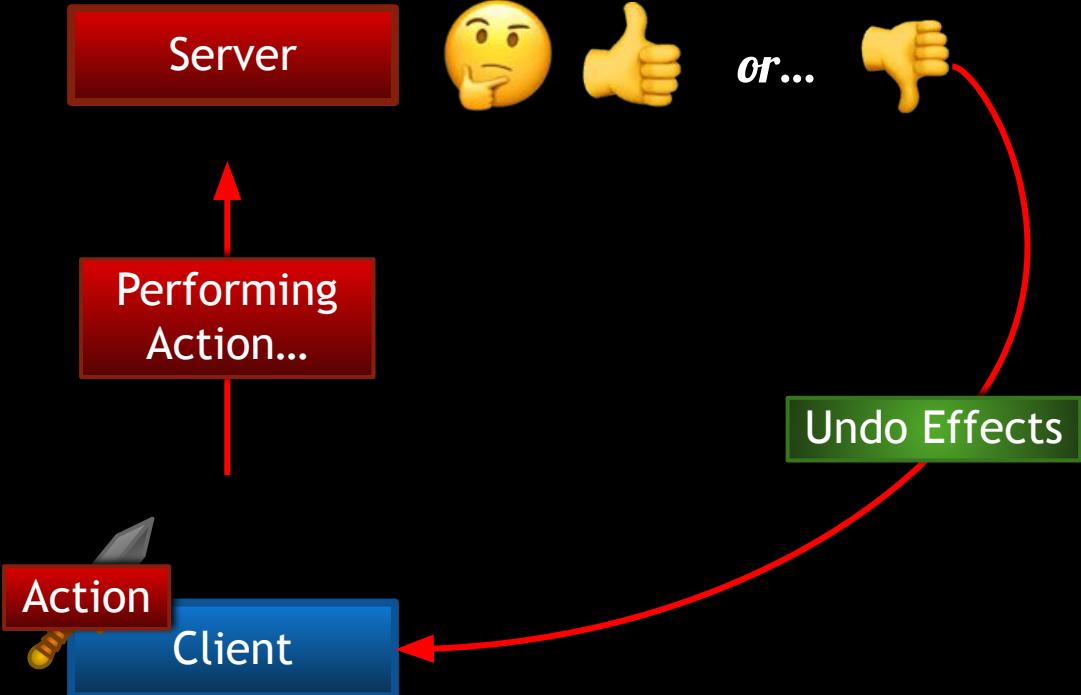
# Target Data



# Prediction



# Prediction



# GAS Prediction

## Prediction in GAS

- Just works

## Developers Didn't Want:

- `if (Authority) Do X`
- `else: Do predicted version...`

## Developers Did Want:

- Automatic
  - Not everything needs to be predicted
    - Footstep sounds, etc.
  - Use mana, heal, do damage, etc.

# GAS Prediction

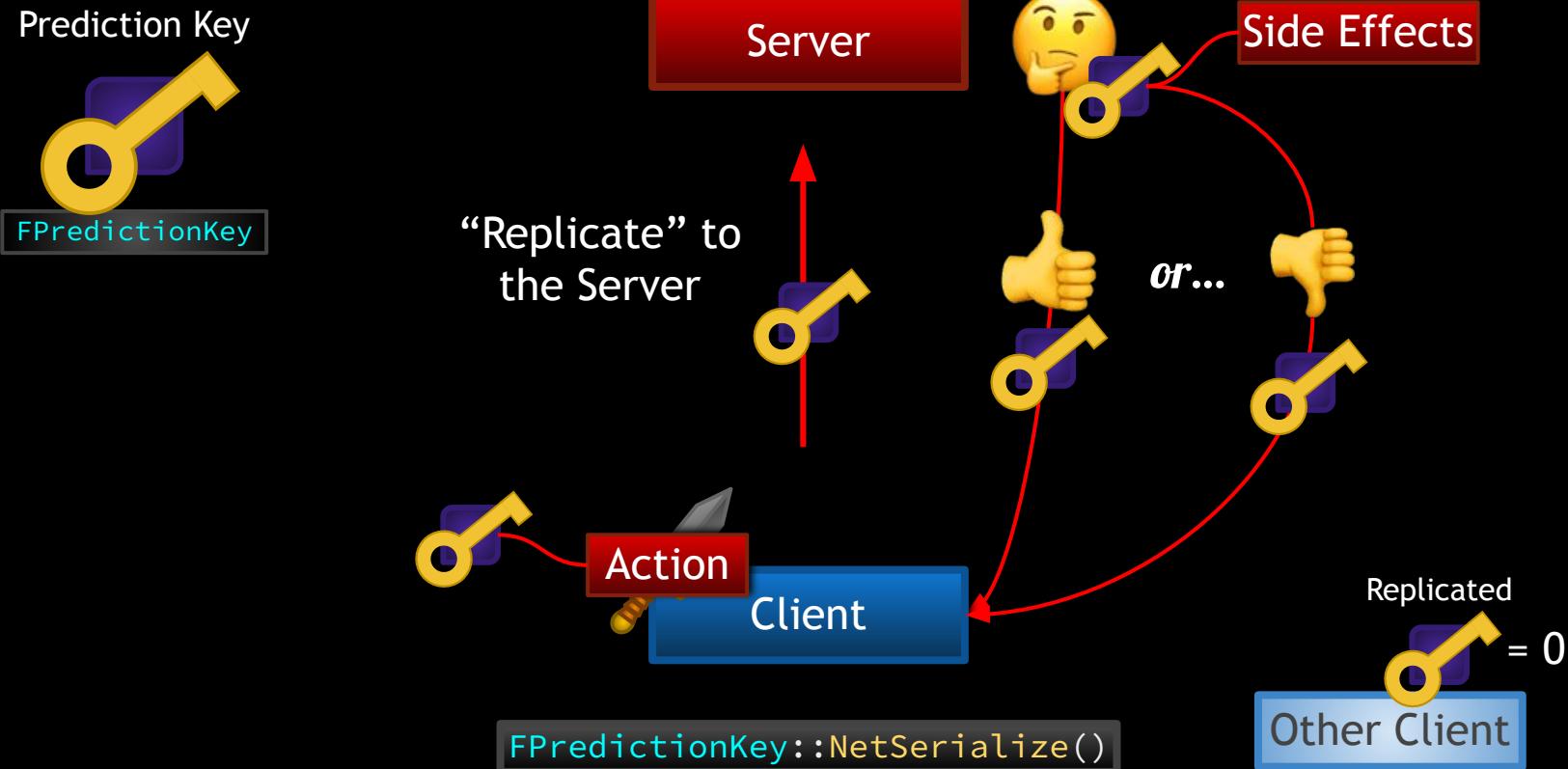
## GAS Automatically Predicts:

- Gameplay Ability Activation
- Triggered Events
- Gameplay Effect Application
  - Attribute Modifiers (not Execution Calculations)
  - GameplayTag Modification
- Gameplay Cue Events
  - From within a predicted Gameplay Ability
  - Their own Events
- Montages
- Movement (UCharacterMovement)

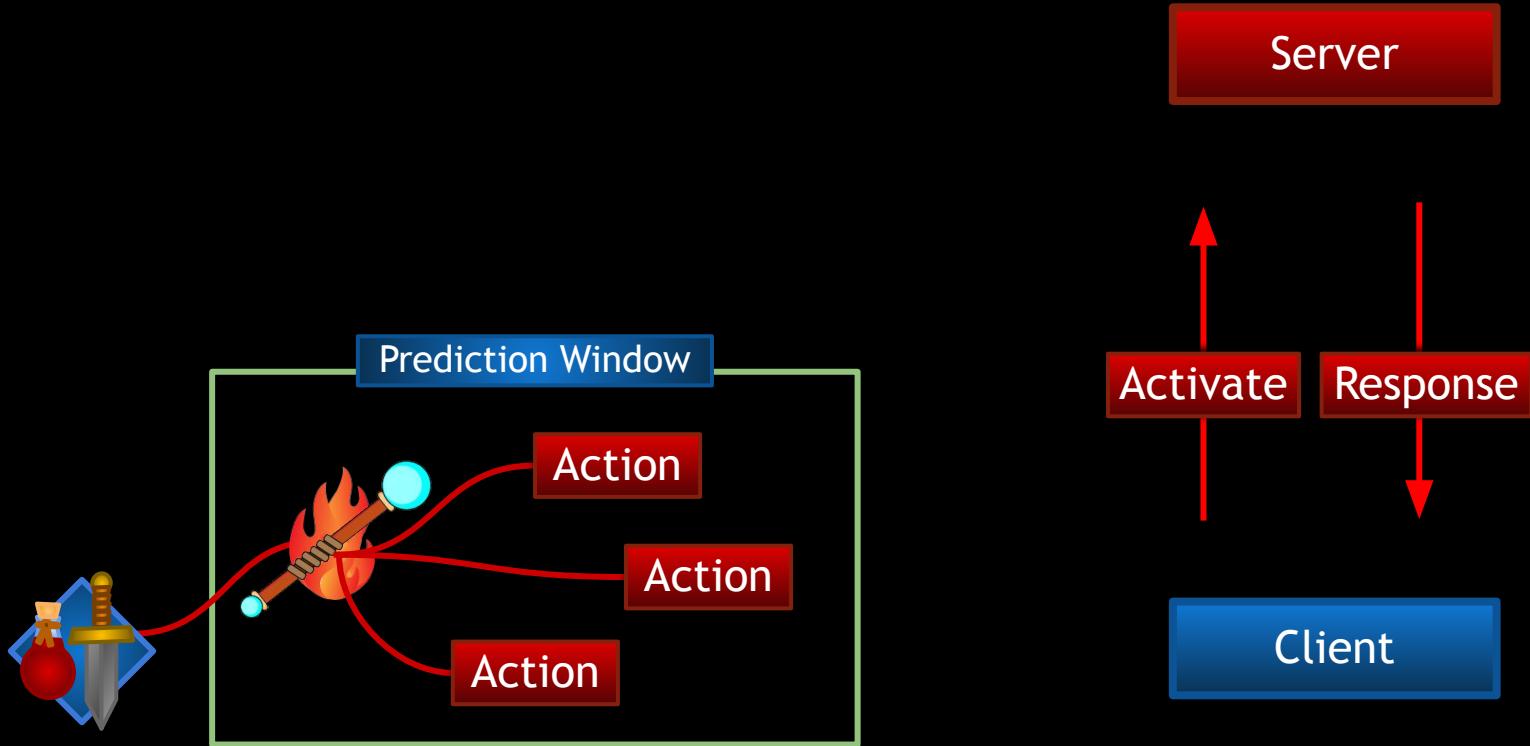
## GAS Does NOT Predict:

- Gameplay Effect Removal
- Gameplay Effect Periodic Effects

# Prediction Key



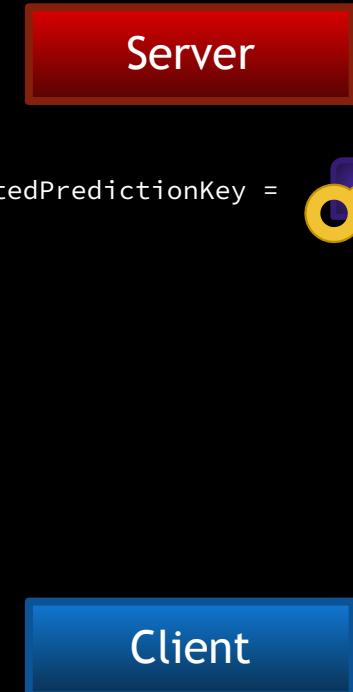
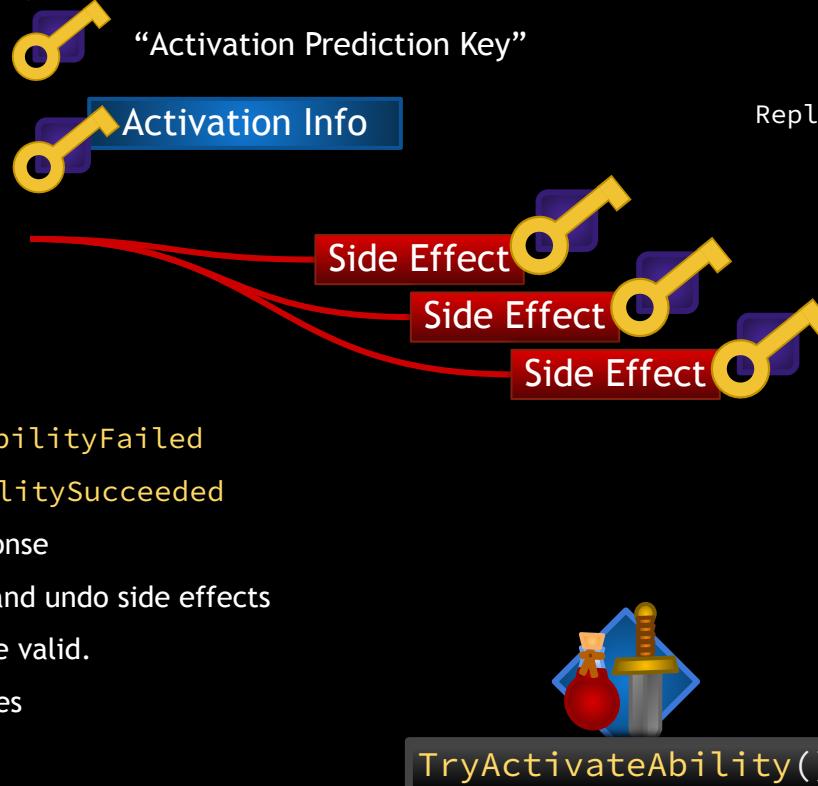
# Ability Activation



# Ability Activation

## TryActivateAbility:

- Client calls TryActivateAbility
  - New `FPredictionKey`
- Client continues
  - Calls `ActivateAbility`
- Client does things
  - Generates side effects
- `ServerTryActivateAbility`
  - Server decides if valid
  - calls `ClientActivateAbilityFailed`
  - or `ClientActivateAbilitySucceeded`
- Client receives the Server's response
  - If failure, kill the ability and undo side effects
  - If success, side effects are valid.
- `ReplicatedPredictionKey` replicates
  - `OnRep_PredictionKey`



# Gameplay Effect Prediction

## Gameplay Effects

- Side effects
- Only applied on Clients if:
  - There is a valid prediction key
- The following are predicted:
  - Attribute Modifications
  - Gameplay Tag Modifications
  - Gameplay Cues
- When the `FActiveGameplayEffect` is created
  - Stores the Prediction Key
- On the server, it gets the same key
- `FActiveGameplayEffect` is replicated
  - Client checks the key
  - If they match, then “OnApplied” logic doesn’t need to be done

ReplicatedPredictionKey



Replicates

OnRep\_PredictionKey



Active Gameplay Effect

# GAS Prediction

For More Info:

- `GameplayPrediction.h`

# RPG Character Classes

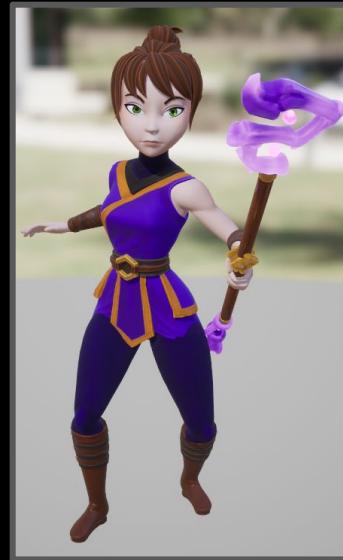
Warrior



Ranger



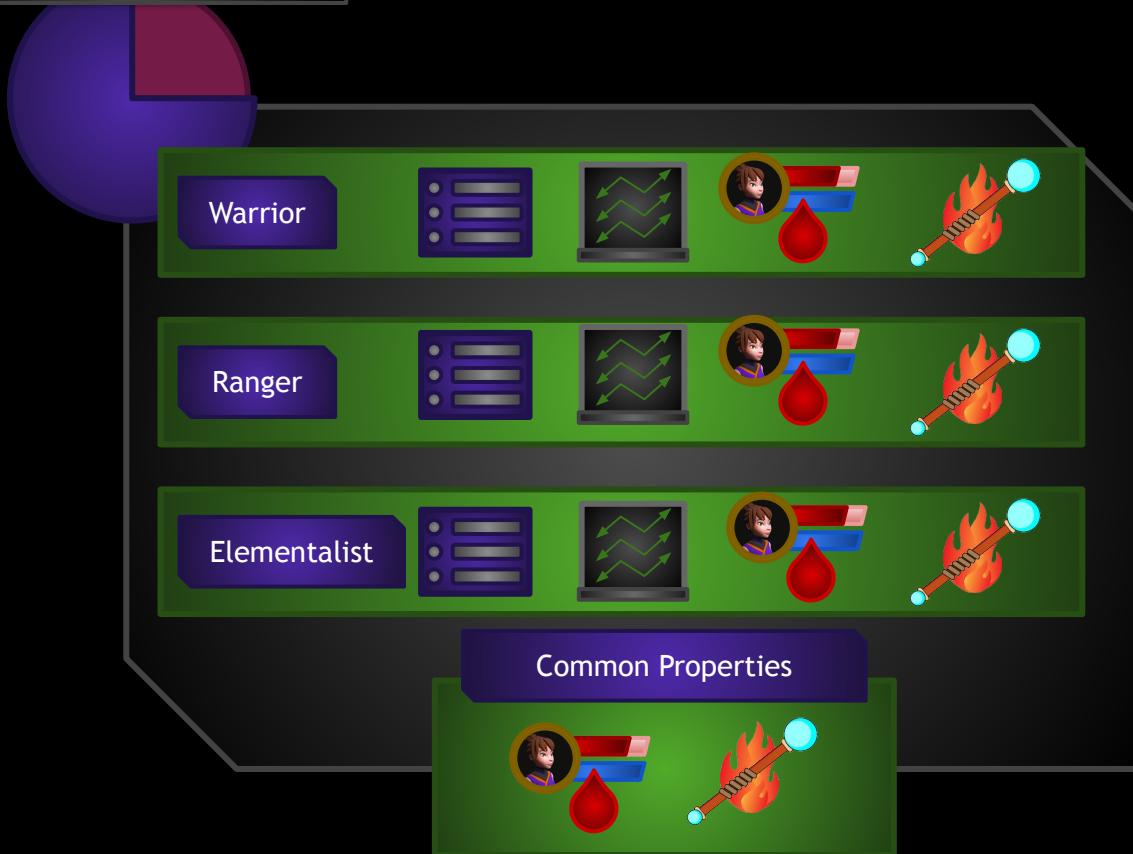
Elementalist



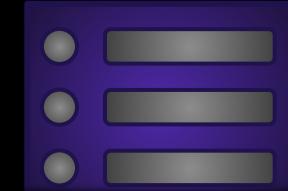
# RPG Character Classes

Data Asset

`UCharacterClassInfo`



Enum



`ECharacterClass`

Curve Table



`CT_PrimaryAttributes`

# RPG Character Classes

## Next Steps

1. Create `UCharacterClassInfo` Data Asset
2. Create `ECharacterClass` enum
3. Curve Tables for Attributes
  - a. `CT_WarriorPrimaryAttributes`
  - b. `CT_RangerPrimaryAttributes`
  - c. `CT_ElementalistPrimaryAttributes`
4. Create GEs for Primary, Secondary, and Vital Attributes
5. Shared Abilities and Effects
6. Function to initialize attributes using the Data Asset

# Meta Attributes

## Gameplay Attribute



- Replicated

## Meta Attribute



- Not Replicated
- Temporary Placeholder
- Allows for Calculations

## Incoming Damage



11



## Attribute Set

- Block?
- Critical Hit?
- Intelligence Bonus?
- Strength Bonus?

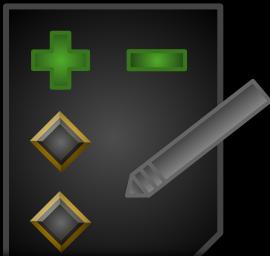
## Change Health



-4

# Execution Calculation

## Execution Calculation



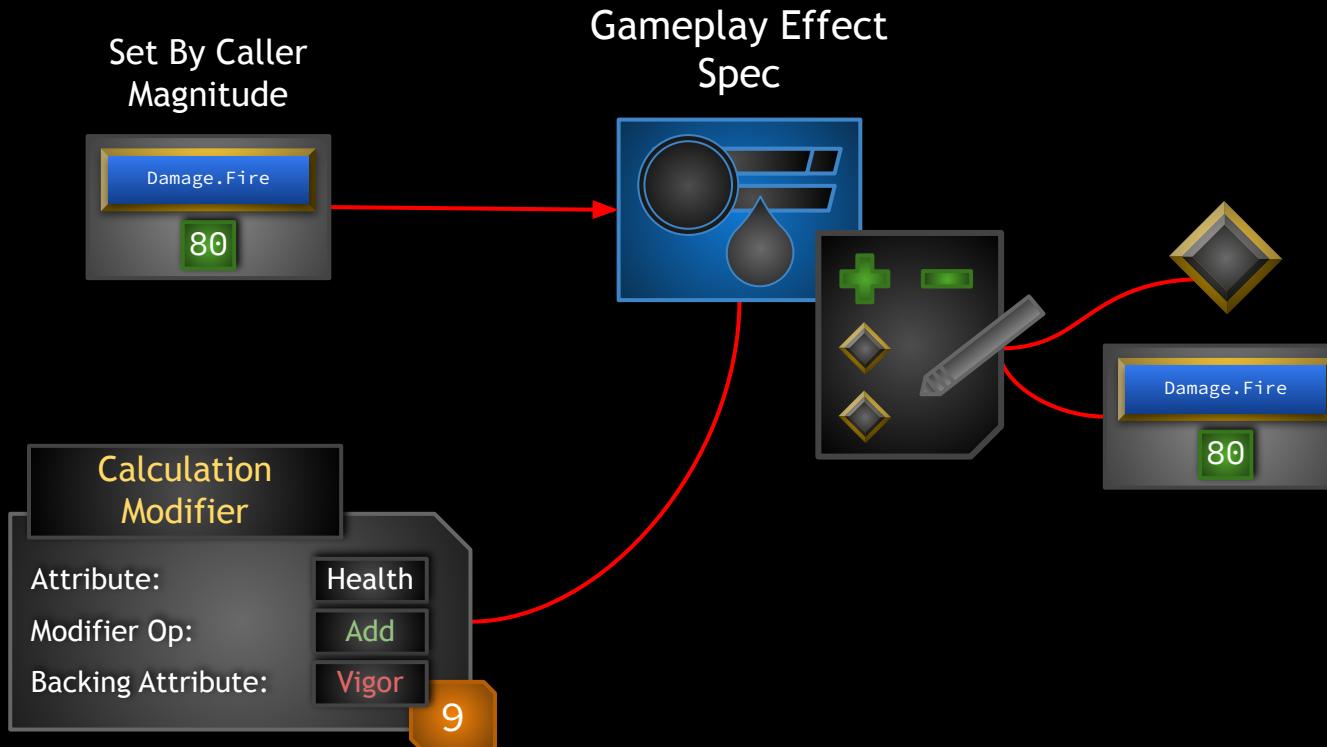
UGameplayEffectExecutionCalculation

- Capture Attributes
- Can change more than one Attribute
- Can have programmer logic
- No prediction
- Only Instant or Periodic Gameplay Effects
- Capturing doesn't run PreAttributeChange; any clamping done there must be done again
- Only executed on the Server from Gameplay Abilities with Local Predicted, Server Initiated, and Server Only Net Execution Policies

## Snapshotting (Source)

- Snapshotting captures the Attribute value when the Gameplay Effect Spec is created
- Not snapshotting captures the Attribute value when the Gameplay Effect is applied
- From the Target, the value is captured on Effect Application only

# Execution Calculation

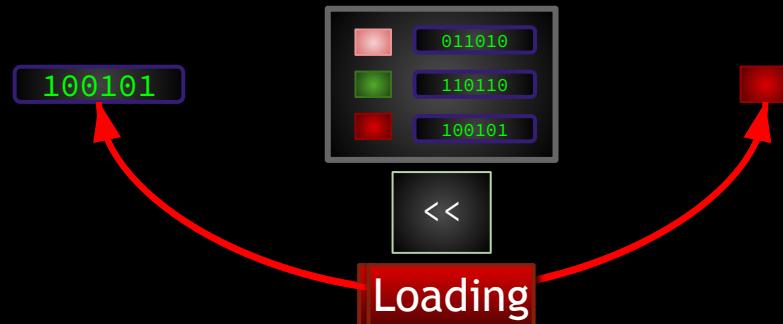


# Net Serialize

## Archive



- Saving Data
- Loading Data
- Storing Data
- Serialized
- Overloads `<<` operator
  - `<<` works both ways depending on context (Saving/Loading)



# Net Serialize

RepBits  
  uint8

= 0000 0000      8 bits

RepBits

| = 1 << 0

RepBits

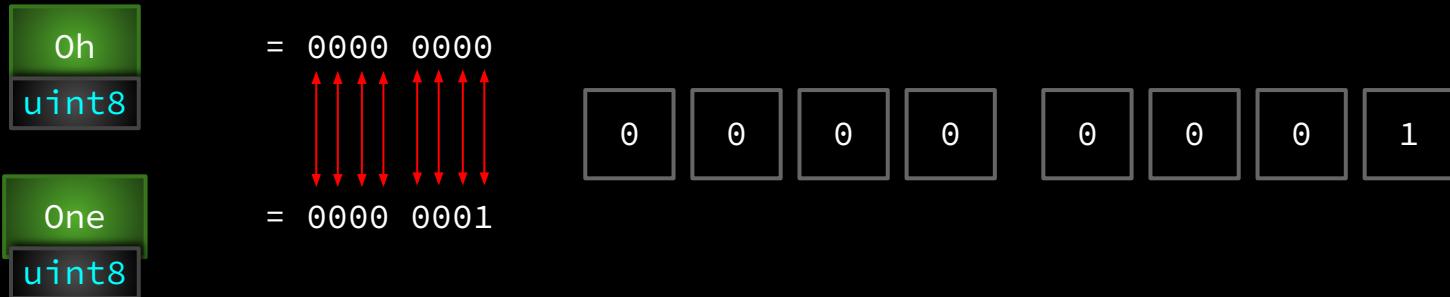
= RepBits |

Bitwise OR

Shift Left

# Net Serialize

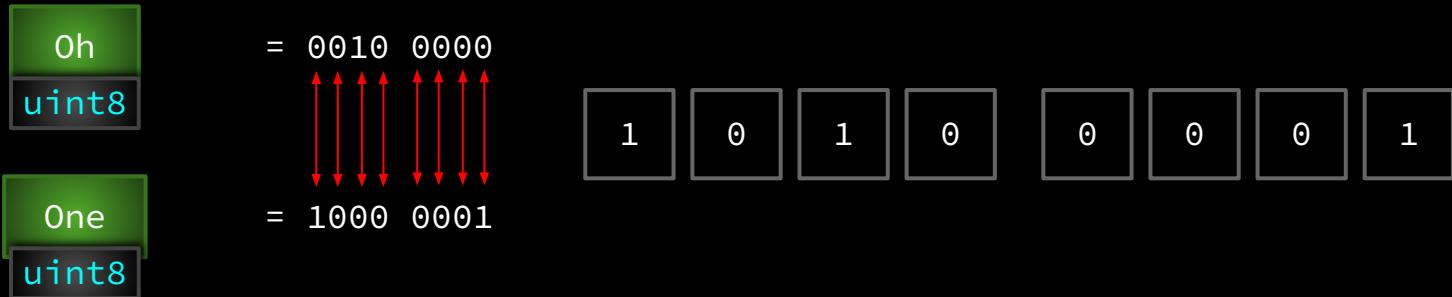
## Bitwise OR



$$\begin{array}{c} \text{0h} \\ | \\ \text{One} \end{array} = ?$$
$$= 0000\ 0001$$

# Net Serialize

## Bitwise OR



$$\begin{array}{c|c} \text{0h} & \text{One} \\ \hline \end{array} = ?$$

`= 1010 0001`

# Net Serialize

Shift Left

$1 \ll 0$



uint32

0 0 0 0 0 0 0 1

Shift left by 0

# Net Serialize

Shift Left

$1 \ll 1$

lost!



0 0 0 0 0 0 0 0 1

Shift left by 1



= 0 0 0 0 0 0 1 0

# Net Serialize

Shift Left

$1 \ll 2$

lost!



0 0 0 0 0 0 0 0 1

Shift left by 2



= 0 0 0 0 0 1 0 0

# Net Serialize

Shift Left

$1 \ll 3$

lost!



0 0 0 0    0 0 0 1

Shift left by 3



= 0 0 0 0    1 0 0 0

# Net Serialize

RepBits  
uint32

= 0000 0000 0000 0000 0000 0000 0000 0000

RepBits

| = 1 << 0

RepBits

= RepBits | 1 << 0

1 << 0 = 0000 0000 0000 0000 0000 0000 0000 0001

Flipped the 0th  
Bit

RepBits | 1 << 0

0000 0000 0000 0000 0000 0000 0000 0000

= 0000 0000 0000 0000 0000 0000 0000 0001

0000 0000 0000 0000 0000 0000 0000 0001



# Net Serialize

RepBits  
uint32

= 0000 0000 0000 0000 0000 0000 0000 0001

RepBits

| = 1 << 1

1 << 1 = 0000 0000 0000 0000 0000 0000 0000 0010

Flipped the 1st  
Bit

RepBits | 1 << 1

0000 0000 0000 0000 0000 0000 0000 0001

= 0000 0000 0000 0000 0000 0000 0000 0011

0000 0000 0000 0000 0000 0000 0000 0010



# Net Serialize

RepBits  
uint32

= 0000 0000 0000 0000 0000 0000 0000 0011

RepBits

| = 1 << 2

1 << 2 = 0000 0000 0000 0000 0000 0000 0000 0100

Flipped the 2nd  
Bit

RepBits

| 1 << 2

0000 0000 0000 0000 0000 0000 0000 0011

0000 0000 0000 0000 0000 0000 0000 0100

= 0000 0000 0000 0000 0000 0000 0000 0111



# Net Serialize

RepBits  
uint32

= 0000 0000 0000 0000 0000 0000 0000 0111

RepBits

& 1 << 0

1 << 0 = 0000 0000 0000 0000 0000 0000 0000 0001

RepBits

&

1 << 0

true

0000 0000 0000 0000 0000 0000 0000 0111

= 0000 0000 0000 0000 0000 0000 0000 0001

& 0000 0000 0000 0000 0000 0000 0000 0001

# Net Serialize

RepBits  
uint32

= 0000 0000 0000 0000 0000 0000 0000 0111

RepBits

& 1 << 3

1 << 3 = 0000 0000 0000 0000 0000 0000 0000 1000

RepBits & 1 << 3

0000 0000 0000 0000 0000 0000 0000 0111

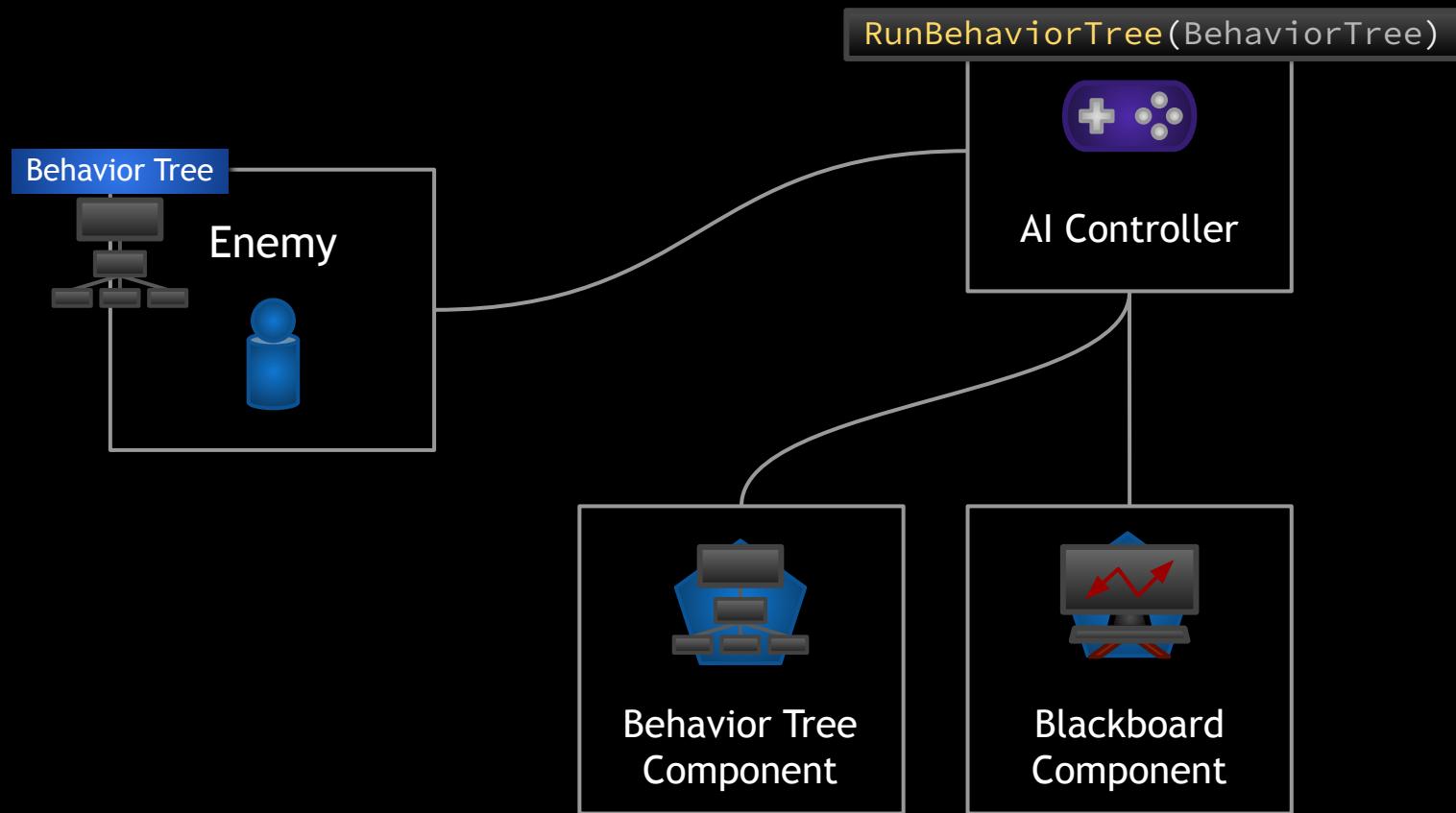
& 0000 0000 0000 0000 0000 0000 0000 1000

= 0000 0000 0000 0000 0000 0000 0000 0000

false



# Enemy AI

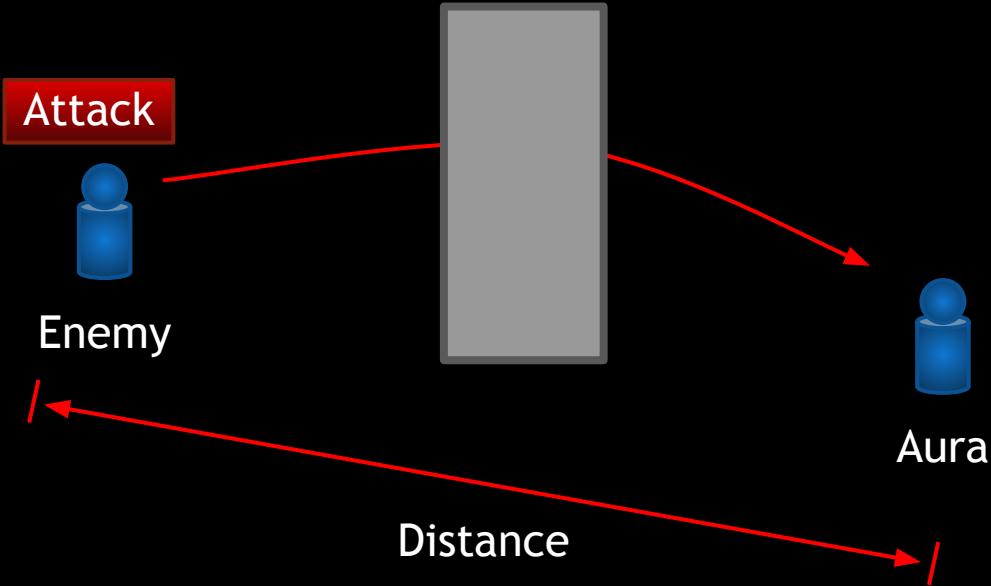


# Enemy AI

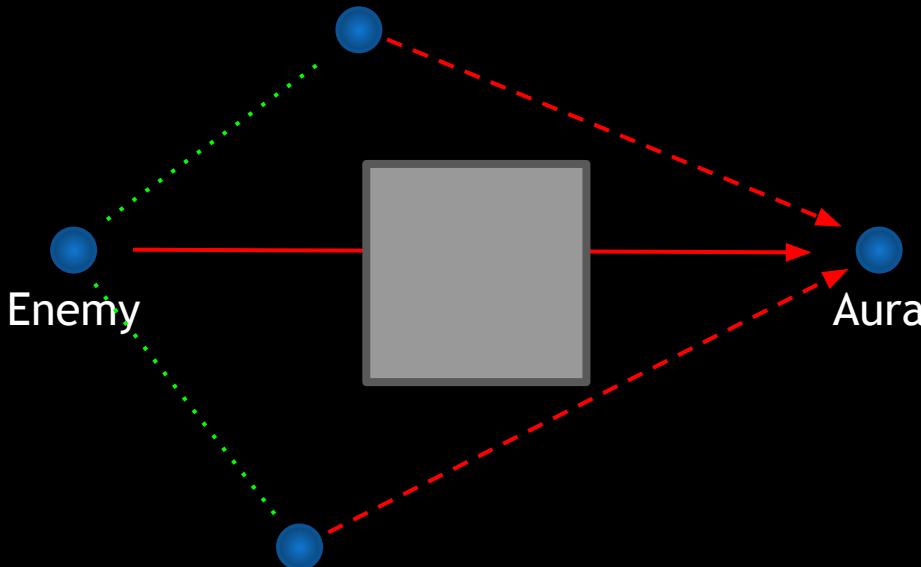
## Next Steps

1. Create an AI Controller class
2. Create a Blackboard and Behavior Tree
3. Add a Blackboard Component and Behavior Tree Component to the AI Controller
4. Add a Behavior Tree to the Aura Enemy
5. Run the Behavior Tree

# Environment Query System



# Environment Query System

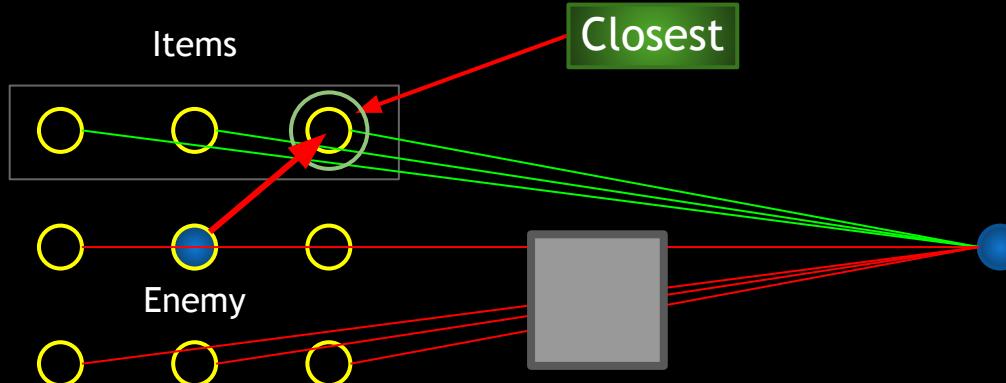


TOP VIEW:

# Environment Query System

## Tests

- Trace
- Distance



TOP VIEW:

# Experience (XP)



Level

Level Up

XP

# Experience (XP)

Level Up Requirement



Level

0

XP

300

300



Level

301

XP

900

900

(600 more)



Level

901

XP

2700

2700

(1800 more)



Level

2701

XP

6500

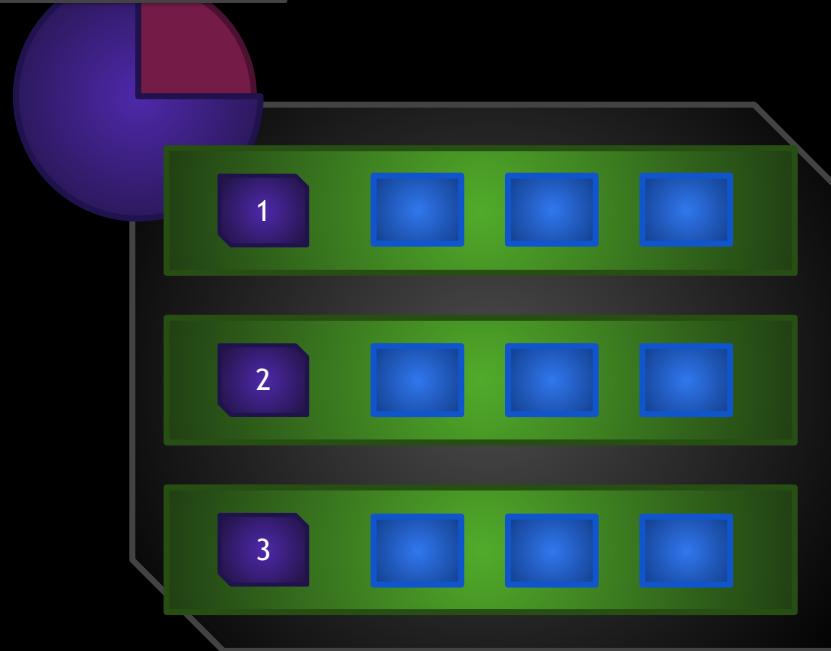
6500

(3800 more)

$$R = 100 * 3^{\text{Level}}$$

# Experience (XP)

Data Asset  
`ULevelUpInfo`



`FindLevelForXP(float XP)`

Level Up Requirement



`int32`

Attribute Point Reward



`int32`

Spell Point Reward



`int32`

# Experience (XP)



XP

180

1180

Double Level Up

+1000 XP!

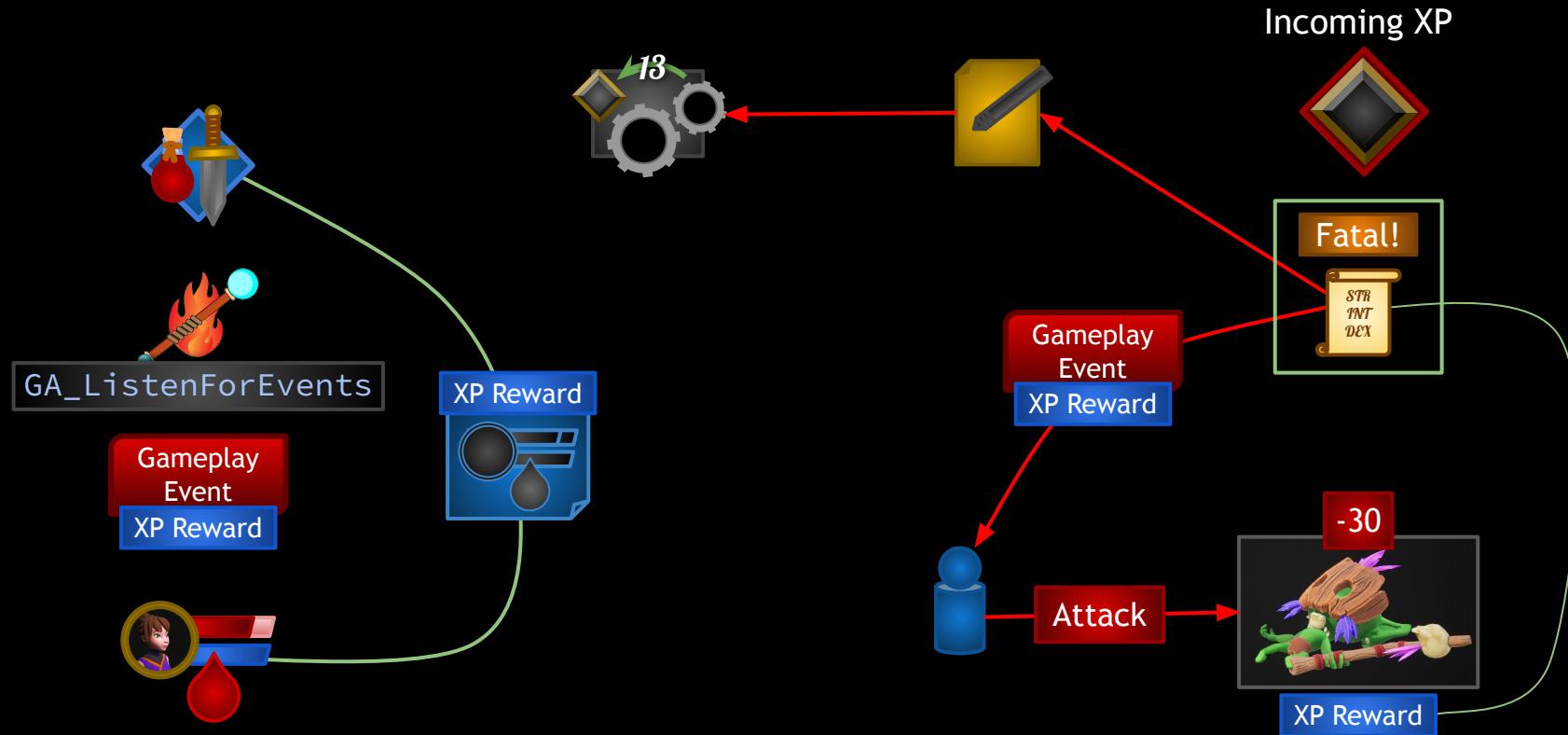


# Experience (XP)

## Next Steps

1. Level Up Info Data Asset
2. Add XP to the Player State (with Rep Notify)
3. Delegate to broadcast when XP changes
4. Widget Controller response to delegate broadcast
5. XP Reward - amount of XP each enemy gives based on its level and class
6. Actually award XP when killing an enemy
7. Handle Level Up (handle potentially multiple Level Ups)
  - a. Reward Attribute Points
  - b. Reward Spell Points

# Awarding XP

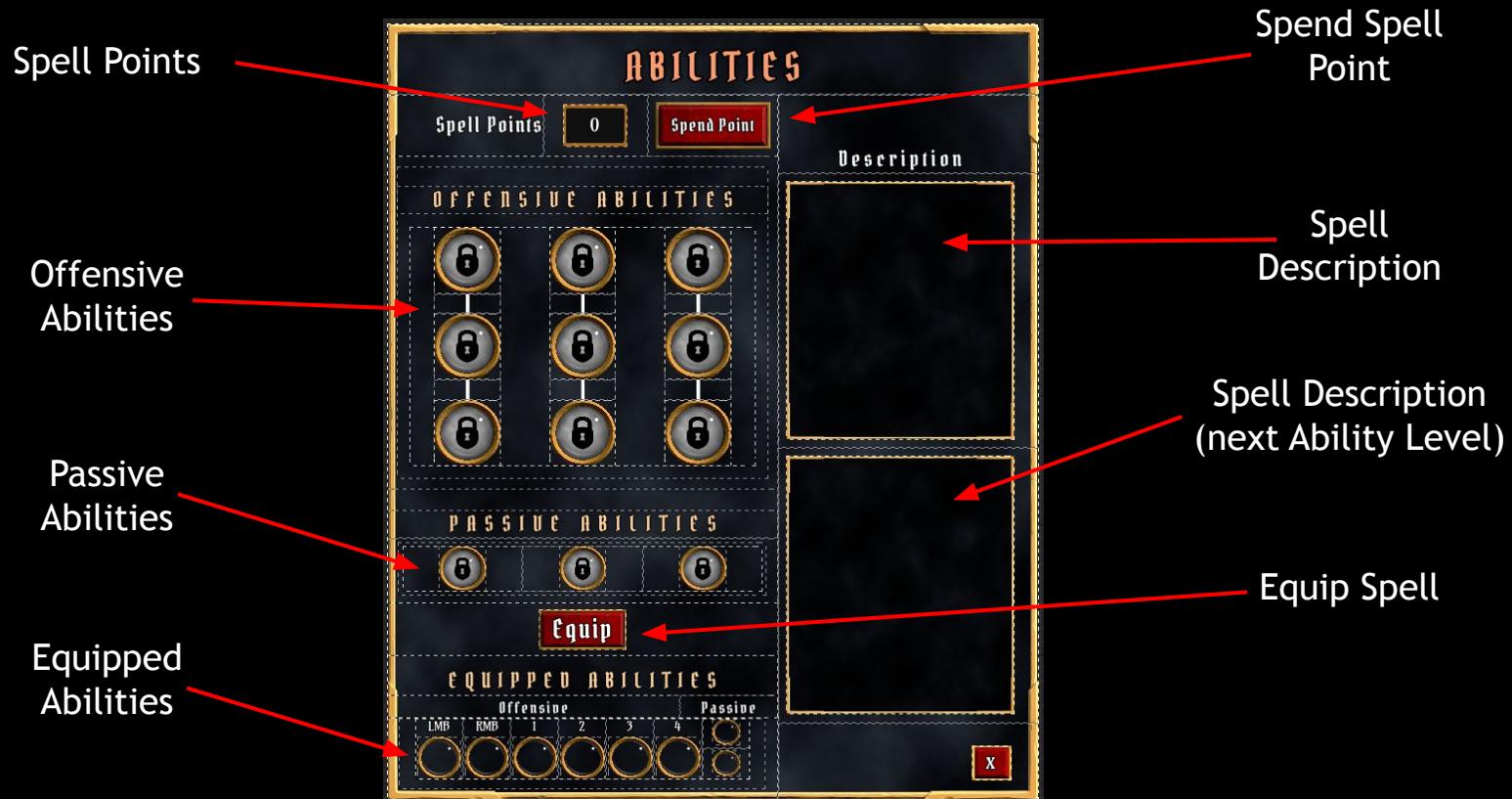


# Awarding XP

## Next Steps

1. XP Reward on Enemies (and a way to get it)
2. Incoming XP Meta Attribute
3. Passive Gameplay Ability, GA\_ListenForEvents (and grant it)
  - a. Gameplay Effect to apply in response to Event
4. Award XP from Attribute Set when damage is fatal
5. Handle Incoming XP in Attribute Set and increase XP on Player State

# Spell Menu



# Spell Menu

Offensive Spell Tree



# Spell Menu

Passive Spell  
Tree



# Ability Status

## Ability Status

Locked



Eligible



Unlocked



Equipped

## Ability Status Tag

Abilities.Status.Locked

Abilities.Status.Eligible

Abilities.Status.Unlocked

Abilities.Status.Equipped

## Ability Type Tag

Abilities.Type.Offensive

Abilities.Type.Passive

Abilities.Type.None

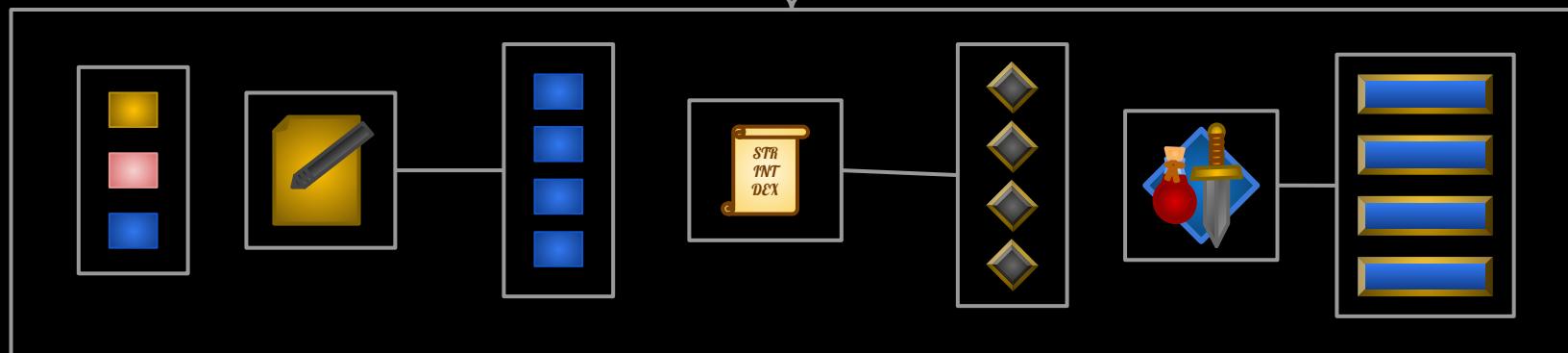


# Saving Progress

- Simple
- Doesn't require external database
- Data is local



- Central Location
- Encrypted
- Dedicated Servers
- Authority
- Requires APIs
- Database knowledge



# Model-View-ViewModel

If you put ten software architects into a room and have them discuss what the Model-View-Controller pattern is, you will end up with twelve different opinions.

-Russel East

# Model-View-ViewModel

