Smart Sound Design using Modularity and Data Inheritance

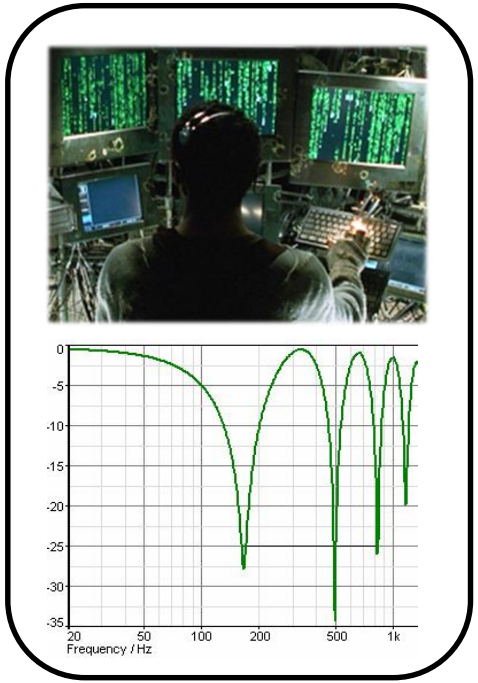
Martin Loxton
Audio Programmer, Frostbite

GAME DEVELOPERS CONFERENCE®
MOSCONE CENTER · SAN FRANCISCO, CA
MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

Overview

- Retrospective
- Modular sound design
- Palette-based sound design
- Model-based sound design
- Separating data from structure
- Applying data inheritance
- Summary
- Future developments
- Conclusion

Retrospective



Retrospective

- Our games were growing in scope
  - More cars, more guns, more creatures, more of everything!
  - Two growing problems

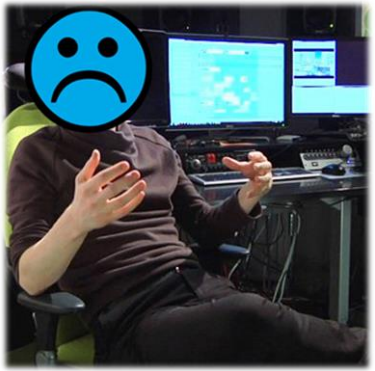
- The limits of our memory budget had been reached
  - Layers of data management were become complicated
  - Loading on demand wasn’t an option
  - Compressing the sample data wasn’t an option

Retrospective

- Creating sample data was taking too much time
  - Creating a large amount of sample data
  - Maintaining consistency in design during production

- Could no longer ship games using only this approach!

Retrospective



Use less memory

Use more CPU

Modular sound design!

Modular sound design

Breakdown

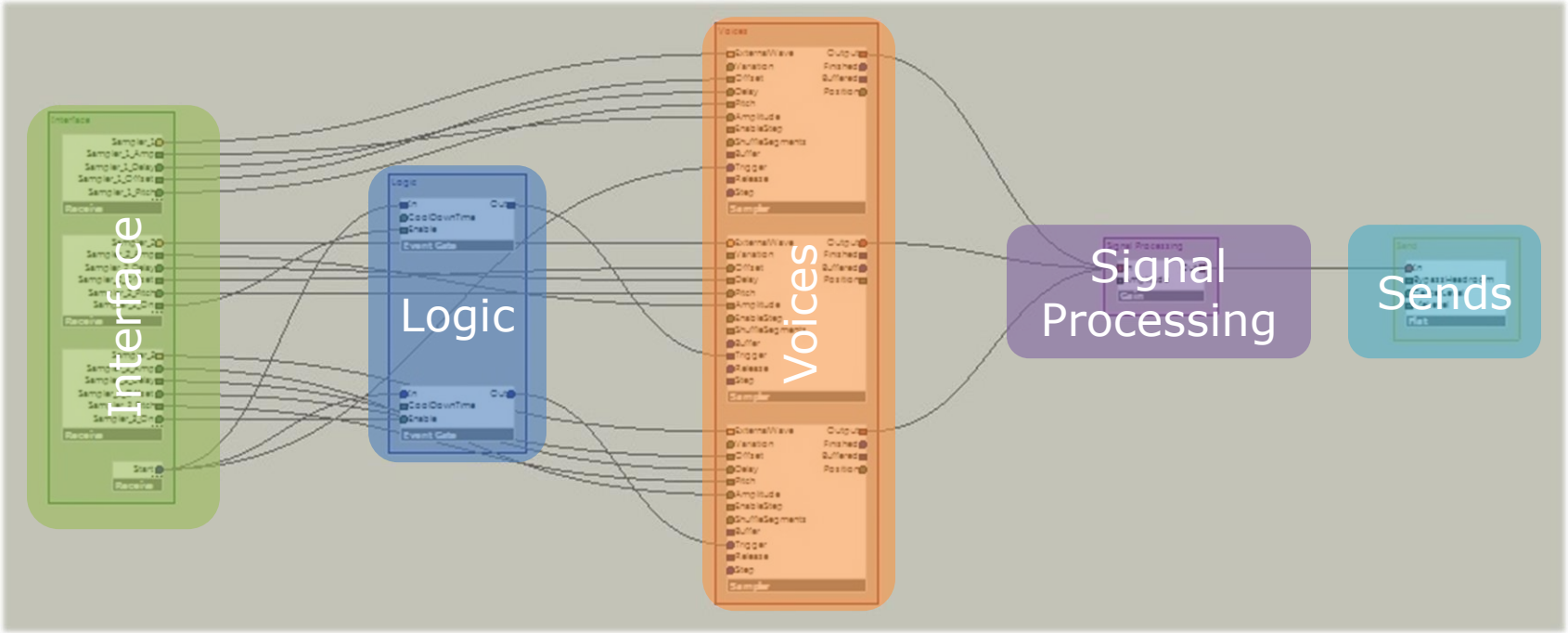
Offline

Gameplay

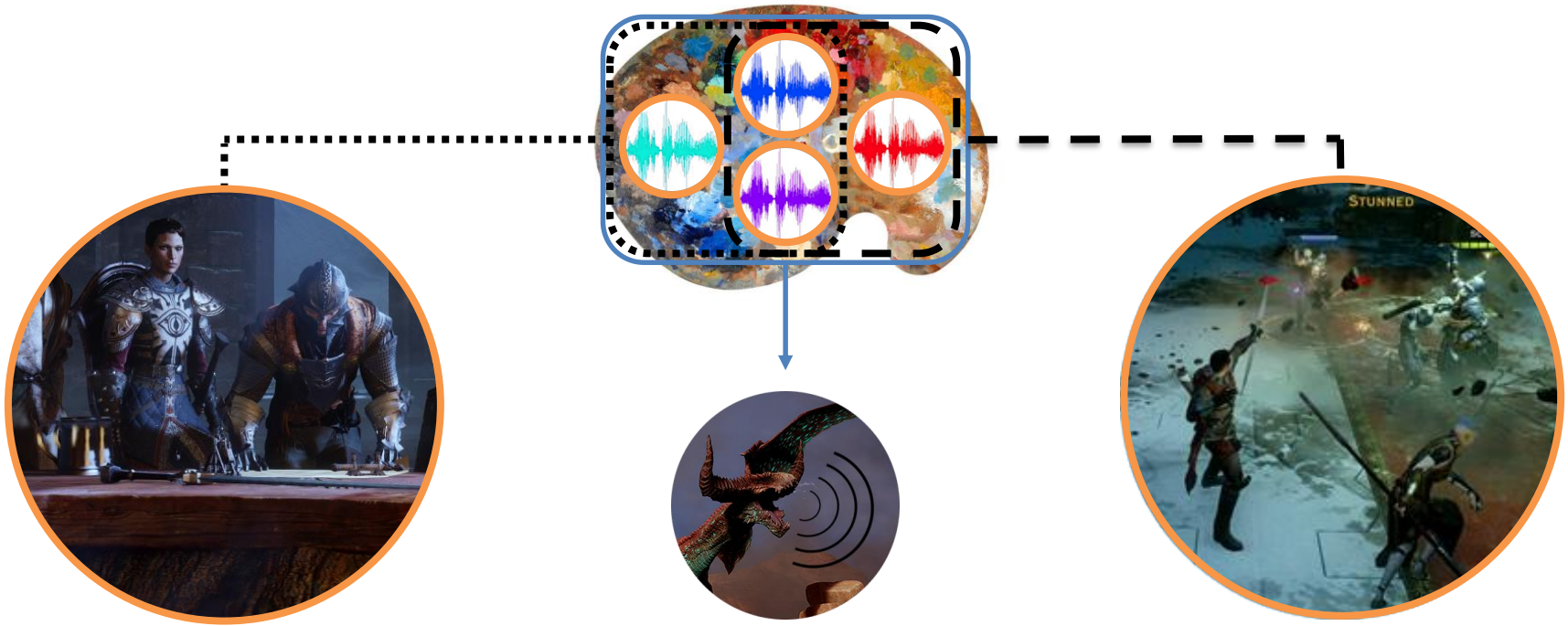
Combine

Runtime

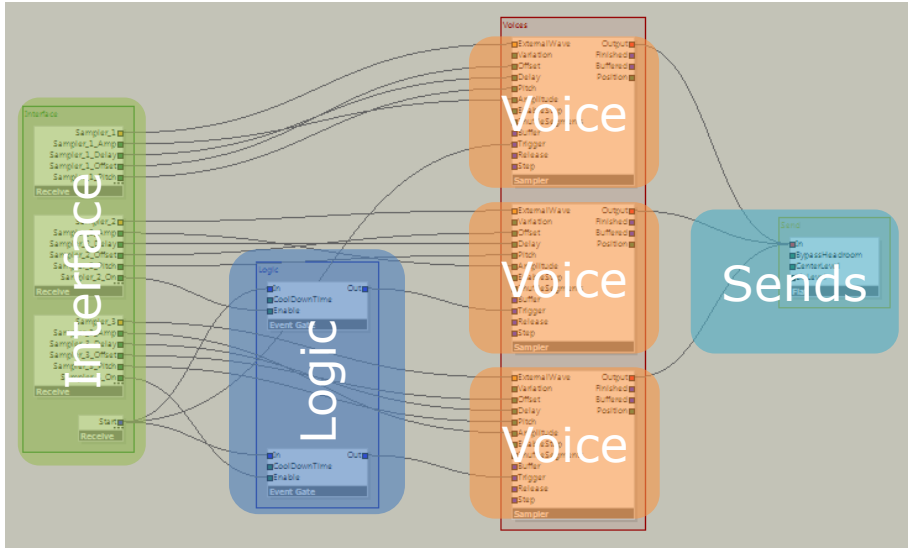
Modular sound design



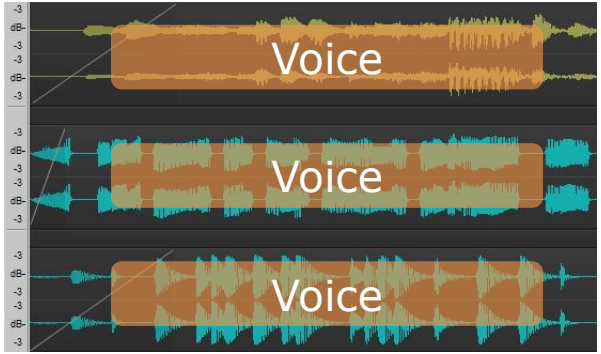
Palette-based sound design



Palette-based sound design



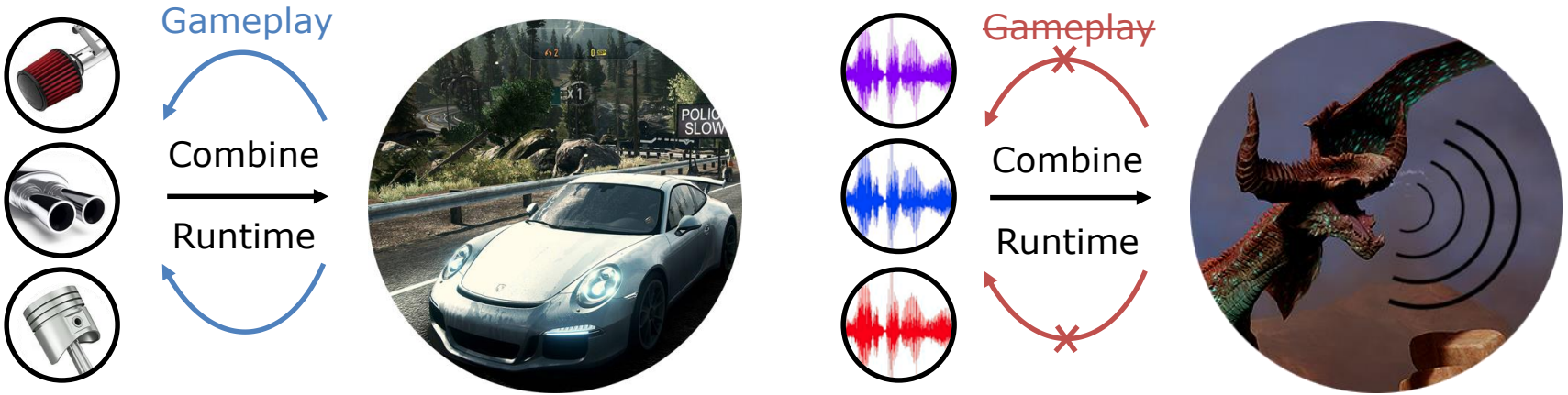
=



Palette-based sound design

- Creating new sounds is efficient
  - No longer requiring new sample data for every new sound
  - Mix and match from the sample data already in the palette
- The name of your sample data should be descriptive
- Design your palette with variety
- No longer a custom sample data for every sound
  - Expect a slightly lower quality bar when you first start with this approach
  - This will be offset as you become more proficient!
    - Allows more time to iterate further on the quality of your sounds
    - Will improve the overall quality of your game!

Model-based sound design



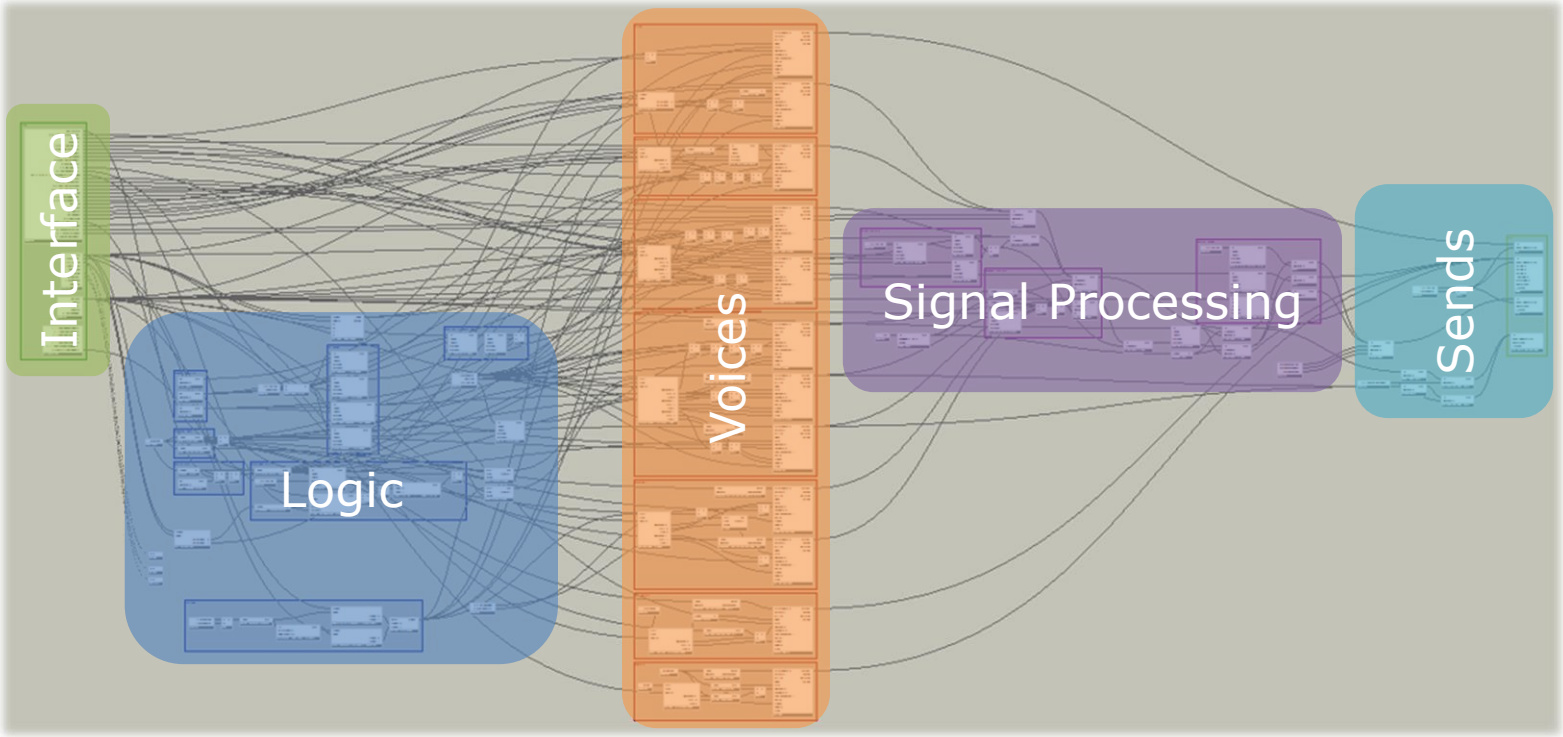
Model-based sound design

- Engine accelerating
- Engine decelerating
- Exhaust accelerating
- Exhaust decelerating
- Intake accelerating

Model-based sound design

- Core Bass
- Hi-Fi
- Noise
- Rattle

Model-based sound design



Separating data from structure

- We create model-based sounds using Sound Patches
  - Each voice is a component
  - Each component can be (optionally) controlled by logic
  - The sound is the result of combining the components together at runtime
- Separate the data from the structure!
  - The Sound Patch becomes a sound template
    - The remaining structure defines the behavior of a sound “type”
  - Different data will result in a different sound
  - A separation between the behavior and the rendered result
- Changes to the template apply to all sounds using it
  - New features
  - Bug fixes

Separating data from structure

- In what ways can the template be configured?
  - The sample data consumed by the voices
  - Node values that cannot be driven externally
  - Default interface values
- The composition of the components are dynamic
  - Driven by configured data or by gameplay data
  - The same sample data will create distinctive sounds with different compositions
  - Sample data can be shared by many sounds!
- How do we create these configurations in Frostbite?
  - Sound Patch Configurations!

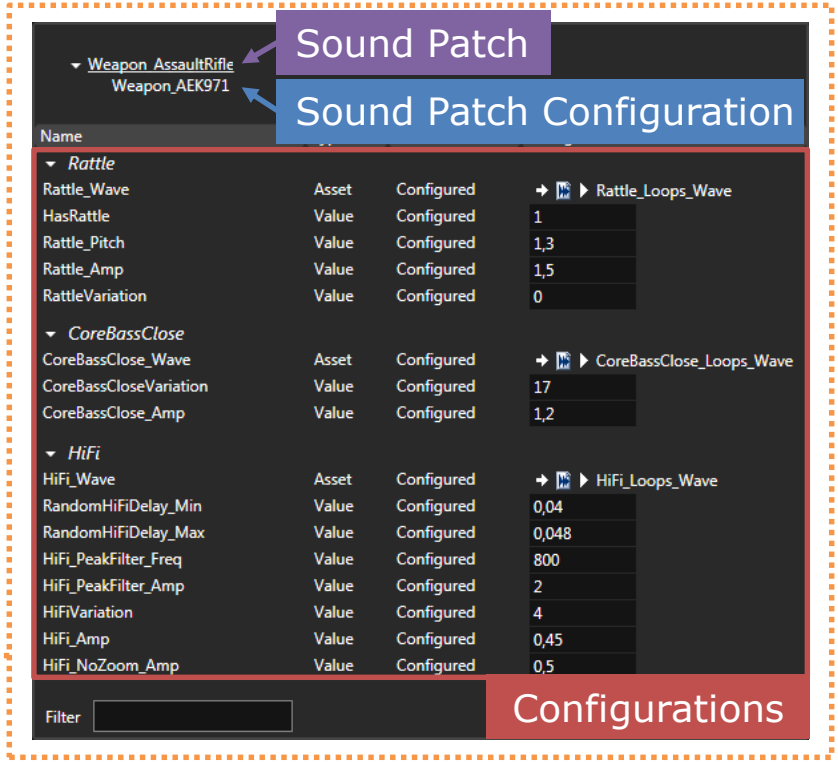
Separating data from structure

Assault Rifle	
Hi-Fi Wave	(none)
Rattle Wave	(none)
Peak Freq.	5000
Master Pitch	1.00

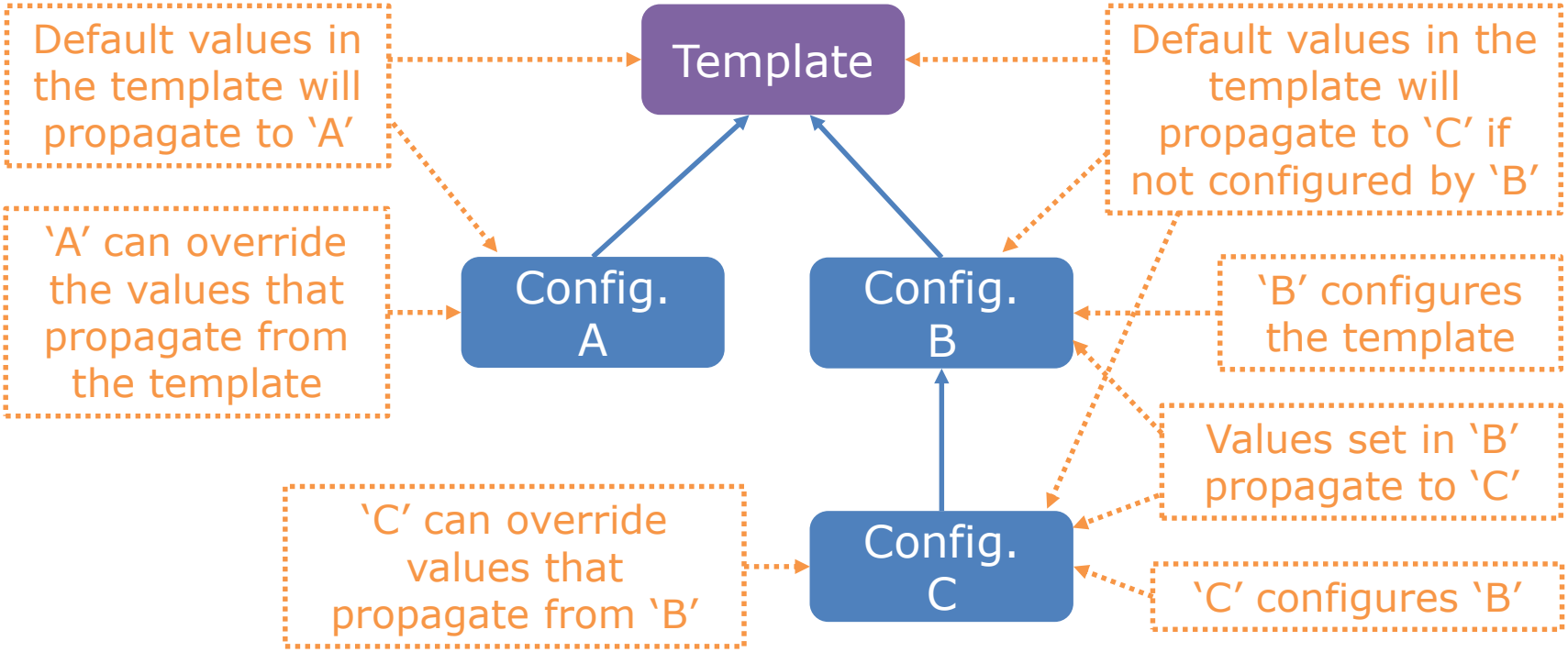
M16A4	
Hi-Fi Wave	HiFi_LoopA
Rattle Wave	Rattle_LoopB
Peak Freq.	4000
Master Pitch	1.00

AEK-971	
Hi-Fi Wave	HiFi_LoopA
Rattle Wave	Rattle_LoopA
Peak Freq.	4500
Master Pitch	1.42

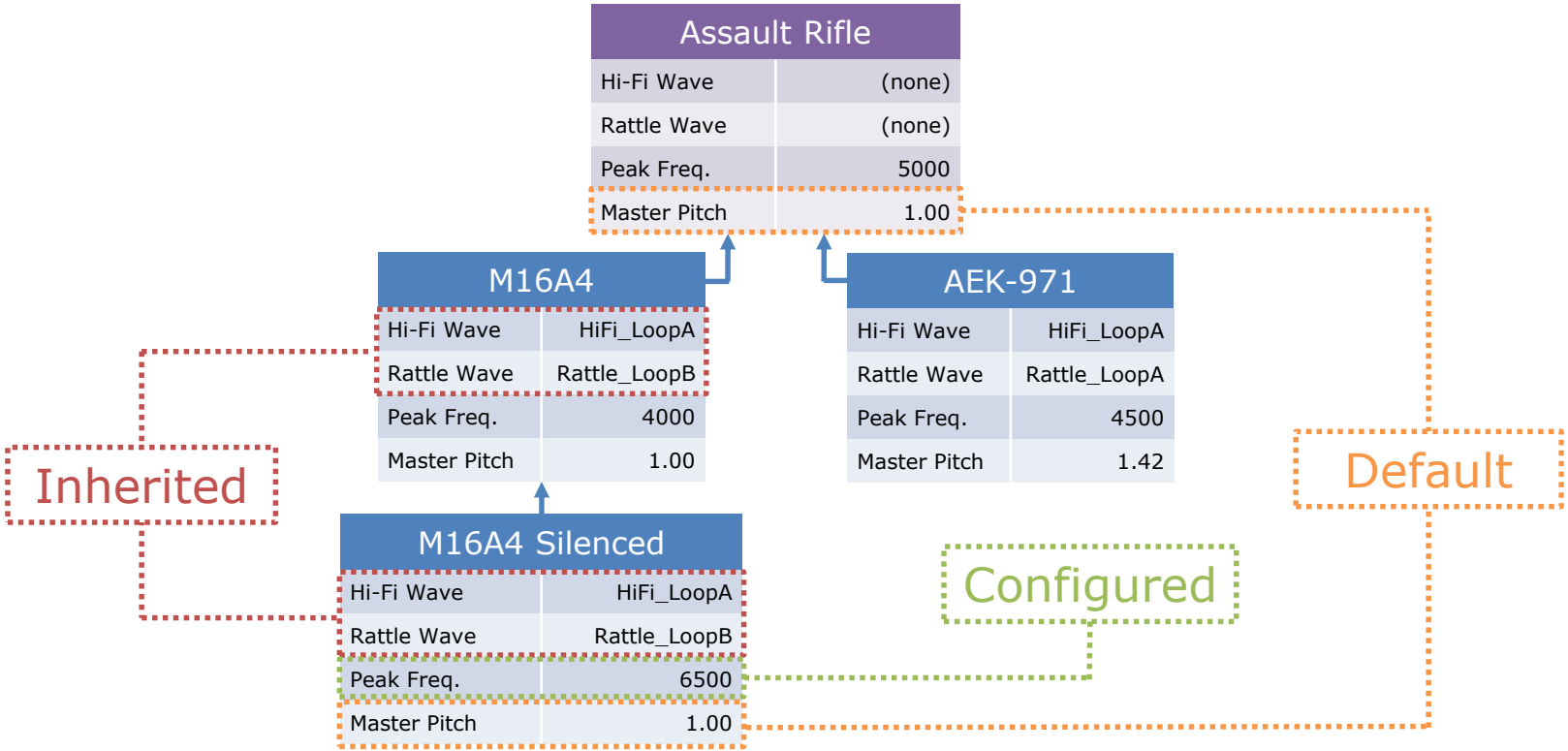
M16A4 Silenced	
Hi-Fi Wave	HiFi_LoopA
Rattle Wave	Rattle_LoopB
Peak Freq.	6500
Master Pitch	1.00



Applying data inheritance



Applying data inheritance



Applying data inheritance

- Consistency between sounds with the same template
  - The behavior of a sound “type” is defined in one place
  - Changes to the template apply to all sounds using it
    - Bug fixes
    - New features
- You can make interactive complex sound
  - Respond to the changing environment and game state
- Creating new sounds is efficient
  - Values drive the behavior
  - Existing sample data is used

Applying data inheritance

- The hierarchy organizes sounds in a logical way
  - "M16A4" is a type of "M16", which is a type of "Assault Rifle"
    - Behavior can be inferred from the hierarchy
  - Collapsed when sounds are built to reduce runtime cost

- Less requests made to disk to avoid repetition
  - Variation from the dynamic combination of components
  - Unique sample data isn't necessarily required
    - The behaviors of the components provides the variety
  - Improve streaming performance for other game content
    - Meshes
    - Textures

Applying data inheritance

- The initial investment of creating models
  - It can take a long time to get the desired behavior
  - Sometimes one model is no longer enough
    - Breaking one model into two models is challenging!
  - Requires a technical understanding of the object you're modelling
    - What should the components be?
  - Is it worth it?

- There is a performance cost
  - More CPU time is required to drive the model
  - More sample data requires decoding
  - Can be executed in parallel

Summary

- Store less unique sample data in memory
- Low latency even with a large variety of sounds
  - Shared sample data stays in memory
- Creating new sounds is efficient
  - Just combine existing data in new and exciting ways
  - Invaluable as the scope of your game grows
  - Components in a model often share the same sample data
  - Mix and match from the sample data already in the palette
  - New sounds often require only tweaks to existing sounds
- Great for DLC!

Summary

- Top 5 categories of usage

Dragon Age: Inquisition		
Category	Sounds	Templates
GUI	257	2
Cinematic	1841	13
Exertions	111	4
Level	2250	19
Impacts	1009	3

Battlefield 4		
Category	Sounds	Templates
GUI	55	4
Weapons	345	36
Vehicles	52	12
Destruction	55	3
Level	963	60

Need For Speed: Rivals		
Category	Sounds	Templates
Engine	220	10
Wheels	25	6
Collisions	99	14
GUI	120	11
Narrative	126	1

336:1

Future developments

- Was more valuable then predicted
  - Started as a relatively minor addition to the tool set
  - Now one of the most critical audio workflows we have
  - Requires more polish
- More data could be exposed as configurable
- Improved hierarchy editing
  - Support side by side comparisons of configurations
  - Visualization of entire configuration hierarchy
  - Value combination, not just override

Conclusion

- Acknowledgements

Jeremie Voillot Audio Director	Mathias Grunwaldt Audio Director	Andreas Almström Lead Sound Designer
BioWare	Ghost Games	DICE

Ben Minto Audio Director	Anders Clerwall Technical Director
DICE	Frostbite

Conclusion

- Questions?

martin.loxton@frostbite.com