Streaming the world of Horizon

Decima Asset Streaming System

Introduction

Killzone Loading System

- Traditional **level**- and **section-based** loading
- **Loading screen** while loading initial sections assets
- **Corridor sections** to unload old/load new sections
- Corridors were mostly one-way
- Could not load content around player dynamically
- File packing caused **long iteration times** for artists

Horizon Streaming System Design Goals

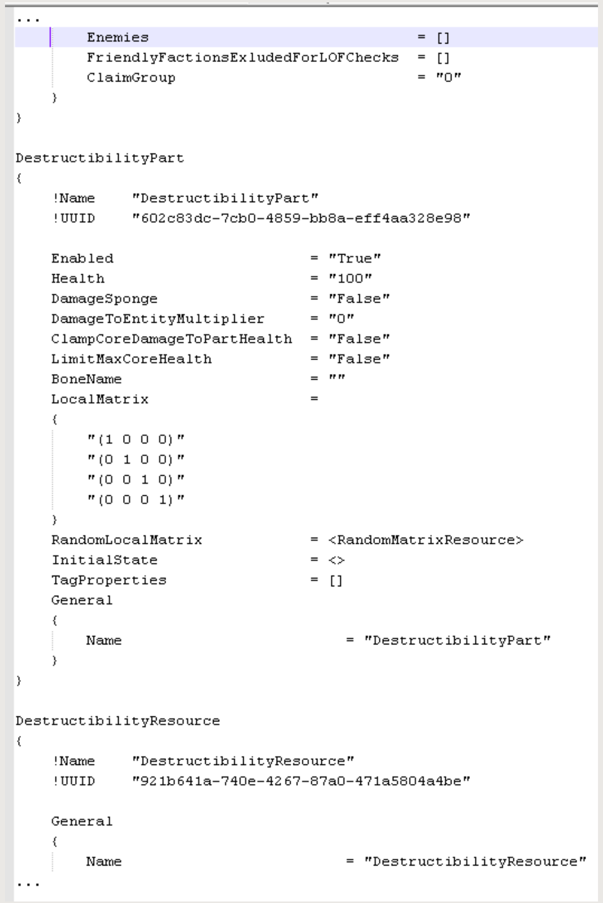
- **No loading screens** except for startup and fast travel
- **No corridors**, content should stream organically
- **Continuous loading** of content around player
- **Faster iteration** by not packing data

Decima Asset Structure

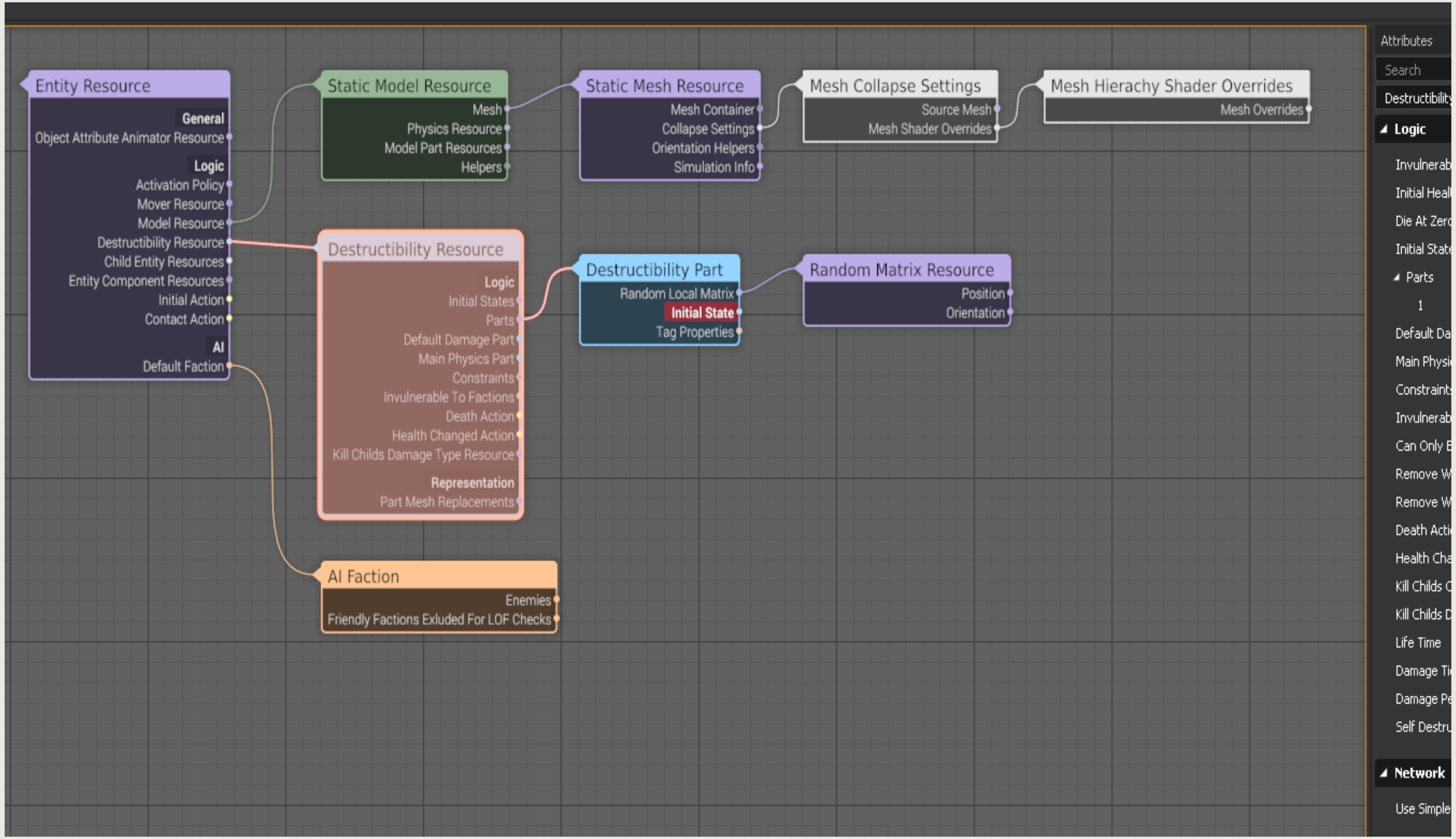
CoreText format

- All objects defined in **custom text format**
- Generated with **in-house editor, Maya**
- Object types map to C++ classes
- Objects have **attributes** and **links** to other objects
- Horizon content:
  - **300,000 files**,
  - **16 million objects**
  - **20 million links**

The CoreText format



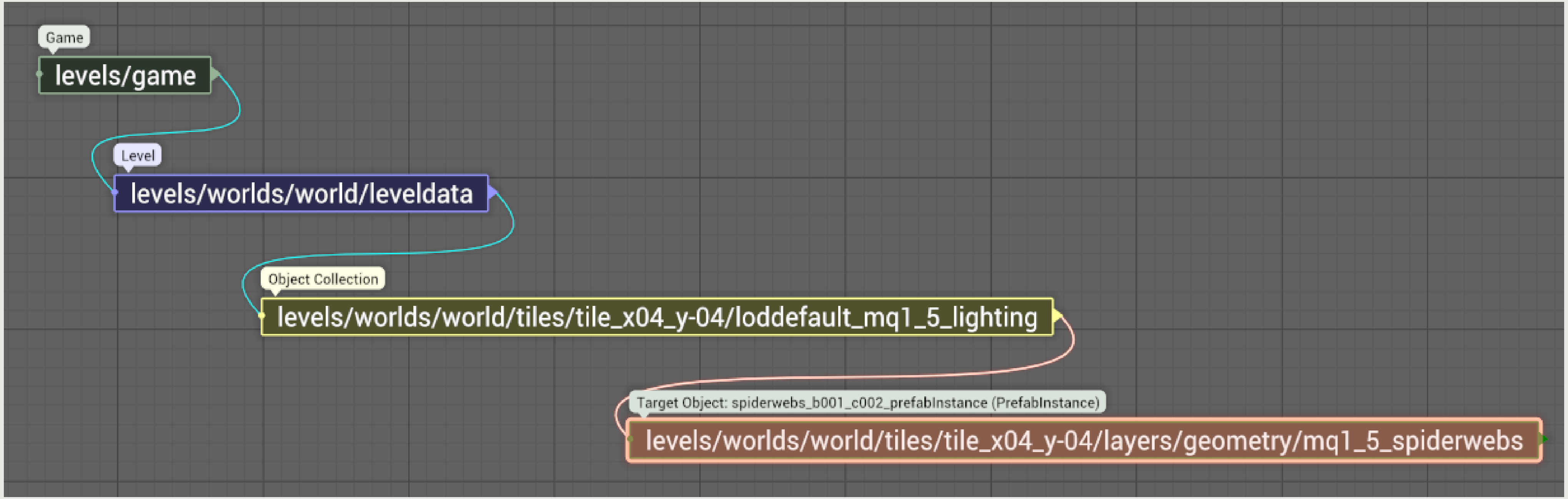
```
...
        Enemies                              = []
        FriendlyFactionsExludedForLOFChecks  = []
        ClaimGroup                           = "0"
    }
}

DestructibilityPart
{
    !Name    "DestructibilityPart"
    !UUID    "602c83dc-7cb0-4859-bb8a-eff4aa328e98"

    Enabled                          = "True"
    Health                           = "100"
    DamageSponge                     = "False"
    DamageToEntityMultiplier         = "0"
    ClampCoreDamageToPartHealth      = "False"
    LimitMaxCoreHealth               = "False"
    BoneName                         = ""
    LocalMatrix                      =
    {
        "(1 0 0 0)"
        "(0 1 0 0)"
        "(0 0 1 0)"
        "(0 0 0 1)"
    }
    RandomLocalMatrix                = <RandomMatrixResource>
    InitialState                     = <>
    TagProperties                    = []
    General
    {
        Name                             = "DestructibilityPart"
    }
}

DestructibilityResource
{
    !Name    "DestructibilityResource"
    !UUID    "921b641a-740e-4267-87a0-471a5804a4be"

    General
    {
        Name                             = "DestructibilityResource"
...
```

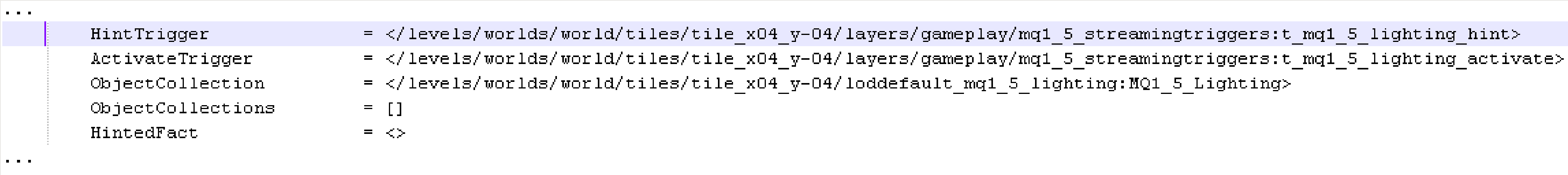


Content graphs

- Any set of **linked files** represent a **graph**
- There are **no cycles** in the content graph
- Graphs must always be loaded fully
- Many **subgraphs partially overlap**
- Subgraphs are separated by **Streaming Links**

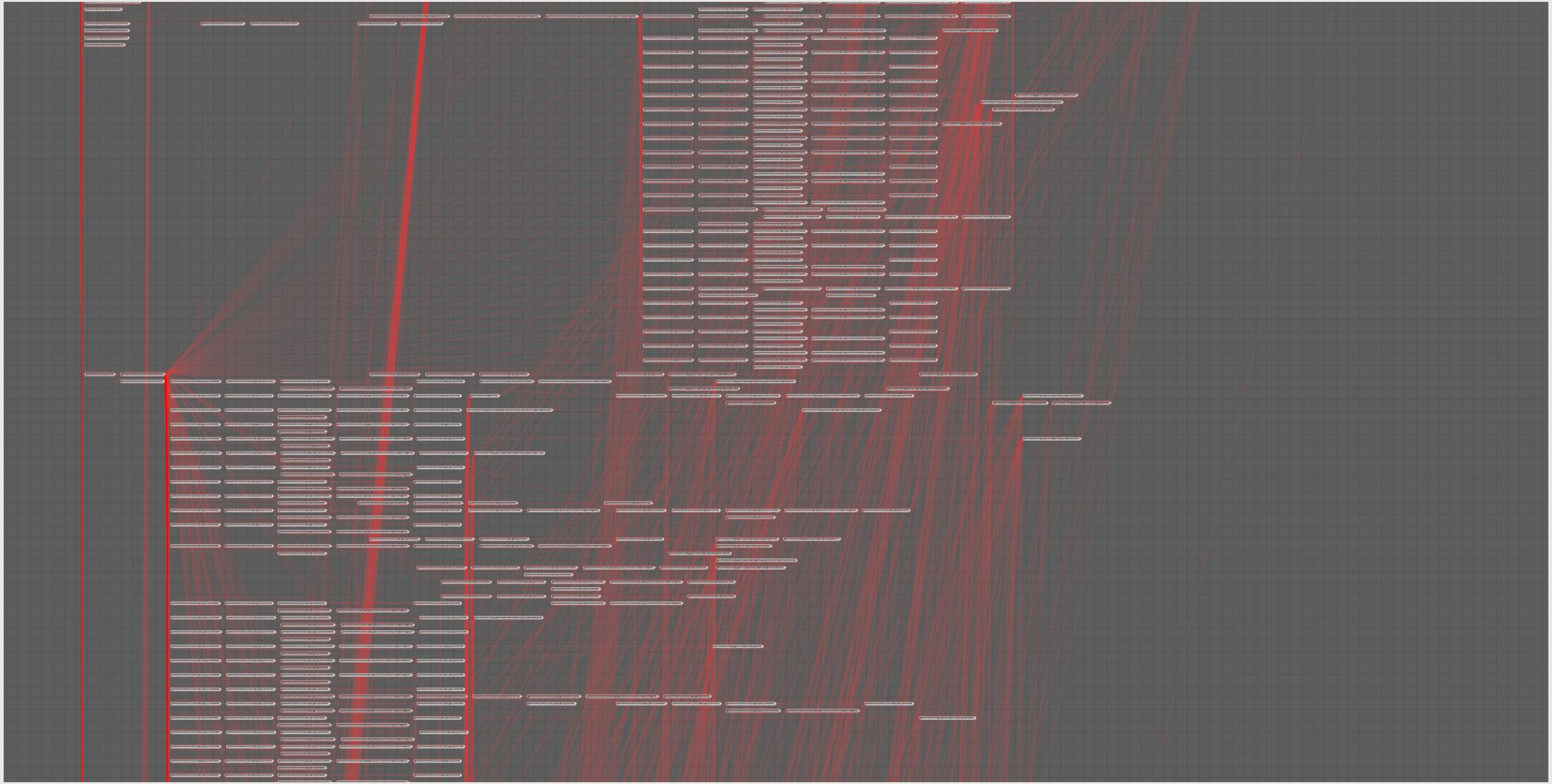
Content Graphs


Game
levels/game
Level
levels/worlds/world/leveldata
Object Collection
levels/worlds/world/tiles/tile_x04_y-04/loddefault_mq1_5_lighting
Target Object: spiderwebs_b001_c002_prefabInstance (PrefabInstance)
levels/worlds/world/tiles/tile_x04_y-04/layers/geometry/mq1_5_spiderwebs



```
...
        HintTrigger              = </levels/worlds/world/tiles/tile_x04_y-04/layers/gameplay/mq1_5_streamingtriggers:t_mq1_5_lighting_hint>
        ActivateTrigger          = </levels/worlds/world/tiles/tile_x04_y-04/layers/gameplay/mq1_5_streamingtriggers:t_mq1_5_lighting_activate>
        ObjectCollection         = </levels/worlds/world/tiles/tile_x04_y-04/loddefault_mq1_5_lighting:MQ1_5_Lighting>
        ObjectCollections        = []
        HintedFact               = <>
...
```

Content Graphs

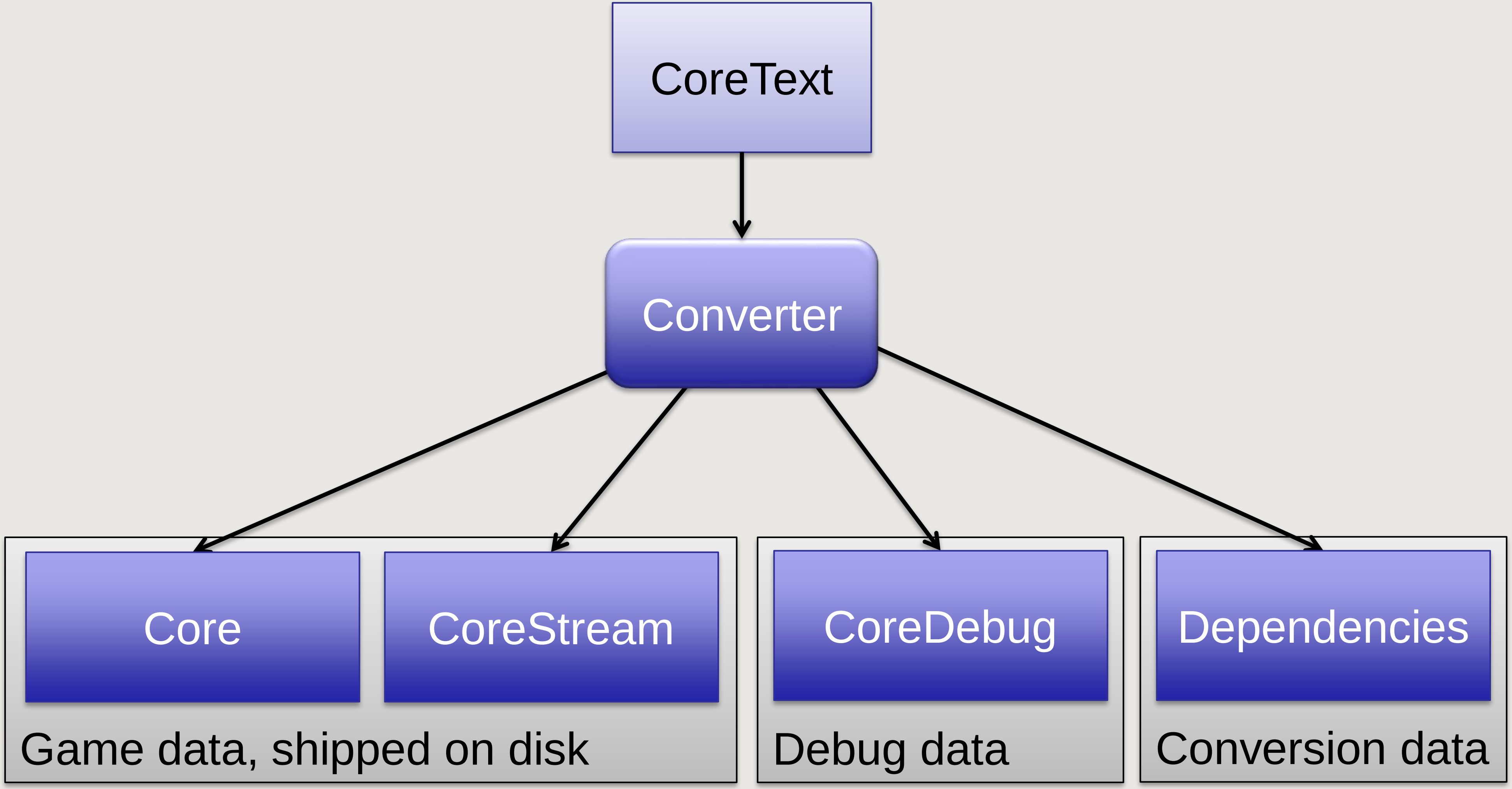


Conversion

Conversion System

- Process content graph **recursively**
- **Translate** each CoreText file into **binary data**
- **Optimize** content for runtime usage
- Generate **loading hints**, runtime **code libraries**

Asset Conversion



Post-Conversion

- For Horizon, **300,000 CoreText files** generated
  - **189,000 Core files**, 20GiB
  - **30,000 CoreStream files**, 15GiB
- **1.2GiB of localized data** per language

File Loading

Overview

- To **load object**, its **file** must be loaded
- The objects in that file often **link to other objects**
- Loading and initializing object graphs is **depth-first**
- **Any object** (and any file) is only **in memory once**

Core File Loading

Objects go through many phases during loading:

- **Deserialization** – create object and read attributes
- **Link resolving** – set pointers to linked objects
- **Initialization** – allow the object to execute init code
- **Activation** – add to world, physics, other systems

Load ordering

- **Ordering** is important:
  - Any objects pointed to must always be initialized first
  - Processing is **depth-first** in graph order
- Full graph must be known when loading assets

Loading Process

- **Determine file graph**
- **Remove** files already loaded
- **Queue** remaining files for async I/O (depth first)
- Create **job graph** for object initialization (depth-first)
- **Deserialize files** into objects
- **Run initialization** jobs for completed files

Reference counting Core Files

- Files are **reference counted**
- When loading a subgraph, **skip loaded files**
- Instead, **take reference** to loaded files
- Files are **unloaded automatically** on last release
- No object is ever loaded more than once

Reference Counting Files

- Let’s start loading a character:

NPC_Blond

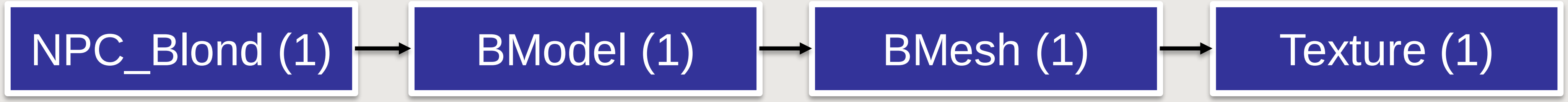
- This character needs these files:

NPC_Blond HeadModel HeadMesh HeadTexture

- No files are loaded at this point

Reference Counting Files

- We load these files and get this graph:



- Each file currently has a reference count of 1

Reference Counting Files

- We wish to load a second character:

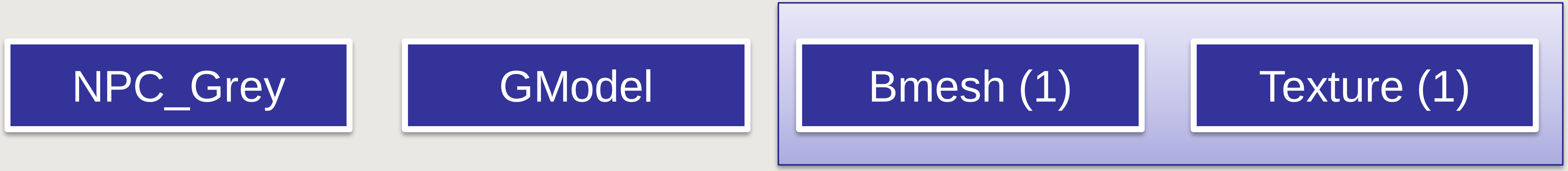
NPC_Grey

- This character consists of these parts:

NPC_Grey GModel BMesh Texture

Reference Counting Files

- We know that two of these files are already loaded:



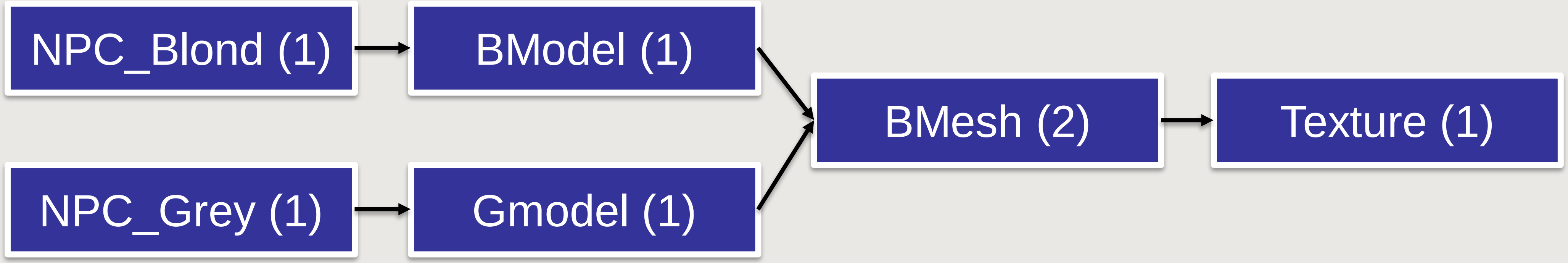
- So we load only these files:

NPC_Grey

GModel

Reference Counting Files

- This leads to the new graph:



NPC_Blond (1)

BModel (1)

NPC_Grey (1)

Gmodel (1)

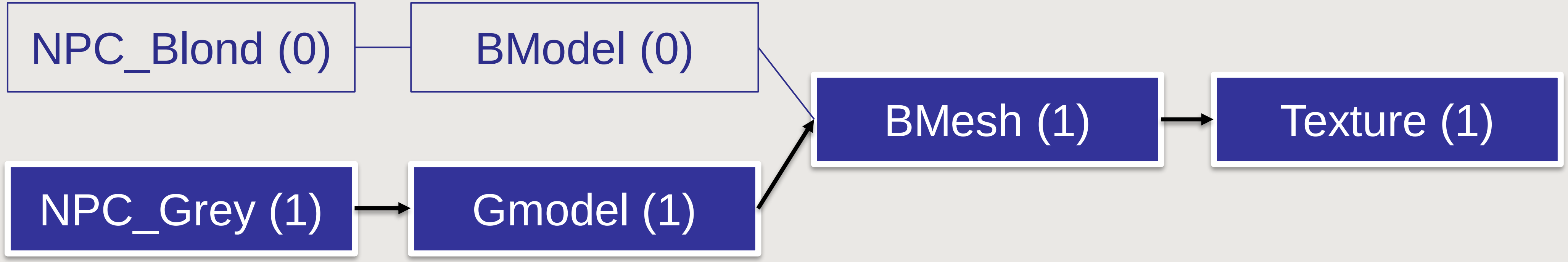
BMesh (2)

Texture (1)

- Now BMesh has two references, and is shared

Reference Counting Files

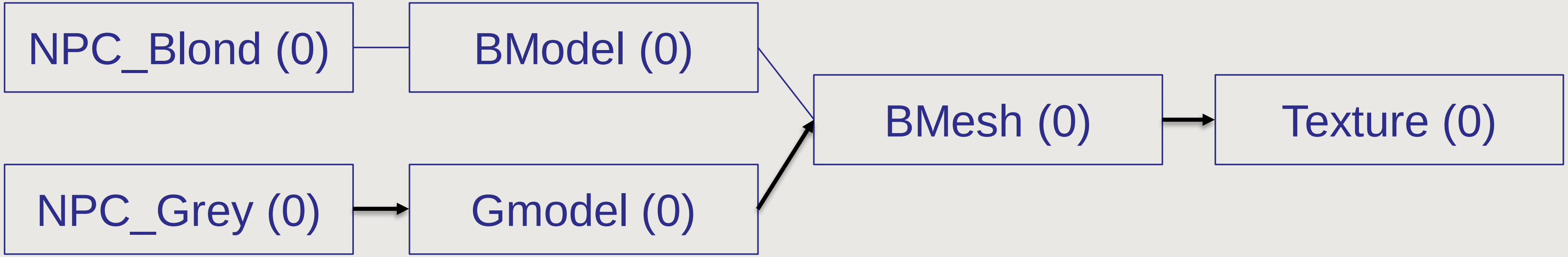
- We unload the first character:



- BMesh now has one reference

Reference Counting Files

- We unload the second character:



- All files have reference count of zero and are unloaded

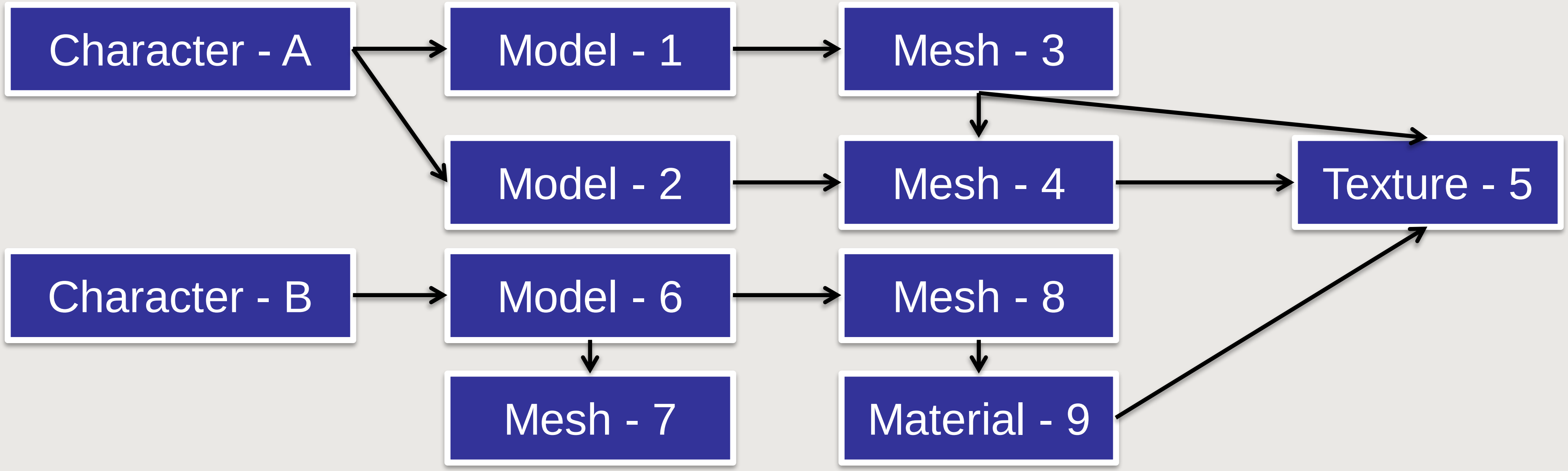
Prefetching

Prefetch Files

- Represent the **file hierarchy** of the entire game
- Generated during conversion
- Simple to **determine file graph** for any given file
- Very **little data**, ~20MiB on disk/in memory

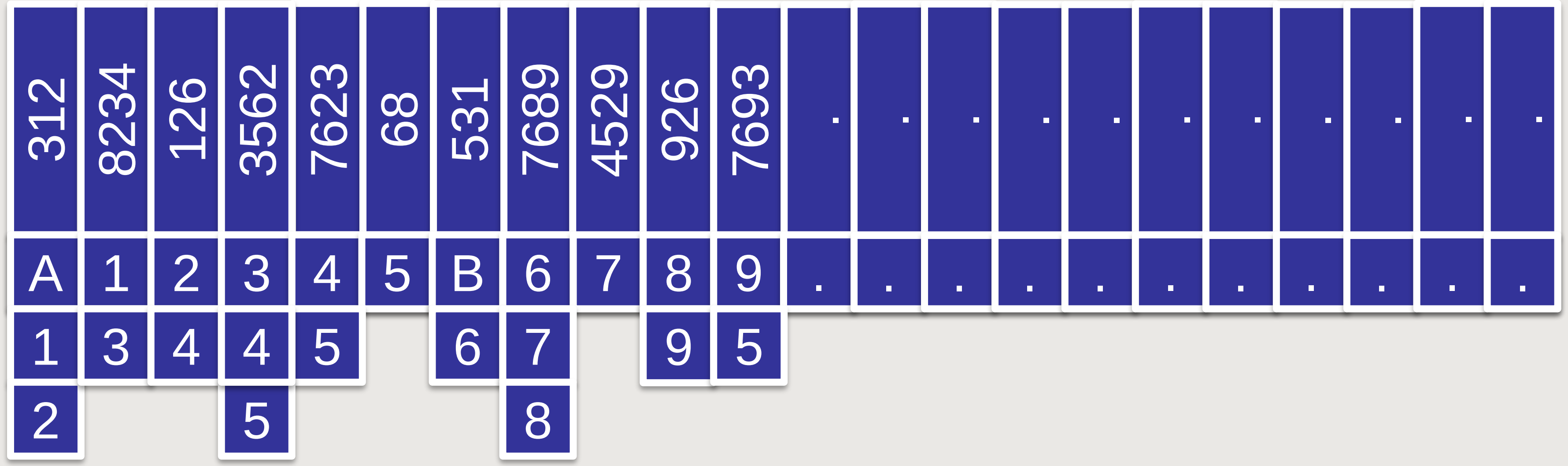
Prefetch Files

Assume we’re working with these file graphs:



Prefetch File

This is the Prefetch list for these files:



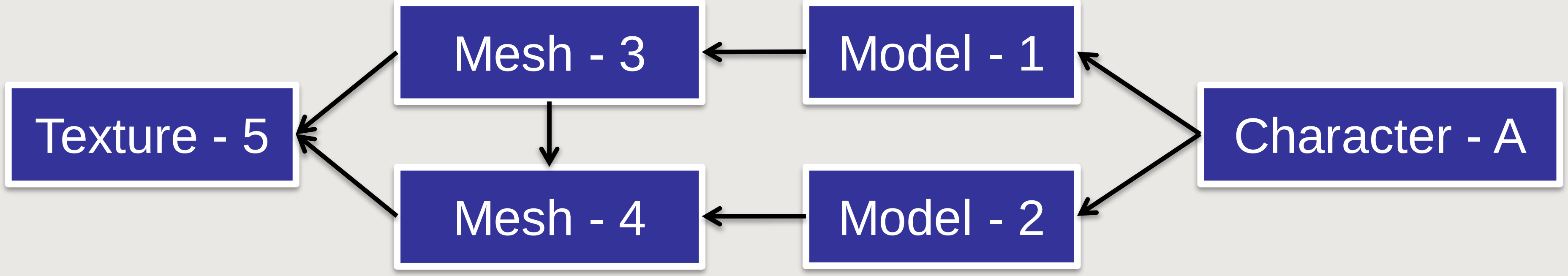
312	8234	126	3562	7623	68	531	7689	4529	926	7693	.	.	.	.	.	.	.	.	.	.	.
A	1	2	3	4	5	B	6	7	8	9	.	.	.	.	.	.	.	.	.	.	.
1	3	4	4	5		6	7		9	5											
2			5				8														

Graph A

- When traversing for A, we get sequence:

5	4	3	2	1	A

- Which corresponds with the depth-first graph:

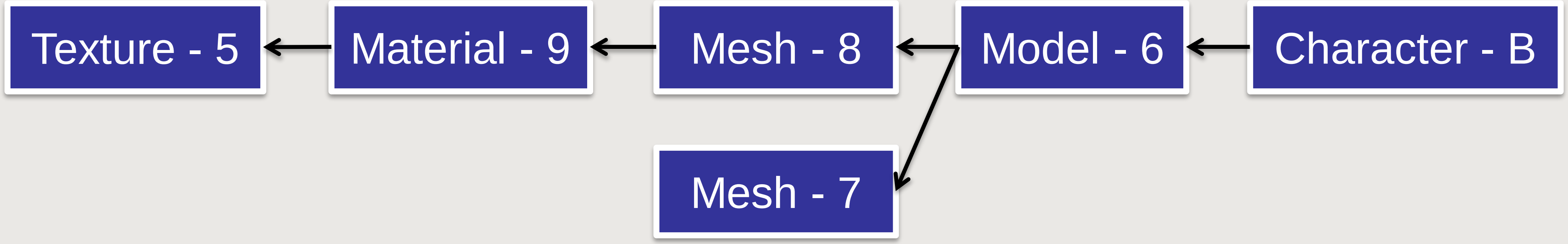


Graph B

- When traversing for B, we get sequence:

5	9	7	8	6	B

- Which corresponds with the depth-first graph:



Streaming Strategies

Streaming Links

- **Roots of subgraphs**
  - **Tiles, Characters, Weapons**
- **Loaded on demand**
- Are loaded by Streaming Strategies
- Often overlap with other loaded graphs

Streaming Strategies

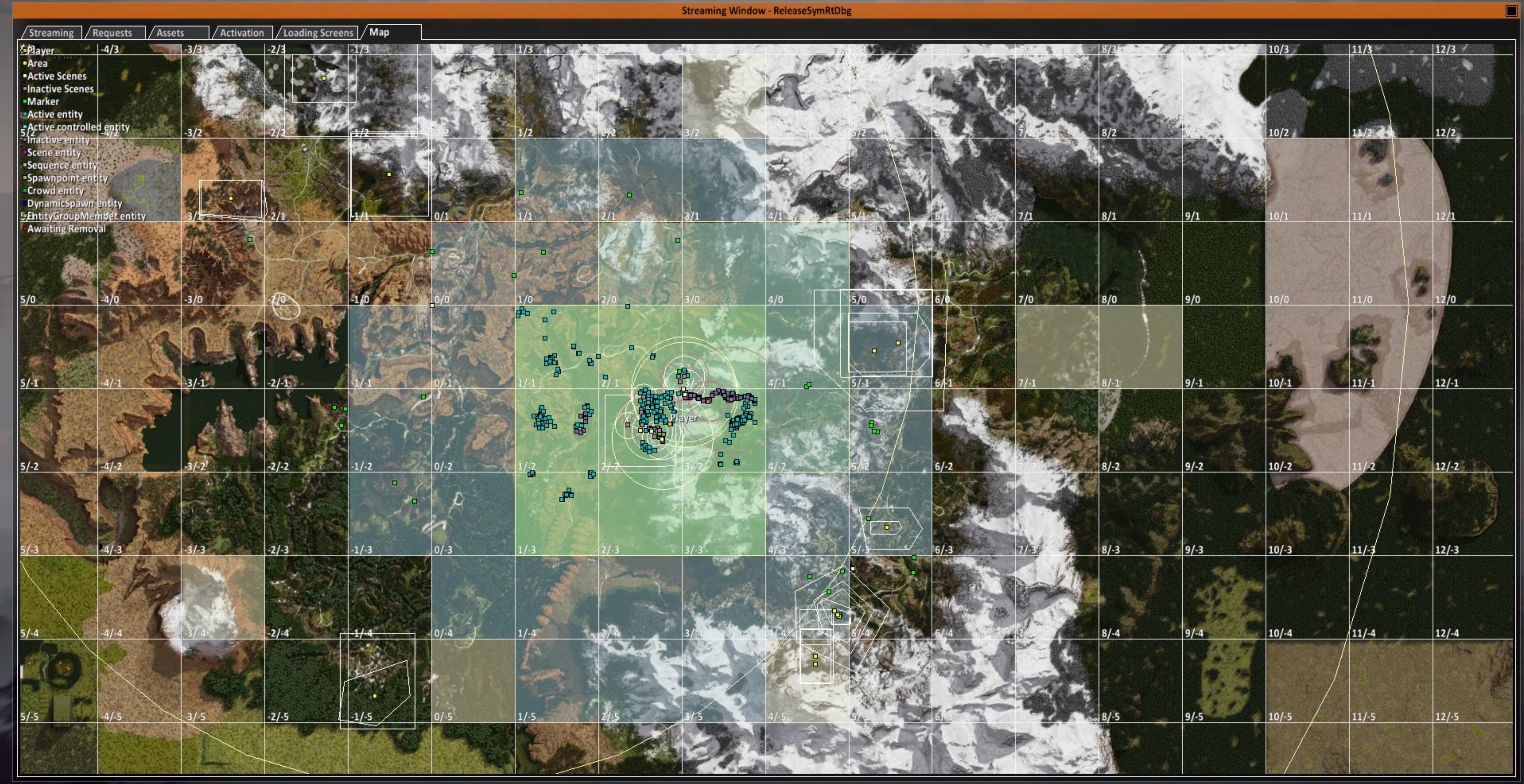
- **Intermediary** between game and streaming system
- Determine when to **load/unload** subgraphs
- **Customizable** logic for different domains
- **Evaluated once per frame** to queue load/unload

AlwaysLoaded Streaming Strategy

- Responsible for loading **initial game content**
  - **System assets**
  - **World data**
  - **Aloy**
- Loaded at startup, never unloaded

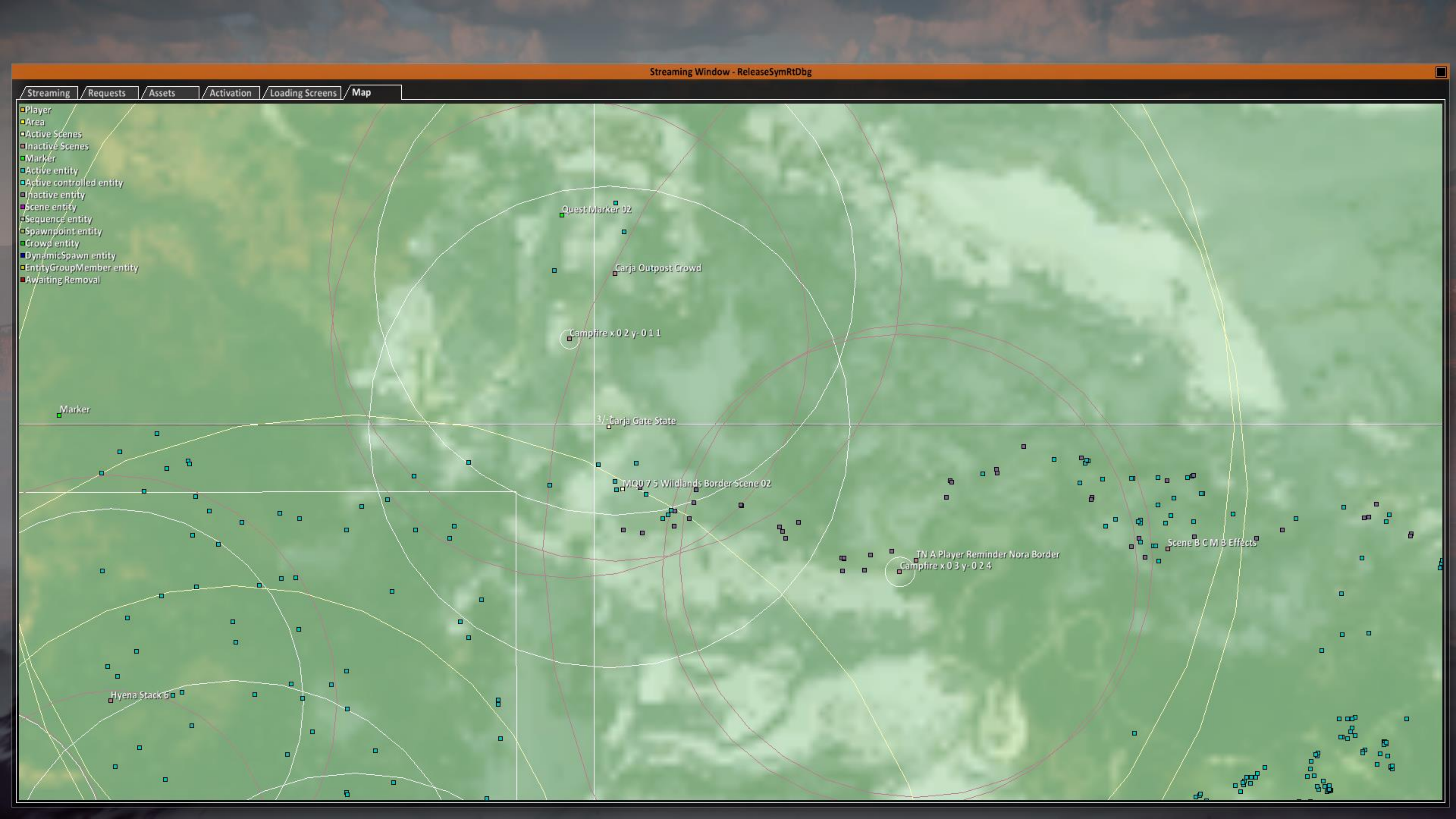
TileBased Streaming Strategy

- Loads/unloads **tiles around player**
- Four tile resolutions:
  - **9 High**, full resolution tiles
  - **9 Medium**, medium res geometry and physics mesh
  - **9 Low**, low res baked geometry
  - **12 Very low**, always loaded


Streaming Window - ReleaseSymRtDbg
Streaming
Requests
Assets
Activation
Loading Screens
Map
Player
Area
Active Scenes
Inactive Scenes
Marker
Active entity
Active controlled entity
Inactive entity
Scene entity
Sequence entity
Spawnpoint entity
Crowd entity
DynamicSpawn entity
EntityGroupMember entity
Awaiting Removal
-4/3
-3/3
-2/3
-1/3
1/3
8/3
10/3
11/3
12/3
-3/2
-2/2
1/2
3/2
8/2
10/2
11/2
12/2
-3/1
-2/1
0/1
1/1
2/1
3/1
4/1
5/1
6/1
7/1
8/1
9/1
10/1
11/1
12/1
5/0
-4/0
-3/0
-2/0
-1/0
0/0
1/0
2/0
3/0
4/0
5/0
6/0
7/0
8/0
9/0
10/0
11/0
12/0
5/-1
-4/-1
-3/-1
-2/-1
-1/-1
0/-1
1/-1
2/-1
4/-1
5/-1
6/-1
7/-1
8/-1
9/-1
10/-1
11/-1
12/-1
Player
5/-2
-4/-2
-3/-2
-2/-2
-1/-2
0/-2
1/-2
4/-2
6/-2
8/-2
9/-2
10/-2
11/-2
12/-2
5/-3
-4/-3
-3/-3
-2/-3
-1/-3
0/-3
1/-3
2/-3
3/-3
4/-3
5/-3
6/-3
7/-3
8/-3
9/-3
10/-3
11/-3
12/-3
5/-4
-4/-4
-2/-4
-1/-4
0/-4
1/-4
3/-4
6/-4
8/-4
9/-4
10/-4
11/-4
12/-4
5/-5
-4/-5
-3/-5
-2/-5
-1/-5
0/-5
1/-5
2/-5
3/-5
4/-5
8/-5
9/-5
10/-5
11/-5
12/-5

AreaBased Streaming Strategy

- **Primary hull** around player for **loading hint**
- **Secondary hull** around player for **activation hint**
- Used for **quests, scenes**, other **dynamic encounters**


Streaming Window - ReleaseSymRtDbg
Streaming
Requests
Assets
Activation
Loading Screens
Map
Player
Area
Active Scenes
Inactive Scenes
Marker
Active entity
Active controlled entity
Inactive entity
Scene entity
Sequence entity
Spawnpoint entity
Crowd entity
DynamicSpawn entity
EntityGroupMember entity
Awaiting Removal
Quest Marker 02
Carja Outpost Crowd
Campfire x 0 2 y- 0 1 1
Marker
3/ Carja Gate State
MQ0 7 5 Wildlands Border Scene 02
TN A Player Reminder Nora Border
Campfire x 0 3 y- 0 2 4
Scene B C M B Effects
Hyena Stack 6

ProgramBased Streaming Strategy

- Evaluates **custom programs** created by designers
- Used for **complex** streaming scenarios
- Uses **player and world facts** for dynamic content

Streaming Strategy Conclusions

- **Well maintainable** streaming system
- Can **easily be extended** by adding strategies
- **Designers can implement** streaming logic

Packaging

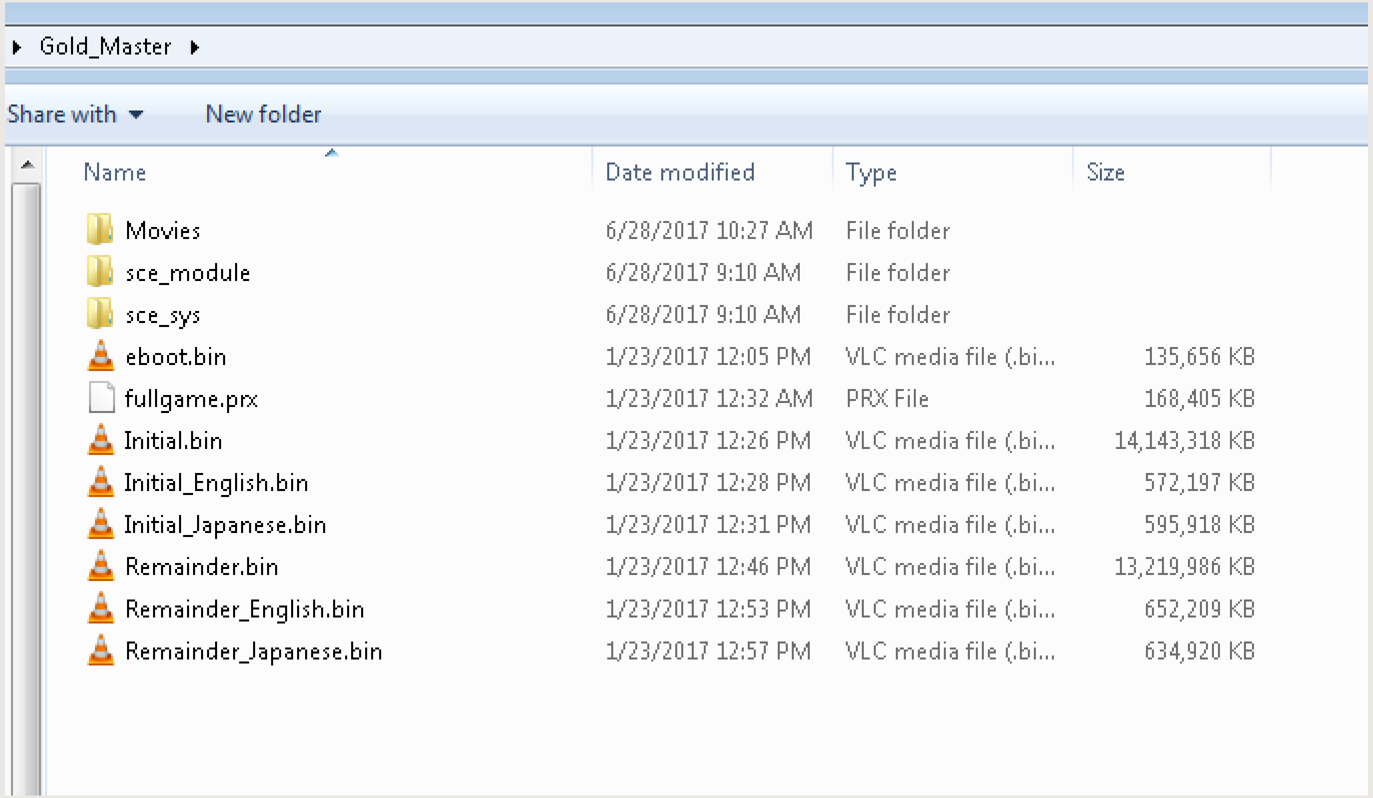
Packaging

- We had **~220,000** files for final package
- **Exceeds limit** of PS4 packages
- Opening and closing so many files is costly
- Needed **a better way** of shipping content

PackFiles

- **Recombine files** into very **small number** of large files
- **Keep files open** at all times,
- Keep **file directory in memory**
- Optimize file order for most **linear access**
- **Compress files** to fit all content on disk

Low File Count



Name	Date modified	Type	Size
Movies	6/28/2017 10:27 AM	File folder	
sce_module	6/28/2017 9:10 AM	File folder	
sce_sys	6/28/2017 9:10 AM	File folder	
eboot.bin	1/23/2017 12:05 PM	VLC media file (.bi...	135,656 KB
fullgame.prx	1/23/2017 12:32 AM	PRX File	168,405 KB
Initial.bin	1/23/2017 12:26 PM	VLC media file (.bi...	14,143,318 KB
Initial_English.bin	1/23/2017 12:28 PM	VLC media file (.bi...	572,197 KB
Initial_Japanese.bin	1/23/2017 12:31 PM	VLC media file (.bi...	595,918 KB
Remainder.bin	1/23/2017 12:46 PM	VLC media file (.bi...	13,219,986 KB
Remainder_English.bin	1/23/2017 12:53 PM	VLC media file (.bi...	652,209 KB
Remainder_Japanese.bin	1/23/2017 12:57 PM	VLC media file (.bi...	634,920 KB

Keeping Files Open

- Files open for **duration of game**
- On start, open files and **read file directories**
- In-memory file directory
- Only use **sceKernelPreadv**,
- **No calls to open, lseek, or close**
- Dramatically improves **performance**

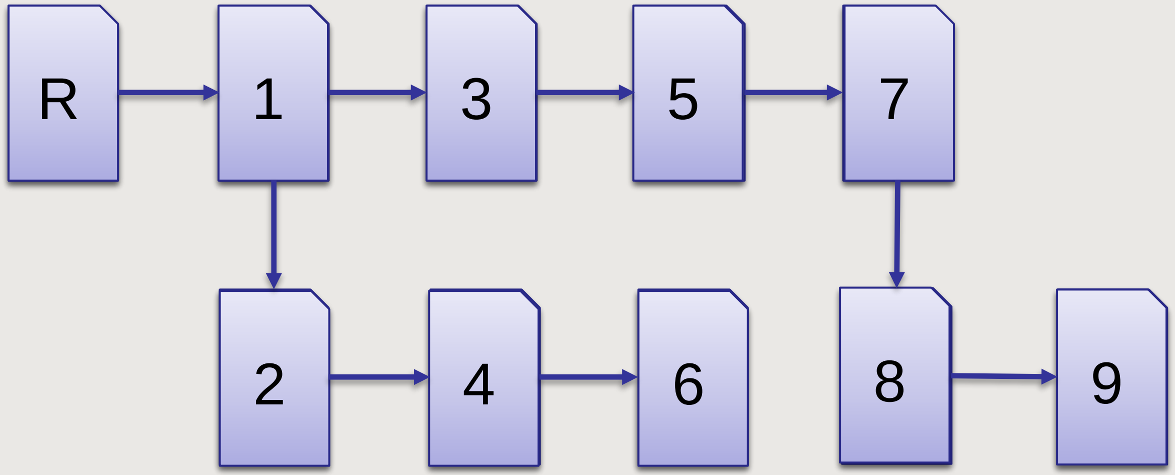
Optimized file order

- **Scan content graph** to discover all file links
- **Split files** between initial/remainder/localized groups
- Group files in **subgraphs** based on streaming links
- Order files in groups on **graph order, depth-first**

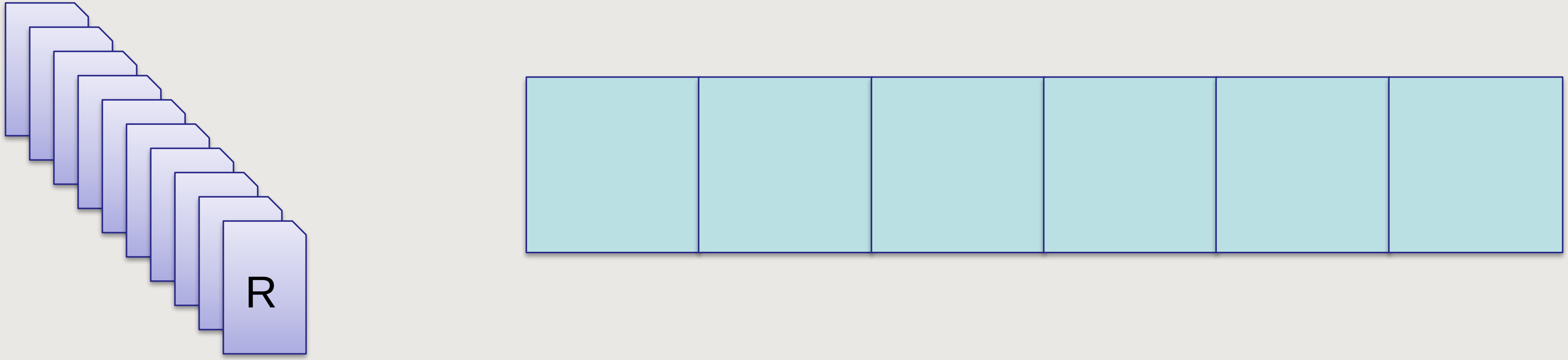
Compression

- Write sorted, uncompressed files to **256KiB blocks**
- **Compress blocks**
- Write compressed blocks **sequentially**
- Index maps from **logical** to **physical** offset

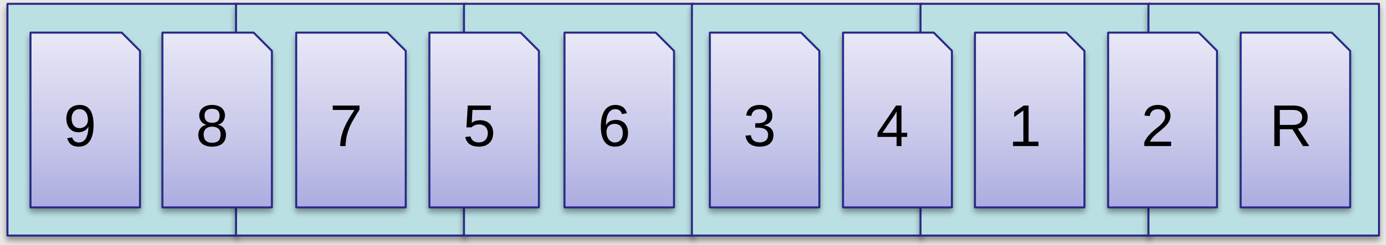
Sorting File Graphs



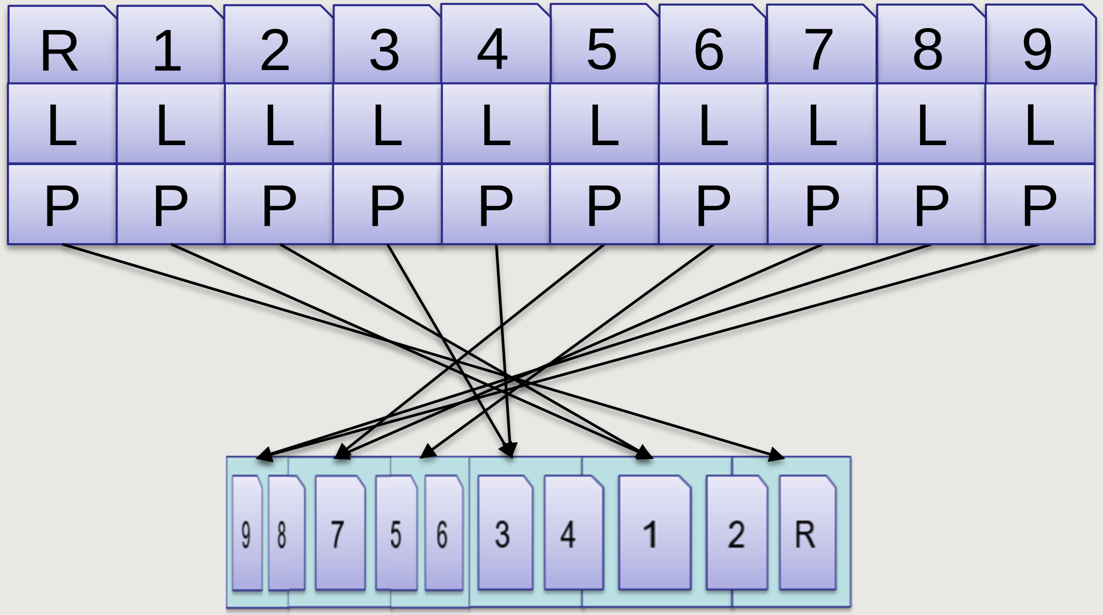
Packing Files



Compressing Blocks



Block Mapping



Patching

- A patch file is a **regular PackFile**
- Contains **added/modified files** since Gold Master
- Index is **overlay on Gold Master** file index
- Lookup **finds patched file entry**
- **Simple code**, no delta compression
- Current 1.30 patch is **only ~98MiB** of content

Development

- **No PackFiles in development**
- Instead, files are loaded from **host PC via socket**
- Host PC keeps **files in memory**
- PS4 HDD only used for testing packages

I/O Performance Results

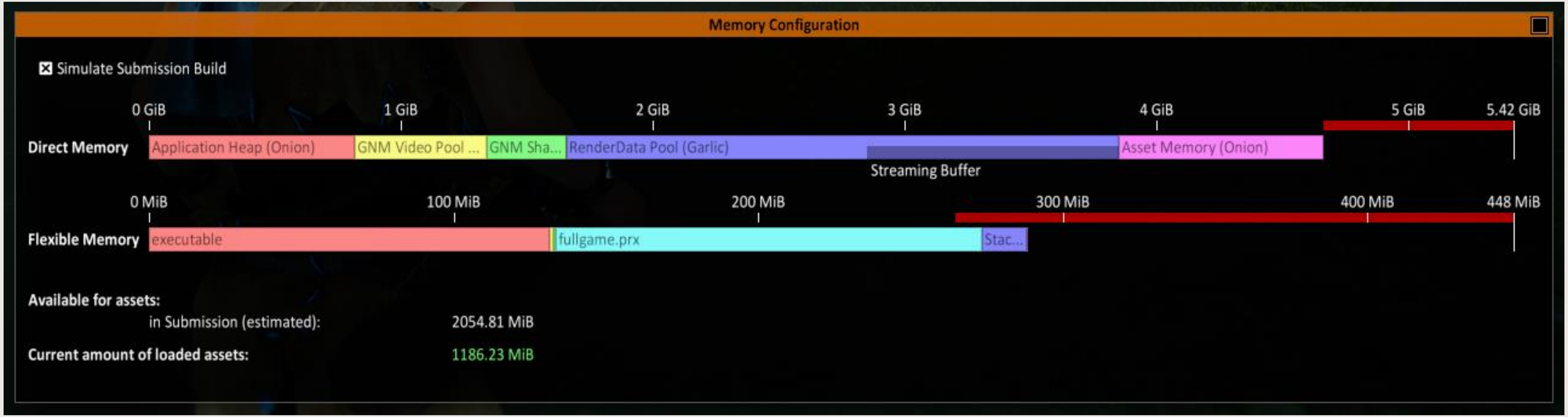
	Package File Count	Logical File Count	Read Average
Killzone Large Files	~3000	~3000	~50MiB/sec
Horizon MemCache	~200,000	~200,000	~90MiB/sec
Horizon Shipped Package	4	~200,000	~60MiB/sec

- Small files **improved iteration times** enormously
- **No need** to do any **packaging** during production
- Packing files **only for shipping** works great

Memory Management

Memory Layout

High-level view of memory layout



Memory Configuration

☒ Simulate Submission Build

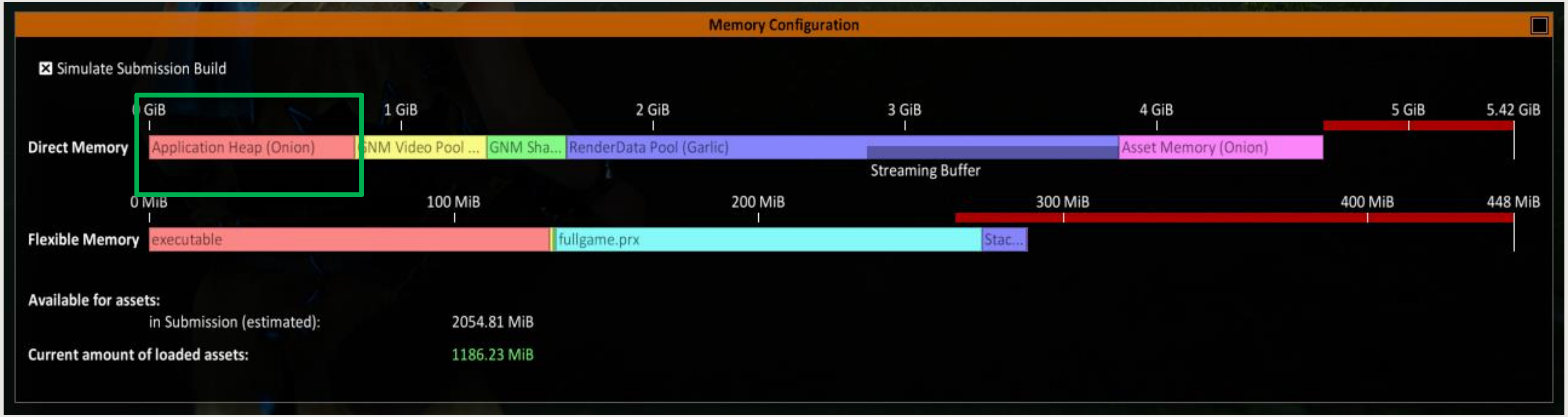
Available for assets:

in Submission (estimated): 2054.81 MiB

Current amount of loaded assets: 1186.23 MiB

Heap

- Fixed size, ~800MiB, Onion
- Managed by DLMalloc



Memory Configuration

☒ Simulate Submission Build

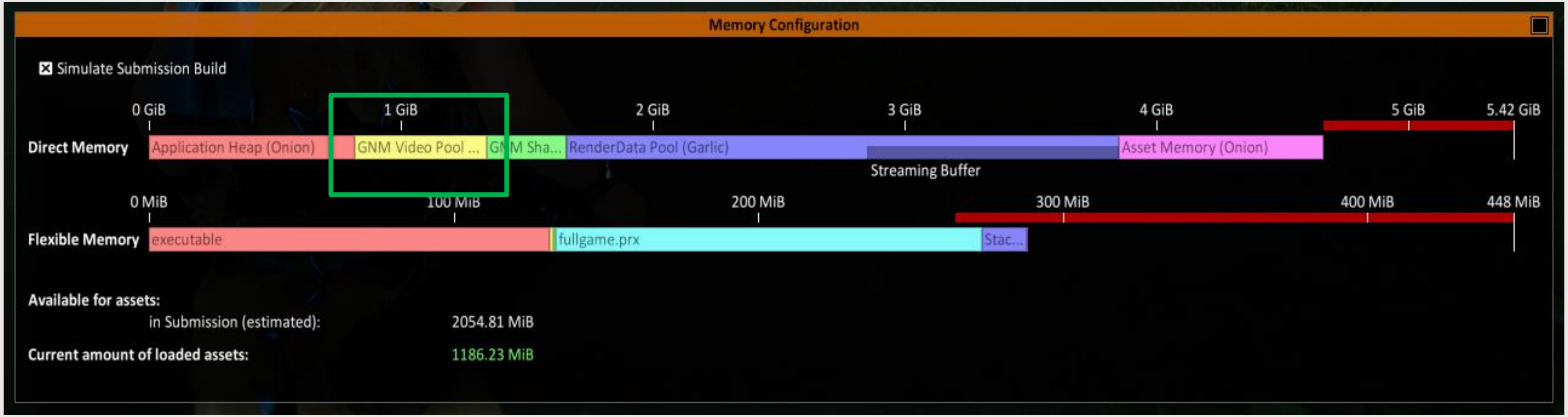
Available for assets:

in Submission (estimated): 2054.81 MiB

Current amount of loaded assets: 1186.23 MiB

GNM Video Pool

- Fixed size, ~500MiB, Garlic
- Render targets, contexts, compute shaders



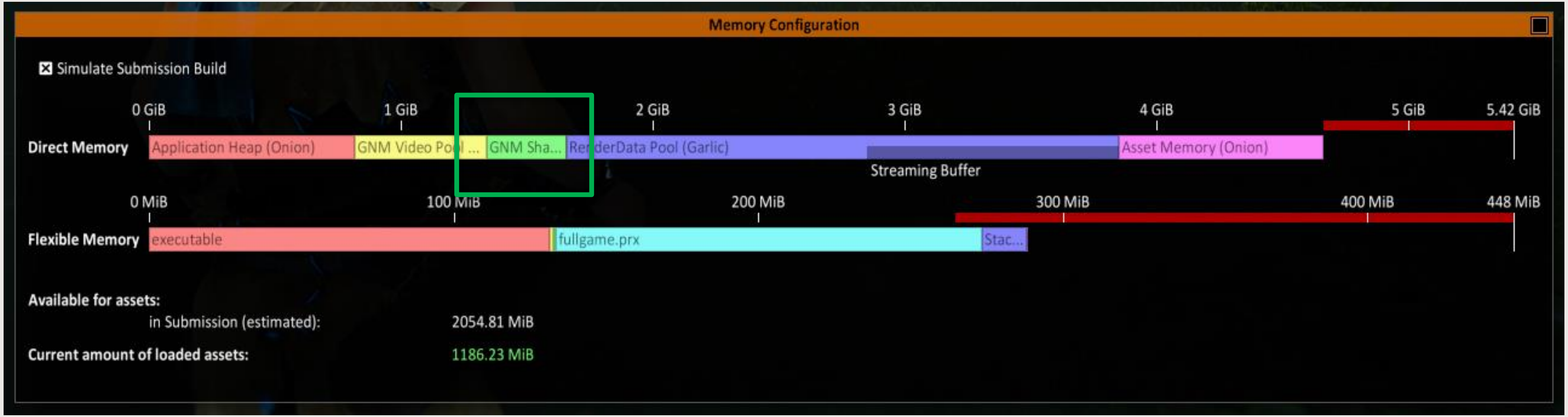
Available for assets:

in Submission (estimated): 2054.81 MiB

Current amount of loaded assets: 1186.23 MiB

GNM Shared Pool

- Fixed size, ~300MiB, Garlic
- Subsystem-specific VRAM data



Memory Configuration

☒ Simulate Submission Build

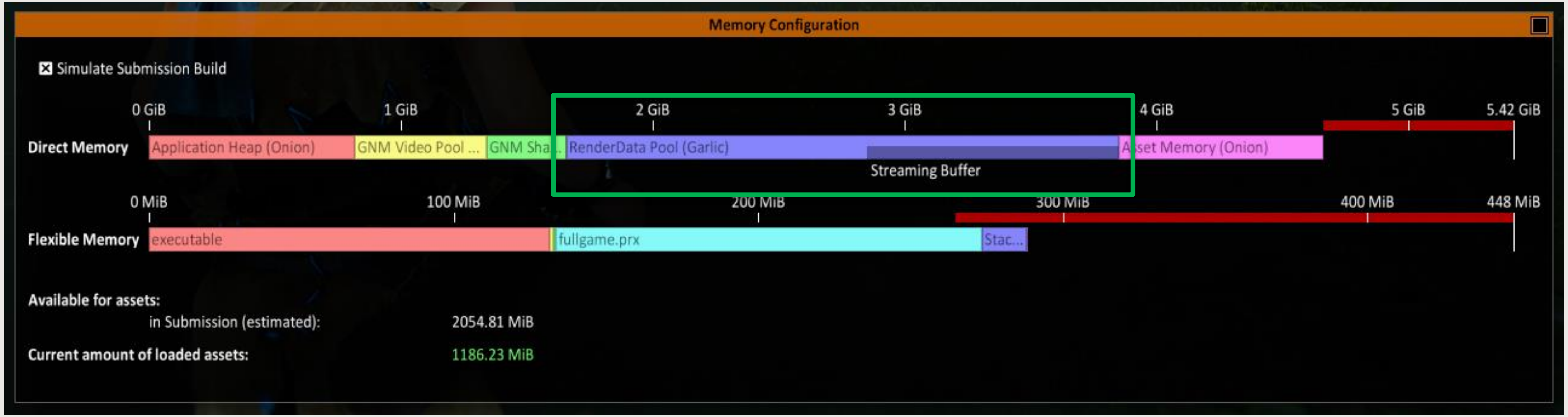
Available for assets:

in Submission (estimated): 2054.81 MiB

Current amount of loaded assets: 1186.23 MiB

RenderData Pool

- Variable size, Garlic
- Contains textures, meshes, shaders



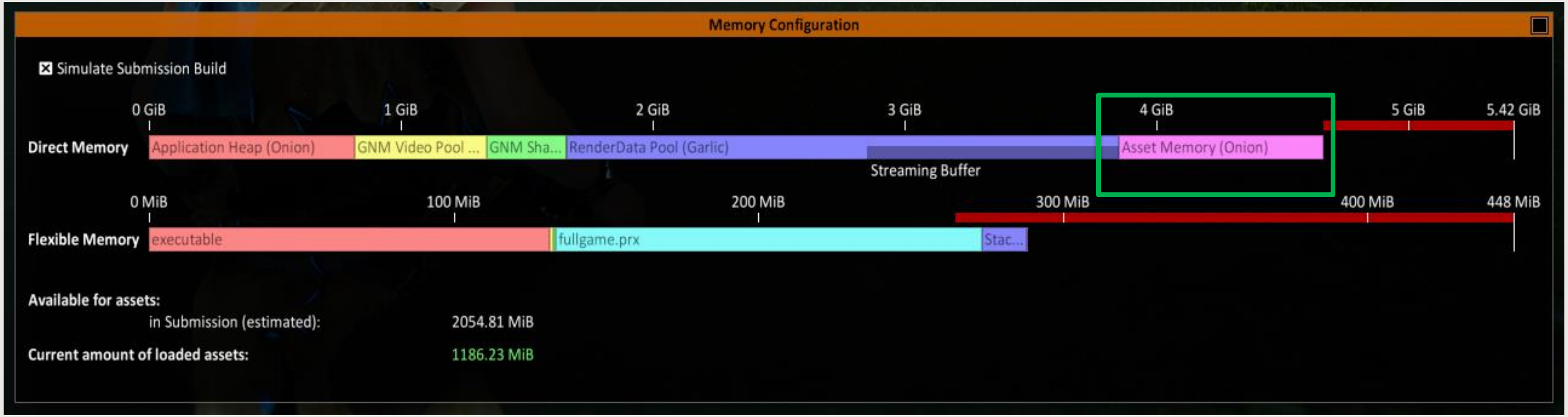
Memory Configuration

☒ Simulate Submission Build

Available for assets:
in Submission (estimated): 2054.81 MiB
Current amount of loaded assets: 1186.23 MiB

AssetMemory Pool

- Variable size, Onion
- Contains object data



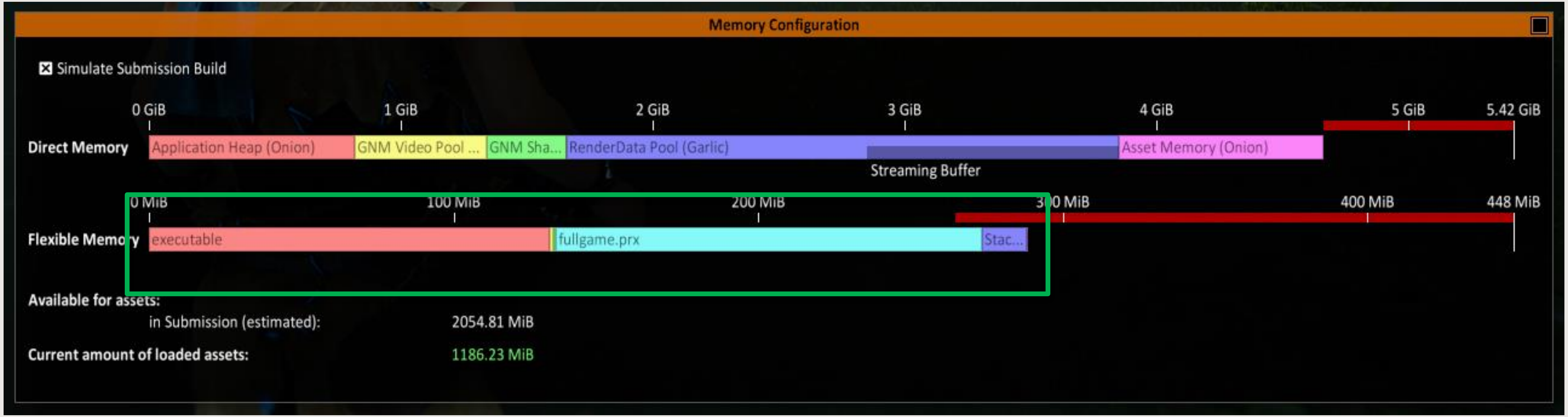
Memory Configuration

☒ Simulate Submission Build

Available for assets:
in Submission (estimated): 2054.81 MiB
Current amount of loaded assets: 1186.23 MiB

Flexible Memory

- ELF, PRX, Stacks
- No application data



Available for assets:
in Submission (estimated): 2054.81 MiB

Current amount of loaded assets: 1186.23 MiB

AssetMemory and RenderData

- Share physical memory, not virtual memory
- All physical memory initially allocated to RenderData
- AssetMemory requests/returns physical memory
- RenderData provides physical memory on demand

RenderData Pool

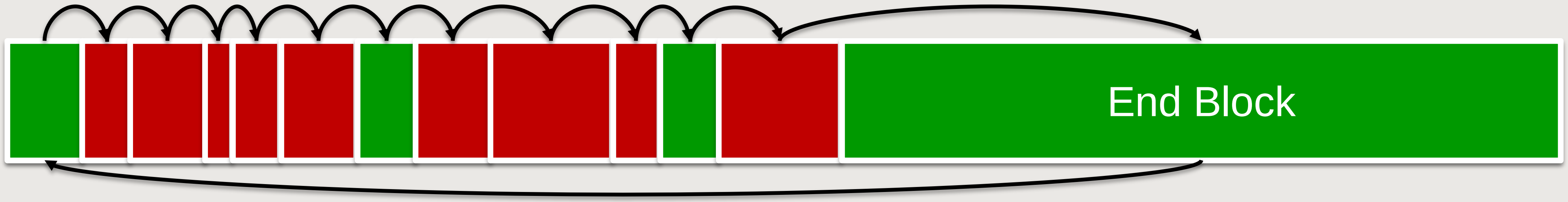
RenderData Pool

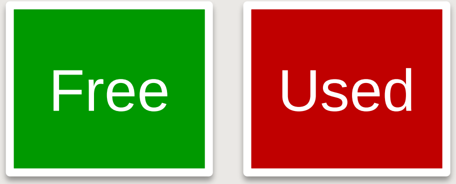
- Manages **VRAM**
- Contains **textures, meshes, shaders**
- Has **static** and **streaming** assets
  - Static: **always loaded** when objects are loaded
  - Streaming: **Optional** mesh LODs/texture MIPs
- **Defragmented** continuously

Layout

- **Contiguous** mapped virtual memory range
- 2MiB page size
- Maintains **block list** (free/used)
- Free blocks moved to end of range
- **Map/unmap** physical memory at end of range

RenderData Pool View





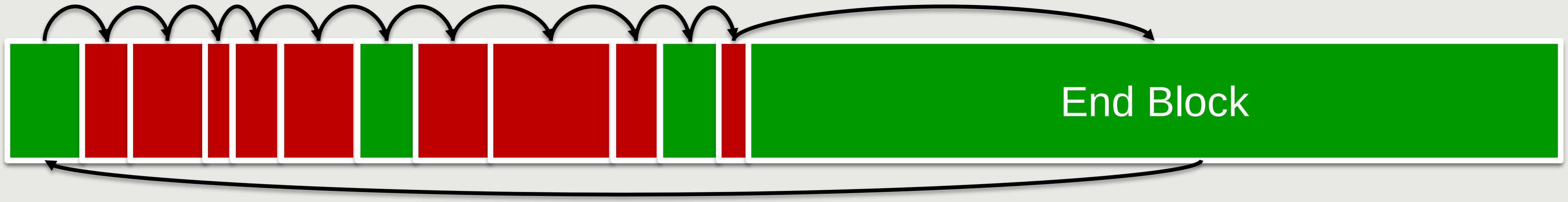
Defragmentation

- Defragmentation has 3 phases:
  1. Frame *M*: Copy used blocks down to fill free space
  2. Frame *M*: Move free blocks up to end of range
  3. Frame *N*: Free copied blocks, then back to (1)
- Used blocks must linger 1 frame, may be in use
- Next frame new address is used, old block freed

Defragmentation Details

- Runs at start of every frame
- CPU determines which blocks to copy
- Maximum of 16MiB copied per frame
- Determines new address for copied blocks
- Schedules copy commands as Async Compute jobs
- After copy, updates handles with new addresses

Defragmentation

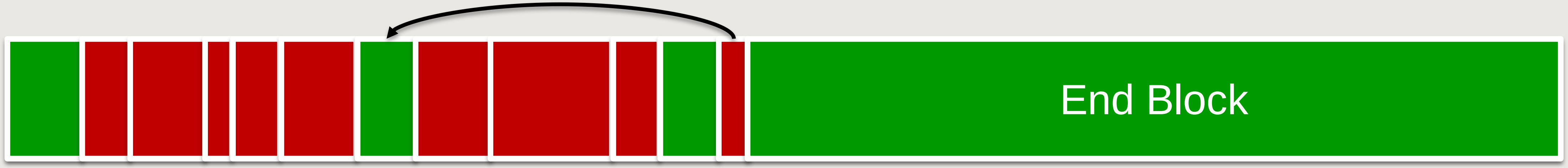


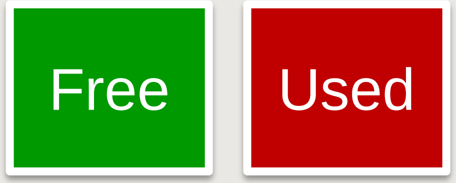
End Block

🟩 Free

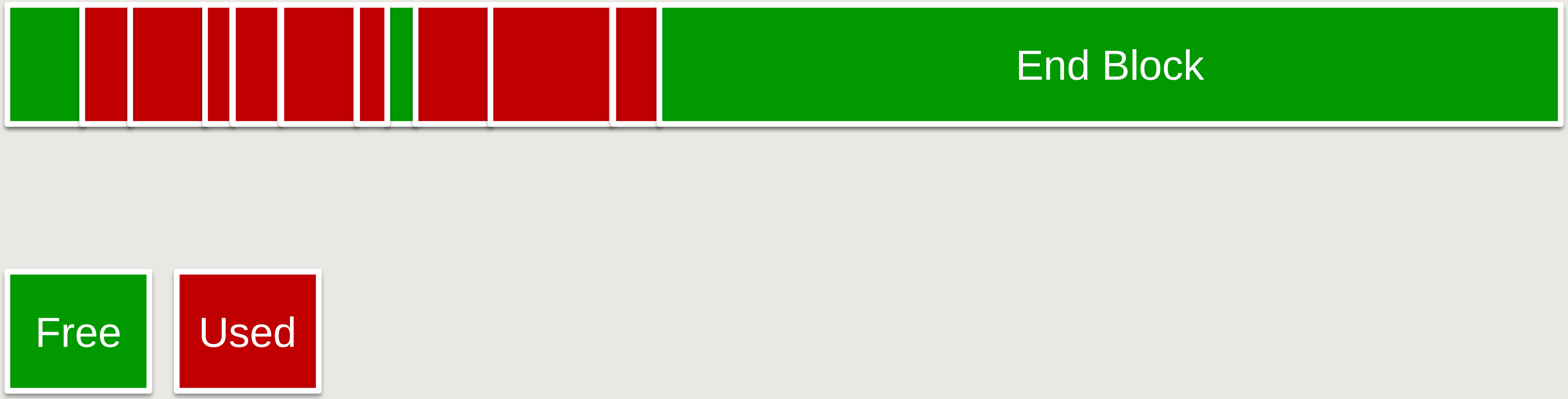
🟥 Used

Move used block down





Move used block down

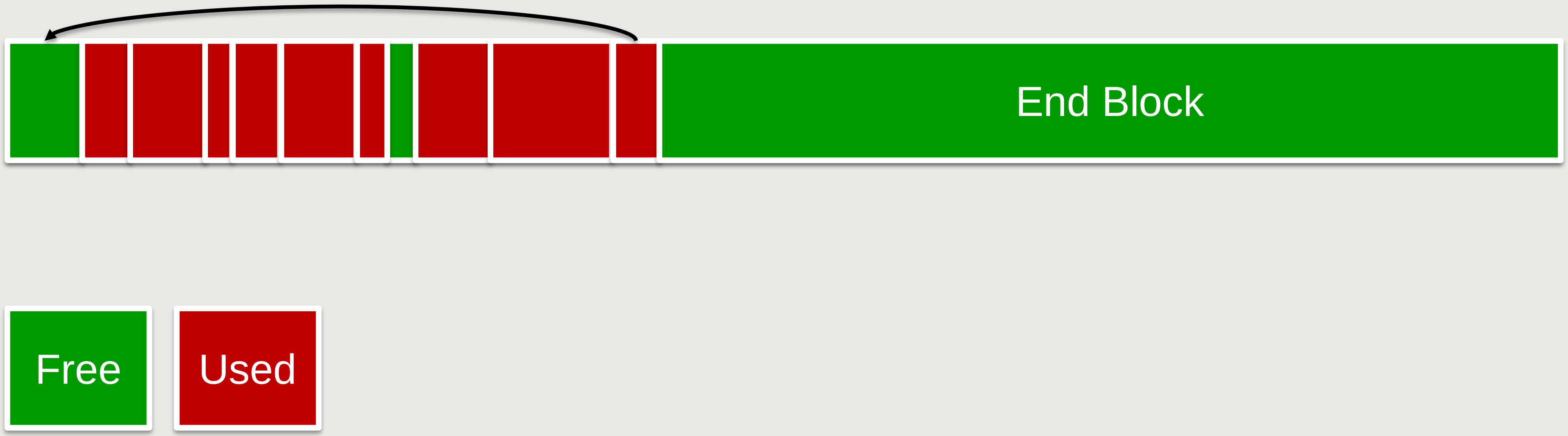


End Block

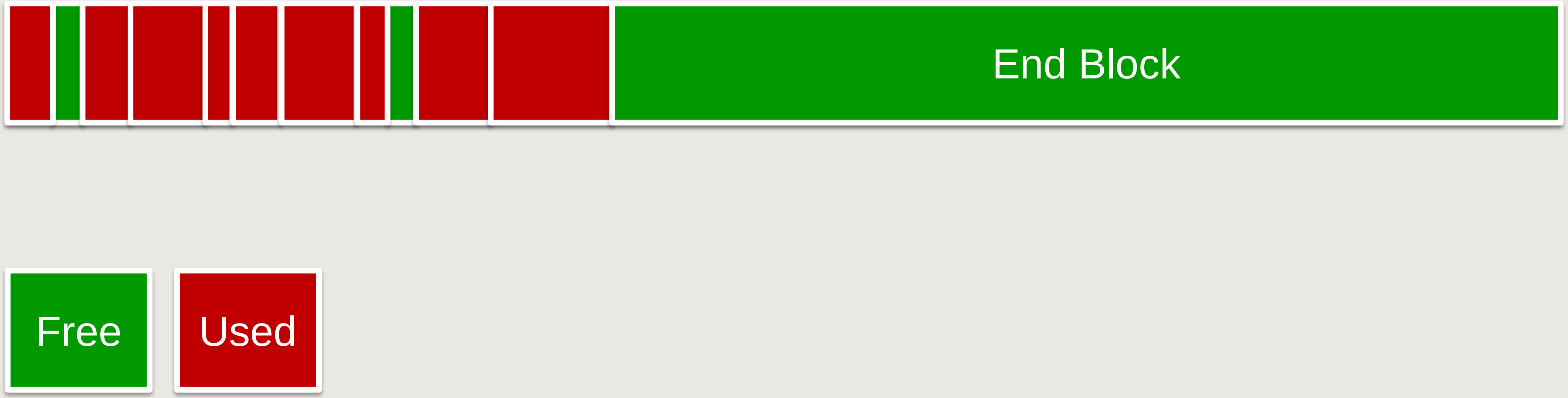
Free

Used

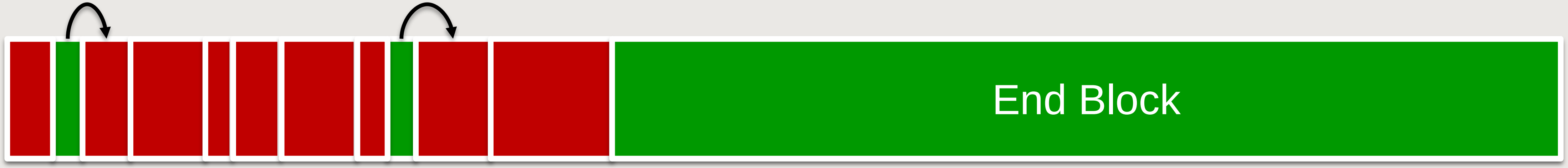
Move used block down

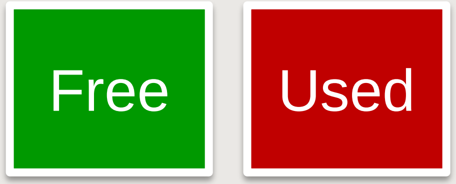


Move used block down

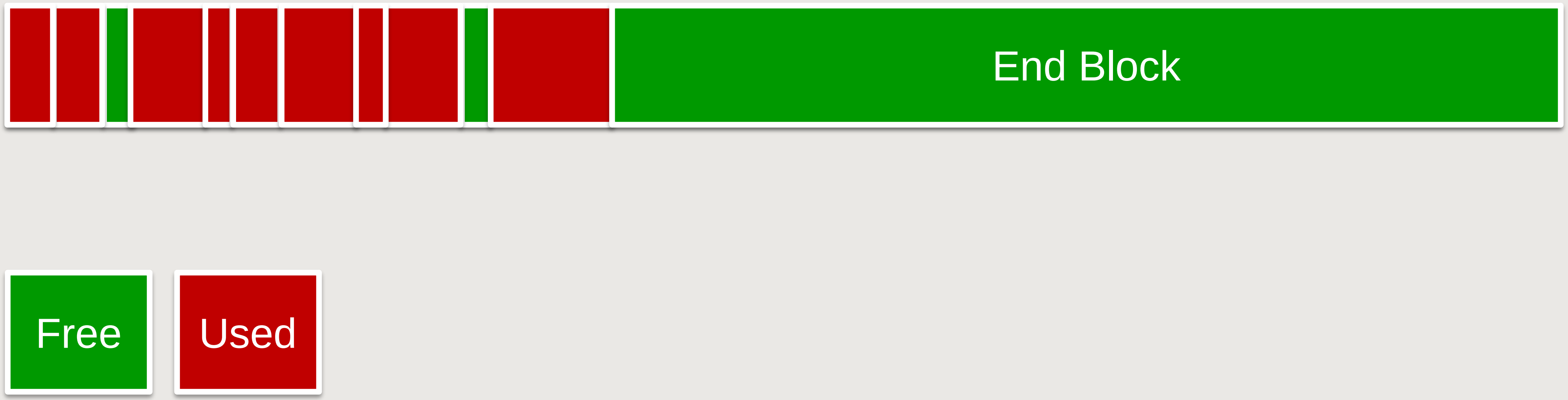


Move free block up

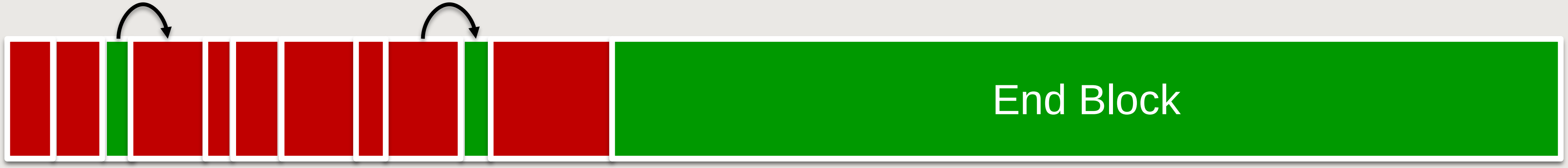


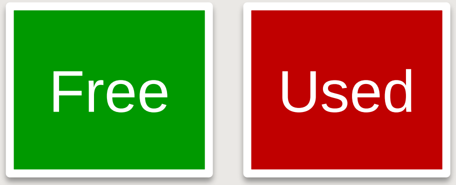


Move free block up

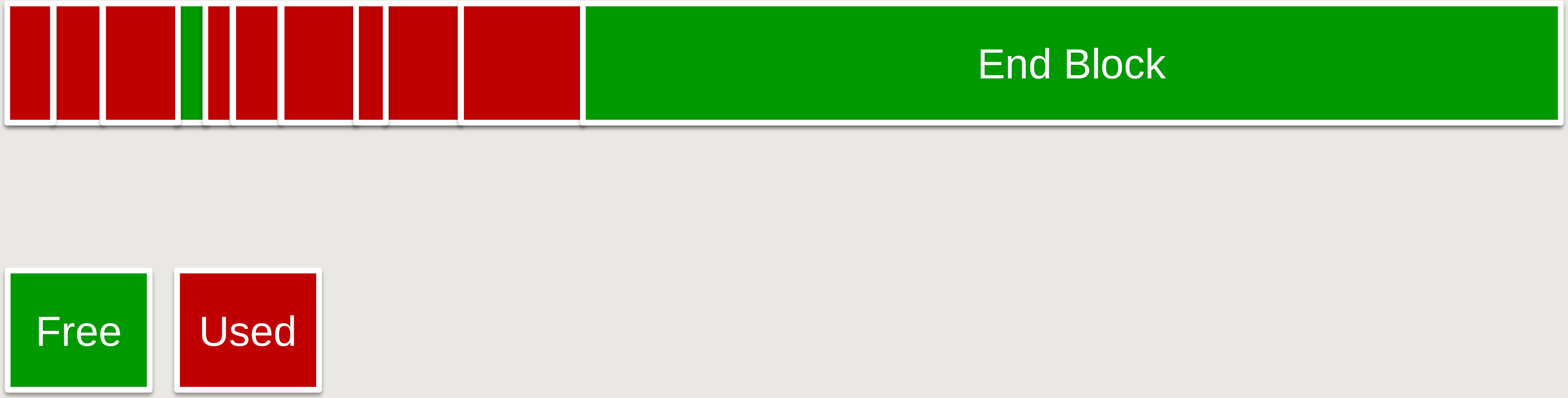


Move free block up





Move free block up

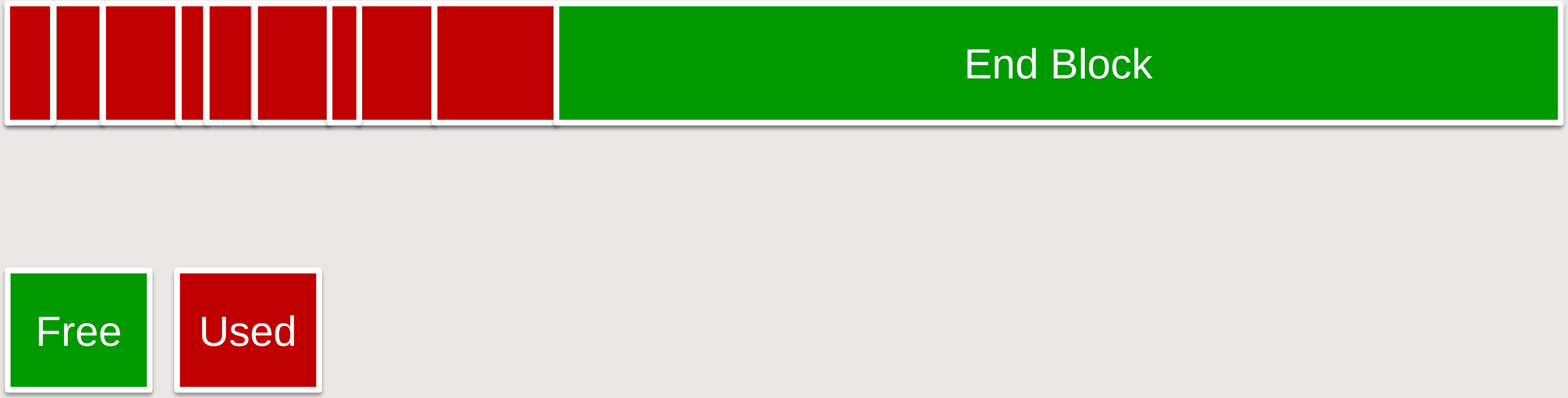


End Block

Free

Used

Fully defragmented



End Block

Free

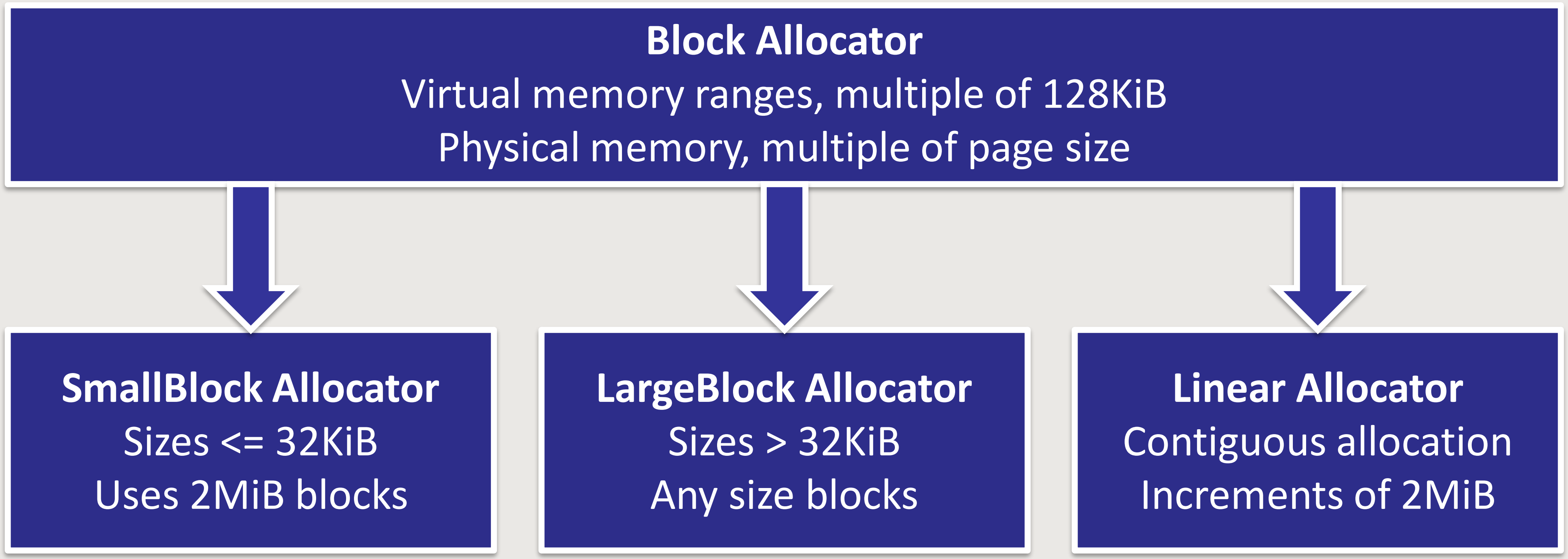
Used

Asset Allocator

Asset Allocator

- Contains objects created through **streaming**
- **Layered** allocator
- Manages **virtual memory ranges**
- Uses **physical memory** requested from **RenderData**

Asset Allocator Structure



Block Allocator
Virtual memory ranges, multiple of 128KiB
Physical memory, multiple of page size

SmallBlock Allocator
Sizes <= 32KiB
Uses 2MiB blocks

LargeBlock Allocator
Sizes > 32KiB
Any size blocks

Linear Allocator
Contiguous allocation
Increments of 2MiB

Block Allocator

- Manages **1GiB regions** of virtual memory
- Splits regions into **128KiB blocks**
- Each block represented by **64B header**
- Header contains pointer to SubAllocator
- **Headers contiguous at start of region**
- 512KiB overhead per **1GiB**

Preventing fragmentation

- Prevent fragmentation of virtual memory:
  - Large virtual memory allocations (128KiB increments)
  - Don’t mix unrelated allocations (lifetime/size)
- Prevent fragmentation of physical memory:
  - Combine equal-size blocks
  - Combine blocks with same lifetime
  - Commit physical memory in 16KiB increments

Allocation

- SubAllocator requests block of size *N*
- BlockAllocator:
  - Ensures enough physical memory is available
  - Allocates **align_up(*N*, 128KiB) virtual** address range
  - Maps **align_up(*N*, 16KiB) physical memory** to range
  - Sets pointer to SubAllocator in block header

Deallocation

- Pointer resolved to SubAllocator
- **SubAllocator**:
  - Updates own bookkeeping for block
  - If block empty, returns it to BlockAllocator
- **BlockAllocator**:
  - Unmaps physical memory
  - Marks virtual range as free
  - Updates physical free size

Obtaining physical memory

- BlockAllocator:
  - **Requests 64MiB from RenderData**
- RenderData:
  - **Unstreams** low prio LODs/MIPs
  - **Defragments** free space to end of range
  - **Unmaps 64MiB** and shrinks RenderData
- Available (unmapped) memory grown by 64MiB

Releasing physical memory

- BlockAllocator:
  - If > 64MiB **physical memory free**, notifies RenderData
- RenderData Pool:
  - **Maps 64MiB** physical memory at end of pool
  - **Grows pool size**
  - Starts streaming LODs/MIPs into available memory

SmallBlockAllocator

- **Manages allocations <= 32KiB**
- Buckets **per size class**
- Each **bucket is linked list** of 2MiB blocks
- Each block is split into **2MiB/(size class) entries**
- **Free list** maintained in empty entries

LargeBlockAllocator

- Single **allocations > 32KiB**
- Allocates **align_up(N, 16KiB)** from BlockAllocator
- BlockAllocator **allocates align_up(*N*, 128KiB)** range
- BlockAllocator **maps align_up(*N*, 16KiB)** memory
- Average overhead:
  - 8KiB physical
  - 64KiB virtual

LinearAllocator

- Collects **multiple >32KiB** allocations
- Maintains **multiple >= 2MiB blocks**
- Only used for **allocations with identical lifetime**
- Memory freed only after all allocations freed
- **Fast alloc**, only increment pointer in large block
- **Fast free**, release all blocks when done
- **Low overhead** for alignment/bookkeeping

Memory Management Conclusions

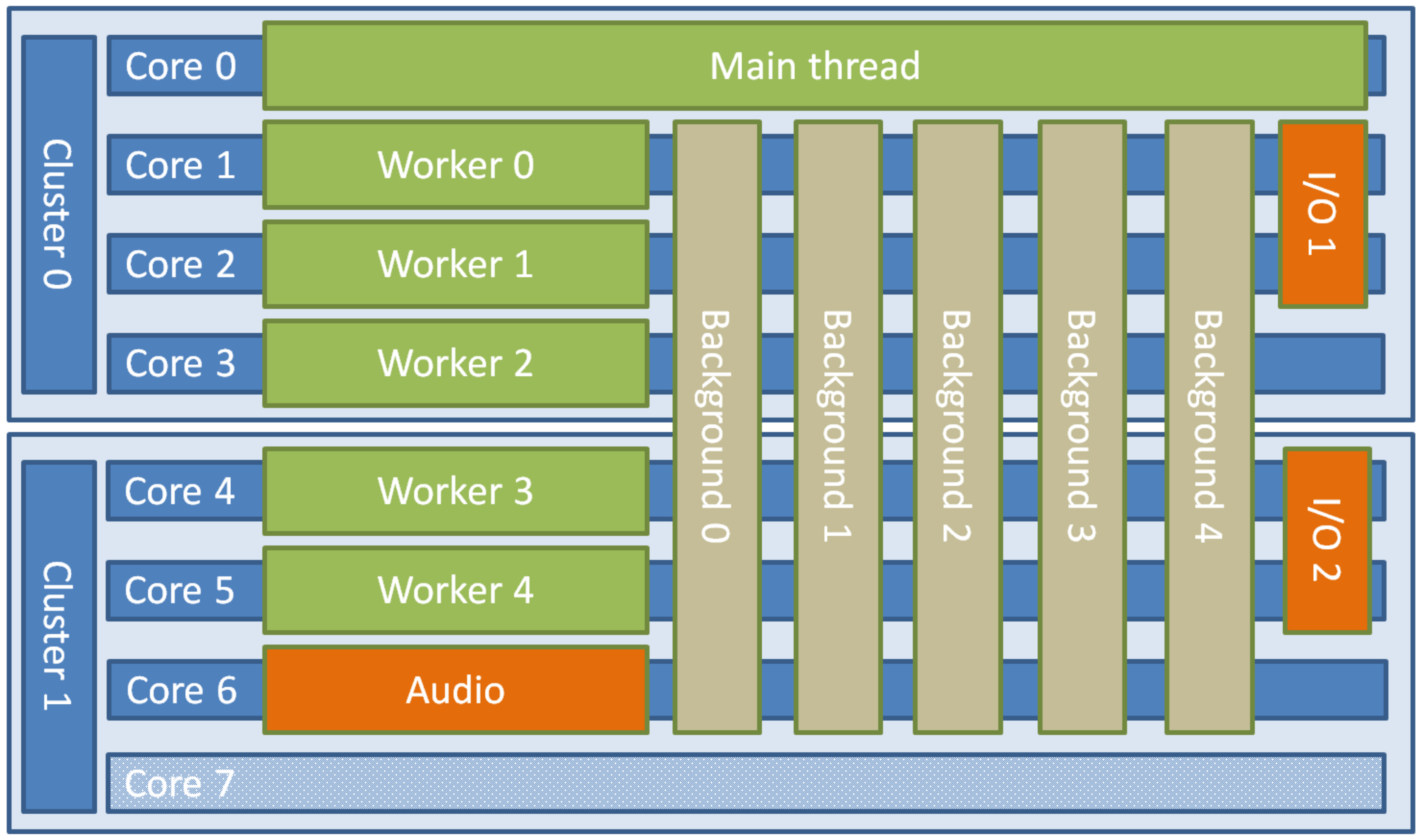
- **Shared memory** between Onion/Garlic works well
- Map/unmap **overhead is low**
- **Allows for dynamic budgets**
- Defragmentation:
  - **Expensive and complex**
  - **But almost no waste**

CPU Scheduling

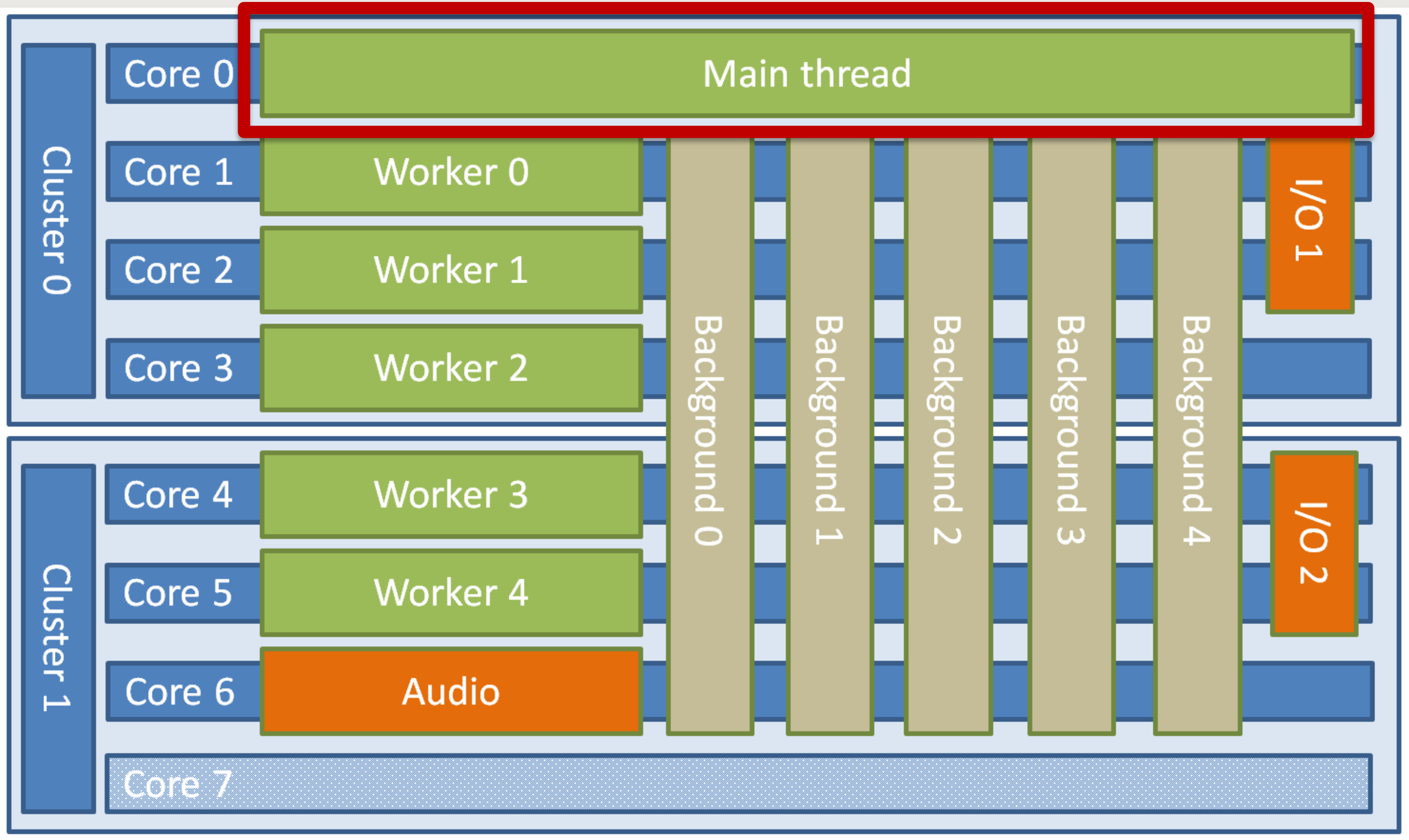
CPU Scheduling

- Threads managed by **job scheduler**
- Two job types:
  - **Frame jobs** (must complete each frame)
  - **Non-frame jobs** (long-running jobs)
- **Three priorities** for each job type
- Carefully selected **thread affinities**

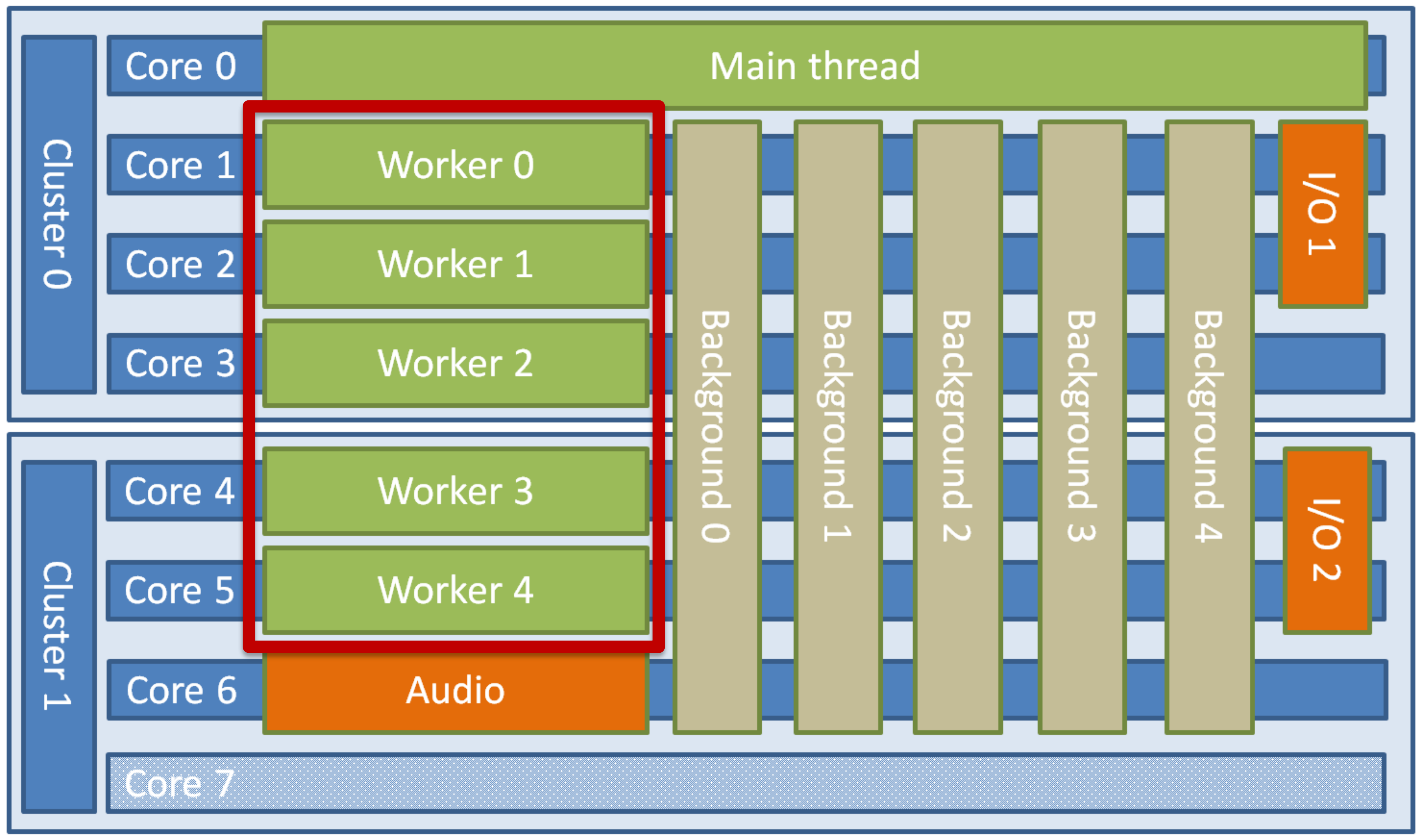
Scheduling and Thread Affinity Model



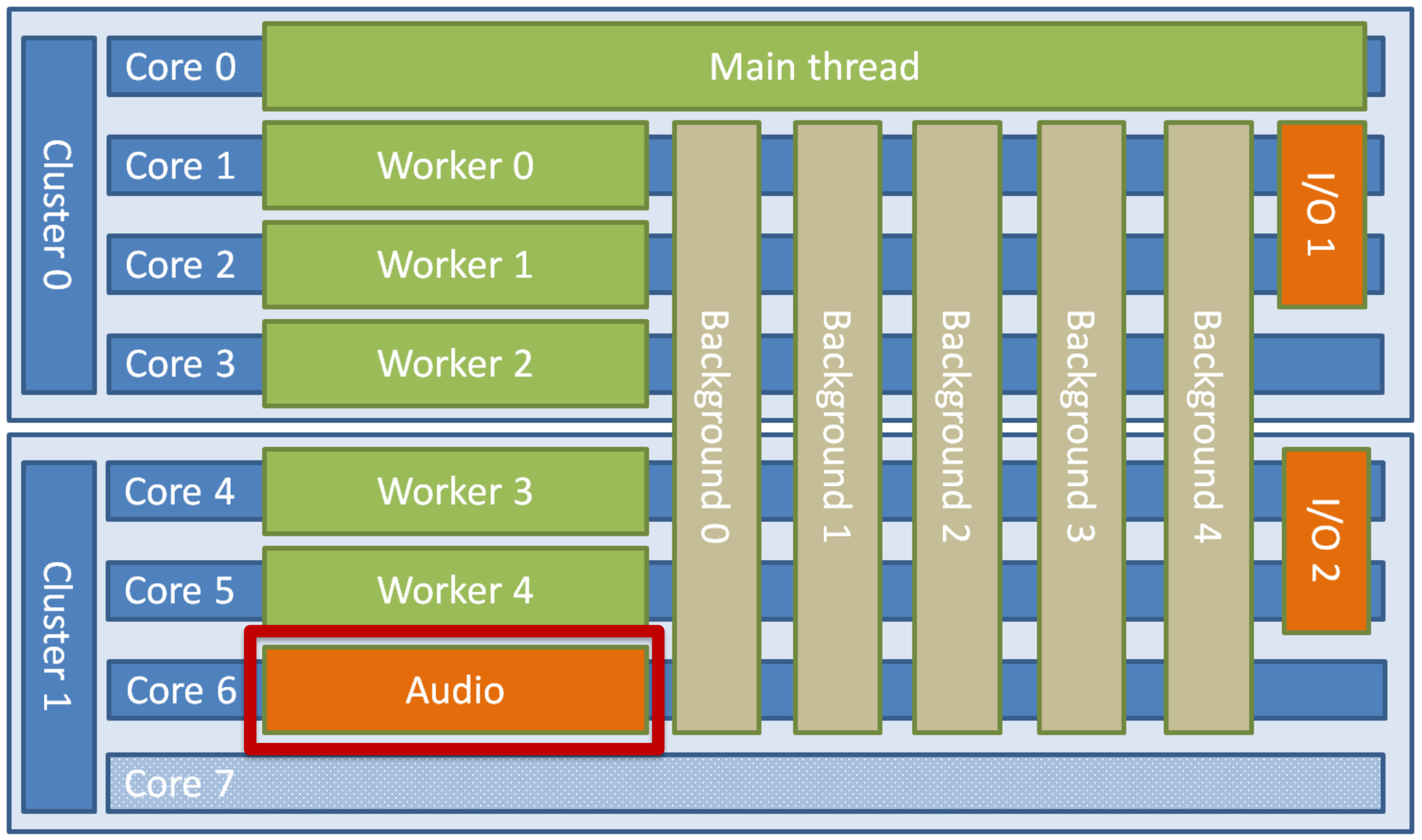
Main Thread


Main thread
Cluster 0
Core 0
Core 1
Worker 0
Core 2
Worker 1
Core 3
Worker 2
Background 0
Background 1
Background 2
Background 3
Background 4
I/O 1
Cluster 1
Core 4
Worker 3
Core 5
Worker 4
Core 6
Audio
I/O 2
Core 7

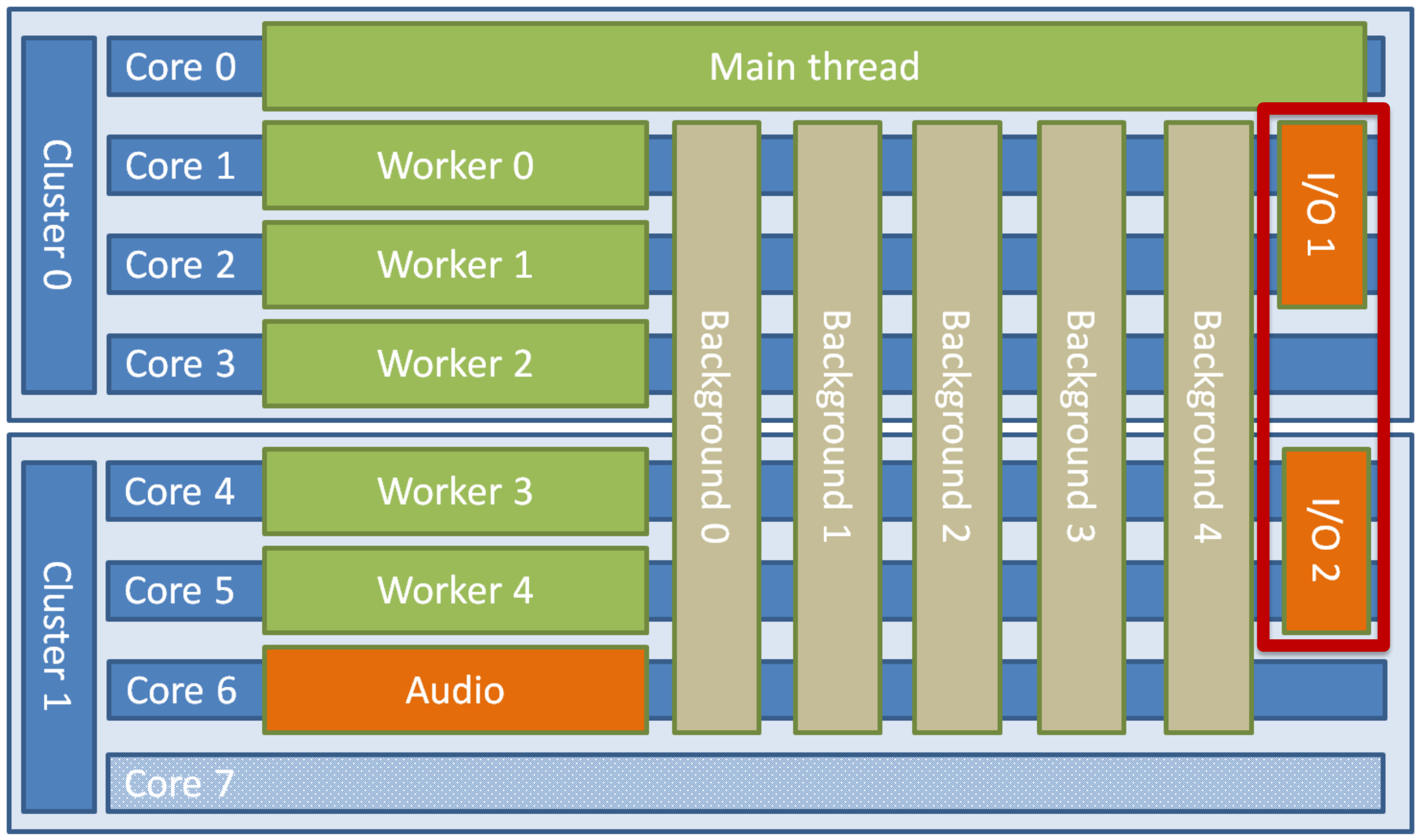
Worker Threads



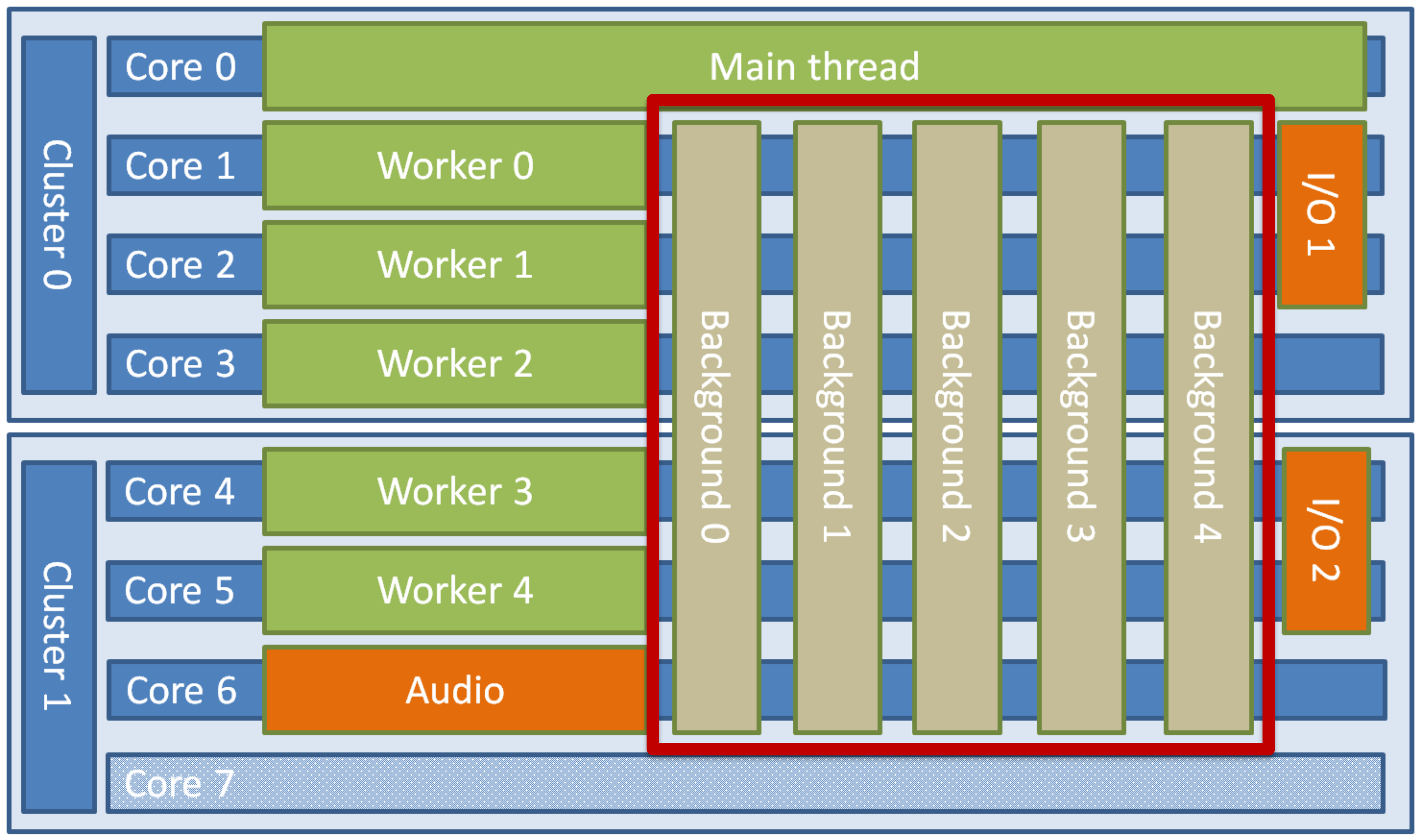
Audio Thread



I/O Threads


Cluster 0
Cluster 1
Core 0
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Main thread
Worker 0
Worker 1
Worker 2
Worker 3
Worker 4
Audio
Background 0
Background 1
Background 2
Background 3
Background 4
I/O 1
I/O 2

Background Threads



CPU Scheduling Conclusions

- **Full core occupancy** achieved
- **Clean separation** between frame and non-frame jobs
- Non-frame jobs run in **idle time** of frame jobs
- **Very few custom threads** due to flexible system
- **Better guarantees** about completion and deadlines

Future Plans

Future plans

- **Load object graphs**, not file graphs
- Use **key-value store** for object storage/retrieval
- **Hybrid VRAM solution**:
  - **Defragmentation** for small allocations
  - **Virtual memory** for large allocations

Questions?