



# Memory and C++ debugging at Electronic Arts

By Scott Wardle

# Introduction

- ▶ Memory interfaces and debug tools for C++ games
  - ▶ 2000 (PS2) embedded C style
  - ▶ 2005 (Xbox 360, PS3) Interface programming, EASTL
  - ▶ Now (PS4, Xbox One) 64 bit address spaces
- ▶ Our Current Tools:
  - ▶ How all of our debugging systems work together

# About me

- Scott Wardle,
- 20+ years Game Dev
- Solving problems through visualization and drawing pictures
- I am also badly dyslexic, so please note spelling mistakes and inform me later after the presentation 😊

# Vocabulary

- Allocators, Arena, Heaps
  - Allocators (an object or interface that can alloc and free)
  - Arena (a set of address ranges controlled by one allocator)
  - From Arena find an Allocator
  - From Allocator find an Arena
  - Heap  $\sim$  Allocator + Arena

# C style 2000s Overview

- Year ~2000: PS2 32M ram
  - Most people are using C++ compilers
  - STL is not used
  - No virtual memory
  - Nearly no OS
  - Similar to embedded systems

# C style 2000s

## Interfaces for speed

- ▶ Macro per class

- ▶ **#define NEW\_DELETE\_OPERATORS**(debug\_name)

- ▶ Good for fixed sized pools or slabs of objects

```
class CollisionChooser {
```

```
public:
```

```
    NEW_DELETE_OPERATORS(CollisionChooser)
```

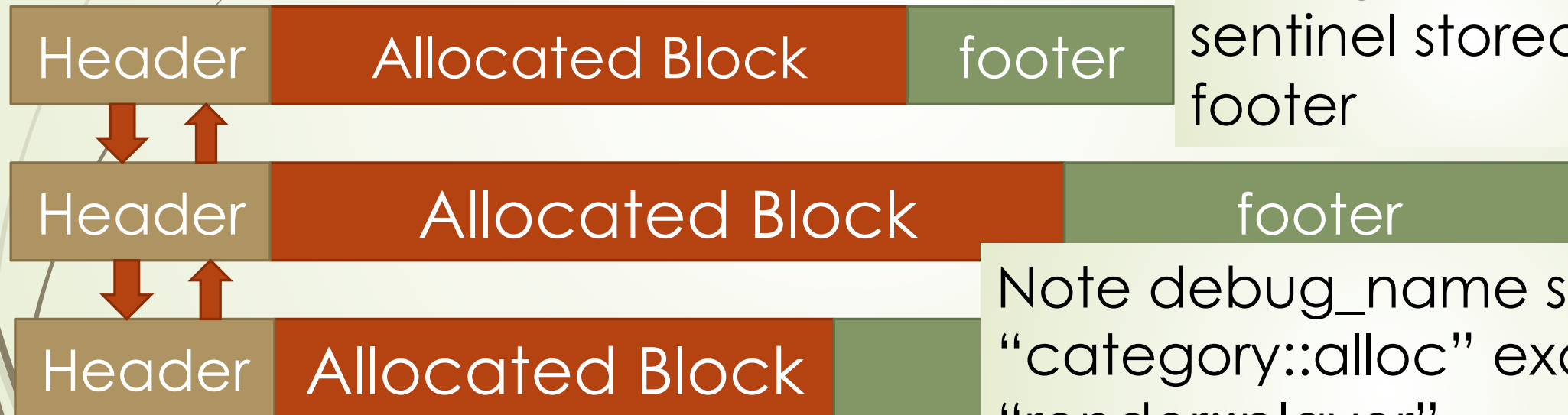
```
    ...
```

```
};
```

# C style 2000s Interfaces for debug

- ▶ Global new
- ▶ `void* operator new(size_t size, const char* debug_name, int flags=MB_LOW)`

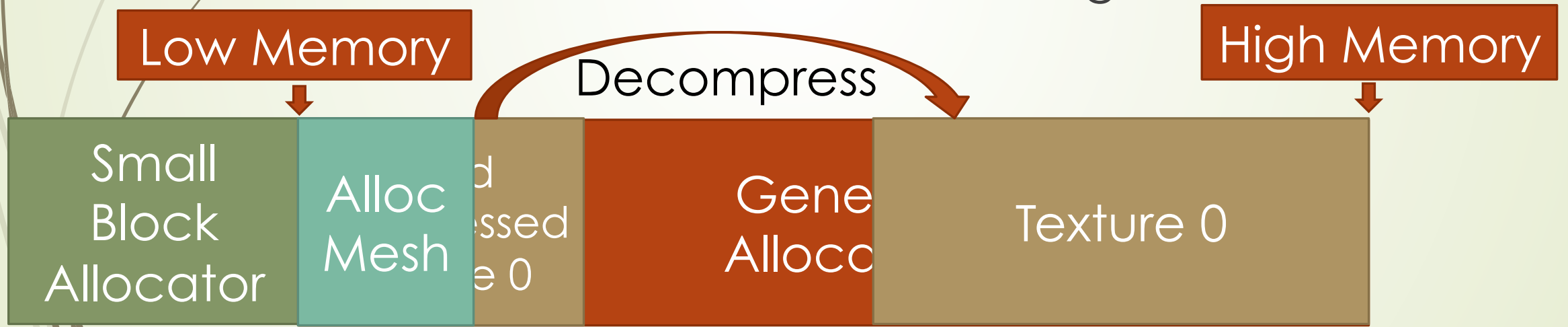
Debug name and sentinel stored in footer



Note debug\_name split into “category::alloc” example: “render::player”, “gameplay::physicsmesh”

# C style 2000s Allocation technology

- Almost all memory is in one heap
- Well we did have a simple small block allocator
- We had to work hard at defragmentation





## 2005 Overview

- 2004 - Xbox 360, PS3 (512M ram)
- Virtual memory! - NO HDD ☹, No GPU support, 32 bit
- All consoles have multiple CPUs
  - (Not just for Sega Saturn)
- The main changes for 2005:
  - Support for multiple allocators
  - Better tracking and logging tools
  - Stomp allocator!!
  - Memory tracking with EASTL

# 2005 Support for Multiple Allocator

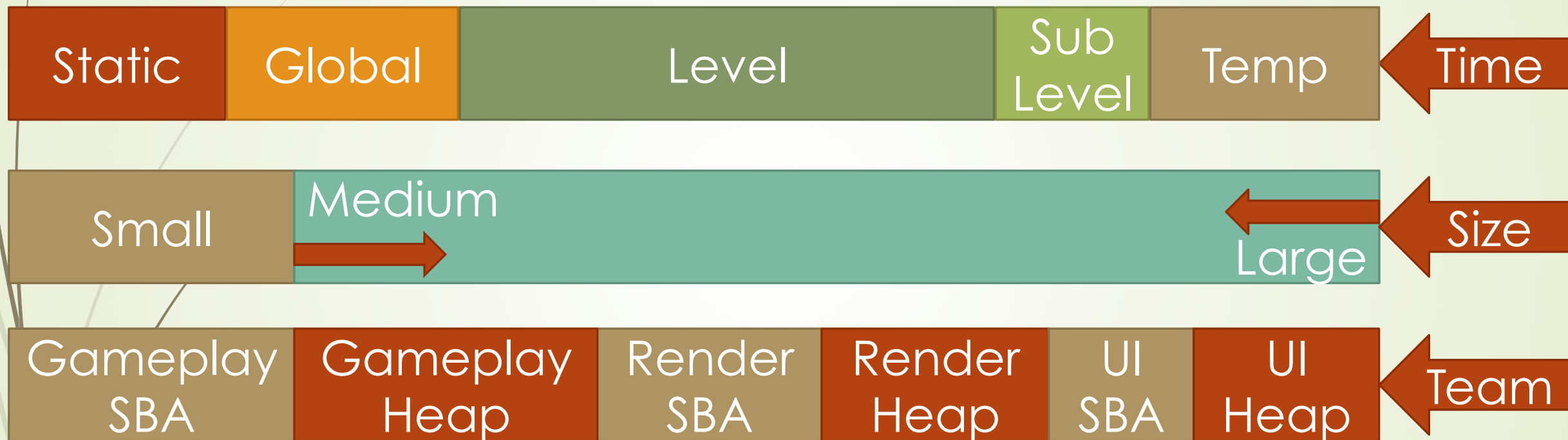
Polymorphic Allocator

```
SQLQuery *NewQuery(ICoreAllocator* a) {  
    return CORE_NEW(a, "sql", MEM_LOW) SQLQuery(a);  
}
```

```
void DeleteQuery(ICoreAllocator* a, SQLQuery *sql){  
    CORE_DELETE(a, sql);  
}
```

Calls ~SQLQuery()  
not delete!!

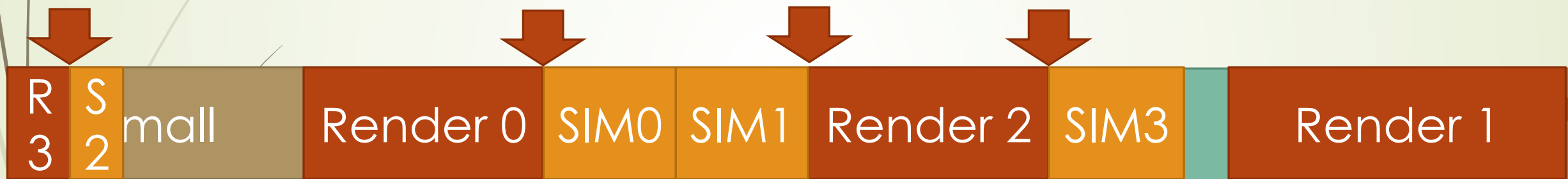
## 2005 Organizing Heaps/Arenas



A mix of time and size gives good defragmentation properties. Organizing by team fragments heaps but easy to set blame. So for my team we use all of these to varying degrees.

## 2005 Team Based Heaps/Arenas vs Team Based Categories

Memory Corruption between teams sucks



Categories are a way to tag allocations so you can budget them together.

Fragmentation between teams is hard. Who to blame when you are out of memory?

# 2005 Better Tracking and Logging

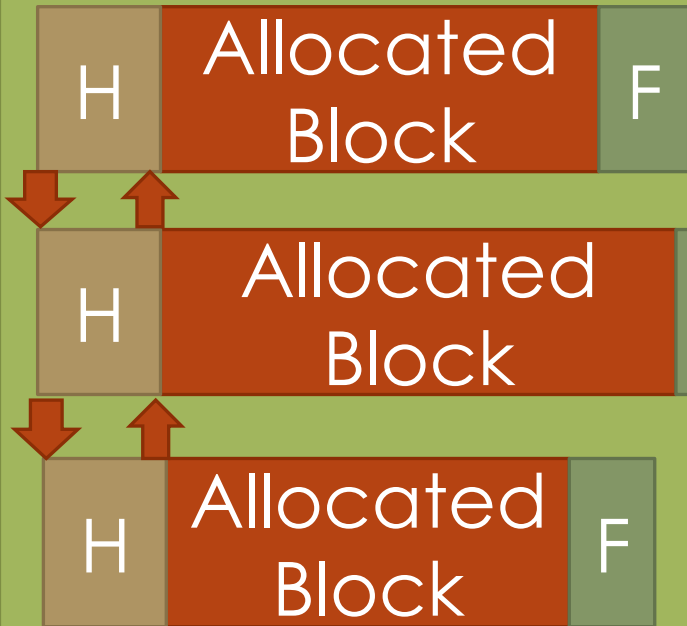
13

Only sentinel stored in footer

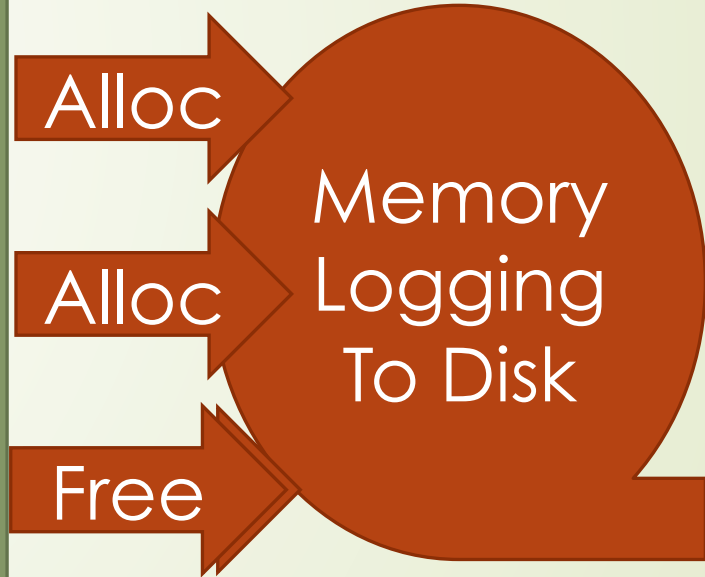
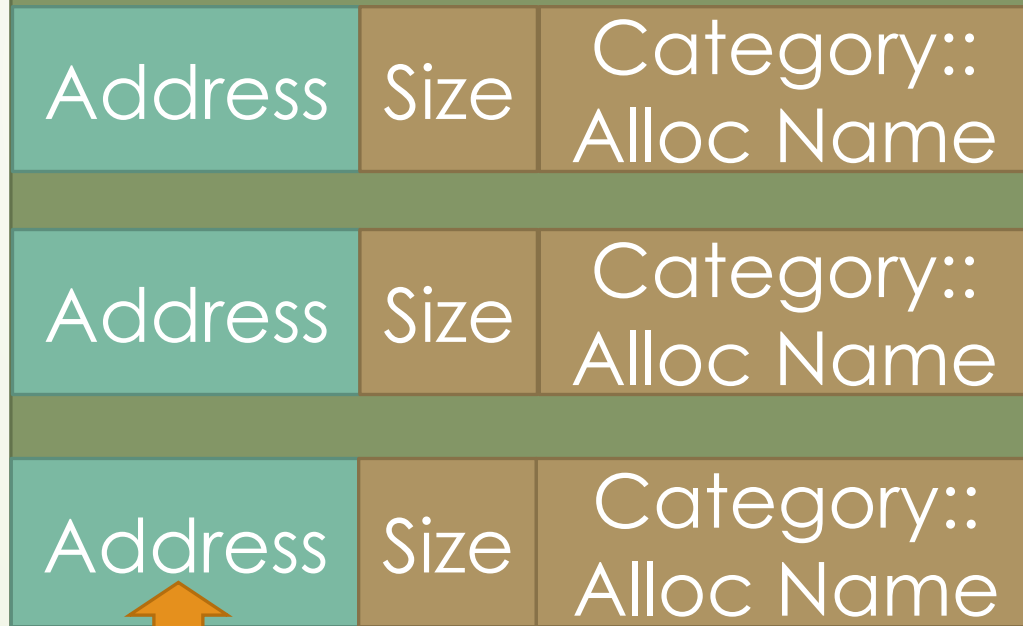
Tracking live allocations in a separate heap.

Memory Logging or tracing system

Normal Heap

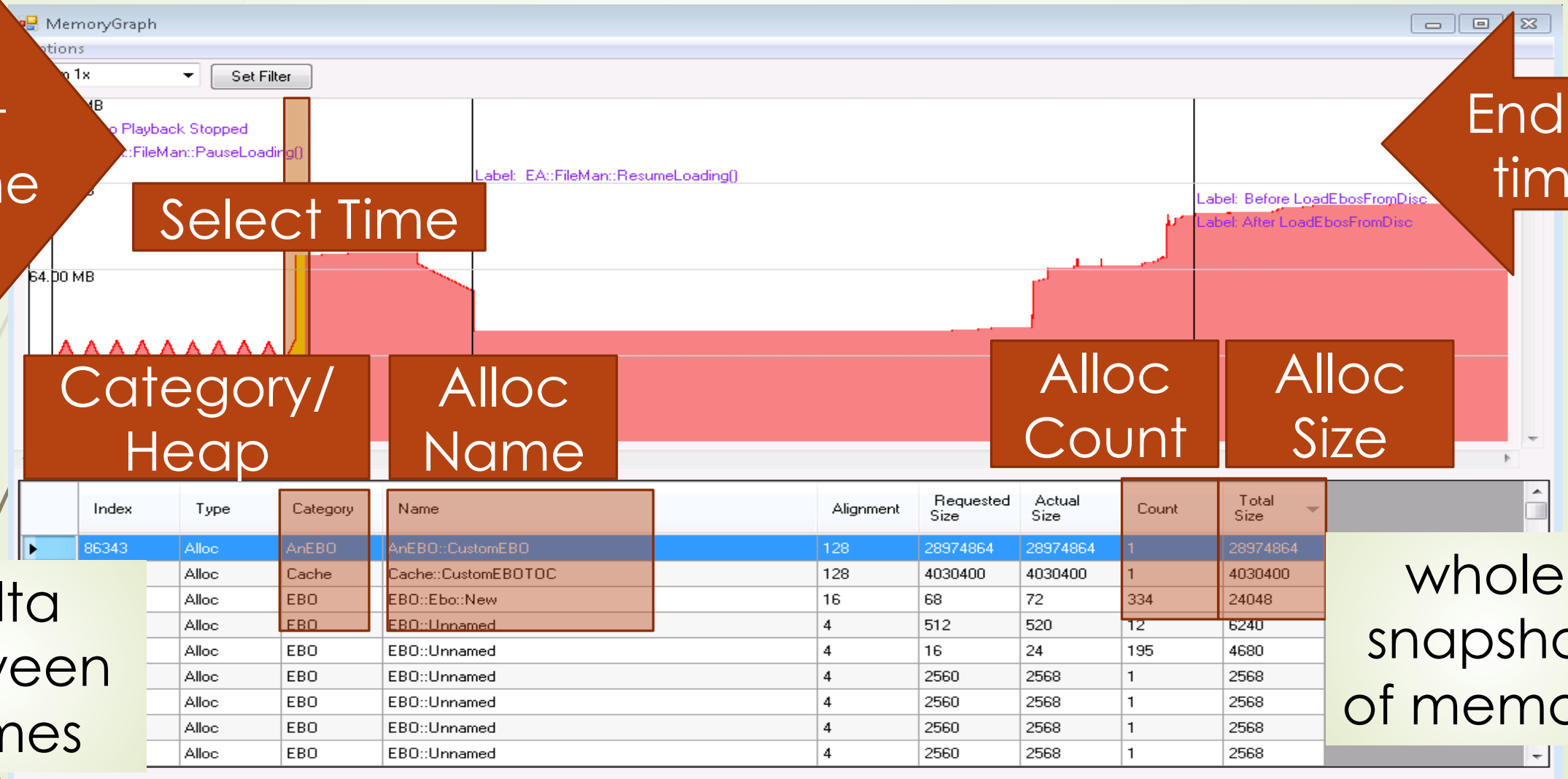


Debug Heap



Hash Key

## 2005 Logging



# 2005 Arena Block View

The screenshot displays the 'Block View - GlobalPP - 2252.675k' window. The main area is a grid representing memory blocks. A yellow block is selected, indicated by a green arrow pointing to it with the text 'Yellow selected block'. Other blocks are colored green (labeled 'Green Systems'), purple (labeled 'Purple Presentation'), and grey (labeled 'Grey Free').

Below the grid, a panel provides information about the selected block:

```
Selection Start Address: 0x02EA2880
Selection End Address: 0x02EB7080
Total In Range: 83,968 B
Used: 83,888 B
Free: 80 B
Usage: 99.90%
Largest Contiguous Free Block: 32 B

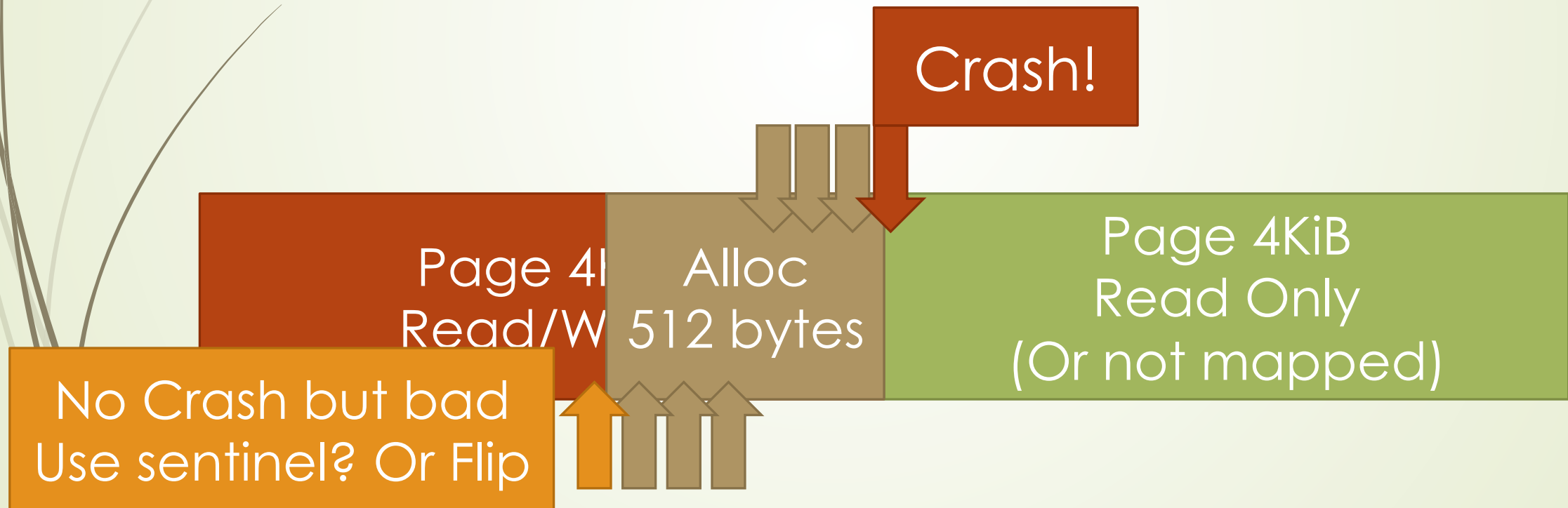
-----
Name: FrontEnd:fefifa:loungeModeMan
Address: 0x02EA2BB0
Callstack:
Temporary: False
Total Allocated Size: 82896
```

A 'Color Legend' window is also visible, showing color mappings for 'Basic Colors' and 'Categories'. The 'Basic Colors' section includes: Empty Block (grey), Outside Heap (black), Block Selection (blue), Allocation Selection (yellow), and Multi-Cat... Block (magenta). The 'Categories' section includes: System (green), Global (red), FrontEnd (blue), Debug (brown), Strings (teal), and Presenta... (purple).

At the bottom left, a status bar shows: '99 Blocks Per Row (99 KB) - 1442.93 KB Total Free'.

## 2005 Stomp Allocator!!

- Stomp Allocator – so good it is worth it's own slide
- Lots of memory, 4k per alloc



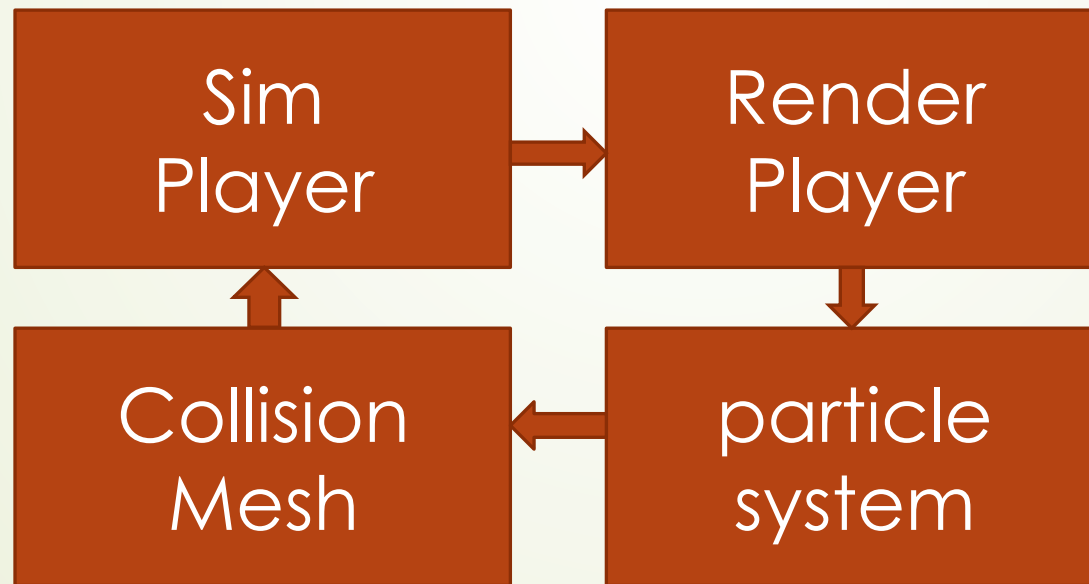


# 2005 Ref Counted Pointers

17

- Add a debug system for ref counts is hard:
  - A Tracking system would be like garbage collector...
  - A Logging system would generate even more data...

Oh No  
memory  
leak!



shared\_ptr  
are useful !!  
  
but use  
unique\_ptr  
or bare pointers  
for easy life times

## 2005 EASTL

- A 2010 version of EASTL is available now from webkit.
- Why EASTL
  - STL allocators are painful to work with
  - Intrusive containers, Ring Buffers, etc...
  - Superior readability and performance
  - Memory is Allocated in empty versions of some STL objects
  - Etc...

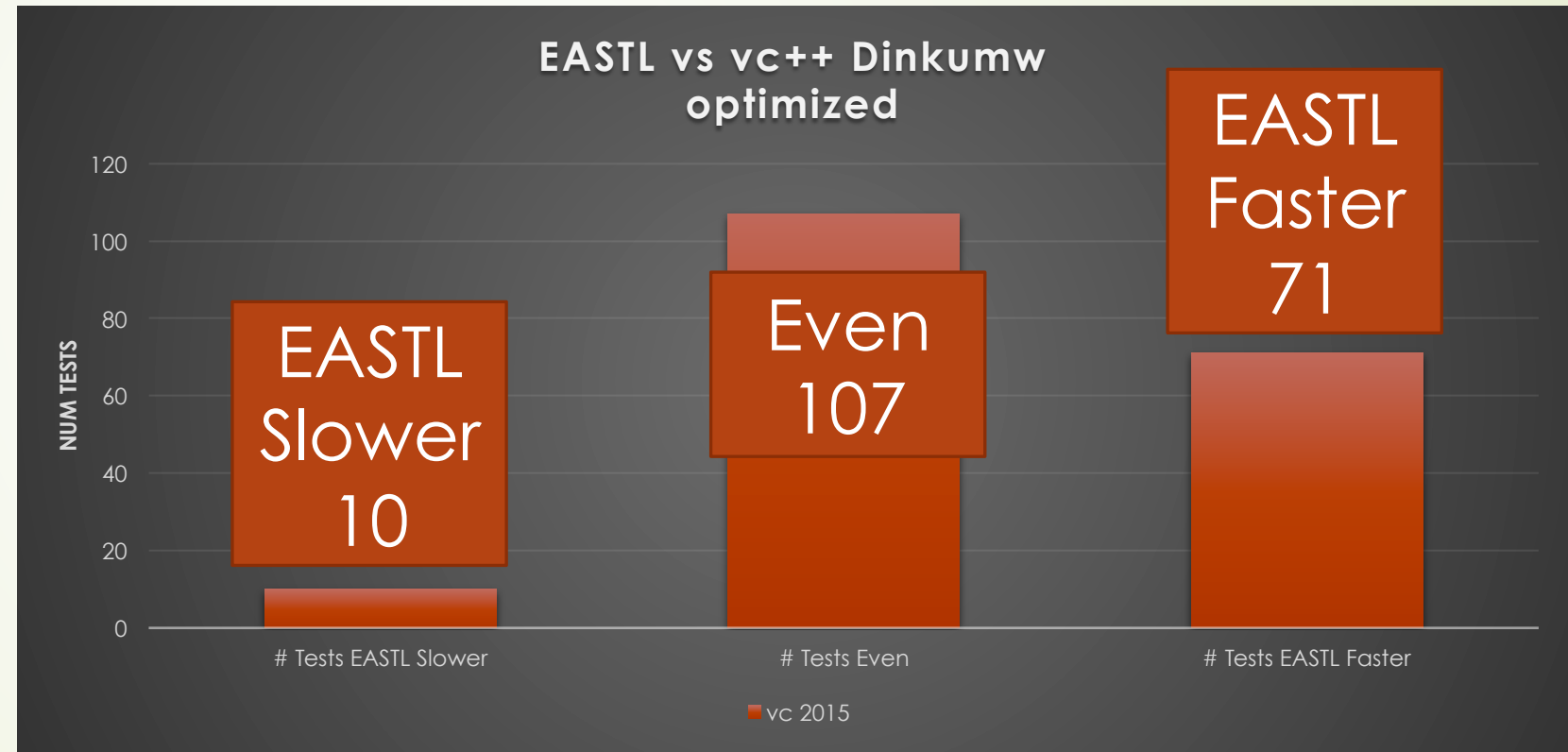
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2271.html>

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4526.pdf>

# EASTL faster for optimized code

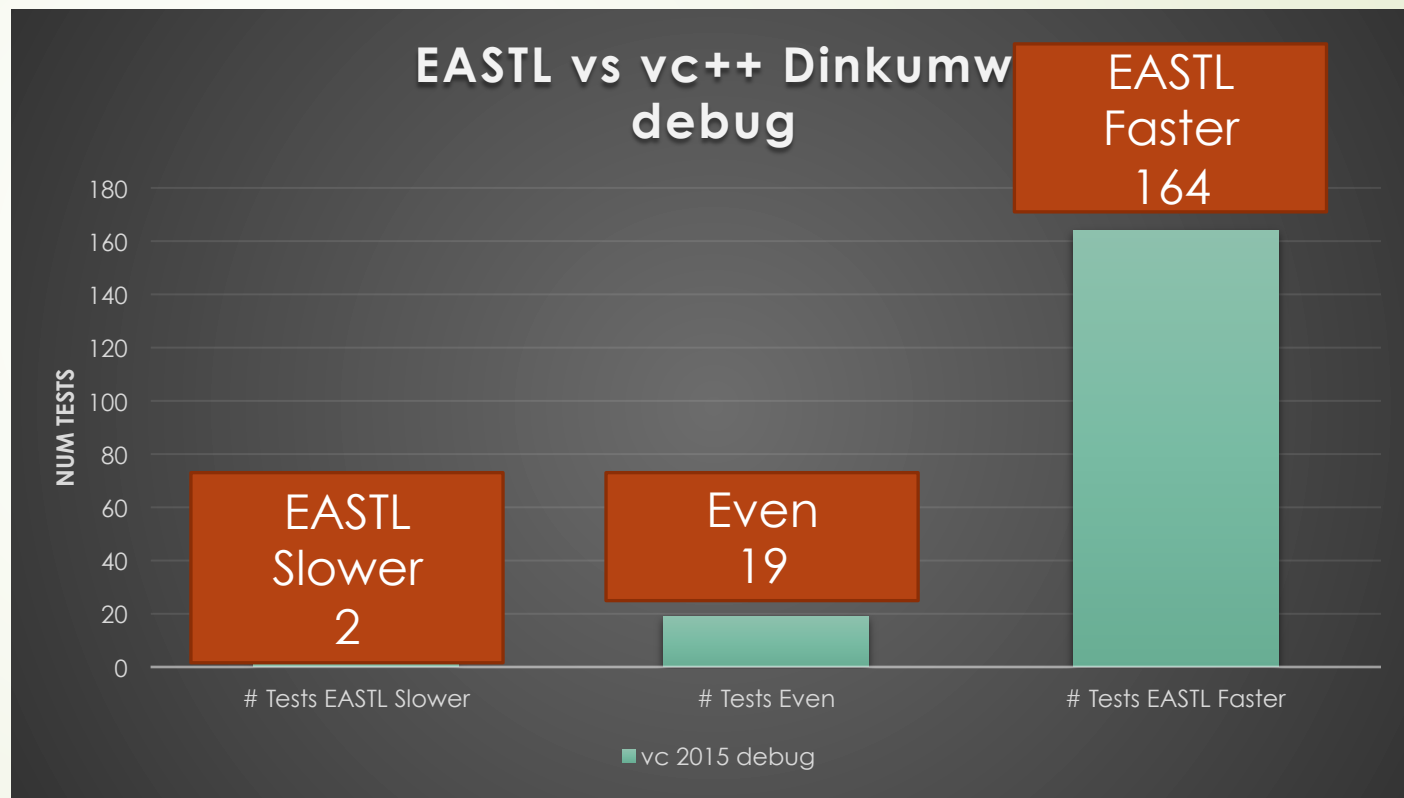
EASTL is often a little faster. In 71 out of 188 tests.

- Faster means 1.3x or better.
- Slower means 0.8x as quick or slower



# EASTL MUCH faster for debug code

The same 188 tests  
compiled in debug



# Open sourcing EASTL

- EA is looking to open source EASTL
- Roberto Parolin will be taking pull requests
- Coming soon to:
  - <https://github.com/electronicarts>
- Technical details announce later to SG14 group

# 2005 EASTL Memory tracking problems

22

➤ EASTL's allocator were painful to track every object


➤ You need to make a new type

```
typedef eastl::vector<int, EASTLCoreAllocator>  
MyVec;
```



➤ Then pass in a defaulted parameter

```
ICoreAllocator* alloc = GetGameplayAllocator();  
MyVec vec(alloc);
```




# 2005 EASTL Memory tracking problems

23

- Default parameters at the end so hard to enforce use.

```
unordered_map ( size_type n = 1000  
  const hasher& hf = hasher(),  
  const key_equal& eql = key_equal(),  
  const allocator_type& alloc = allocator_type() );
```



- Worked on all EASTL types but clumsy

```
EA::ICoreAllocator* alloc = GetRenderAllocator();  
vec.get_allocator().set_allocator(alloc);
```



# 2005 EASTL Memory tracking problems

24

- At first we hacked EASTL to make it easier

`vector`

`v(eastl::allocator( "AI::Piano::Input" ))`



- But this meant our team couldn't share code... with other teams
- (Accessing allocator by name was a bad idea anyways)



# 2005 EASTL Memory tracking problems

25

- We also ran into type erasure problems

```
typedef vector<int, EASTLCoreAllocator> MyVec;
```

```
typedef vector<int> YourVec;
```

```
MyVec myVec;
```

```
YourVec yourVec;
```

```
myVec = yourVec; // what should happen here...
```

- ERROR: no operator found which takes a right-hand operand of type 'YourVec' (or there is no acceptable conversion)

# 2005 “Good?” EASTL usage with EASTLICA

26

- Wrap EASTL with EASTLICA to force usage of polymorphic allocator

```
template <typename T>
class String : public base_string<T, EASTLICoreAllocator>{
    String(ICoreAllocator *alloc, const char*name="Str")
    : basic_string<char, EASTLICoreAllocator>(
        EASTLICoreAllocator( name, alloc ))
    ...
};
ICoreAllocator* alloc = GetStringAllocator();
EASTLICA::String str(alloc);
```

# 2005 "Good?" EASTL usage with EASTLICA

27

- Macro used to implement STL like types for each large system.

```
#define EASTLICA_VECTOR( EASTLICA_TYPE, ,  
 GET_DEFAULT_ALLOC, ALLOC_NAME ) \
```

```
template< typename T> class EASTLICA_TYPE : public  
EASTLICA::Vector<T>
```

- Using Macro to create a STL-like types for a large system

```
EASTLICA_STRING( CareerModeString, ,
```

```
 CareerMode::GetStringDefaultAllocator(), "CareerStr" );
```

# 2005 “Good?” EASTL usage with EASTLICA

28

- This fixed our type type erasure problems.

```
CareerModeString str;
```

```
LocalizedString lstr = getStrId(42);
```

```
str = lstr; // woot no compile error! Both use same allocator.
```

- This also fixed the ownership issues.
- CareerMode owns its strings and localization does not own all strings in the game.
- Allocators are copied sometimes but not always.

# Today's Memory System

29

- PS4, Xbox One – Today 8GB (5GB for the game)
  - GPU memory does not have to be linearly mapped. (GPU assets are still special case however.)
  - 64 bit virtual address space and a HDD to swap to.
- The big changes these days:
  - Debug Memory System
  - EASTL Memory Tracking
  - New debug tools

# Today's Debug Memory System

- Alloc debug names slowly die
  - void\* operator new(size\_t size, **EA::ICoreAllocator\* alloc**)
  - The old interface exists. But uses scopes.
- Scopes are everywhere
  - Resource and Asset Names
  - Alloc Name, Allocator, Category, and Call stacks
  - `FB_MEMORYTRACKER_SCOPE(data->debugNames[i]);`
  - `FB_ALLOC_RES_SCOPE(data->debugNames[i]);`
  - \*(This does mean more thread local storage use)

# Today's EASTL Memory Tracking

- Everyone is still doing this:

```
class Team
```

```
{
```

```
    int teamid;
```

```
    eastl::vector<player> players;
```

```
}
```

```
Team*home = new (allocator) Team;
```

- However EASTL is still a problem

# EASTL use parent arena by default tracking

32

Check What  
Arena parent  
is in

Allocate Child  
using parent  
arena  
as parameter

Gameplay Arena

Team Home  
(One Allocation)

```
int teamId;  
vector<player> players;
```

```
allocator (0 bytes maybe)  
first  
last  
end
```

Player 1

Player 2

Player 3

Don't have to use  
the same arena for  
child

IE: use gameplay's  
small block  
allocator  
not general  
allocator



# EASTL use parent arena by default tracking

- Problems
- It does take some CPU time.
- What about objects on the stack?
- What about move operators?
  - Object in gameplay arena and move it to rendering. Only the parent object will move.
- “You made it you own it” logic works 80% of the time.
- For other cases use EASTLICA patterns.
  - (Systems that are factory for other systems.)

# Today's Debugging Tool DeltaViewer

- History lesson over! Let's look at today's tools!
- DeltaViewer displays a session of data.
- A session is one run of the game
- This data is sent from console to a http server on the SE's or QA's computer
- The data is stored in tables
- These tables can be joined into views

# DeltaViewer

- Some popular views are:
  - TTY events debugging (Trace Log)
  - IO Load profiler (Turbo Tuner)
  - Frame rate and Job thread profiler (Performance Timer)
  - Memory Investigator, reviews memory leaks and changes over time
  - Memory Categorization groups allocations at a given time

# TTY events debugging (Trace Log)

Trace Log Turbo Tuner

Theme: Single Color Threshold: All Filter Text

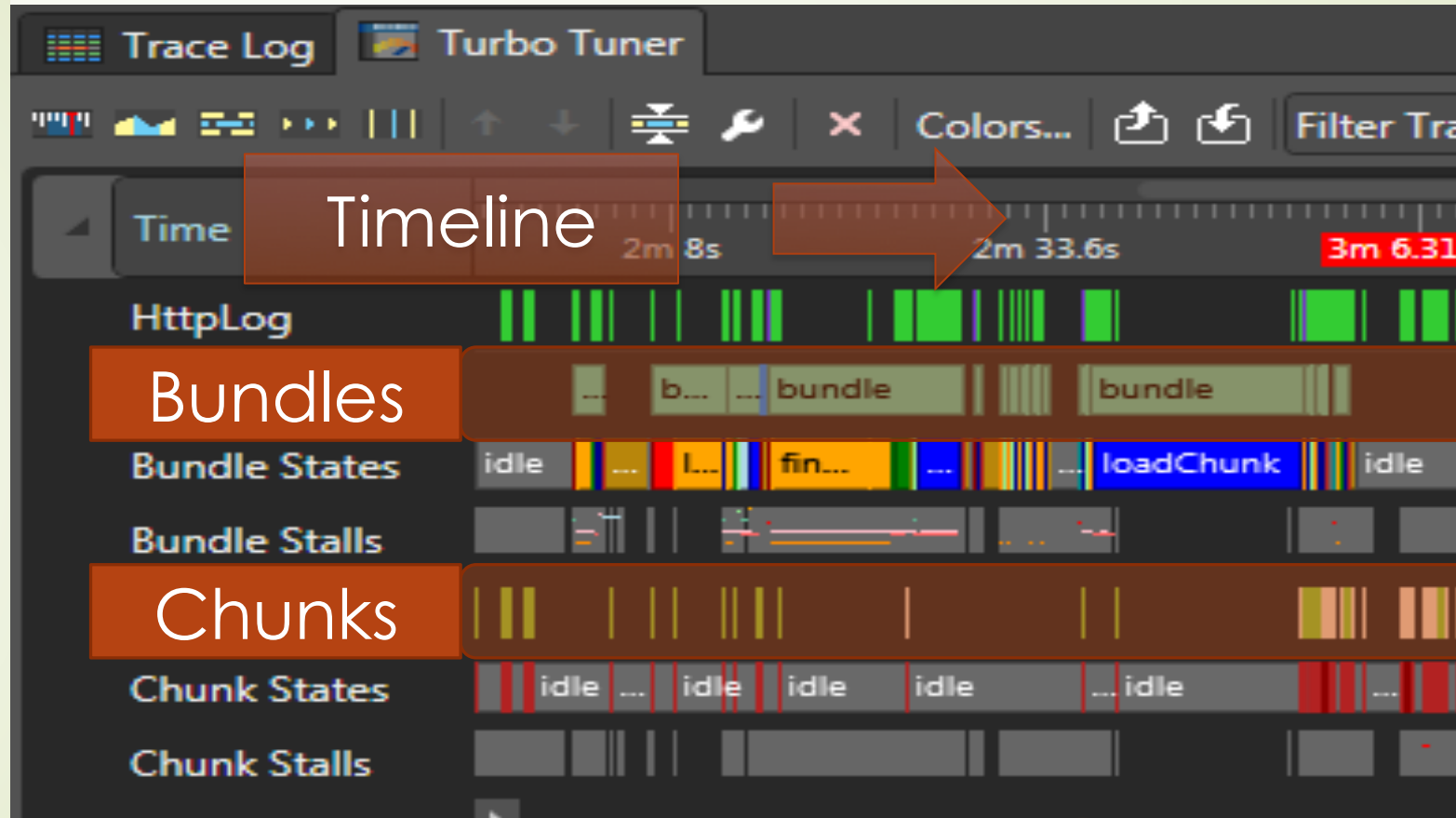
Id	Time	Pri	Channel	Message
1 54	1:05.27400940	☺	LTC	mark: Pause_LanguageSelect
2 56	1:06.85583750	☺	LTC	mark: LegalScreens
3 63	1:07.08605960	☺	LTC	mark: LegalScreens
4 65	1:12.83501840	☺	LTC	mark: NoStartTitle
5 74	2:00.14468260	☺	LTC	start: playMatch_to_practiceArena
6 82	2:06.40368660	☺	LTC	stop: playMatch_to_practiceArena
7 83	2:06.40368850	☺	LTC	start: practice_arena
8 190	2:36.00628670	☺	LTC	stop: practice_arena
9 191	2:36.00629030	☺	LTC	start: Pause_PressStart
10 192	2:37.67105530	☺	LTC	stop: Pause_PressStart
11 193	2:37.67105780	☺	LTC	start: press_start_to_be
12 194	2:37.78294850	☺	LTC	stop: press_start_to_be
13 202	3:45.73428530	☺	LTC	start: playMatch_to_practiceArena
14 211	3:50.50539790	☺	LTC	stop: playMatch_to_practiceArena
15 212	3:50.50539950	☺	LTC	start: practice_arena
16 317	4:21.37343520	☺	LTC	stop: practice_arena
17 318	4:21.37343800	☺	LTC	start: Pause_PressStart
18 319	4:26.07145100	☺	LTC	stop: Pause_PressStart
19 320	4:26.07145300	☺	LTC	start: press_start_to_be

Level 1

Level 2

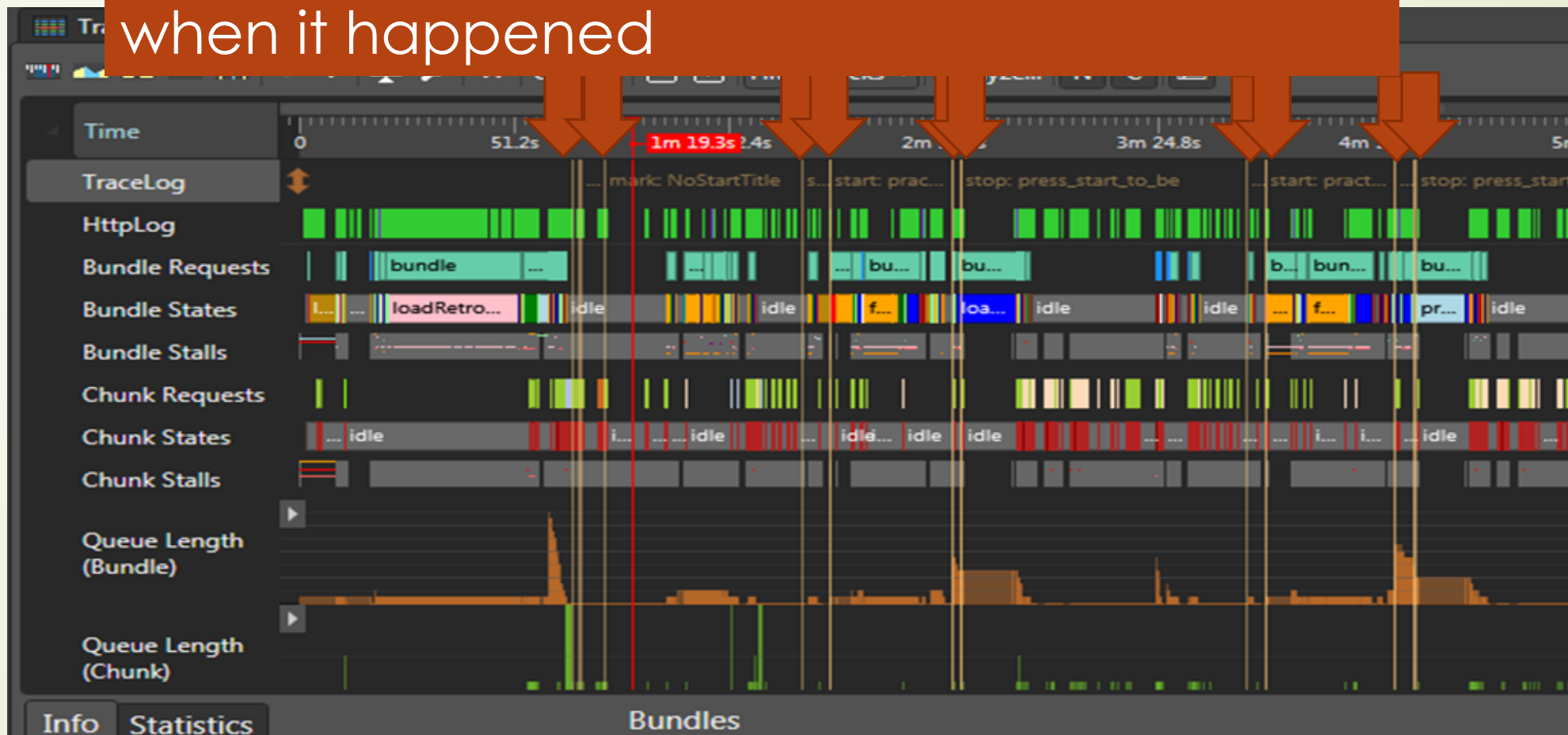
# IO Load profiler (Turbo Tuner)

- Bundle is a group of files that have to be loaded to move the game to the next level or sub level.
- Chunks are blocks of data that are steamed in. Like movies or music or terrain in open world games.

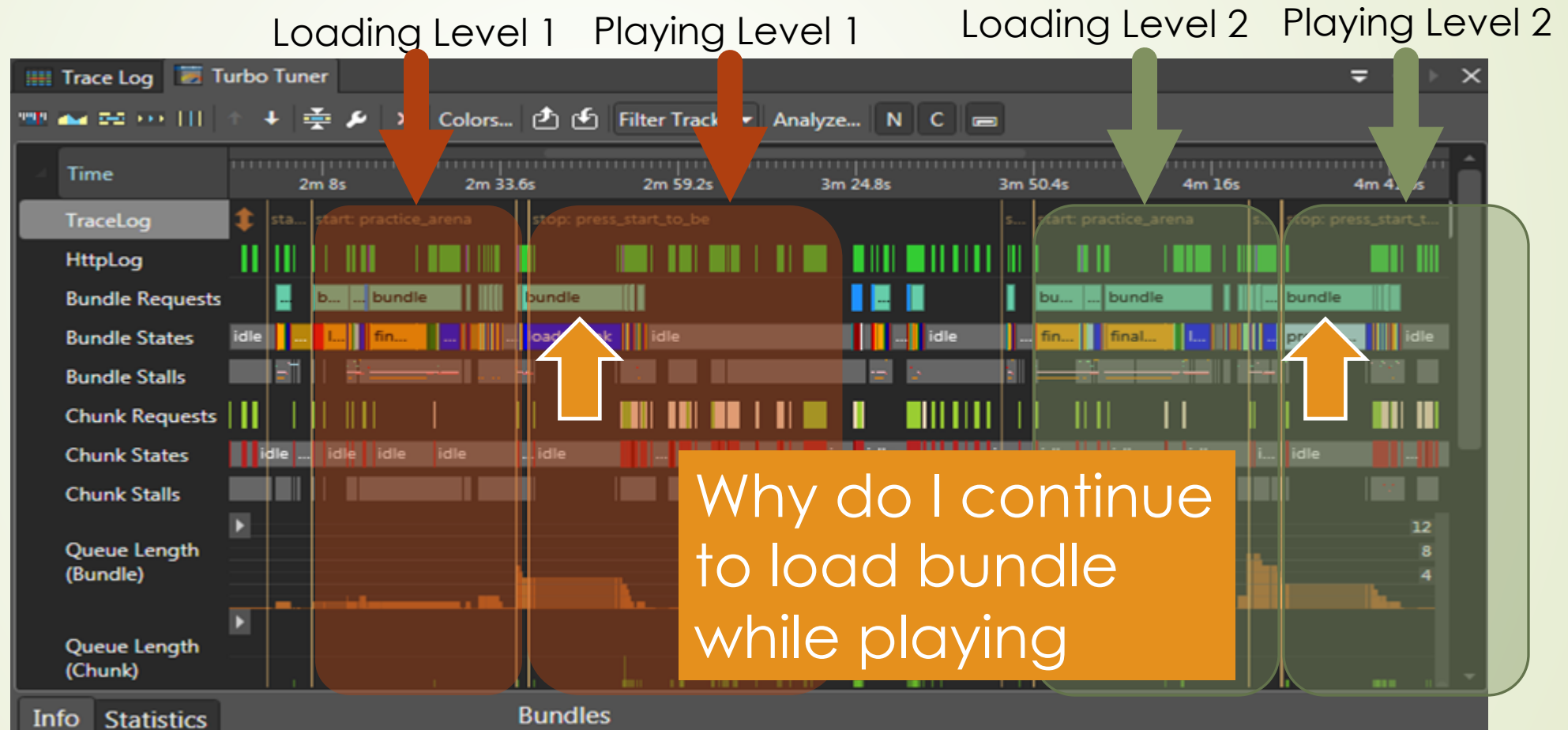


# IO Load profiler (Turbo Tuner)

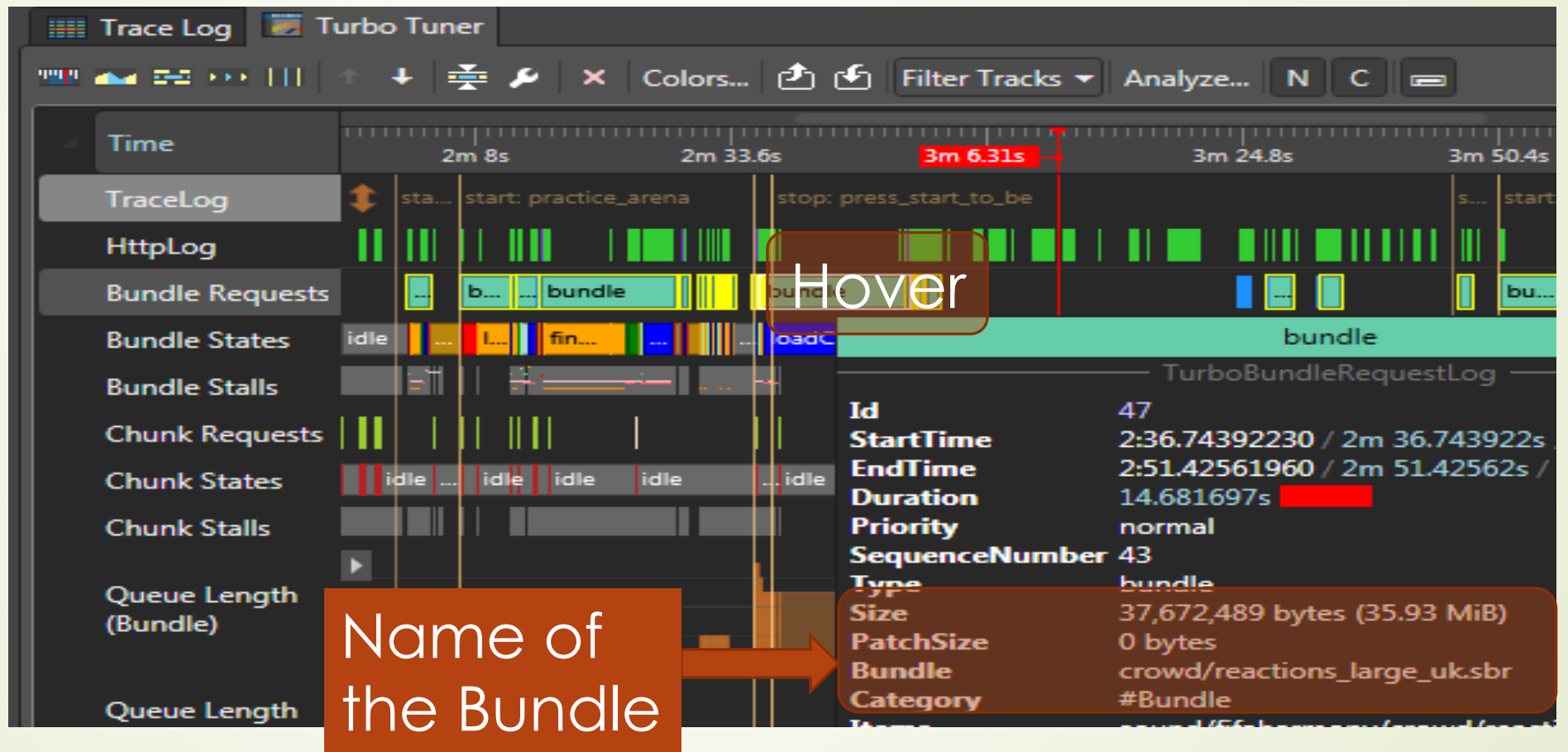
Each Printf on the selected channel gets an event line so you can understand when it happened



# IO Load profiler (Turbo Tuner)

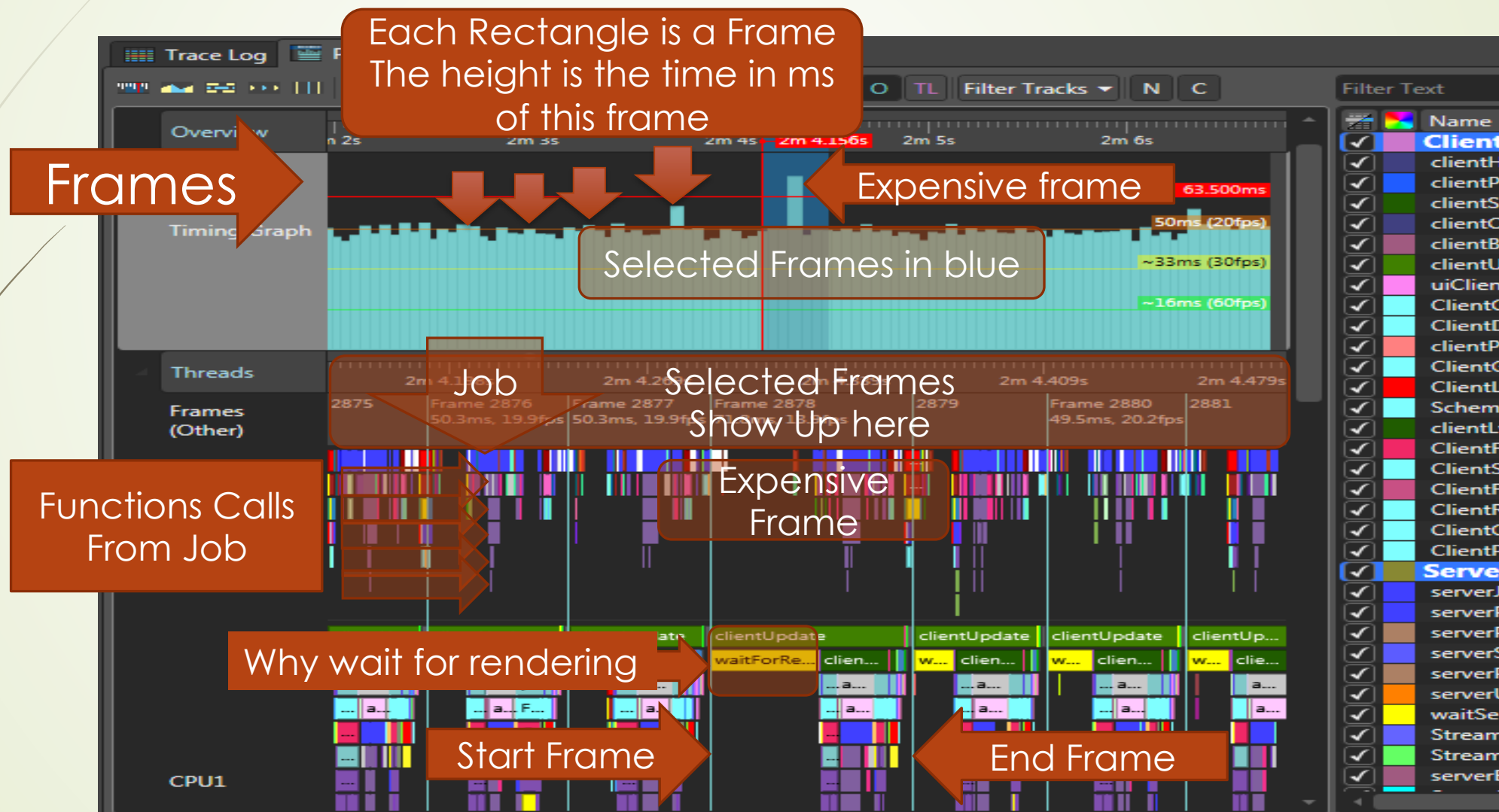


# IO Load profiler (Turbo Tuner)



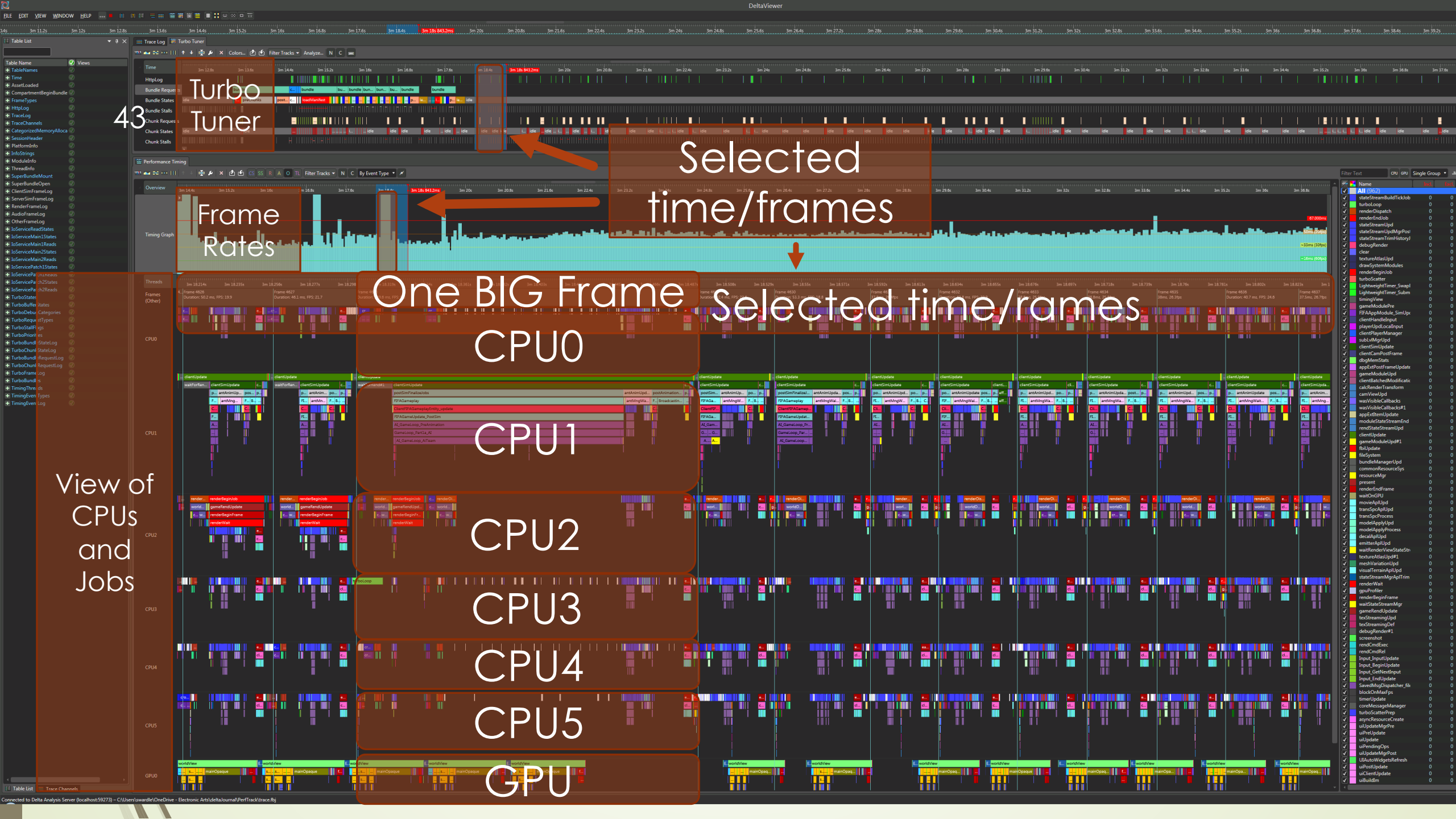


# Frame rate and Job thread profiler (Performance Timer)



# Loading profiler + Frame rate profiler

- We can combine views
- Why?
- Loading is about more than disk performance
  - Decompression
  - Stamping one texture on with a font
  - Recompressing and loading into VRAM
  - Loading is often limited by CPU



43

Turbo Tuner

Selected time/frames

Frame Rates

One BIG Frame  
CPU0

Selected time/frames

View of CPUs and Jobs

CPU1

CPU2

CPU3

CPU4

CPU5

GPU

Name	Count	Time
stateStreamBuildTickJob	0	0.0
turboLoop	0	0.0
renderDispatch	0	0.0
renderEndJob	0	0.0
stateStreamUpd	0	0.0
stateStreamUpdMgrPost	0	0.0
stateStreamTrmHistory	0	0.0
debugRender	0	0.0
clear	0	0.0
textureAtlasUpd	0	0.0
drawSystemModules	0	0.0
renderBeginJob	0	0.0
turboScatter	0	0.0
textureAtlasUpd	0	0.0
drawSystemModules	0	0.0
renderBeginJob	0	0.0
turboScatter	0	0.0
LightweightTimer_Swap	0	0.0
LightweightTimer_Subm	0	0.0
gameModulePre	0	0.0
FIFAAppModule_SimUpd	0	0.0
clientHandingUpd	0	0.0
playerLocUpd	0	0.0
clientPlayerManager	0	0.0
subLvlMgrUpd	0	0.0
clientSimUpd	0	0.0
clientCamPostFrame	0	0.0
dbgMemStats	0	0.0
appExtPostFrameUpd	0	0.0
gameModuleUpd	0	0.0
clientBatchModificatio	0	0.0
calcRenderTransform	0	0.0
camViewUpd	0	0.0
waitVisibleCallouts1	0	0.0
waitVisibleCallouts1	0	0.0
appExtUpd	0	0.0
moduleStateStreamEnd	0	0.0
renderStreamUpd	0	0.0
clientUpdate	0	0.0
gameModuleUpd#1	0	0.0
fbUpd	0	0.0
fileSystem	0	0.0
bundleManagerUpd	0	0.0
commonResourceSys	0	0.0
resourceMgr	0	0.0
present	0	0.0
renderEndFrame	0	0.0
waitOnGPU	0	0.0
movieApplUpd	0	0.0
transpApplUpd	0	0.0
transpProcess	0	0.0
modelApplUpd	0	0.0
modelApplProcess	0	0.0
decapApplUpd	0	0.0
emitterApplUpd	0	0.0
waitRenderViewStateStr	0	0.0
textureAtlasUpd#1	0	0.0
meshVertexUpd	0	0.0
visualTerrainApplUpd	0	0.0
stateStreamMgrApplTrm	0	0.0
renderWait	0	0.0
gpuProfiler	0	0.0
renderBeginFrame	0	0.0
waitStateStreamMgr	0	0.0
gameEndUpd	0	0.0
textStreamUpd	0	0.0
textStreamingDef	0	0.0
debugRender#1	0	0.0
screenShot	0	0.0
renOCmdExec	0	0.0
renOCmdRel	0	0.0
Input_InputUpd	0	0.0
Input_BeginUpd	0	0.0
Input_CmdExecPost	0	0.0
Input_EndUpd	0	0.0
SaveMsgDispatcher_fla	0	0.0
blockCmdMgrPs	0	0.0
timerUpd	0	0.0
coreMessageManager	0	0.0
turboScatterPrep	0	0.0
appResourceCreate	0	0.0
uiUpdateMgrPre	0	0.0
uiPreUpd	0	0.0
uiUpdate	0	0.0
uiRenderingOps	0	0.0
uiUpdateMgrPost	0	0.0
UIAutoWidgetsRefresh	0	0.0
uiPostUpd	0	0.0
uiClientUpd	0	0.0
uiBuildIn	0	0.0

# Use Memory Investigator for leaks

44

## ► Finding memory leaks

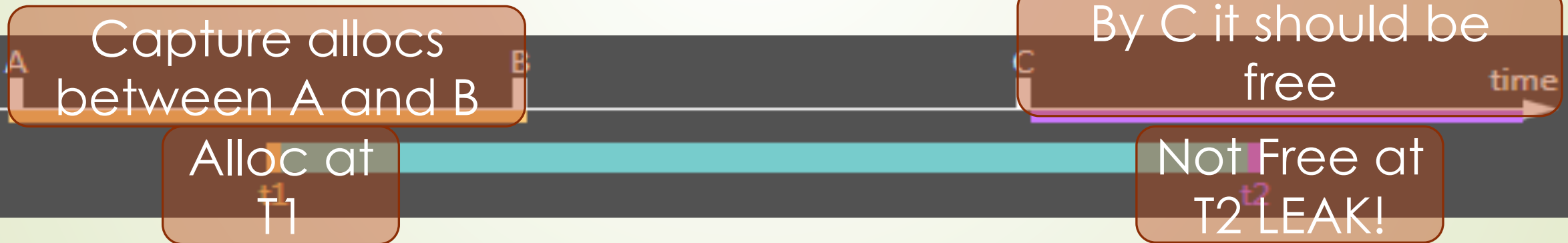
► How to find memory leaks in most games. Find:

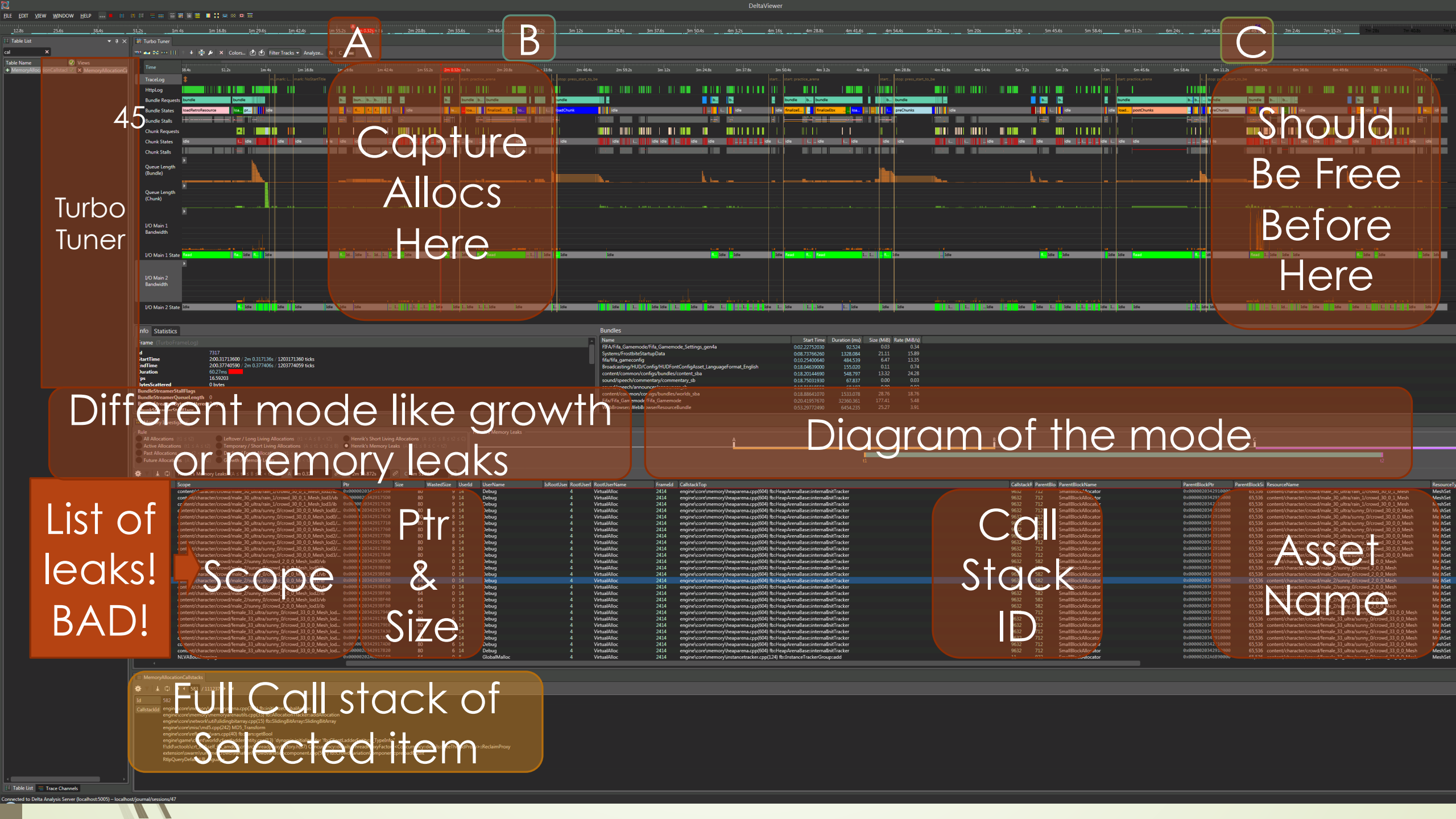
A. Start of loading 1<sup>st</sup> level

B. End of loading 1<sup>st</sup> level

C. End of Loading 2<sup>nd</sup> level

► (Growing objects look like leaks but often after a few levels this goes away, wish we had realloc)





45

Turbo Tuner

Capture Allocs Here

Should Be Free Before Here

Different mode like growth or memory leaks

Diagram of the mode

List of leaks! BAD!

Ptr & Size

Call Stack ID

Asset Name

Full Call stack of Selected item

# Memory Categorization

46

These times are often found using turbo tuner

Scrub to another time

The image shows two side-by-side screenshots of the Visual Studio Memory Investigator tool. The left screenshot is labeled 'Before' and the right is labeled 'After'. Both screenshots show the 'Memory Investigator' window with the 'Category Tree' view selected. The 'Before' screenshot shows a CPU allocation of 1,682,197 and a CPU total of 2.61 GiB. The 'After' screenshot shows a CPU allocation of 2,201,444 and a CPU total of 3.33 GiB. The 'CPU Wasted' column is only visible in the 'After' screenshot, showing 4.16 MiB. The time shown in the top right of the 'After' screenshot is 4m 52.16s, which is highlighted in red. The time shown in the top right of the 'Before' screenshot is 1m 54.24s, which is also highlighted in red. The 'Memory Investigator' window has a toolbar with 'Trace Log', 'Turbo Tuner', and 'Memory Investigator' buttons. The 'Category Tree' view has a toolbar with 'Import...', 'Reload', 'Category Tree', 'Rule Tree', and 'Update' buttons. The 'Category Tree' view shows a tree structure with categories: 'DefaultCategorization', '1\_XLargeAllocs', '2\_LargeAllocs2M', '3\_MidAllocs64k', and '4\_SmallAllocs512bytes'. The 'CPU Allocs' and 'CPU Total' columns are highlighted in orange in both screenshots.

Category Tree	CPU Allocs	CPU Total	CPU Wasted
DefaultCategorization	1,682,197	2.61 GiB	
1_XLargeAllocs	145	1.72 GiB	659
2_LargeAllocs2M	1,868	455.86 MiB	12.38 KiB
3_MidAllocs64k	62,228	378.42 MiB	310.52 KiB
4_SmallAllocs512bytes	1,617,956	77.25 MiB	3.85 MiB

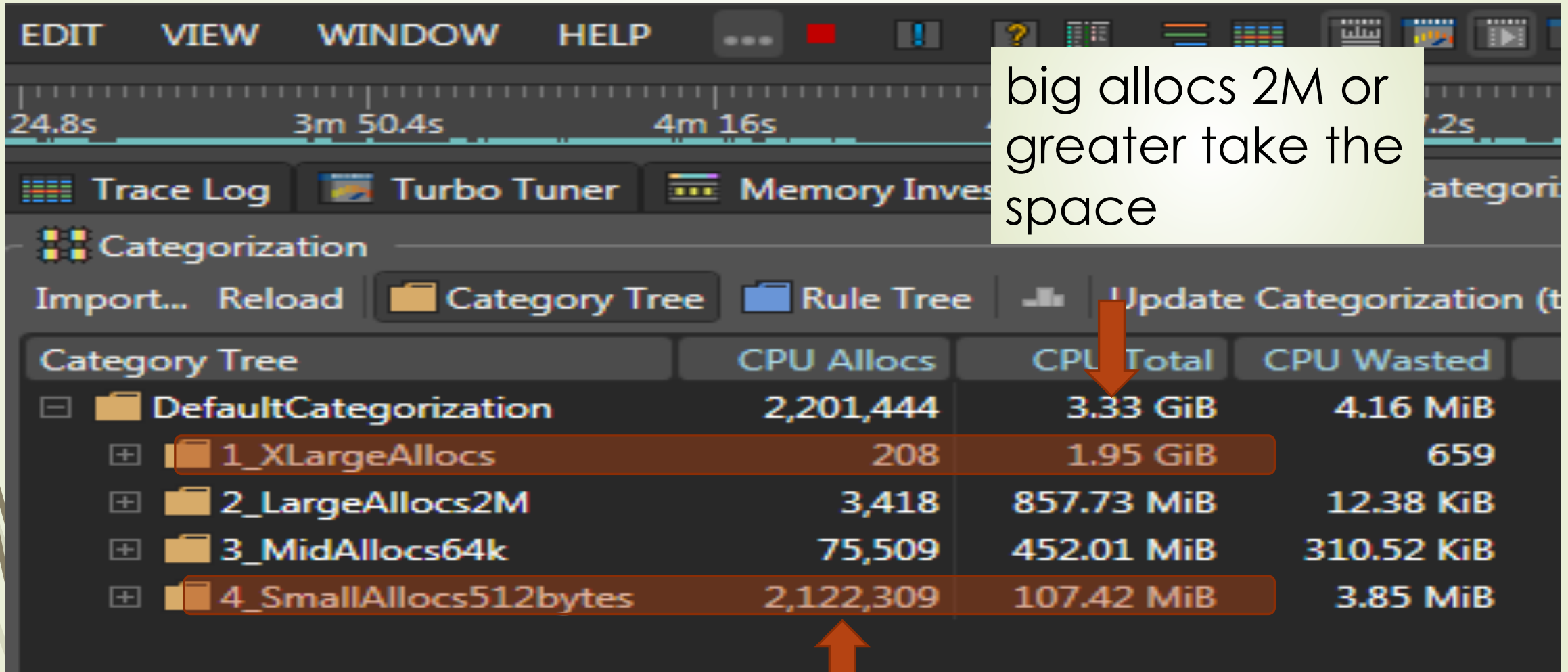
Category Tree	CPU Allocs	CPU Total	CPU Wasted
DefaultCategorization	2,201,444	3.33 GiB	4.16 MiB
1_XLargeAllocs	208	1.95 GiB	659
2_LargeAllocs2M	3,418	857.73 MiB	12.38 KiB
3_MidAllocs64k	75,509	452.01 MiB	310.52 KiB
4_SmallAllocs512bytes	2,122,309	107.42 MiB	3.85 MiB

Before

After

# Memory Categorization

47



The screenshot shows a software interface for memory categorization. At the top, there are menu items: EDIT, VIEW, WINDOW, HELP. Below the menu is a timeline showing 24.8s, 3m 50.4s, and 4m 16s. There are also buttons for Trace Log, Turbo Tuner, and Memory Invest. The main section is titled 'Categorization' and includes buttons for Import..., Reload, Category Tree, Rule Tree, and Update Categorization (t). Below this is a table with the following data:

Category Tree	CPU Allocs	CPU Total	CPU Wasted
[-] DefaultCategorization	2,201,444	3.33 GiB	4.16 MiB
[+] 1_XLargeAllocs	208	1.95 GiB	659
[+] 2_LargeAllocs2M	3,418	857.73 MiB	12.38 KiB
[+] 3_MidAllocs64k	75,509	452.01 MiB	310.52 KiB
[+] 4_SmallAllocs512bytes	2,122,309	107.42 MiB	3.85 MiB

big allocs 2M or greater take the space

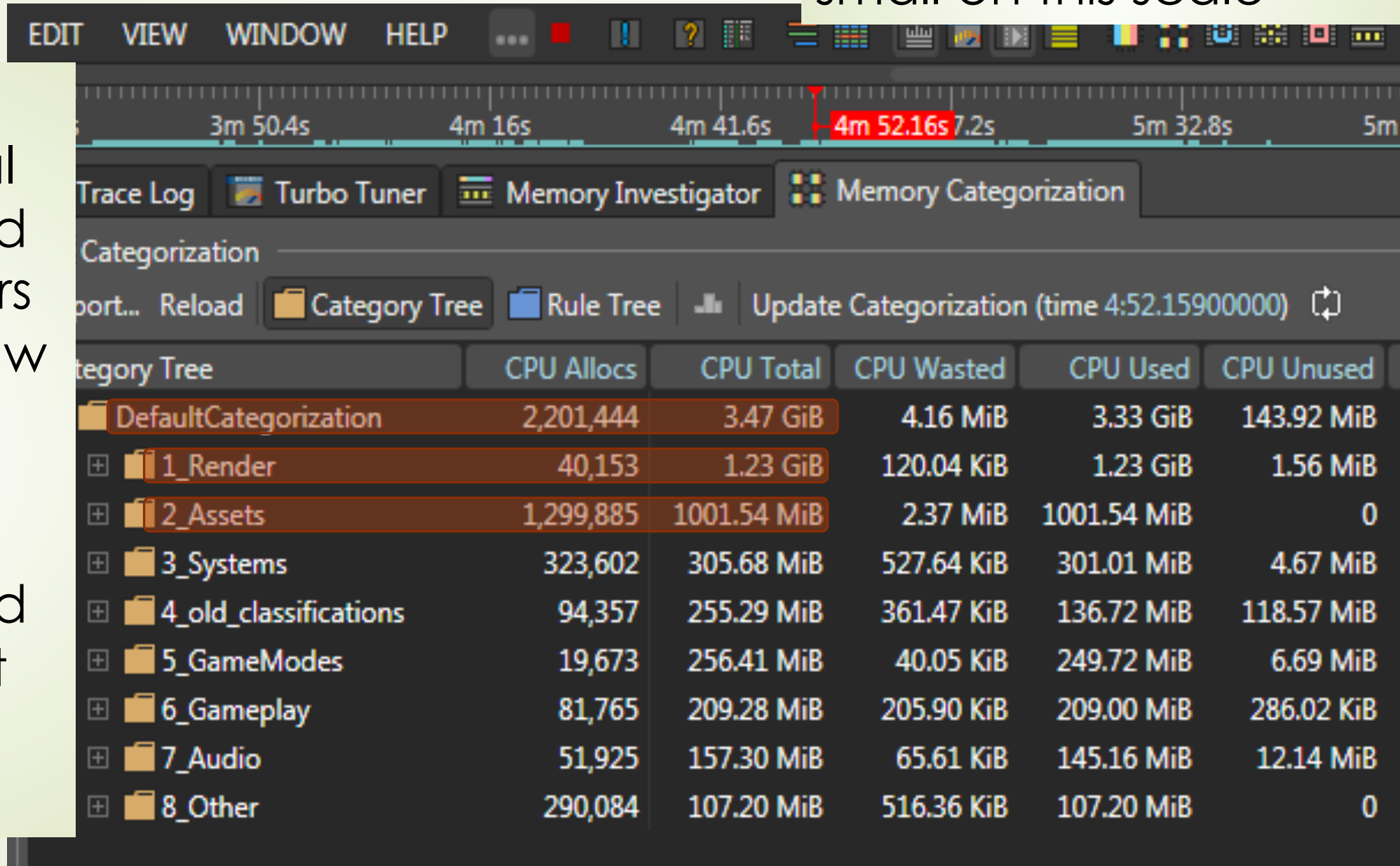
Lots of small allocs 512 bytes or smaller

# Memory Categorization

the code is 50MiB  
small on this scale

48

- Rendering (procedural textures and other buffers used to draw the scene)
- Content, meshes, textures and entities that tie these together



The screenshot shows a software interface for memory categorization. At the top, there is a menu bar with 'EDIT', 'VIEW', 'WINDOW', and 'HELP'. Below the menu bar is a timeline showing time intervals: 3m 50.4s, 4m 16s, 4m 41.6s, 4m 52.16s (highlighted in red), 7.2s, 5m 32.8s, and 5m. The main interface has tabs for 'Trace Log', 'Turbo Tuner', 'Memory Investigator', and 'Memory Categorization'. Below the tabs, there are buttons for 'Category Tree' and 'Rule Tree', and a refresh button. The main area displays a table with the following data:

Category Tree	CPU Allocs	CPU Total	CPU Wasted	CPU Used	CPU Unused
DefaultCategorization	2,201,444	3.47 GiB	4.16 MiB	3.33 GiB	143.92 MiB
+ 1_Render	40,153	1.23 GiB	120.04 KiB	1.23 GiB	1.56 MiB
+ 2_Assets	1,299,885	1001.54 MiB	2.37 MiB	1001.54 MiB	0
+ 3_Systems	323,602	305.68 MiB	527.64 KiB	301.01 MiB	4.67 MiB
+ 4_old_classifications	94,357	255.29 MiB	361.47 KiB	136.72 MiB	118.57 MiB
+ 5_GameModes	19,673	256.41 MiB	40.05 KiB	249.72 MiB	6.69 MiB
+ 6_Gameplay	81,765	209.28 MiB	205.90 KiB	209.00 MiB	286.02 KiB
+ 7_Audio	51,925	157.30 MiB	65.61 KiB	145.16 MiB	12.14 MiB
+ 8_Other	290,084	107.20 MiB	516.36 KiB	107.20 MiB	0



# Summary

- ▶ DeltaViewer
  - ▶ Has many views:
    - ▶ TTY Event Timing
    - ▶ IO and Load times
    - ▶ Jobs and threads
    - ▶ Memory changes
  - ▶ We have a lot of work to do to ship the game I am on 😊
    - ▶ (Good thing I have one year left)

# Summary

- EASTL and STL allocators
  - Hard to track
  - Use the “if you made it you own it rule”
  - Use the “this” pointer of allocator as a parameter for your allocators
- EASTLICA
  - Good at enforcing allocator use for a large group of SEs
  - Helped with type erasure problems in `stl::string` and other classes. `MyString` does not work with `YourString`

# Summary

- ▶ Games in general
  - ▶ Most memory is used by large allocation
  - ▶ Most memory is mostly content (meshes and textures) or rendering
  - ▶ There are a large number of small allocations.
  - ▶ Small block allocators, pool systems, slab allocators are a good idea
  - ▶ Stomp Allocator are great (Use memory map to find who stomped you...)

# Questions?

