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 ~= 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**)

Header | Allocated Block | footer

Debug name and sentinel stored in footer

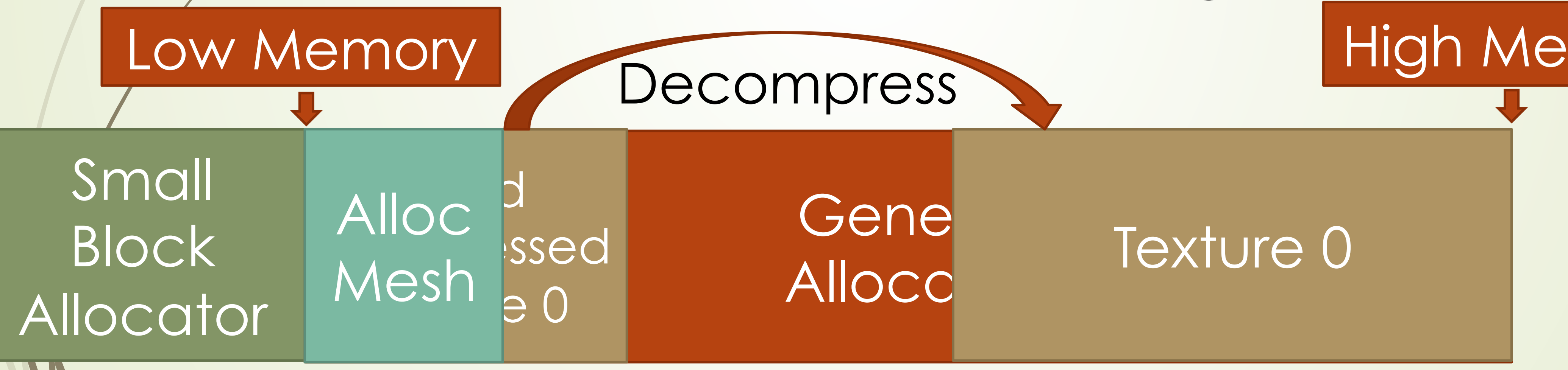
Header | Allocated Block | footer

Header | Allocated Block |

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

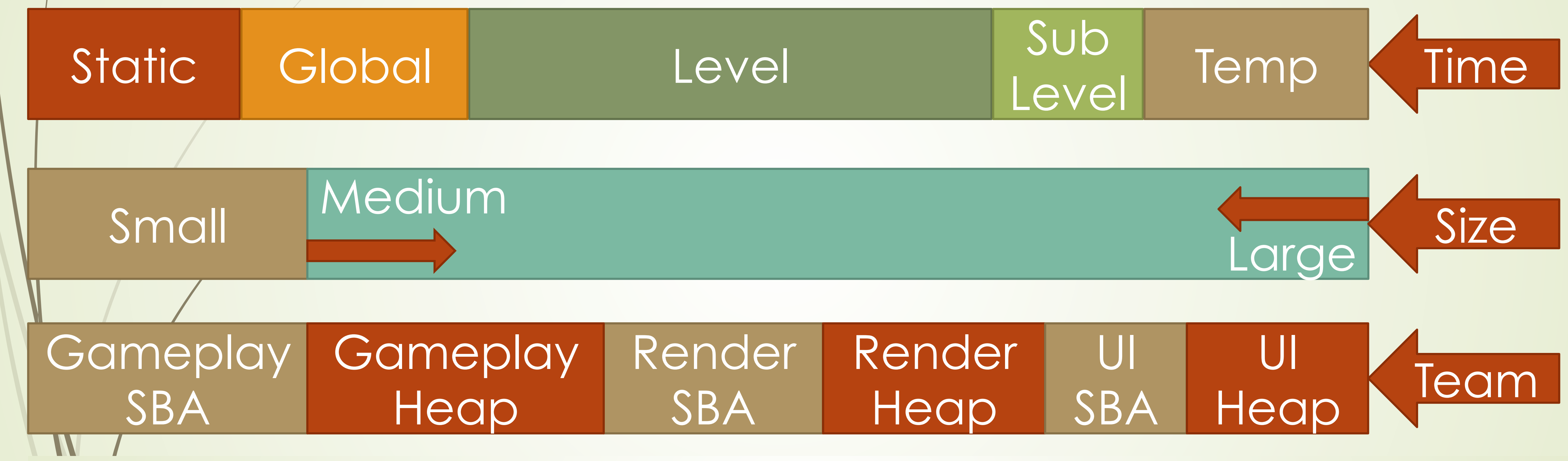
Polymorphic Allocator

```
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

R3	S2	mall	Render 0	SIM0	SIM1	Render 2	SIM3		Render 1

Categories are a way to tags 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

Only sentinel stored in footer

Tracking live allocations in a separate heap.

Memory Logging or tracing system

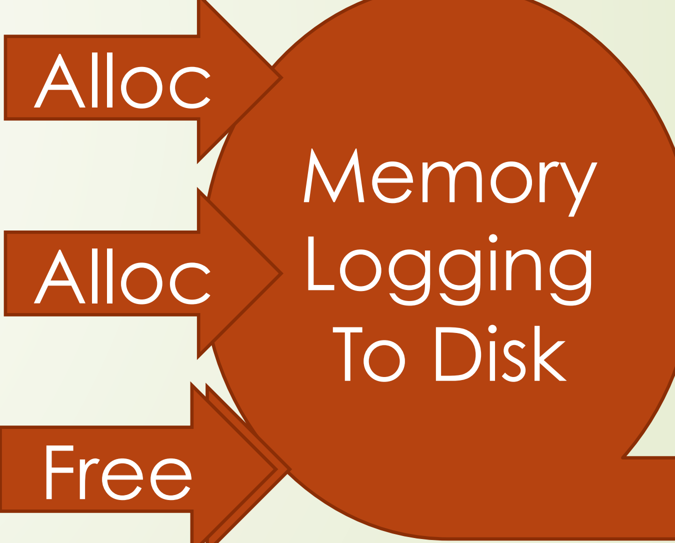
Normal Heap

H	Allocated Block	F
H	Allocated Block	F
H	Allocated Block	F

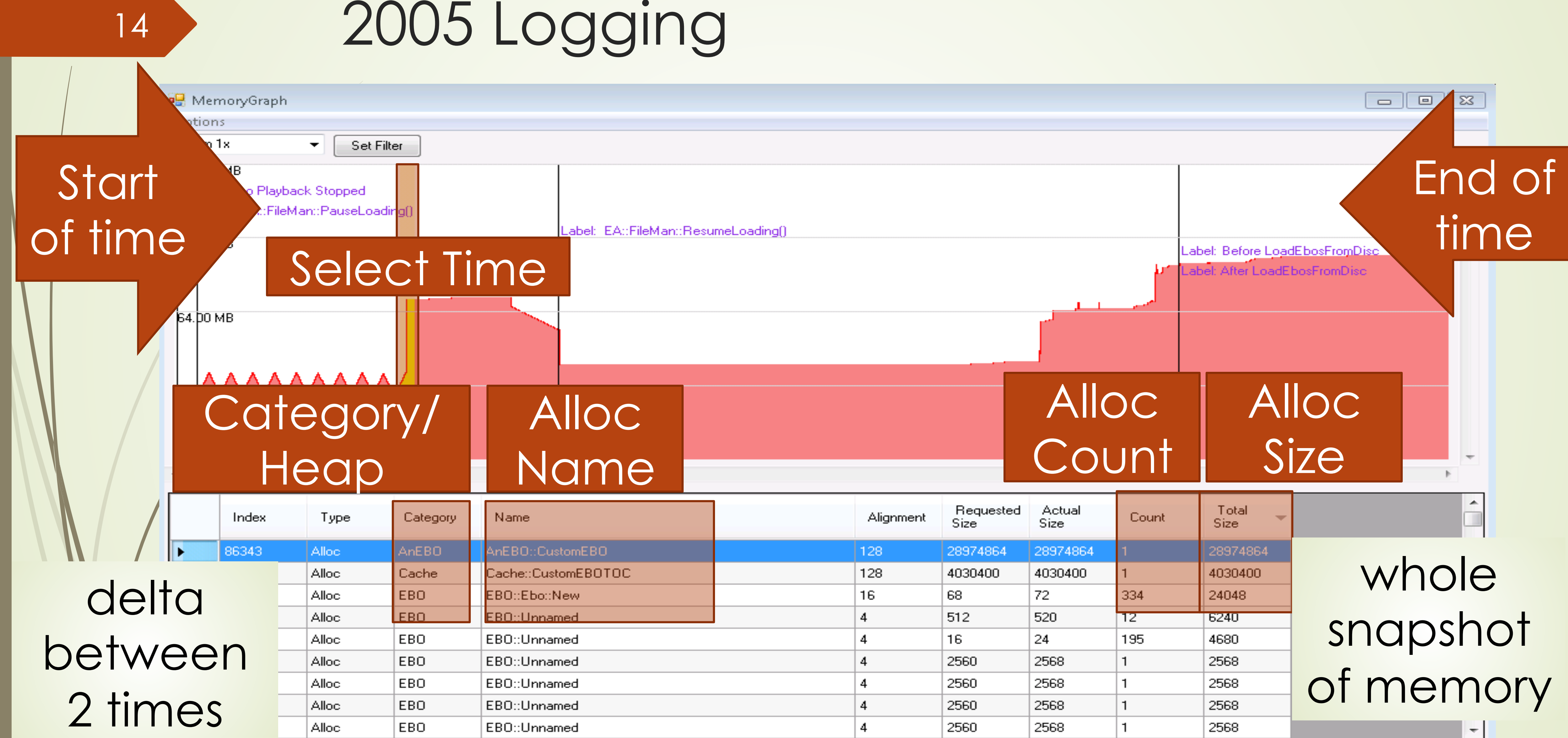
Debug Heap

Address	Size	Category:: Alloc Name
Address	Size	Category:: Alloc Name
Address	Size	Category:: Alloc Name

Hash Key

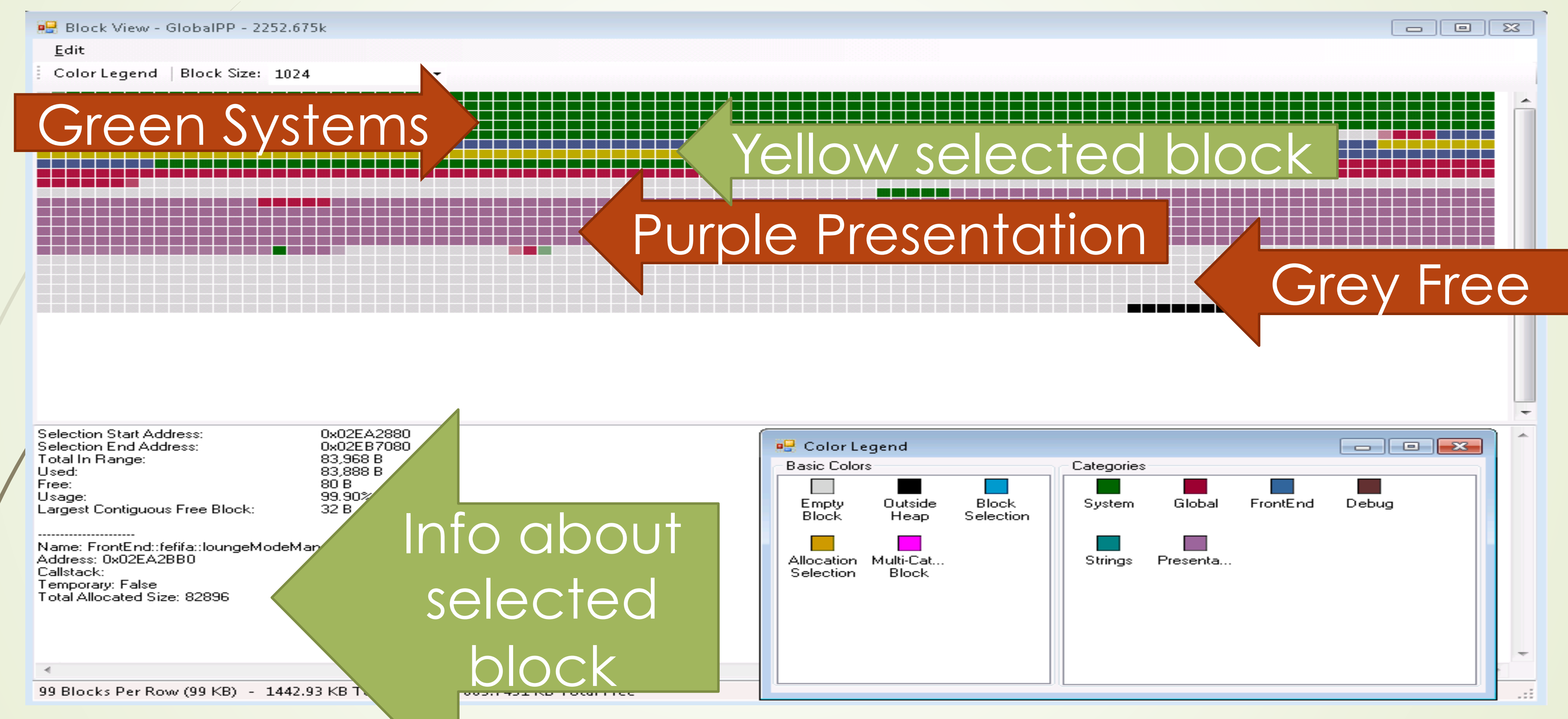


2005 Logging



Index	Type	Category	Name	Alignment	Requested Size	Actual Size	Count	Total Size
86343	Alloc	AnEBO	AnEBO::CustomEBO	128	28974864	28974864	1	28974864
	Alloc	Cache	Cache::CustomEBOTOC	128	4030400	4030400	1	4030400
	Alloc	EBO	EBO::Ebo::New	16	68	72	334	24048
	Alloc	EBO	EBO::Unnamed	4	512	520	12	6240
	Alloc	EBO	EBO::Unnamed	4	16	24	195	4680
	Alloc	EBO	EBO::Unnamed	4	2560	2568	1	2568
	Alloc	EBO	EBO::Unnamed	4	2560	2568	1	2568
	Alloc	EBO	EBO::Unnamed	4	2560	2568	1	2568
	Alloc	EBO	EBO::Unnamed	4	2560	2568	1	2568

2005 Arena Block View



2005 Stomp Allocator!!

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

Crash!

Page 4k
Read/W

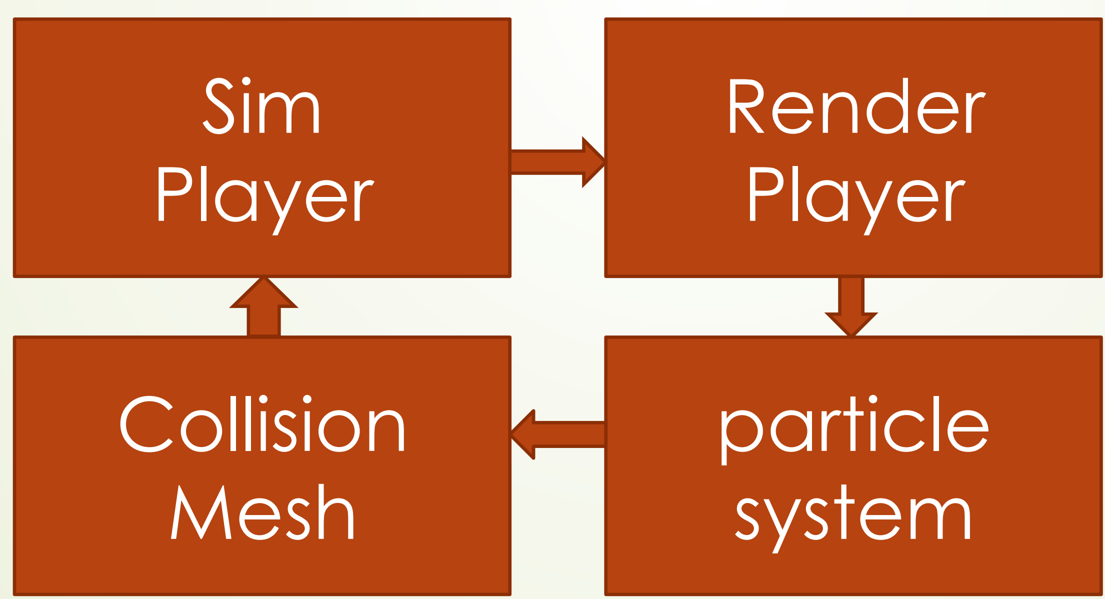
Alloc
512 bytes

Page 4KiB
Read Only
(Or not mapped)

No Crash but bad
Use sentinel? Or Flip

2005 Ref Counted Pointers

- 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…



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…

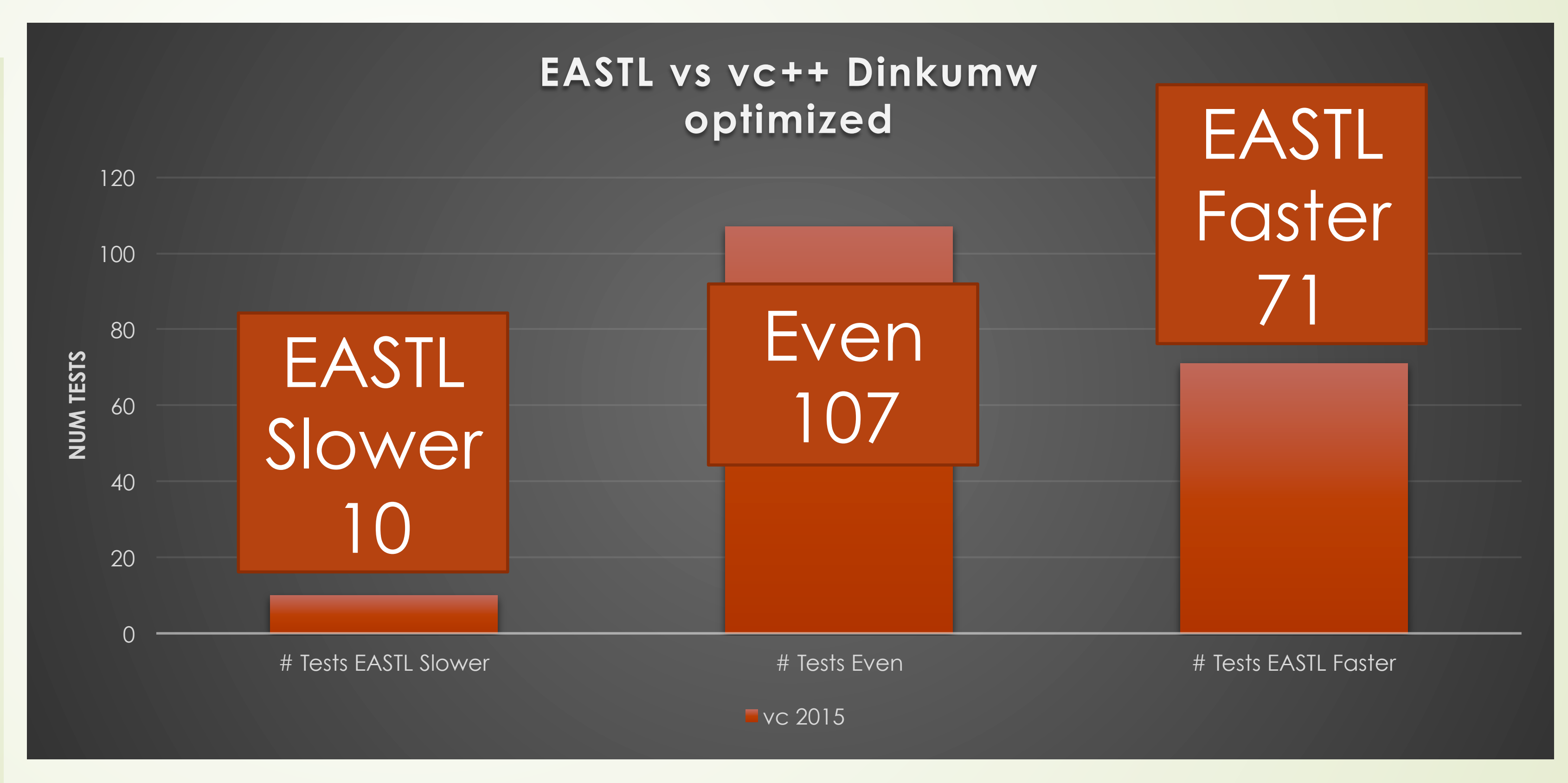
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2271.html

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

EASTL faster for optimized code

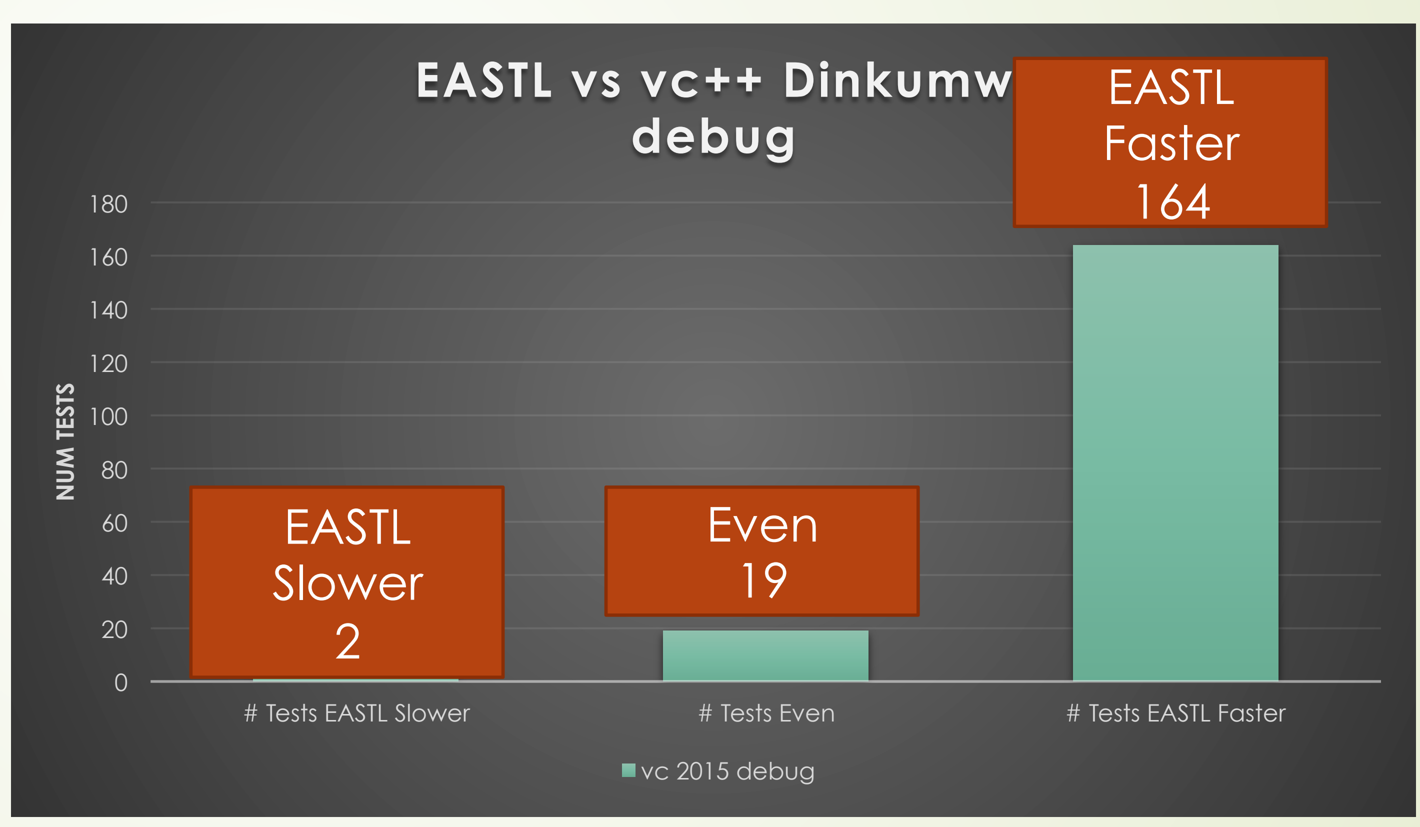
EASTL is often a little faster. In 71 out of 188 tests.

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



EASTL MUCH faster for debug code

The same 188 tests complied 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

- EASTL's allocator were painful to track every object

- You need to make a new type

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

- Then pass in a defaulted parameter

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

2005 EASTL Memory tracking problems

- 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 = GetRendAllocator();
vec.get_allocator().set_allocator(alloc);
```

2005 EASTL Memory tracking problems

- 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

- We also ran into type erasure problems

```
typedef vector<int, EASTLICoreAllocator> 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

- 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

- 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

- 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

- 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

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

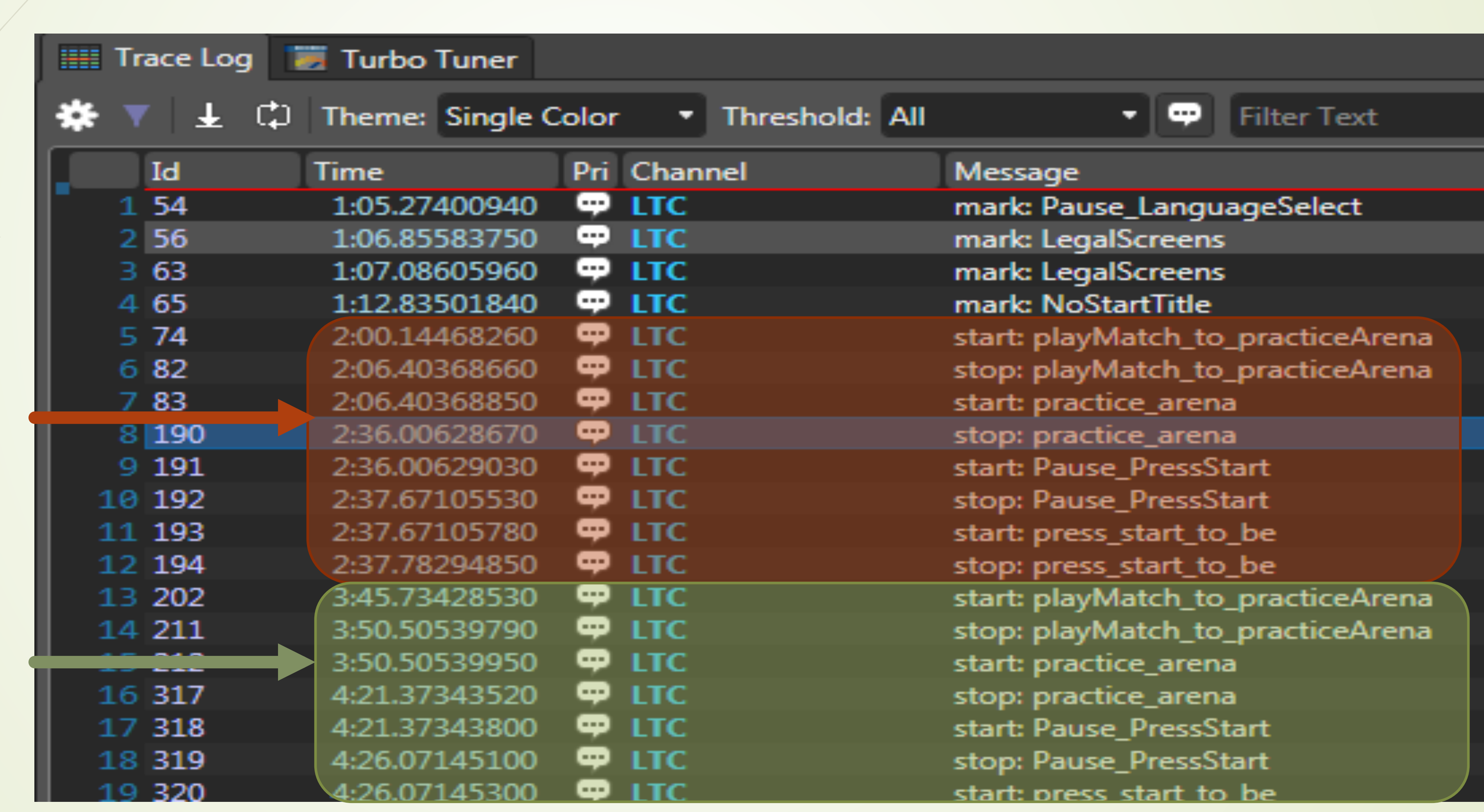
- History lesson over! Let's look at today's tools!

- 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)

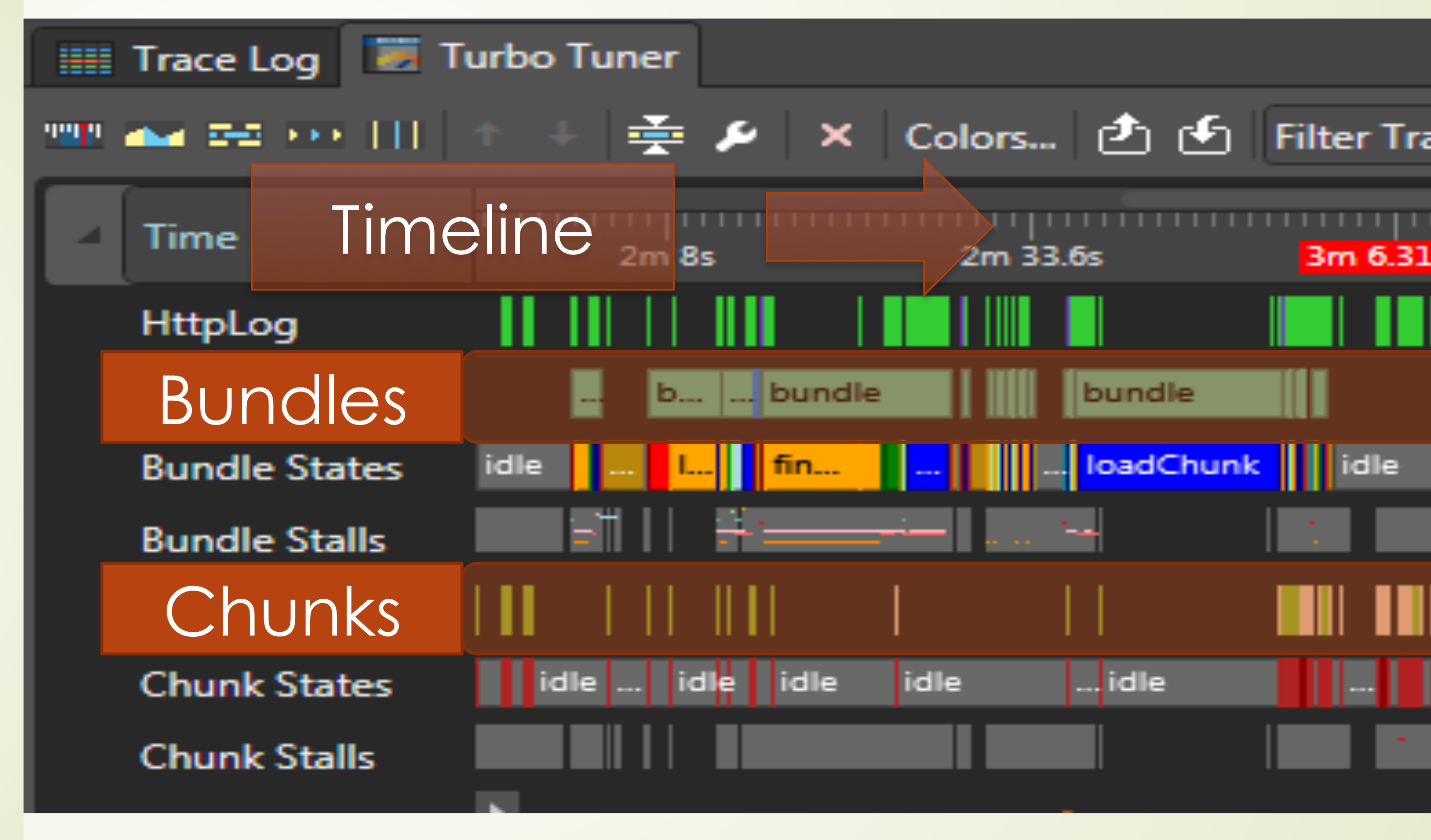


Level 1

Level 2

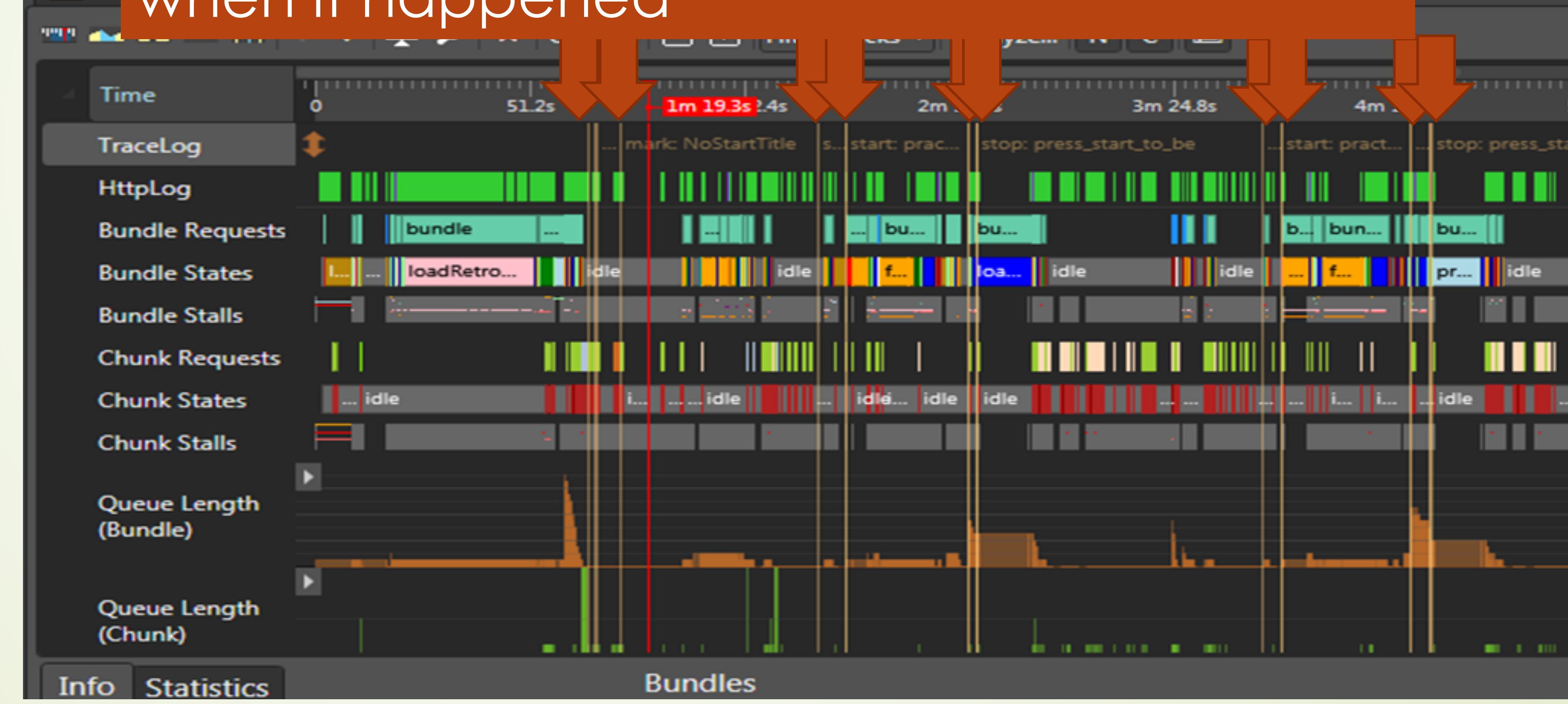
IO Load profiler (Turbo Tuner)

- 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.

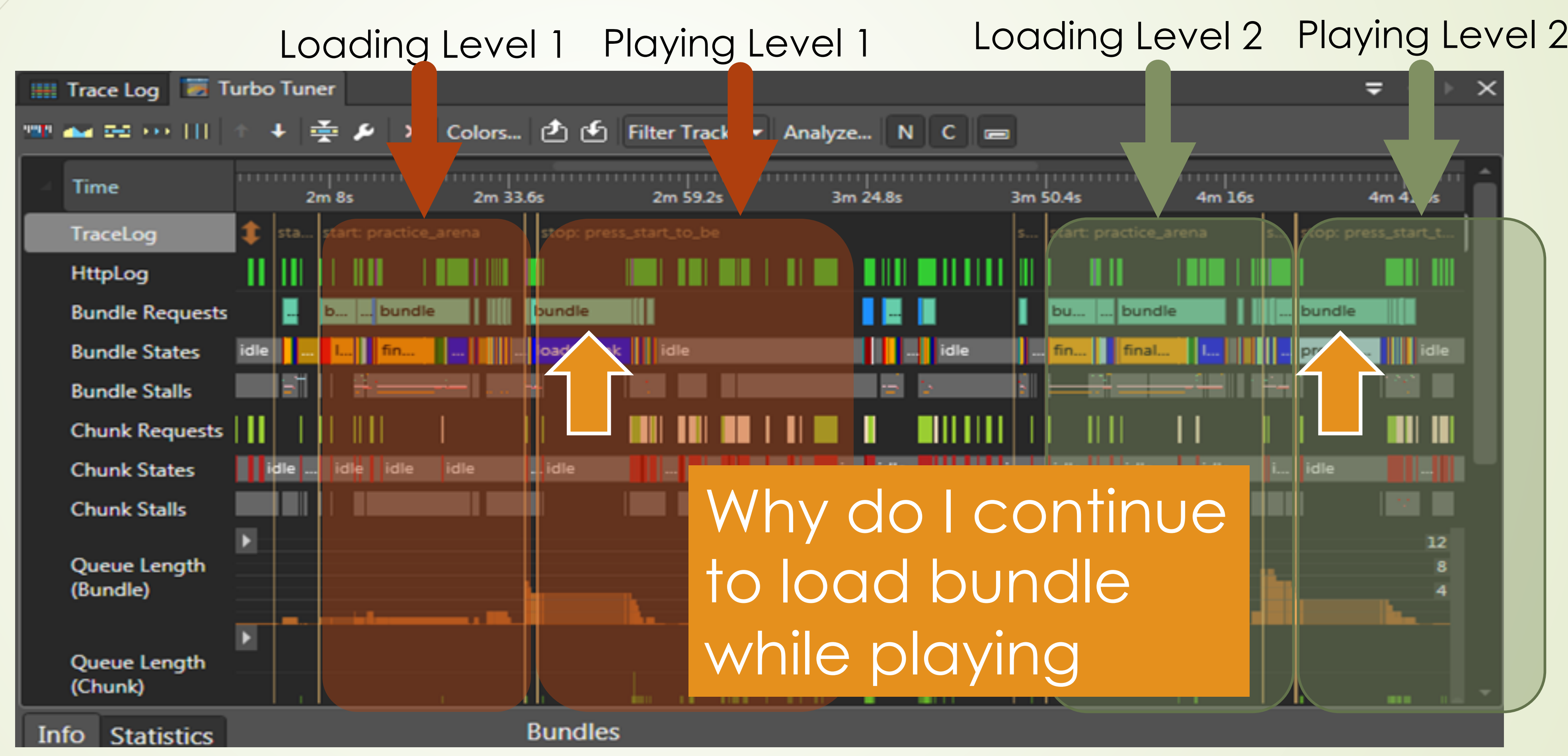


IO Load profiler (Turbo Tuner)

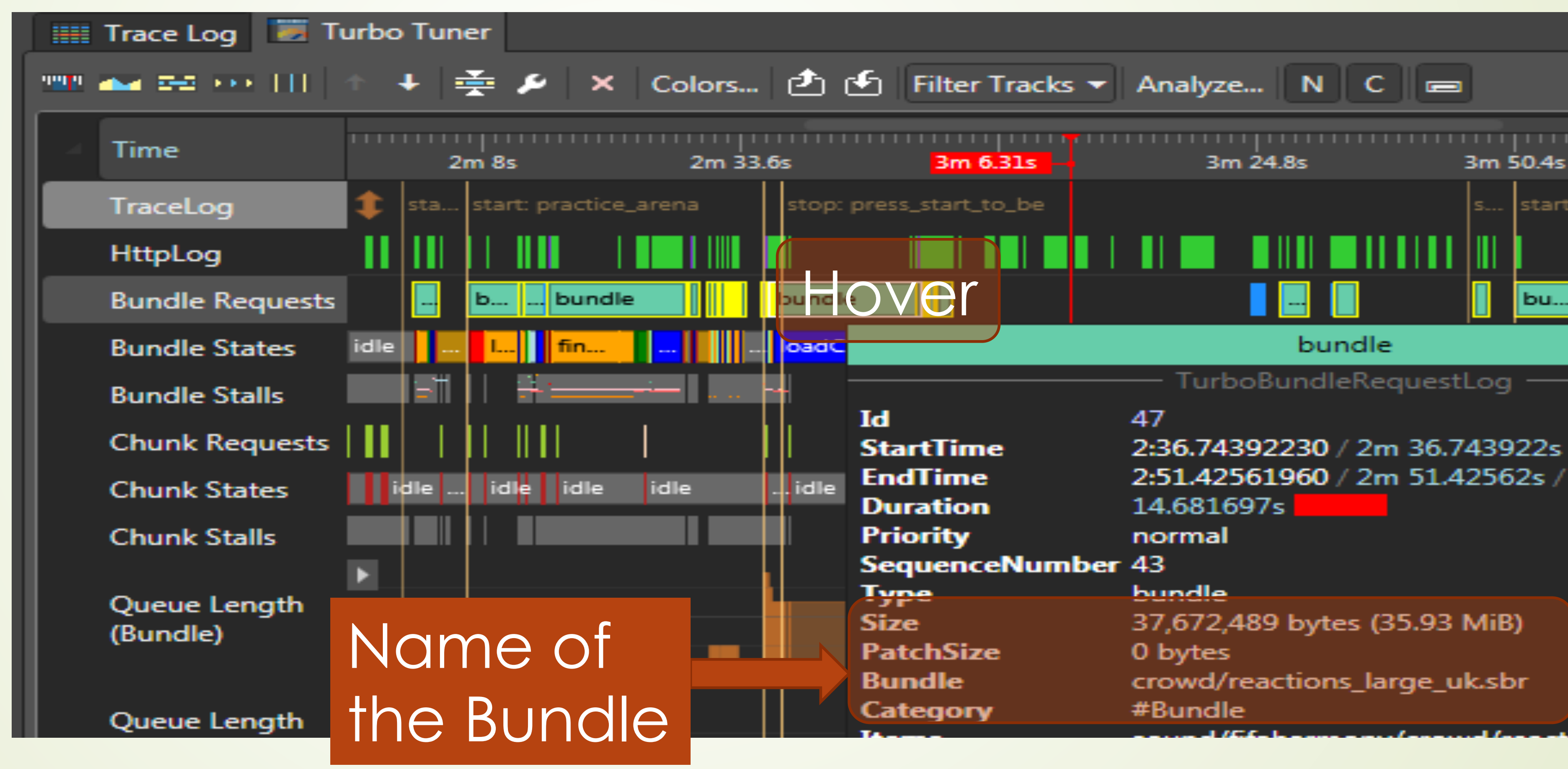
Each Printf on the selected channel gets an event line so you can undersand when it happened



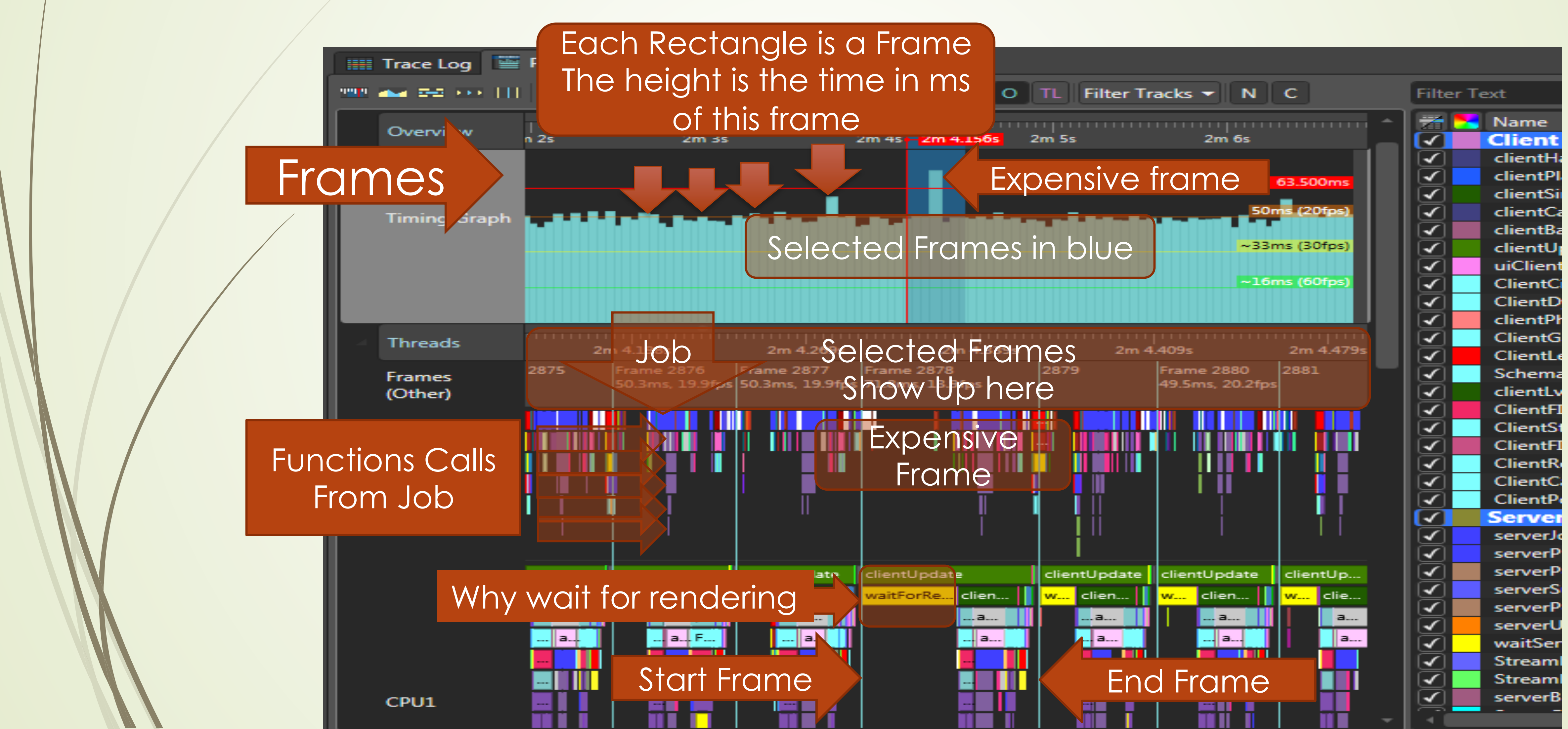
IO Load profiler (Turbo Tuner)



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 then disk performance
  - Decompression
  - Stamping one texture on with a font
  - Recompressing and loading into VRAM
  - Loading is often limited by CPU

DeltaViewer

Turbo Tuner

Selected time/frames

Frame Rates

One BIG Frame

Selected time/rames

CPU0

CPU1

CPU2

CPU3

CPU4

CPU5

GPU

View of CPUs and Jobs

Use Memory Investigator for leaks

- Finding memory leaks
  - How to find memory leaks in most games. Find:
    A. Start of loading 1st level
    B. End of loading 1st level
    C. End of Loading 2nd level
  - (Growing objects look like leaks but often after a few levels this goes away, wish we had realloc)

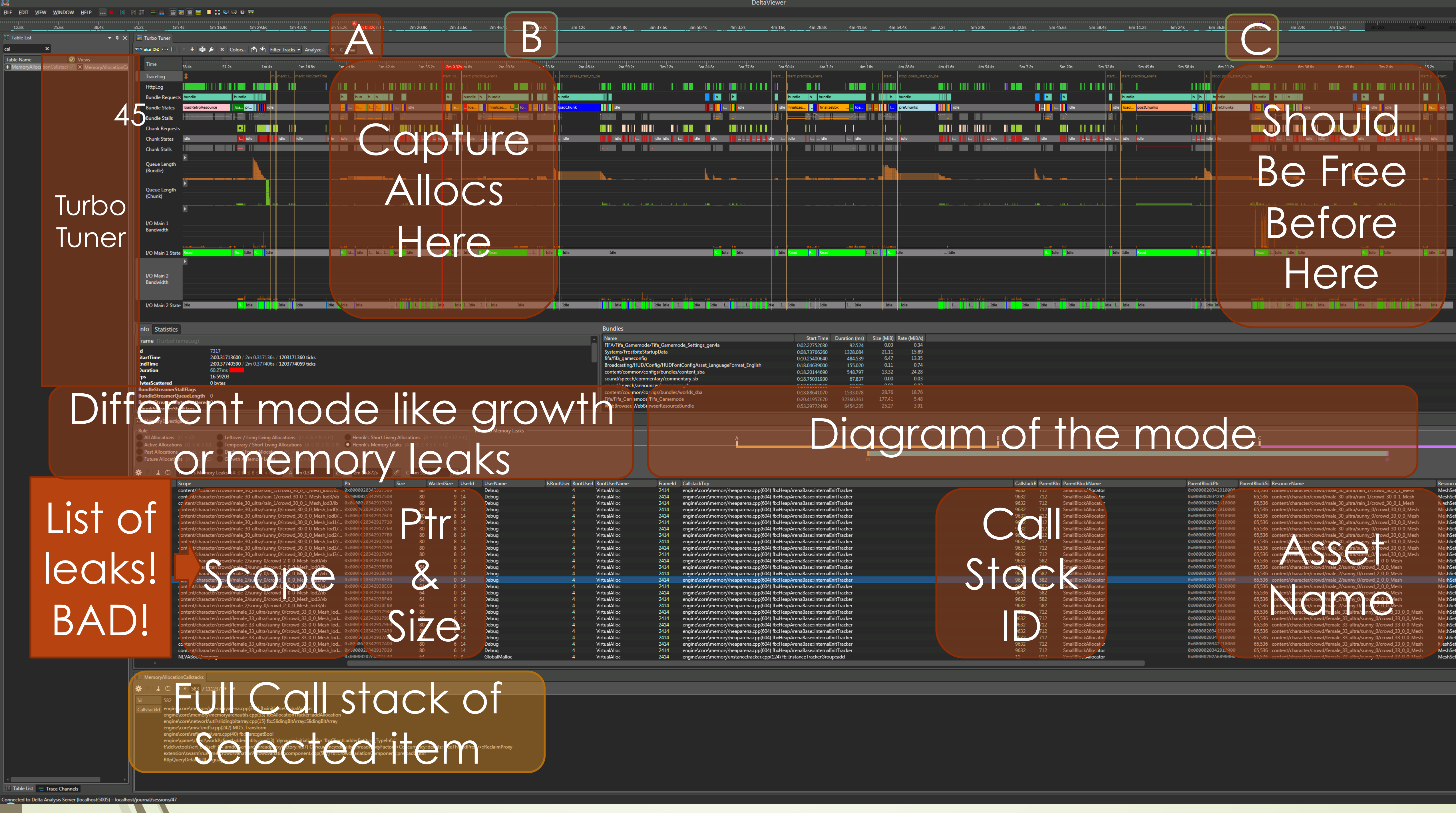
Capture allocs between A and B

By C it should be free

Alloc at T1

Not Free at T2 LEAK!

A B C time t1 t2

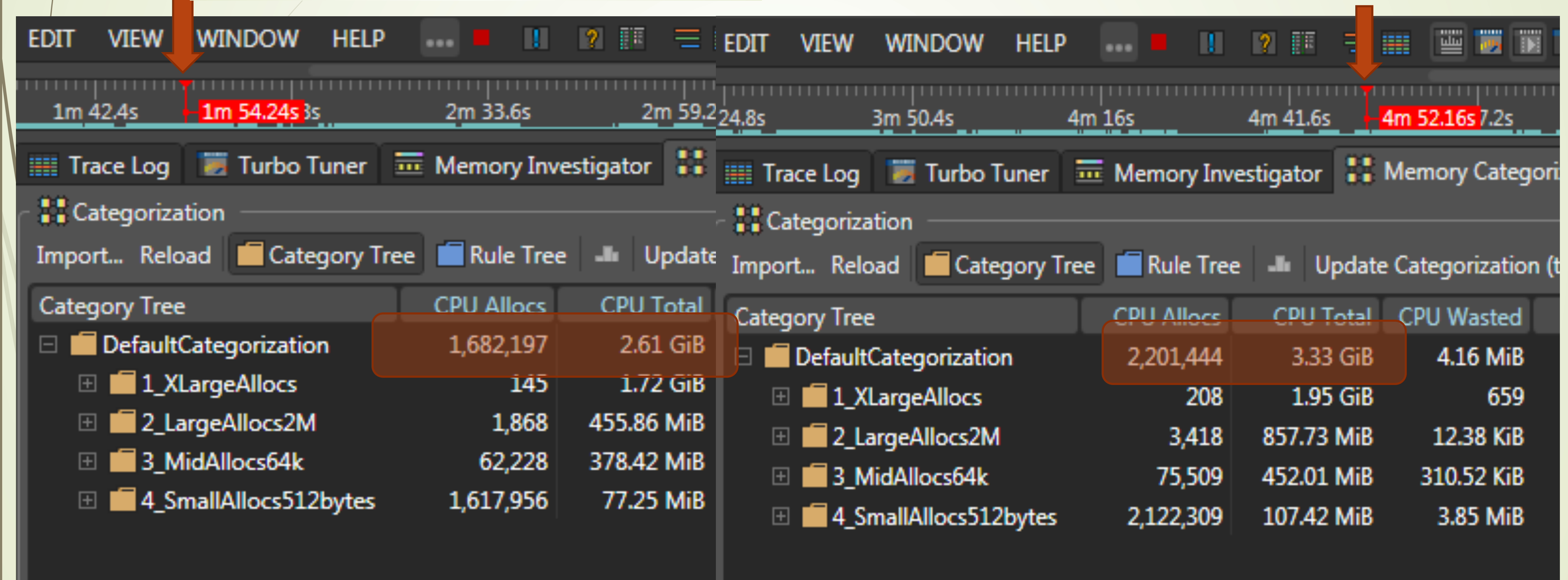


A
B
C
45
Turbo Tuner
Capture Allocs Here
Should Be Free Before Here
Different mode like growth or memory leaks
Diagram of the mode
List of leaks! BAD!
Scope
Ptr & Size
Call Stack ID
Asset Name
Full Call stack of Selected item

Memory Categorization

These times are often found using turbo tuner

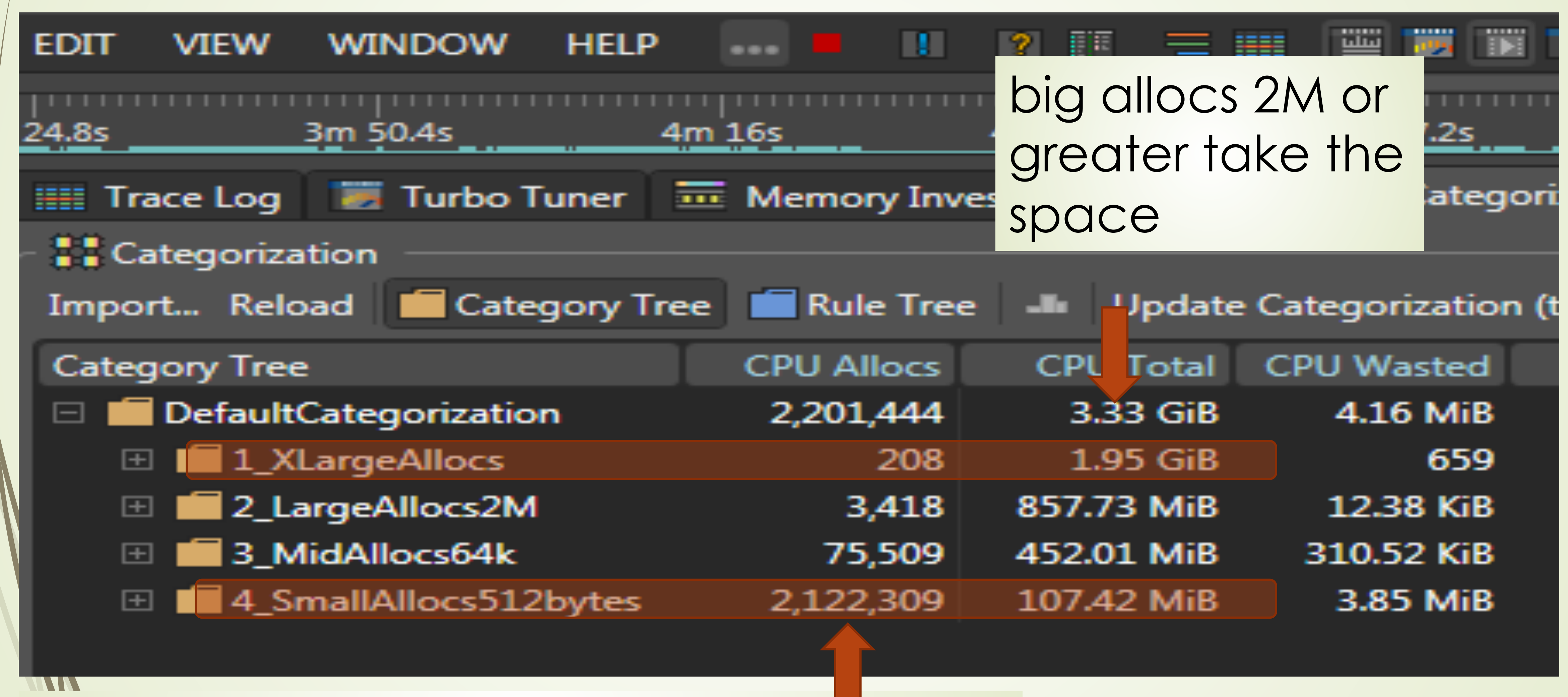
Scrub to another time



Before

After

Memory Categorization

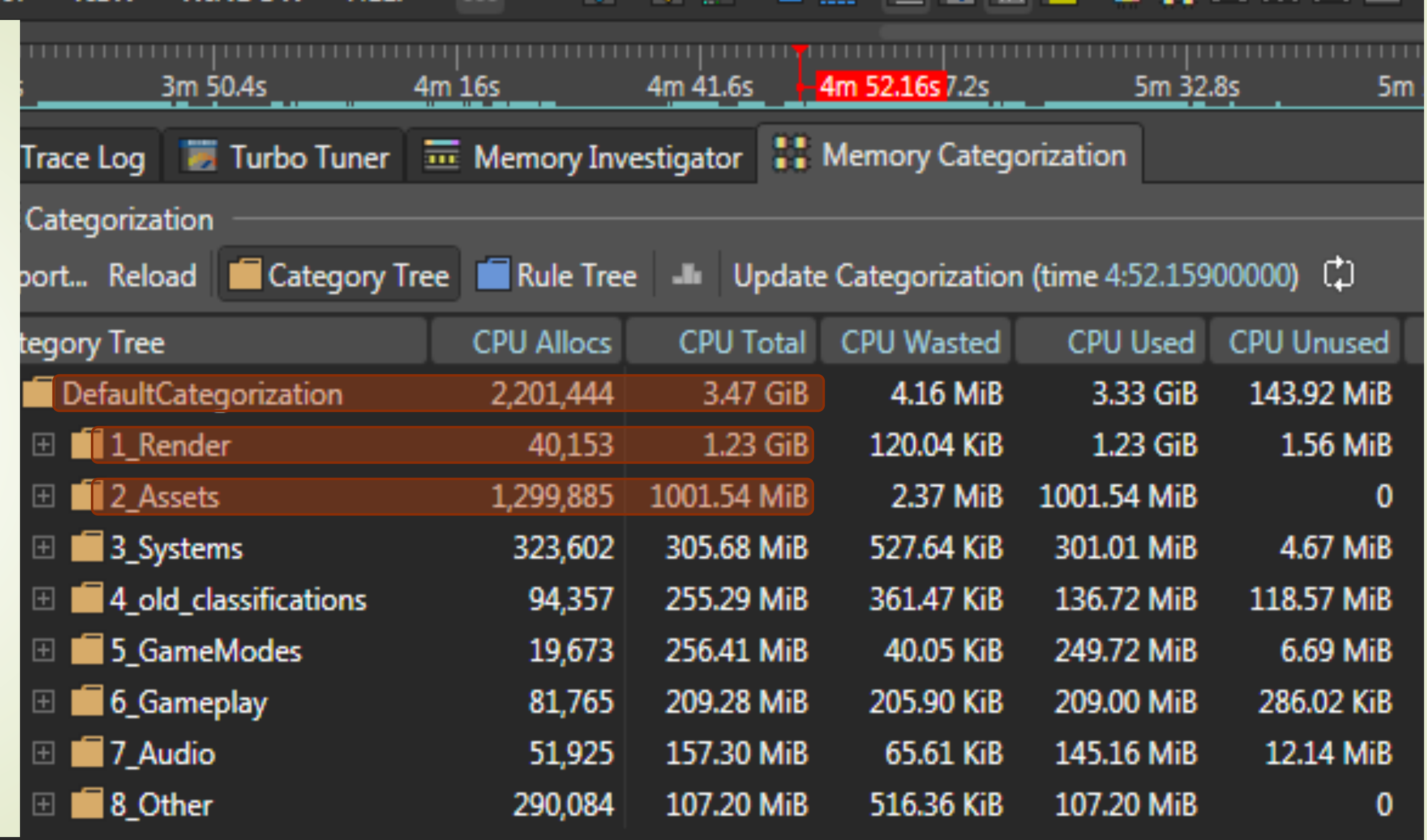


Lots of small allocs 512 bytes or smaller

Memory Categorization

the code is 50MiB small on this scale

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



tegory 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?