GDC

- Cold, Hard Cache
  Insomniac Games' Cache Simulator
- Andreas Fredriksson

  Lead Engine Programmer, Insomniac Games

Hi, I’m Andreas

- I lead the Core tools and infrastructure team at Insomniac

Hi, I’m Andreas

- I lead the Core tools and infrastructure team at Insomniac
- Today’s topic: Insomniac Games’ CacheSim
  - Custom tooling for measuring cache effectiveness

Hi, I’m Andreas

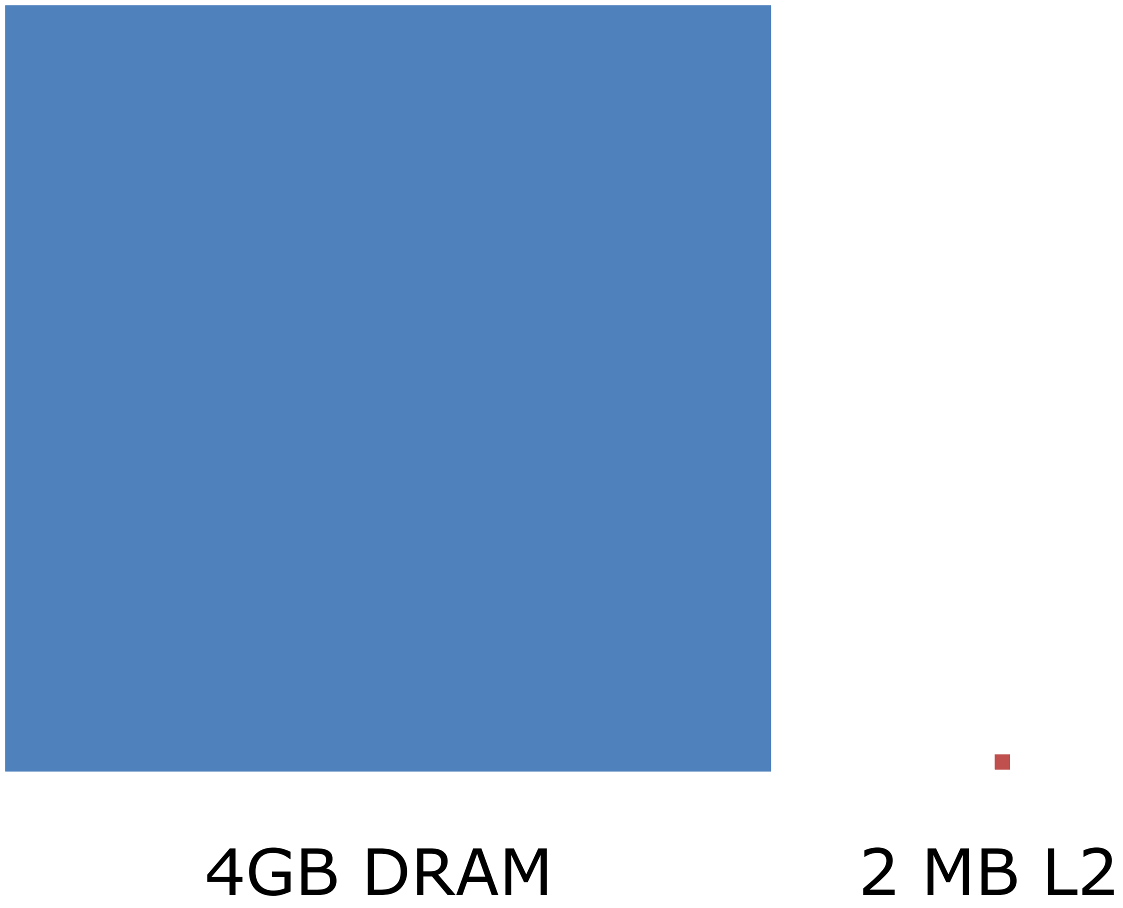
- I lead the Core tools and infrastructure team at Insomniac
- Today’s topic: Insomniac Games’ CacheSim
    - Custom tooling for measuring cache effectiveness
    - Excited to be open sourcing this and sharing with you all today

Cache and memory sizes, visually

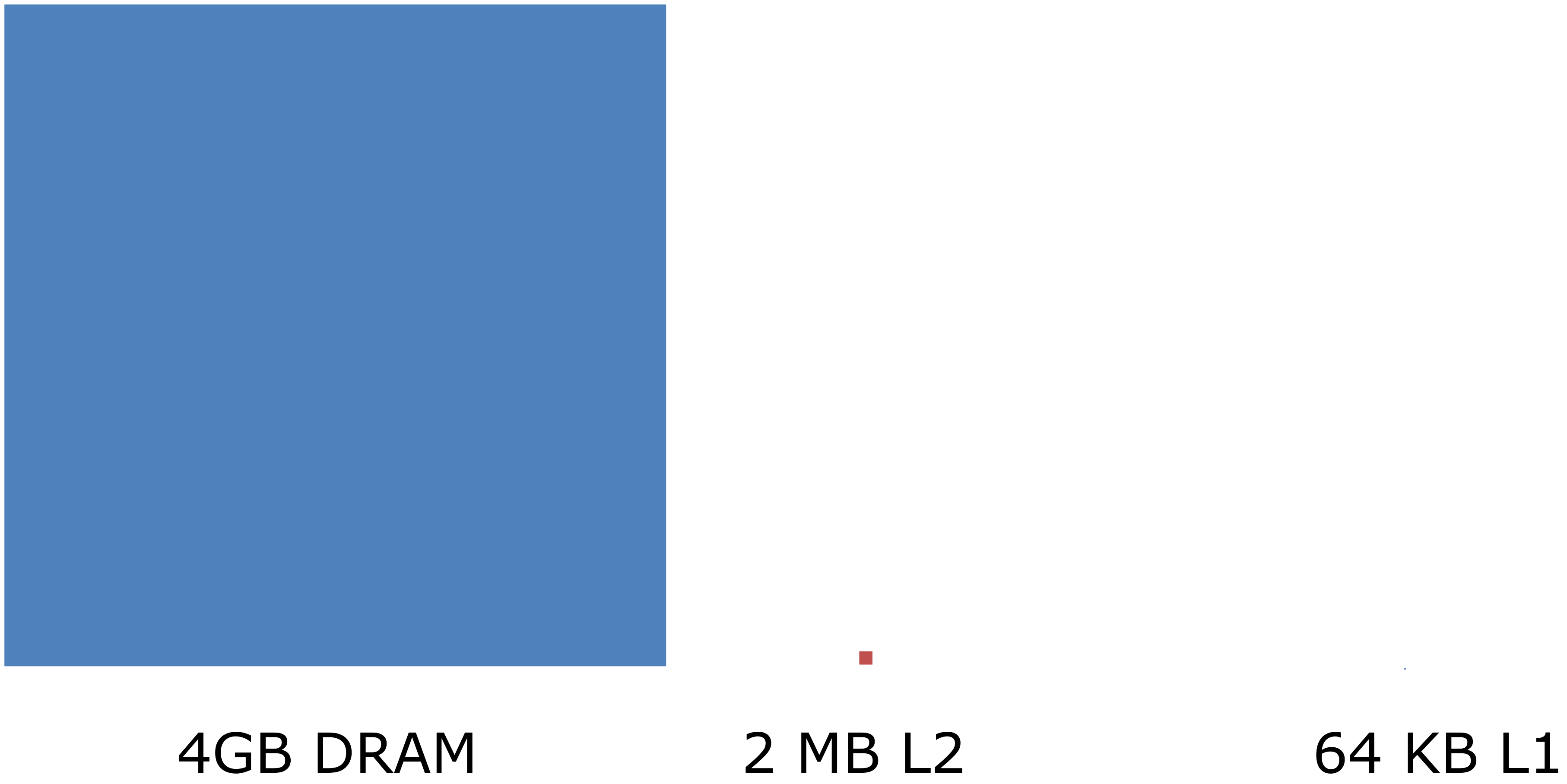
Cache and memory sizes, visually

4GB DRAM

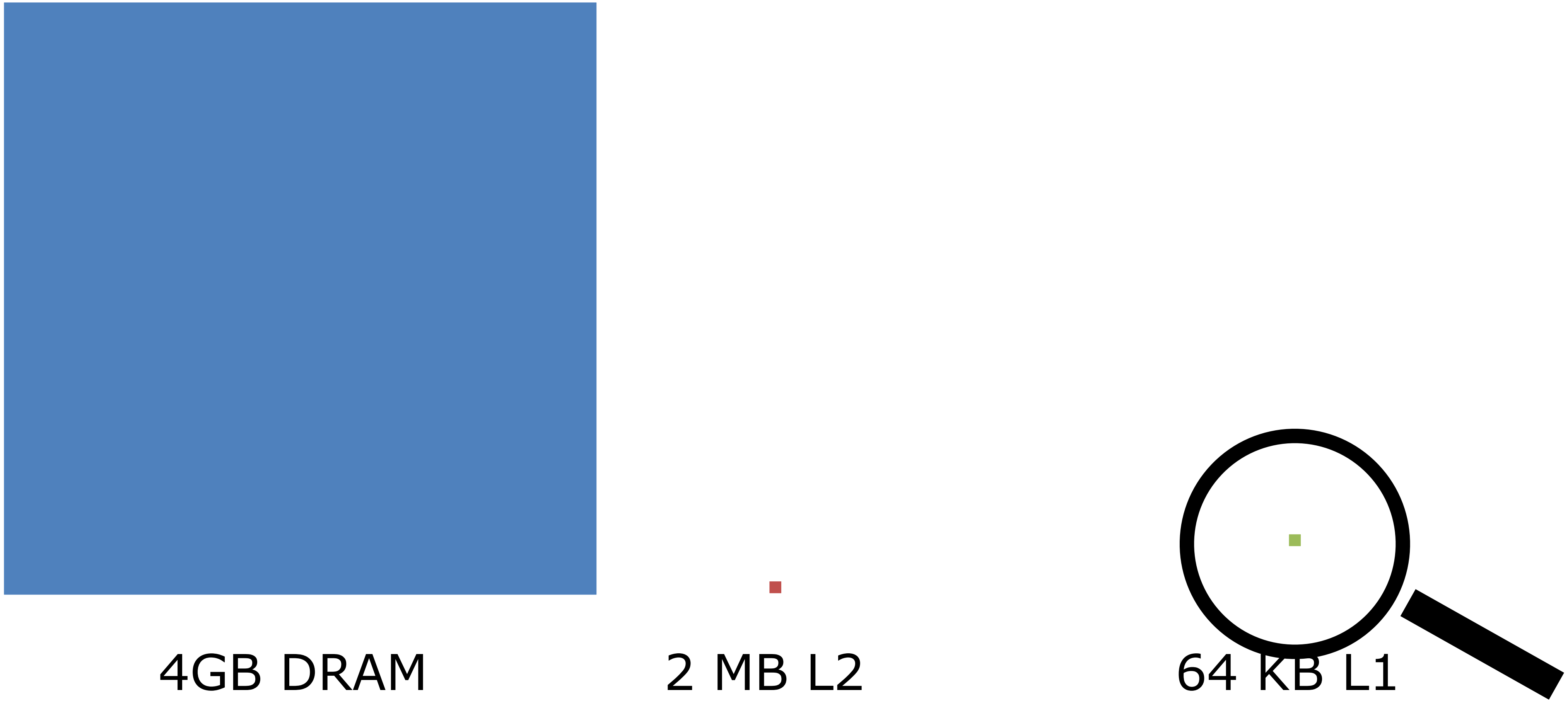
Cache and memory sizes, visually



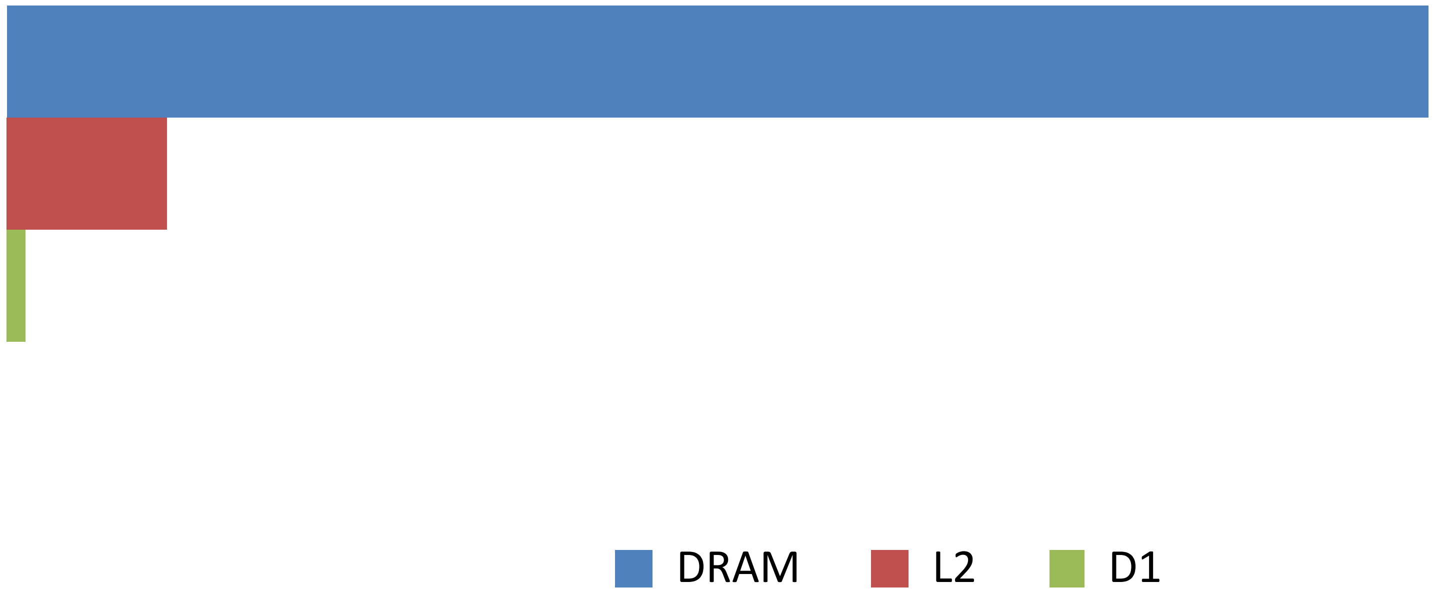
Cache and memory sizes, visually



Cache and memory sizes, visually



Cache, memory access speeds in cycles



Background

- Cache orders of magnitude faster and smaller than RAM
  - What you put into them makes a huge difference

Background

- Cache orders of magnitude faster and smaller than RAM
  - What you put into them makes a huge difference
- Memory operations are extremely easy to add to a program
  - Costs are hidden and non-obvious

Background

- Cache orders of magnitude faster and smaller than RAM
  - What you put into them makes a huge difference
- Memory operations are extremely easy to add to a program
  - Costs are hidden and non-obvious
- We desperately need actionable data on access patterns
  - Not a wealth of options for the performance-aware programmer

Sampling profilers

- They’ve basically won – most profilers are sample based
  - Great for many workflows
  - Leverage CPU features to gather HW stats about cache

Sampling profilers

- They've basically won – most profilers are sample based
  - Great for many workflows
  - Leverage CPU features to gather HW stats about cache
- Limitation: Only every N instructions are sampled (N is large)

Sampling profilers

- They’ve basically won – most profilers are sample based
  - Great for many workflows
  - Leverage CPU features to gather HW stats about cache
- Limitation: Only every N instructions are sampled (N is large)
- Not ideal for smaller, “bursty” workloads
  - Statistical means less reproducible for smaller things
  - Point you in the right direction, but that’s about it

Outside the sampling space

- Cachegrind – part of Valgrind
  - Simulate a cache based on the program's instruction stream

Outside the sampling space

- Cachegrind – part of Valgrind
  - Simulate a cache based on the program’s instruction stream
- Pros:
  - Extremely thorough – every memory access is simulated

Outside the sampling space

- Cachegrind – part of Valgrind
  - Simulate a cache based on the program's instruction stream
- Pros:
  - Extremely thorough – every memory access is simulated
- Cons:
  - Linux only
  - All or nothing
  - Extremely slow to get to a point of interest

Outside the sampling space

- Cachegrind – part of Valgrind
  - Simulate a cache based on the program’s instruction stream
- Pros:
  - Extremely thorough – every memory access is simulated
- Cons:
  - Linux only
  - All or nothing
  - Extremely slow to get to a point of interest
- Also – a vendor-specific tool from prevgen was awesome
  - Could do simulated captures on demand for a short time

Why you want cache simulation tooling

```
void PushBuffer::SetTextureAssets(uint32_t start_slot, uint32_t slot_count,
                                  const TextureAsset** textures_assets, uint32_t slot_mask, uint32_t hq_mask)
{
  // …
  for(uint32_t itex = 0; itex < slot_count; ++itex, tex_unit_test <<= 1)
  {
    Texture const*           tex          = (slot_mask & tex_unit_test ) ? textures_assets[itex]->GetTexture() : NULL;
    // …
    if(tex != NULL)
    {
      view = (textures_assets[itex]->GetFormatFlags() & TextureFormatFlags::kIsCube) ? tex->m_View : tex->m_ViewArray;
      samp = (hq_mask & tex_unit_test) ? textures_assets[itex]->GetAnisoSampler() : tex->m_SamplerState;
    }
    // …
```

Why you want cache simulation tooling

```
void PushBuffer::SetTextureAssets(uint32_t start_slot, uint32_t slot_count,
                                  const TextureAsset** textures_assets, uint32_t slot_mask, uint32_t hq_mask)
{
  // …
  for(uint32_t itex = 0; itex < slot_count; ++itex, tex_unit_test <<= 1)
  {
    Texture const*           tex          = (slot_mask & tex_unit_test ) ? textures_assets[itex]->GetTexture() : NULL;
    // …
    if(tex != NULL)
    {
      view = (textures_assets[itex]->GetFormatFlags() & TextureFormatFlags::kIsCube) ? tex->m_View : tex->m_ViewArray;
      samp = (hq_mask & tex_unit_test) ? textures_assets[itex]->GetAnisoSampler() : tex->m_SamplerState;
    }
    // …
```

2800 L2 misses in a frame

It’s the small things

- A member had moved cache lines
  - Accessing the 16-bit FormatFlags field was now a guaranteed L2 miss

```
//HOT: Cache Line 1
Texture       m_Texture;
TextureAsset* m_Default;

uint32_t      m_ResidentSize;
uint32_t      m_StreamSize;

uint32_t      m_AnisoSamplerIndex;
... other fields...

// COLD: Cache Line 2

uint16_t      m_FormatFlags;
uint16_t      m_Flags;

int16_t       m_TopHeight;
int16_t       m_TopWidth;
... other fields ...
```

It’s the small things

- A member had moved cache lines
  - Accessing the 16-bit FormatFlags field was now a guaranteed L2 miss

```
//HOT: Cache Line 1
Texture       m_Texture;
TextureAsset* m_Default;

uint32_t      m_ResidentSize;
uint32_t      m_StreamSize;

uint32_t      m_AnisoSamplerIndex;
... other fields...

// COLD: Cache Line 2

uint16_t      m_FormatFlags;
uint16_t      m_Flags;

int16_t       m_TopHeight;
int16_t       m_TopWidth;
... other fields ...
```

It’s the small things

- A member had moved cache lines
  - Accessing the 16-bit FormatFlags field was now a guaranteed L2 miss
- Fix: Literally swap two lines in header

```
//HOT: Cache Line 1
Texture       m_Texture;
TextureAsset* m_Default;

uint32_t      m_ResidentSize;
uint32_t      m_StreamSize;

uint32_t      m_AnisoSamplerIndex;
... other fields...

// COLD: Cache Line 2

uint16_t      m_FormatFlags;
uint16_t      m_Flags;

int16_t       m_TopHeight;
int16_t       m_TopWidth;
... other fields ...
```

It’s the small things

- A member had moved cache lines
  - Accessing the 16-bit FormatFlags field was now a guaranteed L2 miss
- Fix: Literally swap two lines in header
- 150-250 us savings depending on view

```
//HOT: Cache Line 1
Texture       m_Texture;
TextureAsset* m_Default;

uint32_t      m_ResidentSize;
uint32_t      m_StreamSize;

uint32_t      m_AnisoSamplerIndex;
... other fields...

// COLD: Cache Line 2

uint16_t      m_FormatFlags;
uint16_t      m_Flags;

int16_t       m_TopHeight;
int16_t       m_TopWidth;
... other fields ...
```

It’s the small things

- A member had moved cache lines
  - Accessing the 16-bit FormatFlags field was now a guaranteed L2 miss
- Fix: Literally swap two lines in header
- 150-250 us savings depending on view
- Time investment: 30 minutes

```
//HOT: Cache Line 1
Texture       m_Texture;
TextureAsset* m_Default;

uint32_t      m_ResidentSize;
uint32_t      m_StreamSize;

uint32_t      m_AnisoSamplerIndex;
... other fields...

// COLD: Cache Line 2

uint16_t      m_FormatFlags;
uint16_t      m_Flags;

int16_t       m_TopHeight;
int16_t       m_TopWidth;
... other fields ...
```

More reasons you want cache tooling

- Small utilities miss cache a lot sometimes
  - `Vec3 SceneObject::GetPosition() { return m_ObjToWorld.v[3]; }`

More reasons you want cache tooling

- Small utilities miss cache a lot sometimes
  - `Vec3 SceneObject::GetPosition() { return m_ObjToWorld.v[3]; }`
- Can’t optimize a single return statement! /tableflip
  - But you can optimize callers of that function to be less naïve

More reasons you want cache tooling

- Small utilities miss cache a lot sometimes
  - `Vec3 SceneObject::GetPosition() { return m_ObjToWorld.v[3]; }`
- Can’t optimize a single return statement! /tableflip
  - But you can optimize callers of that function to be less naïve
- Need to track call stacks as well, pass blame up the stack

More reasons you want cache tooling

- Small utilities miss cache a lot sometimes
  - `Vec3 SceneObject::GetPosition() { return m_ObjToWorld.v[3]; }`
- Can’t optimize a single return statement! /tableflip
  - But you can optimize callers of that function to be less naïve
- Need to track call stacks as well, pass blame up the stack
- Found that 12k out of 14k misses in GetPosition() came from one gameplay system ☺️

Who’s calling SceneObject::GetPosition() with cold data?

```
uint32_t num_groups = 0;
// Get a ton of stuff to work on
ComponentHandle *groups = g_PlacedPedestrianSystem.GetGroups(num_groups);

for (uint32_t idx = 0; idx < num_groups; ++idx)
{
  PlacedPedestrianGroup *group = (PlacedPedestrianGroup*)groups[idx].Resolve();
  if(!group)
    continue;

  const Vec3 &group_pos = group->GetActorPosition();
  // ...
}
```

Who’s calling SceneObject::GetPosition() with cold data?

```
uint32_t num_groups = 0;
// Get a ton of stuff to work on
ComponentHandle *groups = g_PlacedPedestrianSystem.GetGroups(num_groups);

for (uint32_t idx = 0; idx < num_groups; ++idx)
{
  PlacedPedestrianGroup *group = (PlacedPedestrianGroup*)groups[idx].Resolve();
  if(!group)
    continue;

  const Vec3 &group_pos = group->GetActorPosition();
  // ...
}
```

~12,000 L2 misses in a frame

Don’t ask about things you already know

- About 1 ms/frame on PC
  - Thousands of (random order) items being processed every frame

Don’t ask about things you already know

- About 1 ms/frame on PC
  - Thousands of (random order) items being processed every frame
- Digging in: They never move

Don’t ask about things you already know

- About 1 ms/frame on PC
  - Thousands of (random order) items being processed every frame
- Digging in: They never move
- Fix: Keep local cache of positions in parallel array
  - Avoiding the GetPosition() call entirely

Don’t ask about things you already know

- About 1 ms/frame on PC
  - Thousands of (random order) items being processed every frame
- Digging in: They never move
- Fix: Keep local cache of positions in parallel array
  - Avoiding the GetPosition() call entirely
- Result: 650 us/frame saved

Don’t ask about things you already know

- About 1 ms/frame on PC
    - Thousands of (random order) items being processed every frame
- Digging in: They never move
- Fix: Keep local cache of positions in parallel array
    - Avoiding the GetPosition() call entirely
- Result: 650 us/frame saved
- Time investment: 2 hours

Instructions, instructions

- All the data we want is right there, in the instruction stream
    - We "just" need a way to look at each instruction as it executes

Instructions, instructions

- All the data we want is right there, in the instruction stream
  - We "just" need a way to look at each instruction as it executes
- Plan:
  - Flip a switch upon reaching a point of interest
  - Trace every instruction (somehow)
  - Update a simulated cache for each memory access
  - Turn off trace (say at end of frame) & report!

Instructions, instructions

- All the data we want is right there, in the instruction stream
  - We "just" need a way to look at each instruction as it executes
- Plan:
  - Flip a switch upon reaching a point of interest
  - Trace every instruction (somehow)
  - Update a simulated cache for each memory access
  - Turn off trace (say at end of frame) & report!
- But how?

Where to start?

- First: It really helps to have an understanding boss

Where to start?

- First: It really helps to have an understanding boss

Where to start?

- First: It really helps to have an understanding boss
- Pitched 2 week project to dig in..
  - Subject: "I can see crazy town from here"

Where to start?

- First: It really helps to have an understanding boss
- Pitched 2 week project to dig in..
  - Subject: “I can see crazy town from here”
- Approved!
  - Ok, let’s start pitching a tent outside crazy town

Reasoning about instructions

- Binary instrumentation frameworks exist (off the shelf)
  - DynamoRIO, Intel PIN, others

Reasoning about instructions

- Binary instrumentation frameworks exist (off the shelf)
  - DynamoRIO, Intel PIN, others
- Quickly discarded this approach
  - Massive performance problems instrumenting a AAA game executable

Reasoning about instructions

- Binary instrumentation frameworks exist (off the shelf)
  - DynamoRIO, Intel PIN, others
- Quickly discarded this approach
  - Massive performance problems instrumenting a AAA game executable
- Could have value for other things in our space
  - More guided dynamic instrumentation without code changes
  - “How often is this value zero at this spot?”
  - “What are the min/max input values to this function?”

Idea #1

- Somehow write void TraceFunction(func_ptr)

Idea #1

- Somehow write void TraceFunction(func_ptr)
- Somehow, for each instruction:
  - Disassemble the instruction
  - Find memory derefs, update a simulated cache
  - Copy instruction to temp buffer, run in isolation
- Sounds easy!

Idea #1 – not going anywhere

- Branches need to be special handled

Idea #1 – not going anywhere

- Branches need to be special handled
- Implicit uses of RIP (instruction pointer) are everywhere

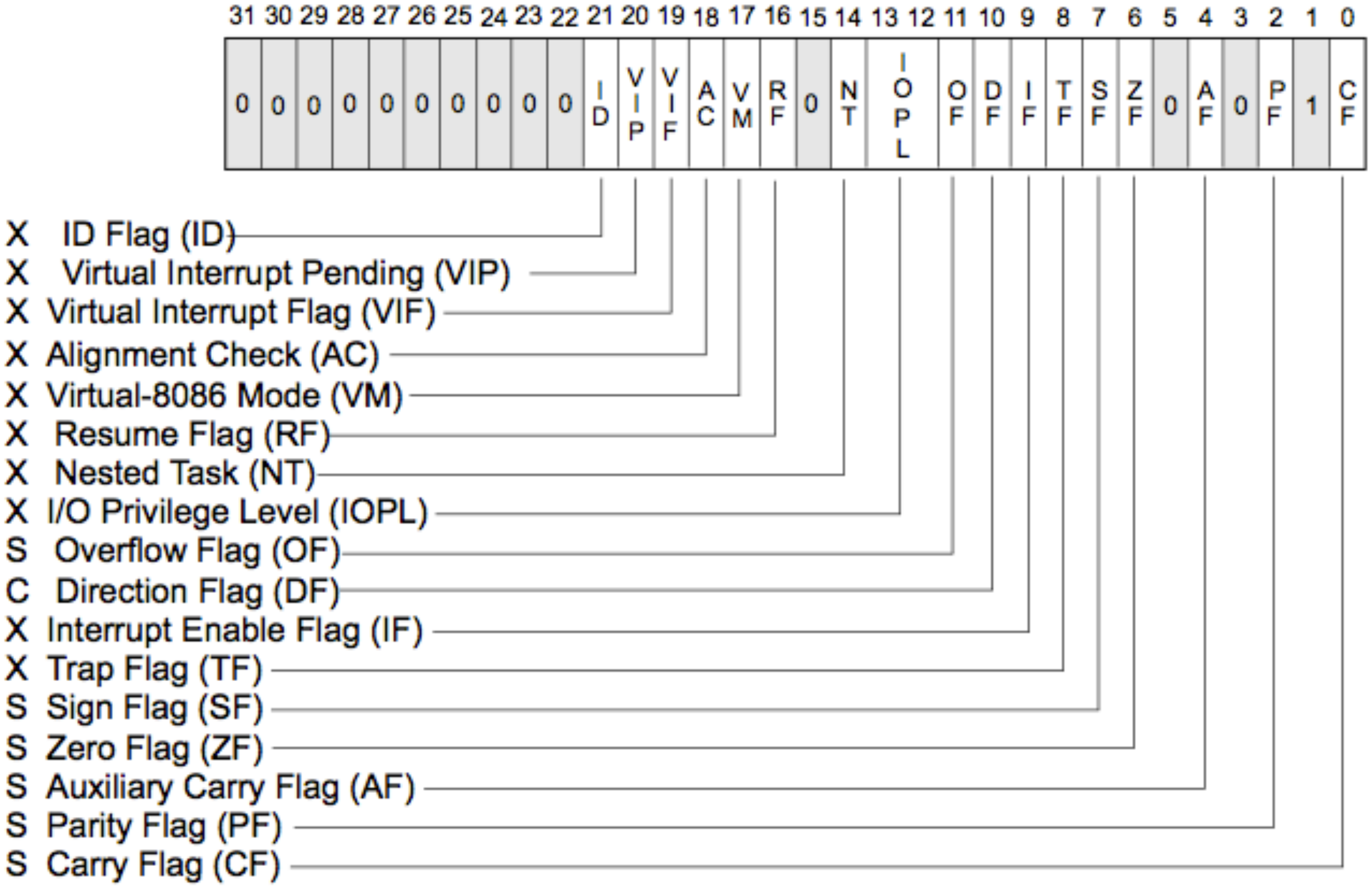
Idea #1 – not going anywhere

- Branches need to be special handled
- Implicit uses of RIP (instruction pointer) are everywhere
- Win64 exception handling has rules we're violating
  - OutputDebugString uses exceptions..

Idea #1 – not going anywhere

- Branches need to be special handled
- Implicit uses of RIP (instruction pointer) are everywhere
- Win64 exception handling has rules we're violating
    - OutputDebugString uses exceptions..
- Super intrusive – need a top-level call to our trace function
    - On every thread

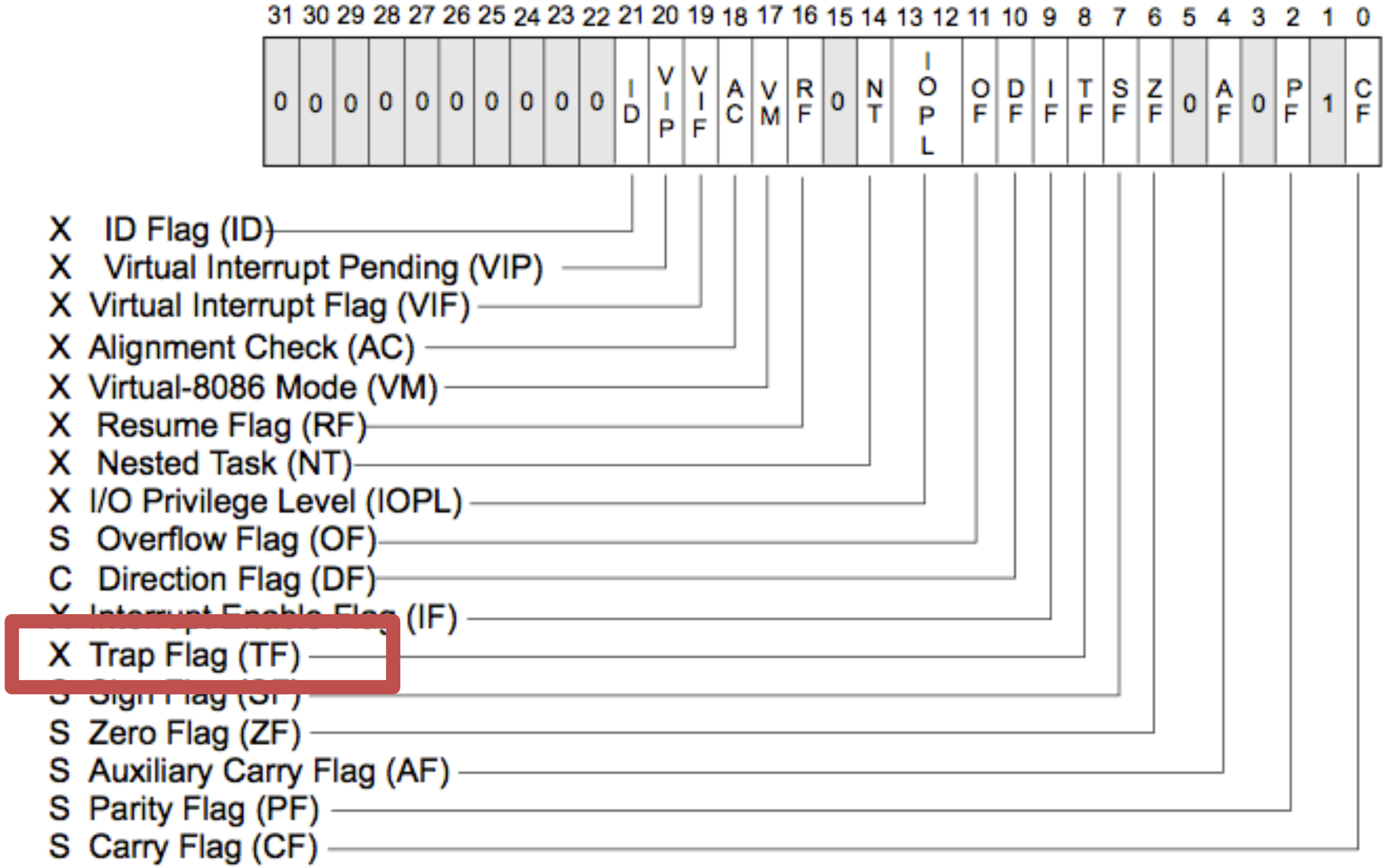
Suddenly: EFLAGS



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

Suddenly: EFLAGS



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

Idea #2: Leveraging EFLAGS

- Single stepping is a CPU feature
  - It’s how F11 in the debugger works
  - Set TRAP bit in EFLAGS

Idea #2: Leveraging EFLAGS

- Single stepping is a CPU feature
  - It’s how F11 in the debugger works
  - Set TRAP bit in EFLAGS
- Routes as an exception through the Windows SEH machinery
  - You can install a handler for it!

Idea #2: Leveraging EFLAGS

- Single stepping is a CPU feature
  - It’s how F11 in the debugger works
  - Set TRAP bit in EFLAGS
- Routes as an exception through the Windows SEH machinery
  - You can install a handler for it!
- But how do you install SEH handlers for all threads?
  - Vectored exception handlers

Revised plan of attack

- To start tracing:
  - Install a VEH to filter TRAP exceptions
  - Set TF EFLAGS bit for all threads we want to capture

Revised plan of attack

- To start tracing:
  - Install a VEH to filter TRAP exceptions
  - Set TF EFLAGS bit for all threads we want to capture
- In the handler:
  - Disassemble instruction, find memory operands
  - Update cache simulation
  - Re-set the TF bit before leaving to keep tracing

Revised plan of attack

- To start tracing:
  - Install a VEH to filter TRAP exceptions
  - Set TF EFLAGS bit for all threads we want to capture
- In the handler:
  - Disassemble instruction, find memory operands
  - Update cache simulation
  - Re-set the TF bit before leaving to keep tracing
- To stop tracing:
  - Set some flag and (ultimately) remove the VEH

Every plan has problems

- Good news: It basically works

Every plan has problems

- Good news: It basically works
- Problem #1: The debugger is really unhappy
  - "What? You want to break in?"
  - Solution: Run detached

Every plan has problems

- Good news: It basically works
- Problem #1: The debugger is really unhappy
  - “What? You want to break in?”
  - Solution: Run detached
- Problem #2: Massive amounts of deadlocks in ntdll.dll
  - Hanging on contended SRW lock protecting the VEH dispatch list
  - Threads waiting on wakeups for locks
  - But no one owns the lock, so it can never wake up

Deadlock woes

- Using SEH way, way more than anyone had anticipated at MS
  - Every thread, every instruction will exercise the exception handling
- Nothing was obviously wrong in the code
- Suspect problem is reentrantly messing with critical sections

Solution: Disable the locking code in ntdll

- VEH are an exotic feature, typically no handlers are installed
  - This is a debugging feature, not something we’re shipping to players
  - Just smash ntdll!RtlpCallVectoredHandlers with a jump to our handler
- It’s ugly, but it gets the job done
- Also: take no OS locks internally, just spinlocks

[cue evil laugh]
memes.com

It works!

- We can inspect instructions one by one!

It works!

- We can inspect instructions one by one!
- Need a disassembler that knows about memory operands
  - Radare2 fork of udis86 fit the bill

It works!

- We can inspect instructions one by one!
- Need a disassembler that knows about memory operands
  - Radare2 fork of udis86 fit the bill

```
uintptr_t rip = ExcInfo->ContextRecord->Rip;

ud_set_input_buffer(ud, (const uint8_t*) rip, 16);
ud_set_pc(ud, rip);
int ilen = ud_disassemble(ud);
GenerateMemoryAccesses(core_index, ud, rip, ilen, ExcInfo->ContextRecord);
```

Generating memory accesses

- Easy, right? Just look at memory operands
  - `mov dword ptr [rax], ebx` => write 4 bytes at `rax`

Generating memory accesses

- Easy, right? Just look at memory operands
  - `mov dword ptr [rax], ebx` => write 4 bytes at `rax`
- Yeah, in theory..
  - Lots of things in x64 have memory operands
  - Some access memory and don't have memory operands!

Lots of special cases to consider

- String instructions, e.g. LODSB, MOVSD (implicit RSI/RDI accesses)
- Stack push/pop
- CALL, RET (also write/read the stack respectively)
- LEA – super common, has mem operand but doesn’t touch memory
- Crazy “long nop” instructions can have memory operands
- FXSTOR/FXRSTOR
- Prefetches and non-temporal loads/stores

Poking the cache simulation

```
// Generate I-cache traffic.
CacheSim::AccessResult r = g_Cache.Access(core_index, rip, ilen, CacheSim::kCodeRead);
stats->m_Stats[r] += 1;
```

Poking the cache simulation

```
// Generate I-cache traffic.
CacheSim::AccessResult r = g_Cache.Access(core_index, rip, ilen, CacheSim::kCodeRead);
stats->m_Stats[r] += 1;

// Generate D-cache traffic.
for (int i = 0; i < read_count; ++i)
{
  CacheSim::AccessResult r = g_Cache.Access(core_index, reads[i].ea, reads[i].sz, CacheSim::kRead);
  stats->m_Stats[r] += 1;
}
```

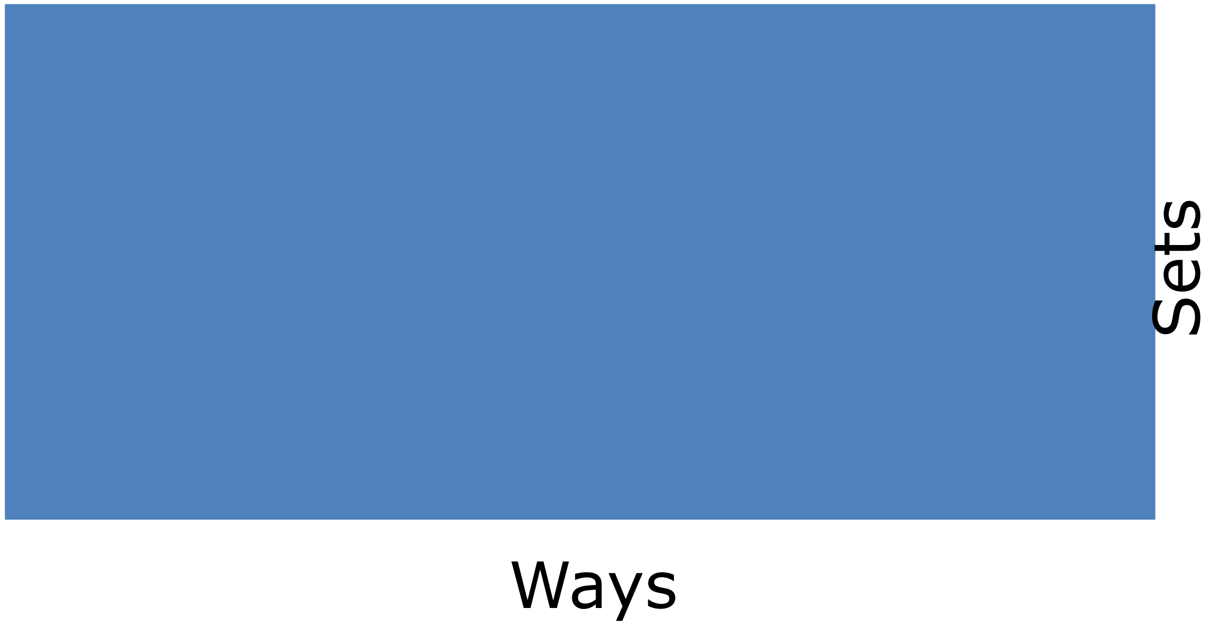
Poking the cache simulation

```
// Generate I-cache traffic.
CacheSim::AccessResult r = g_Cache.Access(core_index, rip, ilen, CacheSim::kCodeRead);
stats->m_Stats[r] += 1;

// Generate D-cache traffic.
for (int i = 0; i < read_count; ++i)
{
  CacheSim::AccessResult r = g_Cache.Access(core_index, reads[i].ea, reads[i].sz, CacheSim::kRead);
  stats->m_Stats[r] += 1;
}

for (int i = 0; i < write_count; ++i)
{
  CacheSim::AccessResult r = g_Cache.Access(core_index, writes[i].ea, writes[i].sz, CacheSim::kWrite);
  stats->m_Stats[r] += 1;
}
```

Can model a set-assoc cache as a 2D array

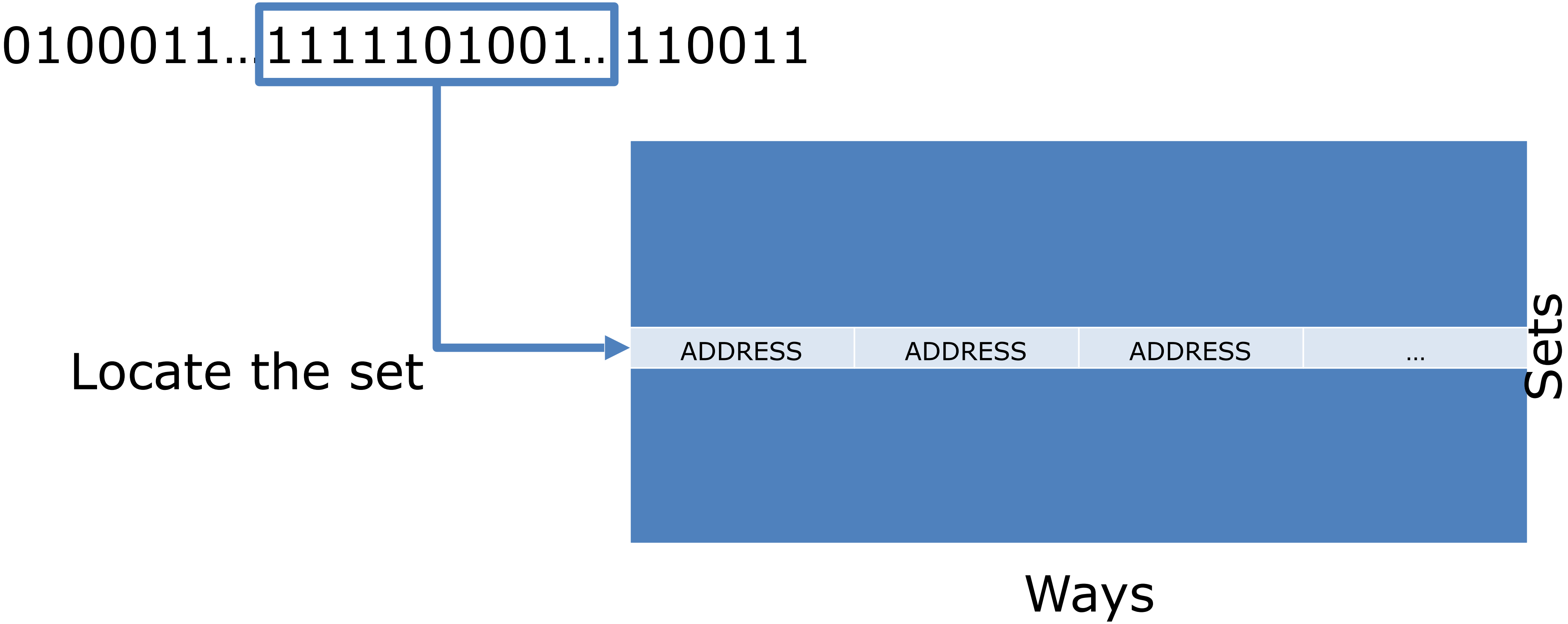
Take apart input address:

0100011…1111101001…110011



Can model a set-assoc cache as a 2D array

Take apart input address:



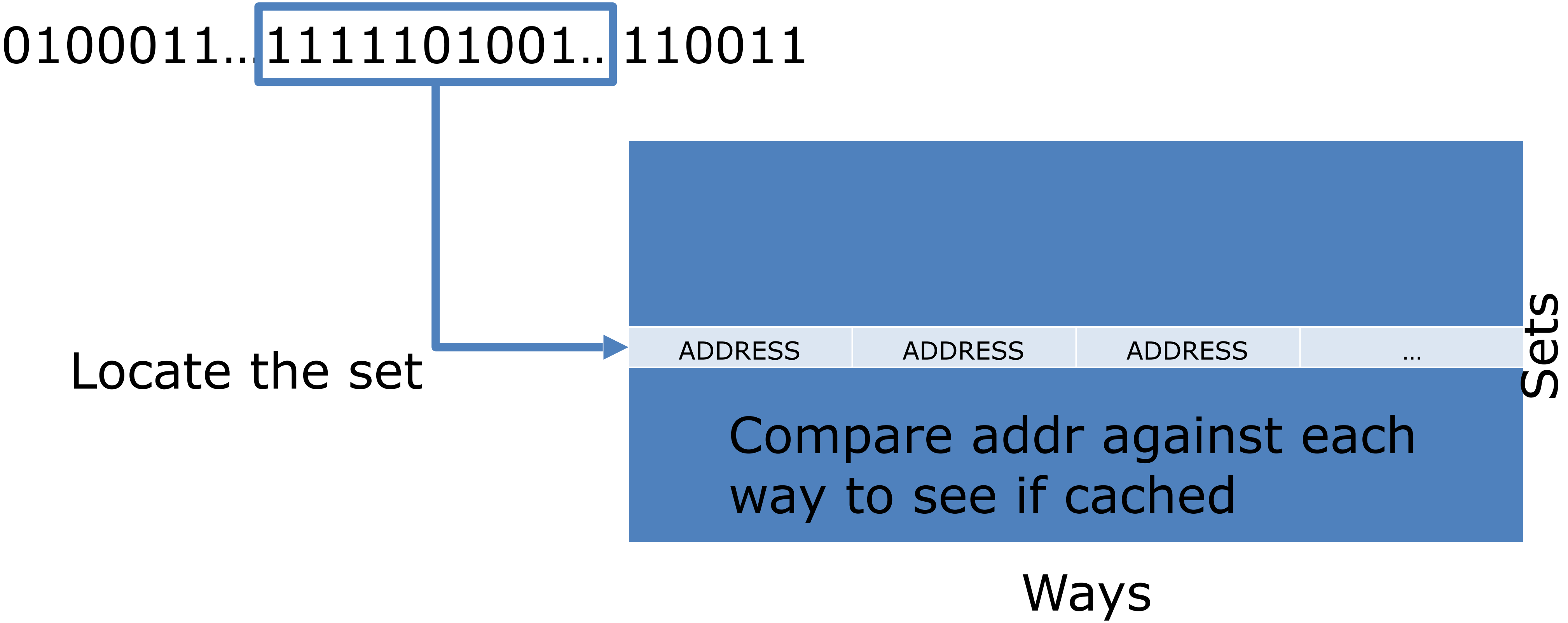
0100011…1111101001…110011

Locate the set

ADDRESS	ADDRESS	ADDRESS	…

Can model a set-assoc cache as a 2D array

Take apart input address:



0100011…1111101001…110011

Locate the set

ADDRESS | ADDRESS | ADDRESS | …

Compare addr against each way to see if cached

Simulating a Jaguar cache

- Console Jaguar has 2 modules
  - Each modules has a shared L2, 4 cores
  - Each core has its own D1, I1 caches

Simulating a Jaguar cache

- Console Jaguar has 2 modules
  - Each modules has a shared L2, 4 cores
  - Each core has its own D1, I1 caches
- Jaguar cache is *inclusive* (lines in D1/I1 must also exist in L2)

Simulating a Jaguar cache

- Console Jaguar has 2 modules
  - Each modules has a shared L2, 4 cores
  - Each core has its own D1, I1 caches
- Jaguar cache is *inclusive* (lines in D1/I1 must also exist in L2)
- Map our set associativity for our array structures
  - I1: 512 lines (32 KB), 2 ways, 256 sets
  - D1: 512 lines (32 KB), 8 ways, 64 sets
  - L2: 32,768 lines (2 MB), 16 ways, 2,048 sets

Defining our caches

```
// Simulating a Jaguar cache

//                      size in byte   |  assoc
using JaguarD1 = Cache<      32 * 1024,      8>;
using JaguarI1 = Cache<      32 * 1024,      2>;
using JaguarL2 = Cache<2 * 1024 * 1024,     16>;
```

Defining our caches

```
// Simulating a Jaguar cache

//                     size in byte    |  assoc
using JaguarD1 = Cache<      32 * 1024,      8>;
using JaguarI1 = Cache<      32 * 1024,      2>;
using JaguarL2 = Cache<2 * 1024 * 1024,     16>;


struct JaguarModule
{
  JaguarD1      m_CoreD1[4];   // 4 cores per module, with private D1 & I1
  JaguarI1      m_CoreI1[4];
  JaguarL2      m_Level2;      // A shared L2
  JaguarModule* m_OtherModule; // Pointer to other module for invalidations
};
```

Updating the cache, in pseudocode

```
For each cache line accessed:
```

Updating the cache, in pseudocode

```
For each cache line accessed:

  If we're writing:
    Kick line out of every other core
    Kick line out of other module's L2
```

Updating the cache, in pseudocode

```
For each cache line accessed:

  If we're writing:
    Kick line out of every other core
    Kick line out of other module's L2

  Hit1 = Lookup+Record Line in D1/I1
  Hit2 = Lookup+Record Line in L2
```

Updating the cache, in pseudocode

```
For each cache line accessed:

  If we're writing:
    Kick line out of every other core
    Kick line out of other module's L2

  Hit1 = Lookup+Record Line in D1/I1
  Hit2 = Lookup+Record Line in L2


  If Hit1 && Hit2:
    return kL1Hit
  Else If Hit2:
    return kL2Hit
  Else:
    return kL2Miss
```

Running it

- Hook up trace machinery to keyboard shortcut in main loop
  - Automatically disable at end of frame

Running it

- Hook up trace machinery to keyboard shortcut in main loop
  - Automatically disable at end of frame
- Data collection takes about 2-3 minutes
  - Depends on workload

Running it

- Hook up trace machinery to keyboard shortcut in main loop
  - Automatically disable at end of frame
- Data collection takes about 2-3 minutes
  - Depends on workload
- Stash results in binary file
  - About 100-150 MB of data for our use case

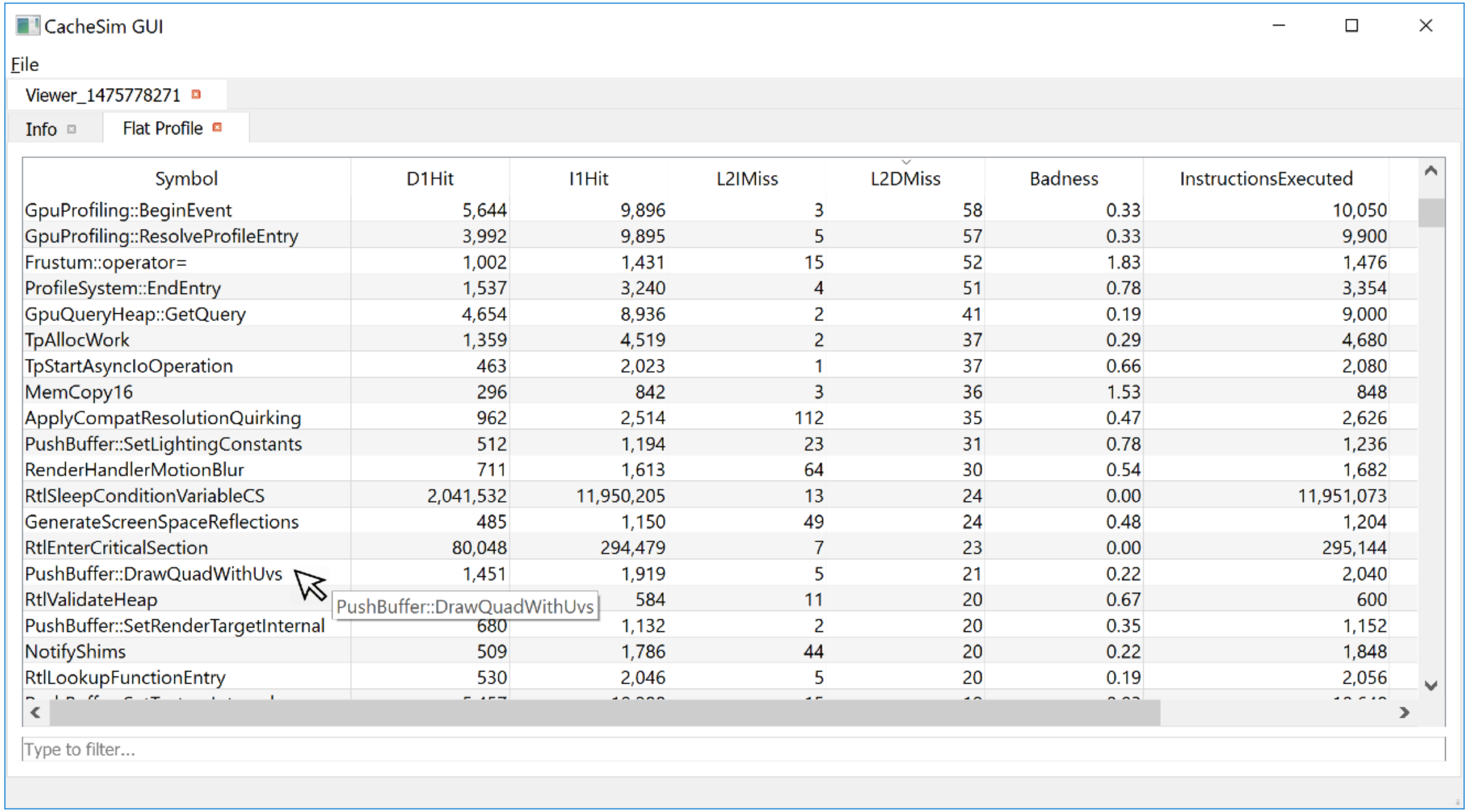
Running it

- Hook up trace machinery to keyboard shortcut in main loop
  - Automatically disable at end of frame
- Data collection takes about 2-3 minutes
  - Depends on workload
- Stash results in binary file
  - About 100-150 MB of data for our use case
- Game resumes running at full framerate after collection!
  - Analyze dump offline

Analysis

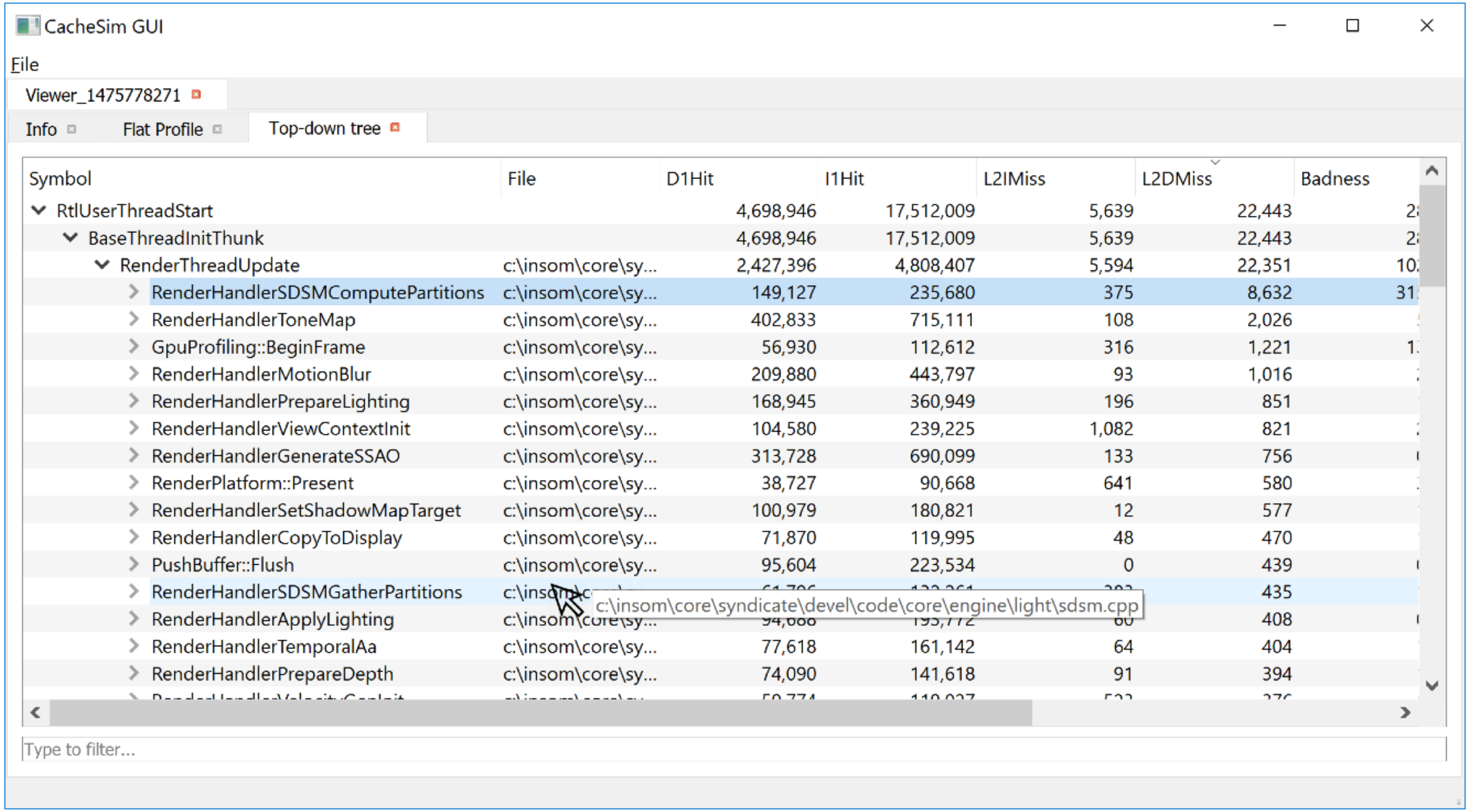
- Stats collected are associated with an instruction
  - Call stack is also captured, used to disambiguate
- Current stats captured by our setup:
  - L1 hit (separate tracking of I1/D1)
  - L2 hit
  - L2 miss (tracks instruction/data separately – better than HW can!)
  - # of explicit prefetches that hit D1 or L2
  - #Instructions executed

Tooling – Flat profile of stats



Symbol	D1Hit	I1Hit	L2IMiss	L2DMiss	Badness	InstructionsExecuted
GpuProfiling::BeginEvent	5,644	9,896	3	58	0.33	10,050
GpuProfiling::ResolveProfileEntry	3,992	9,895	5	57	0.33	9,900
Frustum::operator=	1,002	1,431	15	52	1.83	1,476
ProfileSystem::EndEntry	1,537	3,240	4	51	0.78	3,354
GpuQueryHeap::GetQuery	4,654	8,936	2	41	0.19	9,000
TpAllocWork	1,359	4,519	2	37	0.29	4,680
TpStartAsyncIoOperation	463	2,023	1	37	0.66	2,080
MemCopy16	296	842	3	36	1.53	848
ApplyCompatResolutionQuirking	962	2,514	112	35	0.47	2,626
PushBuffer::SetLightingConstants	512	1,194	23	31	0.78	1,236
RenderHandlerMotionBlur	711	1,613	64	30	0.54	1,682
RtlSleepConditionVariableCS	2,041,532	11,950,205	13	24	0.00	11,951,073
GenerateScreenSpaceReflections	485	1,150	49	24	0.48	1,204
RtlEnterCriticalSection	80,048	294,479	7	23	0.00	295,144
PushBuffer::DrawQuadWithUvs	1,451	1,919	5	21	0.22	2,040
RtlValidateHeap	[illegible]	584	11	20	0.67	600
PushBuffer::SetRenderTargetInternal	680	1,132	2	20	0.35	1,152
NotifyShims	509	1,786	44	20	0.22	1,848
RtlLookupFunctionEntry	530	2,046	5	20	0.19	2,056

Tooling – Top-down tree

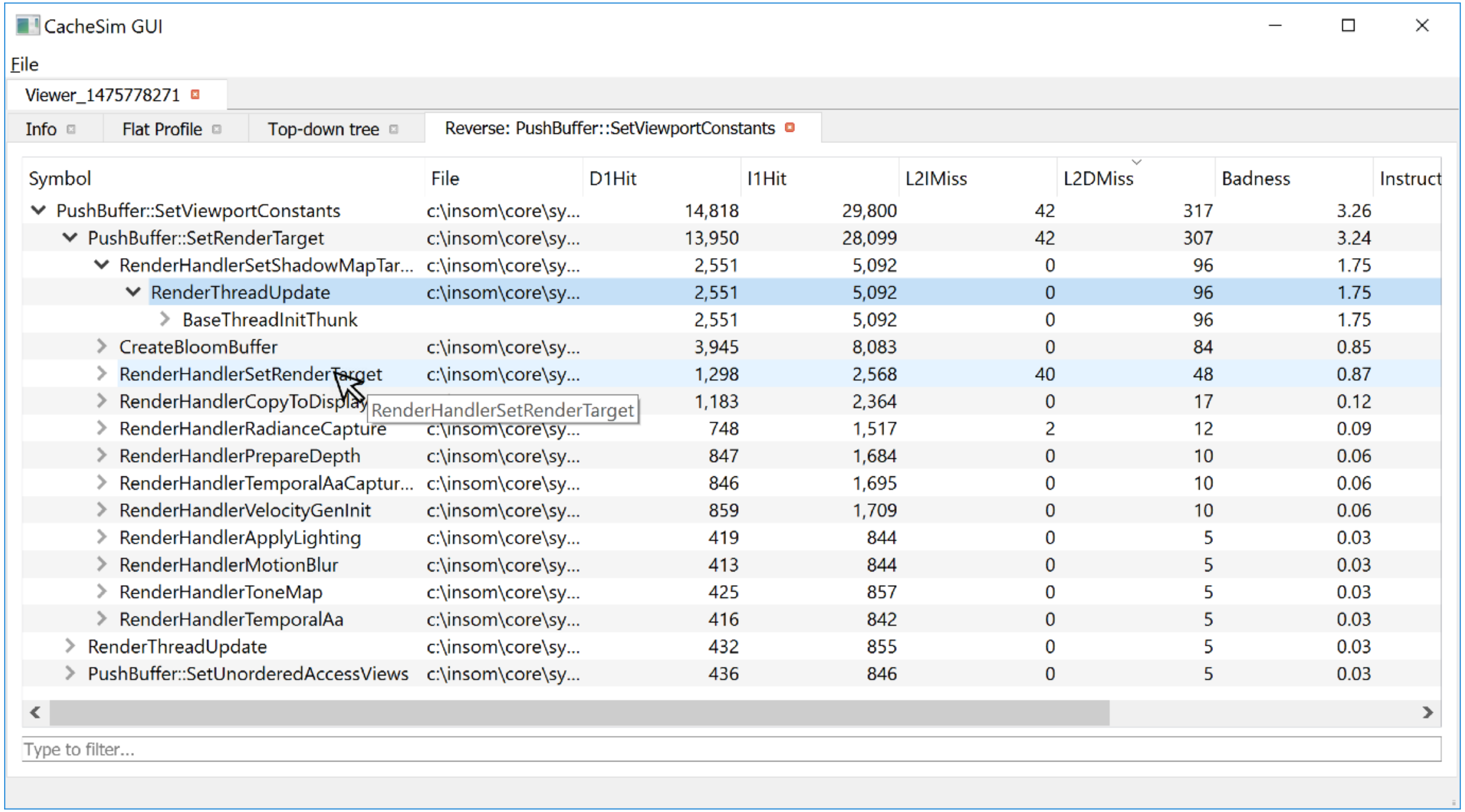


Symbol	File	D1Hit	I1Hit	L2IMiss	L2DMiss	Badness
RtlUserThreadStart		4,698,946	17,512,009	5,639	22,443	2
BaseThreadInitThunk		4,698,946	17,512,009	5,639	22,443	2
RenderThreadUpdate	c:\insom\core\sy...	2,427,396	4,808,407	5,594	22,351	10
RenderHandlerSDSMComputePartitions	c:\insom\core\sy...	149,127	235,680	375	8,632	31
RenderHandlerToneMap	c:\insom\core\sy...	402,833	715,111	108	2,026	
GpuProfiling::BeginFrame	c:\insom\core\sy...	56,930	112,612	316	1,221	1
RenderHandlerMotionBlur	c:\insom\core\sy...	209,880	443,797	93	1,016	
RenderHandlerPrepareLighting	c:\insom\core\sy...	168,945	360,949	196	851	
RenderHandlerViewContextInit	c:\insom\core\sy...	104,580	239,225	1,082	821	
RenderHandlerGenerateSSAO	c:\insom\core\sy...	313,728	690,099	133	756	
RenderPlatform::Present	c:\insom\core\sy...	38,727	90,668	641	580	
RenderHandlerSetShadowMapTarget	c:\insom\core\sy...	100,979	180,821	12	577	
RenderHandlerCopyToDisplay	c:\insom\core\sy...	71,870	119,995	48	470	
PushBuffer::Flush	c:\insom\core\sy...	95,604	223,534	0	439	
RenderHandlerSDSMGatherPartitions	c:\insom\c	61,706	132,261	303	435	
RenderHandlerApplyLighting	c:\insom\core\sy...	94,688	193,772	60	408	
RenderHandlerTemporalAa	c:\insom\core\sy...	77,618	161,142	64	404	
RenderHandlerPrepareDepth	c:\insom\core\sy...	74,090	141,618	91	394	
RenderHandlerVelocityConInit	c:\insom\core\sy...	59,774	110,027	523	376	

c:\insom\core\syndicate\devel\code\core\engine\light\sdsm.cpp

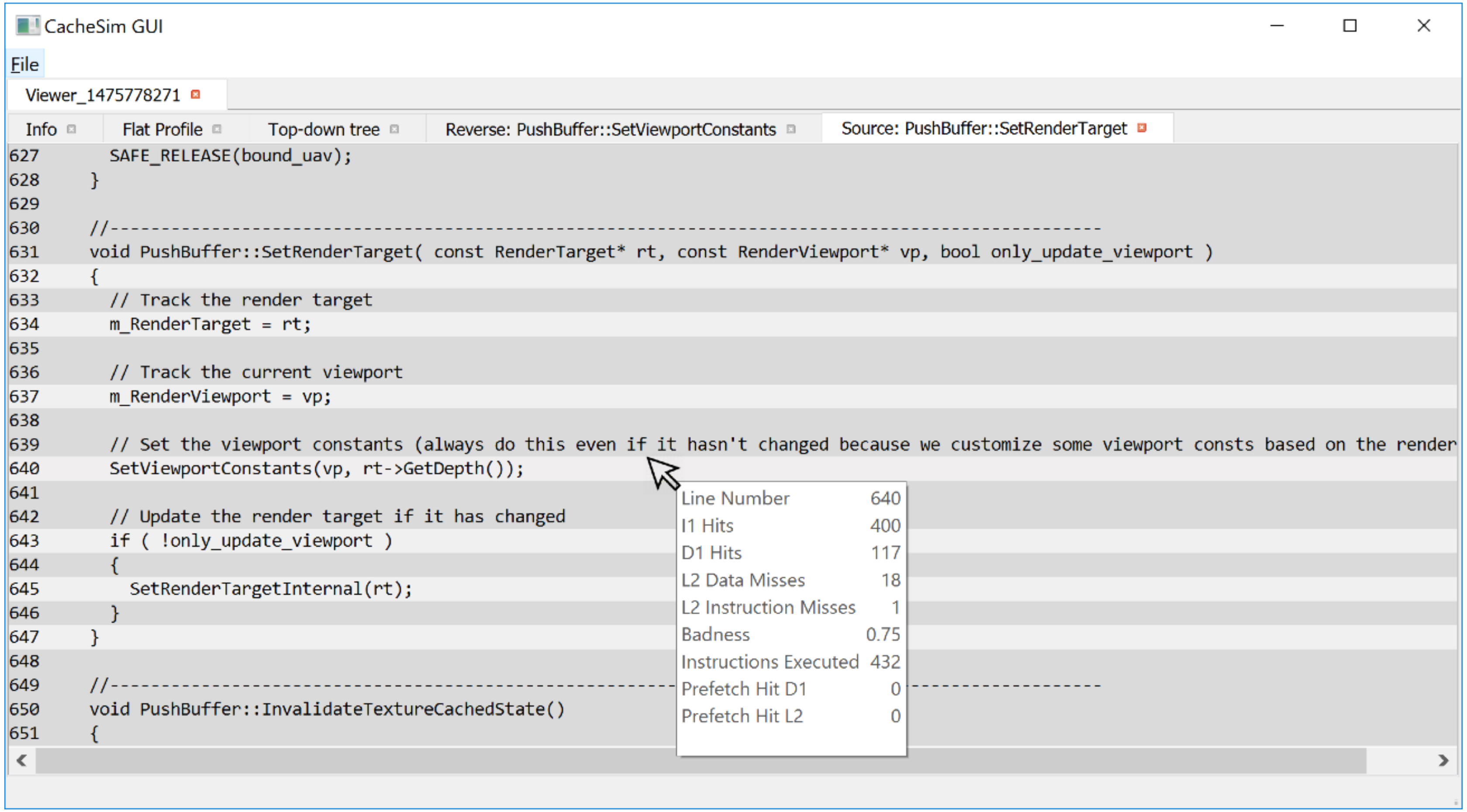
Type to filter...

Tooling – Reverse trees



Symbol	File	D1Hit	I1Hit	L2IMiss	L2DMiss	Badness	Instruct
PushBuffer::SetViewportConstants	c:\insom\core\sy...	14,818	29,800	42	317	3.26	
PushBuffer::SetRenderTarget	c:\insom\core\sy...	13,950	28,099	42	307	3.24	
RenderHandlerSetShadowMapTar...	c:\insom\core\sy...	2,551	5,092	0	96	1.75	
RenderThreadUpdate	c:\insom\core\sy...	2,551	5,092	0	96	1.75	
BaseThreadInitThunk		2,551	5,092	0	96	1.75	
CreateBloomBuffer	c:\insom\core\sy...	3,945	8,083	0	84	0.85	
RenderHandlerSetRenderTarget	c:\insom\core\sy...	1,298	2,568	40	48	0.87	
RenderHandlerCopyToDisplay		1,183	2,364	0	17	0.12	
RenderHandlerRadianceCapture	c:\insom\core\sy...	748	1,517	2	12	0.09	
RenderHandlerPrepareDepth	c:\insom\core\sy...	847	1,684	0	10	0.06	
RenderHandlerTemporalAaCaptur...	c:\insom\core\sy...	846	1,695	0	10	0.06	
RenderHandlerVelocityGenInit	c:\insom\core\sy...	859	1,709	0	10	0.06	
RenderHandlerApplyLighting	c:\insom\core\sy...	419	844	0	5	0.03	
RenderHandlerMotionBlur	c:\insom\core\sy...	413	844	0	5	0.03	
RenderHandlerToneMap	c:\insom\core\sy...	425	857	0	5	0.03	
RenderHandlerTemporalAa	c:\insom\core\sy...	416	842	0	5	0.03	
RenderThreadUpdate	c:\insom\core\sy...	432	855	0	5	0.03	
PushBuffer::SetUnorderedAccessViews	c:\insom\core\sy...	436	846	0	5	0.03	

Tooling – Source annotation


CacheSim GUI
File
Viewer_1475778271
Info
Flat Profile
Top-down tree
Reverse: PushBuffer::SetViewportConstants
Source: PushBuffer::SetRenderTarget

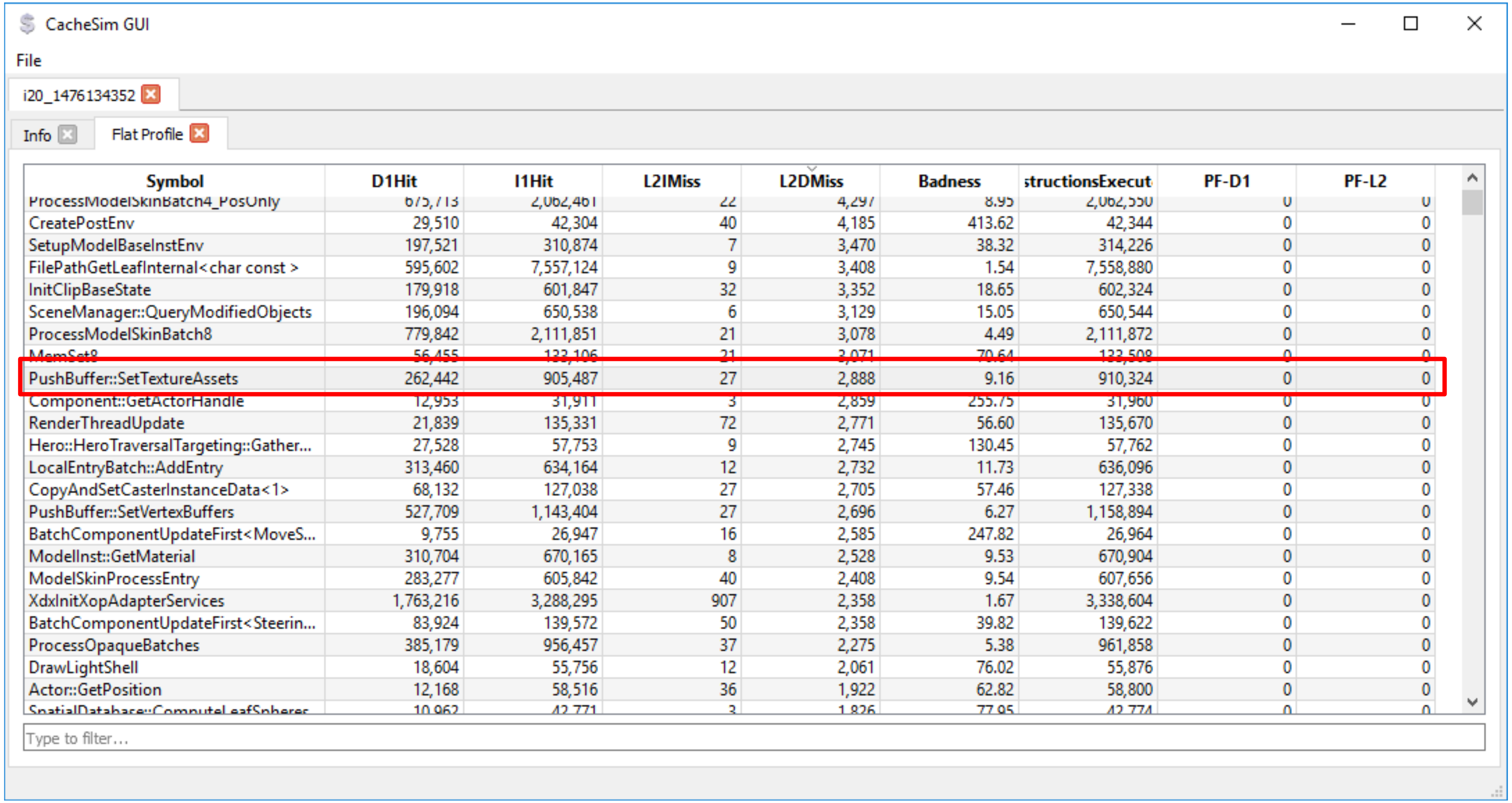
```
    SAFE_RELEASE(bound_uav);
  }

  //---------------------------------------------------------------------------------------------------
  void PushBuffer::SetRenderTarget( const RenderTarget* rt, const RenderViewport* vp, bool only_update_viewport )
  {
    // Track the render target
    m_RenderTarget = rt;

    // Track the current viewport
    m_RenderViewport = vp;

    // Set the viewport constants (always do this even if it hasn't changed because we customize some viewport consts based on the render
    SetViewportConstants(vp, rt->GetDepth());

    // Update the render target if it has changed
    if ( !only_update_viewport )
    {
      SetRenderTargetInternal(rt);
    }
  }

  //---------------------------------------------------------------------------------------------------
  void PushBuffer::InvalidateTextureCachedState()
  {
```

Line Number	640
I1 Hits	400
D1 Hits	117
L2 Data Misses	18
L2 Instruction Misses	1
Badness	0.75
Instructions Executed	432
Prefetch Hit D1	0
Prefetch Hit L2	0

Tooling considerations

- Want to make "obviously bad" data pop
  - Badness Factor = L2miss^2 / #instructions
- Worth investing in a bunch of different views
  - Maximizes value by allowing more analysis on dumps

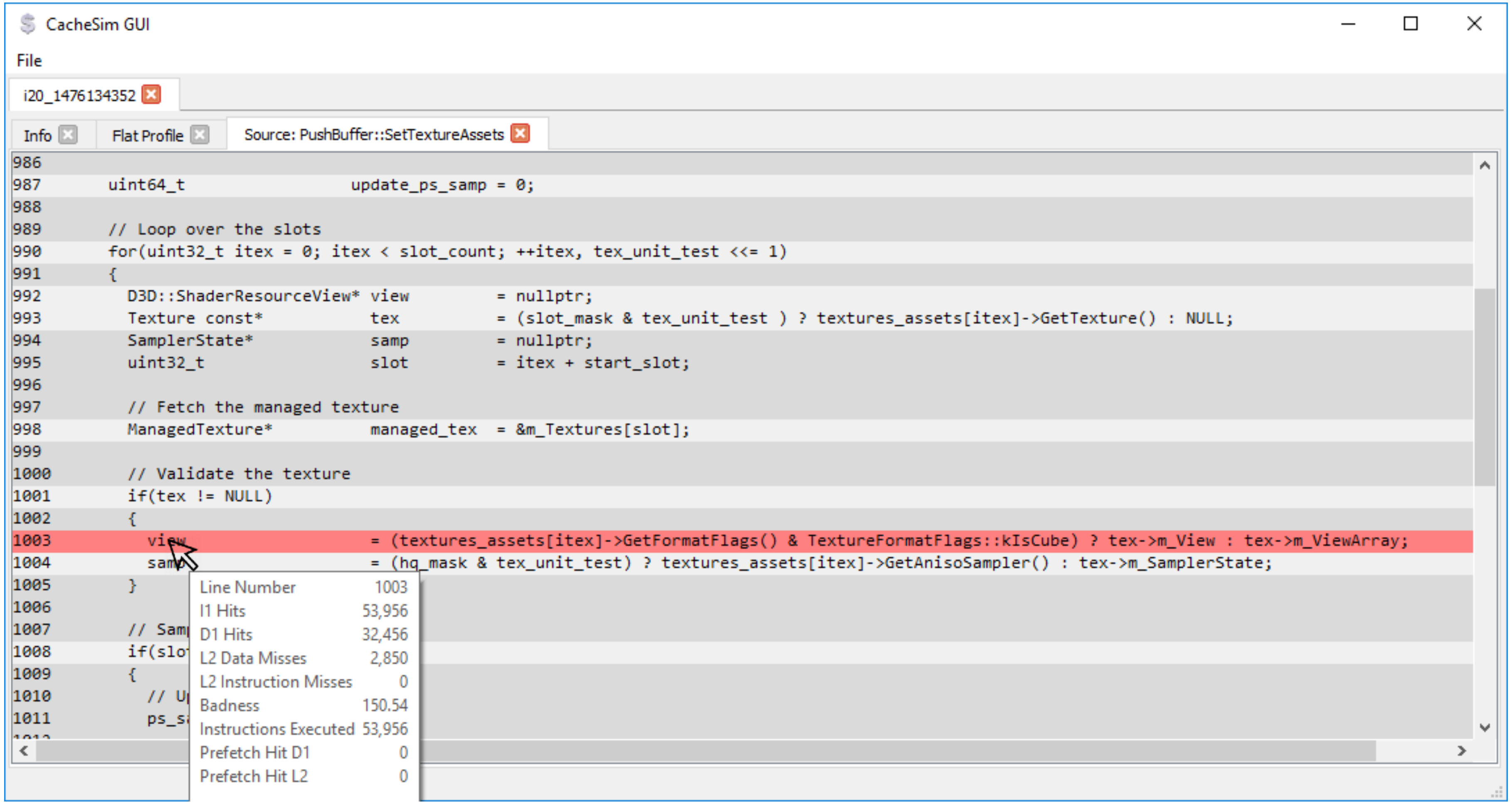
PushBuffer::SetTextureAssets revisited



Symbol	D1Hit	I1Hit	L2IMiss	L2DMiss	Badness	structionsExecut	PF-D1	PF-L2
ProcessModelSkinBatch4_PosOnly	675,713	2,062,461	22	4,297	8.95	2,062,550	0	0
CreatePostEnv	29,510	42,304	40	4,185	413.62	42,344	0	0
SetupModelBaseInstEnv	197,521	310,874	7	3,470	38.32	314,226	0	0
FilePathGetLeafInternal<char const >	595,602	7,557,124	9	3,408	1.54	7,558,880	0	0
InitClipBaseState	179,918	601,847	32	3,352	18.65	602,324	0	0
SceneManager::QueryModifiedObjects	196,094	650,538	6	3,129	15.05	650,544	0	0
ProcessModelSkinBatch8	779,842	2,111,851	21	3,078	4.49	2,111,872	0	0
MemSet8	56,455	133,106	21	3,071	70.64	133,508	0	0
PushBuffer::SetTextureAssets	262,442	905,487	27	2,888	9.16	910,324	0	0
Component::GetActorHandle	12,953	31,911	3	2,859	255.75	31,960	0	0
RenderThreadUpdate	21,839	135,331	72	2,771	56.60	135,670	0	0
Hero::HeroTraversalTargeting::Gather...	27,528	57,753	9	2,745	130.45	57,762	0	0
LocalEntryBatch::AddEntry	313,460	634,164	12	2,732	11.73	636,096	0	0
CopyAndSetCasterInstanceData<1>	68,132	127,038	27	2,705	57.46	127,338	0	0
PushBuffer::SetVertexBuffers	527,709	1,143,404	27	2,696	6.27	1,158,894	0	0
BatchComponentUpdateFirst<MoveS...	9,755	26,947	16	2,585	247.82	26,964	0	0
ModelInst::GetMaterial	310,704	670,165	8	2,528	9.53	670,904	0	0
ModelSkinProcessEntry	283,277	605,842	40	2,408	9.54	607,656	0	0
XdxInitXopAdapterServices	1,763,216	3,288,295	907	2,358	1.67	3,338,604	0	0
BatchComponentUpdateFirst<Steerin...	83,924	139,572	50	2,358	39.82	139,622	0	0
ProcessOpaqueBatches	385,179	956,457	37	2,275	5.38	961,858	0	0
DrawLightShell	18,604	55,756	12	2,061	76.02	55,876	0	0
Actor::GetPosition	12,168	58,516	36	1,922	62.82	58,800	0	0
SpatialDatabase::ComputeLeafSpheres	10,962	42,771	3	1,826	77.95	42,774	0	0

Type to filter…

PushBuffer::SetTextureAssets revisited



```

  uint64_t                 update_ps_samp = 0;

  // Loop over the slots
  for(uint32_t itex = 0; itex < slot_count; ++itex, tex_unit_test <<= 1)
  {
    D3D::ShaderResourceView* view         = nullptr;
    Texture const*           tex          = (slot_mask & tex_unit_test ) ? textures_assets[itex]->GetTexture() : NULL;
    SamplerState*            samp         = nullptr;
    uint32_t                 slot         = itex + start_slot;

    // Fetch the managed texture
    ManagedTexture*          managed_tex  = &m_Textures[slot];

    // Validate the texture
    if(tex != NULL)
    {
      view                   = (textures_assets[itex]->GetFormatFlags() & TextureFormatFlags::kIsCube) ? tex->m_View : tex->m_ViewArray;
      samp                   = (hq_mask & tex_unit_test) ? textures_assets[itex]->GetAnisoSampler() : tex->m_SamplerState;
    }

    // Samp
    if(slot
    {
      // Up
      ps_s
```

Line Number	1003
I1 Hits	53,956
D1 Hits	32,456
L2 Data Misses	2,850
L2 Instruction Misses	0
Badness	150.54
Instructions Executed	53,956
Prefetch Hit D1	0
Prefetch Hit L2	0

CacheSim Pros

- Gathers data for every memory access in the program
- Non-intrusive
- Non-encumbered
- Works on Windows
  - Deeply instruments even graphics drivers, OS calls down to syscall level
- Open Source
  - Can easily extend to more scenarios

CacheSim Cons

- Capture speed could be better
- Only works on Windows
  - Can still simulate a Jaguar cache for console workflows (ignore OS stuff)
- Not 100% hardware accurate (and can’t be)
  - Treats the CPU as an in-order CPU – no OOO scheduling
  - Must use virtual addresses to index cache (minor issue)
  - Array prefetchers not simulated (overly pessimistic about arrays)
  - MESI/Store forwarding buffers/...

Future

- Hardware prefetch simulation
- Non-temporal store simulation
- Speeding up captures
- Extensions

Thanks + Q & A

- Special thanks:
  - Mike Acton, Jonathan Adamczewski & Elan Ruskin
  - Mark Cerny
- http://github.com/insomniacgames/ig-cachesim
  - Play with it and submit your own improvements
- Get in touch
  - afredriksson@insomniacgames.com
  - @deplinenoise