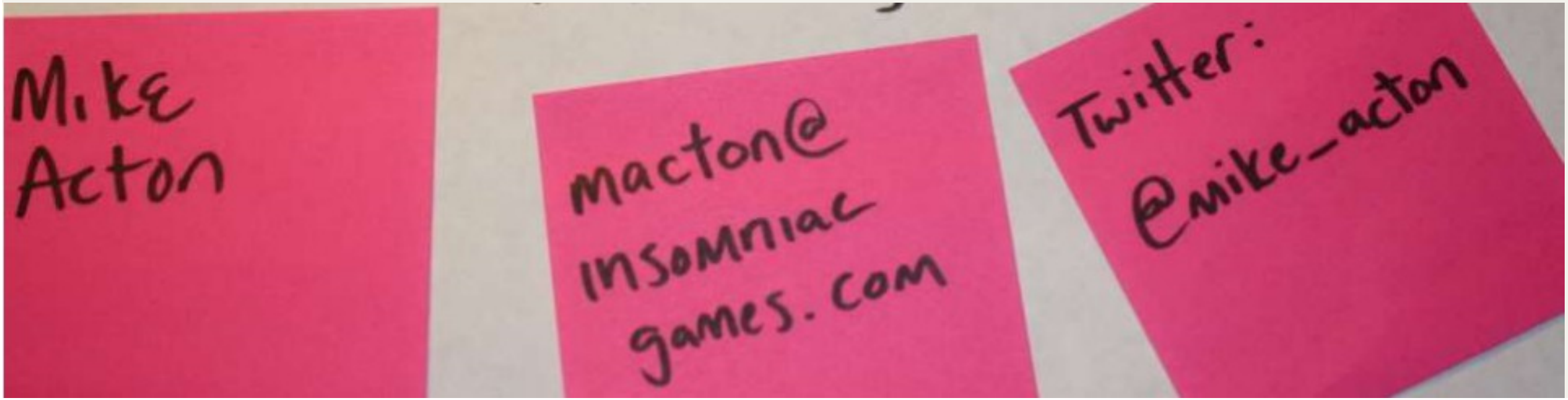
Code Clinic: How to Write Code the Compiler Can Actually Optimize

(2015)

Mike Acton | *Engine Director, Insomniac Games*



I don’t know anything about wood carving, but…

Different problems,
different scales,
different tools.

Tools scale

- Chainsaw
- Dremel
- Laser

Tools scale

- Chainsaw
- Dremel <- Compiler
- Laser

To make best use of a tool (e.g. compiler)

- Solve the part of the problem the tool can’t help with.
- Prepare the area that the tool can help with.
- Solve details missed by using the tool as intended.

Part 1:
Solve the part of the problem the tool can’t help with.

Approaches…

Approach #1

- Estimate resources available
- Triage based on cost and value estimates
- Collect data
- Adapt as cost and/or value predictions change
- AKA Engineering

Approach #2

- Don’t worry about resources
- Desperately scramble to fix when running out.
- *Generously* described as irrational
- Desperate scramble != “optimization”

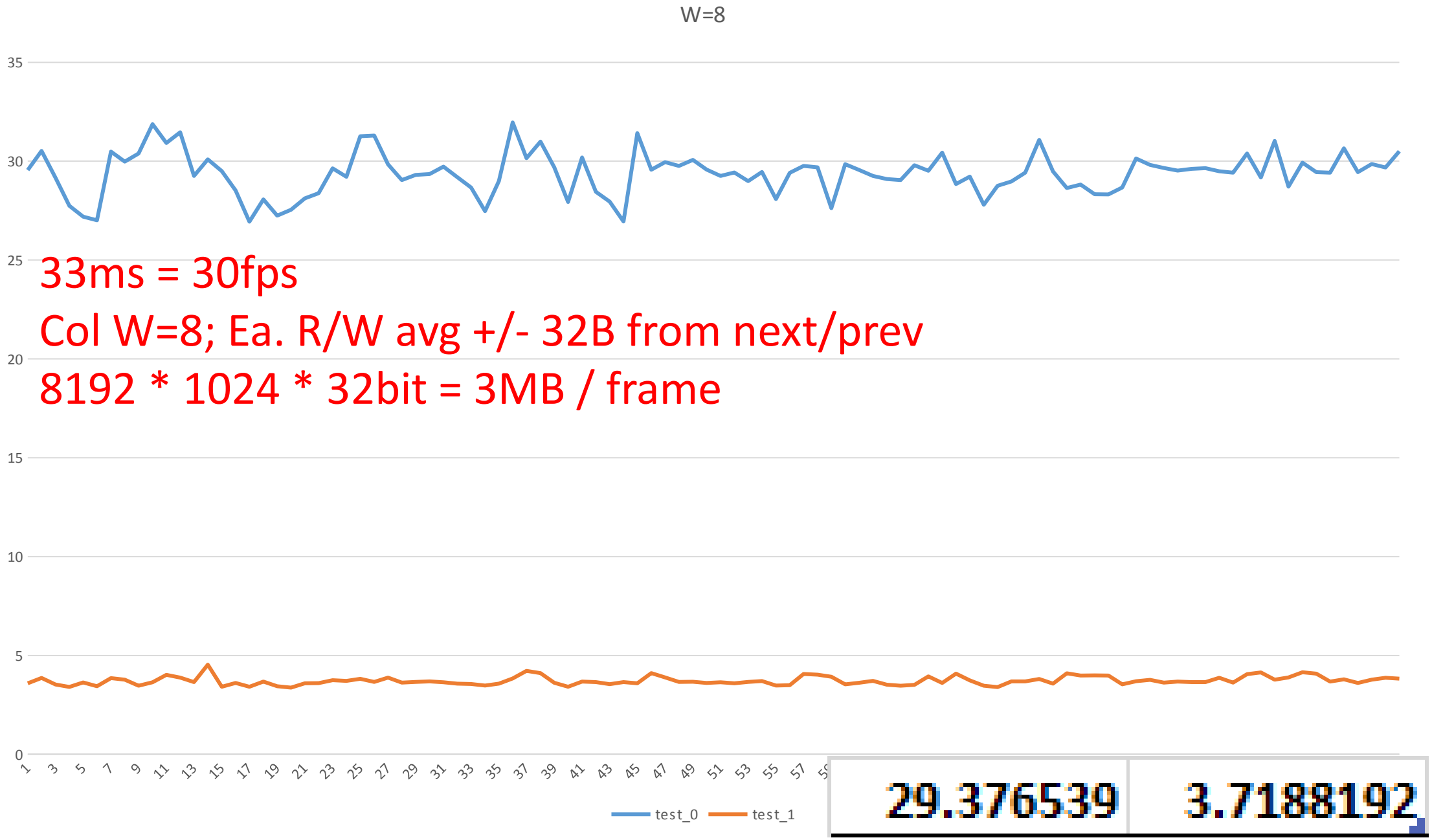
The problems are there even if you ignore them.

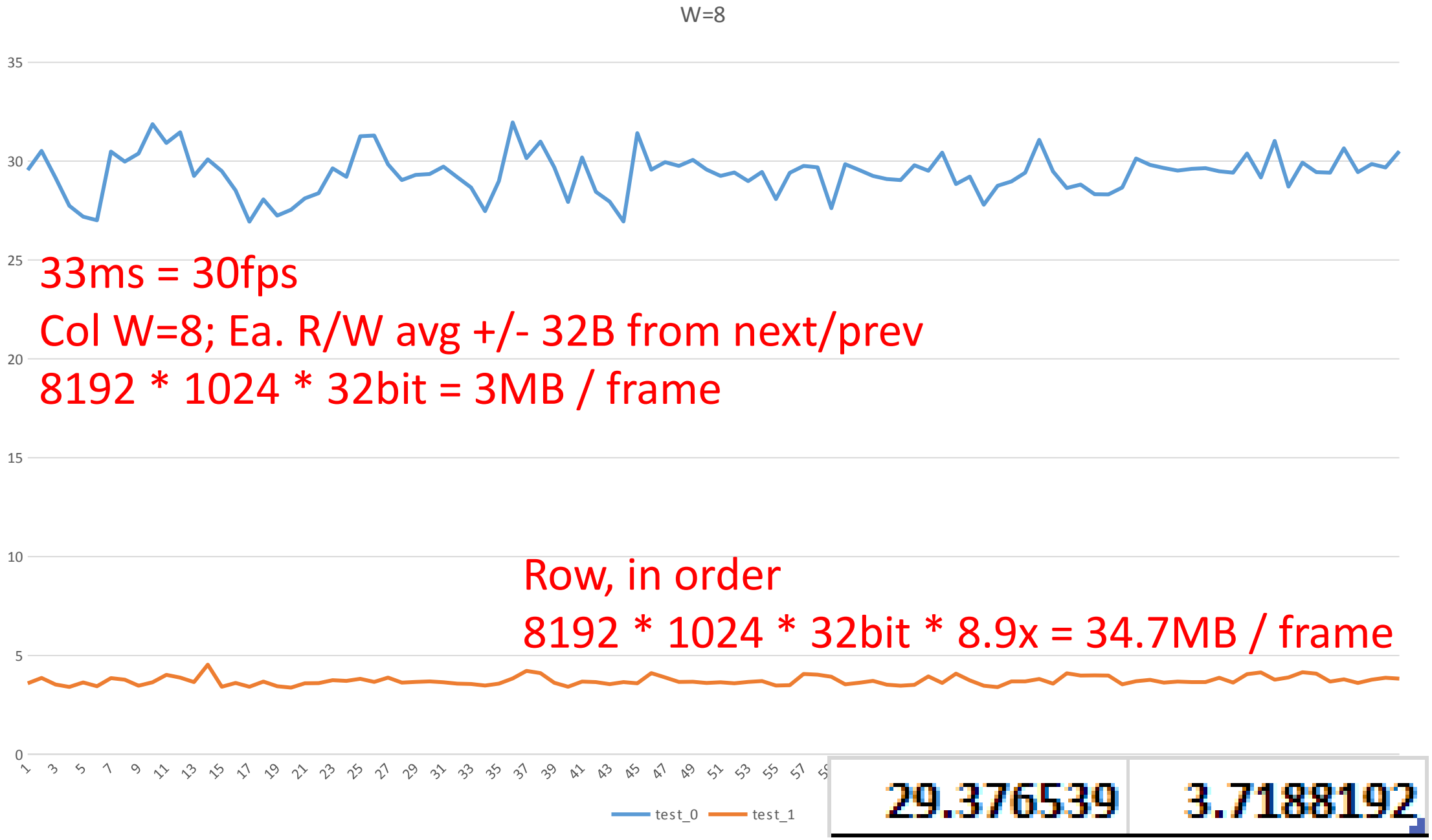
Estimate resources available

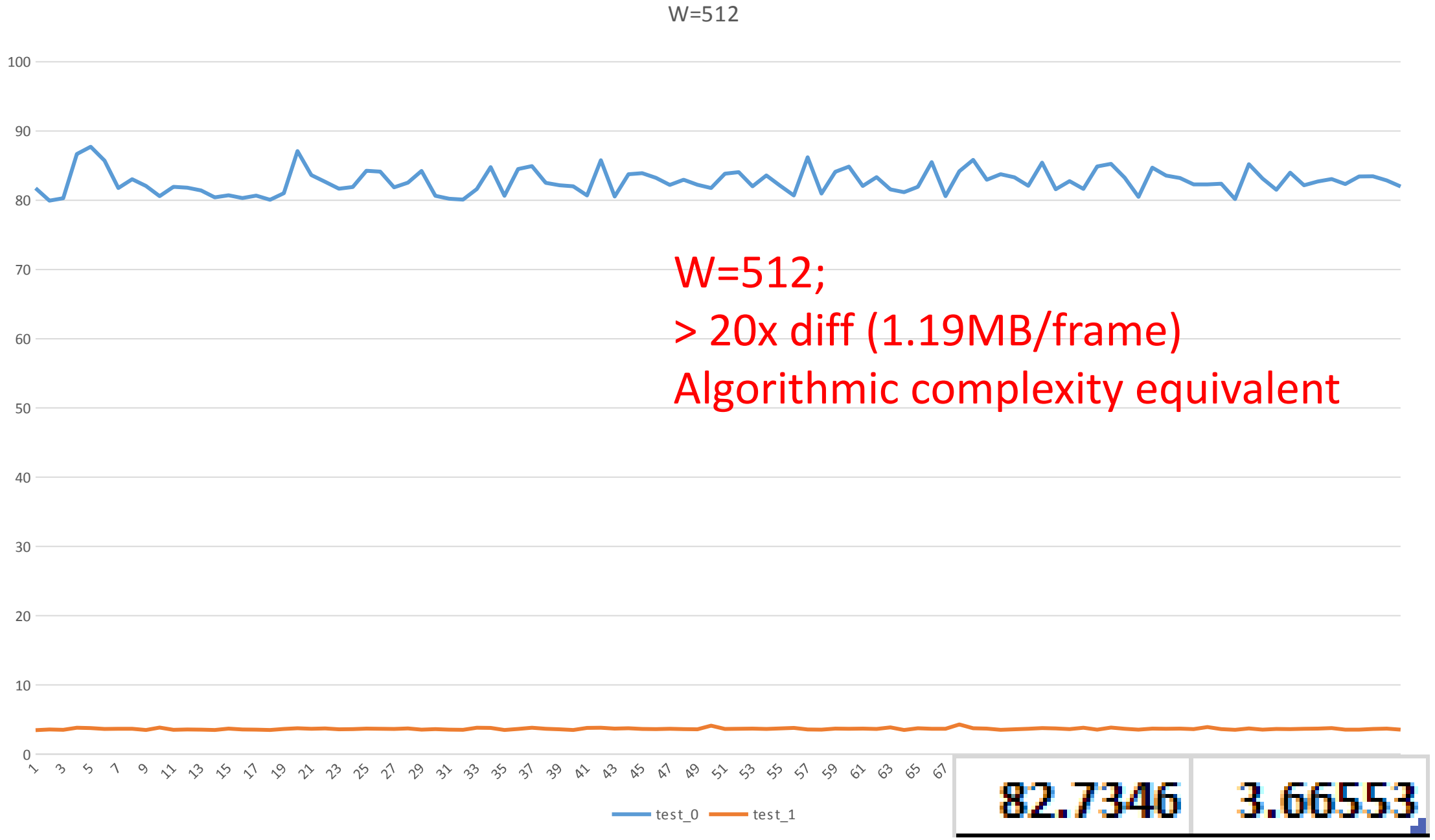
- What are the bottlenecks? i.e. most insufficient for the demand

Shell + Excel: test_0 vs. test_1 (W=2 -> W=512)

What can we infer?


W=8
35
30
25
20
15
10
5
0
33ms = 30fps
Col W=8; Ea. R/W avg +/- 32B from next/prev
8192 * 1024 * 32bit = 3MB / frame
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57
test_0
test_1
29.376539
3.7188192

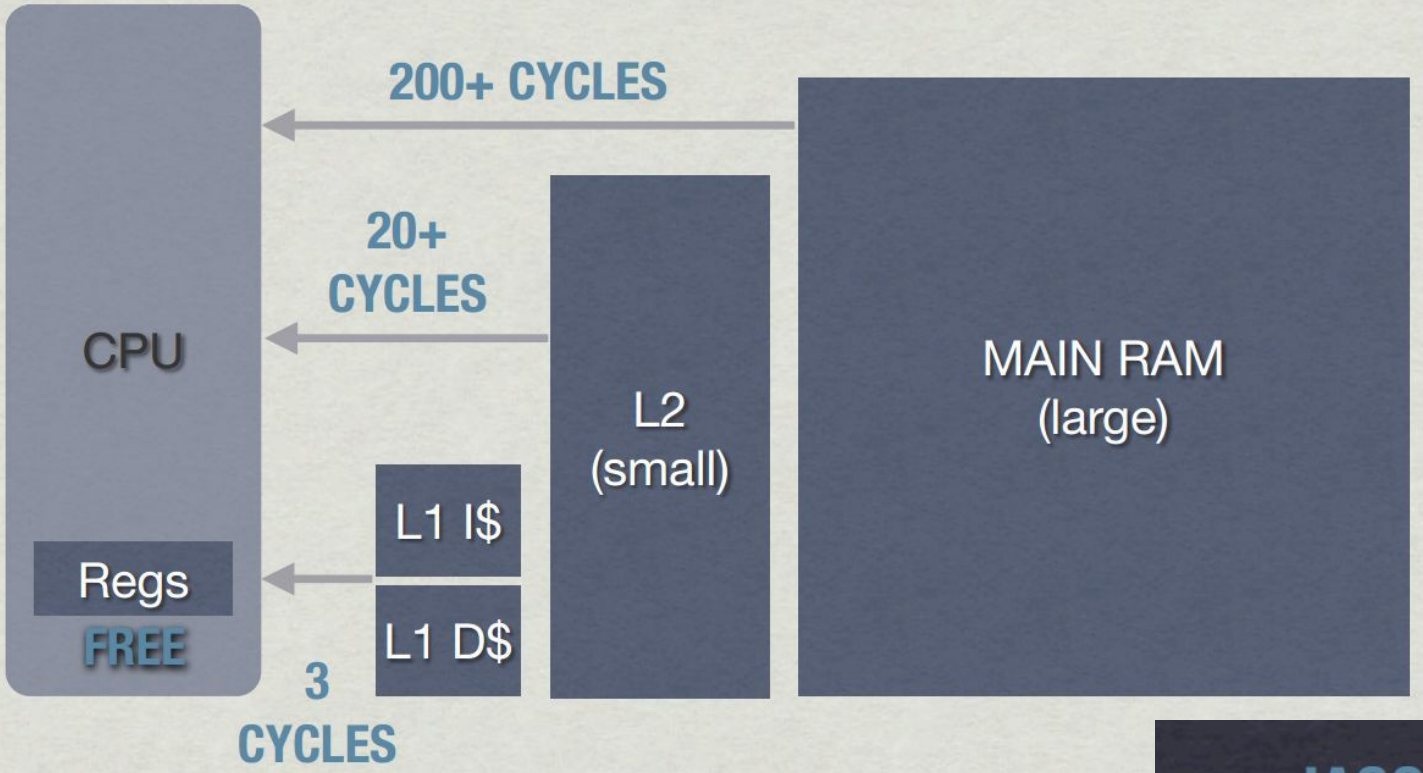

W=8
35
30
25
20
15
10
5
0
33ms = 30fps
Col W=8; Ea. R/W avg +/- 32B from next/prev
8192 * 1024 * 32bit = 3MB / frame
Row, in order
8192 * 1024 * 32bit * 8.9x = 34.7MB / frame
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57
test_0
test_1
29.376539
3.7188192


W=512
100
90
80
70
60
50
40
30
20
10
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
test_0
test_1
W=512;
> 20x diff (1.19MB/frame)
Algorithmic complexity equivalent
82.7346
3.66553


W=512
100
90
80
70
60
50
40
30
20
10
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
Gap = doing nothing.
test_0
test_1
82.7346
3.66553

Why?

Memory Caching



JASON GREGORY

LEAD PROGRAMMER

NAUGHTY DOG, INC.

http://www.gameenginebook.com/SINFO.pdf

By comparison…

(AMD Piledriver)

Instruction	Latency
SQRTSS/PS	13-15
VSQRTPS	14-15
SQRTSD/PD	24-26
VSQRTPD	24-26

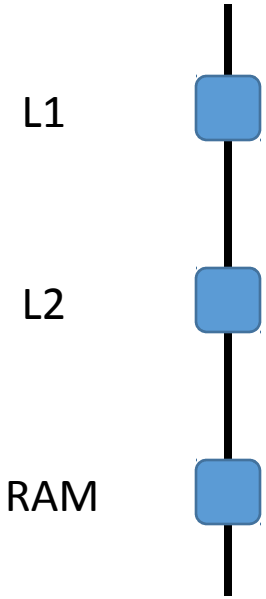
http://www.agner.org/optimize/instruction_tables.pdf

(AMD Piledriver)

Instruction	Latency
FSIN	60-146
FCOS	~154
FSINCOS	86-141
FPTAN	86-204
FPATAN	60-352

http://www.agner.org/optimize/instruction_tables.pdf

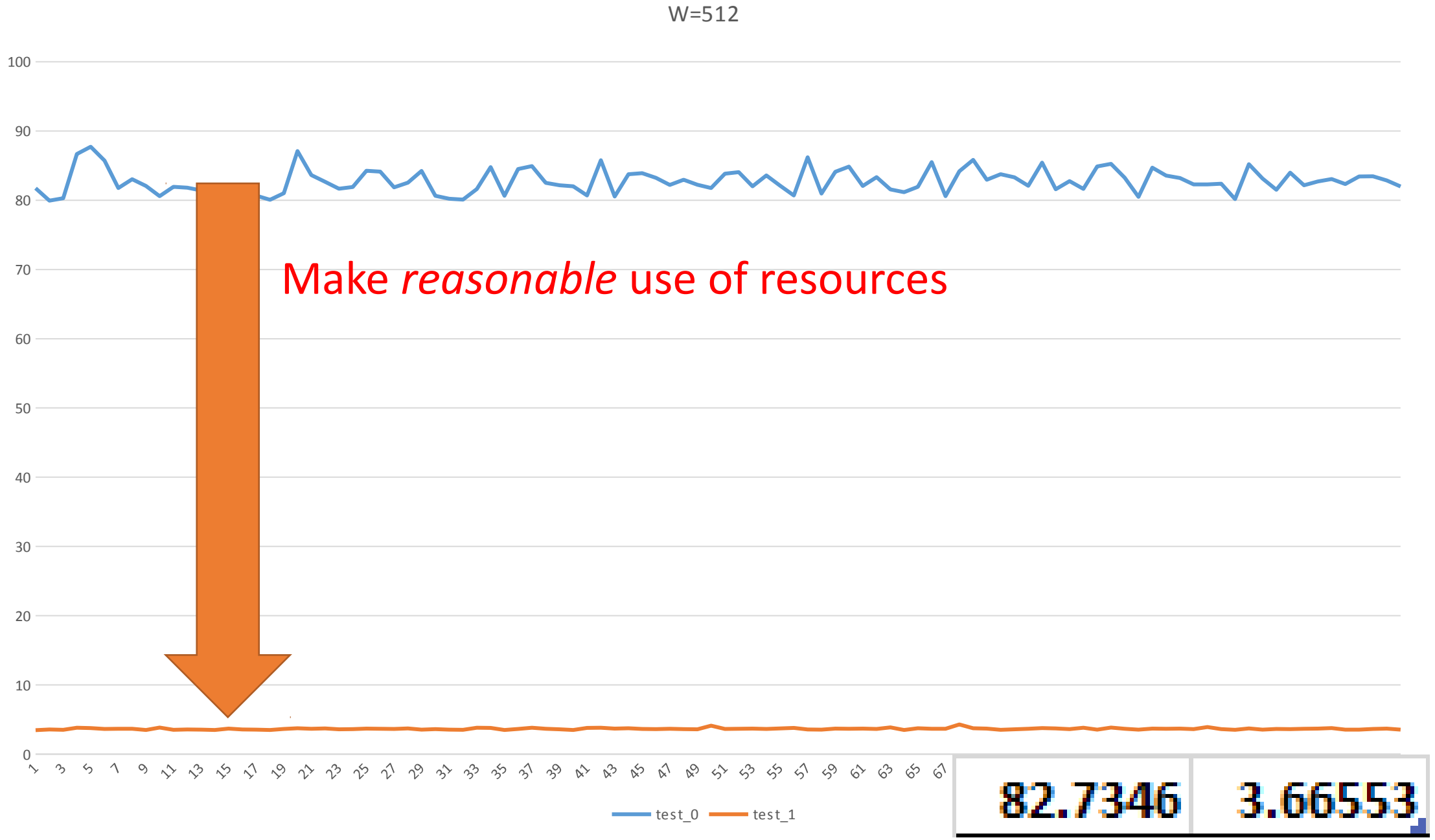
The Battle of North Bridge

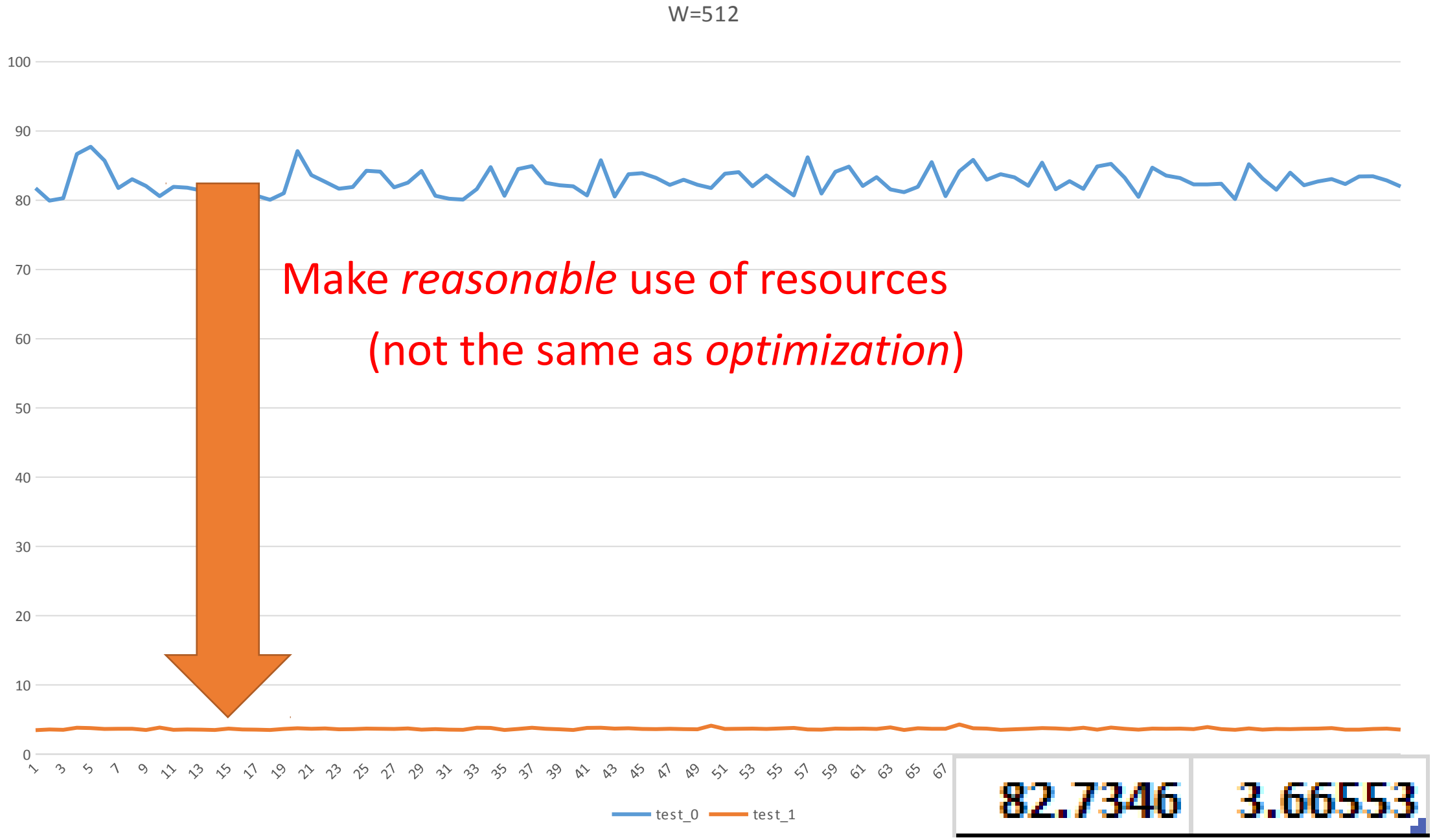


Verify data…

```
void
test_0( void )
{
    for (int x=0;x<W;x+=8)
    {
        for (int y=0;y<H;y++)
        {
            data[(W*y)+x]++;
        }
    }
}
```

Excel: W=64 (int32_t; mod 8)


W=512
100
90
80
70
60
50
40
30
20
10
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
Make *reasonable* use of resources
test_0
test_1
82.7346
3.66553


W=512
100
90
80
70
60
50
40
30
20
10
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
test_0
test_1
Make *reasonable* use of resources
(not the same as *optimization*)
82.7346
3.66553

Memory access is the most significant component.

Bottleneck: most insufficient for the demand

- Q: Is everything always all about memory access / cache?
- A: No. It’s about whatever the most scarce resources are.
- E.g. Disk access seeks; GPU draw counts; etc.

Let’s look at it another way…

Simple, obvious things to look for
+ Back of the envelope calculations
= Substantial wins

http://deplinenoise.wordpress.com/2013/12/28/optimizable-code/

```
class GameObject {
  float m_Pos[2];
  float m_Velocity[2];
  char m_Name[32];
  Model* m_Model;
  // ... other members ...
  float m_Foo;

  void UpdateFoo(float f)
  {
    float mag = sqrtf(
      m_Velocity[0] * m_Velocity[0] +
      m_Velocity[1] * m_Velocity[1]);
      m_Foo += mag * f;
  }
};
```

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

2 x 32bit read; same cache line = ~200

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Waste 56 bytes / 64 bytes

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Float mul, add = ~10

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Let’s assume callq is replaced. Sqrt = ~30

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Mul back to same addr; in L1; = ~3

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Read+add from new line
= ~200

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             # 4-byte Spill
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Waste 60 bytes / 64 bytes

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0         # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

90% waste!

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)             #
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0             # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9                                    Ef
        .cfi_endproc
```

Alternatively,
Only 10% capacity used*

* Not the same as “used well”, but we’ll start here.

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0         # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Time spent waiting for L2 vs. actual work

~10:1

```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0         # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Time spent waiting for L2 vs. actual work

~10:1

This is the compiler's space.

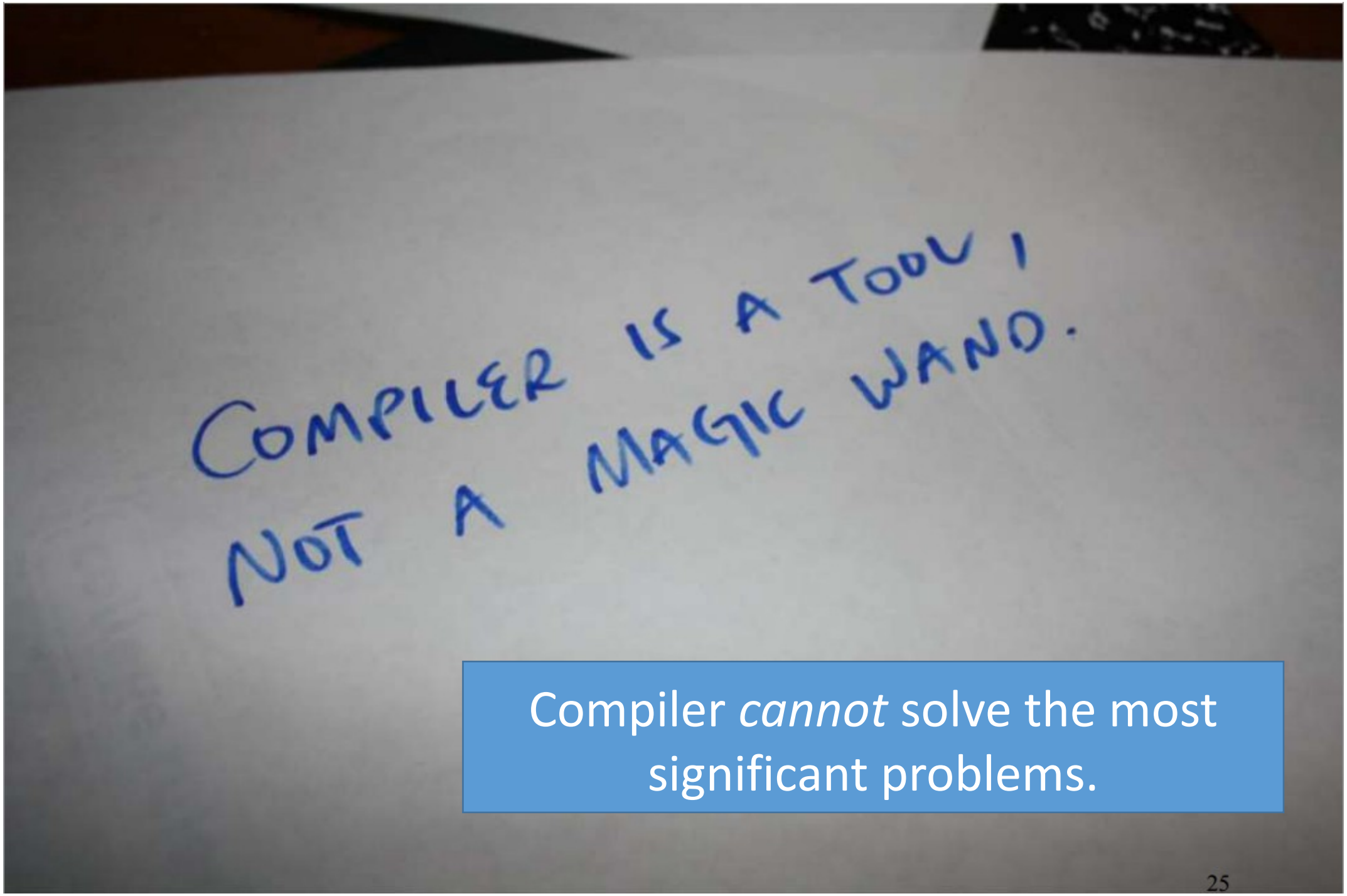
```
_ZN10GameObject9UpdateFooEf:                # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
# BB#0:
        pushq   %rbx
.Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
.Ltmp3:
        .cfi_def_cfa_offset 32
.Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        m
        a
        c
        m
        a
        m
        a
        popq    %rbx
        ret
.Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Time spent waiting for L2 vs. actual work

~10:1

This is the compiler's space.

Compiler can solve about 1-10% of the problem space.
i.e. the vast majority of problems are things the compiler can't reason about



Compiler *cannot* solve the most significant problems.

```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};

struct FooUpdateOut {
  float m_Foo;
};

void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i = 0; i < count; ++i) {
    float mag = sqrtf(
      in[i].m_Velocity[0] * in[i].m_Velocity[0] +
      in[i].m_Velocity[1] * in[i].m_Velocity[1]);
      out[i].m_Foo = in[i].m_Foo + mag * f;
  }
}
```

```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};
```

12 bytes x count(32) = 384 = 64 x 6

```
struct FooUpdateOut {
  float m_Foo;
};
```

4 bytes x count(32) = 128 = 64 x 2

```
void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i = 0; i < count; ++i) {
    float mag = sqrtf(
      in[i].m_Velocity[0] * in[i].m_Velocity[0] +
      in[i].m_Velocity[1] * in[i].m_Velocity[1]);
      out[i].m_Foo = in[i].m_Foo + mag * f;
  }
}
```

```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};
```

12 bytes x count(32) = 384 = 64 x 6

```
struct FooUpdateOut {
  float m_Foo;
};
```

4 bytes x count(32) = 128 = 64 x 2

```
void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i = 0; i < count; ++i) {
    float mag = sqrtf(
      in[i].m_Velocity[0] * in[i].m_Velocity[0] +
      in[i].m_Velocity[1] * in[i].m_Velocity[1]);
      out[i].m_Foo = in[i].m_Foo + mag * f;
  }
}
```

(6/32) = ~5.33 loop/cache line

```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};
```

12 bytes x count(32) = 384 = 64 x 6

```
struct FooUpdateOut {
  float m_Foo;
};
```

4 bytes x count(32) = 128 = 64 x 2

```
void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i = 0; i < count; ++i) {
    float mag = sqrtf(
      in[i].m_Velocity[0] * in[i].m_Velocity[0] +
      in[i].m_Velocity[1] * in[i].m_Velocity[1]);
      out[i].m_Foo = in[i].m_Foo + mag * f;
  }
}
```

(6/32) = ~5.33 loop/cache line

Sqrt + math = ~40 x 5.33 = 213.33 cycles/cache line

```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};
```

12 bytes x count(32) = 384 = 64 x 6

```
struct FooUpdateOut {
  float m_Foo;
};
```

4 bytes x count(32) = 128 = 64 x 2

```
void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i = 0; i < count; ++i) {
    float mag = sqrtf(
      in[i].m_Velocity[0] * in[i].m_Velocity[0] +
      in[i].m_Velocity[1] * in[i].m_Velocity[1]);
      out[i].m_Foo = in[i].m_Foo + mag * f;
  }
}
```

(6/32) = ~5.33 loop/cache line
Sqrt + math = ~40 x 5.33 = 213.33 cycles/cache line
+ streaming prefetch bonus

```
struct FooUpdateIn {
  float m_Velocity[2];
  float m_Foo;
};
```

12 bytes x count(32) = 384 = 64 x 6

```
struct FooUpdateOut {
  float m_Foo;
};
```

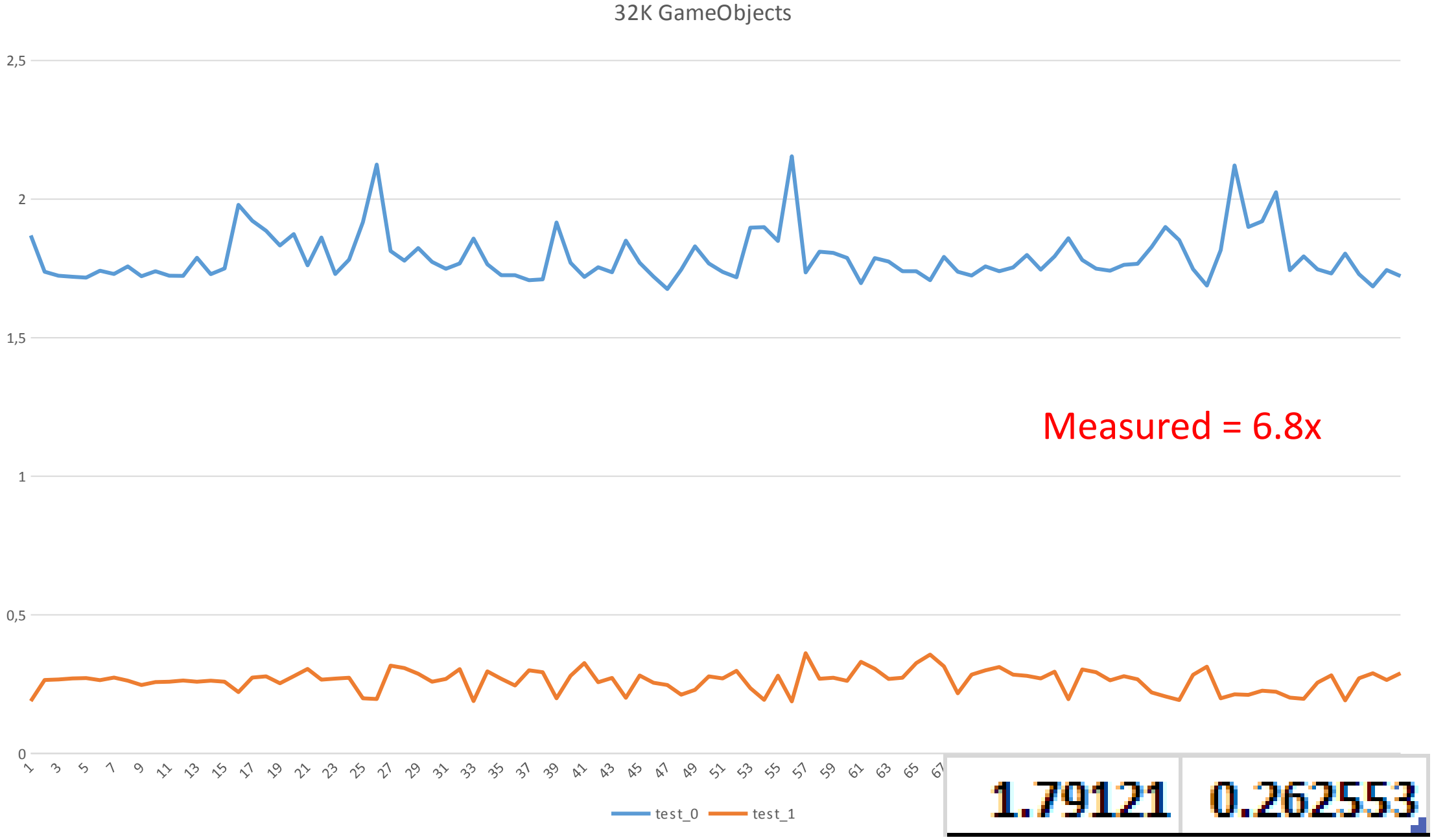
4 bytes x count(32) = 128 = 64 x 2

```
void UpdateFoos(const FooUpdateIn* in, size_t count, FooUpdateOut* out, float f)
{
  for (size_t i
    float mag =
      in[i].m_Ve
      in[i].m_Ve
      out[i].m_F
  }
}
```

Using cache line to capacity* =
Est. 10x speedup

* Used. Still not necessarily as efficiently as possible

(6/32) = ~5.33 loop/cache line
Sqrt + math = ~40 x 5.33 = 213.33 cycles/cache line
+ streaming prefetch bonus


32K GameObjects
2,5
2
1,5
1
0,5
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
test_0
test_1
Measured = 6.8x
1.79121
0.262553

Part 2:
Prepare the area that the tool can help with.

“The high-level design of the code generator”

- http://llvm.org/docs/CodeGenerator.html
- Instruction Selection
  - Map data/code to instructions that exist
- Scheduling and Formation
  - Give opportunity to schedule / estimate against data access latency
- SSA-based Machine Code Optimizations
  - Manual SSA form
- Register Allocation
  - Copy to/from locals in register size/types
- Prolog/Epilog Code Insertion
- Late Machine Code Optimizations
  - Pay careful attention to inlining
- Code Emission
  - Verify asm output regularly

“Compilers are good at applying mediocre optimizations hundreds of times.”
- @deplinenoise

Three easy tips to help compiler

- #1 Analyze one value
- #2 Hoist all loop-invariant reads and branches
- #3 Remove redundant transform redundancy

#1 Analyze one value

- Find any one value (or implicit value) you're interested in, anywhere.
- Printf it out over time. (Helps if you tag it so you can find it.)
- Grep your tty and paste into excel. (Sometimes a quick and dirty script is useful to convert to a more complex piece of data.)
- See what you see. You're bound to be surprised by something.

e.g. SoundSourceBaseComponent::BatchUpdate

```
if ( g_DebugTraceSoundSource )
 {
   Printf("#SS-01 %d m_CountSources\n”,i, component->m_CountSources);
 }
```

A histogram of m_CountSources turned up an interesting fact.

Bin	*Frequency*
0	20035
1	142
2	4
More	0
	20181

A histogram of m_CountSources turned up an interesting fact.

Bin	*Frequency*
0	20035
1	142
2	4
More	0
	20181

Approximately 0% of the sound source components have a source. (i.e. not playing yet)

A histogram of m_CountSources turned up an interesting fact.

Bin	*Frequency*
0	20035
1	142
2	4
More	0
	20181

Approximately 0% of the sound source components have a source. (i.e. not playing yet)

⇒ Switch to evaluating from source->component not component->source

⇒ 20K vs. 200 cache lines

#2 Hoist all loop-invariant reads and branches

- Don’t re-read member values or re-call functions when you *already* have the data.
- Hoist all loop-invariant reads and branches. Even super-obvious ones that should already be in registers (member fields especially.)

How is it used? What does it generate?

```
int
Foo::Bar( int count )
{
  int value = 0;
  for (int i=0;i<count;i++)
  {
    if ( m_NeedParentUpdate )
    {
      value++;
    }
  }
  return (value);
}
```

How is it used? What does it generate?

```
int
Foo::Bar( int count )
{
  int value = 0;
  for (int i=0;i<count;i++)
  {
    if ( m_NeedParentUpdate )
    {
      value++;
    }
  }
  return (value);
}
```

Equivalent to:
return m_NeedParentUpdate?count:0;

```
?Bar@Foo@@QEAAHH@Z PROC                 ; Foo::Bar, COMDAT

; 1696 :   int value = 0;

  00000 33 c0        xor     eax, eax

; 1697 :   for (int i=0;i<count;i++)

  00002 85 d2        test    edx, edx
  00004 7e 11        jle     SHORT $LN2@Bar

; 1696 :   int value = 0;

  00006 44 8a 01     mov     r8b, BYTE PTR [rcx]
  00009 8b ca        mov     ecx, edx
$LL9@Bar:

; 1698 :   {
; 1699 :     if ( m_NeedParentUpdate )

  0000b 45 84 c0     test    r8b, r8b
  0000e 74 02        je  SHORT $LN10@Bar

; 1700 :     {
; 1701 :       value++;

  00010 ff c0        inc     eax
$LN10@Bar:

; 1697 :   for (int i=0;i<count;i++)

  00012 48 ff c9     dec     rcx
  00015 75 f4        jne     SHORT $LL9@Bar
$LN2@Bar:

; 1702 :     }
; 1703 :   }
; 1704 :   return (value);
; 1705 : }
```

MSVC

MSVC

```
?Bar@Foo@@QEAAHH@Z PROC                 ; Foo::Bar, COMDAT

; 1696 :   int value = 0;

  00000 33 c0        xor     eax, eax

; 1697 :   for (int i=0;i<count;i++)

  00002 85 d2        test    edx, edx
  00004 7e 11        jle     SHORT $LN2@Bar

; 1696 :   int value = 0;

  00006 44 8a 01     mov     r8b, BYTE PTR [rcx]
  00009 8b ca        mov     ecx, edx
$LL9@Bar:

; 1698 :   {
; 1699 :     if ( m_NeedParentUpdate )

  0000b 45 84 c0     test    r8b, r8b
  0000e 74 02        je  SHORT $LN10@Bar

; 1700 :     {
; 1701 :       value++;

  00010 ff c0        inc     eax
$LN10@Bar:

; 1697 :   for (int i=0;i<count;i++)

  00012 48 ff c9     dec     rcx
  00015 75 f4        jne     SHORT $LL9@Bar
$LN2@Bar:

; 1702 :     }
; 1703 :   }
; 1704 :   return (value);
; 1705 : }
```

Re-read and re-test…

Increment and loop…

```
?Bar@Foo@@QEAAHH@Z PROC                 ; Foo::Bar, COMDAT

; 1696 :   int value = 0;

  00000 33 c0        xor     eax, eax

; 1697 :   for (int i=0;i<count;i++)

  00002 85 d2        test    edx, edx
  00004 7e 11        jle     SHORT $LN2@Bar

; 1696 :   int value = 0;

  00006 44 8a 01     mov     r8b, BYTE PTR [rcx]
  00009 8b ca        mov     ecx, edx
$LL9@Bar:

; 1698 :   {
; 1699 :     if ( m_NeedParentUpdate )

  0000b 45 84 c0     test    r8b, r8b
  0000e 74 02        je  SHORT $LN10@Bar

; 1700 :     {
; 1701 :       value++;

  00010 ff c0        inc     eax
$LN10@Bar:

; 1697 :   for (int i=0;i<count;i++)

  00012 48 ff c9     dec     rcx
  00015 75 f4        jne     SHORT $LL9@Bar
$LN2@Bar:

; 1702 :     }
; 1703 :   }
; 1704 :   return (value);
; 1705 : }
```

Re-read and re-test…

Increment and loop…

Visual Studio

Why?

Super-conservative aliasing rules…?
Member value might change?

What about something more aggressive…?


```
        .type   _ZN3Foo3BarEi,@function
_ZN3Foo3BarEi:                          # @_ZN3Foo3BarEi
        .cfi_startproc
# BB#0:
        xorl    %eax, %eax
        testl   %esi, %esi
        jle     .LBB1_2
# BB#1:                                 # %.lr.ph
        movzbl  (%rdi), %eax
        negl    %eax
        andl    %esi, %eax
.LBB1_2:
        ret
.Ltmp3:
        .size   _ZN3Foo3BarEi, .Ltmp3-_ZN3Foo3BarEi
        .cfi_endproc
```

What about something more aggressive…?


```
        .type   _ZN3Foo3BarEi,@function
_ZN3Foo3BarEi:                          # @_ZN3Foo3BarEi
        .cfi_startproc
# BB#0:
        xorl    %eax, %eax
        testl   %esi, %esi
        jle     .LBB1_2
# BB#1:                                 # %.lr.ph
        movzbl  (%rdi), %eax
        negl    %eax
        andl    %esi, %eax
.LBB1_2:
        ret
.Ltmp3:
        .size   _ZN3Foo3BarEi, .Ltmp3-_ZN3Foo3BarEi
        .cfi_endproc
```

Test once and return…

Okay, so what about…

```
int
Foo::Bar( int count )
{
  int value = 0;
  for (int i=0;i<count;i++)
  {
    if ( m_NeedParentUpdate )
    {
      value++;
    }
  }
  return (value);
}

int
Foo::Baz( int count )
{
  int value = 0;
  for (int i=0;i<count;i++)
  {
    if ( Bar(count) > 0 )
    {
      value++;
    }
  }
  return (value);
}
```

Okay, so what about…

```
int
Foo::Bar( int count )
{
  int value = 0;
  for (int i=0;i<count;i++)
  {
    if ( m_NeedParentUpdate )
    {
      value++;
    }
  }
  return (value);
}

int
Foo::Baz( int count )
{
  int value = 0;
  for (int i=0;i<count;i++)
  {
    if ( Bar(count) > 0 )
    {
      value++;
    }
  }
  return (value);
}
```

Equivalent to:

return m_NeedParentUpdate?count:0;

```
clang version 3.4 (tags/RELEASE_34/final)
Target: x86_64-unknown-linux-gnu
Thread model: posix
```

…well at least it inlined it?

```
        .type   _ZN3Foo3BazEi,@function
_ZN3Foo3BazEi:                          # @_ZN3Foo3BazEi
        .cfi_startproc
# BB#0:
        xorl    %eax, %eax
        testl   %esi, %esi
        jle     .LBB2_7
# BB#1:                                 # %.lr.ph
        xorl    %eax, %eax
        movl    %esi, %r8d
        andl    $-2, %r8d
        je      .LBB2_2
# BB#3:
        movl    %r8d, %ecx
        xorl    %r9d, %r9d
        .align  16, 0x90
.LBB2_4:                                # %vector.body
                                        # =>This Inner Loop Header: Depth=1
        movzbl  (%rdi), %edx
        negl    %edx
        testl   %esi, %edx
        setg    %dl
        movzbl  %dl, %edx
        addl    %edx, %eax
        addl    %edx, %r9d
        addl    $-2, %ecx
        jne     .LBB2_4
        jmp     .LBB2_5
.LBB2_2:
        xorl    %r8d, %r8d
        xorl    %r9d, %r9d
.LBB2_5:                                # %middle.block
        addl    %r9d, %eax
        cmpl    %esi, %r8d
        je      .LBB2_7
        .align  16, 0x90
.LBB2_6:                                # %_ZN3Foo3BarEi.exit
                                        # =>This Inner Loop Header: Depth=1
        movzbl  (%rdi), %ecx
        negl    %ecx
        testl   %esi, %ecx
        setg    %cl
        movzbl  %cl, %ecx
        addl    %ecx, %eax
        incl    %r8d
        cmpl    %r8d, %esi
        jne     .LBB2_6
.LBB2_7:                                # %._crit_edge
        ret
.Ltmp4:
```

```
  00000 33 c0        xor     eax, eax

; 1711 :   for (int i=0;i<count;i++)

  00002 85 d2        test    edx, edx
  00004 7e 25        jle     SHORT $LN2@Baz

  00006 44 8a 11     mov     r10b, BYTE PTR [rcx]
  00009 44 8b ca     mov     r9d, edx
  0000c 44 8b c2     mov     r8d, edx
$LL4@Baz:

  0000f 33 c9        xor     ecx, ecx
  00011 49 8b d1     mov     rdx, r9
$LL17@Baz:

; 1699 :     if ( m_NeedParentUpdate )

  00014 45 84 d2     test    r10b, r10b
  00017 74 02        je  SHORT $LN18@Baz

; 1701 :       value++;

  00019 ff c1        inc     ecx
$LN18@Baz:

; 1697 :   for (int i=0;i<count;i++)

  0001b 48 ff ca     dec     rdx
  0001e 75 f4        jne     SHORT $LL17@Baz

; 1713 :     if ( Bar(count) )

  00020 85 c9        test    ecx, ecx
  00022 74 02        je  SHORT $LN3@Baz

; 1715 :       value++;

  00024 ff c0        inc     eax
$LN3@Baz:

; 1711 :   for (int i=0;i<count;i++)

  00026 49 ff c8     dec     r8
  00029 75 e4        jne     SHORT $LL4@Baz
$LN2@Baz:
```

MSVC doesn’t fare any better…

```
int
Foo::Bar( int count )
{
  int  value       = 0;
  bool need_update = m_NeedParentUpdate;
  for (int i=0;i<count;i++)
  {
    if ( need_update )
    {
      value++;
    }
  }
  return (value);
}

int
Foo::Baz( int count )
{
  int  value       = 0;
  bool need_update = Bar(count) > 0;
  for (int i=0;i<count;i++)
  {
    if ( need_update )
    {
      value++;
    }
  }
  return (value);
}
```

Ghost reads and writes

Don’t re-read member values or re-call functions when you *already* have the data.

```
clang version 3.4 (tags/RELEASE_34/final)
Target: x86_64-unknown-linux-gnu
Thread model: posix
```

BAM!

```
_ZN3Foo3BazEi:                                  # @_ZN3Foo3BazEi
        .cfi_startproc
# BB#0:
        xorl    %eax, %eax
        testl   %esi, %esi
        jle     .LBB2_2
# BB#1:                                         # %.lr.ph
        movzbl  (%rdi), %ecx
        negl    %ecx
        xorl    %eax, %eax
        testl   %esi, %ecx
        cmovgl  %esi, %eax
.LBB2_2:                                        # %_ZN3Foo3BarEi.exit
        ret
```

```
; 1697 :   bool need_update = m_NeedParentUpdate;
  00000 44 8a 09     mov     r9b, BYTE PTR [rcx]
  00003 33 c0        xor     eax, eax
  00005 45 33 c0     xor     r8d, r8d

; 1698 :   for (int i=0;i<count;i++)
  00008 85 d2        test    edx, edx
  0000a 7e 0f        jle     SHORT $LN8@Baz

; 1711 :   int  value       = 0;
  0000c 8b ca        mov     ecx, edx
$LL10@Baz:

; 1700 :     if ( need_update )
  0000e 45 84 c9     test    r9b, r9b
  00011 74 03        je  SHORT $LN9@Baz
  00013 41 ff c0     inc     r8d
$LN9@Baz:

; 1698 :   for (int i=0;i<count;i++)
  00016 48 ff c9     dec     rcx
  00019 75 f3        jne     SHORT $LL10@Baz
$LN8@Baz:

; 1712 :   bool need_update = Bar(count) > 0;
  0001b 45 85 c0     test    r8d, r8d
  0001e 41 0f 9f c0  setg    r8b

; 1713 :   for (int i=0;i<count;i++)
  00022 85 d2        test    edx, edx
  00024 7e 0e        jle     SHORT $LN2@Baz
  00026 8b ca        mov     ecx, edx
$LL4@Baz:

; 1715 :     if ( need_update )
  00028 45 84 c0     test    r8b, r8b
  0002b 74 02        je  SHORT $LN3@Baz
  0002d ff c0        inc     eax
$LN3@Baz:

; 1713 :   for (int i=0;i<count;i++)
  0002f 48 ff c9     dec     rcx
  00032 75 f4        jne     SHORT $LL4@Baz
$LN2@Baz:

; 1720 :   return (value);
```

:(

```
int
Foo::Bar( int count )
{
  int value       = 0;
  bool need_update = m_NeedParentUpdate;
  if ( need_update )
  {
    for (int i=0;i<count;i++)
    {
      value++;
    }
  }
  return (value);
}

int
Foo::Baz( int count )
{
  int value       = 0;
  bool need_update = Bar(count) > 0;
  if ( need_update )
  {
    for (int i=0;i<count;i++)
    {
      value++;
    }
  }
  return (value);
}
```

Ghost reads and writes

Don’t re-read member values or re-call functions when you *already* have the data.

Hoist all loop-invariant reads and branches. Even super-obvious ones that should already be in registers.

```
?Baz@Foo@@QEAAHH@Z PROC                 ; Foo::Baz, COMDAT

; 1711 :   int  value       = 0;

  00000 33 c0        xor     eax, eax

; 1698 :   if ( need_update )

  00002 38 01        cmp     BYTE PTR [rcx], al
  00004 74 07        je  SHORT $LN3@Baz

; 1699 :   {
; 1700 :     for (int i=0;i<count;i++)

  00006 85 d2        test    edx, edx
  00008 7e 03        jle     SHORT $LN3@Baz

; 1712 :   bool need_update = Bar(count) > 0;
; 1713 :   if ( need_update )

  0000a 0f 4f c2     cmovg   eax, edx
$LN3@Baz:

; 1714 :   {
; 1715 :     for (int i=0;i<count;i++)
; 1716 :     {
; 1717 :       value++;
; 1718 :     }
; 1719 :   }
; 1720 :   return (value);
; 1721 : }

  0000d c3       ret     0
?Baz@Foo@@QEAAHH@Z ENDP                 ; Foo::Baz
```

:)

```
?Baz@Foo@@QEAAHH@Z PROC                 ; Foo::Baz, COMDAT

; 1711 :   int  value       = 0;

  00000 33 c0        xor     eax, eax

; 1698 :   if ( need_update )

  00002 38 01        cmp     BYTE PTR [rcx], al
  00004 74 07        je  SHORT $LN3@Baz

; 1699 :   {
; 1700 :     for (int i=0;i<count;i++)

  00006 85 d2        test    edx, edx
  00008 7e 03        jle     SHORT $LN3@Baz

; 1712 :   bool need_update = Bar(count) > 0;
; 1713 :   if ( need_update )

  0000a 0f 4f c2     cmovg   eax, edx
$LN3@Baz:

; 1714 :   {
; 1715 :     for (int i=0;i<count;i++)
; 1716 :     {
; 1717 :       value++;
; 1718 :     }
; 1719 :   }
; 1720 :   return (value);
; 1721 : }

  0000d c3       ret     0
?Baz@Foo@@QEAAHH@Z ENDP                 ; Foo::Baz
```

Visual Studio

:)

A bit of unnecessary branching, but more-or-less equivalent.

#3 Remove redundant transform redundancy

- Often, especially with small functions, there is not enough context to know when and under what conditions a transform will happen.
- It’s easy to fall into the over-generalization trap
- It’s only real cases, with real data, we’re concerned with.

A little gem…

For instance, I found this in my random search:
X:\core\devel\code\Core\Tools\SceneEditor\ThumbnailMode.cpp

```
if ( m_Delete )
{
  for ( int i = 0; i < m_DeleteFiles.GetCount(); ++i )
  {
    wchar_t* cleanup_file = (wchar_t*) m_DeleteFiles.GetAt( i );
    char file[ kFilePathLenMax ] = "";
    if ( wcstombs_s( NULL, file, ARRAY_SIZE( file ), cleanup_file, _TRUNCATE ) == 0 )
    {
      FileDelete( file );
    }
  }
}
```

For instance, I found this in my random search:
X:\core\devel\code\Core\Tools\SceneEditor\ThumbnailMode.cpp

```
if ( m_Delete )
{
  for ( int i = 0; i < m_DeleteFiles.GetCount(); ++i )
  {
    wchar_t* cleanup_file = (wchar_t*) m_DeleteFiles.GetAt( i );
    char file[ kFilePathLenMax ] = "";
    if ( wcstombs_s( NULL, file, ARRAY_SIZE( file ), cleanup_file, _TRUNCATE ) == 0 )
    {
      FileDelete( file );
    }
  }
}
```

wchar_t* -> char*

Suspicious. I know we don't handle wide char conversion consistently.

So inside:

```
bool FileDelete( const char* file_path )
```

So inside:

```
bool FileDelete( const char* file_path )
```

Here, there's a call to:

```
#if defined(__WIN__) || defined(__DURANGO__)
  WCHAR wide_file_path[kFilePathLenMax];
#if defined(__WIN__)
  bool success = FilePathPrepForWinLongUNC( file_path, wide_file_path, ARRAY_SIZE(wide_file_path)
#else
  bool success = FilePathPrepForWin( file_path, wide_file_path, ARRAY_SIZE(wide_file_path) );
#endif // #else
```

So inside:

```
bool FileDelete( const char* file_path )
```

Here, there's a call to:

```
#if defined(__WIN__) || defined(__DURANGO__)
  WCHAR wide_file_path[kFilePathLenMax];
#if defined(__WIN__)
  bool success = FilePathPrepForWinLongUNC( file_path, wide_file_path, ARRAY_SIZE(wide_file_path)
#else
  bool success = FilePathPrepForWin( file_path, wide_file_path, ARRAY_SIZE(wide_file_path) );
#endif // #else
```

wchar_t* -> char* -> wchar_t*

it takes the char* we created and converts it right back to a whar*

Let's look at where it's written then…

```
bool ThumbnailMode::AddDeleteFile( const char* file )
{
  wchar_t* cleanup_file = (wchar_t*) m_DeleteFiles.AllocBack();
  if ( cleanup_file == NULL )
  {
    return false;
  }

  if ( mbstowcs_s(NULL, cleanup_file, kFilePathLenMax, file, _TRUNCATE ) != 0 )
  {
    m_DeleteFiles.PopBack();
    return false;
  }
```

Let's look at where it's written then…

```
bool ThumbnailMode::AddDeleteFile( const char* file )
{
  wchar_t* cleanup_file = (wchar_t*) m_DeleteFiles.AllocBack();
  if ( cleanup_file == NULL )
  {
    return false;
  }

  if ( mbstowcs_s(NULL, cleanup_file, kFilePathLenMax, file, _TRUNCATE ) != 0 )
  {
    m_DeleteFiles.PopBack();
    return false;
  }
```

What’s happening?

- We convert from char* to wchar* to store it in m_DeleteFiles
- …So we can convert it from wchar* to char* to give to FileDelete
- …So we can convert it from char* to whcar* to give it to Windows DeleteFile.

Where does the char* come from in the first place?

There's in fact only one call site:

```
else if ( strstr( "-list", parameter ) != NULL )
{
  ++i;
  if ( i < argc )
  {
    const char* list = argv[ i ];
    if ( renderer.AddDeleteFile( list ) )
    {
      if ( !ParseThumbnailFile( list, renderer ) )
      {
        Printf( "\n * Failed to parse list file!\n" );
        return false;
      }
    }
```

There's in fact only one call site:

```
else if ( strstr( "-list", parameter ) != NULL )
{
  ++i;
  if ( i < argc )
  {
    const char* list = argv[ i ];
    if ( renderer.AddDeleteFile( list ) )
    {
      if ( !ParseThumbnailFile( list, renderer ) )
      {
        Printf( "\n * Failed to parse list file!\n" );
        return false;
      }
    }
```

Comes from argv. Never needed to touch the memory!

Which turned out to be a command line parameter that was never used, anywhere.

Which brings us back to the wood carving analogy…

Part 3:
Solve details missed by using the tool as intended.

Optimization begins.

See: e.g.

SIMD at Insomniac Games: How We Do the Shuffle

Andreas Fredriksson | Lead Engine Programmer, Insomniac Games

Location: Room 305, South Hall

Date: Thursday, March 5

Time: 10:00am - 11:00am

Format: Session

Track: Programming

Pass Type: All Access Pass, Main Conference Pass - **Get your pass now!**

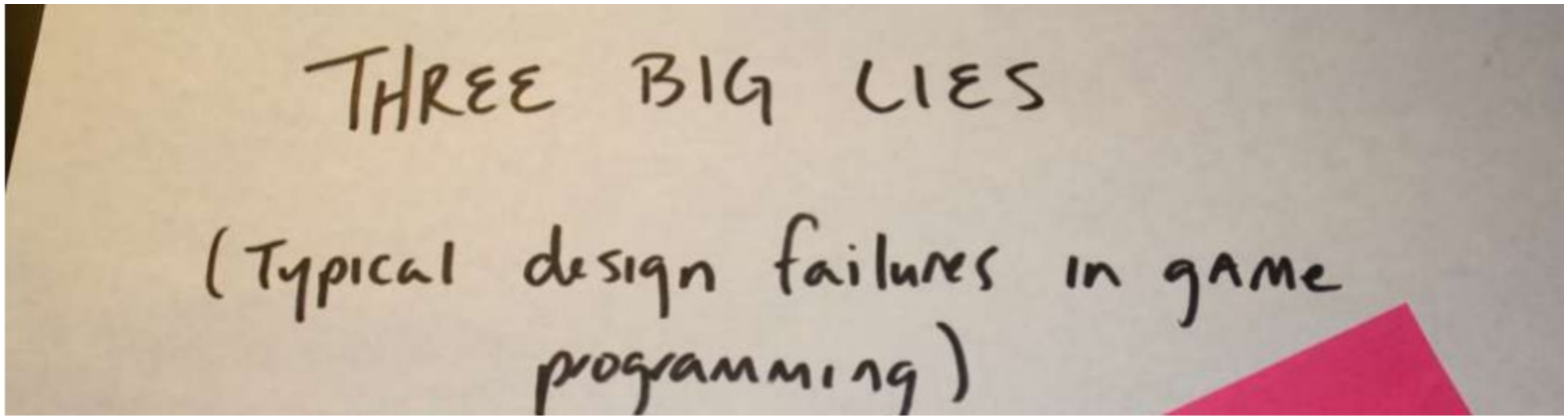
Vault Recording: Video

Audience Level: All

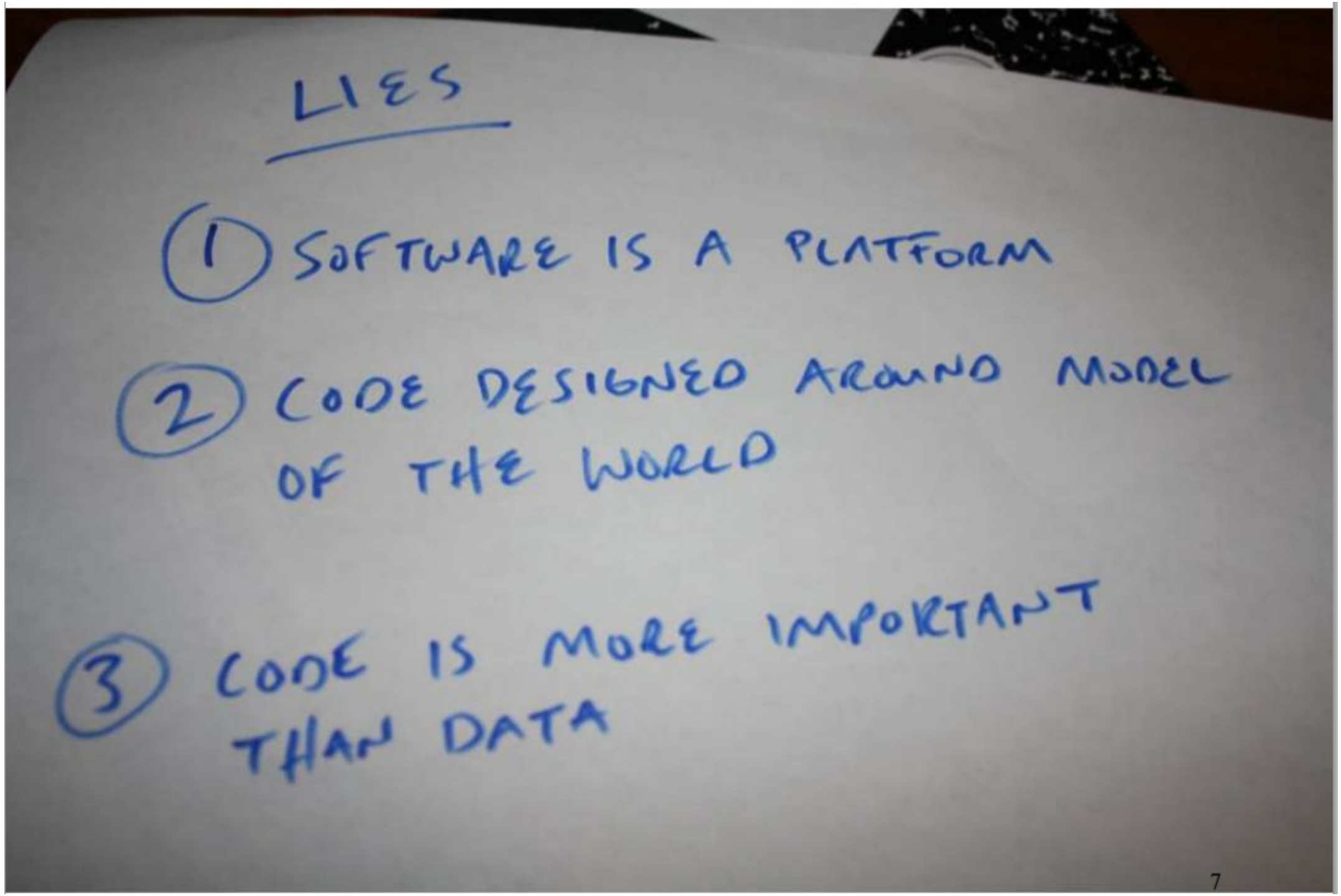
Part 4: Practice

If you don’t practice,
you can’t do it when it matters.

Practice
https://oeis.org/A000081



http://www.insomniacgames.com/three-big-lies-typical-design-failures-in-game-programming-gdc10/


LIES
1 SOFTWARE IS A PLATFORM
2 CODE DESIGNED AROUND MODEL OF THE WORLD
3 CODE IS MORE IMPORTANT THAN DATA