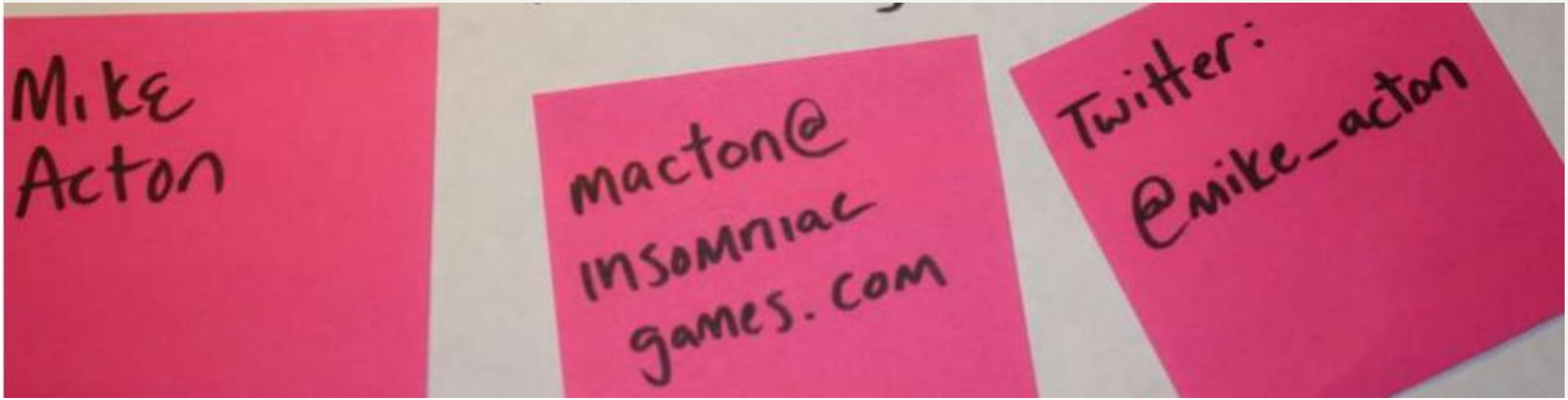
Code Clinic: How to Write Code the Compiler Can Actually Optimize

Mike Acton | *Engine Director, Insomniac Games*



Or…
Yet another privileged CIS white male in the AAA space talking abut data.

What is good code?

Our role is not to write "good" code. Our role is to solve our problems well.

With fixed hardware resources, that often means reducing waste or at least having the potential to reduce waste (i.e. optimizable) so that we can solve bigger and more interesting problems in the same space.

"Good" code in that context is the code that was written based on a rational and reasoned analysis of the actual problems that need solving, hardware resources, and available production time.

i.e. At the very least not using the "pull it out your ass" design method combined with a goal to "solve all problems for everyone, everywhere."

Can’t the compiler do it?

A little review…

(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

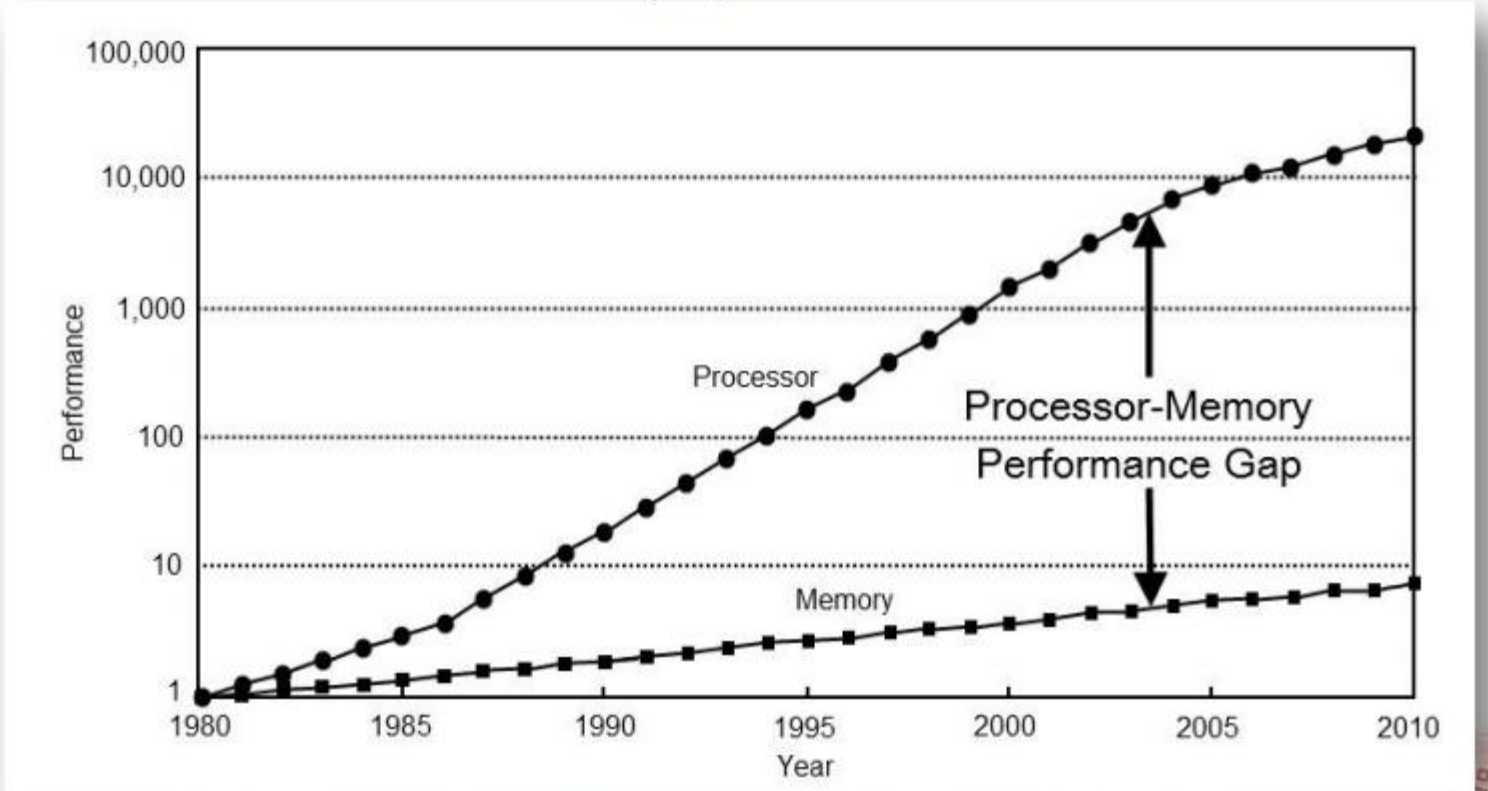
(AMD Piledriver)

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

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

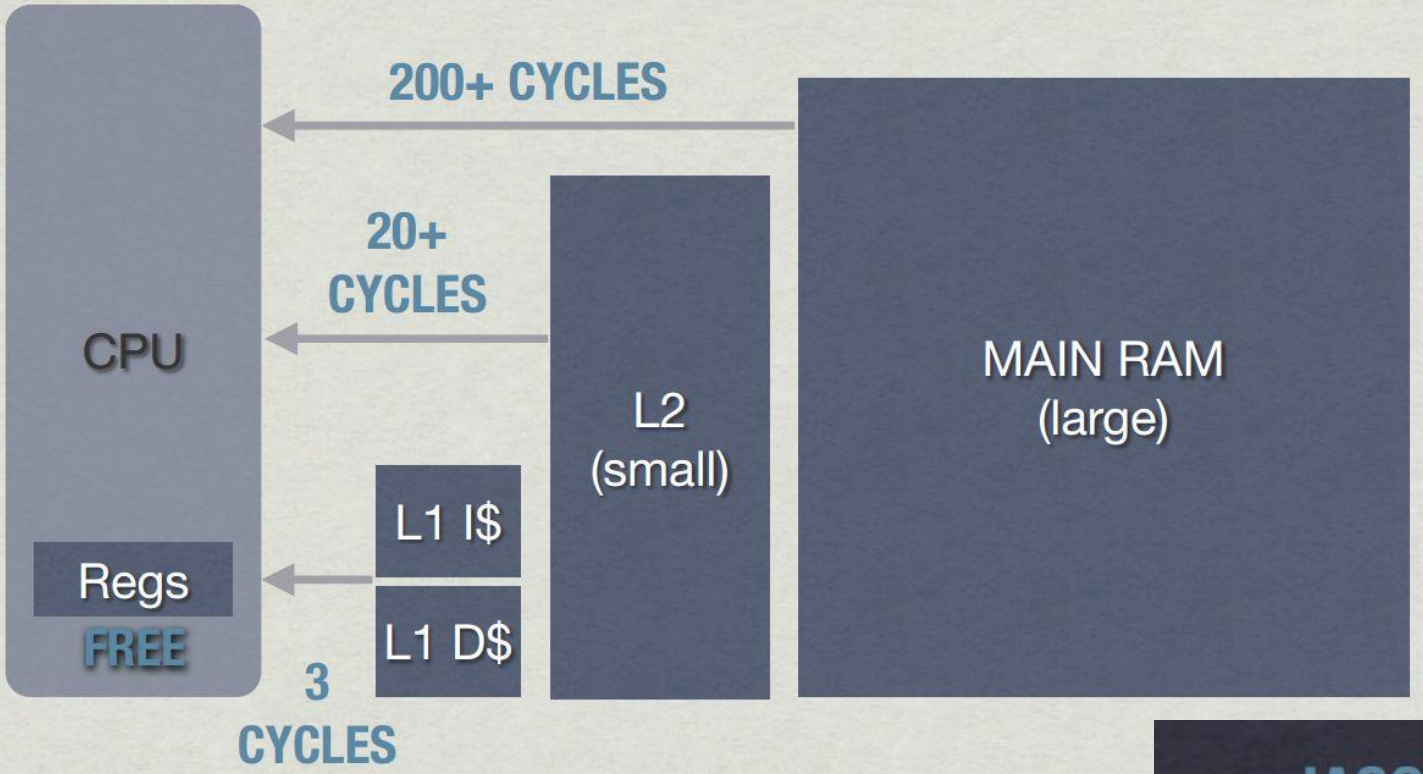
SCEE Research & Development

CPU/Memory performance



http://research.scee.net/files/presentations/gcapaustralia09/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf

Memory Caching



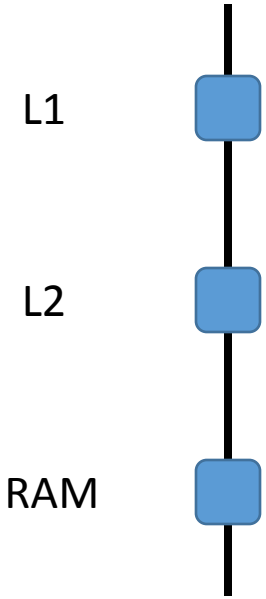
JASON GREGORY

LEAD PROGRAMMER

NAUGHTY DOG, INC.

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

The Battle of North Bridge



L2 cache misses/frame

(Most significant component)

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
```

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)
        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
```

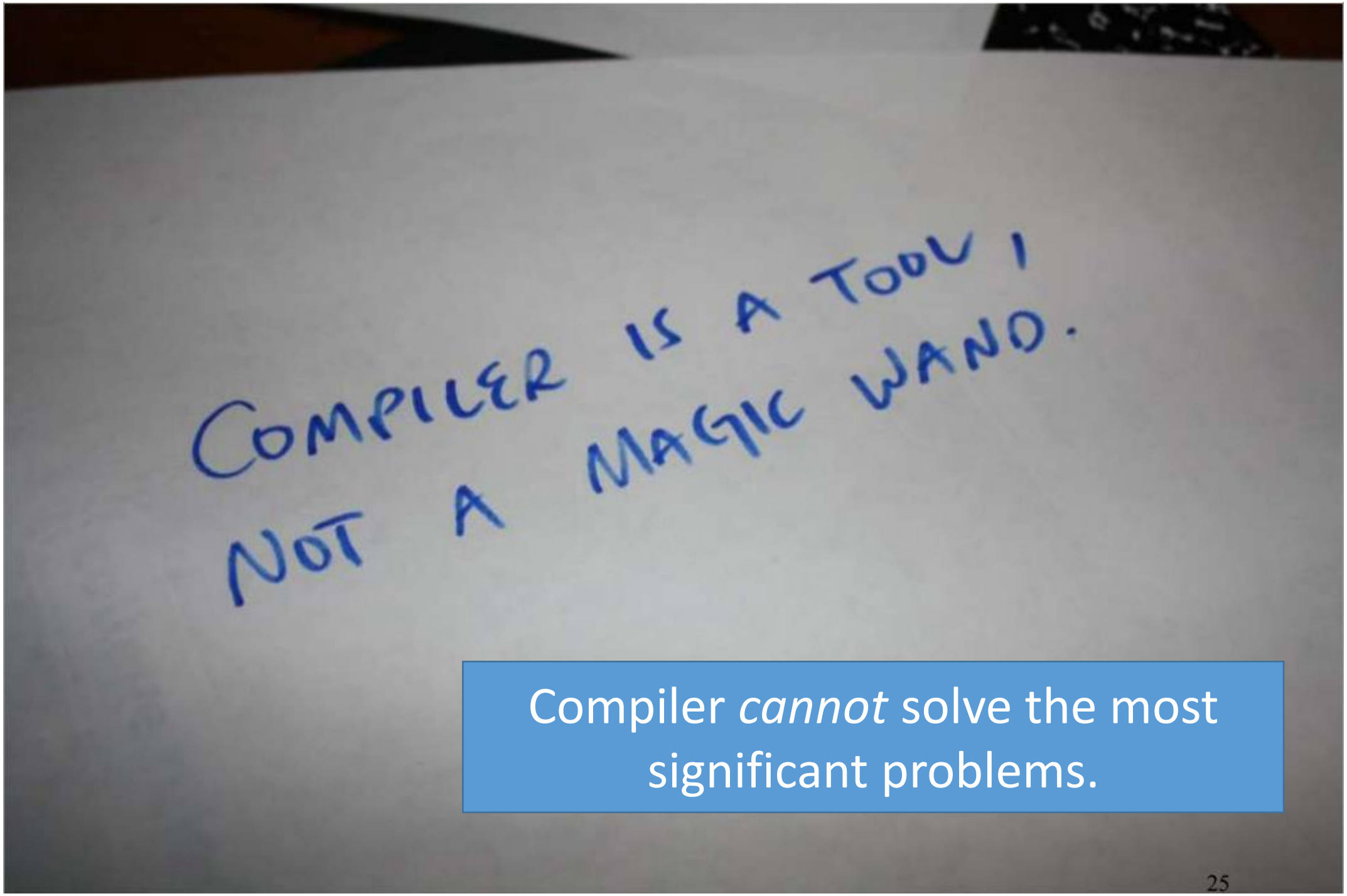
Time spent waiting for L2 vs. actual work

~10:1

This is the compiler's space.

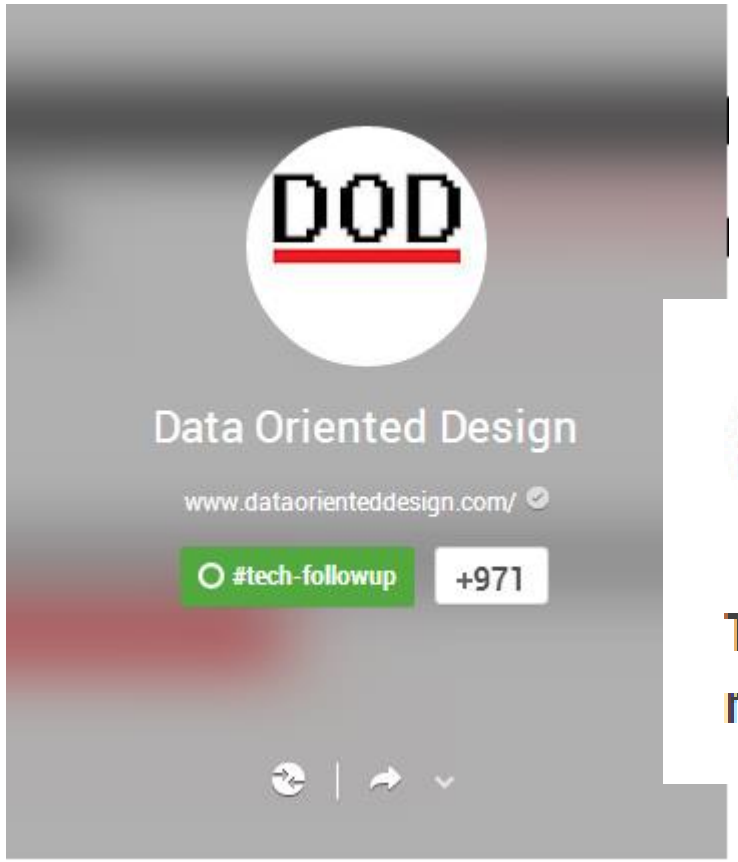
Compiler can solve about 1-10% of the problem space.

i.e. the vast majority of problems are things the compiler can't reason about



Compiler *cannot* solve the most significant problems.

See also:



DOD **Data Oriented Design**

Shared publicly - Jan 4, 2012

Things to consider : #08 Bad code is salvagable - bad data can be a much worse spaghetti to untangle

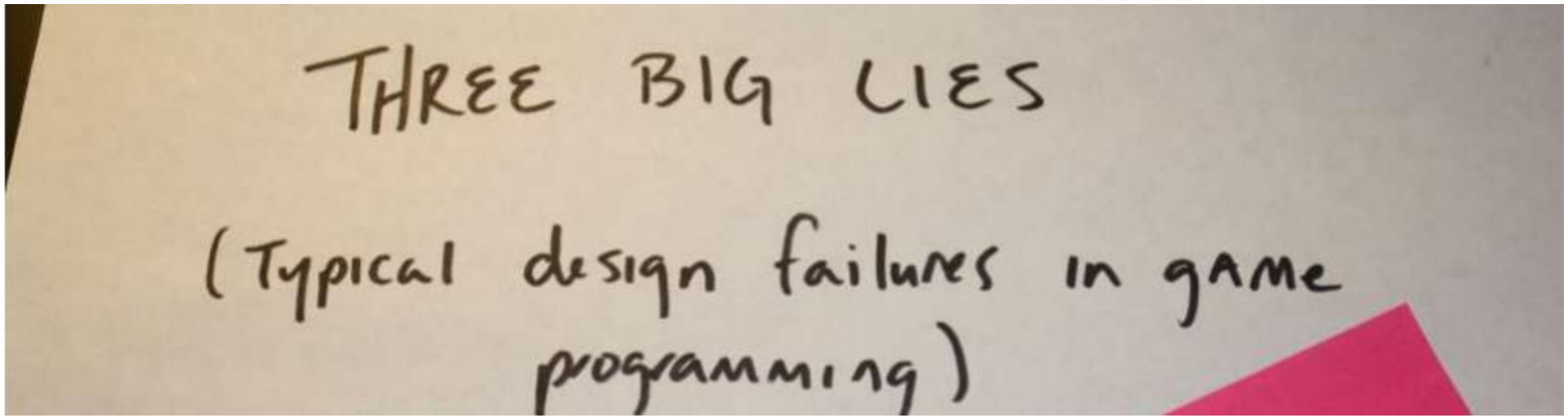
https://plus.google.com/u/0/+Dataorienteddesign/posts

Today's subject:
The 90% of problem space we need to solve that the compiler cannot.

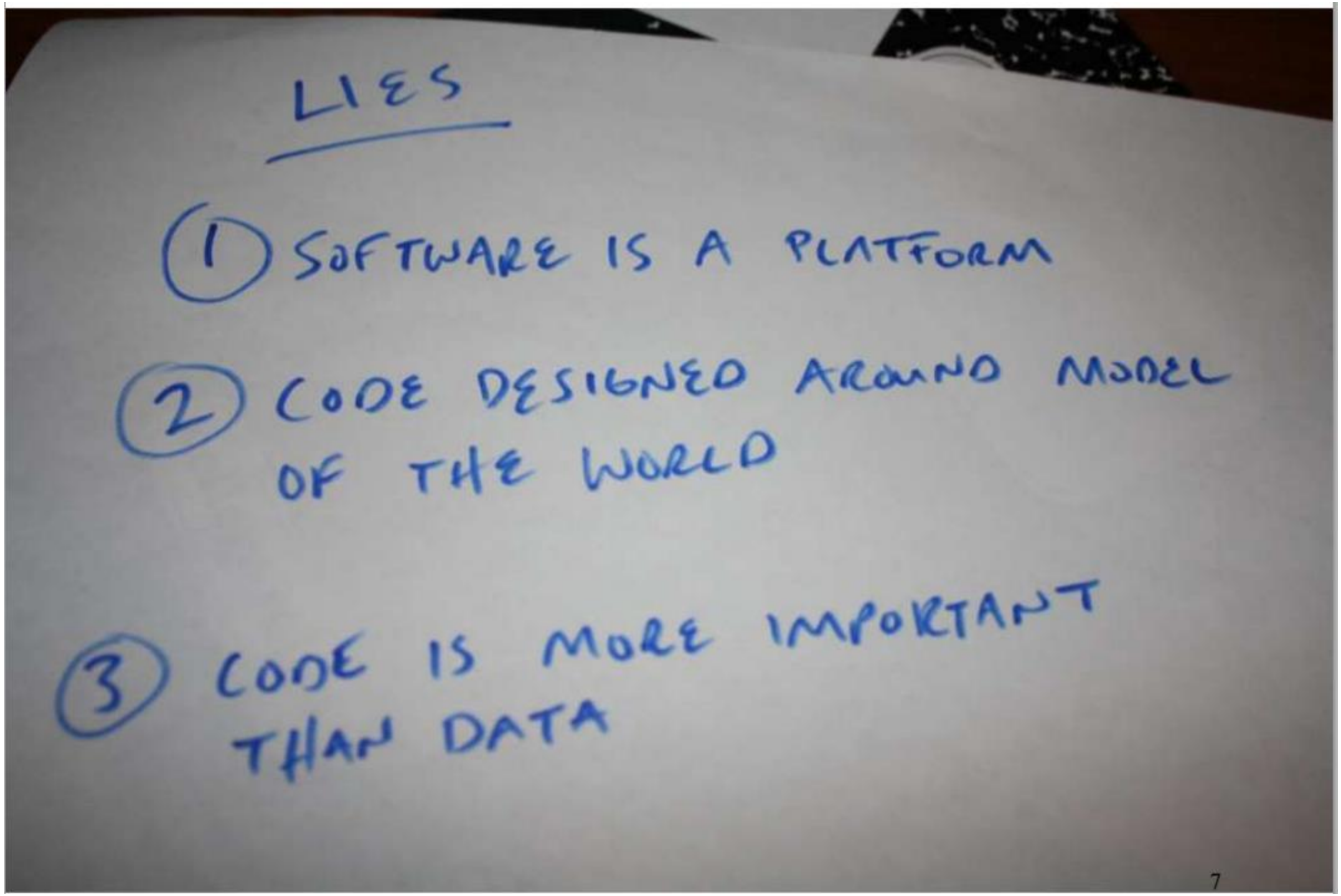
(And how we can help it with the 10% that it can.)

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

What’s the most common cause of waste?



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

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;
  }
};
```

So how do we solve for it?

L2 cache misses/frame

(Don’t waste them!)

```
_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
```

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.

```
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;
};

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;
  }
}
```

12 bytes x count(5) = 72

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

12 bytes x count(5) = 72

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

4 bytes x count(5) = 20

```
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* =
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

In addition…

1. Code is maintainable
2. Code is debugable
3. Can REASON about cost of change

```
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;
  }
}
```

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

In addition…

1. Code is maintainable
2. Code is debugable
3. Can REASON about cost of change

Ignoring inconvenient facts is not engineering;
It's dogma.

```
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;
  }
}
```

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

Let’s review some code…

default | ogre / OgreMain / src / OgreNode.cpp

569ec69 2013-10-15 | Full commit

```
1  /*
2  -----------------------------------------------------------------------------
3  This source file is part of OGRE
4      (Object-oriented Graphics Rendering Engine)
5  For the latest info, see http://www.ogre3d.org/
6
7  Copyright (c) 2000-2013 Torus Knot Software Ltd
```

Mike Acton reviewed the 1.9 version. Perhaps it would've been more interesting to see a review of the 2.0 file which has been refactored to better fit Data Oriented Design principles (and I'm sure there are things I wrote to criticize). Many of the things he criticizes of 1.9 have been fixed. Nevertheless there are things we can learn. Note that if he weren't right, then it would be hard to explain why there was a 5x performance increase between 1.9 and 2.0.

http://yosoygames.com.ar/wp/2013/11/on-mike-actons-review-of-ogrenode-cpp/

```
namespace Ogre {

    NameGenerator Node::msNameGenerator("Unnamed_");
        Node::QueuedUpdates Node::msQueuedUpdates;
    //-----------------------------------------------------------------------
    Node::Node()
                :mParent(0),
                mNeedParentUpdate(false),
                mNeedChildUpdate(false),
                mParentNotified(false),
        mQueuedForUpdate(false),
                mOrientation(Quaternion::IDENTITY),
                mPosition(Vector3::ZERO),
                mScale(Vector3::UNIT_SCALE),
        mInheritOrientation(true),
                mInheritScale(true),
                mDerivedOrientation(Quaternion::IDENTITY),
                mDerivedPosition(Vector3::ZERO),
                mDerivedScale(Vector3::UNIT_SCALE),
                mInitialPosition(Vector3::ZERO),
                mInitialOrientation(Quaternion::IDENTITY),
                mInitialScale(Vector3::UNIT_SCALE),
                mCachedTransformOutOfDate(true),
                mListener(0),
                mDebug(0)
    {
        // Generate a name
        mName = msNameGenerator.generate();

        needUpdate();

    }
```

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

(1) Can’t re-arrange memory (much)

Limited by ABI

Can’t limit unused reads

Extra padding

Can a C++ compiler re-order elements in a struct

http://stackoverflow.com/questions/916600/can-a-c-compiler-re-order-elements-in-a-struct

In theory…

Here's the relevant part of the standard:

Section 9.2.12:

> Nonstatic data members of a (non-union) class declared without an intervening access-specifier are allocated so that later members have higher addresses within a class object. The order of allocation of nonstatic data members separated by an access-specifier is unspecified (11.1)"

share | improve this answer

edited May 27 '09 at 16:54

answered May 27 '09 at 16:20

jalf

136k ●22 ●197 ●412

In practice…

```
class Foo
{
public:
  uint8_t  a0;
  uint16_t b0;
  uint8_t  c0;
  bool     d0;

  uint8_t  a1;
  uint16_t b1;
  uint8_t  c1;
  bool     d1;

  uint8_t  a2;
  uint16_t b2;
  uint8_t  c2;
  bool     d2;
};

class Bar
{
  public: uint8_t  a0;
  public: uint16_t b0;
  public: uint8_t  c0;
  public: bool     d0;

  public: uint8_t  a1;
  public: uint16_t b1;
  public: uint8_t  c1;
  public: bool     d1;

  public: uint8_t  a2;
  public: uint16_t b2;
  public: uint8_t  c2;
  public: bool     d2;
};
```

In practice…

```
class Foo
{
public:
  uint8_t  a0;
  uint16_t b0;
  uint8_t  c0;
  bool     d0;

  uint8_t  a1;
  uint16_t b1;
  uint8_t  c1;
  bool     d1;

  uint8_t  a2;
  uint16_t b2;
  uint8_t  c2;
  bool     d2;
};

class Bar
{
  public: uint8_t  a0;
  public: uint16_t b0;
  public: uint8_t  c0;
  public: bool     d0;

  public: uint8_t  a1;
  public: uint16_t b1;
  public: uint8_t  c1;
  public: bool     d1;

  public: uint8_t  a2;
  public: uint16_t b2;
  public: uint8_t  c2;
  public: bool     d2;
};
```

```
Foo
{
  a0: 0
  b0: 2
  c0: 4
  d0: 5
  a1: 6
  b1: 8
  c1: 10
  d1: 11
  a2: 12
  b2: 14
  c2: 16
  d2: 17
}
Bar
{
  a0: 0
  b0: 2
  c0: 4
  d0: 5
  a1: 6
  b1: 8
  c1: 10
  d1: 11
  a2: 12
  b2: 14
  c2: 16
  d2: 17
}
```

```
(int)offsetof( Foo, a0 ),
(int)offsetof( Foo, b0 ),
(int)offsetof( Foo, c0 ),
(int)offsetof( Foo, d0 ),
(int)offsetof( Foo, a1 ),
(int)offsetof( Foo, b1 ),
(int)offsetof( Foo, c1 ),
(int)offsetof( Foo, d1 ),
(int)offsetof( Foo, a2 ),
(int)offsetof( Foo, b2 ),
(int)offsetof( Foo, c2 ),
(int)offsetof( Foo, d2 ) );
```

(2) Bools and last-minute decision making

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

bools in structs…

(3) Extremely low information density

```
class __attribute__((__packed__)) Foo
{
public:
  uint8_t  a0;
  uint16_t b0;
  uint8_t  c0;
  bool     d0;

  uint8_t  a1;
  uint16_t b1;
  uint8_t  c1;
  bool     d1;

  uint8_t  a2;
  uint16_t b2;
  uint8_t  c2;
  bool     d2;
};
```

```
Foo
{
  a0: 0
  b0: 1
  c0: 3
  d0: 4
  a1: 5
  b1: 6
  c1: 8
  d1: 9
  a2: 10
  b2: 11
  c2: 13
  d2: 14
}
```

bools in structs…

(3) Extremely low information density

How big is your cache line?

```
class __attribute__((__packed__)) Foo
{
public:
  uint8_t  a0;
  uint16_t b0;
  uint8_t  c0;
  bool     d0;

  uint8_t  a1;
  uint16_t b1;
  uint8_t  c1;
  bool     d1;

  uint8_t  a2;
  uint16_t b2;
  uint8_t  c2;
  bool     d2;
};
```

```
Foo
{
  a0: 0
  b0: 1
  c0: 3
  d0: 4
  a1: 5
  b1: 6
  c1: 8
  d1: 9
  a2: 10
  b2: 11
  c2: 13
  d2: 14
}
```

```
class Foo
{
public:
  bool    m_NeedParentUpdate;
  bool    m_NeedChildUpdate;
  bool    m_ParentNotNotified;
  Mat4    m_ObjectWorld;
  bool    m_InheritScale;
  bool    m_InheritOrientation;
};
```

```
Foo
{
  m_NeedParentUpdate:    0
  m_NeedChildUpdate:     1
  m_ParentNotNotified:   2
  m_ObjectWorld:         4
  m_InheritScale:        68
  m_InheritOrientation:  69
}
```

bools in structs…

```
class __attribute__((__packed__)) Foo
{
public:
  uint8_t  a0;
  uint16_t b0;
  uint8_t  c0;
  bool     d0;

  uint8_t  a1;
  uint16_t b1;
  uint8_t  c1;
  bool     d1;

  uint8_t  a2;
  uint16_t b2;
  uint8_t  c2;
  bool     d2;
};
```

```
Foo
{
  a0: 0
  b0: 1
  c0: 3
  d0: 4
  a1: 5
  b1: 6
  c1: 8
  d1: 9
  a2: 10
  b2: 11
  c2: 13
  d2: 14
}
```

(3) Extremely low information density

How big is your cache line?

What's the most commonly accessed data?

```
class Foo
{
public:
  bool    m_NeedParentUpdate;
  bool    m_NeedChildUpdate;
  bool    m_ParentNotNotified;
  Mat4    m_ObjectWorld;
  bool    m_InheritScale;
  bool    m_InheritOrientation;
};
```

```
Foo
{
  m_NeedParentUpdate:    0
  m_NeedChildUpdate:     1
  m_ParentNotNotified:   2
  m_ObjectWorld:         4
  m_InheritScale:        68
  m_InheritOrientation:  69
}
```

64b?

How is it used? What does it generate?

(2) Bools and last-minute decision making

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

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 : }
```

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…

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);
}
```

```
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);
}
```

(4) 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);
}
```

(4) 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
```

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

```
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);
}
```

(4) 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.

Applies to any member fields especially.
(Not particular to bools)

The story so far… How can you help the compiler?

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //---------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

Arrange memory by probability of access.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

Hoist decision making to first-opportunity.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

Maximize memory read value.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

Maximize memory read value.

How can we measure this?

(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn = (choose < m_SpawnChance);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn  = (choose < m_SpawnChance);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

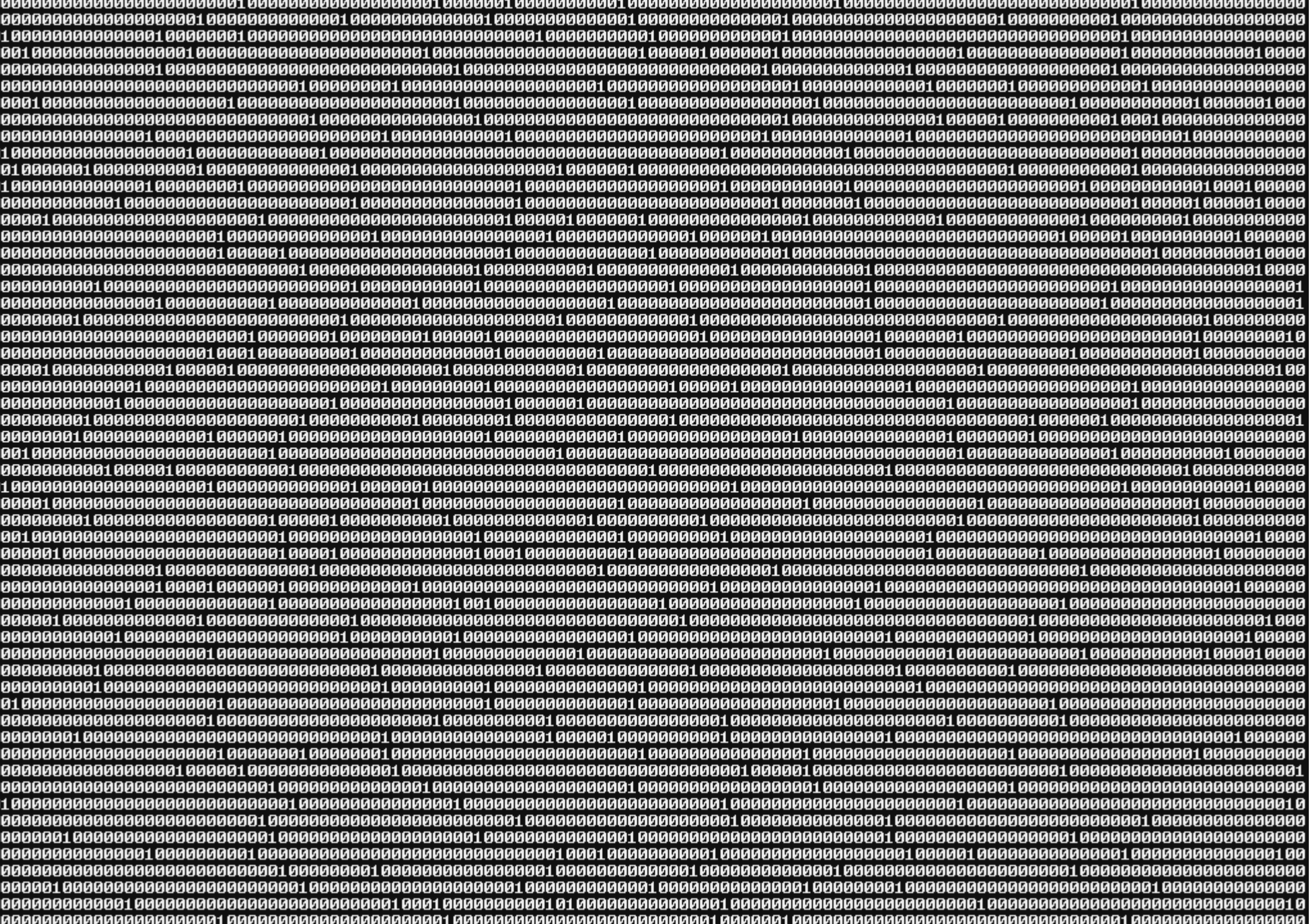
What is the information density for is_spawn over time?

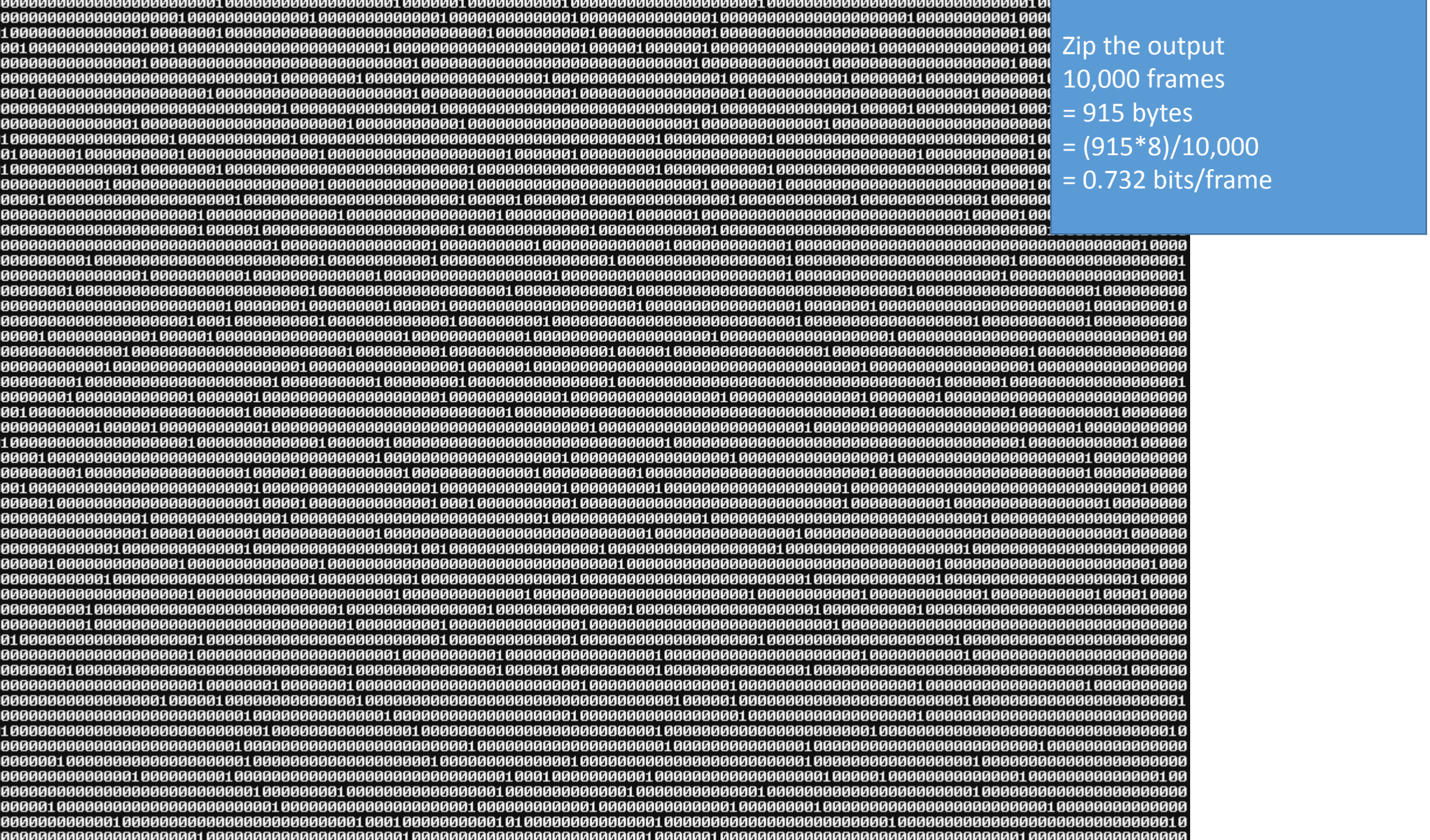
(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn  = (choose < m_SpawnChance);
  printf("%d",is_spawn?1:0);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

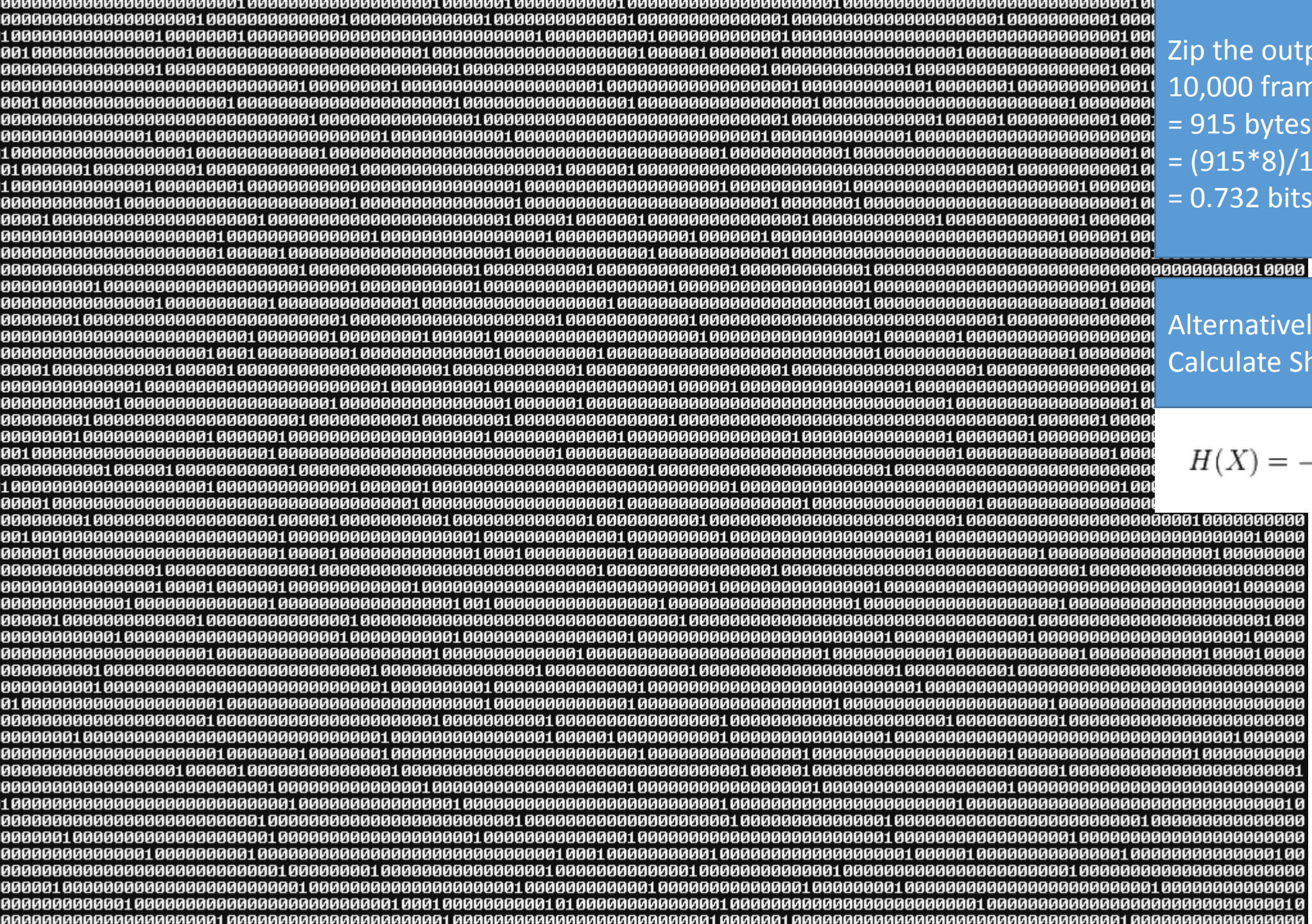
What is the information density for is_spawn over time?

The easy way.





Zip the output
10,000 frames
= 915 bytes
= (915*8)/10,000
= 0.732 bits/frame



Zip the output
10,000 frames
= 915 bytes
= (915*8)/10,000
= 0.732 bits/frame

Alternatively,
Calculate Shannon Entropy:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn = (choose < m_SpawnChance);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

What does that tell us?

(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn = (choose < m_SpawnChance);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

What does that tell us?

Figure (~2 L2 misses each frame) x 10,000
If each cache line = 64b,
128b x 10,000 = 1,280,000 bytes

(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn = (choose < m_SpawnChance);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

What does that tell us?

Figure (~2 L2 misses each frame) x 10,000
If each cache line = 64b,
128b x 10,000 = 1,280,000 bytes

If avg information content = 0.732bits/frame
X 10,000 = 7320 bits
/ 8 = 915 bytes

(3) Extremely low information density

```
void
Foo::Update()
{
  float choose    = RandFloat();
  bool  is_spawn  = (choose < m_SpawnChance);
  if (is_spawn)
  {
    Spawn();
    m_SpawnChance = 0.0f;
  }
  else
  {
    m_SpawnChance += kFooSpawnChanceIncrease;
  }
}
```

What does that tell us?

Figure (~2 L2 misses each frame) x 10,000
If each cache line = 64b,
128b x 10,000 = 1,280,000 bytes

If avg information content = 0.732bits/frame
X 10,000 = 7320 bits
/ 8 = 915 bytes

Percentage waste (Noise::Signal) =
(1,280,000-915)/1,280,000

0.99928515625

What’re the alternatives?

(1) Per-frame…

(1) Per-frame… (decision table)

1 of 512 (8*64) bits used…

(1) Per-frame… (decision table)

1 of 512 (8*64) bits used…

(a) Make same decision x 512

(1) Per-frame… (decision table)

1 of 512 (8*64) bits used…

(a) Make same decision x 512

(b) Combine with other reads / xforms

(1) Per-frame… (decision table)

1 of 512 (8*64) bits used…

(a) Make same decision x 512

(b) Combine with other reads / xforms

Generally simplest.

- But things cannot exist in abstract bubble.
- Will require context.

(2) Over-frames…

(2) Over-frames…

i.e. Only read when needed

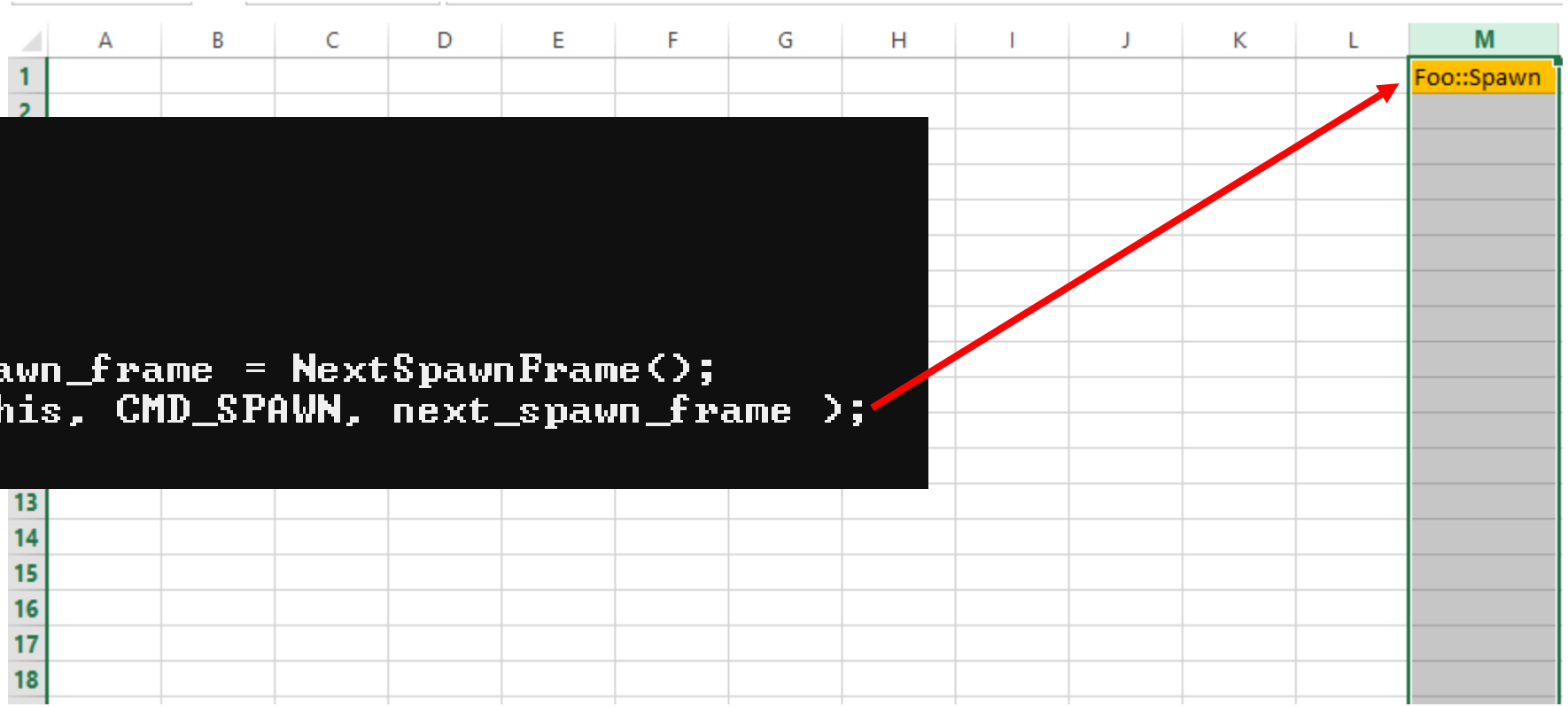
(2) Over-frames…

i.e. Only read when needed

e.g.

```
void
Foo::Spawn()
{
  ...

  uint32_t next_spawn_frame = NextSpawnFrame();
  InsertCommand( this, CMD_SPAWN, next_spawn_frame );
}
```



```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

Maximize memory read value.

How can we measure this?

(Try it.)

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

Maximize memory read value.

How can we measure this?

(Try it.)

All these “code smells” can be viewed as symptoms of information density problems…

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

(4) Ghost reads and writes

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

(4) Ghost reads and writes

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

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

The story so far… How can you help the compiler?

(1) Can’t re-arrange memory (much)

(2) Bools and last-minute decision making

(3) Extremely low information density

(4) Ghost reads and writes

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

Easy to confuse compiler, even within the 10% space

Are we done with the constructor?

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

Complex constructors tend to imply that…

- Reads are unmanaged (one at a time…)

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

Complex constructors tend to imply that…

- Reads are unmanaged (one at a time…)
- Unnecessary reads/writes in destructors

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

Complex constructors tend to imply that…

- Reads are unmanaged (one at a time…)
- Unnecessary reads/writes in destructors
- Unmanaged icache (i.e. virtuals)

=> unmanaged reads/writes

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

Complex constructors tend to imply that…

- Reads are unmanaged (one at a time…)
- Unnecessary reads/writes in destructors
- Unmanaged icache (i.e. virtuals)
  => unmanaged reads/writes
- Unnecessarily complex state machines (back to bools)
  - E.g. 2^7 states

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

Complex constructors tend to imply that…

- Reads are unmanaged (one at a time…)
- Unnecessary reads/writes in destructors
- Unmanaged icache (i.e. virtuals)
  => unmanaged reads/writes
- Unnecessarily complex state machines (back to bools)
  - E.g. 2^7 states

Rule of thumb:
Store each state type separately
Store same states together
(No state value needed)

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

Imply more (wasted) reads because pretending you don’t know what it could be.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

Imply more (wasted) reads because pretending you don’t know what it could be.

e.g. Strings, generally. Filenames, in particular.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

Imply more (wasted) reads because pretending you don’t know what it could be.

e.g. Strings, generally. Filenames, in particular.

Rule of thumb:
The best code is code that doesn’t need to exist.
Do it offline. Do it once.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

(7) Over-solving (computing too much)

Compiler doesn't have enough context to know how to simplify your problems for you.

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

(7) Over-solving (computing too much)

Compiler doesn't have enough context to know how to simplify your problems for you.

But you can make simple tools that do…

- E.g. Premultiply matrices

```
45  namespace Ogre {
46
47      NameGenerator Node::msNameGenerator("Unnamed_");
48          Node::QueuedUpdates Node::msQueuedUpdates;
49      //-----------------------------------------------------------------------
50      Node::Node()
51                  :mParent(0),
52                  mNeedParentUpdate(false),
53                  mNeedChildUpdate(false),
54                  mParentNotified(false),
55          mQueuedForUpdate(false),
56                  mOrientation(Quaternion::IDENTITY),
57                  mPosition(Vector3::ZERO),
58                  mScale(Vector3::UNIT_SCALE),
59          mInheritOrientation(true),
60                  mInheritScale(true),
61                  mDerivedOrientation(Quaternion::IDENTITY),
62                  mDerivedPosition(Vector3::ZERO),
63                  mDerivedScale(Vector3::UNIT_SCALE),
64                  mInitialPosition(Vector3::ZERO),
65                  mInitialOrientation(Quaternion::IDENTITY),
66                  mInitialScale(Vector3::UNIT_SCALE),
67                  mCachedTransformOutOfDate(true),
68                  mListener(0),
69                  mDebug(0)
70      {
71          // Generate a name
72          mName = msNameGenerator.generate();
73
74          needUpdate();
75
76      }
```

Are we done with the constructor?

(5) Over-generalization

(6) Undefined or under-defined constraints

(7) Over-solving (computing too much)

Compiler doesn't have enough context to know how to simplify your problems for you.

But you can make simple tools that do…

- E.g. Premultiply matrices

Work with the (actual) data you have.

- E.g. Sparse or affine matrices

Finish your derivations, please

October 21, 2010

http://fgiesen.wordpress.com/2010/10/21/finish-your-derivations-please/

Fabian Giesen
@rygorous
hund.

Is the compiler going to transform this…

```
float alpha = max( acos( dot( v, n ) ), acos( dot( l, n ) ) );
float beta  = min( acos( dot( v, n ) ), acos( dot( l, n ) ) );
C = sin(alpha) * tan(beta);
```

Into this… for you?

```
float vdotn = dot(v, n);
float ldotn = dot(l, n);
C = sqrt((1.0 - vdotn*vdotn) * (1.0 - ldotn*ldotn))
  / max(vdotn, ldotn);
```

While we're on the subject…
DESIGN PATTERNS:

> "Design patterns are spoonfeed material for brainless programmers incapable of independent thought, who will be resolved to producing code as mediocre as the design patterns they use to create it.

http://realtimecollisiondetection.net/blog/?p=81

http://realtimecollisiondetection.net/blog/?p=44

Okay… Now a quick pass through some other functions.

```
439     //-----------------------------------------------------------------------
440     void Node::translate(const Vector3& d, TransformSpace relativeTo)
441     {
442         switch(relativeTo)
443         {
444         case TS_LOCAL:
445             // position is relative to parent so transform downwards
446             mPosition += mOrientation * d;
447                 break;
448         case TS_WORLD:
449             // position is relative to parent so transform upwards
450             if (mParent)
451             {
452                 mPosition += (mParent->_getDerivedOrientation().Inverse() * d)
453                     / mParent->_getDerivedScale();
454             }
455             else
456             {
457                 mPosition += d;
458             }
459                 break;
460         case TS_PARENT:
461             mPosition += d;
462             break;
463         }
464         needUpdate();
465 
466     }
```

(2) Bools and last-minute decision making

```
439     //-----------------------------------------------------------------------
440     void Node::translate(const Vector3& d, TransformSpace relativeTo)
441     {
442         switch(relativeTo)
443         {
444         case TS_LOCAL:
445             // position is relative to parent so transform downwards
446             mPosition += mOrientation * d;
447                 break;
448         case TS_WORLD:
449             // position is relative to parent so transform upwards
450             if (mParent)
451             {
452                 mPosition += (mParent->_getDerivedOrientation().Inverse() * d)
453                     / mParent->_getDerivedScale();
454             }
455             else
456             {
457                 mPosition += d;
458             }
459                 break;
460         case TS_PARENT:
461             mPosition += d;
462             break;
463         }
464         needUpdate();
465
466     }
```

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    node->m_Position += node->m_Orientation * d;
  }
}

void
NodesTranslateWorld( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}

void
NodesTranslateParent( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    node->m_Position += d;
  }
}
```

Step 1: organize

Separate states so you can reason about them

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    node->m_Position += node->m_Orientation * d;
  }
}

void
NodesTranslateWorld( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}

void
NodesTranslateParent( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    node->m_Position += d;
  }
}
```

Step 1: organize
Separate states so you can reason about them

Step 2: triage
What are the relative values of each case
i.e. p(call) * count

```
void
NodesTranslateLocalEach( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    = *t[i];
    node->m_Position += node->m_Orientation * d;
  }
}

void
NodesTranslateWorldEach( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    = *t[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}

void
NodesTranslateParentEach( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    = *t[i];
    node->m_Position += d;
  }
}
```

Step 1: organize
Separate states so you can reason about them

Step 2: triage
What are the relative values of each case
i.e. p(call) * count

e.g. in-game vs. in-editor

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    node->m_Position += node->m_Orientation * d;
  }
}

void
NodesTranslateWorld( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}

void
NodesTranslateParent( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    node->m_Position += d;
  }
}
```

Step 1: organize
Separate states so you can reason about them

Step 2: triage
What are the relative values of each case
i.e. p(call) * count

Step 3: reduce waste

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += m_Orientation * d;
  }
}
```

(back of the envelope read cost)

~200 cycles x 2 x count

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += m_Orientation * d;
  }
}
```

(back of the envelope read cost)

~200 cycles x 2 x count

```
struct NodeTranslate
{
  Vec3 m_Position;
  Quat m_Orientation;
};

void
NodesTranslateLocal( NodeTranslate* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += node->m_Orientation * d;
  }
}
```

~2.28 count per 200 cycles
= ~88

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += m_Orientation * d;
  }
}
```

(back of the envelope read cost)

~200 cycles x 2 x count

```
struct NodeTranslate
{
  Vec3 m_Position;
  Quat m_Orientation;
};

void
NodesTranslateLocal( NodeTranslate* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += node->m_Orientation * d;
  }
}
```

~2.28 count per 200 cycles
= ~88

```
t = 2 * cross(q.xyz, v)
v' = v + q.w * t + cross(q.xyz, t)
```

```
void
NodesTranslateLocal( Node* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += m_Orientation * d;
  }
}
```

(back of the envelope read cost)

~200 cycles x 2 x count

```
struct NodeTranslate
{
  Vec3 m_Position;
  Quat m_Orientation;
};

void
NodesTranslateLocal( NodeTranslate* nodes, int count, const Vector3& d )
{
  for (int i=0;i<count;i++)
  {
    node->m_Position += node->m_Orientation * d;
  }
}
```

~2.28 count per 200 cycles
= ~88

(close enough to dig in and measure)

```
t = 2 * cross(q.xyz, v)
v' = v + q.w * t + cross(q.xyz, t)
```

Apply the same steps recursively…

```
void
NodesTranslateWorldEach( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    = t[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}
```

Apply the same steps recursively…

```
void
NodesTranslateWorldEach( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    = t[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}
```

Step 1: organize

Separate states so you can reason about them

Root or not; Calling function with context can distinguish

Apply the same steps recursively…

```
void
NodesTranslateWorldEach( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    = t[i];
    if ( node->m_Parent )
    {
      node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                          / node->m_Parent->_getDerivedScale();
    }
    else
    {
      node->m_Position += d;
    }
  }
}
```

Step 1: organize

Separate states so you can reason about them

Root or not; Calling function with context can distinguish

Apply the same steps recursively…

```
void
NodesTranslateWorldEachRoot( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    =  t[i];
    node->m_Position += d;
  }
}

void
NodesTranslateWorldEachWithParent( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    =  t[i];
    node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                      / node->m_Parent->_getDerivedScale();
  }
}
```

Step 1: organize

Separate states so you can reason about them

Apply the same steps recursively…

```
void
NodesTranslateWorldEachRoot( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    =  t[i];
    node->m_Position += d;
  }
}

void
NodesTranslateWorldEachWithParent( Node* nodes, int count, const Vector3* t )
{
  for (int i=0;i<count;i++)
  {
    Node* node = &nodes[i];
    Vec3& d    =  t[i];
    node->m_Position += ( node->m_Parent->_getDerivedOrientation().Inverse() * d );
                      / node->m_Parent->_getDerivedScale();
  }
}
```

Step 1: organize

Separate states so you can reason about them

Can’t reason well about the cost from…

```
// NodeParent
//    Quat       DerivedOrientationInverse
//    float      DerivedScale
//    uint32_t   ChildCount
//    Vector3    ChildPosition[]

void
NodesTranslateWorldEachWithParent( char* nodes, int parentCount, const Vector3* t )
{
  int k=0;
  for (int i=0;i<parentCount;i++)
  {
    Quat*    derivedOrientationInverse = (Quat*)nodes;
    nodes += sizeof(Quat);

    float    derivedScale              = *(float*)nodes;
    nodes += sizeof(float);

    uint32_t childCount                = *(uint32_t*)nodes;
    nodes += sizeof(uint32_t)

    for (int j=0;j<childCount;j++,k++)
    {
      Vector3& d             = t[k];
      Vector3& childPosition = *(Vector3*)nodes;
      nodes += sizeof(Vector3);

      childPosition += (derivedOrientationInverse * d) / derivedScale;
    }
  }
}
```

Step 1: organize

Separate states so you can reason about them

```
// NodeParent
//    Quat       DerivedOrientationInverse
//    float      DerivedScale
//    uint32_t   ChildCount
//    Vector3    ChildPosition[]

void
NodesTranslateWorldEachWithParent( char* nodes, int parentCount, const Vector3* t )
{
  int k=0;
  for (int i=0;i<parentCount;i++)
  {
    Quat*    derivedOrientationInverse = (Quat*)nodes;
    nodes += sizeof(Quat);

    float    derivedScale              = *(float*)nodes;
    nodes += sizeof(float);

    uint32_t childCount                = *(uint32_t*)nodes;
    nodes += sizeof(uint32_t)

    for (int j=0;j<childCount;j++,k++)
    {
      Vector3& d             = t[k];
      Vector3& childPosition = *(Vector3*)nodes;
      nodes += sizeof(Vector3);

      childPosition += (derivedOrientationInverse * d) / derivedScale;
    }
  }
}
```

Step 1: organize

Separate states so you can reason about them

Step 2: triage

What are the relative values of each case

i.e. p(call) * count

Step 3: reduce waste

And here…

```
509     //-----------------------------------------------------------------------
510     void Node::rotate(const Quaternion& q, TransformSpace relativeTo)
511     {
512             // Normalise quaternion to avoid drift
513             Quaternion qnorm = q;
514             qnorm.normalise();
515
516         switch(relativeTo)
517         {
518         case TS_PARENT:
519             // Rotations are normally relative to local axes, transform up
520             mOrientation = qnorm * mOrientation;
521             break;
522         case TS_WORLD:
523             // Rotations are normally relative to local axes, transform up
524             mOrientation = mOrientation * _getDerivedOrientation().Inverse()
525                 * qnorm * _getDerivedOrientation();
526             break;
527         case TS_LOCAL:
528             // Note the order of the mult, i.e. q comes after
529             mOrientation = mOrientation * qnorm;
530             break;
531         }
532         needUpdate();
533     }
```

Before we close, let’s revisit…

```
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;
  }
}
```

Good News:
Most problems are
easy to see.

Good News:
Side-effect of solving the 90% well, compiler can solve the 10% better.

Good News:
Organized data makes maintenance, debugging and concurrency much easier

Bad News:
Good programming is hard.
Bad programming is easy.

PS: Let’s get more women in tech