The DFG JIT, Inside & Out

JavaScriptCore's Optimizing Compiler

JSConf.EU 2012

Andy Wingo

0.1 wingo@igalia.com

Compiler hacker at Igalia

Contract work on language implementations

V8, JavaScriptCore

Schemer

0.2 Hubris

"Now that JavaScriptCore is as fast as V8 on its own benchmark, it's well past time to take a look inside JSC's optimizing compiler, the DFG JIT."

0.3 DFG

Optimizing compiler for JSC

LLInt -> Baseline JIT -> **DFG JIT**

Makes hot code run fast

But how good is it?

0.4 An empirical approach

Getting good code

What: V8 benchmarks

When: Hacked V8 benchmarks

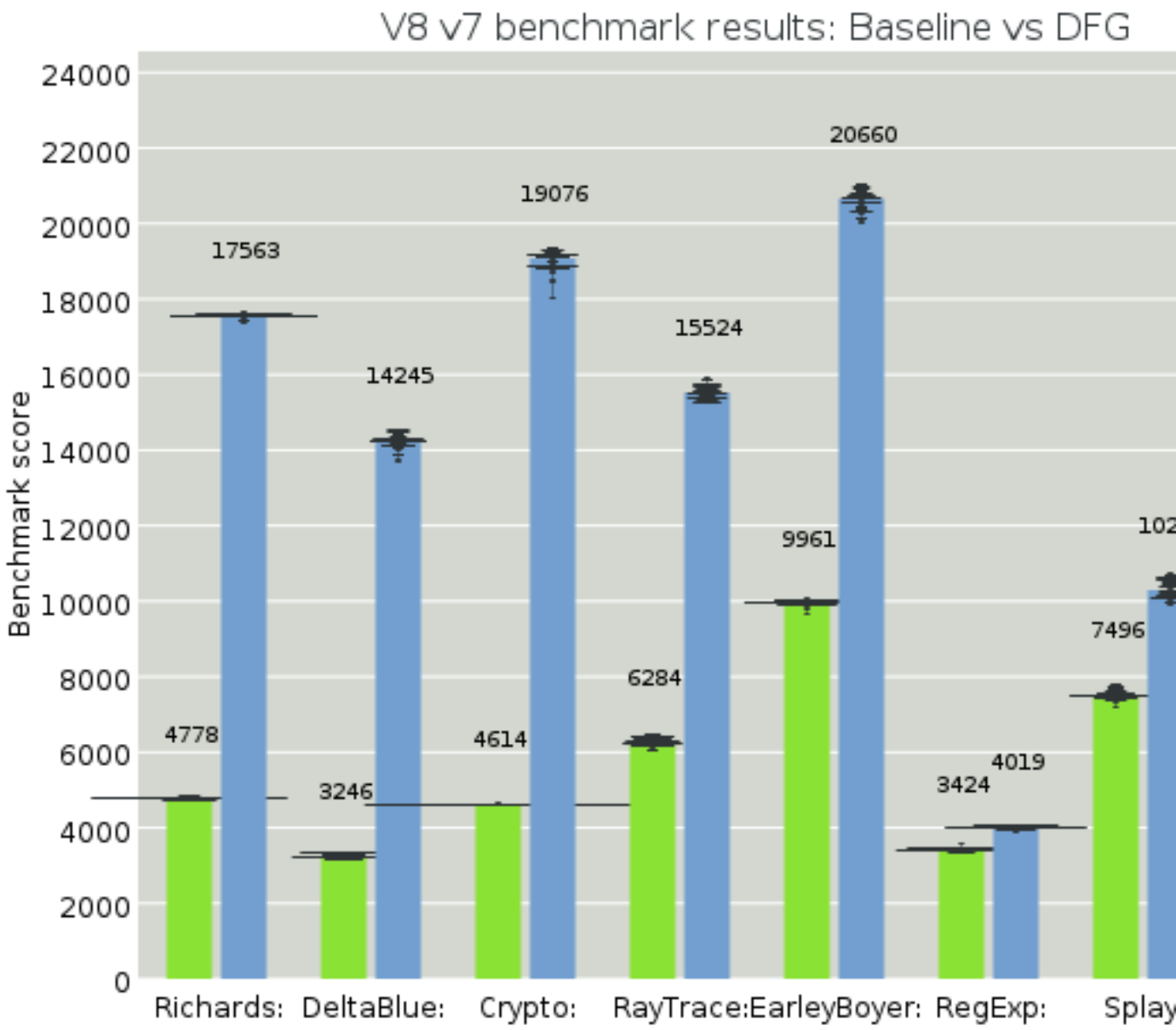
How: Code dive

0.5 The V8 benchmarks

The best performance possible from an optimizing compiler

- full second of warmup
- full second of runtime
- long run amortizes GC pauses

0.6 Baseline JIT vs DFG



0.7 Abusing the V8 benchmarks

When does the DFG kick in?

What does it do?

Idea: V8 benchmarks with variable warmup

- after 0 ms of warmup
- after 5 ms of warmup

- after n ms of warmup

Small fixed runtime (5 ms)

0.8 Caveats

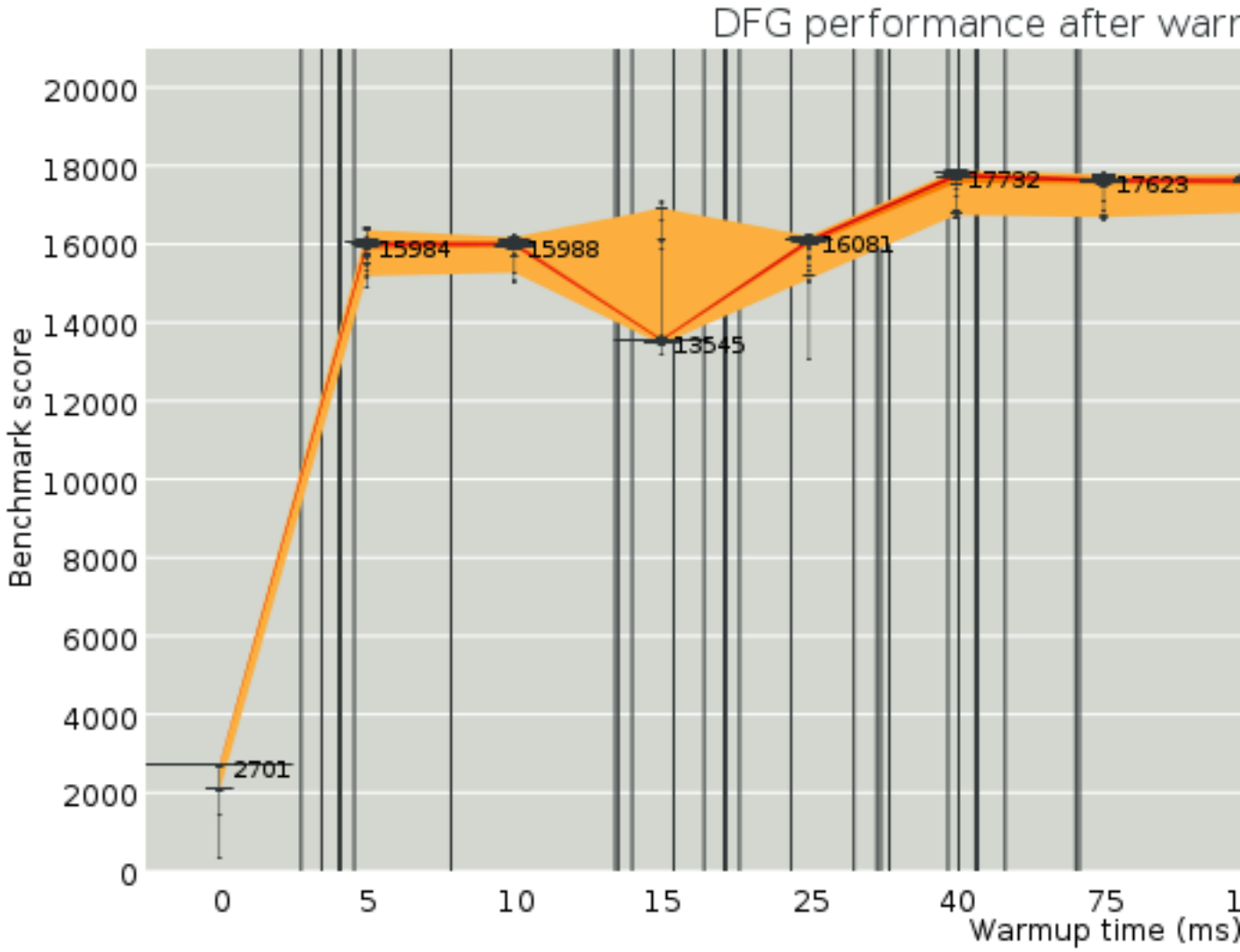
Very sensitive

- GC
- optimization pauses
- timer precision

... but then, so is real code

Keep close eye on distribution of measurements

0.9 Richards



Speedup: 3.7X

Bit ops, properties, prototypes

0.10

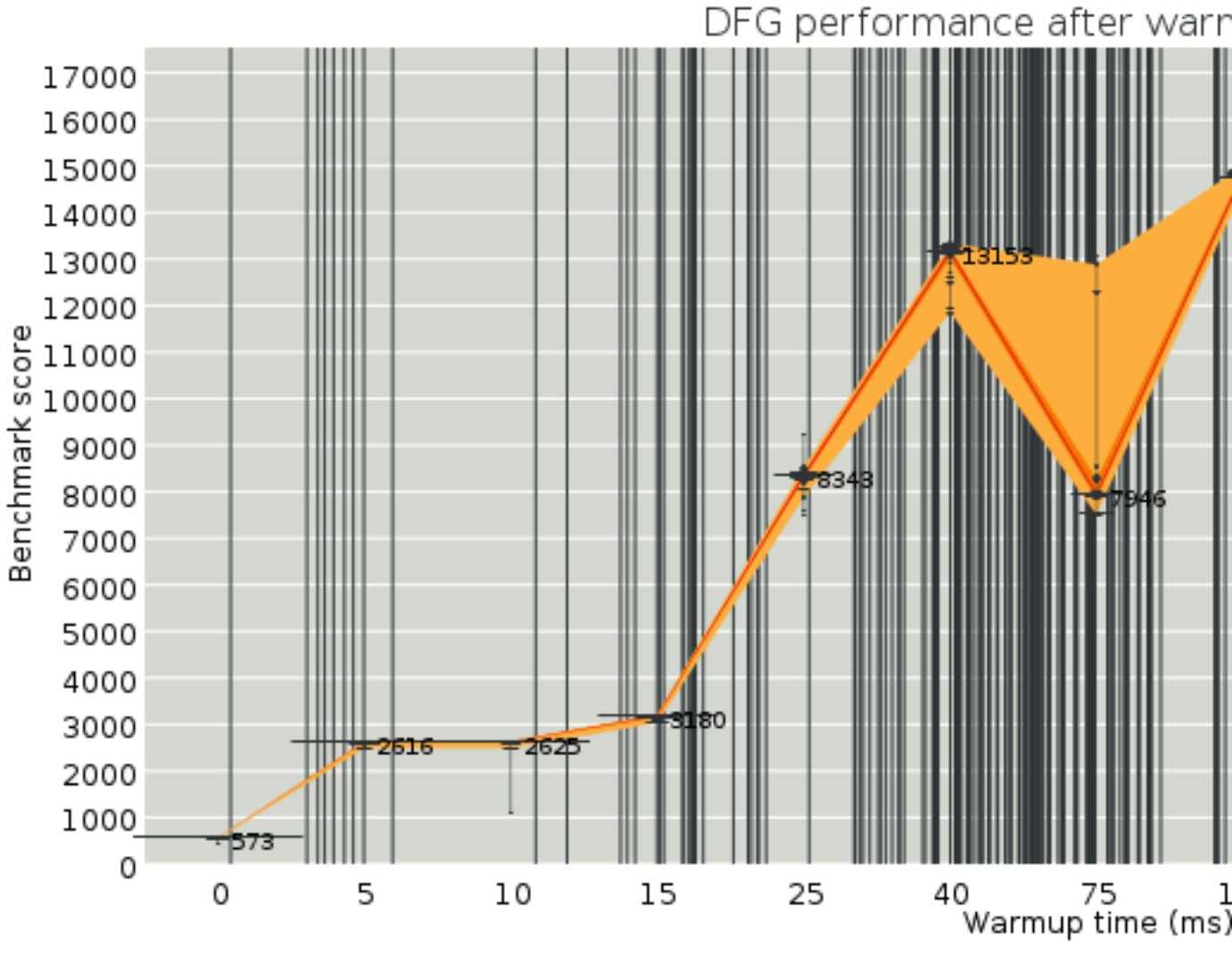
```
TaskControlBlock.prototype.isHeldOrSuspended = function () {
  return (this.state & STATE_HELD) != 0
         || (this.state == STATE_SUSPENDED);
};

GetLocal
          0x7f4d028abbf4: mov -0x38(%r13), %rax
CheckStructure
          0x7f4d028abbf8: mov $0x7f4d00109c80, %r11
          0x7f4d028abc02: cmp %r11, (%rax)
          0x7f4d028abc05: jnz 0x7f4d028abd15
GetByOffset
          0x7f4d028abc0b: mov 0x38(%rax), %rax
GetGlobalVar
          0x7f4d028abc0f: mov $0x7f4d479cdca8, %rdx
          0x7f4d028abc19: mov (%rdx), %rdx
BitAnd
          0x7f4d028abc1c: cmp %r14, %rax
          0x7f4d028abc1f: jb 0x7f4d028abd2b
          0x7f4d028abc25: cmp %r14, %rdx
          0x7f4d028abc28: jb 0x7f4d028abd41
          0x7f4d028abc2e: and %edx, %eax
```

0.11

```
CompareEq
          0x7f4d028abc30: xor %ecx, %ecx
          0x7f4d028abc32: cmp %ecx, %eax
          0x7f4d028abc34: setz %al
          0x7f4d028abc37: movzx %al, %eax
          0x7f4d028abc3a: or $0x6, %eax
LogicalNot
          0x7f4d028abc3d: xor $0x1, %rax
SetLocal
          0x7f4d028abc41: mov %rax, 0x0(%r13)
Branch
          0x7f4d028abc45: test $0x1, %eax
          0x7f4d028abc4b: jnz 0x7f4d028abc87
...
          0x7f4d028abc97: ret
(End Of Main Path)
...
```

0.12 DeltaBlue



Speedup: 4.4X
Prototypes, inlining

0.13 Inlining

At 20ms:

```
Delaying optimization for
  Constraint.prototype.satisfy (in loop)
  because of insufficient profiling.
```

Eventually succeeds after 4 more times and 20 more ms; see `--maximumOptimizationDelay`.

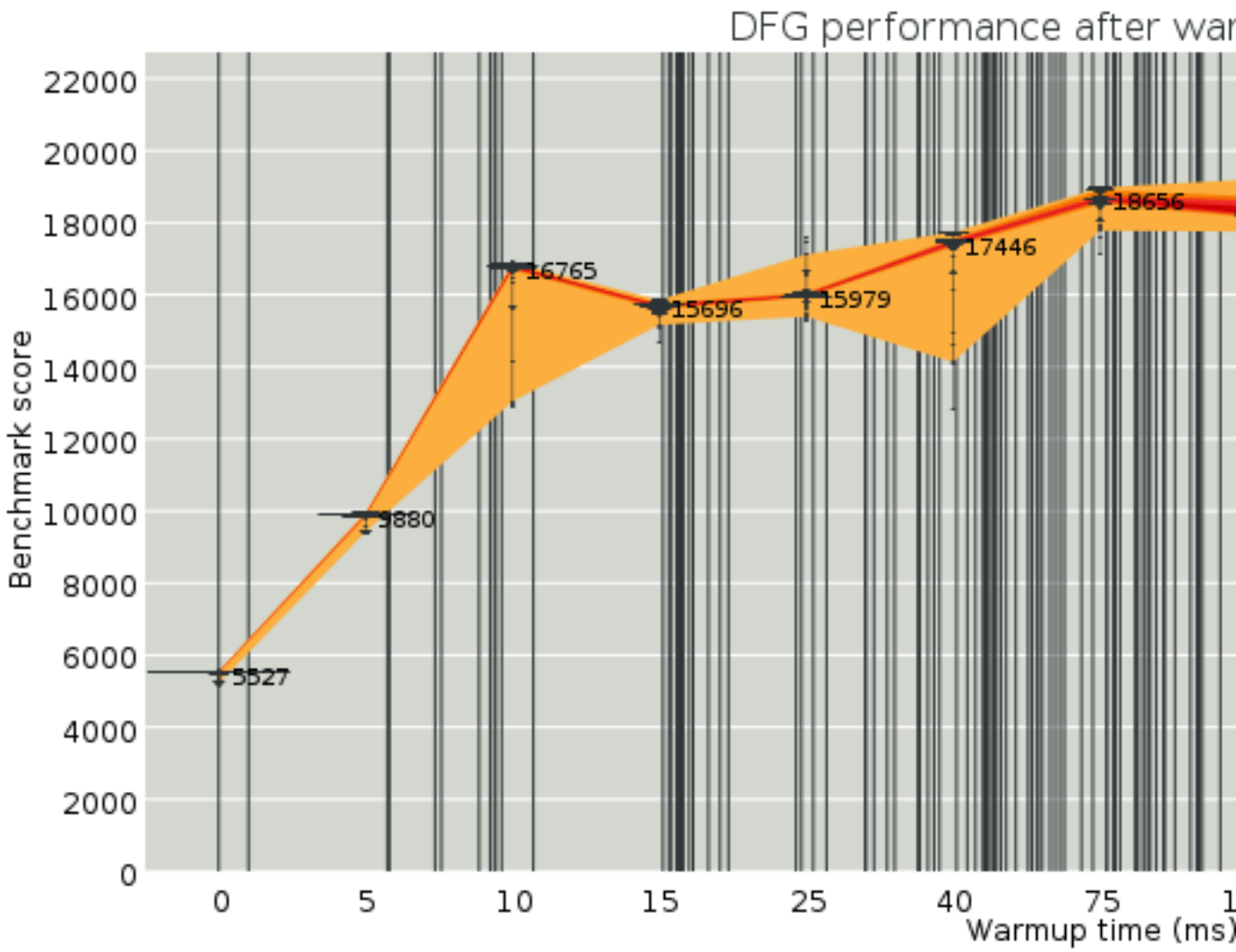
0.14 1000 cuts

One function optimized about 20ms in:

```
Planner.prototype.addConstraintsConsumingTo =
  function (v, coll) {
    var determining = v.determinedBy;
    var cc = v.constraints;
    for (var i = 0; i < cc.size(); i++) {
      var c = cc.at(i);
      if (c != determining && c.isSatisfied())
        coll.add(c);
    }
}
```

Many small marginal gains

0.15 Crypto



Speedup: 4.1X

Integers, arrays

0.16

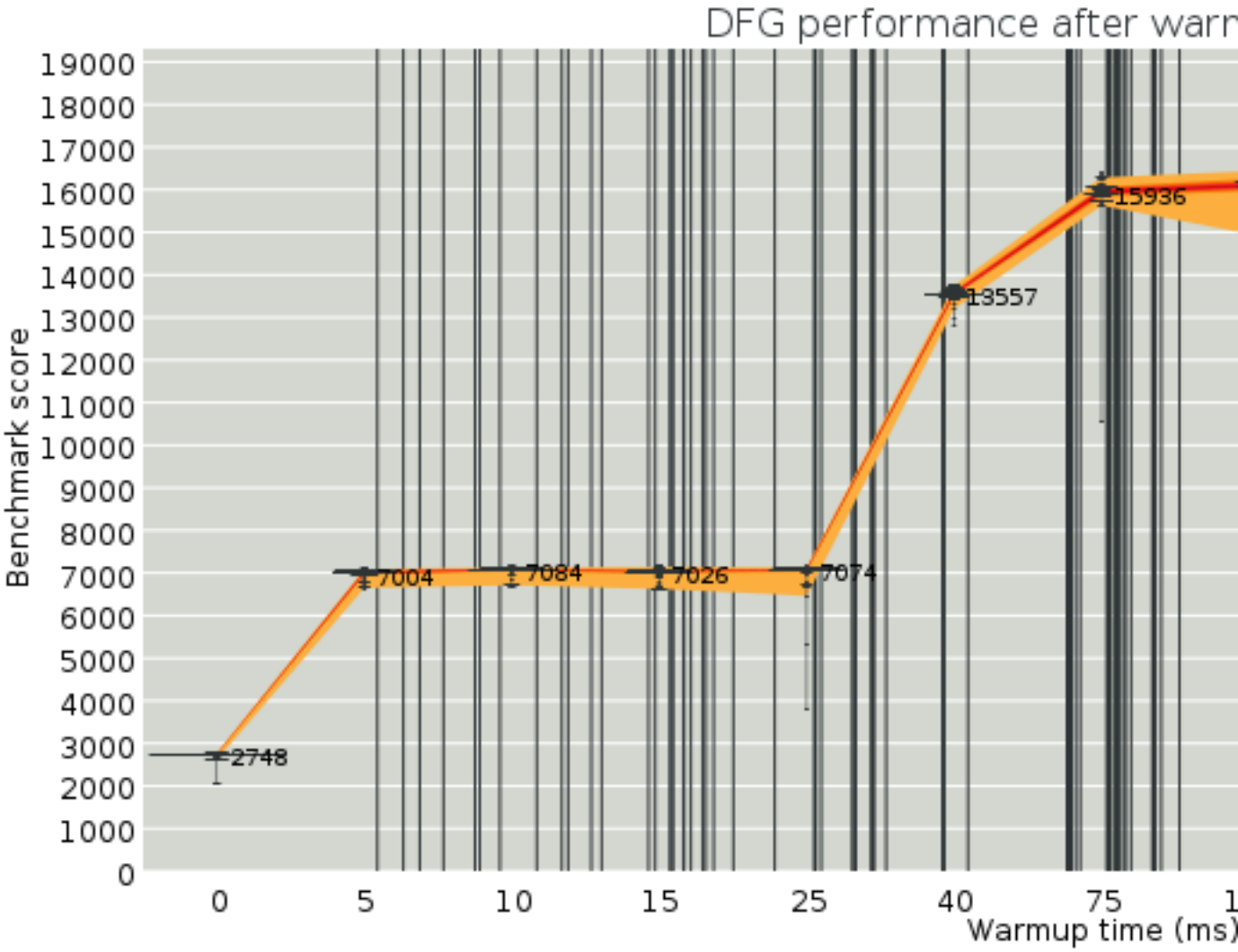
```
function am3(i,x,w,j,c,n) {
  var this_array = this.array;
  var w_array    = w.array;

  var xl = x&0x3fff, xh = x>>14;
  while(--n >= 0) {
    var l = this_array[i]&0x3fff;
    var h = this_array[i++]>>14;
    var m = xh*l+h*xl;
    l = xl*l+((m&0x3fff)<<14)+w_array[j]+c;
    c = (l>>28)+(m>>14)+xh*h;
    w_array[j++] = l&0xffffffff;
  }
  return c;
}
```

0.17 `var l = this_array[i]&0x3fff`

```
      GetLocal: this_array
                0x7f4d02909bf6: mov 0x0(%r13), %r10
      GetLocal: i (int32; type check hoisted)
                0x7f4d02909bfa: mov -0x40(%r13), %eax
      GetButterfly: this_array
                0x7f4d02909bfe: mov 0x8(%r10), %rdx
      GetByVal: this_array[i] (array check hoisted)
                0x7f4d02909c02: cmp -0x4(%rdx), %eax
                0x7f4d02909c05: jae 0x7f4d02909ed2
                0x7f4d02909c0b: mov 0x10(%rdx,%rax,8), %rcx
                0x7f4d02909c10: test %rcx, %rcx
                0x7f4d02909c13: jz 0x7f4d02909ee8
      BitAnd:
                0x7f4d02909c19: cmp %r14, %rcx
                0x7f4d02909c1c: jb 0x7f4d02909efe
                0x7f4d02909c22: mov %rcx, %rbx
                0x7f4d02909c25: and $0x3fff, %ebx
```

0.18 RayTrace



Speedup: 2.5X

Floating point, objects with floating-point fields

0.19 `normalize()`

```
normalize : function() {
    var m = this.magnitude();
    return new Flog.RayTracer.Vector(this.x / m,
                                     this.y / m,
                                     this.z / m);
},
```

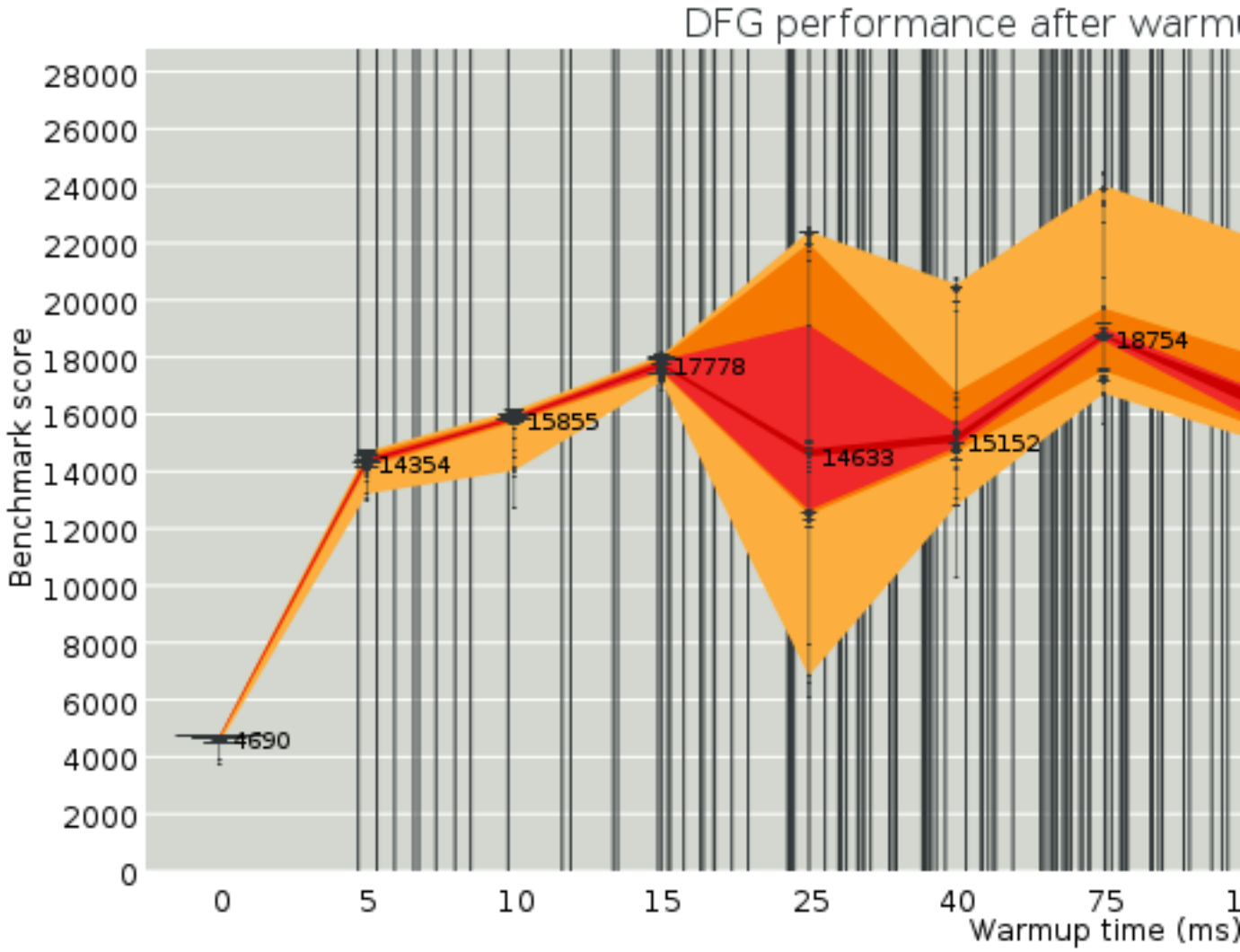
DFG inlines as it compiles: inlines `this.magnitude()`

```
ArithDiv:
          0x7f4d0298164b: divsd %xmm1, %xmm0
```

```
SetLocal:
          0x7f4d0298164f: movd %xmm0, %rdx
          0x7f4d02981654: sub %r14, %rdx
          0x7f4d02981657: mov %rdx, 0x20(%r13)
```

No typed fields (yet)

0.20 EarleyBoyer



Speedup: 2.0X

Function calls, small short-lived allocations

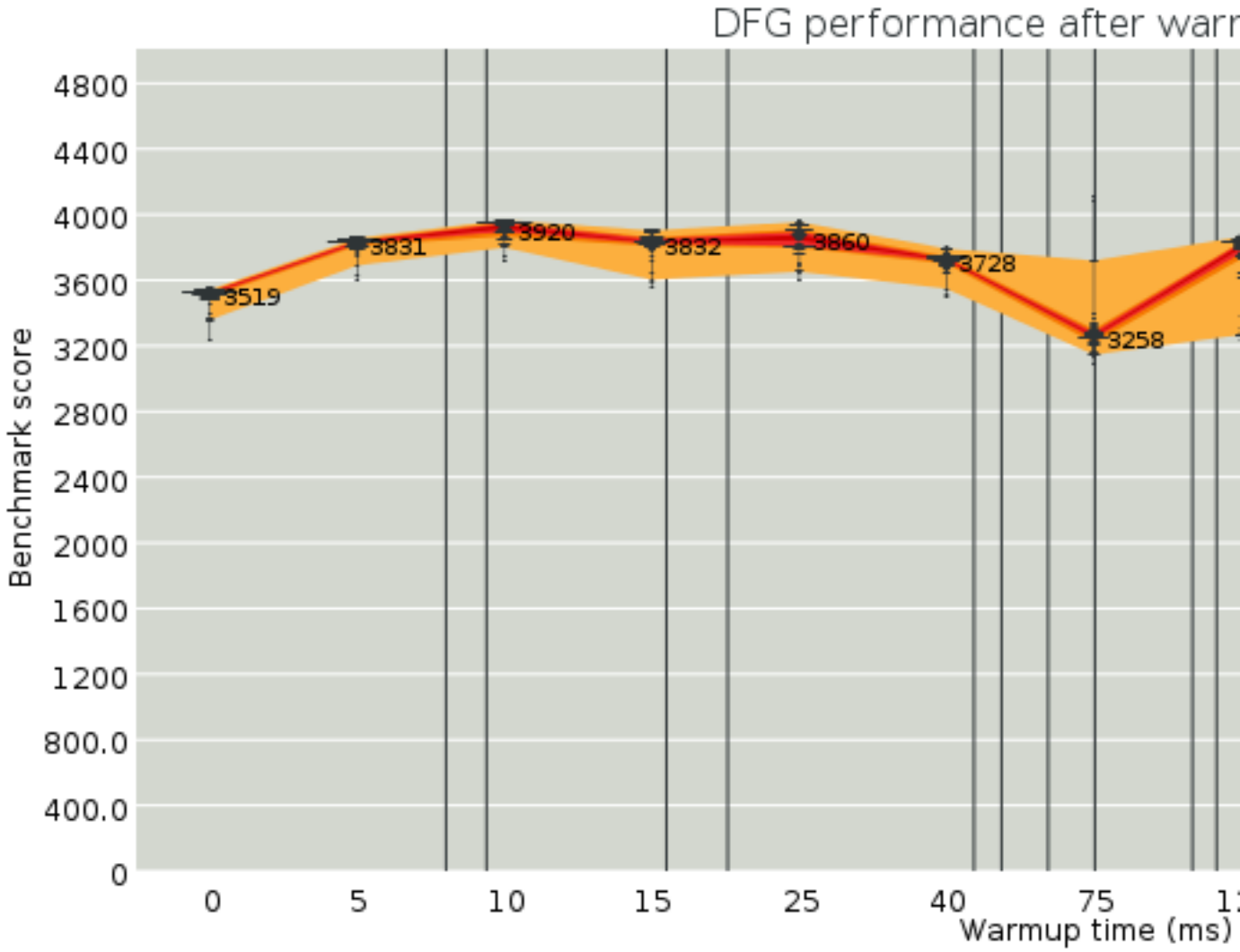
0.21 EarleyBoyer

“Performance is a distribution, not a value”

Wide distribution indicates nonuniform performance

Cause in this case: nonincremental mark GC

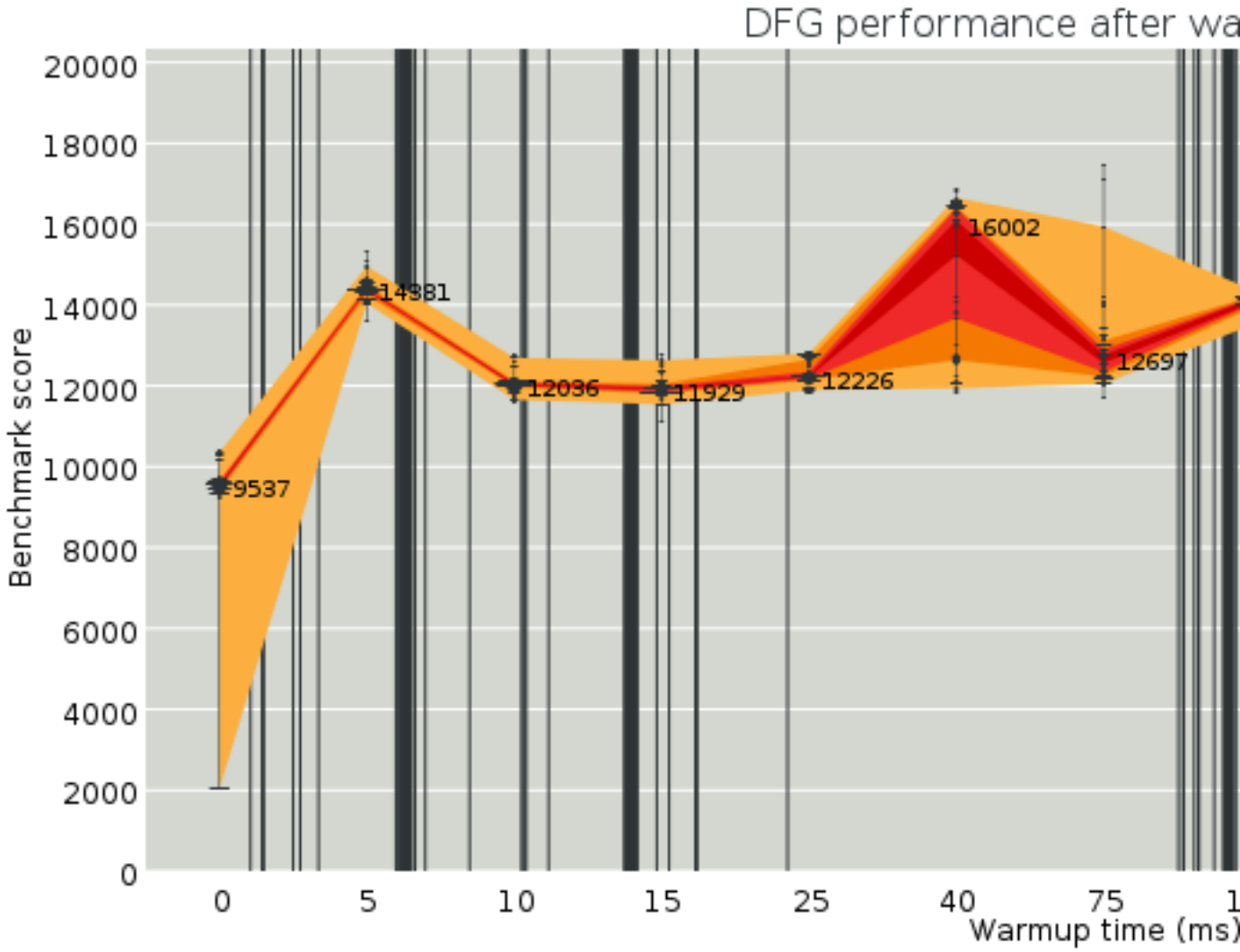
0.22 RegExp



Speedup: 1.2X

Regexp compiler test; DFG of no help

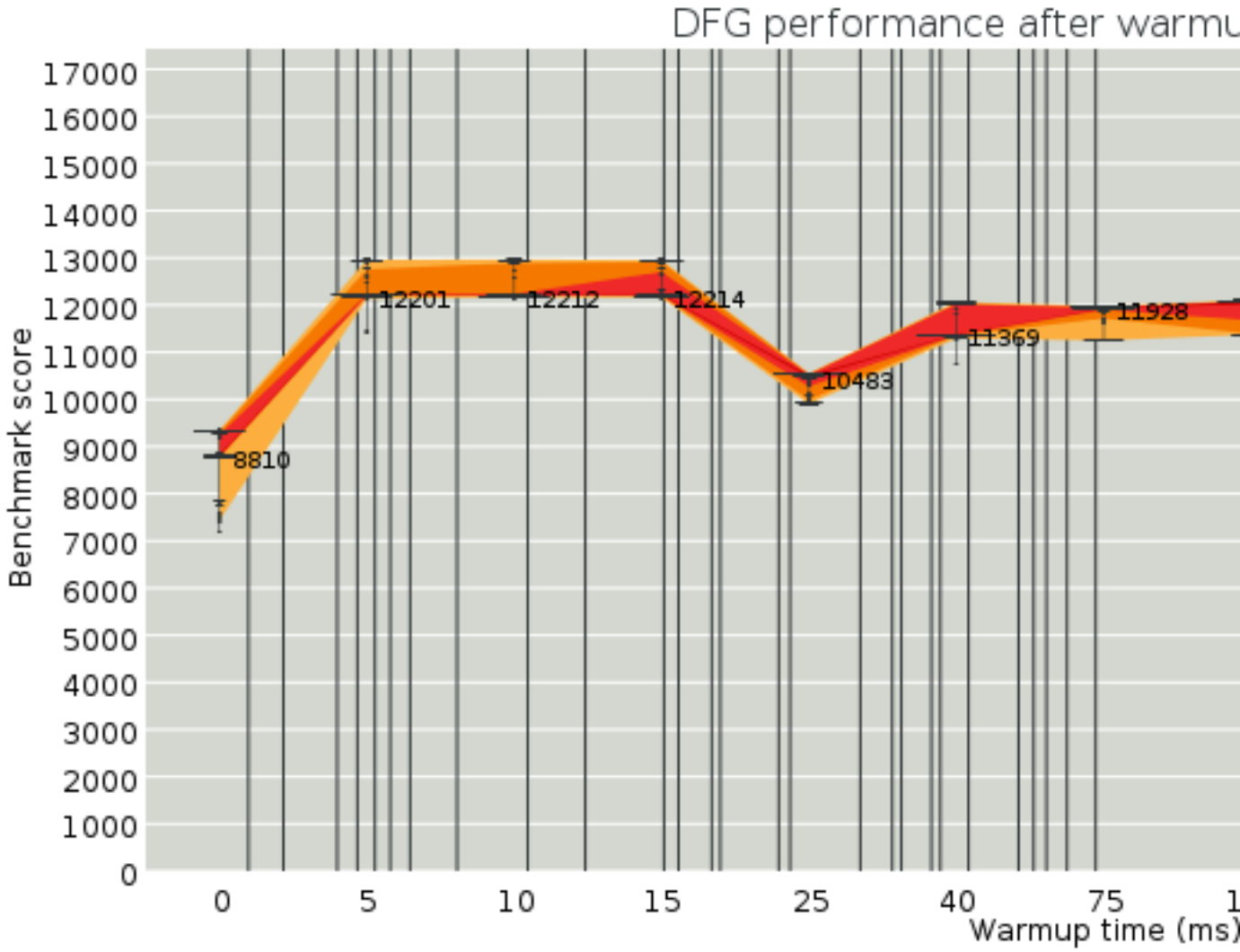
0.23 Splay



Speedup: 1.4X

GC test, huge variance

0.24 NavierStokes



Speedup: 3.0X

Floating point arrays, large floating-point functions

0.25 No automagic double arrays

```
GetByVal:
          0x7f4d02acec1f: cmp -0x4(%rcx), %r9d
          0x7f4d02acec23: jae 0x7f4d02acee0b
          0x7f4d02acec29: mov 0x10(%rcx,%r9,8), %rbx
          0x7f4d02acec2e: test %rbx, %rbx
          0x7f4d02acec31: jz 0x7f4d02acee21
GetLocal:
          0x7f4d02acec37: mov -0x50(%r13), %rdi
Int32ToDouble:
          0x7f4d02acec3b: cmp %r14, %rbx
```

```
0x7f4d02acec3e: jae 0x7f4d02acec5d
0x7f4d02acec44: test %rbx, %r14
0x7f4d02acec47: jz 0x7f4d02acee37
0x7f4d02acec4d: mov %rbx, %rsi
0x7f4d02acec50: add %r14, %rsi
0x7f4d02acec53: movd %rsi, %xmm0
0x7f4d02acec58: jmp 0x7f4d02acec61
0x7f4d02acec5d: cvtsi2sd %ebx, %xmm0
```

0.26 Getting data out of JSC

`jsc --options`

`jsc -d`

`jsc --showDFGDisassembly=true`

`-DJIT_ENABLE_VERBOSE=1`, `-DJIT_ENABLE_VERBOSE_OSR=1` and timestamping hacks on `dataLog`

0.27 Comparative Literature

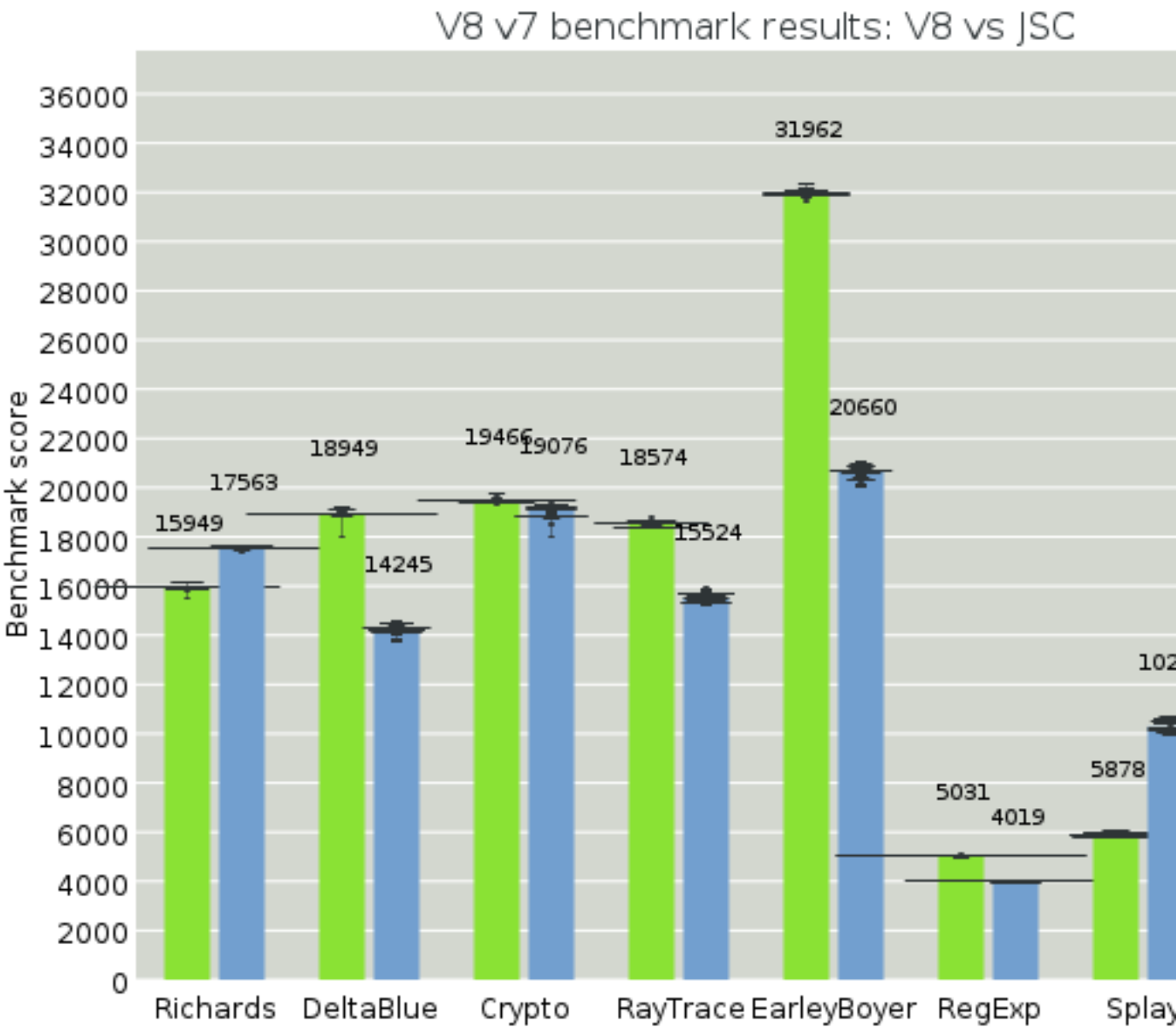
V8 vs JSC: fight!

Does JSC beat V8?

Does JSC meet V8?

Does V8 beat JSC?

0.28 Yes



0.29 Questions?

- igalia.com/compilers
- wingolog.org
- @andywingo
- wingolog.org/pub/jsconf-eu-2012-slides.pdf