WHAT ELSE HAS MY COMPILER DONE FOR ME LATELY?

UNBOLTING THE COMPILER'S LID...AGAIN

Matt Godbolt
@mattgodbolt
matt@godbolt.org

OUTLINE

- Compiler Explorer story
- Assembly 101
- What else has my compiler done for me lately?
- Behind the scenes of Compiler Explorer

BACKSTORY

```
int sum(const vector<int> &v) {
  int result = 0;
  for (size_t i = 0; i < v.size(); ++i)
    result += v[i];
  return result;
}
```

```
int sum(const vector<int> &v) {
  int result = 0;
  for (int x : v) result += x;
  return result;
}
```

Is one better than the other?

Look at compiler output!

WARNING

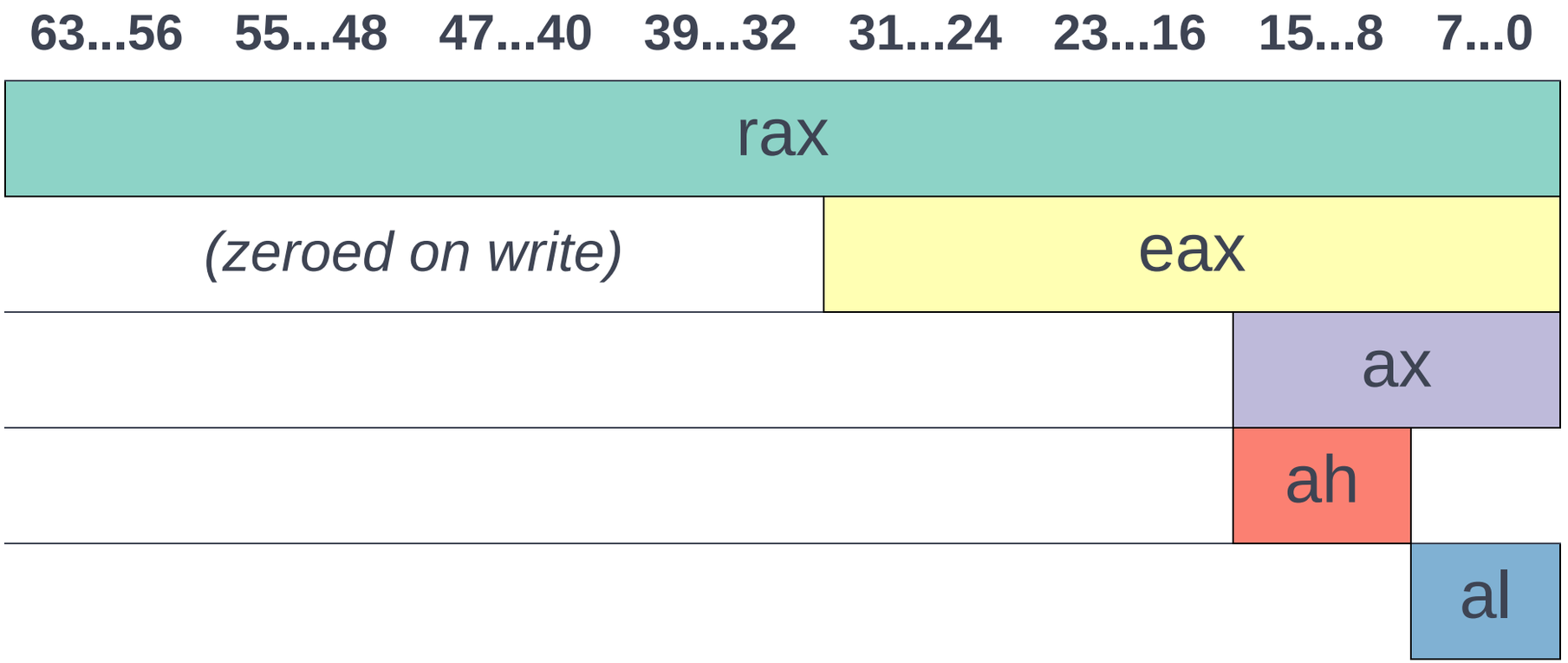
- Reading assembly alone can be misleading
- *Always* measure too
- Google Benchmark
- quick-bench.com

X86 ASSEMBLY 101

REGISTERS

- `rax`, `rbx`, `rcx`, `rdx`, `rsp`, `rbp`, `rsi`, `rdi`, `r8-r15`
- `xmm0-xmm15`
- `rdi`, `rsi`, `rdx`... arguments
- `rax` is return value

REGISTERS



And so on for rdi, rsi, rcx, rdx...

INSTRUCTIONS

```
op
op dest
op dest, src
op dest, src1, src2
```

- `op` is e.g. `call`, `ret`, `add`, `sub`, `cmp`...
- `dest`, `src` is register or memory reference:

  `[base + reg1`$_{\texttt{opt}}$ `+ reg2*(1, 2, 4 or 8)`$_{\texttt{opt}}$`]`

(Intel asm syntax)

INSTRUCTIONS

```
mov eax, DWORD PTR [r14]
add rax, rdi
add eax, DWORD PTR [r14+4]
sub eax, DWORD PTR [r14+4*rbx]
lea rax, [r14+4*rbx]
xor edx, edx
```

```
int eax = *r14;   // int *r14;
rax += rdi;
eax += r14[1];
eax -= r14[rbx];
int *rax = &r14[rbx];
edx = 0;
```

SUMMARY

- Registers: `rax`, `rbx`, `rcx` ...
- Size: `rax`, `eax`, `ax` ...
- Params: `rdi`, `rsi`, `rdx`, `rcx` ...
- Result: `rax`
- `op dest, src`
- `dest`, `src` are registers or memory

WHERE WERE WE?

```
int sum(const vector<int> &v) {
  int result = 0;
  for (size_t i = 0; i < v.size(); ++i)
    result += v[i];
  return result;
}
```

```
int sum(const vector<int> &v) {
  int result = 0;
  for (int x : v) result += x;
  return result;
}
```

Which is better?

COMPILER EXPLORER V0.1

```
$ g++ /tmp/test.cc -O2 -c -S -o - -masm=intel \
    | c++filt \
    | grep -vE '\s+\.'
```

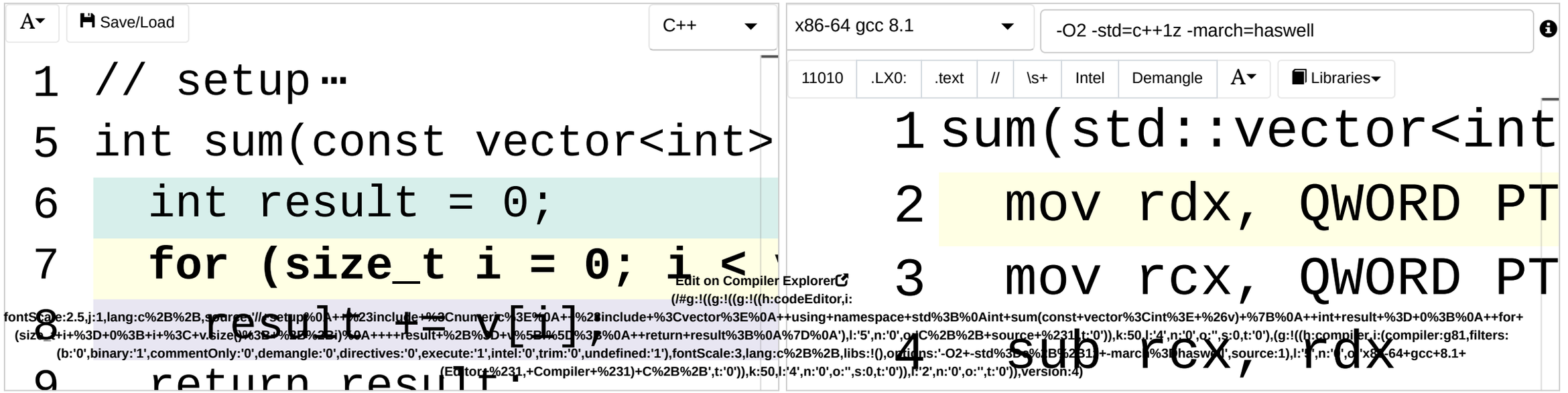
```
sum(std::vector<int, std::allocator<int> > const&):
.LFB786:
    mov rcx, QWORD PTR [rdi]
    mov rax, QWORD PTR 8[rdi]
    sub rax, rcx
    shr rax, 2
    mov rsi, rax
    ...
```

COMPILER EXPLORER V0.1

Not very pretty

TO THE WEB!

DEMO


A
Save/Load
C++
1 // setup…
5 int sum(const vector<int>
6 int result = 0;
7 for (size_t i = 0; i <
8 result += v[i];
9 return result;
x86-64 gcc 8.1
-O2 -std=c++1z -march=haswell
11010
.LX0:
.text
//
\s+
Intel
Demangle
A
Libraries
1 sum(std::vector<int
2 mov rdx, QWORD PT
3 mov rcx, QWORD PT
4 sub rcx, rdx
Edit on Compiler Explorer
(/#g:!((g:!((g:!((h:codeEditor,i:
fontScale:2.5,j:1,lang:c%2B%2B,source:'//+setup%0A++%23include+%3Cnumeric%3E%0A++%23include+%3Cvector%3E%0A++using+namespace+std%3B%0Aint+sum(const+vector%3Cint%3E+%26v)+%7B%0A++int+result+%3D+0%3B%0A++for+
(size_t+i+%3D+0%3B+i+%3C+v.size()%3B+%2B%2Bi)%0A++++result+%2B%3D+v%5Bi%5D%3B%0A++return+result%3B%0A%7D%0A'),l:'5',n:'0',o:'C%2B%2B+source+%231',t:'0')),k:50,l:'4',n:'0',o:'',s:0,t:'0'),(g:!((h:compiler,i:(compiler:g81,filters:
(b:'0',binary:'1',commentOnly:'0',demangle:'0',directives:'0',execute:'1',intel:'0',trim:'0',undefined:'1'),fontScale:3,lang:c%2B%2B,libs:!(),options:'-O2+-std%3Dc%2B%2B1z+-march%3Dhaswell',source:1),l:'5',n:'0',o:'x86-64+gcc+8.1+
(Editor+%231,+Compiler+%231)+C%2B%2B',t:'0')),k:50,l:'4',n:'0',o:'',s:0,t:'0')),l:'2',n:'0',o:'',t:'0')),version:4)

WALKTHROUGH

```
int sum(const vector<int> &v) {
```

```
; rdi = const vector<int> *
mov rdx, QWORD PTR [rdi]   ; rdx = *rdi     ≡ begin()
mov rcx, QWORD PTR [rdi+8] ; rcx = *(rdi+8) ≡ end()
```

```
template<typename T> struct _Vector_impl {
  T *_M_start;
  T *_M_finish;
  T *_M_end_of_storage;
};
```

TRADITIONAL

```
sub rcx, rdx ; rcx = end-begin
mov rax, rcx
shr rax, 2   ; (end-begin)/4
je .L4
add rcx, rdx
xor eax, eax
```

```
size_t size() const noexcept {
  return _M_finish - _M_start;
}
```

RANGE

```
xor eax, eax
cmp rdx, rcx ; begin==end?
je .L4
```

```
auto __begin = begin(v);
auto __end = end(v);
for (auto __it = __begin;
     __it != __end;
     ++it)
```

WALKTHROUGH

```
; rcx ≡ end, rdx = begin, eax = 0
.L3:
  add eax, DWORD PTR [rdx]     ; eax += *rdx
  add rdx, 4                   ; rdx += sizeof(int)
  cmp rdx, rcx                 ; is rdx == end?
  jne .L3                      ;   if not, loop
  ret                          ; we're done
```

BACKSTORY

SO, WHICH APPROACH IS BEST?

ALSO

- Optimizer settings are important
- `std::accumulate`

WHAT HAS MY COMPILER DONE FOR ME LATELY?

MULTIPLICATION

```
int mulByY(int x, int y) {
  return x * y;
}
```

View

```
mulByY(int, int):
  mov eax, edi
  imul eax, esi
  ret
```

MULTIPLICATION

```
int mulByConstant(int x) { return x * 2; }
```

View

MULTIPLICATION

```
int mulBy65599(int a) {
  return (a << 16) + (a << 6) - a;
  //          ^           ^
  //      a * 65536       |
  //                   a * 64
  // 65536a + 64a - 1a = 65599a
}
```

View

DIVISION

```
int divByY(int x, int y) {
  return x / y;
}
int modByY(int x, int y) {
  return x % y;
}
```

View

Haswell 32-bit divide - 22-29 cycles!

```
divByY(int, int):
  mov eax, edi
  cdq
  idiv esi
  ret
modByY(int, int):
  mov eax, edi
  cdq
  idiv esi
  mov eax, edx
  ret
```

DIVISION

```
unsigned divByConstant(unsigned x) { return x / 2; }
```

View

DIVISION

```
mov eax, edi          ; eax = x
mov edx, 0xaaaaaaab
mul edx               ; eax:edx = x * 0xaaaaaaab
mov eax, edx          ;   (x * 0xaaaaaaab) >> 32
                      ; ≡ (x * 0xaaaaaaab) / 0x10000000
                      ; ≡ x * 0.666666667
shr eax               ; x * 0.333333333
ret
```

MODULUS

```
int modBy3(unsigned x) {
  return x % 3;
}
```

View

```
mov eax, edi
mov edx, 0xaaaaaaab
mul edx
mov eax, edx
shr eax
lea eax, [rdx+rdx*2]
sub edi, eax
mov eax, edi
ret
```

WHY MODULUS?

- Bucket selection in hash maps
- libc++ special-cases power-of-two
- boost multi_index

COUNTING BITS

```
int countSetBits(int a) {
  int count = 0;
  while (a) {
    count++;
    a &= (a-1);
  }
  return count;
}
```

View

SUMMATION

```
constexpr int sumTo(int x) {
  int sum = 0;
  for (int i = 0; i <= x; ++i)
    sum += i;
  return sum;
}
int main(int argc, const char *argv[]) {
  return sumTo(20);
}
```

View

SUM(X)

$$\sum_{n=0}^{x} n \equiv \frac{x(x+1)}{2}$$

$$\equiv x + \frac{x(x-1)}{2}$$

WHAT ELSE HAS MY COMPILER DONE FOR ME LATELY?

FUNCTIONS

```
int someFunc(int);
int sum(const vector<int> &v) {
  int result = 0;
  for (size_t i = 0;
       i < v.size(); ++i)
    result += someFunc(v[i]);
  return result;
}
```

View

Accumulate version

```
.L3:
  mov edi, DWORD PTR [rdx+rbx*4]
  inc rbx
  call someFunc(int)
  mov rdx, QWORD PTR [rbp+0]
  add r12d, eax
  mov rax, QWORD PTR [rbp+8]
  sub rax, rdx
  sar rax, 2
  cmp rbx, rax
  jb .L3
```

ALIASING 1/2

```
// Worst API ever...
void sumToFirstZero(
    int *first, int &num,
    int &sum) {
  sum = 0;
  int i;
  for (i = 0; i < num; ++i) {
    if (first[i] == 0) break;
    sum += first[i];
  }
  num = i;
}
```

View

```
.L4:
  inc rax
  mov ecx, DWORD PTR [rdi-4+rax*4]
  test ecx, ecx
  je .L2
.L3:
  add r8d, ecx
  mov r9d, eax
  mov DWORD PTR [rdx], r8d
  cmp DWORD PTR [rsi], eax
  jg .L4
```

Alternative implementation

ALIASING 2/2

```
void maxArray(double* x, double* y) {
  for (int i = 0; i < 65536; i++) {
    if (y[i] > x[i])
      x[i] = y[i];
  }
}
```

View Better

```
maxArray(double*, double*):
  lea rax, [rsi+32]
  cmp rdi, rax
  jnb .L10
  lea rax, [rdi+32]
  cmp rsi, rax
  jb .L9
...
```

HEAP ELISION

```
int func() {
  auto a = make_unique<int>(42);
  auto b = make_unique<int>(24);
  return *a + *b;
}
```

View

TERNARY OPERATOR

```
int func(int i) {
  return (i < 0) ? 1234 : 5678;
}
```

View

```
func(int):
  mov eax, edi
  sar eax, 31
  and eax, -4444
  add eax, 5678
  ret
```

VIRTUAL METHODS

```
struct Coster { virtual int costFor(int x) = 0; };
int totalCost(Coster &coster,
              const vector<int> &v) {
  int result = 0;
  for (auto i : v)
    result += coster.costFor(i);
  return result;
}
```

View

WHAT HAS MY COMPILER DONE FOR ME LATELY?

A lot!

HOW IT WORKS

HOW IT WORKS - BACKEND

- Written in `node.js`
- Runs on Amazon

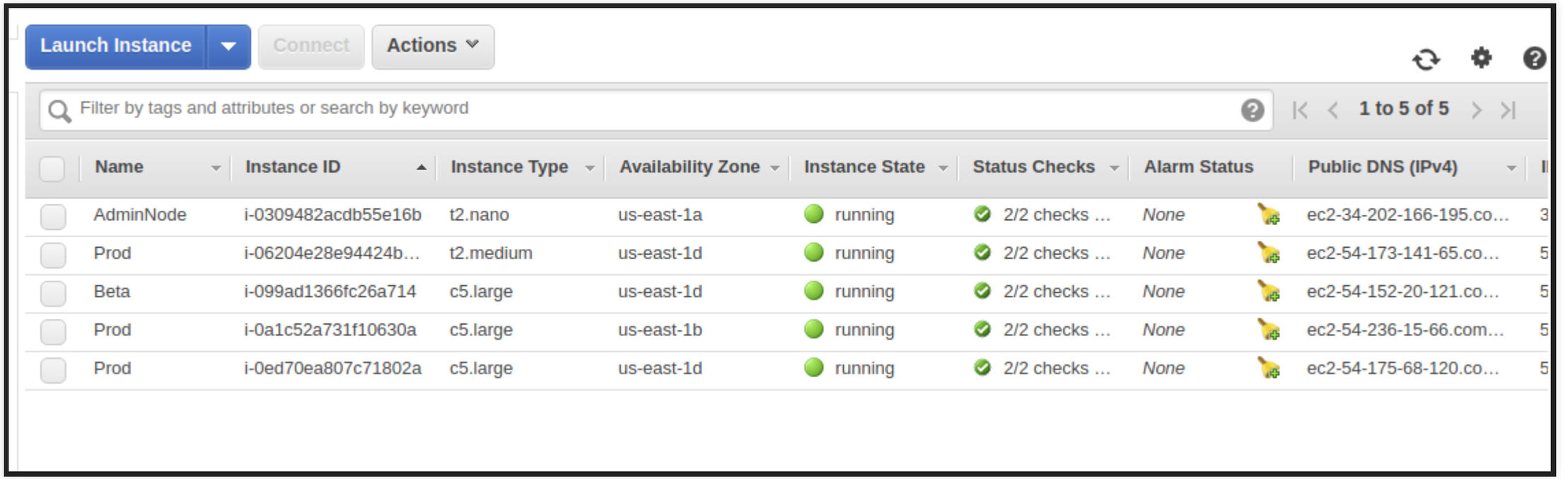
NODE.JS

```
function compile(req, res, next) {
  // exec compiler, feed it req.body, parse output
}
var webServer = express();
var apiHandler = express.Router();
apiHandler.param('compiler',
    function (req, res, next, compiler) {
  req.compiler = compiler;
  next();
});
apiHandler.post('/compiler/:compiler/compile', compile);
webServer.use('/api', apiHandler);
webServer.listen(10240);
```

AMAZON EC2

- Edge cache
- Load balancer
- Virtual machines
- Docker images
- Shared compiler storage

AMAZON EC2



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	I
AdminNode	i-0309482acdb55e16b	t2.nano	us-east-1a	running	2/2 checks ...	None	ec2-34-202-166-195.co...	3
Prod	i-06204e28e94424b...	t2.medium	us-east-1d	running	2/2 checks ...	None	ec2-54-173-141-65.co...	5
Beta	i-099ad1366fc26a714	c5.large	us-east-1d	running	2/2 checks ...	None	ec2-54-152-20-121.co...	5
Prod	i-0a1c52a731f10630a	c5.large	us-east-1b	running	2/2 checks ...	None	ec2-54-236-15-66.com...	5
Prod	i-0ed70ea807c71802a	c5.large	us-east-1d	running	2/2 checks ...	None	ec2-54-175-68-120.co...	5

THE COMPILERS

- Built through docker images
- Compilers stored on S3
- OSS ones publically available
- MS compilers via WINE

HOW IT WORKS - SECURITY

- Compilers: huge attack vector
- Principle of "what's the worst could happen"
- Docker
- `LD_PRELOAD`

HOW IT WORKS - FRONTEND

- Microsoft's Monaco
- GoldenLayout

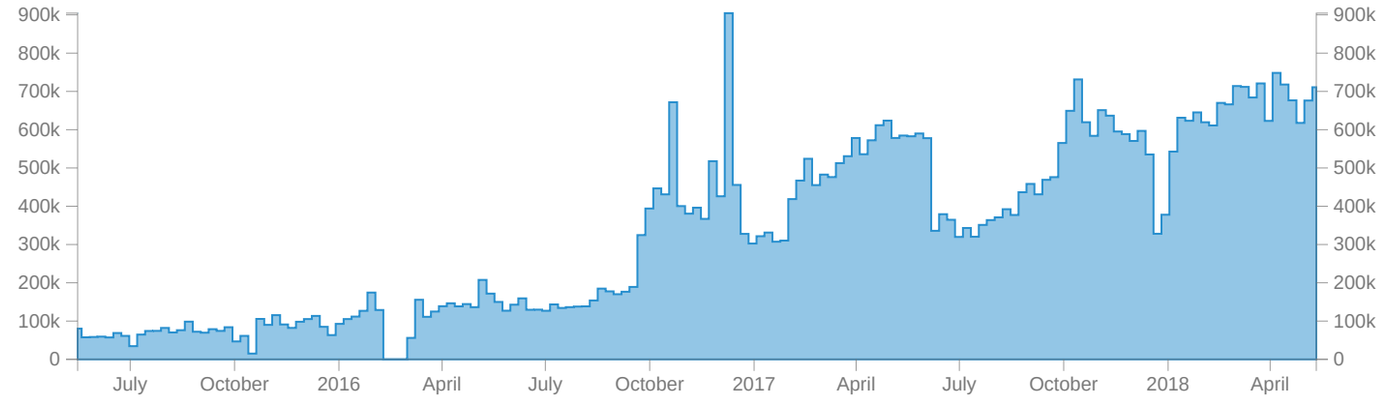
THE CODE

- github.com/mattgodbolt/compiler-explorer
- github.com/mattgodbolt/compiler-explorer-image
- Running locally is easy!

```
$ make
```

USAGE

All Compilations



	latest	min	max	mean	total	std dev	95%	99%
3 years @ 1 week	710,878	15,049	904,444	337,642	51,996,885	229,277	302k – 374k	290k – 385k

StatHat tracks custom stats in your apps with one line of code.

- Up-to-the-minute charts
- Automatic anomaly detection
- 30-day forecasts

OTHER USES

- Code pastebin
- Compiler development
- C++ REPL
- Training resource

COMING SOON...

- Execution support

THANKS

- Thanks to contributors:
  - Rubén Rincón
  - Jared Wyles, Partouf, Simon Brand, Chedy Najjar, Filipe Cabecinhas, Johan Engelen...and the rest!
- Thanks to Patreon folks
- Thanks to awesome C++ community
- Thanks to you!

GO READ SOME ASSEMBLY!

godbolt.org

(AND THANK A COMPILER DEVELOPER)