COMPILER CONFIDENTIAL

ERIC BRUMER

When you think “compiler”…

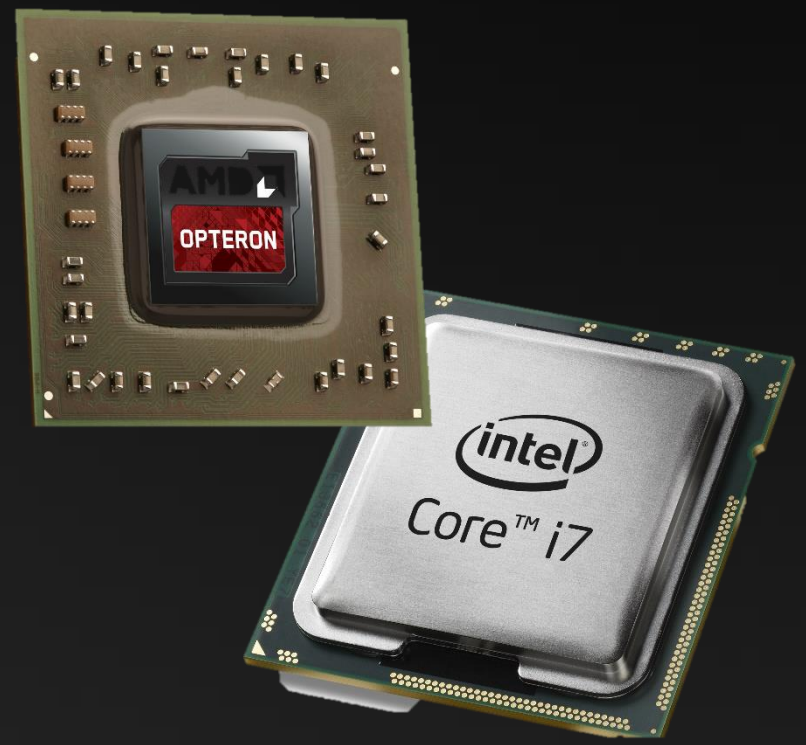
```
c:\work\a.cpp(82): error C2059: syntax error : ')'
c:\work\a.cpp(84): error C2015: too many characters in constant
c:\work\a.cpp(104): error C2015: too many characters in constant
c:\work\a.cpp(104): error C2001: newline in constant
c:\work\a.cpp(116): error C2015: too many characters in constant
c:\work\a.cpp(116): error C2001: newline in constant
c:\work\a.cpp(122): error C2153: hex constants must have at least one hex digit
c:\work\a.cpp(122): error C2001: newline in constant
c:\work\a.cpp(122): error C2015: too many characters in constant
c:\work\a.cpp(134): error C2015: too many characters in constant
c:\work\a.cpp(134): error C2001: newline in constant
c:\work\a.cpp(140): error C2015: too many characters in constant
c:\work\a.cpp(140): error C2001: newline in constant
c:\work\a.cpp(146): error C2001: newline in constant
c:\work\a.cpp(146): error C2015: too many characters in constant
c:\work\a.cpp(154): error C2146: syntax error : missing ';' before identifier 'modern'
c:\work\a.cpp(154): error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
c:\work\a.cpp(154): error C2143: syntax error : missing ';' before '-'
c:\work\a.cpp(154): error C2015: too many characters in constant
c:\work\a.cpp(155): error C2059: syntax error : 'constant'
c:\work\a.cpp(155): error C2059: syntax error : 'bad suffix on number'
c:\work\a.cpp(158): error C2015: too many characters in constant
c:\work\a.cpp(158): error C2059: syntax error : ')'
c:\work\a.cpp(161): error C2001: newline in constant
c:\work\a.cpp(161): error C2015: too many characters in constant
c:\work\a.cpp(164): error C2059: syntax error : 'bad suffix on number'
c:\work\a.cpp(164): error C2059: syntax error : 'constant'
c:\work\a.cpp(168): error C2001: newline in constant
c:\work\a.cpp(168): error C2015: too many characters in constant
c:\work\a.cpp(178): error C2146: syntax error : missing ';' before identifier 'Examples'
c:\work\a.cpp(178): error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
c:\work\a.cpp(178): error C2146: syntax error : missing ';' before identifier 'in'
c:\work\a.cpp(178): error C2146: syntax error : missing ';' before identifier 'C'
c:\work\a.cpp(178): error C2143: syntax error : missing ';' before '++'
c:\work\a.cpp(181): error C2146: syntax error : missing ';' before identifier 'Examples'
c:\work\a.cpp(181): error C4430: missing type specifier - int assumed. Note: C++ does not support default-int
```

```
void test(bool b) {
    try {
        if (b) {
            MyType obj;
            some_func(obj);
            // ...
        }
    } catch (...) {
        // ...
    }
}
```

Destructor placement

Code generation & optimization

```
int binary_search2(std::vector<int> &arr, int value, int lowIndex, int highIndex) {
    int midIndex = (highIndex + lowIndex) / 2;
    int mid = arr[midIndex];
    if (value == mid)
        return midIndex;
    if (lowIndex >= highIndex)
        return -1;
    if (value > mid) {
        int l = midIndex+1;
        int h = highIndex;
        return binary_search2(arr, value, l, h);
    } else { // value < mid
        int l = lowIndex;
        int h = midIndex-1;
        return binary_search2(arr, value, l, h);
    }
}

int binary_search(std::vector<int> &arr, int value) {
    int l = 0;
    int h = arr.size() - 1;
    return binary_search2(arr, value, l, h);
}
```



Make my code run: code generation

Make my code run fast: **optimization**

Mission: expose some optimizer guts

- There will be raw loops
- There will be assembly code
- There will be microarchitecture

I sense much fear in you

AGENDA

CPU hardware landscape

Vectorizing for modern CPUs

Indirect call optimizations

AGENDA

CPU HARDWARE LANDSCAPE

VECTORIZING FOR MODERN CPUS

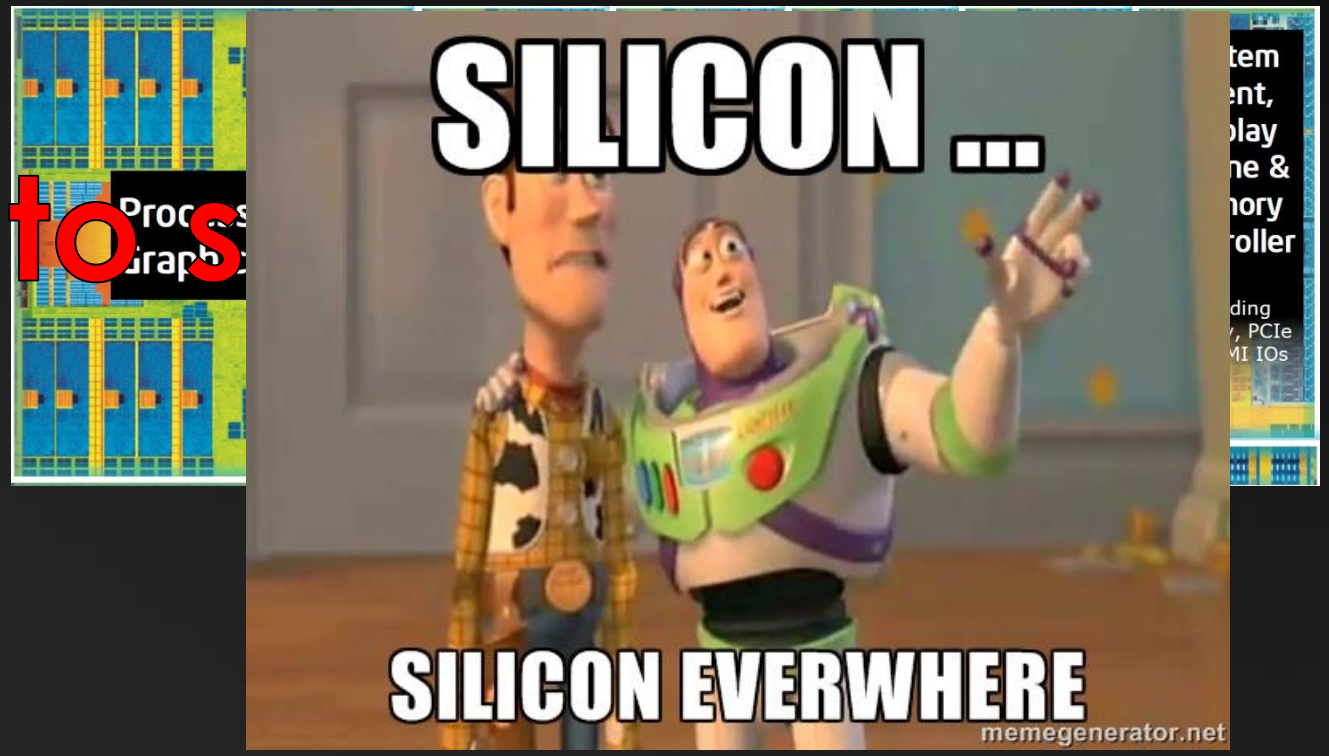
INDIRECT CALL OPTIMIZATIONS

HARDWARE LANDSCAPE

"Yesterday"
3.1 million transistors

Today
1.4 billion transistors

Not to scale



AUTOMATIC VECTORIZATION

- TAKE ADVANTAGE OF (FAST) VECTOR HARDWARE
- EXECUTE MULTIPLE LOOP ITERATIONS IN PARALLEL

```
for (int i=0; i<1000; i++) {
    A[i] = B[i] * C[i];
}
```

Speedup

32 bit operations

Vectorize

```
for (int i=0; i<1000; i+=4) {
    A[i:i+3] = mulps B[i:i+3], C[i:i+3];
}
```

128 bit operations

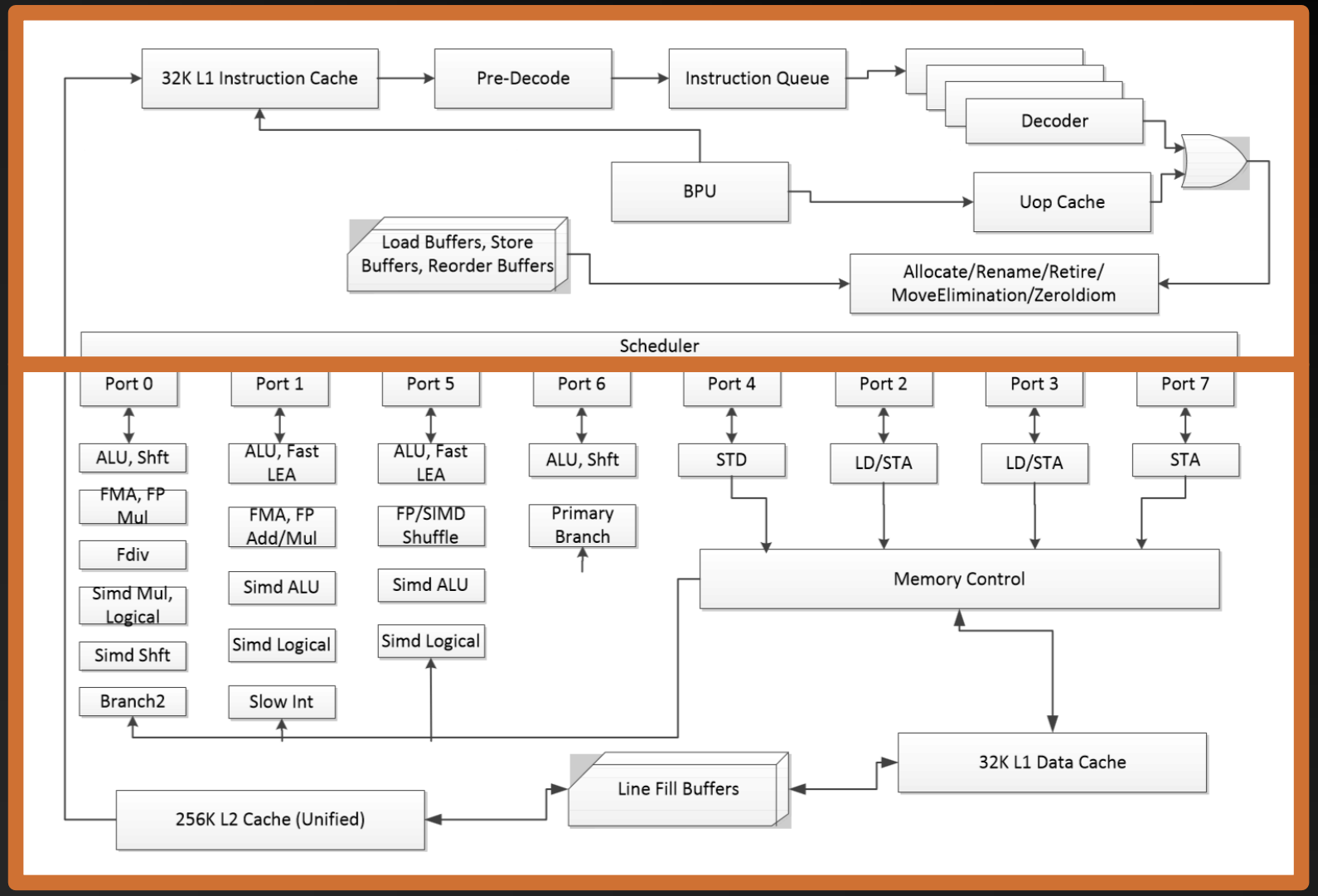
Front-end

- Powerful branch predictor
- Ship instructions to backend as fast as possible

Back-end

- 8 wide super scalar
- Powerful vector units

Haswell core microarchitecture



AGENDA

CPU HARDWARE LANDSCAPE

VECTORIZING FOR MODERN CPUS

INDIRECT CALL OPTIMIZATIONS

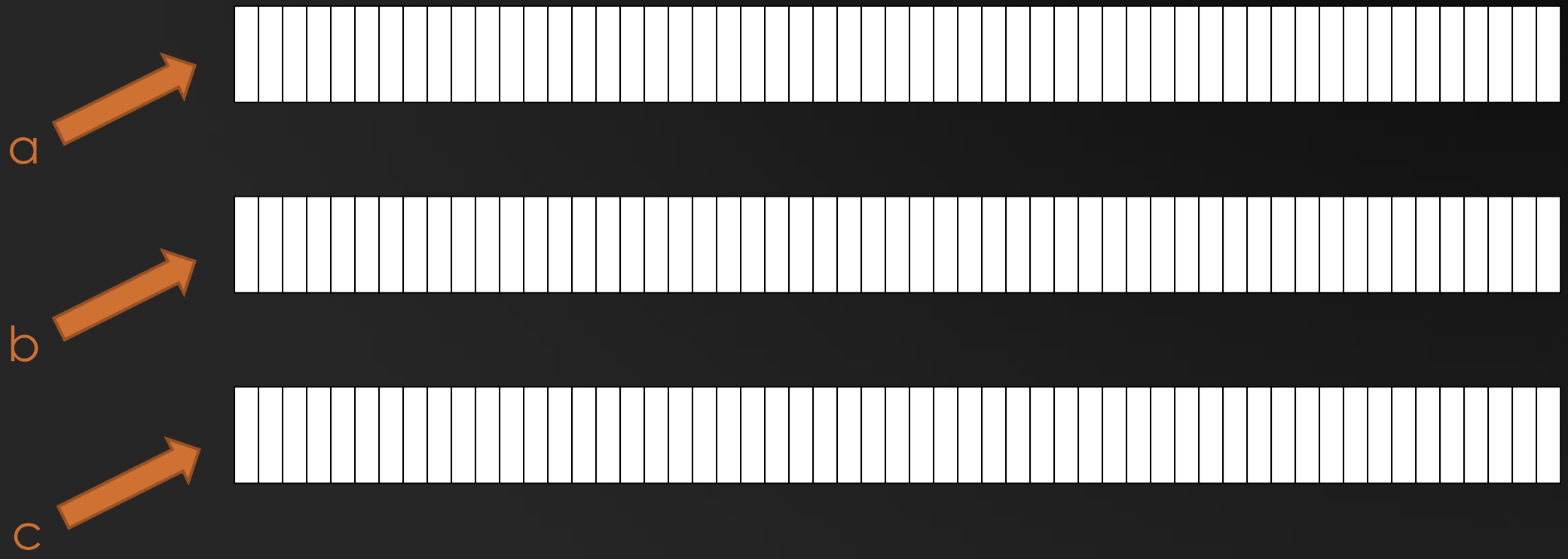
Approach to vectorizing for modern CPUs:

Take advantage of all the extra silicon

Key idea: conditional vectorization

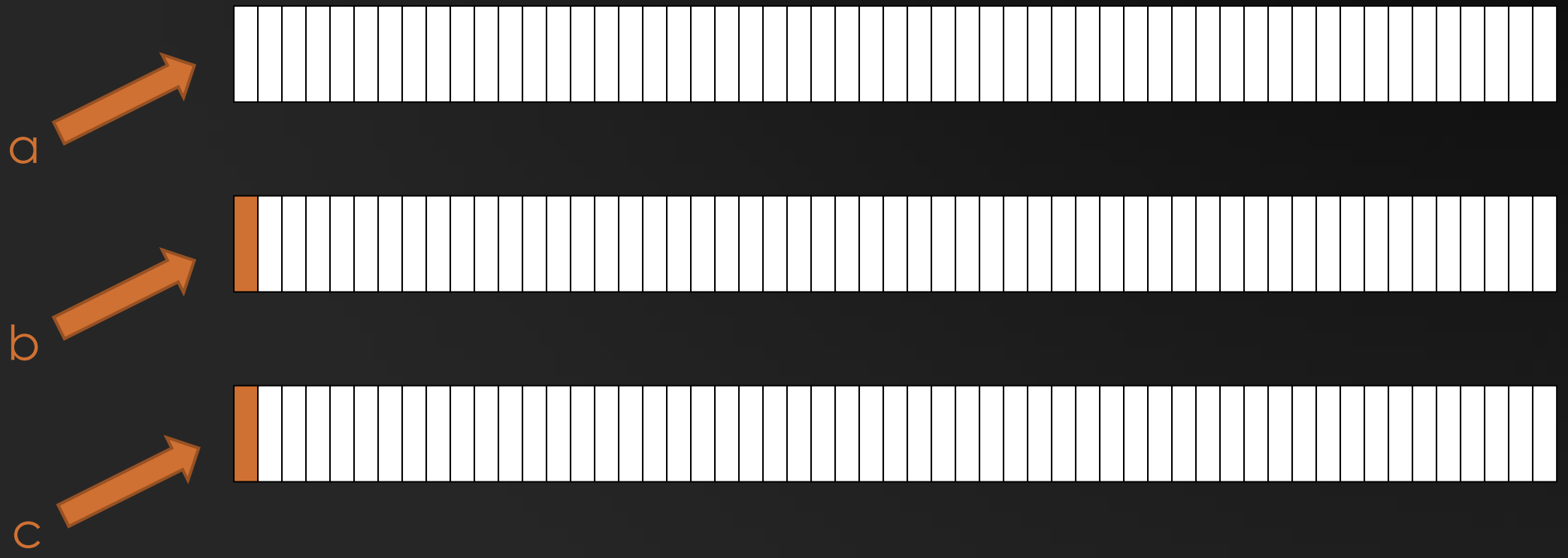
MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```



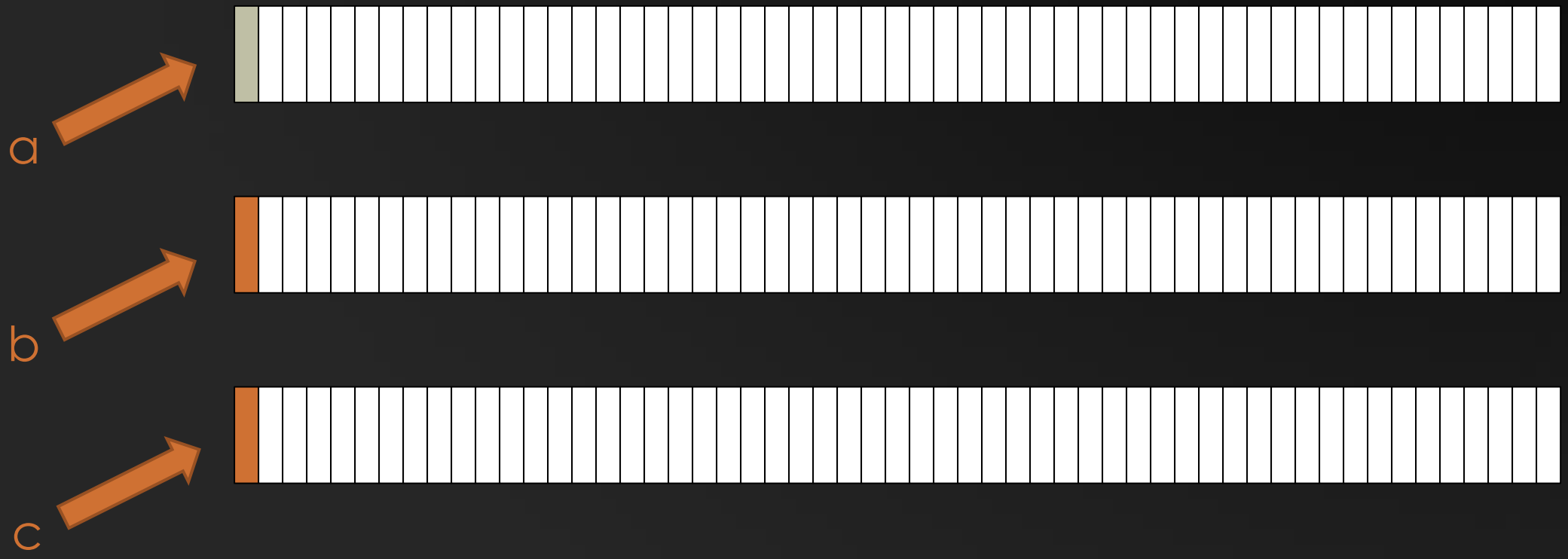
MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```



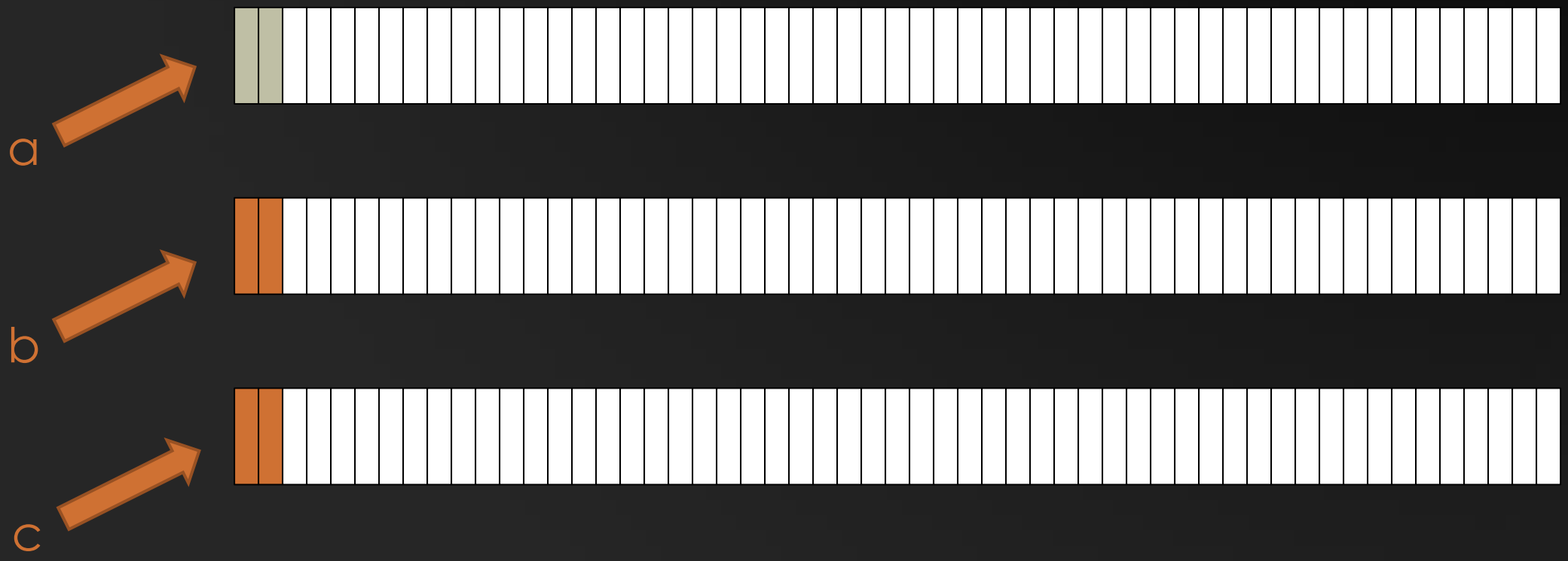
MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```



MOTIVATING EXAMPLE

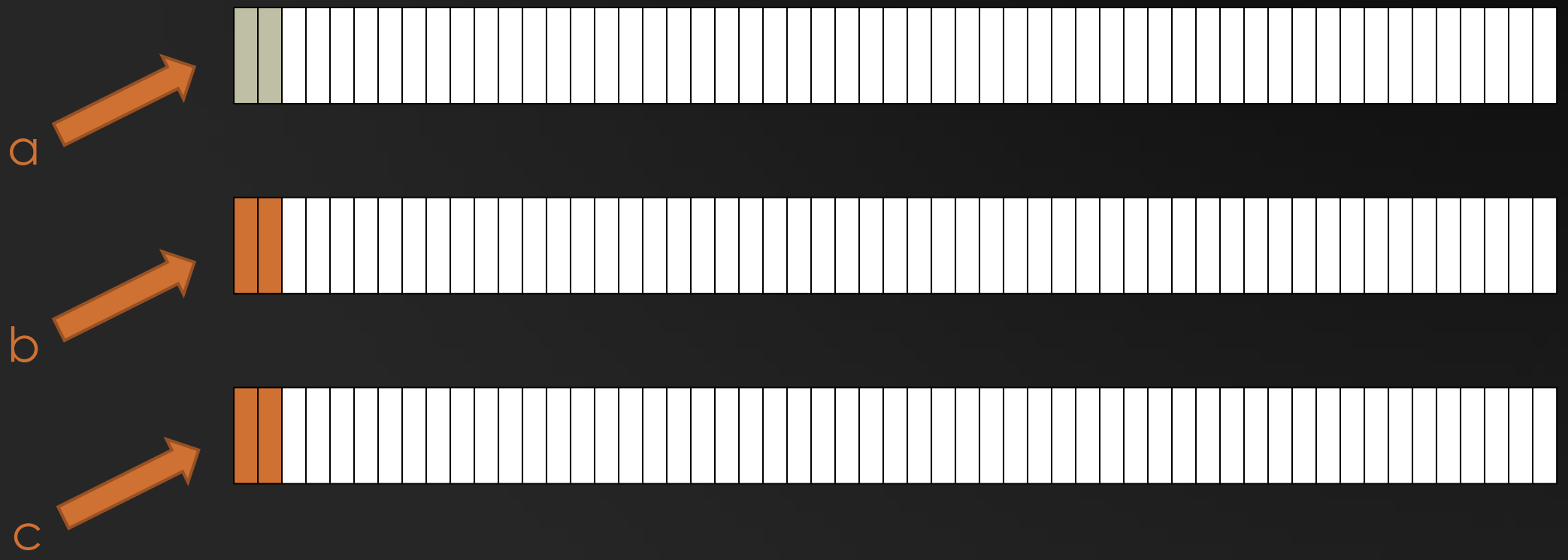
```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```



MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

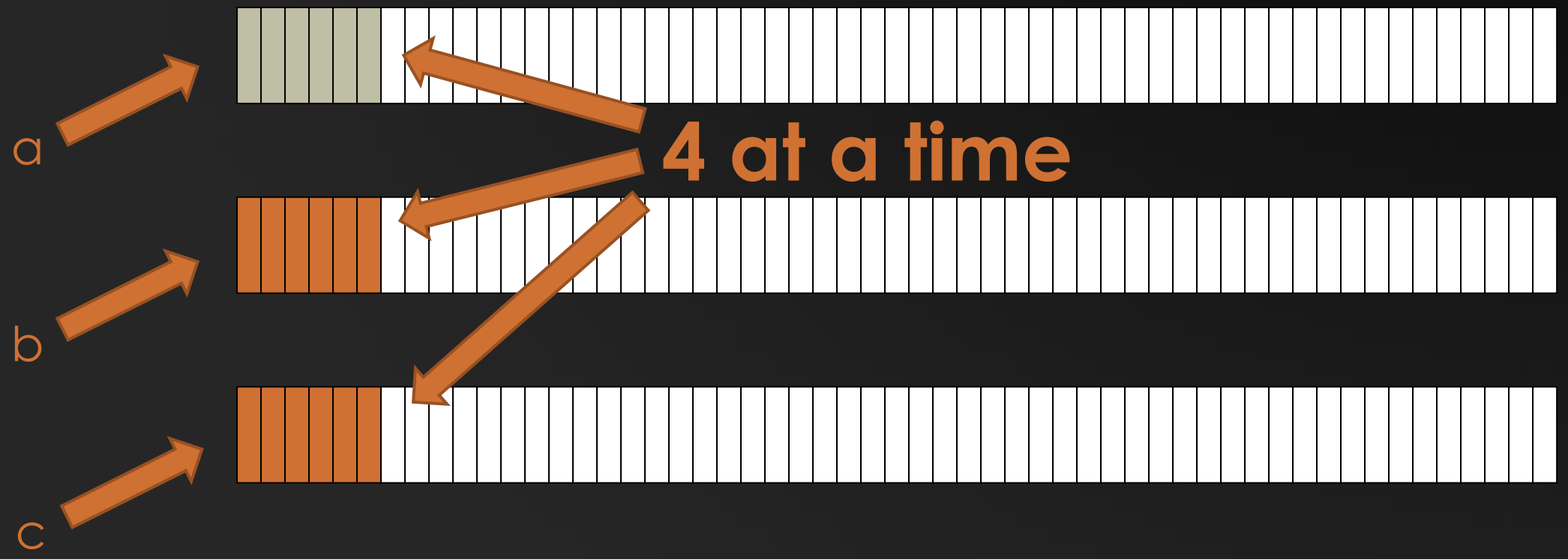
Easy to vectorize



MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

Easy to vectorize



MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

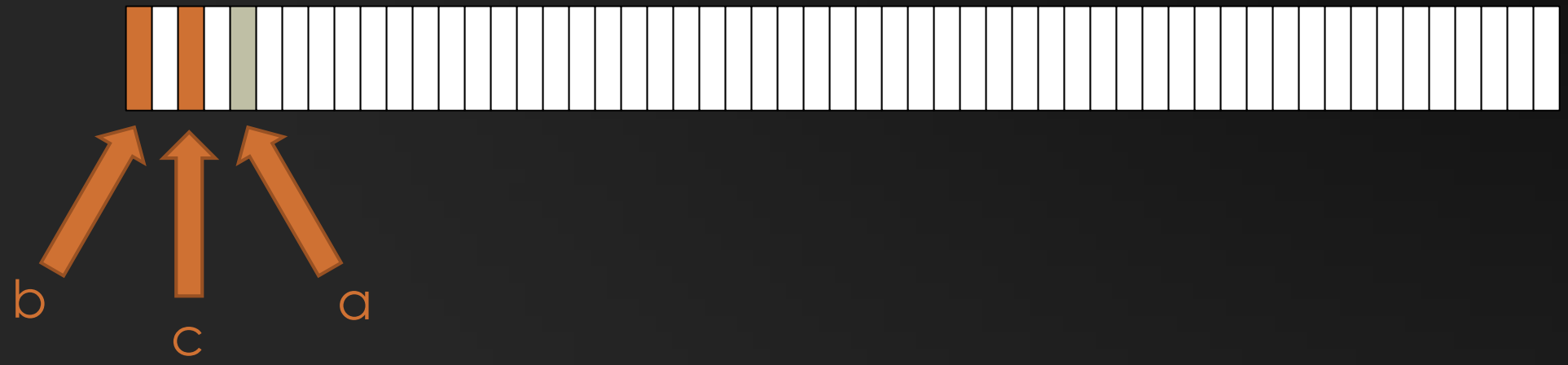
```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?



MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What if there is overlap?

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

Vectorization is not legal!

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

Vectorization is not legal!

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

Vectorization is not legal!

MOTIVATING EXAMPLE

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

Vectorization is not legal!

WRONG! We are reading c[2] without having first stored to a[0]

~~THE PRESENCE OF OVERLAP PROHIBITS VECTORIZATION~~

THE PRESENCE OF *POSSIBLE* OVERLAP PROHIBITS VECTORIZATION

THE COMPILER CAN STILL GENERATE FAST CODE

CONDITIONAL VECTORIZATION #1

Source code:

```
void mul_flt(float *a, float *b, float *c) {
   for (int i=0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

What we generate for you:

```
void mul_flt(float *a, float *b, float *c) {
   if (a overlaps b) goto scalar_loop;
   if (a overlaps c) goto scalar_loop;

   for (int i = 0; i<1000; i+=4)
      a[i:i+3] = mulps b[i:i+3], c[i:i+3];
   return;
scalar_loop:
   for (int i = 0; i<1000; i++)
      a[i] = b[i] * c[i];
}
```

Runtime overlap checks

Vector loop

Scalar duplicate

CONDITIONAL VECTORIZATION #1

```
for (int i=0; i<1000; i++)
    a[i] = b[i] * c[i];
```

- 4 INSTRS OF RUNTIME CHECK, PLUS DUPLICATE LOOP
- `mul_flt()` CODE SIZE INCREASES BY 7X

2.63X SPEEDUP

FOR REFERENCE, 2.64X SPEEDUP FOR VECT W/O RUNTIME CHECK AND THE DUPLICATE LOOP.

WHY?

CONDITIONAL VECTORIZATION #2

Loop

```
for (k = 1; k <= M; k++) {
    mc[k] = mpp[k-1]   + tpmm[k-1];
    if ((sc = ip[k-1]  + tpim[k-1]) > mc[k])  mc[k] = sc;
    if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k])  mc[k] = sc;
    if ((sc = xmb  + bp[k])         > mc[k])  mc[k] = sc;
    mc[k] += ms[k];
    if (mc[k] < -INFTY) mc[k] = -INFTY;

    dc[k] = dc[k-1] + tpdd[k-1];
    if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
    if (dc[k] < -INFTY) dc[k] = -INFTY;

    if (k < M) {
        ic[k] = mpp[k] + tpmi[k];
        if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
        ic[k] += is[k];
        if (ic[k] < -INFTY) ic[k] = -INFTY;
    }
}
```

CONDITIONAL VECTORIZATION #2

```
for (k = 1; k <= M; k++) {
    mc[k] = mpp[k-1]   + tpmm[k-1];
    if ((sc = ip[k-1]  + tpim[k-1]) > mc[k])  mc[k] = sc;
    if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k])  mc[k] = sc;
    if ((sc = xmb  + bp[k])         > mc[k])  mc[k] = sc;
    mc[k] += ms[k];
    if (mc[k] < -INFTY) mc[k] = -INFTY;

    dc[k] = dc[k-1] + tpdd[k-1];
    if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k] = sc;
    if (dc[k] < -INFTY) dc[k] = -INFTY;

    if (k < M) {
        ic[k] = mpp[k] + tpmi[k];
        if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
        ic[k] += is[k];
        if (ic[k] < -INFTY) ic[k] = -INFTY;
    }
}
```

CONDITIONAL VECTORIZATION #2

- 42 RUNTIME CHECKS NEEDED
- 84 CMP/BR INSTRUCTIONS, DUPLICATE LOOP
- LOOP CODE SIZE INCREASES BY 4X

DOESN'T THIS SUCK?

```
for (k = 1; k <= M; k++) {
    mc[k] = mpp[k-1]   + tpmm[k-1];
    if ((sc = ip[k-1]  + tpim[k-1]) > mc[k])  mc
    if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k])  mc
    if ((sc = xmb  + bp[k])         > mc[k])  mc
    mc[k] += ms[k];
    if (mc[k] < -INFTY) mc[k] = -INFTY;

    dc[k] = dc[k-1] + tpdd[k-1];
    if ((sc = mc[k-1] + tpmd[k-1]) > dc[k]) dc[k
    if (dc[k] < -INFTY) dc[k] = -INFTY;

    if (k < M) {
        ic[k] = mpp[k] + tpmi[k];
        if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k
        ic[k] += is[k];
        if (ic[k] < -INFTY) ic[k] = -INFTY;
    }
}
```

2X LOOP SPEEDUP

30% OVERALL BENCHMARK SPEEDUP

FOR REFERENCE, 2.1X SPEEDUP FOR VECT W/O RUNTIME CHECK

AGENDA

CPU HARDWARE LANDSCAPE

VECTORIZING FOR MODERN CPUS

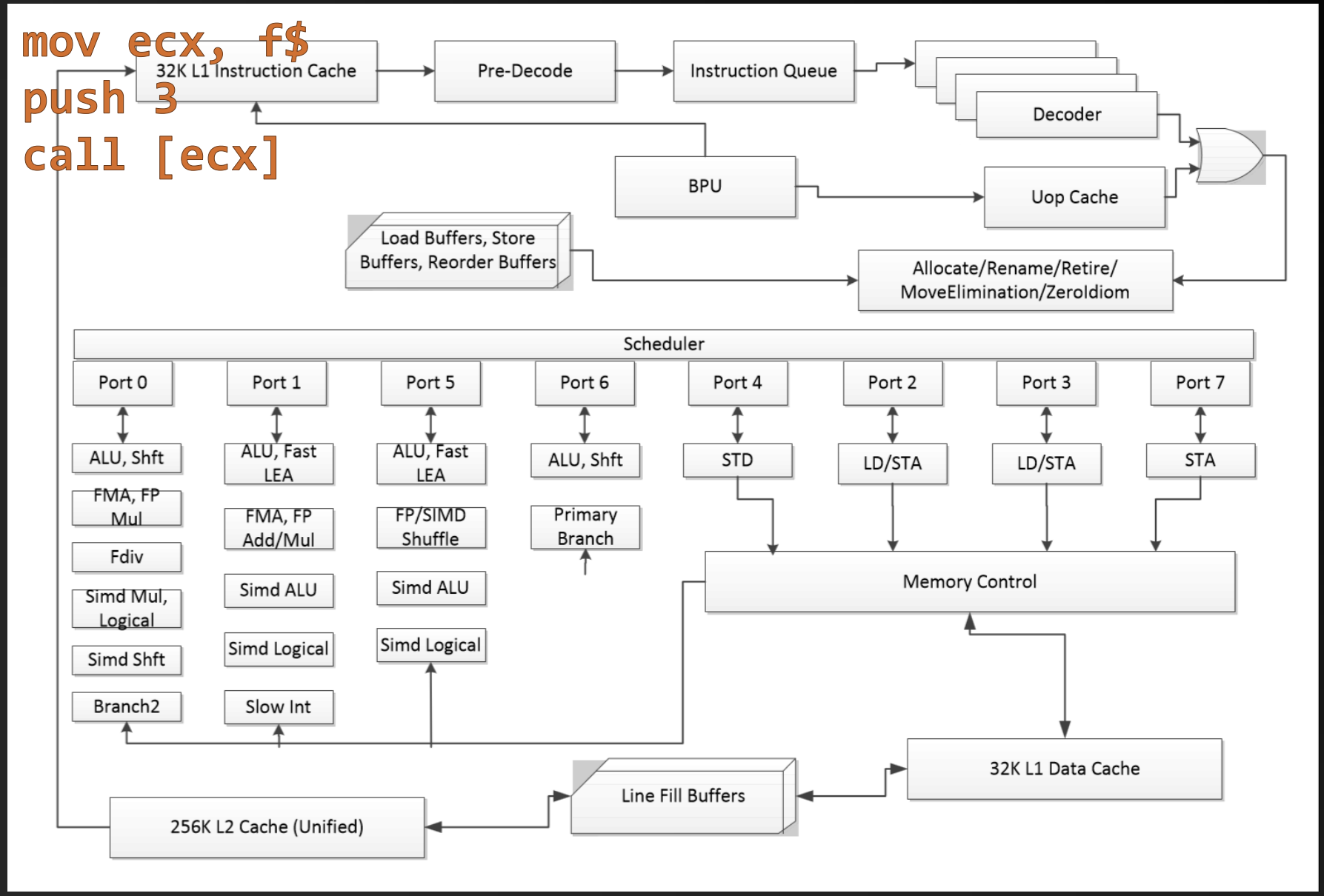
INDIRECT CALL OPTIMIZATIONS

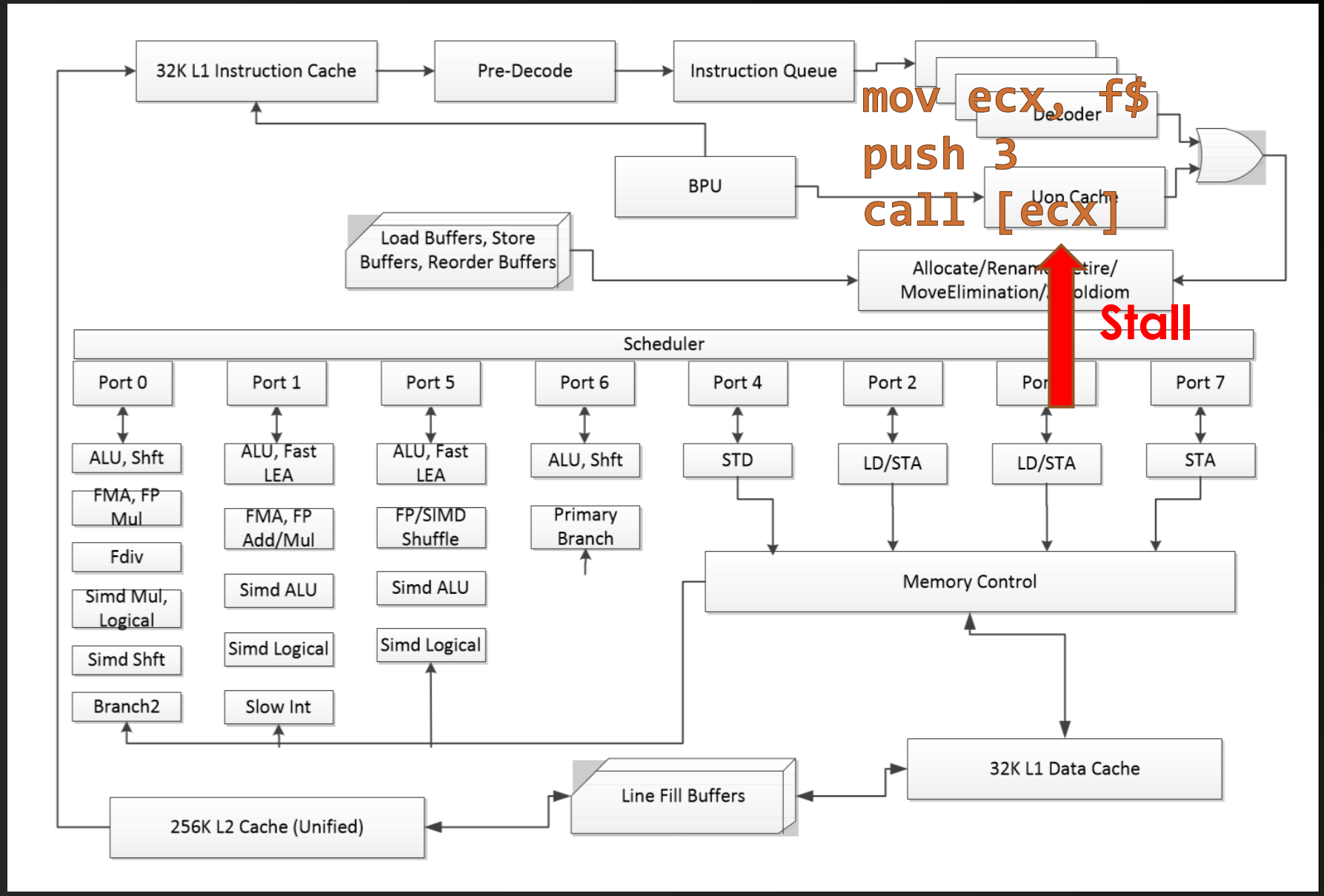
```
typedef int (PFUNC)(int);

int func1(int x) {
    return x + 100;
}

int func2(int x) {
    return x + 200;
}

int test(PFUNC f) {
    return f(3);
}
```

```
mov ecx, f$
push 3
call [ecx]
```

This sucks

```
mov ecx, f$
push 3
call [ecx]
```




32K L1 Instruction Cache
Pre-Decode
Instruction Queue
mov ecx, f$
push 3
call [ecx]
Decoder
BPU
Uop Cache
Load Buffers, Store Buffers, Reorder Buffers
Allocate/Rename/Retire/ MoveElimination/Zeroidiom
Stall
Scheduler
Port 0
Port 1
Port 5
Port 6
Port 4
Port 2
Port 7
ALU, Shft
FMA, FP Mul
Fdiv
Simd Mul, Logical
Simd Shft
Branch2
ALU, Fast LEA
FMA, FP Add/Mul
Simd ALU
Simd Logical
Slow Int
ALU, Fast LEA
FP/SIMD Shuffle
Simd ALU
Simd Logical
ALU, Shft
Primary Branch
STD
LD/STA
LD/STA
STA
Memory Control
32K L1 Data Cache
Line Fill Buffers
256K L2 Cache (Unified)

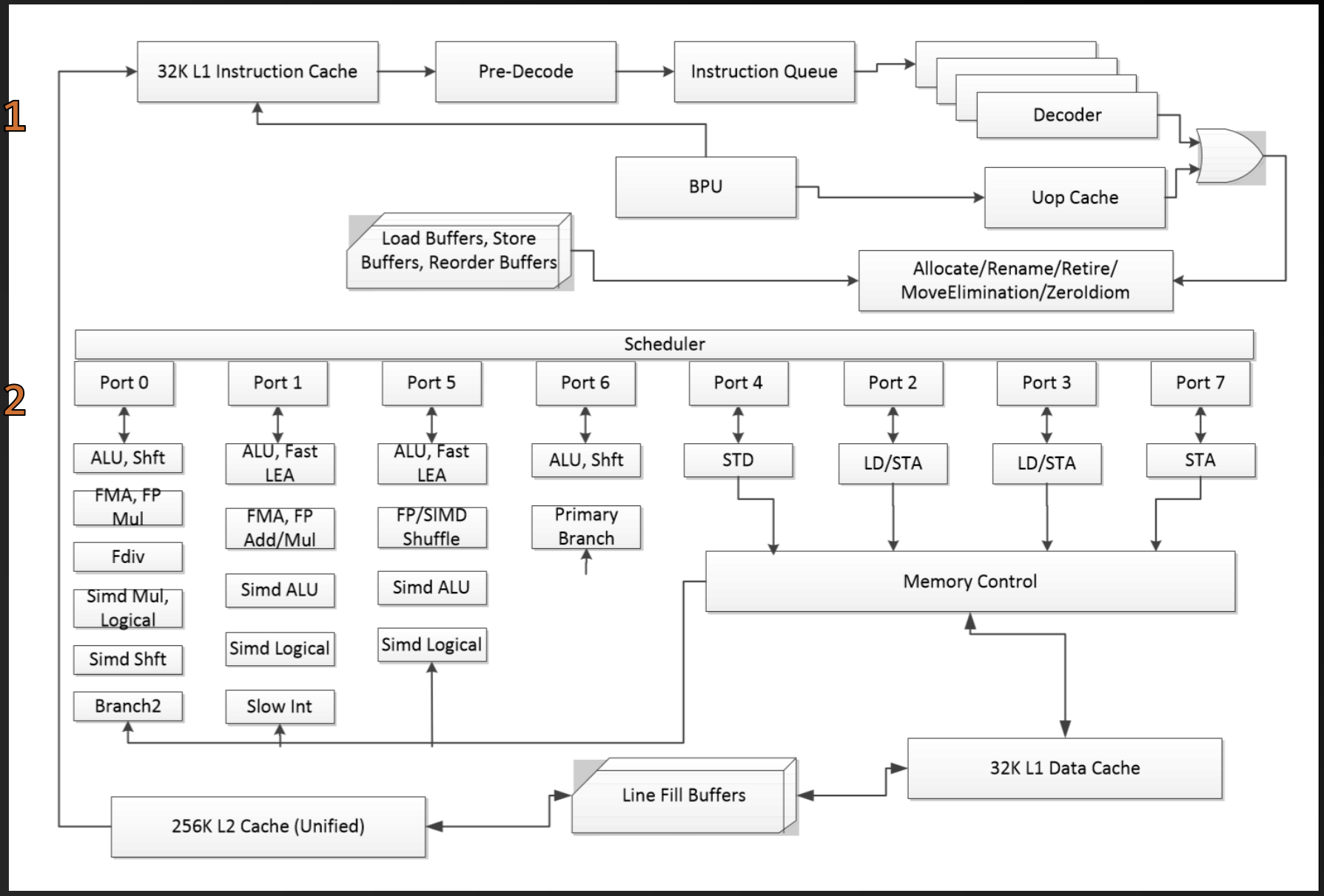
```
typedef int (PFUNC)(int);

int func1(int x) {
    return x + 100;
}

int func2(int x) {
    return x + 200;
}

int test(PFUNC f) {
    return f(3);
}
```

```
int test(PFUNC f) {
    if (f == func1) return func1(3);
    if (f == func2) return func2(3);
    return f(3);
}
```

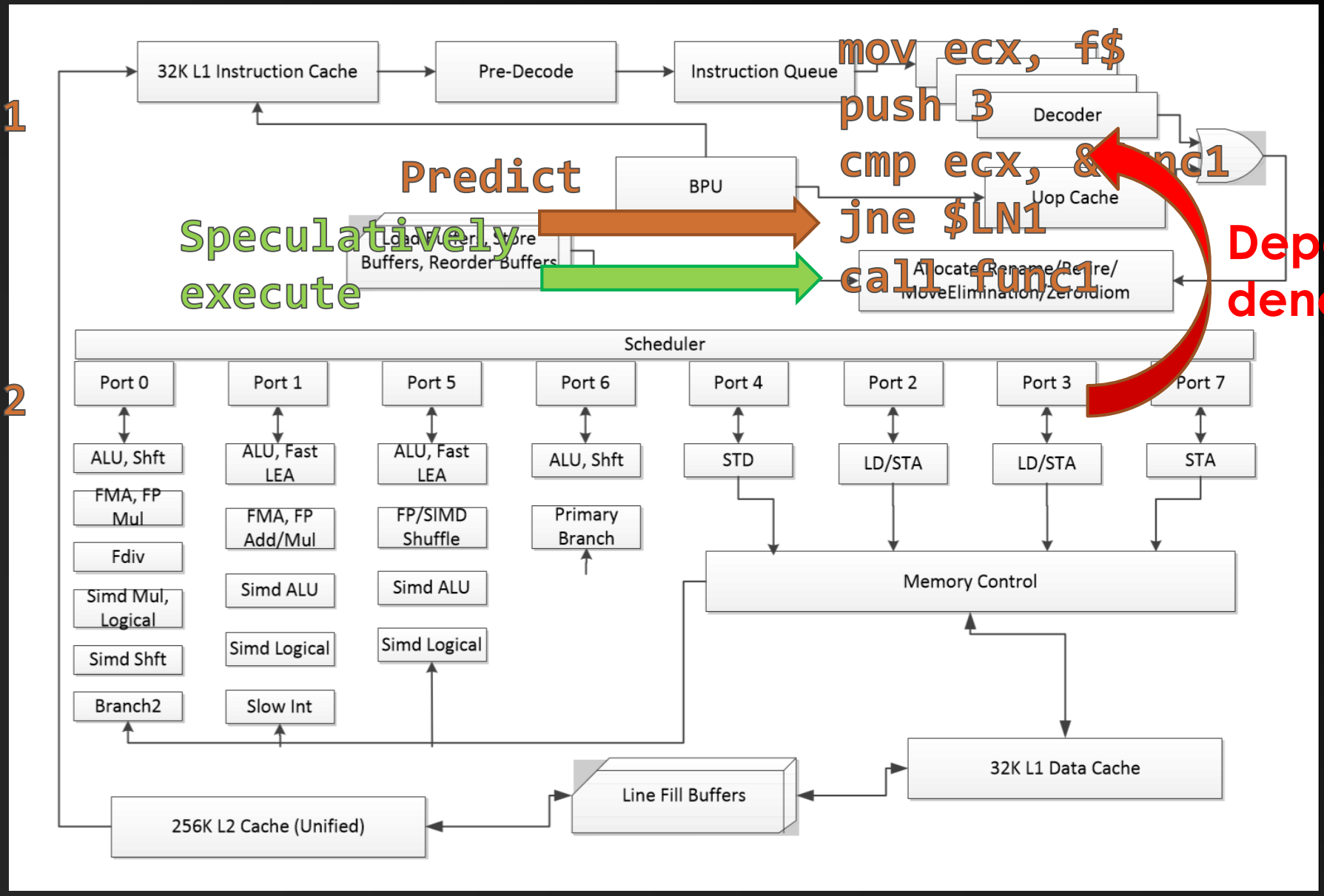
```
  mov ecx, f$
  push 3
  cmp ecx, &func1
  jne $LN1
  call func1
  ret
$LN1:
  cmp ecx, &func2
  jne $LN2
  call func2
  ret
$LN2:
  call [ecx]
```

Leverage branch predictor

```
  mov ecx, f$
  push 3
  cmp ecx, &func1
  jne $LN1
  call func1
  ret
$LN1:
  cmp ecx, &func2
  jne $LN2
  call func2
  ret
$LN2:
  call [ecx]
```



```
  mov ecx, f$
  push 3
  cmp ecx, &func1
  jne $LN1
  call func1
  ret
$LN1:
  cmp ecx, &func2
  jne $LN2
  call func2
  ret
$LN2:
  call [ecx]
```



```
int test(PFUNC f) {
    return f(3);
}
```

```
mov ecx, f$
push 3
call [ecx]
```

Stall

```
int test(PFUNC f) {
    if (f == func1) return func1(3);
    if (f == func2) return func2(3);
    return f(3);
}
```

```
  mov ecx, f$
  push 3
  cmp ecx, &func1
  jne $LN1
  call func1
  ret
$LN1:
  cmp ecx, &func2
  jne $LN2
  call func2
  ret
$LN2:
  call [ecx]
```

Not a stall

Speedup due to if-statements + branch prediction

You could add if-statements by hand…

But with profile counts, the compiler does it for you.

Source code:

```
typedef int (PFUNC)(int);

int func1(int x) {
    return x + 100;
}

int func2(int x) {
    return x + 200;
}

int test(PFUNC f) {
    return f(3);
}
```

If counts say `test()` calls `func1()` as often as `func2()`:

- Compiler inserts two if-checks
- `test()` code size increases 5.4x
- **10%** performance win

```
if (f == func1)
  return func1(3);
if (f == func2)
  return func2(3);
return f(3);
```

If counts say `test()` calls `func1()` way more than `func2()`:

- Compiler inserts one if-check
- `test()` code size increases 3.4x
- **15%** performance win

```
if (f == func1)
  return func1(3);
return f(3);
```

If counts say `test()` calls `func1()` way more than `func2()`, and we decide to inline `func1()`:

- Compiler inserts one if-check
- `test()` code size increases 2.7x
- **30%** performance win

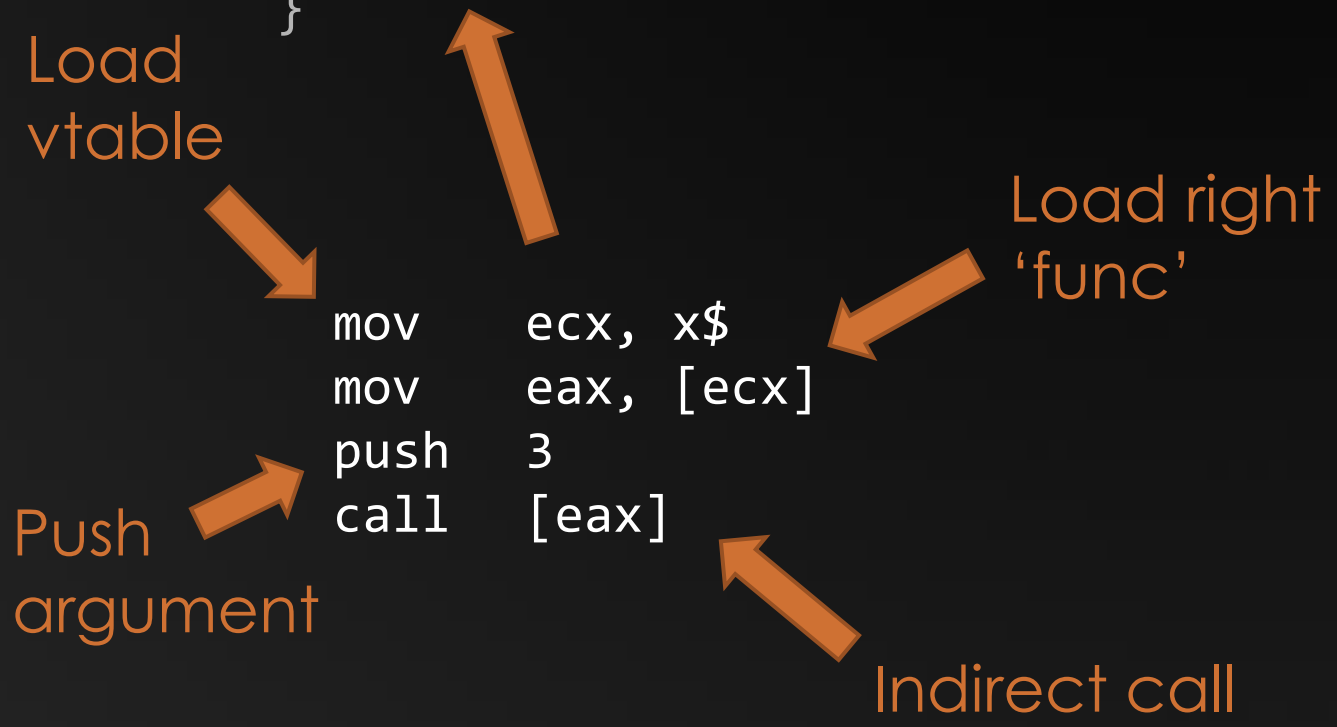
```
if (f == func1)
  return 103;
return f(3);
```

All compiler driven – no code changes!

That's nice, but I don't use function pointers

```
class Base {
public:
   virtual int func(int x) = 0;
};

class A : public Base {
   int func(int x) { return x + 100; };
};

class B : public Base {
   int func(int x) { return x + 200; };
};

class C : public Base {
   int func(int x) { return x + 300; };
};
```

```
int test(Base *x) {
    return x->foo(3);
}
```



Compiler-driven speculative devirtualization & inlining

RECAP & OTHER RESOURCES

- Compiler has to take advantage of silicon
- Guard optimizations with runtime checking
  - `/Qvec-report:2` messages (15xx codes ~ runtime checks)
- Profile counts: profile guided optimizations
- Profiling tools
  - Visual Studio Performance Analysis
  - Intel VTune Amplifier XE
  - AMD CodeXL

Compiler switches

http://msdn.microsoft.com

Automatic vectorization blog & cookbook

http://blogs.msdn.com/b/nativeconcurrency

Visual C++ blog

http://blogs.msdn.com/b/vcblog/

Channel 9 Going Native

http://channel9.msdn.com/Shows/C9-GoingNative

Q&A

BACKUP SLIDES

WILD AND CRAZY RUNTIME CHECKS

```
for (int i=0; i<1000; i++)
    a[i] = b[i] * 2.0f;
```

```
Range of a: &a[0] to &a[999]
Range of b: &b[0] to &b[999]
```

WILD AND CRAZY RUNTIME CHECKS

```
for (int i=0; i<1000; i++)
    a[i] = b[i+1] * 2.0f;
```

```
Range of a: &a[0] to &a[999]
Range of b: &b[1] to &b[1000]
```

WILD AND CRAZY RUNTIME CHECKS

```
for (int i=0; i<1000; i++)
    a[i] = b[i+1] + b[i+5];
```

```
Range of a: &a[0] to &a[999]
Range of b: &b[1] to &b[1004]
```

Messup in the presentation slides. B ends at b[1004].

Another reason why the compiler should do this for you!

WILD AND CRAZY RUNTIME CHECKS

```
for (int i=0; i<1000; i++)
    a[i] = b[i+1] + b[i+x];
```

```
Range of a: &a[0] to &a[999]
Range of b: &b[?] to &b[?]
```

WILD AND CRAZY RUNTIME CHECKS

```
for (int i=lb; i<ub; i++)
    a[i] = b[i*i];
```

```
Range of a: &a[lb] to &a[ub]
Range of b: &b[?] to &b[?]
```