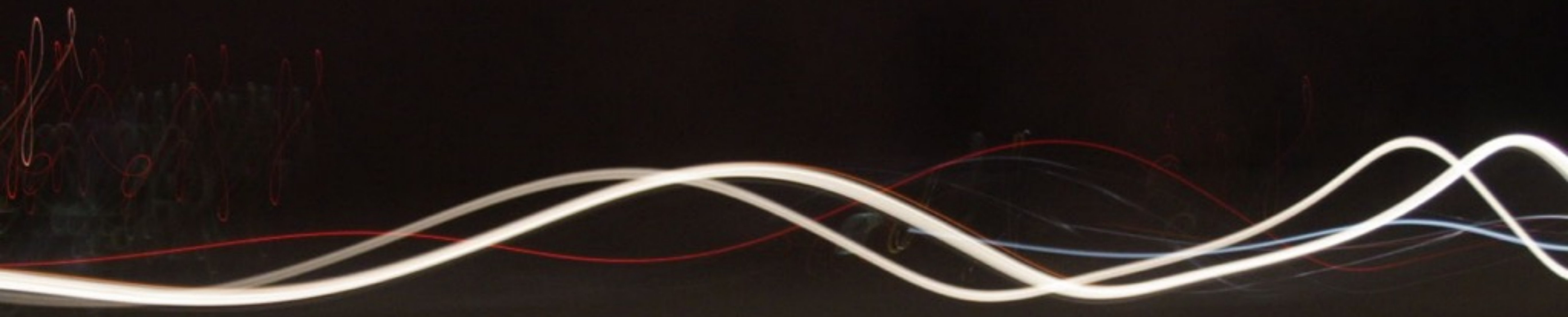
C++ in the Audio Industry, Episode III

Align, Vectorise, Cache, Jump!



Timur Doumler

C++Now 2016, 13 May 2016

Previous Episodes

- C++ in the Audio Industry

  CppCon 2015

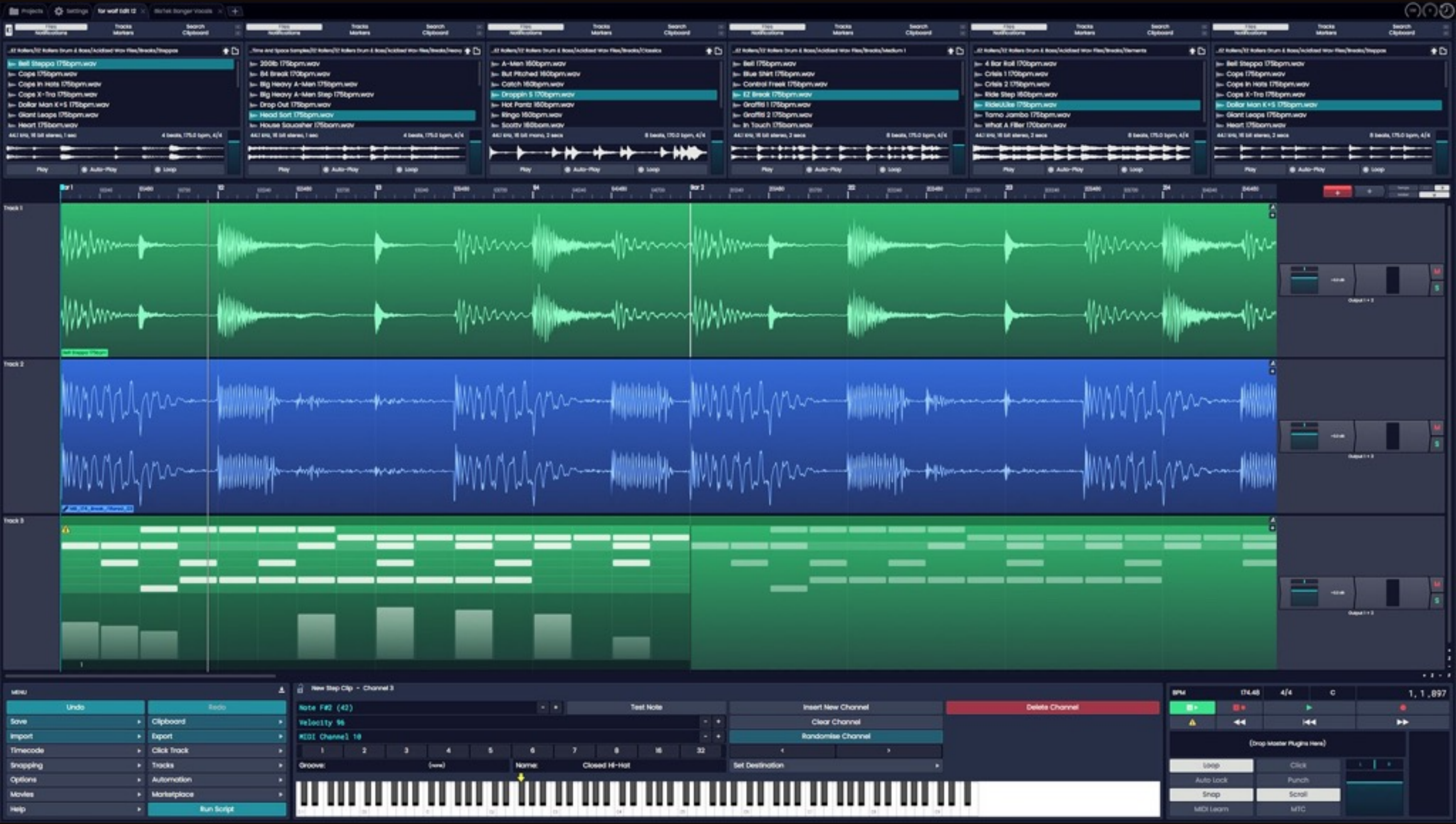
- C++ in the Audio Industry, Episode II: Floating atomics

  JUCE Summit 2015

Background

Visual Studio 2008-15
Xcode (OSX + iOS)
make (Linux)
Android Studio
gradle
ant
Live-coding

Graphics | GUI Widgets | Network | ... | Audio | Build system



New Step Clip - Channel 3

Note F#2 (42)

Velocity 96

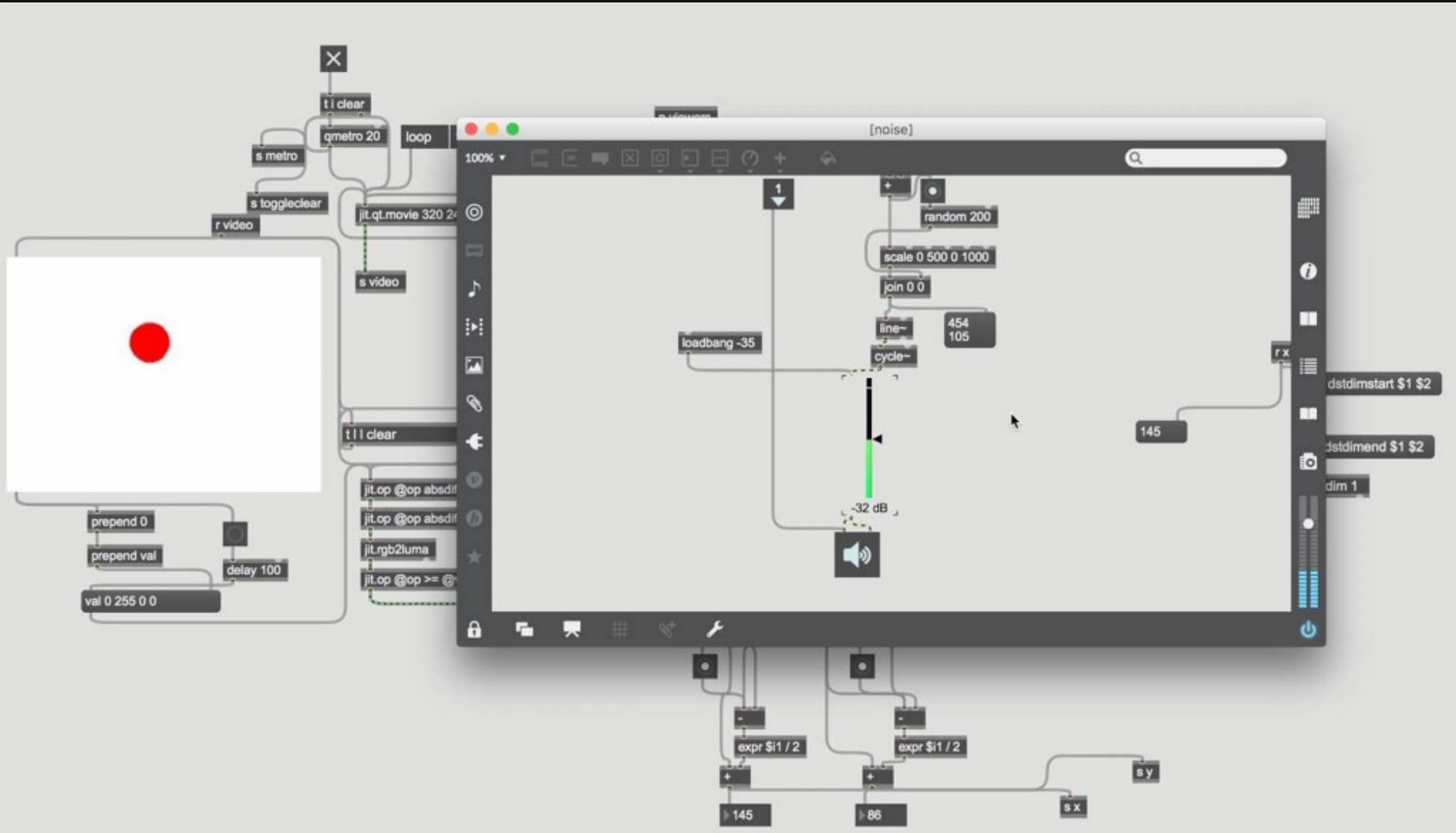
MIDI Channel 10

Groove: (none)

Name: Closed Hi-Hat

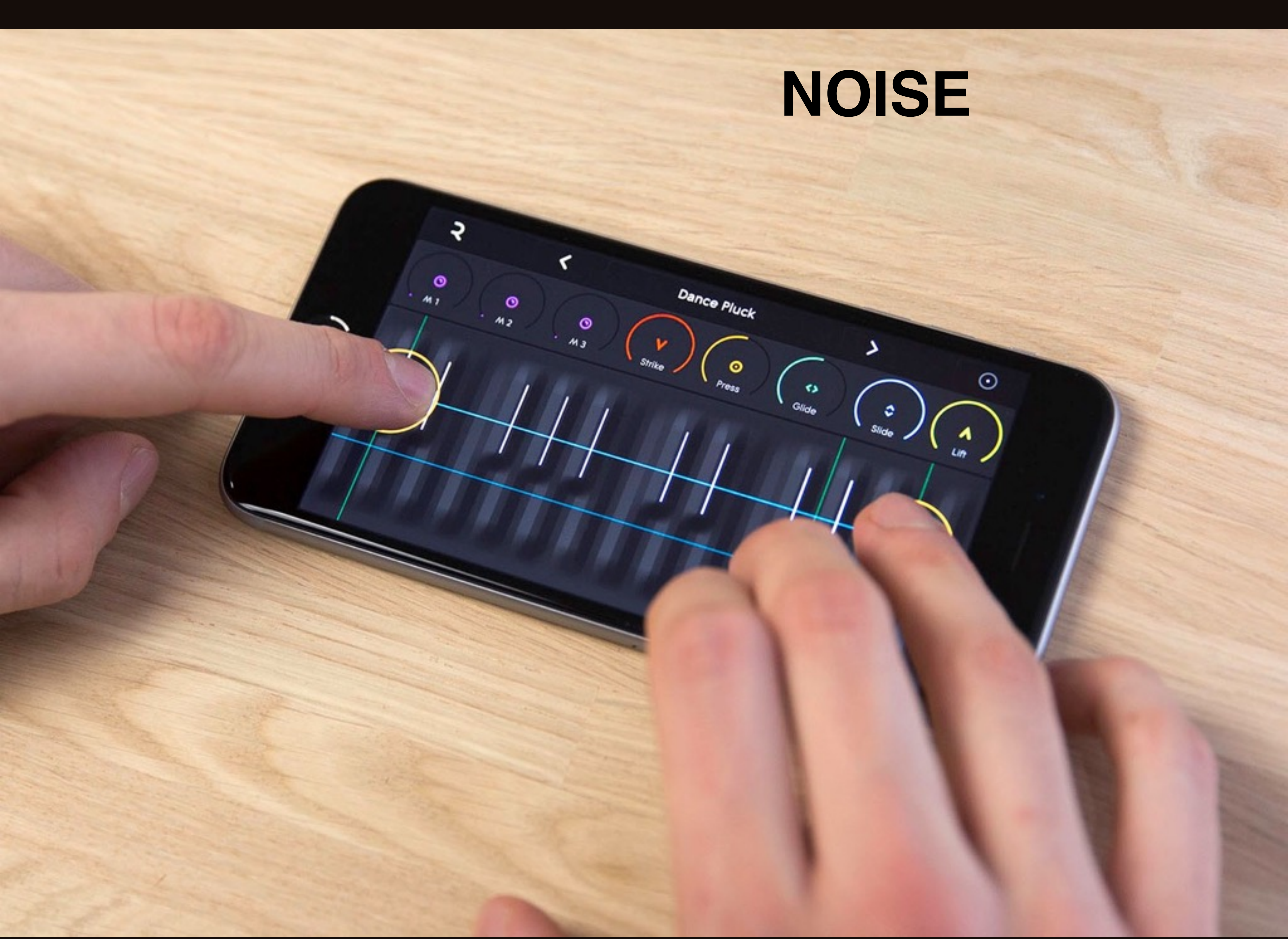
BPM 174.48 4/4 C 1, 1 ,897

Tracktion


t i clear
qmetro 20
loop
s metro
s toggleclear
r video
jit.qt.movie 320 24
s video
t l l clear
jit.op @op absdif
jit.op @op absdif
jit.rgb2luma
jit.op @op >= @
prepend 0
prepend val
delay 100
val 0 255 0 0
[noise]
100%
1
+
random 200
scale 0 500 0 1000
join 0 0
454
105
line~
loadbang -35
cycle~
-32 dB
r x
145
dstdimstart $1 $2
dstdimend $1 $2
dim 1
-
expr $i1 / 2
-
expr $i1 / 2
+
+
s y
s x
145
86

Max

NOISE



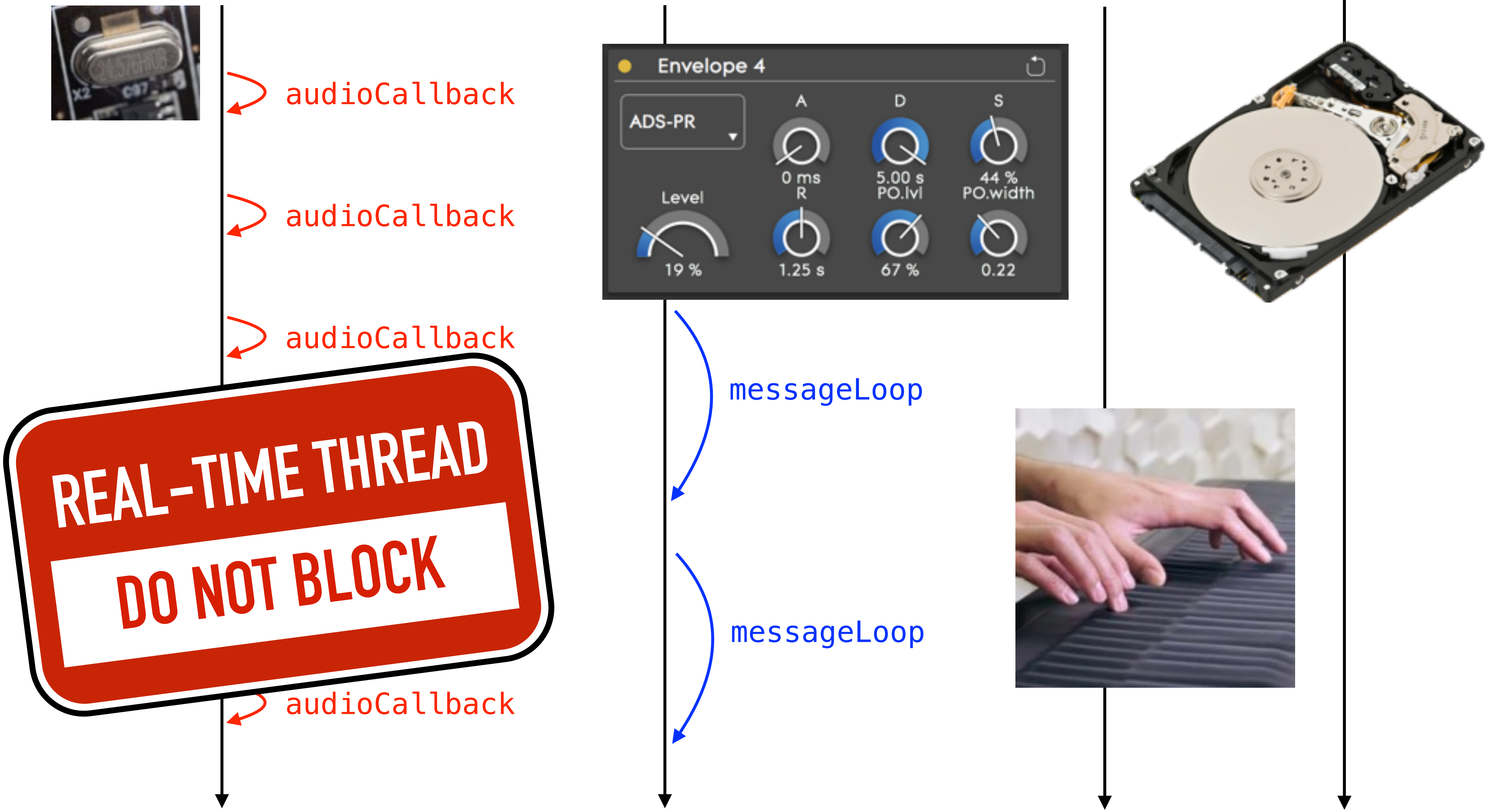
Safe & lock-free thread synchronisation.

audio thread

GUI thread

thread getting
input from keyboard

thread reading
data from disk



audioCallback

audioCallback

audioCallback

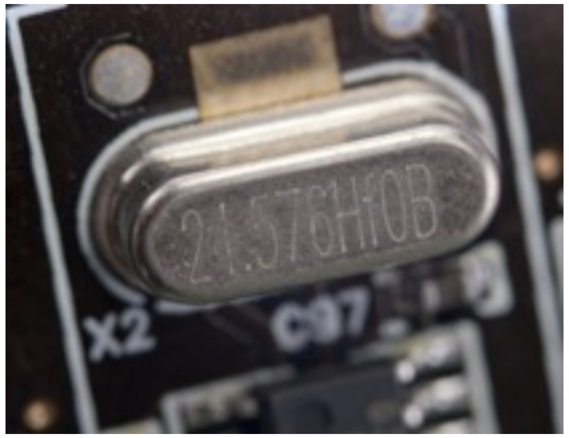
audioCallback

REAL-TIME THREAD
DO NOT BLOCK

messageLoop

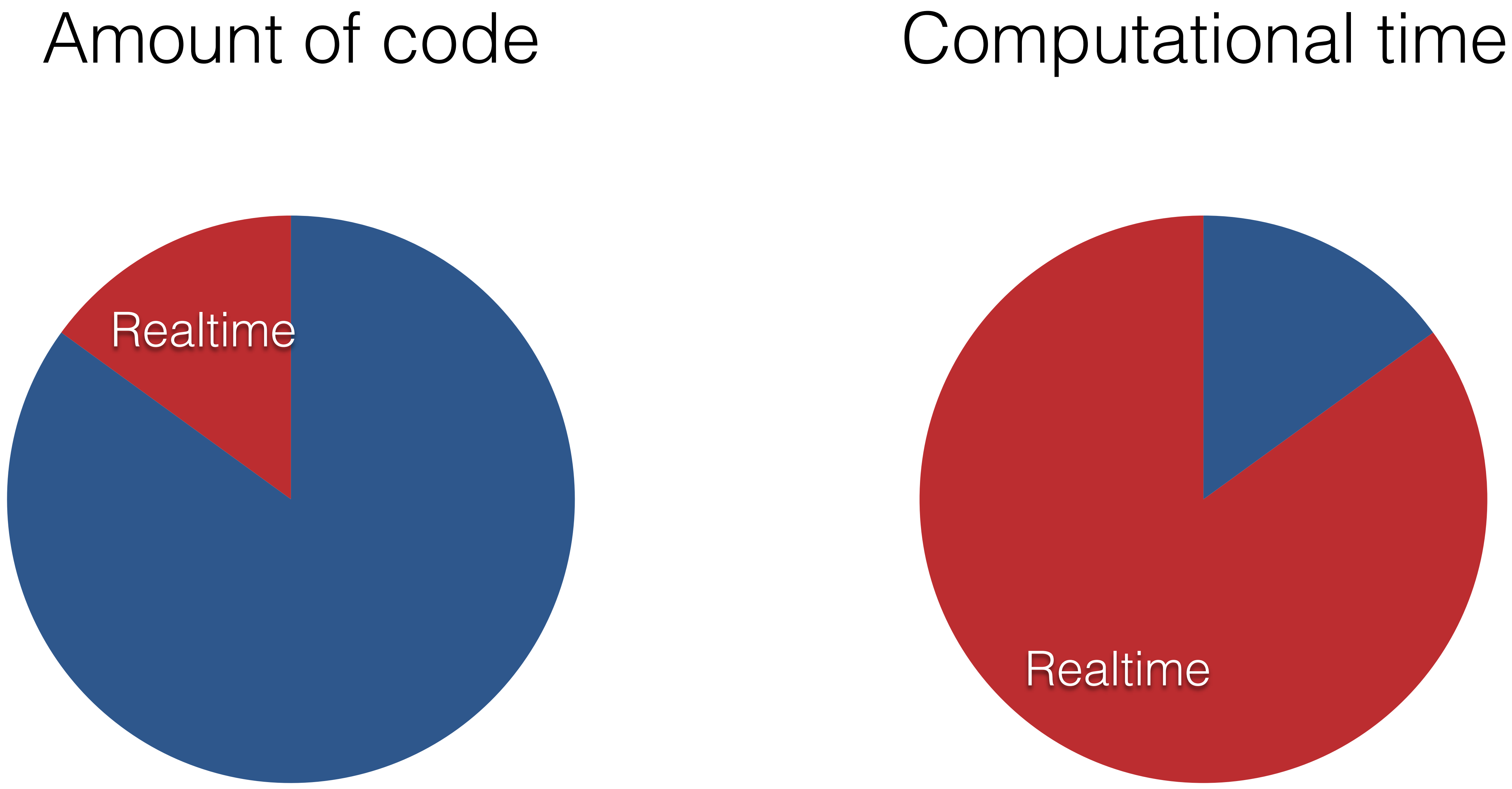
messageLoop

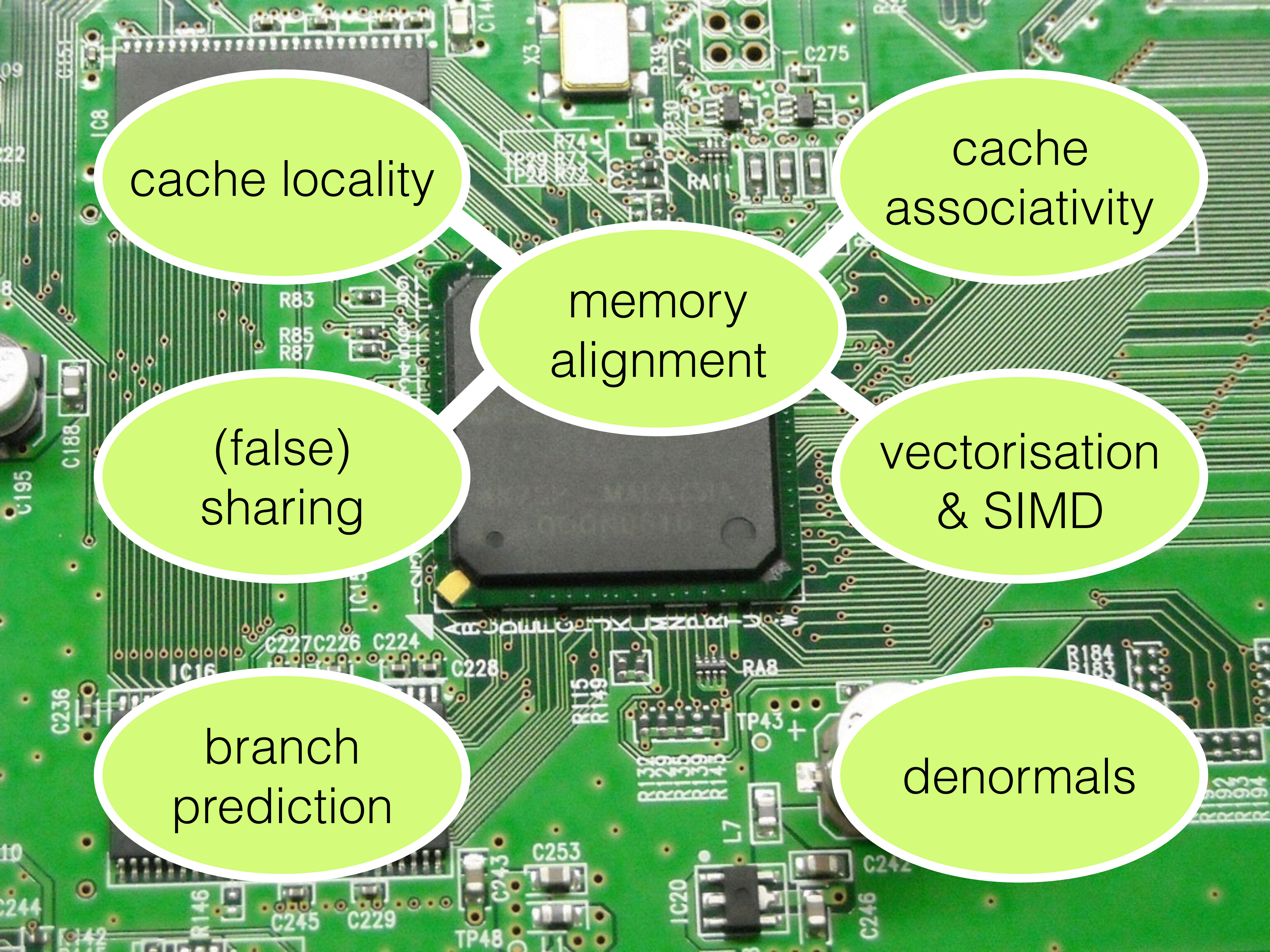
Realtime audio callback



```
void audioCallback (float** channelData,
                    int numChannels,
                    int numSamples)
{
    for (int channel = 0; channel < numChannels; ++channel)
        for (int sample = 0; sample < numSamples; ++sample)
            channelData[channel][sample] = ...
}
```

this should be as
fast as possible


Amount of code
Realtime
Computational time
Realtime



memory alignment

cache locality

cache associativity

(false) sharing

vectorisation & SIMD

branch prediction

denormals

0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
MPH
20
40
60
80
100
120
140
160
180
200
220
240
260
km/h
134.7
5887

Devices used for benchmarks

i7

Core2Duo

iPhone6S

AppleA9

Nexus5X

Snapdragon 808

Compilers used for benchmarks

clang (Xcode 7.3 / Android Studio 2.1)

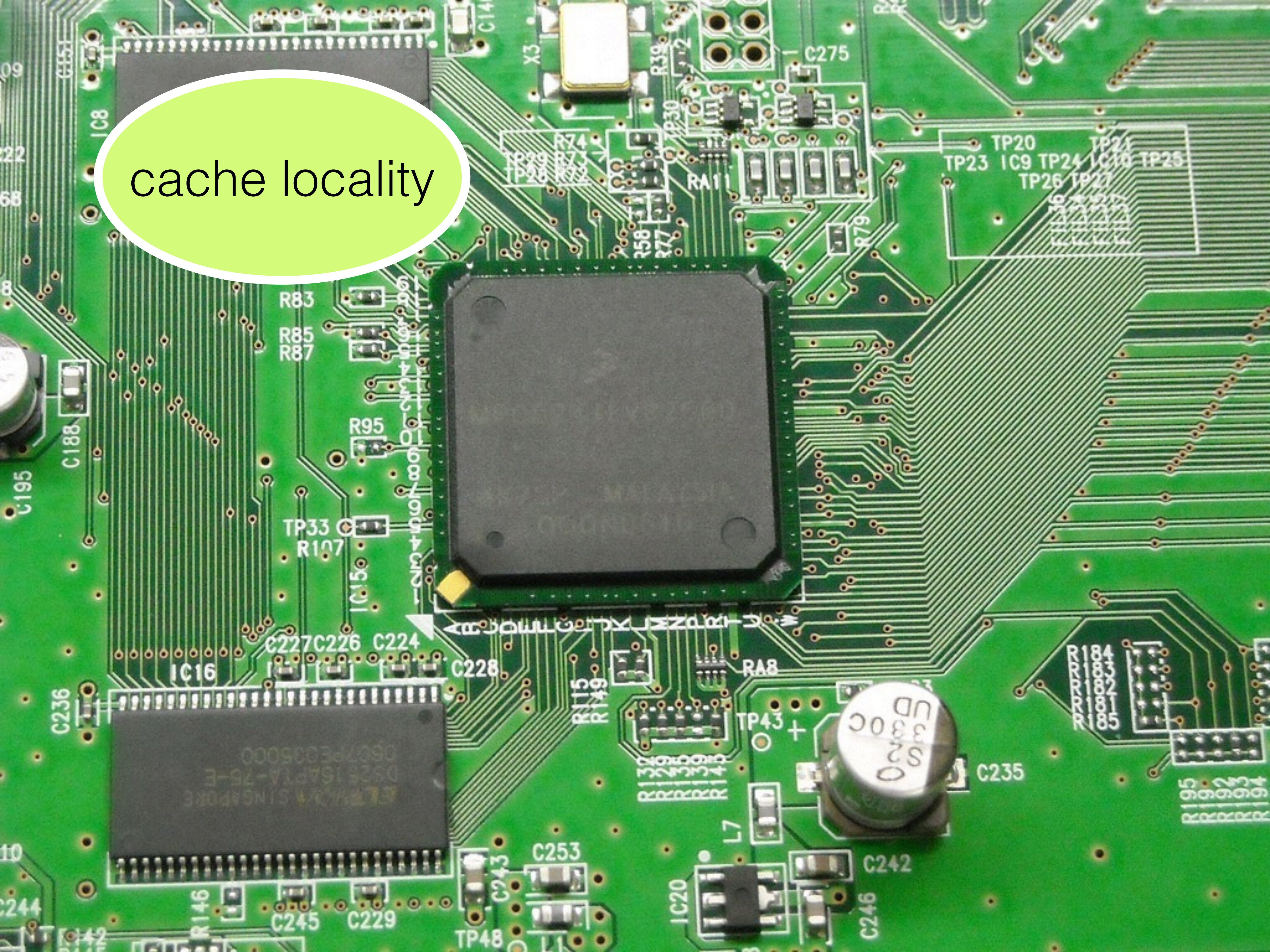
```
clang++ -std=c++11 -stdlib=libc++ -O3
```

GCC 5.3.0 (Homebrew)

```
g++-5 -std=c++11 -O3
```

Visual Studio 2015

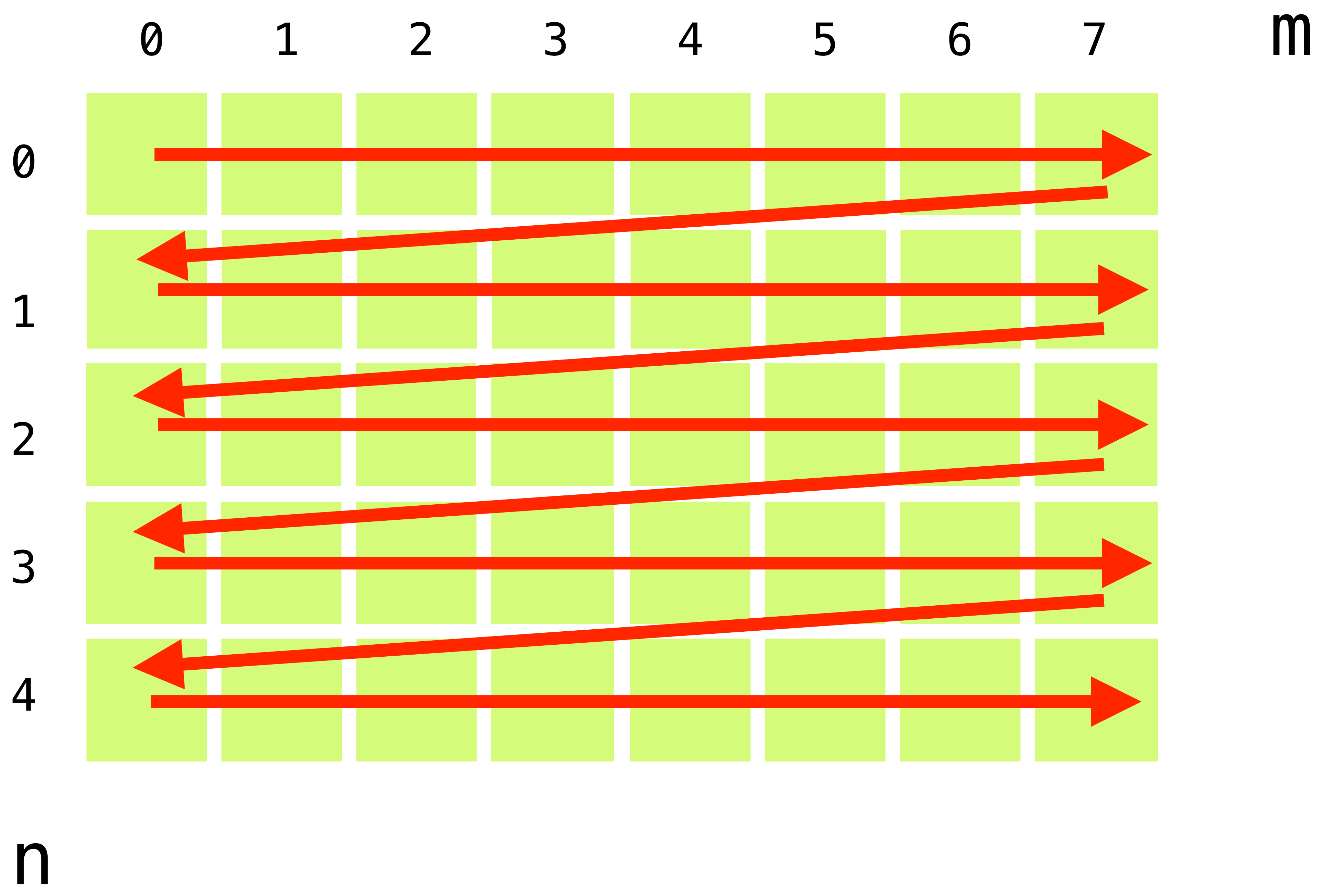
```
cl.exe /Ox
```



cache locality

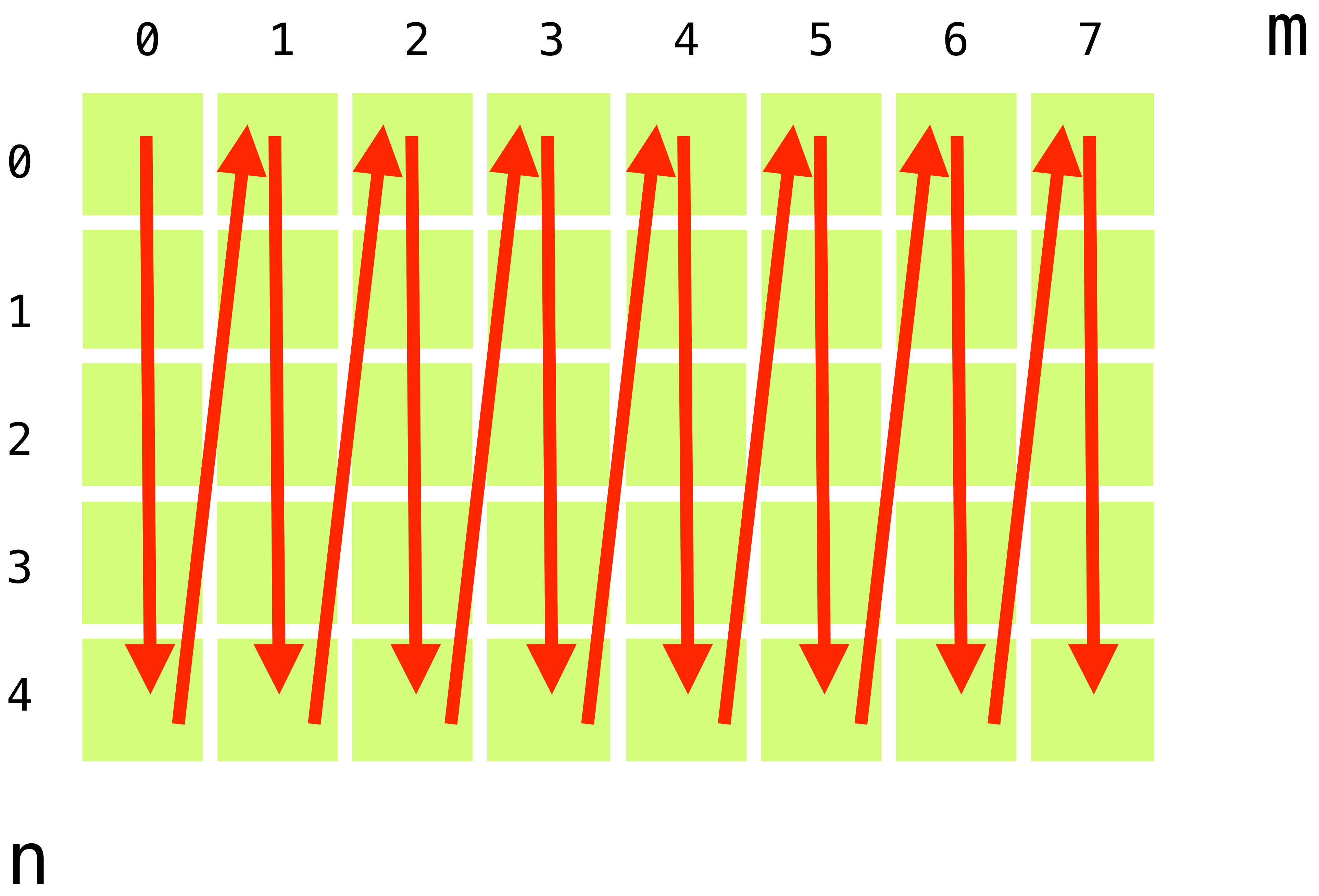
2D Array traversal

```
int array[n][m];
```

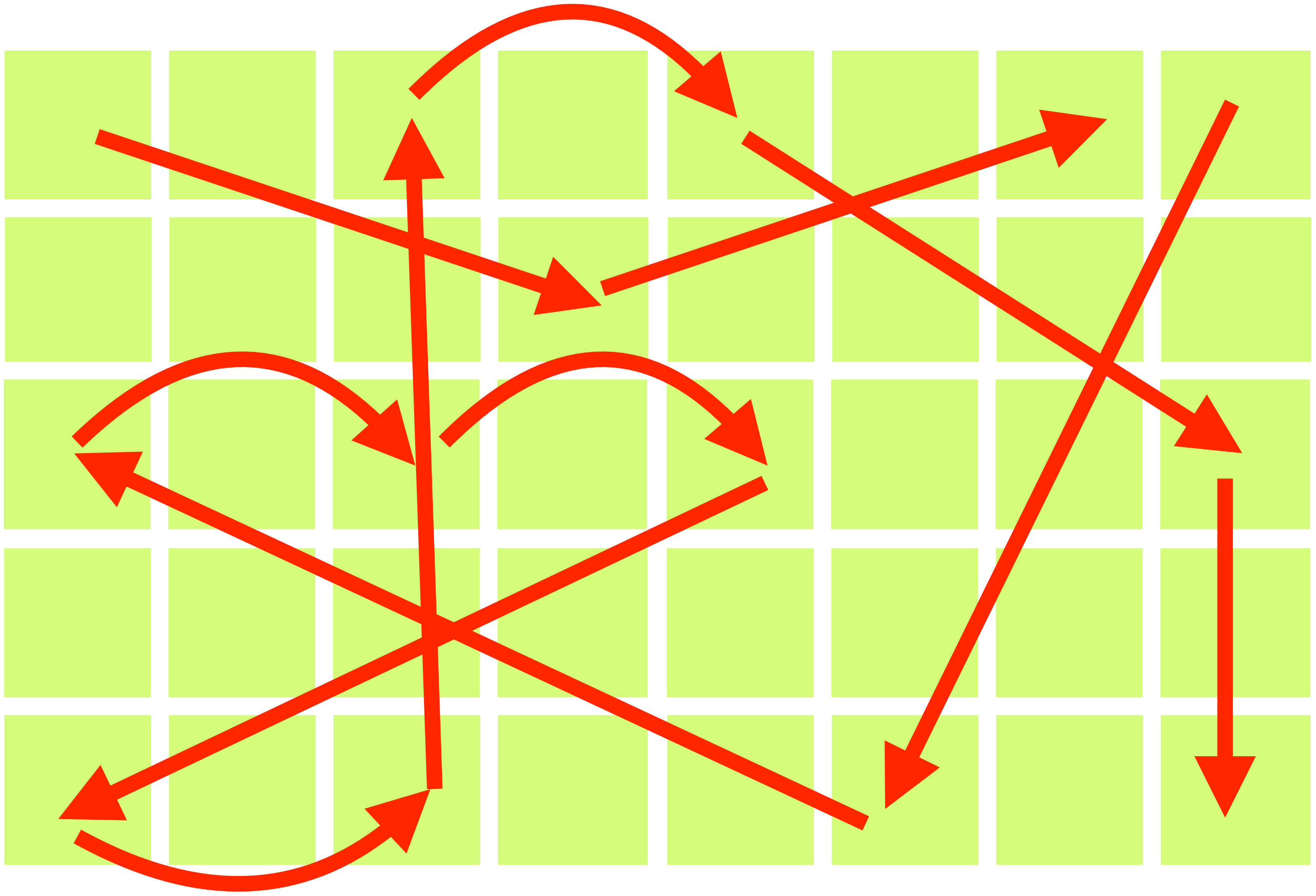


2D Array traversal

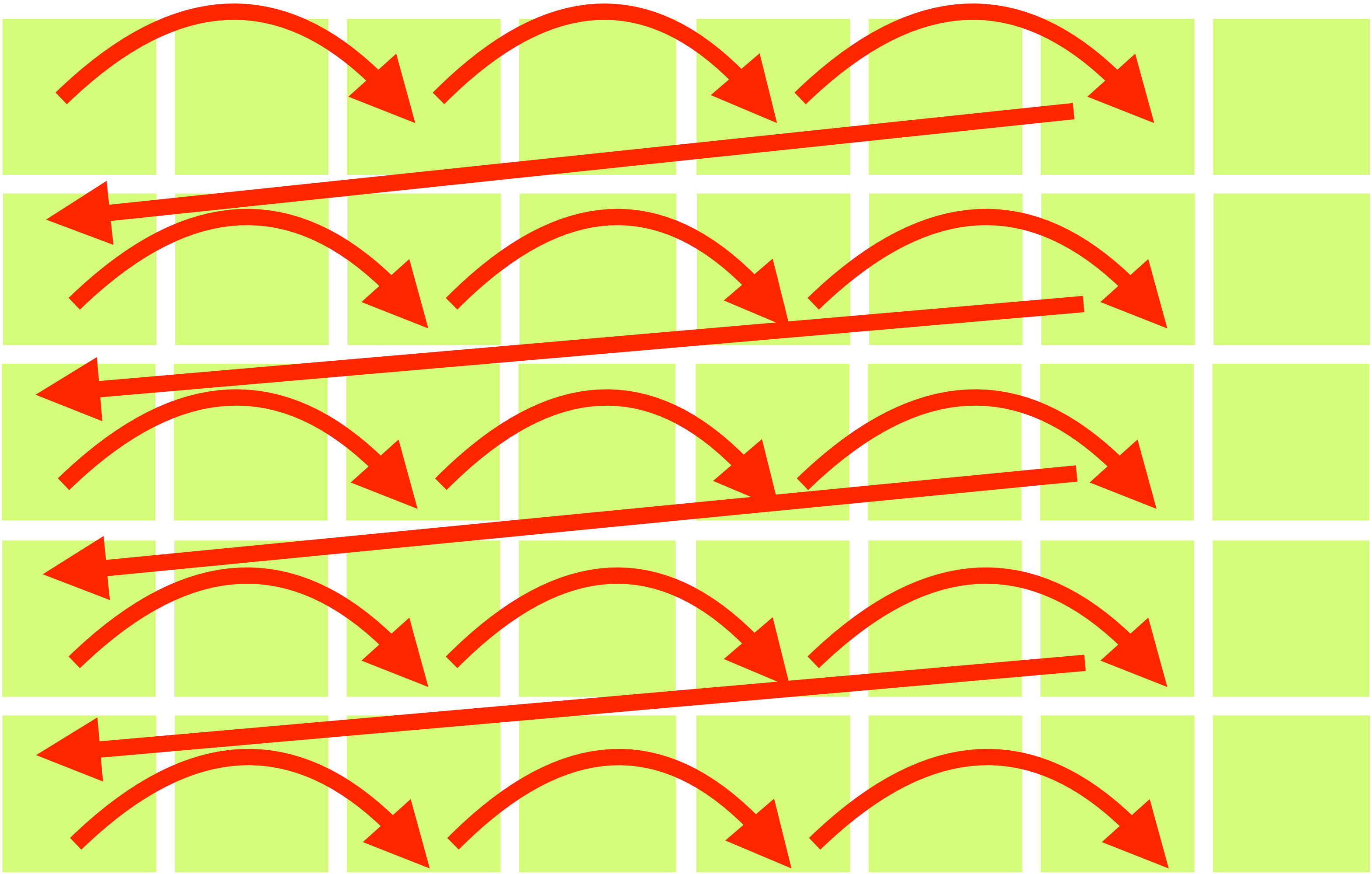
```
int array[n][m];
```



List



List
(newly allocated)

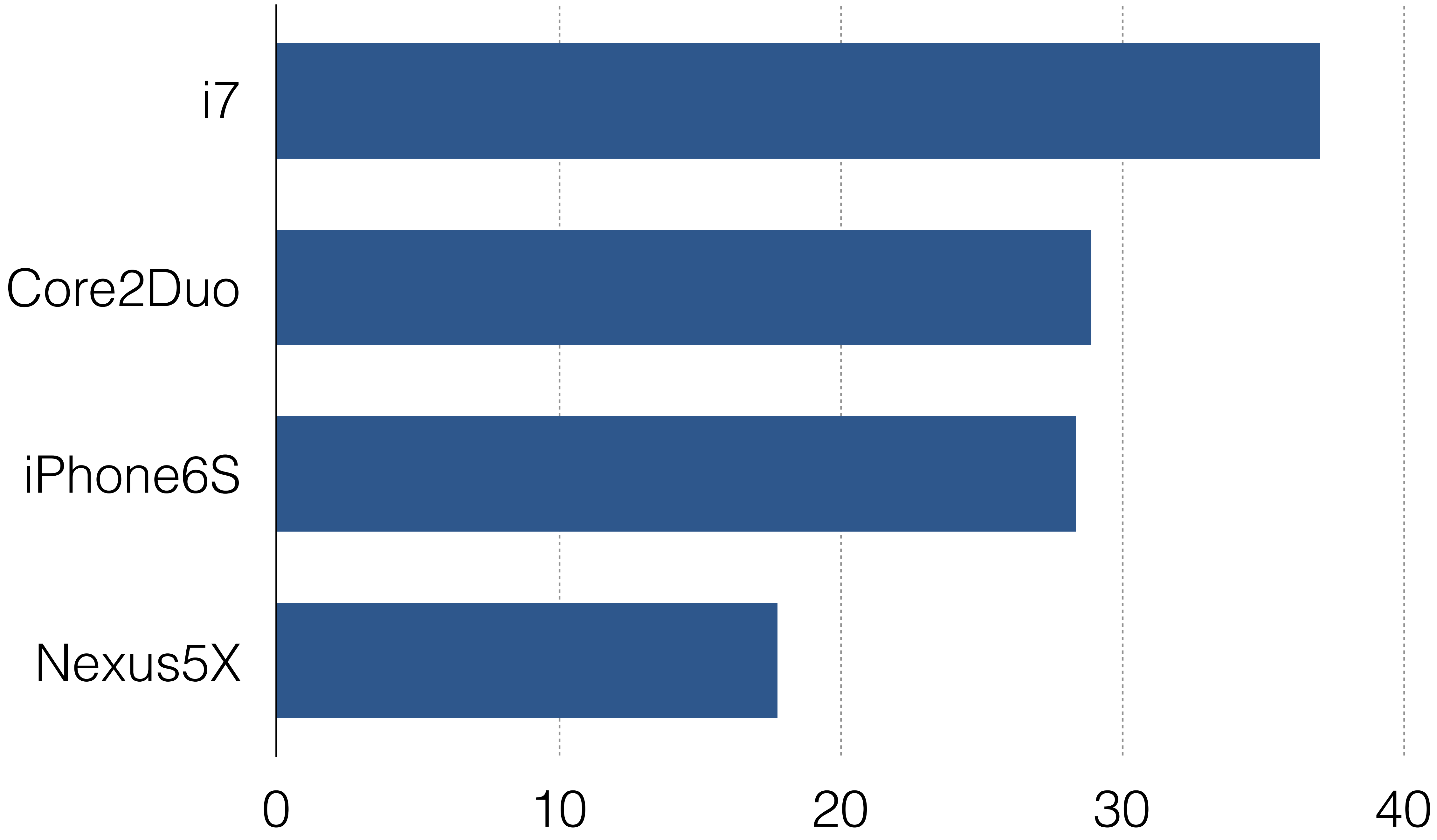


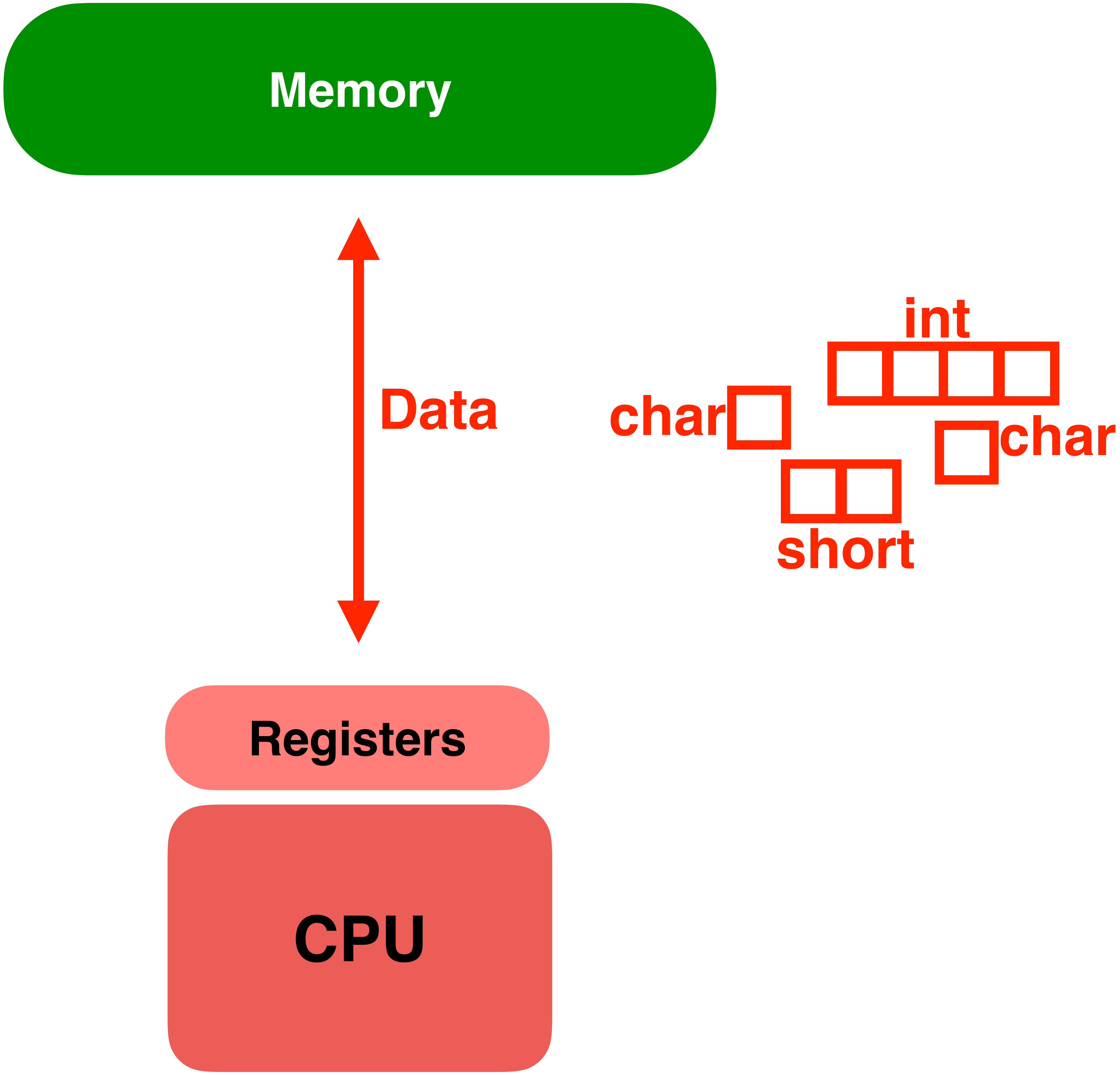
2D Array traversal

```
int array[n][n];


BENCHMARK_START(rowMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[i][j] += j;
}
BENCHMARK_STOP(rowMajor)

BENCHMARK_START(columnMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[j][i] += j;
}
BENCHMARK_STOP(columnMajor)
```

2D Array traversal benchmark

(36 MB array)





Memory

Data

int

char

char

short

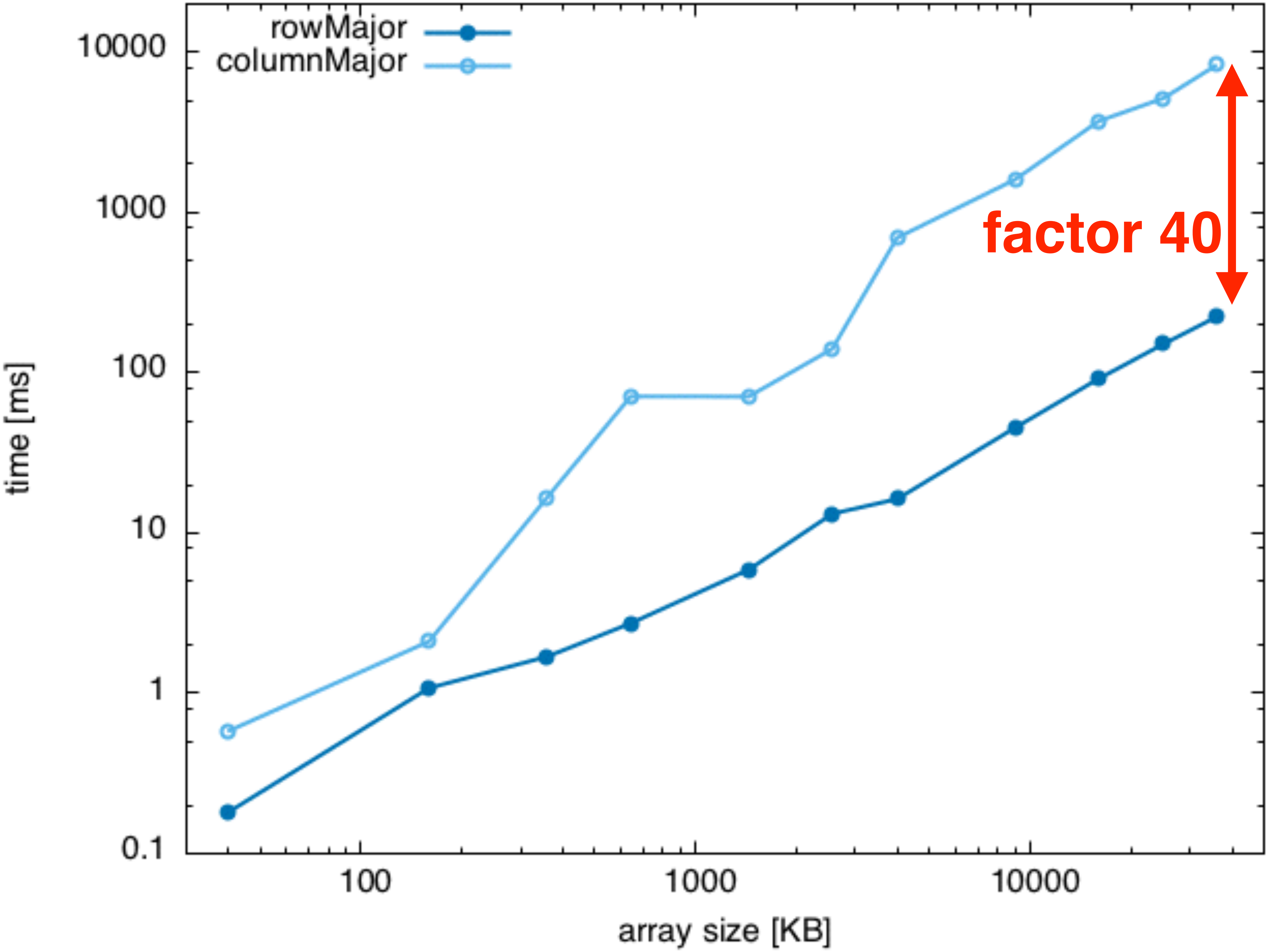
Registers

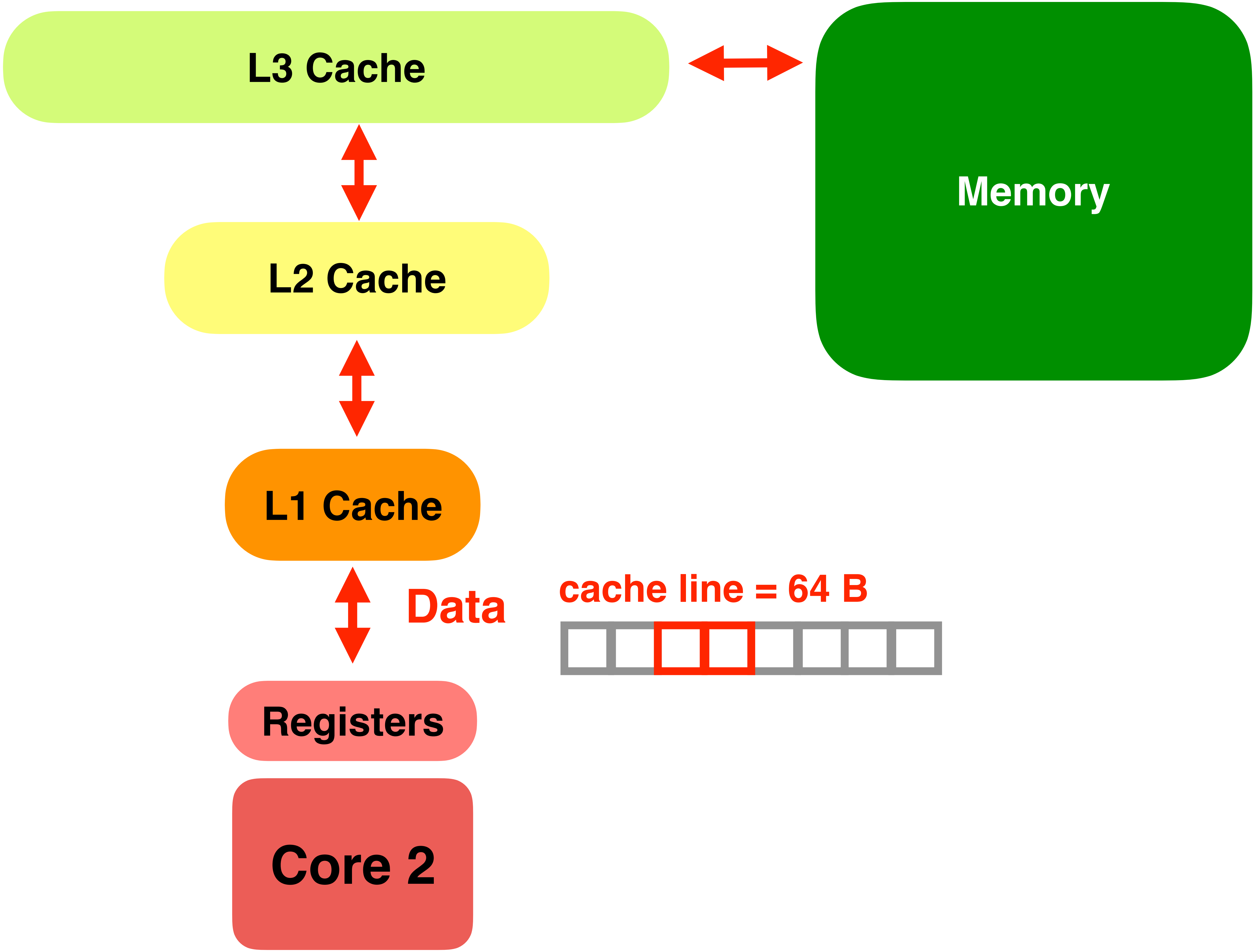
CPU

2D Array traversal

```
int array[n][n];


BENCHMARK_START(rowMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[i][j] += j;
}
BENCHMARK_STOP(rowMajor)

BENCHMARK_START(columnMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[j][i] += j;
}
BENCHMARK_STOP(columnMajor)
```

2D Array traversal benchmark

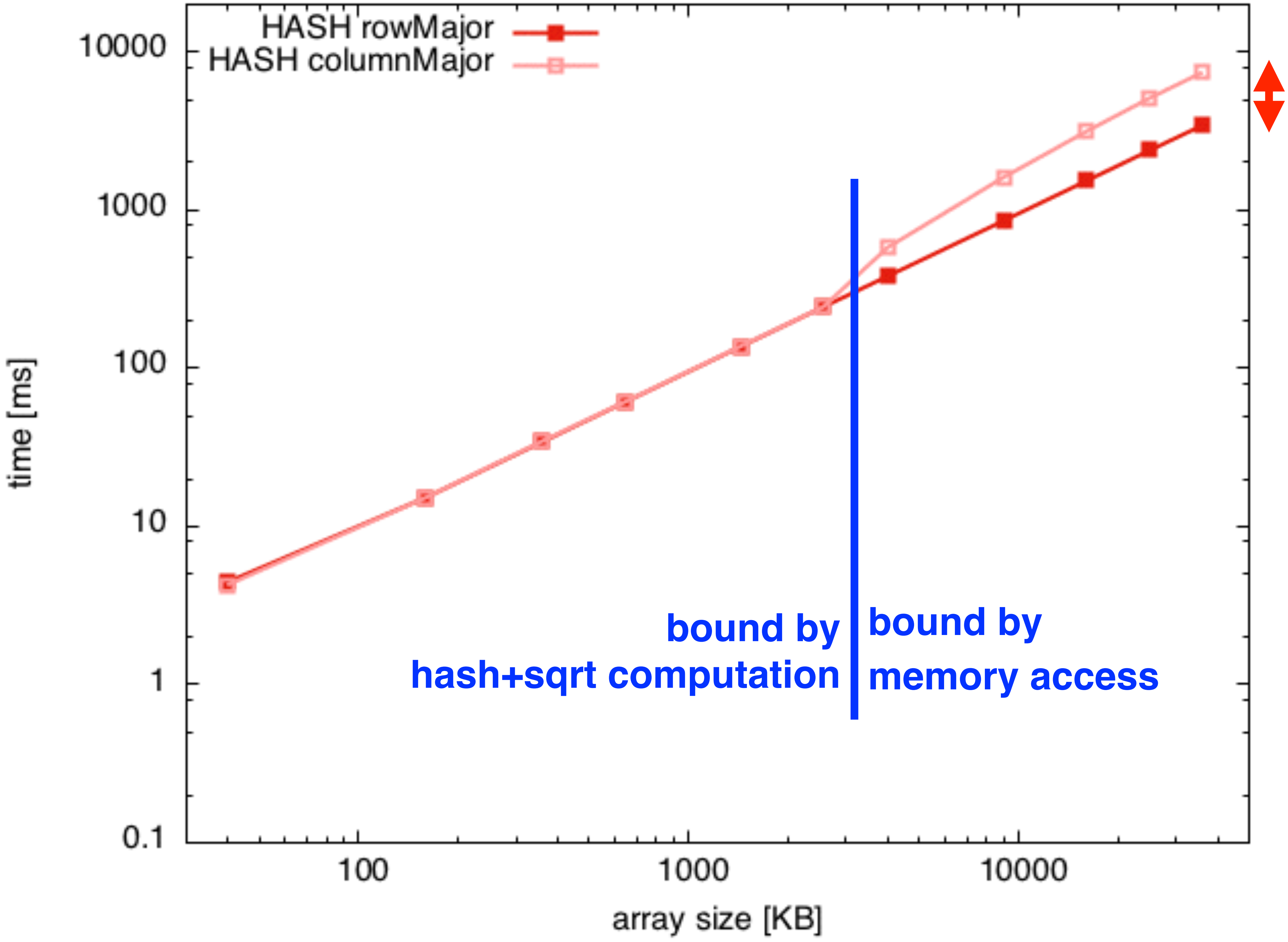



L3 Cache
Memory
L2 Cache
L1 Cache
Data
cache line = 64 B
Registers
Core 2

2D Array traversal + some work

```
float array[n][n];


BENCHMARK_START(rowMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[i][j] += std::sqrt (std::hash<int>() (j * n + i));
}
BENCHMARK_STOP(rowMajor)

BENCHMARK_START(columnMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[j][i] += std::sqrt (std::hash<int>() (j * n + i));
}
BENCHMARK_STOP(columnMajor)
```

2D Array traversal benchmark

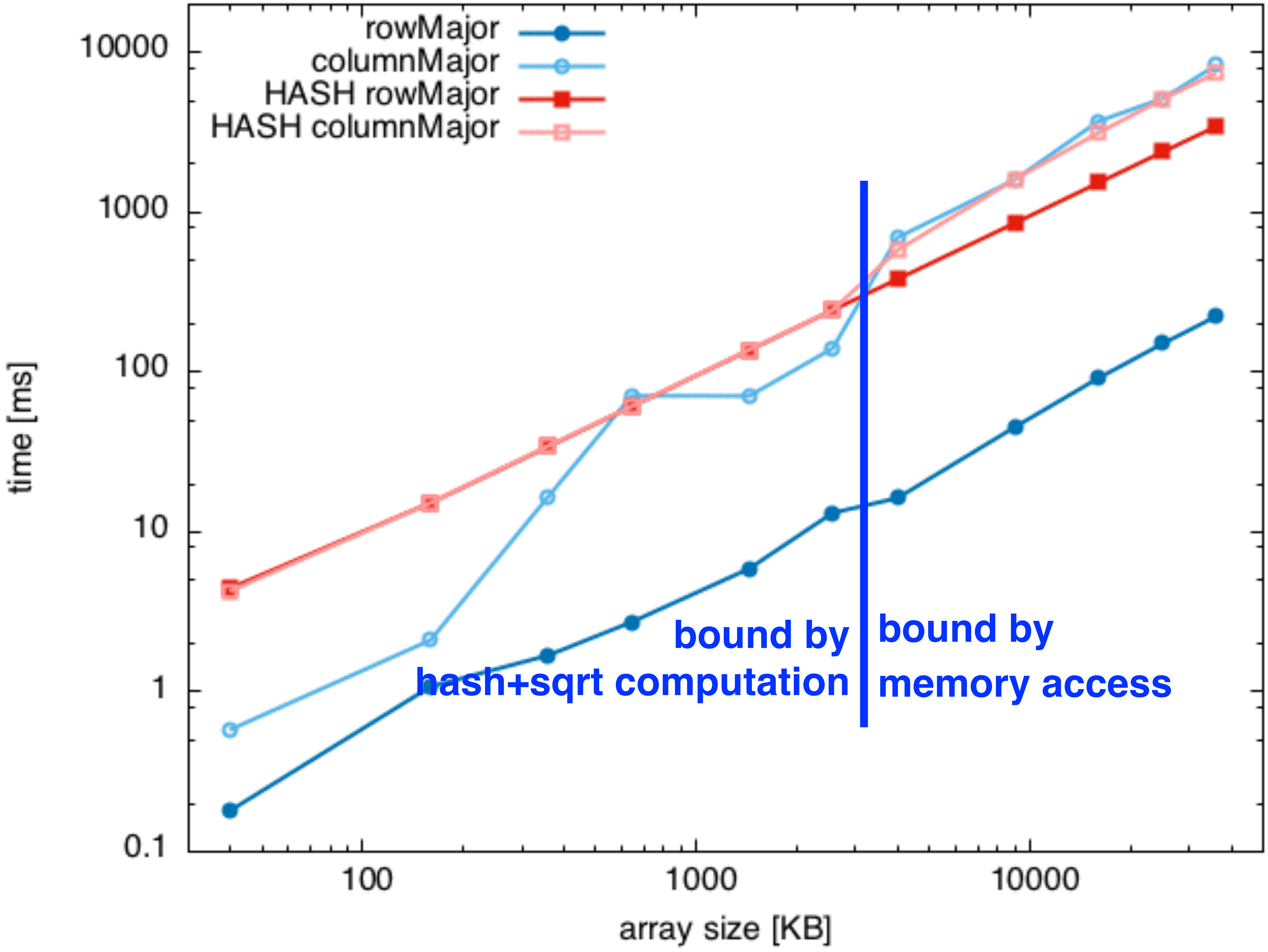


bound by
hash+sqrt computation

bound by
memory access

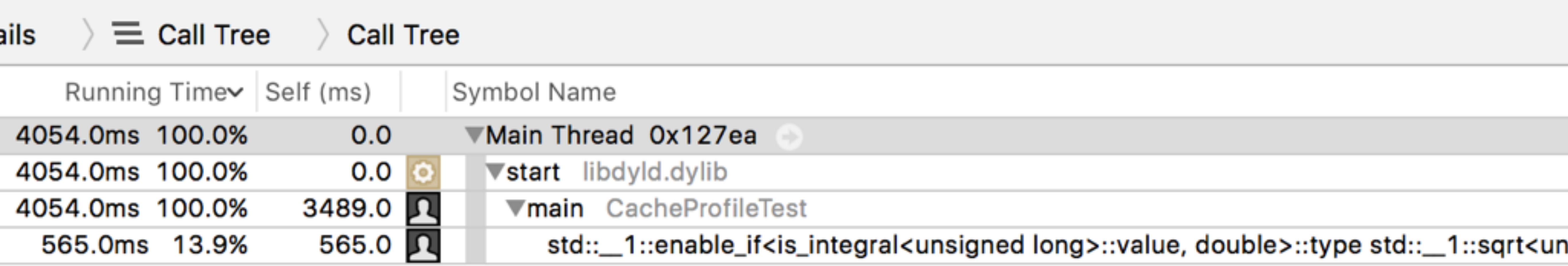
factor
2.2

2D Array traversal benchmark



2D Array traversal: time profile

(Xcode Instruments)



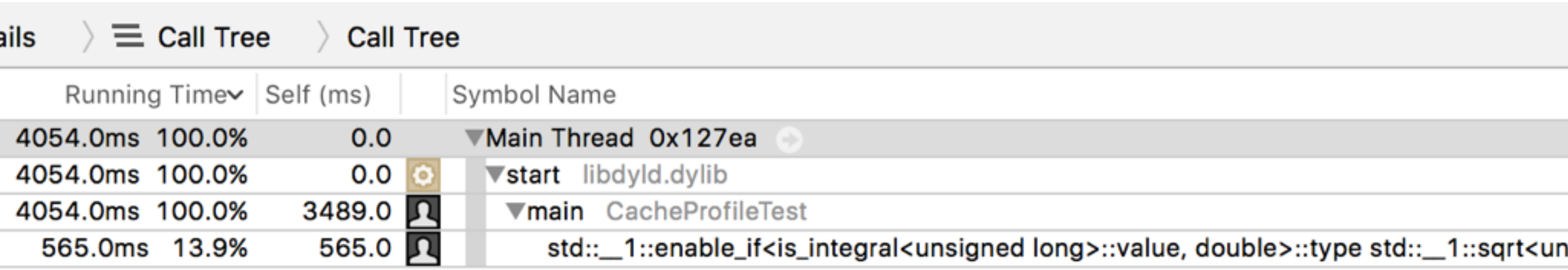
Running Time⌄		Self (ms)		Symbol Name
4054.0ms	100.0%	0.0		▼Main Thread 0x127ea
4054.0ms	100.0%	0.0	⚙	▼start libdyld.dylib
4054.0ms	100.0%	3489.0	👤	▼main CacheProfileTest
565.0ms	13.9%	565.0	👤	std::__1::enable_if<is_integral<unsigned long>::value, double>::type std::__1::sqrt<un

2D Array traversal + some work

```
float array[n][n];


BENCHMARK_START(rowMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[i][j] += std::sqrt (std::hash<int>() (j * n + i));
}
BENCHMARK_STOP(rowMajor)

BENCHMARK_START(columnMajor,100)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            array[j][i] += std::sqrt (std::hash<int>() (j * n + i));
}
BENCHMARK_STOP(columnMajor)
```

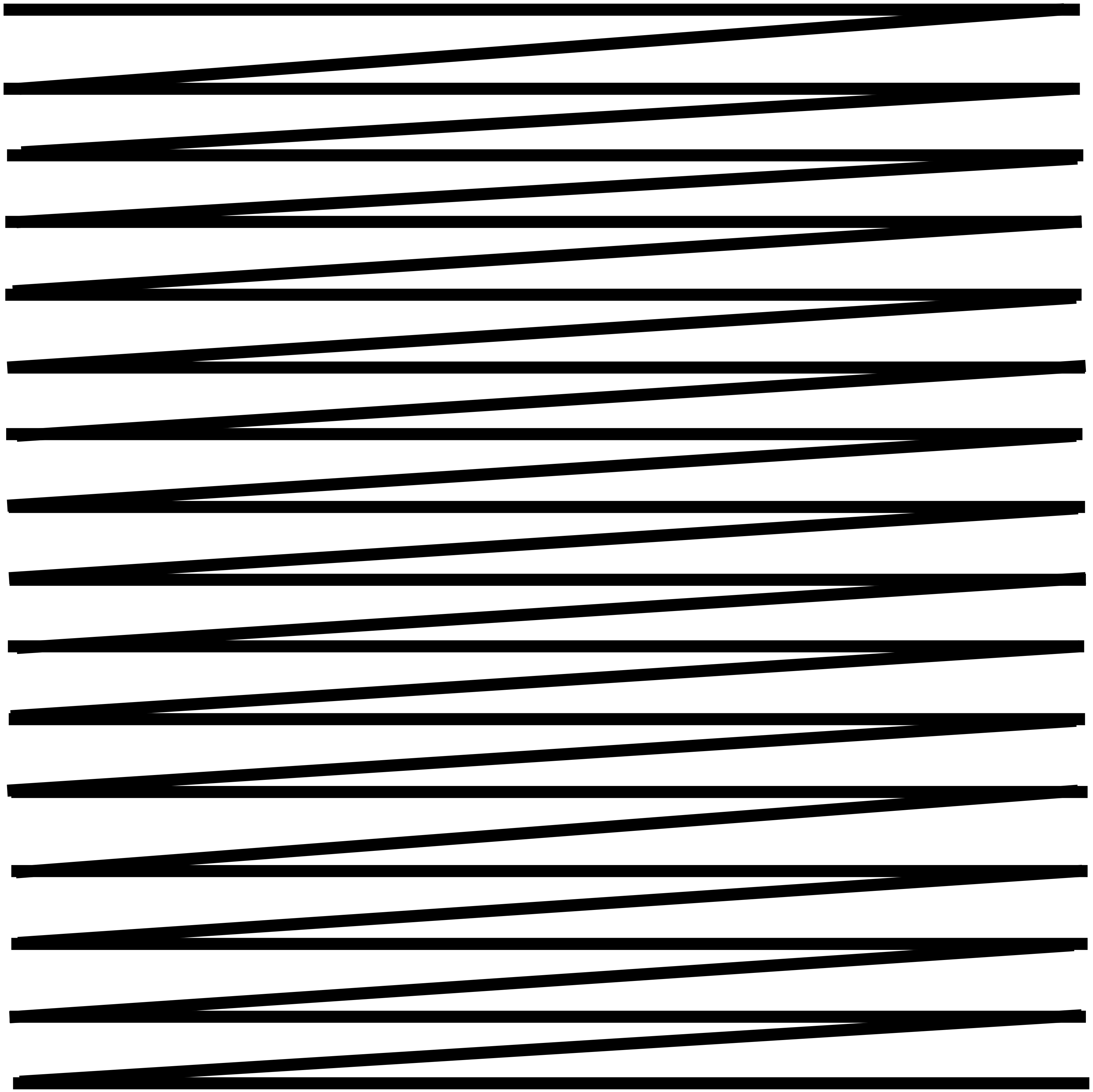
2D Array traversal: time profile

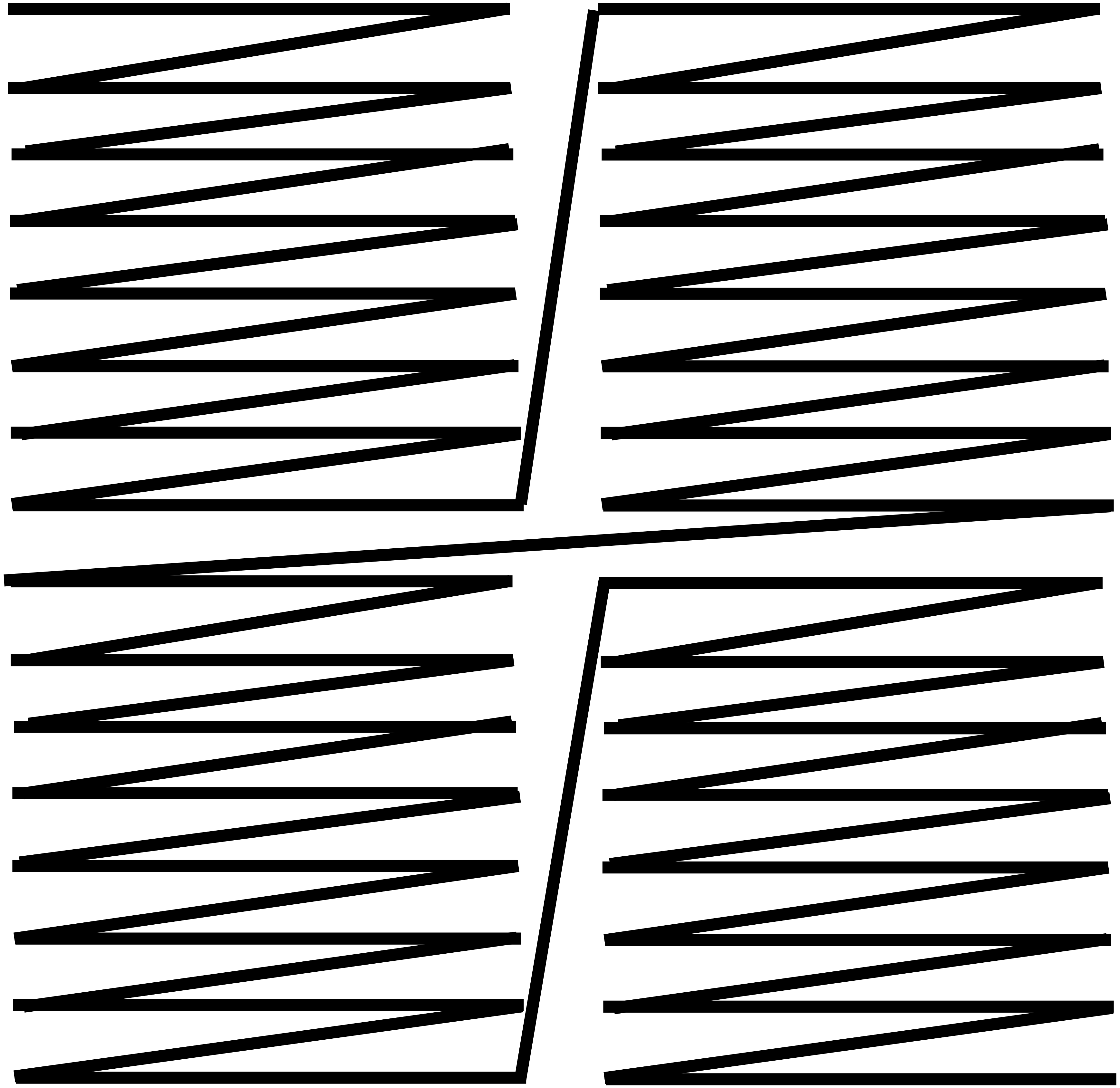
(Xcode Instruments)

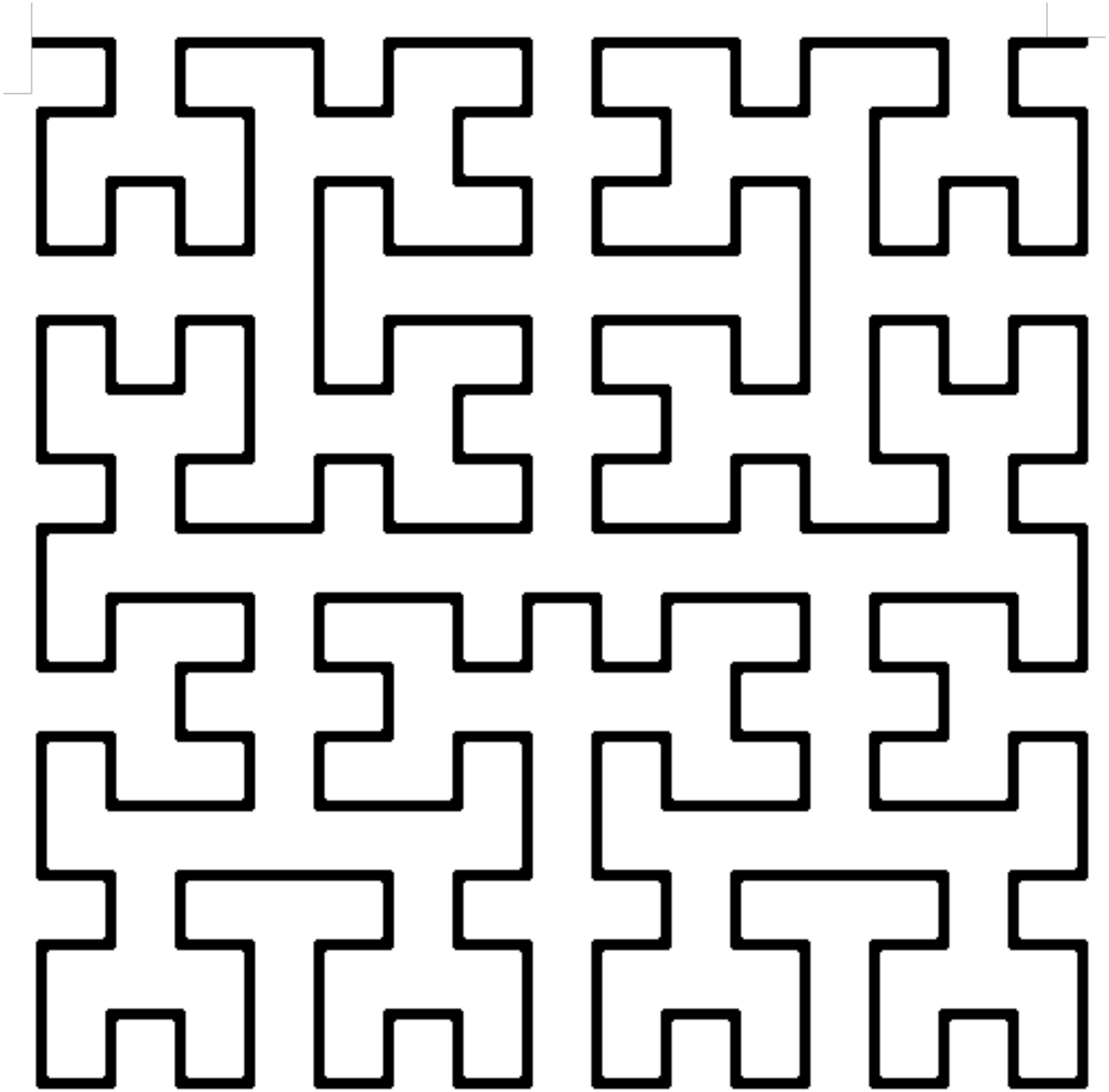


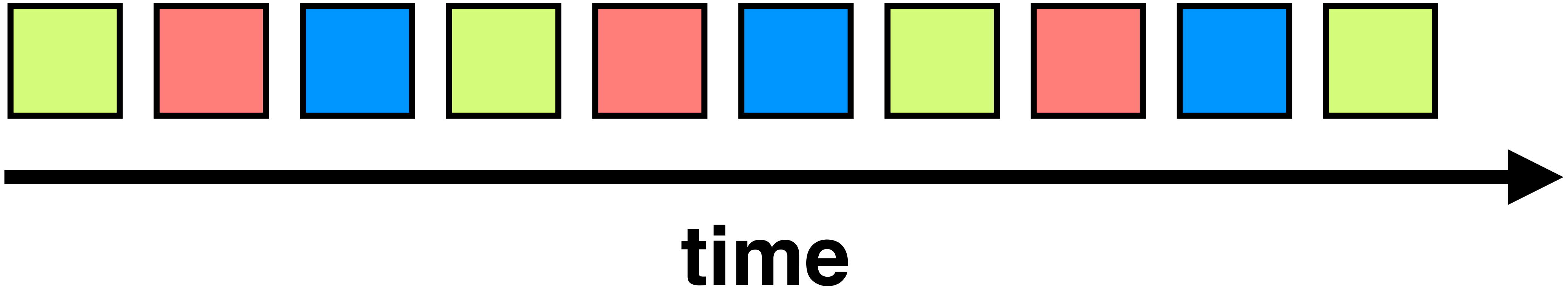
ails 〉 ≡ Call Tree 〉 Call Tree

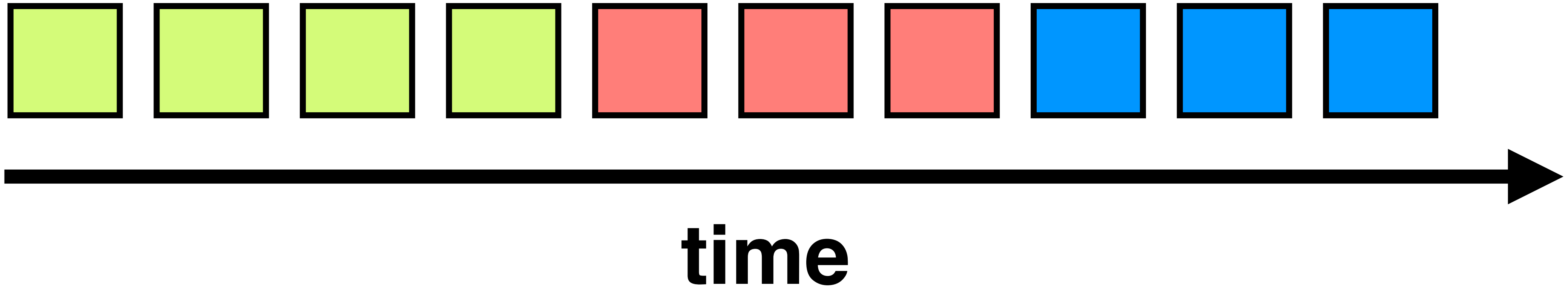
Running Time⌄		Self (ms)		Symbol Name
4054.0ms	100.0%	0.0		▼Main Thread 0x127ea
4054.0ms	100.0%	0.0	⚙	▼start libdyld.dylib
4054.0ms	100.0%	3489.0	👤	▼main CacheProfileTest
565.0ms	13.9%	565.0	👤	std::__1::enable_if<is_integral<unsigned long>::value, double>::type std::__1::sqrt<un








time


time

```
struct Foo
{
    char c;
    double d;
    short s;
    int i;
};
```

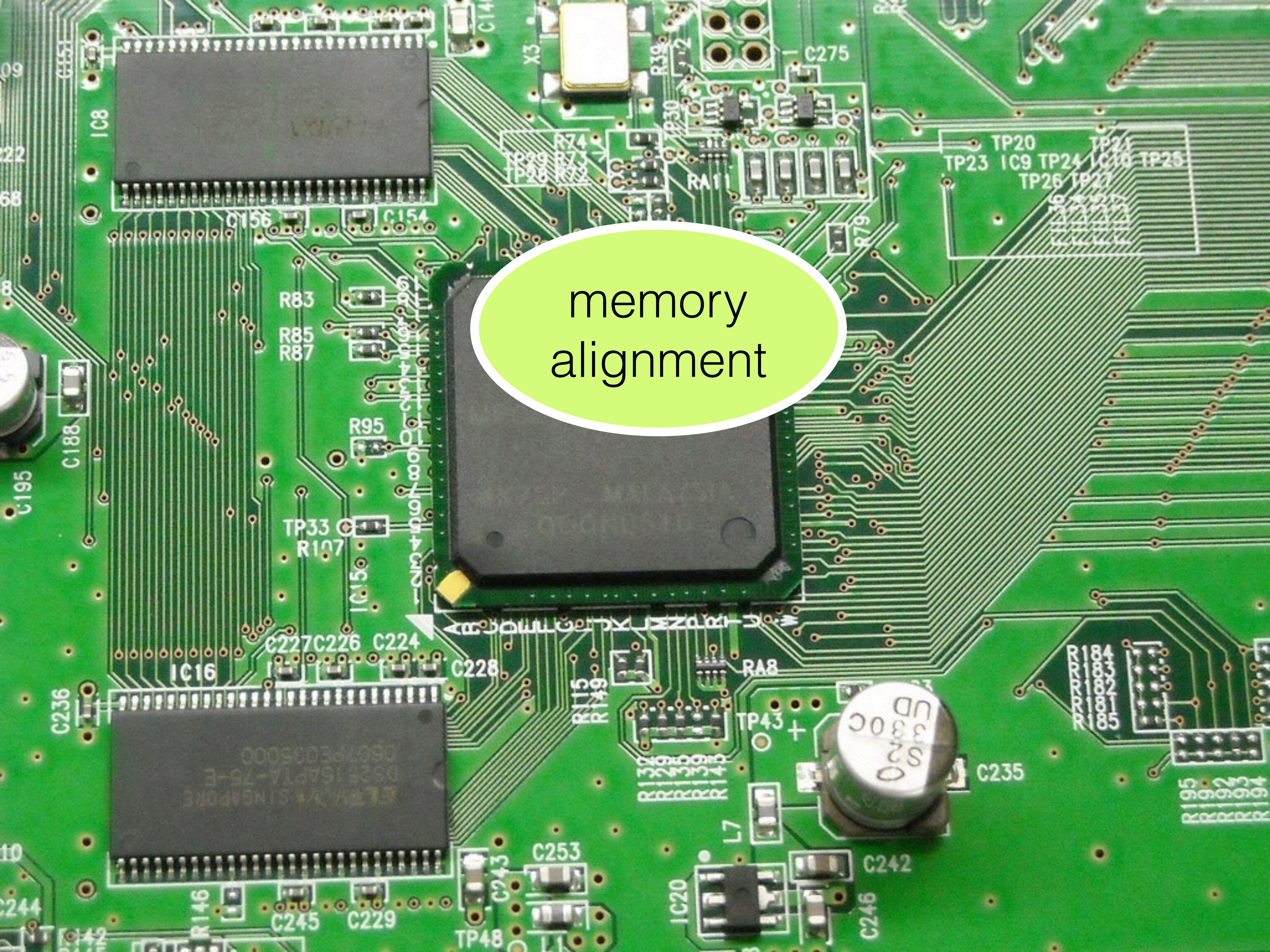
```
class Bar
{
    uint64_t lo;
    uint64_t mid;
    uint64_t hi;
};
```

```
std::vector<Foo> foos (1000);
std::vector<Bar> bars (1000);


doSomething (foos[i], bars[i]);
```

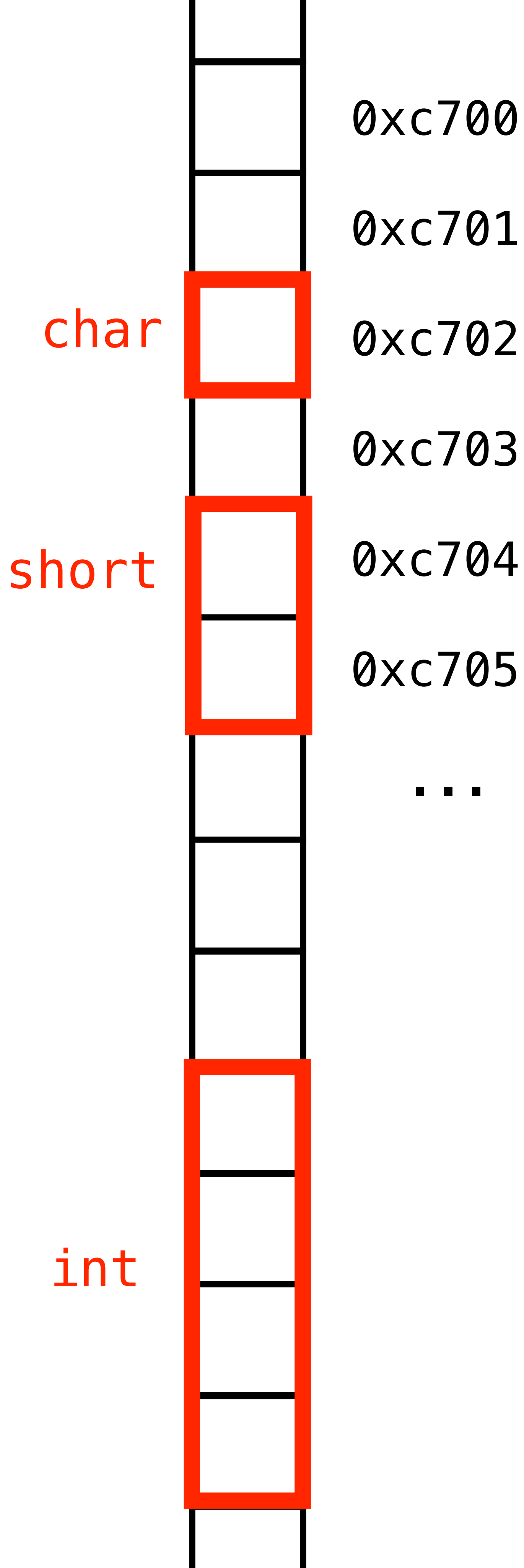
```
class Baz
{
    Foo foo;
    Bar bar;
};
```

```
std::vector<Baz> bazs (1000);
```

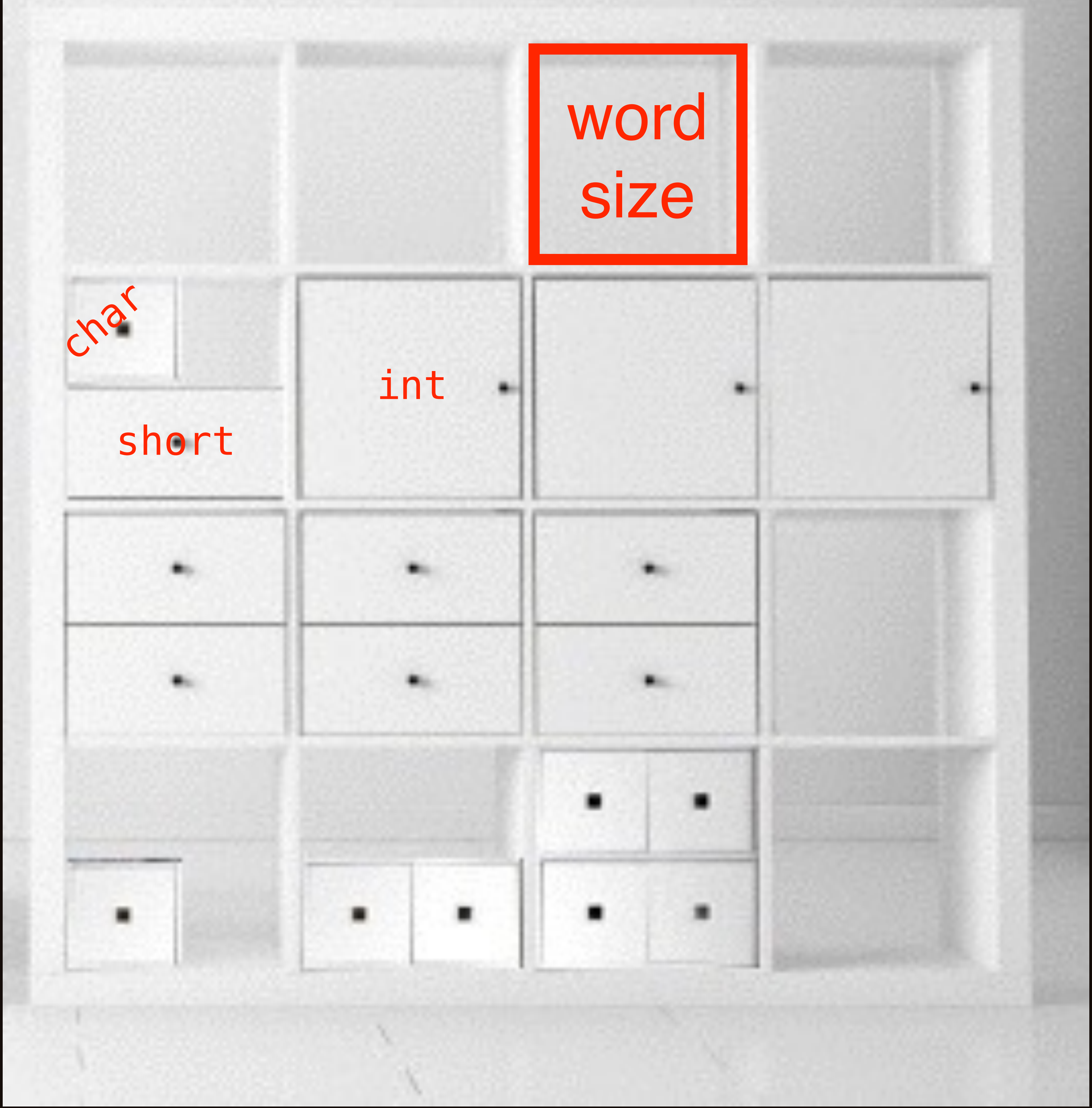
```
doSomething (bazs[i]);
```



memory alignment


0xc700
0xc701
char
0xc702
0xc703
short
0xc704
0xc705
...
int

ALIGNOF



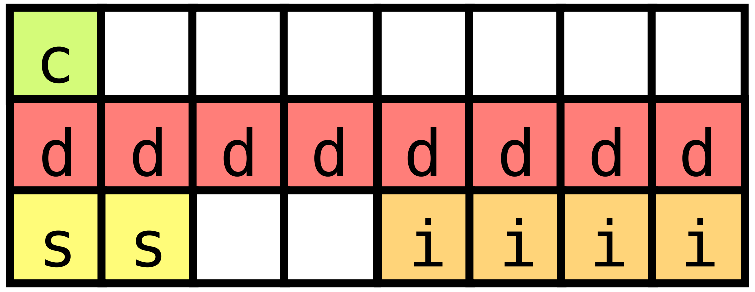
Alignment requirements

(on mainstream desktop & mobile platforms)

T	sizeof (T)	alignof (T)
char	1	1
short	2	2
int	4	4
uint64_t	8	8
long double	8 or 16	8 or 16
std::max_align_t	8 or 16	8 or 16

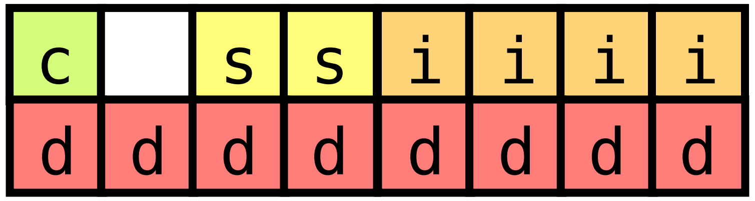
Alignment of user-defined classes

```
struct Foo
{
    char c;
    double d;
    short s;
    int i;
};
```



c							
d	d	d	d	d	d	d	d
s	s			i	i	i	i

```
struct Foo
{
    char c;
    short s;
    int i;
    double d;
};
```

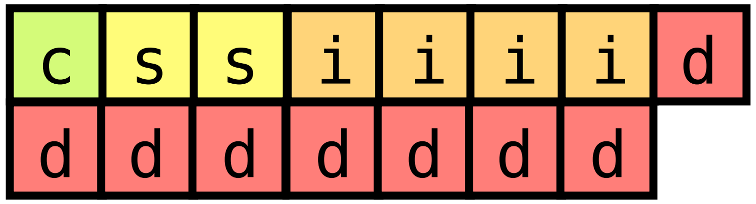


c		s	s	i	i	i	i
d	d	d	d	d	d	d	d

Packed structs

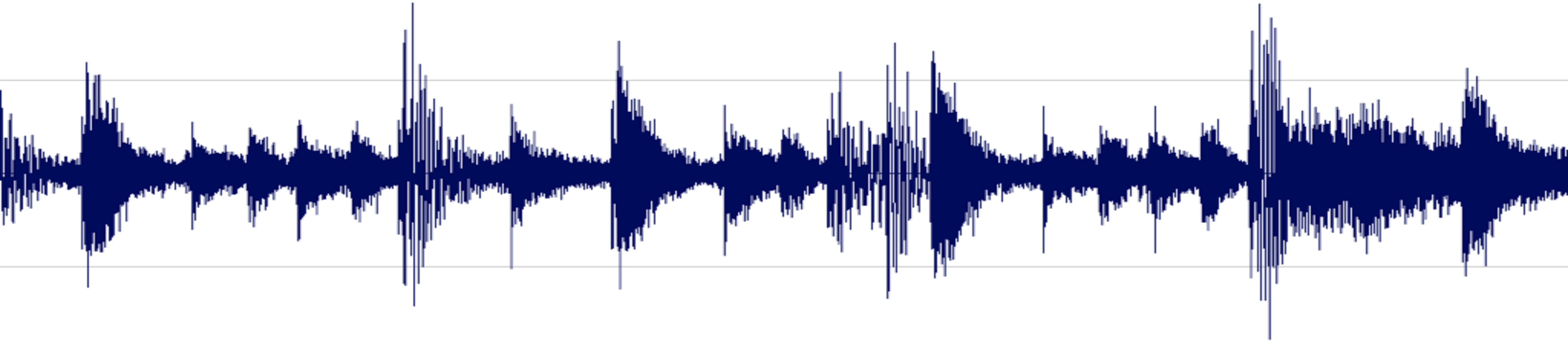
```
struct Foo
{
    char c;
    short s;
    int i;
    double d;
}
__attribute__((packed));
```

🙁

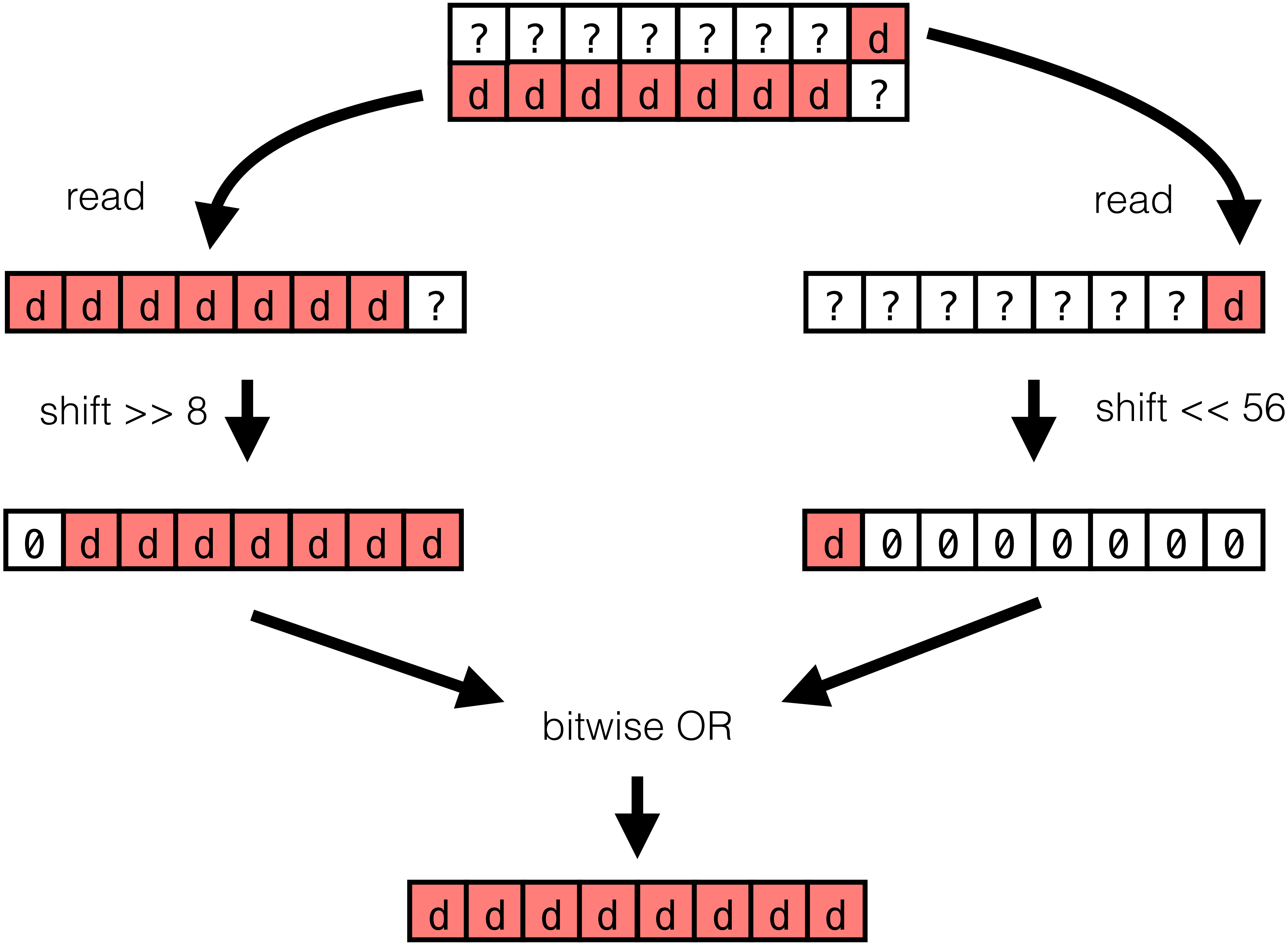


c	s	s	i	i	i	i	d
d	d	d	d	d	d	d	

```
struct BWAVChunk
{
    char     description[256];
    char     originator[32];
    char     originatorRef[32];
    char     originationDate[10];
    char     originationTime[8];
    uint32_t timeRefLow;
    uint32_t timeRefHigh;
    uint16_t version;
    uint8_t  umid[64];
    uint8_t  reserved[190];
    char     codingHistory[1];
}
__attribute__((packed));
```



Unaligned memory access



Unaligned access benchmark

i7

Core2Duo

iPhone6S

AppleA9

Nexus5X

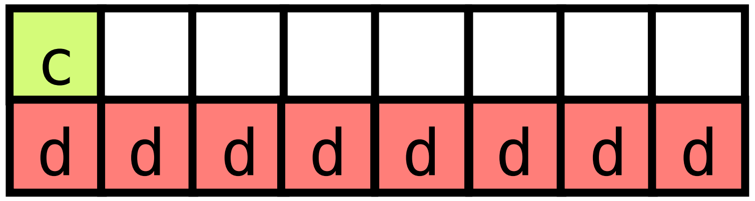
Snapdragon 808

```
clang -x c++ -std=c++11 -stdlib=libc++ -O3
```

Unaligned access benchmark

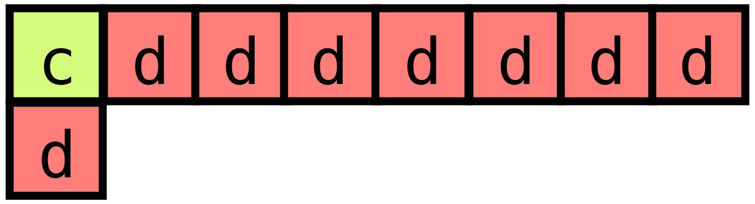
```
template <typename T>
struct AlignedStruct
{
    char c;
    T value;
};
```

AlignedStruct<double>



```
template <typename T>
struct PackedStruct
{
    char c;
    T value;
}
__attribute__((packed));
```

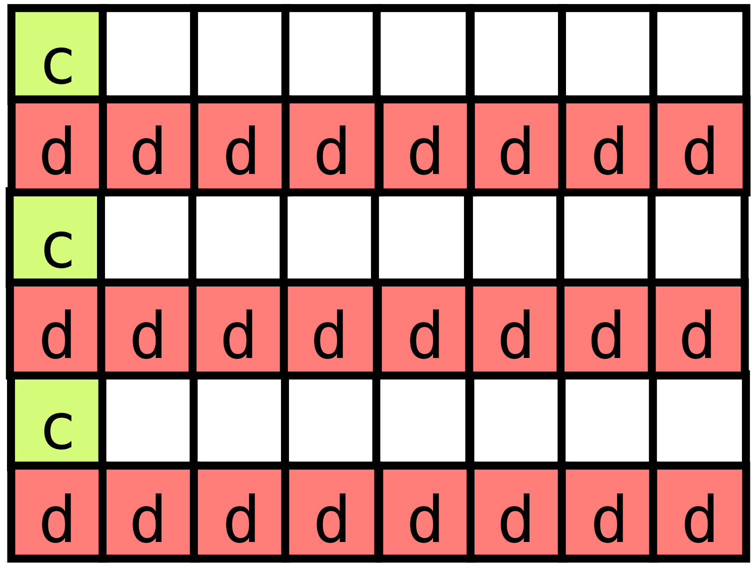
PackedStruct<double>



Unaligned access benchmark

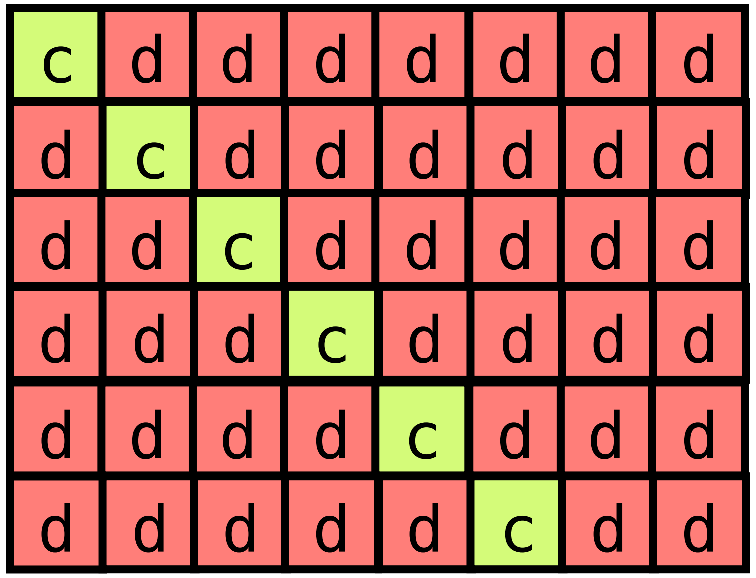
```cpp
template <typename StructT>
auto test (std::size_t size)
{
    std::vector<float> vecFloats (size);

    std::iota (vecFloats.begin(),
               vecFloats.end(),
               0.0);

    std::vector<StructT> vecStructs (size);

    std::generate (vecStructs.begin(),
                   vecStructs.end(),
                   [] { return StructT {
                       'x', double (rand() % 100)};
                   });

    BENCHMARK_START(loop,10000)

    for (int i = 0; i < size; ++i)
        vecStructs[i].value += vecFloats[i];

    BENCHMARK_STOP(loop)

    return vecStructs;
}
```

AlignedStruct<double>



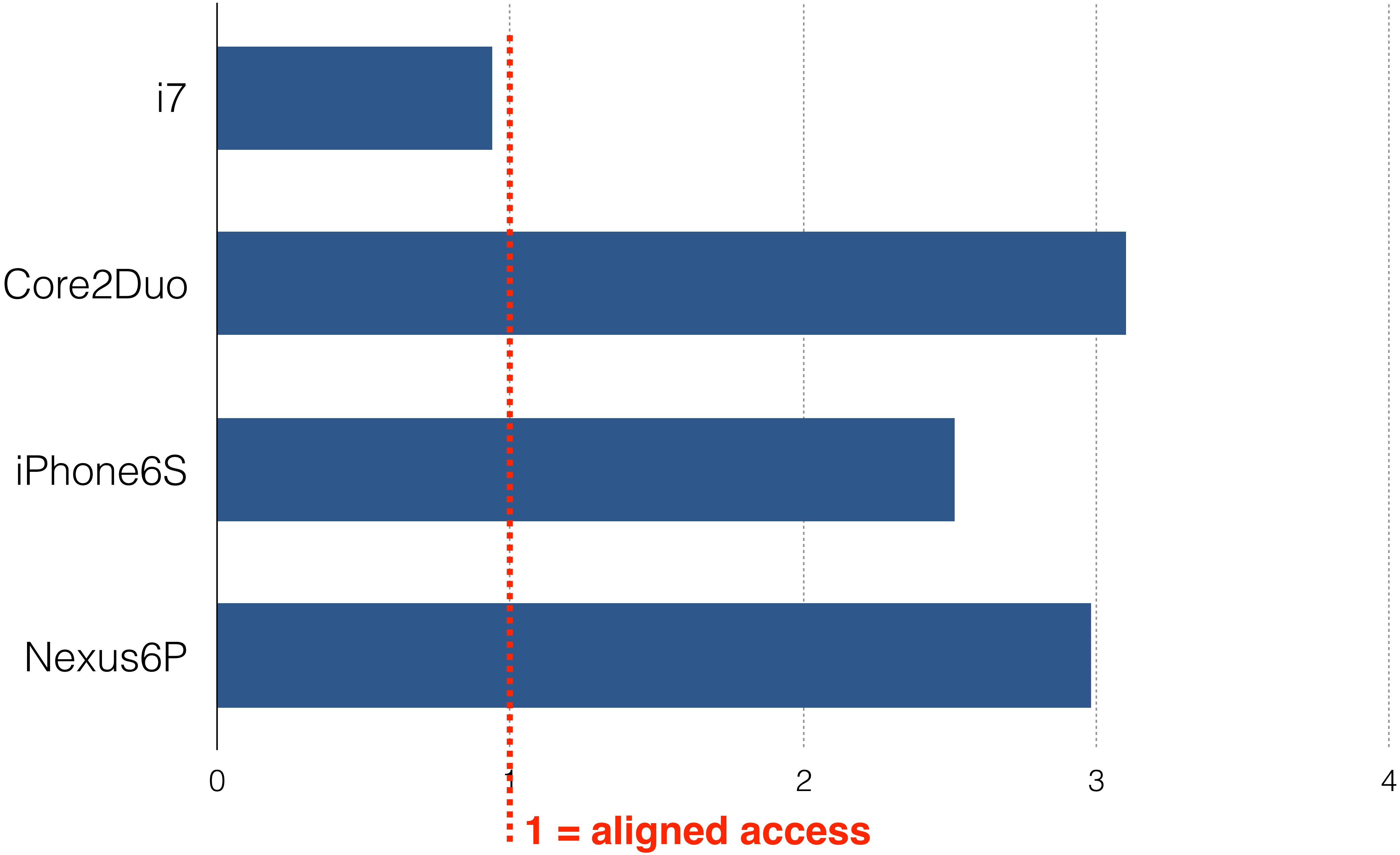
c							
d	d	d	d	d	d	d	d
c							
d	d	d	d	d	d	d	d
c							
d	d	d	d	d	d	d	d

PackedStruct<double>



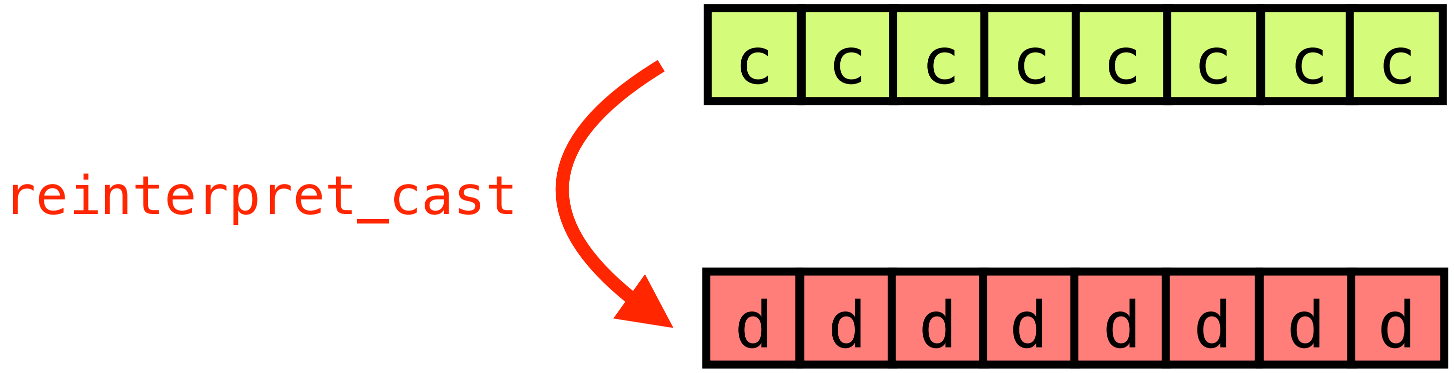
c	d	d	d	d	d	d	d
d	c	d	d	d	d	d	d
d	d	c	d	d	d	d	d
d	d	d	c	d	d	d	d
d	d	d	d	c	d	d	d
d	d	d	d	d	c	d	d

Unaligned access benchmark
(`double` offset by 1 byte)



Casts between differently-aligned types

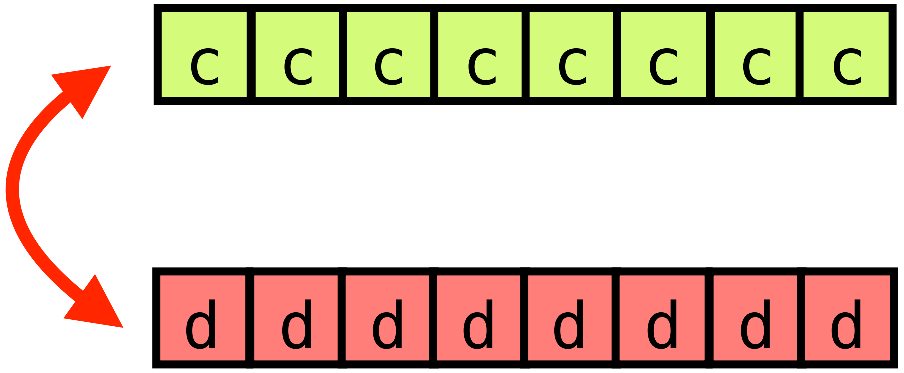
```
struct Foo  // alignof (Foo) == 1
{
    char c[8];

    double interpretAsDouble() noexcept
    {
        return *(reinterpret_cast<double*> (c));
    }
};
```


reinterpret_cast

```
--------- beginning of crash
Fatal signal 7 (SIGBUS), code 1, fault addr 0xc20b8009
```

Casts between differently-aligned types

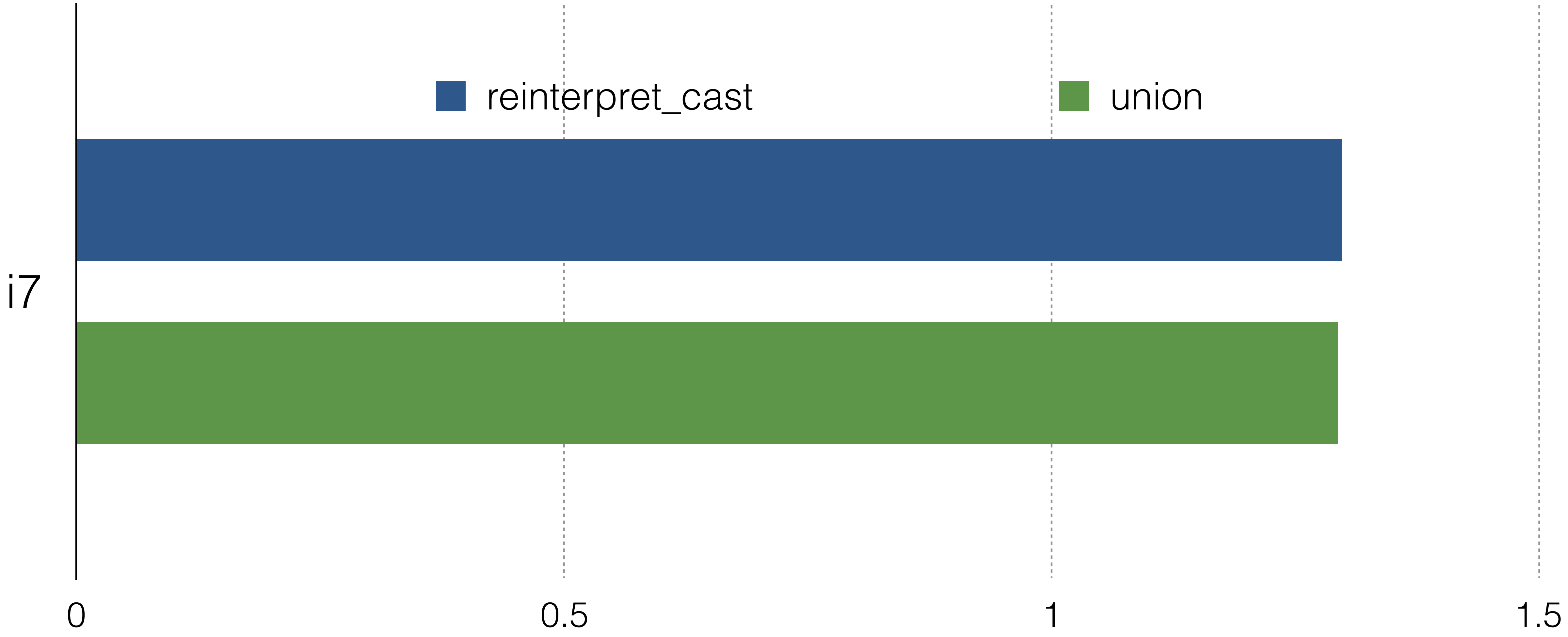
```
union Foo      // alignof (Foo) == 8
{
    char c[8];
    double d;
};
```



reinterpret_cast vs. union

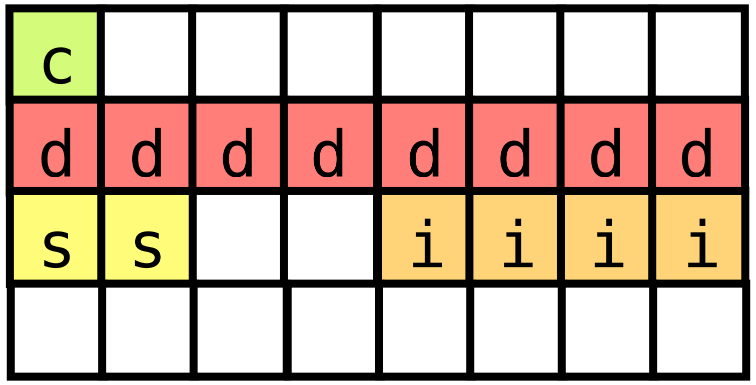
```
uint64_t doubleAsInt (double x) noexcept   // version 1
{
   union { double asDouble; uint64_t asInt; } u;
   u.asDouble = x;
   return u.asInt;
}

uint64_t doubleAsInt (double x) noexcept   // version 2
{
   return *(reinterpret_cast<uint64_t*> (&x));
}

int main()
{
   std::vector<double> vec(4000);
   std::generate (vec.begin(), vec.end(), [] { return double (rand() % 1000); });

   uint64_t sum;

   BENCHMARK_START (doubleAsInt,10000)

   for (auto& x : vec)
     sum += doubleAsInt (x);

   BENCHMARK_STOP (doubleAsInt)

   std::cout << sum << '\n';
}
```

reinterpret_cast vs. union



Overalignment

```
struct alignas (32) Foo
{
    char c;
    double d;
    short s;
    int i;
};
```

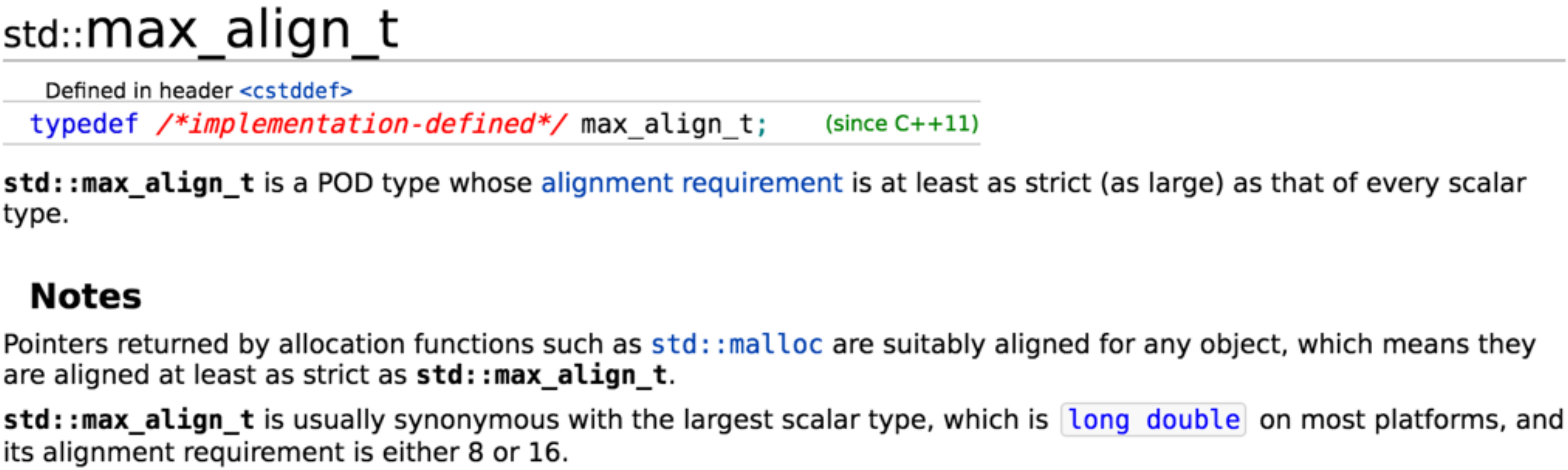


c							
d	d	d	d	d	d	d	d
s	s			i	i	i	i

Alignment on the heap

```
size_t bufferSize = 1024;

void* scratchBuffer = new char[bufferSize];

// ...and make this aligned by 8/16/32
```

Alignment on the heap



std::max_align_t

Defined in header <cstddef>

```
typedef /*implementation-defined*/ max_align_t;      (since C++11)
```

std::max_align_t is a POD type whose alignment requirement is at least as strict (as large) as that of every scalar type.

Notes

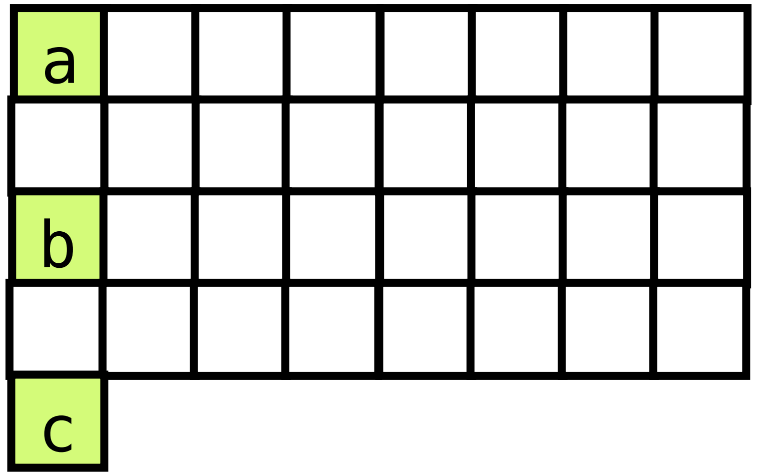
Pointers returned by allocation functions such as std::malloc are suitably aligned for any object, which means they are aligned at least as strict as **std::max_align_t**.

std::max_align_t is usually synonymous with the largest scalar type, which is `long double` on most platforms, and its alignment requirement is either 8 or 16.

Alignment on the heap

```
auto* a = new char;
auto* b = new char;
auto* c = new char;
```

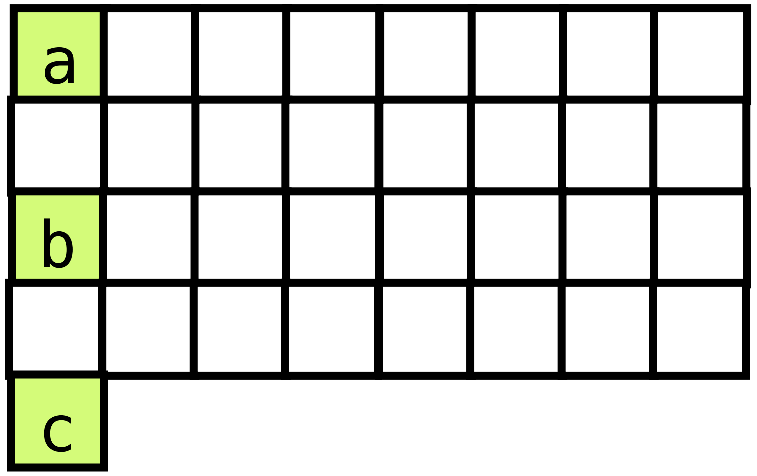
i7



Alignment on the heap

```
auto* a = new char;
auto* b = new char;
auto* c = new char;
```

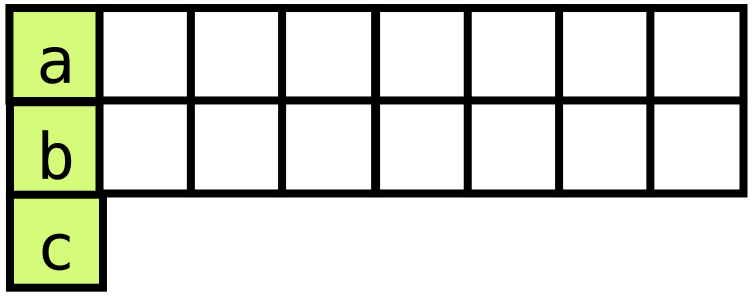
Core2Duo



Alignment on the heap

```
auto* a = new char;
auto* b = new char;
auto* c = new char;
```

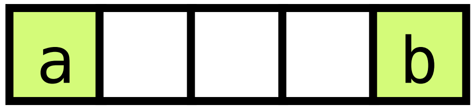
iPhone6S



Alignment on the heap

```
auto* a = new char;
auto* b = new char;
auto* c = new char;
```

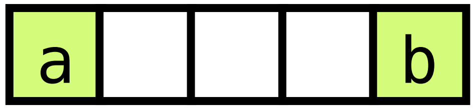
Nexus5X



Alignment on the heap

```
struct alignas (8) Foo
{
   char ch;
}

auto* a = new Foo;
auto* b = new Foo;
auto* c = new Foo;
```

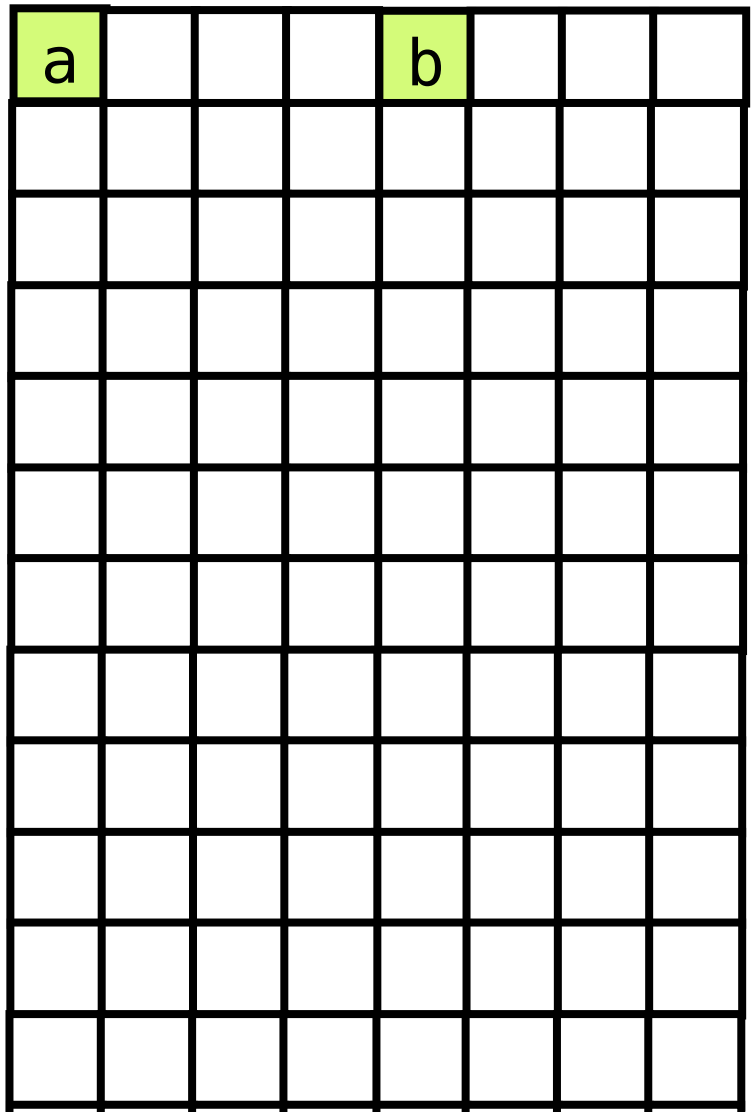
Nexus5X



Alignment on the heap

```
struct alignas (8) Foo
{
   char ch;
}

auto* a = new Foo;
auto* b = new Foo[1024];
```

Nexus5X



Alignment on the heap

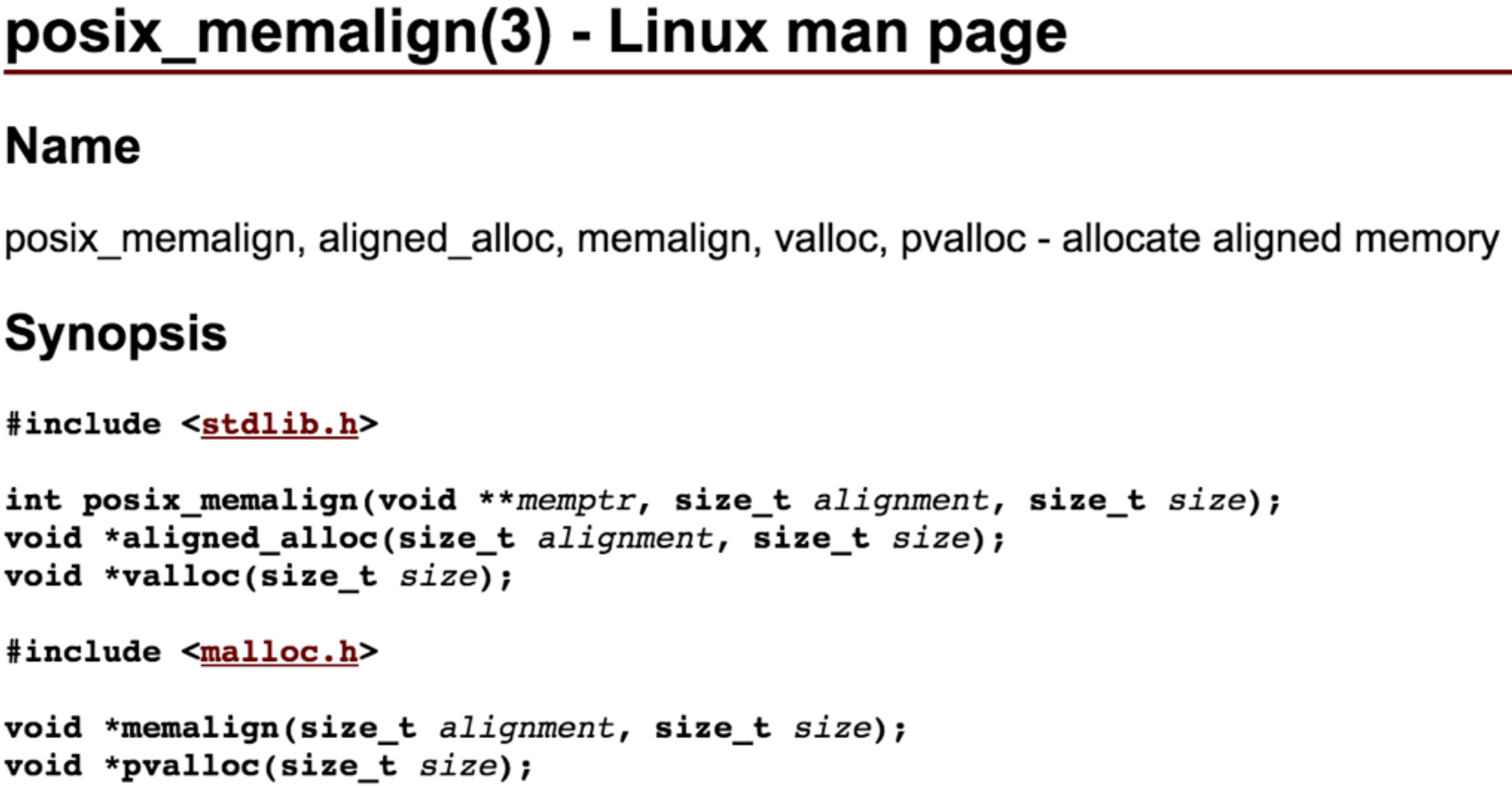
```
template <size_t Size, size_t Align = Size>
struct aligned_storage;

template <size_t Size, typename... Types>
struct aligned_union;
```

Alignment on the heap

```
size_t bufferSize = 1024;

void* scratchBuffer = static_cast<void*> (
  new std::aligned_storage<32, 32>::type[1024 / 32]);
```

Alignment on the heap

```
size_t bufferSize = 1024;
size_t bufferSizeWithPadding = bufferSize + 32;

void* scratchBuffer = new char[bufferSizeWithPadding];

if (! std::align (32, bufferSize,
                  scratchBuffer, bufferSizeWithPadding))
{
    // handle...
}
```



posix_memalign(3) - Linux man page

Name

posix_memalign, aligned_alloc, memalign, valloc, pvalloc - allocate aligned memory

Synopsis

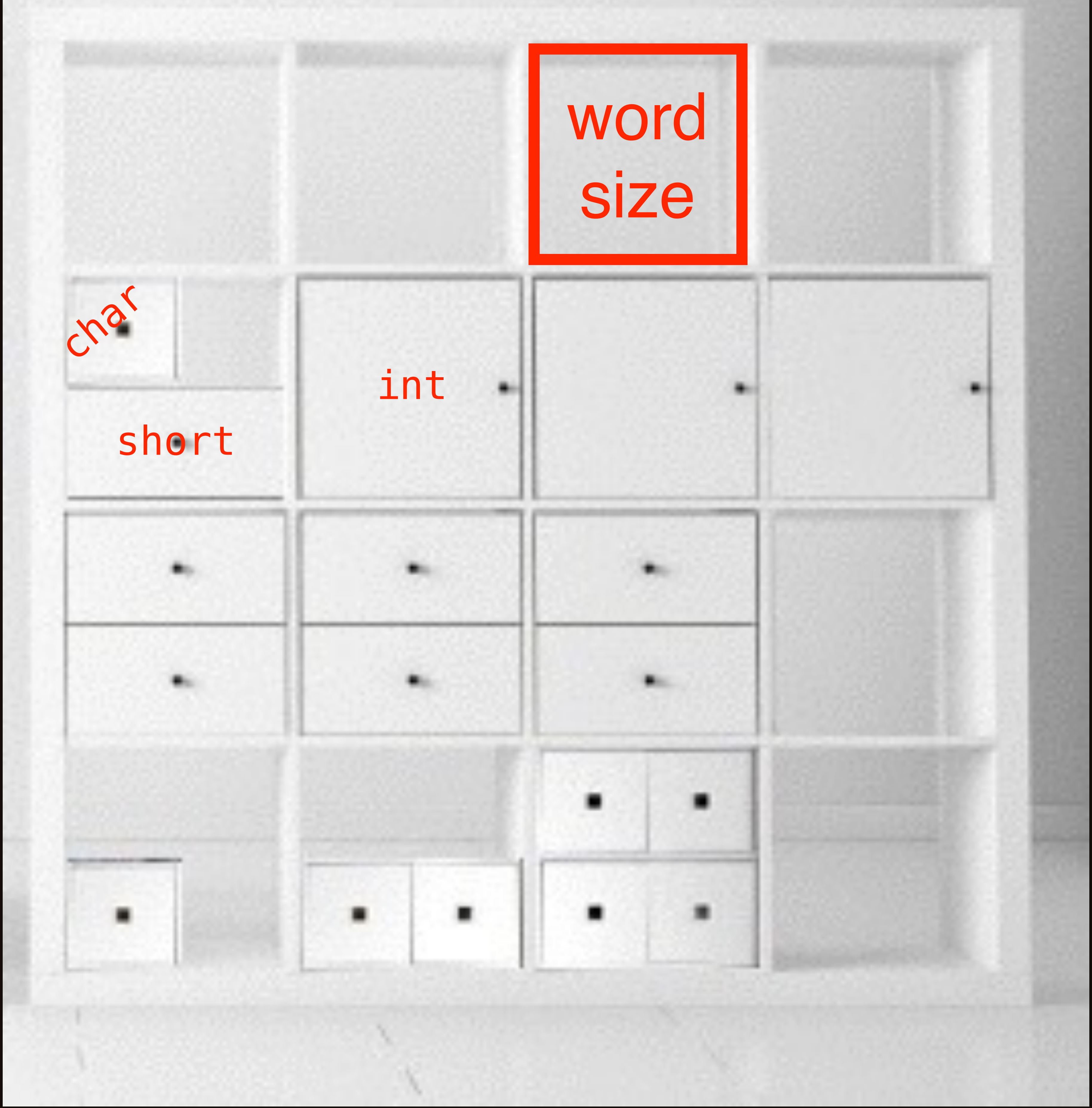
```
#include <stdlib.h>

int posix_memalign(void **memptr, size_t alignment, size_t size);
void *aligned_alloc(size_t alignment, size_t size);
void *valloc(size_t size);

#include <malloc.h>

void *memalign(size_t alignment, size_t size);
void *pvalloc(size_t size);
```

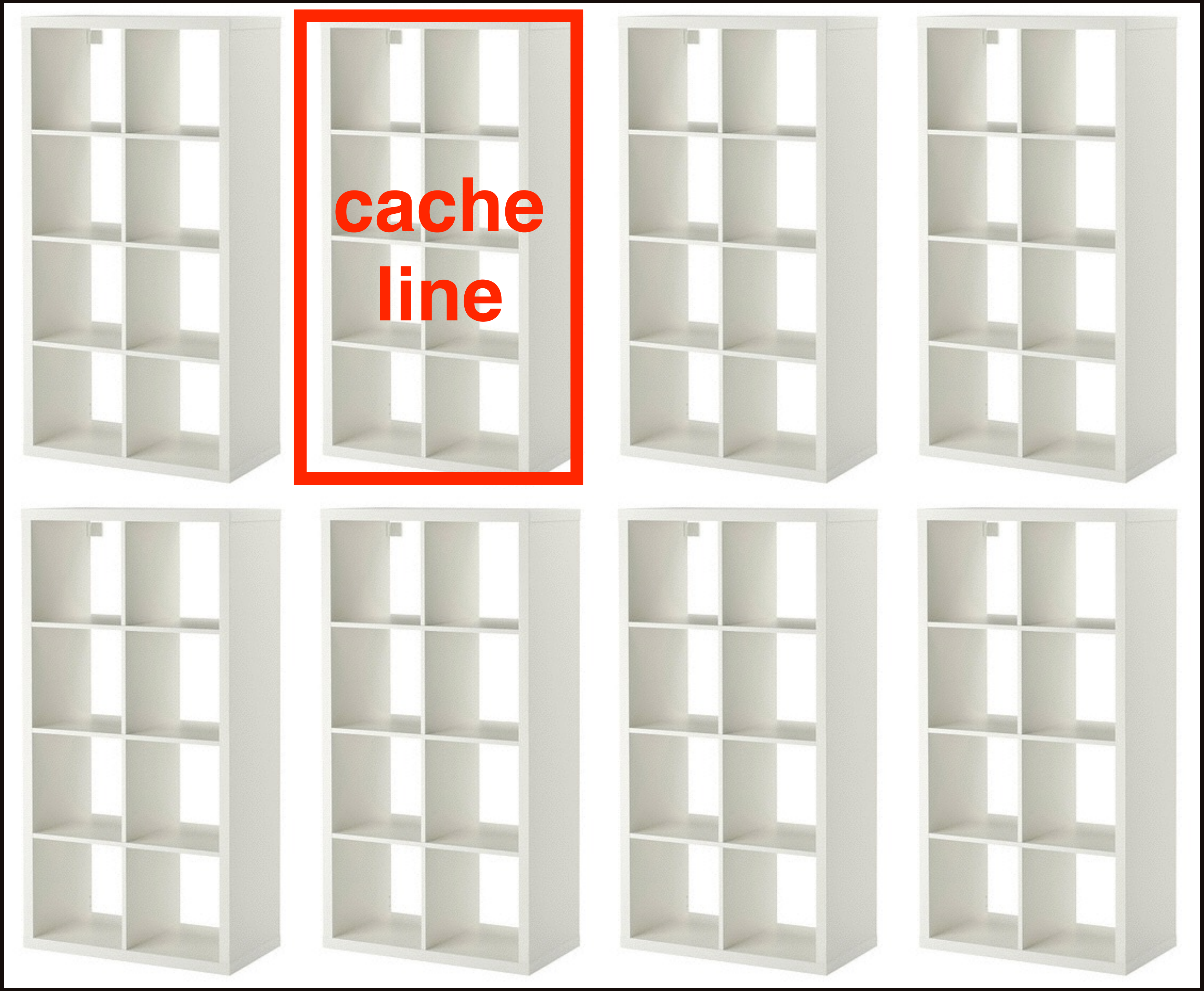
Alignment check

```
template <typename T>
constexpr bool is_aligned (T* ptr, size_t align = alignof (T))
{
    return uintptr_t (ptr) % align == 0;
}

//...or, with checking whether align is power of 2:

template <typename T>
constexpr bool is_aligned (T* ptr, size_t align = alignof (T))
{
    if (! ((align != 0) && ! (align & (align - 1))))
        return false;

    return (uintptr_t (ptr) & (align - 1)) == 0;
}
```

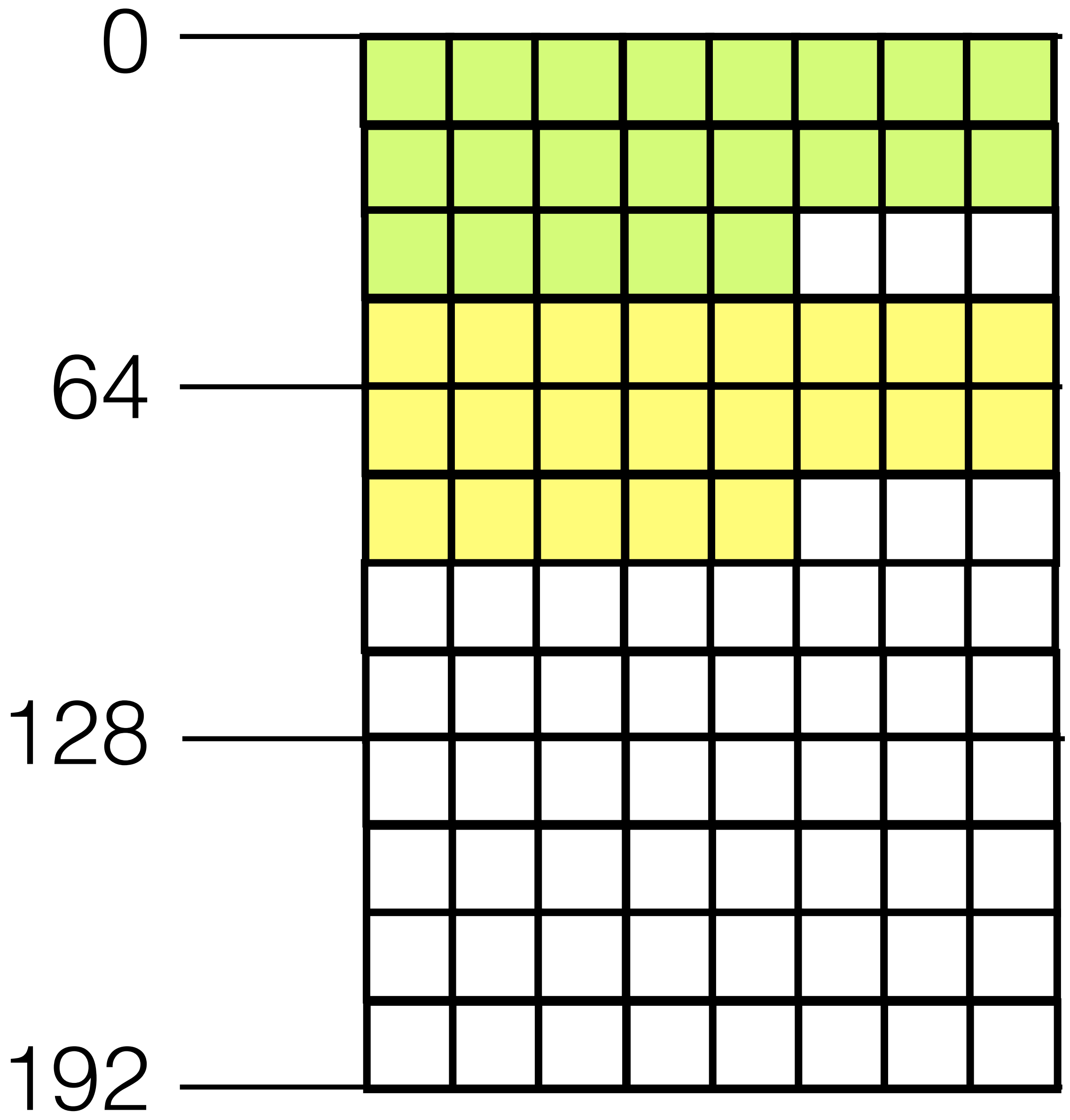
🙁

ALIGNOF

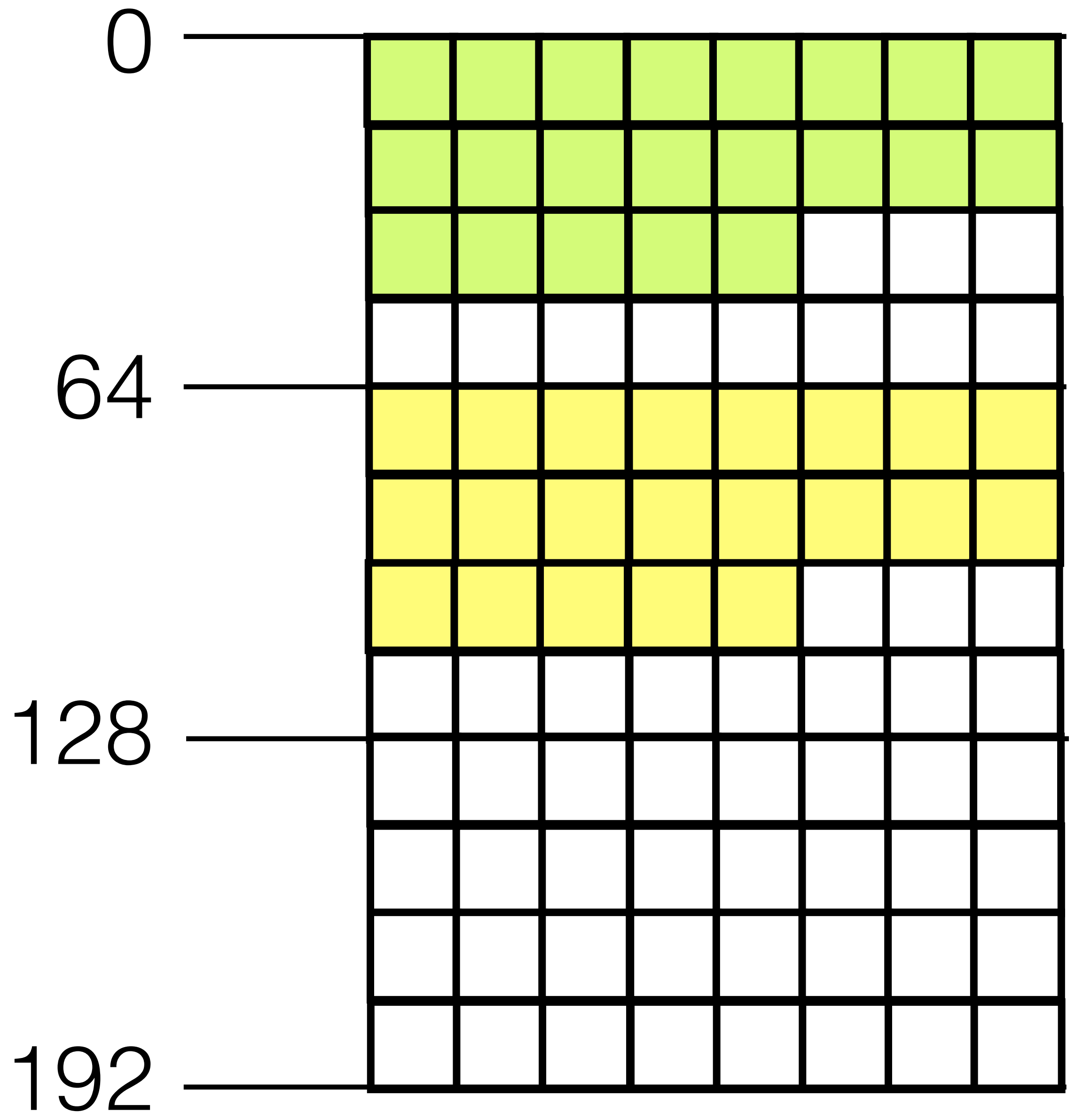



cache
line

Align to cache lines



Align to cache lines

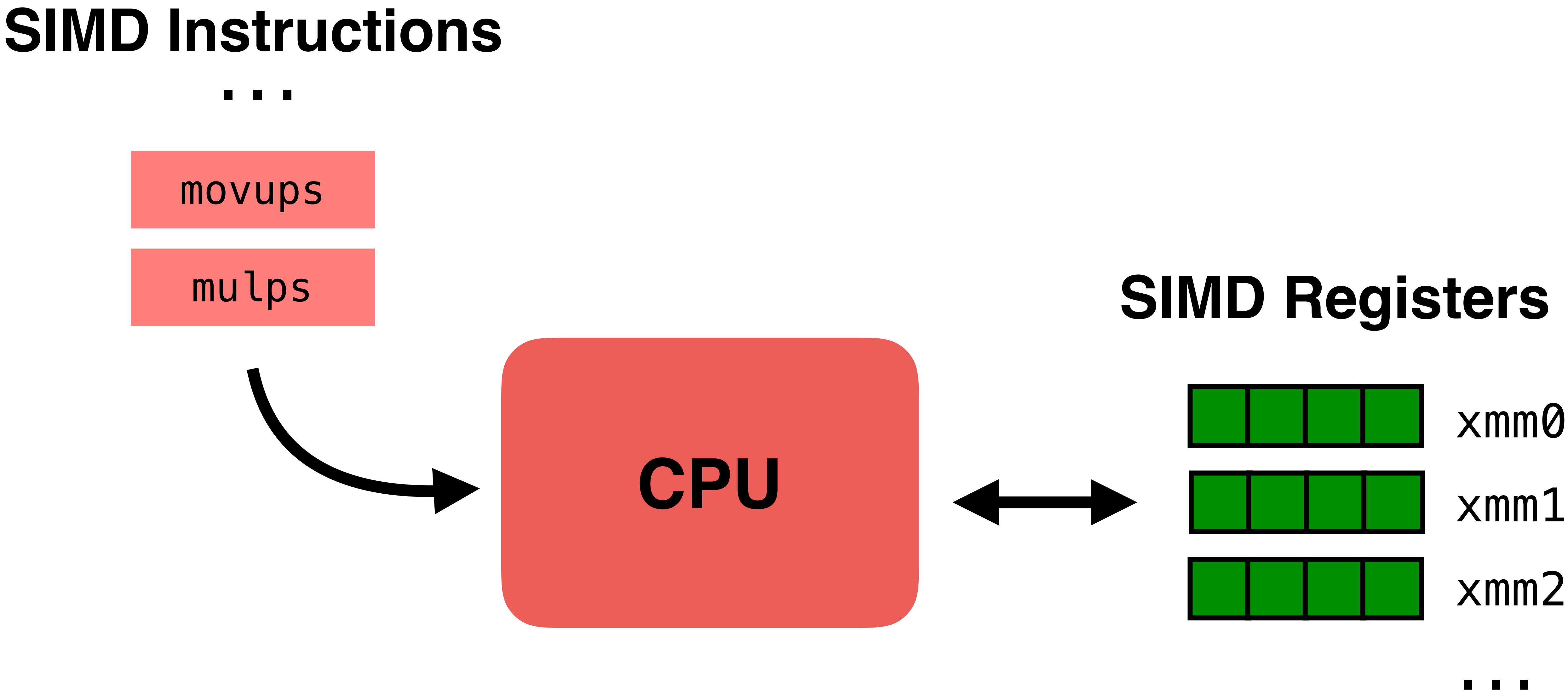


`cacheline_size_t`

🙁

SIMD

Single instruction, multiple data



SIMD flavours

SIMD flavour	Architecture	Registers	Register size
SSE (SSE2/3/4/4.1/4.2)	x86	xmm0, … xmm7	16 Bytes
AVX, AVX2	x86	ymm0, … ymm15	32 Bytes
AVX512	x86	zmm0, … zmm31	64 Bytes
NEON	ARM	q0, … q15 d0, … d31	16 Bytes

SIMD and you

Auto-vectoriser

```
-O3
```

```
for (size_t i = 0; i < size; ++i)
       a[i] *= f;
```

SIMD library

JUCE, Boost.SIMD, veclib, ...

```
Vec4<float> a;
a *= f;
```

SSE intrinsics

```
#include <emmintrin.h>

__m128 xmm0 = _mm_load_ps (a);
__m128 xmm1 = _mm_load1_ps (f);

xmm0 = _mm_mul_ps (xmm0, xmm1);
_mm_store_ps (xmm0, a);
```

Inline assembly

```
__asm
{
   mov    eax,  a
   mov    ebx,  f
   movups xmm0, [eax]
   movups xmm1, [ebx]
   mulps  xmm1, xmm0
   ...
}
```

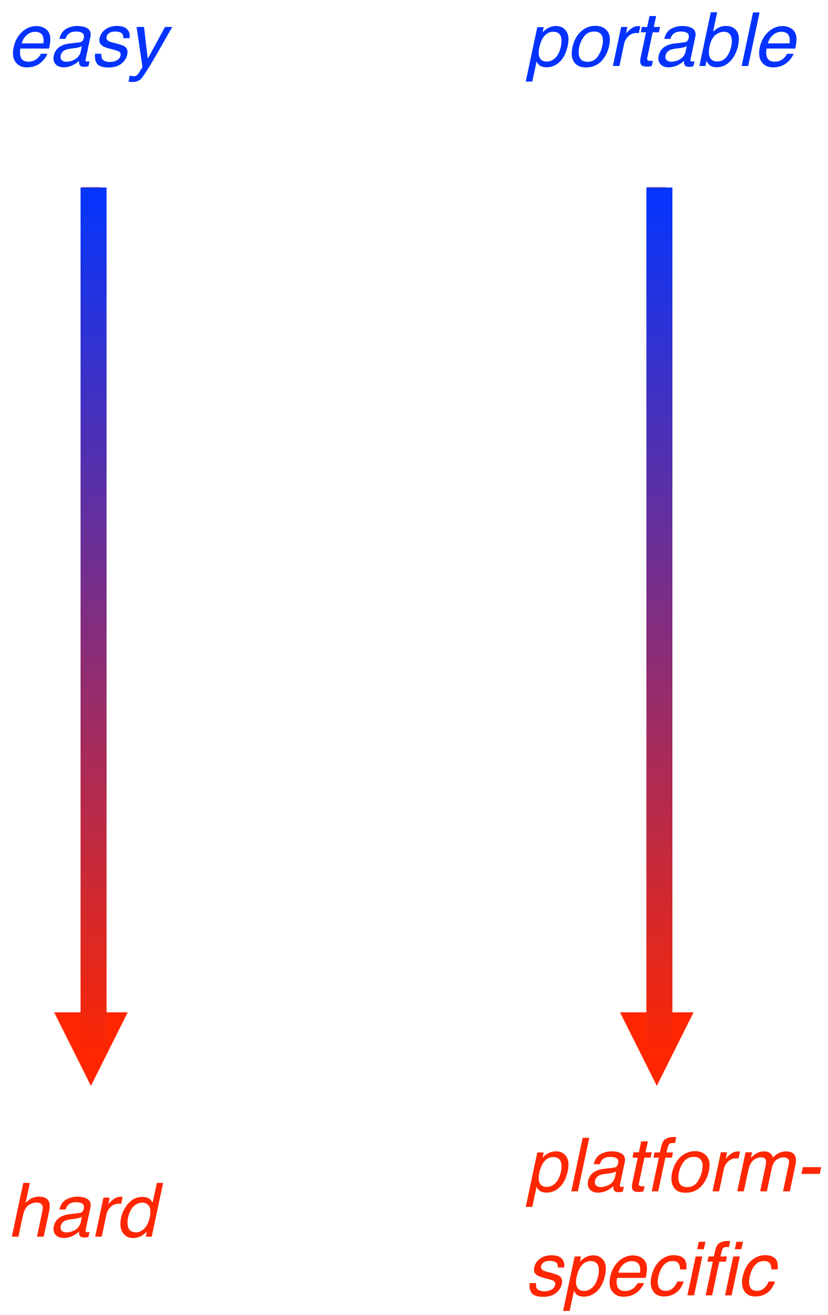
SIMD and you

Auto-vectoriser

SIMD library

SSE intrinsics

Inline assembly



easy

hard

portable

platform-specific

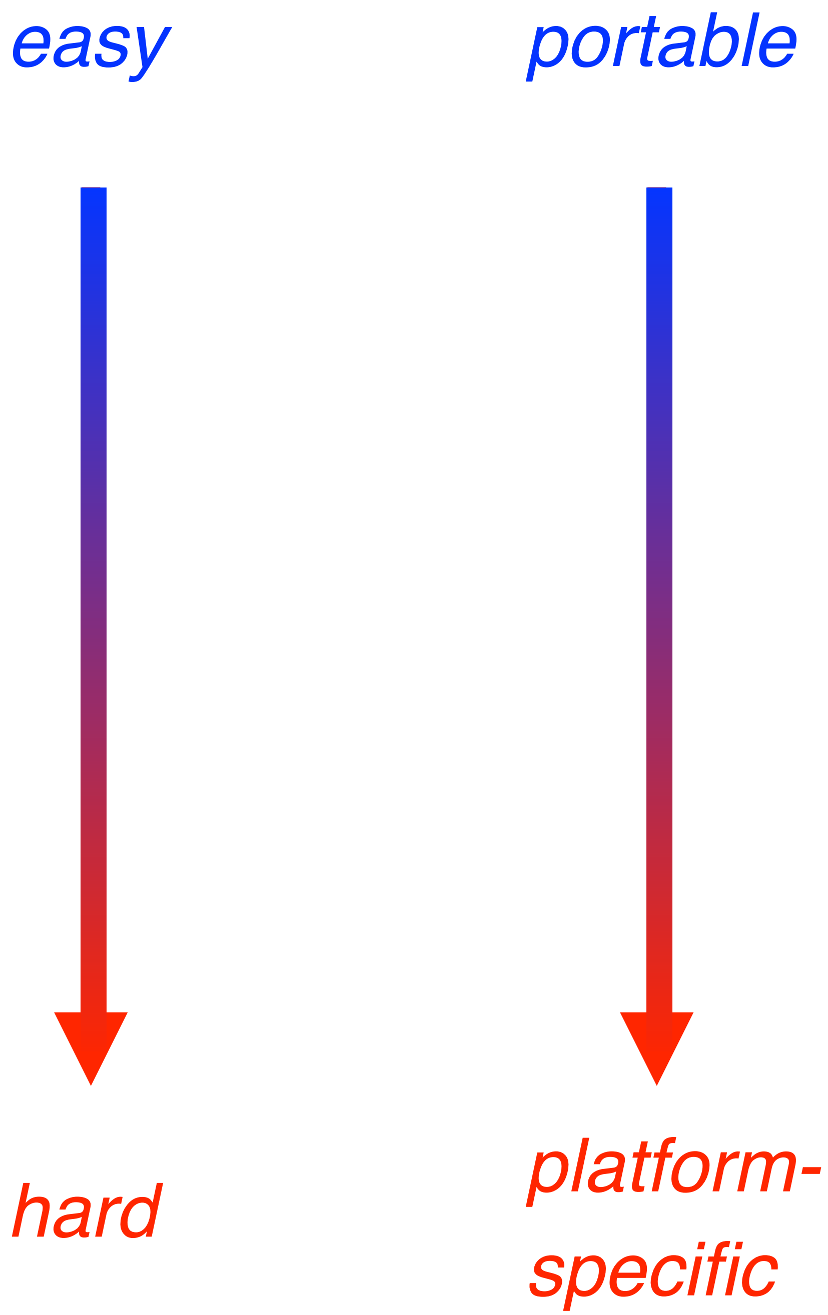
SIMD and you

Auto-vectoriser

SIMD library

SSE intrinsics

Inline assembly

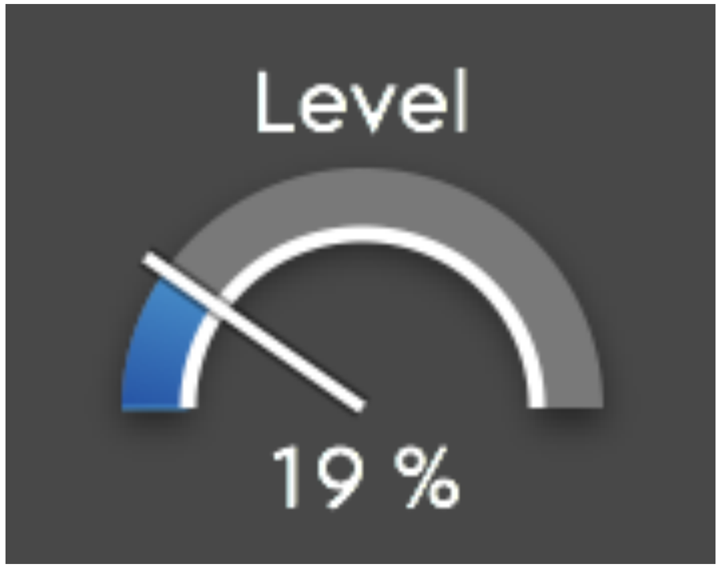


easy

hard

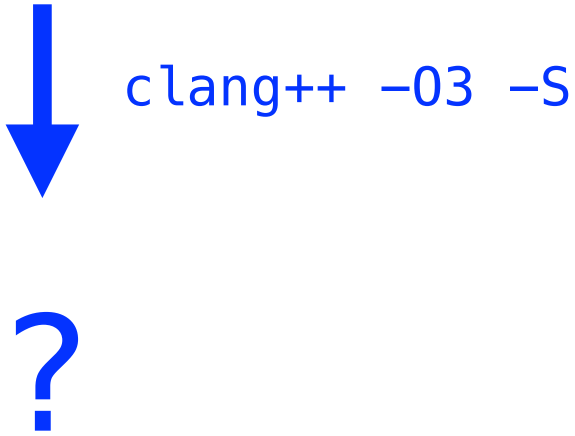
portable

platform-specific


Level
19 %

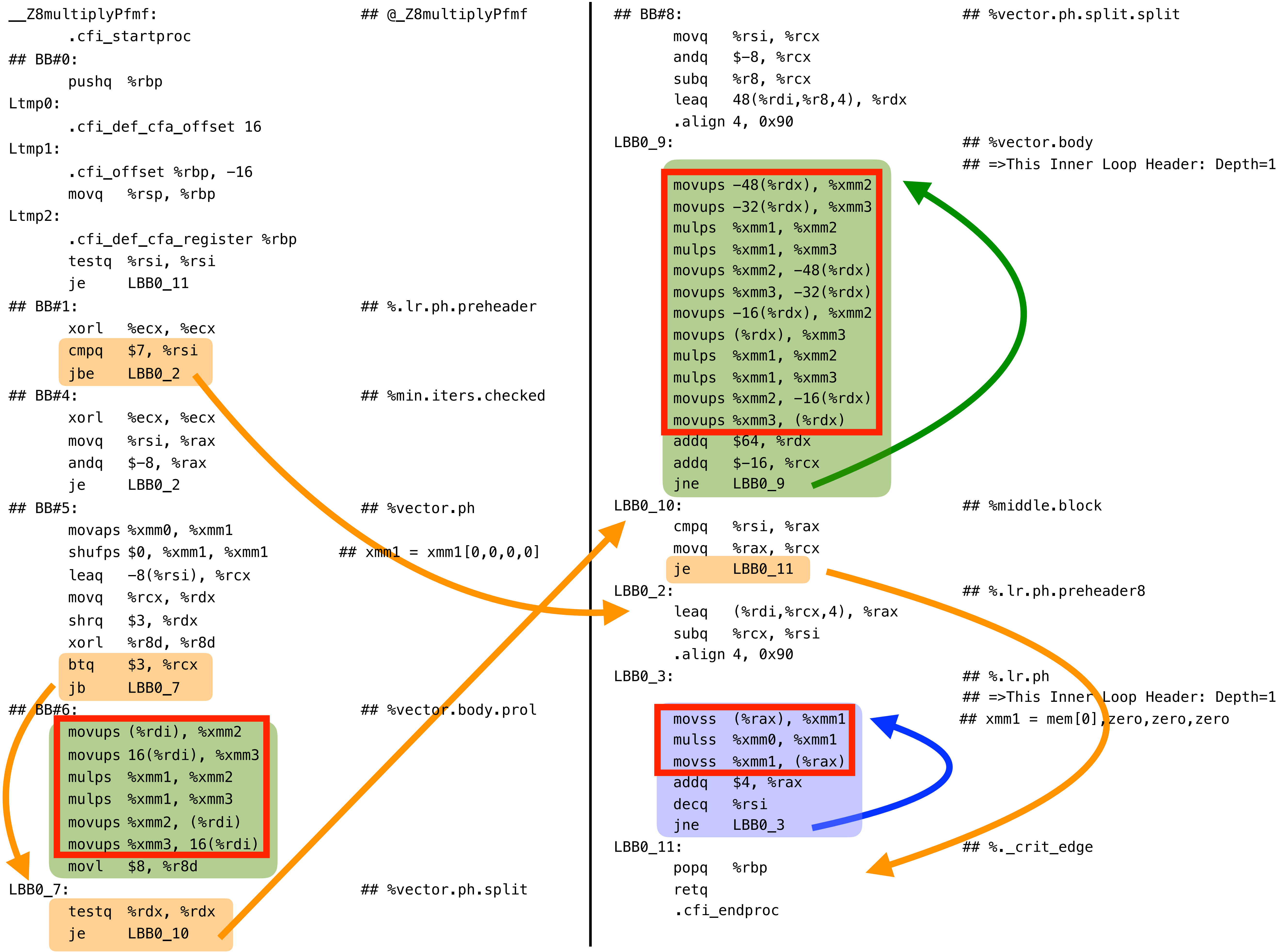
```
void multiply (float* buffer, size_t size, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}
```

```
void multiply (float* buffer, size_t size, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}
```



clang++ -O3 -S

?



```
__Z8multiplyPfmf:                          ## @_Z8multiplyPfmf
        .cfi_startproc
## BB#0:
        pushq  %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, -16
        movq   %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        testq  %rsi, %rsi
        je     LBB0_11
## BB#1:                                   ## %.lr.ph.preheader
        xorl   %ecx, %ecx
        cmpq   $7, %rsi
        jbe    LBB0_2
## BB#4:                                   ## %min.iters.checked
        xorl   %ecx, %ecx
        movq   %rsi, %rax
        andq   $-8, %rax
        je     LBB0_2
## BB#5:                                   ## %vector.ph
        movaps %xmm0, %xmm1
        shufps $0, %xmm1, %xmm1         ## xmm1 = xmm1[0,0,0,0]
        leaq   -8(%rsi), %rcx
        movq   %rcx, %rdx
        shrq   $3, %rdx
        xorl   %r8d, %r8d
        btq    $3, %rcx
        jb     LBB0_7
## BB#6:                                   ## %vector.body.prol
        movups (%rdi), %xmm2
        movups 16(%rdi), %xmm3
        mulps  %xmm1, %xmm2
        mulps  %xmm1, %xmm3
        movups %xmm2, (%rdi)
        movups %xmm3, 16(%rdi)
        movl   $8, %r8d
LBB0_7:                                    ## %vector.ph.split
        testq  %rdx, %rdx
        je     LBB0_10
```

```
## BB#8:                                   ## %vector.ph.split.split
        movq   %rsi, %rcx
        andq   $-8, %rcx
        subq   %r8, %rcx
        leaq   48(%rdi,%r8,4), %rdx
        .align 4, 0x90
LBB0_9:                                    ## %vector.body
                                           ## =>This Inner Loop Header: Depth=1
        movups -48(%rdx), %xmm2
        movups -32(%rdx), %xmm3
        mulps  %xmm1, %xmm2
        mulps  %xmm1, %xmm3
        movups %xmm2, -48(%rdx)
        movups %xmm3, -32(%rdx)
        movups -16(%rdx), %xmm2
        movups (%rdx), %xmm3
        mulps  %xmm1, %xmm2
        mulps  %xmm1, %xmm3
        movups %xmm2, -16(%rdx)
        movups %xmm3, (%rdx)
        addq   $64, %rdx
        addq   $-16, %rcx
        jne    LBB0_9
LBB0_10:                                   ## %middle.block
        cmpq   %rsi, %rax
        movq   %rax, %rcx
        je     LBB0_11
LBB0_2:                                    ## %.lr.ph.preheader8
        leaq   (%rdi,%rcx,4), %rax
        subq   %rcx, %rsi
        .align 4, 0x90
LBB0_3:                                    ## %.lr.ph
                                           ## =>This Inner Loop Header: Depth=1
                                          ## xmm1 = mem[0],zero,zero,zero
        movss  (%rax), %xmm1
        mulss  %xmm0, %xmm1
        movss  %xmm1, (%rax)
        addq   $4, %rax
        decq   %rsi
        jne    LBB0_3
LBB0_11:                                   ## %._crit_edge
        popq   %rbp
        retq
        .cfi_endproc
```

```
void multiply (float* buffer, size_t size, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}
```

↓

```
template <size_t size>
void multiply (float* buffer, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}

template void multiply<16384> (float* buffer, float factor);
```

↓ `clang++ -O3 -S`

?

```
      .cfi_startproc
## BB#0:                                ##
%min.iters.checked
      pushq %rbp
Ltmp0:
      .cfi_def_cfa_offset 16
Ltmp1:
      .cfi_offset %rbp, -16
      movq  %rsp, %rbp
Ltmp2:
      .cfi_def_cfa_register %rbp
      shufps      $0, %xmm0, %xmm0        ## xmm0 =
xmm0[0,0,0,0]
      xorl  %eax, %eax
      .align      4, 0x90
LBB0_1:                                 ## %vector.body
                                        ## =>This Inner
Loop Header: Depth=1
      movups      (%rdi,%rax,4), %xmm1
      movups      16(%rdi,%rax,4), %xmm2
      mulps %xmm0, %xmm1
      mulps %xmm0, %xmm2
      movups      %xmm1, (%rdi,%rax,4)
      movups      %xmm2, 16(%rdi,%rax,4)
      movups      32(%rdi,%rax,4), %xmm1
      movups      48(%rdi,%rax,4), %xmm2
      mulps %xmm0, %xmm1
      mulps %xmm0, %xmm2
      movups      %xmm1, 32(%rdi,%rax,4)
      movups      %xmm2, 48(%rdi,%rax,4)
      addq  $16, %rax
      cmpq  $16384, %rax            ## imm = 0x4000
      jne   LBB0_1
## BB#2:                                ## %middle.block
      popq  %rbp
      retq
      .cfi_endproc
```

```
void multiply (float* buffer, size_t size, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}
```

↓

```
template <size_t size>
void multiply (float* buffer, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}

template void multiply<128> (float* buffer, float factor);
```

↓ `clang++ -O3 -S`

?

```
__Z8multiplyILm128EEvPff:               ##
@_Z8multiplyILm128EEvPff
        .cfi_startproc
## BB#0:                                ## %min.iters.checked
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        shufps  $0, %xmm0, %xmm0        ## xmm0 = xmm0[0,0,0,0]
        movups  (%rdi), %xmm1
        movups  16(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, (%rdi)
        movups  %xmm2, 16(%rdi)
        movups  32(%rdi), %xmm1
        movups  48(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 32(%rdi)
        movups  %xmm2, 48(%rdi)
        movups  64(%rdi), %xmm1
        movups  80(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 64(%rdi)
        movups  %xmm2, 80(%rdi)
        movups  96(%rdi), %xmm1
        movups  112(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 96(%rdi)
        movups  %xmm2, 112(%rdi)
        movups  128(%rdi), %xmm1
```

```
        movups  144(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 128(%rdi)
        movups  %xmm2, 144(%rdi)
        movups  160(%rdi), %xmm1
        movups  176(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 160(%rdi)
        movups  %xmm2, 176(%rdi)
        movups  192(%rdi), %xmm1
        movups  208(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 192(%rdi)
        movups  %xmm2, 208(%rdi)
        movups  224(%rdi), %xmm1
        movups  240(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 224(%rdi)
        movups  %xmm2, 240(%rdi)
        movups  256(%rdi), %xmm1
        movups  272(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 256(%rdi)
        movups  %xmm2, 272(%rdi)
        movups  288(%rdi), %xmm1
        movups  304(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 288(%rdi)
        movups  %xmm2, 304(%rdi)
        movups  320(%rdi), %xmm1
        movups  336(%rdi), %xmm2
```

```
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 320(%rdi)
        movups  %xmm2, 336(%rdi)
        movups  352(%rdi), %xmm1
        movups  368(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 352(%rdi)
        movups  %xmm2, 368(%rdi)
        movups  384(%rdi), %xmm1
        movups  400(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 384(%rdi)
        movups  %xmm2, 400(%rdi)
        movups  416(%rdi), %xmm1
        movups  432(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 416(%rdi)
        movups  %xmm2, 432(%rdi)
        movups  448(%rdi), %xmm1
        movups  464(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 448(%rdi)
        movups  %xmm2, 464(%rdi)
        movups  480(%rdi), %xmm1
        movups  496(%rdi), %xmm2
        mulps   %xmm0, %xmm1
        mulps   %xmm0, %xmm2
        movups  %xmm1, 480(%rdi)
        movups  %xmm2, 496(%rdi)
        popq    %rbp
        retq
        .cfi_endproc
```

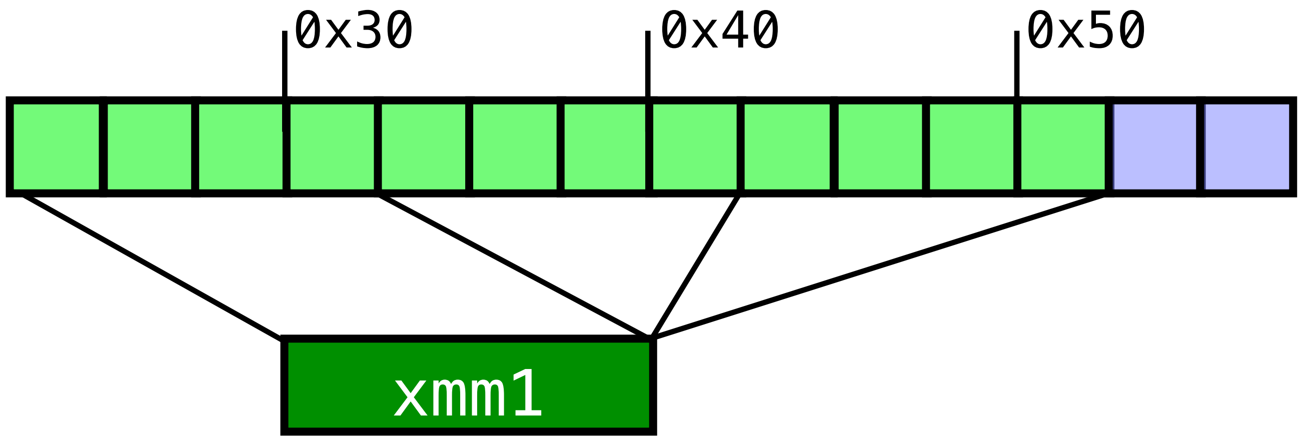
```
void multiply (float* buffer, size_t size, float factor)
{
    for (size_t i = 0; i < size; ++i)
        buffer[i] *= factor;
}
```

↓ `g++ -O3 -S`

?

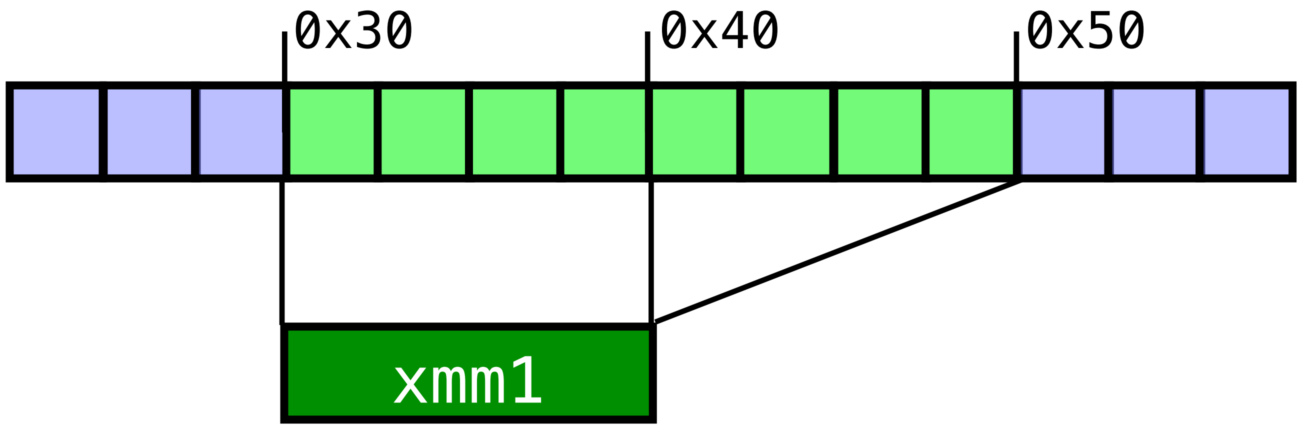
```
__Z8multiplyILm16384EEvPff:
LFB1:
      movq  %rdi, %rax
      andl  $15, %eax
      shrq  $2, %rax
      negq  %rax
      andl  $3, %eax
      je    L8
      movss (%rdi), %xmm1
      cmpq  $1, %rax
      mulss %xmm0, %xmm1
      movss %xmm1, (%rdi)
      je    L9
      movss 4(%rdi), %xmm1
      cmpq  $2, %rax
      mulss %xmm0, %xmm1
      movss %xmm1, 4(%rdi)
      je    L10
      movss 8(%rdi), %xmm1
      movl  $16381, %r9d
      movl  $3, %r8d
      mulss %xmm0, %xmm1
      movss %xmm1, 8(%rdi)
L3:
      movl  $16384, %r10d
      movl  $16380, %r11d
      movl  $4095, %esi
      subq  %rax, %r10
L2:
      leaq  (%rdi,%rax,4), %rcx
      movaps      %xmm0, %xmm2
      xorl  %eax, %eax
      xorl  %edx, %edx
      shufps      $0, %xmm2, %xmm2
      .align 4,0x90
```

```
L5:
      movaps      (%rcx,%rax), %xmm1
      addq  $1, %rdx
      mulps %xmm2, %xmm1
      movaps      %xmm1, (%rcx,%rax)
      addq  $16, %rax
      cmpq  %rsi, %rdx
      jb    L5
      leaq  (%r8,%r11), %rax
      movq  %r9, %rdx
      subq  %r11, %rdx
      cmpq  %r11, %r10
      je    L1
      leaq  (%rdi,%rax,4), %rcx
      cmpq  $1, %rdx
      movss (%rcx), %xmm1
      mulss %xmm0, %xmm1
      movss %xmm1, (%rcx)
      leaq  1(%rax), %rcx
      je    L1
      leaq  (%rdi,%rcx,4), %rcx
      addq  $2, %rax
      cmpq  $2, %rdx
      movss (%rcx), %xmm1
      mulss %xmm0, %xmm1
      movss %xmm1, (%rcx)
      je    L1
      leaq  (%rdi,%rax,4), %rax
      mulss (%rax), %xmm0
      movss %xmm0, (%rax)
      ret
      .align 4,0x90
L1:
      ret
```



movups

(unaligned load/store from/to SSE registers)



movaps

(aligned load/store from/to SSE registers)

multiplyAdd benchmark (SSE)

```
void multiplyAdd (float* buffer1, float* buffer2,
                  float factor, size_t size)
{
    for (size_t i = 0; i < size; ++i)
        buffer1[i] += buffer2[i] * factor;
}
```

multiplyAdd benchmark (SSE)

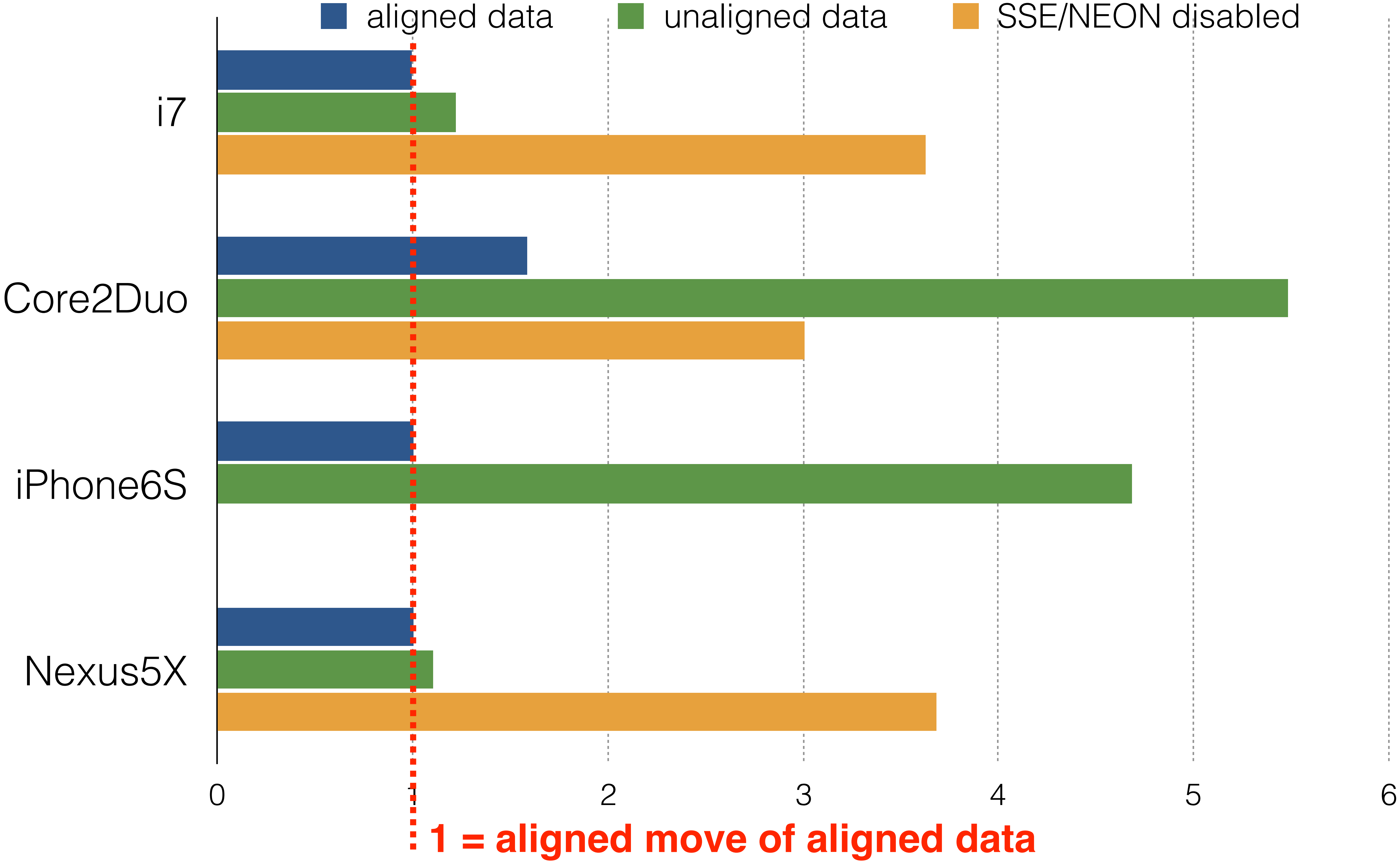
```
void multiplyAdd (float* buffer1, float* buffer2, float factor, std::size_t size)
{
    __m128 f = _mm_load1_ps (&factor);

    for (std::size_t i = 0; i < size; i+=4)
    {
        __m128 b1 = _mm_loadu_ps (buffer1 + i);    // or _mm_load_ps for aligned load
        __m128 b2 = _mm_loadu_ps (buffer2 + i);

        b2 = _mm_mul_ps (b2, f);
        b1 = _mm_add_ps (b1, b2);

        _mm_storeu_ps (buffer1 + i, b1);           // or _mm_store_ps for aligned load
    }
}


int main()
{
    const int size = 16384;

    auto* b1 = new float[size];
    auto* b2 = new float[size];

    // Warm the cache.
    std::fill (b2, b2 + size, 0.2f);
    std::fill (b1, b1 + size, 1.0f);

    BENCHMARK_START(loop, 100000)

    multiplyAdd (b1, b2, 0.00001f, size);       // or multiplyAdd (b1 + 1, b2 + 2, 0.00001f, size - 4);
                                                // for non-SSE-aligned data
    BENCHMARK_STOP(loop)

    std::cout << b1[0] << std::endl;
}
```

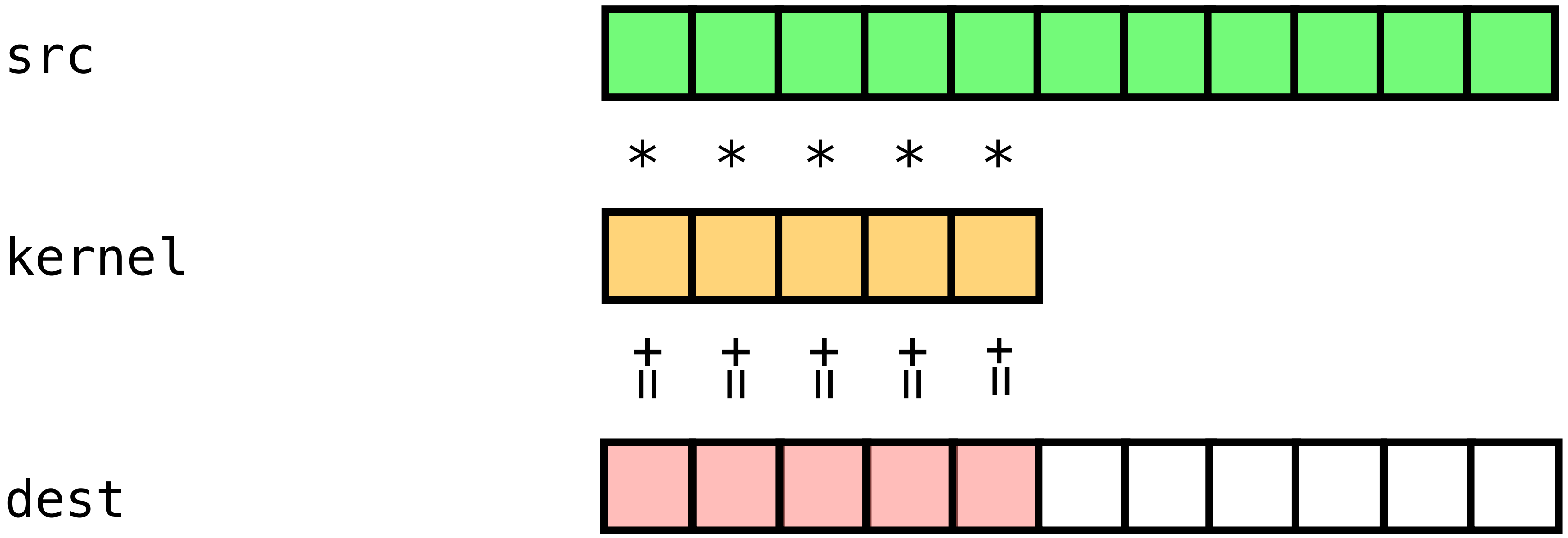
multiplyAdd benchmark (NEON)

```
void multiplyAdd (float* buffer1, float* buffer2, float factor, std::size_t size)
{
    float32x4_t f = vld1q_f32 (&factor);

    for (std::size_t i = 0; i < size; i+=4)
    {
        float32x4_t b1 = vld1q_f32 (buffer1 + i);
        float32x4_t b2 = vld1q_f32 (buffer2 + i);

        b2 = vmulq_f32 (b2, f);
        b1 = vaddq_f32 (b1, b2);

        vst1q_f32 (buffer1 + i, b1);
    }
}


int main()
{
    const int size = 16384;

    auto* b1 = new float[size];
    auto* b2 = new float[size];

    // Warm the cache.
    std::fill (b2, b2 + size, 0.2f);
    std::fill (b1, b1 + size, 1.0f);

    BENCHMARK_START(loop, 100000)

    multiplyAdd (b1, b2, 0.00001f, size);       // or multiplyAdd (b1 + 1, b2 + 2, 0.00001f, size - 4);
                                                // for non-SSE-aligned data
    BENCHMARK_STOP(loop)

    std::cout << b1[0] << std::endl;
}
```

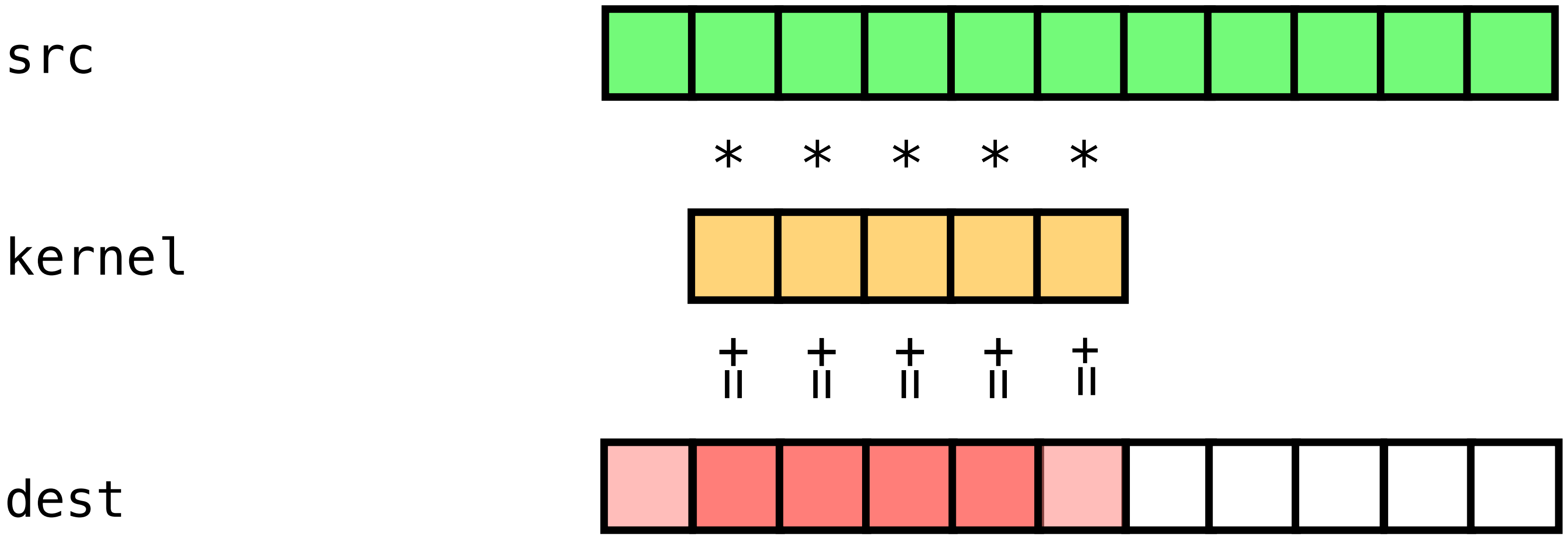
Cost of unaligned move for SSE/NEON registers

(using the simple multiplyAdd benchmark)

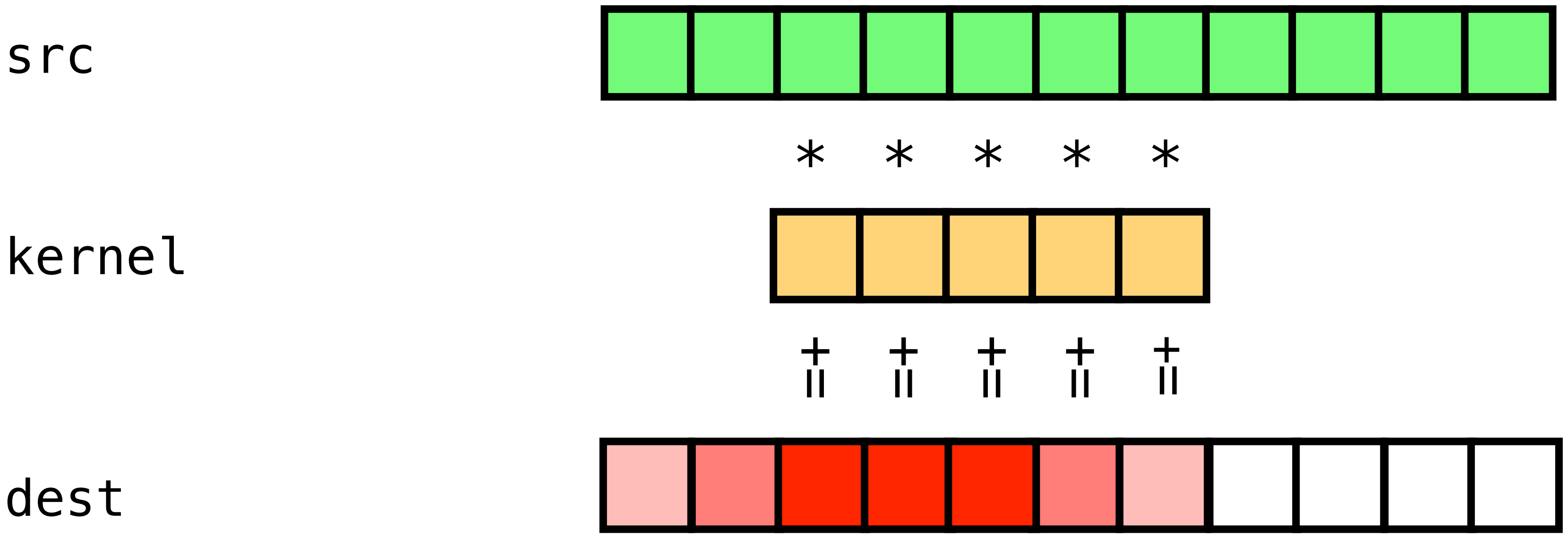




Convolution



Convolution

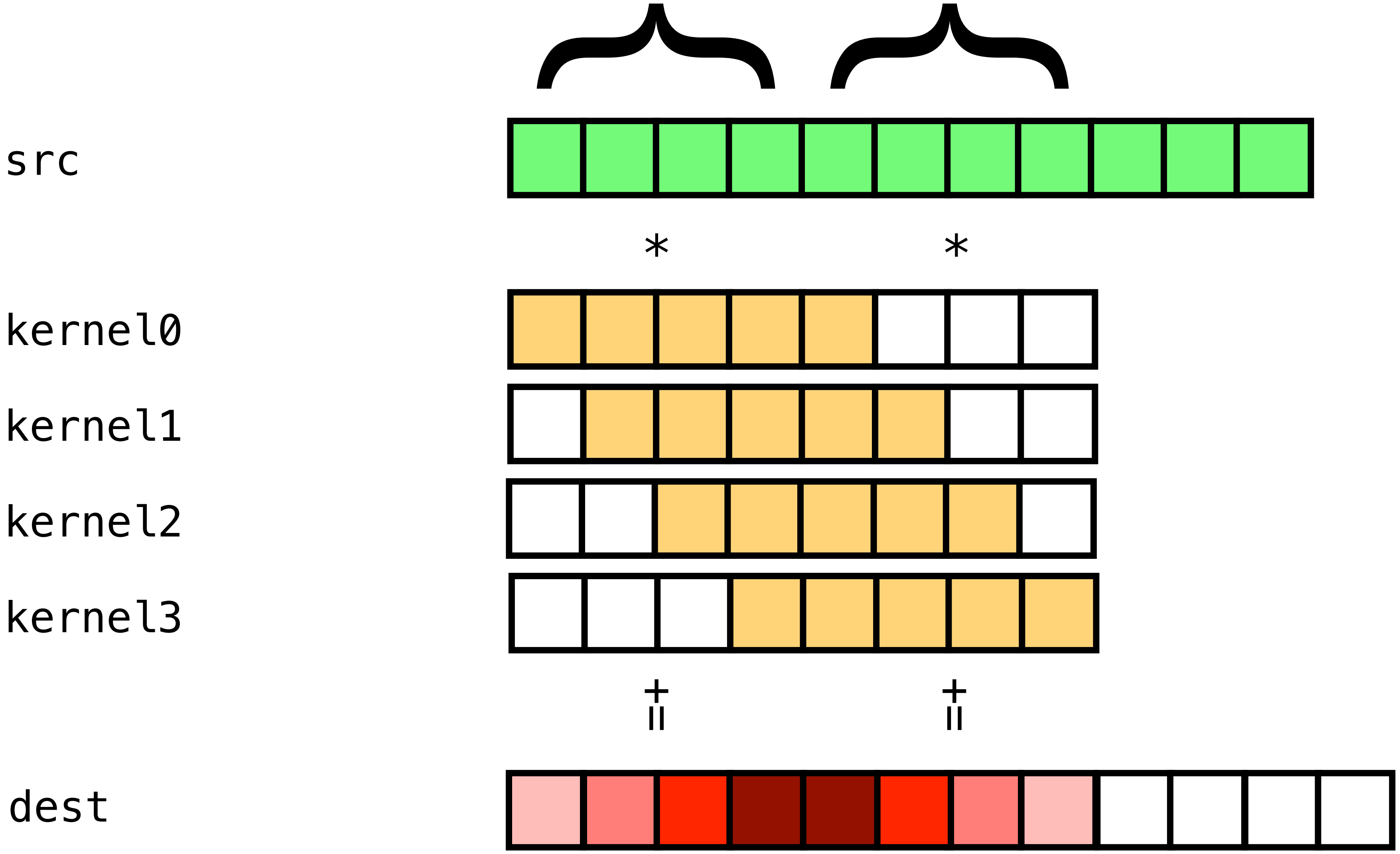


src

kernel

dest

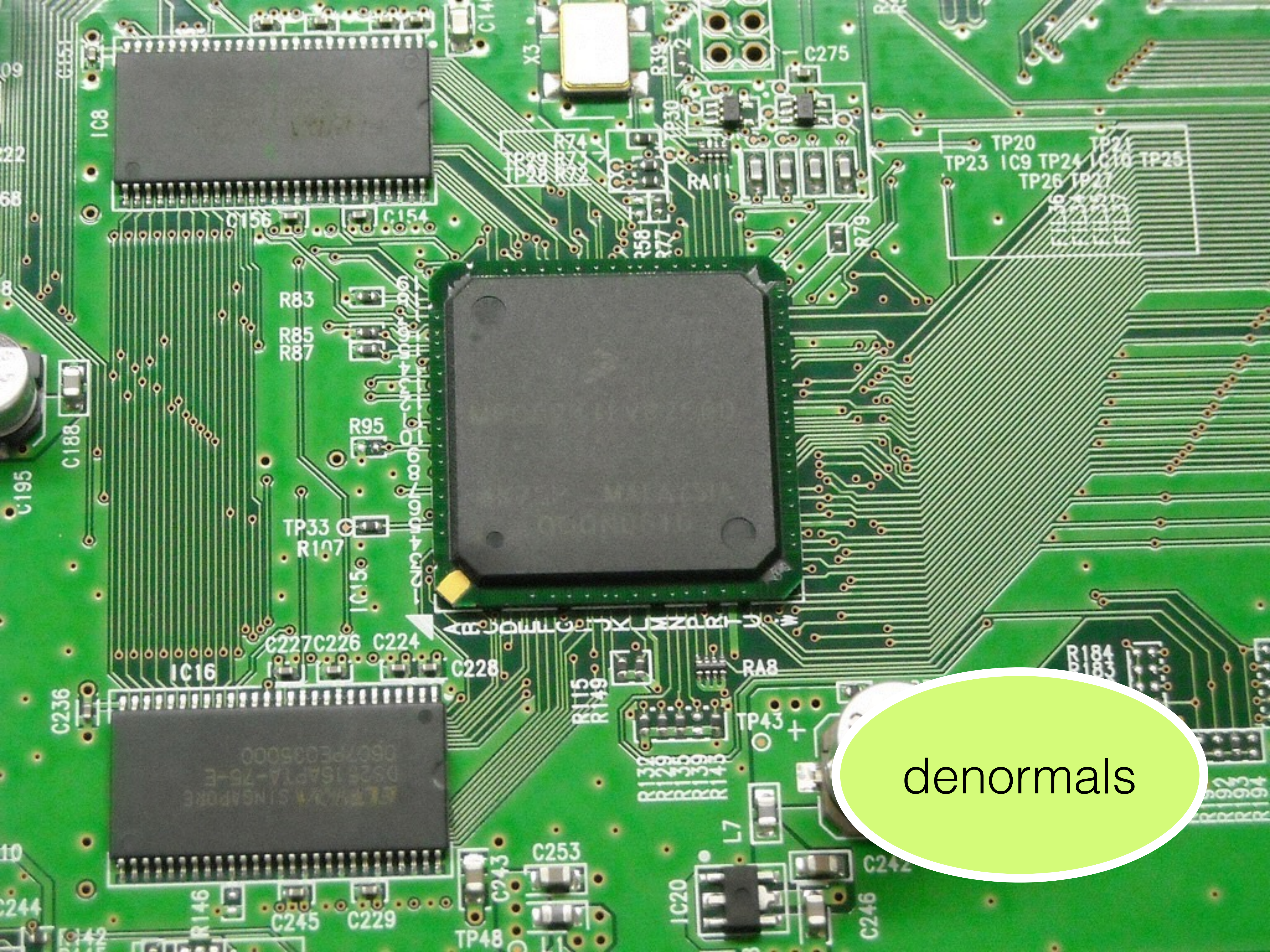
```
const float kernel[128];
float* src  = new float[65536];
float* dest = new float[65536];

for (int i = 0; i < 65536 - 128; ++i)
{
    float sum = 0.0f;

    for (int j = 0; j < 128; ++j)
        sum += src [i + j] * kernel[j];

    dest[i] = sum;
}
```


src
*
*
kernel0
kernel1
kernel2
kernel3
+
=
+
=
dest

```
void multiply (float* buffer, size_t size, float factor)
{
    // how to tell the compiler/optimiser at this point
    // that float* buffer is aligned to 16 bytes??

    for (size_t i = 0; i < size; ++i)
       buffer[i] *= factor;
}
```

```
// ...would be nice to have something like this:

template <typename T, size_t Align>
class aligned_ptr<T>;

// ...and then write something like this:

void multiply (aligned_ptr<float, 16> buffer, ...);
```

🙁


IC8
C151
C156
C154
R74
TP29 R73
TP28 R72
X3
C275
TP30
RA11
R58
R77
R79
TP20
TP21
TP23 IC9 TP24 IC10 TP25
TP26 TP27
R83
R85
R87
R95
TP33
R107
IC15
C188
C195
C227 C226 C224
C228
IC16
C236
R115
R149
RA8
TP43
R184
R183
L7
C253
C243
IC20
C242
C246
C245
C229
R146
C244
C142
TP48

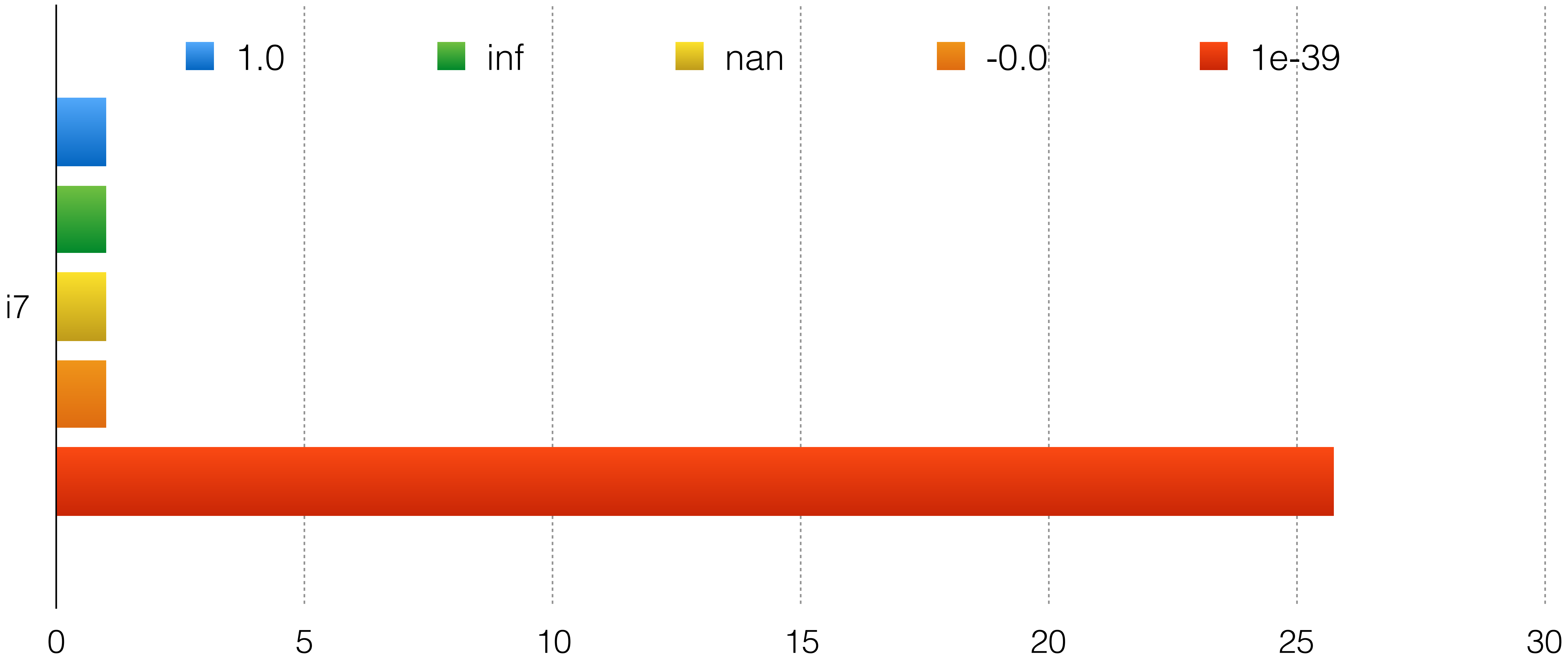
denormals

Crunch some floating point numbers!

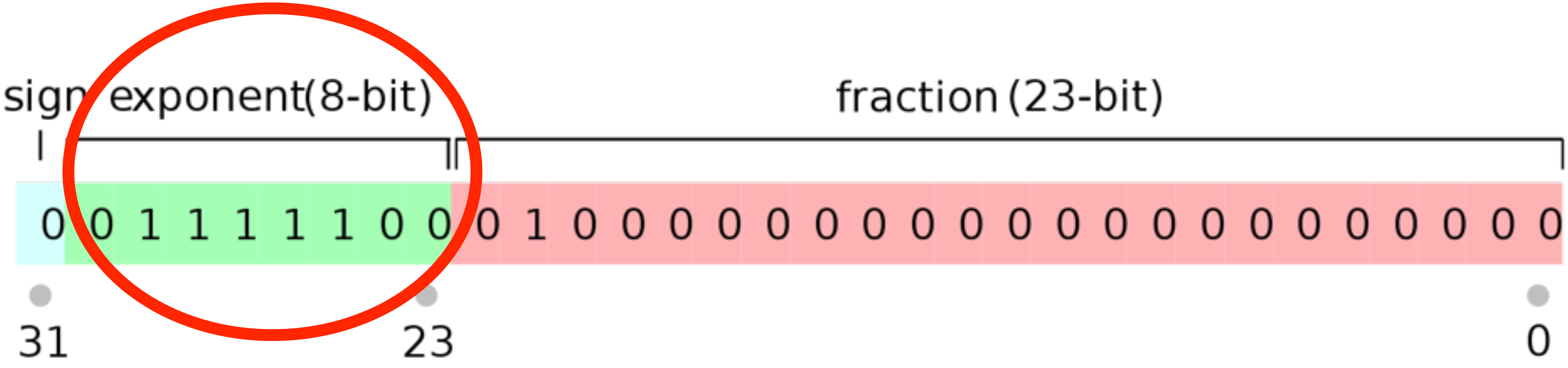
```
float crunchSomeNumbers (float x)
{
    float sum = 0.0f;

    BENCHMARK_START(crunch,100000)
    {
        for (int i = 0; i < 10000; ++i)
            x *= 0.999f;

        sum += x;
    }
    BENCHMARK_STOP(crunch)

    return sum;
}
```

```
x = 1.0;

x = 1.0 / 0.0;

x = 0.0 / 0.0;

x = -0.0;

x = 1e-39;
```

Number crunching benchmark



Denormals



- Exponent = 1111111:
  - fraction is zero —> +inf, -inf
  - fraction is non-zero —> NaNs
- Exponent between 1111110 and 00000001:
  - “normal” floats with implicit leading 1.
- Exponent = 0000000:
  - fraction is zero —> +0, -0
  - fraction is non-zero —> denormals

Check denormals support

```
std::numeric_limits<float>::min();        // 1.17549e-38
std::numeric_limits<float>::denorm_min(); // 1.4013e-45
std::numeric_limits<float>::has_denorm;   // std::denorm_present
```

Flush denormals to zero

```
#include <float.h>

// Flush denormals to zero, both operands and results
_controlfp_s (nullptr, _DN_FLUSH, _MCW_DN);

// Put denormal handling back to normal.
_controlfp_s (nullptr, _DN_SAVE, _MCW_DN);
```

🙁

Thank you!