# CPU design effects
# that can degrade performance of your programs

Jakub Beránek
jakub.beranek@vsb.cz

# whoami

- PhD student @ VSB-TUO, Ostrava, Czech Republic
- Research assistant @ IT4Innovations (HPC center)
- HPC, distributed systems, program optimization

# How do we get maximum performance?

- Select the right algorithm

# How do we get maximum performance?

- Select the right algorithm
- Use a low-overhead language

# How do we get maximum performance?

- Select the right algorithm
- Use a low-overhead language
- Compile properly

# How do we get maximum performance?

- Select the right algorithm
- Use a low-overhead language
- Compile properly
- **Tune to the underlying hardware**

# Why should we care?

- We write code for the C++ abstract machine

# Why should we care?

- We write code for the C++ abstract machine
- Intel CPUs fulfill the contract of this abstract machine

# Why should we care?

- We write code for the C++ abstract machine
- Intel CPUs fulfill the contract of this abstract machine
  - But inside they can do whatever they want

# Why should we care?

- We write code for the C++ abstract machine
- Intel CPUs fulfill the contract of this abstract machine
  - But inside they can do whatever they want
- Understanding CPU trade-offs can get us more performance

# C++ abstract machine example

```
void foo(int* arr, int count)
{
    for (int i = 0; i < count; i++)
    {
        arr[i]++;
    }
}
```

How fast are the individual array increments?

# Hardware effects

- Performance effects caused by a specific CPU/memory implementation

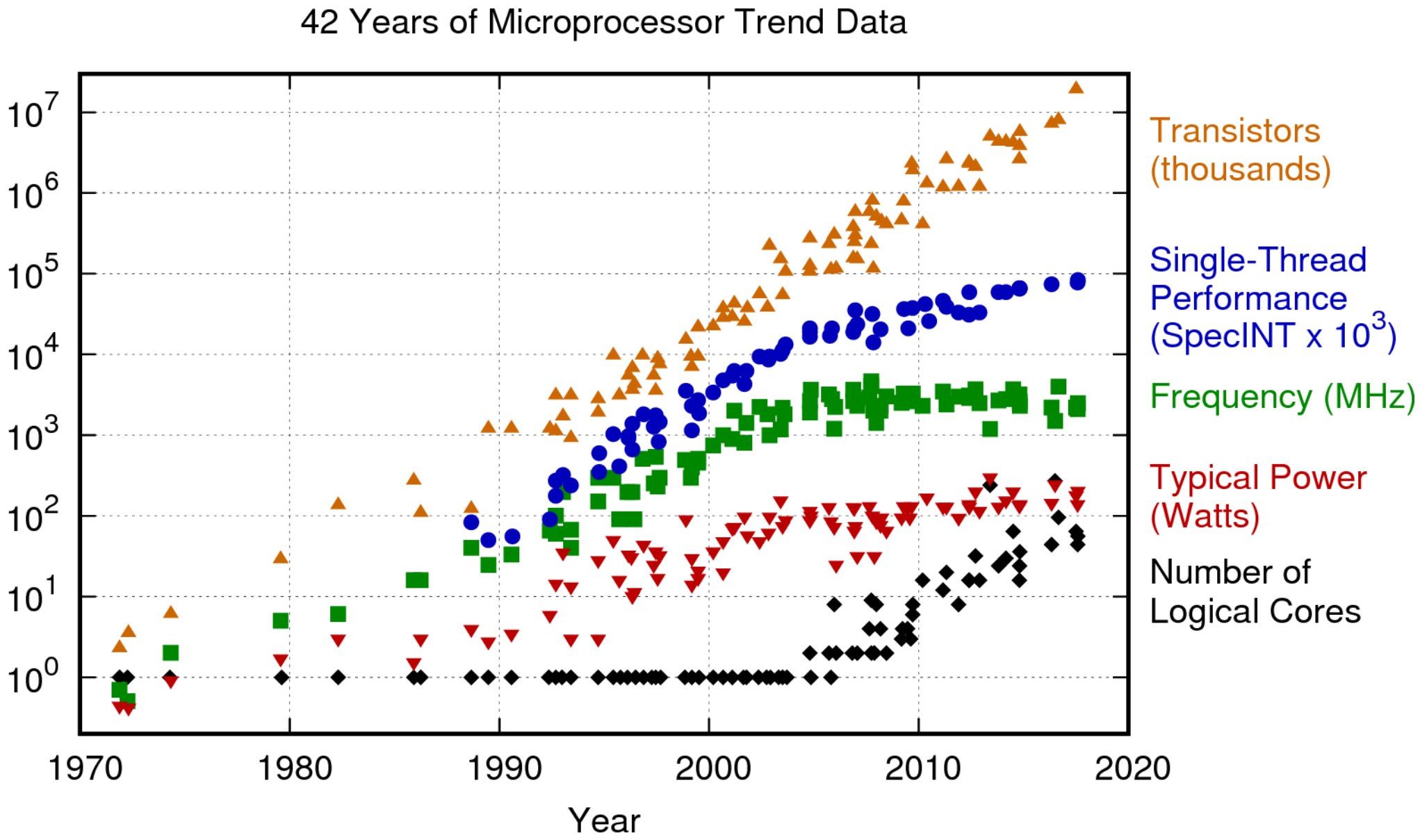
# Hardware effects

- Performance effects caused by a specific CPU/memory implementation
- Demonstrate some CPU/memory trade-off or assumption

# Hardware effects

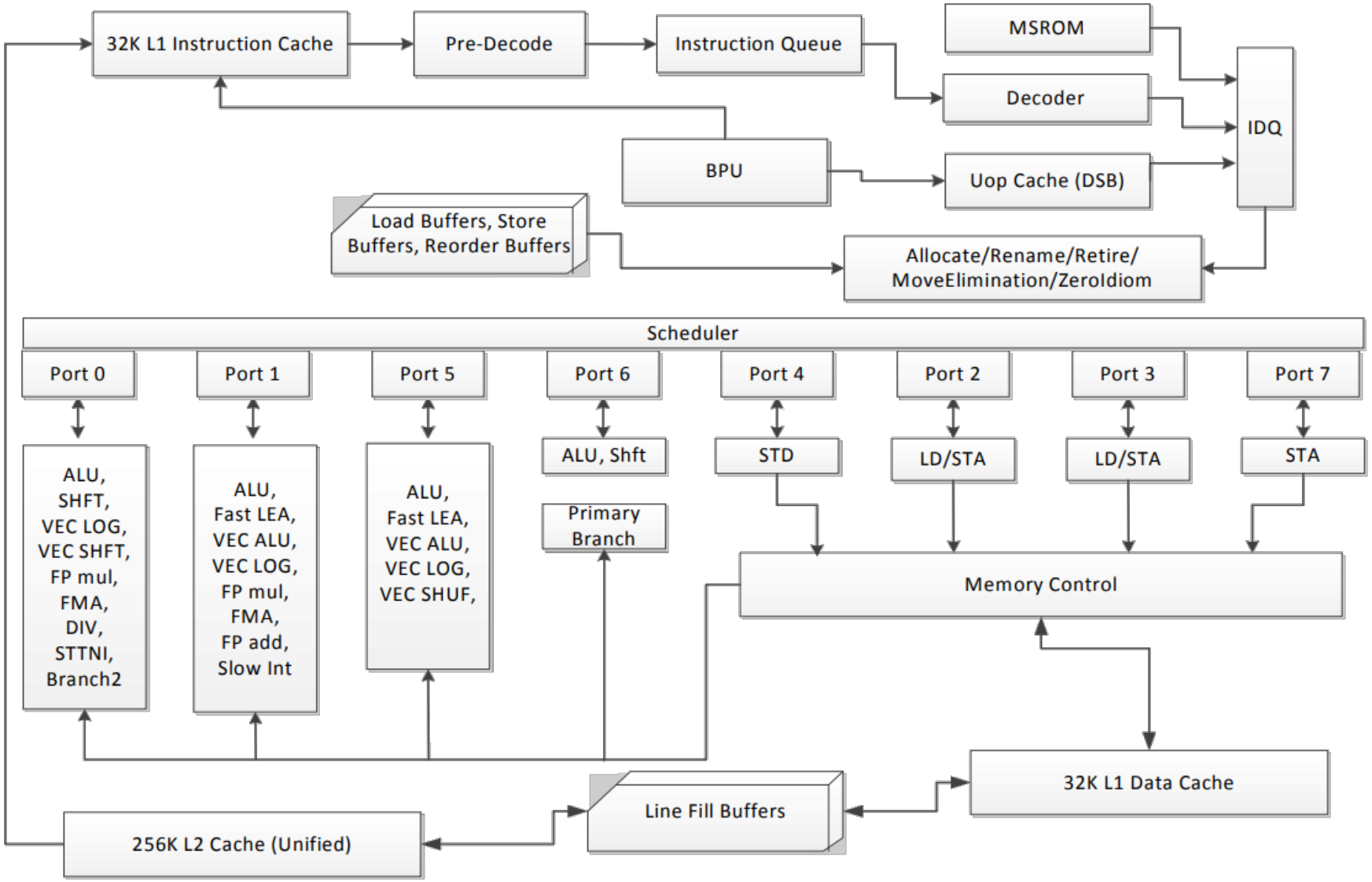
- Performance effects caused by a specific CPU/memory implementation
- Demonstrate some CPU/memory trade-off or assumption
- Impossible to predict from (C++) code alone

# Hardware is getting more and more complex



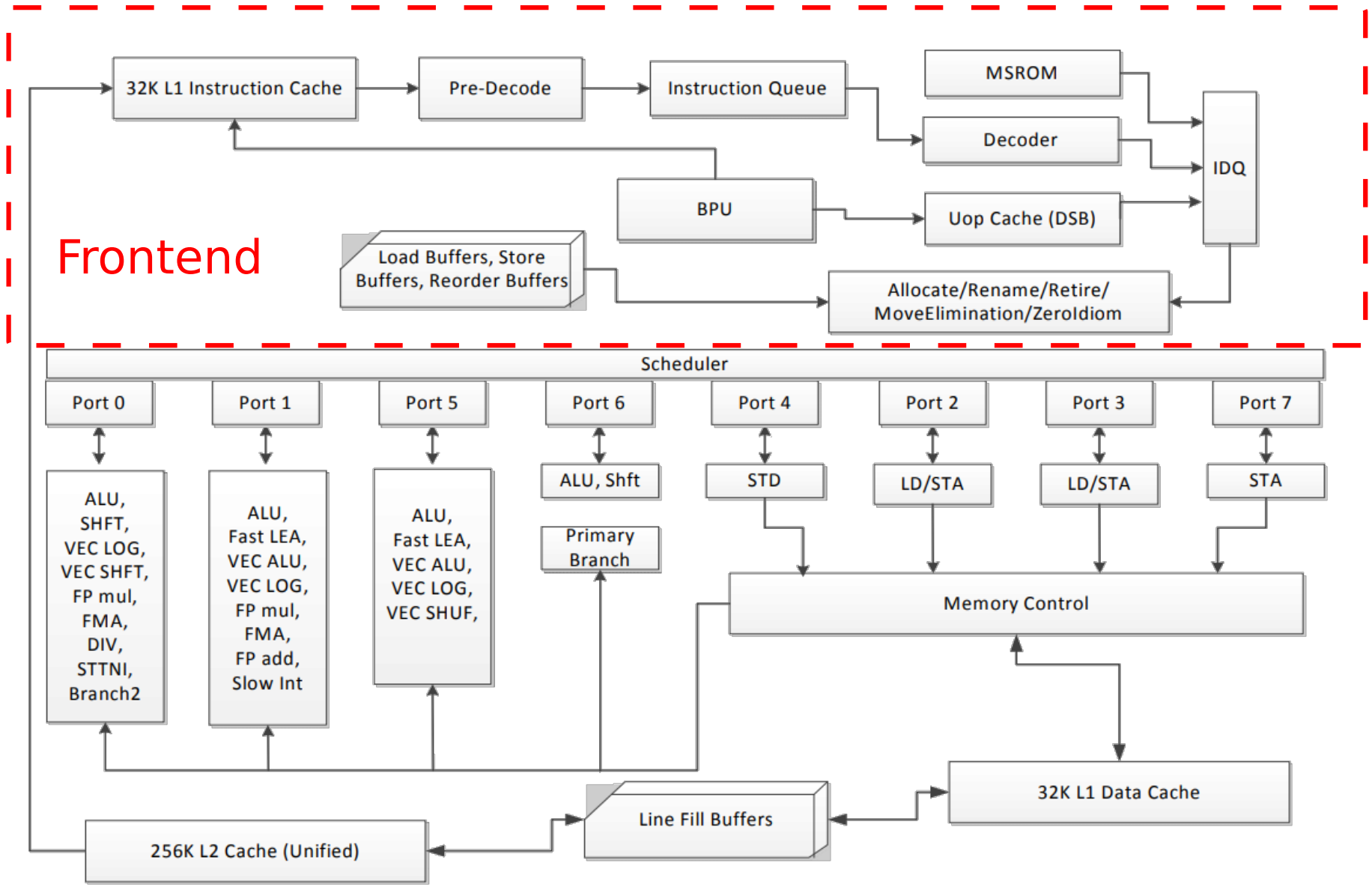

Source: karlrupp.net

# Microarchitecture (Haswell)



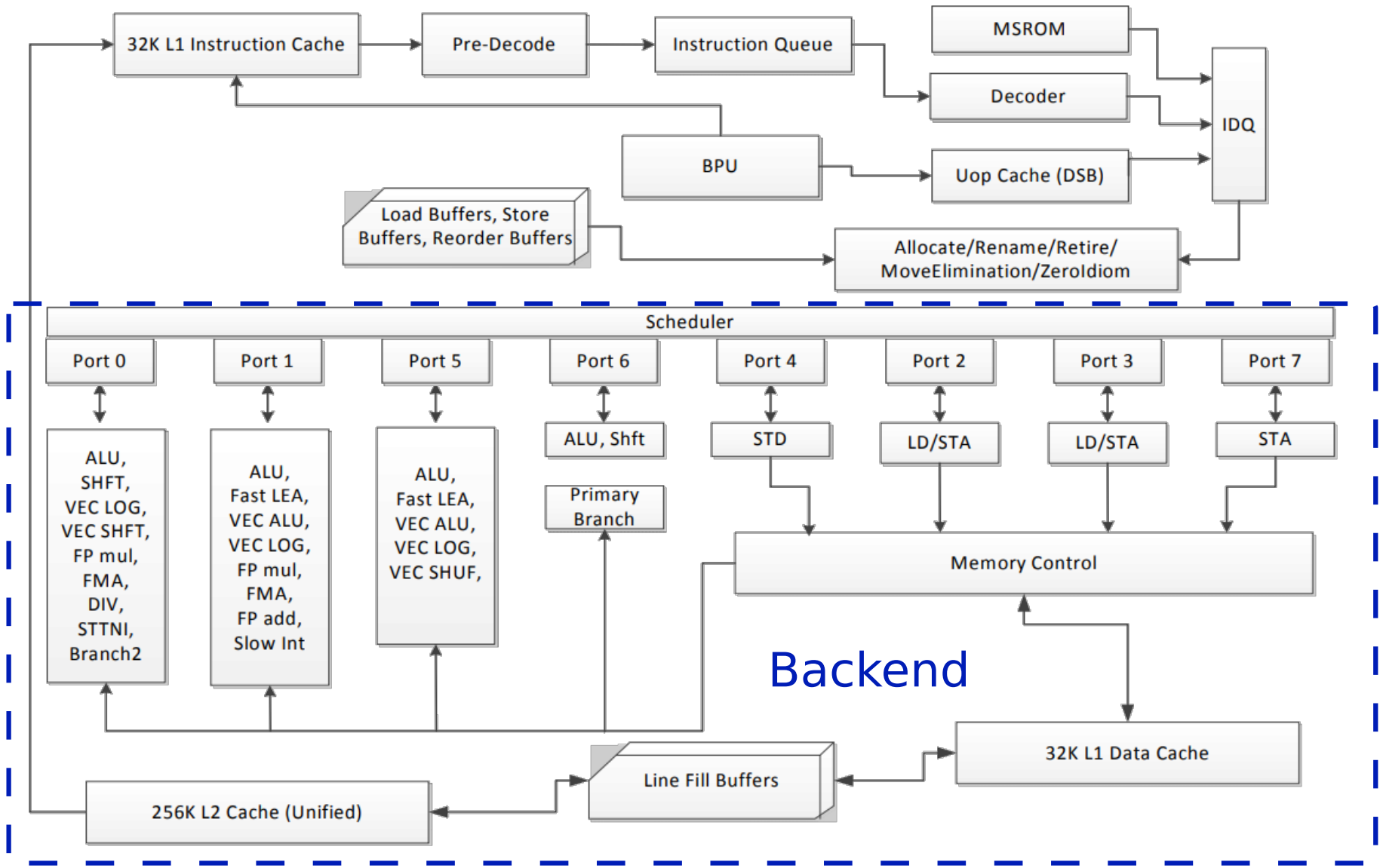

Source: Intel Architectures Optimization Reference Manual

# Microarchitecture (Haswell)




Source: Intel Architectures Optimization Reference Manual

# Microarchitecture (Haswell)




Source: Intel Architectures Optimization Reference Manual

# How bad is it?

- C++ 17 final draft:

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf
https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf
https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf

# How bad is it?

- C++ 17 final draft: 1622 pages

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf
https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf
https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf

# How bad is it?

- C++ 17 final draft: 1622 pages
- Intel x86 manual:

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf
https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf
https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf

# How bad is it?

- C++ 17 final draft: 1622 pages
- Intel x86 manual: **5764** pages!

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf
https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf
https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf

# Plan of attack

- Show example C++ programs

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`
- Demonstrate weird performance behaviour

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`
- Demonstrate weird performance behaviour
- Let you guess what might cause it

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`
- Demonstrate weird performance behaviour
- Let you guess what might cause it
- Explain (a possible) cause

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`
- Demonstrate weird performance behaviour
- Let you guess what might cause it
- Explain (a possible) cause
- Show how to measure and fix it

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`
- Demonstrate weird performance behaviour
- Let you guess what might cause it
- Explain (a possible) cause
- Show how to measure and fix it

- Disclaimer #1: Everything will be Intel x86 specific

# Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with `-O3`
- Demonstrate weird performance behaviour
- Let you guess what might cause it
- Explain (a possible) cause
- Show how to measure and fix it

- Disclaimer #1: Everything will be Intel x86 specific
- Disclaimer #2: I'm not an expert on this and I may be wrong :-)

# Let's see some examples...

# Code (backup)

```
std::vector<float> data = /* 32K random floats in [1, 10] */;
float sum = 0;
// std::sort(data.begin(), data.end());
for (auto x : data)
{
    if (x < 6.0f)
    {
        sum += x;
    }
}
```

# Result (backup)

```
---------------------------------------------------------------------
Benchmark                           Time             CPU   Iterations
---------------------------------------------------------------------
filter_nosort/32768            133460 ns       132992 ns         5284
filter_sorted/32768             63069 ns        62991 ns        12547
```

# Most upvoted Stack Overflow question

## Why is it faster to process a sorted array than an unsorted array?

22447

10294

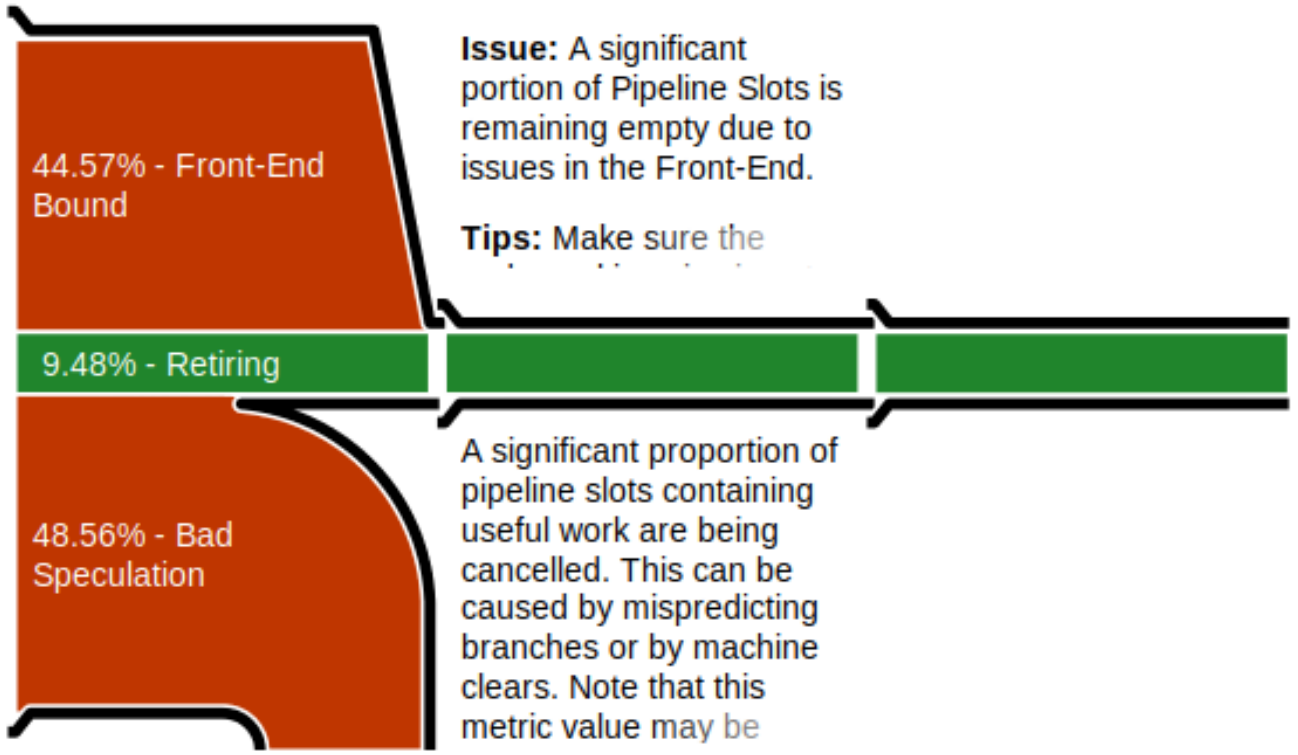
Here is a piece of C++ code that seems very peculiar. For some strange reason, sorting the data miraculously makes the code almost six times faster.

```
#include <algorithm>
#include <ctime>
#include <iostream>

int main()
{
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;

    // !!! With this, the next loop runs faster
    std::sort(data, data + arraySize);

    // Test
    clock_t start = clock();
    long long sum = 0;

    for (unsigned i = 0; i < 100000; ++i)
    {
        // Primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
```

# What is going on? (Intel Amplifier - VTune)

## Elapsed Time: 2.222s

| | | |
|---|---|---|
| Clockticks: | 6,736,800,000 | |
| Instructions Retired: | 2,942,800,000 | |
| CPI Rate: | 2.289 | |
| MUX Reliability: | 0.802 | |
| Retiring: | 9.5% | of Pipeline Slots |
| Front-End Bound: | 44.6% | of Pipeline Slots |
| Front-End Latency: | 31.1% | of Pipeline Slots |
| ICache Misses: | 0.0% | of Clockticks |
| ITLB Overhead: | 0.2% | of Clockticks |
| Branch Resteers: | 13.4% | of Clockticks |
| Mispredicts Resteers: | 12.6% | of Clockticks |
| Clears Resteers: | 0.0% | of Clockticks |
| Unknown Branches: | 0.8% | of Clockticks |
| DSB Switches: | 0.0% | of Clockticks |
| Length Changing Prefixes: | 0.0% | of Clockticks |
| MS Switches: | 4.5% | of Clockticks |
| Front-End Bandwidth: | 13.5% | of Pipeline Slots |
| Bad Speculation: | 48.6% | of Pipeline Slots |
| Branch Mispredict: | 48.6% | of Pipeline Slots |
| Machine Clears: | 0.0% | of Pipeline Slots |
| Back-End Bound: | 0.0% | of Pipeline Slots |
| Total Thread Count: | 1 | |
| Paused Time: | 0s | |



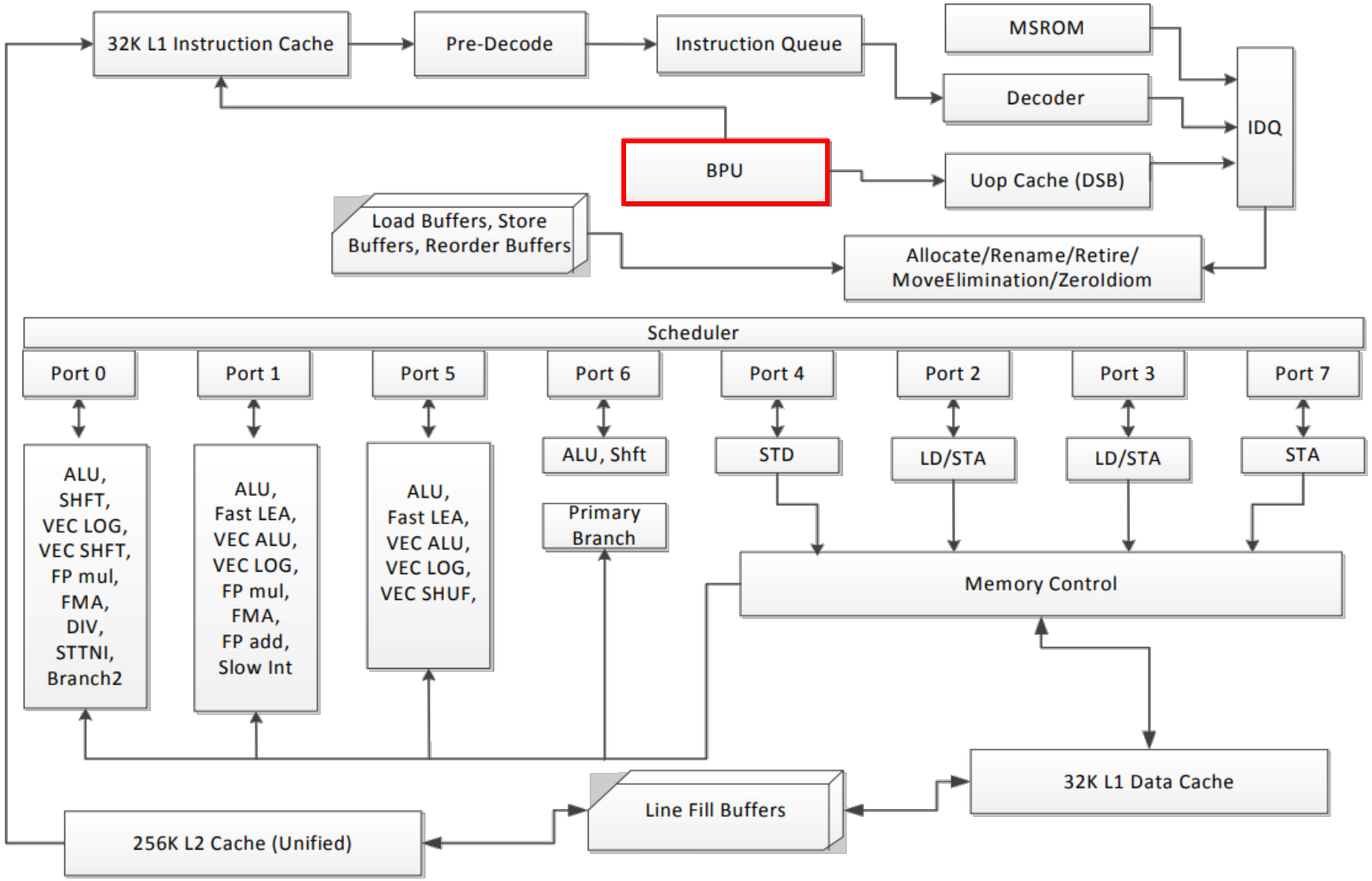

### µPipe

This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.

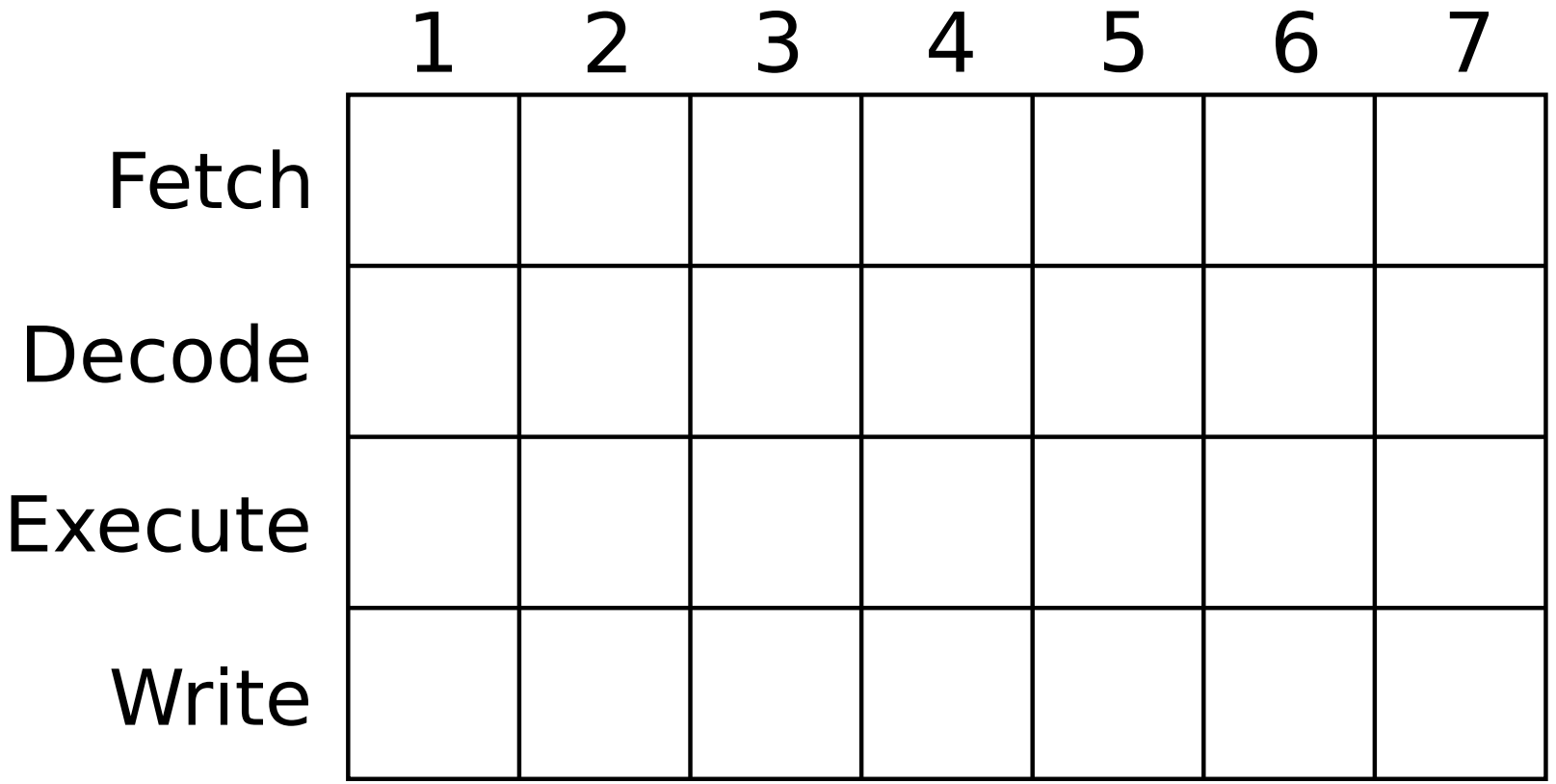
# What is going on? (perf)

```
$ perf stat ./example0a --benchmark_filter=nosort
    853,672012  task-clock (msec) #   0,997 CPUs utilized
            30  context-switches  #   0,035 K/sec
             0  cpu-migrations    #   0,000 K/sec
           199  page-faults       #   0,233 K/sec
 3 159 530 915  cycles            #   3,701 GHz
 1 475 799 619  instructions      #   0,47  insn per cycle
   419 608 357  branches          # 491,533 M/sec
   102 425 035  branch-misses     #  24,41% of all branches
```

# Branch predictor

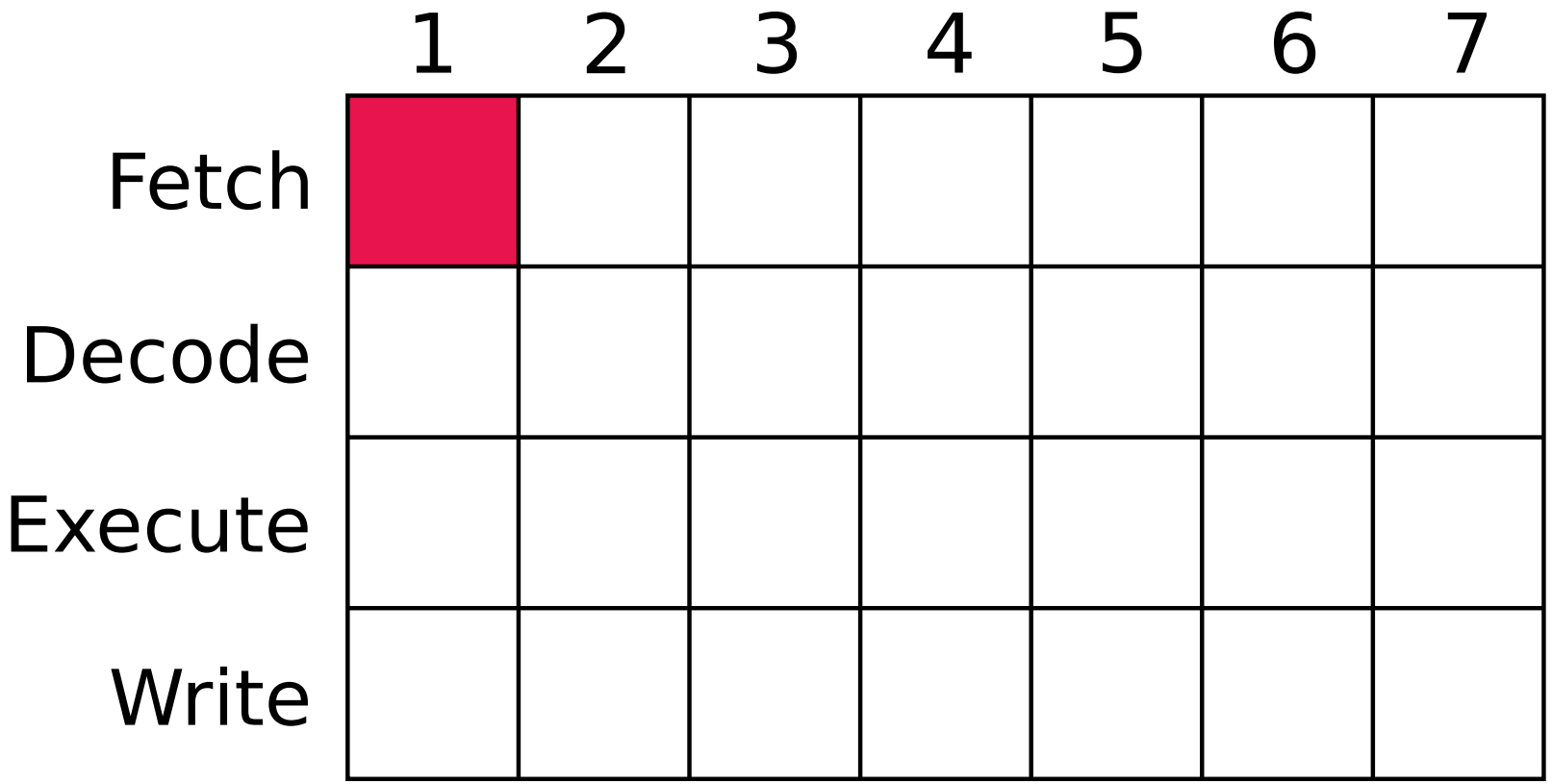


# CPU pipeline 101



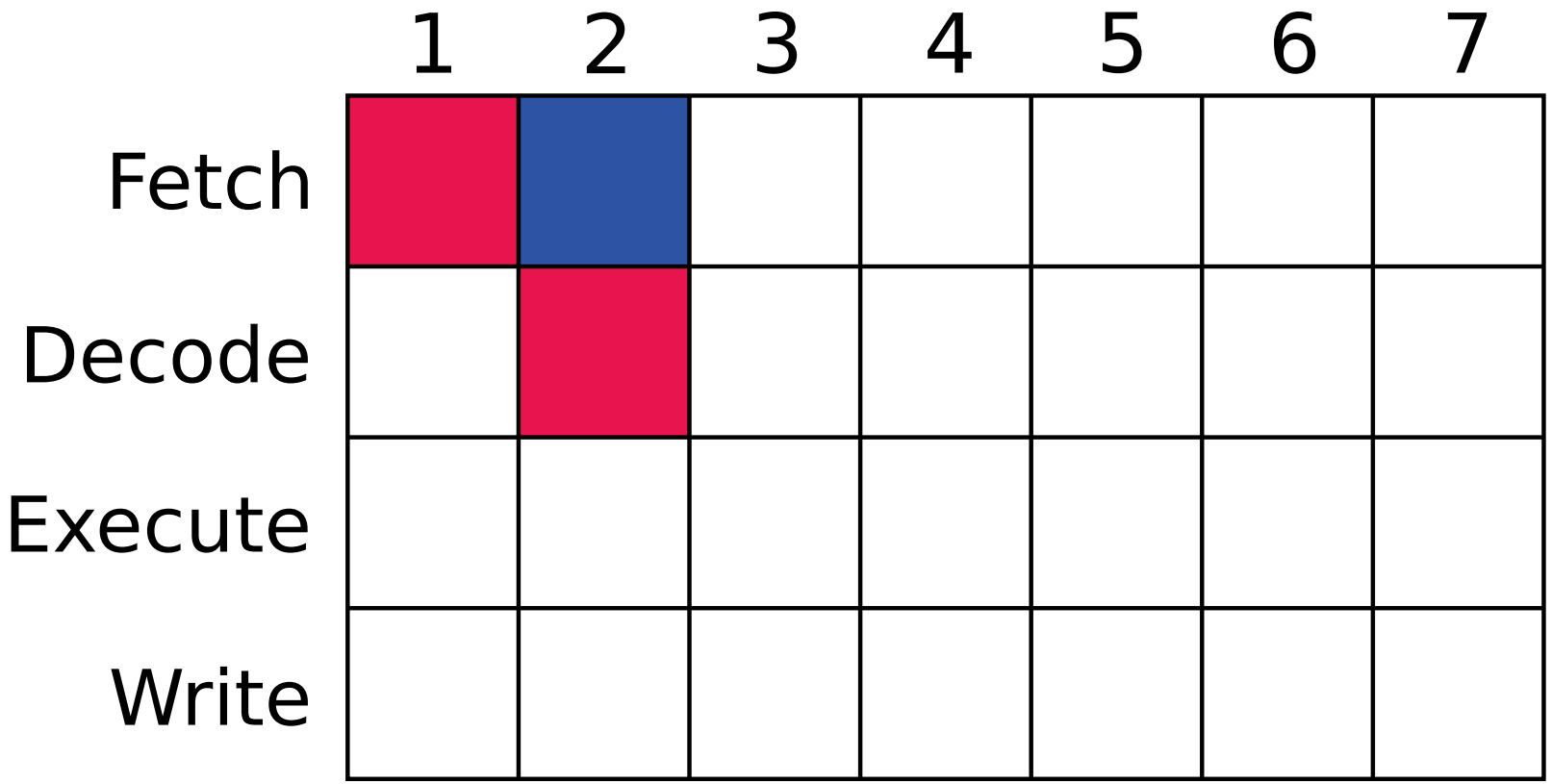

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

# CPU pipeline 101



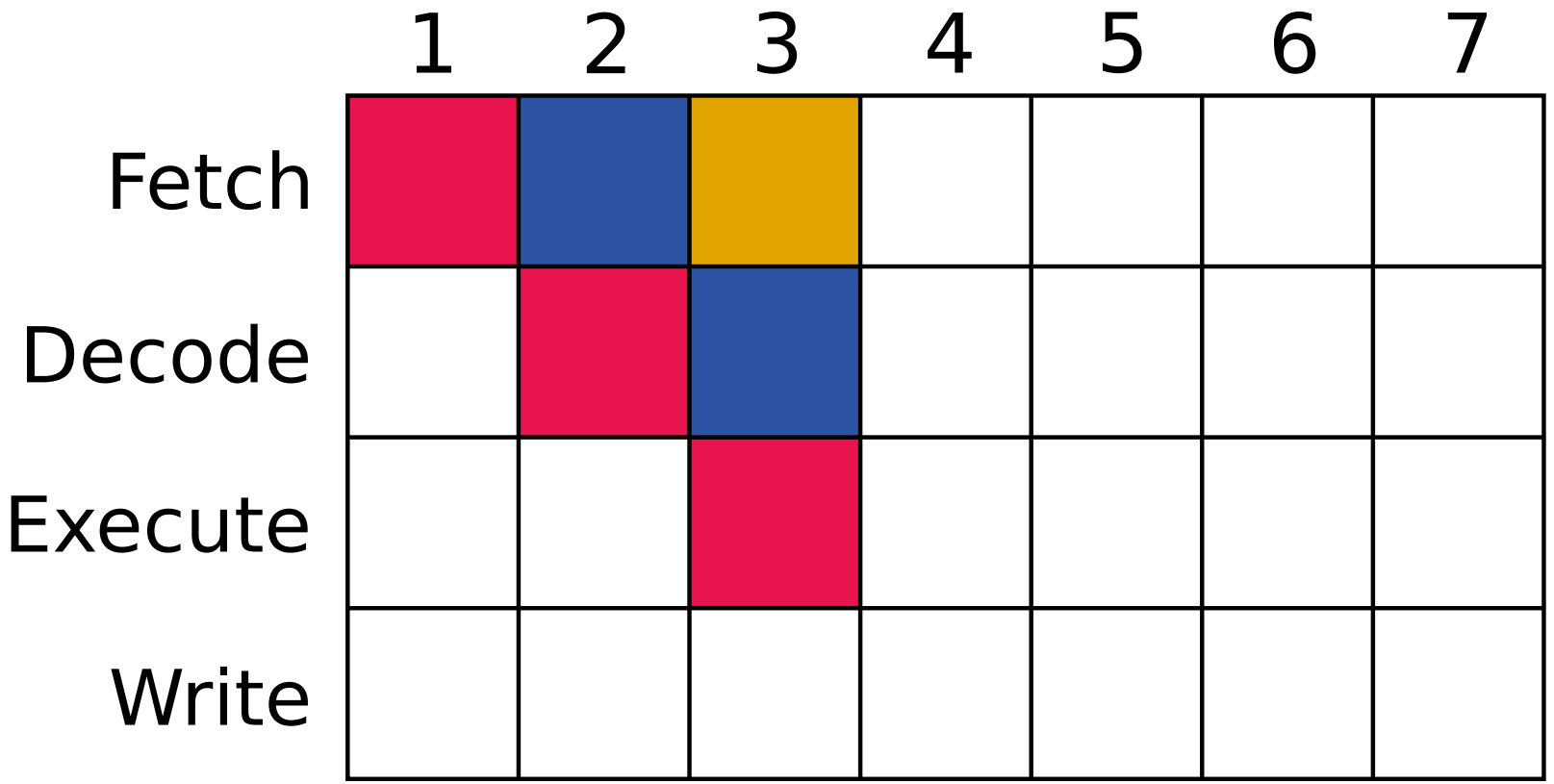

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

# CPU pipeline 101



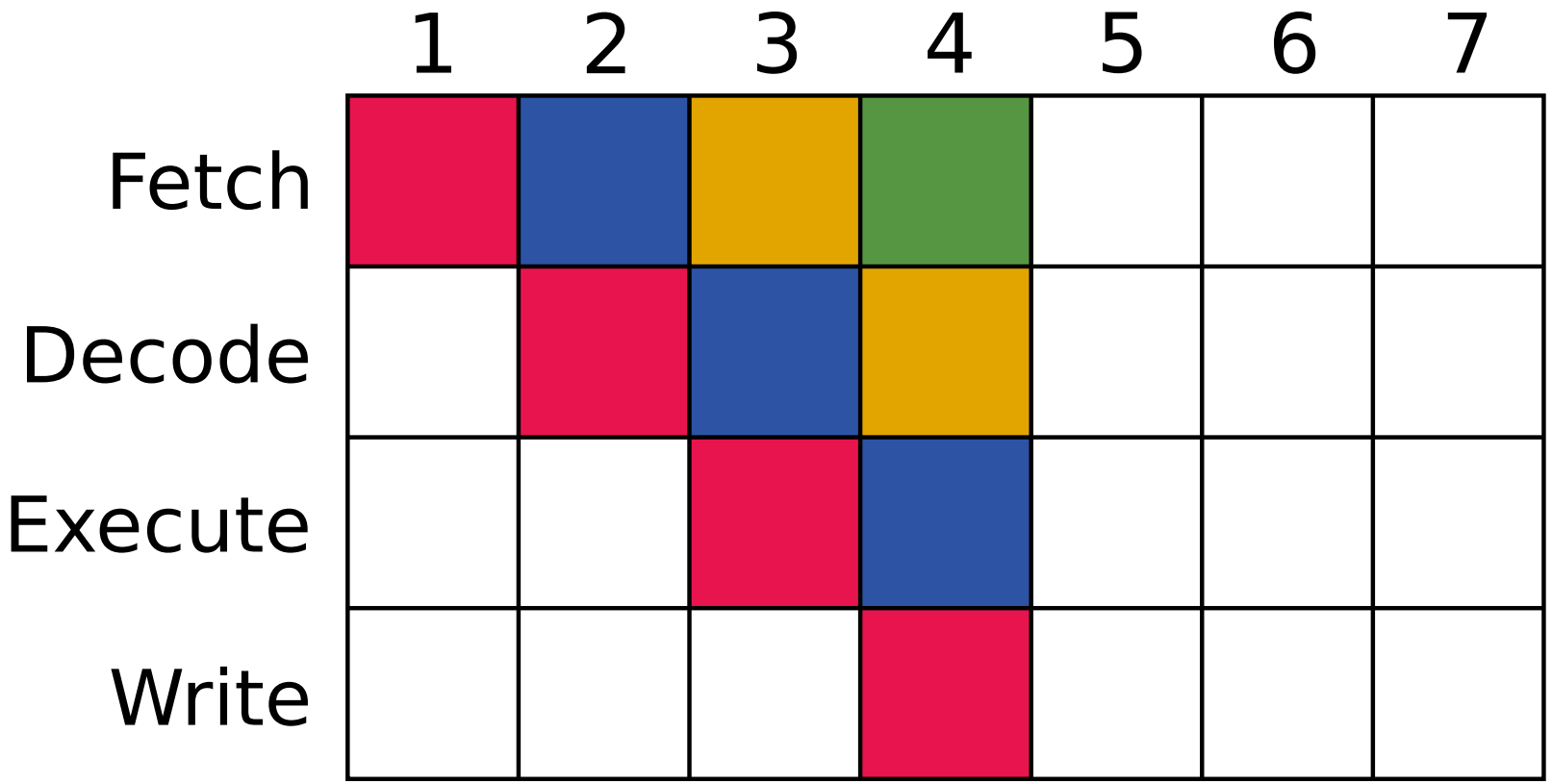

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

# CPU pipeline 101



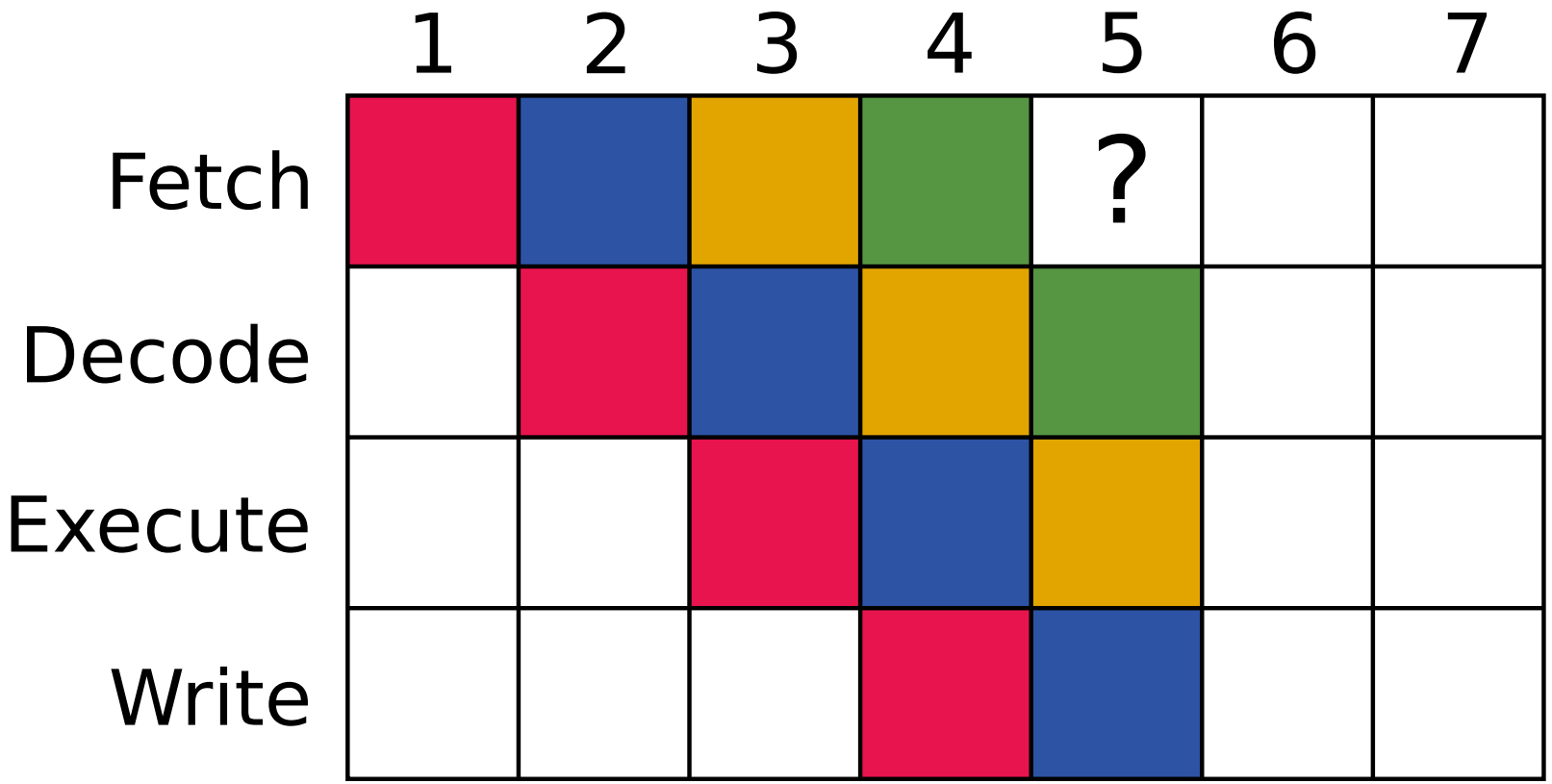

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

# CPU pipeline 101




```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

# CPU pipeline 101




```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

# Branch predictor

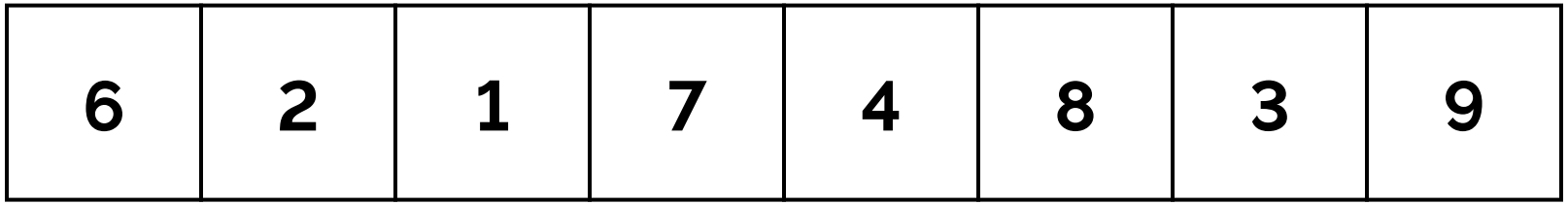
- CPU tries to predict results of branches

# Branch predictor

- CPU tries to predict results of branches
- Misprediction can cost ~15-20 cycles!

# Simple branch predictor - unsorted array

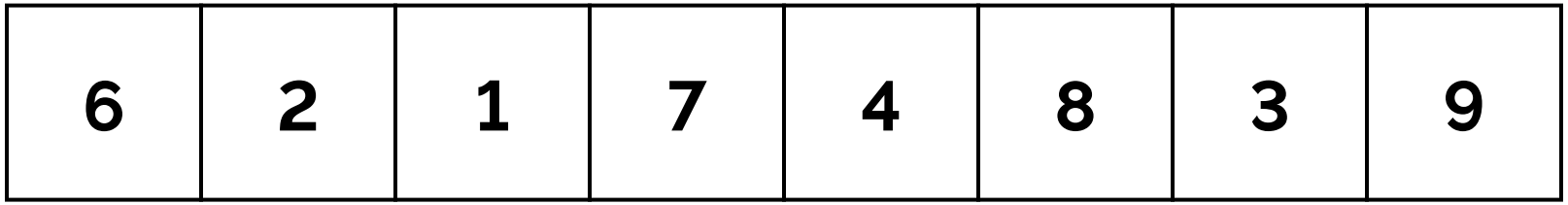
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

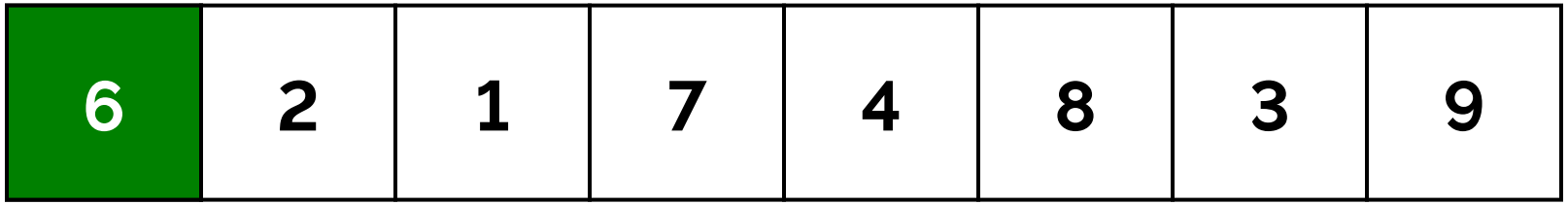



6 < 6?

Prediction: Not taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

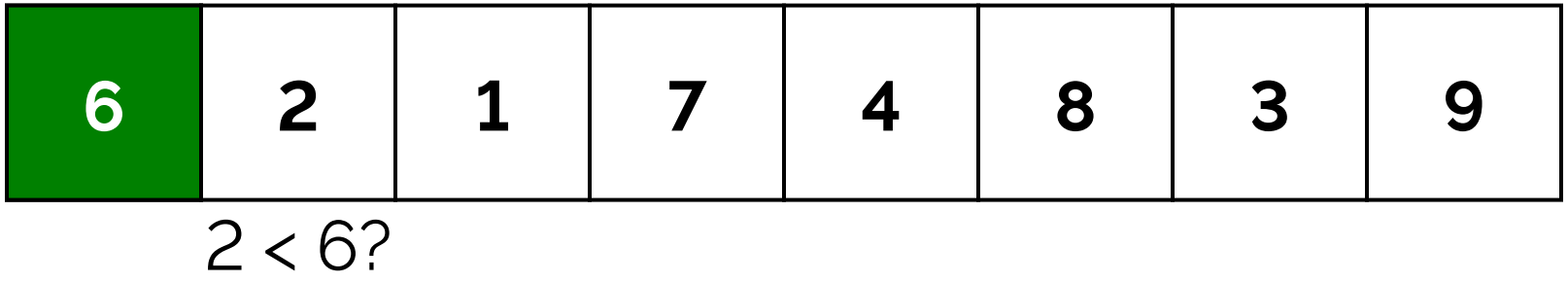



6 < 6?

Prediction: Not taken

# Simple branch predictor - unsorted array

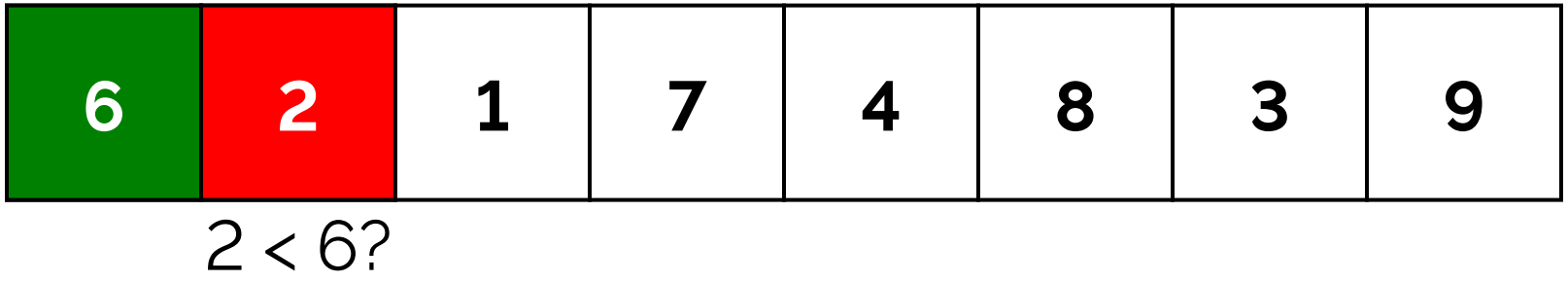
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - unsorted array

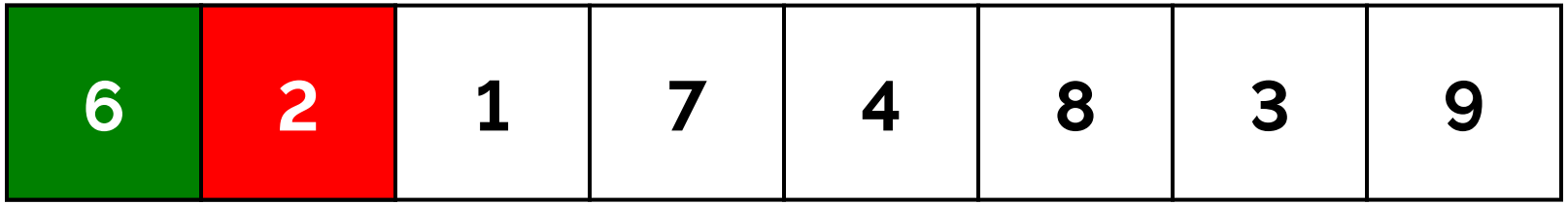
```
if (data[i] < 6) {
    ...
}
```




Prediction: Taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

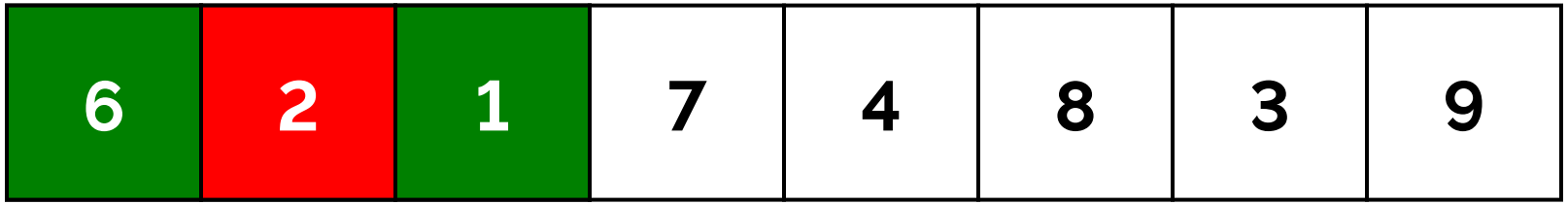



1 < 6?

Prediction: Taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

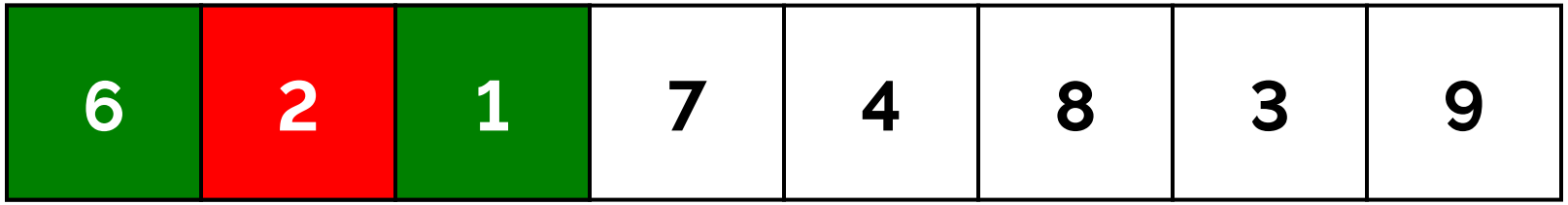



1 < 6?

Prediction: Taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

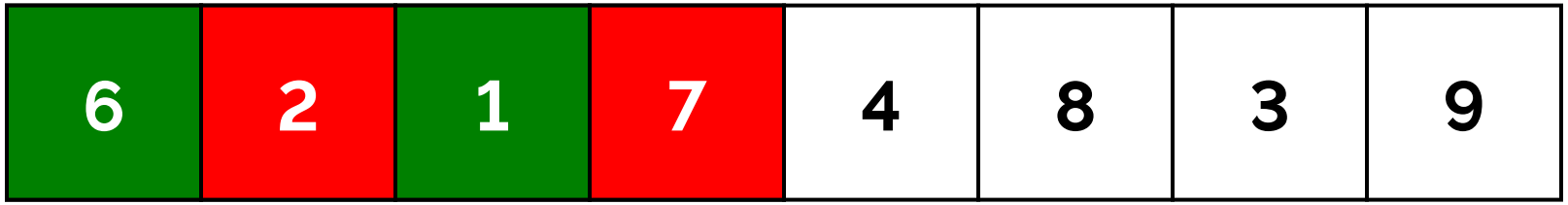



7 < 6?

Prediction: Taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

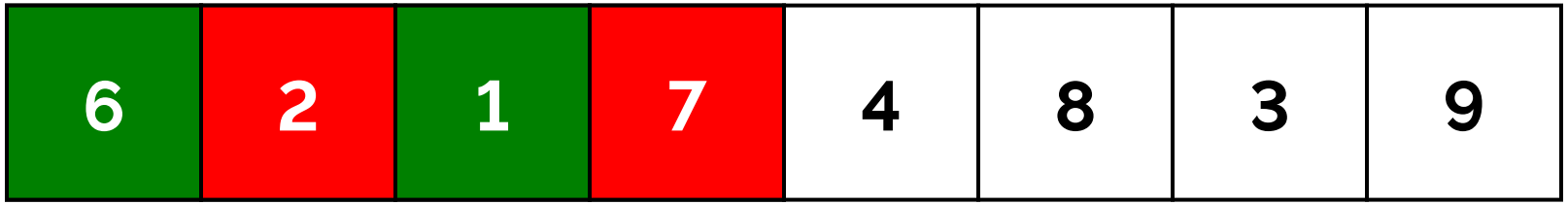



7 < 6?

Prediction: Not taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

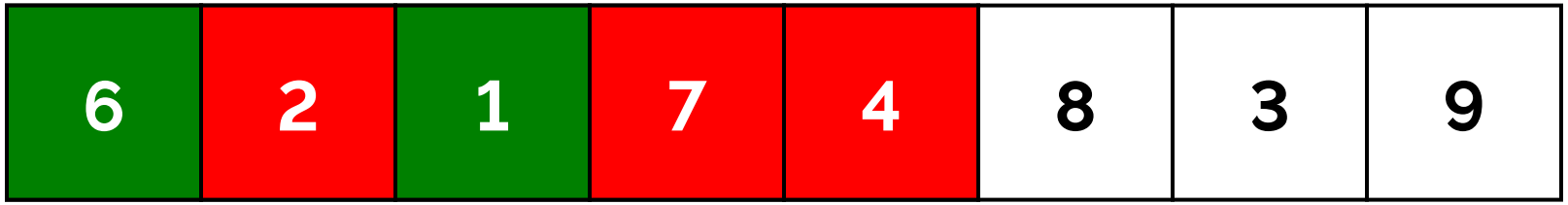



4 < 6?

Prediction: Not taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

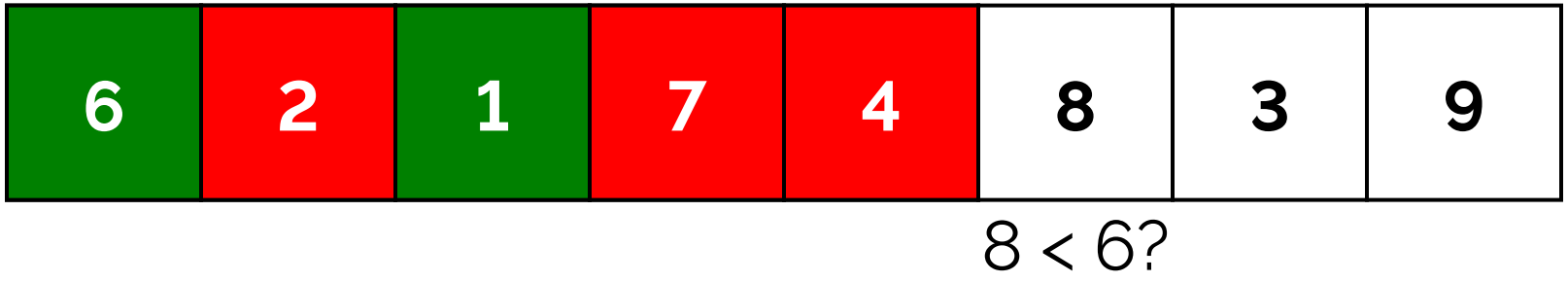



4 < 6?

Prediction: Taken

# Simple branch predictor - unsorted array

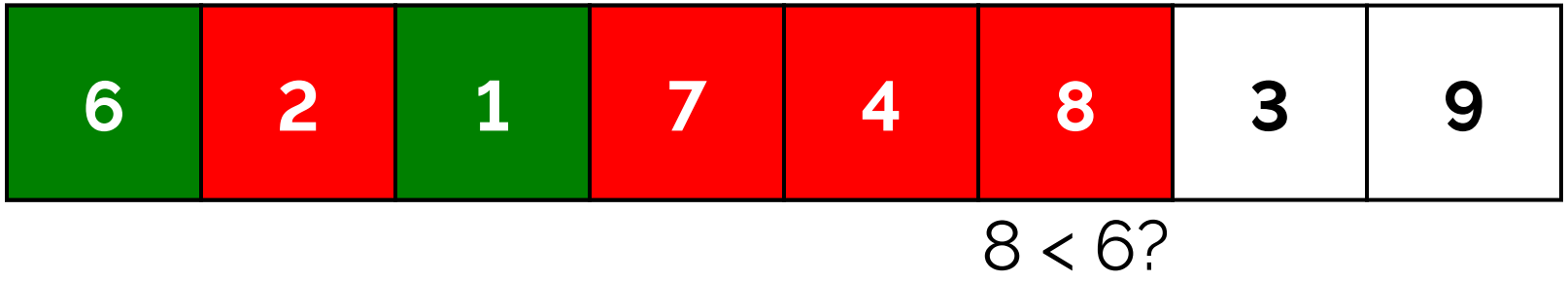
```
if (data[i] < 6) {
    ...
}
```




Prediction: Taken

# Simple branch predictor - unsorted array

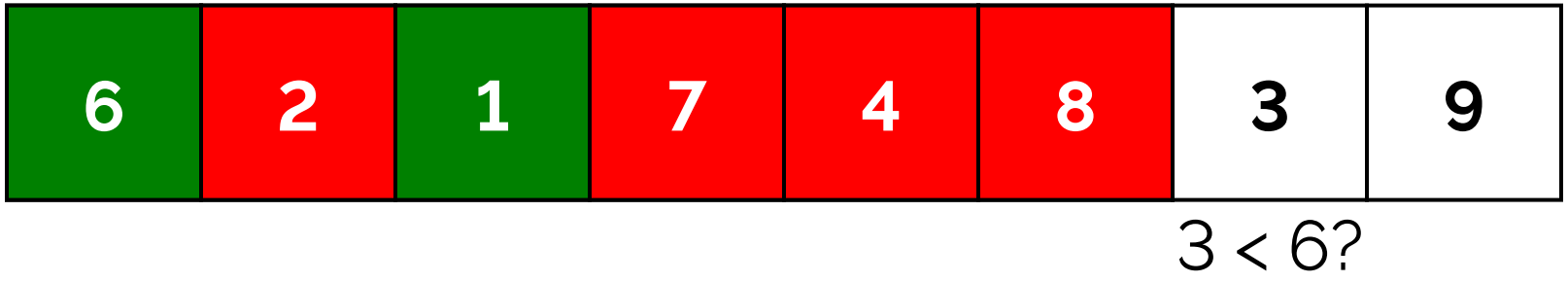
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - unsorted array

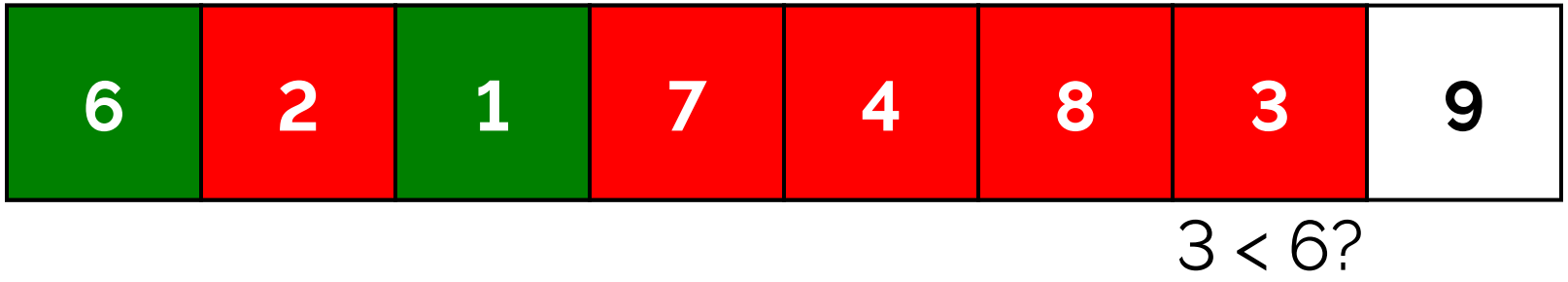
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - unsorted array

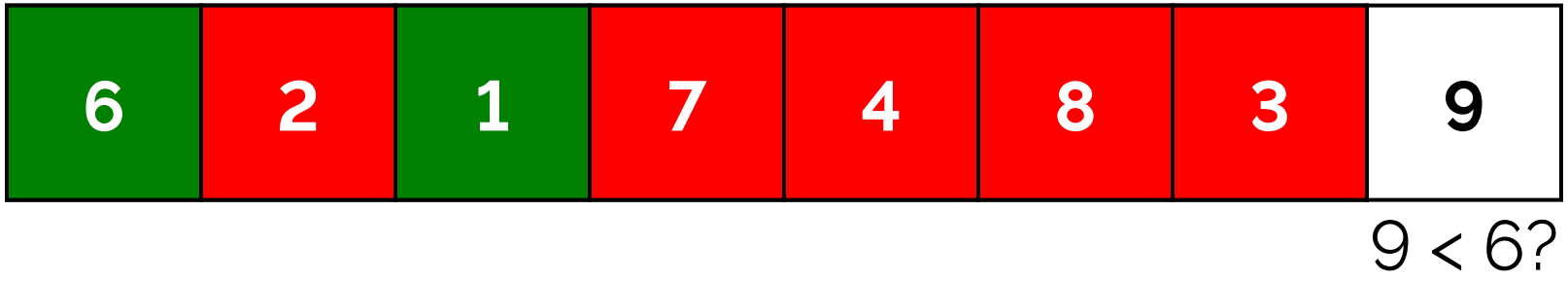
```
if (data[i] < 6) {
    ...
}
```




Prediction: Taken

# Simple branch predictor - unsorted array

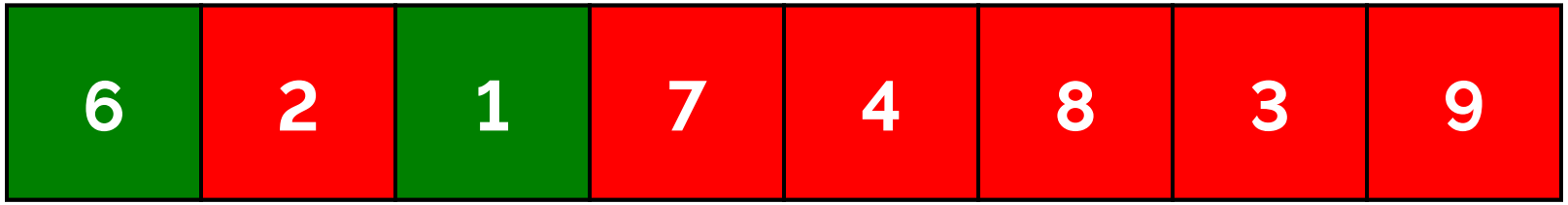
```
if (data[i] < 6) {
    ...
}
```




Prediction: Taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```

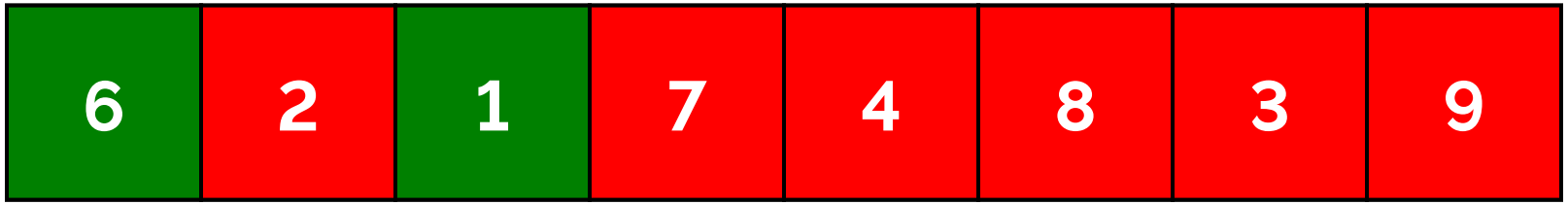



9 < 6?

Prediction: Not taken

# Simple branch predictor - unsorted array

```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

2 hits, 6 misses (25% hit rate)

# Simple branch predictor - sorted array

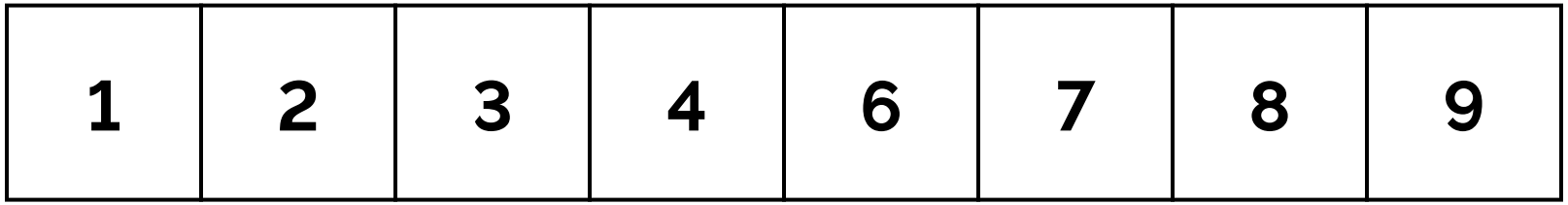
```
if (data[i] < 6) {
    ...
}
```

| 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|

Prediction: Not taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

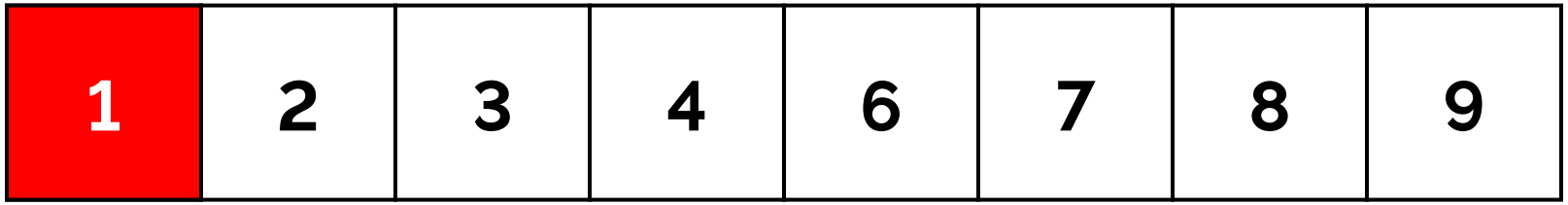



1 < 6?

Prediction: Not taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

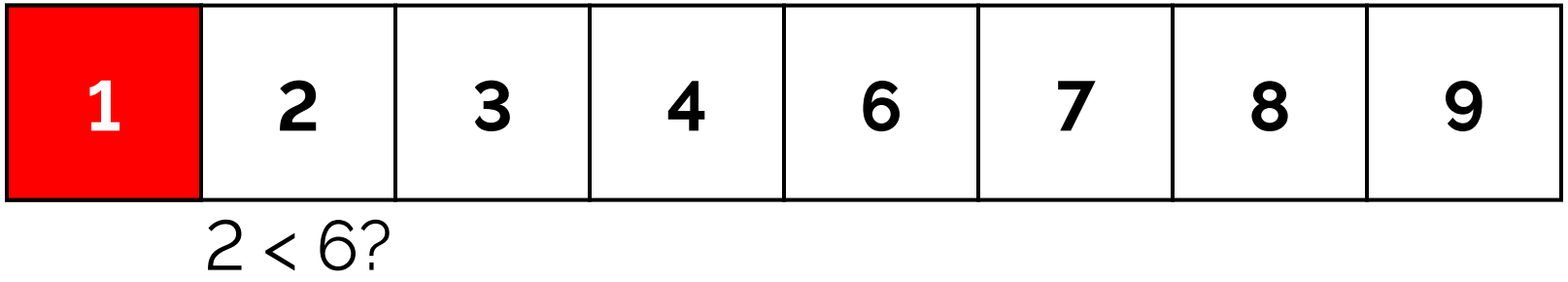



1 < 6?

Prediction: Taken

# Simple branch predictor - sorted array

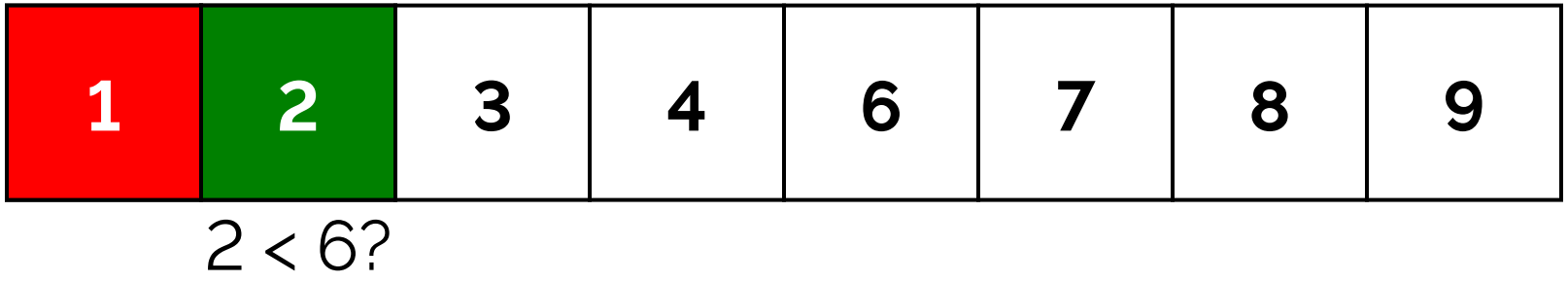
```
if (data[i] < 6) {
    ...
}
```




Prediction: Taken

# Simple branch predictor - sorted array

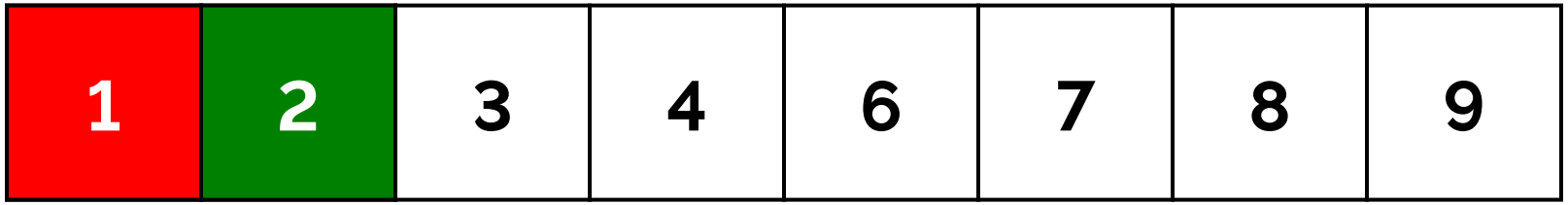
```
if (data[i] < 6) {
    ...
}
```




Prediction: Taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

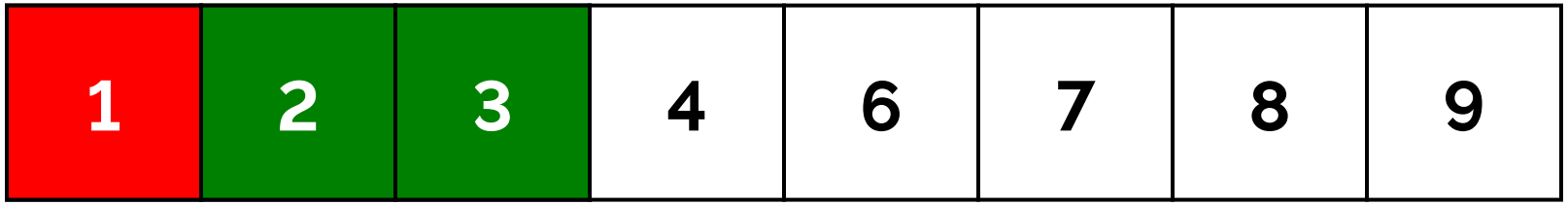



3 < 6?

Prediction: Taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

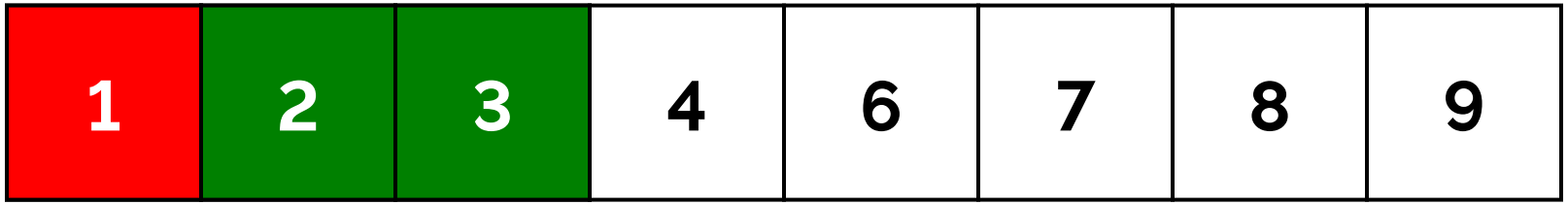



3 < 6?

Prediction: Taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

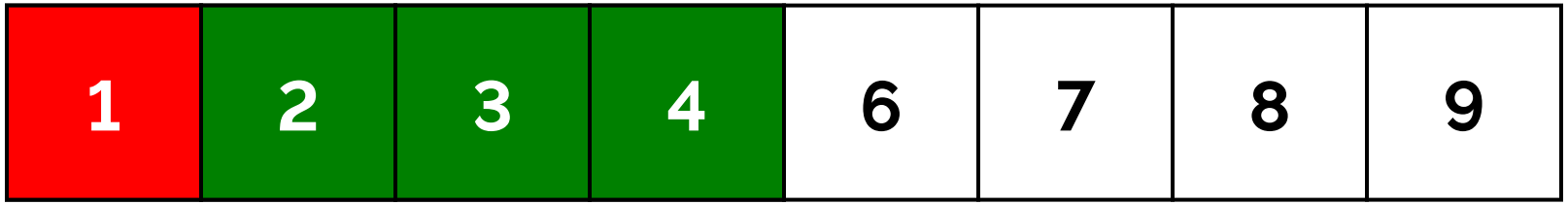



4 < 6?

Prediction: Taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

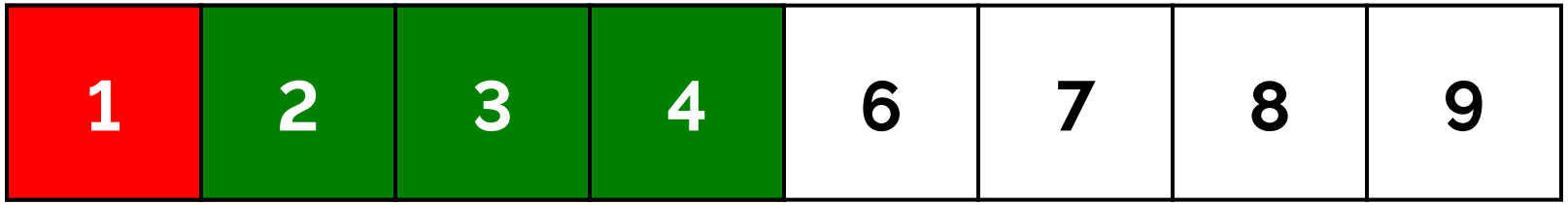



4 < 6?

Prediction: Taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

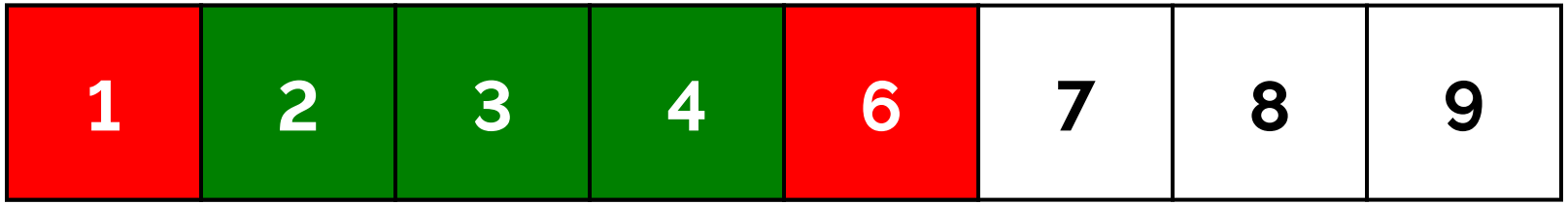



6 < 6?

Prediction: Taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

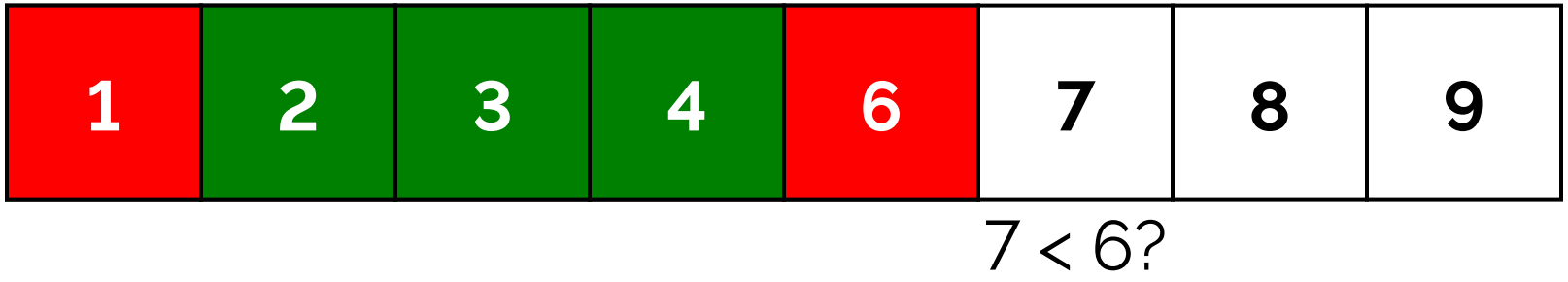



6 < 6?

Prediction: Not taken

# Simple branch predictor - sorted array

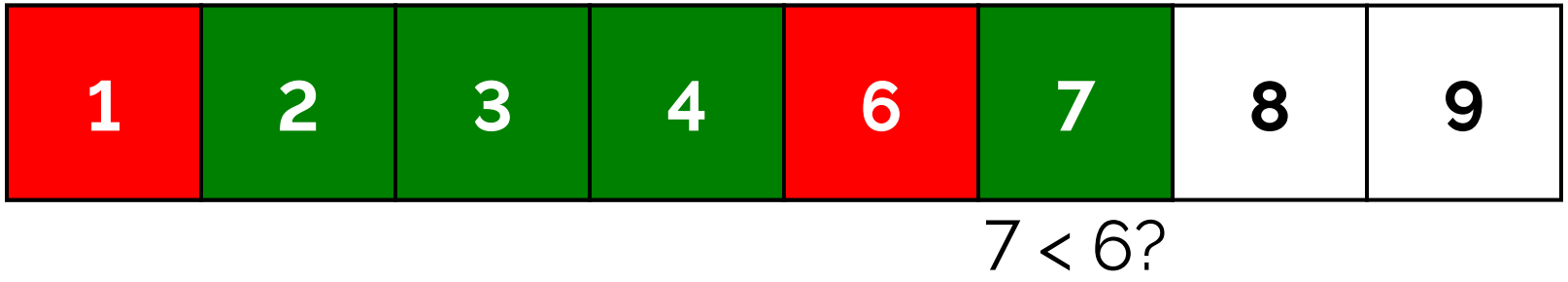
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - sorted array

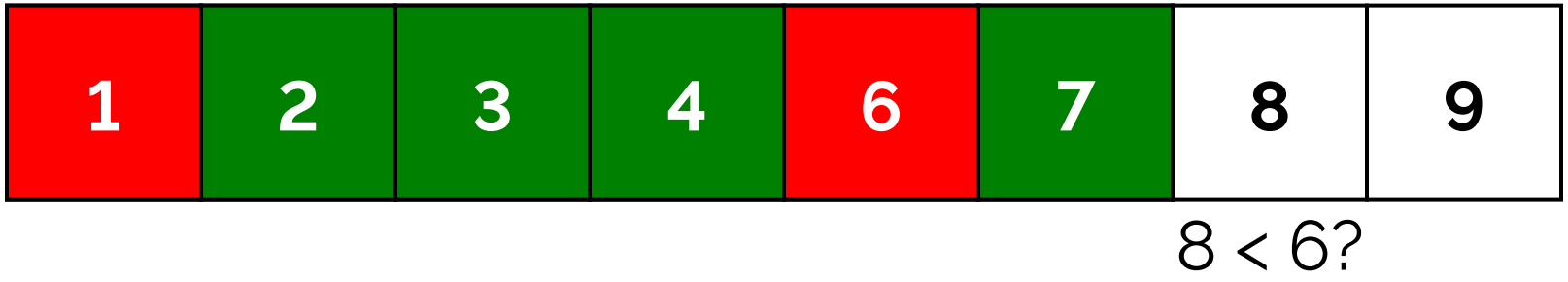
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - sorted array

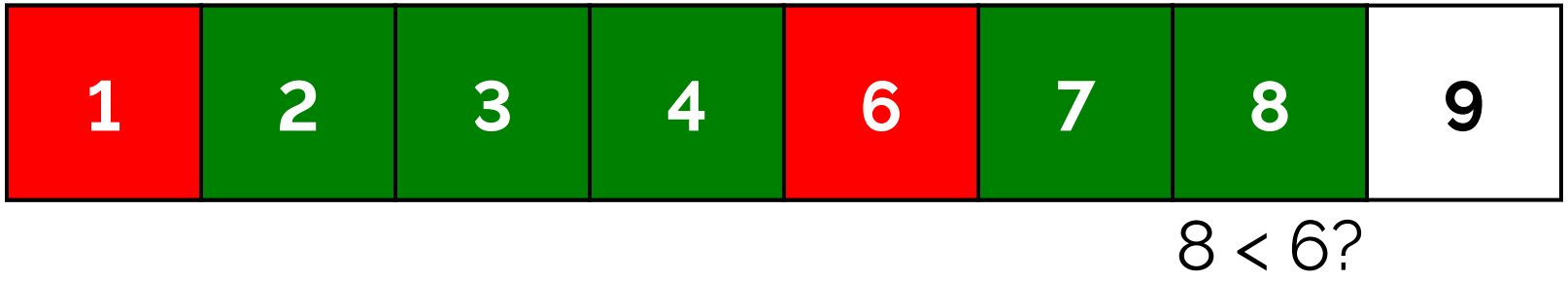
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - sorted array

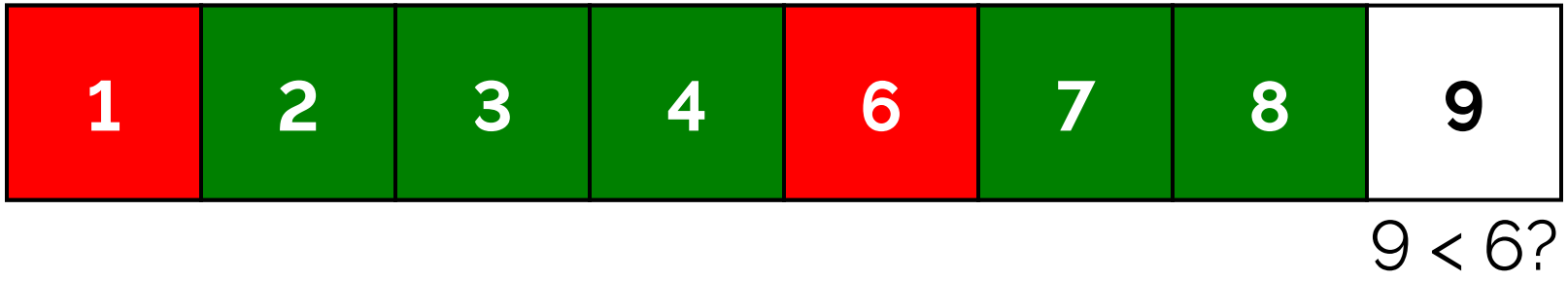
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - sorted array

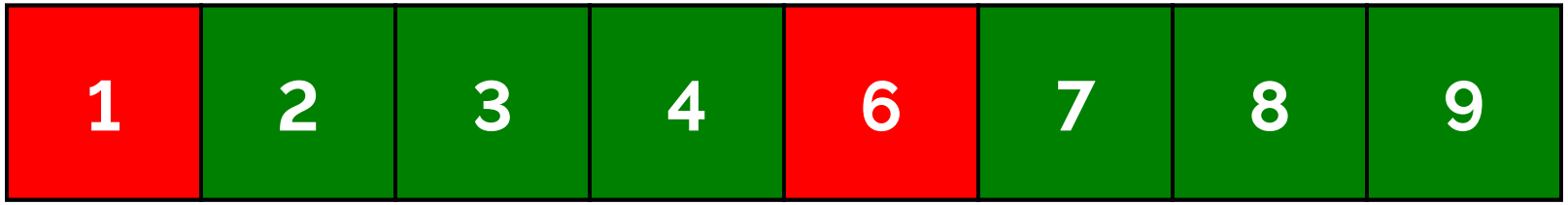
```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

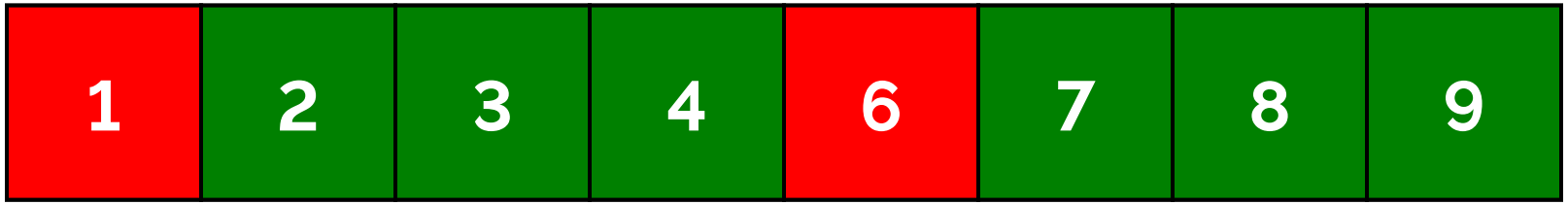



9 < 6?

Prediction: Not taken

# Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```




Prediction: Not taken

6 hits, 2 misses (75% hit rate)

# How can the compiler help?

```
float sum = 0;
for (auto _ : state)
{
    for (auto x : data)
    {
        if (x < 6)
        {
            sum += x;
        }
    }
}
```

```
47          mov     rax, rcx
48          cmp     rcx, rdx
49          je      .L9
50  .L12:
51          movss   xmm0, DWORD PTR [rax]
52          comiss  xmm2, xmm0
53          jbe     .L10
54          addss   xmm1, xmm0
55  .L10:
56          add     rax, 4
57          cmp     rdx, rax
58          jne     .L12
59  .L9:
60          sub     rbx, 1
61          jne     .L13
62          jmp     .L6
```

With `float`, there are two branches per iteration

# How can the compiler help?

```
int sum = 0;
for (auto _ : state)
{
    for (auto x : data)
    {
        if (x < 6)
        {
            sum += x;
        }
    }
}
```

```
41  .L12:
42          mov     rax, r8
43          cmp     r8, rdi
44          je      .L9
45  .L11:
46          mov     edx, DWORD PTR [rax]
47          cmp     edx, 6
48          lea     ecx, [rbx+rdx]
49          cmovl   ebx, ecx
50          add     rax, 4
51          cmp     rdi, rax
52          jne     .L11
53  .L9:
54          sub     rbp, 1
55          jne     .L12
56          mov     rdi, r12
```

With `int`, one branch is removed (using `cmov`)

# How to measure?

`branch-misses`

How many times was a branch mispredicted?

# How to measure?

## `branch-misses`

How many times was a branch mispredicted?

```
$ perf stat -e branch-misses ./example0a
with    sort ->     383 902
without sort -> 101 652 009
```

# How to help the branch predictor?

- More predictable data

# How to help the branch predictor?

- More predictable data
- Profile-guided optimization

# How to help the branch predictor?

- More predictable data
- Profile-guided optimization
- Remove (unpredictable) branches

# How to help the branch predictor?

- More predictable data
- Profile-guided optimization
- Remove (unpredictable) branches
- Compiler hints (use with caution)

```
if (__builtin_expect(will_it_blend(), 0)) {
    // this branch is not likely to be taken
}
```

# Branch target prediction

- Target of a jump is not known at compile time:

# Branch target prediction

- Target of a jump is not known at compile time:
  - Function pointer

# Branch target prediction

- Target of a jump is not known at compile time:
  - Function pointer
  - Function return address

# Branch target prediction

- Target of a jump is not known at compile time:
  - Function pointer
  - Function return address
  - Virtual method

# Code (backup)

```
struct A { virtual void handle(size_t* data) const = 0; };
struct B: public A { void handle(size_t* data) const final { *data += 1; } };
struct C: public A { void handle(size_t* data) const final { *data += 2; } };

std::vector<std::unique_ptr<A>> data = /* 4K random B/C instances */;
// std::sort(data.begin(), data.end(), /* sort by instance type */);
size_t sum = 0;
for (auto& x : data)
{
    x->handle(&sum);
}
```

# Result (backup)

```
-----------------------------------------------------------------
Benchmark                       Time             CPU   Iterations
-----------------------------------------------------------------
handle_nosort/4096          23350 ns        23349 ns        30734
handle_sorted/4096           7448 ns         7448 ns        86814
```

# perf (backup)

```
$ perf stat -e branch-misses ./example0b
with sort    ->      337 274
without sort -> 84 183 161
```

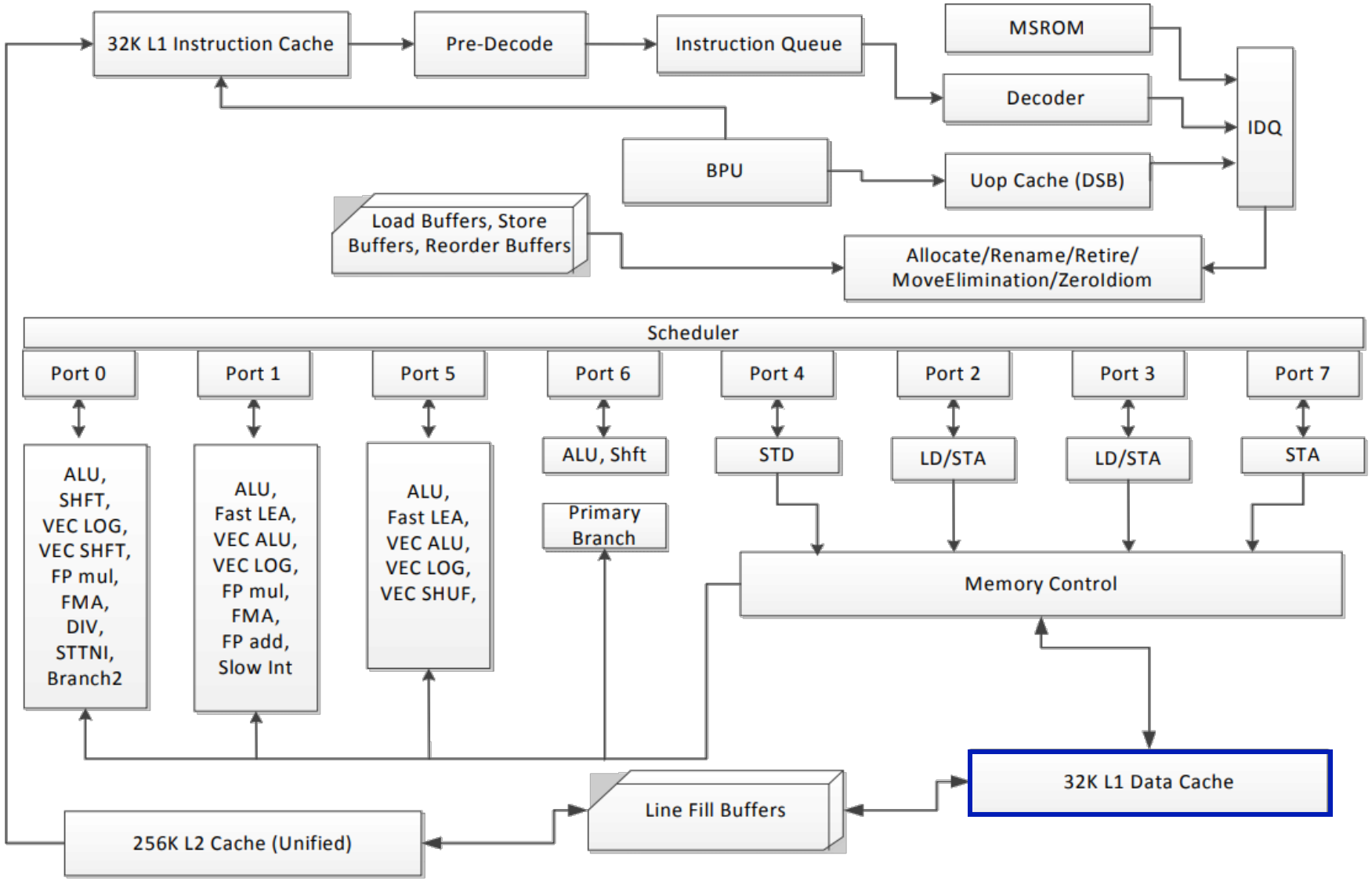
# Code (backup)

```
// Addresses of N integers, each `offset` bytes apart
std::vector<int*> data = ...;
for (auto ptr: data)
{
    *ptr += 1;
}
// Offsets: 4, 64, 4000, 4096, 4128
```

# Result (backup)

| Benchmark | Time | CPU | Iterations |
|---|---|---|---|
| offset4/7 | 2.83 ns | 2.83 ns | 255750233 |
| offset4/8 | 3.03 ns | 3.02 ns | 177109965 |
| offset4/9 | 3.76 ns | 3.75 ns | 157739295 |
| offset4/10 | 4.62 ns | 4.61 ns | 177899906 |
| offset4/11 | 4.93 ns | 4.92 ns | 162959140 |
| offset64/7 | 3.19 ns | 3.18 ns | 179723151 |
| offset64/8 | 3.83 ns | 3.65 ns | 216288609 |
| offset64/9 | 3.74 ns | 3.74 ns | 201008685 |
| offset64/10 | 4.41 ns | 4.40 ns | 159949703 |
| offset64/11 | 4.41 ns | 4.41 ns | 128933855 |
| offset4000/7 | 3.69 ns | 3.69 ns | 187745245 |
| offset4000/8 | 3.27 ns | 3.26 ns | 226401022 |
| offset4000/9 | 4.19 ns | 4.18 ns | 157866983 |
| offset4000/10 | 4.49 ns | 4.48 ns | 173084452 |
| offset4000/11 | 4.53 ns | 4.52 ns | 128906229 |
| offset4096/7 | 9.05 ns | 9.05 ns | 78087527 |
| offset4096/8 | 10.4 ns | 10.4 ns | 67550724 |
| offset4096/9 | 18.7 ns | 18.7 ns | 38875870 |
| offset4096/10 | 25.5 ns | 25.5 ns | 26893946 |
| offset4096/11 | 32.7 ns | 32.7 ns | 21369400 |
| offset4128/7 | 3.23 ns | 3.22 ns | 250263727 |
| offset4128/8 | 3.13 ns | 3.13 ns | 218371877 |
| offset4128/9 | 3.75 ns | 3.71 ns | 157448182 |
| offset4128/10 | 4.28 ns | 4.25 ns | 144839049 |
| offset4128/11 | 5.47 ns | 5.44 ns | 128547528 |

# Cache memory



# How are (L1) caches implemented

- N-way set associative table
  - Hardware hash table

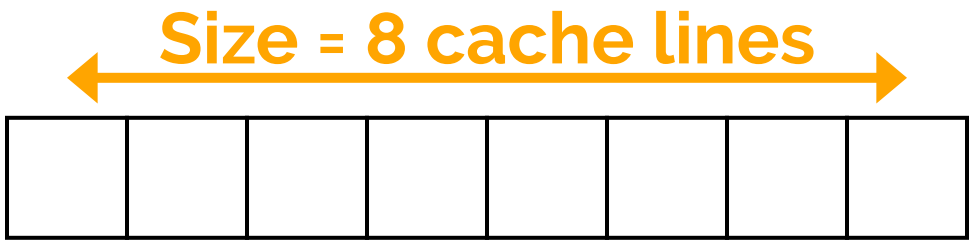
# How are (L1) caches implemented

- N-way set associative table
  - Hardware hash table
- Key = address (8B)

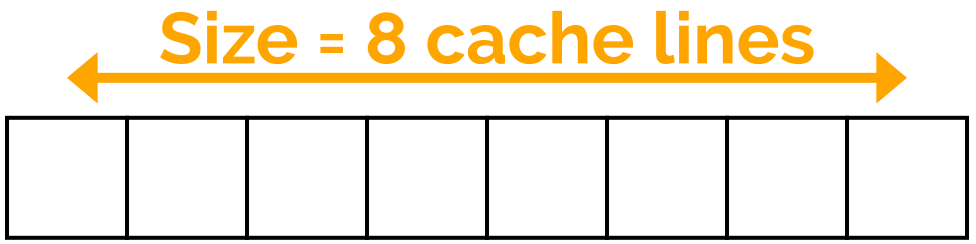
# How are (L1) caches implemented

- N-way set associative table
  - Hardware hash table
- Key = address (8B)
- Entry = cache line (64B)

# N-way set associative cache

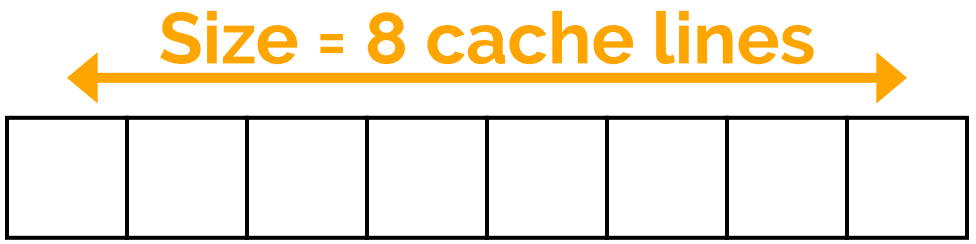


# N-way set associative cache




Associativity (N) - # of cache lines per bucket

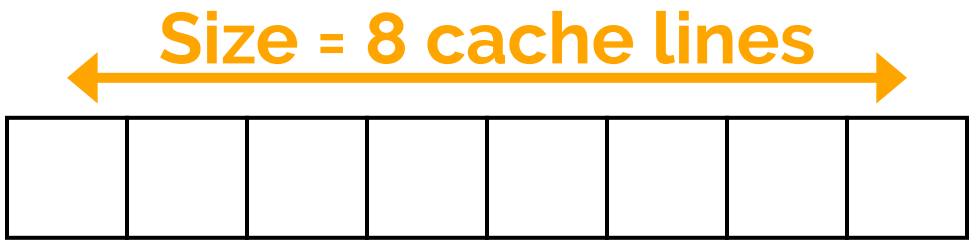
# N-way set associative cache




Associativity (N) - # of cache lines per bucket

# of buckets = Size / N

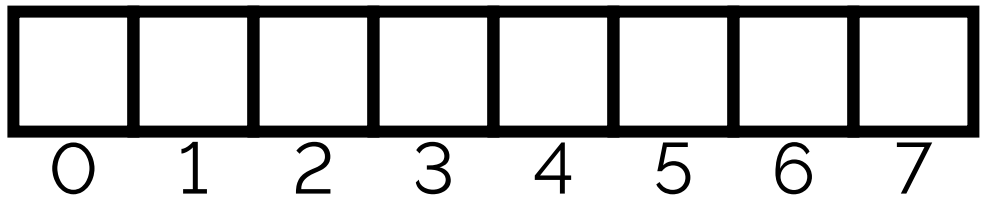
# N-way set associative cache



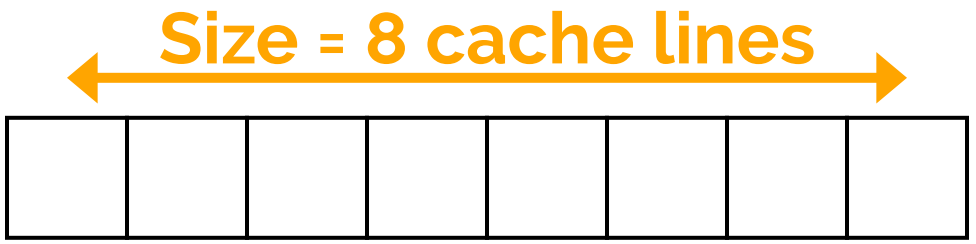

Associativity (N) - # of cache lines per bucket

# of buckets = Size / N

N = 1 (direct mapped)



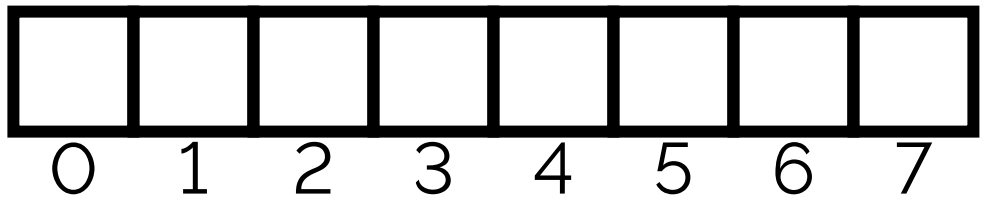

# N-way set associative cache



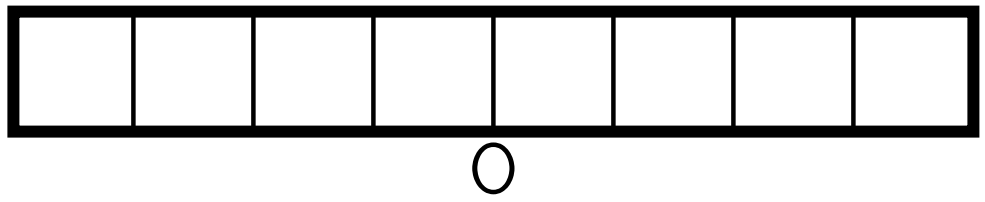

Associativity (N) - # of cache lines per bucket

# of buckets = Size / N

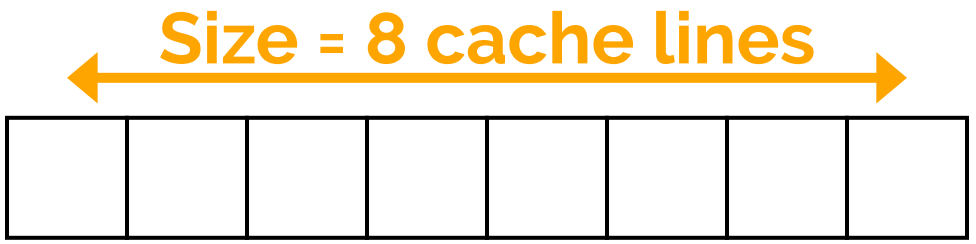
N = 1 (direct mapped)




N = 8 (fully associative)



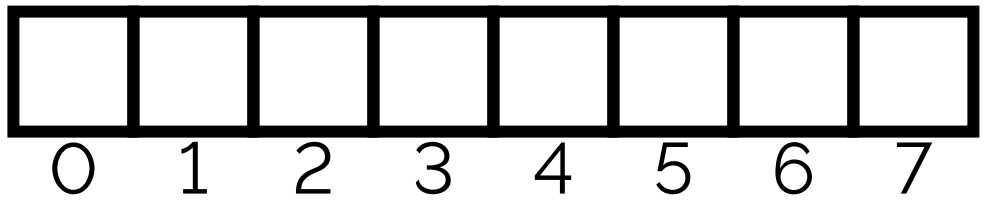

# N-way set associative cache



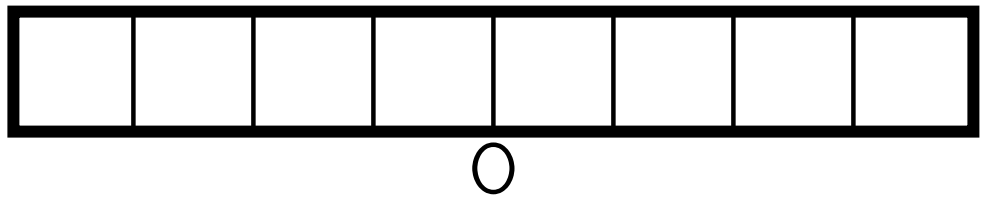

Associativity (N) - # of cache lines per bucket

# of buckets = Size / N

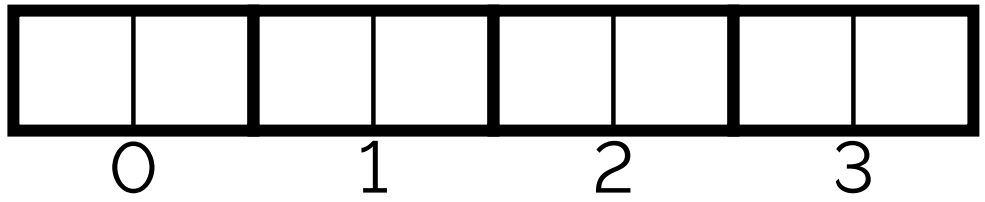
N = 1 (direct mapped)




N = 8 (fully associative)




N = 2



# How are addresses hashed?


# How are addresses hashed?


- **Offset**
  - Selects byte within a cache line
  - log2(cache line size) bits

# How are addresses hashed?

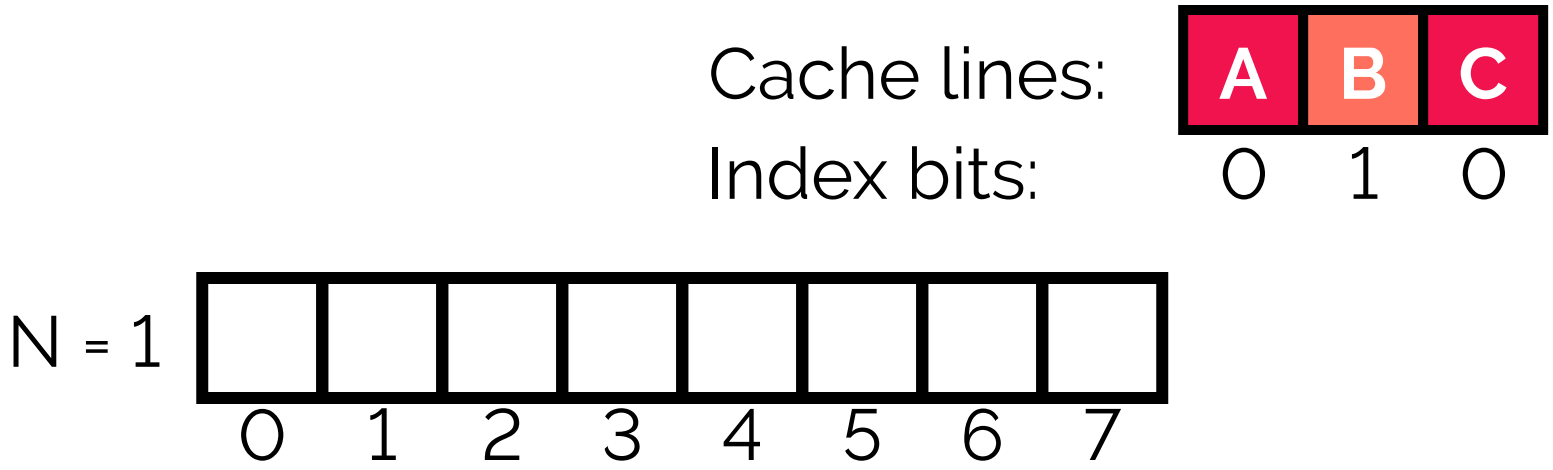

- **Offset**
  - Selects byte within a cache line
  - log2(cache line size) bits
- **Index**
  - Selects bucket within the cache
  - log2(bucket count) bits

# How are addresses hashed?

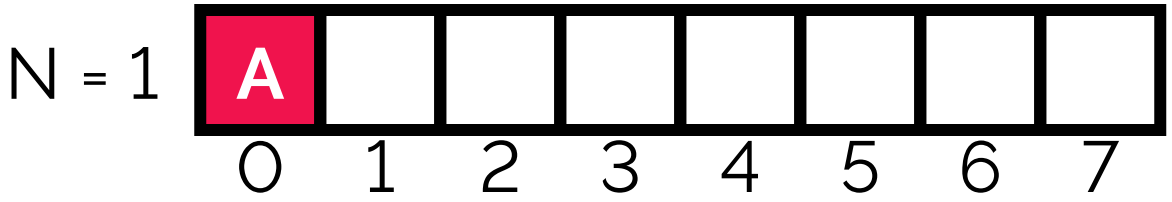

- **Offset**
  - Selects byte within a cache line
  - log2(cache line size) bits
- **Index**
  - Selects bucket within the cache
  - log2(bucket count) bits
- **Tag**
  - Used for matching

# N-way set associative cache

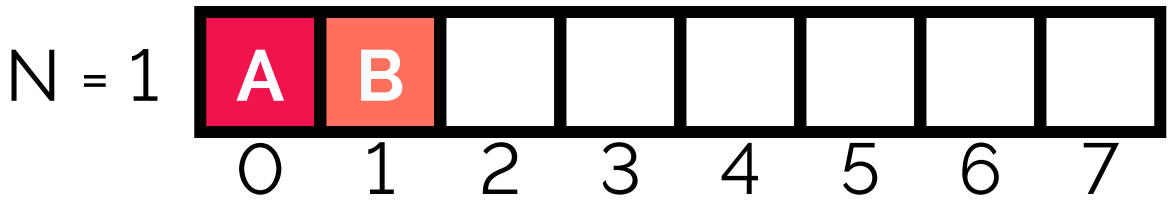

# N-way set associative cache



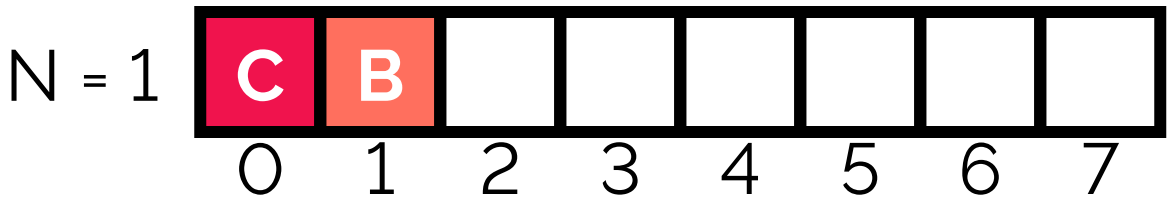
# N-way set associative cache

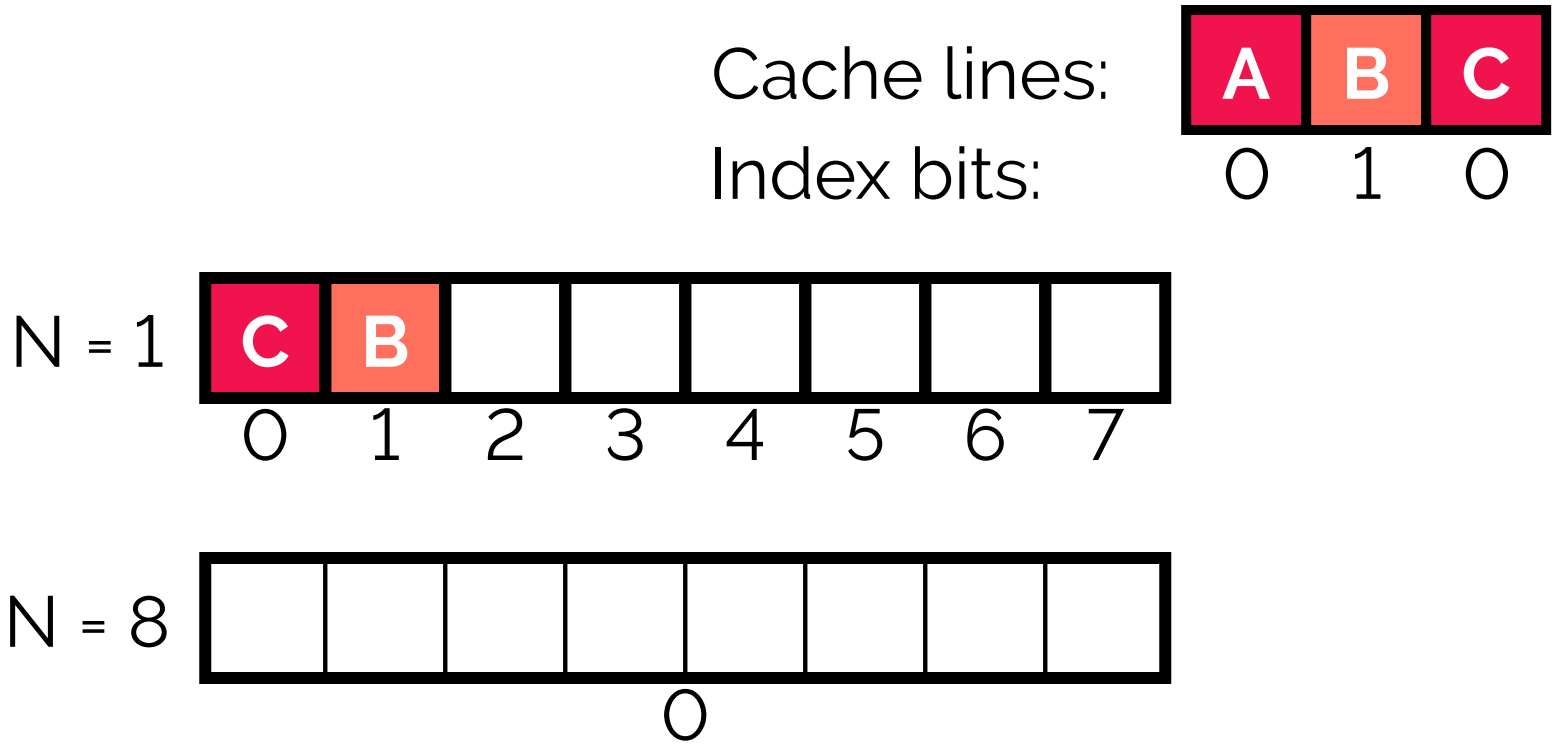
# N-way set associative cache

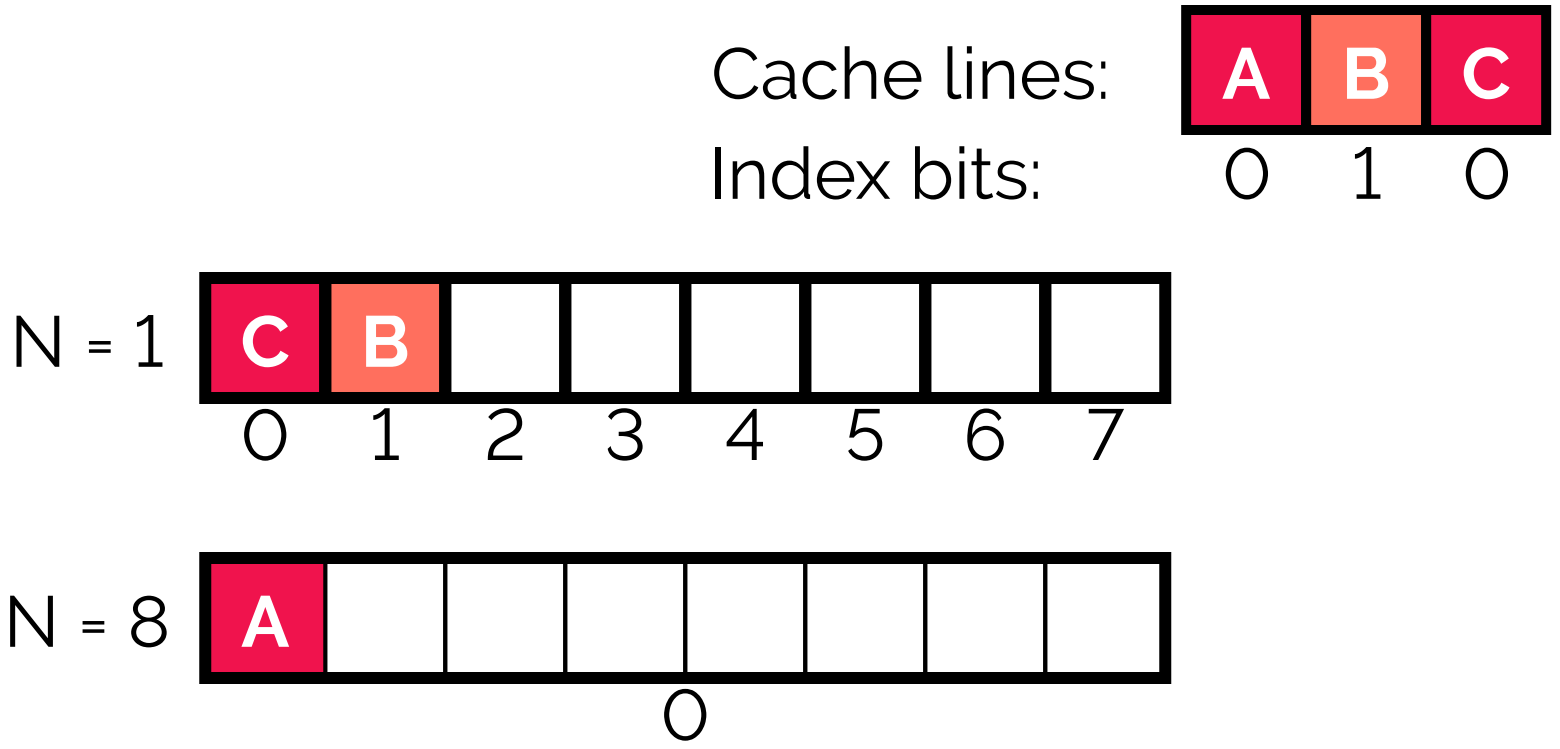
# N-way set associative cache

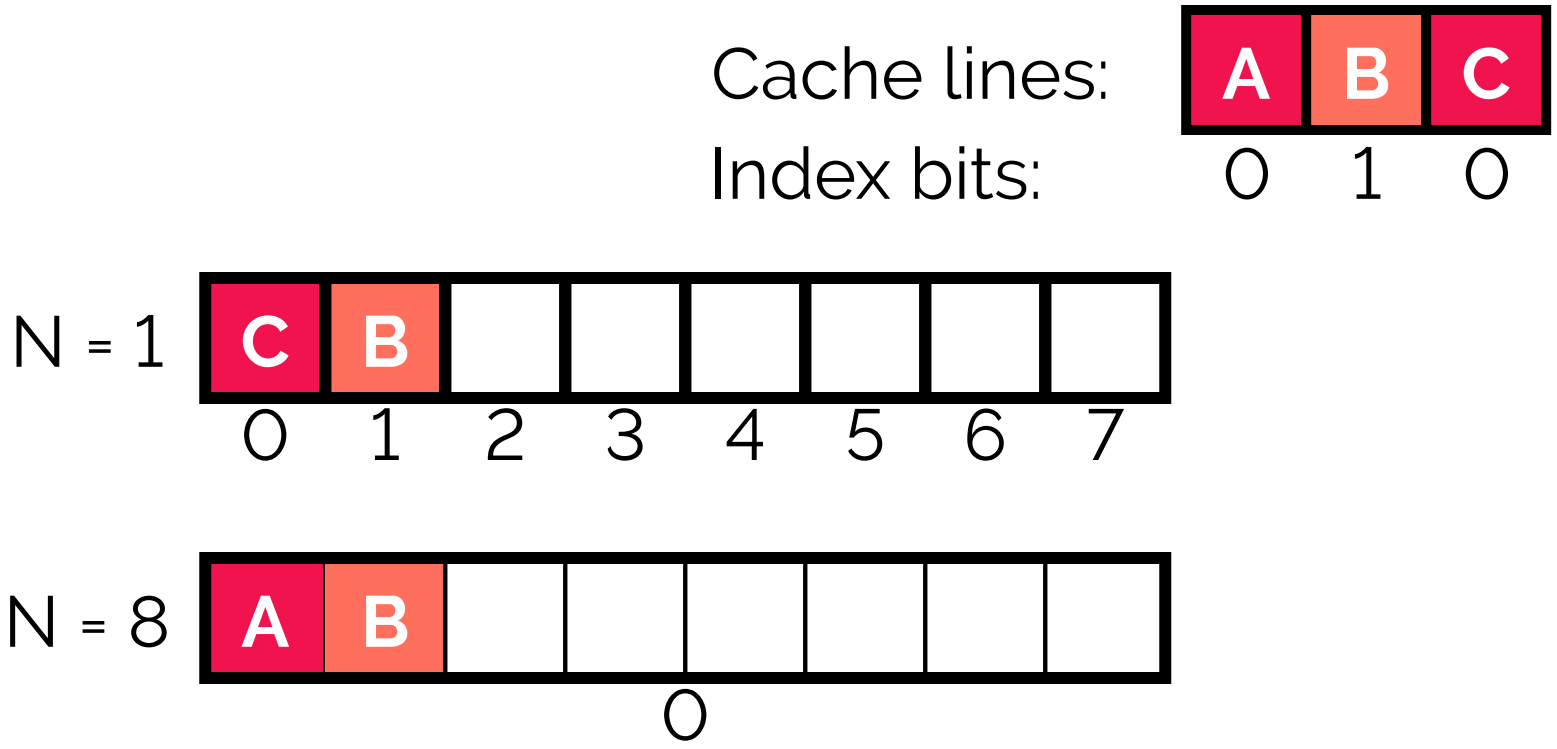
# N-way set associative cache



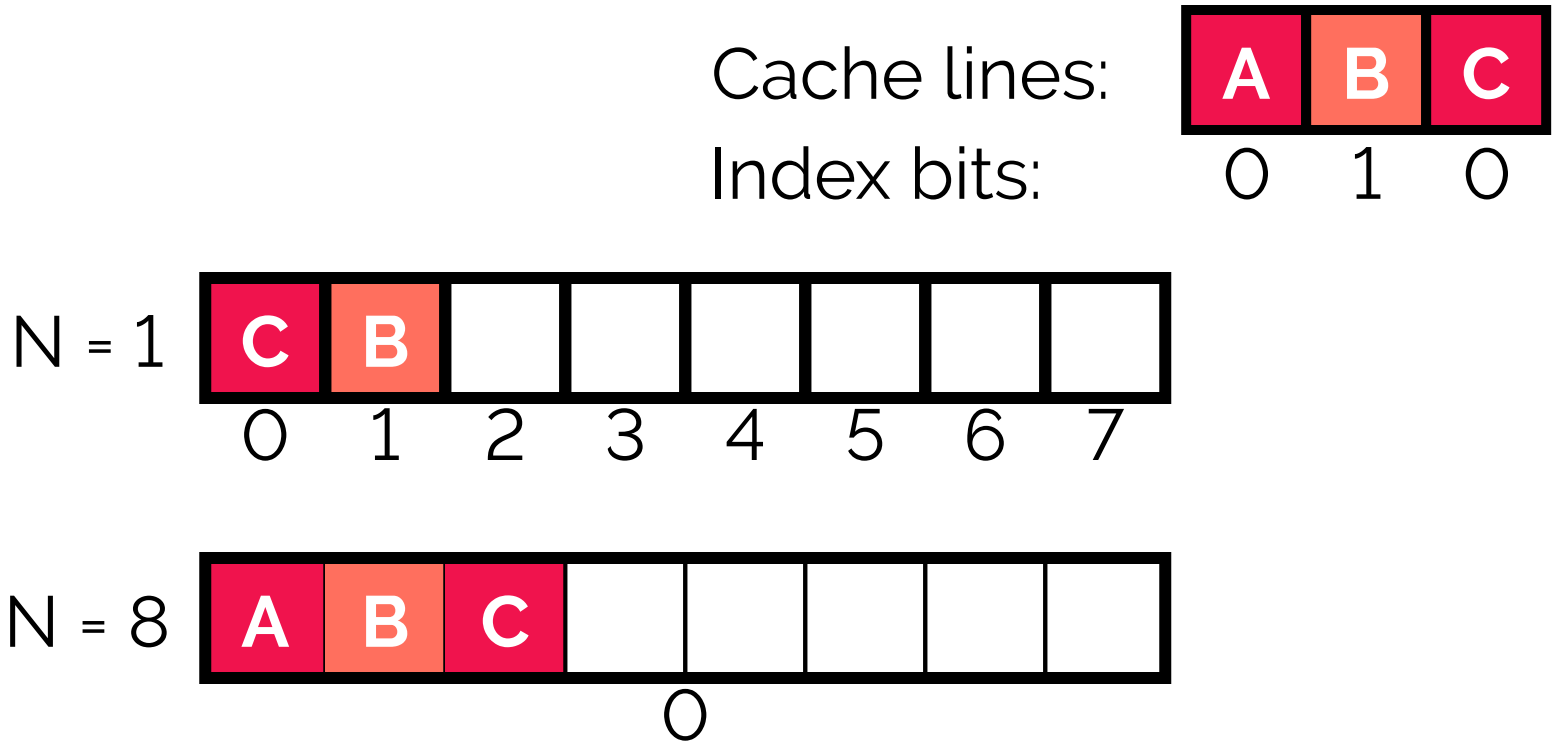
# N-way set associative cache



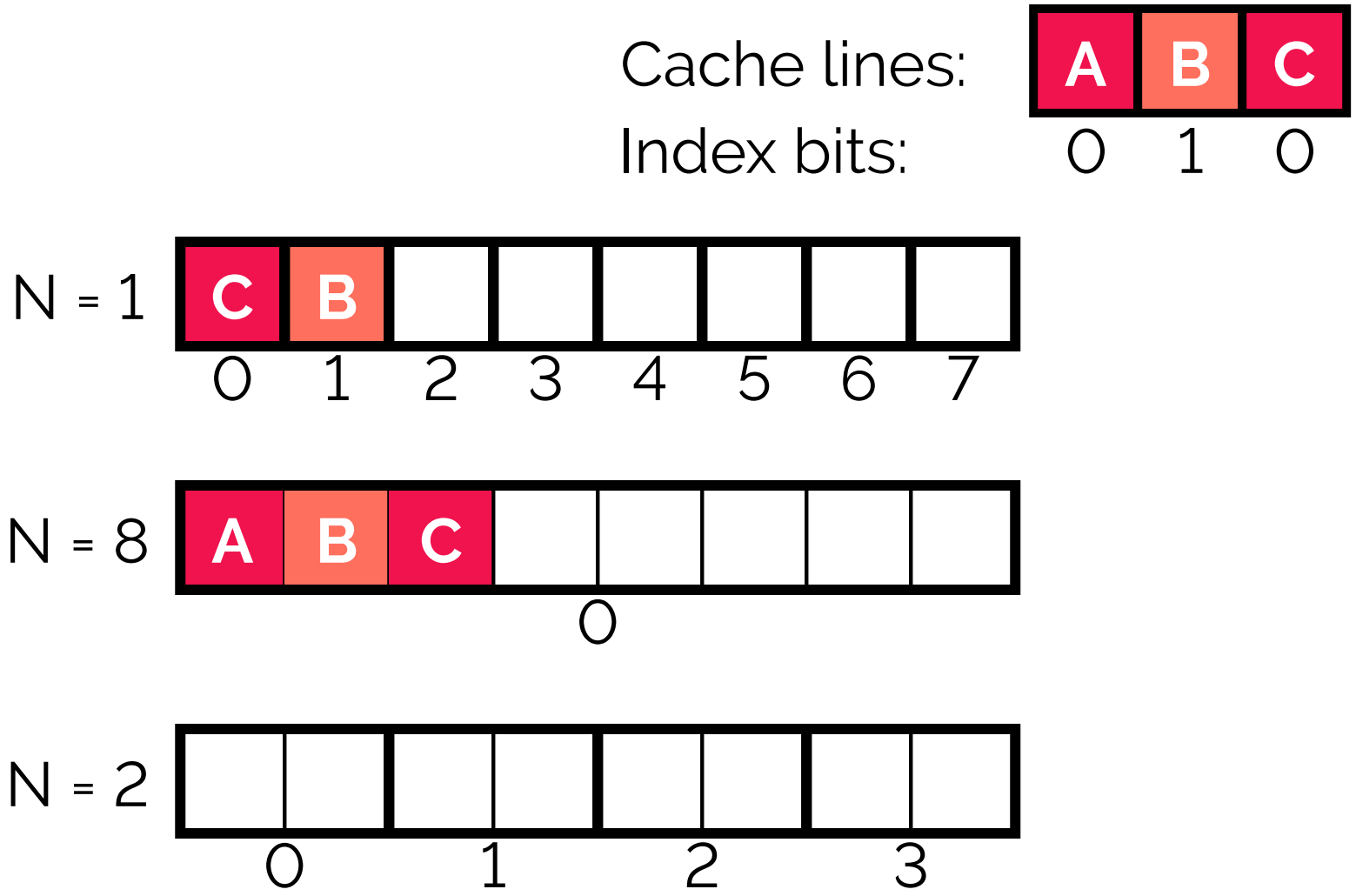
# N-way set associative cache



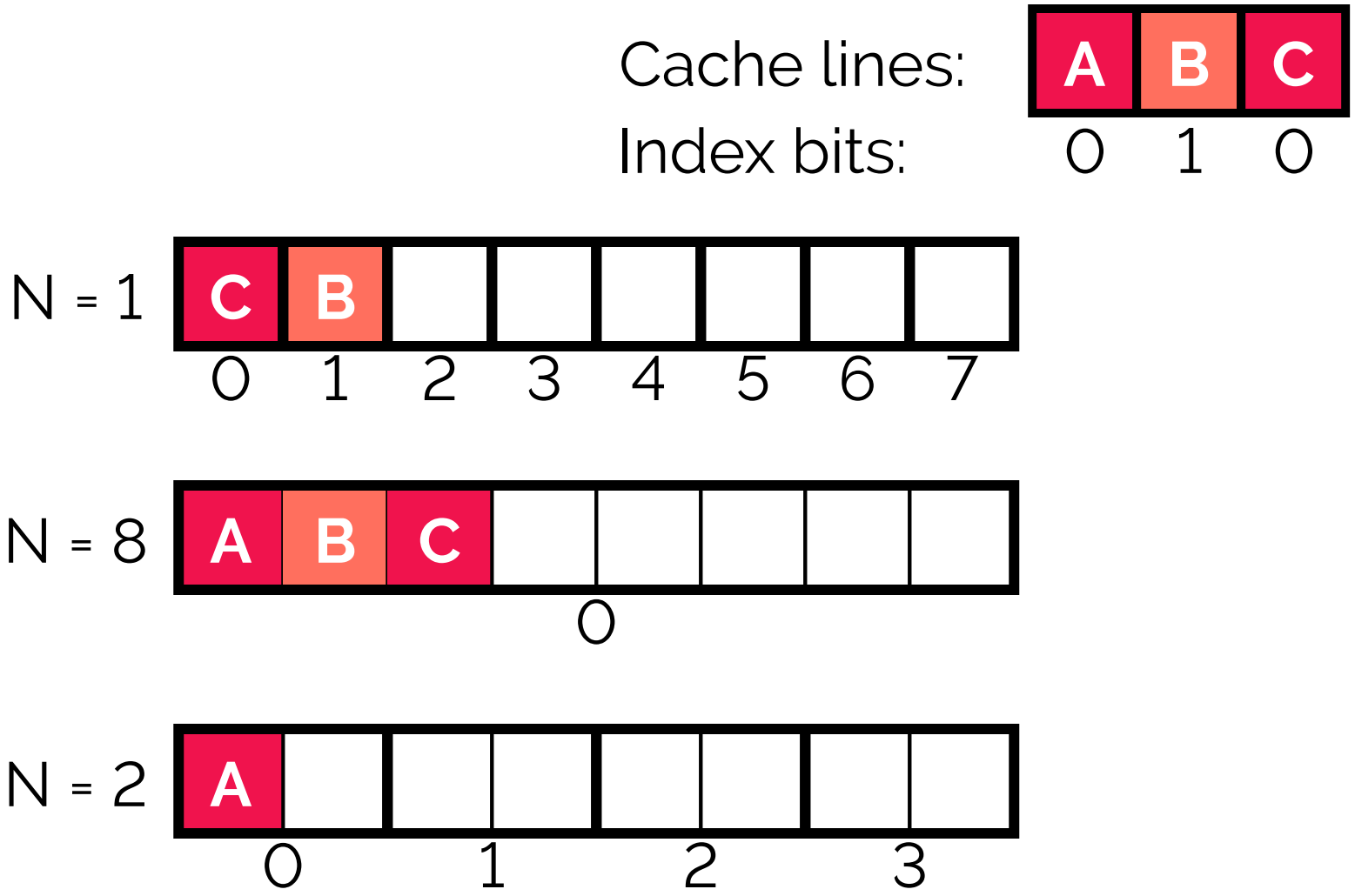
# N-way set associative cache



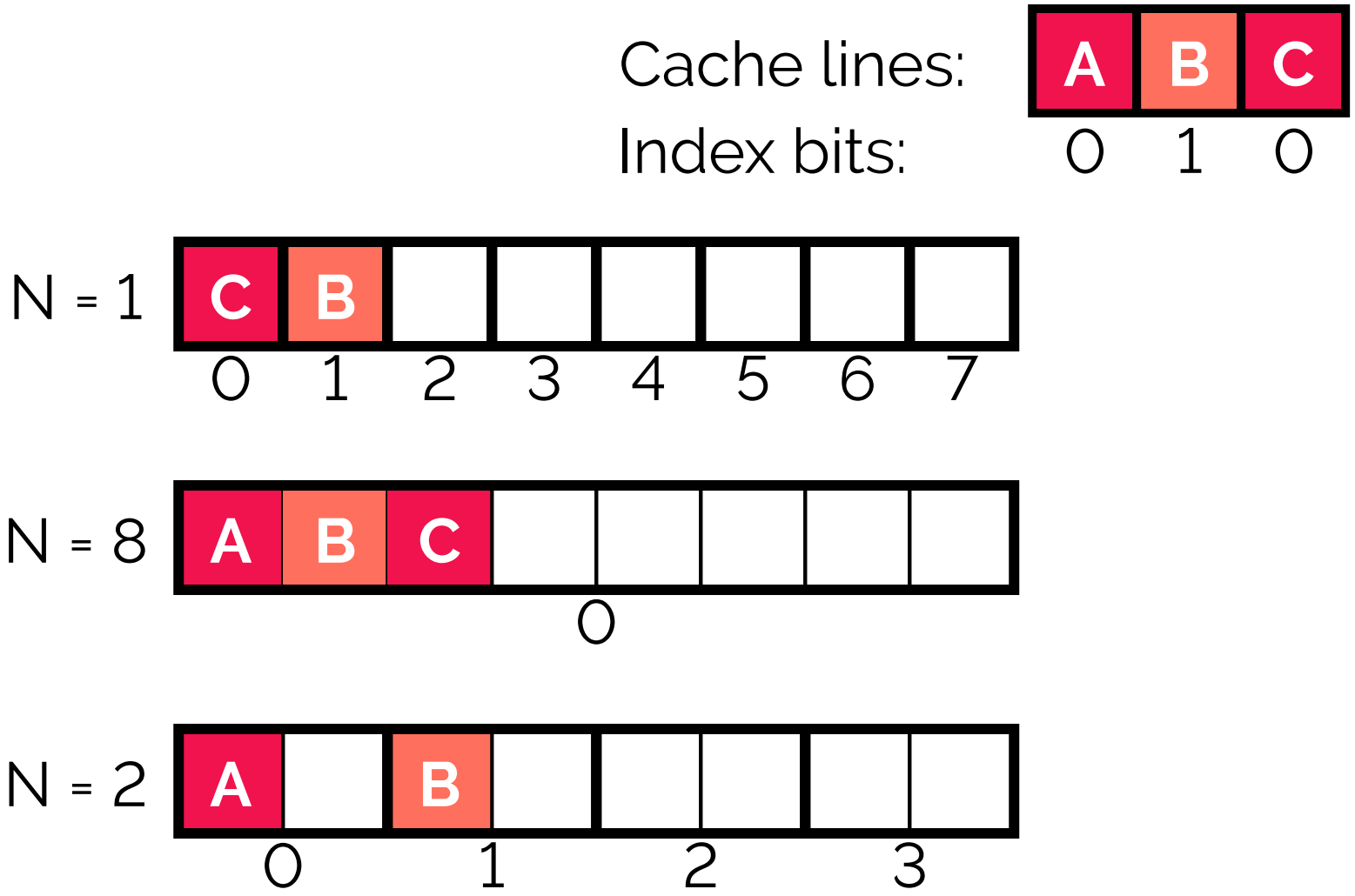
# N-way set associative cache



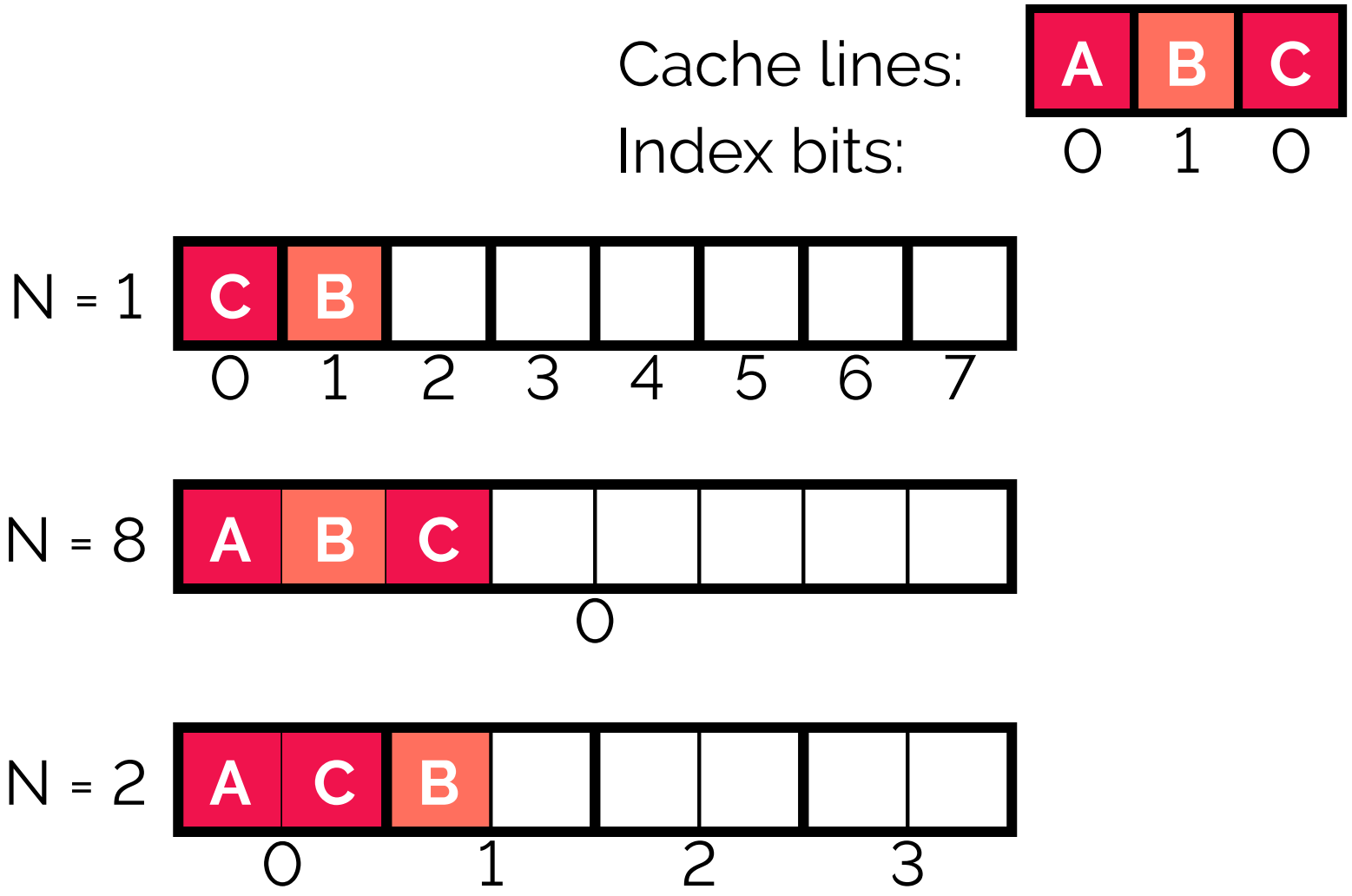
# N-way set associative cache



# N-way set associative cache



# N-way set associative cache



# Intel L1 cache

```
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
```

# Intel L1 cache

```
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
```

- **Cache line size** - 64 B (6 offset bits)

# Intel L1 cache

```
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** (N) - 8

# Intel L1 cache

```
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** (N) - 8
- **Size** - 32768 B

# Intel L1 cache

```
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** (N) - 8
- **Size** - 32768 B
- 32768 / 64 => 512 cache lines

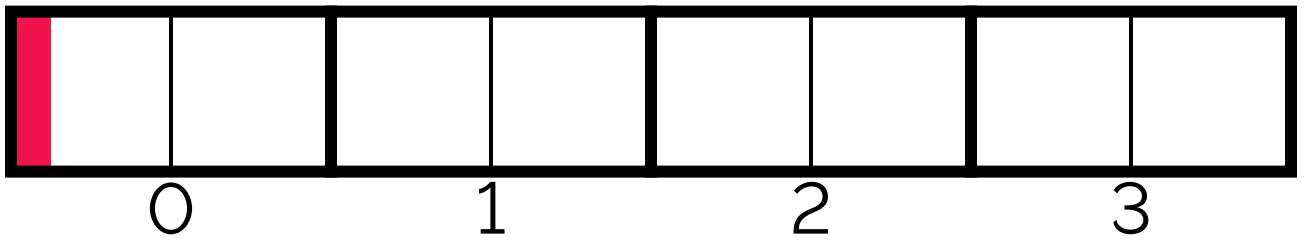
# Intel L1 cache

```
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** (N) - 8
- **Size** - 32768 B
- 32768 / 64 => 512 cache lines
- 512 / 8 => 64 buckets (6 index bits)

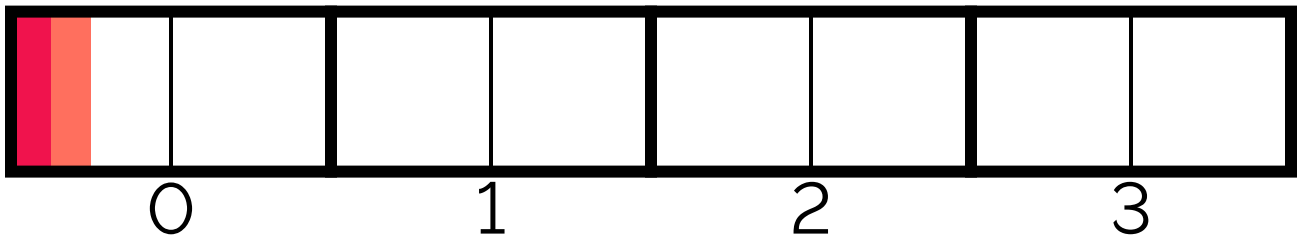
# Offset = 4B

| Number | Tag | Index | Offset |
| --- | --- | --- | --- |
| A | ..100000 | 000000 | 000000 |



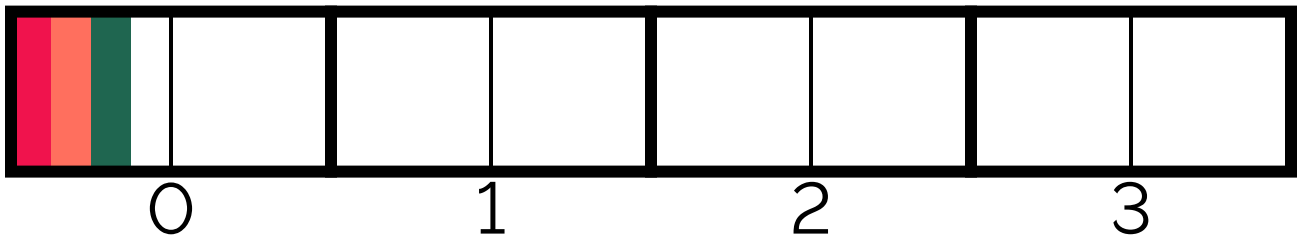

# Offset = 4B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000000 | 000100 |



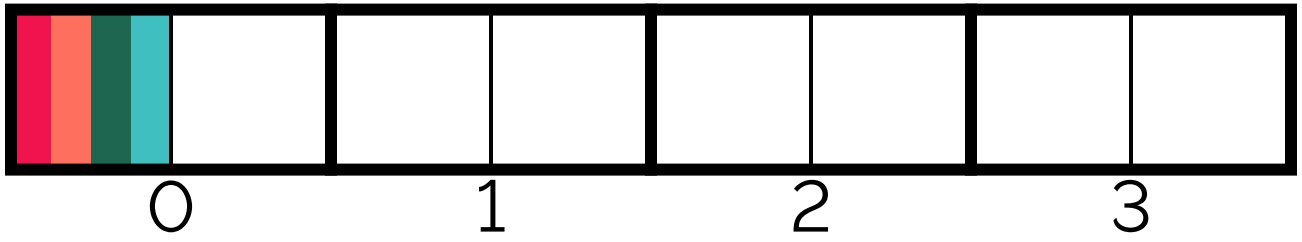

# Offset = 4B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000000 | 000100 |
| C | ..100000 | 000000 | 001000 |



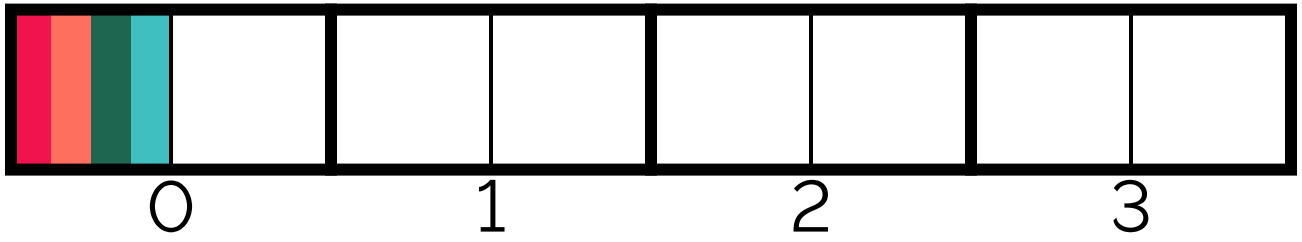
# Offset = 4B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000000 | 000100 |
| C | ..100000 | 000000 | 001000 |
| D | ..100000 | 000000 | 001100 |



# Offset = 4B

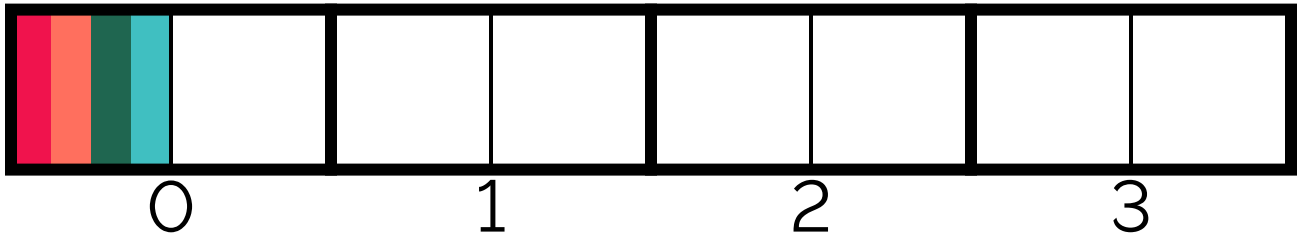
| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000000 | 000100 |
| C | ..100000 | 000000 | 001000 |
| D | ..100000 | 000000 | 001100 |




- Same bucket, same cache line for each number

# Offset = 4B

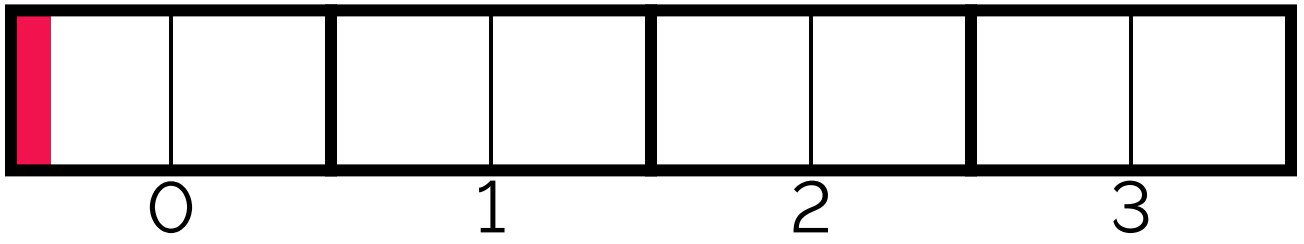
| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000000 | 000100 |
| C | ..100000 | 000000 | 001000 |
| D | ..100000 | 000000 | 001100 |




- Same bucket, same cache line for each number
- Most efficient, no space is wasted

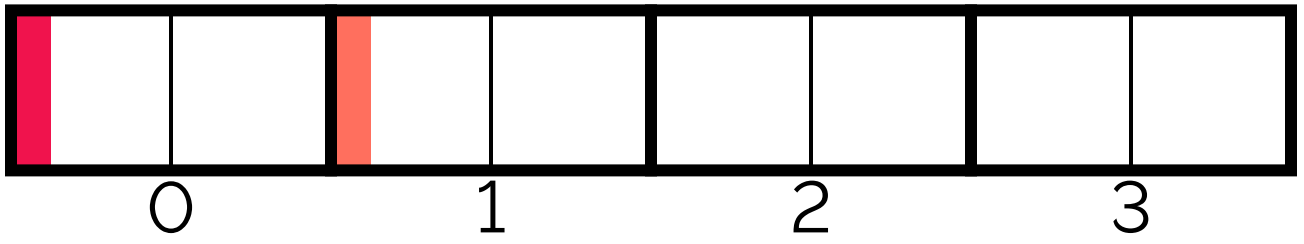
# Offset = 64B

| Number | Tag | Index | Offset |
| --- | --- | --- | --- |
| A | ..100000 | 000000 | 000000 |




# Offset = 64B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000001 | 000000 |



# Offset = 64B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000001 | 000000 |
| C | ..100000 | 000010 | 000000 |


# Offset = 64B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000001 | 000000 |
| C | ..100000 | 000010 | 000000 |
| D | ..100000 | 000011 | 000000 |

# Offset = 64B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000001 | 000000 |
| C | ..100000 | 000010 | 000000 |
| D | ..100000 | 000011 | 000000 |


- Different bucket for each number

# Offset = 64B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000001 | 000000 |
| C | ..100000 | 000010 | 000000 |
| D | ..100000 | 000011 | 000000 |


- Different bucket for each number
- Wastes 60B in each cache line

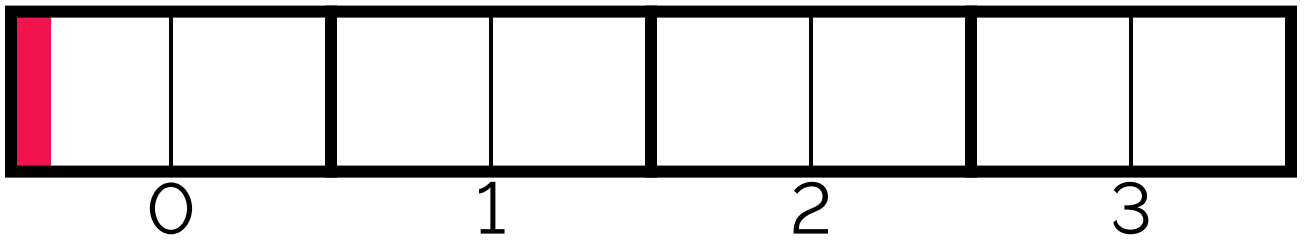
# Offset = 64B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100000 | 000001 | 000000 |
| C | ..100000 | 000010 | 000000 |
| D | ..100000 | 000011 | 000000 |


- Different bucket for each number
- Wastes 60B in each cache line
- Equally distributed among buckets

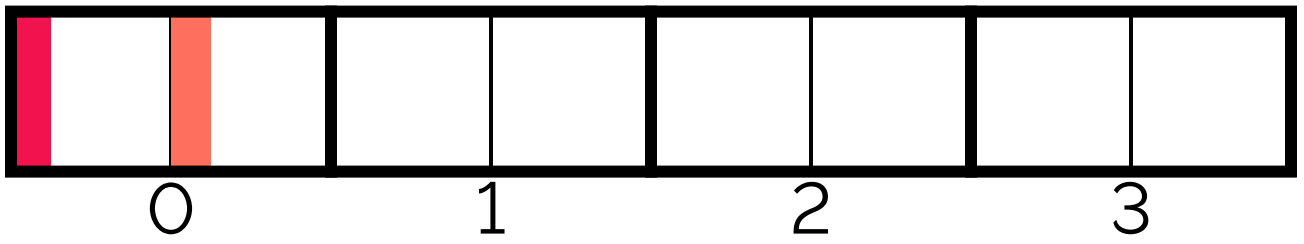
# Offset = 4096B

| Number | Tag | Index | Offset |
| --- | --- | --- | --- |
| A | ..100000 | 000000 | 000000 |



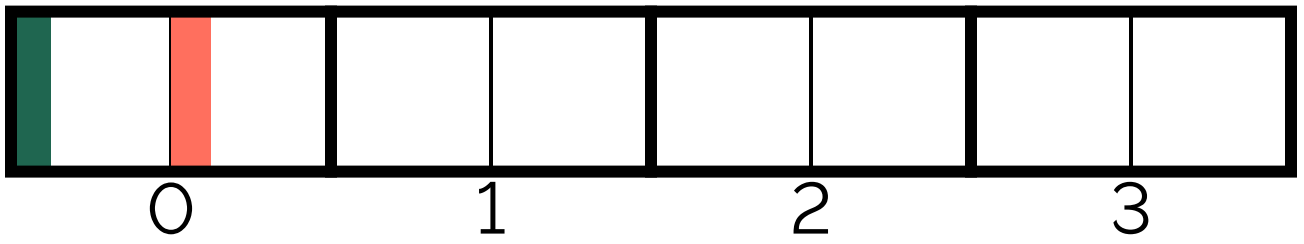

# Offset = 4096B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100001 | 000000 | 000000 |



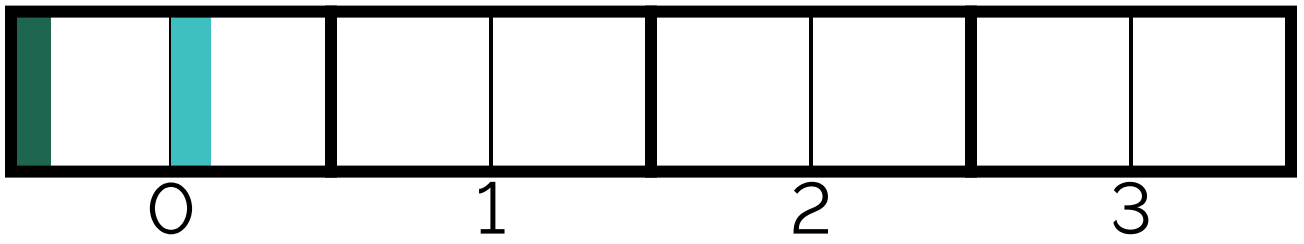

# Offset = 4096B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100001 | 000000 | 000000 |
| C | ..100010 | 000000 | 000000 |



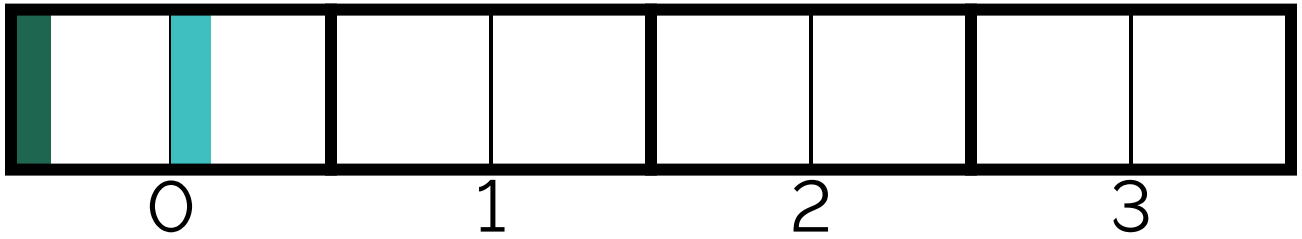
# Offset = 4096B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100001 | 000000 | 000000 |
| C | ..100010 | 000000 | 000000 |
| D | ..100011 | 000000 | 000000 |



# Offset = 4096B

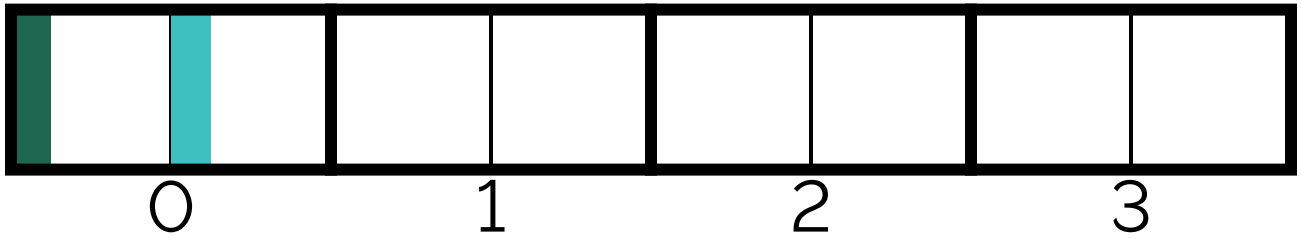
| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100001 | 000000 | 000000 |
| C | ..100010 | 000000 | 000000 |
| D | ..100011 | 000000 | 000000 |




- Same bucket, but different cache lines for each number!

# Offset = 4096B

| Number | Tag | Index | Offset |
|---|---|---|---|
| A | ..100000 | 000000 | 000000 |
| B | ..100001 | 000000 | 000000 |
| C | ..100010 | 000000 | 000000 |
| D | ..100011 | 000000 | 000000 |




- Same bucket, but different cache lines for each number!
- Bucket full => evictions necessary

# How to measure?

`l1d.replacement`

How many times was a cache line loaded into L1?

# How to measure?

## `l1d.replacement`

How many times was a cache line loaded into L1?

```
$ perf stat -e l1d.replacement ./example1
4B    offset ->      149 558
4096B offset -> 426 218 383
```

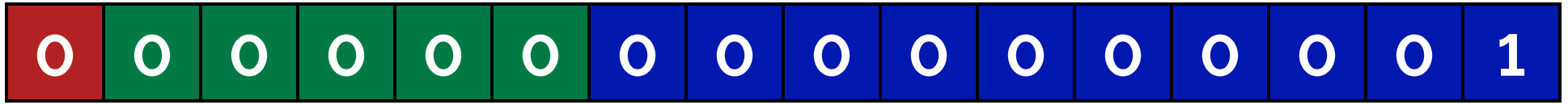
# Code (backup)

```
float F = static_cast<float>(std::stof(argv[1]));
std::vector<float> data(4 * 1024 * 1024, 1);

for (int r = 0; r < 100; r++)
{
    for (auto& item: data)
    {
        item *= F;
    }
}
```

# Result (backup)

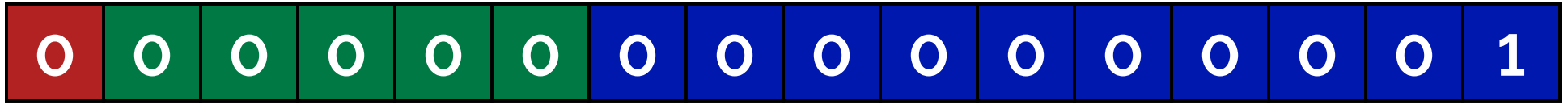
```
> time -p ./example2 0
real 0,12
user 0,12
sys 0,00
> time -p ./example2 0.1
real 0,47
user 0,46
sys 0,00
> time -p ./example2 0.3
real 0,70
user 0,69
sys 0,00
```

# Denormal floating point numbers



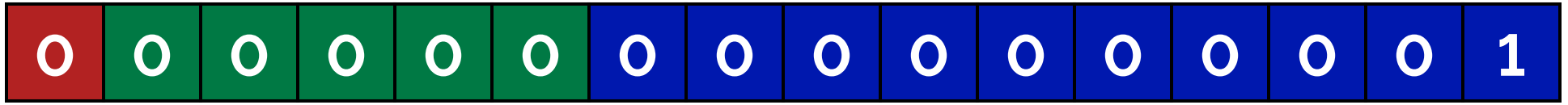

$$(-1)^{0} * 2^{00000-01110} * 0.0000000001$$

# Denormal floating point numbers




$$(-1)^{0} * 2^{00000-\mathbf{01110}} * \mathbf{0}.0000000001$$

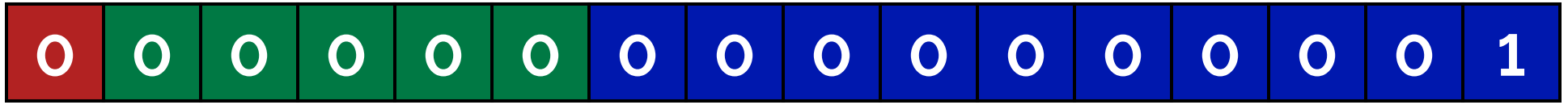
# Denormal floating point numbers




$$(-1)^{0} * 2^{00000-\mathbf{01110}} * \mathbf{0}.0000000001$$

- Numbers close to zero
- Hidden bit = 0, smaller bias

# Denormal floating point numbers

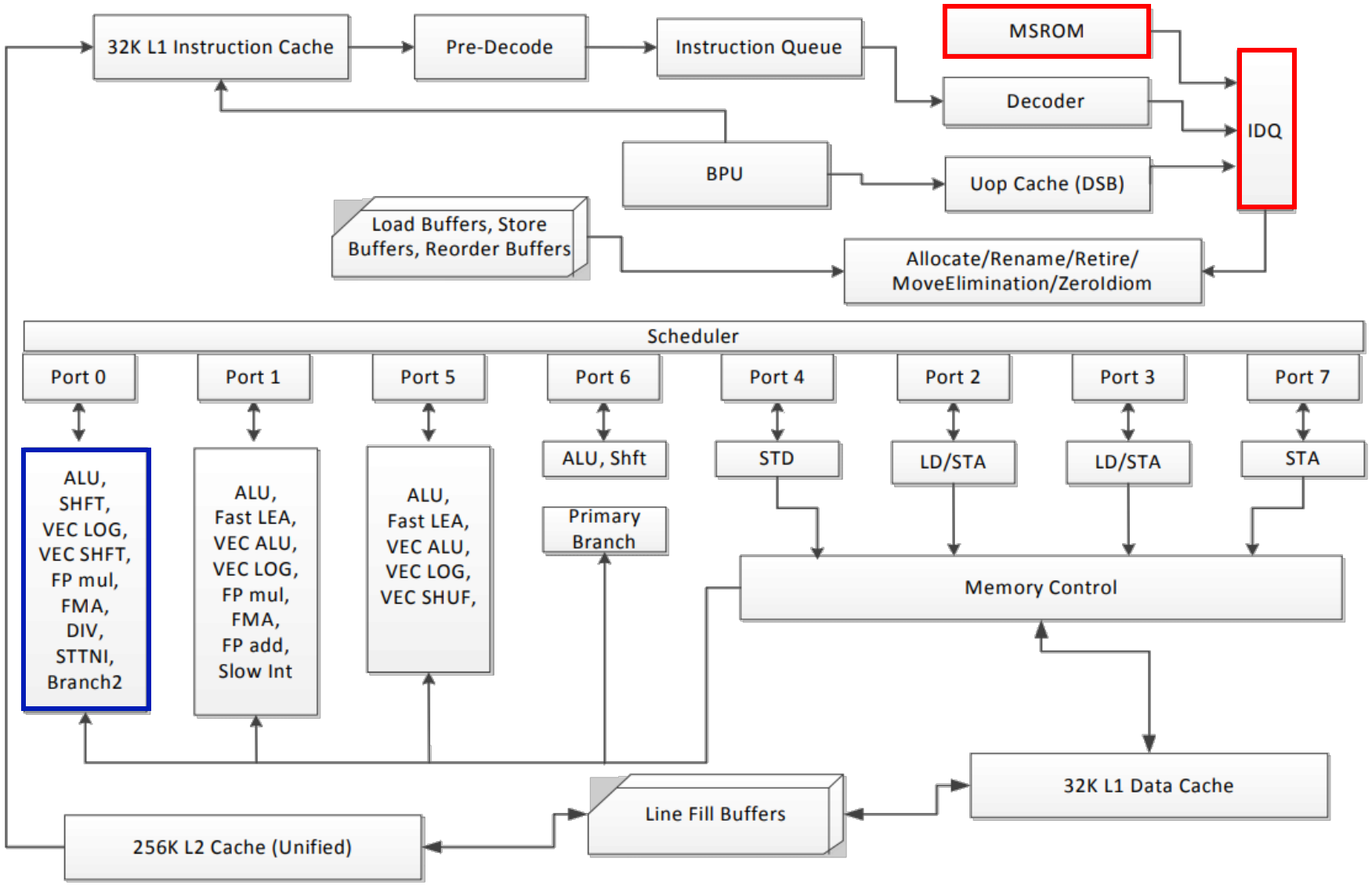



$$(-1)^{0} * 2^{00000-\mathbf{01110}} * \mathbf{0}.0000000001$$

- Numbers close to zero
- Hidden bit = 0, smaller bias

Operations on denormal numbers are slow!

# Floating point handling



# How to measure?

`fp_assist.any`

How many times the CPU switched to the microcode FP handler?

# How to measure?

## `fp_assist.any`

How many times the CPU switched to the microcode FP handler?

```
$ perf stat -e fp_assist.any ./example2
0    ->              0
0.3 -> 15 728 640
```

# How to fix it?

- The nuclear option: `-ffast-math`
  - Sacrifice correctness to gain more FP performance

# How to fix it?

- The nuclear option: `-ffast-math`
  - Sacrifice correctness to gain more FP performance
- Set CPU flags:
  - Flush-to-zero - treat denormal outputs as 0
  - Denormals-to-zero - treat denormal inputs as 0

# How to fix it?

- The nuclear option: `-ffast-math`
  - Sacrifice correctness to gain more FP performance
- Set CPU flags:
  - Flush-to-zero - treat denormal outputs as 0
  - Denormals-to-zero - treat denormal inputs as 0

```
_mm_setcsr(_mm_getcsr() | 0x8040);
// or
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
```

# There are many other effects

- NUMA
- 4k aliasing
- Misaligned accesses, cache line boundaries
- Instruction data dependencies
- Software prefetching
- Non-temporal stores & cache pollution
- Bandwidth saturation
- DRAM refresh intervals
- AVX/SSE transition penalty
- ...

# Thank you!

For more examples visit:
`github.com/kobzol/hardware-effects`

Jakub Beránek

SlideDeck built with `github.com/spirali/elsie`

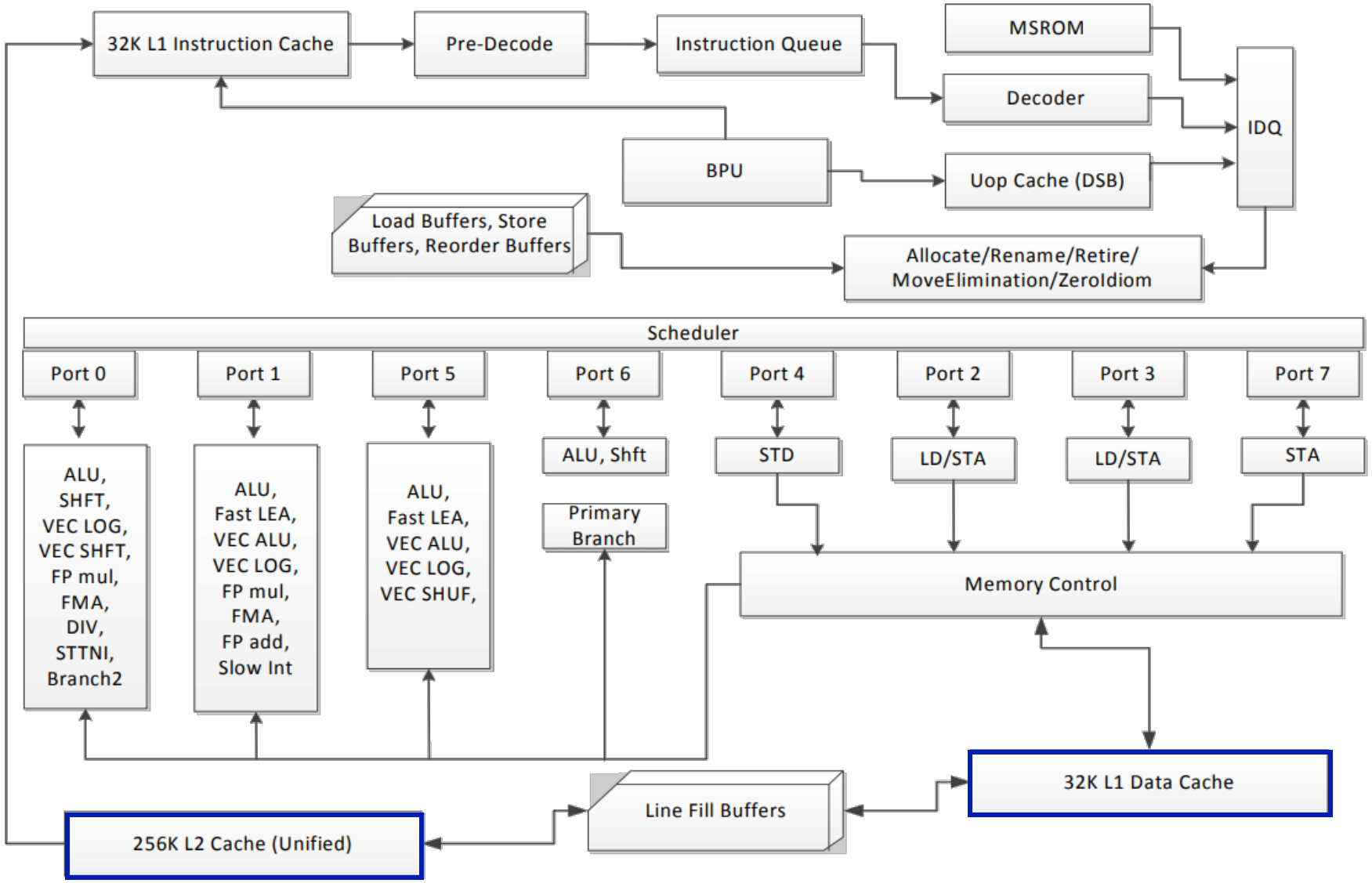
# Code (backup)

```
// tid - [0, NO_OF_THREADS)
void thread_fn(int tid, double* data)
{
    size_t repetitions = 1024 * 1024 * 1024UL;
    for (size_t i = 0; i < repetitions; i++)
    {
        data[tid] *= i;
    }
}
```

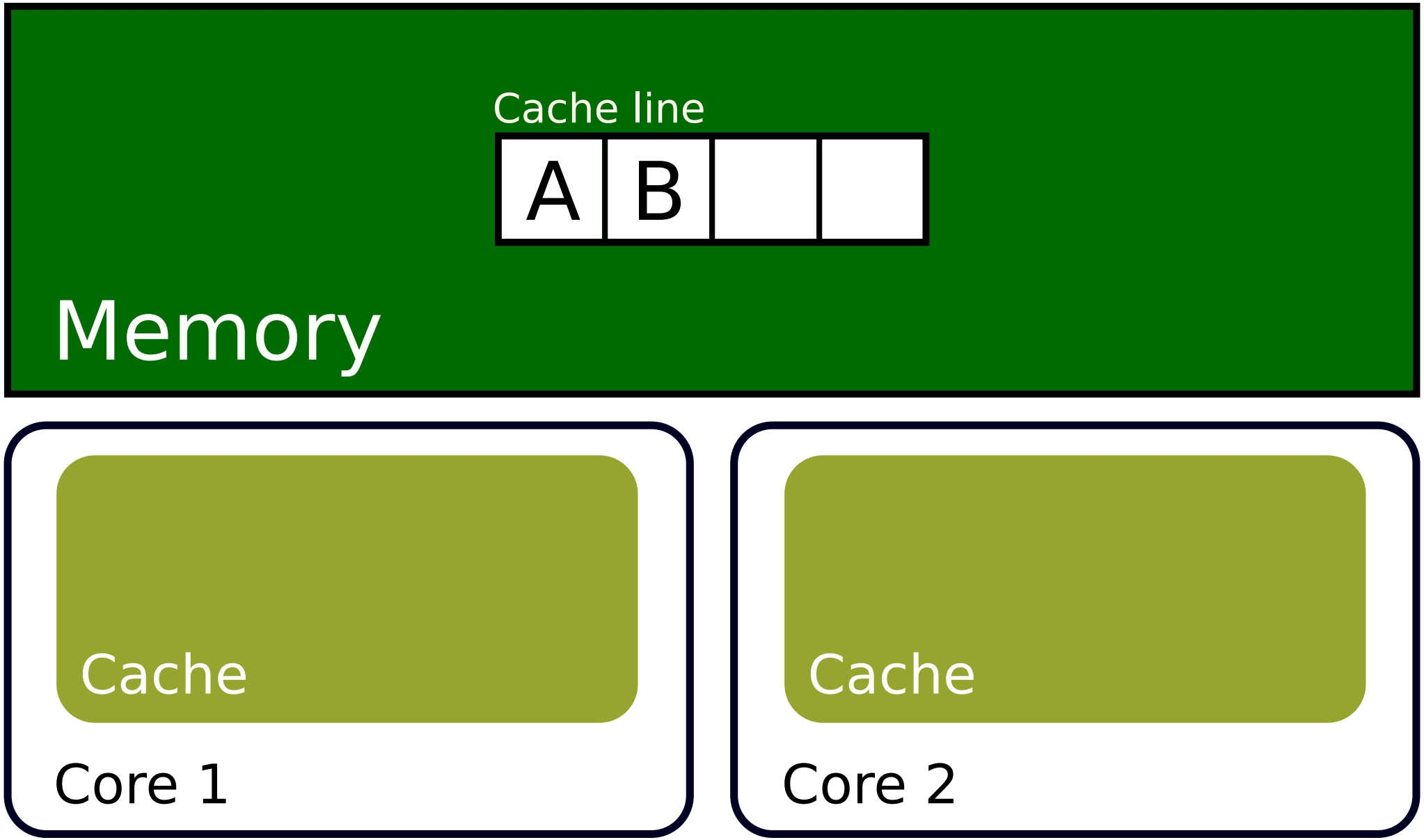
# Result (backup)

```
> time -p ./example2 1
real 2,60
user 2,60
sys 0,00
> time -p ./example2 2
real 2,92
user 5,83
sys 0,00
> time -p ./example2 3
real 3,15
user 9,04
sys 0,01
> time -p ./example2 4
real 3,39
user 13,44
sys 0,00
```

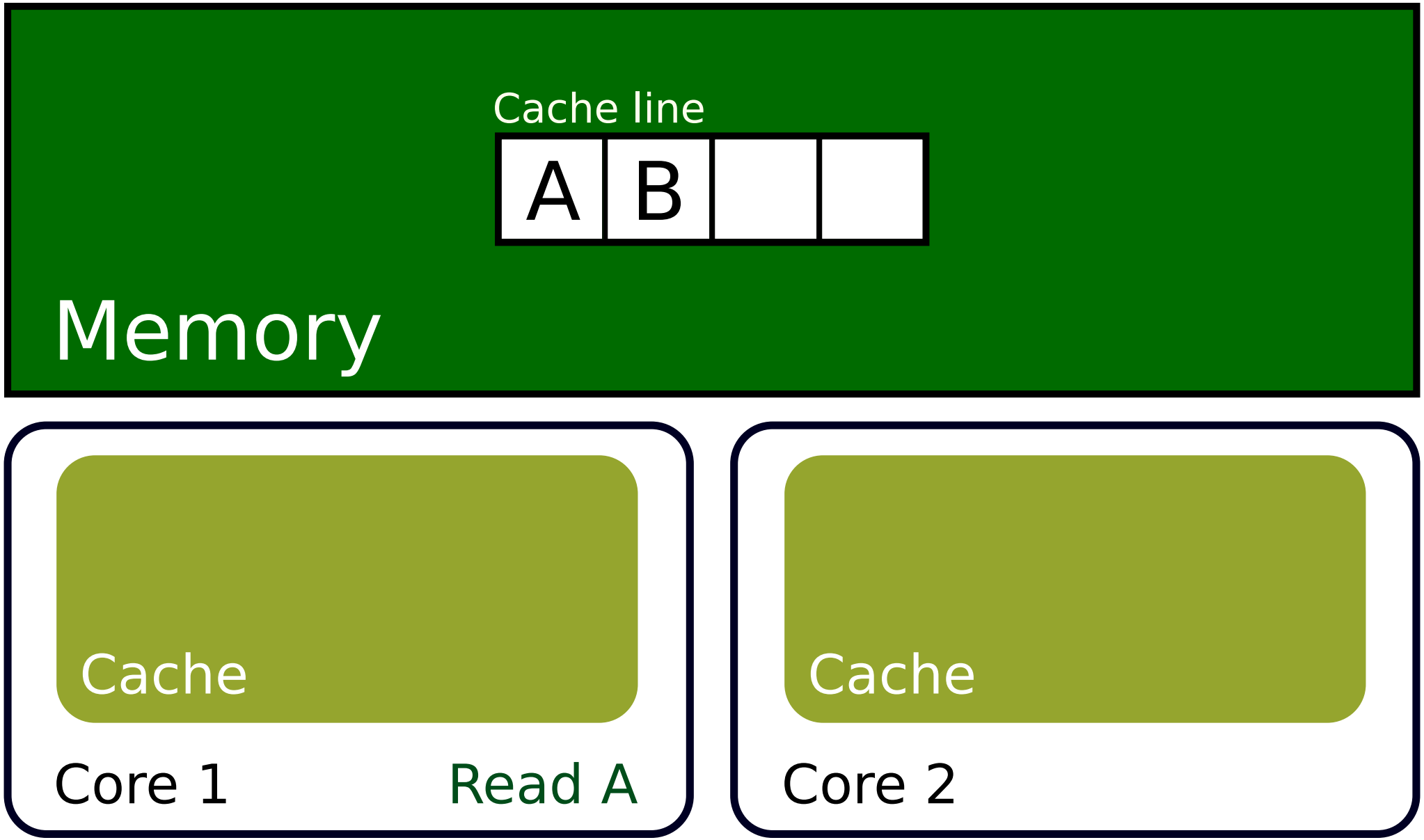
# Cache system



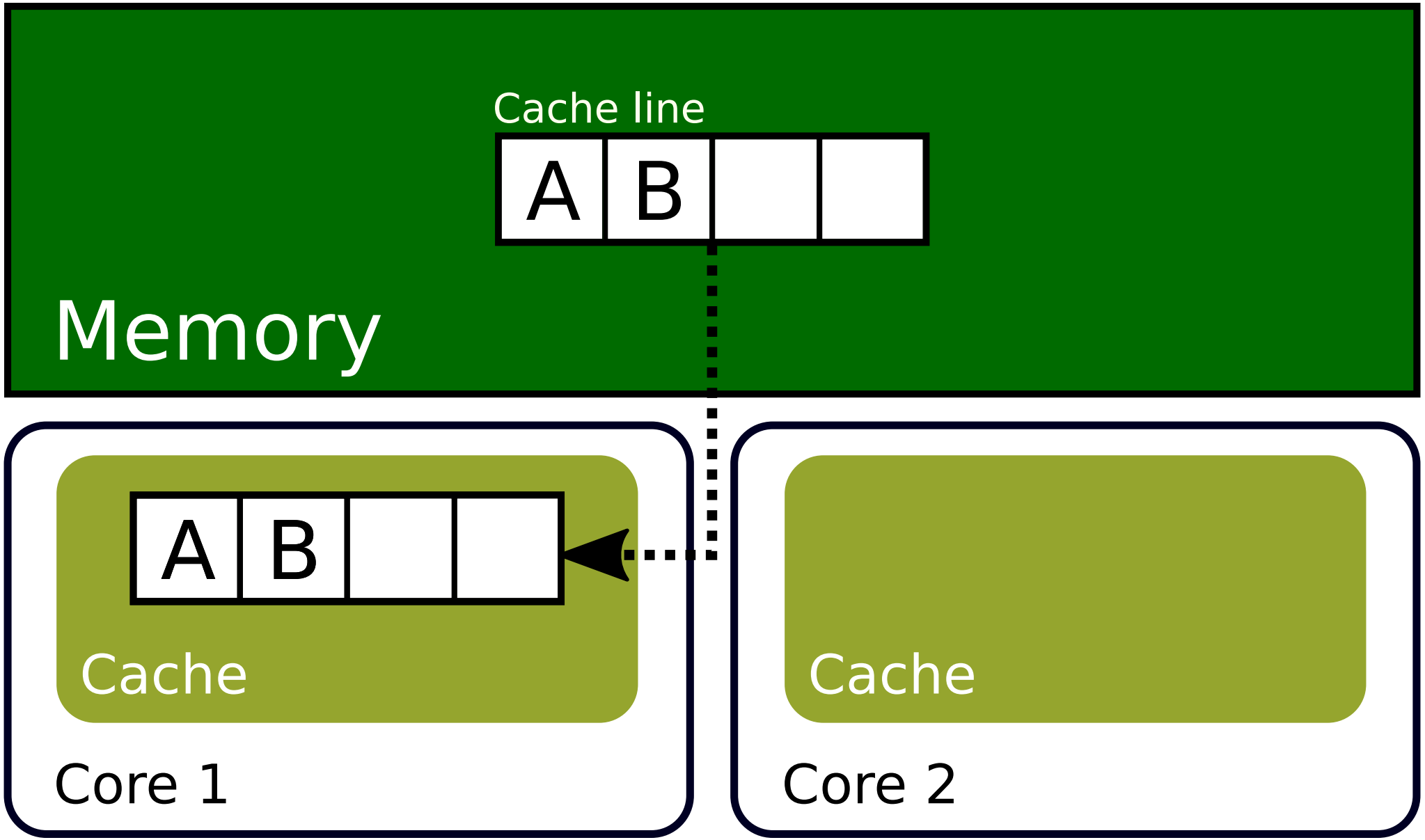
# Cache coherency



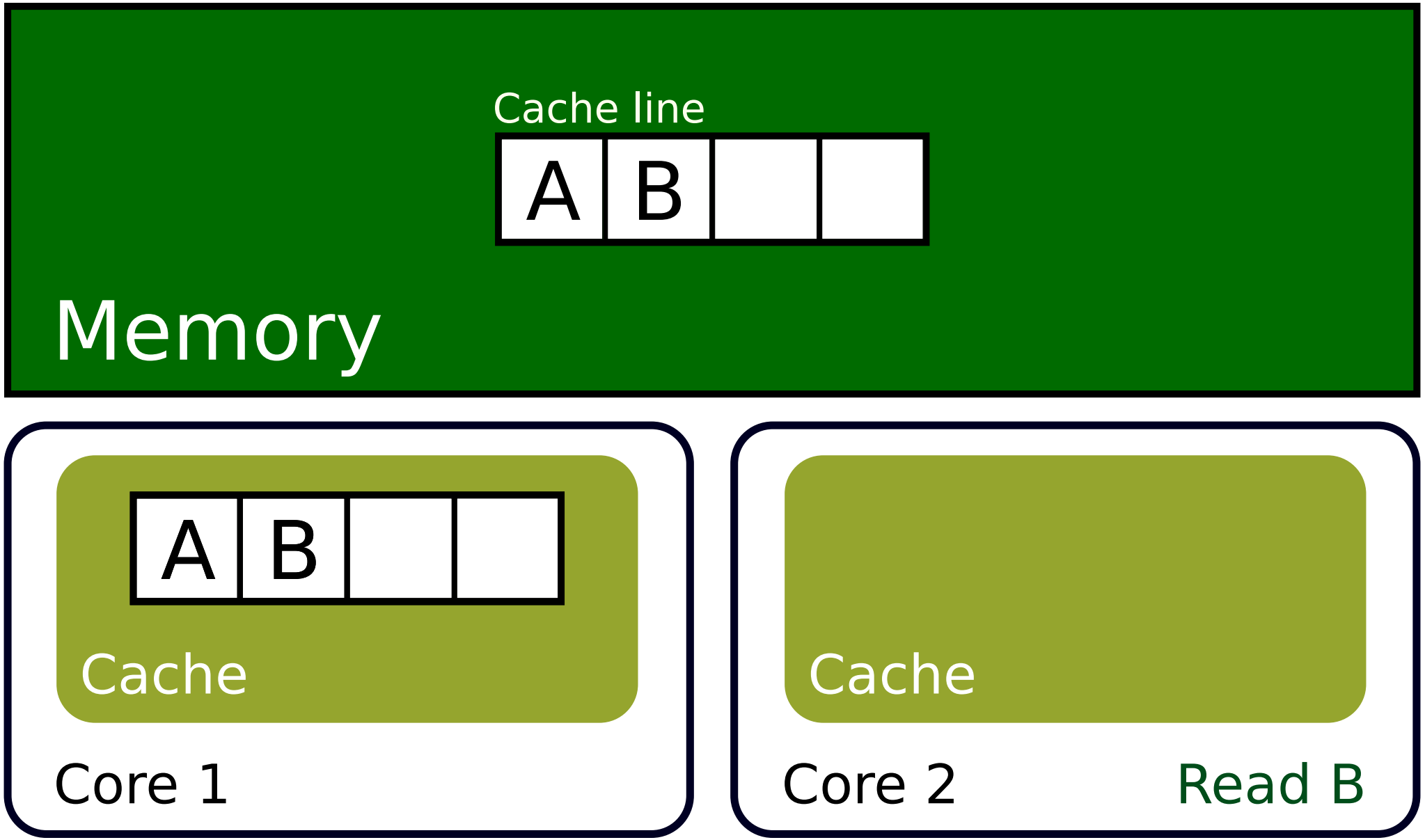
# Cache coherency



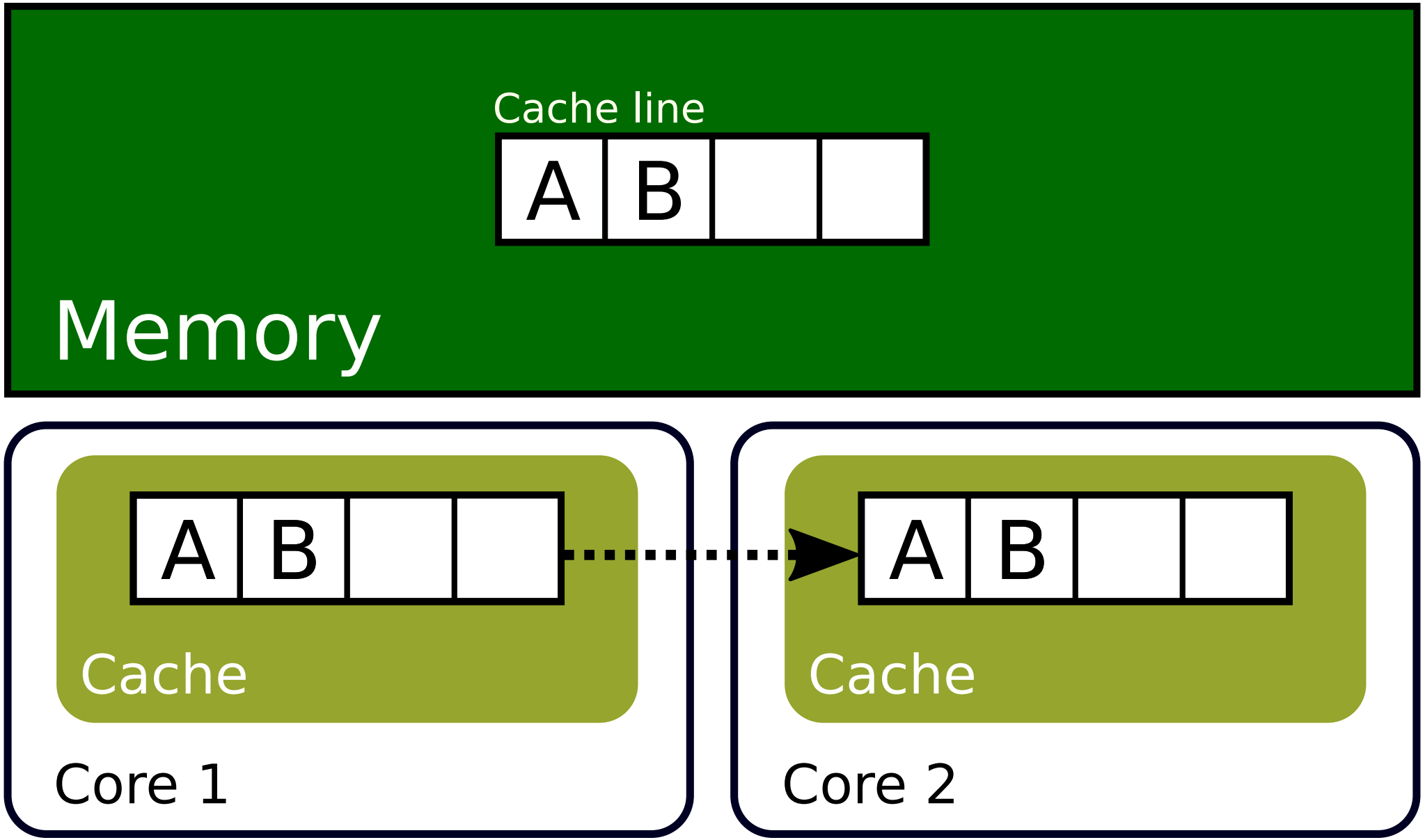
# Cache coherency



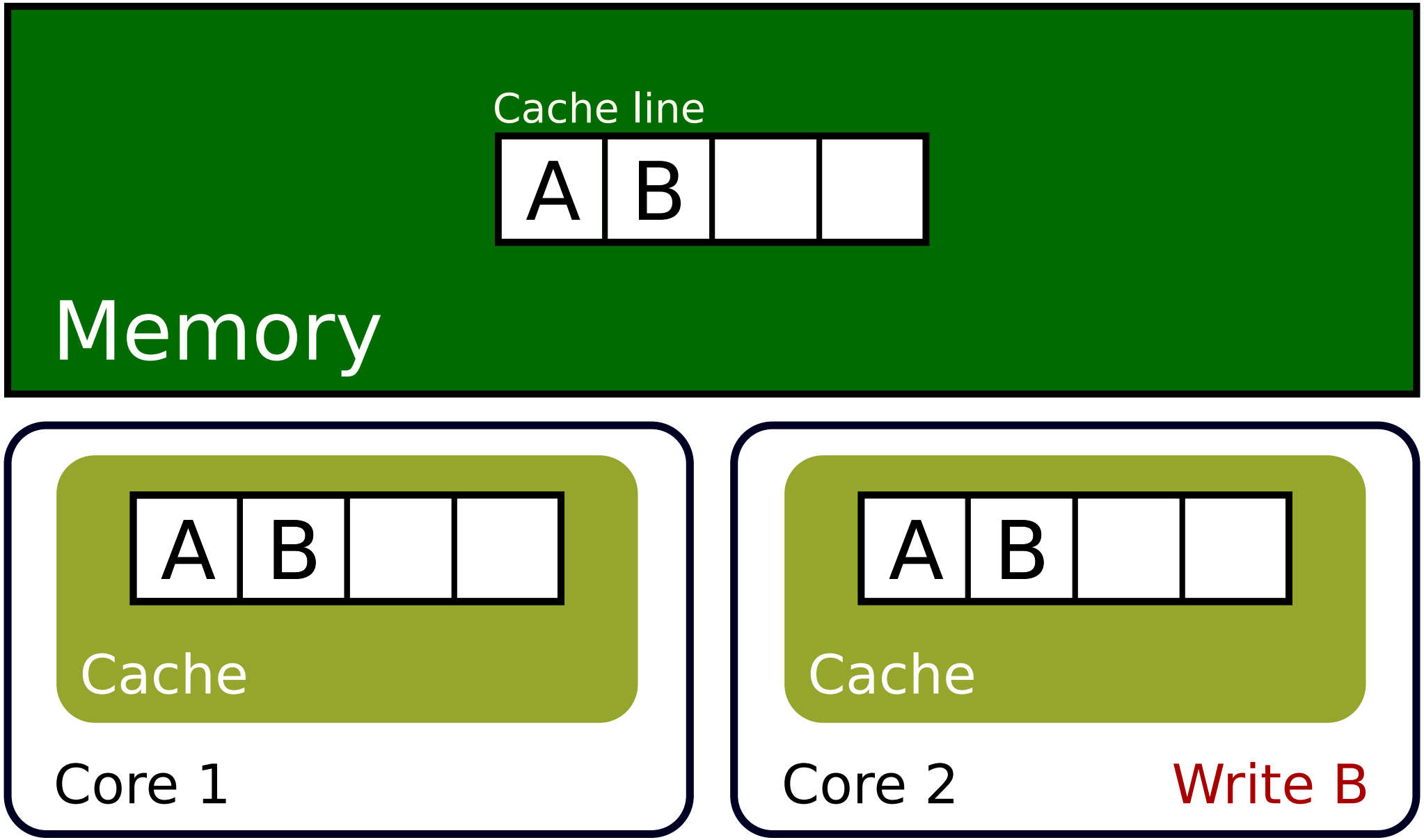
# Cache coherency



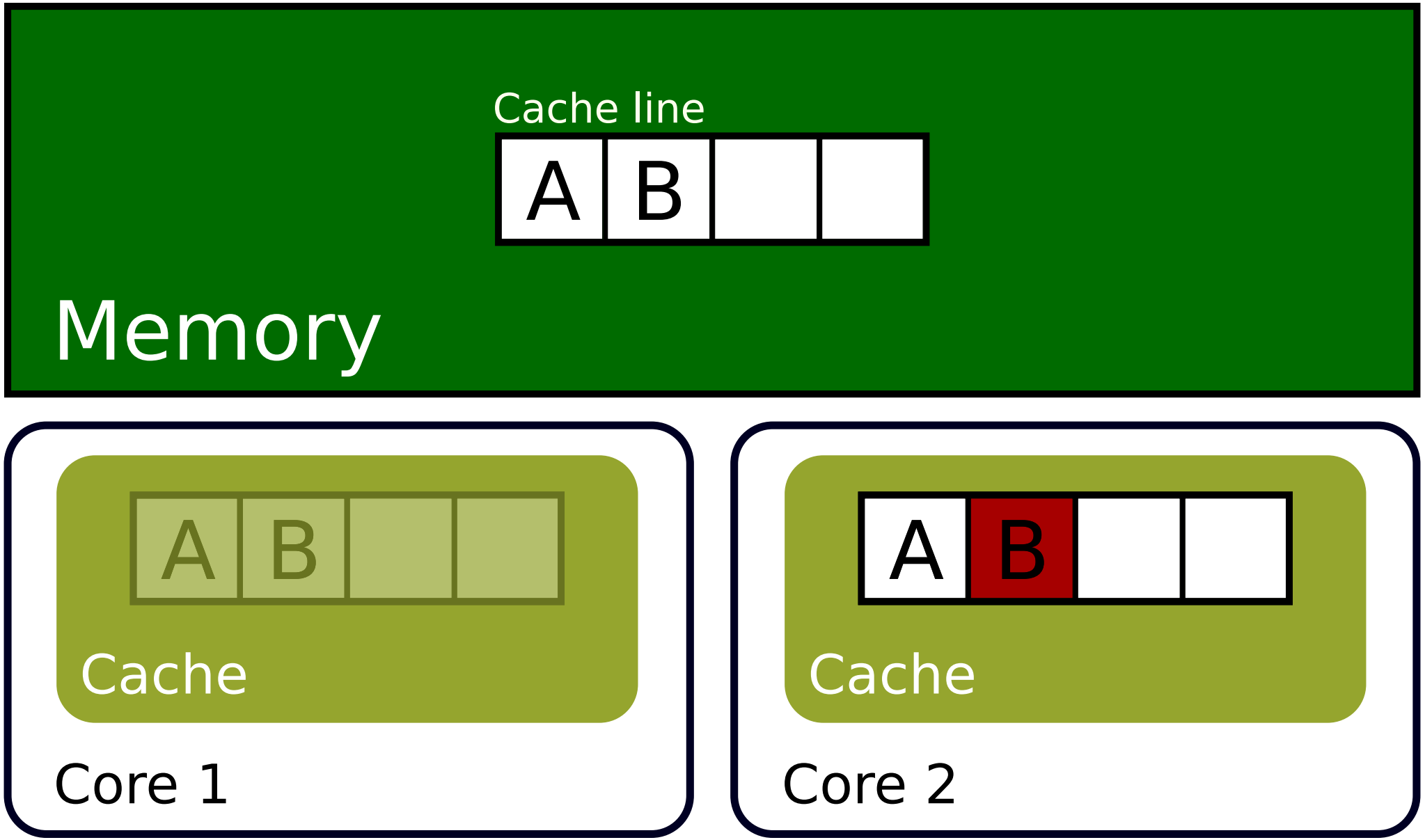
# Cache coherency



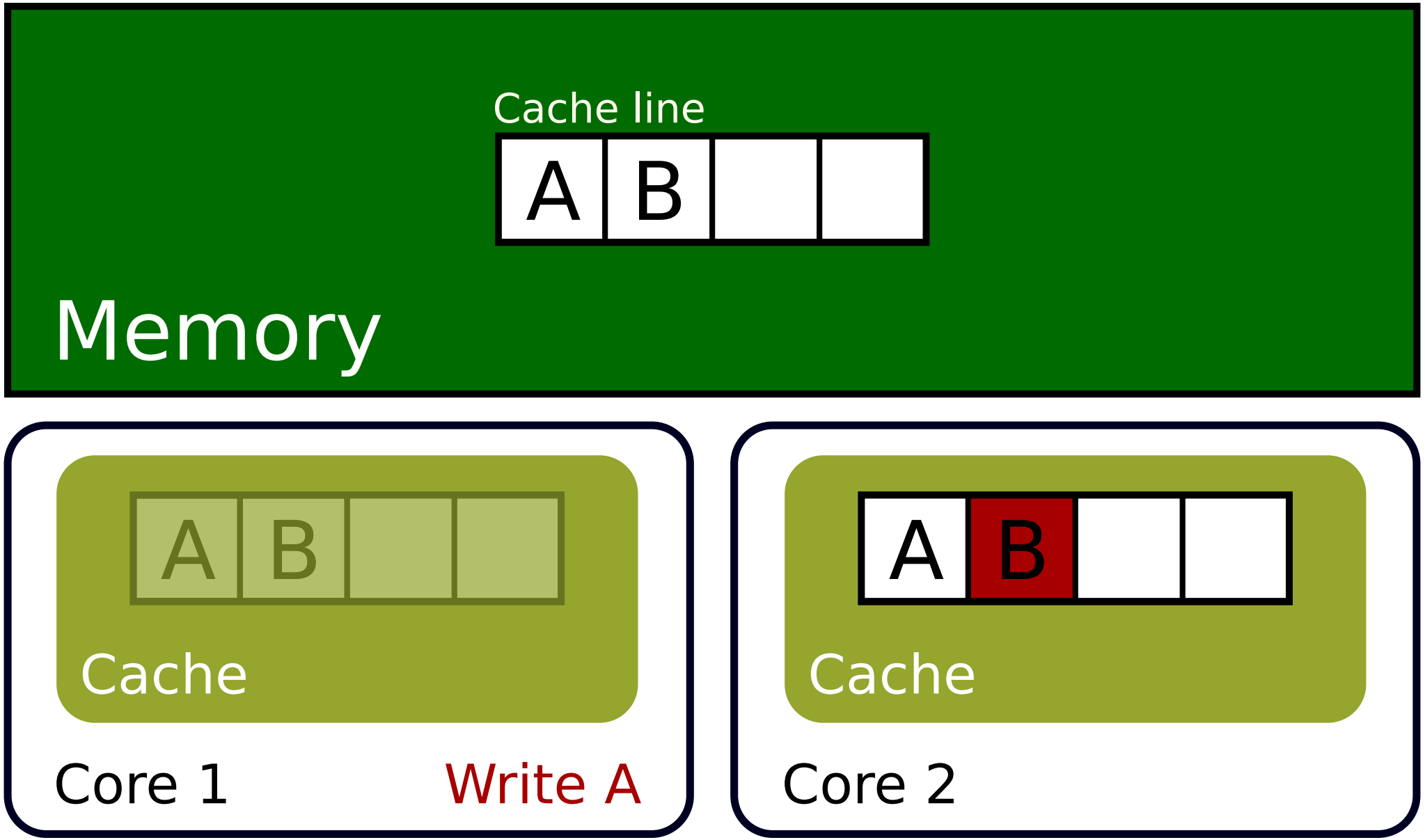
# Cache coherency



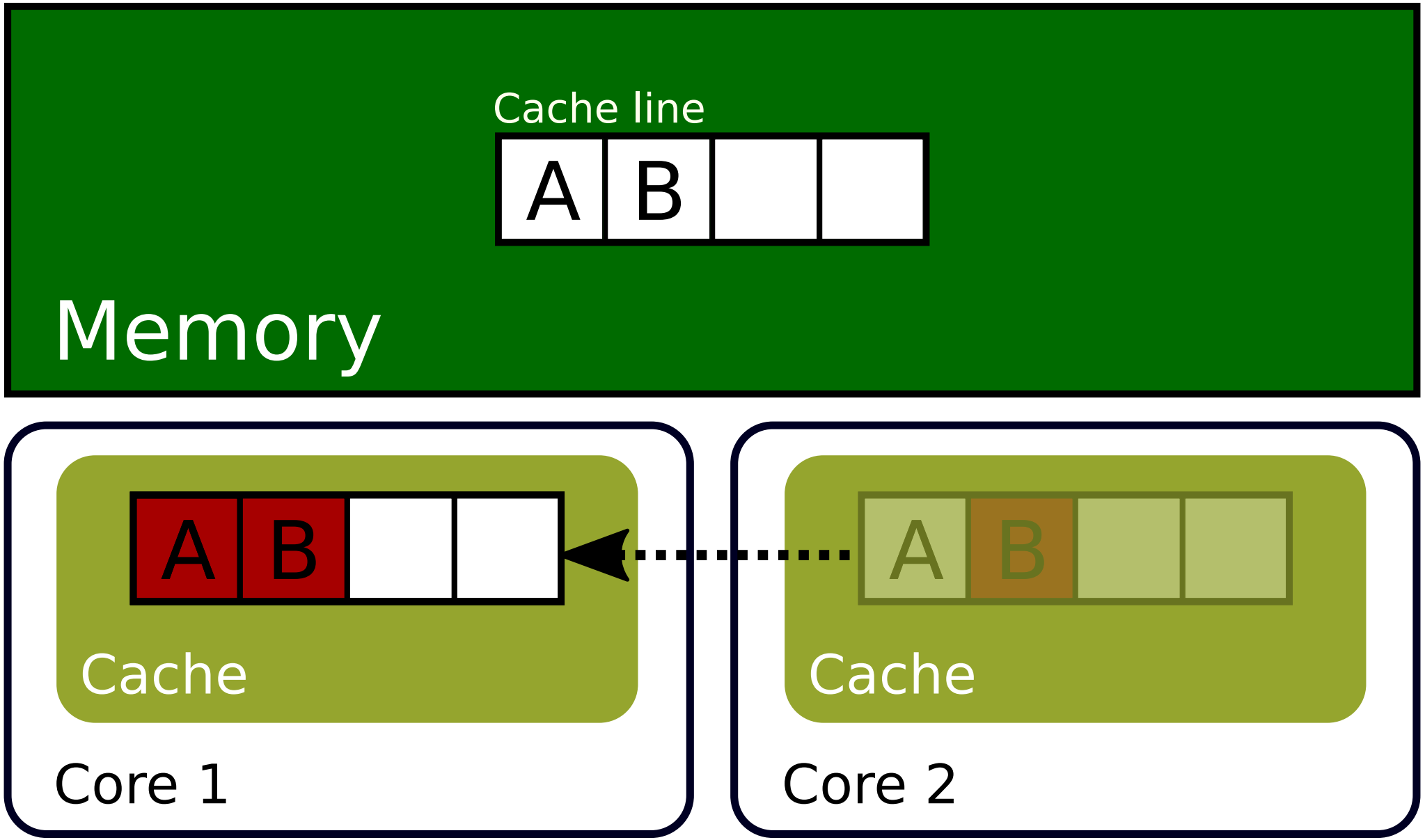
# Cache coherency



# Cache coherency

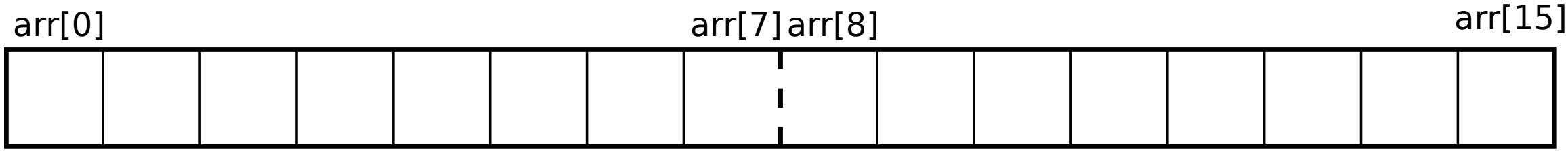


# Cache coherency



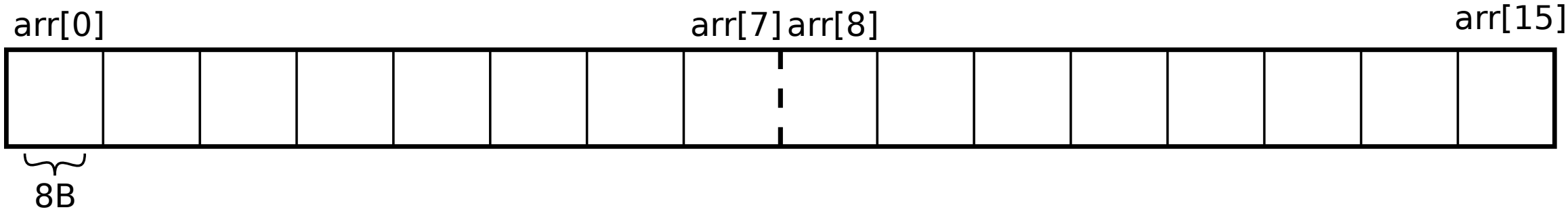
# False sharing

```
double arr[16];
```



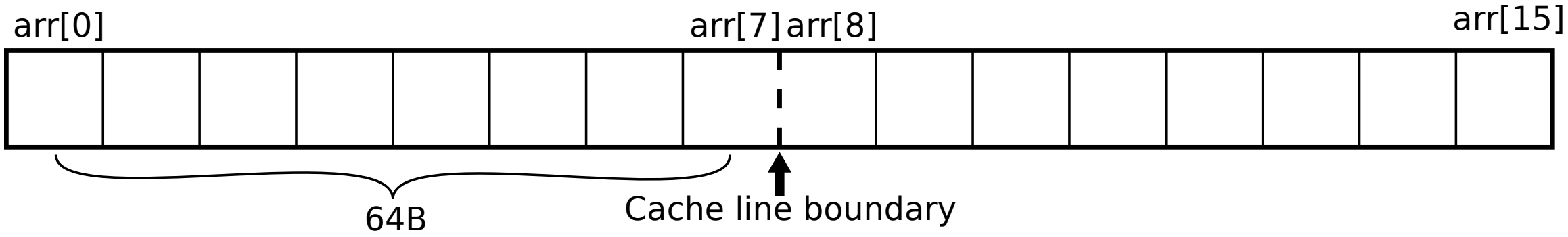
# False sharing

```
double arr[16];
```



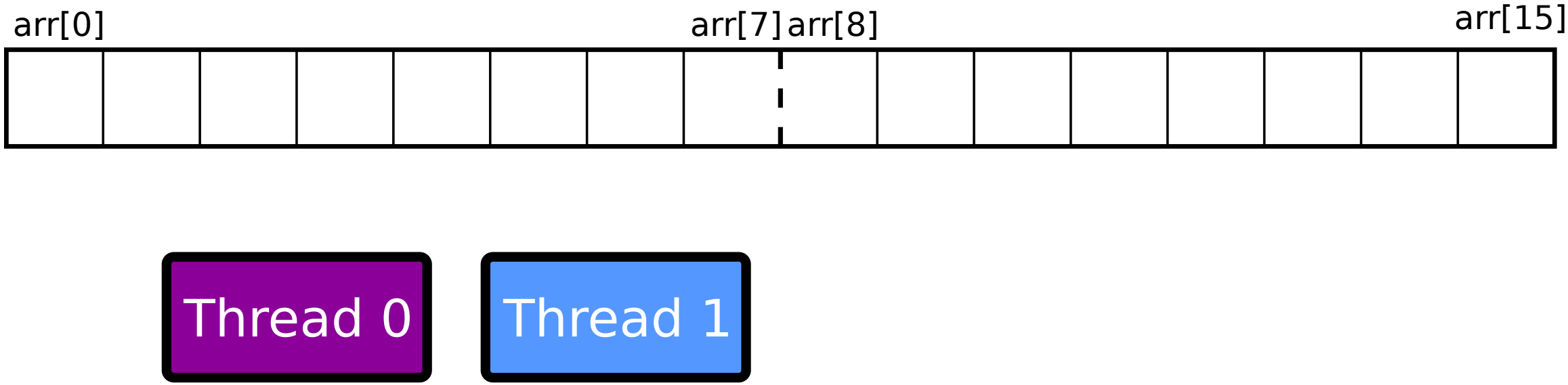
# False sharing

```
double arr[16];
```

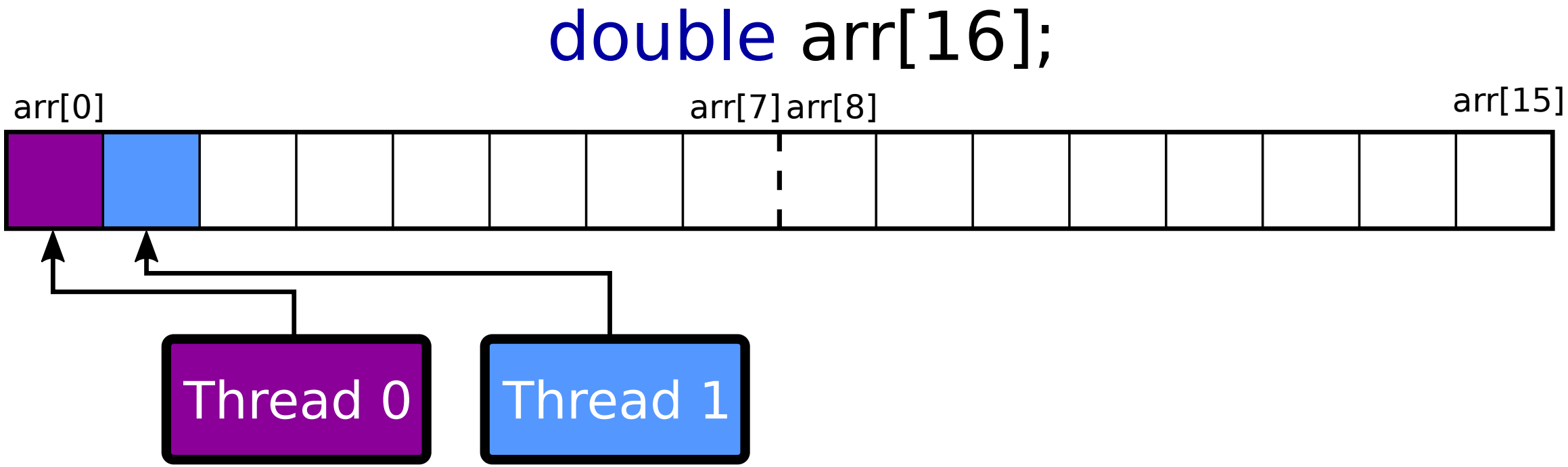


# False sharing

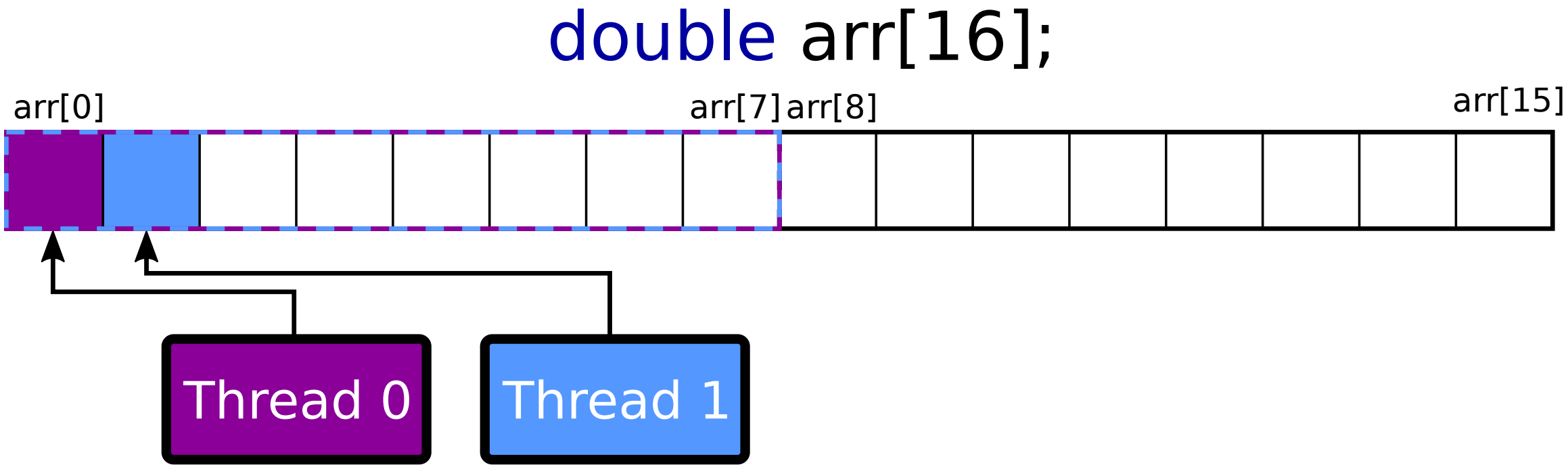
```
double arr[16];
```



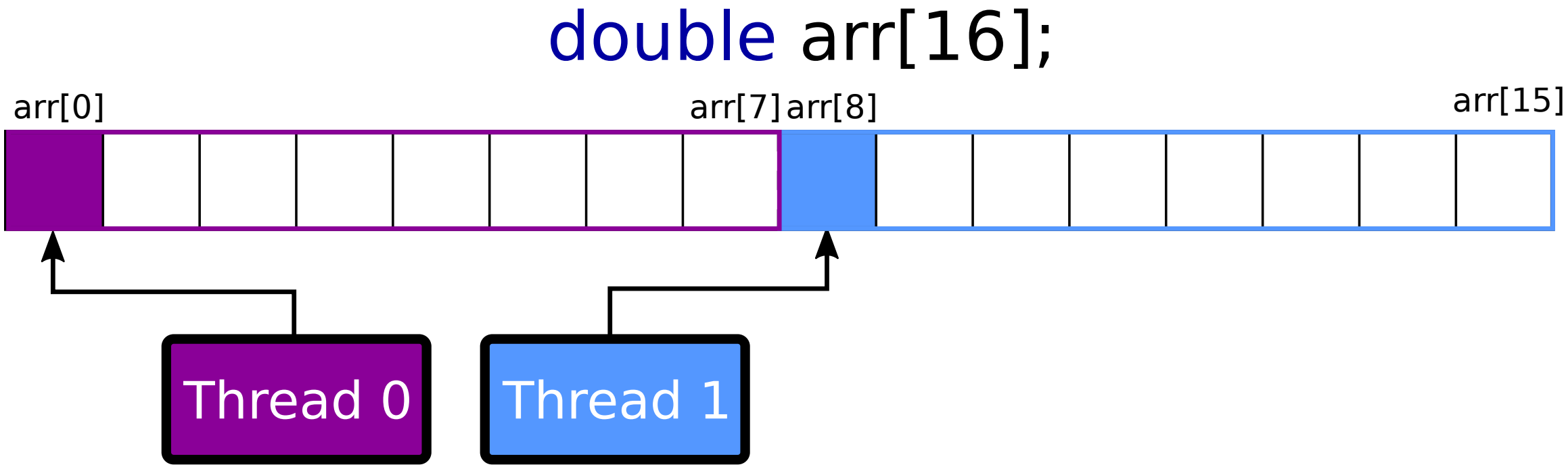
# False sharing



# False sharing



# False sharing



# How to measure?

`l2_rqsts.all_rfo`

How many times some core invalidated data in other cores?

# How to measure?

## `l2_rqsts.all_rfo`

How many times some core invalidated data in other cores?

```
$ perf stat -e l2_rqsts.all_rfo ./example3
1 thread   ->          59 711
2 threads  -> 1 112 258 710
```