Going down the CPU microarchitecture rabbit hole

Ramifications of CPU & memory design

Jakub Beránek

jakub.beranek@vsb.cz

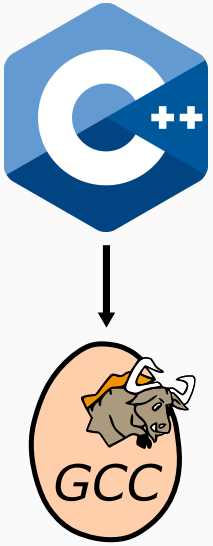
16. 1. 2019

HPC

Foto: Jaroslav Ožana, IT4Innovations

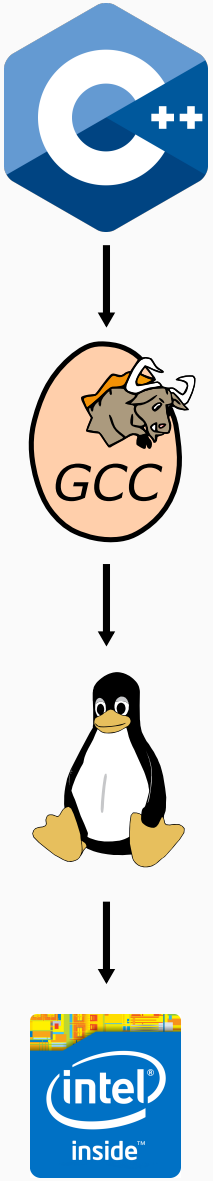
Execution stack

Execution stack



Execution stack

Execution stack



How do you get performance?

How do you get performance?

Language?

```
template <class L, class R>
constexpr auto add_two_numbers(const std::tuple<L, R>& numbers) {
  static_assert(std::is_copy_constructible<decltype(
    std::get<0>(numbers) + std::get<1>(numbers))>::value);

  // use a lambda so it gets inlined
  return [&numbers](const auto& tup) {
    auto [x, y] = tup;
    auto range = {x, y};
    return std::accumulate(std::begin(range), std::end(range), 0);
  }(numbers);
}
```

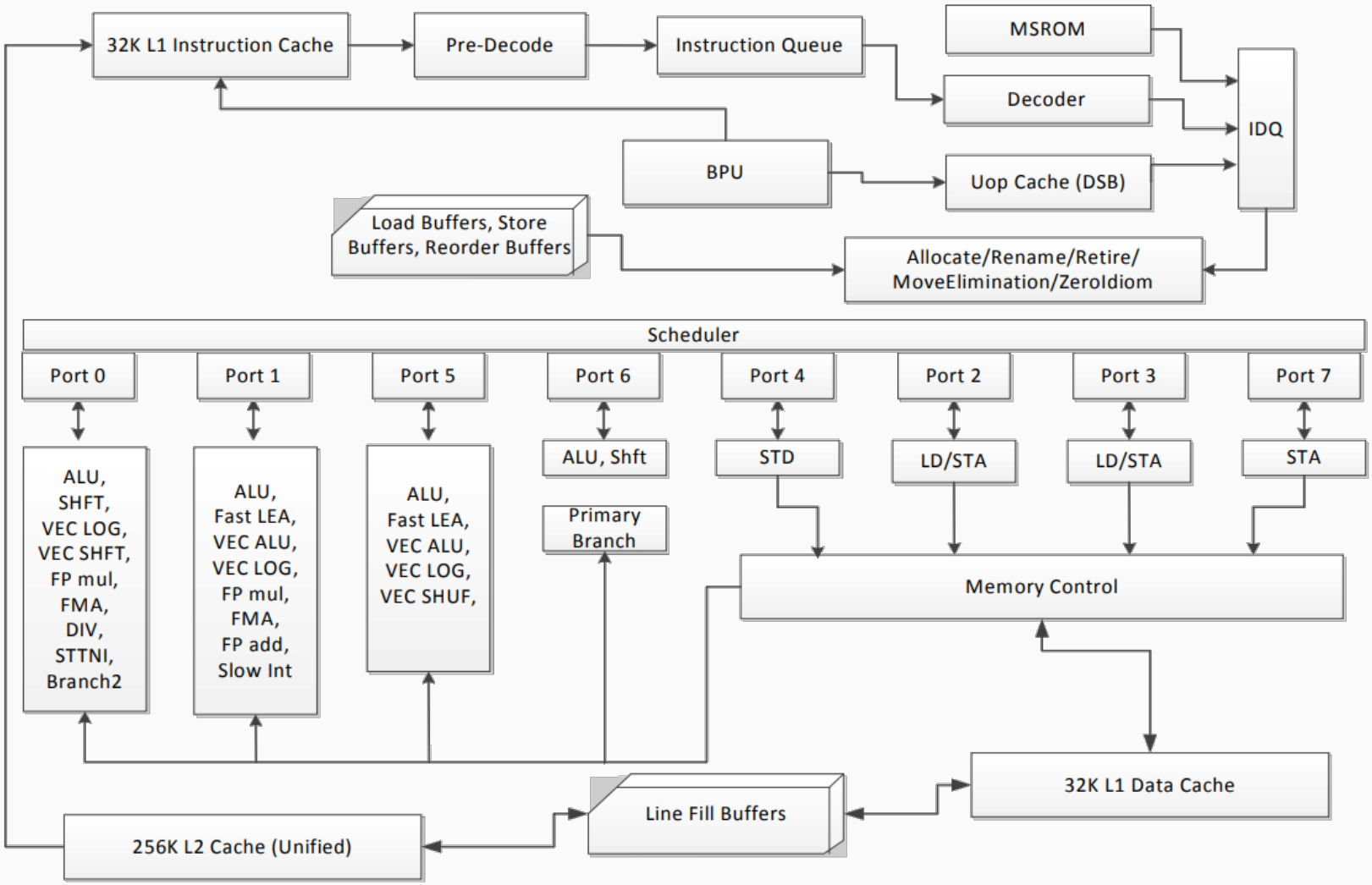
How do you get performance?

Compiler?

```
g++ -O5 -fludicrous-math -pthread-please-scale code.cpp
```

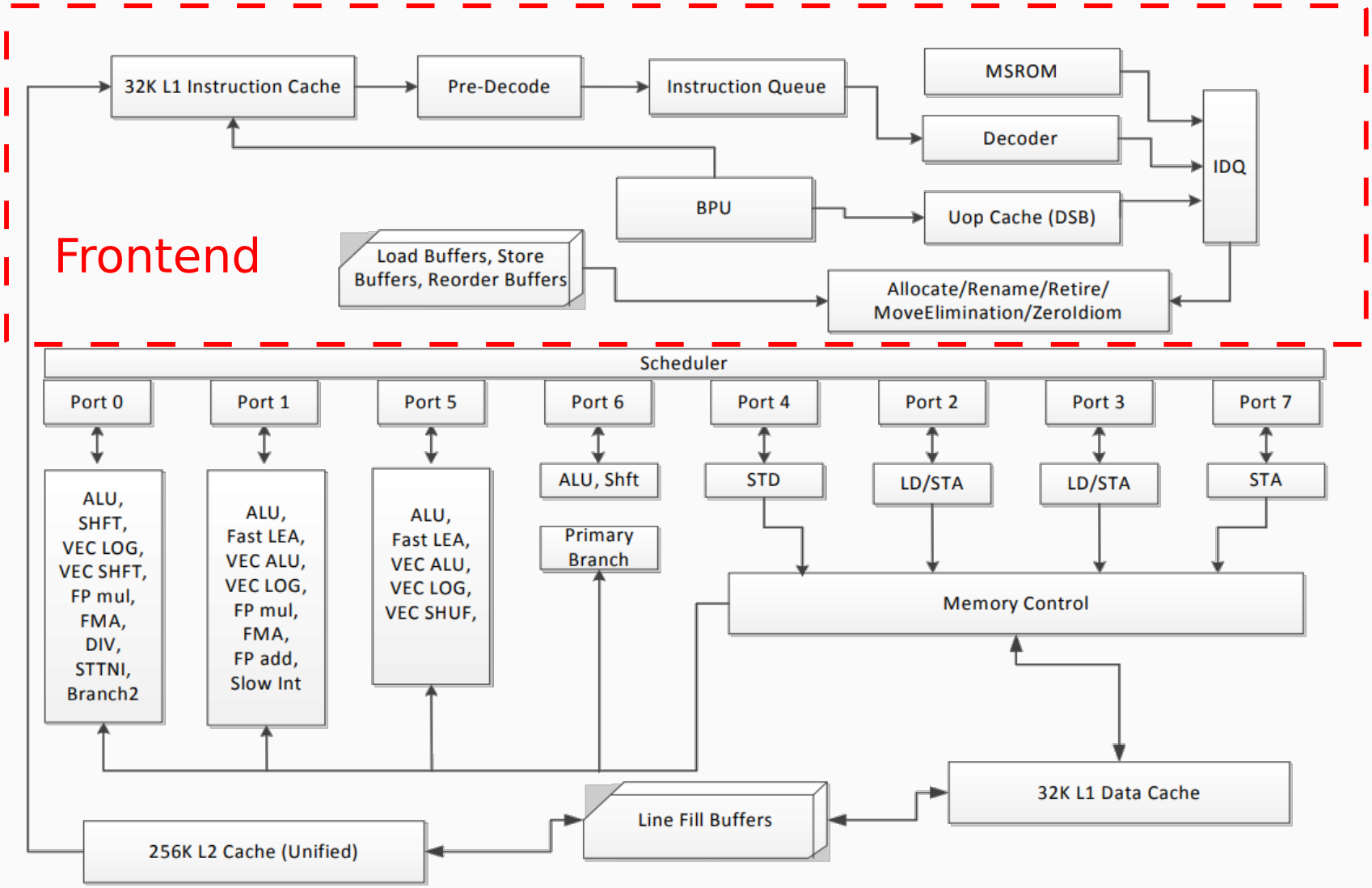
How do you get performance?

Hardware?



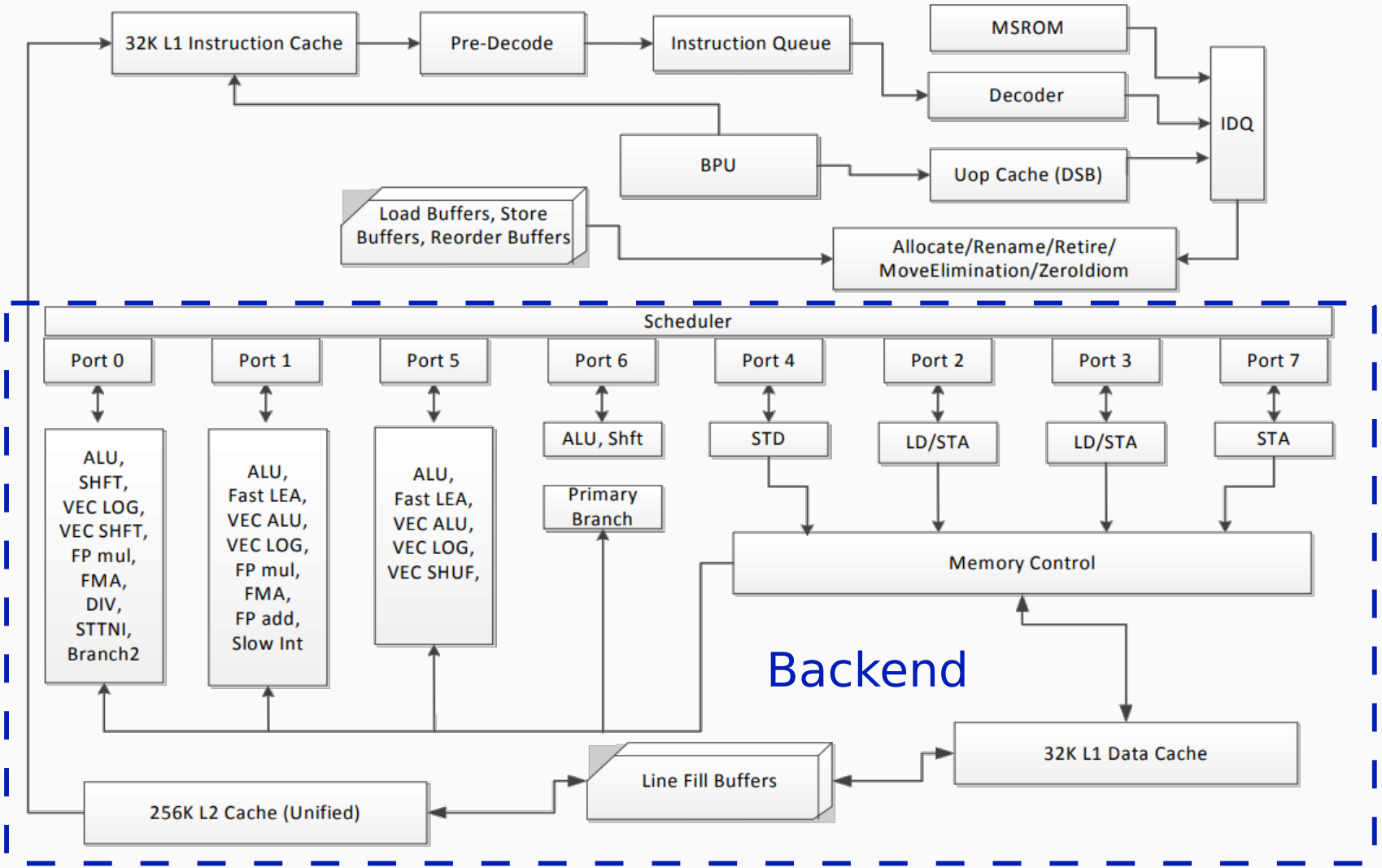
How do you get performance?

Hardware?



How do you get performance?

Hardware?



How do you get performance?

- Use the right algorithm

How do you get performance?

- Use the right algorithm
- Implement and compile it properly

How do you get performance?

- Use the right algorithm
- Implement and compile it properly
- Tune it to the underlying architecture

How do you get performance?

- Use the right algorithm
- Implement and compile it properly
- **Tune it to the underlying architecture**

Software vs hardware complexity

C++ 17 final draft:

Software vs hardware complexity

C++ 17 final draft: 1622 pages

[1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

Software vs hardware complexity

C++ 17 final draft: 1622 pages

Intel x64 manual:

[1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

Software vs hardware complexity

C++ 17 final draft: 1622 pages

Intel x64 manual: **4846** pages!

[1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

[2] https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

CPU demonstration snippet found online

```
#define BEST_TIME(test, expected, pre, repeat, size, verbose)
        do {                                                                \
            if (global_rdtsc_overhead == UINT64_MAX) {                      \
               RDTSC_SET_OVERHEAD(rdtsc_overhead_func(1), repeat);          \
            }                                                               \
            if(verbose) printf("%-60s\t: ", #test);
            fflush(NULL);                                                   \
            uint64_t cycles_start, cycles_final, cycles_diff;               \
            uint64_t min_diff = (uint64_t)-1;                               \
            uint64_t sum_diff = 0;                                          \
            for (int i = 0; i < repeat; i++) {                              \
                pre;                                                        \
                __asm volatile("" ::: /* pretend to clobber */ "memory"); \
                RDTSC_START(cycles_start);                                  \
                if(test != expected) {printf("not expected (%d , %d )",(int)test
                RDTSC_STOP(cycles_final);                                     \
```

Hardware effects

`https://github.com/kobzol/hardware-effects`

Demonstrations of CPU design ramifications

Hardware effects

`https://github.com/kobzol/hardware-effects`

Demonstrations of CPU design ramifications

- Short, comprehensible C++ programs

Hardware effects

`https://github.com/kobzol/hardware-effects`

Demonstrations of CPU design ramifications

- Short, comprehensible `C++` programs
- `-O3`

Let's see some examples

Most upvoted Stack Overflow question

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

22447

10294

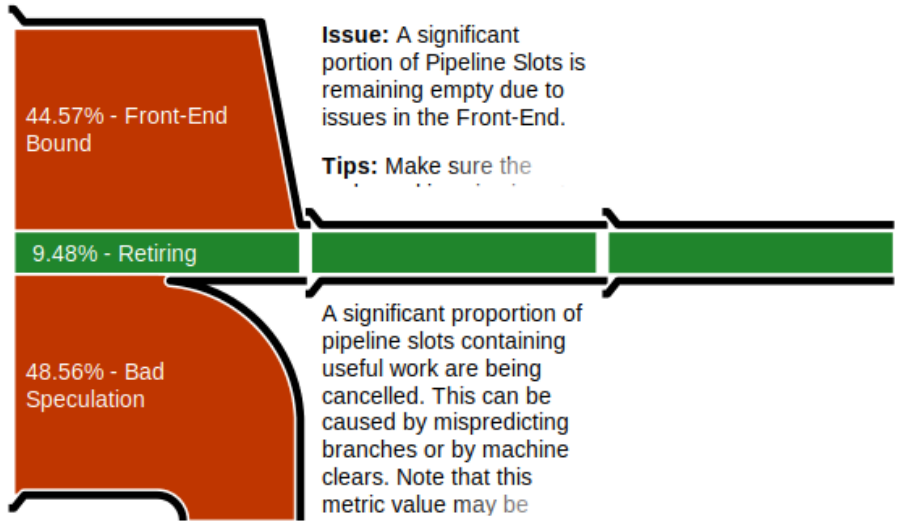
Here is a piece of C++ code that seems very peculiar. For some strange reason, sorting the data miraculously makes the code almost six times faster.

```
#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];
        }
```

Profiling tools - 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	



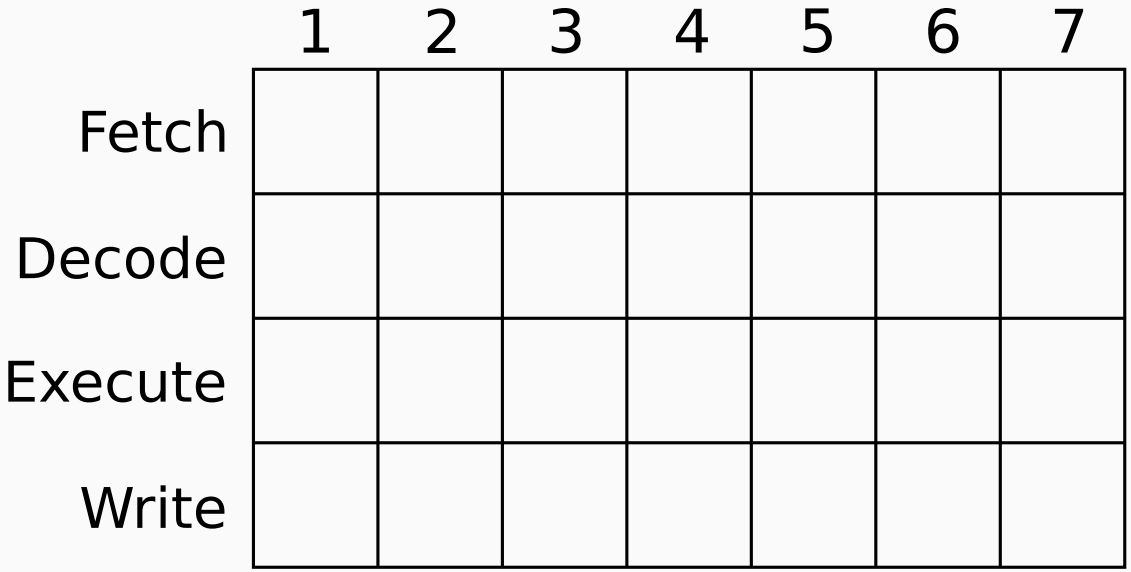
µPipe

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.

Profiling tools - perf

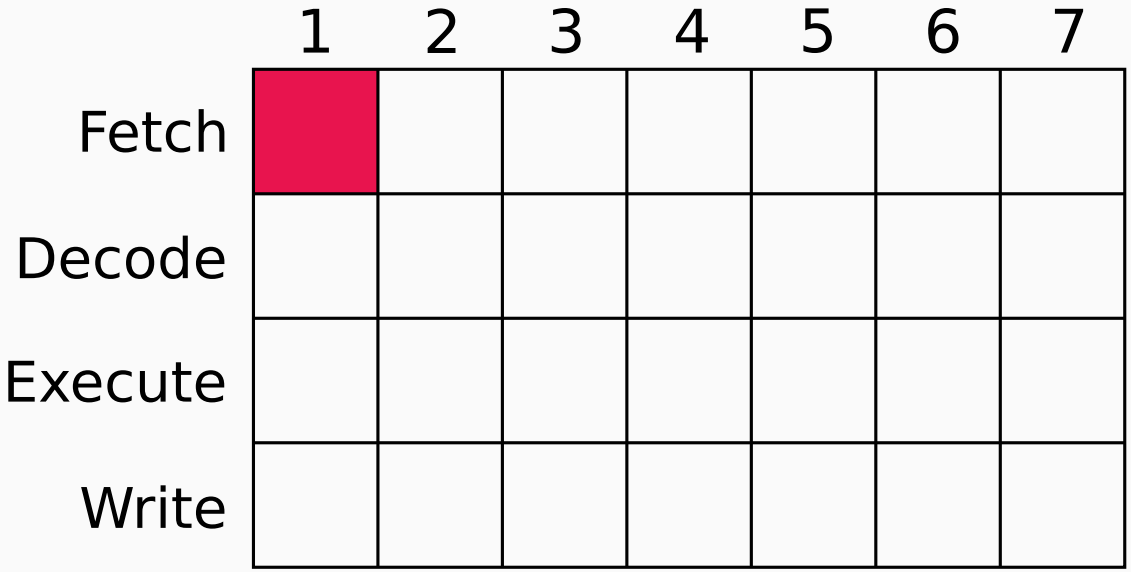
```
$ perf stat ./my-program

                948  task-clock (msec)
                197  page-faults
      3 254 047 397  cycles
      1 483 156 512  instructions
        421 718 831  branches
        102 931 445  branch-misses

$ perf list # lists all available counters
```

CPU pipeline 101



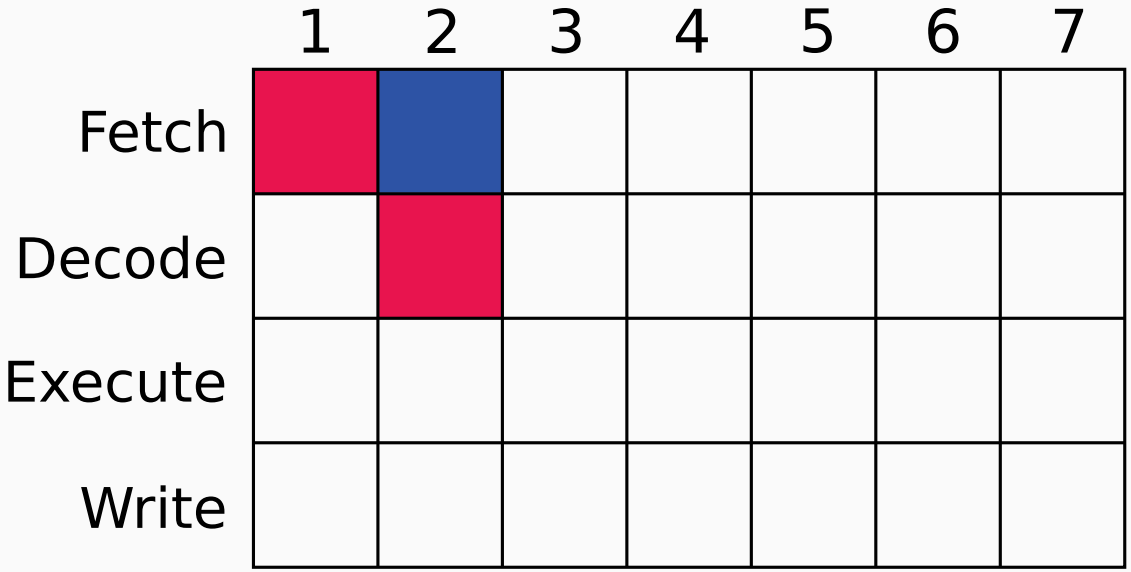

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

CPU pipeline 101



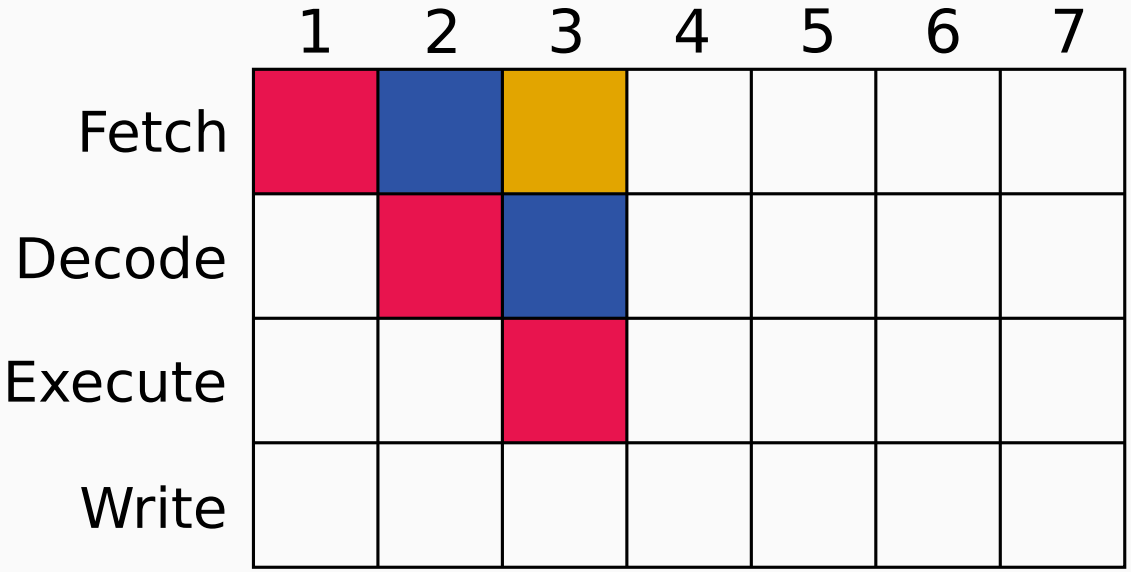

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

CPU pipeline 101



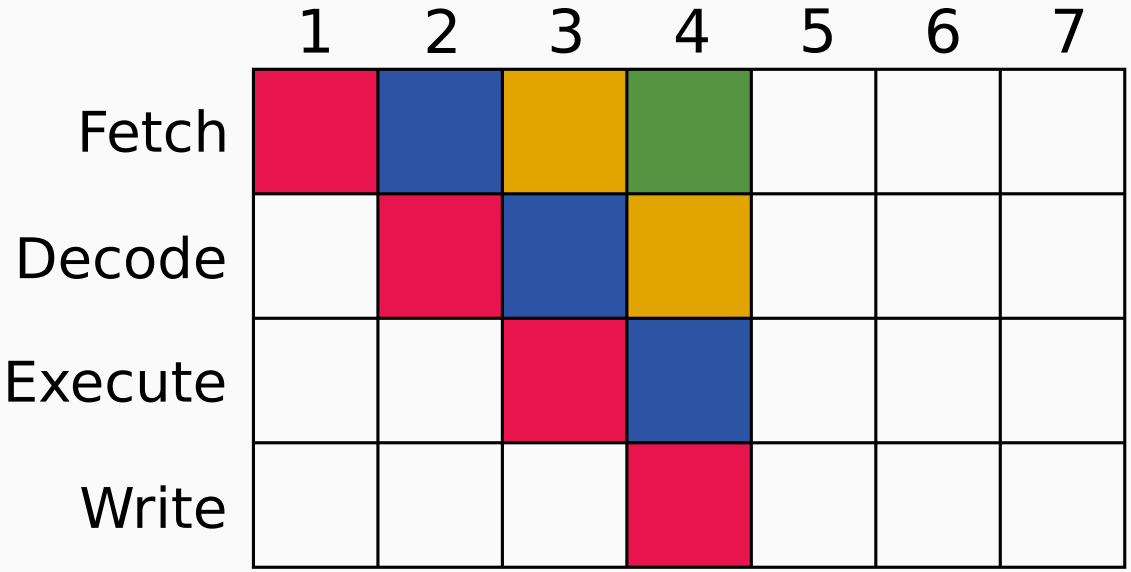

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

CPU pipeline 101



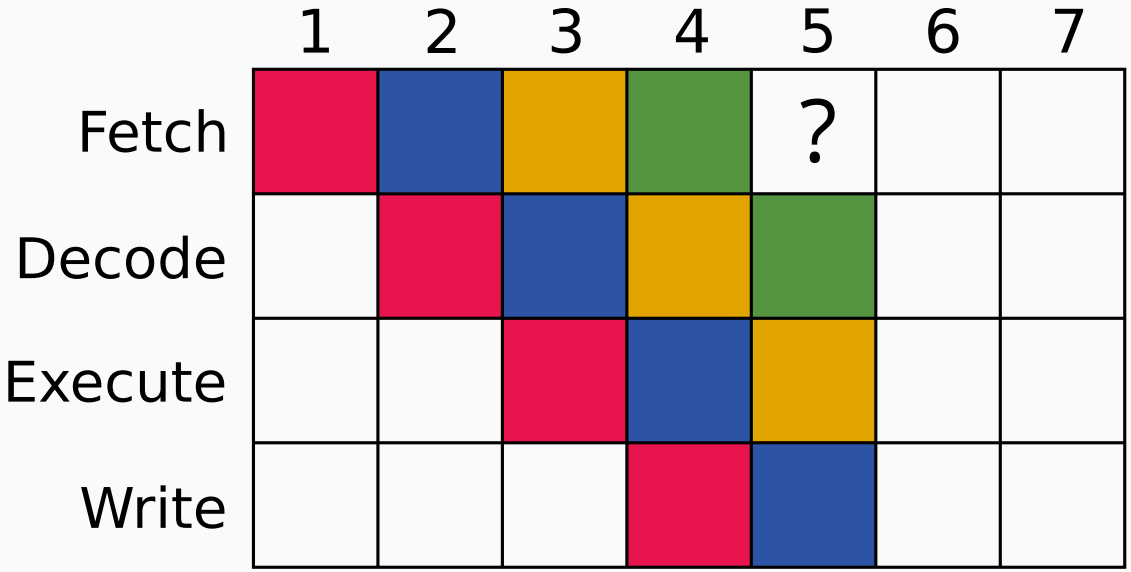

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

CPU pipeline 101



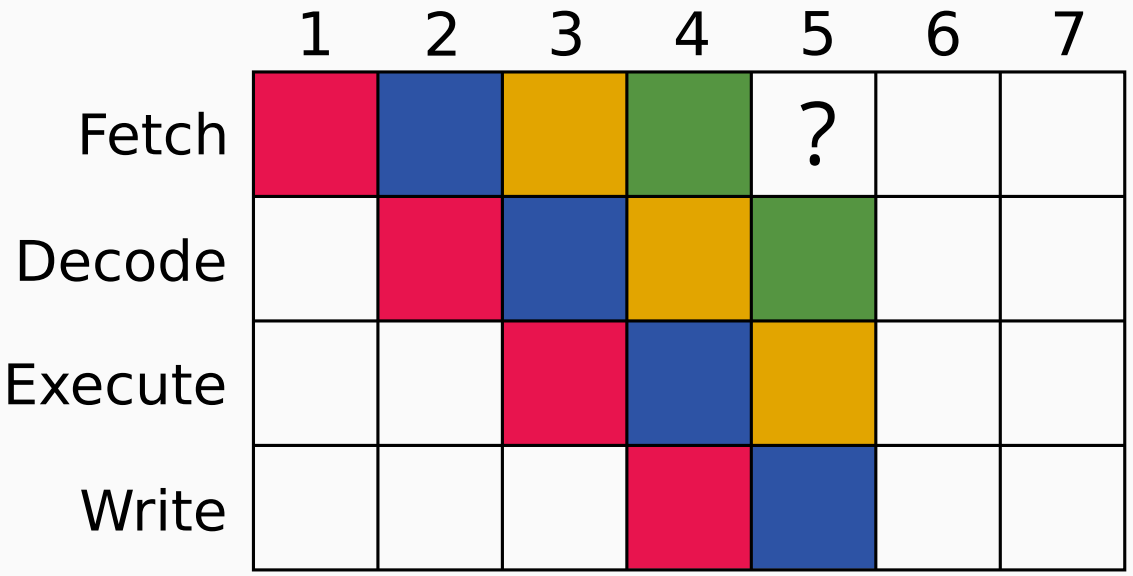

```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

CPU pipeline 101




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

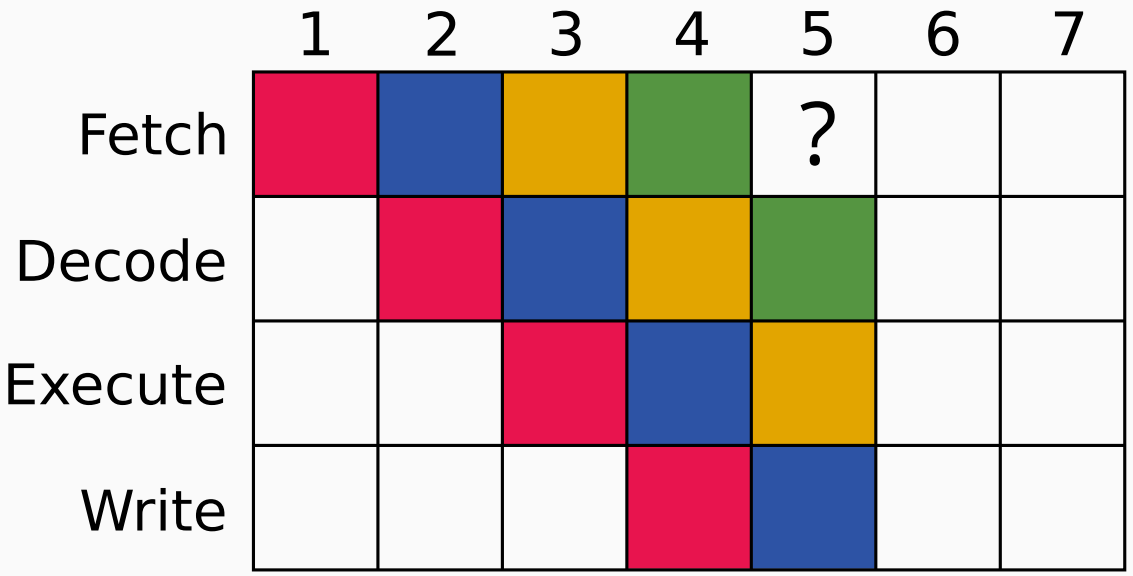
CPU pipeline 101




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

The CPU predicts the result of conditional branches

CPU pipeline 101

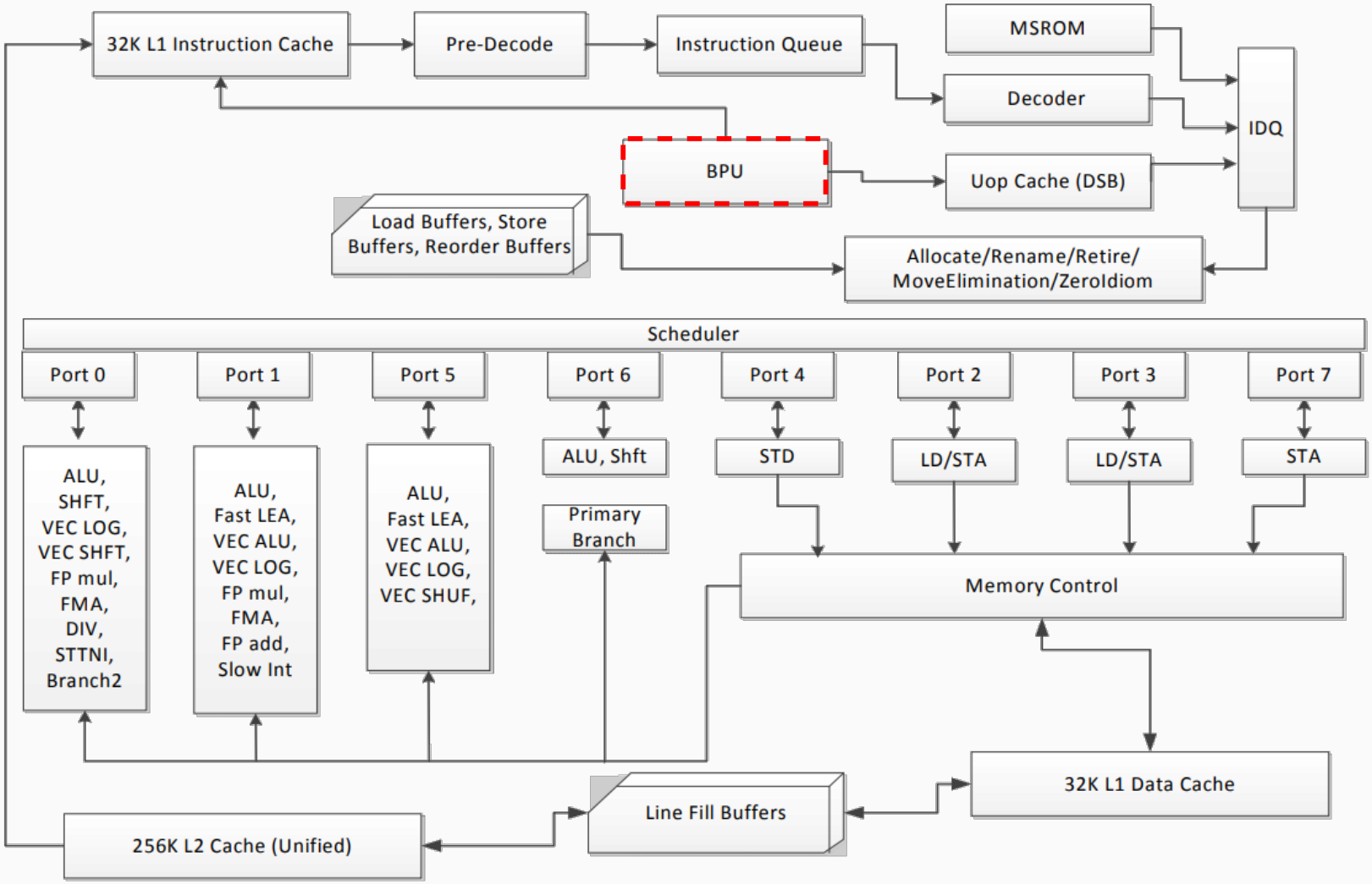



```
7   xor rax,rdx
8   add rax,rcx
9   cmp rax,rbx
10  je 15
11  inc rcx
⋮
15  ret
```

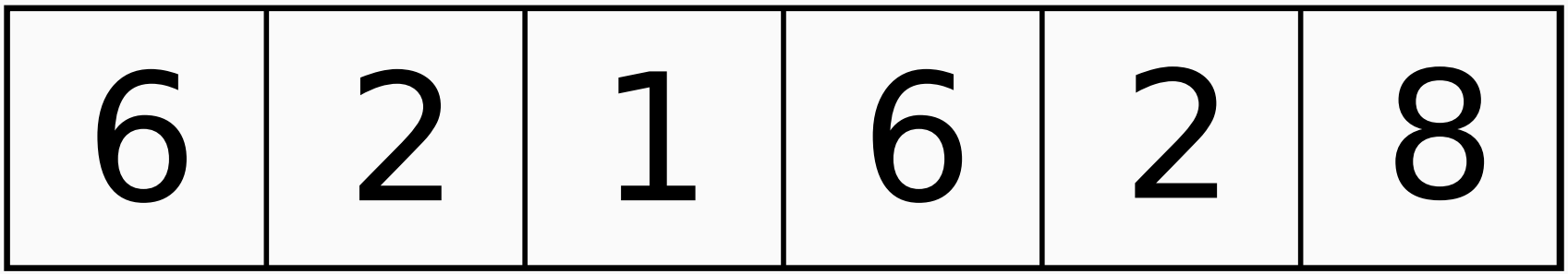
The CPU predicts the result of conditional branches

Misprediction costs ~15-20 cycles!

Branch predictor



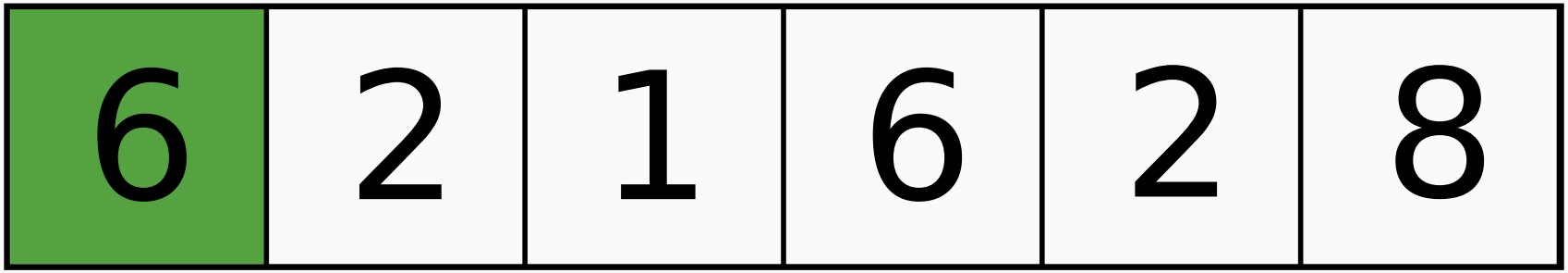
Branch prediction - random array



Prediction:

Not taken

Branch prediction - random array

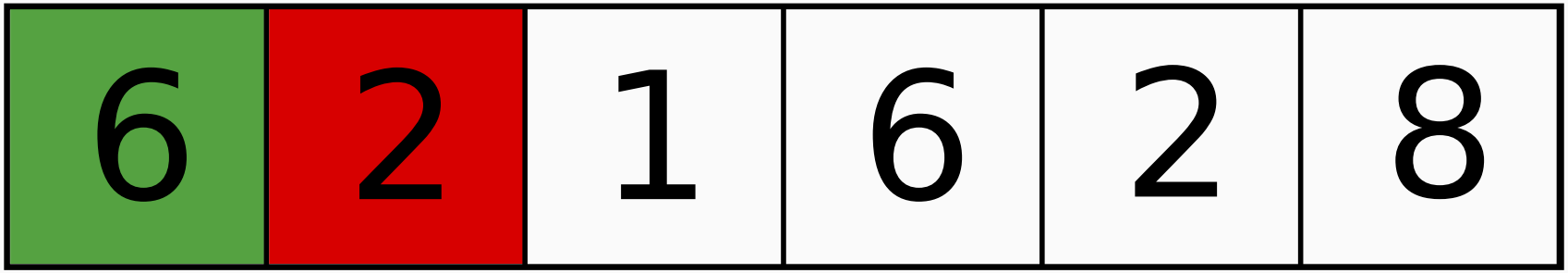


6<6?

Prediction:

Not taken

Branch prediction - random array

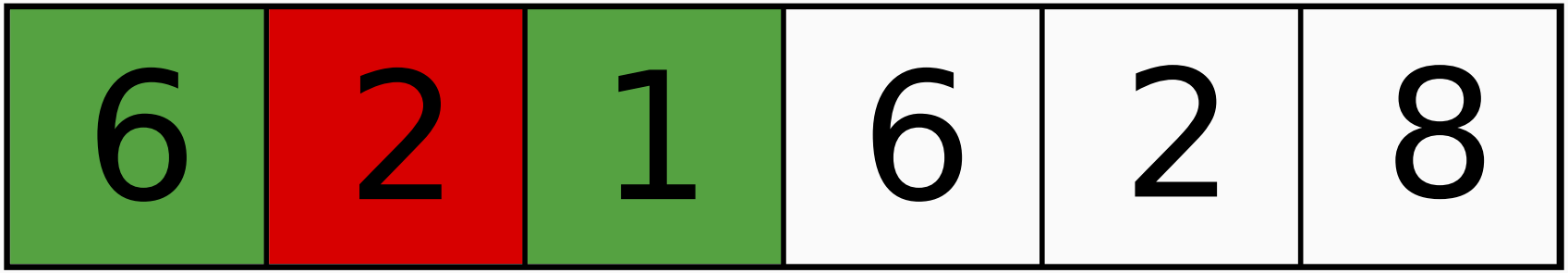


2<6?

Prediction:

Taken

Branch prediction - random array

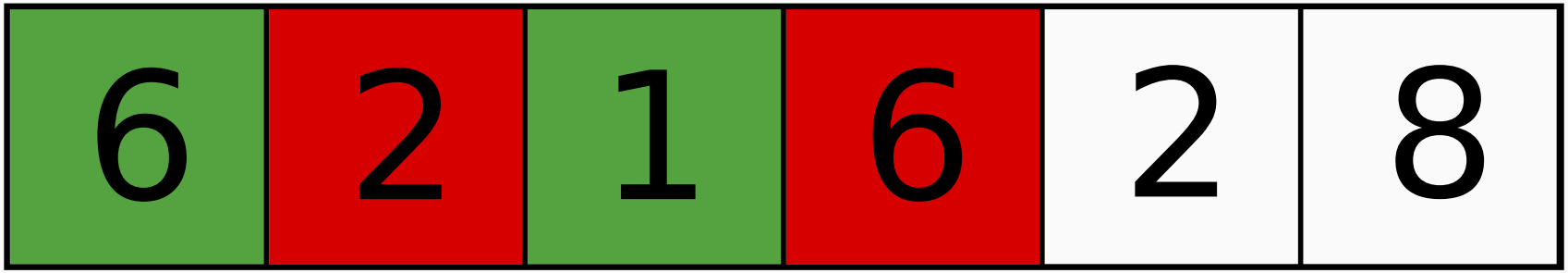


1<6?

Prediction:

Taken

Branch prediction - random array

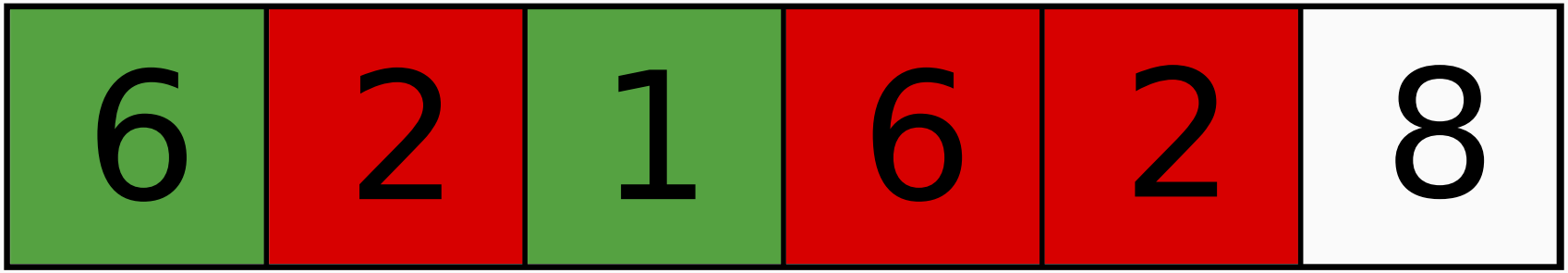


6<6?

Prediction:

Not taken

Branch prediction - random array

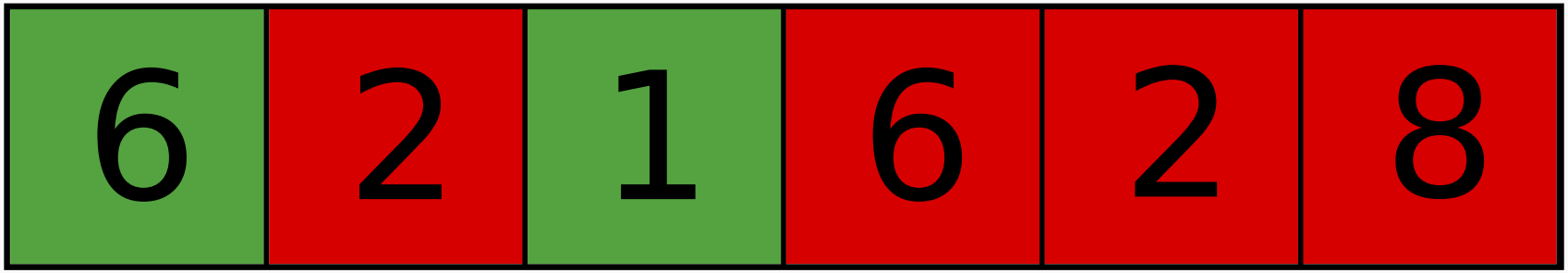


2<6?

Prediction:

Taken

Branch prediction - random array

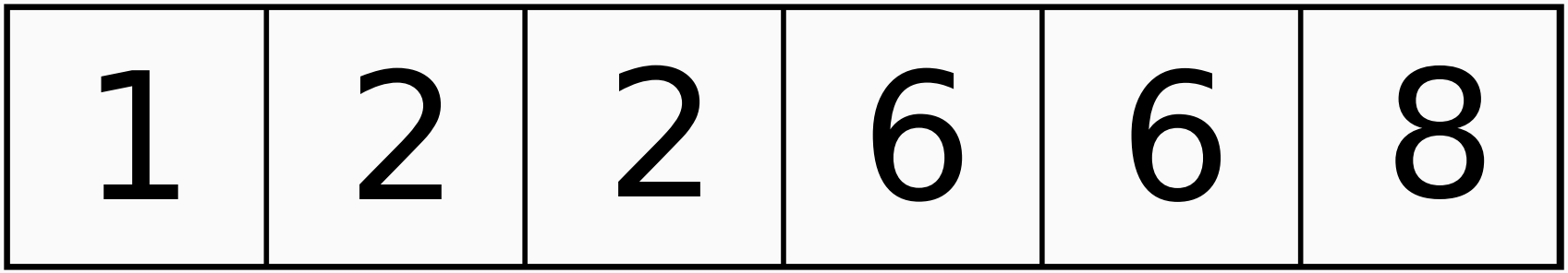


8<6?

Prediction:

Not taken

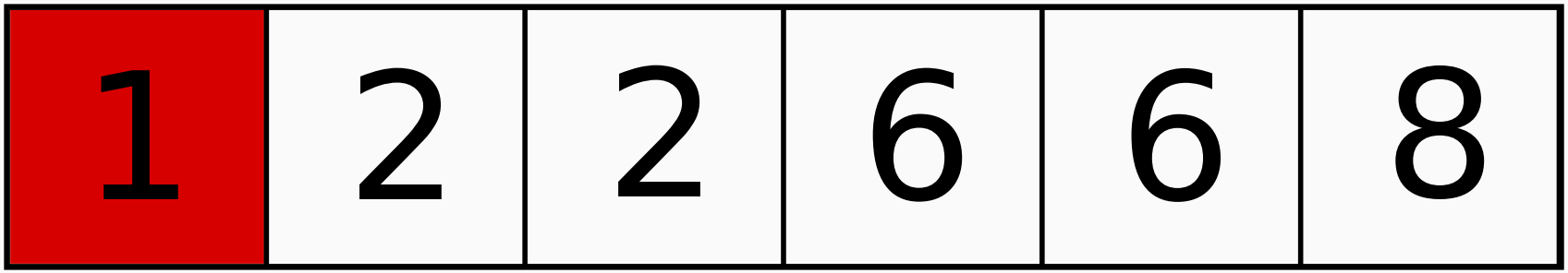
Branch prediction - sorted array



Prediction:

Not taken

Branch prediction - sorted array

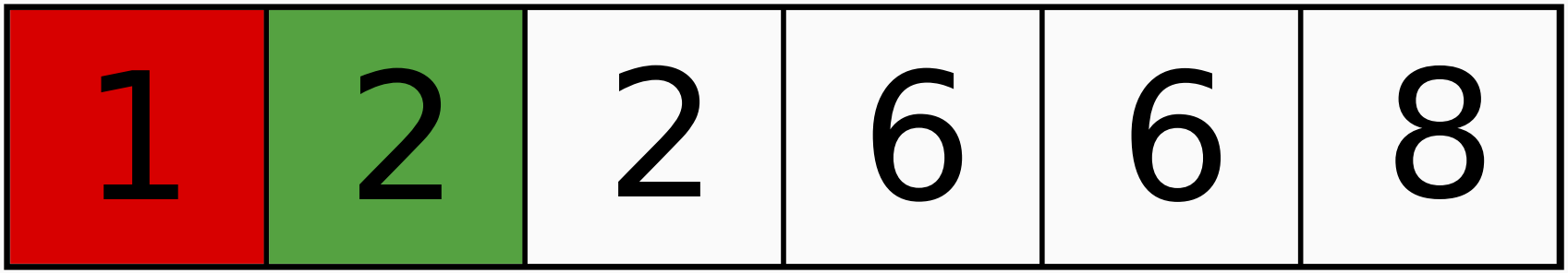


1<6?

Prediction:

Taken

Branch prediction - sorted array

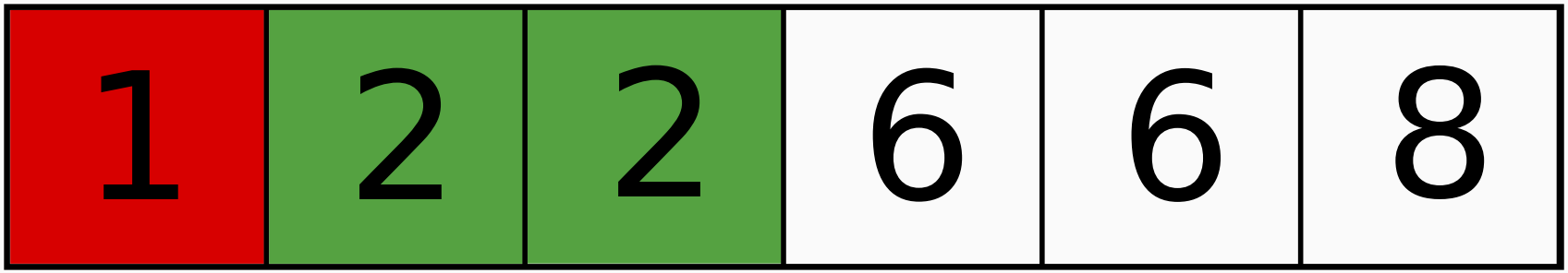


2<6?

Prediction:

Taken

Branch prediction - sorted array

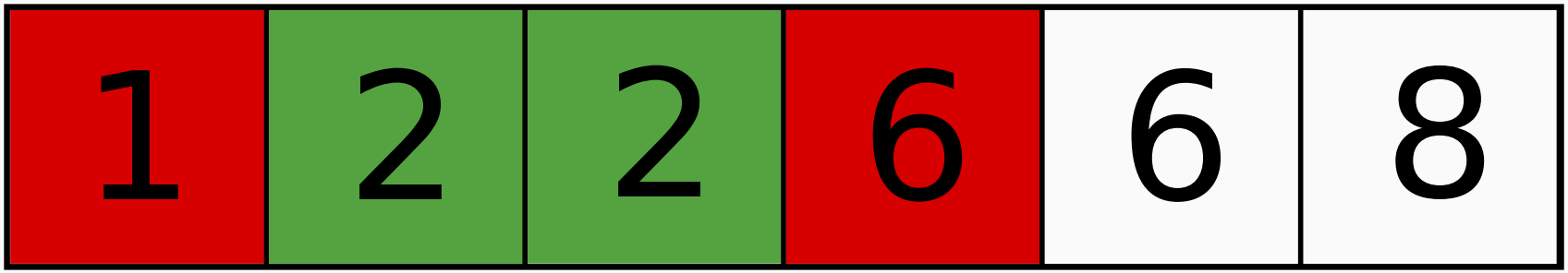


2<6?

Prediction:

Taken

Branch prediction - sorted array

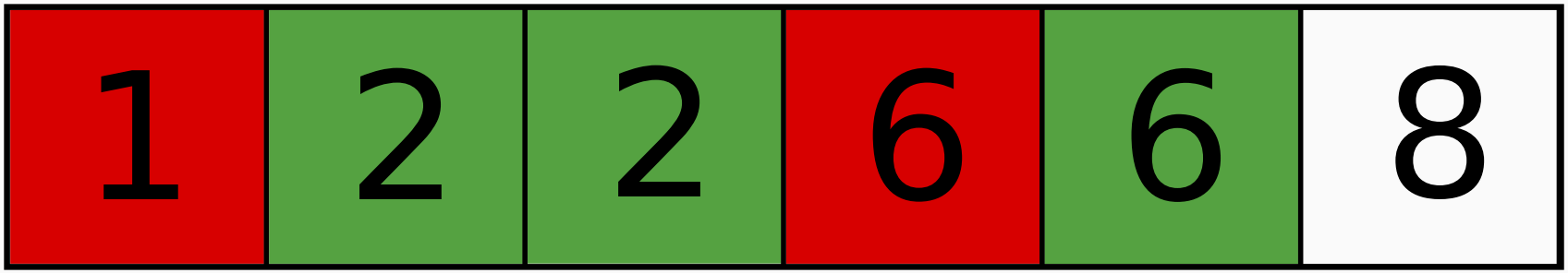


6<6?

Prediction:

Not taken

Branch prediction - sorted array

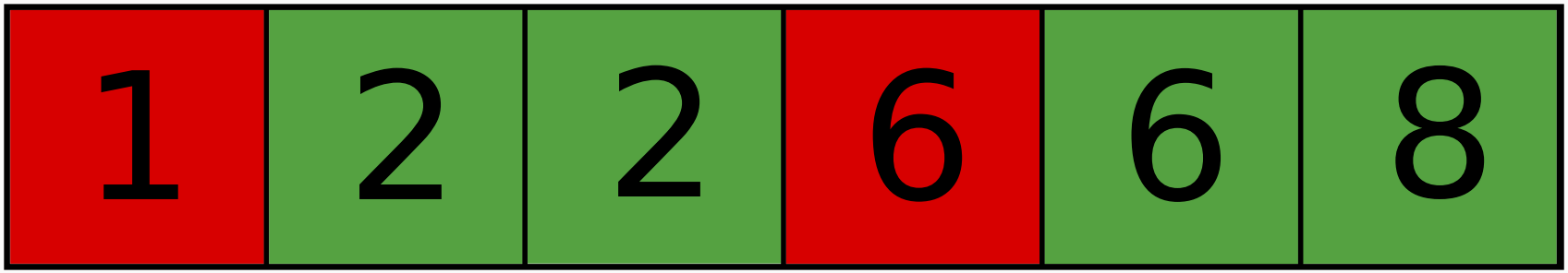


6<6?

Prediction:

Not taken

Branch prediction - sorted array



8<6?

Prediction:

Not taken

How to help the branch predictor?

- More predictable data

How to help the branch predictor?

- More predictable data
- Compiler hints

```
if (__builtin_expect(will_it_blend(), 1)) {
    // this branch is likely to be taken
}
```

How to help the branch predictor?

- More predictable data
- Compiler hints

```
if (__builtin_expect(will_it_blend(), 1)) {
    // this branch is likely to be taken
}
```

  - Automated with Profile-guided optimization (PGO)

How to help the branch predictor?

- More predictable data
- Compiler hints

```
if (__builtin_expect(will_it_blend(), 1)) {
    // this branch is likely to be taken
}
```

  - Automated with Profile-guided optimization (PGO)

- Remove branches

Indirect jumps

Target of the jump is unknown at compile/link time

Indirect jumps

Target of the jump is unknown at compile/link time

- Function pointers

Indirect jumps

Target of the jump is unknown at compile/link time

- Function pointers
- Function return addresses

Indirect jumps

Target of the jump is unknown at compile/link time

- Function pointers
- Function return addresses
- Virtual methods

Virtual method overhead

- Increased object size
  - More data cache misses

Virtual method overhead

- Increased object size
  - More data cache misses
- Extra indirection

Virtual method overhead

- Increased object size
  - More data cache misses
- Extra indirection
- Prevents inlining

Virtual method overhead

- Increased object size
  - More data cache misses
- Extra indirection
- Prevents inlining
- Branch target mispredictions

Virtual method overhead

- Increased object size
  - More data cache misses
- Extra indirection
- Prevents inlining
- Branch target mispredictions
  - Branch target predictor

How to measure it?

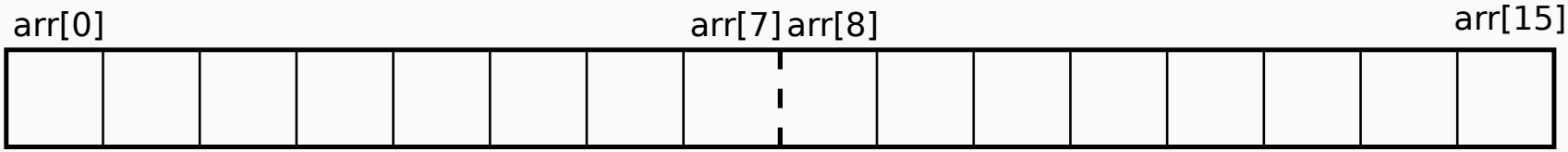
`branch-misses`

How many times CPU mispredicted a branch or an indirect jump?

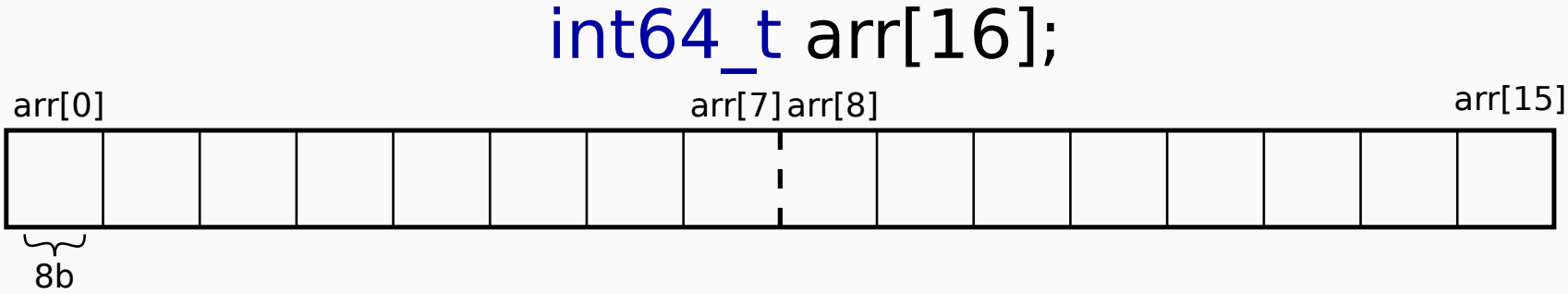
```
$ perf stat -e branch-misses ./my-program

         72 480 534 branch-misses
```

False sharing

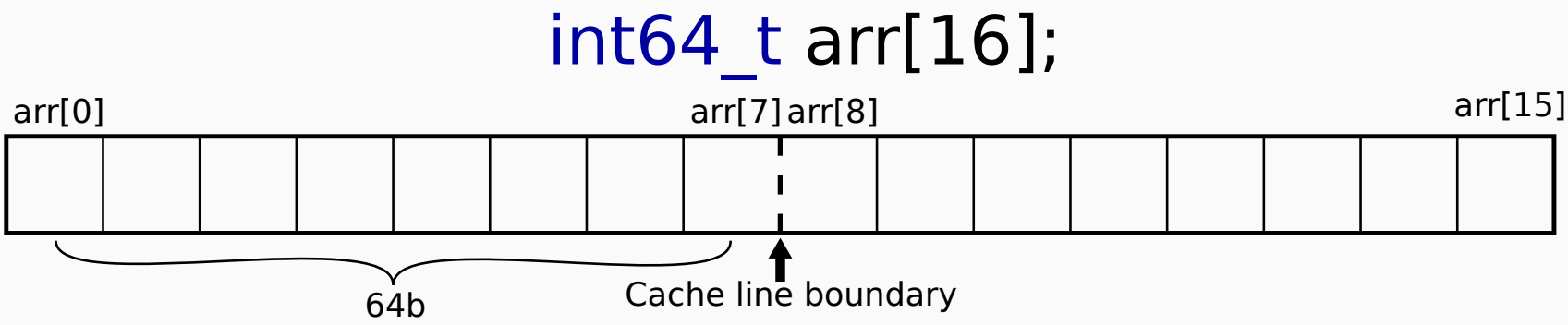
```
int64_t arr[16];
```



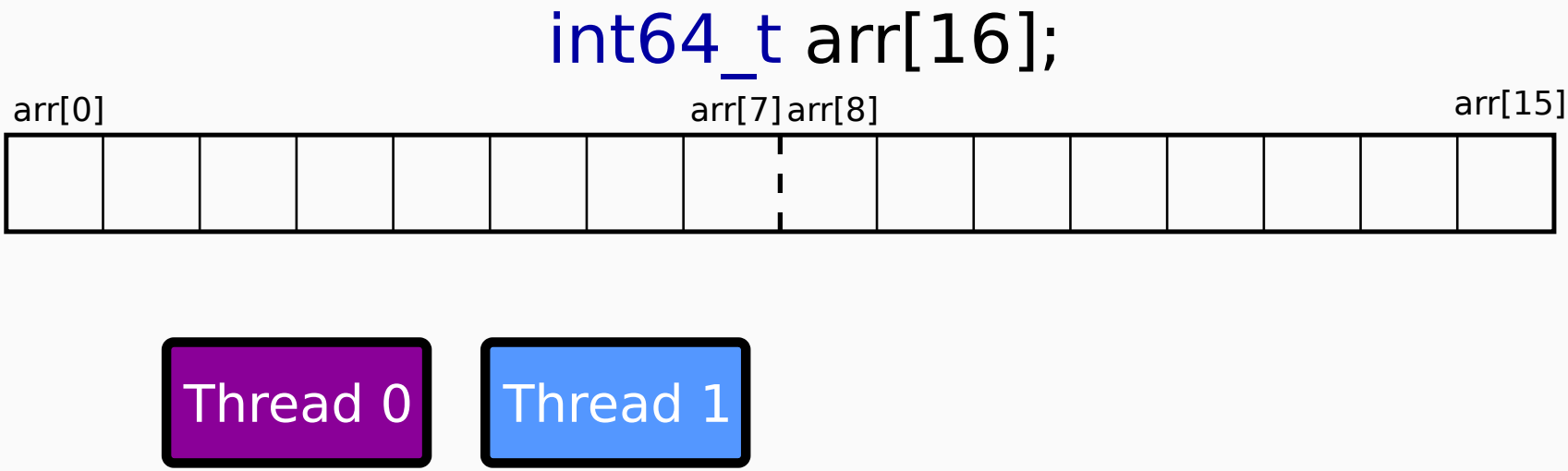
False sharing



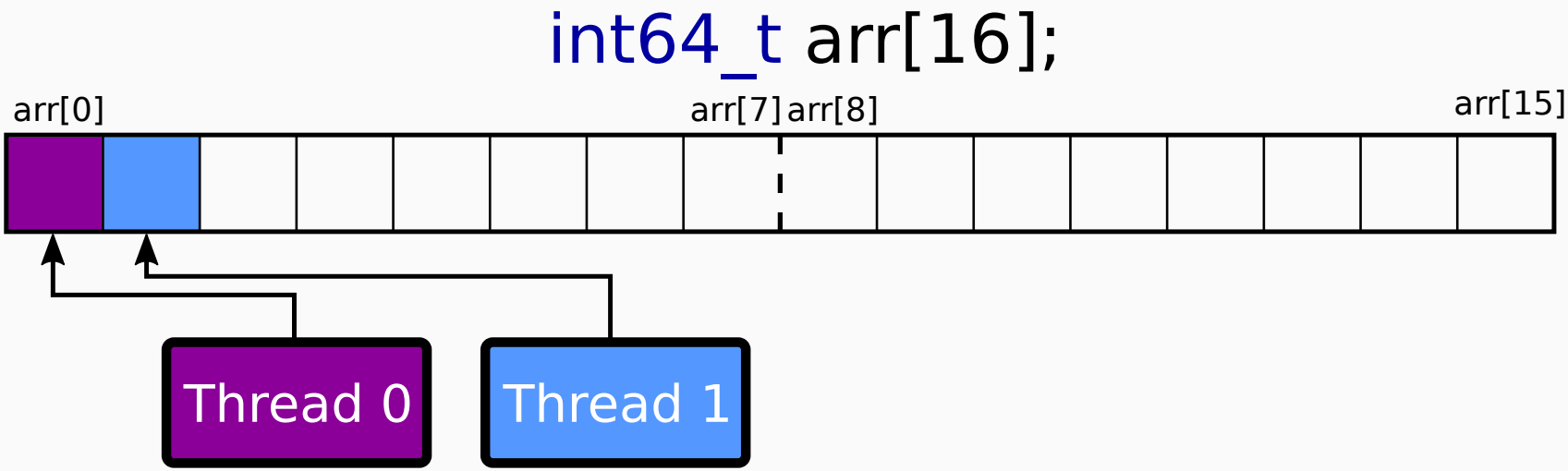
False sharing



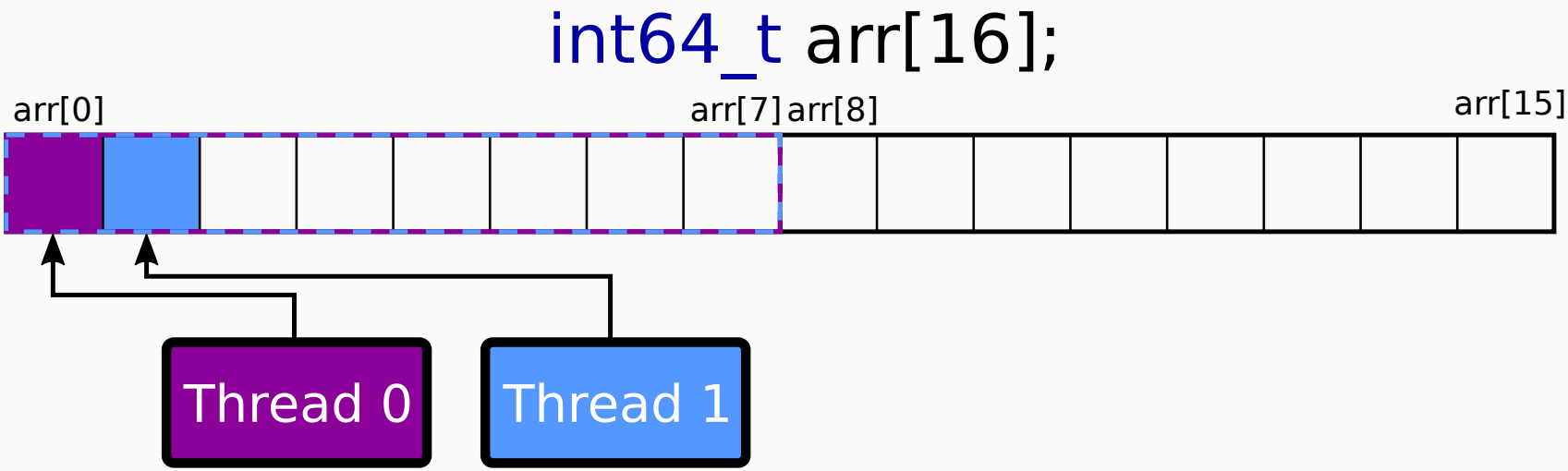
False sharing



False sharing



False sharing



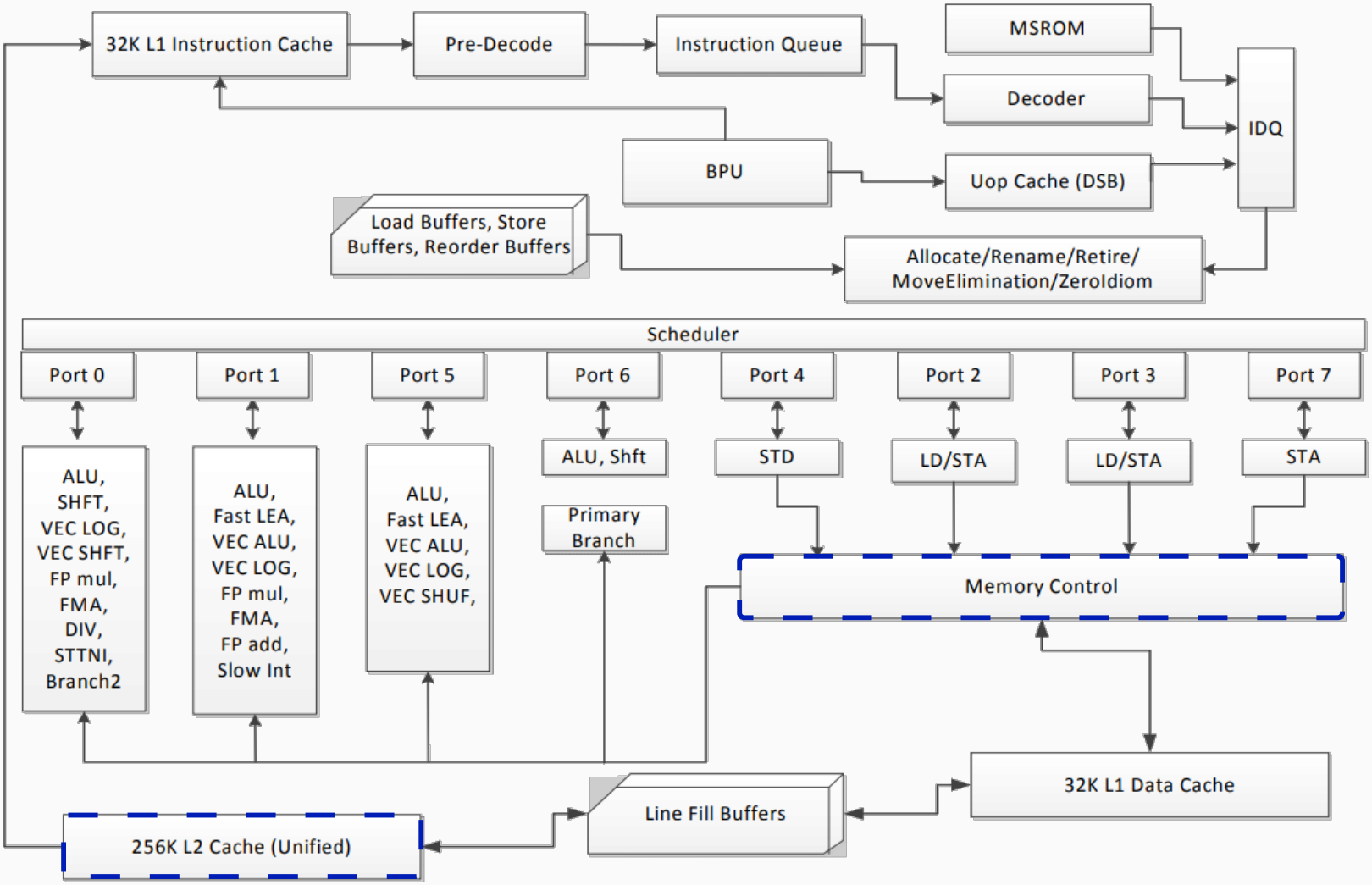
False sharing

- Multiple threads access the same cache line at the same time

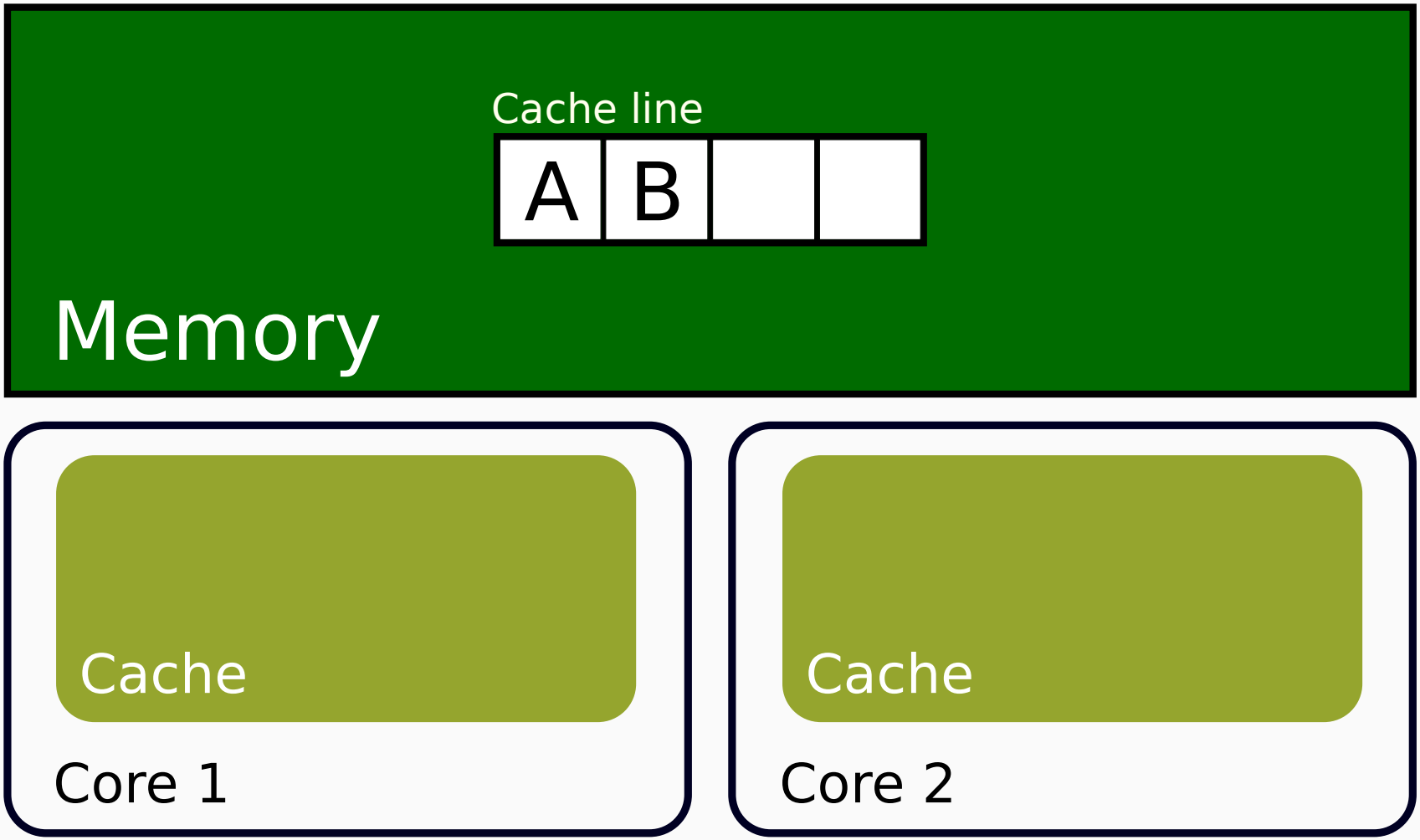
False sharing

- Multiple threads access the same cache line at the same time
- At least one of them writes to it

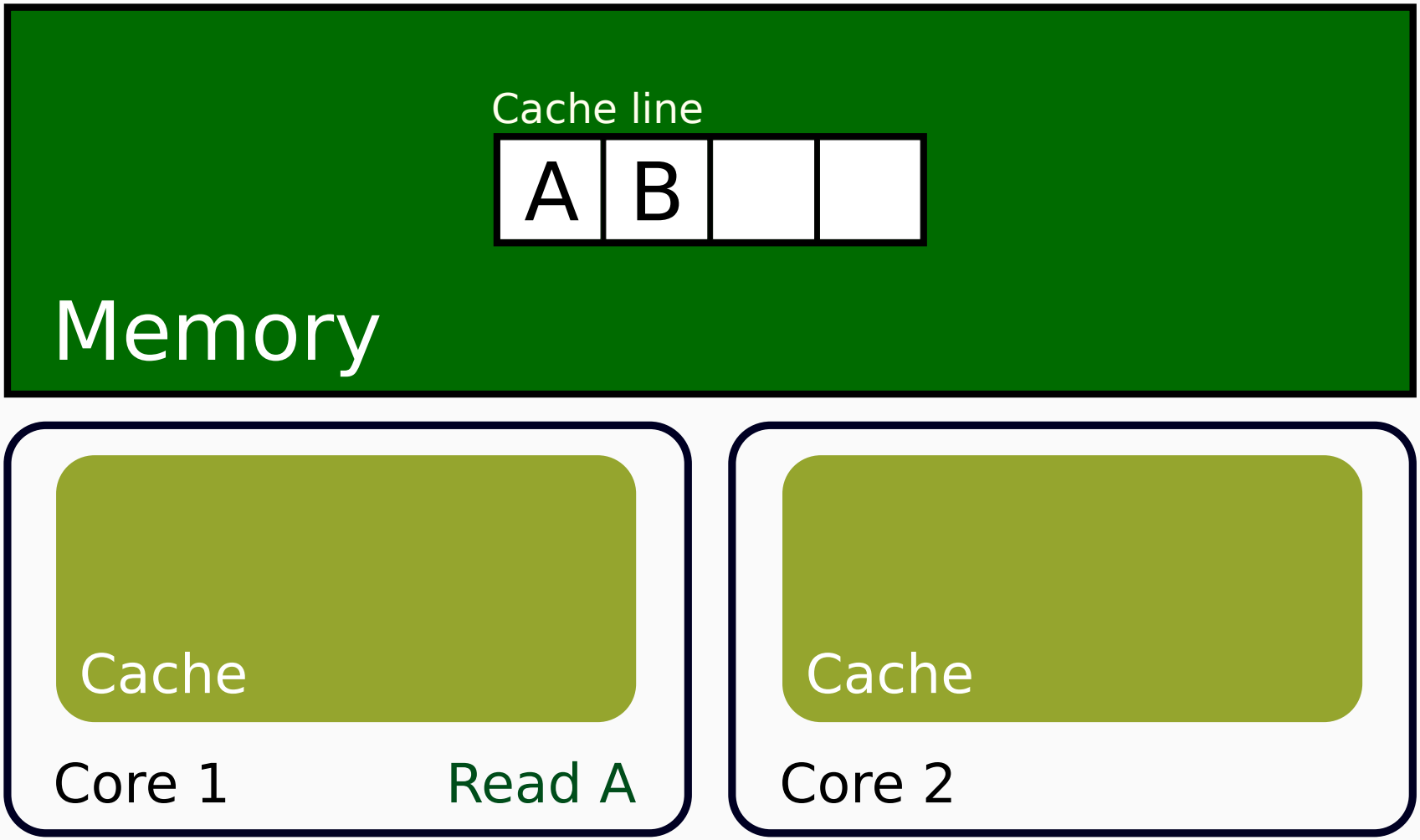
False sharing



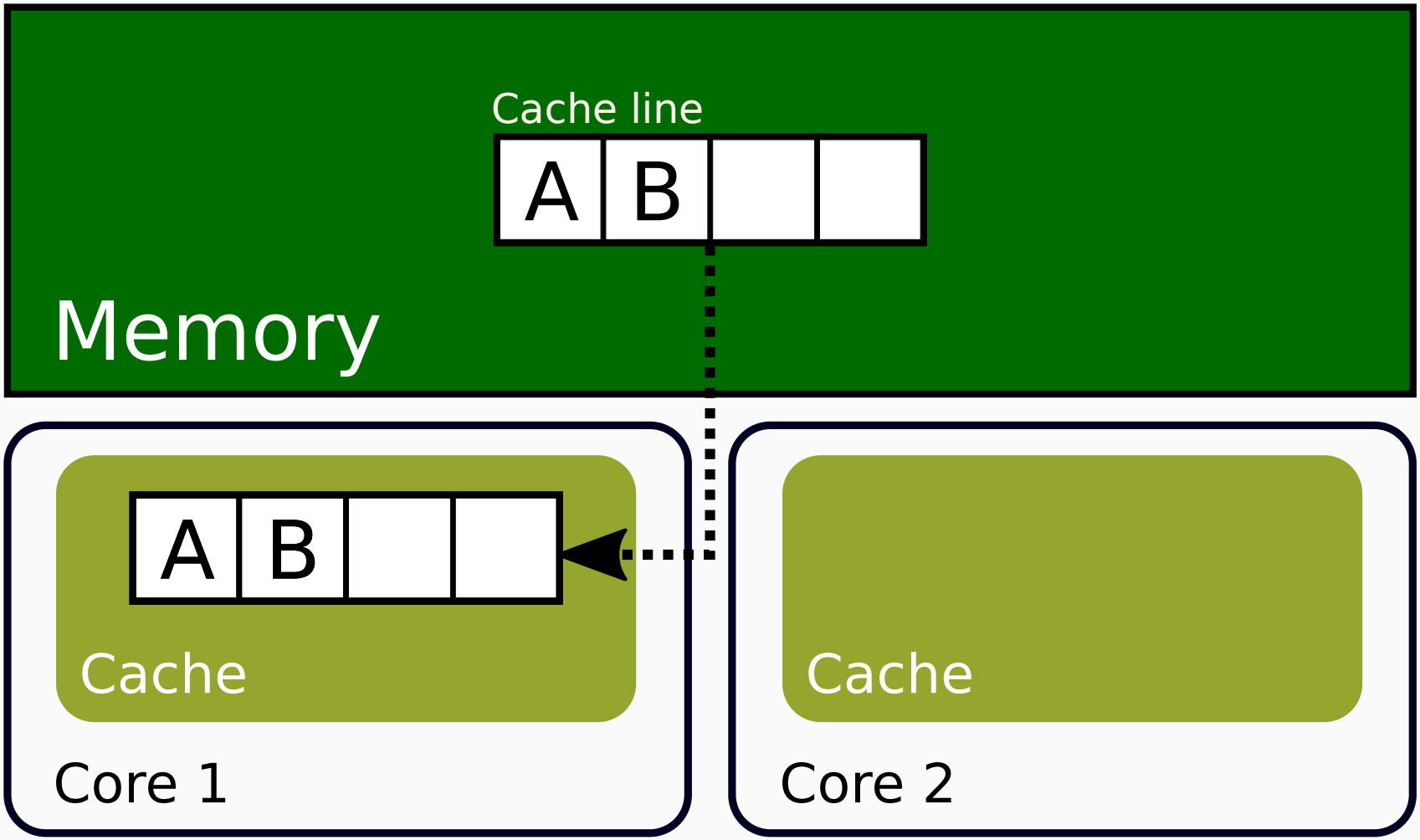
False sharing



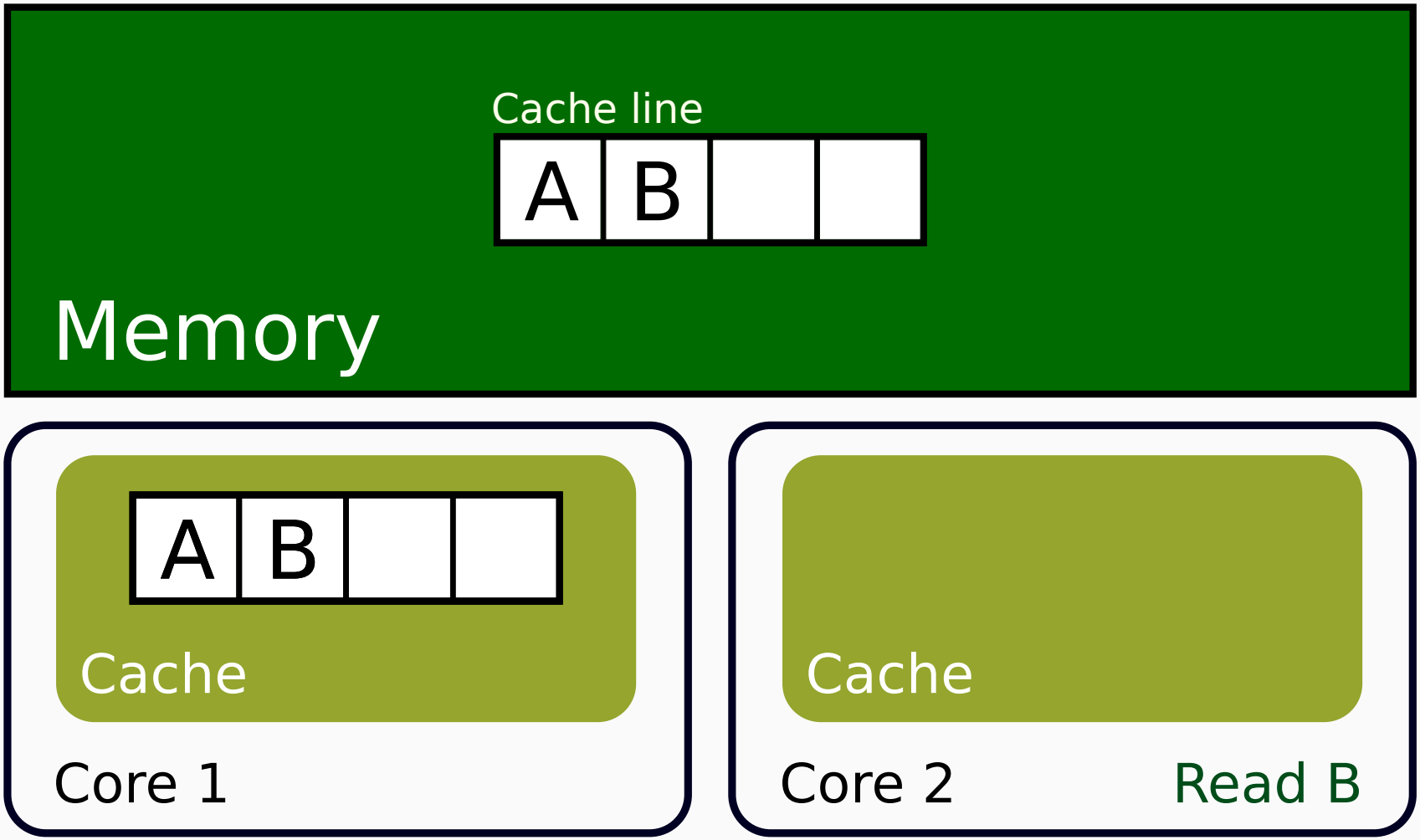
False sharing



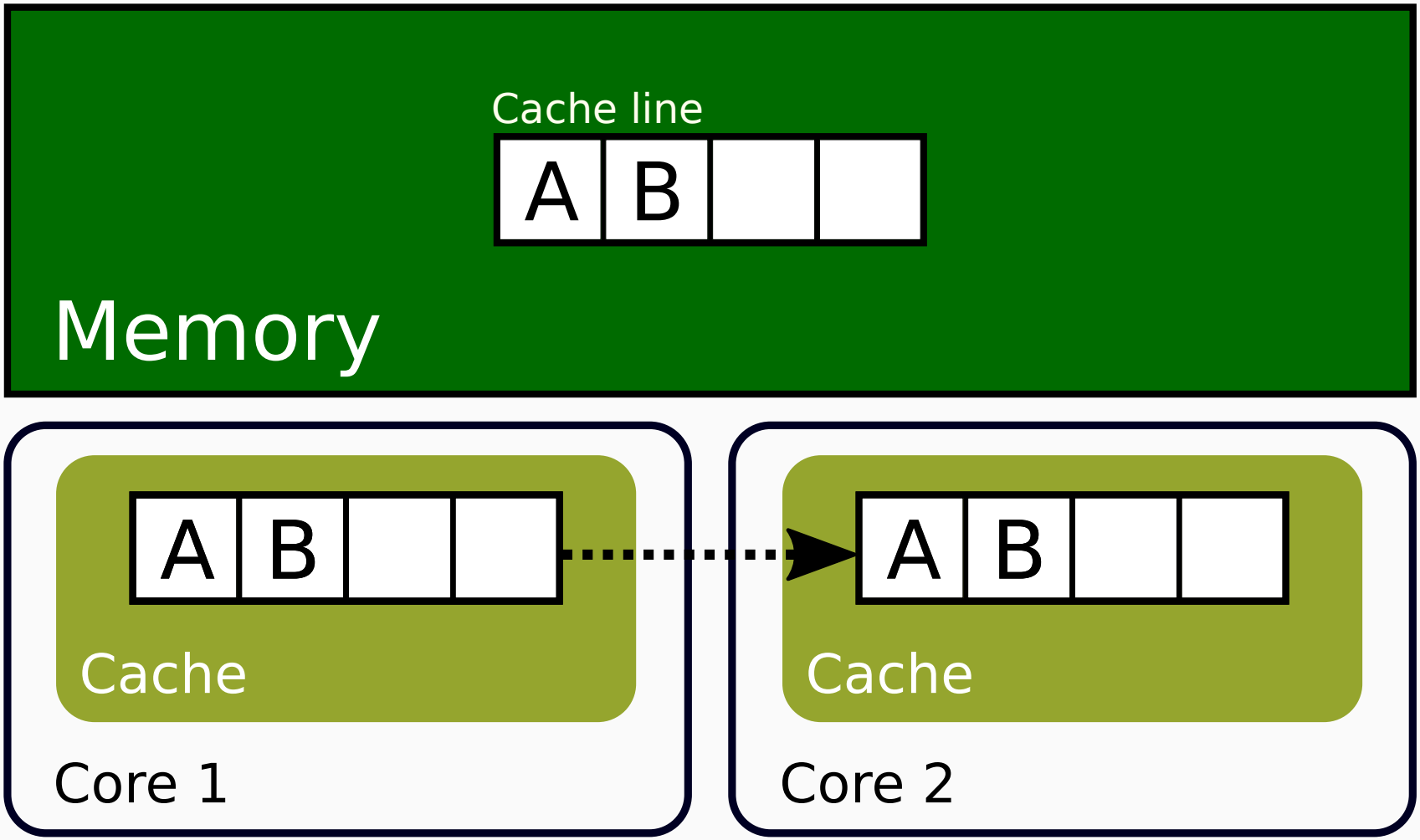
False sharing



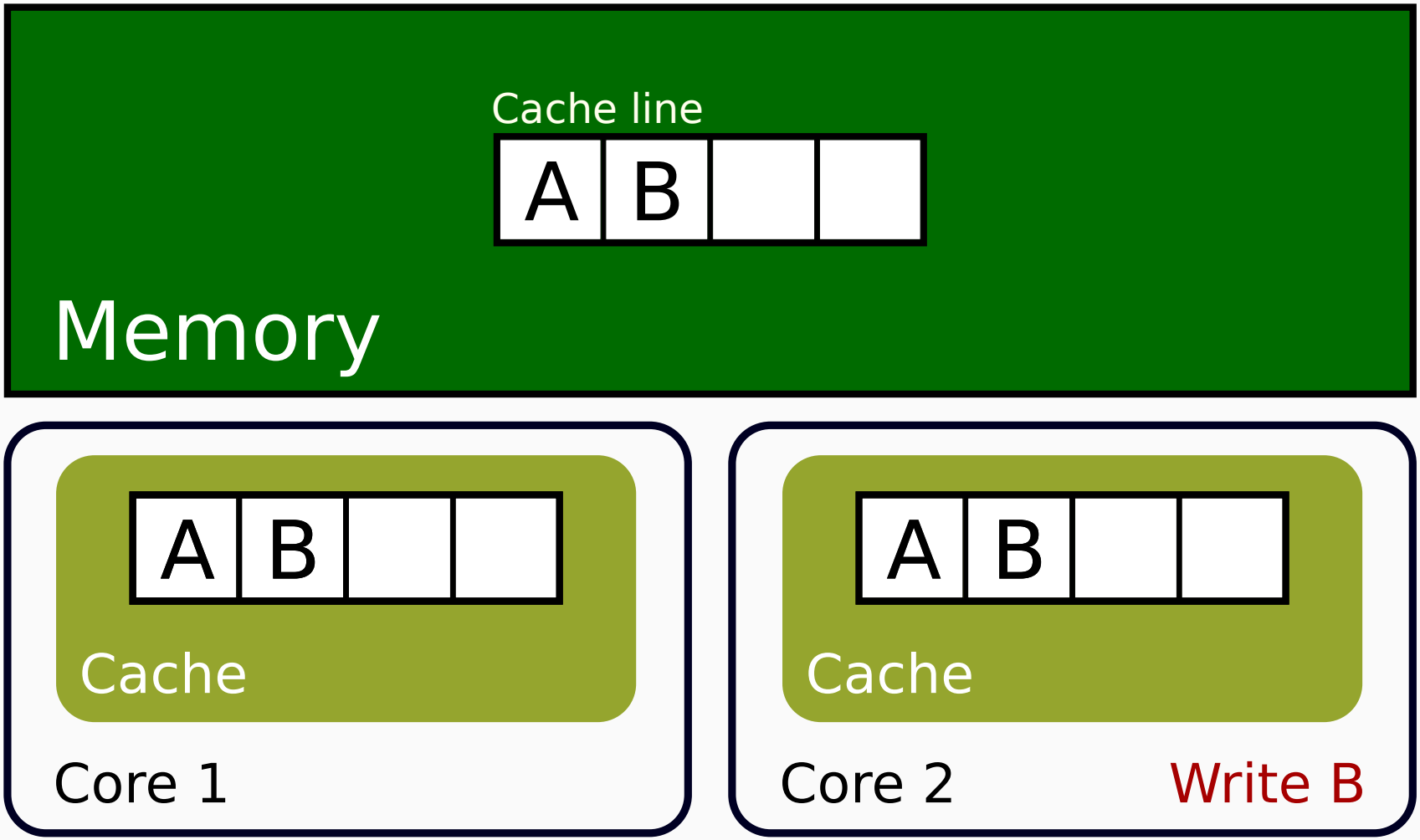
False sharing



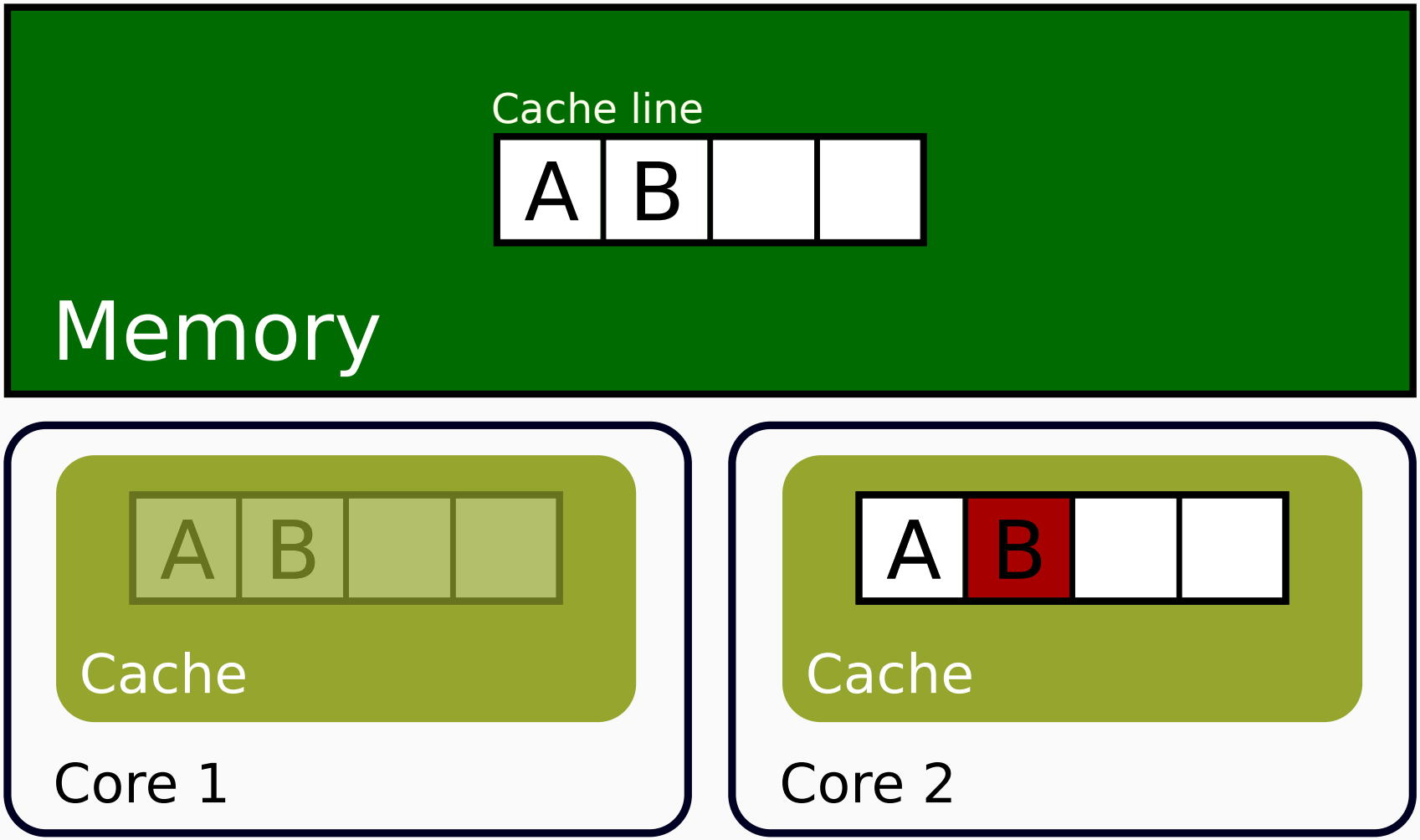
False sharing



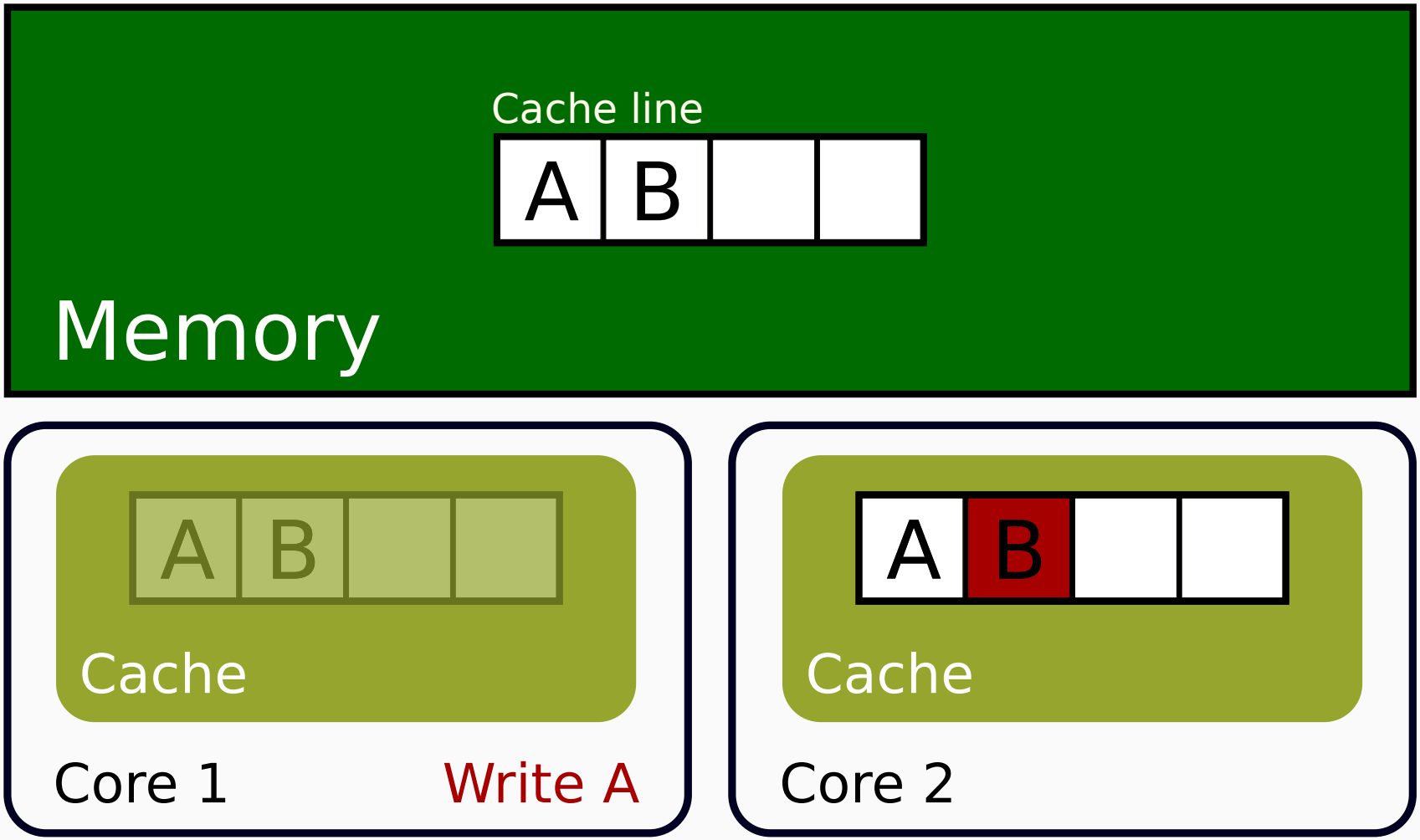
False sharing



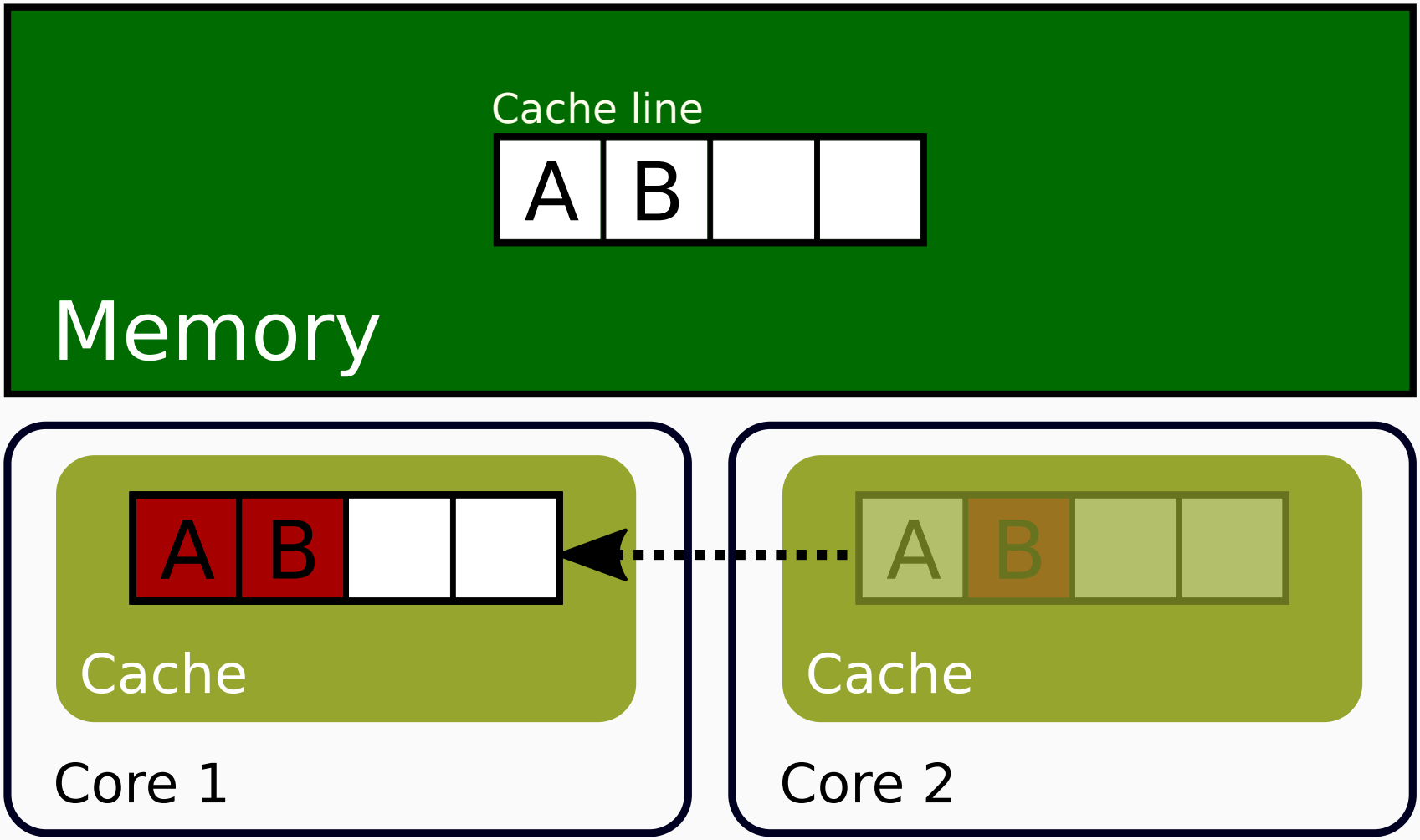
False sharing



False sharing



False sharing



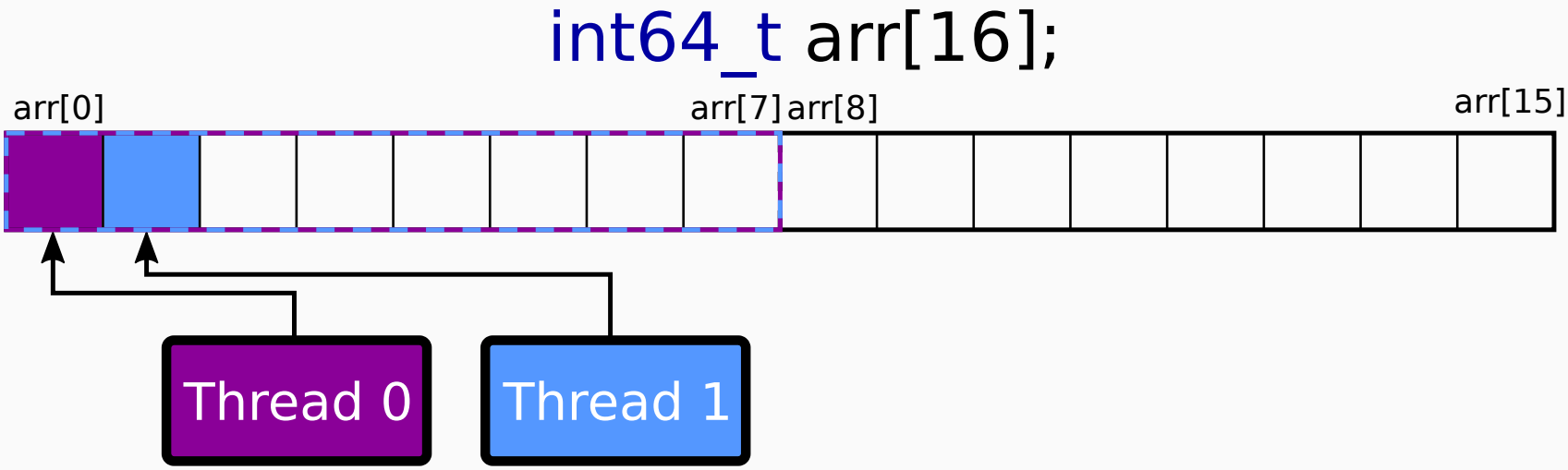
How to measure it?

`l2_rqsts.all_rfo`

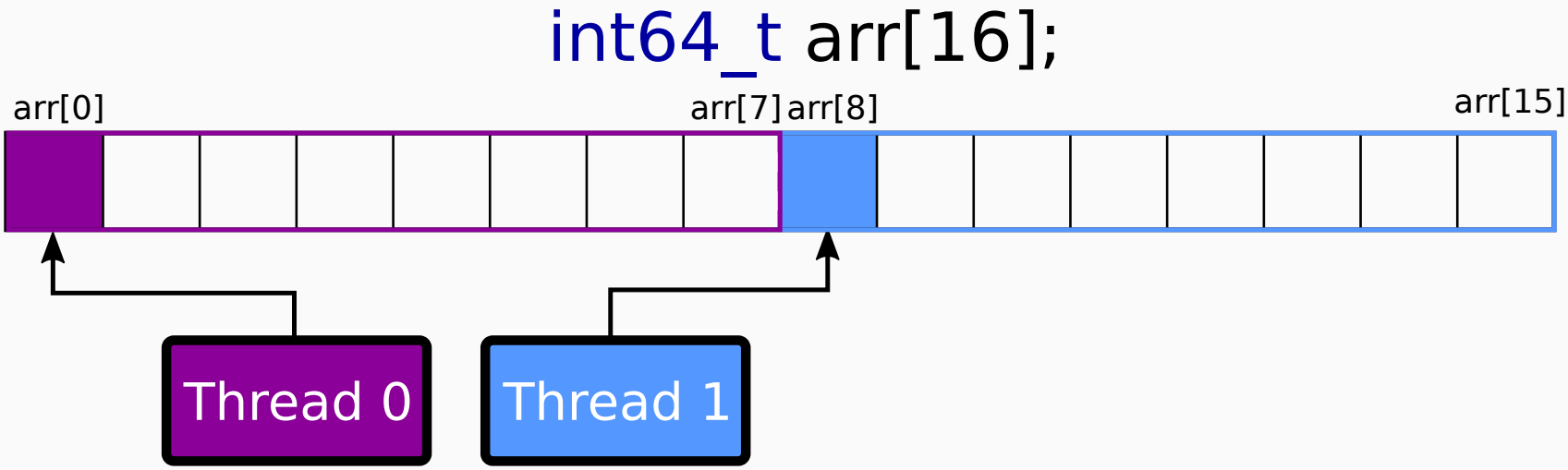
How many times some core invalidated data in other cores?

```
$ perf stat -e l2_rqsts.all_rfo ./my-program

        90 324 547 l2_rqsts.all_rfo
```

How to fix it?



How to fix it?

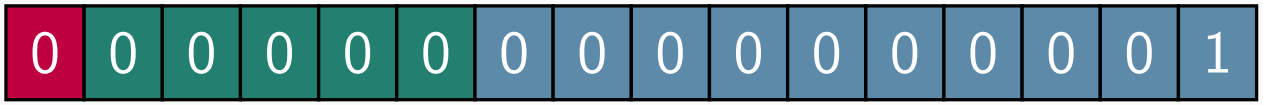


Denormal FP numbers

Numbers with zero exponent and non-zero significand

Denormal FP numbers

Numbers with zero exponent and non-zero significand

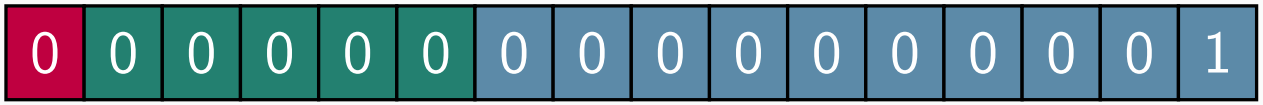


$(-1)^{0} * 2^{00000-\mathbf{01110}} * \mathbf{0}.0000000001$

- Numbers close to zero

Denormal FP numbers

Numbers with zero exponent and non-zero significand

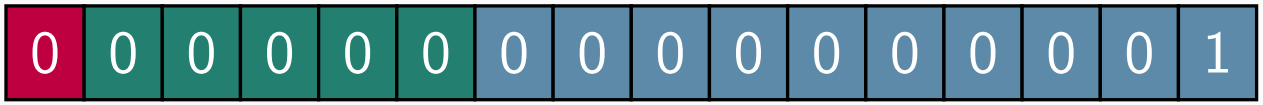


$(-1)^{0} * 2^{00000-\mathbf{01110}} * \mathbf{0}.0000000001$

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

Denormal FP numbers

Numbers with zero exponent and non-zero significand

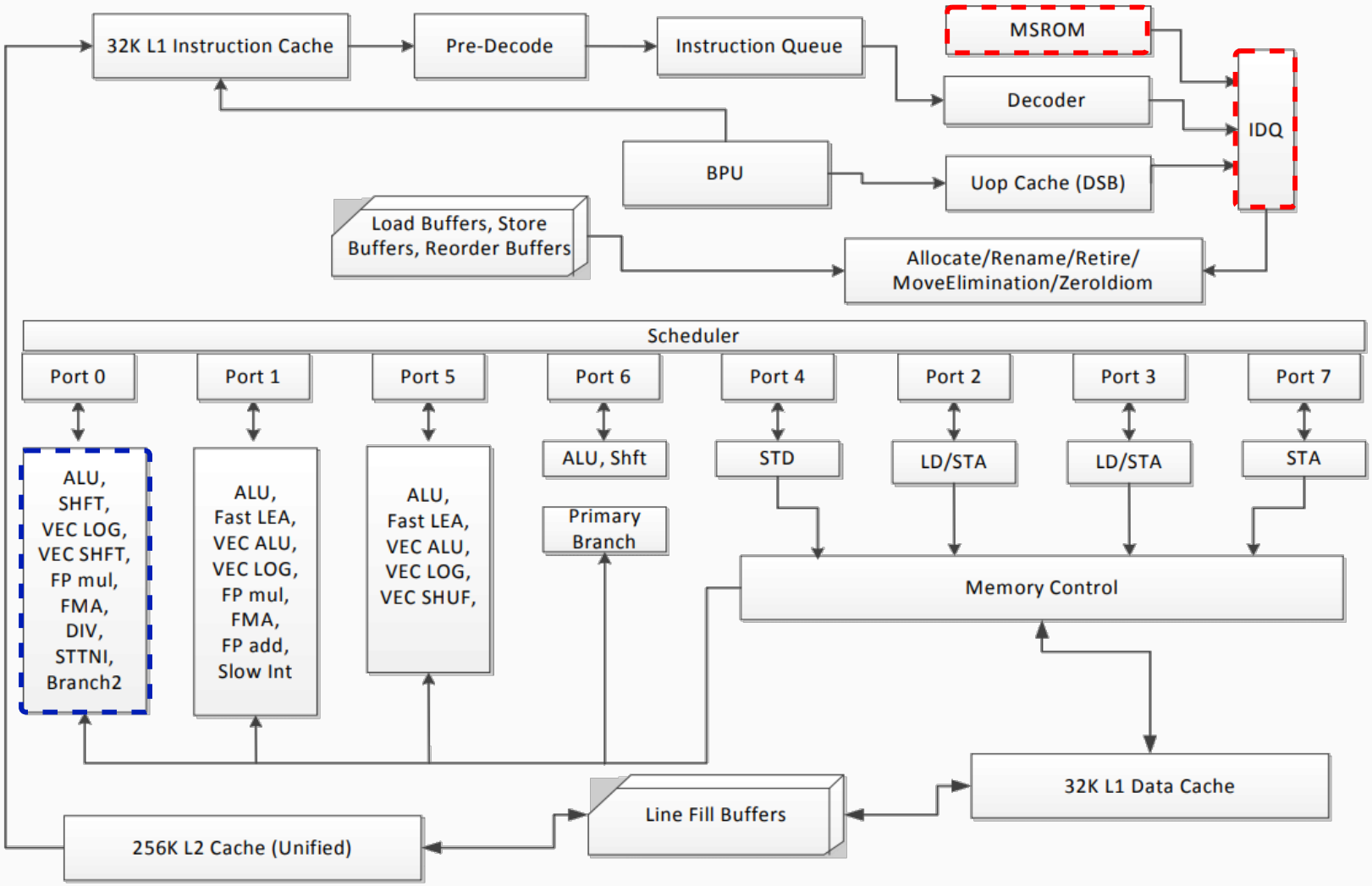


$(-1)^{0} * 2^{00000-\mathbf{01110}} * \mathbf{0}.0000000001$

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

Operations on denormal numbers are slow!

Denormal FP numbers



How to measure it?

`fp_assist.any`

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

```
$ perf stat -e fp_assist.any ./my-program

         9 437 184 fp_assist.any
```

How to fix it?

- Compiler switch `-ffast-math`

How to fix it?

- Compiler switch `-ffast-math`
- Set CPU flags:
  - Flush-to-zero - treat denormal outputs as 0
  - Denormals-as-zero - treat denormal inputs as 0

How to fix it?

- Compiler switch `-ffast-math`
- Set CPU flags:
  - Flush-to-zero - treat denormal outputs as 0
  - Denormals-as-zero - treat denormal inputs as 0

```
_mm_setcsr(_mm_getcsr() | 0x8040);
```

How to fix it?

- Compiler switch `-ffast-math`
- Set CPU flags:
  - Flush-to-zero - treat denormal outputs as 0
  - Denormals-as-zero - treat denormal inputs as 0

```
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
```

There are loads of other effects

- NUMA effects
- Data dependencies between instructions
- 4k aliasing
- Cache conflicts
- Misaligned accesses
- Software prefetching
- Non-temporal stores & cache pollution
- Bandwidth saturation
- AVX/SSE transition penalty
- ...

Thank you :-)

For more examples visit:

`https://github.com/kobzol/hardware-effects`

Jakub Beránek

Useful stuff

- `wikichip.org`
- Agner Fog optimization manual
- Intel x64 developer manual
- Compiler explorer :-)
- Intel intrinsics guide
- SIMD Visualiser