
LOW LATENCY
PROGRAMMING
Patrick Kostjens, Optiver
optiver

ABOUT ME

- MSc Computer Science, Utrecht University
- 8+ years of programming
- 3 years at Optiver

ABOUT THIS TALK

- Based on a Low Latency Workshop
- Process of improving latency
- Some useful tools — mostly Linux
- Actual C++ code
- The concepts can be applied elsewhere too

ABOUT OPTIVER

- Amsterdam, Sydney, Chicago, Shanghai, London
- 1986
- 440 people in Amsterdam
- ~45 nationalities
- Summer internship 2021!
- Graduate developer
- No trading knowledge required

ABOUT OPTIVER

- Market Making
- Provide prices
- Improve the market
- Calculate the correct price
- Being faster than the competition

WHAT IS LOW LATENCY PROGRAMMING

- It's not about throughput!
- Ability to quickly react to an event
- CPU-bound tasks, IO is outside of the scope
- It's all about time

THERE IS NO SILVER BULLET...

- No compiler flags
- No kernel config option
- No library

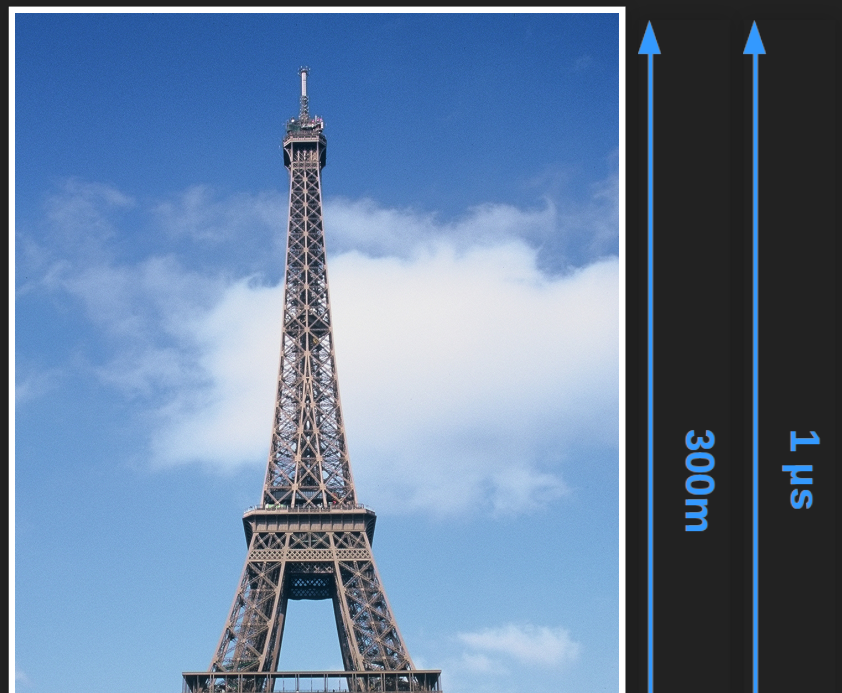
...BUT THERE ARE GOLDEN RULES

- Measure
- Do not optimize prematurely
- Know your problem
- Know your libraries
- Know your hardware

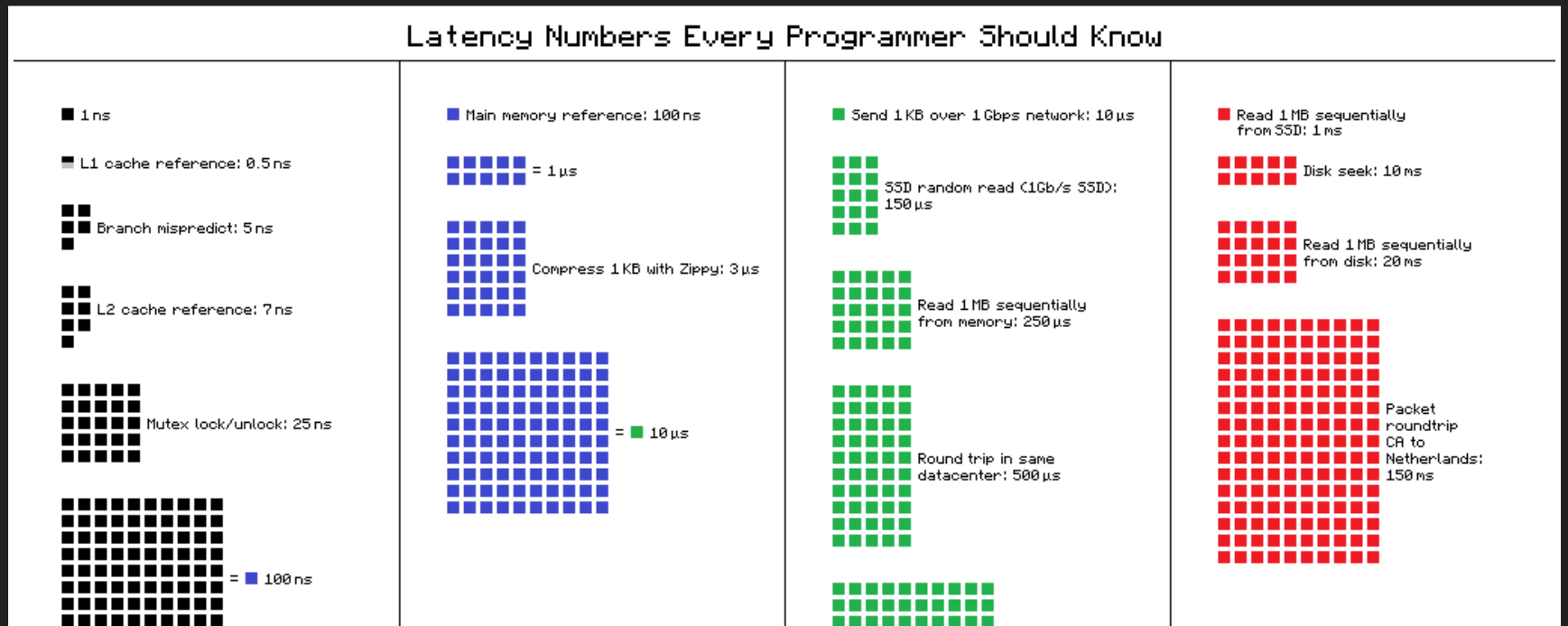
KNOW YOUR PROBLEM

- What *exactly* needs to be fast?
- What will you gain from making it faster?

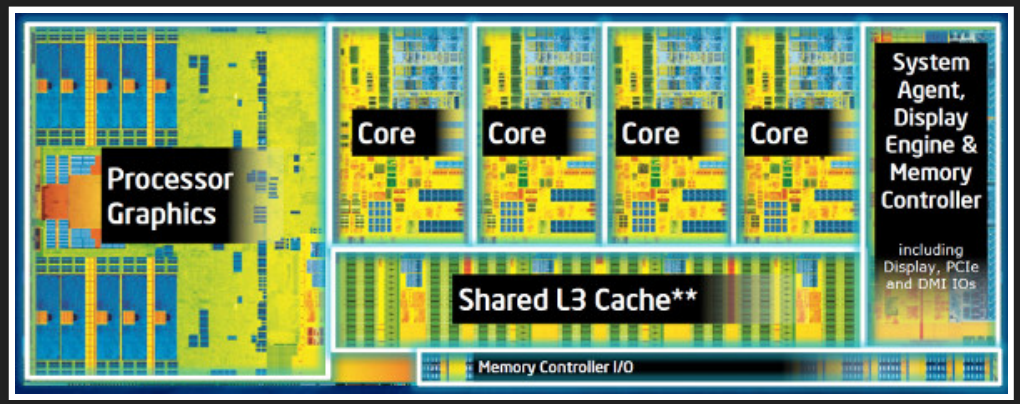
HOW FAST IS FAST?



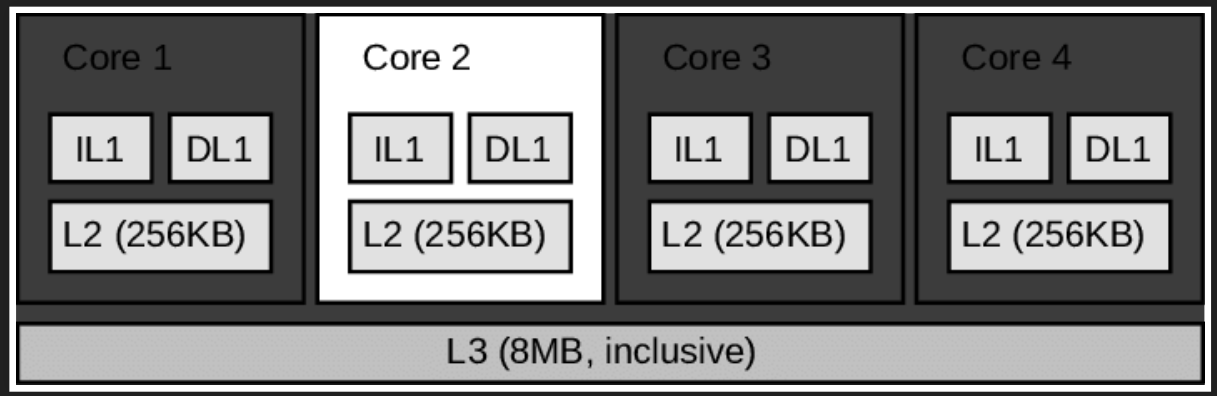
HOW FAST IS FAST?



CPU ARCHITECTURE - INTEL HASWELL



CPU ARCHITECTURE - INTEL HASWELL



STEP 1

IDENTIFY THE HOT PATH

IDENTIFY THE HOT PATH

- Profilers don't work for events
- Know which code is "hot"
- Microbenchmark the hot code

TODAY'S EXCERCISE

```
class Dictionary
{
public:
        Dictionary(const std::vector<std::string>& words);

        bool in_dictionary(std::string_view word) const;
};
```

GOOGLE BENCHMARK

- Allows for easy creation of microbenchmarks
- https://github.com/google/benchmark

GOOGLE BENCHMARK

```
void InDictionary(benchmark::State& state)
{
        int idx = 0;
        Dictionary dict(words);

        for (auto _: state)

        {
                std::string_view word = words[idx];
                idx = (idx+1) % words.size();
                benchmark::DoNotOptimize(dict.in_dictionary(word));
        }
}
```

GOOGLE BENCHMARK

```
void NotInDictionary(benchmark::State& state)
{
        int idx = 0;
        Dictionary dict(words);

        for (auto _: state)

        {
                std::string_view word = unknown_words[idx];
                idx = (idx+1) % unknown_words.size();
                benchmark::DoNotOptimize(dict.in_dictionary(word));
        }
}
```

GOOGLE BENCHMARK

```
BENCHMARK(InDictionary);
BENCHMARK(NotInDictionary);

BENCHMARK_MAIN();
```

GOOGLE BENCHMARK

```
Run on (4 X 3504 MHz CPU s)
2018-01-17 14:19:55
------------------------------------------------------------------------
Benchmark                                 Time           CPU Iterations
------------------------------------------------------------------------
InDictionary                            725 ns        725 ns     884585
NotInDictionary                         812 ns        811 ns     838950
```

NAIVE IMPLEMENTATION

```
class Dictionary
{
public:
        Dictionary(const std::vector<std::string>& words)
                : _container(words.begin(), words.end())
        {}


        bool in_dictionary(std::string_view word) const
        {
                return _container.find({word.data(), word.size()}) != _containe
        }

private:
        std::set<std::string> _container;
};
```

STEP 2

ALGORITHMIC COMPLEXITY

VALGRIND & CALLGRIND

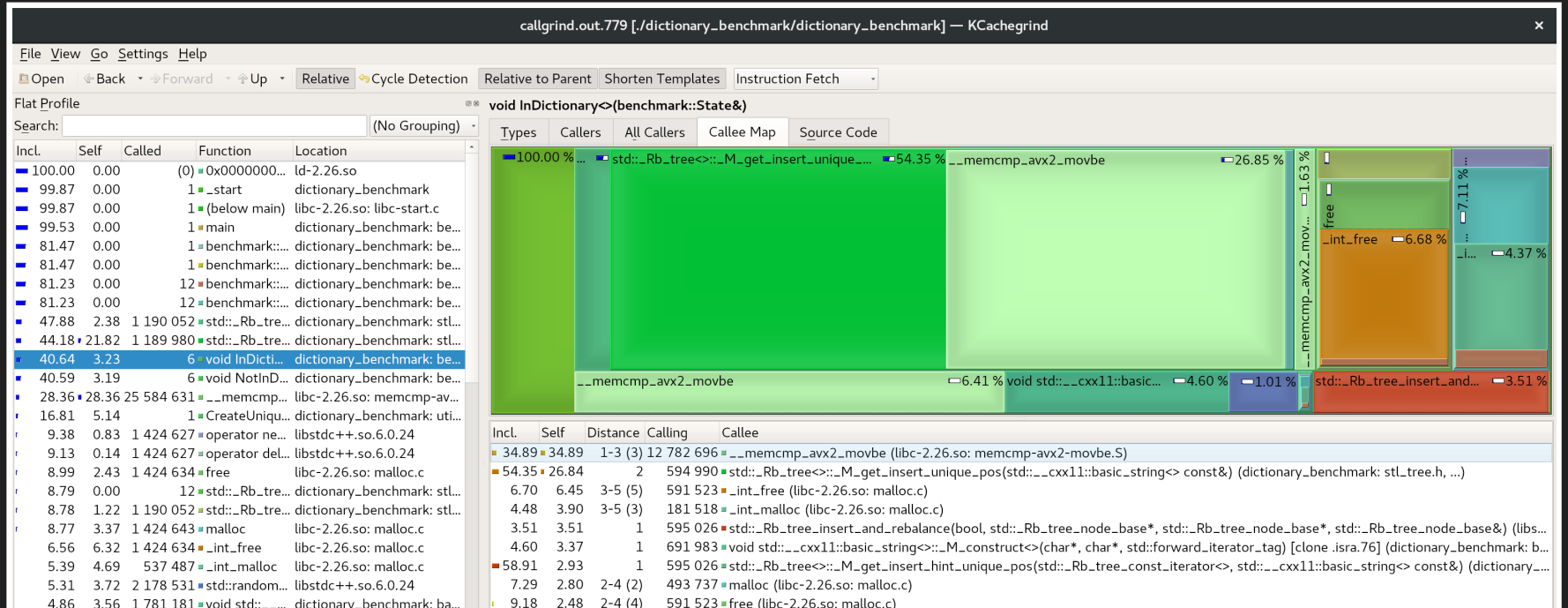
- Instrumentation framework
- Unix only
- Callgrind - profiler
- KCachegrind - GUI

VALGRIND & CALLGRIND

```
$ valgrind --tool=callgrind ./benchmark
$ kcachegrind
```

KCACHEGRIND

```
$ valgrind --tool=callgrind ./benchmark
$ kcachegrind
```



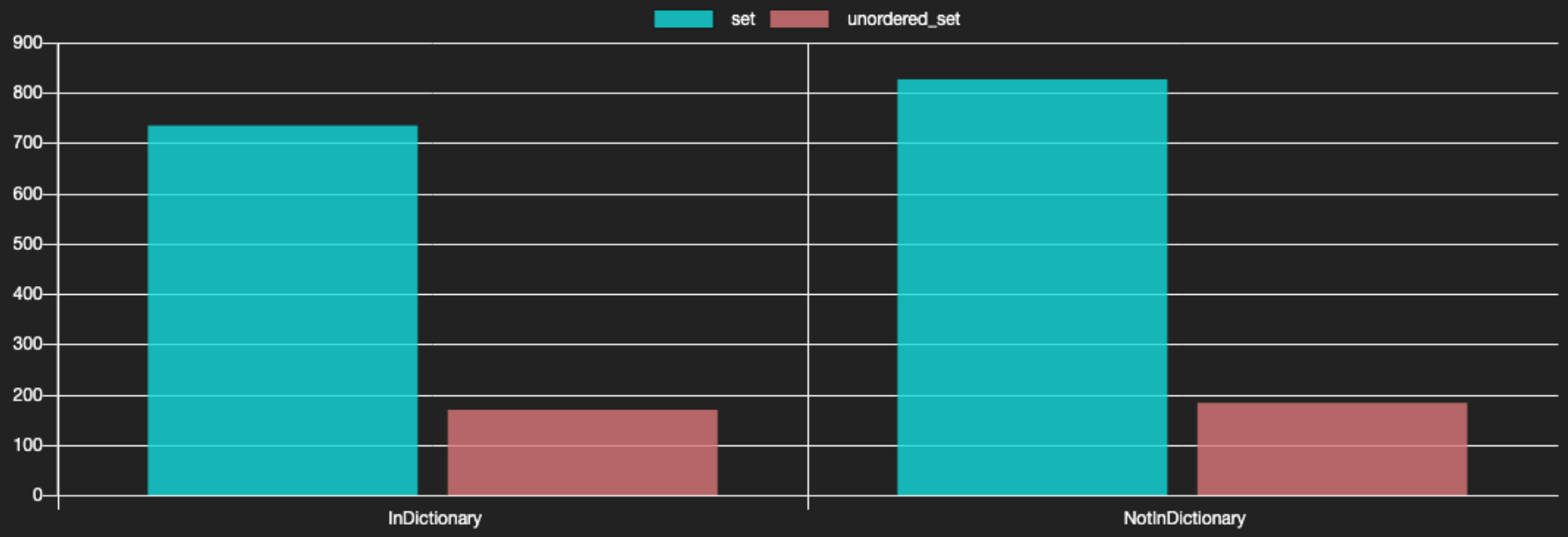
NAIVE IMPLEMENTATION

```
class Dictionary
{
public:
        Dictionary(const std::vector<std::string>& words)
                : _container(words.begin(), words.end())
        {}


        bool in_dictionary(std::string_view word) const
        {
                return _container.find({word.data(), word.size()}) != _containe
        }

private:
        std::set<std::string> _container;
};
```

IMPROVED IMPLEMENTATION

```
class Dictionary
{
public:
        Dictionary(const std::vector<std::string>& words)
                : _container(words.begin(), words.end())
        {}


        bool in_dictionary(std::string_view word) const
        {
                return _container.find({word.data(), word.size()}) != _containe
        }

private:
        std::unordered_set<std::string> _container; // <-- O(1)
};
```

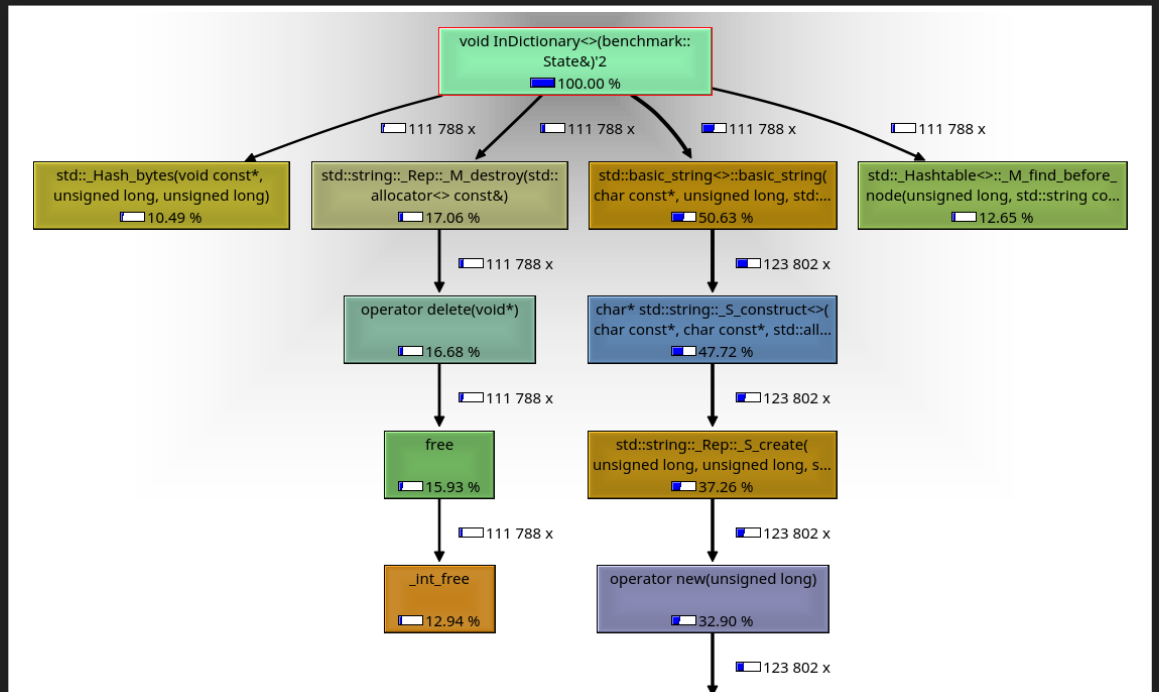
BENCHMARK: IMPROVING ALGORITHMIC COMPLEXITY



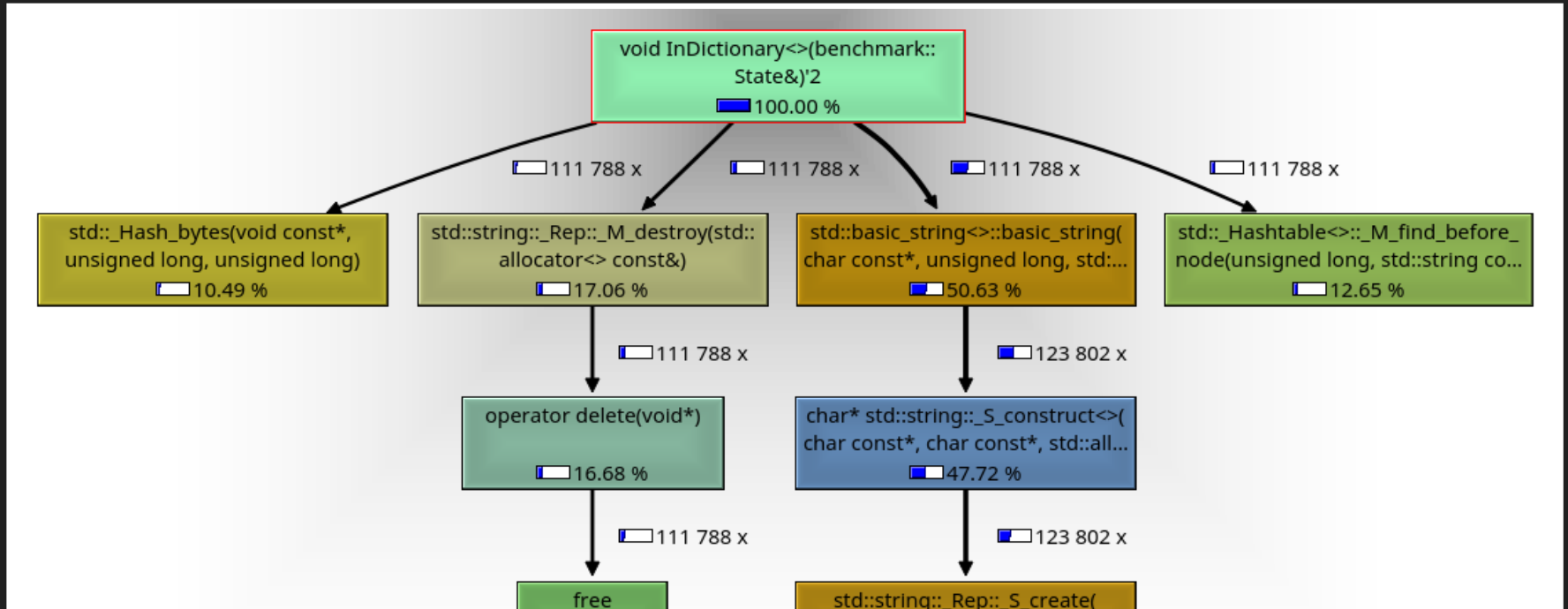
STEP 3

ALLOCATIONS

KCACHEGRIND OUTPUT



KCACHEGRIND OUTPUT



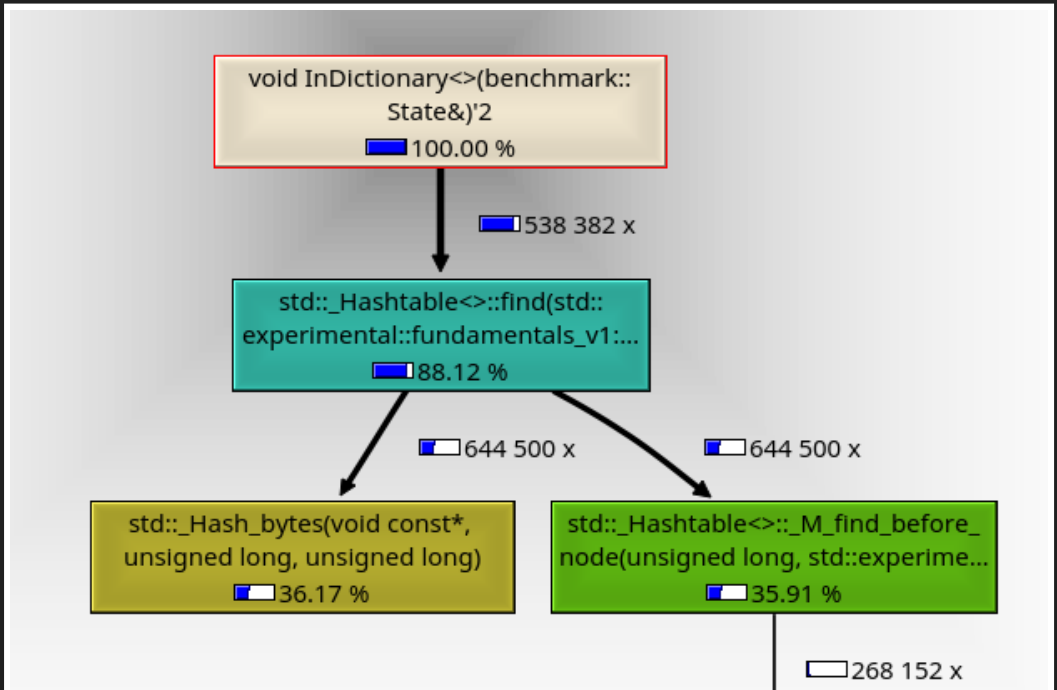
IMPROVED IMPLEMENTATION

```
class Dictionary
{
public:
        Dictionary(const std::vector<std::string>& words)
                : _container(words.begin(), words.end())
        {}


        bool in_dictionary(std::string_view word) const
        {
                // This line allocates, creating std::string from std::string_v
                return _container.find({word.data(), word.size()}) != _containe
        }

private:
        std::unordered_set<std::string> _container; // Stores std::string
};
```

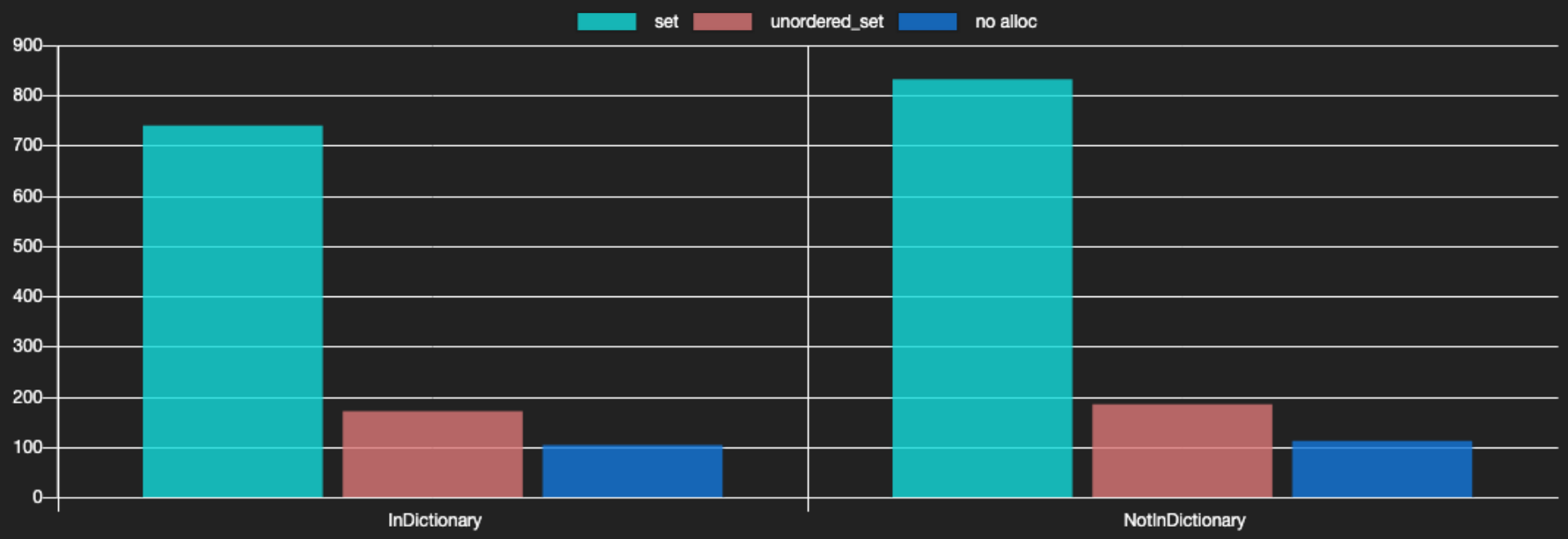
NON-ALLOCATING IMPLEMENTATION

```
class Dictionary
{
public:
        Dictionary(const std::vector<std::string>& words)
                : _data(words.begin(), words.end())
                , _index(_data.begin(), _data.end())
        {}


        bool in_dictionary(std::string_view word) const
        {
                return _index.find(word) != _index.end(); // No allocation!
        }

private:
        std::vector<std::string> _data; // Stores std::string
        std::unordered_set<std::string_view> _index; // Stores views
};
```

KCACHEGRIND OUTPUT



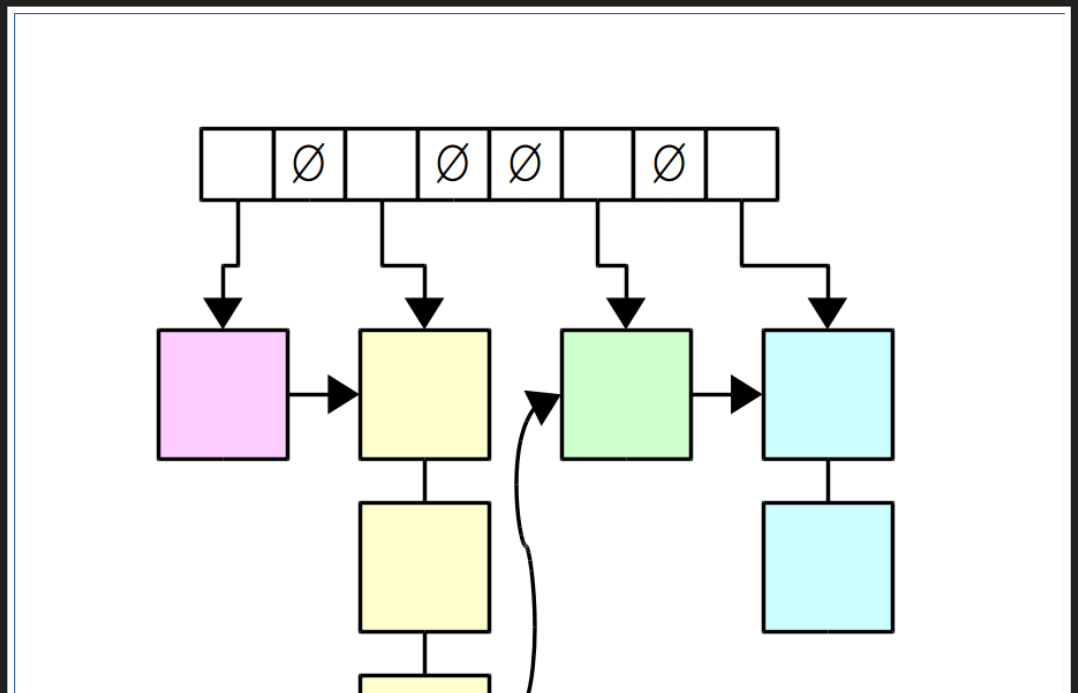
BENCHMARK: REMOVING ALLOCATIONS



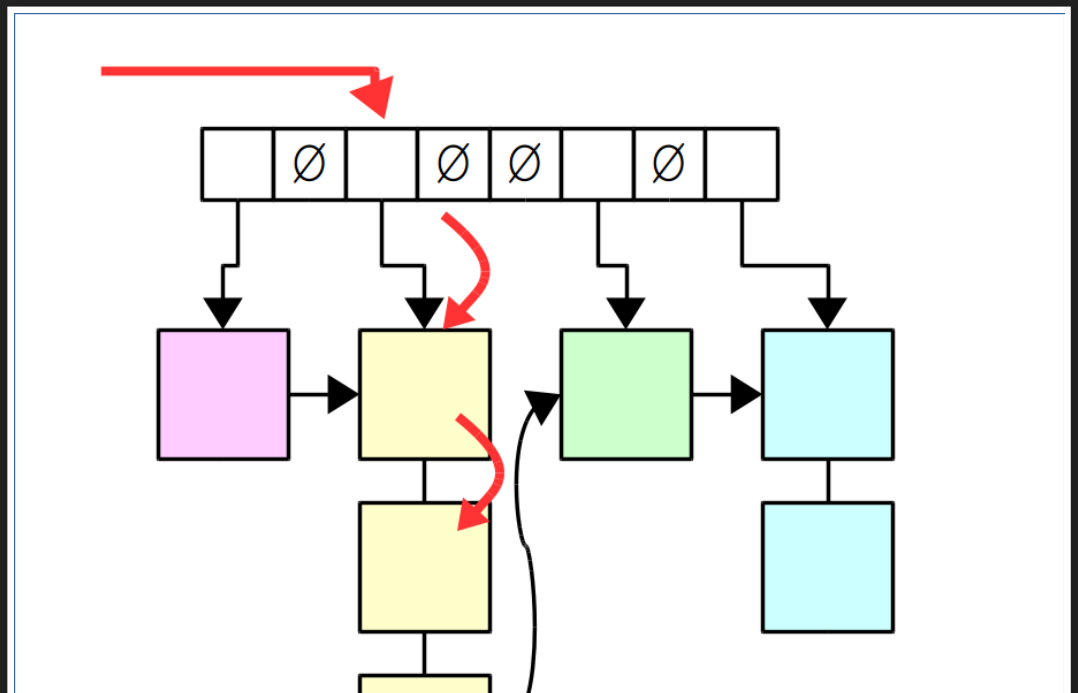
STEP 4

CACHE COHERENCE

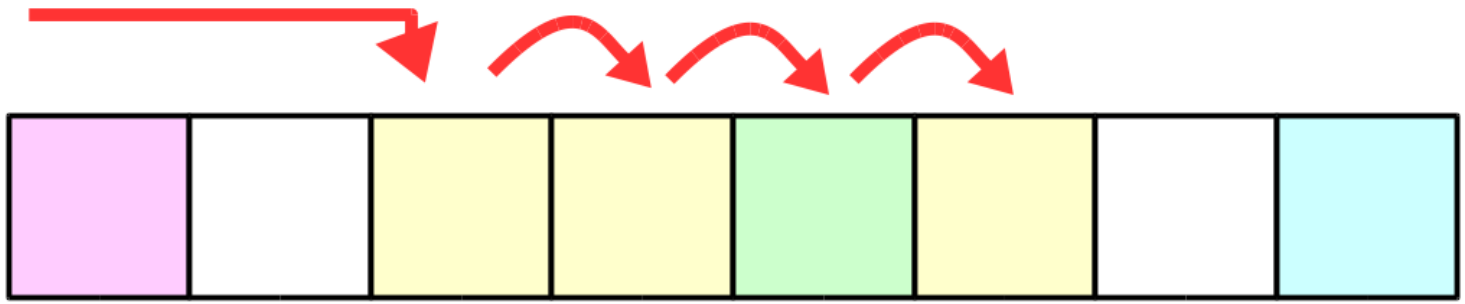
STD::UNORDERED_SET



STD::UNORDERED_SET



OPEN ADDRESSING



PAPI++

- Provides access to Hardware Performance Counters from within the program
- https://github.com/david-grs/papipp

PAPI - EXAMPLE USE

```
papi::event_set<PAPI_TOT_INS, PAPI_TOT_CYC, PAPI_BR_MSP, PAPI_L1_DCM> e
events.start_counters();

// Micro-benchmark loop is here


events.stop_counters();
std::cout
        << events.get<PAPI_L1_DCM>().counter() / state.iterations()
        << " L1 cache misses";
```

PAPI - OUTPUT

```
98 L1 cache misses
16 L1 cache misses
7.89 L1 cache misses
6.617 L1 cache misses
6.3805 L1 cache misses

6.03766 L1 cache misses
6.03299 L1 cache misses
6.03683 L1 cache misses
```

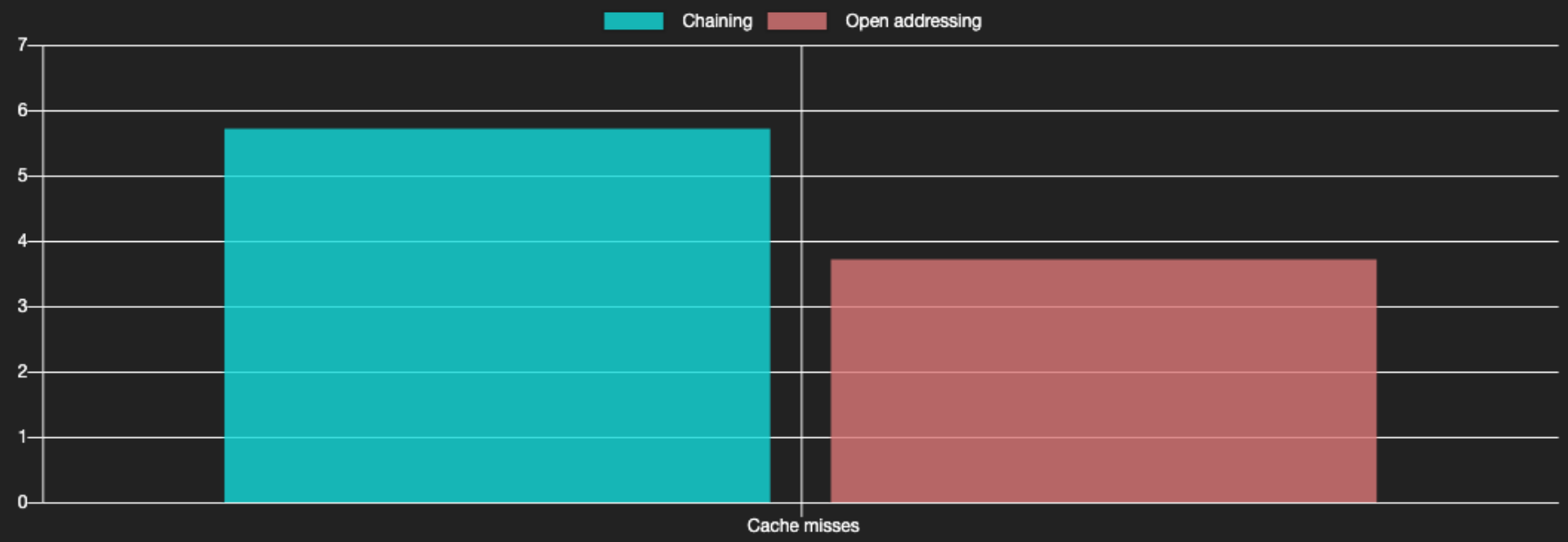
OPEN ADDRESSING IMPLEMENTATION

```
struct Entry
{
        std::size_t hash;
        const std::string* string = nullptr;
};

std::vector<Entry> hashTable_;
```

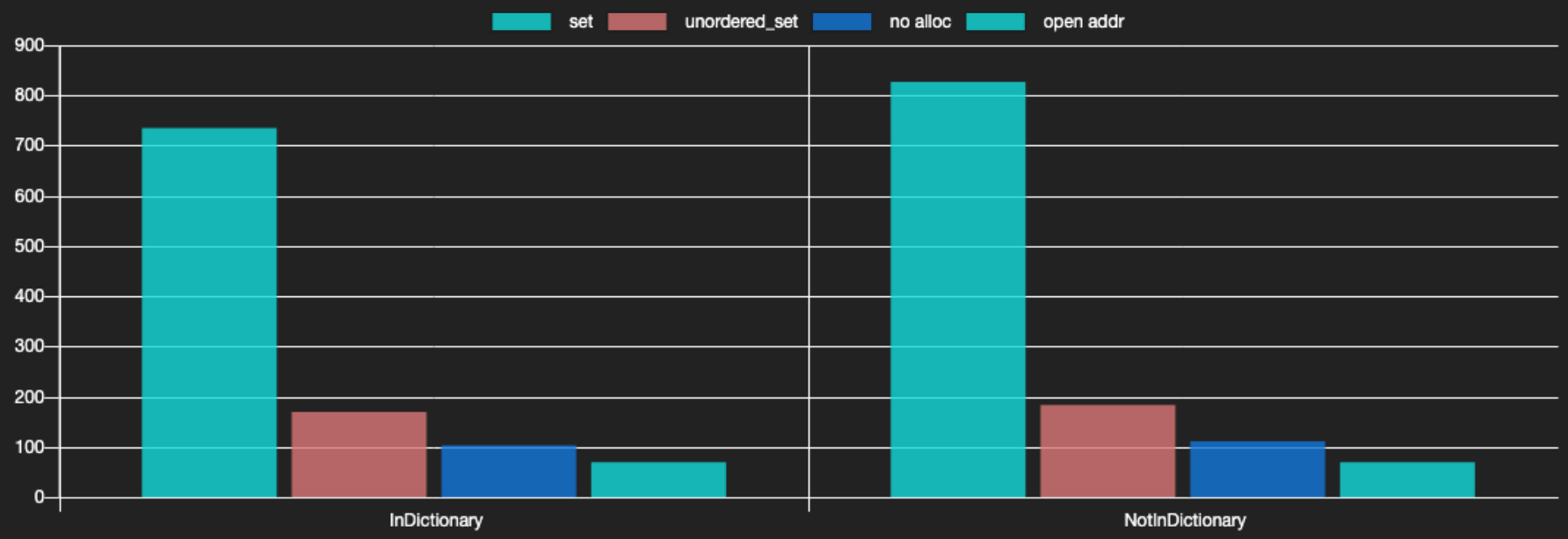
OPEN ADDRESSING IMPLEMENTATION

```
bool in_dictionary(std::string_view word) const
{
        auto hash = std::hash<std::string_view>()(word);
        auto idx = hash % hashTable_.size();

        while (true)
        {
                const Entry& entry = hashTable_[idx];

                // detect bucket
                if (!entry.string)
                        return false;

                // detect match
                if(entry.hash == hash && *entry.string == word)
                        return true;

                // probe
                idx = (idx + 1) % hashTable_.size();
        }
}
```

PAPI++: CACHE MISSES



BENCHMARK: OPEN ADDRESSING



STEP 5

CPU INSTRUCTIONS

PERF

- "The official" Linux profiler
- Has access to the same counters as PAPI
- Instruments running binary
- Produces nice reports
- And so much more!

PERF REPORT

```
 0,12 |        mov    %edx,0x8(%rsp)
      |        mov    $0xc70f6907,%edx
 0,16 |      → callq  std::_Hash_bytes(void const*, unsigned long, unsigned long)@plt
      |                       auto idx = hash % hashTable_.size();
      |        xor    %edx,%edx
 0,12 |        mov    %rax,%r14
18,01 |        div    %rbx
      |            const_reference operator[](size_type __n) const _GLIBCXX_NOEXCEPT
      |            { return *(this->_M_impl._M_start + __n); }
 0,08 |        mov    %rdx,%rax
      |        mov    %rdx,%r12
 0,47 |        shl    $0x4,%rax
 0,27 |        add    %r15,%rax
      |                                  if (!entry.string)
35,52 |        mov    0x8(%rax),%rdx
 0,12 |        mov    %rax,%rcx
      |        test   %rdx,%rdx
 0,47 |      ↓ jne    1a4
      |      ↓ jmpq   230
      |        nop
      |                                  idx = (idx+1) % hashTable_.size();
 3,06 |180:    lea    0x1(%r12),%rax
      |        xor    %edx,%edx
 5,89 |        div    %rbx
      |        mov    %rdx,%rcx
      |        mov    %rdx,%r12
```

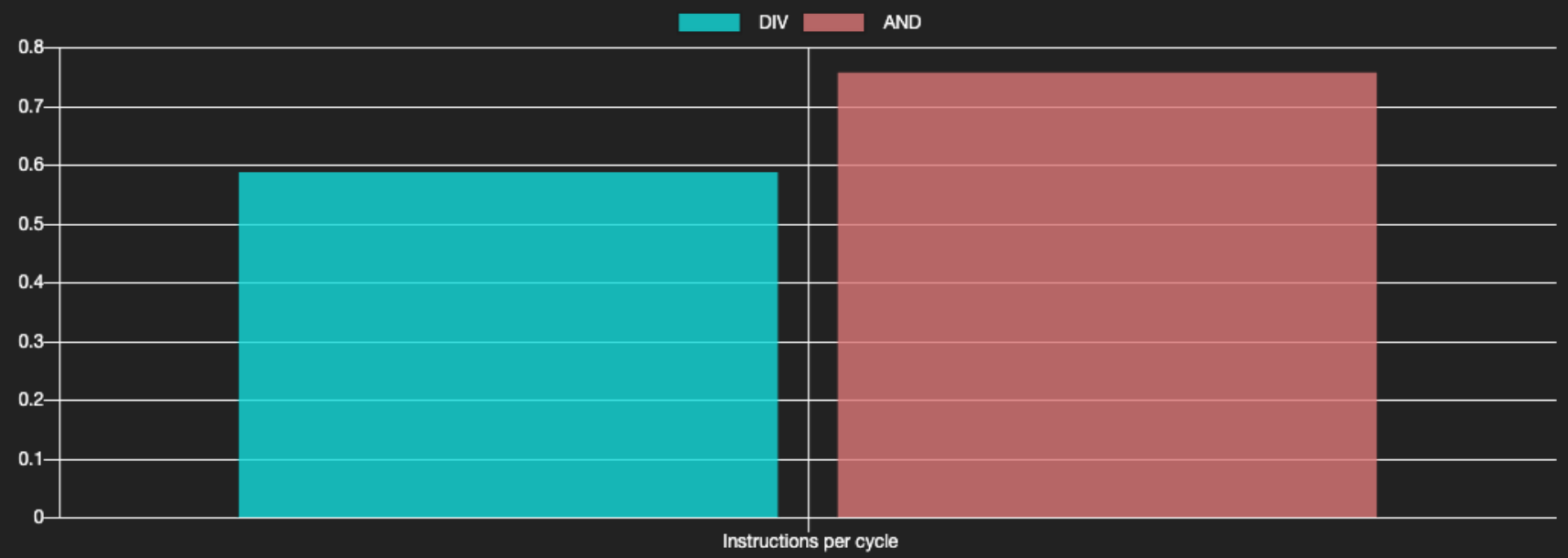
FINDING OFFENDING CODE

```
bool in_dictionary(std::string_view word) const
{
        auto hash = std::hash<std::string_view>()(word);
        auto idx = hash % hashTable_.size(); // <-- Flagged by perf

        while (true)
        {
                const Entry& entry = hashTable_[idx];

                // detect bucket
                if (!entry.string)
                        return false;

                // detect match
                if(entry.hash == hash && *entry.string == word)
                        return true;

                // probe
                idx = (idx + 1) % hashTable_.size(); // <-- Flagged by
        }
}
```

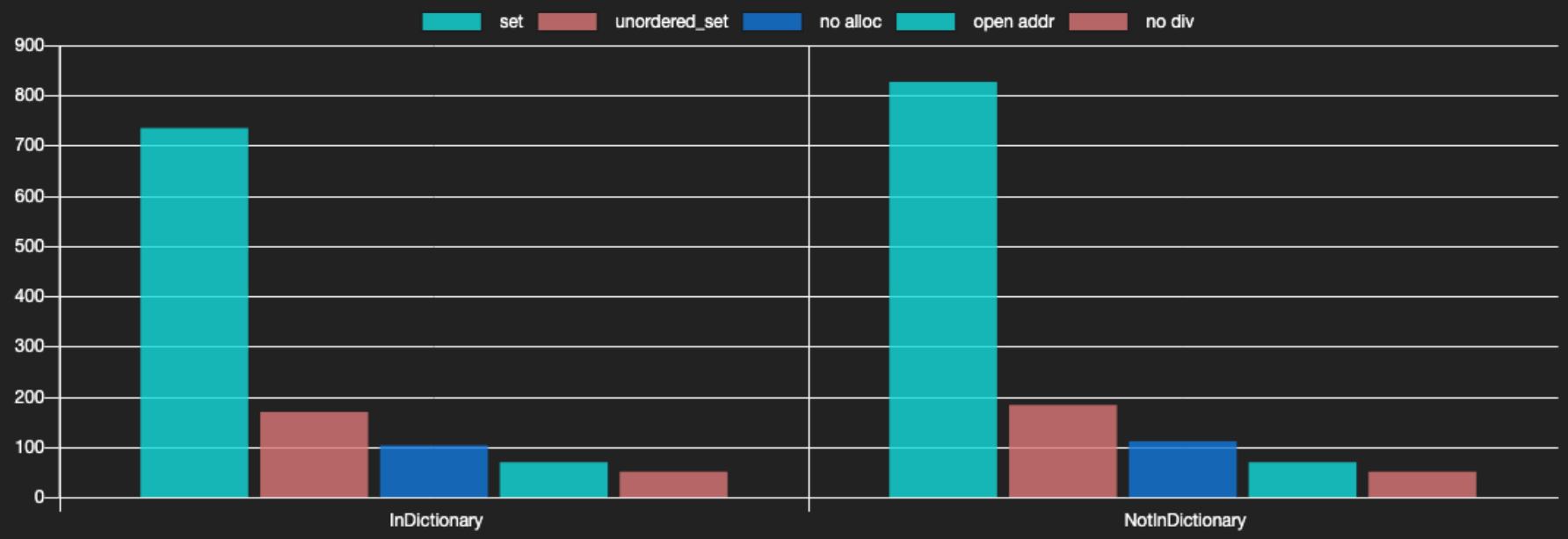
FIXING OFFENDING CODE

```
bool in_dictionary(std::string_view word) const
{
        auto hash = std::hash<std::string_view>()(word);
        auto idx = hash & (hashTable_.size() - 1); // Fixed?

        while (true)
        {
                const Entry& entry = hashTable_[idx];

                // detect bucket
                if (!entry.string)
                        return false;

                // detect match
                if(entry.hash == hash && *entry.string == word)
                        return true;

                // probe
                idx = (idx + 1) & (hashTable_.size() - 1); // Fixed?
        }
}
```

PAPI++: INSTRUCTIONS PER CYCLE



BENCHMARK: REMOVING DIV



SUMMARY

- Always measure
- Keep an eye on algorithmic complexity
- Avoid allocations / synchronization
- Use cache efficiently
- Look into generated assembly

LINKS, RESOURCES, TOOLS

- “When a Microsecond Is an Eternity", Carl Cook, CppCon 2017
- Papi++ : https://github.com/david-grs/papipp
- Google Benchmark
- valgrind --tool=callgrind, KCachegrind
- perf
- **https://www.optiver.com**


QUESTIONS?