

The background image shows a canal scene at night in Amsterdam. Traditional Dutch houses with warm lights glowing from their windows are reflected in the dark water. A stone bridge arches over the canal, and a small boat is visible on the right side.

LOW LATENCY PROGRAMMING

Patrick Kostjens, Optiver

optiver △

A night photograph of a canal in Amsterdam. Traditional Dutch houses with gabled roofs and warm interior lights are reflected in the dark water. A stone bridge arches over the canal, and a small boat is visible on the right.

ABOUT ME

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

The background image shows a canal scene in Amsterdam at night. Traditional Dutch houses with many windows and gabled roofs are reflected in the dark water of the canal. A bridge arches over the water, and several boats are moored along the bank. The scene is lit by streetlights and the warm glow from the windows of the buildings.

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

A night photograph of a canal in Amsterdam. Traditional Dutch houses with multiple gables and warm lights from windows and street lamps are reflected in the dark water. A bridge arches over the canal, and boats are visible on the right.

ABOUT OPTIVER

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

The background image shows a canal scene in Amsterdam at night. Traditional Dutch houses with multiple gables are reflected in the dark water. A bridge arches over the canal, and streetlights illuminate the scene.

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

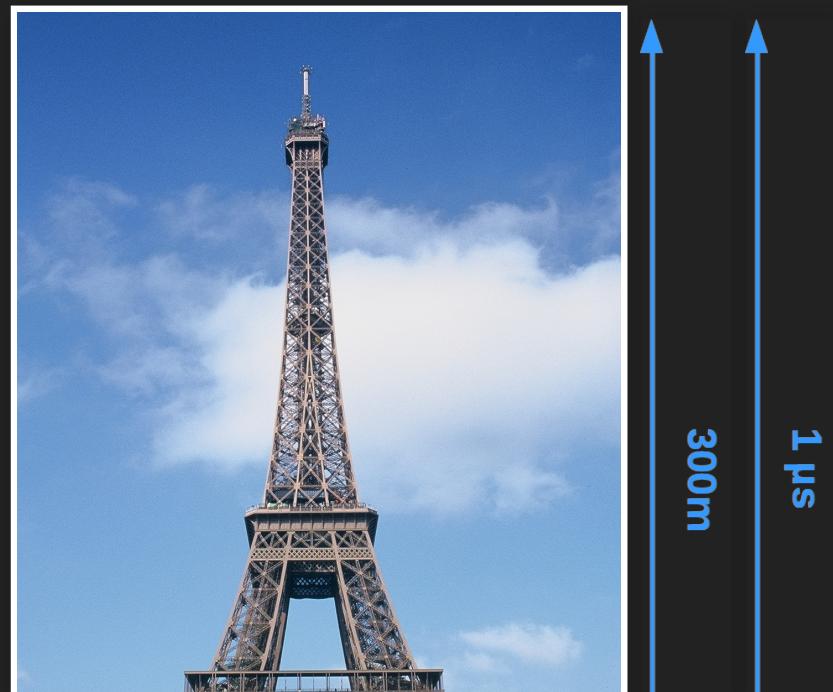
...BUT THERE ARE GOLDEN RULES

- 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?

Latency Numbers Every Programmer Should Know

■ 1 ns

■ L1 cache reference: 0.5 ns

■ Branch mispredict: 5 ns

■ L2 cache reference: 7 ns

■ Mutex lock/unlock: 25 ns

■ 100 ns

■ Main memory reference: 100 ns

■ = 1 μs

■ Compress 1 KB with Zippy: 3 μs

■ = 10 μs

■ Send 1 KB over 1 Gbps network: 10 μs

■ SSD random read (1Gb/s SSD): 150 μs

■ Read 1MB sequentially from memory: 250 μs

■ Round trip in same datacenter: 500 μs

■

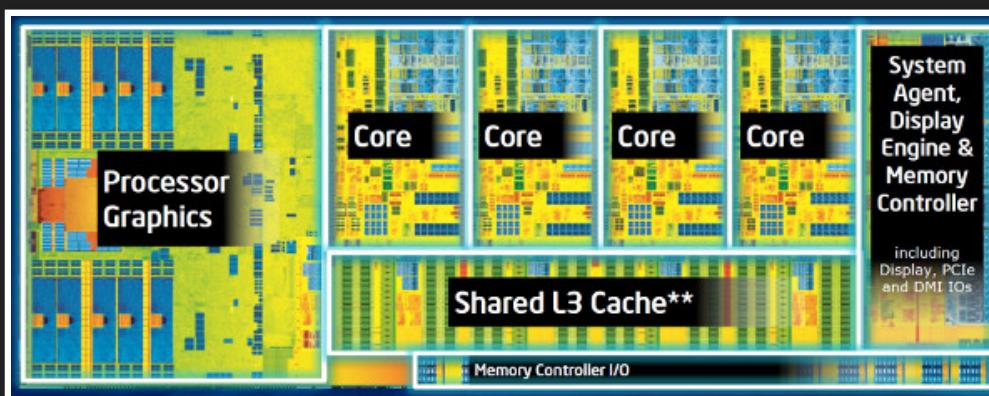
■ Read 1 MB sequentially from SSD: 1 ms

■ Disk seek: 10 ms

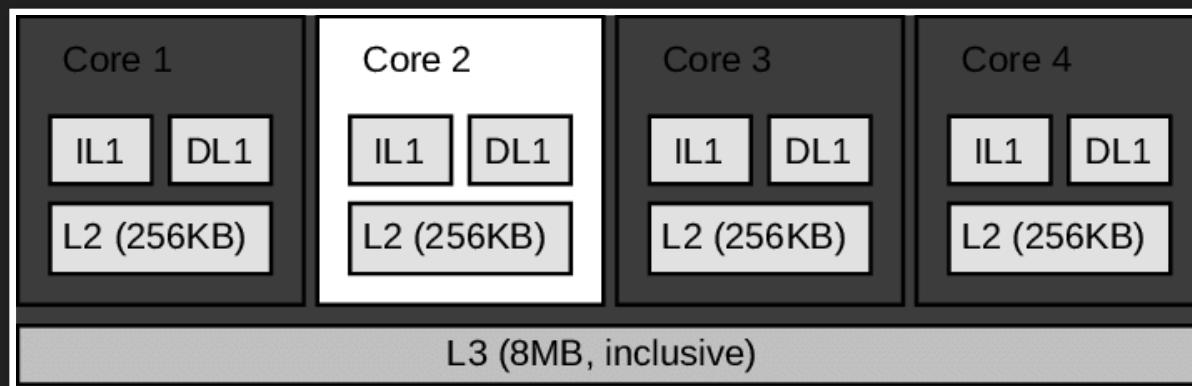
■ Read 1 MB sequentially from disk: 20 ms

■ Packet roundtrip CA to Netherlands: 150 ms

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()}) != _container.end();
    }

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

STEP 2

ALGORITHMIC COMPLEXITY

VALGRIND & CALLGRIND

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

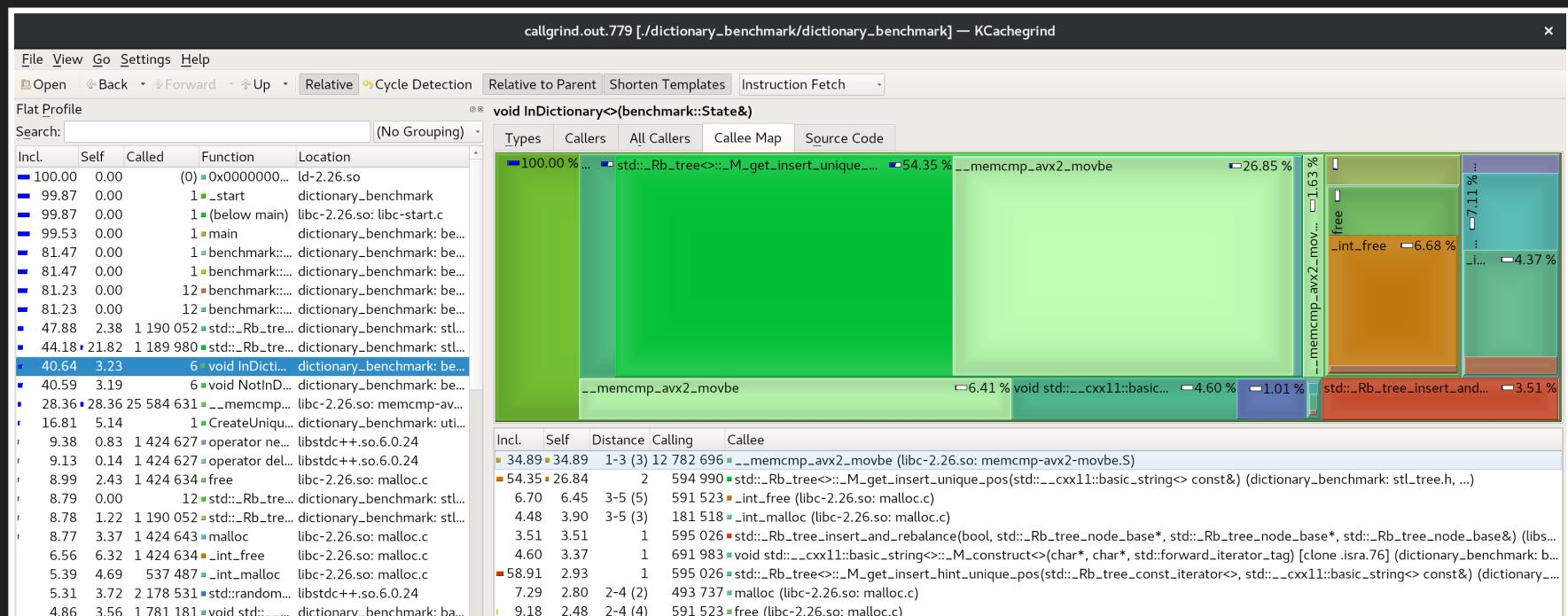


VALGRIND & CALLGRIND

```
$ 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()}) != _container.end();
    }

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

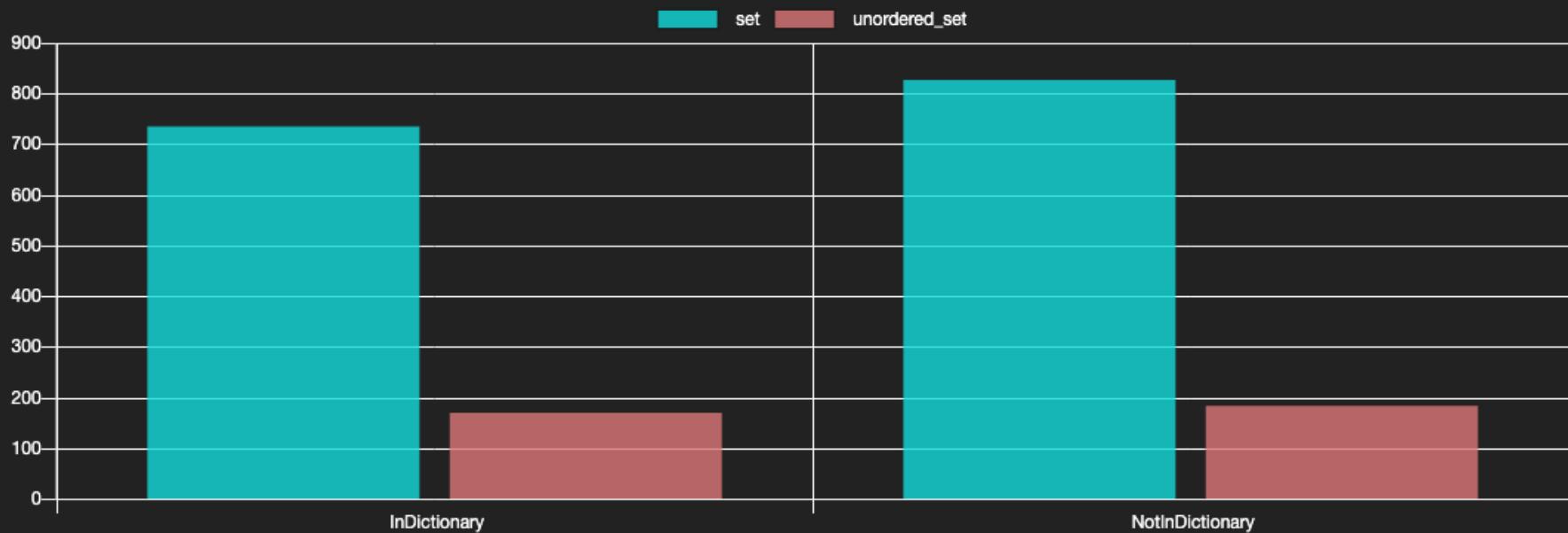
IMPROVED IMPLEMENTATION

```
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()}) != _container.end();
    }

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

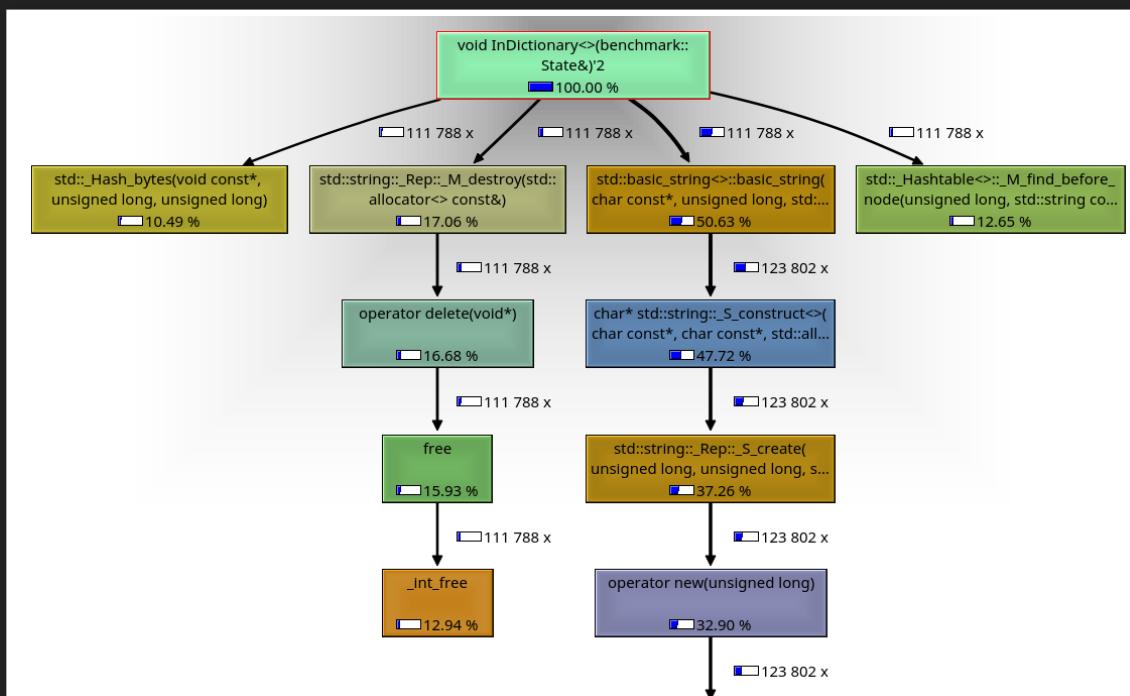
BENCHMARK: IMPROVING ALGORITHMIC COMPLEXITY



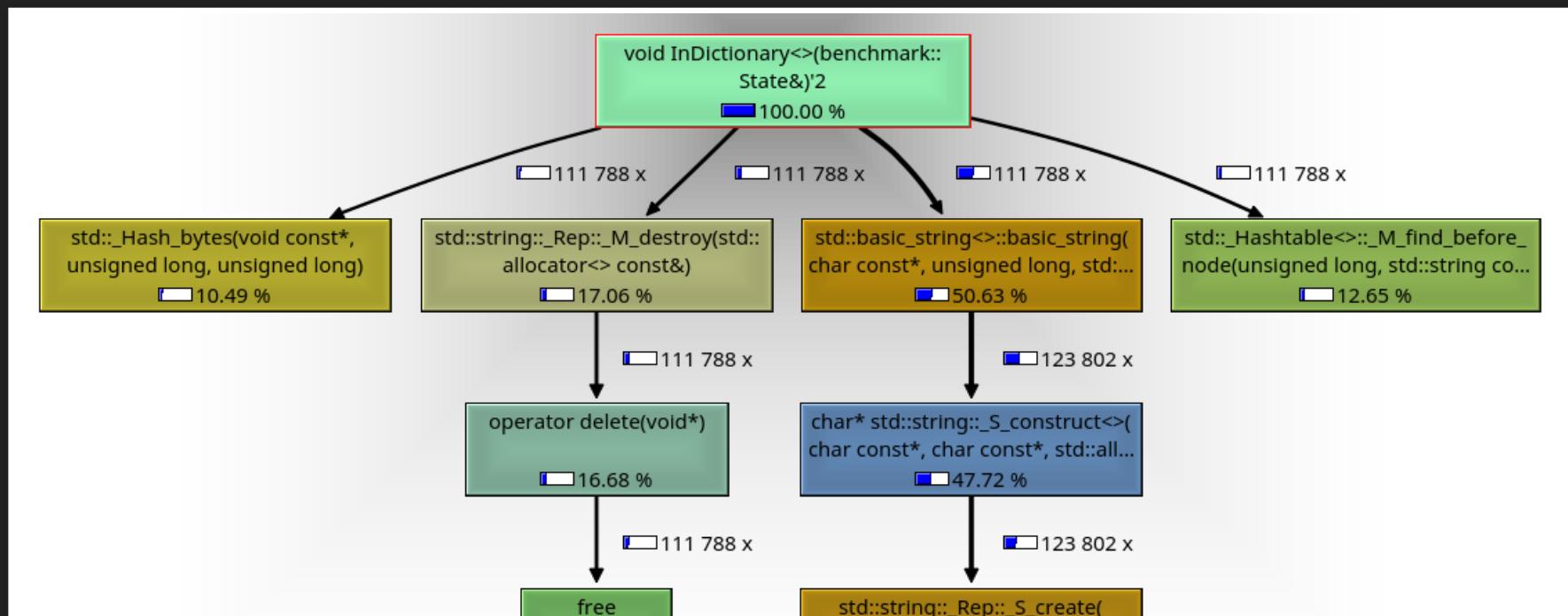
STEP 3

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_view
        return _container.find({word.data(), word.size()}) != _container.end();
    }

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

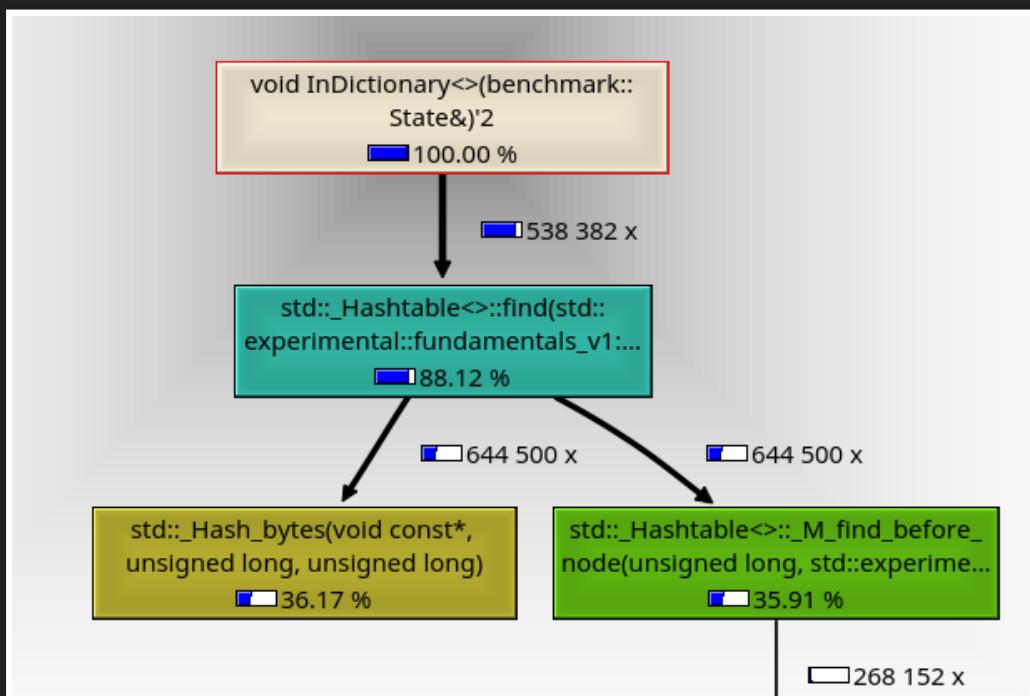
NON-ALLOCATING IMPLEMENTATION

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

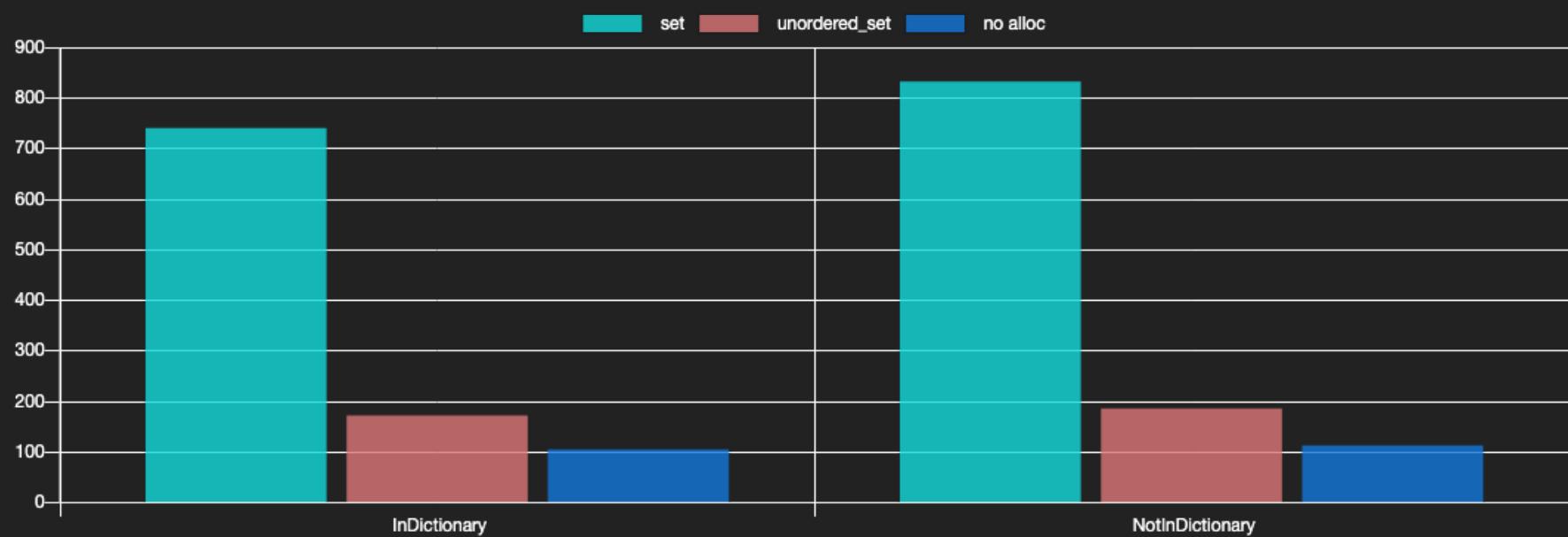
    bool in_dictionary(std::string_view word) const
    {
        return _index.find(word) != _index.end(); // No allocation!
    }

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

KCACHEGRIND OUTPUT



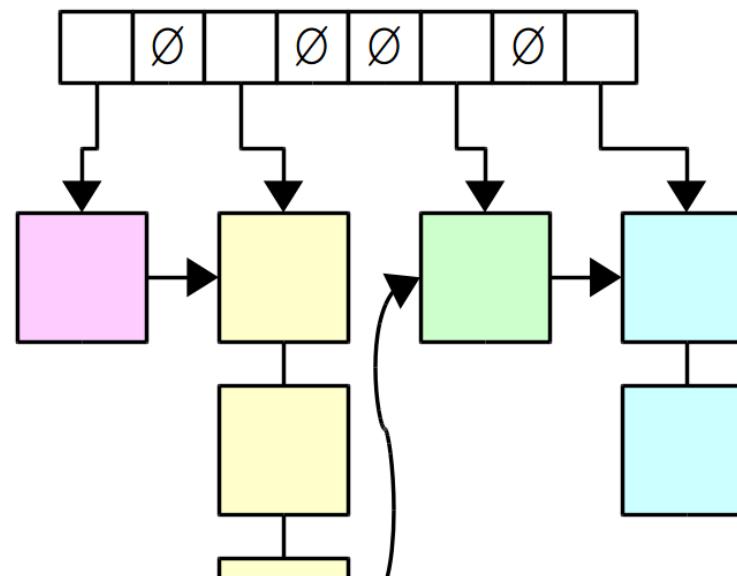
BENCHMARK: REMOVING ALLOCATIONS



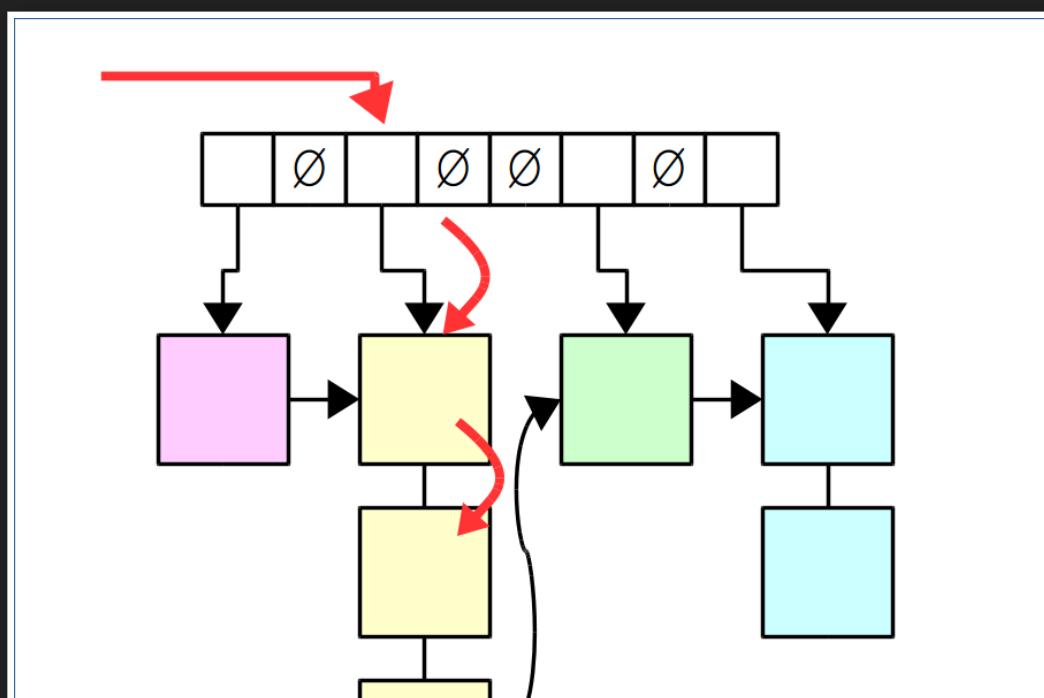
STEP 4

CACHE COHERENCE

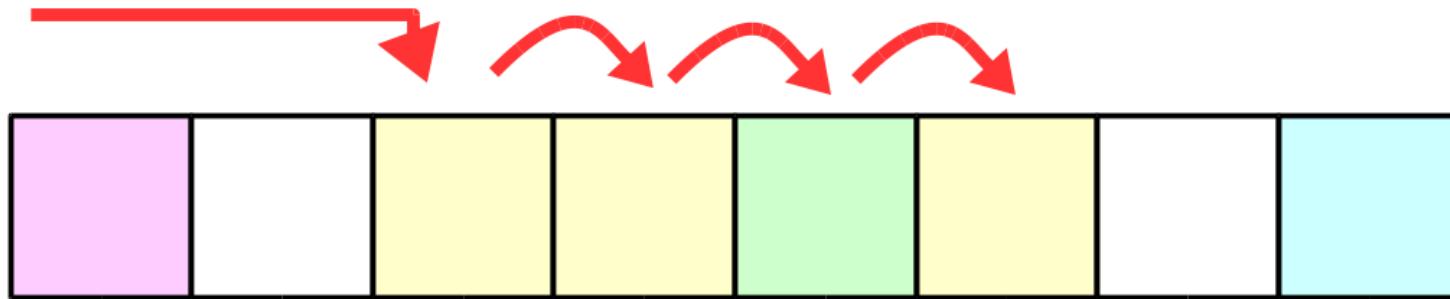
STD::UNORDERED_SET



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;

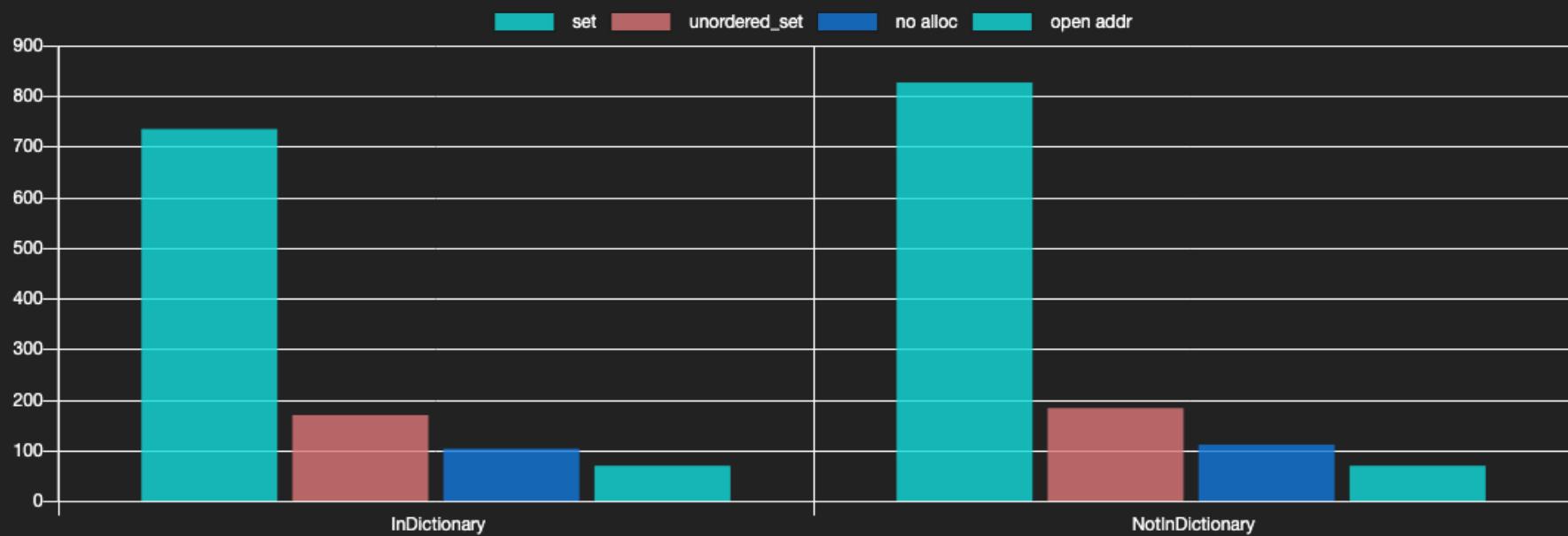
        // detect match
        if(entry.hash == hash && *entry.string == word)
            return true;

        // 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?

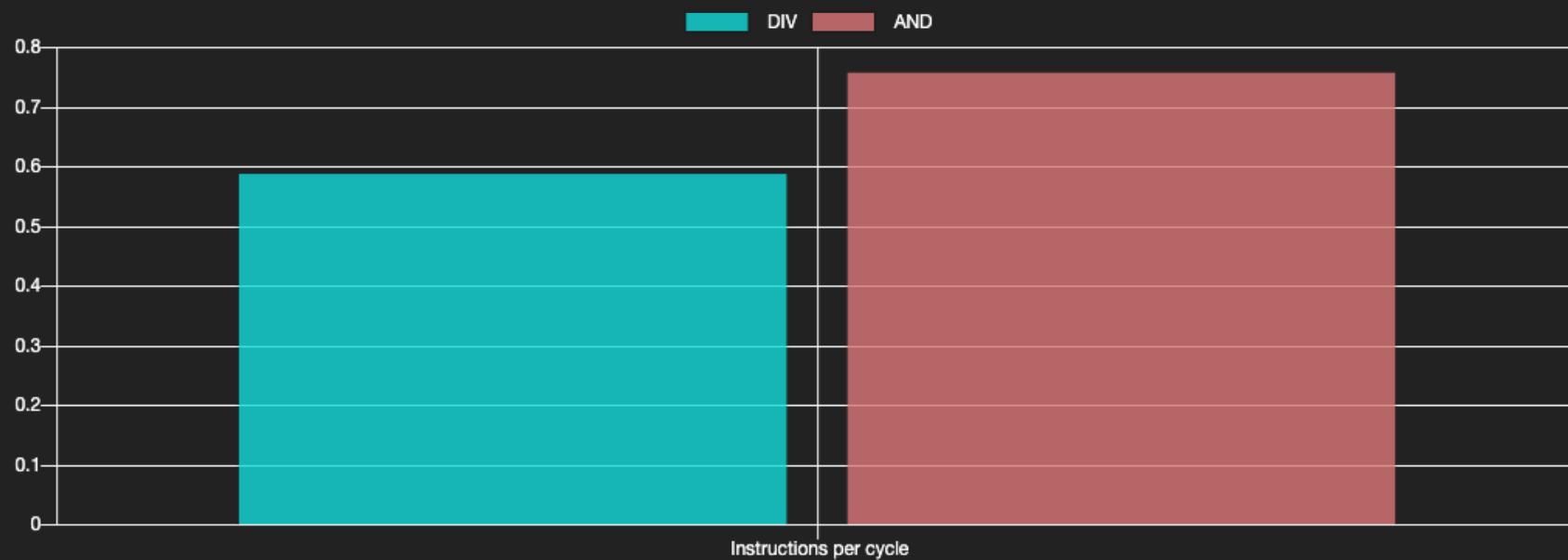
    while (true)
    {
        const Entry& entry = hashTable_[idx];

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

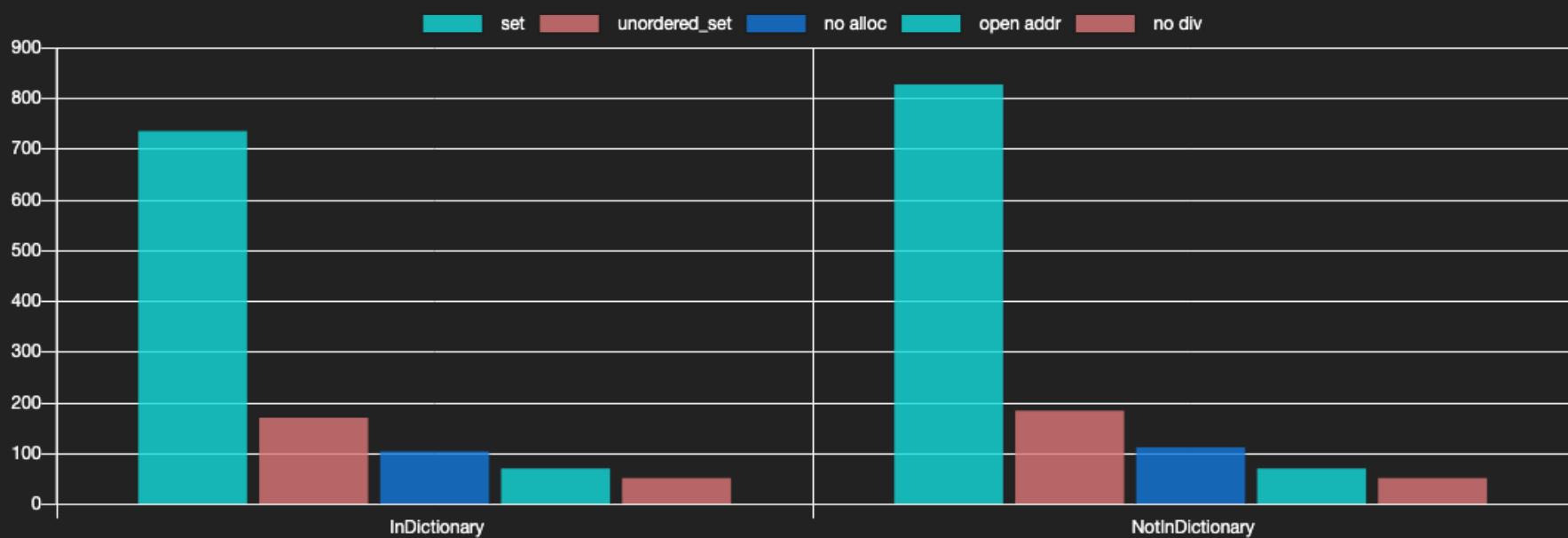
        // detect match
        if (entry.hash == hash && *entry.string == word)
            return true;

        // 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>

A night photograph of a canal scene in Amsterdam. In the foreground, a dark canal reflects the warm lights from the buildings and street lamps. A stone arch bridge spans the canal on the left. On the right bank, several traditional Dutch houses with gabled roofs and multiple stories are built close together. Some windows are lit from within, and street lamps along the sidewalk illuminate the area. A few small boats are moored along the right bank.

QUESTIONS?