



LOW LATENCY PROGRAMMING

Patrick Kostjens, Optiver

optiver 

A night view of a canal in Amsterdam. In the foreground, a stone bridge with two arches spans the canal. The water reflects the lights from the buildings and street lamps. In the background, a row of traditional Dutch buildings with gabled roofs and many windows is visible. Some windows are lit up, and the buildings are illuminated by street lamps. The sky is dark blue, and there are trees on the left and right sides of the canal.

ABOUT ME

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

A night view of a canal in Amsterdam. The scene features traditional Dutch architecture with gabled roofs and many windows, some of which are illuminated from within. A stone bridge with several arches spans the canal in the foreground. Bicycles are parked along the bridge. The water in the canal reflects the lights from the buildings and the bridge. The sky is a deep blue, suggesting dusk or early evening.

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

A night view of a canal in Amsterdam. In the foreground, a stone bridge with two arches spans the canal. Several bicycles are parked on the bridge. The canal water reflects the lights from the buildings and the bridge. In the background, a row of traditional Dutch buildings with gabled roofs and many windows is visible. The windows are lit up, and the buildings are illuminated by streetlights. The sky is dark blue, suggesting dusk or night.

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

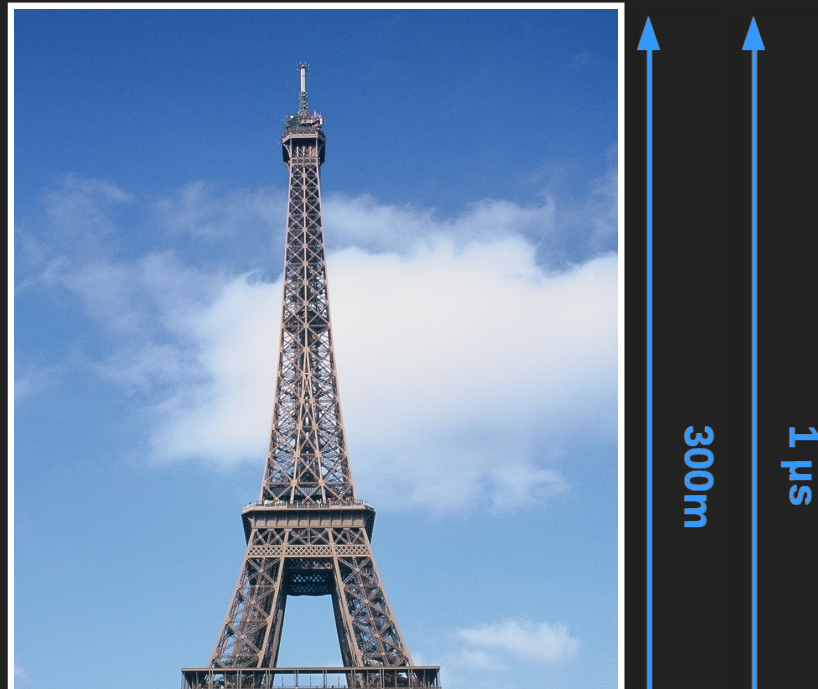
...BUT THERE ARE GOLDEN RULES

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

KNOW YOUR PROBLEM

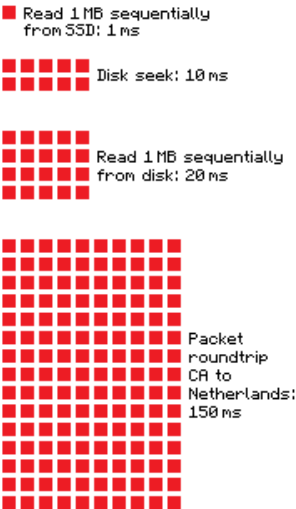
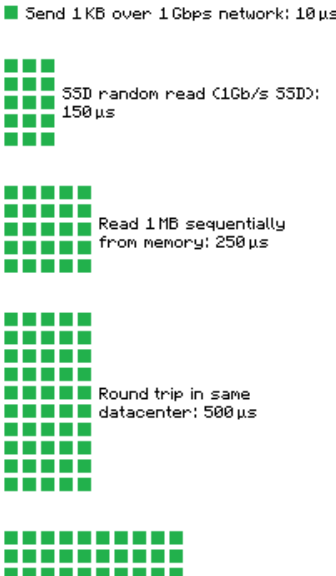
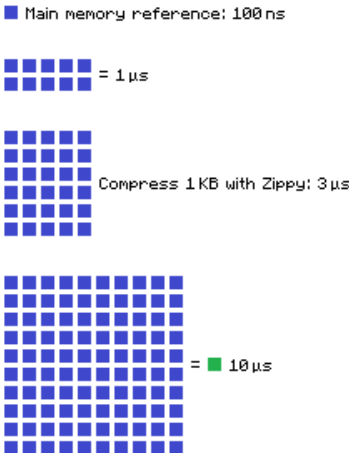
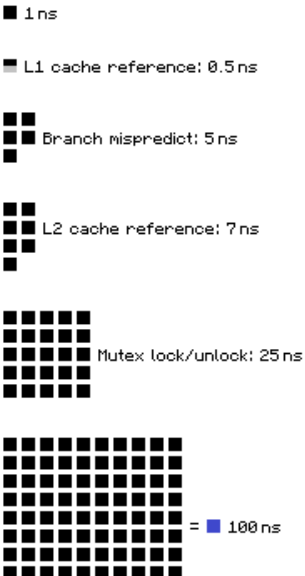
- What *exactly* needs to be fast?
- What will you gain from making it faster?

HOW FAST IS FAST?

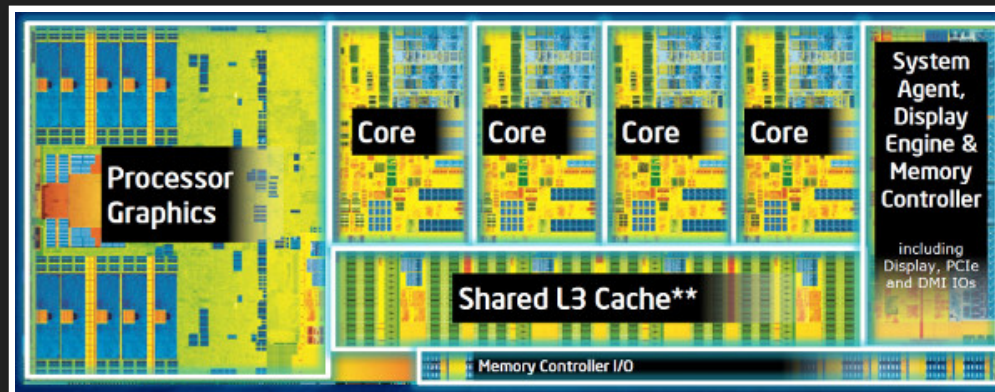


HOW FAST IS FAST?

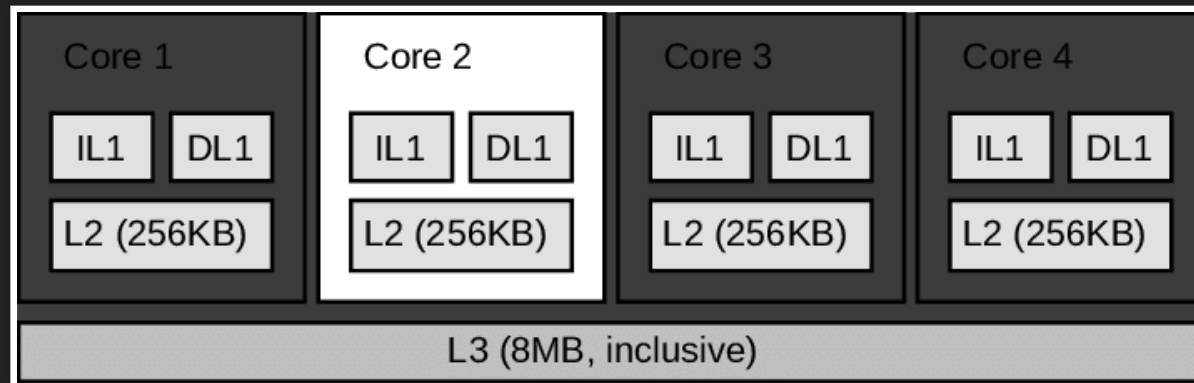
Latency Numbers Every Programmer Should Know



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 EXERCISE

```
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
```

KCACHTEGRIND

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

The screenshot shows the KCachegrind application window titled "callgrind.out.779 [./dictionary_benchmark/dictionary_benchmark] — KCachegrind". The interface includes a menu bar (File, View, Go, Settings, Help) and a toolbar with buttons for Open, Back, Forward, Up, Relative, Cycle Detection, Relative to Parent, Shorten Templates, and Instruction Fetch. The main window is divided into two main sections: a "Flat Profile" table on the left and a call graph visualization on the right.

Flat Profile Table:

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000000...	ld-2.26.so
99.87	0.00	1	_start	dictionary_benchmark
99.87	0.00	1	(below main)	libc-2.26.so: libc-start.c
99.53	0.00	1	main	dictionary_benchmark: be...
81.47	0.00	1	benchmark::...	dictionary_benchmark: be...
81.47	0.00	1	benchmark::...	dictionary_benchmark: be...
81.23	0.00	12	benchmark::...	dictionary_benchmark: be...
81.23	0.00	12	benchmark::...	dictionary_benchmark: be...
47.88	2.38	1 190 052	std::_Rb_tre...	dictionary_benchmark: stl...
44.18	21.82	1 189 980	std::_Rb_tre...	dictionary_benchmark: stl...
40.64	3.23	6	void InDicti...	dictionary_benchmark: be...
40.59	3.19	6	void NotInD...	dictionary_benchmark: be...
28.36	28.36	25 584 631	__memcmp...	libc-2.26.so: memcmp-av...
16.81	5.14	1	CreateUniqu...	dictionary_benchmark: uti...
9.38	0.83	1 424 627	operator ne...	libstdc++.so.6.0.24
9.13	0.14	1 424 627	operator del...	libstdc++.so.6.0.24
8.99	2.43	1 424 634	free	libc-2.26.so: malloc.c
8.79	0.00	12	std::_Rb_tre...	dictionary_benchmark: stl...
8.78	1.22	1 190 052	std::_Rb_tre...	dictionary_benchmark: stl...
8.77	3.37	1 424 643	malloc	libc-2.26.so: malloc.c
6.56	6.32	1 424 634	_int_free	libc-2.26.so: malloc.c
5.39	4.69	537 487	_int_malloc	libc-2.26.so: malloc.c
5.31	3.72	2 178 531	std::random...	libstdc++.so.6.0.24
4.86	3.56	1 781 181	void std::...	dictionary_benchmark: ba...

Call Graph Visualization:

The call graph shows a hierarchical view of function calls. The root node is "std::_Rb_tree<>::_M_get_insert_unique_pos" (54.35%). It calls "__memcmp_avx2_movbe" (26.85%), which in turn calls "void std::_cxx11::basic_string<>::_M_construct" (6.41%), "void std::_cxx11::basic_string<>::_M_construct" (4.60%), and "std::_Rb_tree_insert_and_rebalance" (3.51%). Other nodes include "free" (6.68%), "std::_Rb_tree_insert_and_rebalance" (7.11%), and "std::_Rb_tree_insert_and_rebalance" (4.37%).

Call Graph Table:

Incl.	Self	Distance	Calling	Callee
34.89	34.89	1-3	(3) 12 782 696	__memcmp_avx2_movbe (libc-2.26.so: memcmp-avx2-movbe.S)
54.35	26.84	2	594 990	std::_Rb_tree<>::_M_get_insert_unique_pos(std::_cxx11::basic_string<> const&) (dictionary_benchmark: stl_tree.h, ...)
6.70	6.45	3-5	(5) 591 523	_int_free (libc-2.26.so: malloc.c)
4.48	3.90	3-5	(3) 181 518	_int_malloc (libc-2.26.so: malloc.c)
3.51	3.51	1	595 026	std::_Rb_tree_insert_and_rebalance(bool, std::_Rb_tree_node_base*, std::_Rb_tree_node_base*, std::_Rb_tree_node_base&) (libs...
4.60	3.37	1	691 983	void std::_cxx11::basic_string<>::_M_construct(char*, char*, std::forward_iterator_tag) [clone isra.76] (dictionary_benchmark: b...
58.91	2.93	1	595 026	std::_Rb_tree<>::_M_get_insert_hint_unique_pos(std::_Rb_tree_const_iterator<>, std::_cxx11::basic_string<> const&) (dictionary_...
7.29	2.80	2-4	(2) 493 737	malloc (libc-2.26.so: malloc.c)
9.18	2.48	2-4	(4) 591 523	free (libc-2.26.so: malloc.c)

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;
};
```

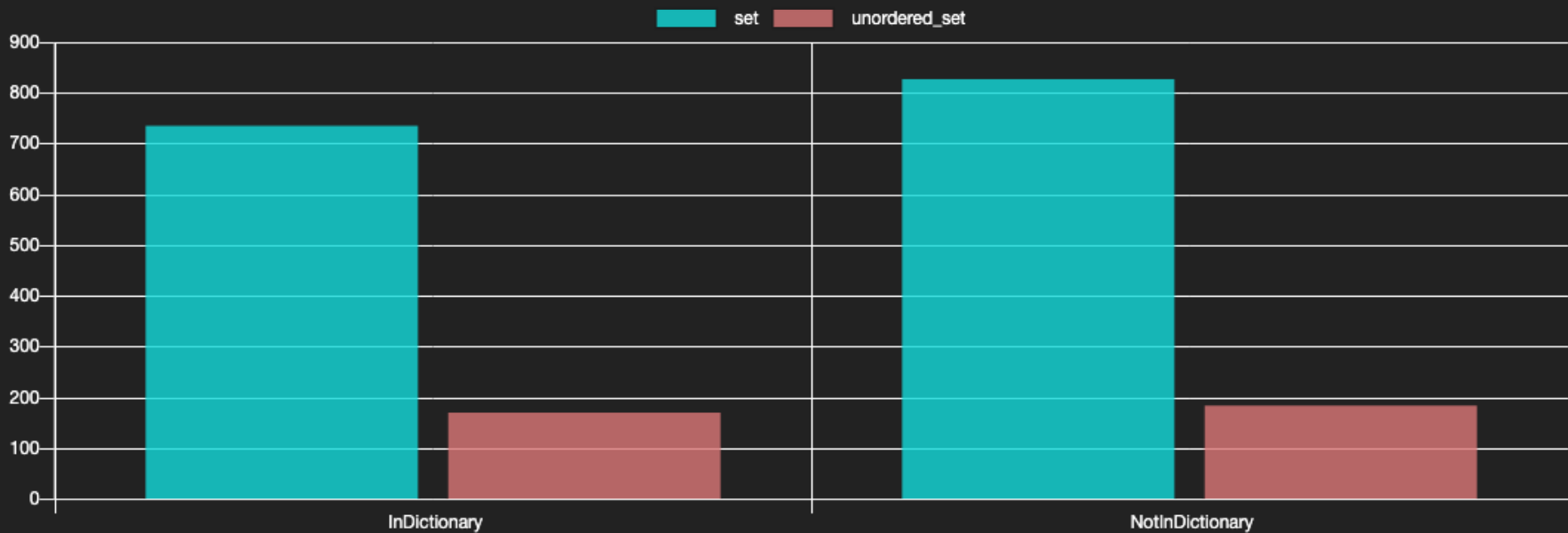
IMPROVED IMPLEMENTATION

```
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)
};
```

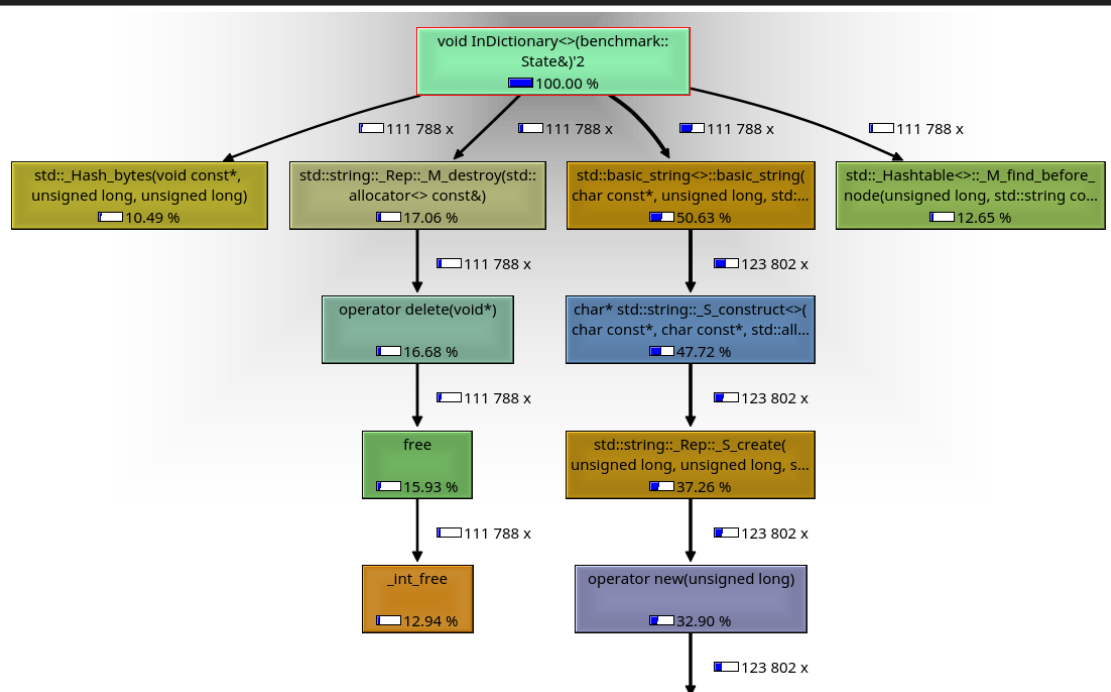
BENCHMARK: IMPROVING ALGORITHMIC COMPLEXITY



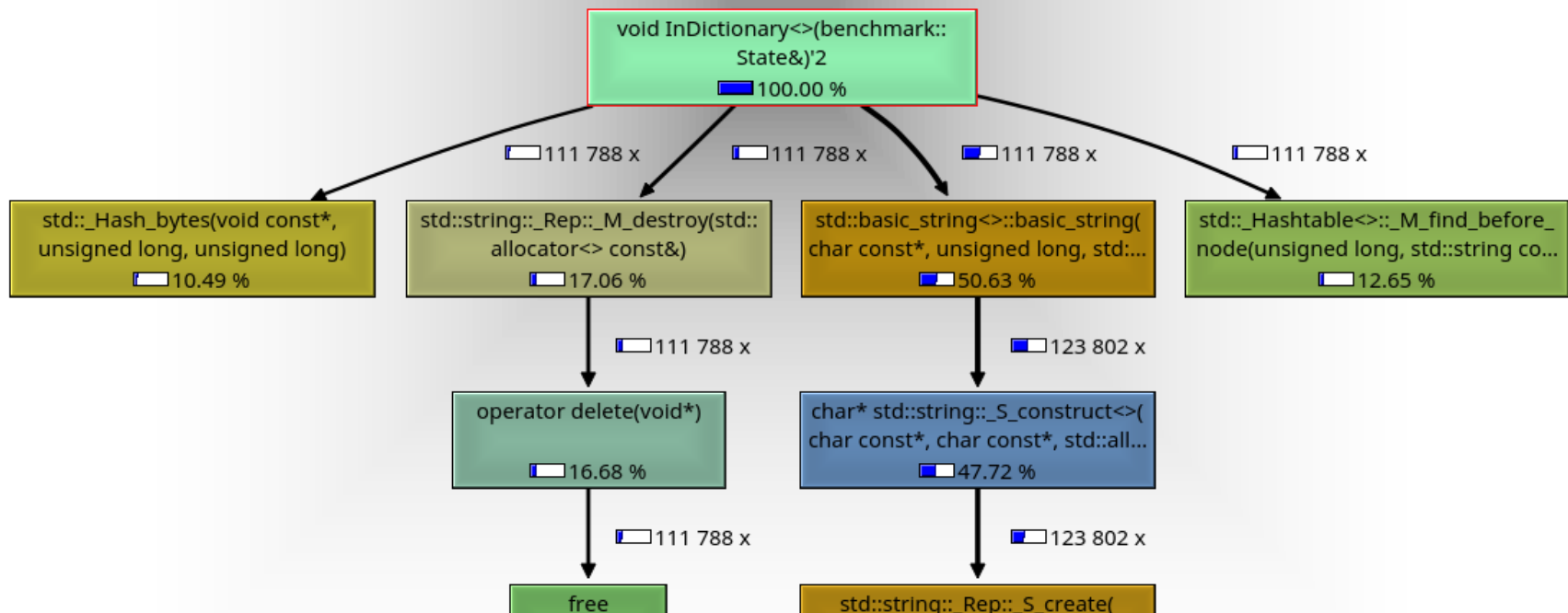
STEP 3

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_v
        return _container.find({word.data(), word.size()}) != _containe
    }

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

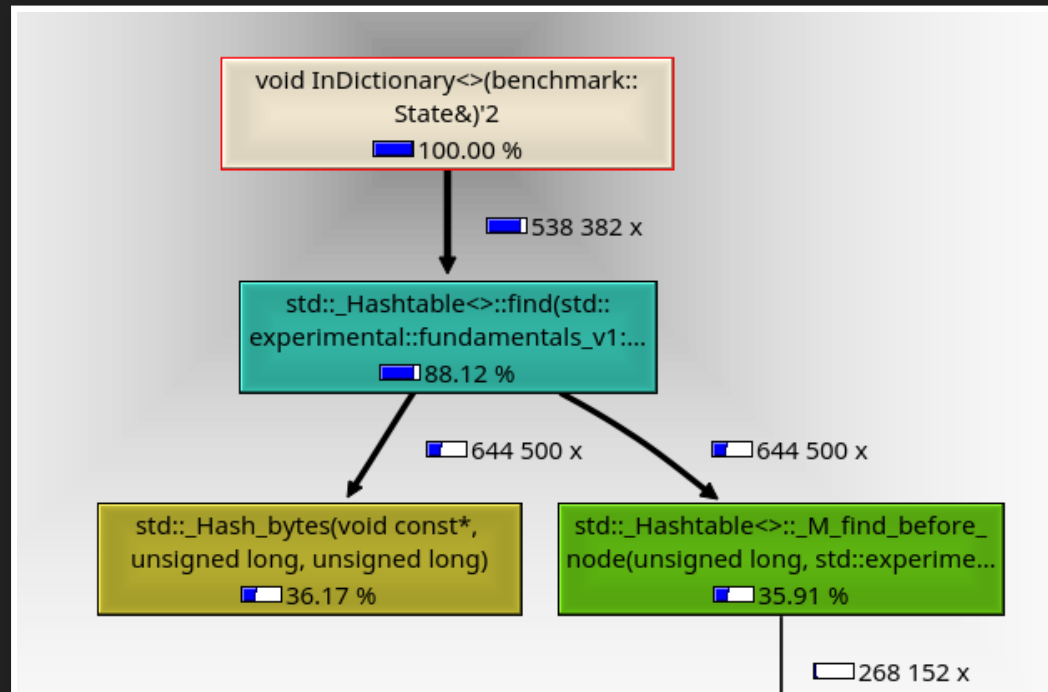
NON-ALLOCATING IMPLEMENTATION

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

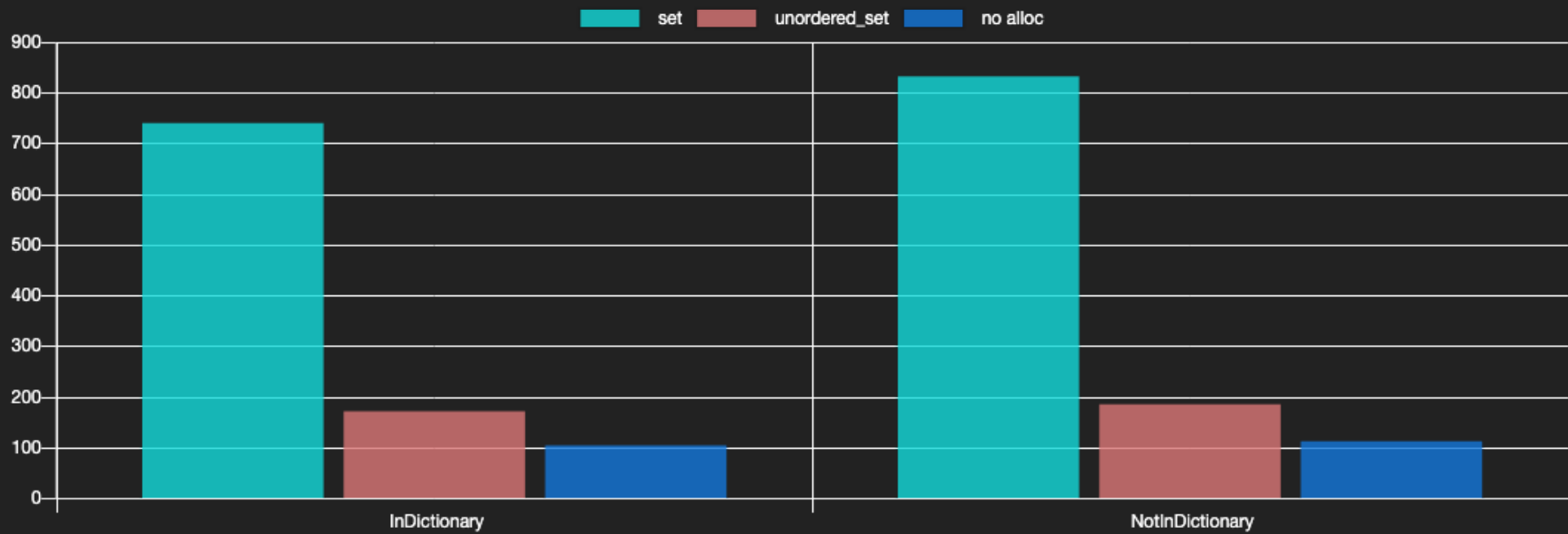
    bool in_dictionary(std::string_view word) const
    {
        return _index.find(word) != _index.end(); // No allocation!
    }

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

KCACHEGRIND OUTPUT



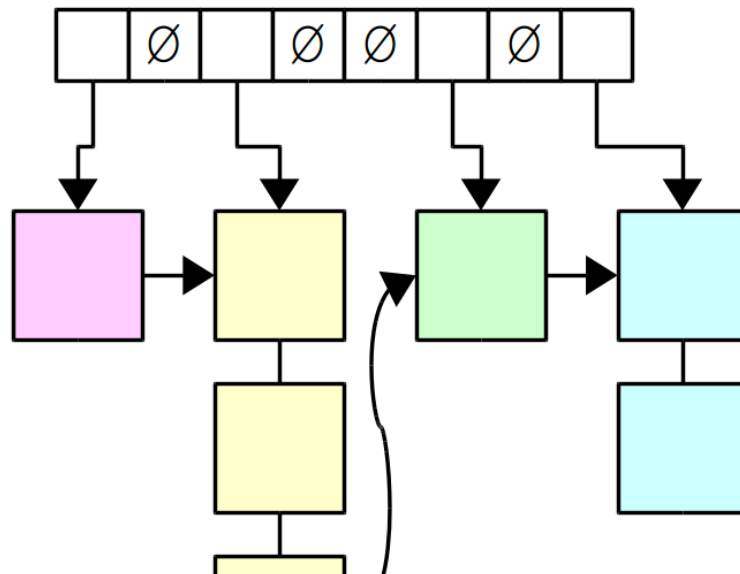
BENCHMARK: REMOVING ALLOCATIONS



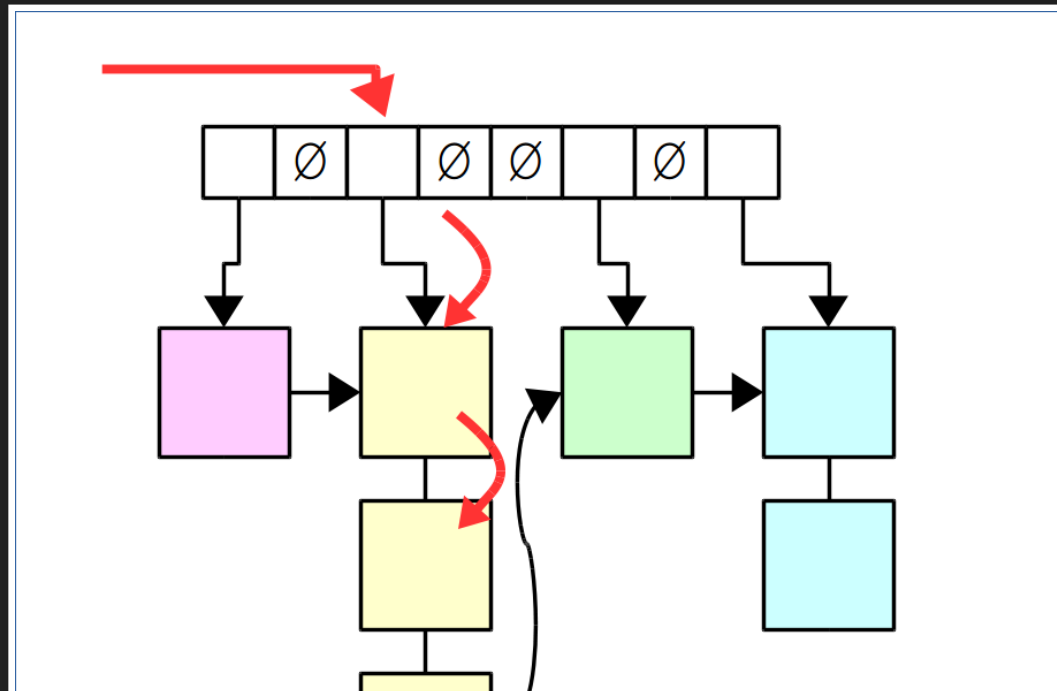
STEP 4

CACHE COHERENCE

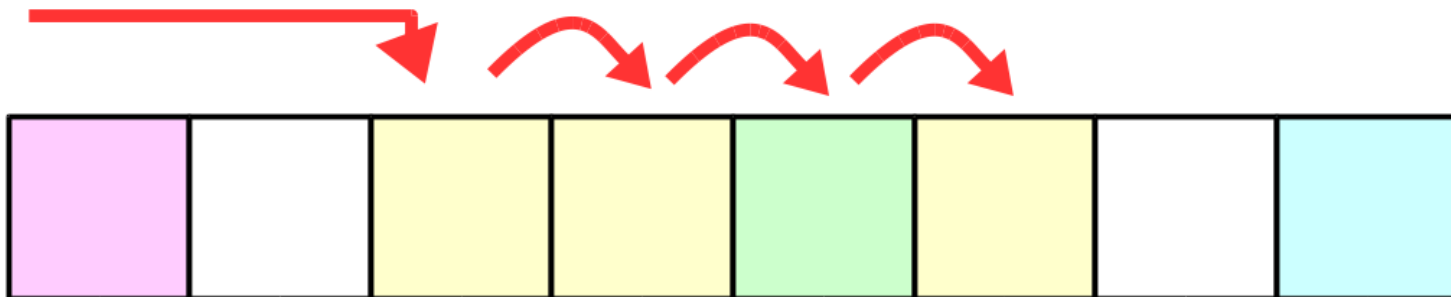
STD::UNORDERED_SET



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();

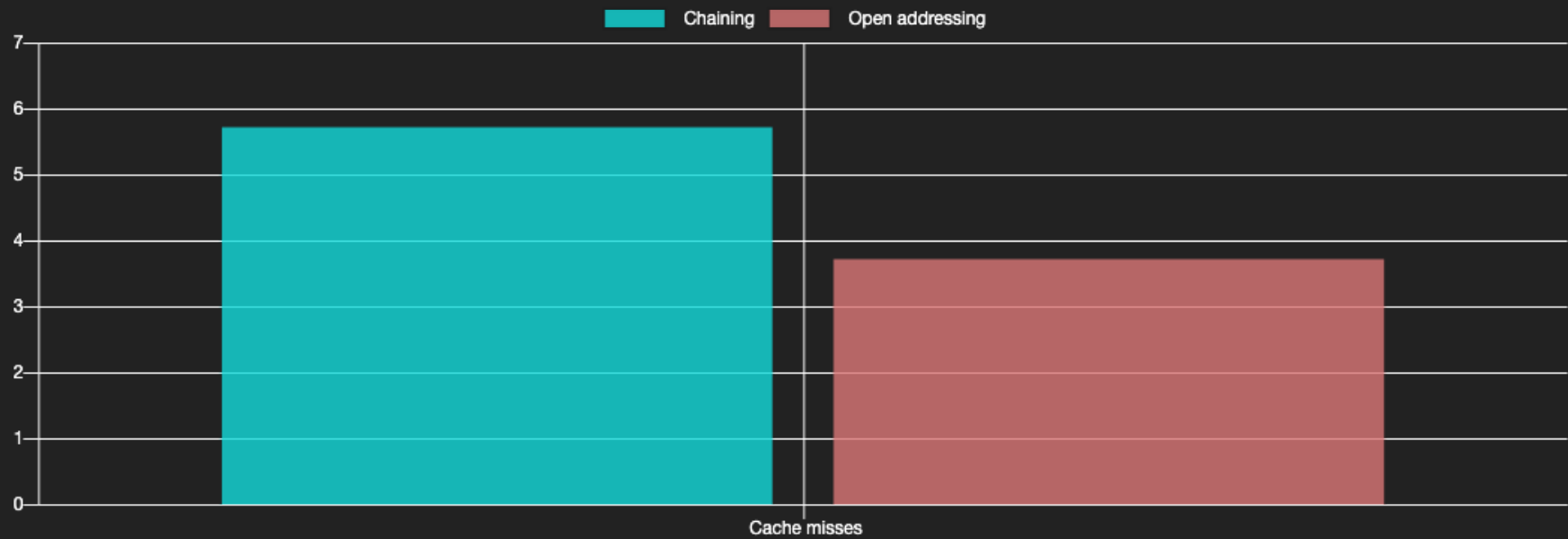
    while (true)
    {
        const Entry& entry = hashTable_[idx];

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

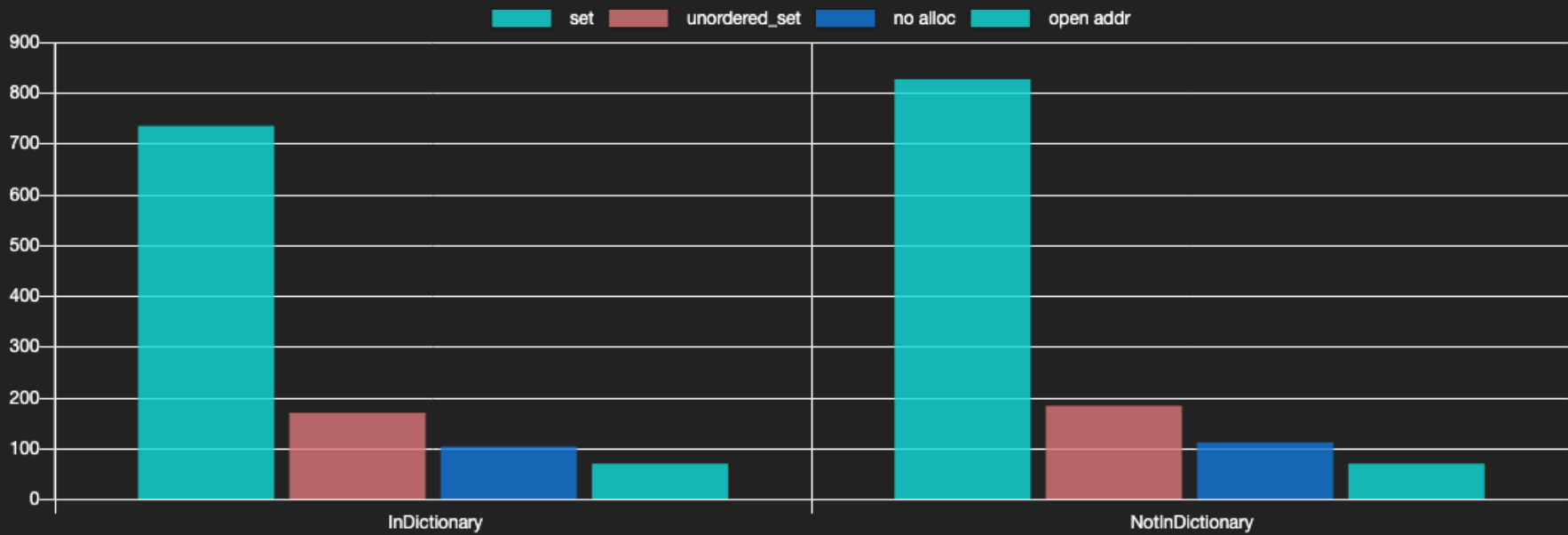
        // detect match
        if(entry.hash == hash && *entry.string == word)
            return true;

        // 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?

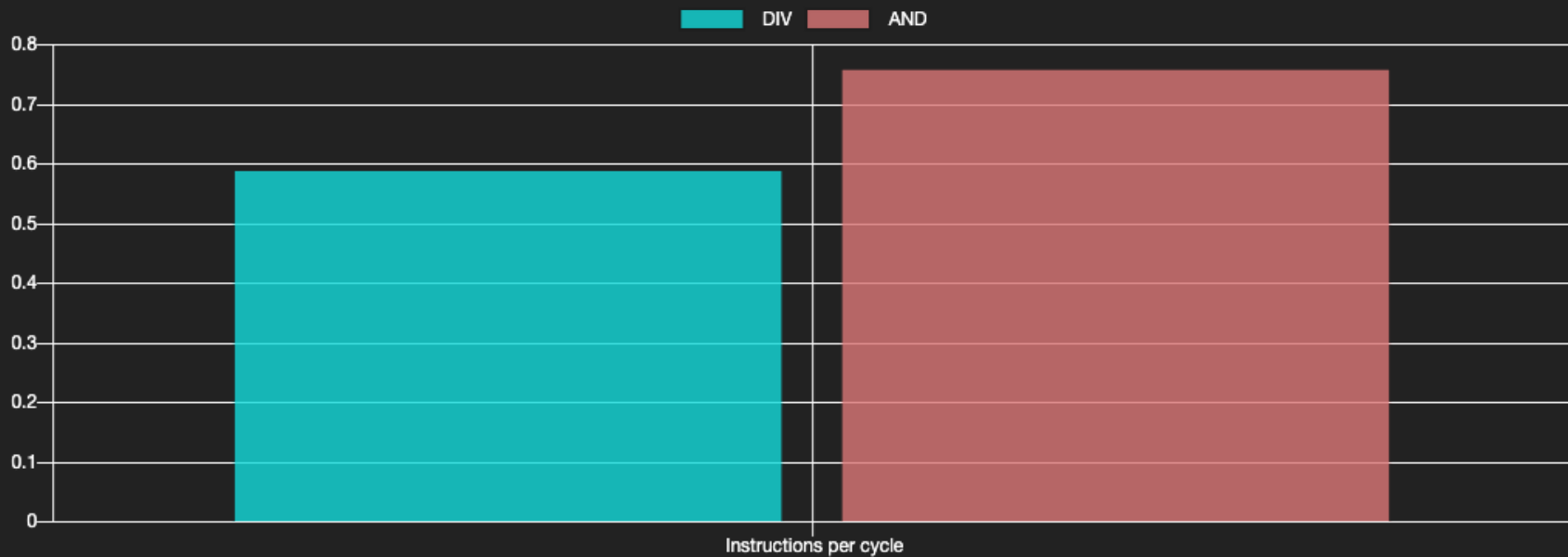
    while (true)
    {
        const Entry& entry = hashTable_[idx];

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

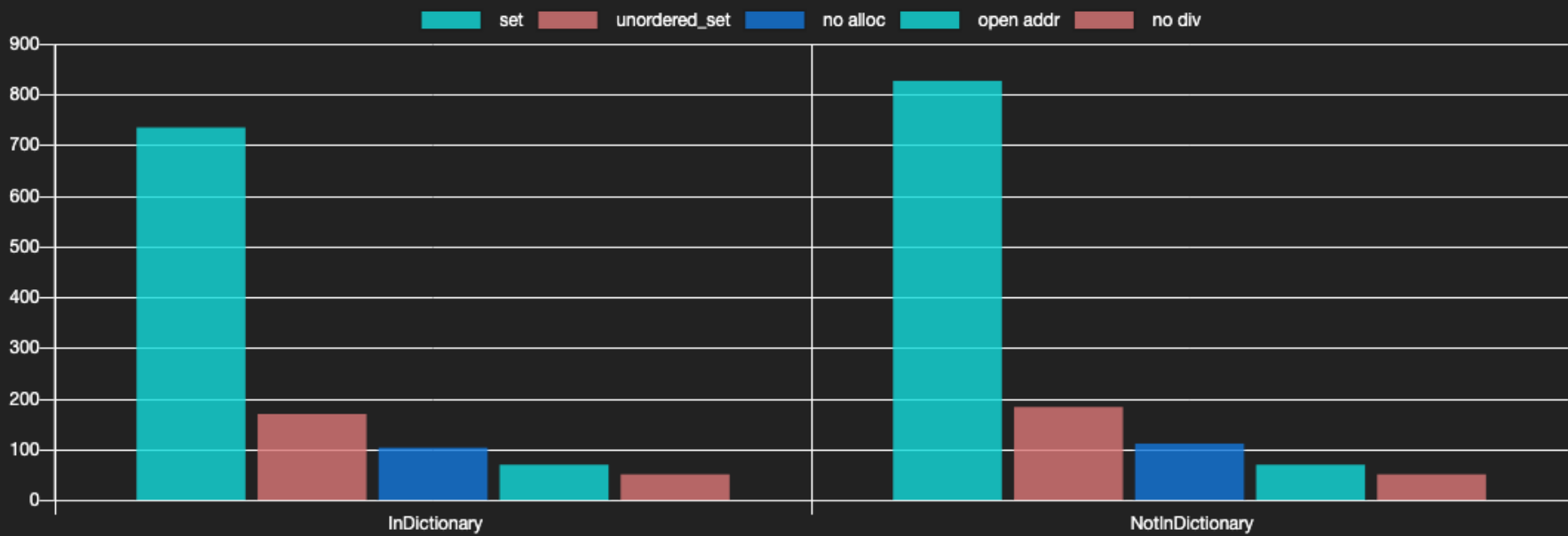
        // detect match
        if(entry.hash == hash && *entry.string == word)
            return true;

        // 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?