Google

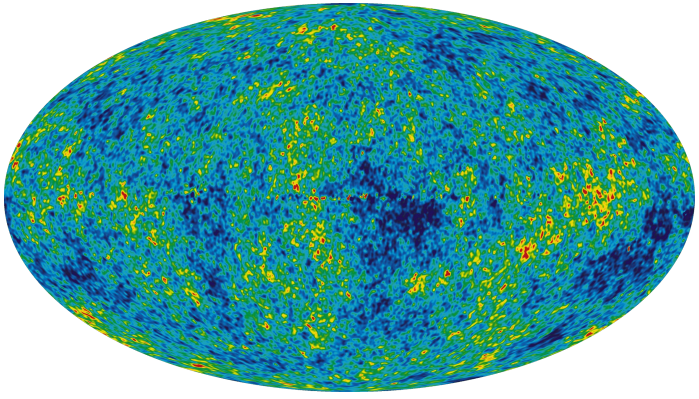
Abseil’s Open Source Hashtable

2 Years In

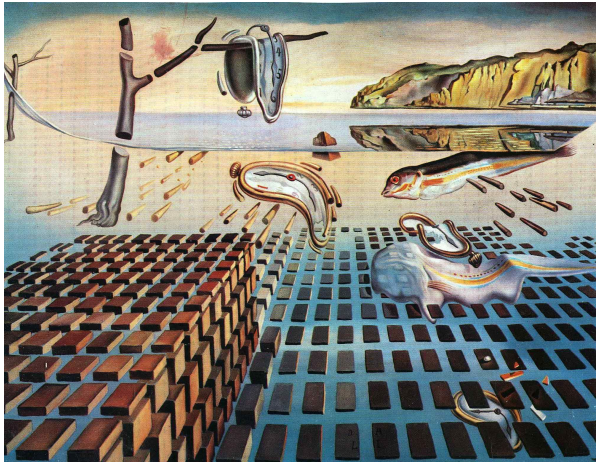
by Matt Kulukundis

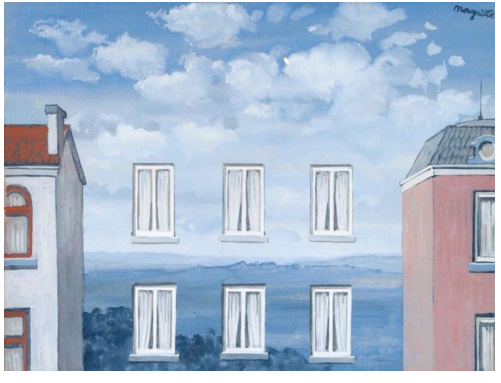
September 2019

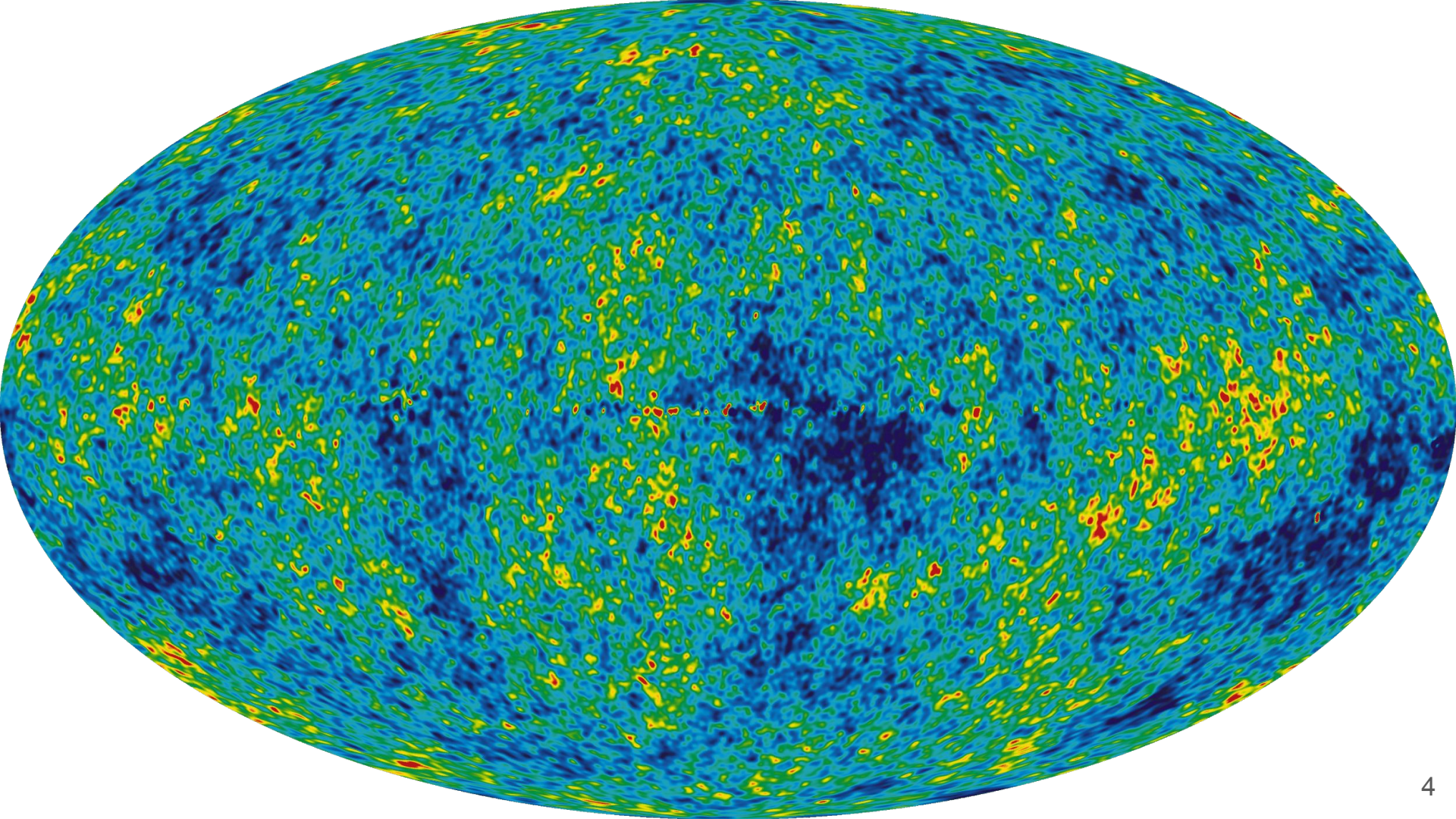
Alkis Evlogimenos
Jeff Dean
Jeffrey Lim
Matt Kulukundis
Roman Perepelitsa
Sam Benzaquen
Sanjay Ghemawat
Shaindel Schwartz
Vitaly Goldstein



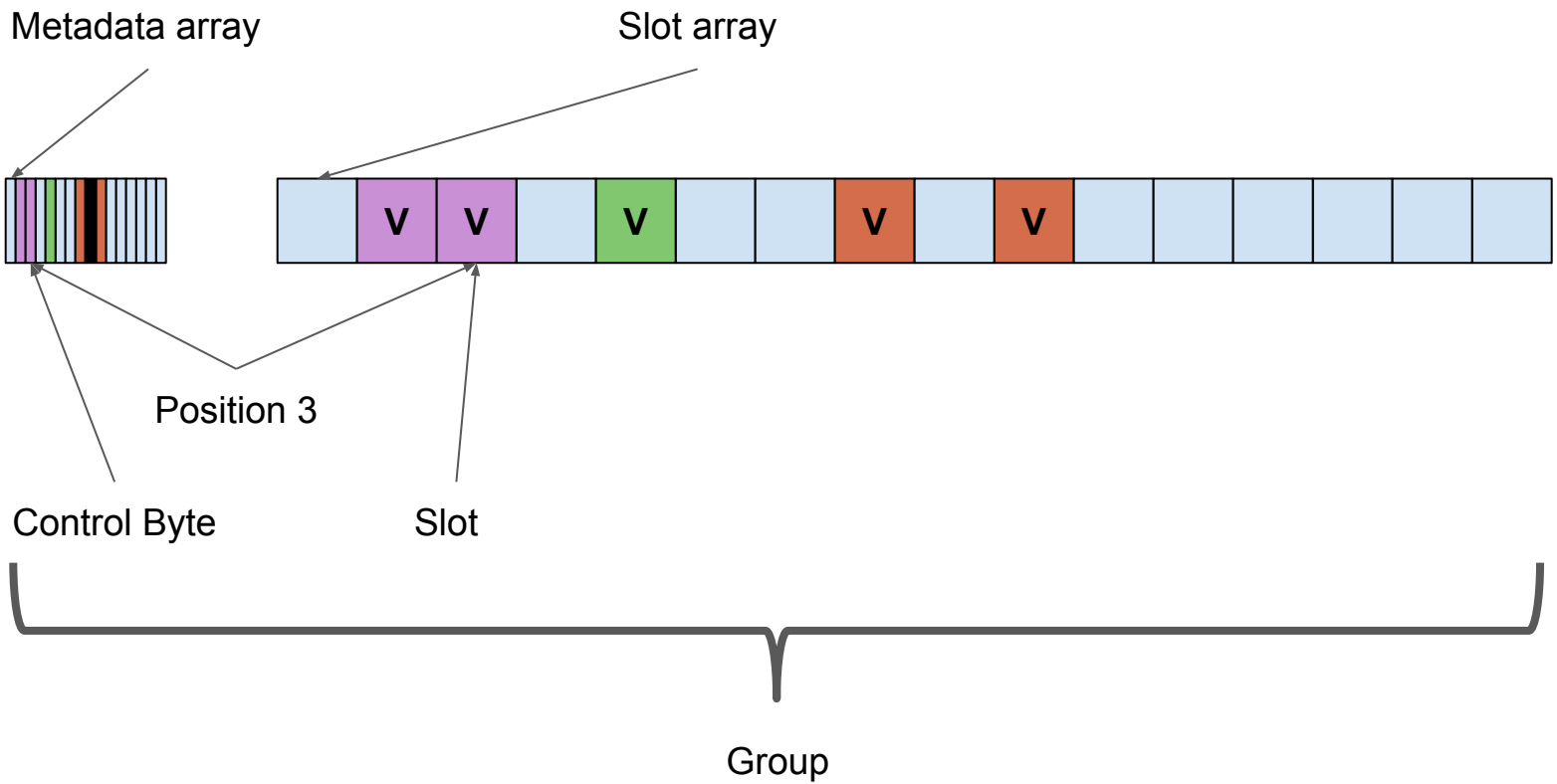

STOP



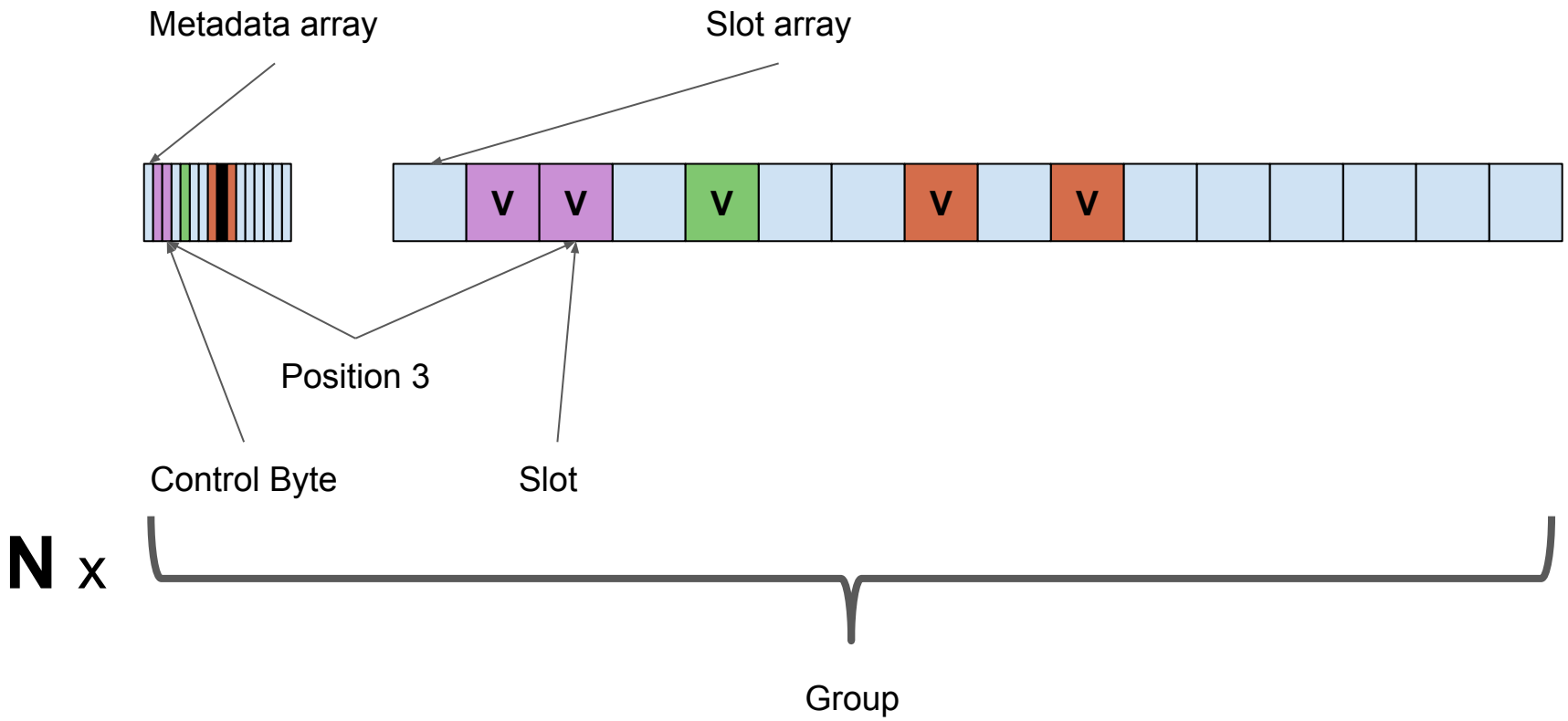


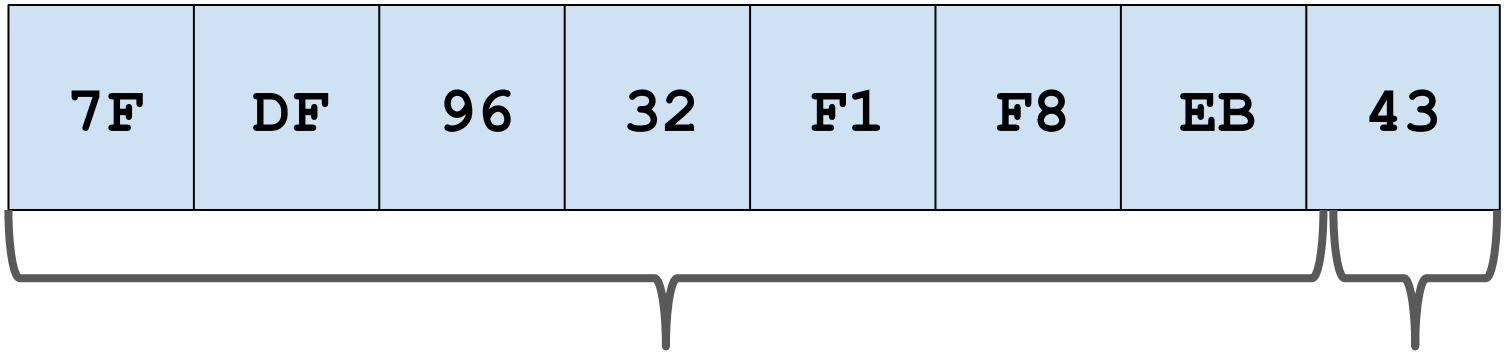


flat_hash_set



flat_hash_set





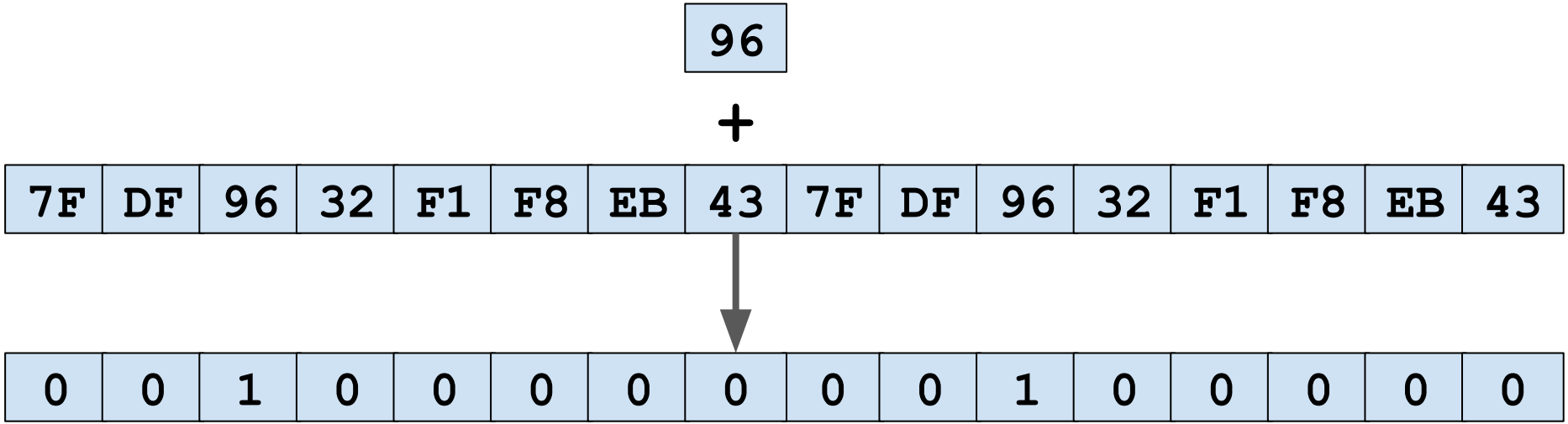
H1
57-bits
(position in array)

H2
7-bits
(metadata)

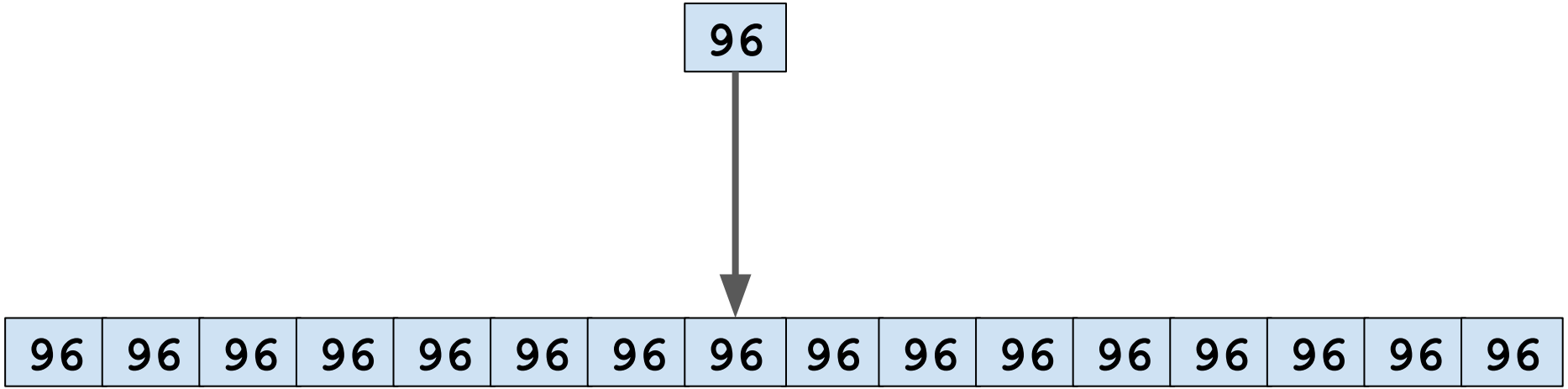
flat_hash_set


```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```

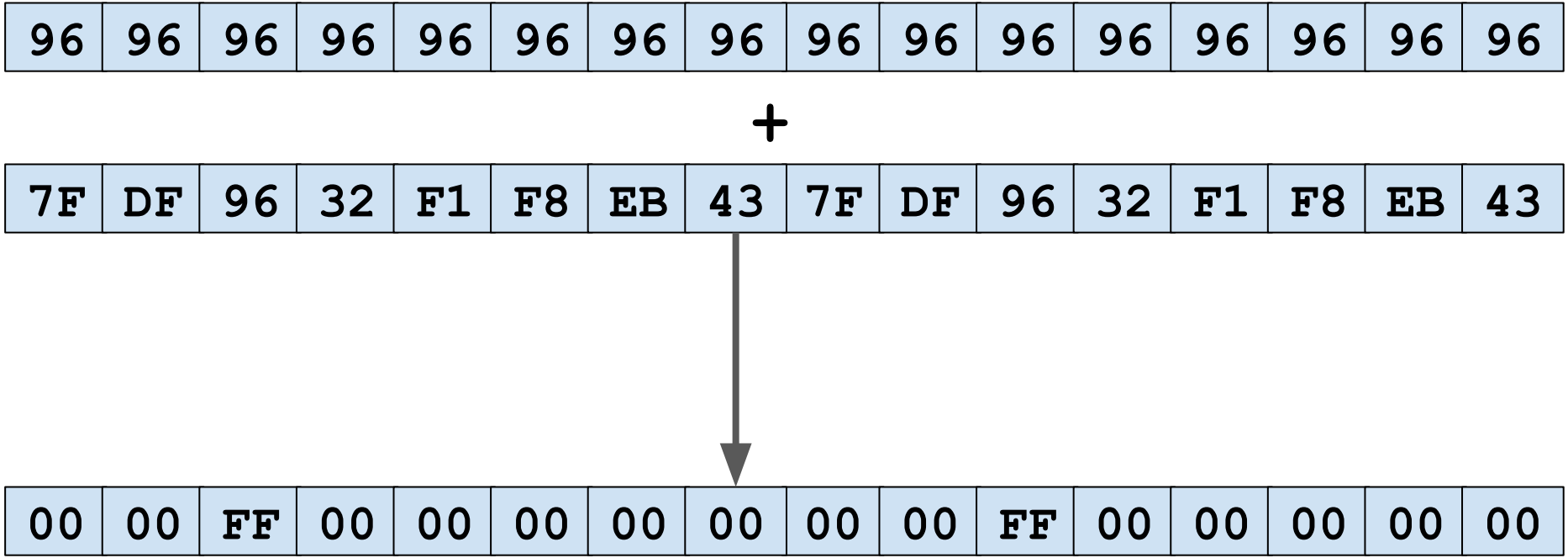
```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```



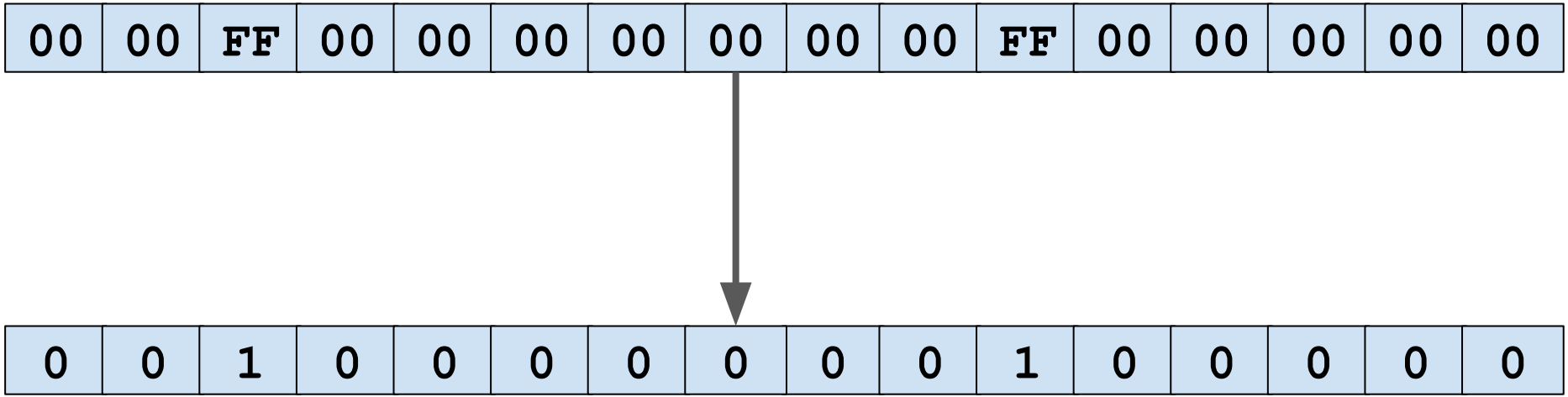
_mm_set1_epi8



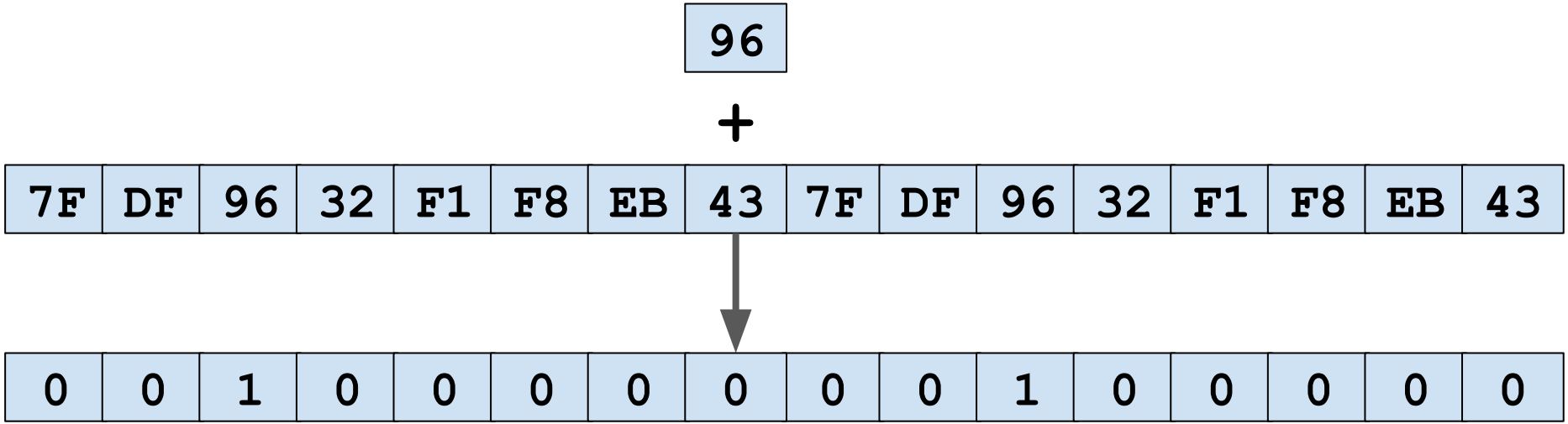
_mm_cmpeq_epi8



_mm_movemask_epi8



```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```



flat_hash_set


```
BitMask<uint32_t> Match(h2_t hash) const {
  auto match = _mm_set1_epi8(hash);
  return BitMask<uint32_t>(
      _mm_movemask_epi8(_mm_cmpeq_epi8(match, ctrl)));
}
```

flat_hash_set


```
iterator find(const K& key, size_t hash) const {

}
```

flat_hash_set


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};




    group = (group + 1) % num_groups_;
  }
}
```

flat_hash_set


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {



    }

    group = (group + 1) % num_groups_;
  }
}
```

flat_hash_set


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }

    group = (group + 1) % num_groups_;
  }
}
```

flat_hash_set

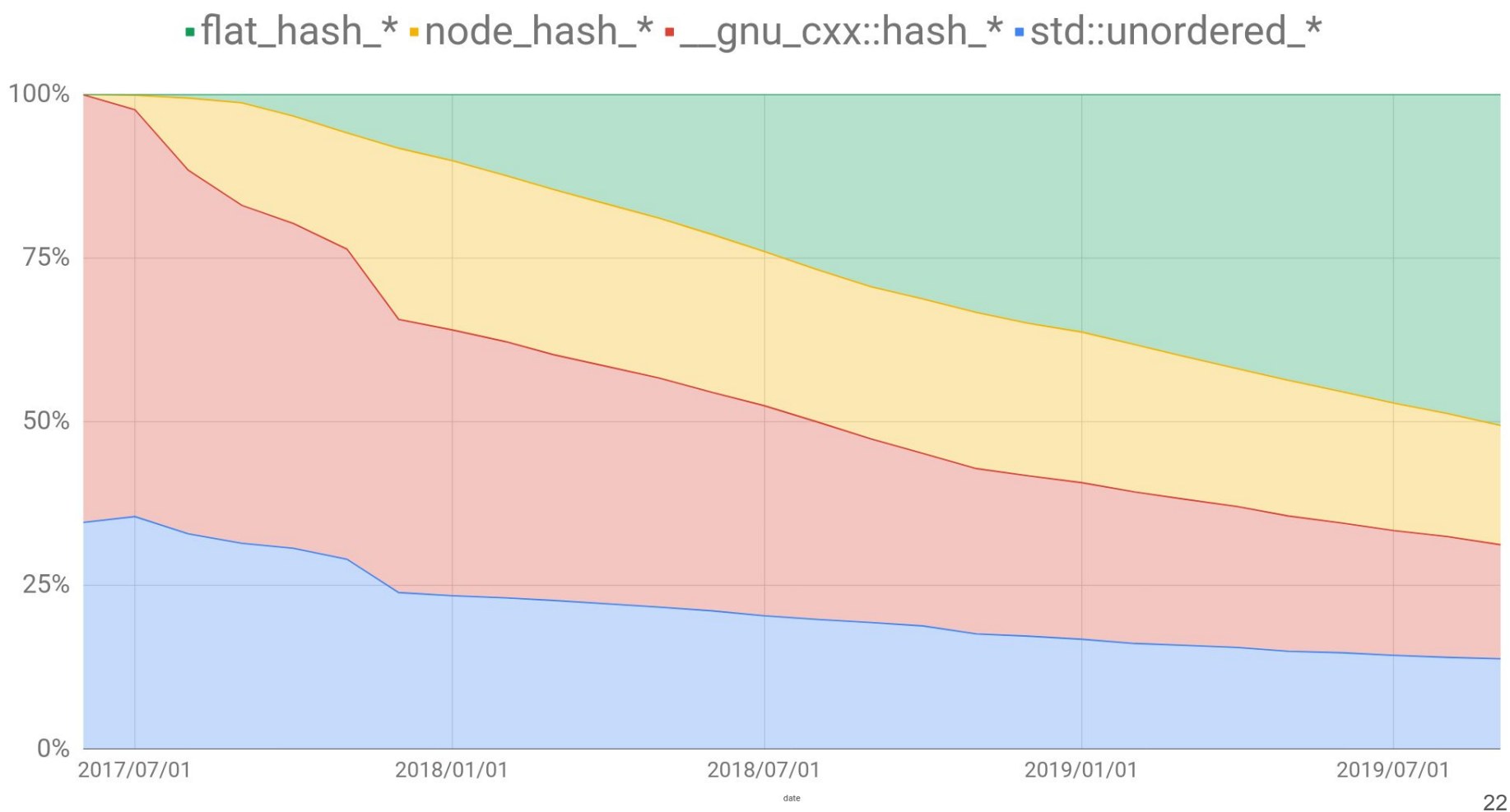

```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```

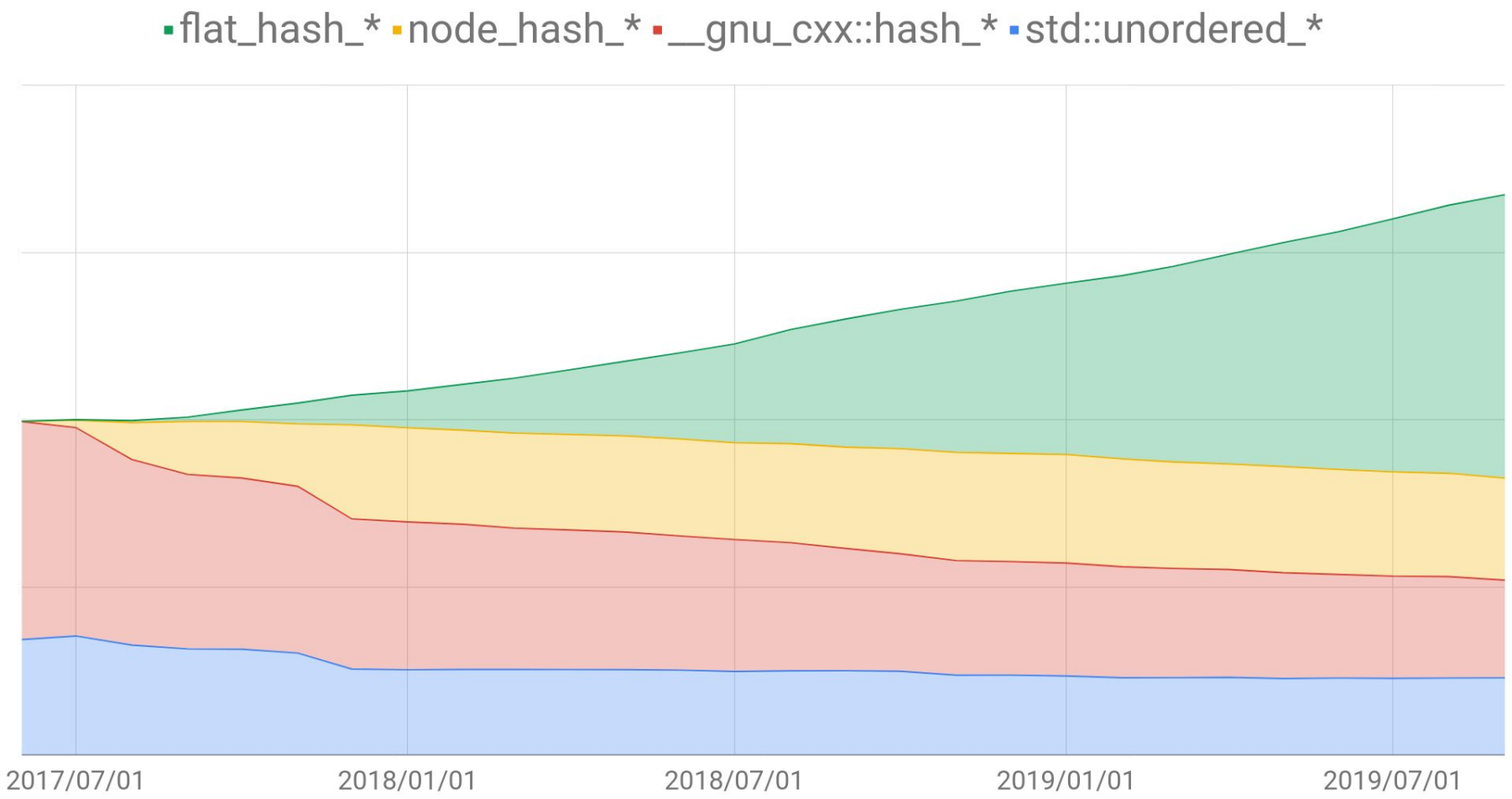
flat_hash_set


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```

flat_hash_set


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```


flat_hash_*
node_hash_*
__gnu_cxx::hash_*
std::unordered_*
100%
75%
50%
25%
0%
2017/07/01
2018/01/01
2018/07/01
2019/01/01
2019/07/01
date


flat_hash_*
node_hash_*
__gnu_cxx::hash_*
std::unordered_*
2017/07/01
2018/01/01
2018/07/01
2019/01/01
2019/07/01
date

Hyrum’s Law

With a sufficient number of users of an API,

it does not matter what you promise in the contract,

all observable behaviors of your system

will be depended on by somebody.

— Hyrum Wright

```cpp
int main() {
  std::unordered_set<int> s = {1, 2, 3};
  for (auto i : s) {
    std::cout << i << std::endl;
  }
}
```

```
TEST(Service, DedupTest) {
  Request r = PARSE_TEXT_PROTO(R"(
    id: 1
    id: 2
    id: 3
    id: 3
  )");

  EXPECT_THAT(service.Dedup(r), EqualsProto(R"(
    id: 3
    id: 1
    id: 2
  )");
}
```

```
size_t H1(size_t hash, const ctrl_t* ctrl) {
 return (hash >> 7) ^
        (reinterpret_cast<uintptr_t>(ctrl) >> 12);
}
```

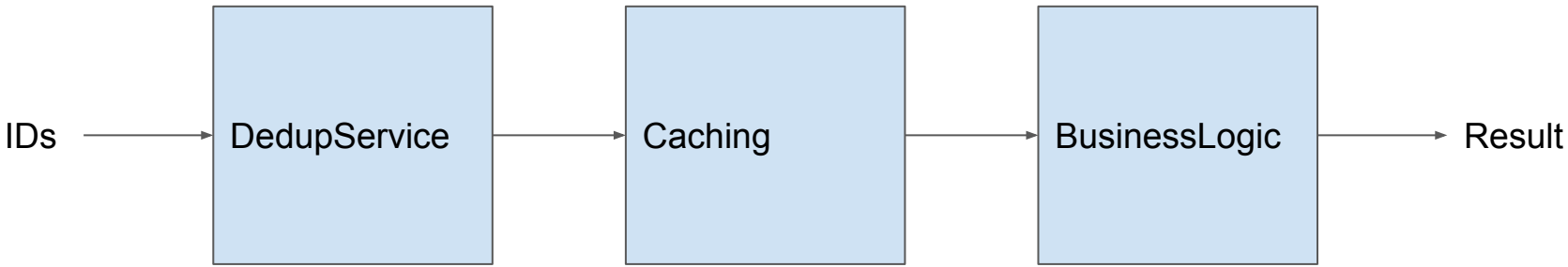
```
size_t find_first_non_full(size_t hash) {
    while (true) {
      size_t group = hash & num_groups_;
      if (MatchEmptyOrDeleted(group)) {
#if !defined(NDEBUG)
        if (ShouldInsertBackwards(hash, ctrl_)) {
          return group + mask.HighestBitSet();
        }
#endif
        return group + mask.LowestBitSet();
      }
      ++group;
    }
  }
}
```

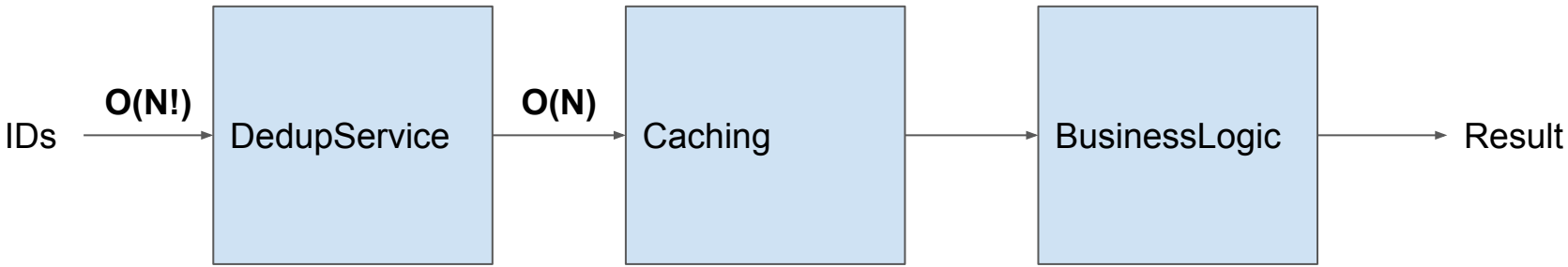
```
bool demo() {
  absl::flat_hash_set<int> i1 = { 0, 1 };
  absl::flat_hash_set<int> i2 = { 0, 1 };
  return *i1.begin() == *i2.begin();
}
```

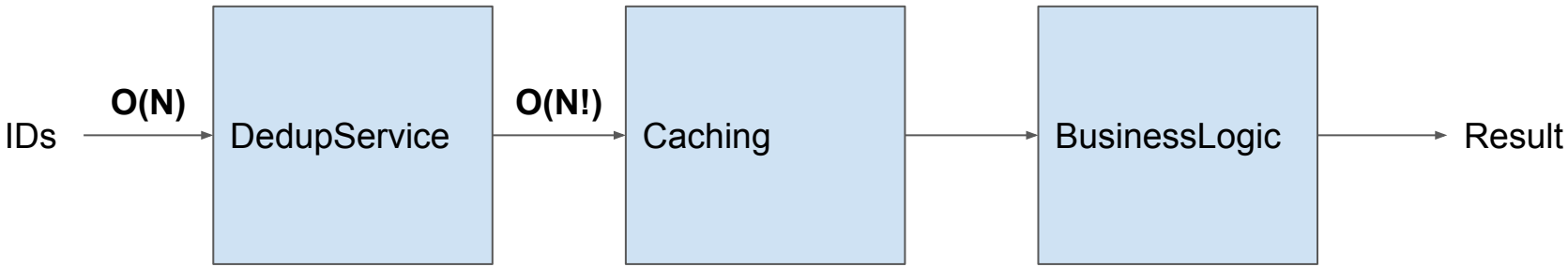
`true` - 50.3% of the time
(in debug mode)

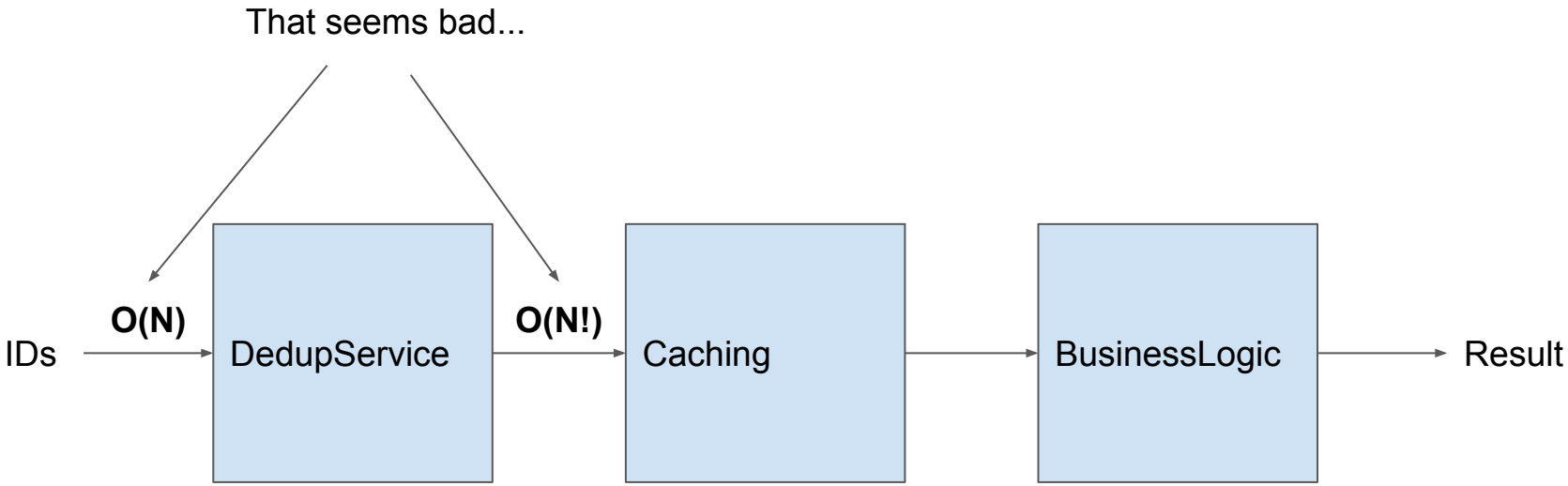
```
TEST(Service, DedupTest) {
  Request r = PARSE_TEXT_PROTO(R"(
    id: 1
    id: 2
    id: 3
    id: 3
  )");

  EXPECT_THAT(service.Dedup(r), EqualsProto(R"(
    id: 3
    id: 1
    id: 2
  )");
}
```


IDs
DedupService
BusinessLogic
Result


IDs
DedupService
Caching
BusinessLogic
Result


IDs
O(N!)
DedupService
O(N)
Caching
BusinessLogic
Result


IDs
O(N)
DedupService
O(N!)
Caching
BusinessLogic
Result

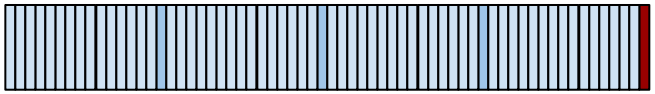

That seems bad...
IDs
O(N)
DedupService
O(N!)
Caching
BusinessLogic
Result

```
Response Dedup(Request req) {
  Response res;
  std::unordered_set<int> seen;
  for (int i : req.ids()) seen.insert(i);
  for (int i : seen) res.add_id(i);
  return res;
}
```

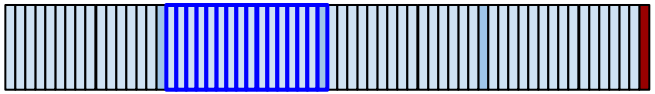
```
Response Dedup(Request req) {
  Response res;
  absl::flat_hash_set<int> seen;
  for (int i : req.ids()) {
    if (seen.insert(i).second) res.add_id(i);
  }
  return res;
}
```

```cpp
double AvgScore(absl::flat_hash_map<std::string, double> scores) {
  double total = 0.0;
  for (const auto& kv : scores) total += kv.second;
  return total / scores.size();
}
```

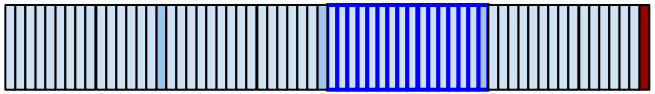

magritte



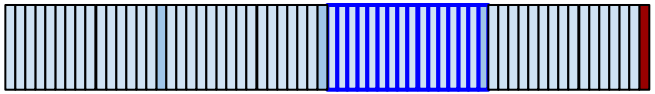
```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```



```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```

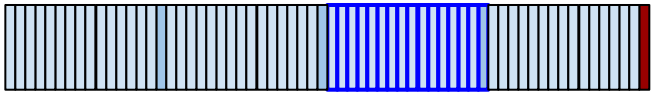


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```

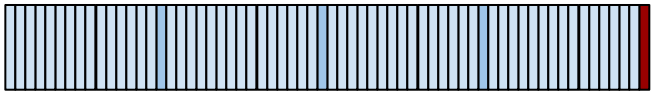


```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```

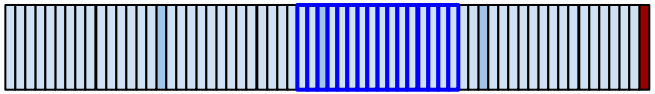

magritte



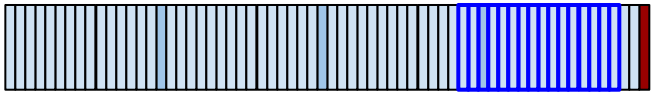
```
iterator find(const K& key, size_t hash) const {
  size_t group = H1(hash) % num_groups_;
  while (true) {
    Group g{ctrl_ + group * 16};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[group * 16 + i])
        return iterator_at(group * 16 + i);
    }
    if (g.MatchEmpty()) return end();
    group = (group + 1) % num_groups_;
  }
}
```



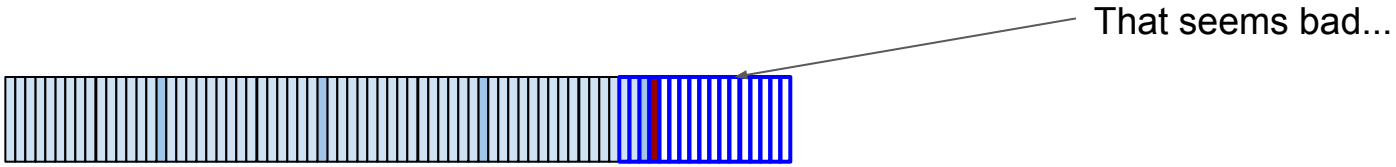
```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```



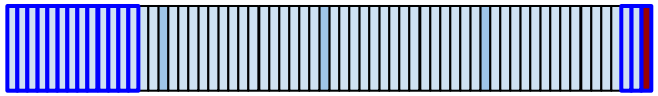
```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```



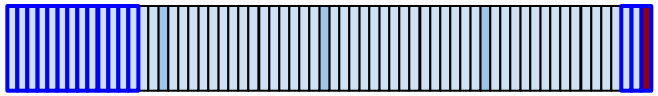
```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```



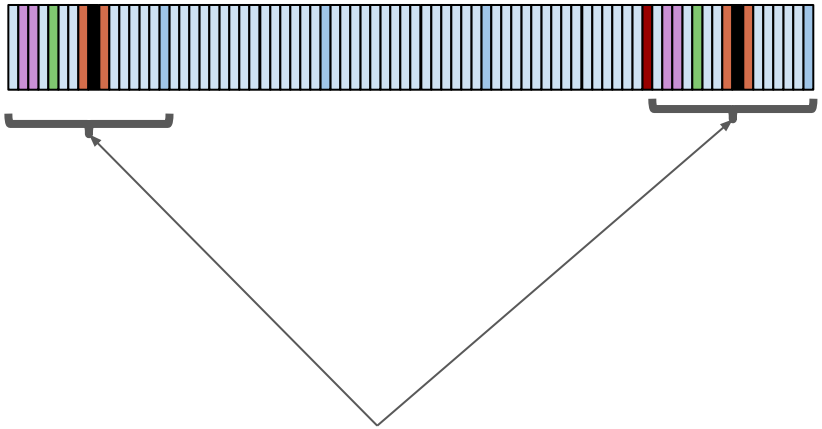

```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```



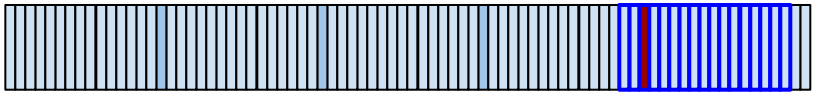
```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```



```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```



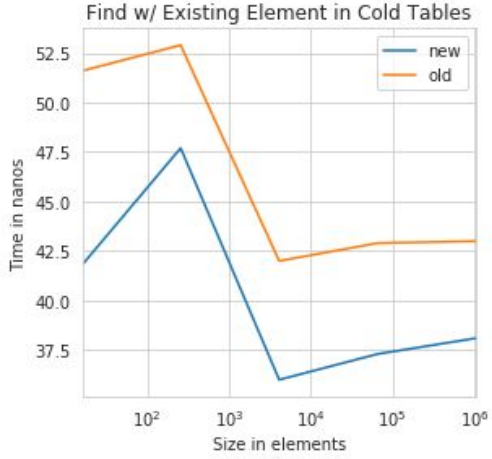

```
void set_ctrl(size_t i, ctrl_t h) {
  assert(i < capacity_);
  ctrl_[i] = h;
  if (i < 16) ctrl_[i + 1 + capacity_] = h;
}
```

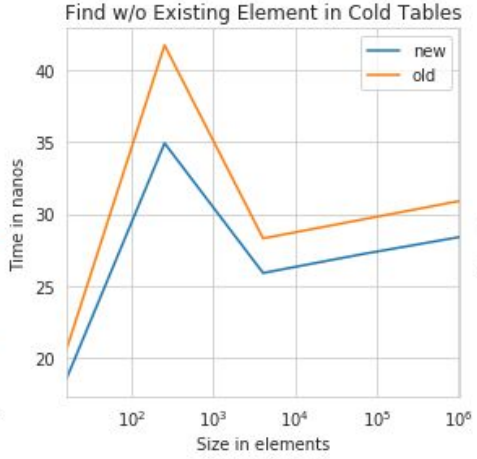


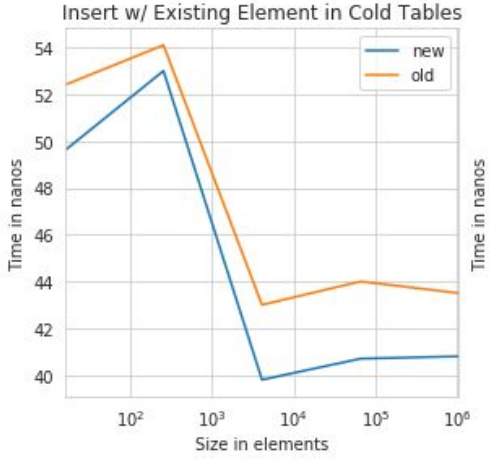
```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```

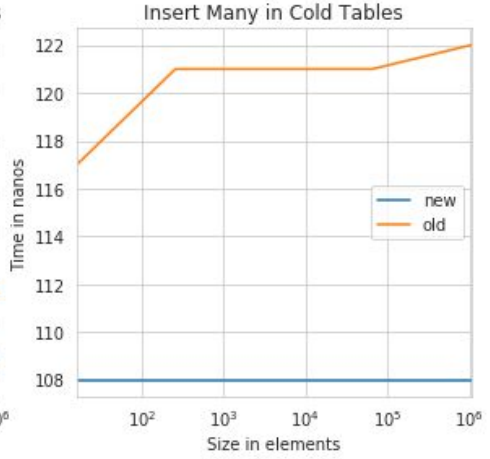

magritte

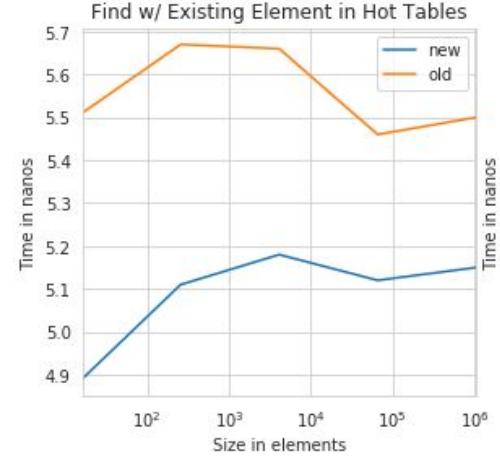
```
BM<FindHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/16              5.51ns ± 0%    4.89ns ± 0%  -11.31%       (p=0.000 n=10+10)
BM<FindHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/256             5.67ns ± 0%    5.11ns ± 0%   -9.96%       (p=0.000 n=10+8)
BM<FindHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/4k              5.66ns ± 0%    5.18ns ± 0%   -8.53%       (p=0.000 n=10+10)
BM<FindHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/64k             5.46ns ± 0%    5.12ns ± 0%   -6.07%       (p=0.000 n=10+10)
BM<FindHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/1M              5.50ns ± 0%    5.15ns ± 1%   -6.28%       (p=0.000 n=10+10)
BM<FindHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/16             51.6ns ± 3%    41.8ns ± 3%  -19.00%       (p=0.000 n=10+10)
BM<FindHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/256            52.9ns ± 2%    47.7ns ± 3%   -9.92%       (p=0.000 n=9+10)
BM<FindHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/4k             42.0ns ± 4%    36.0ns ± 2%  -14.31%       (p=0.000 n=10+10)
BM<FindHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/64k            42.9ns ± 2%    37.3ns ± 2%  -13.04%       (p=0.000 n=10+10)
BM<FindHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/1M             43.0ns ± 1%    38.1ns ± 2%  -11.53%       (p=0.000 n=10+10)
BM<FindMiss_Hot, gtl::flat_hash_set, 64, Density::kMax>/16             3.12ns ± 0%    2.93ns ± 0%   -6.09%       (p=0.000 n=10+8)
BM<FindMiss_Hot, gtl::flat_hash_set, 64, Density::kMax>/256            4.51ns ± 0%    4.24ns ± 1%   -5.92%       (p=0.000 n=8+10)
BM<FindMiss_Hot, gtl::flat_hash_set, 64, Density::kMax>/4k             4.84ns ± 2%    4.65ns ± 1%   -3.76%       (p=0.000 n=10+10)
BM<FindMiss_Hot, gtl::flat_hash_set, 64, Density::kMax>/64k            4.89ns ± 0%    4.68ns ± 0%   -4.27%       (p=0.000 n=8+9)
BM<FindMiss_Hot, gtl::flat_hash_set, 64, Density::kMax>/1M             4.88ns ± 0%    4.73ns ± 0%   -3.09%       (p=0.000 n=10+10)
BM<FindMiss_Cold, gtl::flat_hash_set, 64, Density::kMax>/16            20.5ns ± 1%    18.5ns ± 2%   -9.81%       (p=0.000 n=9+9)
BM<FindMiss_Cold, gtl::flat_hash_set, 64, Density::kMax>/256           41.7ns ± 5%    34.9ns ± 2%  -16.24%       (p=0.000 n=10+10)
BM<FindMiss_Cold, gtl::flat_hash_set, 64, Density::kMax>/4k            28.3ns ± 4%    25.9ns ± 2%   -8.54%       (p=0.000 n=10+10)
BM<FindMiss_Cold, gtl::flat_hash_set, 64, Density::kMax>/64k           29.6ns ± 4%    27.2ns ± 2%   -8.13%       (p=0.000 n=10+10)
BM<FindMiss_Cold, gtl::flat_hash_set, 64, Density::kMax>/1M            30.9ns ± 1%    28.4ns ± 2%   -8.31%       (p=0.000 n=10+10)
BM<InsertHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/16            6.16ns ± 0%    5.35ns ± 0%  -13.15%       (p=0.000 n=8+8)
BM<InsertHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/256           6.31ns ± 0%    5.55ns ± 0%  -11.97%       (p=0.000 n=10+10)
BM<InsertHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/4k            6.31ns ± 0%    5.63ns ± 0%  -10.73%       (p=0.000 n=10+8)
BM<InsertHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/64k           6.32ns ± 0%    5.51ns ± 0%  -12.87%       (p=0.000 n=10+8)
BM<InsertHit_Hot, gtl::flat_hash_set, 64, Density::kMax>/1M            6.37ns ± 0%    5.55ns ± 0%  -12.87%       (p=0.000 n=7+6)
BM<InsertHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/16           52.4ns ± 3%    49.6ns ± 5%   -5.37%       (p=0.000 n=10+10)
BM<InsertHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/256          54.1ns ± 3%    53.0ns ± 1%   -2.02%       (p=0.029 n=10+9)
BM<InsertHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/4k           43.0ns ± 3%    39.8ns ± 1%   -7.64%       (p=0.000 n=10+10)
BM<InsertHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/64k          44.0ns ± 4%    40.7ns ± 2%   -7.35%       (p=0.000 n=10+10)
BM<InsertHit_Cold, gtl::flat_hash_set, 64, Density::kMax>/1M           43.5ns ± 2%    40.8ns ± 1%   -6.19%       (p=0.000 n=10+10)
BM<InsertManyUnordered_Hot, gtl::flat_hash_set, 64, Density::kMax>/16  10.2ns ± 0%    12.9ns ± 0%  +26.96%       (p=0.000 n=10+10)
BM<InsertManyUnordered_Hot, gtl::flat_hash_set, 64, Density::kMax>/256  8.18ns ± 0%    12.00ns ± 0%  +46.70%       (p=0.000 n=9+7)
BM<InsertManyUnordered_Hot, gtl::flat_hash_set, 64, Density::kMax>/4k  22.1ns ± 4%    25.1ns ± 3%  +13.57%       (p=0.000 n=10+9)
BM<InsertManyUnordered_Hot, gtl::flat_hash_set, 64, Density::kMax>/64k 23.1ns ± 4%    26.0ns ± 1%  +12.46%       (p=0.000 n=10+9)
BM<InsertManyUnordered_Hot, gtl::flat_hash_set, 64, Density::kMax>/1M   106ns ± 1%     109ns ± 1%   +2.78%       (p=0.000 n=9+10)
BM<InsertManyUnordered_Cold, gtl::flat_hash_set, 64, Density::kMax>/16  117ns ±27%    108ns ± 3%     ~          (p=0.218 n=10+10)
BM<InsertManyUnordered_Cold, gtl::flat_hash_set, 64, Density::kMax>/256 121ns ±30%    108ns ± 1%     ~          (p=0.153 n=10+10)
BM<InsertManyUnordered_Cold, gtl::flat_hash_set, 64, Density::kMax>/4k  121ns ±31%    108ns ± 1%     ~          (p=0.204 n=10+10)
BM<InsertManyUnordered_Cold, gtl::flat_hash_set, 64, Density::kMax>/64k 121ns ±31%    108ns ± 1%     ~          (p=0.396 n=10+10)
BM<InsertManyUnordered_Cold, gtl::flat_hash_set, 64, Density::kMax>/1M  122ns ±31%    108ns ± 1%     ~          (p=0.447 n=10+10)
BM<InsertManyOrdered_Hot, gtl::flat_hash_set, 64, Density::kMax>/16    13.0ns ± 0%    12.6ns ± 0%   -2.69%       (p=0.000 n=8+10)
BM<InsertManyOrdered_Hot, gtl::flat_hash_set, 64, Density::kMax>/256   12.9ns ± 1%    12.9ns ± 1%   -0.69%       (p=0.012 n=10+10)
BM<InsertManyOrdered_Hot, gtl::flat_hash_set, 64, Density::kMax>/4k    14.1ns ± 1%    14.0ns ± 1%   -0.57%       (p=0.045 n=10+10)
BM<InsertManyOrdered_Hot, gtl::flat_hash_set, 64, Density::kMax>/64k   14.3ns ± 0%    13.9ns ± 0%   -3.07%       (p=0.000 n=10+8)
BM<InsertManyOrdered_Hot, gtl::flat_hash_set, 64, Density::kMax>/1M    22.3ns ±48%    19.5ns ± 5%     ~          (p=0.269 n=10+10)
BM<InsertManyOrdered_Cold, gtl::flat_hash_set, 64, Density::kMax>/16   118ns ±28%     109ns ± 2%     ~          (p=0.132 n=10+10)
BM<InsertManyOrdered_Cold, gtl::flat_hash_set, 64, Density::kMax>/256  122ns ±30%     108ns ± 1%     ~          (p=0.717 n=10+10)
BM<InsertManyOrdered_Cold, gtl::flat_hash_set, 64, Density::kMax>/4k   120ns ±31%     108ns ± 1%     ~          (p=0.341 n=10+10)
BM<InsertManyOrdered_Cold, gtl::flat_hash_set, 64, Density::kMax>/64k  17.4ns ± 9%    21.0ns ± 4%  +20.93%       (p=0.000 n=10+10)
BM<InsertManyOrdered_Cold, gtl::flat_hash_set, 64, Density::kMax>/1M   18.2ns ± 8%    24.8ns ±47%     ~          (p=0.052 n=8+10)
```

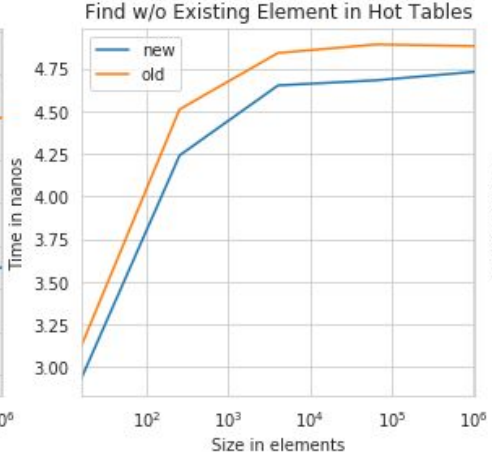

Find w/ Existing Element in Cold Tables
new
old
Time in nanos
Size in elements

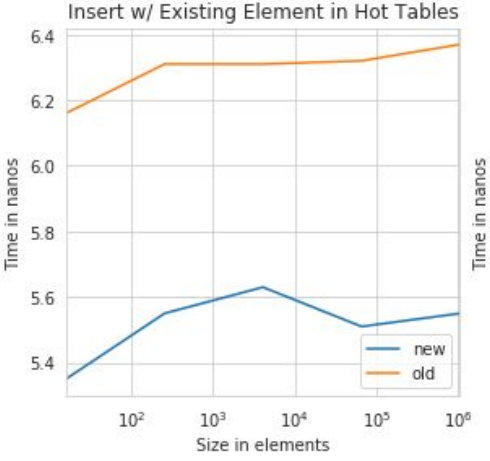

Find w/o Existing Element in Cold Tables
new
old
Time in nanos
Size in elements

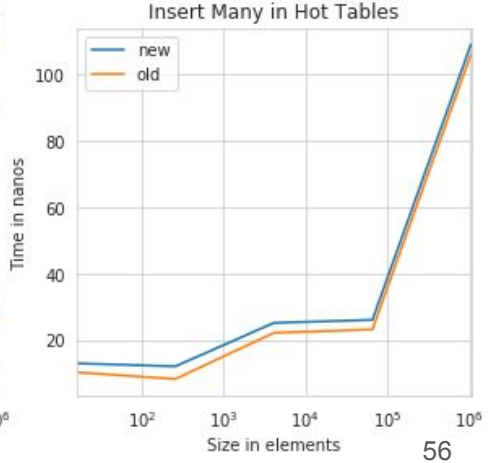

Insert w/ Existing Element in Cold Tables
new
old
Time in nanos
Size in elements

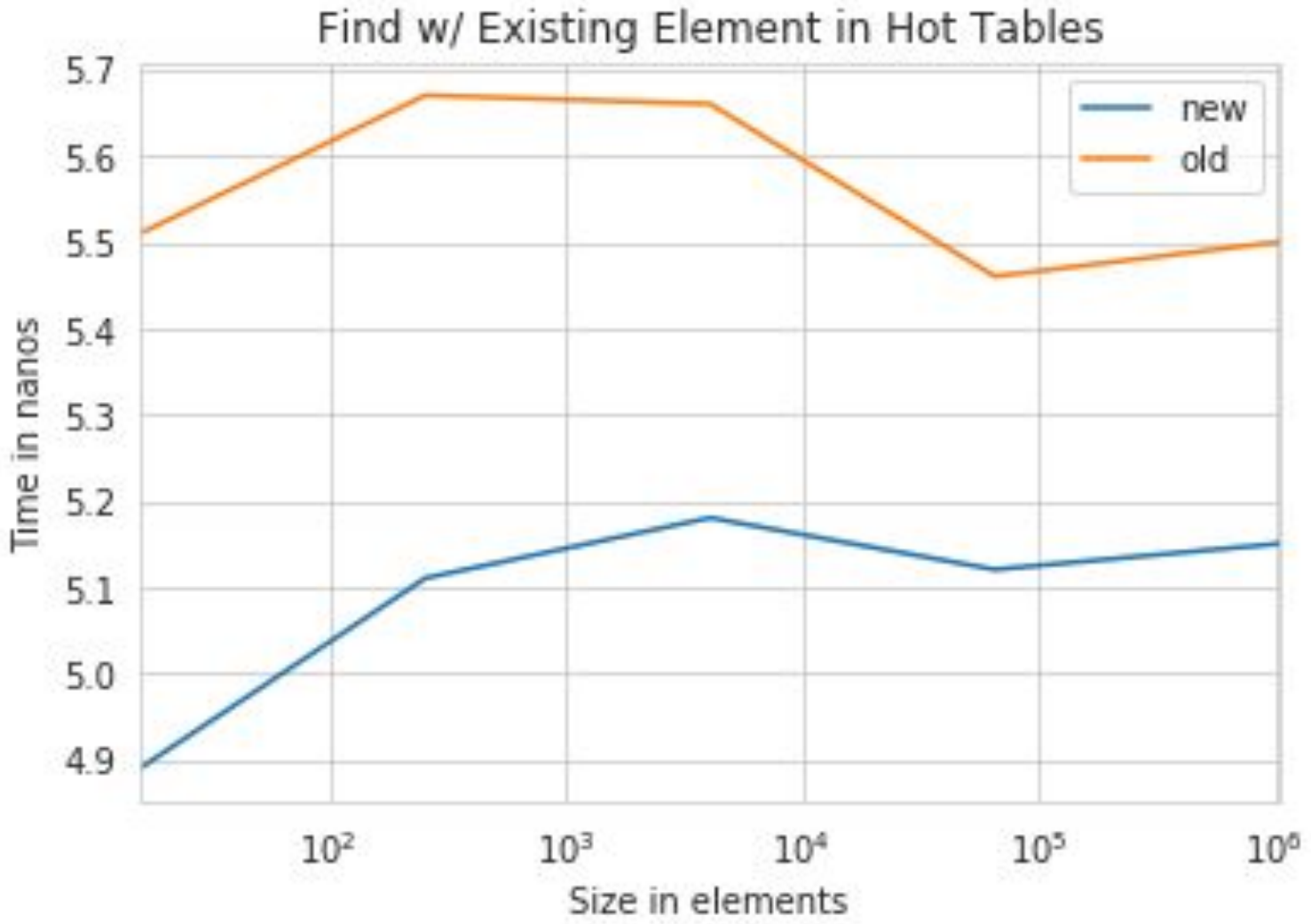

Insert Many in Cold Tables
new
old
Time in nanos
Size in elements

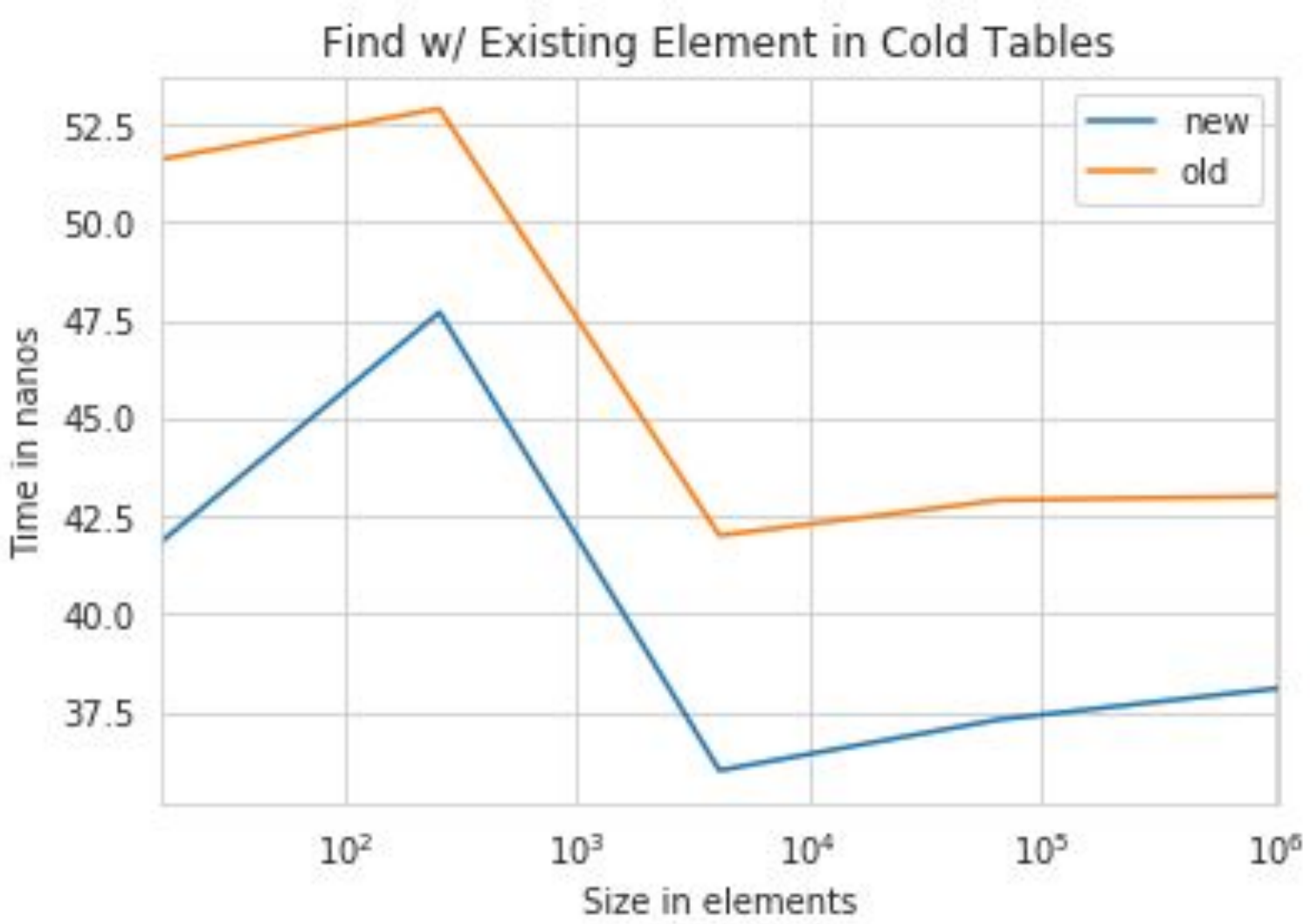

Find w/ Existing Element in Hot Tables
new
old
Time in nanos
Size in elements

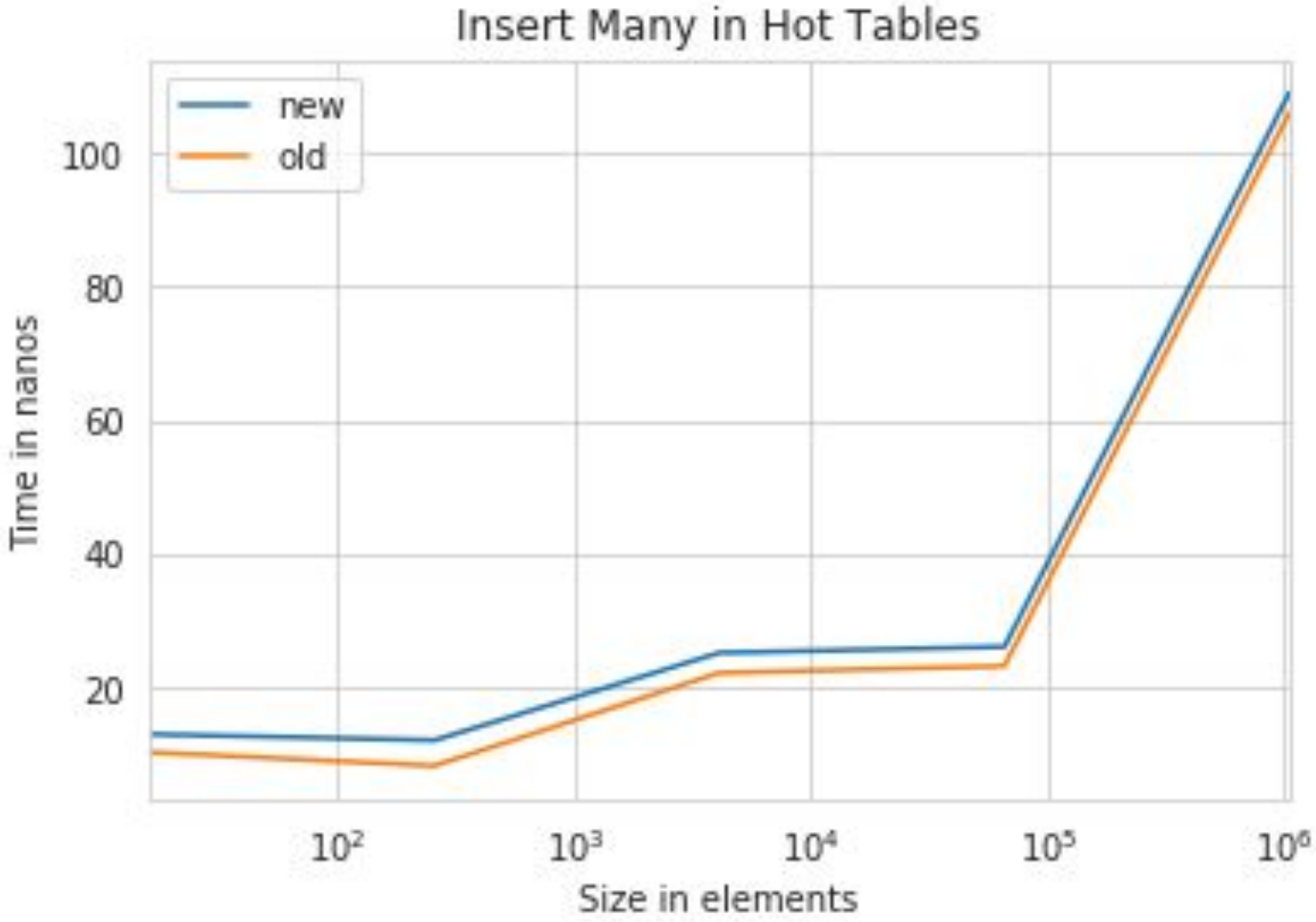

Find w/o Existing Element in Hot Tables
new
old
Time in nanos
Size in elements

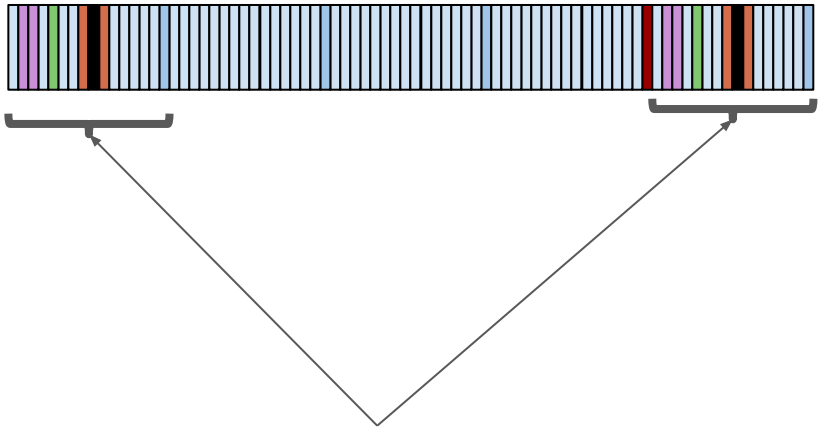

Insert w/ Existing Element in Hot Tables
new
old
Time in nanos
Size in elements


Insert Many in Hot Tables
new
old
Time in nanos
Size in elements


Find w/ Existing Element in Hot Tables
5.7
5.6
5.5
5.4
5.3
5.2
5.1
5.0
4.9
Time in nanos
new
old
10^2
10^3
10^4
10^5
10^6
Size in elements


Find w/ Existing Element in Cold Tables
new
old
Time in nanos
52.5
50.0
47.5
45.0
42.5
40.0
37.5
10^2
10^3
10^4
10^5
10^6
Size in elements


Insert Many in Hot Tables
new
old
100
80
60
40
20
Time in nanos
10^2
10^3
10^4
10^5
10^6
Size in elements

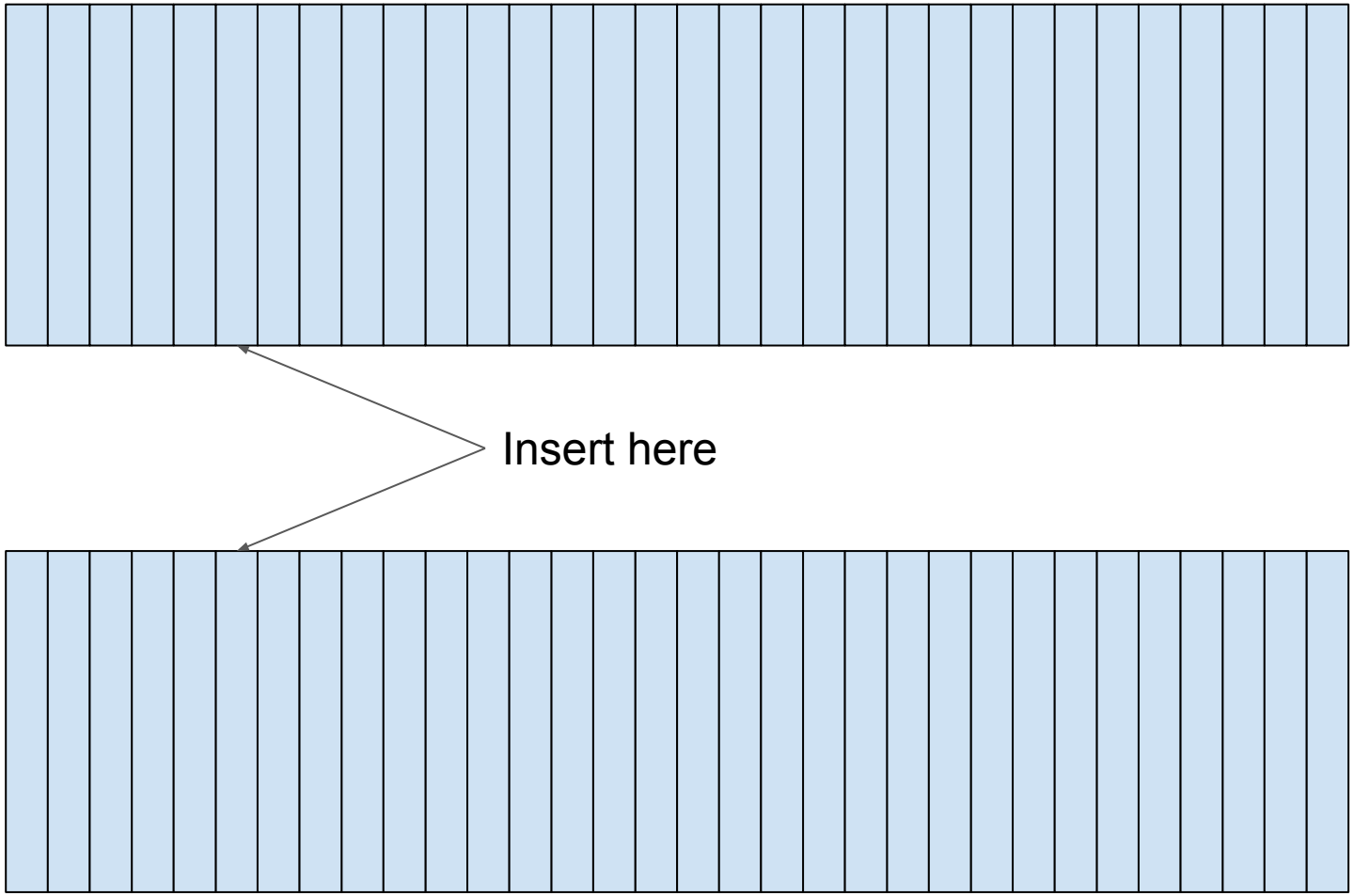


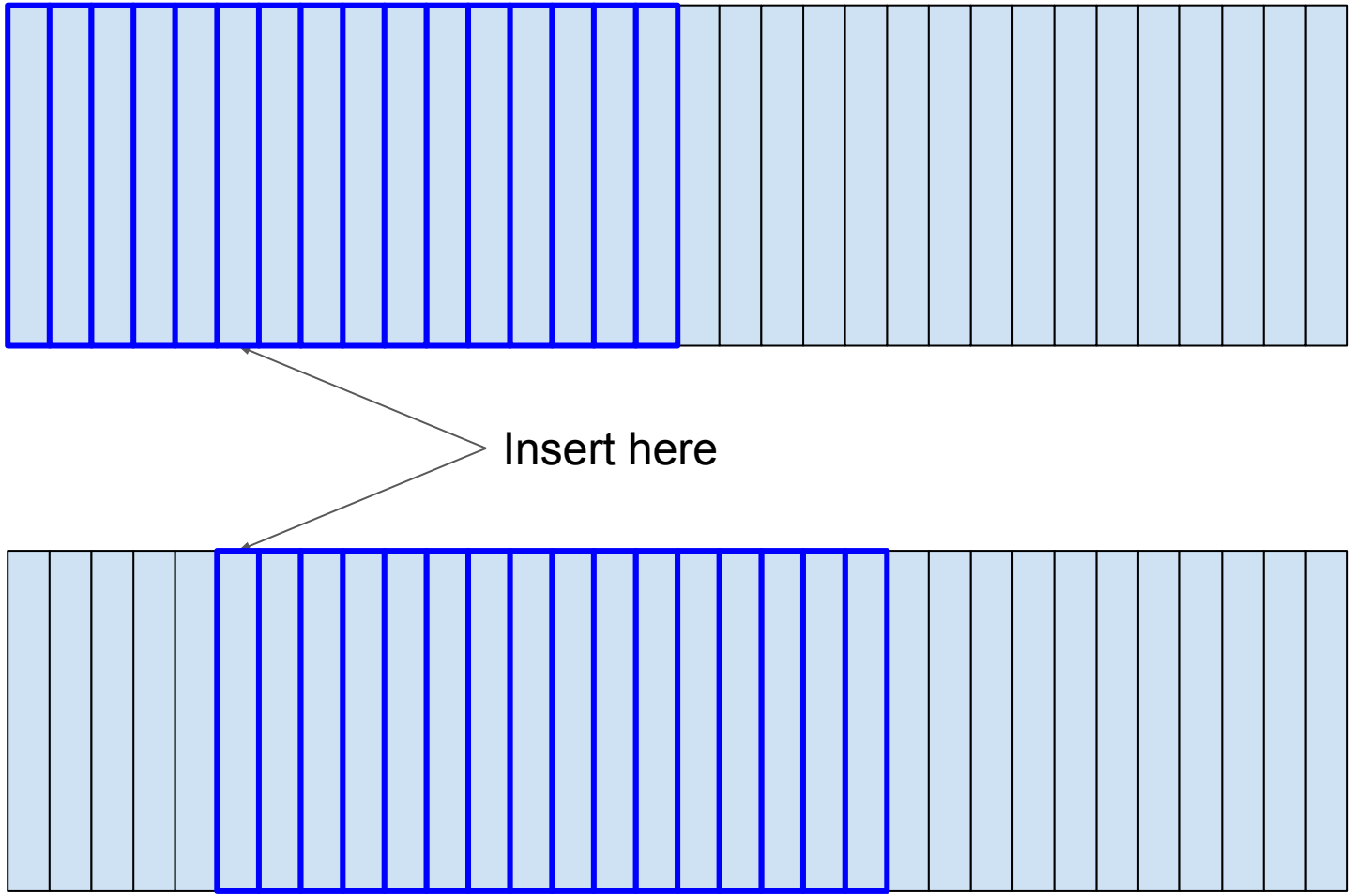

```
void set_ctrl(size_t i, ctrl_t h) {
  assert(i < capacity_);
  ctrl_[i] = h;
  if (i < 16) ctrl_[i + 1 + capacity_] = h;
}
```

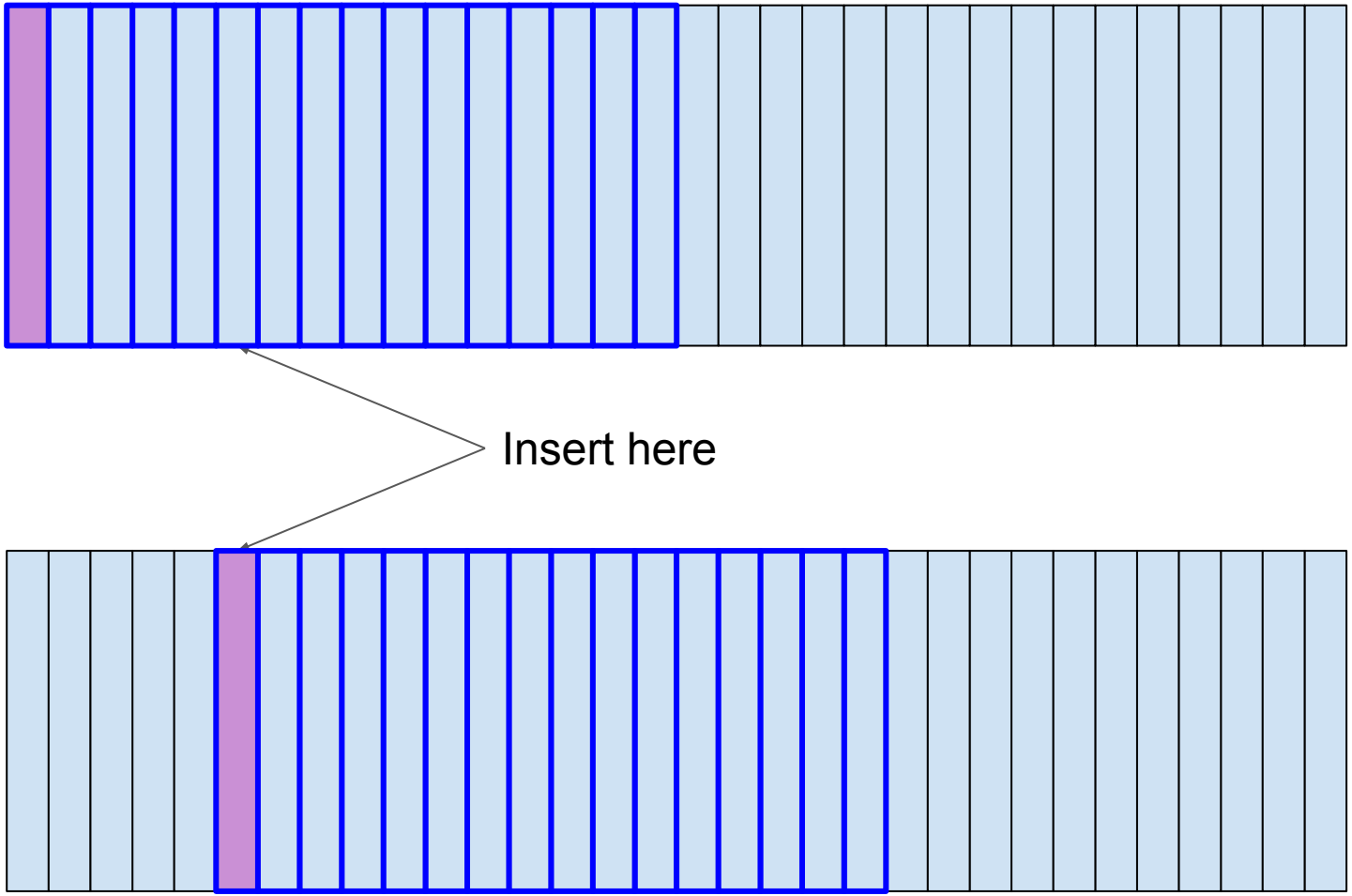
Old

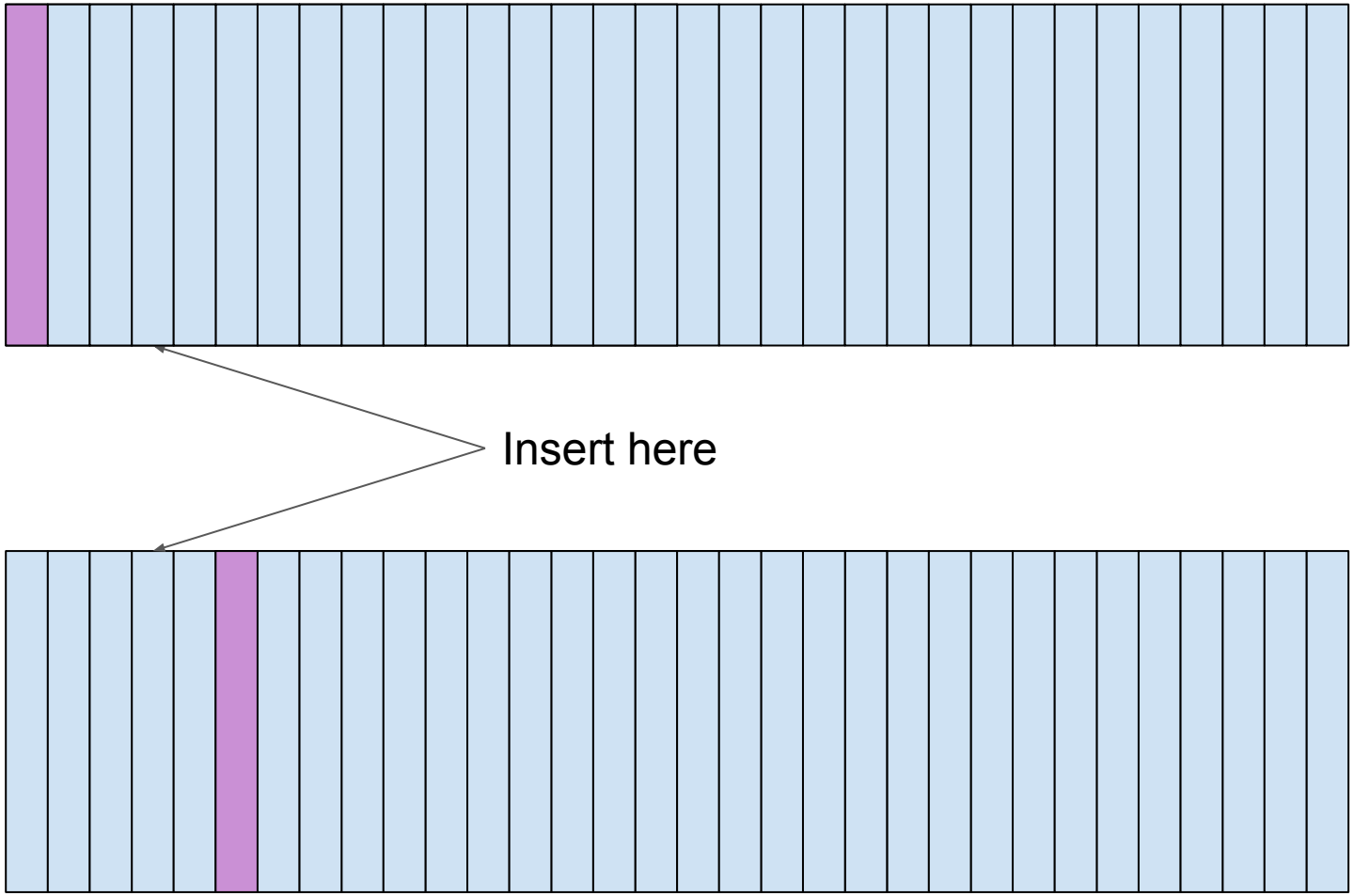
vs

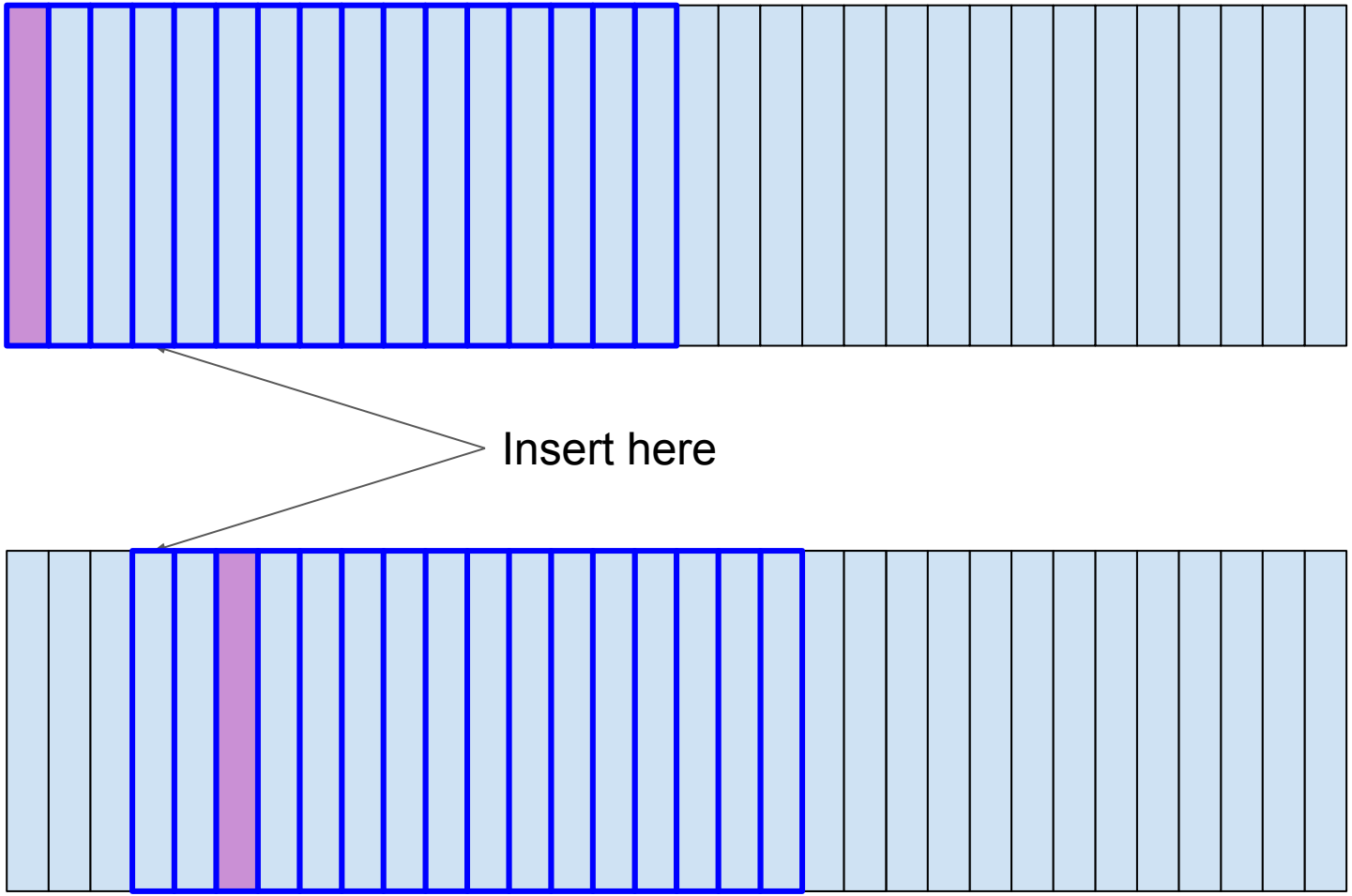
New

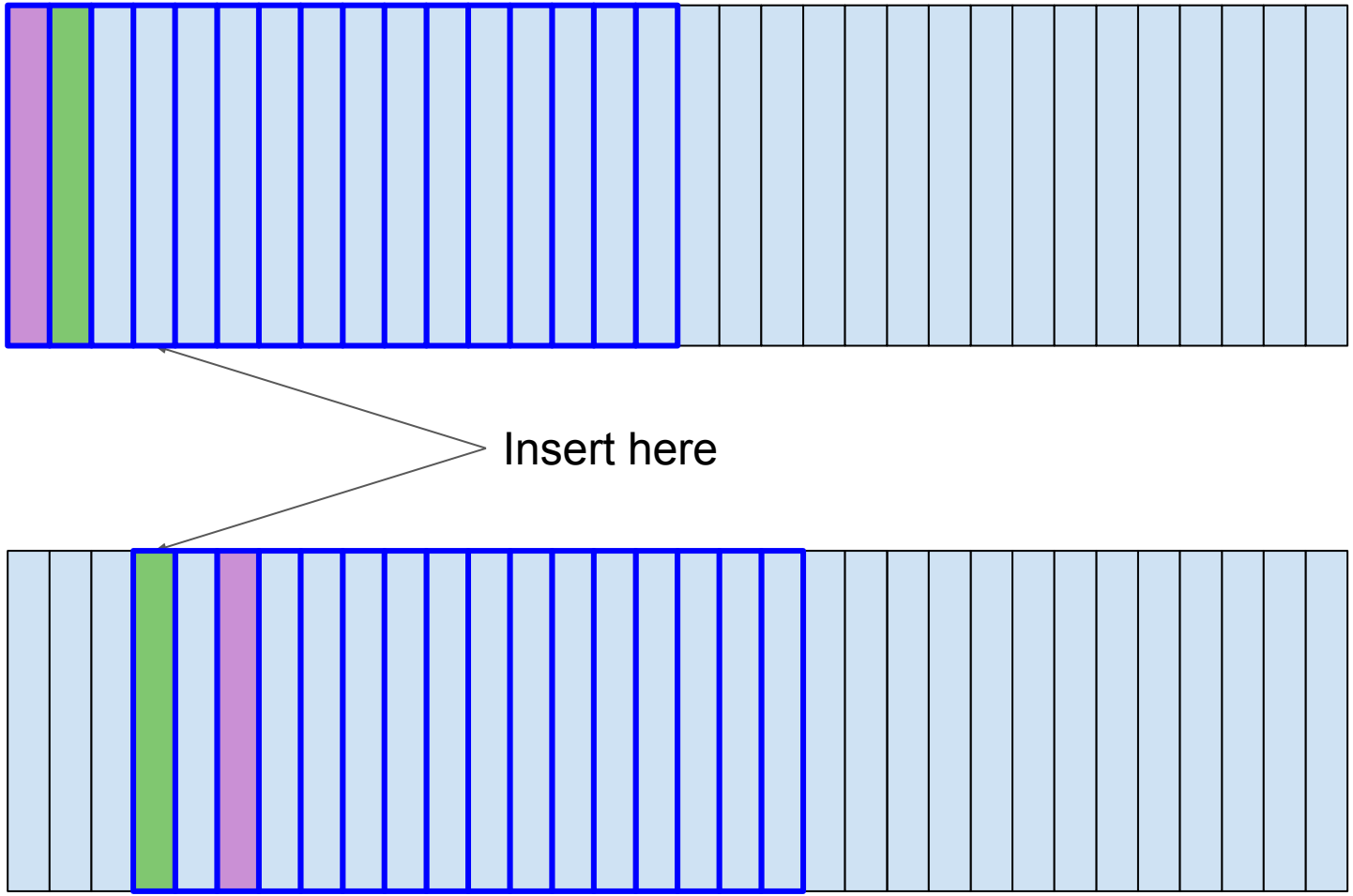

Insert here

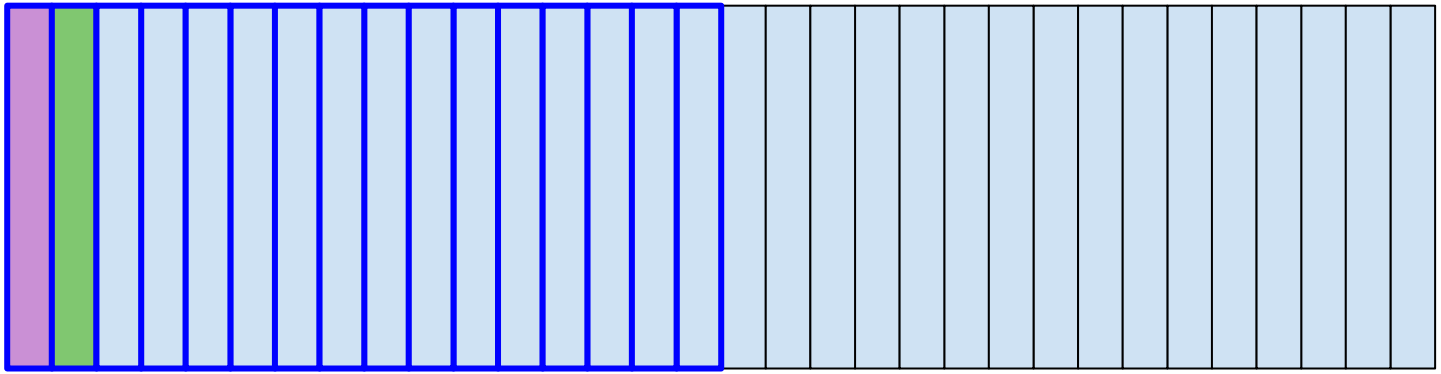

Insert here

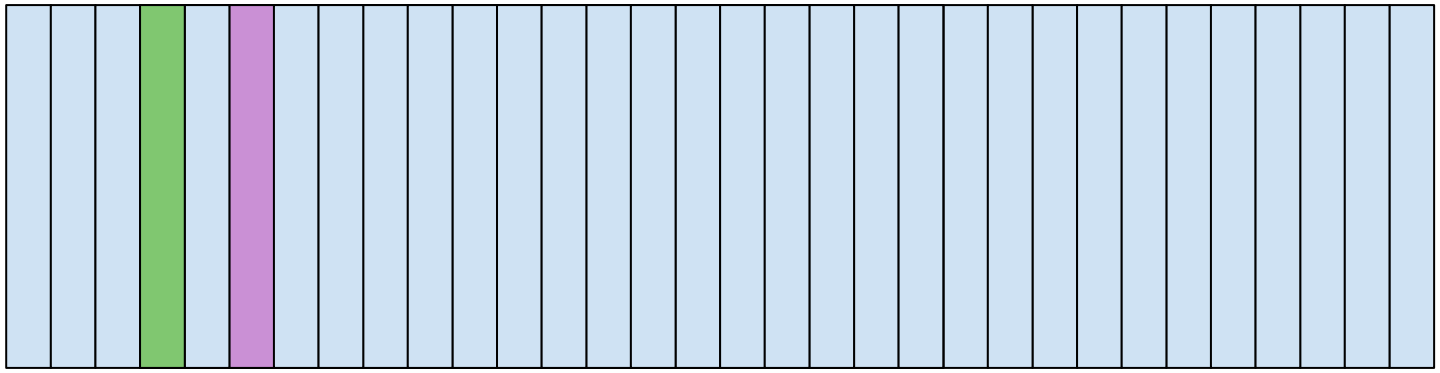

Insert here

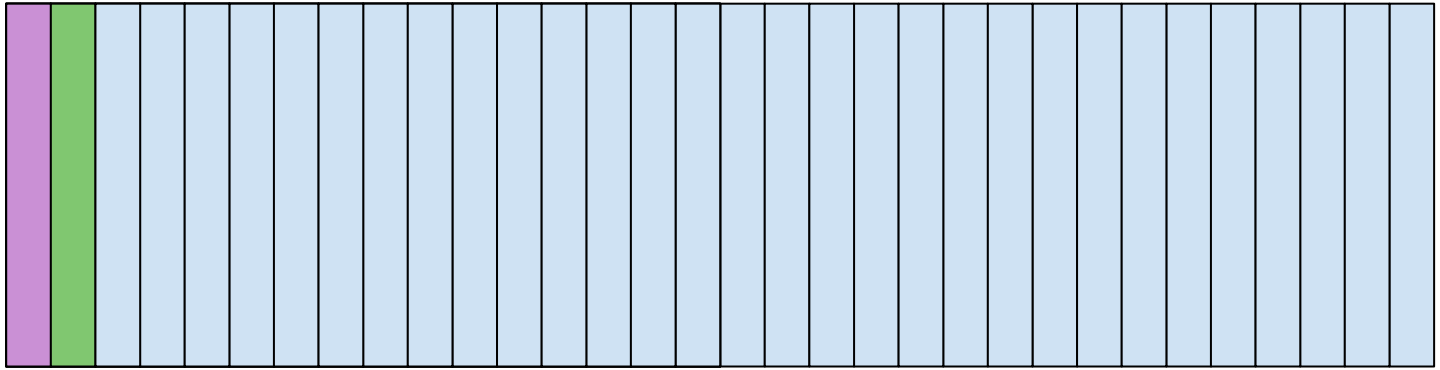

Insert here

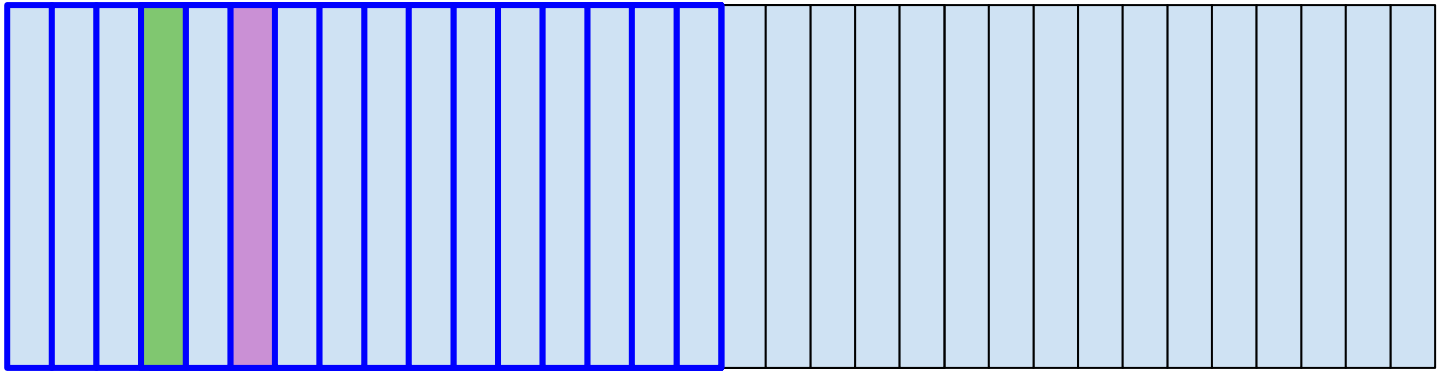

Insert here


Insert here





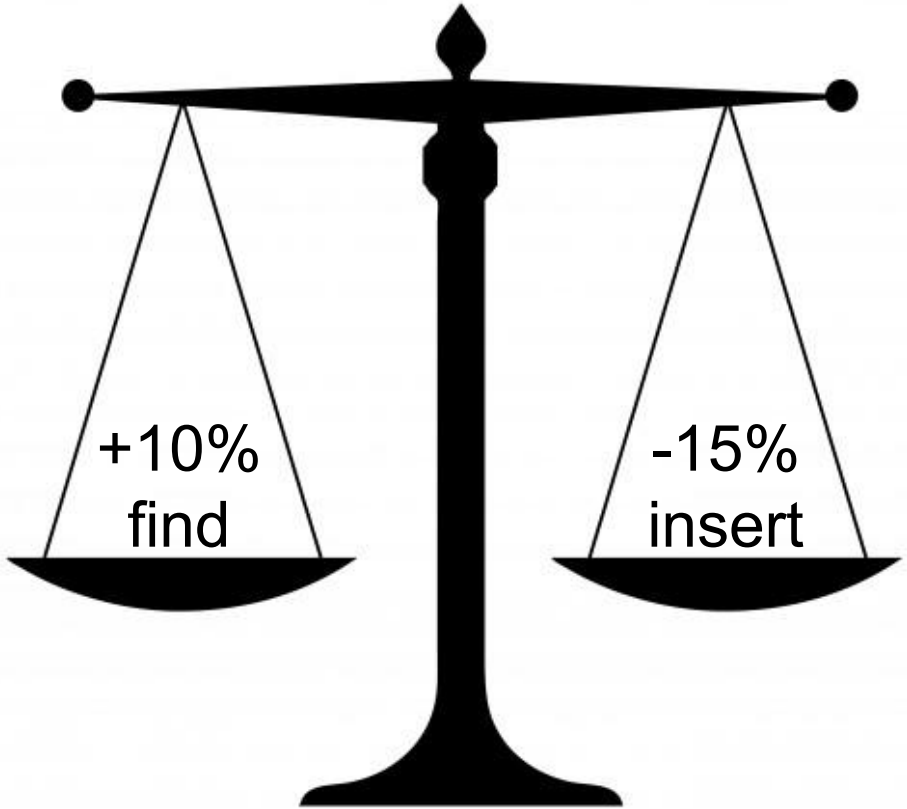




```
-|  size_t group = H1(hash) % num_groups_;
+|  size_t pos = H1(hash) % capacity_;
 |  while (true) {
-|    Group g{ctrl_ + group * 16};
+|    Group g{ctrl_ + pos};
 |    for (int i : g.Match(H2(hash))) {
-|      if (key == slots_[group * 16 + i])
-|        return iterator_at(group * 16 + i);
+|      if (key == slots_[pos + i])
+|        return iterator_at(pos + i);
 |    }
 |    if (g.MatchEmpty()) return end();
-|    group = (group + 1) % num_groups_;
+|    pos = (pos + 16) % capacity_;
 |  }
```

```
-|  size_t group = H1(hash) % num_groups_;
+|  size_t pos = H1(hash) % capacity_;
 |  while (true) {
-|    Group g{ctrl_ + group * 16};
+|    Group g{ctrl_ + pos};
 |    for (int i : g.Match(H2(hash))) {
-|      if (key == slots_[group * 16 + i])
-|        return iterator_at(group * 16 + i);
+|      if (key == slots_[pos + i])
+|        return iterator_at(pos + i);
 |    }
 |    if (g.MatchEmpty()) return end();
-|    group = (group + 1) % num_groups_;
+|    pos = (pos + 16) % capacity_;
 |  }
```

```
-|  size_t group = H1(hash) % num_groups_;
+|  size_t pos = H1(hash) % capacity_;
 |  while (true) {
-|    Group g{ctrl_ + group * 16};
+|    Group g{ctrl_ + pos};
 |    for (int i : g.Match(H2(hash))) {
-|      if (key == slots_[group * 16 + i])
-|        return iterator_at(group * 16 + i);
+|      if (key == slots_[pos + i])
+|        return iterator_at(pos + i);
 |    }
 |    if (g.MatchEmpty()) return end();
-|    group = (group + 1) % num_groups_;
+|    pos = (pos + 16) % capacity_;
 |  }
```


+10%
find
-15%
insert

---- queries_per_sec ----------

XXXX +/- YY (24 vals) | %diff: 0.5404 +/-0.2922%

XXXX +/- YY (24 vals) | Possible significant effect (p=0.001)

---- avg_working_time (usec) ----------

XXXX +/- YY (24 vals) | %diff: -0.8128 +/-0.3565%

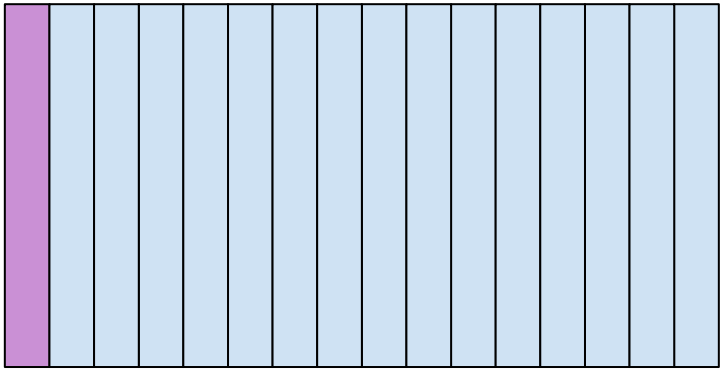
XXXX +/- YY (24 vals) | Possible significant effect (p=0.000)


STOP

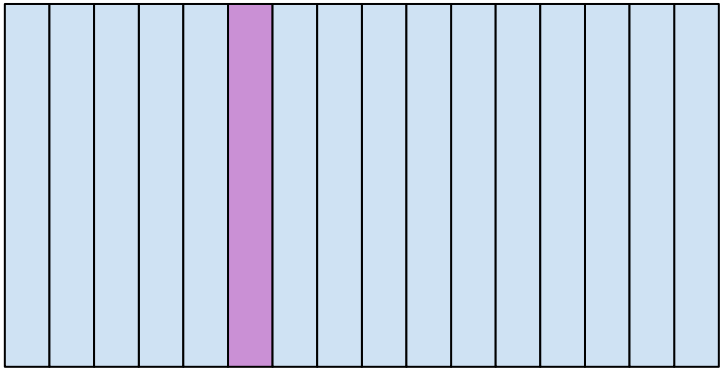
Old

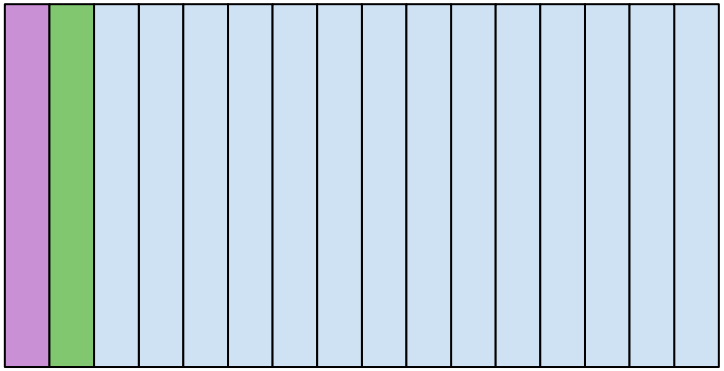
vs

New

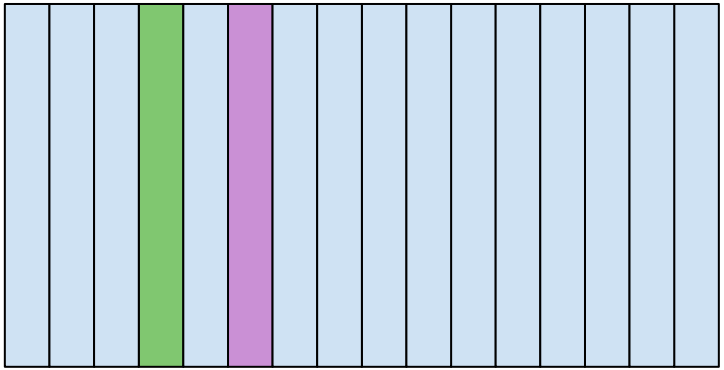


vs



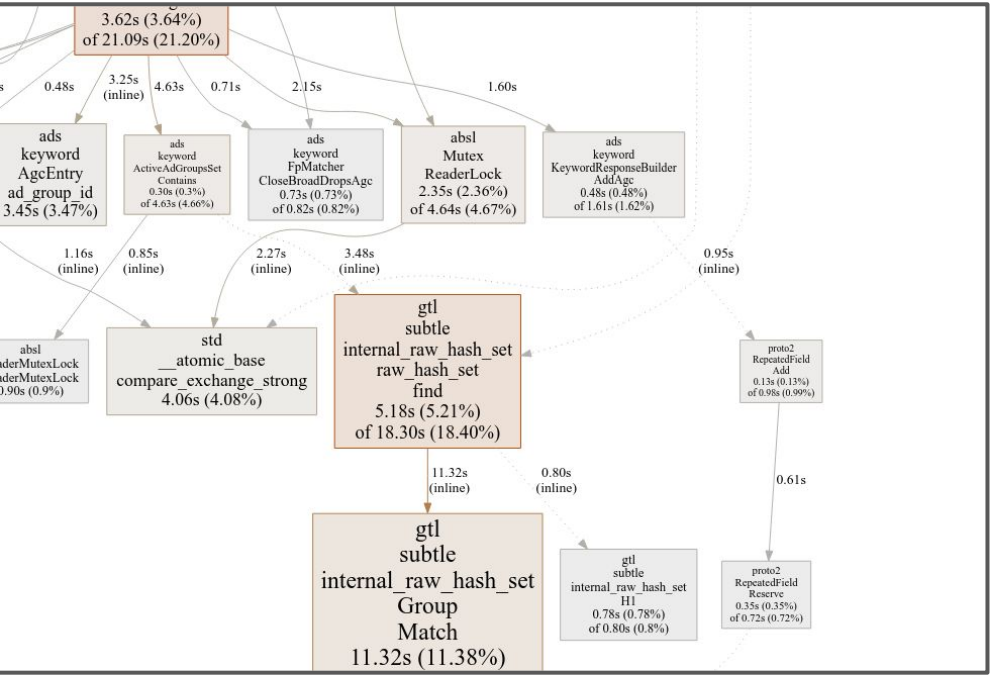


vs

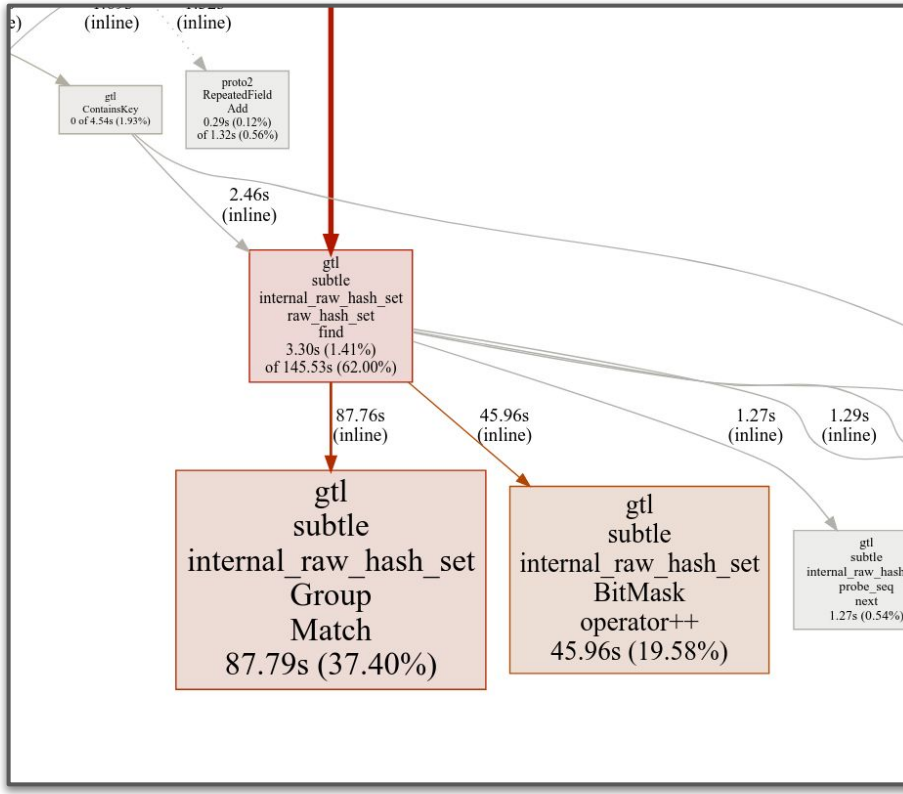



magritte

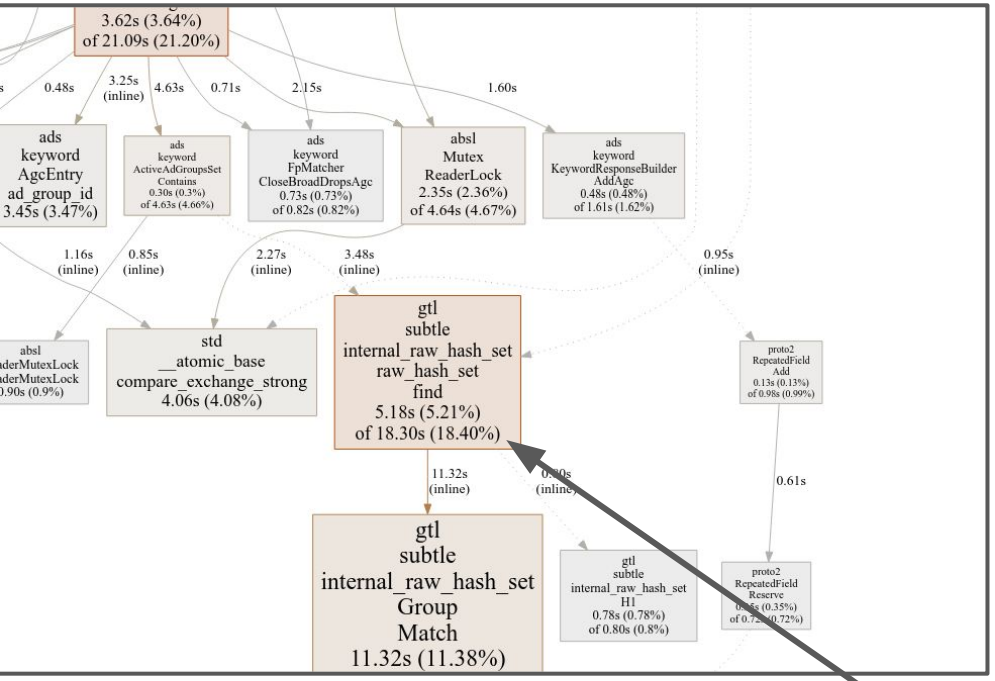
Before



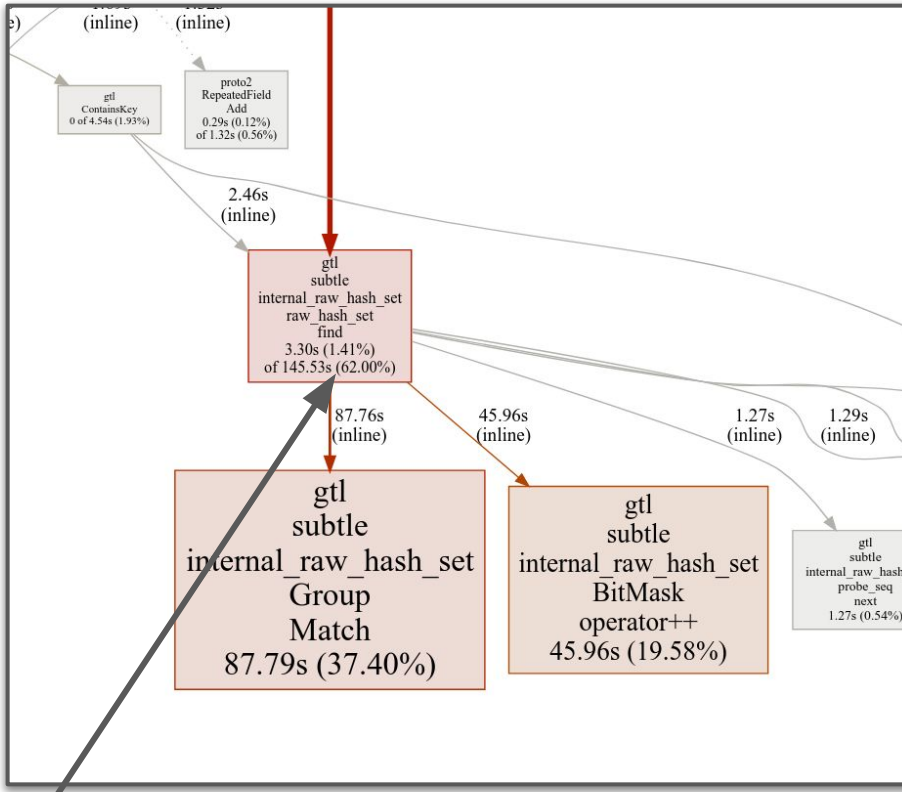
After



Before



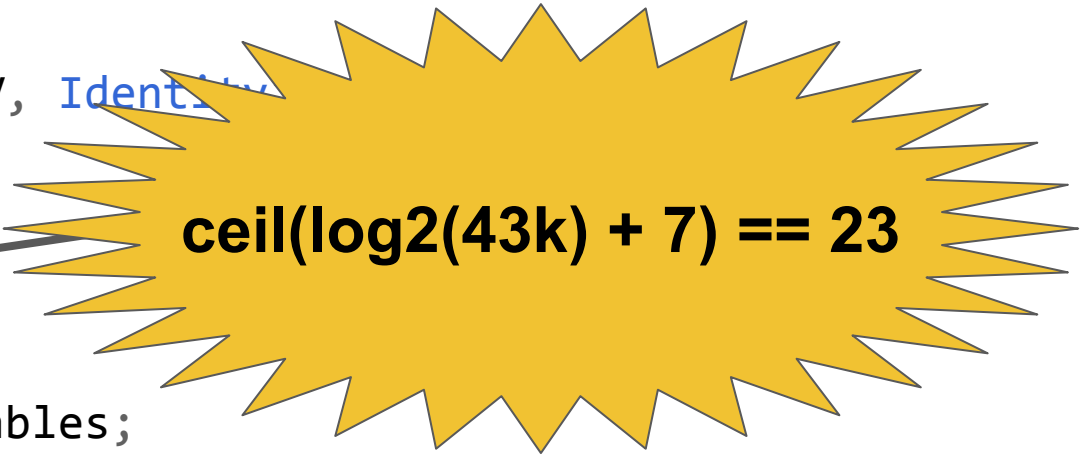
After



That seems bad...

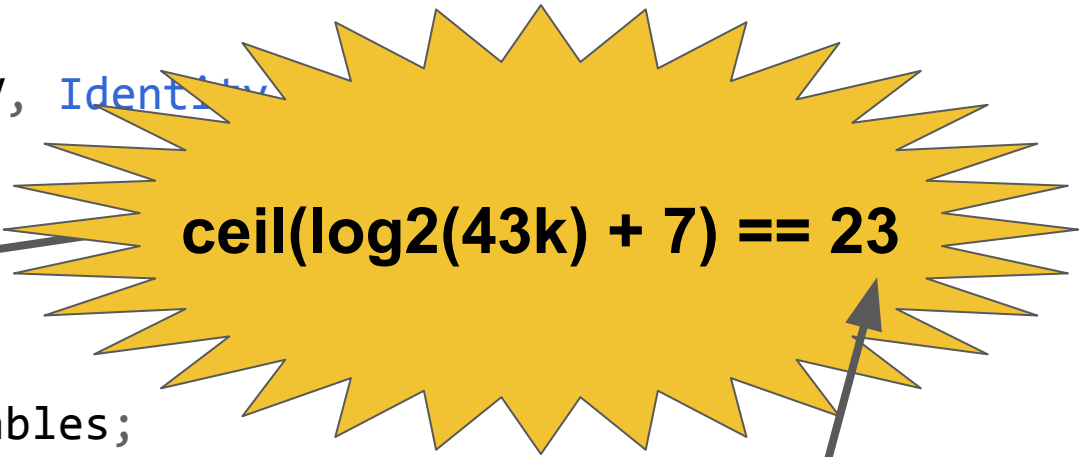
```
struct ShardedTable {
  absl::flat_hash_map<uint64, V, Identity> table;
  absl::Mutex mu;
};

// ~43K elements per shard
std::array<ShardedTable, 32> tables;

uint64 TableOffset(uint64 fprint) { return (fprint >> 20) & 31; }

T Find(uint64 fprint) {
  ShardedTable& t = tables[TableOffset(fprint)];
  absl::ReaderMutexLock l(&t.mu);
  return t.table.at(fprint);
}
```

```
struct ShardedTable {
  absl::flat_hash_map<uint64, V, Identity
  absl::Mutex mu;
};

// ~43K elements per shard
std::array<ShardedTable, 32> tables;

uint64 TableOffset(uint64 fprint) { return (fprint >> 20) & 31; }

T Find(uint64 fprint) {
  ShardedTable& t = tables[TableOffset(fprint)];
  absl::ReaderMutexLock l(&t.mu);
  return t.table.at(fprint);
}
```



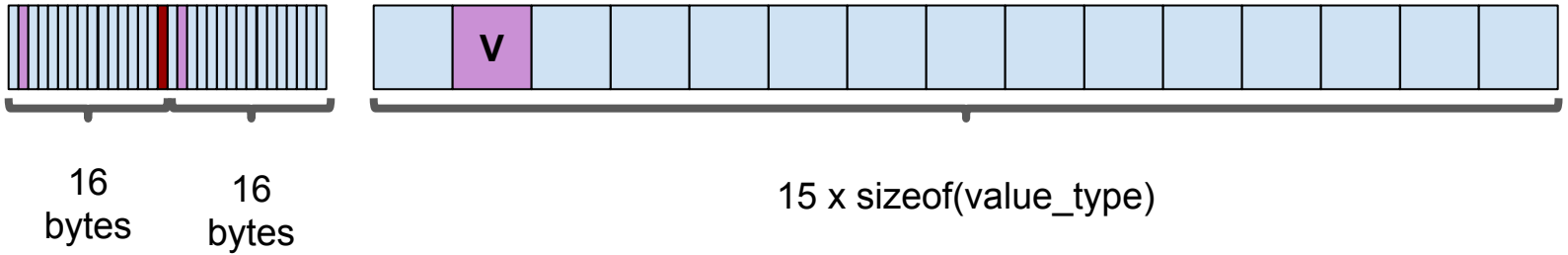
```
-|  size_t group = H1(hash) % num_groups_;
+|  size_t pos = H1(hash) % capacity_;
 |  while (true) {
-|    Group g{ctrl_ + group * 16};
+|    Group g{ctrl_ + pos};
 |    for (int i : g.Match(H2(hash))) {
-|      if (key == slots_[group * 16 + i])
-|        return iterator_at(group * 16 + i);
+|      if (key == slots_[pos + i])
+|        return iterator_at(pos + i);
 |    }
 |    if (g.MatchEmpty()) return end();
-|    group = (group + 1) % num_groups_;
+|    pos = (pos + 16) % capacity_;
 |  }
```

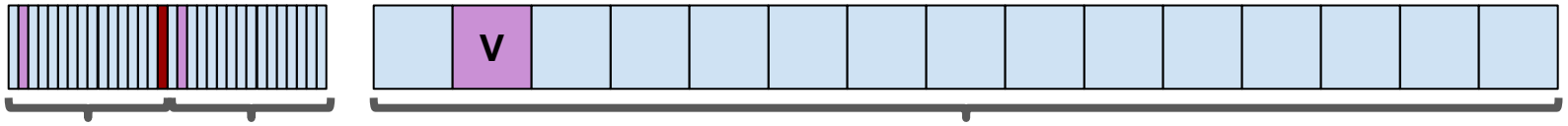
```
struct ShardedTable {
  absl::flat_hash_map<uint64, V, Identity
  absl::Mutex mu;
};

// ~43K elements per shard
std::array<ShardedTable, 32> tables;

uint64 TableOffset(uint64 fprint) { return (fprint >> 20) & 31; }

T Find(uint64 fprint) {
  ShardedTable& t = tables[TableOffset(fprint)];
  absl::ReaderMutexLock l(&t.mu);
  return t.table.at(fprint);
}
```



```
absl::flat_hash_set<int> bar = {42};
```

```
absl::flat_hash_set<std::string> foo = {"hello world!"};
```

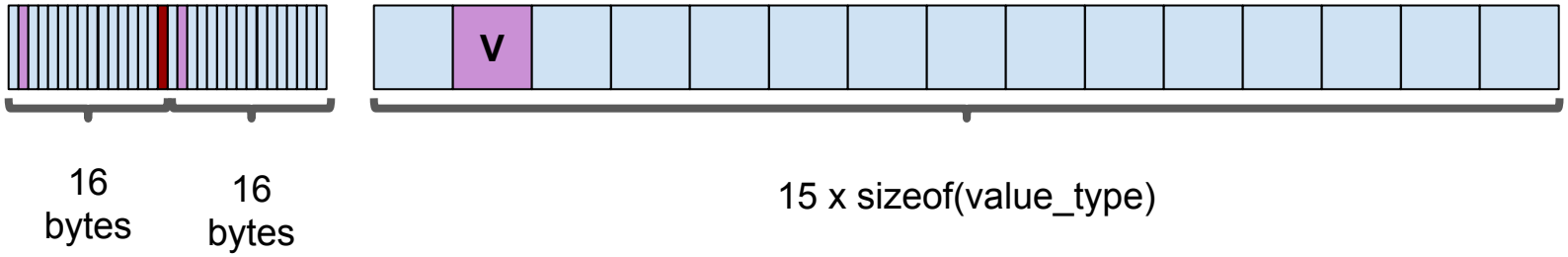

V
16
bytes
16
bytes
15 x sizeof(value_type)




```
absl::flat_hash_set<int> bar = {42};
```

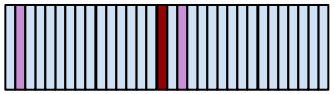
92 bytes!

```
absl::flat_hash_set<std::string> foo = {"hello world!"};
```

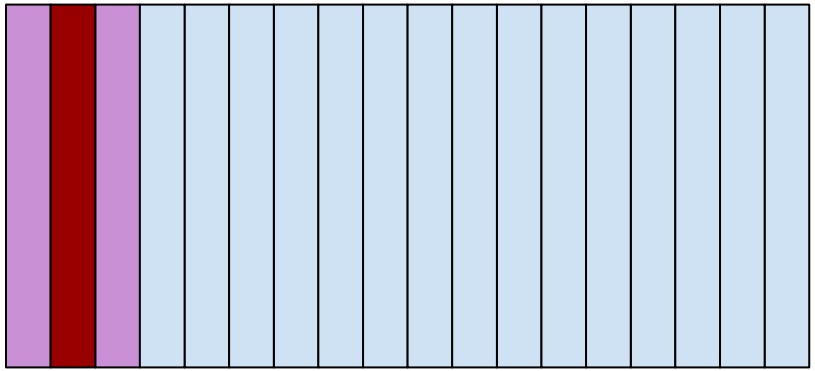


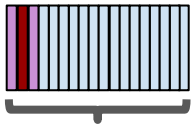

```
absl::flat_hash_set<int> bar = {42};
```

```
absl::flat_hash_set<std::string> foo = {"he
```



```
iterator find(const K& key, size_t hash) const {
  size_t pos = H1(hash) % capacity_;
  while (true) {
    Group g{ctrl_ + pos};
    for (int i : g.Match(H2(hash))) {
      if (key == slots_[pos + i])
        return iterator_at(pos + i);
    }
    if (g.MatchEmpty()) return end();
    group = (pos + 16) % capacity_;
  }
}
```





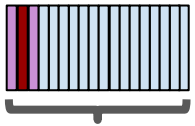
18
bytes

padding
bytes

sizeof(value_type)

```
absl::flat_hash_set<int> bar = {42};
```

```
absl::flat_hash_set<std::string> foo = {"hello world!"};
```



18
bytes

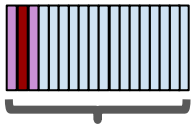
padding
bytes

sizeof(value_type)

```
absl::flat_hash_set<int> bar = {42};
```

24 bytes!

```
absl::flat_hash_set<std::string> foo = {"hello world!"};
```



18
bytes

padding
bytes

sizeof(value_type)

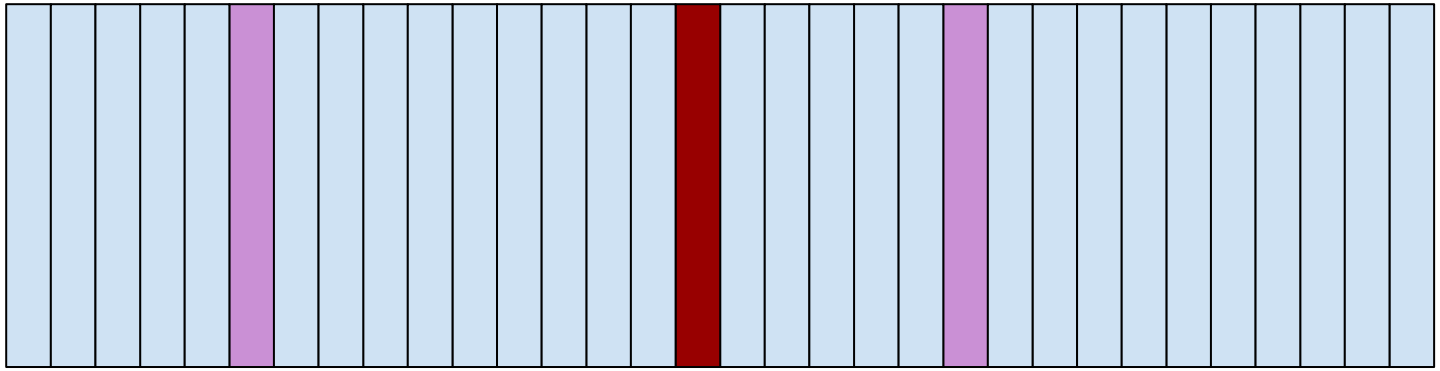
```
absl::flat_hash_set<int> bar = {42};
```

```
absl::flat_hash_set<std::string> foo = {"he
```

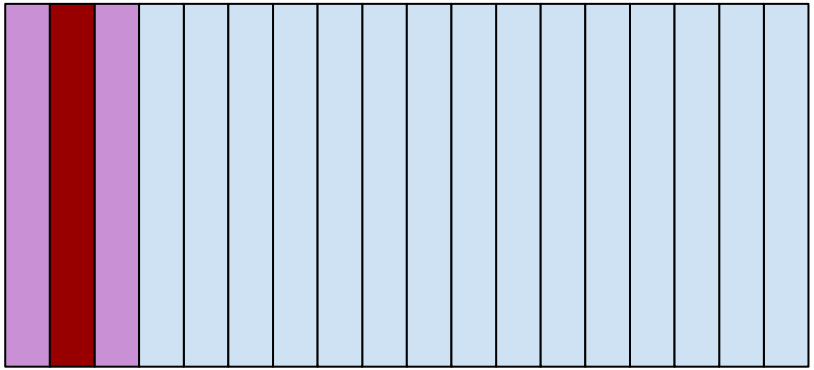
Old

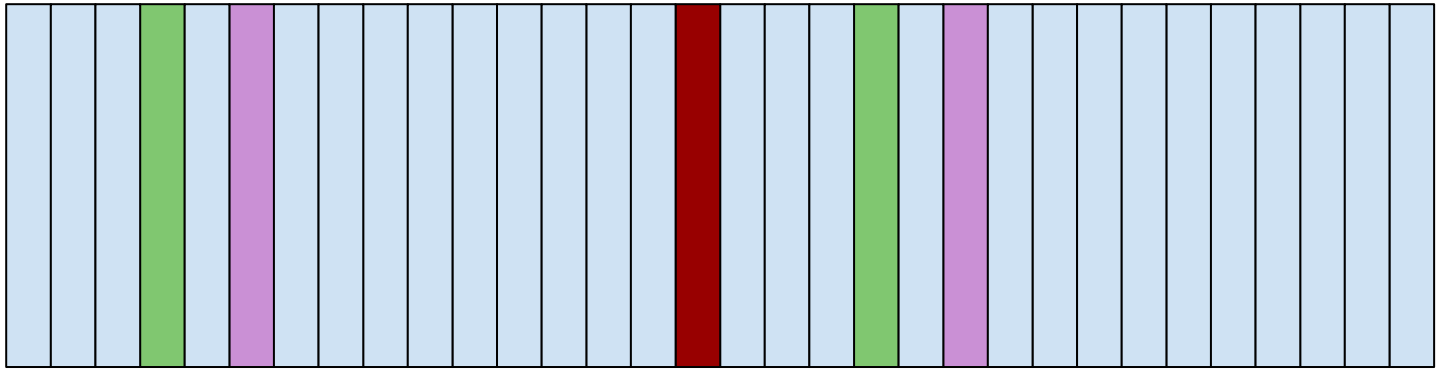
vs

New

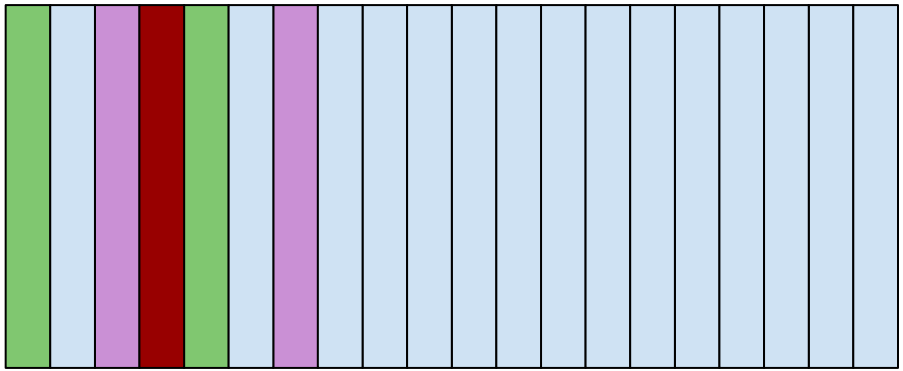


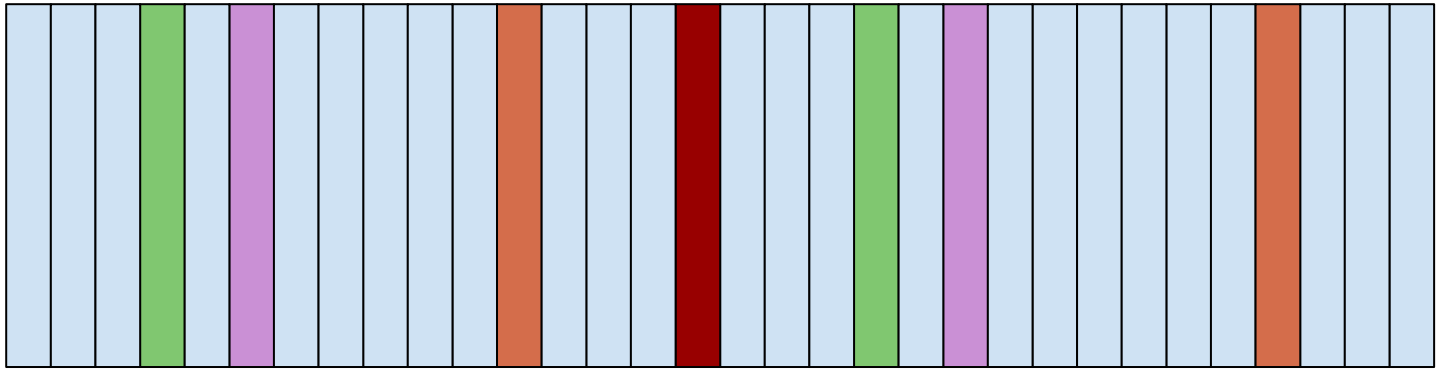
vs



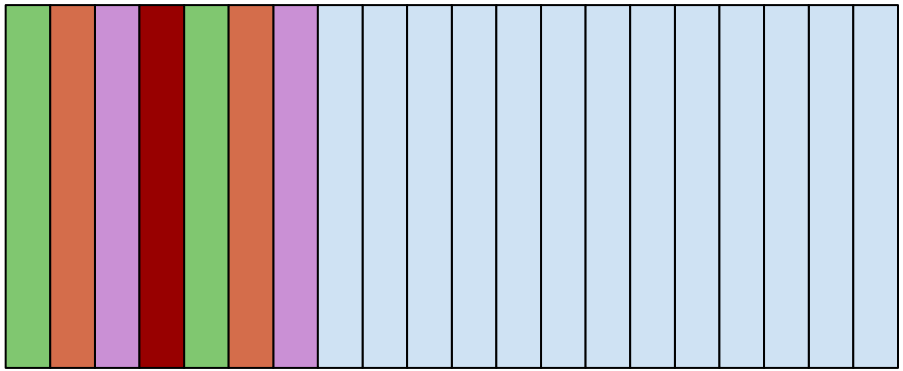


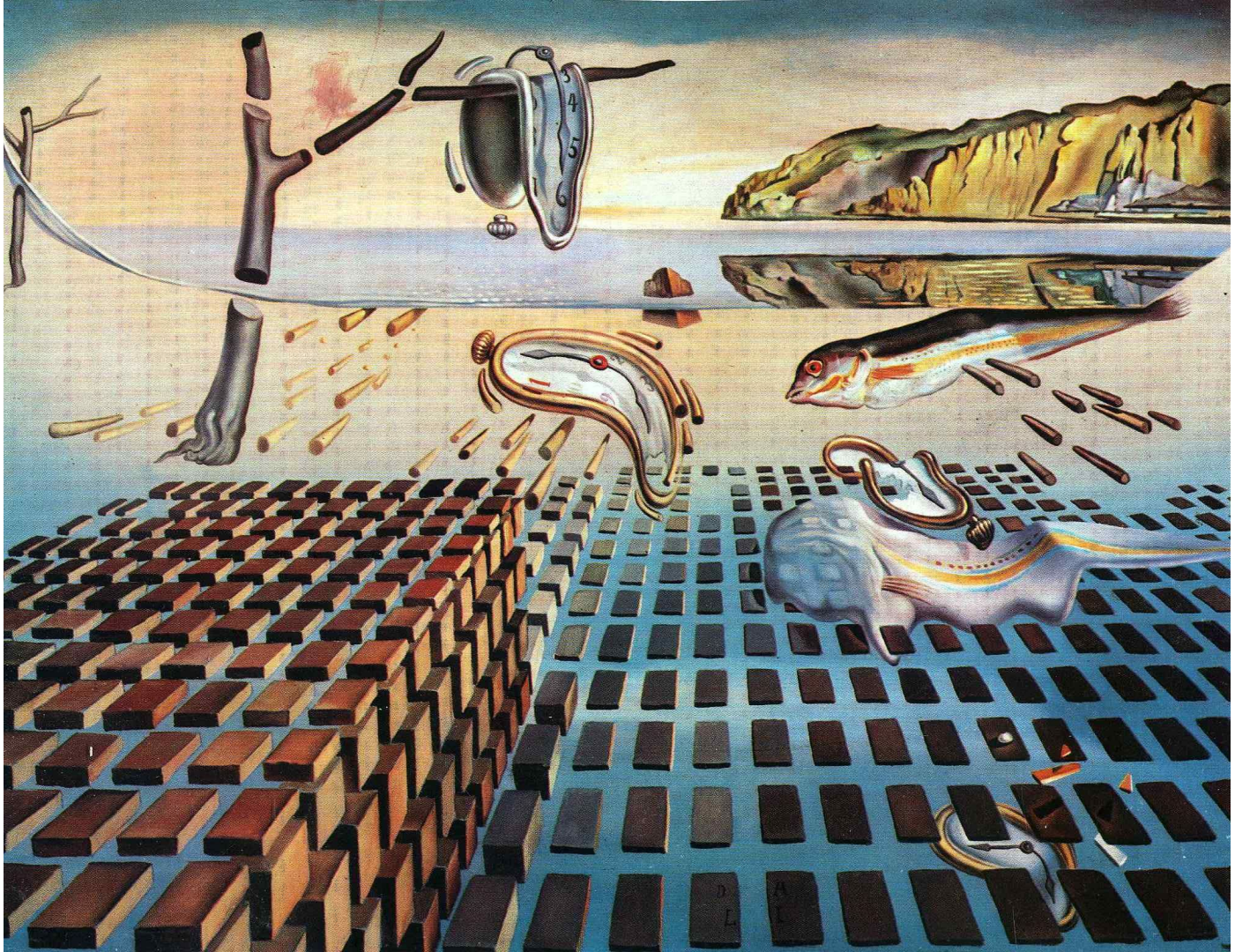
vs





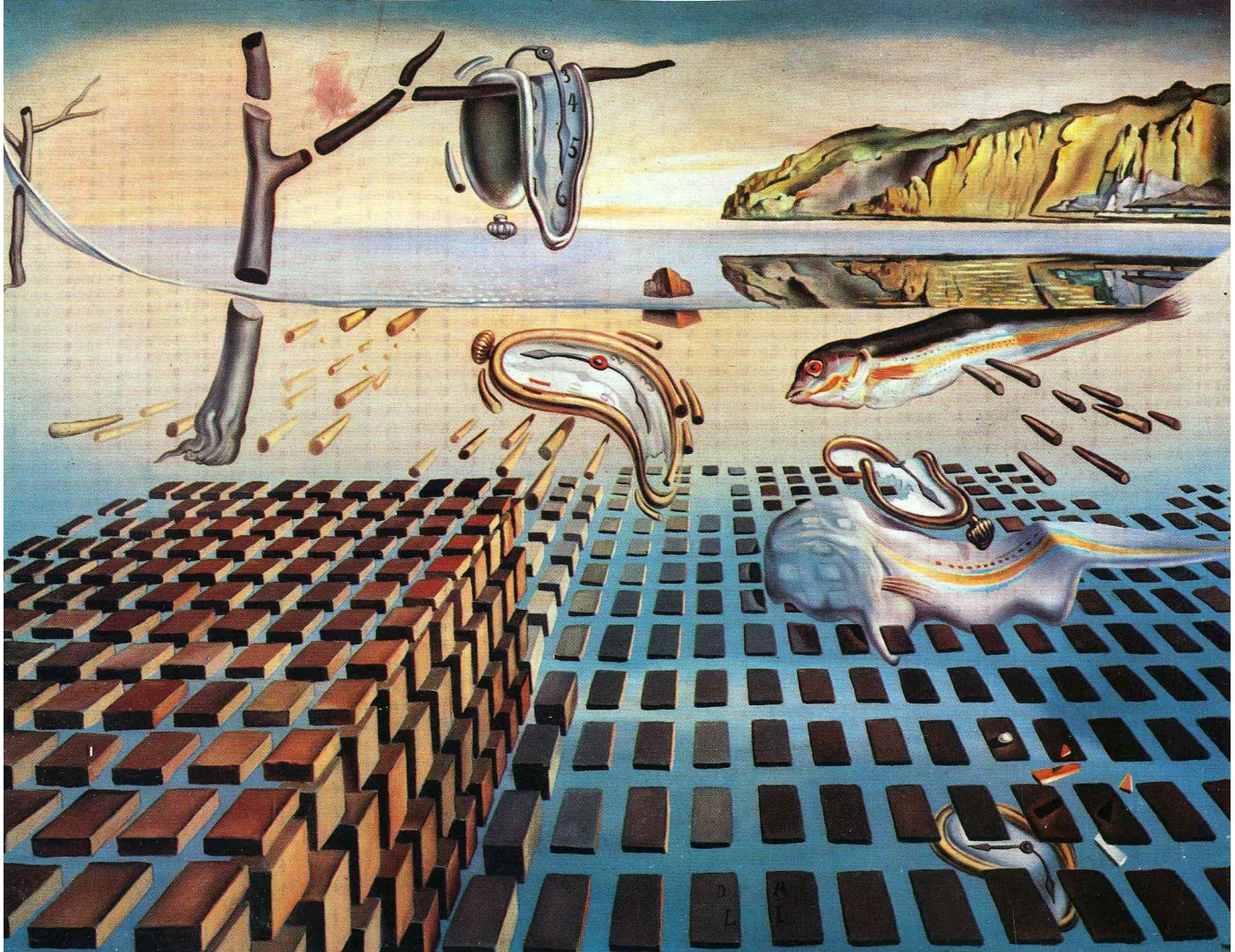
vs



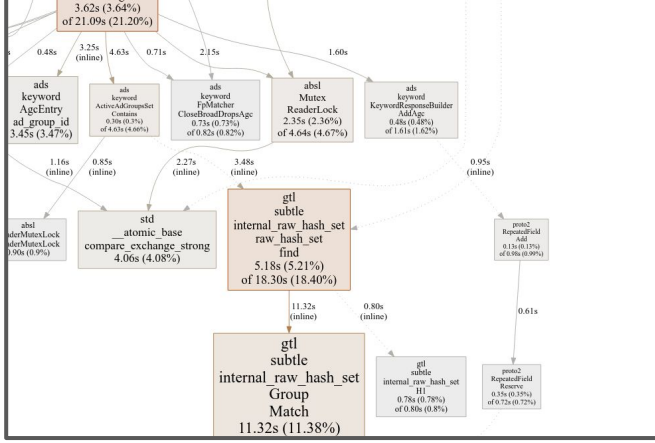


```
(pprof) top 4
      flat  flat%   sum%        cum   cum%
    14.08s 46.65% 46.65%     28.38s 94.04%  perftools_gwp::HashtableBenchmark::Run
     7.91s 26.21% 72.86%     14.64s 48.51%  raw_hash_set::prepare_insert
     6.47s 21.44% 94.30%      6.70s 22.20%  raw_hash_set::resize
     1.04s  3.45% 97.75%      1.04s  3.45%  RandenHwAes::Generate
```

```
(pprof) top 4
      flat  flat%   sum%        cum   cum%
     9.77s 32.30% 32.30%     10.35s 34.21%  raw_hash_set::resize
     9.69s 32.03% 64.33%     25.78s 85.22%  perftools_gwp::HashtableBenchmark::Run
     5.88s 19.44% 83.77%     16.27s 53.79%  raw_hash_set::prepare_insert
     3.25s 10.74% 94.51%      3.25s 10.74%  RandenHwAes::Generate
```

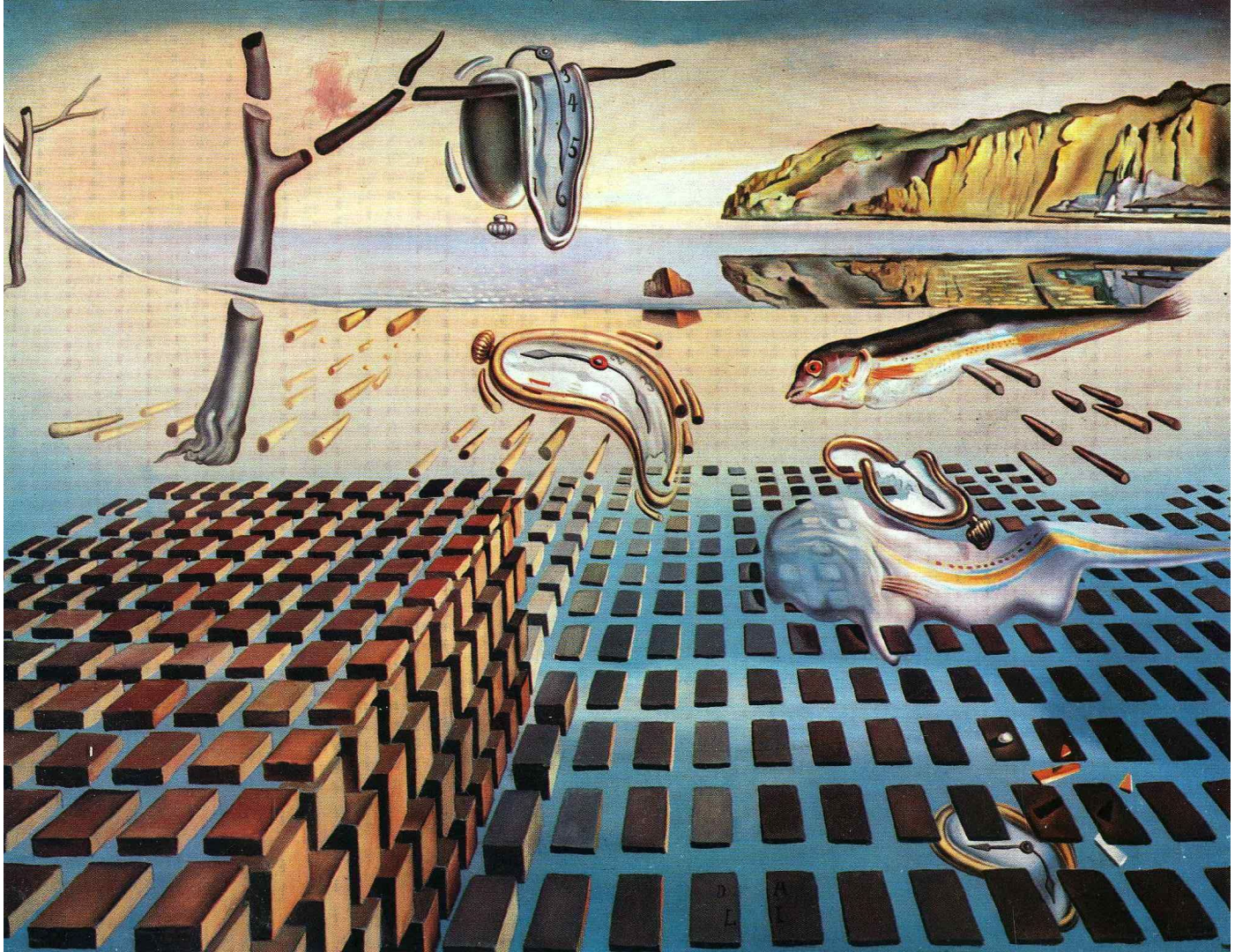


github.com/google/pprof



github.com/google/pprof

```
 flat  flat%   sum%        cum   cum%
9.77s 32.30% 32.30%     10.35s 34.21%  raw_hash_set::resize
9.69s 32.03% 64.33%     25.78s 85.22%  perftools_gwp::HashtableBenchmark::Run
5.88s 19.44% 83.77%     16.27s 53.79%  raw_hash_set::prepare_insert
3.25s 10.74% 94.51%      3.25s 10.74%  RandenHwAes::Generate
```



```
flat_hash_set() {
  if (absl::Random(absl::BitGenerator(), 0,
                   kSampleRate) == 0) {
    sample_info_ = HashtableSampler::Sample();
  }
}
```

```
SampleHandle Sample() {
  if (absl::Random(absl::BitGenerator(), 0, kSampleRate)) {
    return SampleHandle(nullptr);
  }
  return SampleHandle(new SampleInfo());
}

class flat_hash_set {
  SampleHandle sample_info_ = HashtableSampler::Sample();
}
```

```
SampleHandle Sample() {
  if (absl::Random(absl::BitGenerator(), 0, kSampleRate)) {
    return SampleHandle(nullptr);
  }
  return SampleHandle(new SampleInfo());
}
```

```
SampleHandle Sample() {
  static int64 next_sample = kSampleRate;
  if (ABSL_PREDICT_TRUE(--next_sample > 0)) {
    return SampleHandle(nullptr);
  }
  next_sample = kSampleRate;
  return SampleHandle(new SampleInfo());
}
```

```
SampleHandle Sample() {
  if (absl::Random(absl::BitGenerator(), 0, kSampleRate)) {
    return SampleHandle(nullptr);
  }
  return SampleHandle(new SampleInfo());
}
```

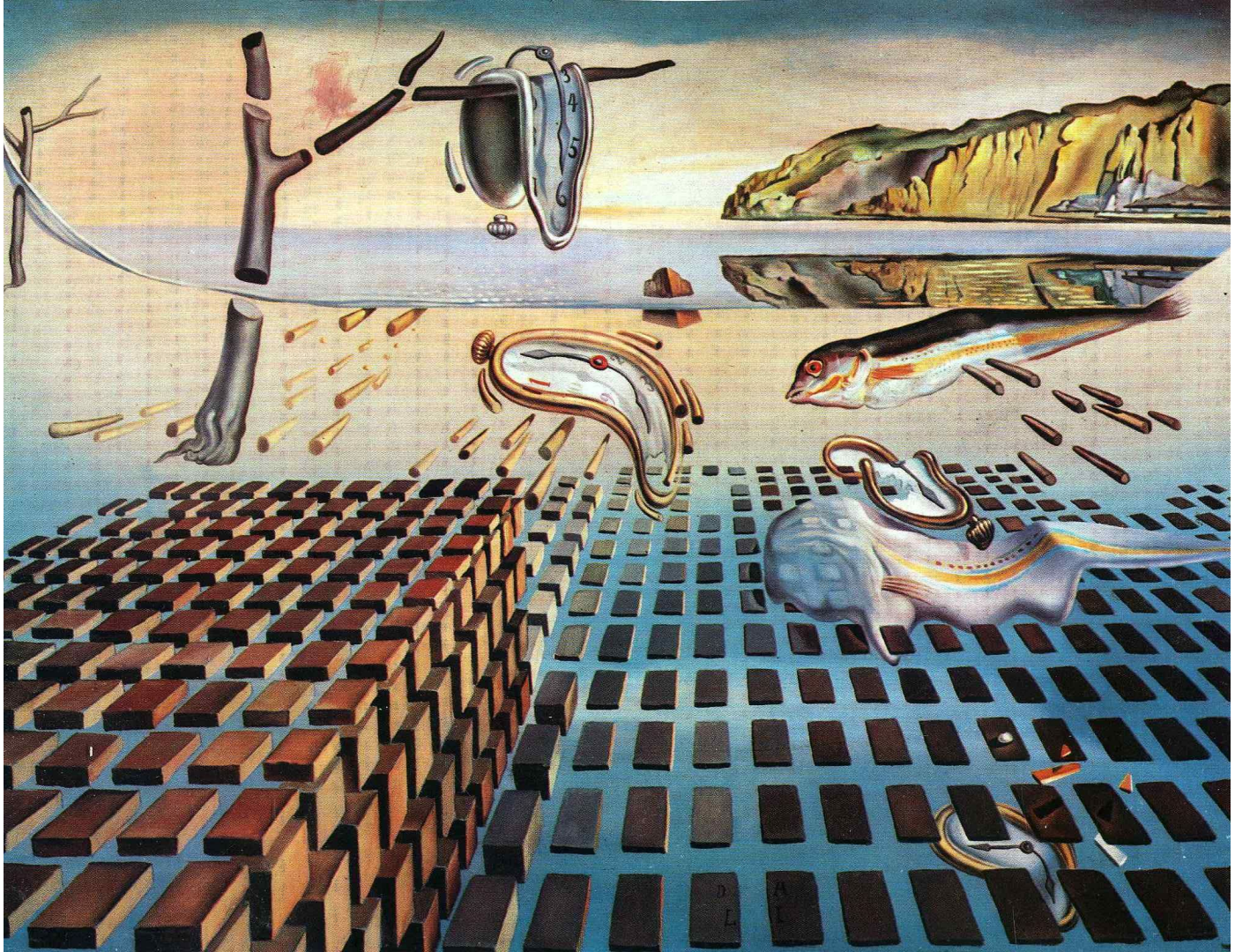
```
SampleHandle Sample() {
  if (absl::Bernoulli(absl::BitGenerator(), 1.0 / kSampleRate)) {
    return SampleHandle(nullptr);
  }
  return SampleHandle(new SampleInfo());
}
```

```
SampleHandle Sample() {
  thread_local int64 next_sample =
      GeometricDistribution(kSampleRate);
  if (ABSL_PREDICT_TRUE(--next_sample > 0)) {
    return SampleHandle(nullptr);
  }
  next_sample = GeometricDistribution(kSampleRate);
  return SampleHandle(new SampleInfo());
}
```

```
SampleHandle Sample() {
  thread_local int64 next_sample =
      GeometricDistribution(kSampleRate);
  if (ABSL_PREDICT_TRUE(--next_sample > 0)) {
    return SampleHandle(nullptr);
  }
  next_sample = GeometricDistribution(kSampleRate);
  return SampleHandle(new SampleInfo());
}
```

```
class flat_hash_set {
  SampleHandle sample_info_ = HashtableSampler::Sample();
}
```

```
void initialize_slots() {
  if (slots_ == nullptr) {
    sample_info_ = Sample();
  }

  auto layout = MakeLayout(capacity_);
  char* mem = layout.Allocate();
  ctrl_ = layout.template Pointer<0>(mem);
  slots_ = layout.template Pointer<1>(mem);
  reset_ctrl();
  reset_growth_left();
}
```



```
struct SampleInfo {
};
```

```
struct SampleInfo {
  size_t capacity;
};
```

```
struct SampleInfo {
  size_t capacity;
  size_t size;
};
```

```
struct SampleInfo {
  size_t capacity;
  size_t size;
  size_t num_erases;
  size_t max_probe_length;
  size_t total_probe_length;
};
```

```
struct SampleInfo {
  size_t capacity;
  size_t size;
  size_t num_erases;
  size_t max_probe_length;
  size_t total_probe_length;
  // good hash function?
};
```

```
struct SampleInfo {
  size_t capacity;
  size_t size;
  size_t num_erases;
  size_t max_probe_length;
  size_t total_probe_length;
  // all inserted hashes?
};
```

```
struct SampleInfo {
  size_t capacity;
  size_t size;
  size_t num_erases;
  size_t max_probe_length;
  size_t total_probe_length;
  size_t hashes_bitwise_or;
  size_t hashes_bitwise_and;
};
```

```
struct SampleInfo {
  size_t capacity;
  size_t size;
  size_t num_erases;
  size_t max_probe_length;
  size_t total_probe_length;
  size_t hashes_bitwise_or;   // -> 0xFFFFFFFF
  size_t hashes_bitwise_and;  // -> 0x00000000
};
```

```
struct SampleInfo {
  std::atomic<size_t> capacity;
  std::atomic<size_t> size;
  std::atomic<size_t> num_erases;
  std::atomic<size_t> max_probe_length;
  std::atomic<size_t> total_probe_length;
  std::atomic<size_t> hashes_bitwise_or;
  std::atomic<size_t> hashes_bitwise_and;
};
```

```
struct SampleInfo {
  std::atomic<size_t> capacity;
  std::atomic<size_t> size;
  std::atomic<size_t> num_erases;
  std::atomic<size_t> max_probe_length;
  std::atomic<size_t> total_probe_length;
  std::atomic<size_t> hashes_bitwise_or;
  std::atomic<size_t> hashes_bitwise_and;
  int32_t depth;
  void* stack[kMaxStackDepth];
};
```


PETZL

```
void DoIt(absl::flat_hash_set<int>& s) {
  s.insert(1);
}
```

```
template <typename T>
using custom_set = absl::flat_hash_set<T,
    absl::flat_hash_set<T>::hasher,
    absl::flat_hash_set<T>::key_equal,
    zero_malloc_allocator<T>>;


void DoIt(custom_set<int>& s) {
  s.insert(1);
}
```

```
void DoIt(absl::flat_hash_set<int>& s) {
  s.insert(1);
}
```

```
template <typename T>
using custom_set = absl::flat_hash_set<T,
    absl::flat_hash_set<T>::hasher,
    absl::flat_hash_set<T>::key_equal,
    zero_malloc_allocator<T>>;


void DoIt(custom_set<int>& s) {
  s.insert(1);
}
```

```
template <typename T>
using custom_set = absl::node_hash_set<T,
    std::hash<T>, std::equal_to<T>,
    arena_allocator>;

void DoIt() {
  char* buffer = new char[1024];
  arena_allocator a(buffer);
  char* space = a.allocate(sizeof(custom_set<int>));
  custom_set<int>* s = new (space) custom_set<int>(a);
  s->insert(1);
  delete[] buffer;
}
```


PETZL

```
template <typename T>
using custom_set = absl::node_hash_set<T,
    std::hash<T>, std::equal_to<T>,
    arena_allocator>;

void DoIt() {
  char* buffer = new char[1024];
  arena_allocator a(buffer);
  char* space = a.allocate(sizeof(custom_set<int>));
  custom_set<int>* s = new (space) custom_set<int>(a);
  s->insert(1);
  delete[] buffer;
}
```

```
template <typename T>
using custom_set = absl::node_hash_set<T,
    std::hash<T>, std::equal_to<T>,
    arena_allocator>;

void DoIt() {
  char* buffer = new char[1024];
  arena_allocator a(buffer);
  char* space = a.allocate(sizeof(custom_set<int>));
  custom_set<int>* s = new (space) custom_set<int>(a);
  s->insert(1);
  delete[] buffer;
}
```

```
void DoIt() {
  absl::flat_hash_set<int, std::hash<int>, std::equal_to<int>> s;
  s.insert(1);
}
```

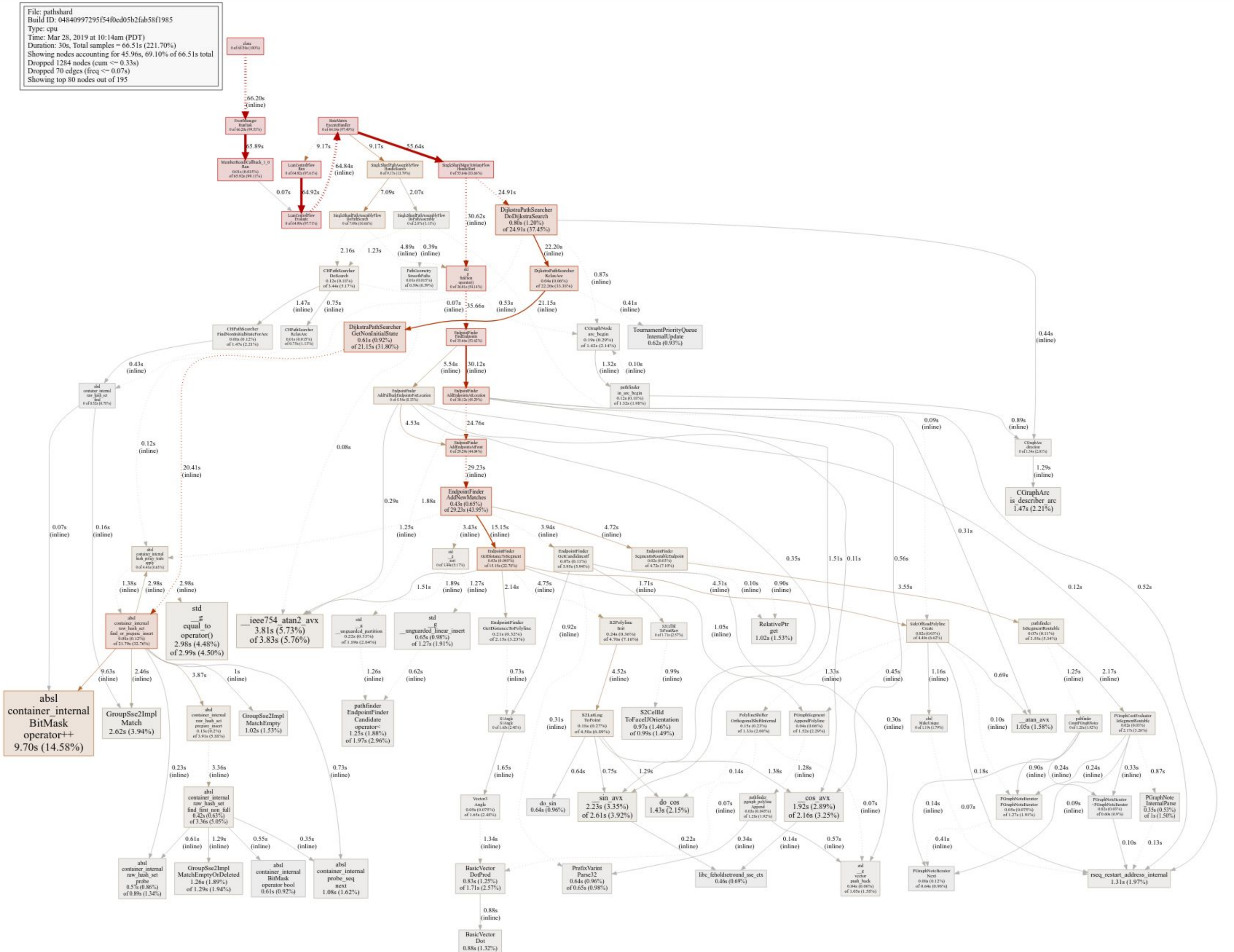
```
void DoIt() {
  absl::flat_hash_set<int> s;
  s.insert(1);
}
```

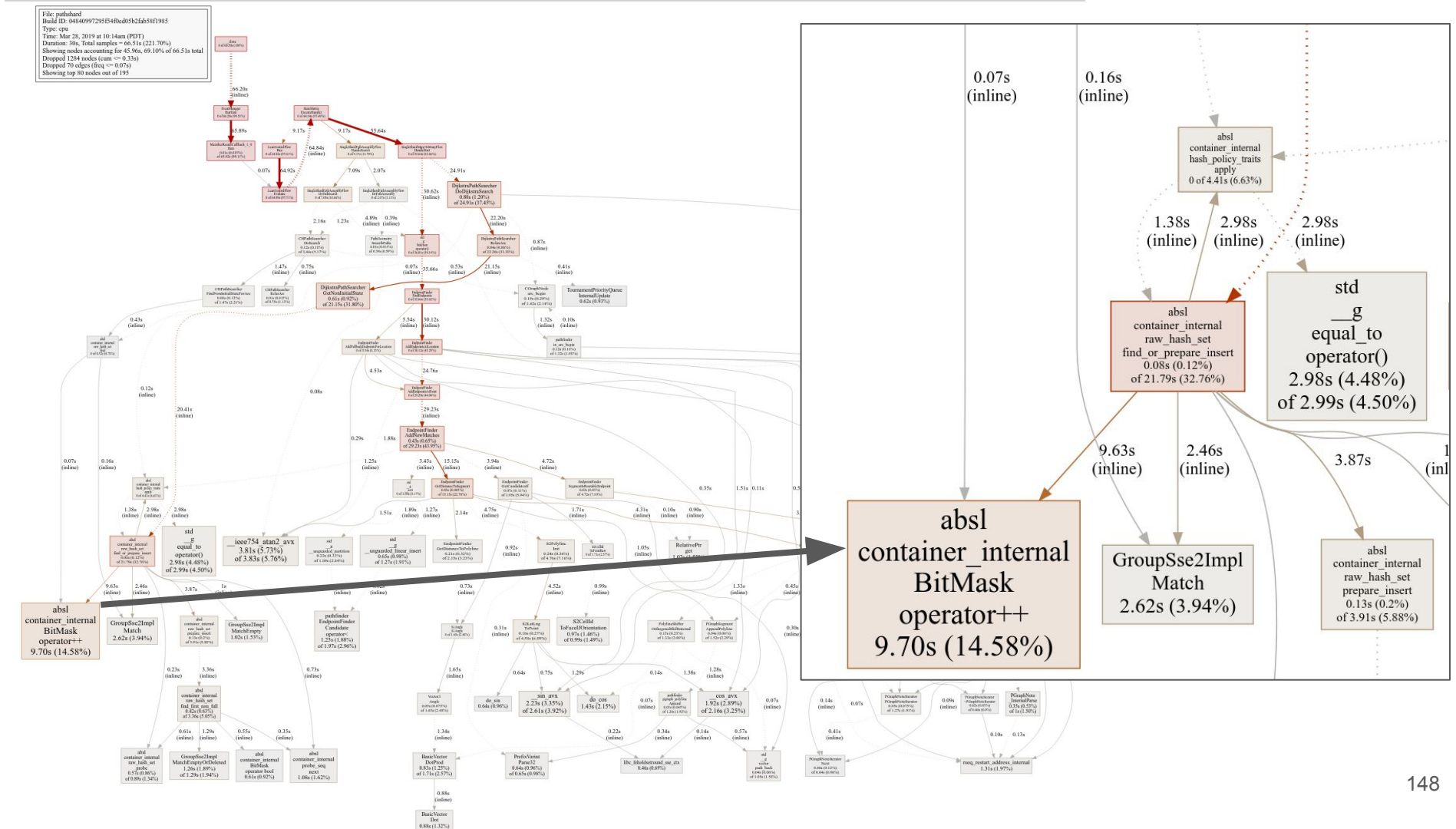
```
template <typename T>
using custom_set = absl::node_hash_set<T,
    std::hash<T>, std::equal_to<T>,
    arena_allocator>;

void DoIt() {
  char* buffer = new char[1024];
  arena_allocator a(buffer);
  char* space = a.allocate(si
  custom_set<int>* s = new (s
  s->insert(1);
  delete[] buffer;
}
```

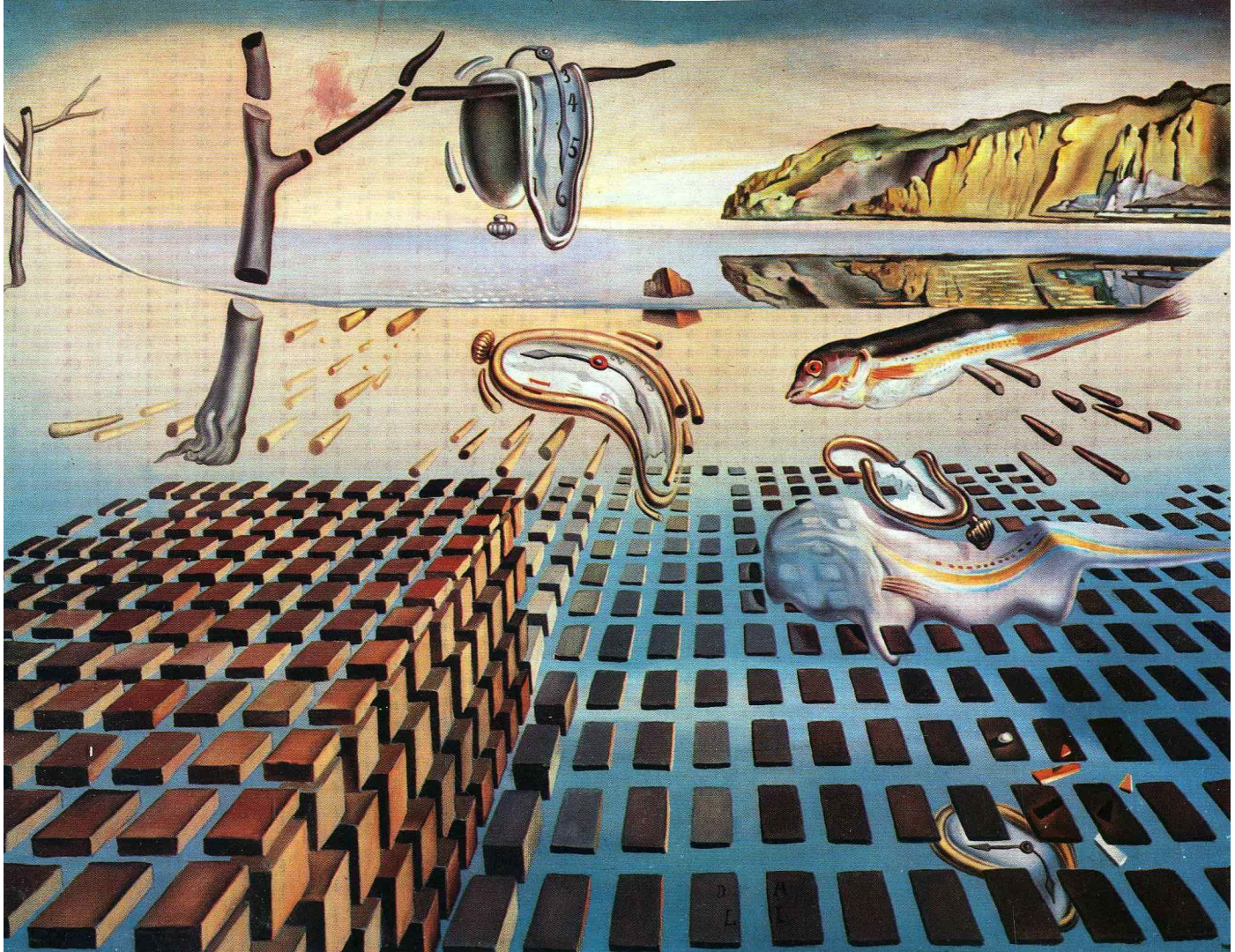
```
void DoIt() {
  absl::flat_hash_set<int> s;
  s.insert(1);
}
```

Can anyone claim credit for rescuing Japan?


健康こみち
禁止
くつをぬいで
ご使用ください
スタート


File: pathshard
Build ID: 04840997295f54f0ed05b2fab58f1985
Type: cpu
Time: Mar 28, 2019 at 10:14am (PDT)
Duration: 30s, Total samples = 66.51s (221.70%)
Showing nodes accounting for 45.96s, 69.10% of 66.51s total
Dropped 1284 nodes (cum <= 0.33s)
Dropped 70 edges (freq <= 0.07s)
Showing top 80 nodes out of 195
DijkstraPathSearcher DoDijkstraSearch 0.80s (1.20%) of 24.91s (37.45%)
DijkstraPathSearcher GetNonInitialState 0.61s (0.92%) of 21.15s (31.80%)
TournamentPriorityQueue InternalUpdate 0.62s (0.93%)
EndpointFinder AddNewMatches 0.43s (0.65%) of 29.23s (43.95%)
CGraphArc is_describer_arc 1.47s (2.21%)
std __g equal_to operator() 2.98s (4.48%) of 2.99s (4.50%)
__ieee754_atan2_avx 3.81s (5.73%) of 3.83s (5.76%)
std __g __unguarded_linear_insert 0.65s (0.98%) of 1.27s (1.91%)
S2Polyline Init 0.24s (0.36%) of 4.76s (7.16%)
RelativePtr get 1.02s (1.53%)
absl container_internal BitMask operator++ 9.70s (14.58%)
GroupSse2Impl Match 2.62s (3.94%)
GroupSse2Impl MatchEmpty 1.02s (1.53%)
pathfinder EndpointFinder Candidate operator< 1.25s (1.88%) of 1.97s (2.96%)
S2CellId ToFaceIJOrientation 0.97s (1.46%) of 0.99s (1.49%)
__atan_avx 1.05s (1.58%)
do_sin 0.64s (0.96%)
__sin_avx 2.23s (3.35%) of 2.61s (3.92%)
do_cos 1.43s (2.15%)
__cos_avx 1.92s (2.89%) of 2.16s (3.25%)
PGraphNote InternalParse 0.35s (0.53%) of 1s (1.50%)
absl container_internal raw_hash_set find_first_non_full 0.42s (0.63%) of 3.36s (5.05%)
GroupSse2Impl MatchEmptyOrDeleted 1.26s (1.89%) of 1.29s (1.94%)
absl container_internal BitMask operator bool 0.61s (0.92%)
absl container_internal probe_seq next 1.08s (1.62%)
BasicVector DotProd 0.83s (1.25%) of 1.71s (2.57%)
PrefixVarint Parse32 0.64s (0.96%) of 0.65s (0.98%)
libc_feholdsetround_sse_ctx 0.46s (0.69%)
rseq_restart_address_internal 1.31s (1.97%)
BasicVector Dot 0.88s (1.32%)


File: pathsbard
Build ID: 04840997295f54f8ed05b2fab58f1985
Type: cpu
Time: Mar 28, 2019 at 10:14am (PDT)
Duration: 30s, Total samples = 66.51s (221.70%)
Showing nodes accounting for 45.96s, 69.10% of 66.51s total
Dropped 1284 nodes (cum <= 0.33s)
Dropped 70 edges (freq <= 0.07s)
Showing top 80 nodes out of 195
0.07s (inline)
0.16s (inline)
absl container_internal hash_policy_traits apply 0 of 4.41s (6.63%)
1.38s (inline)
2.98s (inline)
2.98s (inline)
absl container_internal raw_hash_set find_or_prepare_insert 0.08s (0.12%) of 21.79s (32.76%)
std __g equal_to operator() 2.98s (4.48%) of 2.99s (4.50%)
9.63s (inline)
2.46s (inline)
3.87s
absl container_internal BitMask operator++ 9.70s (14.58%)
GroupSse2Impl Match 2.62s (3.94%)
absl container_internal raw_hash_set prepare_insert 0.13s (0.2%) of 3.91s (5.88%)
absl container_internal BitMask operator++ 9.70s (14.58%)
GroupSse2Impl Match 2.62s (3.94%)
std __g equal_to operator() 2.98s (4.48%) of 2.99s (4.50%)
ieee754_atan2_avx 3.81s (5.73%) of 3.83s (5.76%)
std __unguarded_linear_insert 0.65s (0.98%) of 1.27s (1.91%)
pathfinder EndpointFinder Candidate operator< 1.25s (1.88%) of 1.97s (2.96%)
S2CellId ToFaceIJOrientation 0.97s (1.46%) of 0.99s (1.49%)
sin_avx 2.23s (3.35%) of 2.61s (3.92%)
do_sin 0.64s (0.96%)
do_cos 1.43s (2.15%)
cos_avx 1.92s (2.89%) of 2.16s (3.25%)
BasicVector Dot 0.88s (1.32%)
PrefixVarint Parse32 0.64s (0.96%) of 0.65s (0.98%)
req_restart_address_internal 1.31s (1.97%)



```
(pprof) top 4
      flat  flat%   sum%        cum   cum%
      6997 95.63% 95.63%       6997 95.63%  perftools_gwp::HashtableBenchmark::Run
       127  1.74% 97.36%        127  1.74%  stats_census::store::TagQuota::TagQuota
       126  1.72% 99.08%        126  1.72%  proto2::FileDescriptorTables::FileDescriptorTables
         0     0% 99.08%        127  1.74%  (anonymous namespace)::GetDefaultUserName
```

```
(pprof) top 4
      flat  flat%   sum%        cum   cum%
     15466 98.32% 98.32%      15466 98.32%  perftools_gwp::HashtableBenchmark::Run
       258  1.64%   100%        258  1.64%  proto2::FileDescriptorTables::FileDescriptorTables
         0     0%   100%        261  1.66%  (anonymous namespace)::GetDefaultUserName
         0     0%   100%        261  1.66%  (anonymous namespace)::GetDefaultUserNameNonBlocking
```

```
(pprof) tags
 max_probe_length: Total 7124.0
                   2557.0 (35.89%): 20
                   1023.0 (14.36%): 21
                   1023.0 (14.36%): 22
                   1023.0 (14.36%): 31
                    511.0 ( 7.17%): 11
                    511.0 ( 7.17%): 12
                    255.0 ( 3.58%): 6
                    158.0 ( 2.22%): 1
                     63.0 ( 0.88%): 2
...
```

```
(pprof) tags
 max_probe_length: Total 15594.0
                   8182.0 (52.47%): 3
                   5111.0 (32.78%): 2
                   1023.0 ( 6.56%): 4
                   1023.0 ( 6.56%): 6
                    255.0 ( 1.64%): 1
...
```

```
(pprof) tagfocus=max_probe_length=4count:
(pprof) top 1
      flat  flat%   sum%        cum   cum%
      6903 94.34% 94.34%       6903 94.34%  perftools_gwp::HashtableBenchmark::Run
```

```
(pprof) tagfocus=max_probe_length=4count:
(pprof) top 1
      flat  flat%   sum%        cum   cum%
      2046 13.01% 13.01%       2046 13.01%  perftools_gwp::HashtableBenchmark::Run
```

```
(pprof) tagfocus=max_probe_length=4count:
(pprof) tags
 max_probe_length: Total 6903.0
    ...

 size: Total 6903.0
    ...

 stuck_bits: Total 6903.0
             6903.0 (  100%): -2048
```

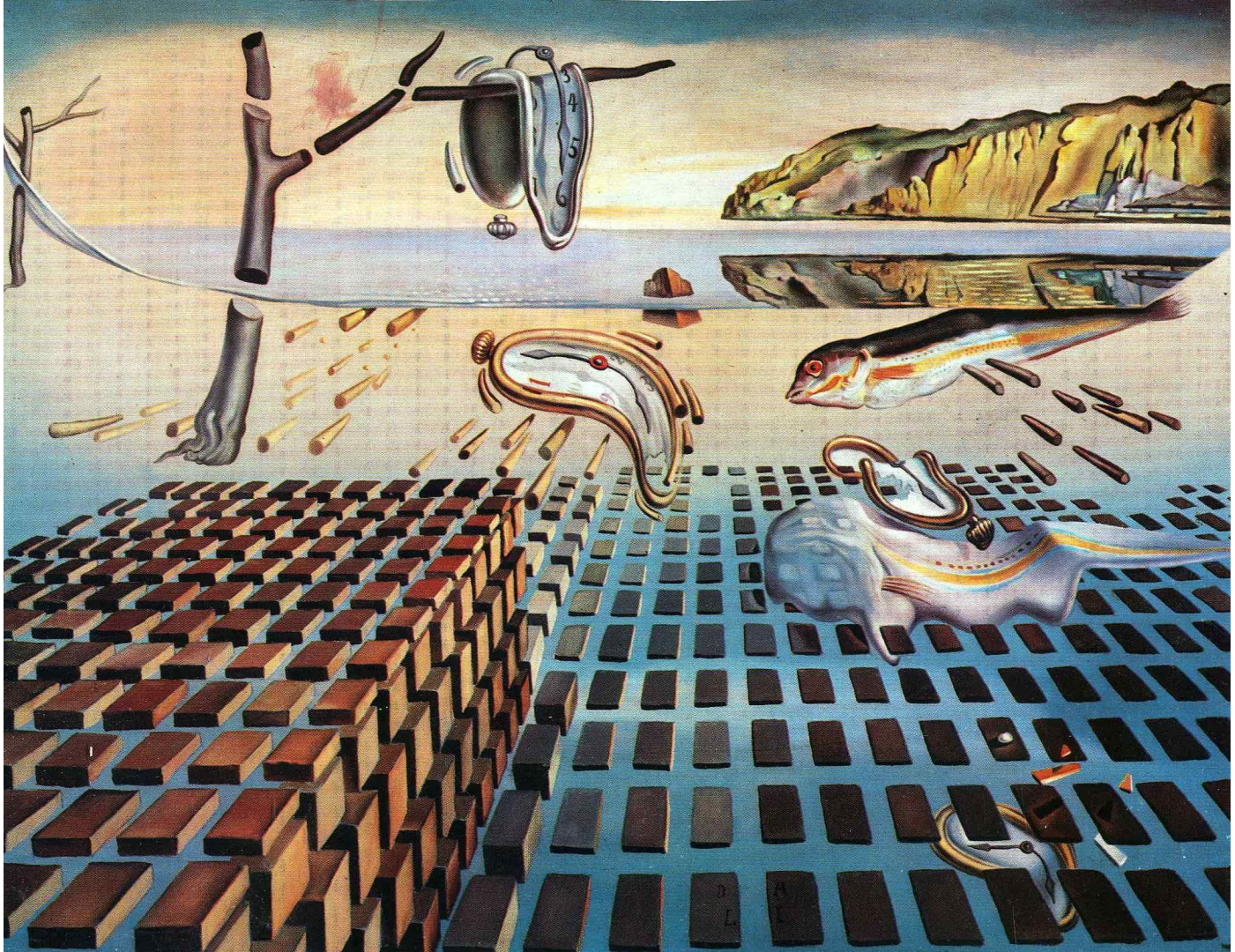
```
(pprof) tagfocus=max_probe_length=4count:
(pprof) tags
 max_probe_length: Total 2046.0
    ...

 size: Total 2046.0
    ...
```

```
(pprof) tagfocus=max_probe_length=4count:
(pprof) tags
 max_probe_length: Total 6903.0
    ...

 size: Total 6903.0
    ...

 stuck_bits: Total 6903.0
             6903.0 (  100%): -2048
```

```
(pprof) tagfocus=max_probe_length=4count:
(pprof) tags
 max_probe_length: Total 2046.0
    ...

 size: Total 2046.0
    ...
```

```
(pprof) tagfocus=max_probe_length=4count:
(pprof) tags
 max_probe_length: Total 6903.0
    ...

 size: Total 6903.0
    ...

 stuck_bits: Total 6903.0
             6903.0 (  100%): -2048
```


```
(pprof) tagfocus=max_probe_length=4count:
(pprof) tags
 max_probe_length: Total 2046.0
    ...

 size: Total 2046.0
    ...
```



Questions?