Cache-friendly Design in Robot Path Planning

BRIAN CAIR

2023 | October

About Me

- This is my first time at CppCon!
- Regularly program in C++ for robotics

About Me

I've worked at a few robotics startups:

- Fetch Robotics (2016-2022)
- Thirdwave.ai (2022-2023)
- Tortuga AgTech (present)

About Me

Mostly doing:

- General architecture and frameworks around robot systems
- Navigation and perception systems

About Me

Other interests:

- Game engines
- Meta-programming

Robot path planning

Robot path planning

Path planning

Path planning

How an (autonomous) agent figures out how to get from one location to another before actually moving.

Cache-friendly design

Cache-friendly design

Program design focused on optimizing code by avoiding pathological affects of memory access through the caching structure of a computer system.

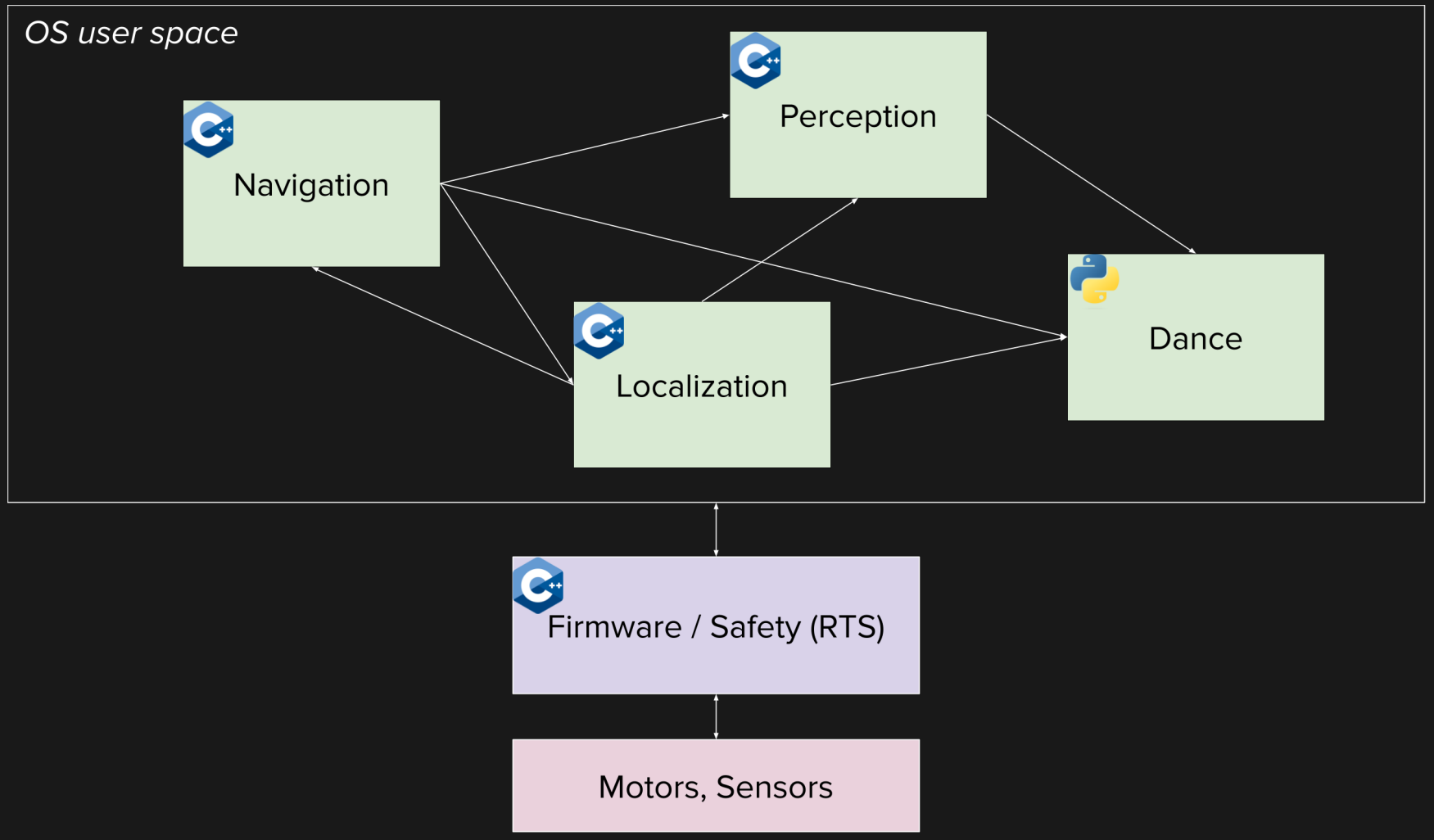
Cache-friendly design

> *Modern memory pipelines are so complex you are basically optimizing for the cache*

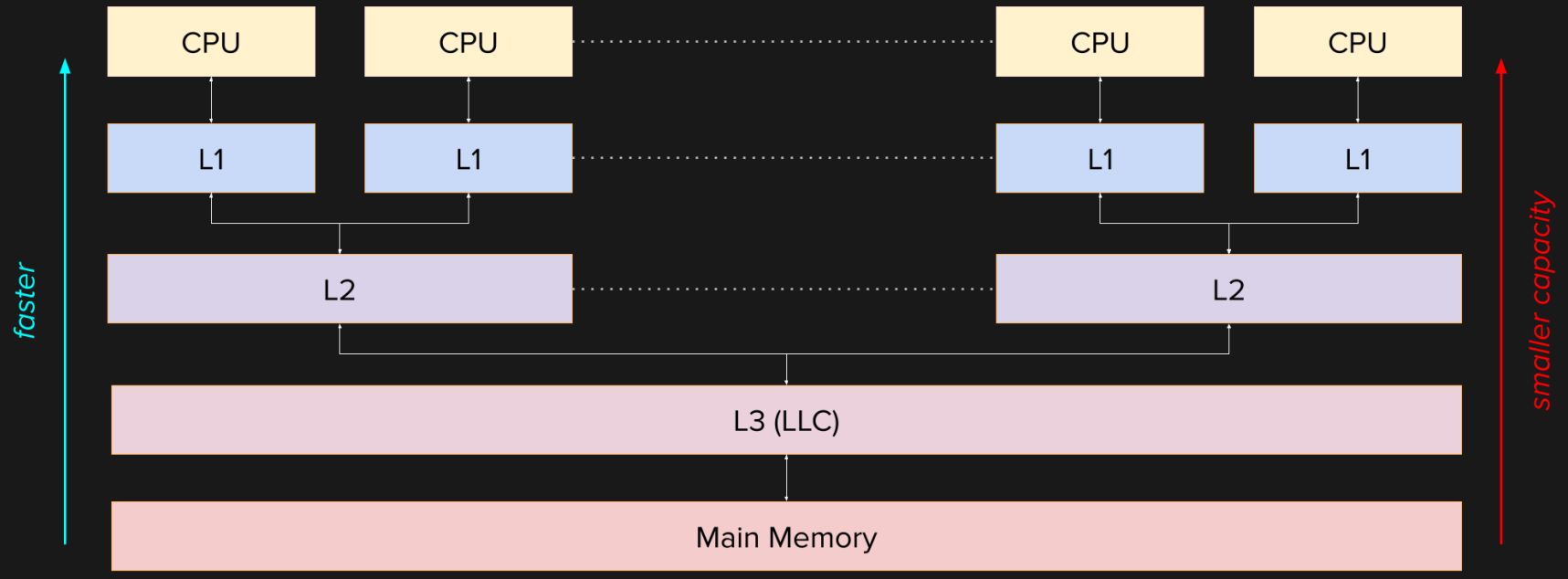
\- Random person on StackOverflow

Cache-friendly design

Cache-friendly design



Cache-friendly design



Cache-friendly design

- Accessing data means accessing memory through the cache

Cache-friendly design

- Accessing data means accessing memory through the cache
- Running instructions means accessing memory through the cache

Cache-friendly design

- Accessing data means accessing memory through the cache
- Running instructions means accessing memory through the cache
- Besides the data the CPU needs now, the cache is populated with data likely to be accessed soon (pre-fetching)

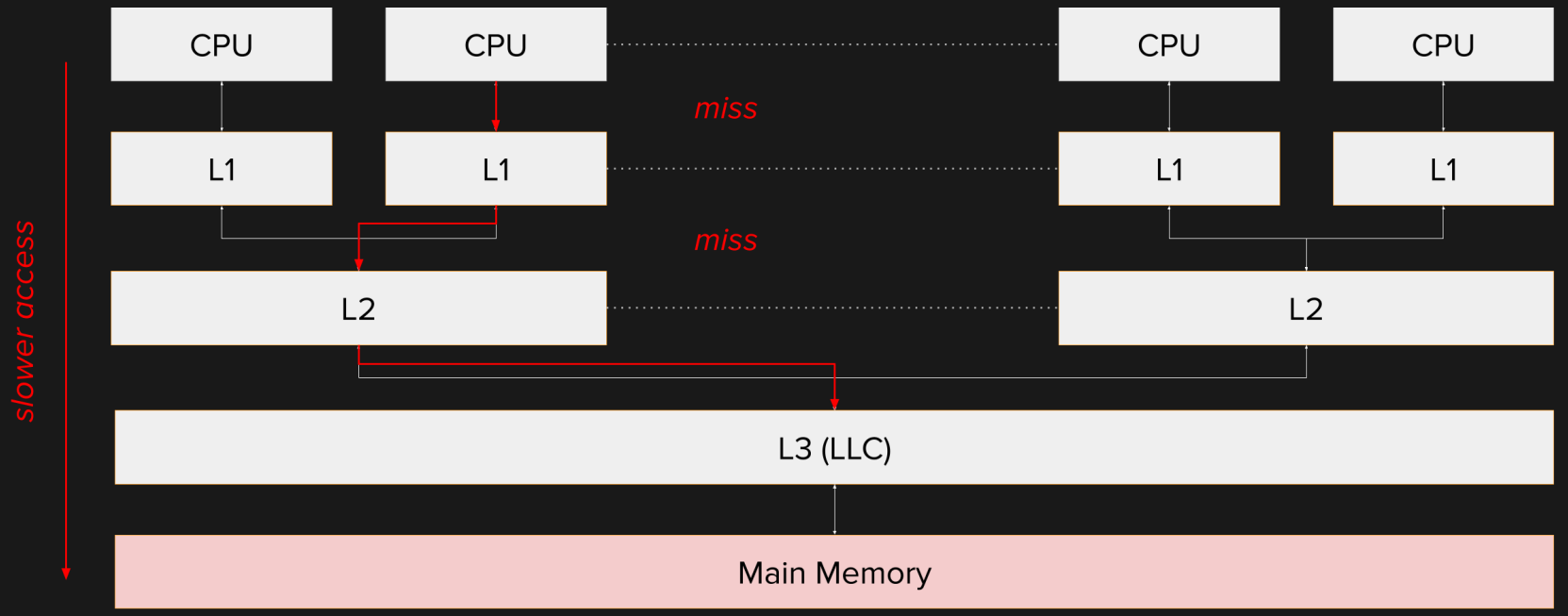
Cache-friendly design

- Accessing data means accessing memory through the cache
- Running instructions means accessing memory through the cache
- Besides the data the CPU needs now, the cache is populated with data likely to be accessed soon (pre-fetching)
- Programs may attempt to access data that is not in the cache or has been removed (evicted) from the cache

Cache-friendly design

- Accessing data means accessing memory through the cache
- Running instructions means accessing memory through the cache
- Besides the data the CPU needs now, the cache is populated with data likely to be accessed soon (pre-fetching)
- Programs may attempt to access data that is not in the cache or has been removed (evicted) from the cache
- If a program attempts to access something that not in the cache, this is a *cache miss*

Cache-friendly design



Cache-friendly design

- Cache misses introduce latency

Cache-friendly design

- Cache misses introduce latency
- Access through each level is slower than the previous level

Cache-friendly design

- Cache misses introduce latency
- Access through each level is slower than the previous level
- CPUs can run out of things to do while waiting for the next cache line from memory

Cache-friendly design

- Cache misses introduce latency
- Access through each level is slower than the previous level
- CPUs can run out of things to do while waiting for the next cache line from memory
- This is called a CPU stall

Cache-friendly design

- Cache misses introduce latency
- Access through each level is slower than the previous level
- CPUs can run out of things to do while waiting for the next cache line from memory
- This is called a CPU stall
- There are mitigations, e.g. out-of-order execution, hyperthreading

Cache-friendly design

Optimizes how a program uses data already in the cache.

Cache-friendly design

Minimizes cache misses!

High level goals of this talk

High level goals of this talk

- Implement cache-friendly path planning code in C++

High level goals of this talk

- Implement cache-friendly path planning code in C++
- Do so using the STL

High level goals of this talk

- Implement cache-friendly path planning code in C++
- Do so using the STL
- Try to keep our choices simple

High level goals of this talk

- Implement cache-friendly path planning code in C++
- Do so using the STL
- Try to keep our choices simple
- Measure

The motivating problem

The motivating problem

Robots typically need to move.

The motivating problem

Mobile Robot Systems

The motivating problem

Mobile Robot Systems

- Warehouse robots: forklifts, pallet-movers, etc.

The motivating problem

Mobile Robot Systems

- Warehouse robots: forklifts, pallet-movers, etc.
- Home robots: vacuums, etc.

The motivating problem

Mobile Robot Systems

- Warehouse robots: forklifts, pallet-movers, etc.
- Home robots: vacuums, etc.
- Outdoor robots: tractors, berry-pickers, delivery platforms, etc.

The motivating problem

Mobile Robot Systems

- These robots operate in highly dynamic environments.

The motivating problem

Mobile Robot Systems

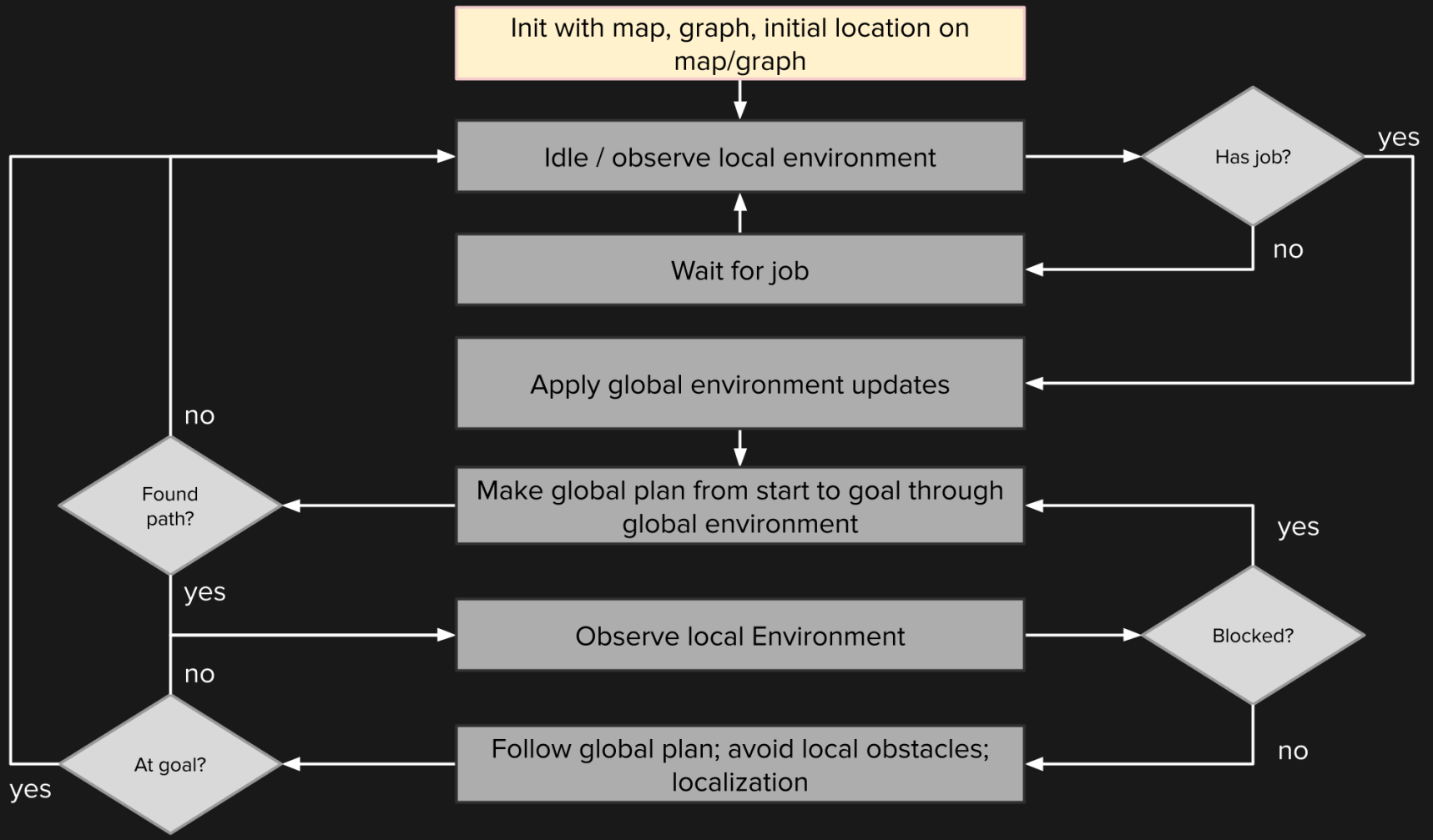
- These robots operate in highly dynamic environments.
- Robots might be informed about blockages via local observations or some centralized coordination system.

The motivating problem

Mobile Robot Systems

- These robots operate in highly dynamic environments.
- Robots might be informed about blockages via local observations or some centralized coordination system.
- A robot might need to figure out how to move multiple times while making its way to a goal.

The motivating problem



The motivating problem

- Figuring out how to move is computationally expensive.

The motivating problem

- Figuring out how to move is computationally expensive.
- This only gets worse as the state-space of the problem gets bigger.

The motivating problem

- Figuring out how to move is computationally expensive.
- This only gets worse as the state-space of the problem gets bigger.
- We are concerned about computation time as it affects overall system latency stack-up.

The motivating problem

- Figuring out how to move is computationally expensive.
- This only gets worse as the state-space of the problem gets bigger.
- We are concerned about computation time as it affects overall system latency stack-up.
- When we tell a robot to do something, we want it moving quickly.

The motivating problem

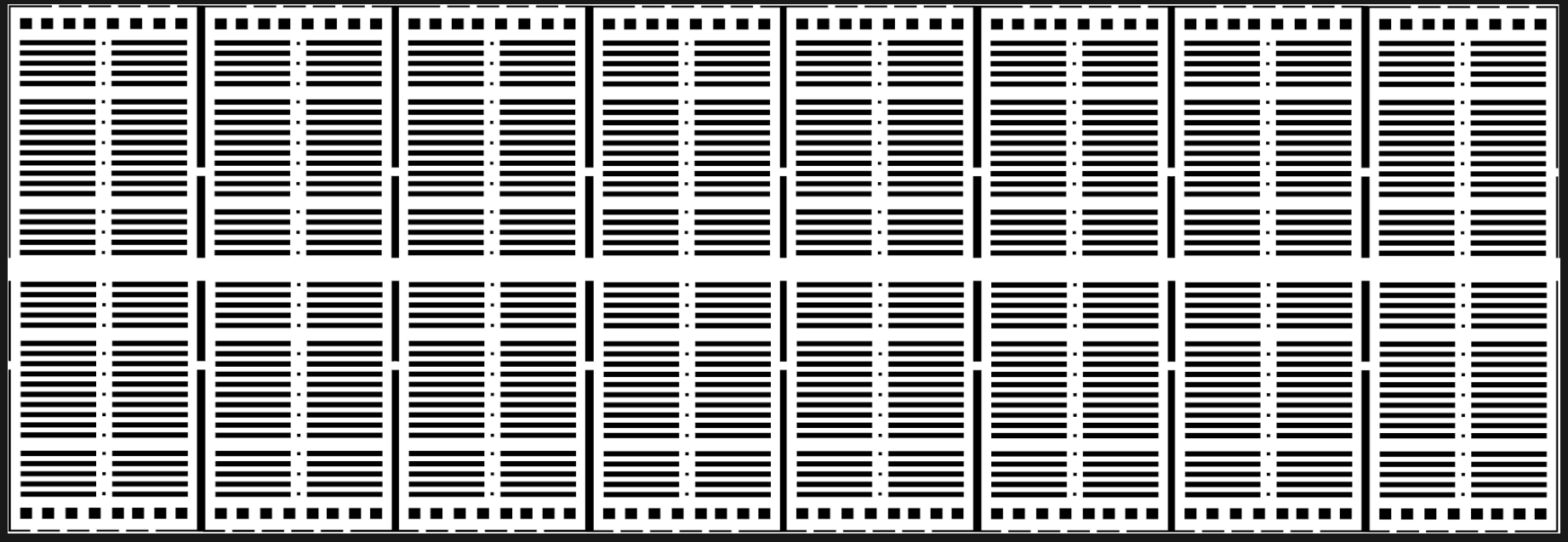
- Figuring out how to move is computationally expensive.
- This only gets worse as the state-space of the problem gets bigger.
- We are concerned about computation time as it affects overall system latency stack-up.
- When we tell a robot to do something, we want it moving quickly.
- Fast planning algorithms == less time robots are sitting around.

The operating environment

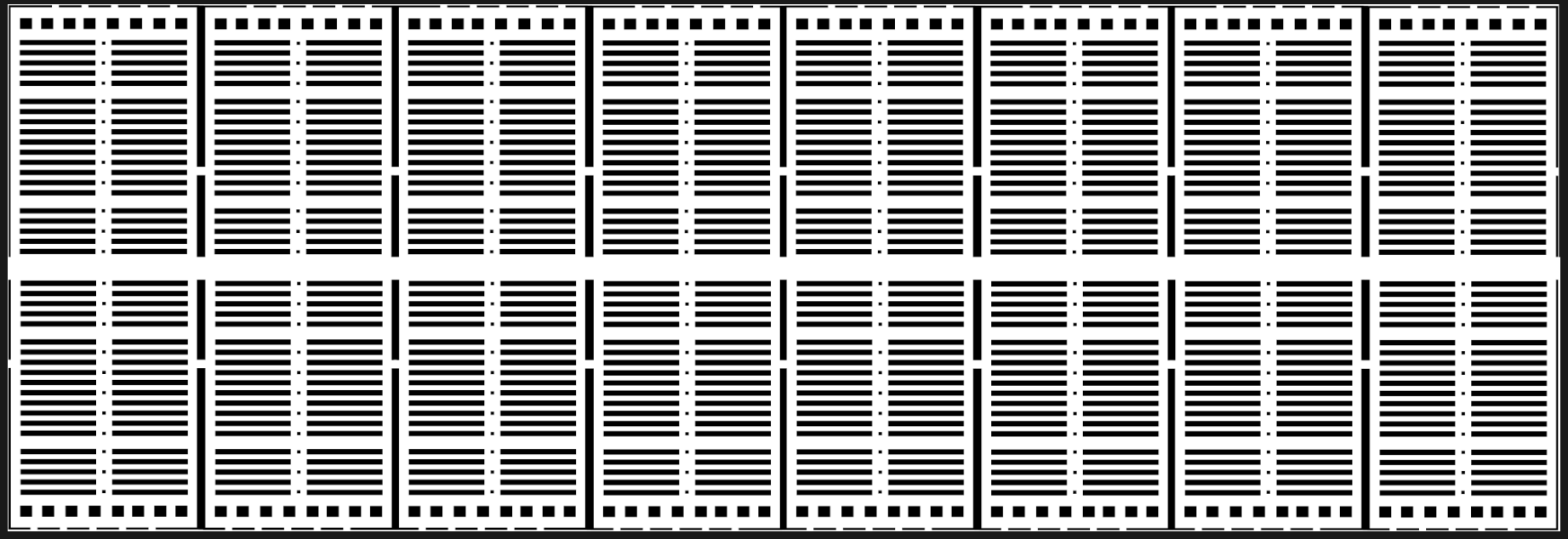
The operating environment

Robots operating in a very big warehouse.

The operating environment



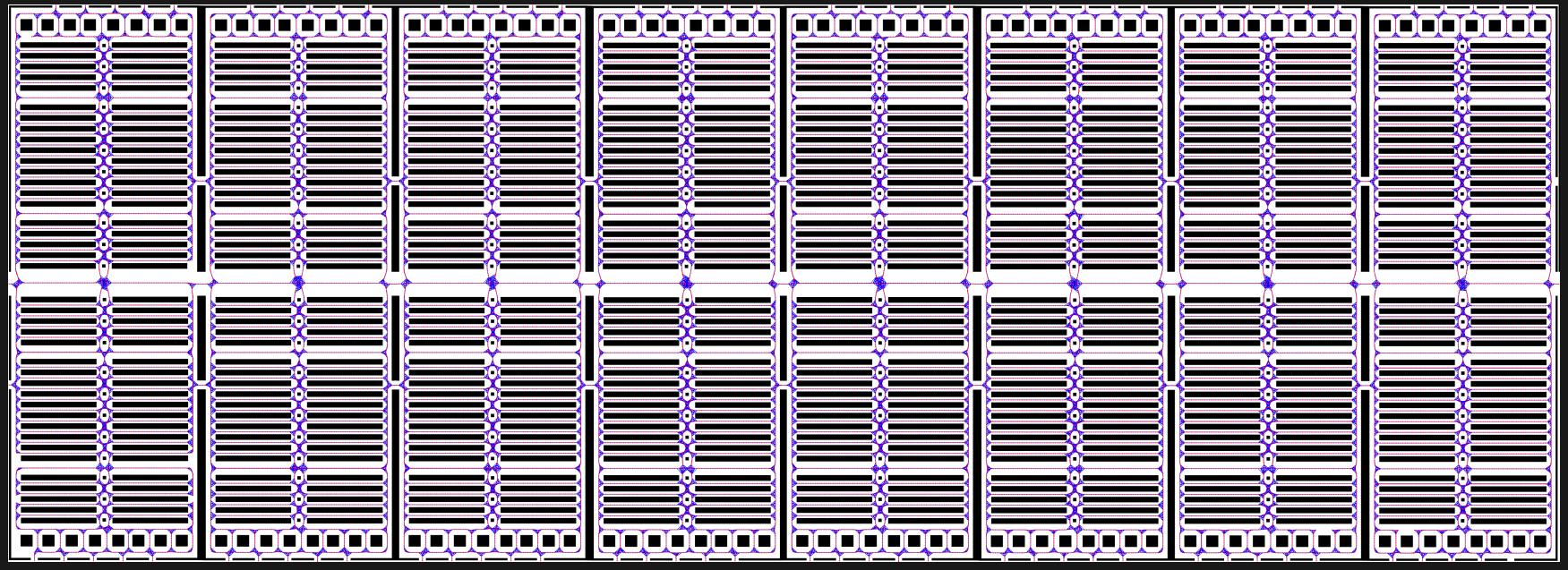
The operating environment



This is a map of a make-believe distribution facility that is:

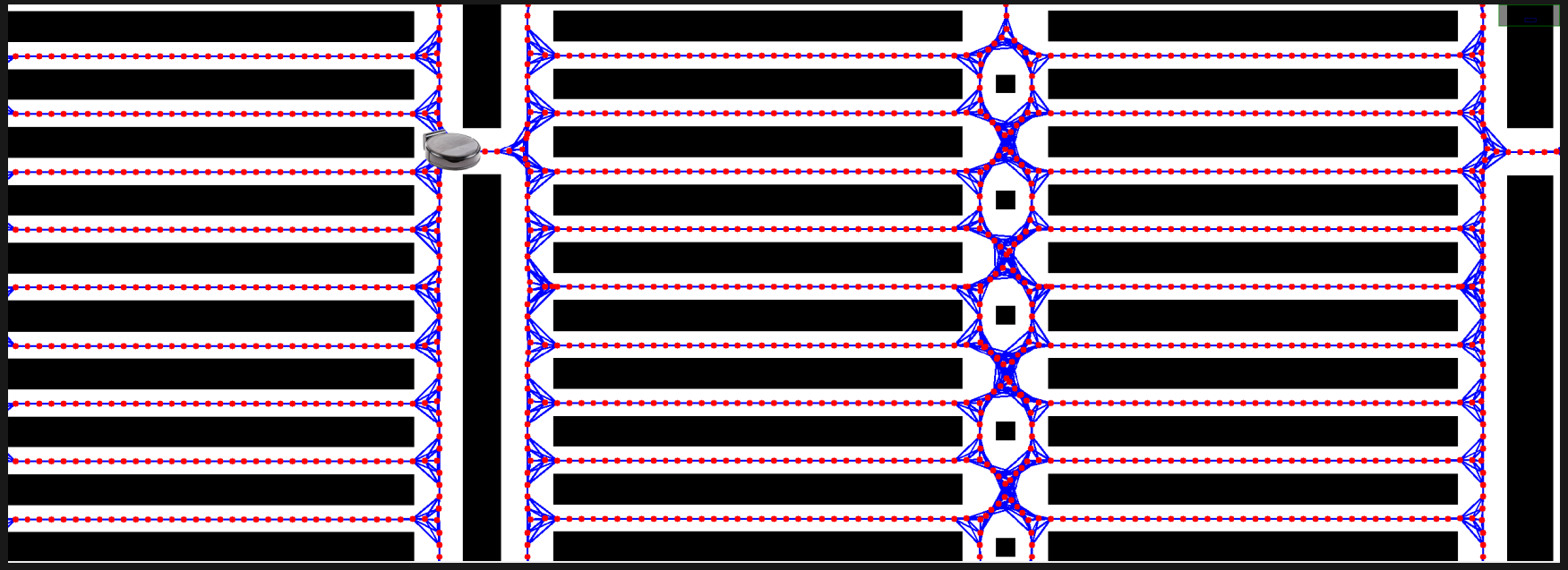
- 0.8 km long
- 0.4 km deep

The operating environment



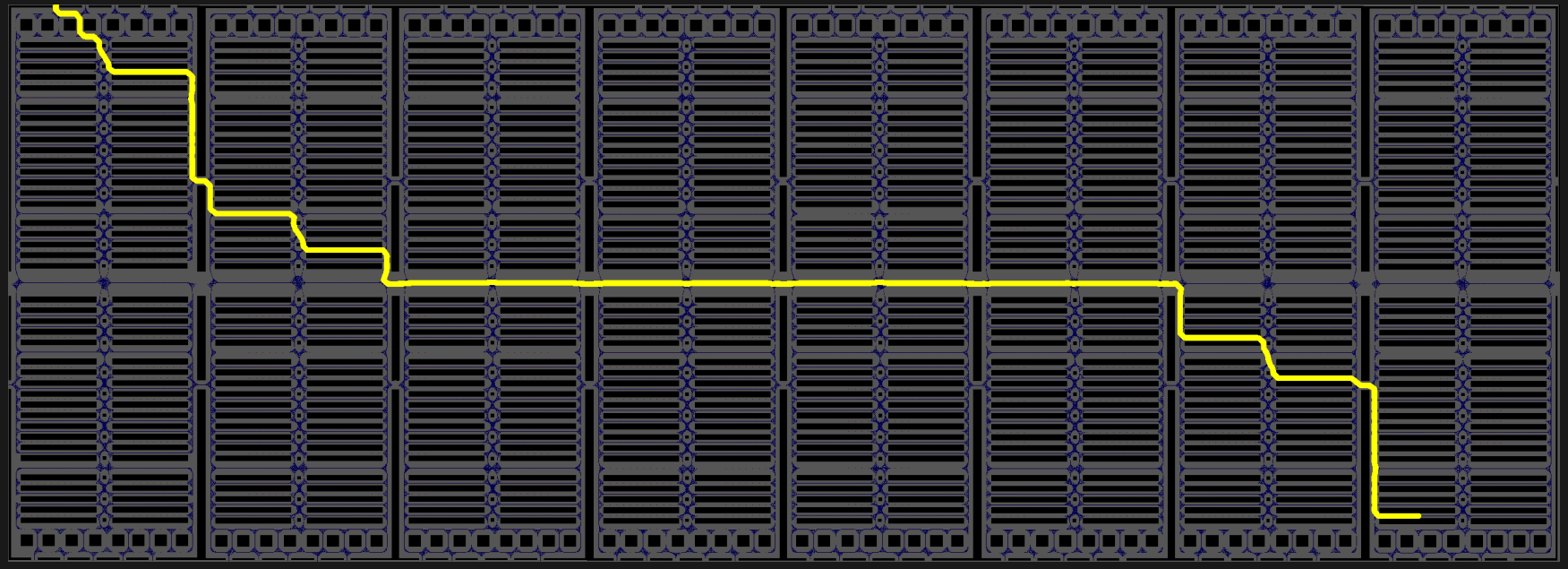
This is a spatial graph extracted from the map.

The operating environment



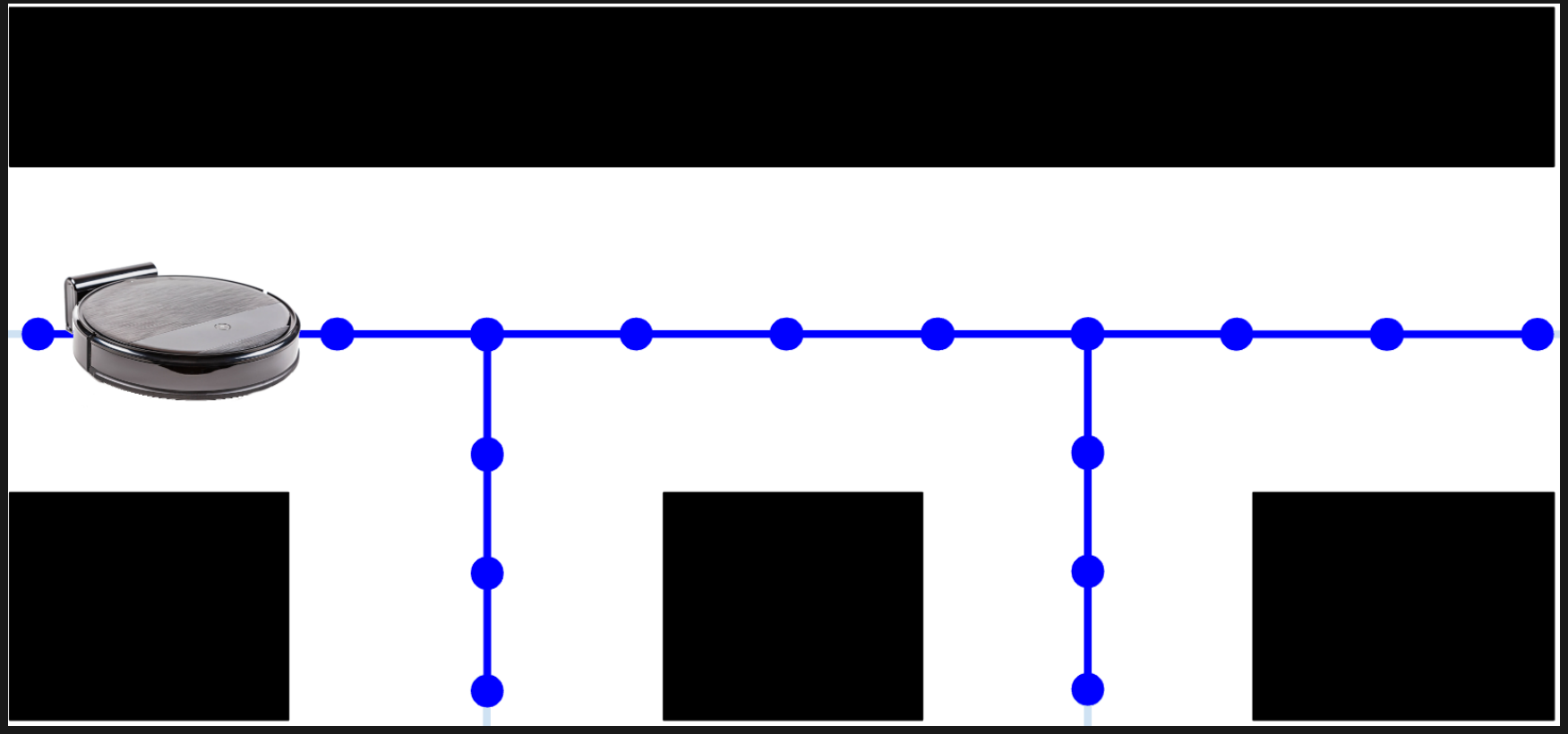
This is a spatial graph informs the robot about good "roadways" to travel along.

The operating environment

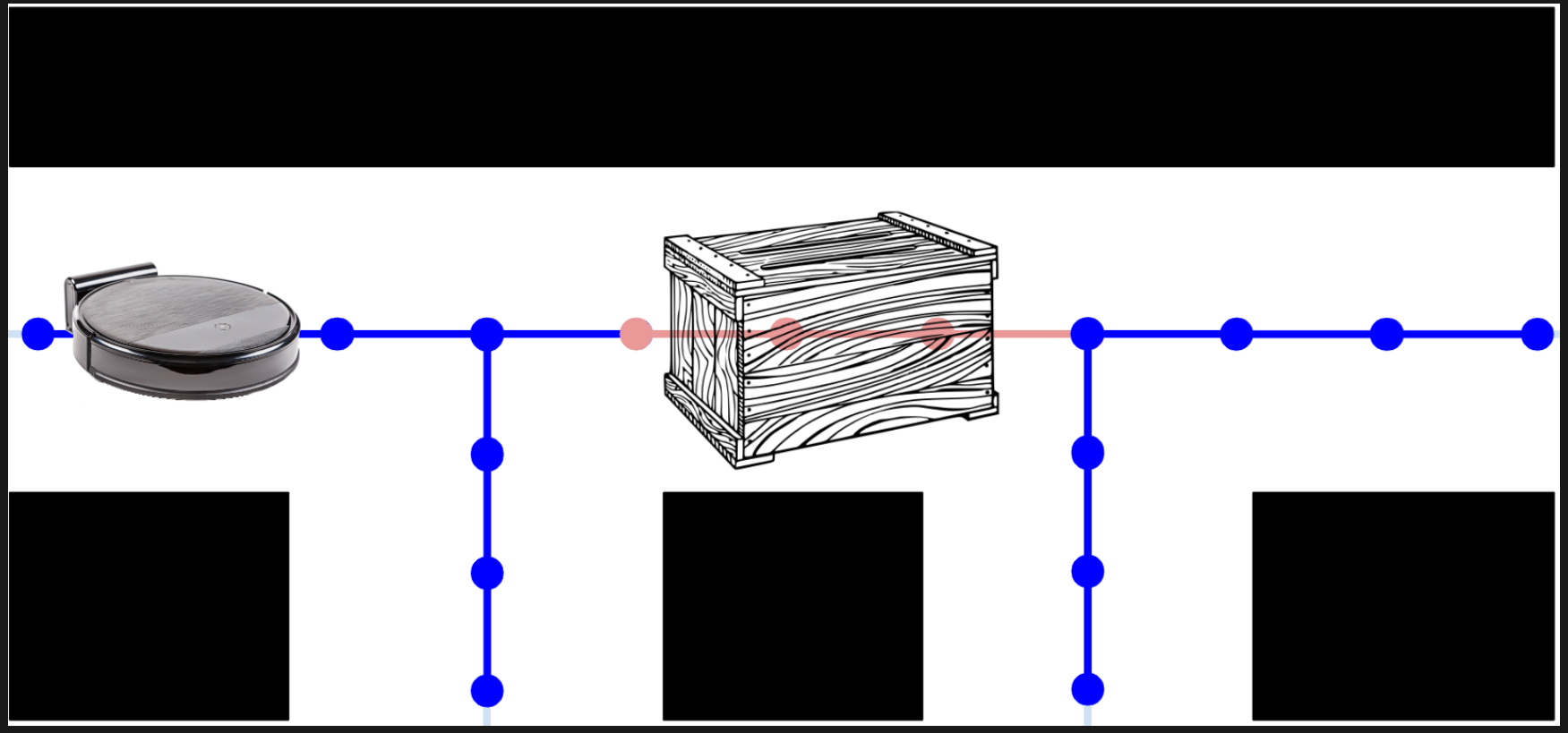


A robot could use this graph to make global plans to get around the warehouse.

Why not just pre-compute all paths?



Why not just pre-compute all paths?



What do we do?

Shortest-path search algorithms

Shortest-path search algorithms

- Dijkstra's algorithm

Shortest-path search algorithms

- Dijkstra's algorithm
- A*

Shortest-path search algorithms

- Dijkstra's algorithm
- A*
- Lifelong planning A* (LPA*)

Shortest-path search algorithms

- Dijkstra's algorithm
- A*
- Lifelong planning A* (LPA*)
- D*

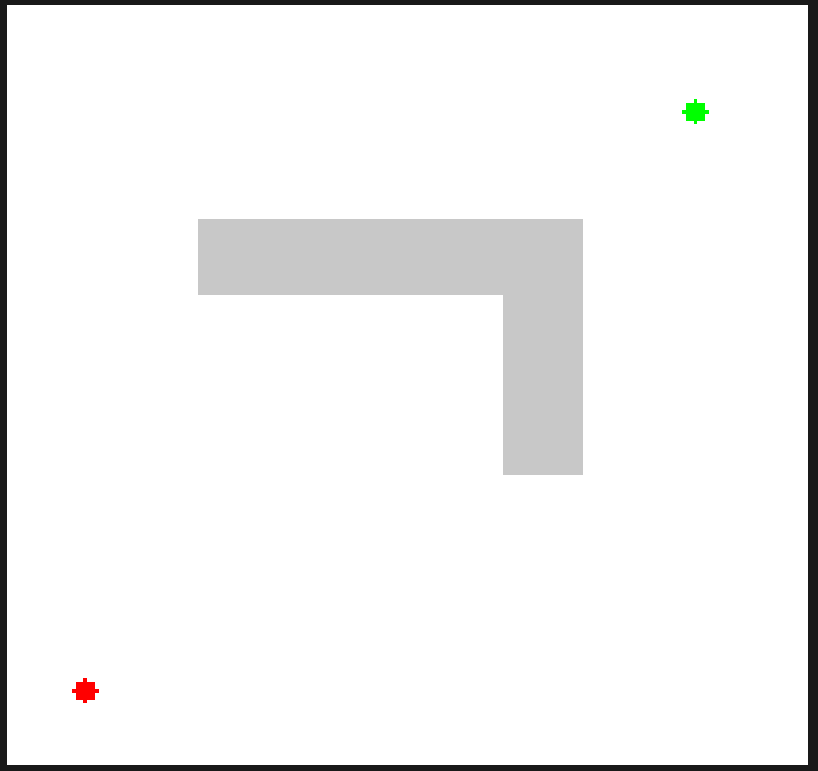
Shortest-path search algorithms

- Dijkstra's algorithm
- A*
- Lifelong planning A* (LPA*)
- D*

Shortest-path search algorithms

- Dijkstra's algorithm
- A*
- Lifelong planning A* (LPA*)
- D*
- etc.

Dijkstra's algorithm (early stopping)



Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                              # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q     # create a priority queue
6    Q.add_with_priority((start, start, 0))             # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                   # remove next best vertex (u) and its
10                                                      # predecessor (p)
11     if prev[u] is not UNDEFINED:                     # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                      # otherwise, set predecessor of (u)
14
15     if u is target:                                  # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)             # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))            # enqueue (v,u,dist)
20
21   return prev[]                                      # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)          # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
function Dijkstras(Graph, source, target):

  for each vertex v in Graph.Vertices:
    prev[v] ← UNDEFINED                                  # predecessors of v (visited set)
  create (vertex, vertex, dist) priority queue Q         # create a priority queue
  Q.add_with_priority((start, start, 0))                 # add start vertex to priority queue

  while Q is not empty:
    (p,u,dist_u) ← Q.extract_min()                       # remove next best vertex (u) and its
                                                         # predecessor (p)
    if prev[u] is not UNDEFINED:                         # if we have visited (u), ignore
      continue
    prev[u] ← p                                          # otherwise, set predecessor of (u)

    if u is target:                                      # end if we are at the target vertex
      break
    for each neighbor v of u with prev[v] is UNDEFINED:
      dist_v ← dist_u + Graph.Edge(u, v)                 # compute distance to (v)
      Q.add_with_priority((u, v, dist_v))                # enqueue (v,u,dist)

  return prev[]                                          # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
function Dijkstras(Graph, source, target):

  for each vertex v in Graph.Vertices:
    prev[v] ← UNDEFINED                              # predecessors of v (visited set)
  create (vertex, vertex, dist) priority queue Q     # create a priority queue
  Q.add_with_priority((start, start, 0))             # add start vertex to priority queue

  while Q is not empty:
    (p,u,dist_u) ← Q.extract_min()                   # remove next best vertex (u) and its
                                                     # predecessor (p)
    if prev[u] is not UNDEFINED:                     # if we have visited (u), ignore
      continue
    prev[u] ← p                                      # otherwise, set predecessor of (u)

    if u is target:                                  # end if we are at the target vertex
      break
    for each neighbor v of u with prev[v] is UNDEFINED:
      dist_v ← dist_u + Graph.Edge(u, v)             # compute distance to (v)
      Q.add_with_priority((u, v, dist_v))            # enqueue (v,u,dist)

  return prev[]                                      # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)          # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)          # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)          # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

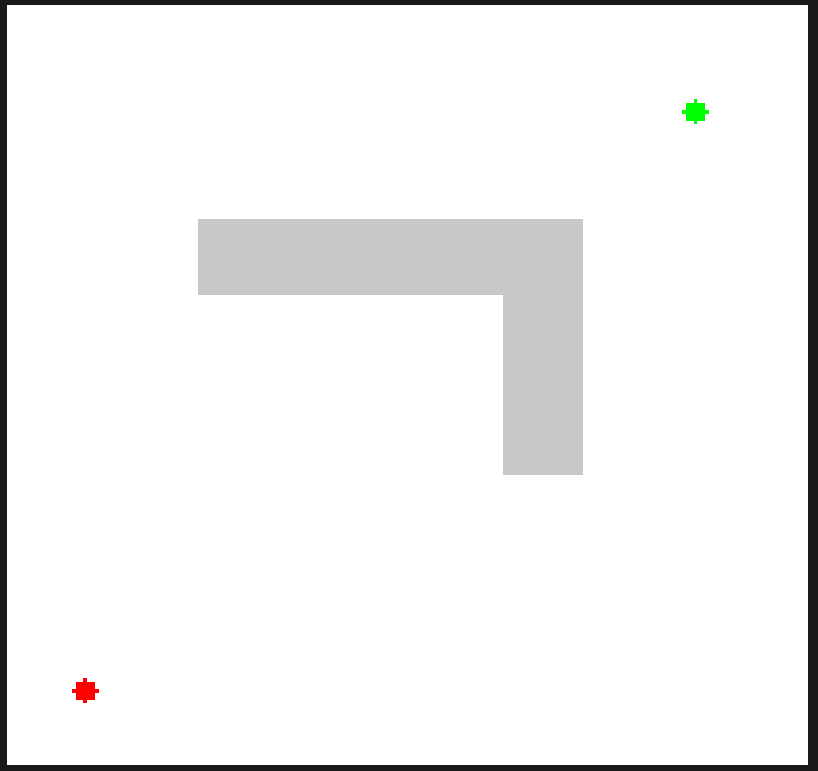
Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)          # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

Dijkstra's algorithm (early stopping)

```
1  function Dijkstras(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v)          # compute distance to (v)
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

A* (early stopping)



A* (early stopping)

```
1  function Astar(Graph, source, target):
2
3    for each vertex v in Graph.Vertices:
4      prev[v] ← UNDEFINED                          # predecessors of v (visited set)
5    create (vertex, vertex, dist) priority queue Q  # create a priority queue
6    Q.add_with_priority((start, start, 0))          # add start vertex to priority queue
7
8    while Q is not empty:
9      (p,u,dist_u) ← Q.extract_min()                # remove next best vertex (u) and its
10                                                   # predecessor (p)
11     if prev[u] is not UNDEFINED:                  # if we have visited (u), ignore
12       continue
13     prev[u] ← p                                   # otherwise, set predecessor of (u)
14
15     if u is target:                               # end if we are at the target vertex
16       break
17     for each neighbor v of u with prev[v] is UNDEFINED:
18       dist_v ← dist_u + Graph.Edge(u, v) + H(v)   # compute distance to (v) + heuristic
19       Q.add_with_priority((u, v, dist_v))         # enqueue (v,u,dist)
20
21   return prev[]                                   # return predecessors to recover path
```

Implementation Goals

Implementation Goals

- Implement Dijkstra's algorithm in C++

Implementation Goals

- Implement Dijkstra's algorithm in C++
- Use the STL: containers and other facilities

Implementation Goals

- Implement Dijkstra's algorithm in C++
- Use the STL: containers and other facilities
- Measure cache performance

Implementation Goals

- Implement Dijkstra's algorithm in C++
- Use the STL: containers and other facilities
- Measure cache performance
- Incrementally improve performance with better choices

What do we need?

What do we need?

The search algorithm!

Here is Dijkstra's algorithm as C++ code

```cpp
template<SearchContext C, SearchGraph G>
bool search(C& ctx, const G& graph, VertexID start, VertexID goal)
{
  ctx.reset(graph, start, goal);

  while (ctx.is_queue_not_empty())
  {
    const auto [from, to, to_dist] = ctx.dequeue();

    if (ctx.is_visited(to))
      continue;
    else
      ctx.mark_visited(to, from);

    if (ctx.is_terminal(to))
      return true;
    else
    {
      graph.for_each_edge(
        to,
        [&ctx, to, to_dist](VertexID v, const EdgeProperties& edge) mutable {
        if (edge.valid and !ctx.is_visited(v)) {
          ctx.enqueue(to, v, edge.dist + to_dist);
        }
      });
```

Here is Dijkstra's algorithm as C++ code

```
  ctx.reset(graph, start, goal);

  while (ctx.is_queue_not_empty())
  {
    const auto [from, to, to_dist] = ctx.dequeue();

    if (ctx.is_visited(to))
      continue;
    else
      ctx.mark_visited(to, from);

    if (ctx.is_terminal(to))
      return true;
    else
    {
      graph.for_each_edge(
        to,
        [&ctx, to, to_dist](VertexID v, const EdgeProperties& edge) mutable {
        if (edge.valid and !ctx.is_visited(v)) {
          ctx.enqueue(to, v, edge.dist + to_dist);
        }
      });
    }
  }
  return false;
}
```

Here is Dijkstra's algorithm as C++ code

```cpp
template<SearchContext C, SearchGraph G>
bool search(C& ctx, const G& graph, VertexID start, VertexID goal)
{
  ctx.reset(graph, start, goal);

  while (ctx.is_queue_not_empty())
  {
    const auto [from, to, to_dist] = ctx.dequeue();

    if (ctx.is_visited(to))
      continue;
    else
      ctx.mark_visited(to, from);

    if (ctx.is_terminal(to))
      return true;
    else
    {
      graph.for_each_edge(
        to,
        [&ctx, to, to_dist](VertexID v, const EdgeProperties& edge) mutable {
        if (edge.valid and !ctx.is_visited(v)) {
          ctx.enqueue(to, v, edge.dist + to_dist);
        }
      });
```

Here is Dijkstra's algorithm as C++ code

```
  while (ctx.is_queue_not_empty())
  {
    const auto [from, to, to_dist] = ctx.dequeue();

    if (ctx.is_visited(to))
      continue;
    else
      ctx.mark_visited(to, from);

    if (ctx.is_terminal(to))
      return true;
    else
    {
      graph.for_each_edge(
        to,
        [&ctx, to, to_dist](VertexID v, const EdgeProperties& edge) mutable {
        if (edge.valid and !ctx.is_visited(v)) {
          ctx.enqueue(to, v, edge.dist + to_dist);
        }
      });
    }
  }
  return false;
}
```

Here is Dijkstra's algorithm as C++ code

```
  while (ctx.is_queue_not_empty())
  {
    const auto [from, to, to_dist] = ctx.dequeue();

    if (ctx.is_visited(to))
      continue;
    else
      ctx.mark_visited(to, from);

    if (ctx.is_terminal(to))
      return true;
    else
    {
      graph.for_each_edge(
        to,
        [&ctx, to, to_dist](VertexID v, const EdgeProperties& edge) mutable {
        if (edge.valid and !ctx.is_visited(v)) {
          ctx.enqueue(to, v, edge.dist + to_dist);
        }
      });
    }
  }
  return false;
}
```

Here is Dijkstra's algorithm as C++ code

```
  while (ctx.is_queue_not_empty())
  {
    const auto [from, to, to_dist] = ctx.dequeue();

    if (ctx.is_visited(to))
      continue;
    else
      ctx.mark_visited(to, from);

    if (ctx.is_terminal(to))
      return true;
    else
    {
      graph.for_each_edge(
        to,
        [&ctx, to, to_dist](VertexID v, const EdgeProperties& edge) mutable {
        if (edge.valid and !ctx.is_visited(v)) {
          ctx.enqueue(to, v, edge.dist + to_dist);
        }
      });
    }
  }
  return false;
}
```

Here is the code to recover a path on a successful search

```
template<typename OutputIteratorT, SearchContext C>
OutputIteratorT get_reverse_path(OutputIteratorT out, const C& ctx, VertexID from)
 {
  (*out) = from;
  while (true)
  {
    if (const auto to = ctx.predecessor(from); to == from)
    {
      break;
    }
    else
    {
      (*out) = to;
      from = to;
    }
  }
  return out;
}
```

Here is the code to recover a path on a successful search

```
template<typename OutputIteratorT, SearchContext C>
OutputIteratorT get_reverse_path(OutputIteratorT out, const C& ctx, VertexID from)
 {
  (*out) = from;
  while (true)
  {
    if (const auto to = ctx.predecessor(from); to == from)
    {
      break;
    }
    else
    {
      (*out) = to;
      from = to;
    }
  }
  return out;
}
```

What else do we need?

What else do we need?

Something to hold the graph!

What else do we need?

Something to hold the graph!

- Vertex adjacencies (edges)

What else do we need?

Something to hold the graph!

- Vertex adjacencies (edges)
- Vertex properties (locations, etc.)

What else do we need?

Something to hold the graph!

- Vertex adjacencies (edges)
- Vertex properties (locations, etc.)
- Edge properties (distance, validity, etc.)

What else do we need?

```
template <typename T>
concept SearchGraph =
  requires(T&& g)
  {
      { g.for_each_edge(VertexID{}, [](VertexID, const EdgeProperties&) {}) };
      { g.vertex_count() };
      { g.vertex(VertexID{}) };
  };
```

```cpp
struct VertexProperties { double x; double y; /* ... */ };

struct EdgeProperties { bool valid; EdgeWeight weight; /* ... */ };

using Edge = std::pair<VertexID, EdgeProperties>;

class Graph
{
public:
  /// Returns vertex properties for a given vertex
  const VertexProperties& vertex(VertexID q) const;

  /// Returns the total number of vertices
  std::size_t vertex_count() const;

  /// Invokes a visitor on each edge of "q"
  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const;
};
```

```cpp
struct VertexProperties { double x; double y; /* ... */ };

struct EdgeProperties { bool valid; EdgeWeight weight; /* ... */ };

using Edge = std::pair<VertexID, EdgeProperties>;

class Graph
{
public:
  /// Returns vertex properties for a given vertex
  const VertexProperties& vertex(VertexID q) const;

  /// Returns the total number of vertices
  std::size_t vertex_count() const;

  /// Invokes a visitor on each edge of "q"
  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const;
};
```

```cpp
struct VertexProperties { double x; double y; /* ... */ };

struct EdgeProperties { bool valid; EdgeWeight weight; /* ... */ };

using Edge = std::pair<VertexID, EdgeProperties>;

class Graph
{
public:
  /// Returns vertex properties for a given vertex
  const VertexProperties& vertex(VertexID q) const;

  /// Returns the total number of vertices
  std::size_t vertex_count() const;

  /// Invokes a visitor on each edge of "q"
  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const;
};
```

```cpp
struct VertexProperties { double x; double y; /* ... */ };

struct EdgeProperties { bool valid; EdgeWeight weight; /* ... */ };

using Edge = std::pair<VertexID, EdgeProperties>;

class Graph
{
public:
  /// Returns vertex properties for a given vertex
  const VertexProperties& vertex(VertexID q) const;

  /// Returns the total number of vertices
  std::size_t vertex_count() const;

  /// Invokes a visitor on each edge of "q"
  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const;
};
```

What else do we need?

What else do we need?

Something to hold search state!

What else do we need?

Something to hold search state!

- A container to hold visited vertices (and their relationships)

What else do we need?

Something to hold search state!

- A container to hold visited vertices (and their relationships)
- A min-sorted priority queue of vertices

What else do we need?

```
template <typename T>
concept SearchContext =
  requires(T&& ctx)
  {
      { ctx.is_queue_not_empty() };
      { ctx.is_visited(VertexID{}) };
      { ctx.is_terminal(VertexID{}) };
      { ctx.mark_visited(VertexID{}, VertexID{}) };
      { ctx.enqueue(VertexID{}, VertexID{}, EdgeWeight{}) };
      { ctx.dequeue() };
      { ctx.predecessor(VertexID{}) };
  };
```

```cpp
struct Transition { VertexID from; VertexID to; EdgeWeight weight; };

class TerminateAtGoal
{
public:
  /// Resets internal state; sets start and goal vertices
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g);

  /// Returns true if a vertex is the goal
  bool is_terminal(VertexID q) const;

  /// Returns true if a vertex has been visited
  bool is_visited(VertexID q) const;
  /// Sets a vertex as visited
  void mark_visited(VertexID from, VertexID to);
  /// Returns the predecessor of a vertex
  VertexID predecessor(VertexID q) const;

  /// Returns true if there are still elements in the queue
  bool is_queue_not_empty() const;
  /// Get vertex from queue next best distance
  Transition dequeue();
  /// Adds vertex to queue with a distance
  void enqueue(VertexID from, VertexID to, EdgeWeight w);
```

```cpp
struct Transition { VertexID from; VertexID to; EdgeWeight weight; };

class TerminateAtGoal
{
public:
  /// Resets internal state; sets start and goal vertices
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g);

  /// Returns true if a vertex is the goal
  bool is_terminal(VertexID q) const;

  /// Returns true if a vertex has been visited
  bool is_visited(VertexID q) const;
  /// Sets a vertex as visited
  void mark_visited(VertexID from, VertexID to);
  /// Returns the predecessor of a vertex
  VertexID predecessor(VertexID q) const;

  /// Returns true if there are still elements in the queue
  bool is_queue_not_empty() const;
  /// Get vertex from queue next best distance
  Transition dequeue();
  /// Adds vertex to queue with a distance
  void enqueue(VertexID from, VertexID to, EdgeWeight w);
```

```cpp
struct Transition { VertexID from; VertexID to; EdgeWeight weight; };

class TerminateAtGoal
{
public:
  /// Resets internal state; sets start and goal vertices
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g);

  /// Returns true if a vertex is the goal
  bool is_terminal(VertexID q) const;

  /// Returns true if a vertex has been visited
  bool is_visited(VertexID q) const;
  /// Sets a vertex as visited
  void mark_visited(VertexID from, VertexID to);
  /// Returns the predecessor of a vertex
  VertexID predecessor(VertexID q) const;

  /// Returns true if there are still elements in the queue
  bool is_queue_not_empty() const;
  /// Get vertex from queue next best distance
  Transition dequeue();
  /// Adds vertex to queue with a distance
  void enqueue(VertexID from, VertexID to, EdgeWeight w);
```

```
2
3  class TerminateAtGoal
4  {
5  public:
6    /// Resets internal state; sets start and goal vertices
7    template<SearchGraph G>
8    void reset(G&& graph, VertexID s, VertexID g);
9
10   /// Returns true if a vertex is the goal
11   bool is_terminal(VertexID q) const;
12
13   /// Returns true if a vertex has been visited
14   bool is_visited(VertexID q) const;
15   /// Sets a vertex as visited
16   void mark_visited(VertexID from, VertexID to);
17   /// Returns the predecessor of a vertex
18   VertexID predecessor(VertexID q) const;
19
20   /// Returns true if there are still elements in the queue
21   bool is_queue_not_empty() const;
22   /// Get vertex from queue next best distance
23   Transition dequeue();
24   /// Adds vertex to queue with a distance
25   void enqueue(VertexID from, VertexID to, EdgeWeight w);
26 };
```

```
2
3  class TerminateAtGoal
4  {
5  public:
6    /// Resets internal state; sets start and goal vertices
7    template<SearchGraph G>
8    void reset(G&& graph, VertexID s, VertexID g);
9
10   /// Returns true if a vertex is the goal
11   bool is_terminal(VertexID q) const;
12
13   /// Returns true if a vertex has been visited
14   bool is_visited(VertexID q) const;
15   /// Sets a vertex as visited
16   void mark_visited(VertexID from, VertexID to);
17   /// Returns the predecessor of a vertex
18   VertexID predecessor(VertexID q) const;
19
20   /// Returns true if there are still elements in the queue
21   bool is_queue_not_empty() const;
22   /// Get vertex from queue next best distance
23   Transition dequeue();
24   /// Adds vertex to queue with a distance
25   void enqueue(VertexID from, VertexID to, EdgeWeight w);
26 };
```

Implementation 0

```cpp
#include <algorhitm> // std::for_each
#include <map>       // std::map, std::multimap
#include <tuple>     // std::apply

class Graph
{
private:
  std::multimap<VertexID, Edge> adjacencies_;

  std::map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
```

```cpp
#include <algorhitm> // std::for_each
#include <map>       // std::map, std::multimap
#include <tuple>     // std::apply

class Graph
{
private:
  std::multimap<VertexID, Edge> adjacencies_;

  std::map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
```

```cpp
#include <algorhitm> // std::for_each
#include <map>       // std::map, std::multimap
#include <tuple>     // std::apply

class Graph
{
private:
  std::multimap<VertexID, Edge> adjacencies_;

  std::map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
```

```cpp
#include <algorhitm> // std::for_each
#include <map>       // std::map, std::multimap
#include <tuple>     // std::apply

class Graph
{
private:
  std::multimap<VertexID, Edge> adjacencies_;

  std::map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
```

```
2  #include <map>       // std::map, std::multimap
3  #include <tuple>     // std::apply
4
5  class Graph
6  {
7  private:
8    std::multimap<VertexID, Edge> adjacencies_;
9
10   std::map<VertexID, VertexProperties> vertices_;
11
12 public:
13   const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }
14
15   std::size_t vertex_count() const { return vertices_.size(); }
16
17   template<typename EdgeVisitorT>
18   void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
19   {
20     const auto [first, last] = adjacencies_.equal_range(q);
21     std::for_each(
22       first,
23       last,
24       [visitor](const auto& parent_and_edge) mutable
25       {
26         std::apply(visitor, parent and edge.second);
```

```
class Graph
{
private:
  std::multimap<VertexID, Edge> adjacencies_;

  std::map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
        std::apply(visitor, parent_and_edge.second);
      });
  }
};
```

```
class Graph
{
private:
  std::multimap<VertexID, Edge> adjacencies_;

  std::map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
        std::apply(visitor, parent_and_edge.second);
      });
  }
};
```

```cpp
#include <functional>  // std::greater
#include <queue>       // std::priority_queue
#include <map>         // std::map
#include <vector>      // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

```cpp
#include <functional>  // std::greater
#include <queue>       // std::priority_queue
#include <map>         // std::map
#include <vector>      // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

```cpp
#include <functional>  // std::greater
#include <queue>       // std::priority_queue
#include <map>         // std::map
#include <vector>      // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

```cpp
#include <functional>  // std::greater
#include <queue>       // std::priority_queue
#include <map>         // std::map
#include <vector>      // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

```cpp
#include <functional>  // std::greater
#include <queue>       // std::priority_queue
#include <map>         // std::map
#include <vector>      // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

```cpp
#include <functional>  // std::greater
#include <queue>       // std::priority_queue
#include <map>         // std::map
#include <vector>      // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

```
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
    enqueue(s, s, 0);
  }

  bool is_queue_not_empty() const { return !queue_.empty(); }

  bool is_visited(VertexID q) const { return visited_.count(q); }

  bool is_terminal(VertexID q) const { return goal_ == q; }
```

```
29  bool is_queue_not_empty() const { return !queue_.empty(); }
30
31  bool is_visited(VertexID q) const { return visited_.count(q); }
32
33  bool is_terminal(VertexID q) const { return goal_ == q; }
34
35  void mark_visited(VertexID from, VertexID to) { visited_.emplace(to, from); }
36
37  VertexID predecessor(VertexID q) const
38  {
39    if (const auto itr = visited_.find(q); itr == visited_.end())
40    {
41      return q;
42    }
43    else
44    {
45      return itr->second;
46    }
47  }
48
49  Transition dequeue()
50  {
51    auto t = queue_.top();
52    queue_.pop();
53    return t;
```

Measuring cache performance

Measuring cache performance

- Run searches over our example warehouse graph

Measuring cache performance

- Run searches over our example warehouse graph
- Select some percentage of all vertices, N

Measuring cache performance

- Run searches over our example warehouse graph
- Select some percentage of all vertices, N
- Run N^2 searches between all vertices as start/goal pairs

Measuring cache performance

Run under cachegrind

Measuring cache performance

Run under cachegrind

```
> valgrind --tool=cachegrind --cache-sim=yes ./run_search ...
```

Measuring cache performance

```
==131579== Cachegrind, a cache and branch-prediction profiler
==131579== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
…
==131579== I   refs:      20,721,901,316
==131579== I1  misses:             4,744
==131579== LLi misses:             3,909
==131579== I1  miss rate:           0.00%
==131579== LLi miss rate:           0.00%
==131579==
==131579== D   refs:       7,762,293,651  (6,285,352,547 rd   + 1,476,941,104 wr)
==131579== D1  misses:       615,102,179  (  604,712,021 rd   +    10,390,158 wr)
==131579== LLd misses:         3,336,167  (    2,266,653 rd   +     1,069,514 wr)
==131579== D1  miss rate:            7.9% (          9.6%     +           0.7%  )
==131579== LLd miss rate:            0.0% (          0.0%     +           0.1%  )
==131579==
==131579== LL refs:          615,106,923  (  604,716,765 rd   +    10,390,158 wr)
==131579== LL misses:          3,340,076  (    2,270,562 rd   +     1,069,514 wr)
==131579== LL miss rate:             0.0% (          0.0%     +           0.1%  )
```

Measuring cache performance

```
==131579== Cachegrind, a cache and branch-prediction profiler
==131579== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
…
==131579== I   refs:      20,721,901,316
==131579== I1  misses:             4,744
==131579== LLi misses:             3,909
==131579== I1  miss rate:           0.00%
==131579== LLi miss rate:           0.00%
==131579==
==131579== D   refs:       7,762,293,651  (6,285,352,547 rd   + 1,476,941,104 wr)
==131579== D1  misses:       615,102,179  (  604,712,021 rd   +    10,390,158 wr)
==131579== LLd misses:         3,336,167  (    2,266,653 rd   +     1,069,514 wr)
==131579== D1  miss rate:            7.9% (          9.6%     +           0.7%  )
==131579== LLd miss rate:            0.0% (          0.0%     +           0.1%  )
==131579==
==131579== LL refs:          615,106,923  (  604,716,765 rd   +    10,390,158 wr)
==131579== LL misses:          3,340,076  (    2,270,562 rd   +     1,069,514 wr)
==131579== LL miss rate:             0.0% (          0.0%     +           0.1%  )
```

Measuring cache performance

```
==131579== Cachegrind, a cache and branch-prediction profiler
==131579== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
…
==131579== I   refs:      20,721,901,316
==131579== I1  misses:             4,744
==131579== LLi misses:             3,909
==131579== I1  miss rate:           0.00%
==131579== LLi miss rate:           0.00%
==131579==
==131579== D   refs:       7,762,293,651  (6,285,352,547 rd   + 1,476,941,104 wr)
==131579== D1  misses:       615,102,179  (  604,712,021 rd   +    10,390,158 wr)
==131579== LLd misses:         3,336,167  (    2,266,653 rd   +     1,069,514 wr)
==131579== D1  miss rate:            7.9% (          9.6%     +           0.7%  )
==131579== LLd miss rate:            0.0% (          0.0%     +           0.1%  )
==131579==
==131579== LL refs:          615,106,923  (  604,716,765 rd   +    10,390,158 wr)
==131579== LL misses:          3,340,076  (    2,270,562 rd   +     1,069,514 wr)
==131579== LL miss rate:             0.0% (          0.0%     +           0.1%  )
```

Measuring cache performance

perf

Measuring cache performance

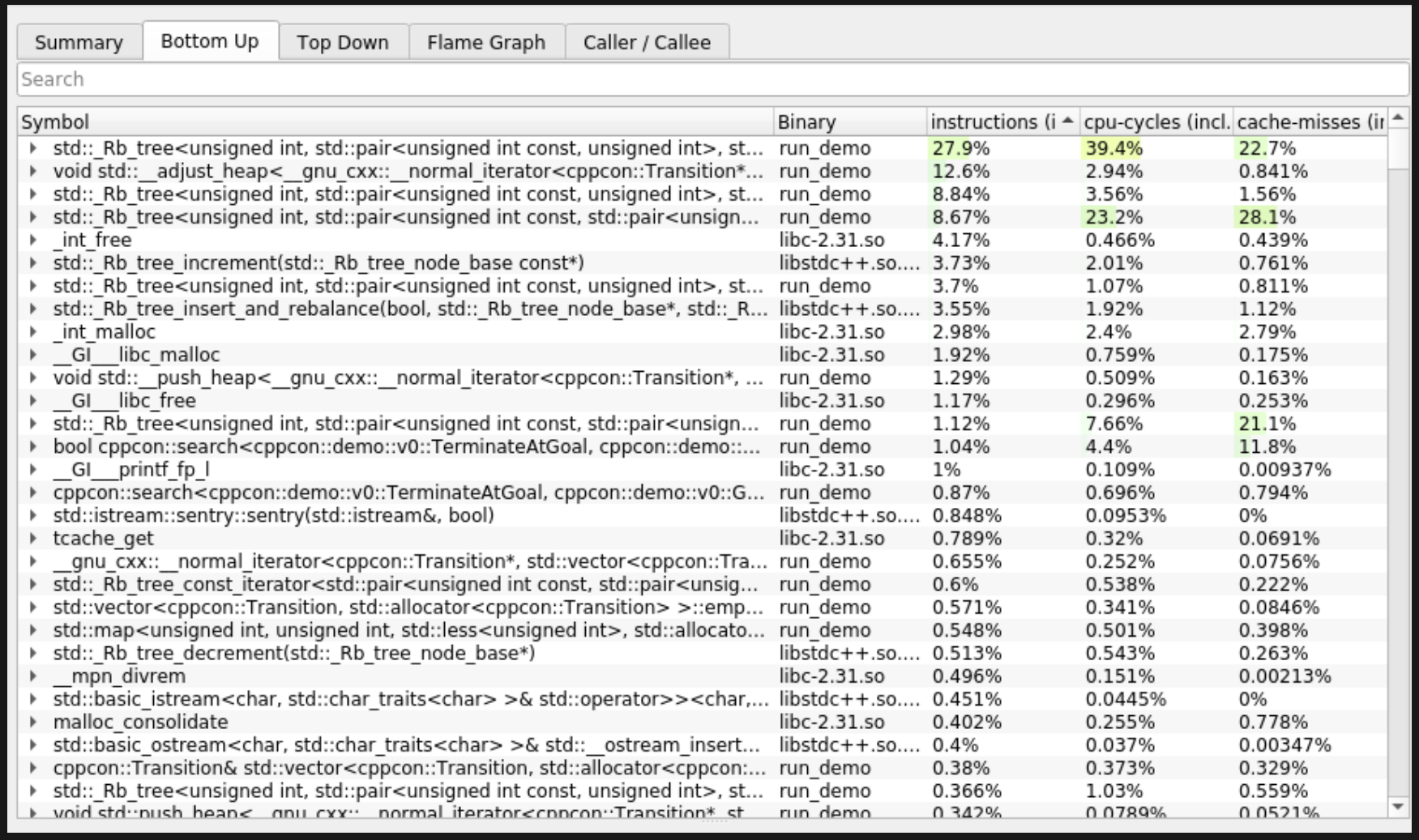
perf

```
> sudo apt install linux-tools-6.2.0-33-generic

> perf record \
 -o ~/perf.data \
 --call-graph dwarf \
 --event instructions,cpu-cycles,cache-misses,branches,branch-misses \
 --aio \
 --sample-cpu <exec>
```

Measuring cache performance

perf

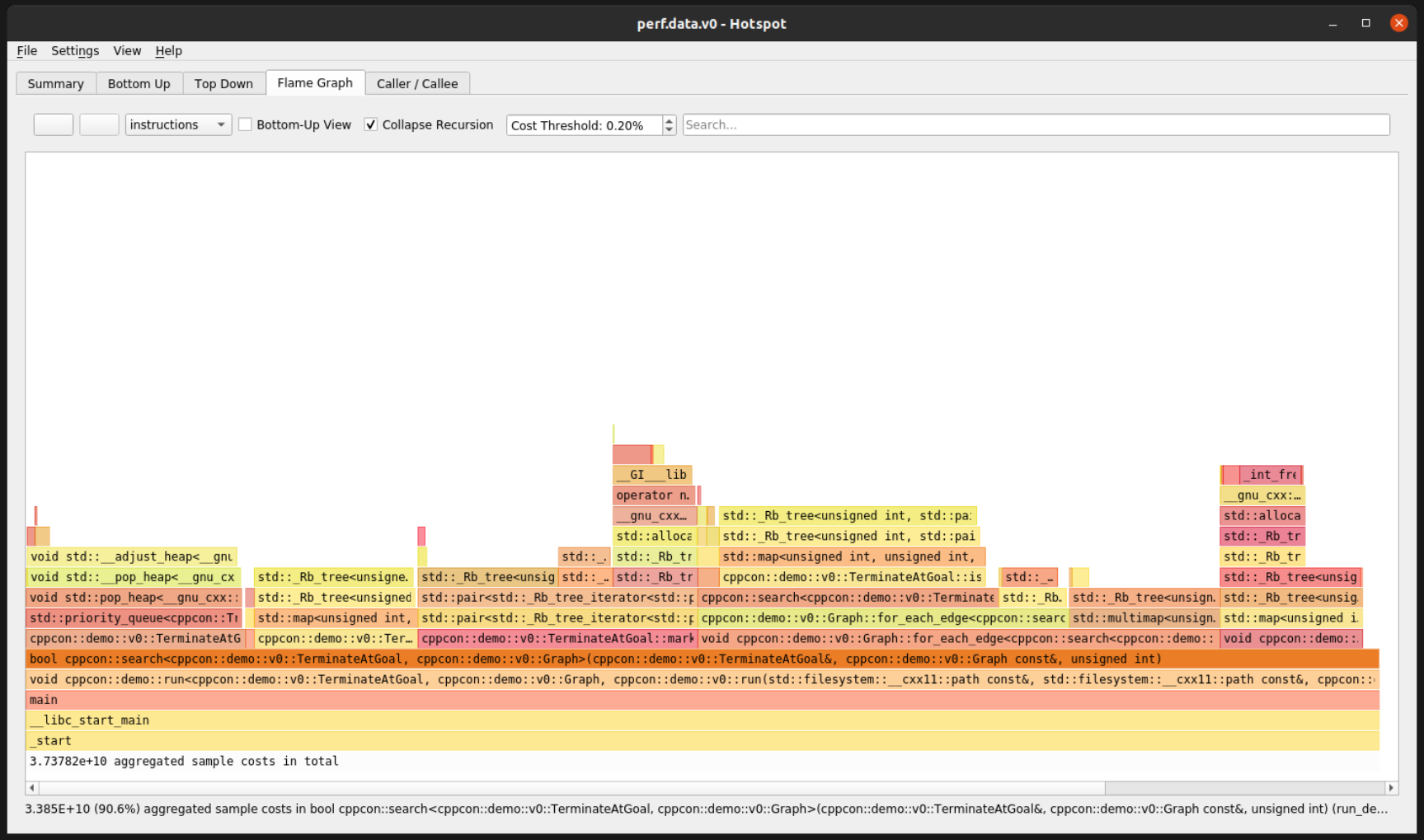
```
> sudo apt install hotspot

> sudo hotspot
```



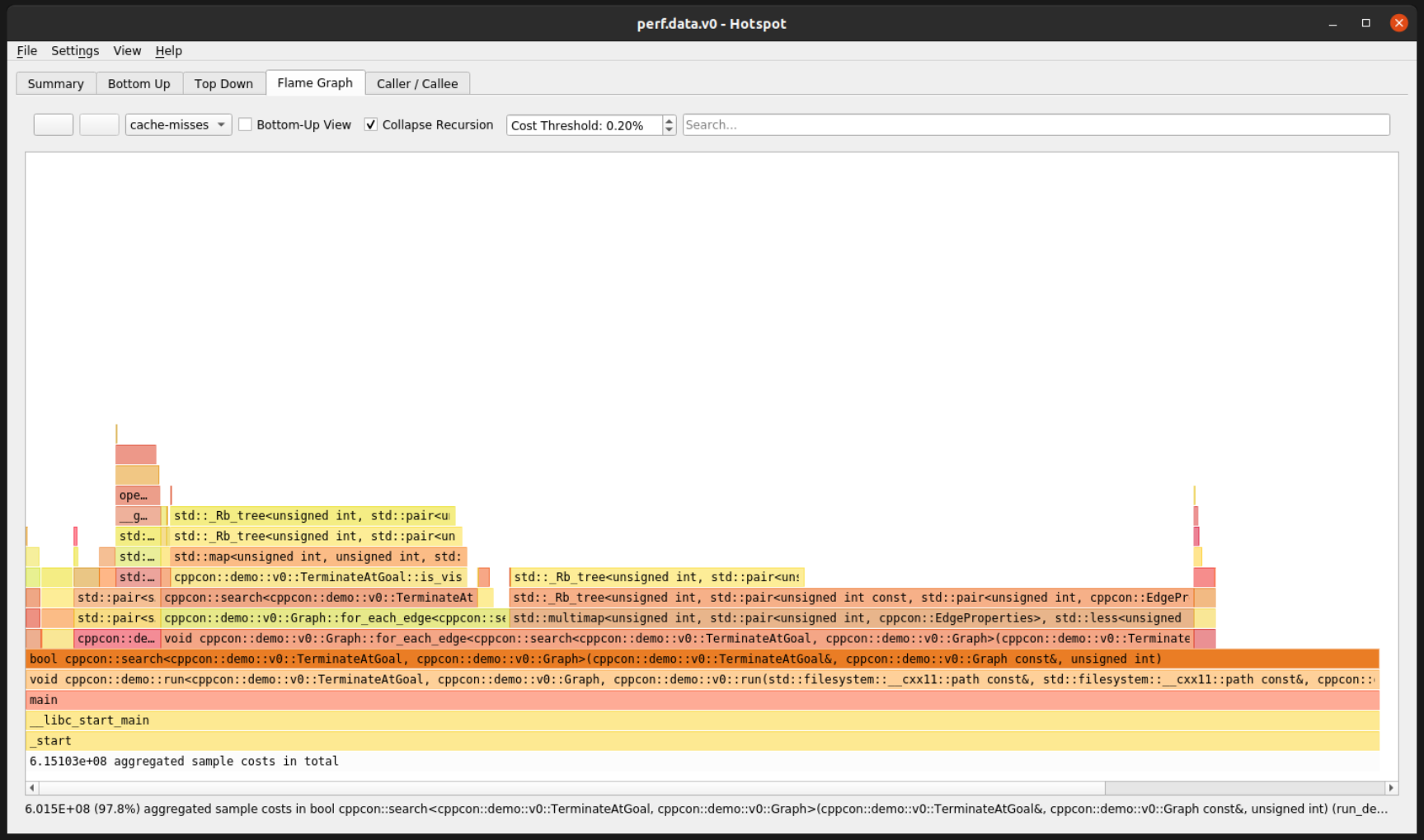
Summary | Bottom Up | Top Down | Flame Graph | Caller / Callee

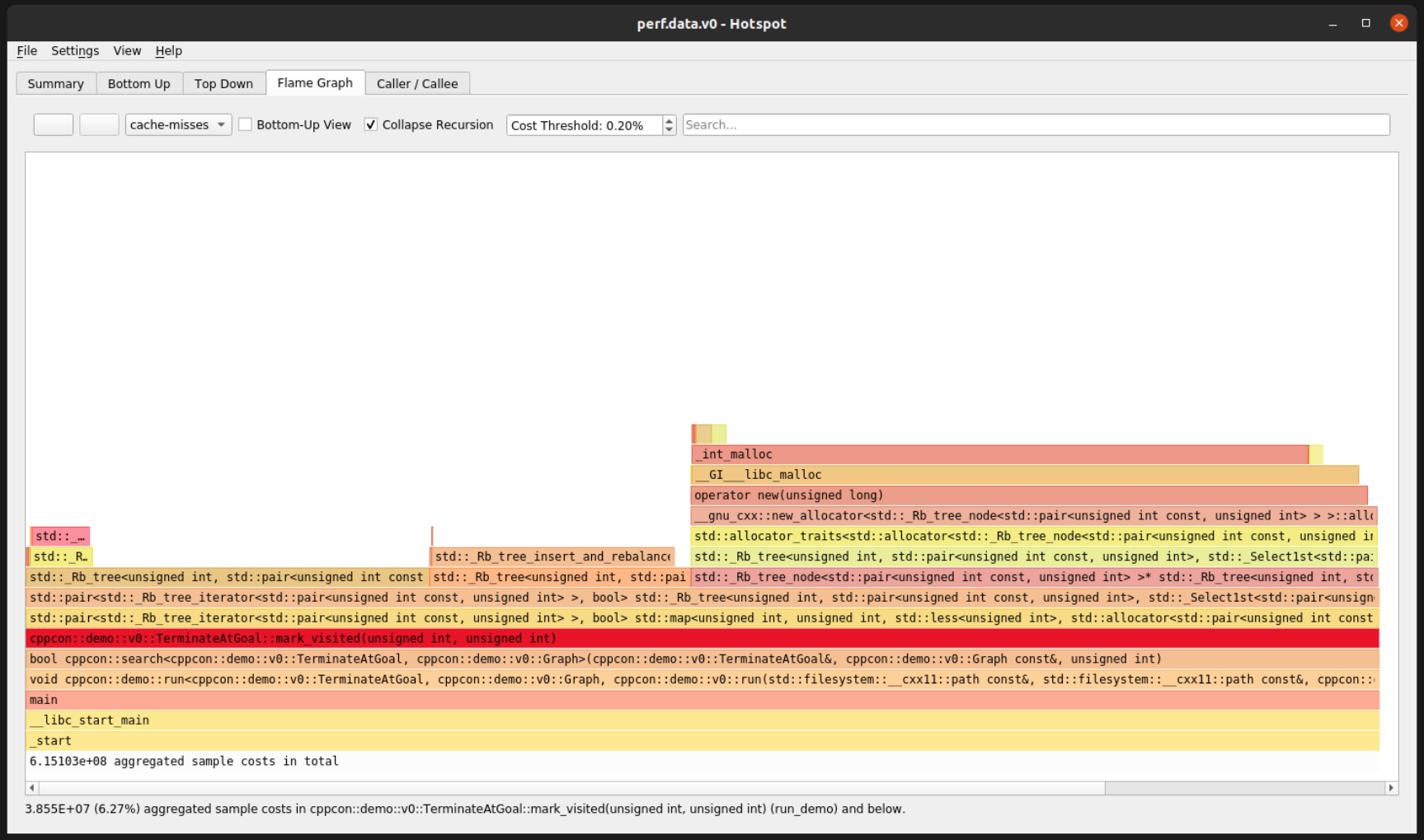
Search

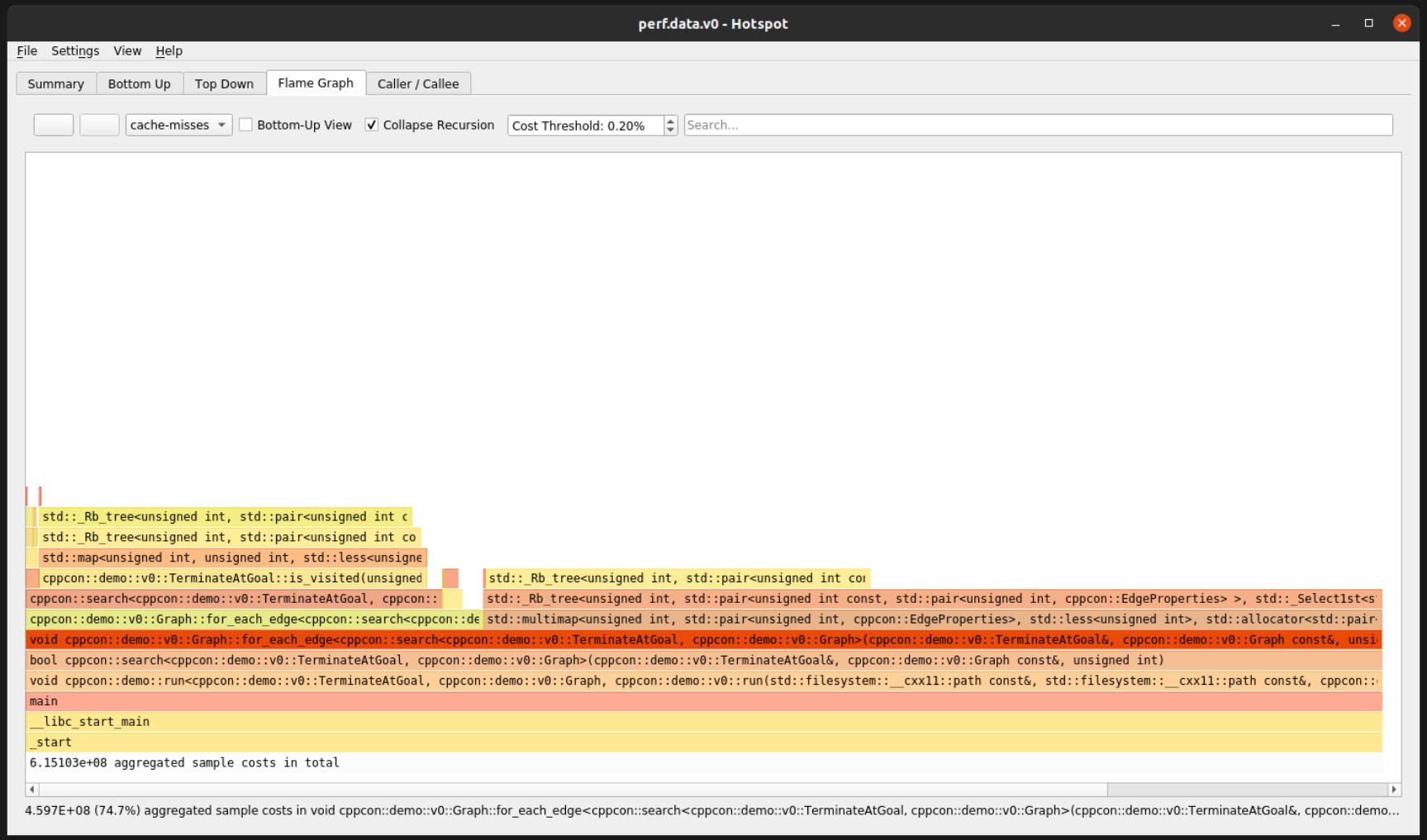
Symbol	Binary	instructions (i ▲	cpu-cycles (incl.	cache-misses (ir
std::_Rb_tree<unsigned int, std::pair<unsigned int const, unsigned int>, st...	run_demo	27.9%	39.4%	22.7%
void std::__adjust_heap<__gnu_cxx::__normal_iterator<cppcon::Transition*...	run_demo	12.6%	2.94%	0.841%
std::_Rb_tree<unsigned int, std::pair<unsigned int const, unsigned int>, st...	run_demo	8.84%	3.56%	1.56%
std::_Rb_tree<unsigned int, std::pair<unsigned int const, std::pair<unsign...	run_demo	8.67%	23.2%	28.1%
_int_free	libc-2.31.so	4.17%	0.466%	0.439%
std::_Rb_tree_increment(std::_Rb_tree_node_base const*)	libstdc++.so....	3.73%	2.01%	0.761%
std::_Rb_tree<unsigned int, std::pair<unsigned int const, unsigned int>, st...	run_demo	3.7%	1.07%	0.811%
std::_Rb_tree_insert_and_rebalance(bool, std::_Rb_tree_node_base*, std::_R...	libstdc++.so....	3.55%	1.92%	1.12%
_int_malloc	libc-2.31.so	2.98%	2.4%	2.79%
__GI___libc_malloc	libc-2.31.so	1.92%	0.759%	0.175%
void std::__push_heap<__gnu_cxx::__normal_iterator<cppcon::Transition*, ...	run_demo	1.29%	0.509%	0.163%
__GI___libc_free	libc-2.31.so	1.17%	0.296%	0.253%
std::_Rb_tree<unsigned int, std::pair<unsigned int const, std::pair<unsign...	run_demo	1.12%	7.66%	21.1%
bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::...	run_demo	1.04%	4.4%	11.8%
__GI___printf_fp_l	libc-2.31.so	1%	0.109%	0.00937%
cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::G...	run_demo	0.87%	0.696%	0.794%
std::istream::sentry::sentry(std::istream&, bool)	libstdc++.so....	0.848%	0.0953%	0%
tcache_get	libc-2.31.so	0.789%	0.32%	0.0691%
__gnu_cxx::__normal_iterator<cppcon::Transition*, std::vector<cppcon::Tra...	run_demo	0.655%	0.252%	0.0756%
std::_Rb_tree_const_iterator<std::pair<unsigned int const, std::pair<unsig...	run_demo	0.6%	0.538%	0.222%
std::vector<cppcon::Transition, std::allocator<cppcon::Transition> >::emp...	run_demo	0.571%	0.341%	0.0846%
std::map<unsigned int, unsigned int, std::less<unsigned int>, std::allocato...	run_demo	0.548%	0.501%	0.398%
std::_Rb_tree_decrement(std::_Rb_tree_node_base*)	libstdc++.so....	0.513%	0.543%	0.263%
__mpn_divrem	libc-2.31.so	0.496%	0.151%	0.00213%
std::basic_istream<char, std::char_traits<char> >& std::operator>><char,...	libstdc++.so....	0.451%	0.0445%	0%
malloc_consolidate	libc-2.31.so	0.402%	0.255%	0.778%
std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert...	libstdc++.so....	0.4%	0.037%	0.00347%
cppcon::Transition& std::vector<cppcon::Transition, std::allocator<cppcon:...	run_demo	0.38%	0.373%	0.329%
std::_Rb_tree<unsigned int, std::pair<unsigned int const, unsigned int>, st...	run_demo	0.366%	1.03%	0.559%
void std::push_heap<__gnu_cxx::__normal_iterator<cppcon::Transition*, st...	run_demo	0.342%	0.0789%	0.0521%


perf.data.v0 - Hotspot
File Settings View Help
Summary Bottom Up Top Down Flame Graph Caller / Callee
instructions
Bottom-Up View
Collapse Recursion
Cost Threshold: 0.20%
Search...
__GI___lib
operator n.
__gnu_cxx:…
std::_Rb_tree<unsigned int, std::pa:
std::alloca
std::_Rb_tr
void std::__adjust_heap<__gnu
std::_Rb_tr
std::map<unsigned int, unsigned int,
void std::__pop_heap<__gnu_cx
std::_Rb_tree<unsigne.
std::_Rb_tree<unsig
cppcon::demo::v0::TerminateAtGoal::is
std::_Rb_tree<unsig
void std::pop_heap<__gnu_cxx::
std::_Rb_tree<unsigned
std::pair<std::_Rb_tree_iterator<std::p
cppcon::search<cppcon::demo::v0::Terminate
std::_Rb_tree<unsign.
std::priority_queue<cppcon::T
std::map<unsigned int,
std::pair<std::_Rb_tree_iterator<std::p
cppcon::demo::v0::Graph::for_each_edge<cppcon::searc
std::multimap<unsign.
std::map<unsigned i.
cppcon::demo::v0::TerminateAtG
cppcon::demo::v0::Ter…
cppcon::demo::v0::TerminateAtGoal::mark
void cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::demo::
void cppcon::demo::
bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int)
void cppcon::demo::run<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph, cppcon::demo::v0::run(std::filesystem::__cxx11::path const&, std::filesystem::__cxx11::path const&, cppcon::
main
__libc_start_main
_start
3.73782e+10 aggregated sample costs in total
3.385E+10 (90.6%) aggregated sample costs in bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int) (run_de…


perf.data.v0 - Hotspot
File Settings View Help
Summary Bottom Up Top Down Flame Graph Caller / Callee
cpu-cycles
Bottom-Up View
Collapse Recursion
Cost Threshold: 0.20%
Search...
__GI_…
oper…
__gnu…
std::_Rb_tree<unsigned int, std::pair<unsigned int c
std::a…
std::_Rb_tree<unsigned int, std::pair<unsigned int co
void…
std::_…
std::map<unsigned int, unsigned int, std::less<unsigne
void …
std::_Rb_tree<unsigned
std::_…
std::_…
cppcon::demo::v0::TerminateAtGoal::is_visited(unsigned
std::_Rb_tree<
void …
std::_Rb_tree<unsigned i
std::pair<std::_Rb_
cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon:
std::_Rb_tree<unsigned int, std::pair<unsigned int const, st
std::…
std::map<unsigned int, u
std::pair<std::_Rb_
cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::d
std::multimap<unsigned int, std::pair<unsigned int, cppcon::
cppc…
cppcon::demo::v0::Termin…
cppcon::demo::v0::T
void cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon
bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int)
void cppcon::demo::run<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph, cppcon::demo::v0::run(std::filesystem::__cxx11::path const&, std::filesystem::__cxx11::path const&, cppcon::
main
__libc_start_main
_start
1.26993e+11 aggregated sample costs in total
1.245E+11 (98.1%) aggregated sample costs in bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int) (run_de…


perf.data.v0 - Hotspot
File Settings View Help
Summary Bottom Up Top Down Flame Graph Caller / Callee
cache-misses
Bottom-Up View
Collapse Recursion
Cost Threshold: 0.20%
Search...
ope…
__g…
std:…
std:…
std:…
std::_Rb_tree<unsigned int, std::pair<u
std::_Rb_tree<unsigned int, std::pair<un
std::map<unsigned int, unsigned int, std:
cppcon::demo::v0::TerminateAtGoal::is_vis
std::_Rb_tree<unsigned int, std::pair<uns
std::pair<s
cppcon::search<cppcon::demo::v0::TerminateAt
std::_Rb_tree<unsigned int, std::pair<unsigned int const, std::pair<unsigned int, cppcon::EdgePr
std::pair<s
cppcon::demo::v0::Graph::for_each_edge<cppcon::se
std::multimap<unsigned int, std::pair<unsigned int, cppcon::EdgeProperties>, std::less<unsigned
cppcon::de…
void cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::Terminate
bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int)
void cppcon::demo::run<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph, cppcon::demo::v0::run(std::filesystem::__cxx11::path const&, std::filesystem::__cxx11::path const&, cppcon::
main
__libc_start_main
_start
6.15103e+08 aggregated sample costs in total
6.015E+08 (97.8%) aggregated sample costs in bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int) (run_de...


perf.data.v0 - Hotspot
File Settings View Help
Summary Bottom Up Top Down Flame Graph Caller / Callee
cache-misses
Bottom-Up View
Collapse Recursion
Cost Threshold: 0.20%
Search...
_int_malloc
__GI___libc_malloc
operator new(unsigned long)
__gnu_cxx::new_allocator<std::_Rb_tree_node<std::pair<unsigned int const, unsigned int> > >::allo
std::allocator_traits<std::allocator<std::_Rb_tree_node<std::pair<unsigned int const, unsigned in
std::_Rb_tree<unsigned int, std::pair<unsigned int const, unsigned int>, std::_Select1st<std::pa
std::_Rb_tree_node<std::pair<unsigned int const, unsigned int> >* std::_Rb_tree<unsigned int, st
std::_…
std::_R…
std::_Rb_tree_insert_and_rebalanc
std::_Rb_tree<unsigned int, std::pair<unsigned int const
std::_Rb_tree<unsigned int, std::pai
std::pair<std::_Rb_tree_iterator<std::pair<unsigned int const, unsigned int> >, bool> std::_Rb_tree<unsigned int, std::pair<unsigned int const, unsigned int>, std::_Select1st<std::pair<unsign
std::pair<std::_Rb_tree_iterator<std::pair<unsigned int const, unsigned int> >, bool> std::map<unsigned int, unsigned int, std::less<unsigned int>, std::allocator<std::pair<unsigned int const
cppcon::demo::v0::TerminateAtGoal::mark_visited(unsigned int, unsigned int)
bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int)
void cppcon::demo::run<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph, cppcon::demo::v0::run(std::filesystem::__cxx11::path const&, std::filesystem::__cxx11::path const&, cppcon::
main
__libc_start_main
_start
6.15103e+08 aggregated sample costs in total
3.855E+07 (6.27%) aggregated sample costs in cppcon::demo::v0::TerminateAtGoal::mark_visited(unsigned int, unsigned int) (run_demo) and below.


perf.data.v0 - Hotspot
File Settings View Help
Summary Bottom Up Top Down Flame Graph Caller / Callee
cache-misses
Bottom-Up View
Collapse Recursion
Cost Threshold: 0.20%
Search...
std::_Rb_tree<unsigned int, std::pair<unsigned int c
std::_Rb_tree<unsigned int, std::pair<unsigned int co
std::map<unsigned int, unsigned int, std::less<unsigne
cppcon::demo::v0::TerminateAtGoal::is_visited(unsigned
std::_Rb_tree<unsigned int, std::pair<unsigned int co
cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::
std::_Rb_tree<unsigned int, std::pair<unsigned int const, std::pair<unsigned int, cppcon::EdgeProperties> >, std::_Select1st<s
cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::de
std::multimap<unsigned int, std::pair<unsigned int, cppcon::EdgeProperties>, std::less<unsigned int>, std::allocator<std::pair
void cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsi
bool cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo::v0::Graph const&, unsigned int)
void cppcon::demo::run<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph, cppcon::demo::v0::run(std::filesystem::__cxx11::path const&, std::filesystem::__cxx11::path const&, cppcon::
main
__libc_start_main
_start
6.15103e+08 aggregated sample costs in total
4.597E+08 (74.7%) aggregated sample costs in void cppcon::demo::v0::Graph::for_each_edge<cppcon::search<cppcon::demo::v0::TerminateAtGoal, cppcon::demo::v0::Graph>(cppcon::demo::v0::TerminateAtGoal&, cppcon::demo...

Achieving good (data) cache performance

Achieving good (data) cache performance

Spatial Locality

Achieving good (data) cache performance

Spatial Locality

- Data that we access should be close together in memory.

Achieving good (data) cache performance

Temporal Locality

Achieving good (data) cache performance

Temporal Locality

- Maximize the number of times we access data in the cache.

Achieving good (data) cache performance

Rules of thumb

Achieving good (data) cache performance

Rules of thumb

- Access data sequentially.

Achieving good (data) cache performance

Rules of thumb

- Access data sequentially.
- Only access data that we need.

Achieving good (data) cache performance

Rules of thumb

- Access data sequentially.
- Only access data that we need.
- Reduce indirections through pointers.

Achieving good (data) cache performance

Rules of thumb

- Access data sequentially.
- Only access data that we need.
- Reduce indirections through pointers.
- Don't allocate memory during execution.

Achieving good (data) cache performance

Does our first implementation do any of these things?

Achieving good (data) cache performance

Does our first implementation do any of these things?

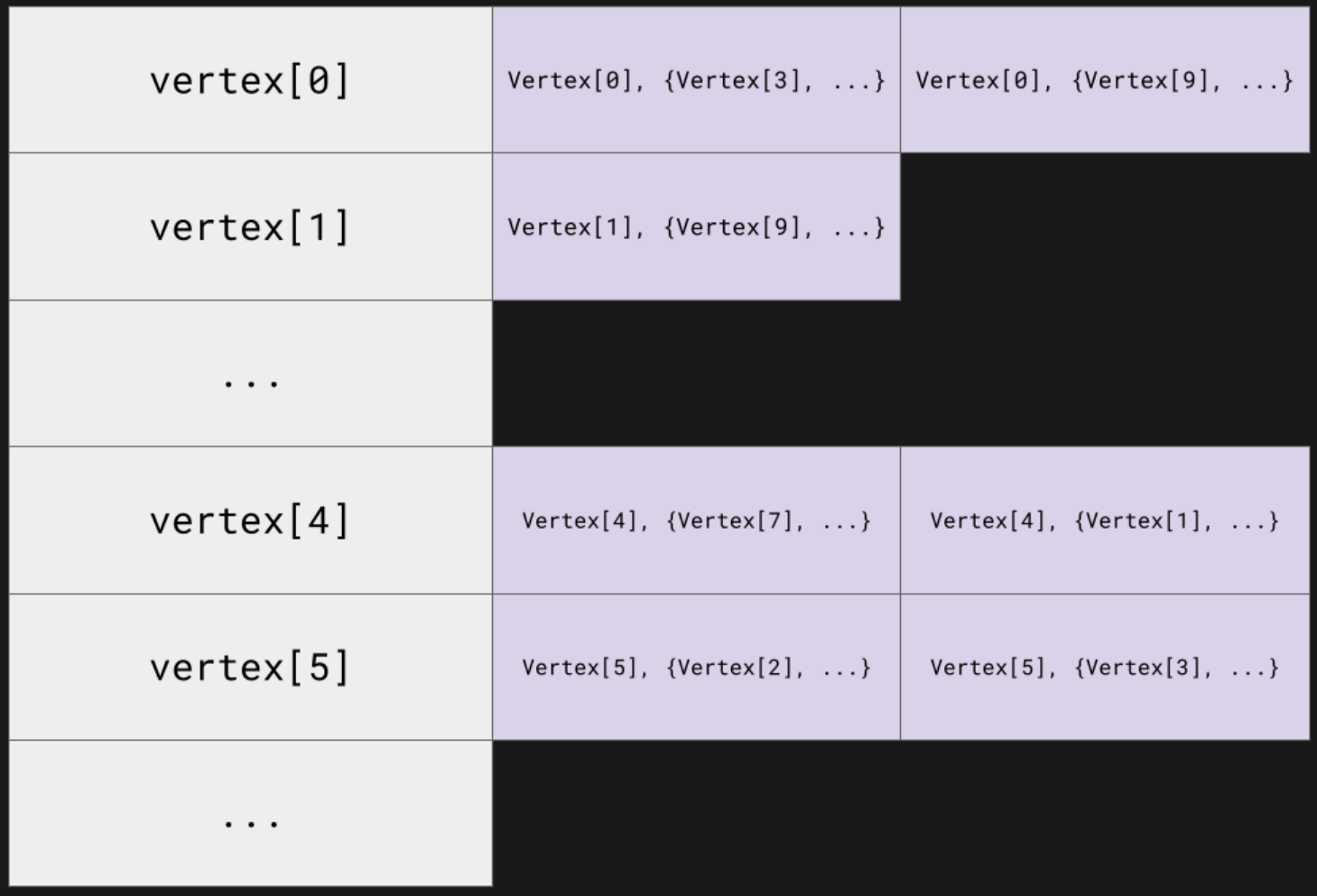
No not really.

Let's take a closer look

`std::multimap`

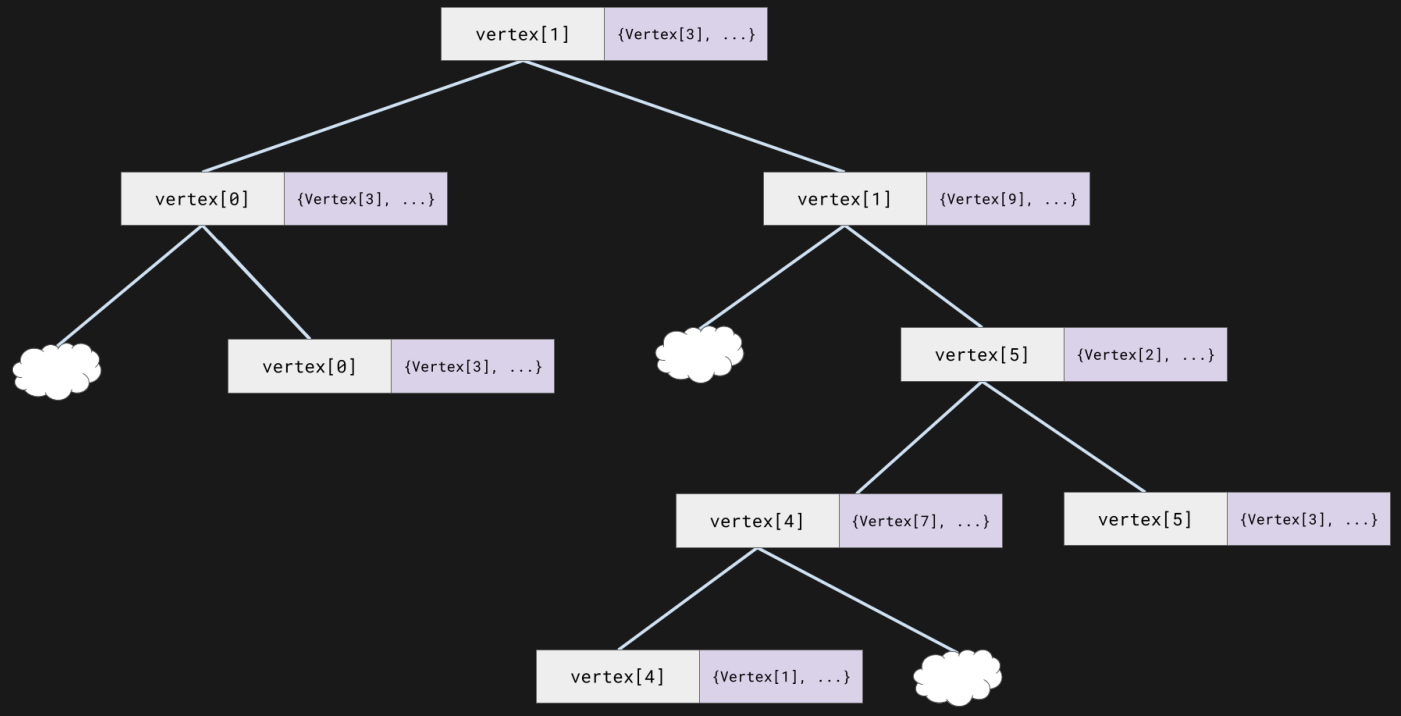
Let's take a closer look

std::multimap



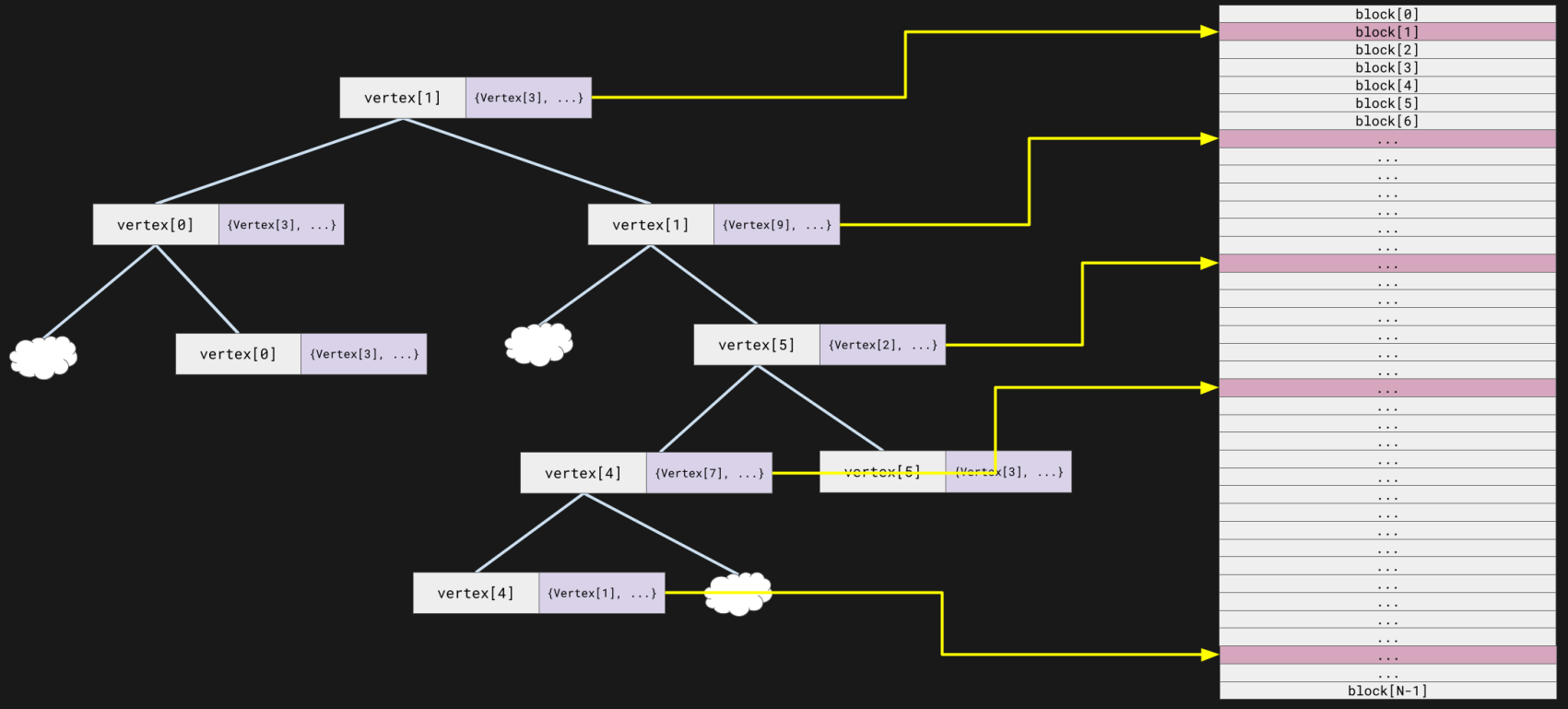
Let's take a closer look

std::multimap



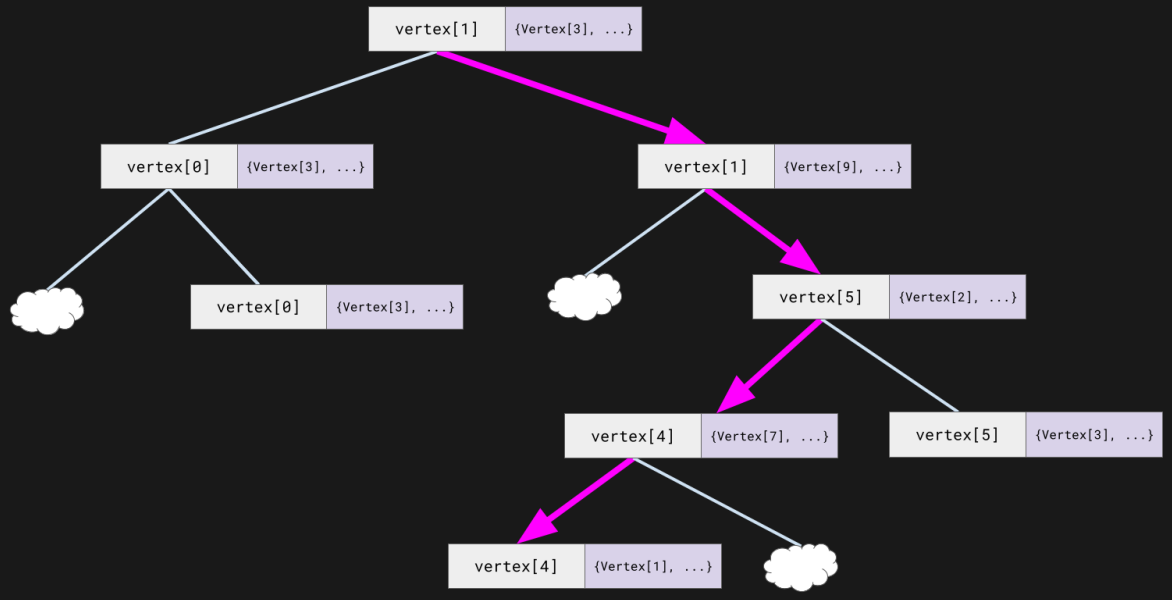
Let's take a closer look

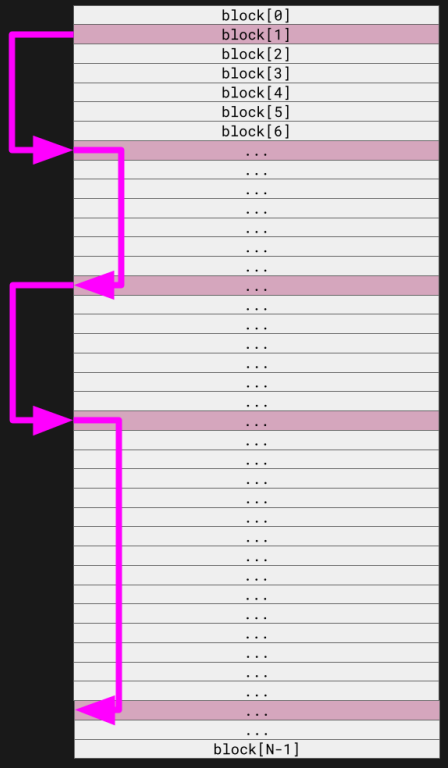
std::multimap



Let's take a closer look

std::multimap::equal_range(4)





Implementation 1

```cpp
#include <algorithm>     // std::for_each
#include <tuple>         // std::apply
#include <unordered_map> // std::unordered_map, std::unordered_multimap

class Graph
{
private:
  std::unordered_multimap<VertexID, Edge> adjacencies_;

  std::unordered_map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
```

```
class Graph
{
private:
  std::unordered_multimap<VertexID, Edge> adjacencies_;

  std::unordered_map<VertexID, VertexProperties> vertices_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_.at(q); }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    const auto [first, last] = adjacencies_.equal_range(q);
    std::for_each(
      first,
      last,
      [visitor](const auto& parent_and_edge) mutable
      {
        std::apply(visitor, parent_and_edge.second);
      });
  }
};
```

```cpp
#include <functional>
#include <queue>
#include <unordered_map>
#include <vector>

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::unordered_map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
```

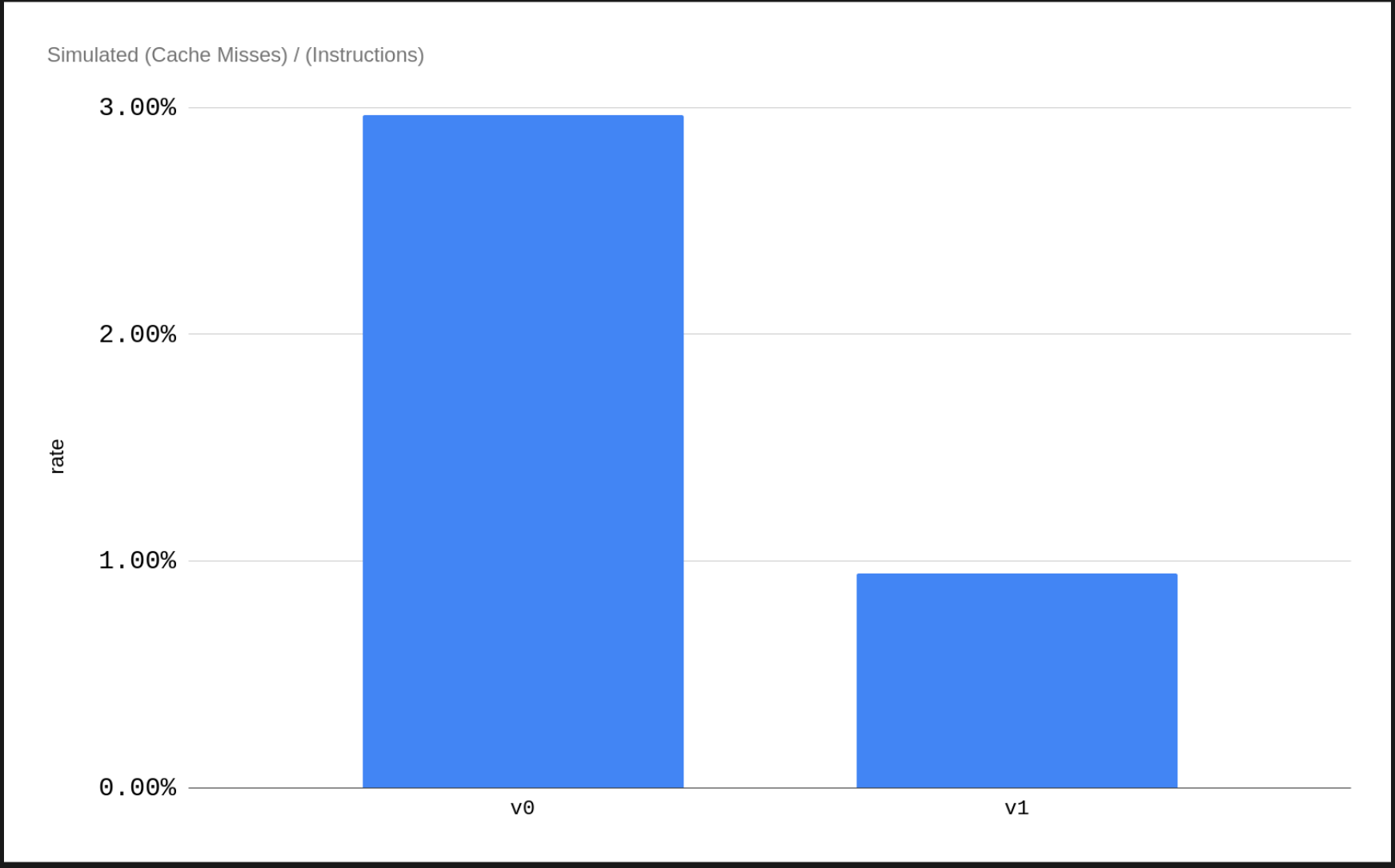
Measuring cache performance

```
==132432== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==132432== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
…
==132432== I   refs:      11,574,779,293
==132432== I1  misses:             3,778
==132432== LLi misses:             3,700
==132432== I1  miss rate:           0.00%
==132432== LLi miss rate:           0.00%
==132432==
==132432== D   refs:       4,525,697,105  (2,946,691,588 rd   + 1,579,005,517 wr)
==132432== D1  misses:       109,346,888  (  103,780,511 rd   +     5,566,377 wr)
==132432== LLd misses:         3,231,301  (    2,205,723 rd   +     1,025,578 wr)
==132432== D1  miss rate:            2.4% (          3.5%     +           0.4%  )
==132432== LLd miss rate:            0.1% (          0.1%     +           0.1%  )
==132432==
==132432== LL refs:          109,350,666  (  103,784,289 rd   +     5,566,377 wr)
==132432== LL misses:          3,235,001  (    2,209,423 rd   +     1,025,578 wr)
==132432== LL miss rate:             0.0% (          0.0%     +           0.1%  )
```

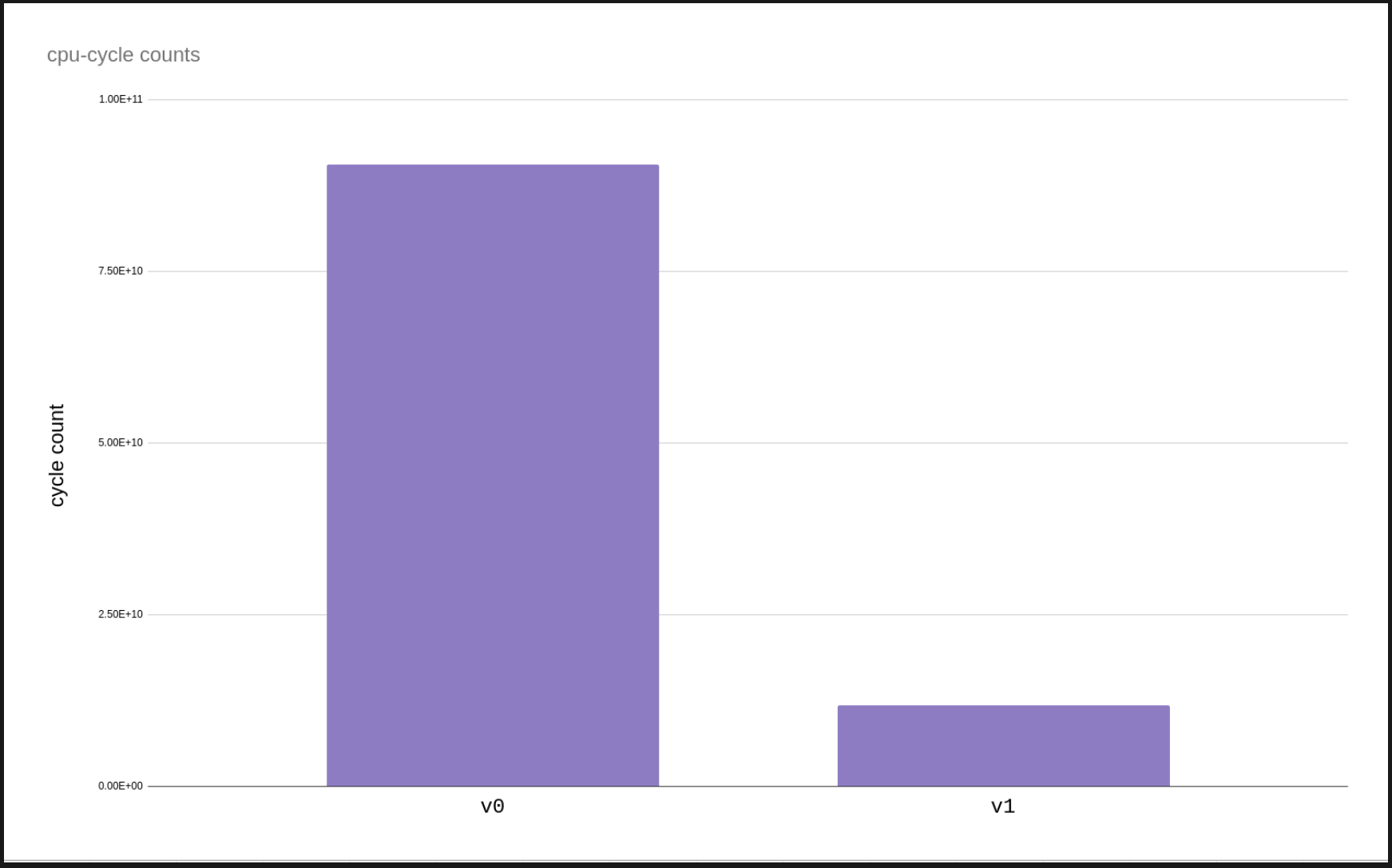
Measuring cache performance

```
==132432== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==132432== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
…
==132432== I   refs:      11,574,779,293
==132432== I1  misses:             3,778
==132432== LLi misses:             3,700
==132432== I1  miss rate:           0.00%
==132432== LLi miss rate:           0.00%
==132432==
==132432== D   refs:       4,525,697,105  (2,946,691,588 rd   + 1,579,005,517 wr)
==132432== D1  misses:       109,346,888  (  103,780,511 rd   +     5,566,377 wr)
==132432== LLd misses:         3,231,301  (    2,205,723 rd   +     1,025,578 wr)
==132432== D1  miss rate:            2.4% (          3.5%     +           0.4%  )
==132432== LLd miss rate:            0.1% (          0.1%     +           0.1%  )
==132432==
==132432== LL refs:          109,350,666  (  103,784,289 rd   +     5,566,377 wr)
==132432== LL misses:          3,235,001  (    2,209,423 rd   +     1,025,578 wr)
==132432== LL miss rate:             0.0% (          0.0%     +           0.1%  )
```

Measuring cache performance

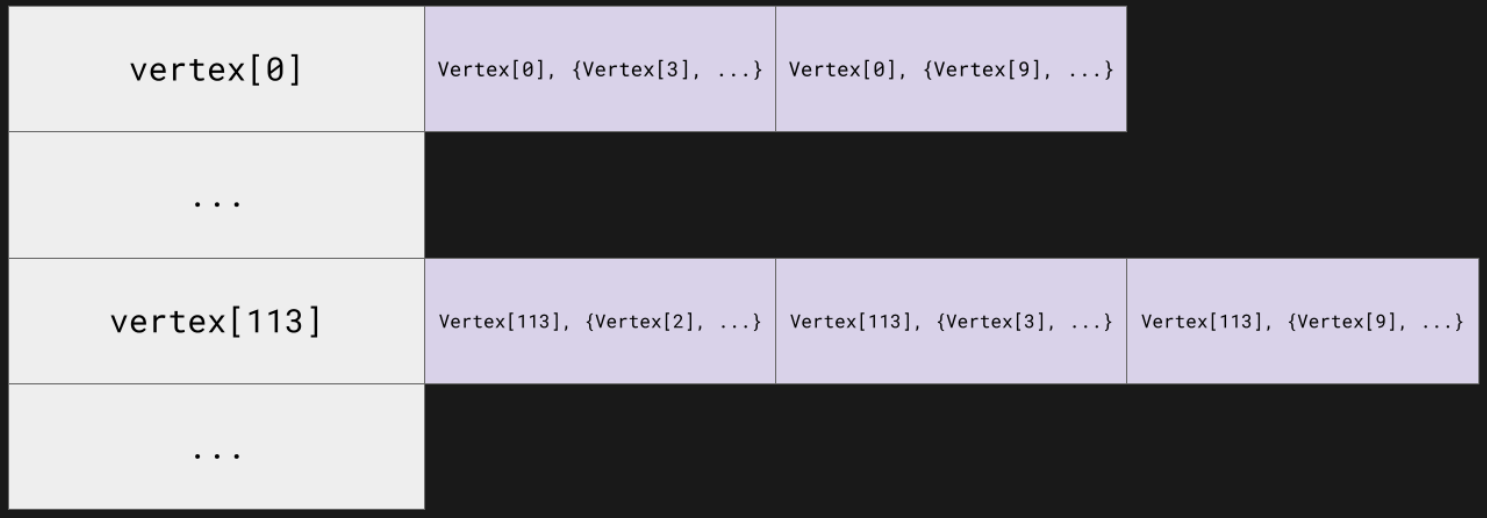


Measuring cache performance



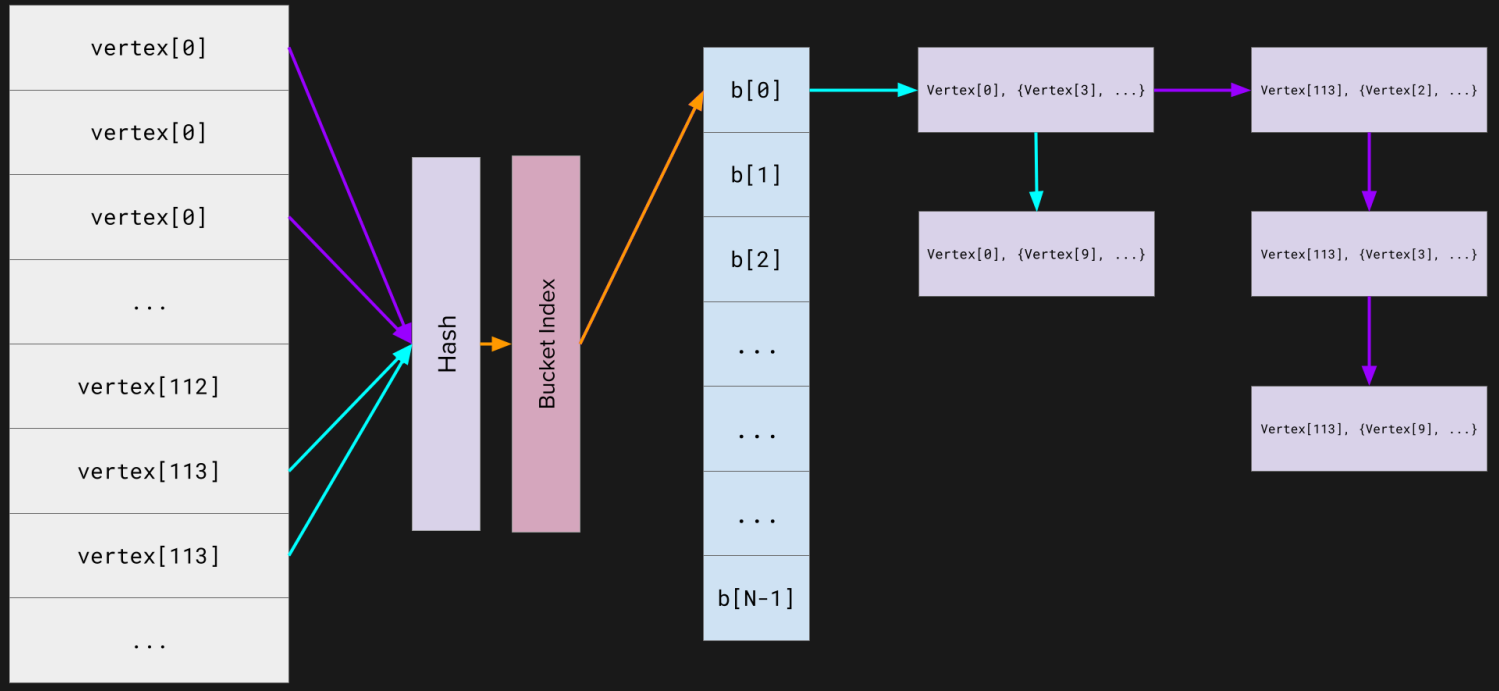
Measuring cache performance

std::unordered_multimap



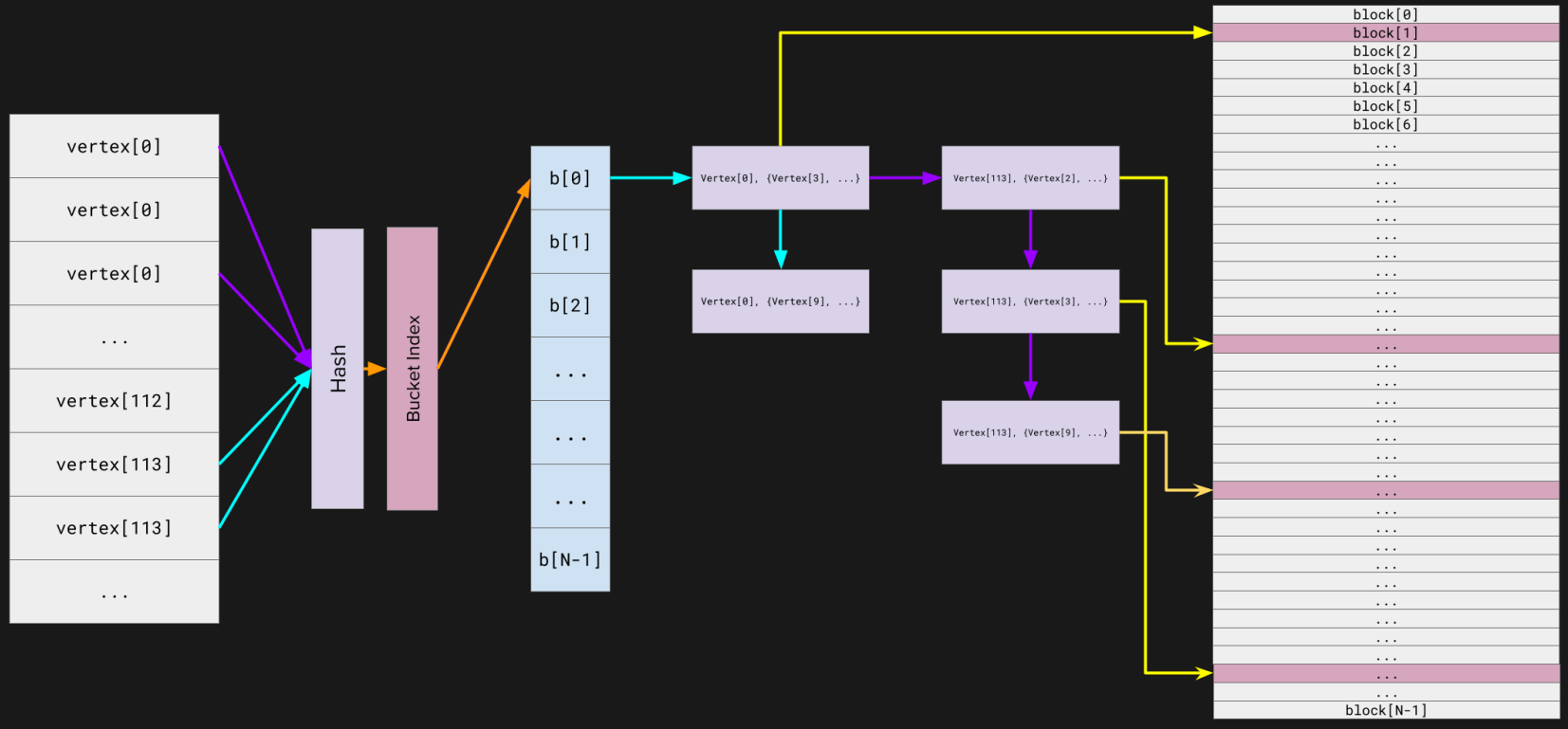
Measuring cache performance

std::unordered_multimap



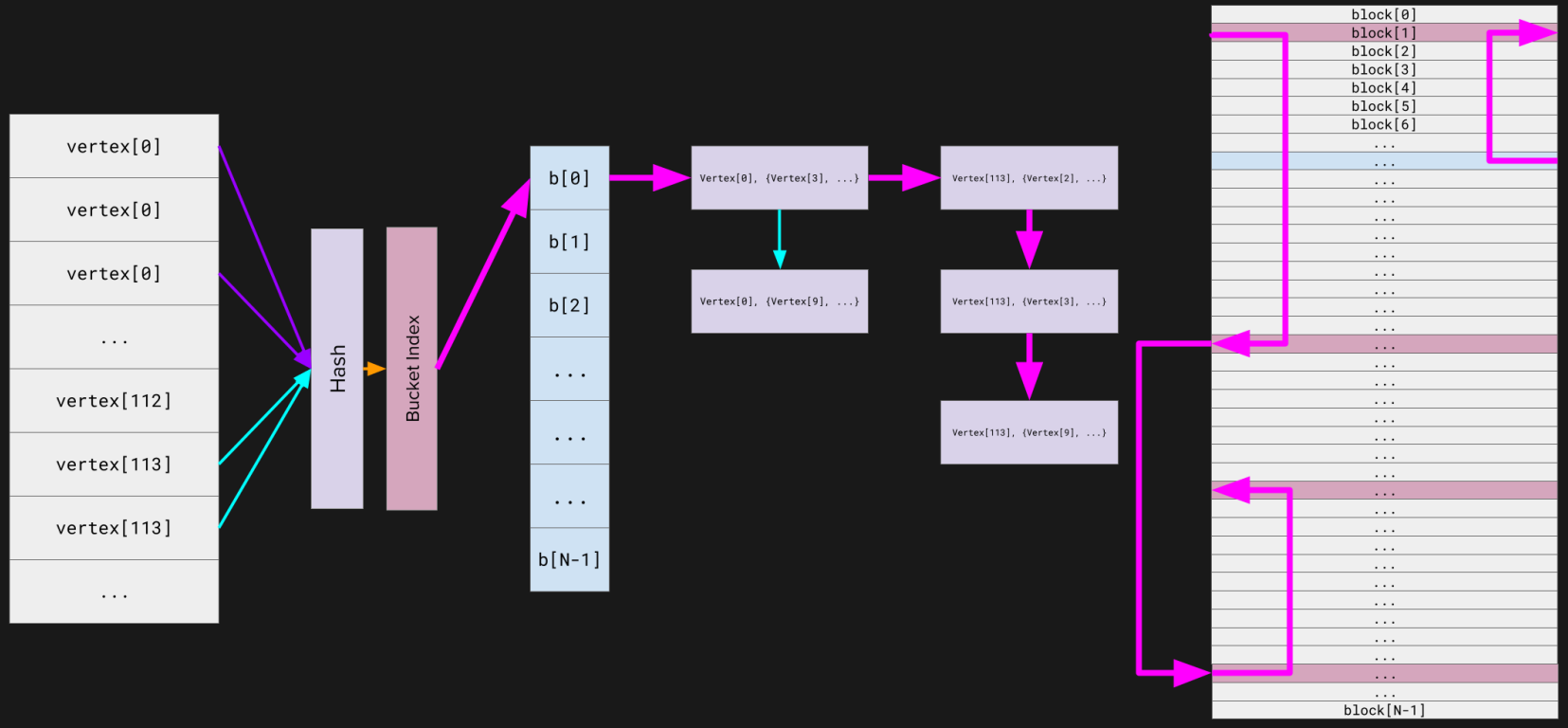
Measuring cache performance

std::unordered_multimap



Measuring cache performance

`std::unordered_multimap::equal_range(113)`

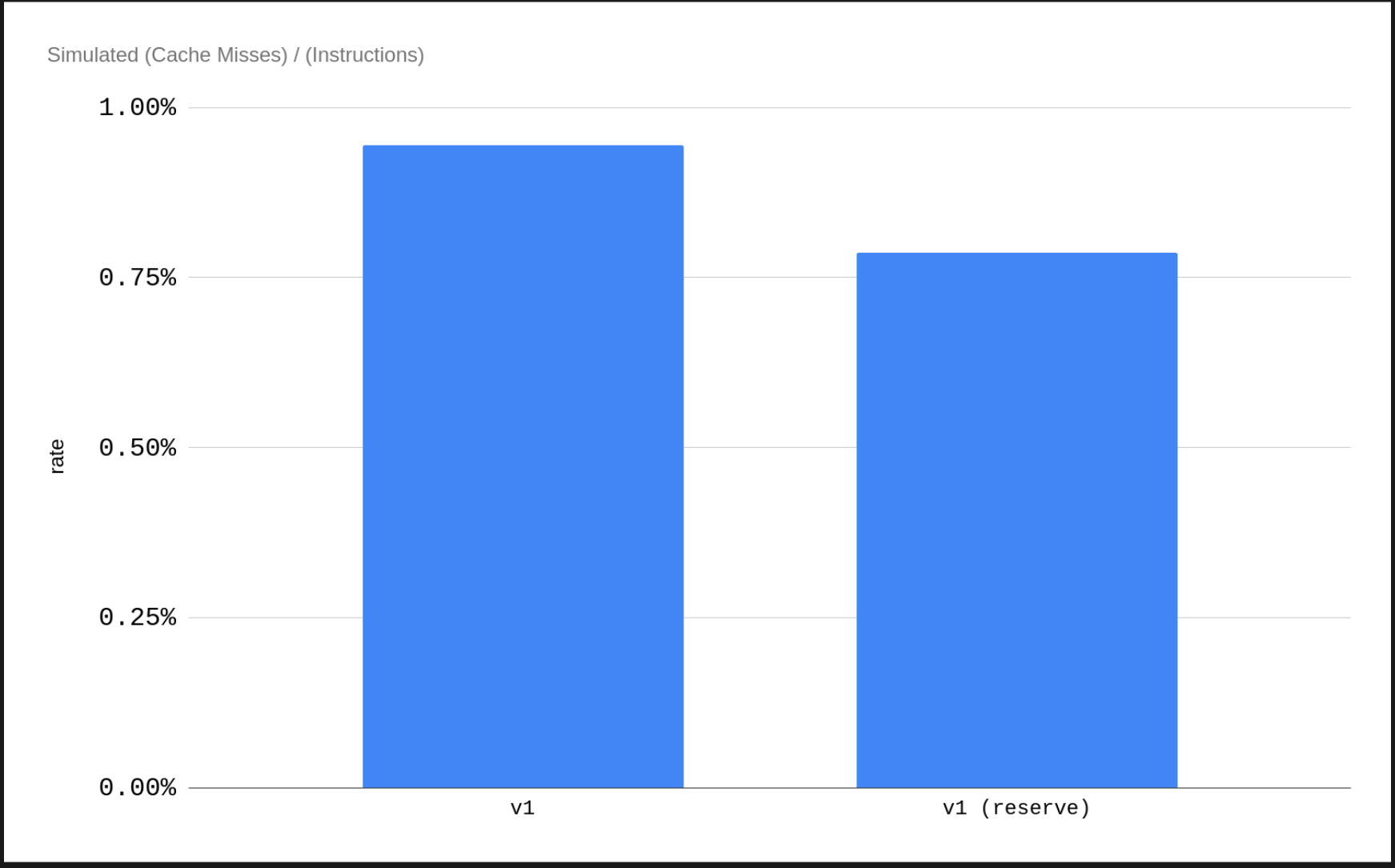


```cpp
#include <functional>     // std::greater
#include <queue>          // std::priority_queue
#include <unordered_map>  // std::unordered_map
#include <vector>         // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::unordered_map<VertexID, VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.clear();
    visited_.reserve(graph.vertex_count());
    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
```

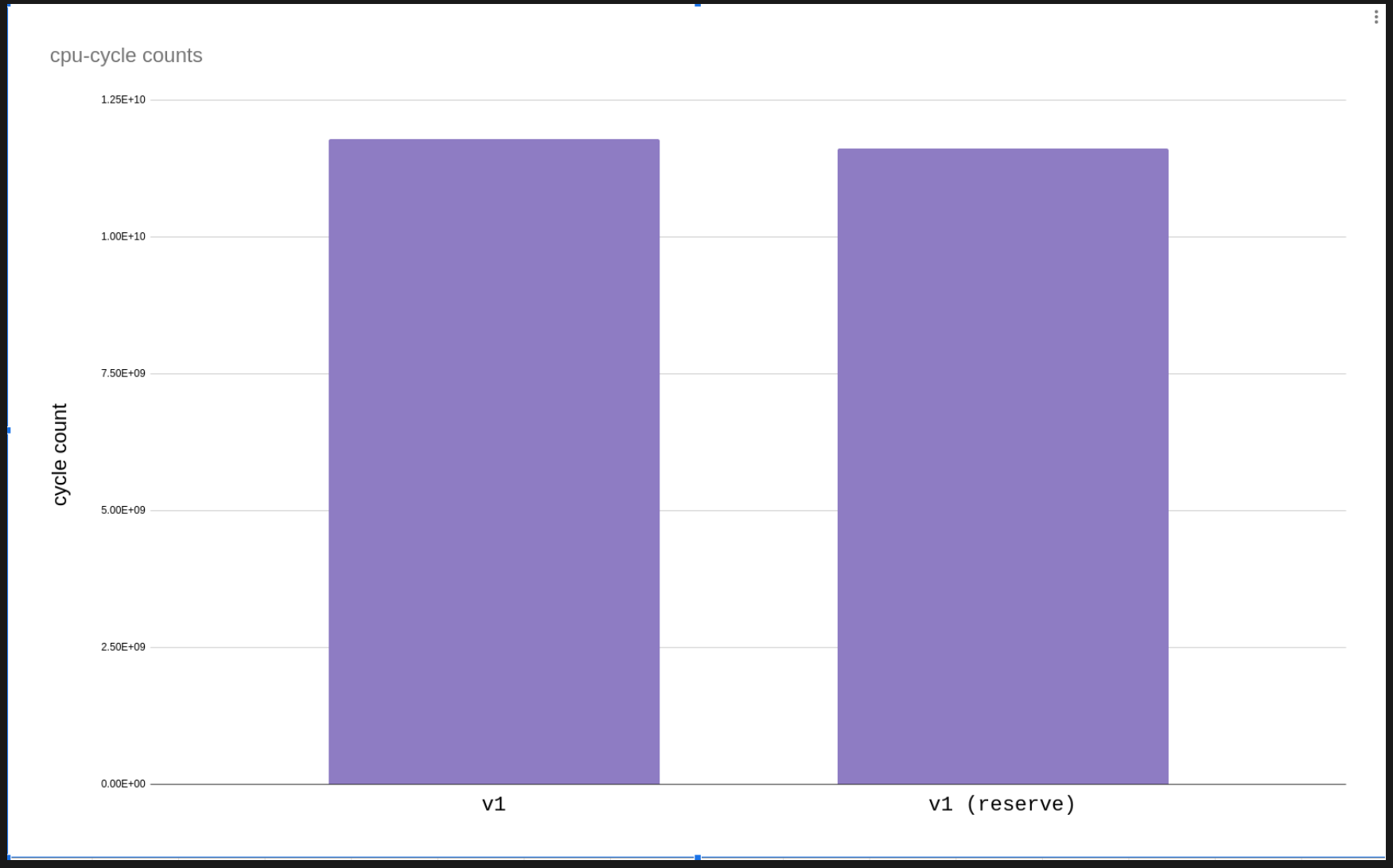
Measuring cache performance

```
==133350== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==133350== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
…
==133350== I   refs:      11,574,779,289
==133350== I1  misses:             6,017
==133350== LLi misses:             3,935
==133350== I1  miss rate:           0.00%
==133350== LLi miss rate:           0.00%
==133350==
==133350== D   refs:       4,525,697,103  (2,946,691,588 rd   + 1,579,005,515 wr)
==133350== D1  misses:        90,999,080  (   85,486,438 rd   +     5,512,642 wr)
==133350== LLd misses:         3,231,302  (    2,205,724 rd   +     1,025,578 wr)
==133350== D1  miss rate:            2.0% (          2.9%     +           0.3%  )
==133350== LLd miss rate:            0.1% (          0.1%     +           0.1%  )
==133350==
==133350== LL refs:           91,005,097  (   85,492,455 rd   +     5,512,642 wr)
==133350== LL misses:          3,235,237  (    2,209,659 rd   +     1,025,578 wr)
==133350== LL miss rate:             0.0% (          0.0%     +           0.1%  )
```

Measuring cache performance

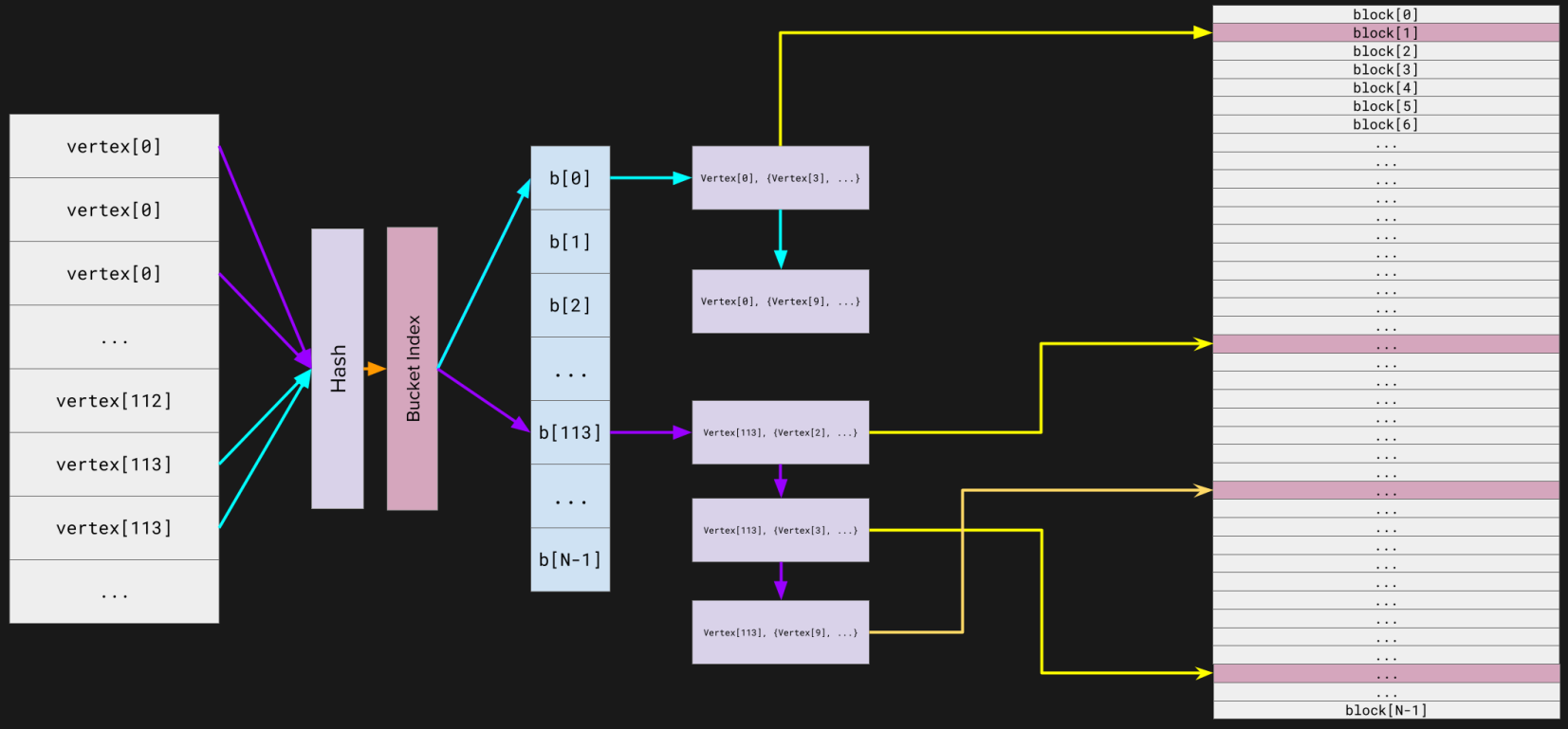


Measuring cache performance



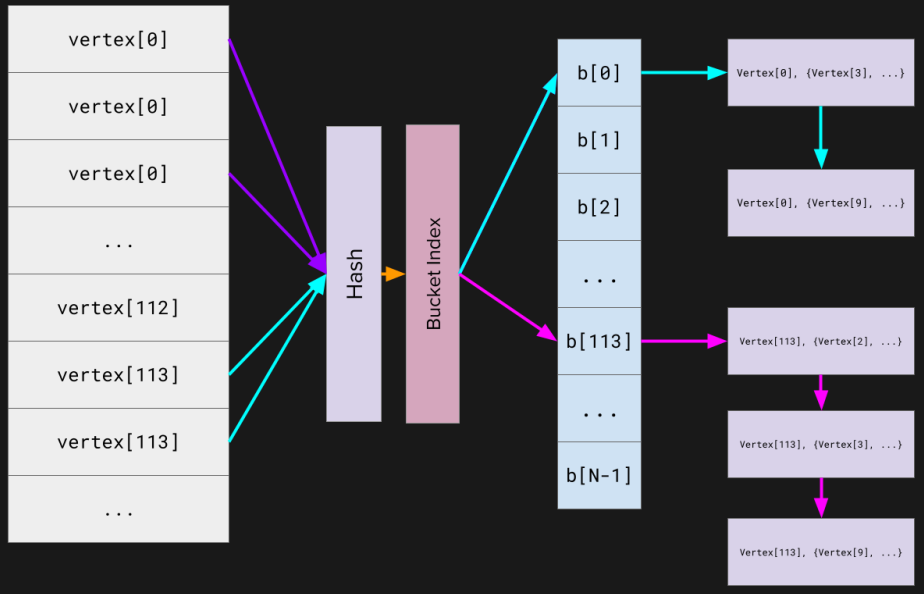
Measuring cache performance

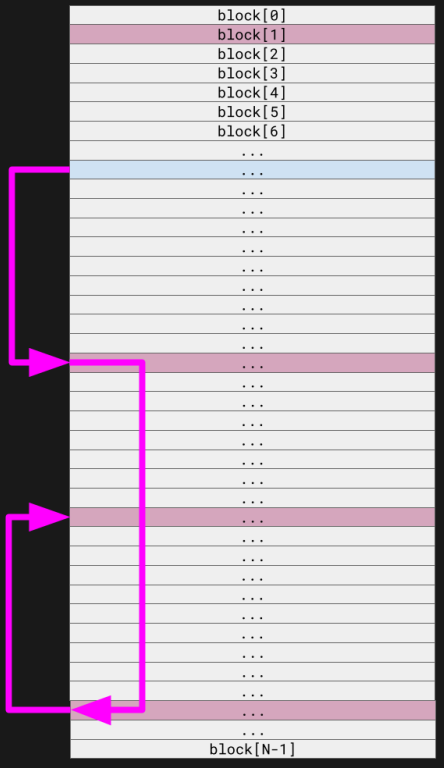
std::unordered_multimap



Measuring cache performance

`std::unordered_multimap::equal_range(113)`





Implementation 2

```cpp
#include <algorithm> // std::for_each
#include <tuple>     // std::apply
#include <vector>    // std::vector

class Graph
{
private:
  std::vector<VertexProperties> vertices_;

  std::vector<std::vector<Edge>> adjacencies_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_[q]; }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    std::for_each(
      adjacencies_[q].begin(),
      adjacencies_[q].end(),
      [q, visitor](const auto& child_and_edge_weight) mutable
      {
        const auto& [succ, edge_weight] = child_and_edge_weight;
```

```cpp
#include <algorithm> // std::for_each
#include <tuple>     // std::apply
#include <vector>    // std::vector

class Graph
{
private:
  std::vector<VertexProperties> vertices_;

  std::vector<std::vector<Edge>> adjacencies_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_[q]; }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    std::for_each(
      adjacencies_[q].begin(),
      adjacencies_[q].end(),
      [q, visitor](const auto& child_and_edge_weight) mutable
      {
        const auto& [succ, edge_weight] = child_and_edge_weight;
```

```
class Graph
{
private:
  std::vector<VertexProperties> vertices_;

  std::vector<std::vector<Edge>> adjacencies_;

public:
  const VertexProperties& vertex(VertexID q) const { return vertices_[q]; }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(VertexID q, EdgeVisitorT&& visitor) const
  {
    std::for_each(
      adjacencies_[q].begin(),
      adjacencies_[q].end(),
      [q, visitor](const auto& child_and_edge_weight) mutable
      {
        const auto& [succ, edge_weight] = child_and_edge_weight;
        visitor(succ, edge_weight);
      });
  }
};
```

```cpp
#include <functional> // std::greater
#include <queue>      // std::priority_queue
#include <vector>     // std::vector

class TerminateAtGoal
{
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::vector<VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.resize(graph.vertex_count());
    visited_.assign(graph.vertex_count(), graph.vertex_count());

    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
```

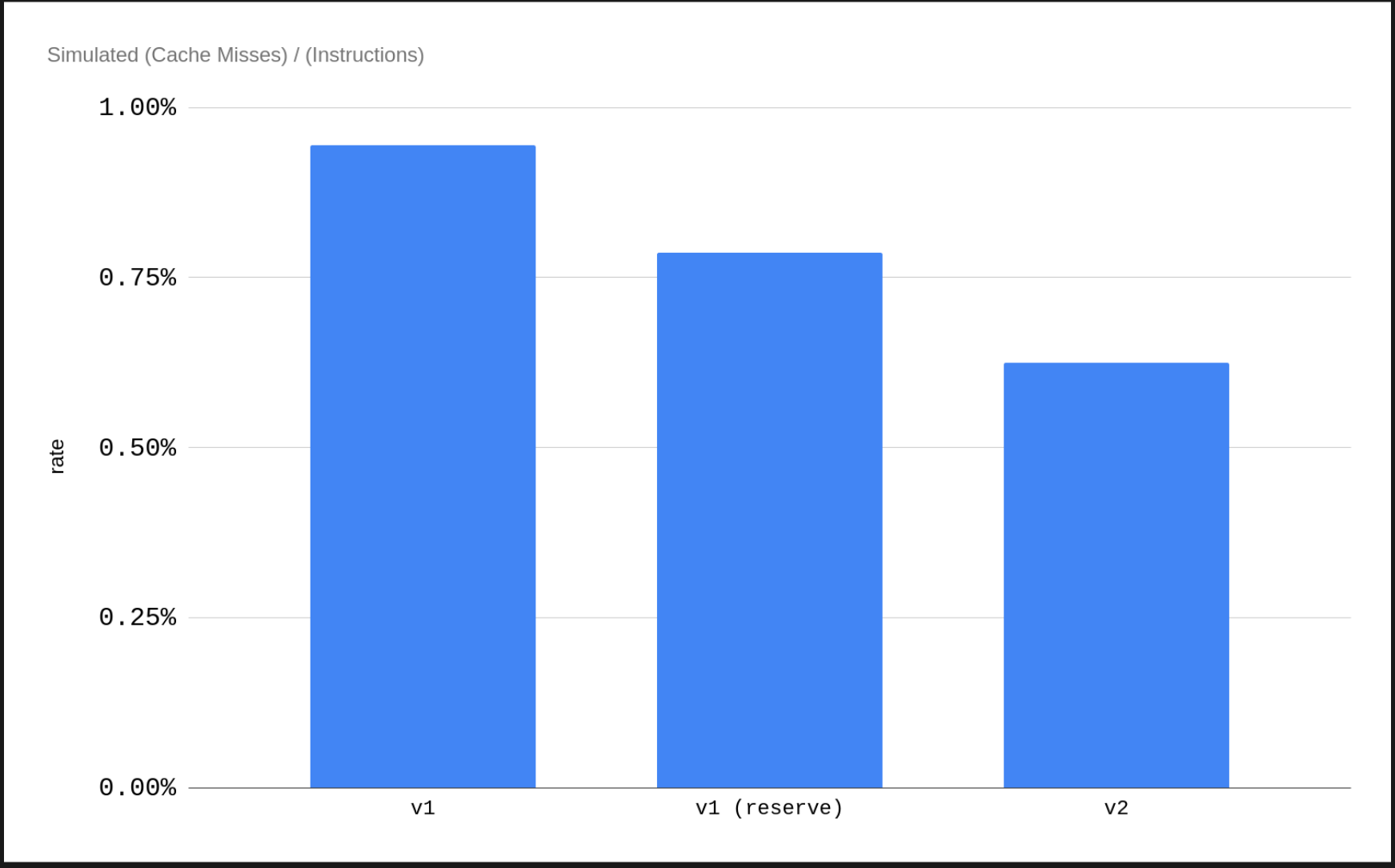
```
private:
  VertexID goal_;

  std::priority_queue<Transition, std::vector<Transition>, std::greater<Transition>> queue_;

  std::vector<VertexID> visited_;

public:
  template<SearchGraph G>
  void reset(G&& graph, VertexID s, VertexID g)
  {
    // Clear visited set
    visited_.resize(graph.vertex_count());
    visited_.assign(graph.vertex_count(), graph.vertex_count());

    // Clear any stragglers in the queue
    while (!queue_.empty()) { queue_.pop(); }
    // Set current goal
    goal_ = g;
    // Add start as first vertex in queue
    enqueue(s, s, 0);
  }

  bool is_queue_not_empty() const { return !queue_.empty(); }

  bool is_visited(VertexID g) const { return g < visited_.size(); }
```

```
22      // Clear any stragglers in the queue
23      while (!queue_.empty()) { queue_.pop(); }
24      // Set current goal
25      goal_ = g;
26      // Add start as first vertex in queue
27      enqueue(s, s, 0);
28    }
29
30    bool is_queue_not_empty() const { return !queue_.empty(); }
31
32    bool is_visited(VertexID q) const { return q != visted_.size(); }
33
34    bool is_terminal(VertexID q) const { return goal_ == q; }
35
36    void mark_visited(VertexID from, VertexID to) { visited_[to] = from; }
37
38    VertexID predecessor(VertexID q) const
39    {
40      return visited_[q];
41    }
42
43    Transition dequeue()
44    {
45      auto t = queue_.top();
46      queue_.pop();
```

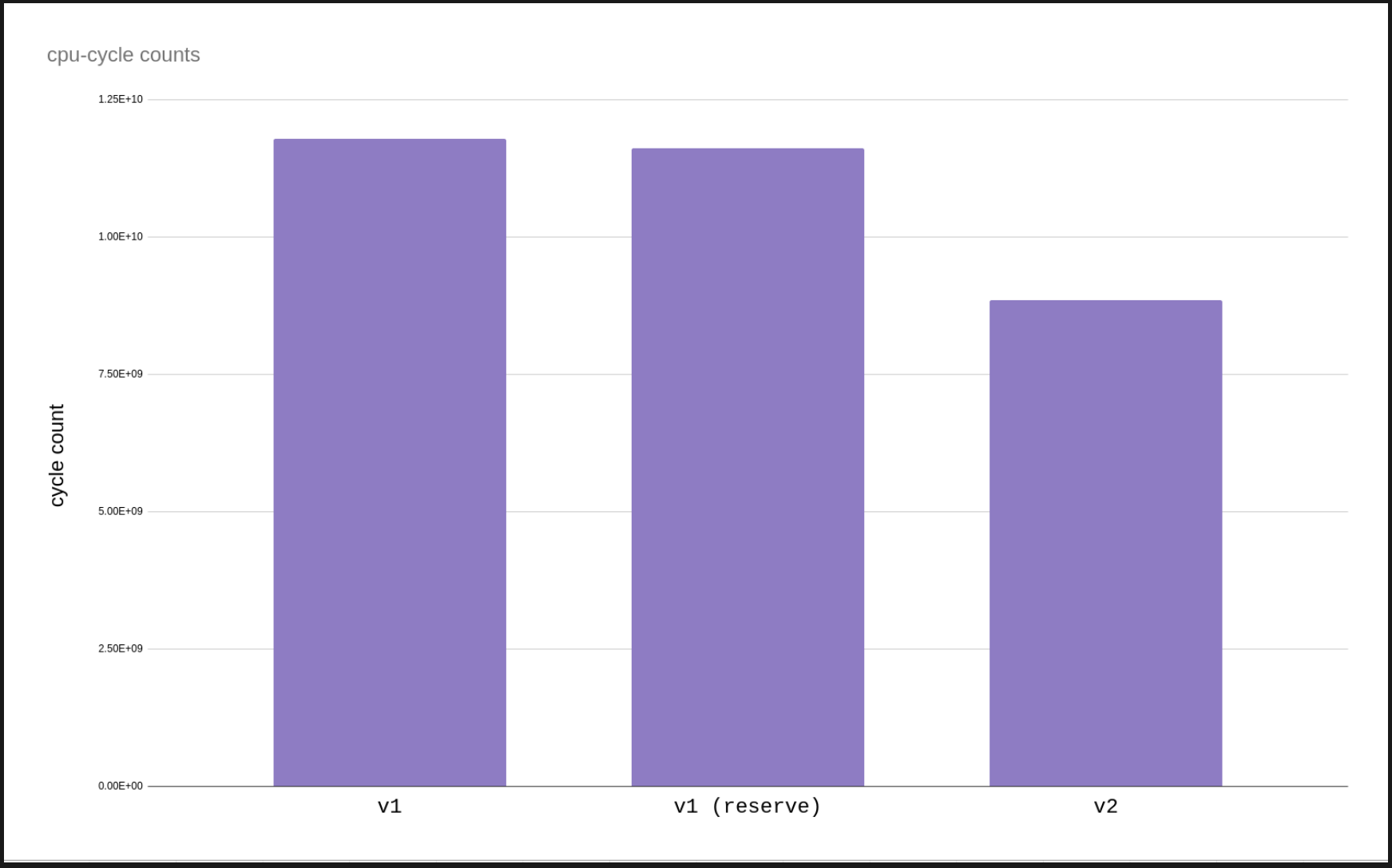
Measuring cache performance

```
==135295== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==135295== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
…
==135295== I   refs:      6,919,489,865
==135295== I1  misses:            6,058
==135295== LLi misses:            3,887
==135295== I1  miss rate:          0.00%
==135295== LLi miss rate:          0.00%
==135295==
==135295== D   refs:      2,516,605,627  (1,700,911,074 rd   + 815,694,553 wr)
==135295== D1  misses:       43,267,746  (   40,602,167 rd   +   2,665,579 wr)
==135295== LLd misses:        3,054,946  (    2,183,488 rd   +     871,458 wr)
==135295== D1  miss rate:           1.7% (          2.4%     +         0.3%  )
==135295== LLd miss rate:           0.1% (          0.1%     +         0.1%  )
==135295==
==135295== LL refs:          43,273,804  (   40,608,225 rd   +   2,665,579 wr)
==135295== LL misses:         3,058,833  (    2,187,375 rd   +     871,458 wr)
==135295== LL miss rate:            0.0% (          0.0%     +         0.1%  )
```

Measuring cache performance

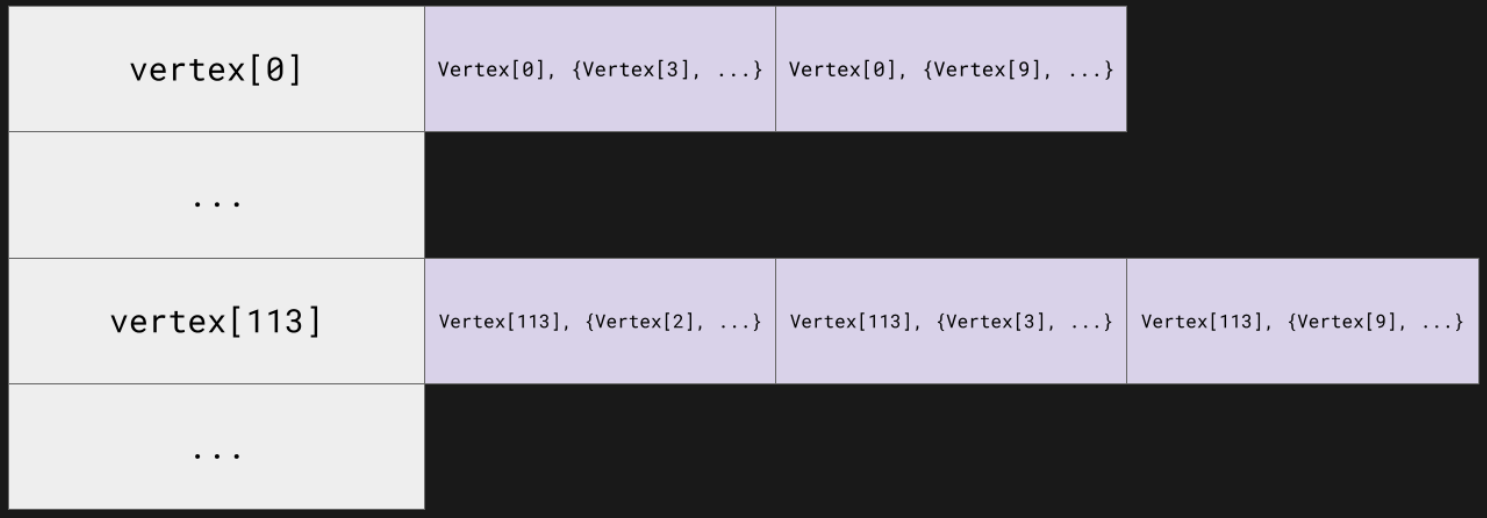


Measuring cache performance



Let's take a closer look

`std::vector::operator[] -> std::vector`



Let's take a closer look

`std::vector::operator[] -> std::vector`

Impact of ordering of the graph

```cpp
#include <algorithm>  // std::for_each
#include <ranges>     // std::ranges::subrange
#include <vector>     // std::vector

class Graph
{
private:
  std::vector<VertexProperties> vertices_;
  std::vector<std::ranges::subrange<const Edge*>> adjacencies_;
  std::vector<Edge> edges_;

public:
  const VertexProperties& vertex(vertex_id_t q) const { return vertices_[q]; }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(vertex_id_t q, EdgeVisitorT&& visitor) const
  {
    std::for_each(
      adjacencies_[q].begin(),
      adjacencies_[q].end(),
      [q, visitor](const auto& child_and_edge_weight) mutable
      {
        const auto& [succ, edge_weight] = child_and_edge_weight;
```

```cpp
#include <algorithm>  // std::for_each
#include <ranges>     // std::ranges::subrange
#include <vector>     // std::vector

class Graph
{
private:
  std::vector<VertexProperties> vertices_;
  std::vector<std::ranges::subrange<const Edge*>> adjacencies_;
  std::vector<Edge> edges_;

public:
  const VertexProperties& vertex(vertex_id_t q) const { return vertices_[q]; }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(vertex_id_t q, EdgeVisitorT&& visitor) const
  {
    std::for_each(
      adjacencies_[q].begin(),
      adjacencies_[q].end(),
      [q, visitor](const auto& child_and_edge_weight) mutable
      {
        const auto& [succ, edge_weight] = child_and_edge_weight;
```

```cpp
#include <algorithm>  // std::for_each
#include <ranges>     // std::ranges::subrange
#include <vector>     // std::vector

class Graph
{
private:
  std::vector<VertexProperties> vertices_;
  std::vector<std::ranges::subrange<const Edge*>> adjacencies_;
  std::vector<Edge> edges_;

public:
  const VertexProperties& vertex(vertex_id_t q) const { return vertices_[q]; }

  std::size_t vertex_count() const { return vertices_.size(); }

  template<typename EdgeVisitorT>
  void for_each_edge(vertex_id_t q, EdgeVisitorT&& visitor) const
  {
    std::for_each(
      adjacencies_[q].begin(),
      adjacencies_[q].end(),
      [q, visitor](const auto& child_and_edge_weight) mutable
      {
        const auto& [succ, edge_weight] = child_and_edge_weight;
```

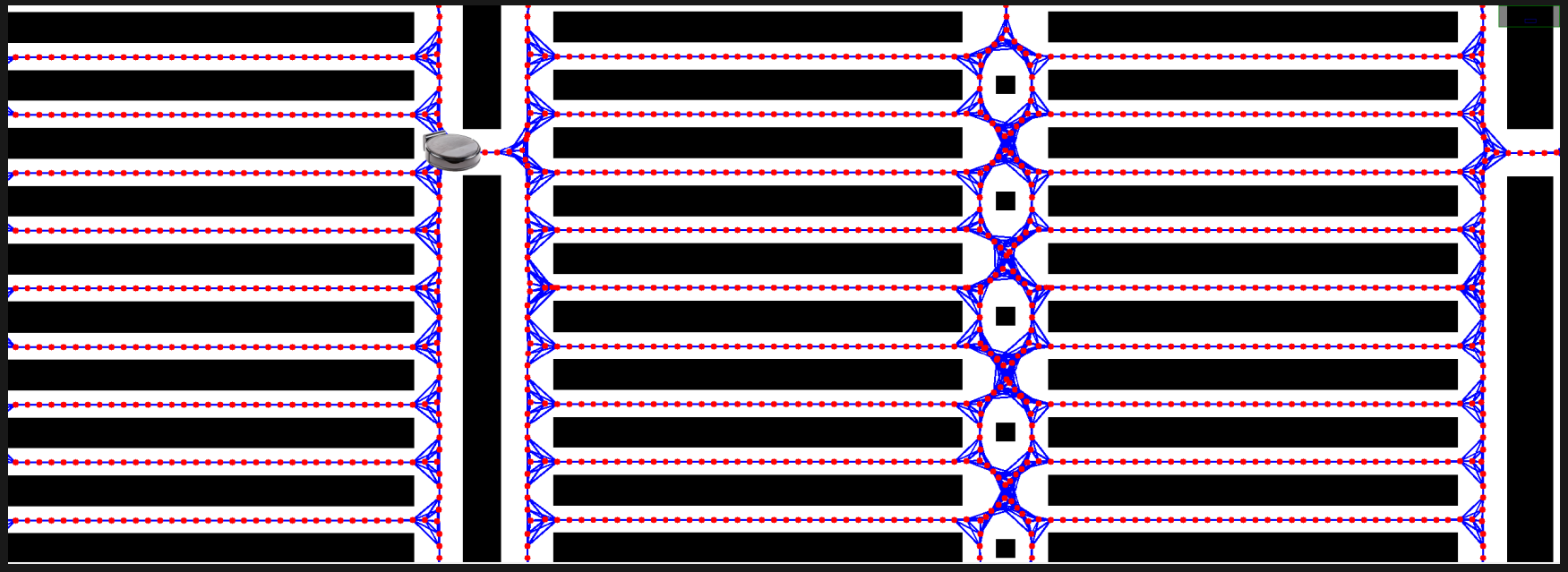
Impact of ordering of the graph

- We can re-order vertices and their adjacencies with `std::sort`

Impact of ordering of the graph

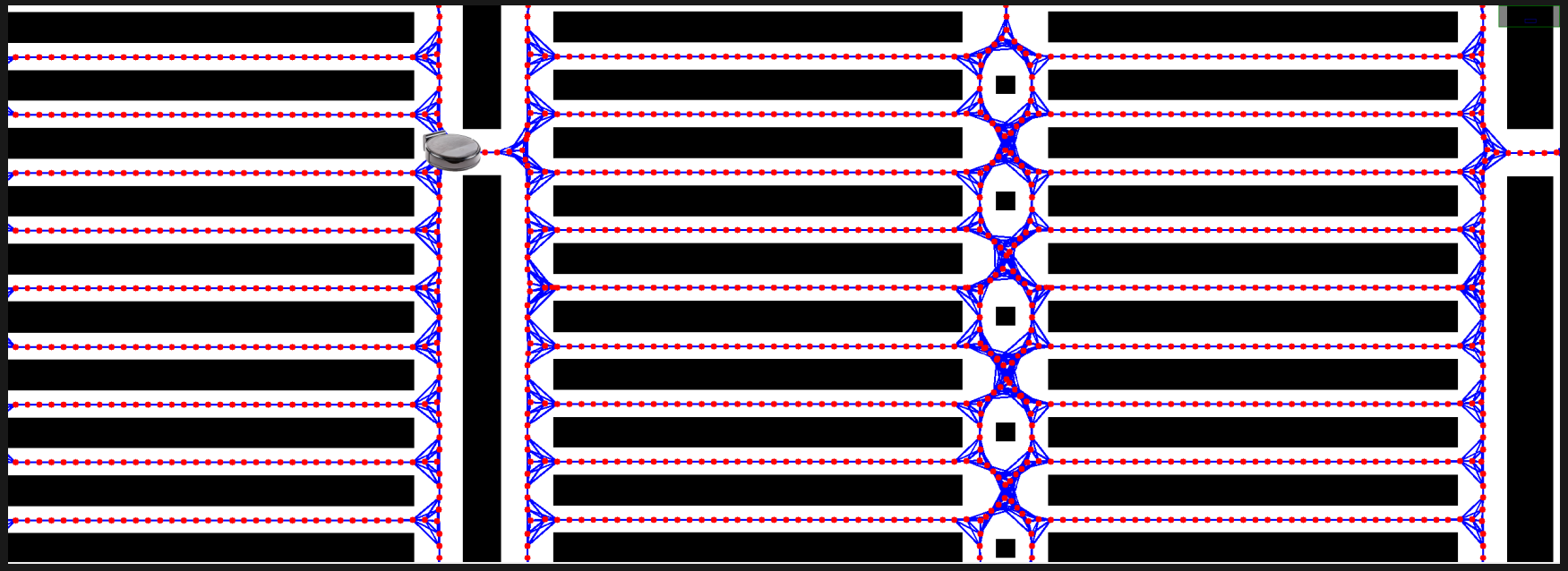
- We can re-order vertices and their adjacencies with std::sort
- If we can find a single sort-order which improves performance for most start and goal pairs, then its probably practical to do

Impact of ordering of the graph



- We see that vertices which are close in space are more likely to be neighbors

Impact of ordering of the graph



- We see that vertices which are close in space are more likely to be neighbors
- We can order vertex adjacency data in memory such that the probability of accessing nearby vertex data in the same cache line is higher

Impact of ordering of the graph

```
std::vector<std::size_t> remapping;
remapping.resize(graph.vertex_count());
std::iota(remapping.begin(), remapping.end(), 0);

std::sort(
  remapping.begin(),
  remapping.end(),
  [&graph](VertexID lhs, VertexID rhs) -> bool
  {
    const auto& lv = graph.vertex(lhs);
    const auto& rv = graph.vertex(rhs);
    return ((lv.x * lv.x) + (lv.y + lv.y)) < ((rv.x * rv.x) + (rv.y + rv.y));
  });

graph.remap(remapping);
```

Impact of ordering of the graph

```
1  std::vector<std::size_t> remapping;
2  remapping.resize(graph.vertex_count());
3  std::iota(remapping.begin(), remapping.end(), 0);
4
5  std::sort(
6    remapping.begin(),
7    remapping.end(),
8    [&graph](VertexID lhs, VertexID rhs) -> bool
9    {
10     const auto& lv = graph.vertex(lhs);
11     const auto& rv = graph.vertex(rhs);
12     return ((lv.x * lv.x) + (lv.y + lv.y)) < ((rv.x * rv.x) + (rv.y + rv.y));
13   });
14
15 graph.remap(remapping);
```

Impact of ordering of the graph

```
1  std::vector<std::size_t> remapping;
2  remapping.resize(graph.vertex_count());
3  std::iota(remapping.begin(), remapping.end(), 0);
4
5  std::sort(
6    remapping.begin(),
7    remapping.end(),
8    [&graph](VertexID lhs, VertexID rhs) -> bool
9    {
10     const auto& lv = graph.vertex(lhs);
11     const auto& rv = graph.vertex(rhs);
12     return ((lv.x * lv.x) + (lv.y + lv.y)) < ((rv.x * rv.x) + (rv.y + rv.y));
13   });
14
15 graph.remap(remapping);
```

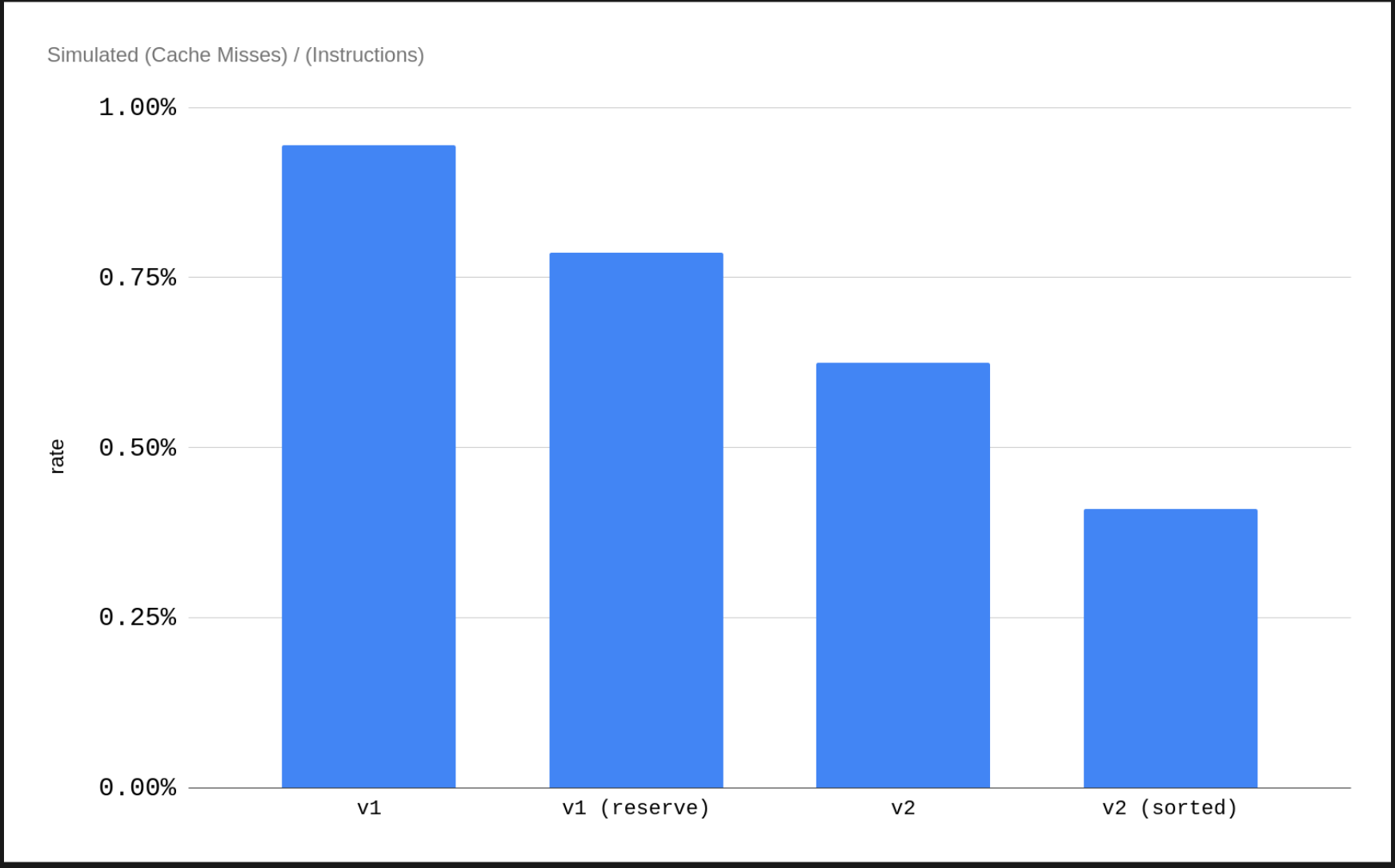
Impact of ordering of the graph

```
1  std::vector<std::size_t> remapping;
2  remapping.resize(graph.vertex_count());
3  std::iota(remapping.begin(), remapping.end(), 0);
4
5  std::sort(
6    remapping.begin(),
7    remapping.end(),
8    [&graph](VertexID lhs, VertexID rhs) -> bool
9    {
10     const auto& lv = graph.vertex(lhs);
11     const auto& rv = graph.vertex(rhs);
12     return ((lv.x * lv.x) + (lv.y + lv.y)) < ((rv.x * rv.x) + (rv.y + rv.y));
13   });
14
15 graph.remap(remapping);
```

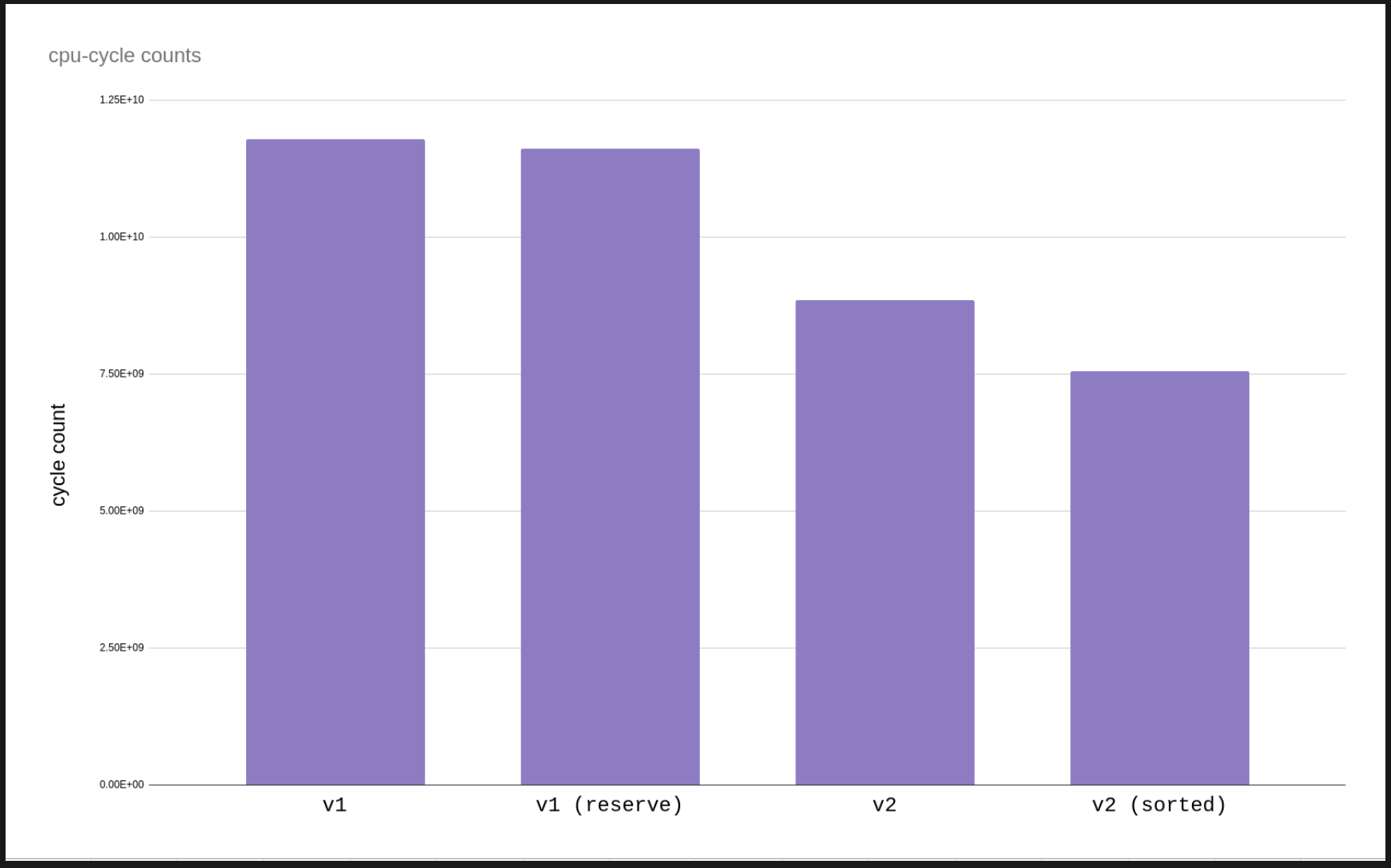
Measuring cache performance

```
==135709== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==135709== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
…
==135709== I   refs:      6,944,720,850
==135709== I1  misses:            6,115
==135709== LLi misses:            3,906
==135709== I1  miss rate:          0.00%
==135709== LLi miss rate:          0.00%
==135709==
==135709== D   refs:      2,526,314,915  (1,707,693,876 rd   + 818,621,039 wr)
==135709== D1  misses:       28,448,985  (   25,770,132 rd   +   2,678,853 wr)
==135709== LLd misses:        3,054,896  (    2,183,489 rd   +     871,407 wr)
==135709== D1  miss rate:           1.1% (          1.5%     +         0.3%  )
==135709== LLd miss rate:           0.1% (          0.1%     +         0.1%  )
==135709==
==135709== LL refs:          28,455,100  (   25,776,247 rd   +   2,678,853 wr)
==135709== LL misses:         3,058,802  (    2,187,395 rd   +     871,407 wr)
==135709== LL miss rate:            0.0% (          0.0%     +         0.1%  )
```

Measuring cache performance



Measuring cache performance



Some final remarks

Some final remarks

- Cache performance has a non-trivial effect on any algorithm

Some final remarks

- Cache performance has a non-trivial effect on any algorithm
- We can optimize the code for our search based planning (SBP) algorithms before even optimizing the algorithm itself and make an appreciable performance impact

Some final remarks

- Cache performance has a non-trivial effect on any algorithm
- We can optimize the code for our search based planning (SBP) algorithms before even optimizing the algorithm itself and make an appreciable performance impact
- We can use the STL to effectively implement a SBP algorithm

Some final remarks

- Cache performance has a non-trivial effect on any algorithm
- We can optimize the code for our search based planning (SBP) algorithms before even optimizing the algorithm itself and make an appreciable performance impact
- We can use the STL to effectively implement a SBP algorithm
- Memory ordering has a large impact (but we do not need to jump straight to writing allocators)

Some final remarks

- Cache performance has a non-trivial effect on any algorithm
- We can optimize the code for our search based planning (SBP) algorithms before even optimizing the algorithm itself and make an appreciable performance impact
- We can use the STL to effectively implement a SBP algorithm
- Memory ordering has a large impact (but we do not need to jump straight to writing allocators)
- Cache pipelines are complex, so measurement is our best bet!

Thanks!

https://github.com/briancairl/cppcon2023