Query Processing on Dynamic Networks with Customizable Contraction Hierarchies on Neo4j

Marius Hahn

Grouf of Database and Information Systems
Department of Computer and Information Science
Faculty of Sciences
Universität Konstanz

Master Colloquium, 14th December 2023

Table of Contents

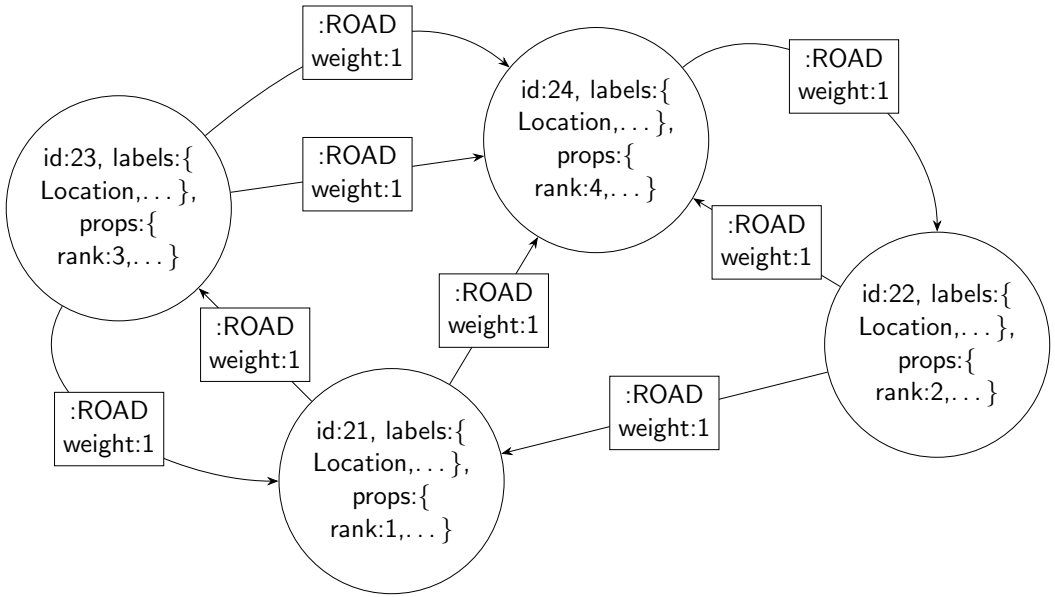
Table of Contents

Motivation and Context

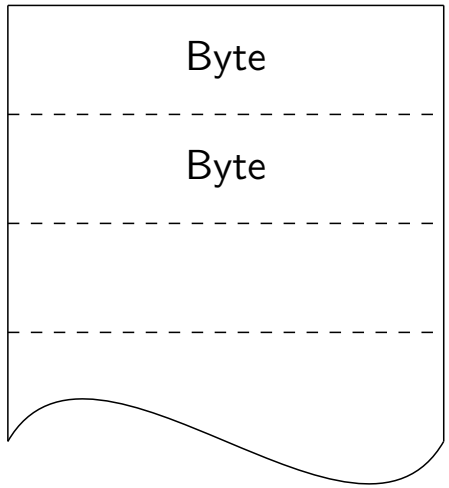
- Context $\Rrightarrow$ Graph Databases
- External memory
- Accelerate Shortest Path Queries in Databases
- Why Customizable Contraction Hierarchies?
  - fast for main memory applications
  - reasonable preprocessing time
  - It is updatable
- Test Data $\Rrightarrow$ Road Networks

Obstacles

Property Graph:



File:

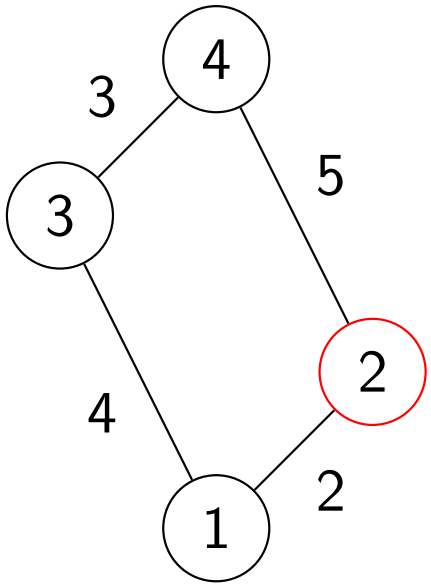


- transformation to data structure with a single dimension
- Databases use HDDs $\Rightarrow$ slow random access

Table of Contents

Dijkstra

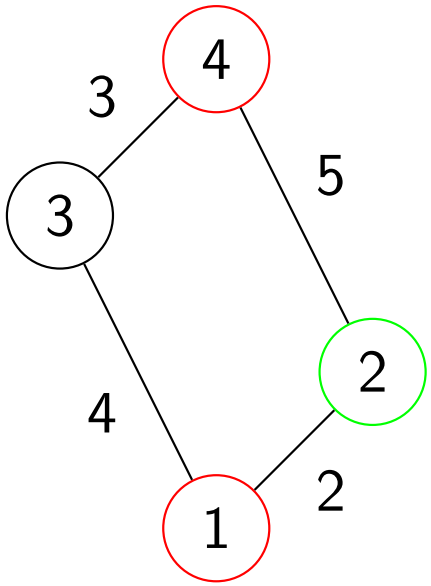
Let's go from v_2 to v_3



id	dist	settled
2	0	false

Dijkstra

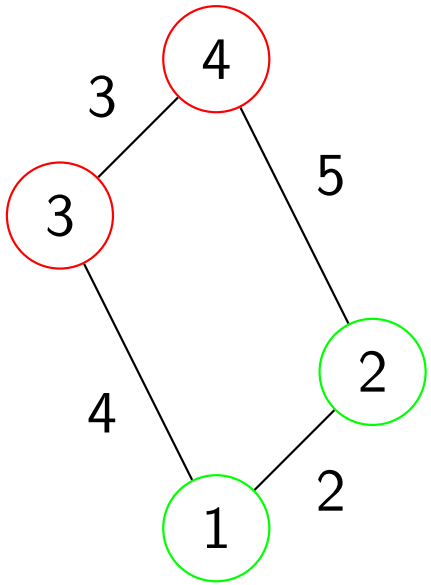
Let's go from v_2 to v_3



id	dist	settled
2	0	true
1	2	false
4	5	false

Dijkstra

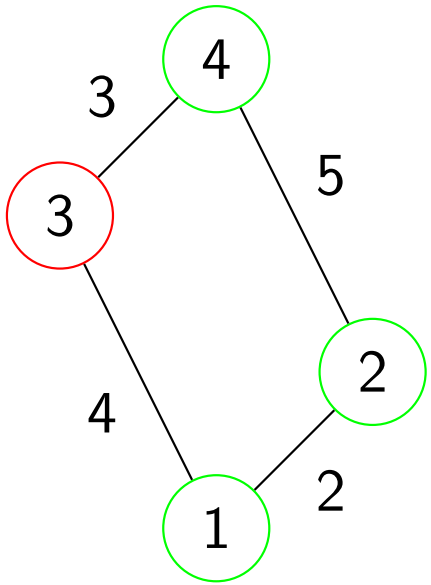
Let's go from v_2 to v_3



id	dist	settled
2	0	true
1	2	true
4	5	false
3	6	false

Dijkstra

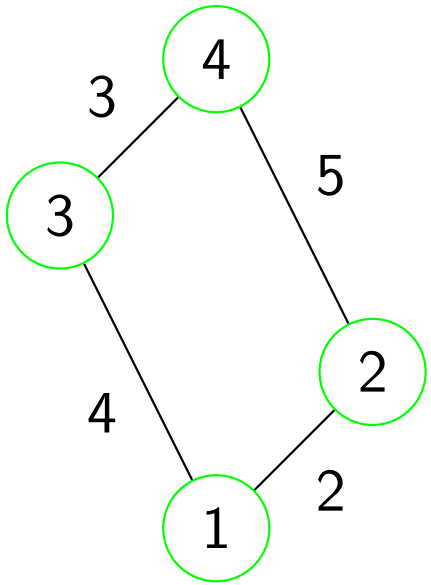
Let's go from v_2 to v_3



id	dist	settled
2	0	true
1	2	true
4	5	true
3	6	false

Dijkstra

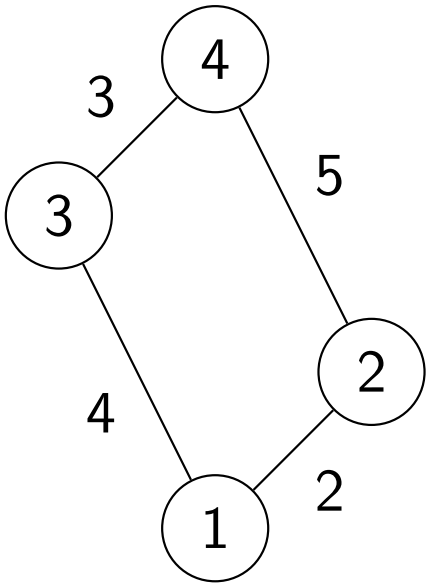
Let's go from v_2 to v_3



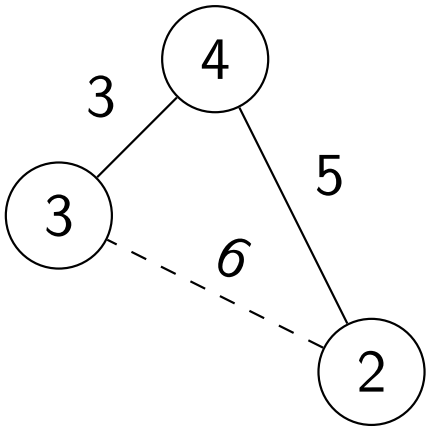
id	dist	settled
2	0	true
1	2	true
4	5	true
3	6	true

Table of Contents

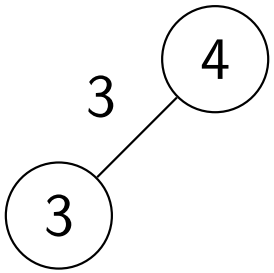
Contraction Hierarchies Example



Contraction Hierarchies Example



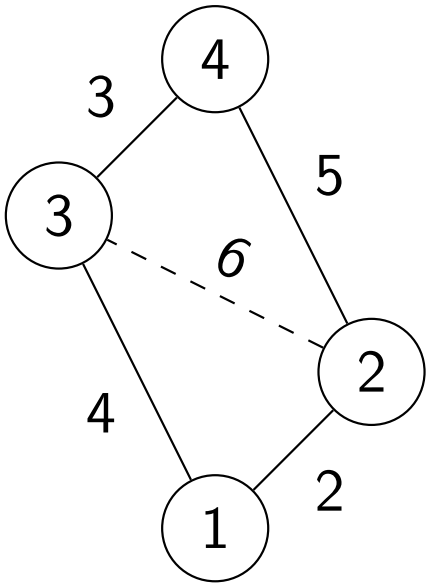
Contraction Hierarchies Example



Contraction Hierarchies Example

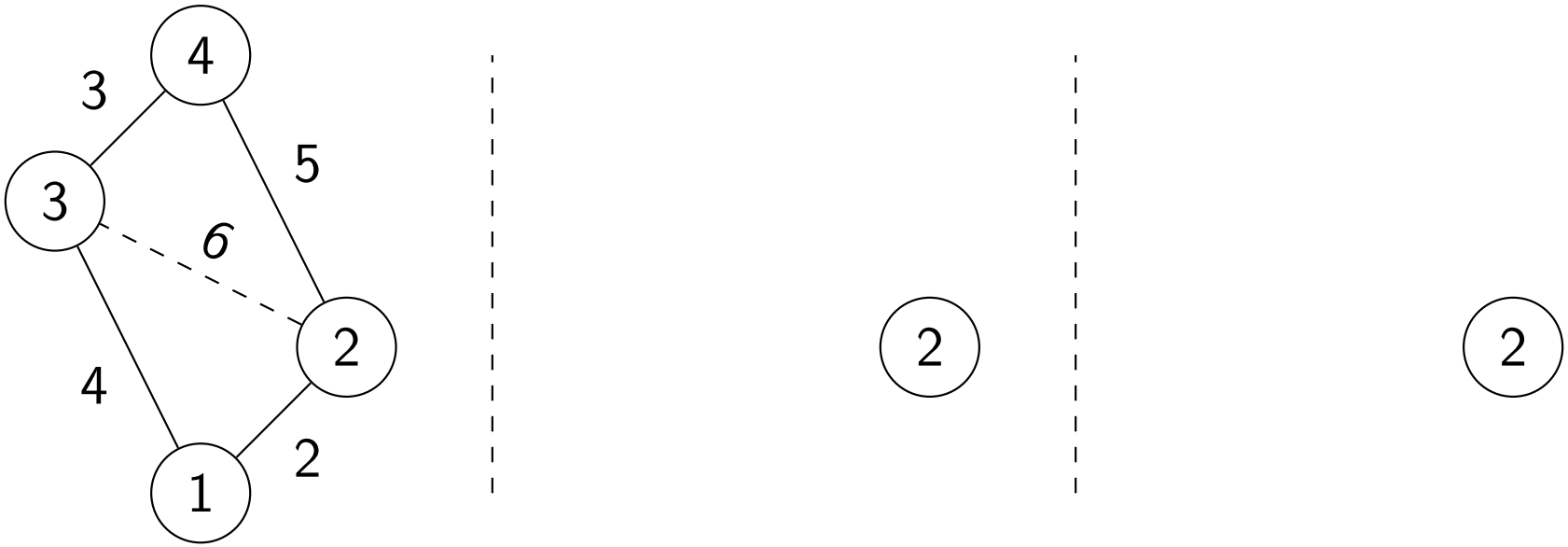
Contraction Hierarchies Example

Let's go from v_2 to v_3



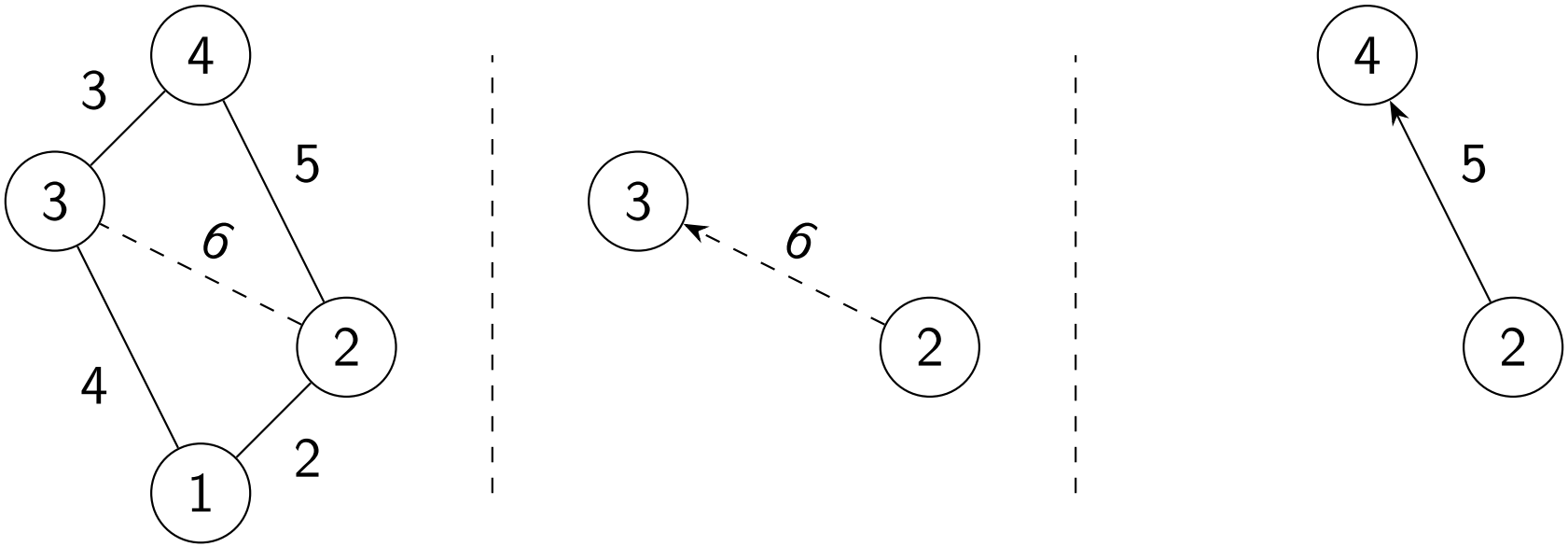
Contraction Hierarchies Example

Let's go from v_2 to v_3



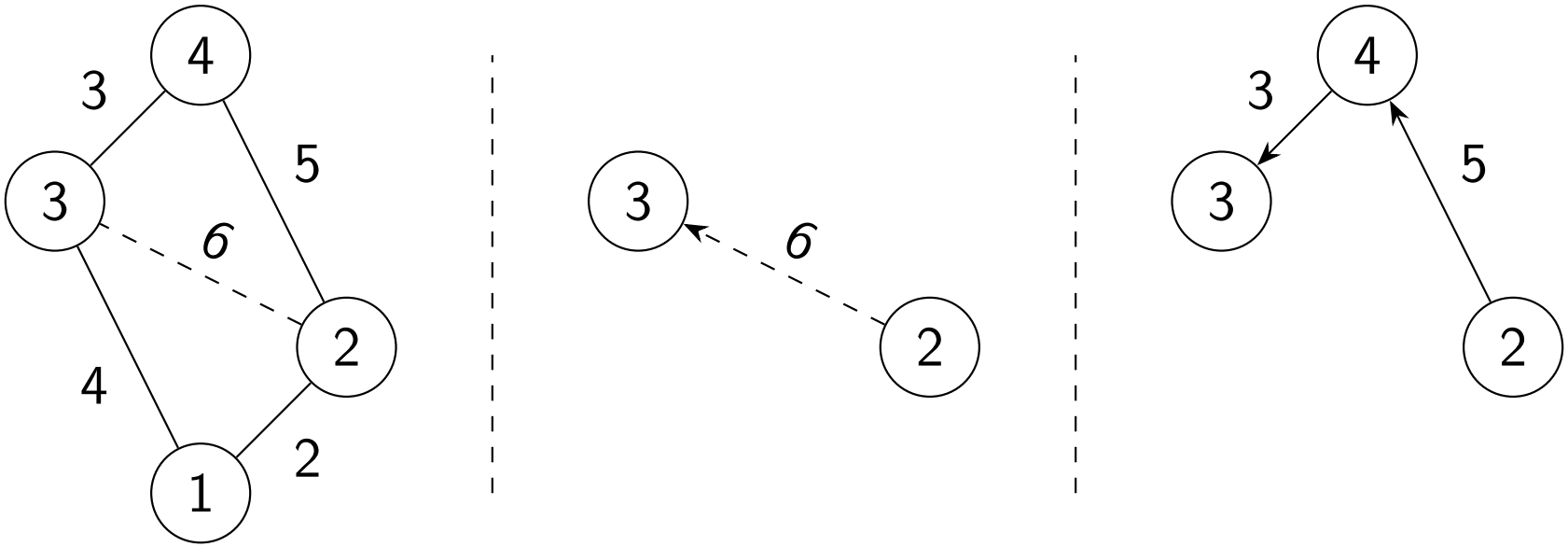
Contraction Hierarchies Example

Let's go from v_2 to v_3



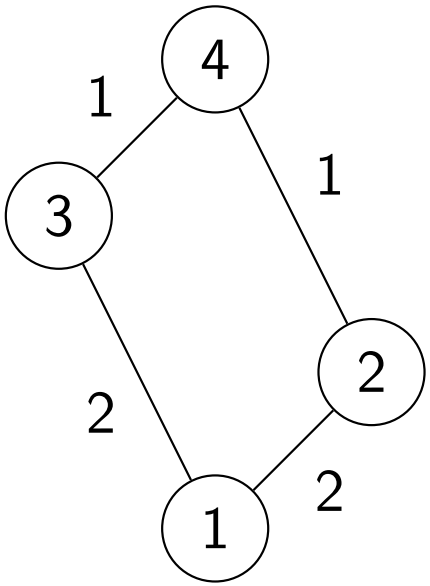
Contraction Hierarchies Example

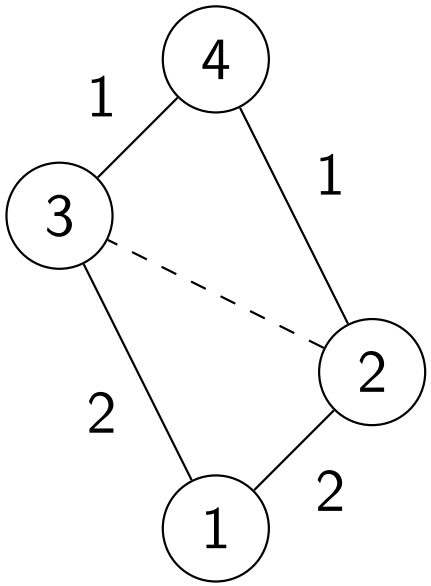
Let's go from v_2 to v_3



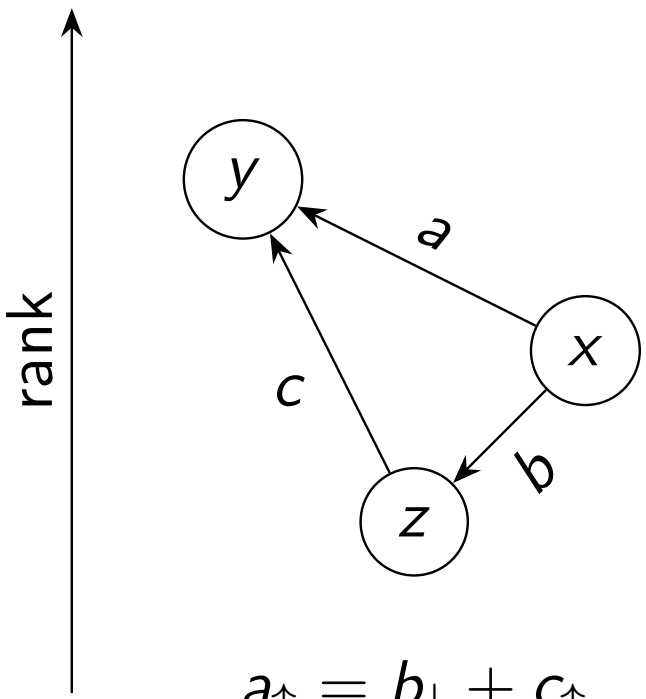
Customizable Contraction Hierarchies

- CH insert shortcut if shortest path property is violated
- CCH insert shortcut if there is no direct connection

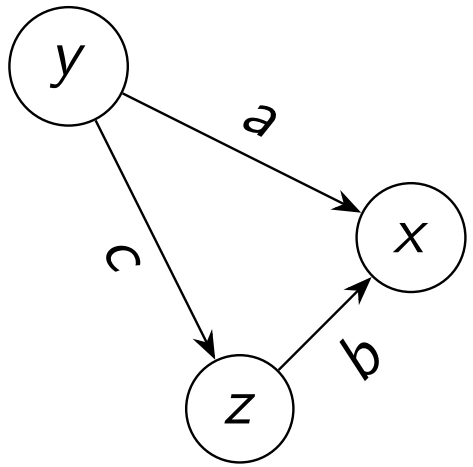




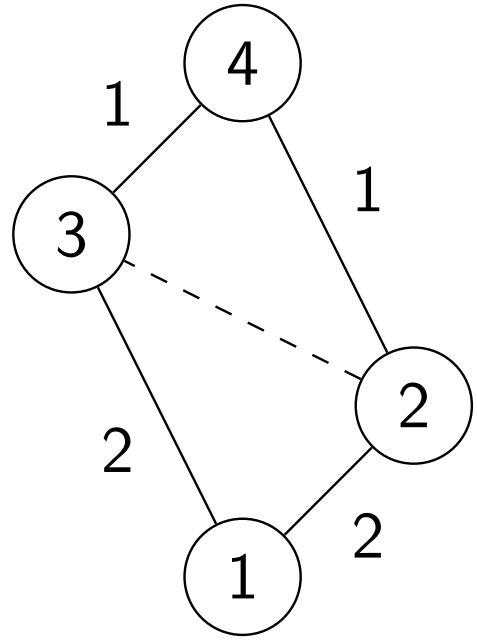
CCH Weights — Lower Triangles



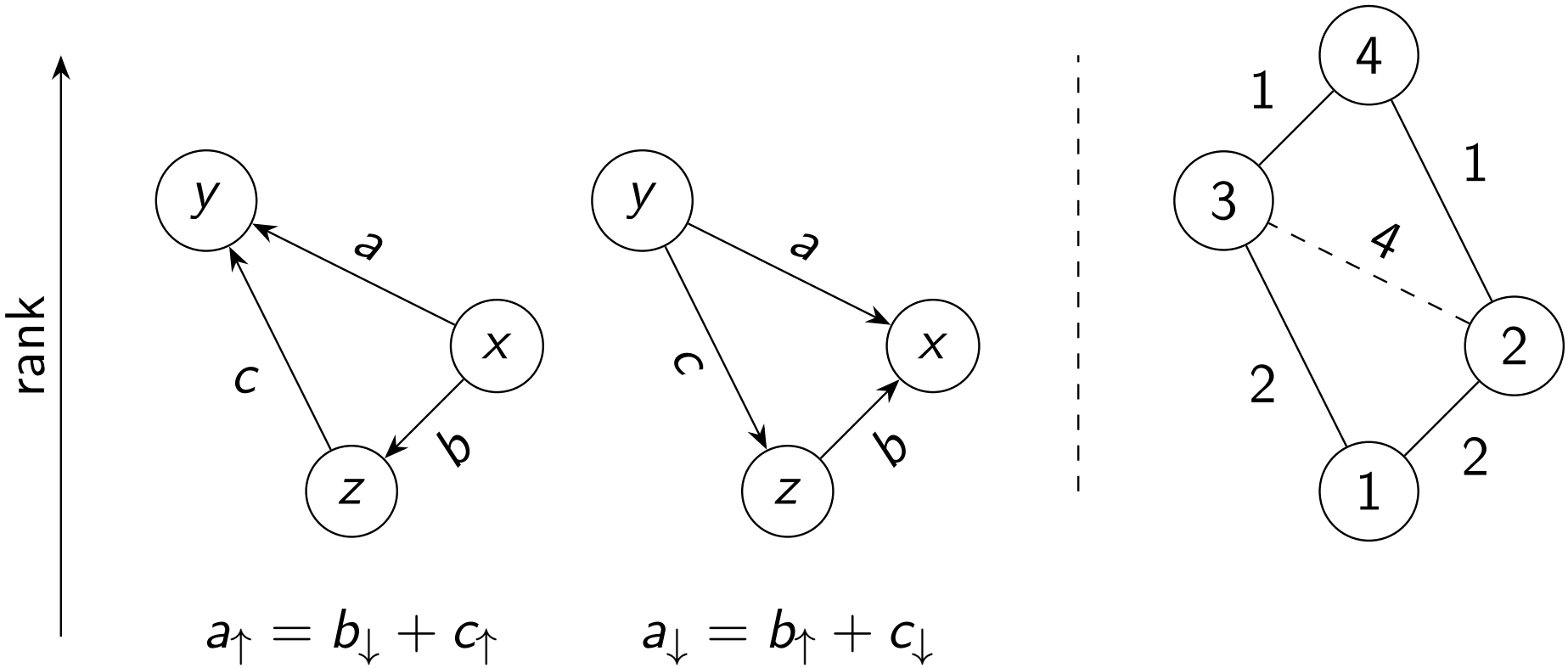
$$a_{\uparrow} = b_{\downarrow} + c_{\uparrow}$$



$$a_{\downarrow} = b_{\uparrow} + c_{\downarrow}$$

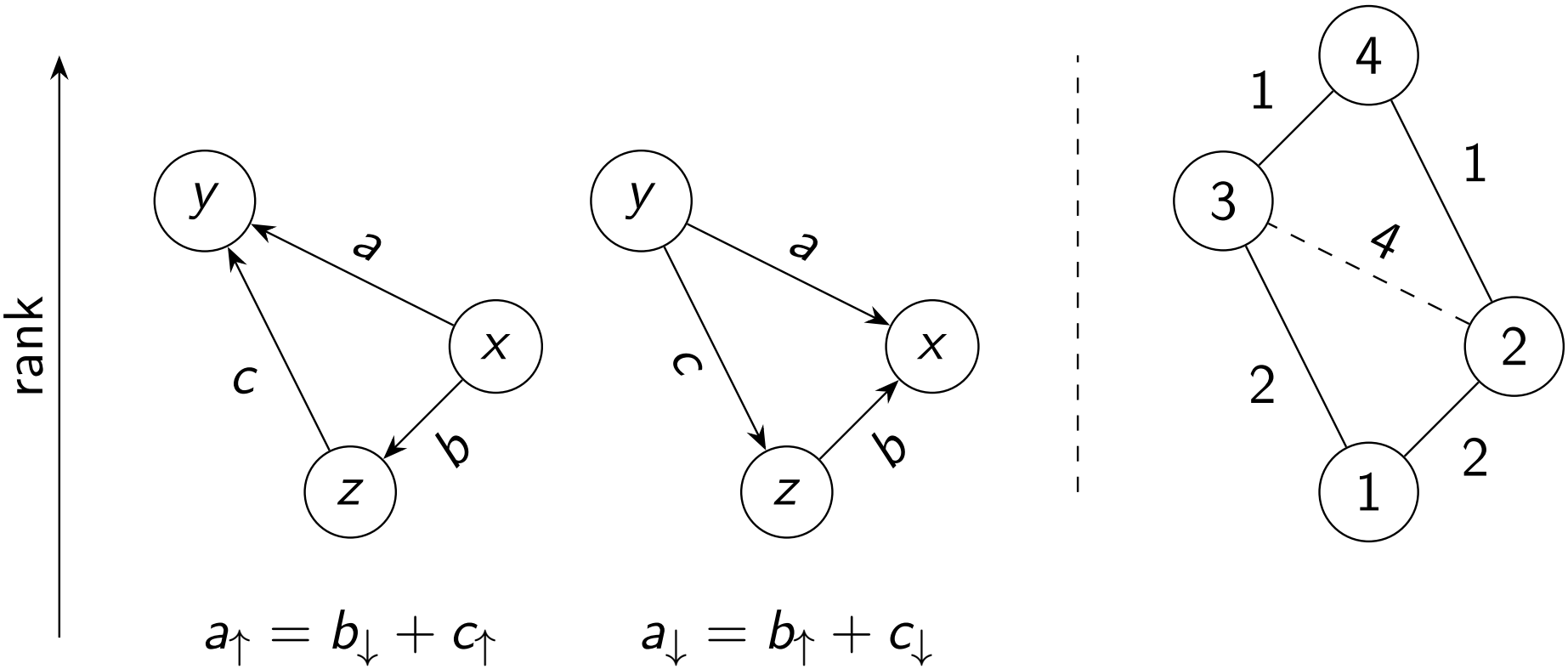


CCH Weights — Lower Triangles



$$a_{\uparrow} = b_{\downarrow} + c_{\uparrow} \qquad a_{\downarrow} = b_{\uparrow} + c_{\downarrow}$$

CCH Weights — Lower Triangles



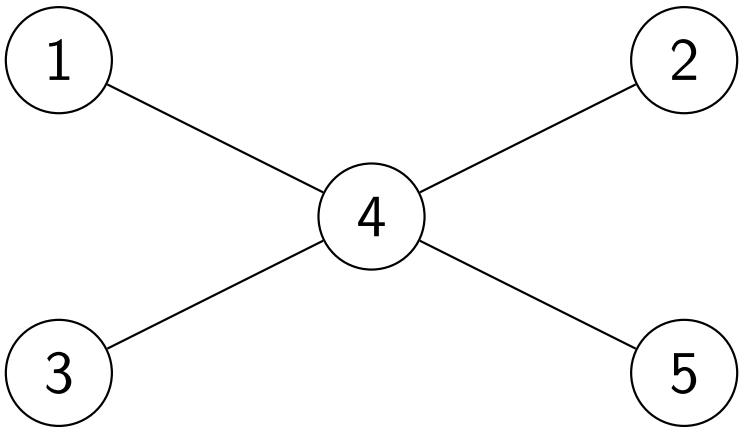
$$a_{\uparrow} = b_{\downarrow} + c_{\uparrow} \qquad a_{\downarrow} = b_{\uparrow} + c_{\downarrow}$$

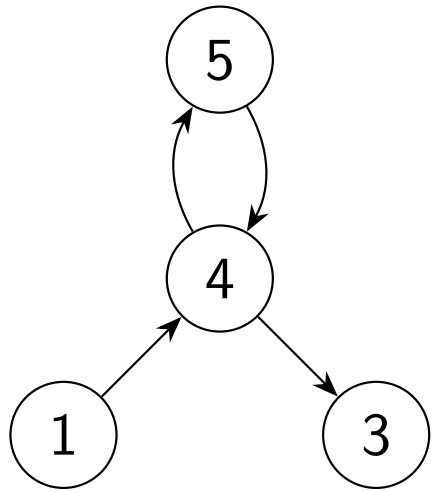
Bottom up for every arc! Also original arcs

Table of Contents

Important vertex not contraced Last

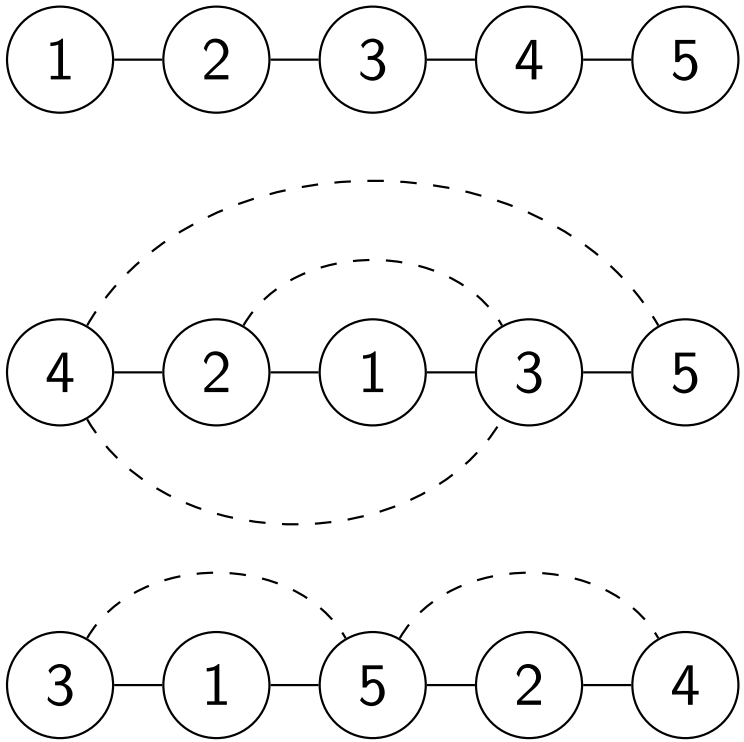
- Contracted Using Edge Difference
- Go from v(1) to v(3)
- Forward and Backward search are deeper that the should be
- Switch contraction order of v(4) and v(5)





Linear Contraction

1. linear contraction
   - No Shortcuts
   - Could happen with ED
   - four vertices to expand
2. middle vertex first
   - Three Shortcuts
   - four vertices to expand
3. good contraction order
   - Three Shortcuts
   - four vertices to expand



Importance Calculation

- add a level $l(v) = 0$ to each node
- if contracting v: $l(v) = max\{l(v) + 1, l(w)\} \forall w \epsilon N(v)$
- $A(v)$ set of added arcs
- $D(v)$ set of deleted arcs
- $h(a)$ hops an arc represents if unpacked

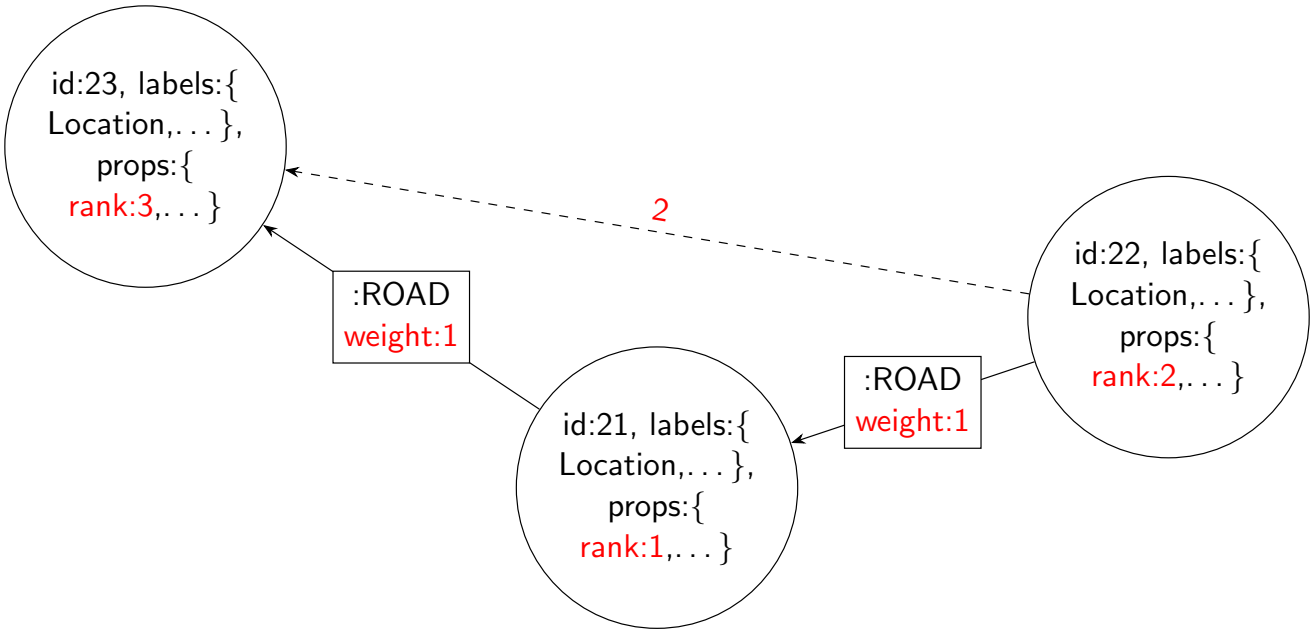
$$i(v) = l(v) + \frac{|A(v)|}{|D(v)|} + \frac{\sum_{a \epsilon A(v)} h(a)}{\sum_{a \epsilon D(v)} h(a)}$$

Important theorem

more shortcuts inserted but improves query time!

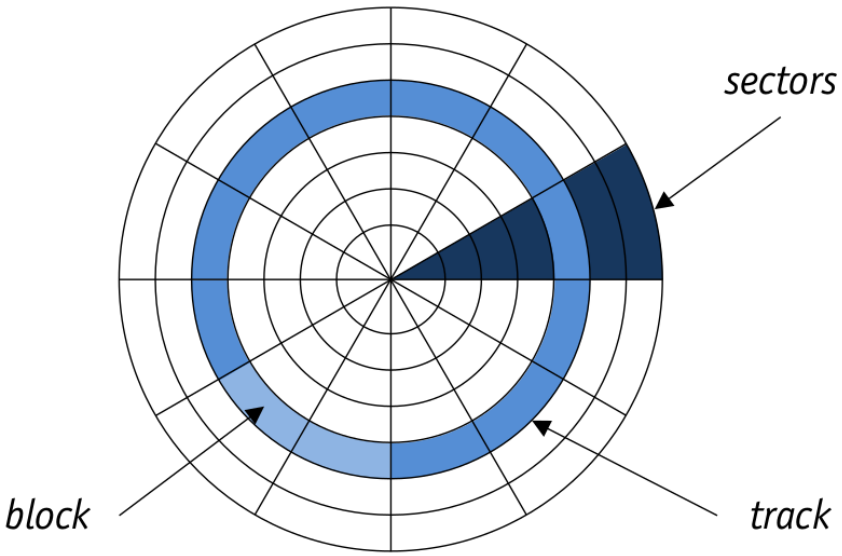
Table of Contents

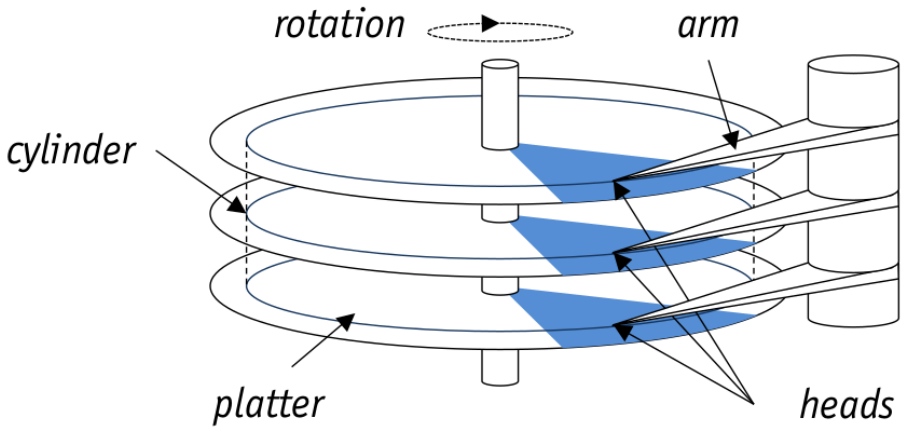
Persistance Objectives



- keep only necessary data
  - rank → to do the mapping to the input graph
  - arc weight
- Store edges that are likely to be request together spacial close
- Use as few space as possible → the less you write the less you read

Magnetic Disks





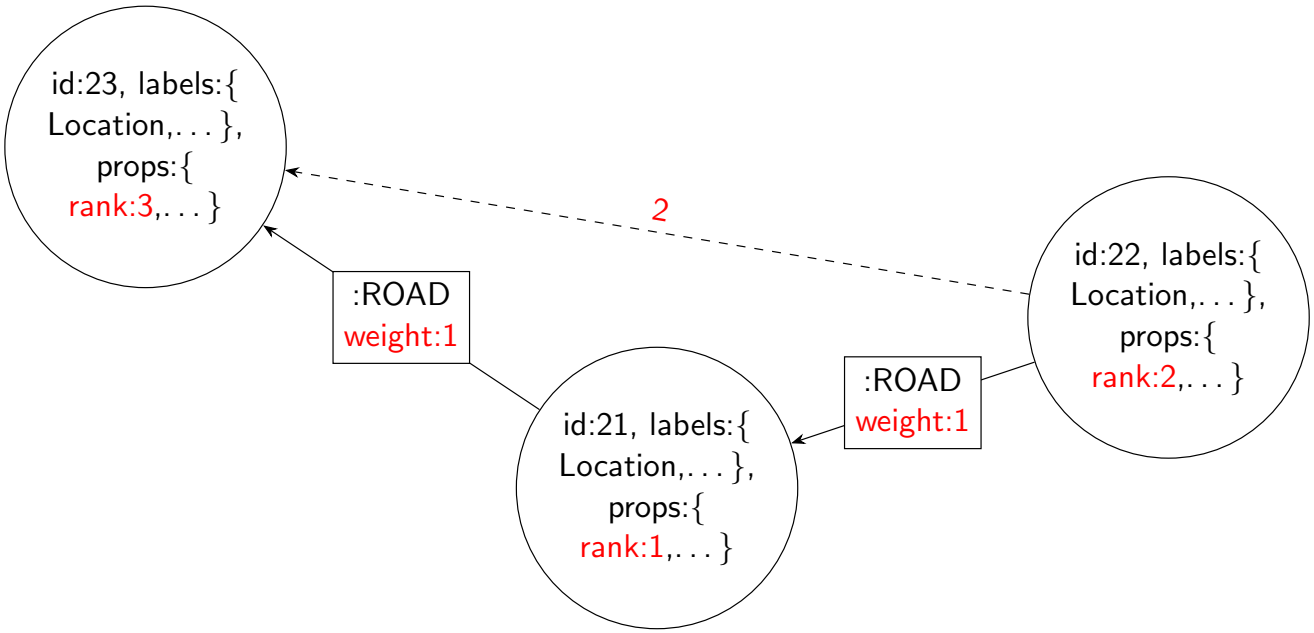
- Data is arranged in concentric rings (tracks) on platters
- Tracks are divided into arc-shaped sectors

One by One

Data is read from and written to disk one **block** at a time

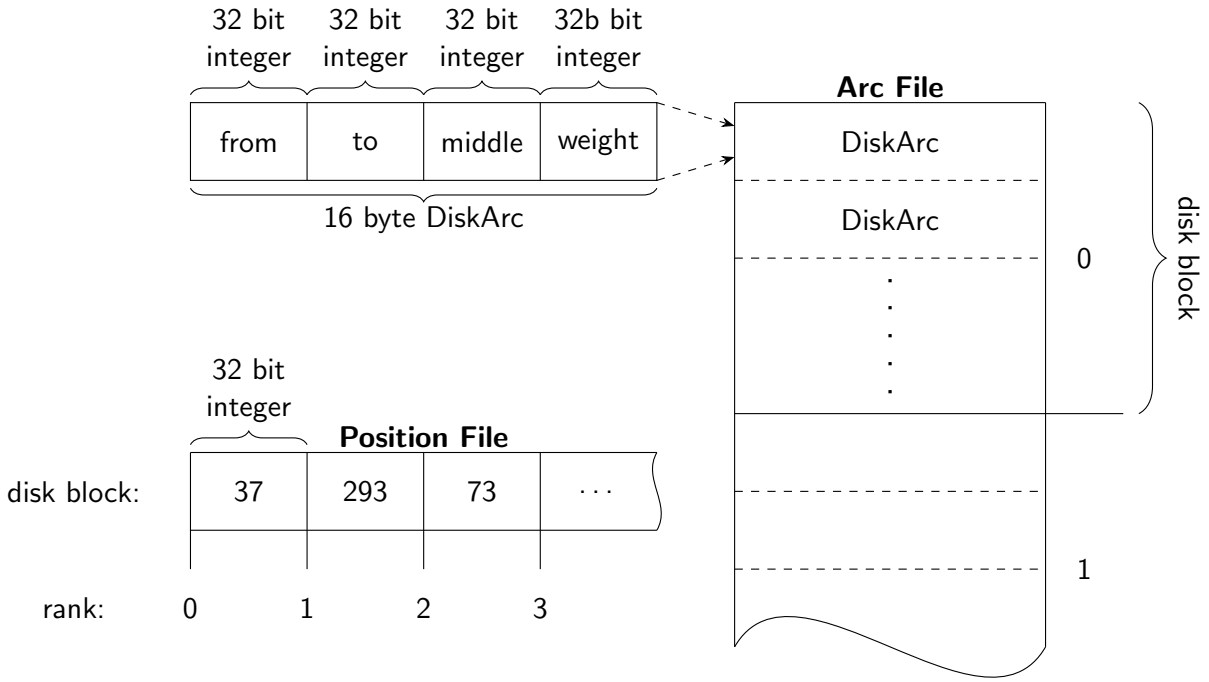
Transformation to a Table

- Depth-First-Search starting at highest rank
- retrieve only arcs. vertices will be reconstructed form arcs
- remember middle node



start rank	end rank	middle rank	weight
1	3	-1	1
2	3	1	2

Store Example



- fill all arcs of a vertex into a block
- add block number of rank to position file. $G_{\uparrow}$ use *from* ; $G_{\downarrow}$ use *to*
- if next vertex' arcs don't fit anymore $\rightarrow$ flush block and take next

Min Block Size

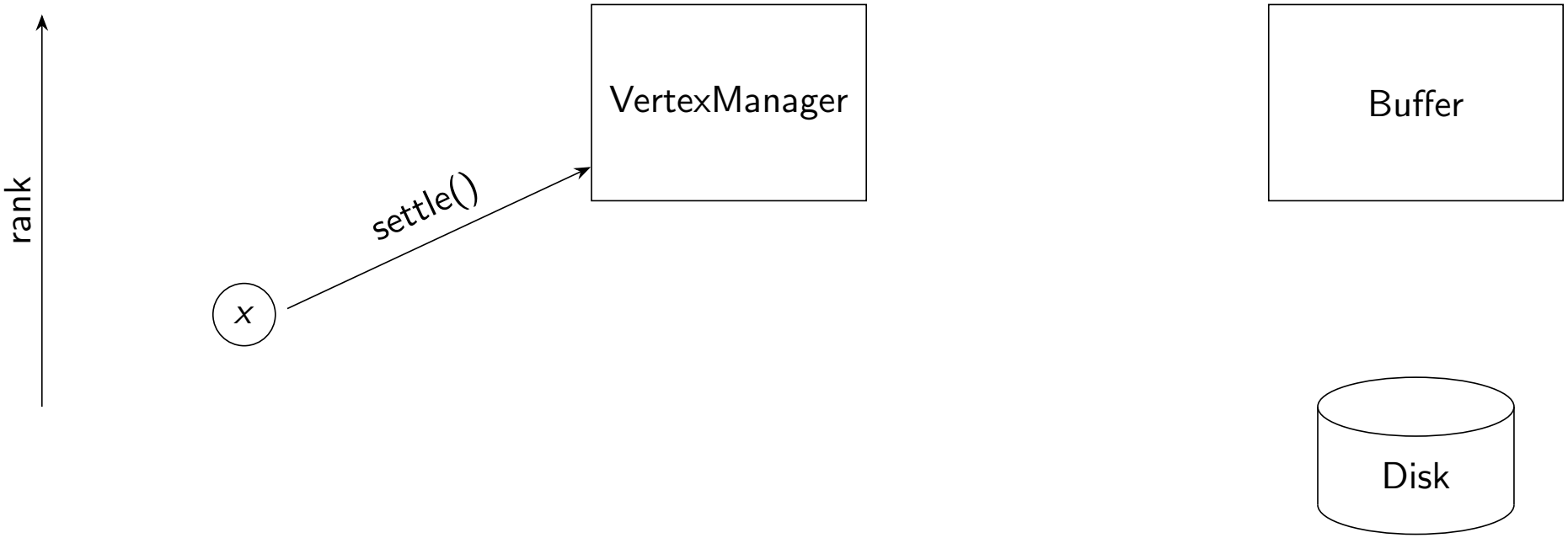
$$max(d_{\uparrow max}(v), d_{\downarrow max}(v)) \leqslant \frac{diskBlockSize}{16}$$

CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rightarrow$ only start node is loaded without arcs

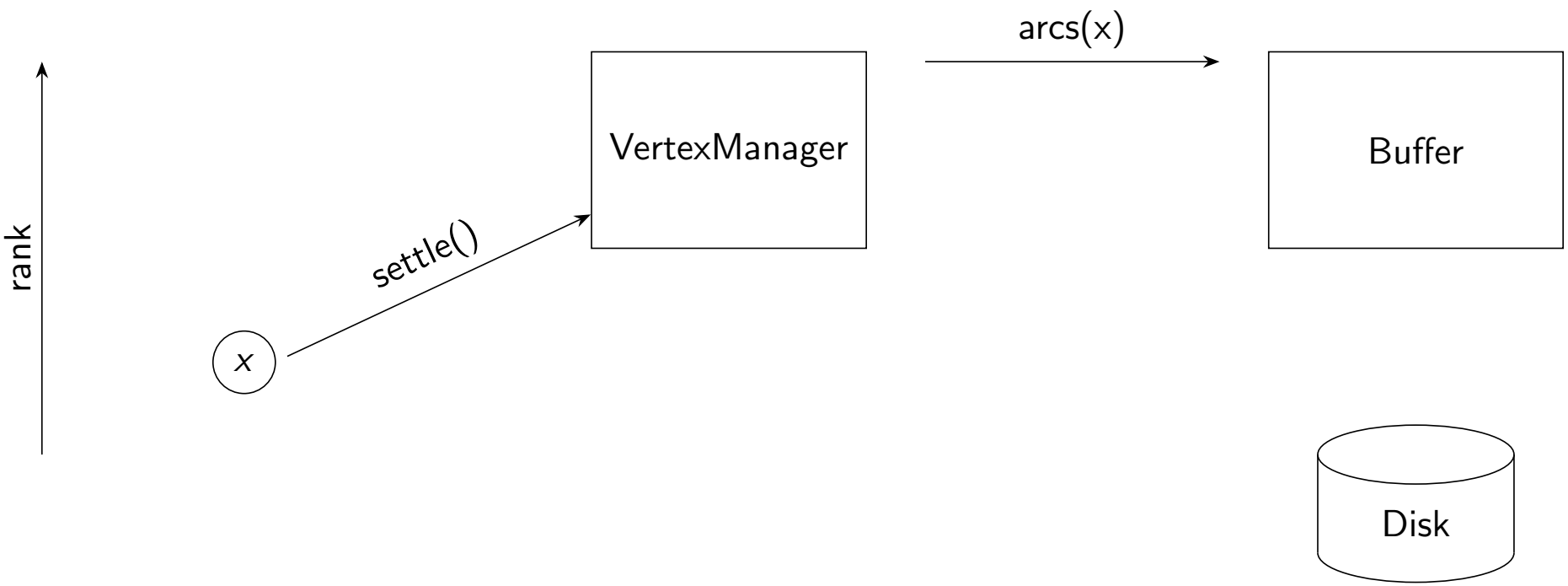
CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rrightarrow$ only start node is loaded without arcs
2. settle vertex (right before expanding it)



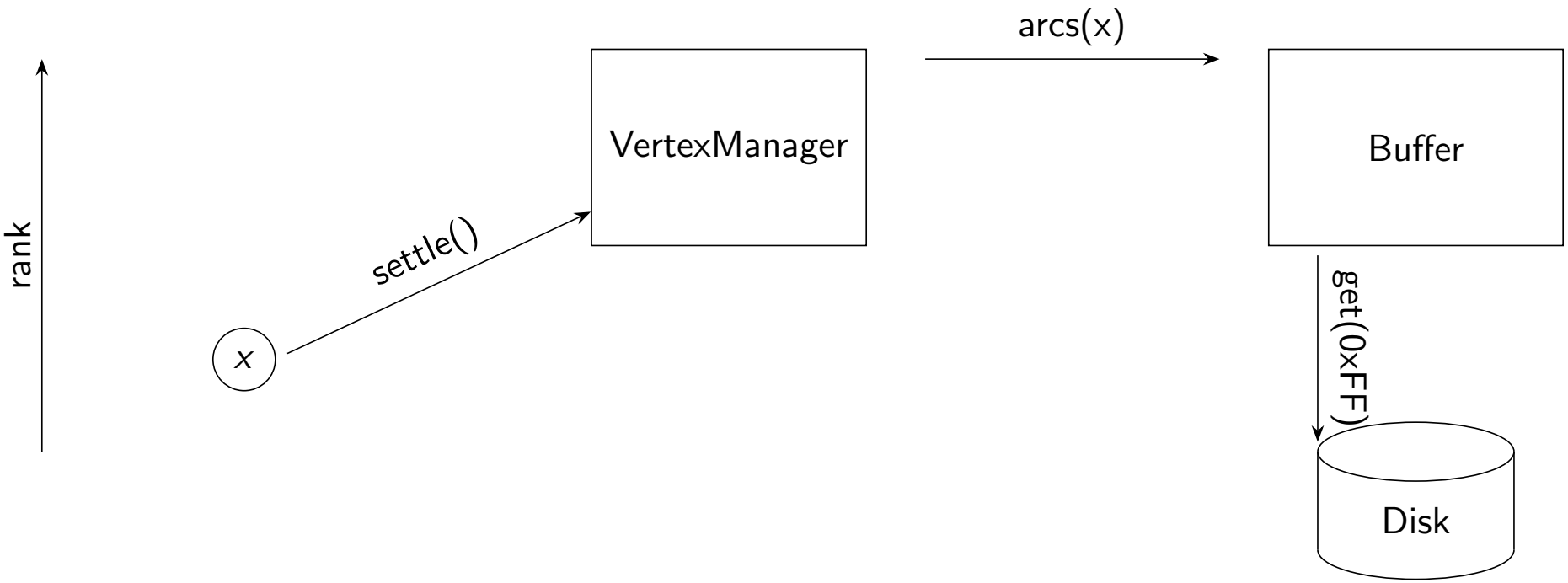
CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rightarrow$ only start node is loaded without arcs
2. settle vertex (right before expanding it)
   - VertexManager requests arc of v(x) from buffer



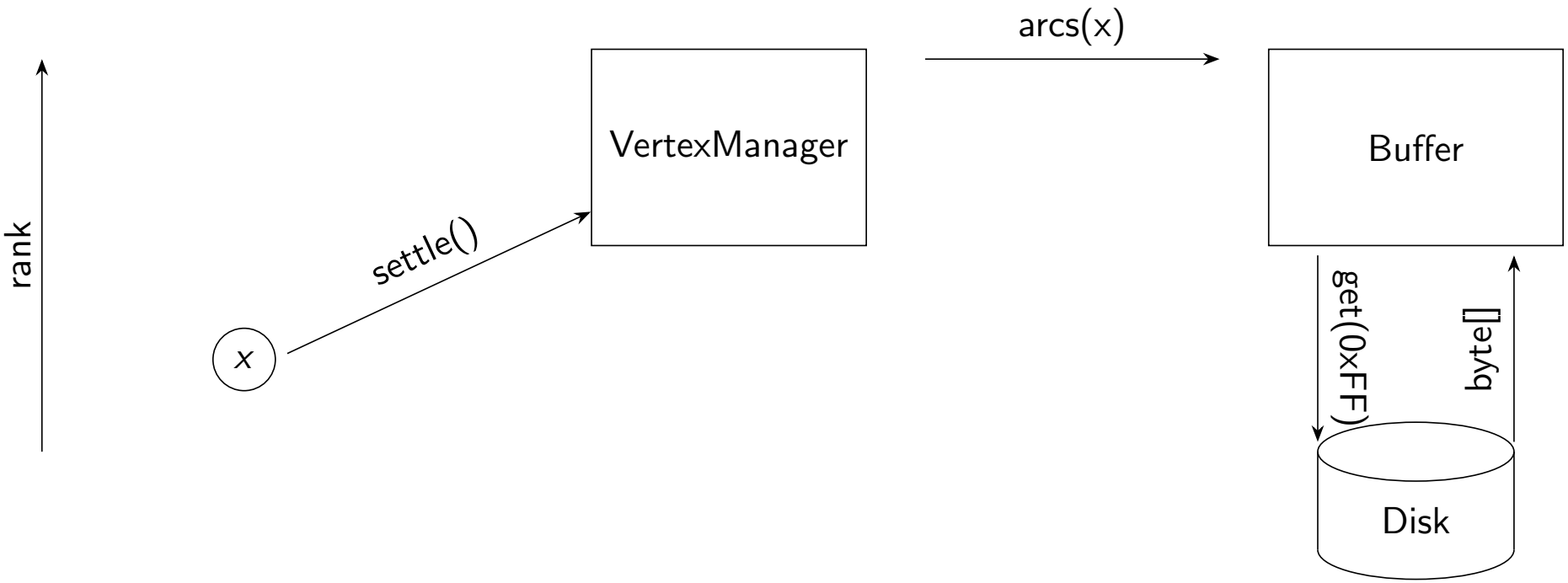
CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rightarrow$ only start node is loaded without arcs
2. settle vertex (right before expanding it)
   - VertexManager requests arc of v(x) from buffer
   - Buffer requests arcs from disk if not cached yet



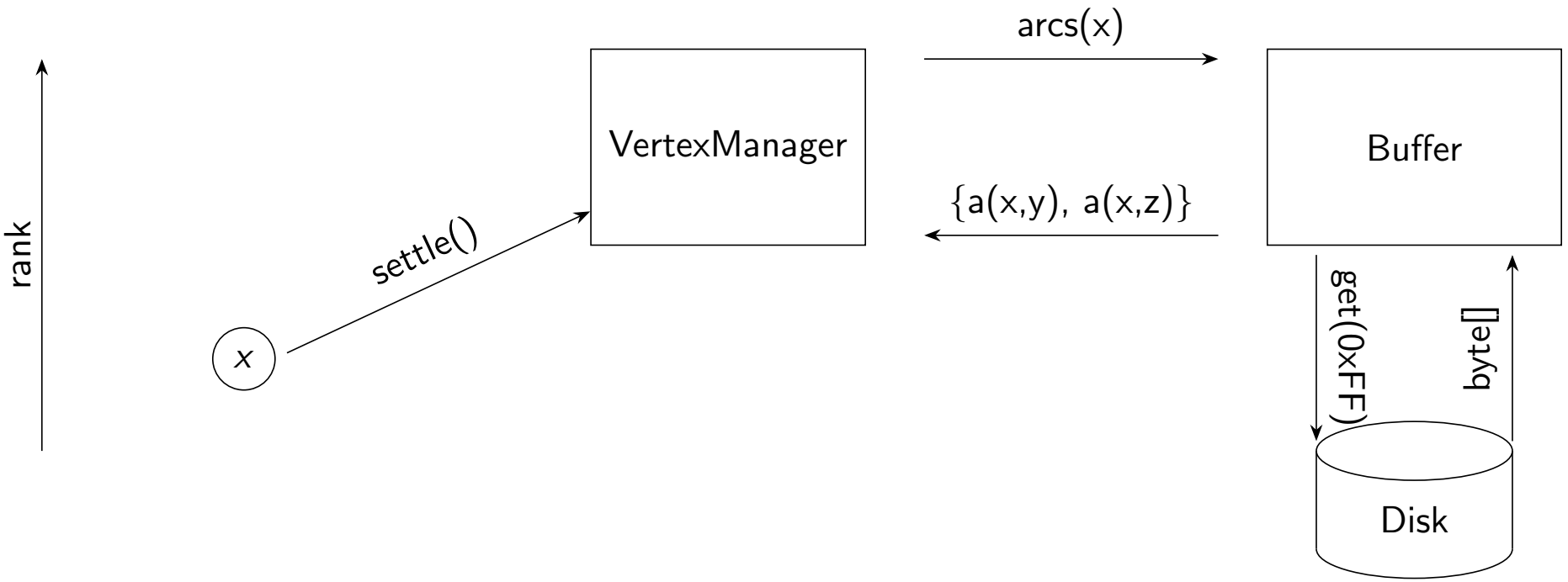
CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rightarrow$ only start node is loaded without arcs
2. settle vertex (right before expanding it)
   - VertexManager requests arc of v(x) from buffer
   - Buffer requests arcs from disk if not cached yet



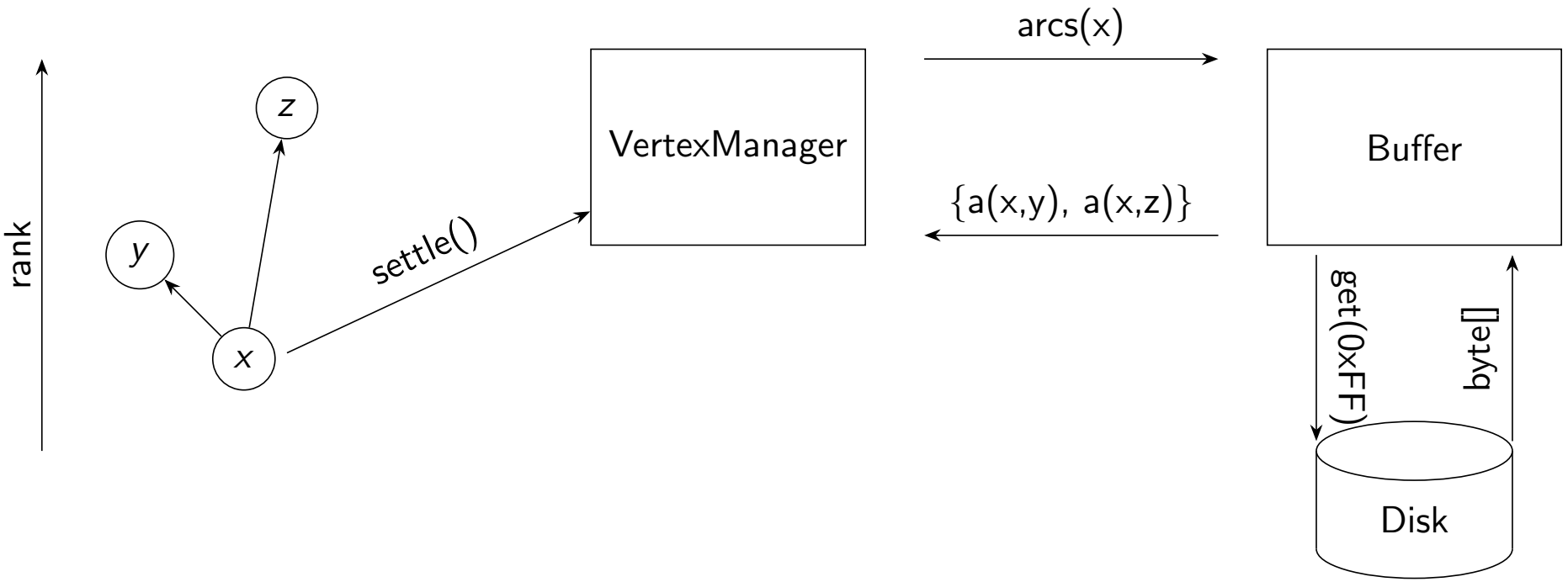
CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rightarrow$ only start node is loaded without arcs
2. settle vertex (right before expanding it)
   - VertexManager requests arc of v(x) from buffer
   - Buffer requests arcs from disk if not cached yet
   - Buffer returns arcs

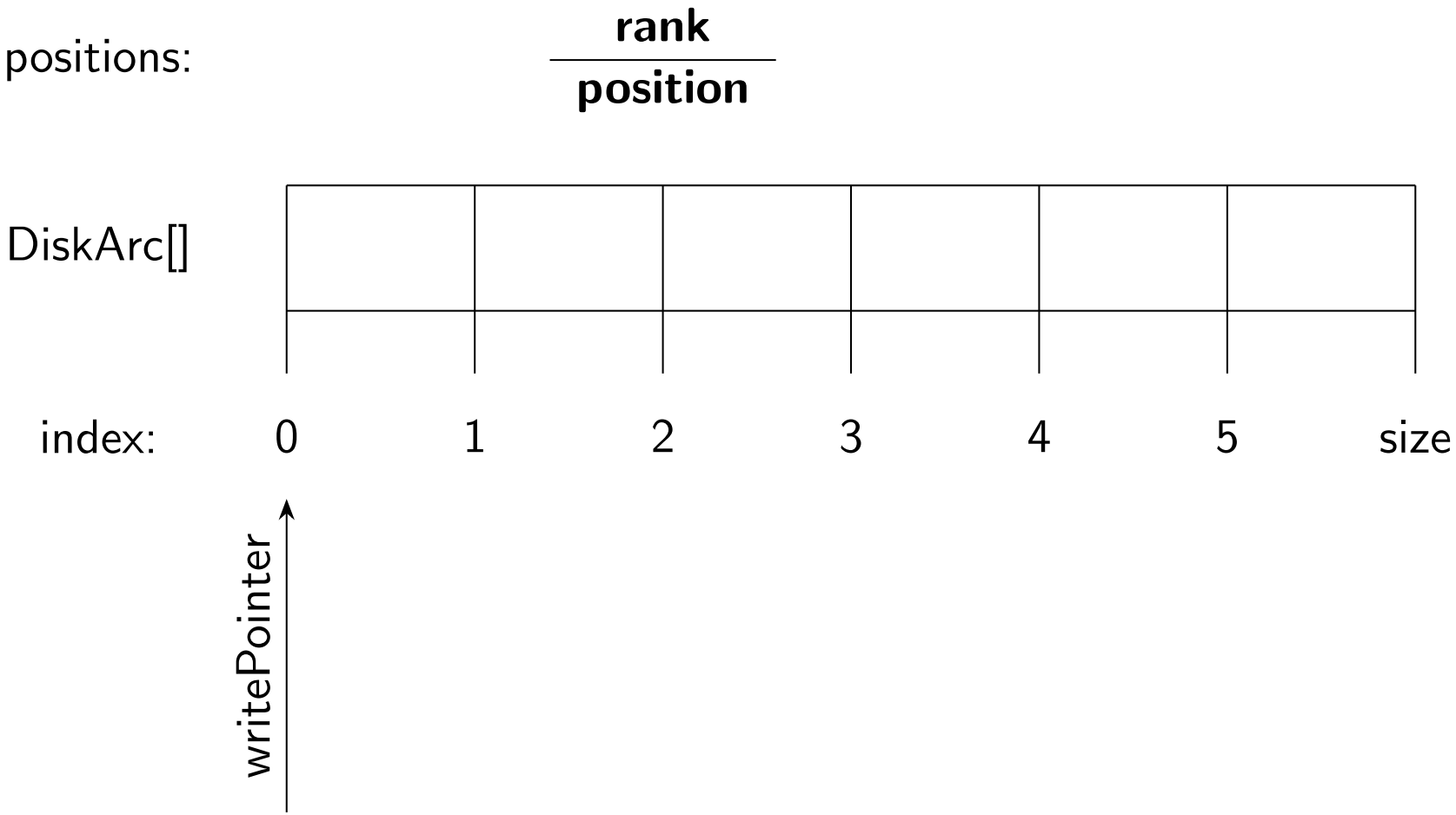


CCH Disk Search (upwards graph example)

1. lazy load vertices $\Rightarrow$ only start node is loaded without arcs
2. settle vertex (right before expanding it)
   - VertexManager requests arc of v(x) from buffer
   - Buffer requests arcs from disk if not cached yet
   - Buffer returns arcs
   - VertexManager attaches arcs to v(x)



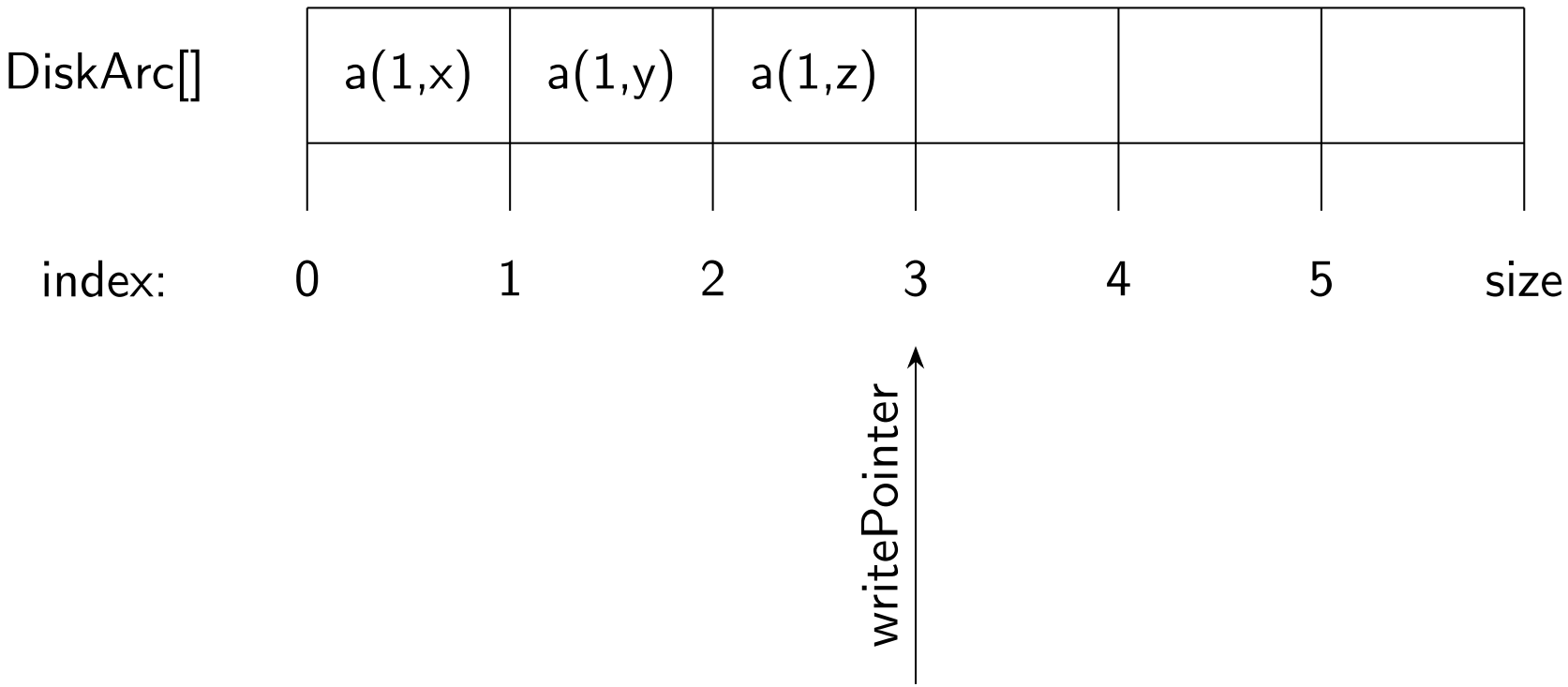
Circular Buffer



Circular Buffer

positions:

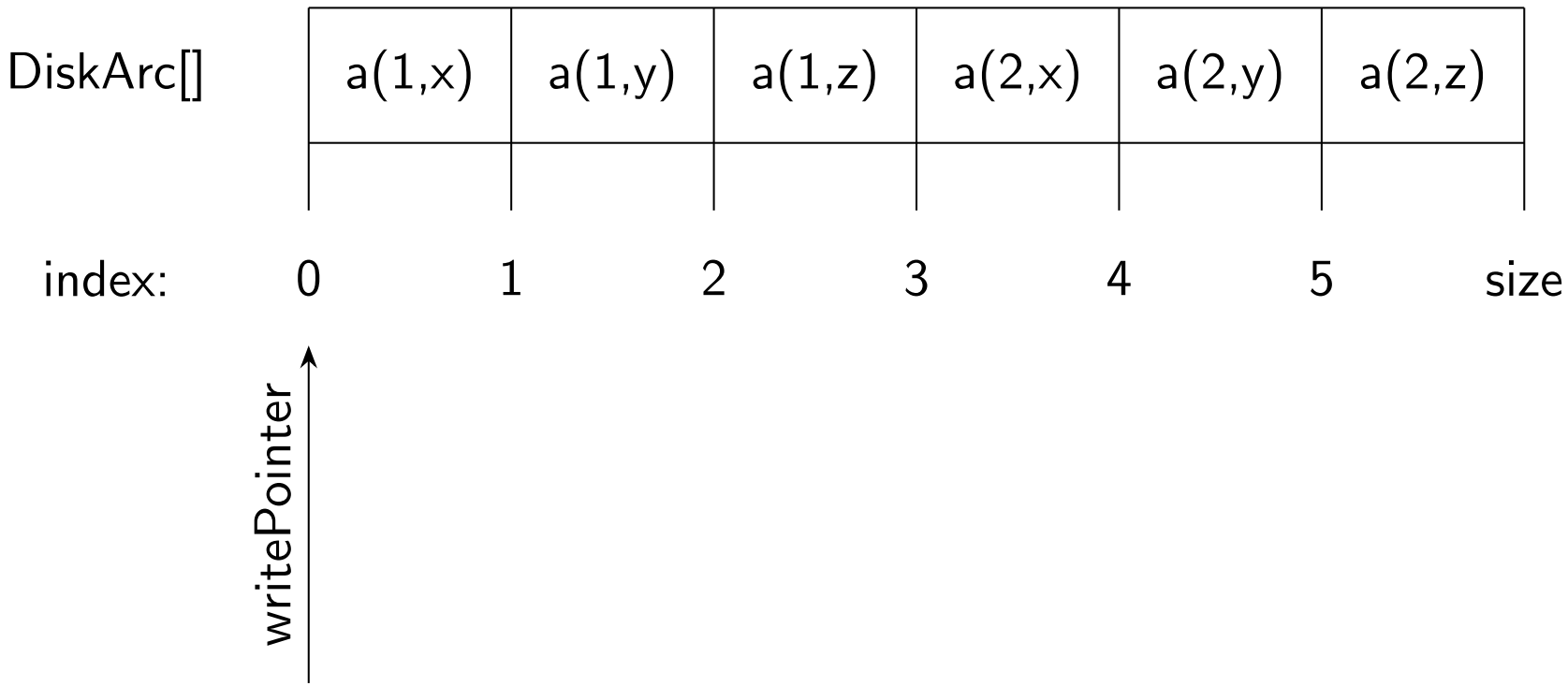
rank	1
position	2



Circular Buffer

positions:

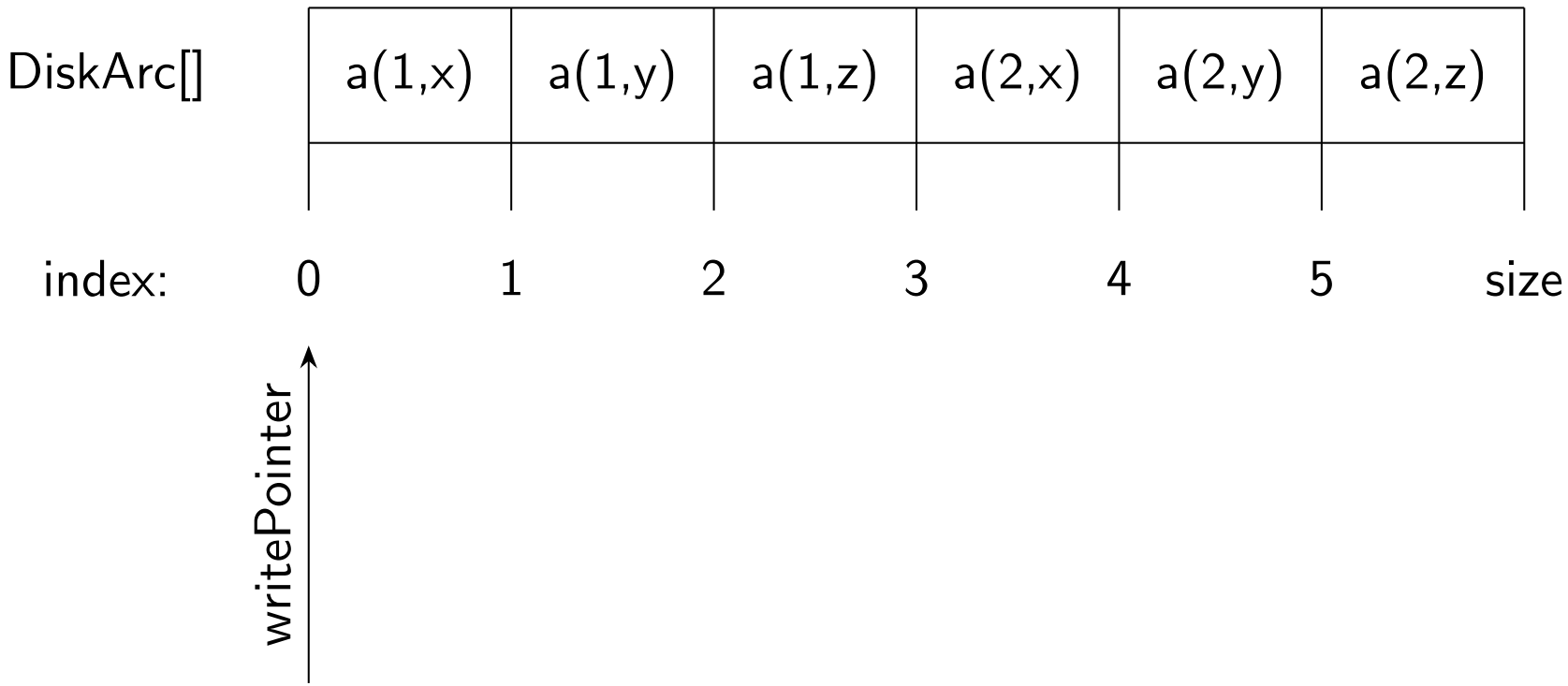
rank	1	2
position	2	5



Circular Buffer

positions:

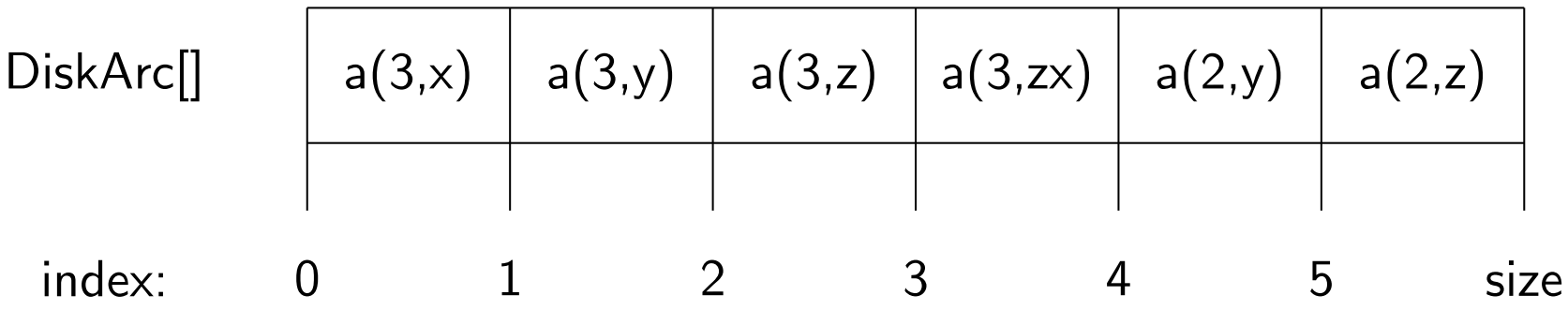
rank	1	2	max(rank)
position	2	5	-1



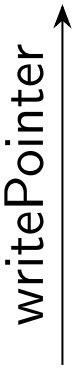
Circular Buffer

positions:

rank	1	2	3	max(rank)
position	2	5	3	-1



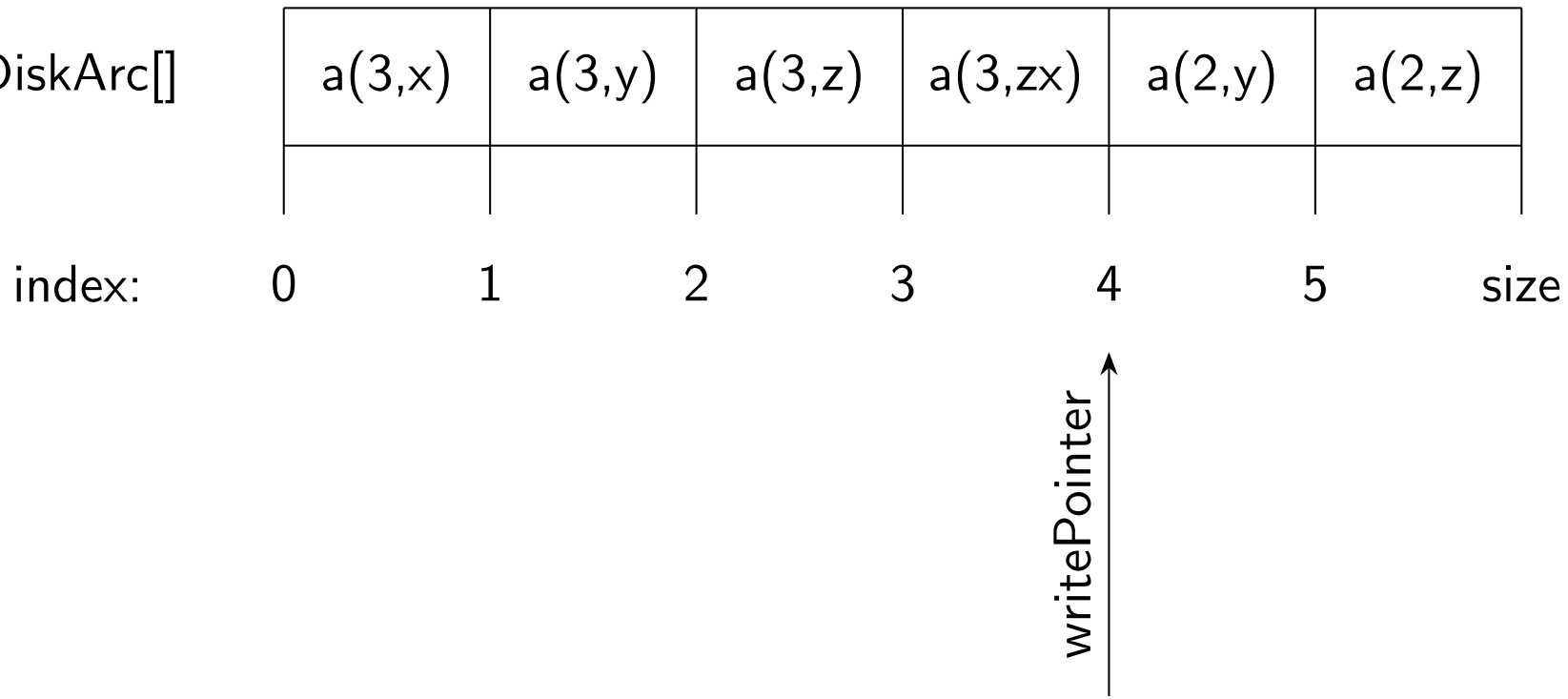
remove incomplete edge set from position



Circular Buffer

positions:

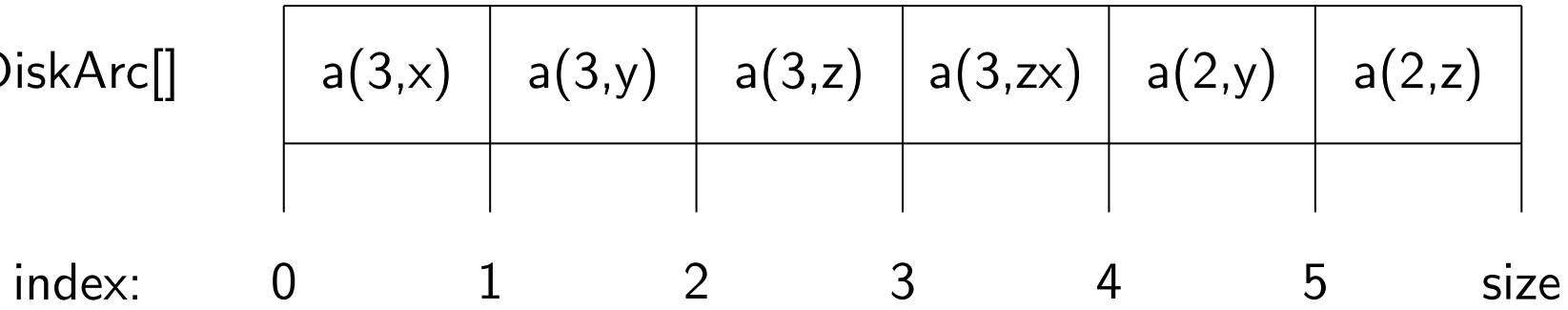
rank	1	3	max(rank)
position	2	3	-1



Circular Buffer

positions:

rank	1	3	max(rank)
position	2	3	-1



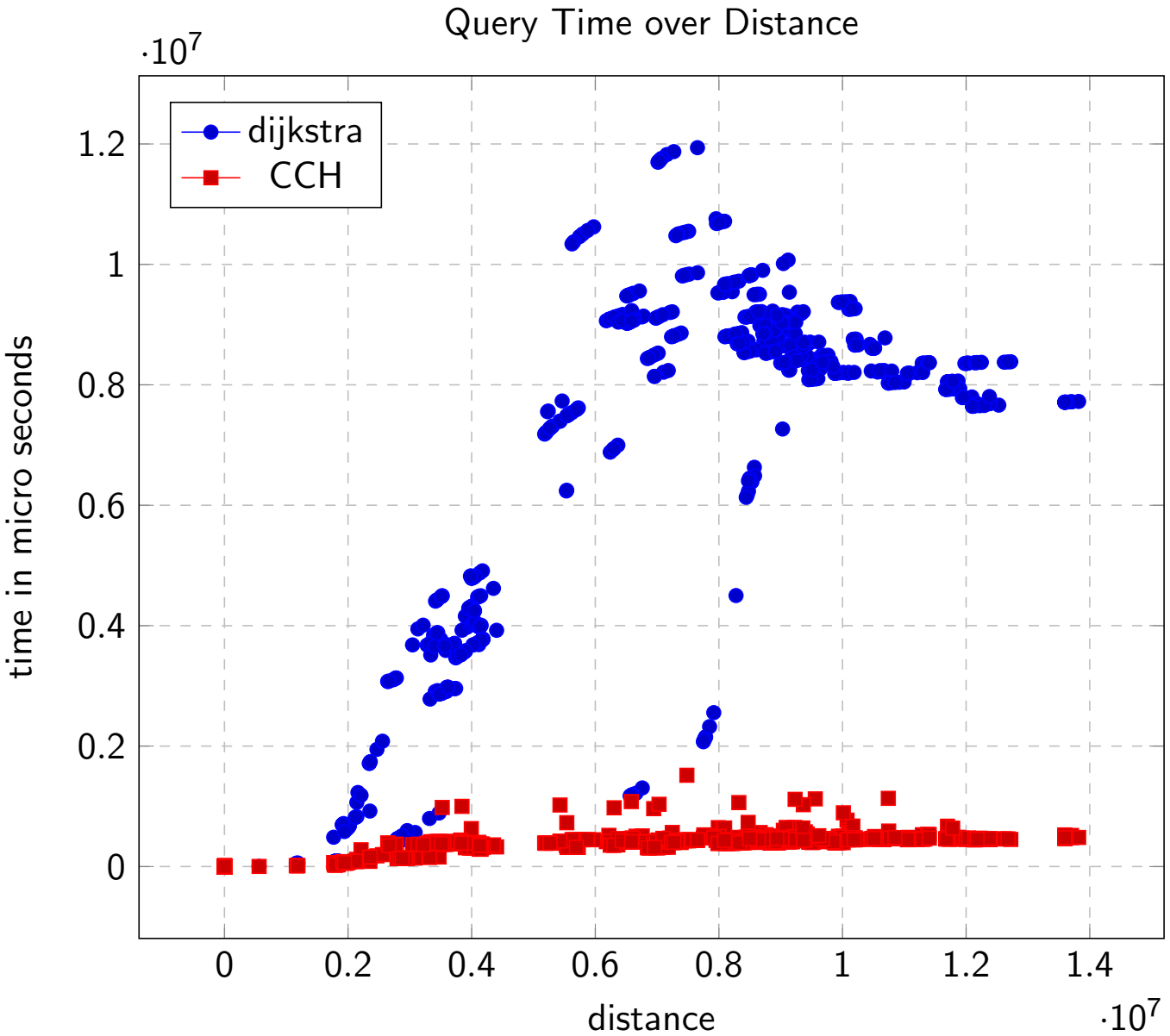
- Retrieve Arcs $\Rightarrow$ iterate backwards from position until start vertex differs
- If arc is doesn't start with requested rank $\Rightarrow$ remove position and refetch

Table of Contents

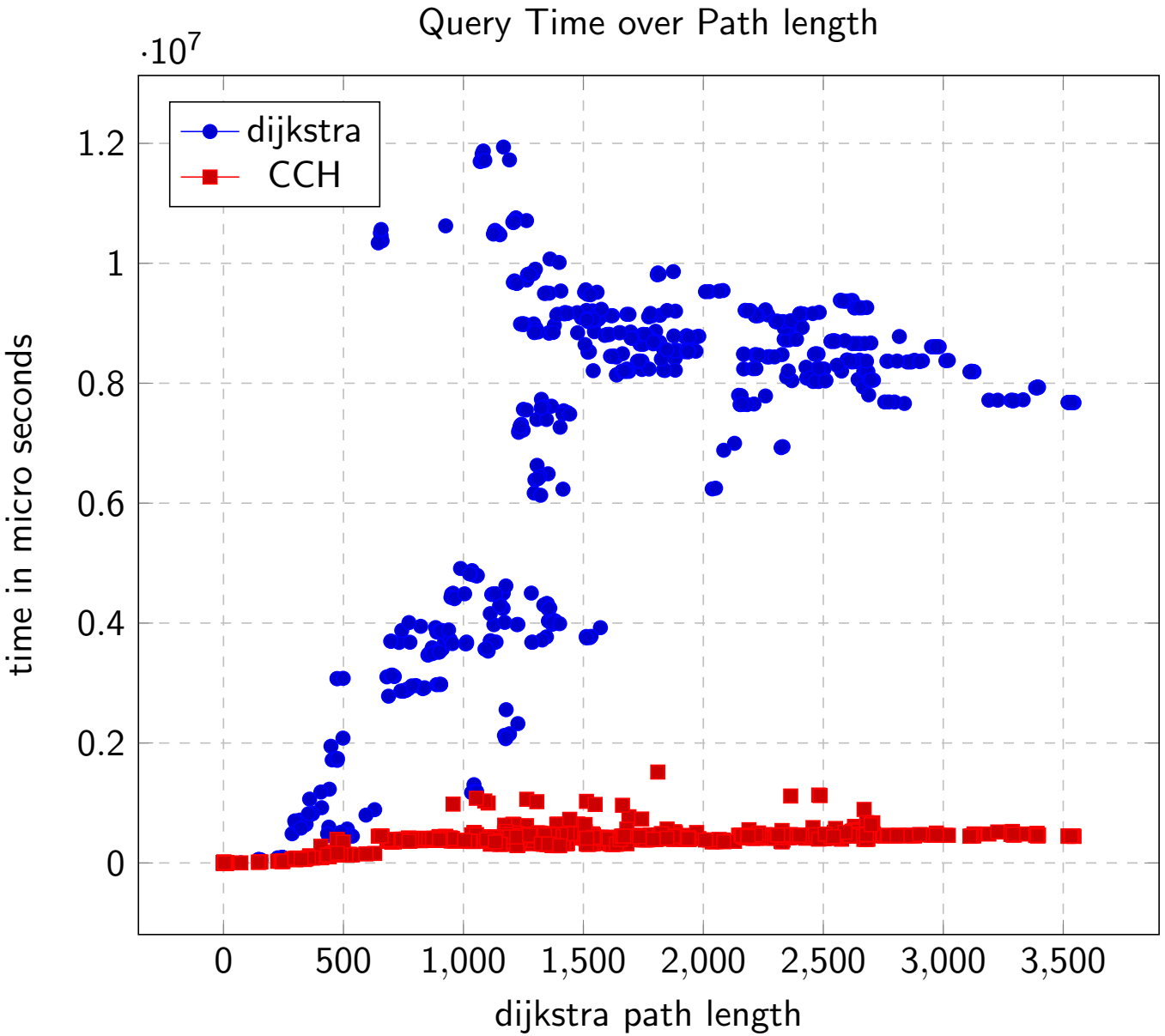
General Results

	New York	Colorado	Florida	California +Nevada	Great Lakes	Eastern USA
$\|V\|$	264,346	435,666	1,070,376	1,890,815	2,758,119	3,598,623
$\|A\|$	730,100	1,042,400	2,687,902	4,630,444	6,794,808	8,708,058
$\|S\|$	2,153,002	1,680,290	4,397,804	8,598,552	17,833,050	17,712,722
$\frac{\|S\|}{\|A\|}$	2.93	1.59	1.62	1.85	2.62	2.03
$t_{contraction}$	545 s	233 s	579 s	4,384 s	25.29 h	23.29 h
$max(d(v))$	1,150	629	785	1,252	2,433	2,391
$\|\bigcirc A_{\uparrow}\|$	23.1 MB	21.9 MB	56.9 MB	107 MB	201 MB	215 MB
pos-file$_{\uparrow}$	1.1 MB	1.8 MB	1.3MB	7.6 MB	11.1 MB	14.4 MB
$t_{dijkstra}$	0.816 s	0.549 s	2.630 s	4.858 s	5.425 s	5.387 s
t_{cch}^{640kB}	0.140 s	0.122 s	0.147 s	0.289 s	0.732 s	0.727 s
I/O^{640kB}	574	437	500	899	1671	1572
t_{update}	90 s	51 s	142 s	444 s	1827s	1557s
$t_{cch-updated}^{640kB}$	0.147 s	0.129 s	0.150 s	0.302 s	0.783 s	0.855 s
$I/O_{cch-upd.}^{640kB}$	569	457	516	924	2779	2716
$t_{cch-updated}^{20\%}$	0.136 s	0.130 s	0.092 s	0.183 s	0.660 s	0.680 s
$I/O_{cch-upd.}^{20\%}$	315	307	226	283	804	680
$t_{cch-updated}^{100\%}$	0.062 s	0.038 s	0.039 s	0.099 s	0.438 s	0.479 s
$I/O_{cch-upd.}^{100\%}$	0	0	0	0	1	0

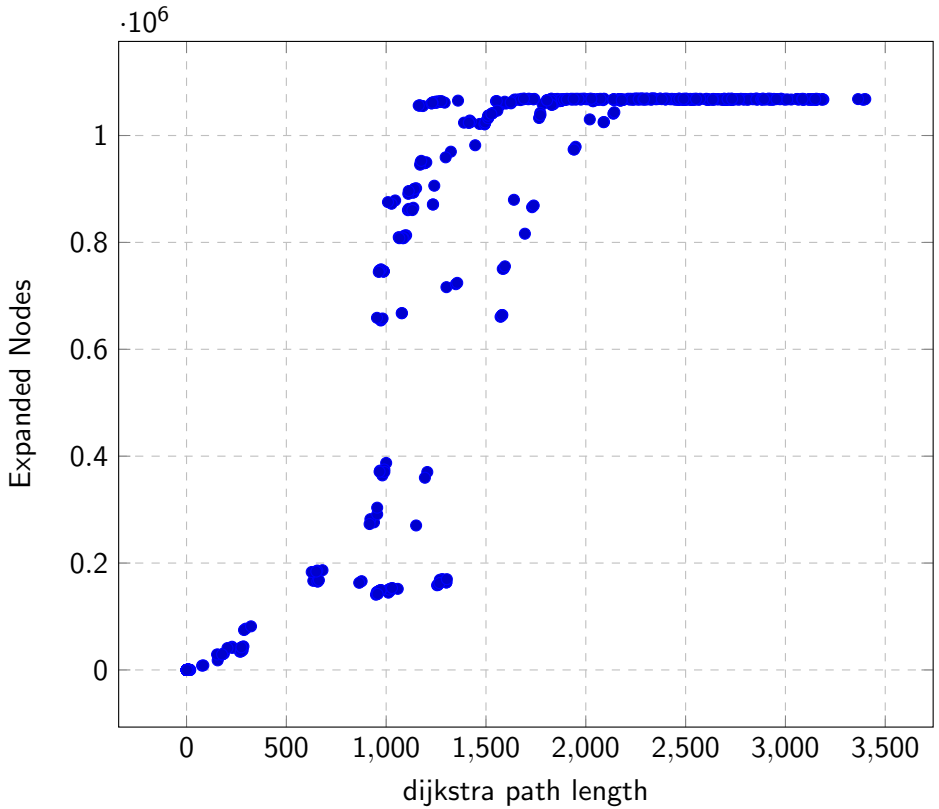
Dijkstra vs. CCH — Query Time

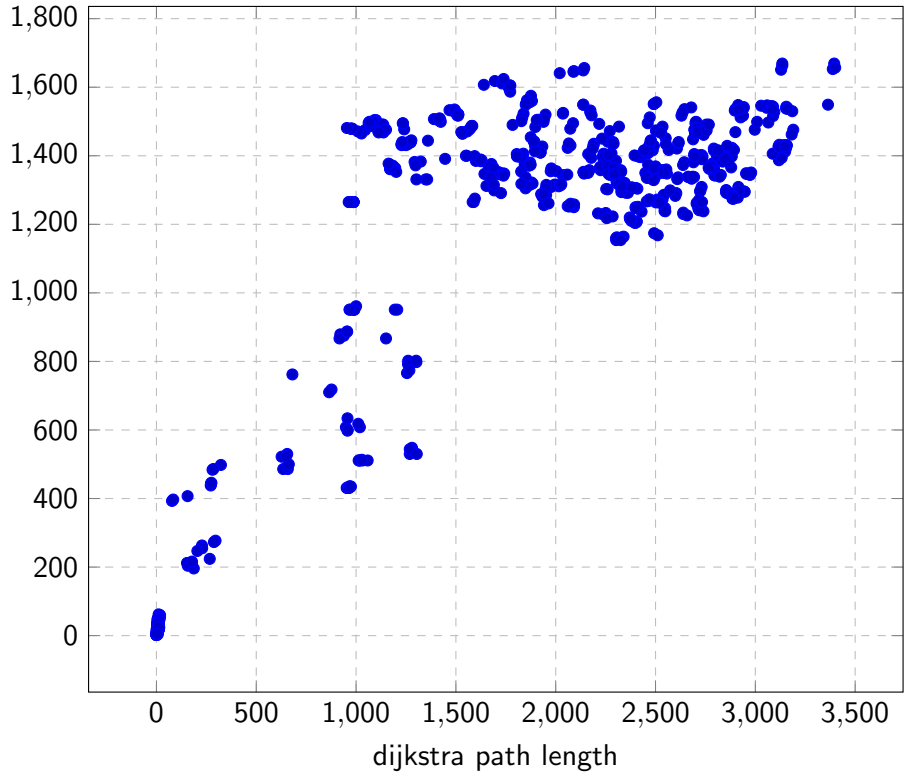


Dijkstra vs. CCH — Query Time

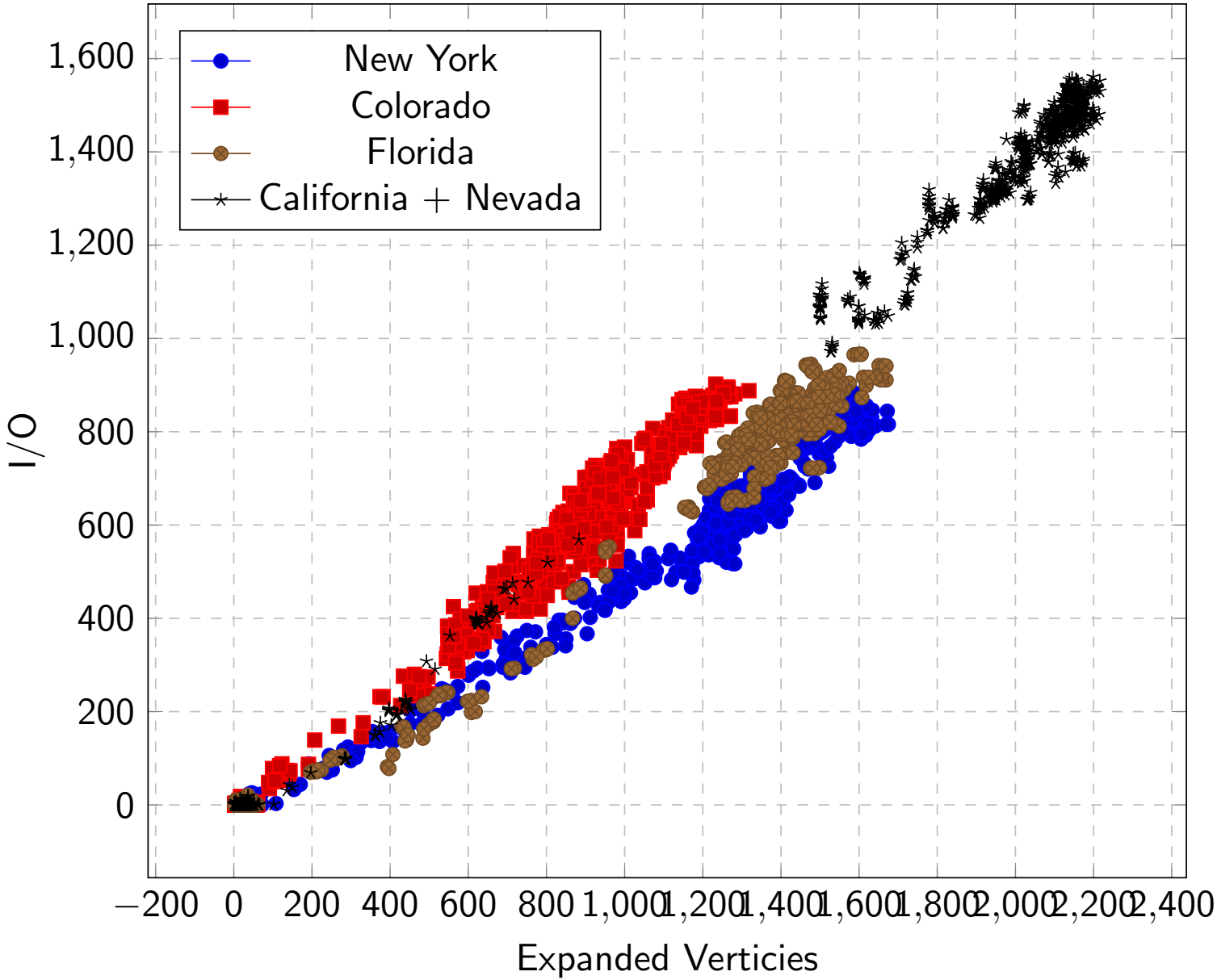


Expanded Vertices

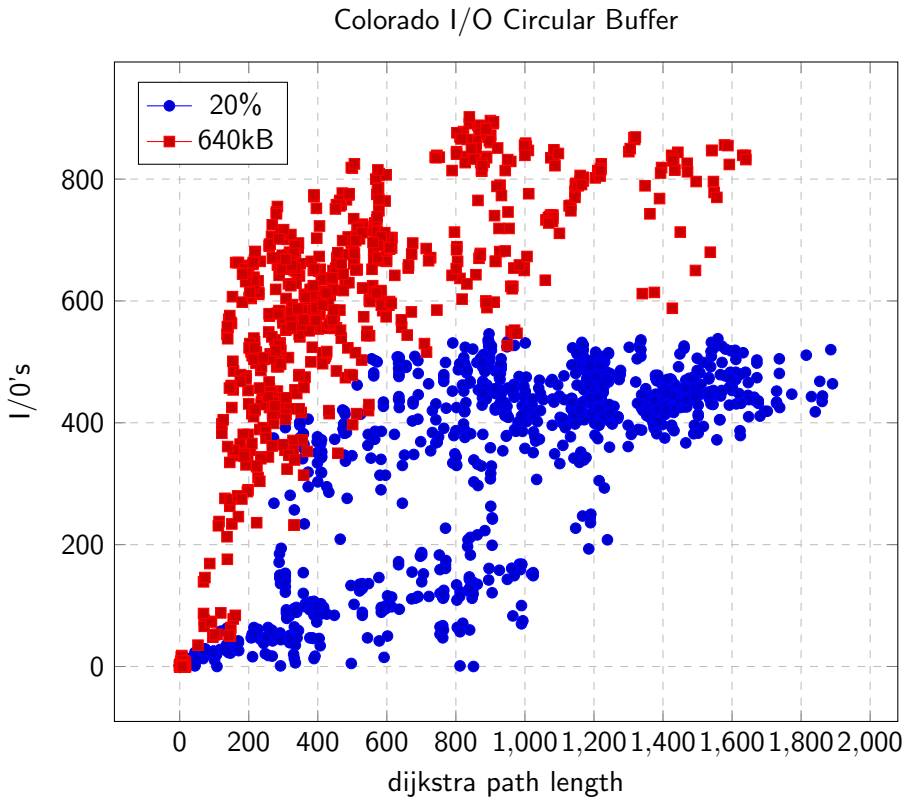




Buffer 640 kB



Buffer 20% Arc Count



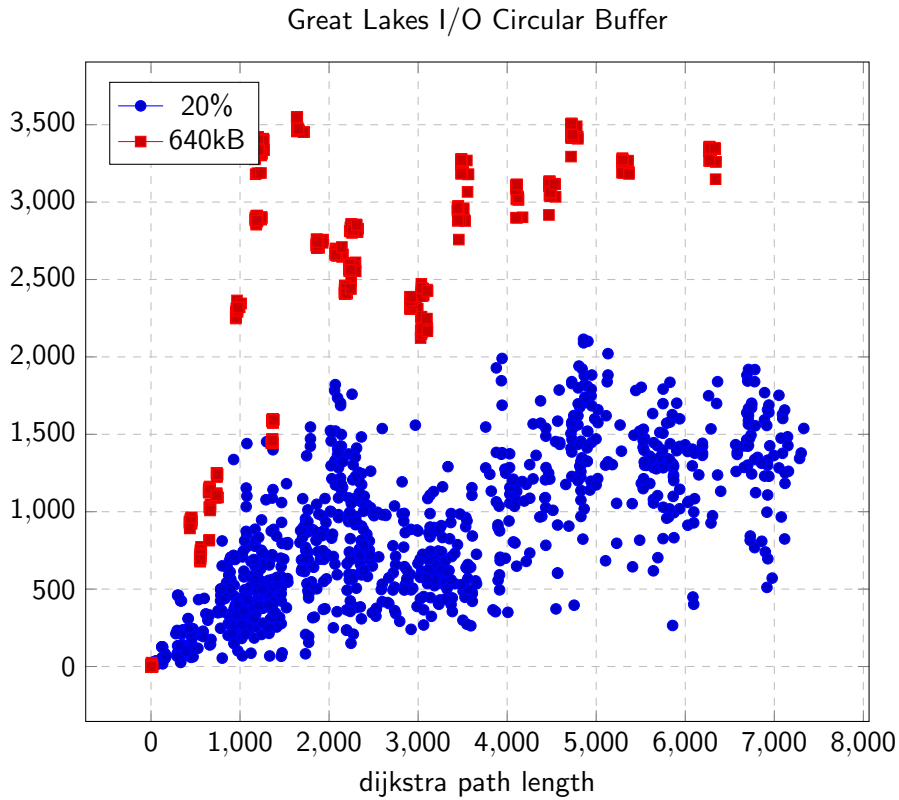


Table of Contents

Conclusion

- We can accelerate Graph Databases with CCH
- The major problem is to flatten the graph
- Try it with a Relational Database