Optimizing Indirect Memory References with `milk`

Vladimir Kiriansky, Yunming Zhang, Saman Amarasinghe

MIT

PACT ’16

September 13, 2016, Haifa, Israel

Indirect Accesses

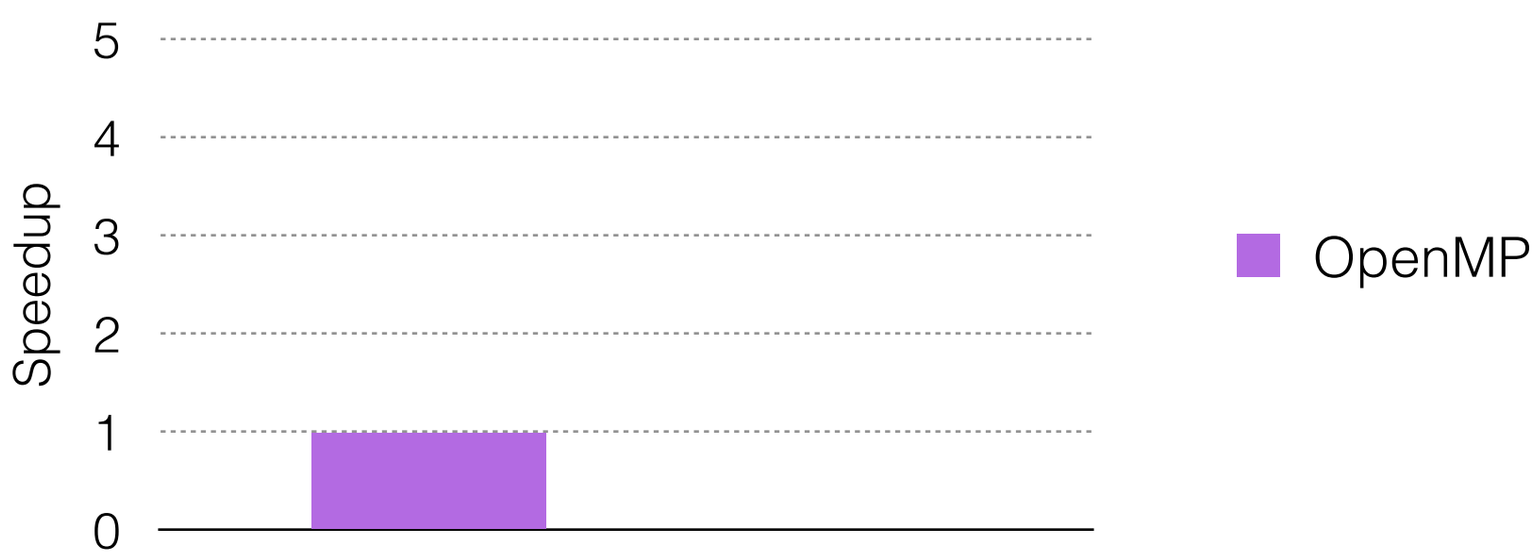
```
for(int i=0; i<N; i++)
  count[d[i]]++;
```

Indirect Accesses with OpenMP

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]++;
```

Indirect Accesses with OpenMP

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]++;
```

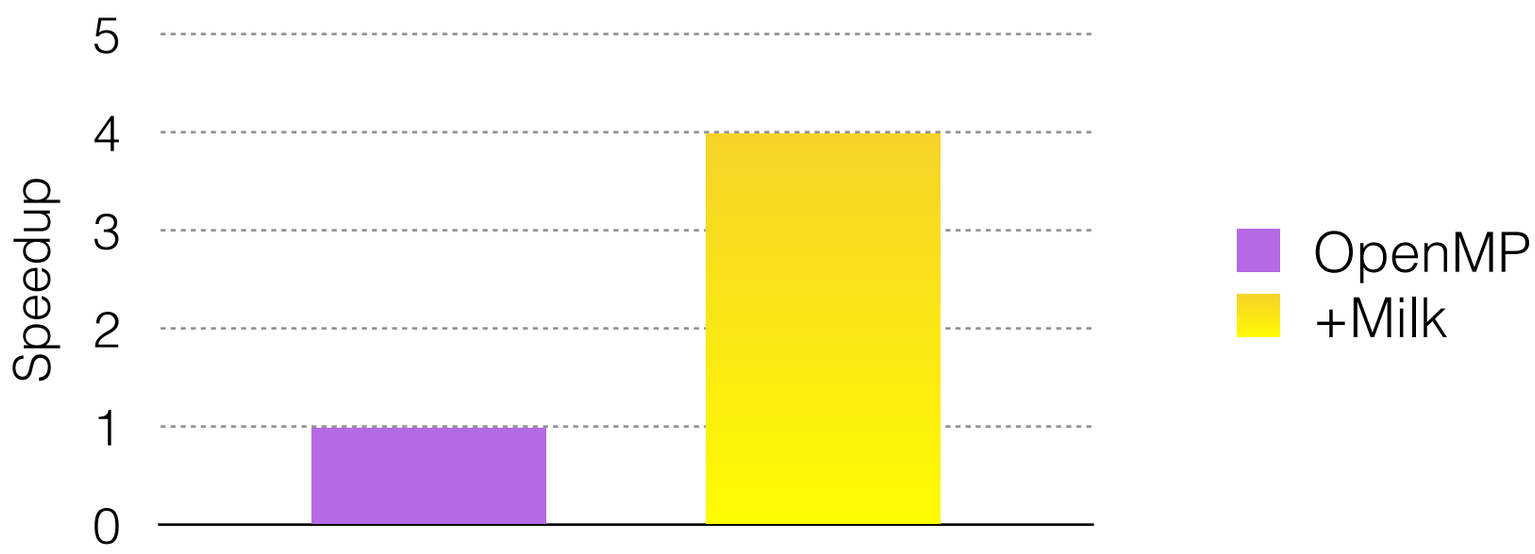


uniform [0..100M)

8 threads, 8MB L3

Indirect Accesses with **milk**

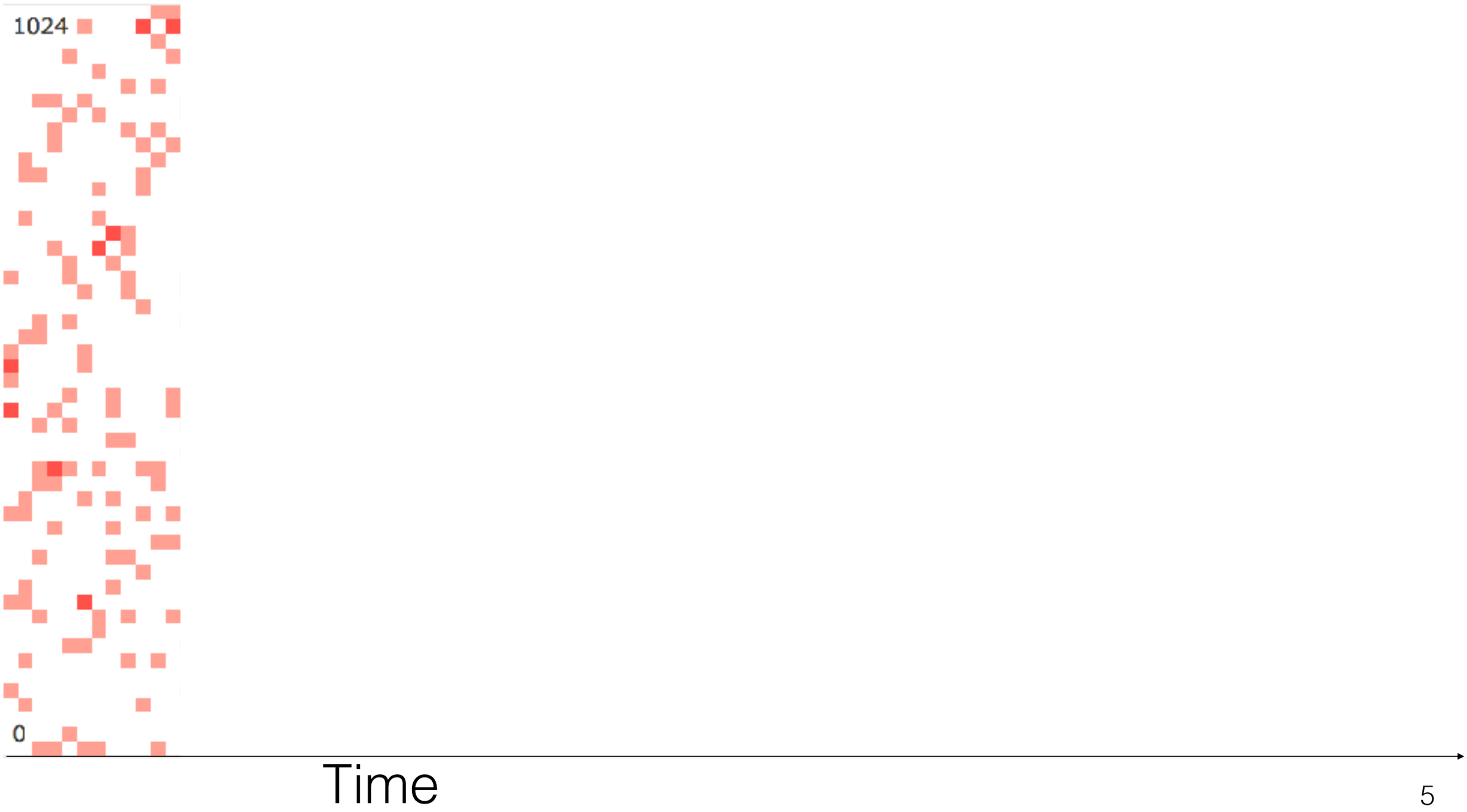
```
01 #pragma omp parallel for milk
02 for(int i=0; i<N; i++)
03     #pragma omp atomic   if(!milk)
04     count[d[i]]++;
```



uniform [0..100M)

8 threads, 8MB L3

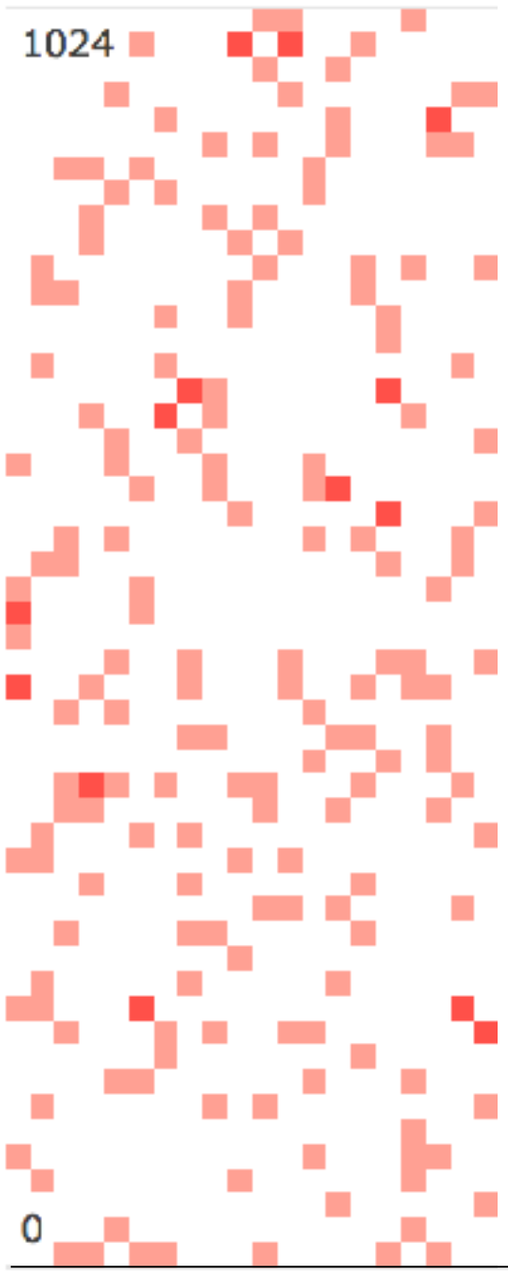
No Locality?



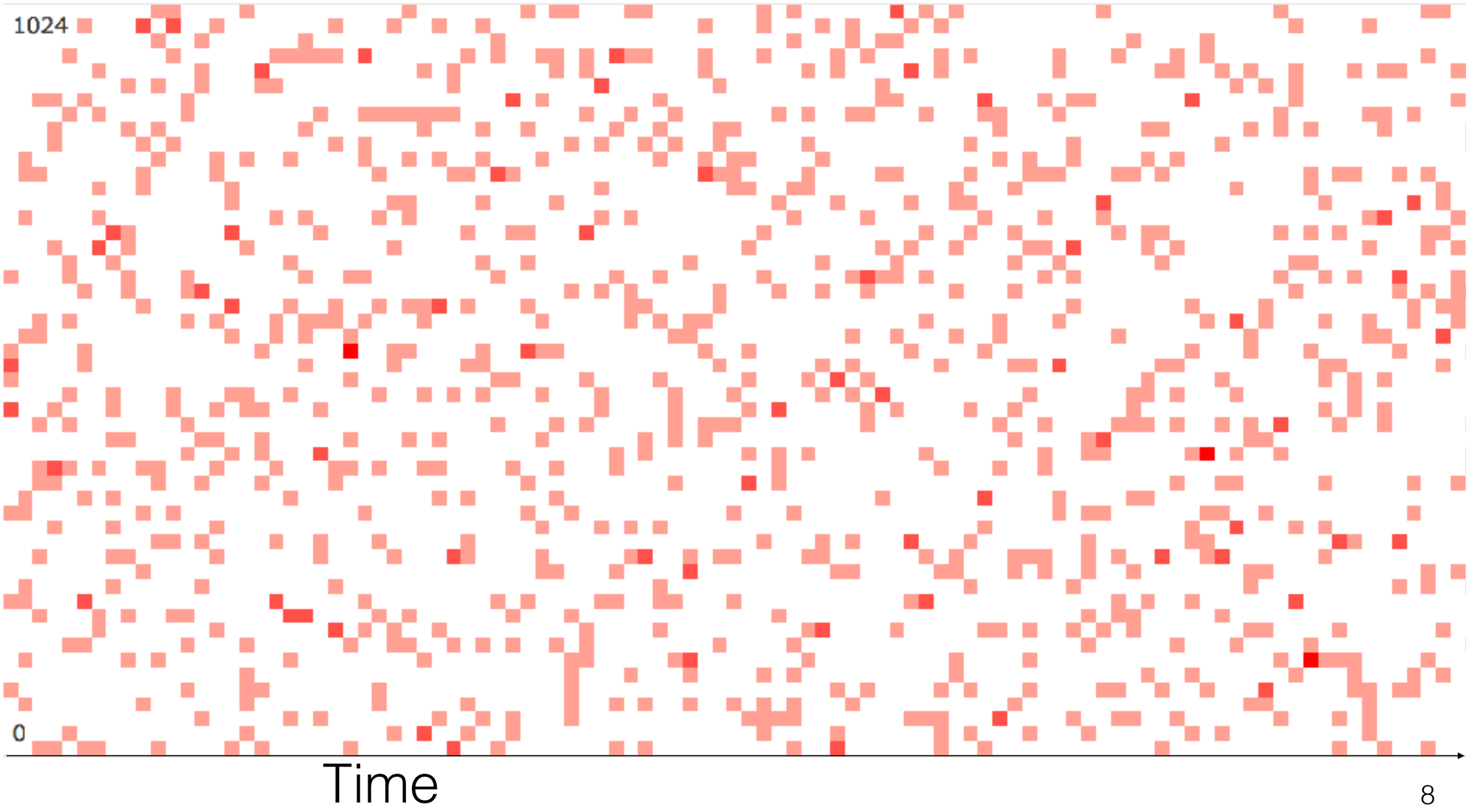
No Locality?

- Cache miss
- TLB miss
- DRAM row miss
- No prefetching

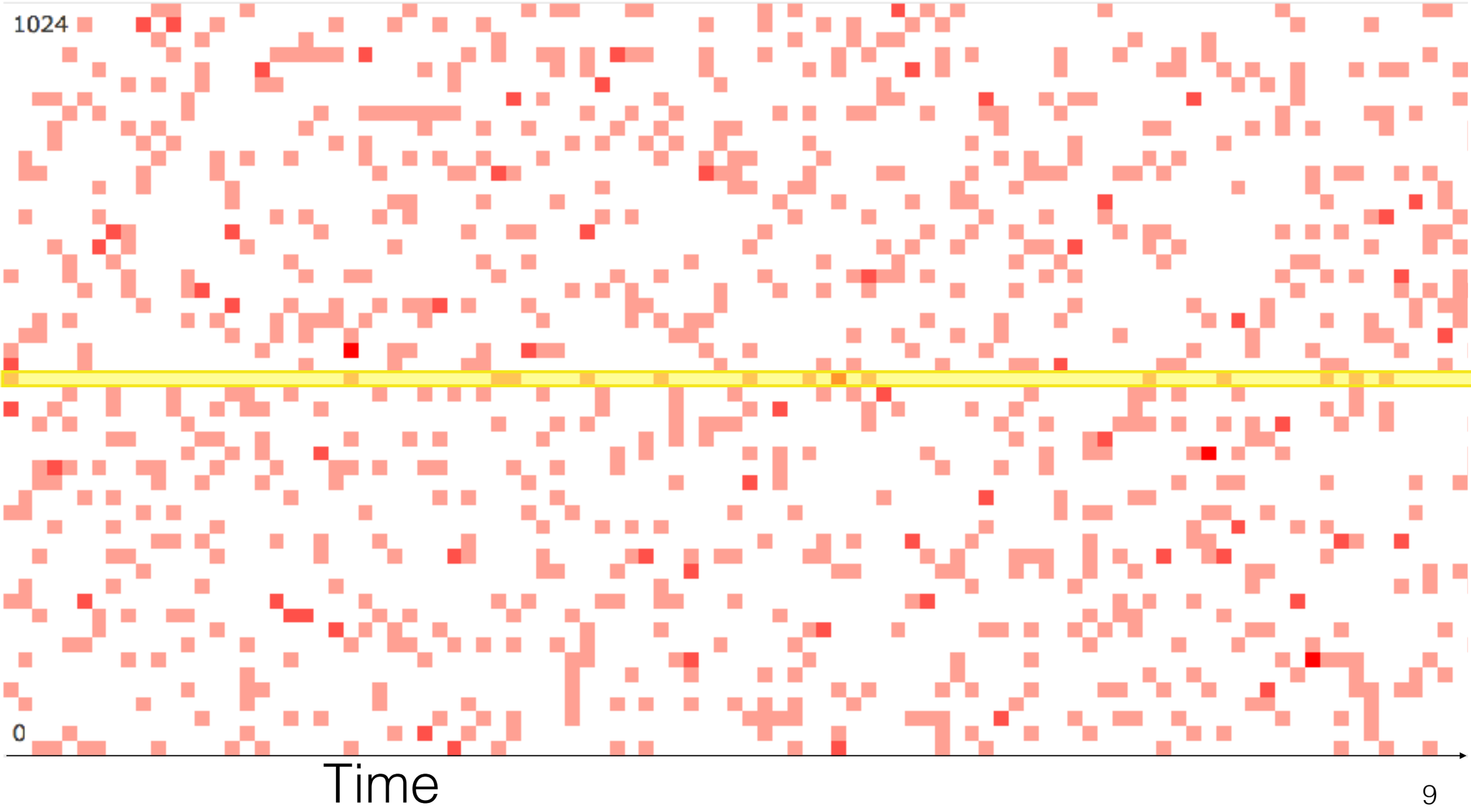
No Locality?



No Locality?

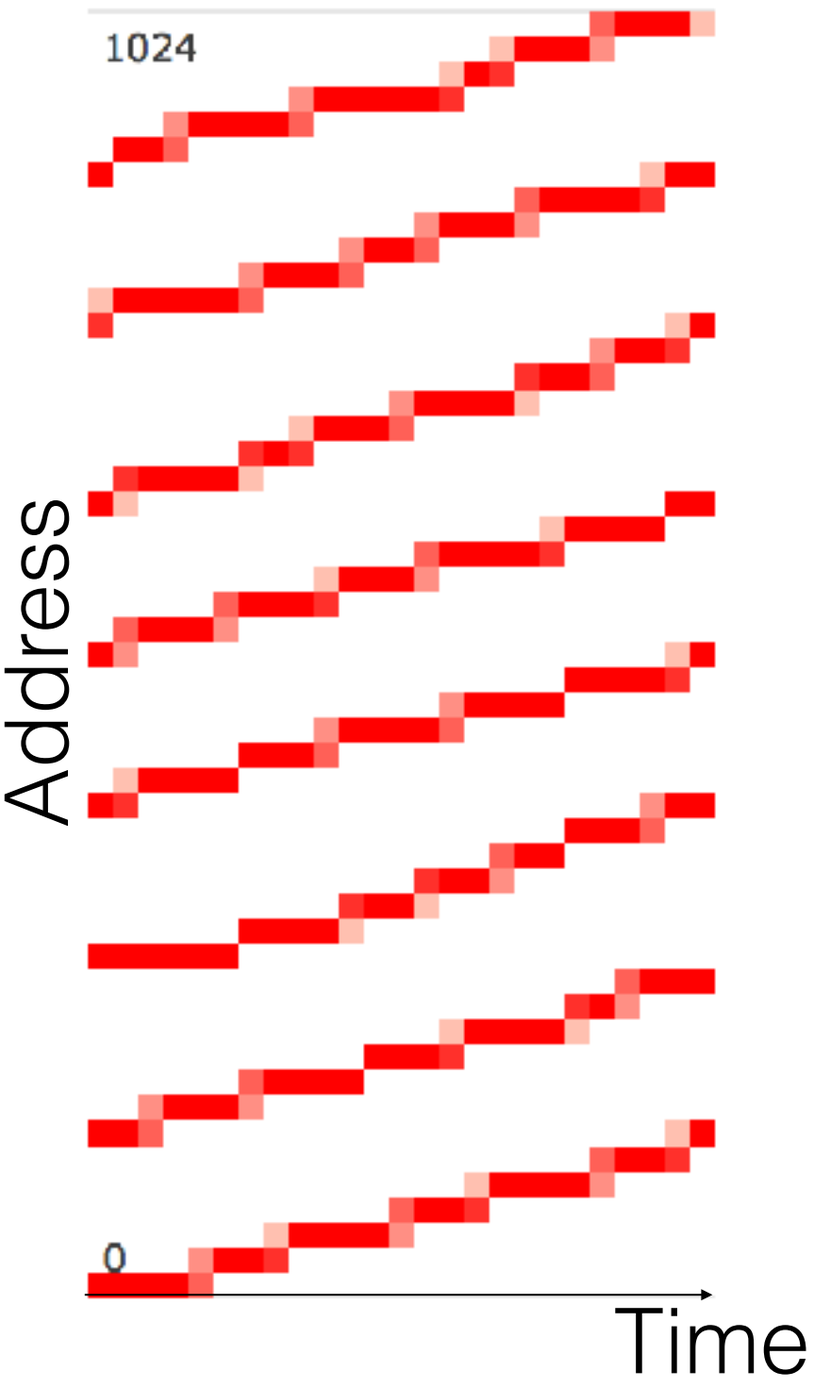


No Locality?

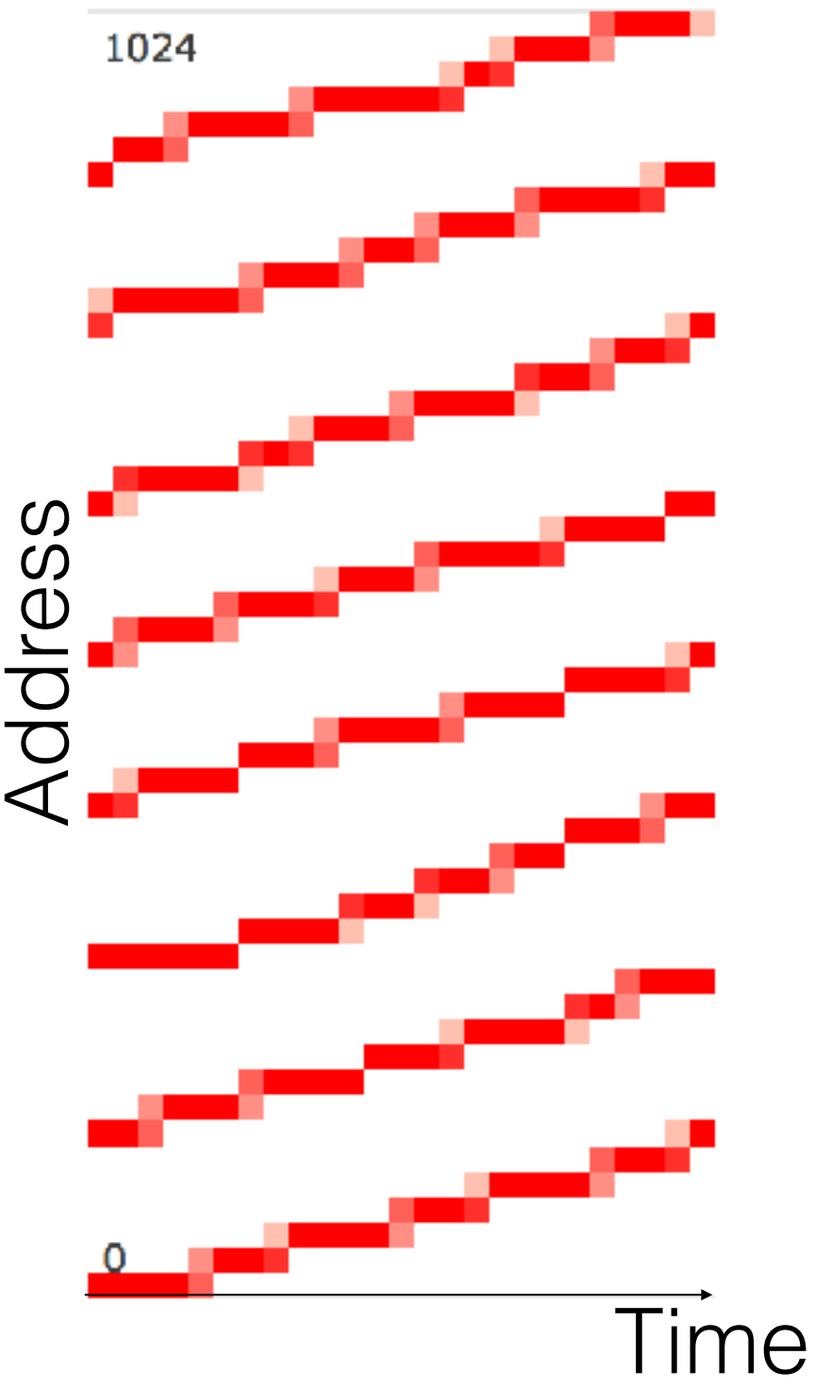


Milk Clustering

8 threads

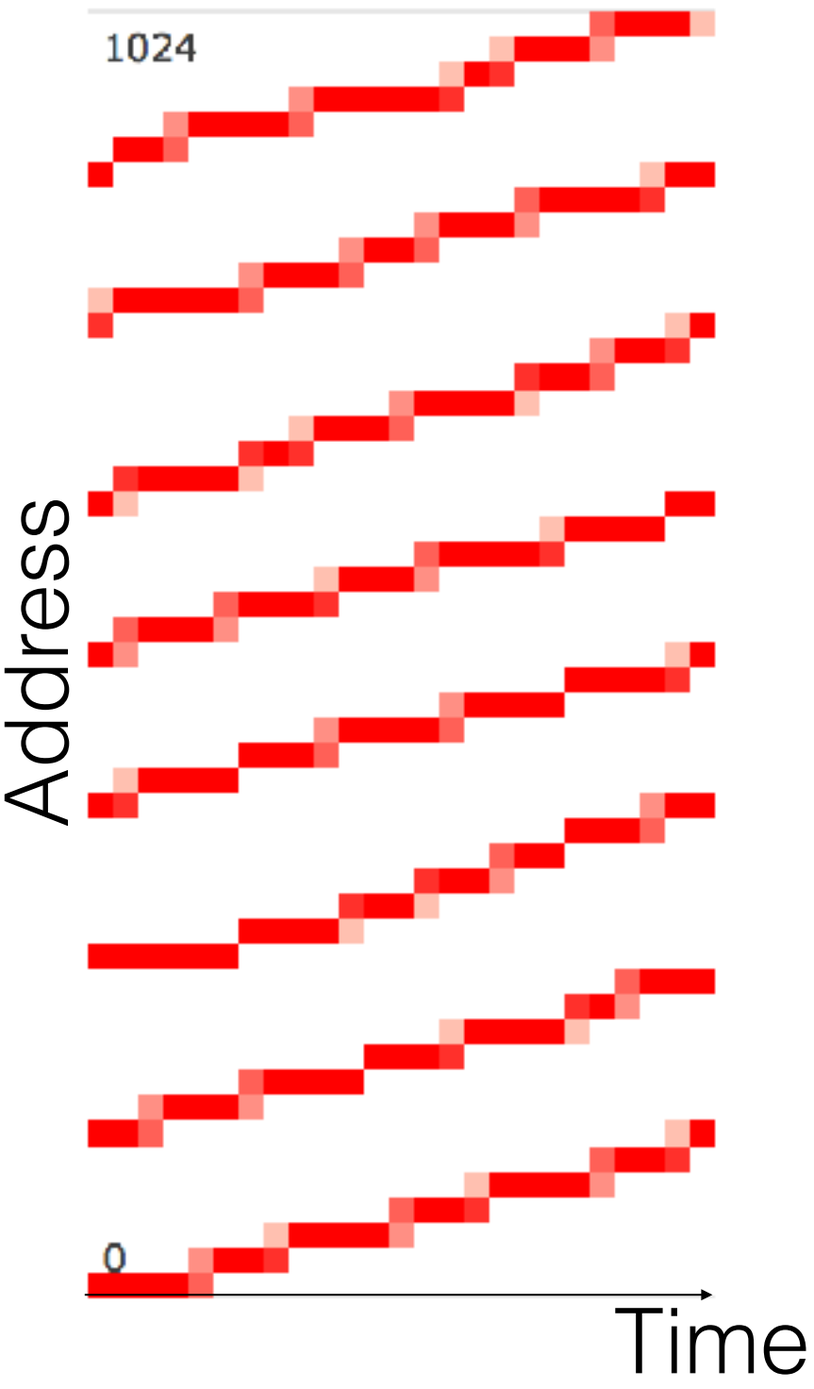


Milk Clustering



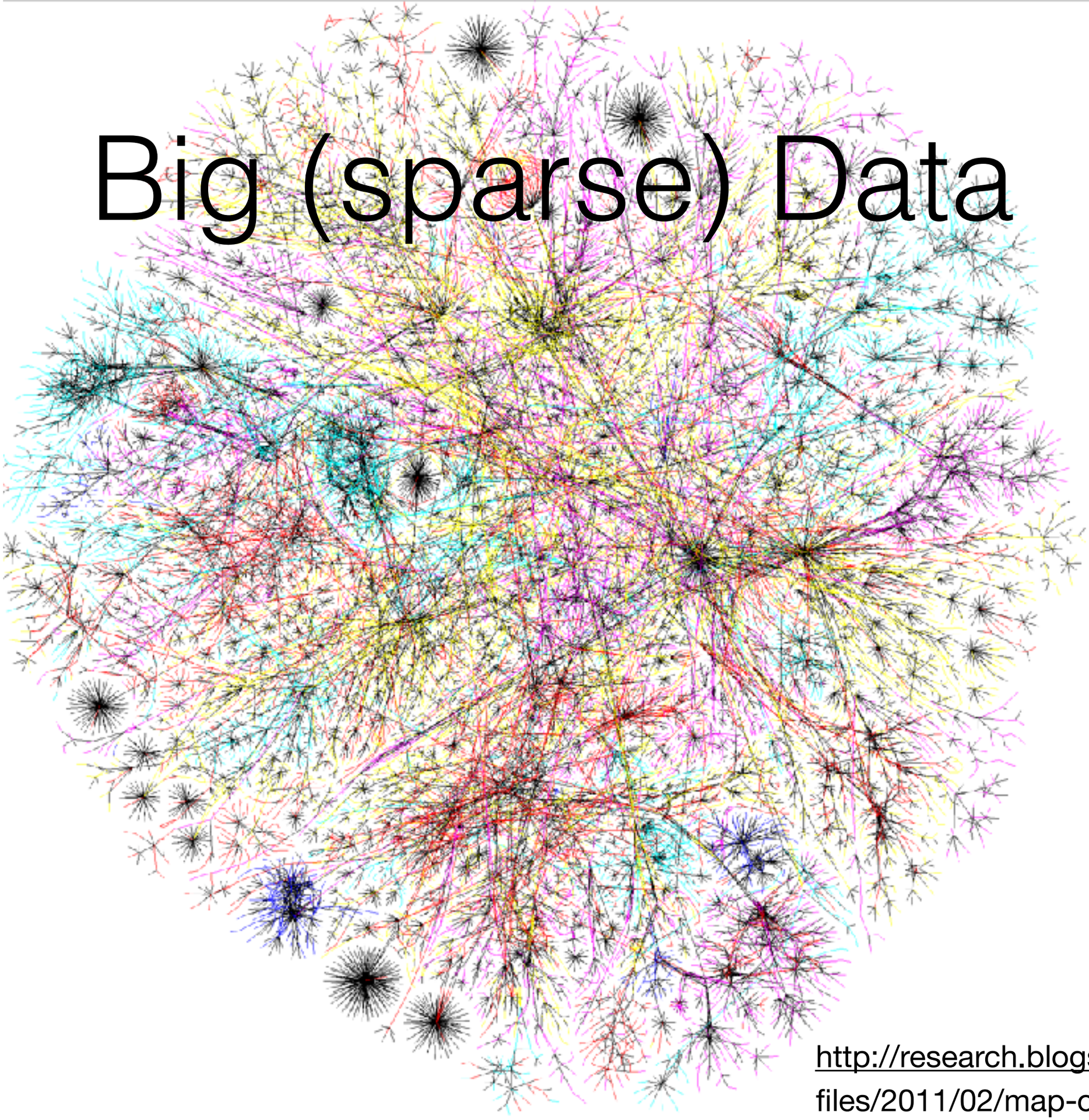
- Cache hit
- TLB hit
- DRAM row hit
- Effective prefetching

Milk Clustering



- Cache hit
- TLB hit
- DRAM row hit
- Effective prefetching
- No need for atomics!

Big (sparse) Data



http://research.blogs.lincoln.ac.uk/files/2011/02/map-of-internet.png

Big (sparse) Data

- Terabyte Working Sets
  - AWS 2TB VM
- In-memory Databases, Key-value stores
- Machine Learning
- Graph Analytics

Outline

- Milk programming model

- **`milk`** syntax

- MILK compiler and runtime

Foundations

- Milk programming model — extending BSP

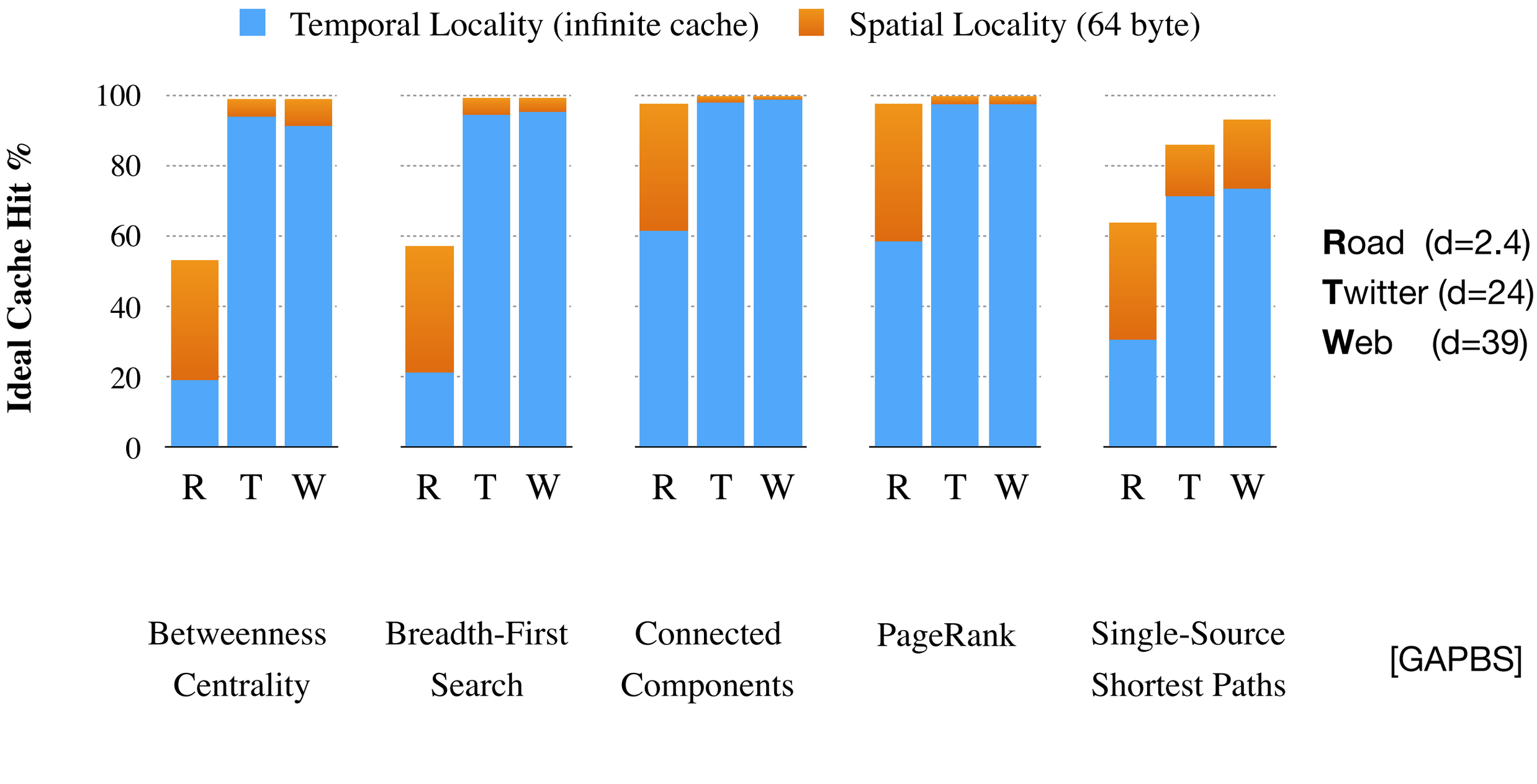
- **`milk`** syntax — OpenMP for C/C++

- **MILK** compiler and runtime — LLVM/Clang

Milk — BSP extension

- Bulk-synchronous parallel (BSP) *superstep*
  - updates visible after a barrier
- Milk *virtual processors* can access only
  - One random cache line from DRAM
  - Sequential streams
  - Cache-resident data

Superstep Locality in Graph Applications



Milk Execution Model

- Collection
- Distribution
- Delivery

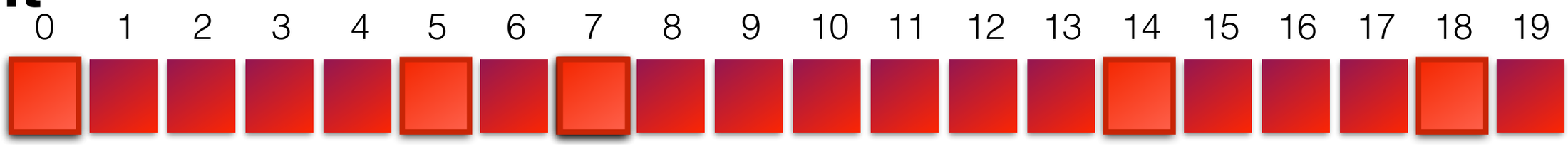
Collection

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```

d

0	1	2	3	4	5	6	7
7	0	14	5	18	7	0	7

count



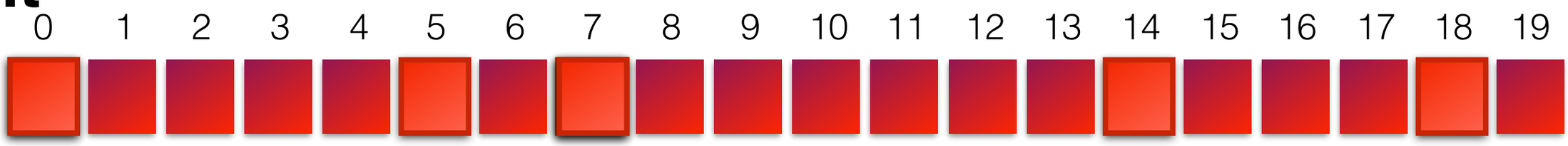
Collection

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```

d

0	1	2	3	4	5	6	7
7	0	14	5	18	7	0	7

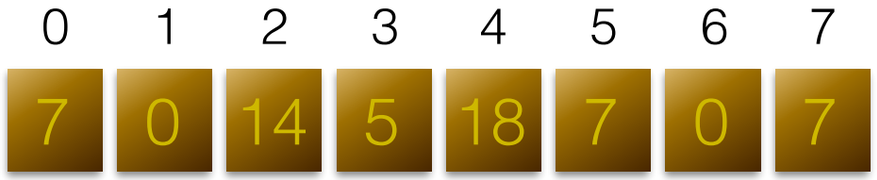
count



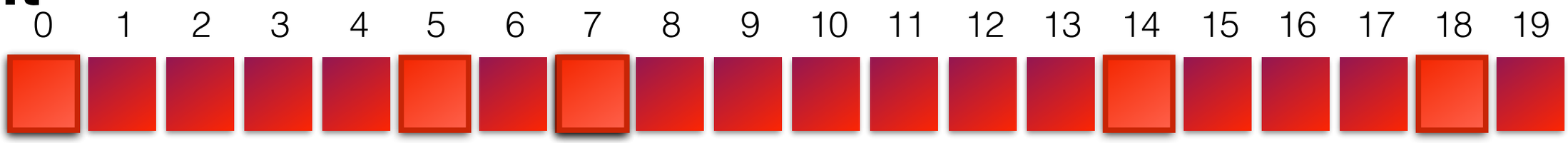
Distribution

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```

d

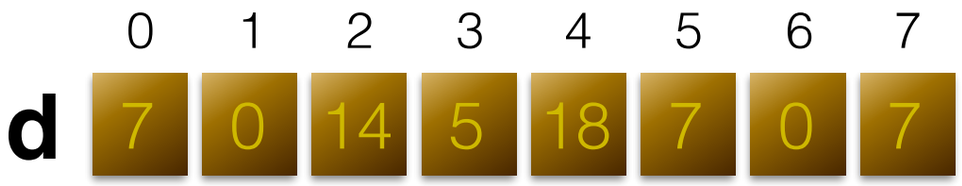


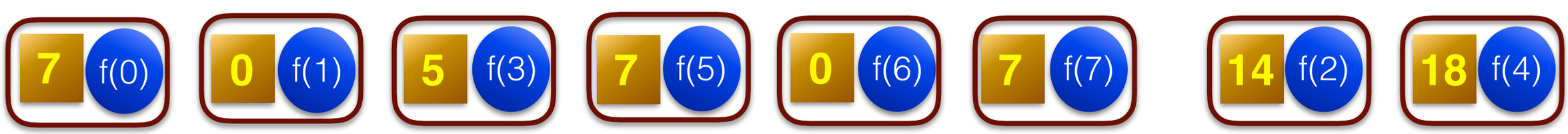
count

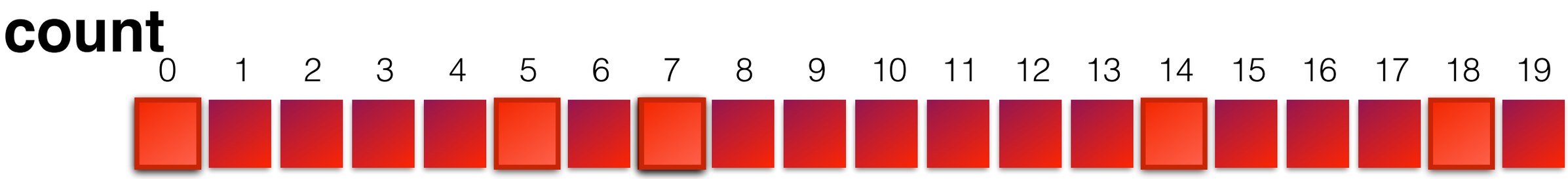


Distribution

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```

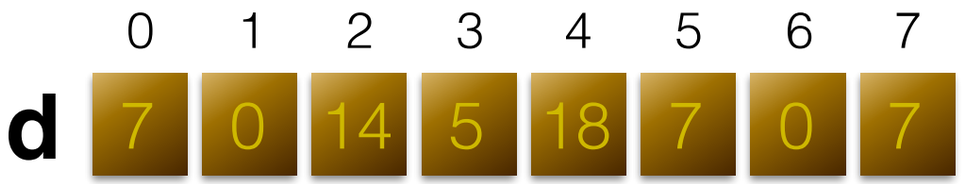


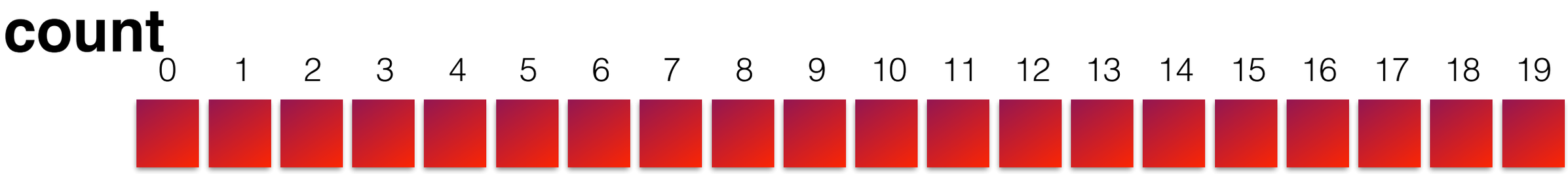




Delivery

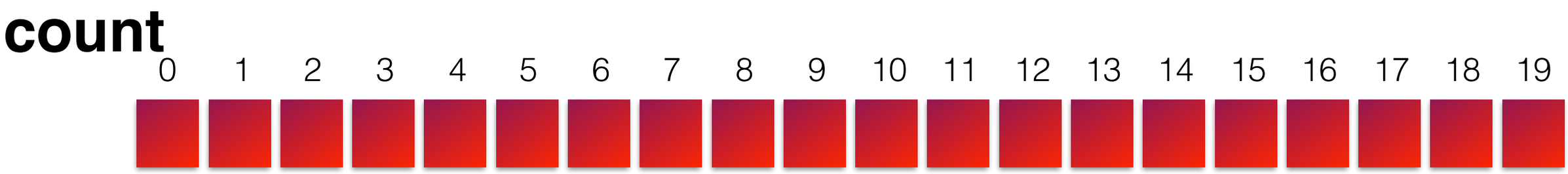
```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```





Delivery

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```



`milk` syntax

- **`milk`** clause in parallel loop
- **`milk`** directive per indirect access

0 `tag(i)` — address to group by

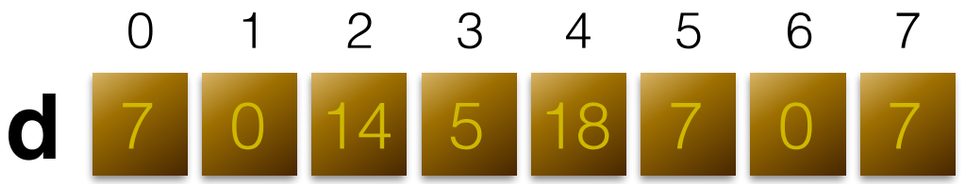
f(1) `pack(v)` — additional state

`pack` Combiners

```
pack(v[:all])
pack(v:+|*|min|max|any)
```

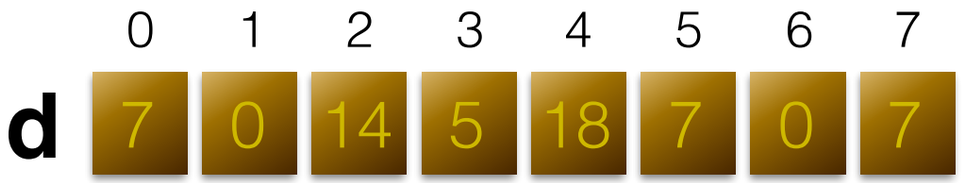
Combiners

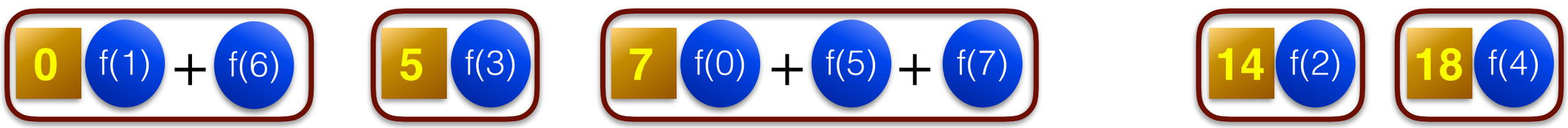
```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]+· += f(i);
```



Combiners

```
01 #pragma omp parallel for
02 for(int i=0; i<N; i++)
03     #pragma omp atomic
04     count[d[i]]++;
```





MILK compiler and runtime

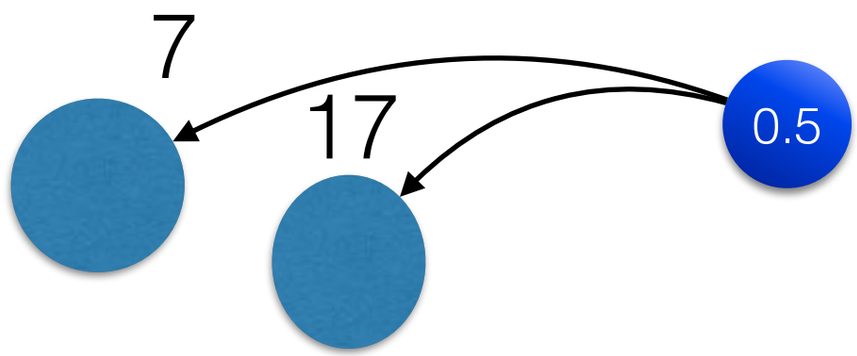
- Collection — loop transformation
- Delivery — outlined function with continuation
- Distribution — runtime library
  parallel multipass radix partitioning

Example: PageRank

```
vector<float> contrib, new_rank;

void PageRank_Push() {
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))


      new_rank[v] += contribU;
  }
}
```

Example: PageRank

```
vector<float> contrib, new_rank;

void PageRank_Push() {
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))


      new_rank[v] += contribU;
  }
}
```



PageRank with OpenMP

```
vector<float> contrib, new_rank;

void PageRank_Push() {
#pragma omp parallel for
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))

#pragma omp atomic
      new_rank[v] += contribU;
  }
}
```

PageRank with **milk**

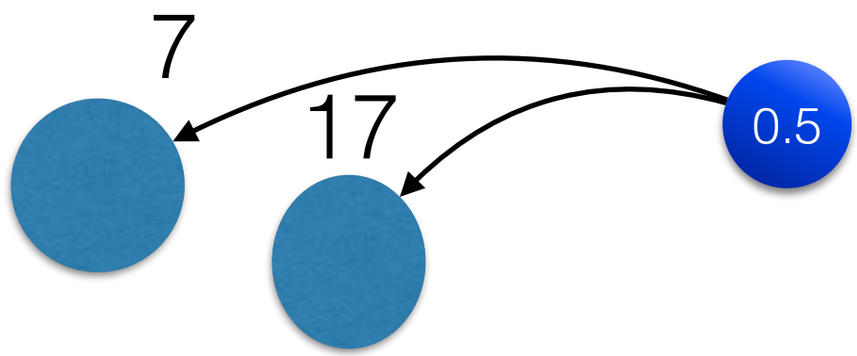
```
vector<float> contrib, new_rank;

void PageRank_Push() {
#pragma omp parallel for milk
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))

#pragma omp atomic if(!milk)
      new_rank[v] += contribU;
  }
}
```

PageRank with **milk**

```
vector<float> contrib, new_rank;

void PageRank_Push() {
#pragma omp parallel for milk
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))
#pragma milk pack(contribU : +) tag(v)
#pragma omp atomic if(!milk)
      new_rank[v] += contribU;
  }
}
```

PageRank with **milk**

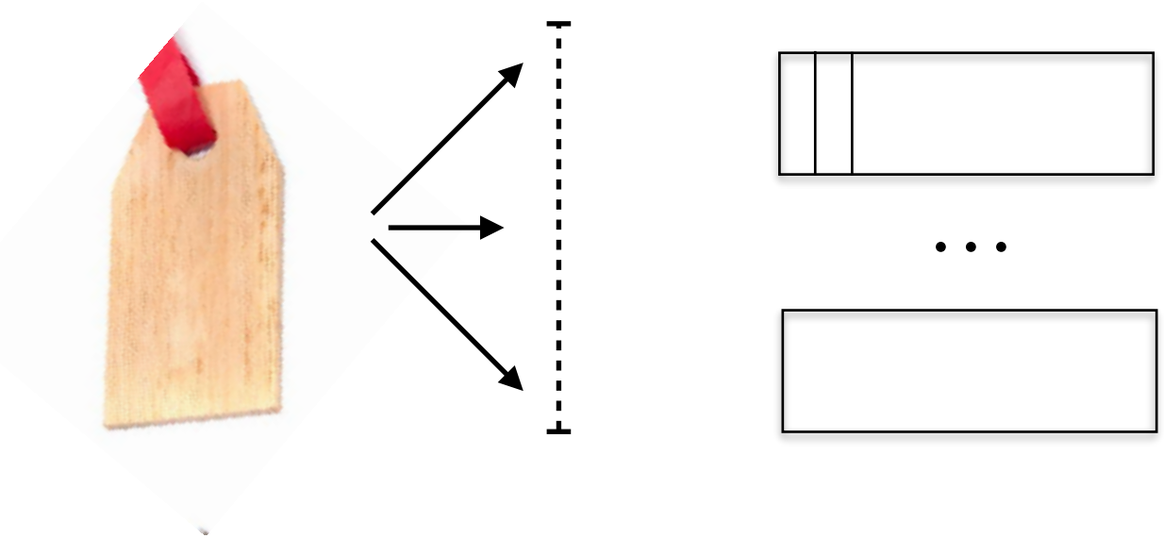
```
vector<float> contrib, new_rank;

void PageRank_Push() {
#pragma omp parallel for milk
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))
#pragma milk pack(contribU : +) tag(v)
#pragma omp atomic if(!milk)
      new_rank[v] += contribU;
  }
}
```



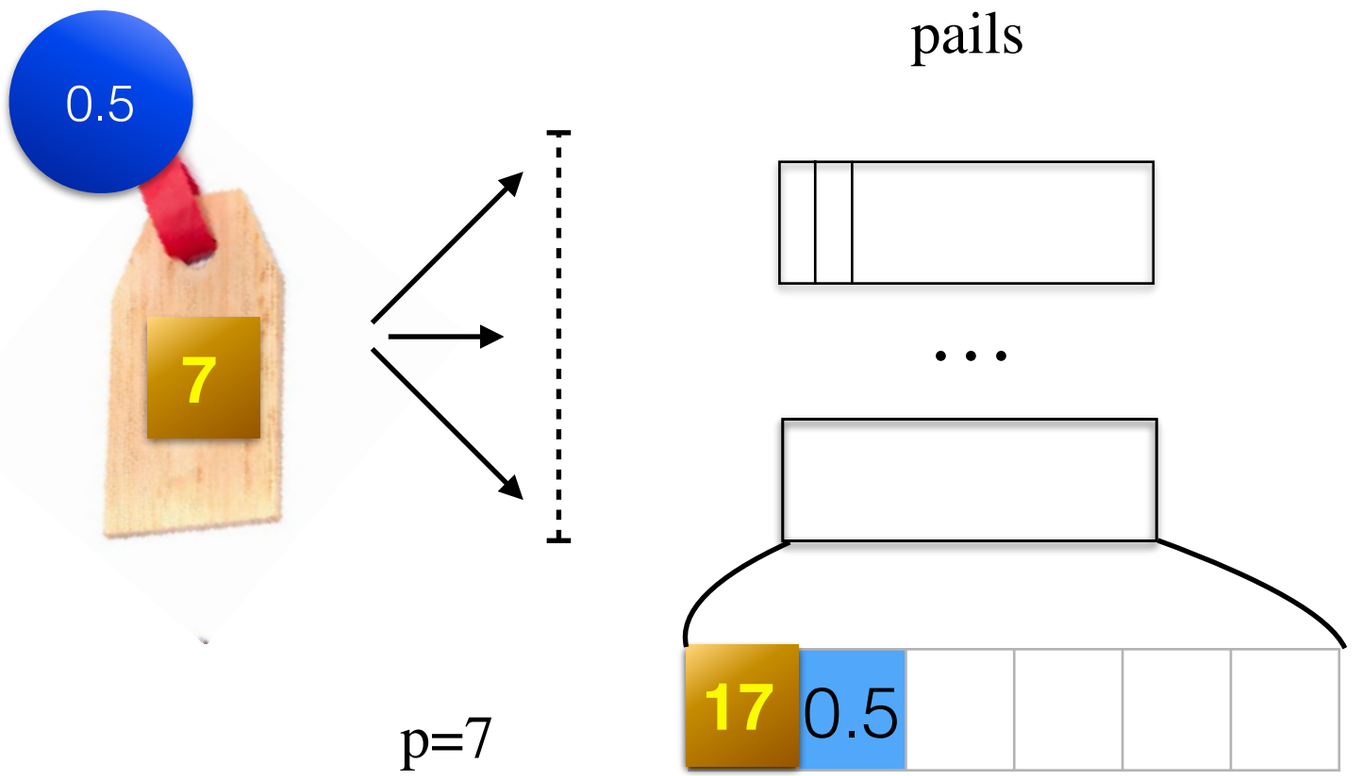
PageRank: Collection

```
vector<float> contrib, new_rank;

void PageRank_Push() {
#pragma omp parallel for milk
  for (Node u=0; u < g.num_nodes(); u++) {
    float contribU = contrib[u];
    for (Node v : g.out_neigh(u))
#pragma milk pack(contribU : +) tag(v)


  }
}
```

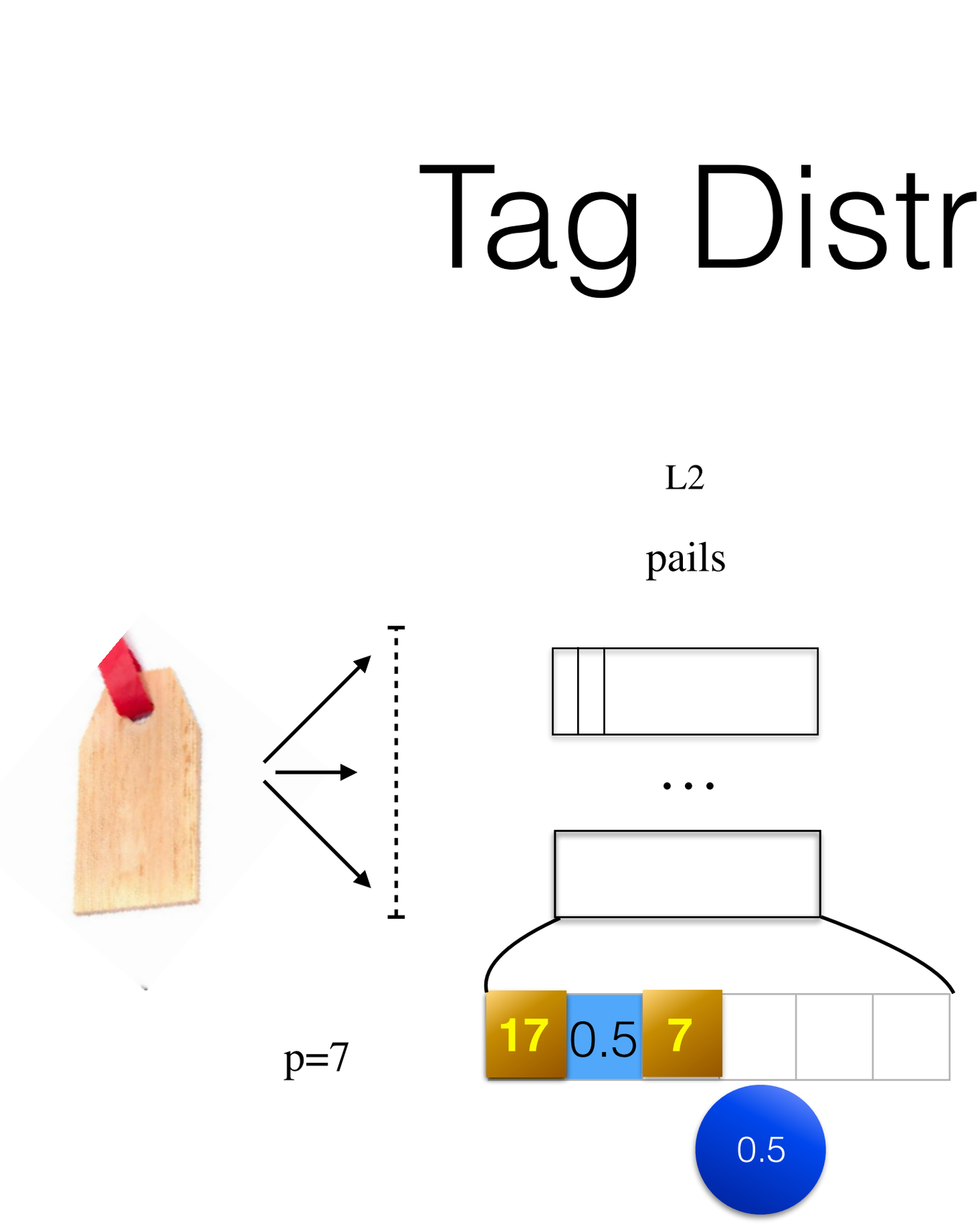
Tag Distribution



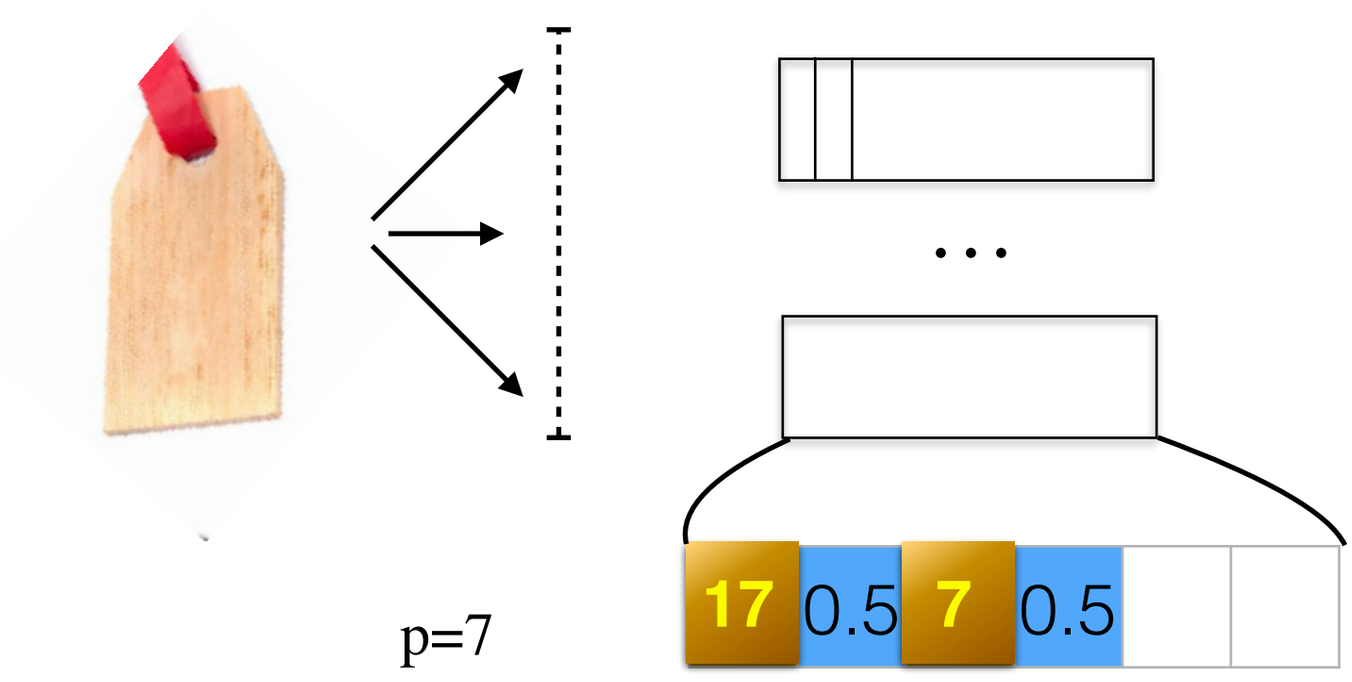
Tag Distribution



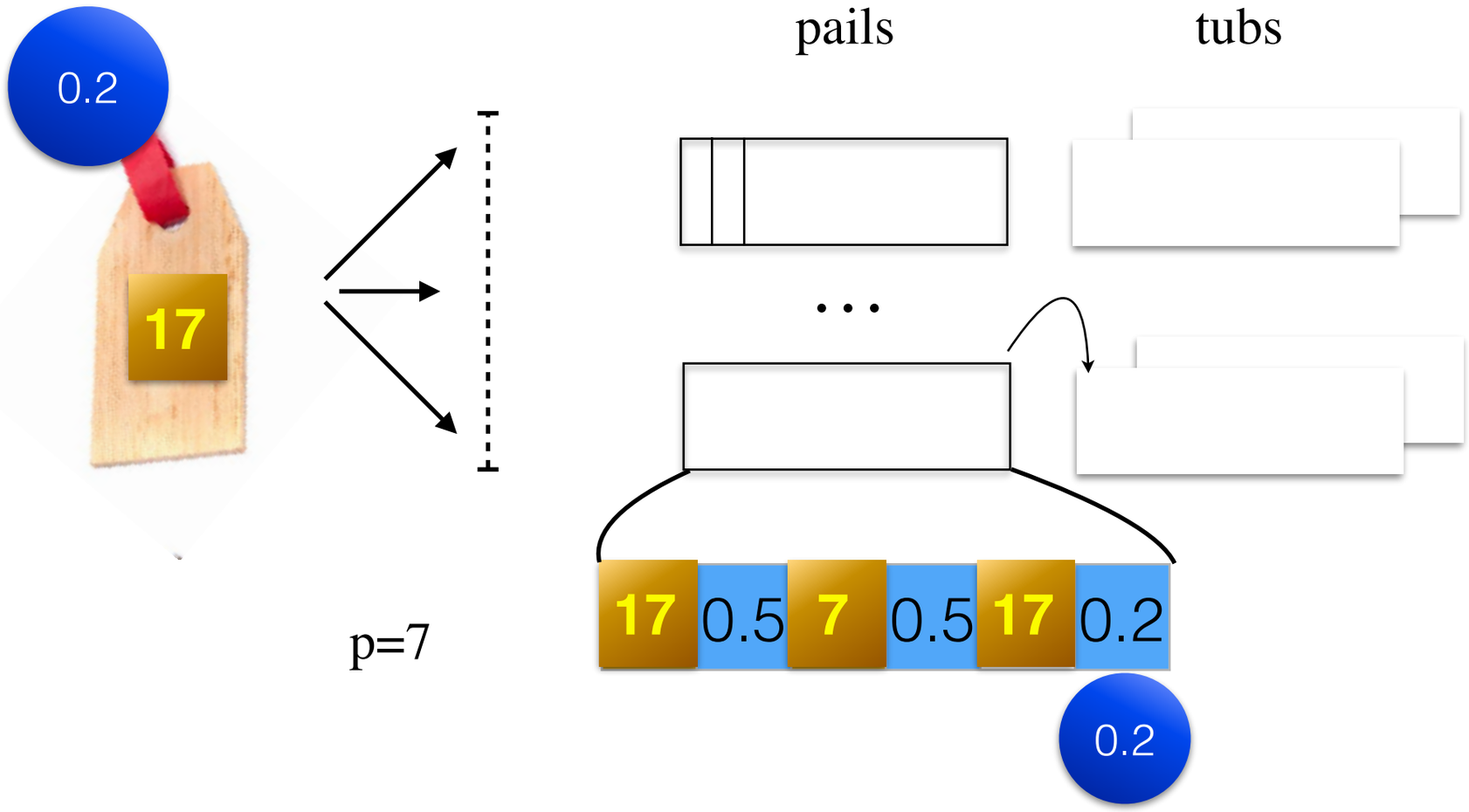
Tag Distribution



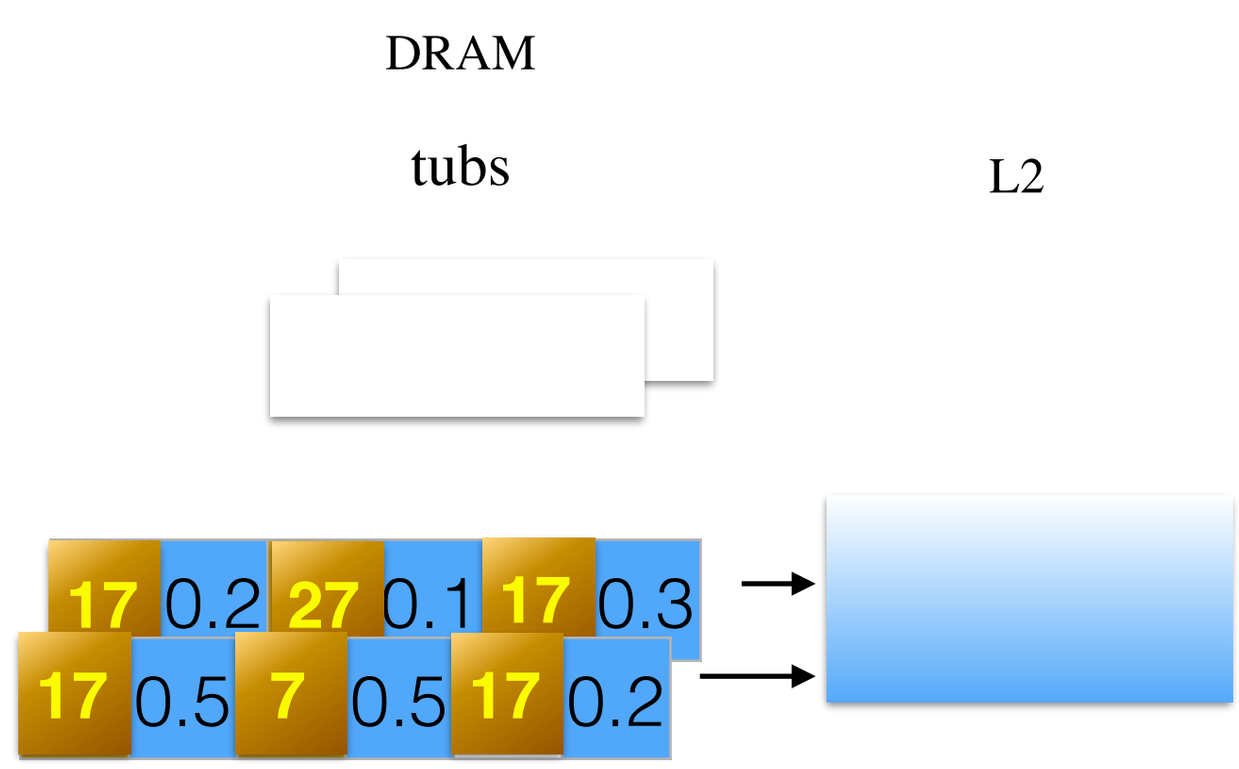
Tag Distribution



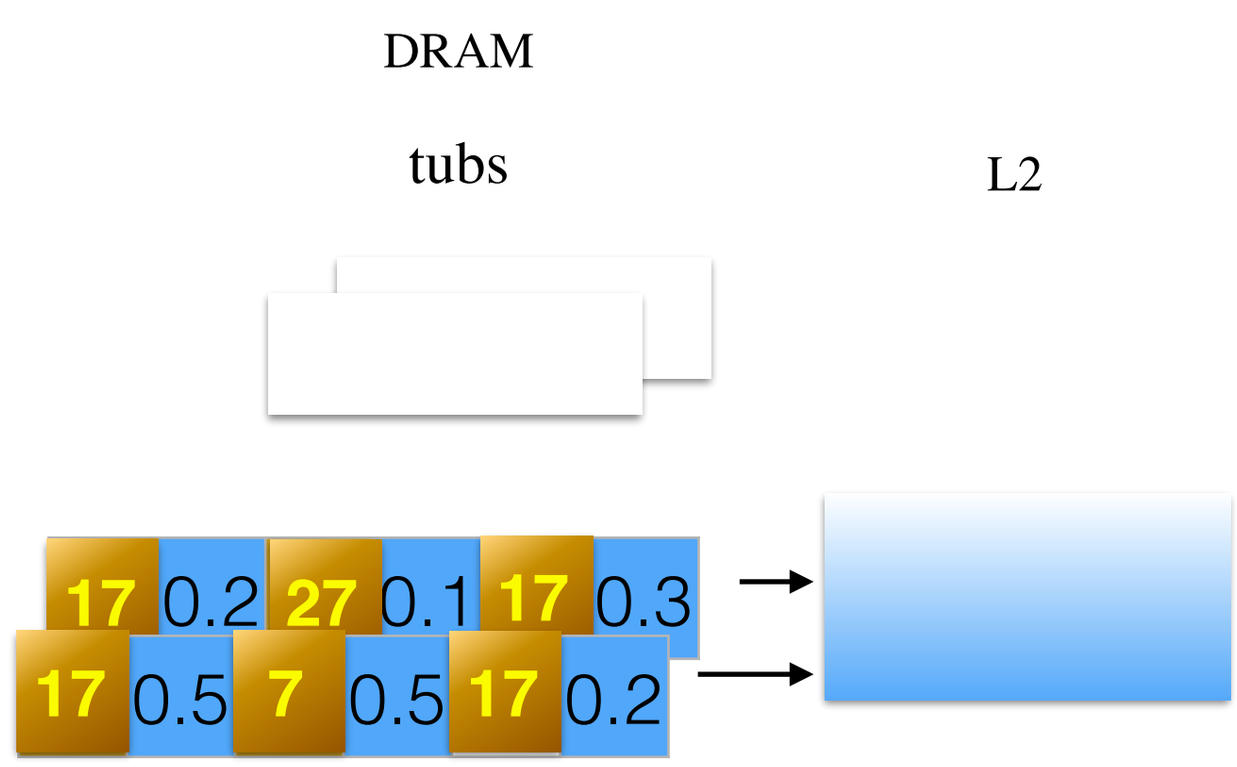
Distribution: Pail Overflow



Milk Delivery



Milk Delivery

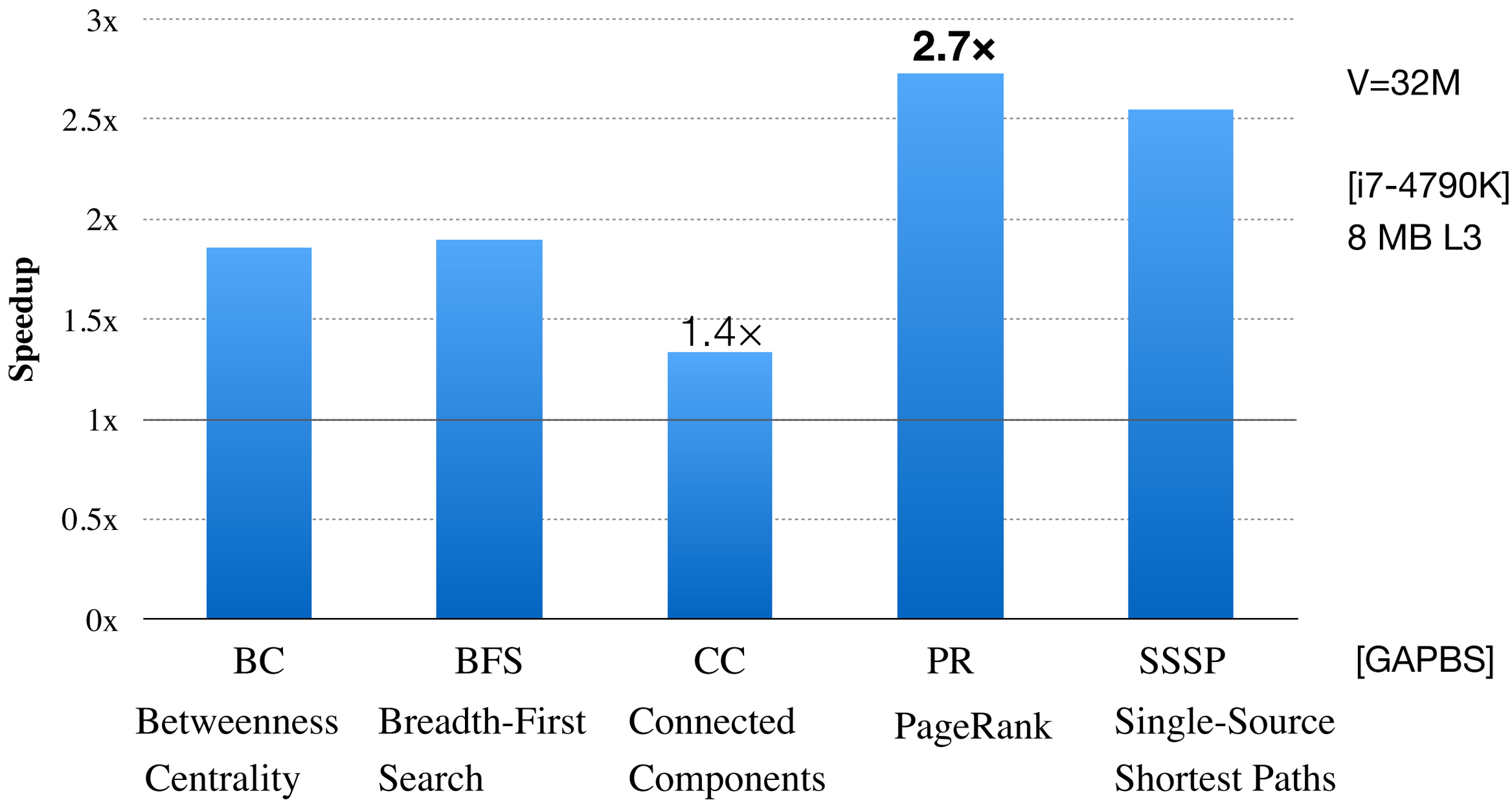



```
#pragma milk pack(contribU : +) tag(v)
#pragma omp atomic if(!milk)
        new_rank[v] += contribU;
```

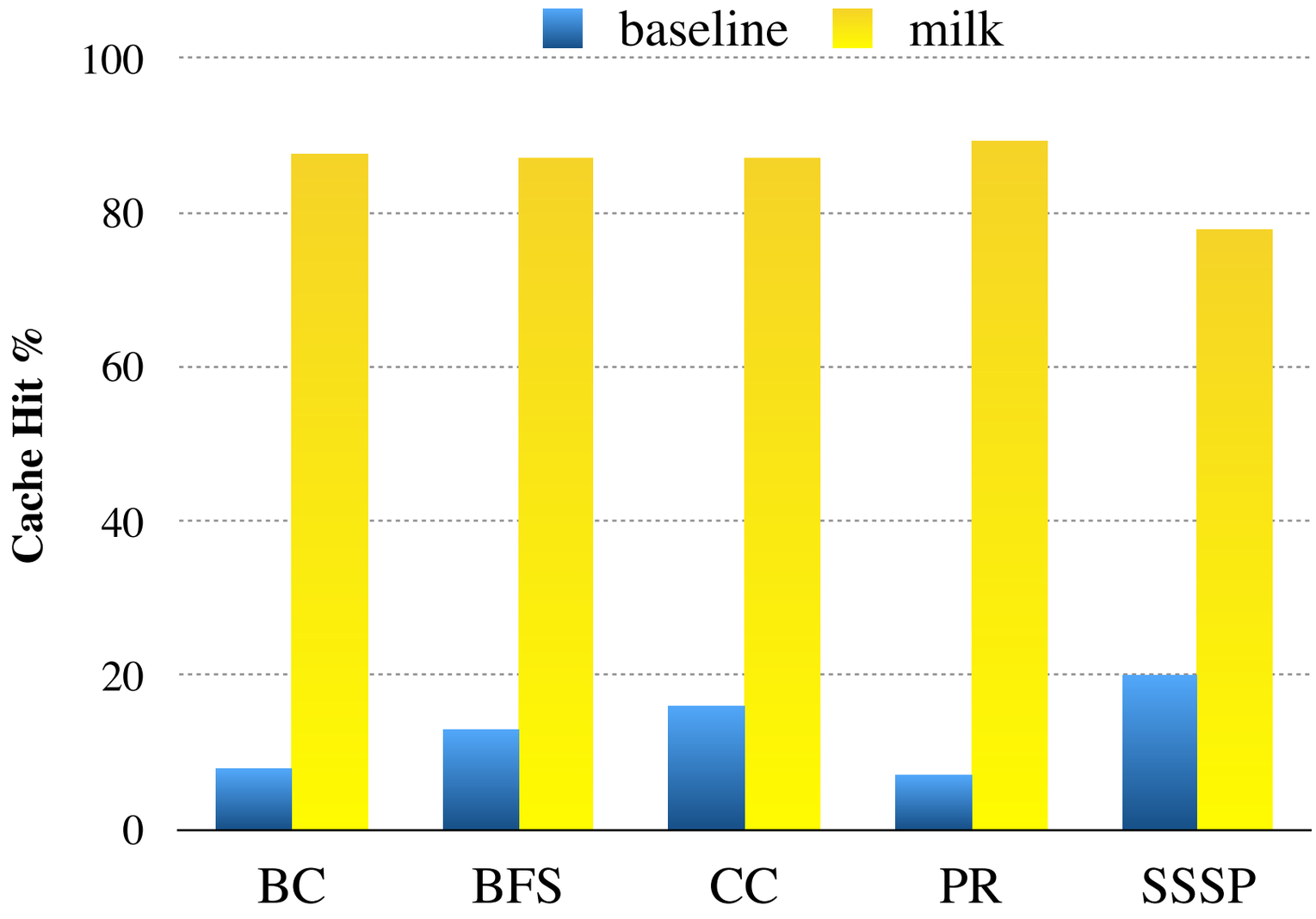
Related Work

- Database JOIN optimizations
- [Shatdal94] cache partitioning
- [Manegold02, Kim09, Albutiu12, Balkesen15]
  TLB, SIMD, NUMA,
  non-temporal writes, software write buffers

Overall Speedup with **`milk`**



Indirect Access Cache Hit%

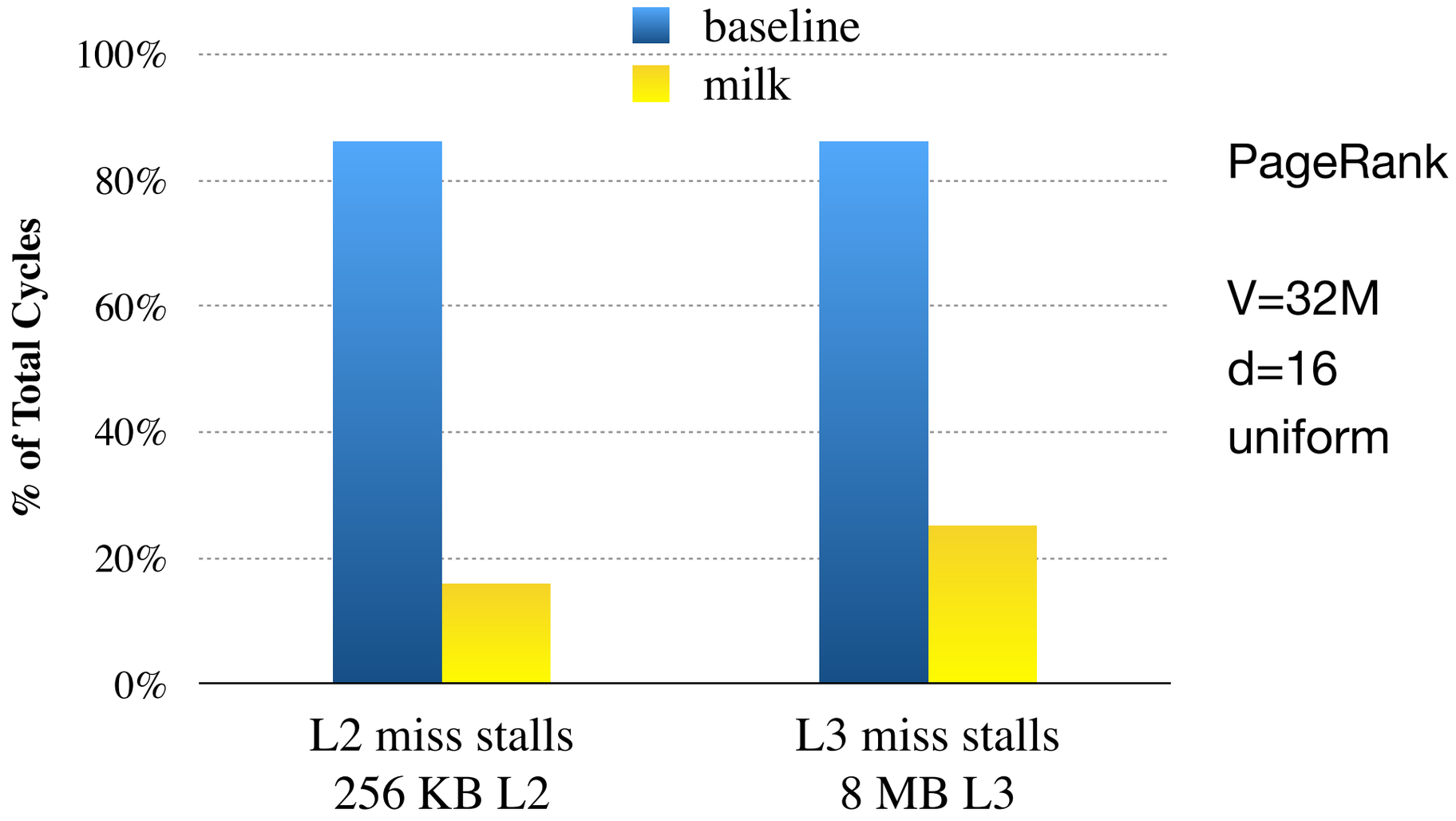


V=32M

[i7-4790K]
8 MB L3
256KB L2

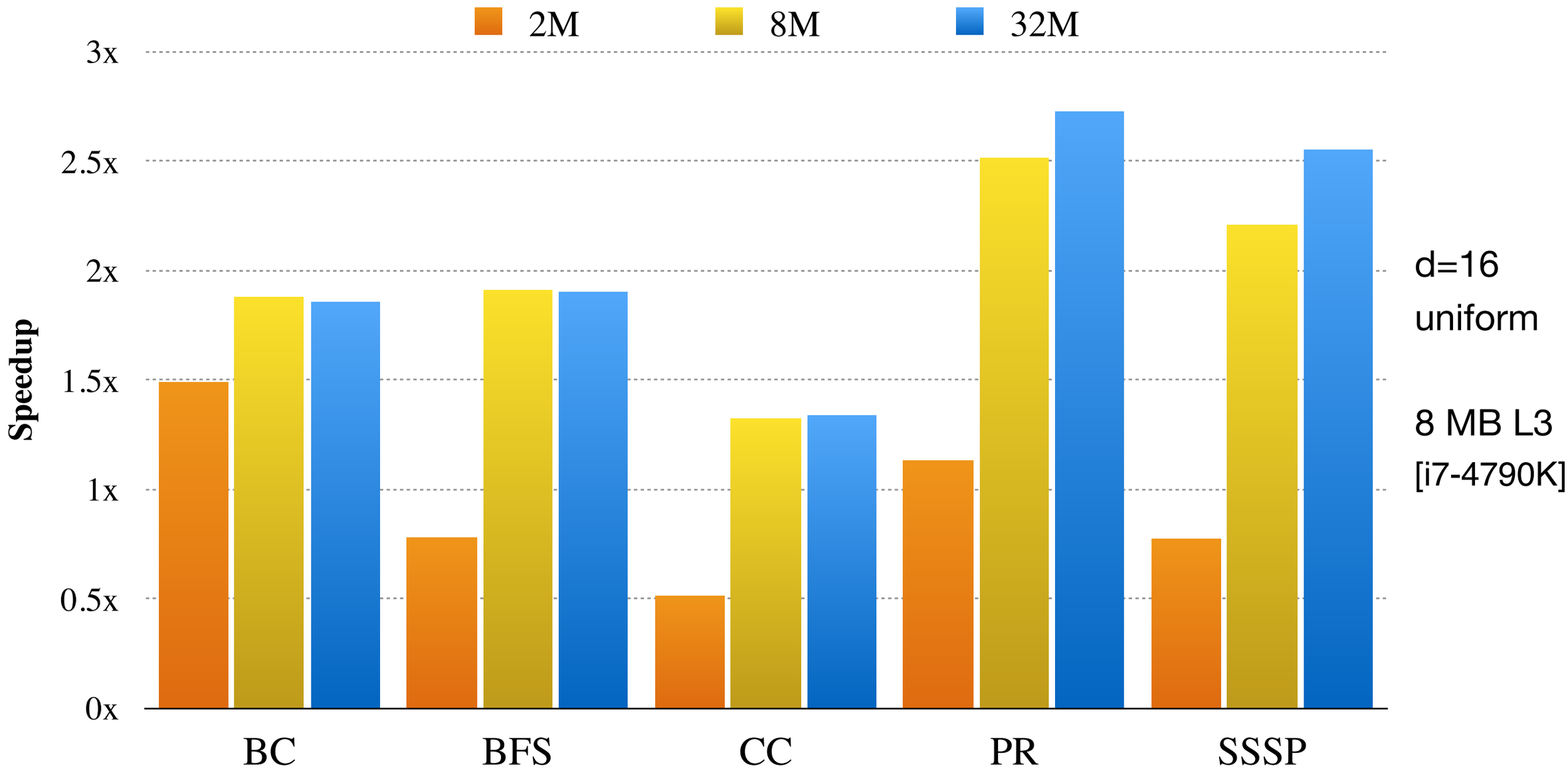
> 80% DRAM → < 22%

Stall Cycle Reduction

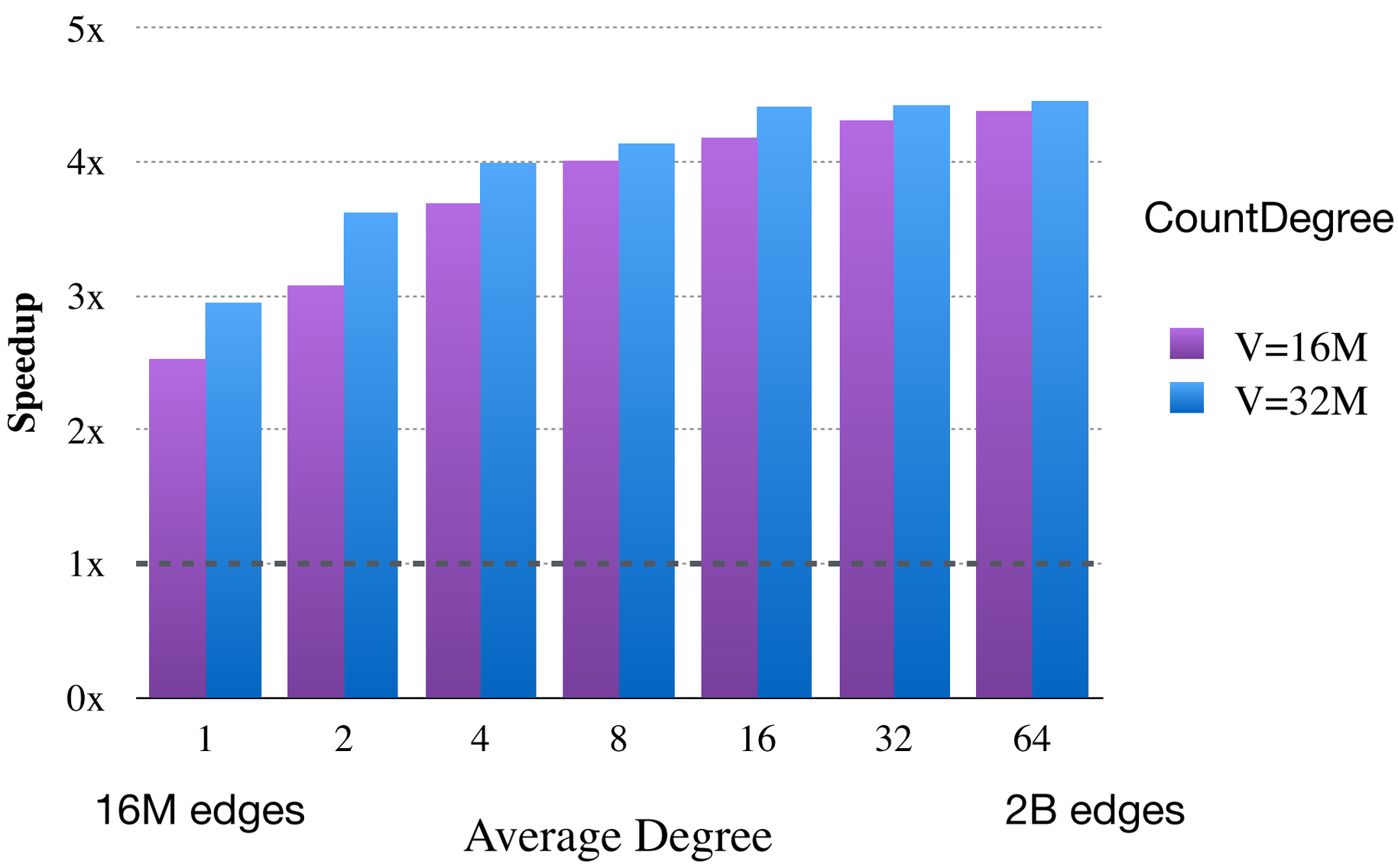


baseline: 6 of 7 cycles stalled!

Larger Graphs
→ Larger Speedups



Higher Degree
→ Higher Locality



Q & A

http://milk-lang.org/

Backup Slides

Graph Datasets

	Social			Web		Road
Graph	**Facebook**	**Twitter**	**Twitter62**	***CC12***	**.sk**	**US**
Vertices	1.5 B	300 M	62 M	3.5 B	51 M	24 M
Degree	290	200	24	36	39	2.4

[Backstrom14][Ching15][Beamer15] [CommonCrawl]

Degree Distribution