Towards cache-optimal address allocation:

How slow is your code?

Mario Preishuber

Master's Thesis

Department of Computer Sciences
University of Salzburg

Problem Statement

Given a trace T of load and store instructions

find metrics that characterize the trace performance
for a given cache C and

implement an execution engine for computing their
quantities.

Problem Statement

Given a **trace T** of load and store instructions

find **metrics** that characterize the trace **performance**
for a given cache C and

implement an **execution engine** for computing their quantities.

Problem Statement

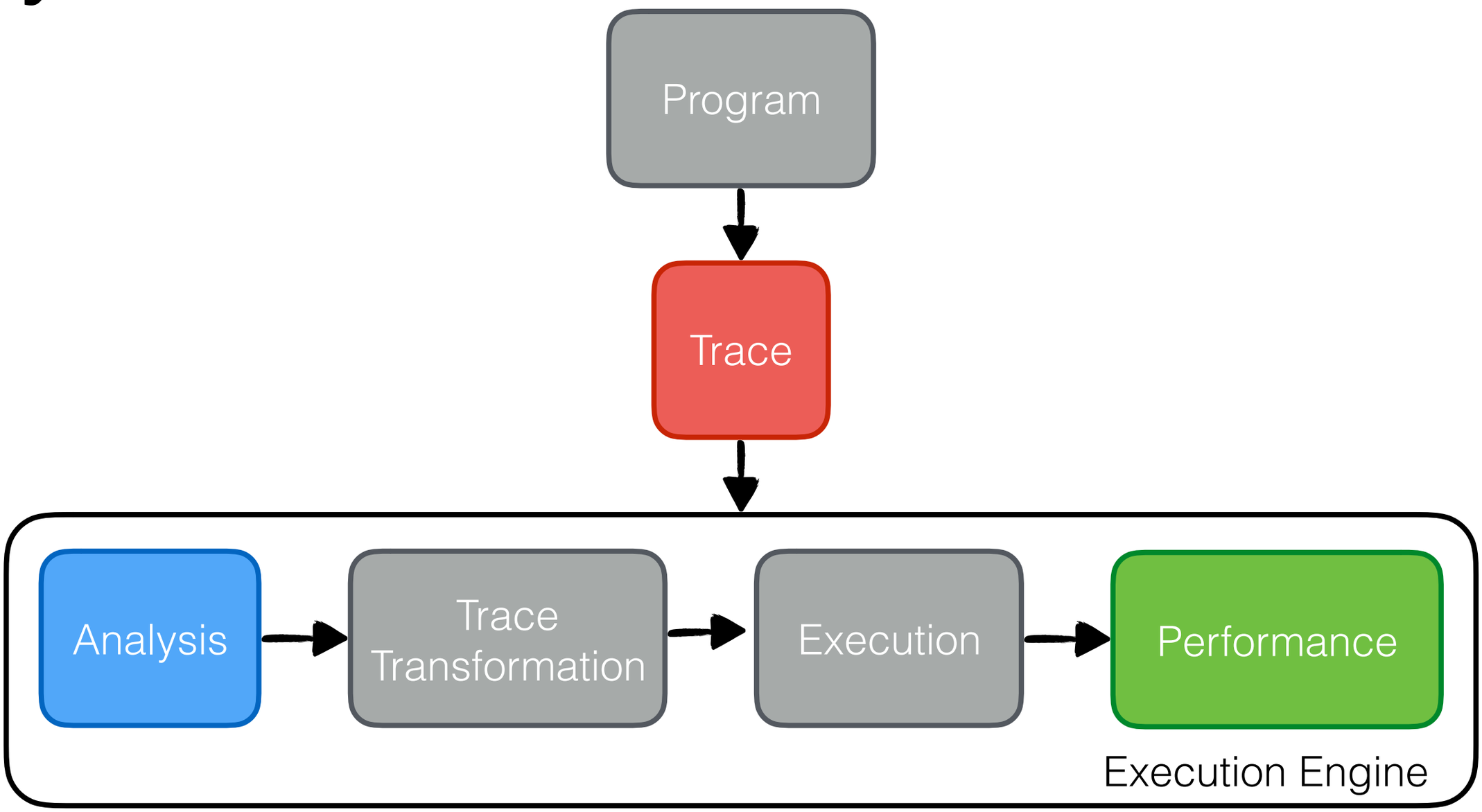
Given a **trace T** of load and store instructions

find **metrics** that characterize the trace **performance**
for a given cache C and

implement an **execution engine** for computing their quantities.

Collaboration with Alexander Miller

Overview
System



Preparation
Memory Access Trace

```
void main ()
{
   int sum, x, y, z;
   sum = 0;
   x = 1;
   y = 2;
   sum = sum + x;
   z = 3;
   sum = sum + y;
   sum = sum + z;
}
```

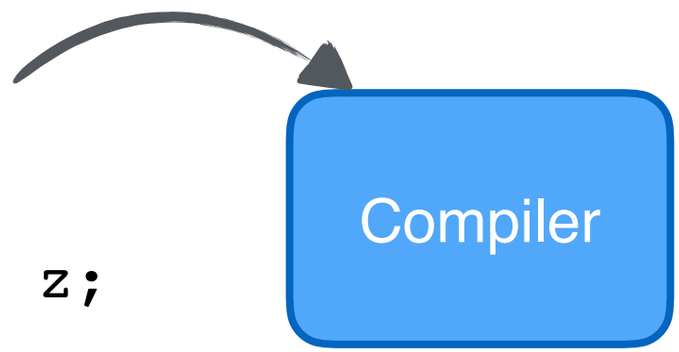
Preparation
Memory Access Trace

```
void main ()
{
   int sum, x, y, z;
   sum = 0;
   x = 1;
   y = 2;
   sum = sum + x;
   z = 3;
   sum = sum + y;
   sum = sum + z;
}
```

```
store &sum
store &x
store &y
load  &sum
load  &x
store &sum
store &z
load  &sum
load  &y
store &sum
load  &sum
load  &z
store &sum
```

Preparation
Memory Access Trace

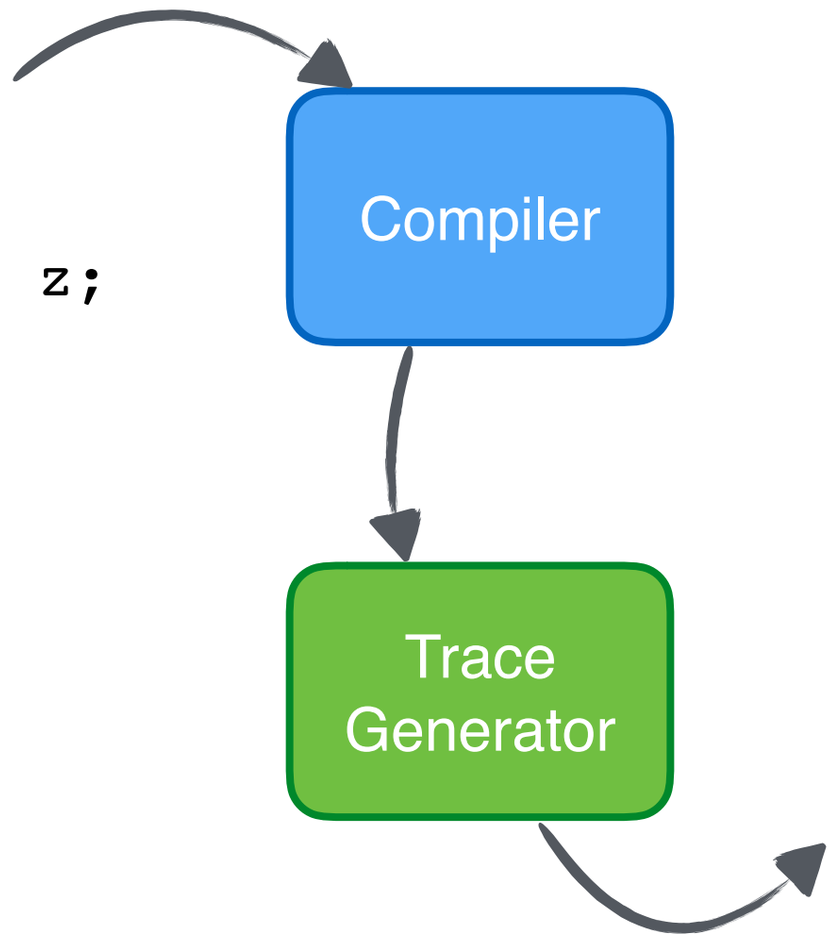
```
void main ()
{
   int sum, x, y, z;
   sum = 0;
   x = 1;
   y = 2;
   sum = sum + x;
   z = 3;
   sum = sum + y;
   sum = sum + z;
}
```




```
store &sum
store &x
store &y
load  &sum
load  &x
store &sum
store &z
load  &sum
load  &y
store &sum
load  &sum
load  &z
store &sum
```

Preparation
Memory Access Trace

```
void main ()
{
   int sum, x, y, z;
   sum = 0;
   x = 1;
   y = 2;
   sum = sum + x;
   z = 3;
   sum = sum + y;
   sum = sum + z;
}
```

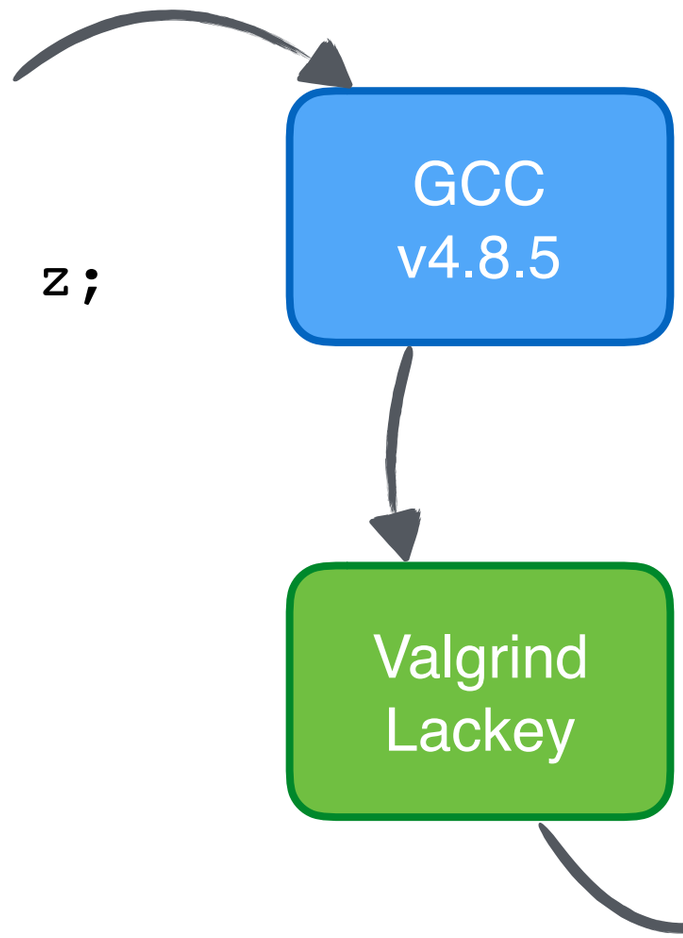



```
store &sum
store &x
store &y
load  &sum
load  &x
store &sum
store &z
load  &sum
load  &y
store &sum
load  &sum
load  &z
store &sum
```

Preparation

Memory Access Trace

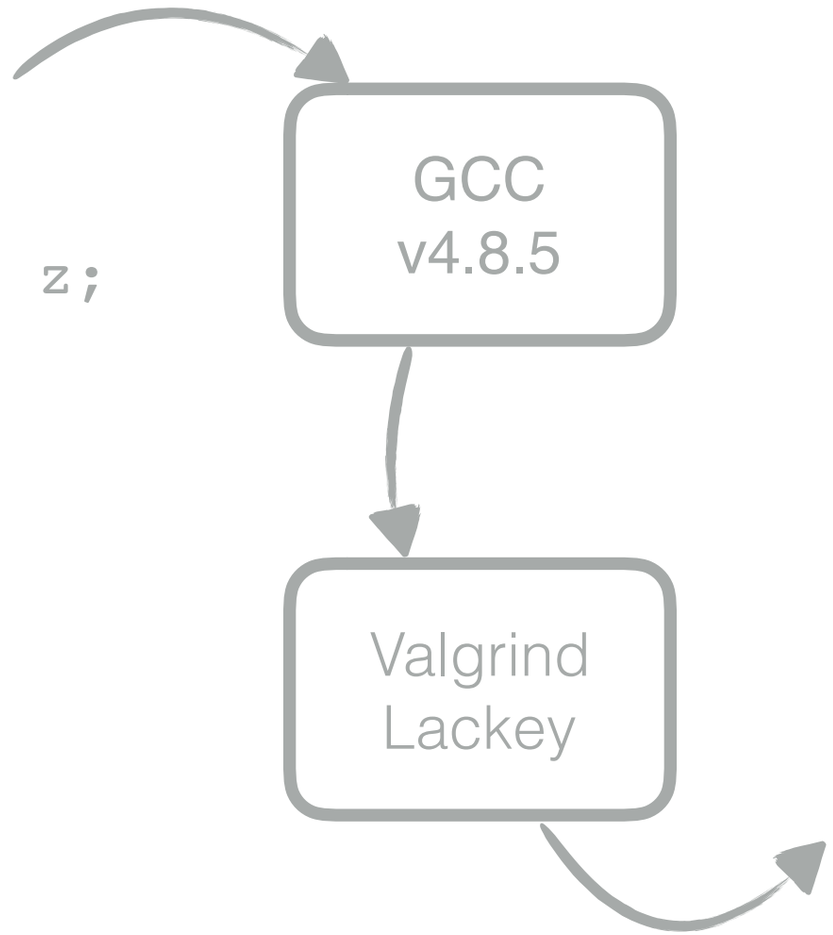
```
void main ()
{
   int sum, x, y, z;
   sum = 0;
   x = 1;
   y = 2;
   sum = sum + x;
   z = 3;
   sum = sum + y;
   sum = sum + z;
}
```




```
store &sum
store &x
store &y
load  &sum
load  &x
store &sum
store &z
load  &sum
load  &y
store &sum
load  &sum
load  &z
store &sum
```

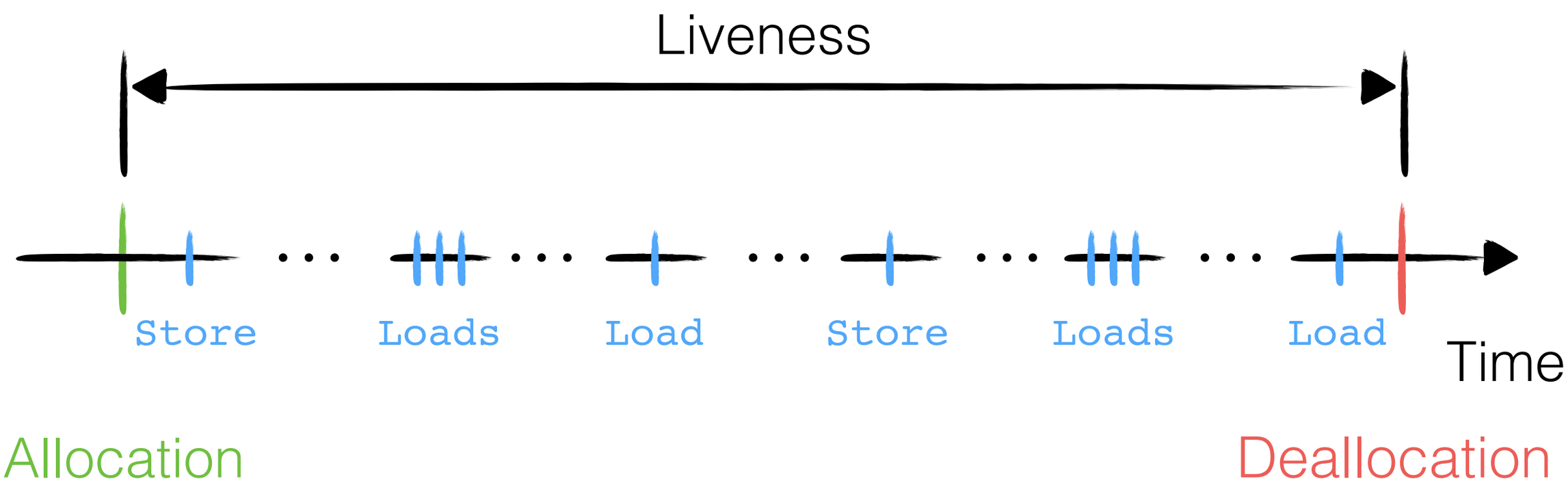
Preparation
Memory Access Trace

```
void main ()
{
   int sum, x, y, z;
   sum = 0;
   x = 1;
   y = 2;
   sum = sum + x;
   z = 3;
   sum = sum + y;
   sum = sum + z;
}
```

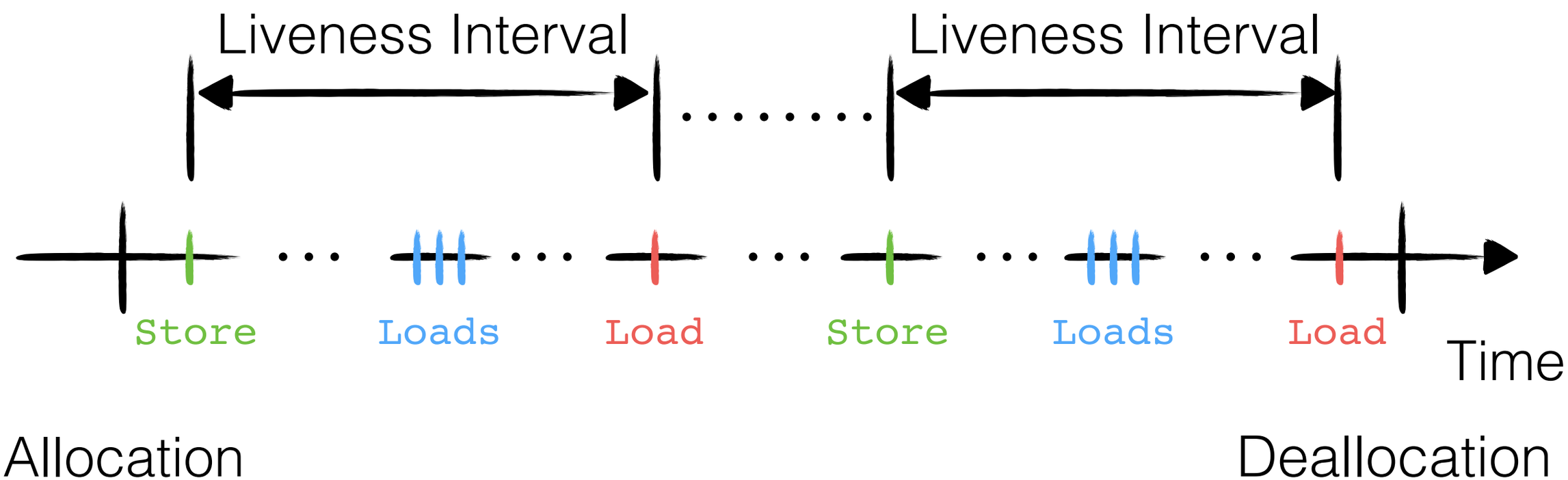



```
store &sum
store &x
store &y
load  &sum
load  &x
store &sum
store &z
load  &sum
load  &y
store &sum
load  &sum
load  &z
store &sum
```

Analysis
Liveness

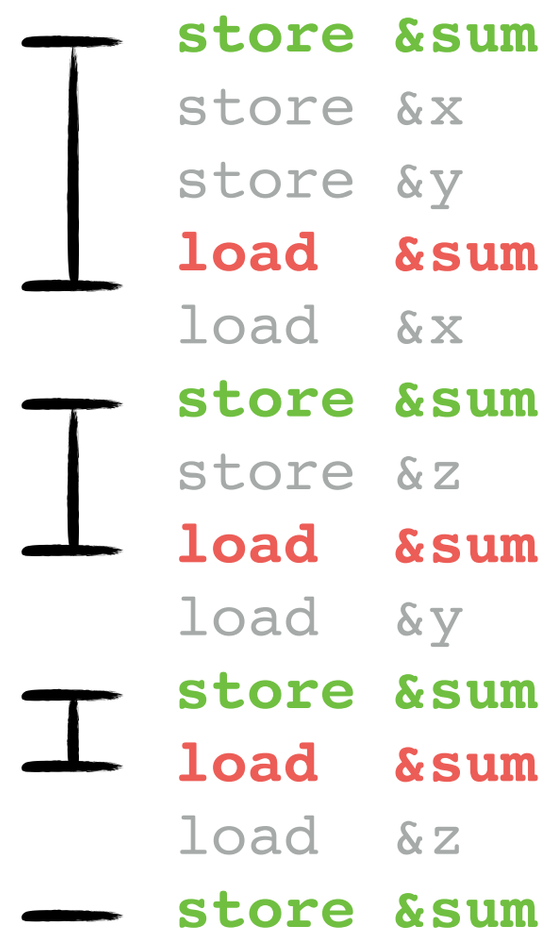


Analysis
Liveness Intervals (LI)

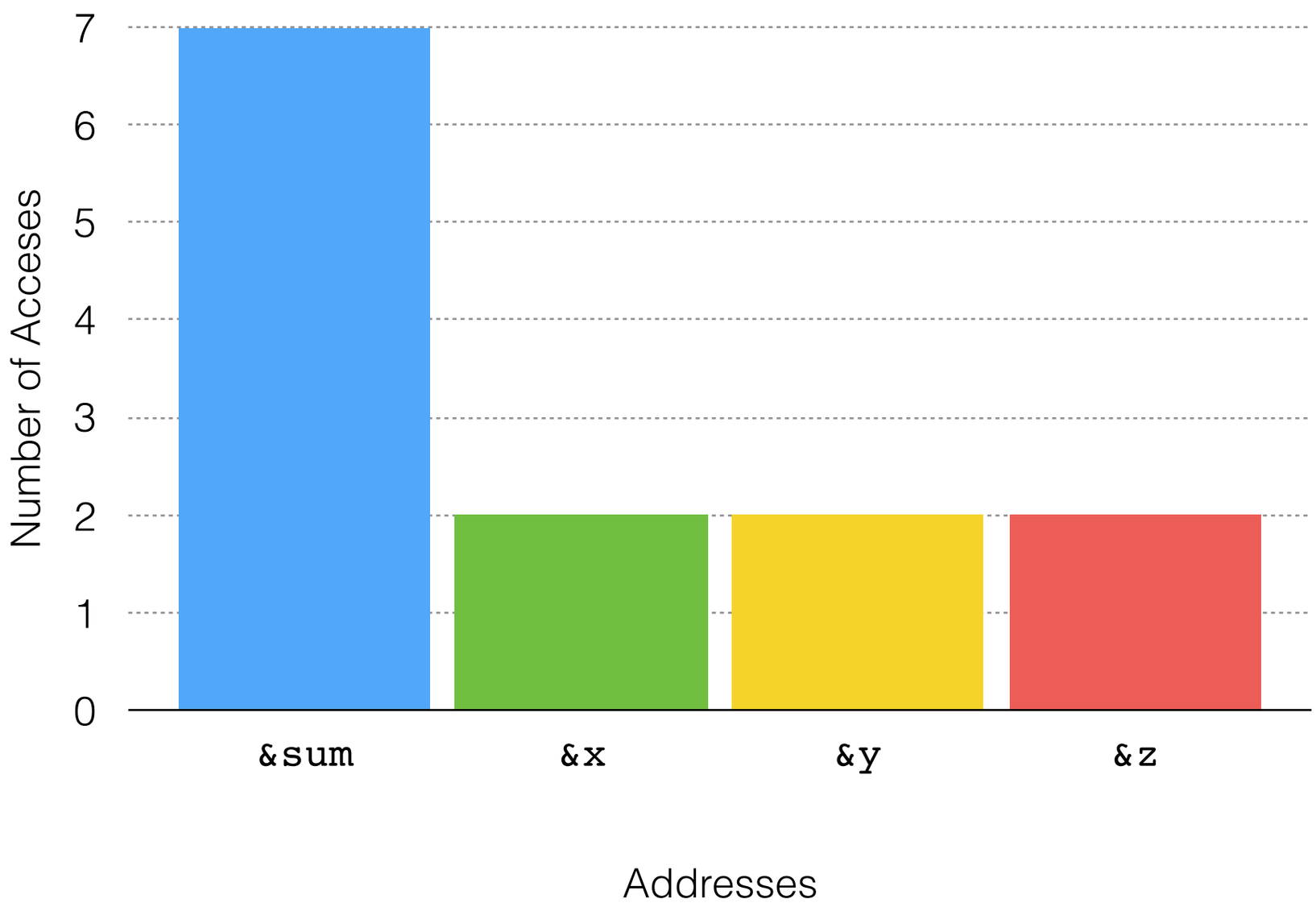


Analysis

Liveness Intervals (LI)



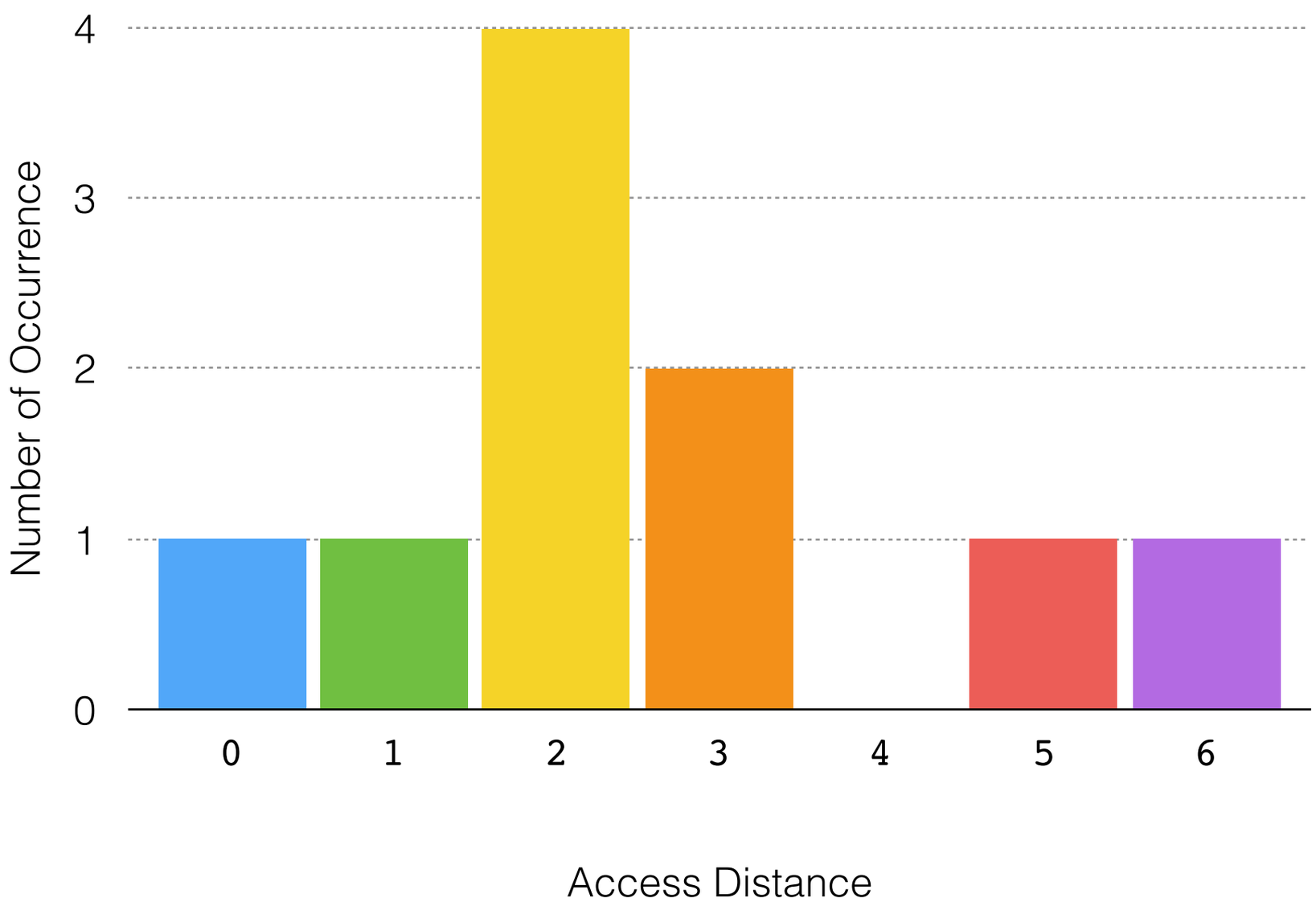
Analysis
Metric: Accesses




```
 1: store &sum
 2: store &x
 3: store &y
 4: load  &sum
 5: load  &x
 6: store &sum
 7: store &z
 8: load  &sum
 9: load  &y
10: store &sum
11: load  &sum
12: load  &z
13: store &sum
```

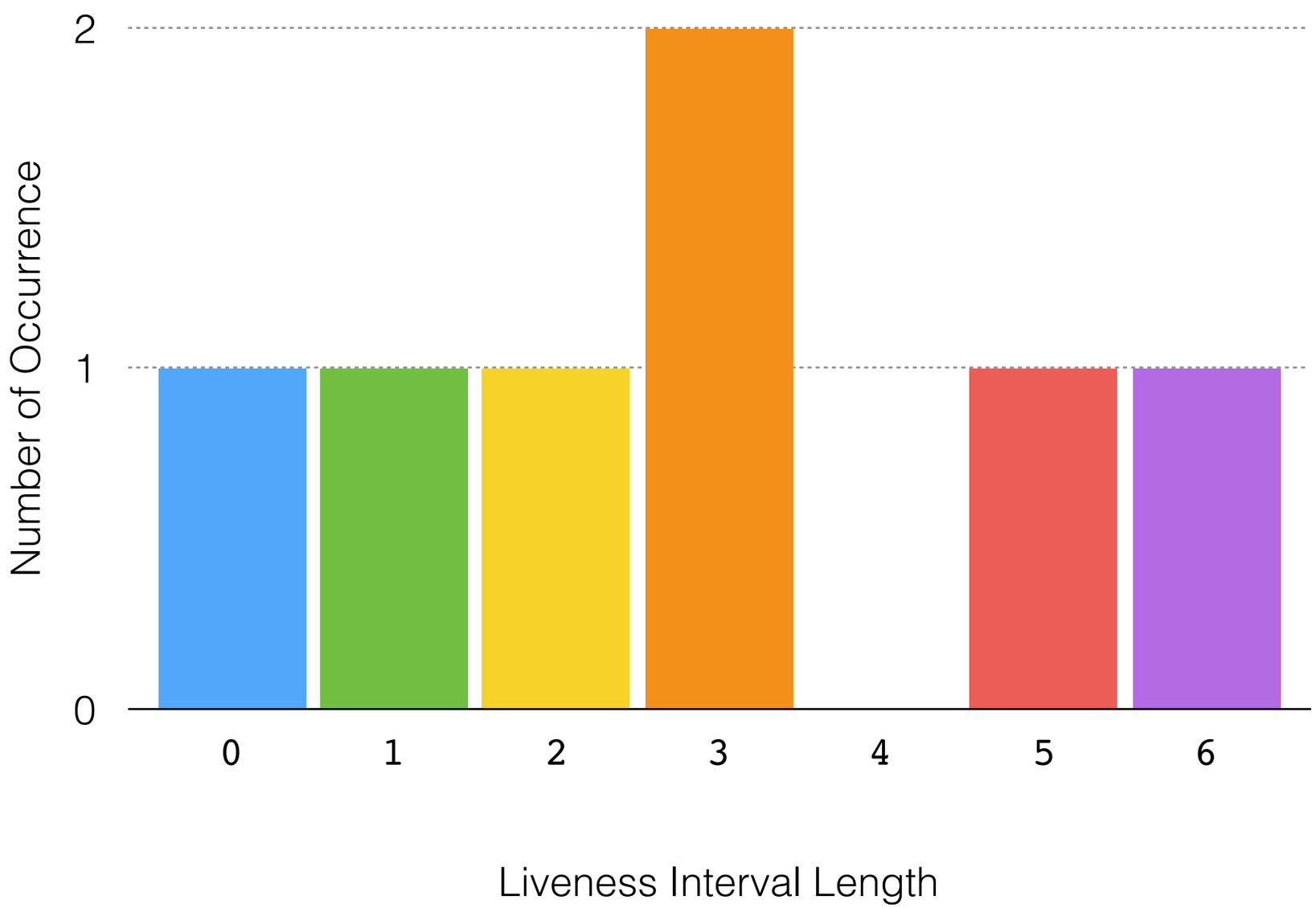
Analysis

Metric: Access Distance




```
1:  store  &sum
2:  store  &x
3:  store  &y
4:  load   &sum
5:  load   &x
6:  store  &sum
7:  store  &z
8:  load   &sum
9:  load   &y
10: store  &sum
11: load   &sum
12: load   &z
13: store  &sum
```

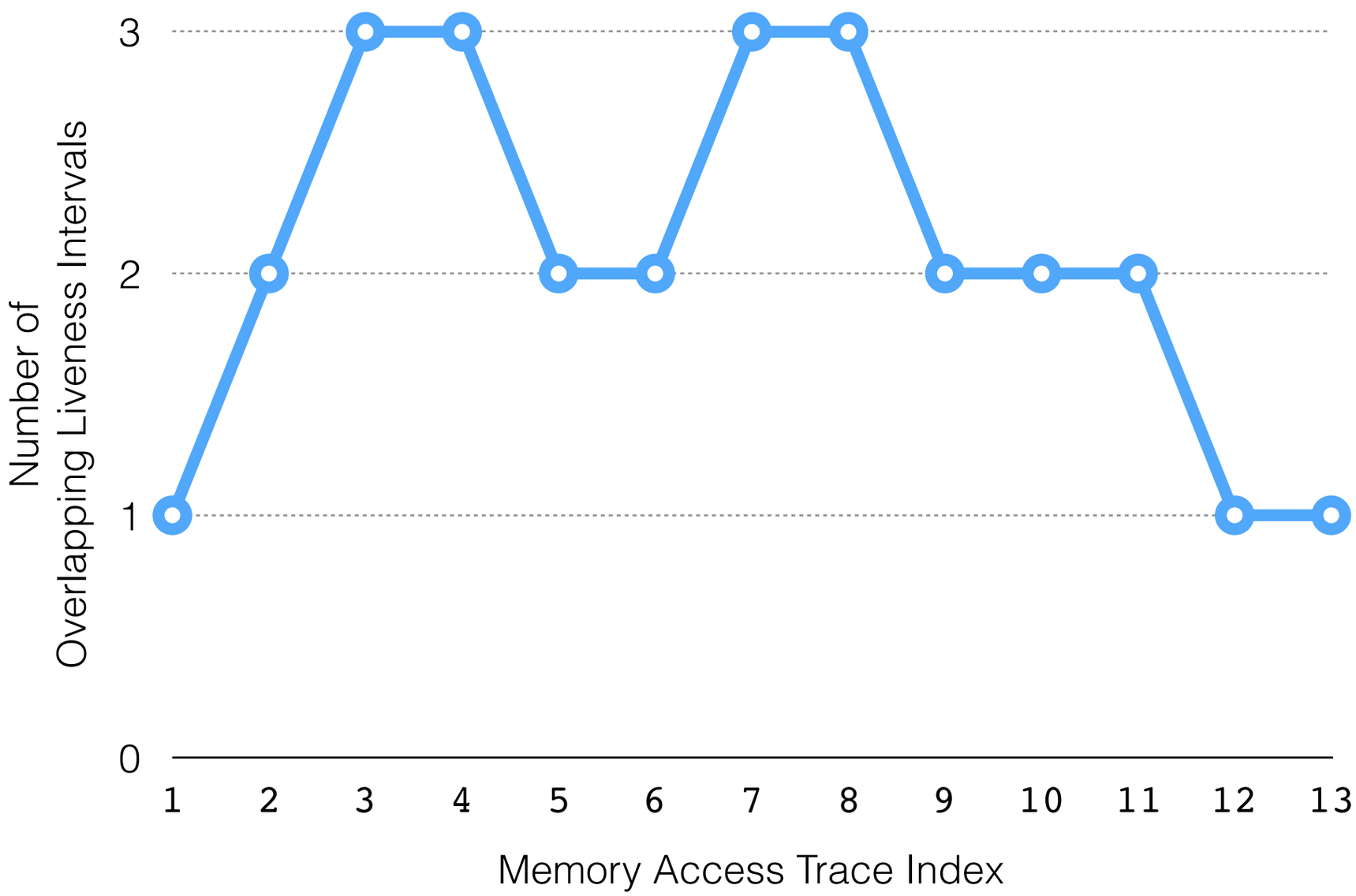
Analysis
Metric: LI Length




```
 1: store &sum
 2: store &x
 3: store &y
 4: load  &sum
 5: load  &x
 6: store &sum
 7: store &z
 8: load  &sum
 9: load  &y
10: store &sum
11: load  &sum
12: load  &z
13: store &sum
```

Analysis

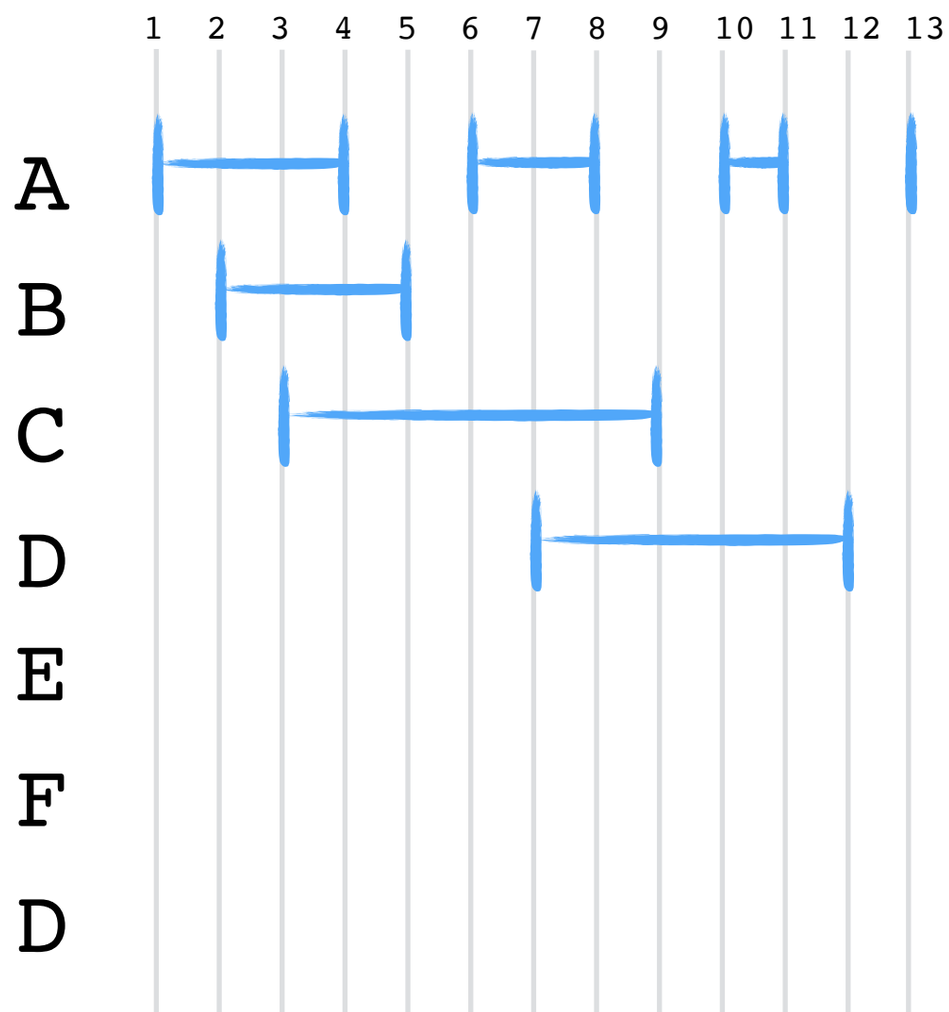
Metric: Overlapping LI




```
1:  store &sum
2:  store &x
3:  store &y
4:  load  &sum
5:  load  &x
6:  store &sum
7:  store &z
8:  load  &sum
9:  load  &y
10: store &sum
11: load  &sum
12: load  &z
13: store &sum
```

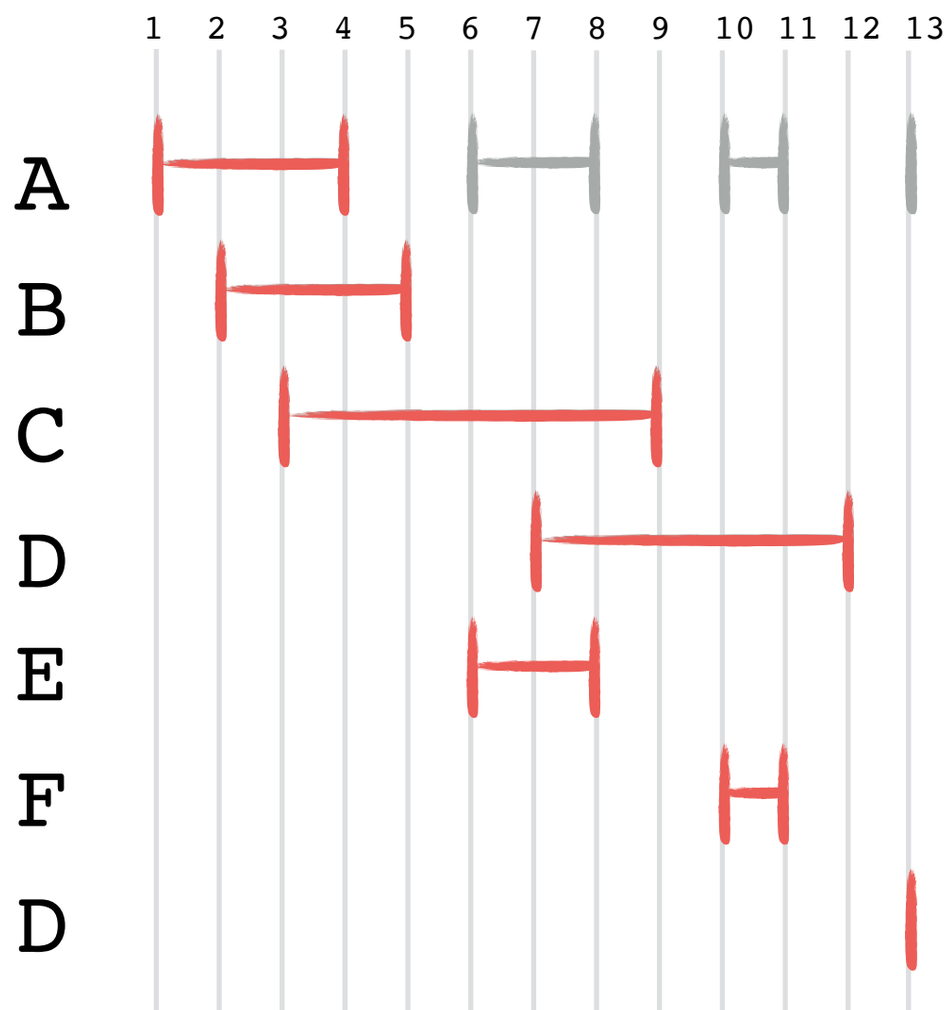
Trace Transformation

Identity



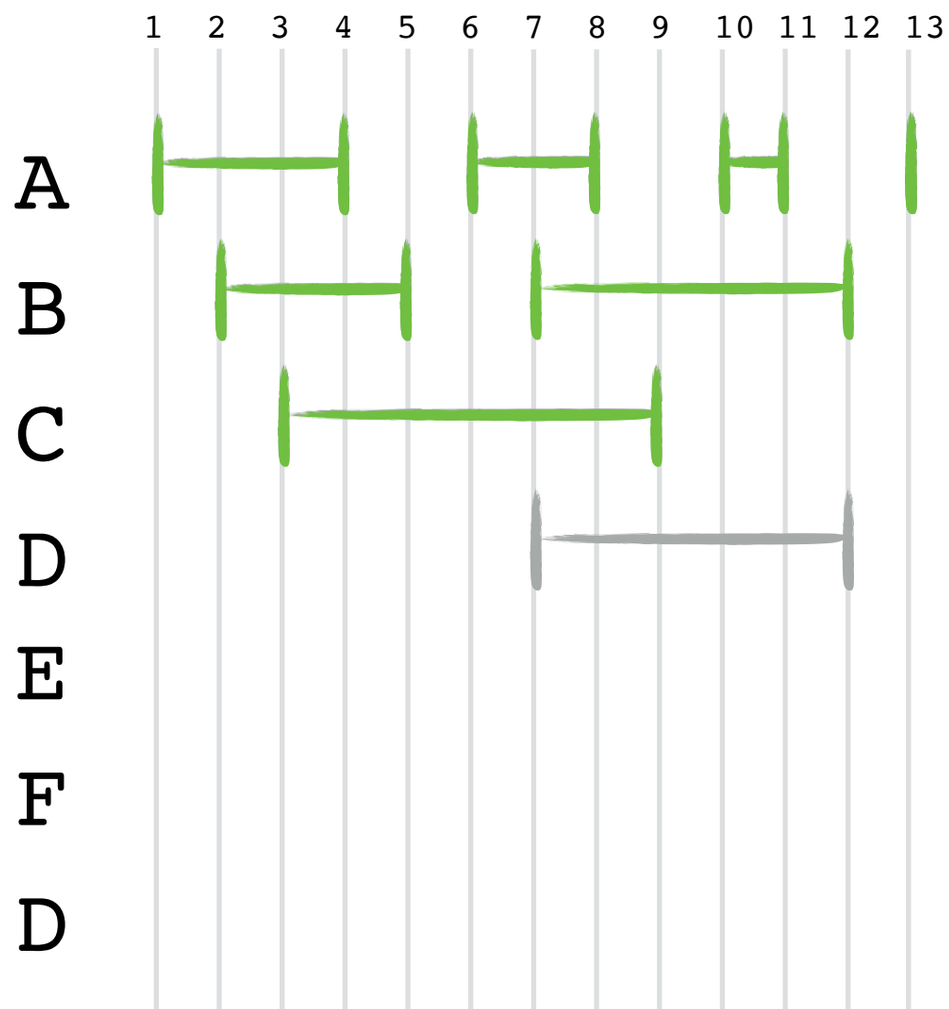
Trace Transformation

Single Assignment

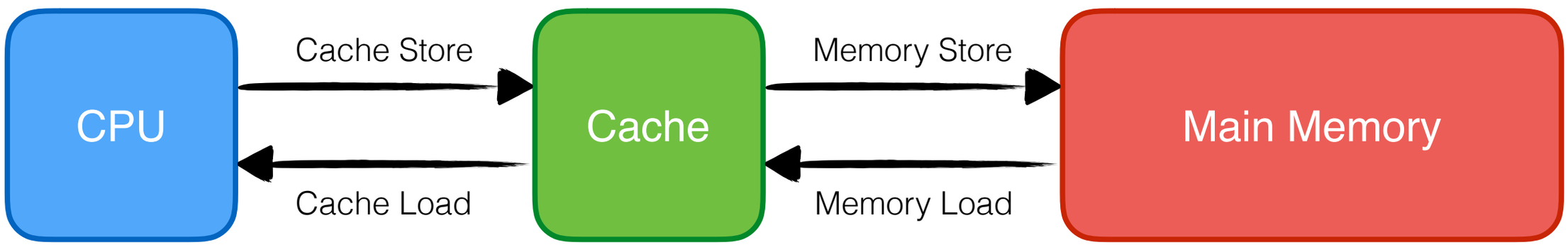


Trace Transformation

Compact



Execution
Hardware Model



Execution
Performance

$$\texttt{CPA(T,C)} = \frac{\texttt{Sum of cycles(T,C)}}{\texttt{Sum of accesses(T)}}$$

Key

Trace … `T`

Cache … `C`

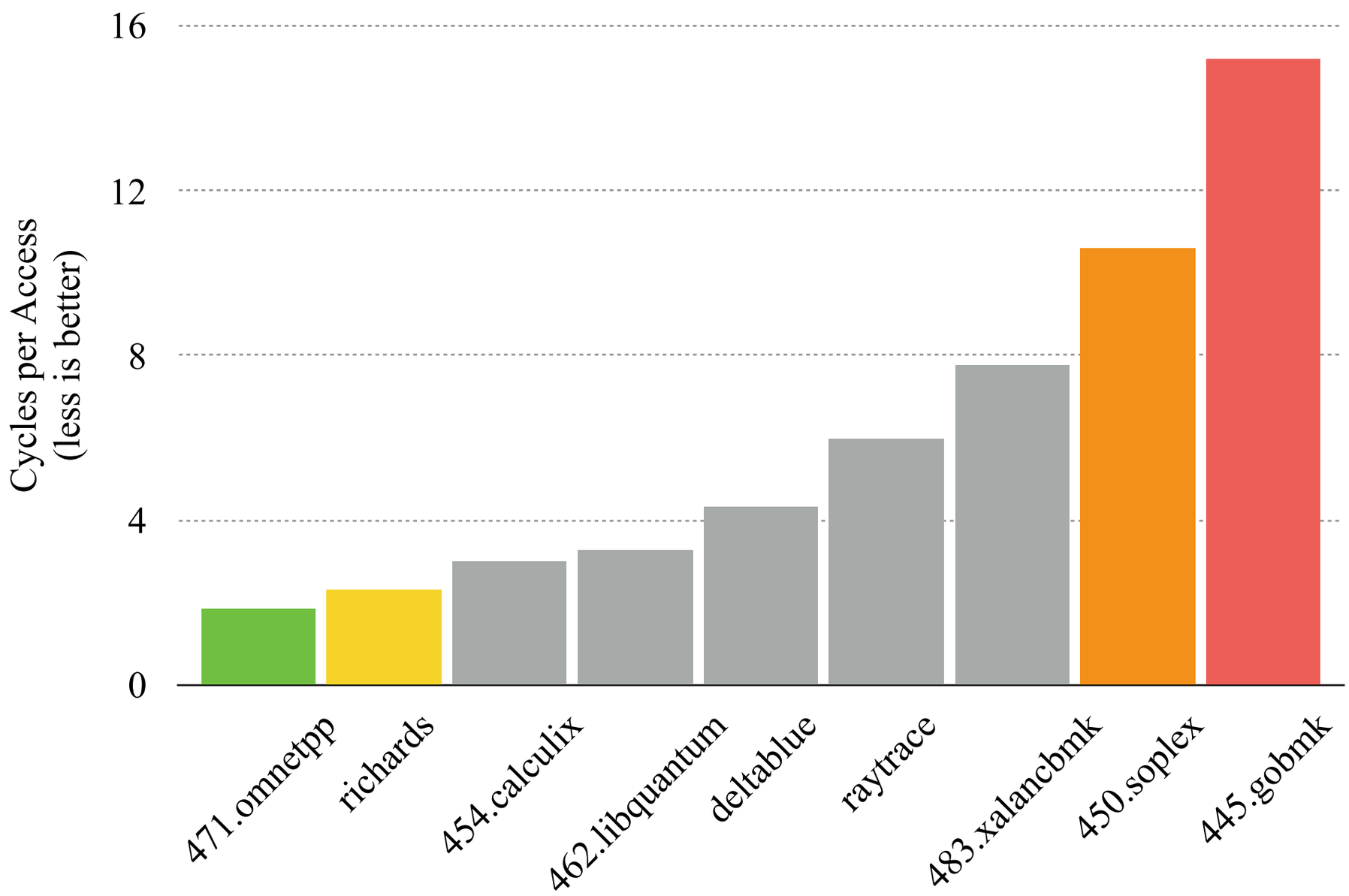
Cycles per Access … `CPA`

Costs in Cycles

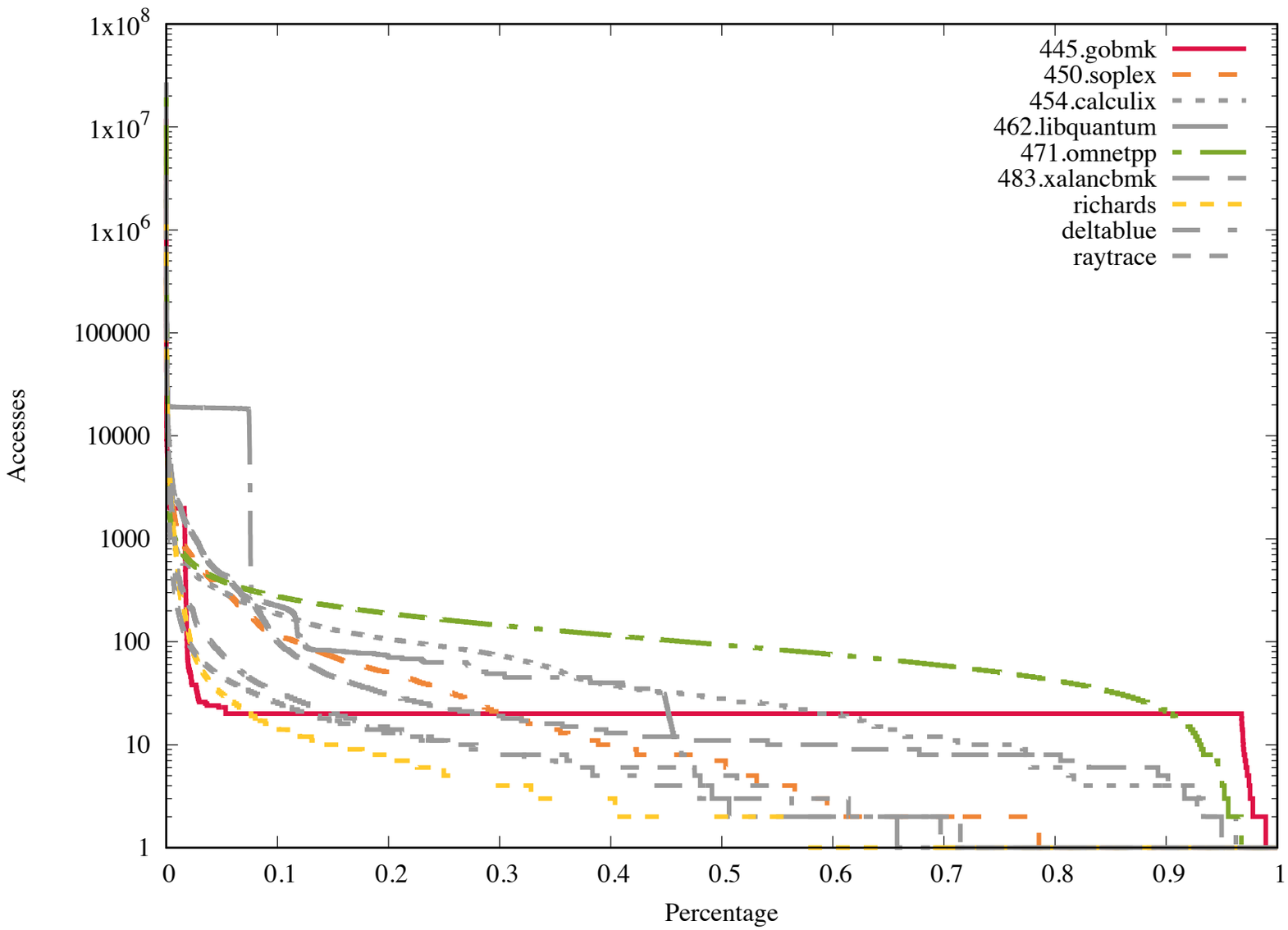
Cache Load / Store: **1**

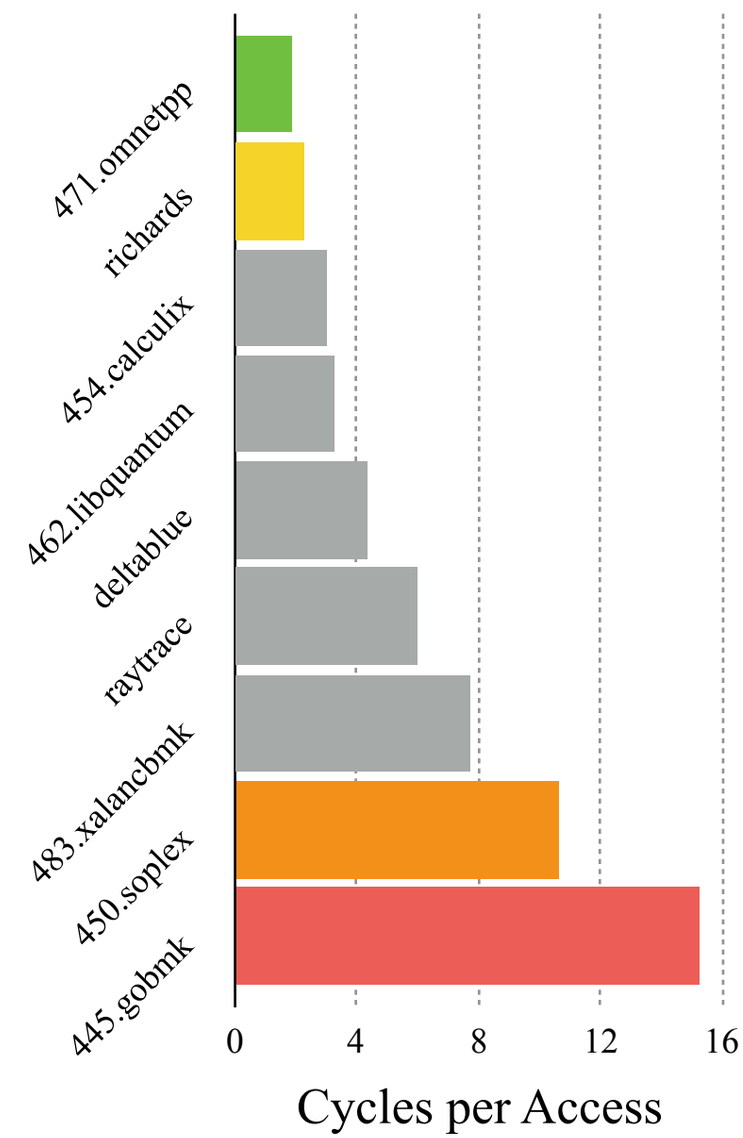
Memory Load / Store: **5**

Experiment
Performance



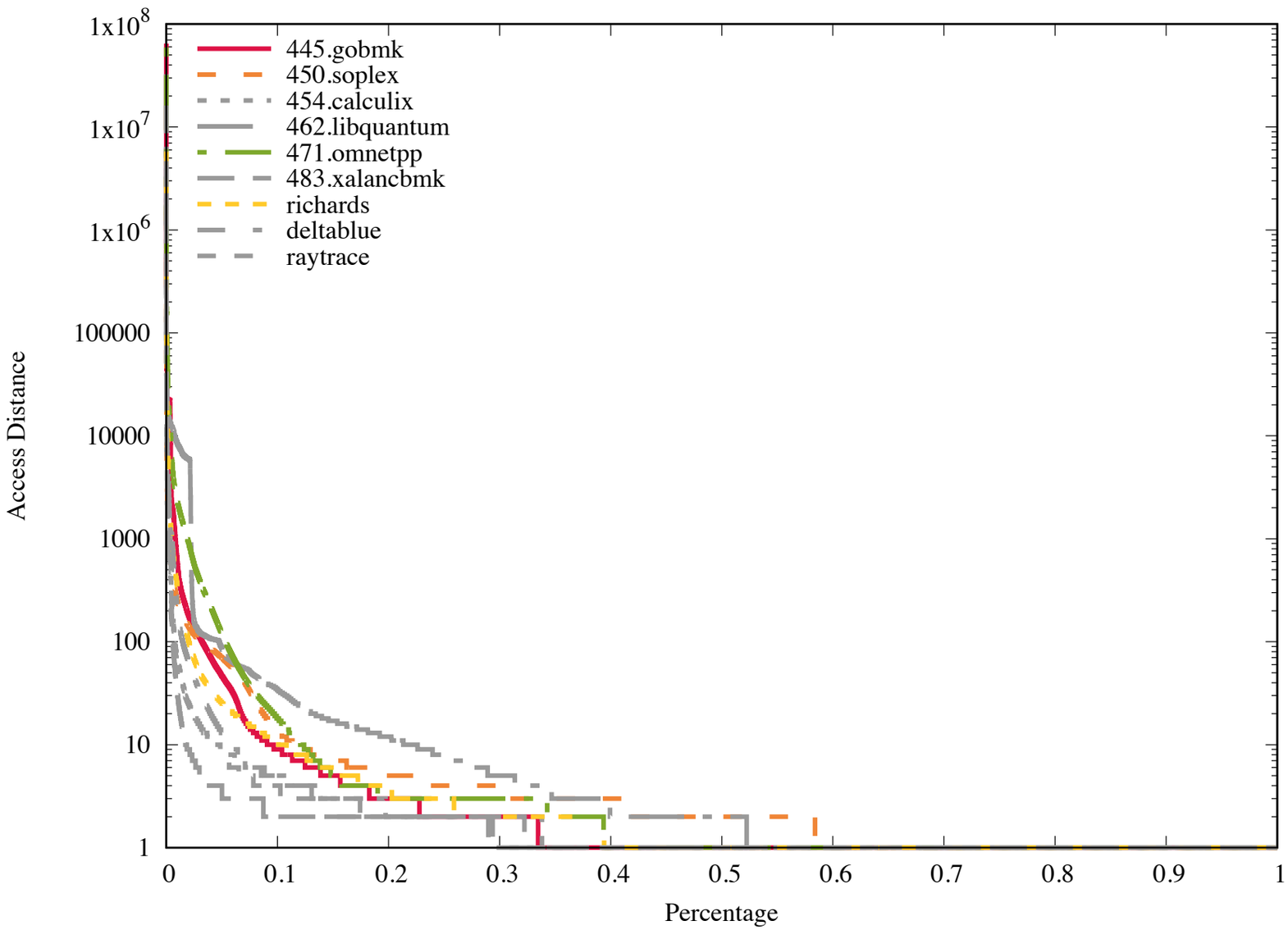
Experiment
Metric: Accesses

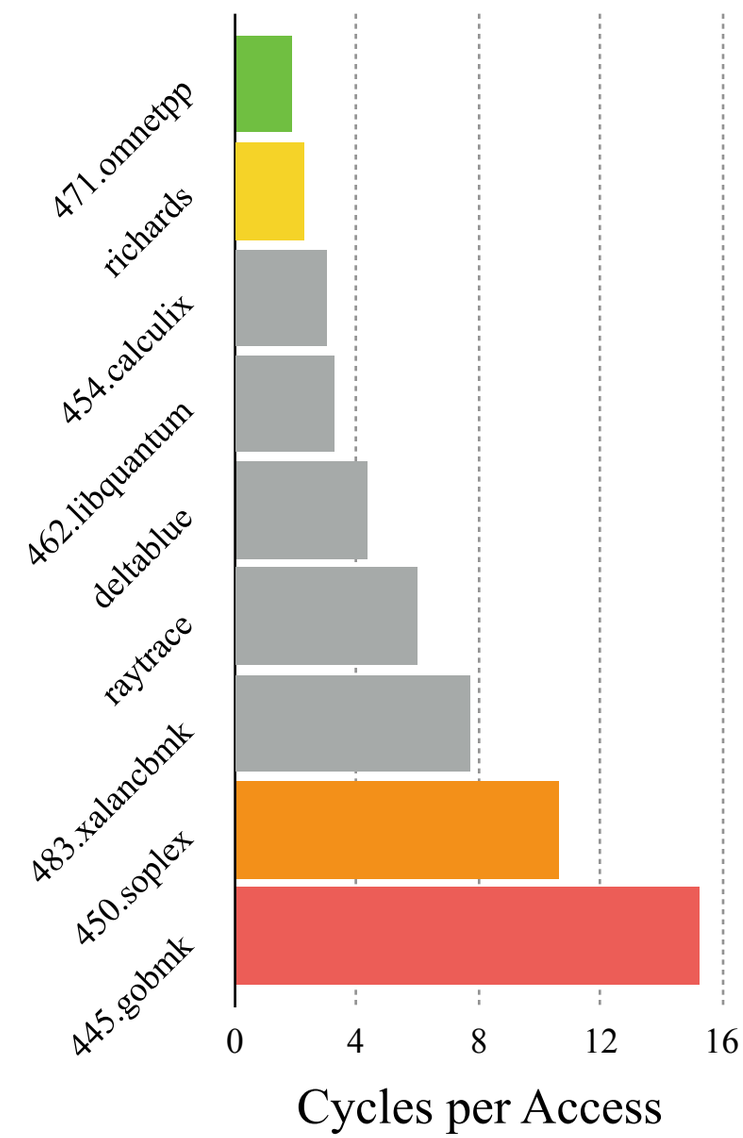




Experiment

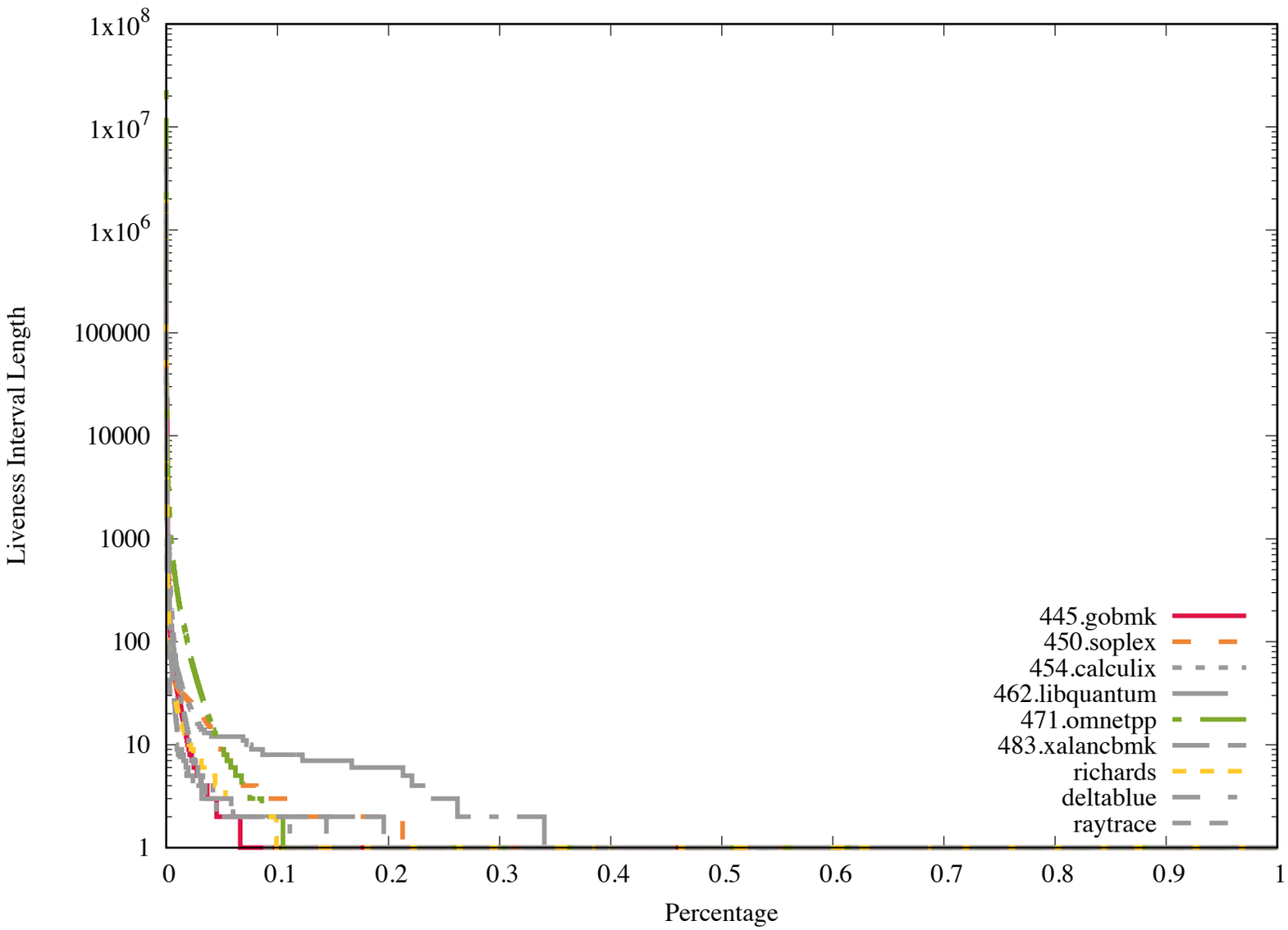
Metric: Access Distance

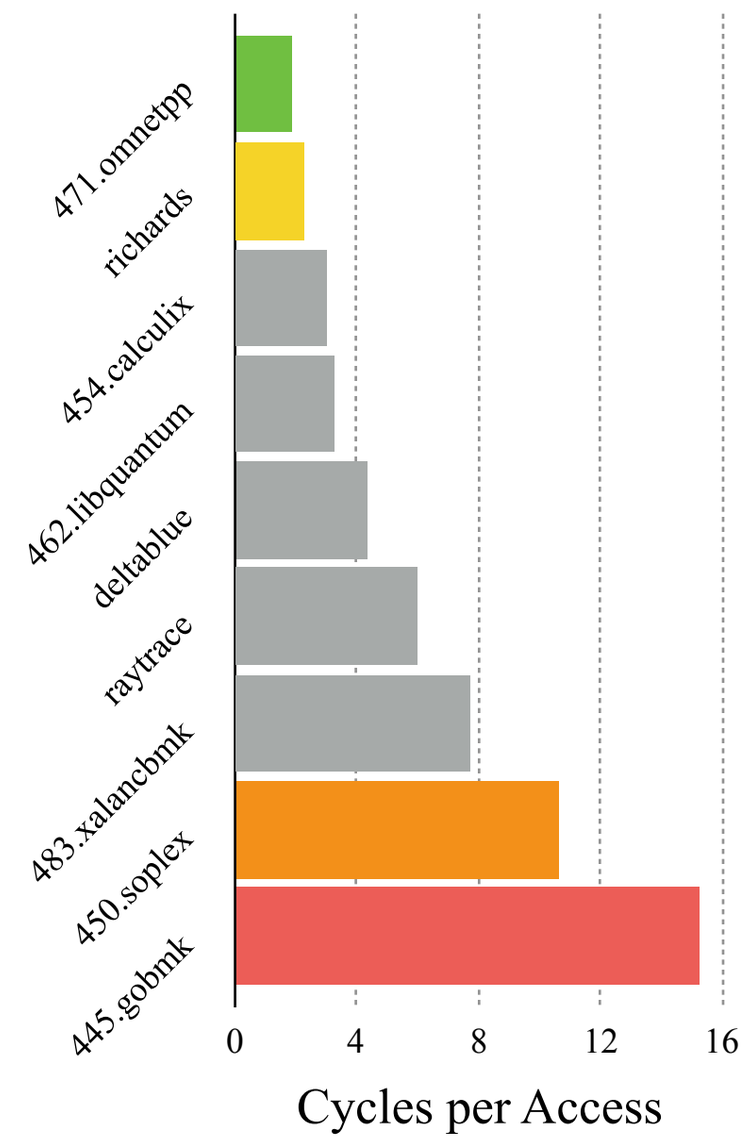




Experiment

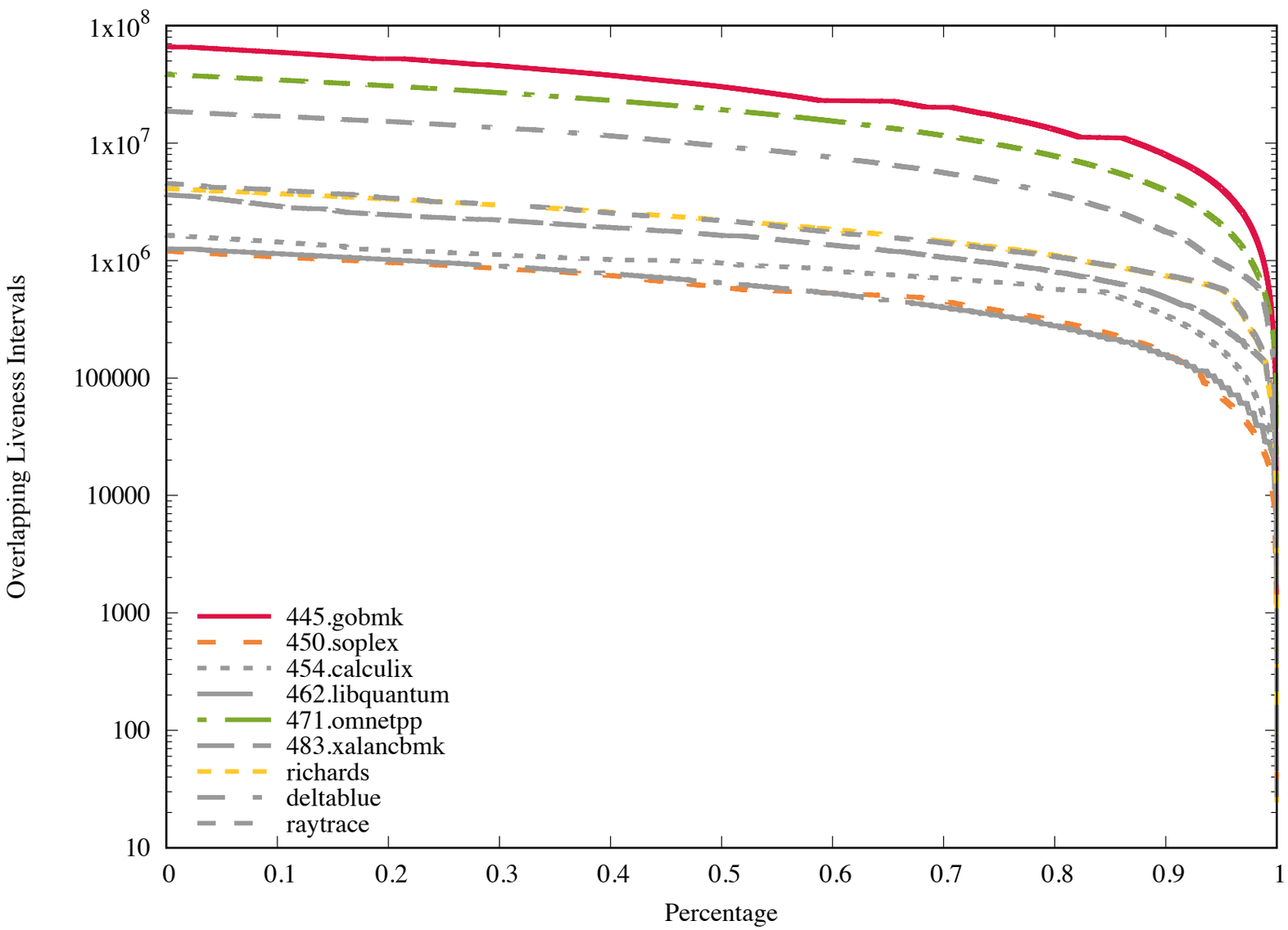
Metric: LI Length

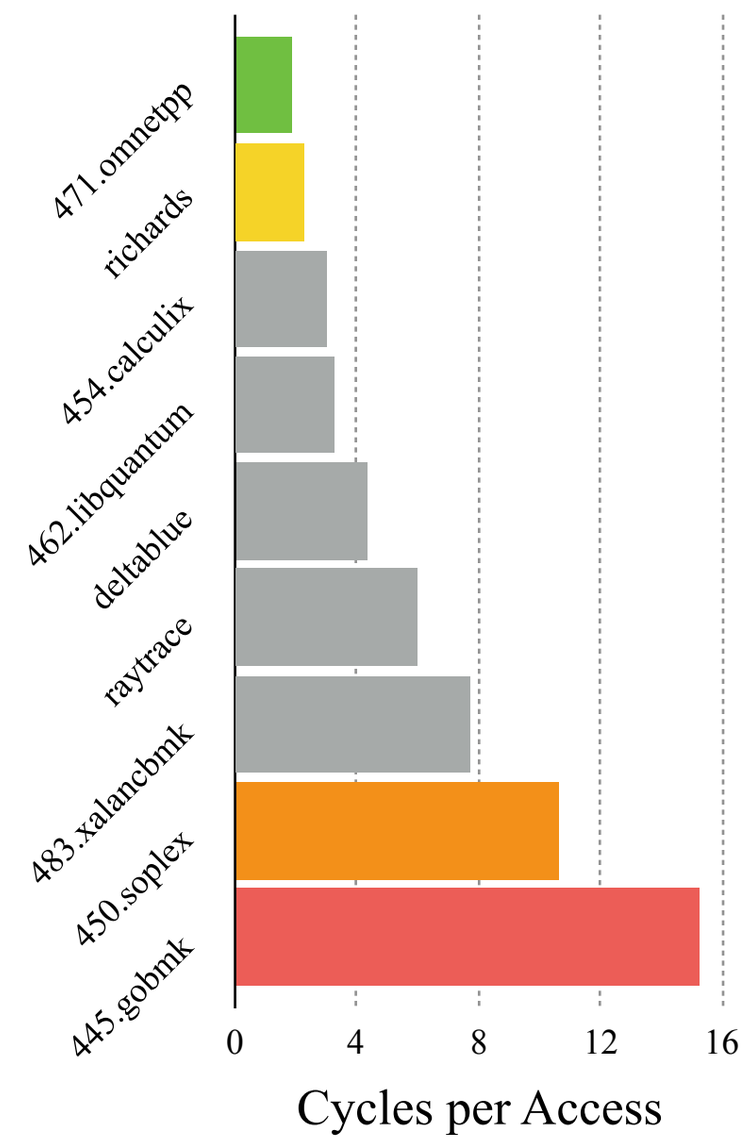




Experiment

Metric: Overlapping LI





Conclusion
Metrics

445.gobmk

- Few accesses on the majority of address
- Nearly constant number of accesses
- Many overlapping liveness intervals

471.omnetpp

- Many accesses per address
- Few addresses with significantly more accesses
- Many overlapping liveness intervals

Conclusion

4 metrics

- accesses
- access distance
- liveness interval length
- overlapping liveness

9 benchmarks from 2 benchmark suits

- SPEC 2006
- V8

The defined metrics illustrate tendencies for improvement rather than unique characteristics.

Thank you.

Q&A