How FIFO is Your Concurrent FIFO Queue?

Andreas Haas, Christoph M. Kirsch,
Michael Lippautz, Hannes Payer

University of Salzburg

RACES Workshop, October 2012

Strict vs. Relaxed FIFO Queues

strict FIFO queue implementations

relaxed FIFO queue implementations

Strict vs. Relaxed FIFO Queues

strict FIFO queue implementations

linearizable with respect to
strict FIFO queue semantics

relaxed FIFO queue implementations

Strict vs. Relaxed FIFO Queues

strict FIFO queue implementations

linearizable with respect to
strict FIFO queue semantics

relaxed FIFO queue implementations

linearizable with respect to
relaxed FIFO queue semantics

Strict vs. Relaxed FIFO Queues

strict FIFO queue implementations

linearizable with respect to strict FIFO queue semantics

relaxed FIFO queue implementations

linearizable with respect to relaxed FIFO queue semantics

bounded out-of-order treatment of queue elements possible

Strict vs. Relaxed FIFO Queues

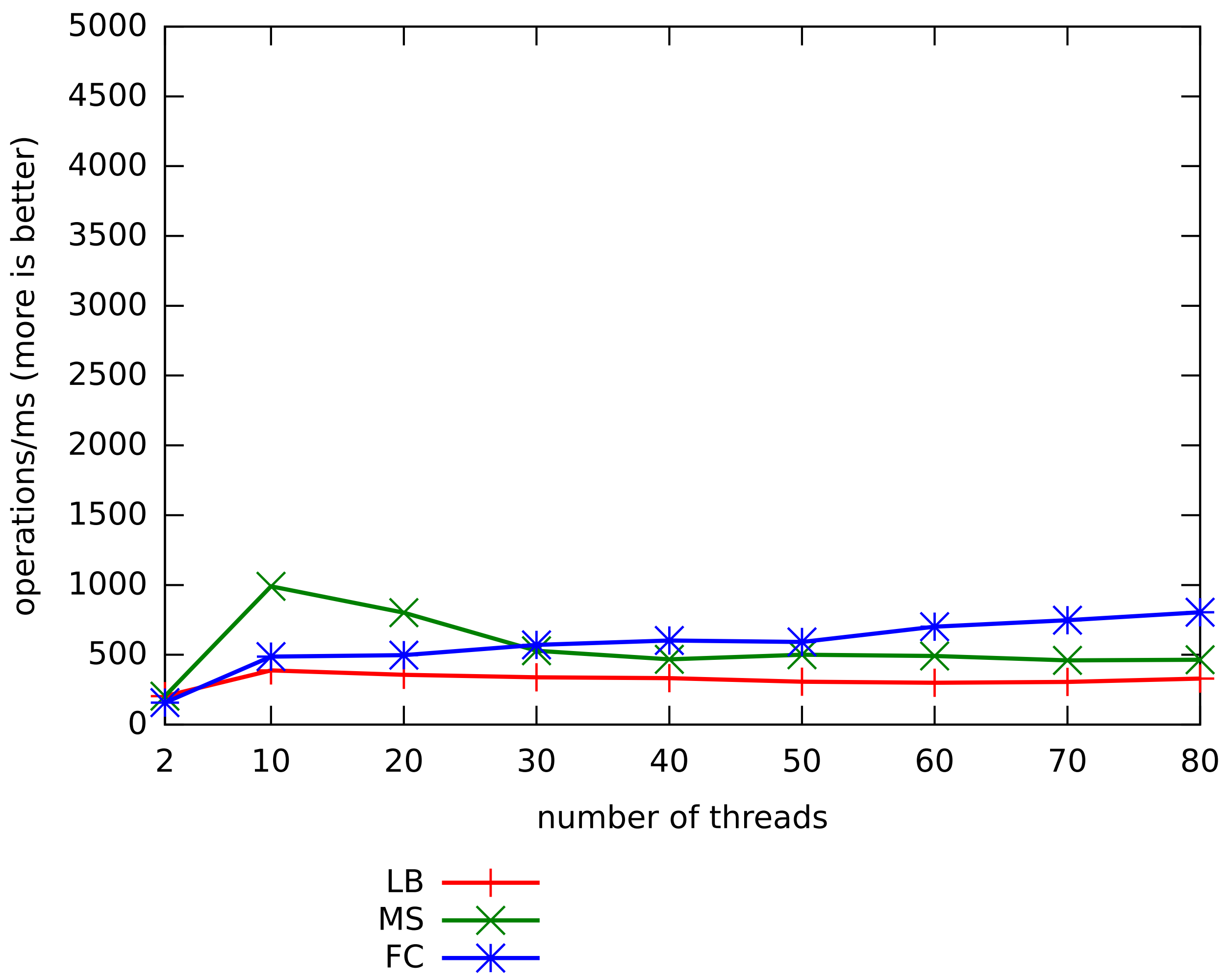
strict FIFO queue implementations

linearizable with respect to strict FIFO queue semantics

relaxed FIFO queue implementations

linearizable with respect to relaxed FIFO queue semantics

bounded out-of-order treatment of queue elements possible



Strict vs. Relaxed FIFO Queues

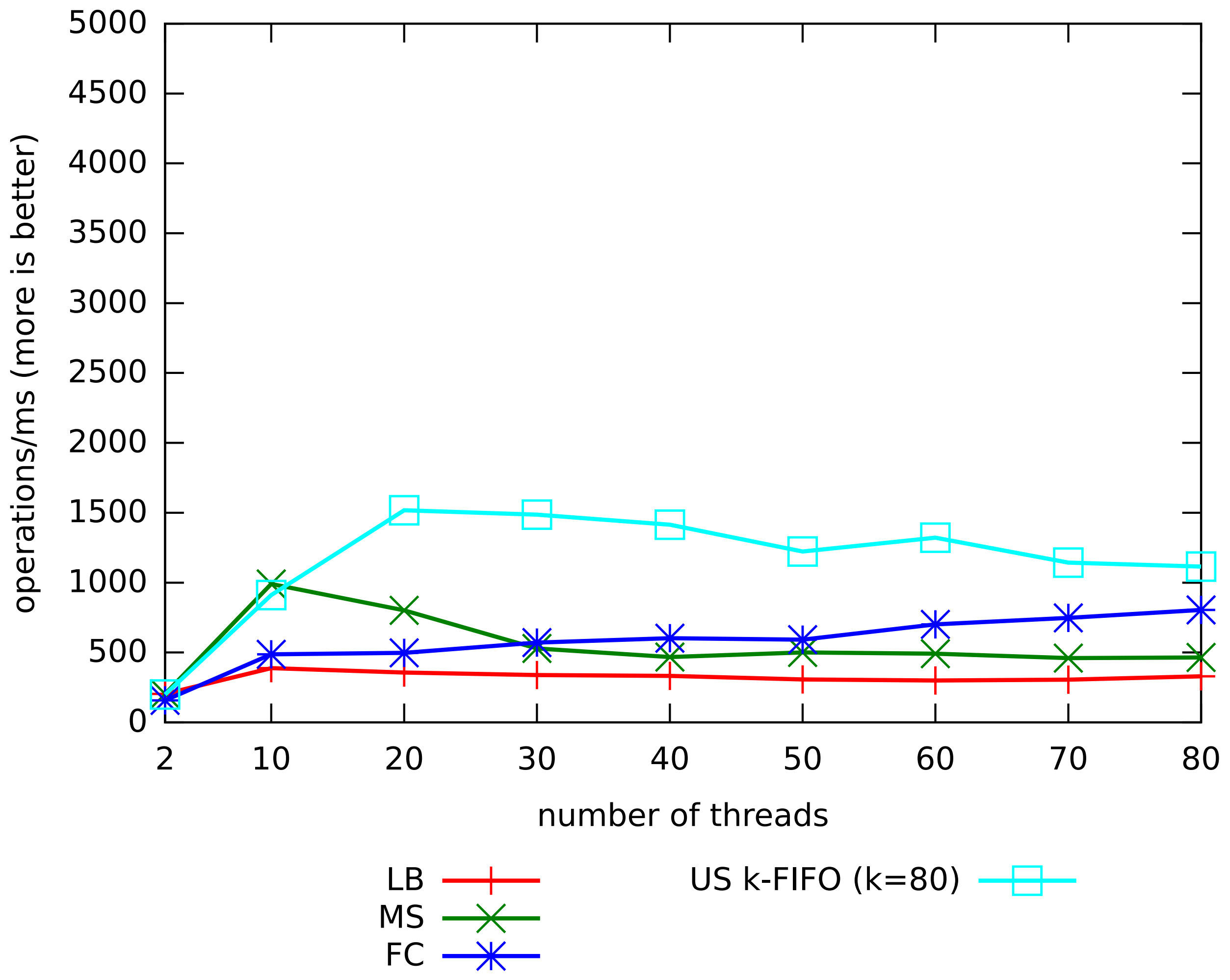
strict FIFO queue implementations

linearizable with respect to
strict FIFO queue semantics

relaxed FIFO queue implementations

linearizable with respect to
relaxed FIFO queue semantics

bounded
out-of-order treatment
of queue elements
possible



Strict vs. Relaxed FIFO Queues

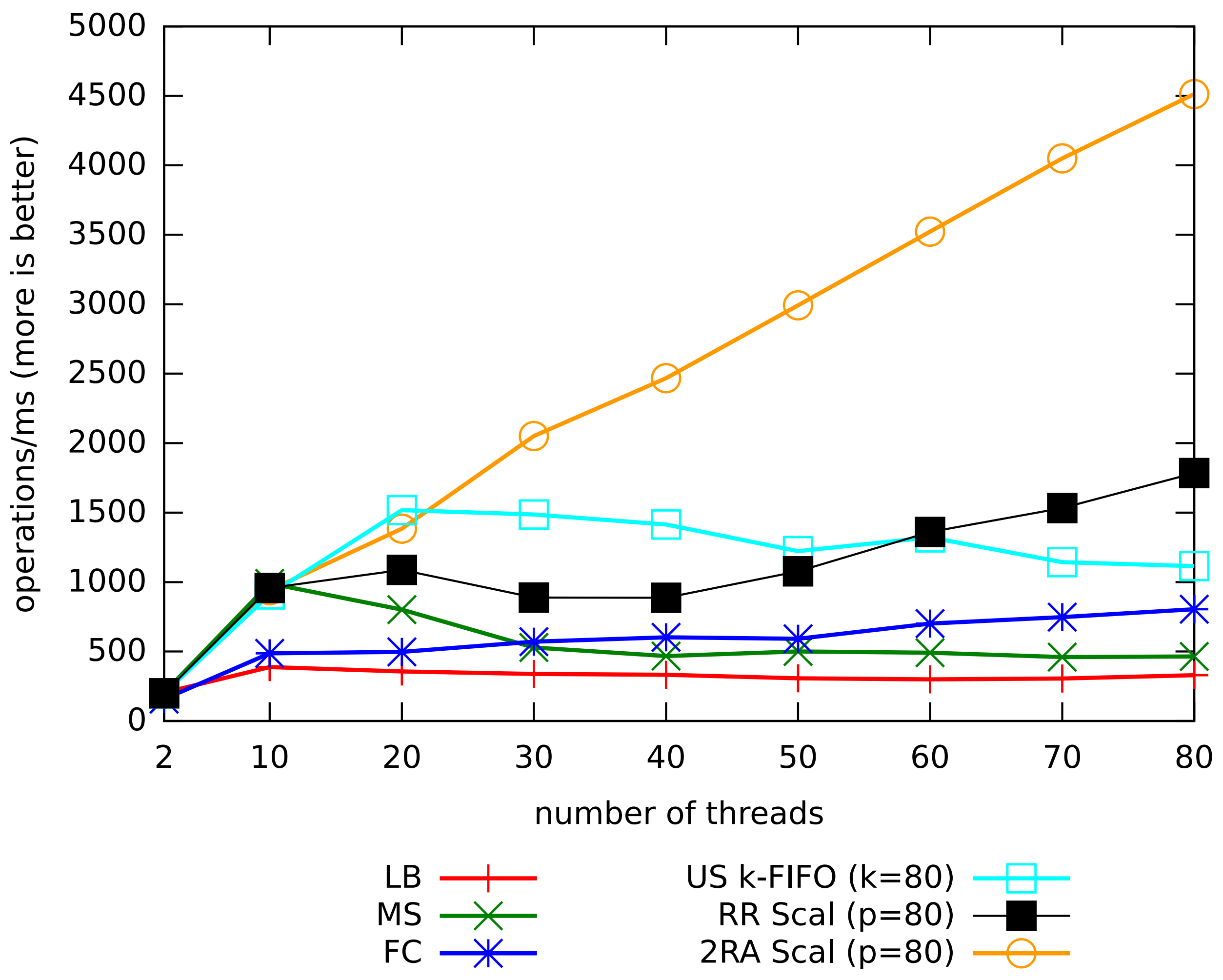
strict FIFO queue implementations

linearizable with respect to strict FIFO queue semantics

relaxed FIFO queue implementations

linearizable with respect to relaxed FIFO queue semantics

bounded out-of-order treatment of queue elements possible



How FIFO are Relaxed FIFO Queues?

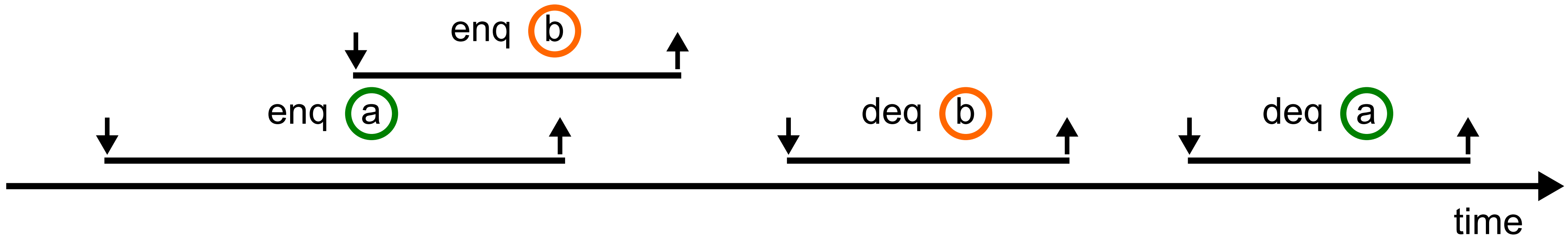
- Some people say relaxed FIFO queues are not enough FIFO.
  - No applications for relaxed FIFO queues.

How FIFO are Relaxed FIFO Queues?

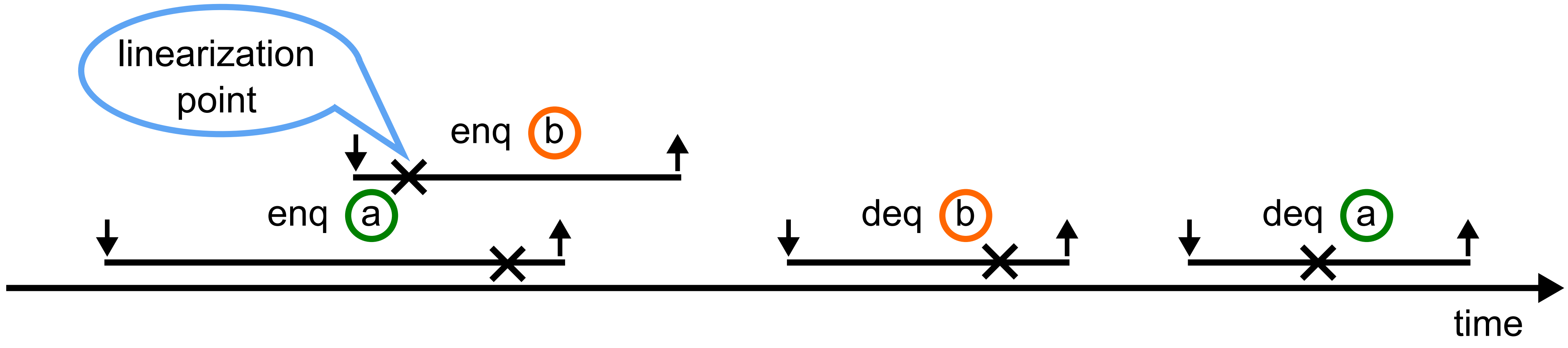
- Some people say relaxed FIFO queues are not enough FIFO.
  - No applications for relaxed FIFO queues.

We say relaxed FIFO queue implementations can be even more FIFO than strict FIFO queue implementations.

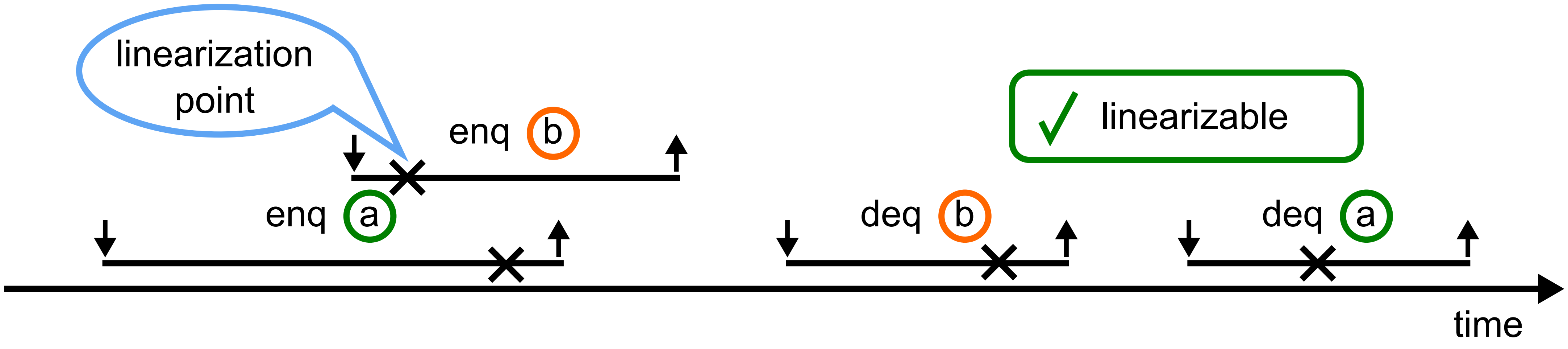
Example



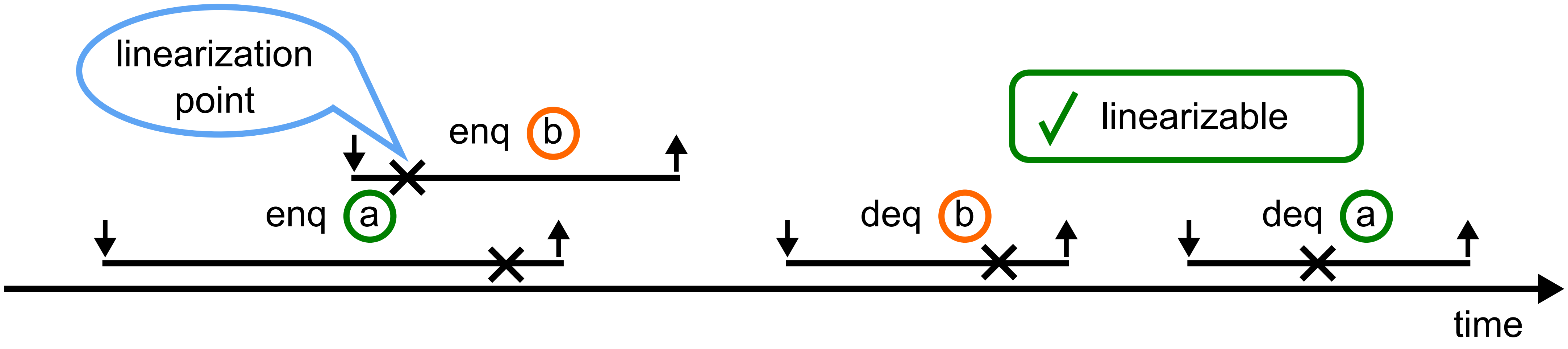
Example

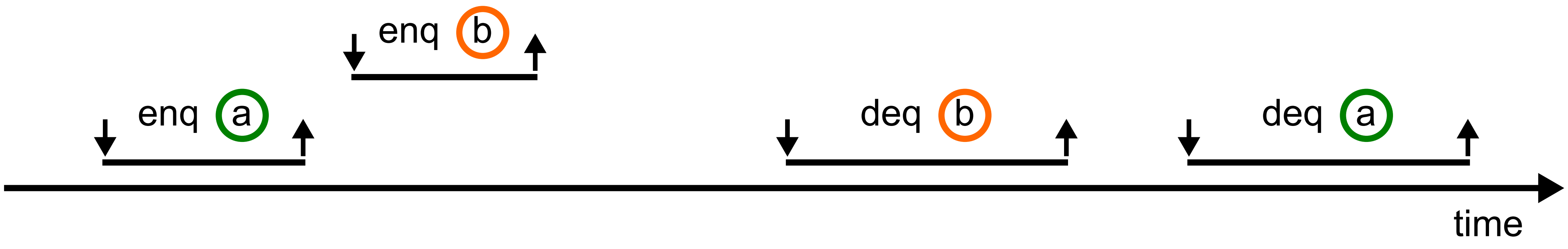


Example

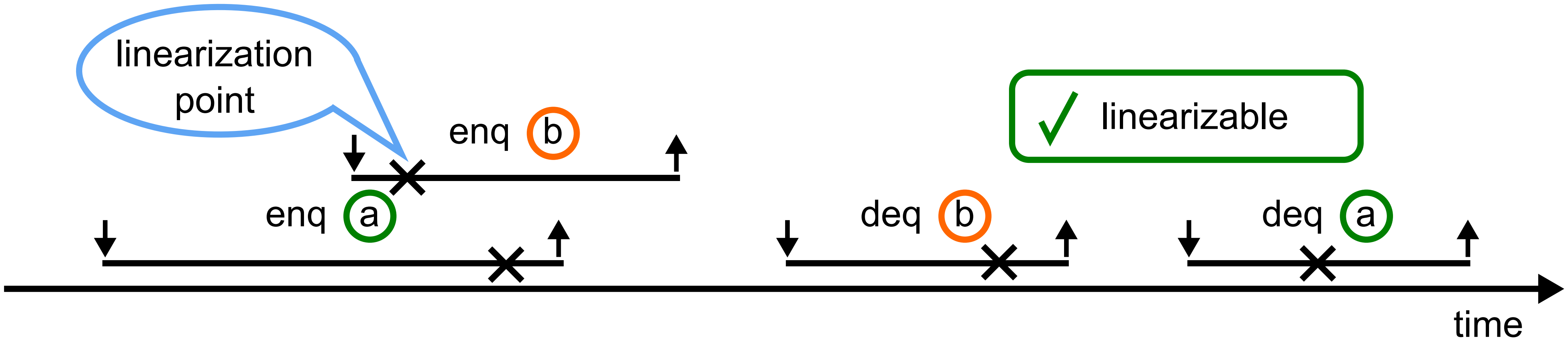

linearization point
enq b
enq a
deq b
deq a
✓ linearizable
time

Example

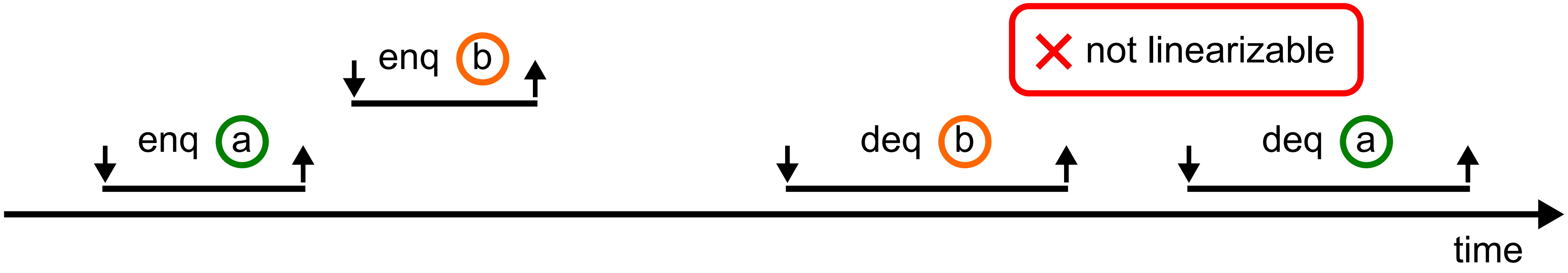

linearization point
enq b
enq a
deq b
deq a
✓ linearizable
time


enq b
enq a
deq b
deq a
time

Example

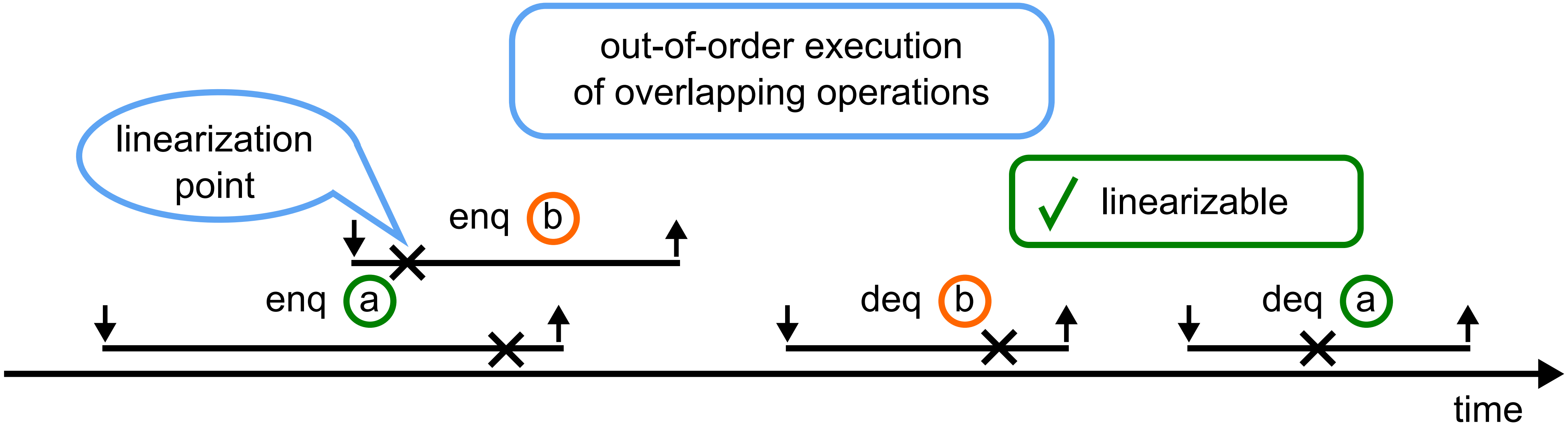


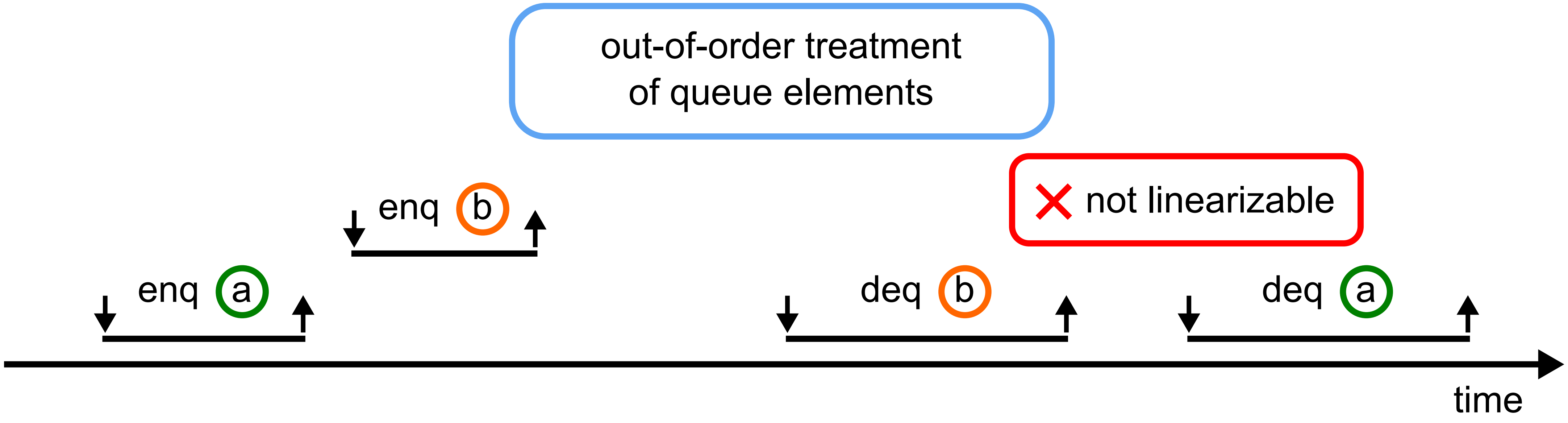
linearization point
enq b
enq a
✓ linearizable
deq b
deq a
time



enq b
enq a
✗ not linearizable
deq b
deq a
time

Example


out-of-order execution
of overlapping operations
linearization
point
enq b
enq a
✓ linearizable
deq b
deq a
time


out-of-order treatment
of queue elements
enq b
enq a
✗ not linearizable
deq b
deq a
time

Key Idea

1.) Record concurrent histories of various FIFO queue implementations.

2.) Analyze these concurrent histories using only the invocation times of operations.

Key Idea

1.) Record concurrent histories of various FIFO queue implementations.

2.) Analyze these concurrent histories using only the invocation times of operations.

- Ideally operations would take zero time

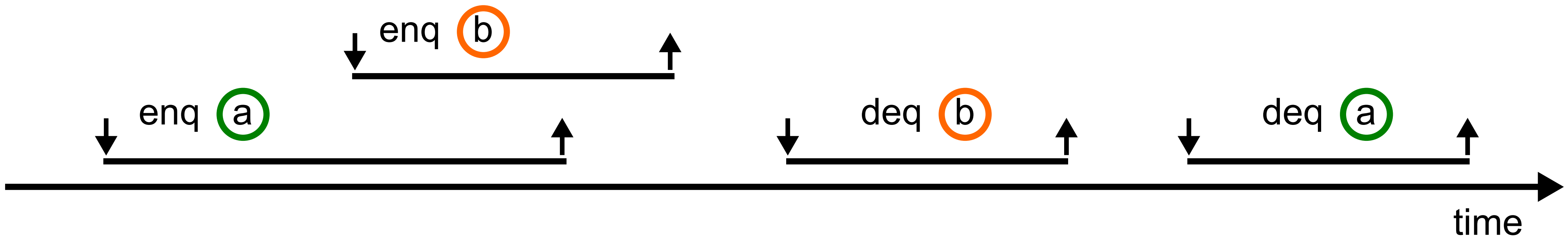
Key Idea

1.) Record concurrent histories of various FIFO queue implementations.

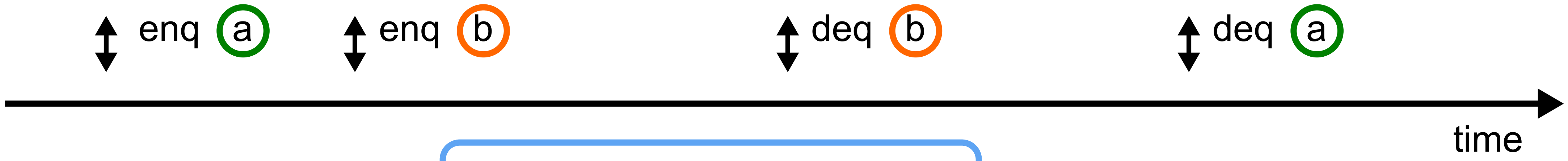
2.) Analyze these concurrent histories using only the invocation times of operations.

- Ideally operations would take zero time
- Independent of the execution time of operations

Element-Fairness



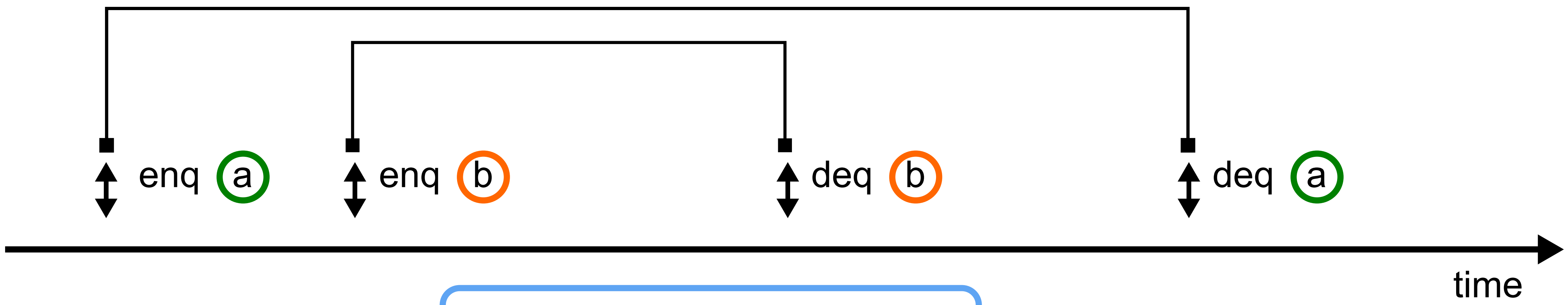
Element-Fairness



zero-time linearization

Element-Fairness

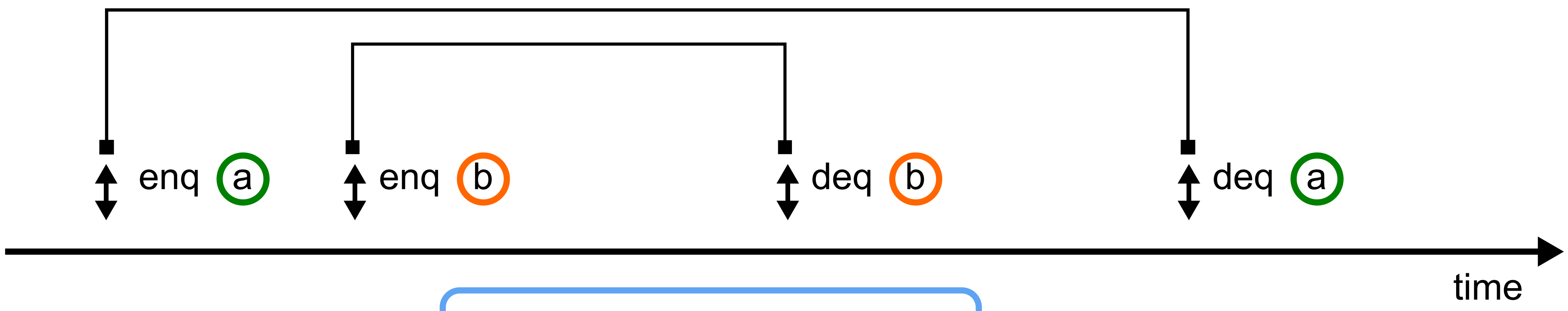
element b overtakes element a



zero-time linearization

Element-Fairness

element (b) overtakes element (a)



zero-time linearization

Definition

element-fairness =
number of overtakings in the zero-time linearization

Experiments

all threads do in parallel

```
for 10.000 iterations
{
    enqueue unique element

    dequeue element

}
```

Experiments

all threads do in parallel

```
for 10.000 iterations
{
        enqueue unique element
        calculate Pi
        dequeue element
        calculate Pi
}
```

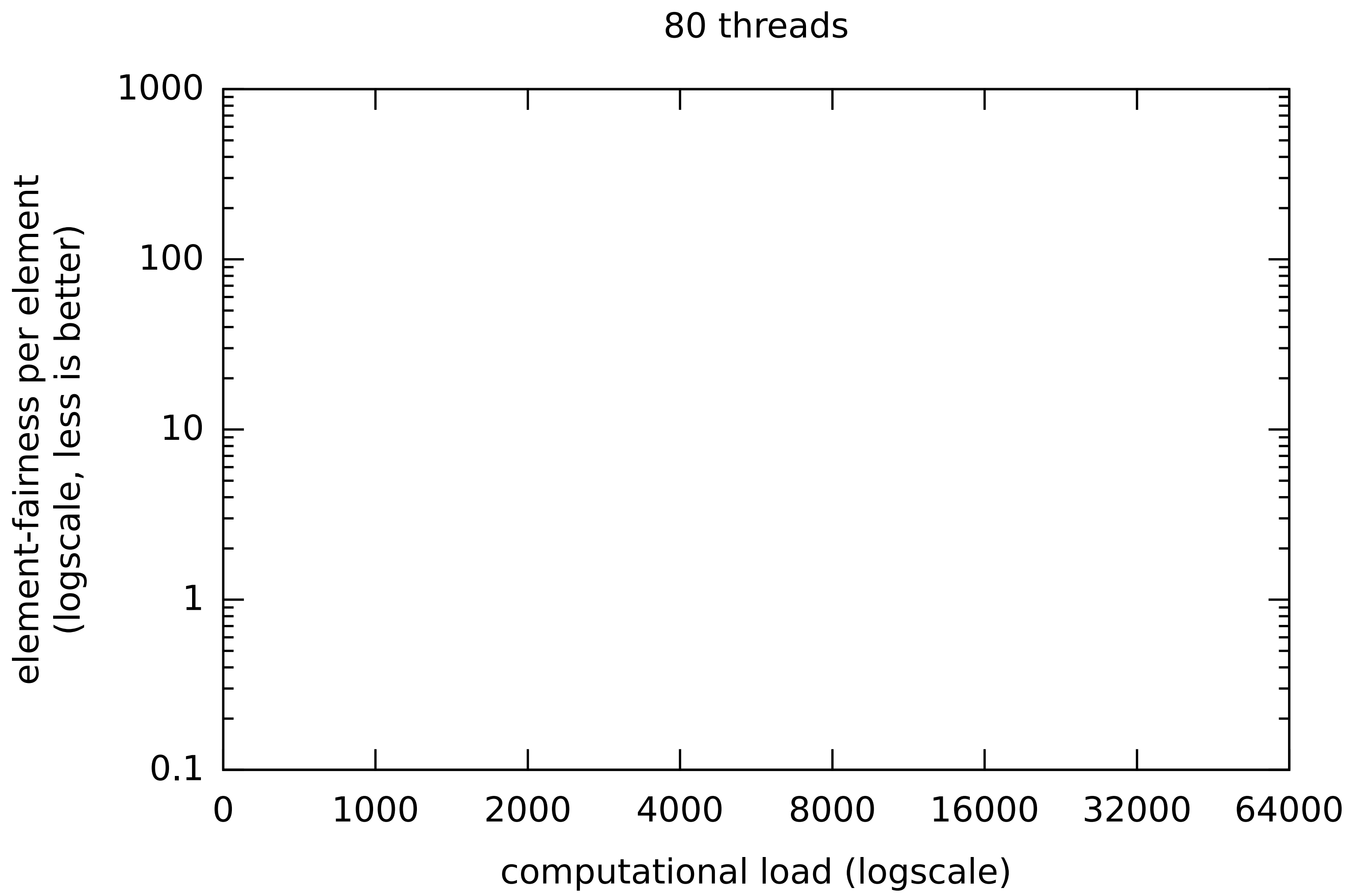
Experiments

all threads do in parallel

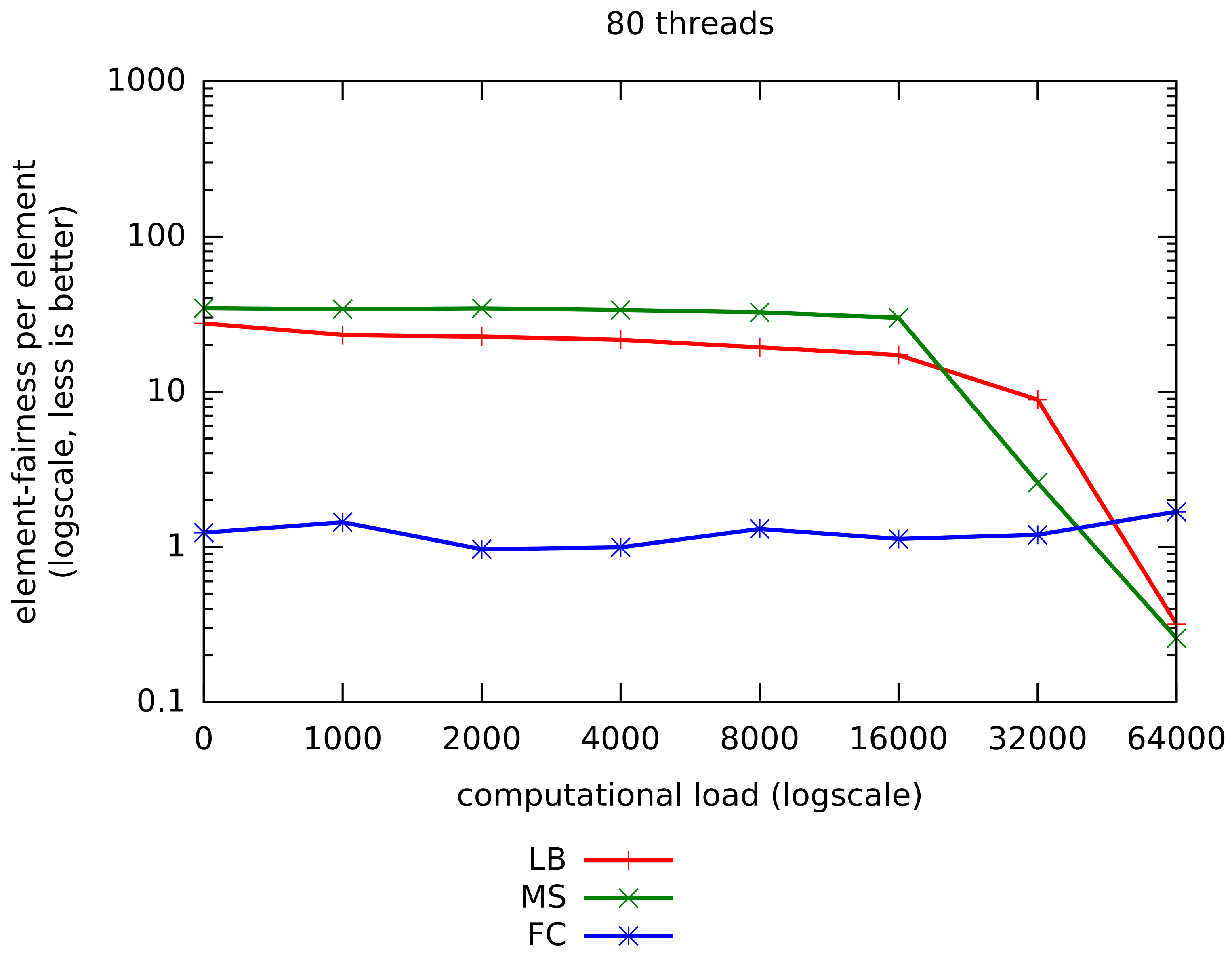
```
for 10.000 iterations
{
        enqueue unique element
        calculate Pi
        dequeue element
        calculate Pi
}
```

- No dequeues in the first 200 iterations to avoid empty checks
- No enqueues in the last 200 iterations to empty the queue.

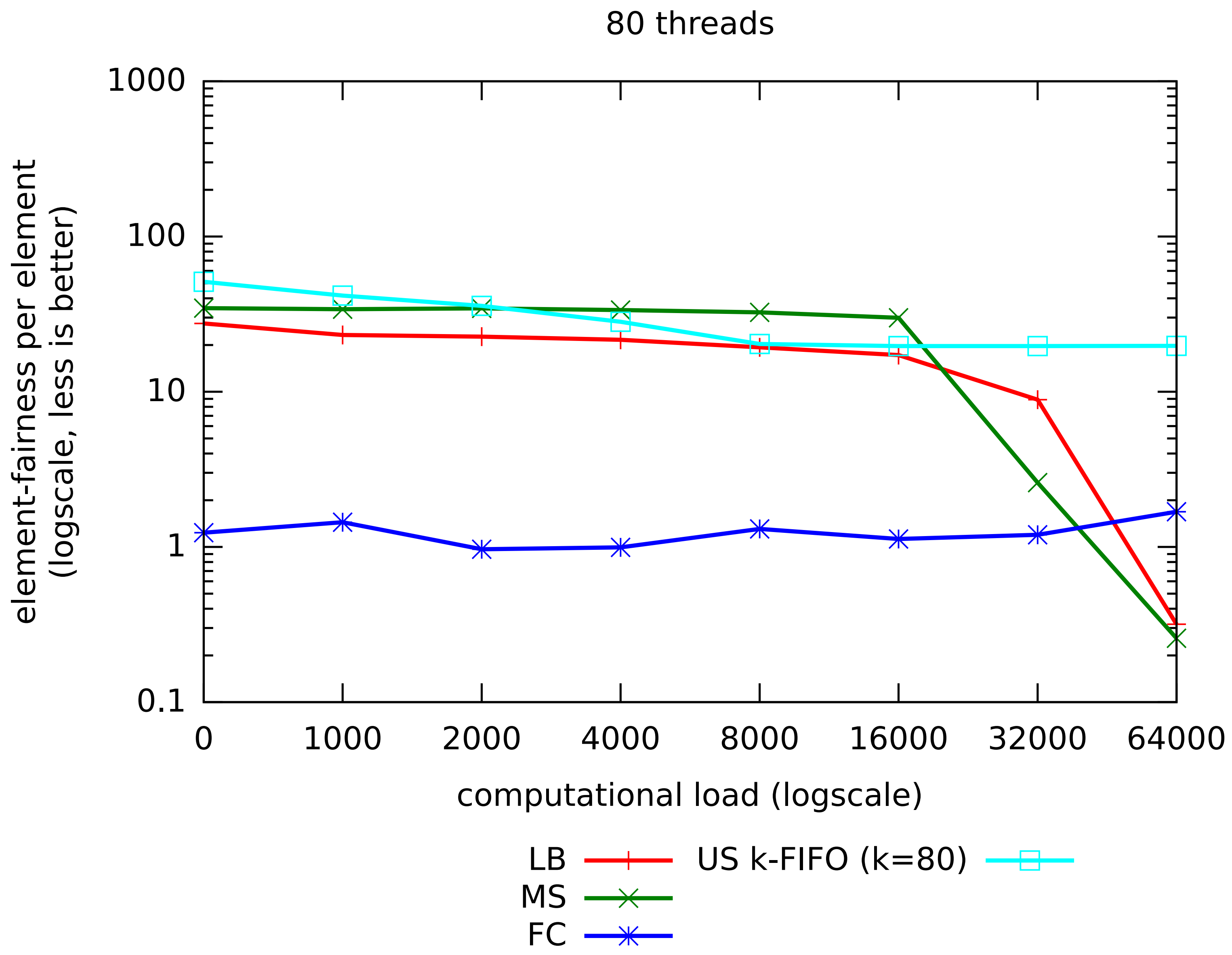
Element-Fairness per Element



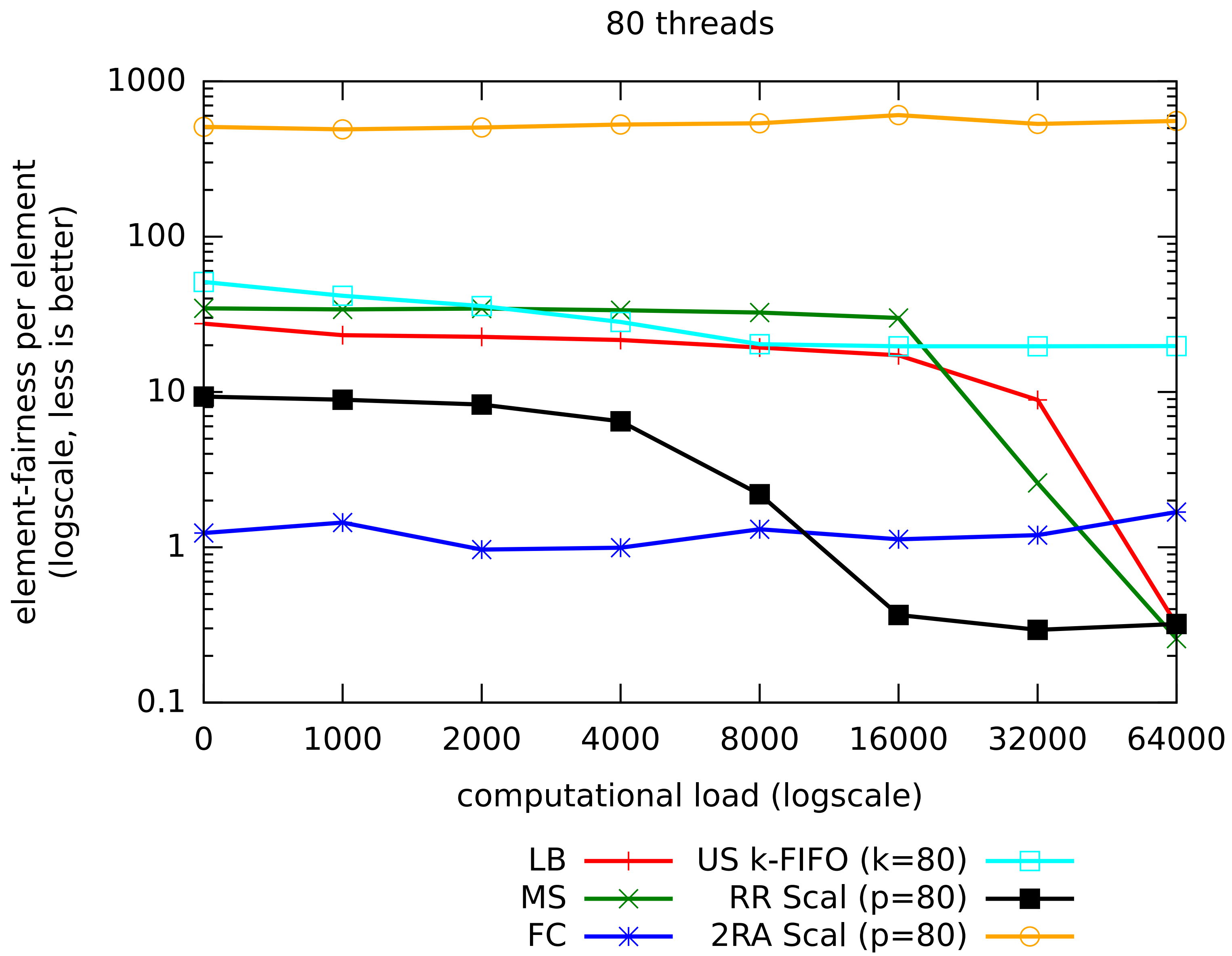
Element-Fairness per Element



Element-Fairness per Element



Element-Fairness per Element



Operation-Fairness

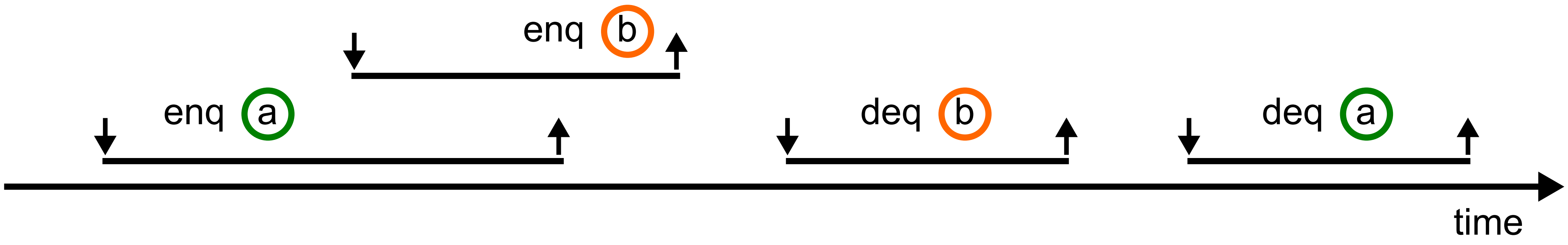
- Measure the out-of order execution of single operations

Operation-Fairness

- Measure the out-of order execution of single operations
- Observation: Linearization points induce a strict order on the queue operations

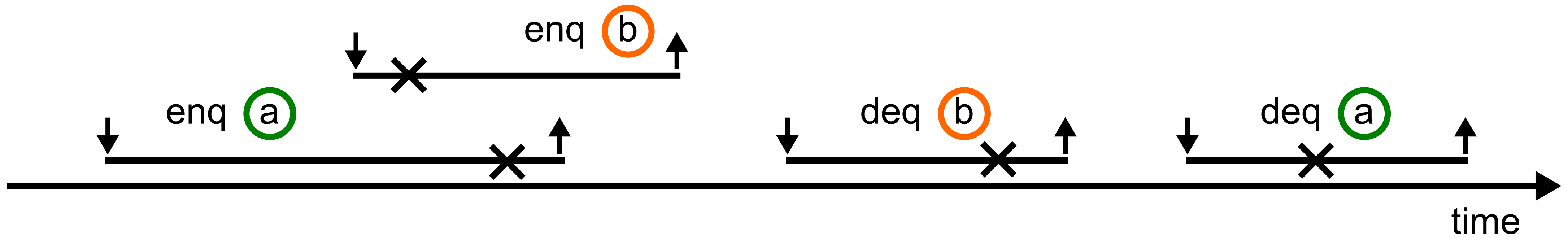
Operation-Fairness

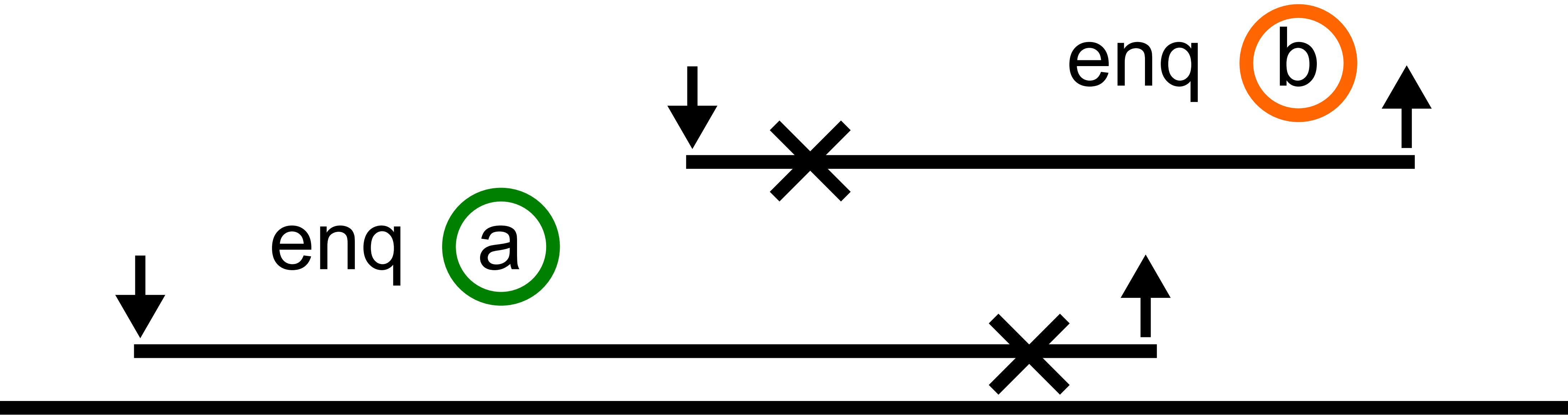
- Measure the out-of order execution of single operations
- Observation: Linearization points induce a strict order on the queue operations



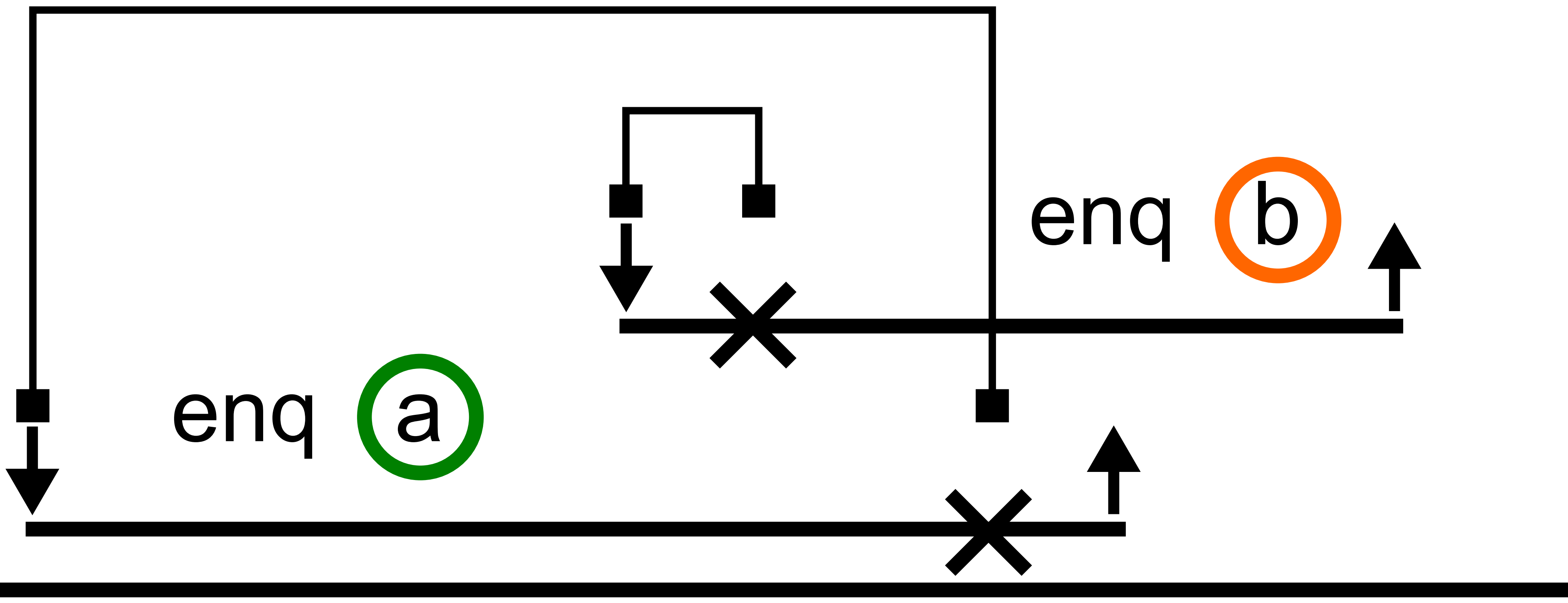
Operation-Fairness

- Measure the out-of order execution of single operations
- Observation: Linearization points induce a strict order on the queue operations

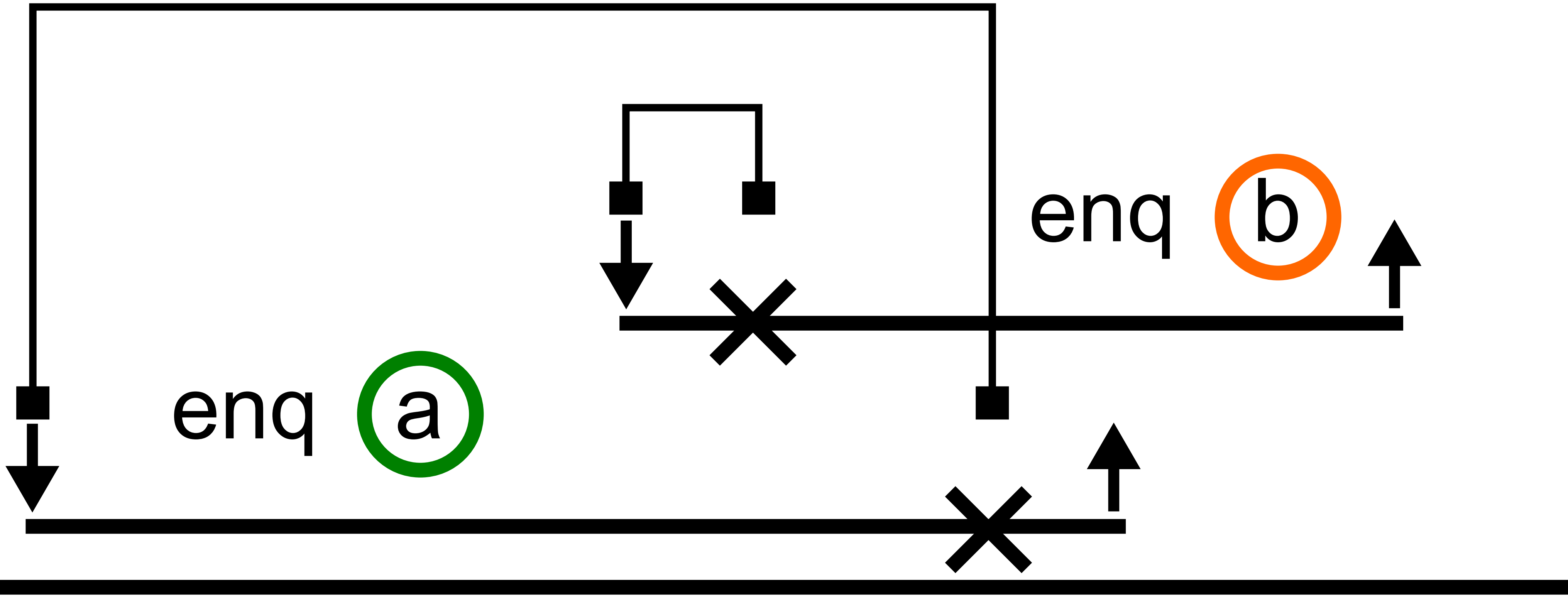



enq b
enq a

operation enq ⓑ overtakes operation enq ⓐ



operation enq ⓑ overtakes operation enq ⓐ



Definition

operation-fairness =
number of overtakings in a concurrent history

Operation-Age and Operation-Lateness

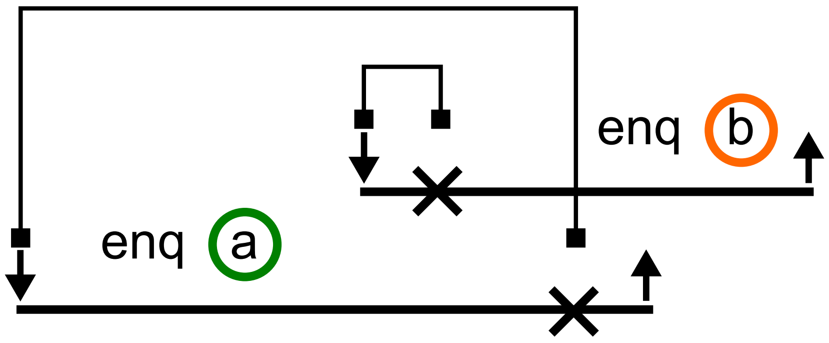
Definition

age (op) =
number of operations op overtakes

Operation-Age and Operation-Lateness

Definition

age (op) =
number of operations op overtakes



age(enq a) = 0
age(enq b) = 1

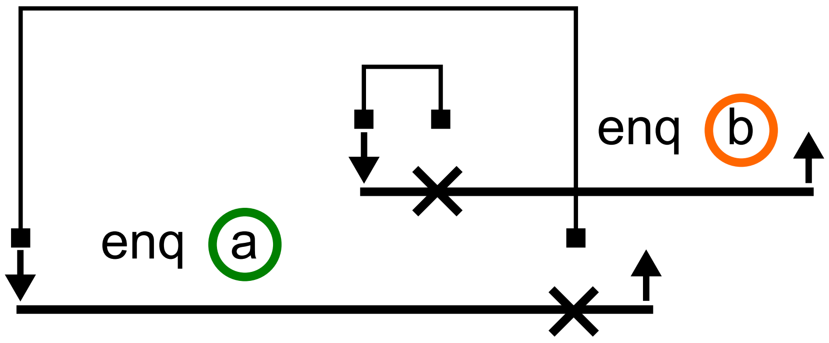
Operation-Age and Operation-Lateness

Definition

age (op) =
number of operations op overtakes

Definition

lateness (op) =
number of operations which overtake op



age(enq a) = 0
age(enq b) = 1

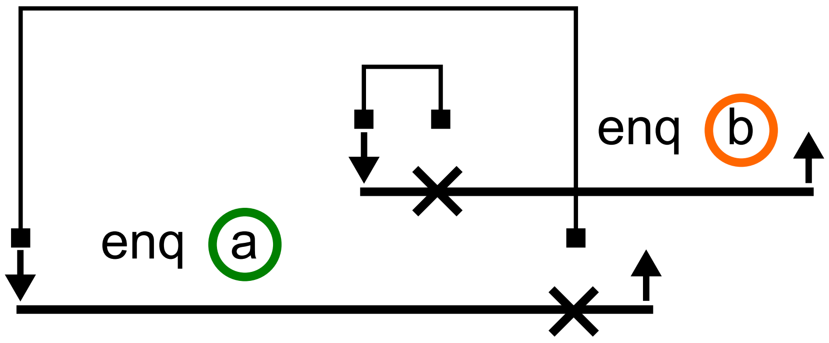
Operation-Age and Operation-Lateness

Definition

age (op) =
number of operations op overtakes

Definition

lateness (op) =
number of operations which overtake op



age(enq a) = 0
age(enq b) = 1

lateness(enq a) = 1
lateness(enq b) = 0

Experiments (2)

- Only for strict FIFO queue implementations at the moment.
  - Measuring relaxed implementations is future work.

Experiments (2)

- Only for strict FIFO queue implementations at the moment.
  - Measuring relaxed implementations is future work.

```
all threads do in parallel

for 10.000 iterations
{
        enqueue unique element
        calculate Pi
}
```

Experiments (2)

- Only for strict FIFO queue implementations at the moment.
  - Measuring relaxed implementations is future work.

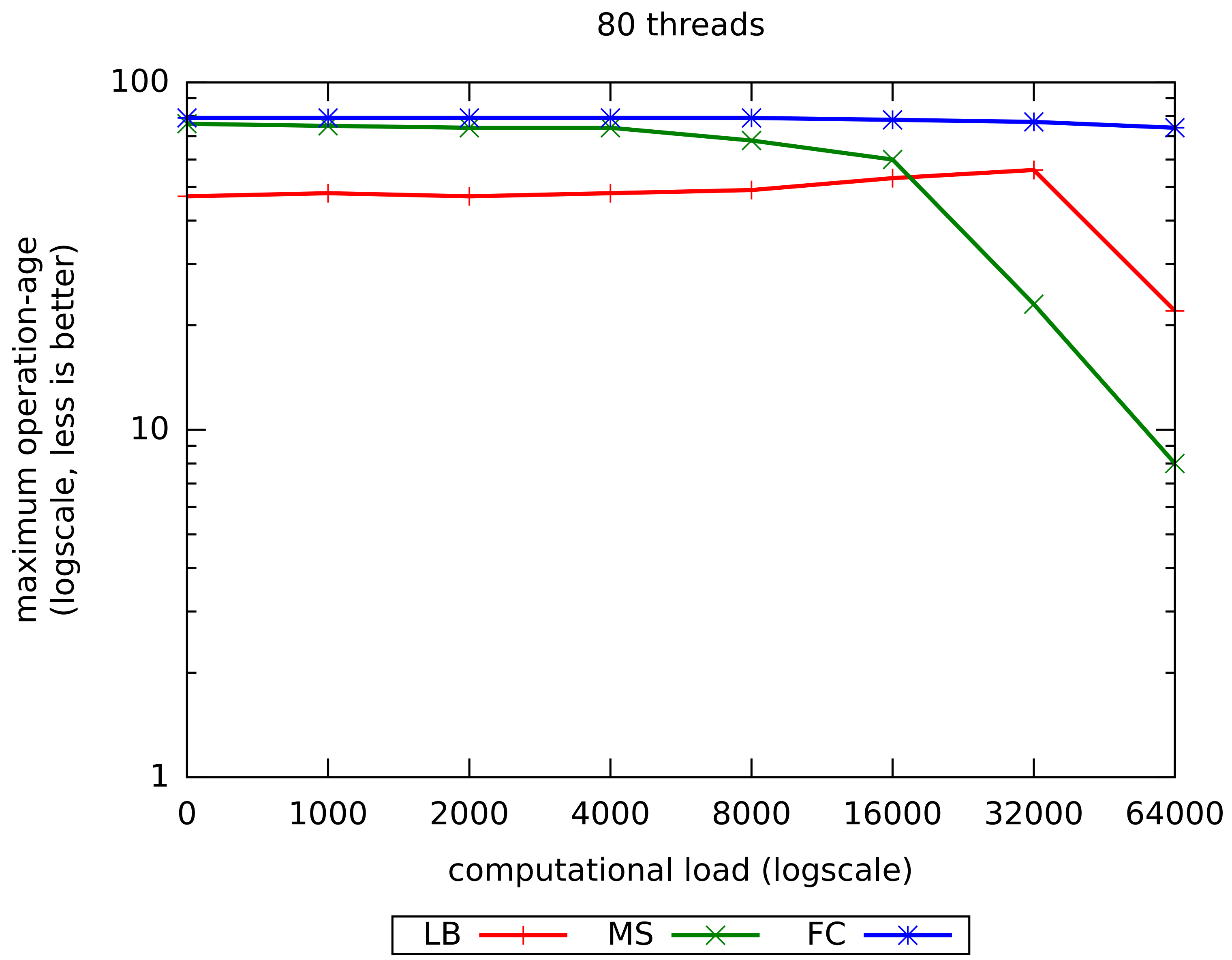
```
all threads do in parallel

for 10.000 iterations
{
    enqueue unique element
    calculate Pi
}
```

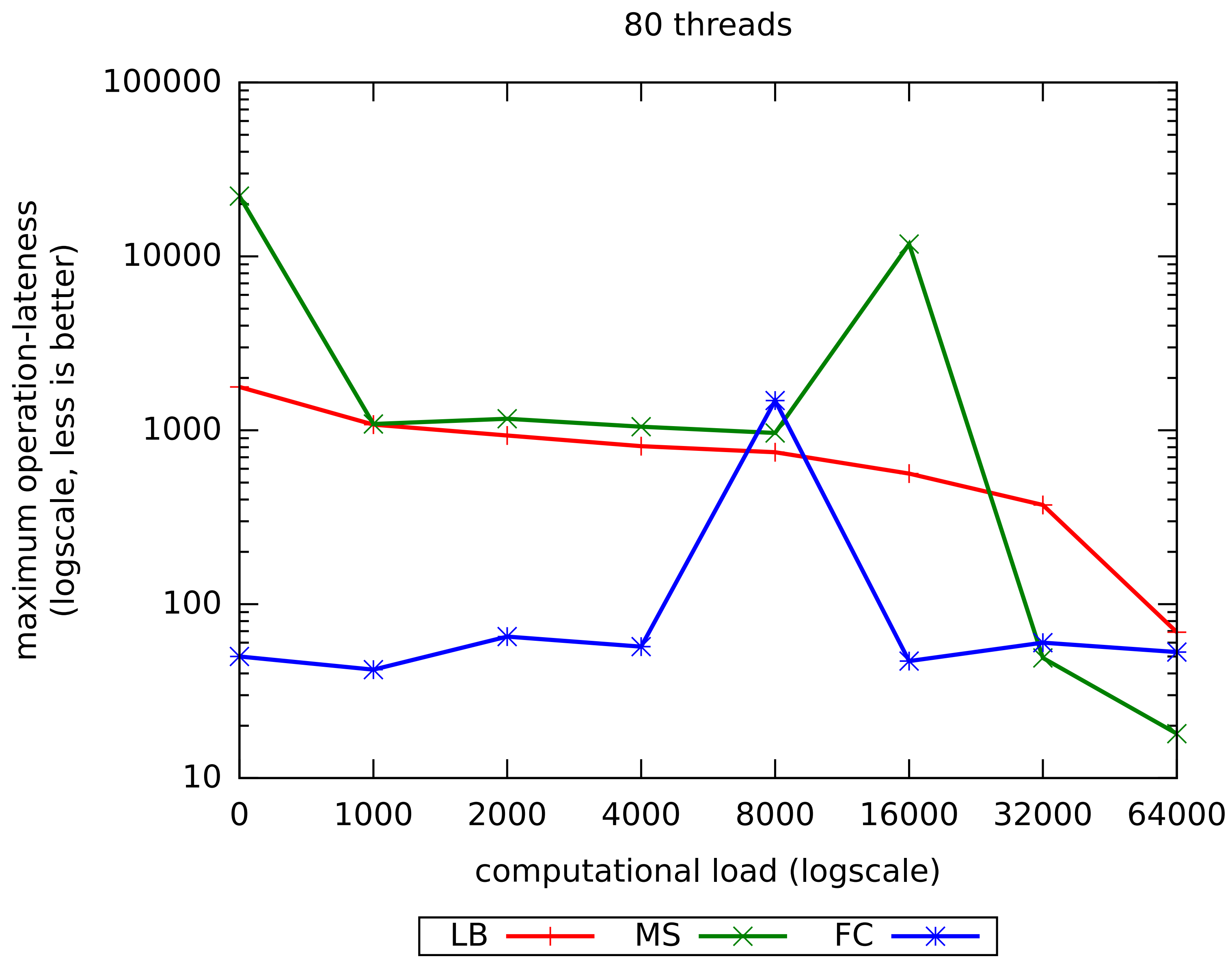
```
one thread does

dequeue all elements sequentially
```

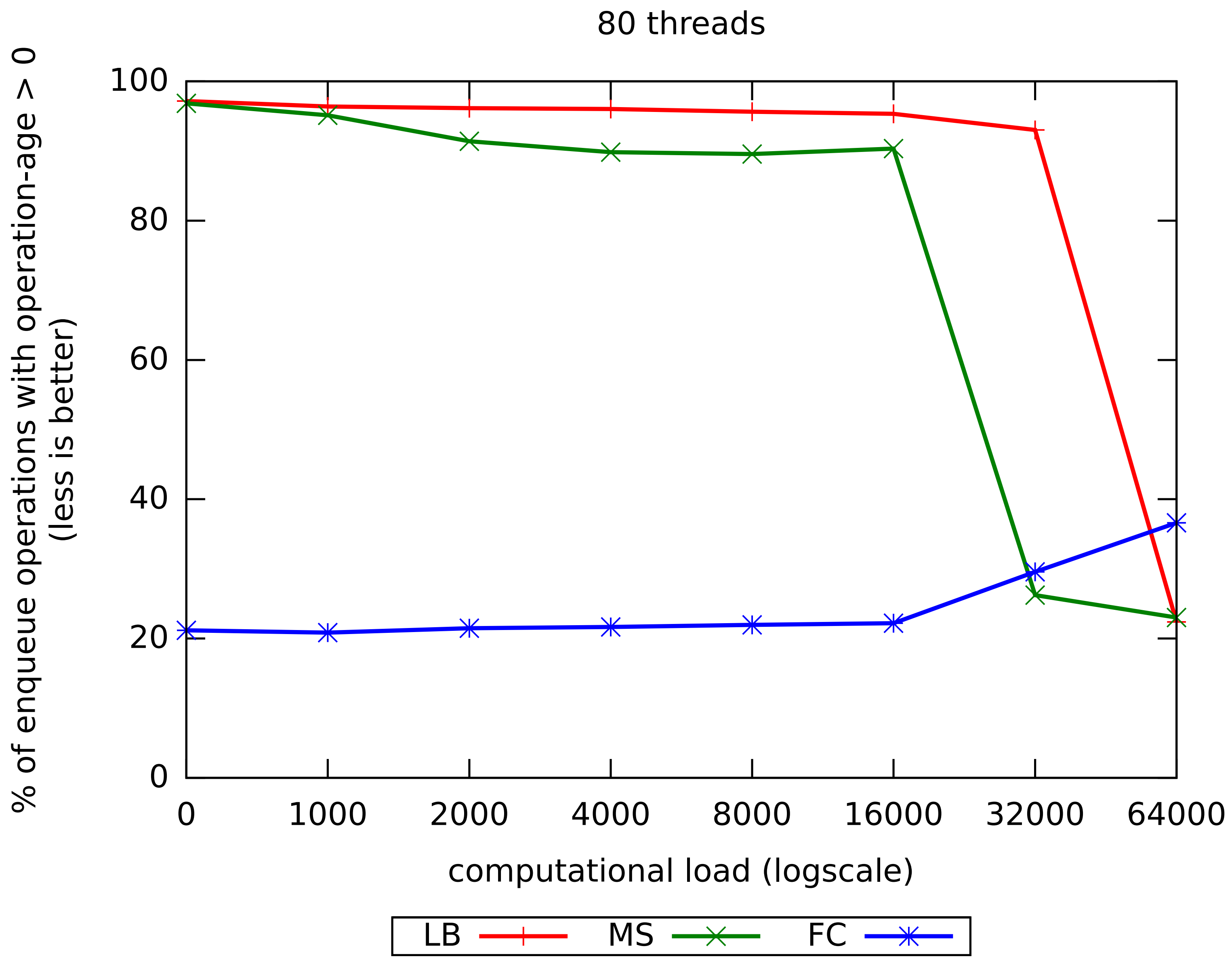
Maximum Operation-Age



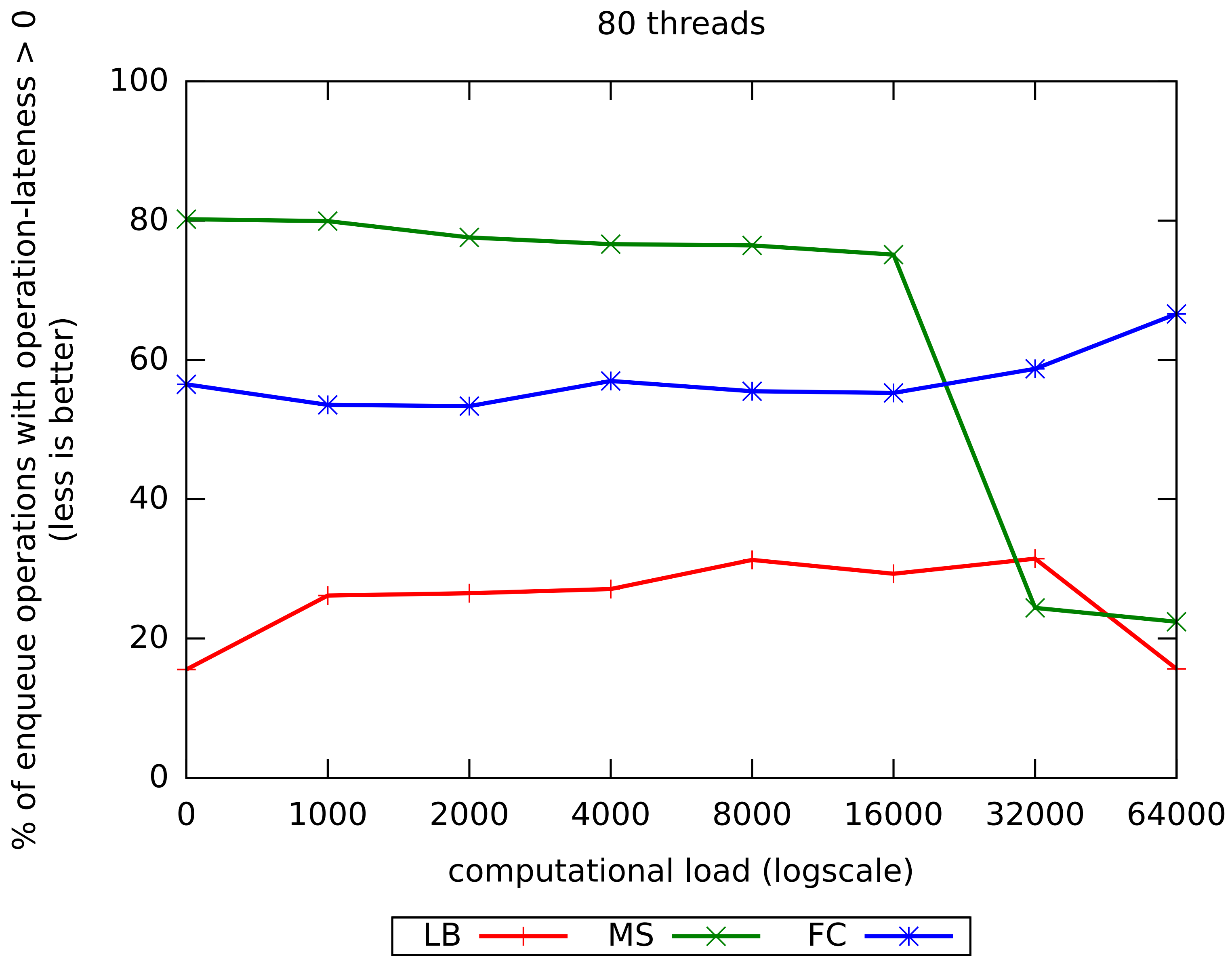
Maximum Operation-Lateness



Number of Overtaking Operations



Number of Overtaken Operations



Conclusion

- We introduced metrics to compare the behavior of various FIFO queue implementations.
    - Relaxed implementation can appear more FIFO than strict implementations.
- Future work
    - Measure operation-fairness of relaxed FIFO queue implementations.
    - Use element-fairness to analyze implementation of other data structures, e.g. stacks.

Thank You

For more information about the queue implementations see
http://scal.cs.uni-salzburg.at/

Additional measurement results can be seen on
http://scal.cs.uni-salzburg.at/races12/