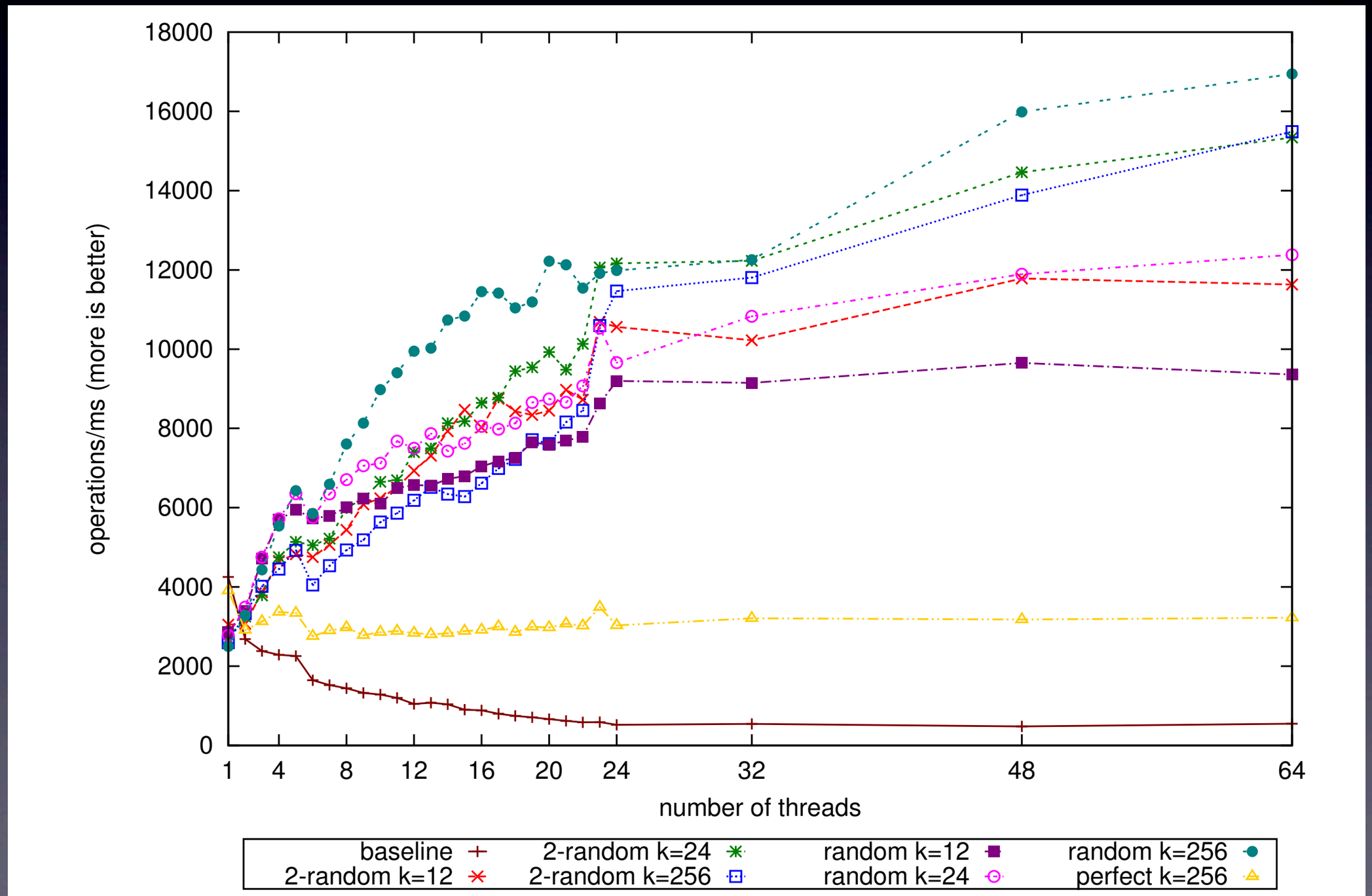
Scal☠: Non-Linearizable Computing Breaks the Scalability Barrier

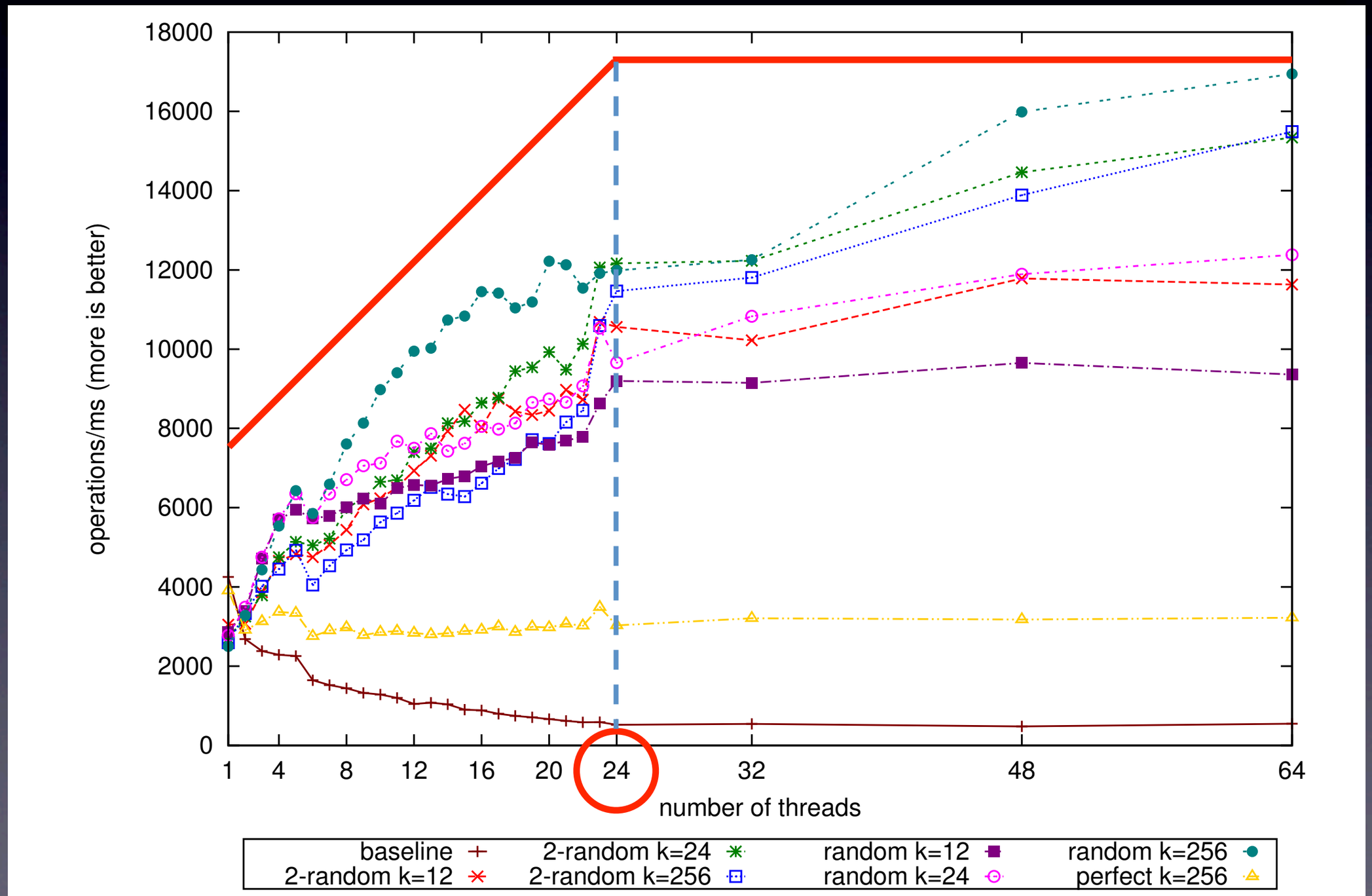
Christoph Kirsch, Hannes Payer, Harald Röck
Universität Salzburg

CHESS Seminar, UC Berkeley, November 2010

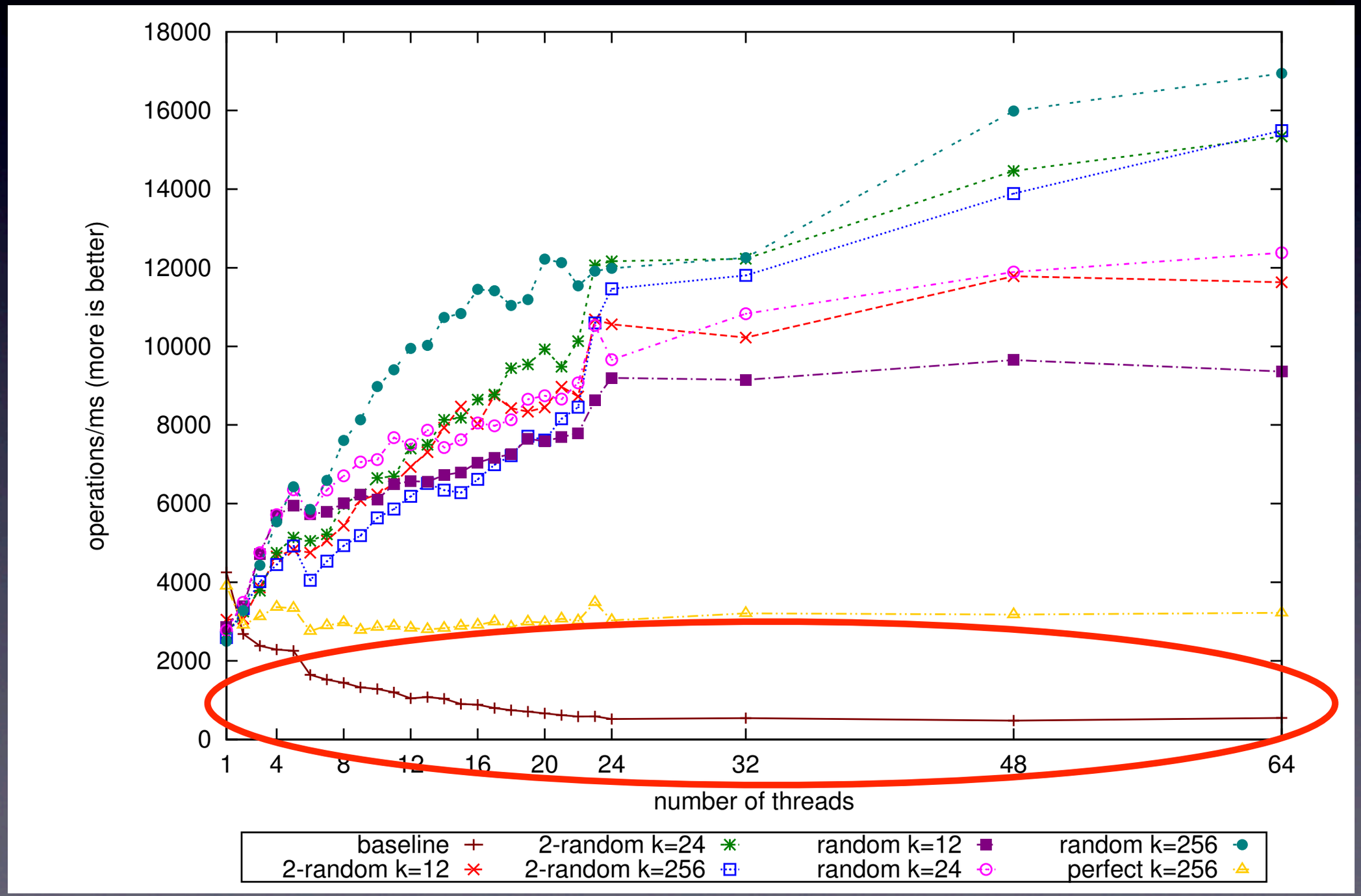
Multicore Scalability



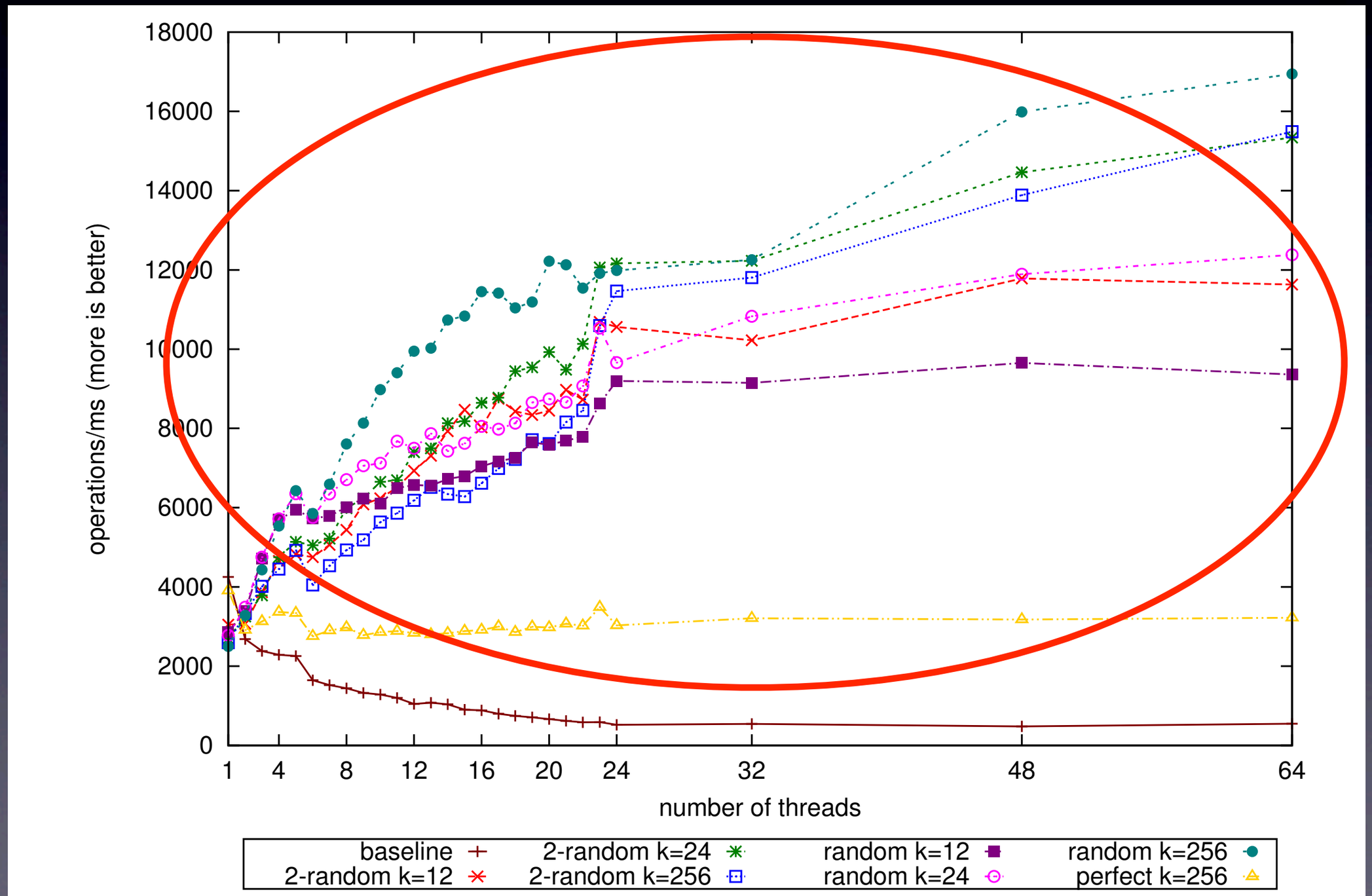
Ideal 24-Core Performance



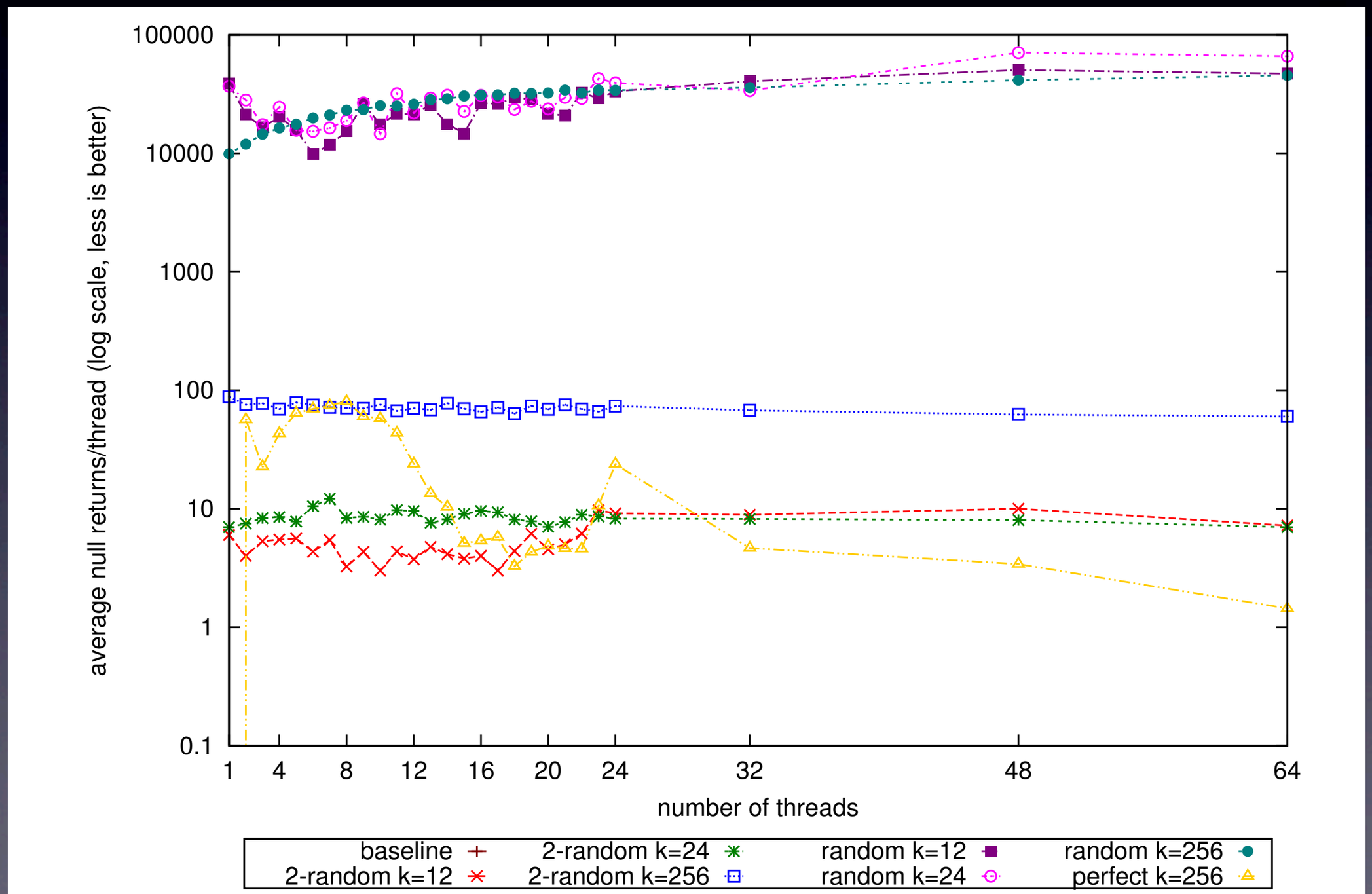
Actual Lock-free FIFO Queue



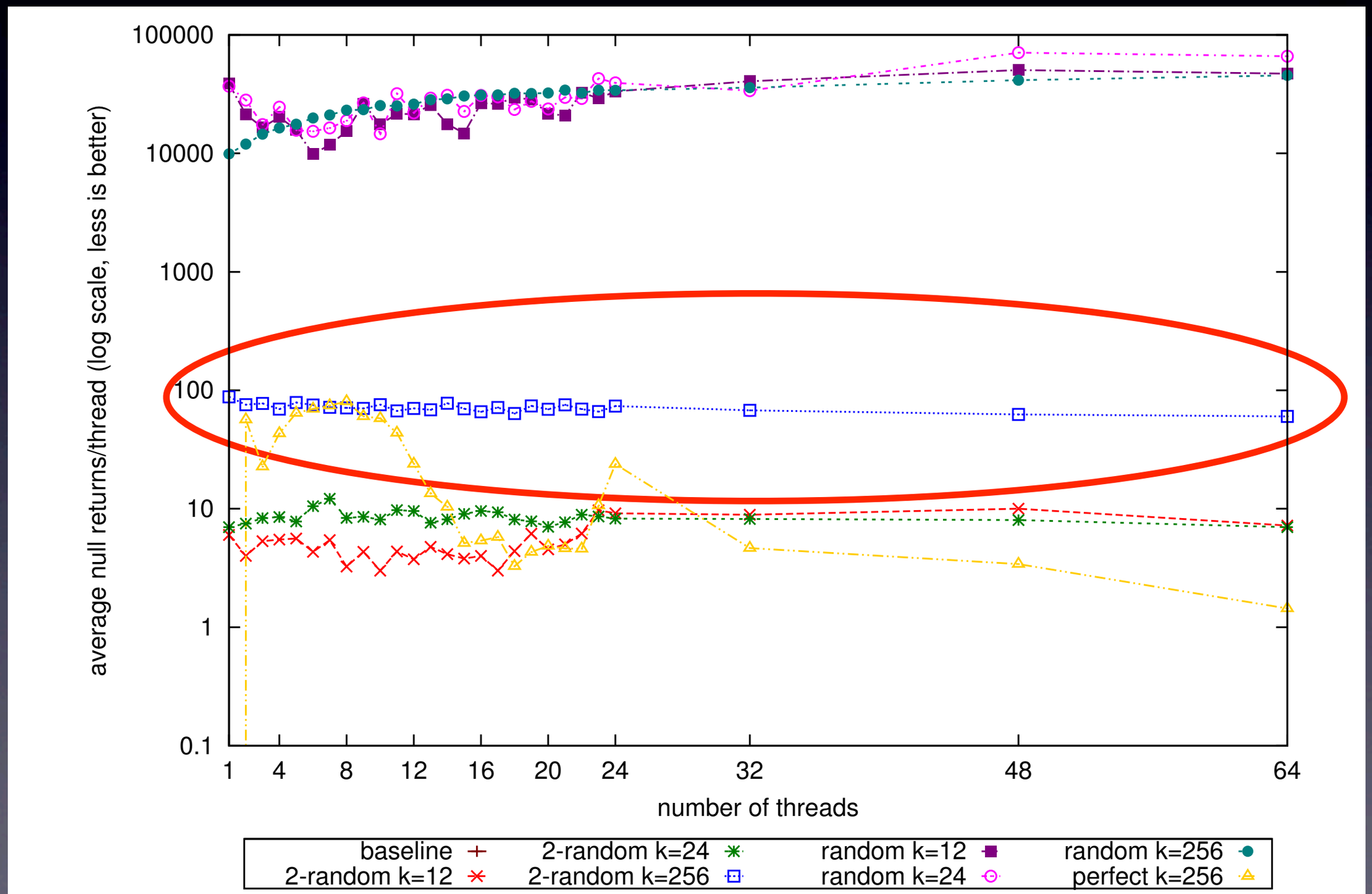
k-Linearizable



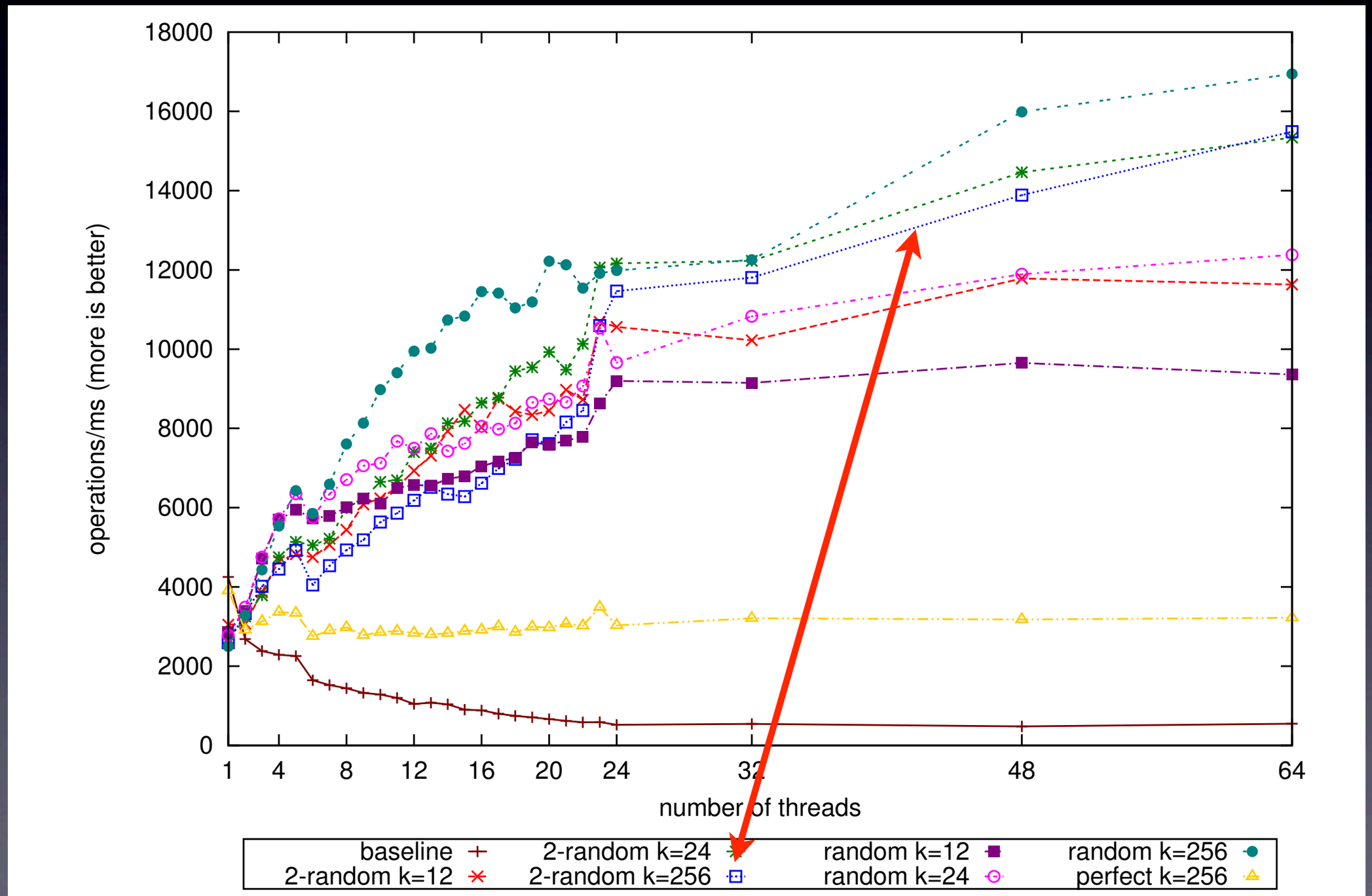
Semantics vs. Scalability



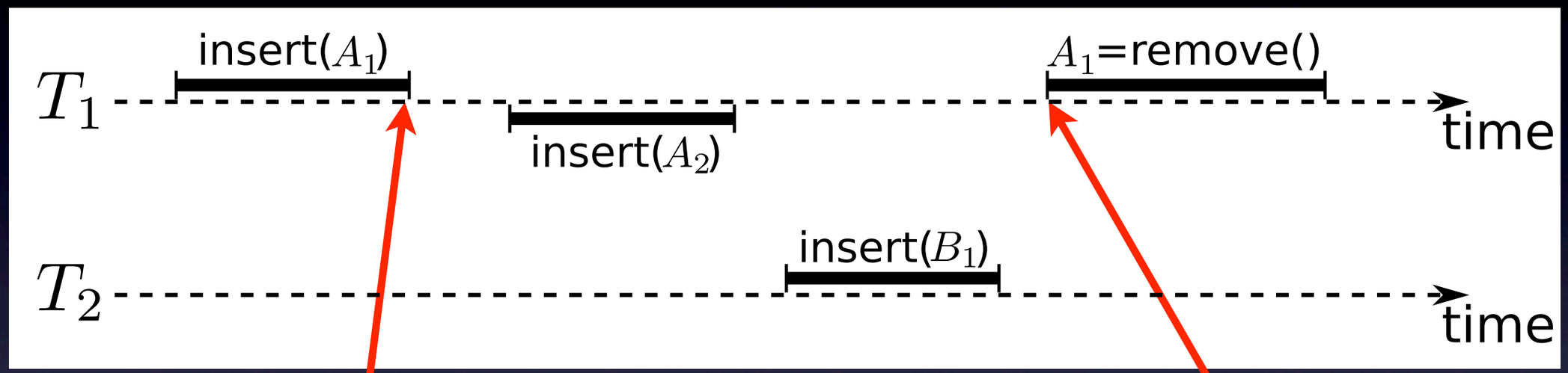
Best Trade-off



2-random k=256

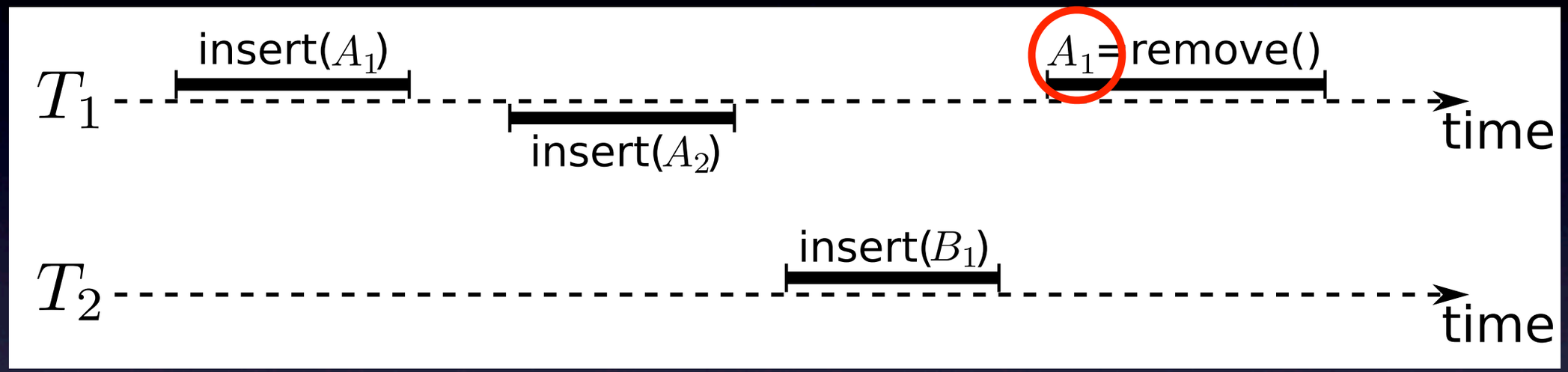


History



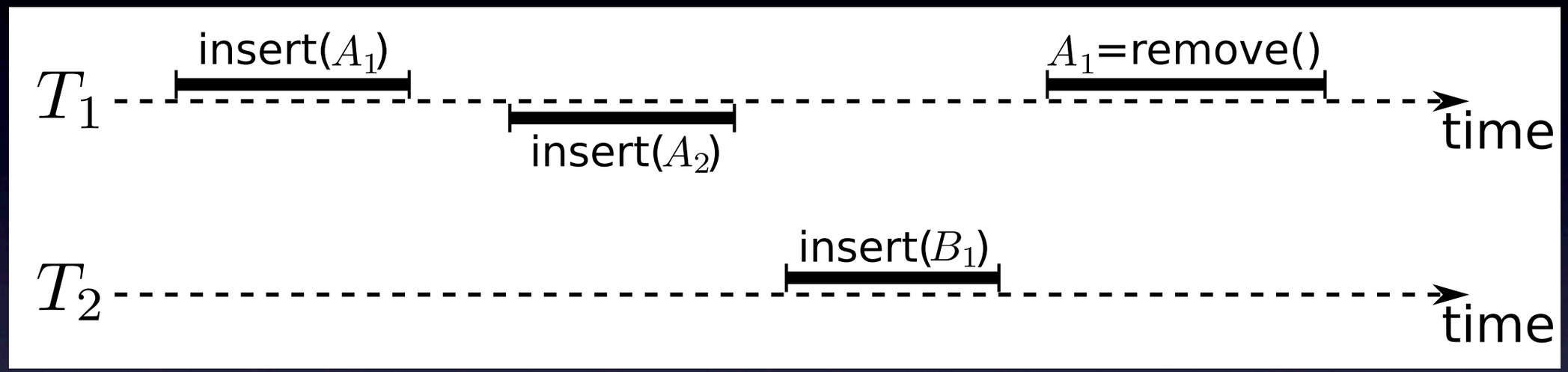
- a history is a finite sequence of invocations and responses of operations
- an operation is atomic if its invocation is immediately followed by its response
- an operation P1 precedes an operation P2 if P1's response happens before P2's invocation

Sequentiality



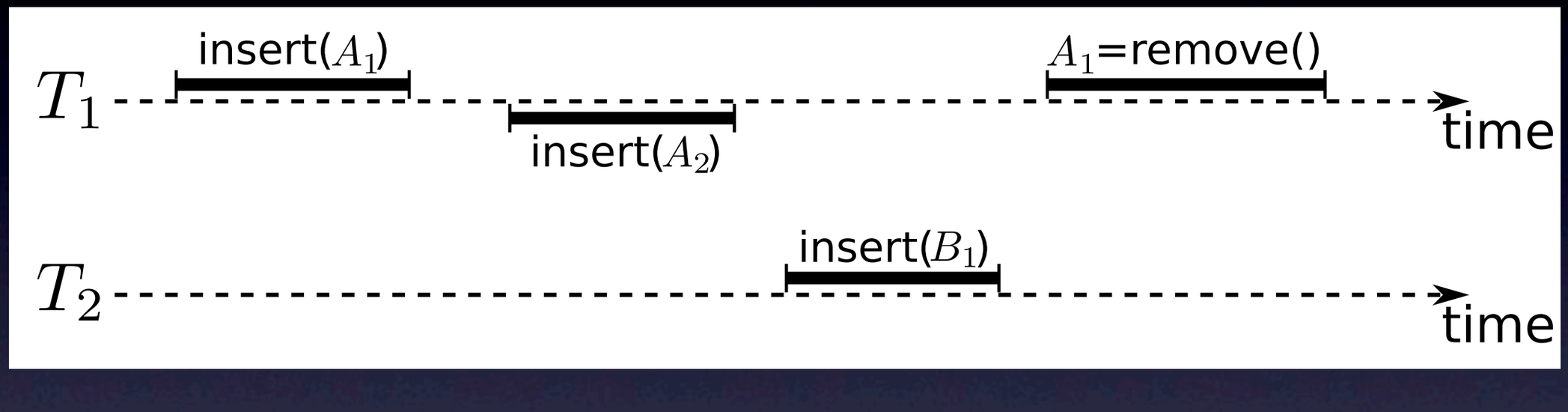
- a sequential history provides atomicity while preserving single-threaded precedence
- e.g. ins(A1)-ins(A2)-ins(B1)-rem() is sequential
- in fact, any atomic occurrence of ins(B1) is
- however, ins(B1)-ins(A1)-ins(A2)-rem() is not serializable for, e.g. a FIFO queue, if A1=rem()

Serializability



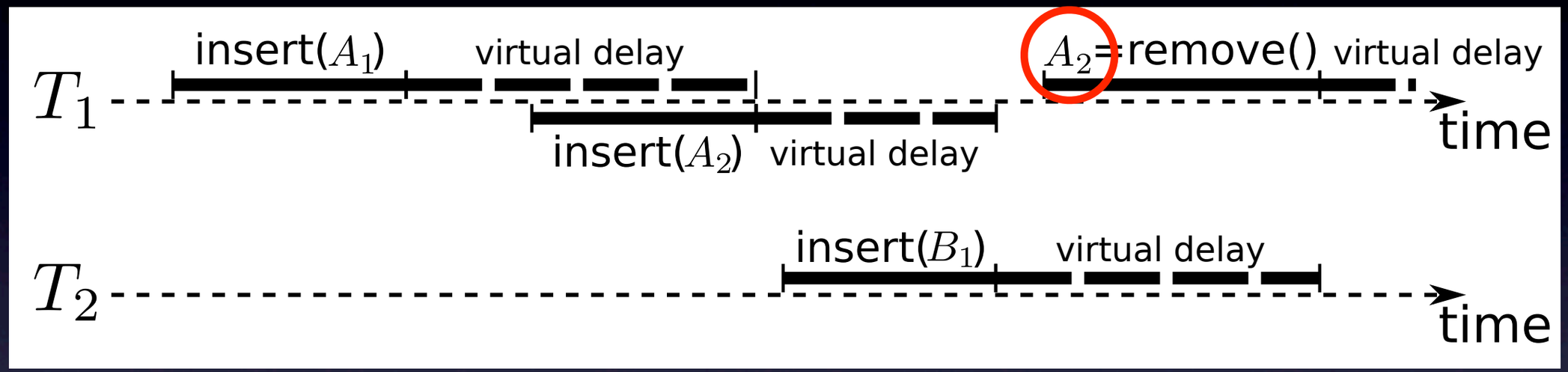
- a history H is serializable with respect to an object O if H has a sequential history that preserves the semantics of O
- ins(A1)-ins(B1)-ins(A2)-A1=rem(): <B1,A2>
  ins(A1)-ins(A2)-ins(B1)-A1=rem(), and
  ins(A1)-ins(A2)-A1=rem()-ins(B1): <A2,B1>

Linearizability



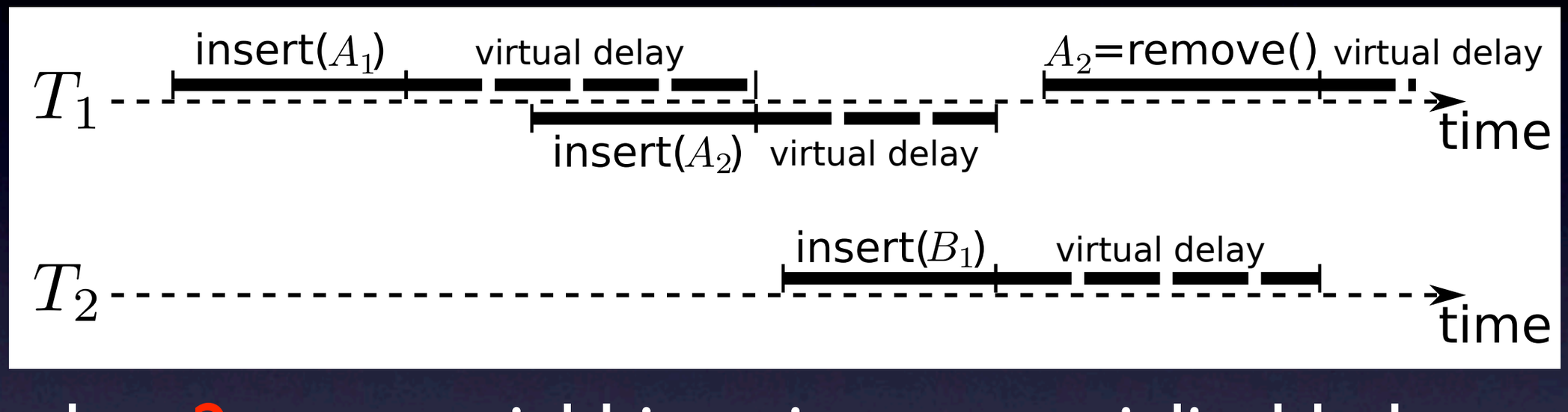
- yet only ins(A1)-ins(A2)-ins(B1)-A1=rem() with <A2,B1> is linearizable
- a history H is linearizable if H has a sequential history that is serializable and preserves multi-threaded precedence
- linearizability is compositional!

k-Sequentiality



- a k-sequential history is a sequential history where each response is virtually delayed until the (k-1)th response
- 1-sequentiality is equivalent to sequentiality
- here: any sequential history is not serializable, if A2=rem(), and thus not linearizable

k-Linearizability



- but 2-sequential histories are serializable here: ins(A2)-ins(A1)-ins(B1)-A2=rem() and ins(A2)-ins(A1)-A2=rem()-ins(B1) with <A1,B1>, and thus even 2-linearizable!
- a history H is k-linearizable if H has a k-sequential history that is serializable and preserves multi-threaded precedence

Compositionality

- 1-linearizability is equivalent to linearizability
- k-linearizability remains compositional!

Monotonicity

- semantics (and scalability) of k-linearizability is monotone in k, for example:
  - a k-linearizable stack may not return the youngest but up to the k-youngest element
  - a k-linearizable FIFO queue may not return the oldest but up to the k-oldest element

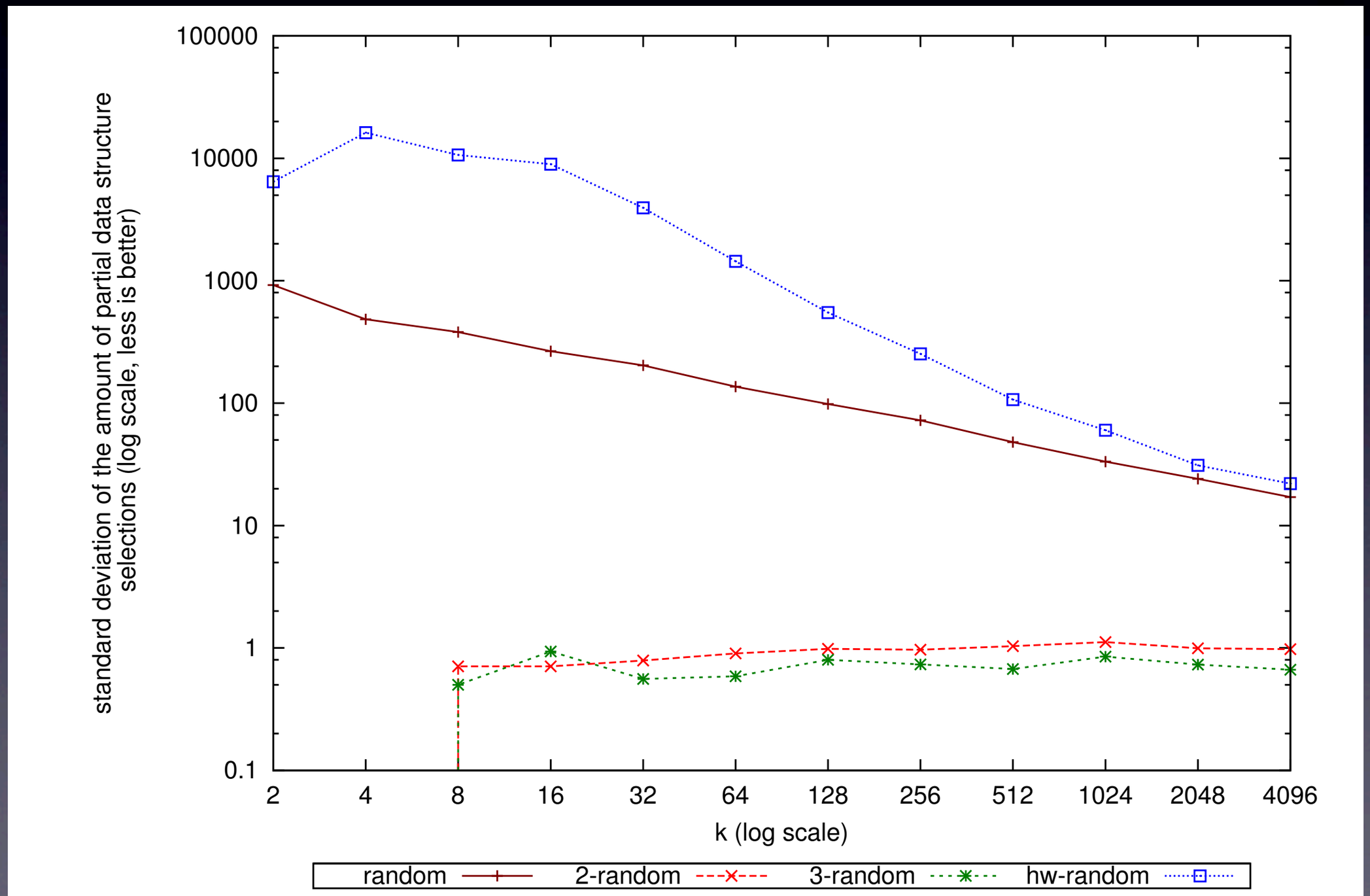
Select Function

Listing 1: Scal generic structure

```
1 op(data_structure, parameters) {
2   partial_ds = select(data_structure);
3   return partial_op(partial_ds, parameters);
4 }
```

- Scal implements a k-linearizable version of a given data structure using a select function and k identical instances of the given structure
- perfect select: 100% k-linearizable
- random select: k-linearizable with high prob.
- d-random select: k-linearizable with high prob.

Quality of Random

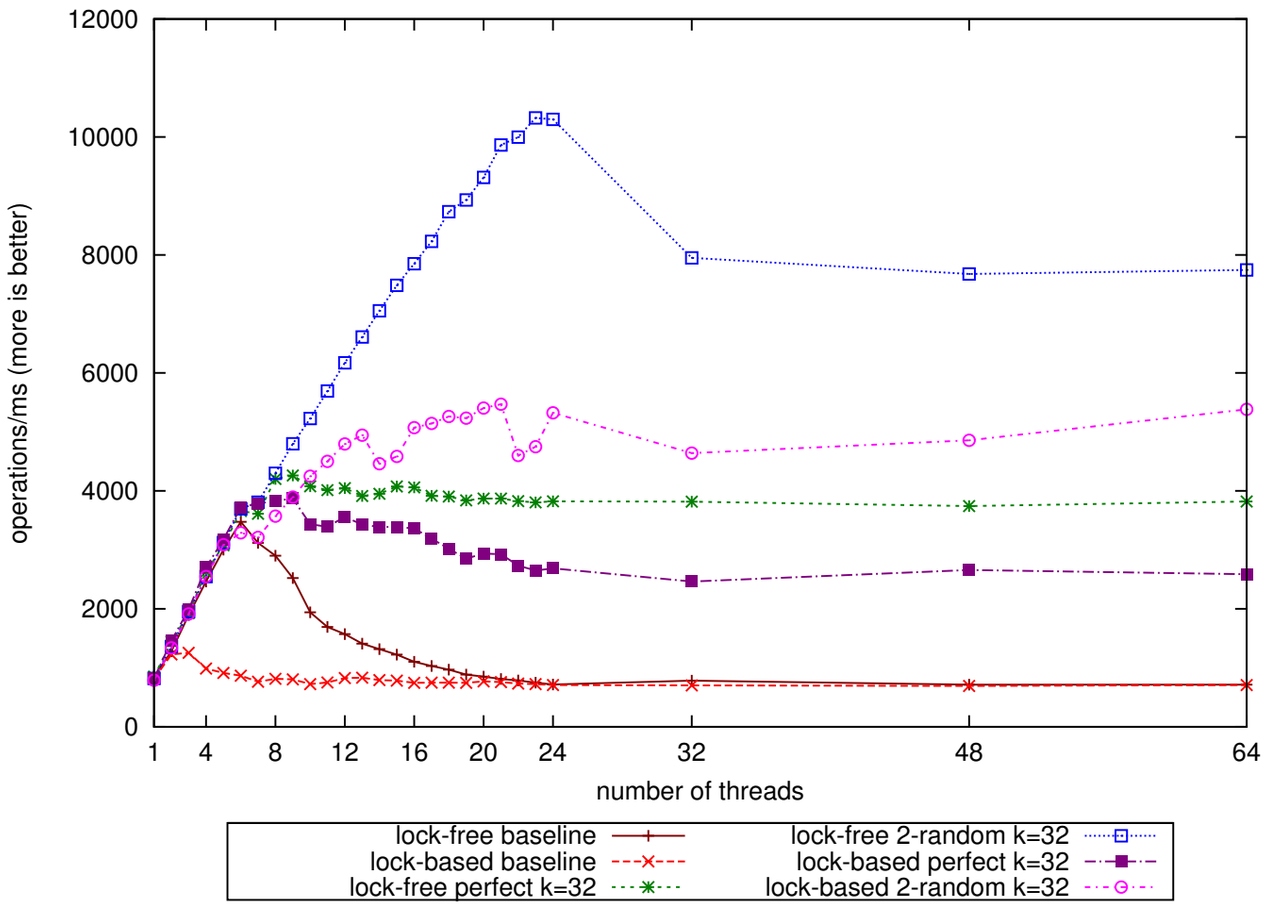


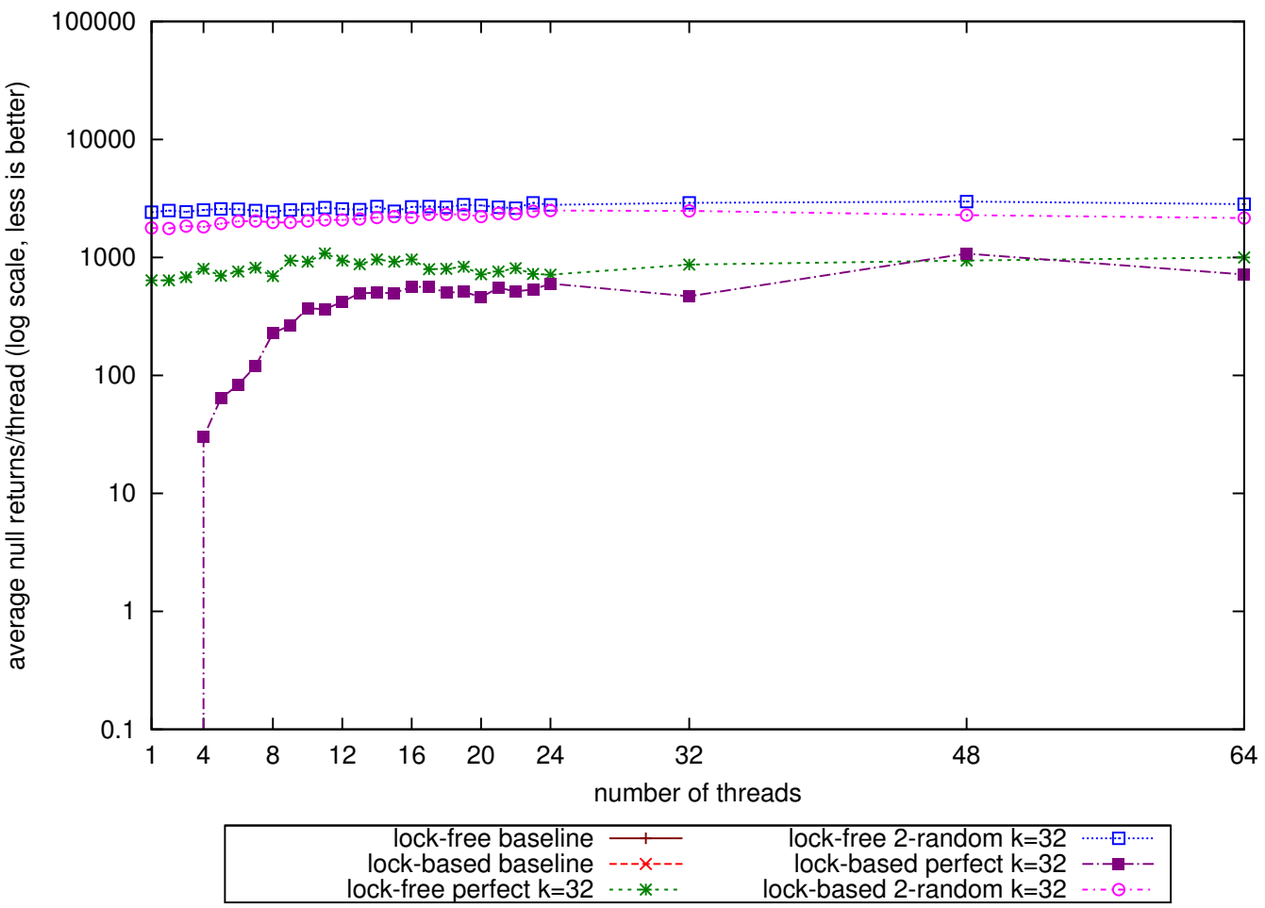
No vs. High Contention

select function	no contention	high contention
perfect	51 ns	3113 ns
random	59 ns	64 ns
2-random	108 ns	259 ns

Table 1: Select function overhead

Medium Computational Load

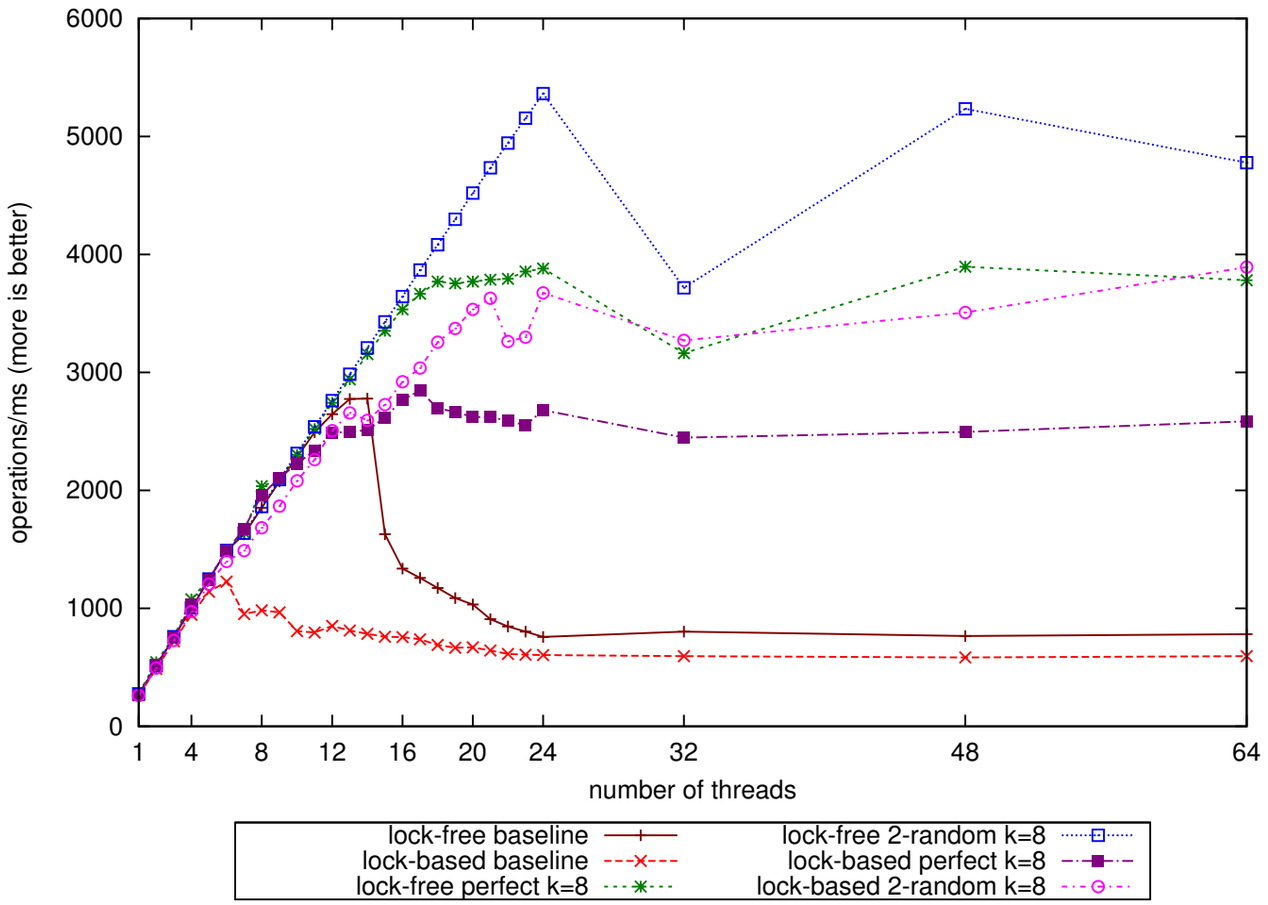


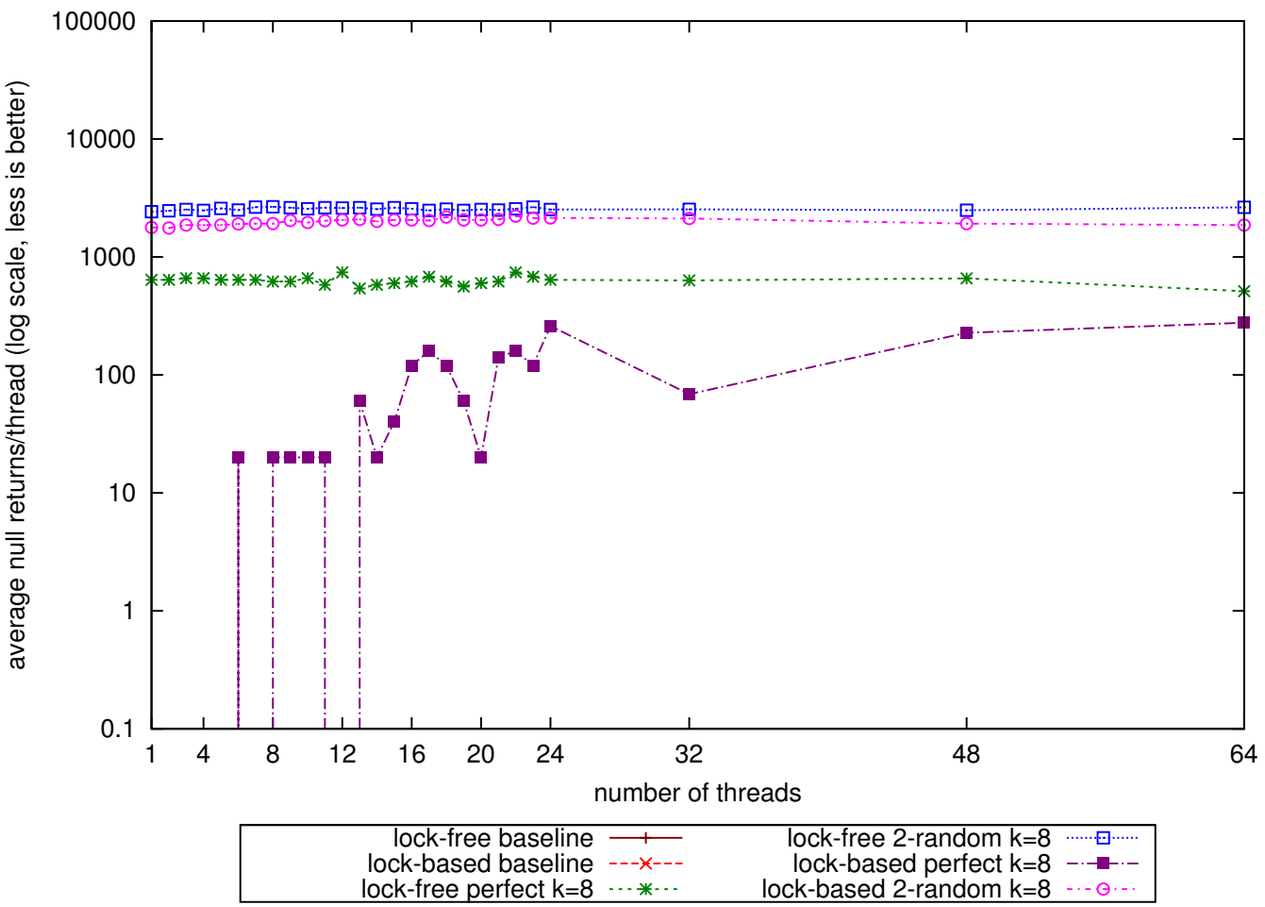


Performance

Semantics

High Computational Load

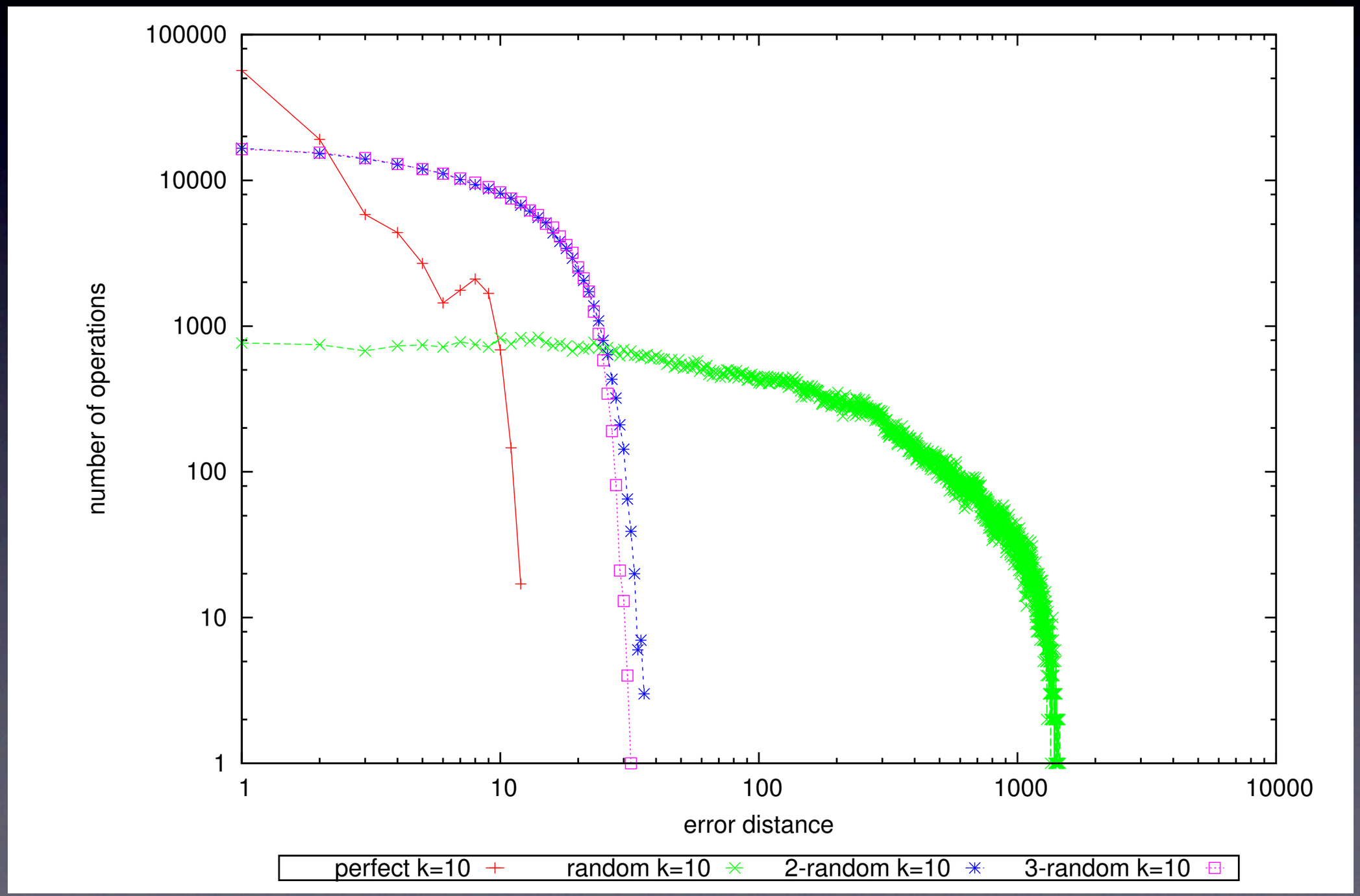




Performance

Semantics

Semantics for k=10



Precise Backoff

Listing 2: Precise backoff algorithm

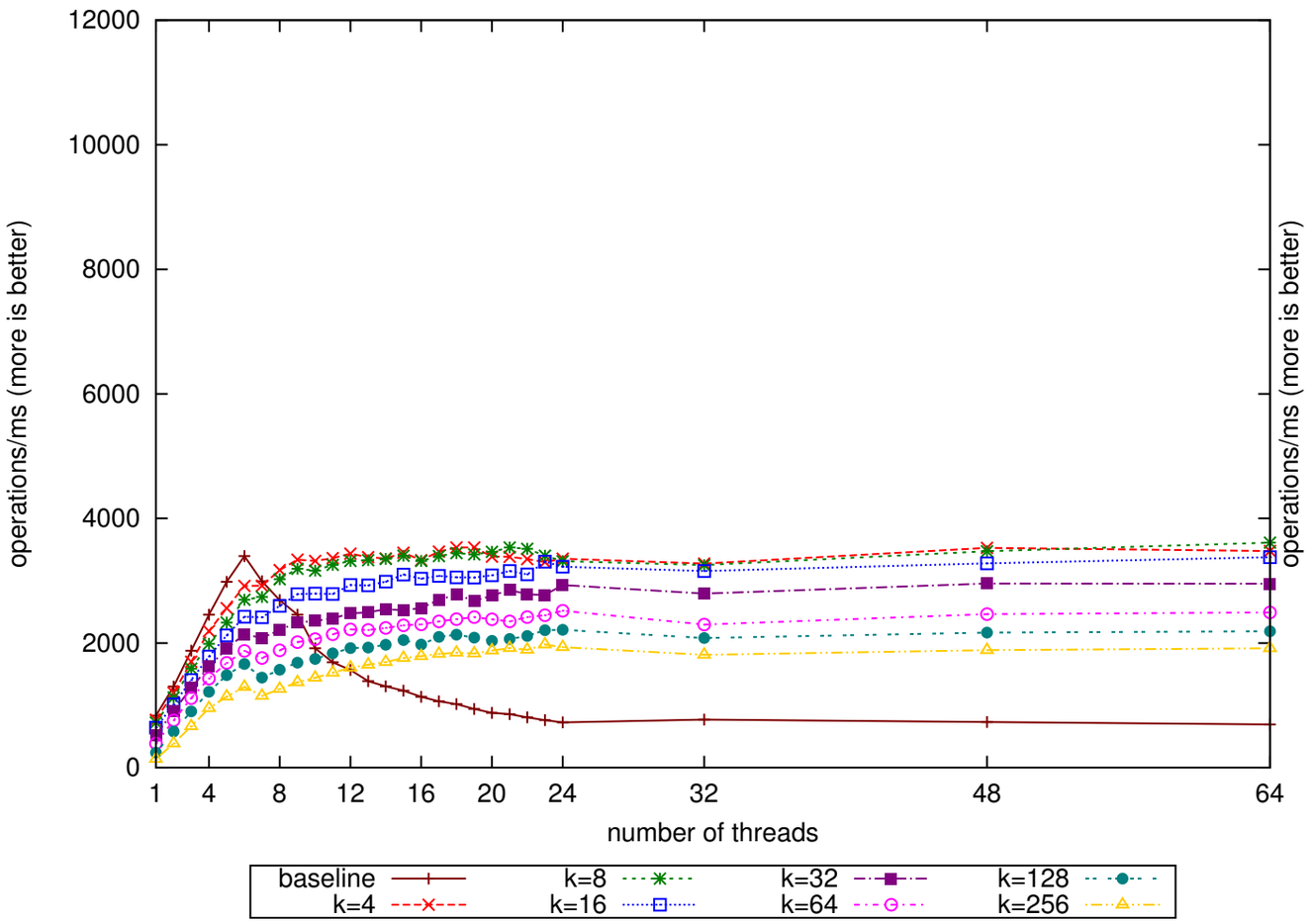
```
1 op(data_structure, parameters) {
2   do {
3     partial_ds = select(data_structure);
4     elem = partial_op(partial_ds, parameters);
5     if (test(elem)) {
6       update(counter, parameters);
7       return elem;
8     }
9   } while (!complete(counter, parameters));
10
11  return null;
12 }
```

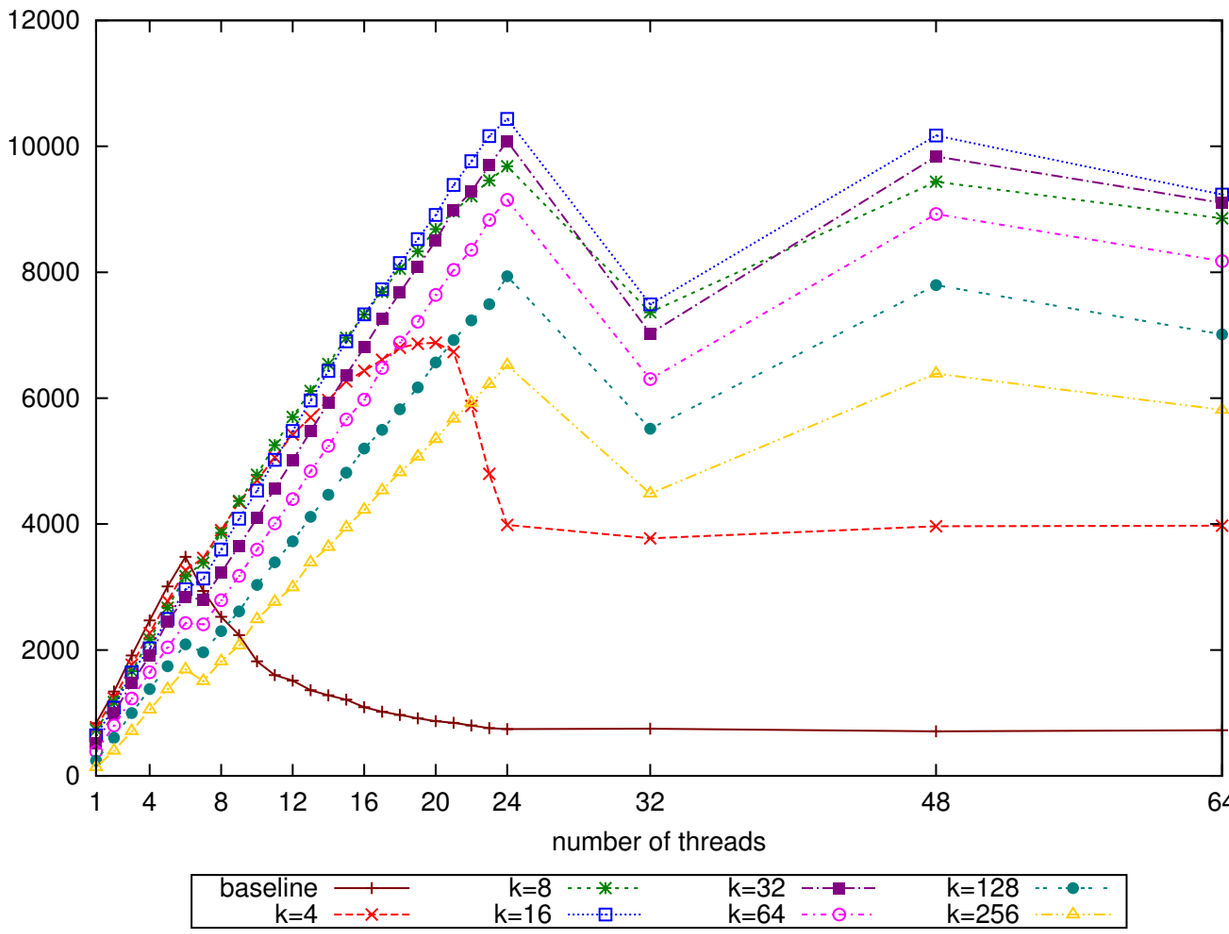
Heuristic Backoff

Listing 3: Heuristic backoff algorithm

```
1 op(data_structure, parameters) {
2   checks = MAX_CHECKS;
3   while (checks != 0) {
4     partial_ds = select(data_structure);
5     elem = partial_op(partial_ds, parameters);
6     if (test(elem))
7       return elem;
8     else if (checks == 0)
9       return null;
10    checks--;
11  }
12 }
```

Medium Computational Load (Backoff)





Precise Backoff

Heuristic Backoff

Thank you

Check out:
eurosys2011.cs.uni-salzburg.at