ONE DOES NOT SIMPLY
WALK INTO A CONCURRENT HASH TABLE
memegenerator.net

Catching up with Cuckoo

Bin Fan, Xiaozhou Li, Michael Kaminsky, Mike Freedman, David G. Andersen
CMU, Intel, Princeton

Hashing - it’s useful!

CS 101 version

- map[“dog”] = 5
- print map[“dog”] —> 5

Hashing - it’s fun!

CS 201 version

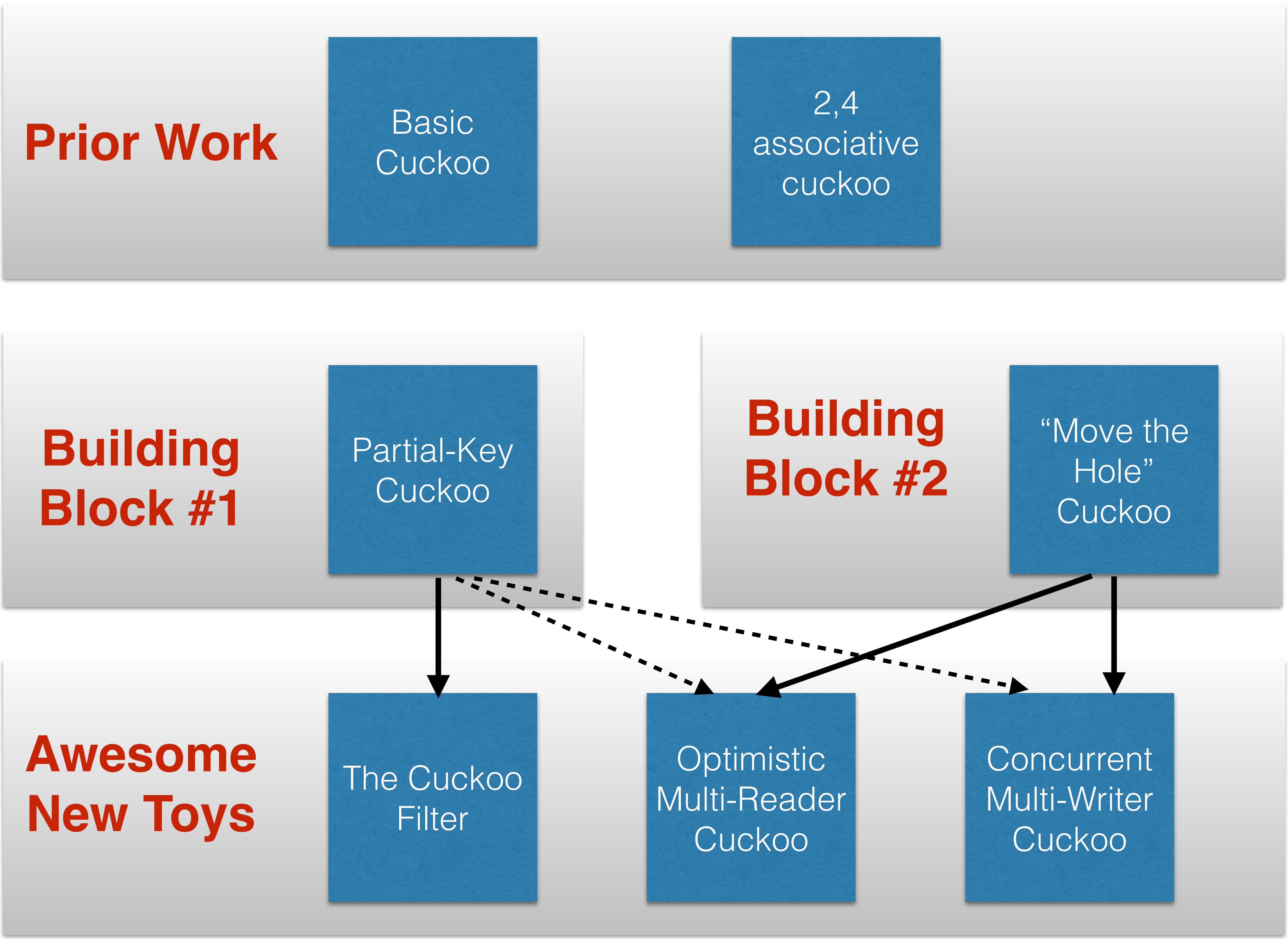
- O(1) insert / lookup / delete
- Linear probing
- Chaining

Standard methods: Either slow, non-concurrent, or waste memory

Hashing - it's cool again!

Grad school

- Cuckoo Hashing
  - Seriously memory efficient
  - But before our work, **slow in practice, non-concurrent (or inefficient)**



Prior Work

Basic Cuckoo

2,4 associative cuckoo

Building Block #1

Partial-Key Cuckoo

Building Block #2

“Move the Hole” Cuckoo

Awesome New Toys

The Cuckoo Filter

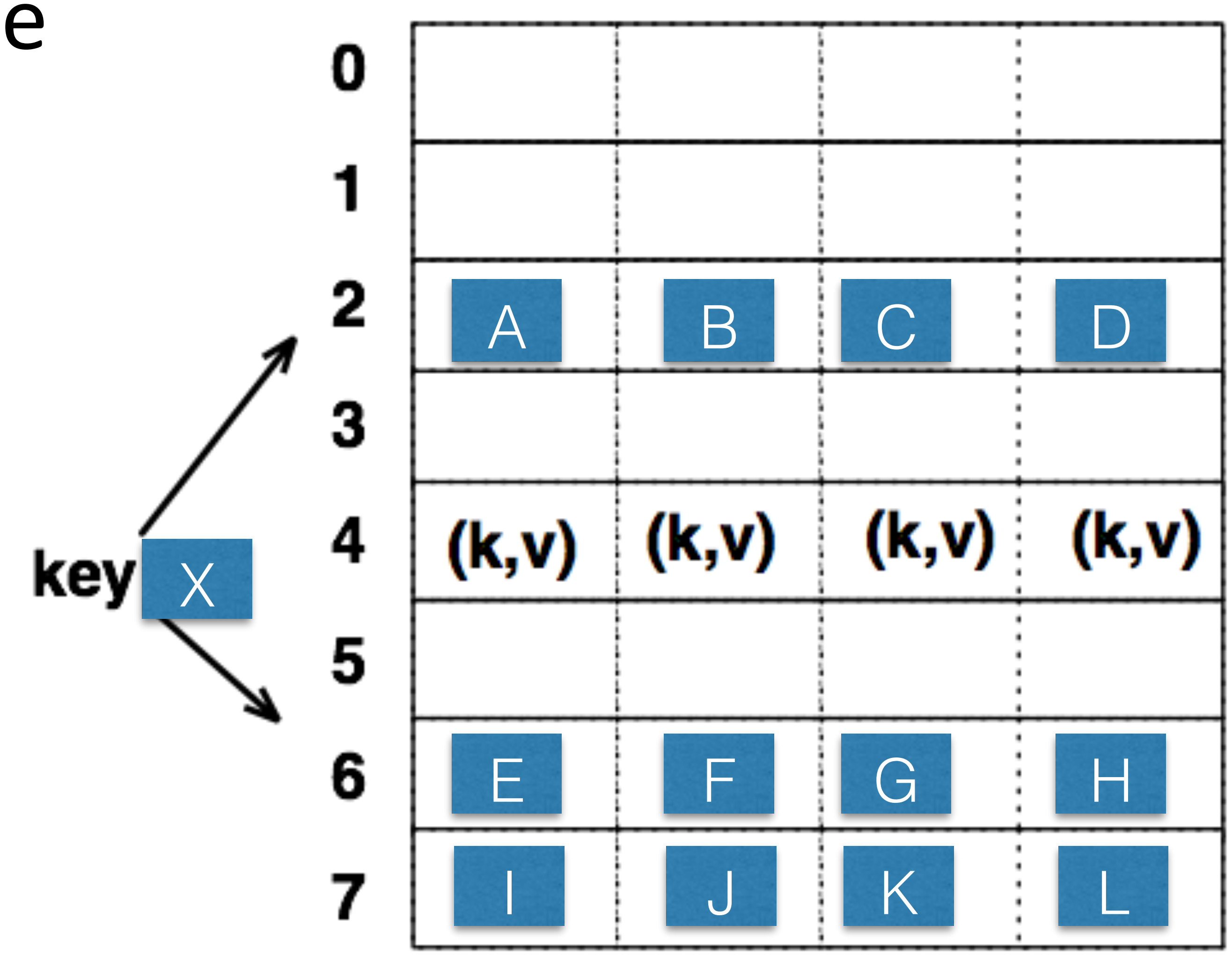
Optimistic Multi-Reader Cuckoo

Concurrent Multi-Writer Cuckoo

Background

Cuckoo Hashing

- Hash item to two possible buckets
  H1(key) —> bucket 1
  H2(key) —> bucket 2



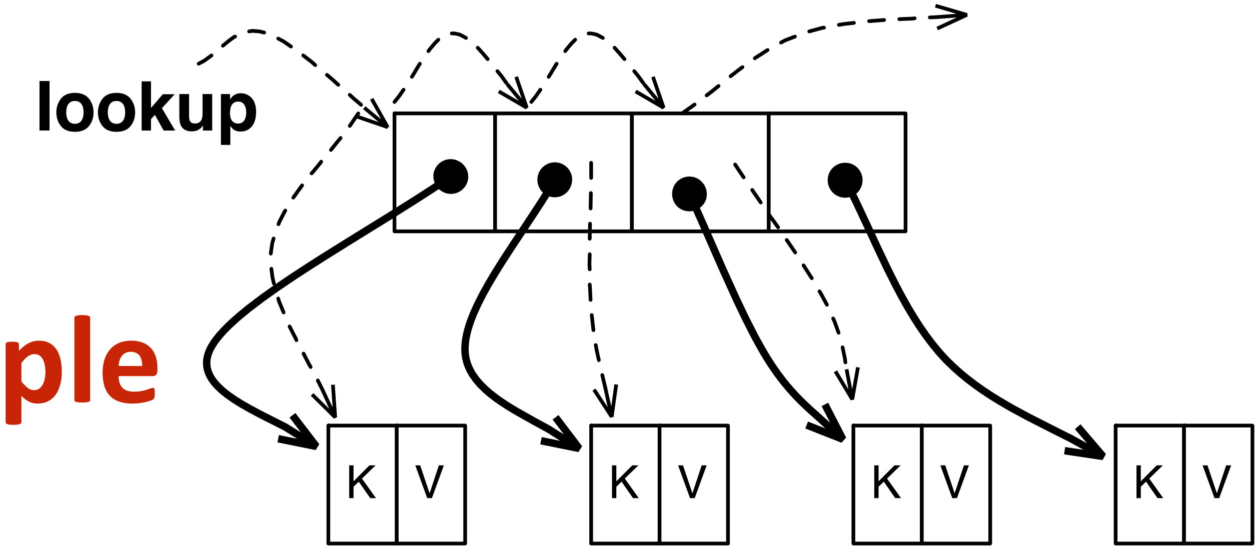
What if keys are not stored in table?

Challenge

Expensive Key Retrieval

- Why not store key-value in table?
  - support variable-len keys
  - to store key-value in external storage

- **Lookup requires multiple retrievals for key comparison**

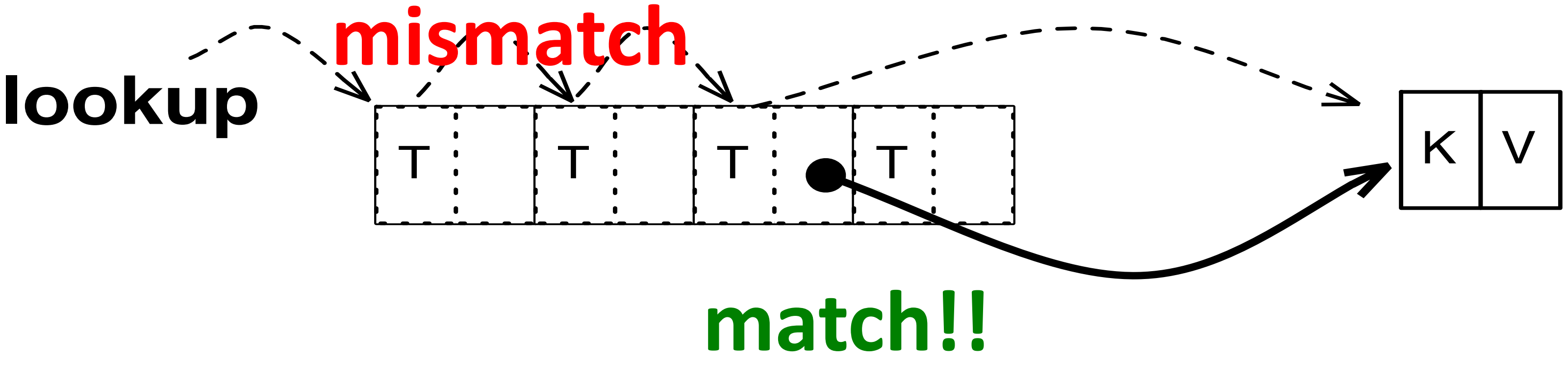


Solution

Partial-key Cuckoo Hashing

- Definition: a **tag**
  - a small hash value, 1 Byte in our implementation
  - `tag("foo") = 0x3f`

- Store tags in table to reduce false retrievals
  - Read K-V only on tag match



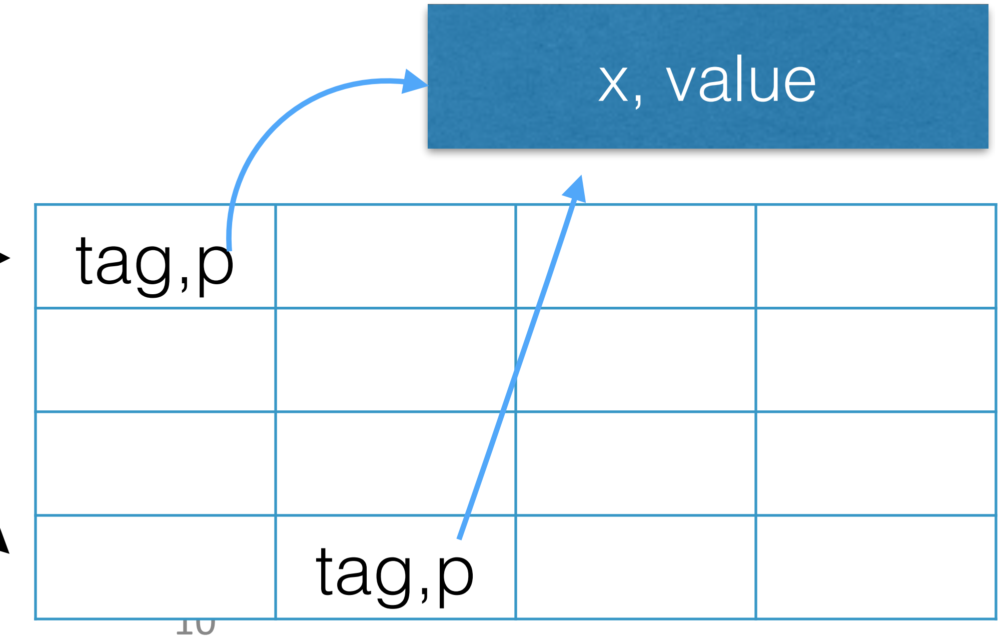
Solution

Cuckoo Move without *Pointer

use current location to compute alternate

```
b1 = HASH(tag(x))            // 1st bucket
b2 = b1 ⊕ HASH(tag(x))       // 2nd bucket
```

```
alt = cur ⊕ HASH(tag(x))
```



Building Block #1: Partial-key cuckoo hashing

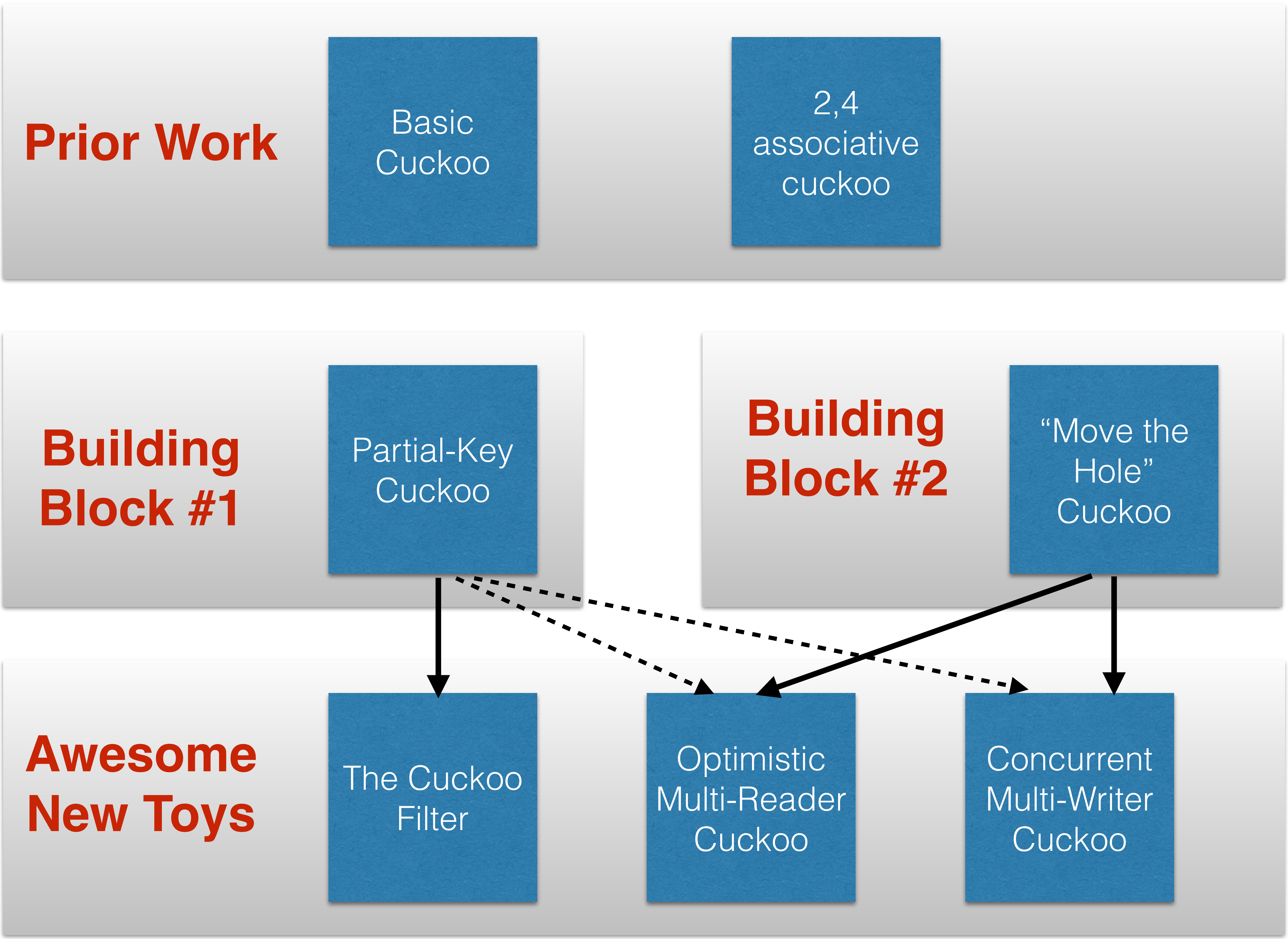
Benefits:

✓ **Compact, fixed-sized fields in hash table**

✓ **Only 1+ε pointer dereferences for lookup**

✓ **No pointer dereference needed for cuckooing**

Hey, Dave - haven’t you heard of multicore?



Prior Work

Basic Cuckoo

2,4 associative cuckoo

Building Block #1

Partial-Key Cuckoo

Building Block #2

“Move the Hole” Cuckoo

Awesome New Toys

The Cuckoo Filter

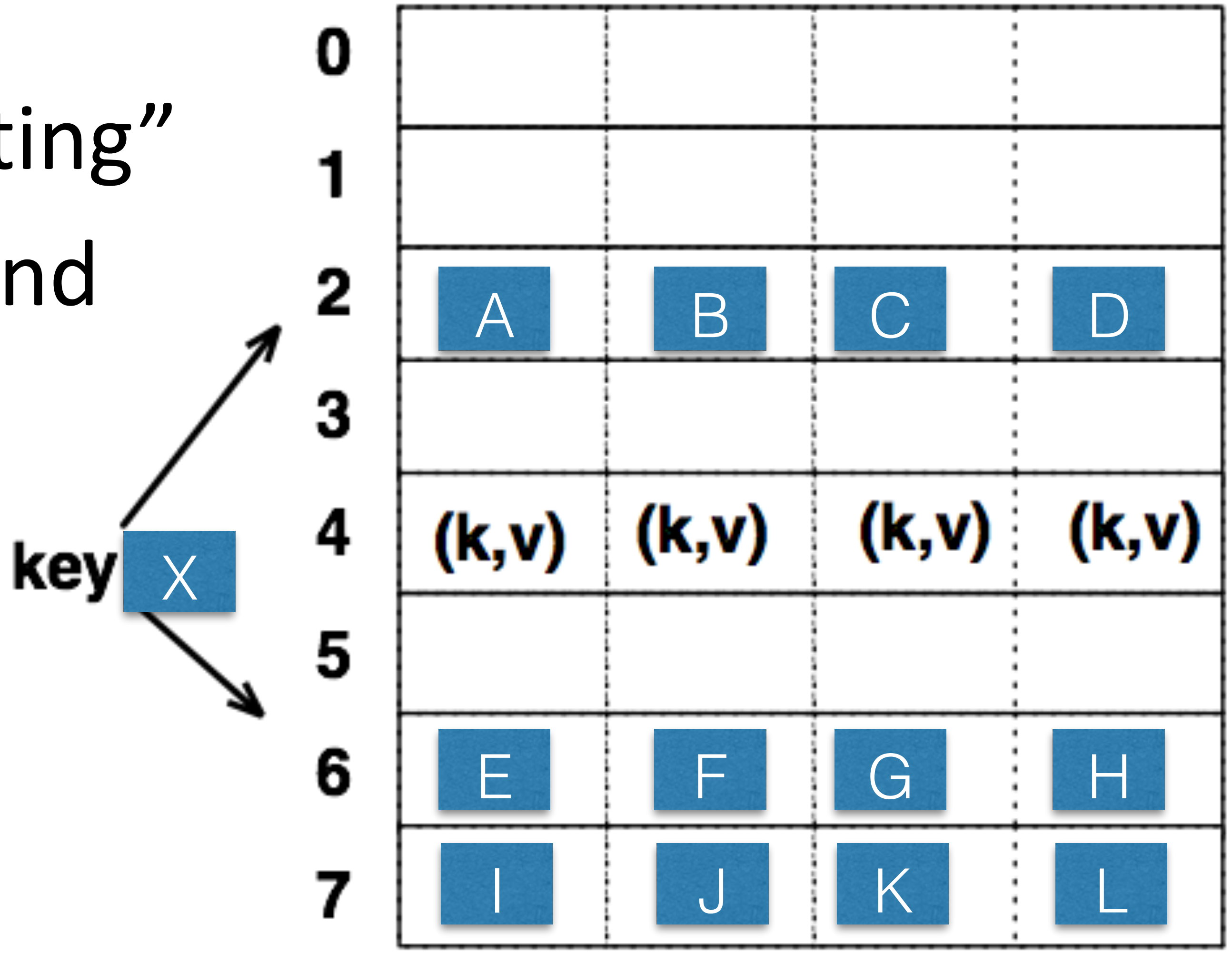
Optimistic Multi-Reader Cuckoo

Concurrent Multi-Writer Cuckoo

Why “Move The Hole”?

- During insertion…
  - one key is always “floating”
  - That key cannot be found by get()

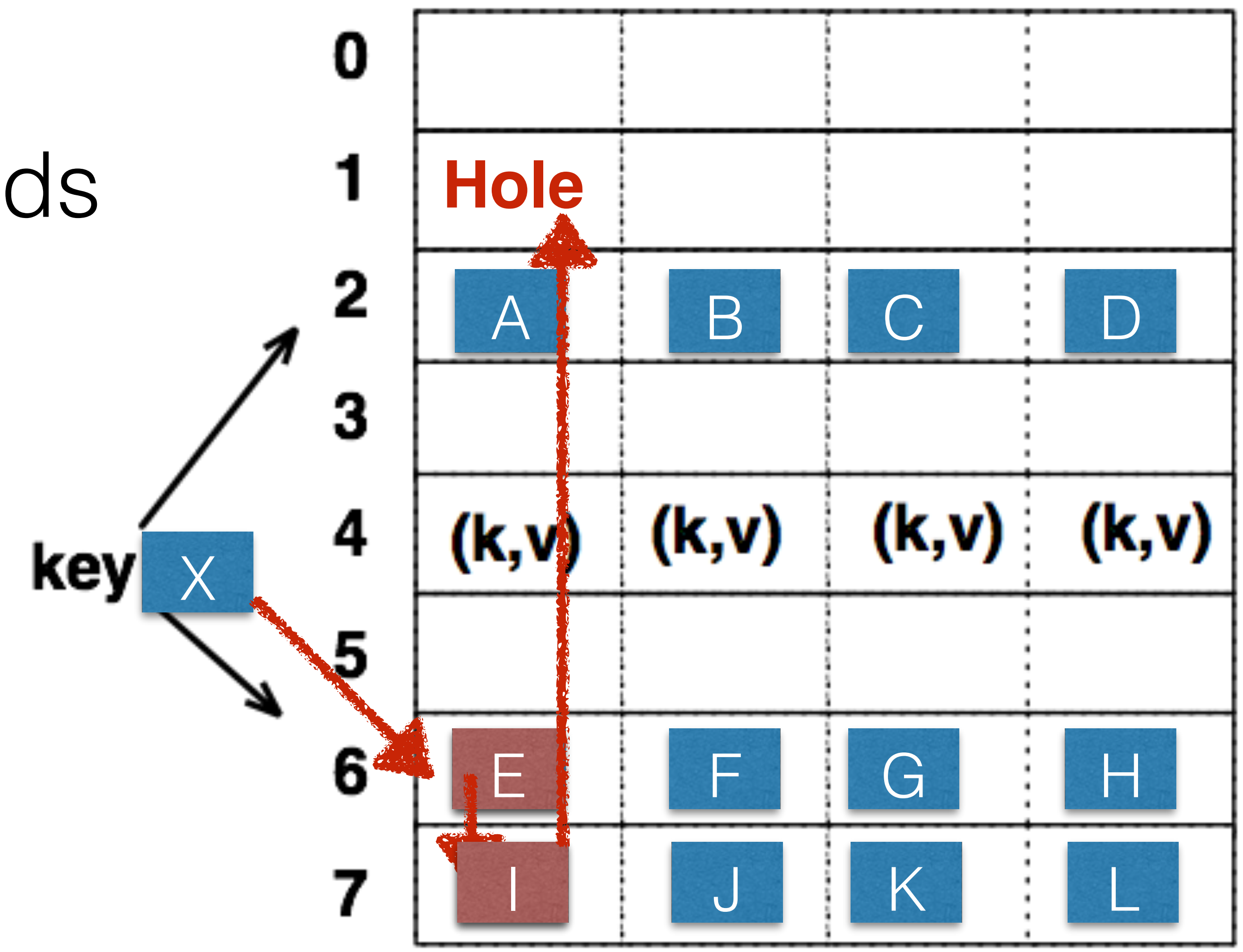
- Prior solutions caused previous concurrent cuckoo tables to waste space [Herlihy]



Move The Hole: Find Path First

Then move hole backwards

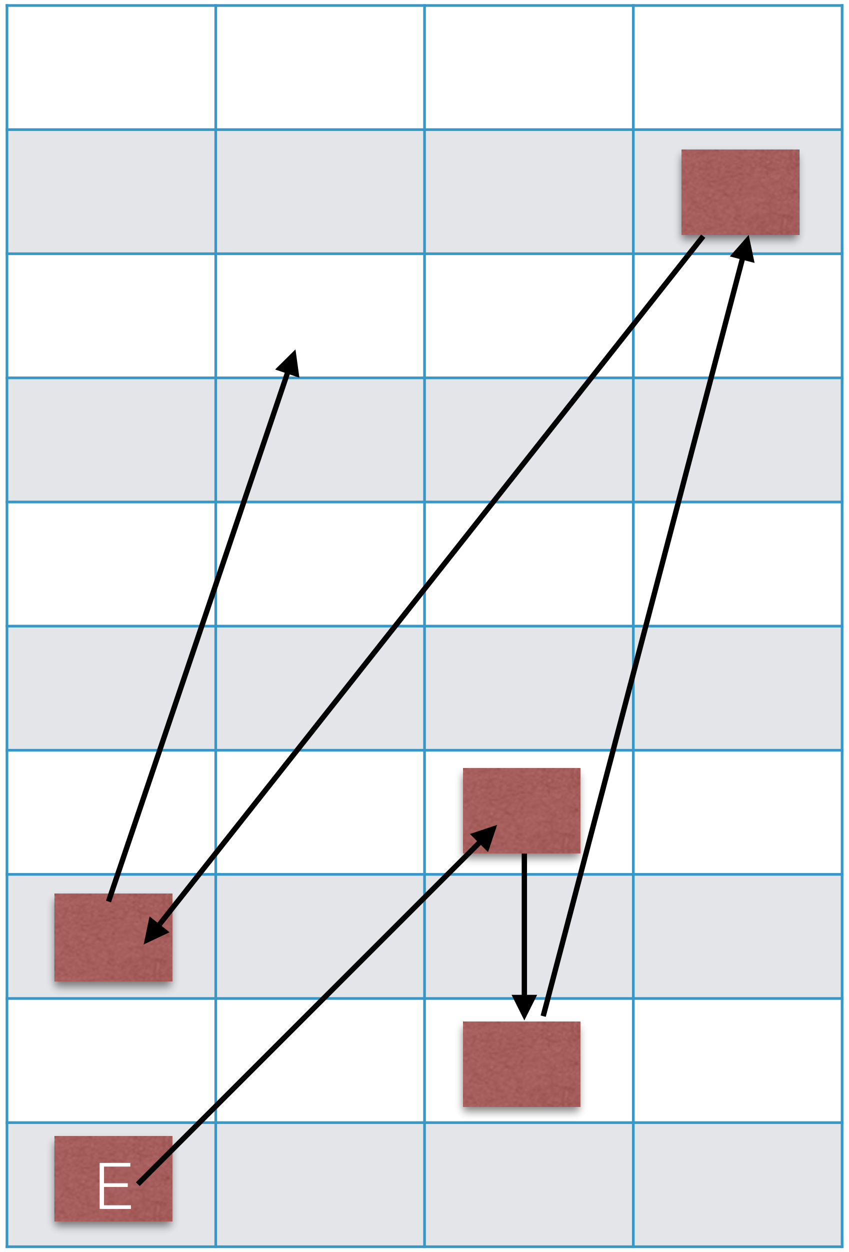
- ✓ **Items never disappear**
- ✓ **Only individual swaps must be atomic**



Full Tables: Lots of Motion

Up to 500 moves needed at 95% occupancy!

Large potential for concurrency conflicts



Optimization Strategy

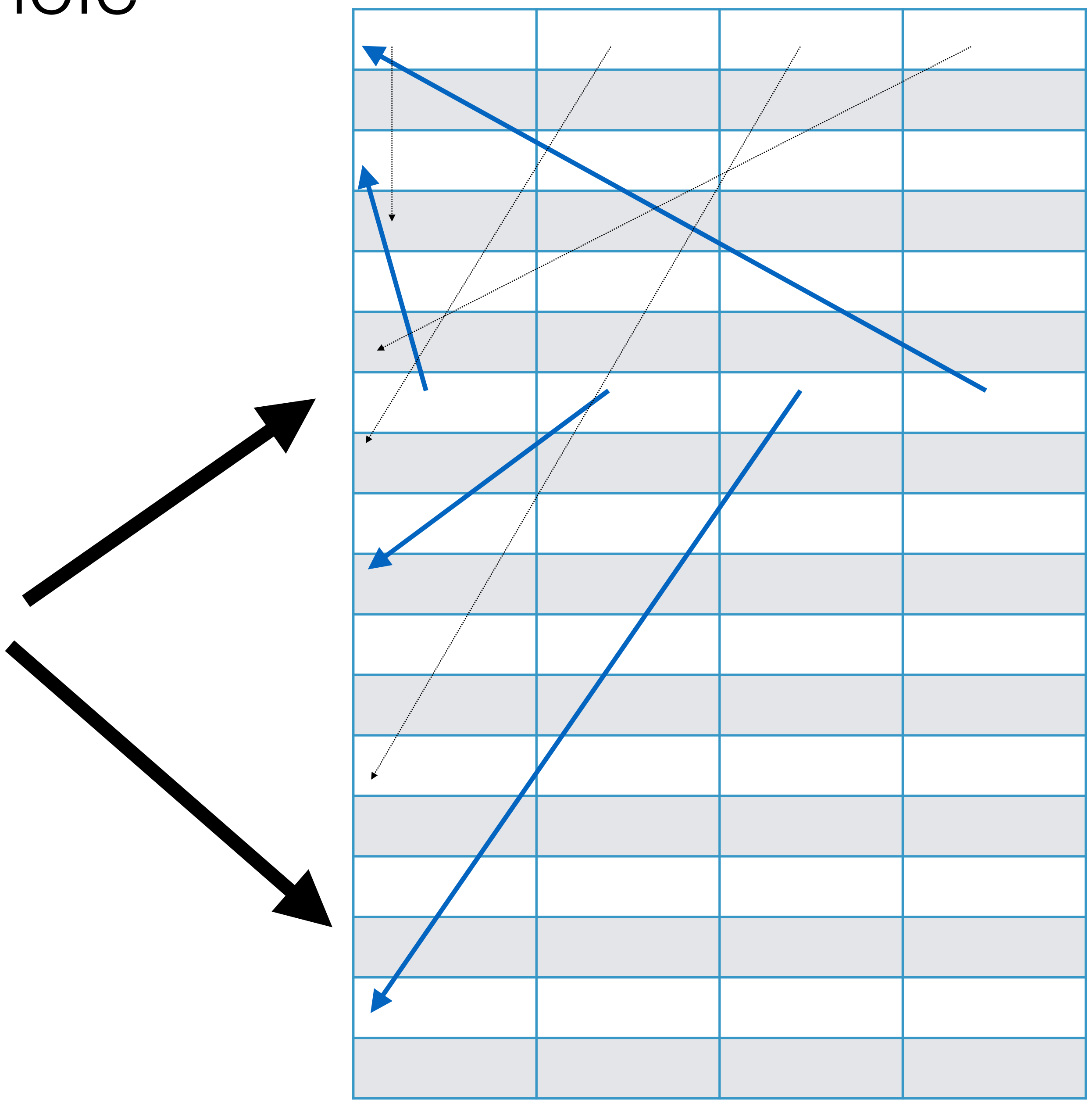
- Move work outside lock (**done: search first**)
- **Reduce number of moves needed**??

Breadth-first search for hole instead of depth-first

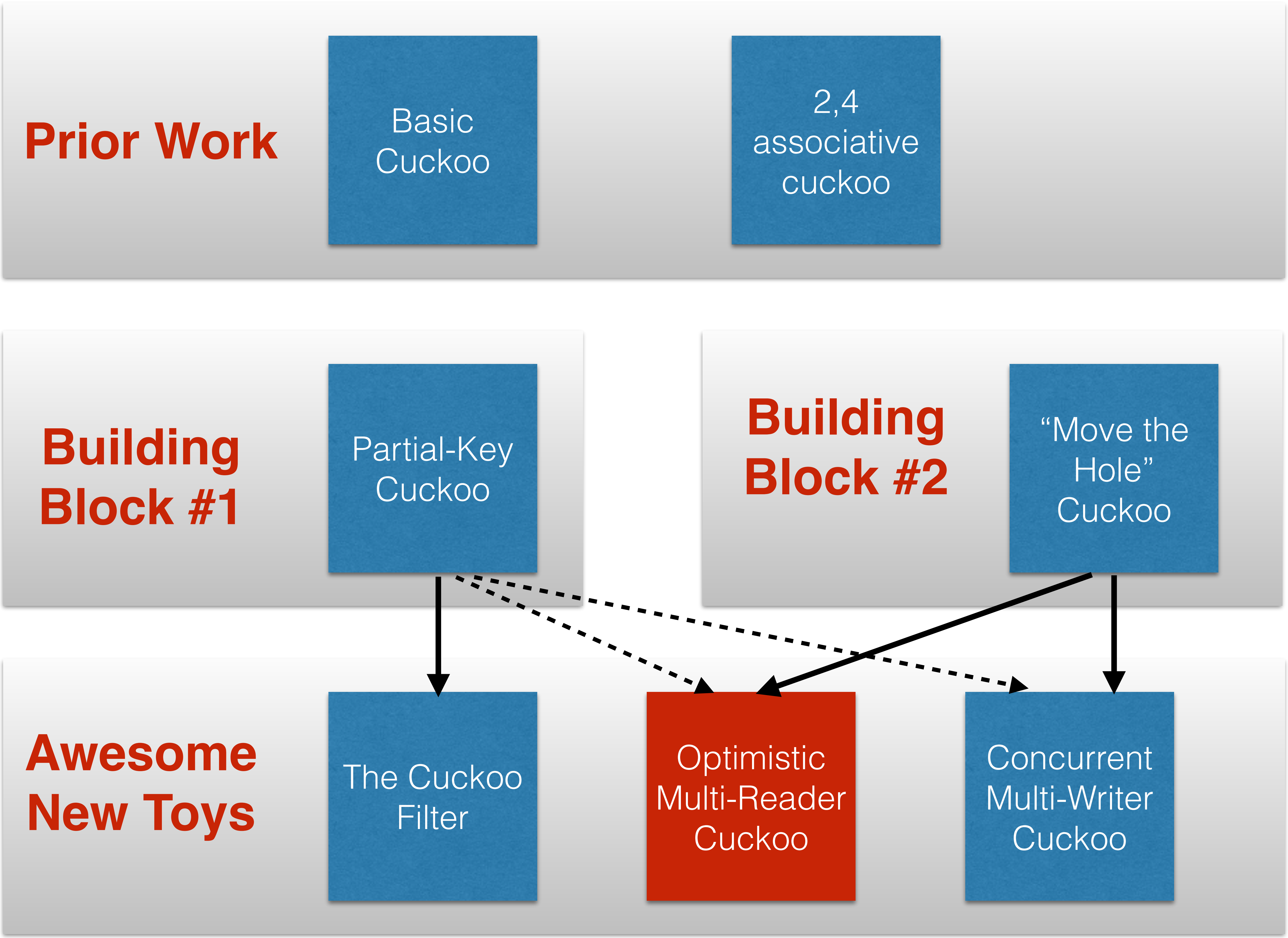
Same *search* work
Less *move* work:

~500 bins examined

~5 bins moved



Effective for locking, flash, NVRAM, …



Prior Work

Basic Cuckoo

2,4 associative cuckoo

Building Block #1

Partial-Key Cuckoo

Building Block #2

"Move the Hole" Cuckoo

Awesome New Toys

The Cuckoo Filter

Optimistic Multi-Reader Cuckoo

Concurrent Multi-Writer Cuckoo

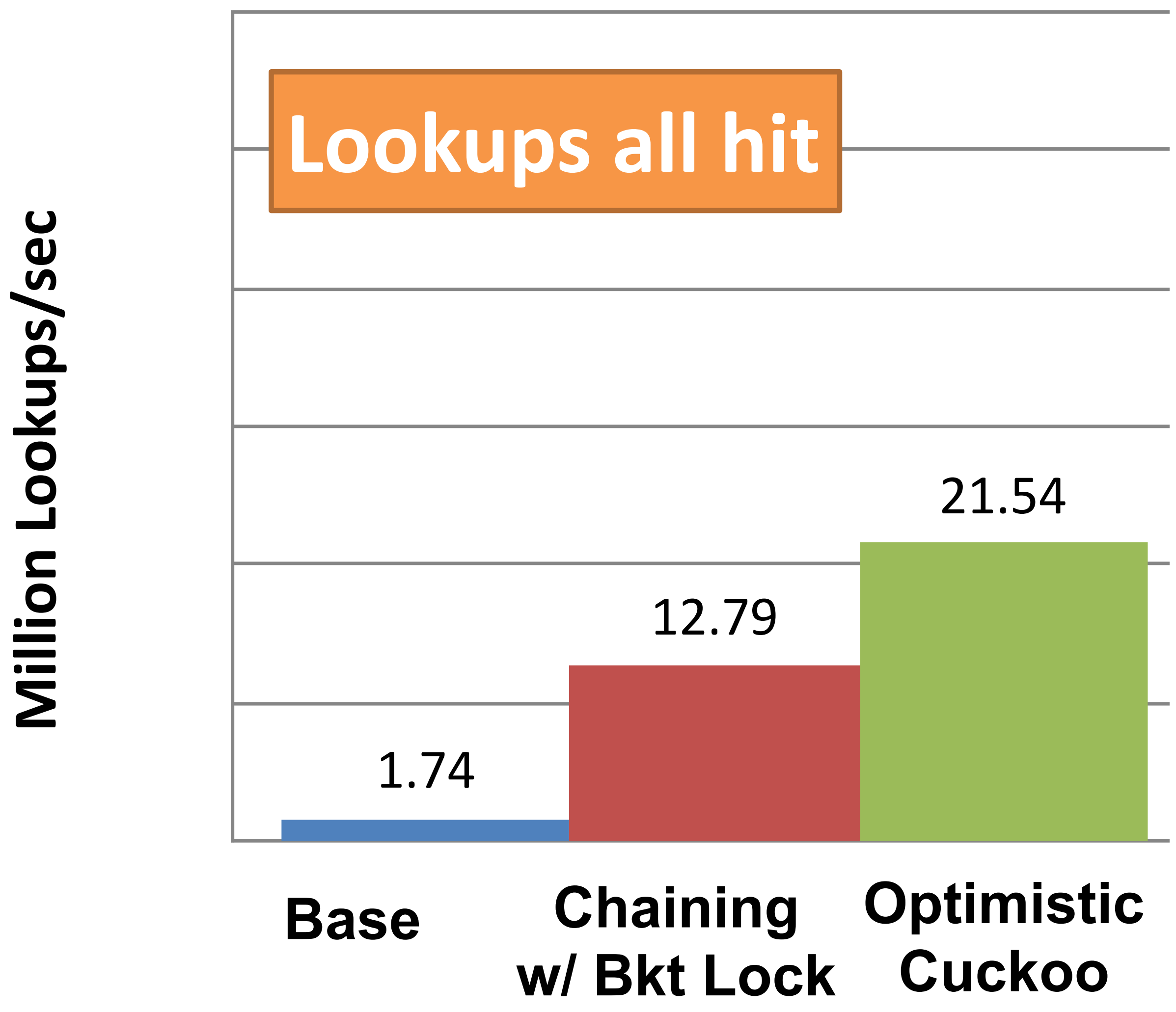
Historical Reminder

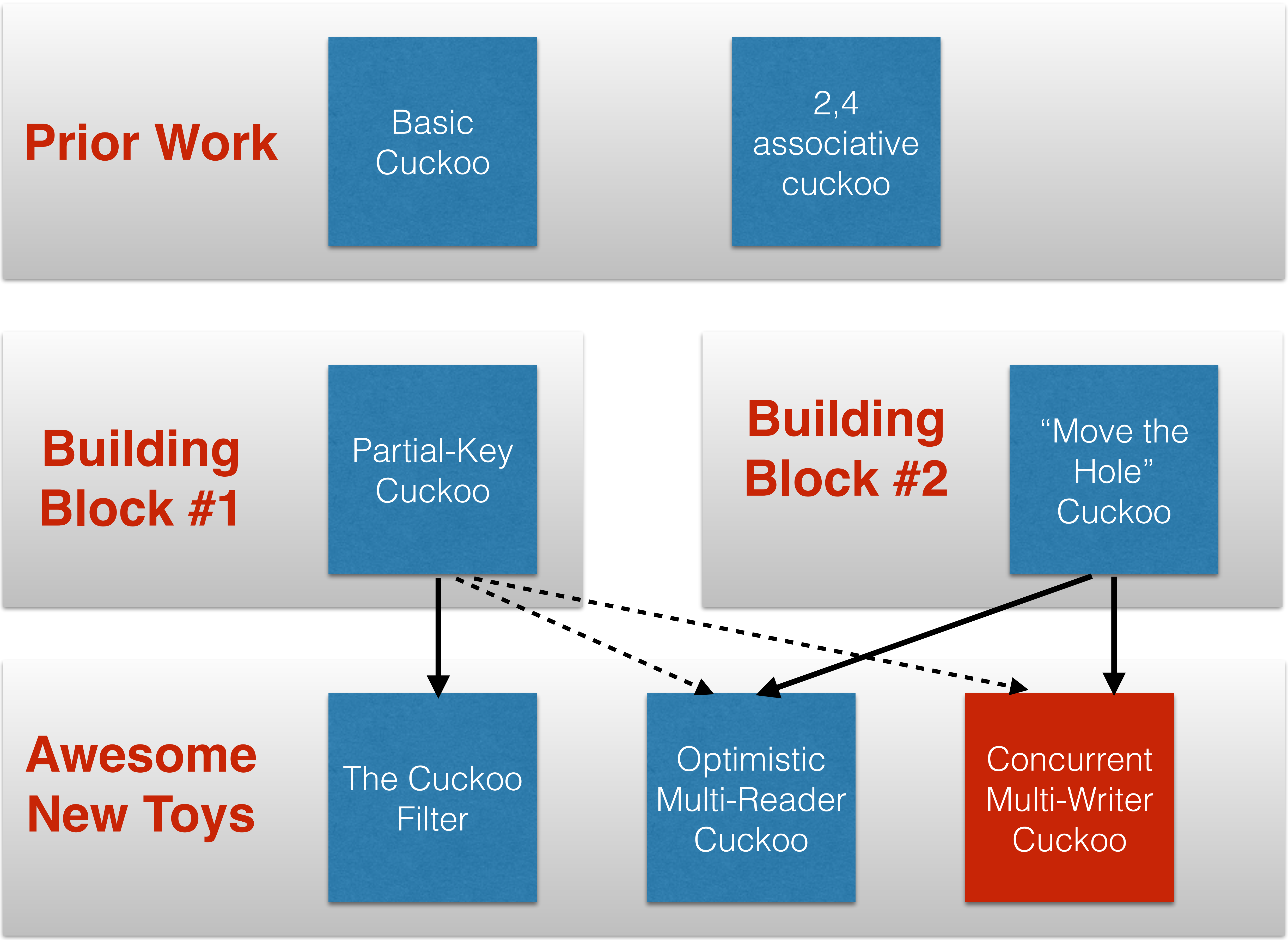
MemC3 nsdi2013
4.3M ops/sec over the network

**Single-writer;
optimized for ~95% reads (Facebook)**

Hash Table Microbenchmark

Low Power Xeon CPU (12 cores), 12 MB L3 cache
6 threads reading a ~ 1GB hash table





Prior Work

Basic Cuckoo

2,4 associative cuckoo

Building Block #1

Partial-Key Cuckoo

Building Block #2

“Move the Hole” Cuckoo

Awesome New Toys

The Cuckoo Filter

Optimistic Multi-Reader Cuckoo

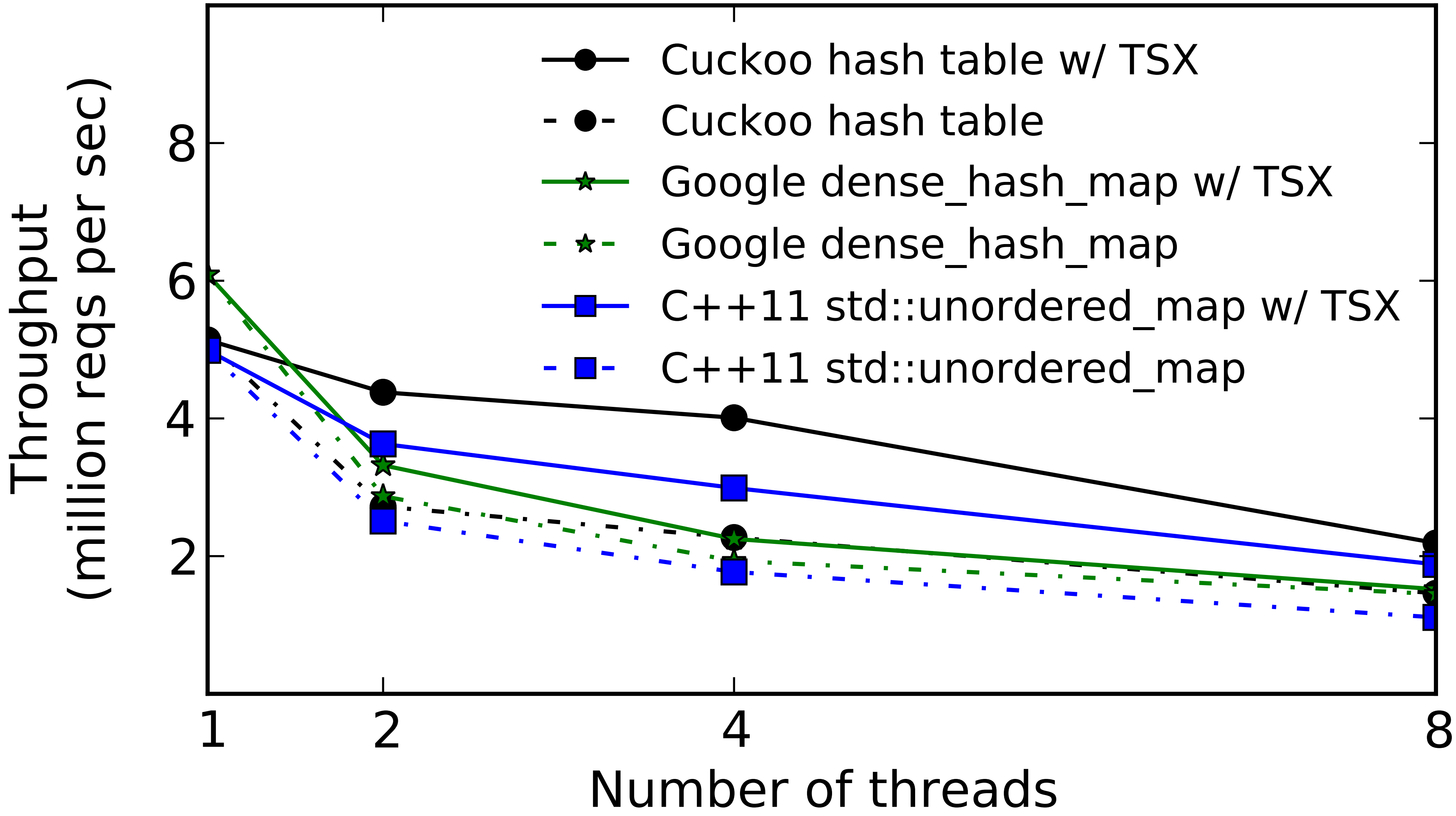
Concurrent Multi-Writer Cuckoo

Interlude: Hardware Transactional Memory

- We made two versions of Concurrent Cuckoo:
  - Fine-grained conventional spinlocks
  - One that used Intel’s new Hardware Transactional Memory (TSX)
- They perform similarly. I’ll show the TSX results.

Without Concurrency Optimization

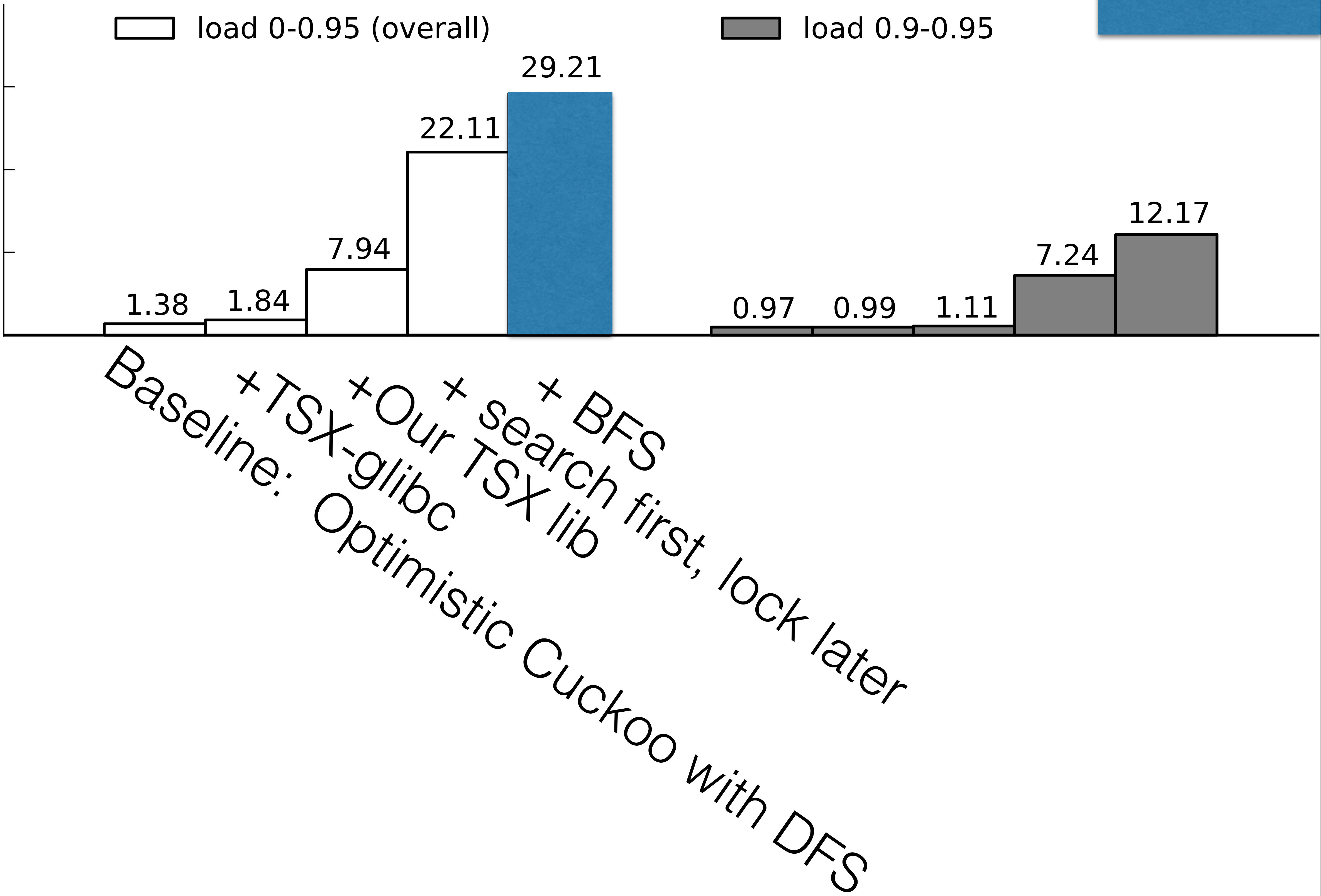
4 core
Haswell
Desktop



Total throughput drops with more threads

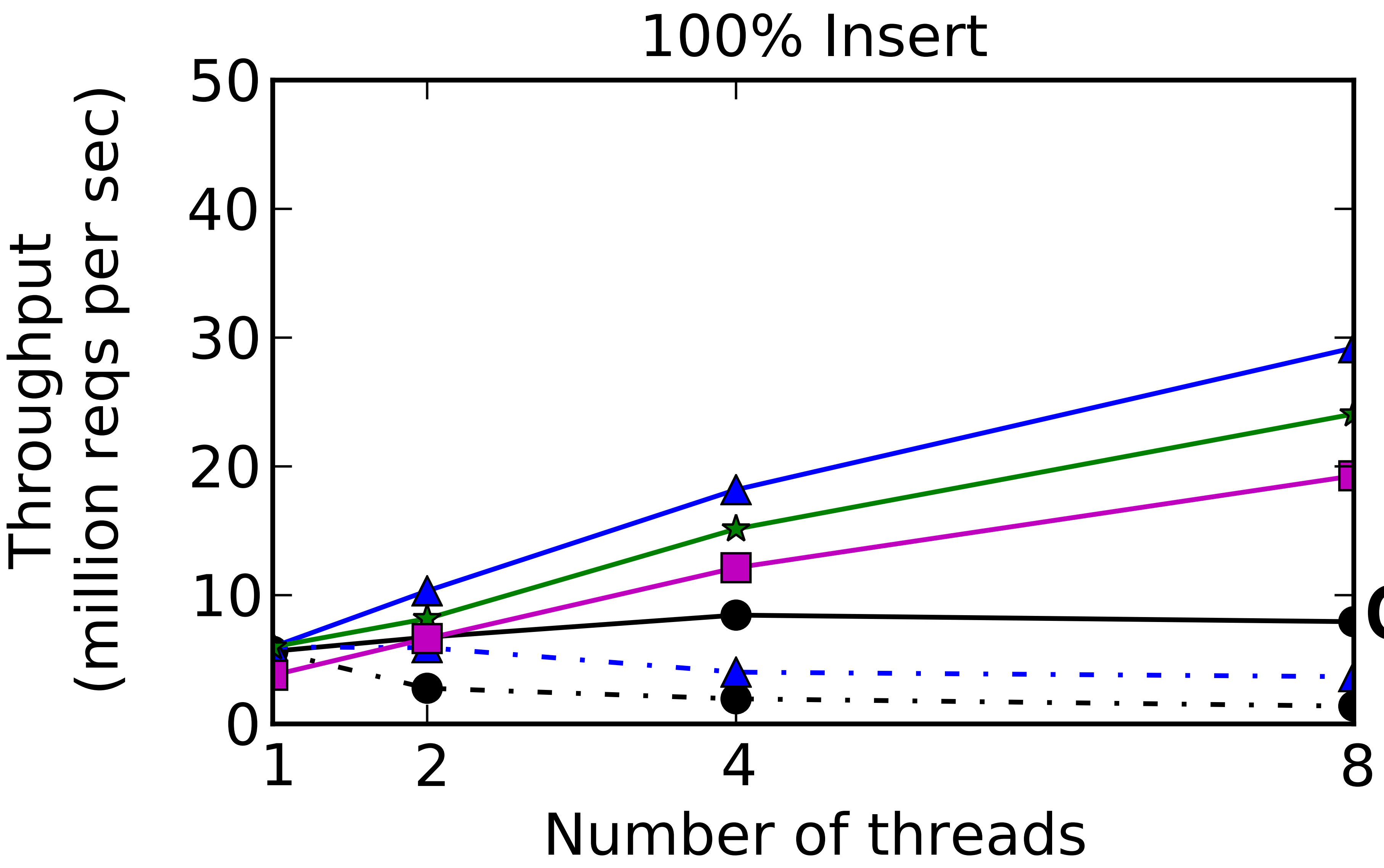
Optimizations: 8 thread throughput

4 core Haswell Desktop



vs. The Competition

4 core Haswell Desktop

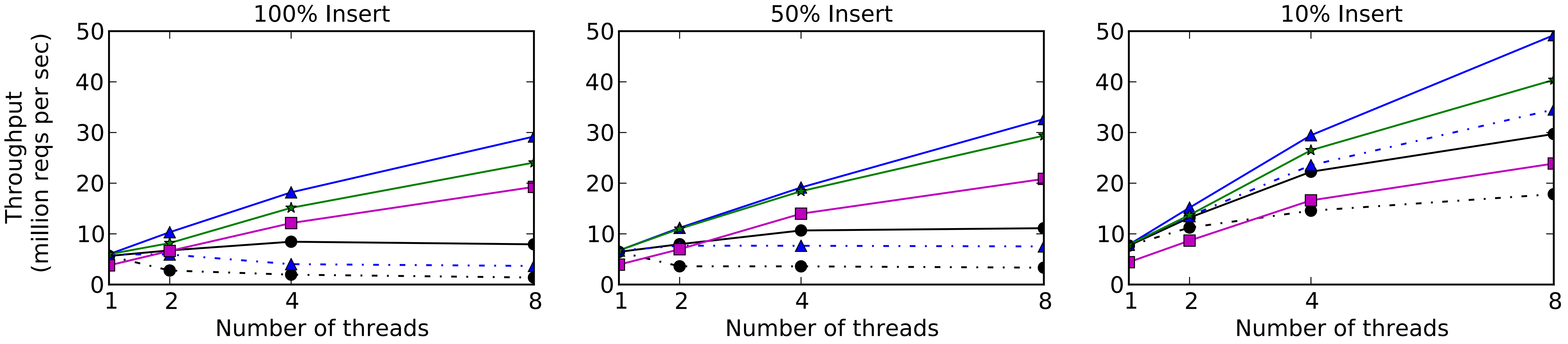


Cuckoo-TSX

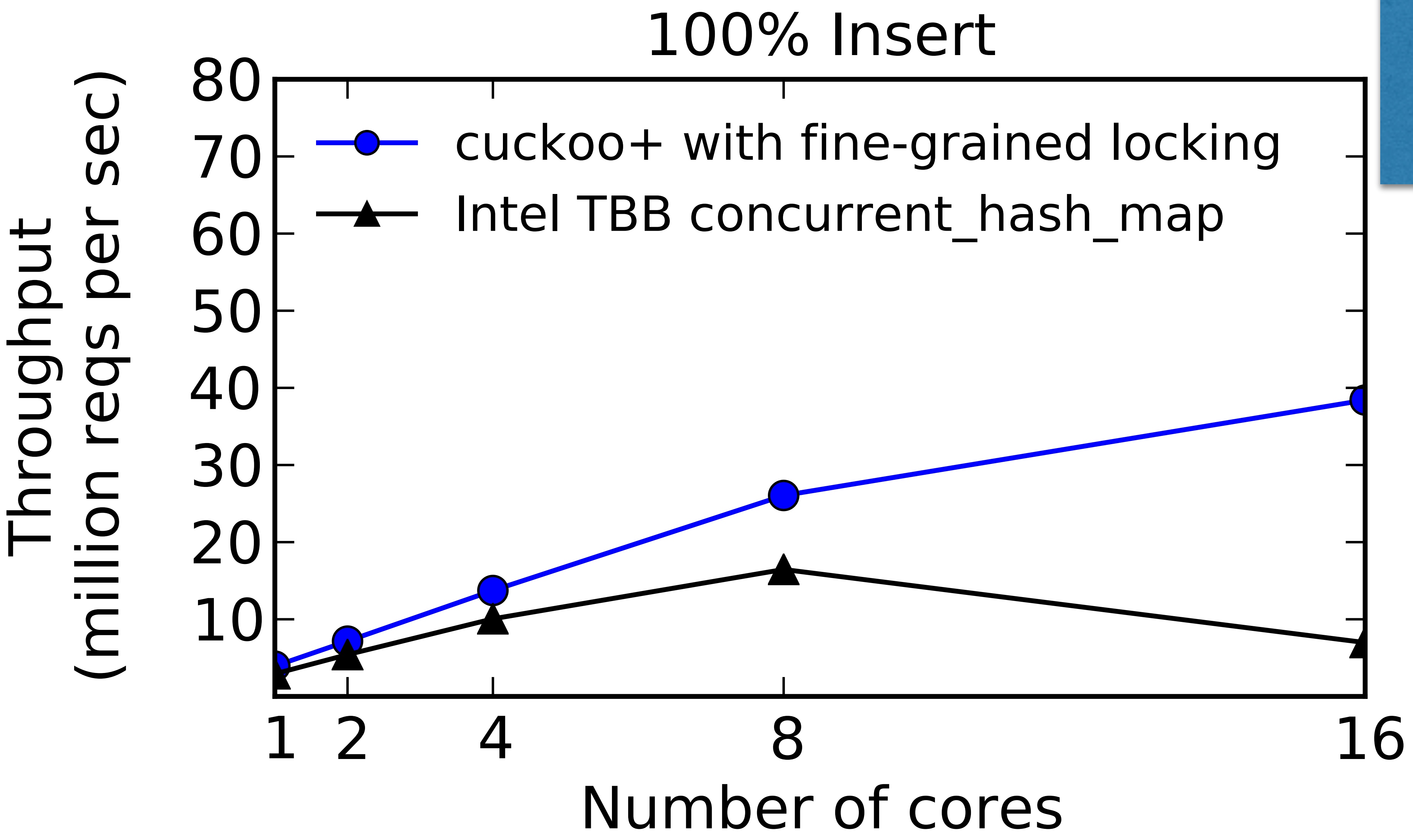
Cuckoo-Spinlock

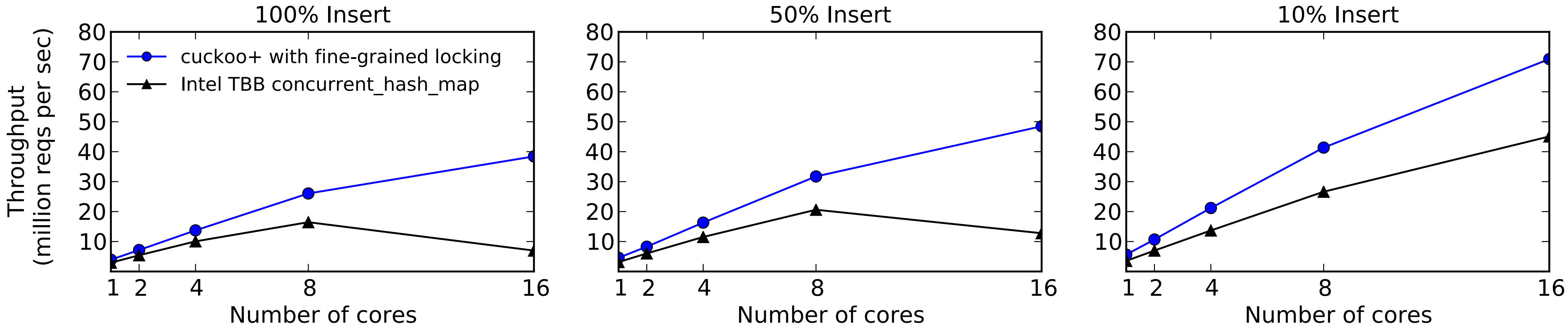
TBB hash_map

Cuckoo-opt-global

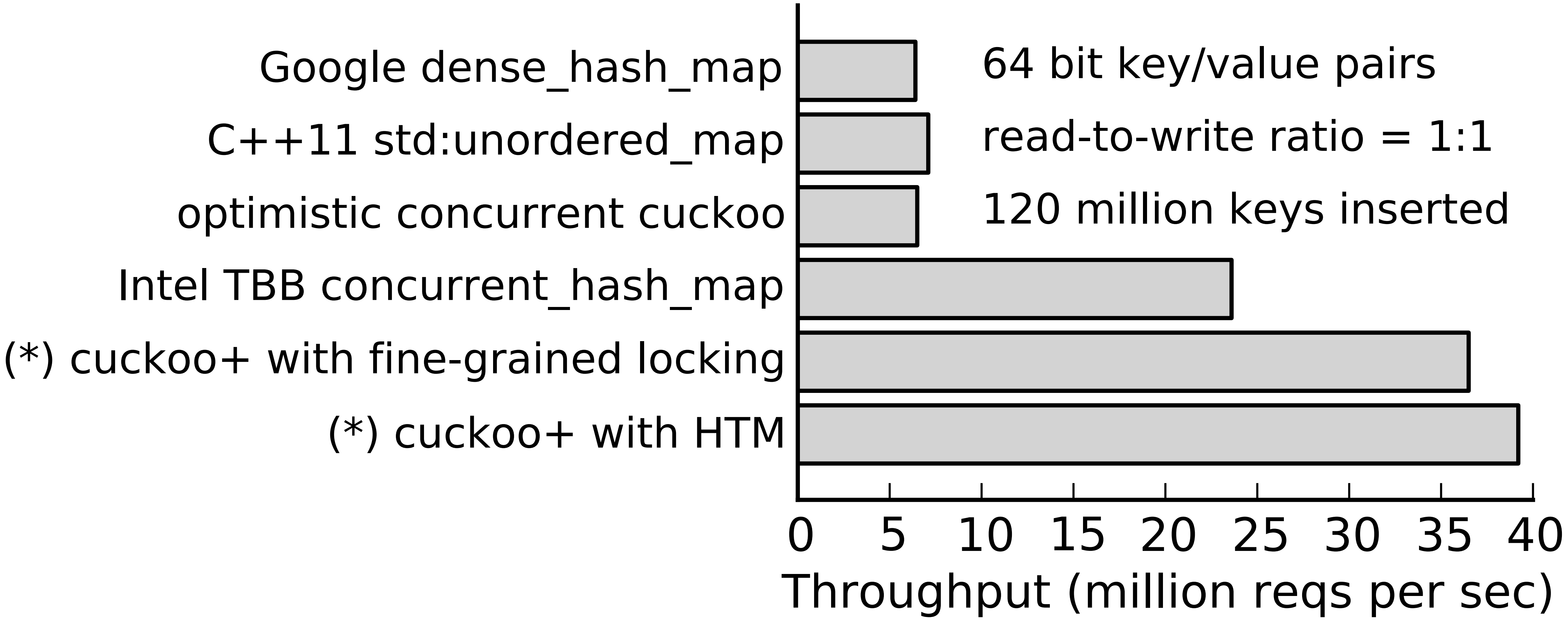


16 Core
Xeon


100% Insert
cuckoo+ with fine-grained locking
Intel TBB concurrent_hash_map
Throughput
(million reqs per sec)
80
70
60
50
40
30
20
10
1
2
4
8
16
Number of cores

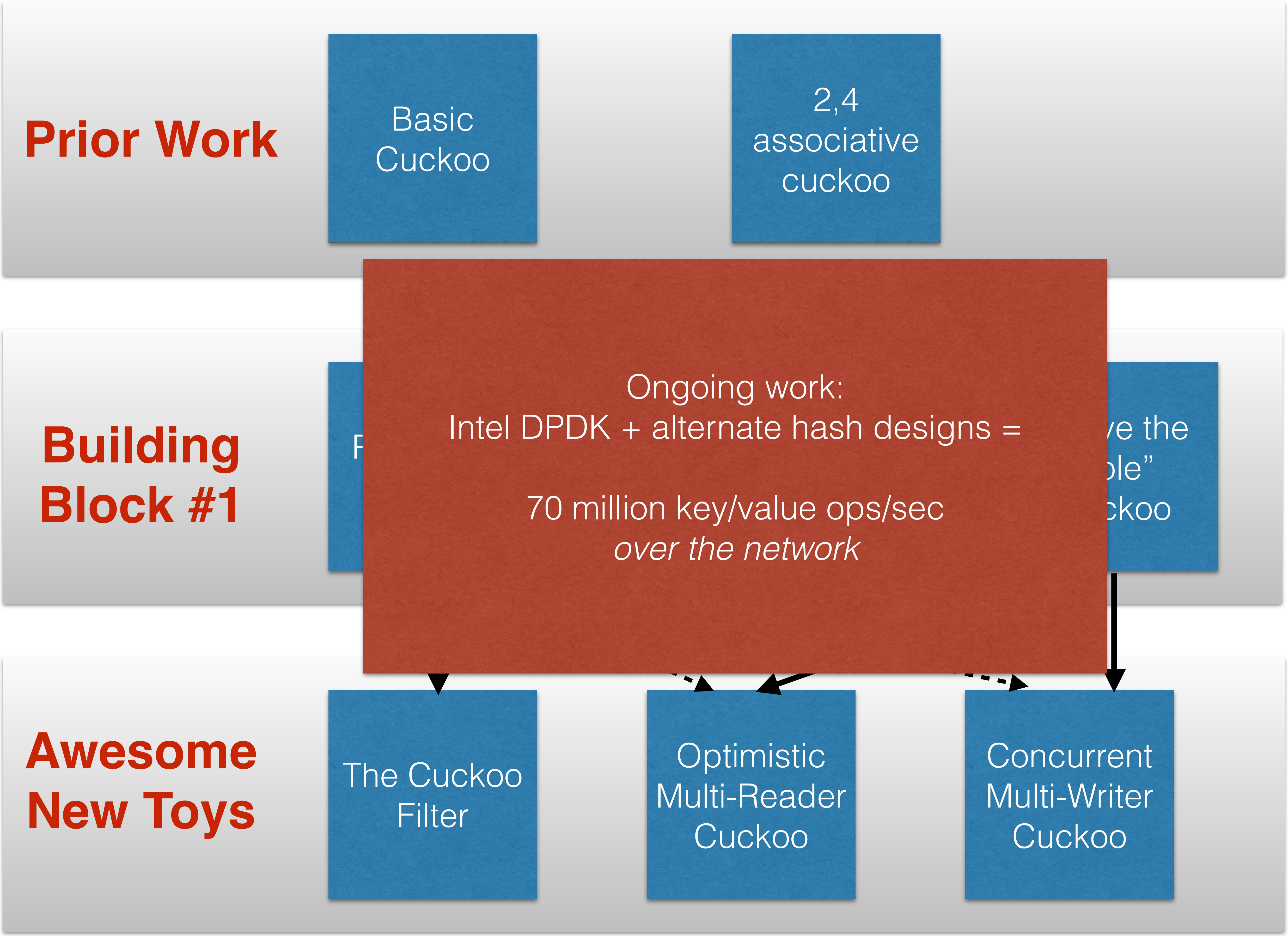

100% Insert
cuckoo+ with fine-grained locking
Intel TBB concurrent_hash_map
Throughput
(million reqs per sec)
80
70
60
50
40
30
20
10
1
2
4
8
16
Number of cores
50% Insert
80
70
60
50
40
30
20
10
1
2
4
8
16
Number of cores
10% Insert
80
70
60
50
40
30
20
10
1
2
4
8
16
Number of cores

Concurrent beats non-concurrent



Concurrent Cuckoo

A really tasty memory-efficient,
concurrent hash table



Prior Work

Basic Cuckoo

2,4 associative cuckoo

Building Block #1

F

ve the
ble”
ckoo

Awesome New Toys

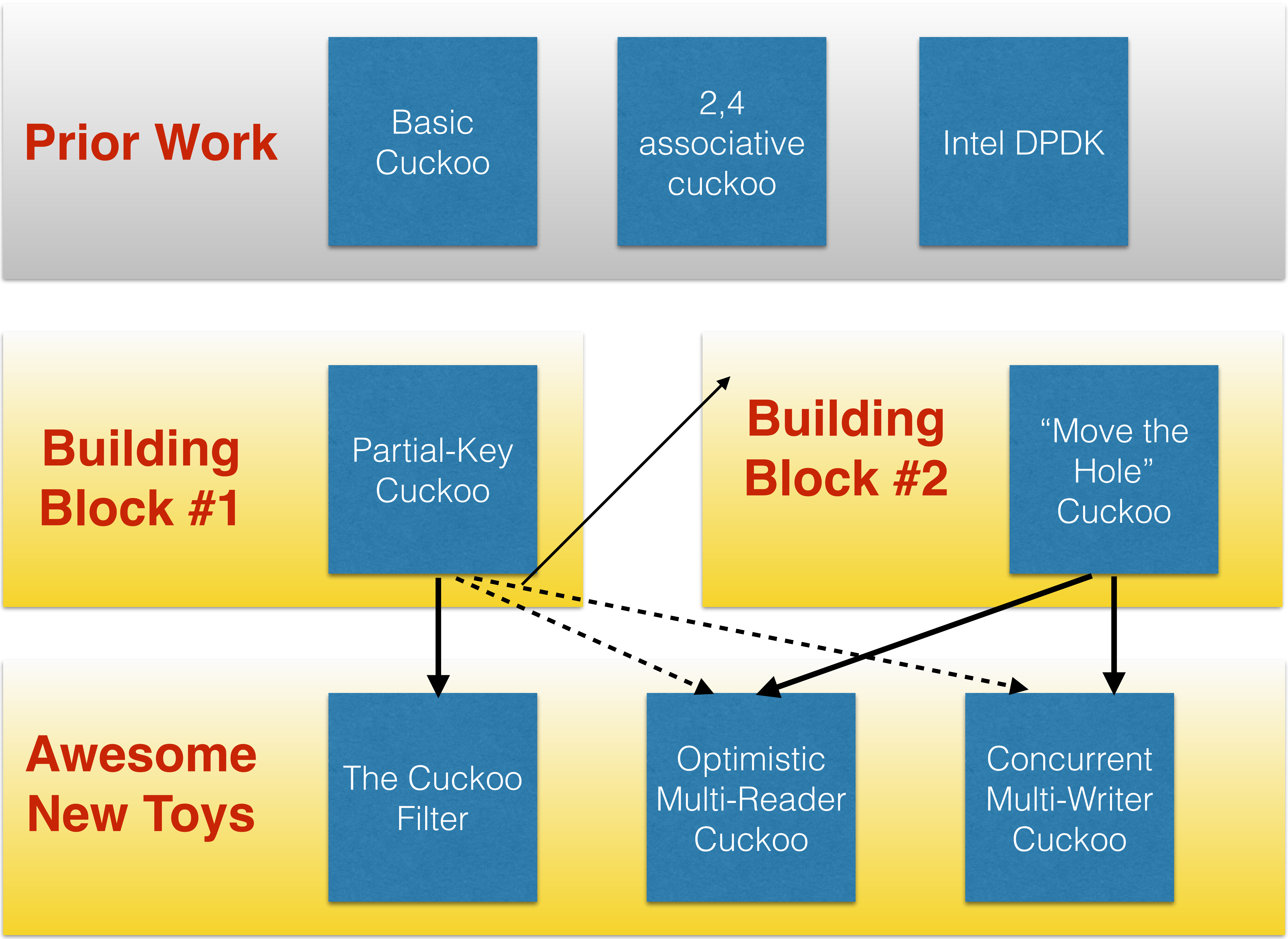
The Cuckoo Filter

Optimistic Multi-Reader Cuckoo

Concurrent Multi-Writer Cuckoo

Ongoing work:
Intel DPDK + alternate hash designs =

70 million key/value ops/sec
over the network



Prior Work

Basic Cuckoo

2,4 associative cuckoo

Intel DPDK

Building Block #1

Partial-Key Cuckoo

Building Block #2

“Move the Hole” Cuckoo

Awesome New Toys

The Cuckoo Filter

Optimistic Multi-Reader Cuckoo

Concurrent Multi-Writer Cuckoo