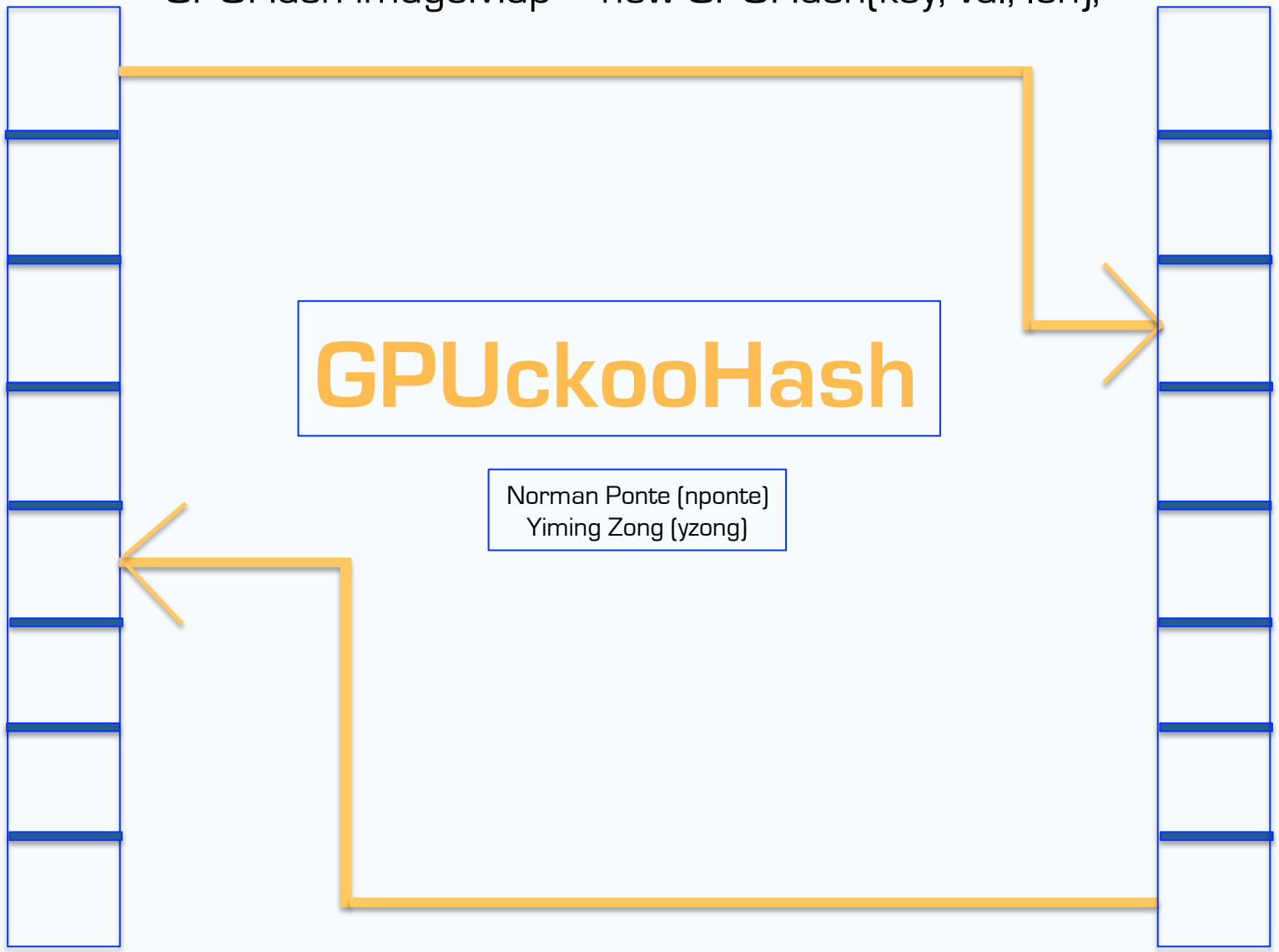
#import “GPUHash.h”

GPUHash imageMap = new GPUHash(key, val, len);



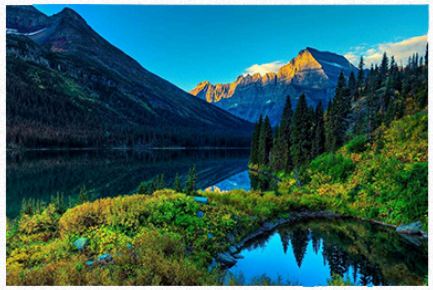
*(Motivation)

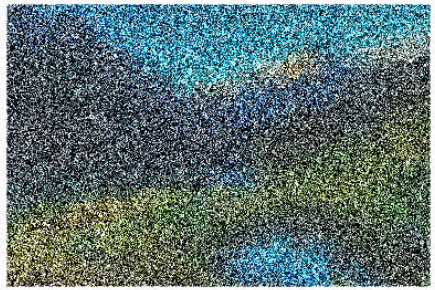
- Hash large images to short numbers
- Similar characteristic hash similarly
- Hamming Distance – difference in bits between two hashes
- Spatial Hashing

4,000 / sec

58,000,000 / day

376 TB





2014 - 880,000,000,000

Parallel Hashing?

- Synchronization
- Variable workloads
- High memory bandwidth
- Large data structures

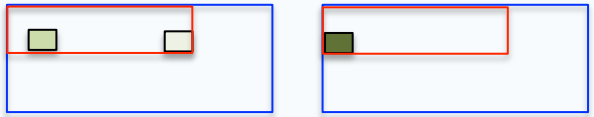
sparsehash / **sparsehash**

<> Code | Issues 8 | Pull requests 0 | Pulse | Graphs

Automatically exported from code.google.com/p/sparsehash

```
RESOURCE USAGE
--------------
* sparse_hash_map has memory overhead of about 2 bits per hash-map
  entry.
* dense_hash_map has a factor of 2-3 memory overhead: if your
  hashtable data takes X bytes, dense_hash_map will use 3X-4X memory
  total.
```

hash(key1, val1) – Bucket 1 -
hash(key2, val2) – Bucket 2 -
hash(key3, val3) – Bucket 1 -



Xeon Phi L1 Cache :: 64 bytes

M

PH?

CC

H

D

B

BS

MvS

B.I

B.II

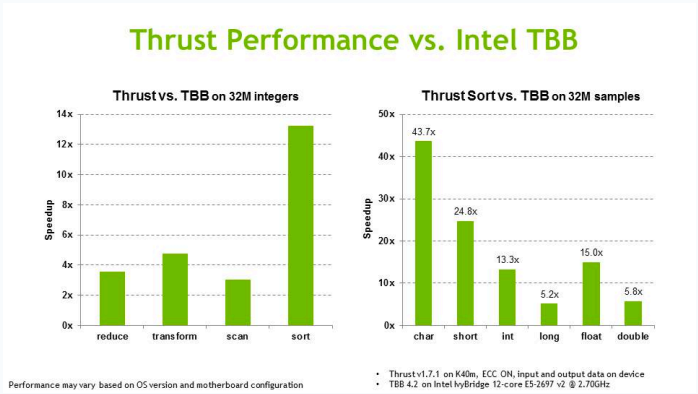
B.III

Optimizing for Our Common Case

3 Metrics : Speed, Memory Footprint, Construction Time

- Care about mostly lookup
- Want rapid construction and deletion
- User defined Space vs. Speed tradeoff
- Batch Processing
- CUDA thrust library

```
/** @brief Look into hash table and return values for
  *        given keys.
  *
  *  Probe the hash table and return value for input     a
  *  array key. 0 means failure.
  *
  *  @param keys – keys array to lookup
  *  @param val_ret – array to fill with values for
  *                   corresponding keys
  *  @param len – length of keys
  *  @return Void.
  */
void GPUHash::lookupTable(int* key, int* val_ret, const int len)
```



M

PH?

CC

H

D

B

BS

MvS

B.I

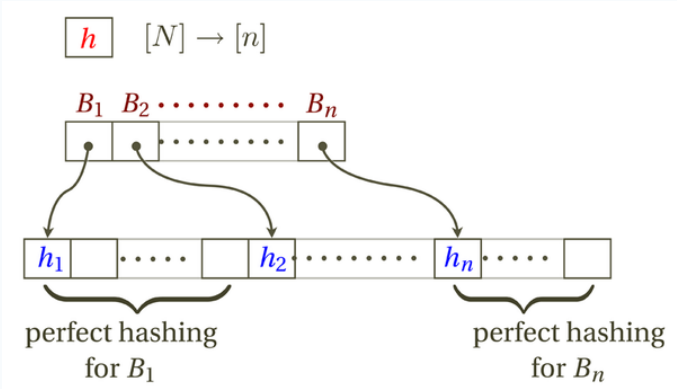
B.II

B.III

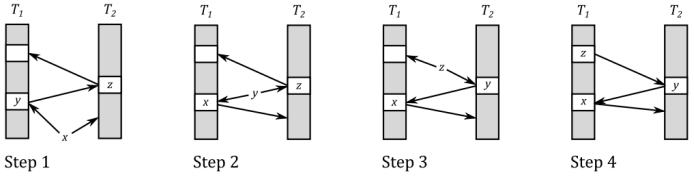
FKS Hashing + Cuckoo Hashing

- Lock Free
- O(1) - Lookup time
- 3 – Seed Cuckoo Hashing
- Hash to FKS Bucket
- Hash to 3 Bucket inside FKS Bucket using cuckoo hash functions
- Random Seed

FKS Hashing



Cuckoo Hashing



M

PH?

CC

H

D

B

BS

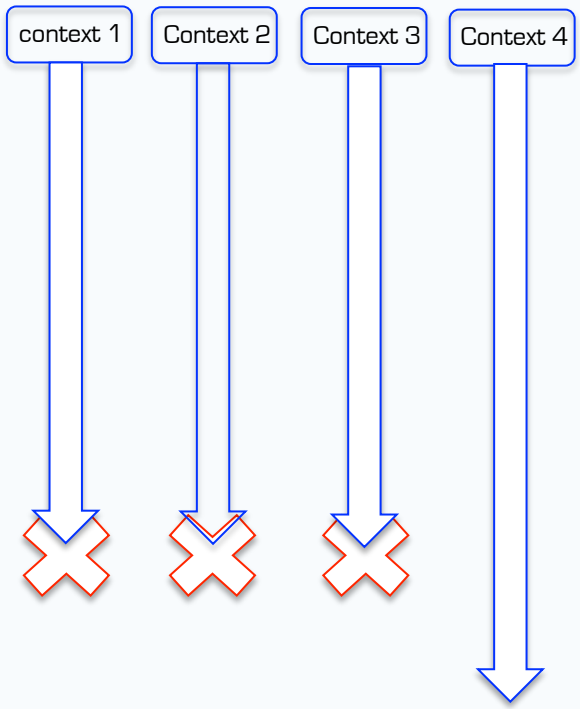
MvS

B.I

B.II

B.III

Divergence




```
// Loop for all cuckoo hashing functions
for (int i = 0; i < NUM_FUNC; i++) {
  // Use one of the hash functions
  unsigned int hash;
  switch (i) {
    case 0: hash = hashKey(key, b->get_hash1());
            break;
    case 1: hash = hashKey(key, b->get_hash2());
            break;
    case 2: hash = hashKey(key, b->get_hash3());
            break;
  }

  //Bucket does atomic exchange
  newkeyval = b->add_slot(key, val, i, hash);
  // If I got into the spot then exit routine
  if (newkeyval == 0) {
    return thrust::make_pair(0,0);
  }
  key = get_key(newkeyval);
  val = get_val(newkeyval);
}
```

- cudaThreads carry out 3 stage cuckoo hashing
- ~68% find a spot on first hash

- M
- PH?
- CC
- H
- D
- B
- BS
- MvS
- B.I
- B.II
- B.III

Balancing

Load Factor	Best Loop to Converge
.3	3
.5	5
.8	14
.9	30

Decrease Bucket Factor Low	Increase Bucket Factor High
- Waste of Space	- Space Efficient
- Faster runtime	- Slow runtime
- Divergence	- Could total Reseed
- Poor cache usage	- Contention for atomic variable

- Modify buckets based on number of elements
- Reseed FKS buckets if one bucket is full >1%
- Good Hash Function 4x
- Loop iterations depends on load factor

M

PH?

CC

H

D

B

BS

MvS

B.I

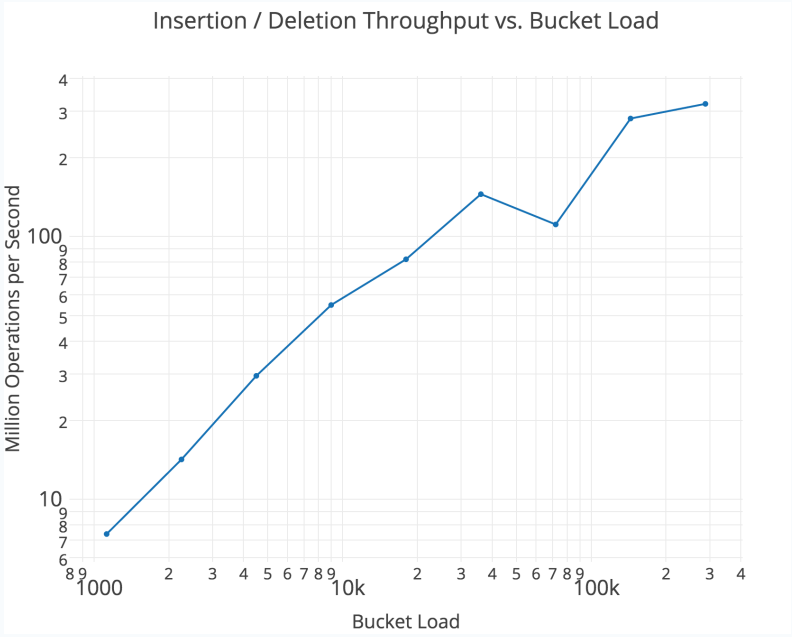
B.II

B.III

Bucket Size

BUCKET_LOAD = ?

- Bigger Bucket - can't fit in cache (512 KB)
- 80% time spent in sort
- Smaller bucket – pigeon hole principle
- 36 Byte Overhead per bucket
- Rebalancing cost is higher



M

PH?

CC

H

D

B

BS

MvS

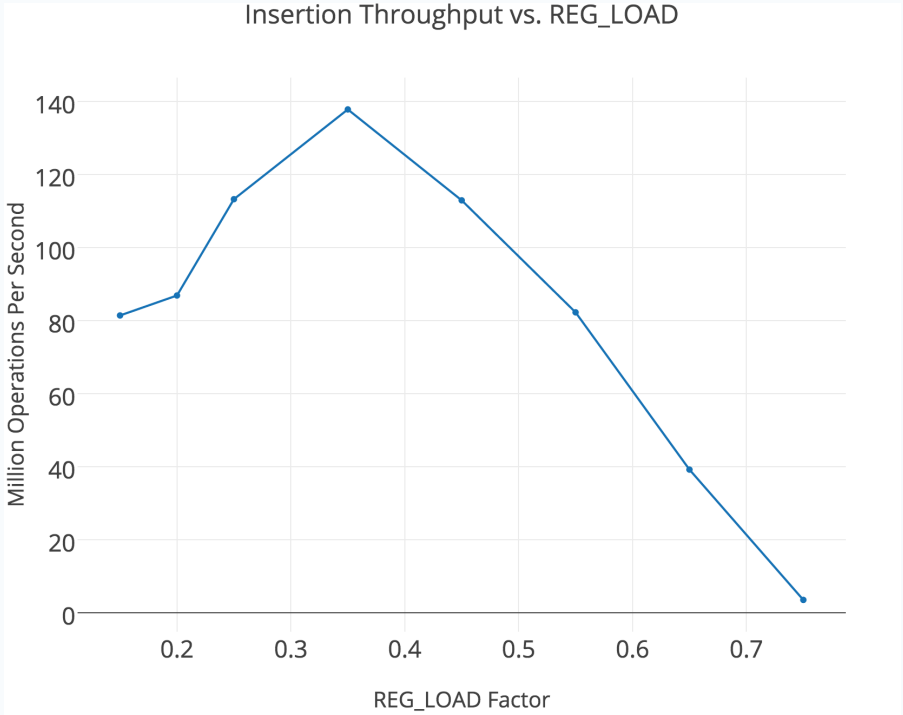
B.I

B.II

B.III

Memory and Speed

- Auto-rebalance for speed retention
- Perfect hashing N^2 space
- Permanent Memory Overhead 0
- Want to remain near the peak



M

PH?

CC

H

D

B

BS

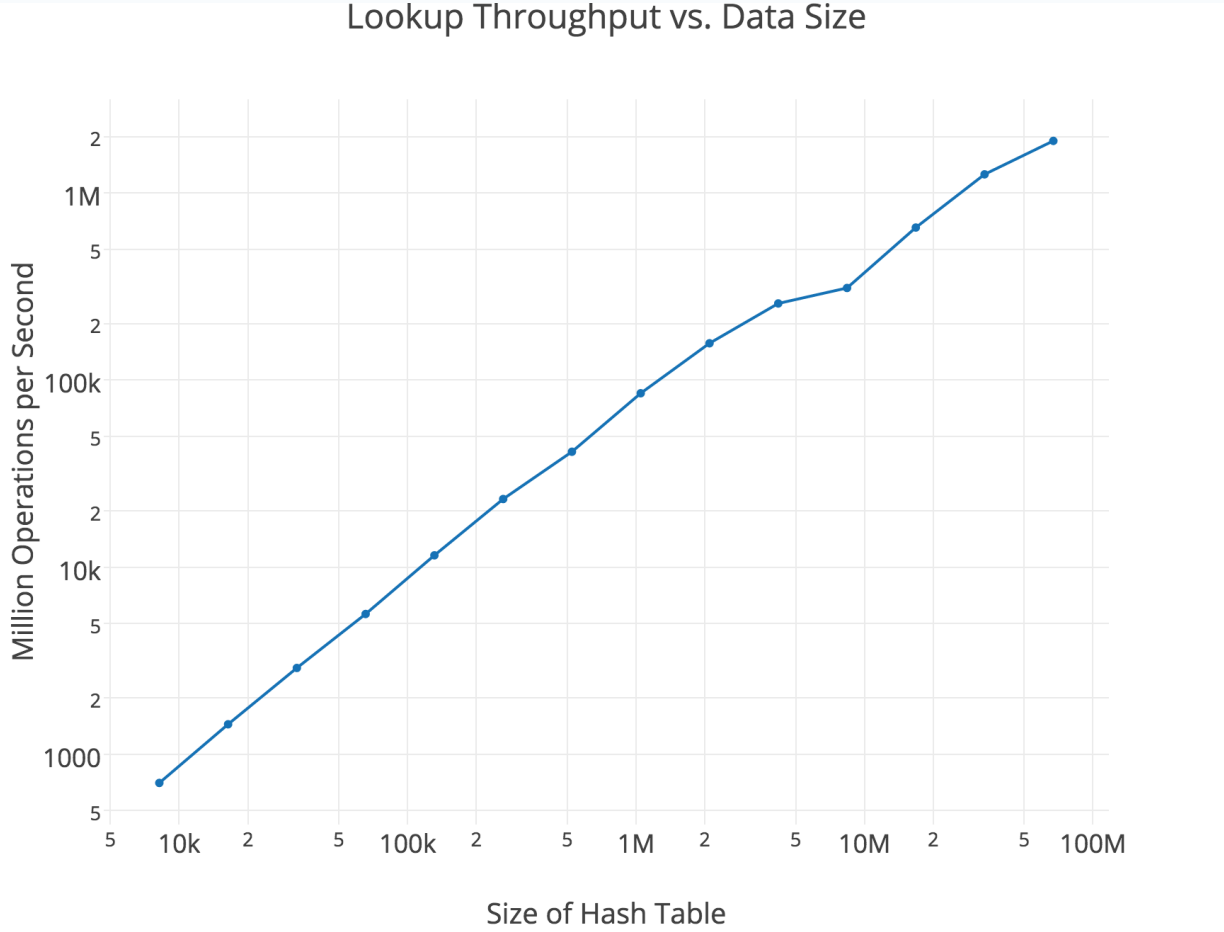
MvS

B.I

B.II

B.III

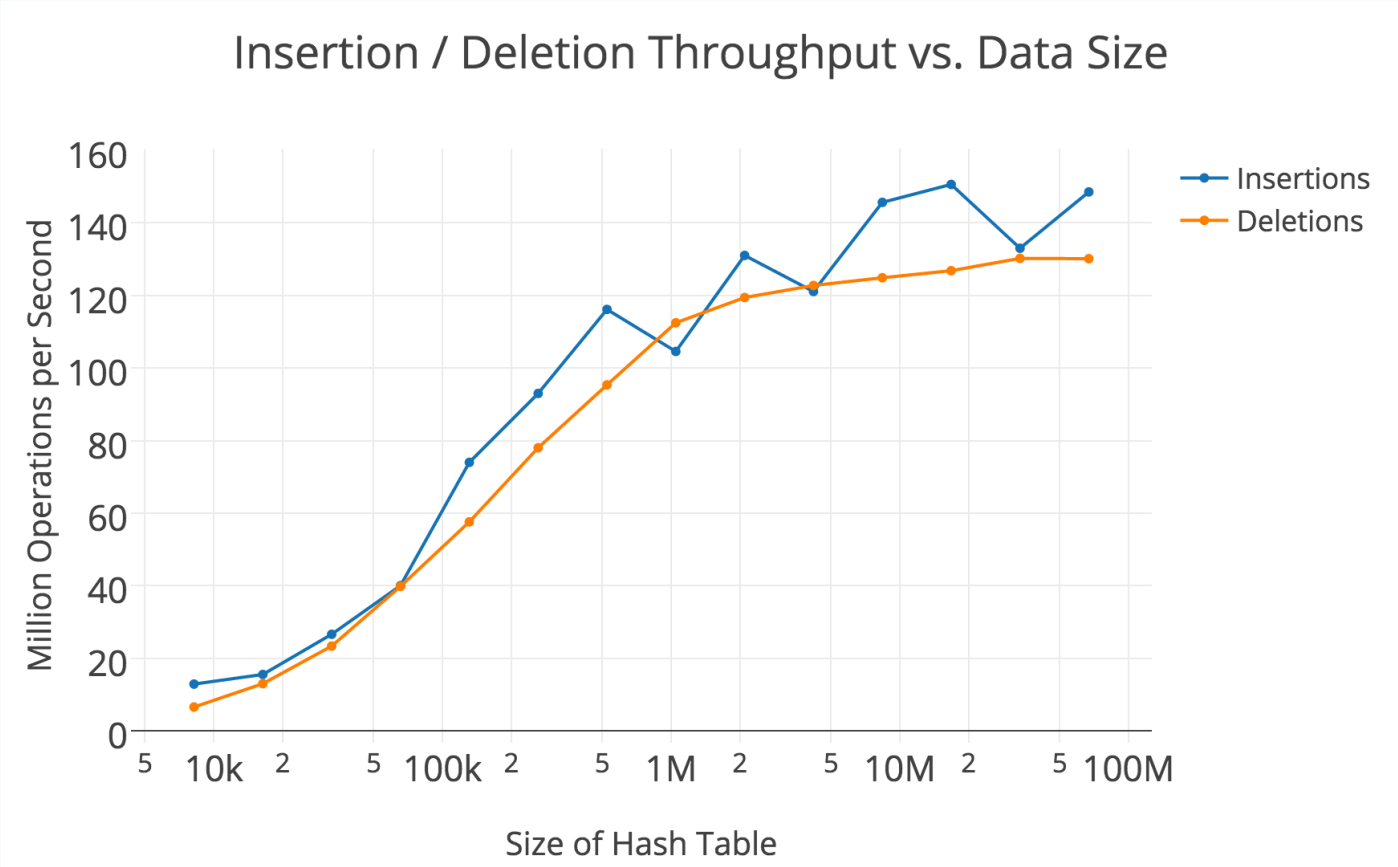
Benchmark :: Lookup



- Most important graph O(1) – 3 Lookups

- M
- PH?
- CC
- H
- D
- B
- BS
- MvS
- B.I
- B.II
- B.III

Benchmark :: insert + delete



- No rebalancing

M

PH?

CC

H

D

B

BS

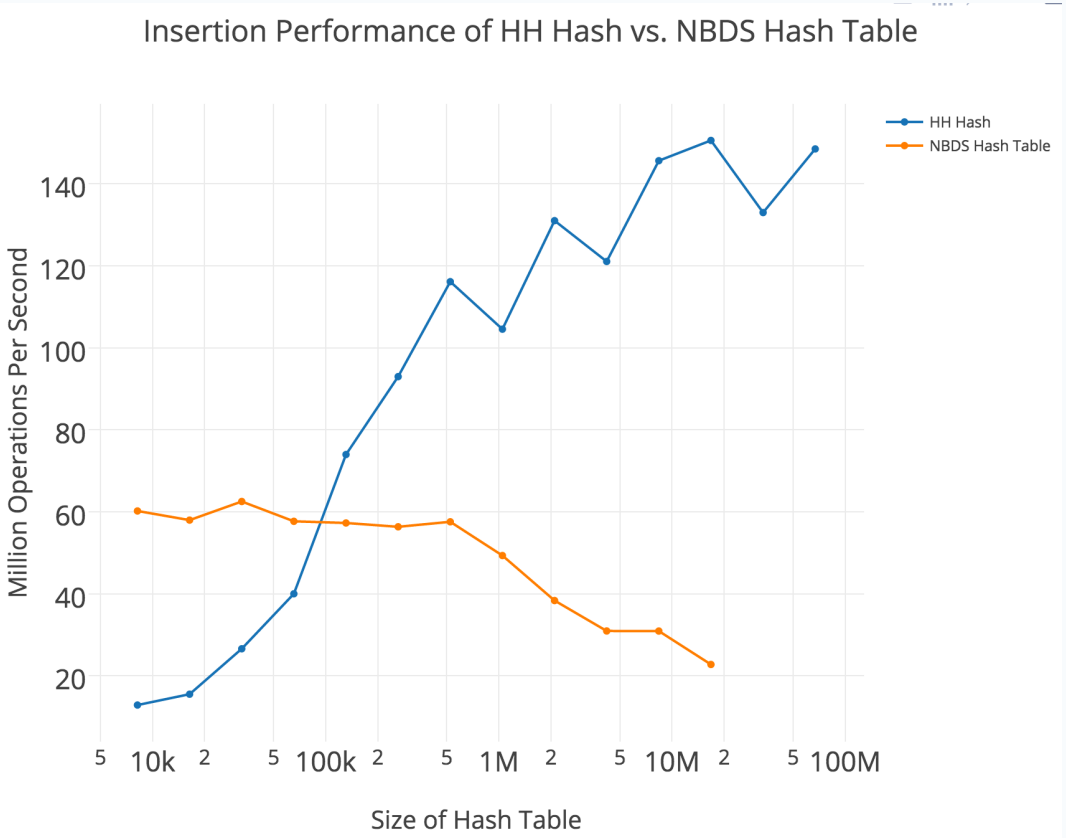
MvS

B.I

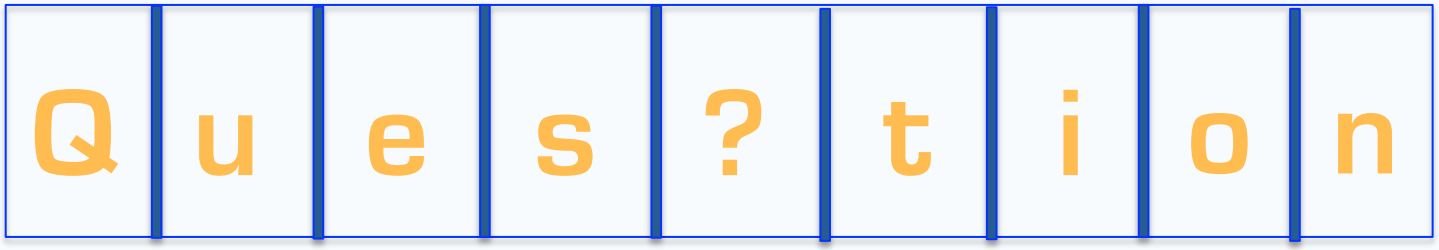
B.II

B.III

Benchmark :: the field



- Considered to be one of the best lock free hash table
- Memory error


Ques?tion