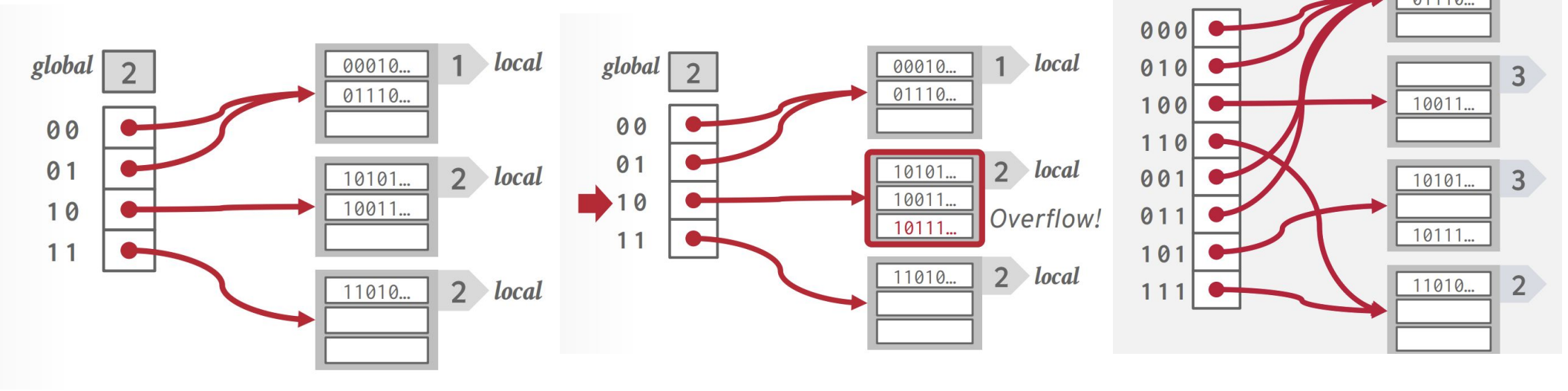
Carnegie Mellon University

Concurrent Extendible Hash Table

Shivang Dalal & Ashwin Rao
MSCS 2nd year

What is Extendible Hashing

- Ubiquitous data structure in Databases. Critical for indexes and joins.
- 2-level dynamic data structure that can perform bucket level re-hashing on overflow.
- Space efficient for sparse tables and optimized for being disk backed.
- Example of inserting a key below.



Coarse and Fine Grained locking

Coarse grained locking:

- Single directory-level read-write mutex guards all operations.
  - Correct since all hash table operations go through the directory.
- Shared lock for read operations (multiple readers simultaneously).
- Exclusive lock for write operations (ex: splitting, merging buckets).

Fine grained locking:

- Directory-level read-write mutex guards changes to directory.
- Bucket-level mutex guards data operations on a bucket.
- Early lock release strategy for contention reduction.
  - Acquire directory lock on insert/delete only for bucket split/merge.
- Global lock ordering for deadlock prevention.

Lock Free Hash Table

Lock free Bucket:

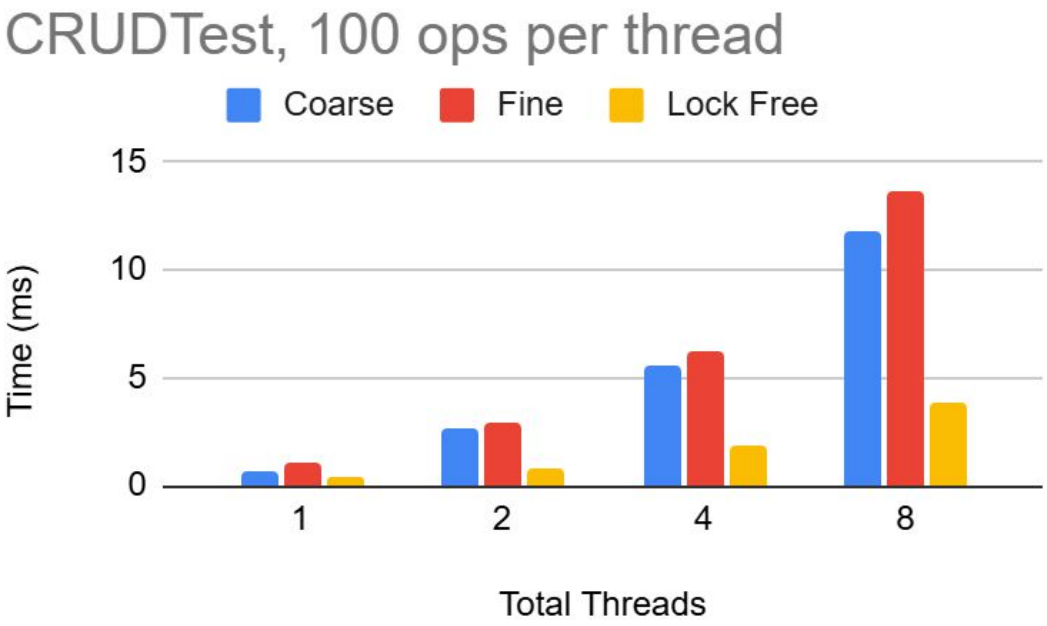
- Implemented using a lock-free linked list, kept sorted for efficiency.
- Nodes have atomic ‘next’ pointers updated using Compare and Swap.
- If CAS fails (due to a concurrent operation), the operation is retried in a loop until it succeeds.

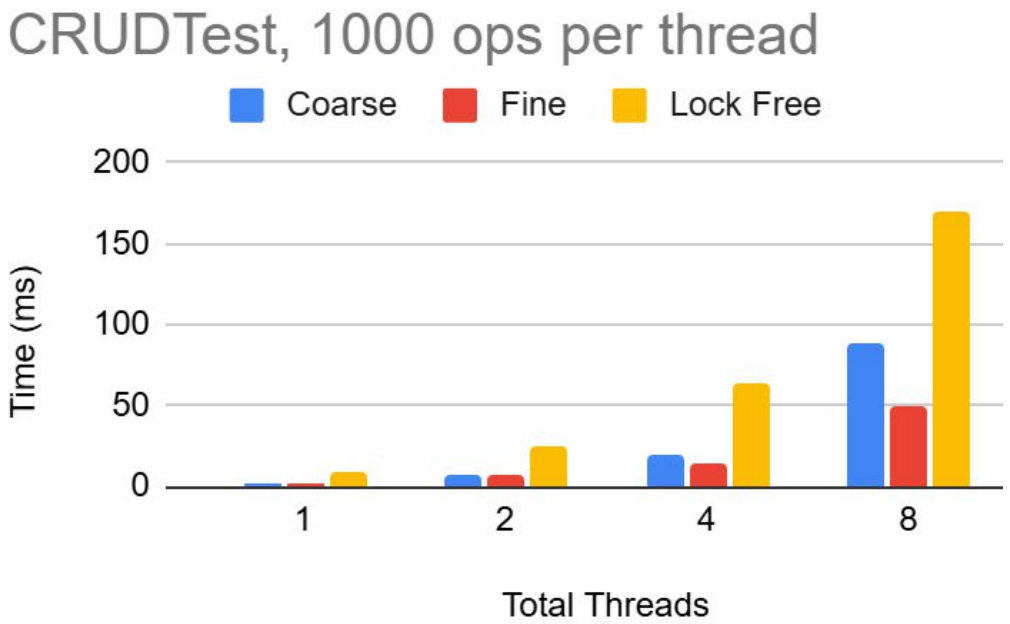
Lock free Extendible Hash Table:

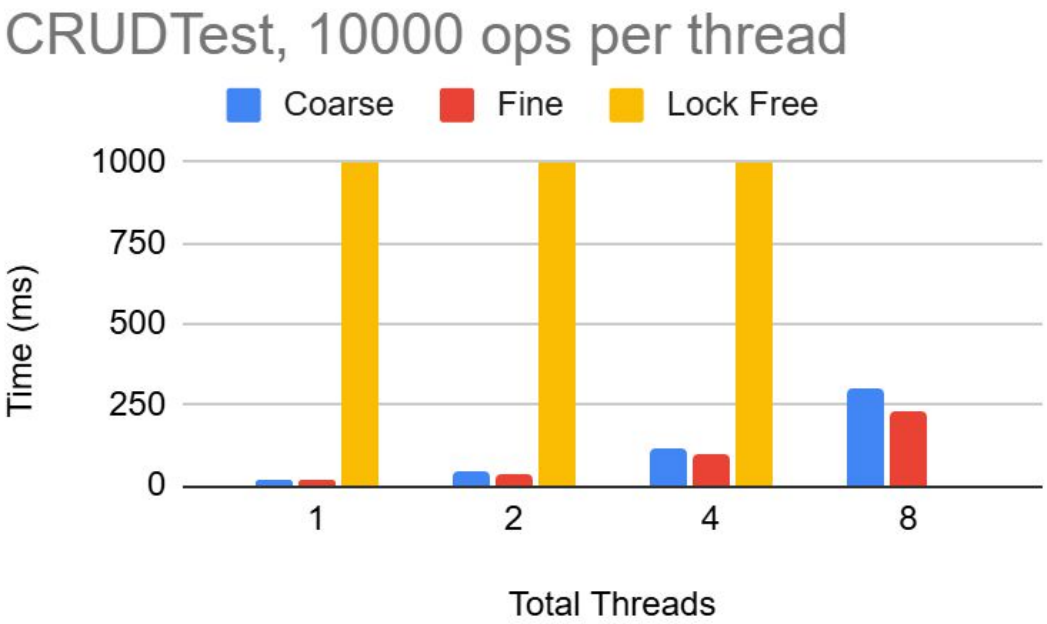
- No Directory lock. Replaced with an atomic array of Bucket pointers.
- On Directory changes, the array (Bucket**) is modified using CAS.
- Memory optimization: lock-free Buckets initialized on demand.
  - CAS to ensure that each bucket is initialized by a single thread.
- No deadlocks due to no locking. Some thread always makes progress.

CRUD Benchmark

- Create, Read and Delete by every thread
- Fine grained clear favorite under high load
- Lock free better under low to medium load

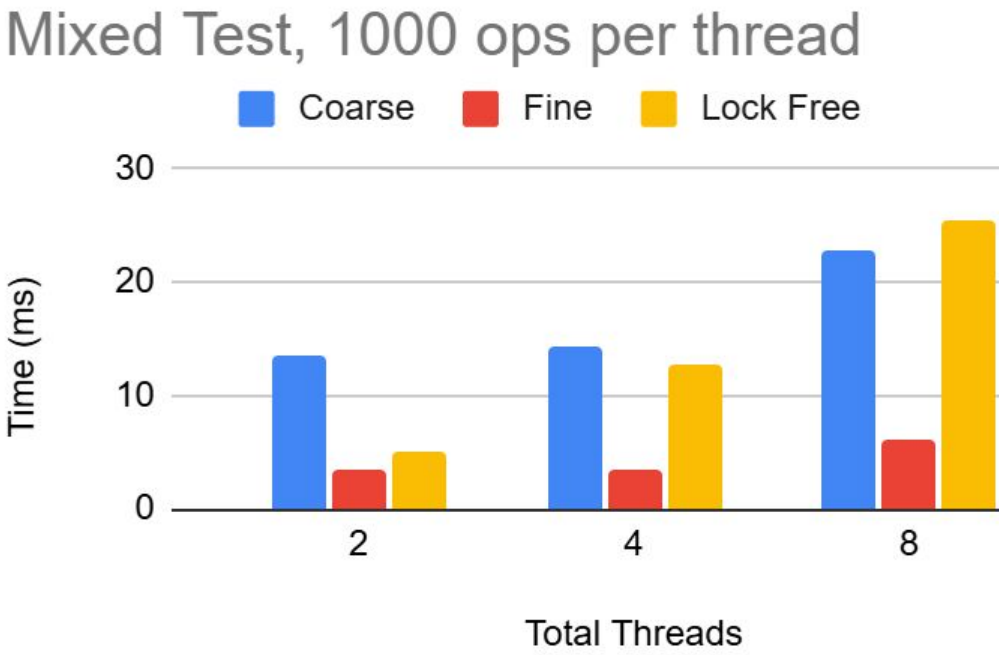


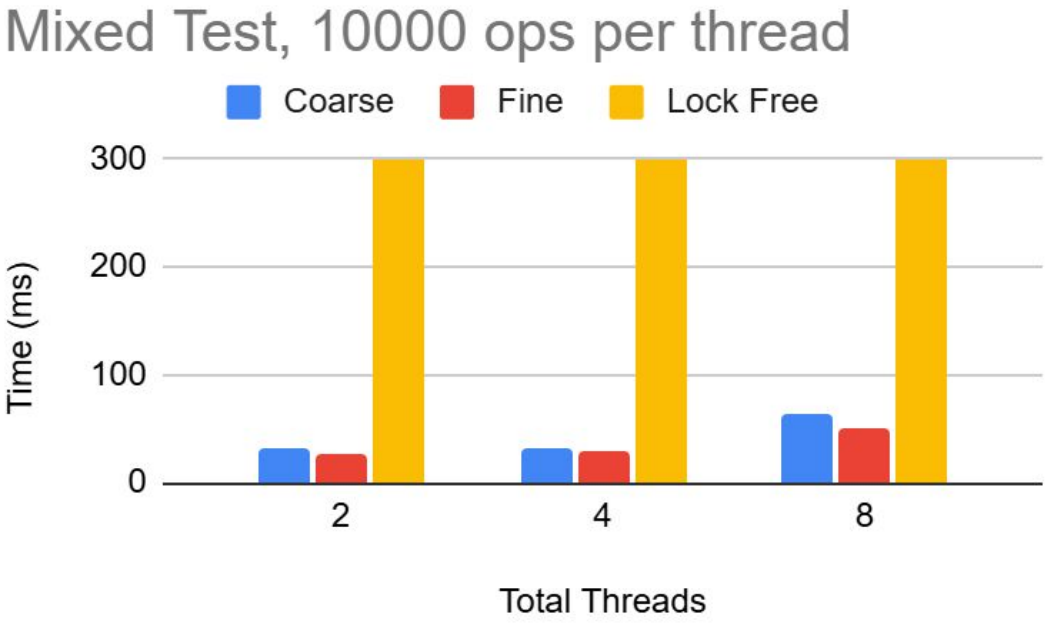




Mixed Benchmark

- Parallel pure readers and writers. (equal ratio)
- Fine grain performs even better under high load since it can support reads more efficiently.





Memory and speedup

- **All three variants within 5% of each other in experiments.**
- More space efficient buckets may have seen a bigger % difference.

- Lack of critical section computation makes speedup very hard to calculate.
- Parallel always performs worse compared to serial due to overhead of sharing data and locks.
- **Artificial delays of even 10 us in worker threads showed good speedup (7-7.5x) though.**

Conclusion and recommendations

Feature	Coarse Grained	Fine Grained	Lock Free
Locking Strategy	Single directory mutex	Two-level locking	No locks. Atomic operations
Concurrency under light to medium load	Moderate	Moderate	Best
Concurrency heavy load	Bad	Best	Horrendous
Complexity	Simple	Moderate	High

- Fine-grained locking is the most suitable choice for production environments for consistent performance under varying loads.
- Lock-free implementations could be beneficial for specialized light-workload scenarios.