Concurrency in Linear Hashing

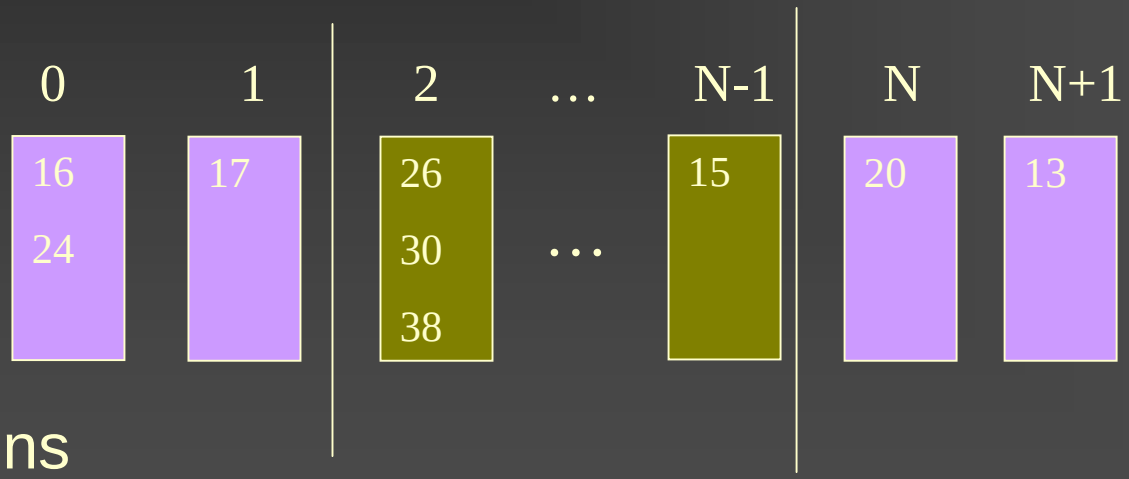
Huxia Shi

York University

Apr 22, 2009

Linear Hashing Review

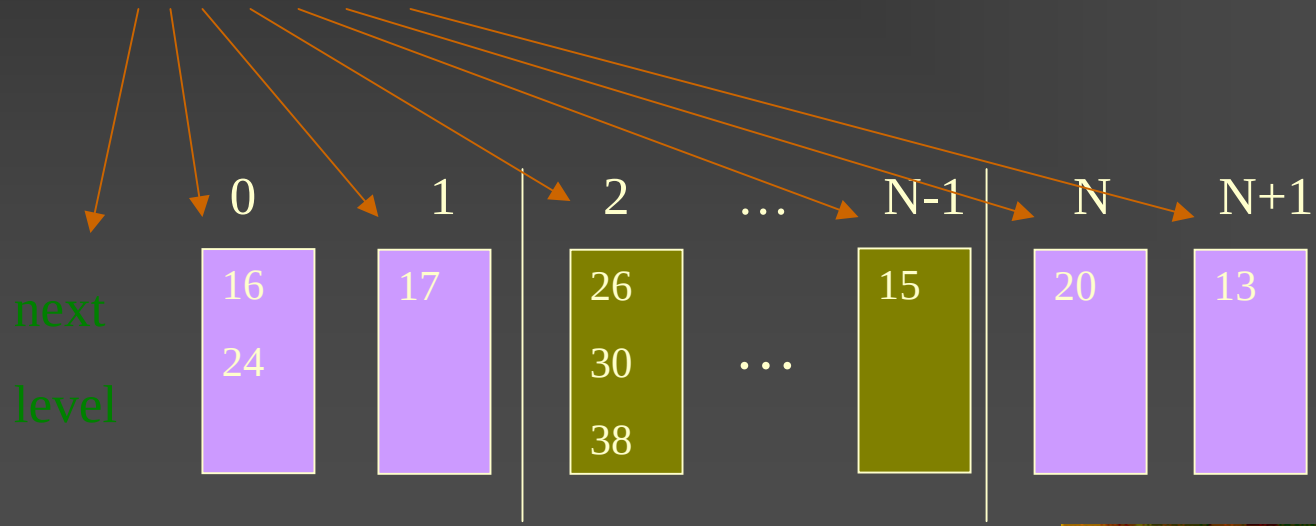
- A technique of dynamic hashing
- Data structure
  - Root variables: next and level
  - Sequence of Bucket Chains



- Operations
  - Find, Insert, Delete, Split, Merge

Linear Hashing Concurrent Solution

- Carla Schlatter Ellis. Concurrency in linear hashing. *ACM Transactions on Database Systems (TODS)*, 12(2): 195-217, June 1987.
- Three lock types
  - Read Lock, Selective Lock, Exclusive Lock



Locks

	Existing lock		
Lock Request	Read lock	Selective lock	Exclusive lock
Read lock	yes	yes	no
Selective lock	yes	no	no
Exclusive lock	no	no	no

3 Locks Implementation – Lock class

```
public class Lock {
    public static final int LOCKTYPE_READ = 1;
    public static final int LOCKTYPE_SELECTIVE = 2;
    public static final int LOCKTYPE_EXCLUSIVE = 3;

    public synchronized void requestLock(int locktype) {
        …
    }
    public synchronized void releaseLock(int locktype)  {
        …
    }
}
```

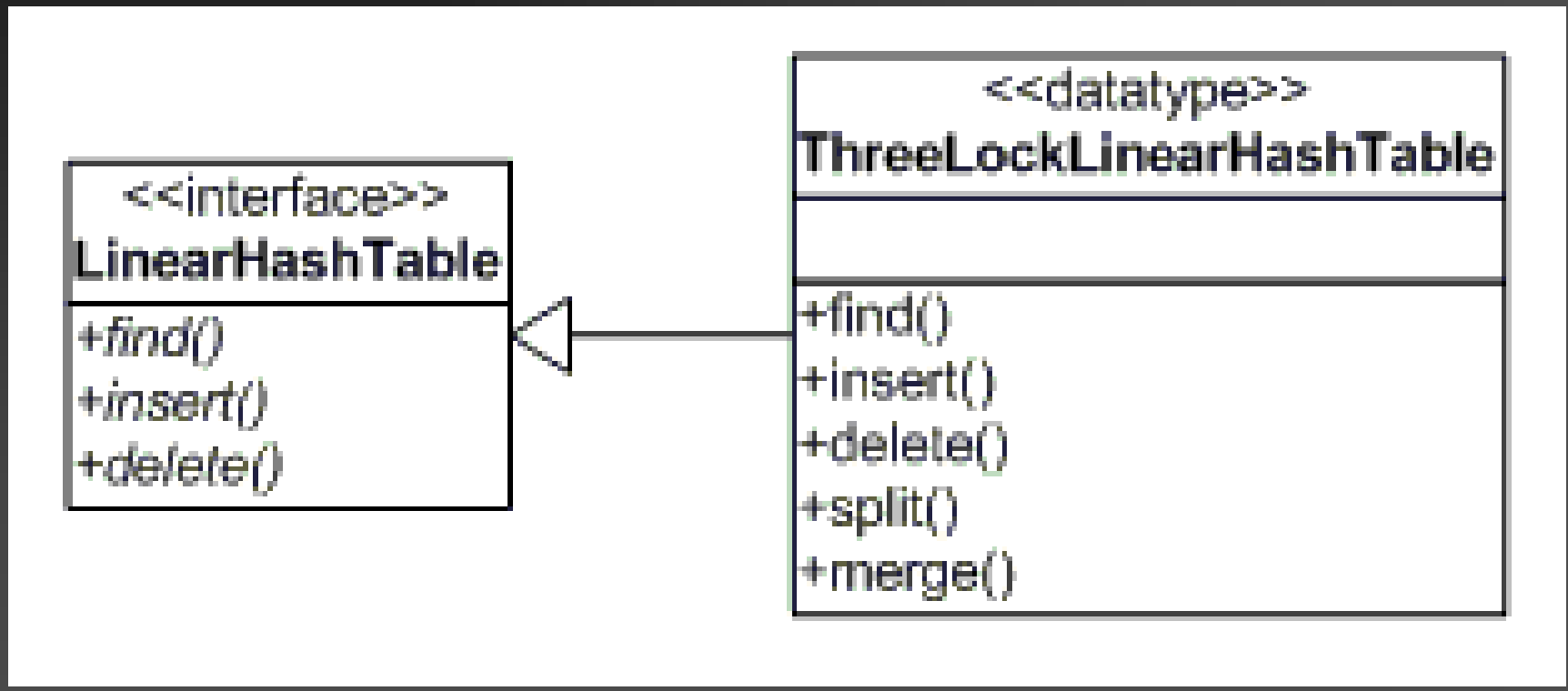
3 Locks Implementation – Lock Class

```
public synchronized void requestLock(int locktype) {
    while (true) {
        boolean getLock = false;

        switch (locktype) {
        case LOCKTYPE_READ:
            if (this.exclusiveLockNum == 0) {
                this.readLockNum++;
                getLock = true;
            }
            break;
        … …
        }

        if (getLock) {break; }
        else {
            try {this.wait();}
            catch (InterruptedException e) {}
        }
    }
}
```

3 Locks Implementation – Lock Class

```
public synchronized void releaseLock(int locktype) {
    switch(locktype) {
    case LOCKTYPE_READ:
        this.readLockNum --;
        break;
    case LOCKTYPE_SELECTIVE:
        this.selectiveLockNum --;
        break;
    case LOCKTYPE_EXCLUSIVE:
        this.exclusiveLockNum --;
        break;
    default:
        throw new RuntimeException("Internal Error: unknown lock type "
                + locktype);
    }

    this.notifyAll();
}
```

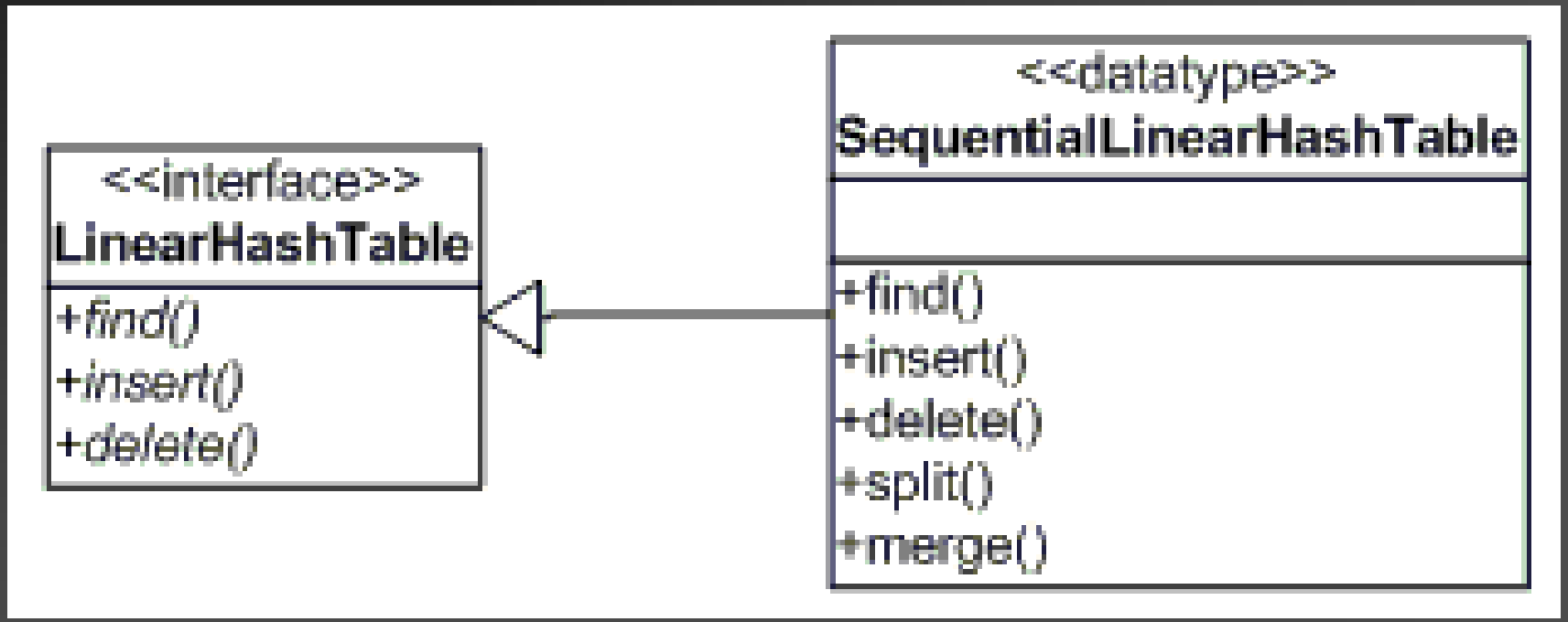
3 Locks Implementation – ThreeLockLinearHashTable



3 Locks Implementation – ThreeLockLinearHashTable

```
public class ThreeLockLinearHashTable implements LinearHashTable
{

    private int level;
    private int next;
    private Lock rootLock = new Lock();

    private Node[] bucketChainList = new Node[2000];
    … …
}

public class Node {
    LocalLevelBucketChain chain;
    Lock chainLock;
}
```

Monitor Implementation – SequentialLinearHashTable



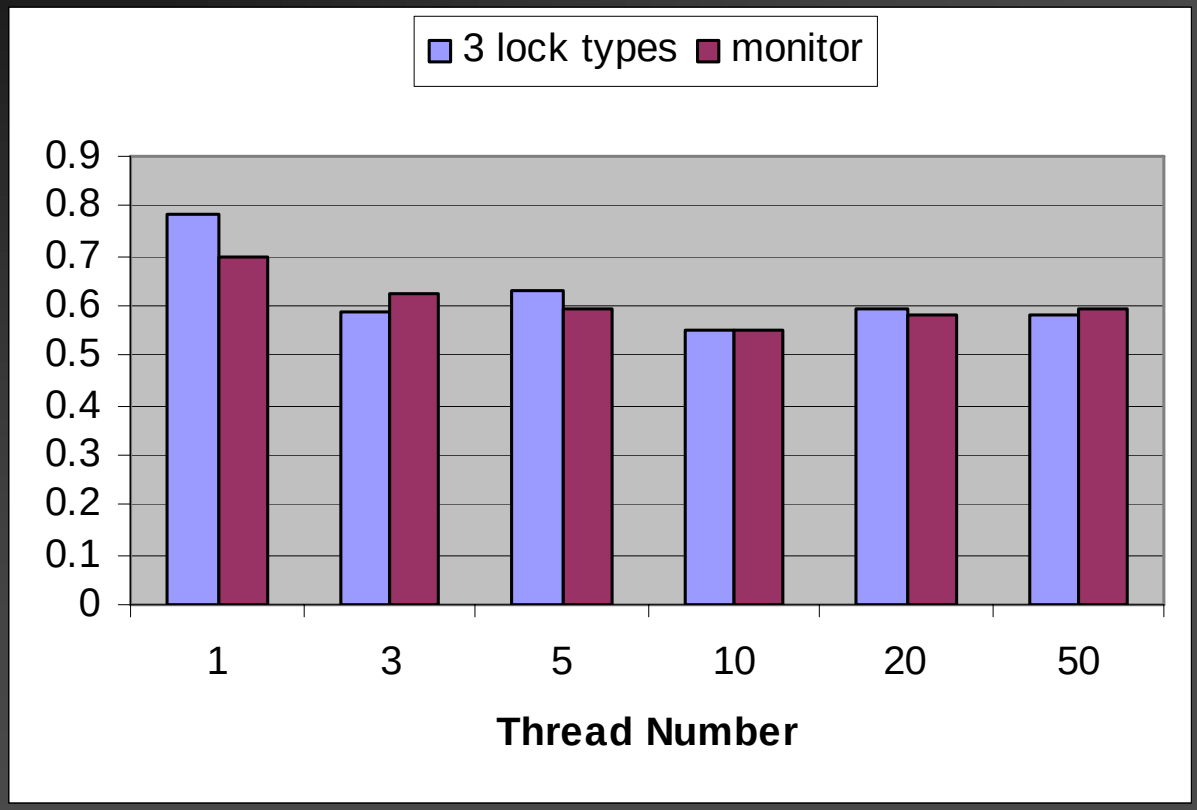
Monitor Implementation – SequentialLinearHashTable

```
public class SequentialLinearHashTable implements LinearHashTable
{
    private int level;
    private int next;

    private BucketChain[] bucketChainList = new BucketChain[2000];

    public synchronized boolean find(int key) { … }
    public synchronized void insert(int key) { … }
    public synchronized void delete(int key) { … }

    private void split() { … }
    private void merge() { … }
}
```

Test Environment

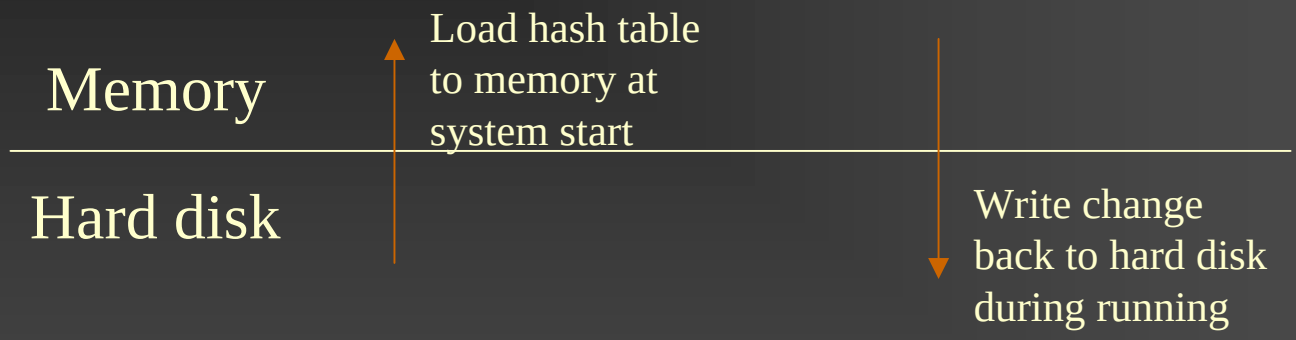
- Server
  - prime.atkinson.yorku.ca
- CPU
  - 4 Intel Xeon 5160 3 GHz Dual Core Processors
- Memory
  - 16 GB

Performance



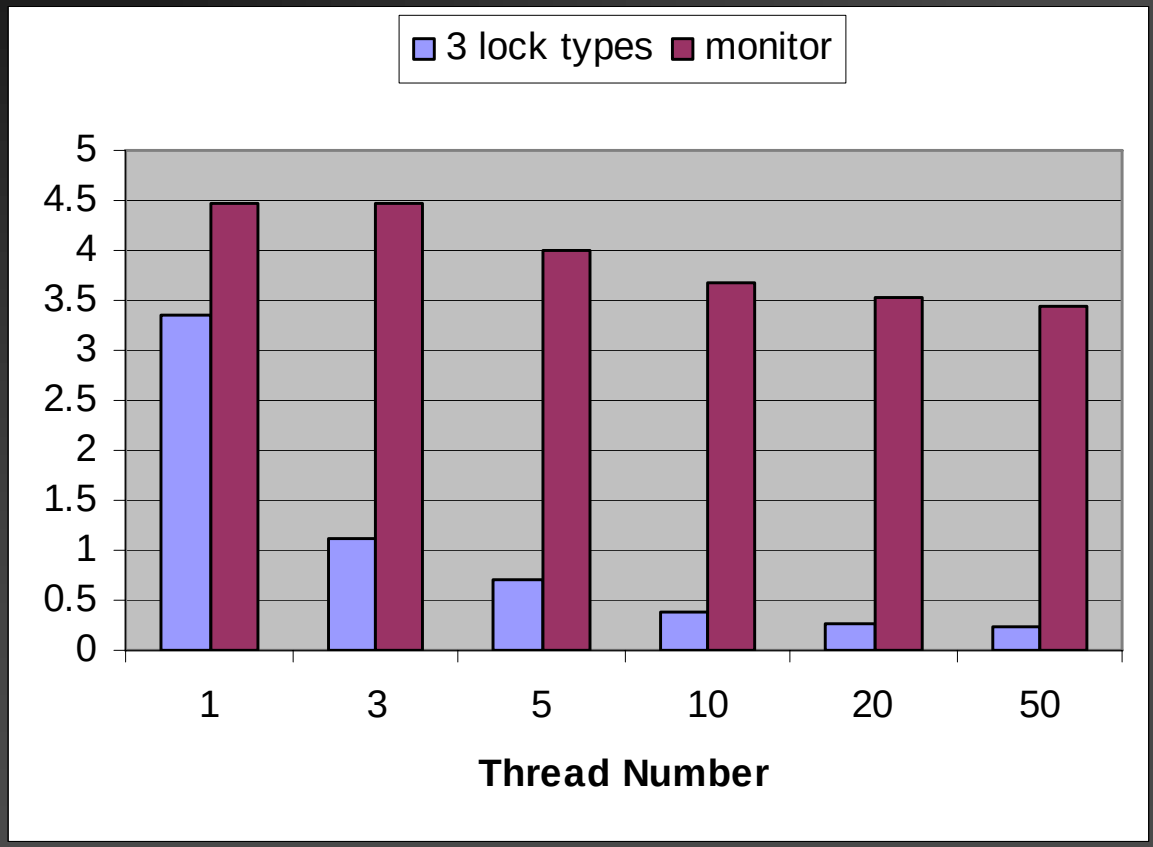
Thread number n : n insert threads, n delete threads, n find threads at the same time

Performance – Real application



- Add delay to simulate writing back
  - sleep 10 ms at insert and delete
  - Only apply to big hash table 50MB/s * 10ms = 500KB

Performance – With delay



Thread number n : n insert threads, n delete threads, n find threads at the same time

Looking Ahead

- Verify deadlock freedom and some other properties using Java PathFinder

The End

Thanks

Q&A