Characterizing the Energy Efficiency of Java's Thread-Safe Collections in a Multi-Core Environment

Gustavo Pinto
ghlp@cin.ufpe.br

Fernando Castor
castor@cin.ufpe.br

VIRTUS IMPAVIDA

ines

SPG
SOFTWARE·PRODUCTIVITY·GROUP

Centro de Informática
UFPE

Motivation (1/3)

- **First**, **energy consumption** is a concern for unwired devices and also for data centers

- **Second**, there is a large body of work in hardware/architecture, OS, runtime systems

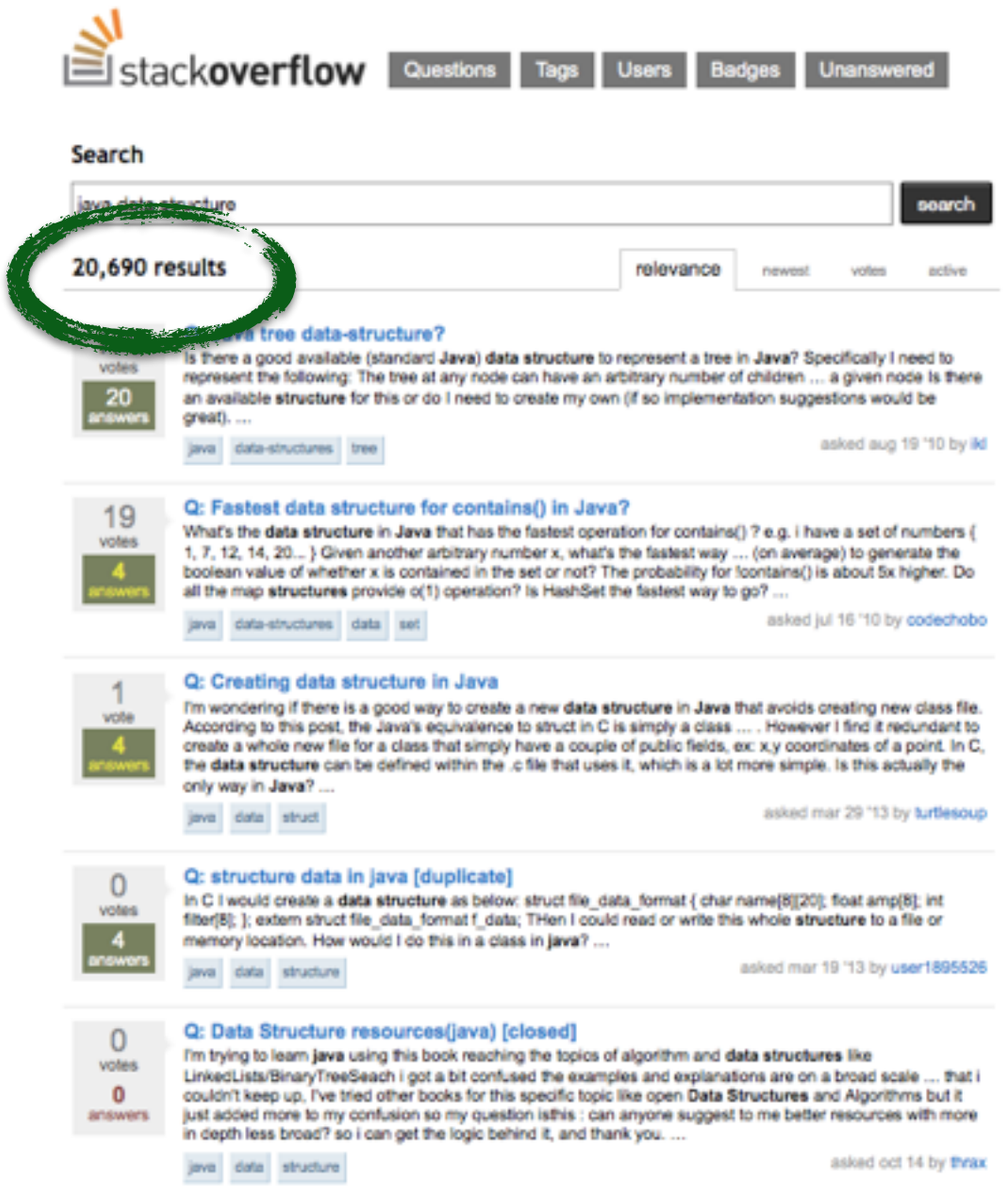
- **However**, little is known about the application level

Motivation (2/3)

- **First, multicore** CPUs are ubiquitous
- **Second,** more cores used → more power consumed
- **However**, little is known about the energy-efficiency of multicore programs

Motivation (3/3)

- Data structures are the fundamentals of computer programming



List<Object> lists = …;

- **ArrayList**
- **LinkedList**

List<Object> lists = **new ArrayList<>();**

Thread 

Not thread safe!

List<Object> lists = **new ArrayList<>();**

List<Object> lists = …;

- **ArrayList**
- **LinkedList**

- **Vector**
- **Collections.synchronizedList()**
- **CopyOnWriteArrayList**

List<Object> lists = **new Vector<>();**

Thread 

List<Object> lists = **new Vector<>();**

Thread 

Thread-safe!

List<Object> lists = **new Vector<>();**

Thread

Thread-safe!

List<Object> lists = **new Vector<>();**

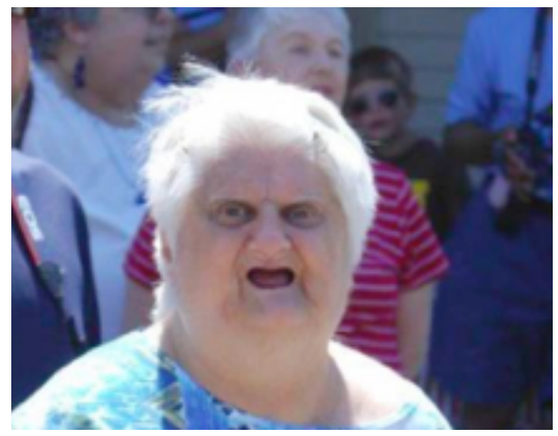
```
public class Vector<E> {
    public synchronized void addElement(E obj) {..
    public synchronized boolean removeElement(Object obj) {..
    public synchronized E get(int index) {..
    ...
}
```

Thread 

Thread-safe!

List<Object> lists = **new Vector<>();**

```
public class Vector<E> {
    public synchronized void addElement(E obj) {..
    public synchronized boolean removeElement(Object obj) {..
    public synchronized E get(int index) {..
    ...
}
```



```
List<Object> lists = new CopyOnWriteArrayList<>();
```

Thread

List<Object> lists = **new CopyOnWriteArrayList<>();**

Thread

Thread-safe!

List<Object> lists = **new CopyOnWriteArrayList<>();**

Thread

Thread-safe!

List<Object> lists = **new CopyOnWriteArrayList<>();**

```
public class CopyOnWriteArrayList<E> {

    private E get(Object[] a, int index) {
        return (E) a[index];
    }

}
```

Thread

Thread-safe!

List<Object> lists = **new CopyOnWriteArrayList<>();**

```
public class CopyOnWriteArrayList<E> {

    private E get(Object[] a, int index) {
        return (E) a[index];
    }
```

```
}
```

Thread

Thread-safe!

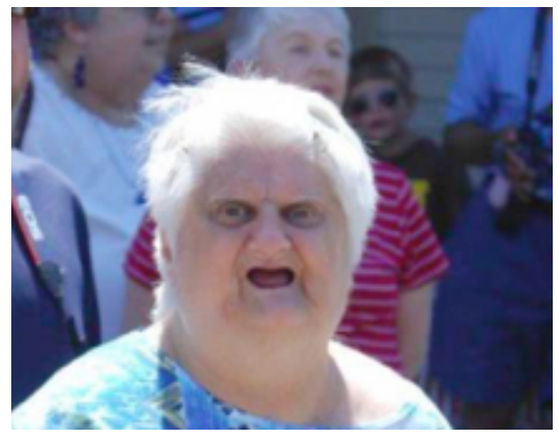
List<Object> lists = **new CopyOnWriteArrayList<>();**

```
public class CopyOnWriteArrayList<E> {

    private E get(Object[] a, int index) {
        return (E) a[index];
    }

    public boolean add(E e) {
        final ReentrantLock lock = this.lock;
        lock.lock();
        try {
            Object[] elements = getArray();
            int len = elements.length;
            Object[] newElements = Arrays.copyOf(elements, len + 1);
            newElements[len] = e;
            setArray(newElements);
            return true;
        } finally {
            lock.unlock();
        }
    }

}
```



List<Object> lists = …;

- **ArrayList**
- **LinkedList**
- **….**

- **Vector**
- **Collections.synchronizedList()**
- **CopyOnWriteArrayList**
- **….**

List<Object> lists = …;

Set<Objects> sets = …;

Map<Object, Object> maps = …;

List<Object> lists = …;

Set<Objects> sets = …;

Map<Object, Object> maps = …;

Research Questions

- **RQ1:** Do different implementations of the same collection have different impacts on energy consumption?

- **RQ2:** Do different operations in the same implementation of a collection consume energy differently?

16 Benchmarks

List	Set	Map
ArrayList	LinkedHashSet	LinkedHashMap
Vector	———	Hashtable
Collections.syncList()	Collections.syncSet()	Collections.syncMap()
CopyOnWriteArrayList	CopyOnWriteArraySet	———
———	ConcurrentSkipListSet	ConcurrentSkipListMap
———	ConcurrentHashSet	ConcurrentHashMap
———	ConcurrentHashSetV8	ConcurrentHashMapV8

16 Benchmarks

List	Set	Map
ArrayList	LinkedHashSet	LinkedHashMap
Vector	———	Hashtable
Collections.syncList()	Collections.syncSet()	Collections.syncMap()
CopyOnWriteArrayList	CopyOnWriteArraySet	———
———	ConcurrentSkipListSet	ConcurrentSkipListMap
———	ConcurrentHashSet	ConcurrentHashMap
———	ConcurrentHashSetV8	ConcurrentHashMapV8

- Not-thread safe
- Thread safe

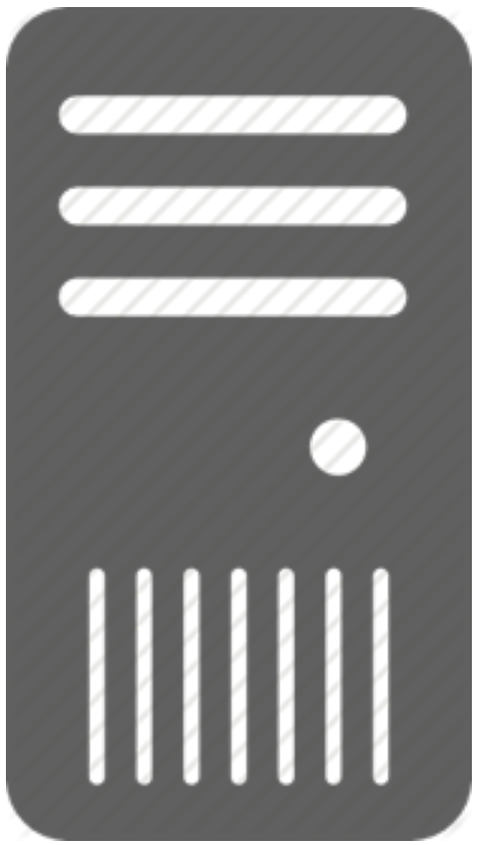
16 Benchmarks

List	Set	Map
ArrayList	LinkedHashSet	LinkedHashMap
Vector	———	Hashtable
Collections.syncList()	Collections.syncSet()	Collections.syncMap()
CopyOnWriteArrayList	CopyOnWriteArraySet	———
———	ConcurrentSkipListSet	ConcurrentSkipListMap
———	ConcurrentHashSet	ConcurrentHashMap
———	ConcurrentHashSetV8	ConcurrentHashMapV8

x 3 Operations

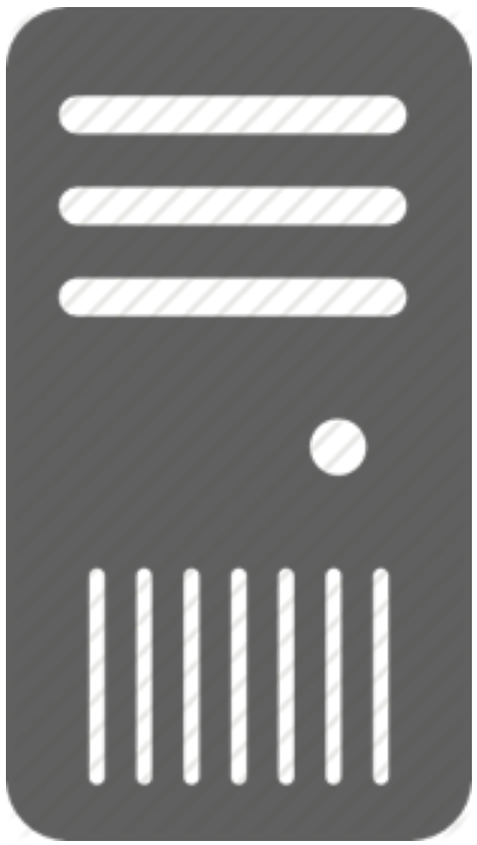
Traversal	Insertion	Removal

Experimental Environment

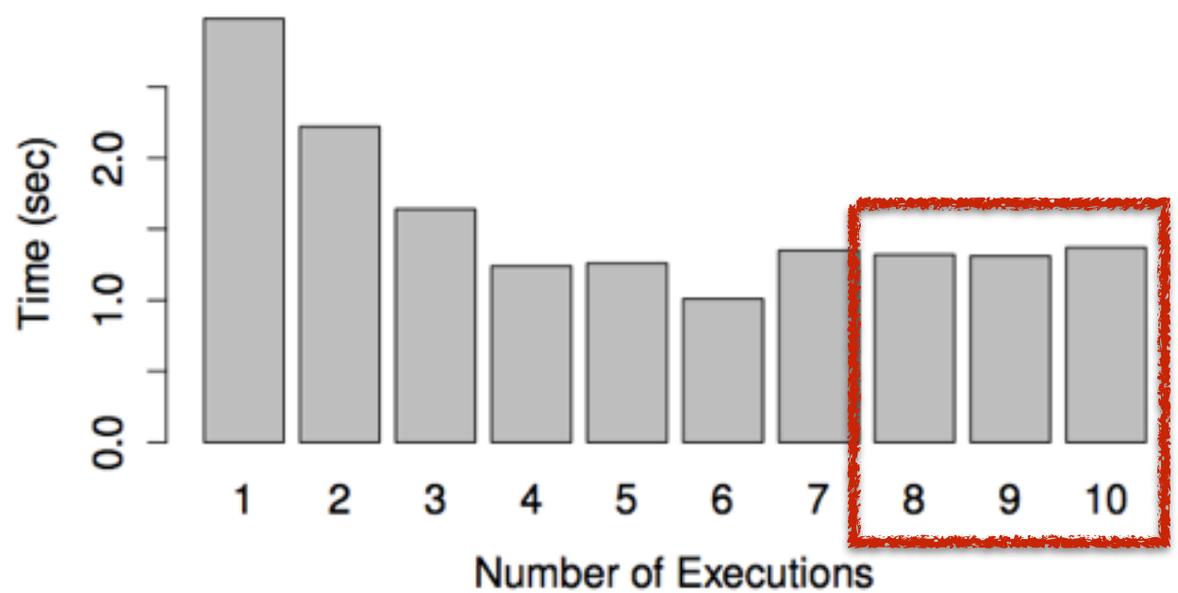


A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21.

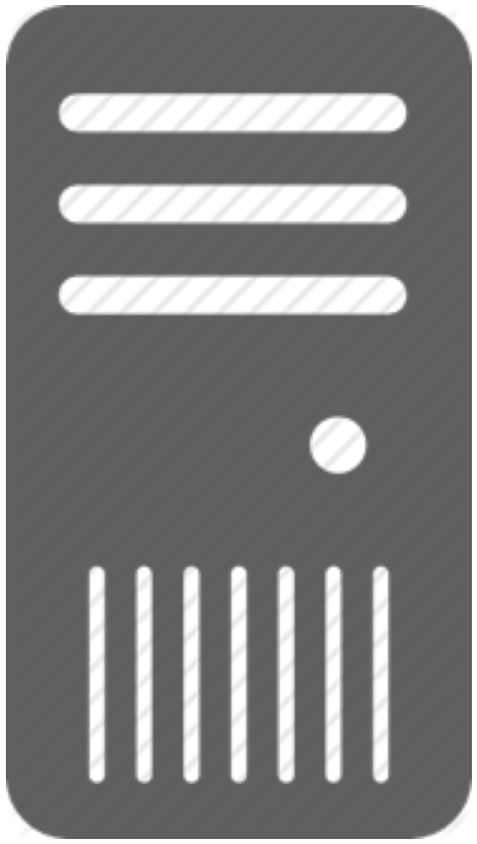
Experimental Environment



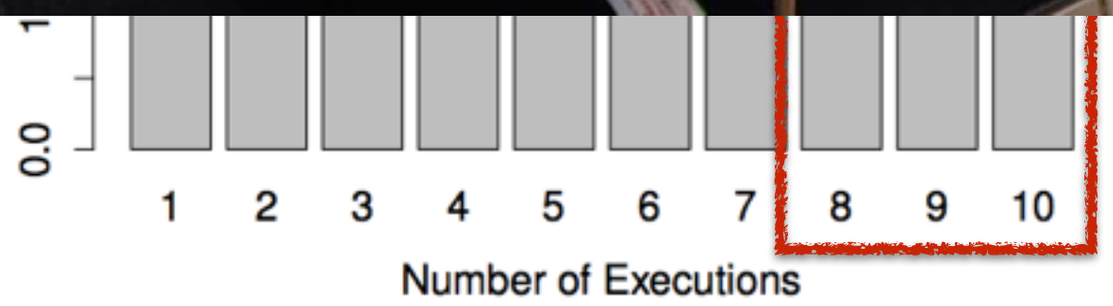
A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21.



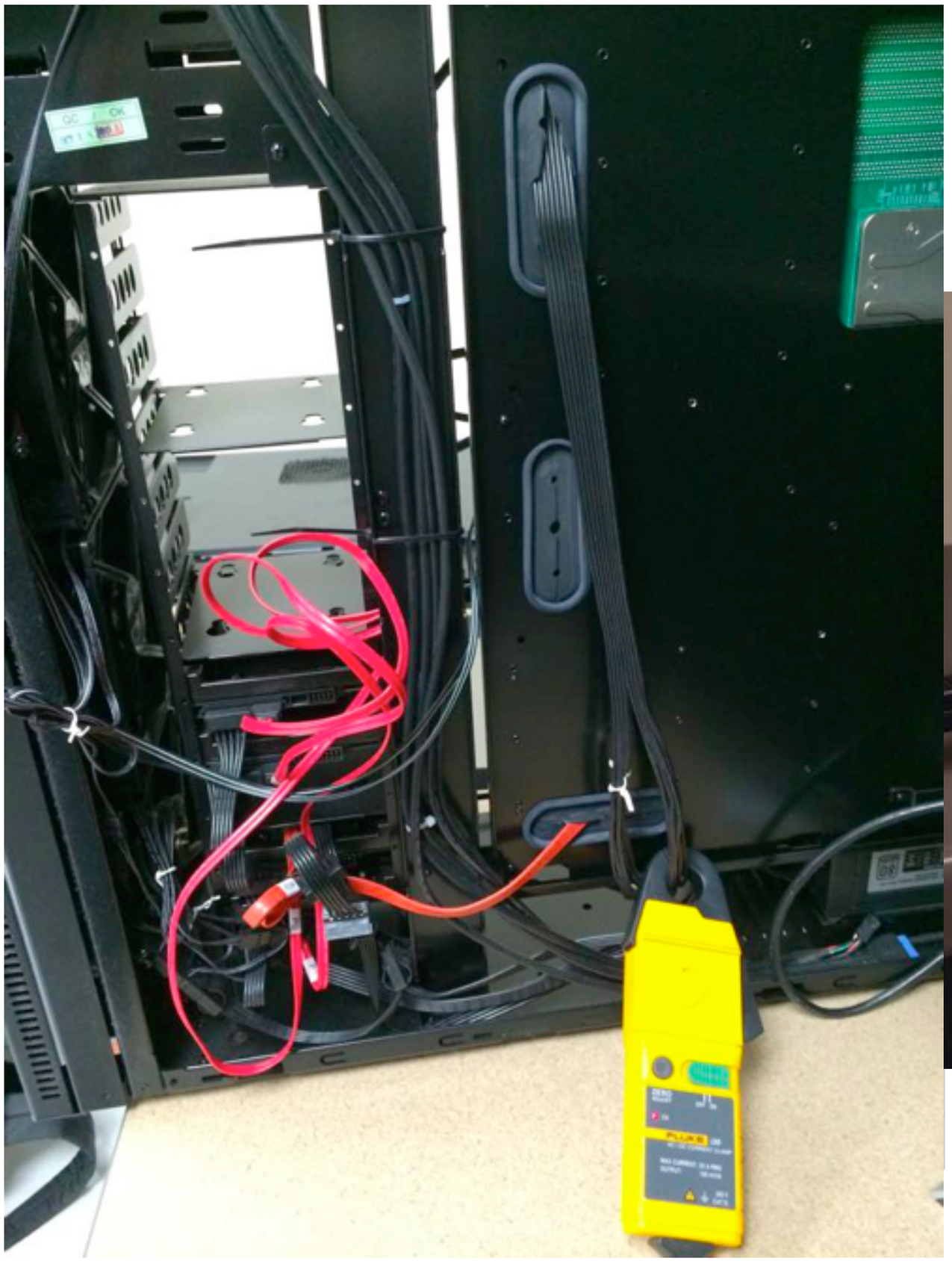
Experimental Environment

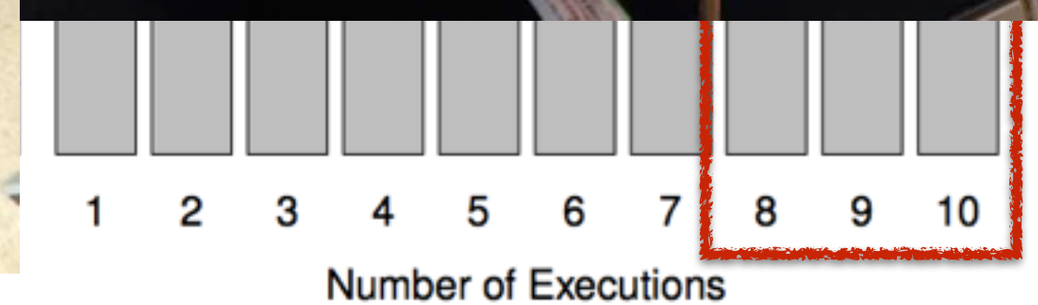


A 2×16-core
Linux, 64GB
11, build 21



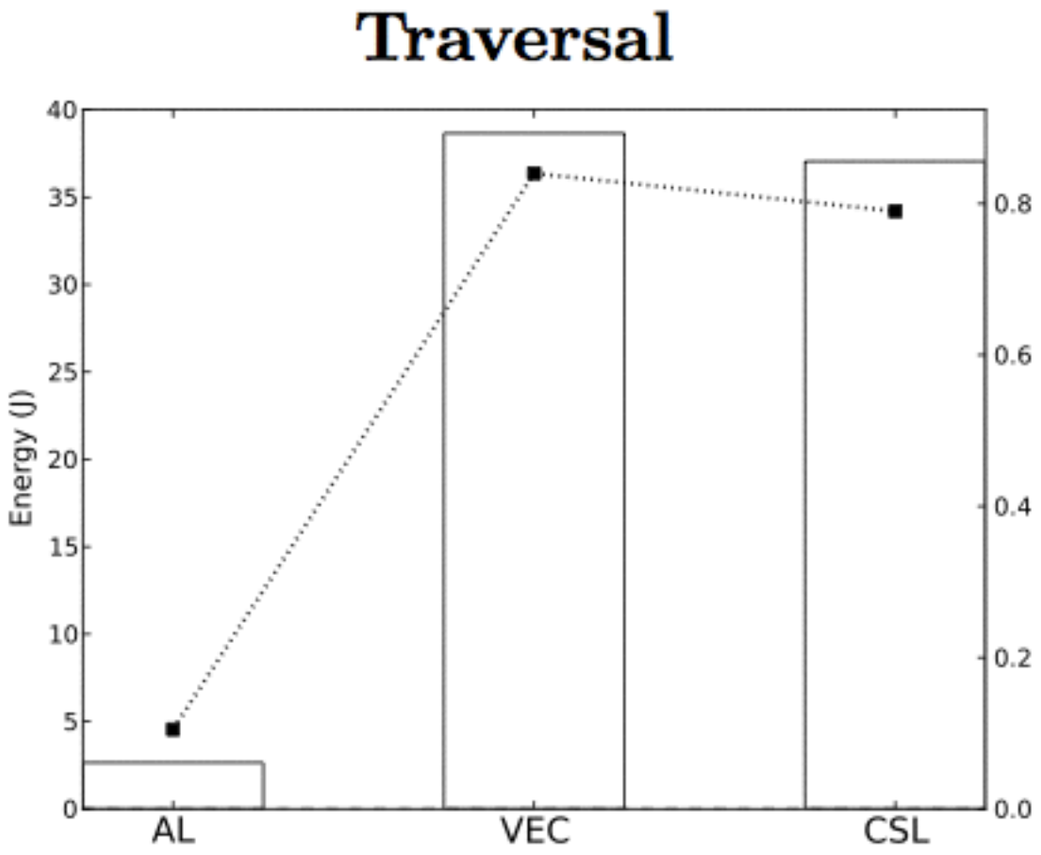
l Environment

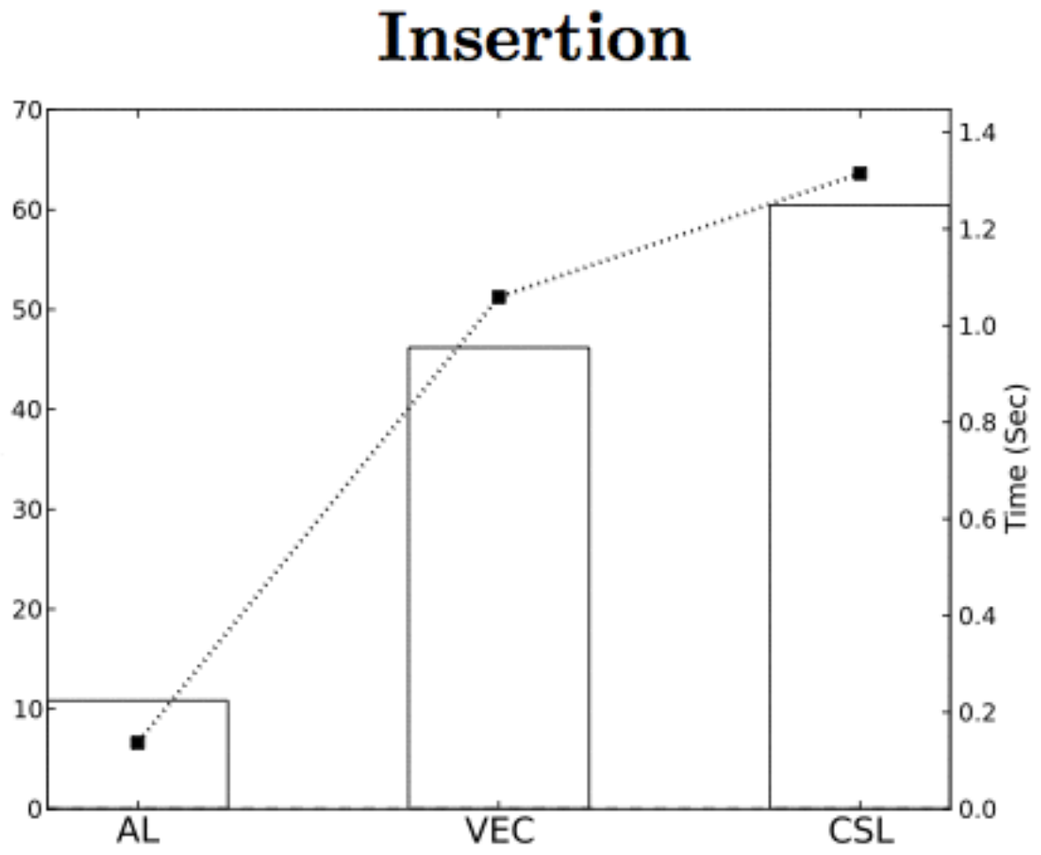




Time

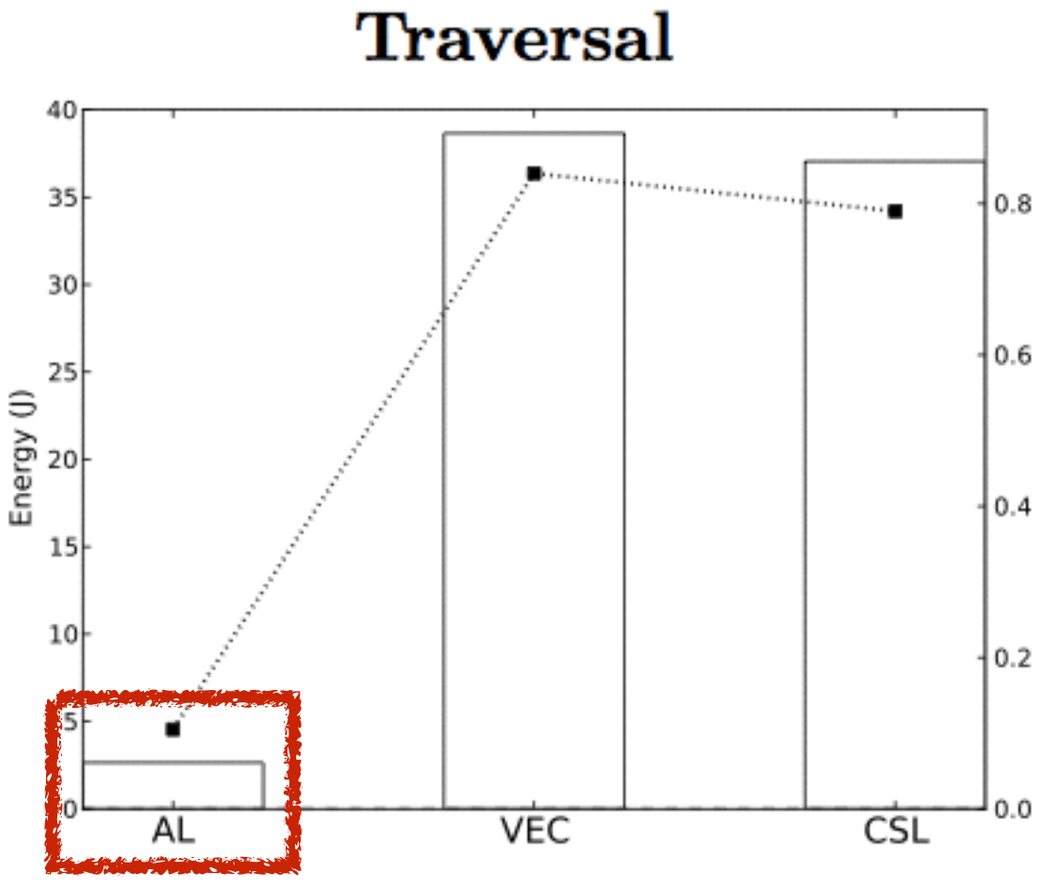
List

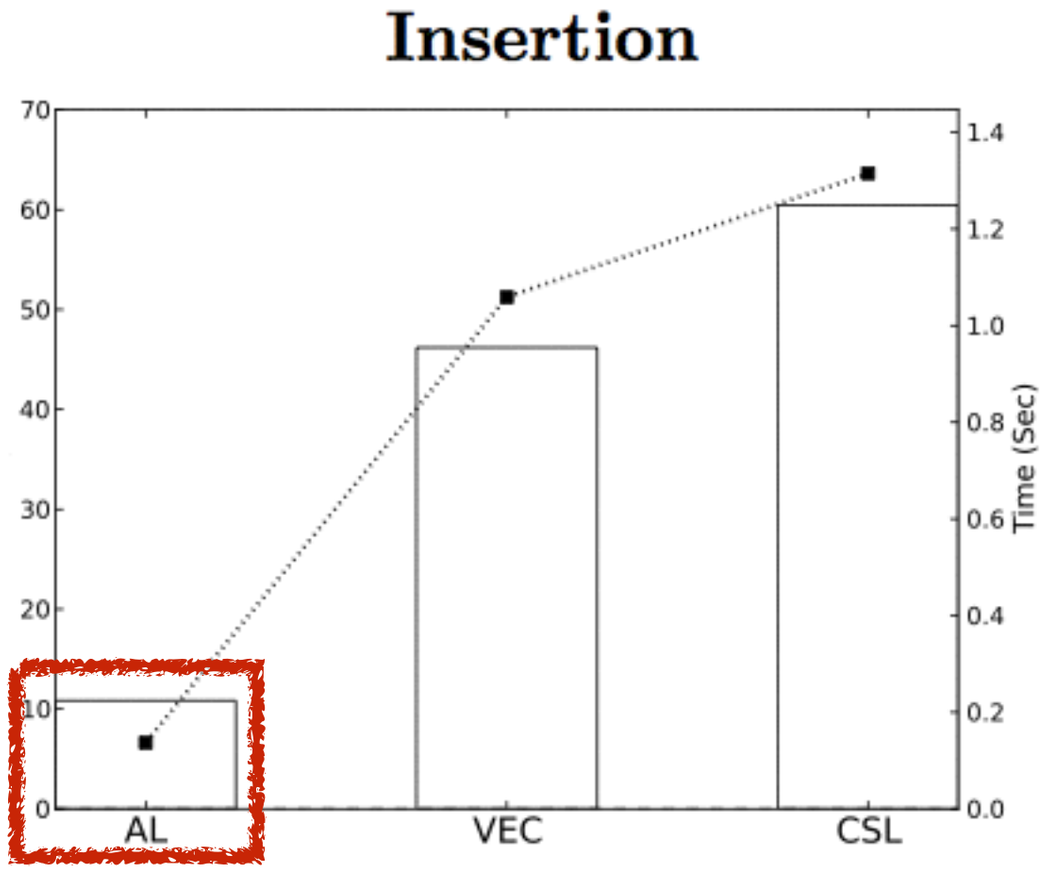

Traversal
Energy (J)
Time (Sec)
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
AL
VEC
CSL


Insertion
Time (Sec)
0
10
20
30
40
50
60
70
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
AL
VEC
CSL

Time

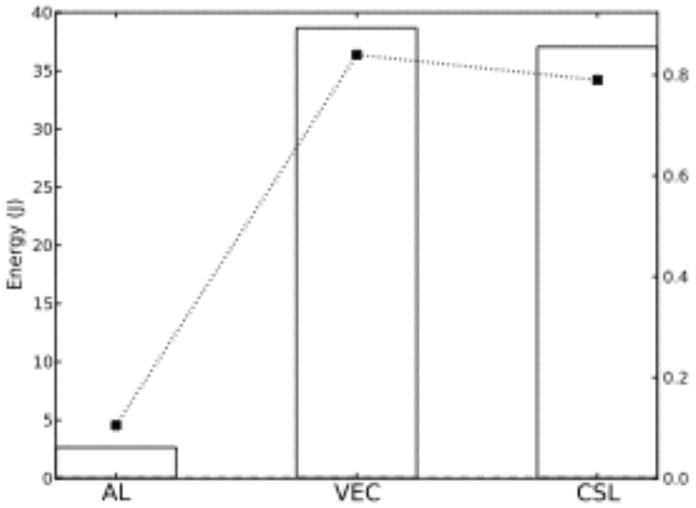
List


Traversal
Energy (J)
Time (Sec)
AL
VEC
CSL


Insertion
Time (Sec)
AL
VEC
CSL

Time

List



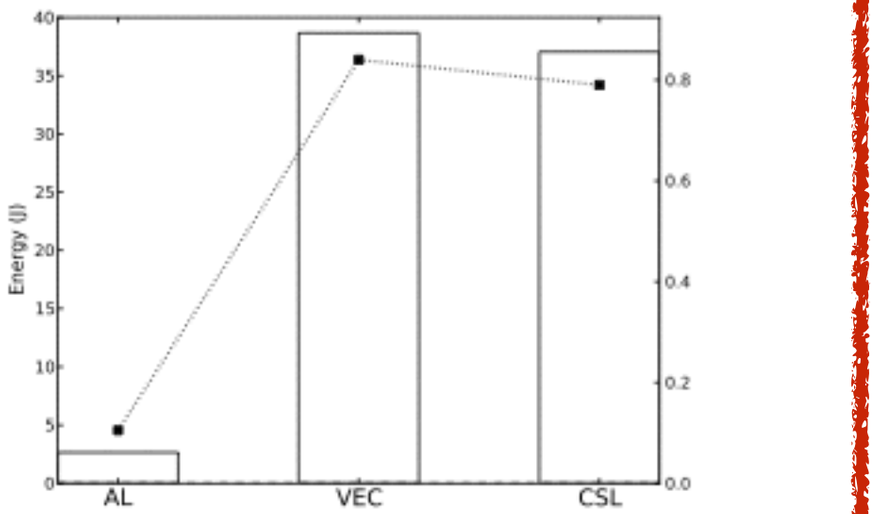

```
List<Object> lists = new ArrayList<>();
int size = lists.size();
for (int i = 0; i < size; i++) {
    lists.get(i);
}
```

```
List<Object> lists = new ArrayList<>();
for (int i = 0; i < lists.size(); i++) {
    lists.get(i);
}
```

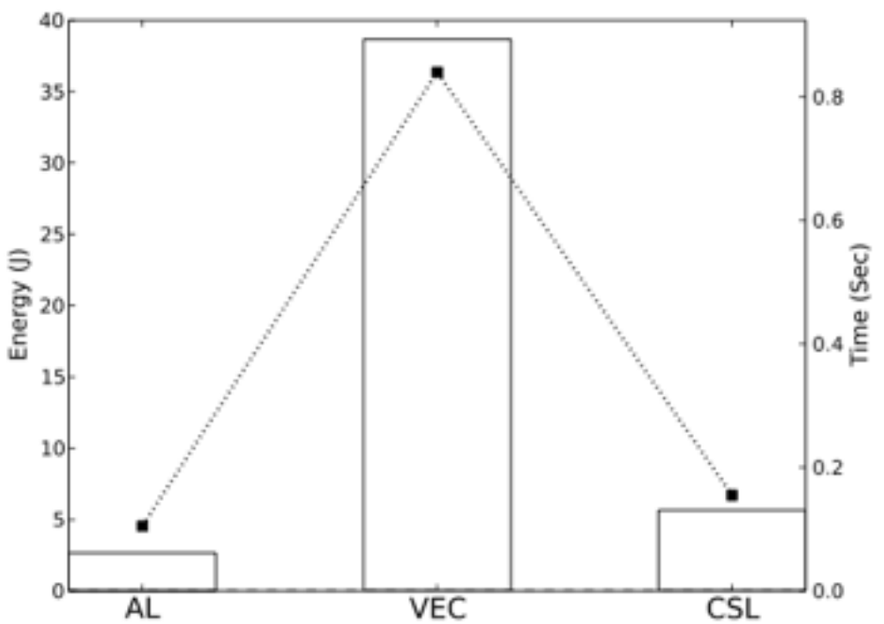
1.98x more energy!

Time

List



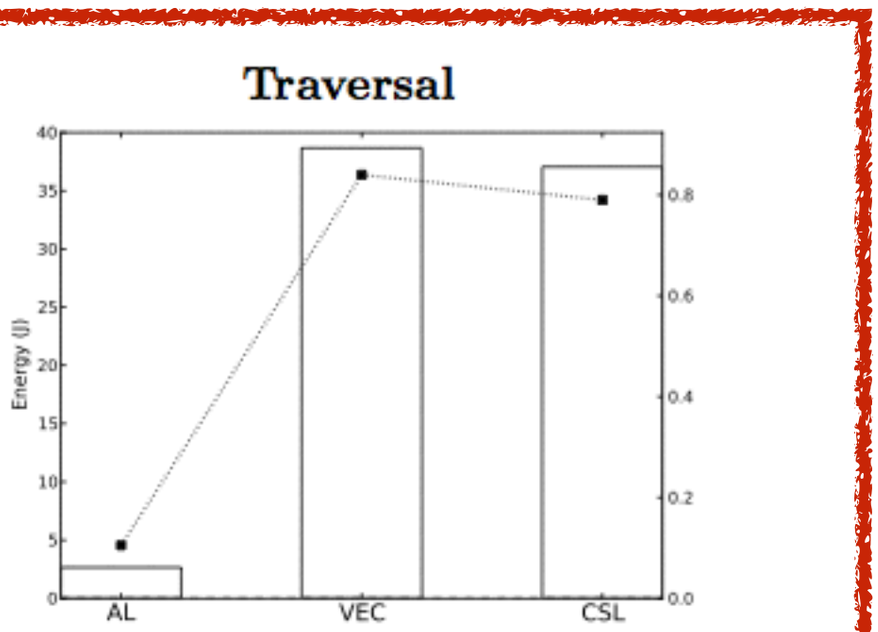

```
List<Object> lists = new ArrayList<>();
int size = lists.size();
for (int i = 0; i < size; i++) {
    lists.get(i);
}
```



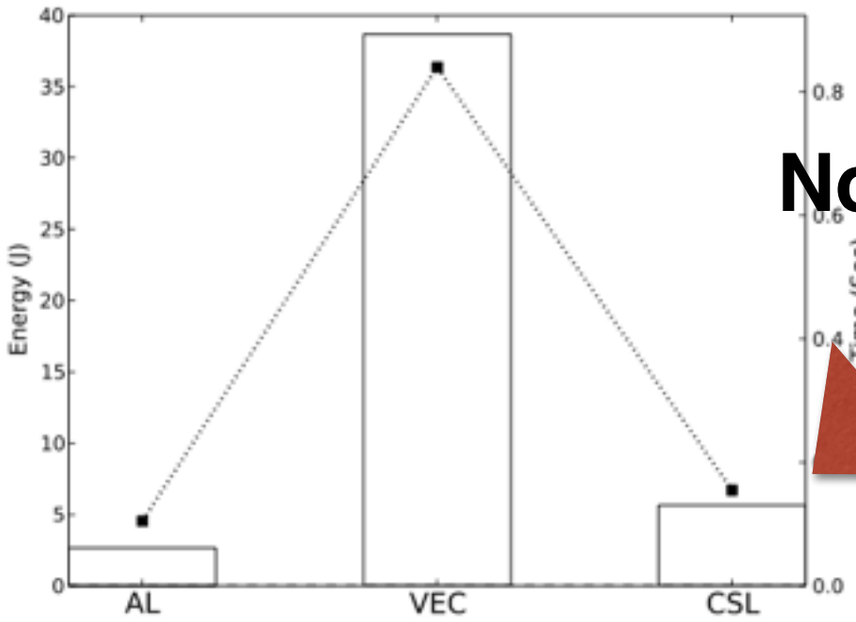

```
List<Object> lists = new ArrayList<>();
for (Object o: lists) {
    // do stuff
}
```

Time

List




```
List<Object> lists = new ArrayList<>();
int size = lists.size();
for (int i = 0; i < size; i++) {
    lists.get(i);
}
```



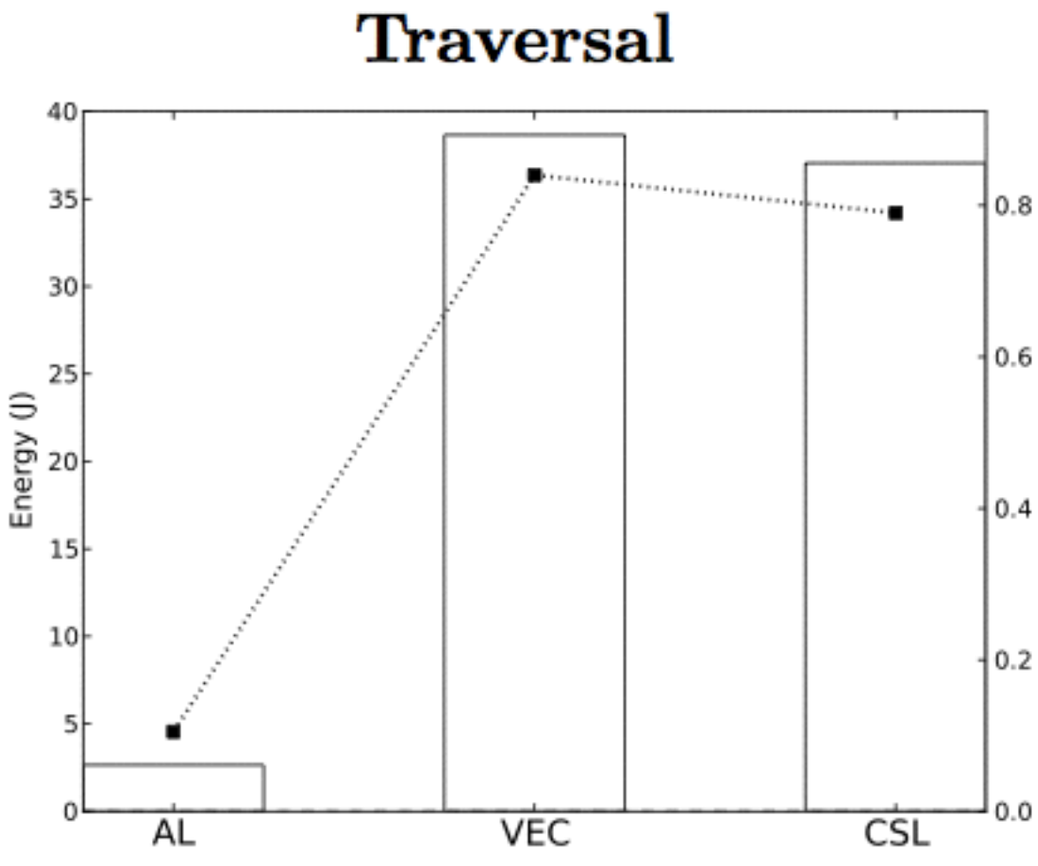
No sync!

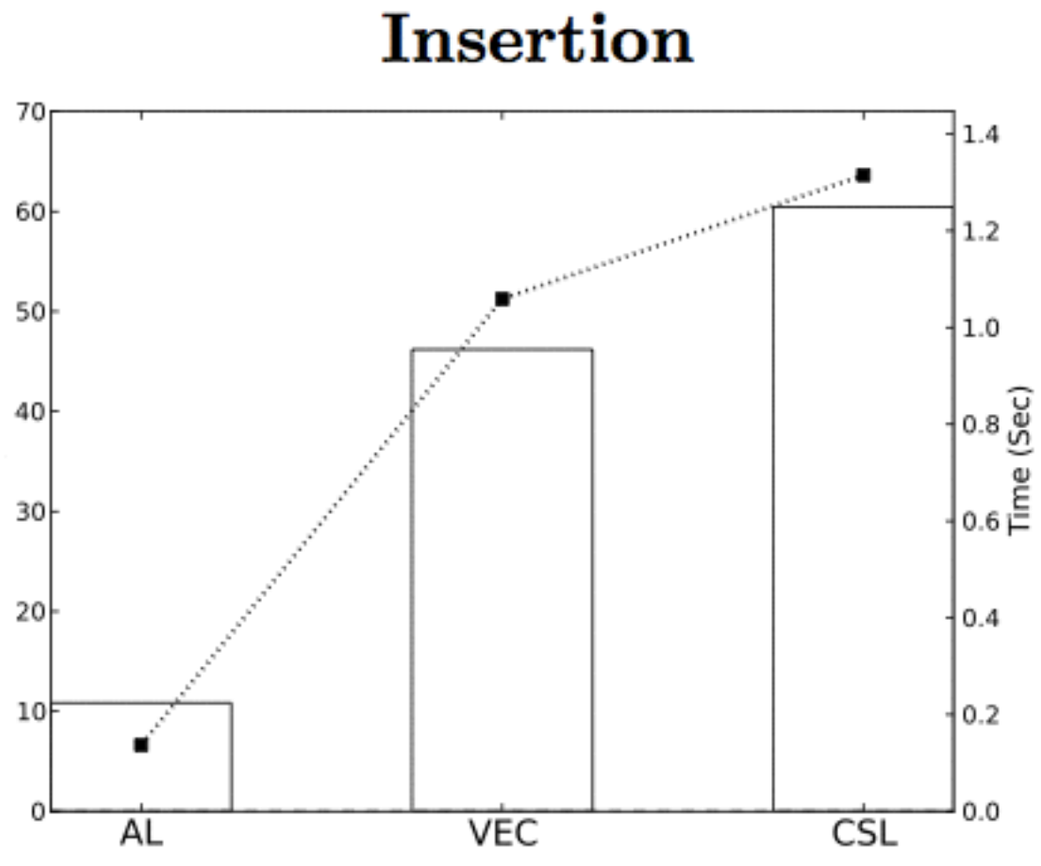
```
List<Object> lists = new ArrayList<>();
for (Object o: lists) {
    // do stuff
}
```

Time ■·····■

List

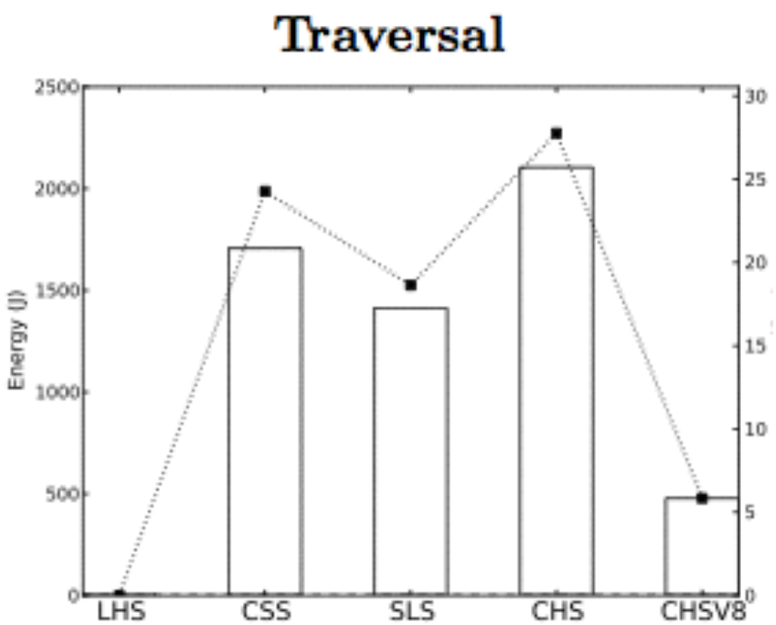
CopyOnWriteArrayList: +152x

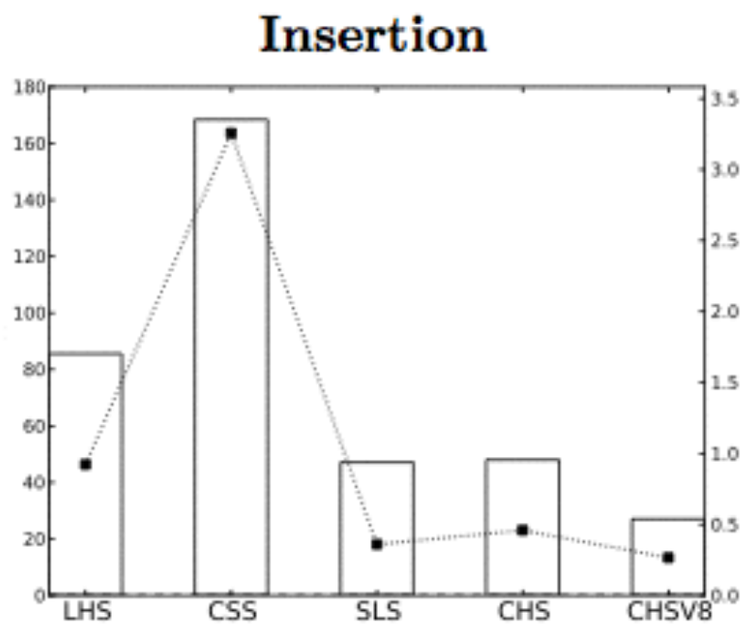


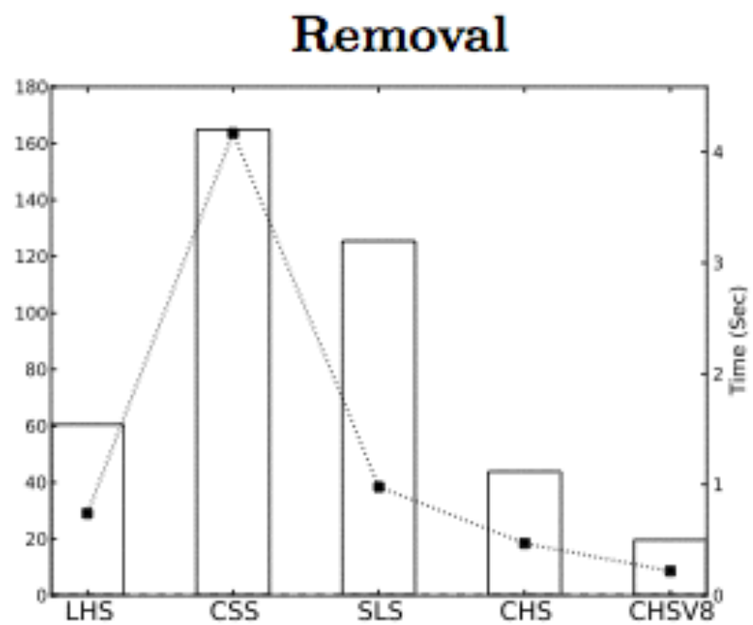


Time

Set

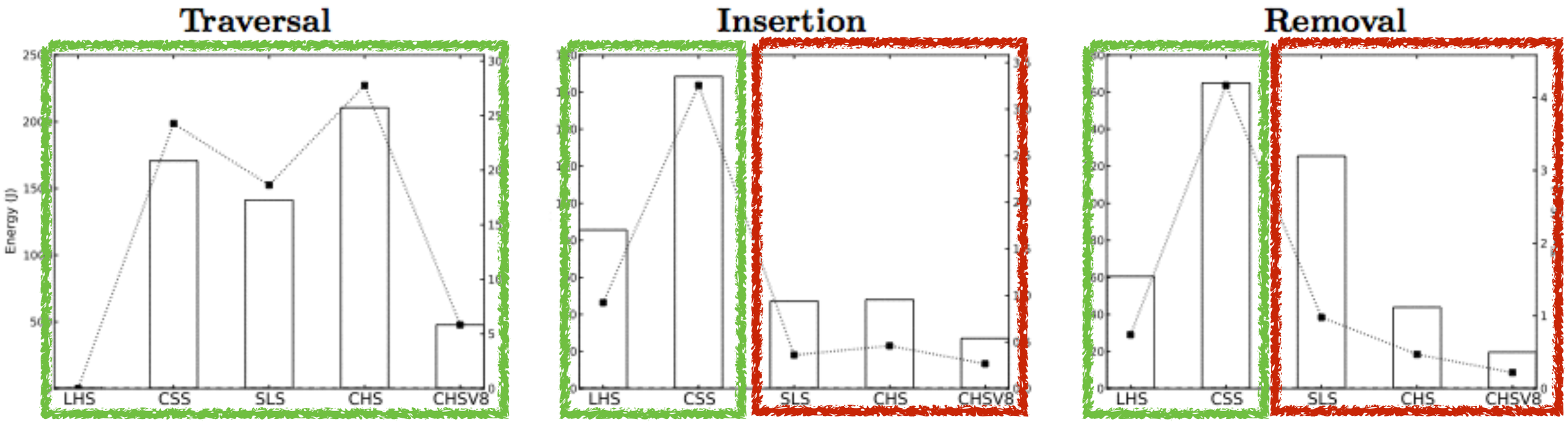

Traversal
Energy (J)
2500
2000
1500
1000
500
0
30
25
20
15
10
5
0
LHS
CSS
SLS
CHS
CHSV8


Insertion
180
160
140
120
100
80
60
40
20
0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
LHS
CSS
SLS
CHS
CHSV8


Removal
180
160
140
120
100
80
60
40
20
0
4
3
2
1
0
Time (Sec)
LHS
CSS
SLS
CHS
CHSV8

Time

Set

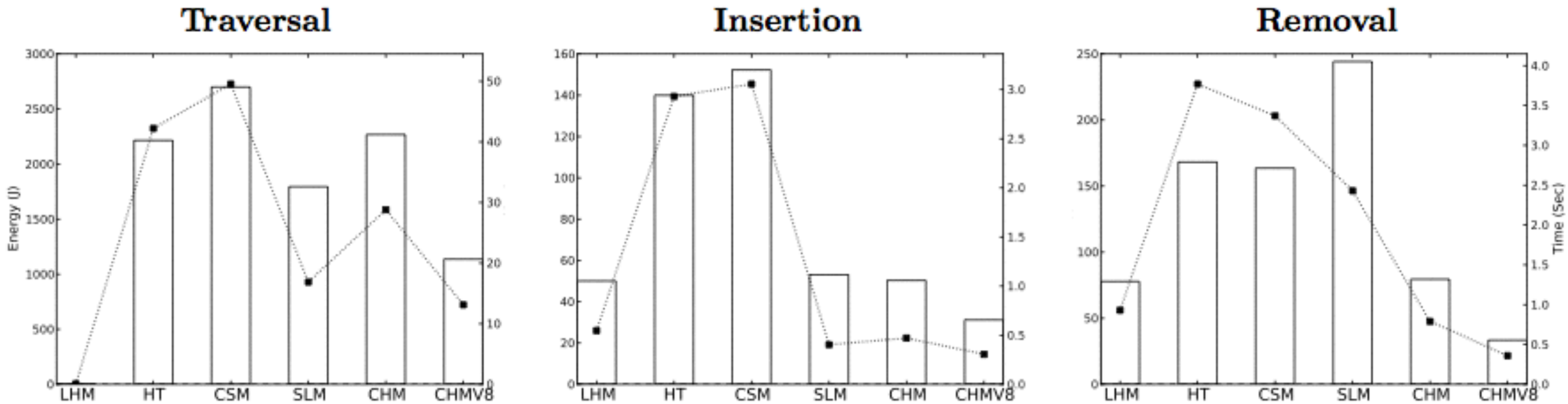


Time **is** proportional to energy

Time **is not** proportional to energy

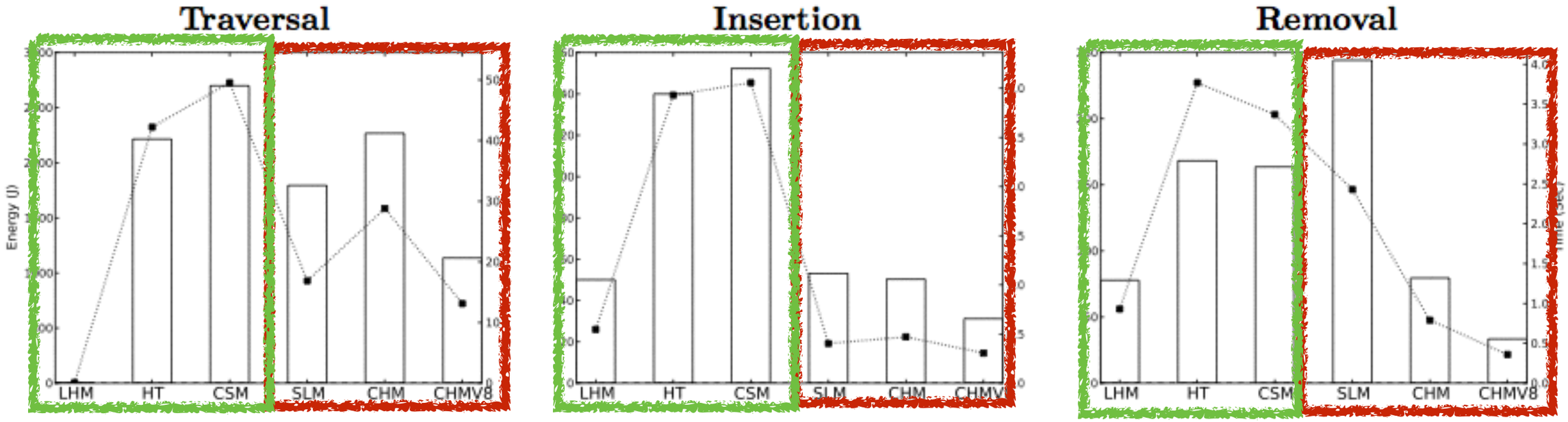
Time

Map


Traversal
Insertion
Removal
Energy (J)
Time (Sec)
LHM
HT
CSM
SLM
CHM
CHMV8

Time

Map

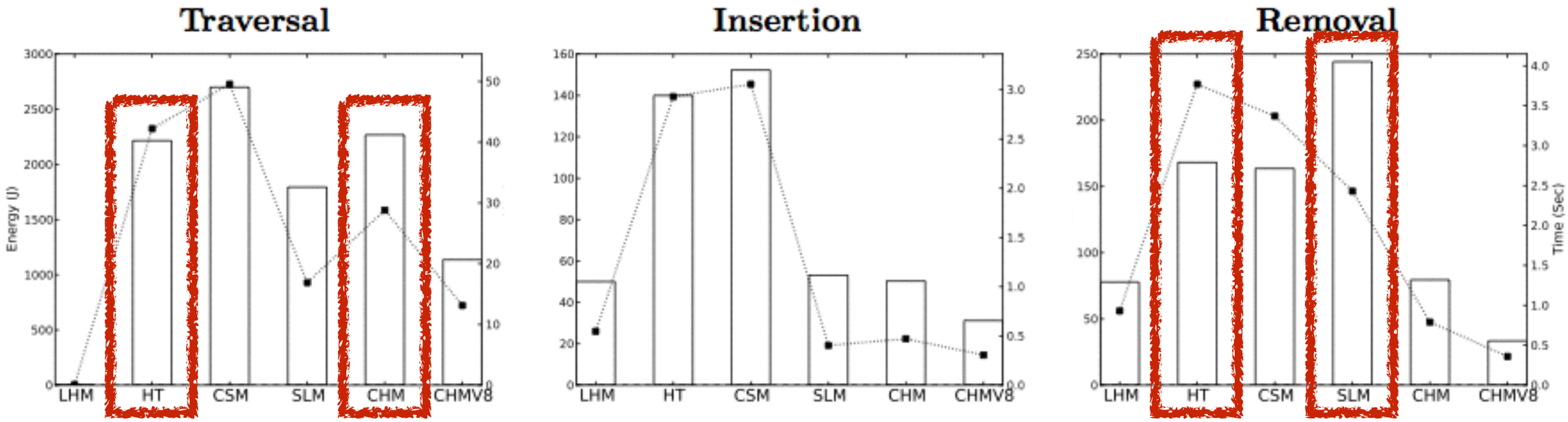


Time **is** proportional to energy

Time **is not** proportional to energy

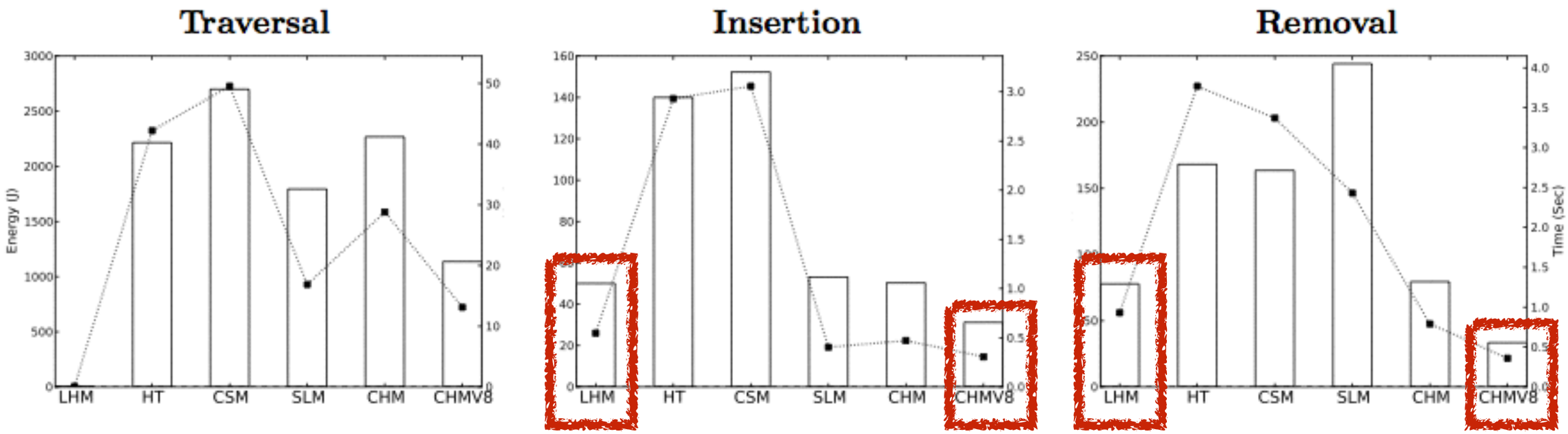
Time

Map


Traversal
Insertion
Removal
Energy (J)
Time (Sec)
LHM
HT
CSM
SLM
CHM
CHMV8

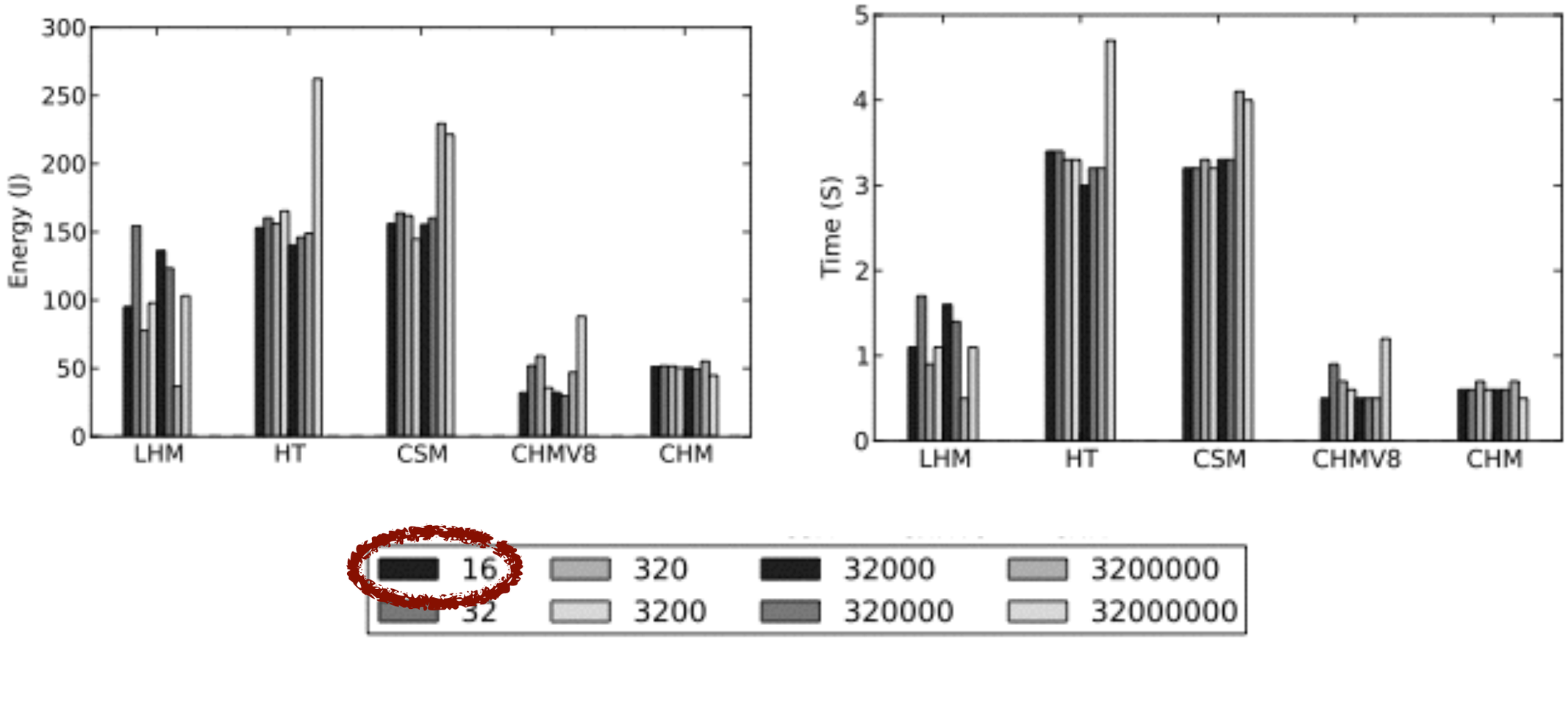
Time

Map


Traversal
Insertion
Removal
Energy (J)
Time (Sec)
LHM
HT
CSM
SLM
CHM
CHMV8

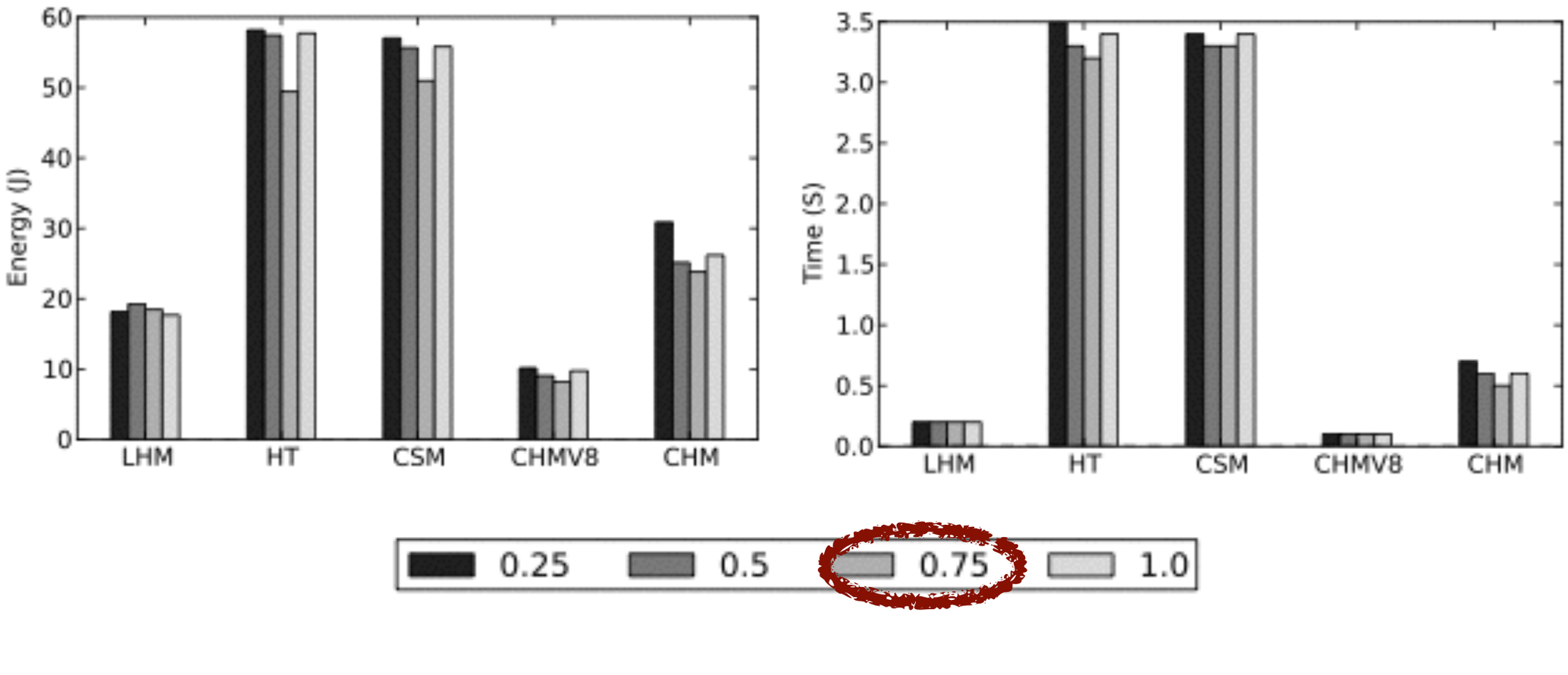
Map “Tuning Knobs”

Initial capacity



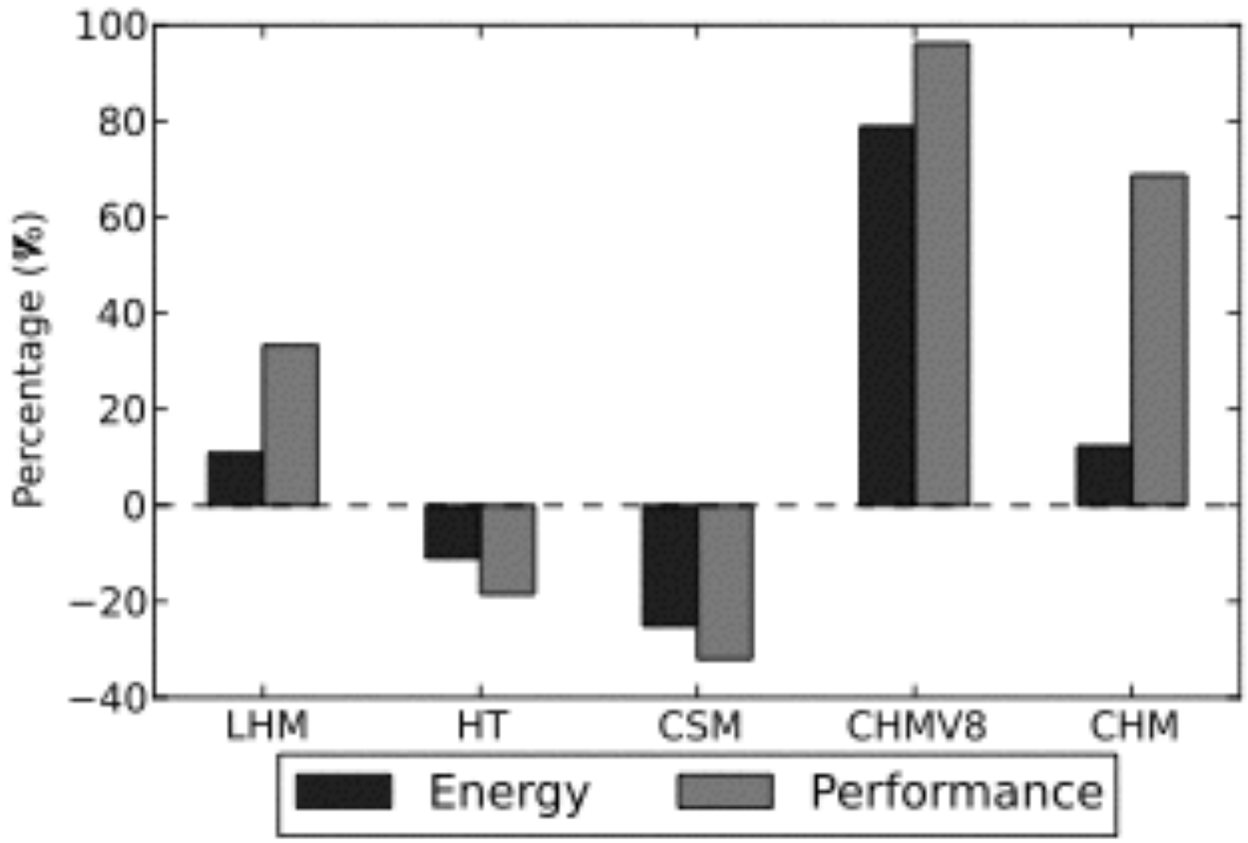
Map “Tuning Knobs”

Load factor



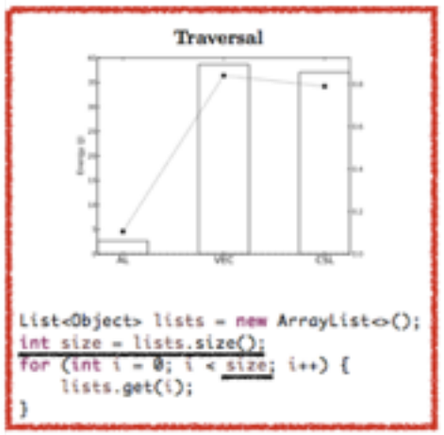
Map “Tuning Knobs”

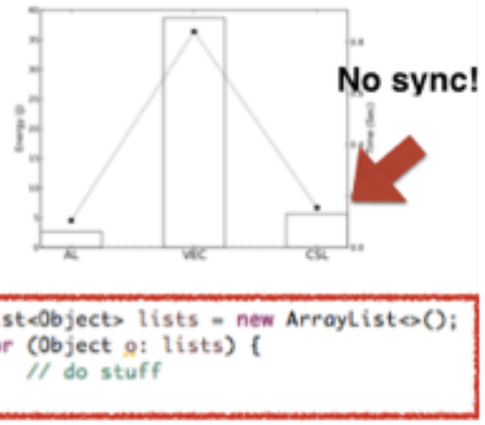
Collision 100%



Time

List

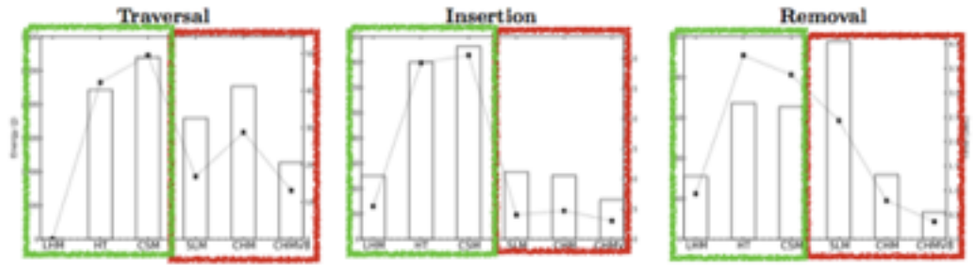




42

Time

Map



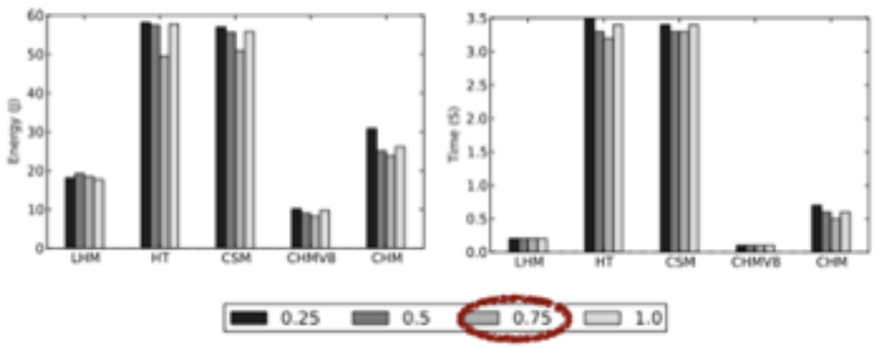
Time **is** proportional to energy

Time **is not** proportional to energy

51

Map “Tuning Knobs”

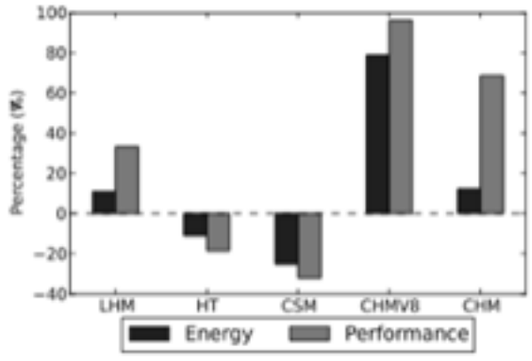
Load factor



53

Map “Tuning Knobs”

Collision 100%



54

Future Work

- Perform the “removal” operations on Lists
- Vary the number of threads accessing the data structure
- Perform the experiments in another machine

Characterizing the Energy Efficiency of Java's Thread-Safe Collections in a Multi-Core Environment

Gustavo Pinto
ghlp@cin.ufpe.br

Fernando Castor
castor@cin.ufpe.br

VIRTUS IMPAVIDA

ines

SPG
SOFTWARE·PRODUCTIVITY·GROUP

Centro de Informática
UFPE