

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



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)

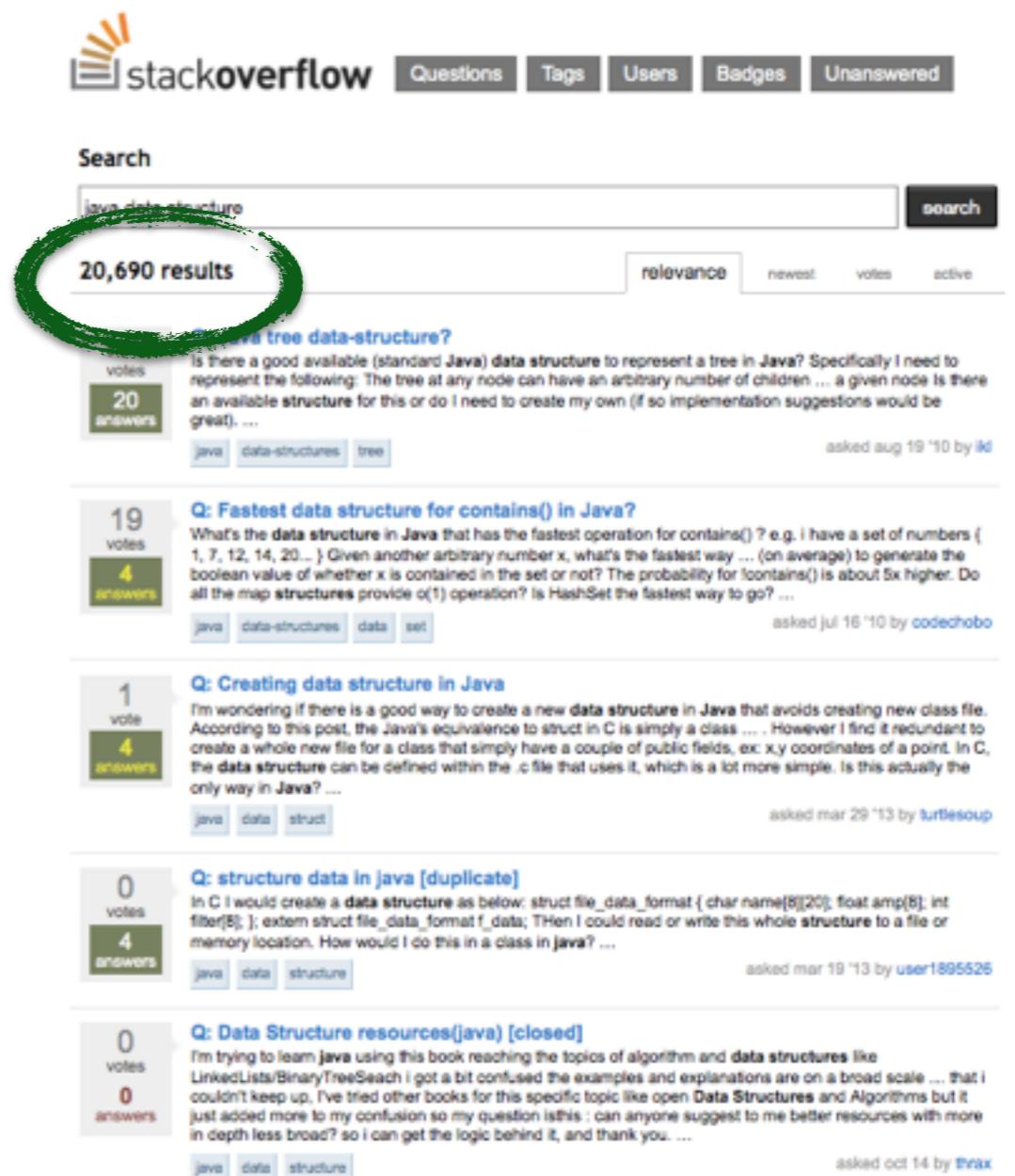


- **First, multicore** CPUs are ubiquitous
- **Second,** more cores used → more power consumed
- **However,** little is known about the energy-efficiency of multicore programs



Motivation (3/3)

- Data structures are the fundamentals of computer programming



The screenshot shows the Stack Overflow search interface. The search bar contains the query "java data-structure". Below the search bar, it displays "20,690 results". A green circle highlights this result count. The search results list several questions related to Java data structures:

- Q: Java tree data-structure?**
Is there a good available (standard Java) data structure to represent a tree in Java? Specifically I need to represent the following: The tree at any node can have an arbitrary number of children ... a given node is there an available structure for this or do I need to create my own (if so implementation suggestions would be great). ...
20 answers
- Q: Fastest data structure for contains() in Java?**
What's the data structure in Java that has the fastest operation for contains() ? e.g. I have a set of numbers { 1, 7, 12, 14, 20... } Given another arbitrary number x, what's the fastest way ... (on average) to generate the boolean value of whether x is contained in the set or not? The probability for contains() is about 5x higher. Do all the map structures provide o(1) operation? Is HashSet the fastest way to go? ...
4 answers
- Q: Creating data structure in Java**
I'm wondering if there is a good way to create a new data structure in Java that avoids creating new class file. According to this post, the Java's equivalence to struct in C is simply a class However I find it redundant to create a whole new file for a class that simply have a couple of public fields, ex: x,y coordinates of a point. In C, the data structure can be defined within the .c file that uses it, which is a lot more simple. Is this actually the only way in Java? ...
4 answers
- Q: structure data in java [duplicate]**
In C I would create a data structure as below: struct file_data_format { char name[8][20], float amp[8], int filter[8]; }; extern struct file_data_format f_data; Then I could read or write this whole structure to a file or memory location. How would I do this in a class in java? ...
4 answers
- Q: Data Structure resources(java) [closed]**
I'm trying to learn java using this book reaching the topics of algorithm and data structures like LinkedLists/BinaryTreeSearch I got a bit confused the examples and explanations are on a broad scale ... that I couldn't keep up. I've tried other books for this specific topic like open Data Structures and Algorithms but it just added more to my confusion so my question is this : can anyone suggest to me better resources with more in depth less broad? so I can get the logic behind it, and thank you.
0 answers

```
List<Object> lists = ...;
```

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

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

Thread →



Not thread safe!

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

A diagram illustrating thread safety issues. It shows a variable declaration: 'List<Object> lists = new ArrayList<>();'. Above the declaration, several blue hand-drawn style arrows point from different directions towards the 'lists' variable, indicating that multiple threads are accessing the same shared resource.

```
List<Object> lists = ...;
```

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

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

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

Thread →

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

A hand-drawn style diagram consisting of several blue arrows pointing from various directions towards the variable 'lists' in the code. There are approximately six arrows in total, some pointing up, some pointing down, and some pointing diagonally.

Thread →

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



Thread-safe!

Thread →

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



Thread-safe!

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

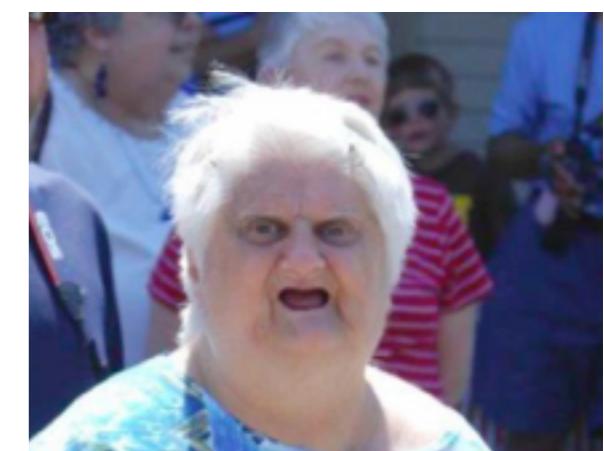
Thread →

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



Thread-safe!

```
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 →



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



Thread-safe!

Thread



Thread-safe!

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

```
public class CopyOnWriteArrayList<E> {  
    private E[] get(Object[] a, int index) {  
        return (E) a[index];  
    }  
}
```

Thread



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

```
public class CopyOnWriteArrayList<E> {  
    private E[] get(Object[] a, int index) {  
        return (E) a[index];  
    }  
}
```



Thread-safe!



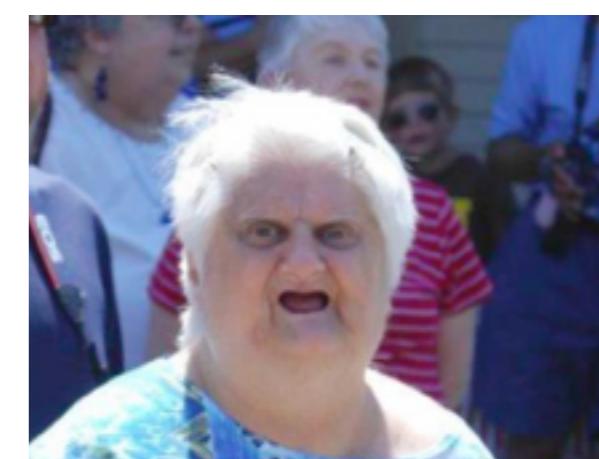
Thread



✓ 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

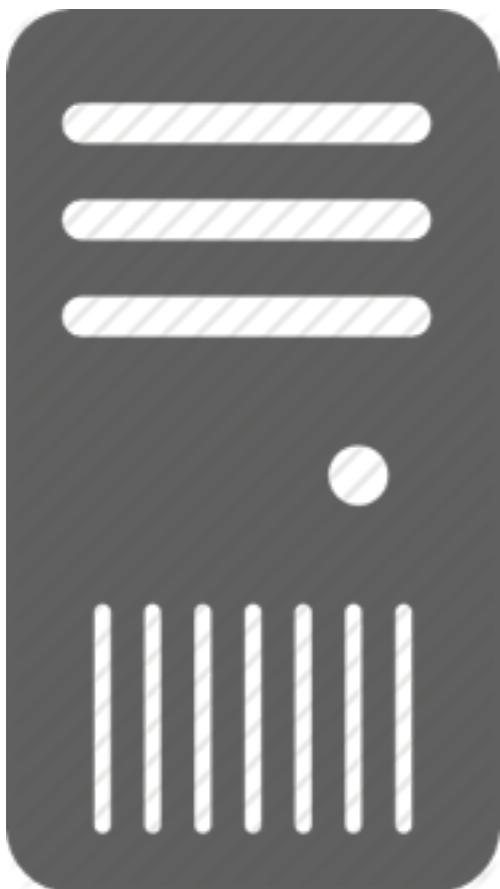
16 Benchmarks

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

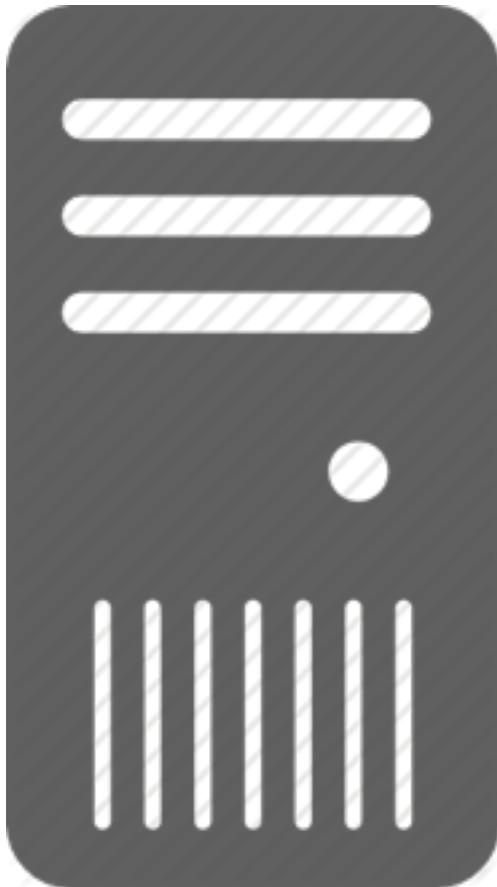
Traversal	Insertion	Removal
-----------	-----------	---------

Experimental Environment

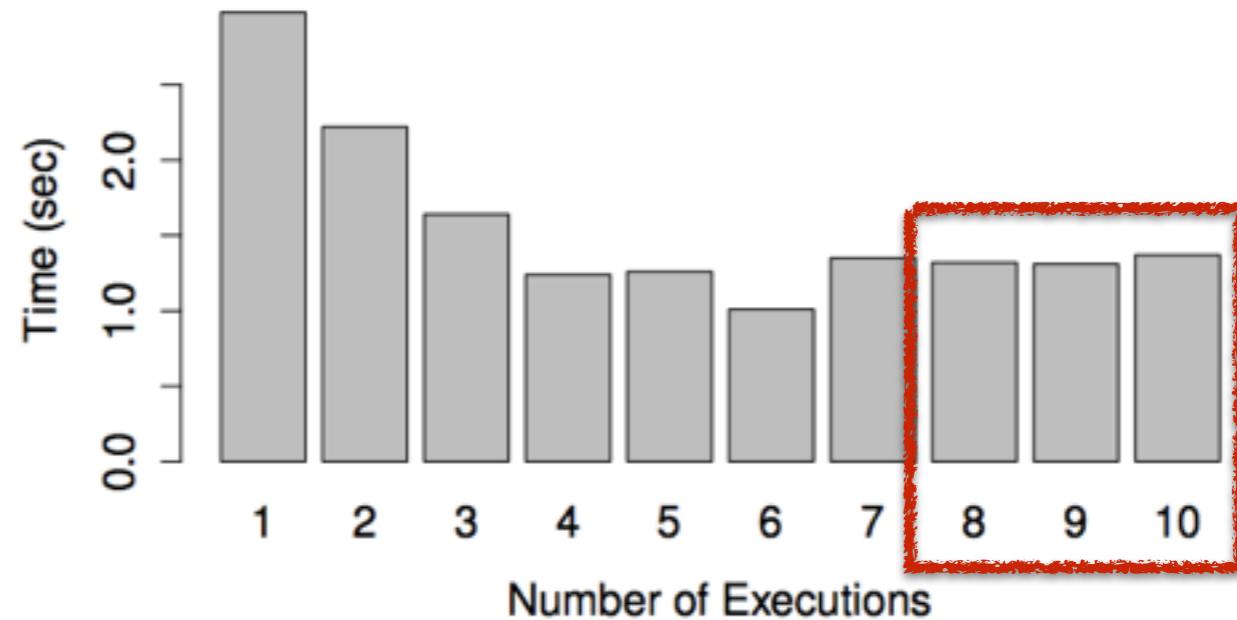


A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21.

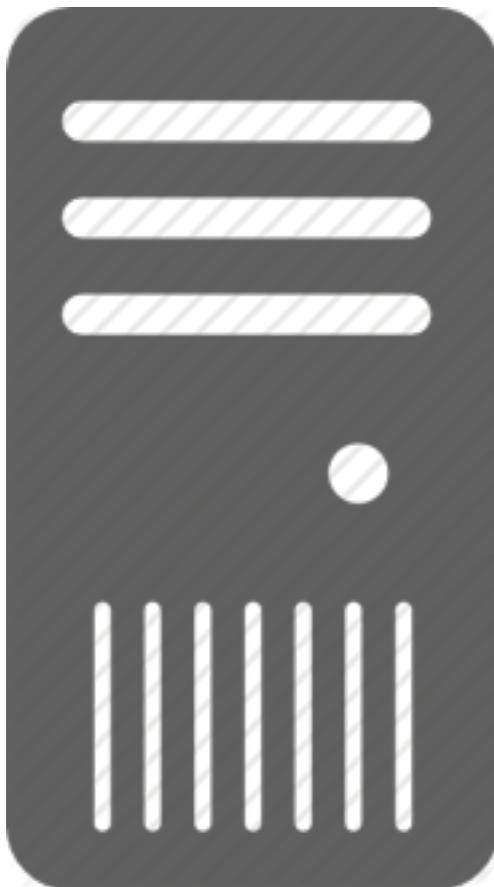
Experimental Environment



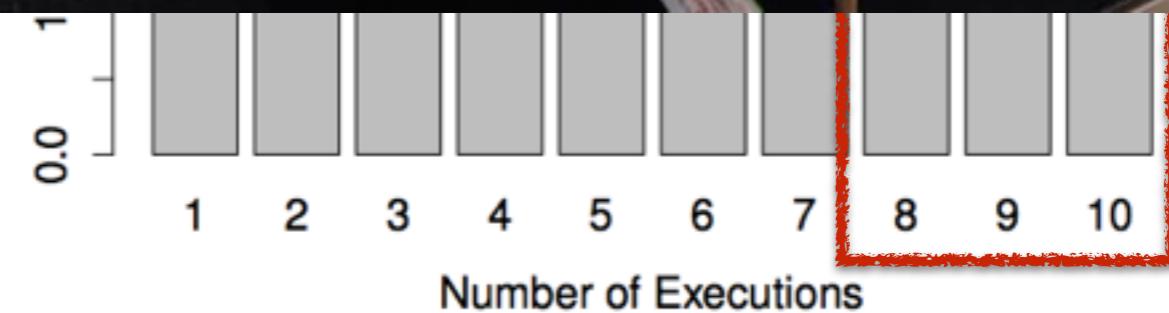
A 2×16-core AMD CPUs, running Debian Linux, 64GB of memory, JDK version 1.7.0 11, build 21.



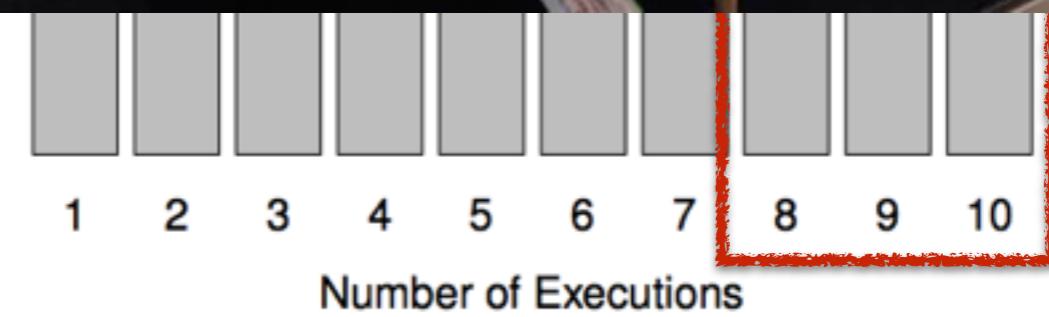
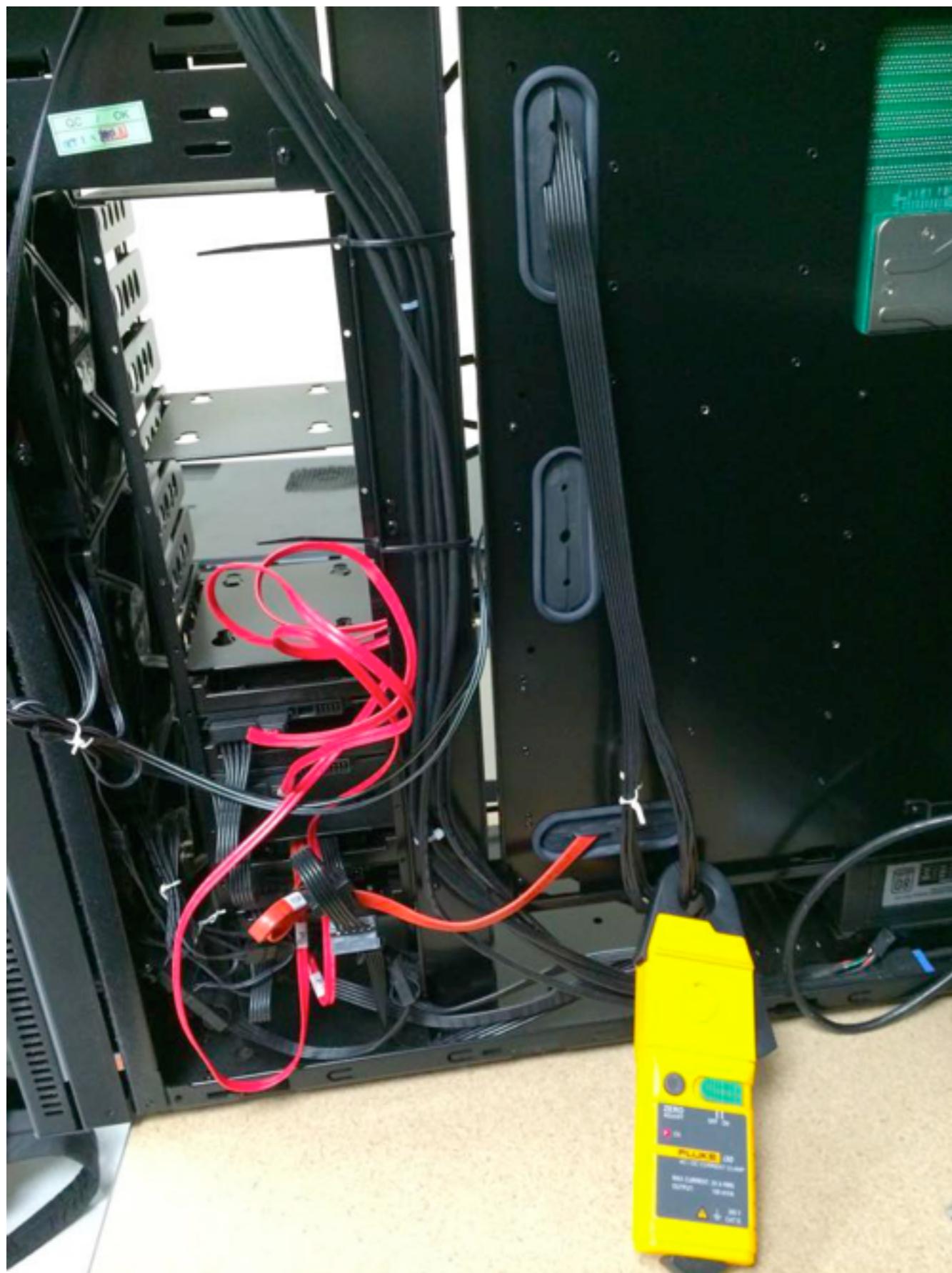
Experimental Environment



A 2×16-core
Linux, 64GB
11, build 21



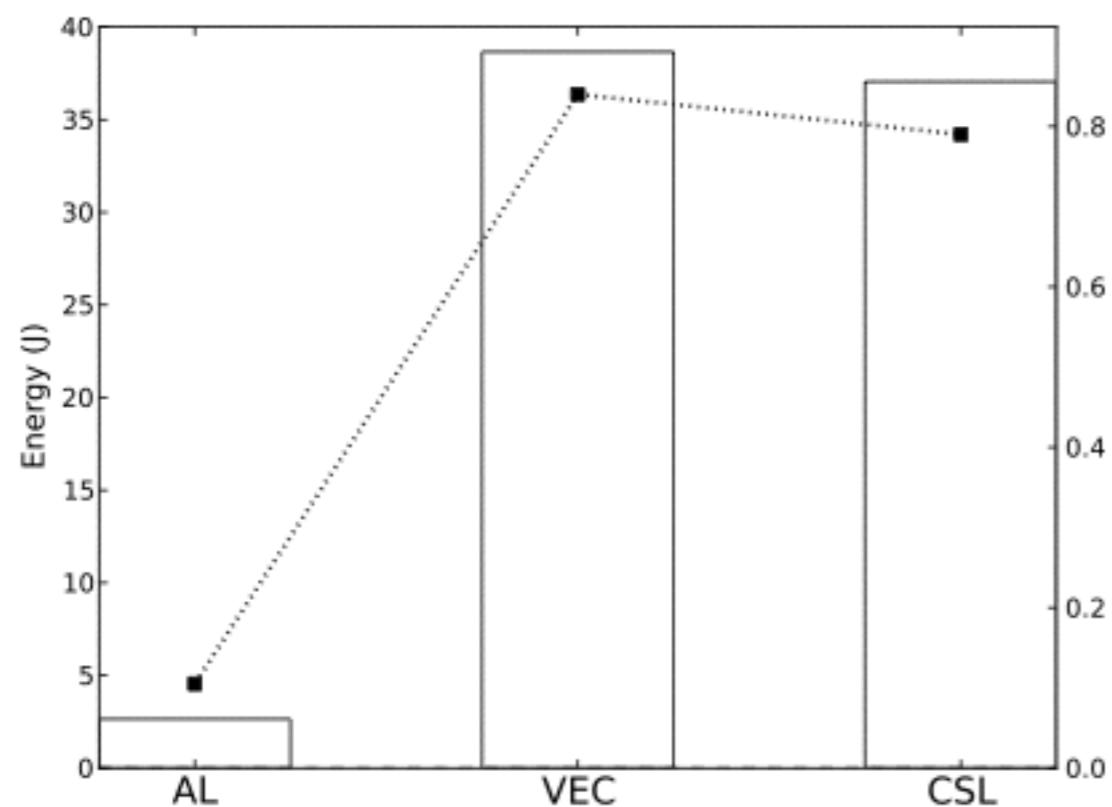
I Environment



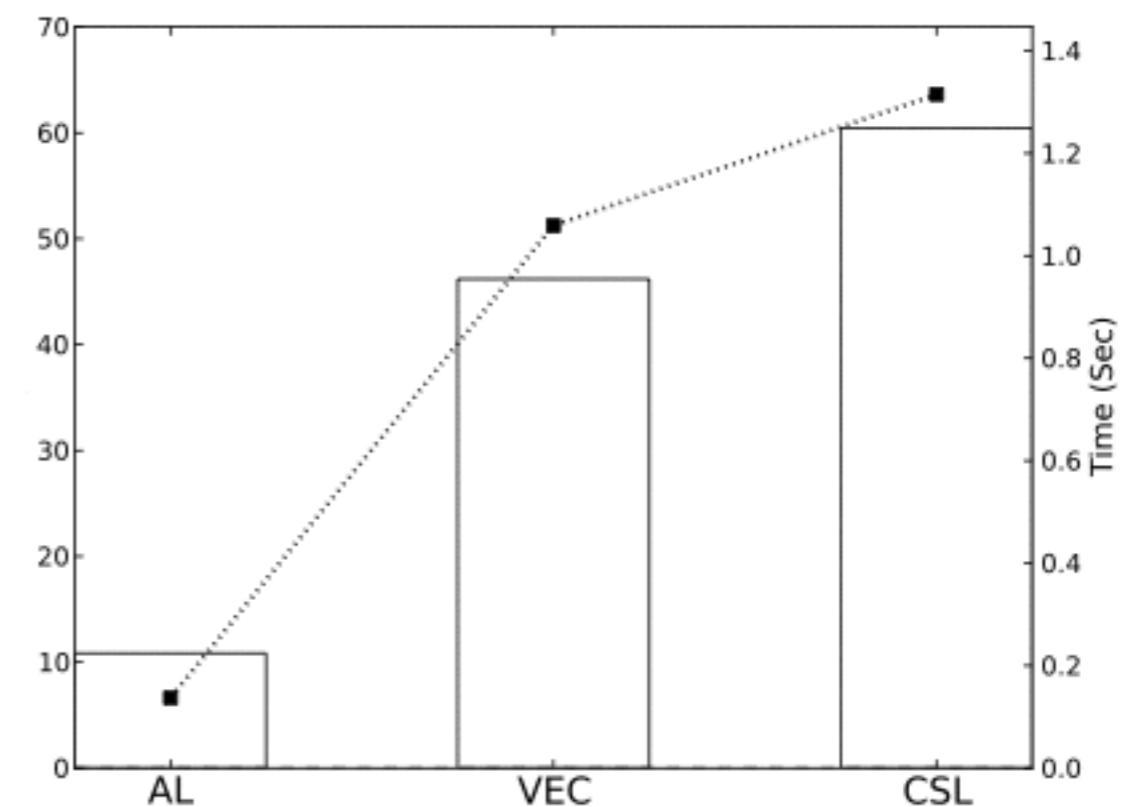
Time

List

Traversal



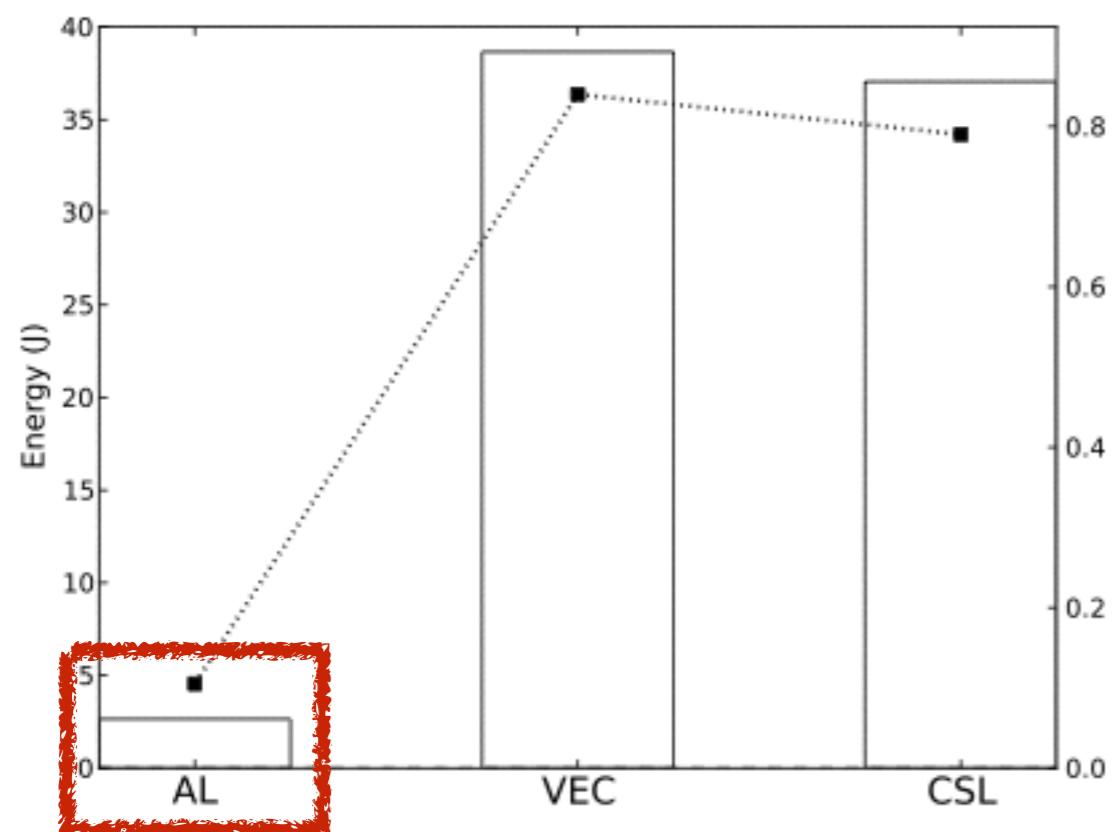
Insertion



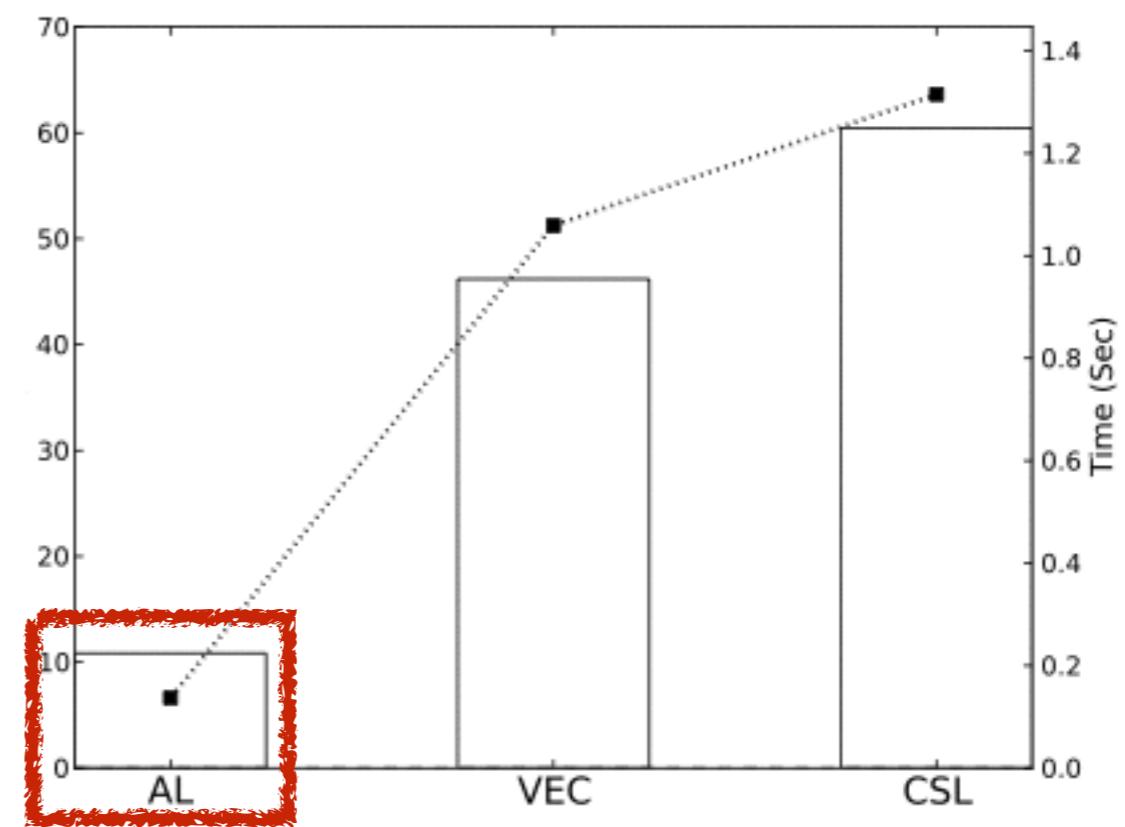
Time

List

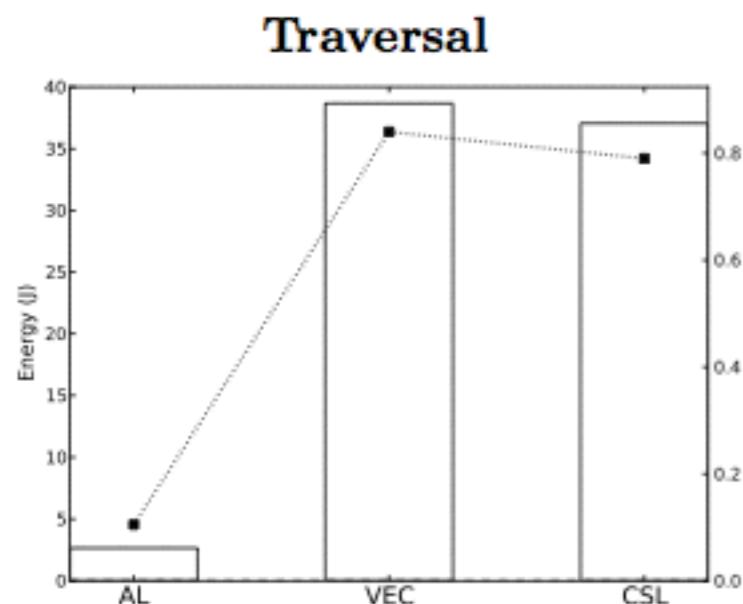
Traversal



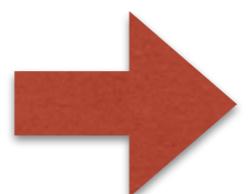
Insertion



List



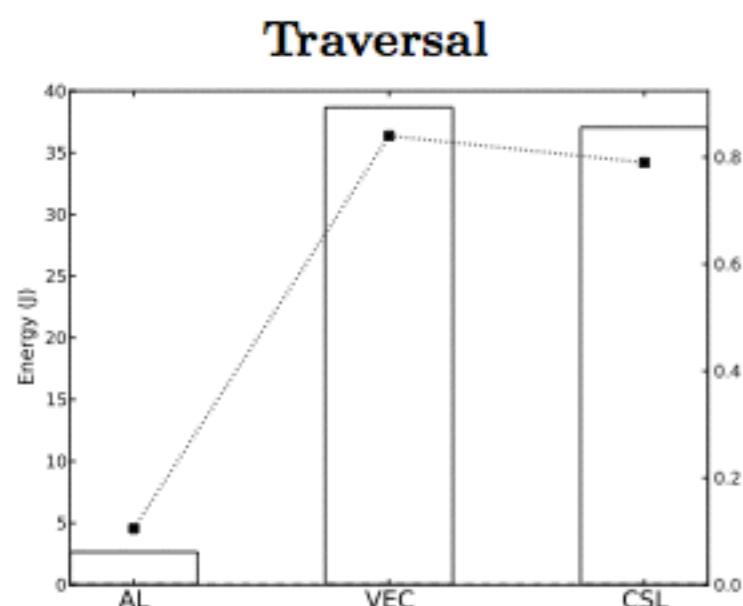
```
List<Object> lists = new ArrayList<>();  
int size = lists.size();  
for (int i = 0; i < size; i++) {  
    lists.get(i);  
}
```



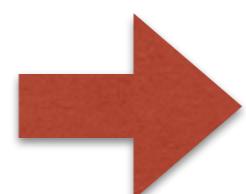
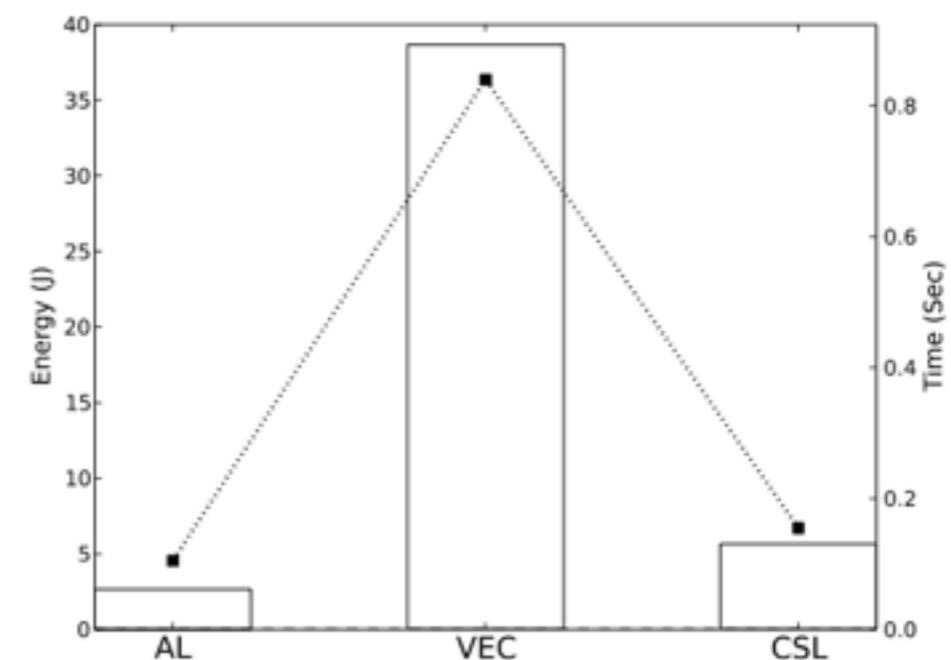
```
List<Object> lists = new ArrayList<>();  
for (int i = 0; i < lists.size(); i++) {  
    lists.get(i);  
}
```

1.98x more energy!

List

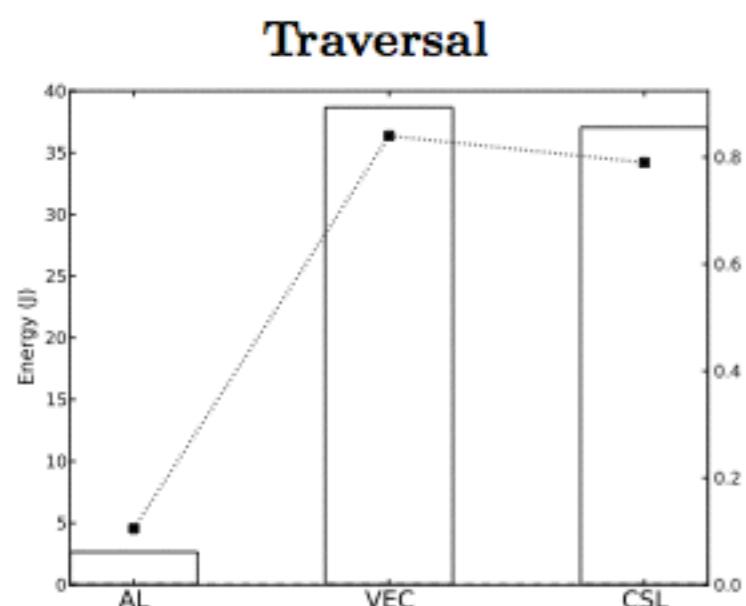


```
List<Object> lists = new ArrayList<>();
int size = lists.size();
for (int i = 0; i < size; i++) {
    lists.get(i);
}
```

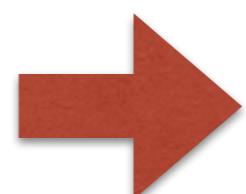
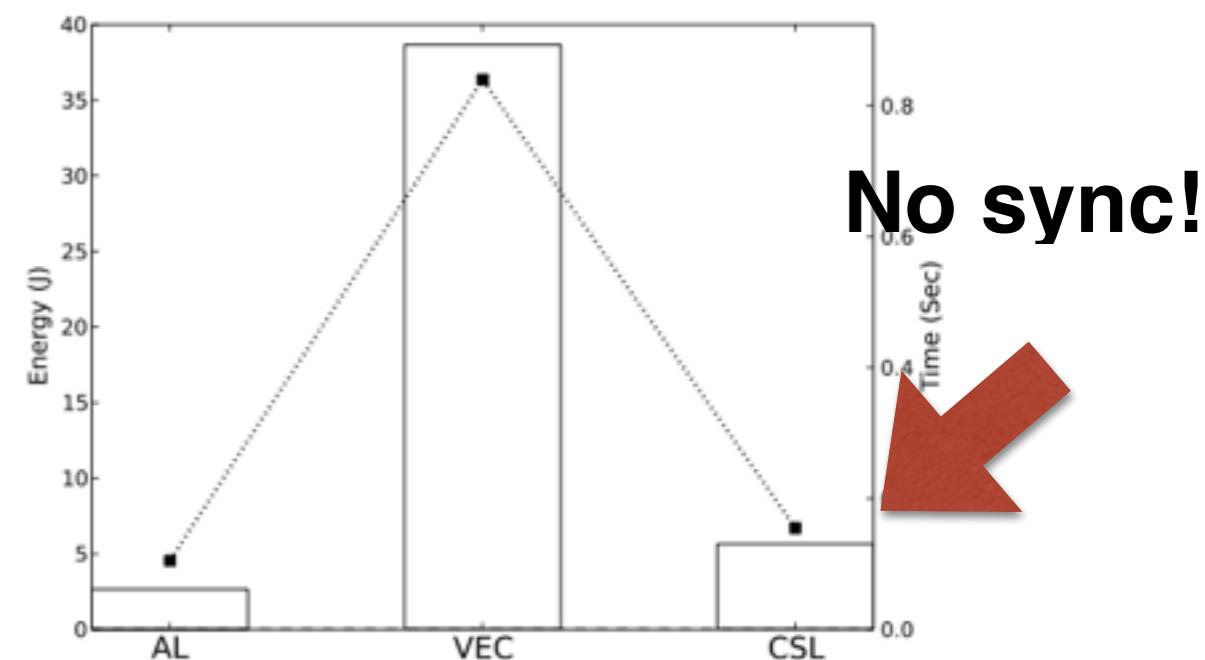


```
List<Object> lists = new ArrayList<>();
for (Object o: lists) {
    // do stuff
}
```

List



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



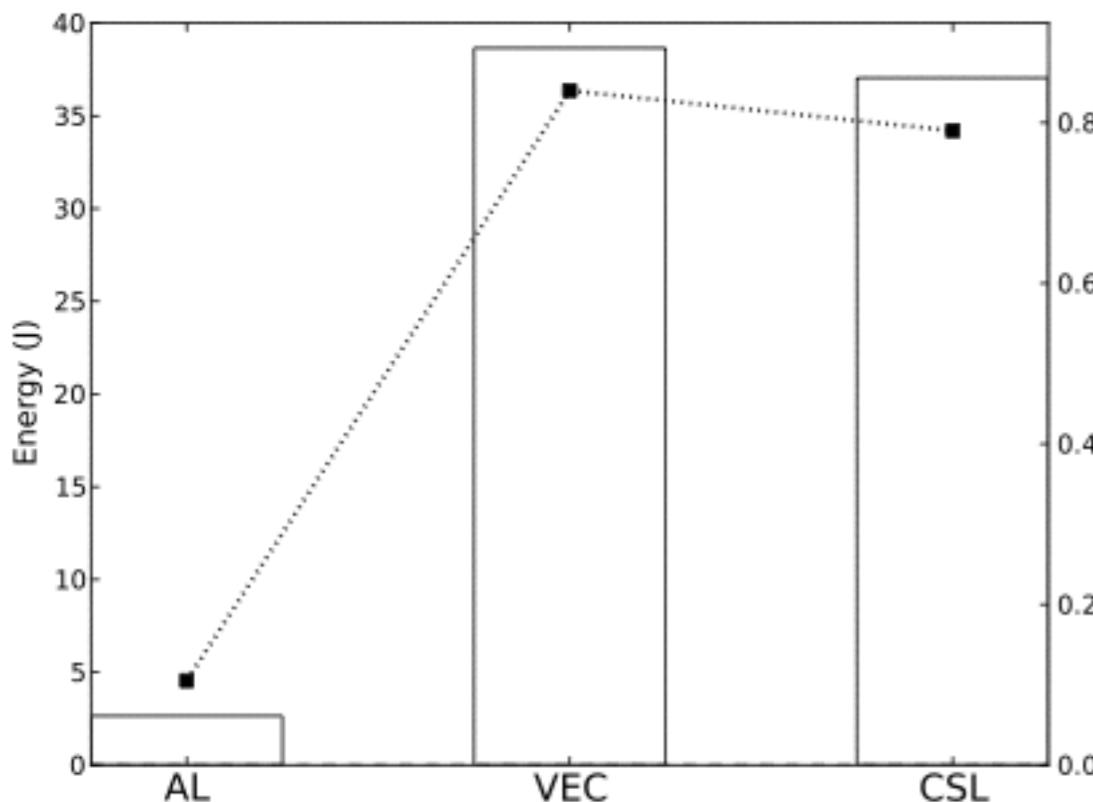
```
List<Object> lists = new ArrayList<>();
for (Object o: lists) {
    // do stuff
}
```

Time

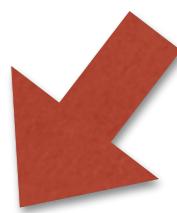
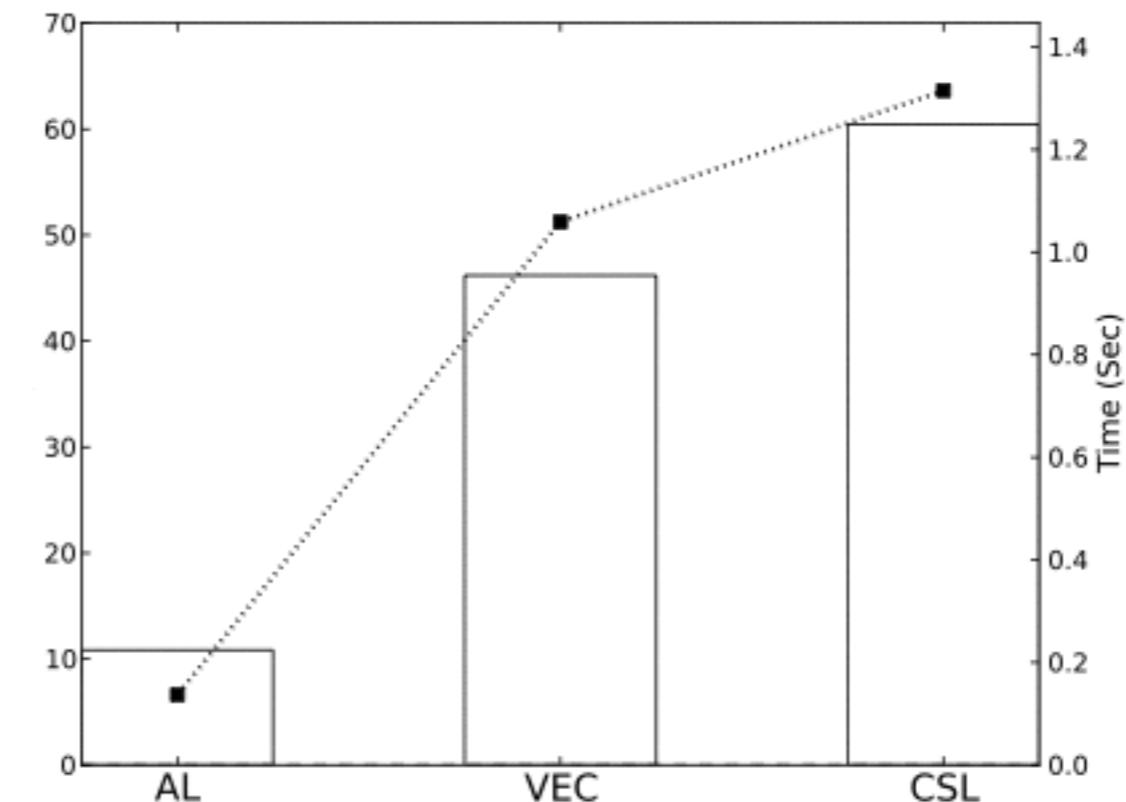
List

CopyOnWriteArrayList: +152x

Traversal



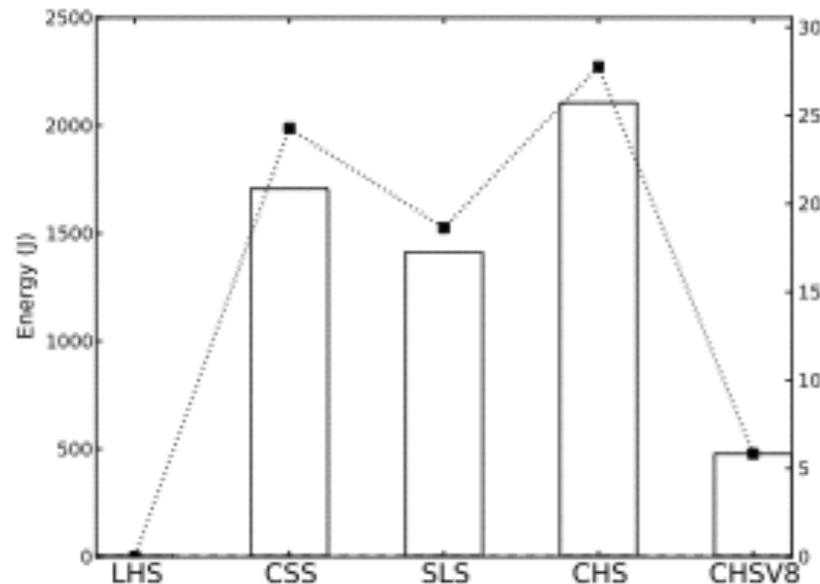
Insertion



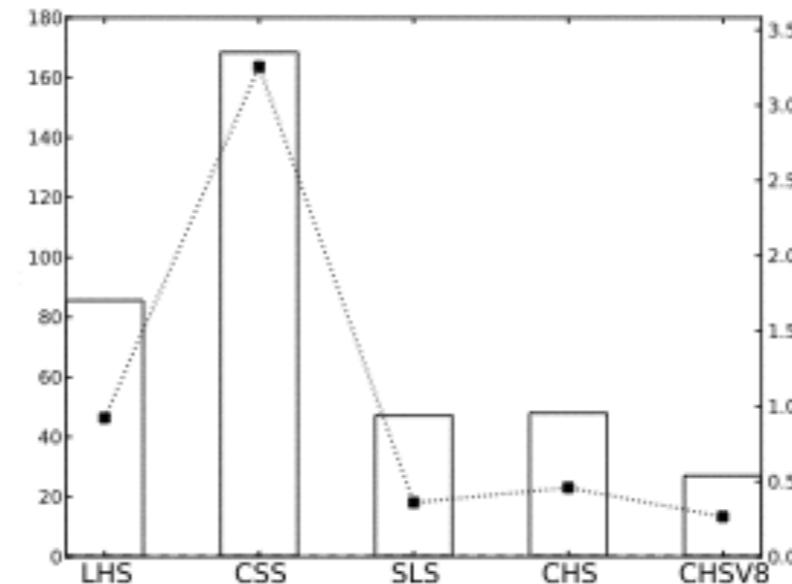
Time

Set

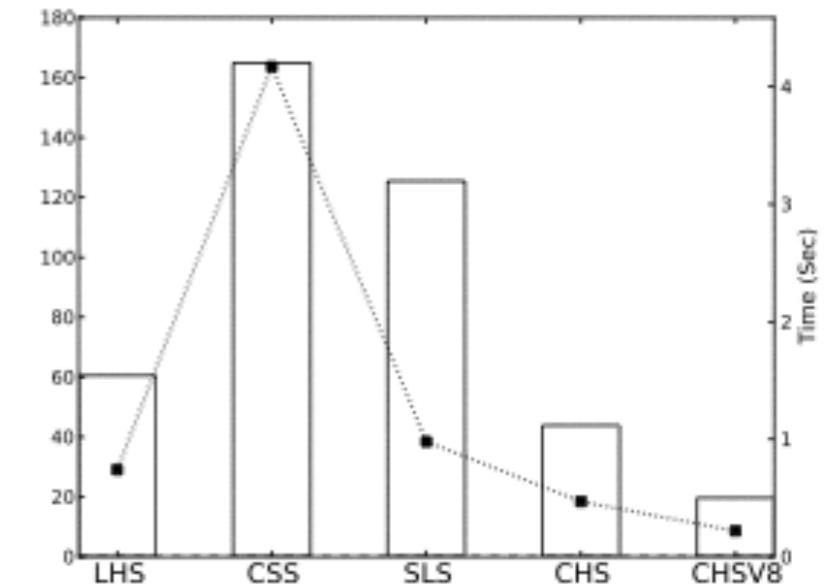
Traversal



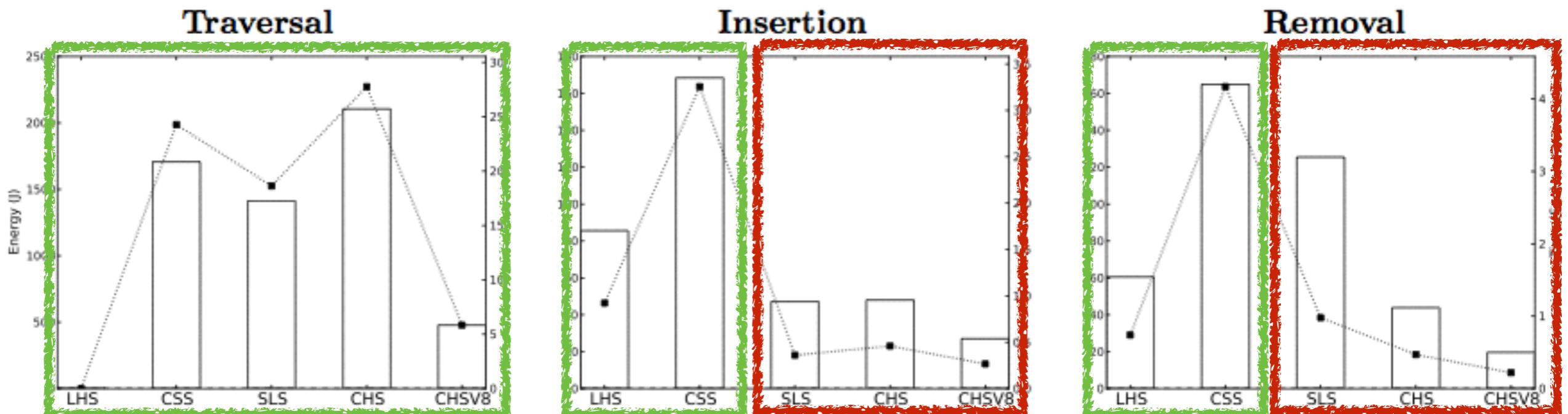
Insertion



Removal



Set



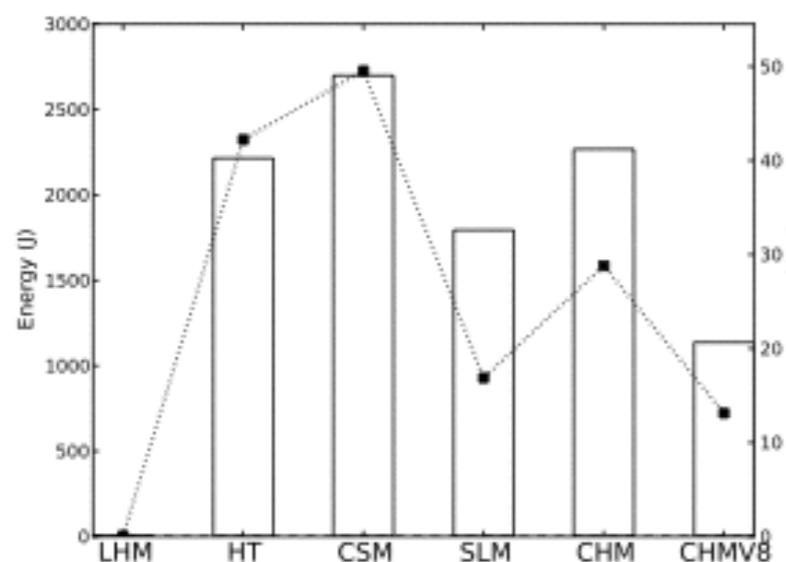
■ Time **is** proportional to energy

■ Time **is not** proportional to energy

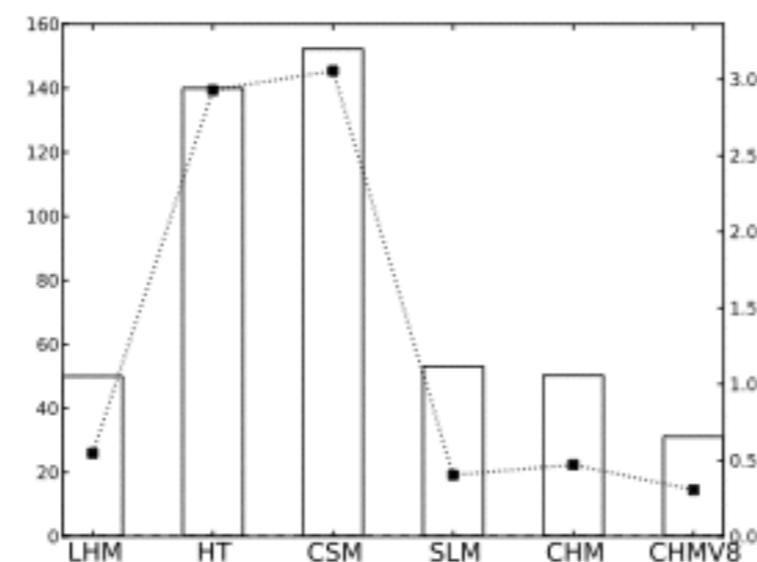
Time

Map

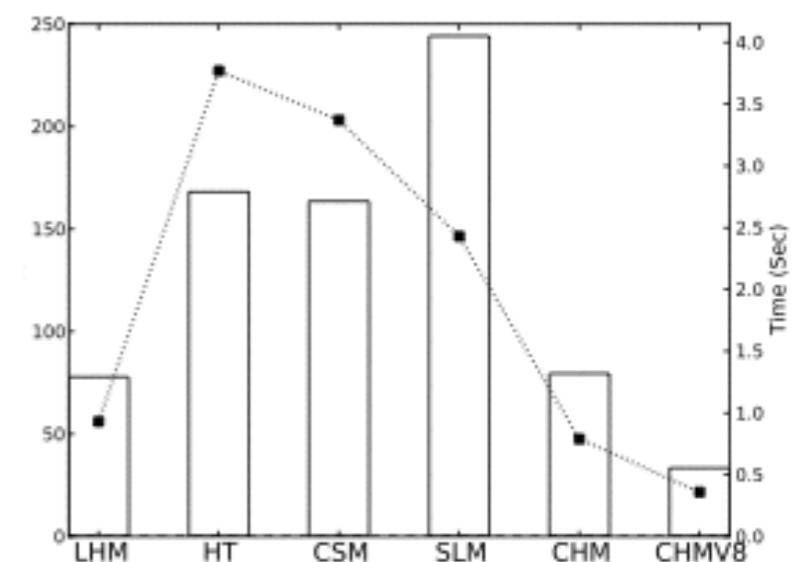
Traversal



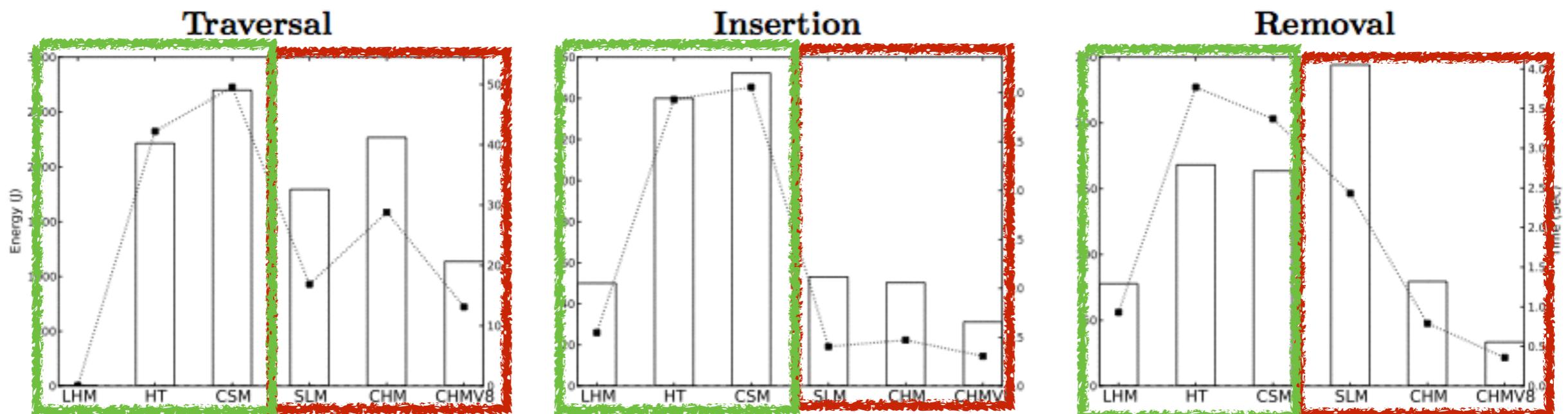
Insertion



Removal



Map



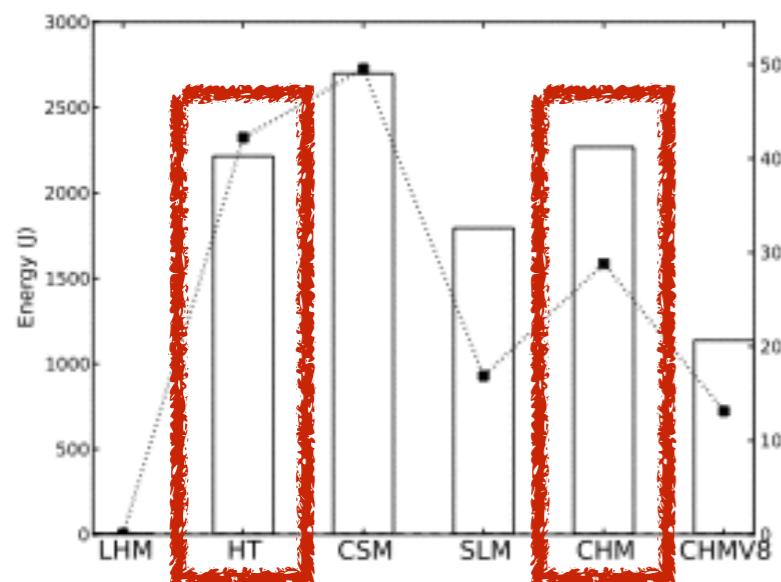
Time **is** proportional to energy

Time **is not** proportional to energy

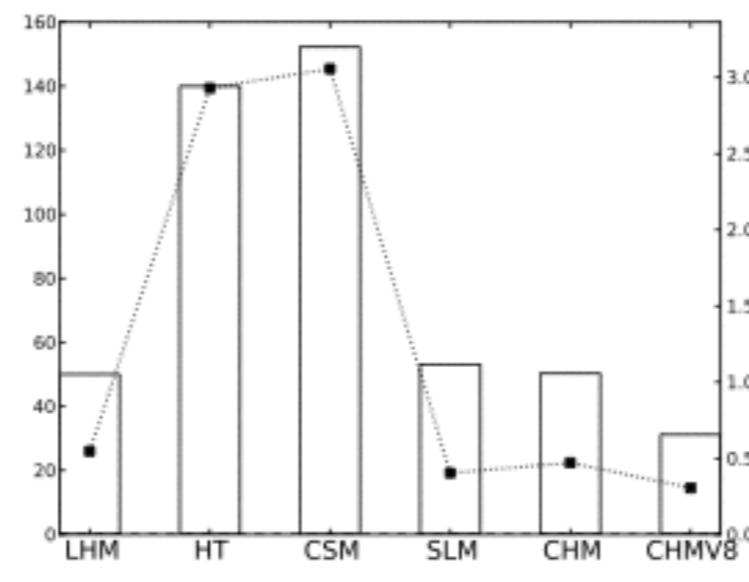
Time 

Map

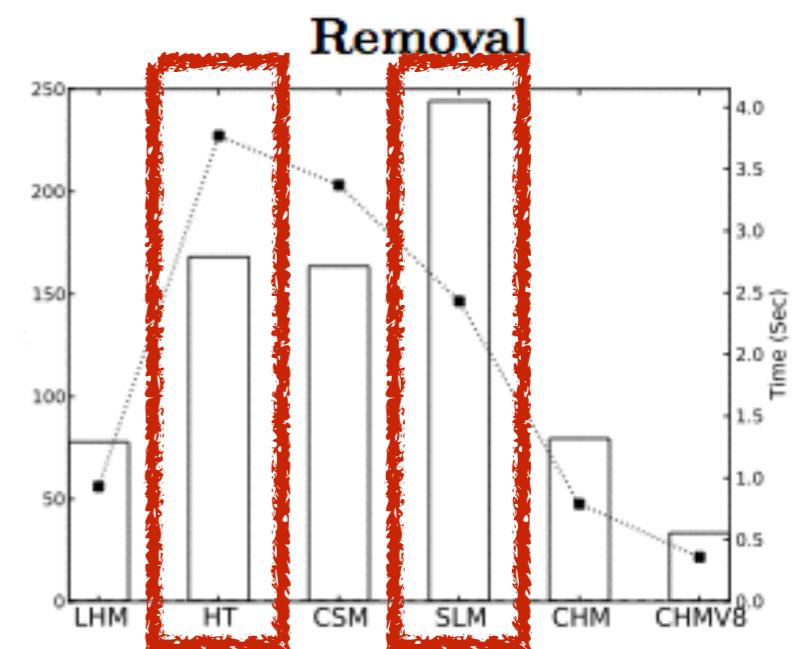
Traversal



Insertion



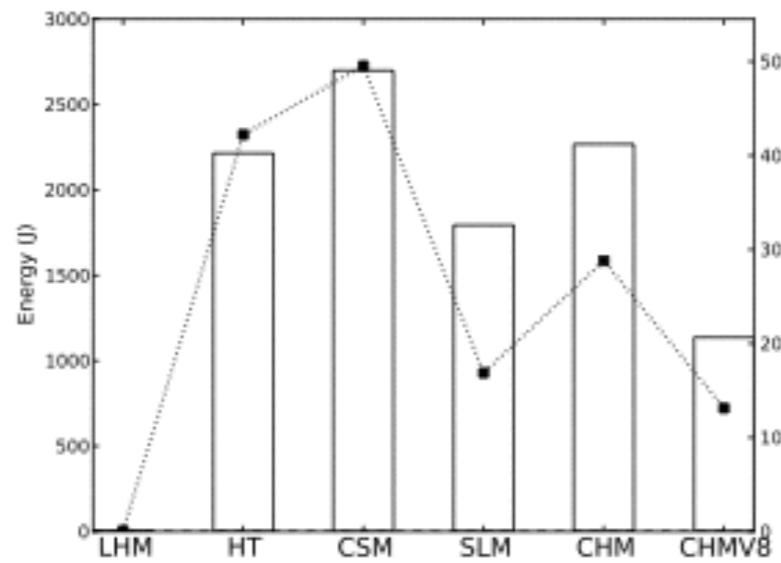
Removal



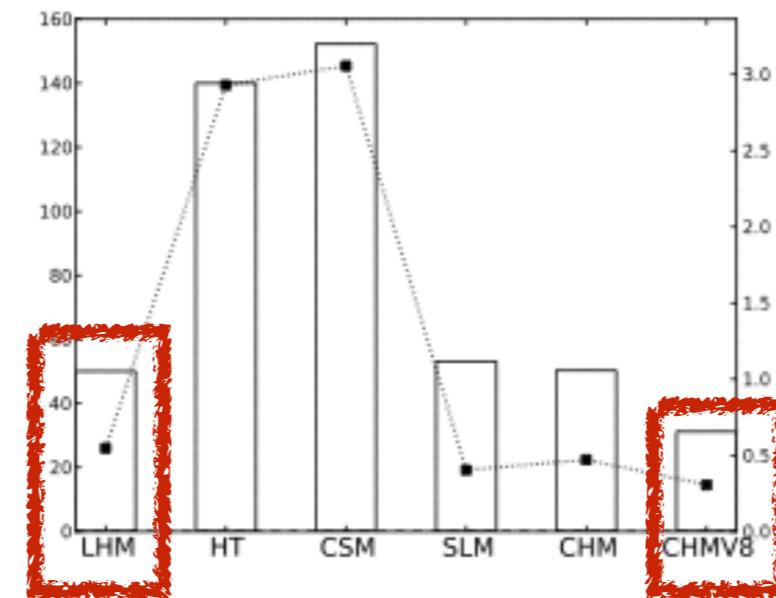
Time

Map

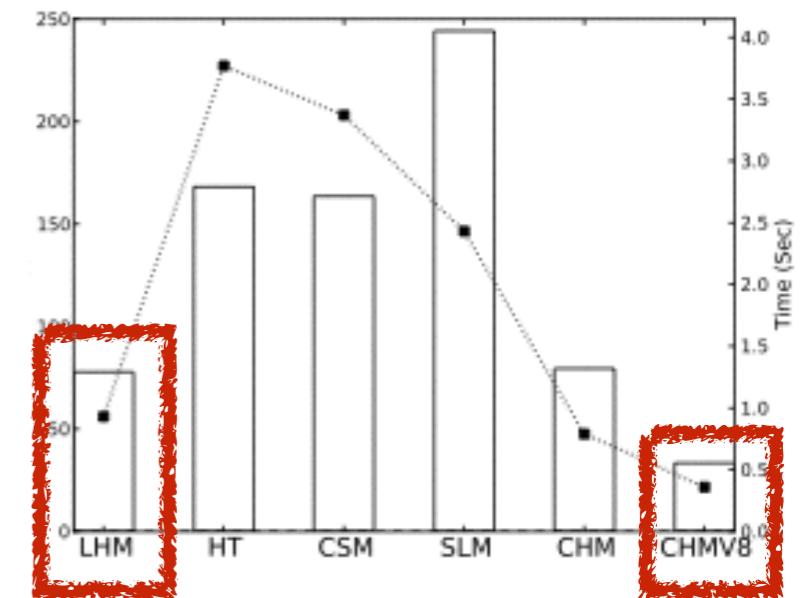
Traversal



Insertion

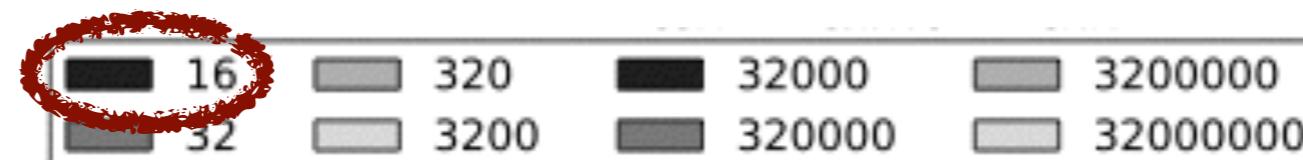
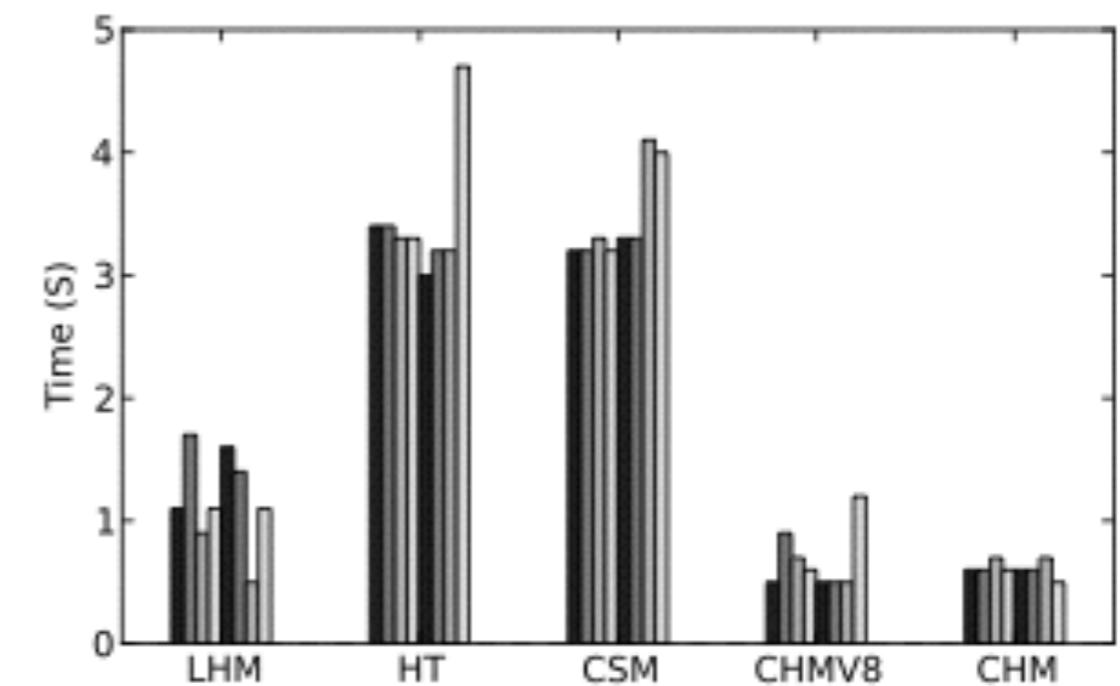
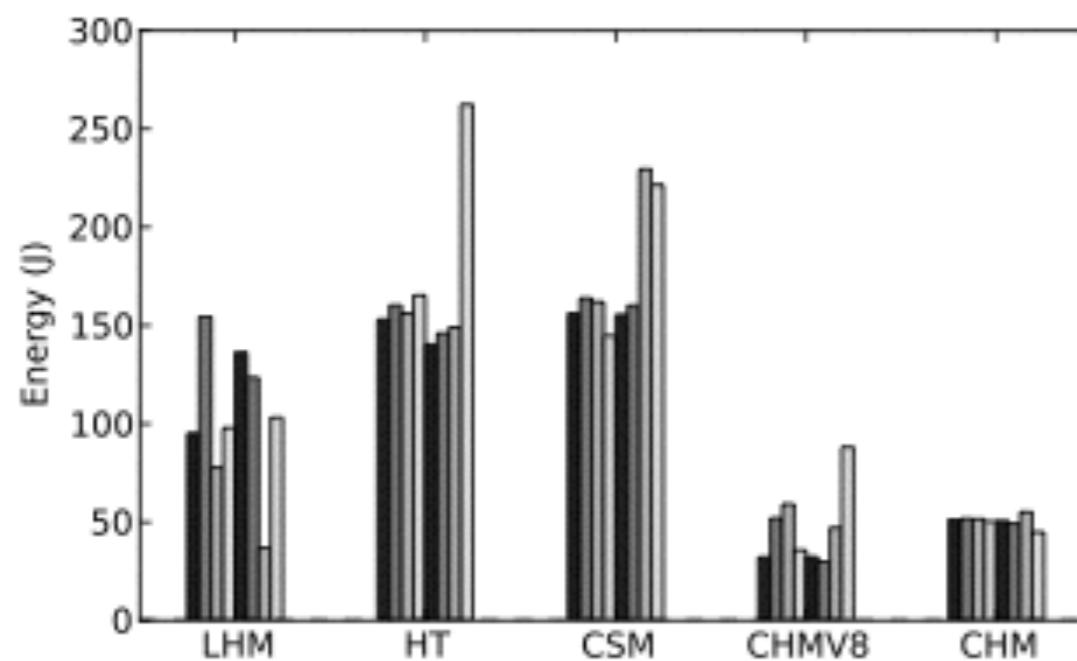


Removal



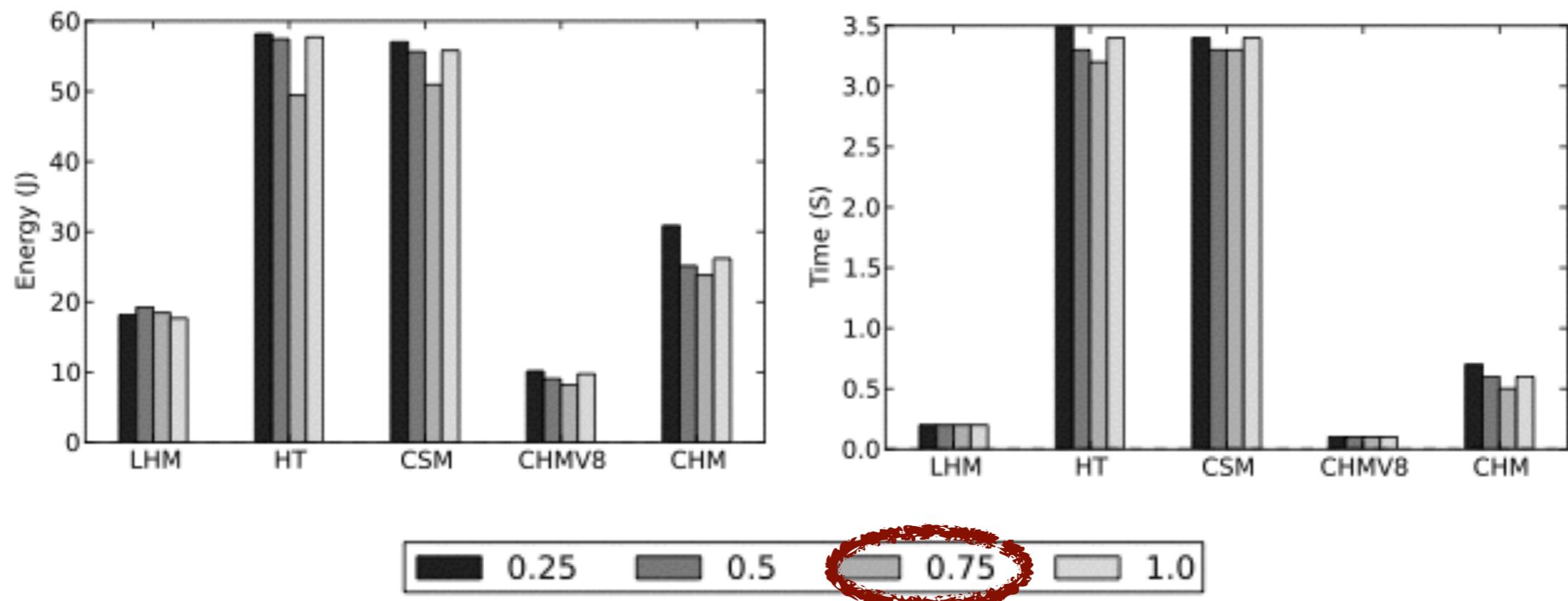
Map “Tuning Knobs”

Initial capacity



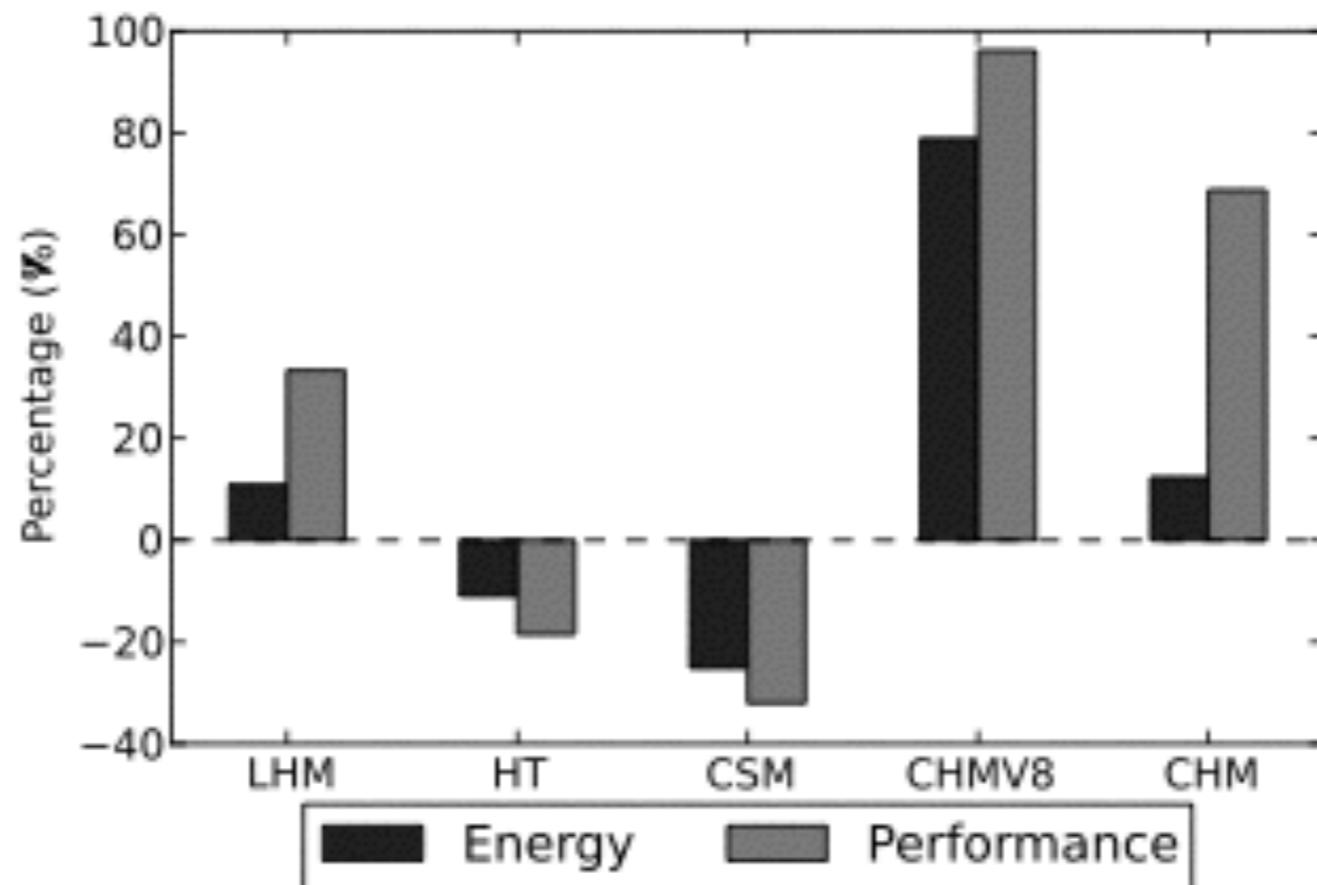
Map “Tuning Knobs”

Load factor



Map “Tuning Knobs”

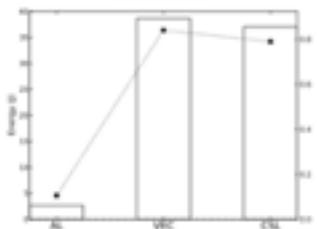
Collision 100%



Time

List

Traversal



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

No sync!



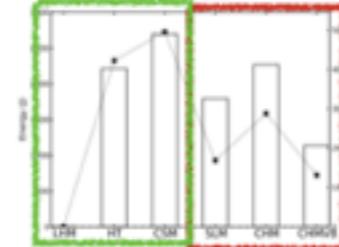
```
List<Object> lists = new ArrayList<>();
for (Object o: lists) {
    // do stuff
}
```

42

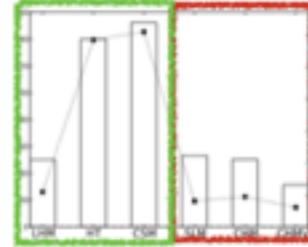
Time

Map

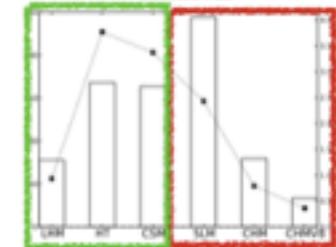
Traversal



Insertion



Removal



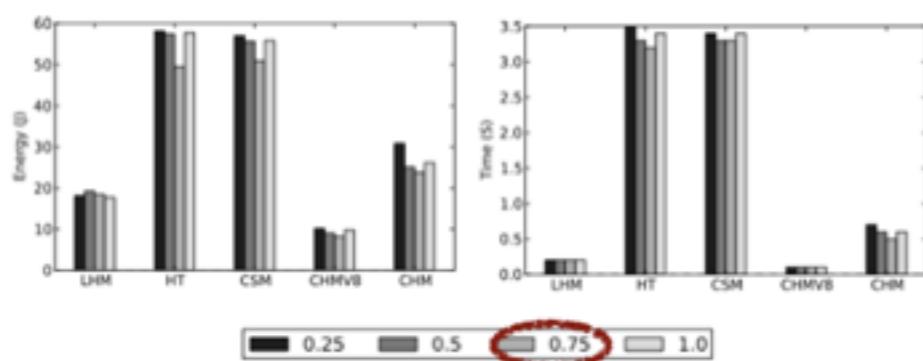
Time **is** proportional to energy

Time **is not** proportional to energy

51

Map “Tuning Knobs”

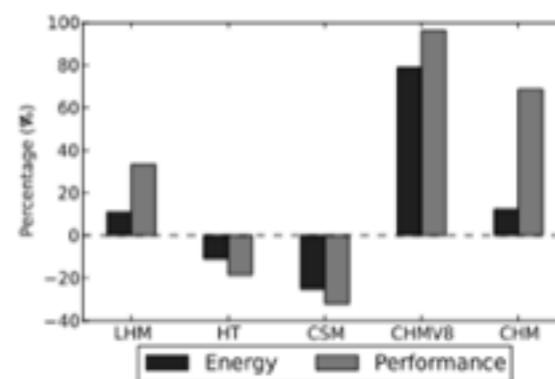
Load factor



53

Map “Tuning Knobs”

Collision 100%



54

46

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

