A Comprehensive Study on the Energy Efficiency of Java Thread-Safe Collections

Gustavo Pinto

Kenan Liu

Fernando Castor

David Liu

Disclaimer

Miguel turns 8mo today!

Motivations

First, energy consumption is not only a concern for unwired devices, but also for data centers

First, energy consumption is not only a concern for unwired devices, but also for data centers

Second, multicore CPUs are ubiquitous

First, energy consumption is not only a concern for unwired devices, but also for data centers

Second, multicore CPUs are ubiquitous

Third, data structures are the fundamentals of computer programming

In case you are a Java programmer...

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

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

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

Thread

List<Object> lists = ...;

Non Thread-Safe

- ArrayList
- LinkedList

Thread-Safe

- Vector
- Collections.synchronizedList()
- CopyOnWriteArrayList

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

Thread 

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

Thread 

Thread-safe!

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

Thread 

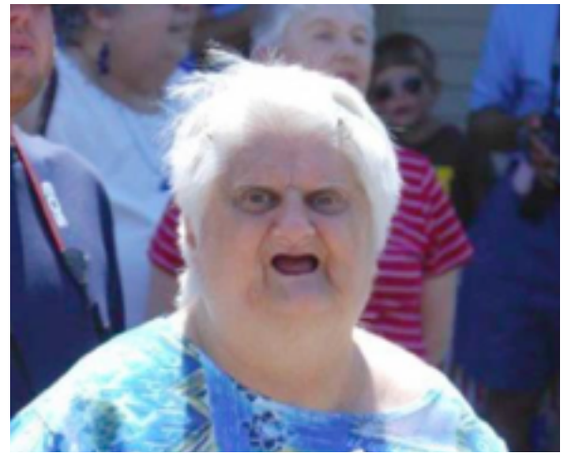
```
List<Object> lists = new Vector<>();
```

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

Thread

```
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 

 Thread-safe!

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

Thread

Thread-safe!

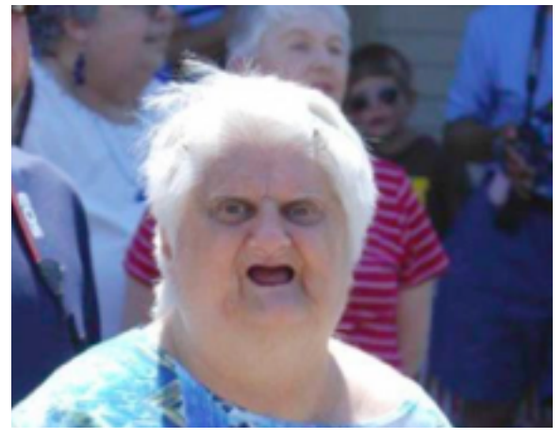
List<Object> lists = new CopyOnWriteArrayList<>();

```
public class CopyOnWriteArrayList<E> {

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

    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 = ...;

Non Thread-Safe

- ArrayList
- LinkedList

Thread-Safe

- Vector
- Collections.synchronizedList()
- CopyOnWriteArrayList

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

Set<Objects> sets = ...;

Map<Object, Object> maps = ...;
```

List<Object> lists = ...;

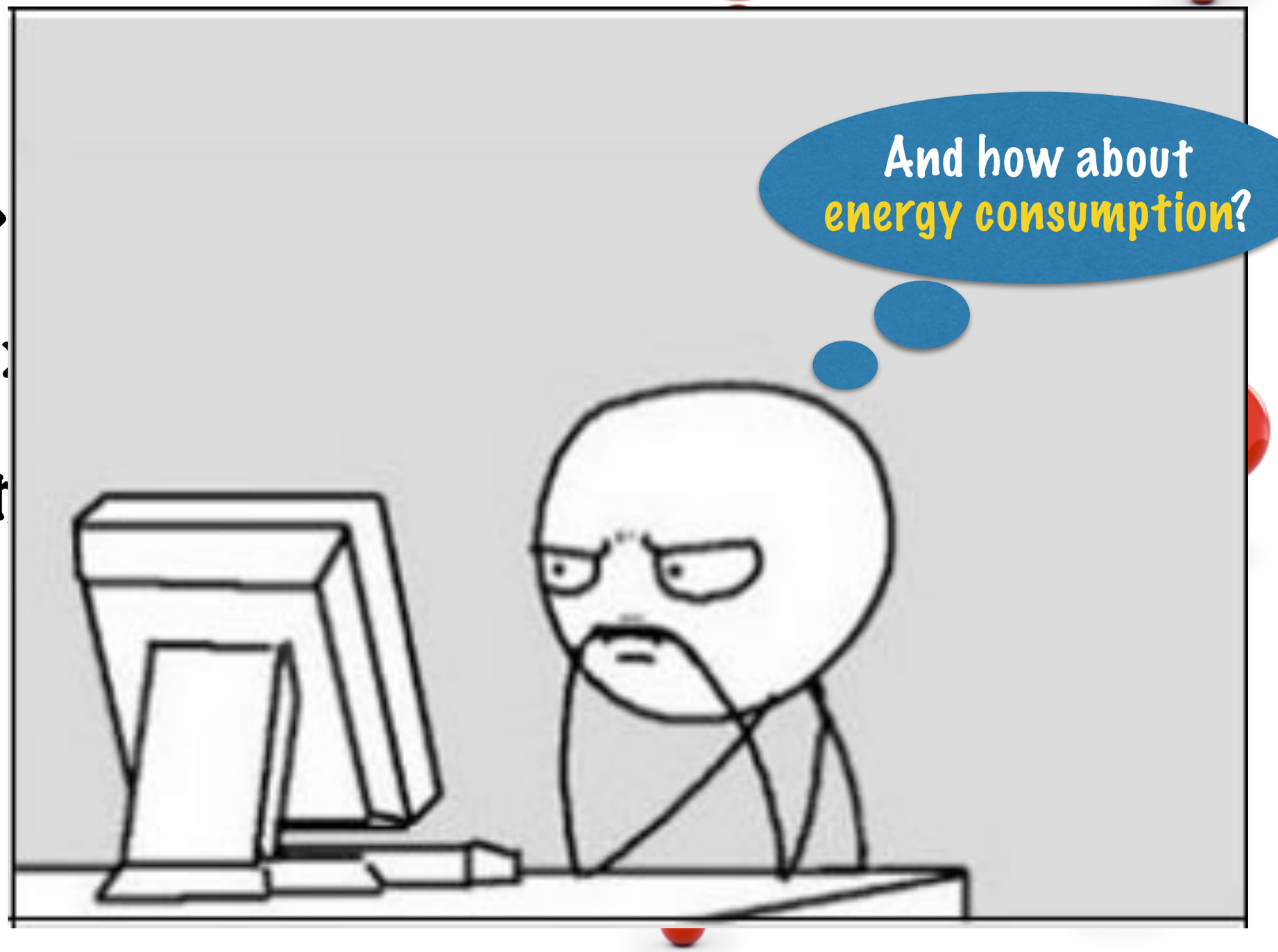
Set<Objects> sets = ...;

Map<Object, Object> maps = ...;

List<Object>

Set<Objects>

Map<Object



16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

Non thread-safe

Thread-safe

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

Java 7

Java 8

16 Collections

List
ArrayList
Vector
Collections.syncList()
CopyOnWriteArrayList

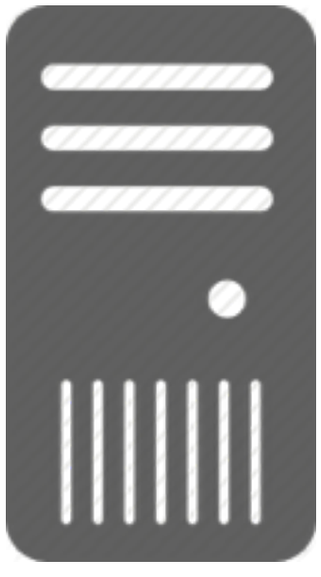
Set
LinkedHashSet
Collections.syncSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.syncMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

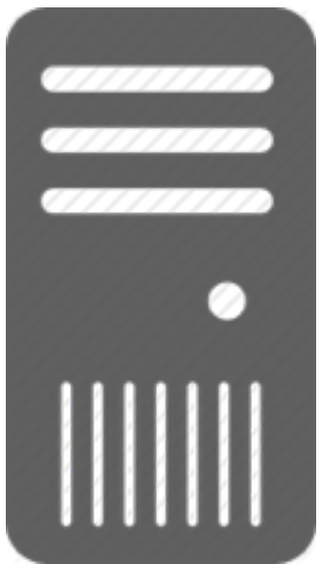
x 3 Operations

Traversal	Insertion	Removal

2 Environments

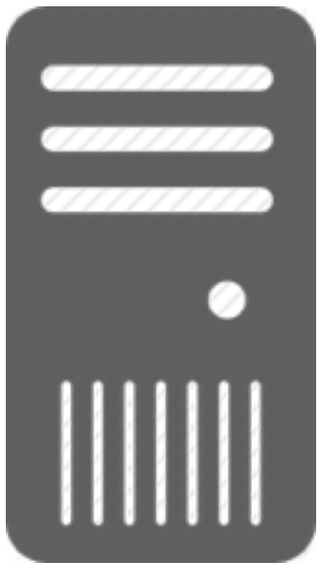


AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0_11, build 21.



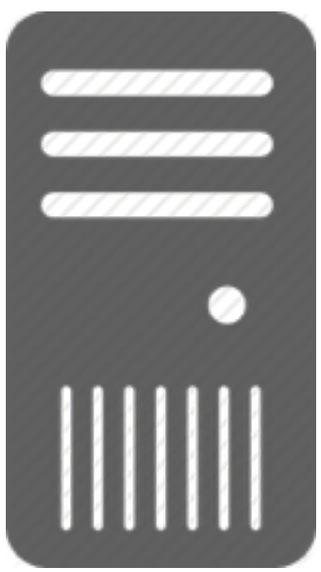
Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0_71, build 14.

2 Environments



AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

Hardware-based energy measurement



Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

Software-based energy measurement

More details about how the experiments were conducted?

Artifacts Track

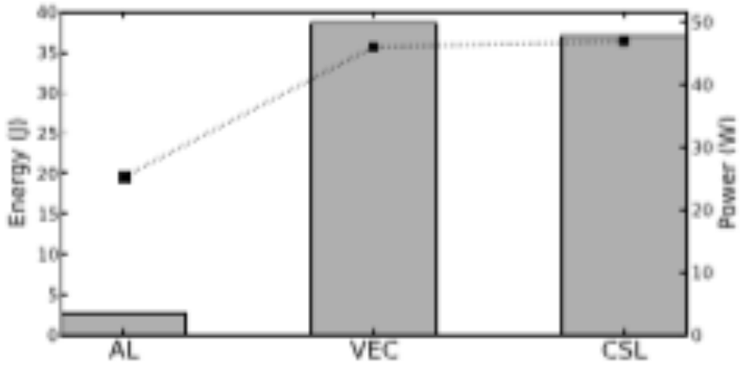
Friday (10:45 AM - 12:15 PM, University)

- Breno D. Cruz, Paul W. McBurney and Collin McMillan. *TraceLab Components for Reproducing Source Code Summarization Experiments*
- Neil Walkinshaw and Mathew Hall. *Data and Analysis code for GP EFSM Inference (Scorecard)*
- Mona Rahimi and Jane Cleland-Huang. *Cassandra Source Code, Feature Descriptions across 27 versions, with Starting and Ending Version Trace Matrices*
- Leonard Punt, Sjoerd Visscher and Vadim Zaytsev. *A Tool for Detecting and Refactoring the A?B*A Pattern in CSS*
- Gustavo Pinto, Kenan Liu, Fernando Castor and Yu David Liu. *A Comprehensive Study on the Energy Efficiency of Java's Thread-Safe Collections*
- Leonard Punt, Sjoerd Visscher and Vadim Zaytsev. *The A?B*A Pattern: Undoing Style in CSS and Refactoring Opportunities it Presents*

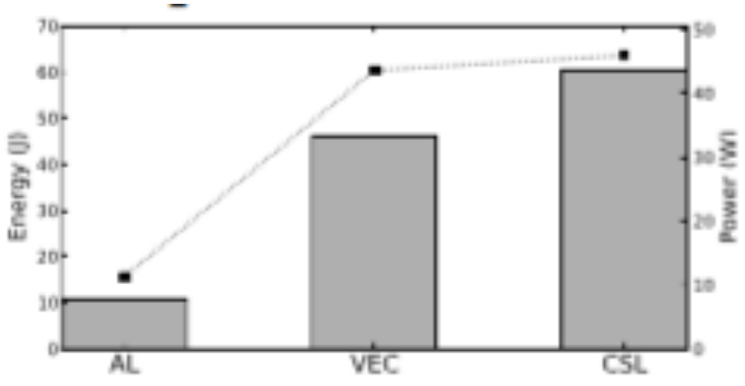
http://icsme2016.github.io/program/accepted.html

Results

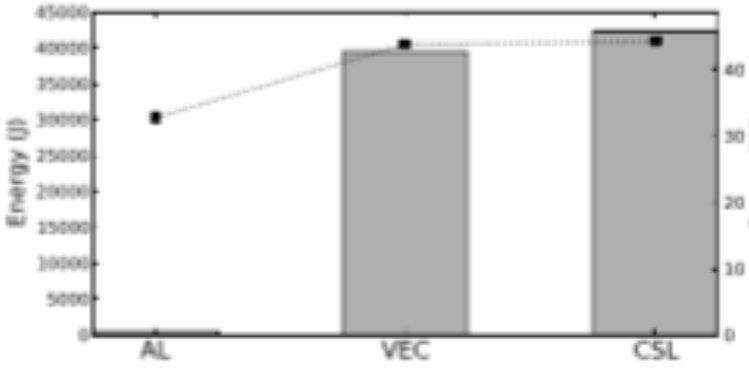
AMD CPU



Traversal

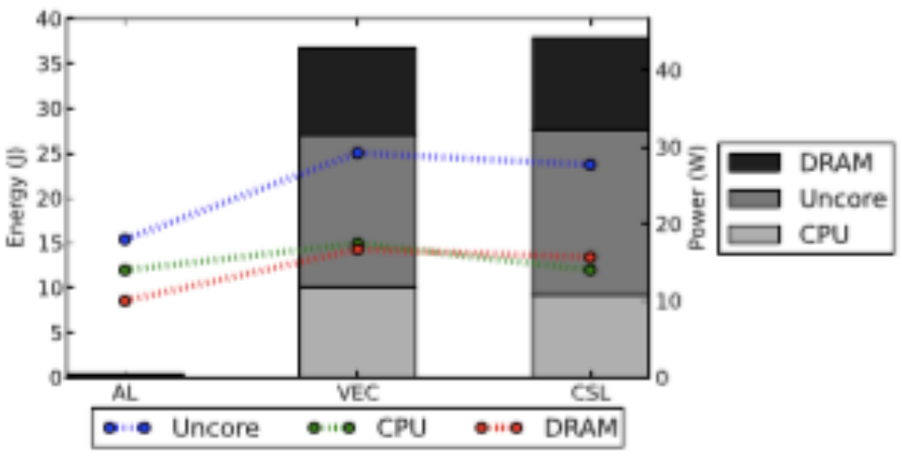


Insertion

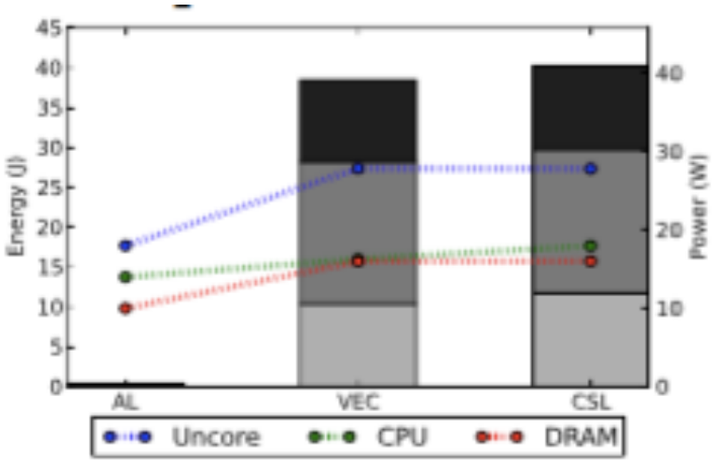


Removal

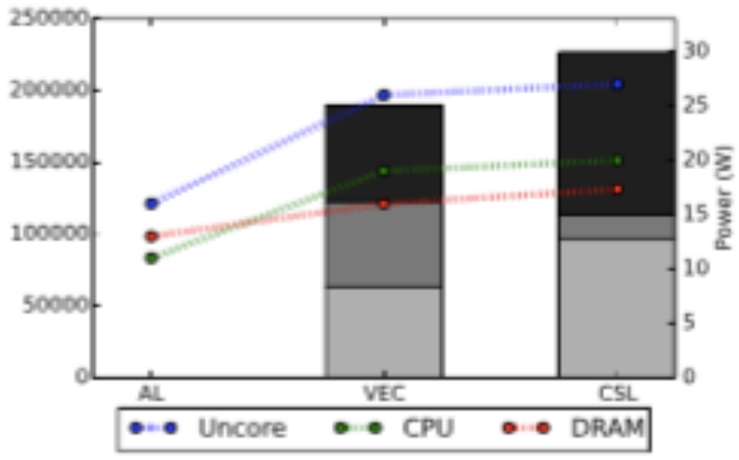
Intel CPU



Traversal

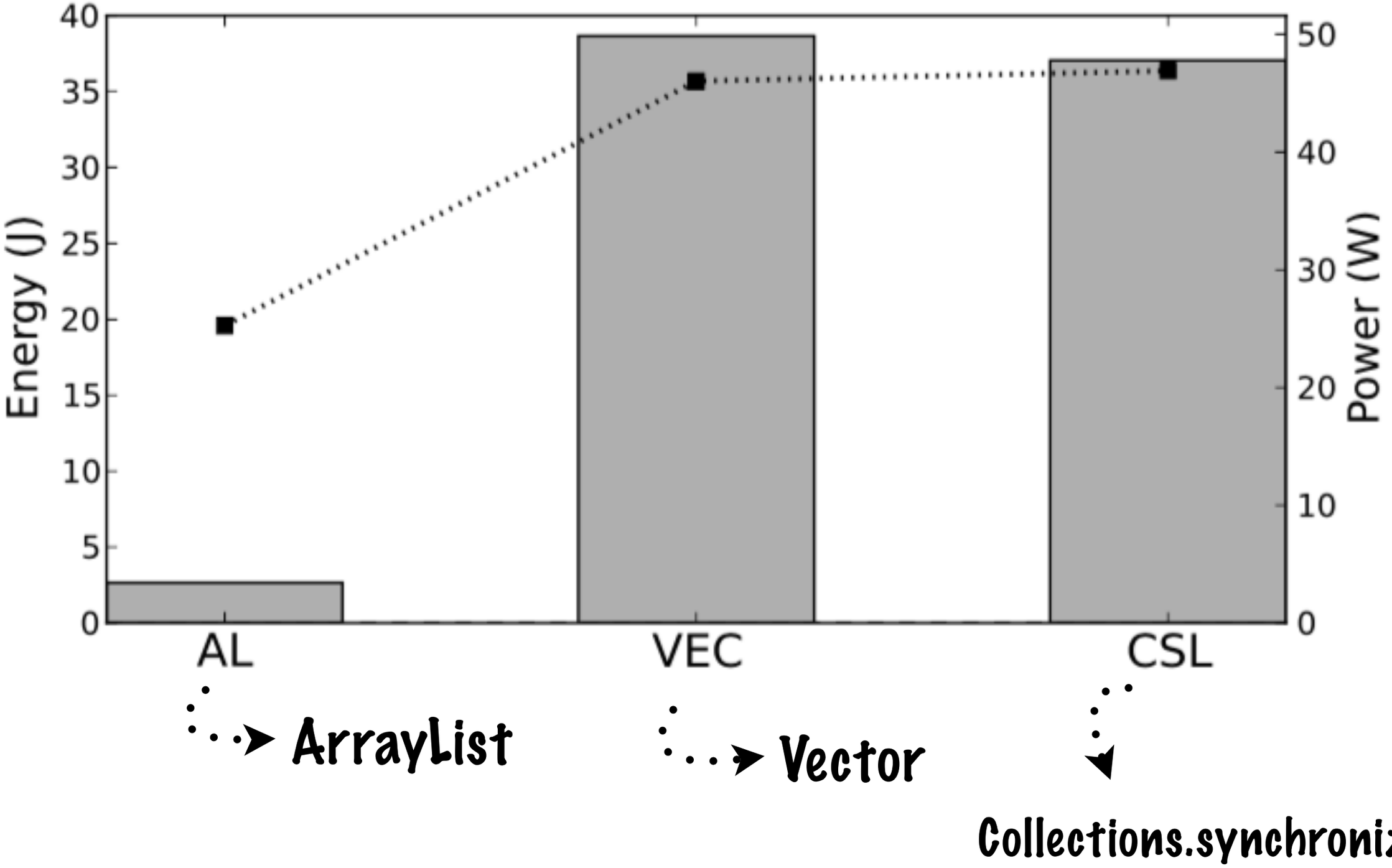


Insertion

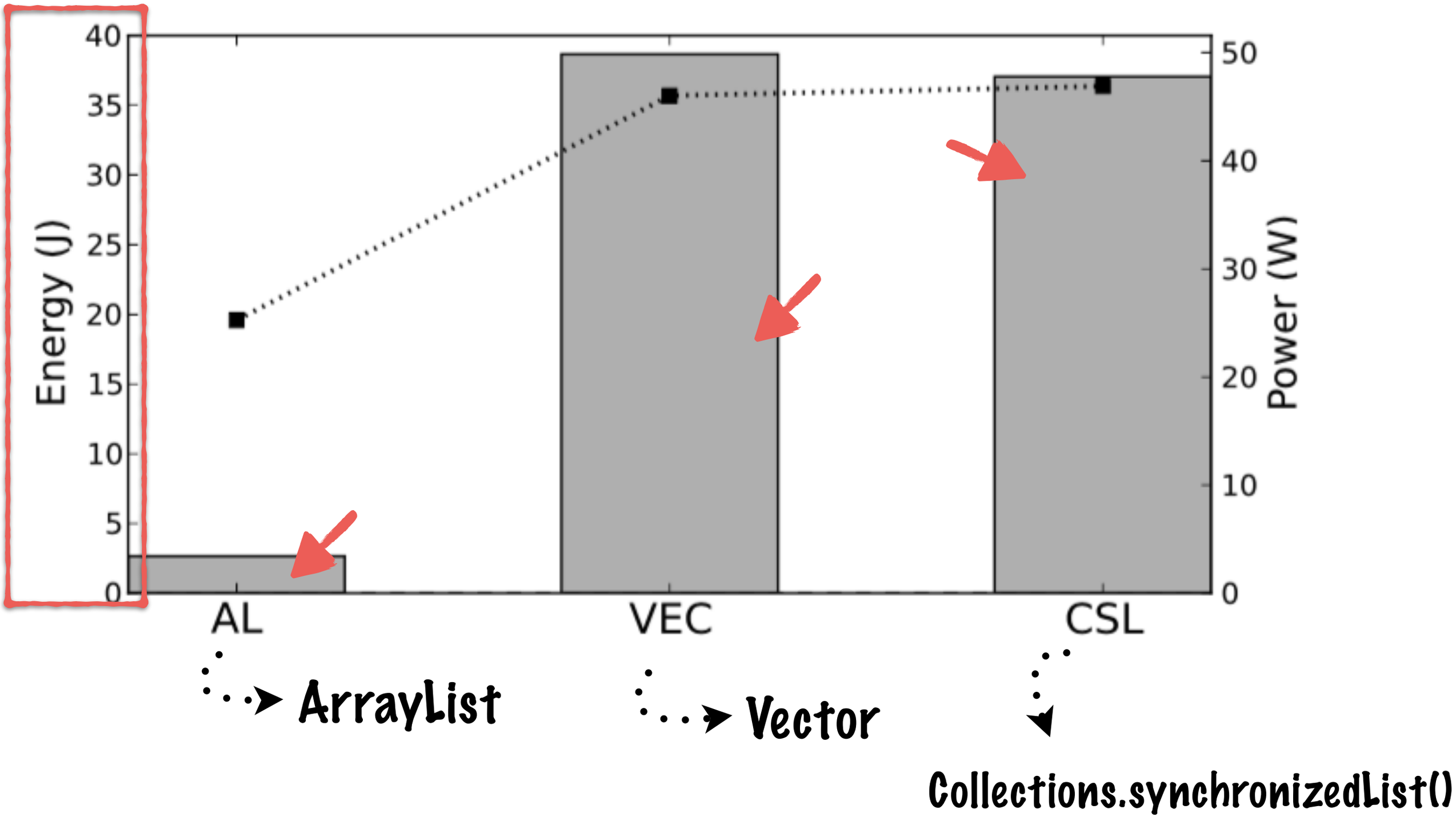


Removal

Lists


Energy (J)
Power (W)
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50
AL
VEC
CSL
ArrayList
Vector
Collections.synchronizedList()

Energy (Joules)



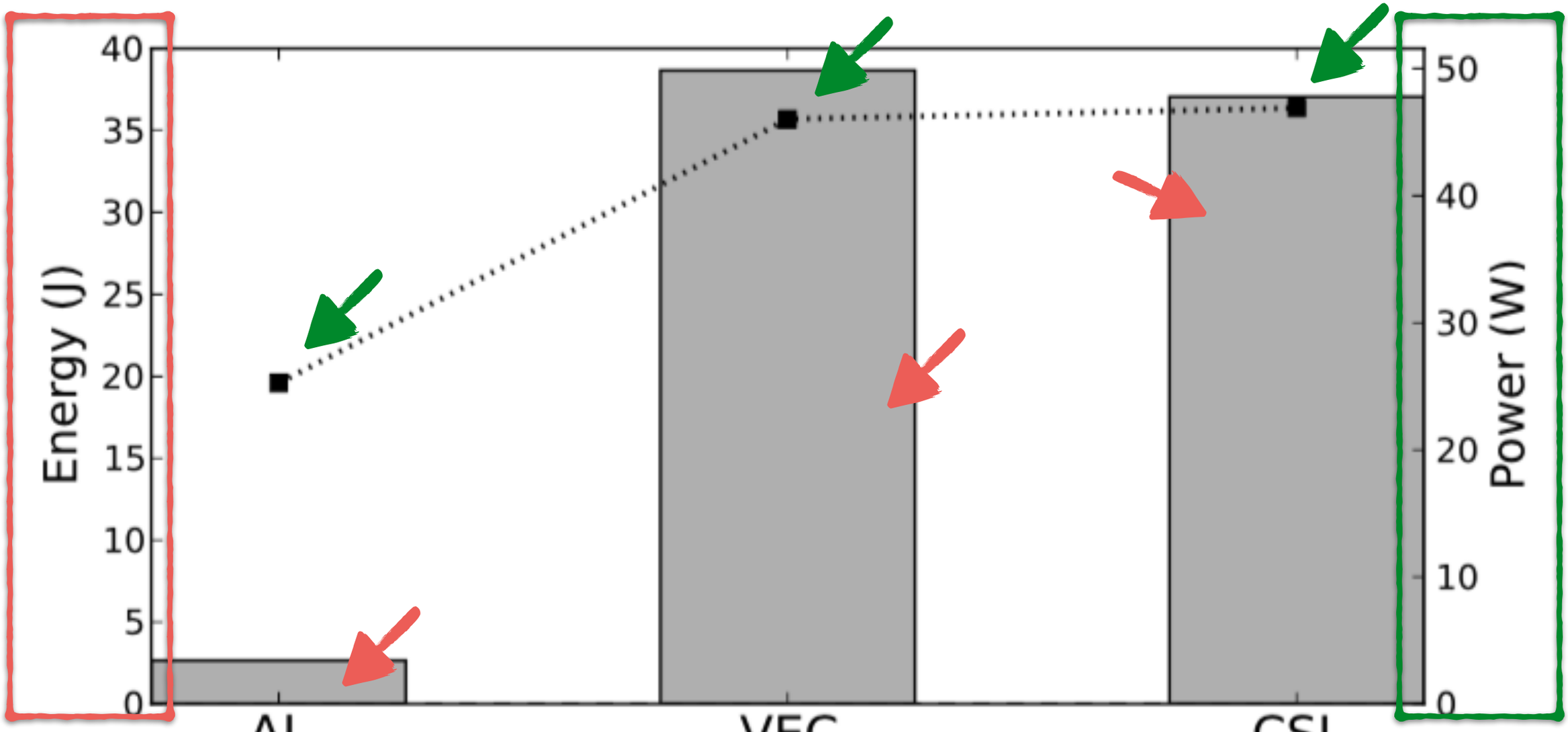
ArrayList

Vector

Collections.synchronizedList()

Energy (Joules)

Power (Watts)



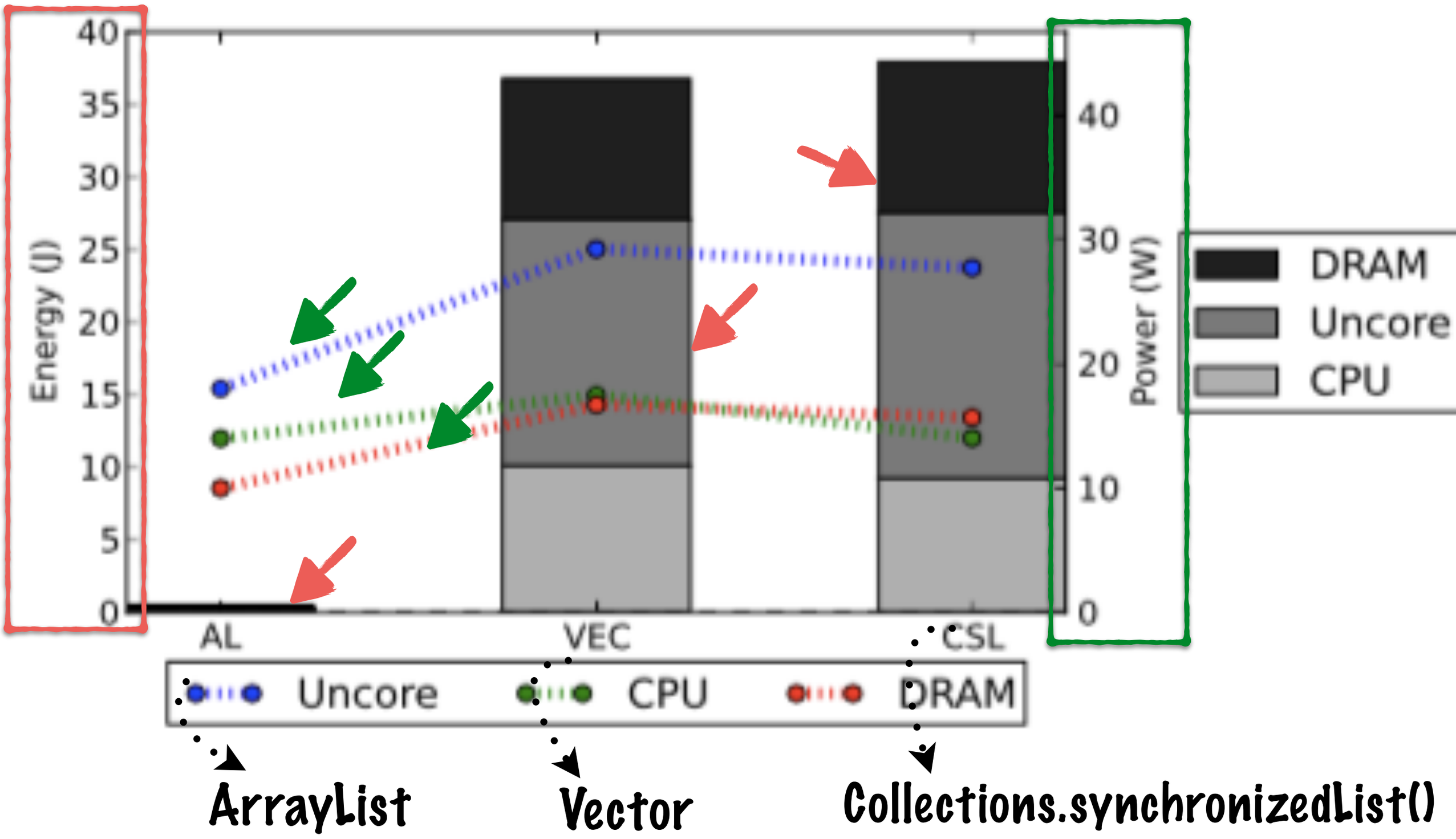
ArrayList

Vector

Collections.synchronizedList()

Energy (Joules)

Power (Watts)

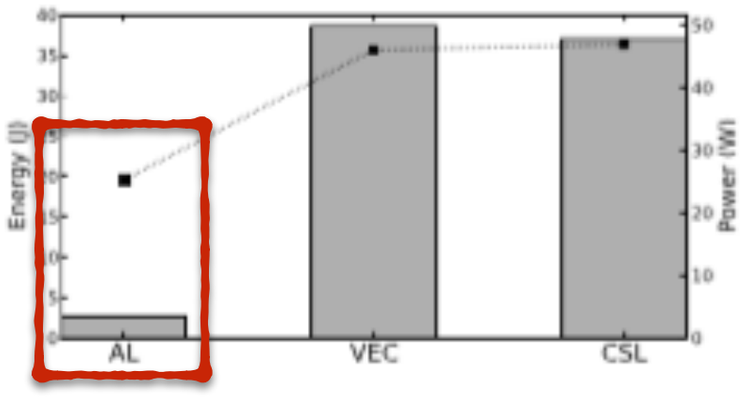


ArrayList

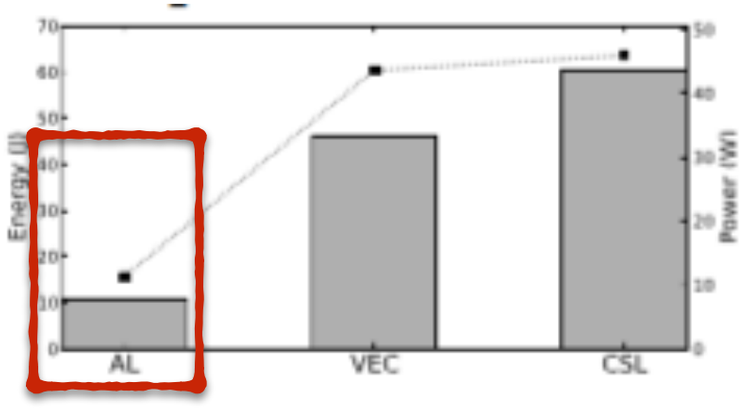
Vector

Collections.synchronizedList()

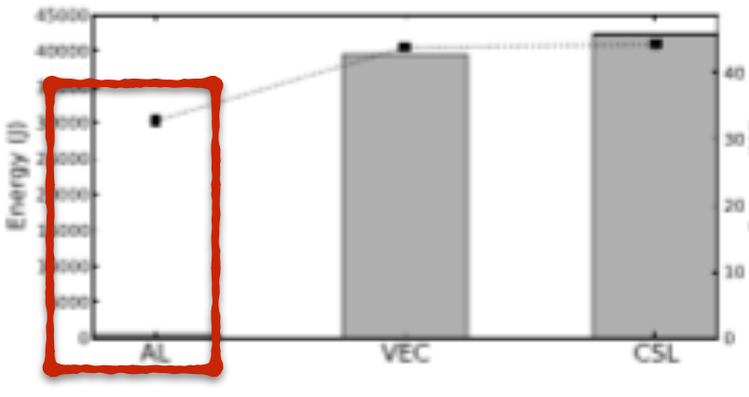
AMD CPU



Traversal

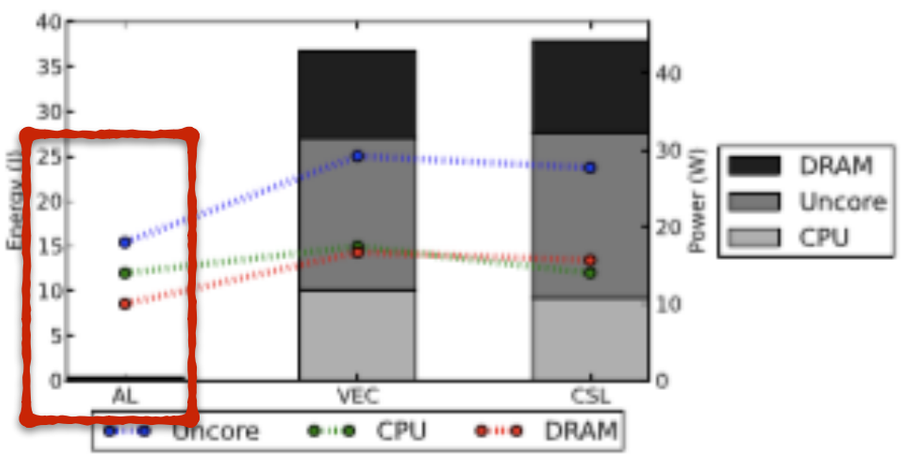


Insertion

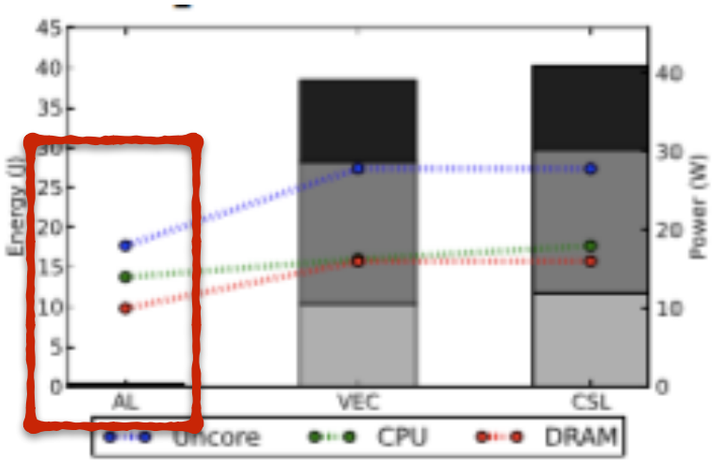


Removal

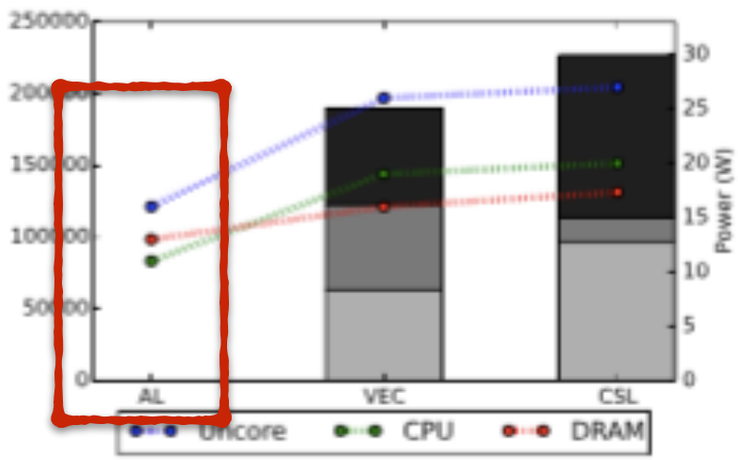
Intel CPU



Traversal

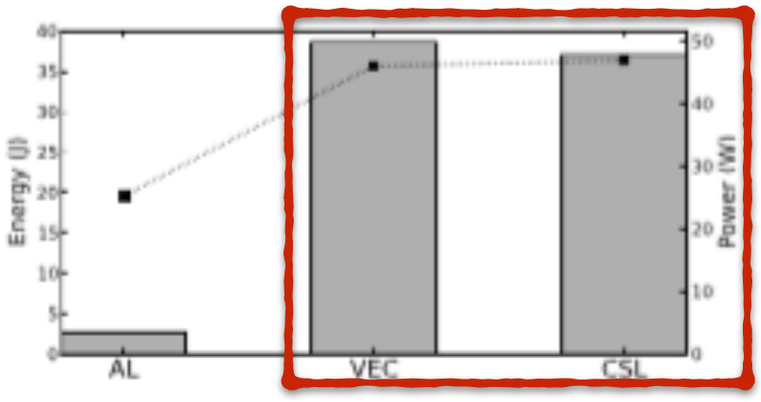


Insertion

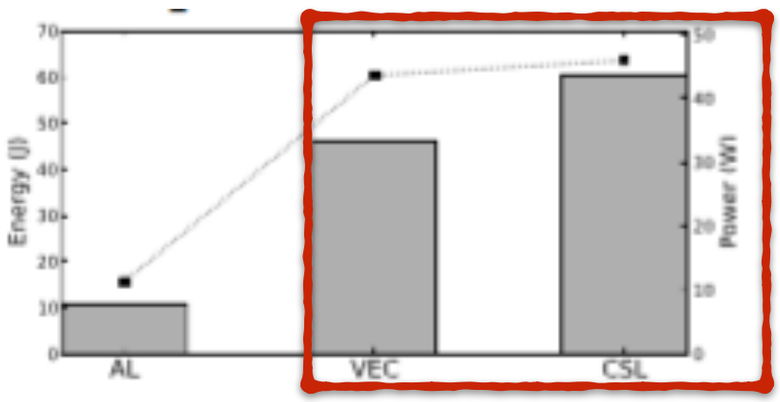


Removal

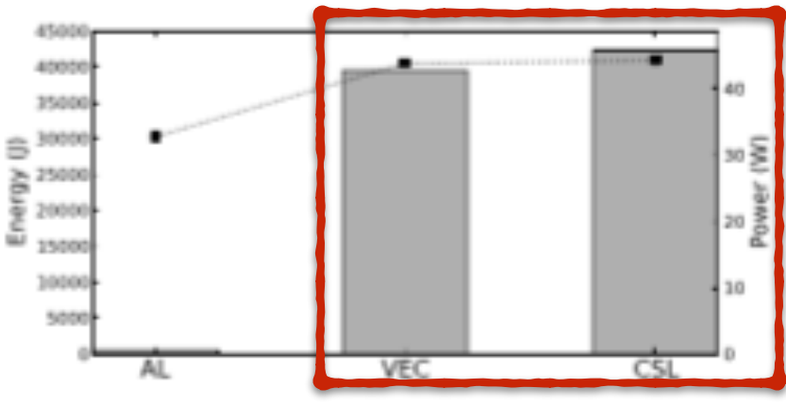
AMD CPU



Traversal

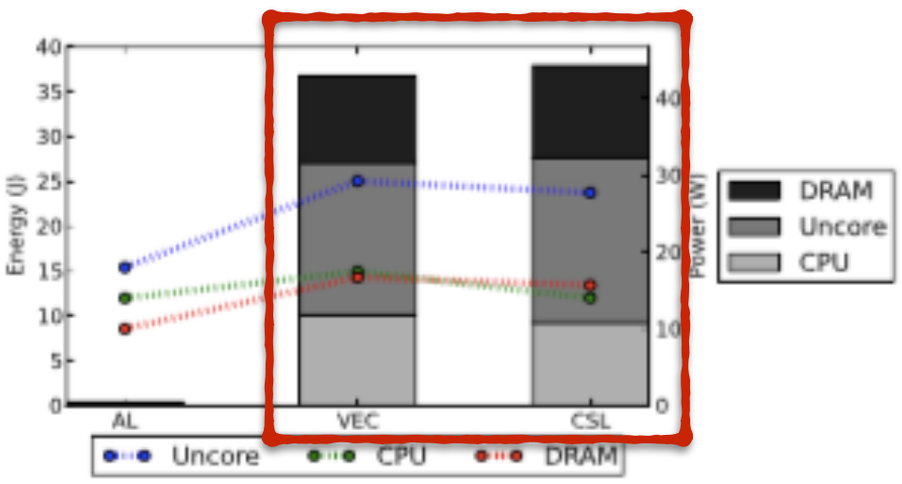


Insertion

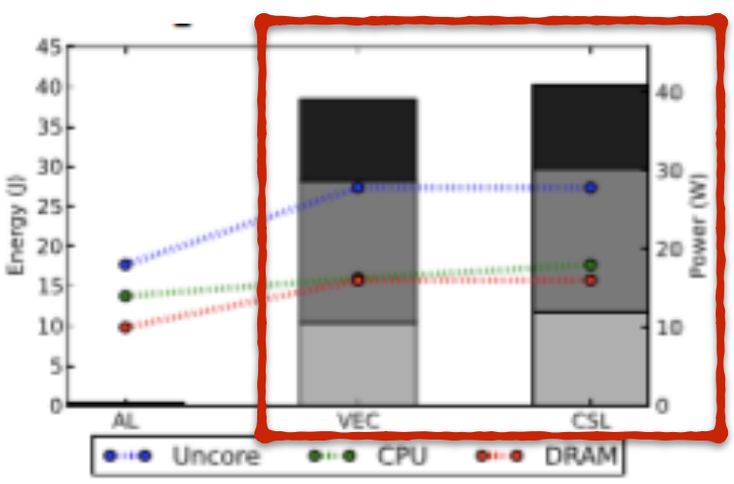


Removal

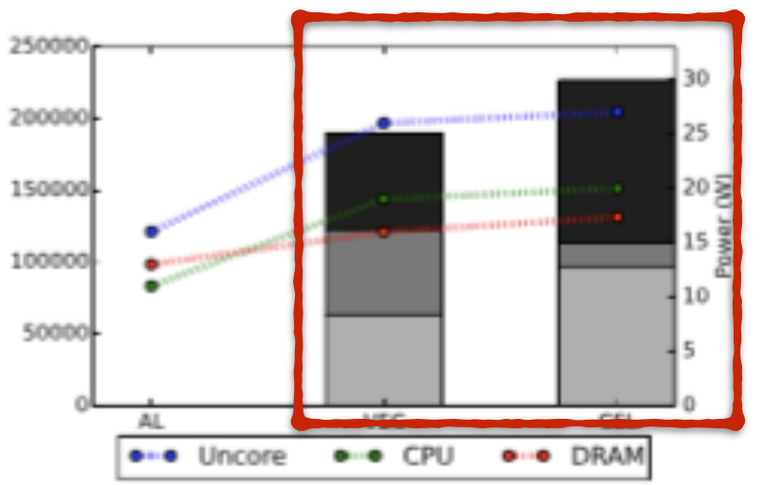
Intel CPU



Traversal



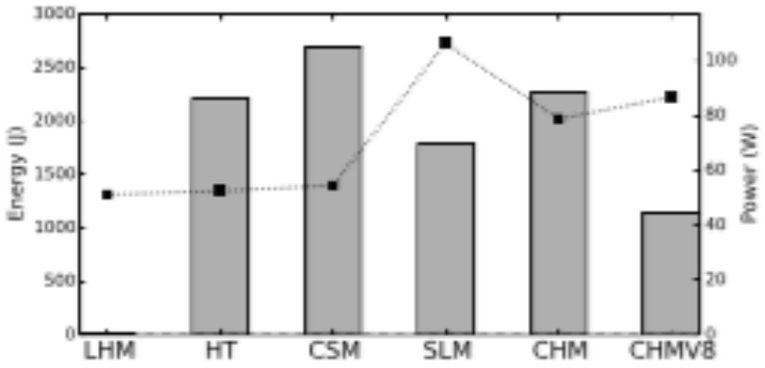
Insertion



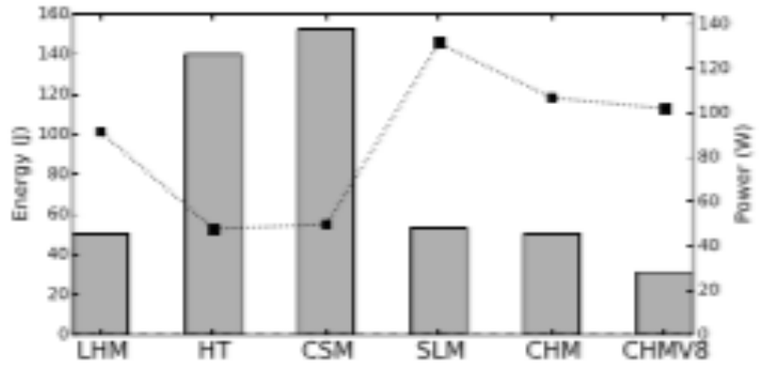
Removal

Maps

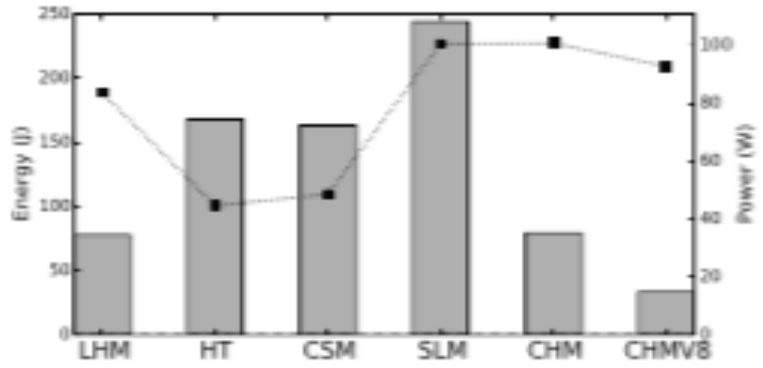
AMD CPU



Traversal

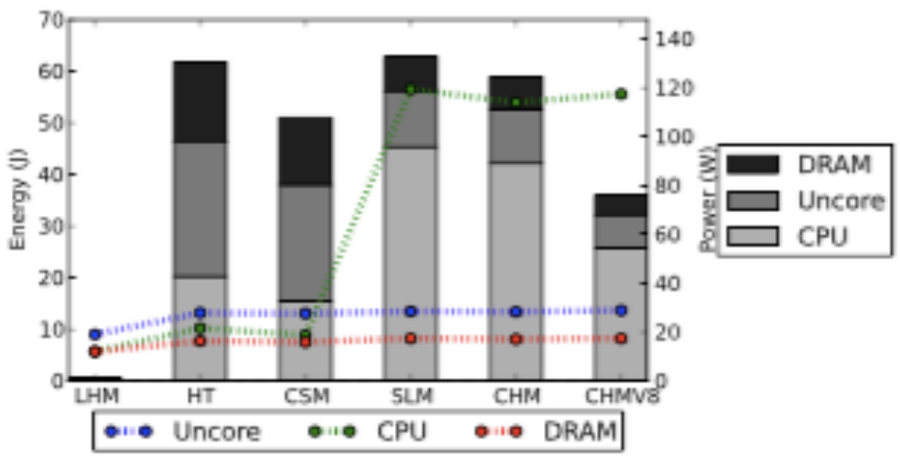


Insertion

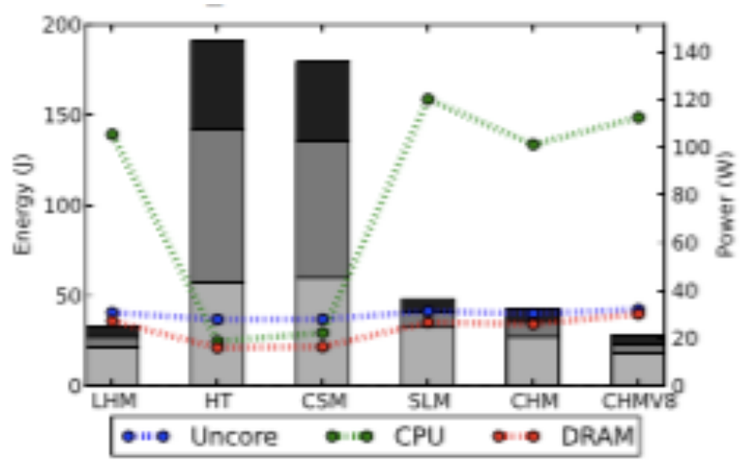


Removal

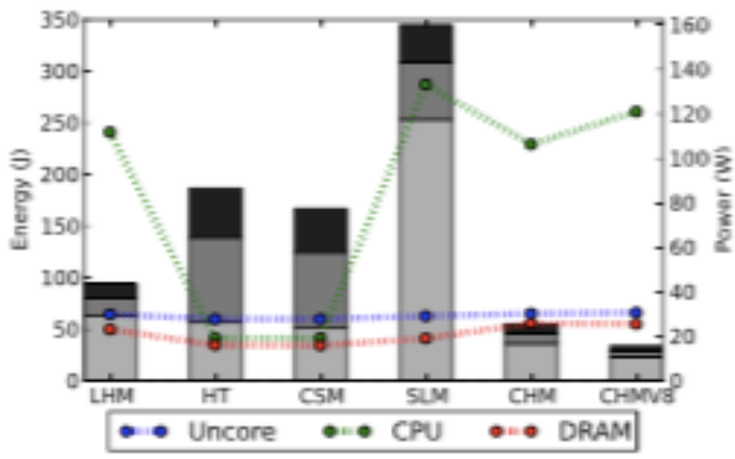
Intel CPU



Traversal



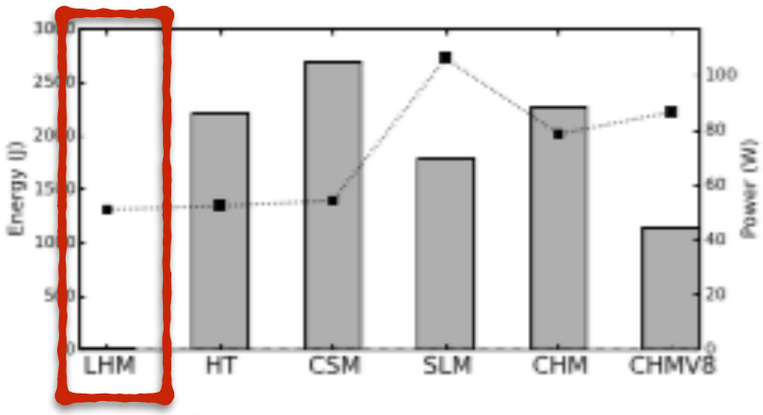
Insertion



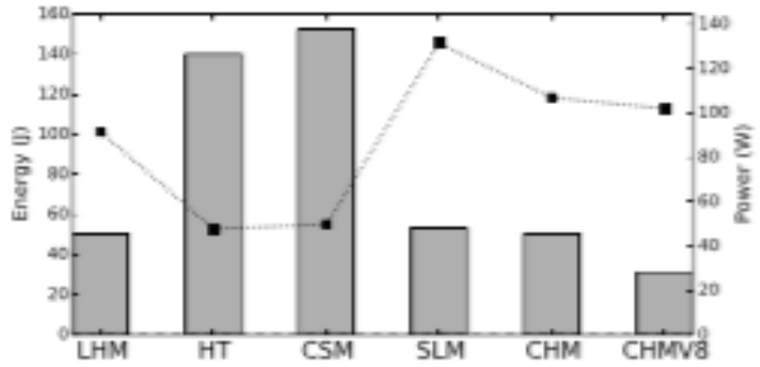
Removal

Maps

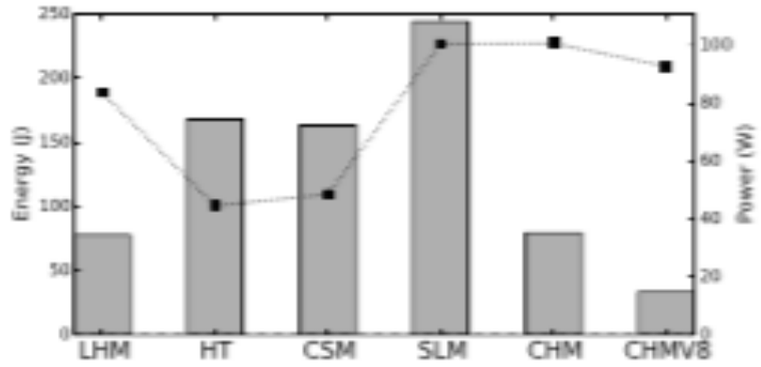
AMD CPU



Traversal

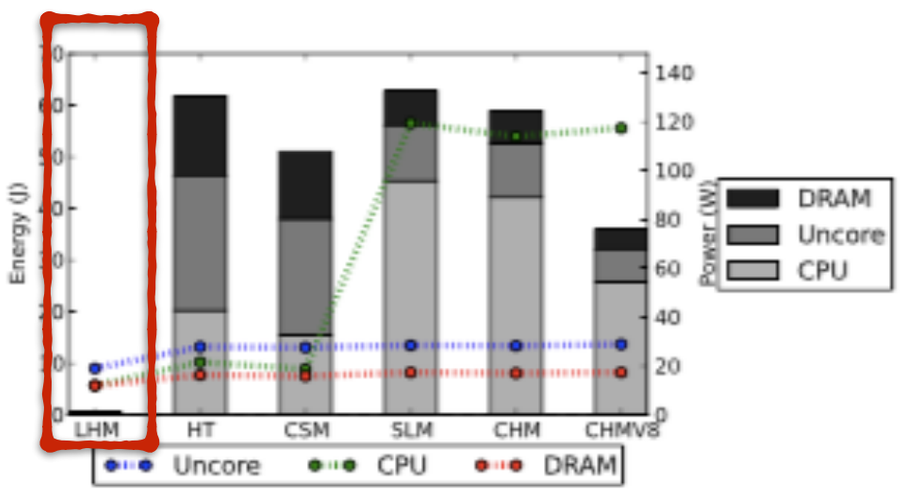


Insertion

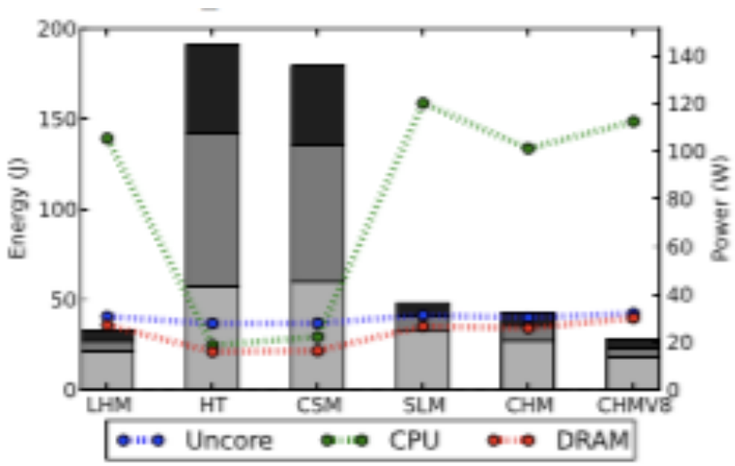


Removal

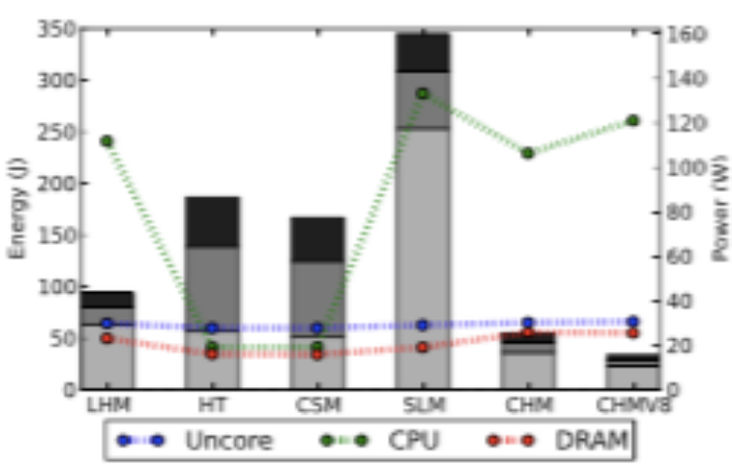
Intel CPU



Traversal



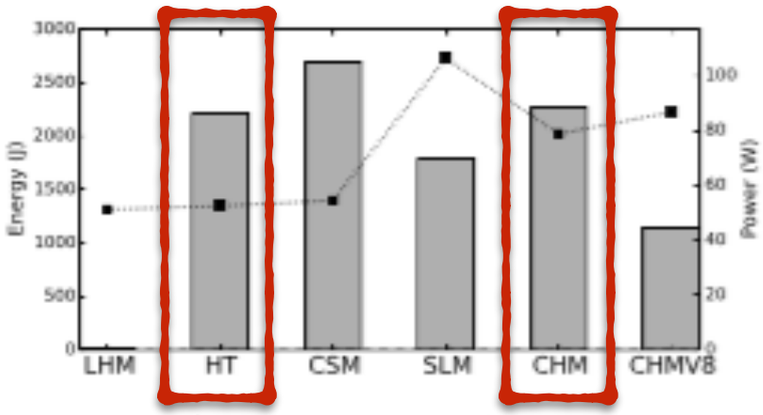
Insertion



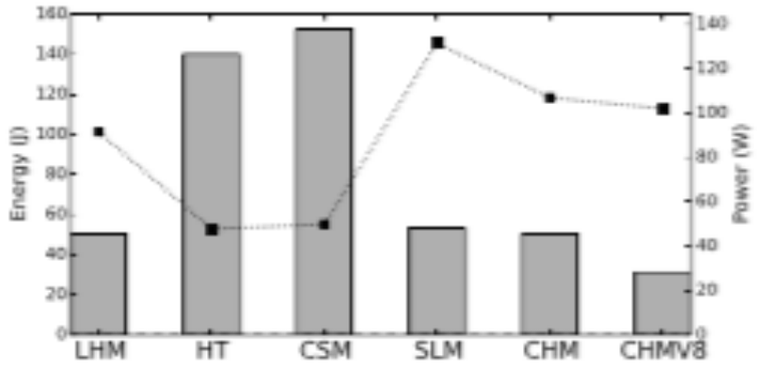
Removal

Maps

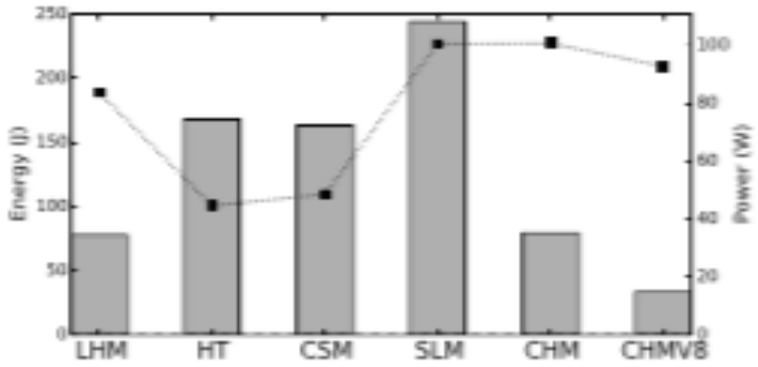
AMD CPU



Traversal

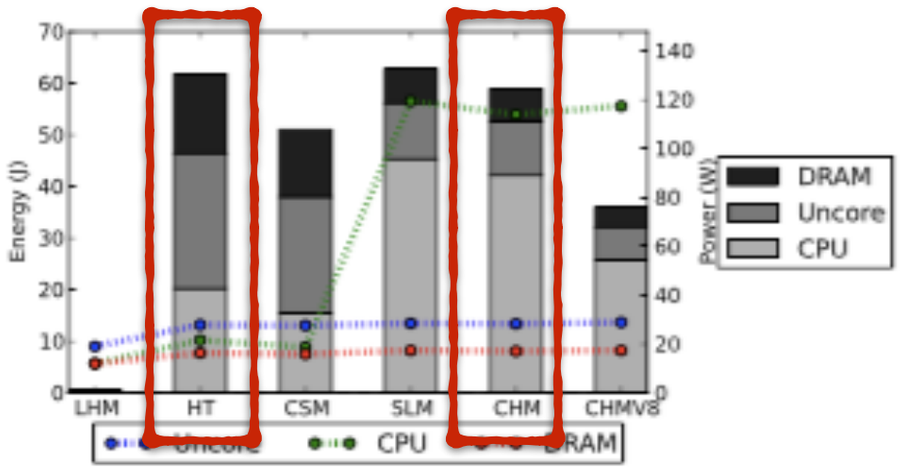


Insertion

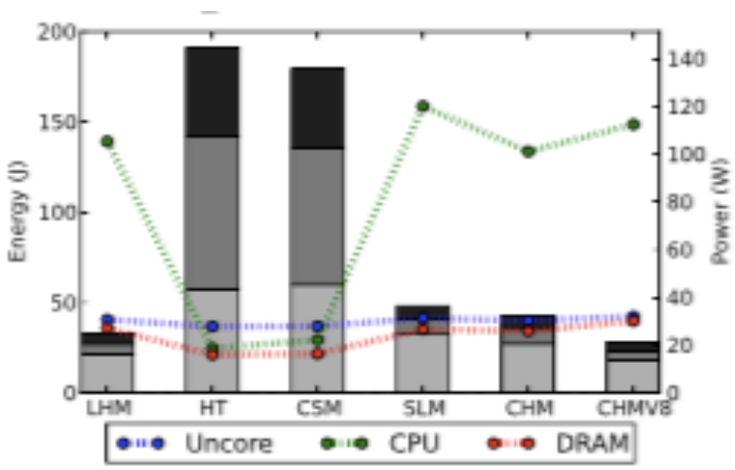


Removal

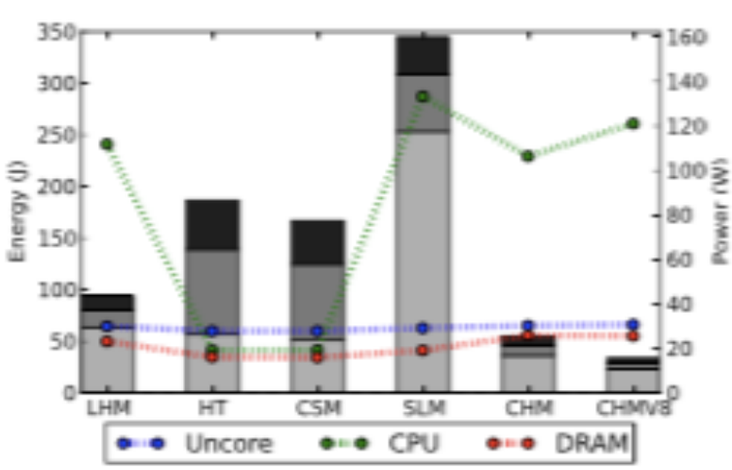
Intel CPU



Traversal



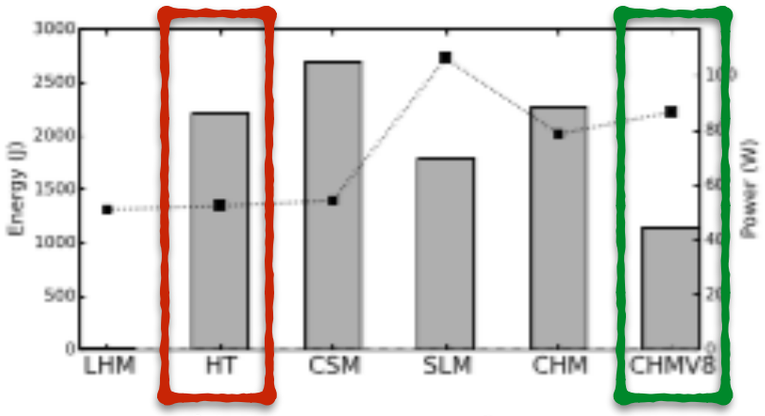
Insertion



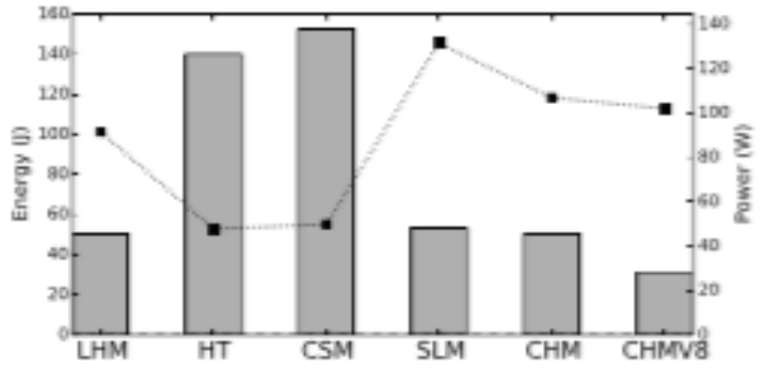
Removal

Maps

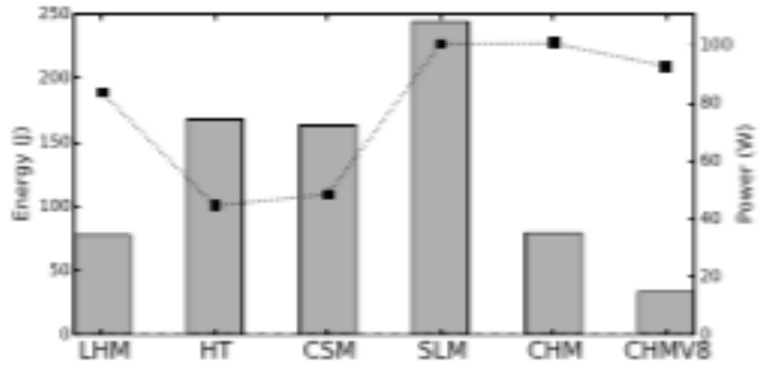
AMD CPU



Traversal

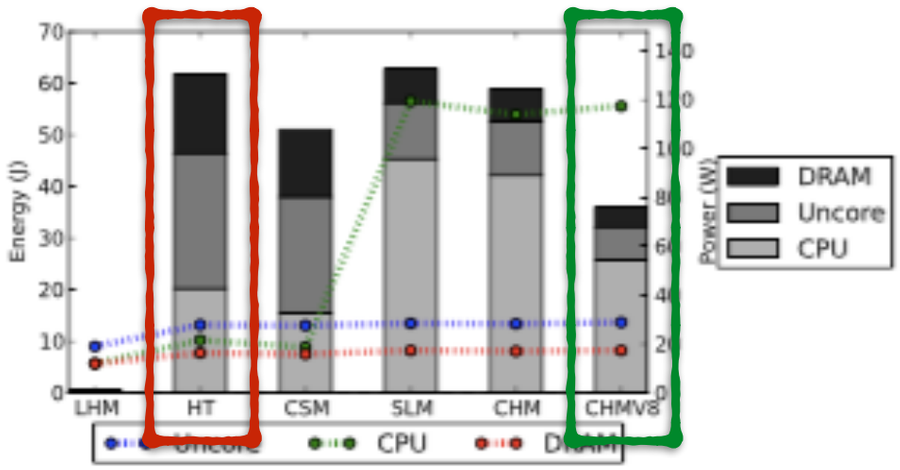


Insertion

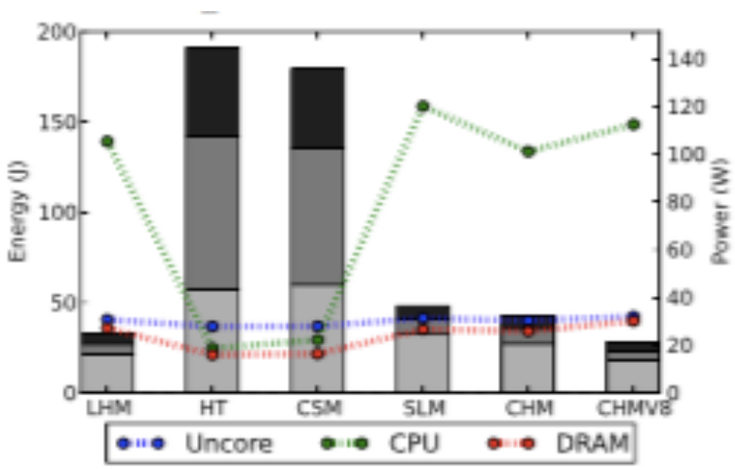


Removal

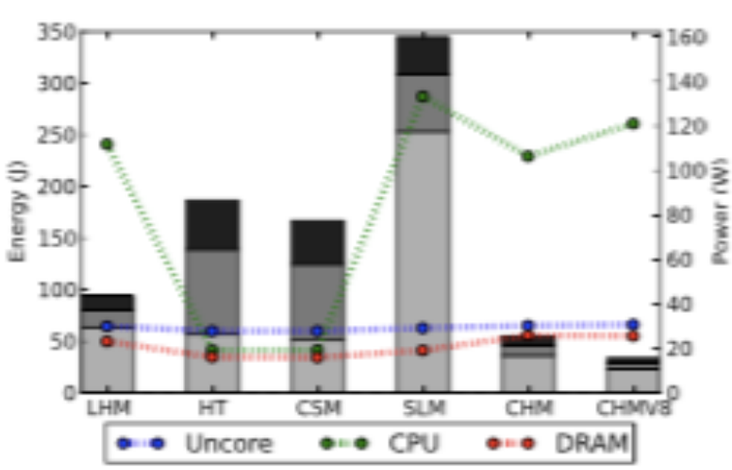
Intel CPU



Traversal



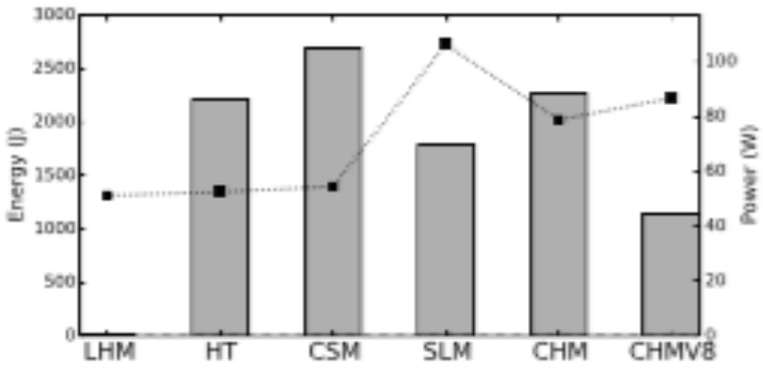
Insertion



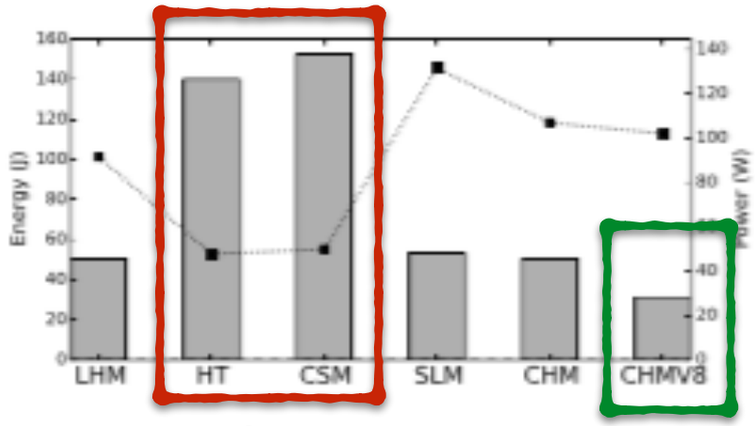
Removal

Maps

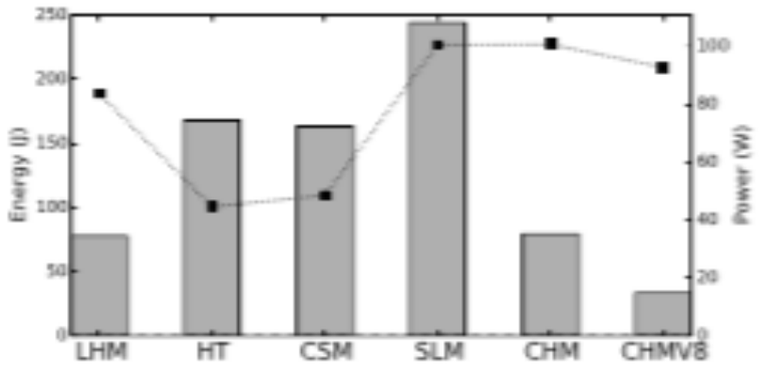
AMD CPU



Traversal

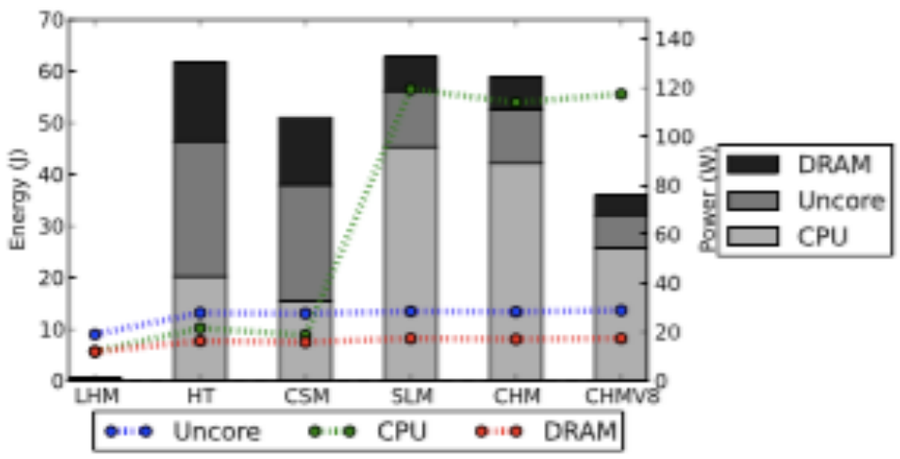


Insertion

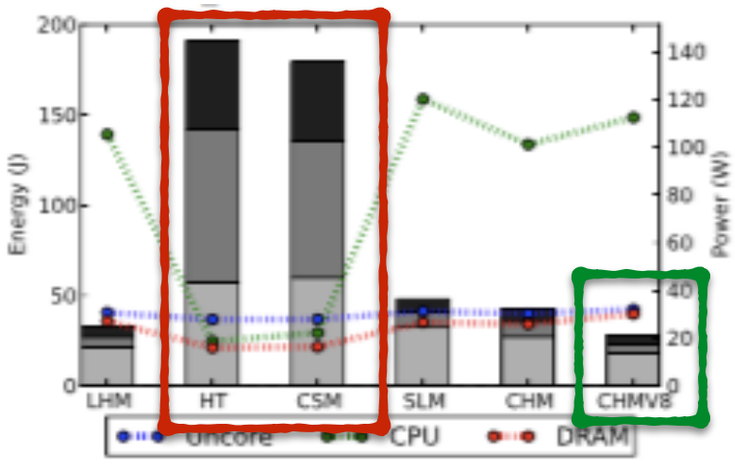


Removal

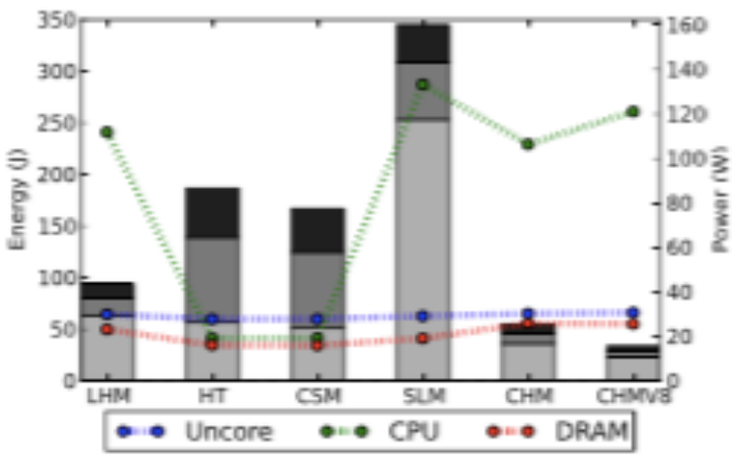
Intel CPU



Traversal



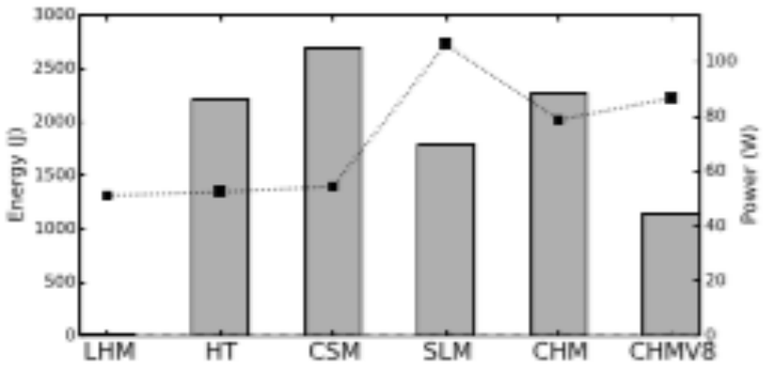
Insertion



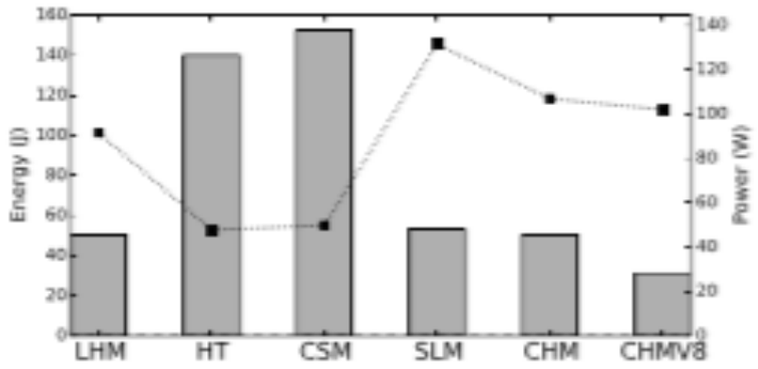
Removal

Maps

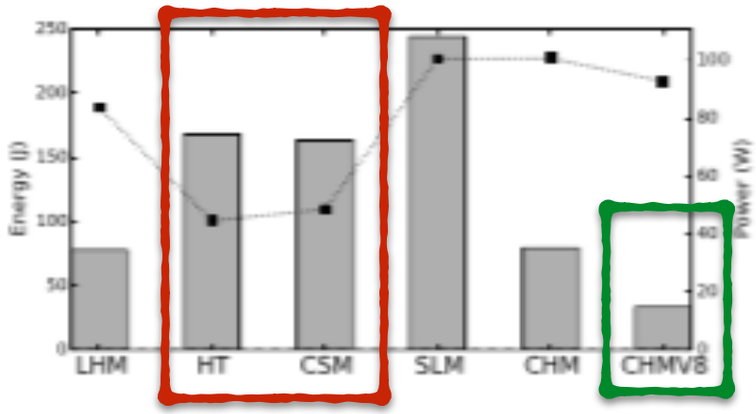
AMD CPU



Traversal

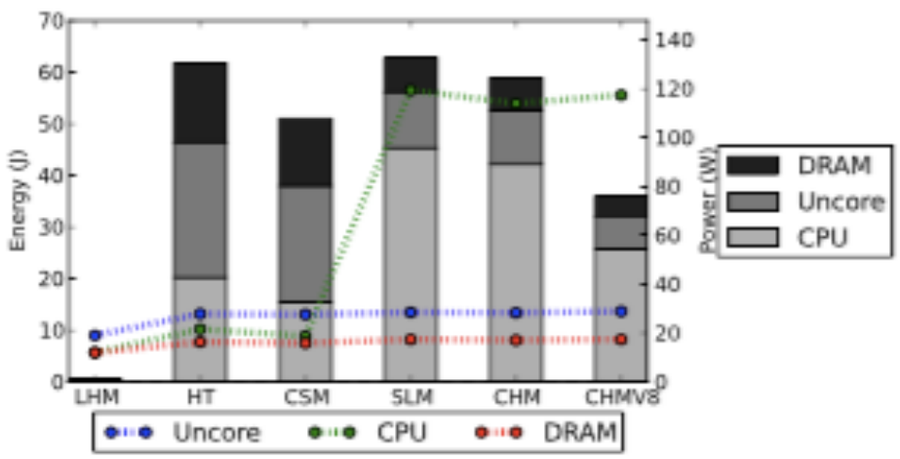


Insertion

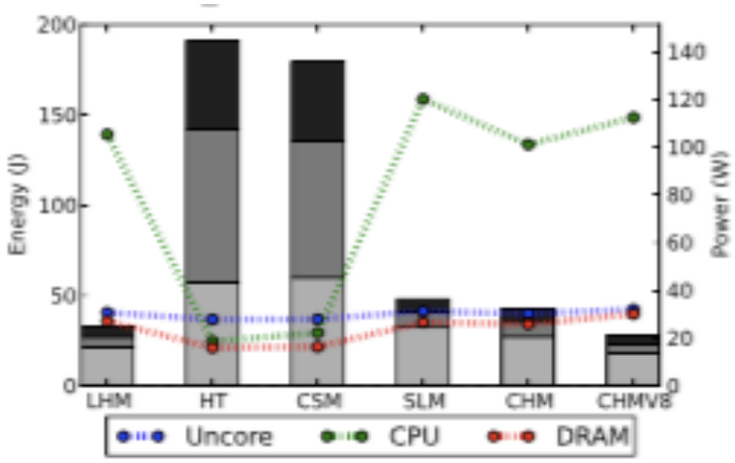


Removal

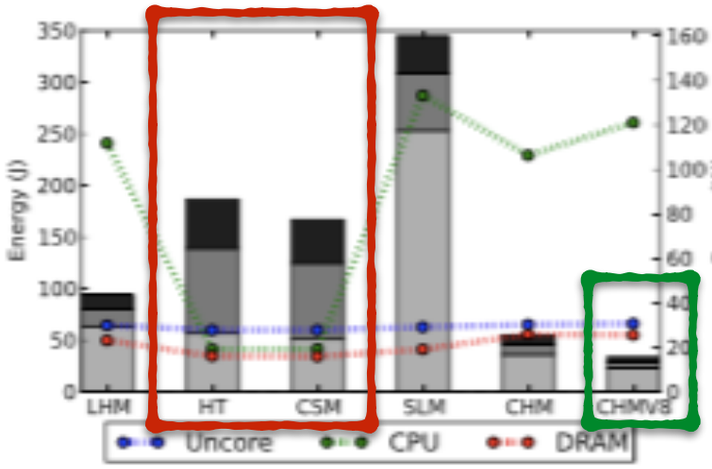
Intel CPU



Traversal



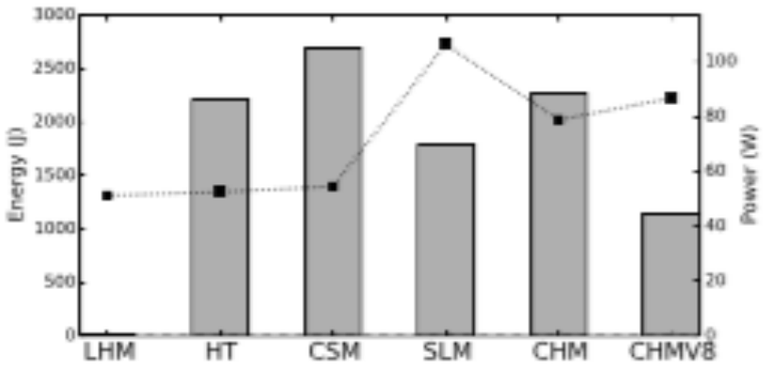
Insertion



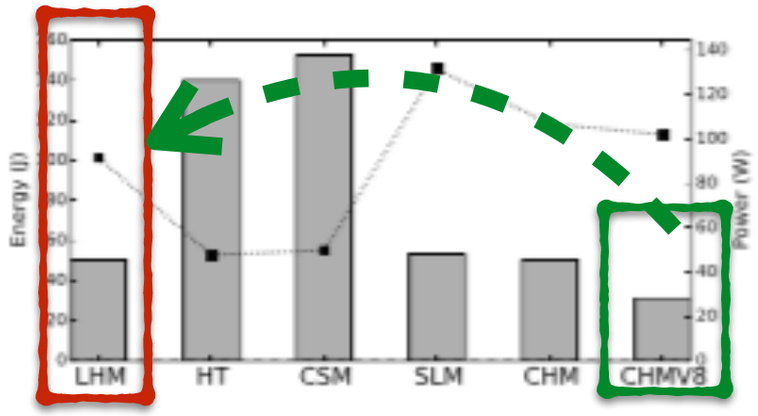
Removal

Maps

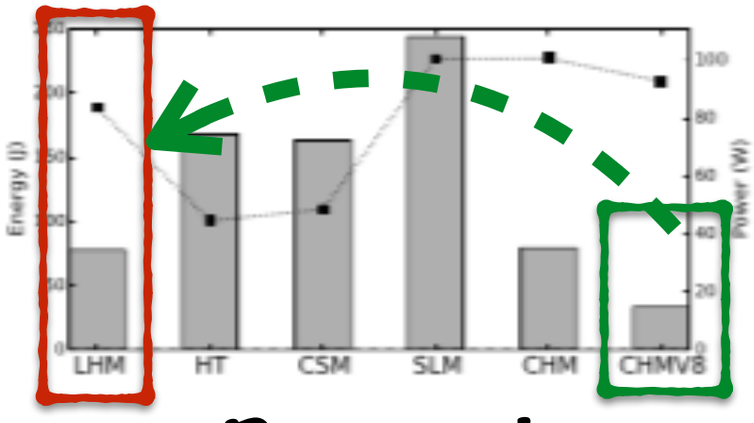
AMD CPU



Traversal

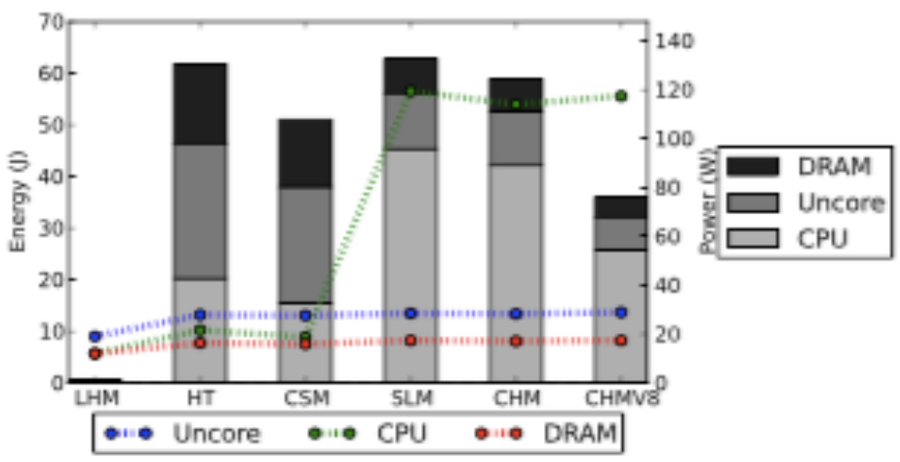


Insertion

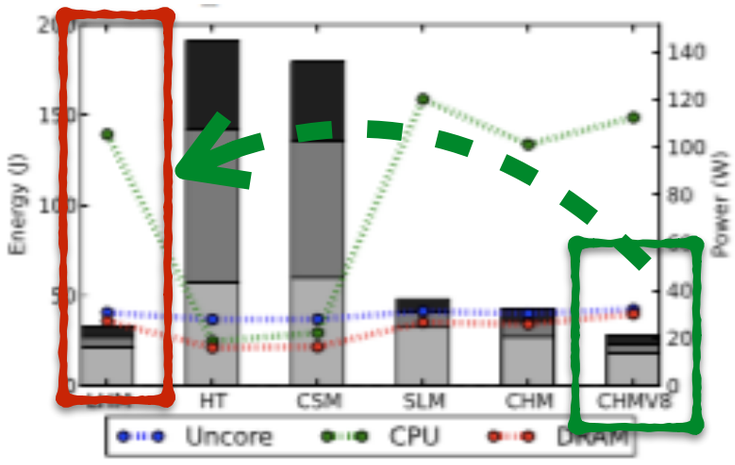


Removal

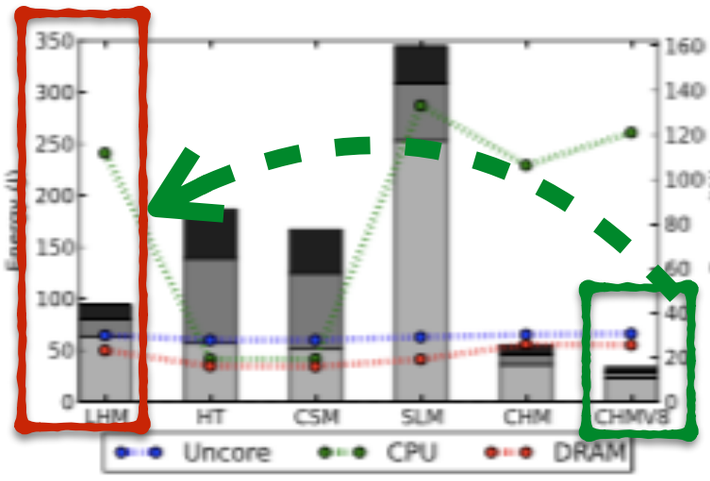
Less energy than the non thread-safe implementation!



Traversal



Insertion



Removal

Case Study

the DaCapo benchmark suite

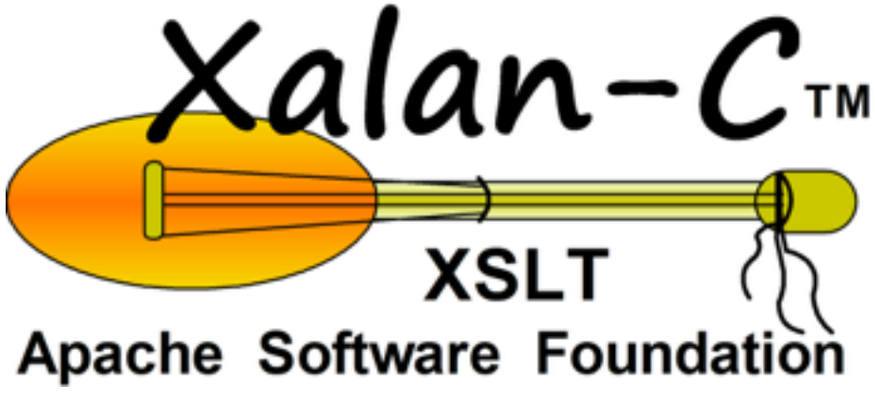
Tomcat

- A web server
- More than 170K lines of Java code
- More than 300 Hashtables

Tomcat

- A web server
- More than 170K lines of Java code
- More than 300 Hashtables

Xalan

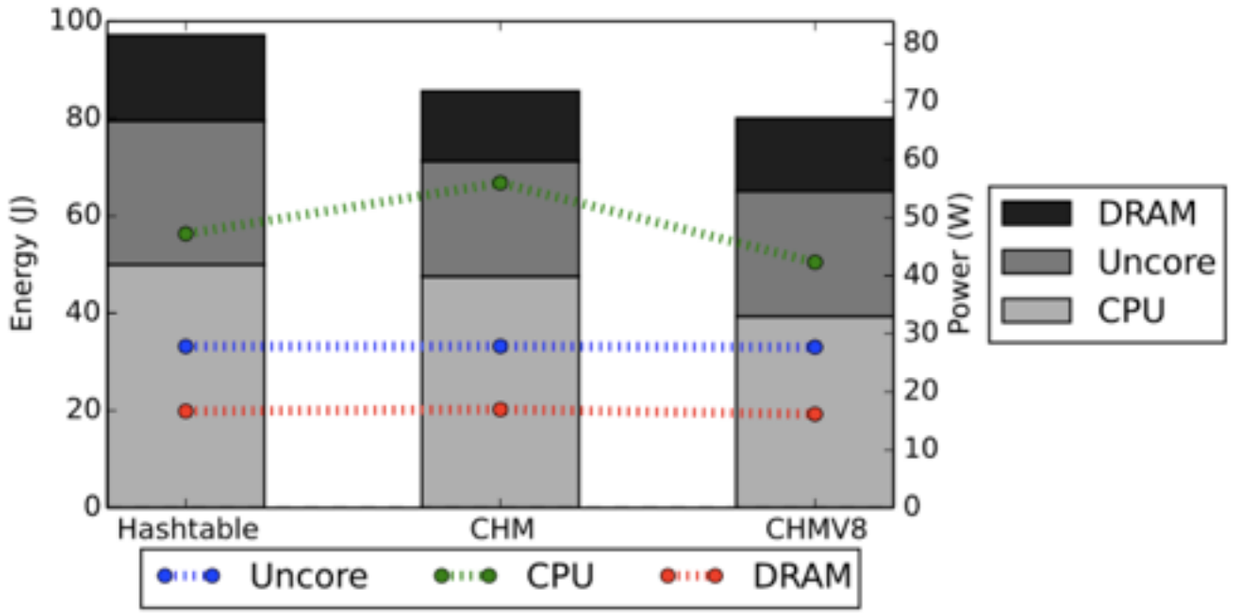


- Parses XML in HTML documents
- More than 188K lines of Java code
- More than 140 Hashtables

Task:

For each Hashtable instance, change it for a ConcurrentHashMap one. Do it again for ConcurrentHashMapV8

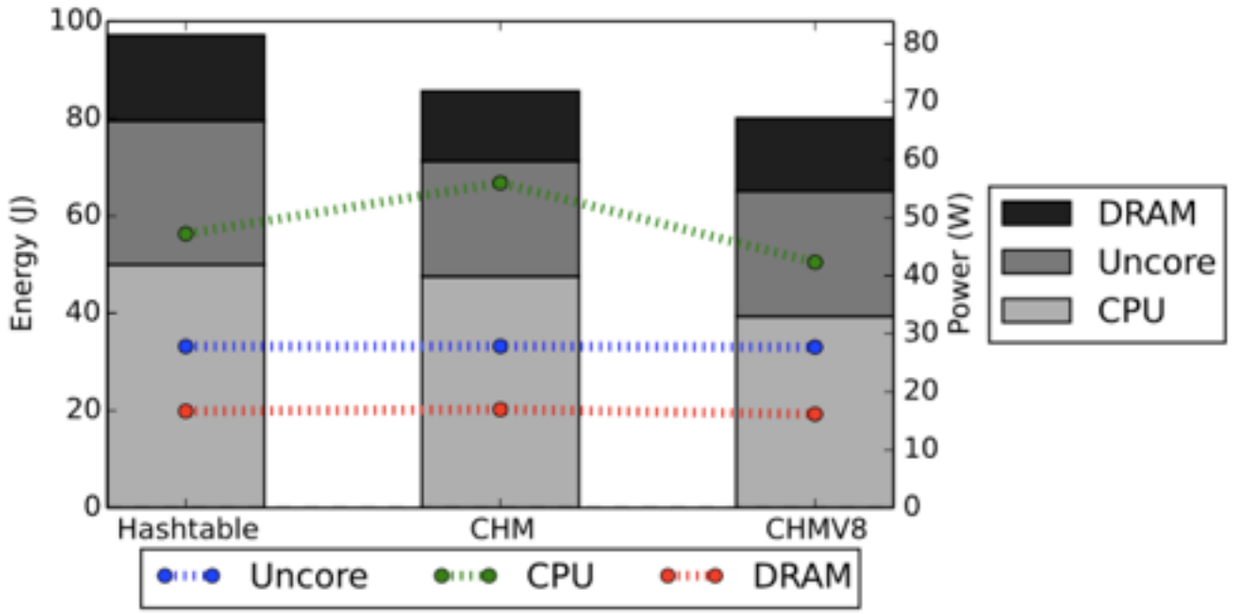
Tomcat



Hashtable to CHM: -12.21%

Hashtable to CHM8: -17.82%

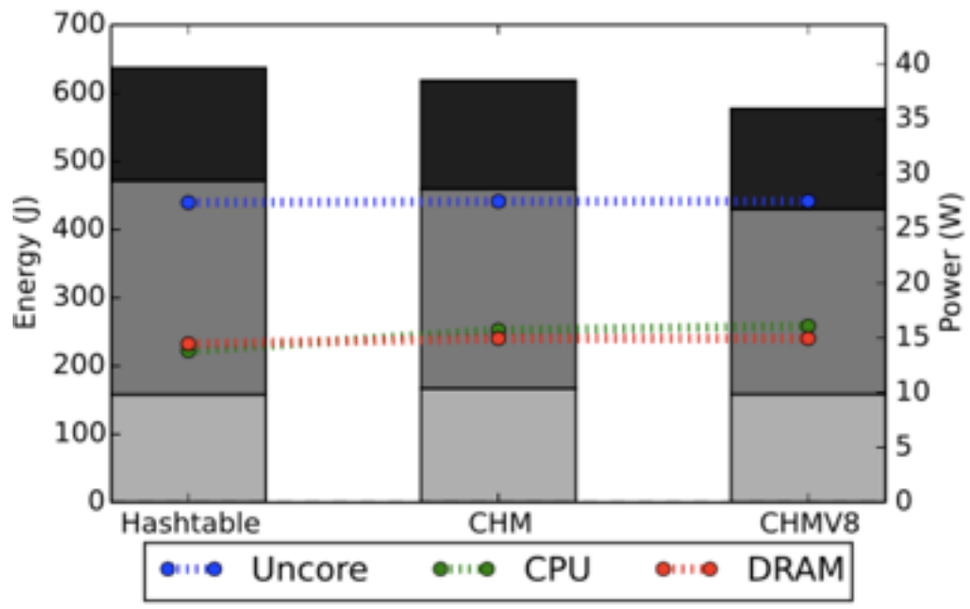
Tomcat



Hashtable to CHM: -12.21%

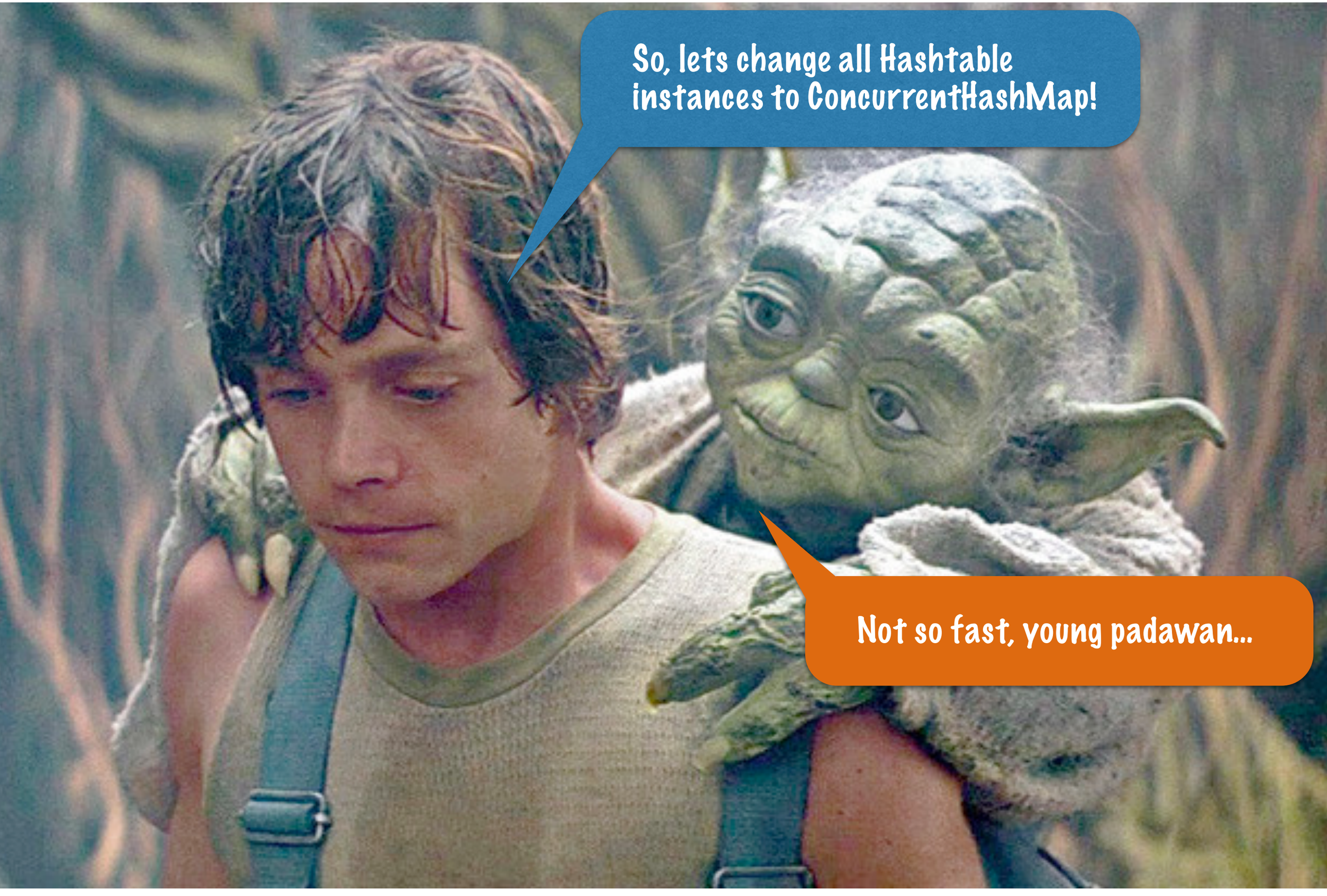
Hashtable to CHM8: -17.82%

Xalan

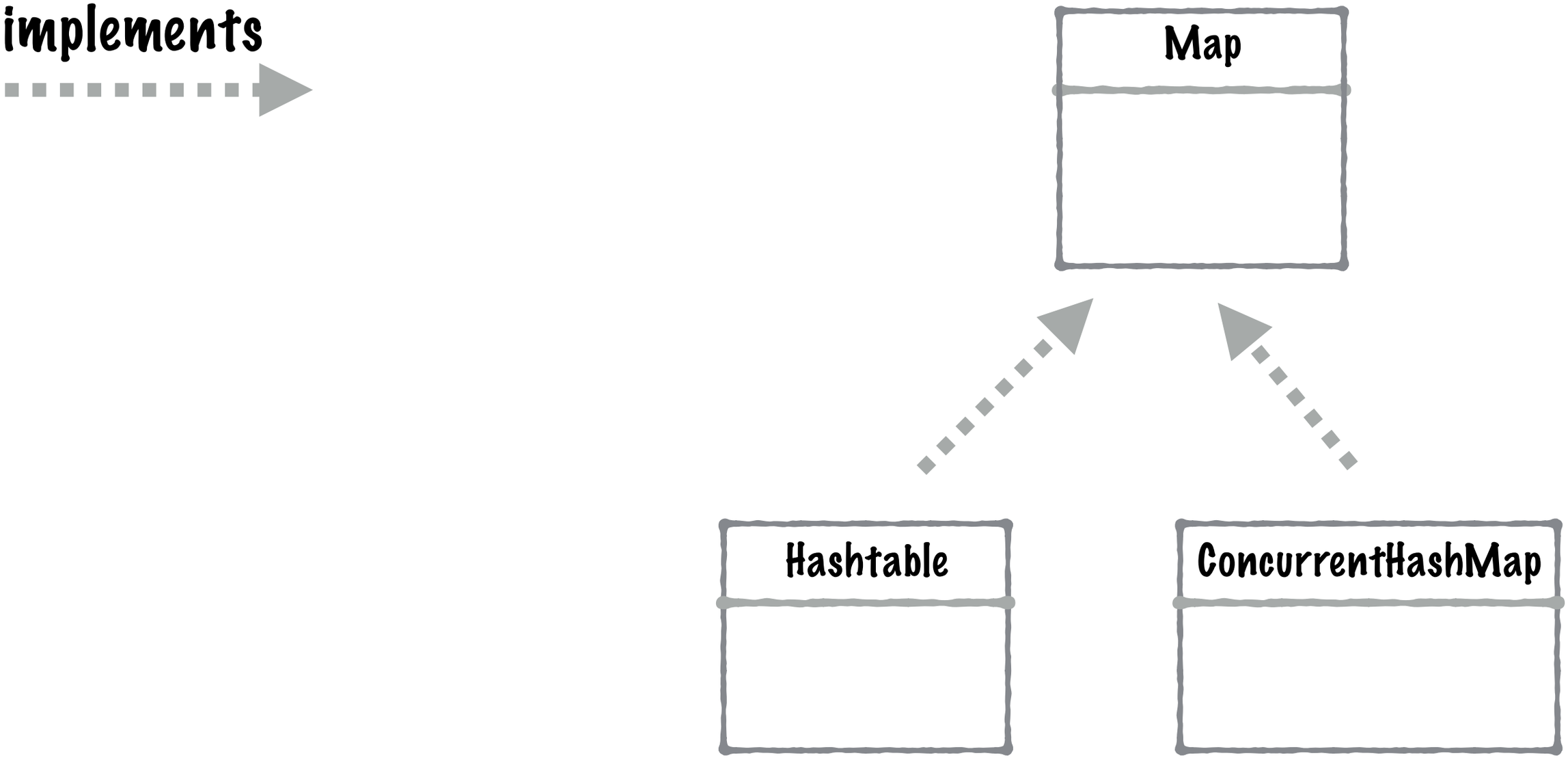


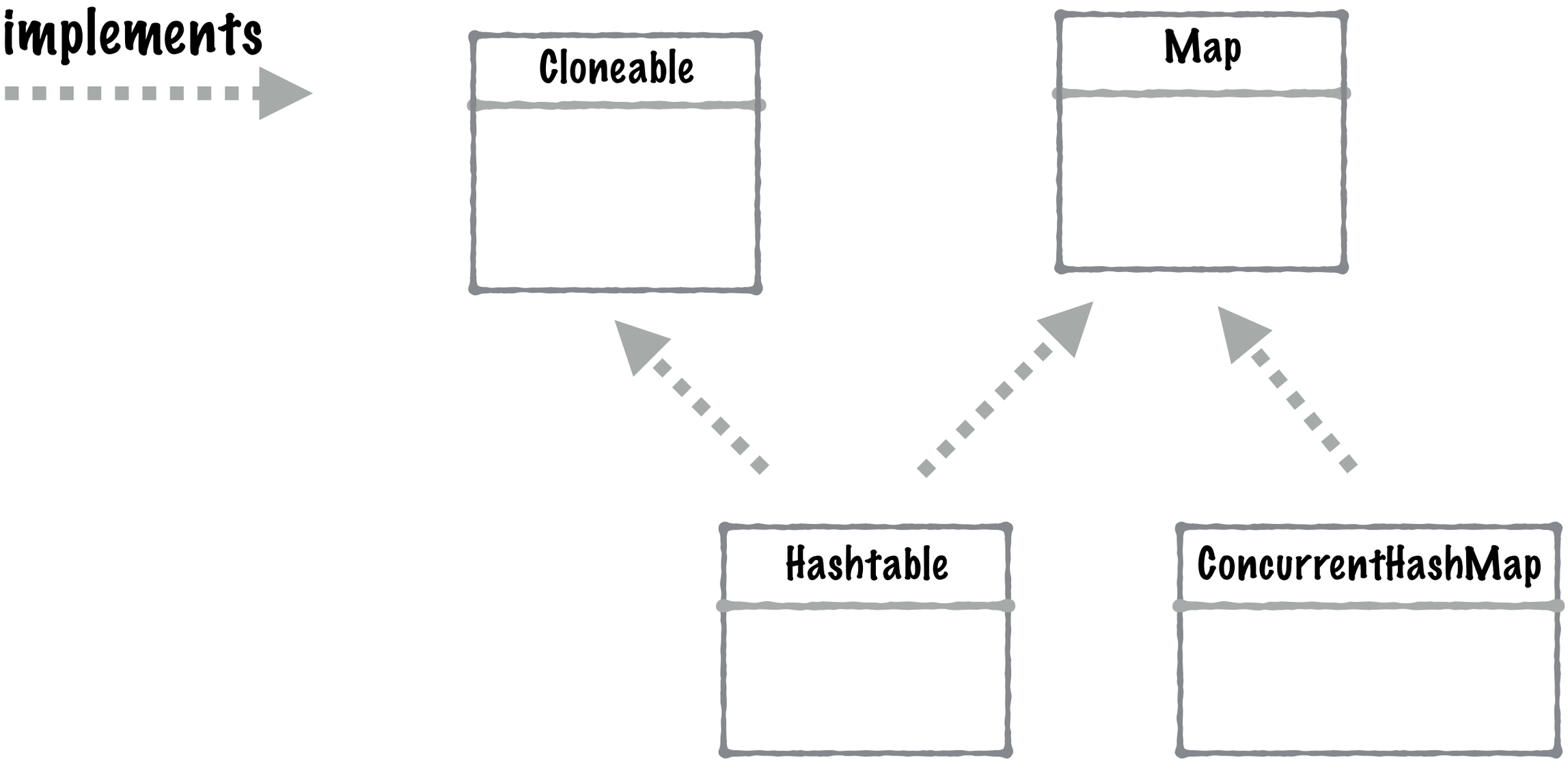
Hashtable to CHM: -5.82%

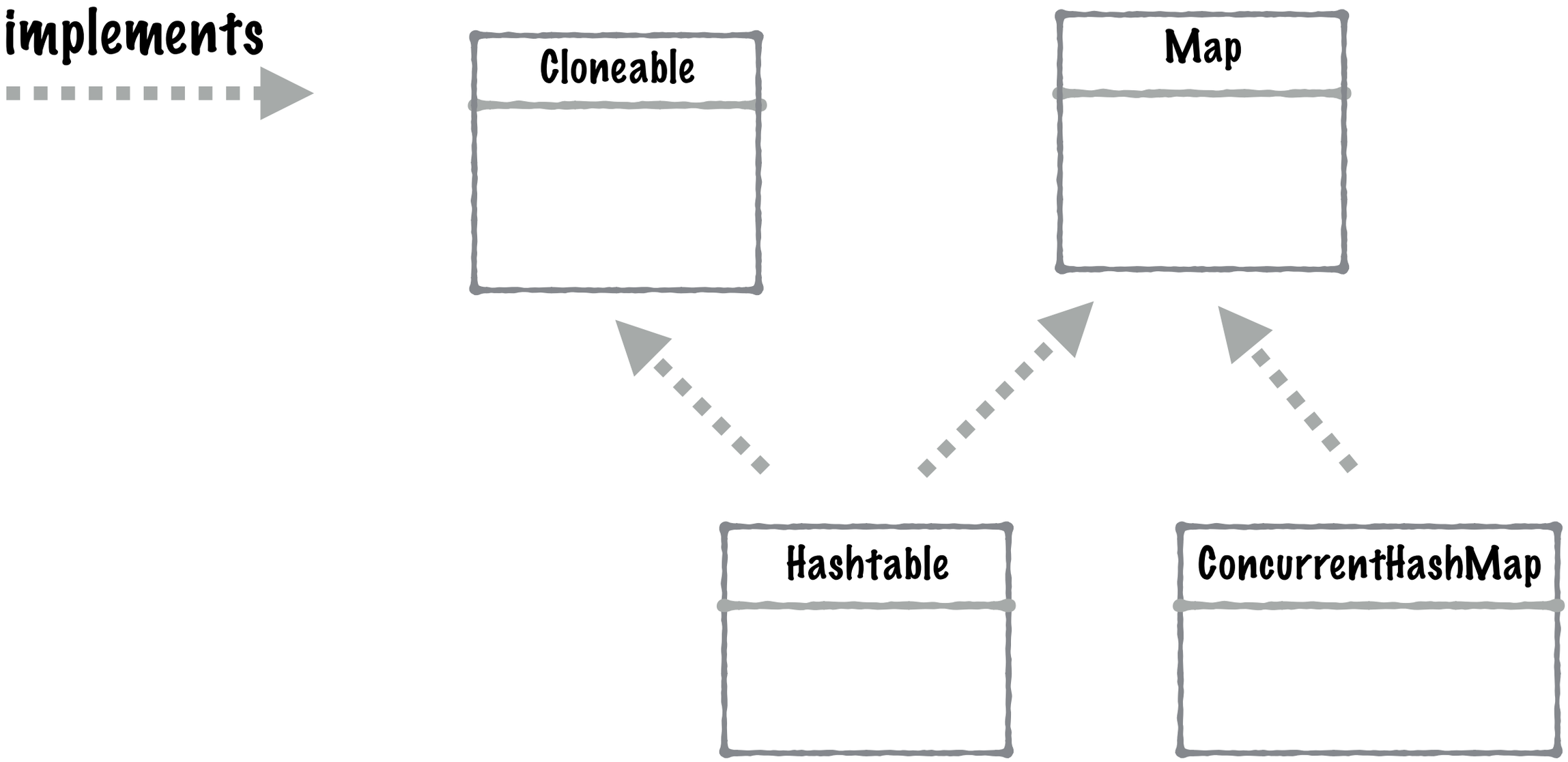
Hashtable to CHM8: -9.32%


So, lets change all Hashtable instances to ConcurrentHashMap!
Not so fast, young padawan...

implements

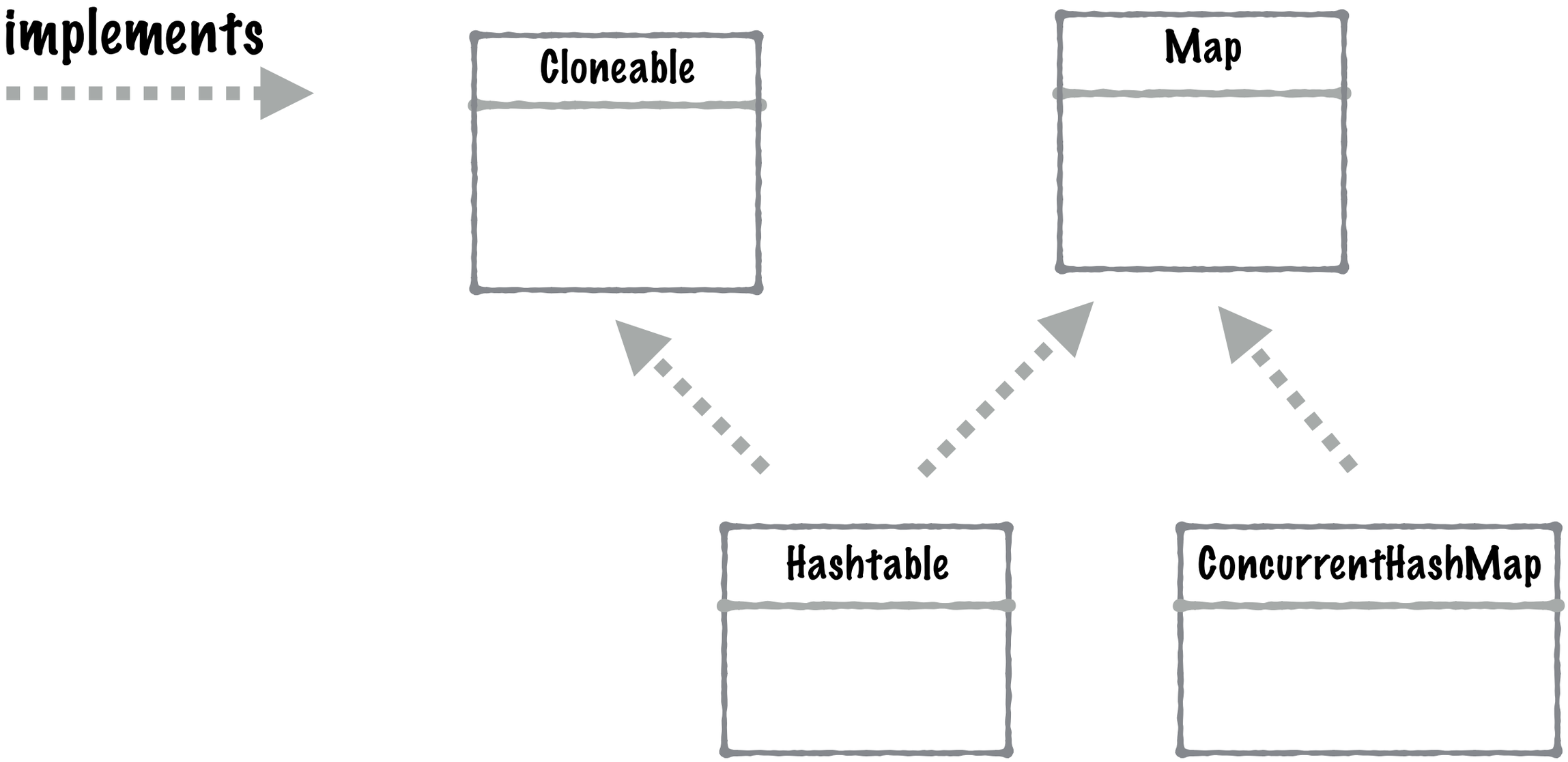



implements
Cloneable
Map
Hashtable
ConcurrentHashMap




```
List<Object> obj = new Hashtable<>();
obj.clone();
```

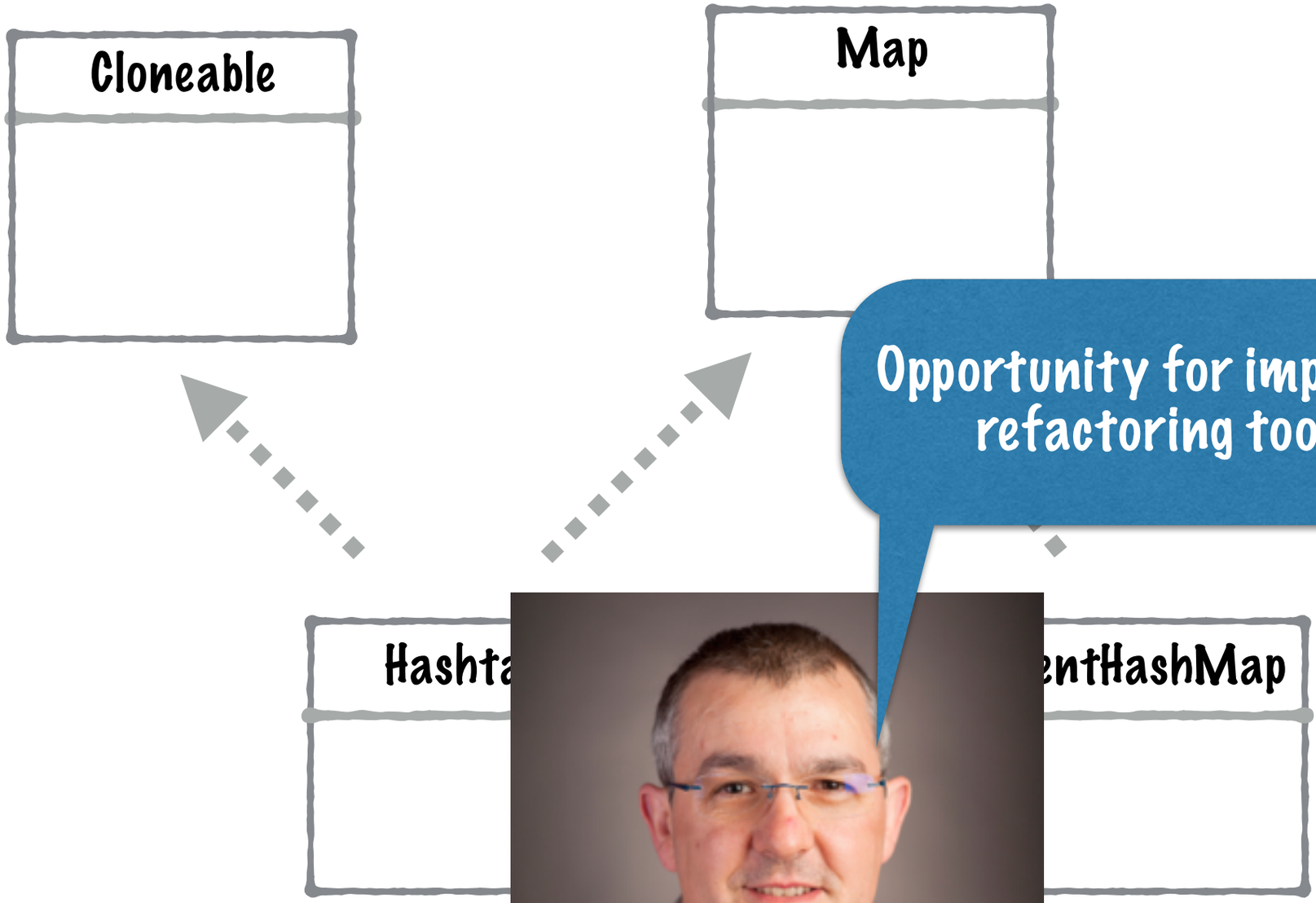
// works fine




```
List<Object> obj = new Hashtable<>();
obj.clone();                                    // works fine

List<Object> obj = new ConcurrentHashMap<>();
obj.clone();                                    // compiler error
```

implements



Opportunity for improving refactoring tools!

Danny Dig

```
List<Object> obj = new Hashtable<>();
obj.clone();
```

ks fine

```
List<Object> obj = new ConcurrentHashMap<>();
obj.clone();
```

// compiler error

Take Away Messages


Small changes can have GREAT impacts, when it comes to energy consumption

Stay up-to-date with the new releases of the java.util.concurrent library

Doug Lea

In Summary

Experimental Environments

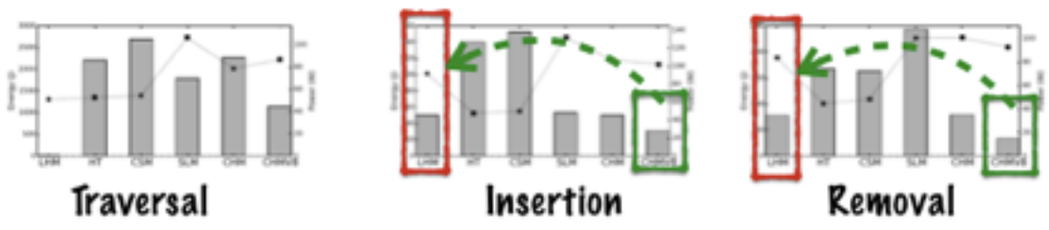
AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

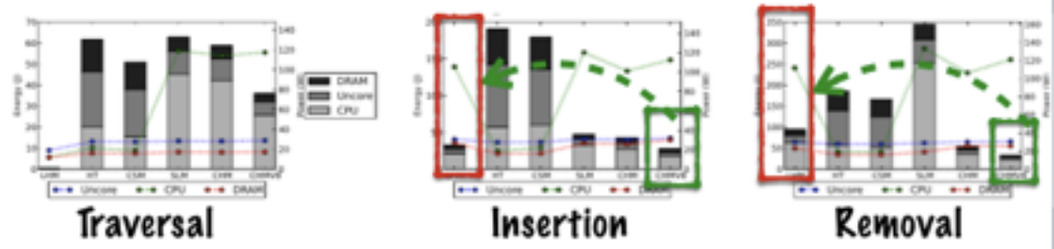
27

Maps

AMD CPU

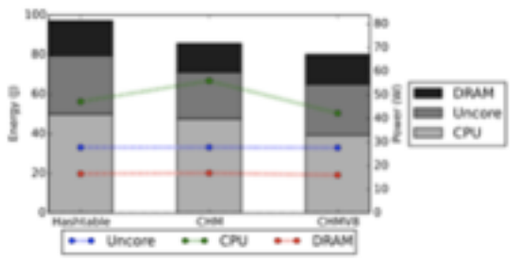


Less energy than the non thread-safe implementation!



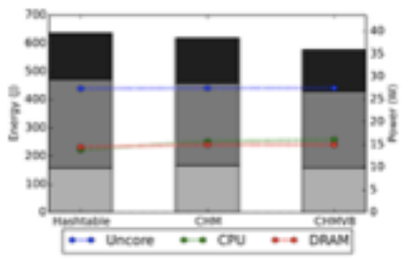
45

Tomcat



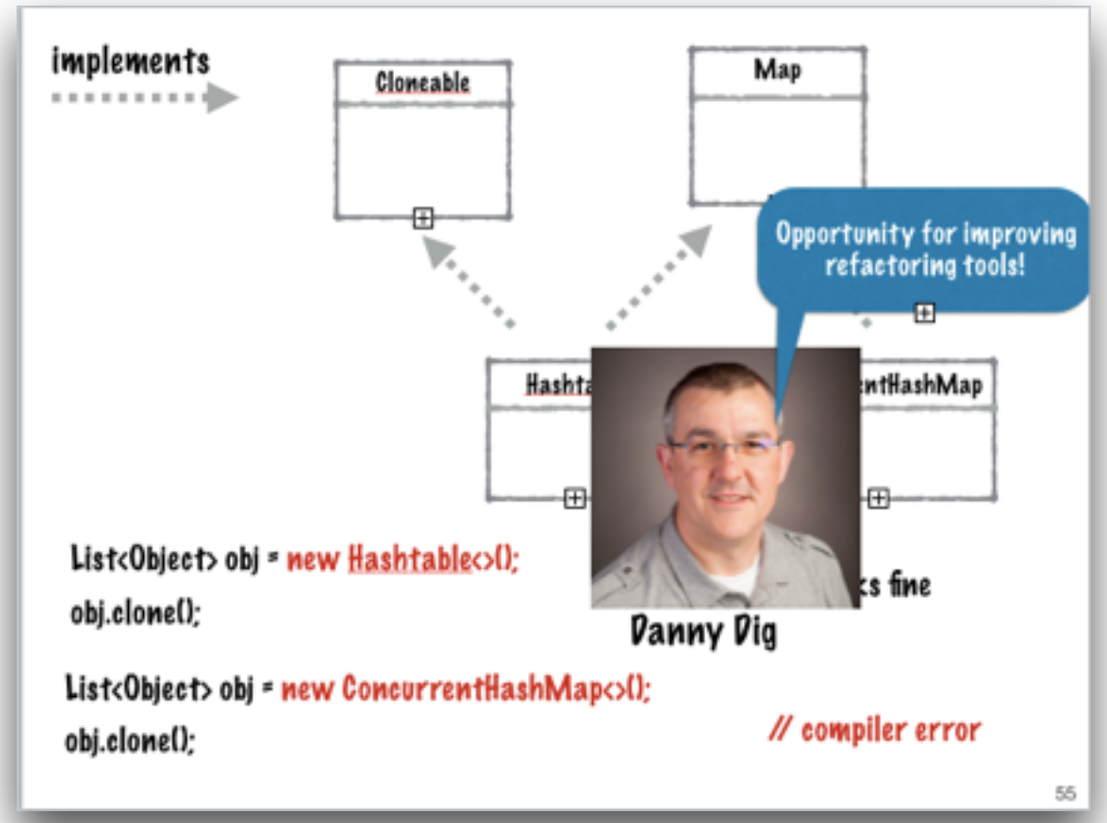
Hashtable to CHM: -12.21%
Hashtable to CHM8: -17.82%

Xalan



Hashtable to CHM: -5.82%
Hashtable to CHM8: -9.32%

51



55

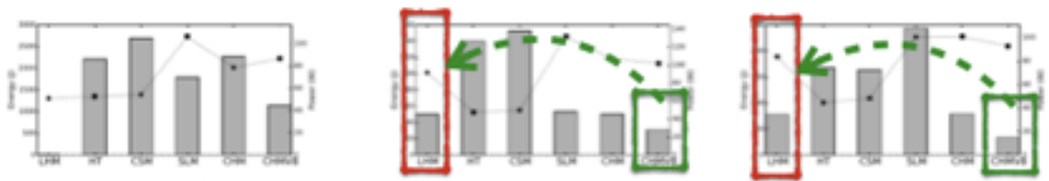
Experimental Environments

AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0_11, build 21.

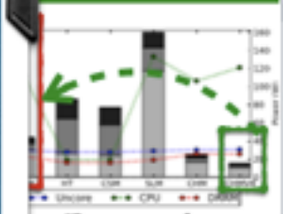
In
ru
ve

Maps

AMD CPU

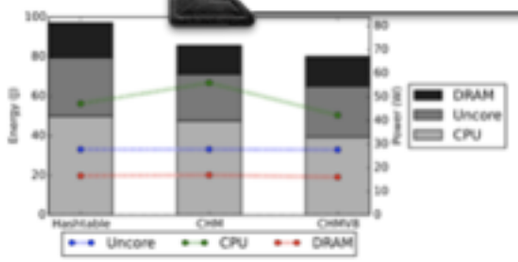


Less energy than the non thread-safe

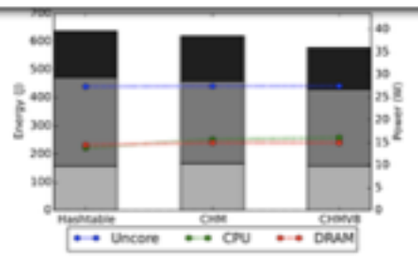


THANKS

Tom



Hashtable to CHM: -12.21%
Hashtable to CHM8: -17.82%



Hashtable to CHM: -5.82%
Hashtable to CHM8: -9.32%

51


```
List<Object> obj = new Hashtable<>();
obj.clone();
```

s fine

Danny Dig

```
List<Object> obj = new ConcurrentHashMap<>();
obj.clone();
```

// compiler error

55

A Comprehensive Study on the Energy Efficiency of Java Thread-Safe Collections

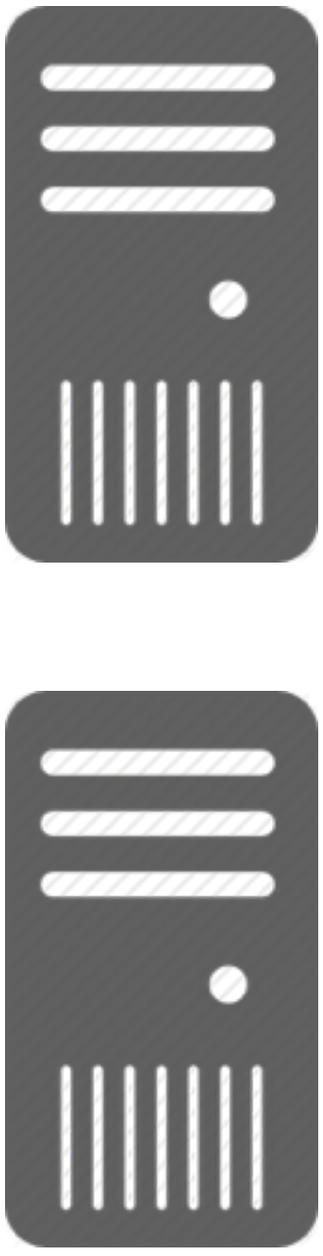
Gustavo Pinto

Kenan Liu

Fernando Castor

David Liu

Experimental Environments



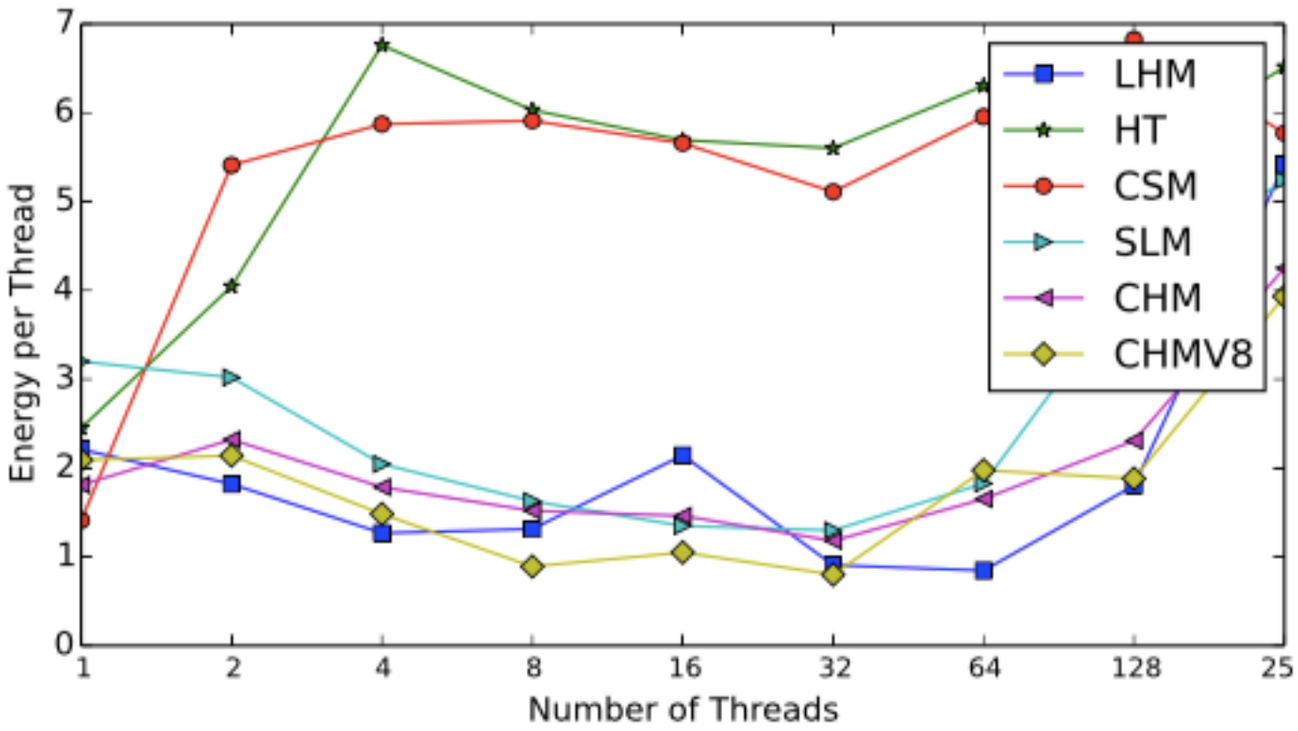
AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0_11, build 21.

Intel CPU: A 2×8-core
running Debian, 2.60G
version 1.7.0_71, build 1

Maps (scaling number of threads)

Intel CPU

Traversal



Insertion