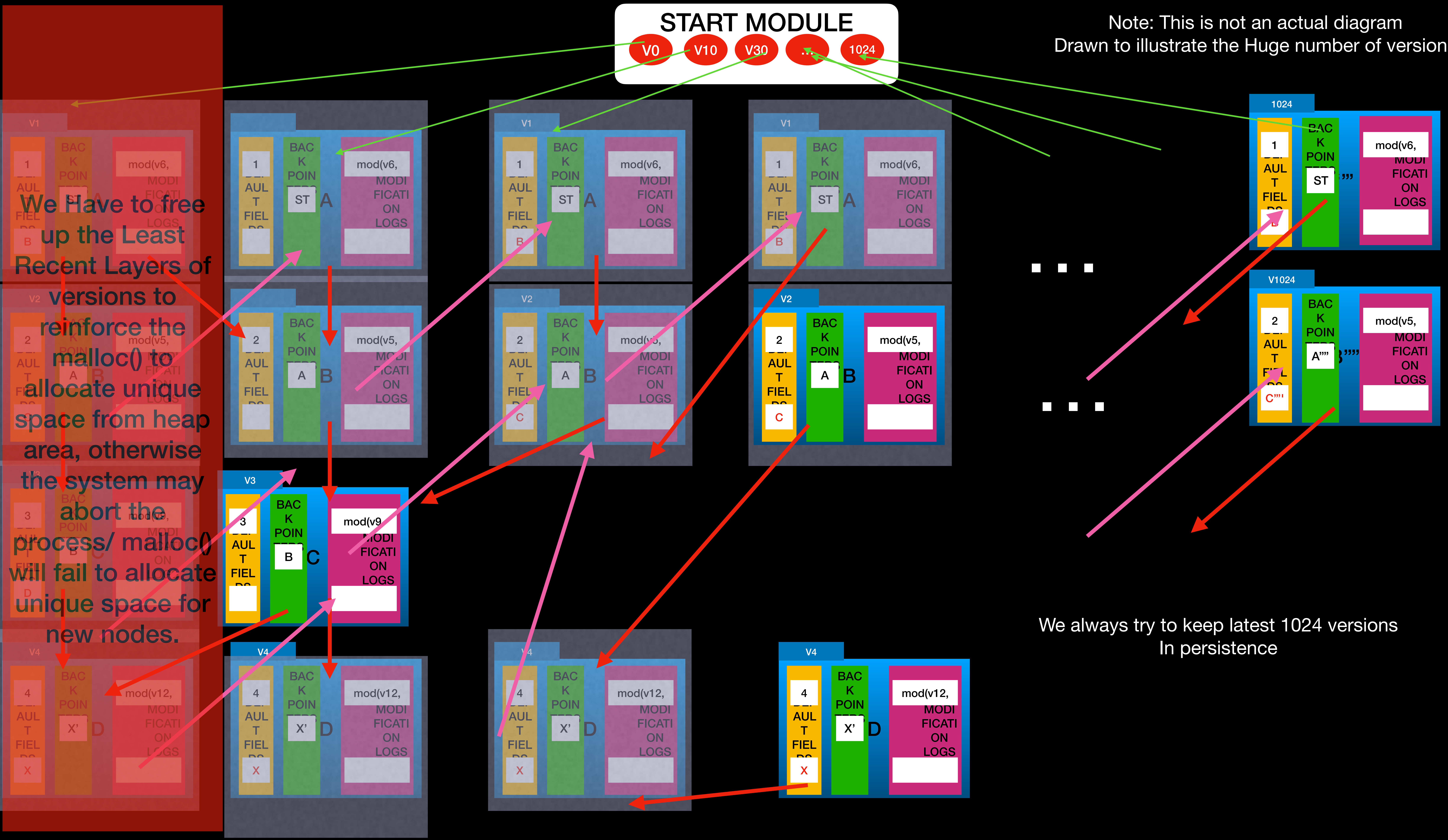
Persistent Data structure

PROJECT MEMBERS: ANANNYO DEY, SOUMYAJIT RUDRA SARMA , DEBASMIT ROY, KANKO GHOSH AND KUSHAL DAS

1. Try To Keep A Limited Number of Latest Version in Persistence

Because, the PPDS are very memory consuming.

Note: This is not an actual diagram
Drawn to illustrate the Huge number of versions

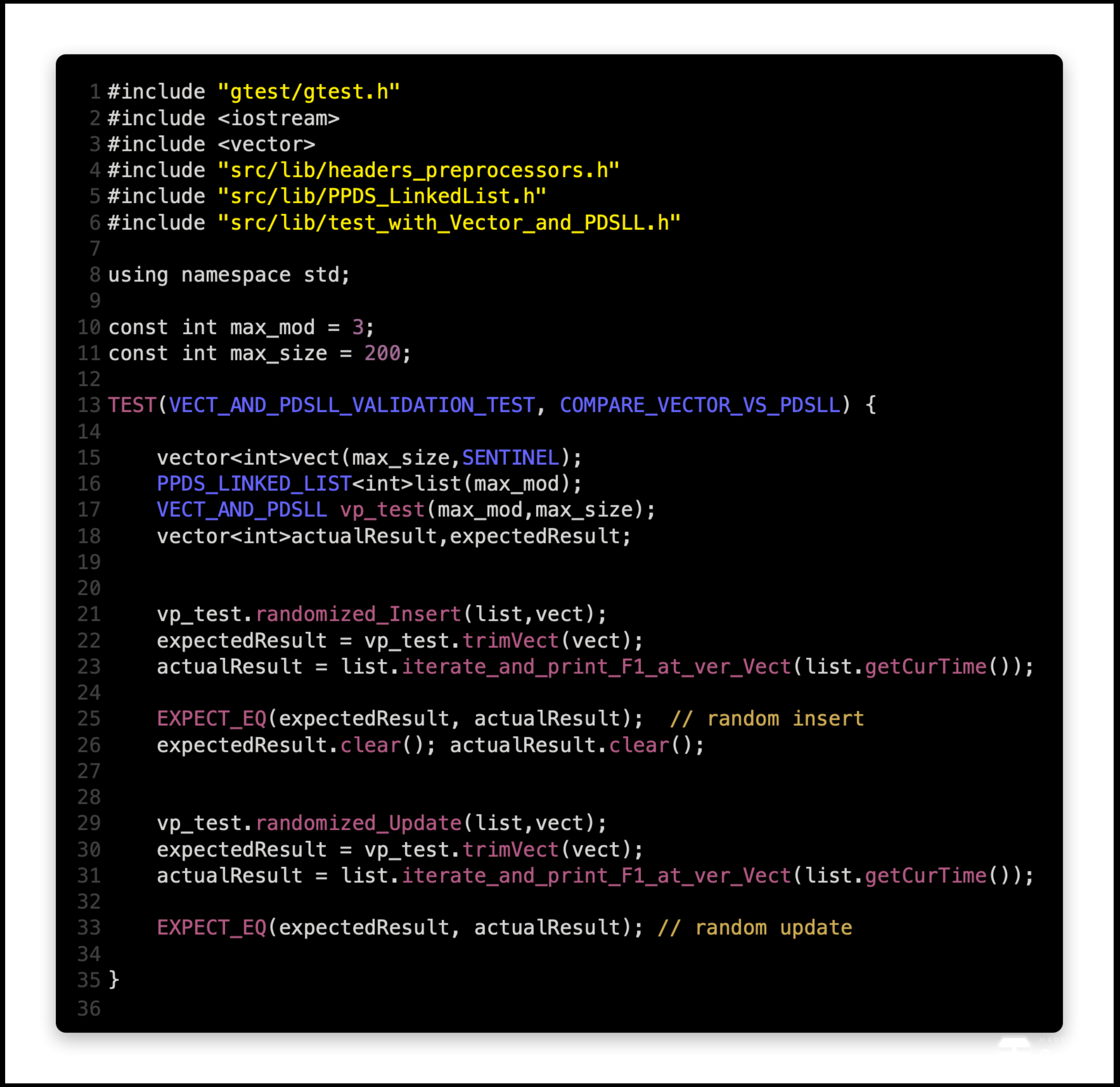


We Have to free up the Least Recent Layers of versions to reinforce the malloc() to allocate unique space from heap area, otherwise the system may abort the process/ malloc() will fail to allocate unique space for new nodes.

We always try to keep latest 1024 versions
In persistence

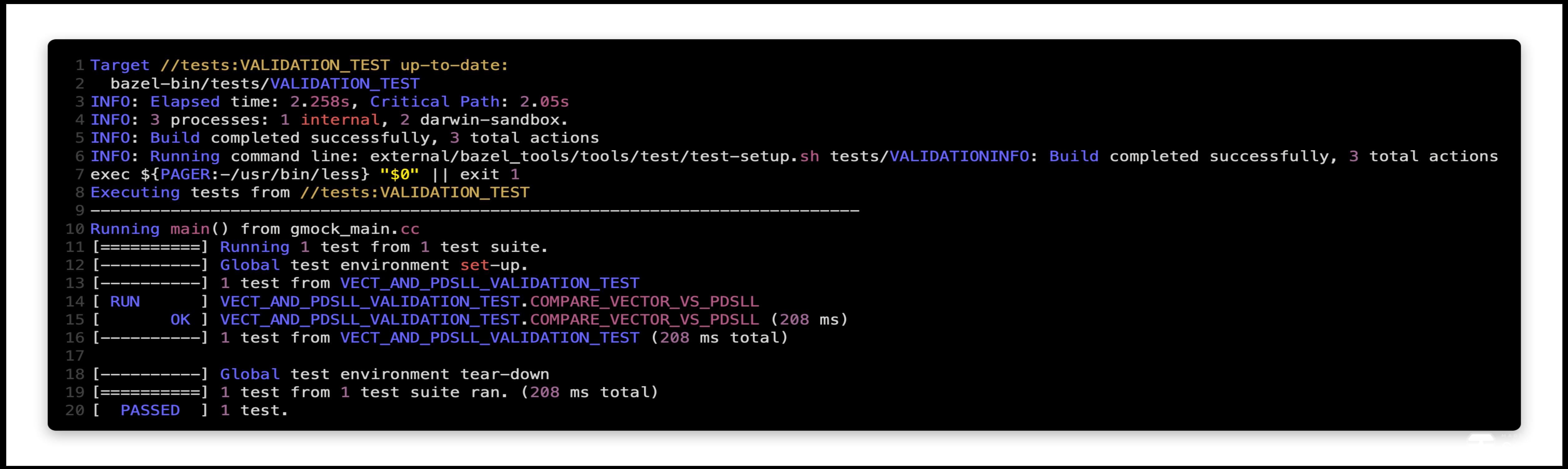
Supporting Code

Unit Test To Validate The Expected Result And Actual Result



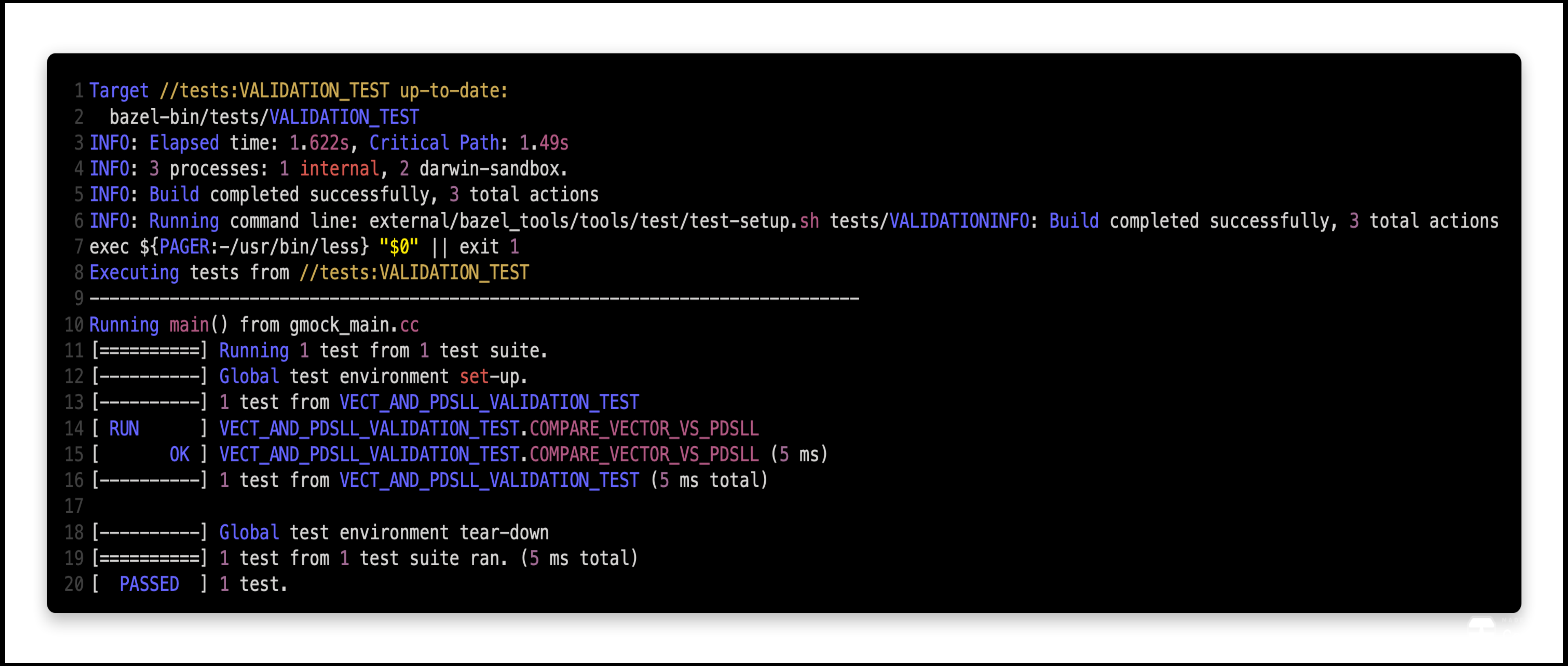
```cpp
#include "gtest/gtest.h"
#include <iostream>
#include <vector>
#include "src/lib/headers_preprocessors.h"
#include "src/lib/PPDS_LinkedList.h"
#include "src/lib/test_with_Vector_and_PDSLL.h"

using namespace std;

const int max_mod = 3;
const int max_size = 200;

TEST(VECT_AND_PDSLL_VALIDATION_TEST, COMPARE_VECTOR_VS_PDSLL) {

    vector<int>vect(max_size,SENTINEL);
    PPDS_LINKED_LIST<int>list(max_mod);
    VECT_AND_PDSLL vp_test(max_mod,max_size);
    vector<int>actualResult,expectedResult;


    vp_test.randomized_Insert(list,vect);
    expectedResult = vp_test.trimVect(vect);
    actualResult = list.iterate_and_print_F1_at_ver_Vect(list.getCurTime());

    EXPECT_EQ(expectedResult, actualResult);  // random insert
    expectedResult.clear(); actualResult.clear();


    vp_test.randomized_Update(list,vect);
    expectedResult = vp_test.trimVect(vect);
    actualResult = list.iterate_and_print_F1_at_ver_Vect(list.getCurTime());

    EXPECT_EQ(expectedResult, actualResult); // random update

}

```

max_size = 200



```
Target //tests:VALIDATION_TEST up-to-date:
  bazel-bin/tests/VALIDATION_TEST
INFO: Elapsed time: 2.258s, Critical Path: 2.05s
INFO: 3 processes: 1 internal, 2 darwin-sandbox.
INFO: Build completed successfully, 3 total actions
INFO: Running command line: external/bazel_tools/tools/test/test-setup.sh tests/VALIDATIONINFO: Build completed successfully, 3 total actions
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests:VALIDATION_TEST
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from VECT_AND_PDSLL_VALIDATION_TEST
[ RUN      ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL
[       OK ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL (208 ms)
[----------] 1 test from VECT_AND_PDSLL_VALIDATION_TEST (208 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (208 ms total)
[  PASSED  ] 1 test.
```

max_size = 2000

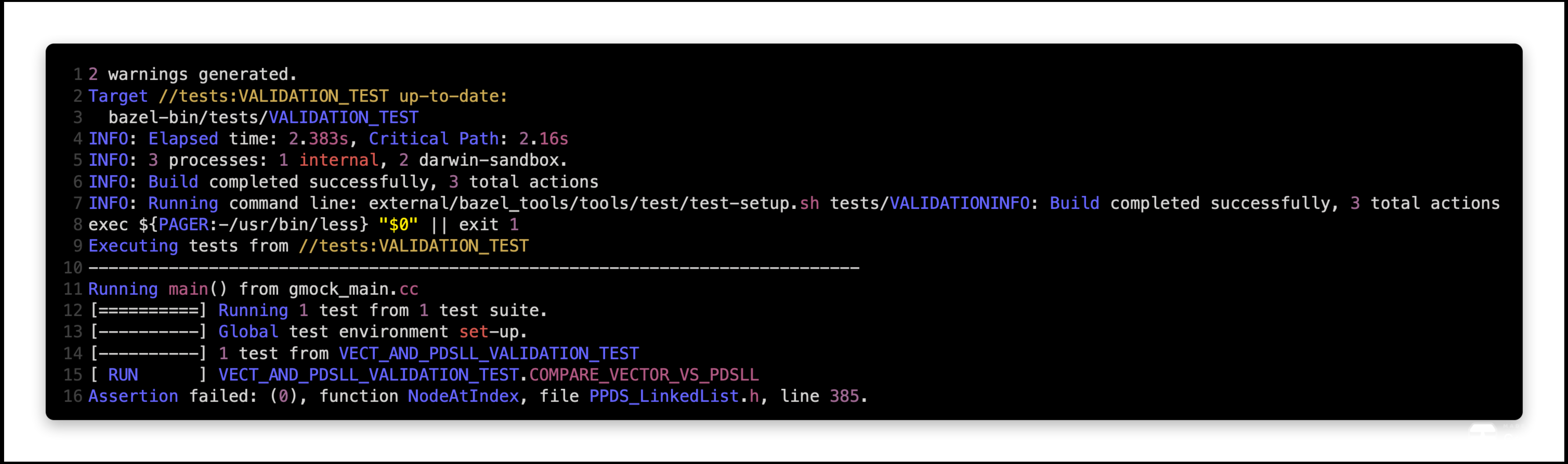


```
Target //tests:VALIDATION_TEST up-to-date:
  bazel-bin/tests/VALIDATION_TEST
INFO: Elapsed time: 1.622s, Critical Path: 1.49s
INFO: 3 processes: 1 internal, 2 darwin-sandbox.
INFO: Build completed successfully, 3 total actions
INFO: Running command line: external/bazel_tools/tools/test/test-setup.sh tests/VALIDATIONINFO: Build completed successfully, 3 total actions
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests:VALIDATION_TEST
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from VECT_AND_PDSLL_VALIDATION_TEST
[ RUN      ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL
[       OK ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL (5 ms)
[----------] 1 test from VECT_AND_PDSLL_VALIDATION_TEST (5 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (5 ms total)
[  PASSED  ] 1 test.
```

These Results Passed

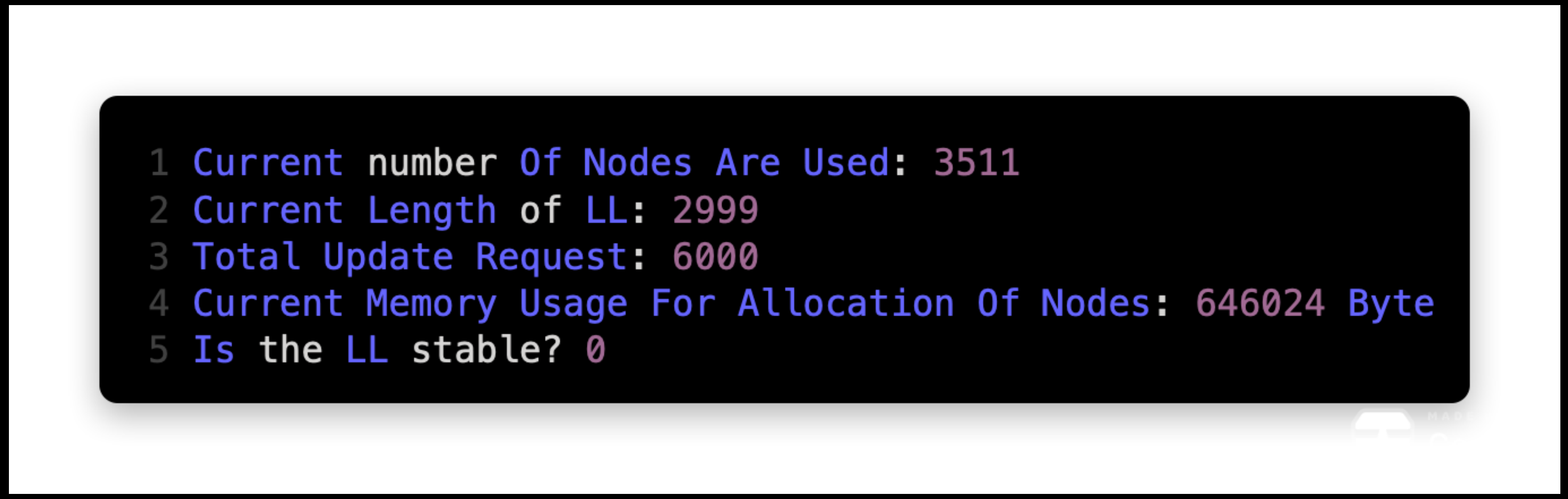
Supporting Code

max_size = 3000



```
2 warnings generated.
Target //tests:VALIDATION_TEST up-to-date:
  bazel-bin/tests/VALIDATION_TEST
INFO: Elapsed time: 2.383s, Critical Path: 2.16s
INFO: 3 processes: 1 internal, 2 darwin-sandbox.
INFO: Build completed successfully, 3 total actions
INFO: Running command line: external/bazel_tools/tools/test/test-setup.sh tests/VALIDATIONINFO: Build completed successfully, 3 total actions
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests:VALIDATION_TEST
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from VECT_AND_PDSLL_VALIDATION_TEST
[ RUN      ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL
Assertion failed: (0), function NodeAtIndex, file PPDS_LinkedList.h, line 385.
```

Failed To Validate The Result



```
Current number Of Nodes Are Used: 3511
Current Length of LL: 2999
Total Update Request: 6000
Current Memory Usage For Allocation Of Nodes: 646024 Byte
Is the LL stable? 0
```

We can see that the length of the LL is not 3000 anymore

2. Application in JAVA

The Java programming language is not particularly functional. Despite this, the core JDK package java.util.concurrent includes CopyOnWriteArrayList and CopyOnWriteArraySet which are **partial persistent** structures, implemented using copy-on-write techniques

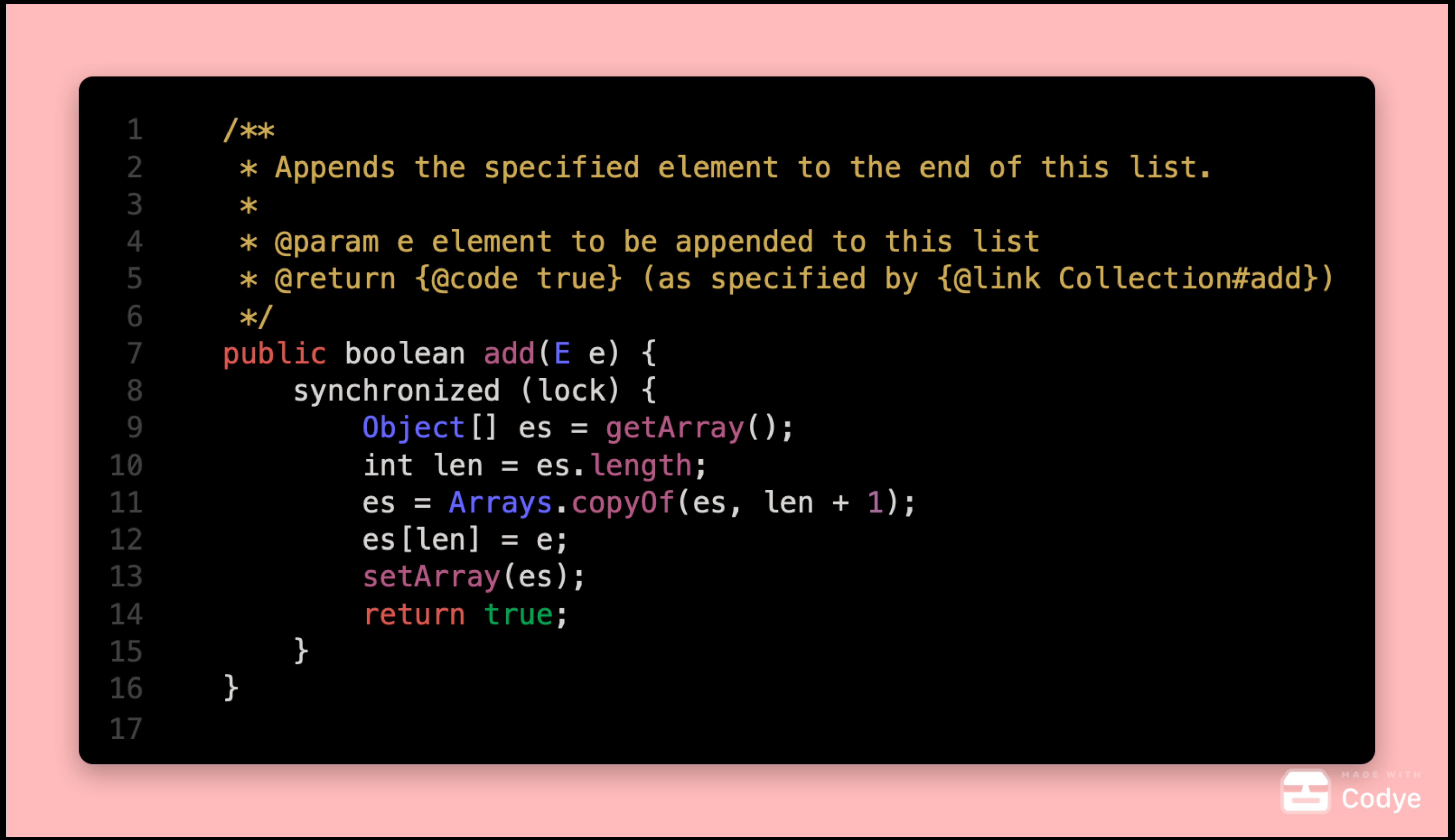
Class declaration

```
public class CopyOnWriteArrayList
   extends Object
implements List, RandomAccess, Cloneable, Serializable
```

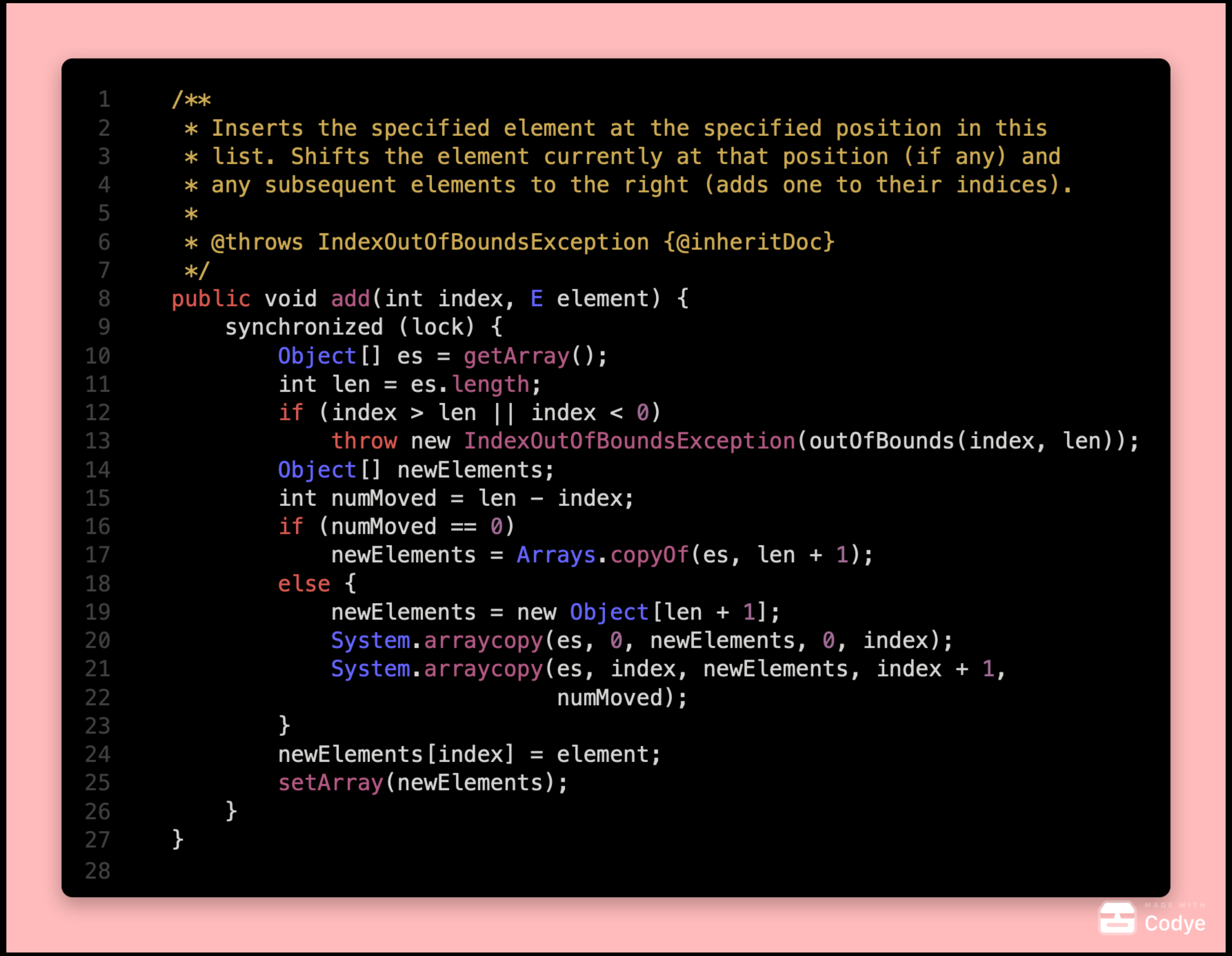
1. CopyOnWriteArrayList is a thread-safe variant of ArrayList where operations which can change the ArrayList (add, update, set methods) creates a clone of the underlying array.
2. CopyOnWriteArrayList is to be used in a Thread based environment where read operations are very frequent and update operations are rare.
3. Iterator of CopyOnWriteArrayList will never throw ConcurrentModificationException.
4. Any type of modification to CopyOnWriteArrayList will not reflect during iteration since the iterator was created.
5. List modification methods like remove, set and add are not supported in the iteration.

This method will throw UnsupportedOperationException.

null can be added to the list.



```java
/**
 * Appends the specified element to the end of this list.
 *
 * @param e element to be appended to this list
 * @return {@code true} (as specified by {@link Collection#add})
 */
public boolean add(E e) {
    synchronized (lock) {
        Object[] es = getArray();
        int len = es.length;
        es = Arrays.copyOf(es, len + 1);
        es[len] = e;
        setArray(es);
        return true;
    }
}
```

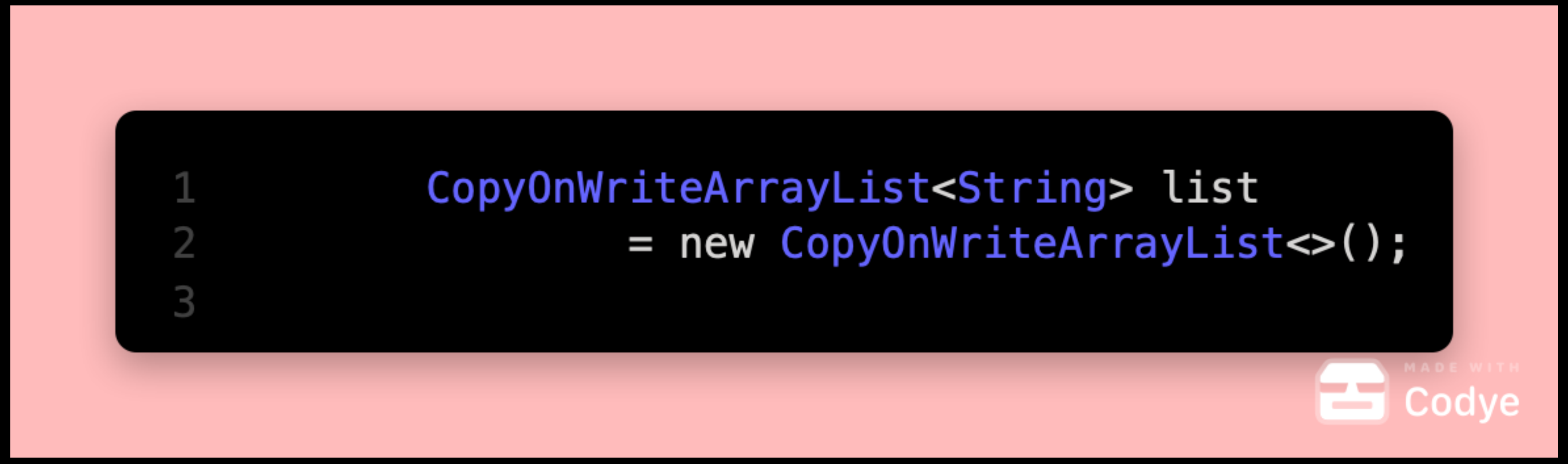


```java
/**
 * Inserts the specified element at the specified position in this
 * list. Shifts the element currently at that position (if any) and
 * any subsequent elements to the right (adds one to their indices).
 *
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public void add(int index, E element) {
    synchronized (lock) {
        Object[] es = getArray();
        int len = es.length;
        if (index > len || index < 0)
            throw new IndexOutOfBoundsException(outOfBounds(index, len));
        Object[] newElements;
        int numMoved = len - index;
        if (numMoved == 0)
            newElements = Arrays.copyOf(es, len + 1);
        else {
            newElements = new Object[len + 1];
            System.arraycopy(es, 0, newElements, 0, index);
            System.arraycopy(es, index, newElements, index + 1,
                             numMoved);
        }
        newElements[index] = element;
        setArray(newElements);
    }
}
```

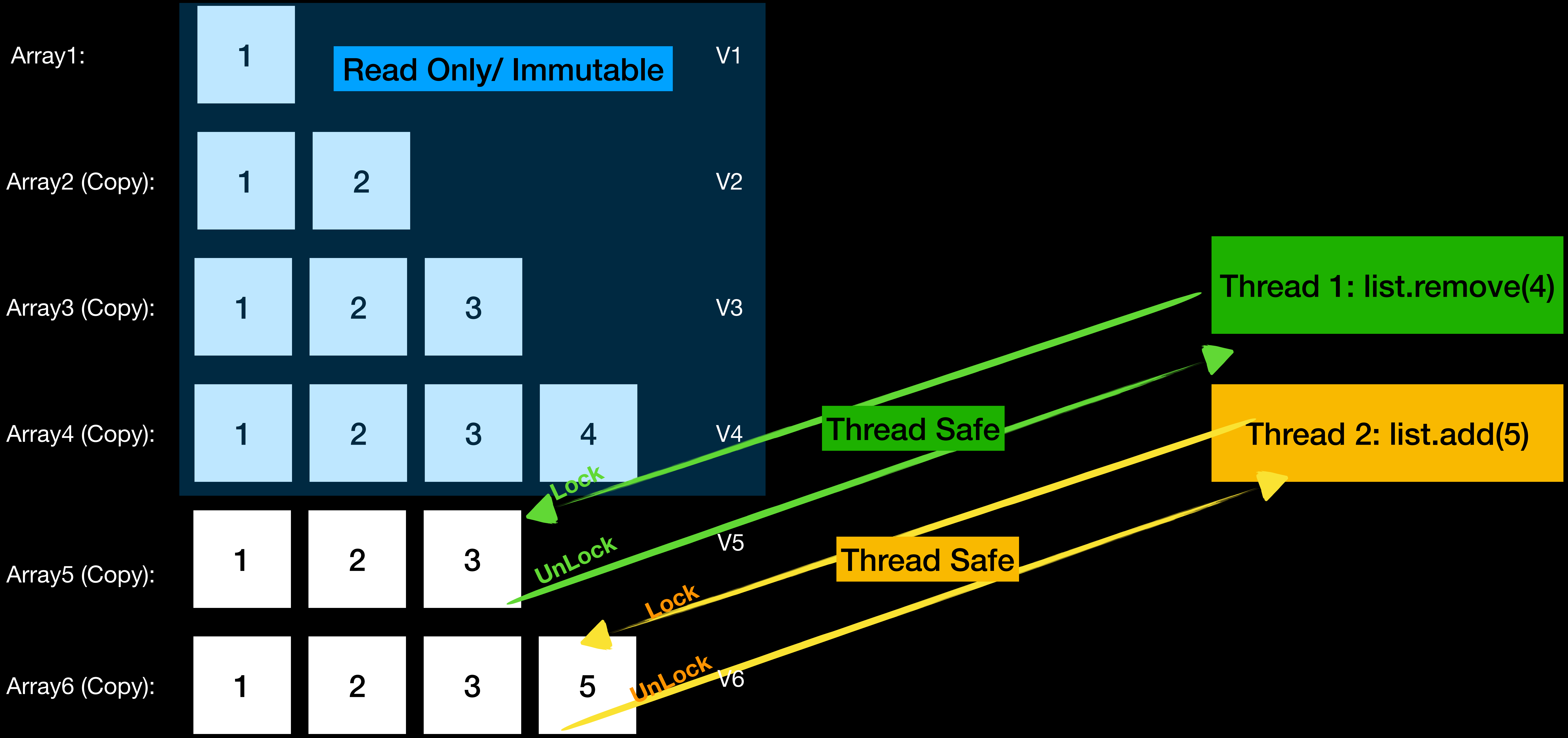
Code Taken From underlying source code of "CopyOnWriteArrayList" class

Note: We have tried to implement the same Functionality in C++.
[Code Attached]

Idea:



```
CopyOnWriteArrayList<String> list
        = new CopyOnWriteArrayList<>();
```



3. Application In Clojure

Clojure

Like many programming languages in the Lisp family, Clojure contains an implementation of a linked list, but unlike other dialects its implementation of a Linked List has enforced persistence instead of being persistent by convention.[19] Clojure also has efficient implementations of persistent vectors, maps, and sets based on persistent hash array mapped tries. These data structures implement the mandatory read-only parts of the Java collections framework.[20]

The designers of the Clojure language advocate the use of persistent data structures over mutable data structures because they have value semantics which gives the benefit of making them freely shareable between threads with cheap aliases, easy to fabricate, and language independent.[21]

These data structures form the basis of Clojure's support for parallel computing since they allow for easy retries of operations to sidestep data races and atomic compare and swap semantics.[22]

[Slides in Clojure are attached Separately]

Extract Taken from wikipedia