

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

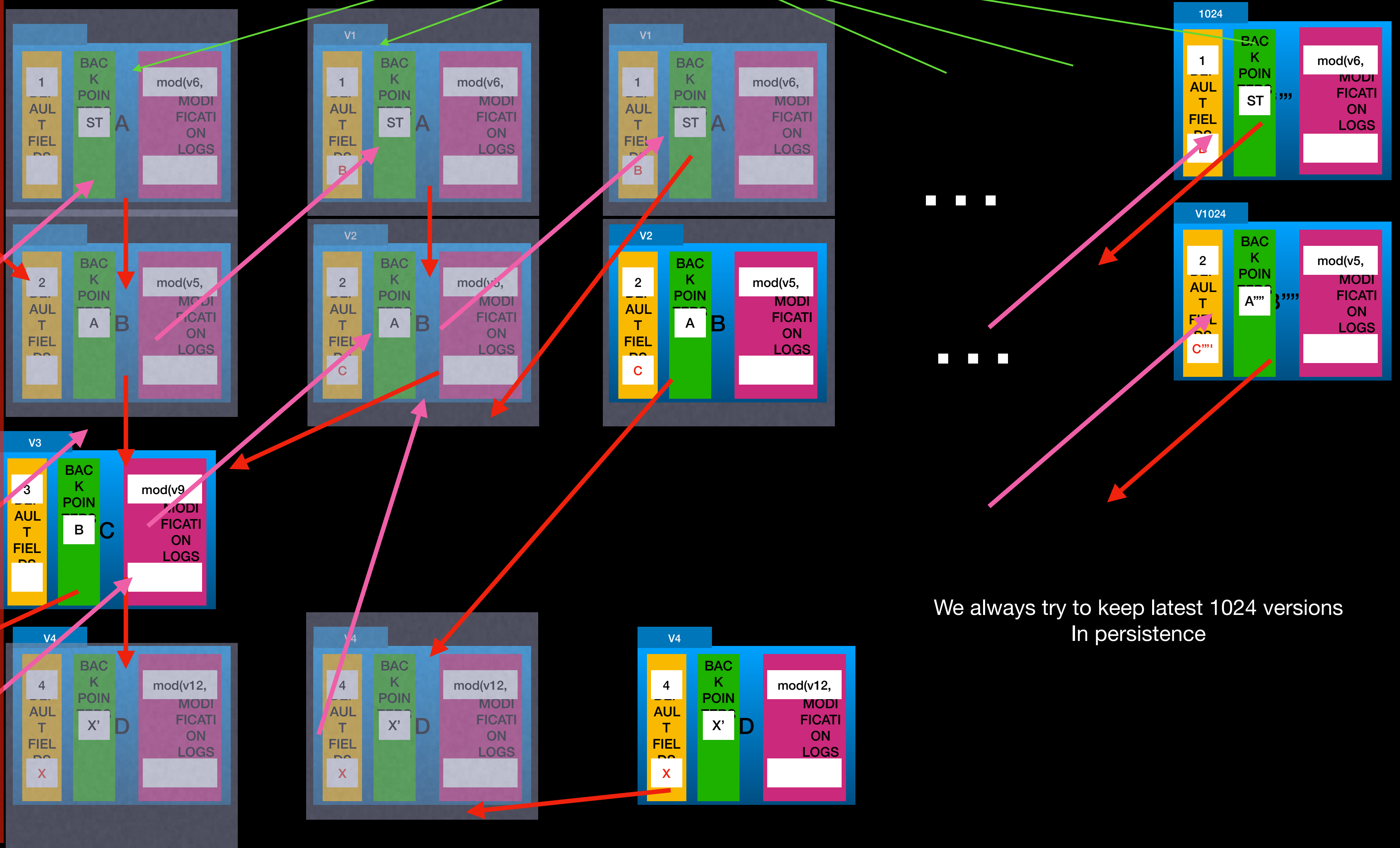
Because, the PPDS are very memory consuming.

START MODULE

V0 V10 V30 ... 1024

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

```
1 #include "gtest/gtest.h"
2 #include <iostream>
3 #include <vector>
4 #include "src/lib/headers_preprocessors.h"
5 #include "src/lib/PPDS_LinkedList.h"
6 #include "src/lib/test_with_Vector_and_PDSLL.h"
7
8 using namespace std;
9
10 const int max_mod = 3;
11 const int max_size = 200;
12
13 TEST(VECT_AND_PDSLL_VALIDATION_TEST, COMPARE_VECTOR_VS_PDSLL) {
14
15     vector<int>vect(max_size,SENTINEL);
16     PPDS_LINKED_LIST<int>list(max_mod);
17     VECT_AND_PDSLL vp_test(max_mod,max_size);
18     vector<int>actualResult,expectedResult;
19
20
21     vp_test.randomized_Insert(list,vect);
22     expectedResult = vp_test.trimVect(vect);
23     actualResult = list.iterate_and_print_F1_at_ver_Vect(list.getCurTime());
24
25     EXPECT_EQ(expectedResult, actualResult); // random insert
26     expectedResult.clear(); actualResult.clear();
27
28
29     vp_test.randomized_Update(list,vect);
30     expectedResult = vp_test.trimVect(vect);
31     actualResult = list.iterate_and_print_F1_at_ver_Vect(list.getCurTime());
32
33     EXPECT_EQ(expectedResult, actualResult); // random update
34
35 }
36
```

max_size = 200

```
1 Target //tests:VALIDATION_TEST up-to-date:
2   bazel-bin/tests/VALIDATION_TEST
3 INFO: Elapsed time: 2.258s, Critical Path: 2.05s
4 INFO: 3 processes: 1 internal, 2 darwin-sandbox.
5 INFO: Build completed successfully, 3 total actions
6 INFO: Running command line: external/bazel_tools/tools/test/test-setup.sh tests/VALIDATIONINFO: Build completed successfully, 3 total actions
7 exec ${PAGER:-/usr/bin/less} "$@" || exit 1
8 Executing tests from //tests:VALIDATION_TEST
9 -----
10 Running main() from gmock_main.cc
11 [=====] Running 1 test from 1 test suite.
12 [-----] Global test environment set-up.
13 [-----] 1 test from VECT_AND_PDSLL_VALIDATION_TEST
14 [ RUN    ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL
15 [      OK ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL (208 ms)
16 [-----] 1 test from VECT_AND_PDSLL_VALIDATION_TEST (208 ms total)
17
18 [-----] Global test environment tear-down
19 [=====] 1 test from 1 test suite ran. (208 ms total)
20 [ PASSED ] 1 test.
```

max_size = 2000

```
1 Target //tests:VALIDATION_TEST up-to-date:
2   bazel-bin/tests/VALIDATION_TEST
3 INFO: Elapsed time: 1.622s, Critical Path: 1.49s
4 INFO: 3 processes: 1 internal, 2 darwin-sandbox.
5 INFO: Build completed successfully, 3 total actions
6 INFO: Running command line: external/bazel_tools/tools/test/test-setup.sh tests/VALIDATIONINFO: Build completed successfully, 3 total actions
7 exec ${PAGER:-/usr/bin/less} "$@" || exit 1
8 Executing tests from //tests:VALIDATION_TEST
9 -----
10 Running main() from gmock_main.cc
11 [=====] Running 1 test from 1 test suite.
12 [-----] Global test environment set-up.
13 [-----] 1 test from VECT_AND_PDSLL_VALIDATION_TEST
14 [ RUN    ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL
15 [      OK ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL (5 ms)
16 [-----] 1 test from VECT_AND_PDSLL_VALIDATION_TEST (5 ms total)
17
18 [-----] Global test environment tear-down
19 [=====] 1 test from 1 test suite ran. (5 ms total)
20 [ PASSED ] 1 test.
```

These Results Passed

Supporting Code

max_size = 3000

```
1 2 warnings generated.
2 Target //tests:VALIDATION_TEST up-to-date:
3   bazel-bin/tests/VALIDATION_TEST
4 INFO: Elapsed time: 2.383s, Critical Path: 2.16s
5 INFO: 3 processes: 1 internal, 2 darwin-sandbox.
6 INFO: Build completed successfully, 3 total actions
7 INFO: Running command line: external/bazel_tools/tools/test/test-setup.sh tests/VALIDATIONINFO: Build completed successfully, 3 total actions
8 exec ${PAGER:-/usr/bin/less} "$0" || exit 1
9 Executing tests from //tests:VALIDATION_TEST
10 -----
11 Running main() from gmock_main.cc
12 [=====] Running 1 test from 1 test suite.
13 [-----] Global test environment set-up.
14 [-----] 1 test from VECT_AND_PDSLL_VALIDATION_TEST
15 [ RUN      ] VECT_AND_PDSLL_VALIDATION_TEST.COMPARE_VECTOR_VS_PDSLL
16 Assertion failed: (0), function NodeAtIndex, file PPDS_LinkedList.h, line 385.
```

Failed To Validate The Result

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

*In-depth analysis of this failure ,
we shall do after exam*

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.

```

1  /**
2   * Appends the specified element to the end of this list.
3   *
4   * @param e element to be appended to this list
5   * @return {@code true} (as specified by {@link Collection#add})
6   */
7  public boolean add(E e) {
8      synchronized (lock) {
9          Object[] es = getArray();
10         int len = es.length;
11         es = Arrays.copyOf(es, len + 1);
12         es[len] = e;
13         setArray(es);
14         return true;
15     }
16 }
17

```



```

1  /**
2   * Inserts the specified element at the specified position in this
3   * list. Shifts the element currently at that position (if any) and
4   * any subsequent elements to the right (adds one to their indices).
5   *
6   * @throws IndexOutOfBoundsException {@inheritDoc}
7   */
8  public void add(int index, E element) {
9      synchronized (lock) {
10         Object[] es = getArray();
11         int len = es.length;
12         if (index > len || index < 0)
13             throw new IndexOutOfBoundsException(outOfBounds(index, len));
14         Object[] newElements;
15         int numMoved = len - index;
16         if (numMoved == 0)
17             newElements = Arrays.copyOf(es, len + 1);
18         else {
19             newElements = new Object[len + 1];
20             System.arraycopy(es, 0, newElements, 0, index);
21             System.arraycopy(es, index, newElements, index + 1,
22                             numMoved);
23         }
24         newElements[index] = element;
25         setArray(newElements);
26     }
27 }
28

```



**Code Taken From underlying source code of
“CopyOnWriteArrayList” class**

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

Idea:

```
1 CopyOnWriteArrayList<String> list
2   = new CopyOnWriteArrayList<>();
3
```



Array1:



Array2 (Copy):



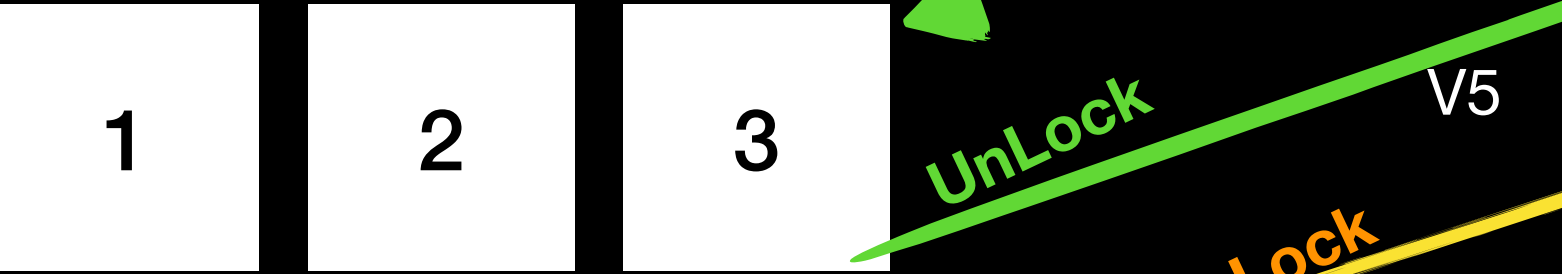
Array3 (Copy):



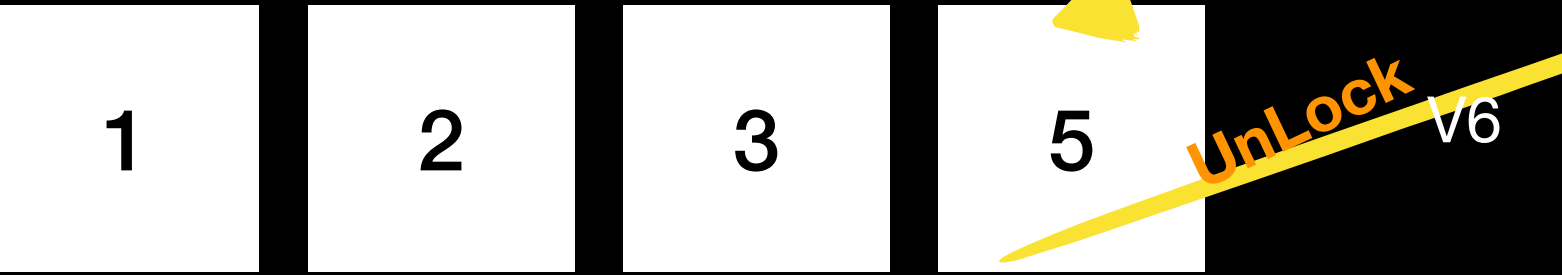
Array4 (Copy):



Array5 (Copy):



Array6 (Copy):



Thread 1: list.remove(4)

Thread 2: list.add(5)

Thread Safe

Thread Safe

Lock

UnLock

Lock

UnLock

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]