

Partial Persistent Data-structure

Pointer Machine Model

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

Ephemeral Linked List

Operations:

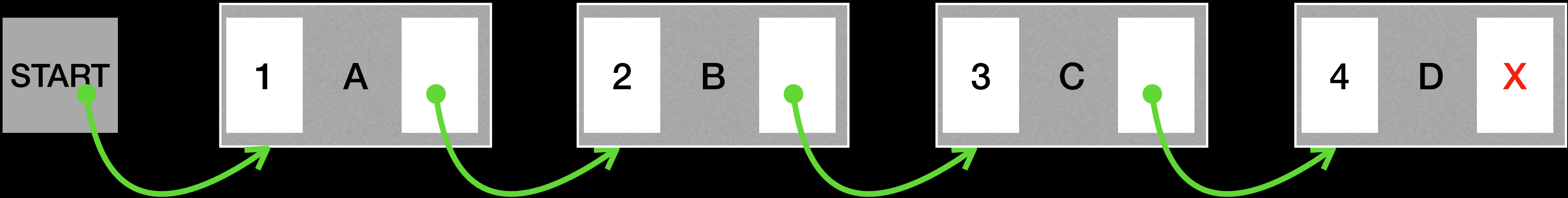
- `start = init()` : To initiate linked list and “start” pointer holds the starting position
- `add(x, y)` : Add new node x after y
- `remove(x)` : Remove node x
- `iterate_over_LL()` : Iterate over the whole linked list
- `update(f_i,x,val)` : Update the i-th field in node x to new value ‘val’

Ephemeral Linkedlist

Node structure



LINKED LIST

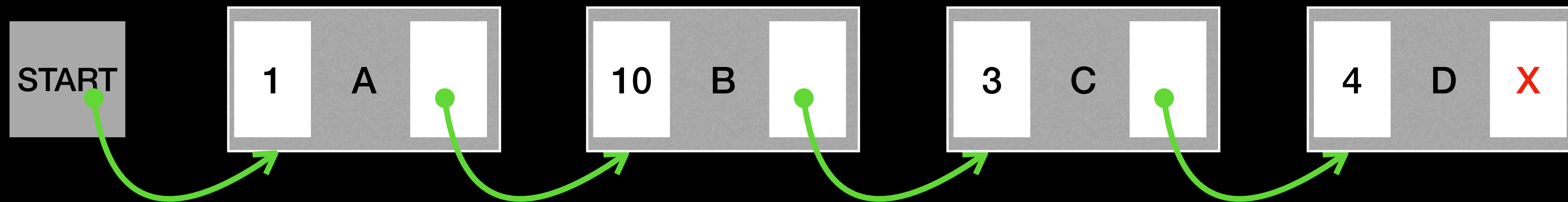


HERE IN-DEGREE OF A NODE IS ≤ 1 AND OUT-DEGREE OF A NODE IS ≤ 1

SOME OPERATIONS

update(f1,B,10)

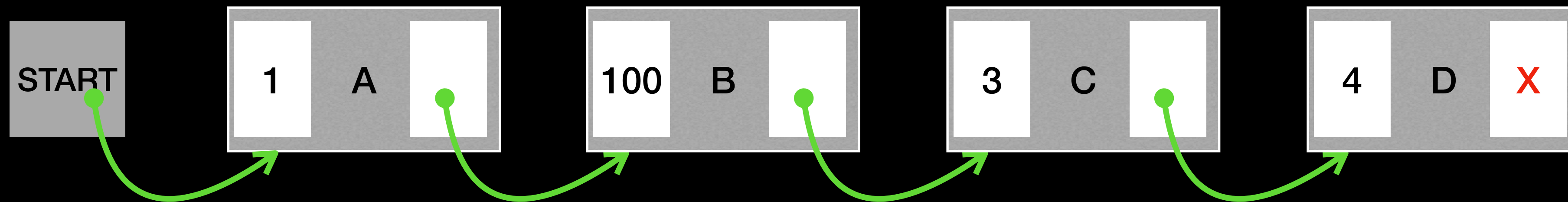
LINKED LIST



SOME OPERATIONS

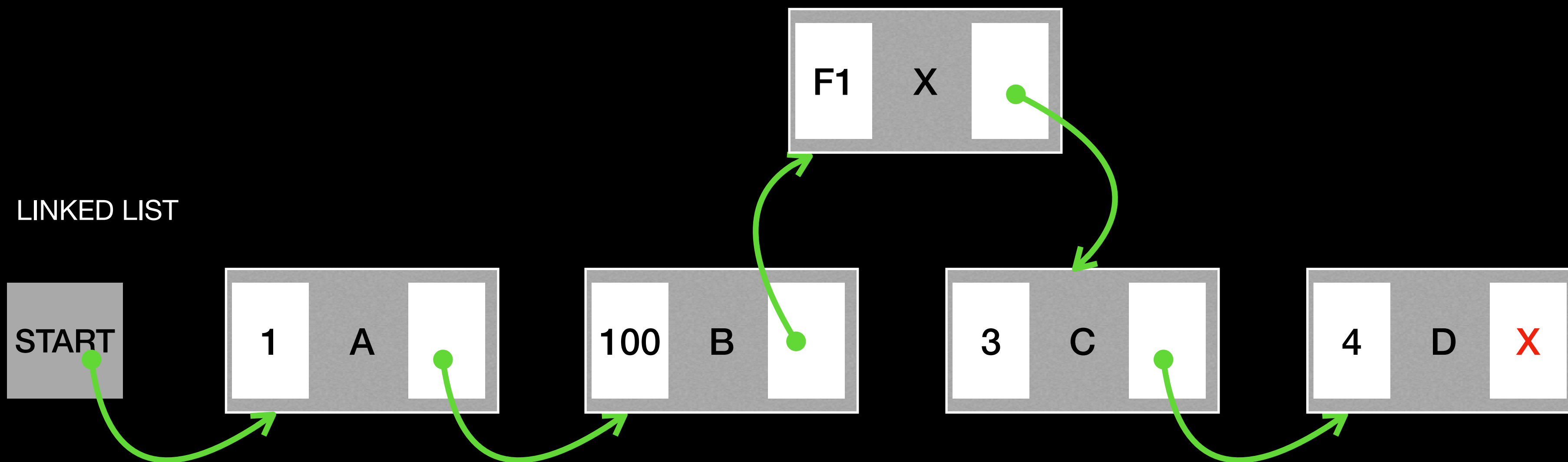
`update(f1,B,100)`

LINKED LIST



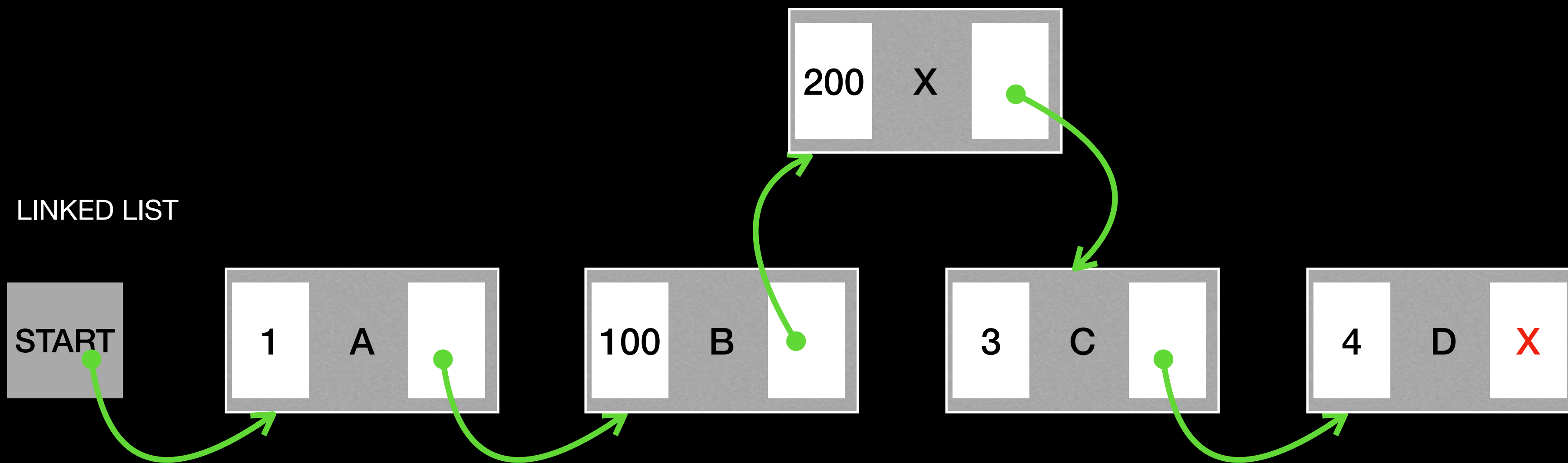
SOME OPERATIONS

add(X,B)



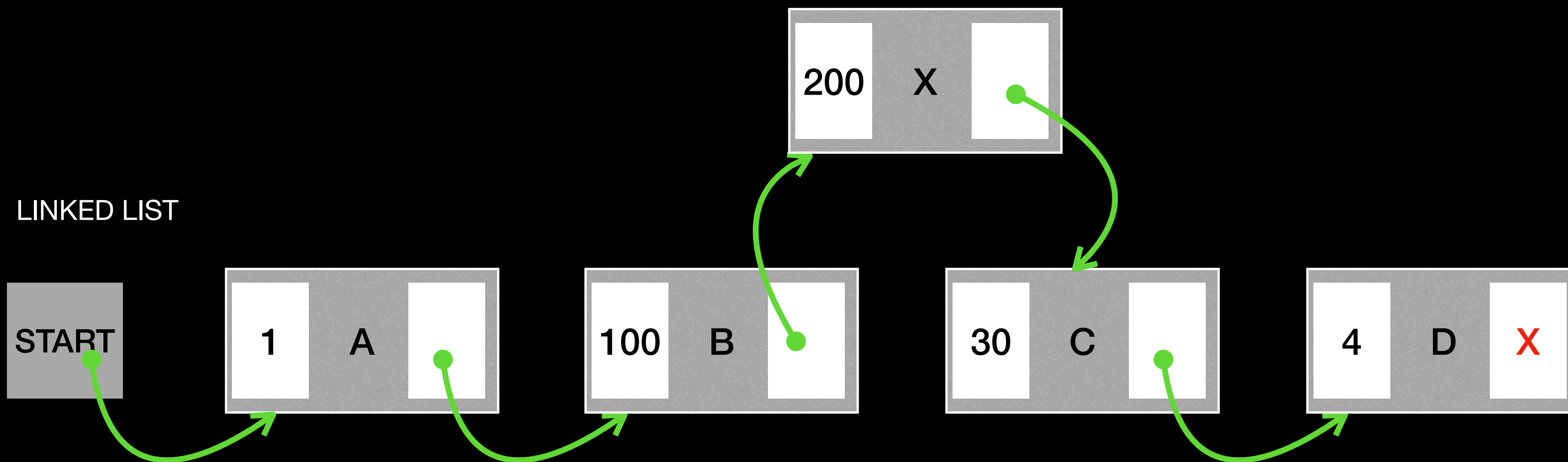
SOME OPERATIONS

update(f1,X,200)



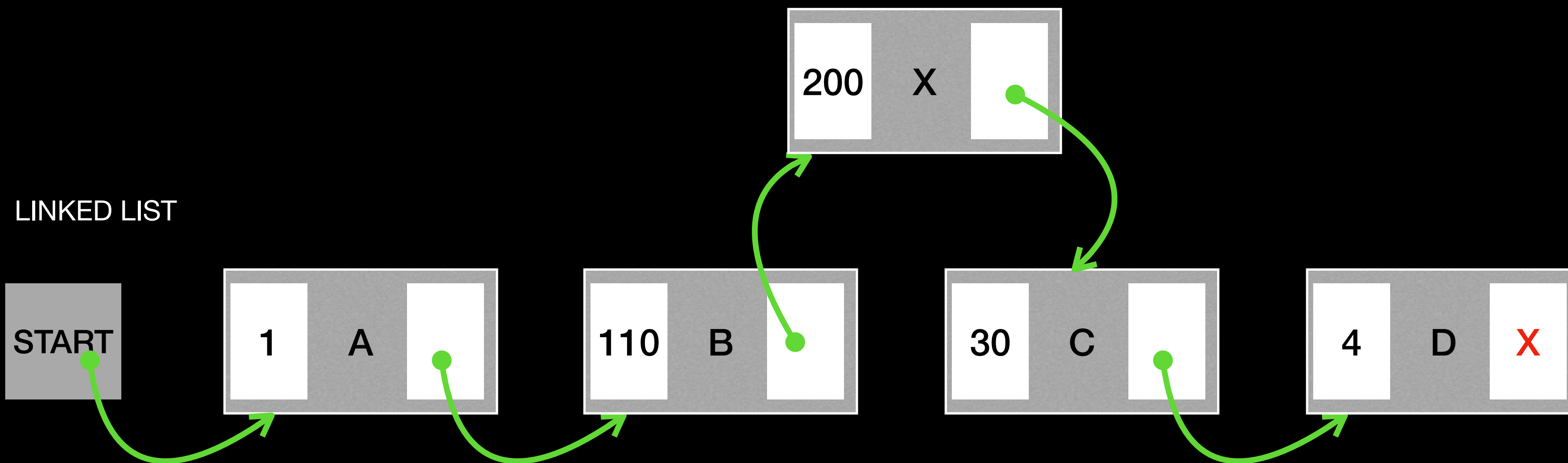
SOME OPERATIONS

update(f1,C,30)



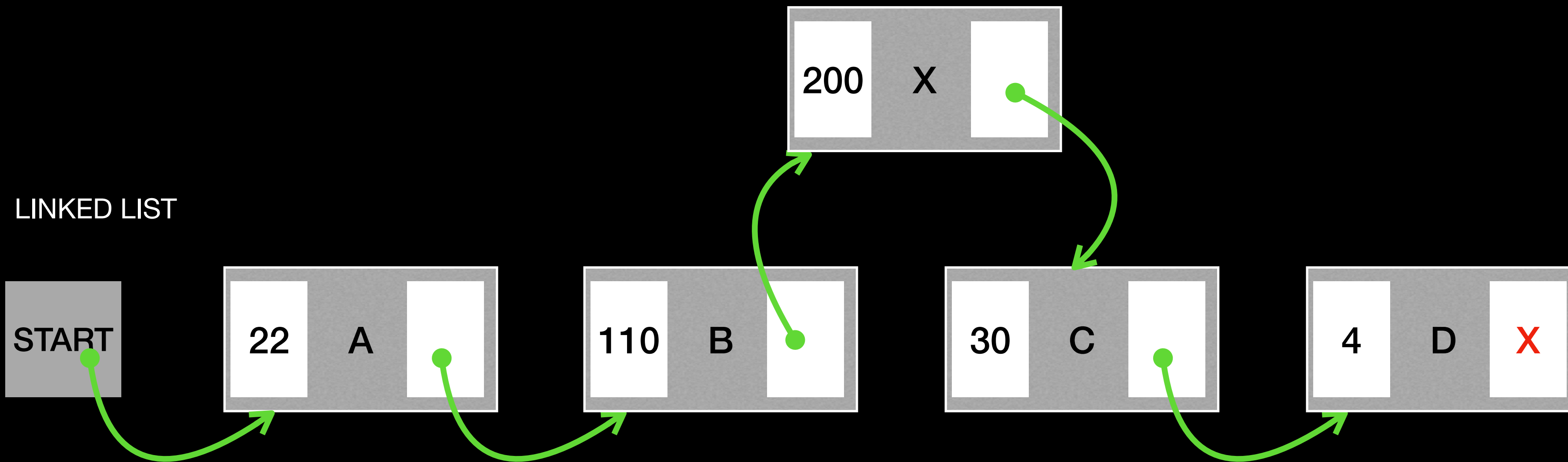
SOME OPERATIONS

update(f1,B,110)



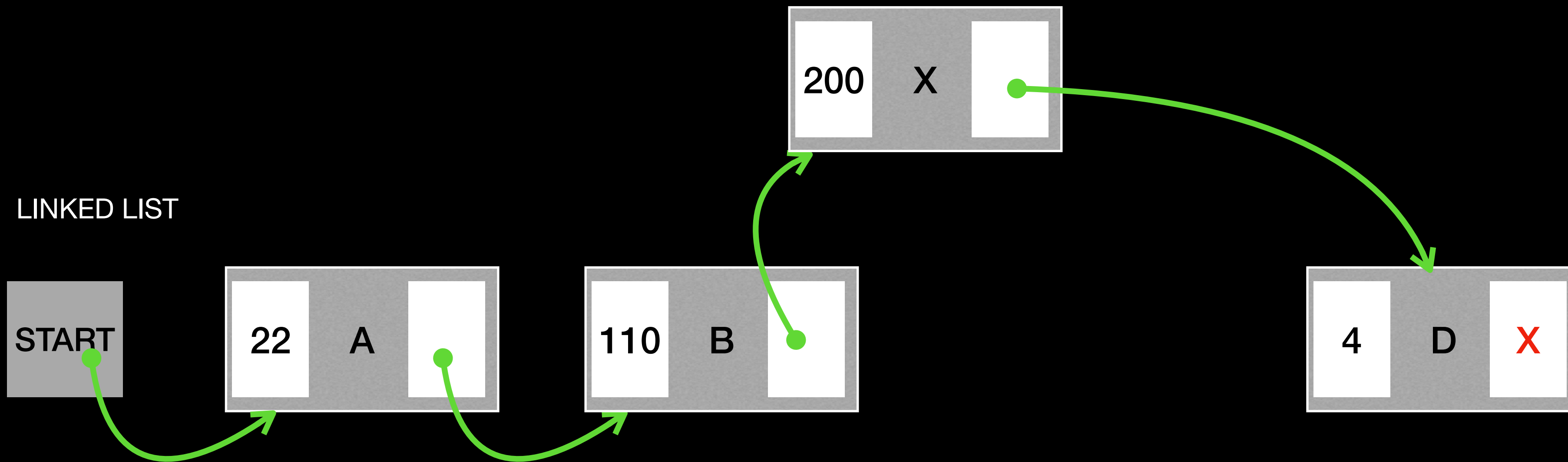
SOME OPERATIONS

update(f1,A,22)



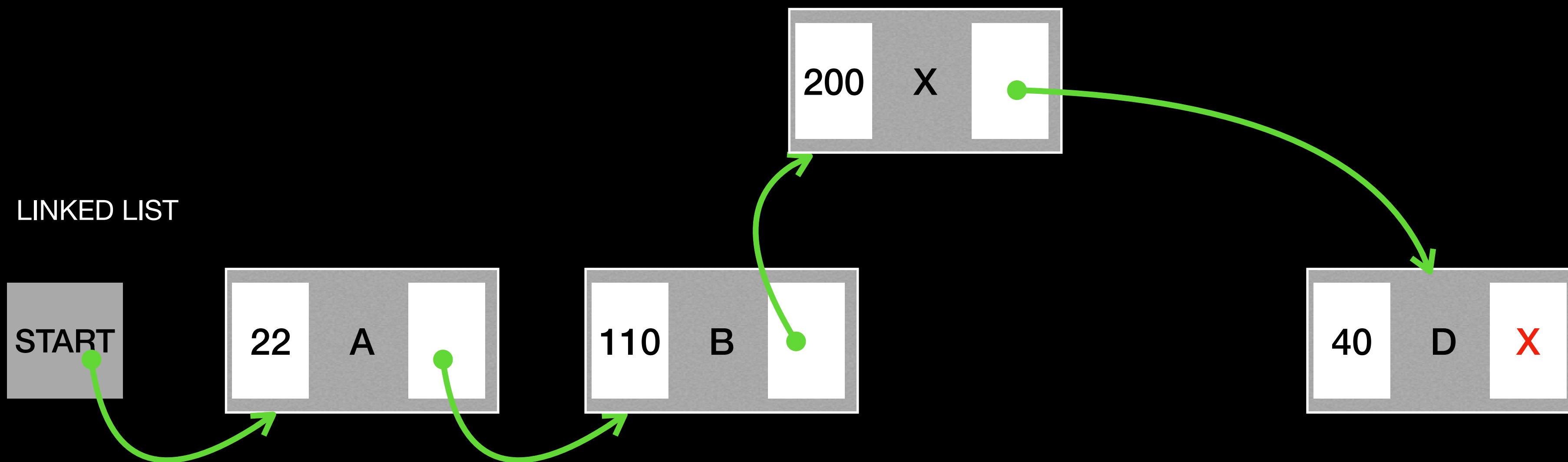
SOME OPERATIONS

remove(C)



SOME OPERATIONS

update(f1,D,40)



Partial Persistent Linked List

Operations:

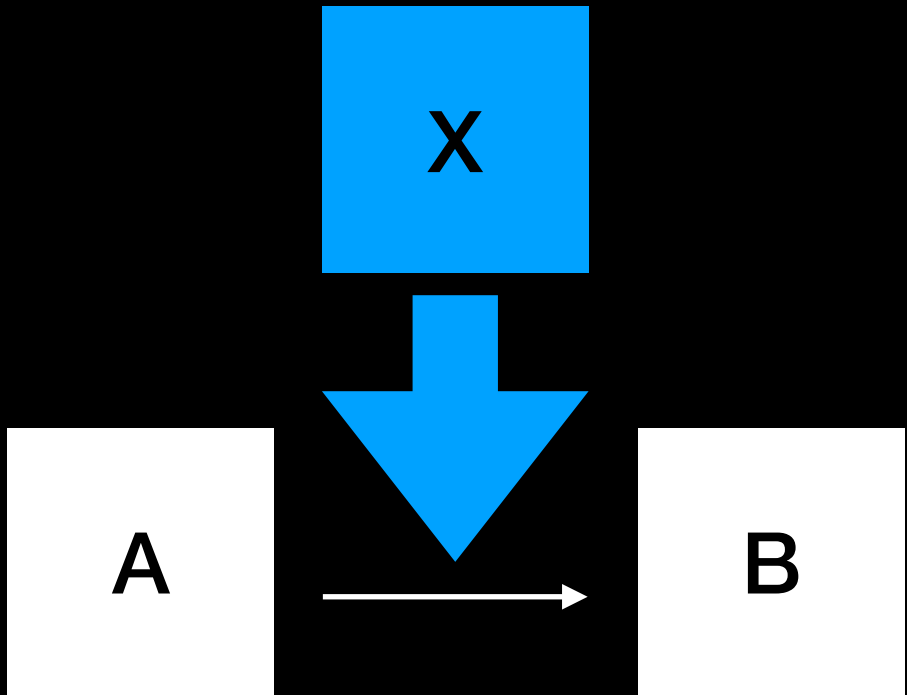
- `start = init()` : To initiate linked list and “start” pointer holds the starting position in `v0`
- `create_node(x, a)`: Allocate a new node `x` with `f1 = a` and `f2 = NULL`, and set its default version to current version
- `add(x, y, a)` : Add new node `x` with value = `a` after `y` and update the version
- `remove(x)` : Remove node `x` and update the version
- `iterate_over_LL(v)` : Iterate over the whole linked list in version `v`
- `update(f_i, x, val)` : Update the `i`-th field in node `x` to new value ‘`val`’ and update the version

Interesting thing!

add(x,y) and remove(x) are not Elementary operations

add(X,C,123) consists of

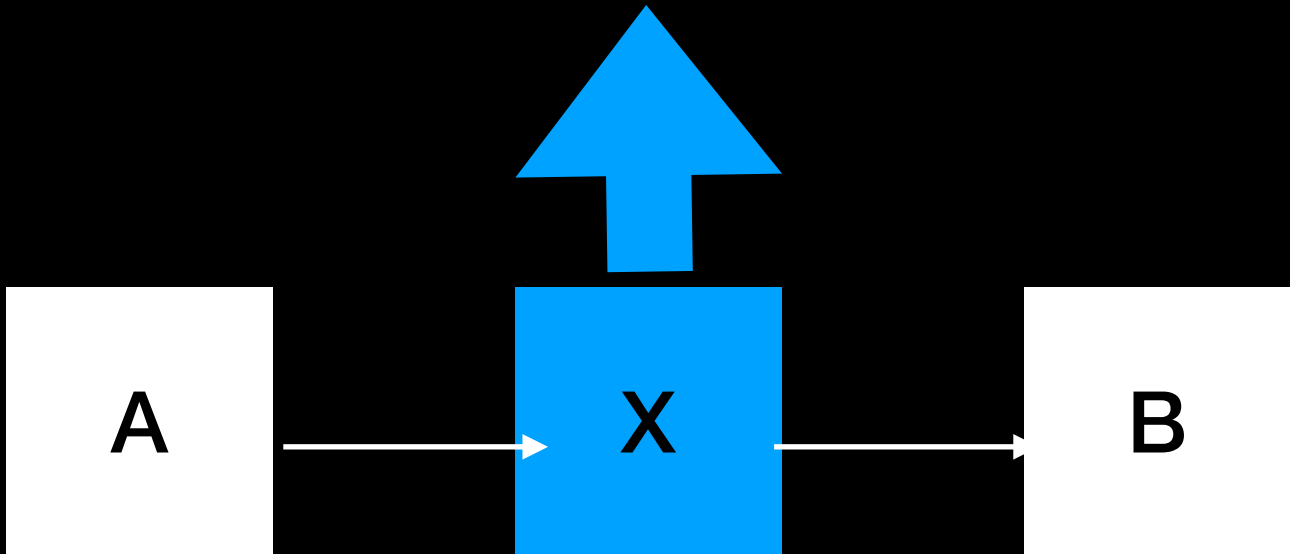
Create node X with value 123
Modify f2 of A to X : update(f2, A, X)
Modify f2 of X to C : update(f2, X, C)
Set BP of X to A: X.bp = A
Set BP of B to X: B.bp = X
Set f1 of X(optional)



remove(x) consists of

Modify F2 of Parent C (i.e., X) -> F2 of C (successor of C)

update(f2, A, B)
Set BP of B to A: B.bp = A
If all shared reference of X is removed Then free up the memory associated with X



Elementary Operations:

- `start = init()`

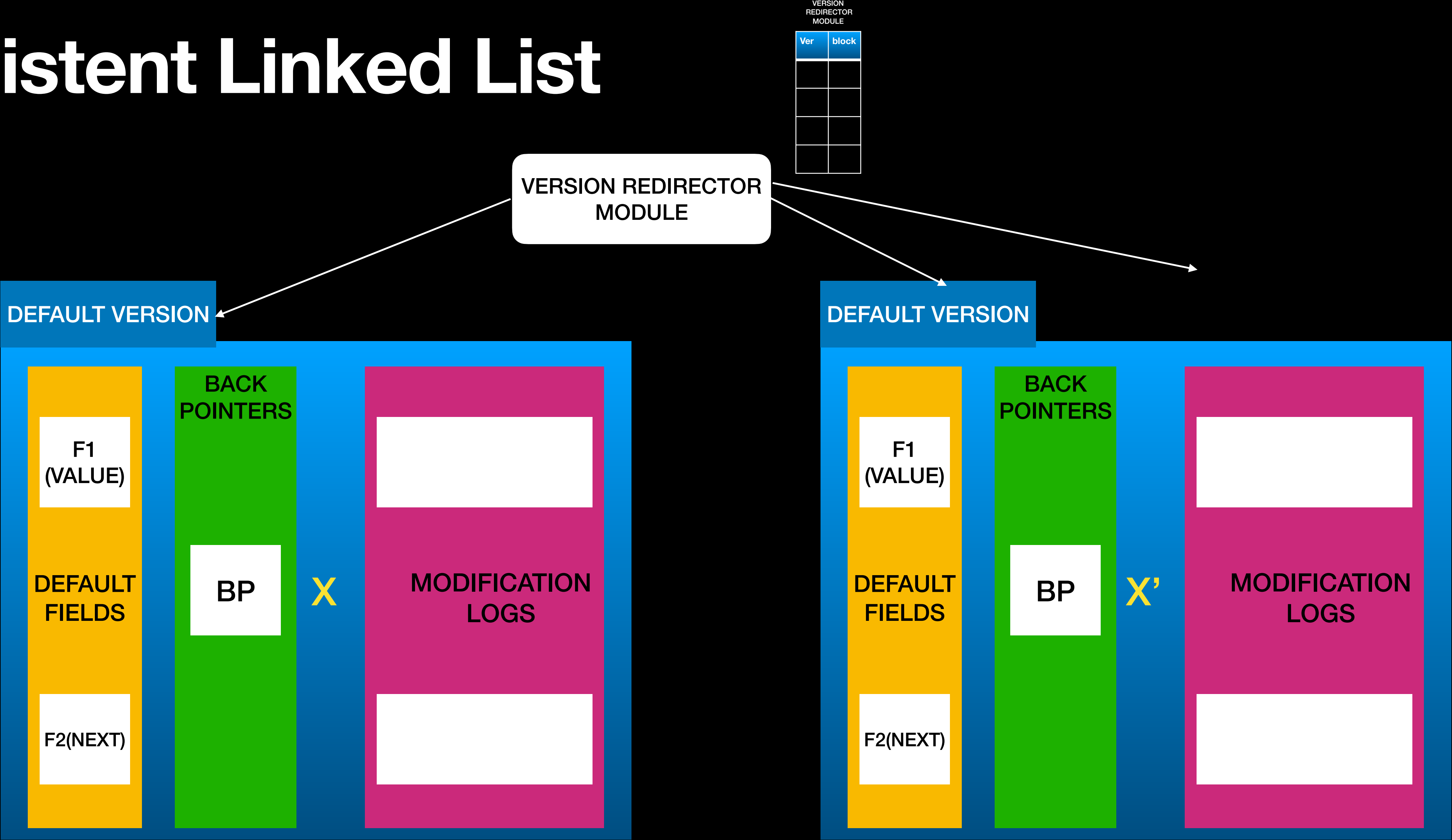
- `create_node(x, a)`

- `iterate_over_LL(v)`

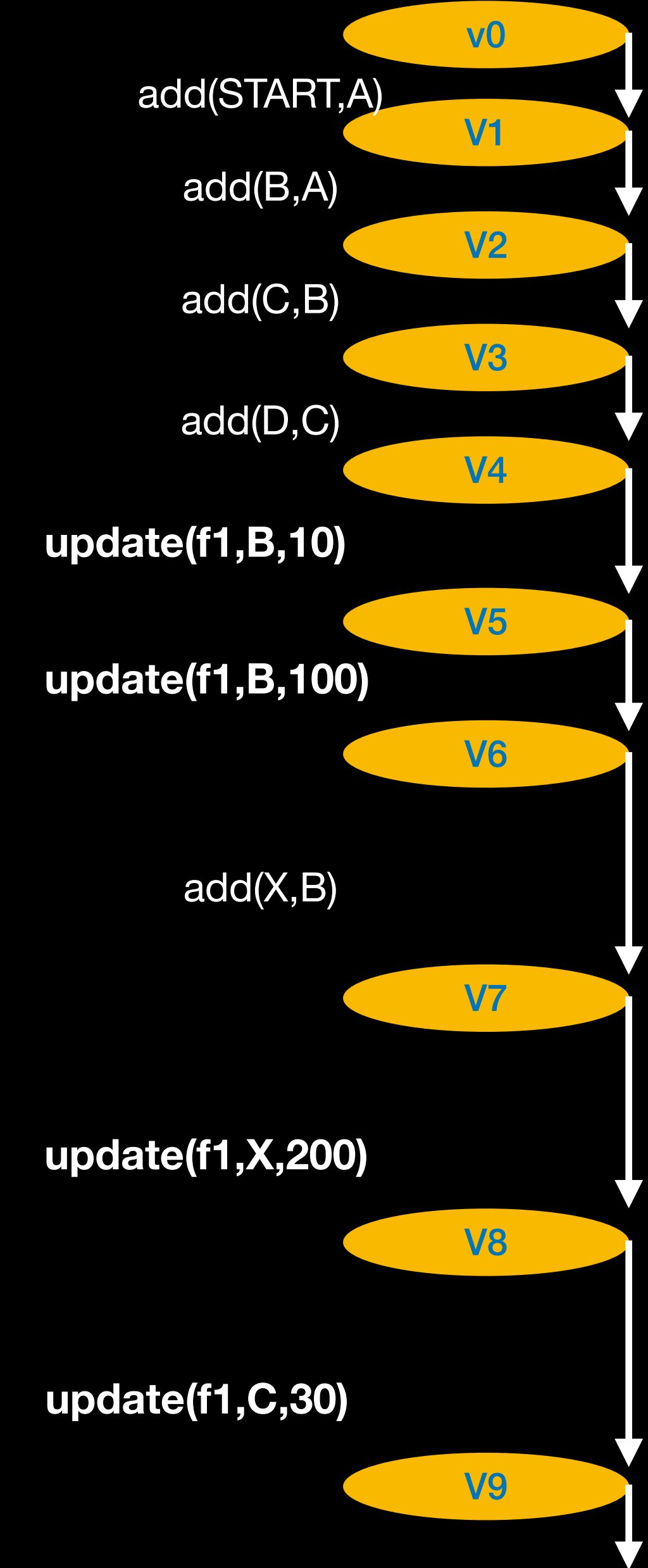
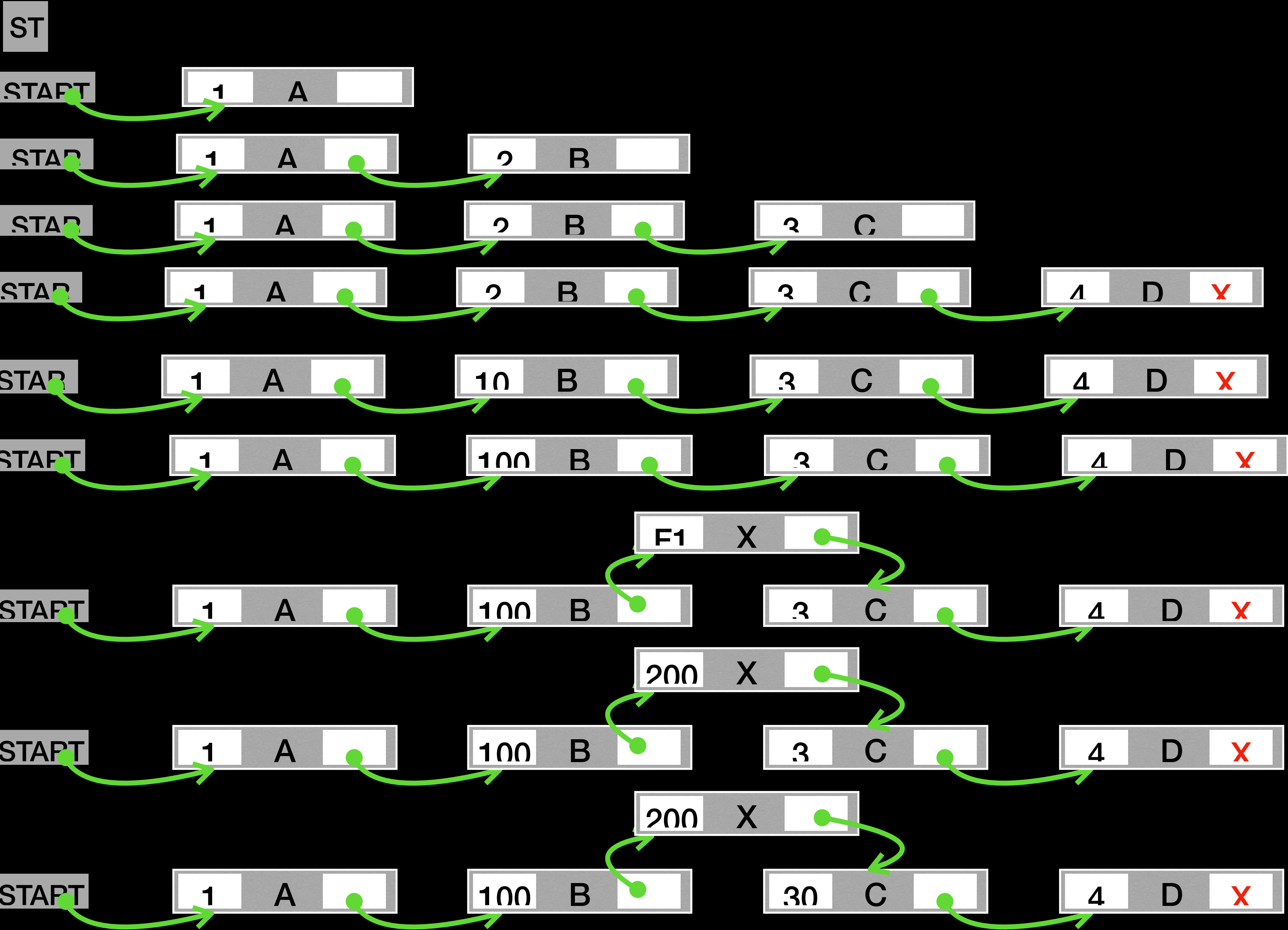
- `update(f_i,x,val)`

Persistent Linked List

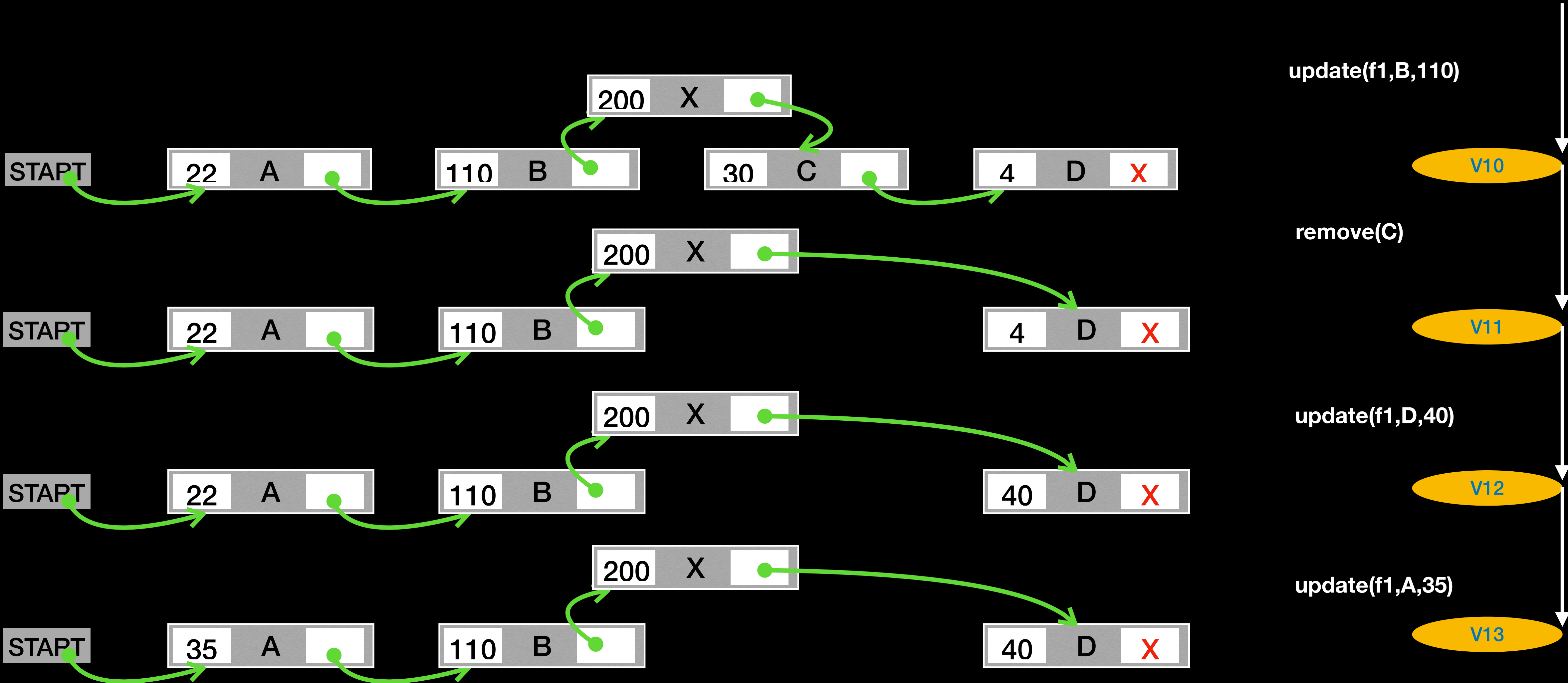
Node



Ephemeral List and Corresponding Versions



Ephemeral List and Corresponding Versions (contd.)



IMPLEMENTATION IN PERSISTENT POINTER MACHINE MODEL

START MODULE

v0

Current Version: v1

START MODULE

V0

add(A,START,1)

V1

1

DEFAULT
FIELDS

X

BACK
POINTER
S

X

A

MODIFICATION
LOGS

VERSION
REDIRECTOR
MODULE

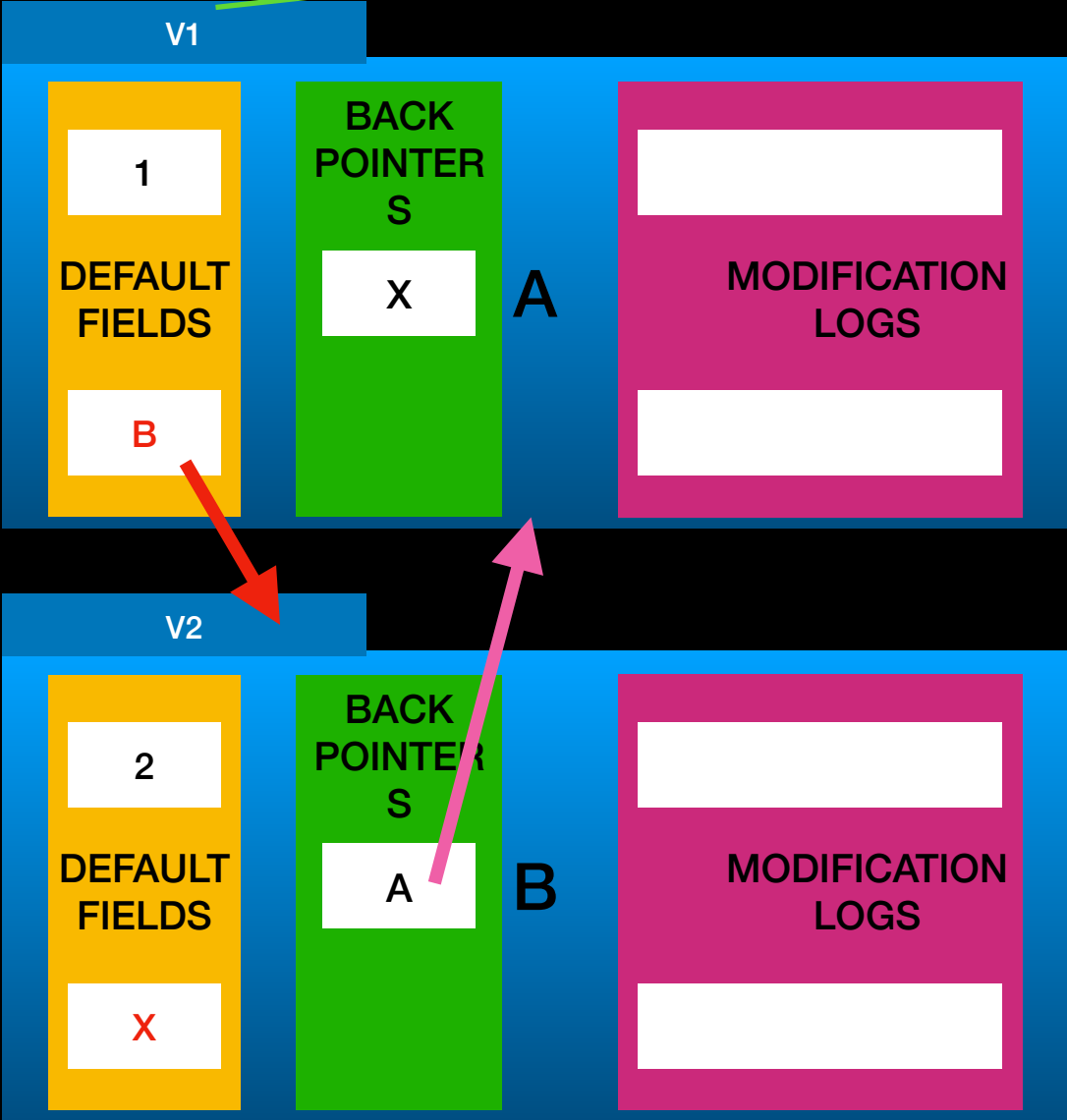
Ver	block
1	A

Current Version: v2

START MODULE

V0

add(B,A,2)



VERSION
REDIRECTOR
MODULE

Ver	block
1	A

VERSION
REDIRECTOR
MODULE

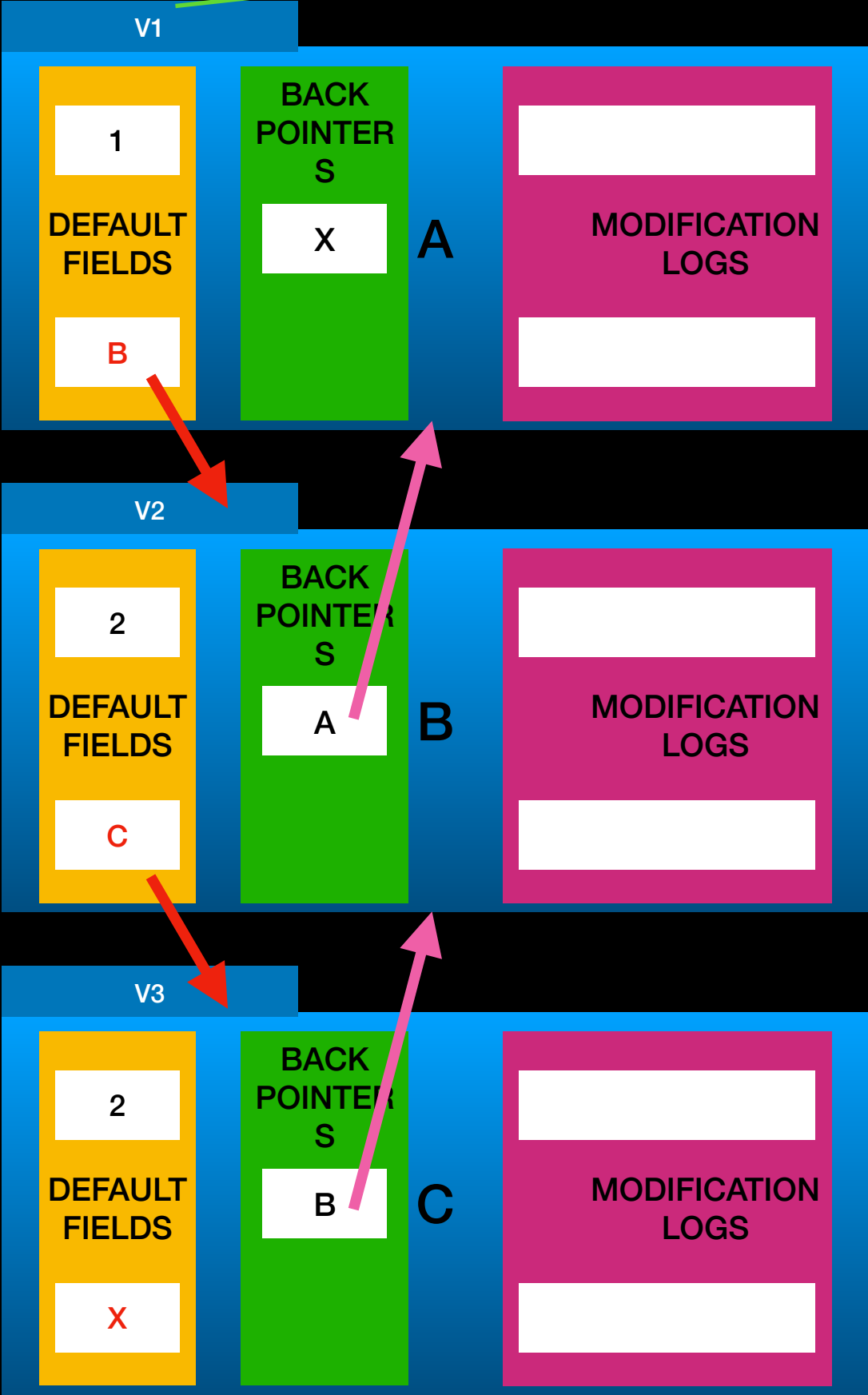
Ver	block
2	B

Current Version: v3

START MODULE

V0

add(C,B,3)



VERSION
REDIRECTOR
MODULE

Ver	block
1	A

VERSION
REDIRECTOR
MODULE

Ver	block
2	B

VERSION
REDIRECTOR
MODULE

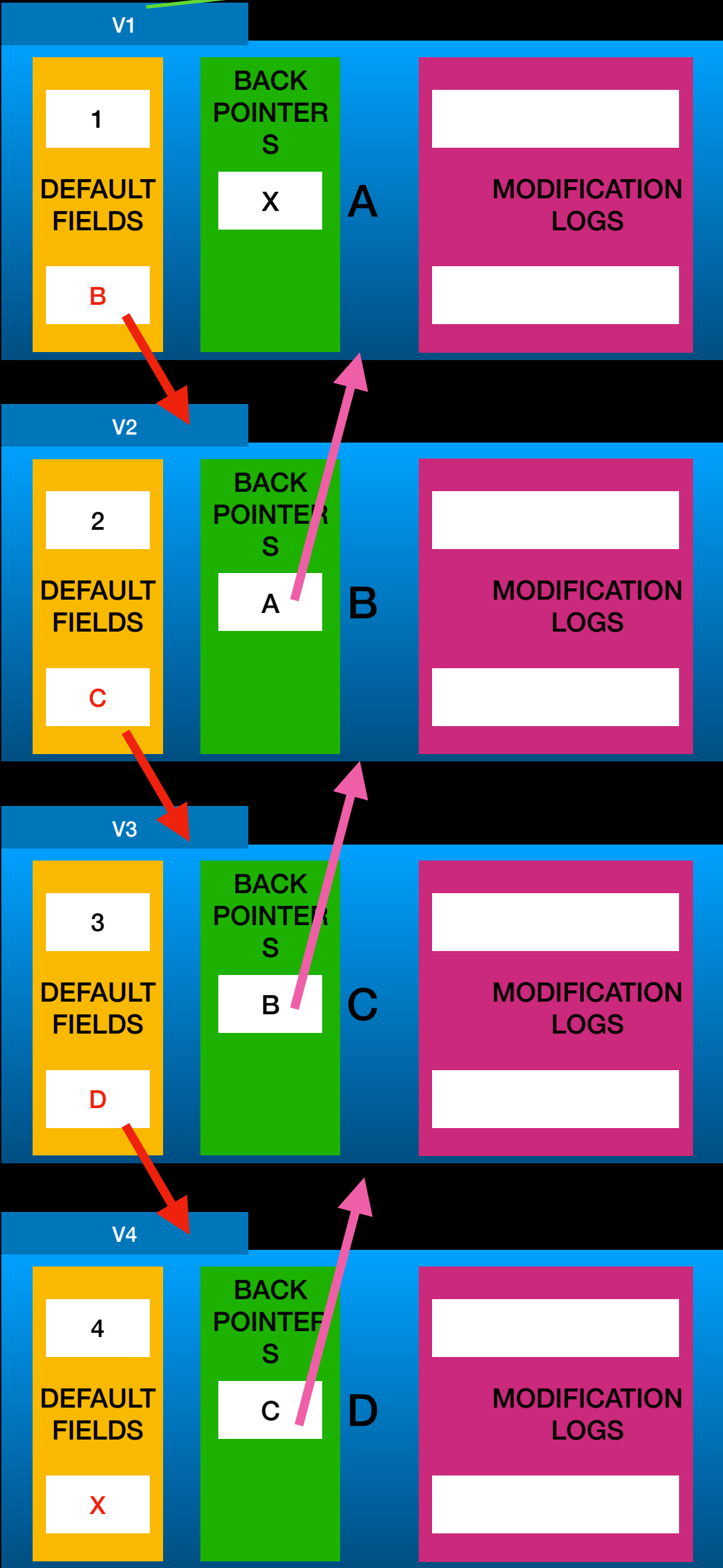
Ver	block
3	C

Current Version: v4

START MODULE

V0

add(D,C,4)



VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B

VERSION REDIRECTOR MODULE

Ver	block
3	C

VERSION REDIRECTOR MODULE

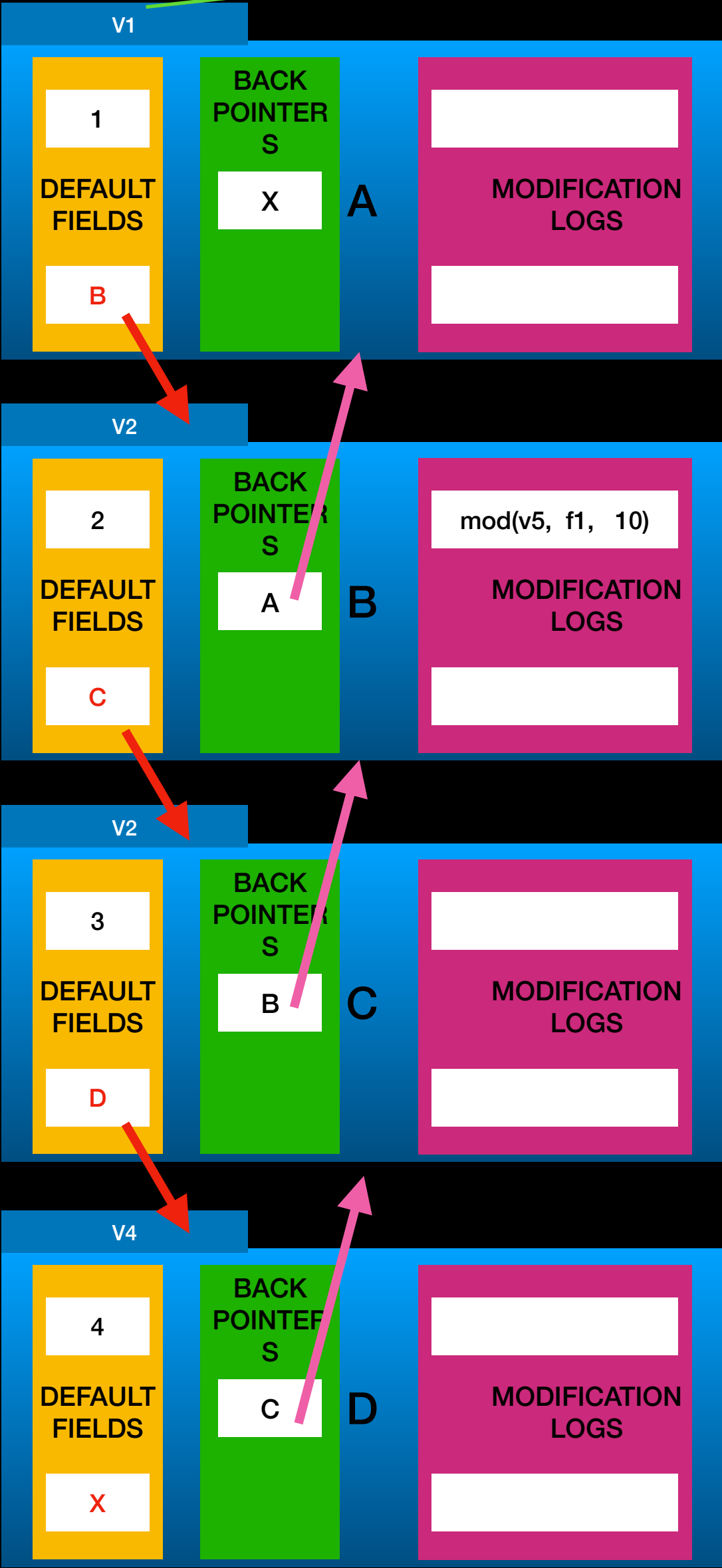
Ver	block

Current Version: v5

START MODULE

V0

update(f1,B,10)



VERSION
REDIRECTOR
MODULE

Ver	block
1	A

VERSION
REDIRECTOR
MODULE

Ver	block
2	B

VERSION
REDIRECTOR
MODULE

Ver	block
3	C

VERSION
REDIRECTOR
MODULE

Ver	block
4	D

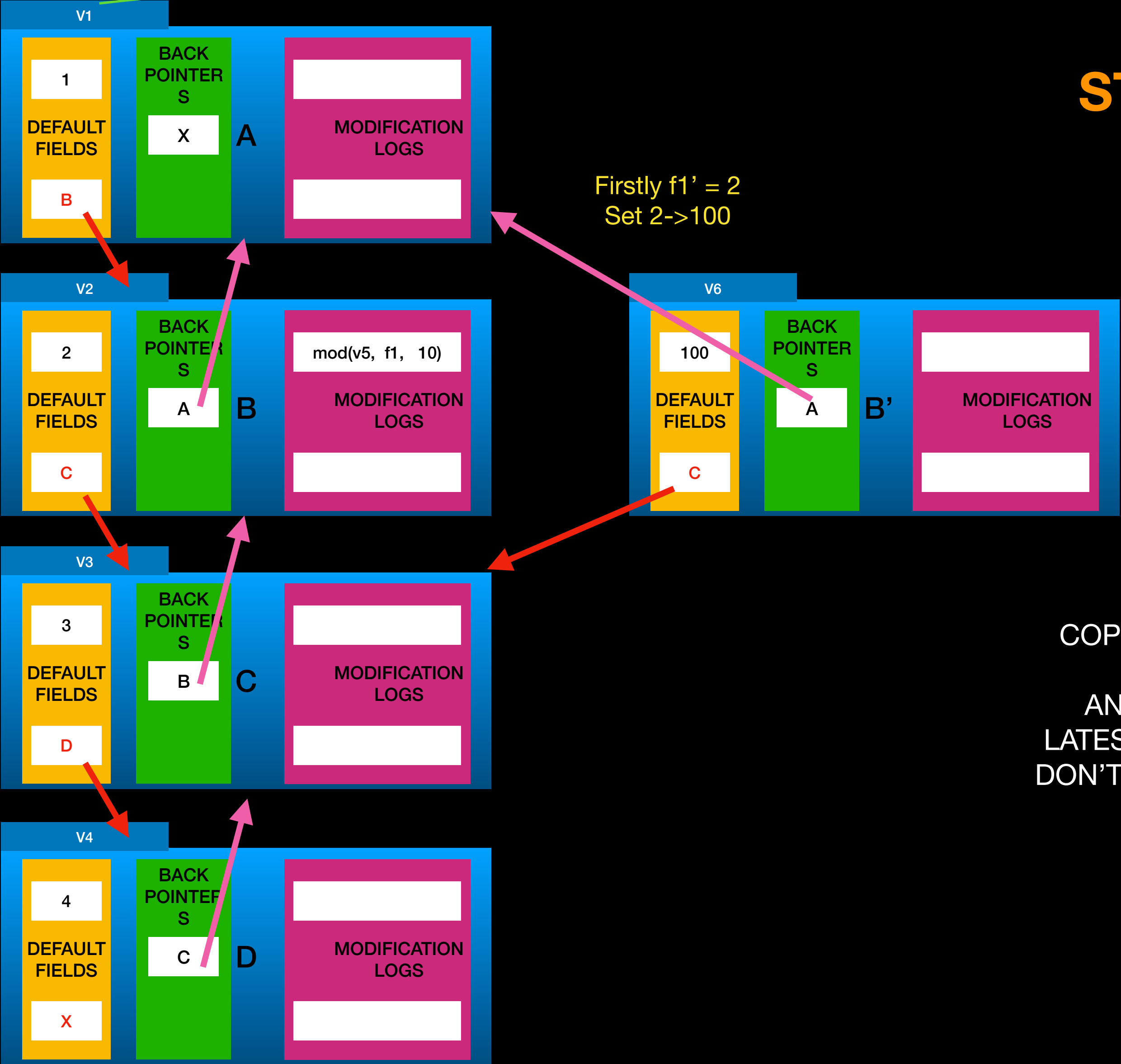
START MODULE

V0

STEP 1

Firstly f1' = 2
Set 2->100

COPY THE WHOLE
NODE
AND KEEP THE
LATEST VALUES AND
DON'T COPY MODLOG



VERSION REDIRECTOR MODULE	
Ver	block
1	A

VERSION REDIRECTOR MODULE	
Ver	block
2	B
6	B'

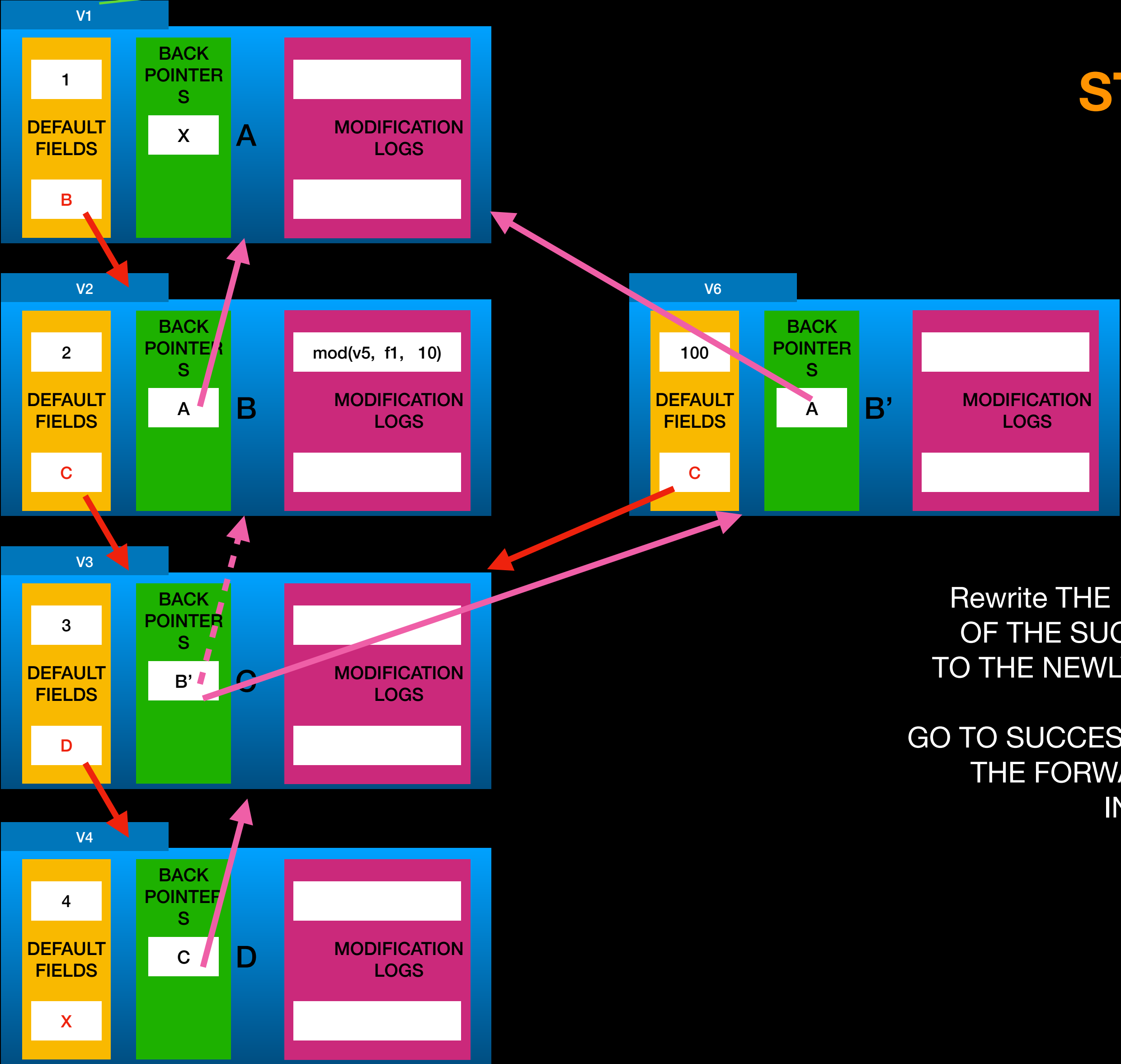
VERSION REDIRECTOR MODULE	
Ver	block
3	C

VERSION REDIRECTOR MODULE	
Ver	block
4	D

START MODULE

V0

STEP 2



VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

VERSION REDIRECTOR MODULE

Ver	block
3	C

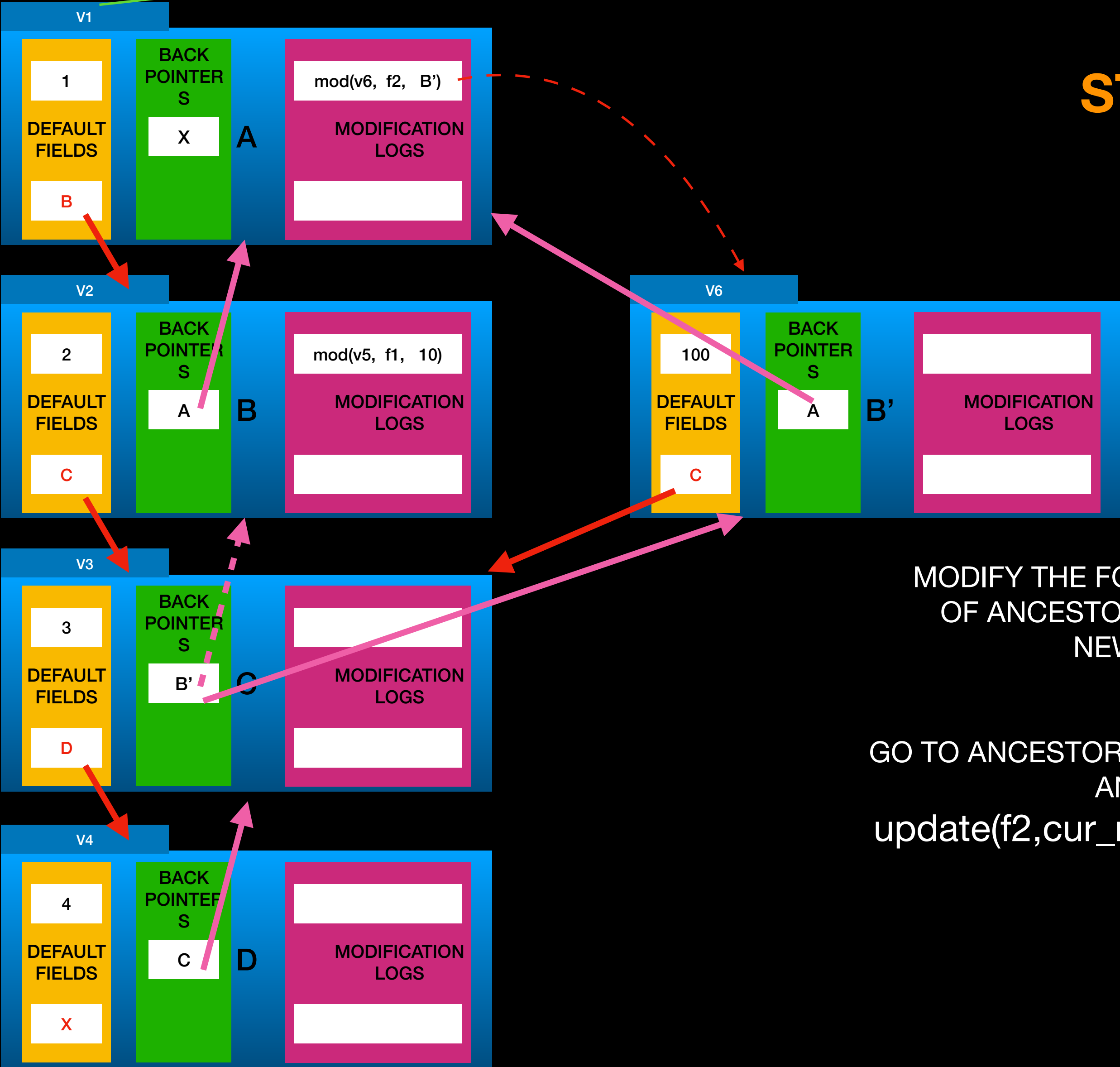
VERSION REDIRECTOR MODULE

Ver	block
4	D

START MODULE

V0

STEP 3



MODIFY THE FORWARD POINTERS
OF ANCESTOR NODES TO THE
NEW NODE

GO TO ANCESTOR NODES RECURSIVELY
AND DO
update(f2,cur_node,NEW_NODE)

VERSION REDIRECTOR MODULE	
Ver	block
1	A

VERSION REDIRECTOR MODULE	
Ver	block
2	B
6	B'

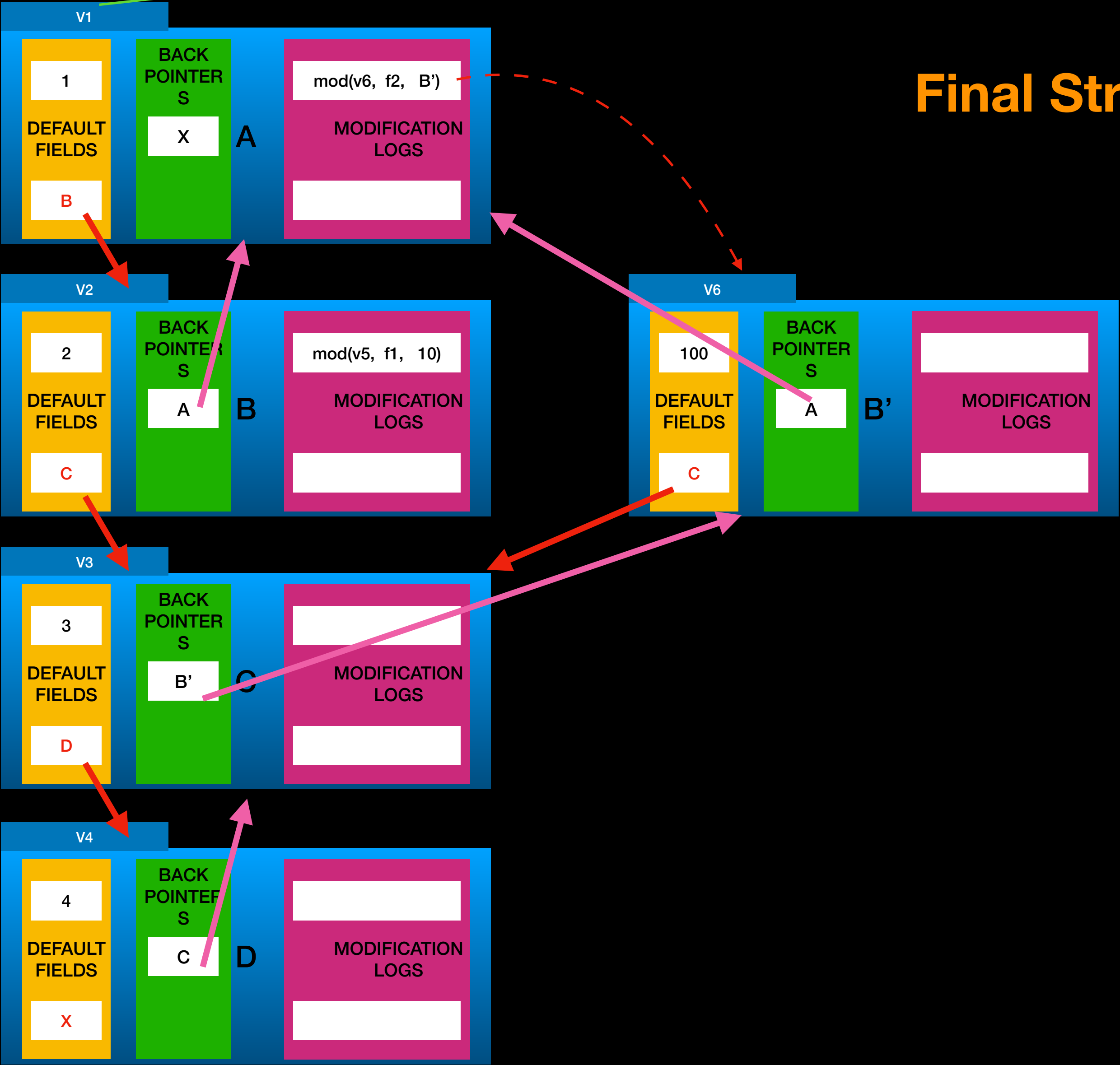
VERSION REDIRECTOR MODULE	
Ver	block
3	C

VERSION REDIRECTOR MODULE	
Ver	block
4	D

START MODULE

V0

Final Structure of v6



VERSION REDIRECTOR MODULE	
Ver	block
1	A

VERSION REDIRECTOR MODULE	
Ver	block
2	B
6	B'

VERSION REDIRECTOR MODULE	
Ver	block
3	C

VERSION REDIRECTOR MODULE	
Ver	block
4	D

Current Version: v7

START MODULE

V0

add(X,B,_) consists of

add(X,B,_) consists of

Create node X

Modify f2 of B' to X

Modify f2 of X to C + Set BP of X to B'

Update BP of C to X

Set f1 of X

VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

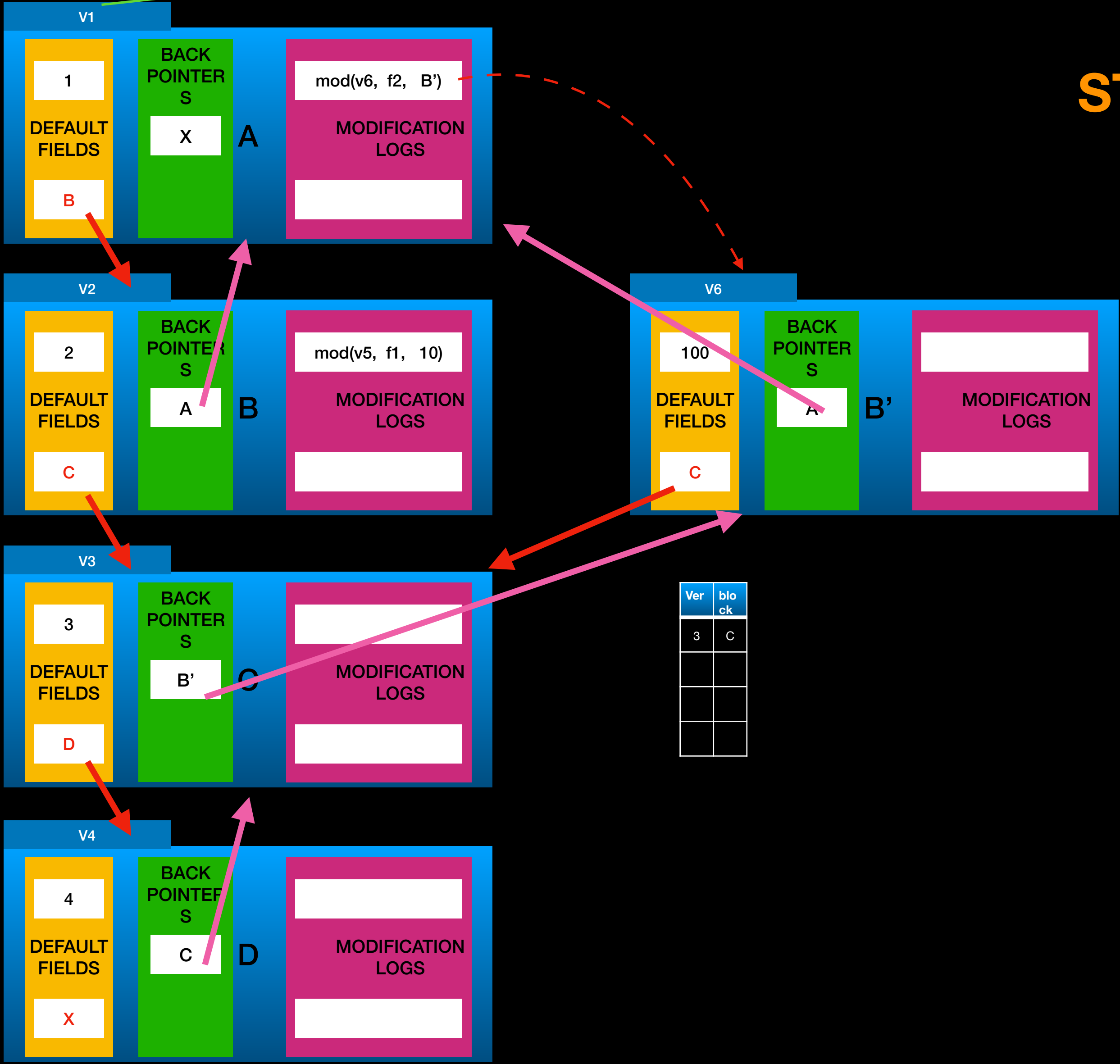
VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

STEP 1



Current Version: v7

START MODULE

V0

add(X,B,_) consists of

add(X,B,_) consists of

Create node X

Modify f2 of B' to X

Modify f2 of X to C + Set BP of X to B'

Update BP of C to X

Set f1 of X

VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

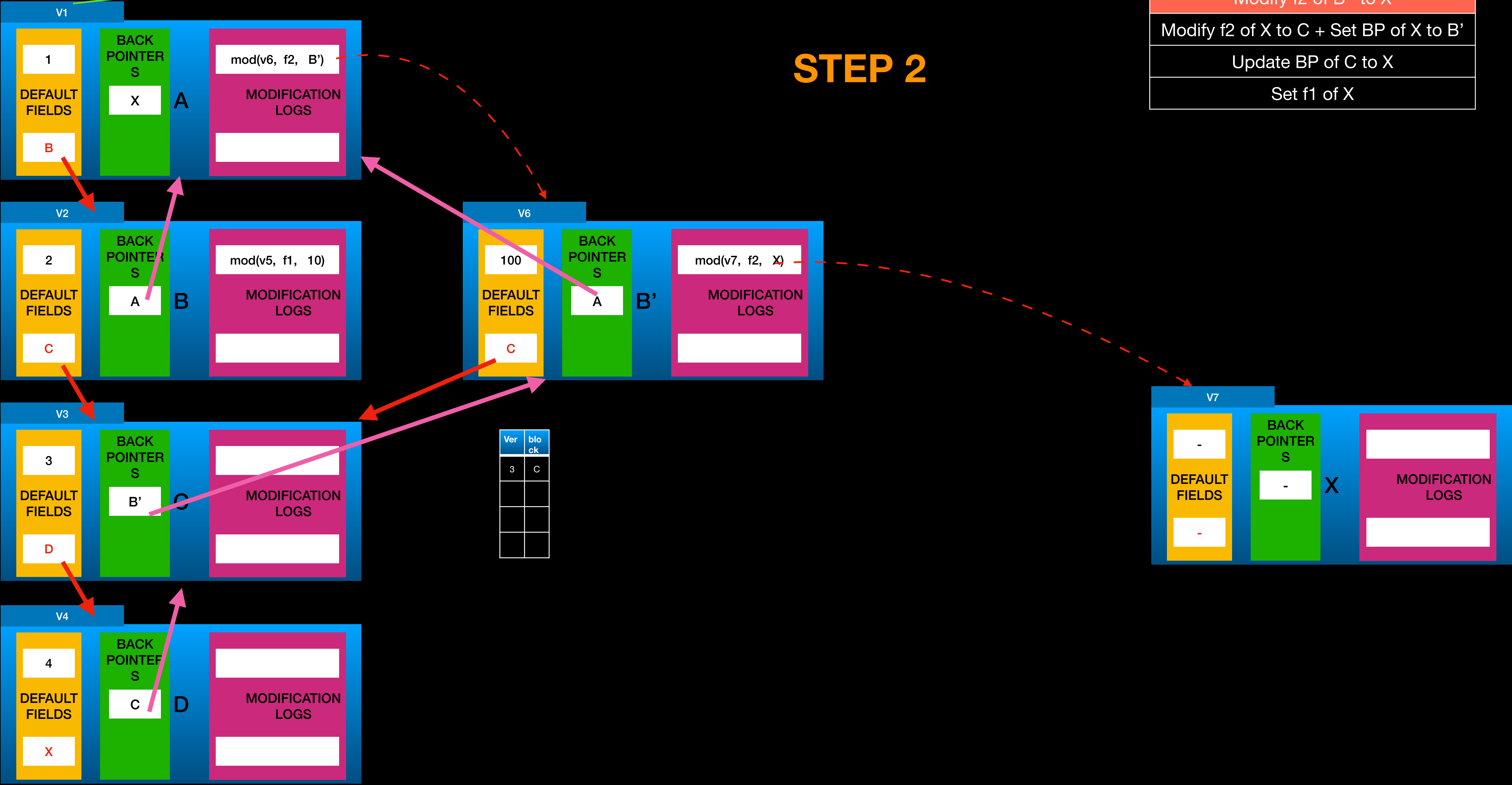
VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

STEP 2



Ver	block
3	C

Current Version: v7

START MODULE

V0

add(X,B,_) consists of

Create node X
Modify f2 of B' to X
Modify f2 of X to C + Set BP of X to B'
Update BP of C to X
Set f1 of X

add(X,B,_) consists of

VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

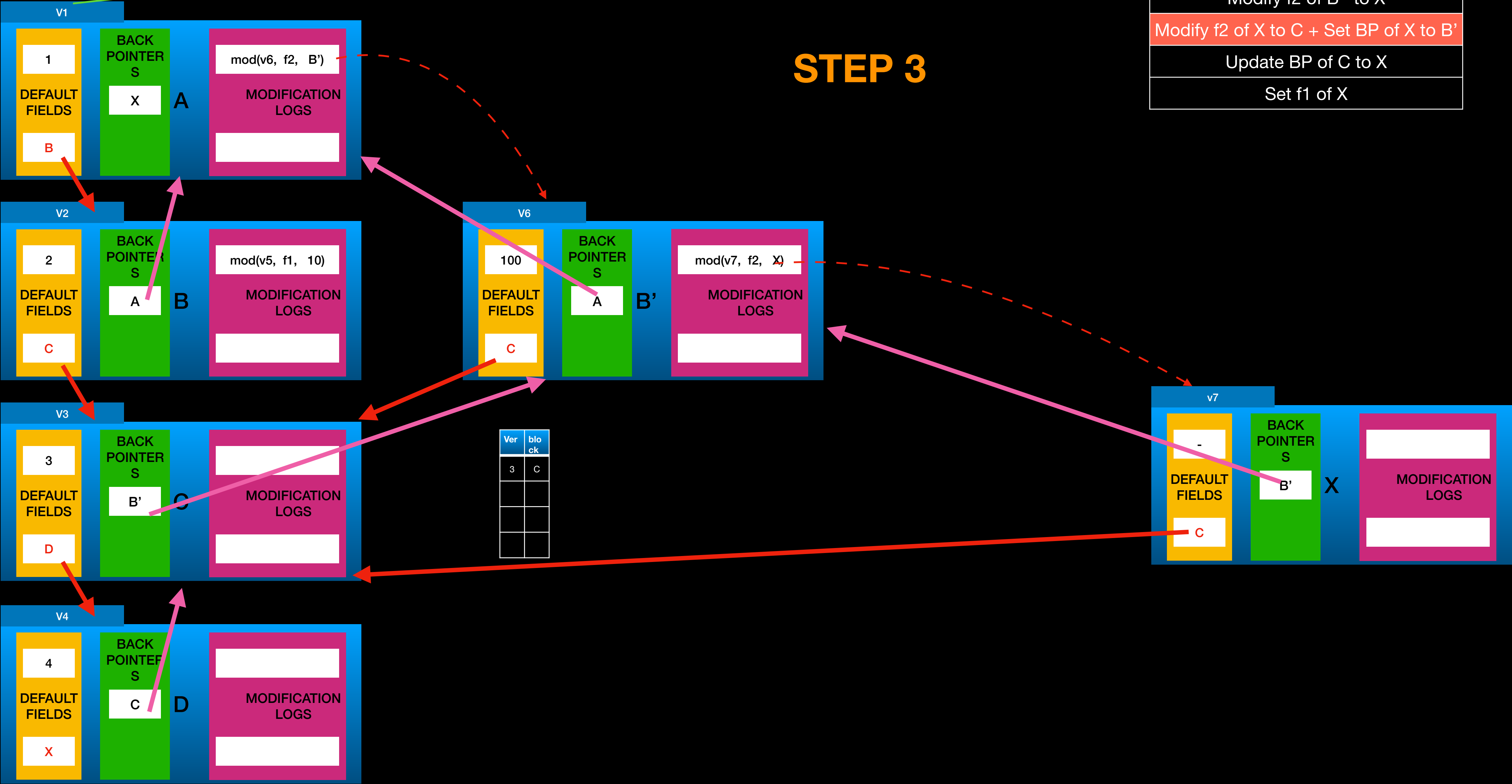
VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

STEP 3



Current Version: v7

START MODULE

V0

add(X,B,_) consists of

Create node X
Modify f2 of B' to X
Modify f2 of X to C + Set BP of X to B'
Update BP of C to X
Set f1 of X

VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

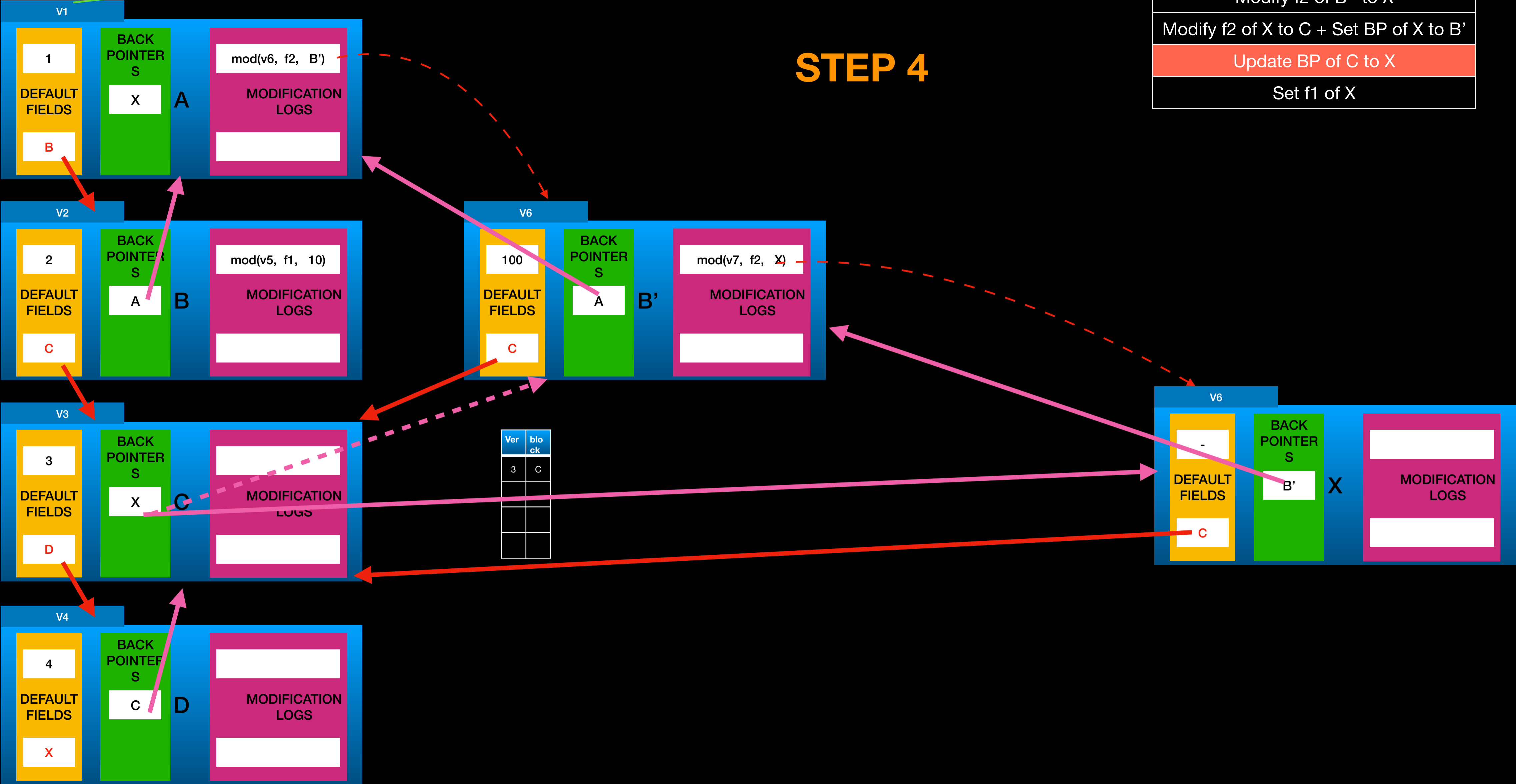
VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

STEP 4



Current Version: v7

START MODULE

V0

add(X,B,_) consists of

Create node X
Modify f2 of B' to X
Modify f2 of X to C + Set BP of X to B'
Update BP of C to X
Set f1 of X

VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

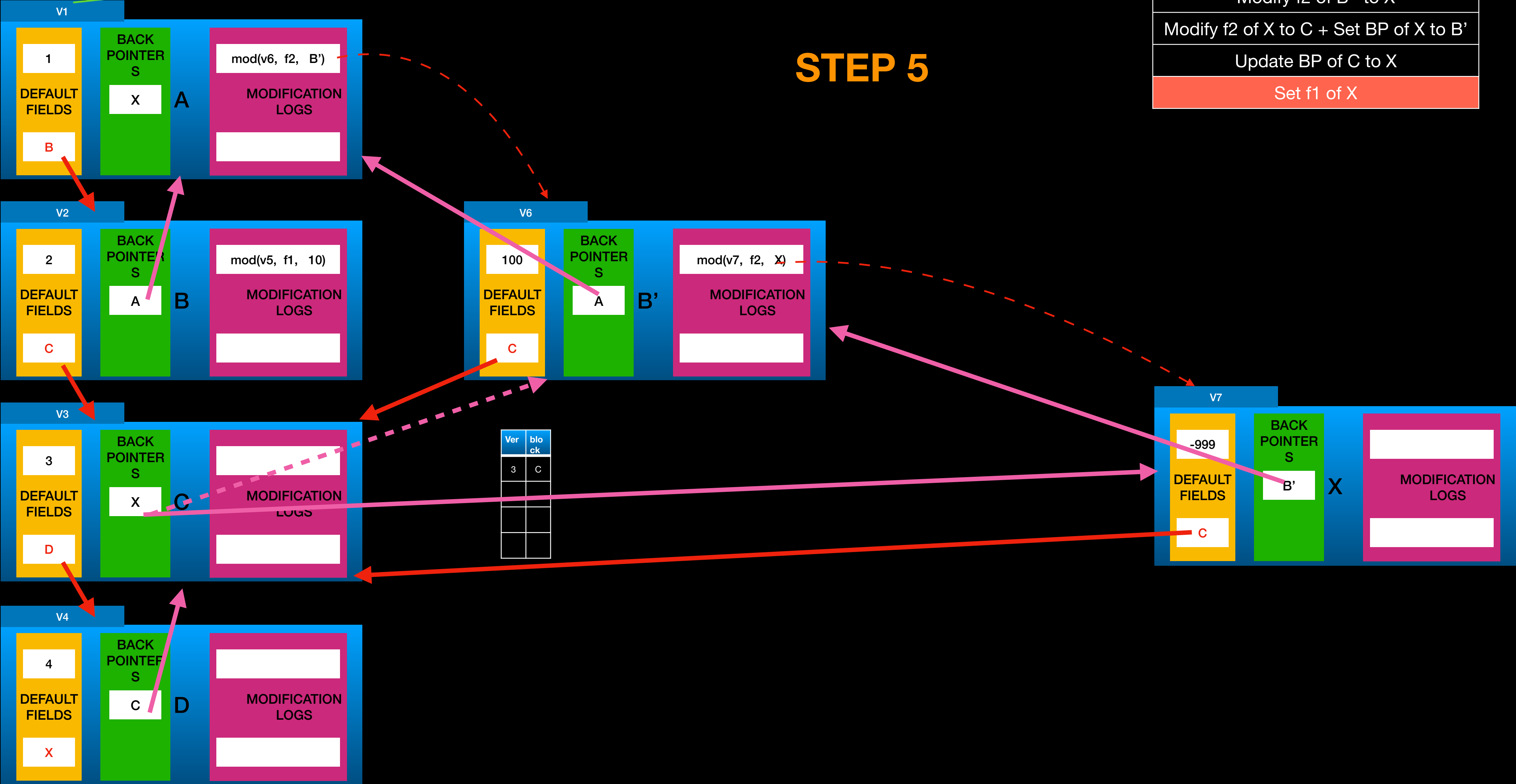
VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

STEP 5



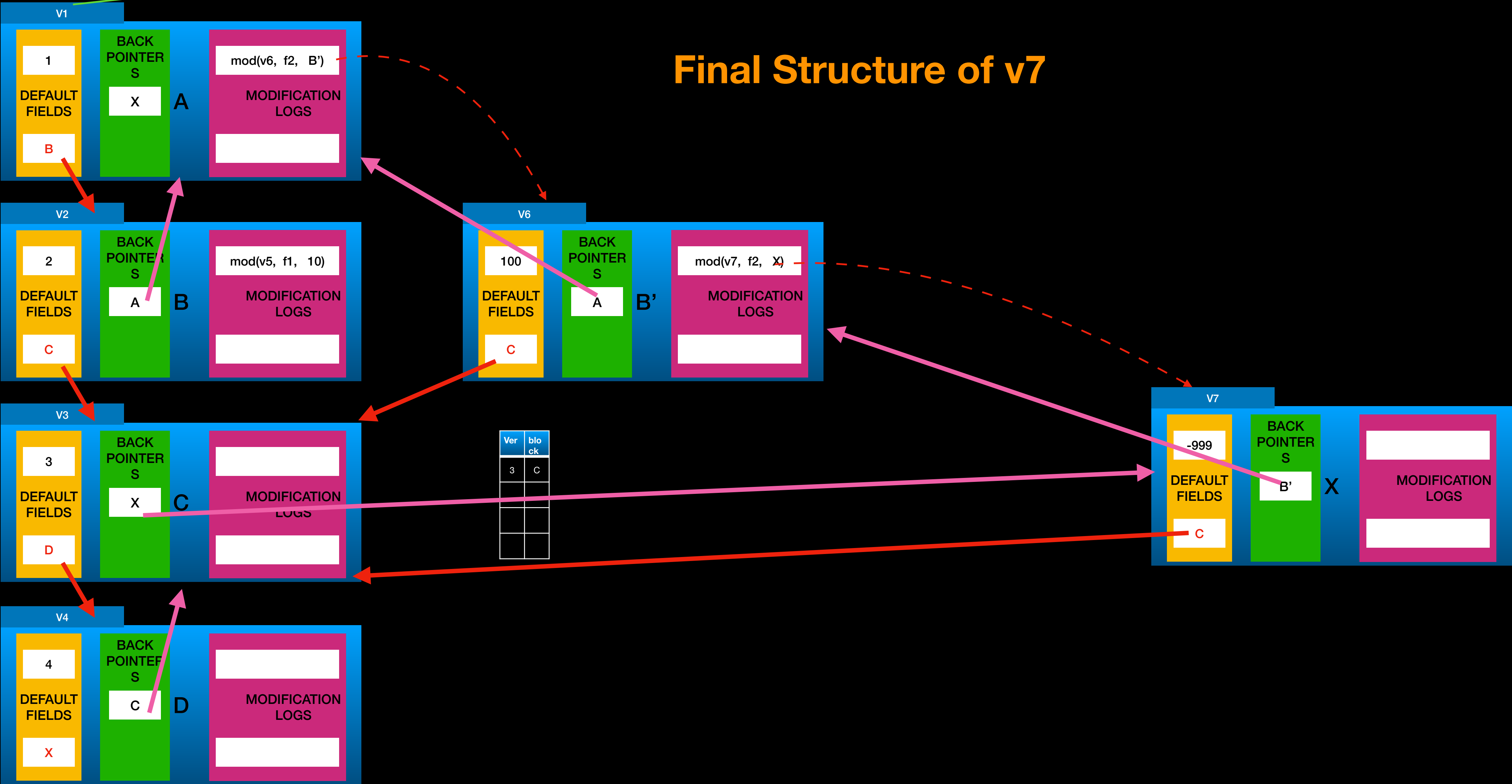
Current Version: v7

START MODULE

vo

add(X,B,_)

Final Structure of v7



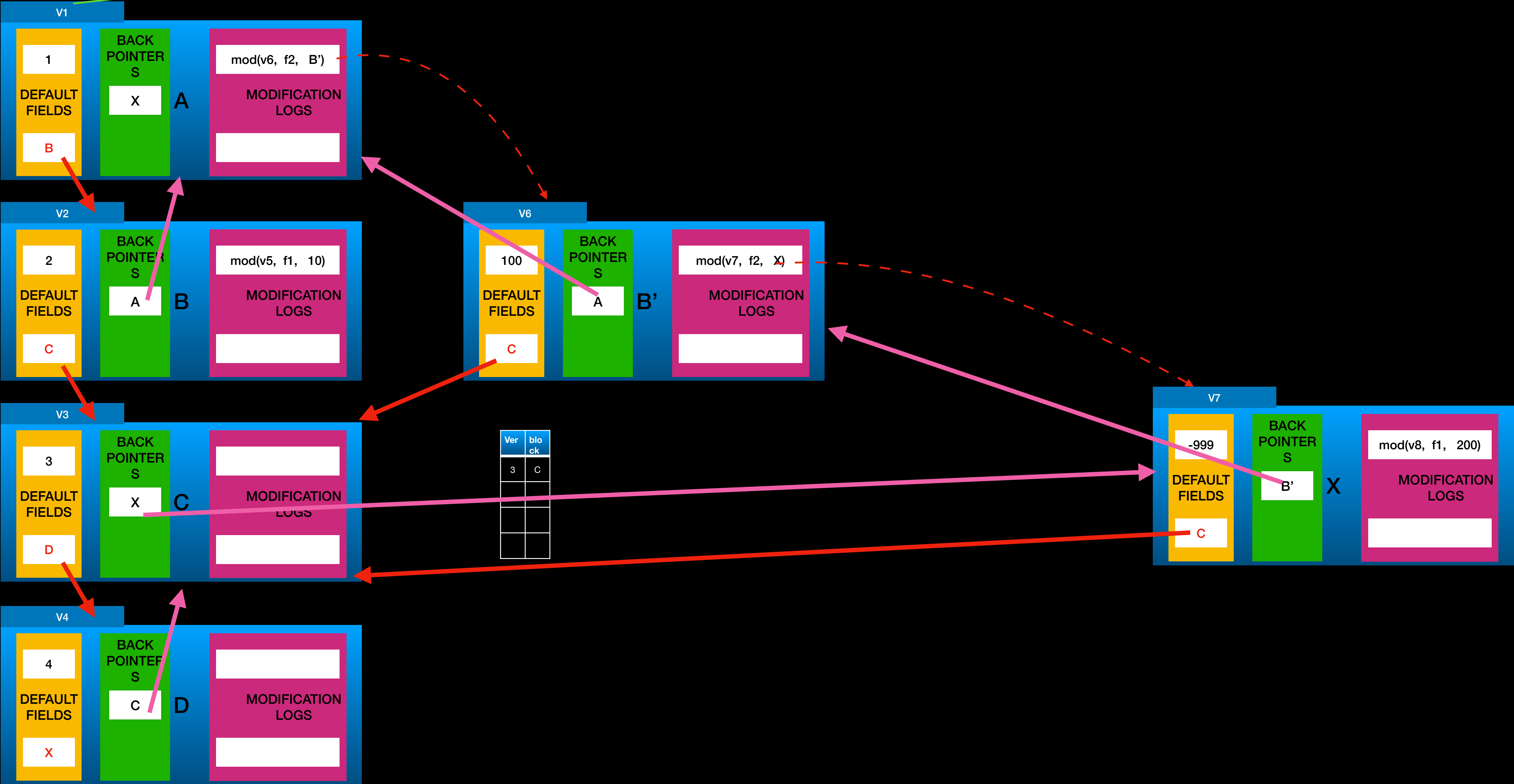
VERSION REDIRECTOR MODULE	
Ver	block
1	A

VERSION REDIRECTOR MODULE	
Ver	block
2	B
6	B'

VERSION REDIRECTOR MODULE	
Ver	block
7	X

VERSION REDIRECTOR MODULE	
Ver	block
4	D

V0



VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'

VERSION REDIRECTOR MODULE

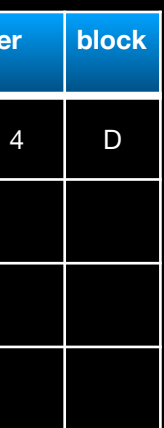
Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

START MODULE

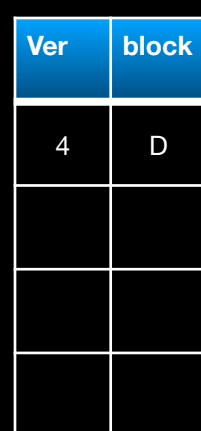
VO



START MODULE

VO

Firstly $f_1' = 100$
Set $100 \rightarrow 110$



COPY THE WHOLE
NODE
AND KEEP THE
LATEST VALUES AND
DON'T COPY MODLOG

START MODULE

V0

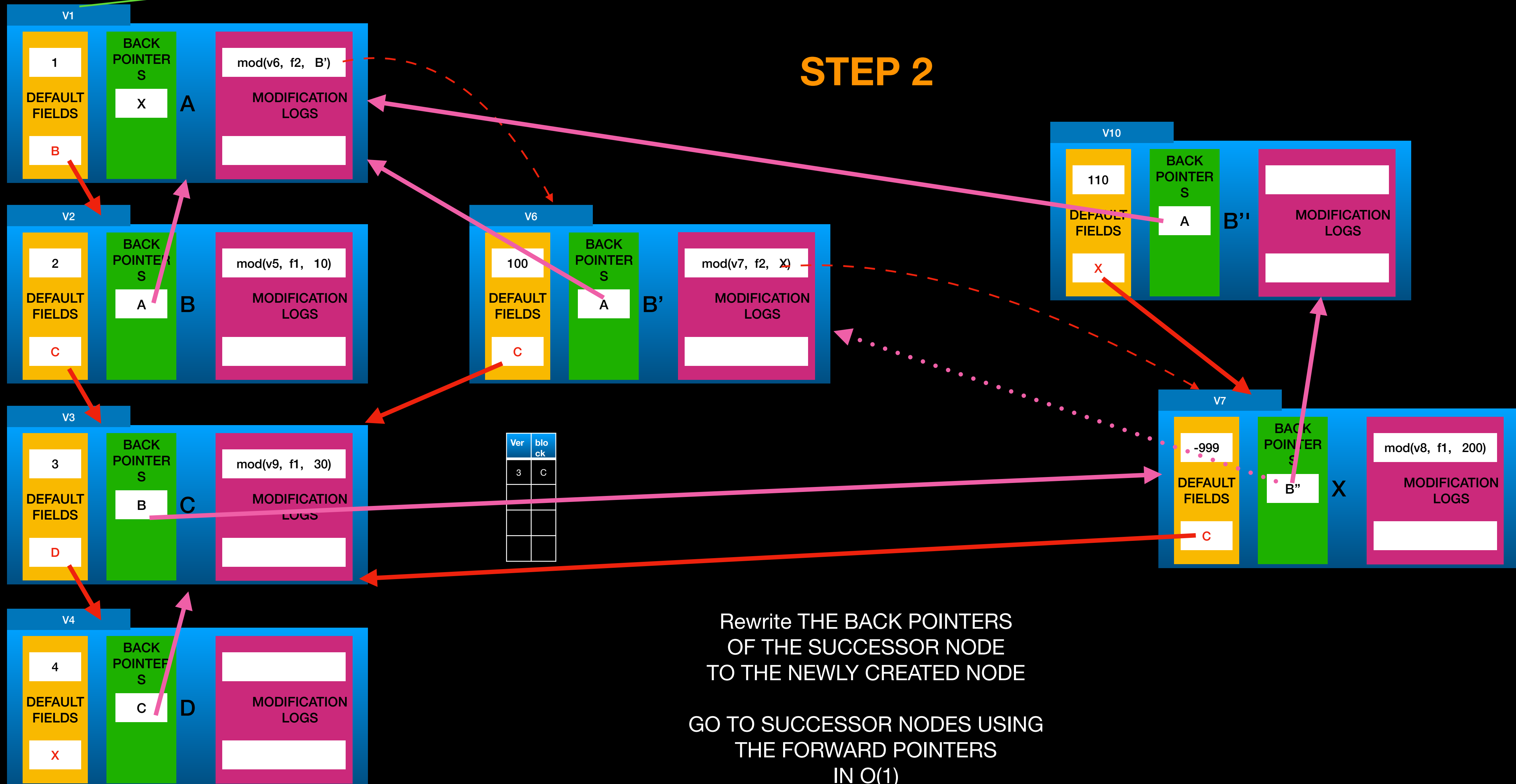
VERSION
REDIRECTOR
MODULE

VERSION
REDIRECTOR
MODULE

VERSION
REDIRECTOR
MODULE

VERSION
REDIRECTOR
MODULE

er	block
4	D



Rewrite THE BACK POINTERS
OF THE SUCCESSOR NODE
TO THE NEWLY CREATED NODE

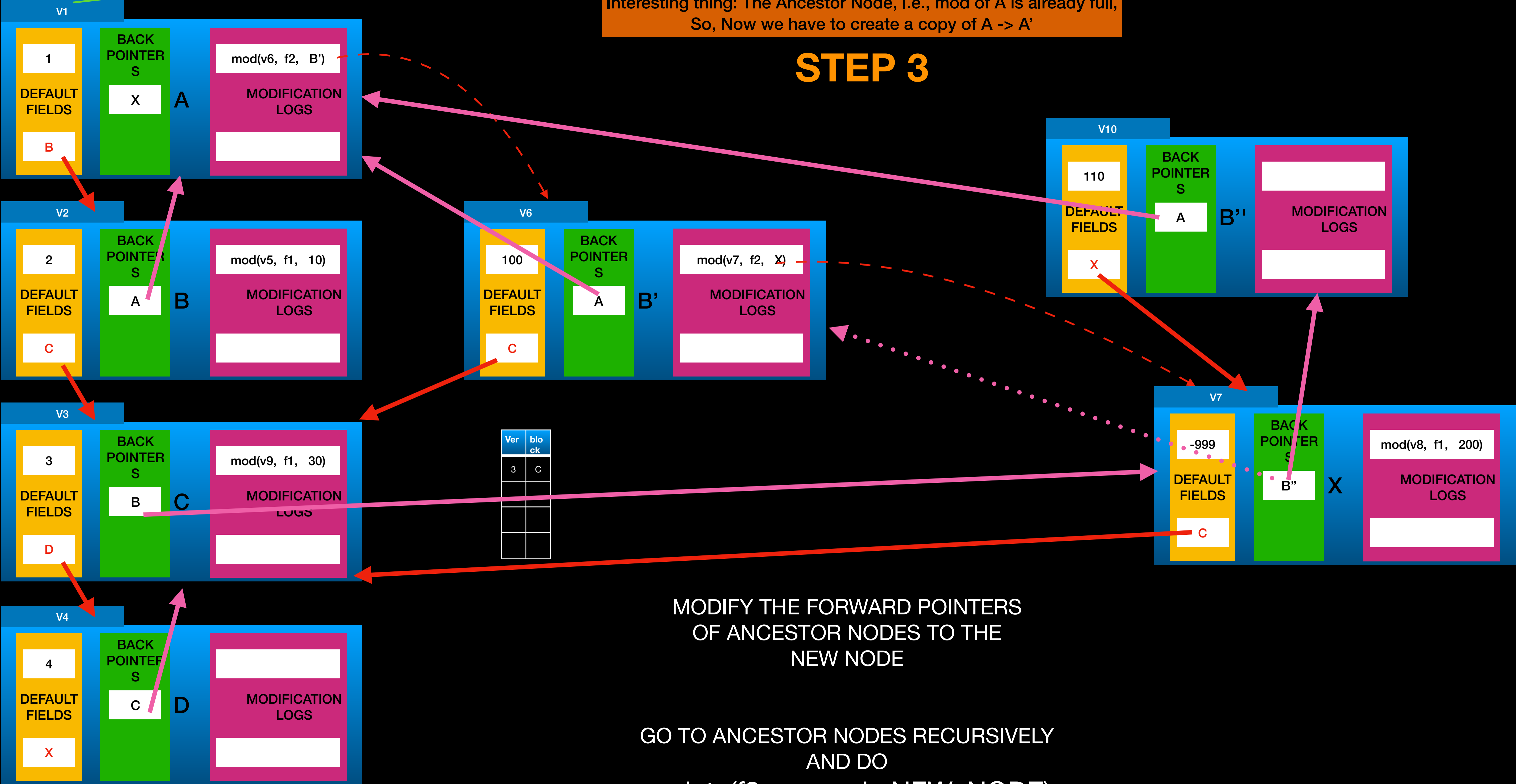
GO TO SUCCESSOR NODES USING
THE FORWARD POINTERS
IN $O(1)$

START MODULE

V0

Interesting thing: The Ancestor Node, I.e., mod of A is already full,
So, Now we have to create a copy of A -> A'

STEP 3



VERSION REDIRECTOR MODULE

Ver	block
1	A

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

Ver	block
4	D

START MODULE

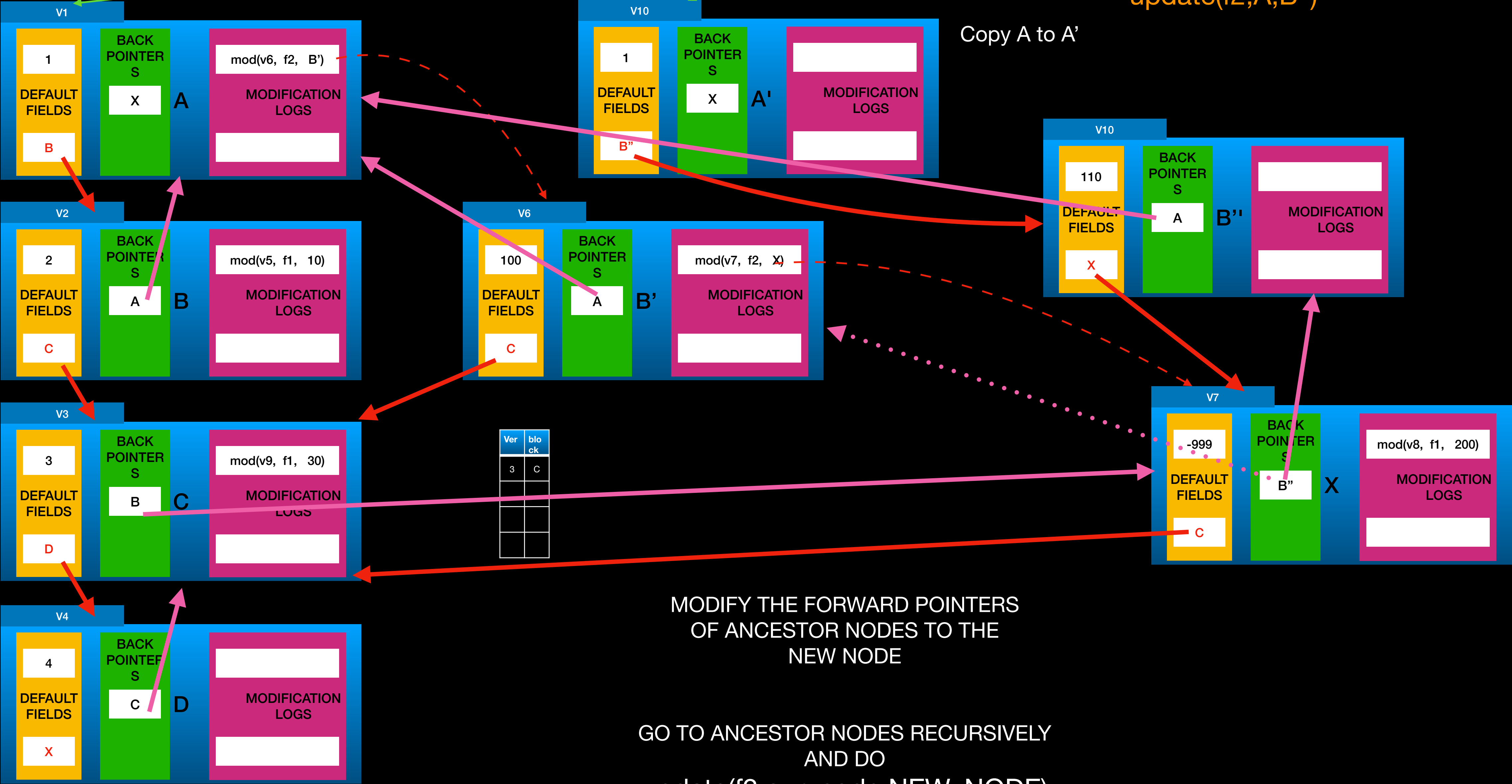
V0V10

STEP 3

SUBSTEP 1

update(f2,A,B'')

update(f1, B,110)



VERSION REDIRECTOR MODULE	
Ver	block
1	A
10	A''

VERSION REDIRECTOR MODULE	
Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE	
Ver	block
7	X

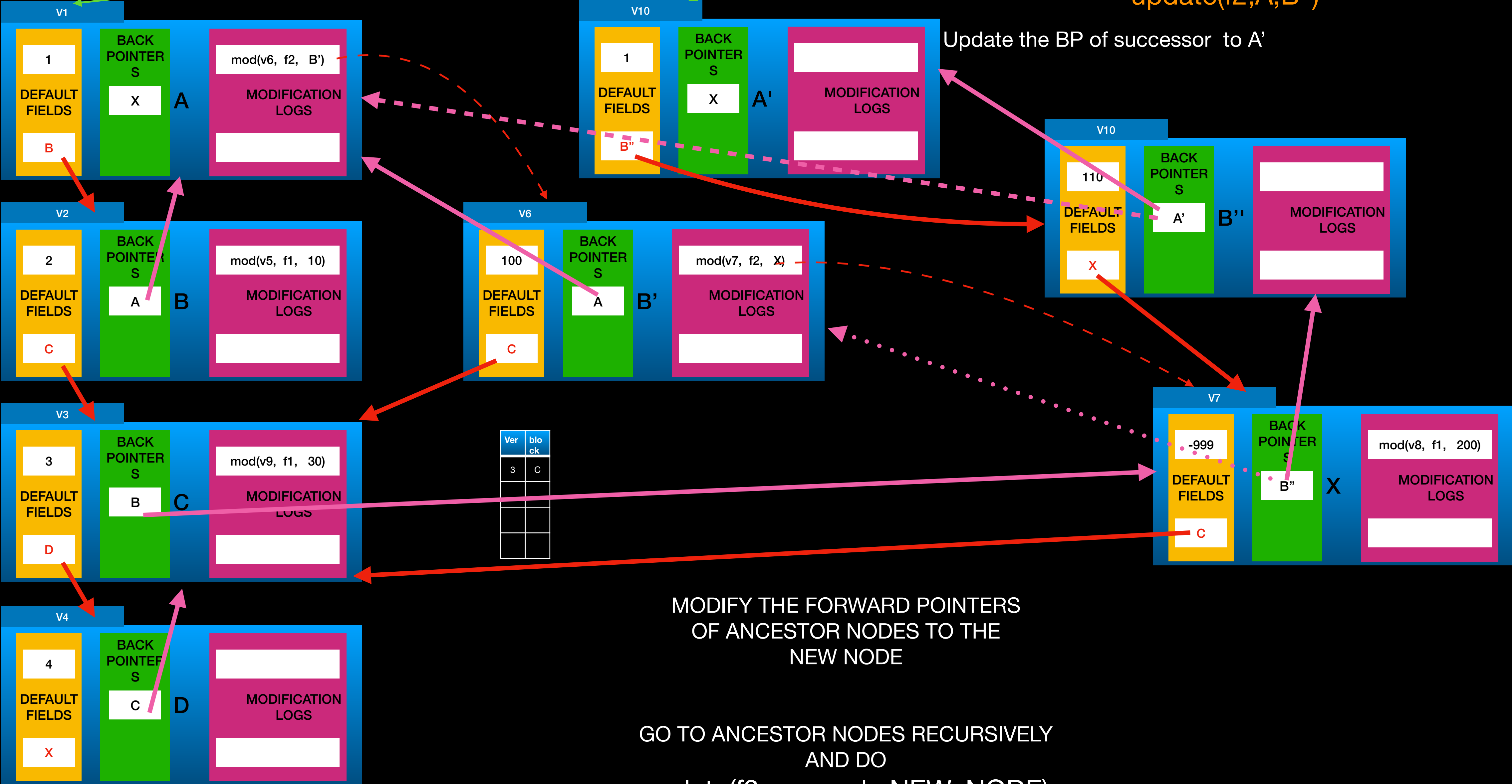
VERSION REDIRECTOR MODULE	
Ver	block
4	D

START MODULE

V0V10

STEP 3
SUBSTEP 2
update(f2,A,B'')

update(f1, B,110)



VERSION REDIRECTOR MODULE

Ver	block
1	A
10	A''

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE

Ver	block
7	X

VERSION REDIRECTOR MODULE

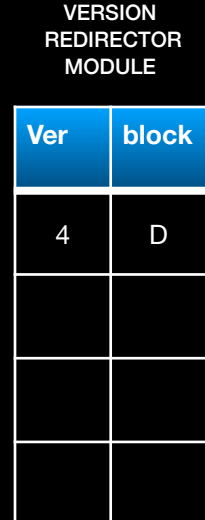
Ver	block
4	D

START MODULE

STEP 3

SUBSTEP 3

update(f2,A,B")



Recursion Ends too

MODIFY THE FORWARD POINTERS OF ANCESTOR NODES TO THE NEW NODE

GO TO ANCESTOR NODES RECURSIVELY
AND DO
update(f2,cur_node,NEW_NODE)

START MODULE



er	block
1	A
10	A''

er	block
2	B
6	B'
10	B''

er	block
7	X

er	block
4	D

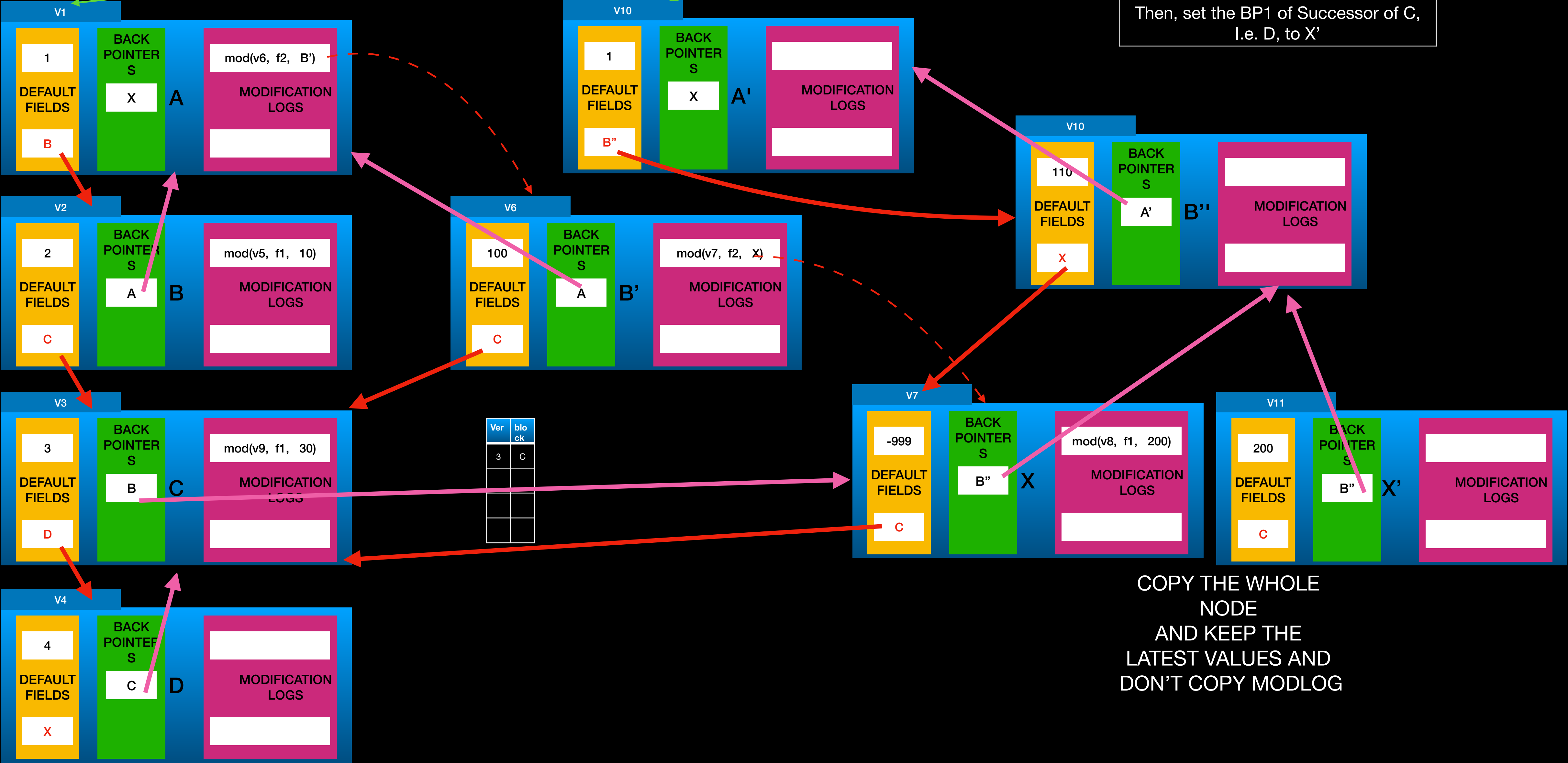
START MODULE

V0V10

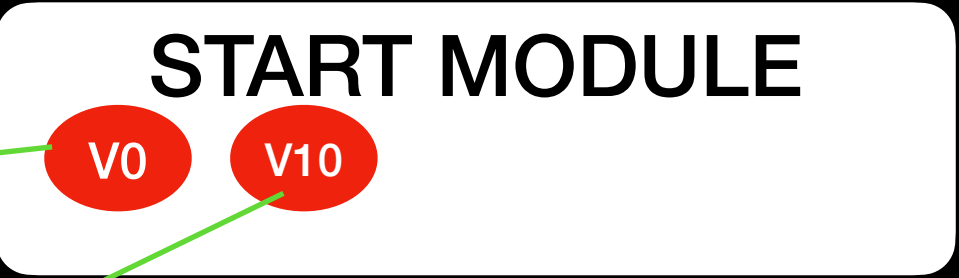
Remove C consists of

Modify F2 of Parent of C (i.e., X) -> F2 of C (successor of C)

Then, set the BP1 of Successor of C, I.e. D, to X'



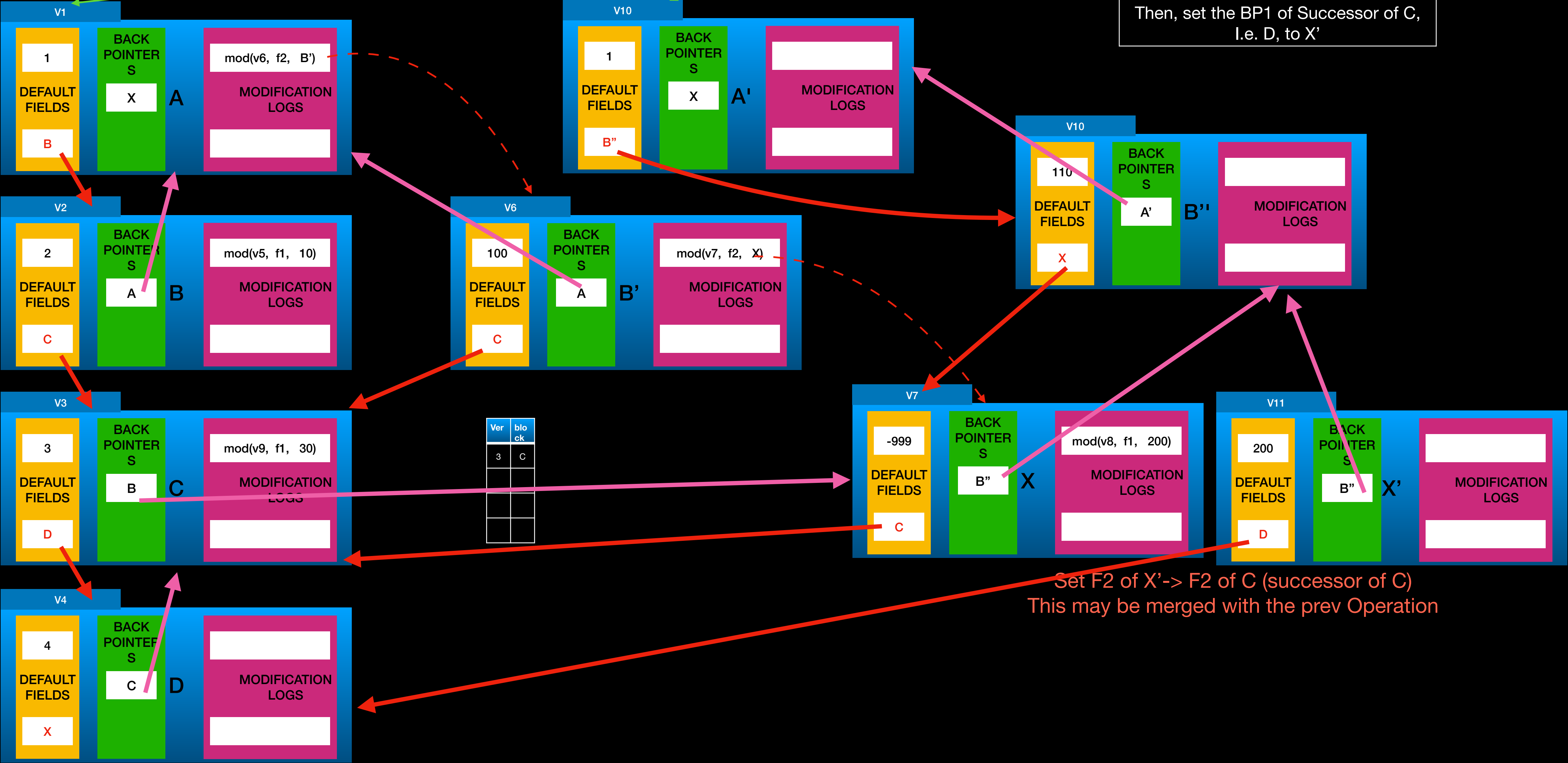
COPY THE WHOLE
NODE
AND KEEP THE
LATEST VALUES AND
DON'T COPY MODLOG



Remove C consists of

Modify F2 of Parent of C (i.e., X) -> F2 of C (successor of C)

Then, set the BP1 of Successor of C, I.e. D, to X'



VERSION REDIRECTOR MODULE

Ver	block
1	A
10	A"

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B"

VERSION REDIRECTOR MODULE

Ver	block
7	X
11	X'

VERSION REDIRECTOR MODULE

Ver	block
4	D

Set F2 of X'-> F2 of C (successor of C)
This may be merged with the prev Operation

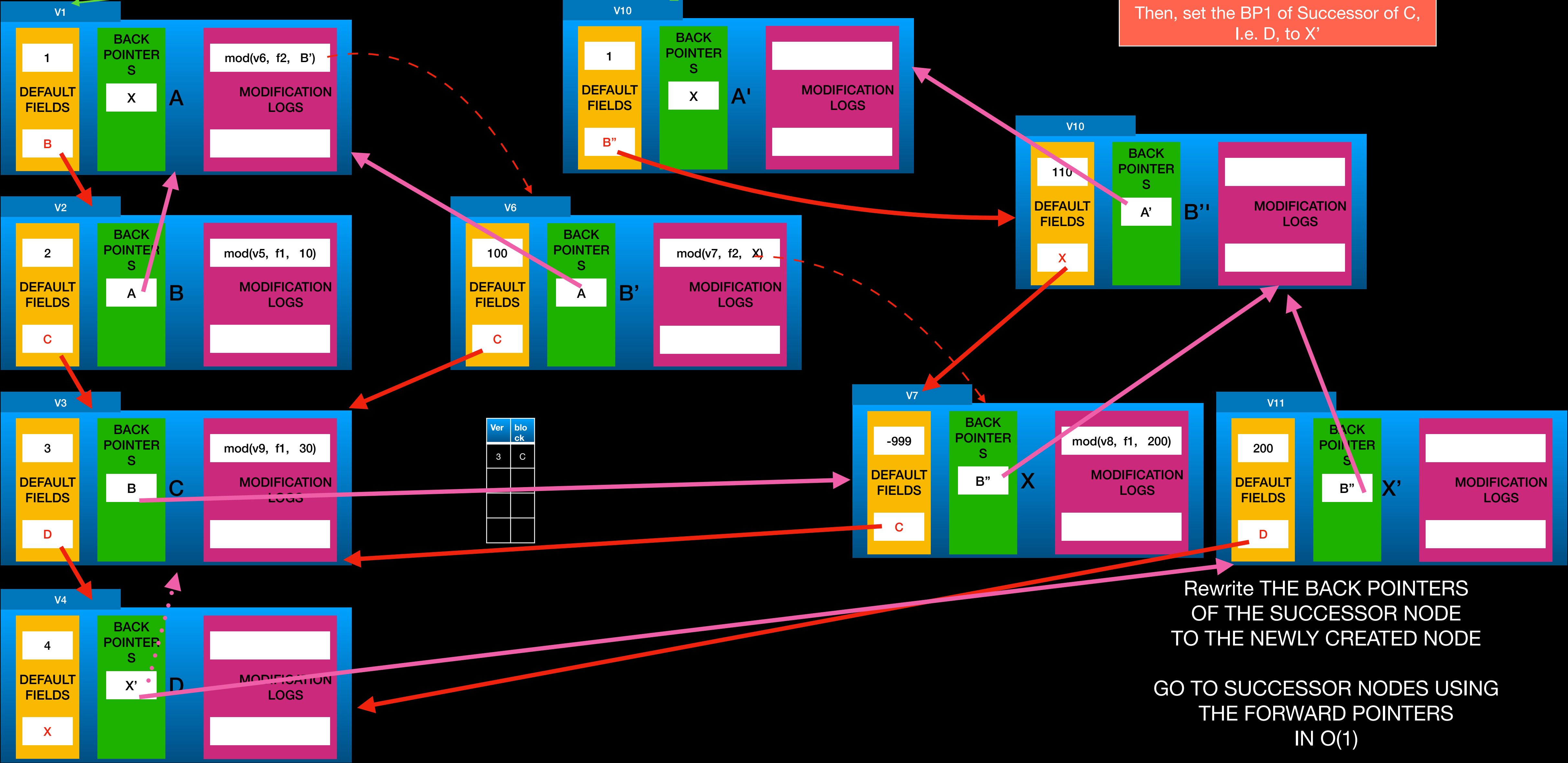
START MODULE

V0V10

Remove C consists of

Modify F2 of Parent of C (i.e., X) -> F2 of C (successor of C)

Then, set the BP1 of Successor of C, i.e. D, to X'



Rewrite THE BACK POINTERS OF THE SUCCESSOR NODE TO THE NEWLY CREATED NODE

GO TO SUCCESSOR NODES USING THE FORWARD POINTERS IN O(1)

START MODULE

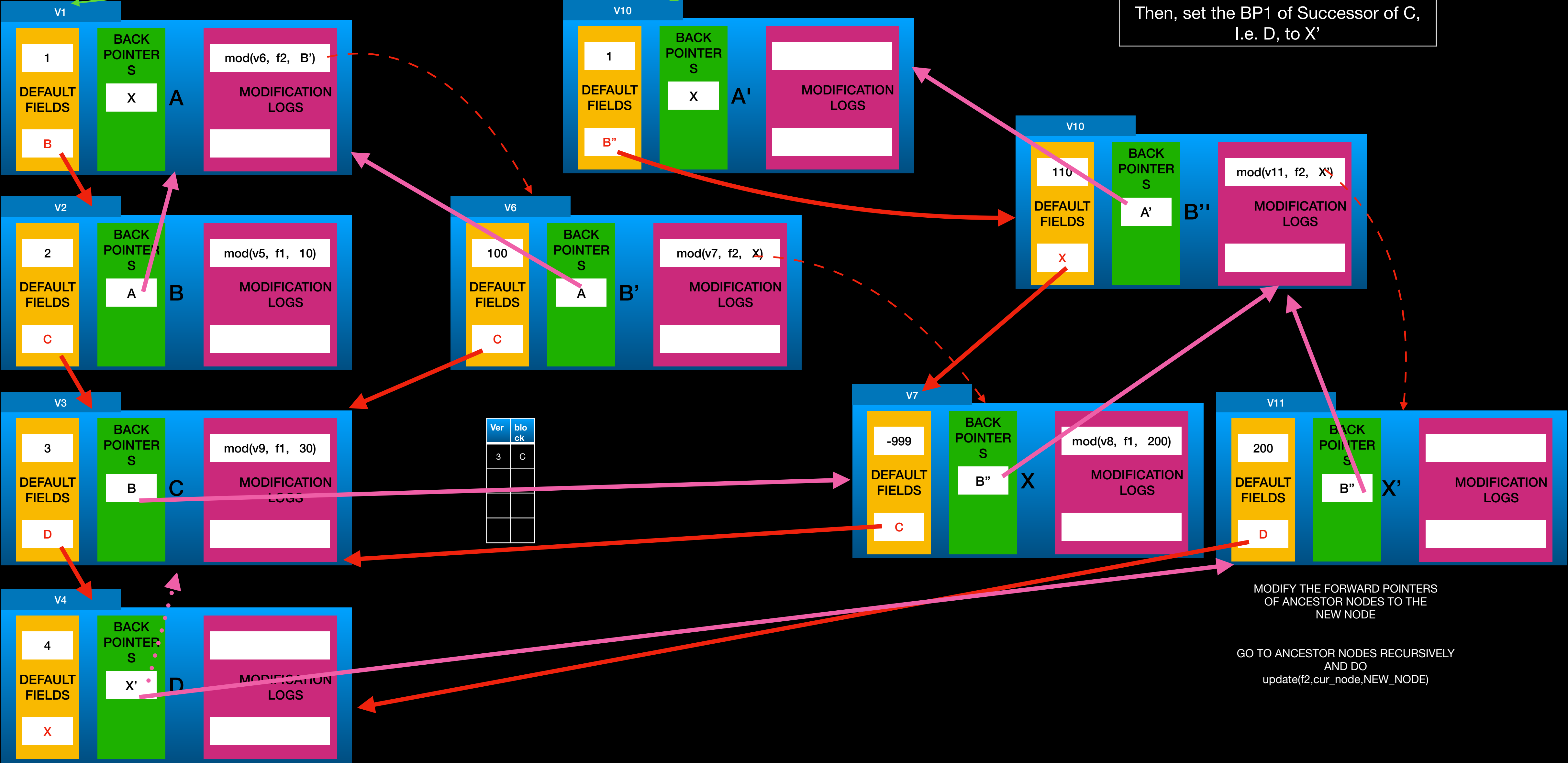
V0V10

Remove C consists of

Modify F2 of Parent of C (i.e., X) -> F2 of C (successor of C)

Then, set the BP1 of Successor of C, I.e. D, to X'

remove(C)



VERSION REDIRECTOR MODULE

Ver	block
1	A
10	A''

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE

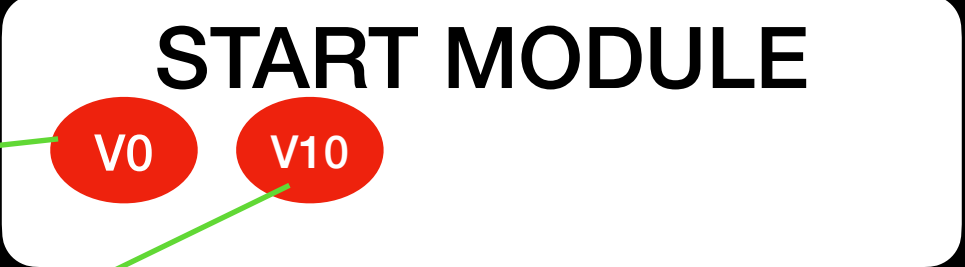
Ver	block
7	X
11	X'

VERSION REDIRECTOR MODULE

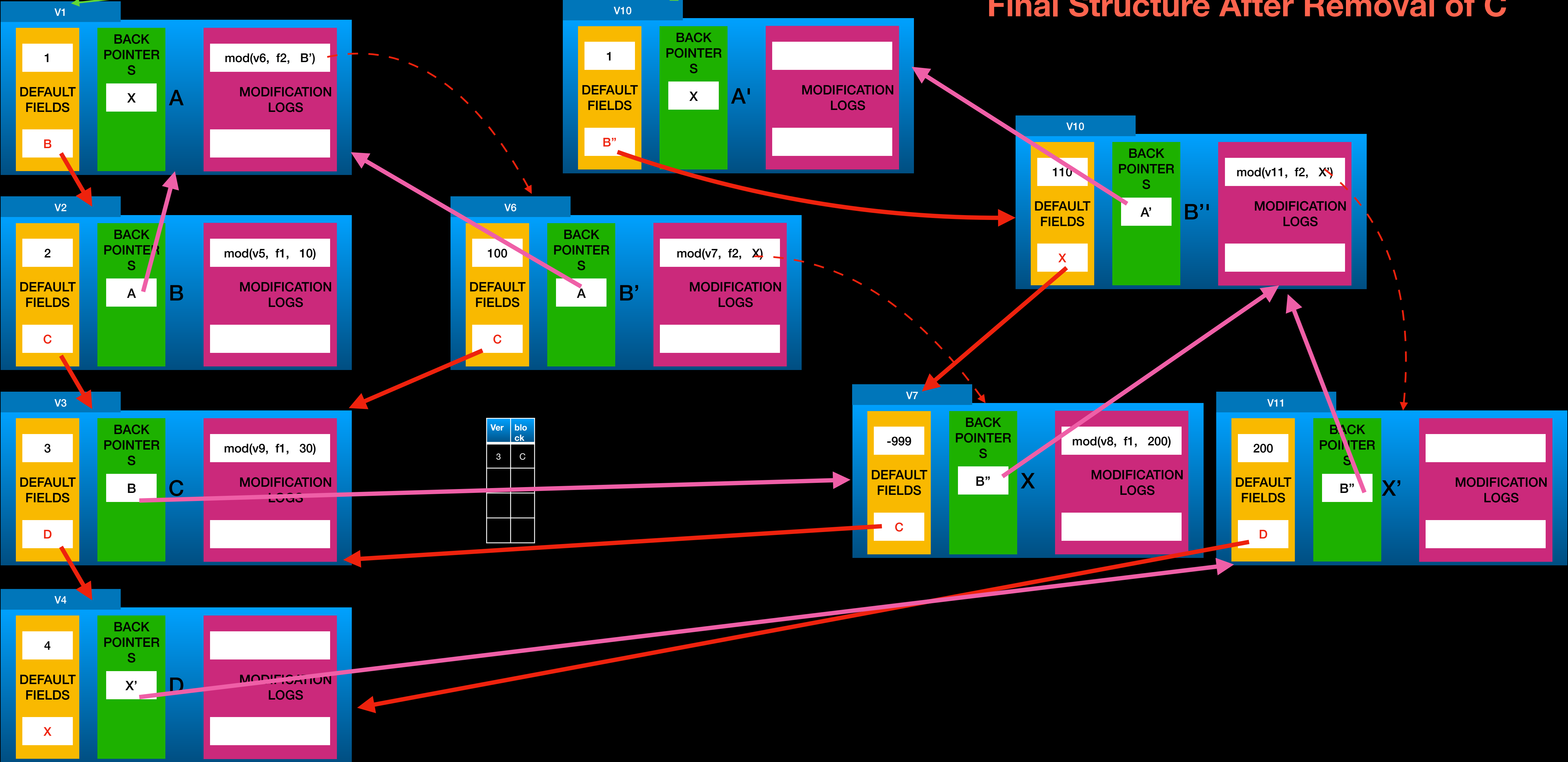
Ver	block
4	D

MODIFY THE FORWARD POINTERS OF ANCESTOR NODES TO THE NEW NODE

GO TO ANCESTOR NODES RECURSIVELY AND DO
update(f2,cur_node,NEW_NODE)



Final Structure After Removal of C



VERSION REDIRECTOR MODULE

Ver	block
1	A
10	A''

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE

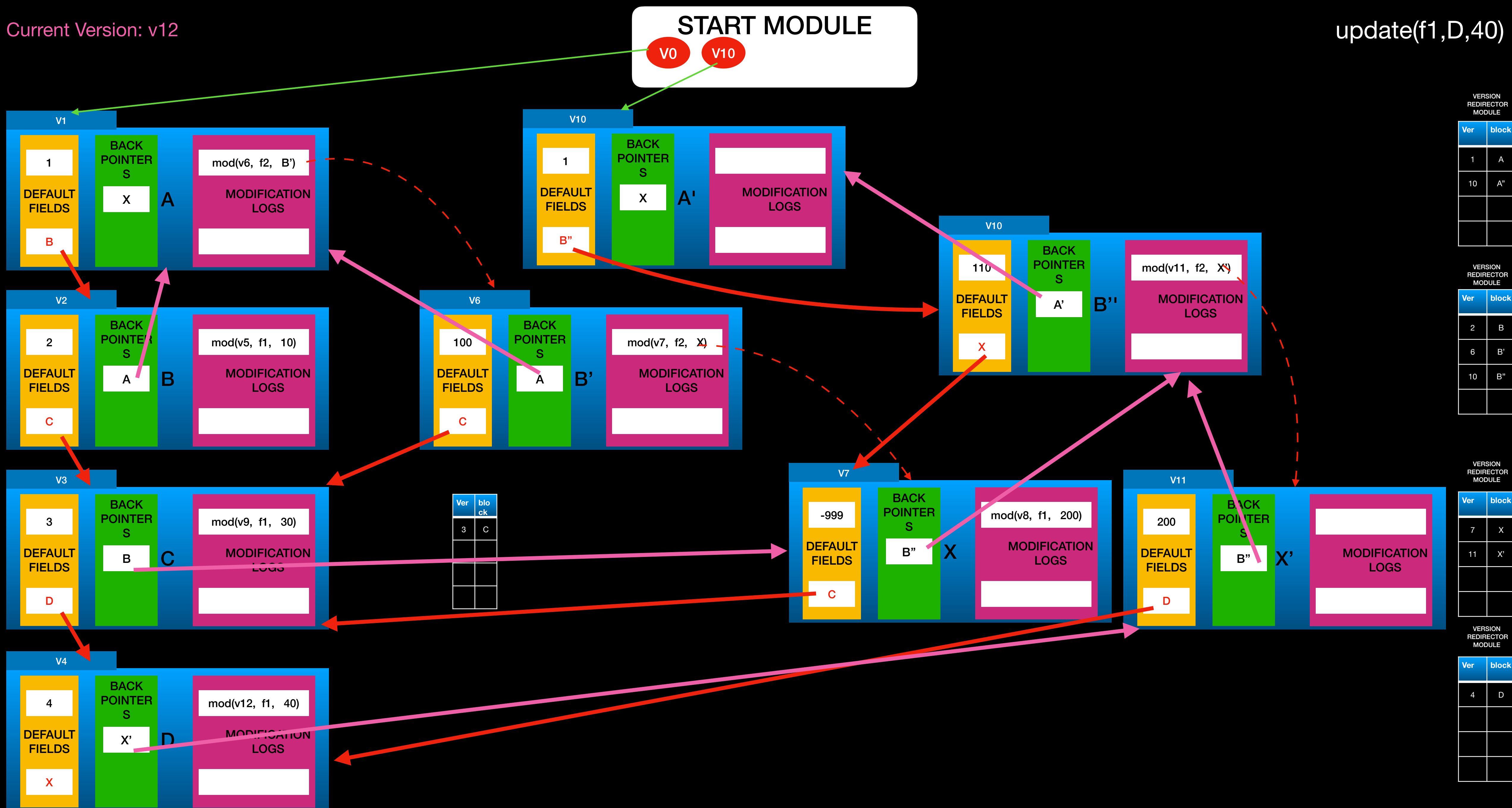
Ver	block
7	X
11	X'

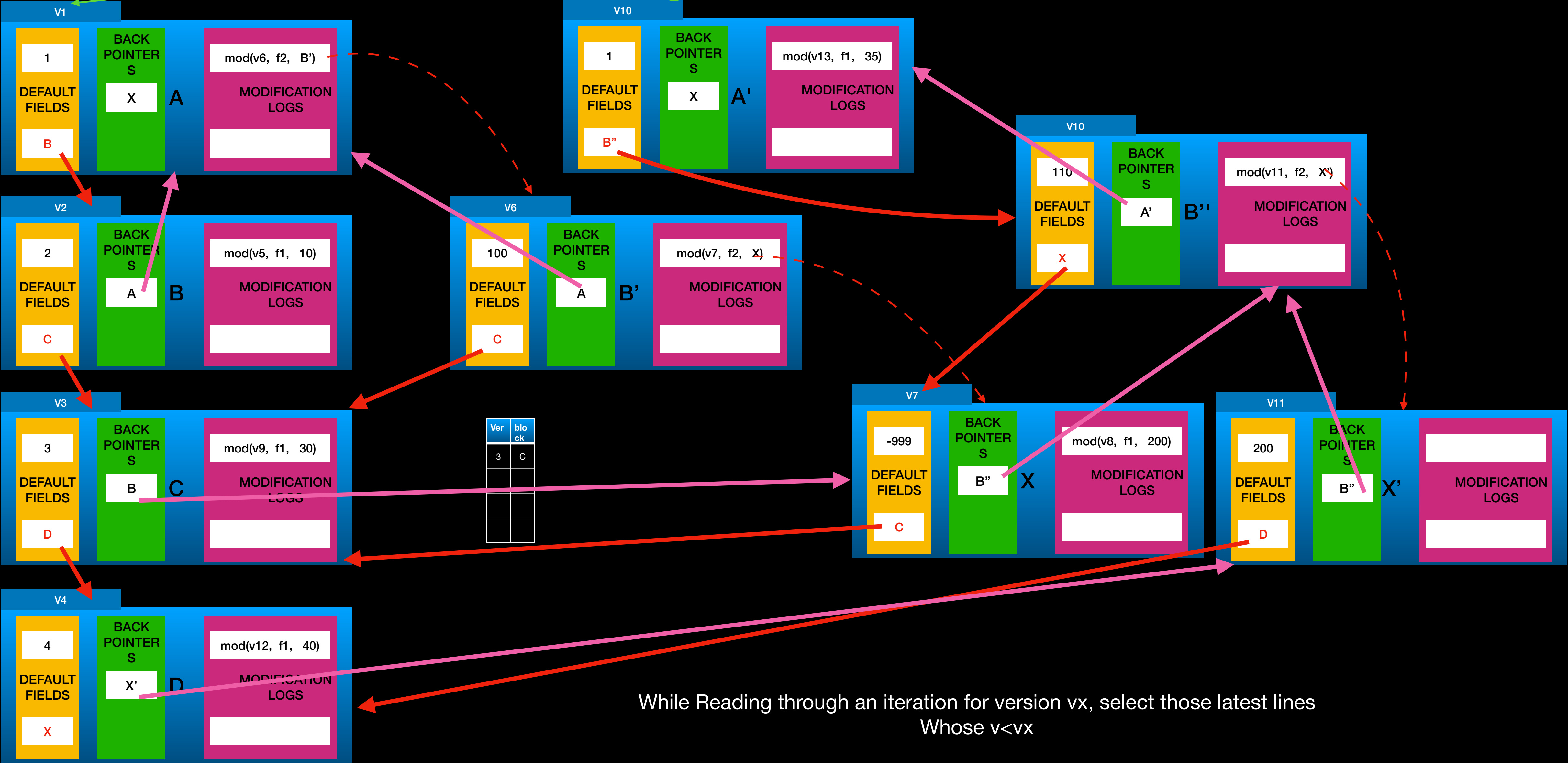
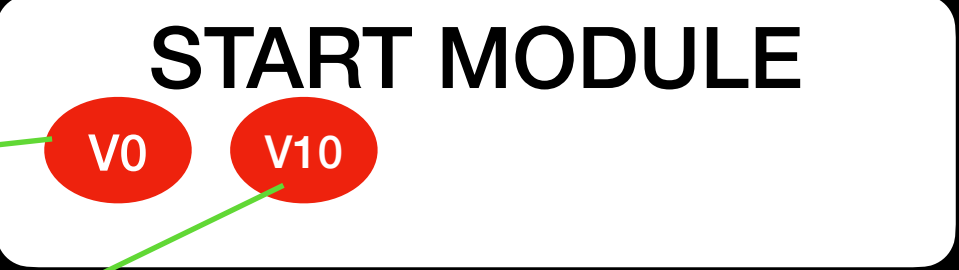
VERSION REDIRECTOR MODULE

Ver	block
4	D

Current Version: v12

```
update(f1,D,40)
```





VERSION REDIRECTOR MODULE

Ver	block
1	A
10	A''

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE

Ver	block
7	X
11	X'

VERSION REDIRECTOR MODULE

Ver	block
4	D

While Reading through an iteration for version vx, select those latest lines
Whose v<vx

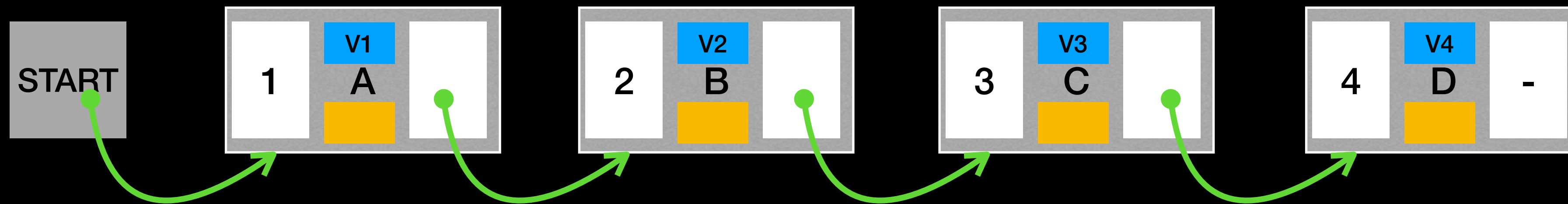
iterate_LL_at_v(vx)

To print the list at v_x

- Start from start module
- Choose those lines whose version **IS JUST LESS THAN OR EQUAL TO** v_x (as, suppose if we are traversing for v5 , either v0,v1,v2,v3, v4 or v5 can be on that path, lines>v5 can't be on that).
- The word “JUST LESS” is written because if a NODE has two version lines in that, e.g. v2 and v4 and we are searching for v5, then we should prefer v4 line over v2.

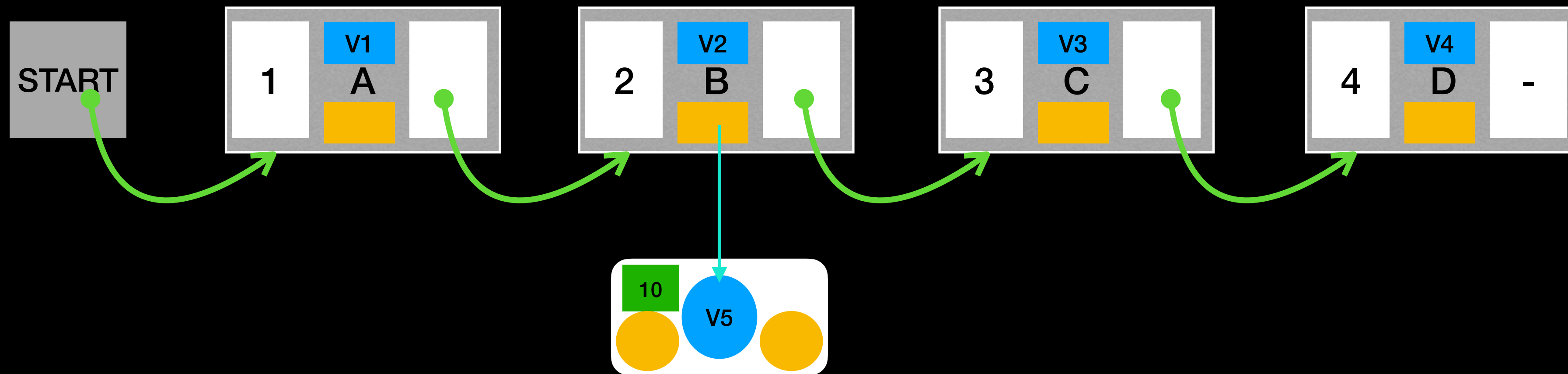
Why such approach is better ? 🥲

Normal Implementation (using balanced BST [may use just LL])



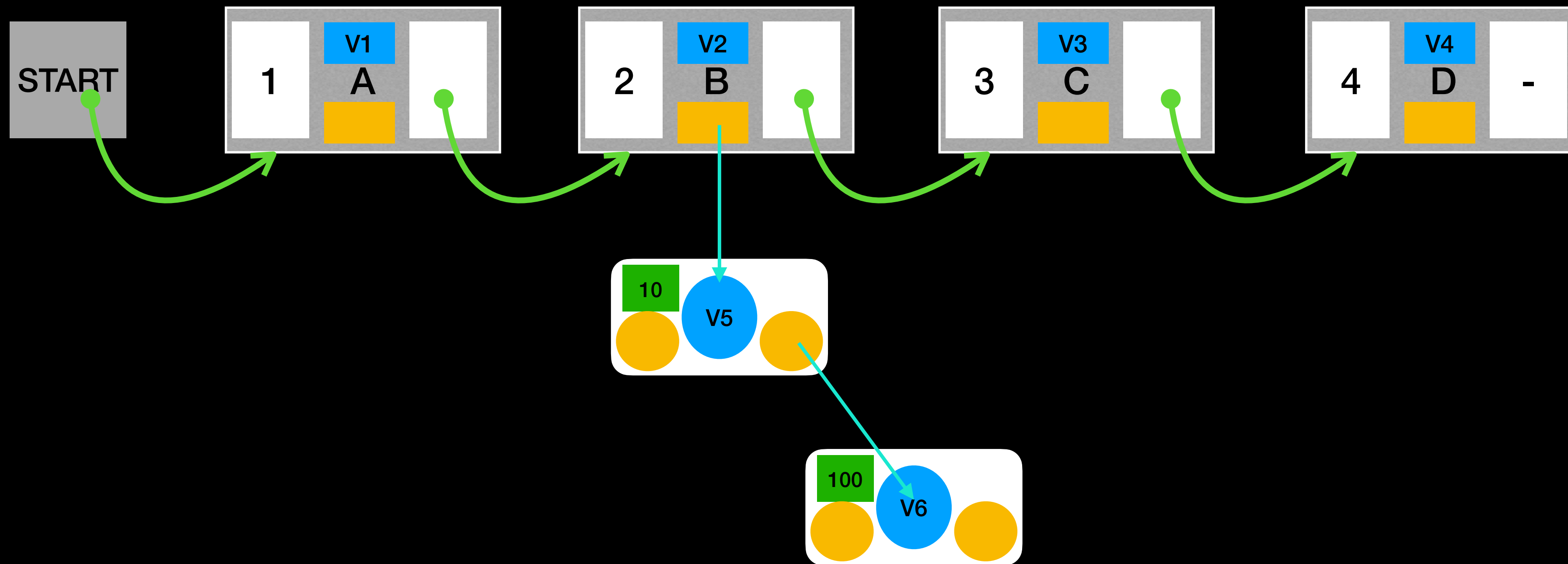
Why such approach is better ? 🥲

Normal Implementation (using balanced BST)



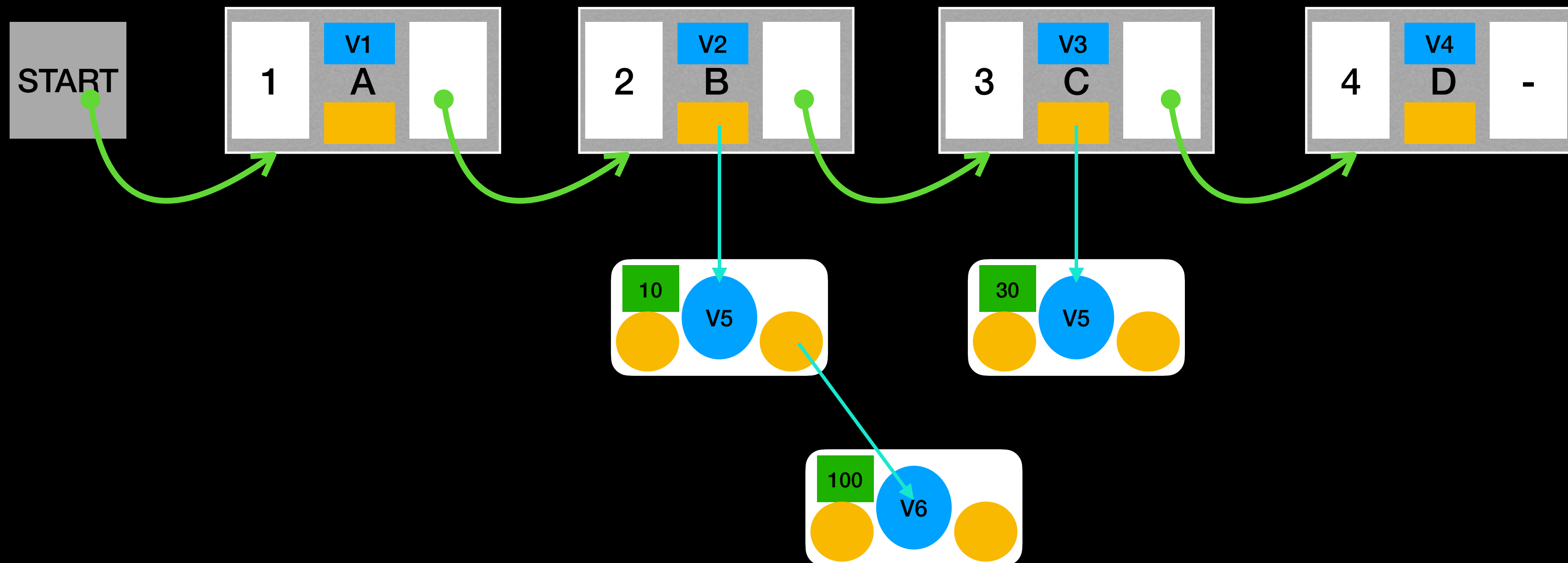
Why such approach is better ? 🥲

Normal Implementation (using balanced BST)



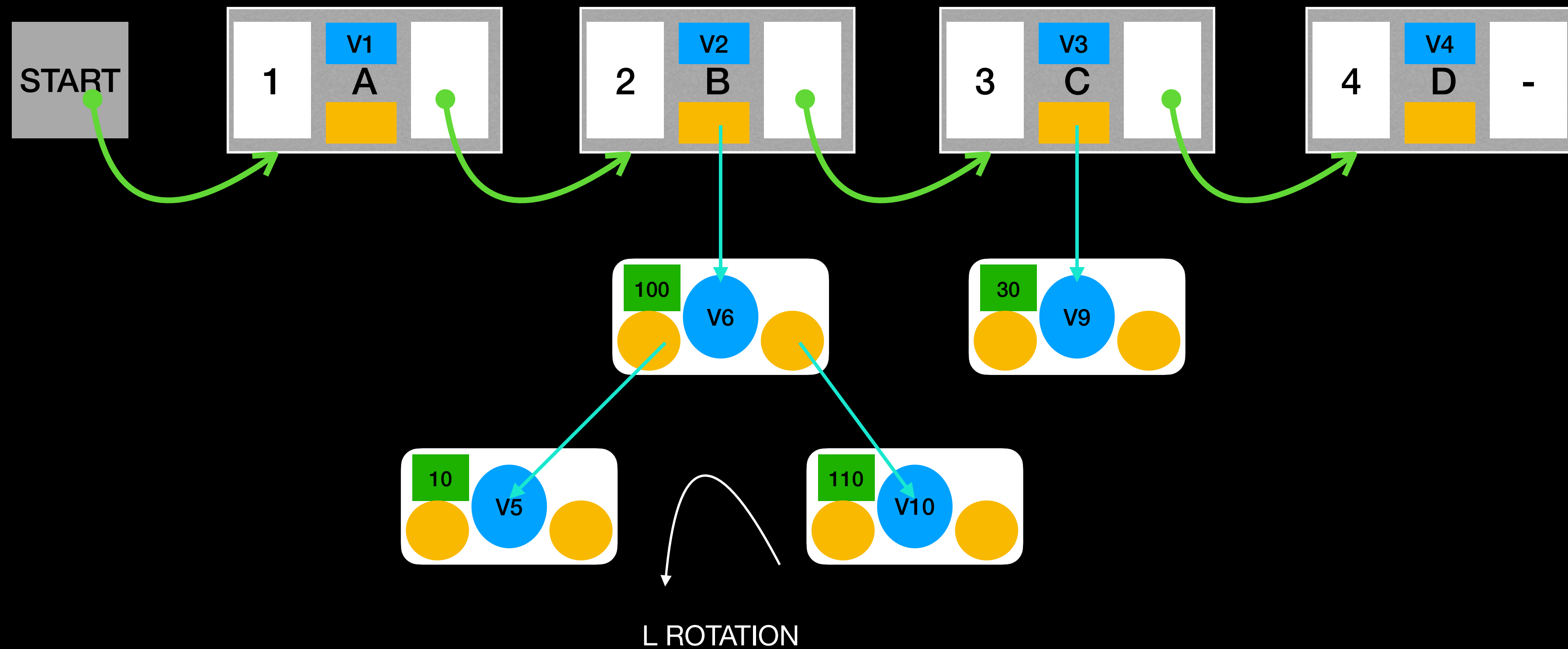
Why such approach is better ? 🥲

Normal Implementation (using balanced BST)



Why such approach is better ? 🥲

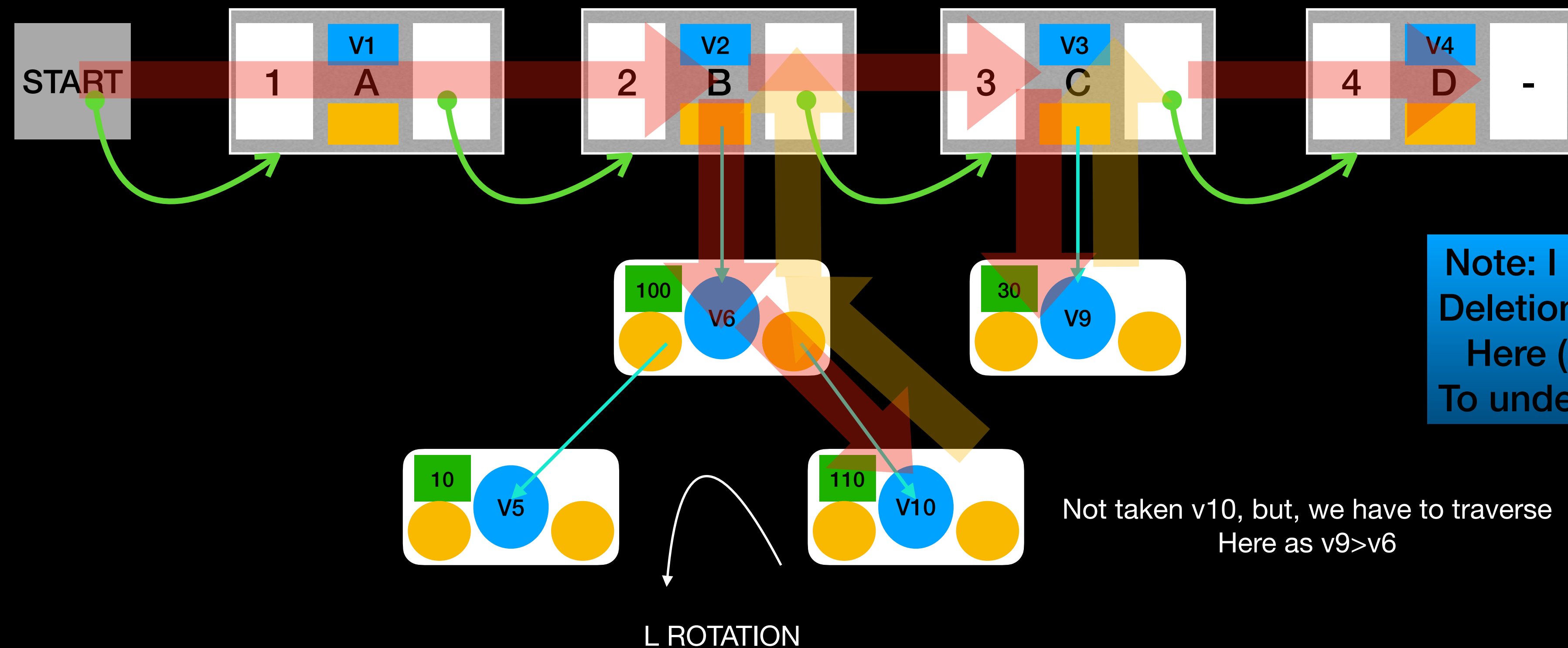
Normal Implementation (using balanced BST)



Why such approach is better ? 🥲

Normal Implementation (using balanced BST) (Reading v9)

Output: 1 -> 100 -> 30 -> 4



Note: I have not considered the Deletion and Insertion operation Here (But, this is not relevant To understand the performance)

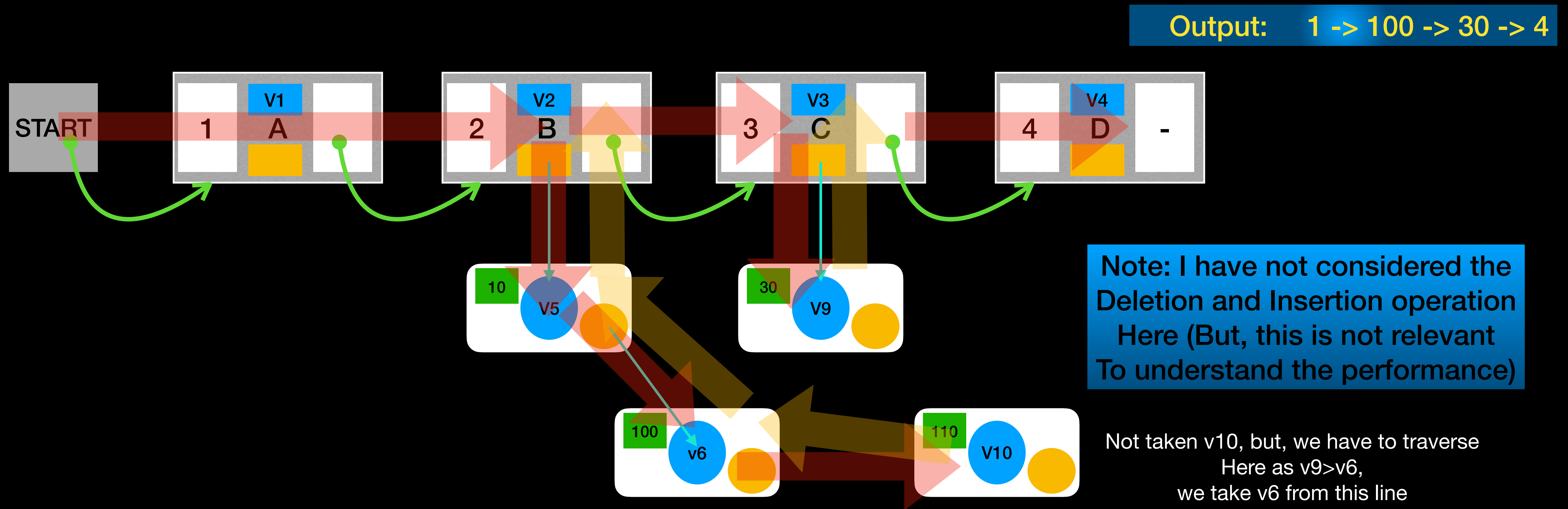
Not taken v10, but, we have to traverse Here as $v9 > v6$

So, the time complexity of reading list v9 ... $n * O(\log m)$
n is length of LL and m is #mods per node

Also the creation of the AVL of version to node
Takes $O(\log m)$ each time

Why such approach is better ? 🥲

Normal Implementation (Using List) (Reading v9)



So, the time complexity of reading list v9 ... $n * O(m)$
n is length of LL and m is is #mods per node

The creation of the List of version to node
Takes $O(1)$ each time

Why such approach is better ? 🥲

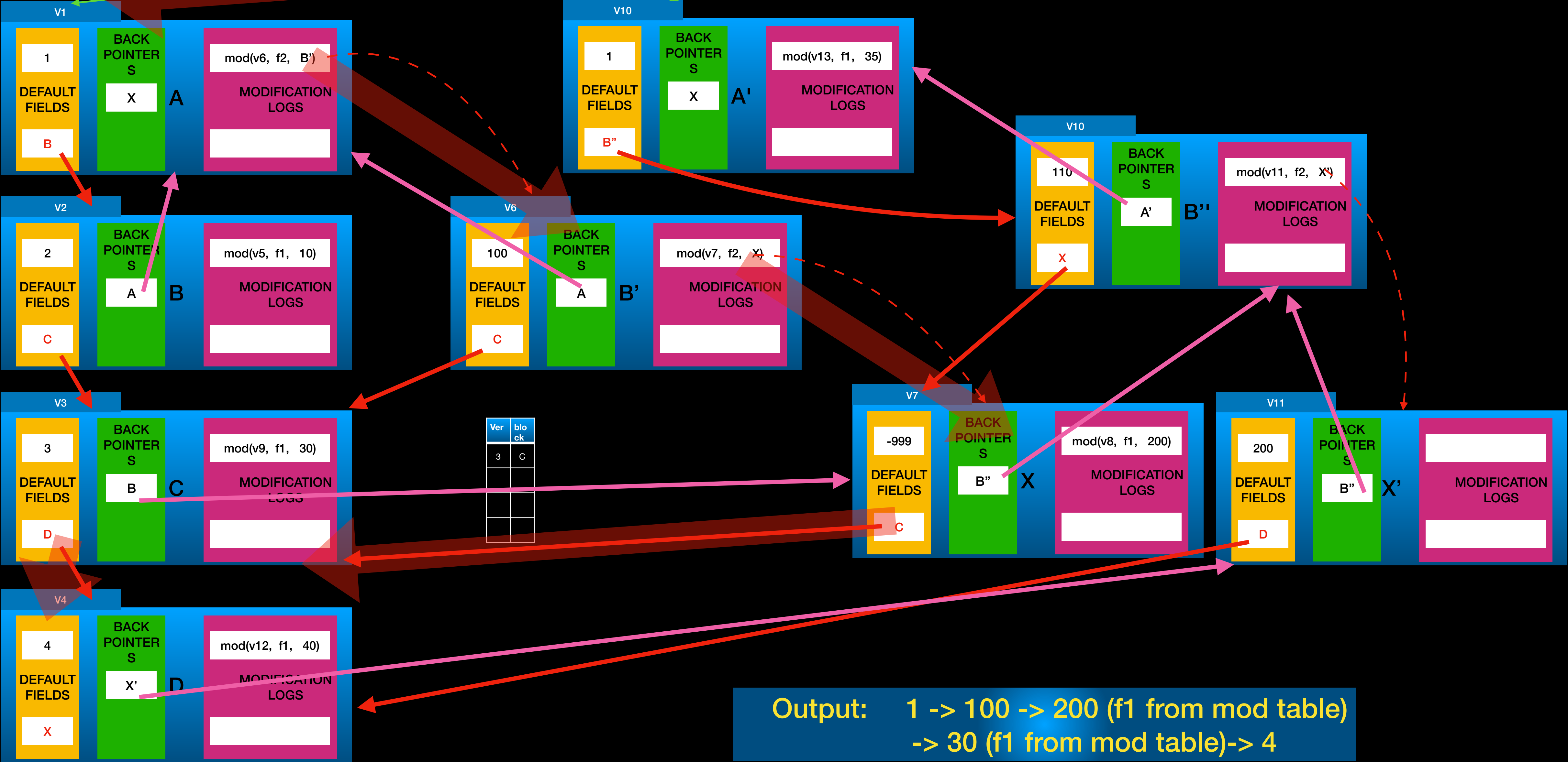
Pointer Machine (Reading v9)

We are taking $O(2k.n) \rightarrow O(n)$ time for iterating over the Linked List
Here k is the in-degree (Here 1) of the DS

We are taking $O(2)$ amortised time for modifying/adding each node to the Linked List
Here k is the in-degree (Here 1) of the DS

START MODULE

V0 V10



VERSION REDIRECTOR MODULE

Ver	block
1	A
10	A''

VERSION REDIRECTOR MODULE

Ver	block
2	B
6	B'
10	B''

VERSION REDIRECTOR MODULE

Ver	block
7	X
11	X'

VERSION REDIRECTOR MODULE

Ver	block
4	D

Output: 1 -> 100 -> 200 (f1 from mod table) -> 30 (f1 from mod table)-> 4

FINAL NOTE

START MODULE

