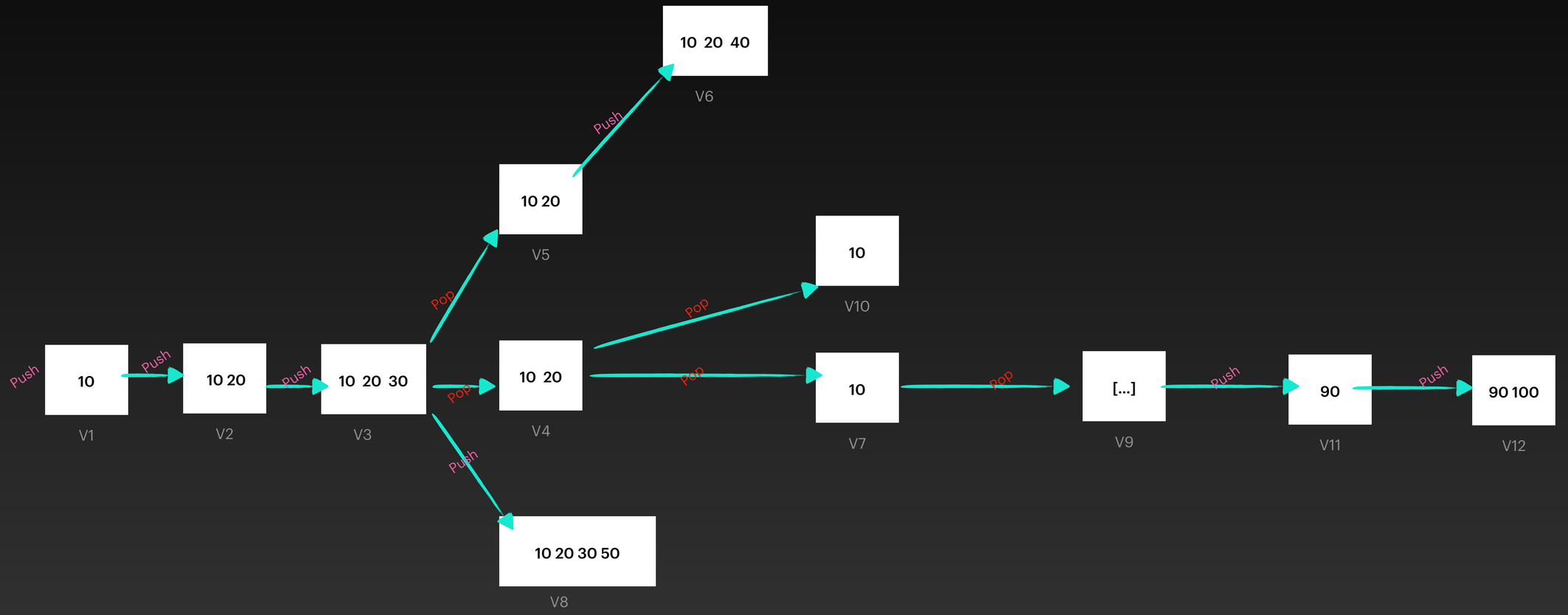
Full Persistent Stack

Using Directed Acyclic Graph

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

Ephemeral Equivalent



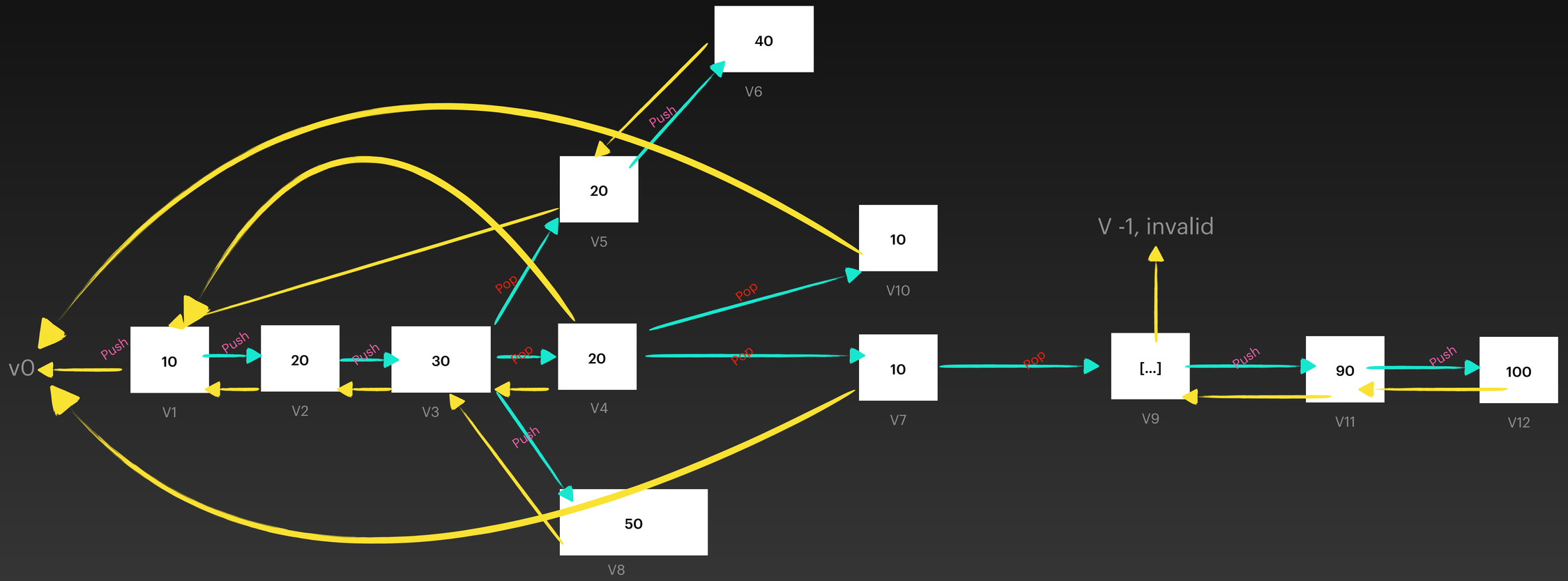
To Make It Persistent

-> First Of All ... We Have to think it as a Directed Graph

-> Here, the vertices of the Graph represent the versions and it holds the ***TOP Value*** of the stack at that version

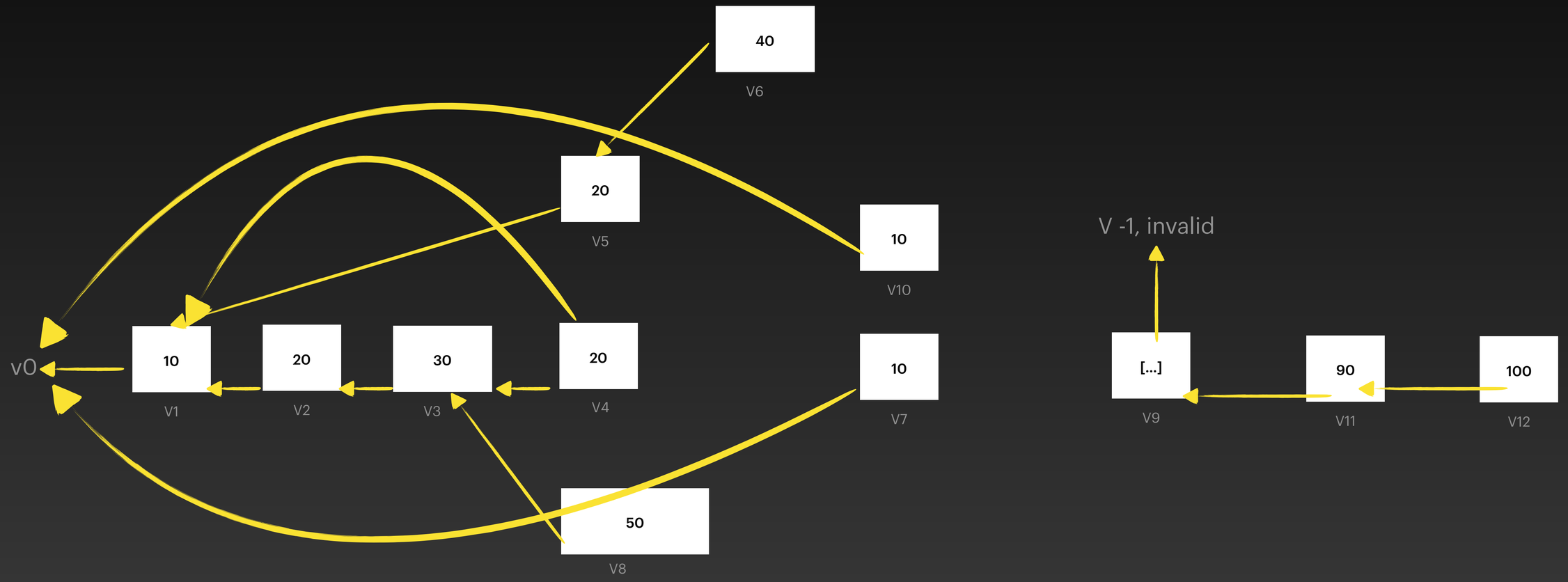
-> Here, the SKY Line shows the Forward Branching Of The Versions.

-> Here, the YELLOW Line directs the TOP at version V(say) to the NODE which is just Before TOP at version V.



We don’t need to store the SKY / Forward Lines, as Propagation through STACK happens only in backward direction

So Now It Looks Like A Directed Acyclic Graph (DAG)



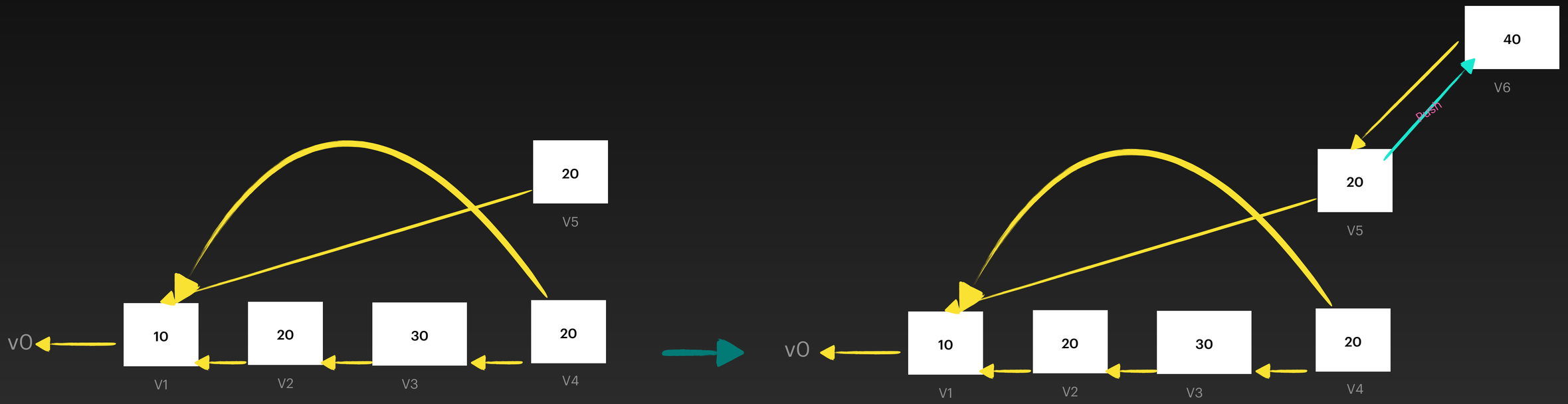
How To Push Element at any version ?

```
    // to push data at par_ver
    void push(int data, int par_ver);
```

Example: `s.push(40, 5);`

Strategy

1. Just Create A Version At V
2. Set The DATA field of version V as data
3. Set Back Pointer Of The V as par_ver



How To Pop Element at any version ?

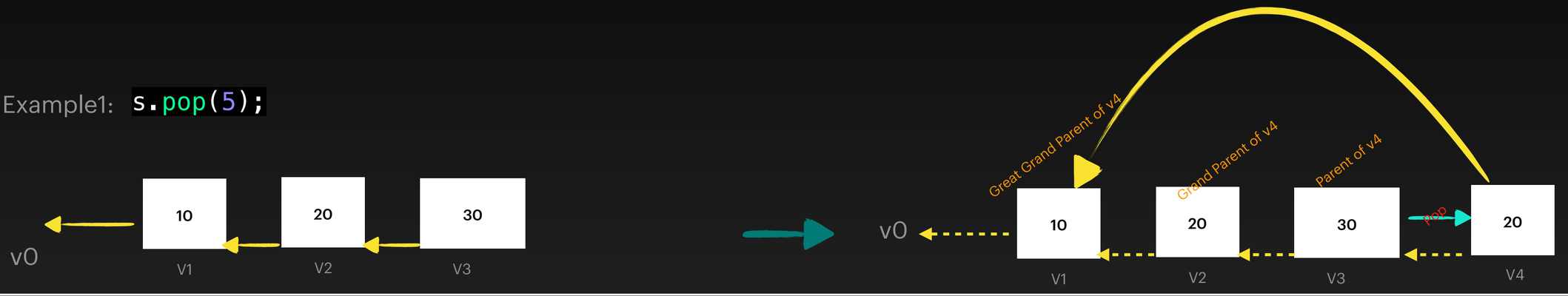
```
// to pop data at par_ver
void pop(int par_ver);
```

NOTE: HERE WE HAVE CONSIDERED PARENT OF VERSION WITH EMPTY QUEUE AS -1

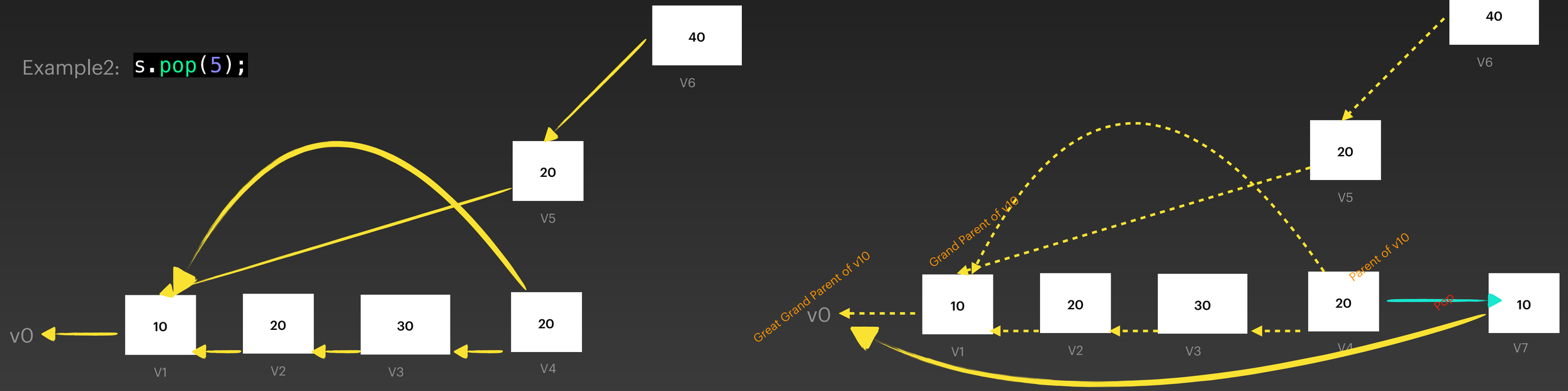
Strategy

1. Just Create A Version At V
2. Set The DATA field of version V as TOP of its grand_parent version
3. Set Back Pointer Of The V as its Great_Grand_parent version

Example1: `s.pop(5);`



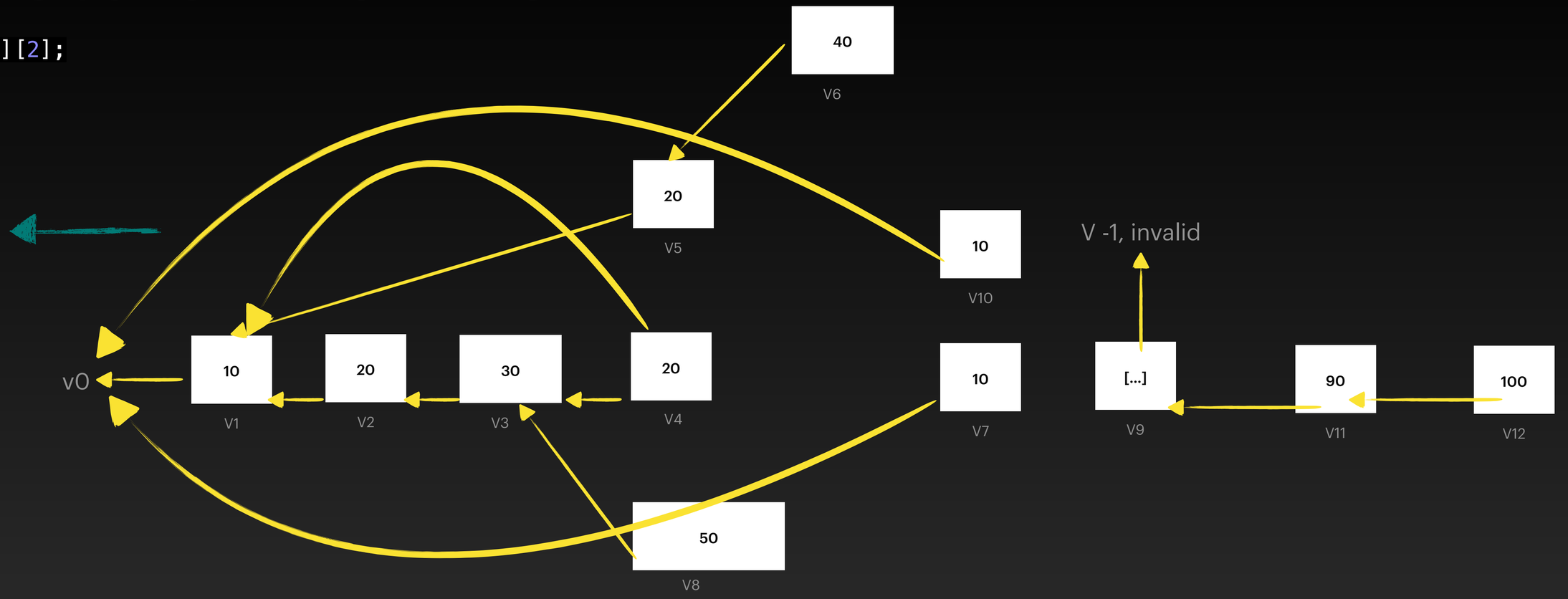
Example2: `s.pop(5);`



How to Store This DAG?

```
int MAP[MAX_NO_VER][2];
```

	TOP	BP
0	-999	-1
1	10	0
2	20	1
3	30	2
4	20	1
5	20	1
6	40	5
7	10	0
8	50	3
9	-999	-1
10	10	0
11	90	9
12	100	11



Implementation 1

Full Persistent Stack Class (shown in C++)

```
class FP_STACK
{
private:
    int MAP[MAX_NO_VER][2];
    int cur_time;

public:
    FP_STACK();
    // to push data at par_ver
    void push(int data, int par_ver);
    // to pop data at par_ver
    int pop(int par_ver);
    // to get TOP at par_ver
    int getTop(int ver);
    // to iterate from TOP to BOTTOM at par_ver
    void iterateFromTop(int ver);
    // to get current Real Time
    int getCurTime();
};

```

Implementation 2

C++ Code For PUSH

```cpp
void FP_STACK::push(int data, int par_ver)
{
    CHECK_VERSION_(par_ver);
    if(cur_time == MAX_NO_VER){
        cout<<"No Support to hold further versions in RAM ... You store them in Disk\n";
        return;
    }
    cur_time++;
    MAP[cur_time][0] = data;     // TOP
    MAP[cur_time][1] = par_ver;  // BP
}

```

C++ Code For POP

```cpp
int FP_STACK::pop(int par_ver)
{
    CHECK_VERSION(par_ver);
    if(cur_time == MAX_NO_VER){
        cout<<"No Support to hold further versions in RAM ... You store them in Disk\n";
        return SENTINEL;
    }

    // previously no element present
    if(MAP[par_ver][1]==-1){
        cout<<"Stack UnderFlowed!!\n";
        return SENTINEL;
    }

    cur_time++;
    int grand_par_ver = MAP[par_ver][1];

    // previously only element was present
    if(grand_par_ver==0){
        MAP[cur_time][0] = SENTINEL; // TOP
        MAP[cur_time][1] = -1;       // BP
        return MAP[par_ver][0];
    }

    int great_grand_par_ver = MAP[grand_par_ver][1];

    MAP[cur_time][0] = MAP[grand_par_ver][0];//TOP
    MAP[cur_time][1] = great_grand_par_ver;// BP
    return MAP[par_ver][0];
}

```

Time Complexity

- push(data,par_ver): O(1)
- pop(par_ver): O(1)
- getTop(ver): O(1)

Auxiliary Space Complexity

- O(V) to Hold The MAP