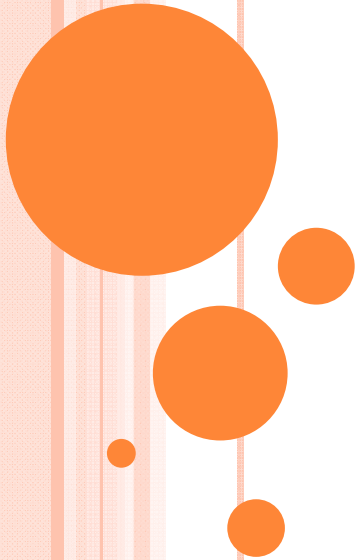


2014 EECS 22 Assignment 5

Che-Wei Chang

ASSIGNMENT 5

- A Command-line Movie Processing Program
  [100 pts + 10 bonus pts]
- Deadline : 2014/12/9, Tuesday, 11:00 pm
- Read the handout carefully.
- Xming is required to view the movie
  - Activate Xming
  - Enable Connection→ SSH→ X11→Enable X11 forwarding in Putty
- Goal
  - Create a Command-Line Movie Processing Program
  - Main function use function calls to
    [create data structure for movie]
    [input/output movie]
    [process image in the image list]
    [crop, fast forward or reverse the movie]

Command-Line Movie Processing

- Command line
  `>./MovieLab [options]`
  - Options inlcude:
    - Specifying Input / Output name
    - Specifying the number of frames to be read
    - Specifying the size of the frame in the movie
    - Specifying image processing option [bw, vflip, hmirror, edge, resize…]
    - Specifying movie processing option [crop, fast, Mandelbrot set…]
    - Showing help information for the program
    - Options are allowed to be specified in any order !!
- Example:

```
>./MovieLab -i anteater -o out -f 150 -s 352x288 –vflip

The movie file anteater.yuv has been read successfully!
Operation VFlip is done!
The movie file out.yuv has been written successfully!
150 frames are written to the file out.yuv in total
```

Command-Line Movie Processing

- > ./MovieLab -h
- Format on command line is: MovieLab [option]
- -i [file_name] to provide the input file name
- -o [file_name] to provide the output file name
- -f [no_frames] to specify the no. of frames to be read
- -s [WidthxHeight] to set resolution of the input stream (widthxheight)
- -bw to activate the conversion to black and white
- -vflip to activate vertical flip
- -hmirror to activate horizontal flip
- -edge to activate edge filter
- -poster to activate posterize filter
- -cut [Start-End] to crop the frames from frame[Start] to frame[End]
- -resize [factor] to resize the video with the provided factor [1-100]
- -fast [factor] to fast forward the video with the provided factor
- -rvs to reverse the frame order of the input stream
- -m to generate the movie with Mandelbrot sequences
- -h to show this usage information

Command-line Arguments

- `int main(int argc, char *argv[])`

- Example:
  `>./MovieLab -i anteater -o out -f 150 -s 352x288 –vflip`

  - `argc = 10`
  - `argv[0][] = ./MovieLab`
  - `argv[1][] = -i`
  - `argv[2][] = anteater`
  - `argv[3][] = -o`
  - `argv[4][] = out`
  - `argv[5][] = -f`
  - `argv[6][] = 150`
  - `argv[7][] = -s`
  - `argv[8][] = 352x288`
  - `argv[9][] = -vflip`
  - Options are allowed to be specified in any order !!

Command-line Arguments

```
int main(int argc, char *argv[]){
    int x = 0;
    char *fin = NULL, *fout = NULL;
    while(x < argc) {
       if(0 == strcmp(&argv[x][0], "-i")) {
          if(x < argc - 1) {
             fin = (char *)malloc(sizeof(char) *
                  (strlen(&argv[x + 1][0]) + strlen(".yuv") + 1));
             strcpy(fin, argv[x + 1]);
             strcat(fin, ".yuv");
          } /*fi*/
          else { /*Error Handling : if –i is followed by nothing*/
             printf("Missing argument for input name!");
             free(fin);
             free(fout);
             return 5;
          } /*else*/
          x += 2;
          continue;
      } /*fi*/
      ...
```

```
>./MovieLab -i anteater ...
```

argv[x][] argv[x+1][]

Command-line Arguments (Cont.)

- Use sscanf to read formatted data from the string
- Example: read the size of frame from the command line arguments

```
...
if(0 == strcmp(argv[x], "-s")) {
   if(x < argc - 1) {
      if (sscanf(argv[x + 1], "%dx%d", &Width, &Height) != 2) {
         /* Error Handling Here */
      }
   }
   else {
      /* Error Handling Here */
   }
   x += 2;
   continue;
} /*fi*/
...
```

Double Linked List for Movie

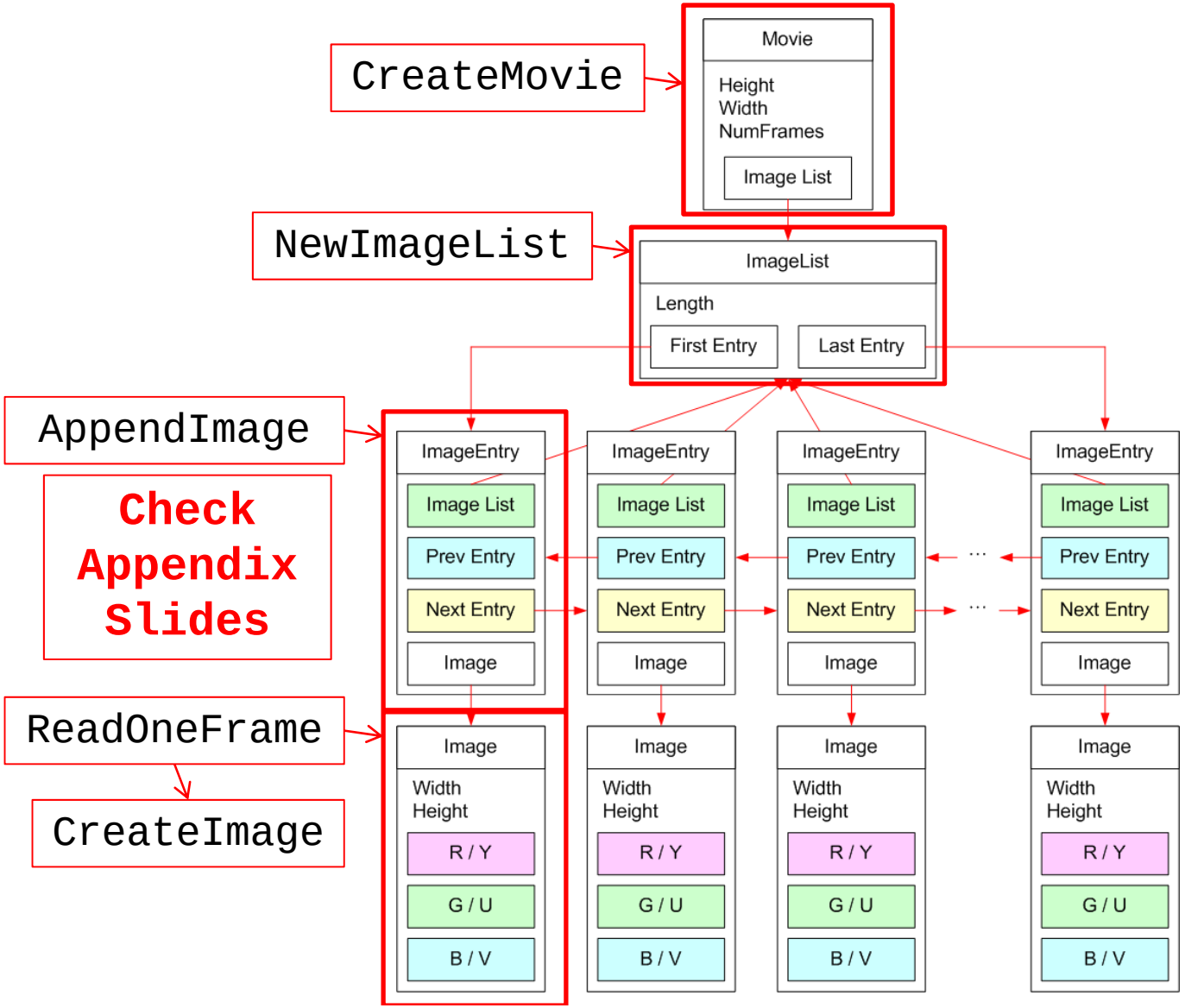


Image Processing Options Concept

- BlackNWhite, Vflip, Hmirror, Edge, Posterize, Resize
- Reuse the DIP functions defined in the previous assignment
- Traverse the list and apply the DIP function to the image in the Entry.
- List Traversal
  - Start from the first entry in the image list.
  - Use the Next pointer in current entry to find the next entry.
  - End when there is no more next entry in the list.

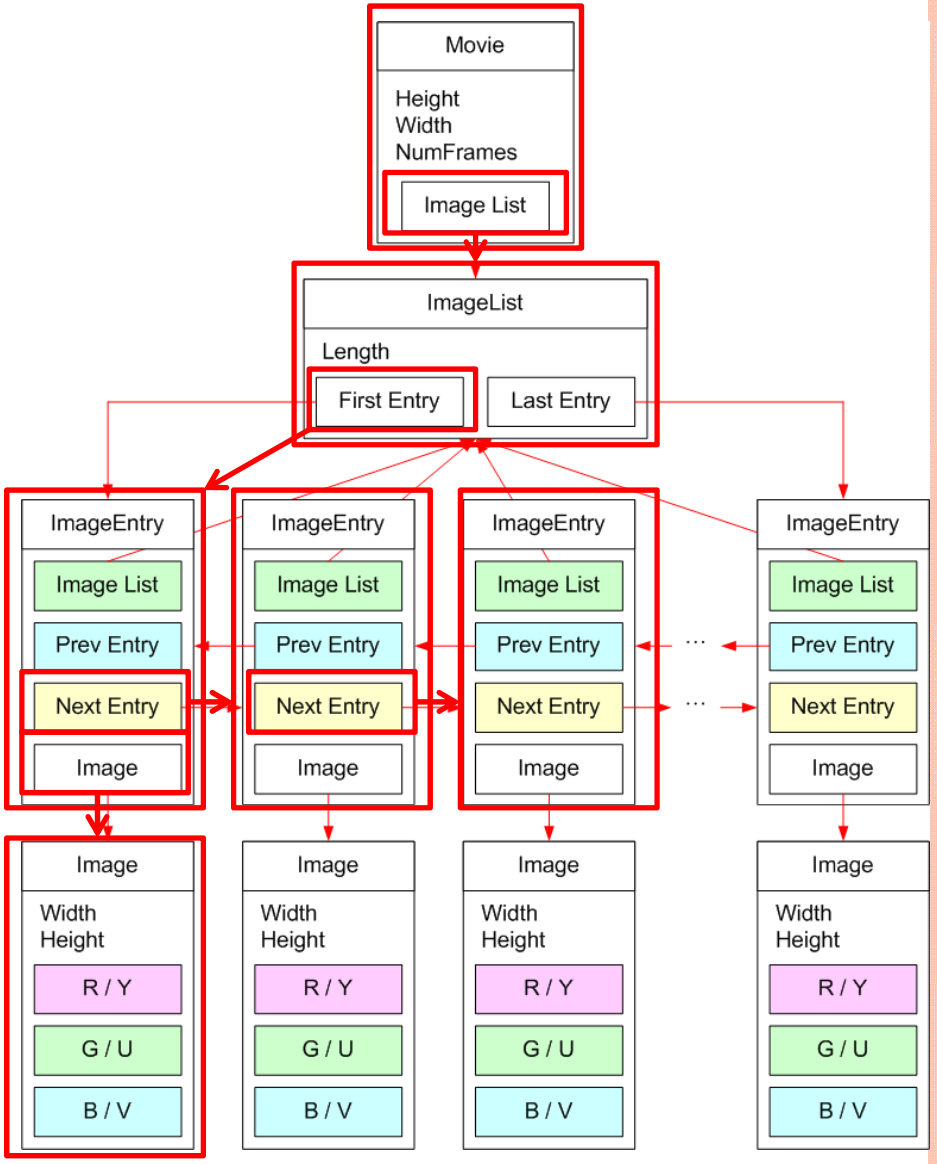
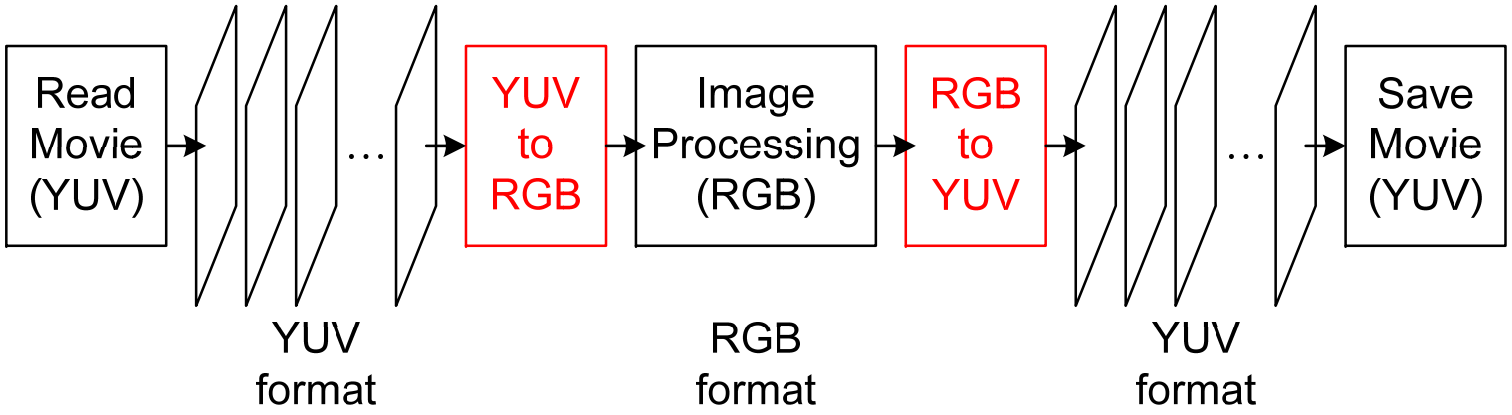


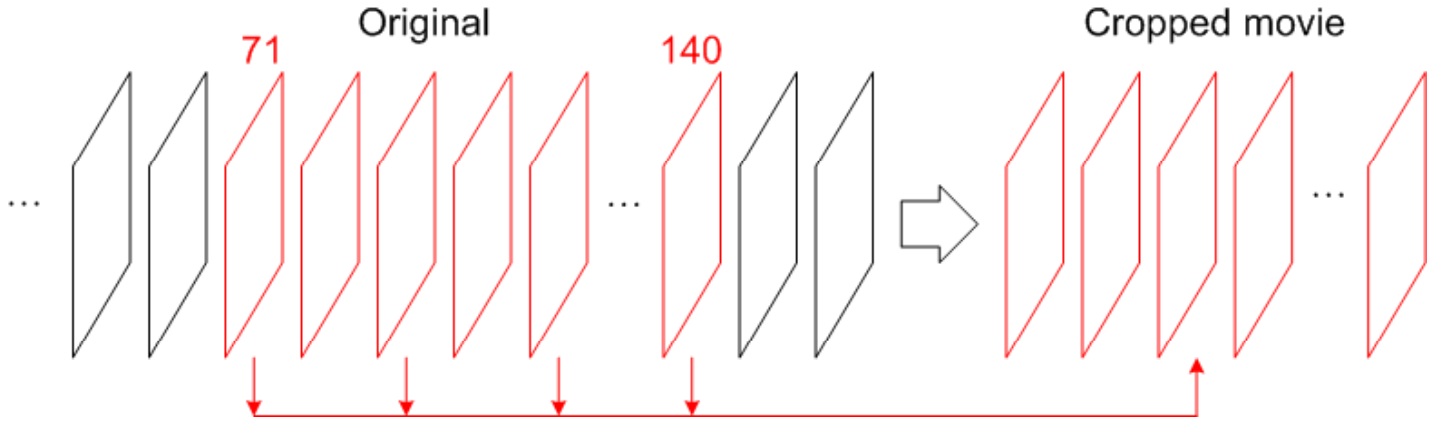
Image Processing Options
Function Pointer

- Reuse functions created in the previous assignment.
  - BlackNWhite, Horizontally Mirror, Vertically Flip, Edge, Posterize, Resize
- It is mandatory to use function pointer to perform DIP functions `bw`, `hmirror`, `vflip`, `edge`, `posterize`. (set pbits = 5 in posterize function) (See Appendix)
- YUV ←→ RGB conversion



Movie Processing Option: Cropping

- Goal:
  For a given start frame s and end frame e, creating a cropped movie by taking frame from s to e from the original movie.

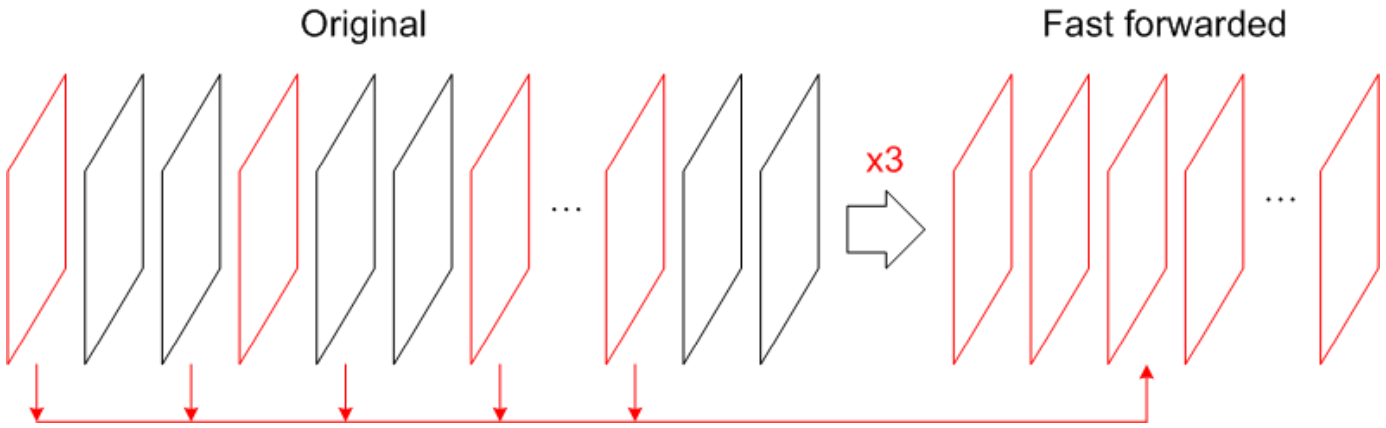


- `./MovieLab -i anteater -o out -f 150 -s 352x288 -cut 71-140`

```
The movie file anteater.yuv has been read successfully!
Operation Fast Forward is done! Number of frames = 70
The movie file out.yuv has been written successfully!
70 frames are written to the file out.yuv in total
```

Movie Processing Option: Fast Forwarding

- Goal:
  For a given fast forwarding factor n, creating a fast forwarded movie by taking every n -th frames and generating a shortened movie.



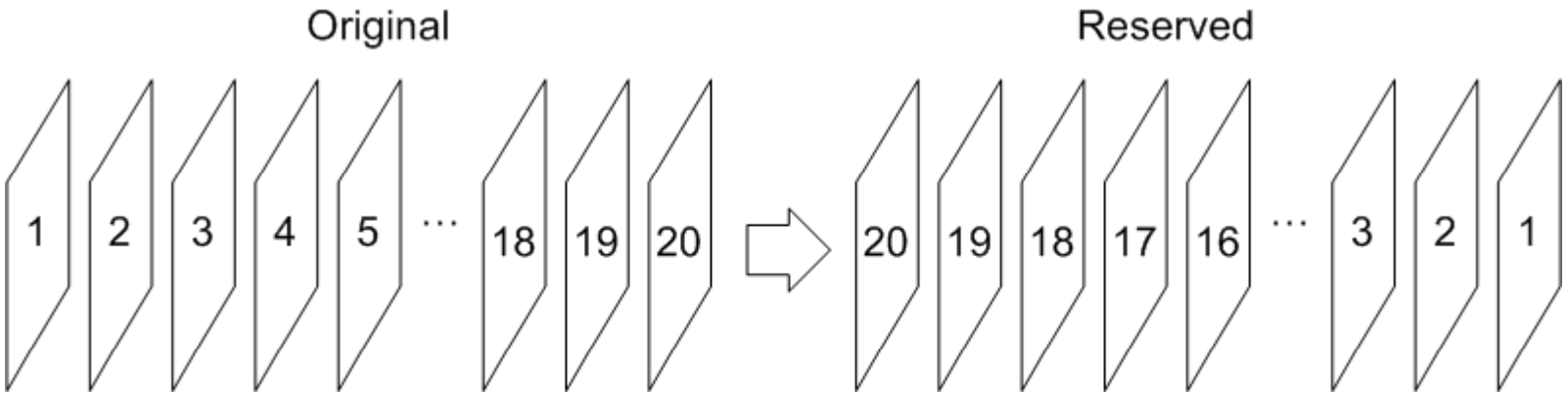
- `./MovieLab -i anteater -o out -f 150 -s 352x288 -fast 3`

```
The movie file anteater.yuv has been read successfully!
Operation Fast Forward is done! Number of frames = 50
The movie file out.yuv has been written successfully!
50 frames are written to the file out.yuv in total
```

Movie Processing Option: Reverse

Check Appendix Slides

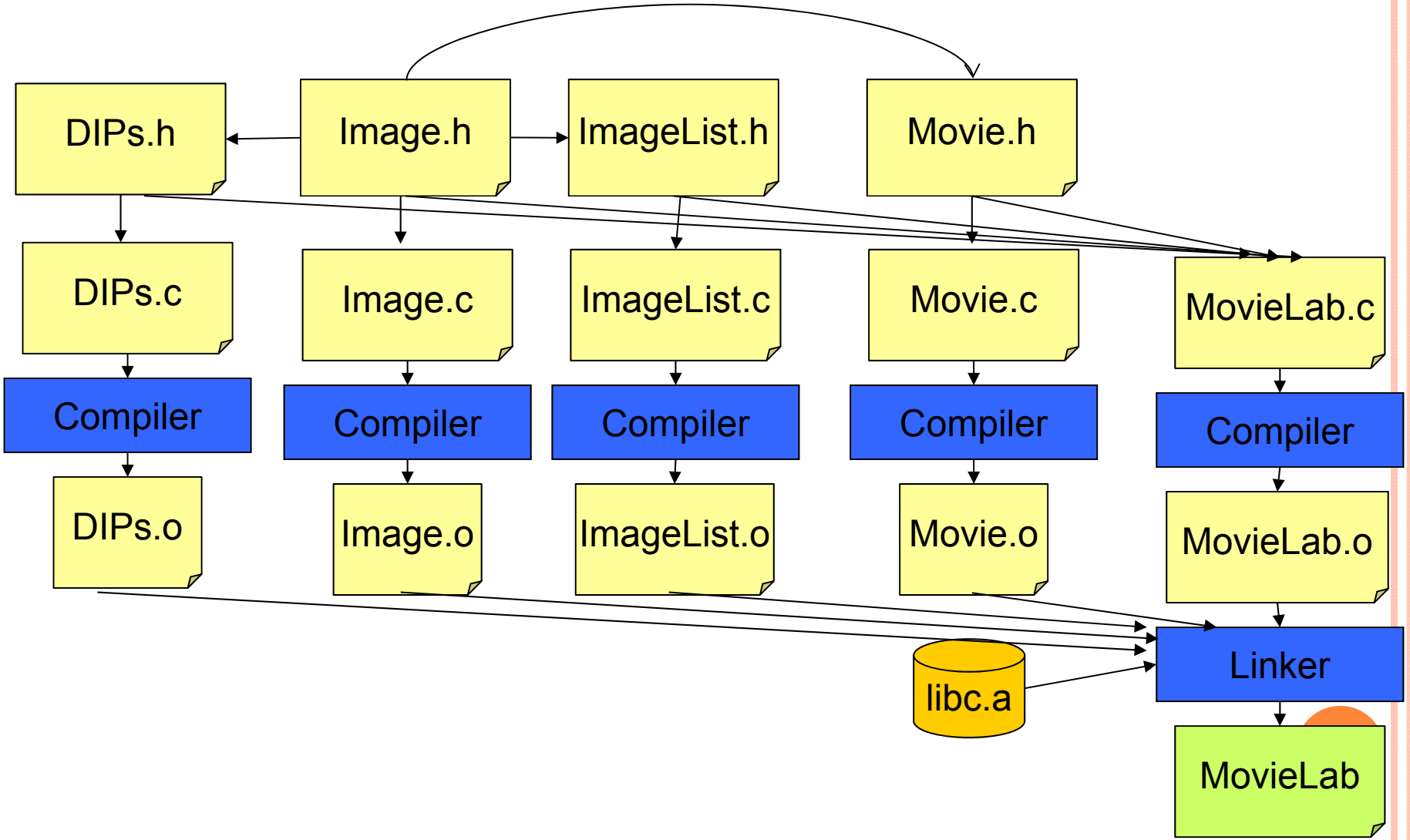
- Goal:
  Manipulate the double-linked list and save the frames to the output movie in the reverse order.



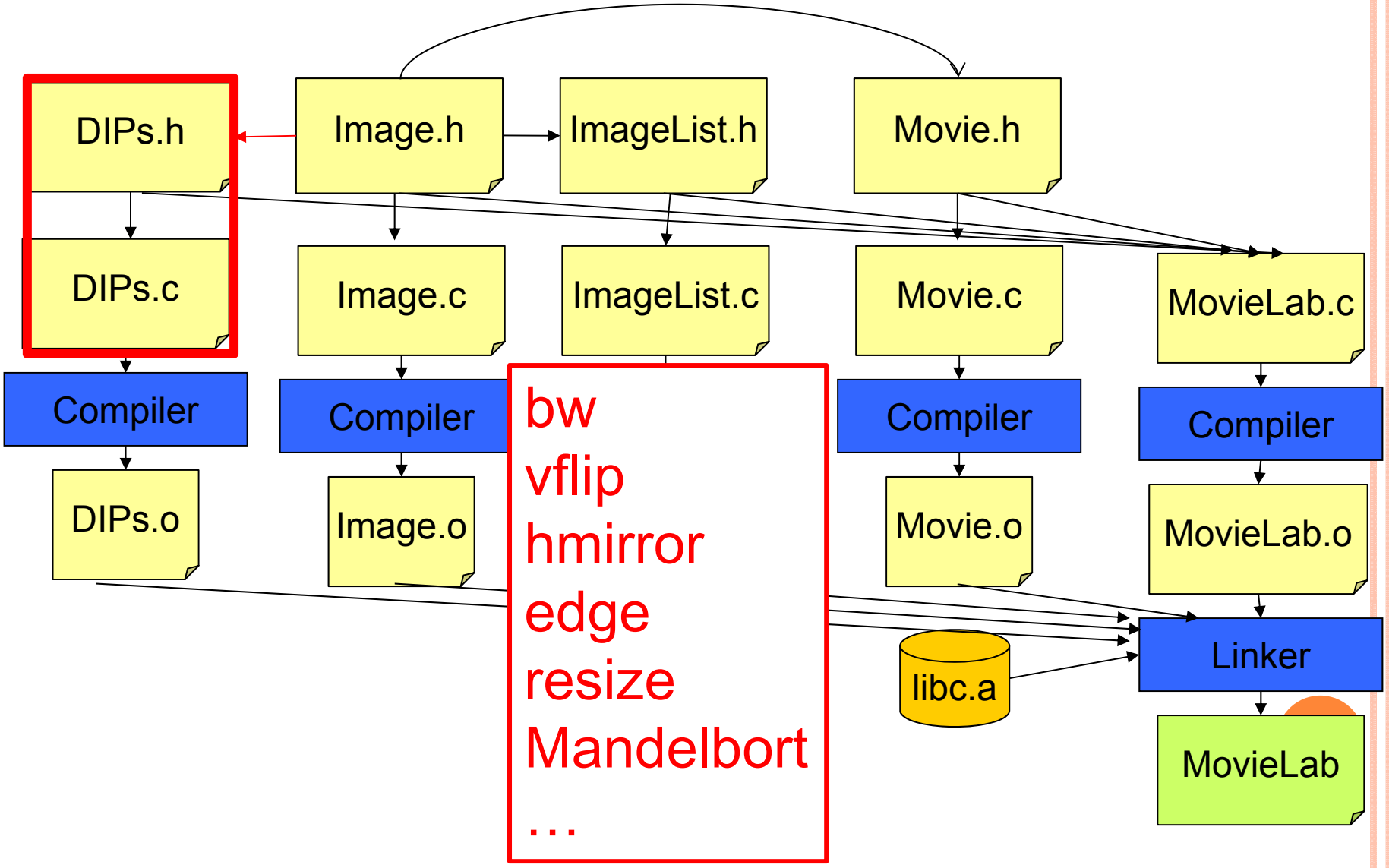
- `./MovieLab -i anteater -o out -f 150 -s 352x288 -rvs`

```
The movie file anteater.yuv has been read successfully!
Operation Reverse is done!
The movie file out.yuv has been written successfully!
150 frames are written to the file out.yuv in total
```

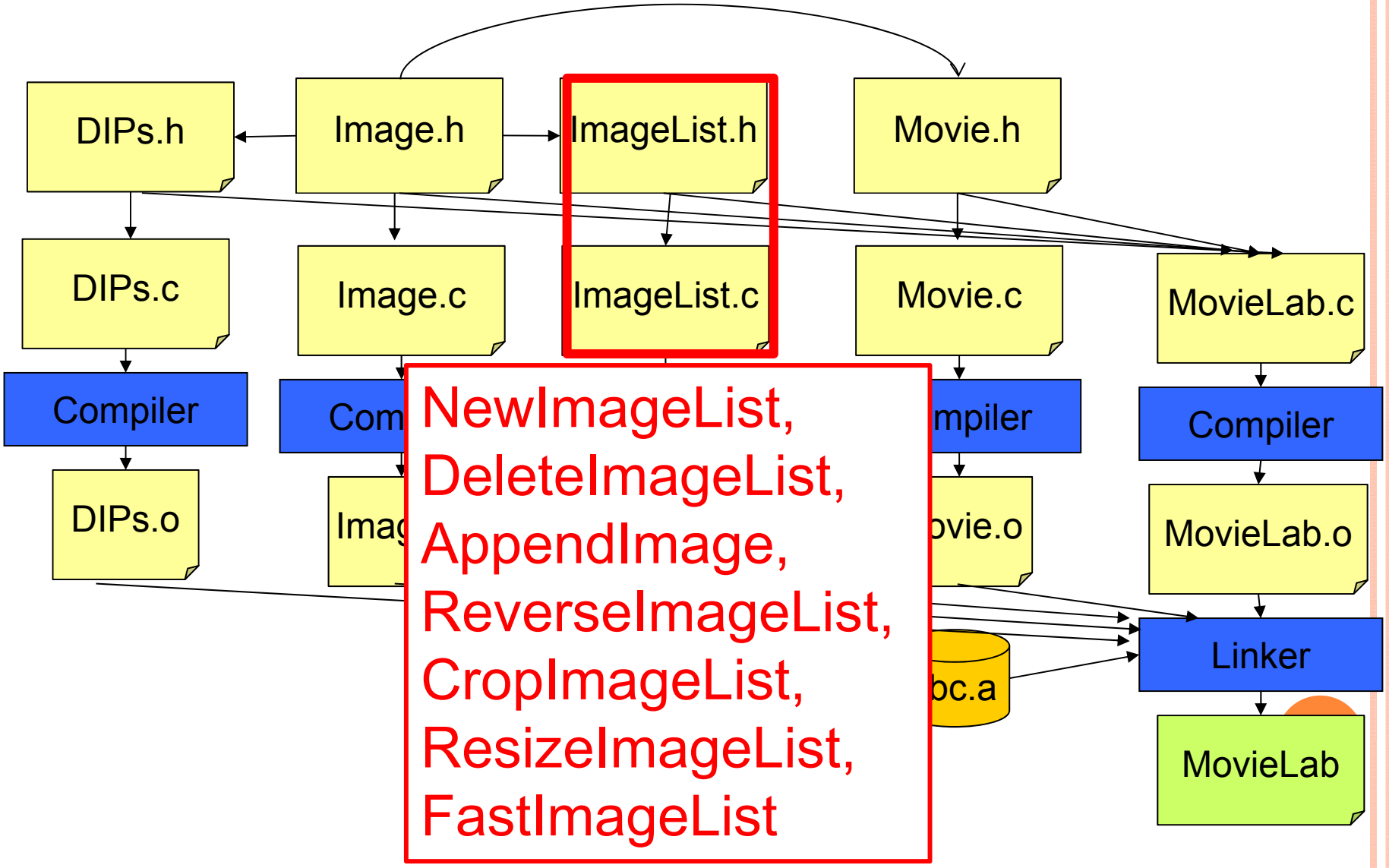
MovieLab Modules



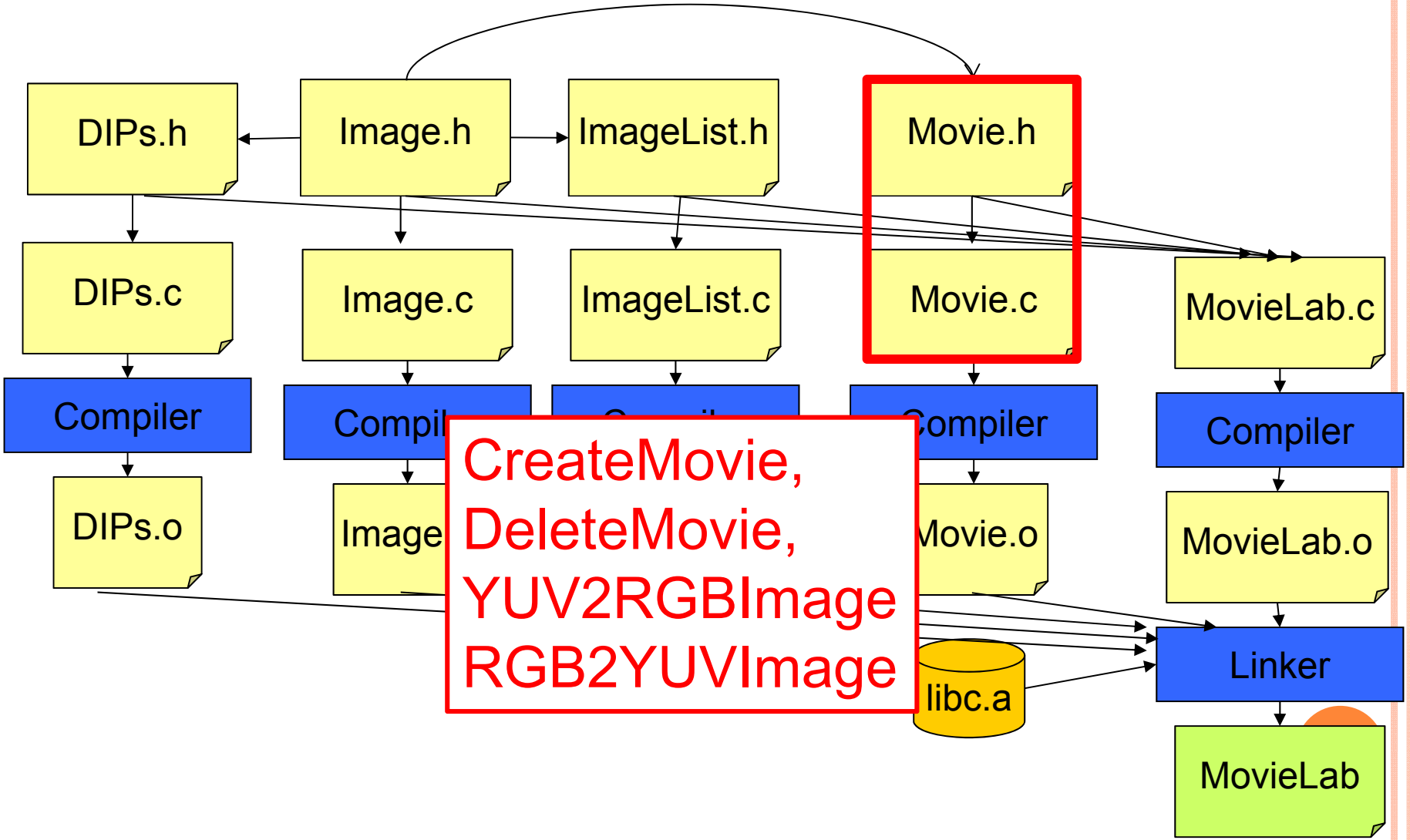
MovieLab Modules



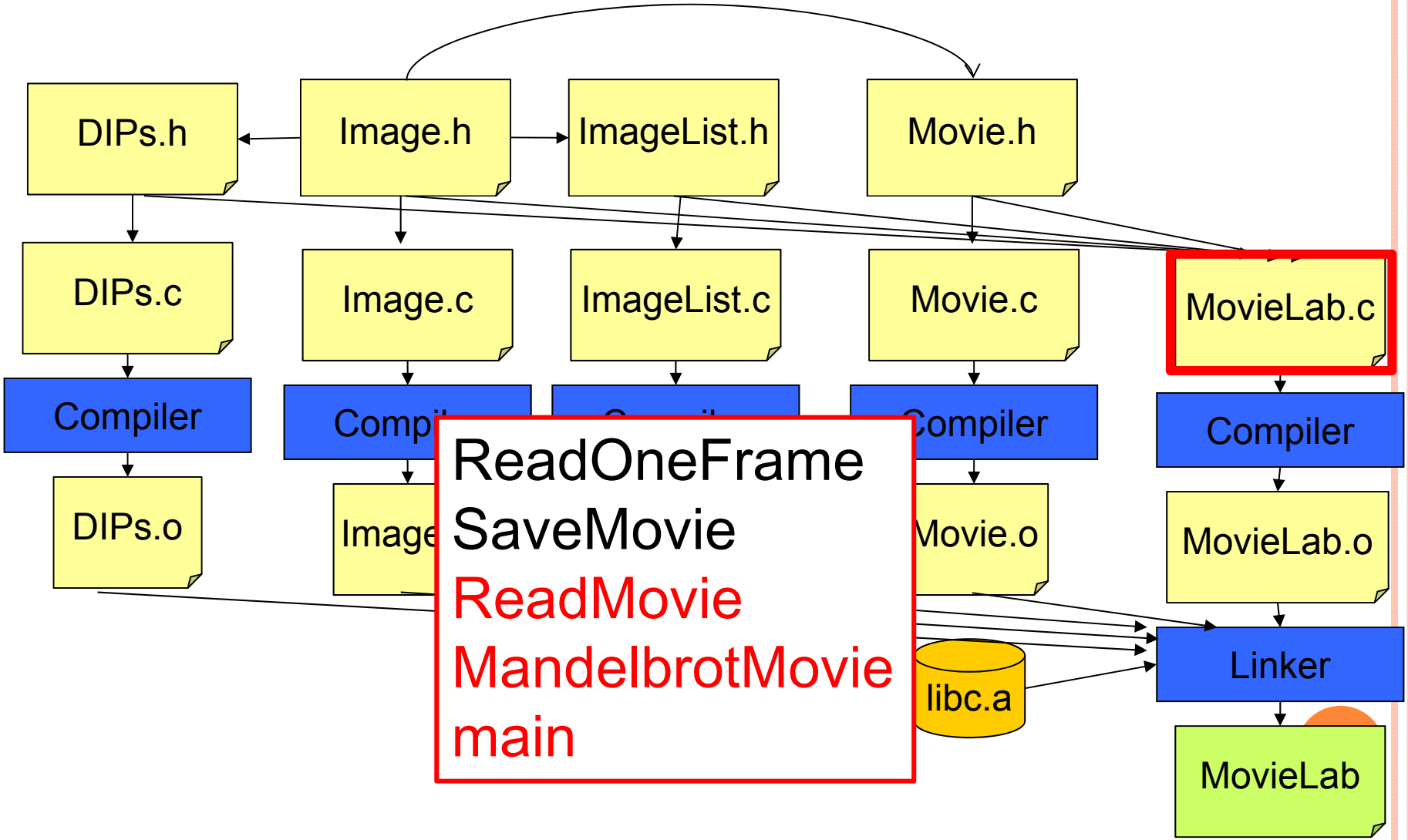
MovieLab Modules



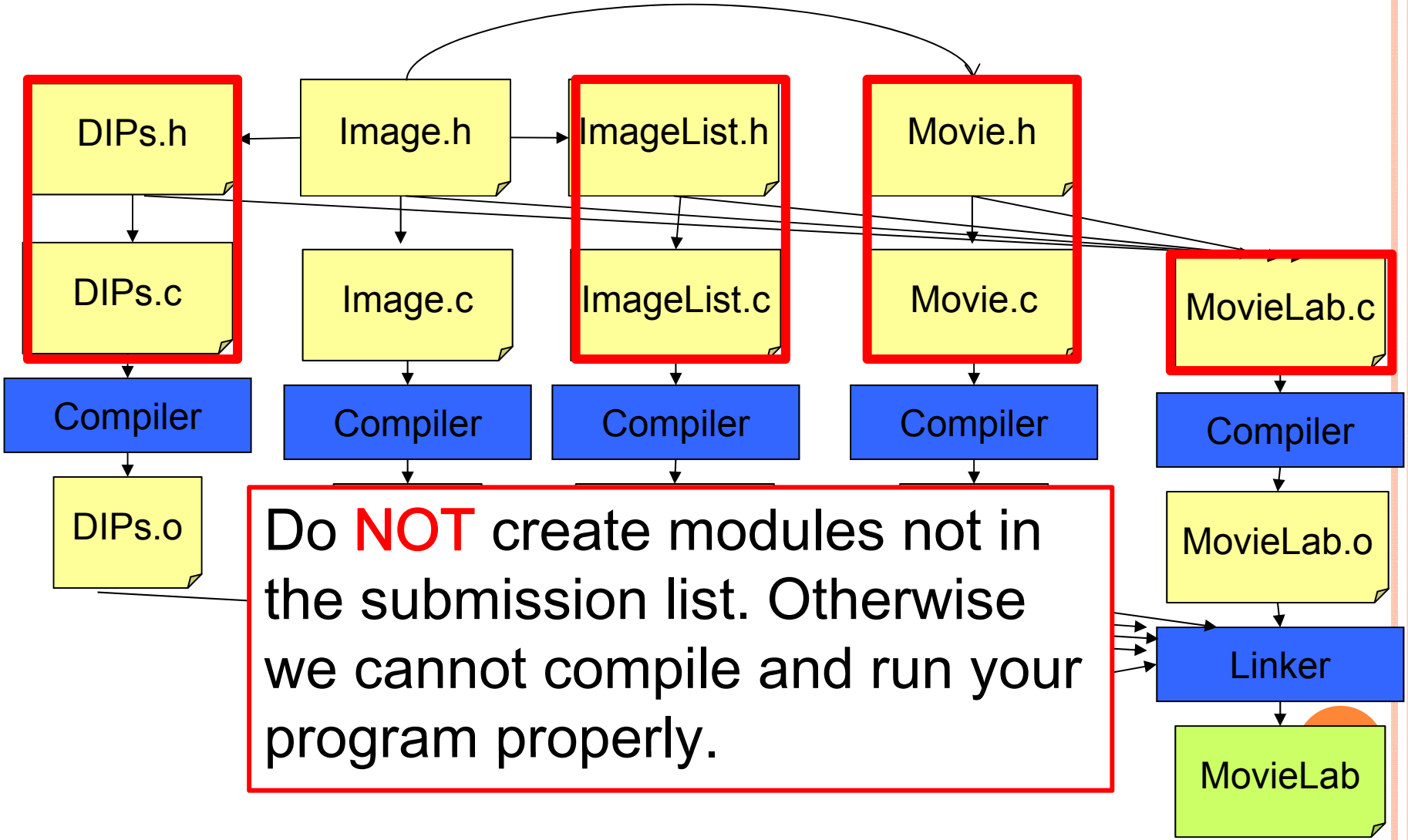
MovieLab Modules



MovieLab Modules



MovieLab Modules



Budgeting Your Time

- Phase 1:
  - Design the ImageList modules
  - Design the Movie modules
  - Read the movie(s) into the program and save the cropped/concatenated/fast-forwarded movie to the output
  - Build the Makefile
- Phase 2:
  - Design the MovieLab.c
  - Add the command-line argument in the main function
  - Add the image processing option(s) to the program
  - Use Valgrind to check memory usage
  - Script the result and submit your work.

APPENDICES

- Example of Function Pointer
- Building and Deleting a Double-Linked List
- Reverse a Double-Linked List

Image Processing Options
Function Pointer

- Example:
  - Processing all elements in an array with Add, Sub, and Mul function.

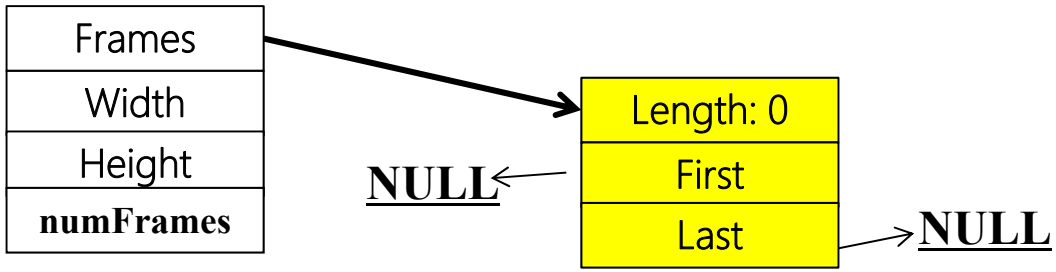
```
int add (int in1, int in2) { return in1 + in2 ; }
 int mul (int in1, int in2) { return in1 * in2 ; }
 int sub (int in1, int in2) { return in1 - in2 ; }

 typedef int MOP_F(int, int) ;
 void MathOperations(MOP_F *MathOp, int in1[3], int in2[3])
 {  int i ;
    for (i = 0; i < 3; i++)
      { in1[i] = MathOp (in1[i], in2[i]); }
 }
```

Image Processing Options Function Pointer

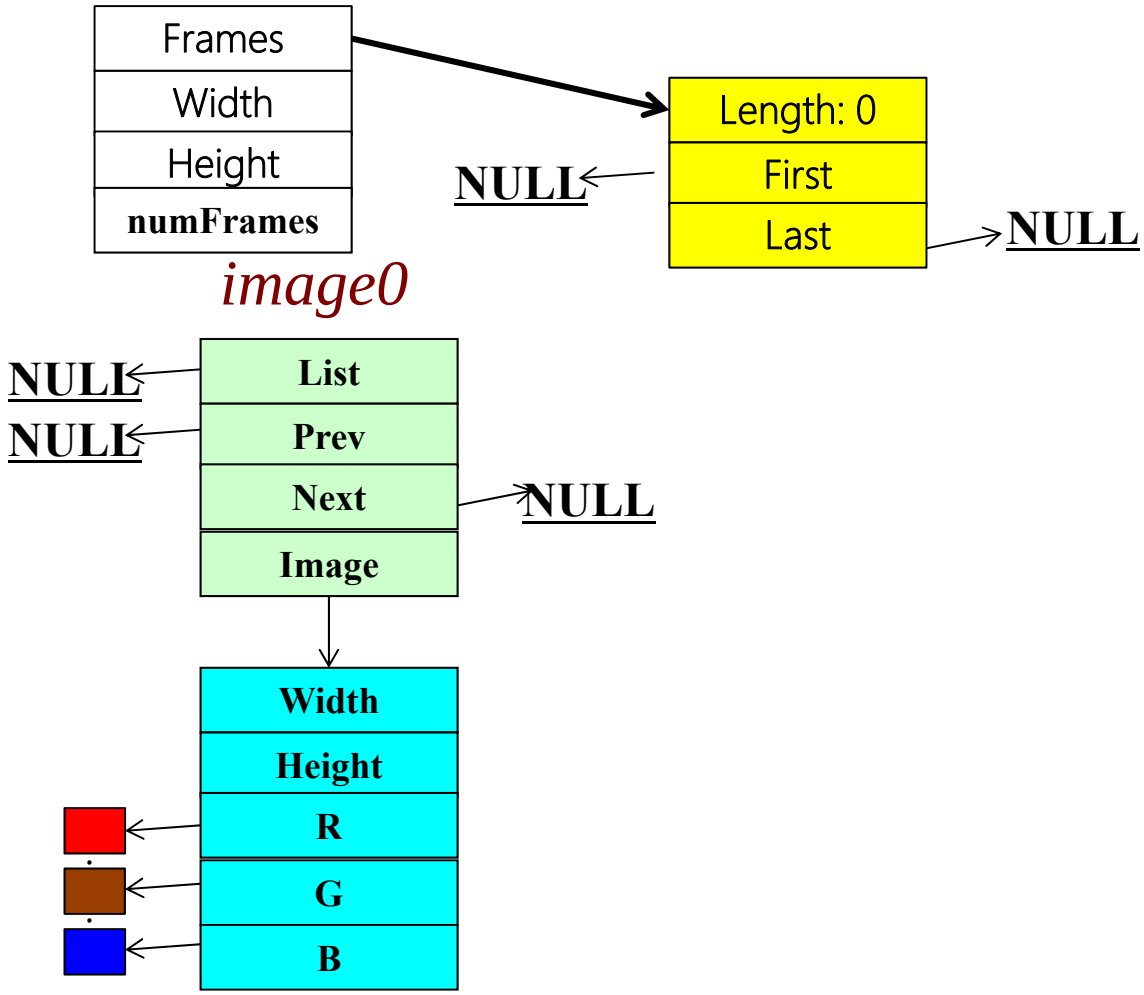
- Example (continue)

```
int main (void)
{  int op, in1[3], in2[3] ;
   MOP_F* func_ptr ;
   scanf ... /*read option and data in array*/
   switch (op){
     case 0 : func_ptr = &add ; break ;
     case 1 : func_ptr = &sub ; break ;
     case 2 : func_ptr = &mul ; break ;
     default : break;
   }
   MathOperations (func_ptr, in1, in2);
   printf ... /*print results*/
   return 0 ;
}
```

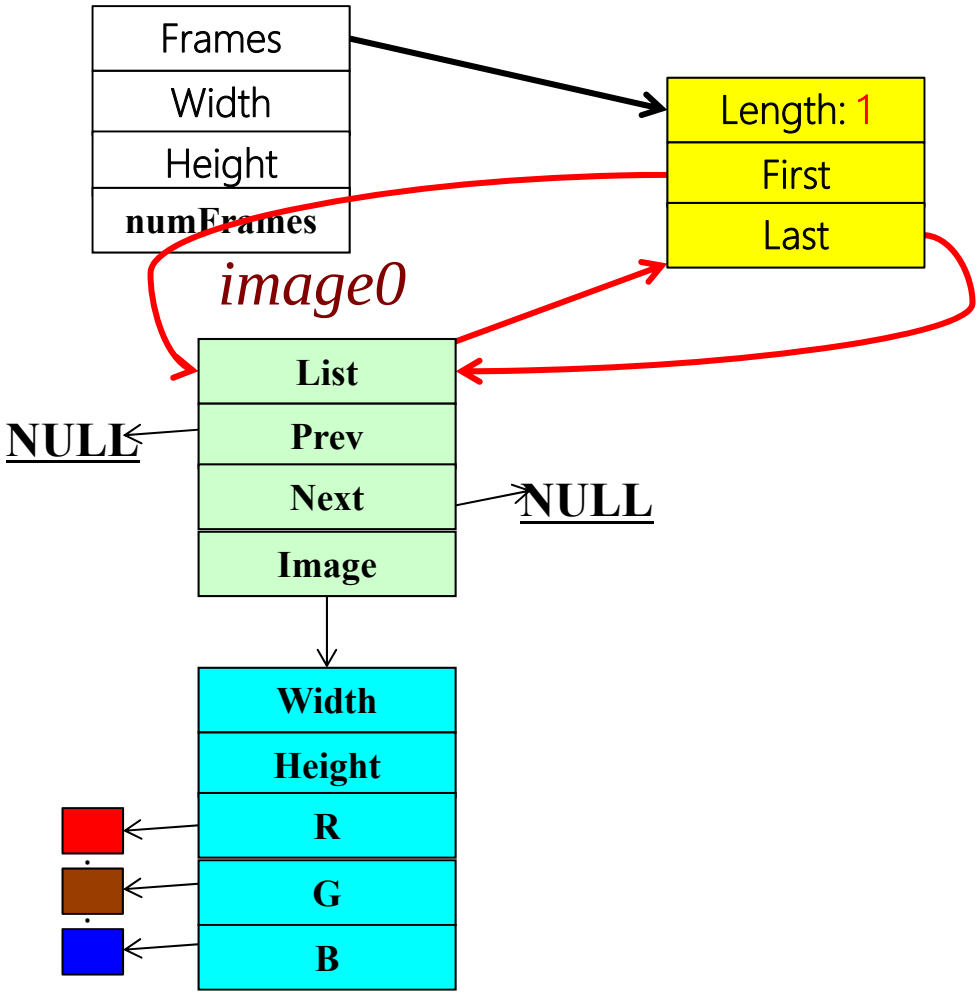
DOUBLE-LINKED LIST: EMPTY



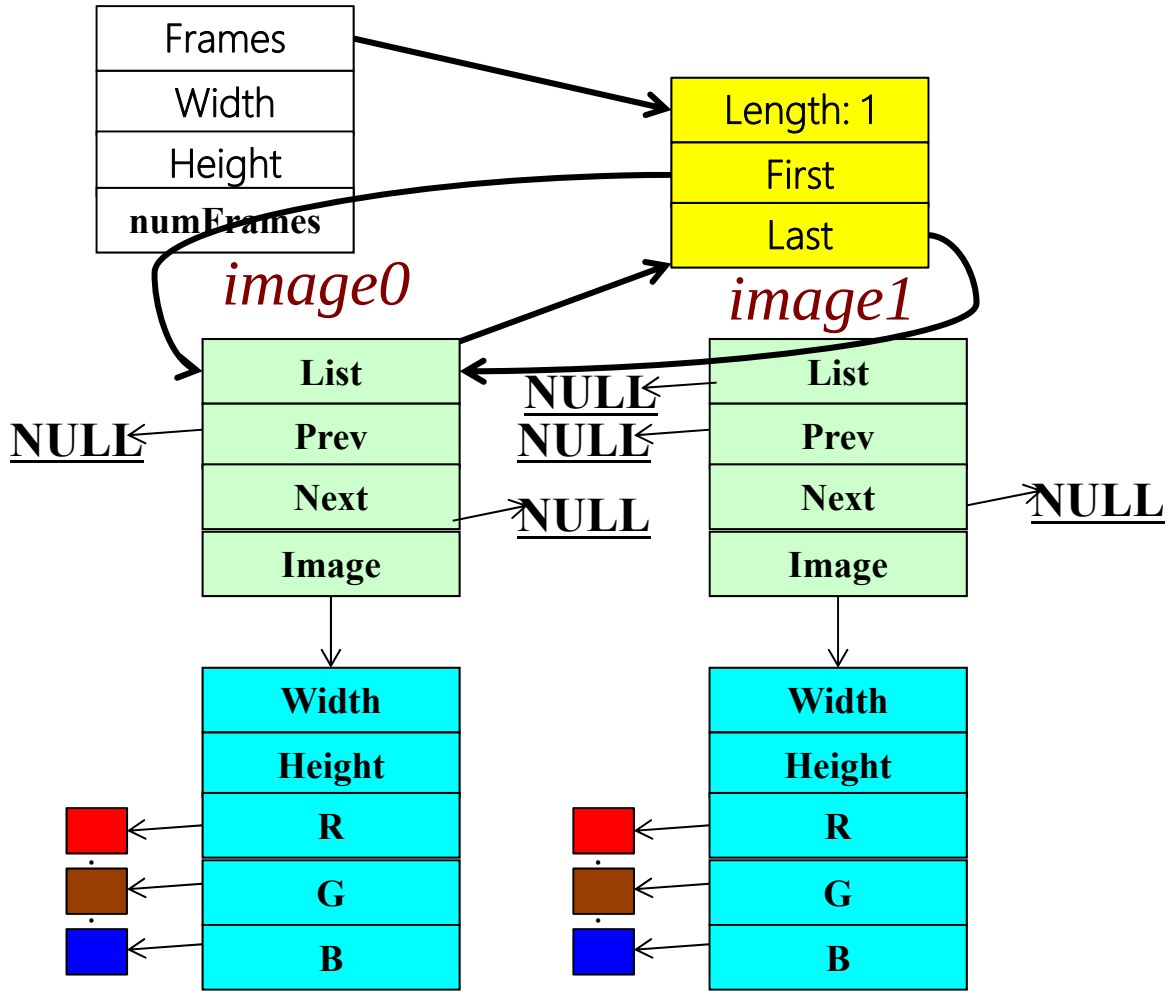
DOUBLE-LINKED LIST: APPEND



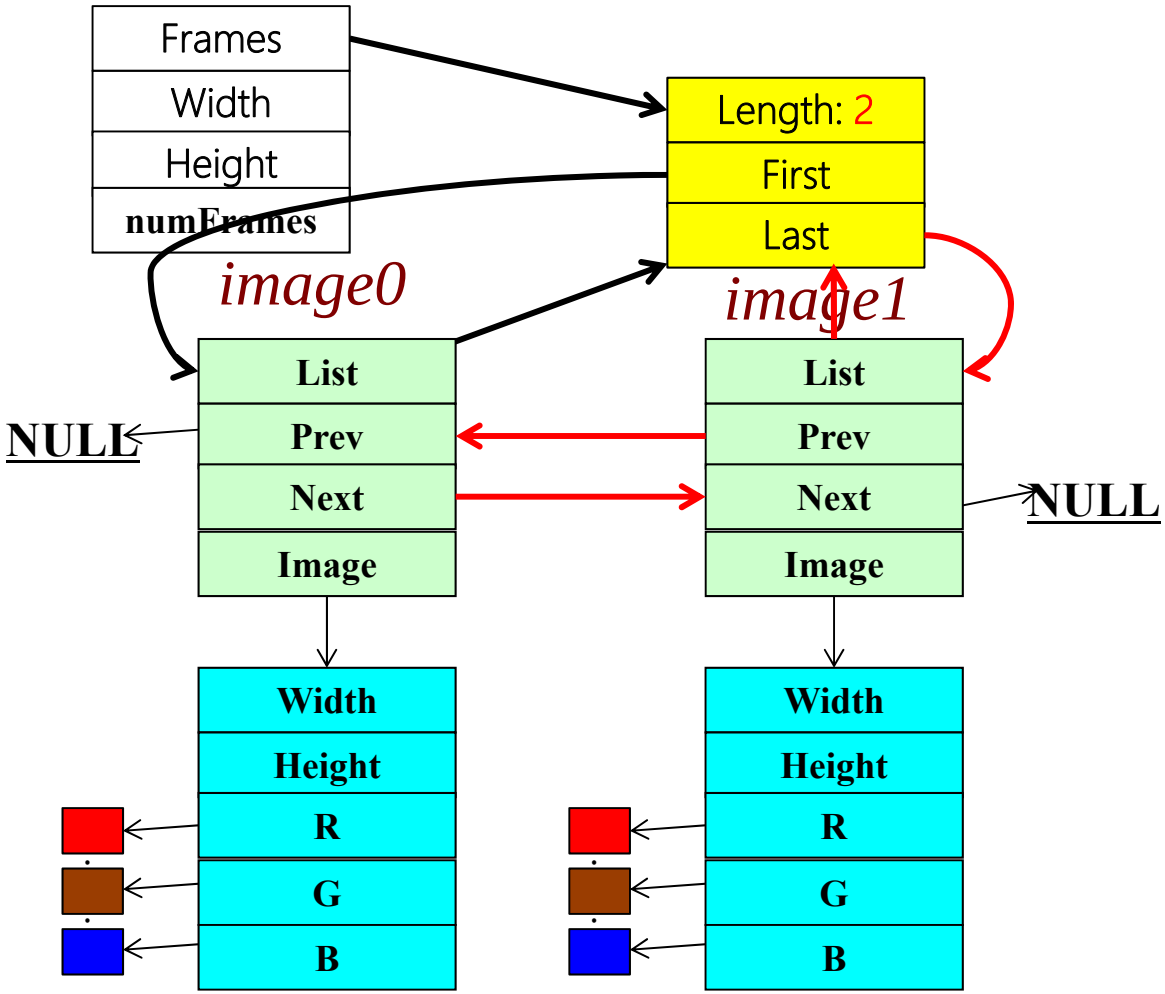
DOUBLE-LINKED LIST: LENGTH = 1



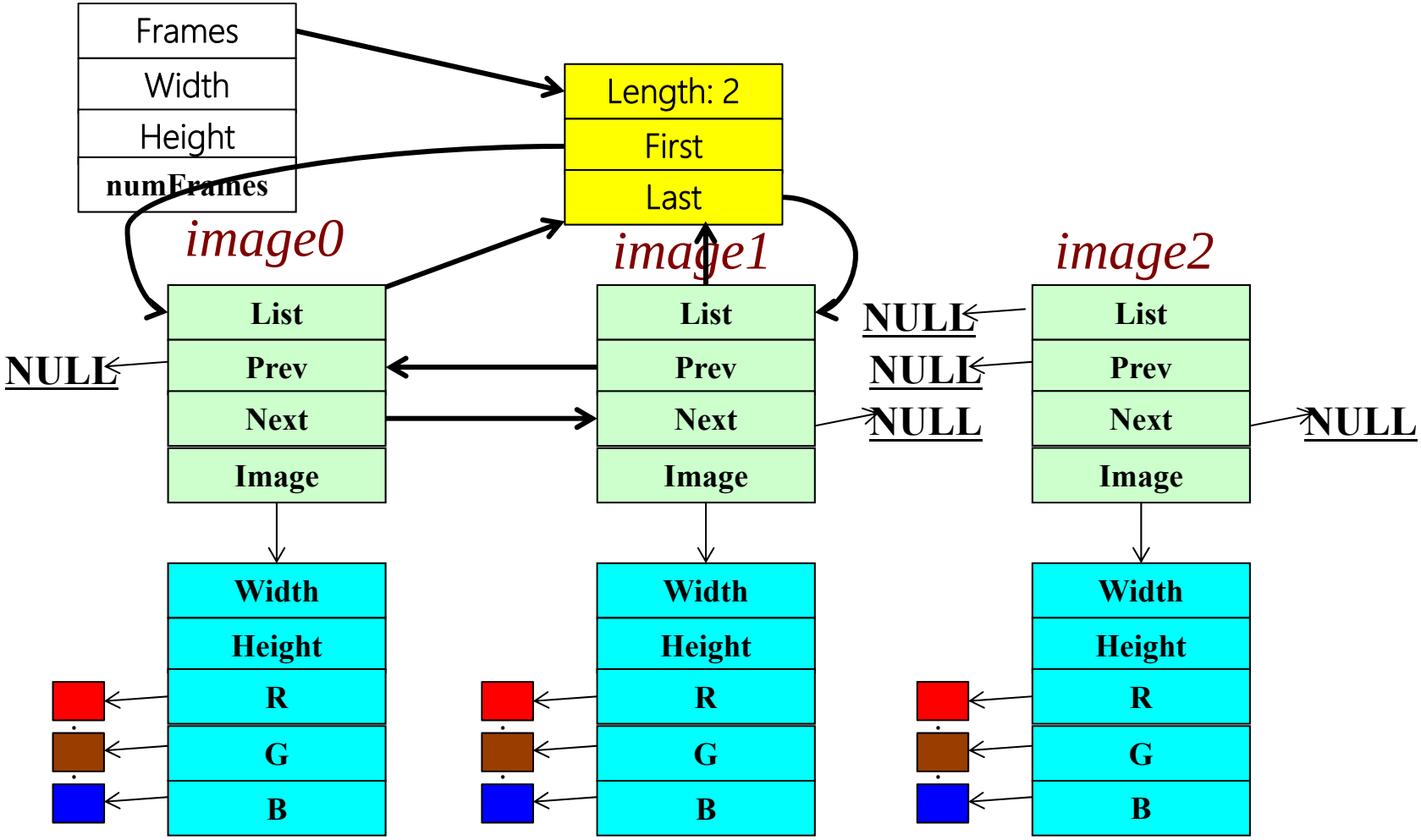
DOUBLE-LINKED LIST: APPEND



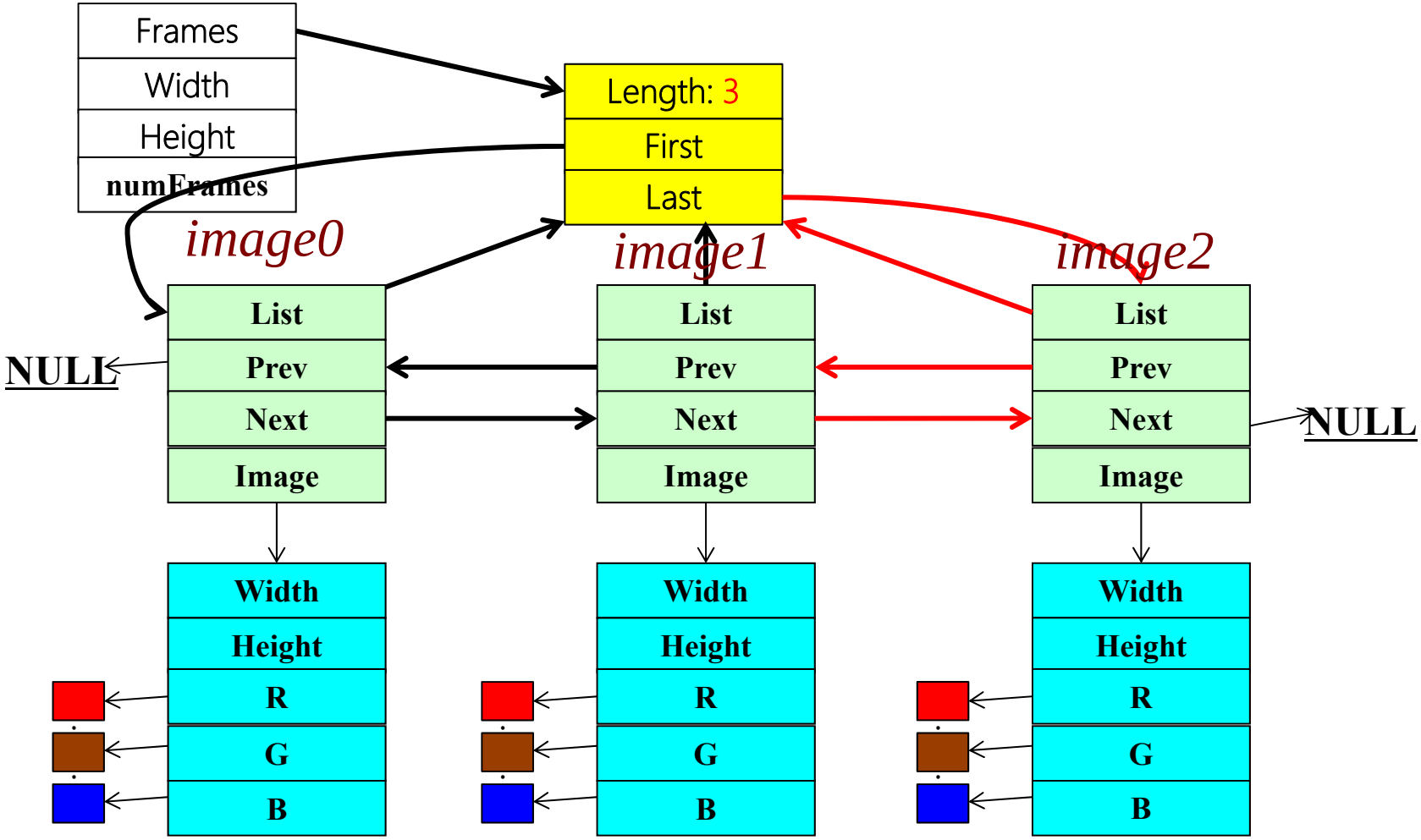
DOUBLE-LINKED LIST: LENGTH = 2



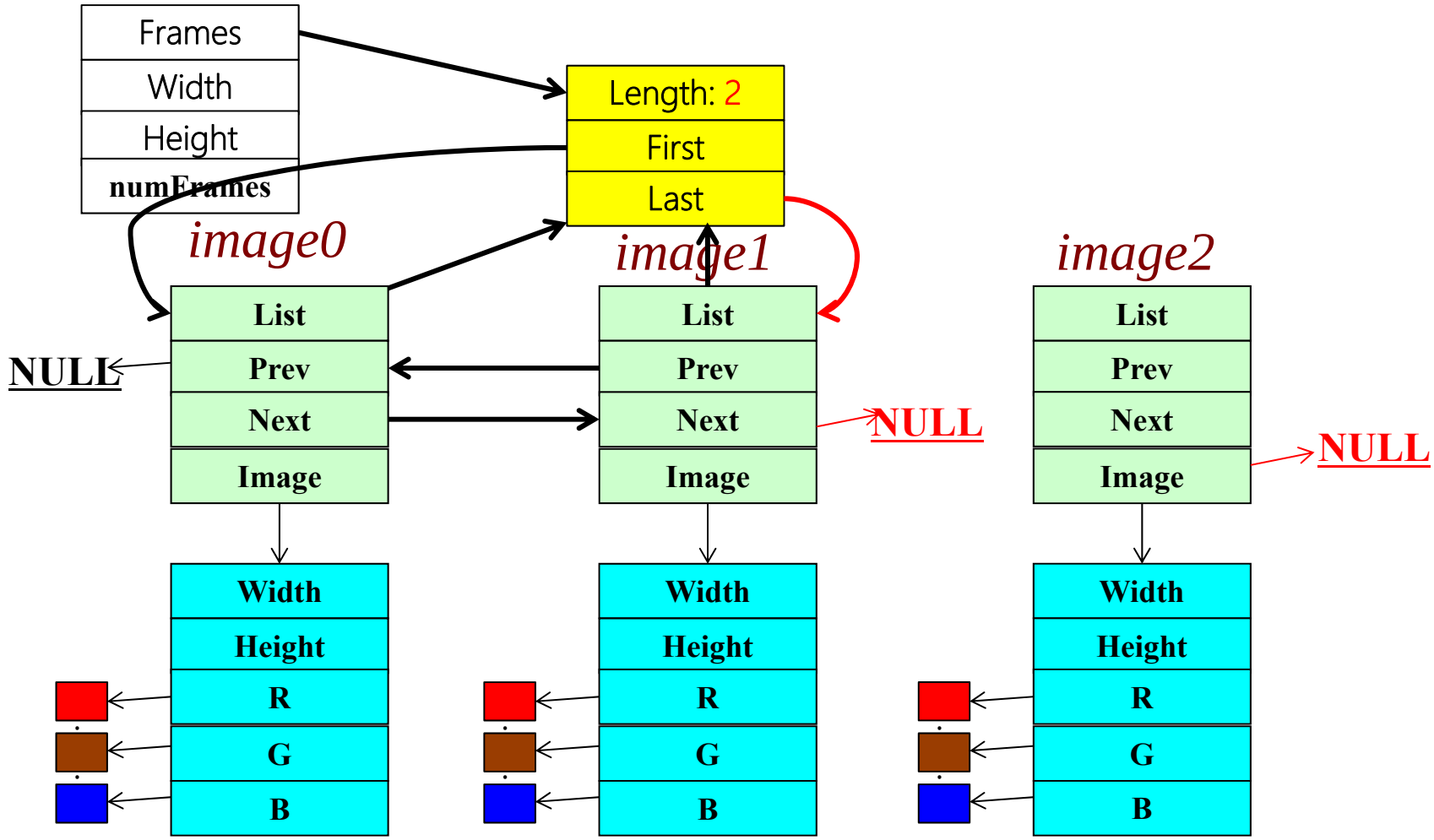
Double-Linked List: Append



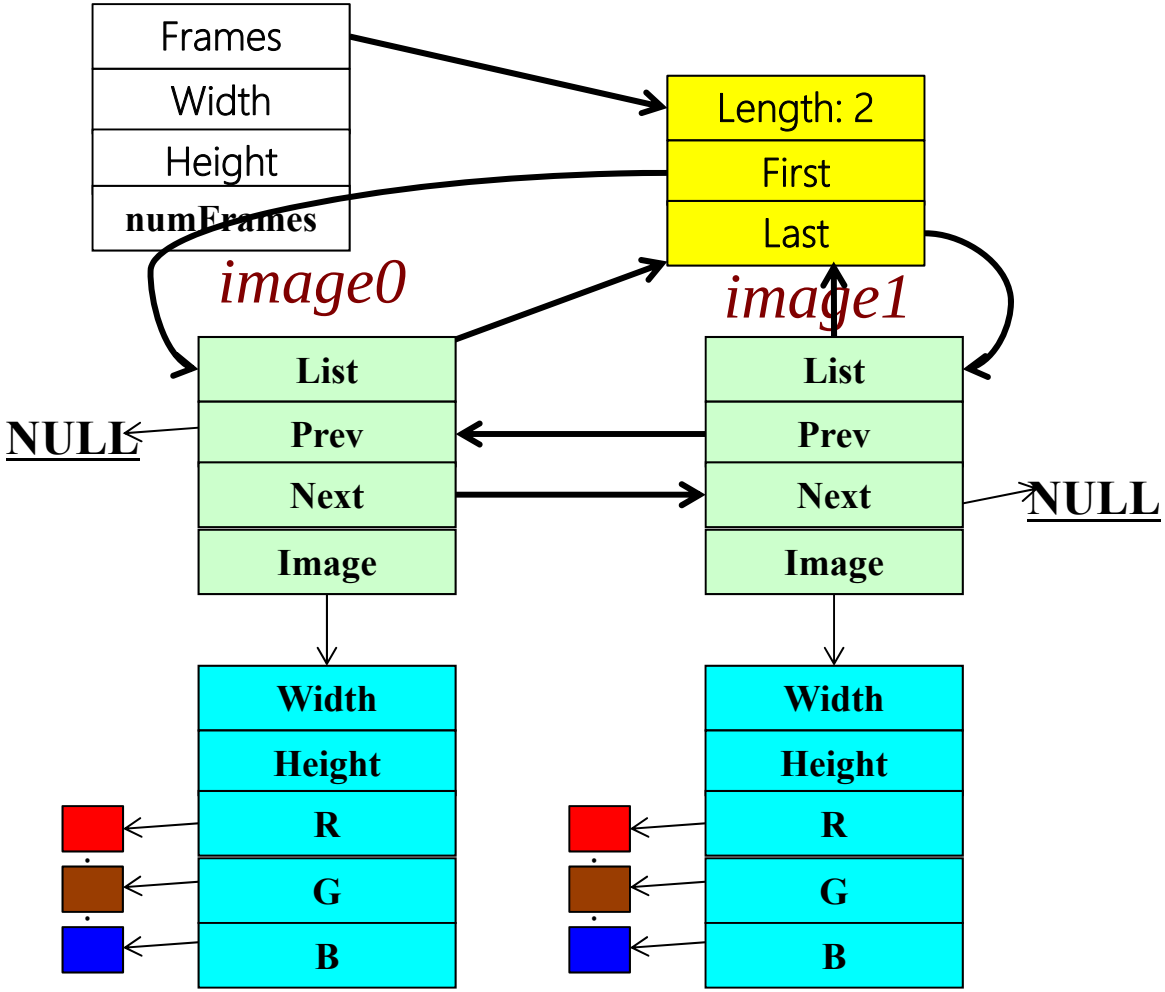
DOUBLE-LINKED LIST: LENGTH = 3



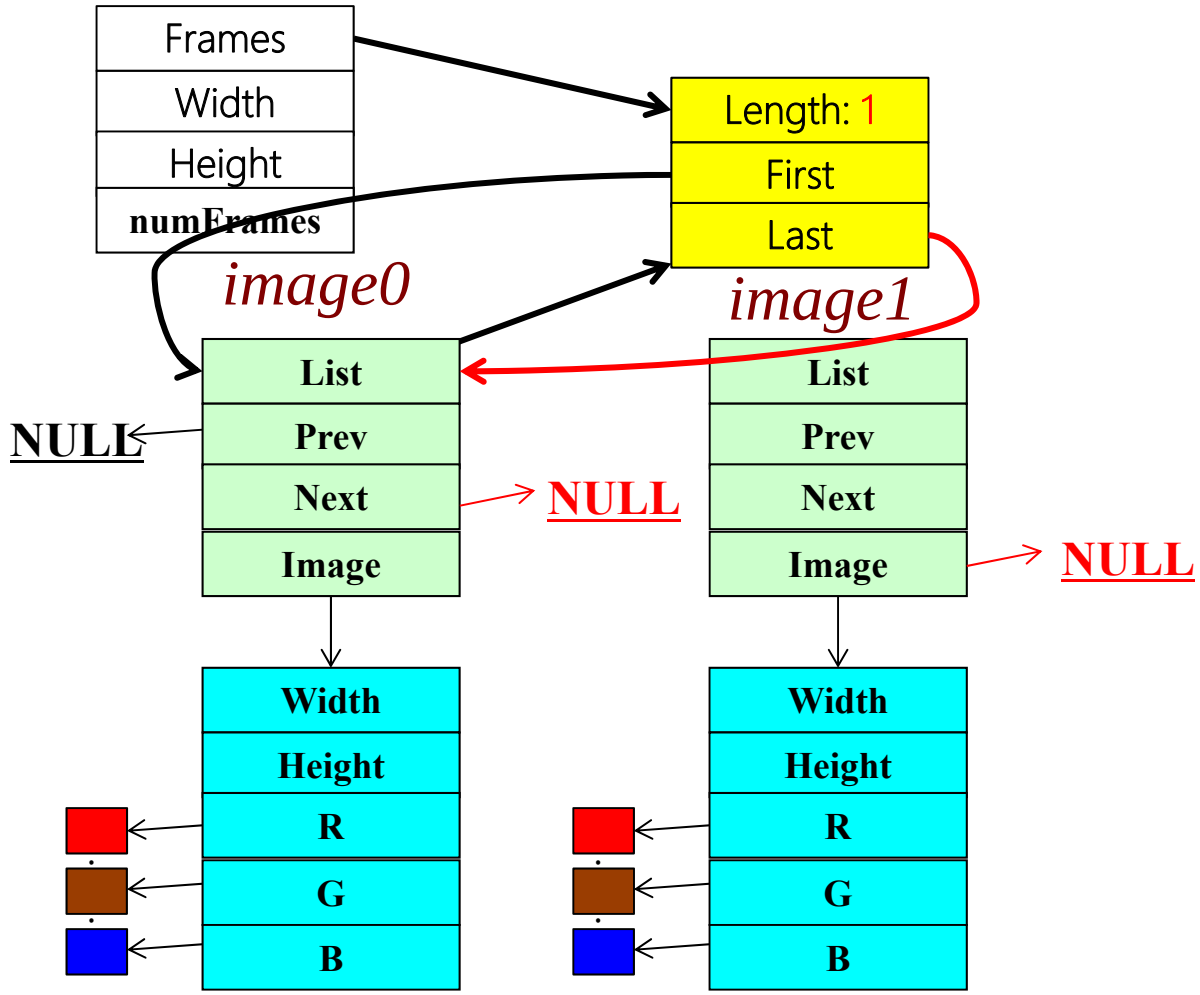
DOUBLE-LINKED LIST: REMOVELAST



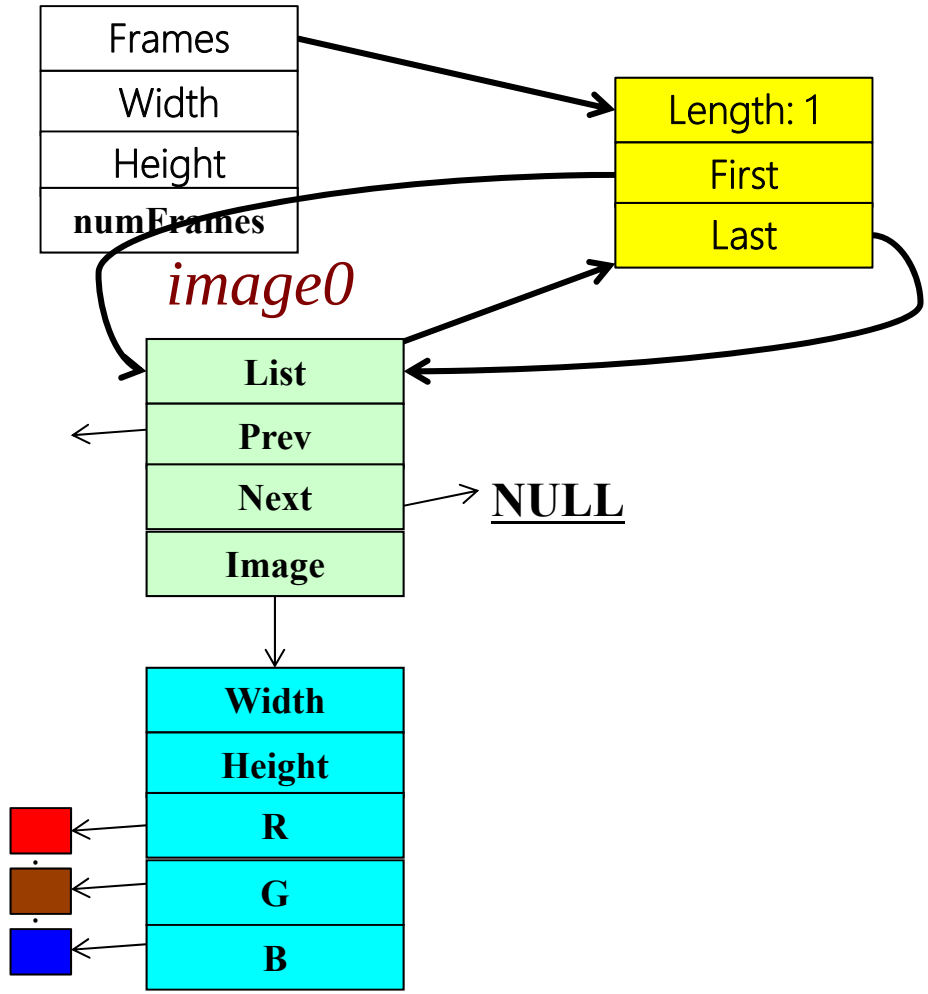
DOUBLE-LINKED LIST: LENGTH = 2



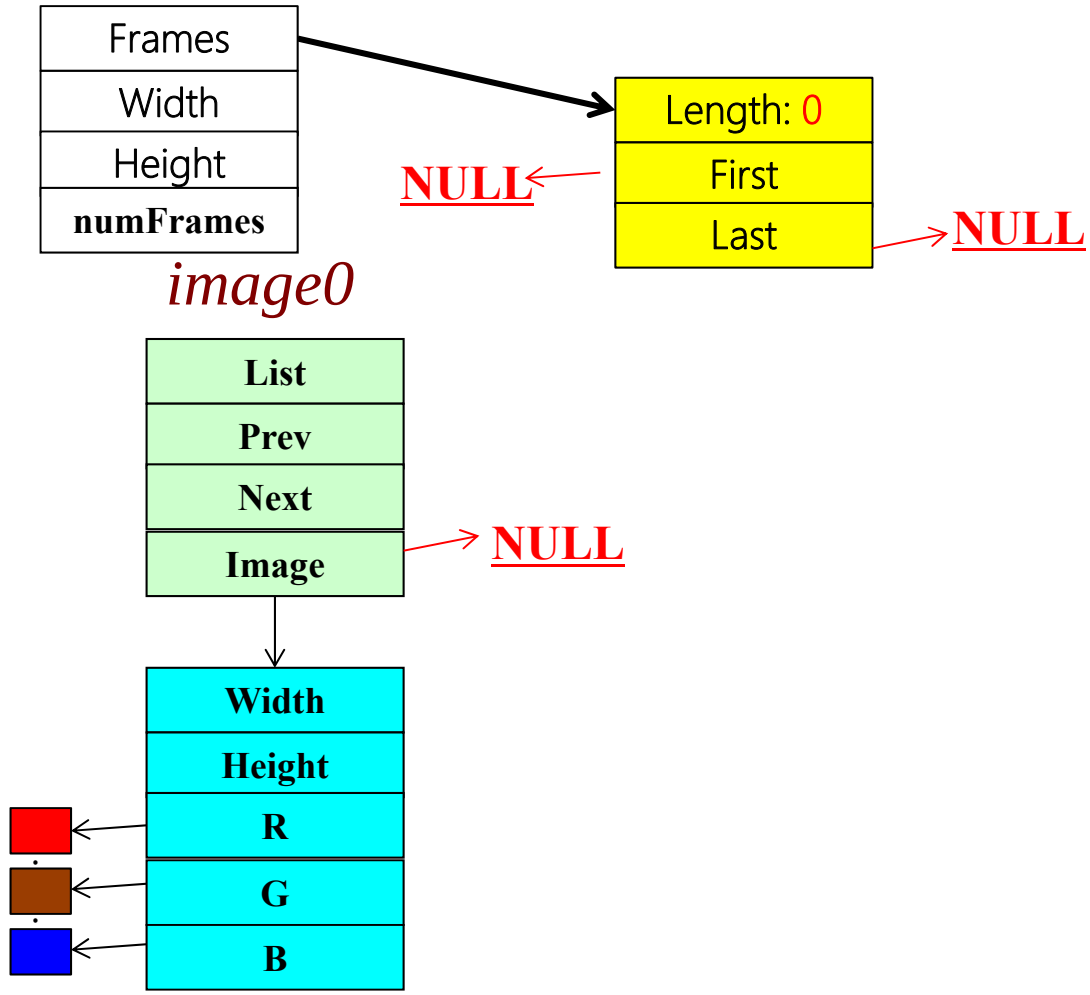
DOUBLE-LINKED LIST: REMOVELAST



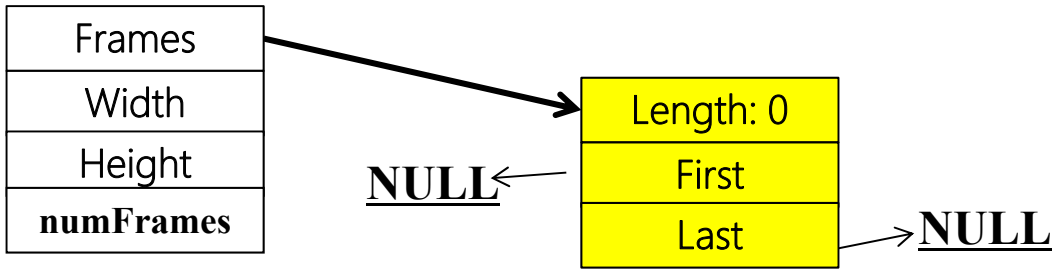
DOUBLE-LINKED LIST: LENGTH = 1



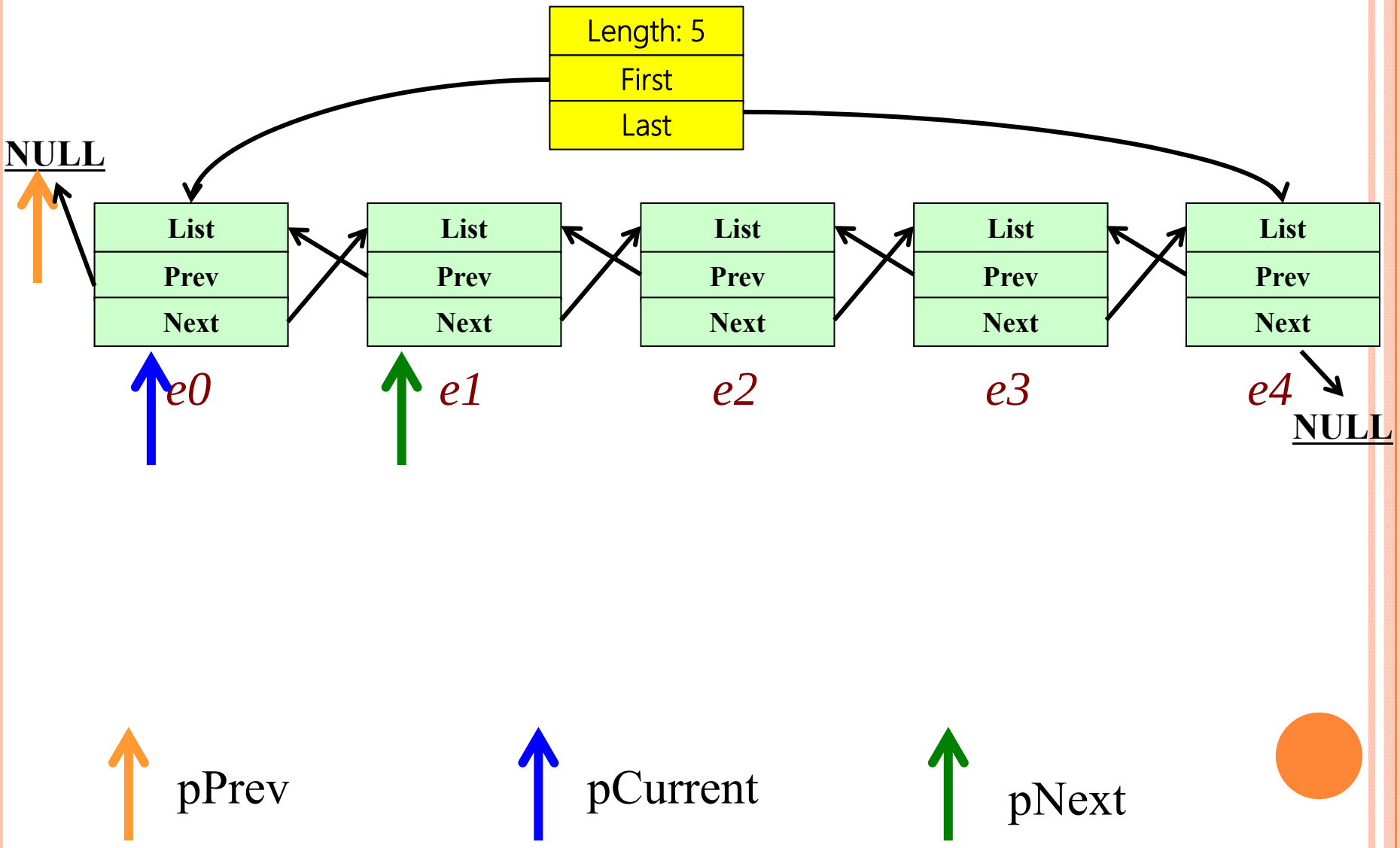
DOUBLE-LINKED LIST: REMOVELAST



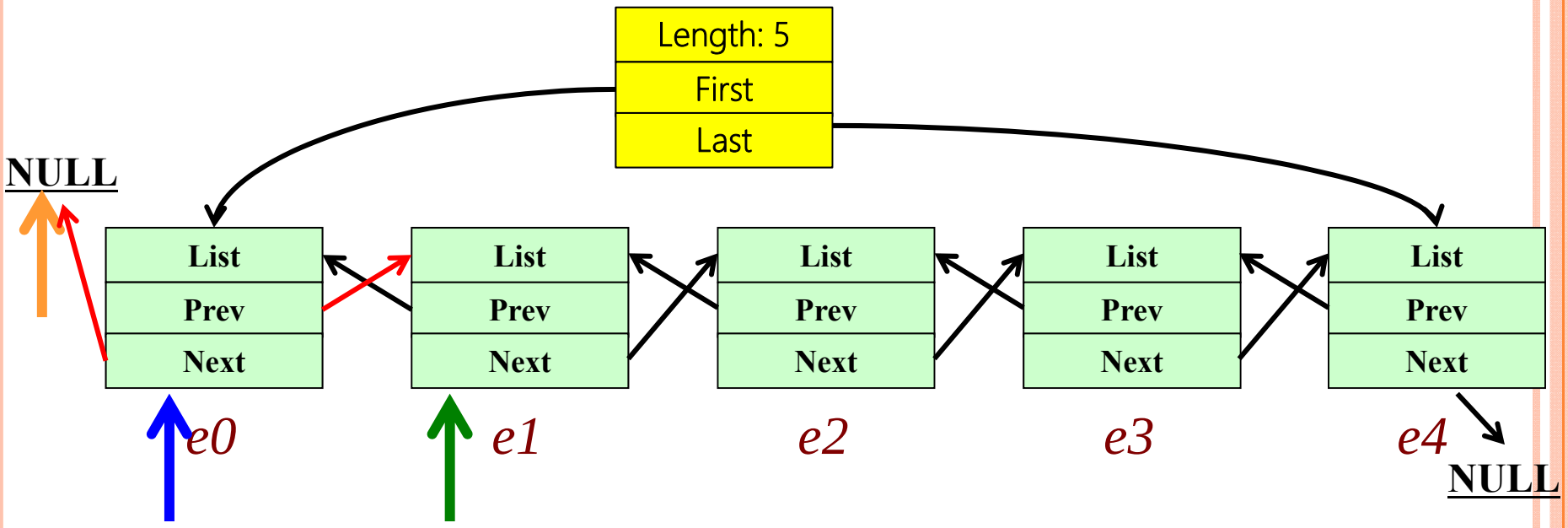
DOUBLE-LINKED LIST: EMPTY



Double-Linked List: Reverse, Initial



Double-Linked List: Reverse, Step 1



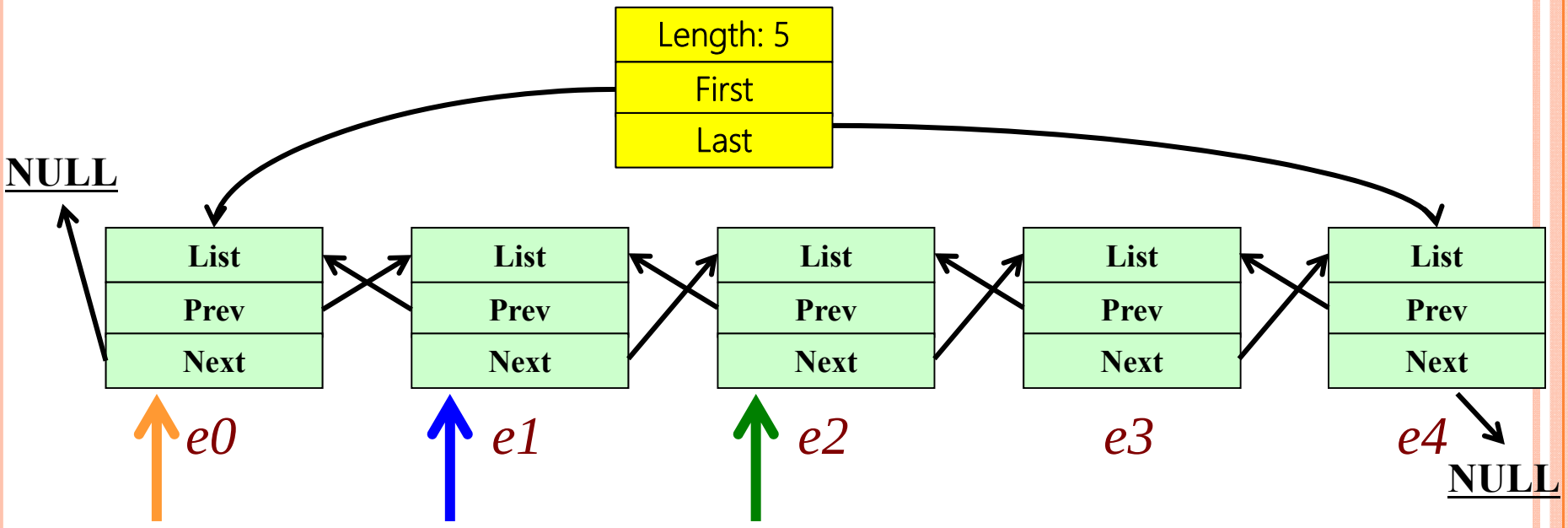

```
pCurrent->Next = pPrev;
pCurrent->Prev = pNext;
```

pPrev

pCurrent

pNext

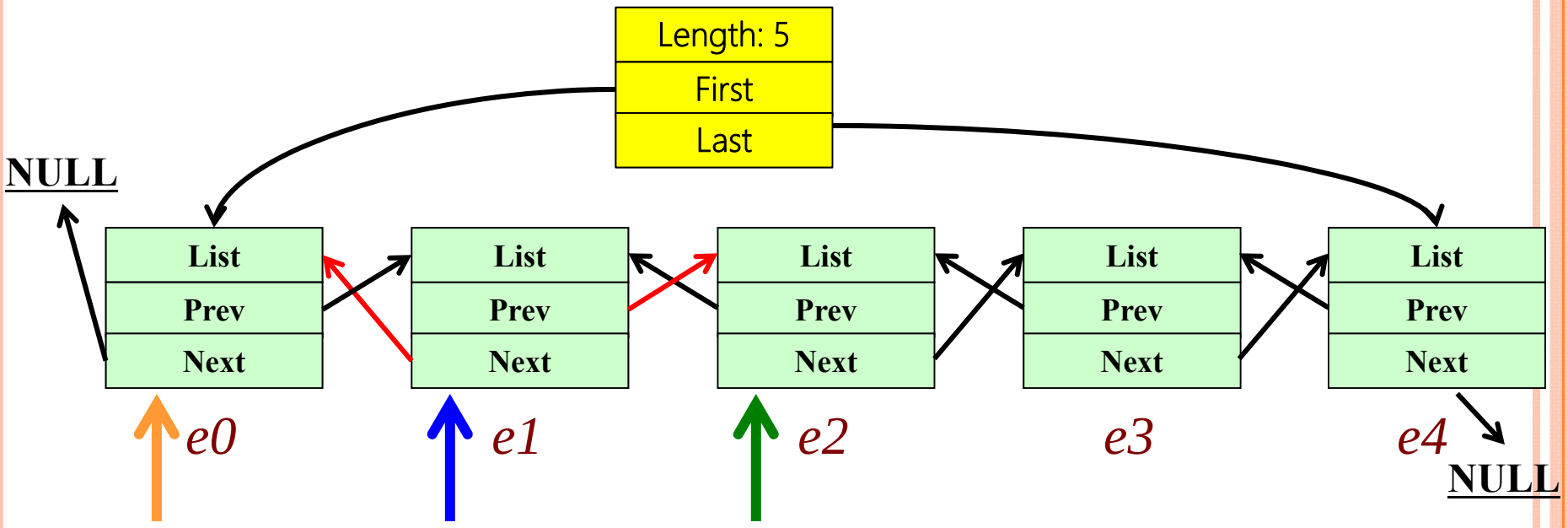
Double-Linked List: Reverse, Step 2



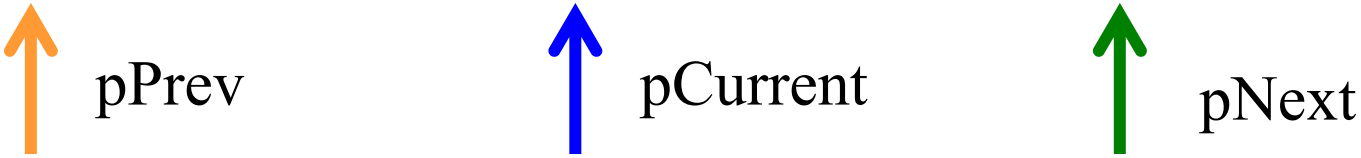

```
pPrev = pCurrent;
pCurrent = pNext;
pNext = pNext->Next;
```

pPrev pCurrent pNext

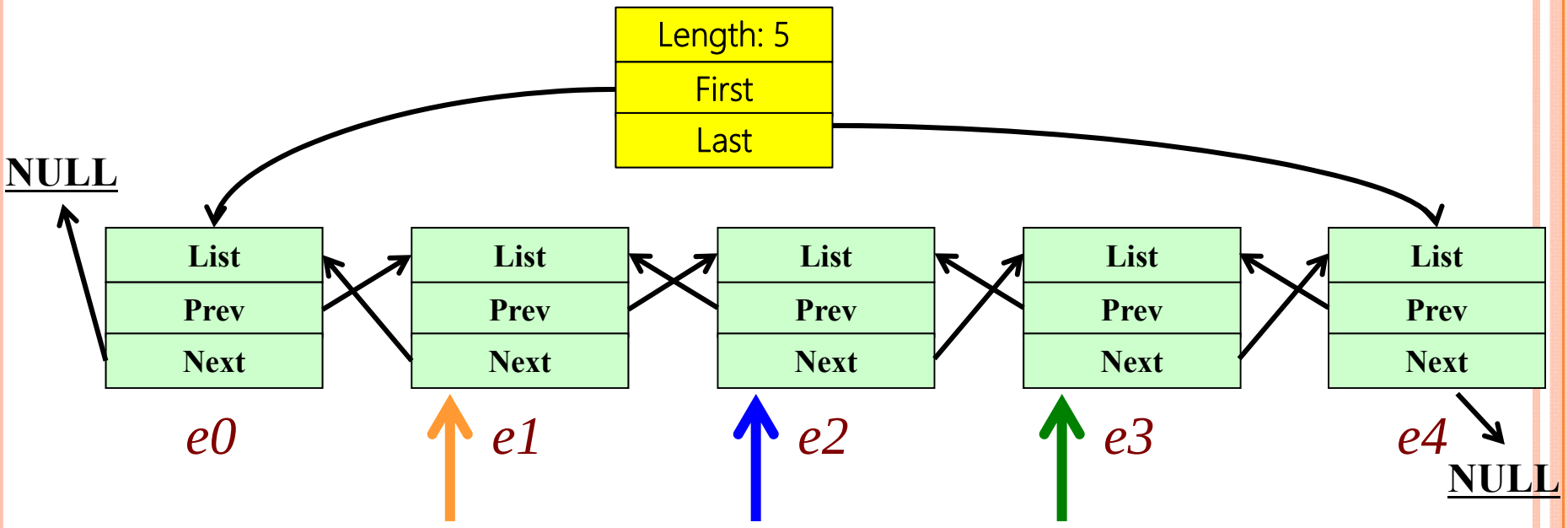
Double-Linked List: Reverse, Step 2




```
pCurrent->Next = pPrev;
pCurrent->Prev = pNext;
```



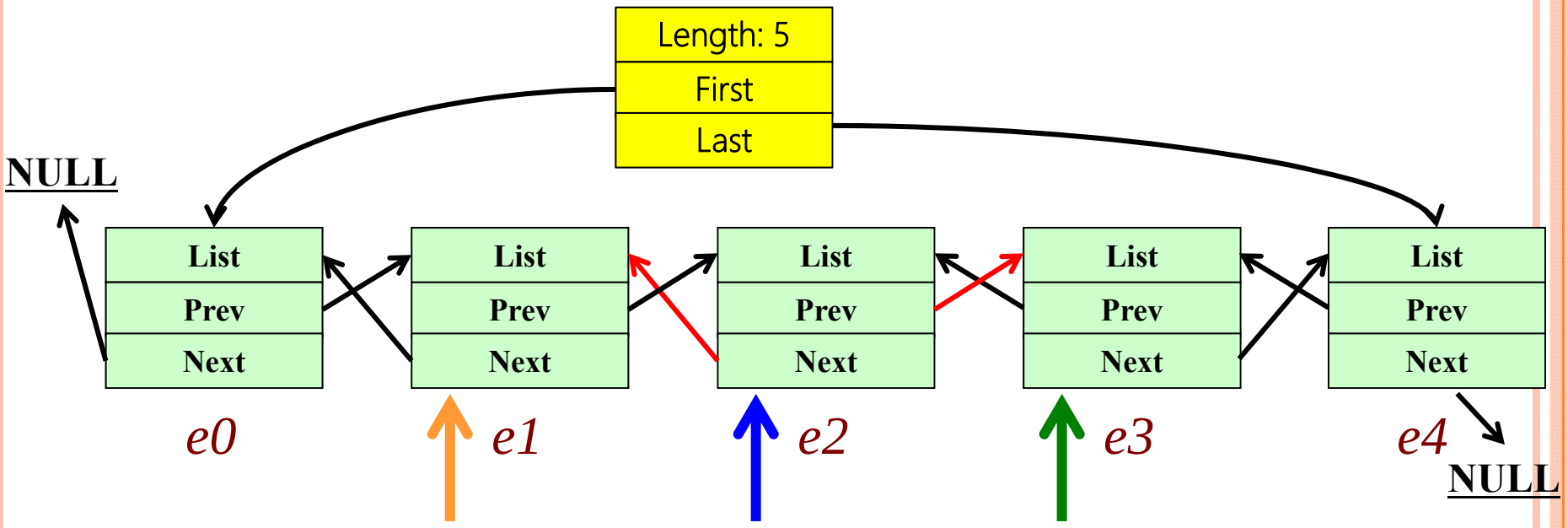
Double-Linked List: Reverse, Step 3




```
pPrev = pCurrent;
pCurrent = pNext;
pNext = pNext->Next;
```

pPrev pCurrent pNext

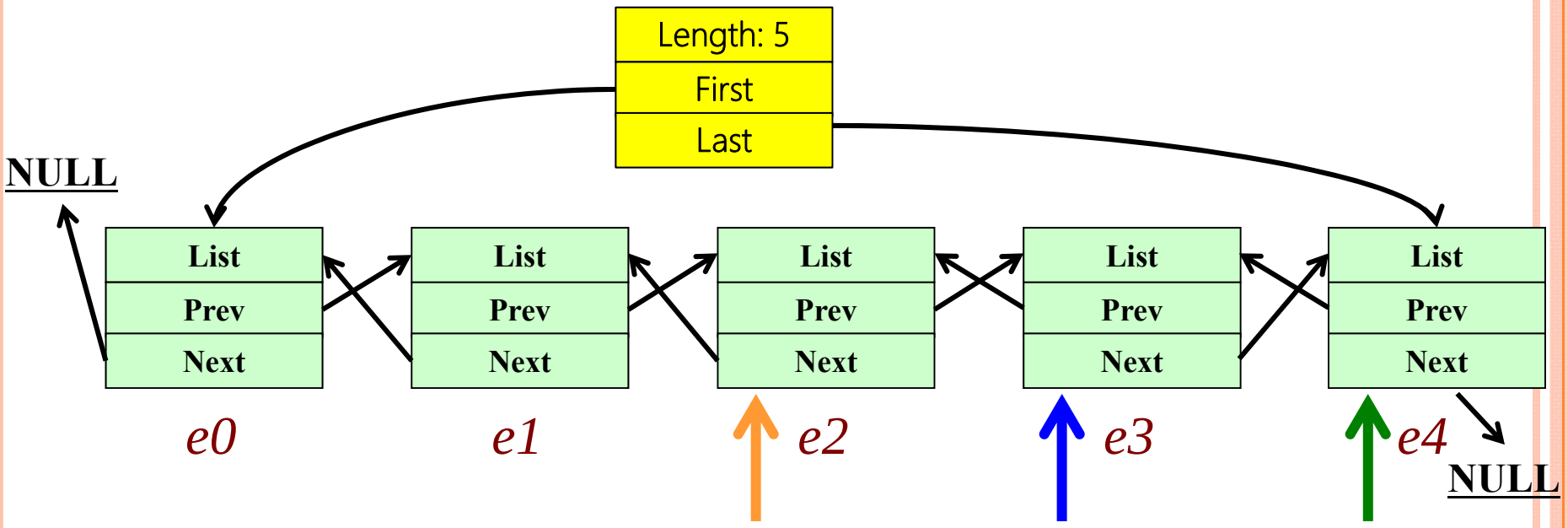
Double-Linked List: Reverse, Step 3



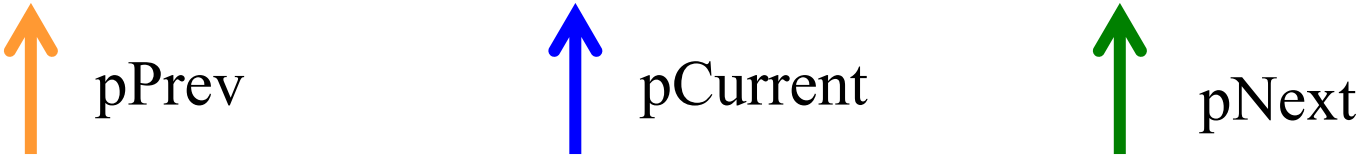

```
pCurrent->Next = pPrev;
pCurrent->Prev = pNext;
```

pPrev pCurrent pNext

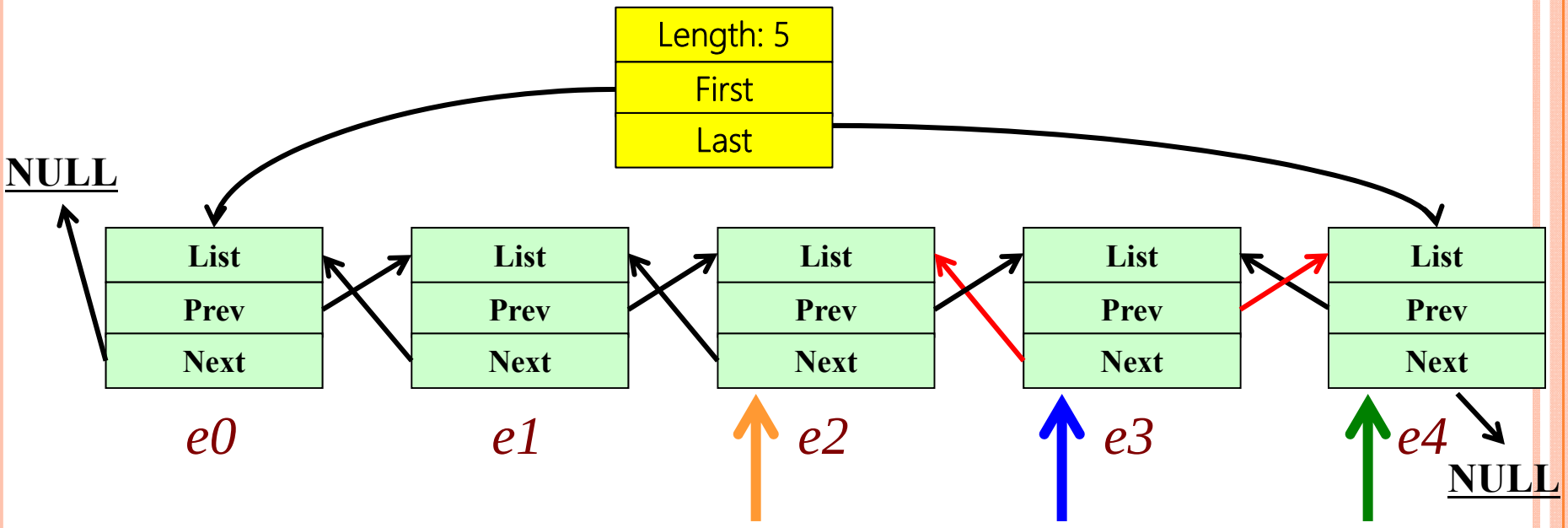
Double-Linked List: Reverse, Step 4




```
pPrev = pCurrent;
pCurrent = pNext;
pNext = pNext->Next;
```



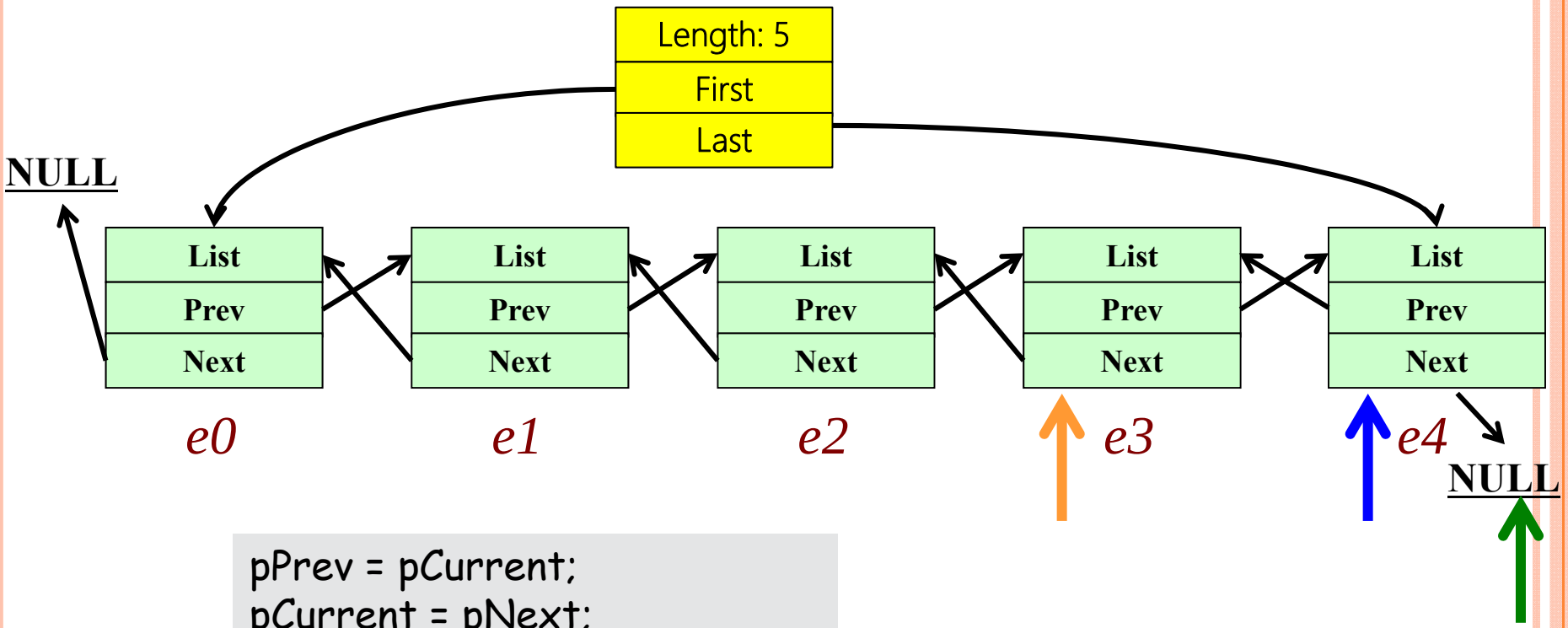
Double-Linked List: Reverse, Step 4



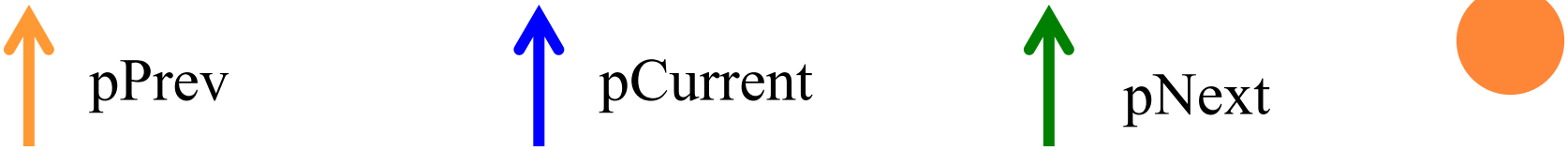

```
pCurrent->Next = pPrev;
pCurrent->Prev = pNext;
```

pPrev pCurrent pNext

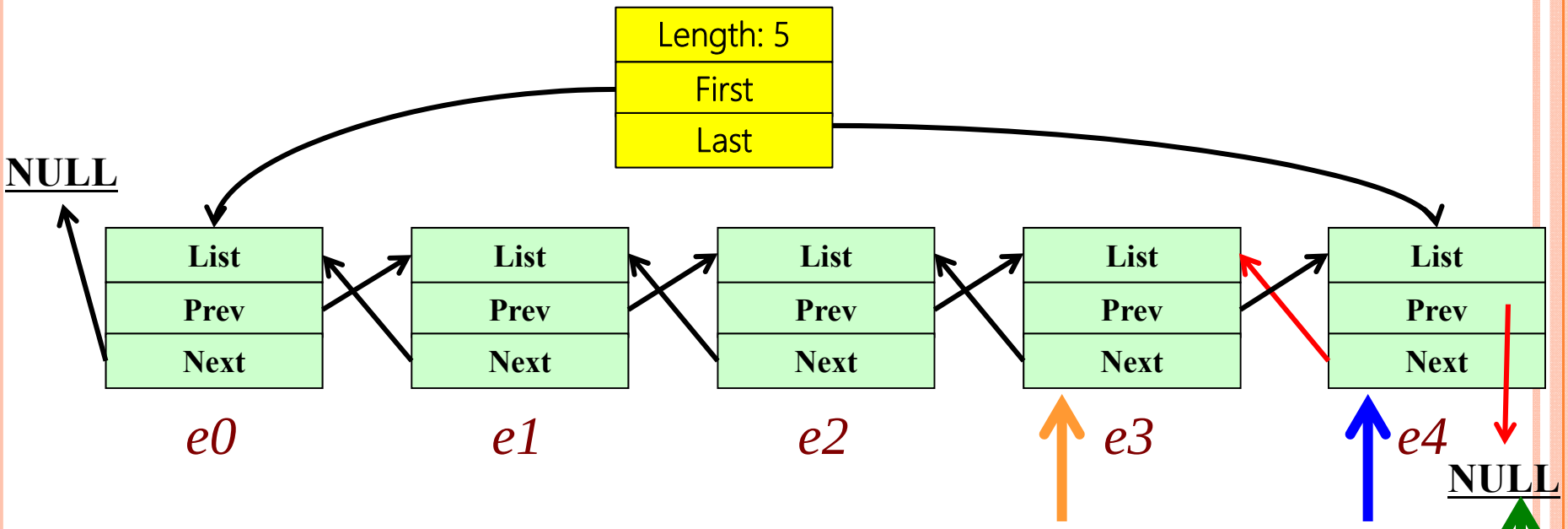
Double-Linked List: Reverse, Step 5




```
pPrev = pCurrent;
pCurrent = pNext;
pNext = pNext->Next;
```



Double-Linked List: Reverse, Step 5




```
pCurrent->Next = pPrev;
pCurrent->Prev = pNext;
```

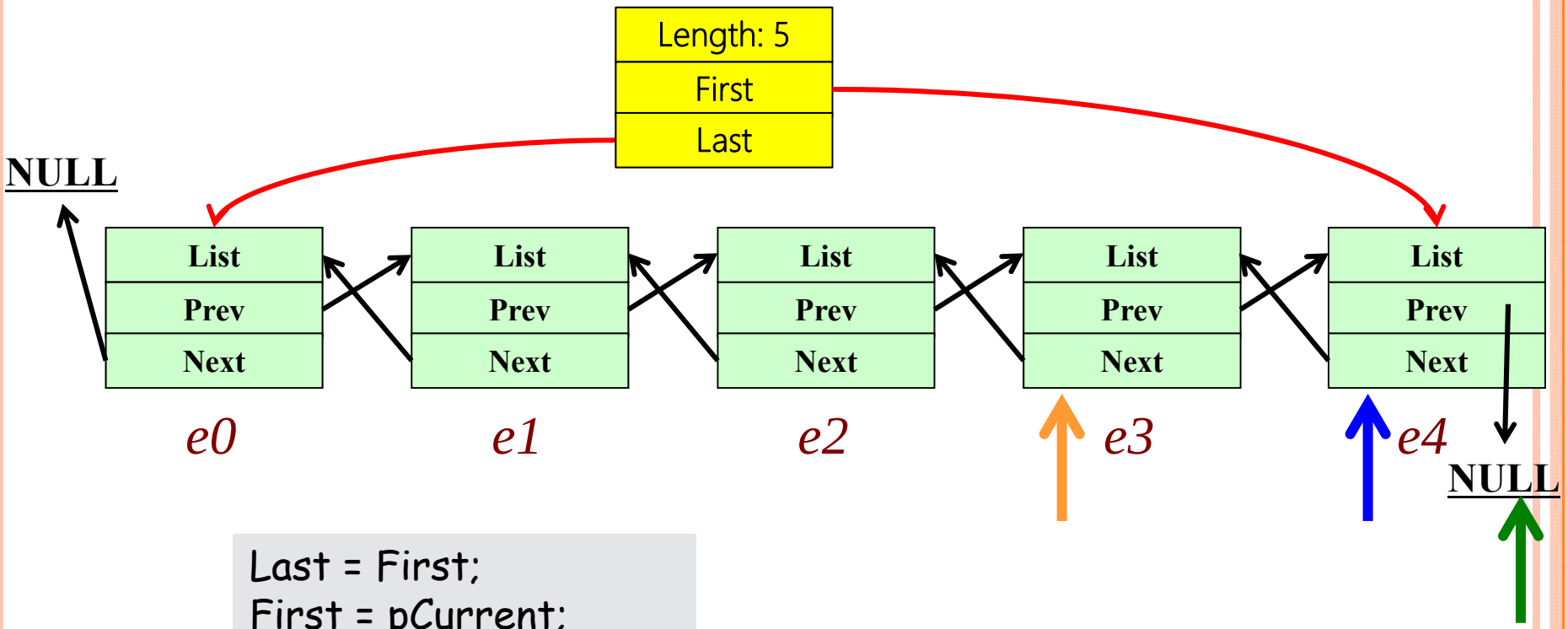
pNext == NULL, stop

pPrev

pCurrent

pNext

Double-Linked List: Reverse, Final Step




```
Last = First;
First = pCurrent;
```

pPrev pCurrent pNext