ECE 220

Lecture x000C - 02/27

Slides based on material originally by: Yuting Chen & Thomas Moon

Recap

Recap

- Last time we discussed sorting & searching

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recap

- Last time we discussed sorting & searching
  - Selection sort

Recap

- Last time we discussed sorting & searching
    - Selection sort
    - Insertion sort

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - Insertion sort
  - Quick sort

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - Insertion sort
  - Quick sort
  - Bubble sort

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - Insertion sort
  - Quick sort
  - Bubble sort

A. Select next element and move things to place it in proper spot

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - Insertion sort
  - Quick sort
  - Bubble sort

A. Select next element and move things to place it in proper spot

B. Keep comparing pairs and swapping them till no more swaps

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - Insertion sort
  - Quick sort
  - Bubble sort

A. Select next element and move things to place it in proper spot

B. Keep comparing pairs and swapping them till no more swaps

C. Find minimum in each pass and bring to appropriate spot

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - Insertion sort
  - Quick sort
  - Bubble sort

A. Select next element and move things to place it in proper spot

B. Keep comparing pairs and swapping them till no more swaps

C. Find minimum in each pass and bring to appropriate spot

D. Pick pivot & move elements to left (lesser) or right (greater) of pivot

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - A Insertion sort
  - Quick sort
  - Bubble sort

A. Select next element and move things to place it in proper spot

B. Keep comparing pairs and swapping them till no more swaps

C. Find minimum in each pass and bring to appropriate spot

D. Pick pivot & move elements to left (lesser) or right (greater) of pivot

Recap

- Last time we discussed sorting & searching
  - Selection sort
  - A • Insertion sort
  - Quick sort
  - B • Bubble sort

A. Select next element and move things to place it in proper spot
B. Keep comparing pairs and swapping them till no more swaps
C. Find minimum in each pass and bring to appropriate spot
D. Pick pivot & move elements to left (lesser) or right (greater) of pivot

Recap

- Last time we discussed sorting & searching
  - C • Selection sort
  - A • Insertion sort
  - • Quick sort
  - B • Bubble sort

A. Select next element and move things to place it in proper spot
B. Keep comparing pairs and swapping them till no more swaps
C. Find minimum in each pass and bring to appropriate spot
D. Pick pivot & move elements to left (lesser) or right (greater) of pivot

Recap

- Last time we discussed sorting & searching

C • Selection sort

A • Insertion sort

D • Quick sort

B • Bubble sort

A. Select next element and move things to place it in proper spot
B. Keep comparing pairs and swapping them till no more swaps
C. Find minimum in each pass and bring to appropriate spot
D. Pick pivot & move elements to left (lesser) or right (greater) of pivot

Recap

- Last time we discussed sorting & searching

C • Selection sort

A • Insertion sort

D • Quick sort

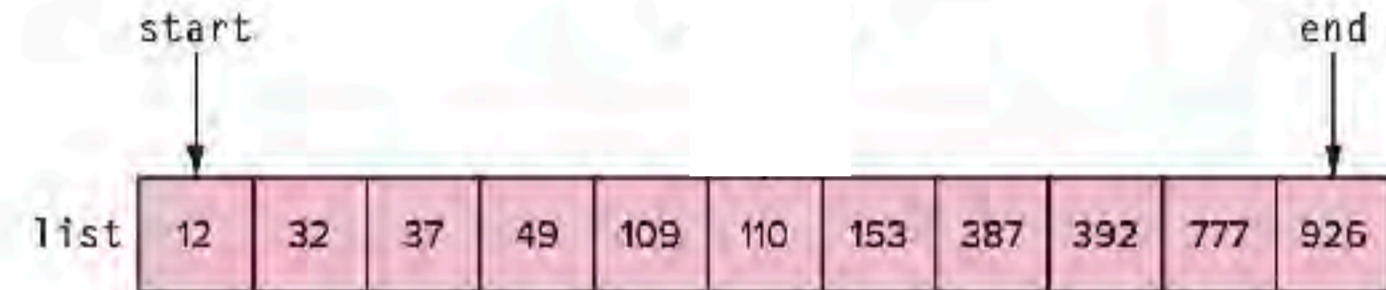
B • Bubble sort

A. Select next element and move things to place it in proper spot
B. Keep comparing pairs and swapping them till no more swaps
C. Find minimum in each pass and bring to appropriate spot
D. Pick pivot & move elements to left (lesser) or right (greater) of pivot

Complete codes for A, C, D available on Gitlab: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Recap binary search

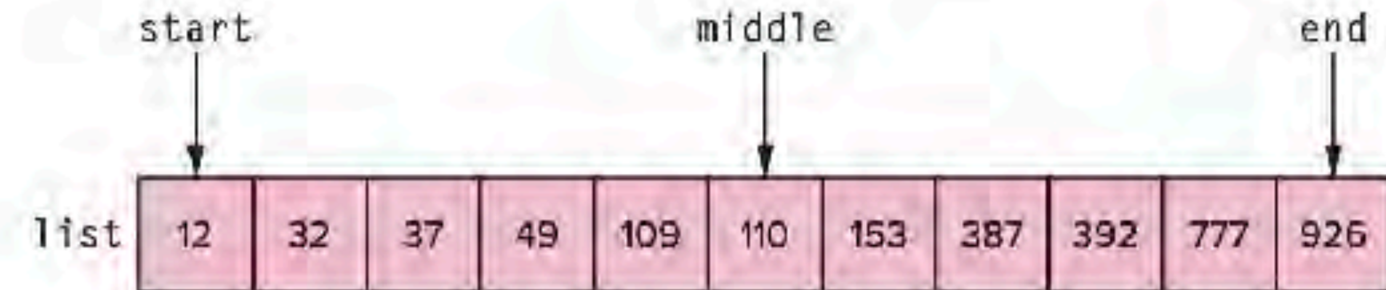
Key = 109



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recap binary search

Key = 109 **Pick middle**



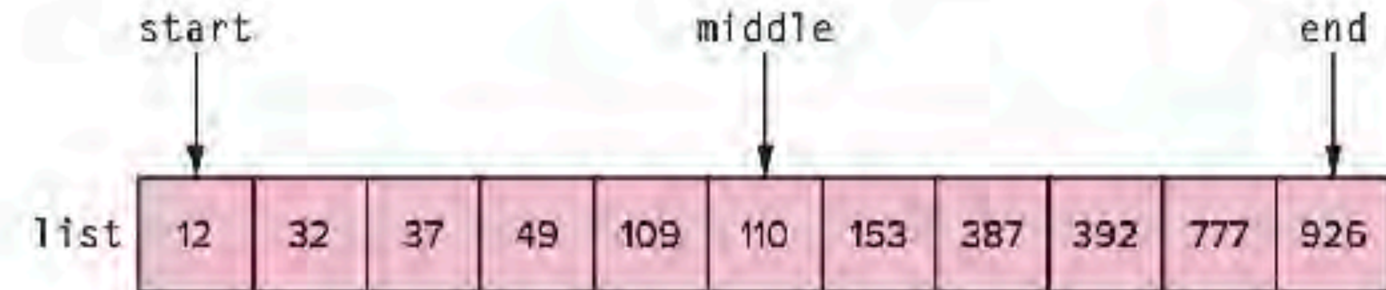
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recap binary search

Key = 109

Pick middle

Is middle == key?



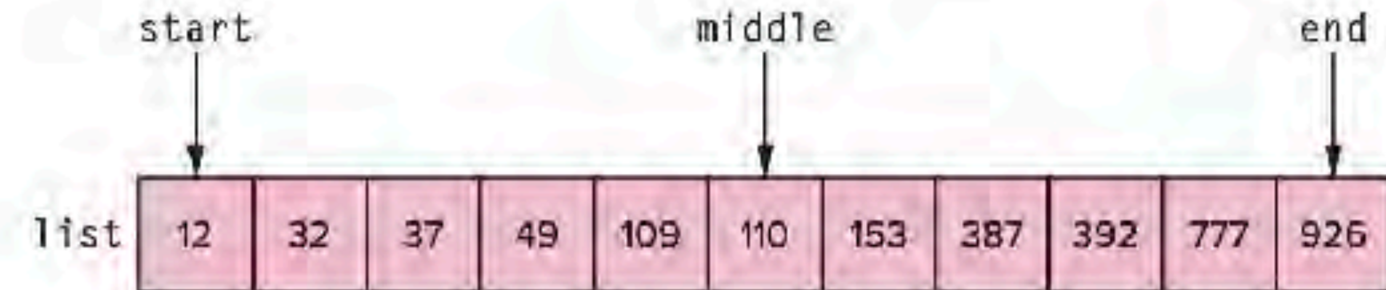
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recap binary search

Key = 109

Pick middle
Is middle > key?

Is middle == key?



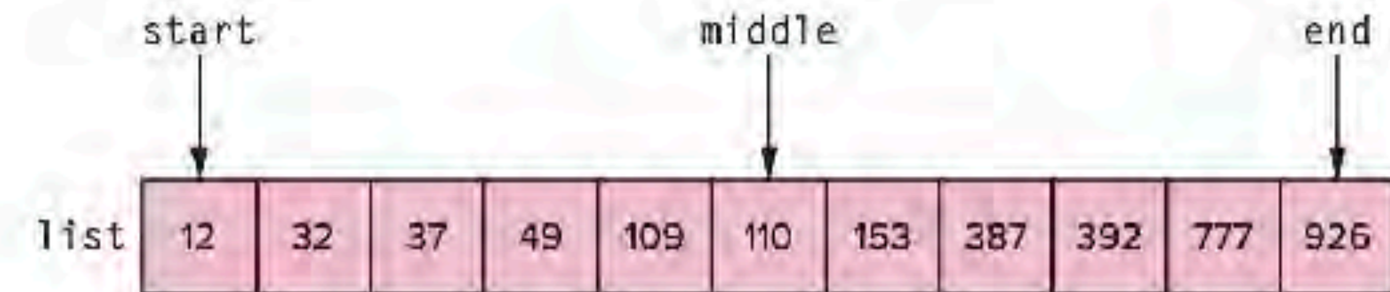
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

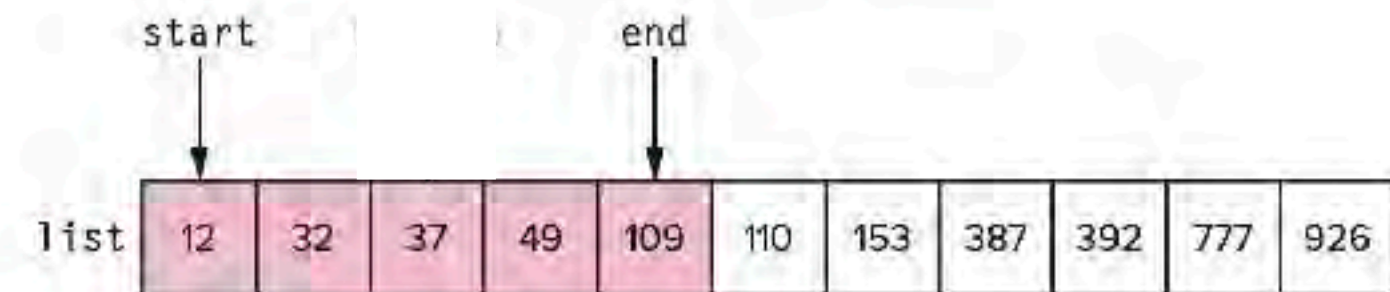
Recap binary search

Key = 109

Pick middle
Is middle > key?

Is middle == key?
Go left





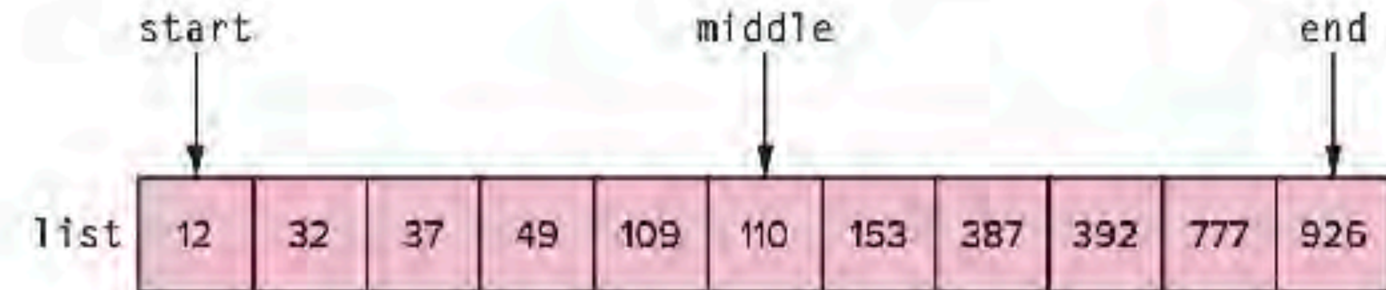
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recap binary search

Key = 109

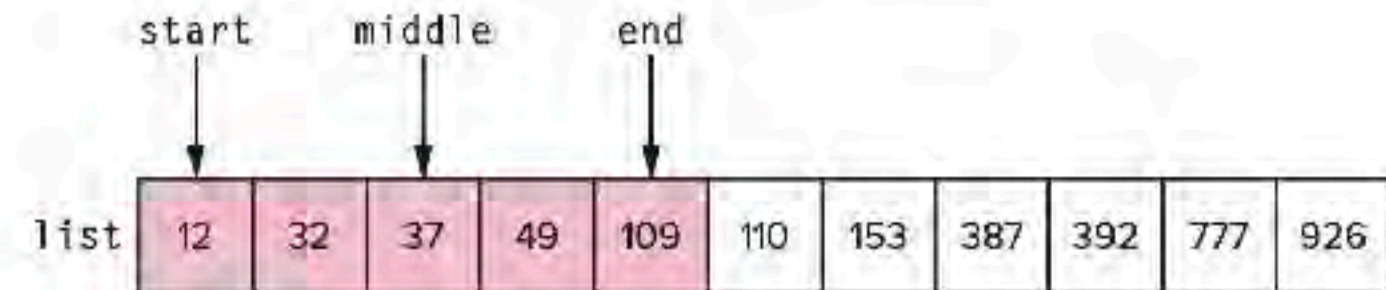
Pick middle
Is middle > key?

Is middle == key?
Go left



Pick middle

Is middle == key?

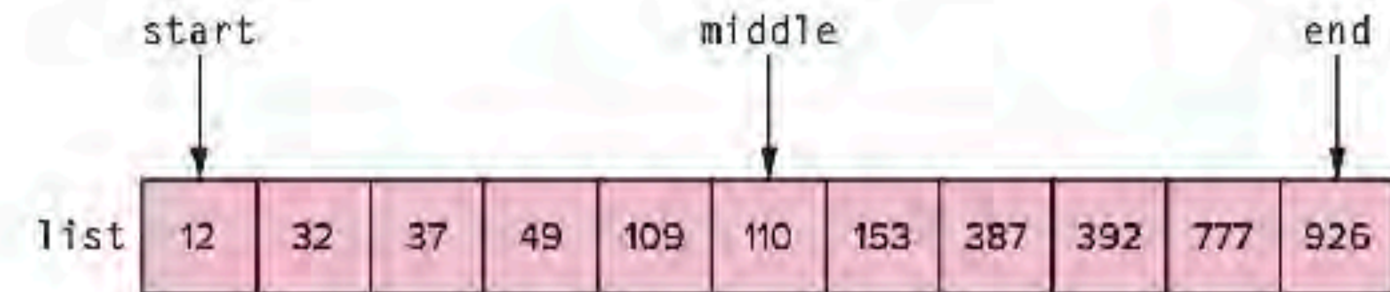


Recap binary search

Key = 109

Pick middle
Is middle > key?

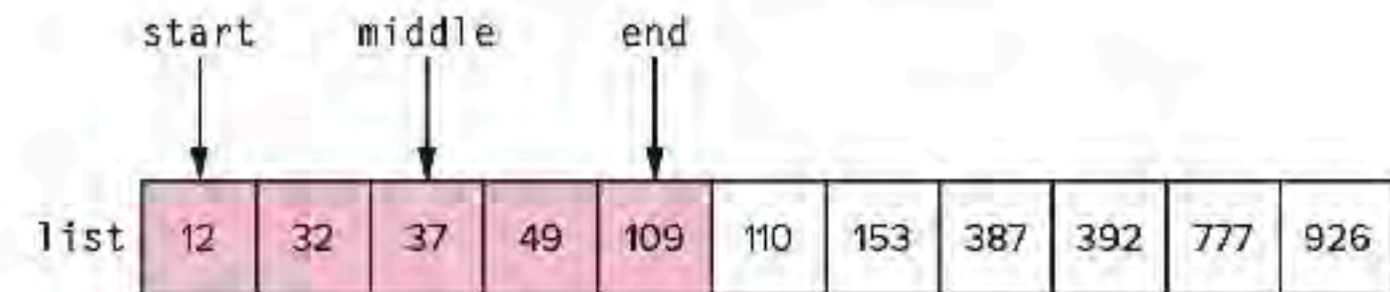
Is middle == key?
Go left



Pick middle

Is middle > key?

Is middle == key?



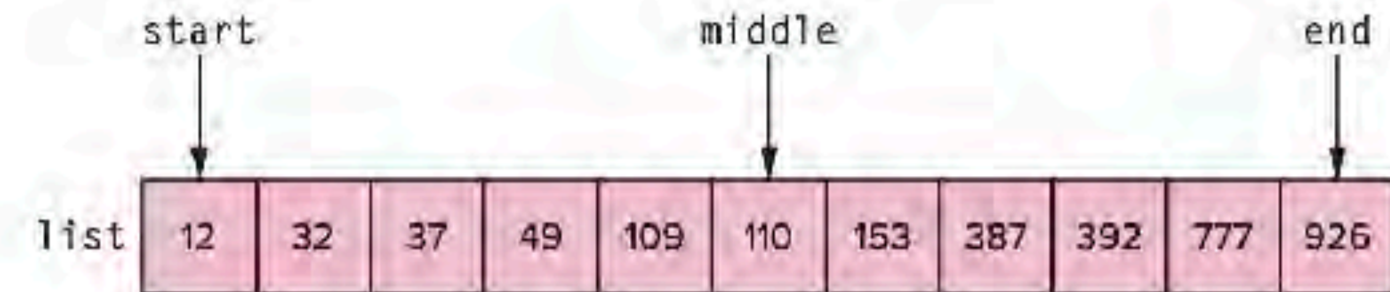
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recap binary search

Key = 109

Pick middle
Is middle > key?

Is middle == key?
Go left

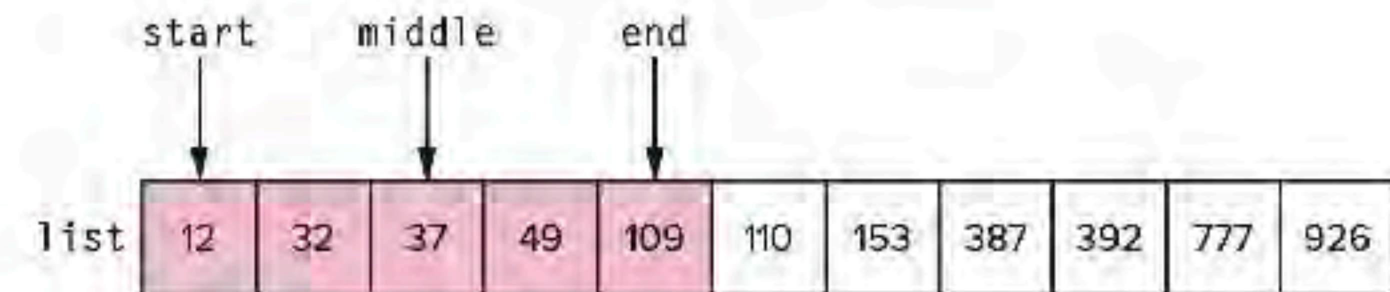


Pick middle

Is middle > key?

Is middle == key?

Is middle < key?

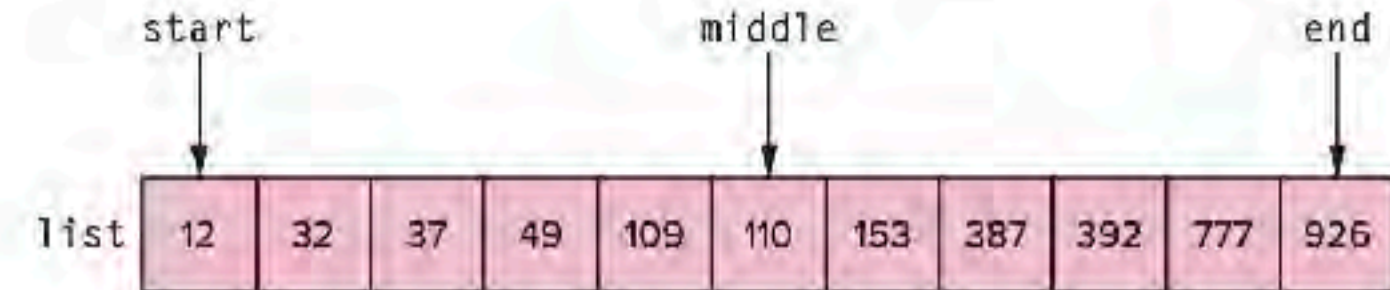


Recap binary search

Key = 109

Pick middle
Is middle > key?

Is middle == key?
Go left



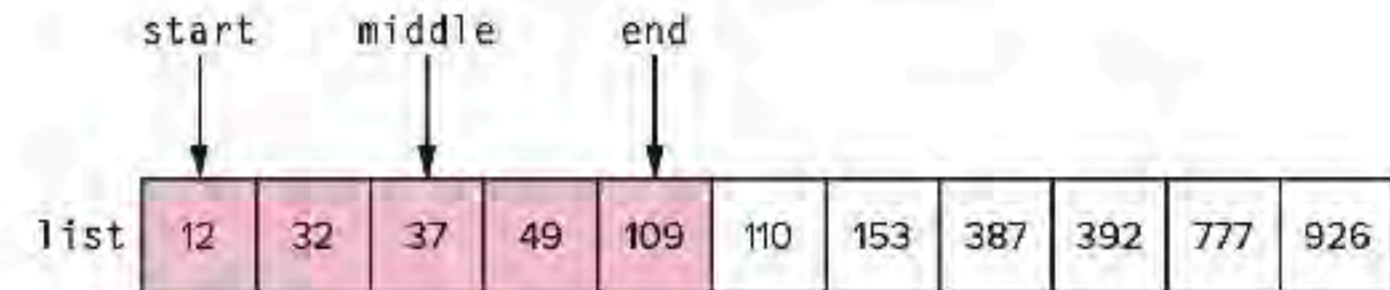
Pick middle

Is middle > key?

Is middle == key?

Is middle < key?

Go right

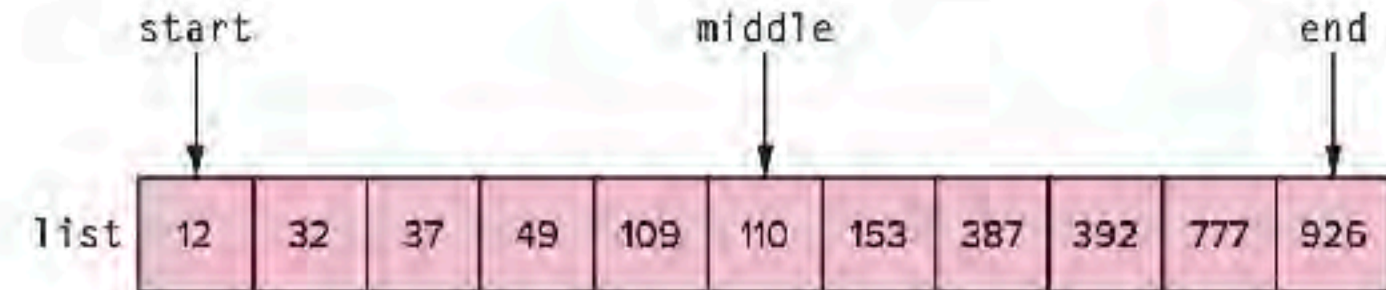


Recap binary search

Key = 109

Pick middle
Is middle > key?

Is middle == key?
Go left



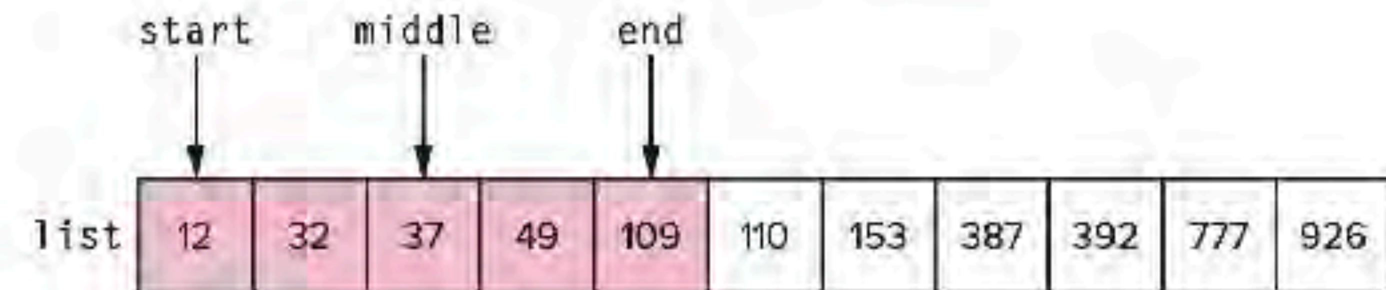
Pick middle

Is middle > key?

Is middle == key?

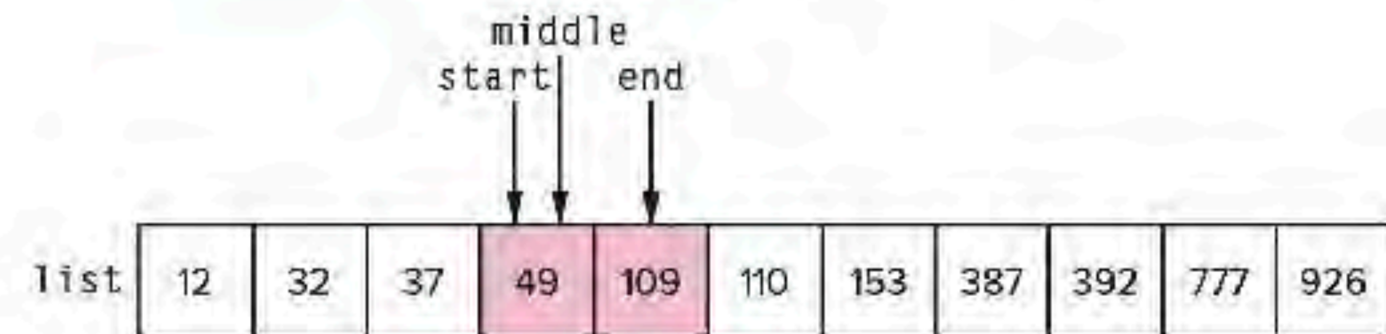
Is middle < key?

Go right



Pick middle

Is middle == key?

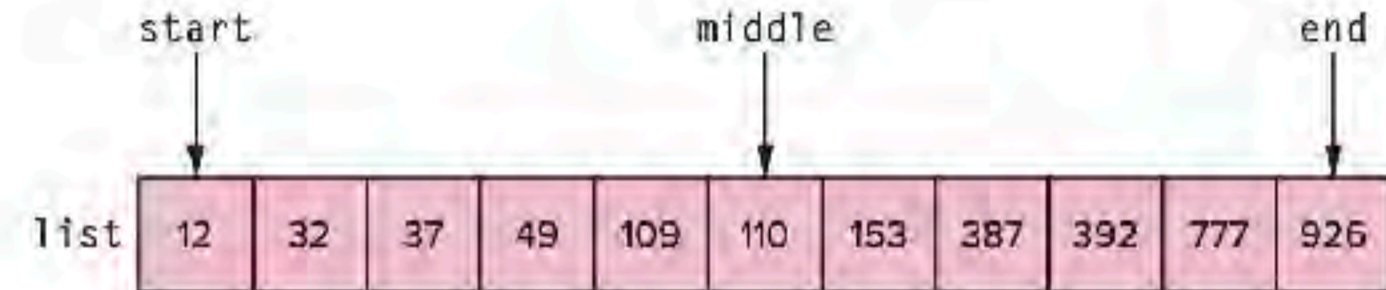


Recap binary search

Key = 109

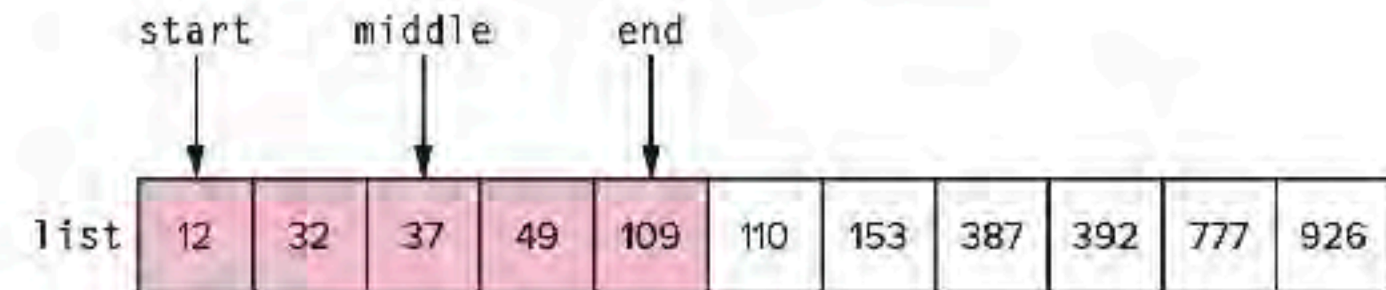
Pick middle
Is middle > key?

Is middle == key?
Go left



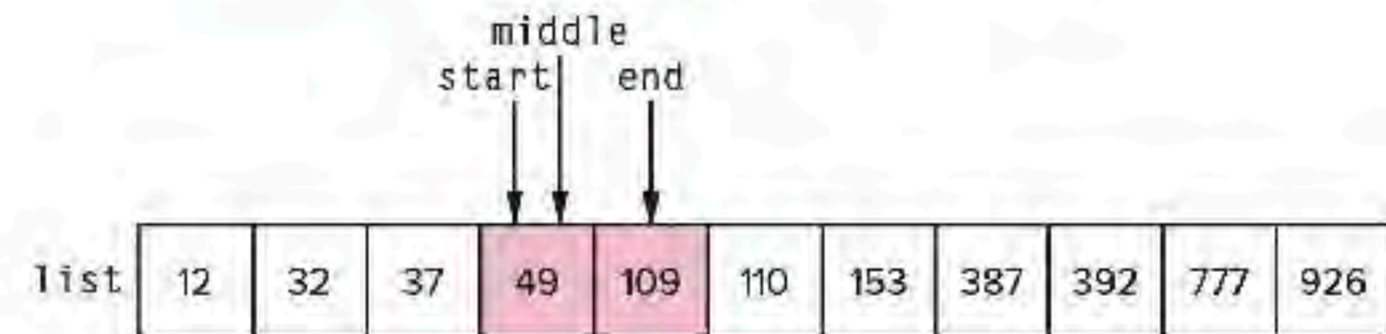
Pick middle
Is middle > key?

Is middle == key?
Is middle < key?
Go right



Pick middle
Is middle > key?

Is middle == key?

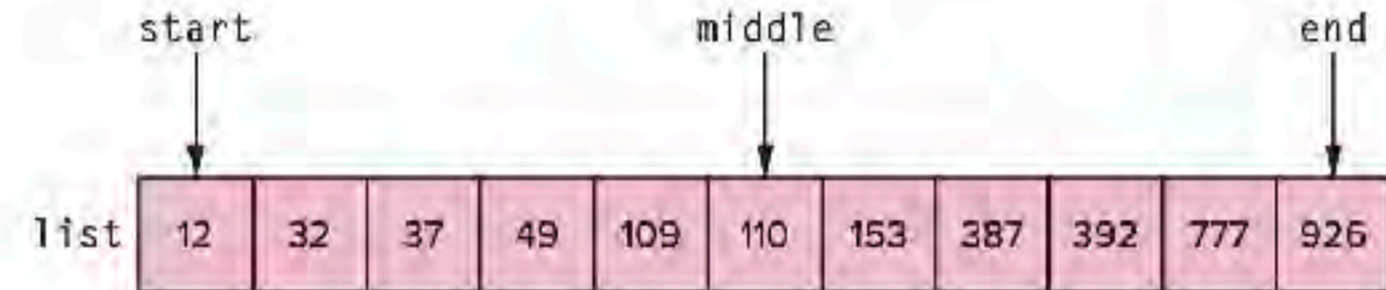


Recap binary search

Key = 109

Pick middle
Is middle > key?

Is middle == key?
Go left



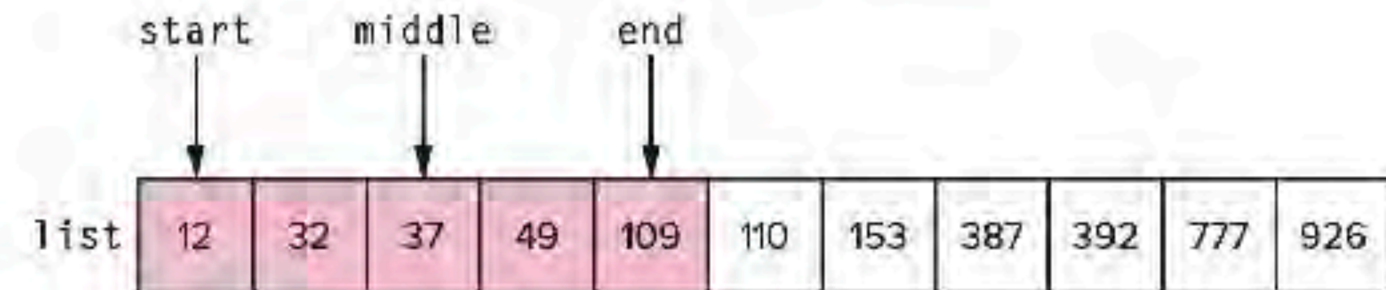
Pick middle

Is middle > key?

Is middle == key?

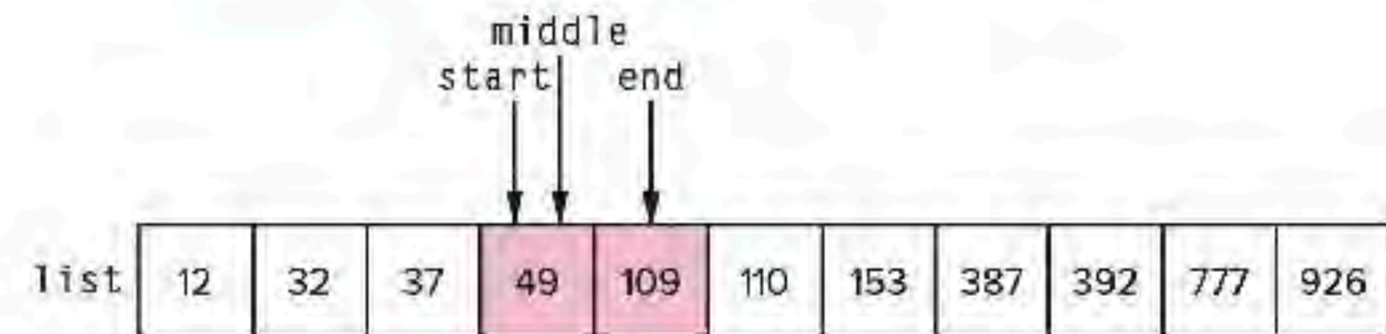
Is middle < key?

Go right



Pick middle
Is middle > key?

Is middle == key?
Is middle < key?

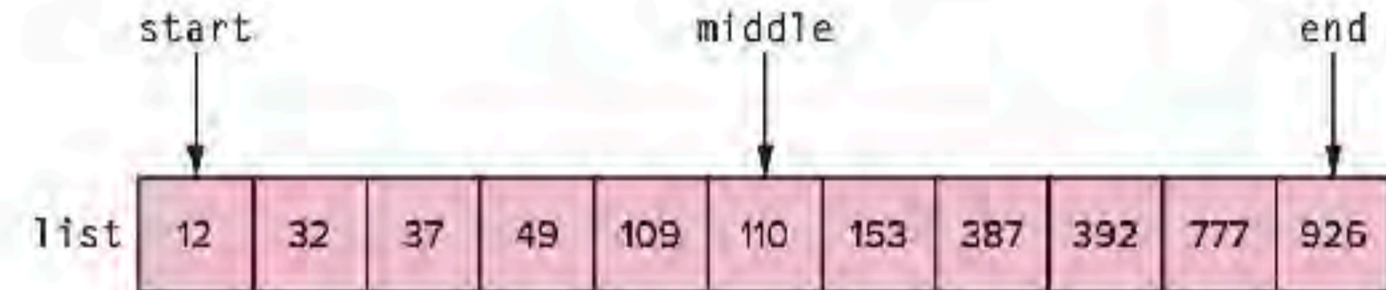


Recap binary search

Key = 109

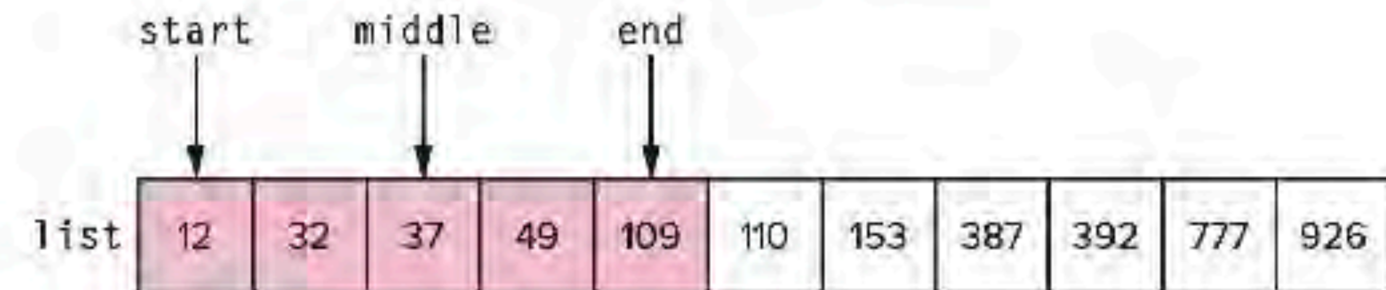
Pick middle
Is middle > key?

Is middle == key?
Go left



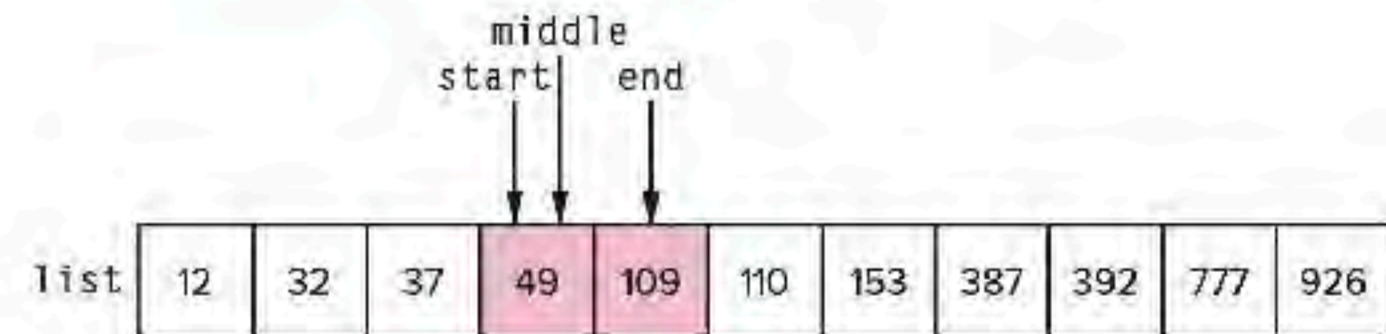
Pick middle
Is middle > key?

Is middle == key?
Is middle < key?
Go right



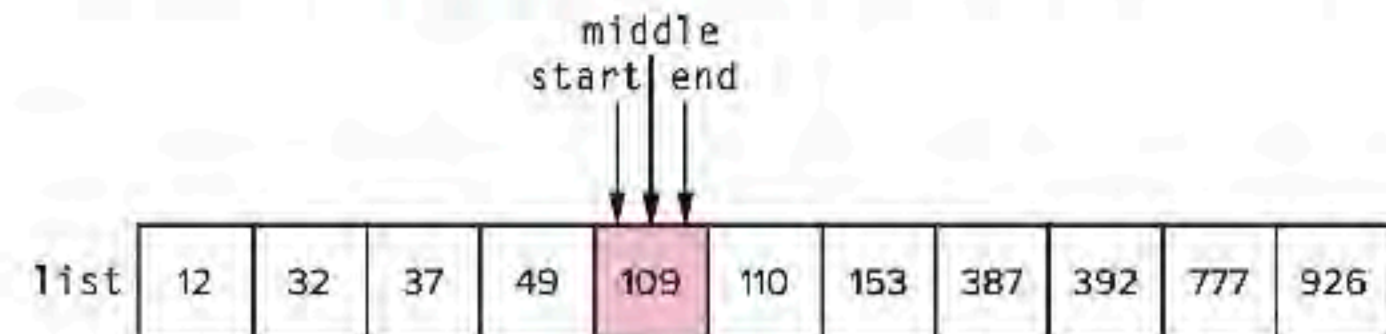
Pick middle
Is middle > key?

Is middle == key?
Is middle < key?
Go right

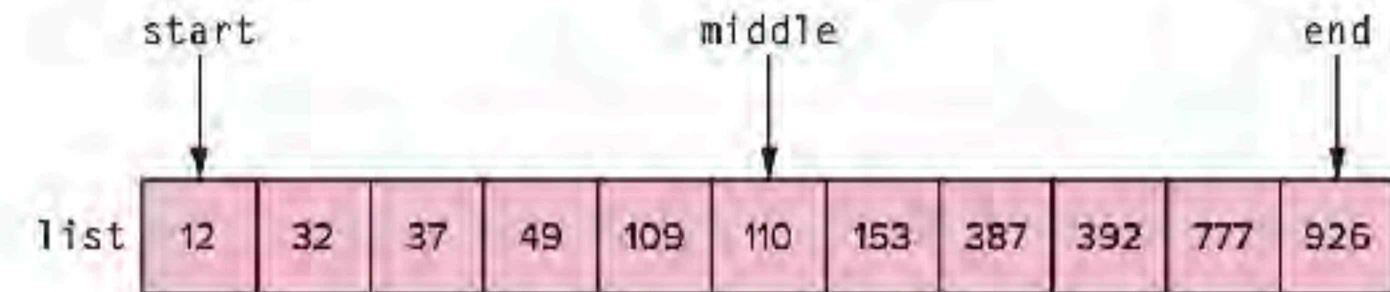


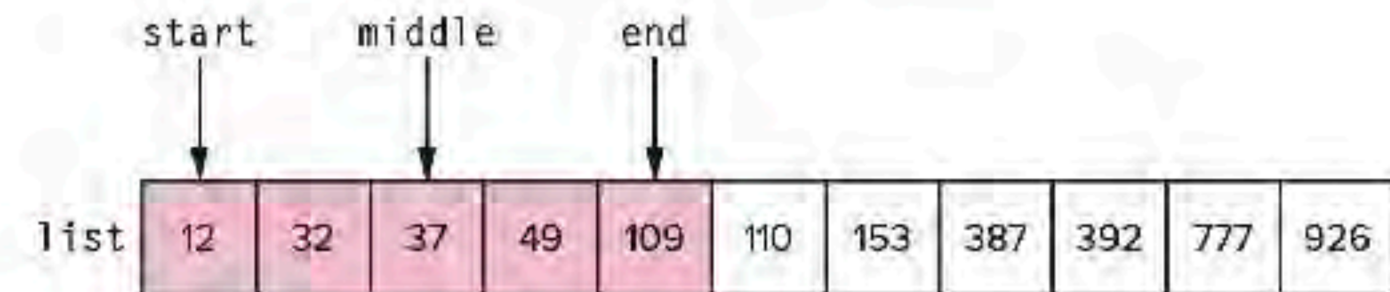
Pick middle

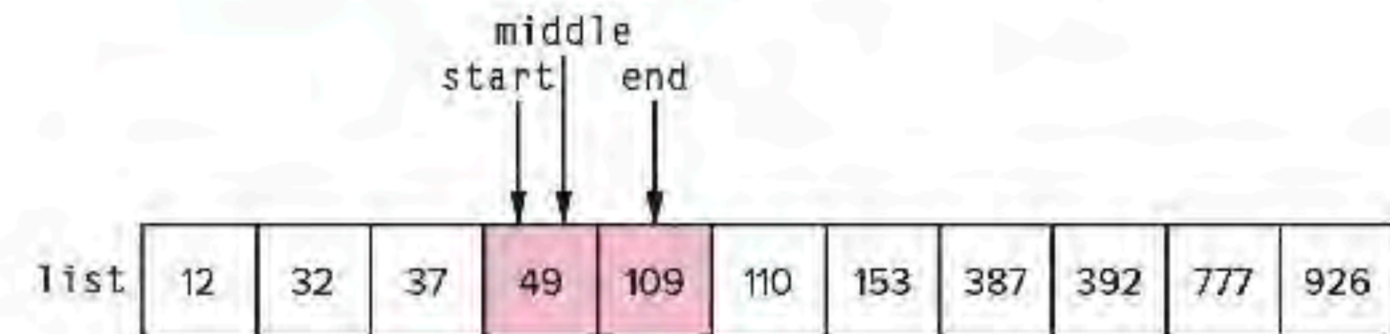
Is middle == key?

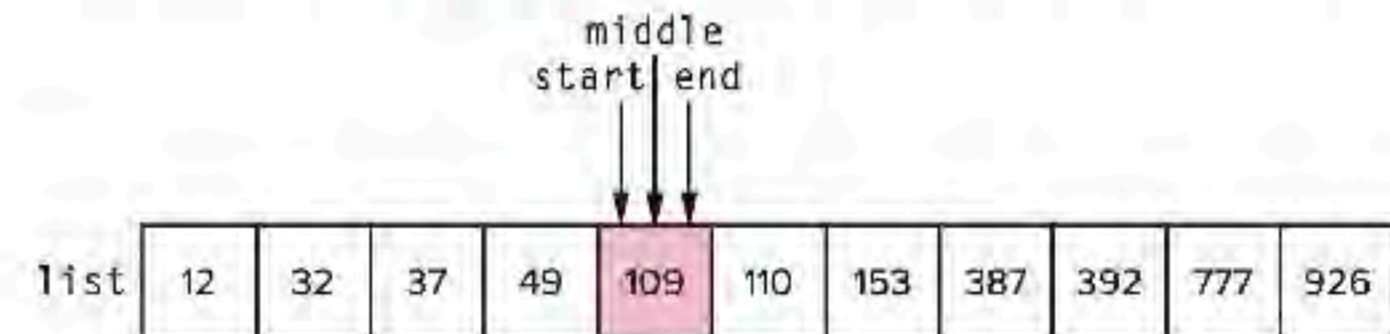


Recap binary search





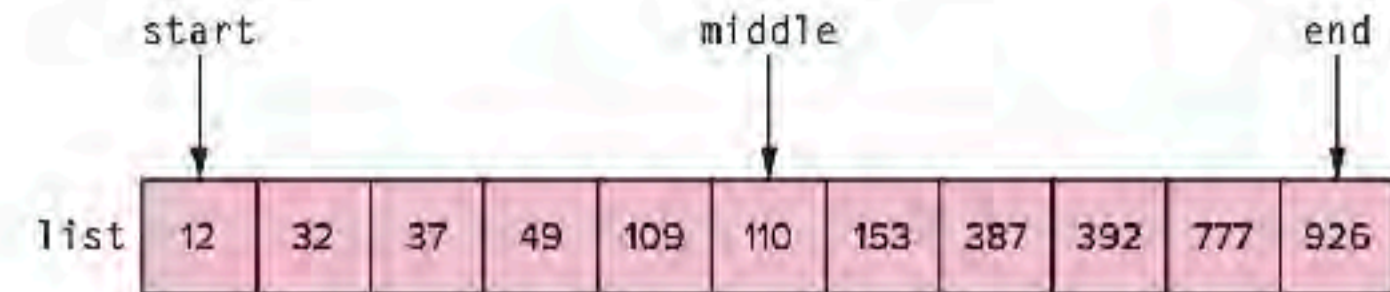


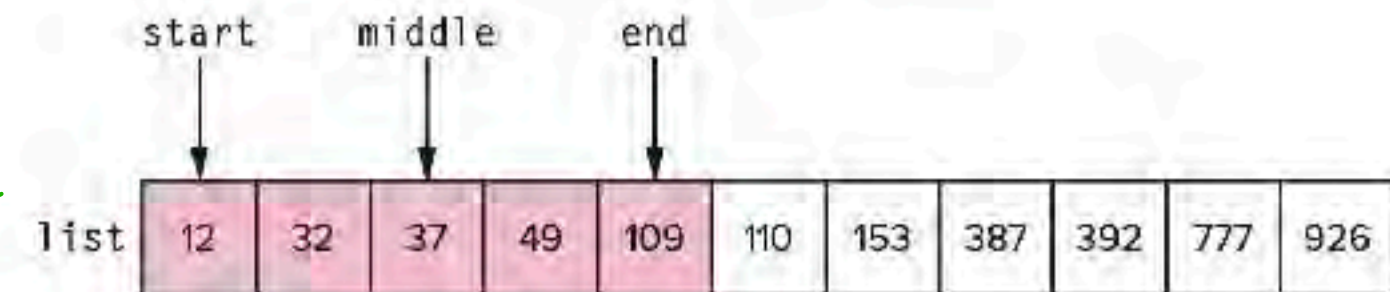


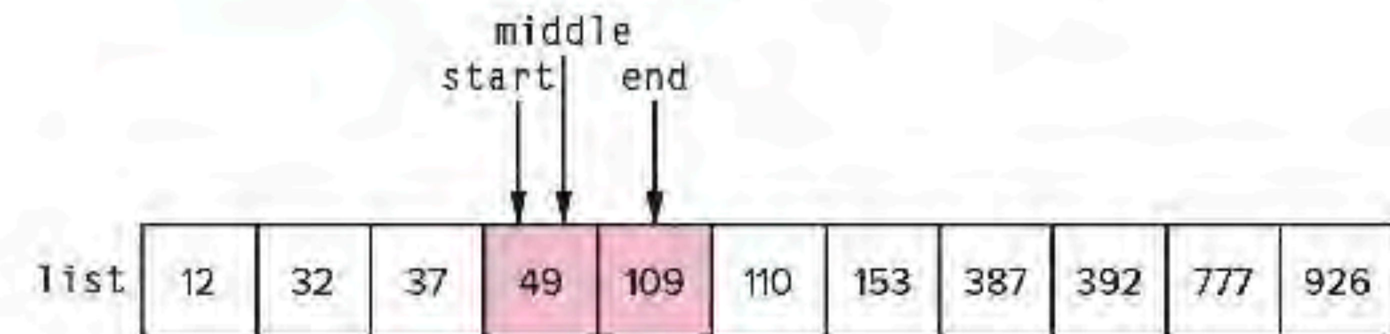
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

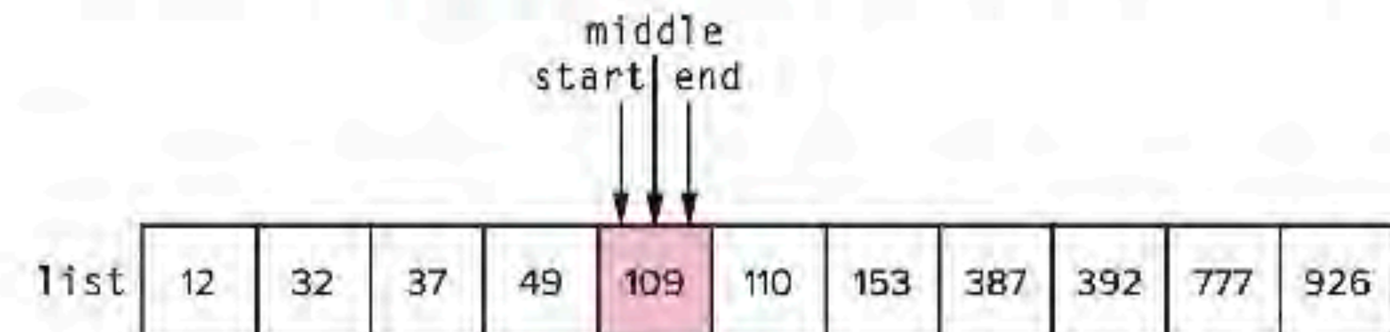
Recap binary search

```
int binary(int arr[], int n, int key){
  int start = 0;       // Left pointer
  int end = _____; // Right pointer

  while (end >= start){
    int mid = (___________) / 2; // Pick middle element

    // Logic to focus search on left or right of mid
    if (key == arr[mid])
      return mid;
    else if (key < arr[mid])
      end = _______;
    else
      start = ________;
  }
  return -1; // Loop exited, element not present.
}
```



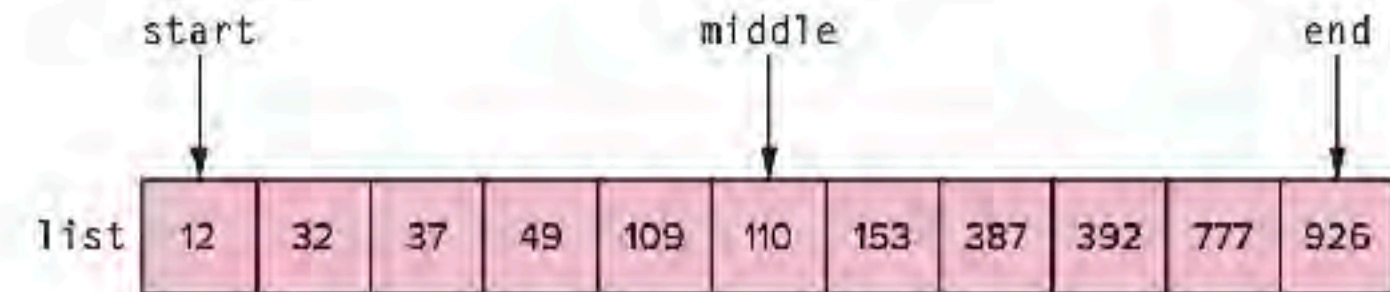


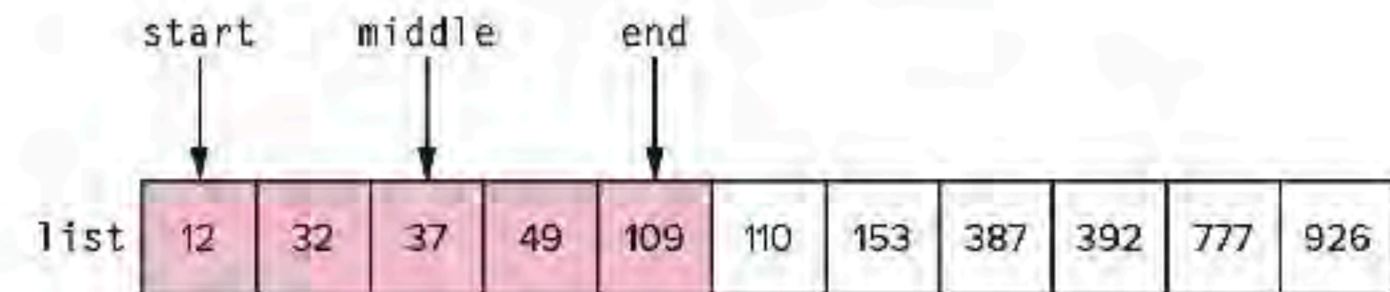


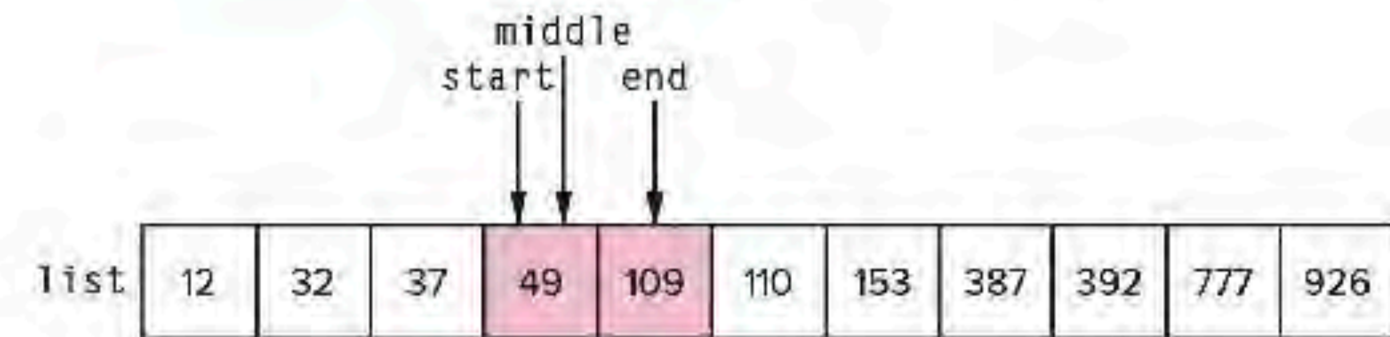


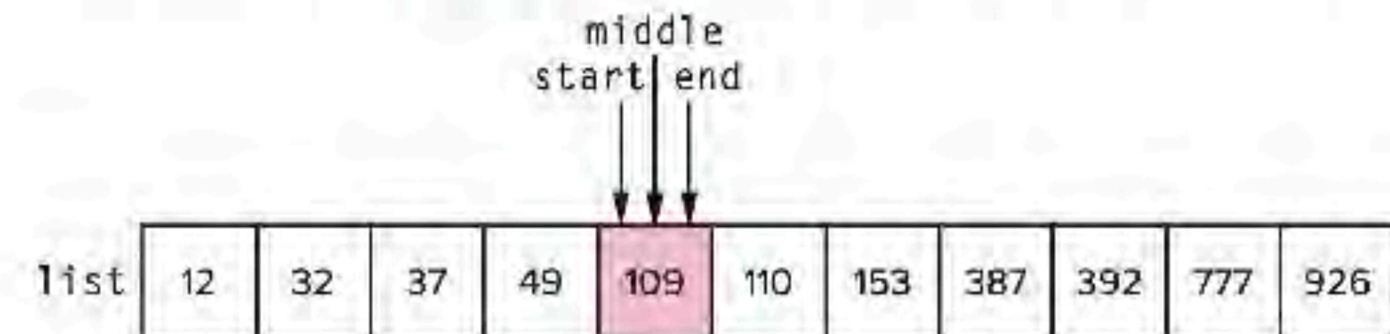
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Binary search





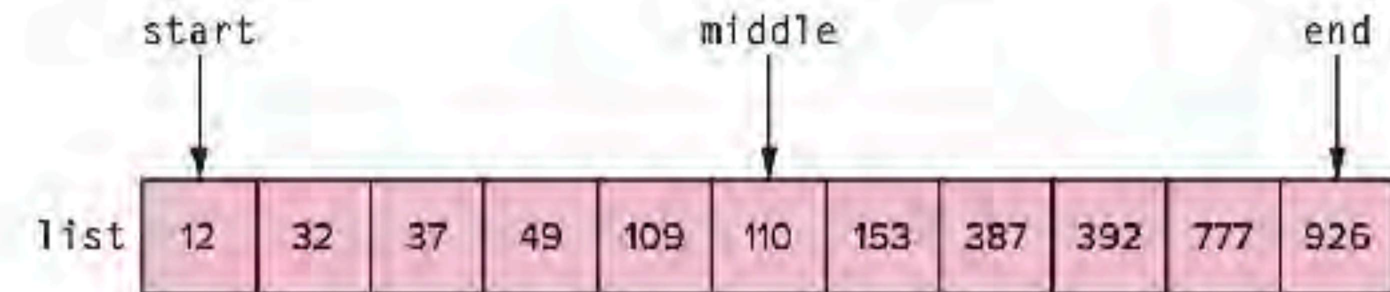


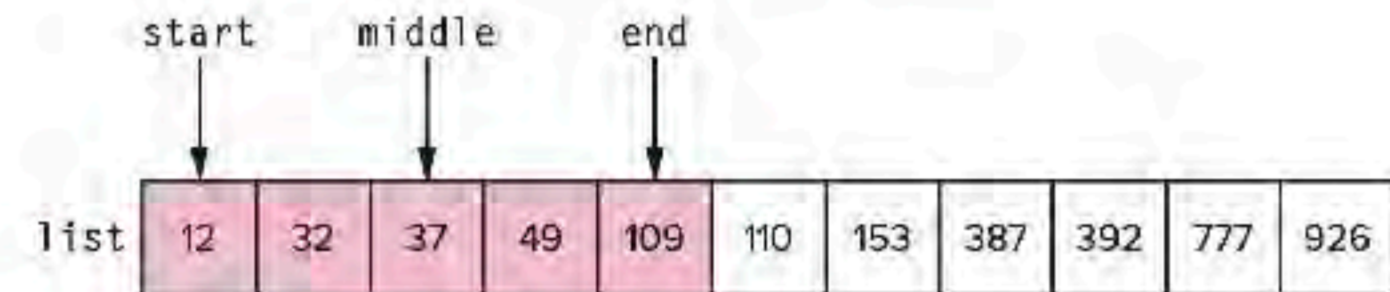


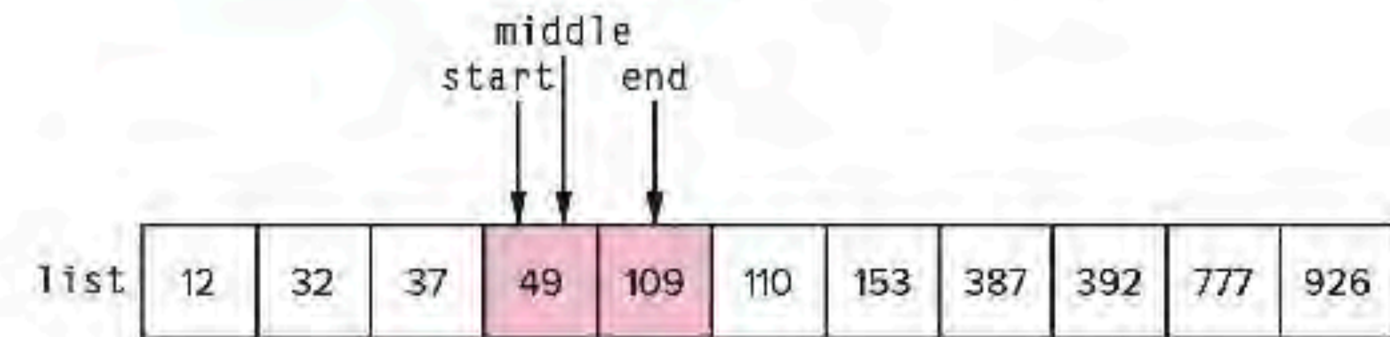
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

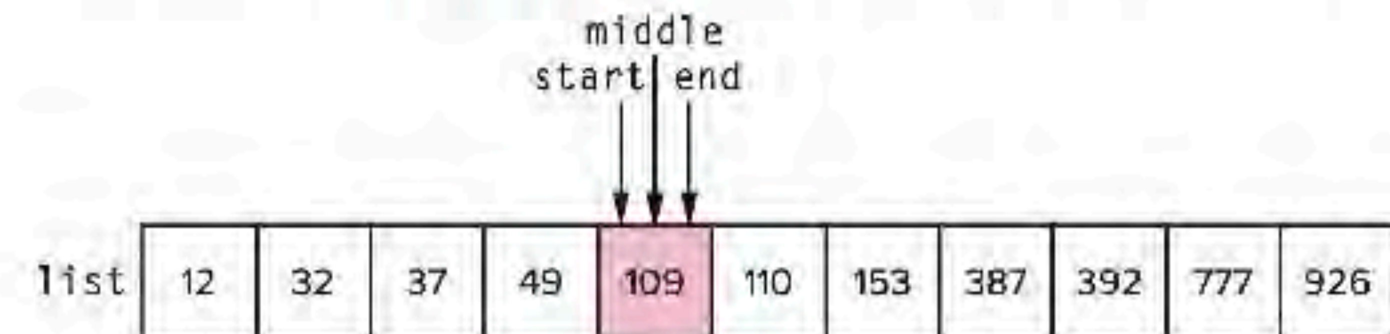
Binary search

- We are repeating the same process of finding `mid` and going left or right of `mid` on each *subarray*.





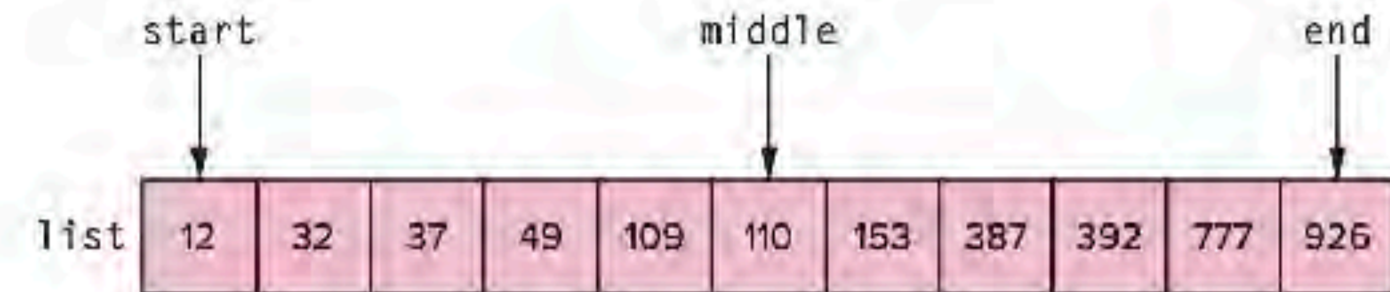


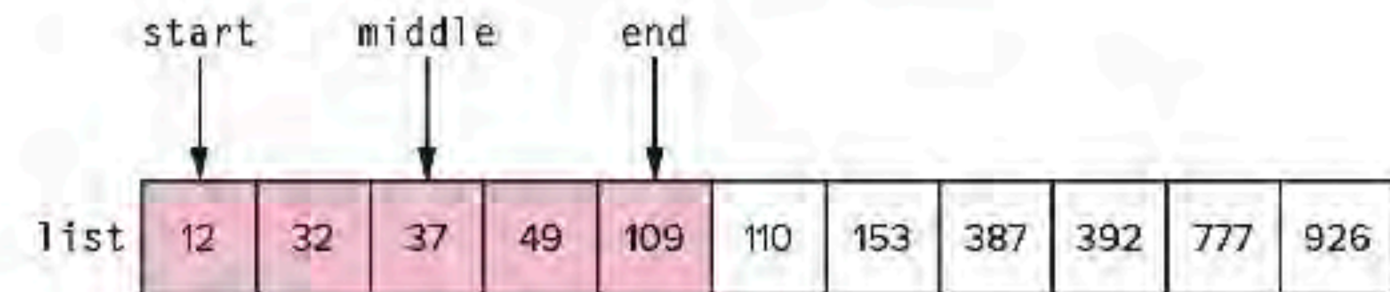


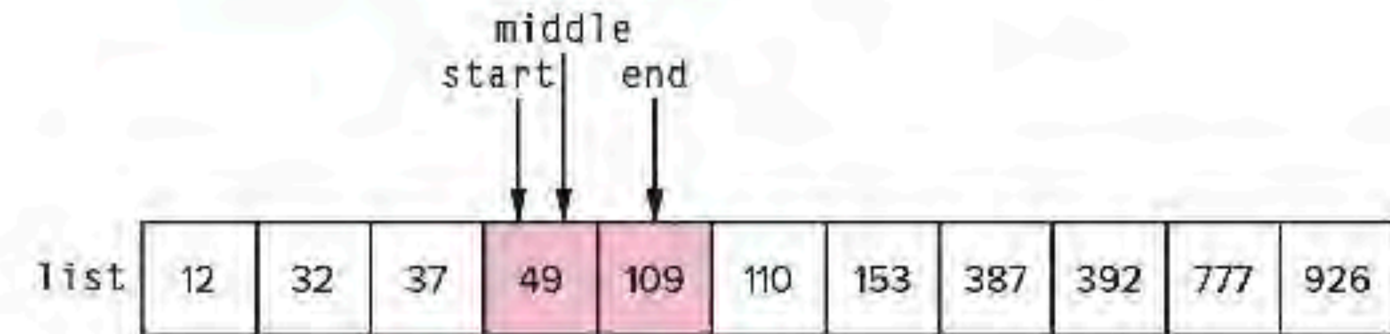
Binary search

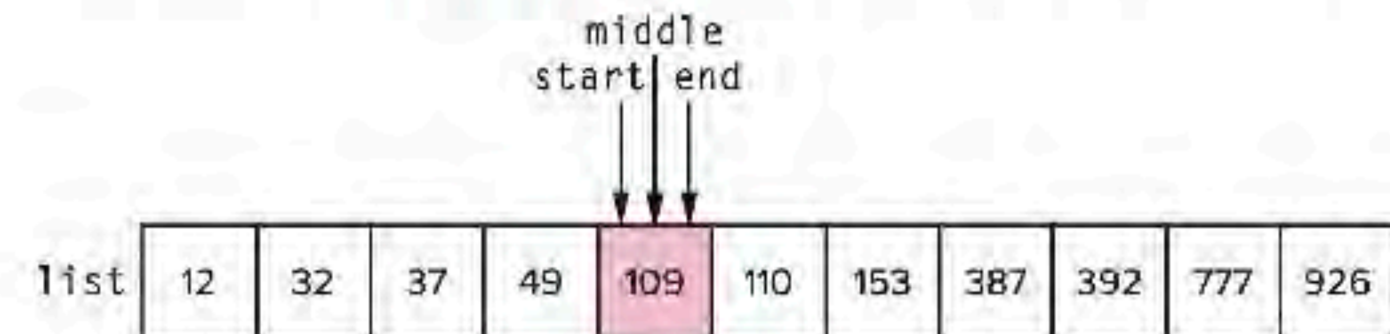
- We are repeating the same process of finding `mid` and going left or right of `mid` on each *subarray.*

- Can we apply







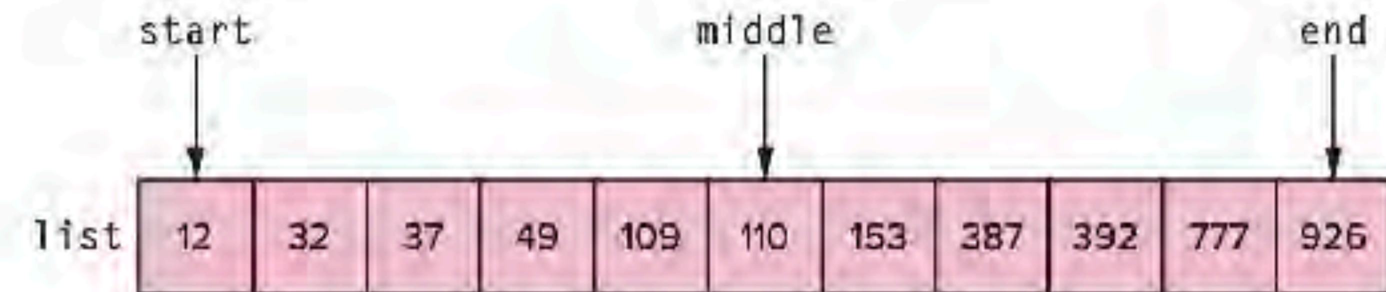


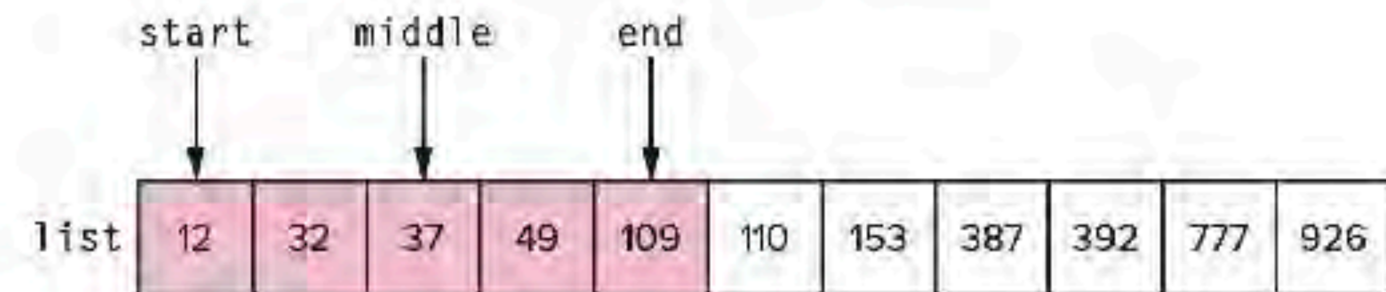
Binary search

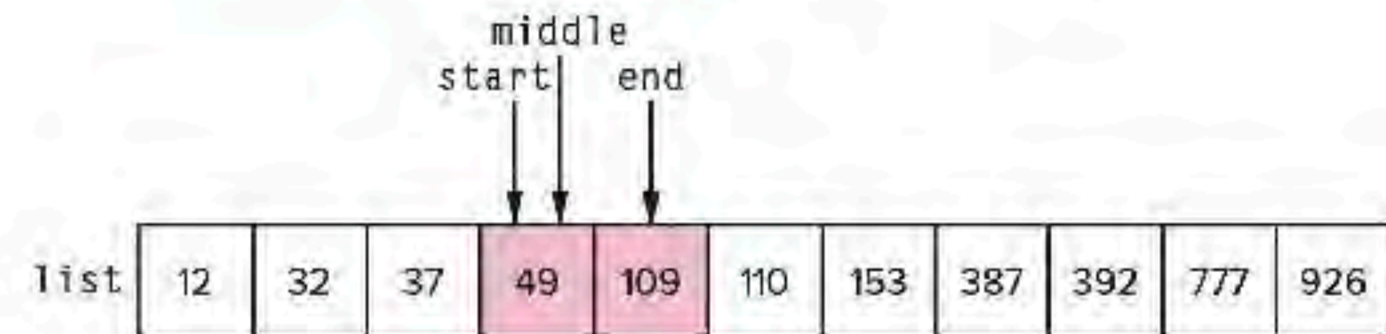
- We are repeating the same process of finding `mid` and going left or right of `mid` on each *subarray.*

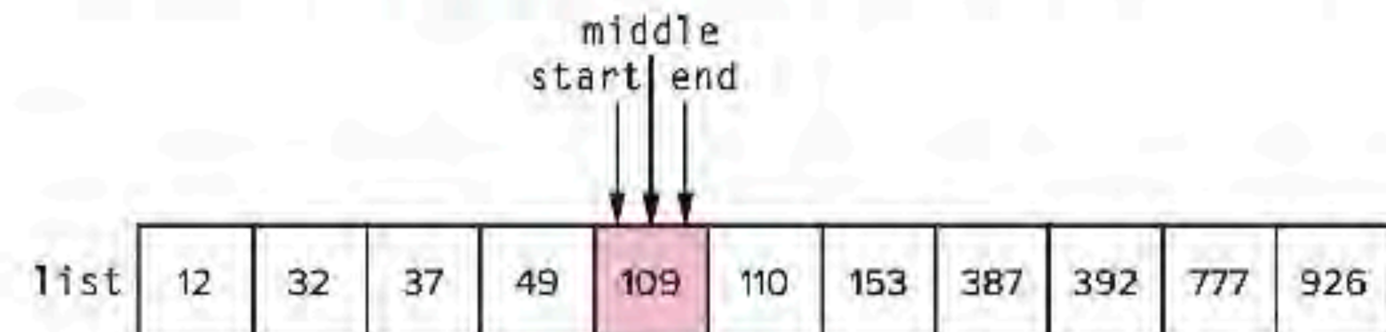
- Can we apply

```
binary(arr[], n, key)
```









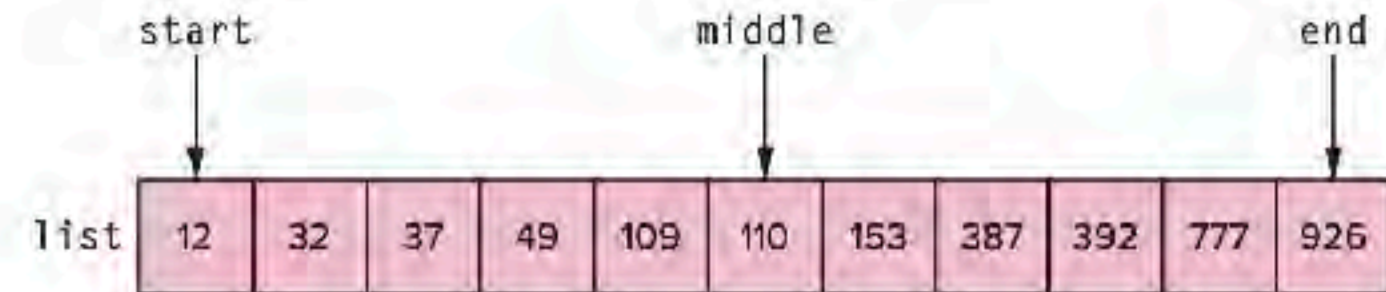
Binary search

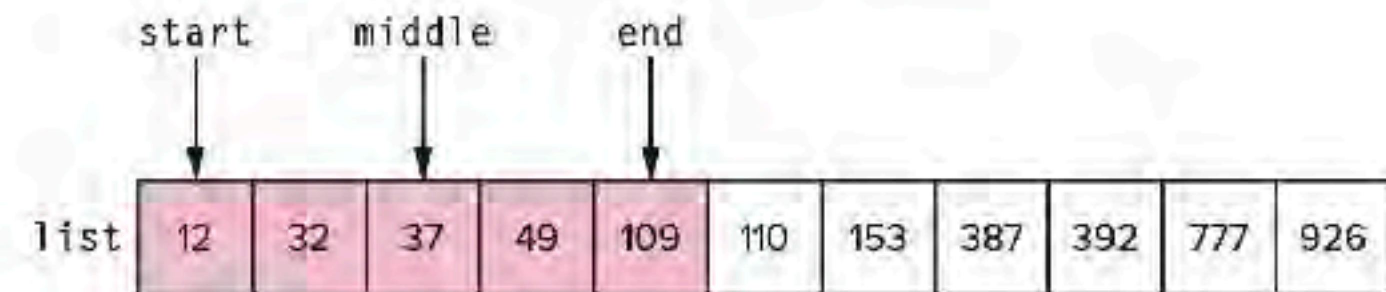
- We are repeating the same process of finding `mid` and going left or right of `mid` on each *subarray.*

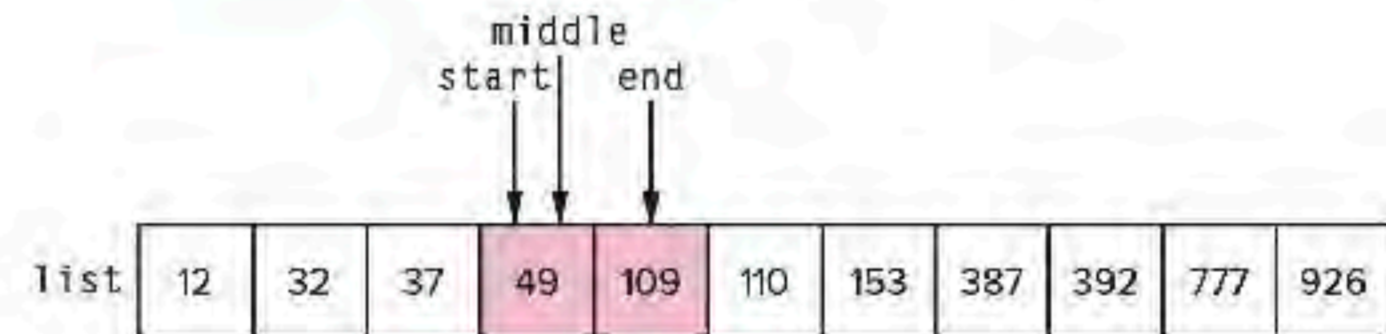
- Can we apply

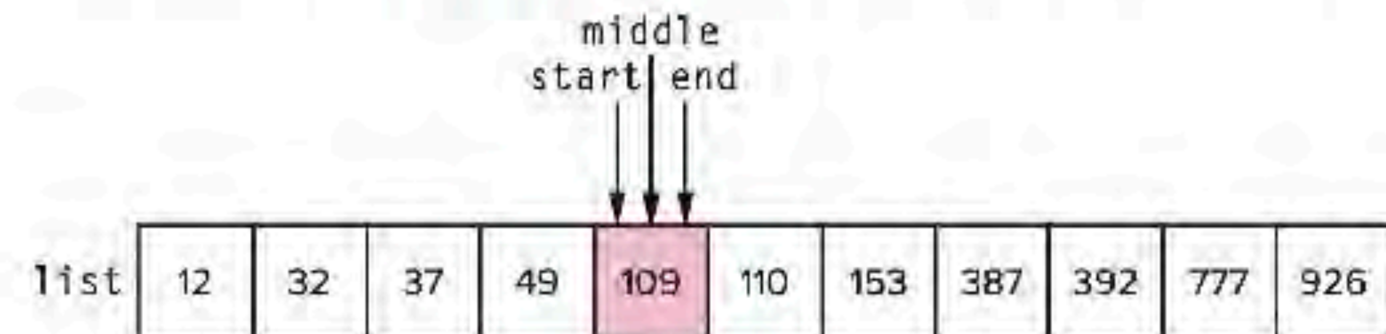
```
binary(arr[], n, key)
```

on each subarray?









UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Binary search

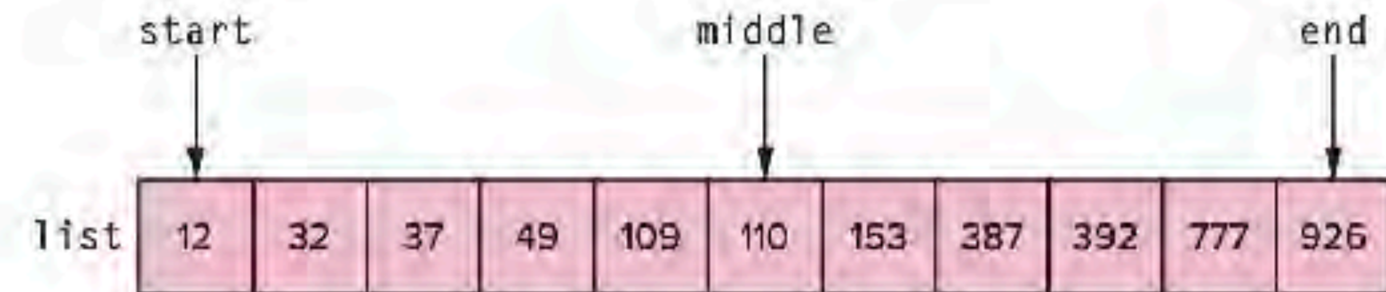
- We are repeating the same process of finding `mid` and going left or right of `mid` on each *subarray.*

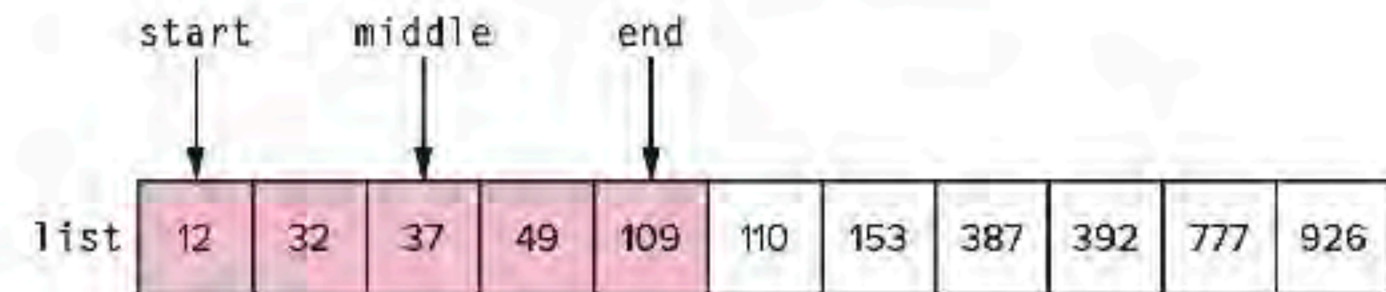
- Can we apply

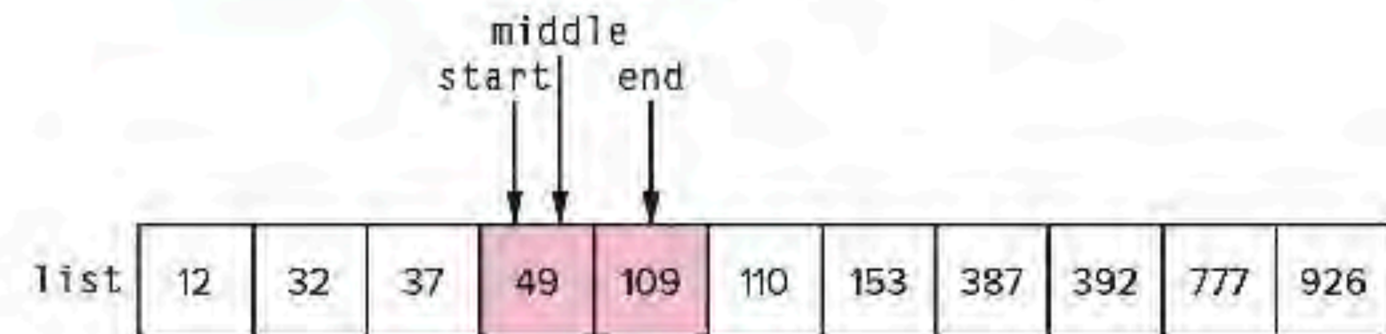
```
binary(arr[], n, key)
```

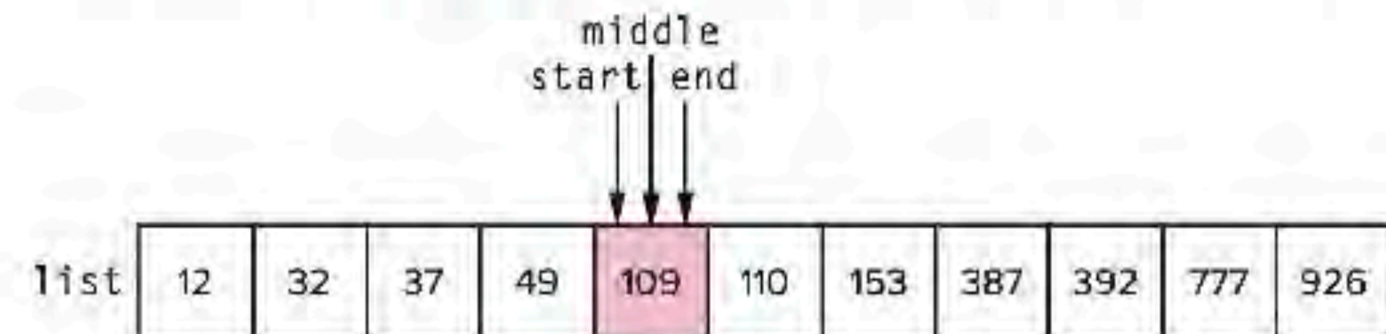
on each subarray?

- Idea is called *recursion*.









UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion

Recursion

A **recursive function** is one that solves its task by **calling itself** on smaller pieces of data.

Recursion

A **recursive function** is one that solves its task by **calling itself** on smaller pieces of data.

- Similar to recurrence function in mathematics

Recursion

A **recursive function** is one that solves its task by **calling itself** on smaller pieces of data.

- Similar to recurrence function in mathematics
- Like iteration — can be used interchangeably; sometimes recursion results in simpler solution … but not always!

Recursion

A **recursive function** is one that solves its task by **calling itself** on smaller pieces of data.

- Similar to recurrence function in mathematics
- Like iteration — can be used interchangeably; sometimes recursion results in simpler solution … but not always!
- Must have **at least** *one* base case (terminal case) that ends the recursive process; similar to loop needing condition to exit.

Recursion

A **recursive function** is one that solves its task by **calling itself** on smaller pieces of data.

- Similar to recurrence function in mathematics
- Like iteration — can be used interchangeably; sometimes recursion results in simpler solution … but not always!
- Must have **at least** *one* base case (terminal case) that ends the recursive process; similar to loop needing condition to exit.

Examples: Factorial function, Fibonacci series, binary search, etc.

Recursive function

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursive function

- Base case (= terminating case)

Recursive function

- Base case (= terminating case)
  - This case is **required** so the recurrence can terminate.

Recursive function

- Base case (= terminating case)
  - This case is **required** so the recurrence can terminate.
  - The base case must provide a condition that will eventually become true and returns from the function. **Otherwise, the run-time stack will overflow**.

Recursive function

- Base case (= terminating case)
  - This case is **required** so the recurrence can terminate.
  - The base case must provide a condition that will eventually become true and returns from the function. **Otherwise, the run-time stack will overflow**.
- Recursive case (= induction case)

Recursive function

- Base case (= terminating case)
  - This case is **required** so the recurrence can terminate.
  - The base case must provide a condition that will eventually become true and returns from the function. **Otherwise, the run-time stack will overflow**.
- Recursive case (= induction case)
  - This case returns a recursive call to function itself. It breaks down the problem into smaller chunks that can be solved over and over by the same function.

Recursive function

- Base case (= terminating case)
  - This case is **required** so the recurrence can terminate.
  - The base case must provide a condition that will eventually become true and returns from the function. **Otherwise, the run-time stack will overflow**.
- Recursive case (= induction case)
  - This case returns a recursive call to function itself. It breaks down the problem into smaller chunks that can be solved over and over by the same function.
  - The input to the next call gets induced gradually.

Example: Factorial

Example: Factorial

- Mathematical definition

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form

Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form

$$n! = \begin{cases} 1, & \text{if} \quad n = 1 \\ n \cdot (n-1)!, & \text{else} \end{cases}$$

Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

$\boxed{4!}$

- Recursive form

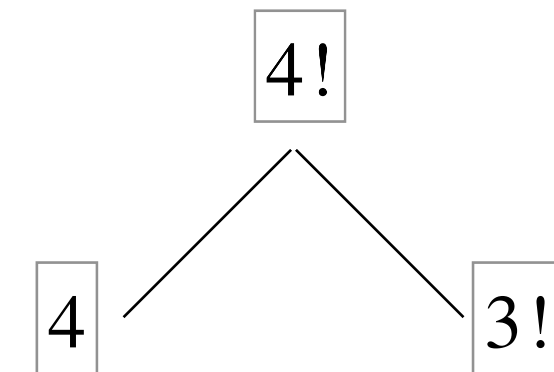
$$n! = \begin{cases} 1, & \text{if} \quad n = 1 \\ n \cdot (n-1)!, & \text{else} \end{cases}$$

Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form



$$n! = \begin{cases} 1, & \text{if} \quad n = 1 \\ n \cdot (n-1)!, & \text{else} \end{cases}$$

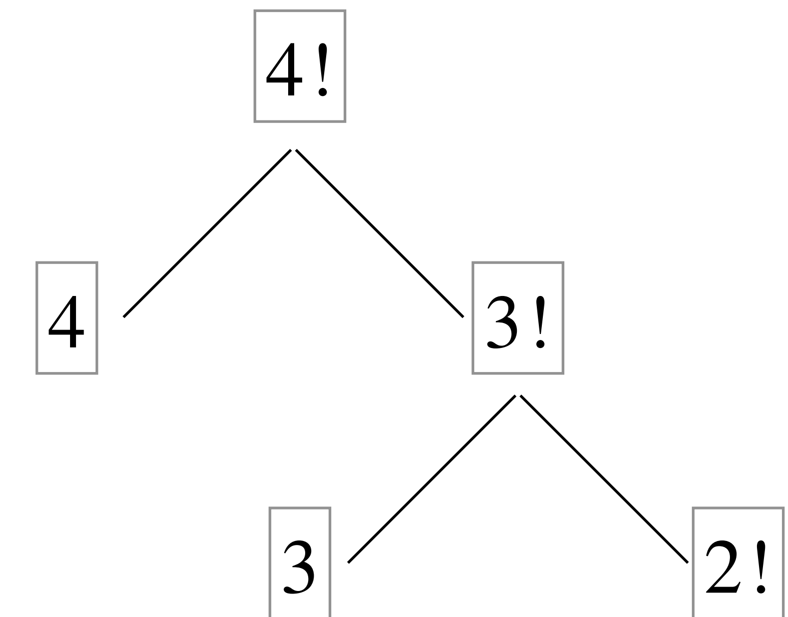
Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form

$$n! = \begin{cases} 1, & \text{if} \quad n = 1 \\ n \cdot (n-1)!, & \text{else} \end{cases}$$



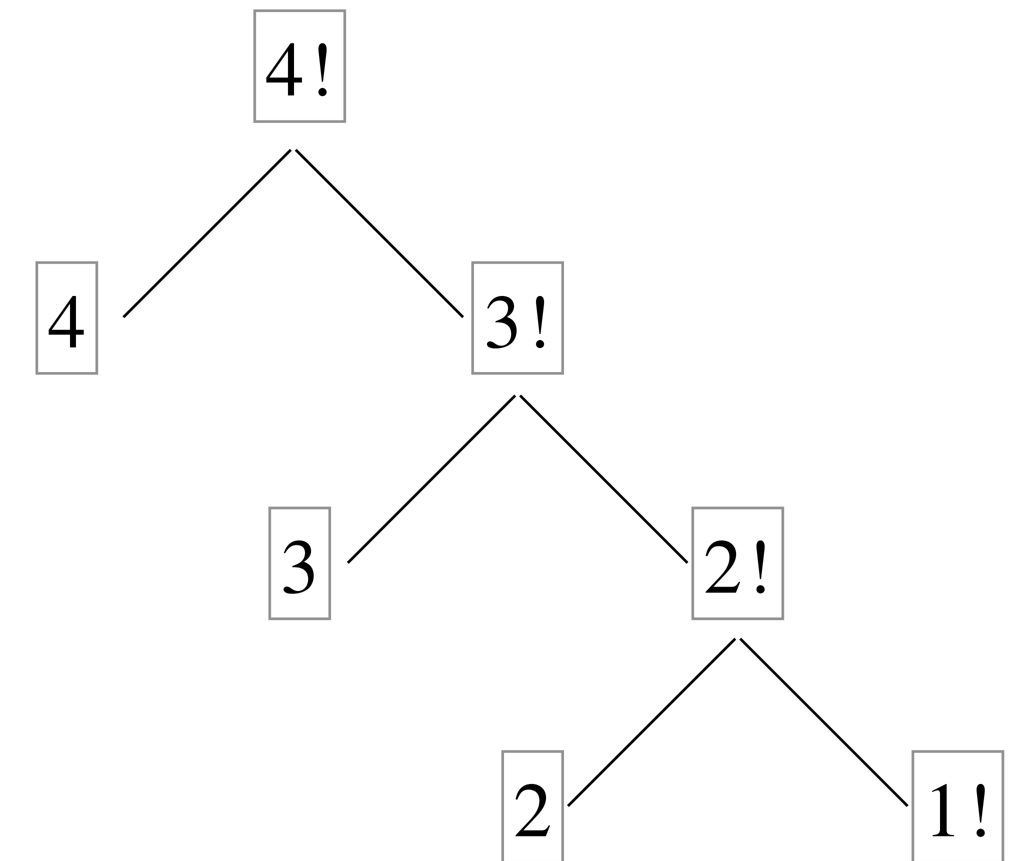
Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form

$$n! = \begin{cases} 1, & \text{if} \quad n = 1 \\ n \cdot (n-1)!, & \text{else} \end{cases}$$



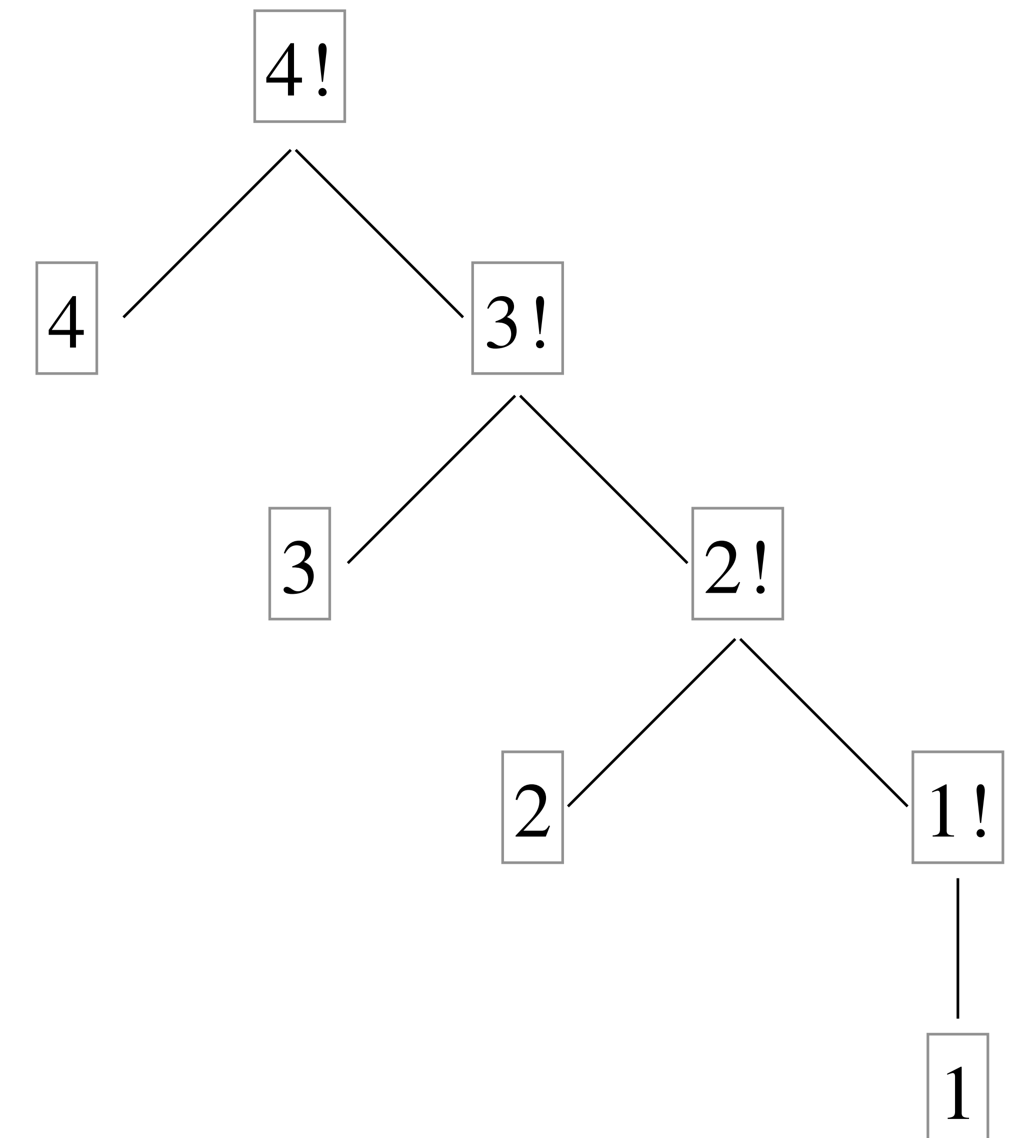
Example: Factorial

- Mathematical definition

$$n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1 \qquad for \quad n > 0$$

- Recursive form

$$n! = \begin{cases} 1, & \text{if} \quad n = 1 \\ n \cdot (n-1)!, & \text{else} \end{cases}$$



Example: Factorial

Example: Factorial

```
int Factorial(int n){
    if (n == 1)
        return 1;
    else
        return n * Factorial(n - 1);
}
```

Example: Factorial

```
fn = Factorial(4);
```

```
int Factorial(int n){
     if (n == 1)
          return 1;
     else
          return n * Factorial(n - 1);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Factorial

```
fn = Factorial(4);
```

```
int Factorial(int n){
    if (n == 1)
        return 1;
    else
        return n * Factorial(n - 1);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Factorial

```
fn = Factorial(4);
```

```
return 4 * Factorial(3);
```

```
return 3 * Factorial(2);
```

```
int Factorial(int n){
     if (n == 1)
          return 1;
     else
          return n * Factorial(n - 1);
}
```

Example: Factorial

```
fn = Factorial(4);
```

```
return 4 * Factorial(3);
```

```
return 3 * Factorial(2);
```

```
return 2 * Factorial(1);
```

```
int Factorial(int n){
    if (n == 1)
        return 1;
    else
        return n * Factorial(n - 1);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Factorial

```
fn = Factorial(4);
```

```
return 4 * Factorial(3);
```

```
return 3 * Factorial(2);
```

```
return 2 * Factorial(1);
```

```
return 1;
```

```
int Factorial(int n){
	if (n == 1)
		return 1;
	else
		return n * Factorial(n - 1);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Factorial

```
fn = Factorial(4);
```

```
return 4 * Factorial(3);
```

```
return 3 * Factorial(2);
```

```
return 2 * Factorial(1);
```

return value = 1

```
return 1;
```

```
int Factorial(int n){
    if (n == 1)
        return 1;
    else
        return n * Factorial(n - 1);
}
```

Example: Factorial

```
fn = Factorial(4);
```

```
return 4 * Factorial(3);
```

```
return 3 * Factorial(2);
```

return value = 2

```
return 2 * Factorial(1);
```

return value = 1

```
return 1;
```

```
int Factorial(int n){
     if (n == 1)
          return 1;
     else
          return n * Factorial(n - 1);
}
```

Example: Factorial

```
fn = Factorial(4);
```

```
return 4 * Factorial(3);
```

return value = 6

```
return 3 * Factorial(2);
```

return value = 2

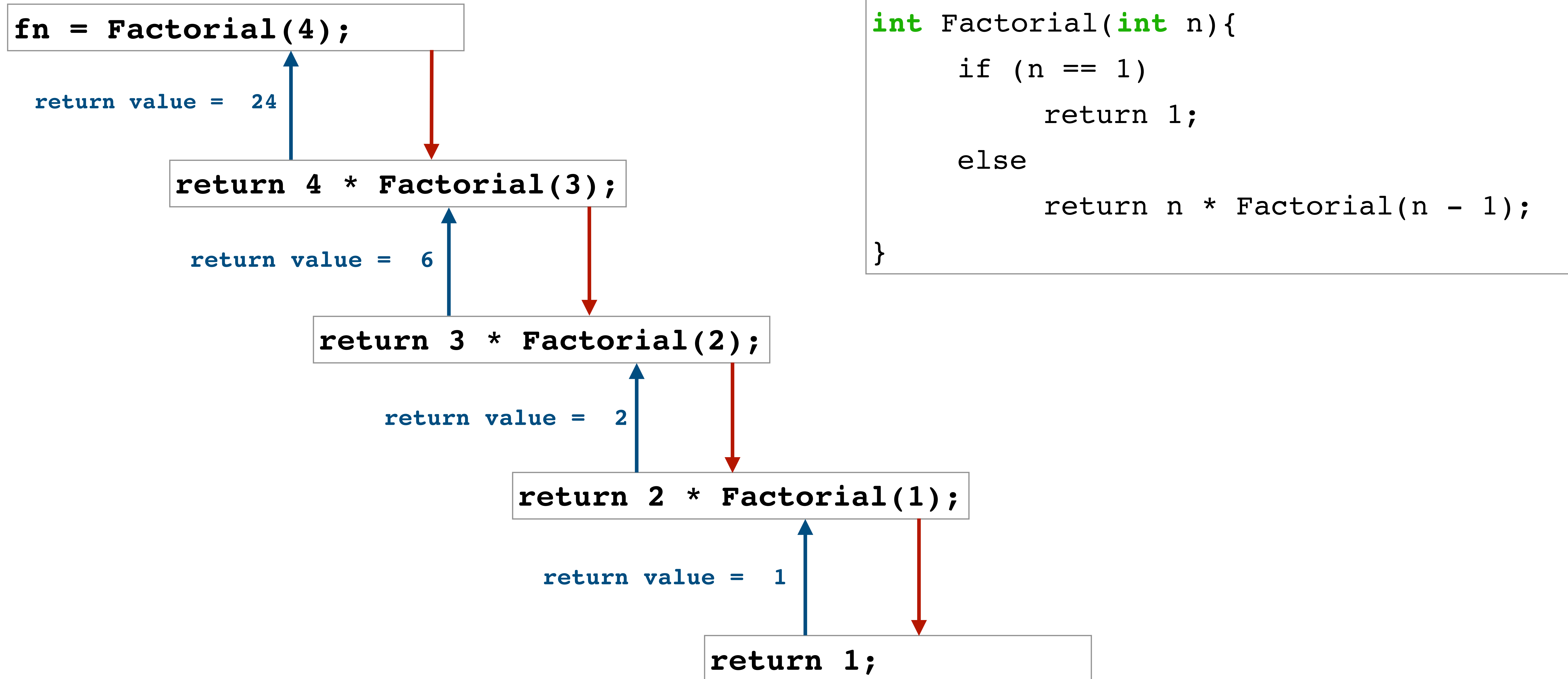
```
return 2 * Factorial(1);
```

return value = 1

```
return 1;
```

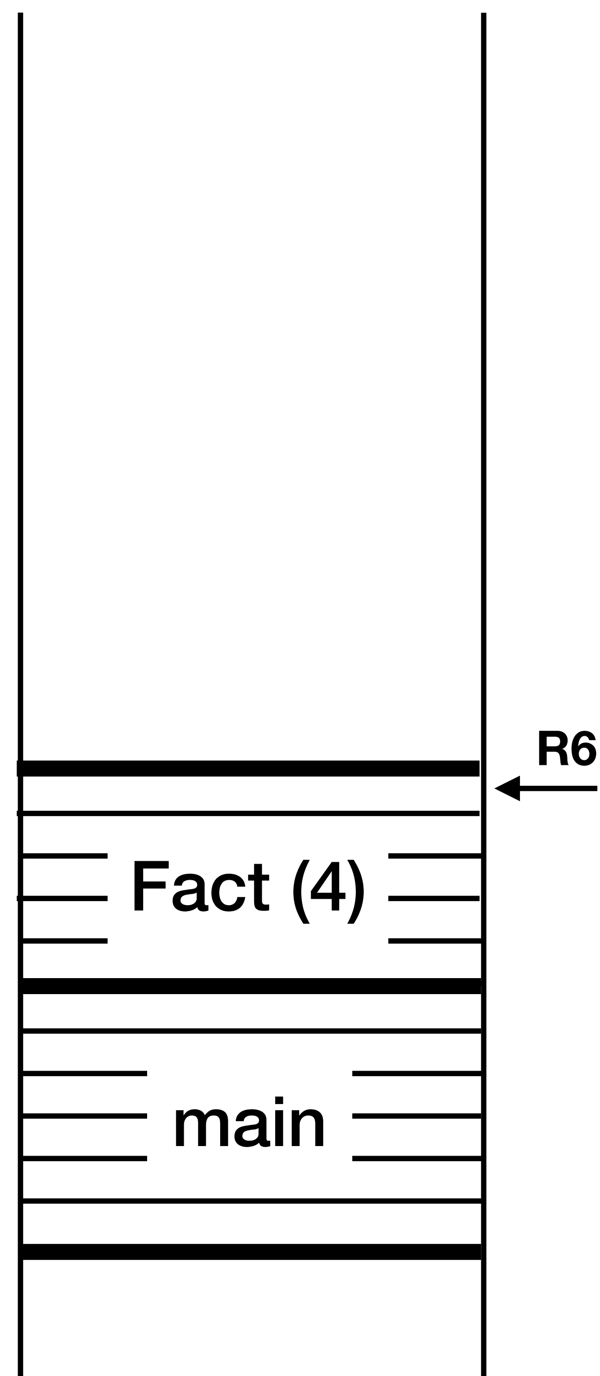
```
int Factorial(int n){
     if (n == 1)
          return 1;
     else
          return n * Factorial(n - 1);
}
```

Example: Factorial



RTS During Execution of Factorial

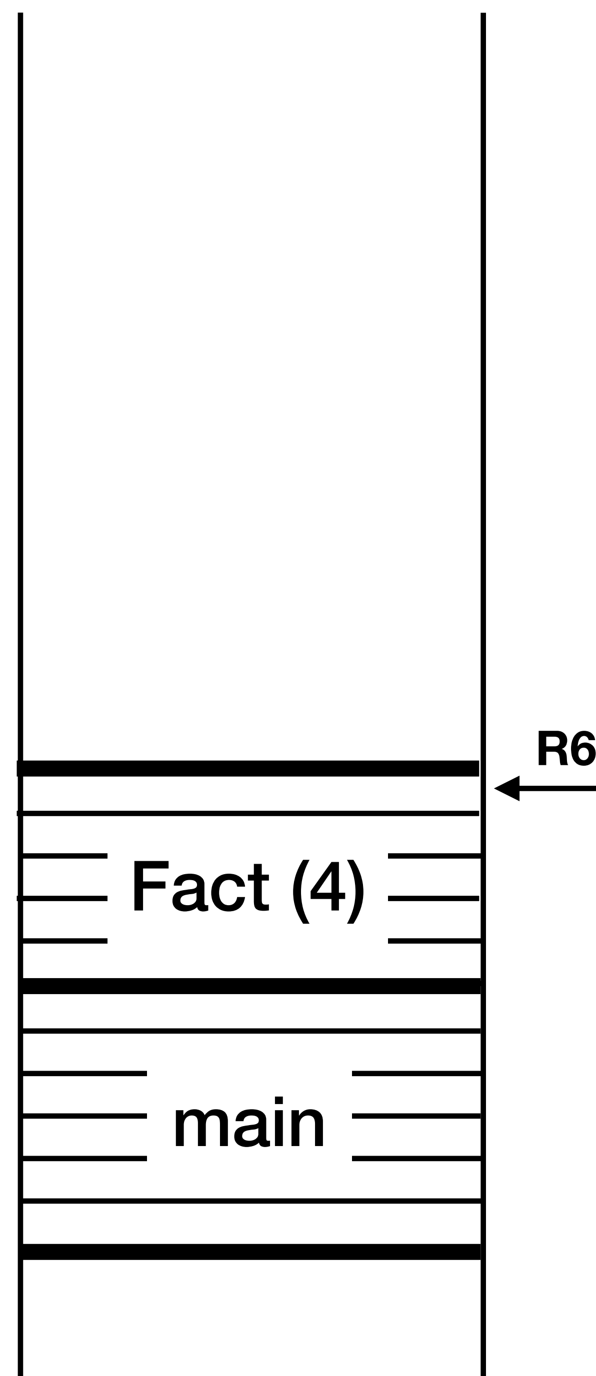
main calls Factorial(4)



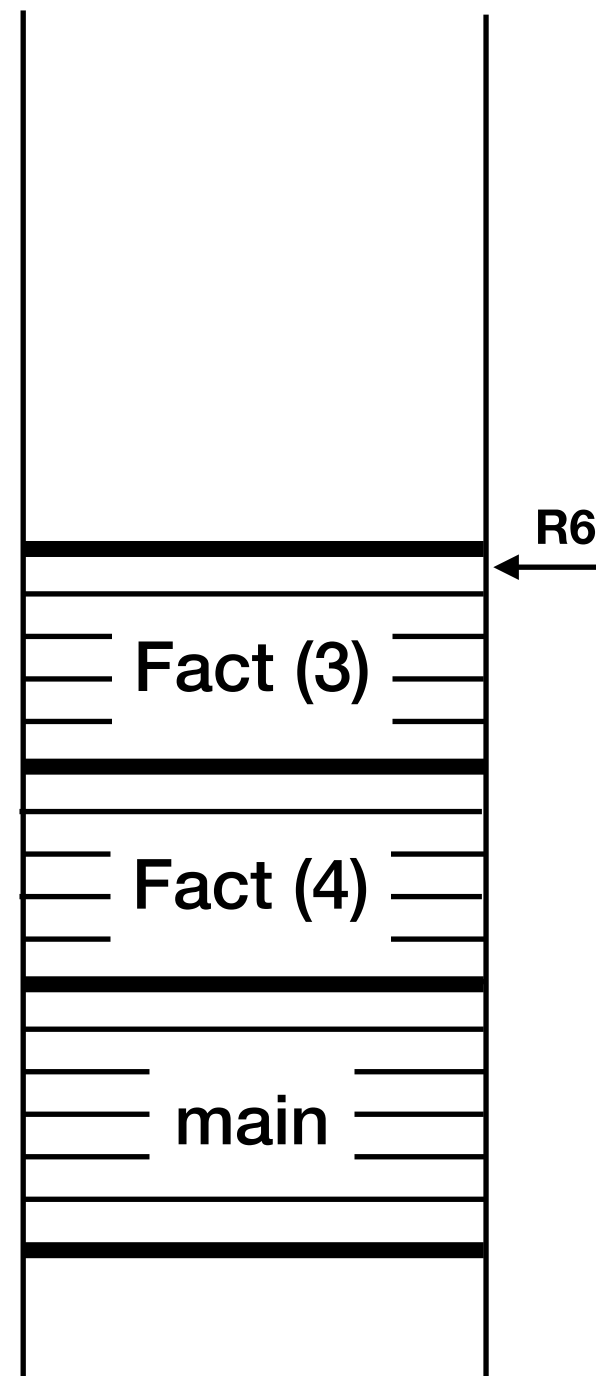
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

RTS During Execution of Factorial

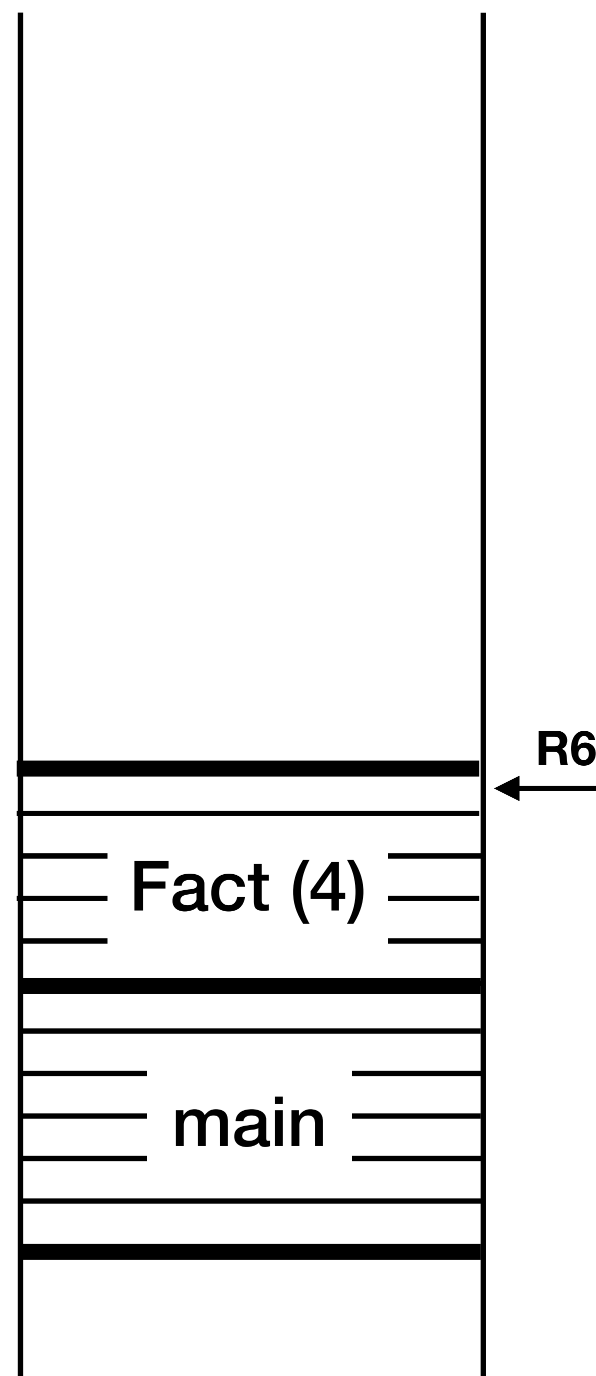
main calls Factorial(4)

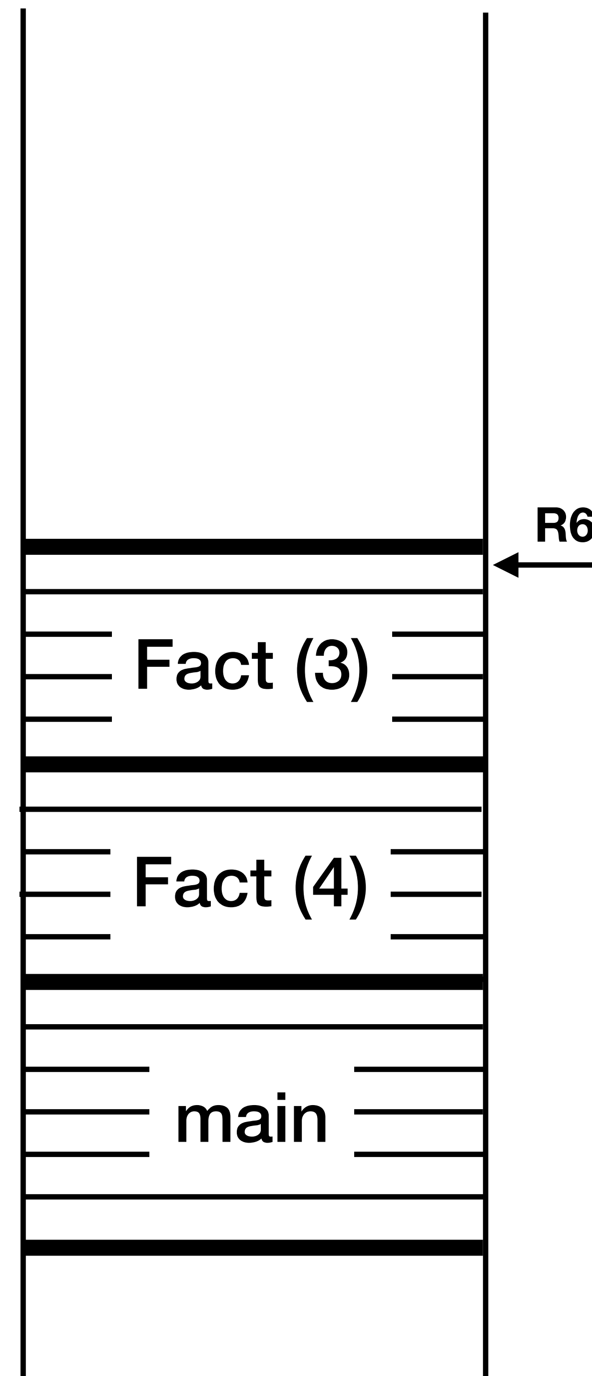


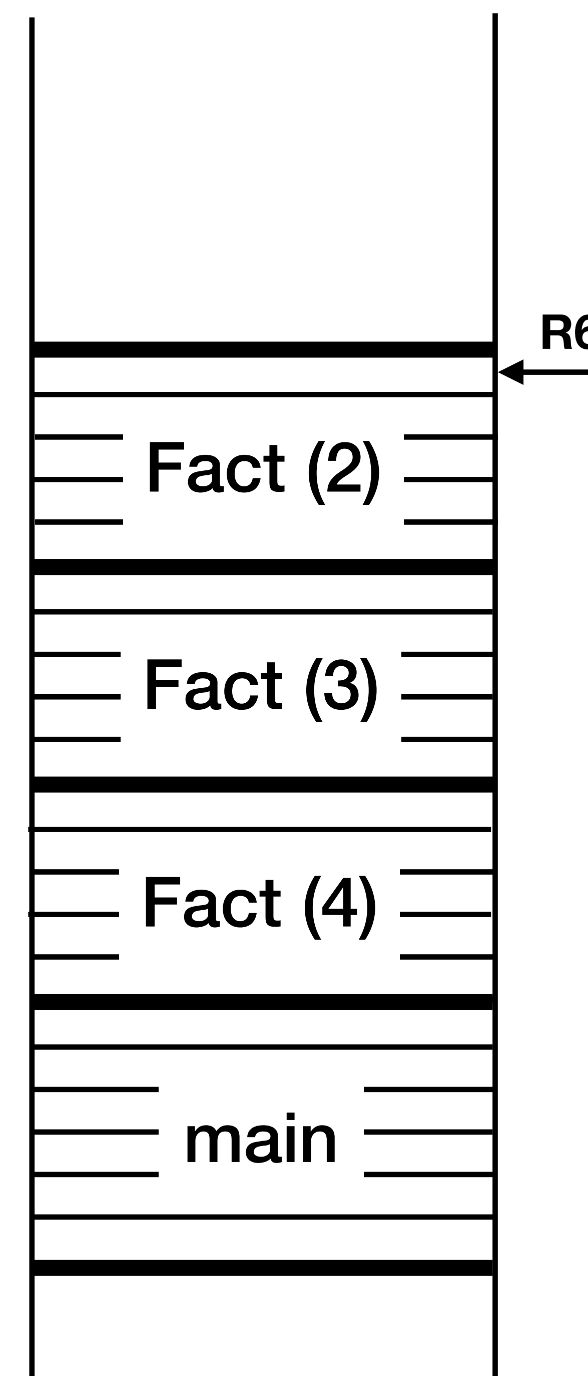
Factorial(4) calls Factorial(3)



RTS During Execution of Factorial

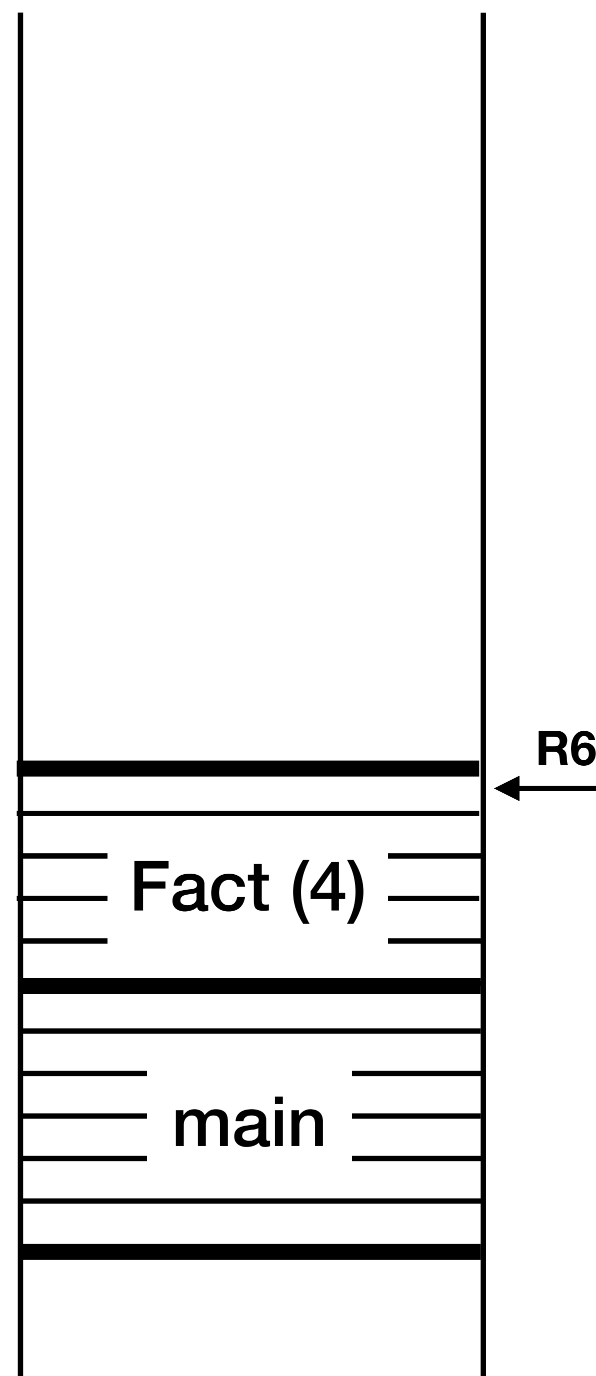




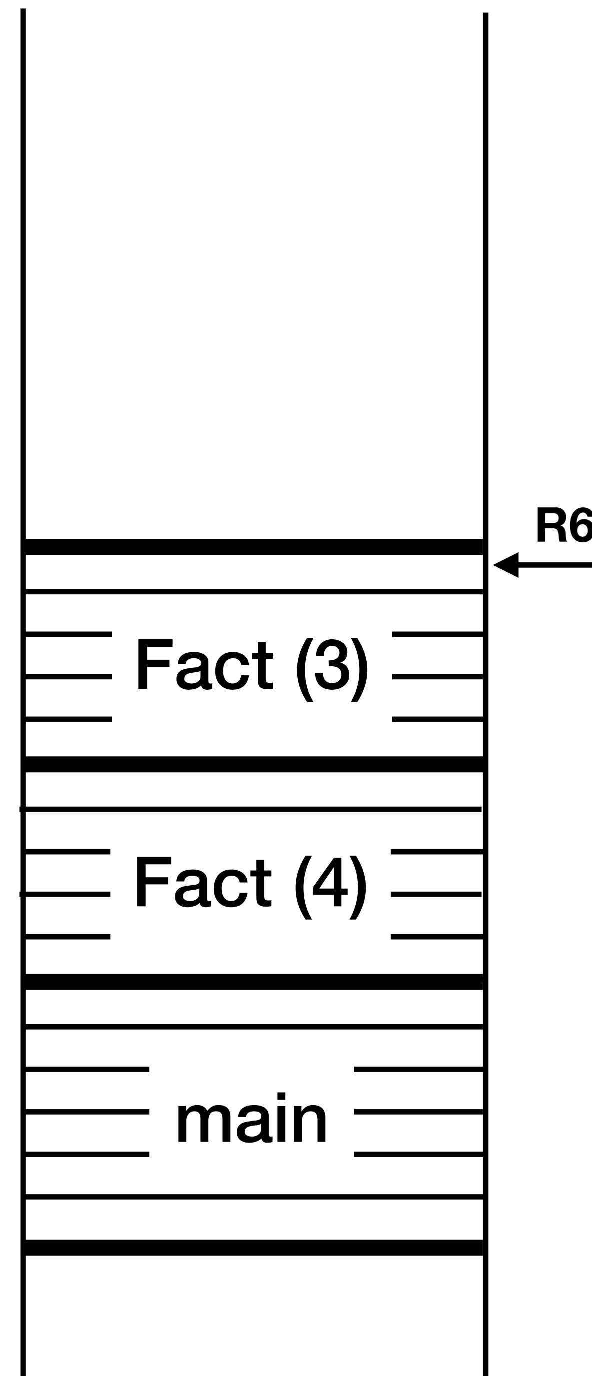


RTS During Execution of Factorial

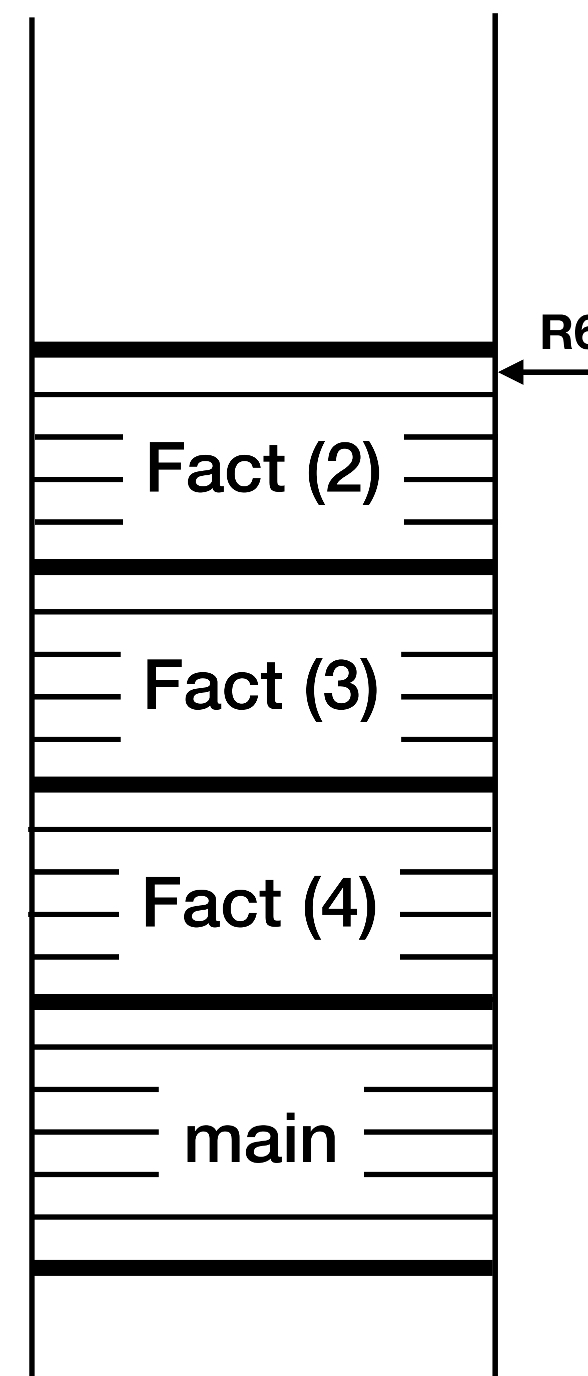
main calls Factorial(4)



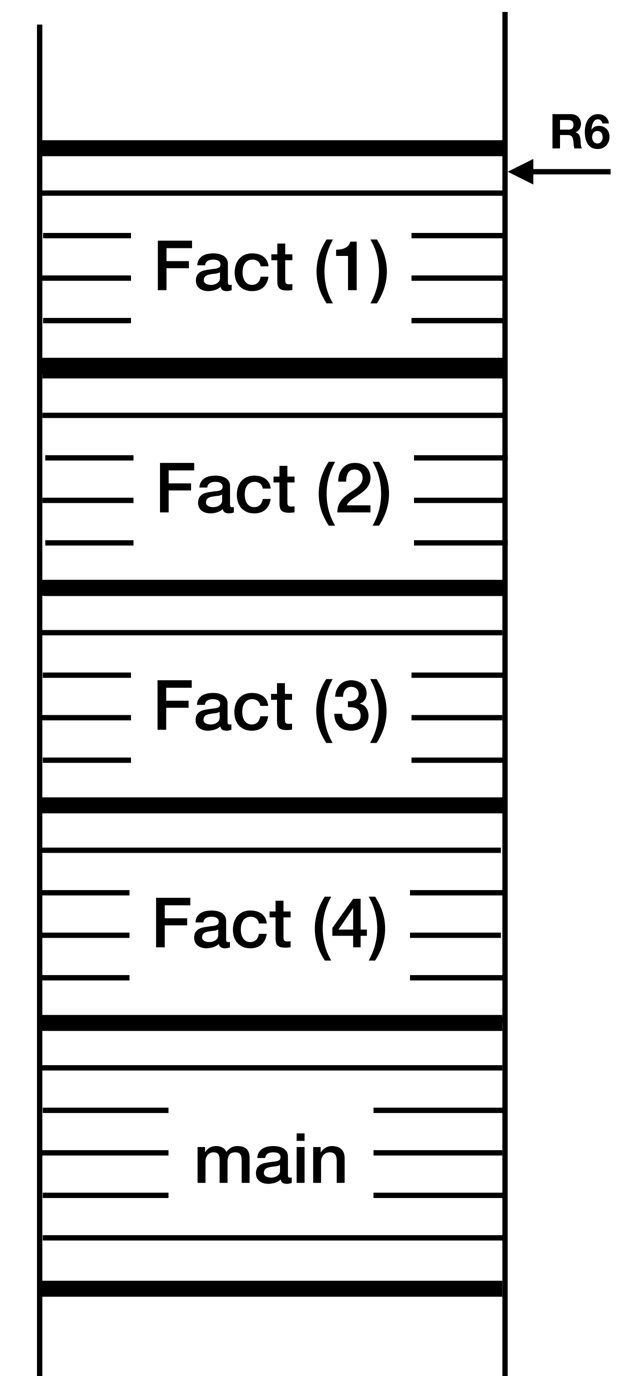
Factorial(4) calls Factorial(3)



Factorial(3) calls Factorial(2)

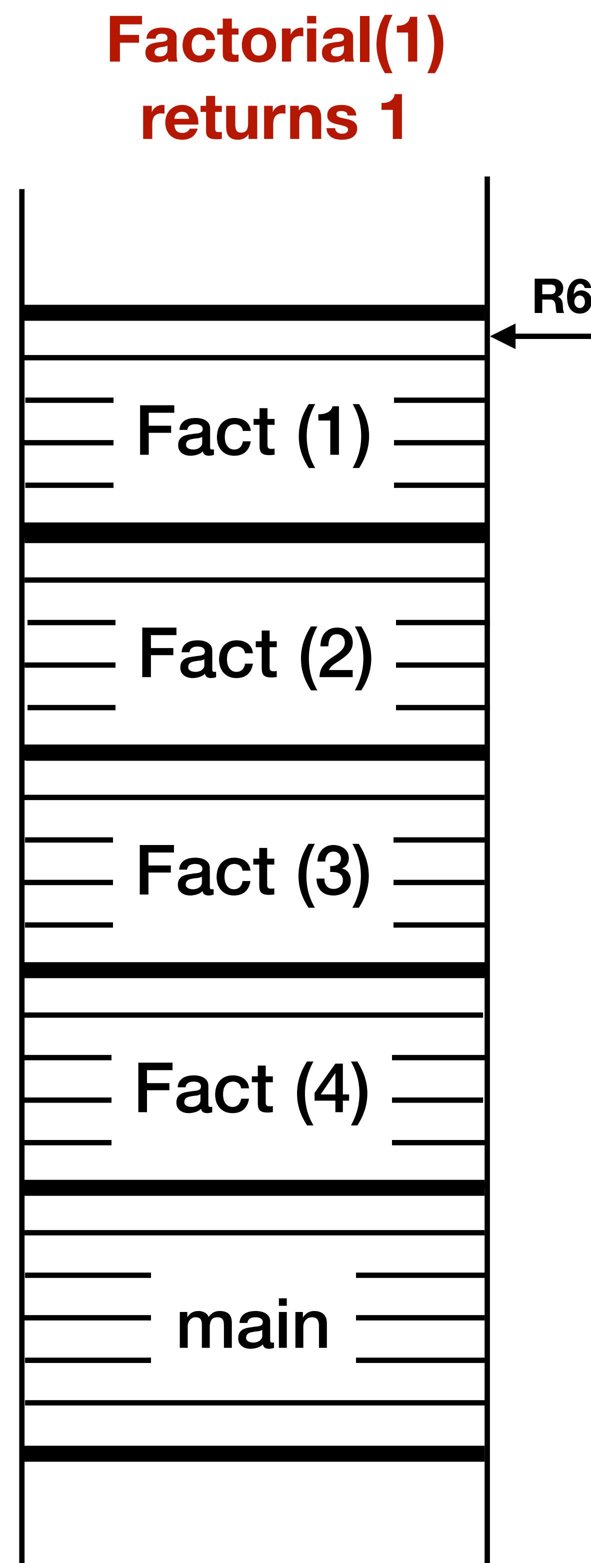


Factorial(2) calls Factorial(1)



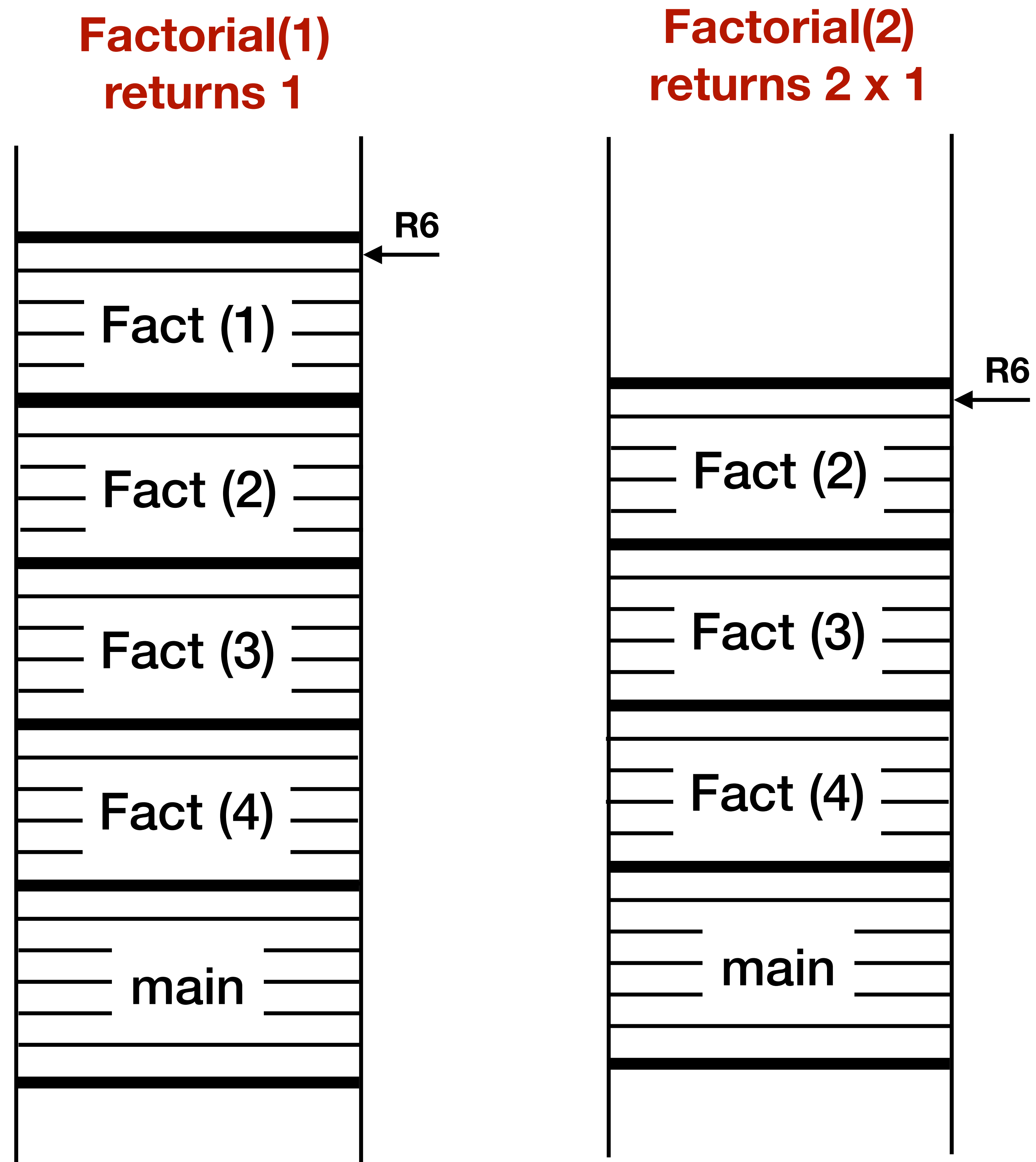
RTS During Execution of Factorial

RTS During Execution of Factorial



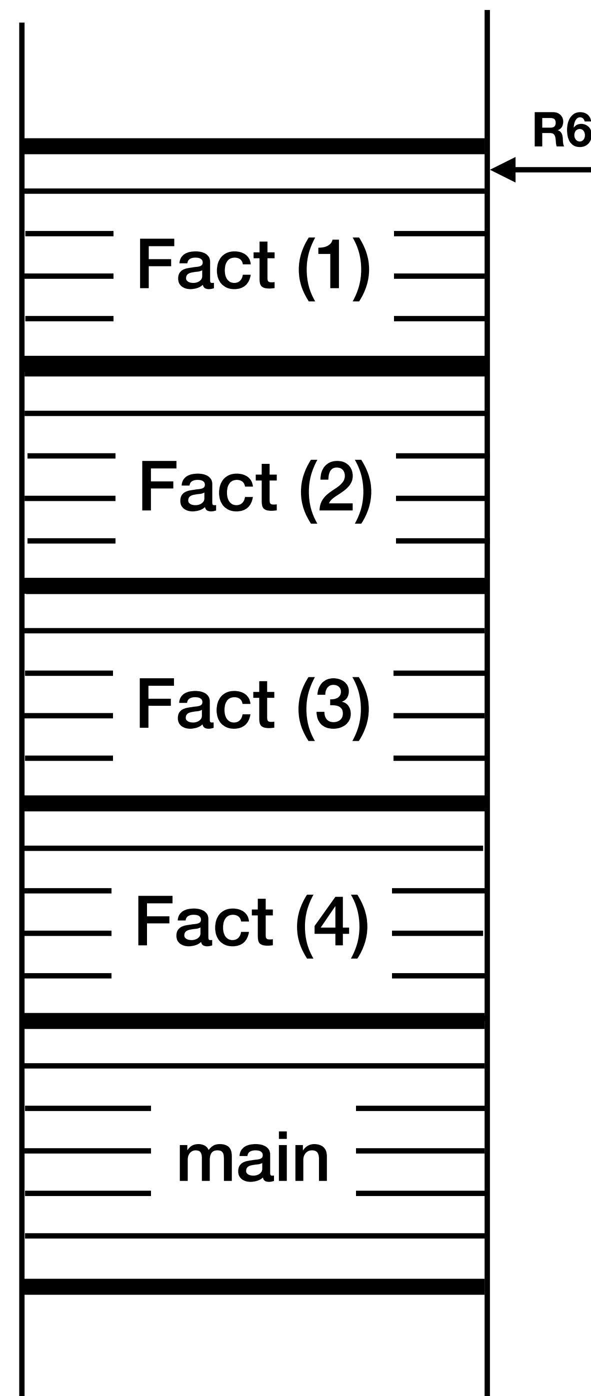
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

RTS During Execution of Factorial

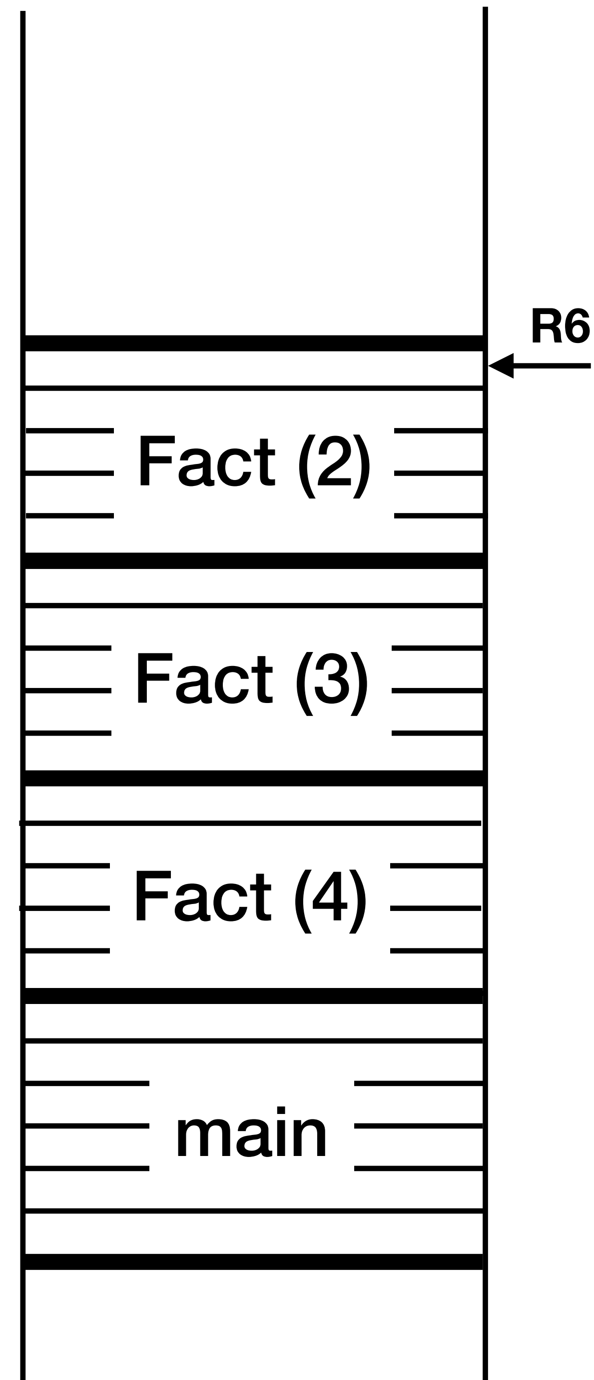


RTS During Execution of Factorial

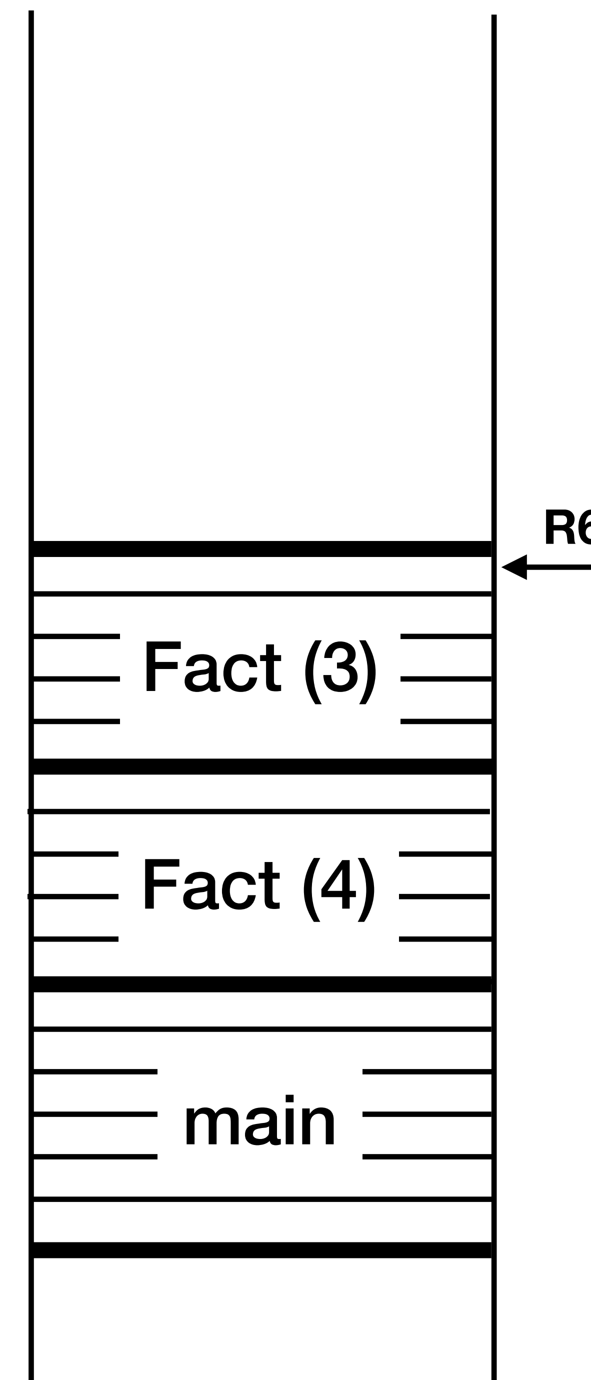
Factorial(1) returns 1



Factorial(2) returns 2 x 1

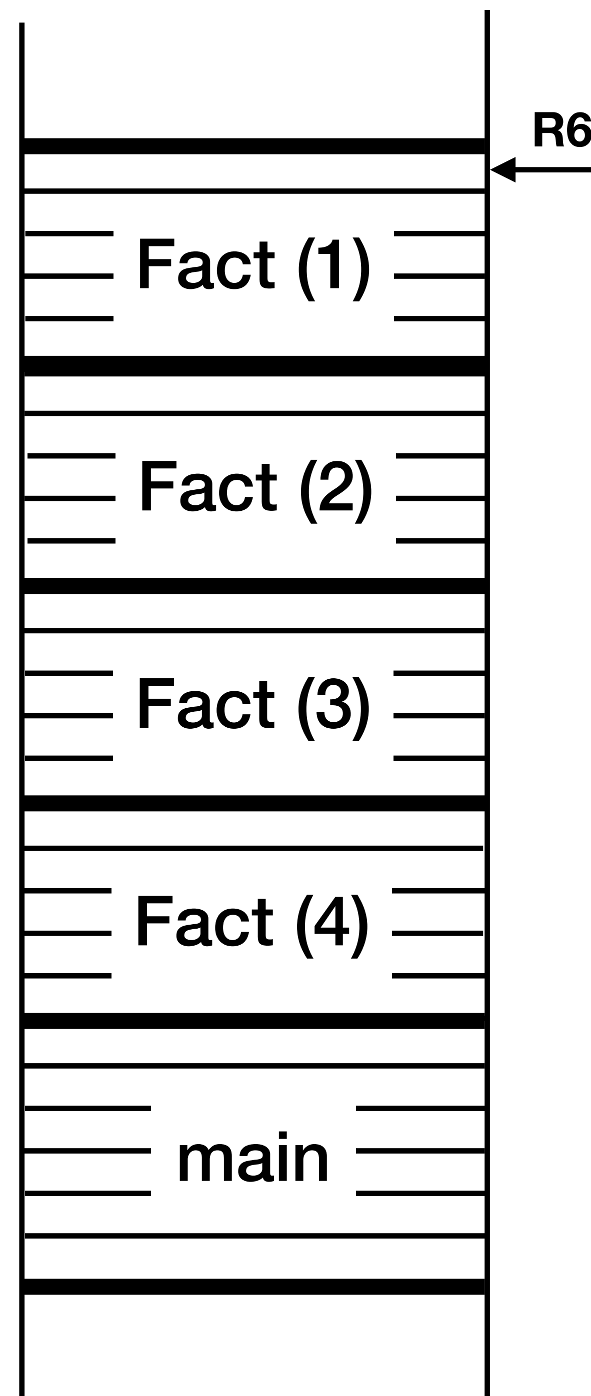


Factorial(3) returns 3 x 2

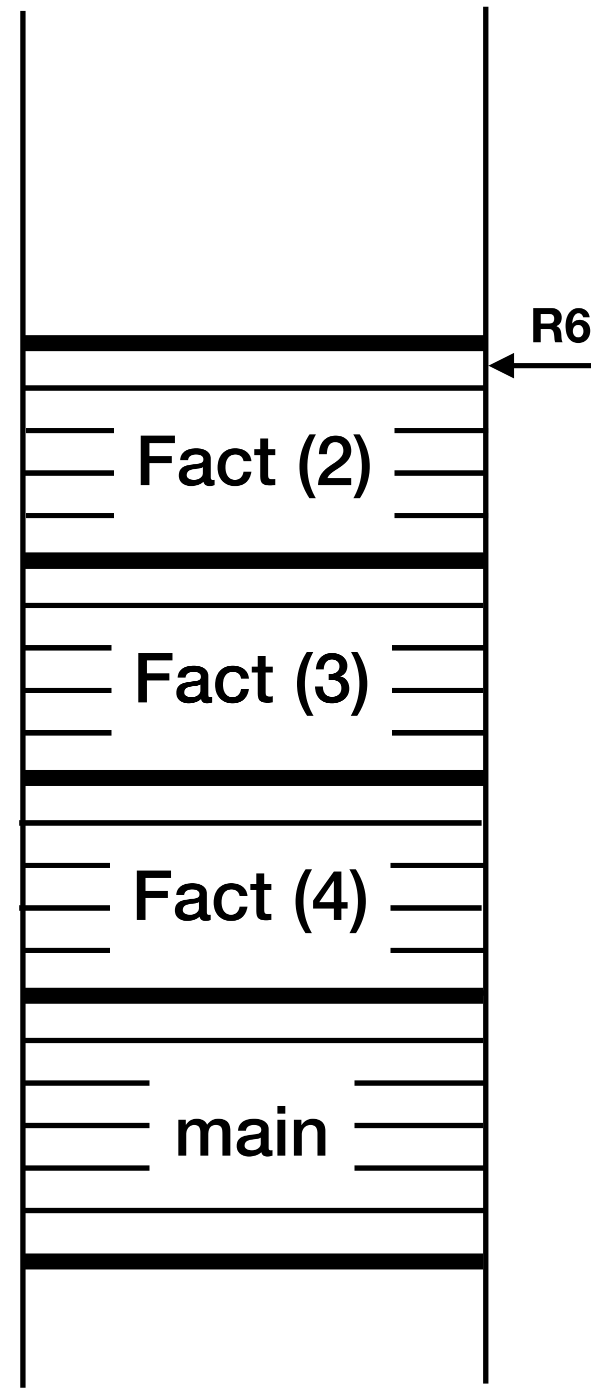


RTS During Execution of Factorial

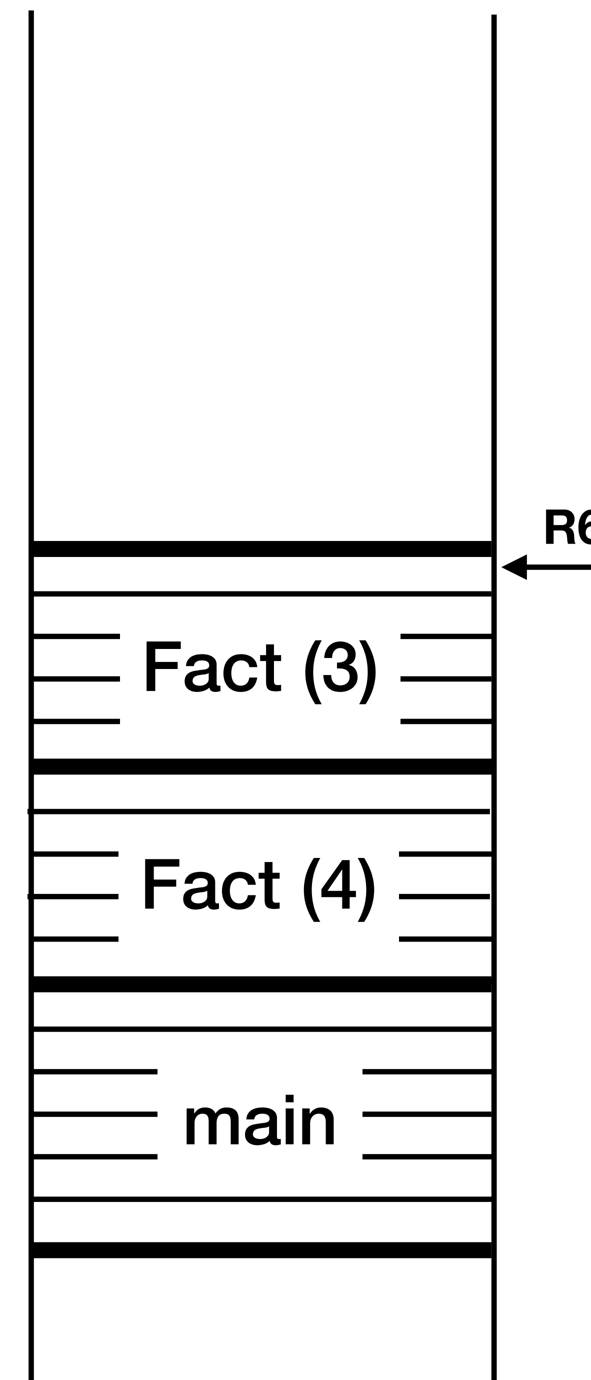
Factorial(1) returns 1



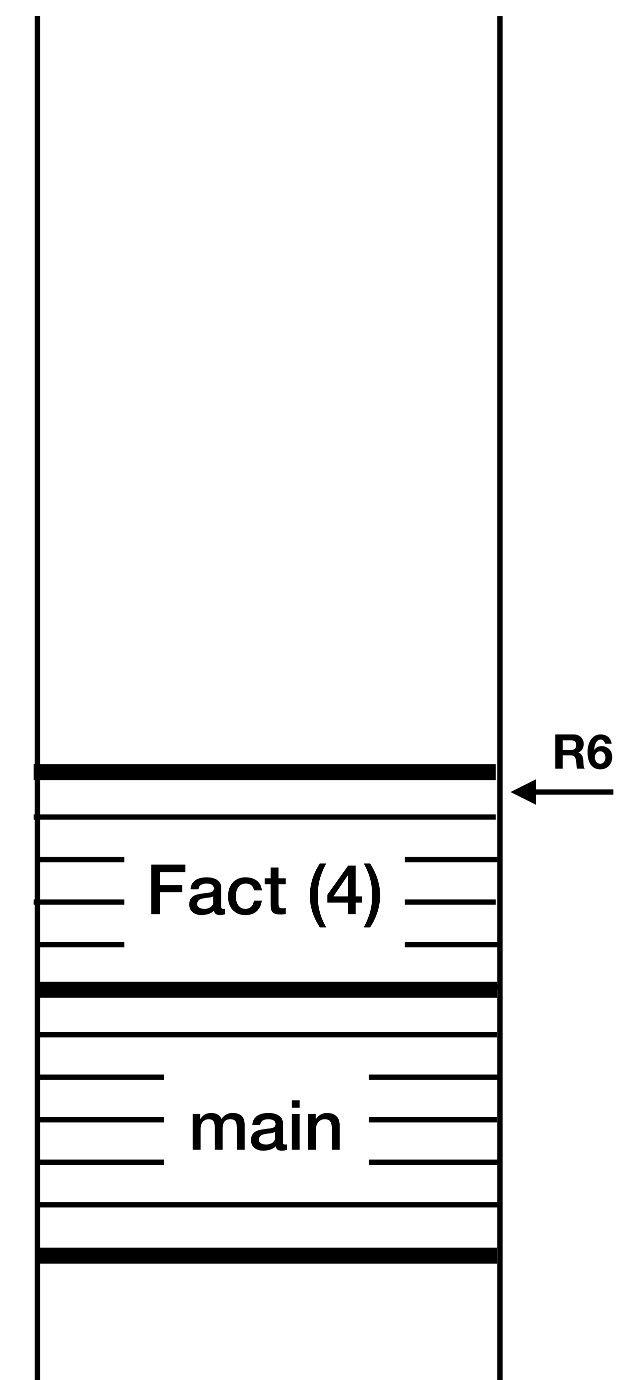
Factorial(2) returns 2 x 1



Factorial(3) returns 3 x 2



Factorial(4) returns 4 x 6



Example: Fibonacci series

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Fibonacci series

Mathematical definition:

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

$$f(n) = f(n-1) + f(n-2)$$

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1$$

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1$$

$$f(0) = 1$$

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1$$

$$f(0) = 1$$

Fibonacci (3) = Fibonacci(2) + Fibonacci(1)

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1$$

$$f(0) = 1$$

Fibonacci (3) = Fibonacci(2) + Fibonacci(1)

= (Fibonacci(1) + Fibonacci(0)) + Fibonacci(1)

Example: Fibonacci series

Mathematical definition:

Fibonacci Series: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1$$

$$f(0) = 1$$

Fibonacci (3) = Fibonacci(2) + Fibonacci(1)

= (Fibonacci(1) + Fibonacci(0)) + Fibonacci(1)

= 1 + 1 + 1 = 3

Example: Fibonacci series

Example: Fibonacci series

```
int Fibonacci (int n) {
    int sum;

    if (n == 0 || n == 1)
        return 1;
    else {
        sum = (Fibonacci(n-1) + Fibonacci(n-2));
        return sum;
        }
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Binary search

```
int binary(int arr[], int n, int key){
  int start = 0;       // Left pointer
  int end = n - 1; // Right pointer

  while (end >= start){
    int mid = (start + end) / 2; // Pick middle element

    // Logic to focus search on left or right of mid
    if (key == arr[mid])
      return mid;
    else if (key < arr[mid])
      end = mid - 1;
    else
      start = mid + 1;
  }
  return -1; // Loop exited, element not present.
}
```

Can we implement binary search in *recursive* way?

Example: Binary search

```
int binary(int arr[], int n, int key){
  int start = 0;       // Left pointer
  int end = n - 1; // Right pointer

  while (end >= start){
    int mid = (start + end) / 2; // Pick middle element

    // Logic to focus search on left or right of mid
    if (key == arr[mid])
      return mid;
    else if (key < arr[mid])
      end = mid - 1;
    else
      start = mid + 1;
  }
  return -1; // Loop exited, element not present.
}
```

Can we implement binary search in *recursive* way?

- **Option 1**: Find a mechanism to keep track of the start and end indices across recursive calls; local variables won't do (why?).

Example: Binary search

```
int binary(int arr[], int n, int key){
  int start = 0;      // Left pointer
  int end = n - 1; // Right pointer

  while (end >= start){
    int mid = (start + end) / 2; // Pick middle element

    // Logic to focus search on left or right of mid
    if (key == arr[mid])
      return mid;
    else if (key < arr[mid])
      end = mid - 1;
    else
      start = mid + 1;
  }
  return -1; // Loop exited, element not present.
}
```

Can we implement binary search in *recursive* way?

- **Option 1**: Find a mechanism to keep track of the start and end indices across recursive calls; local variables won't do (why?).
- **Option 2**: Pass in subarrays & their lengths to `binary` itself.

Example: Binary search

Example: Binary search

```
int binary_opt1(int item, int list[], int start, int end){
    int middle = (end + start)/2;
    if (end < start)
      return -1;                      // Did not find key
    else if (_____________________) // Found item!
        return middle;
    else if (____________________)  // Search left half
        return binary_opt1(item, list, start, middle-1);
    else                            // Search right half
        return binary_opt1(__________________________);
}
```

Example: Binary search

Option A

```
int binary_opt1(int item, int list[], int start, int end){
    int middle = (end + start)/2;
    if (end < start)
      return -1;                          // Did not find key
    else if (____________________) // Found item!
        return middle;
    else if (___________________)  // Search left half
        return binary_opt1(item, list, start, middle-1);
    else                                  // Search right half
        return binary_opt1(_________________________);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Binary search

```
int binary_opt1(int item, int list[], int start, int end){
    int middle = (end + start)/2;
    if (end < start)
      return -1;                          // Did not find key
    else if (_____________________) // Found item!
        return middle;
    else if (____________________)  // Search left half
        return binary_opt1(item, list, start, middle-1);
    else                                  // Search right half
        return binary_opt1(__________________________);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Binary search

```
int binary_opt1(int item, int list[], int start, int end){
    int middle = (end + start)/2;
    if (end < start)
      return -1;                          // Did not find key
    else if (list[middle] == item) // Found item!
        return middle;
    else if (____________________)  // Search left half
        return binary_opt1(item, list, start, middle-1);
    else                              // Search right half
        return binary_opt1(__________________________);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Binary search

```
int binary_opt1(int item, int list[], int start, int end){
    int middle = (end + start)/2;
    if (end < start)
      return -1;                        // Did not find key
    else if (list[middle] == item) // Found item!
        return middle;
    else if (item < list[middle])  // Search left half
        return binary_opt1(item, list, start, middle-1);
    else                                // Search right half
        return binary_opt1(__________________________);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Binary search

```
int binary_opt1(int item, int list[], int start, int end){
    int middle = (end + start)/2;
    if (end < start)
      return -1;                       // Did not find key
    else if (list[middle] == item) // Found item!
        return middle;
    else if (item < list[middle])  // Search left half
        return binary_opt1(item, list, start, middle-1);
    else                           // Search right half
        return binary_opt1(item, list, middle+1, end);
}
```

Example: Binary search

What if we *insist* the user shouldn't have to deal with the start, end pointers etc.?

We can always wrap `binary_opt1` in another function.

Below is *another* option.

```
int main() {
  int arr[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23};
  int size = sizeof(arr) / sizeof(arr[0]);
  int index, key = 11;

  int *result = binary_opt2(arr, size, key);
  if (result == NULL)
    printf("Element not found");
  else{
    // Find index based on pointer
    index = (result-arr);
    printf("Element found at index: %d", index);
  }
}
```

Example: Binary search

What if we *insist* the user shouldn't have to deal with the start, end pointers etc.?

We can always wrap `binary_opt1` in another function.

Below is *another* option.

```
int main() {
  int arr[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23};
  int size = sizeof(arr) / sizeof(arr[0]);
  int index, key = 11;

  int *result = binary_opt2(arr, size, key);
  if (result == NULL)
    printf("Element not found");
  else{
    // Find index based on pointer
    index = (result-arr);
    printf("Element found at index: %d", index);
  }
}
```

Index must be calculated manually.

Example: Binary search

What if we *insist* the user shouldn't have to deal with the start, end pointers etc.?

We can always wrap `binary_opt1` in another function.

Below is *another* option.

Function returns a pointer

```
int main() {
  int arr[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23};
  int size = sizeof(arr) / sizeof(arr[0]);
  int index, key = 11;

  int *result = binary_opt2(arr, size, key);
  if (result == NULL)
    printf("Element not found");
  else{
    // Find index based on pointer
    index = (result-arr);
    printf("Element found at index: %d", index);
  }
}
```

Index must be calculated manually.

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Binary search

What if we *insist* the user shouldn't have to deal with the start, end pointers etc.?

```
int * binary(int arr[], int n, int key){
  int mid;
  if (n > 1){ // If not singleton do search
    mid=n/2;
    if (key == arr[mid]) // Check middle
      return &arr[mid];
    else if (key > arr[mid]) // Search right 1/2
      return binary(&arr[mid+1], n-mid, key);
    else
      return binary(arr, mid, key); // Left 1/2
  }
  else if (arr[0]==key) // If singleton check equality
    return arr;
  else // Didn't find element
    return NULL;
}
```

We can always wrap `binary_opt1` in another function.

Below is *another* option.

Function returns a pointer

```
int main() {
  int arr[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23};
  int size = sizeof(arr) / sizeof(arr[0]);
  int index, key = 11;

  int *result = binary_opt2(arr, size, key);
  if (result == NULL)
    printf("Element not found");
  else{
    // Find index based on pointer
    index = (result-arr);
    printf("Element found at index: %d", index);
  }
}
```

Index must be calculated manually.

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …

- Recall the steps ….

1. Choose first element of given array as pivot.

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …

- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot
4. Decrement right pointer while the element it points to is greater than pivot.

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot
4. Decrement right pointer while the element it points to is greater than pivot.
    1. If pointers cross/overlap, **split array** & **recurse** on each subarray.

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot
4. Decrement right pointer while the element it points to is greater than pivot.
    1. If pointers cross/overlap, **split array** & **recurse** on each subarray.
5. If neither pointers can move swap elements.

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot
4. Decrement right pointer while the element it points to is greater than pivot.
    1. If pointers cross/overlap, **split array** & **recurse** on each subarray.
5. If neither pointers can move swap elements.
6. Repeat 3-5 while left pointer < right pointer.

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot
4. Decrement right pointer while the element it points to is greater than pivot.
    1. If pointers cross/overlap, **split array** & **recurse** on each subarray.
5. If neither pointers can move swap elements.
6. Repeat 3-5 while left pointer < right pointer.

We will write one function to split array

Example: Quicksort

- We already saw Quicksort last time and remarked it was recursive …
- Recall the steps ….

1. Choose first element of given array as pivot.
2. Maintain pointers from the left and right.
3. Increment left pointer while the element it points to is less than pivot
4. Decrement right pointer while the element it points to is greater than pivot.
    1. If pointers cross/overlap, **split array** & **recurse** on each subarray.
5. If neither pointers can move swap elements.
6. Repeat 3-5 while left pointer < right pointer.

And another which will perform recursion after split

We will write one function to split array

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, ________);

    // And right half
    QuickSort(arr, ____________, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = ___________;
  int i = __________; // Initialize left
  int j = ________;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (_________________);

    do j--; // Decrement right till …
    while (_________________);

    if (______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, ____________, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = __________;
  int i = _________; // Initialize left
  int j = _______;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (_________________);

    do j--; // Decrement right till …
    while (_________________);

    if (______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```c
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```c
int partition(int arr[], int start, int end){

  int pivotVal = __________;
  int i = __________; // Initialize left
  int j = ________;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (____________________);

    do j--; // Decrement right till …
    while (____________________);

    if (_______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = arr[start];
  int i = _________; // Initialize left
  int j = ________;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (____________________);

    do j--; // Decrement right till …
    while (____________________);

    if (_______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = arr[start];
  int i = start - 1; // Initialize left
  int j = _______;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (_________________);

    do j--; // Decrement right till …
    while (_________________);

    if (______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = arr[start];
  int i = start - 1; // Initialize left
  int j = end + 1;   // Initialize right

 while(1){
   do i++; // Increment left till …
   while (_________________);

   do j--; // Decrement right till …
   while (_________________);

   if (______) // If overlap need to split
     return j;

   Swap(&arr[i], &arr[j]);
 }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = arr[start];
  int i = start - 1; // Initialize left
  int j = end + 1;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (arr[i] < pivotVal);

    do j--; // Decrement right till …
    while (_________________);

    if (______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Implementation

```
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```
int partition(int arr[], int start, int end){

  int pivotVal = arr[start];
  int i = start - 1; // Initialize left
  int j = end + 1;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (arr[i] < pivotVal);

    do j--; // Decrement right till …
    while (arr[j] > pivotVal);

    if (______) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

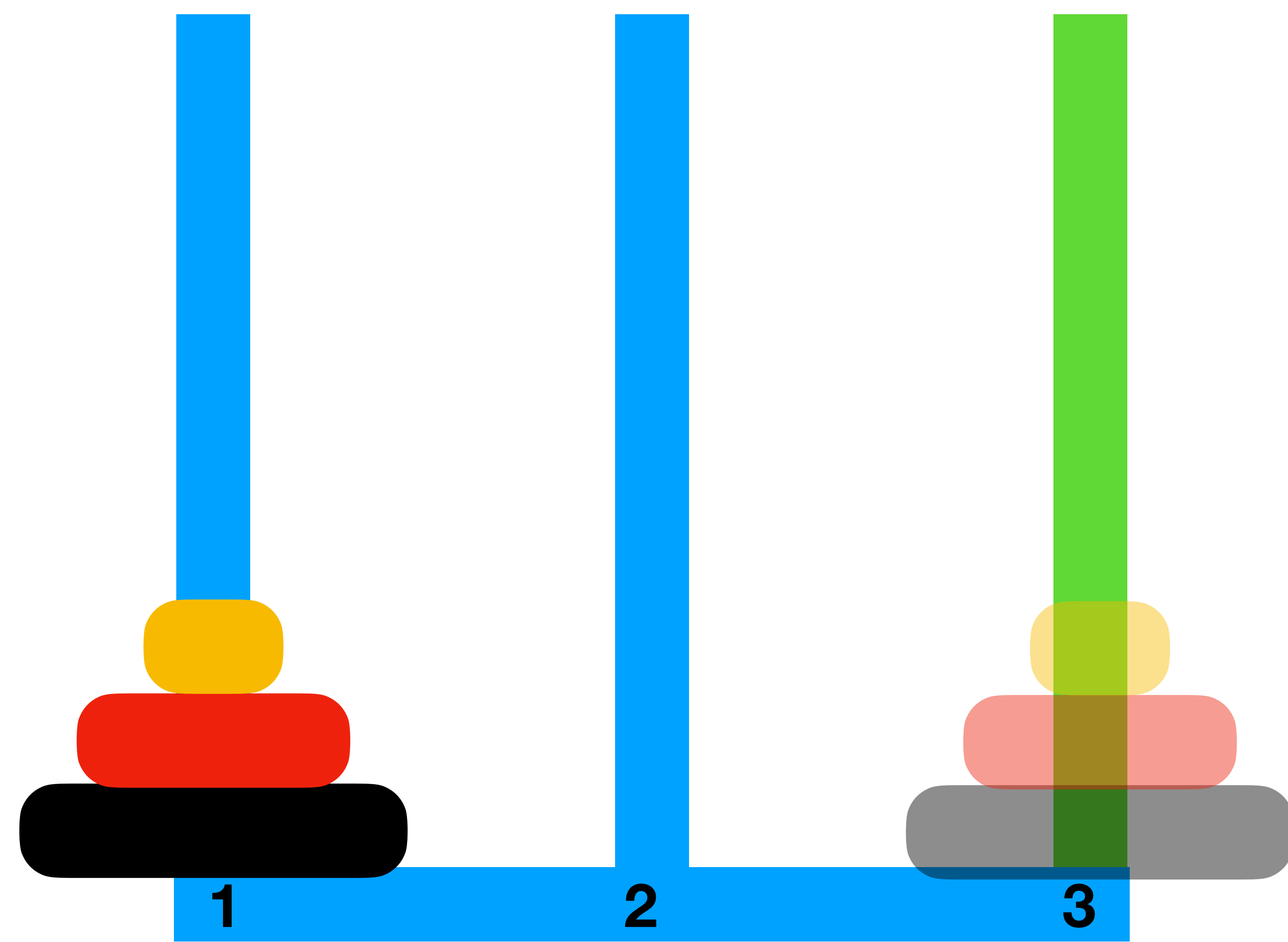
Implementation

```c
void Swap(int* one, int* two){
  int temp = *one;
  *one = *two;
  *two = temp;
}

void QuickSort(int arr[], int start, int end){
  if (start < end){
    int pivotVal = partition(arr, start, end);

    // Now sort left half
    QuickSort(arr, start, pivotVal);

    // And right half
    QuickSort(arr, pivotVal + 1, end);
  }
}
```

```c
int partition(int arr[], int start, int end){

  int pivotVal = arr[start];
  int i = start - 1; // Initialize left
  int j = end + 1;   // Initialize right

  while(1){
    do i++; // Increment left till …
    while (arr[i] < pivotVal);

    do j--; // Decrement right till …
    while (arr[j] > pivotVal);

    if (i >= j) // If overlap need to split
      return j;

    Swap(&arr[i], &arr[j]);
  }
}
```

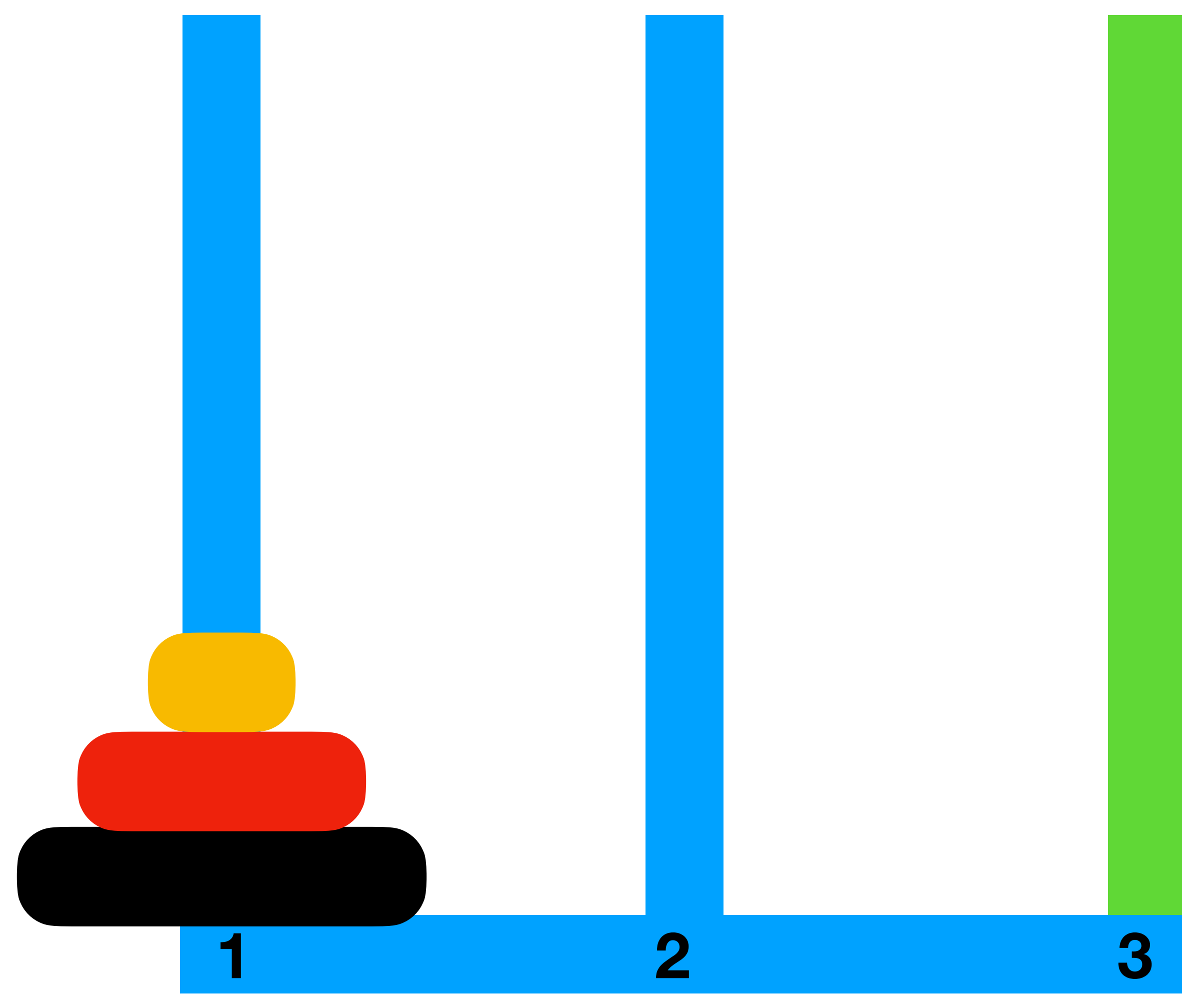
Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Example: Towers of Hanoi

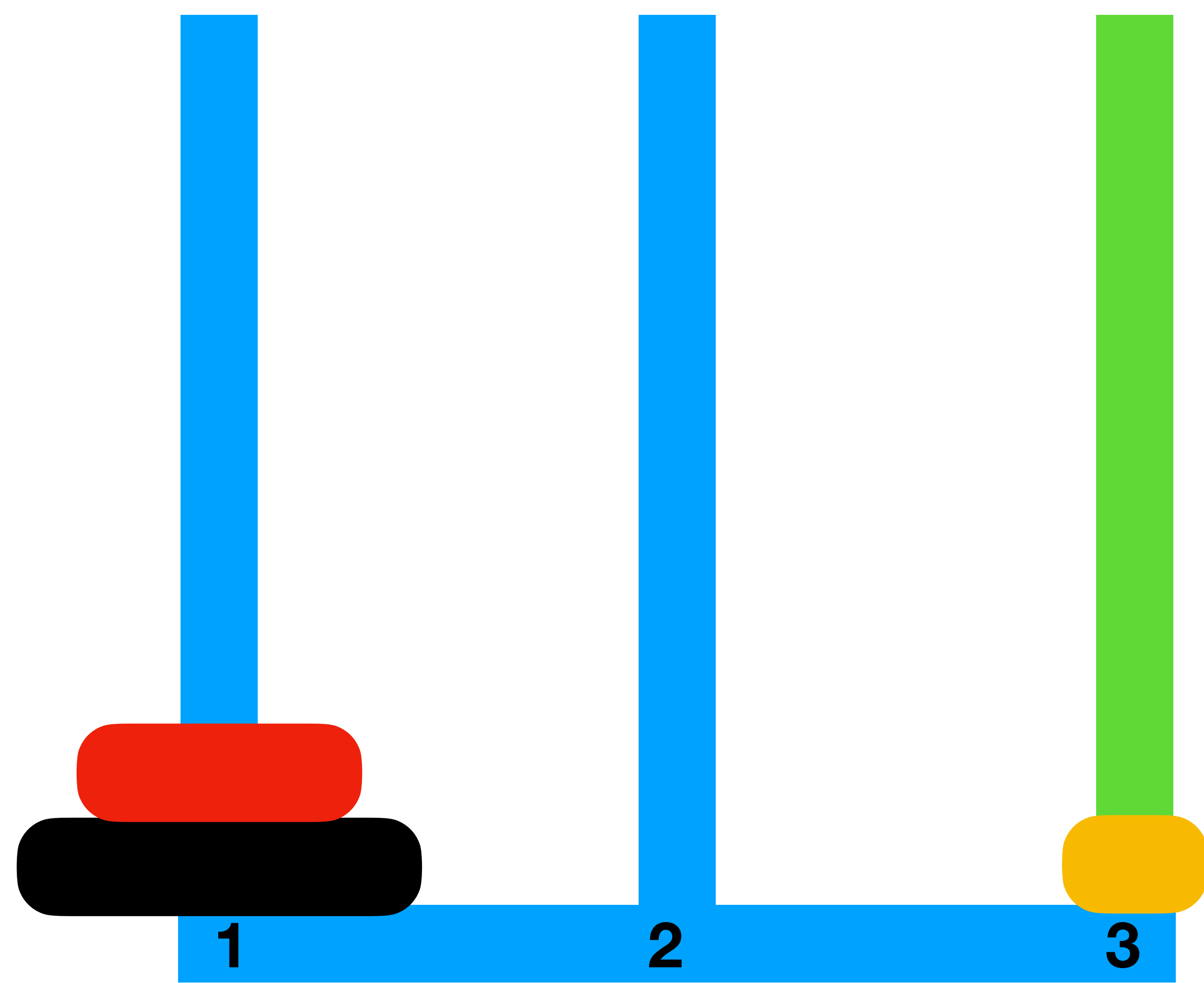


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Towers of Hanoi

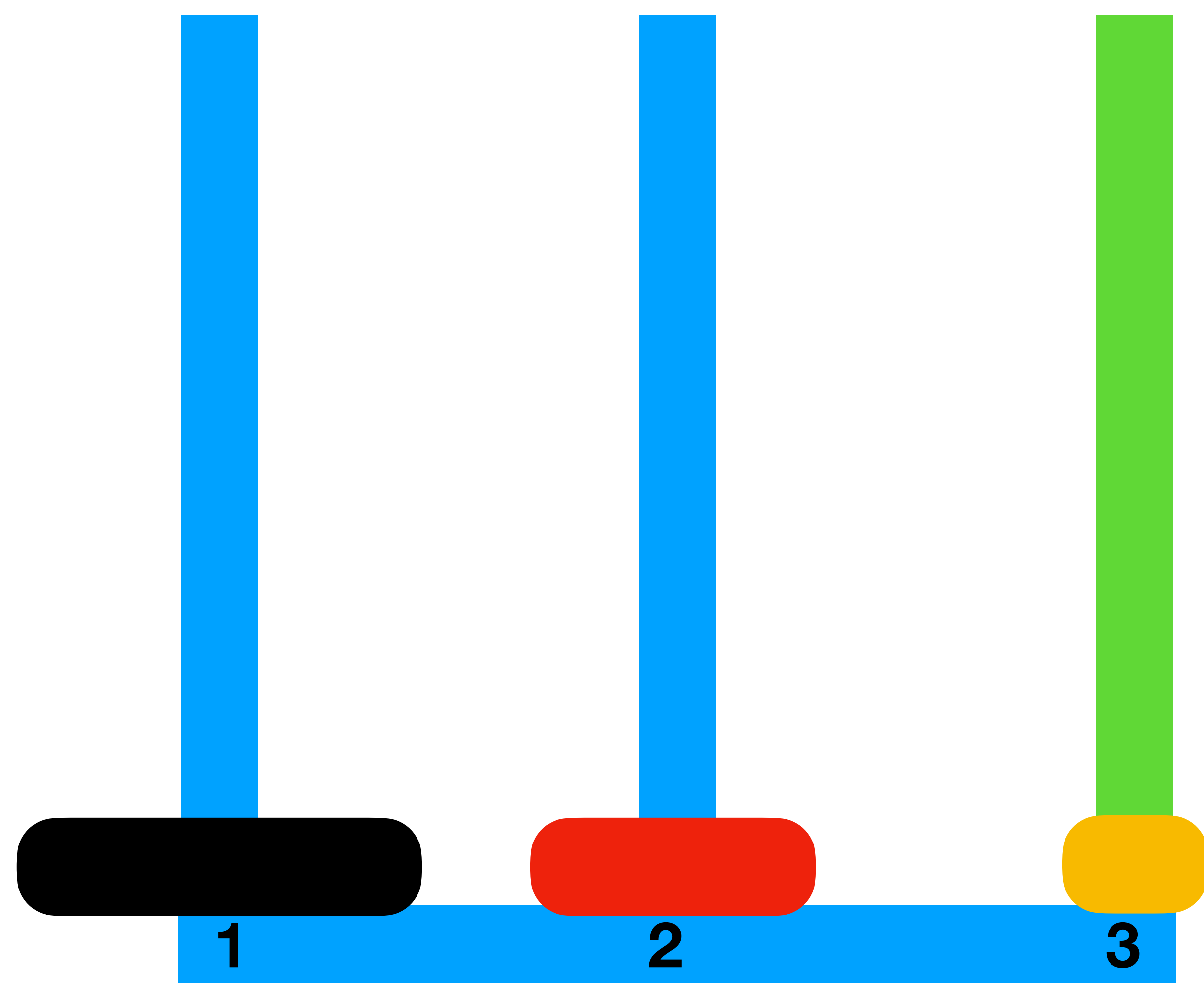


Example: Towers of Hanoi

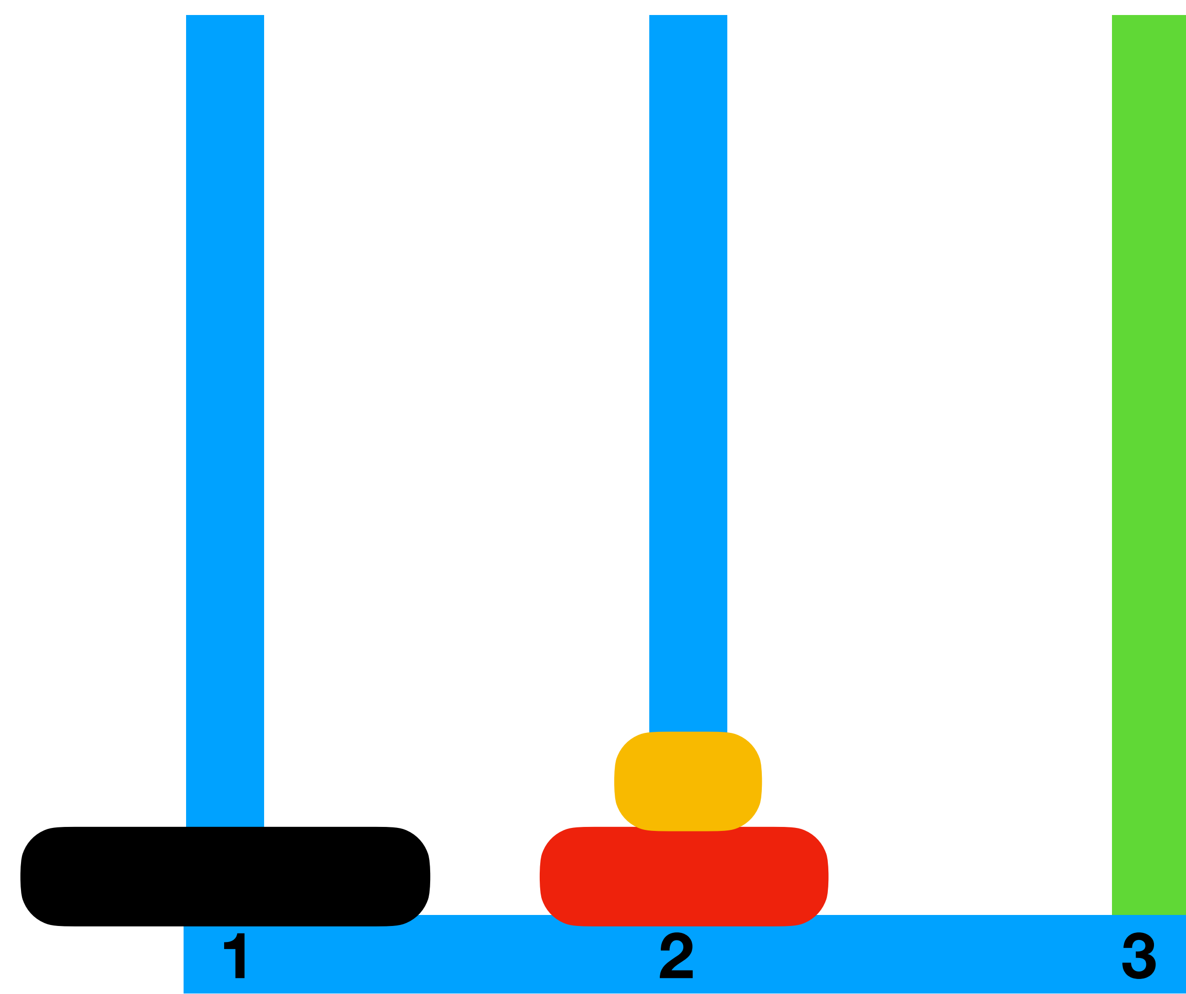


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

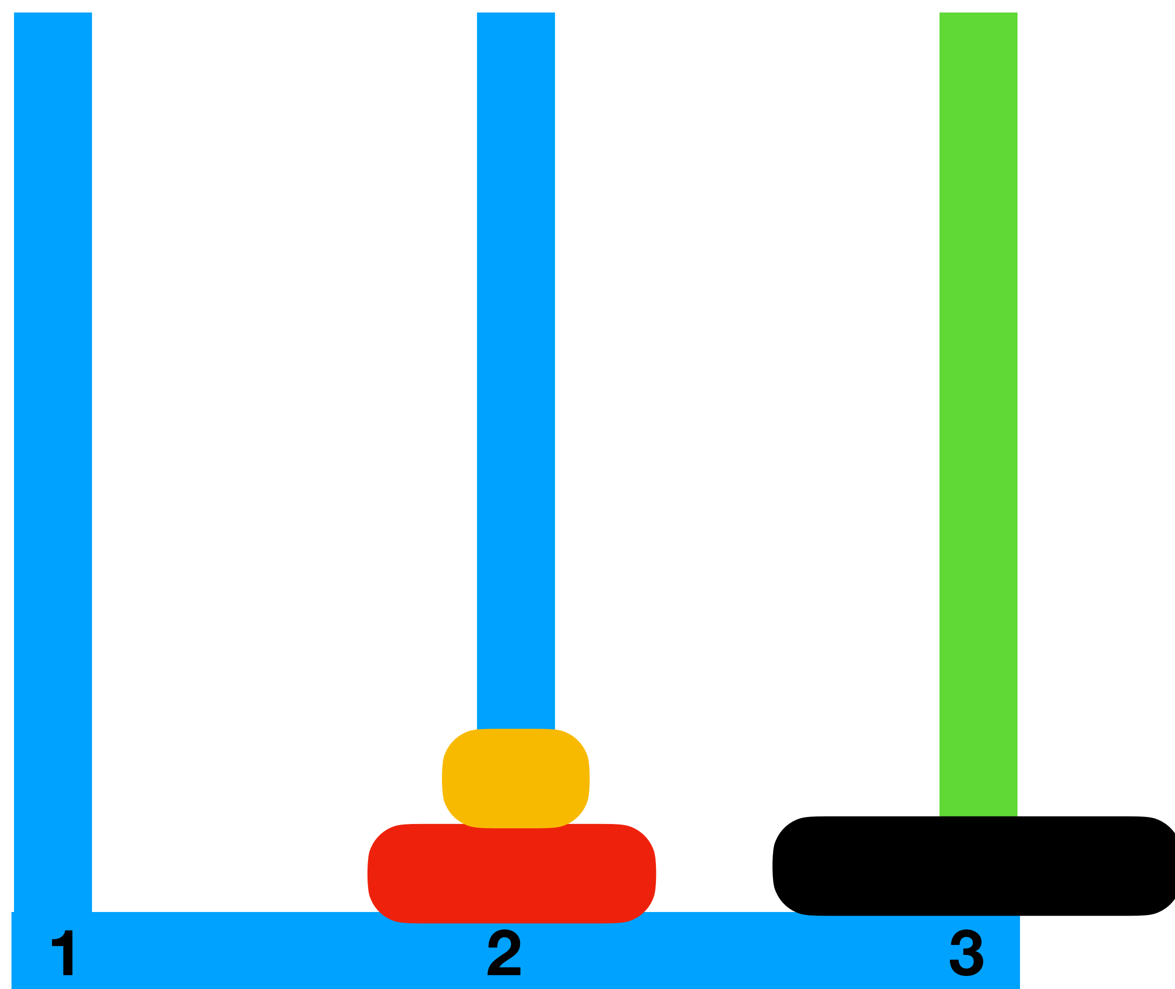
Example: Towers of Hanoi



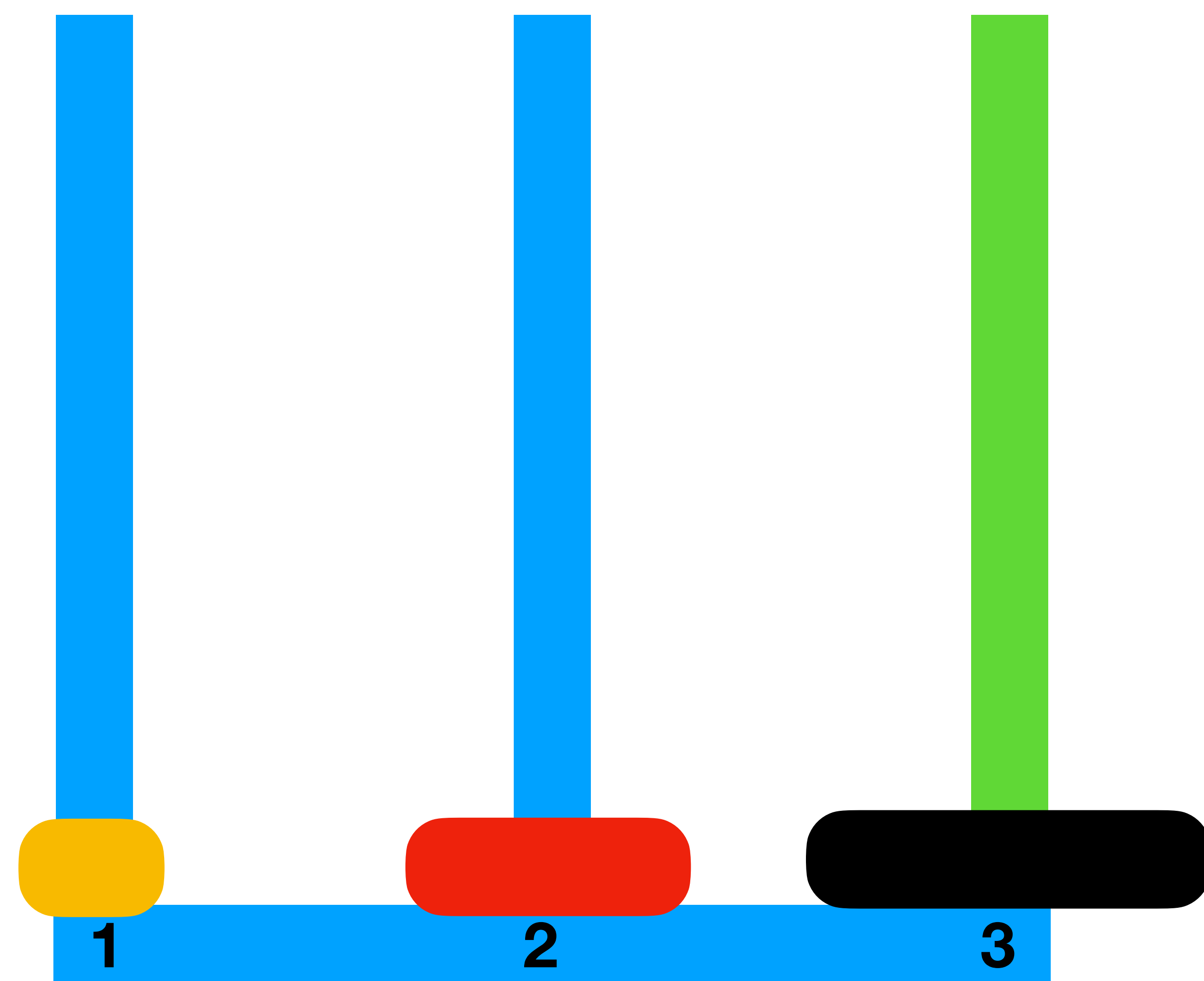
Example: Towers of Hanoi



Example: Towers of Hanoi

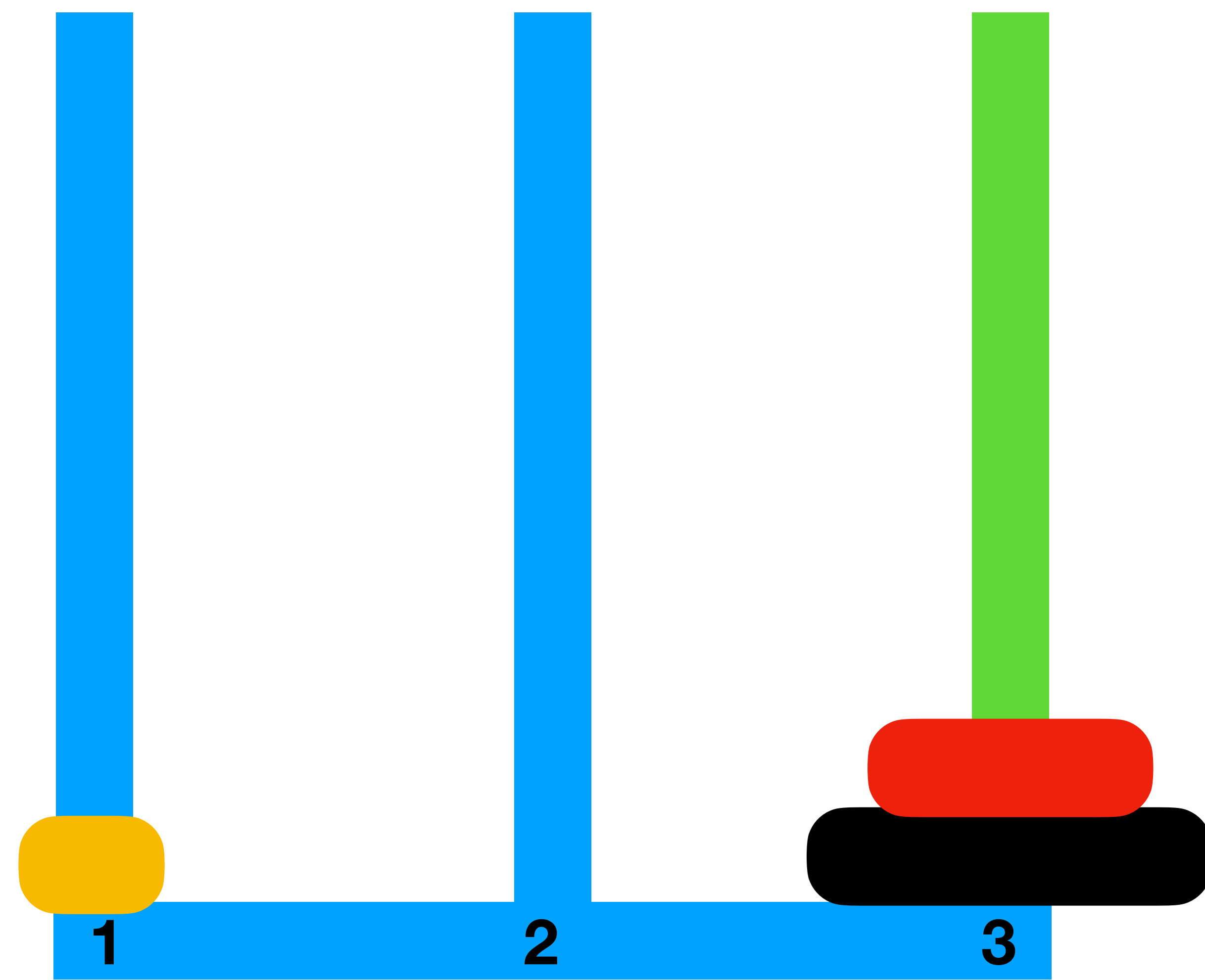


Example: Towers of Hanoi



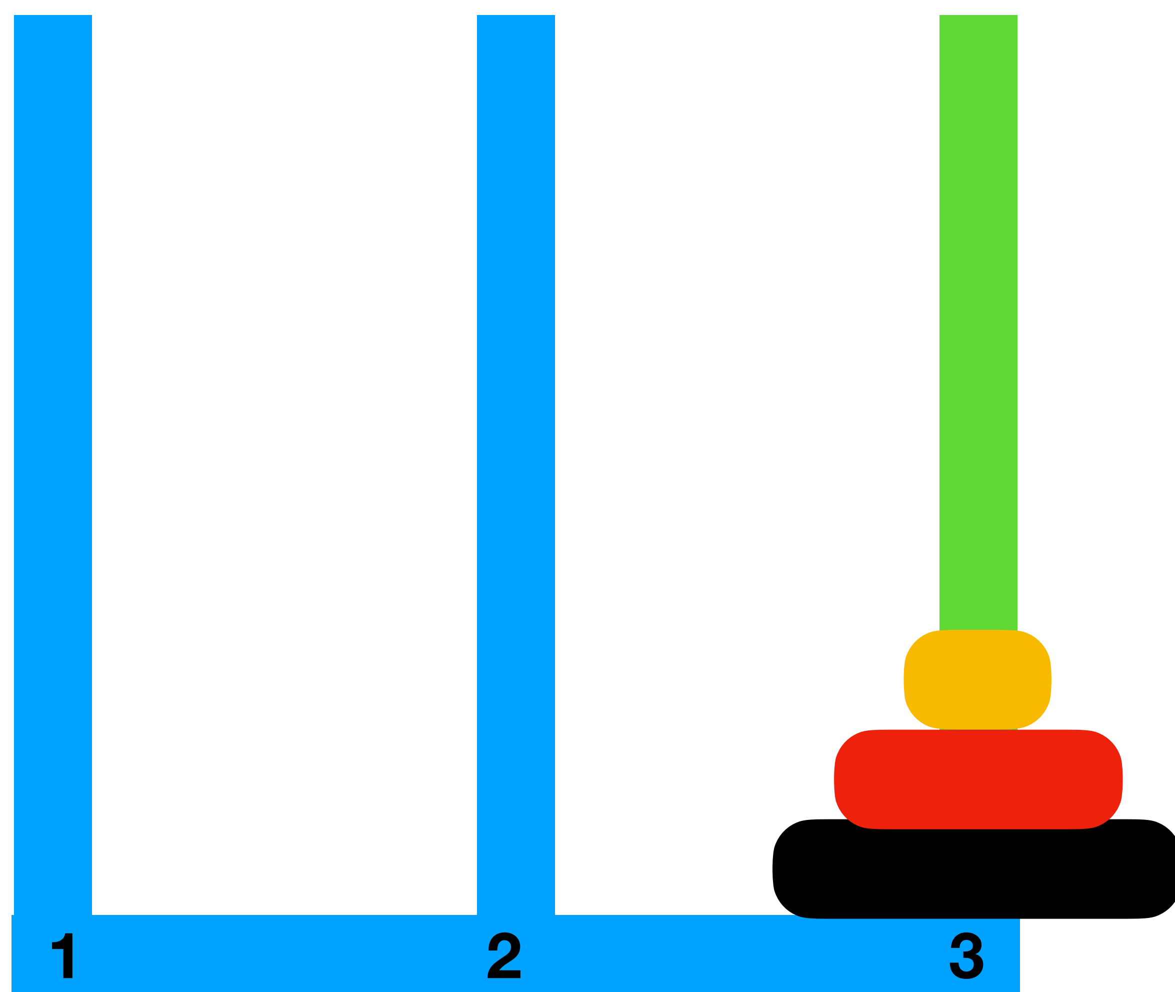
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Towers of Hanoi

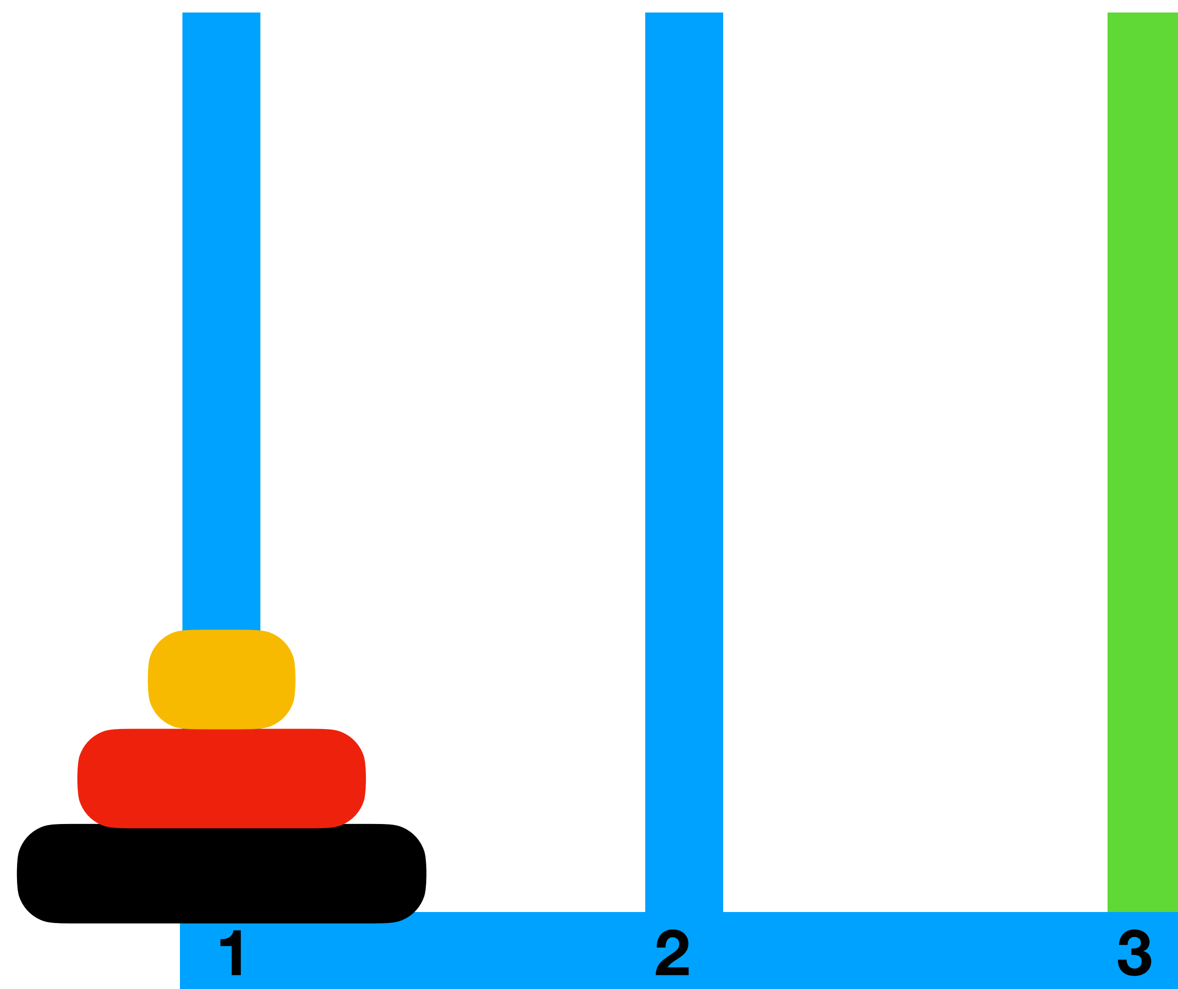


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Towers of Hanoi



Example: Towers of Hanoi

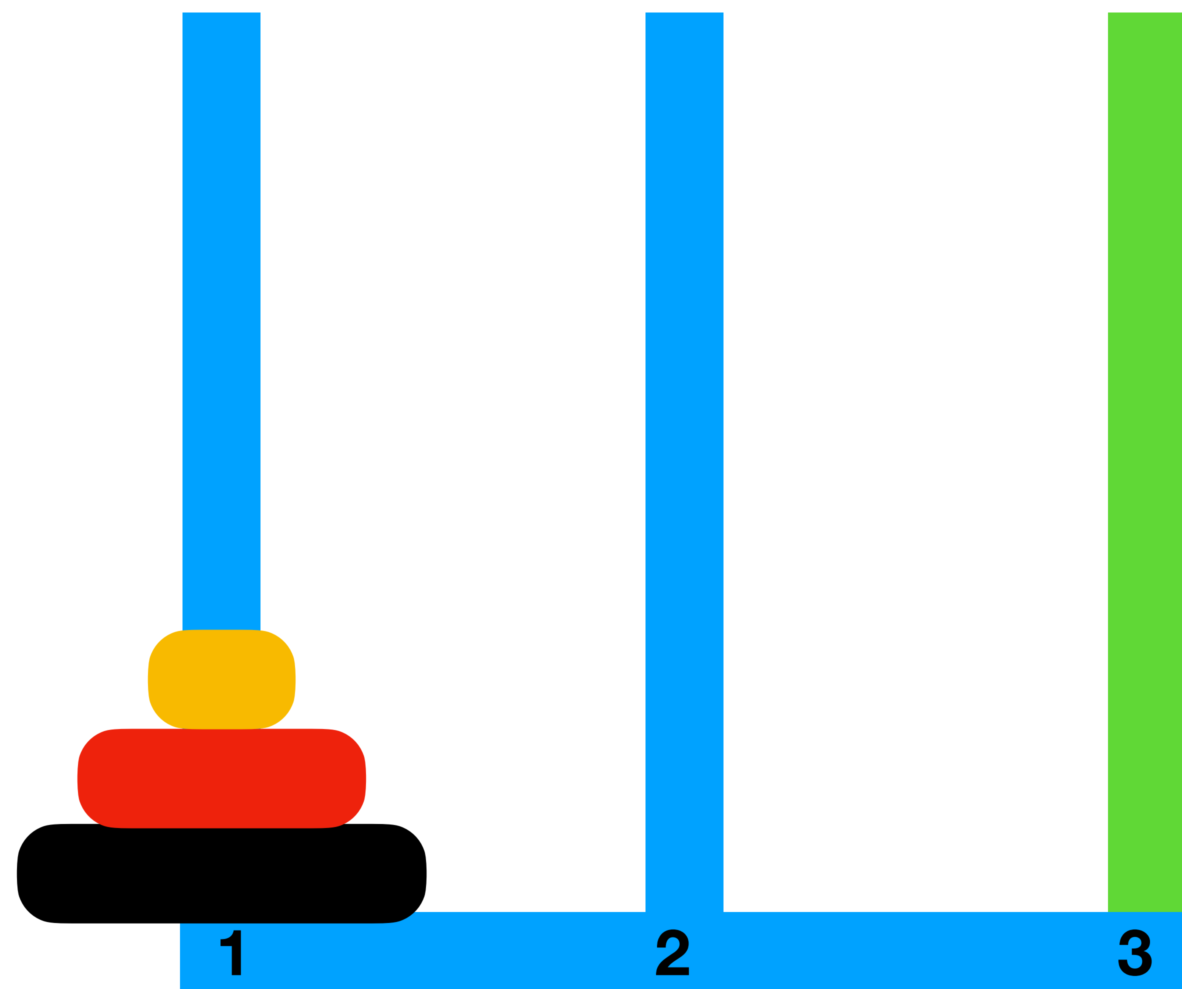


move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

move(disks3, r1, r3, r2)

Example: Towers of Hanoi

move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

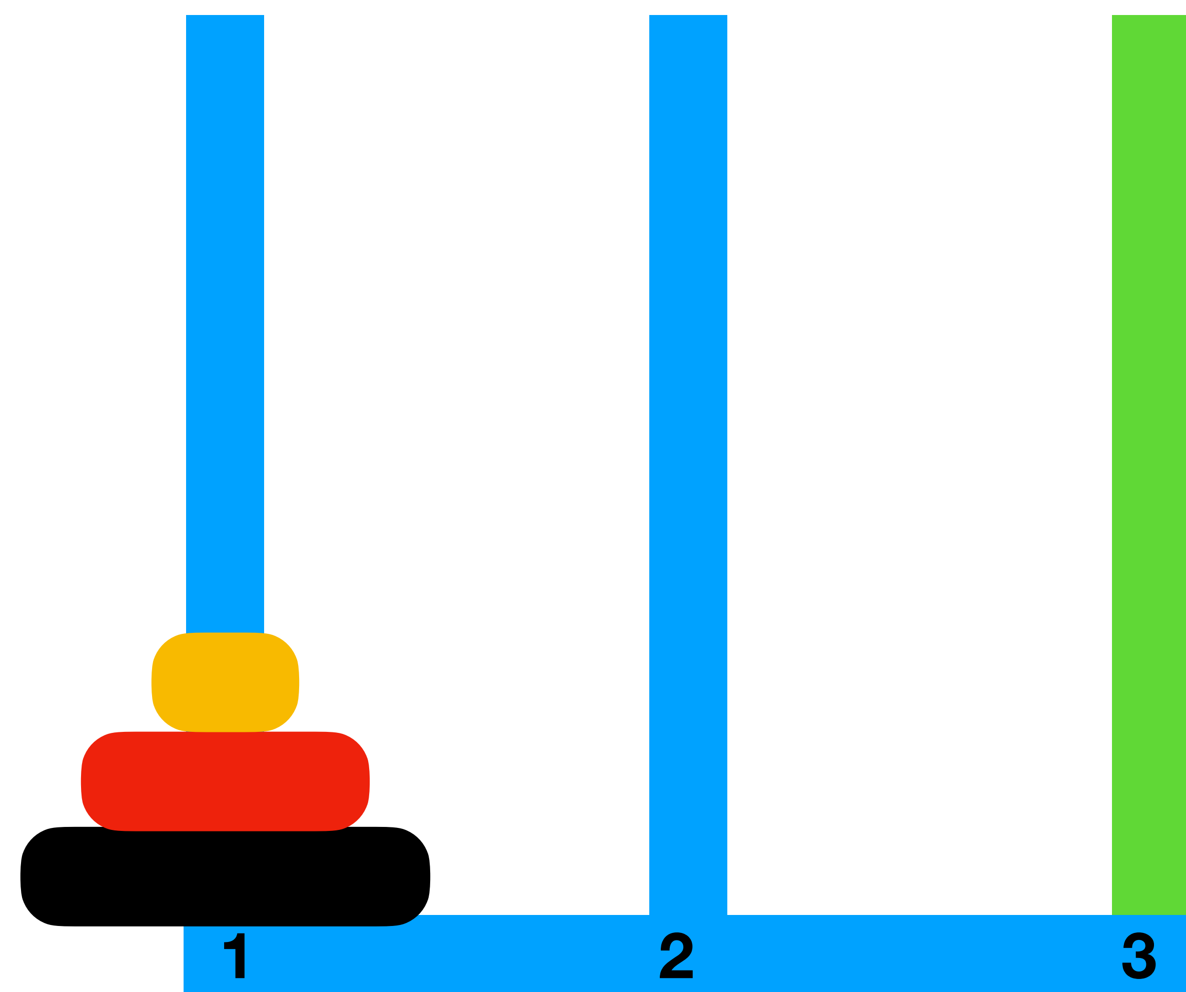


move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

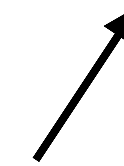
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

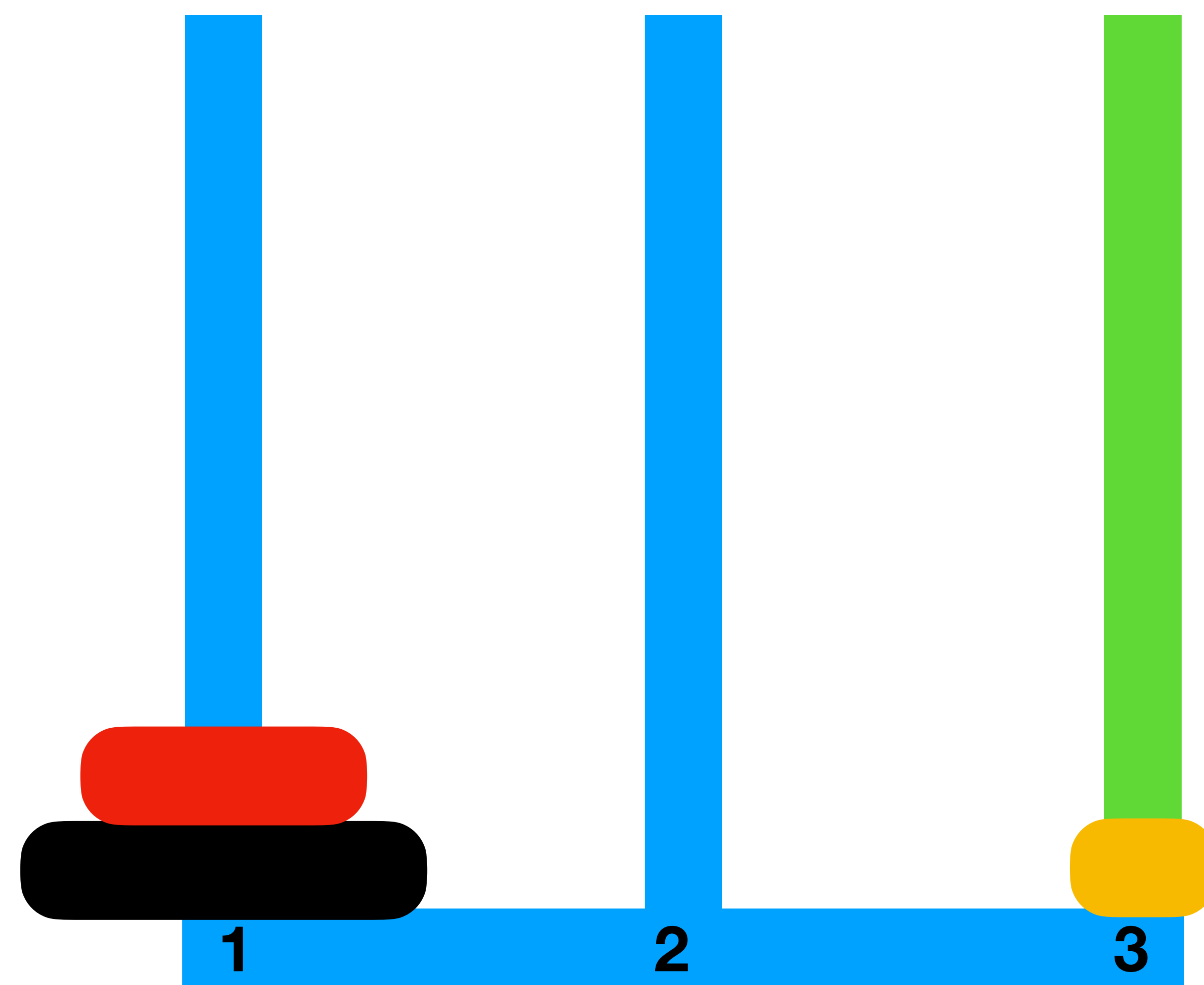
move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)



dmove(disk_1, r1, r3)

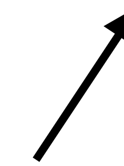
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

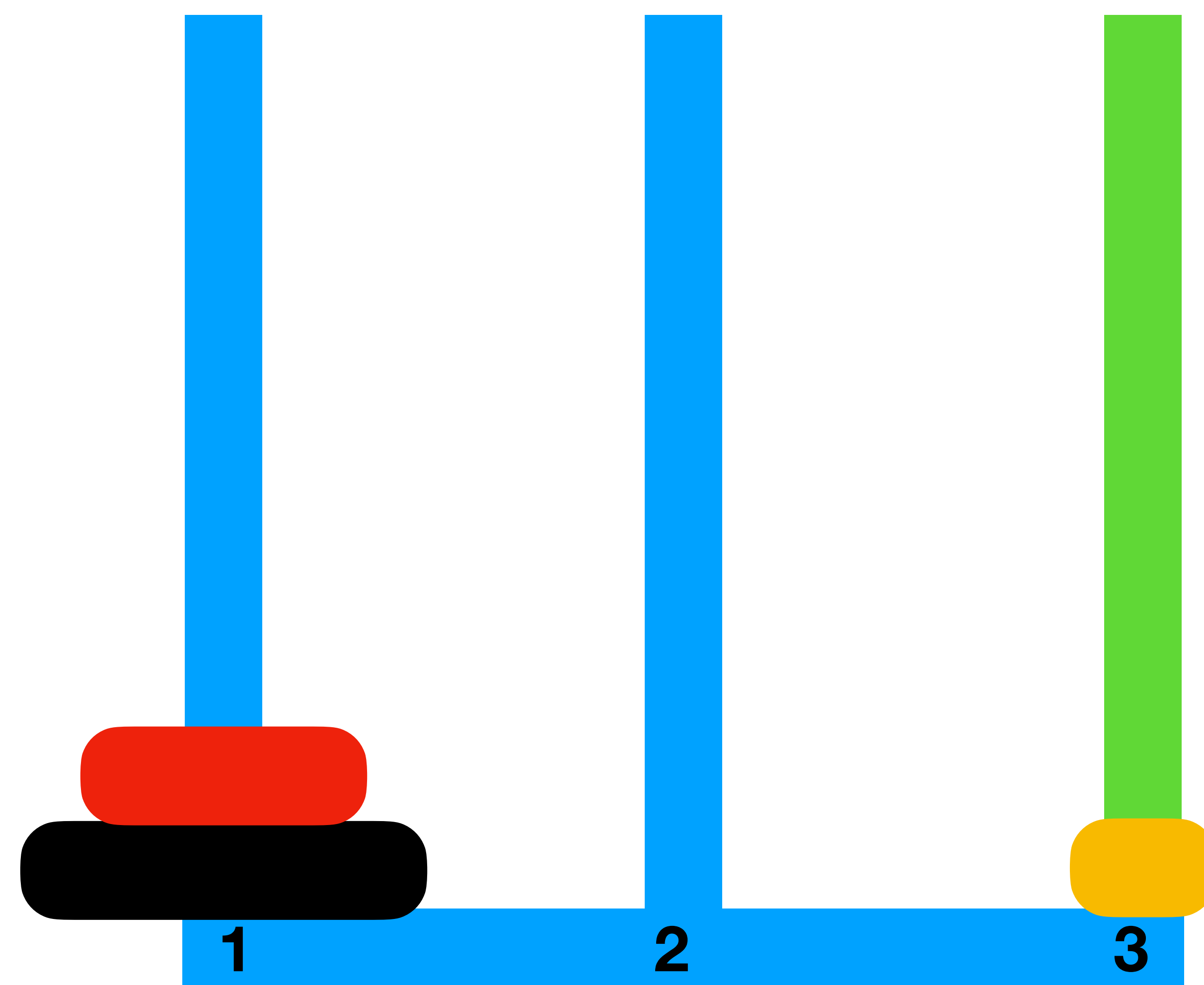
move(disks3, r1, r3, r2)

dmove(disk_1, r1, r3)



move(disks2, r1, r2, r3)

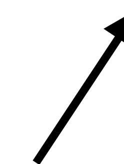
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)

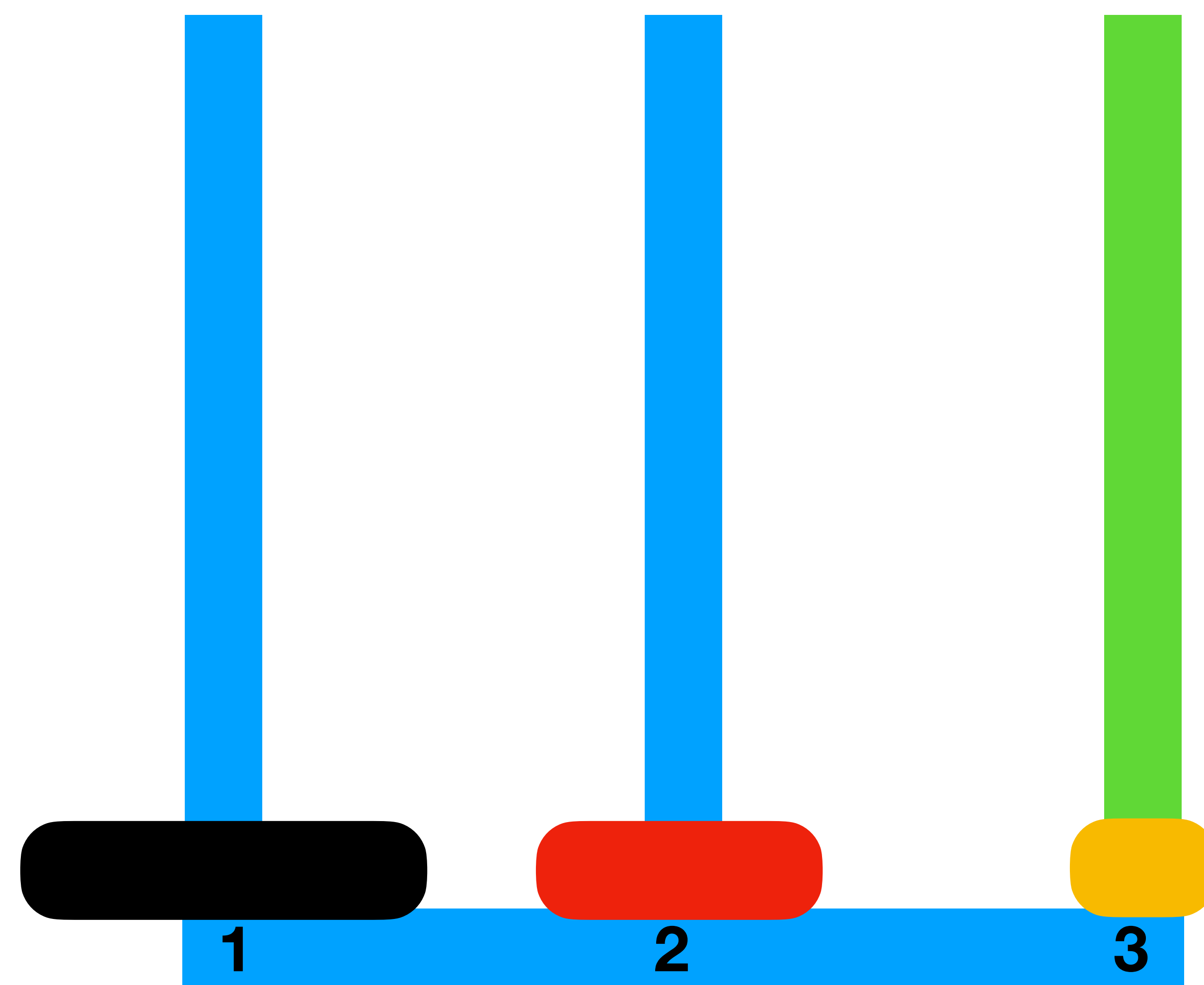


dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

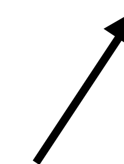
Example: Towers of Hanoi

move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk



move(disks3, r1, r3, r2)

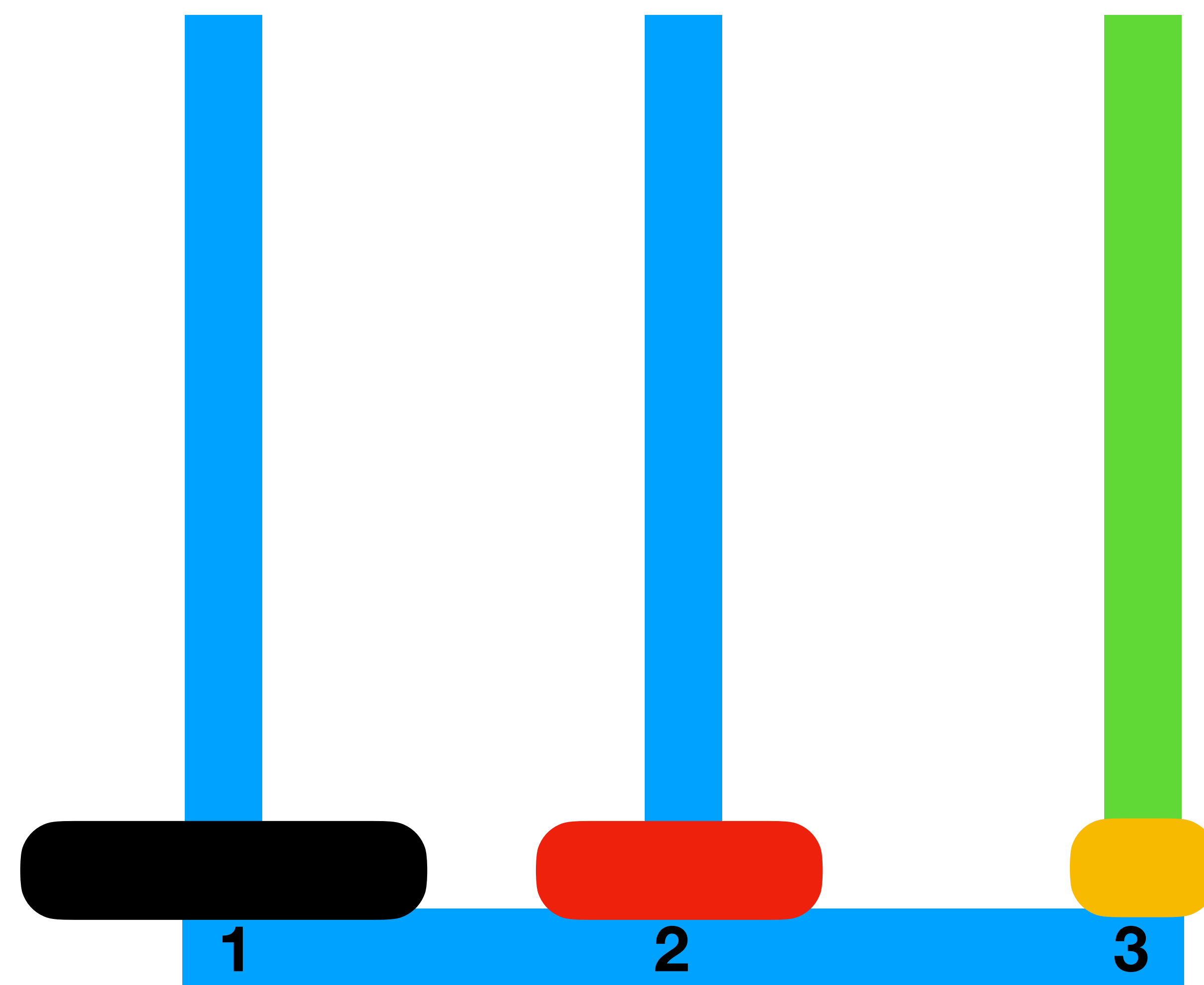
move(disks2, r1, r2, r3)



dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

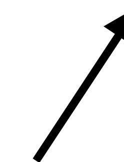
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

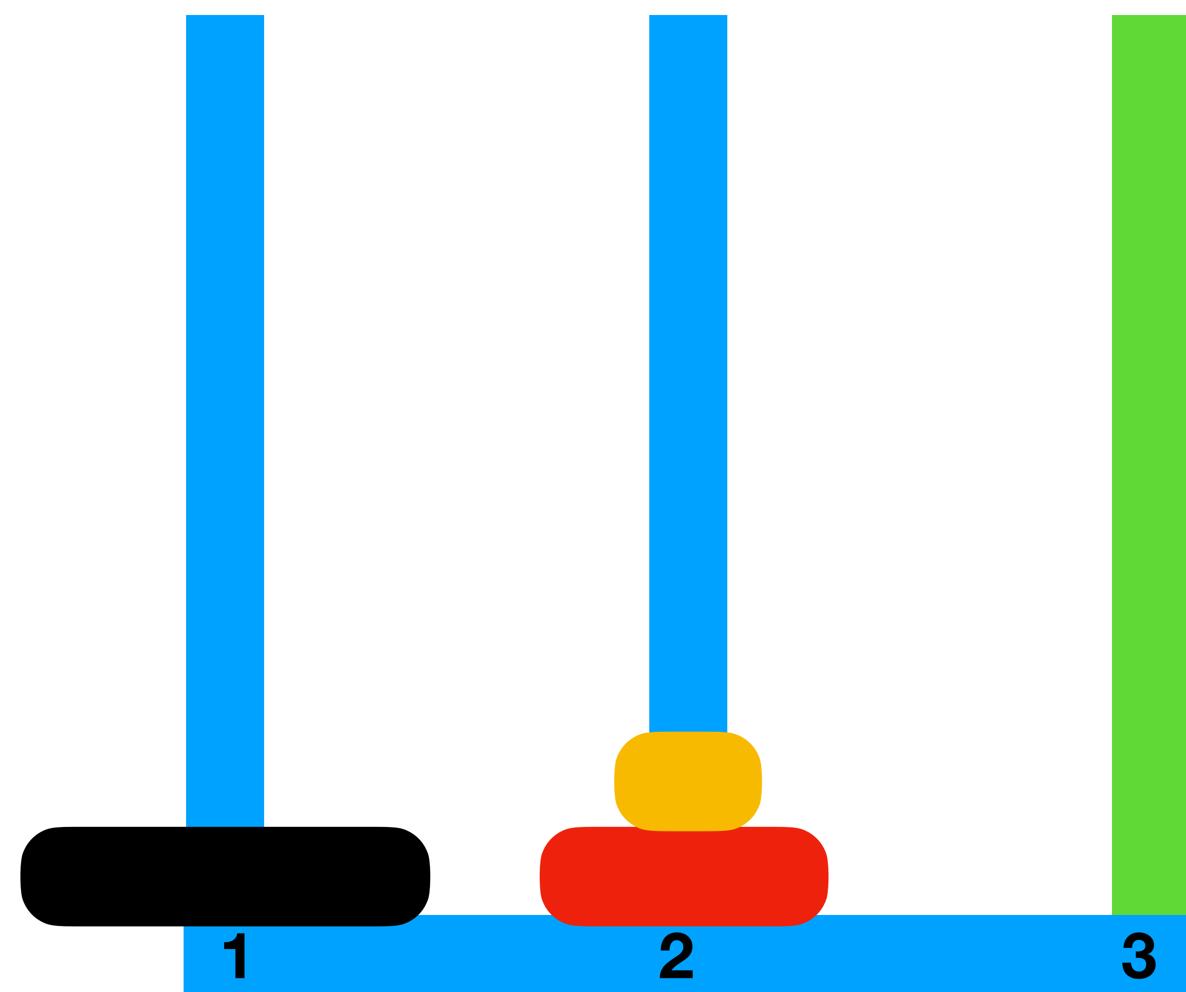
move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)



dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

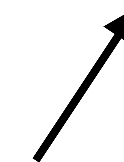
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

move(disks3, r1, r3, r2)

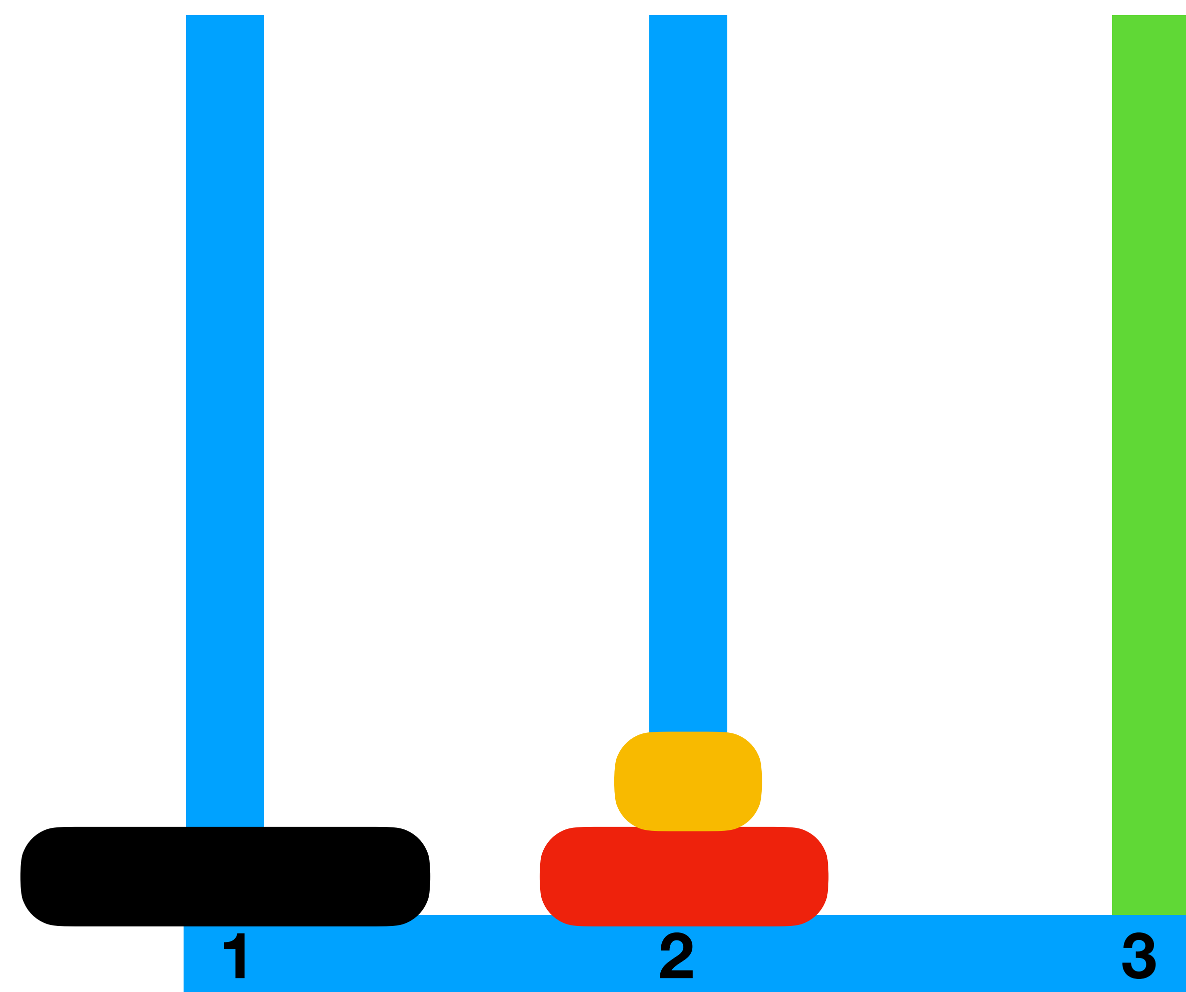
move(disks2, r1, r2, r3)



dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

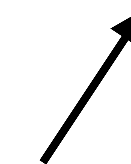
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

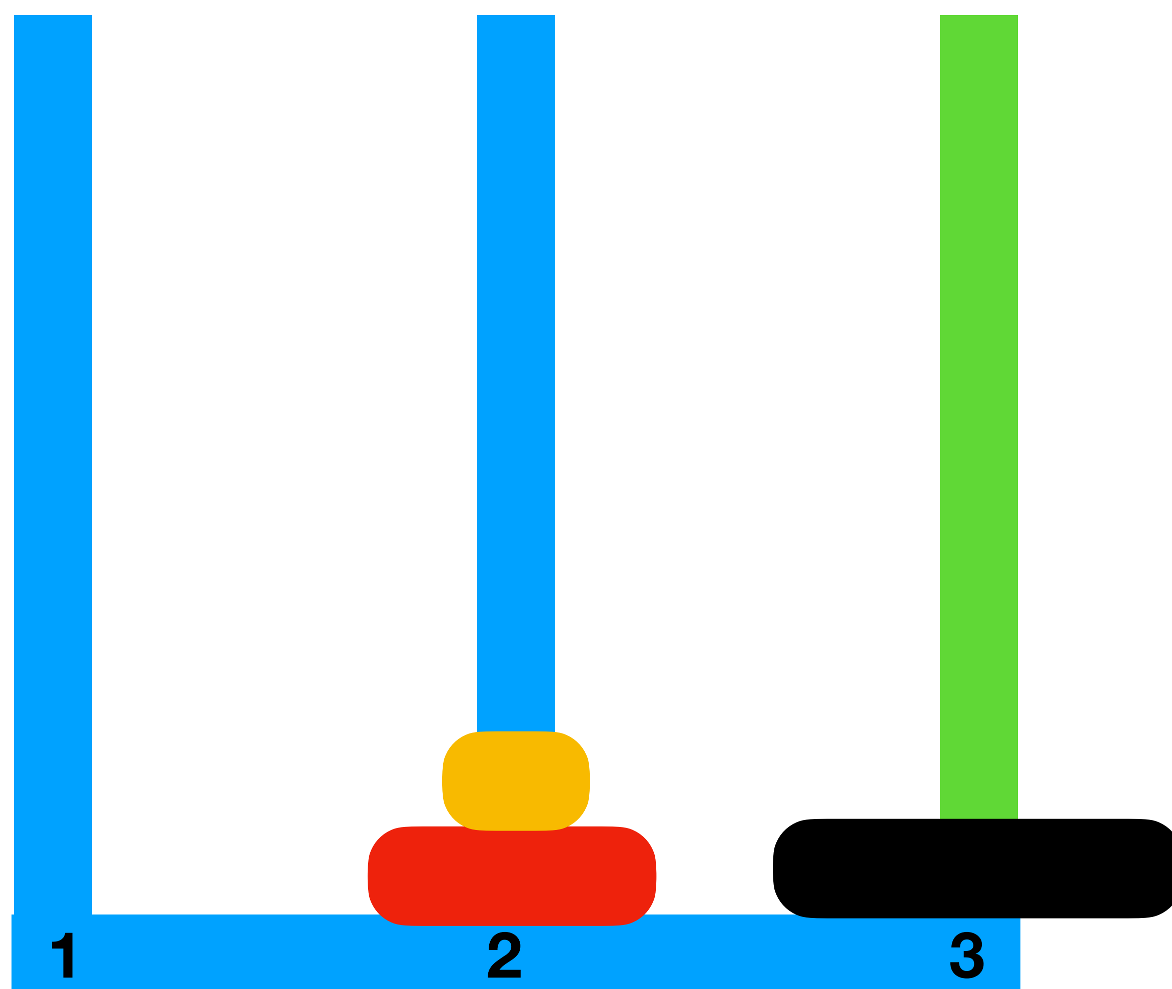
move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)



dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

Example: Towers of Hanoi



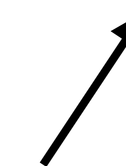
move(diskStacks, src, dest, temp) *// Move 2+ disks*
dmove(disk_num, src, dest) *// Move single disk*

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)

+

dmove(disk_3, r1, r3)

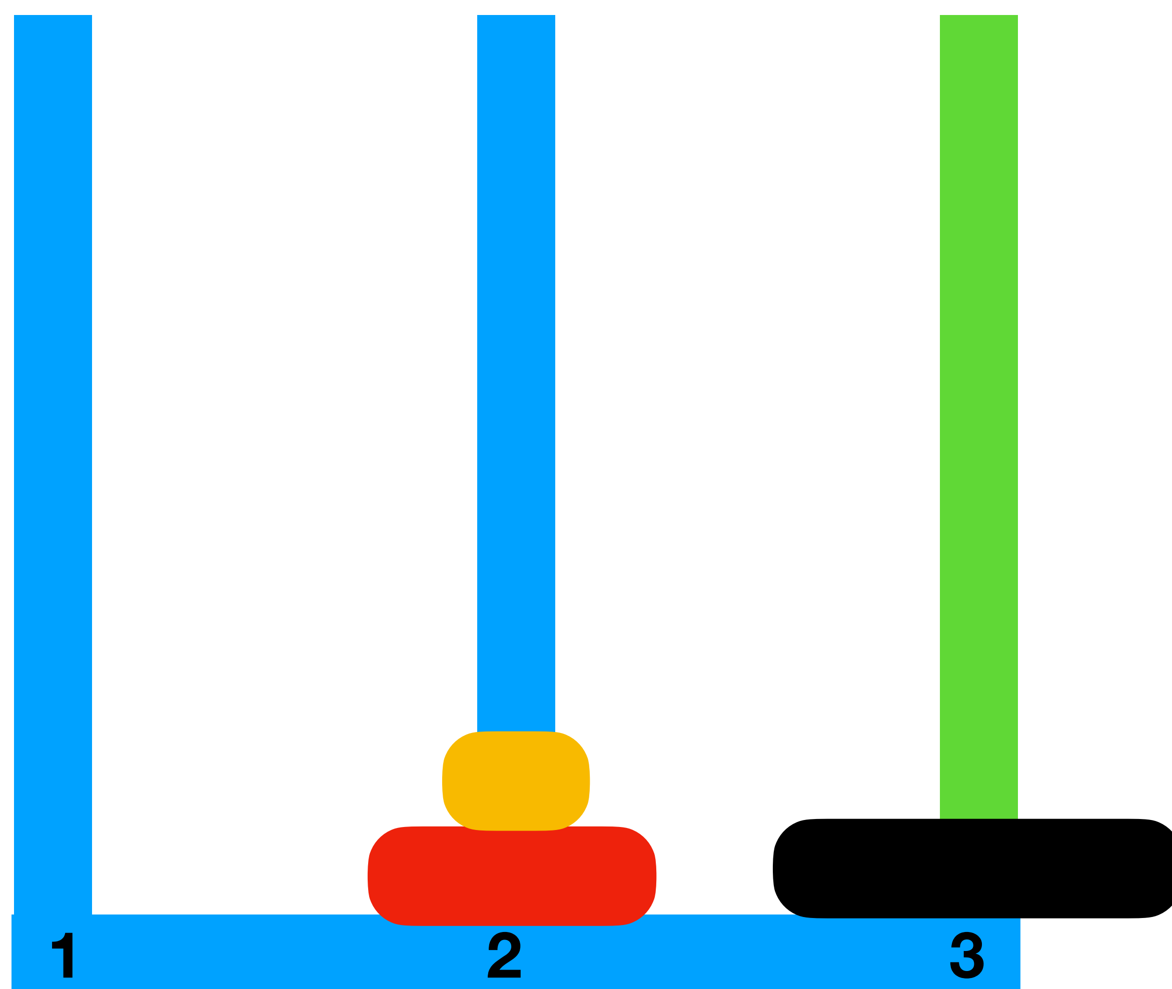


dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

dmove(disk_1, r3, r2)

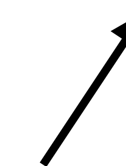
Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

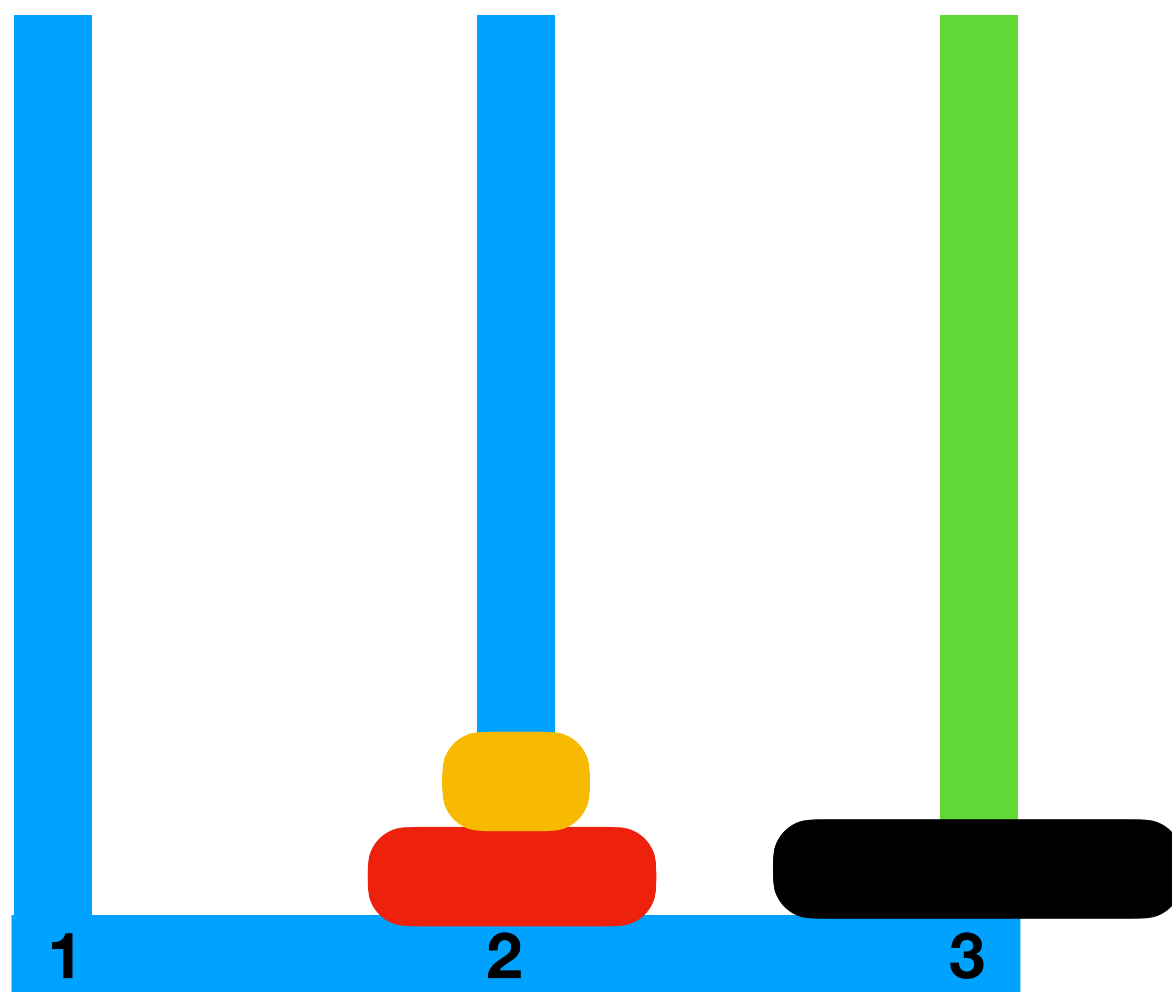


dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Towers of Hanoi

move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk



move(disks3, r1, r3, r2)

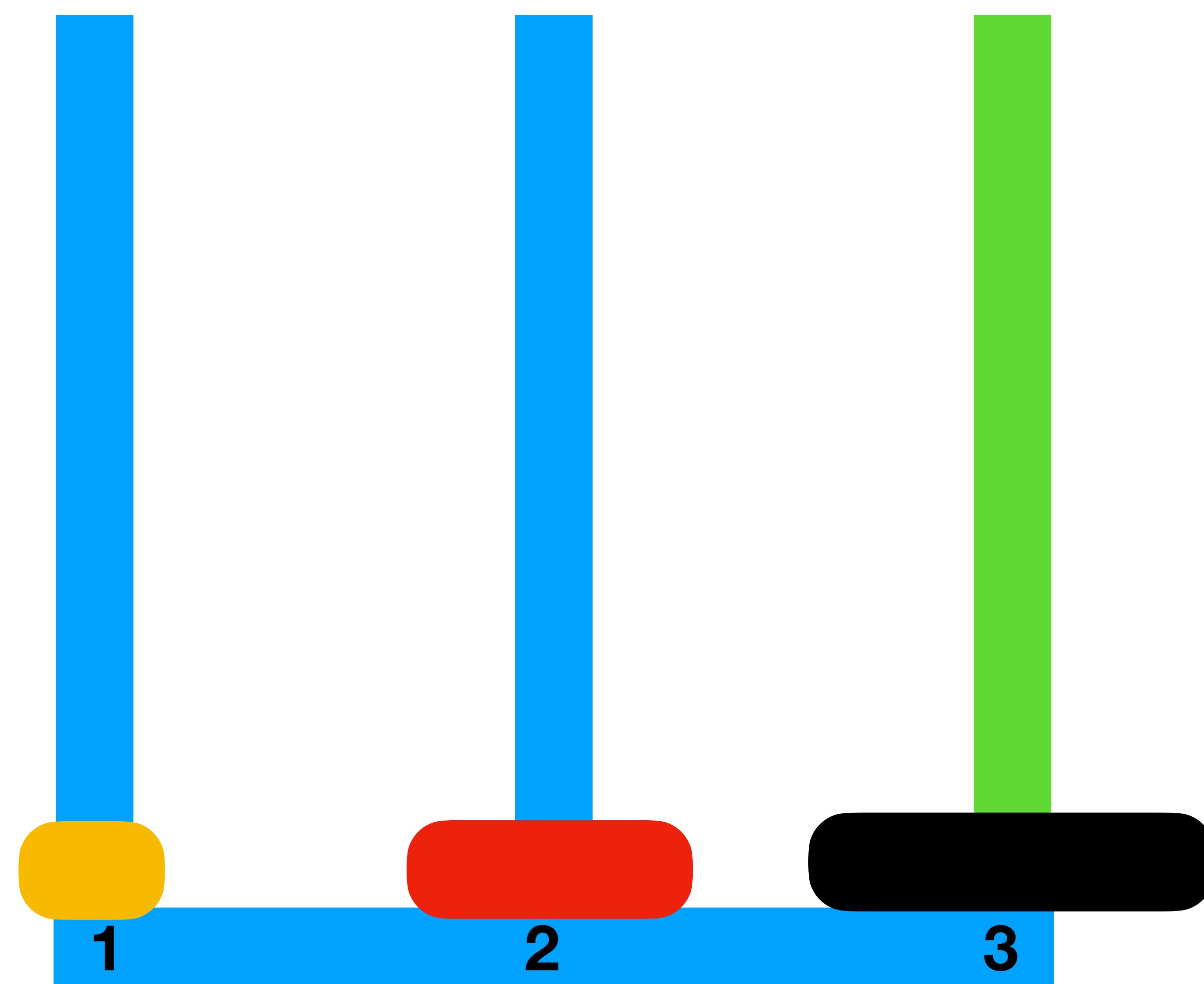
move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

dmove(disk_1, r2, r1)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

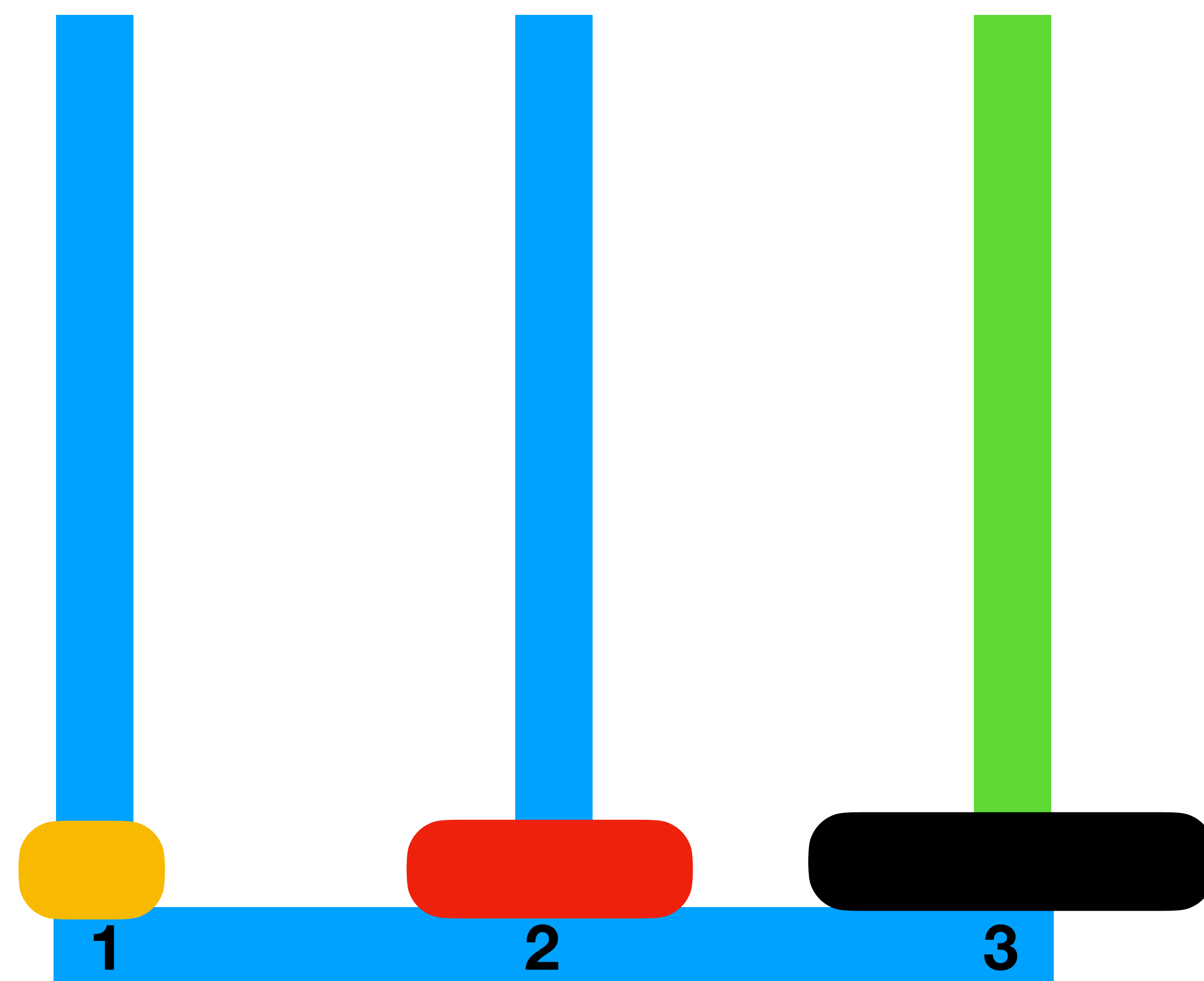
move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

dmove(disk_1, r2, r1)

Example: Towers of Hanoi



move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk

move(disks3, r1, r3, r2)

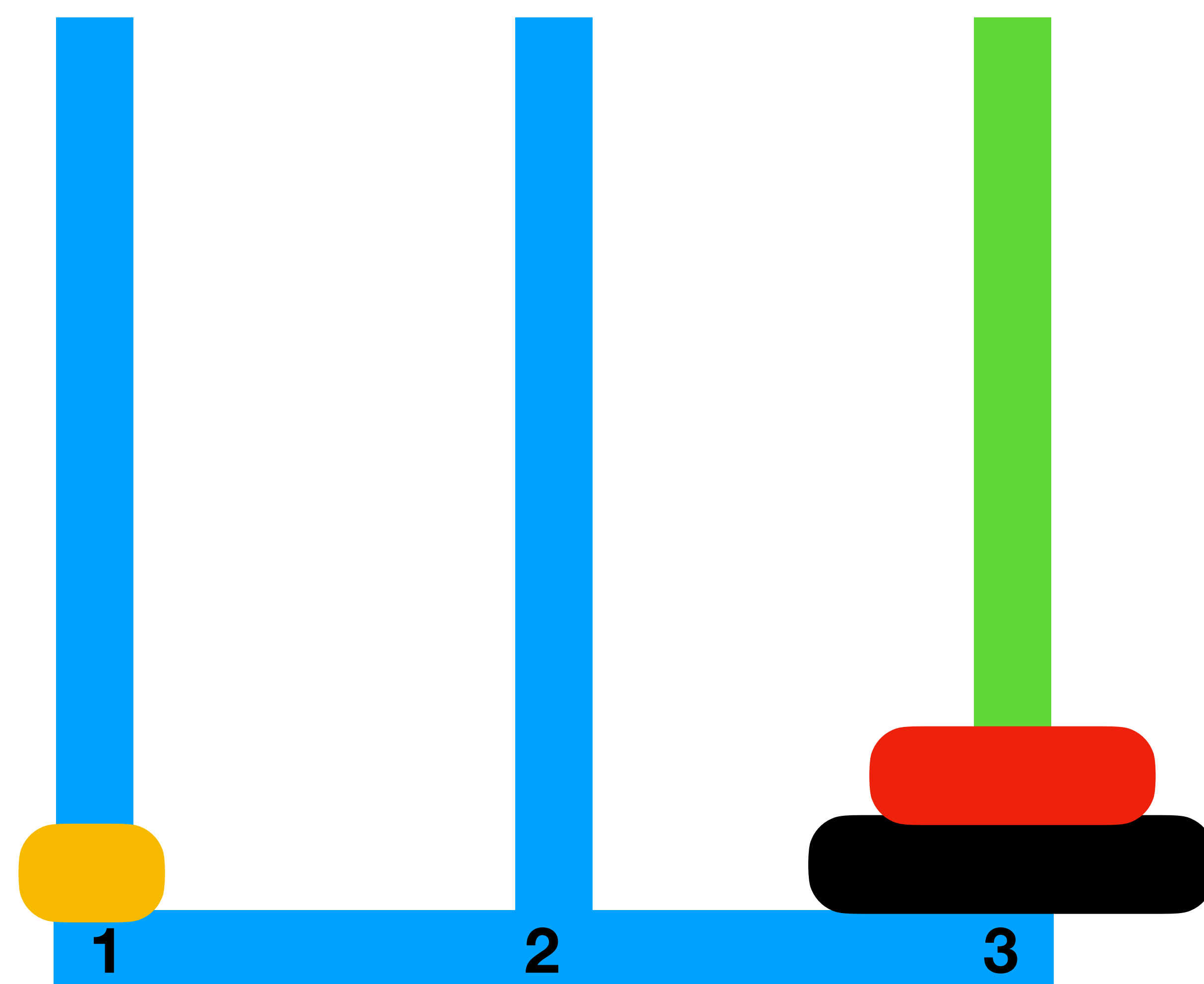
move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

dmove(disk_1, r2, r1)
dmove(disk_2, r2, r3)

Example: Towers of Hanoi

move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk



move(disks3, r1, r3, r2)

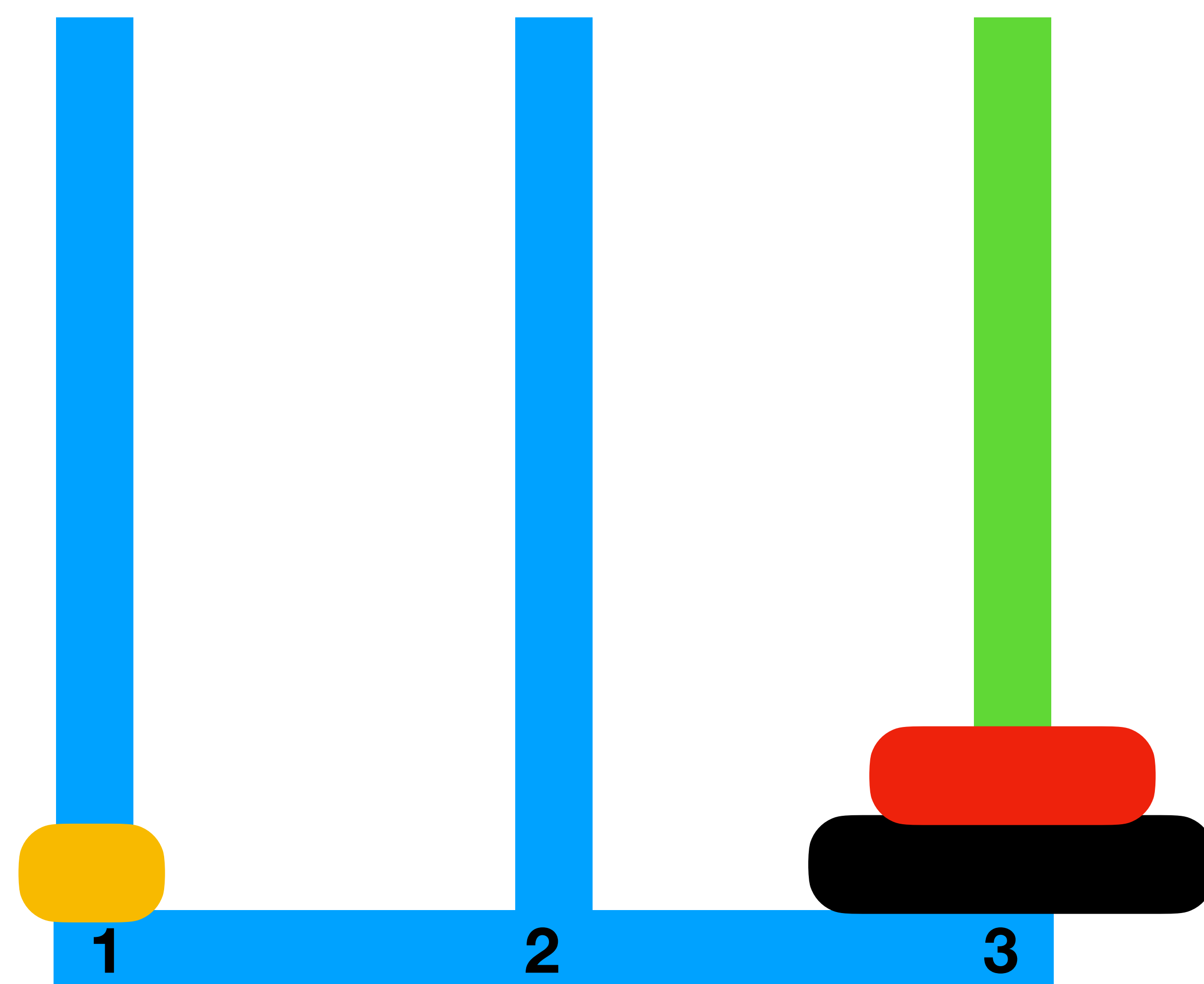
move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

dmove(disk_1, r2, r1)
dmove(disk_2, r2, r3)

Example: Towers of Hanoi

move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk



move(disks3, r1, r3, r2)

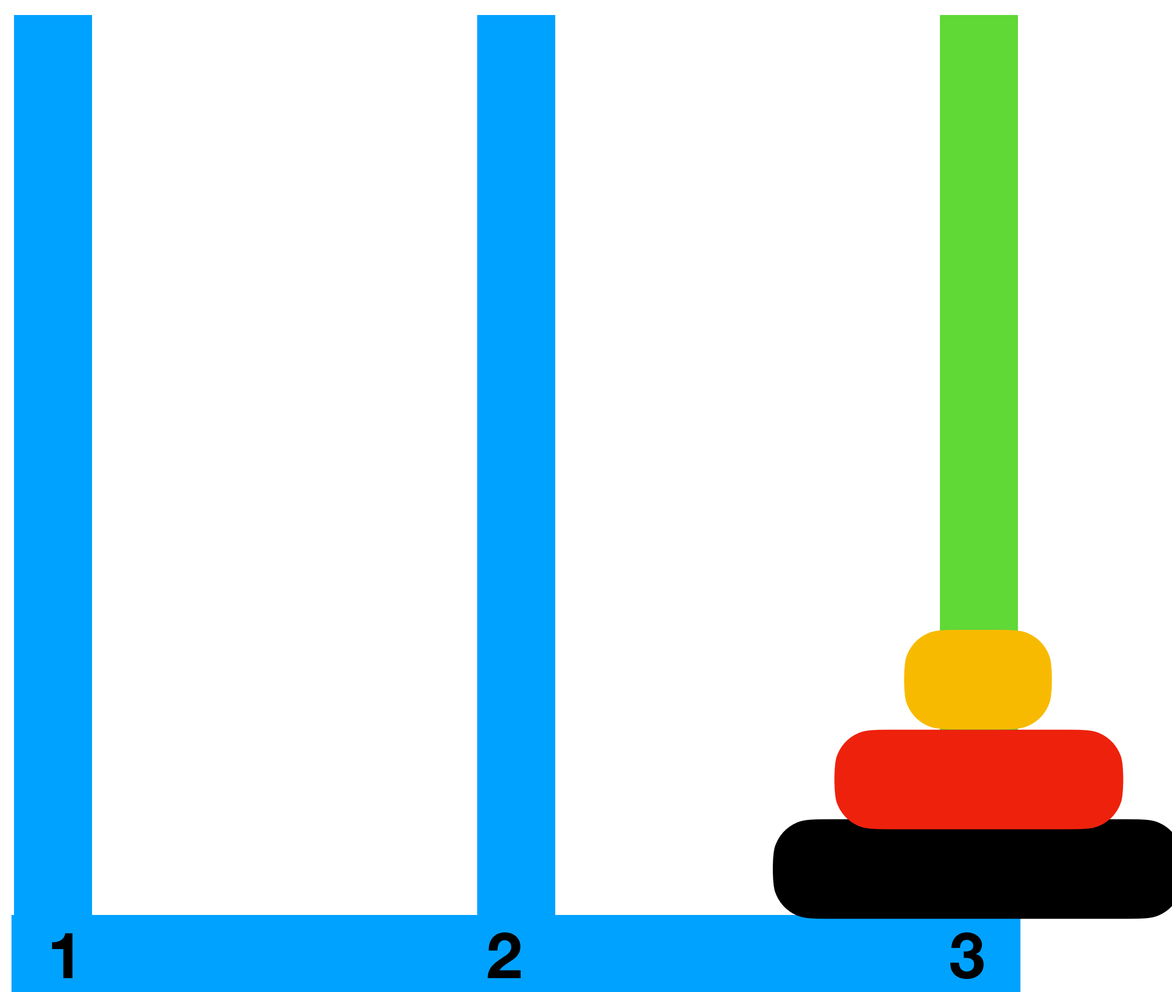
move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

dmove(disk_1, r2, r1)
dmove(disk_2, r2, r3)
dmove(disk_1, r1, r3)

Example: Towers of Hanoi

move(diskStacks, src, dest, temp) // Move 2+ disks
dmove(disk_num, src, dest) // Move single disk



move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)
+
dmove(disk_3, r1, r3)
+
move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)
dmove(disk_2, r1, r2)
dmove(disk_1, r3, r2)

dmove(disk_1, r2, r1)
dmove(disk_2, r2, r3)
dmove(disk_1, r1, r3)

Recursion in Towers of Hanoi

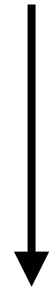
Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

move(disks4, r1, r3, r2)

Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)



move(disks2, r1, r2, r3)

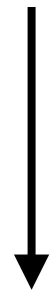
move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)

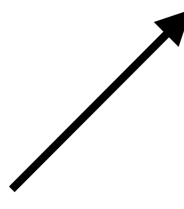
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)



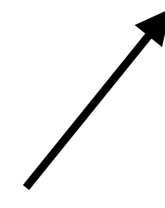
move(disks2, r1, r2, r3)



dmove(disk_1, r1, r3)

move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)

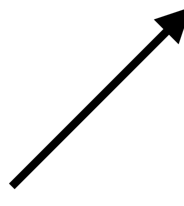


move(disks2, r1, r3, r2)

Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

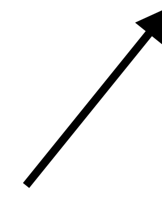
move(disks2, r1, r2, r3)



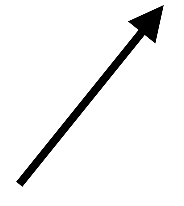
dmove(disk_1, r1, r3)

move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)



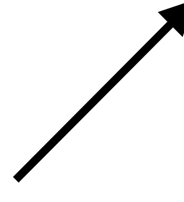
move(disks2, r1, r3, r2)



Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)

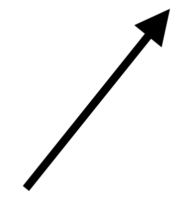


dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

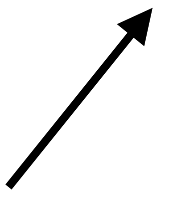
move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)



move(disks2, r1, r3, r2)

dmove(disk_3, r1, r2)

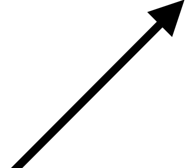


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)



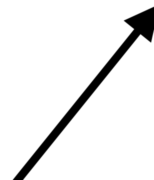
dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

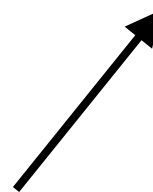
dmove(disk_1, r3, r2)

move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)



move(disks2, r1, r3, r2)



dmove(disk_3, r1, r2)

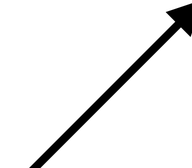
move(disks2, r3, r2, r1)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)



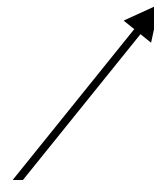
dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

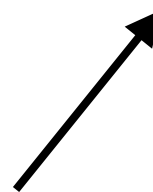
dmove(disk_1, r3, r2)

move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)

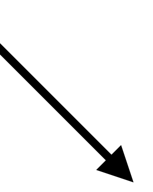


move(disks2, r1, r3, r2)



dmove(disk_3, r1, r2)

move(disks2, r3, r2, r1)



Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)

\+

dmove(disk_3, r1, r3)

dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

dmove(disk_1, r3, r2)

move(disks4, r1, r3, r2)

move(disks3, r1, r2, r3)

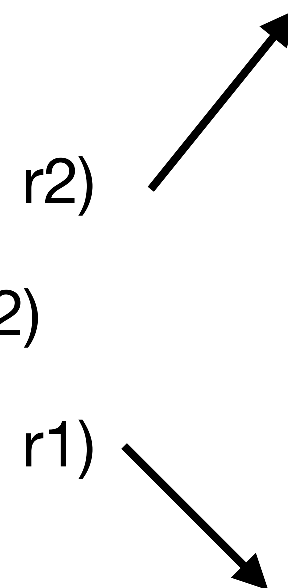
\+

dmove(disk_4, r1, r3)

move(disks2, r1, r3, r2)

dmove(disk_3, r1, r2)

move(disks2, r3, r2, r1)



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in Towers of Hanoi

move(disks3, r1, r3, r2)

move(disks2, r1, r2, r3)

\+

dmove(disk_3, r1, r3)

\+

move(disks2, r2, r3, r1)

dmove(disk_1, r1, r3)

dmove(disk_2, r1, r2)

dmove(disk_1, r3, r2)

move(disks4, r1, r3, r2)

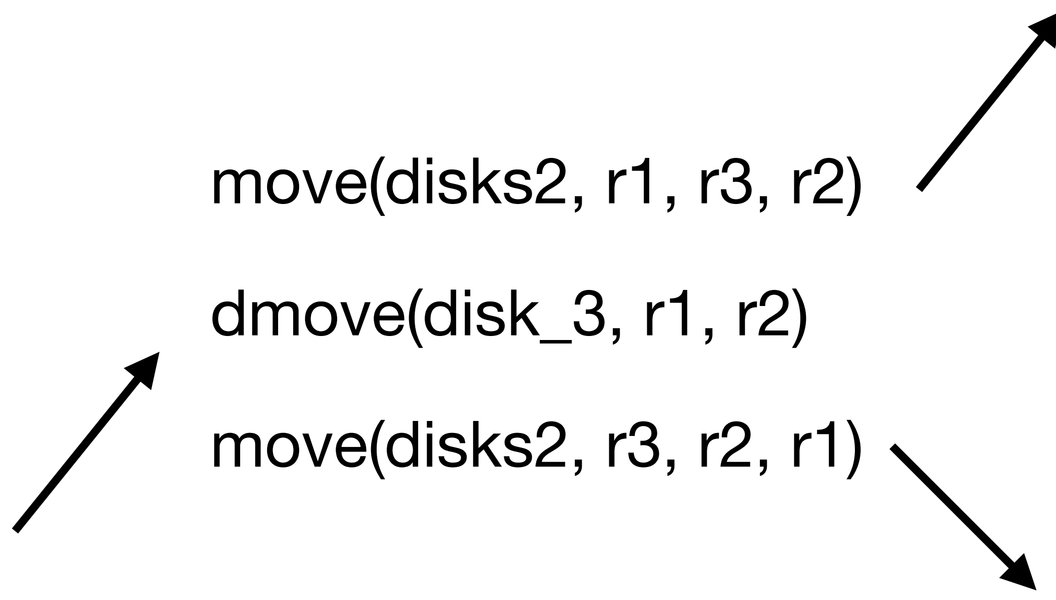
move(disks3, r1, r2, r3)

\+

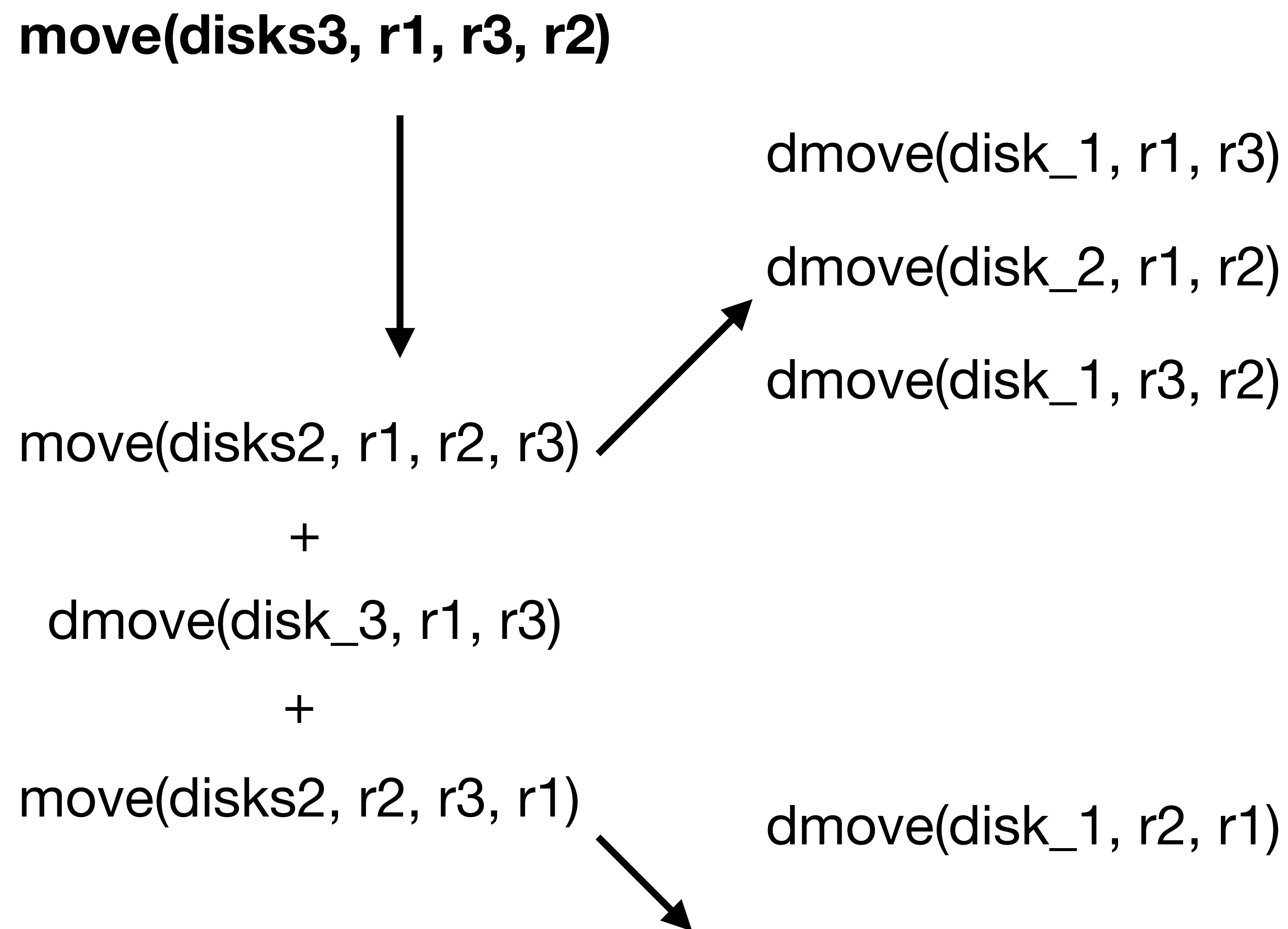
dmove(disk_4, r1, r3)

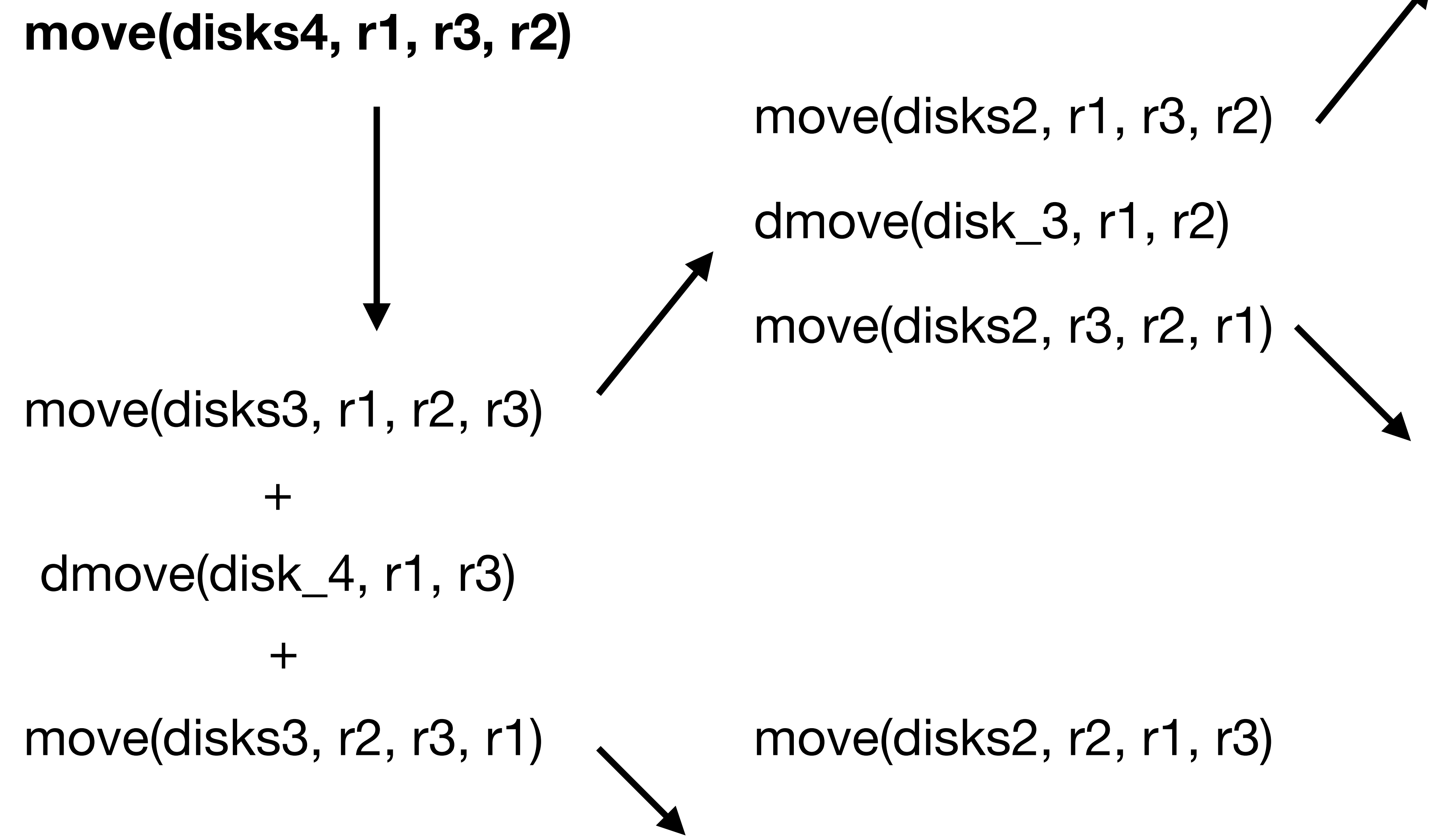
\+

move(disks3, r2, r3, r1)

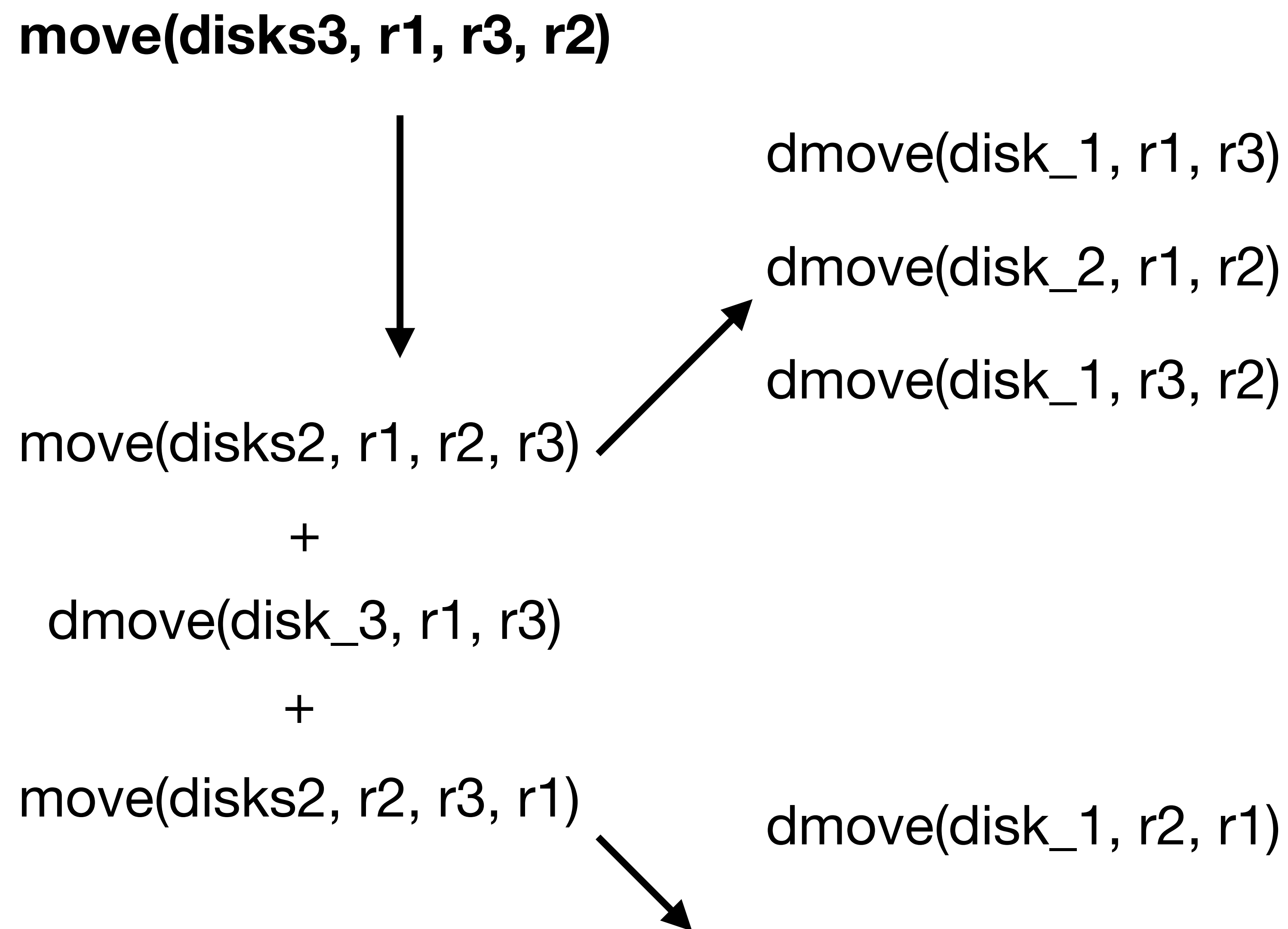


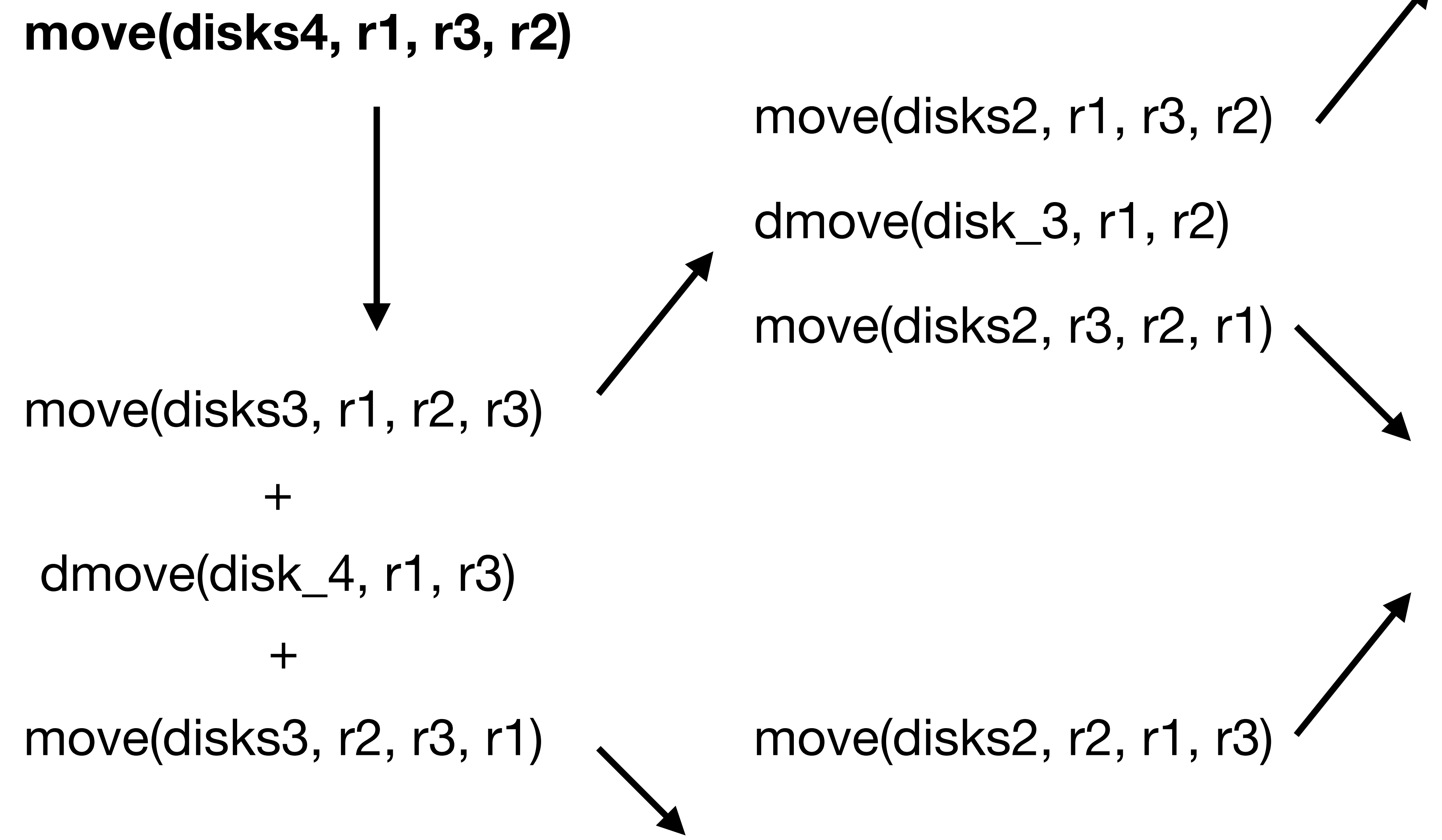
Recursion in Towers of Hanoi



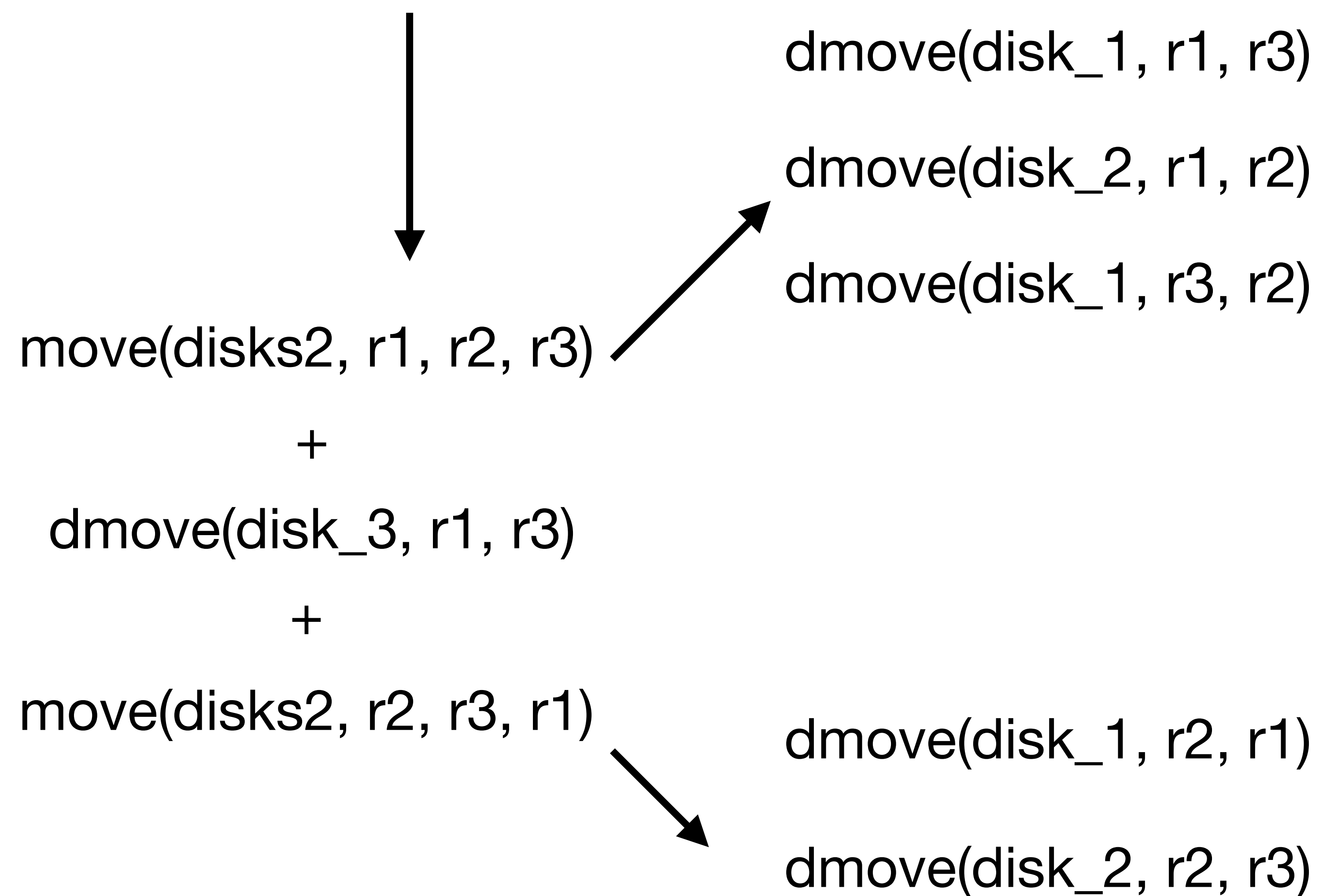


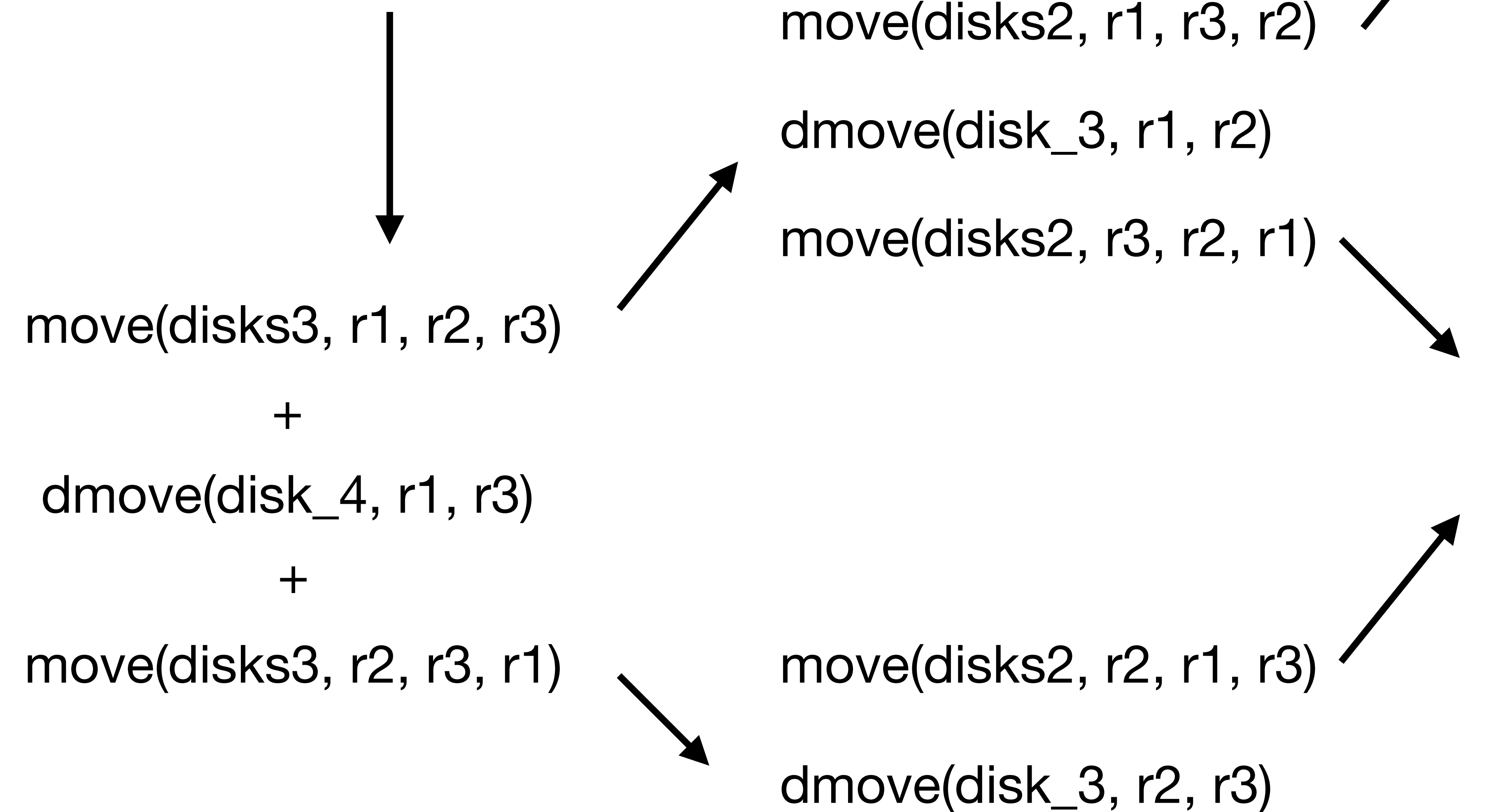
Recursion in Towers of Hanoi



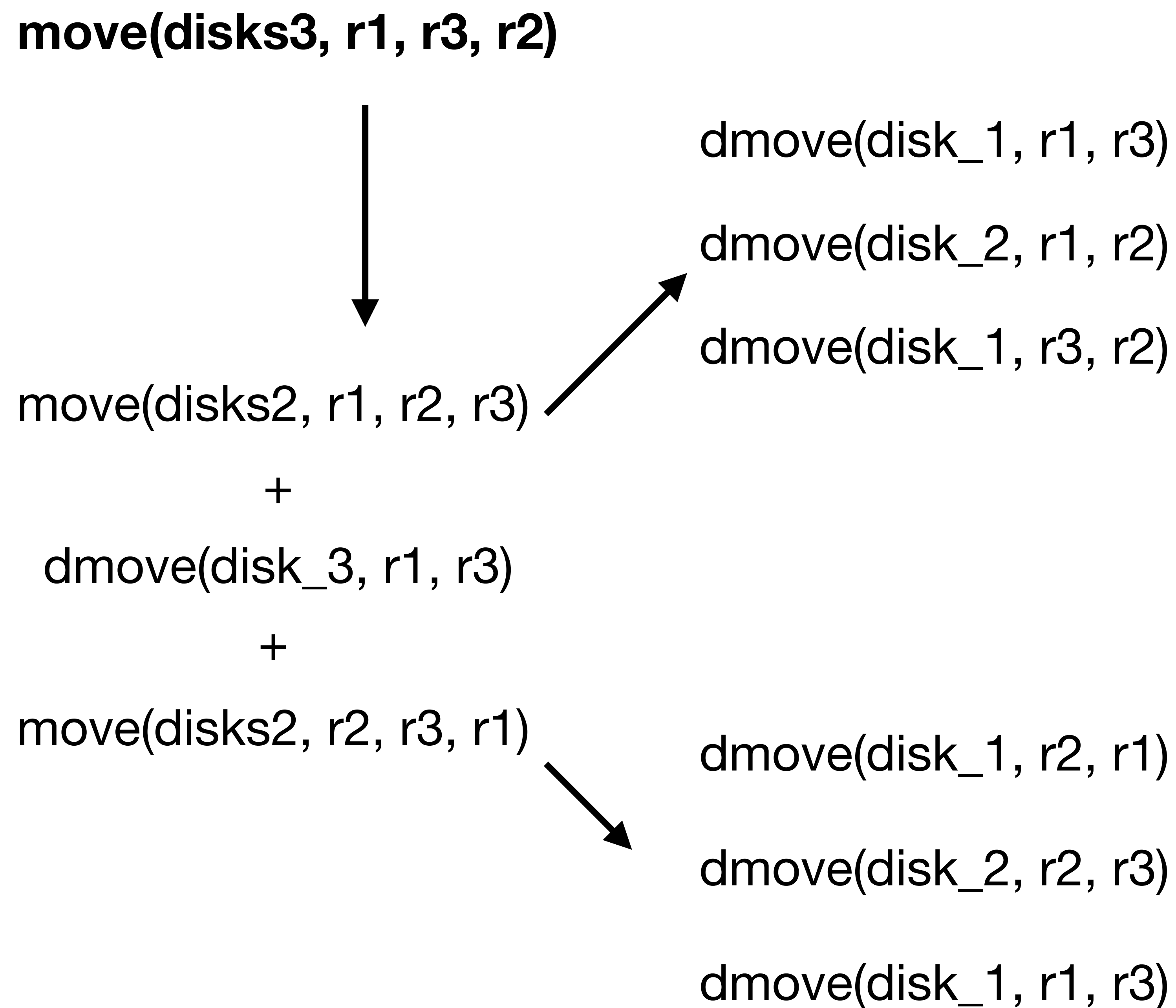


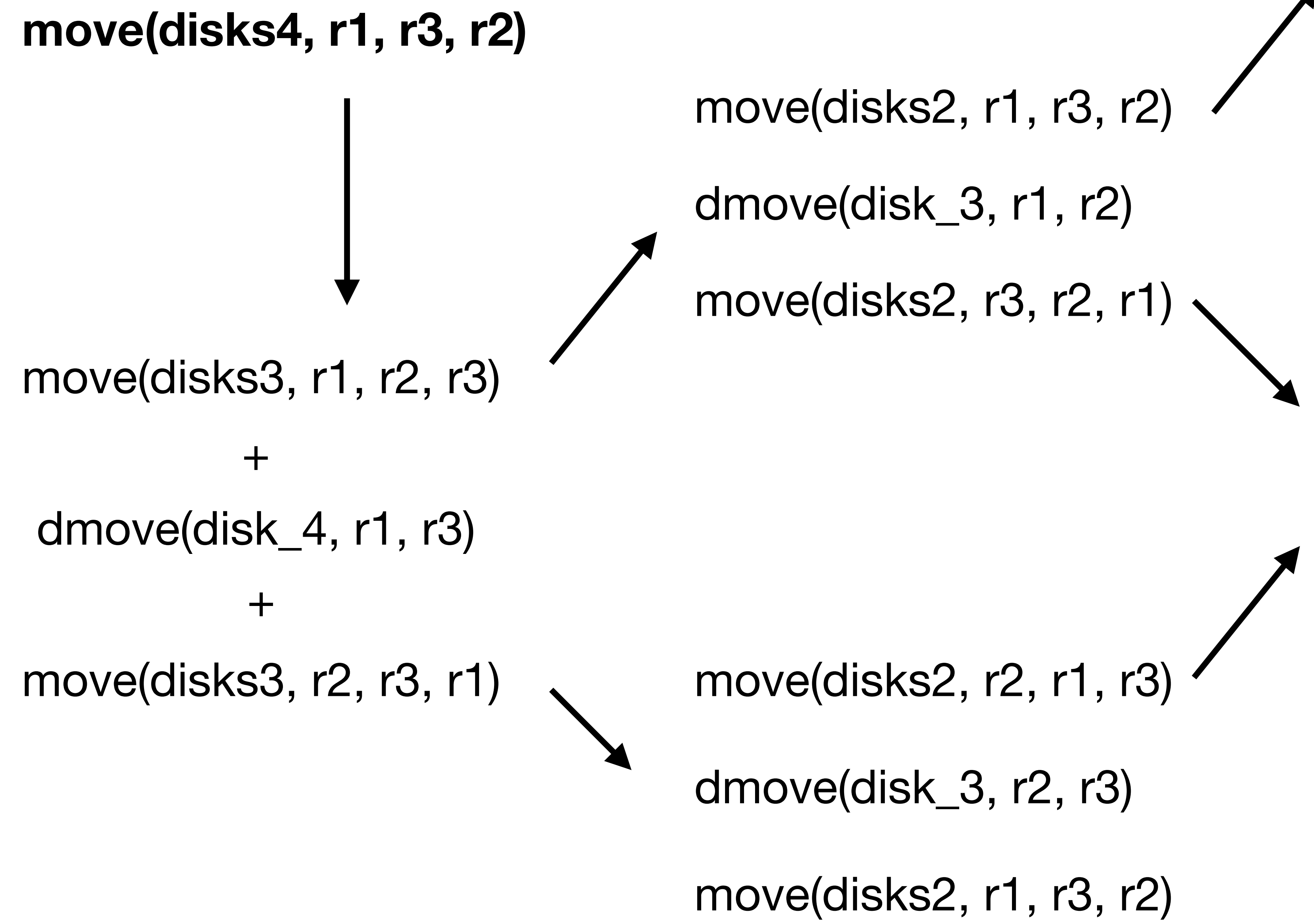
Recursion in Towers of Hanoi



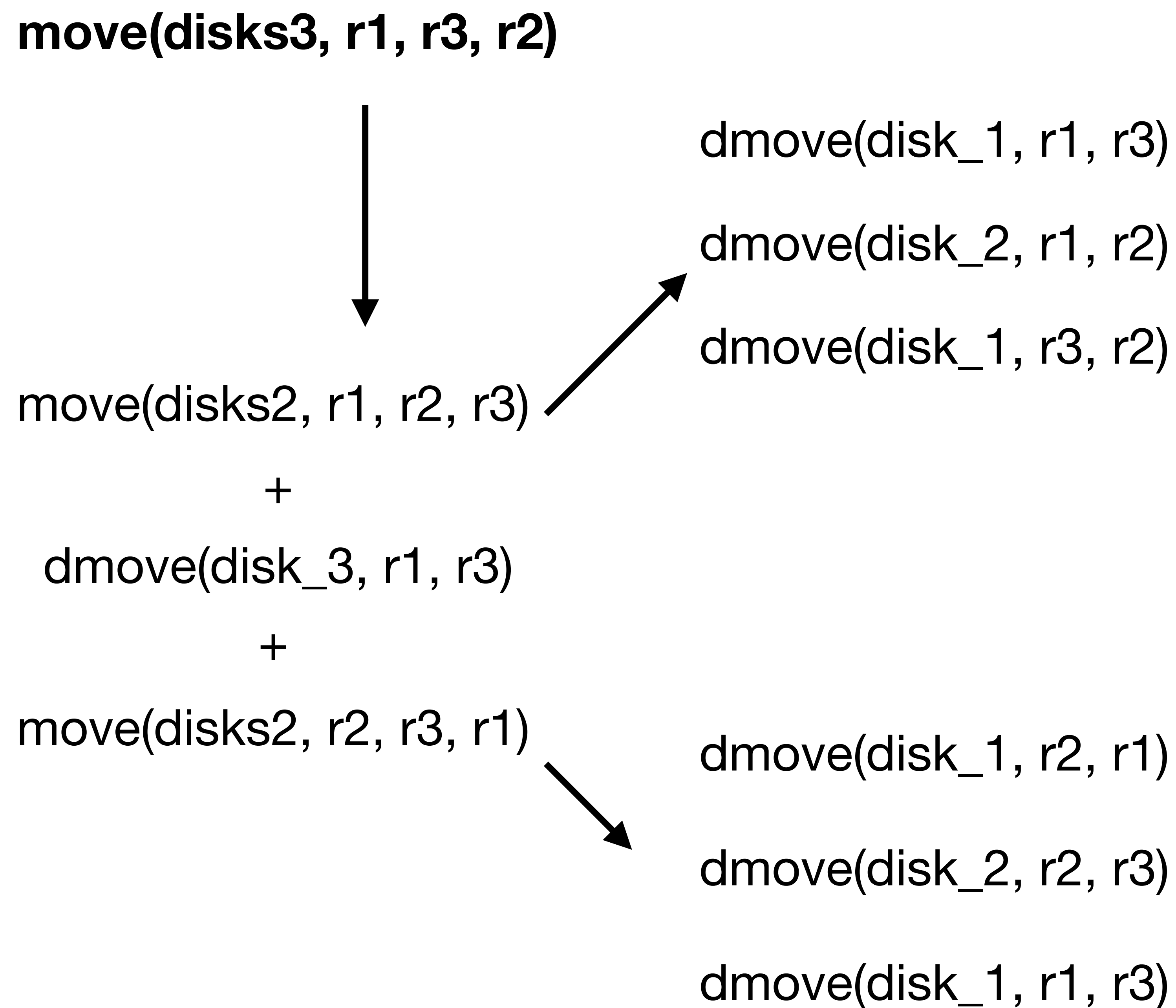


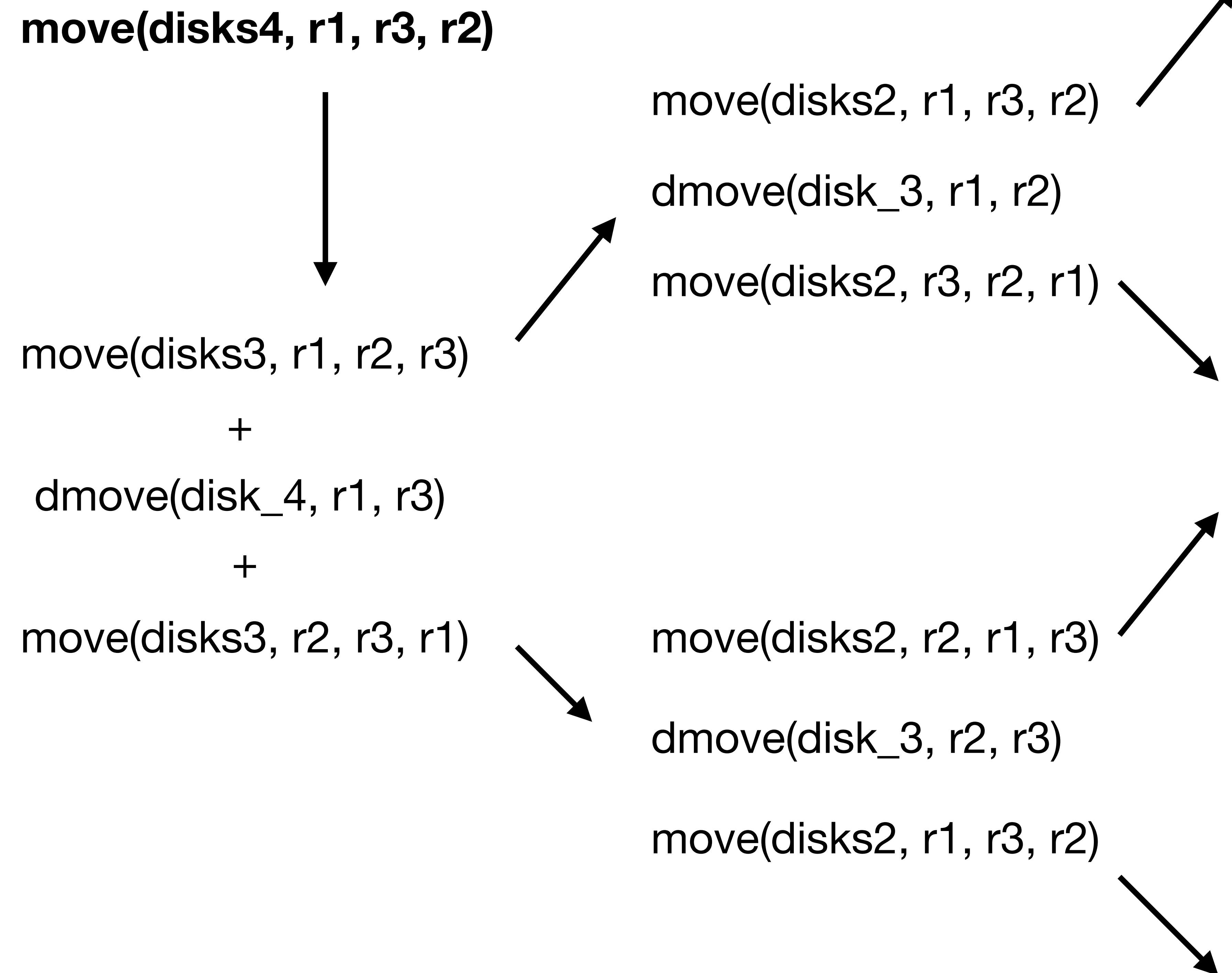
Recursion in Towers of Hanoi





Recursion in Towers of Hanoi





Towers of Hanoi: General formula?

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Towers of Hanoi: General formula?

```
void move(disknum, src, dest, temp){
  if (disknum > 1){
    move(disknum - 1, src, temp, dest);

    printf("Moved disk %d from rod %d to %d\n", disknum, src, dest);

    move(disknum - 1, temp, dest, src);
  }
  else
    printf("Moved disk 1 from rod %d to %d\n", src, dest);
}
```

Towers of Hanoi: General formula?

```
void move(disknum, src, dest, temp){
  if (disknum > 1){
    move(disknum - 1, src, temp, dest);

    printf("Moved disk %d from rod %d to %d\n", disknum, src, dest);

    move(disknum - 1, temp, dest, src);
  }
  else
    printf("Moved disk 1 from rod %d to %d\n", src, dest);
}
```

Move a sub-stack to from src to intermediate rod

Towers of Hanoi: General formula?

```
void move(disknum, src, dest, temp){
  if (disknum > 1){
    move(disknum - 1, src, temp, dest);

    printf("Moved disk %d from rod %d to %d\n", disknum, src, dest);

    move(disknum - 1, temp, dest, src);
  }
  else
    printf("Moved disk 1 from rod %d to %d\n", src, dest);
}
```

Move a sub-stack to from src to intermediate rod

Direct move of single disk.

Towers of Hanoi: General formula?

```
void move(disknum, src, dest, temp){
  if (disknum > 1){
    move(disknum - 1, src, temp, dest);

    printf("Moved disk %d from rod %d to %d\n", disknum, src, dest);

    move(disknum - 1, temp, dest, src);
  }
  else
    printf("Moved disk 1 from rod %d to %d\n", src, dest);
}
```

Move a sub-stack to from src to intermediate rod

Direct move of single disk.

Move the sub-stack from intermediate to dest rod

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Towers of Hanoi: General formula?

```
void move(disknum, src, dest, temp){
  if (disknum > 1){
    move(disknum - 1, src, temp, dest);

    printf("Moved disk %d from rod %d to %d\n", disknum, src, dest);

    move(disknum - 1, temp, dest, src);
  }
  else
    printf("Moved disk 1 from rod %d to %d\n", src, dest);
}
```

Move a sub-stack to from src to intermediate rod

Direct move of single disk.

Move the sub-stack from intermediate to dest rod

Base case; all moves involving sub-stacks end-up here.

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in LC3

Recursion in LC3

```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 5;
    running_sum(5);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in LC3

```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 5;
    running_sum(5);
}
```

How can we write equivalent LC3 code?

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Recursion in LC3

```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 5;
    running_sum(5);
}
```

How can we write equivalent LC3 code?

Recall function calls are implemented using the run time stack.

Recursion in LC3

```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 5;
    running_sum(5);
}
```

How can we write equivalent LC3 code?

Recall function calls are implemented using the run time stack.

Recursive calls need not be treated any different from normal function calls!

Generating an activation record

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)

Caller

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)

Caller

3. *Callee* build-up: (push bookkeeping info and local variables onto stack)

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)
3. *Callee* build-up: (push bookkeeping info and local variables onto stack)
4. Execute function

Caller

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR`/`JSRR`)
3. *Callee* build-up: (push bookkeeping info and local variables onto stack)
4. Execute function
5. *Callee* tear-down (update return value, pop local variables, caller's frame pointer and return address from stack)

Caller

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)
3. *Callee* build-up: (push bookkeeping info and local variables onto stack)
4. Execute function
5. *Callee* tear-down (update return value, pop local variables, caller's frame pointer and return address from stack)
6. Return to caller (`RET`)

Caller

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)
3. *Callee* build-up: (push bookkeeping info and local variables onto stack)
4. Execute function
5. *Callee* tear-down (update return value, pop local variables, caller's frame pointer and return address from stack)
6. Return to caller (`RET`)

Caller

Callee

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)
3. *Callee* build-up: (push bookkeeping info and local variables onto stack)
4. Execute function
5. *Callee* tear-down (update return value, pop local variables, caller's frame pointer and return address from stack)
6. Return to caller (`RET`)
7. *Caller* tear-down (pop callee's return value and arguments from stack)

Caller

Callee

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

Generating an activation record

1. *Caller* build-up: Push callee's arguments onto stack
2. Pass control to callee (`JSR/JSRR`)

Caller

3. *Callee* build-up: (push bookkeeping info and local variables onto stack)
4. Execute function
5. *Callee* tear-down (update return value, pop local variables, caller's frame pointer and return address from stack)
6. Return to caller (`RET`)

Callee

7. *Caller* tear-down (pop callee's return value and arguments from stack)

Caller

Check Gitlab for reference material: https://gitlab.engr.illinois.edu/itabrah2/ece220-sp24

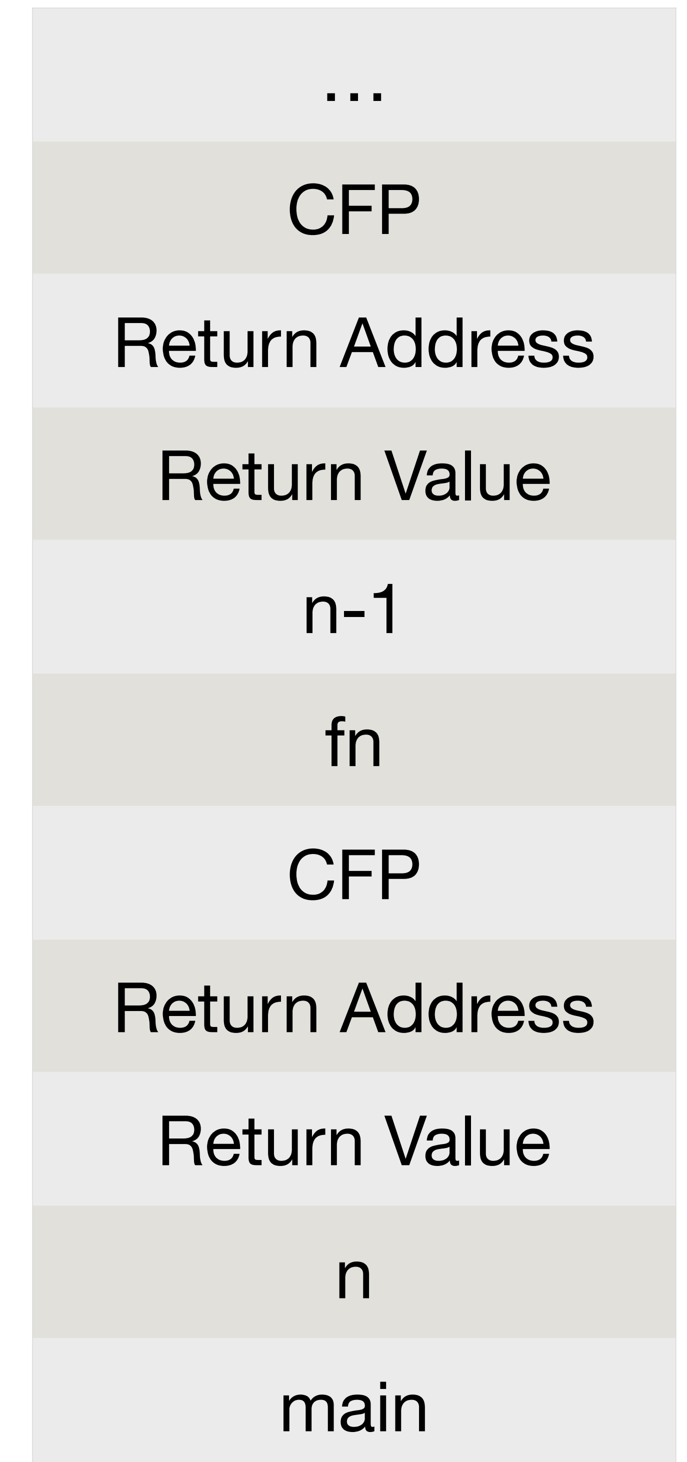
Recursion in LC3

```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 4;
    running_sum(4);
}
```


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

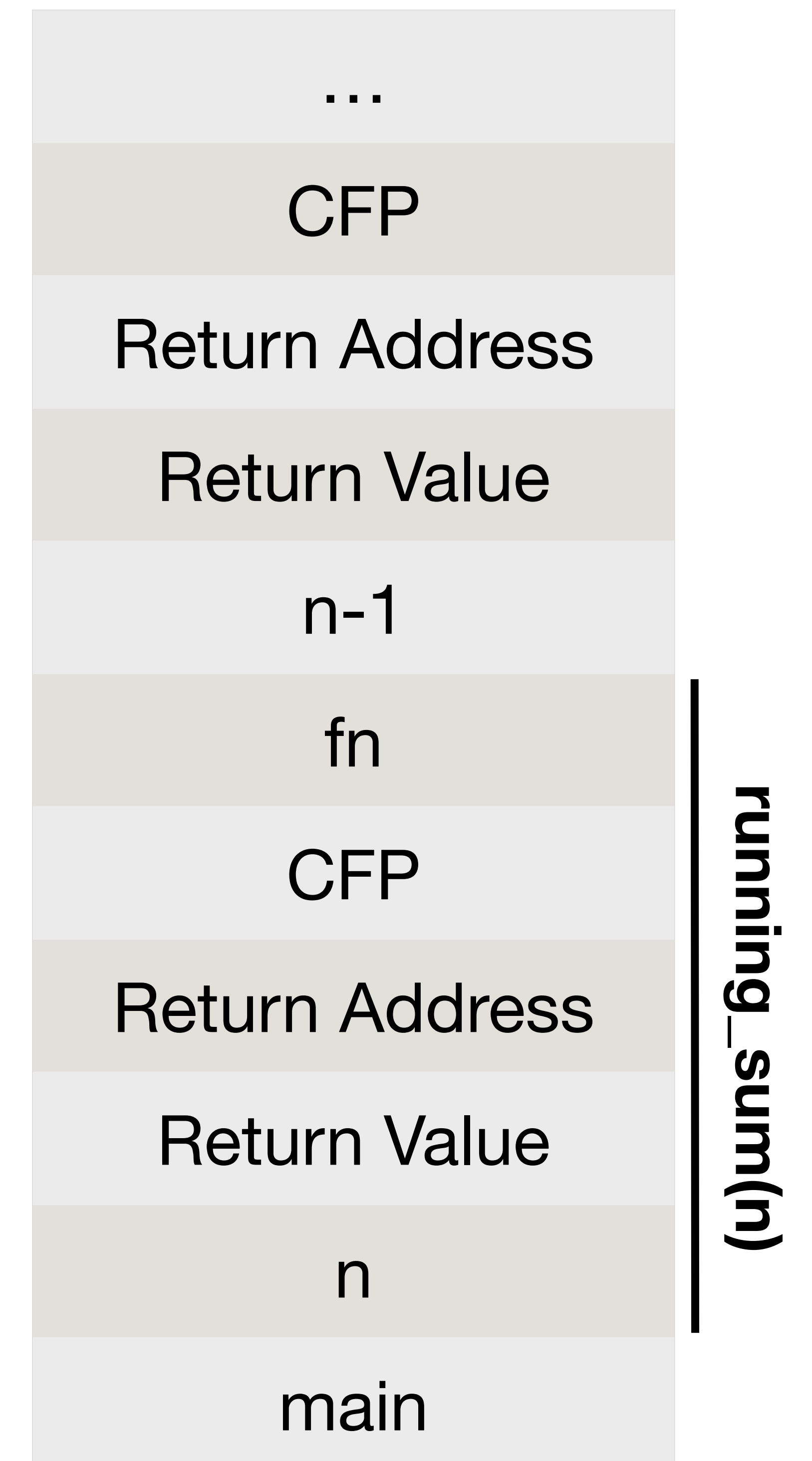
Recursion in LC3

```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 4;
    running_sum(4);
}
```



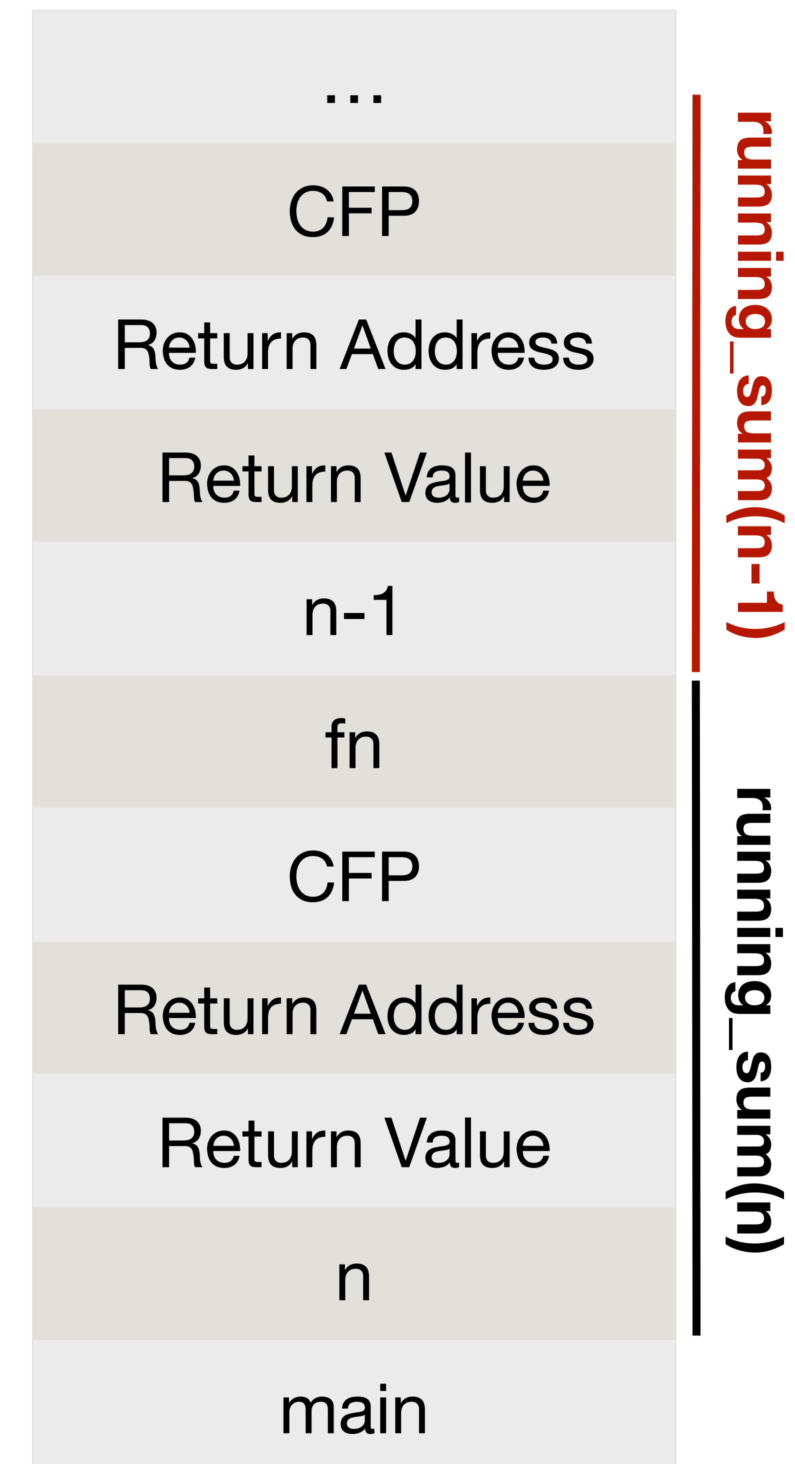
Recursion in LC3

```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 4;
    running_sum(4);
}
```



Recursion in LC3

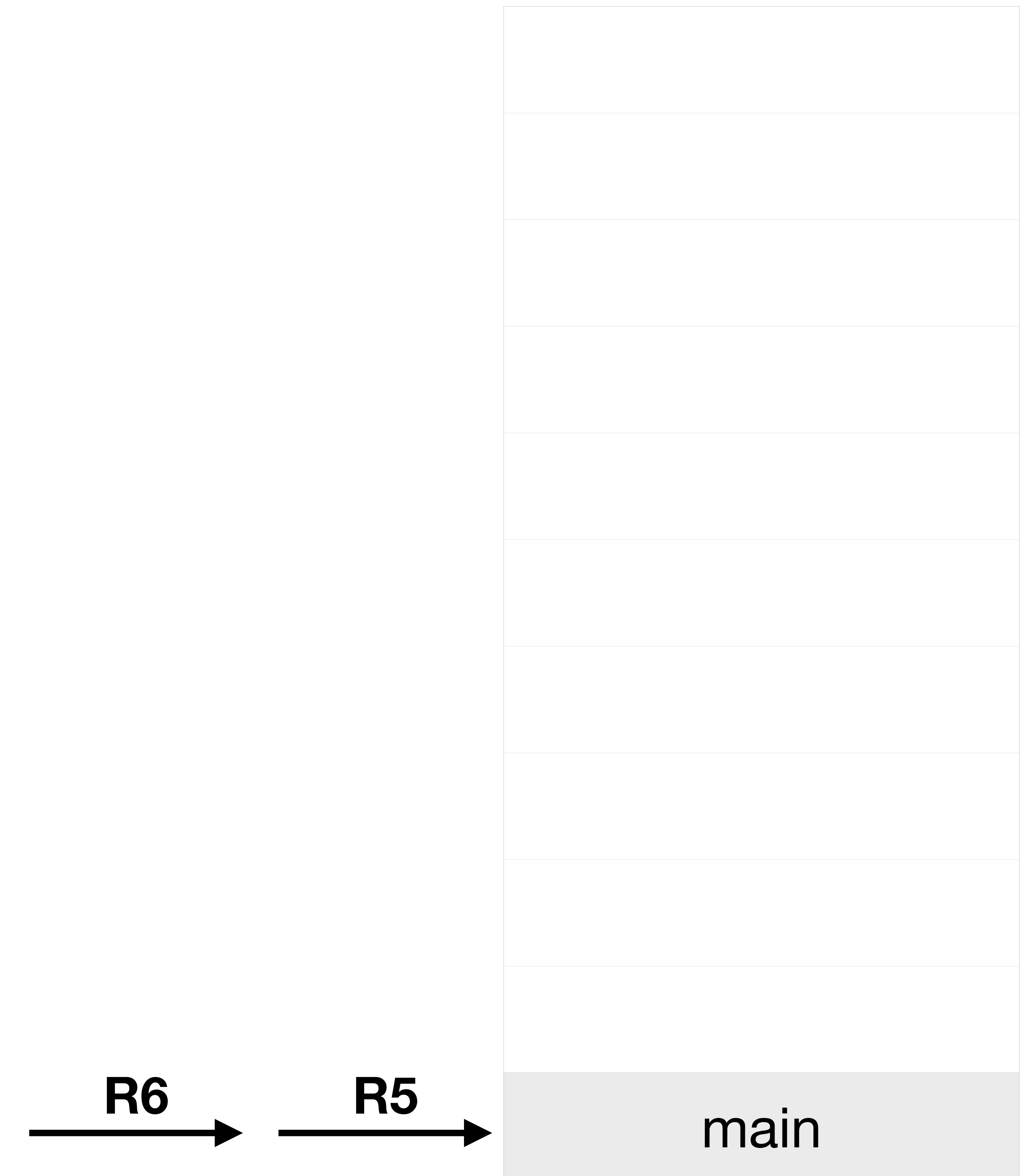
```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}

int main(void){
    int n = 4;
    running_sum(4);
}
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

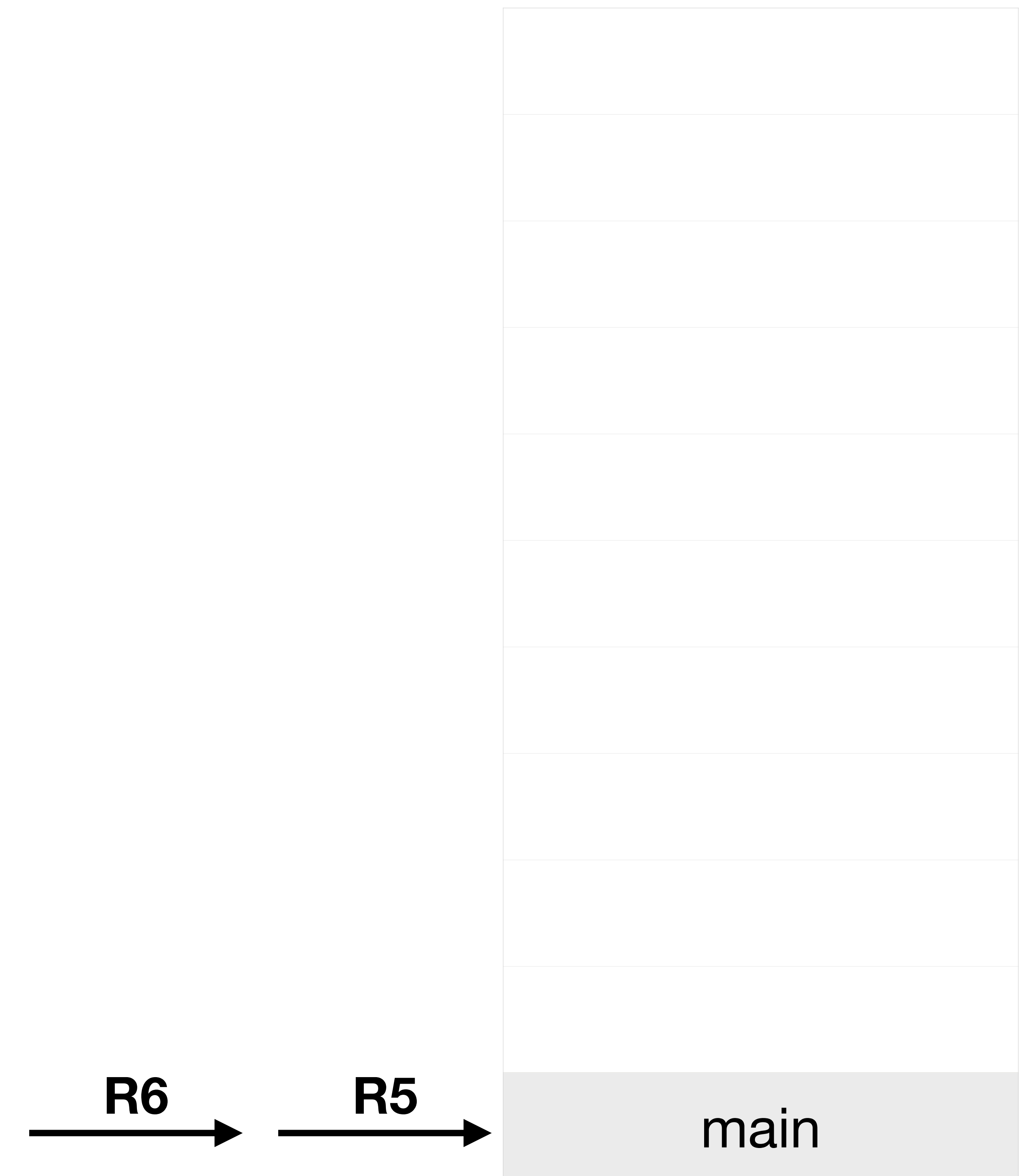
```
;Caller set-up
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

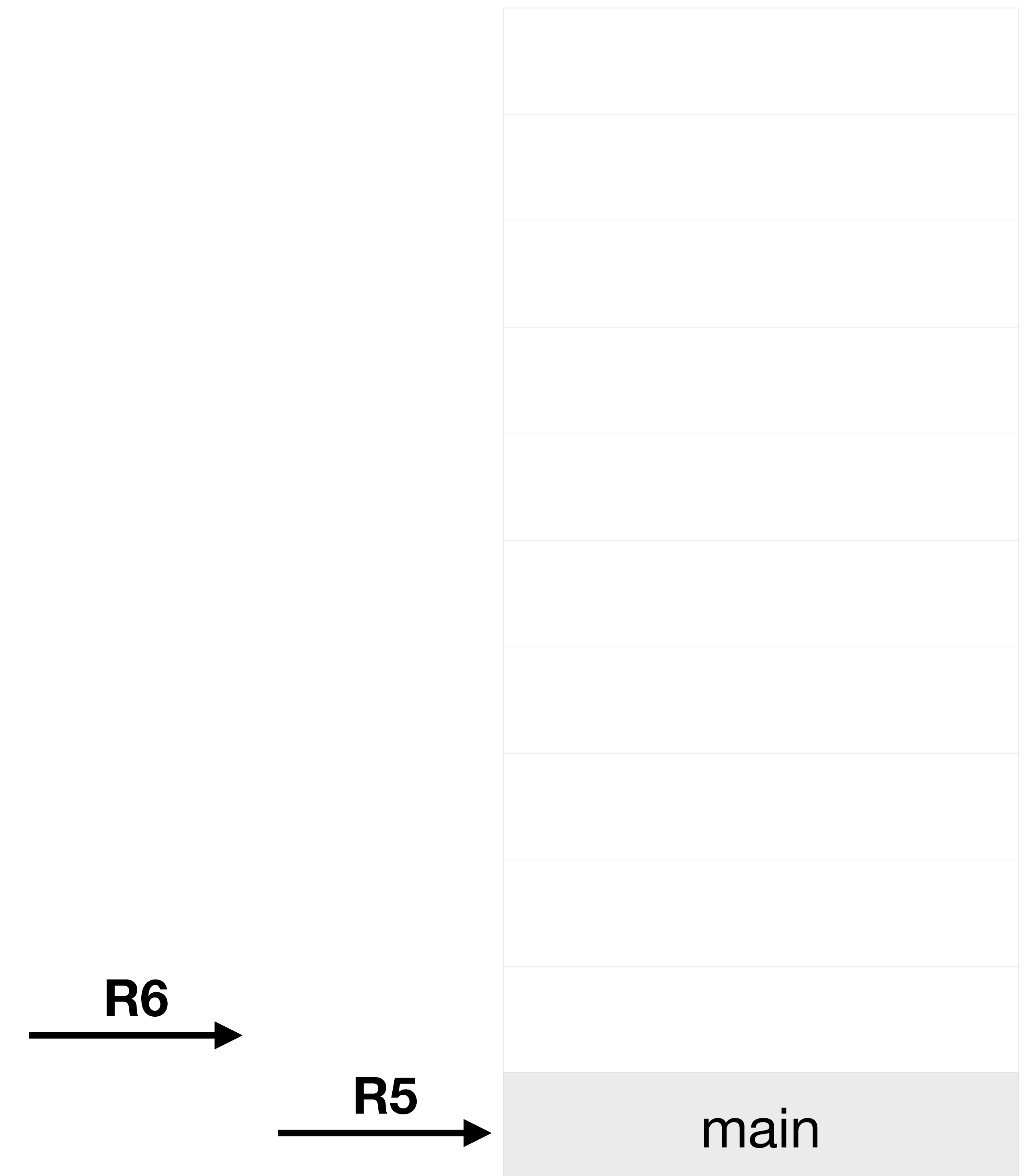
```
;Caller set-up
ADD R0, R0, #4
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

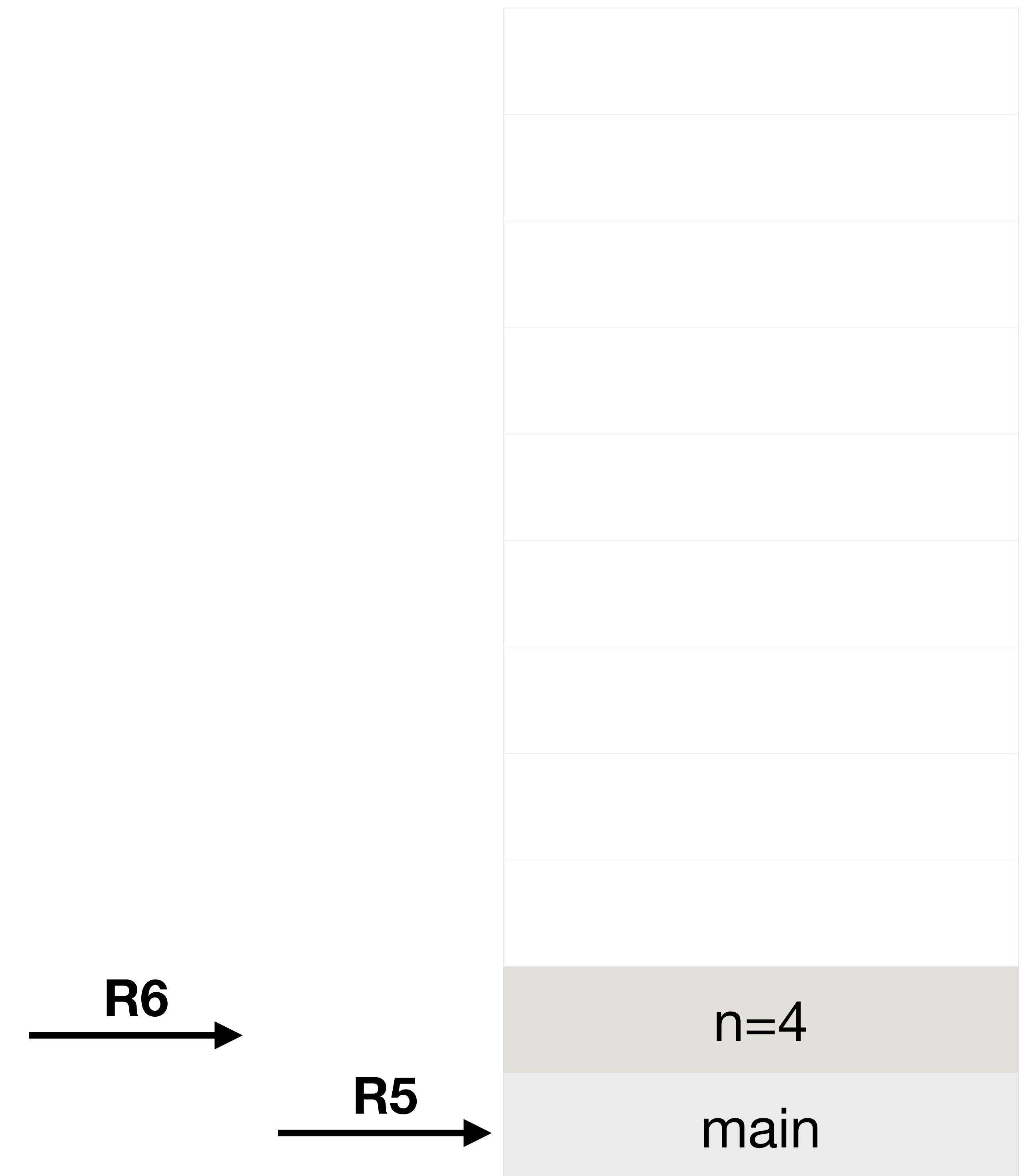
```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0
```

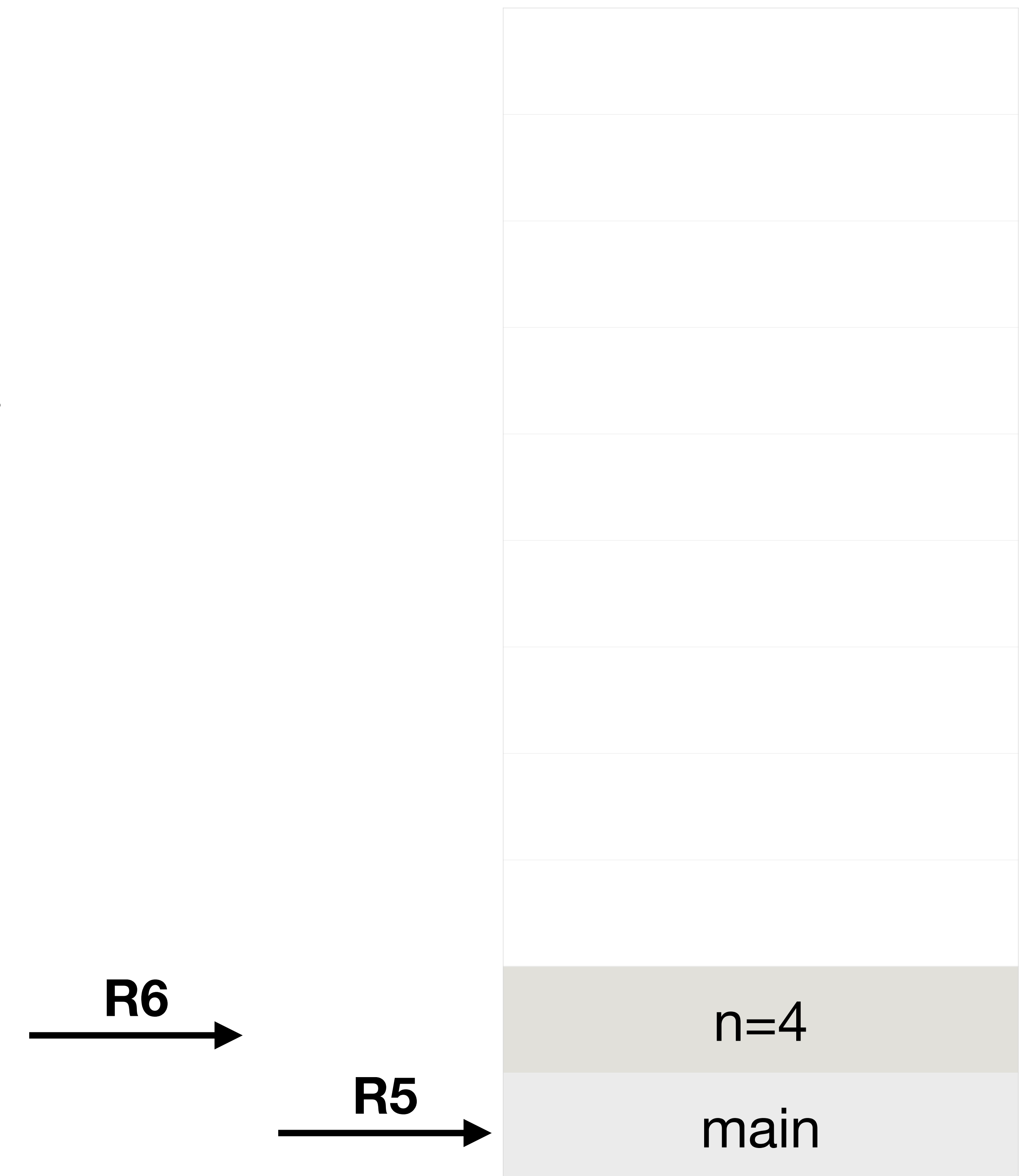


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0
```

```
RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
```

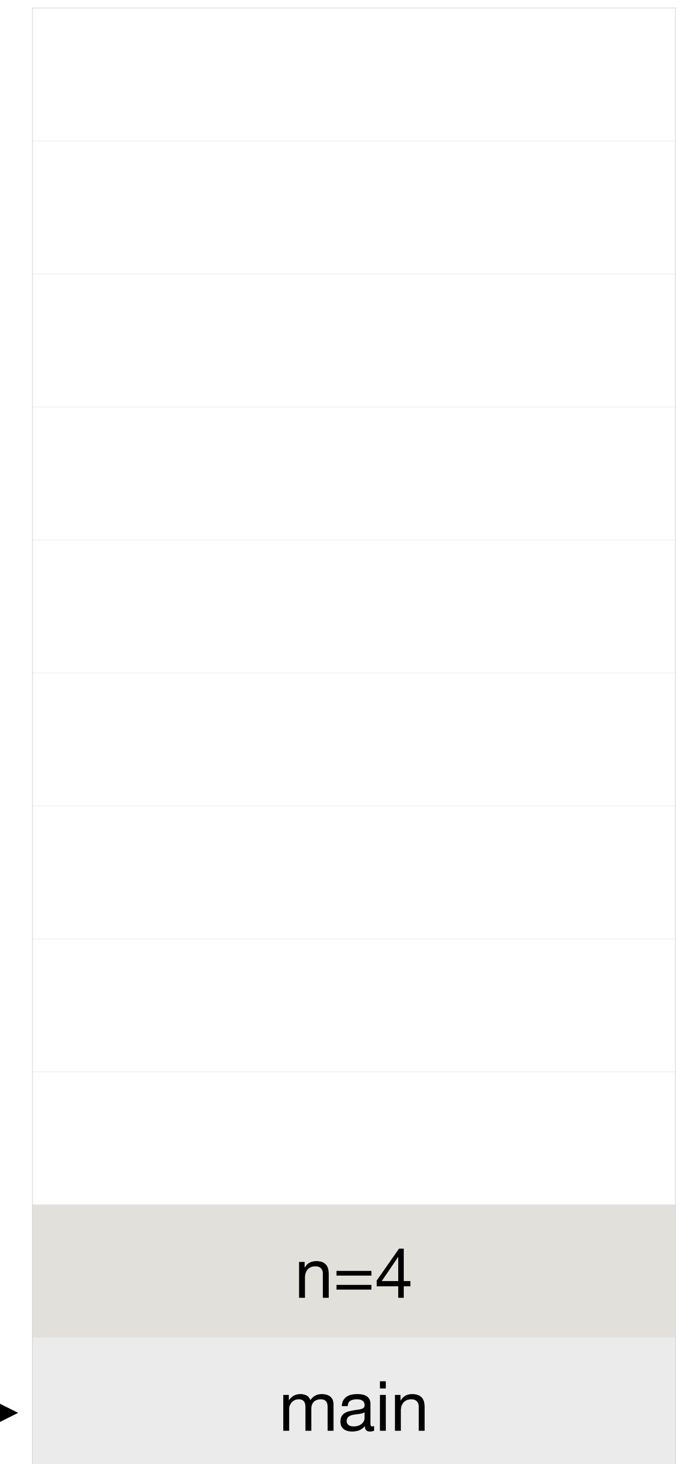


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

R0 | 4

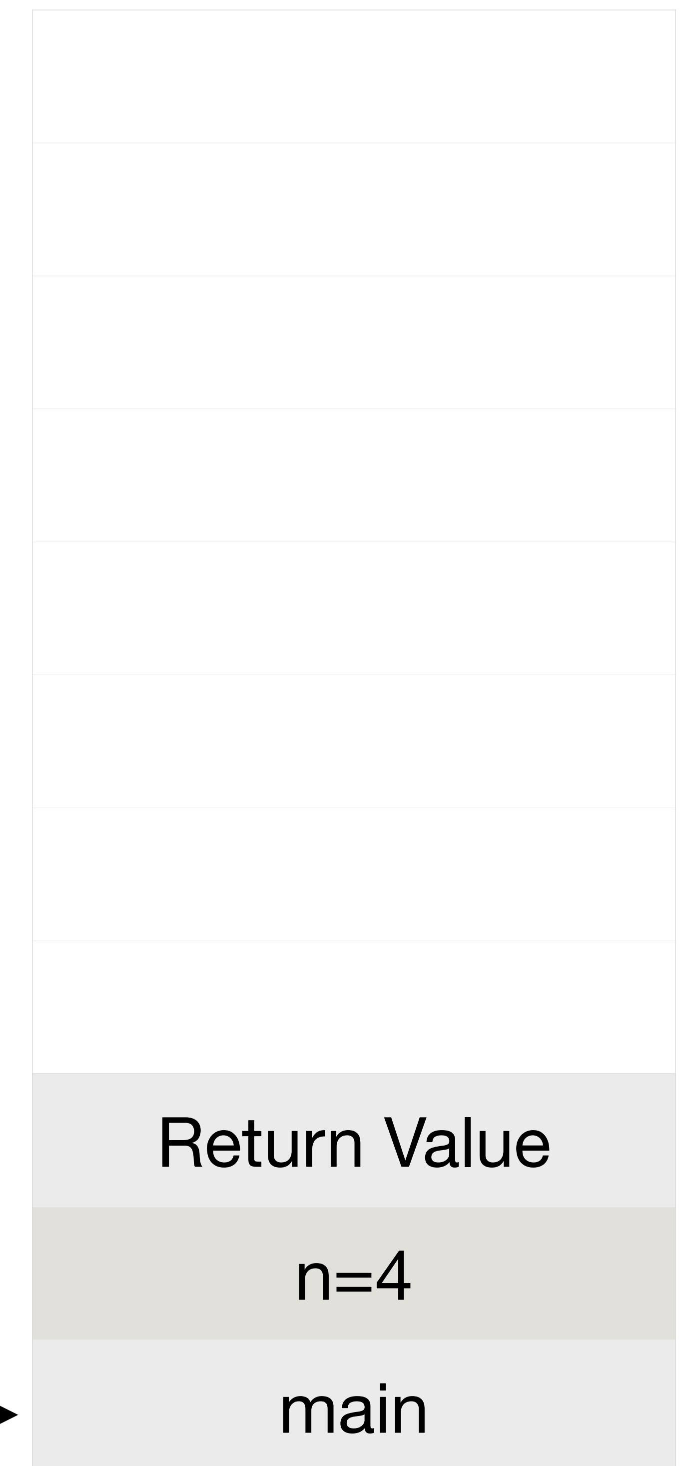
```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

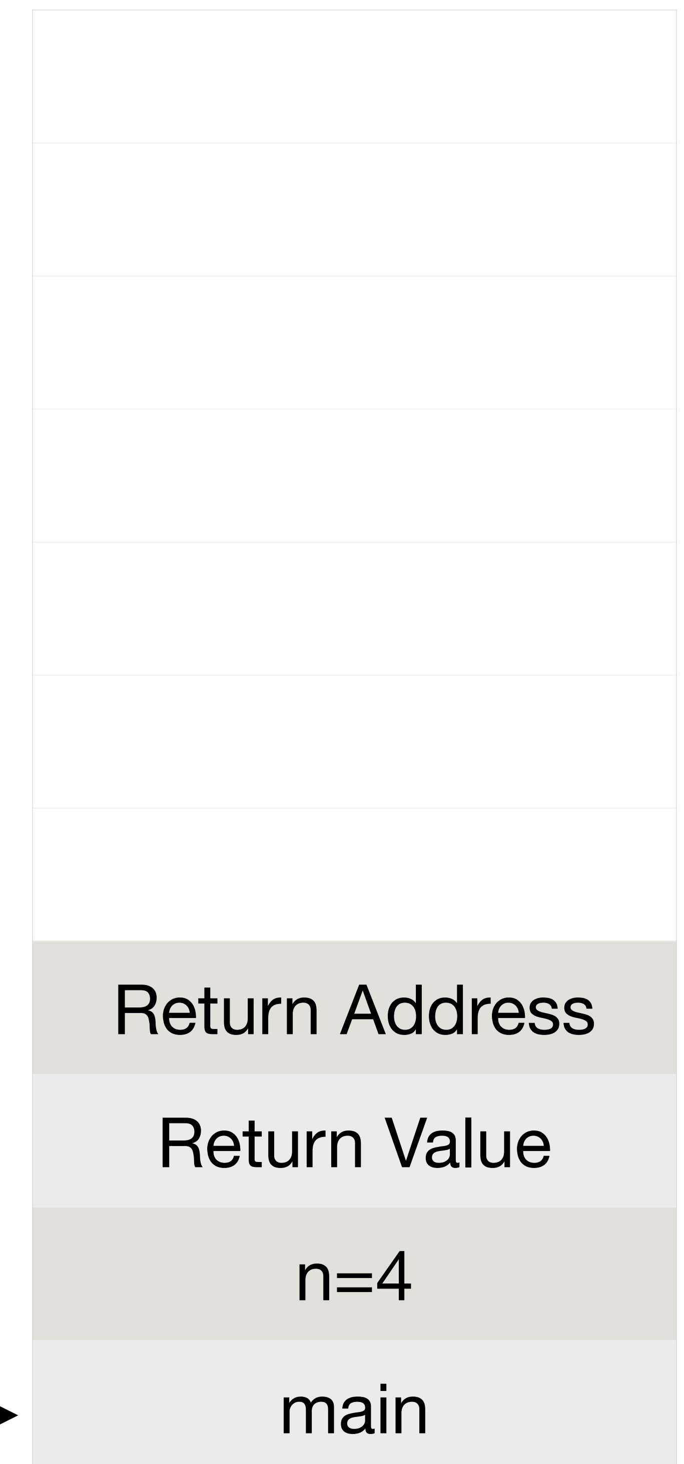
```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

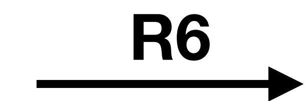
```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
```

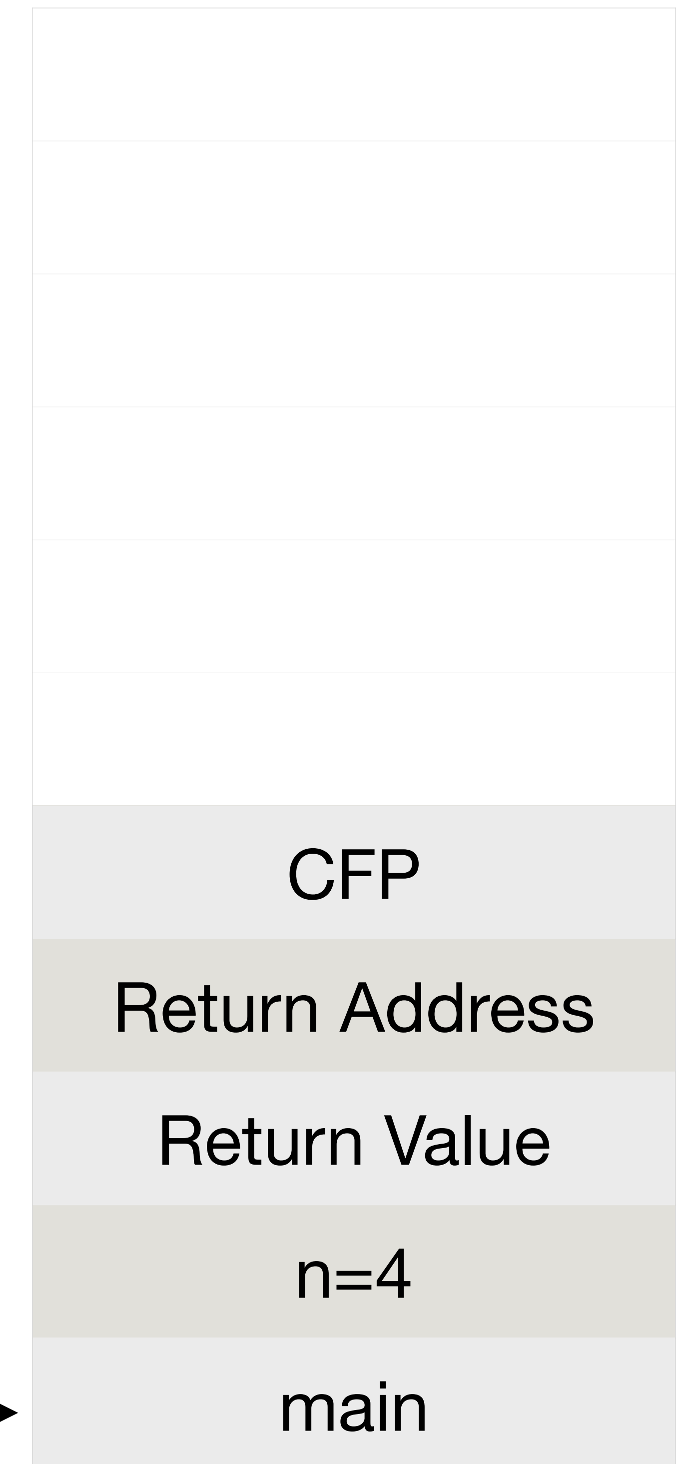


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0
```

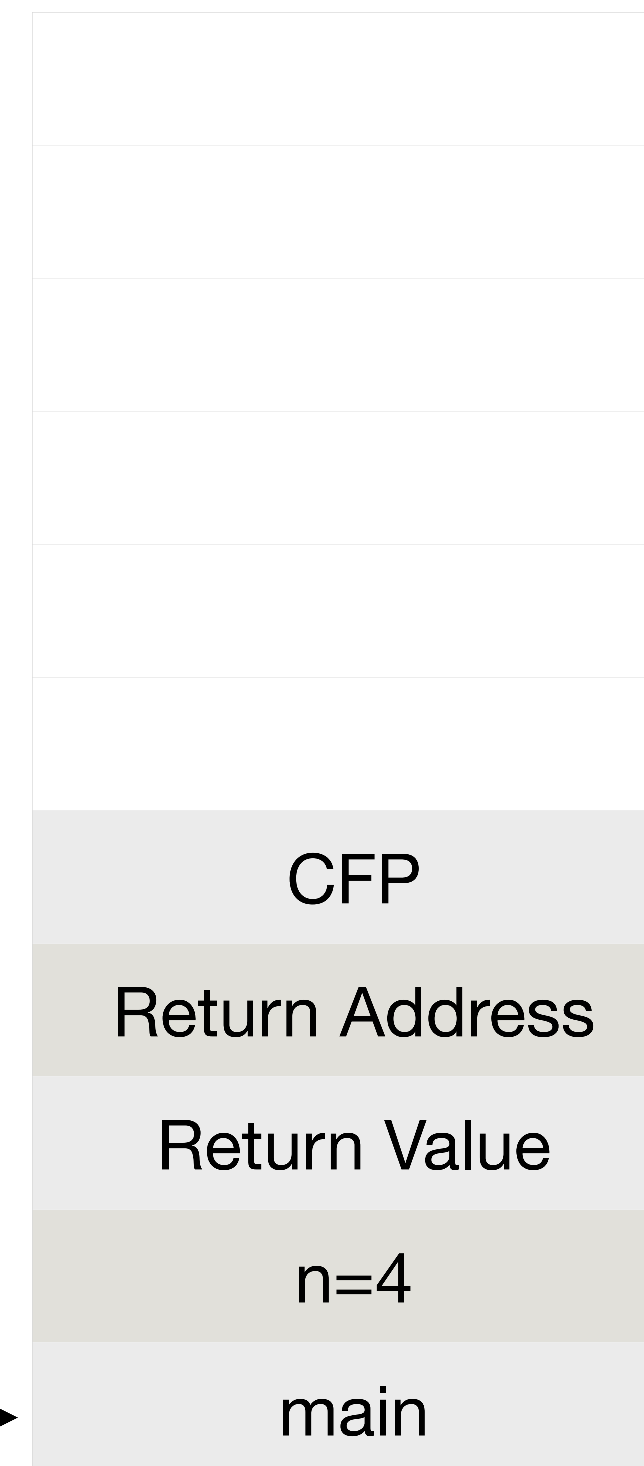




```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
```

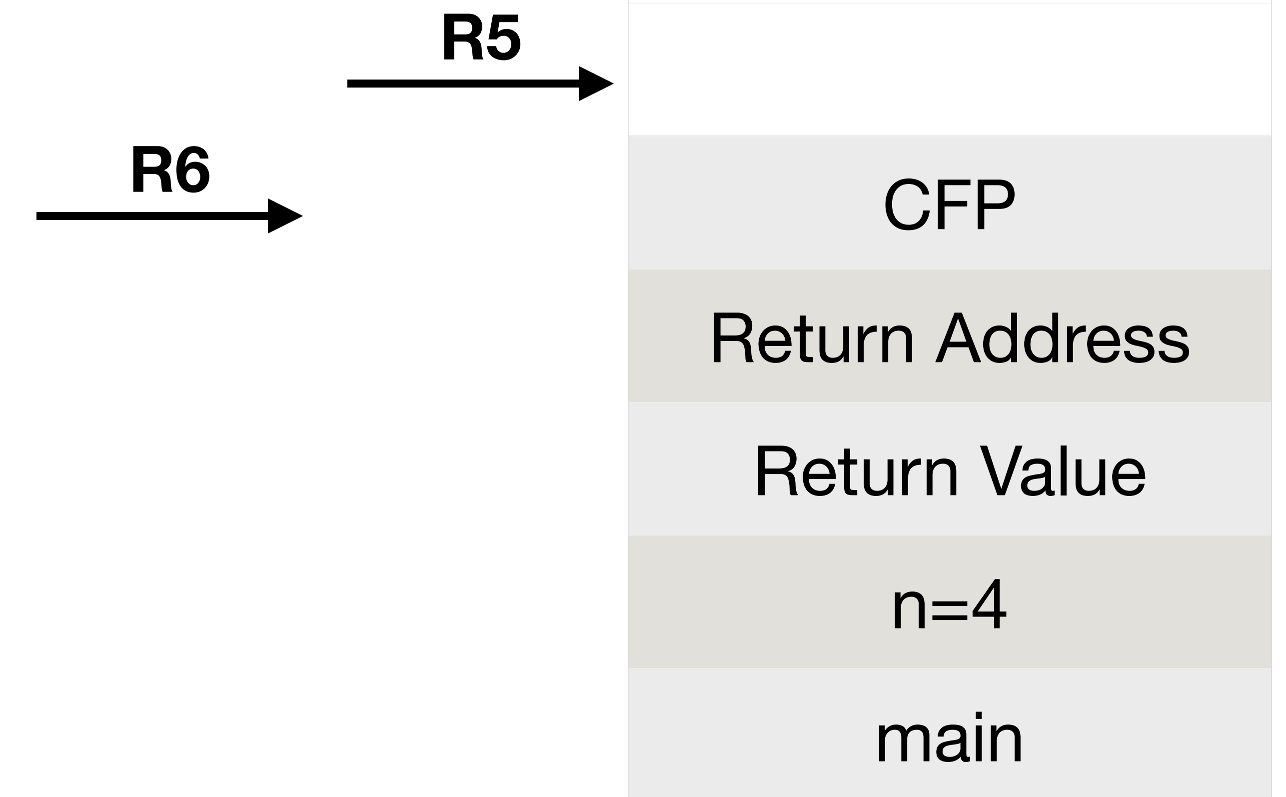


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

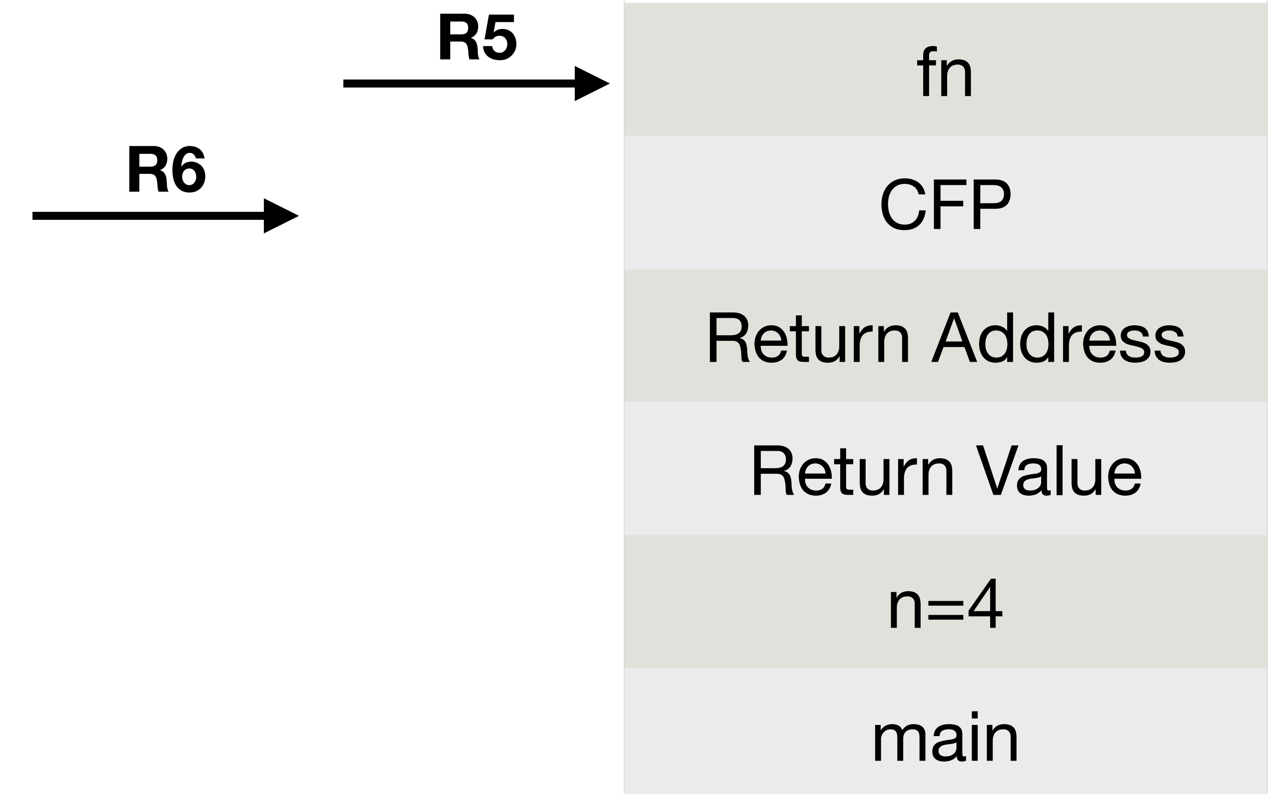
```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
```

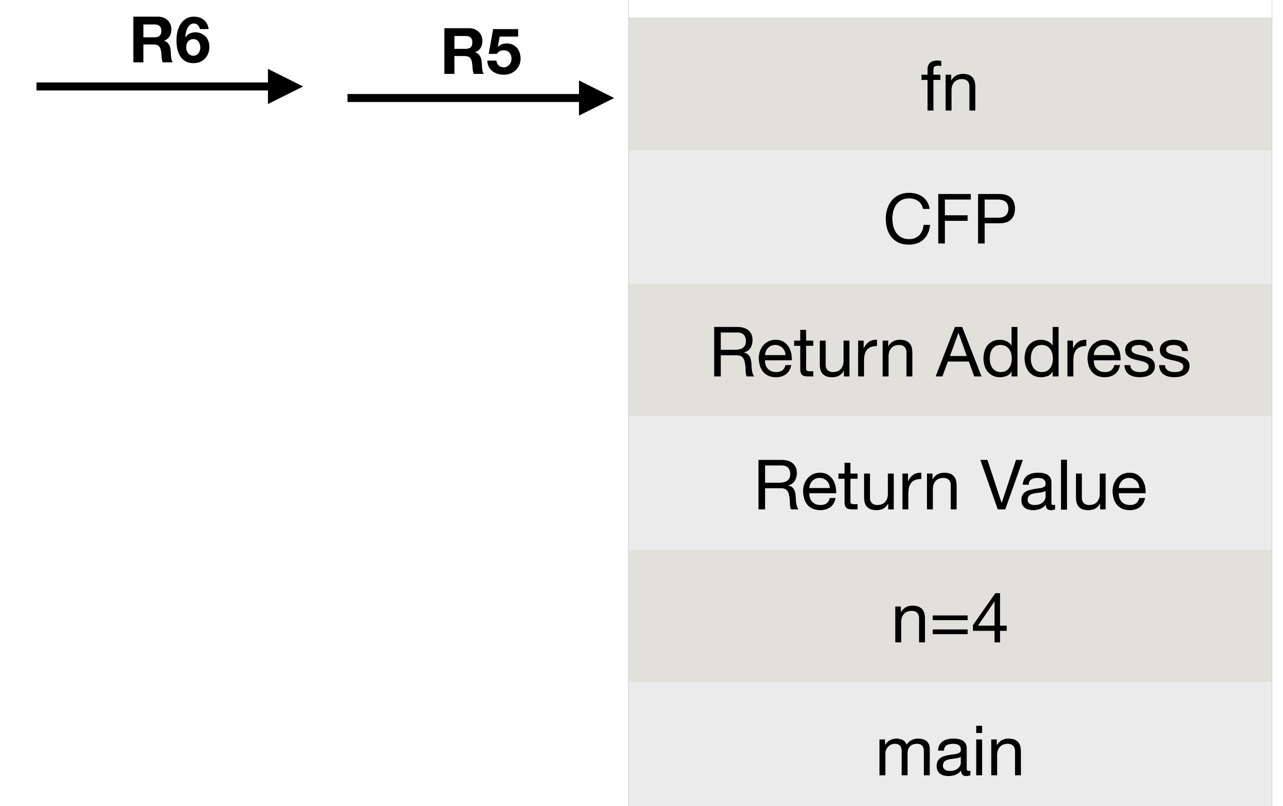


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
ADD R6, R6, #-1
```

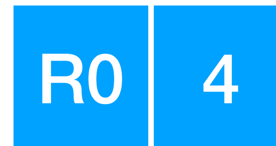
R0	4

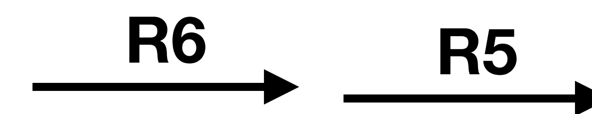


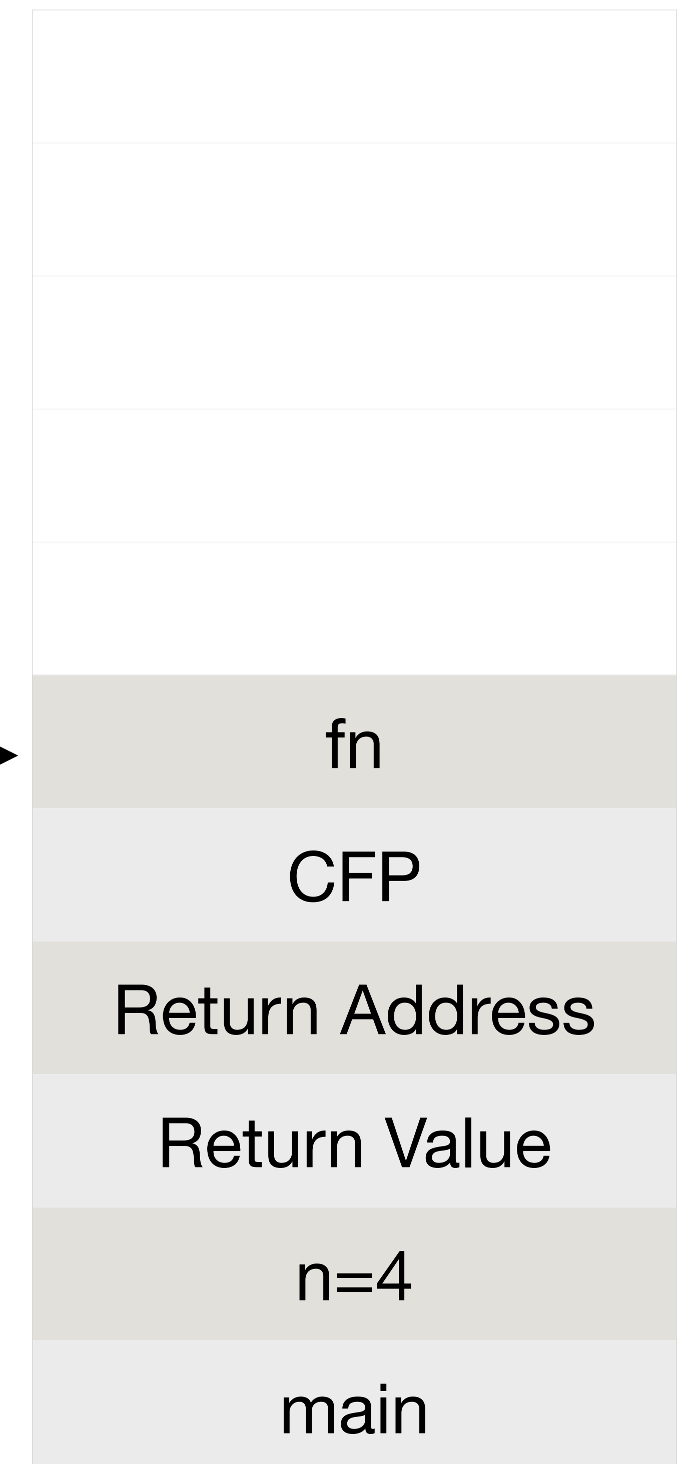
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
ADD R6, R6, #-1

;function logic
```



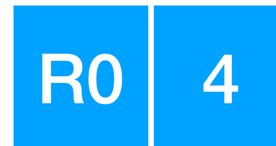


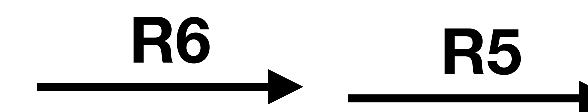


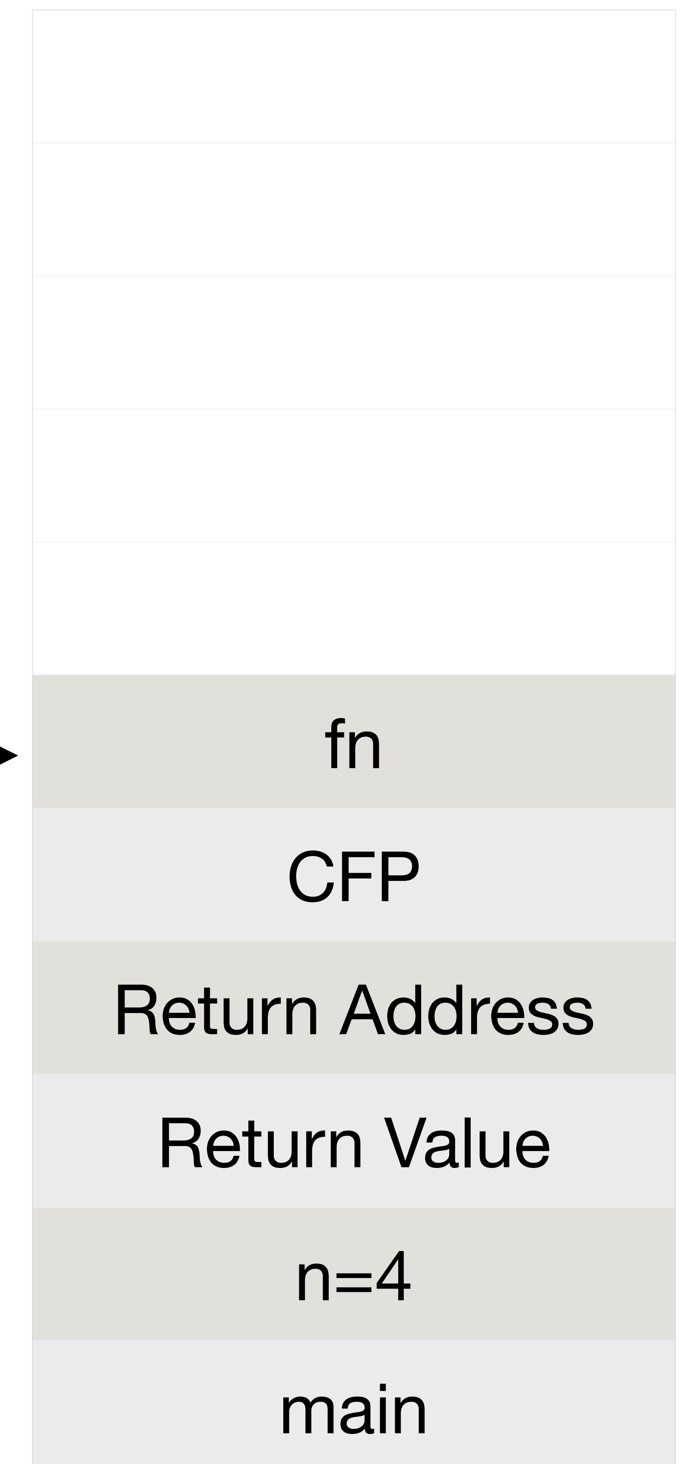
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
ADD R6, R6, #-1

;function logic
;base case (n==1)
```





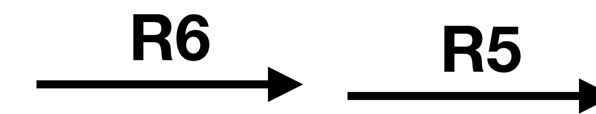


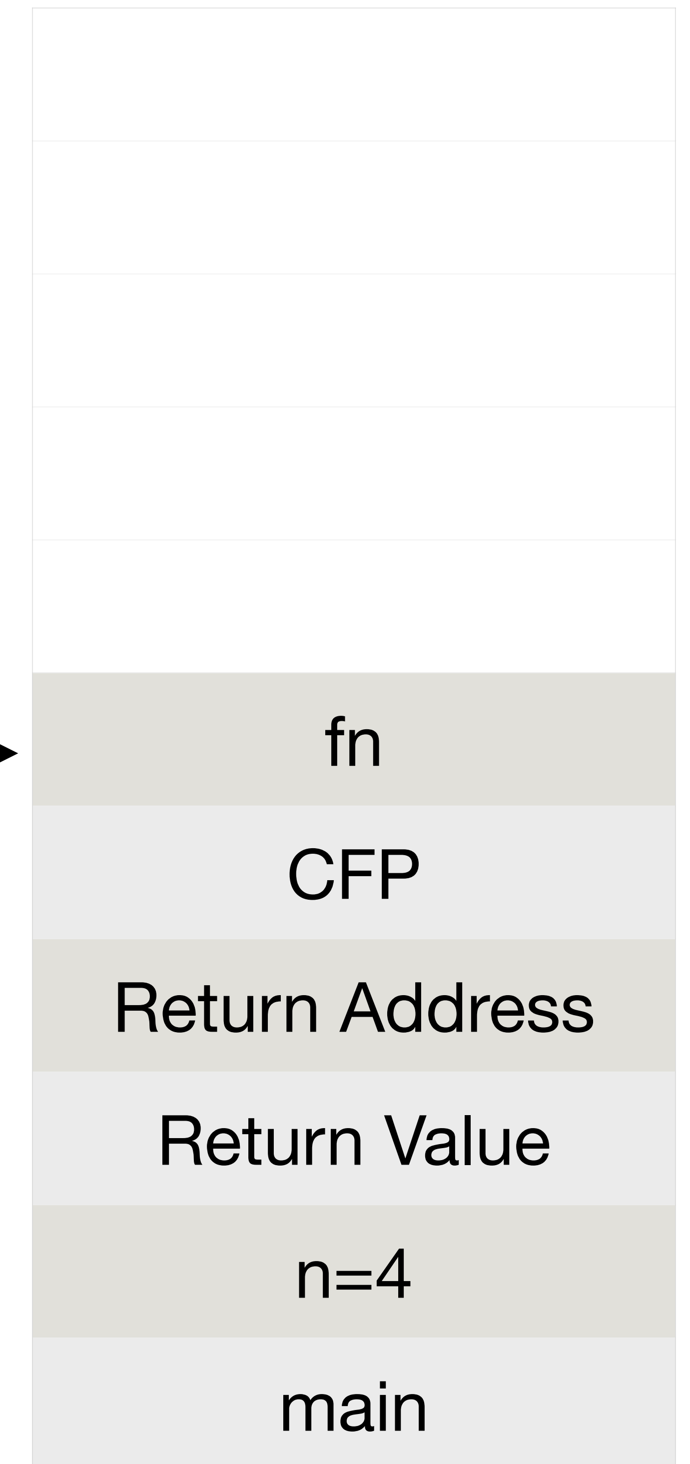
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
ADD R6, R6, #-1

;function logic
;base case (n==1)
LDR R1, R5, #4
```

R0	4
R1	4

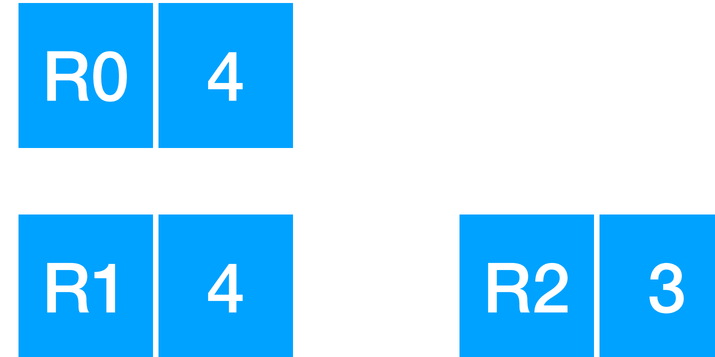


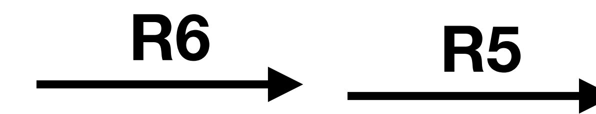


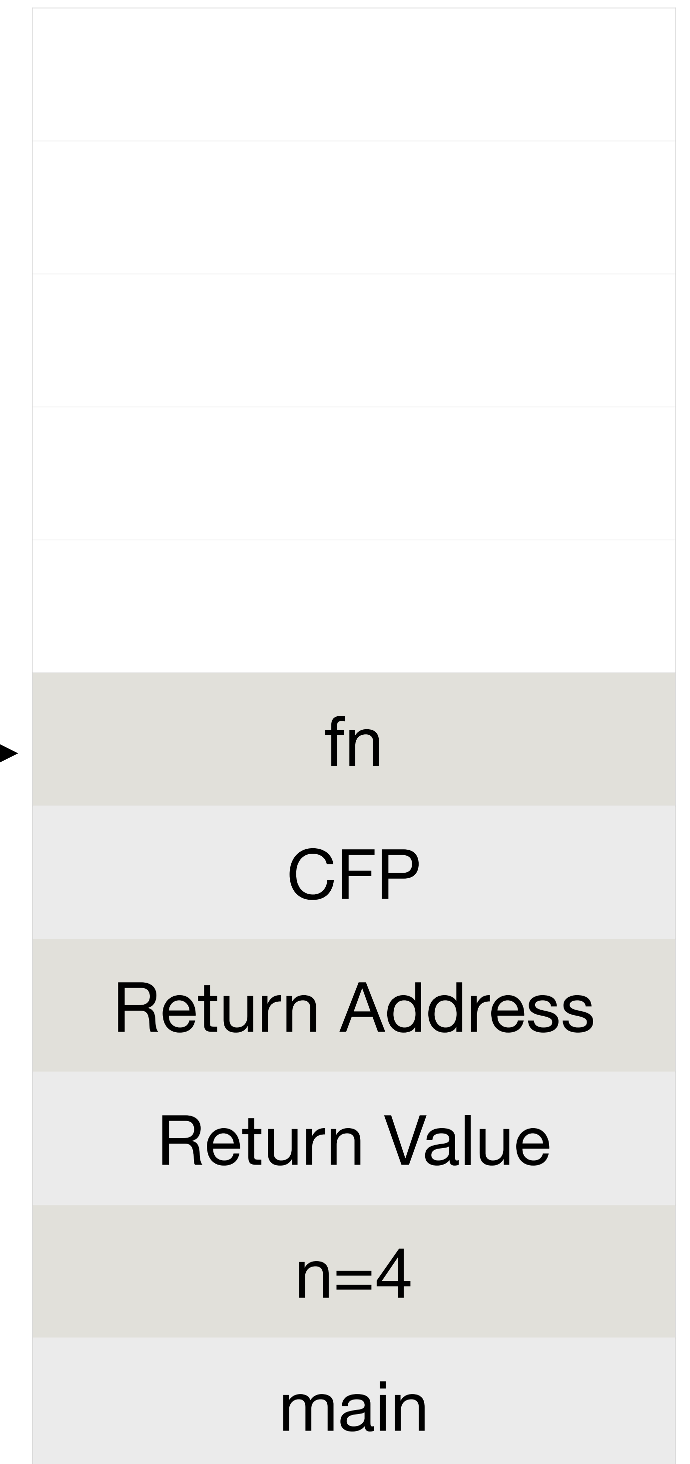
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
ADD R6, R6, #-1

;function logic
;base case (n==1)
LDR R1, R5, #4
ADD R2, R1, #-1
```



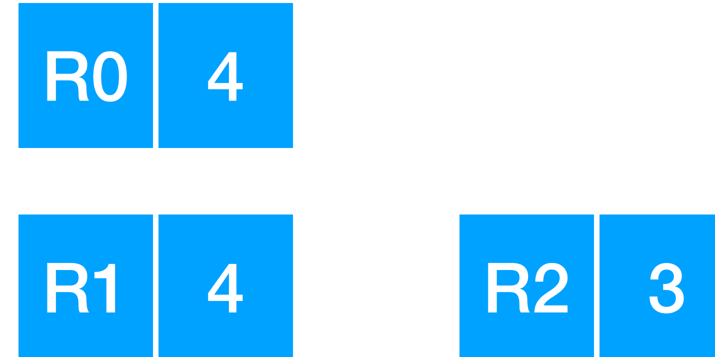




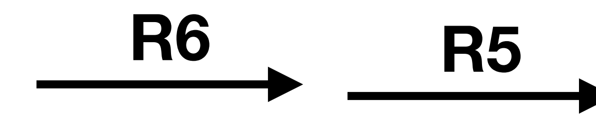
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

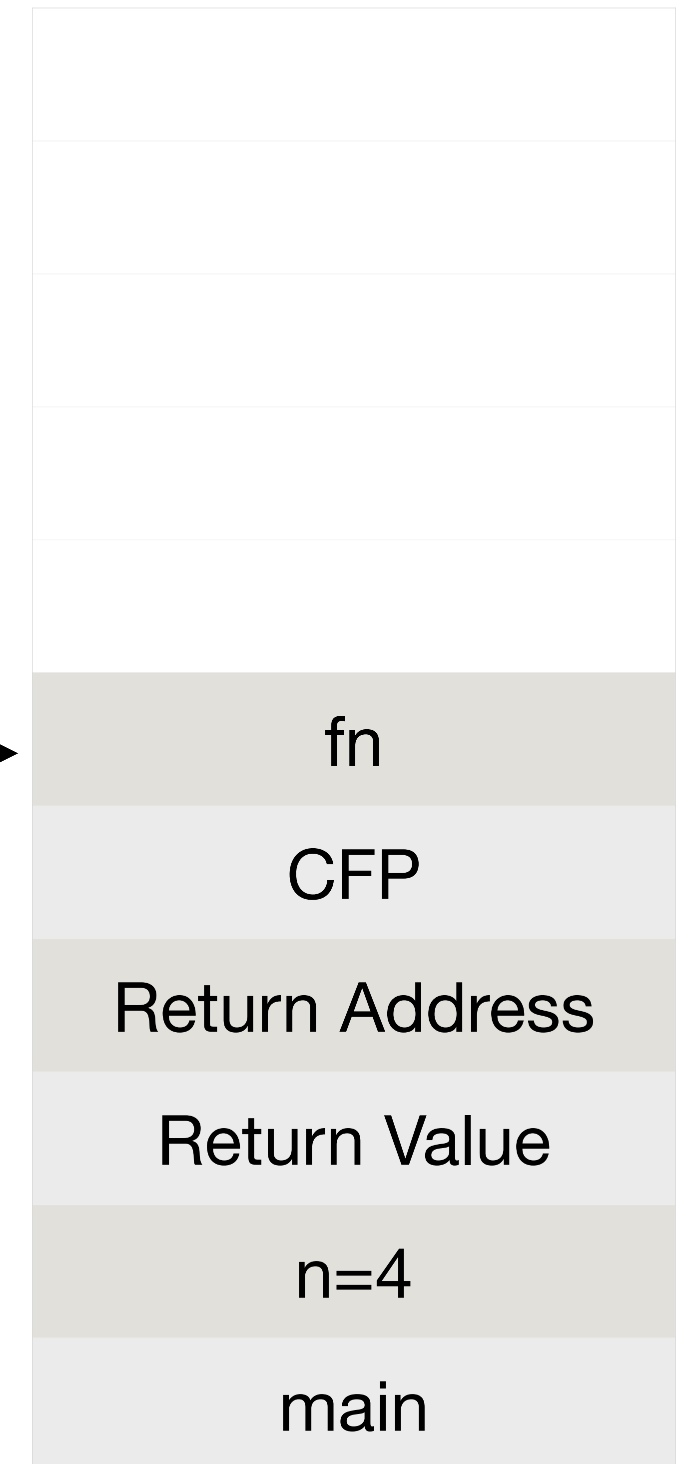
```
;Caller set-up
ADD R0, R0, #4
ADD R6, R6, #-1
STR R0, R6, #0

RUNNING
;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer
ADD R6, R6, #-3
STR R7, R6, #1
STR R5, R6, #0

;push local variables & update frame pointer
ADD R5, R6, #-1
ADD R6, R6, #-1

;function logic
;base case (n==1)
LDR R1, R5, #4
ADD R2, R1, #-1
BRz BASE_CASE
```



R2 3

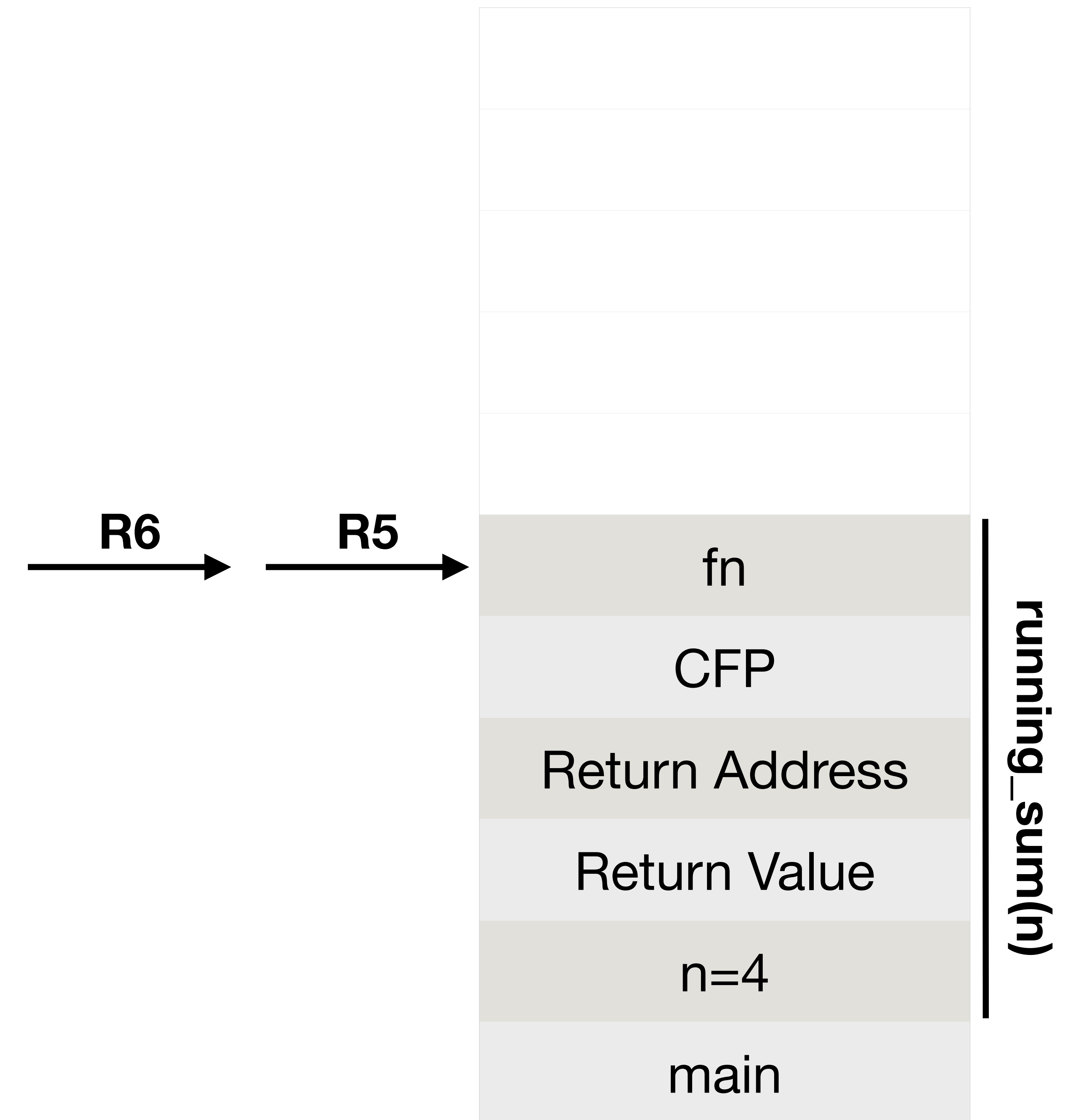


R5



```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

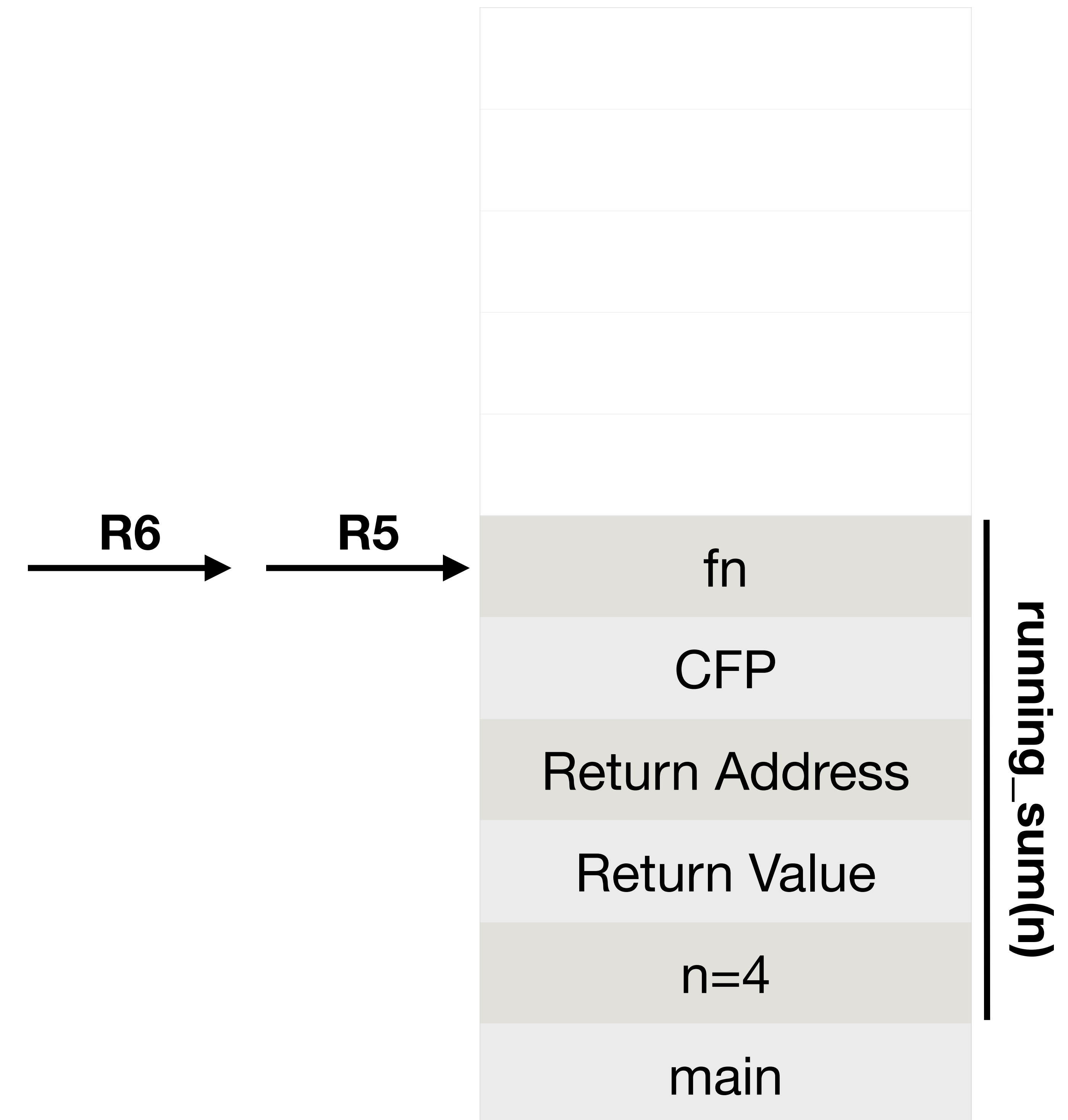
Recursion in LC3



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

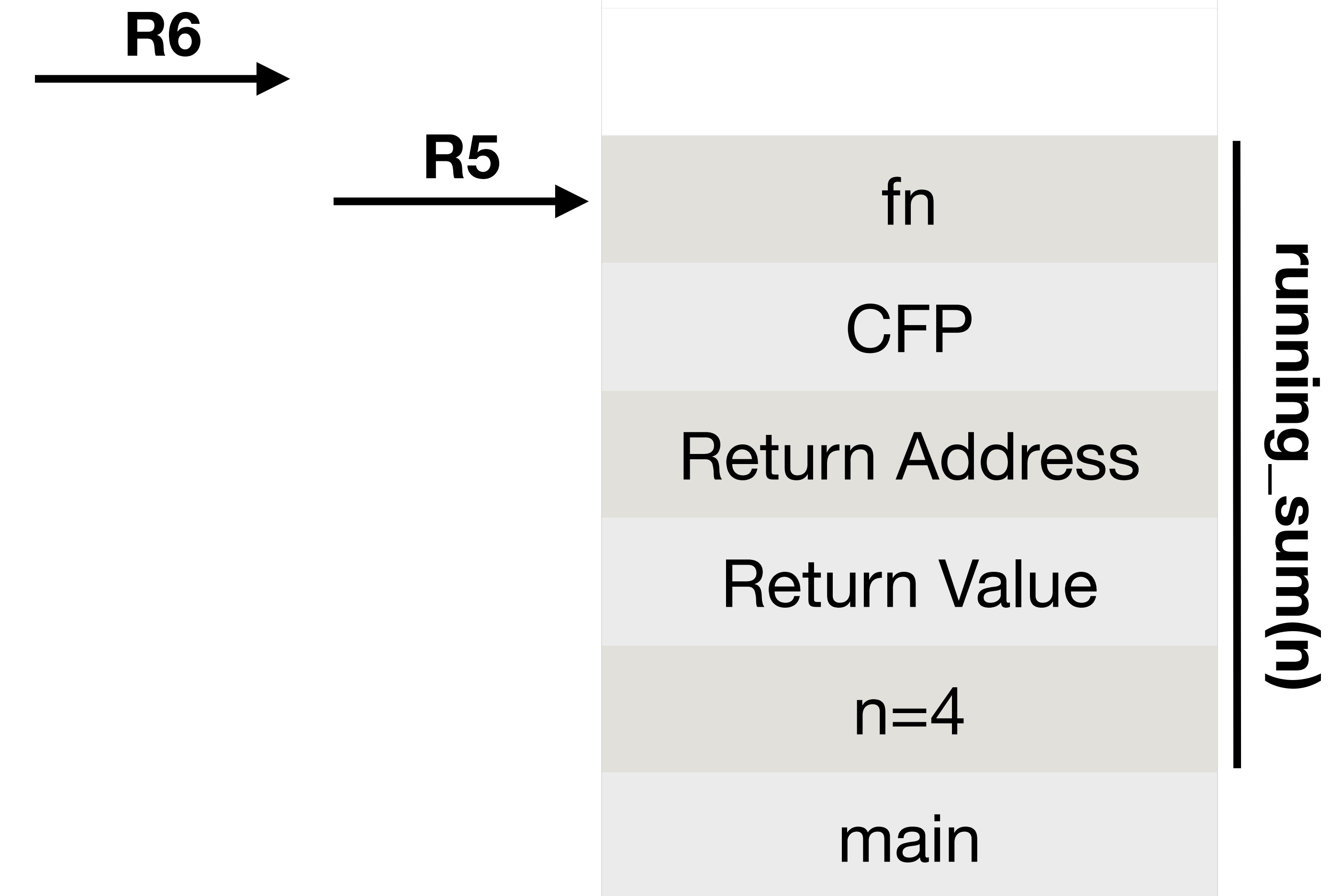
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

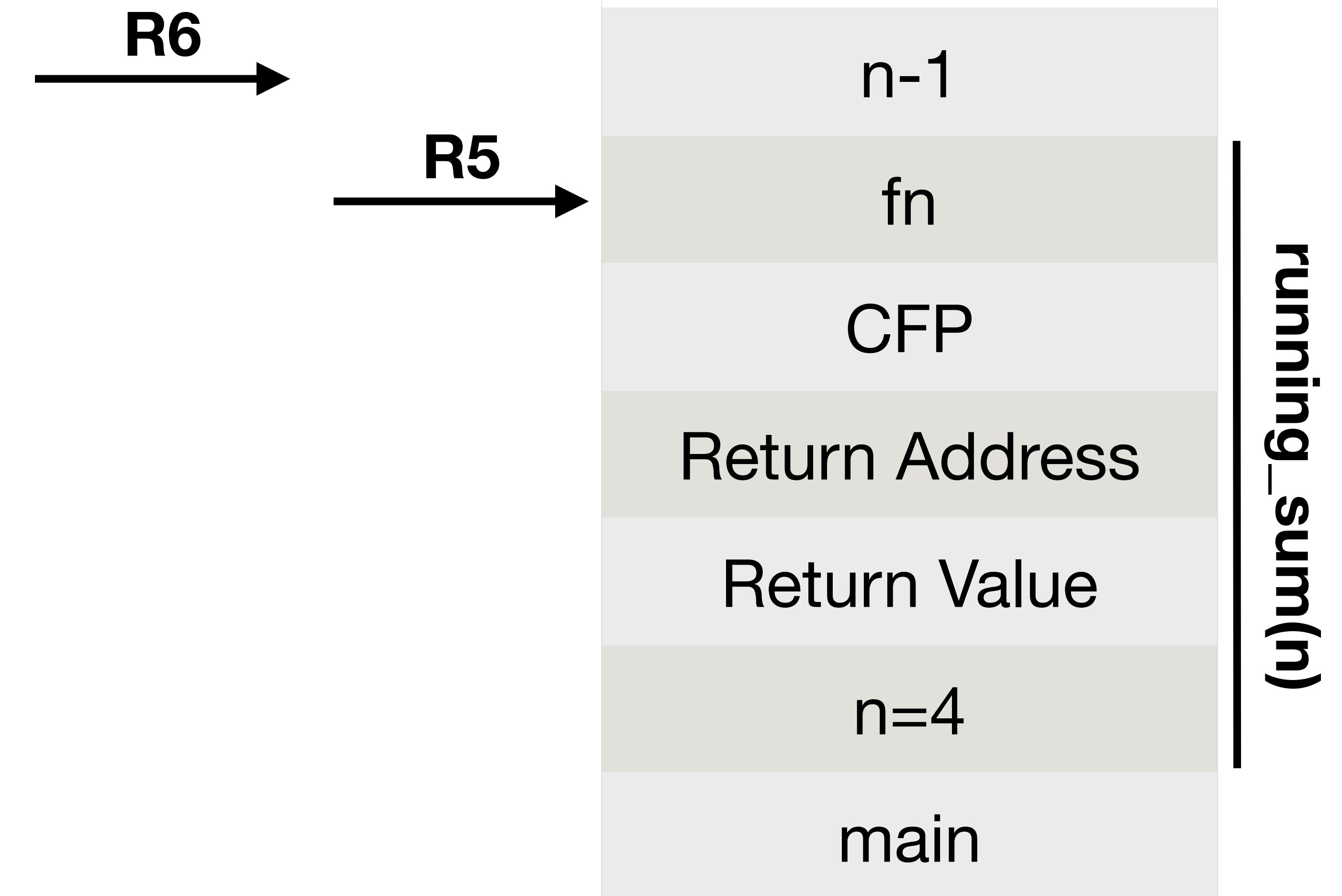
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

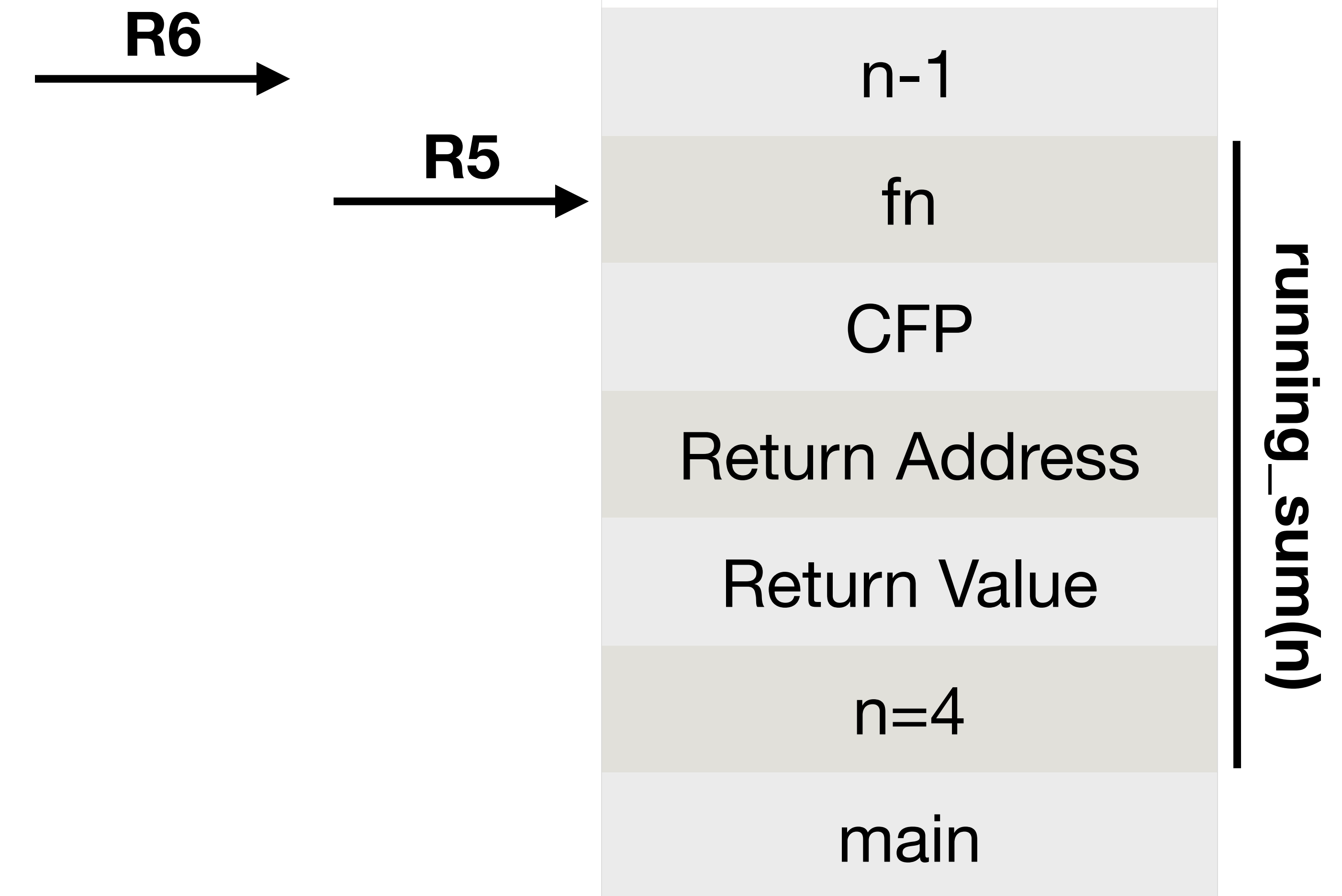
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

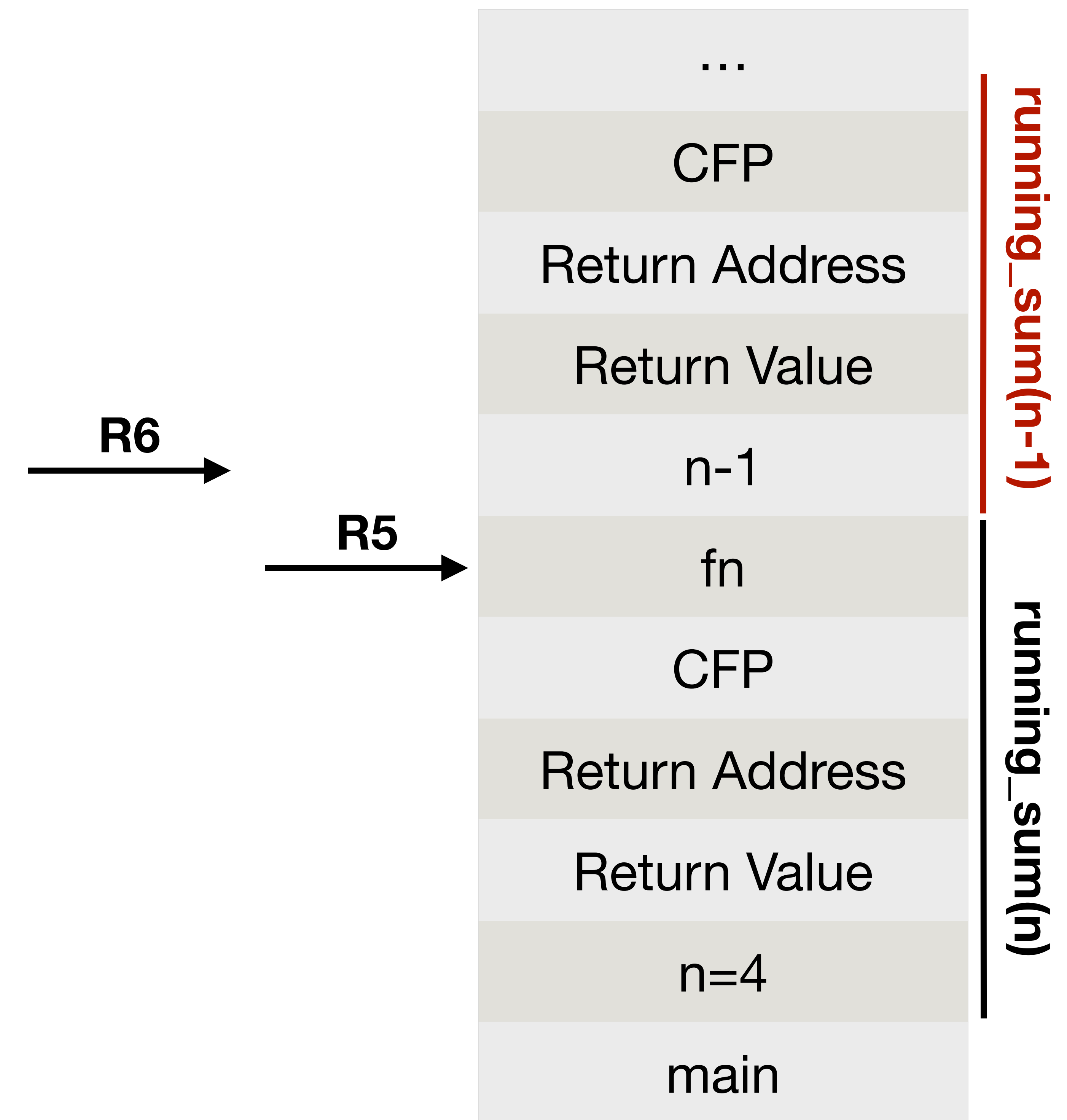
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

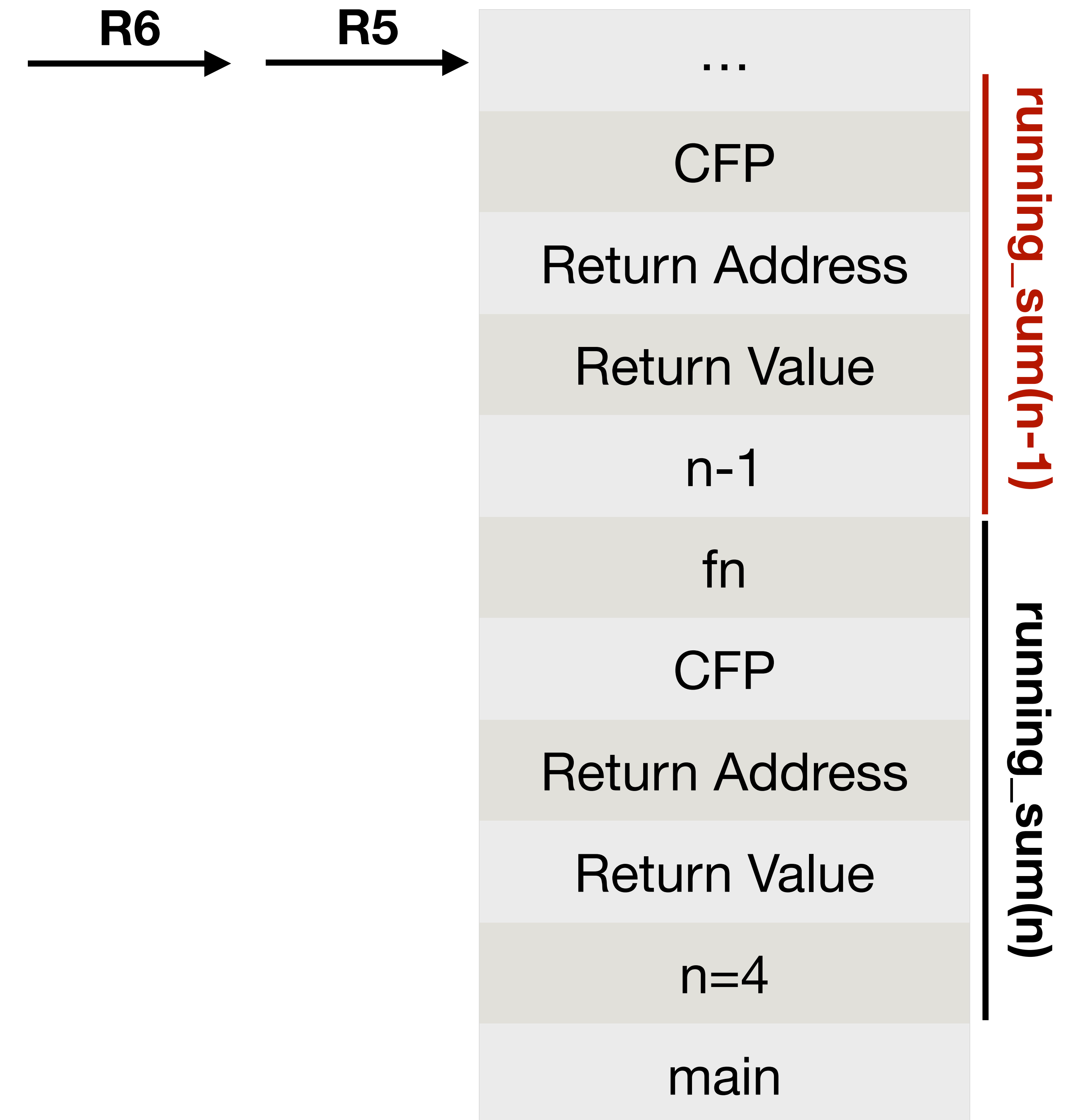
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

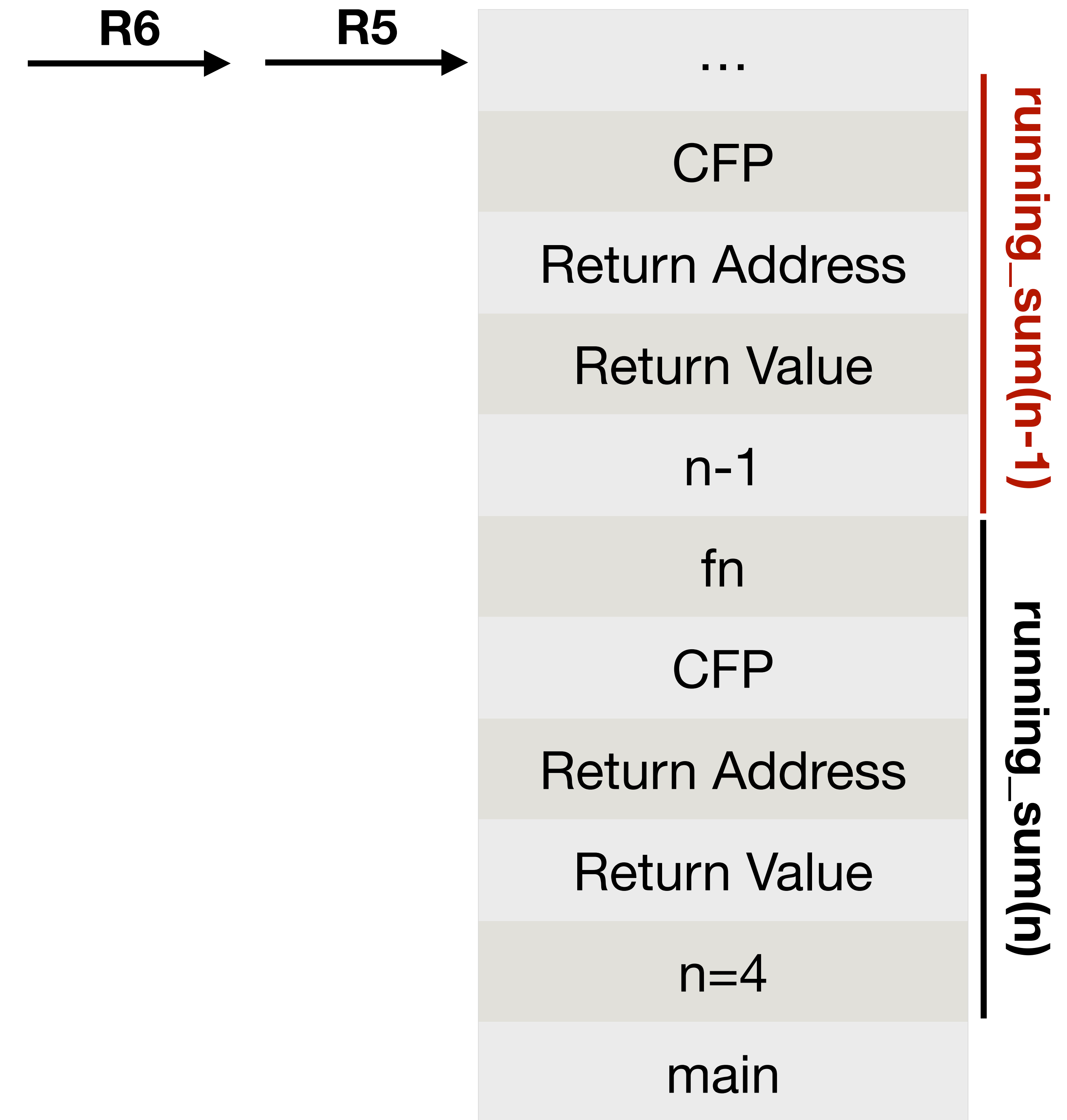
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

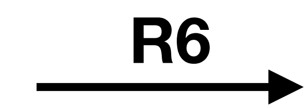
```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown
```

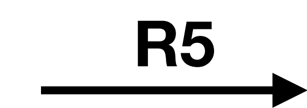


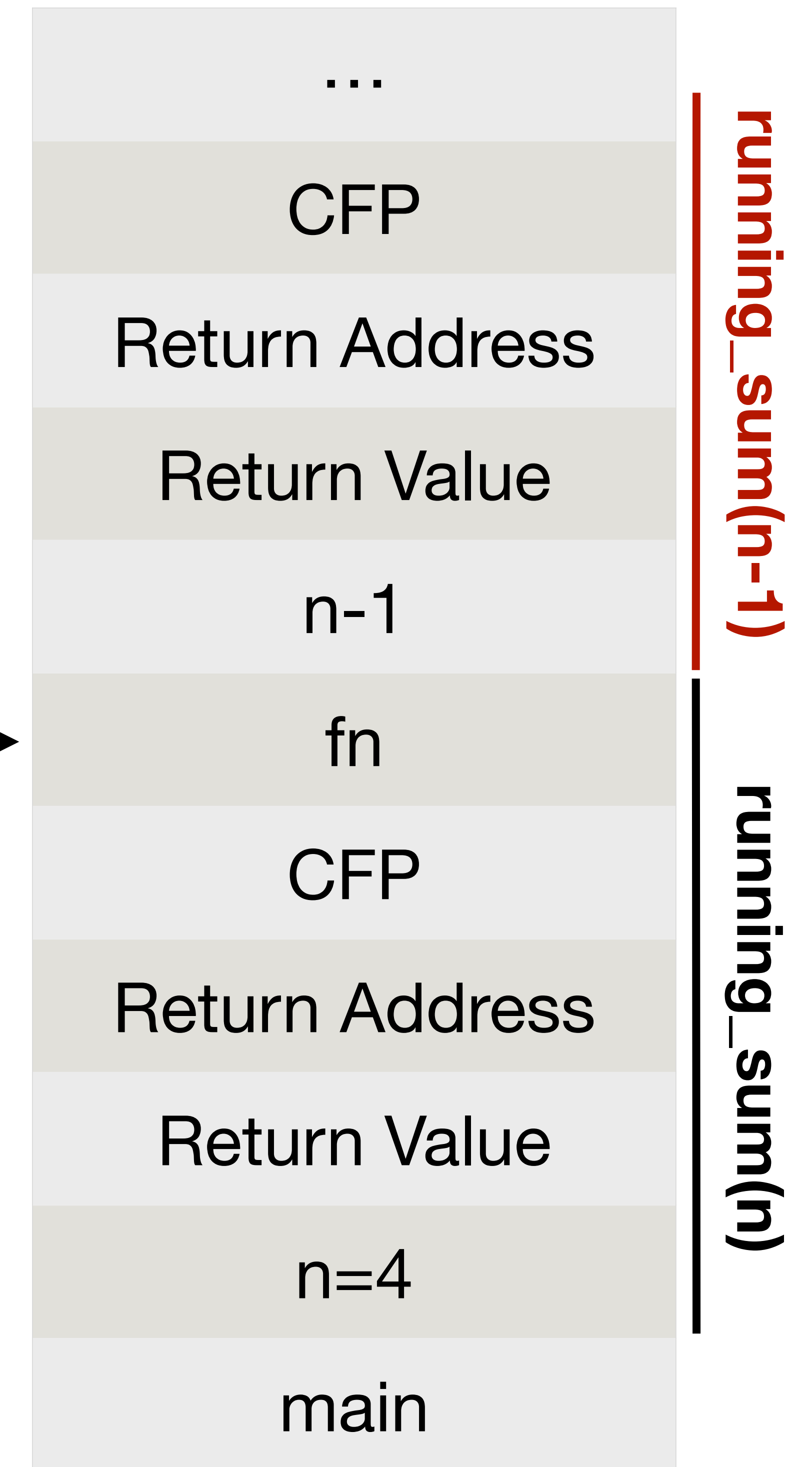
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown
```





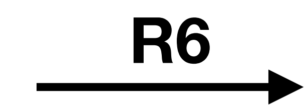


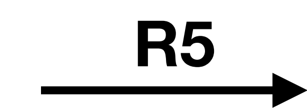
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

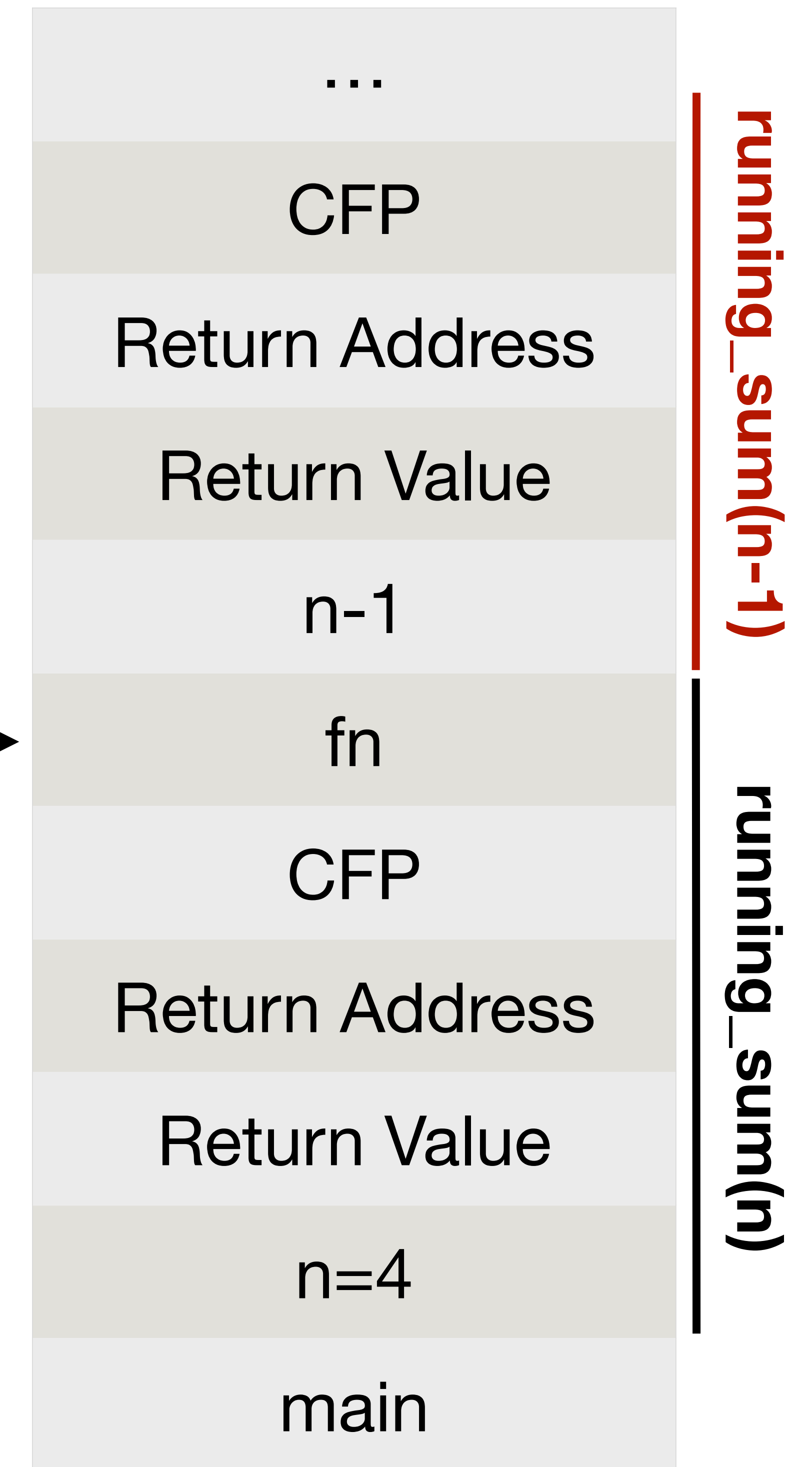
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
```





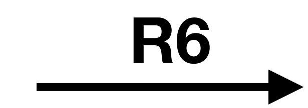


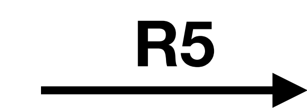
ILLINOIS

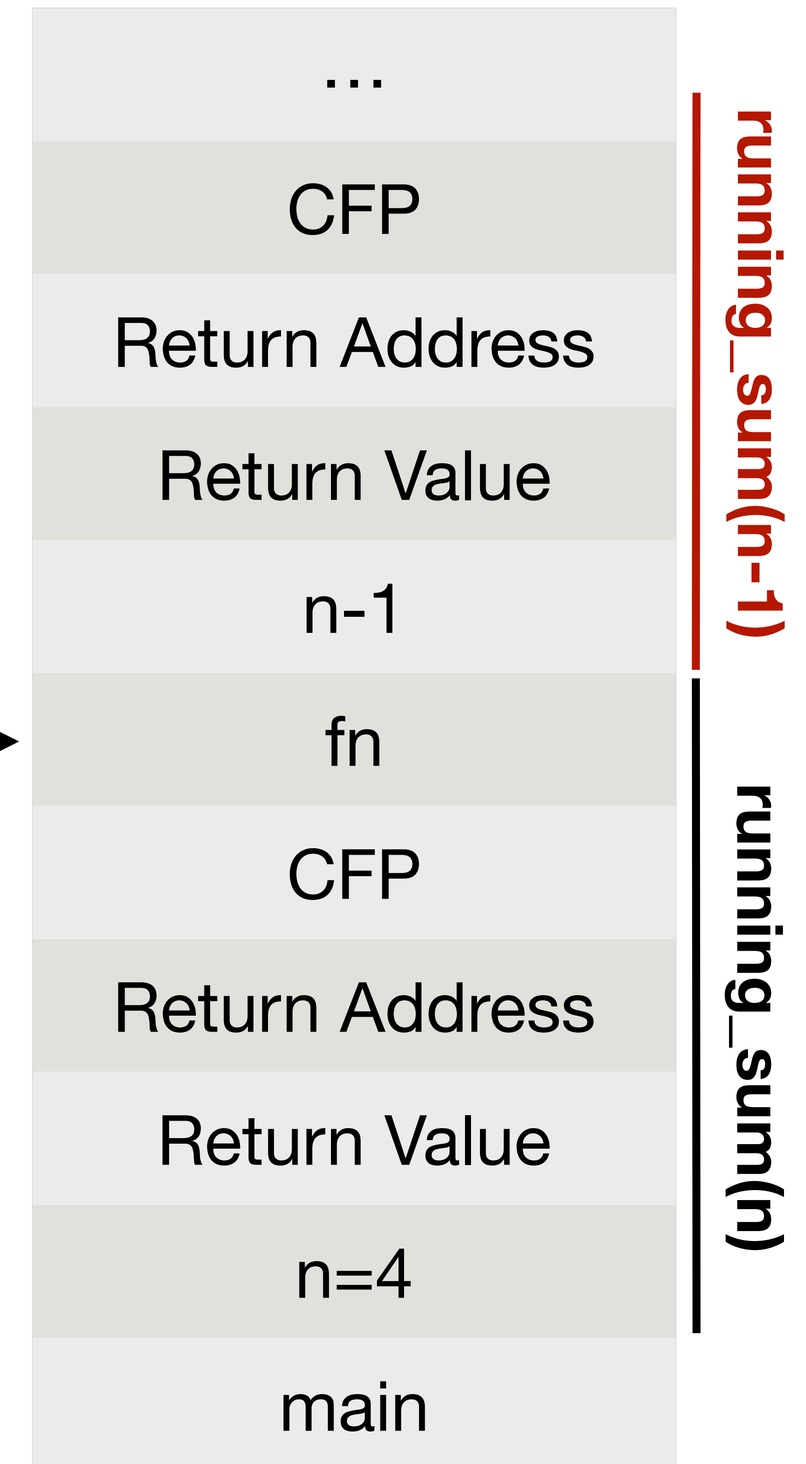
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
```



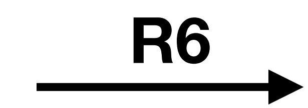


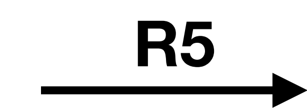


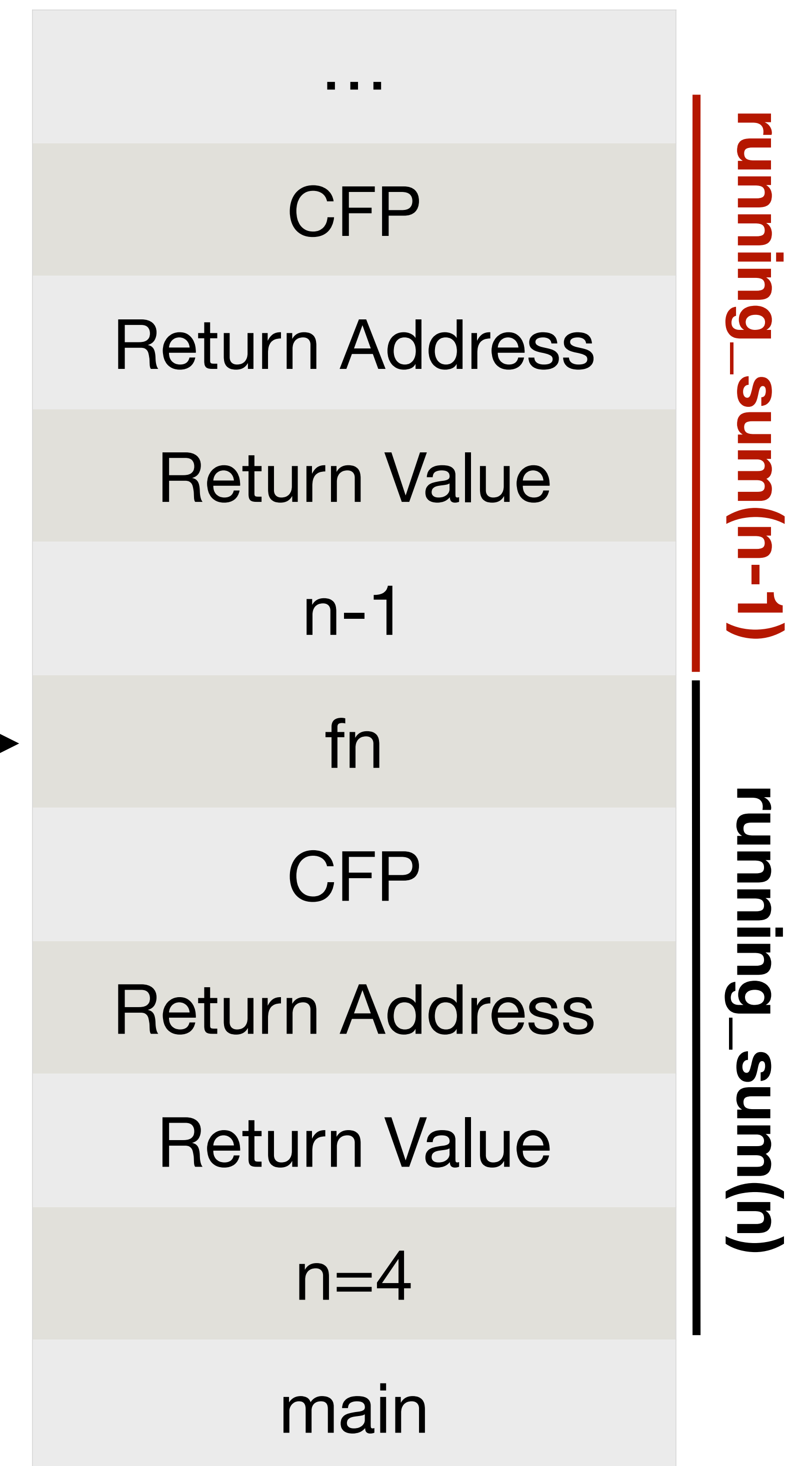
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
```



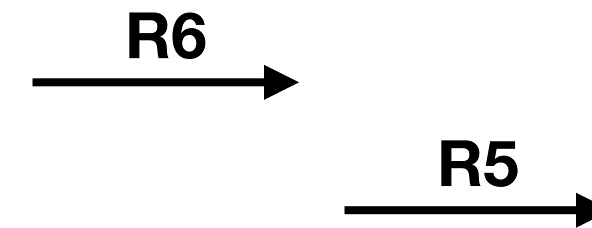


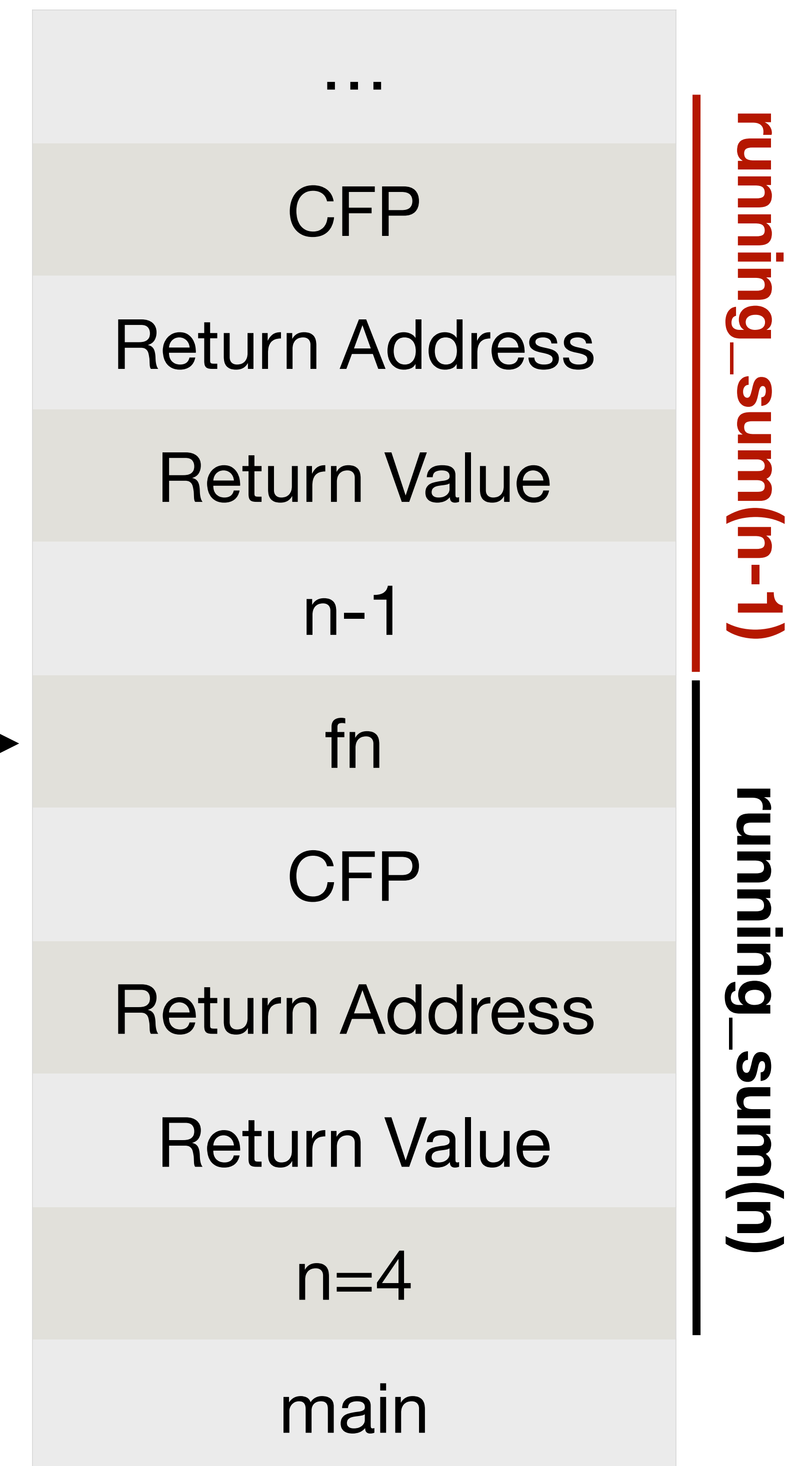


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1
```

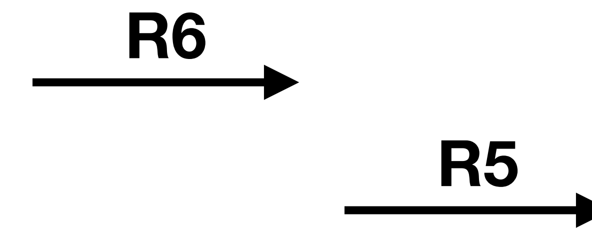


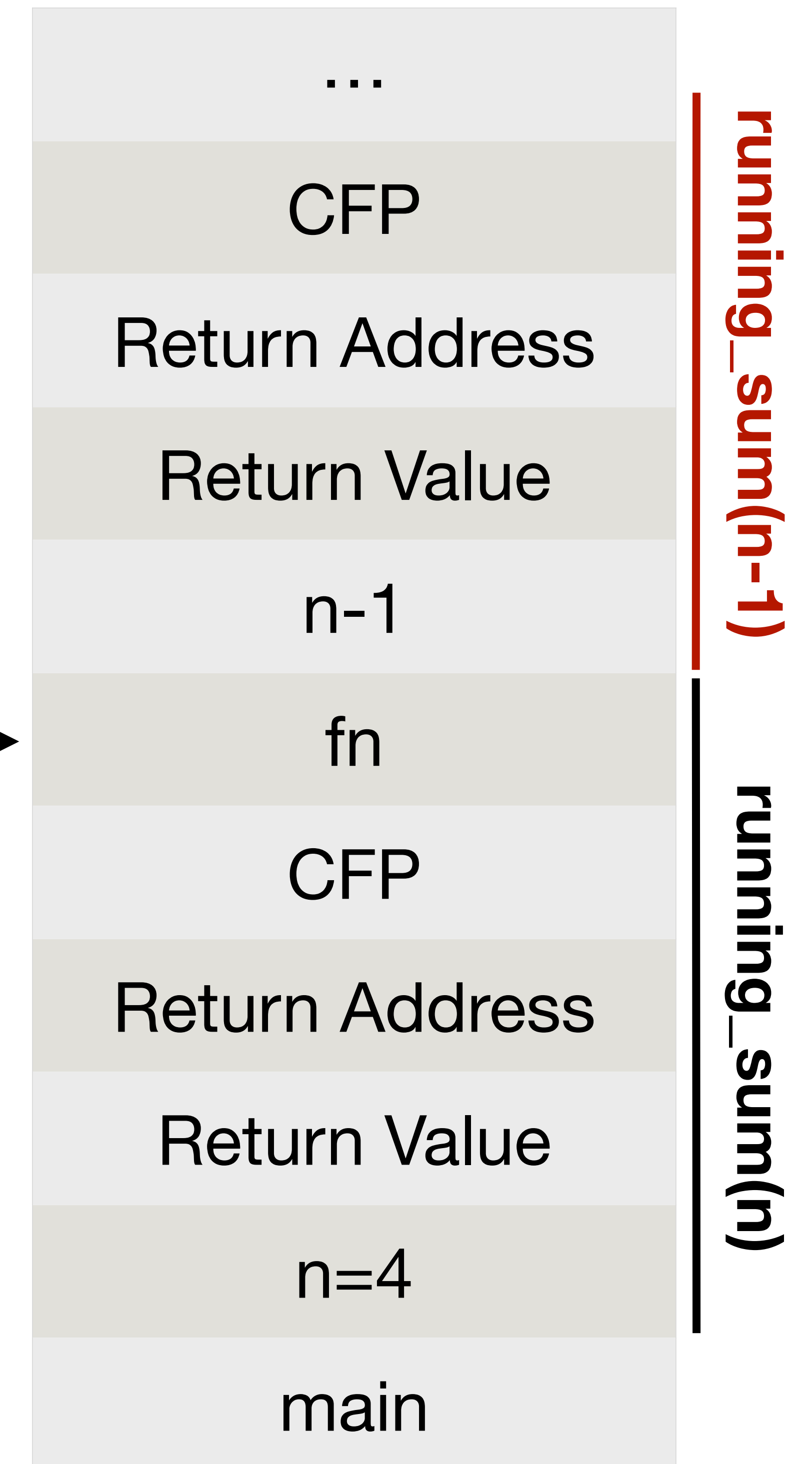


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
```

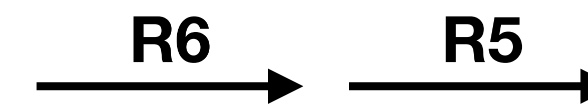


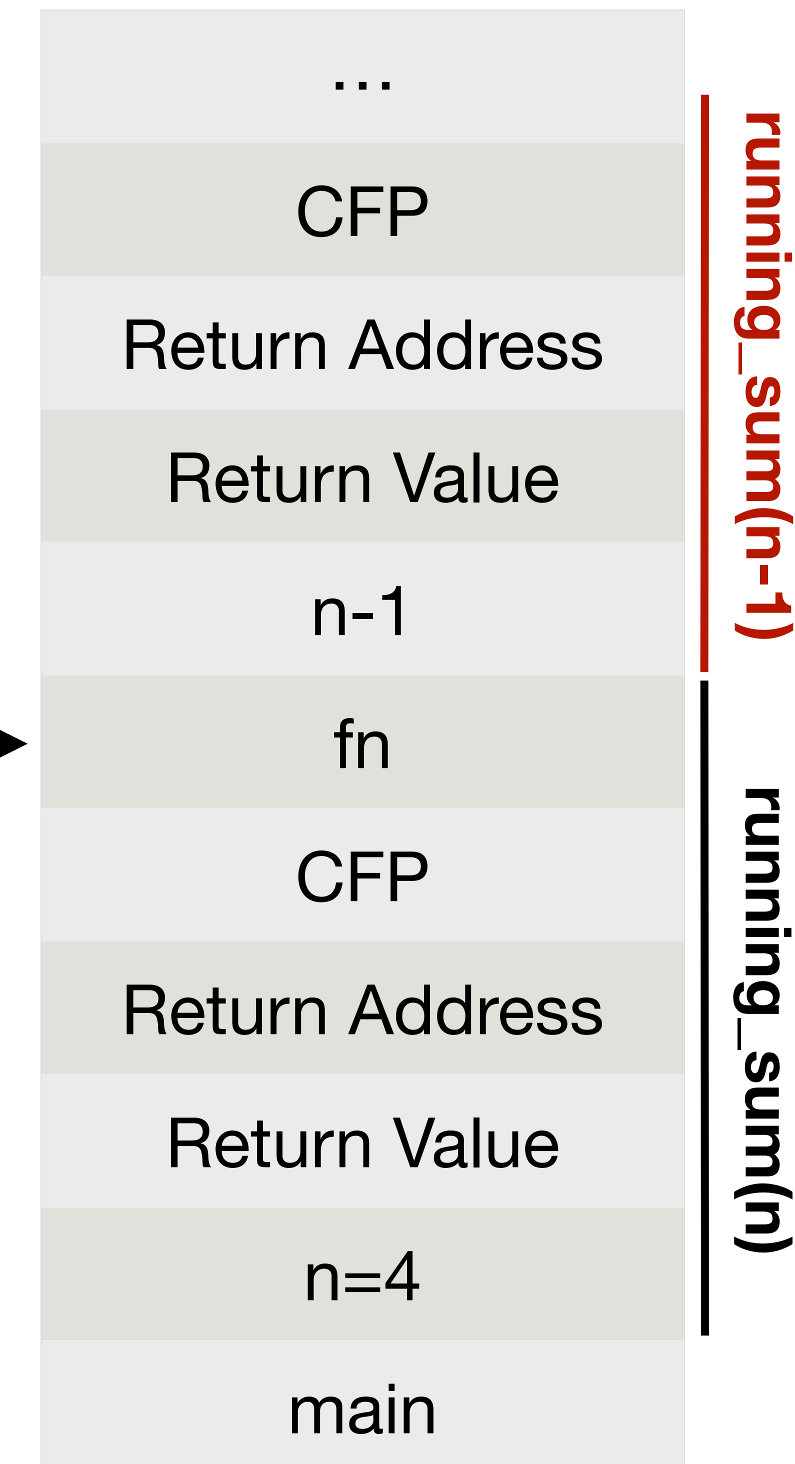


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1
```

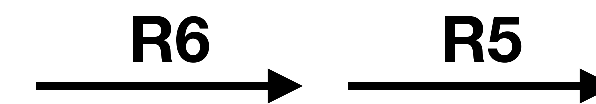


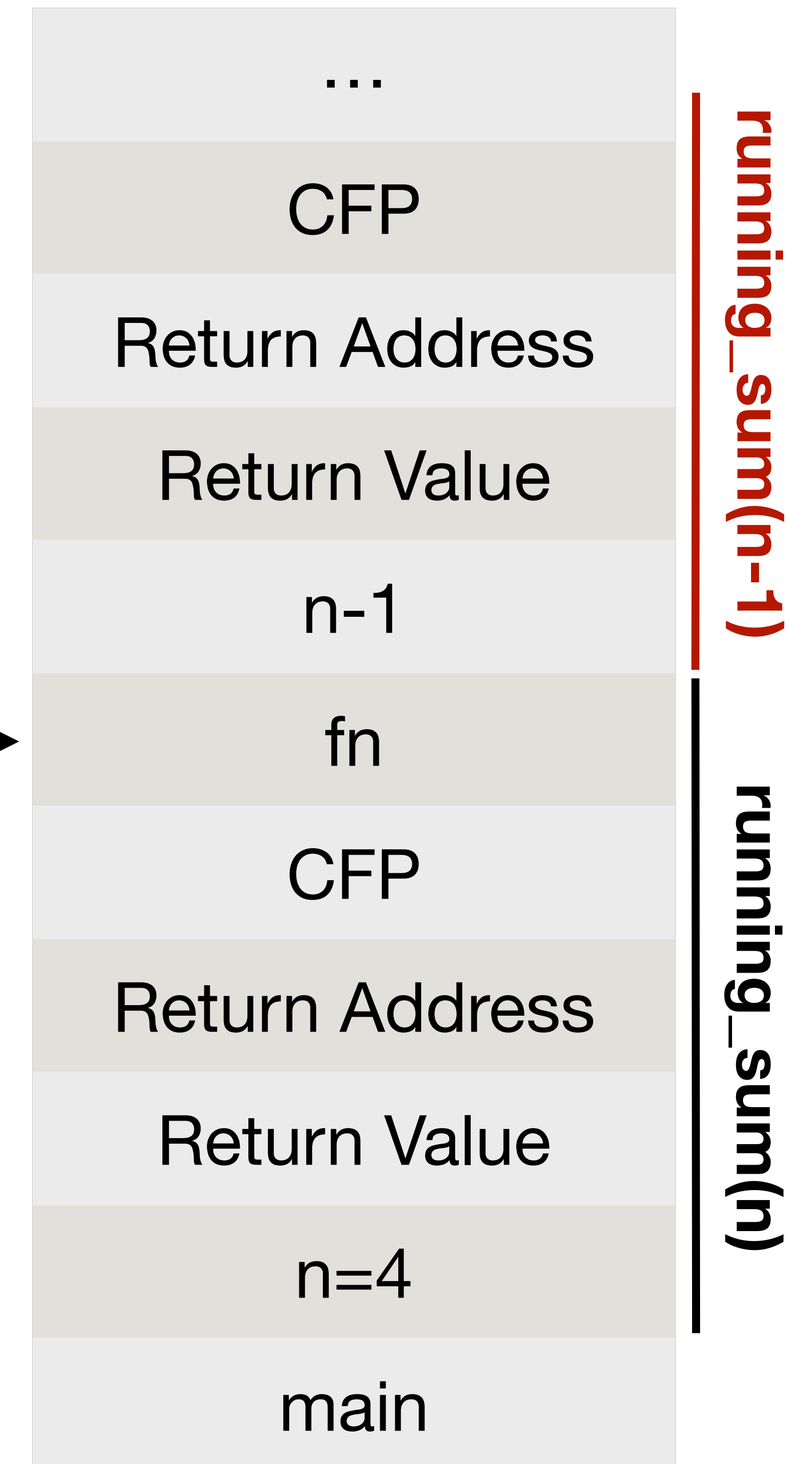


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1

;calculate n + Running(n-1)
```





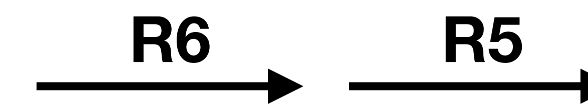
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

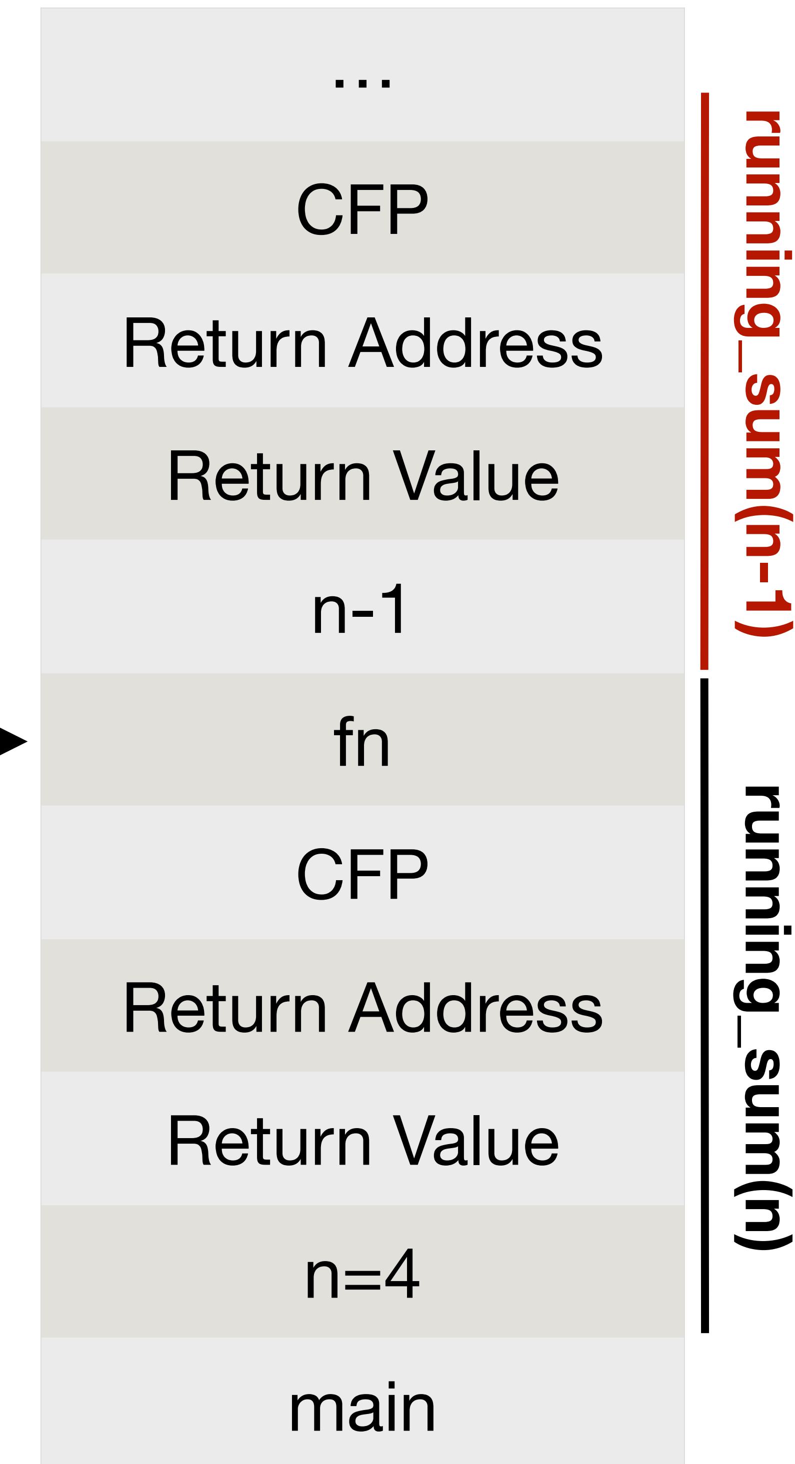
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1

;calculate n + Running(n-1)
LDR R1, R5, #4
```

R0	6
R1	4



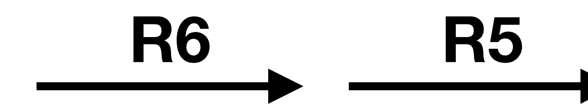


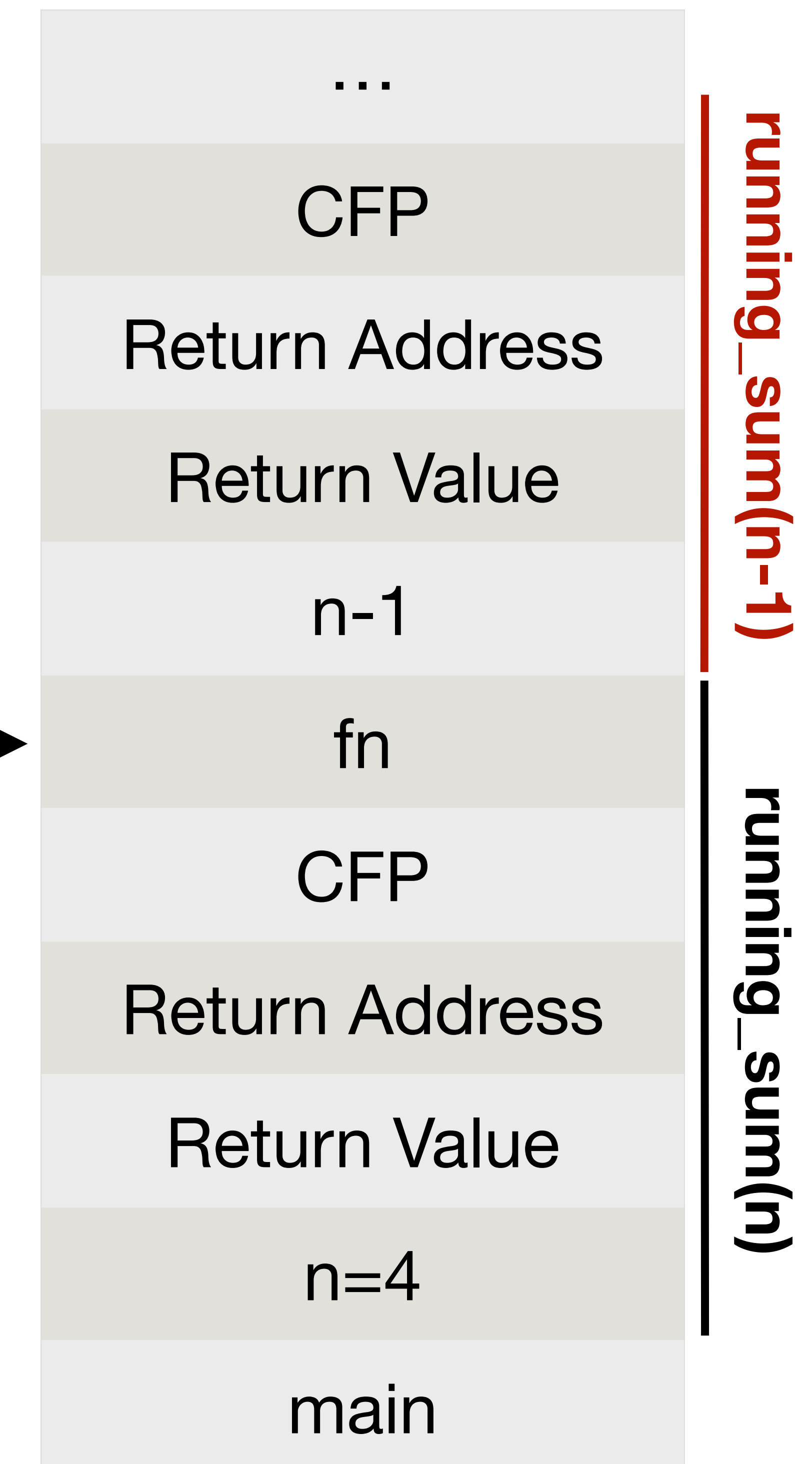
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1

;calculate n + Running(n-1)
LDR R1, R5, #4
ADD R0, R1, R0
```

R0	10
R1	4



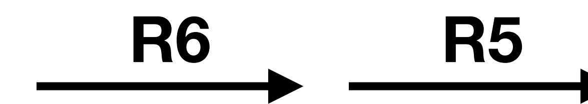


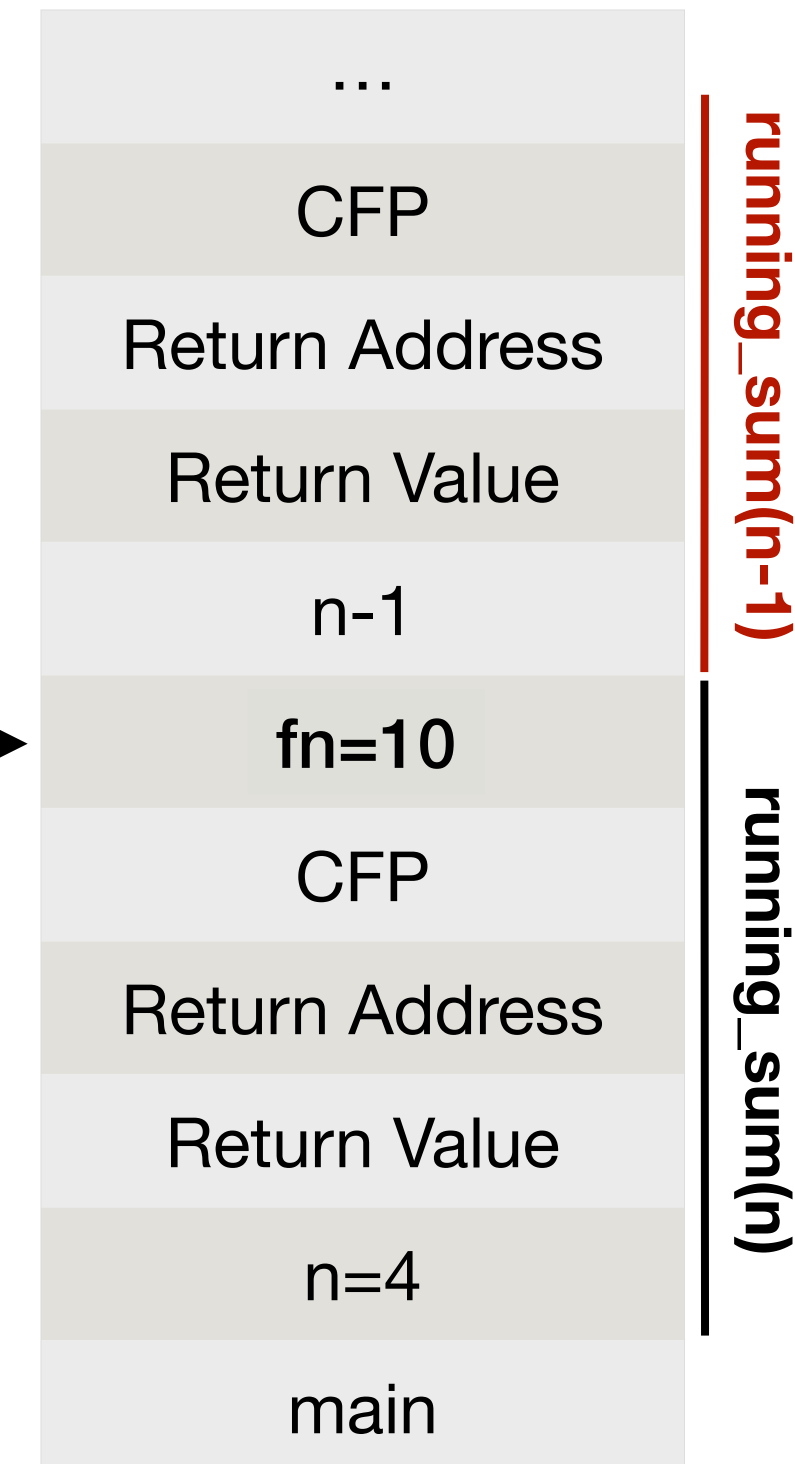
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1

;calculate n + Running(n-1)
LDR R1, R5, #4
ADD R0, R1, R0
STR R0, R5, #0 ;store result in fn
```

R0	10
R1	4



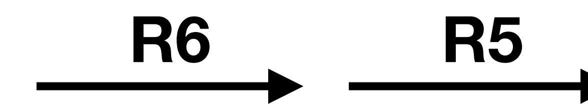


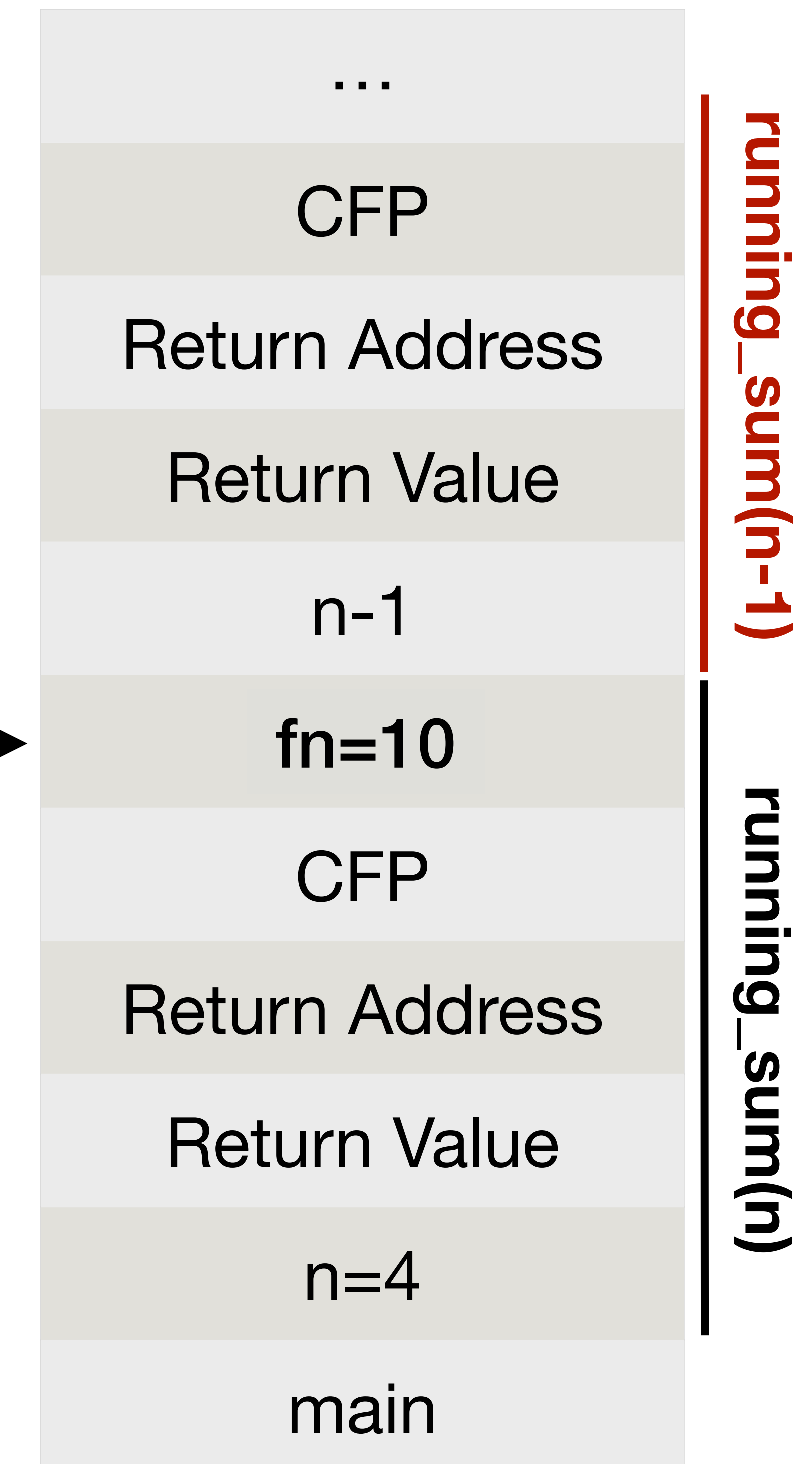
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1

;calculate n + Running(n-1)
LDR R1, R5, #4
ADD R0, R1, R0
STR R0, R5, #0 ;store result in fn

;ready to return
```

R0	10
R1	4





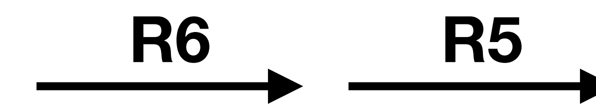
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

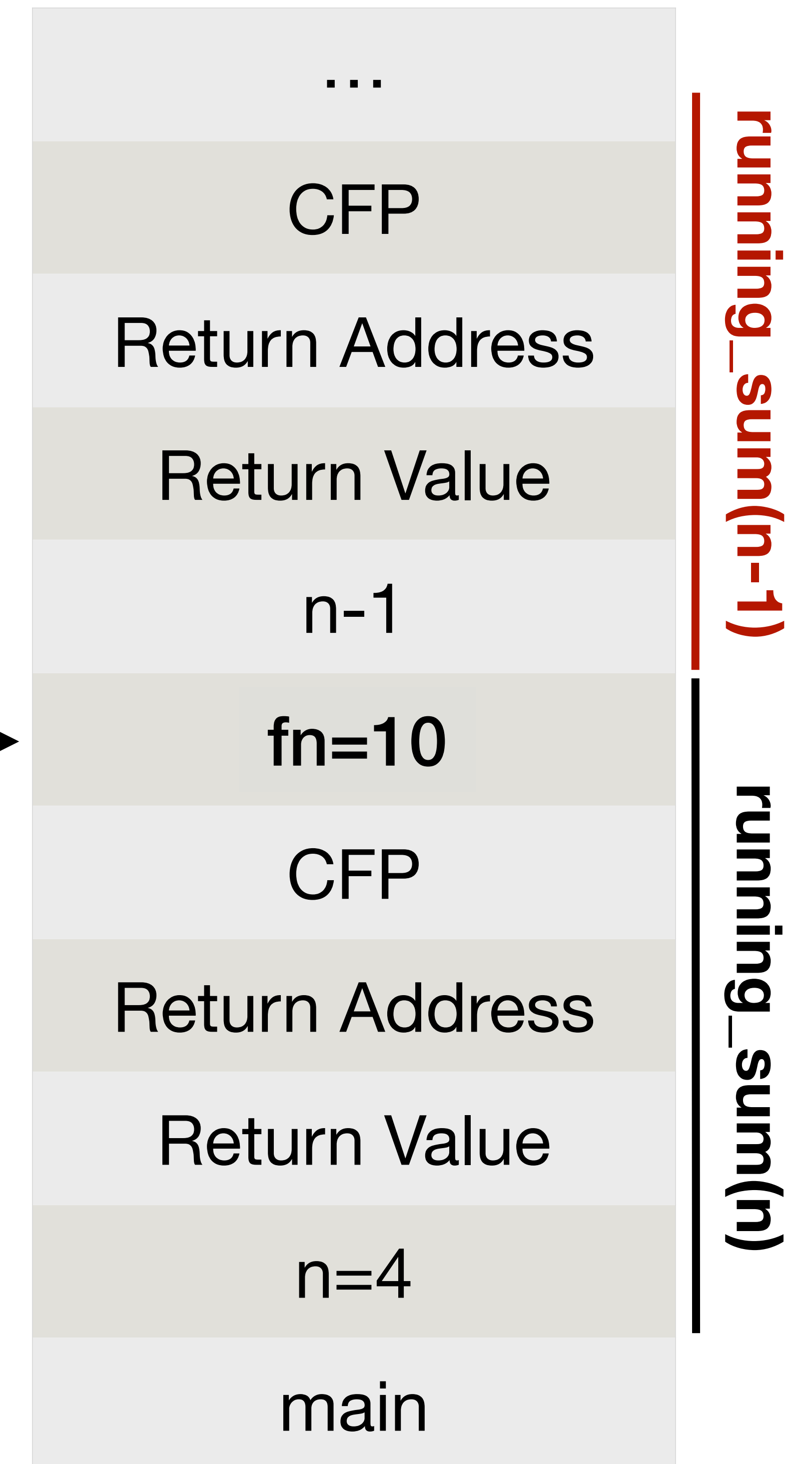
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
;Recursive case
;Caller setup: push argument n-1 onto RTS
ADD R6, R6, #-1
STR R2, R6, #0 ; R2 = n - 1
JSR RUNNING ; call Running(n-1)

;Callee tear-down for Running(n-1) not shown

;Caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R0
LDR R0, R6, #0
ADD R6, R6, #1

;pop Running(n-1)'s argument
ADD R6, R6, #1

;calculate n + Running(n-1)
LDR R1, R5, #4
ADD R0, R1, R0
STR R0, R5, #0 ;store result in fn

;ready to return
BRnzp RETURN
```

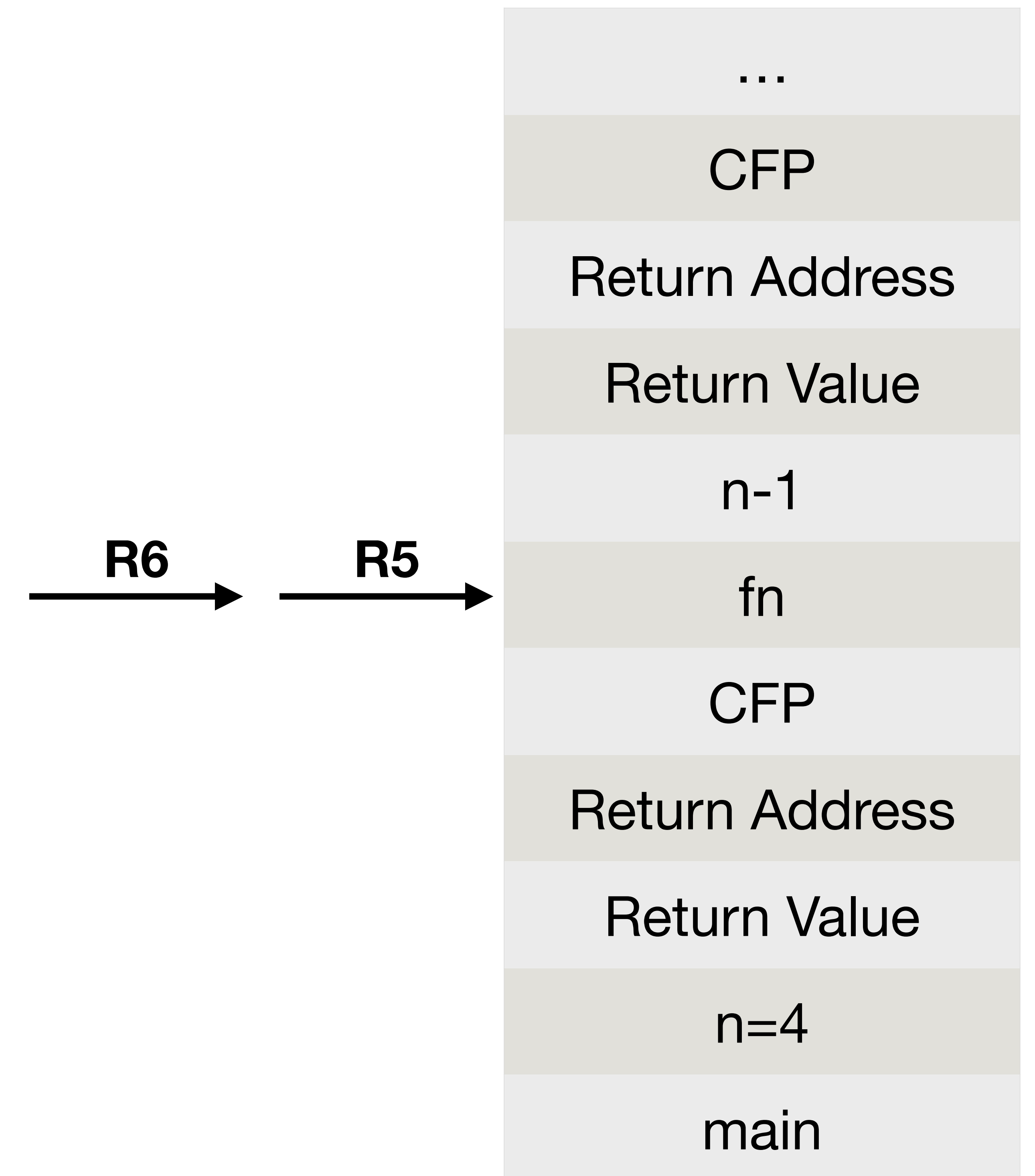
R0	10
R1	4





```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

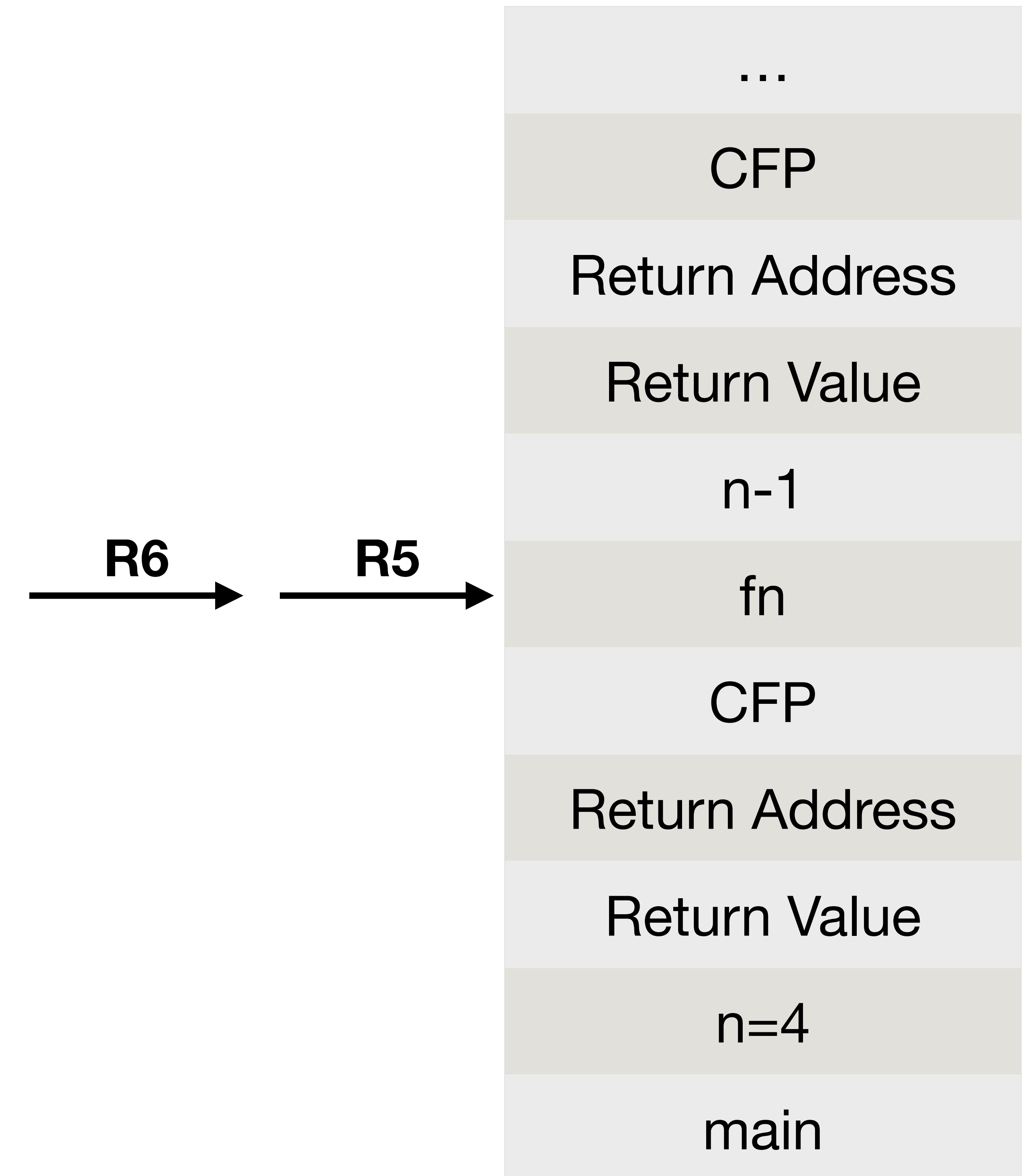
Recursion in LC3



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

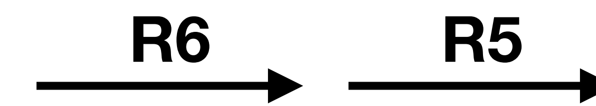
```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
```

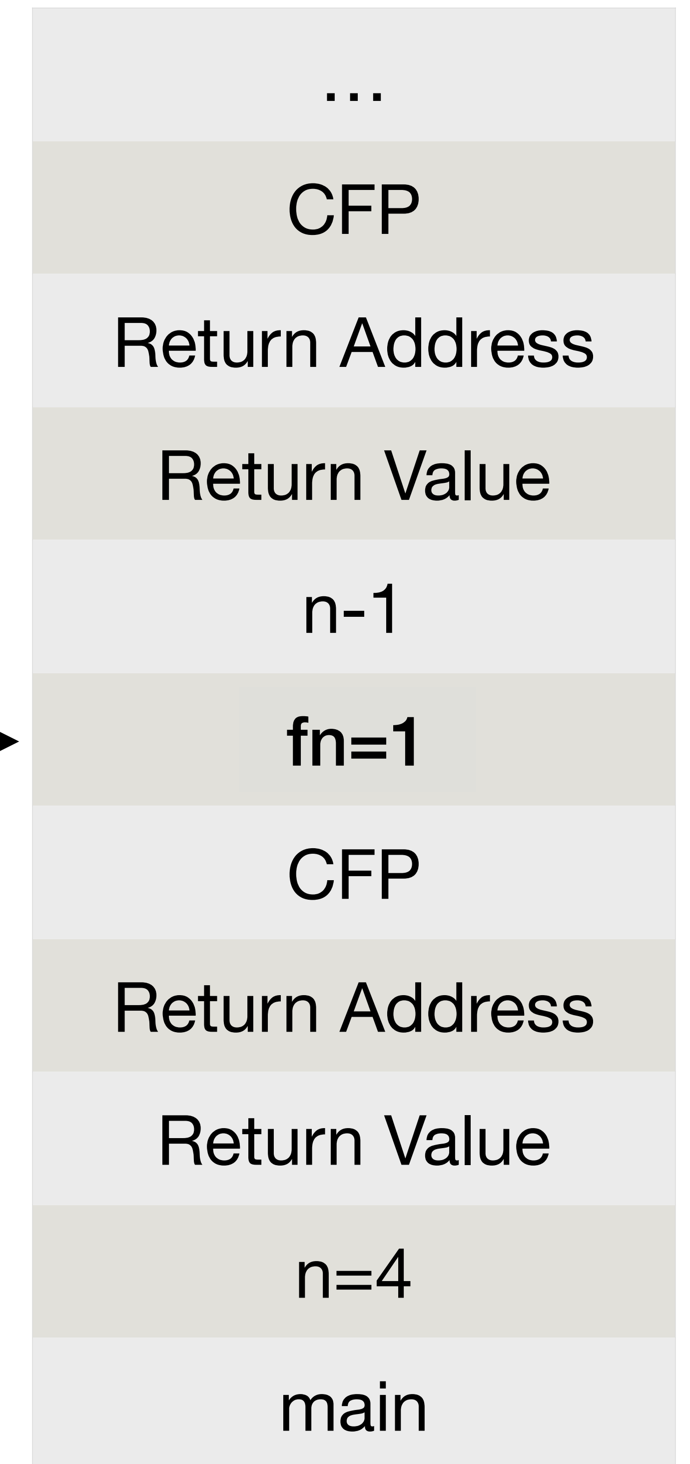


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1
```

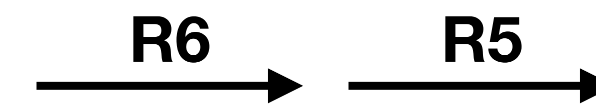


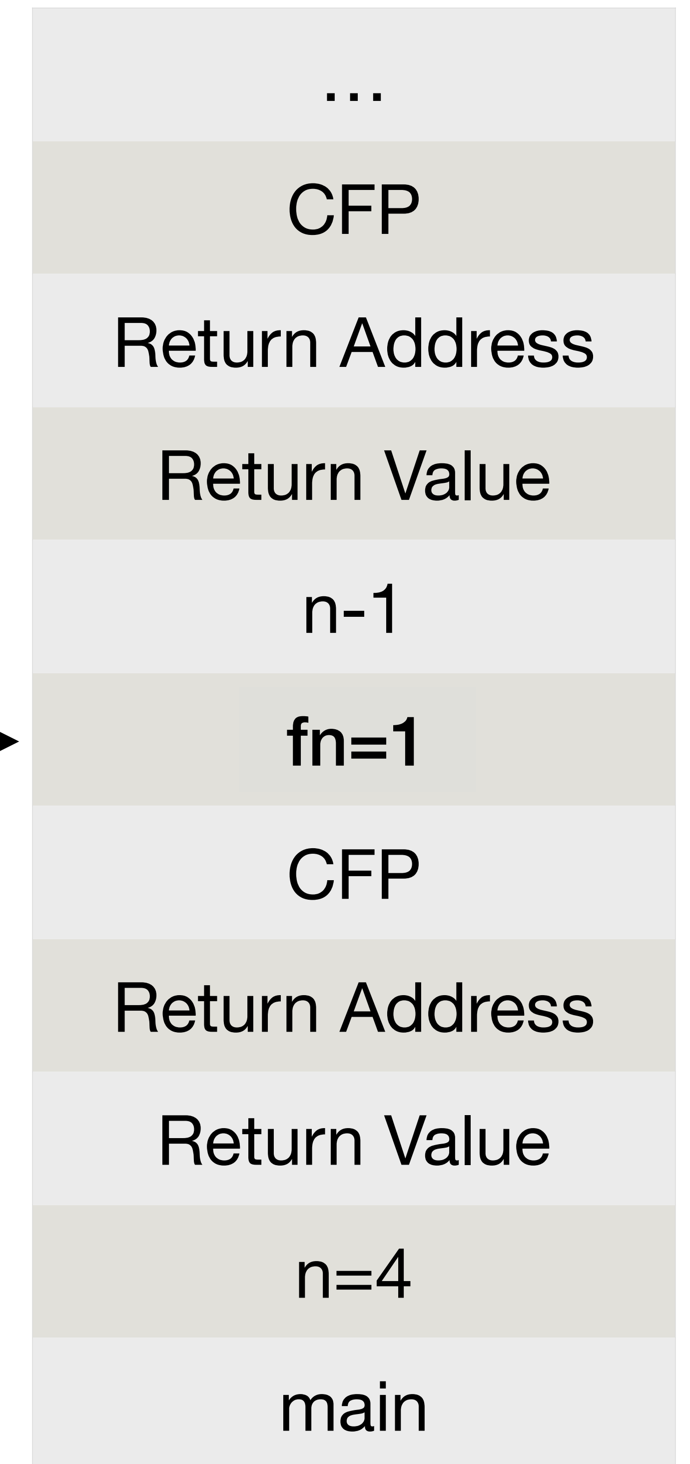


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3
```

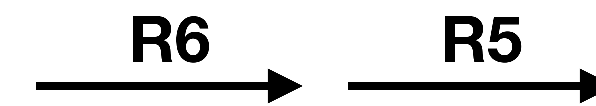


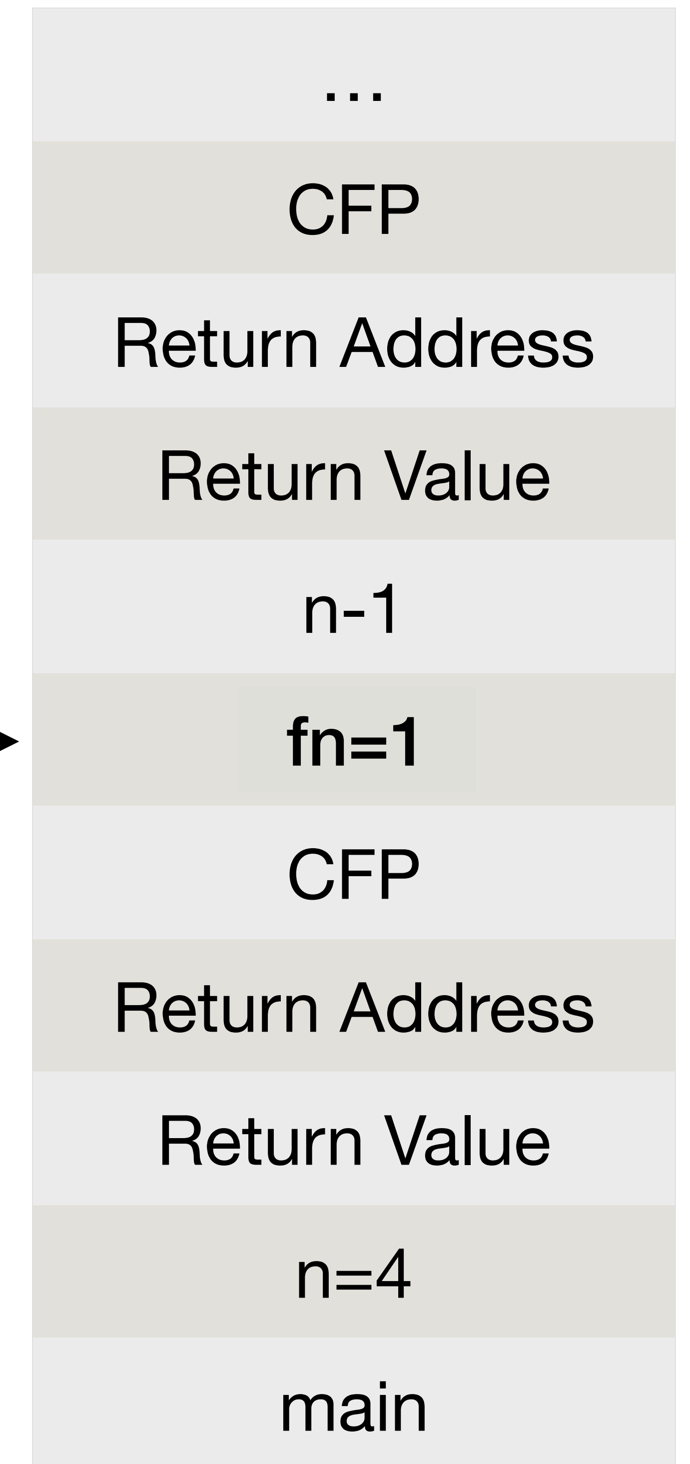


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
```

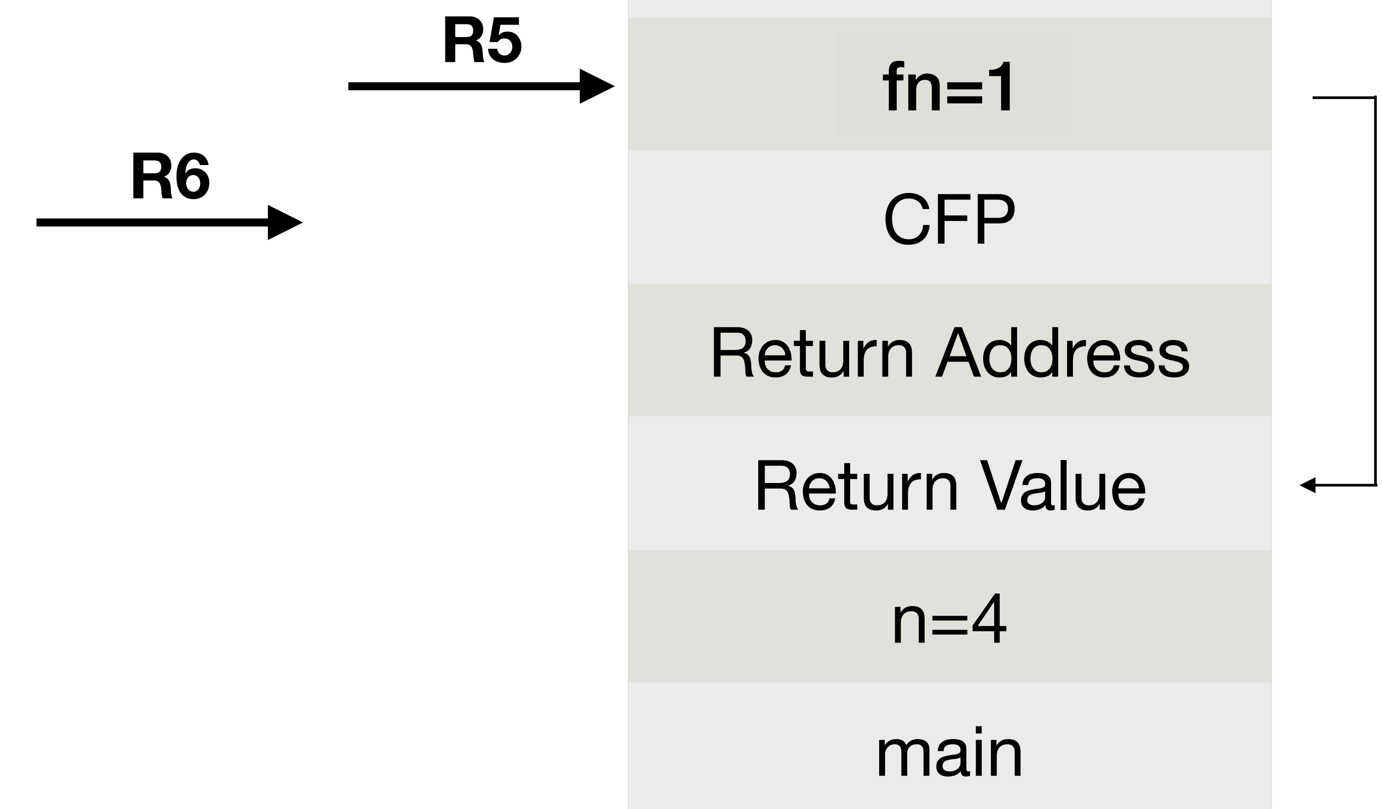




```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

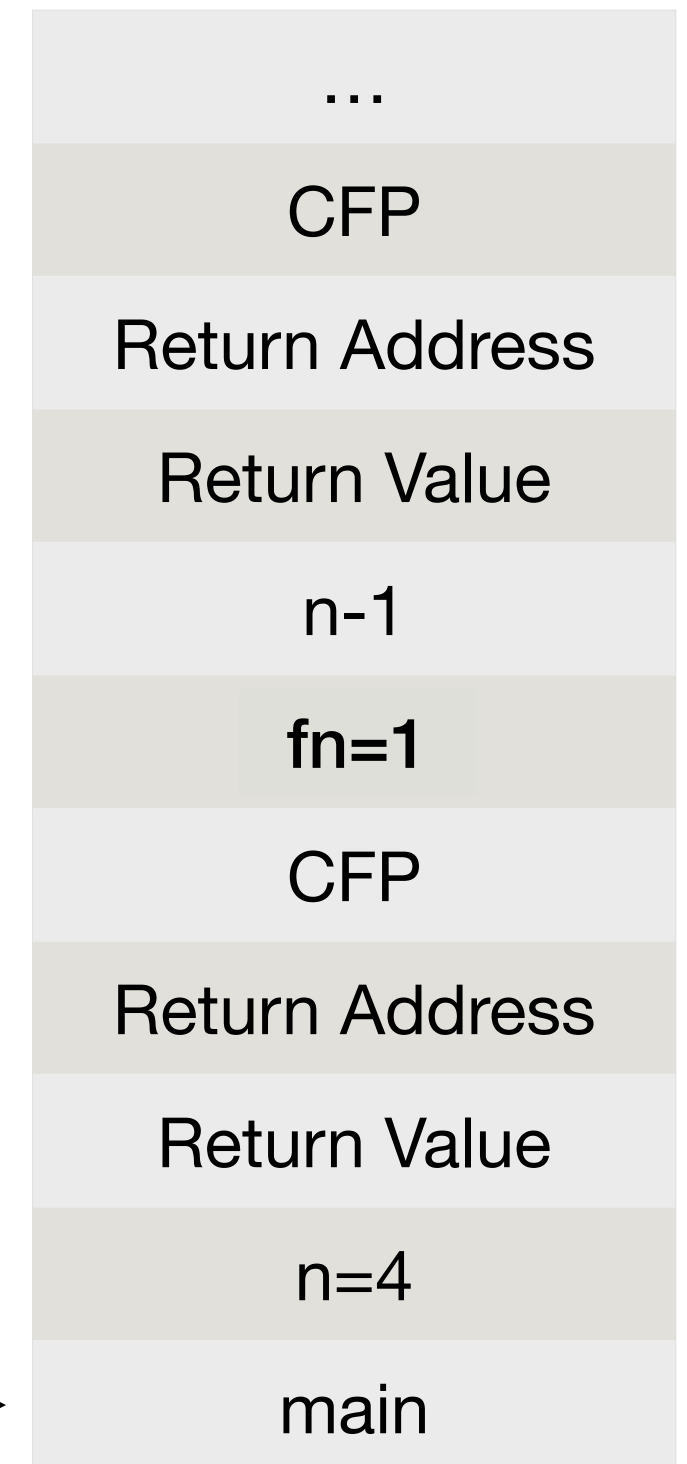
```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

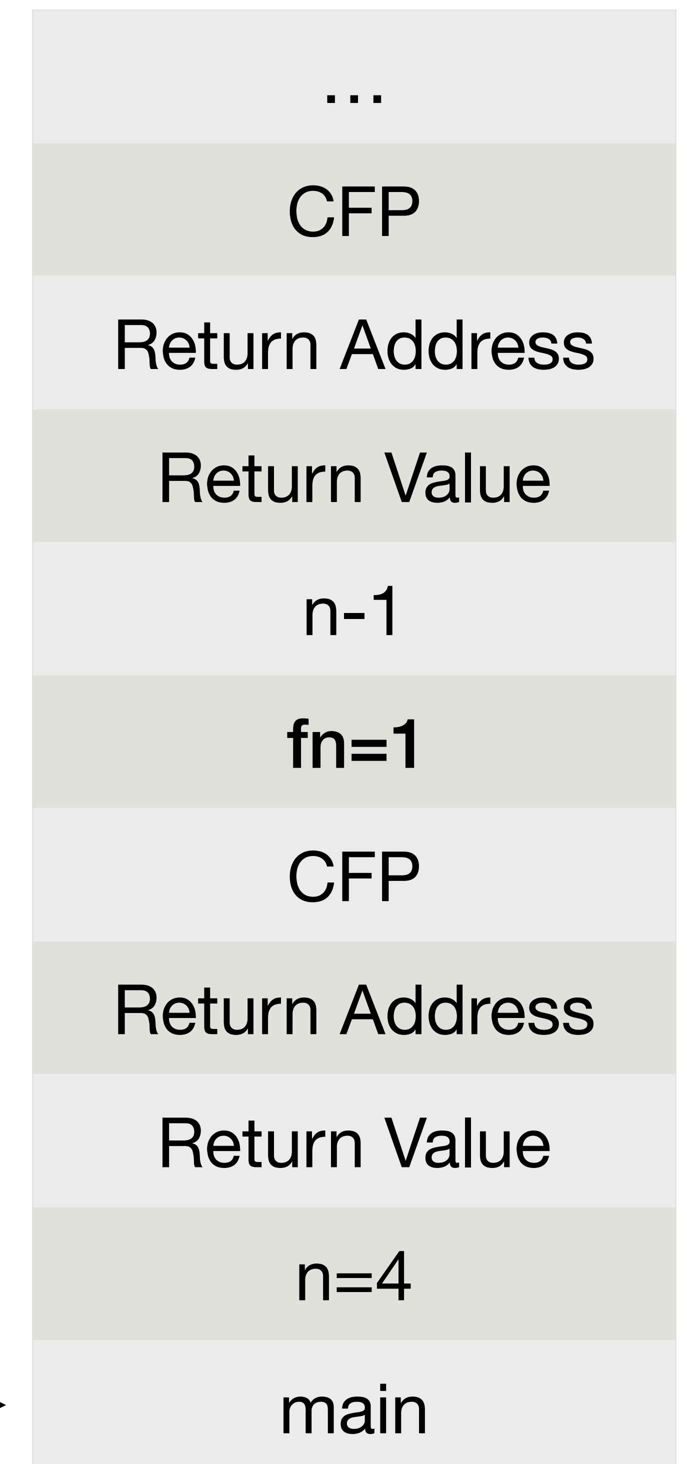
```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables

;restore caller's frame pointer and return address
LDR R5, R6, #0
```



```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

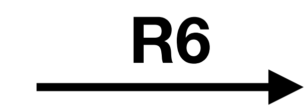
```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables

;restore caller's frame pointer and return address
LDR R5, R6, #0
ADD R6, R6, #1
```

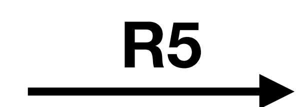


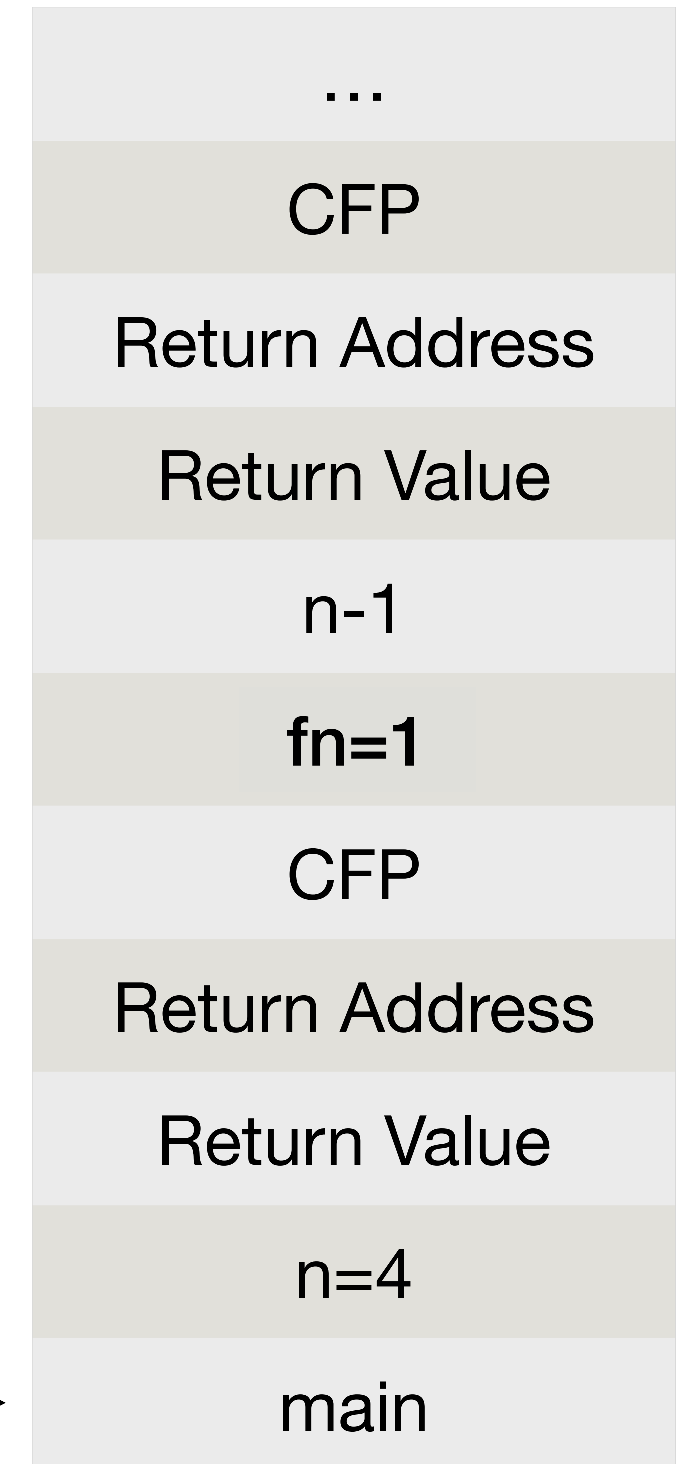
```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables

;restore caller's frame pointer and return address
LDR R5, R6, #0
ADD R6, R6, #1
LDR R7, R6, #0
```



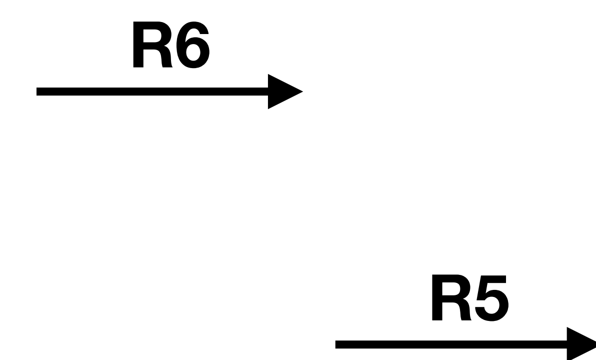


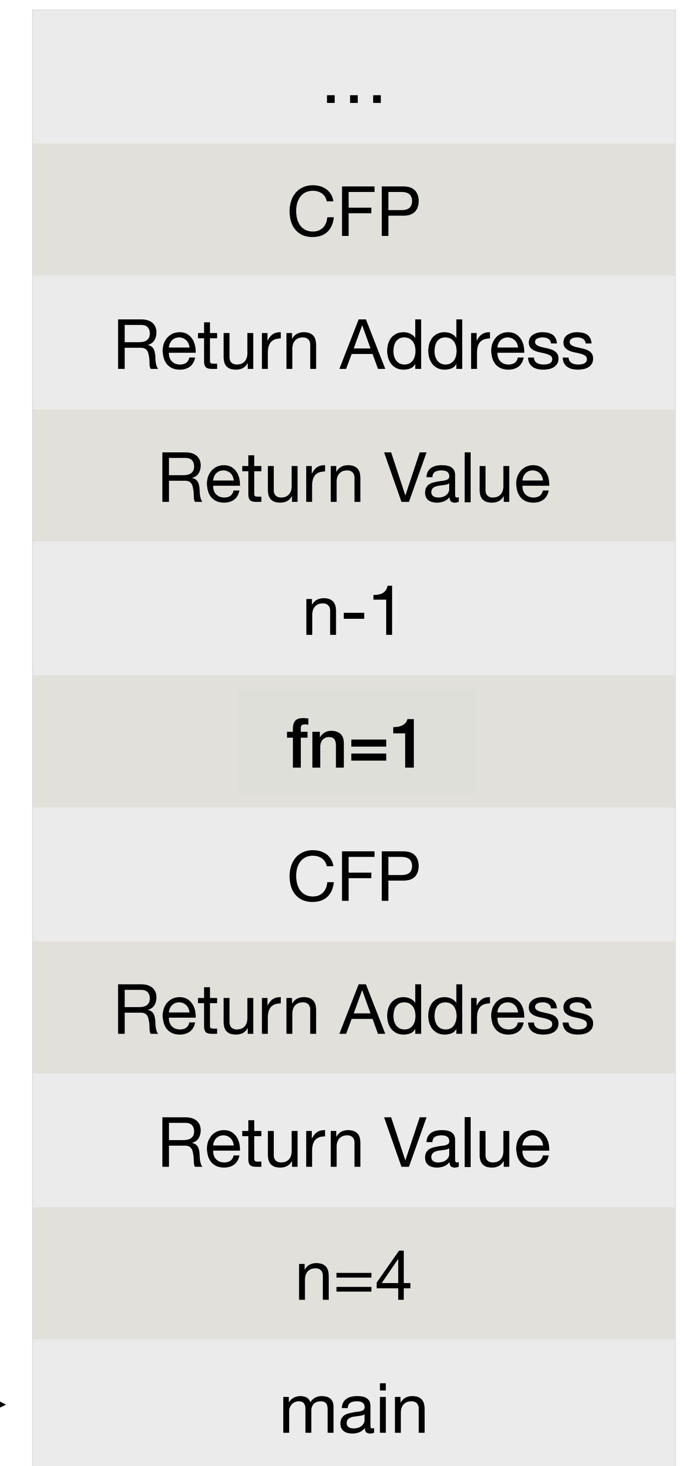


```
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables

;restore caller's frame pointer and return address
LDR R5, R6, #0
ADD R6, R6, #1
LDR R7, R6, #0
ADD R6, R6, #1
```

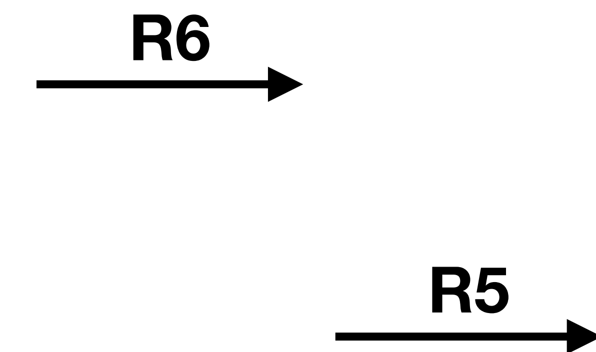


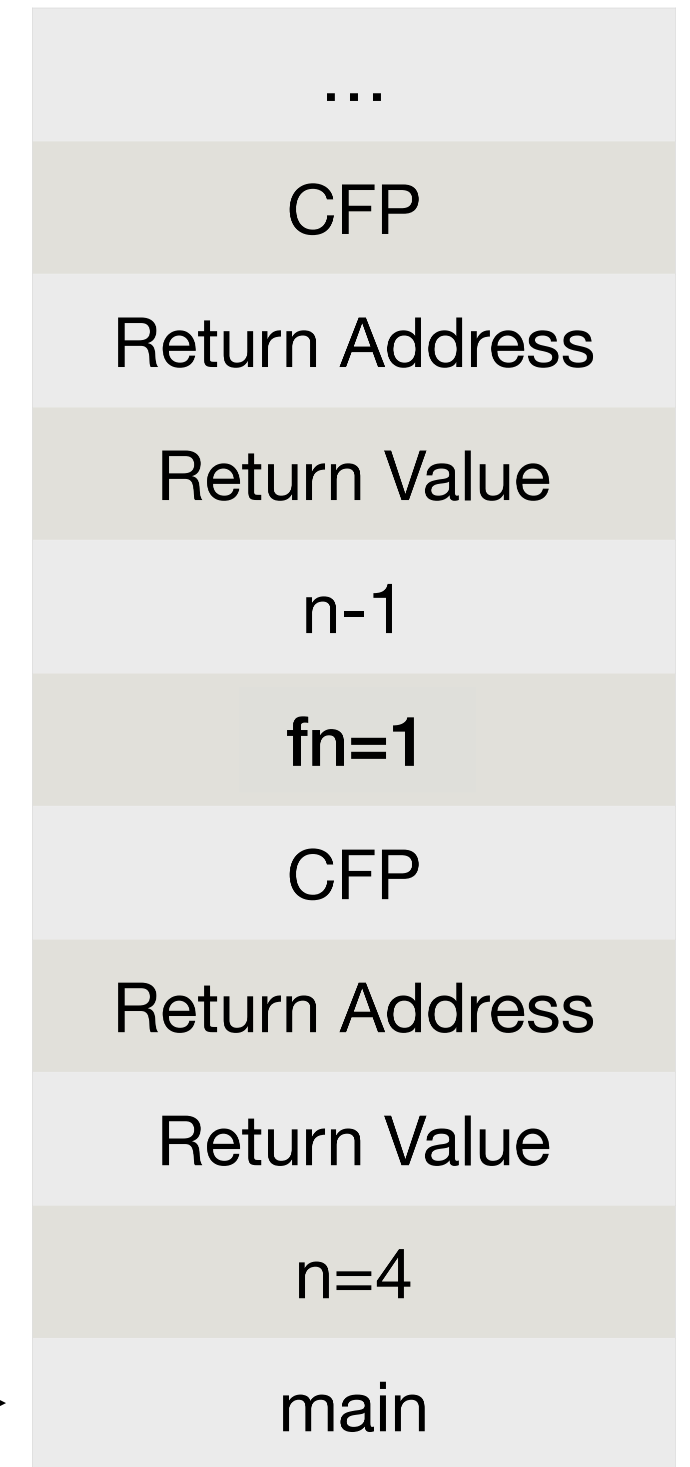


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables

;restore caller's frame pointer and return address
LDR R5, R6, #0
ADD R6, R6, #1
LDR R7, R6, #0
ADD R6, R6, #1

;return to caller
```

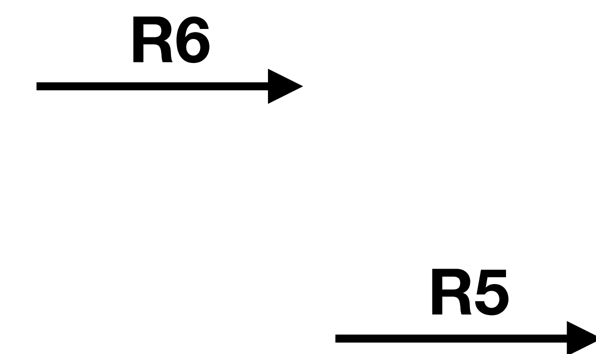


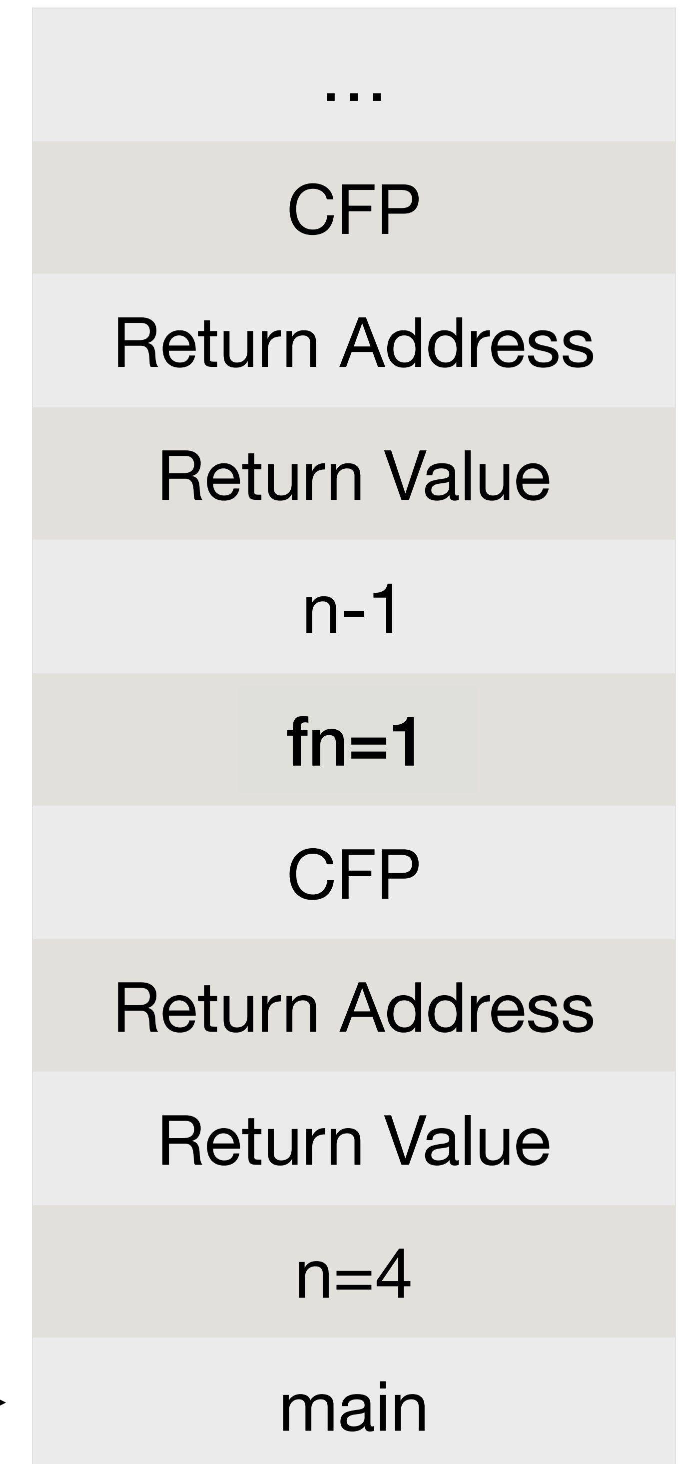


```c
int running_sum(int n){
    int fn;
    if (n==1)
        fn = 1;
    else
        fn = n + running_sum(n-1);
    return fn;
}
```

Recursion in LC3

```
BASE_CASE
AND R2, R2, #0
ADD R2, R2, #1
STR R2, R5, #0 ;set fn = 1

RETURN
;set return value
LDR R0, R5, #0
STR R0, R5, #3

;callee tear-down of Running(n)'s activation record
ADD R6, R6, #1 ;pop local variables

;restore caller's frame pointer and return address
LDR R5, R6, #0
ADD R6, R6, #1
LDR R7, R6, #0
ADD R6, R6, #1

;return to caller
RET
```





Next time

- More problem solving with recursion.
  - A small chess problem
  - Solving a maze
- When is recursion good vs. bad?