ECE 2574: Data Structures and Algorithms - Stack ADT

C. L. Wyatt

Today we will define and use one of the most fundamental data structures in computing, a stack.

- Warmup
- Introduction to stacks
- Example: permutations
- Example: image processing

A stack is a list in which all insertions and deletions are done at one end, denoted the top.

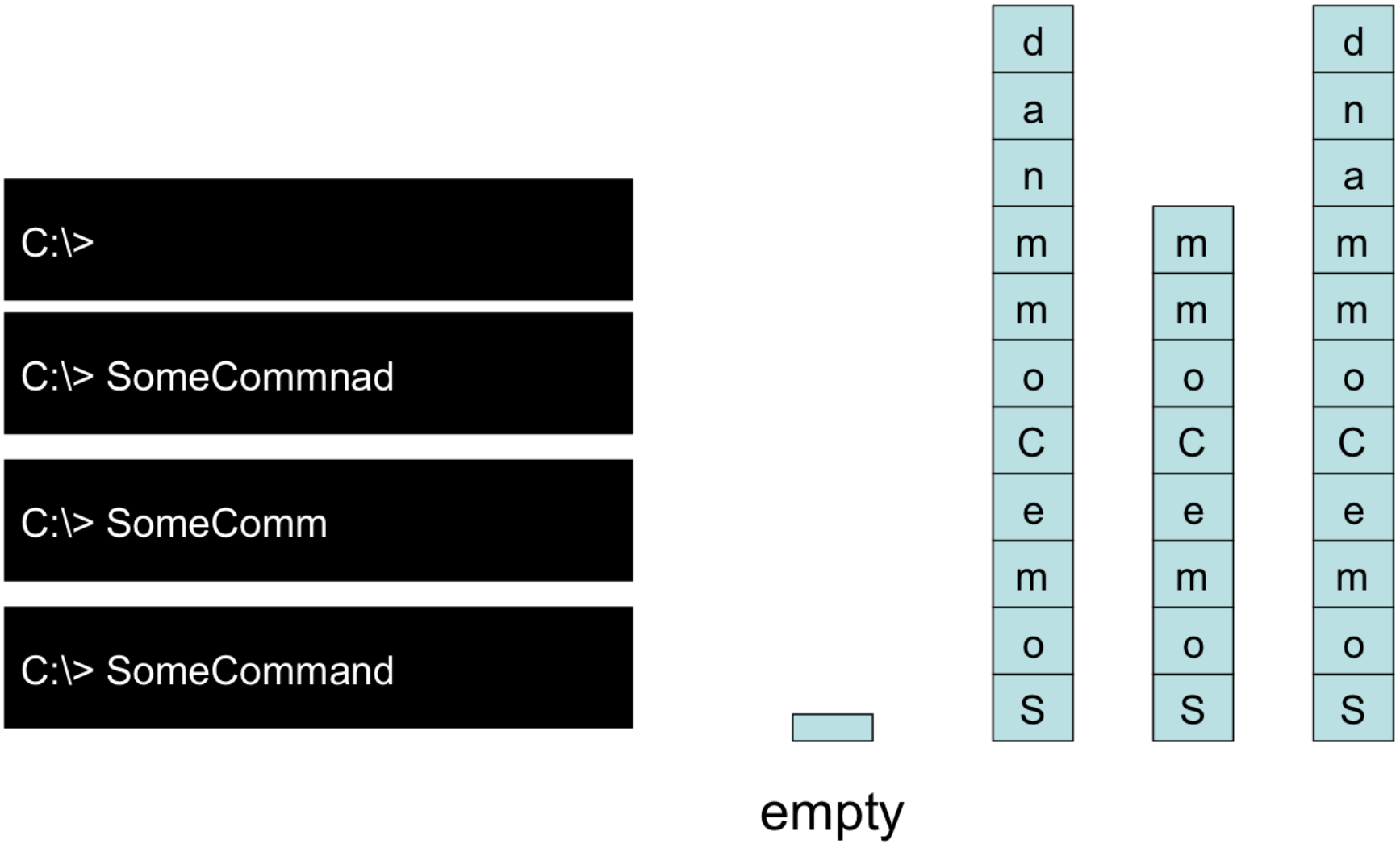
The basic stack ADT has 7 operations.

- create a stack
- destroy a stack
- empty query
- insert (push)
- remove (pop)
- retrieve (peek or top)

In the stack, **all access is limited to the top**.

A stack is also called a Last-In-First-Out (LIFO) Queue.

Example of using a stack: Entering text



Basic stack operations are similar to the Bag ADT.

```
// Create an empty stack
+createStack()

// Destroy a stack
+destroyStack

//Determine if a stack is empty
//Precondition: None
//Postcondition: returns true is the stack is empty,
//                    else false
+isEmpty(): boolean
```

Inserting onto a stack is called a push.

```
// adds new item to the top of the stack
// Precondition: valid stack
// Postcondition: stack has new item at top,
//                stack is 1 larger
// returns true/false if push succeeds/fails
+push(in newItem:StackItemType): boolean
```

Removing an item from the stack is called a pop.

```
// remove the top item in the stack
// Precondition: valid stack
// Postcondition: stack is 1 smaller, top item removed
// returns true/false if push succeeds/fails
+pop(): boolean

// retrieve and remove the top item in the stack
// Precondition: valid stack
// Postcondition: stack is 1 smaller, top item removed
// returns true/false if push succeeds/fails
+pop(out stackTop:StackItemType): boolean
```

Retrieving from the stack top without removing is sometimes called peek.

```
// retrieves the item currently at the stack top.
// Precondition: valid stack
// Postcondition: places stack top in stackTop
// output returns true/false if push succeeds/fails
+getTop(outstackTop:StackItemType): boolean
```

Warmup

Determine the stack contents after the operation on each line is executed. Be sure to indicate the top of the stack.

```
1    stack<int> s;
2    s.push(1);
3    s.push(2);
4    s.pop();
5    s.pop();
6    s.push(34);
7    s.push(-12);
8    s.push (15);
9    s.pop();
10   s.push(100);
11   s.push(0);
```

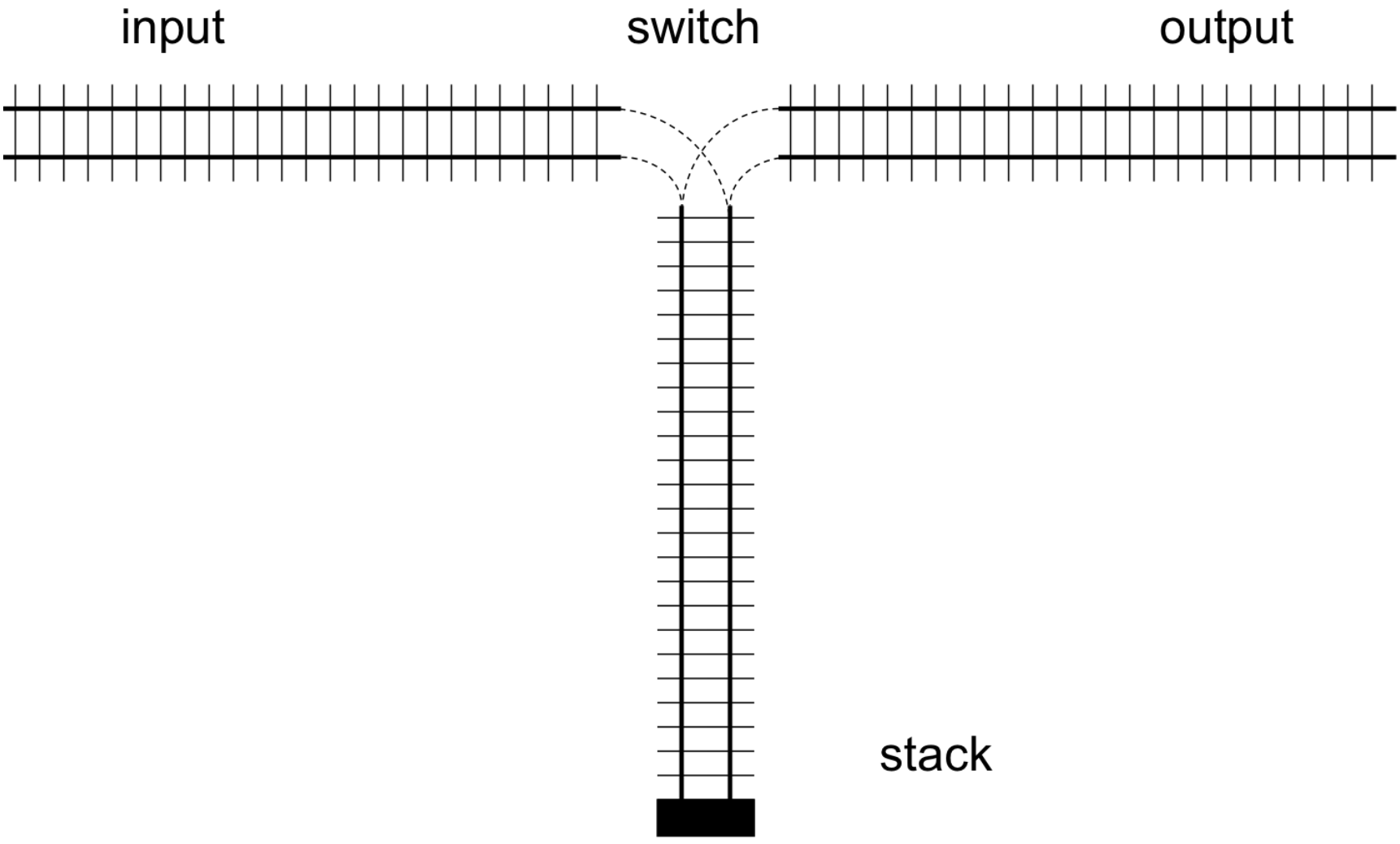
Of the 45 who submitted, 91% correct.

Stacks are prevalent in computer systems.

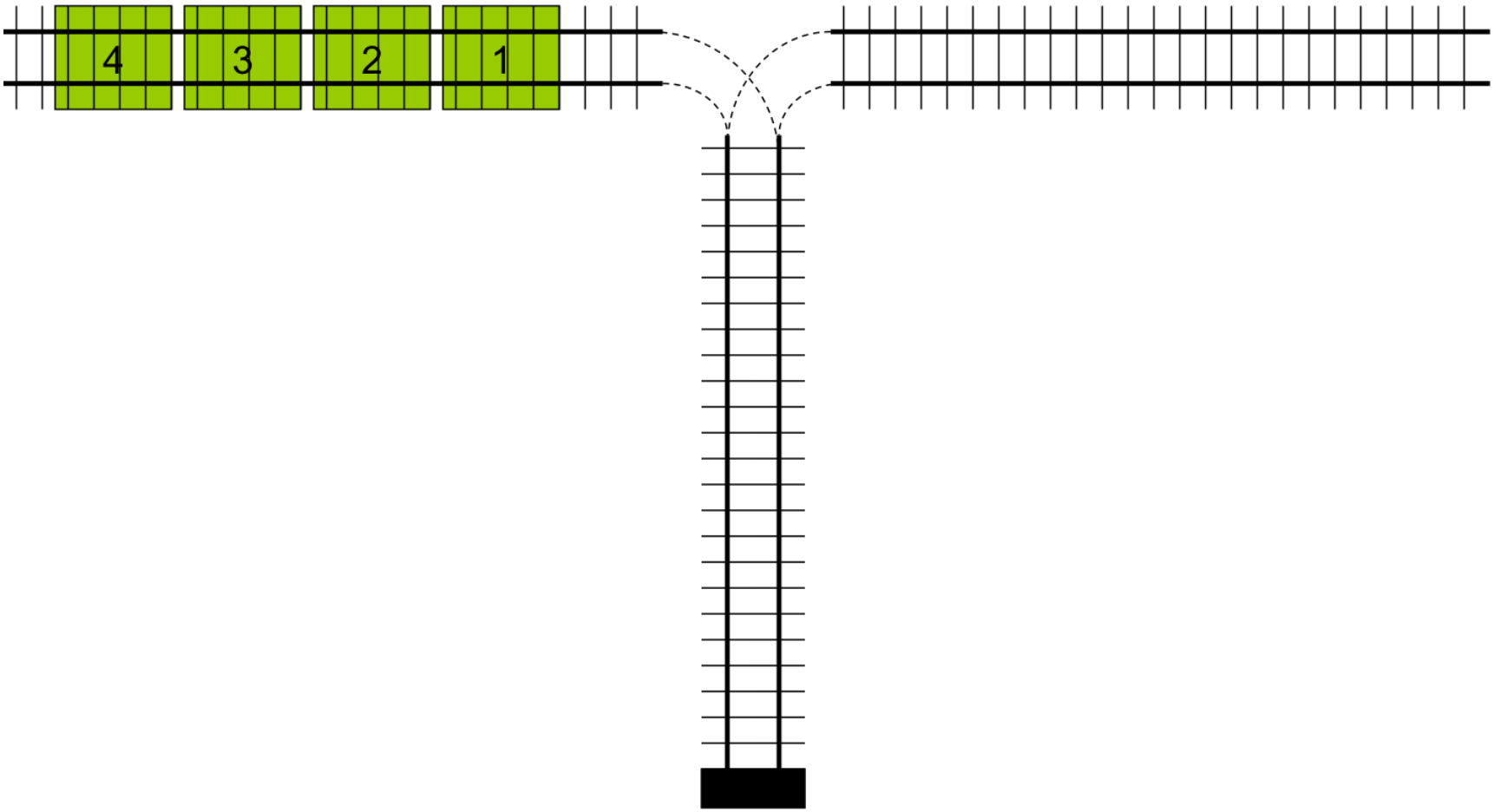
At a low-level stacks are used to store local variables, function arguments, return addresses, etc. Many algorithms are conveniently described in terms of a stack concept.

Stacks are called push-down lists in automata theory.

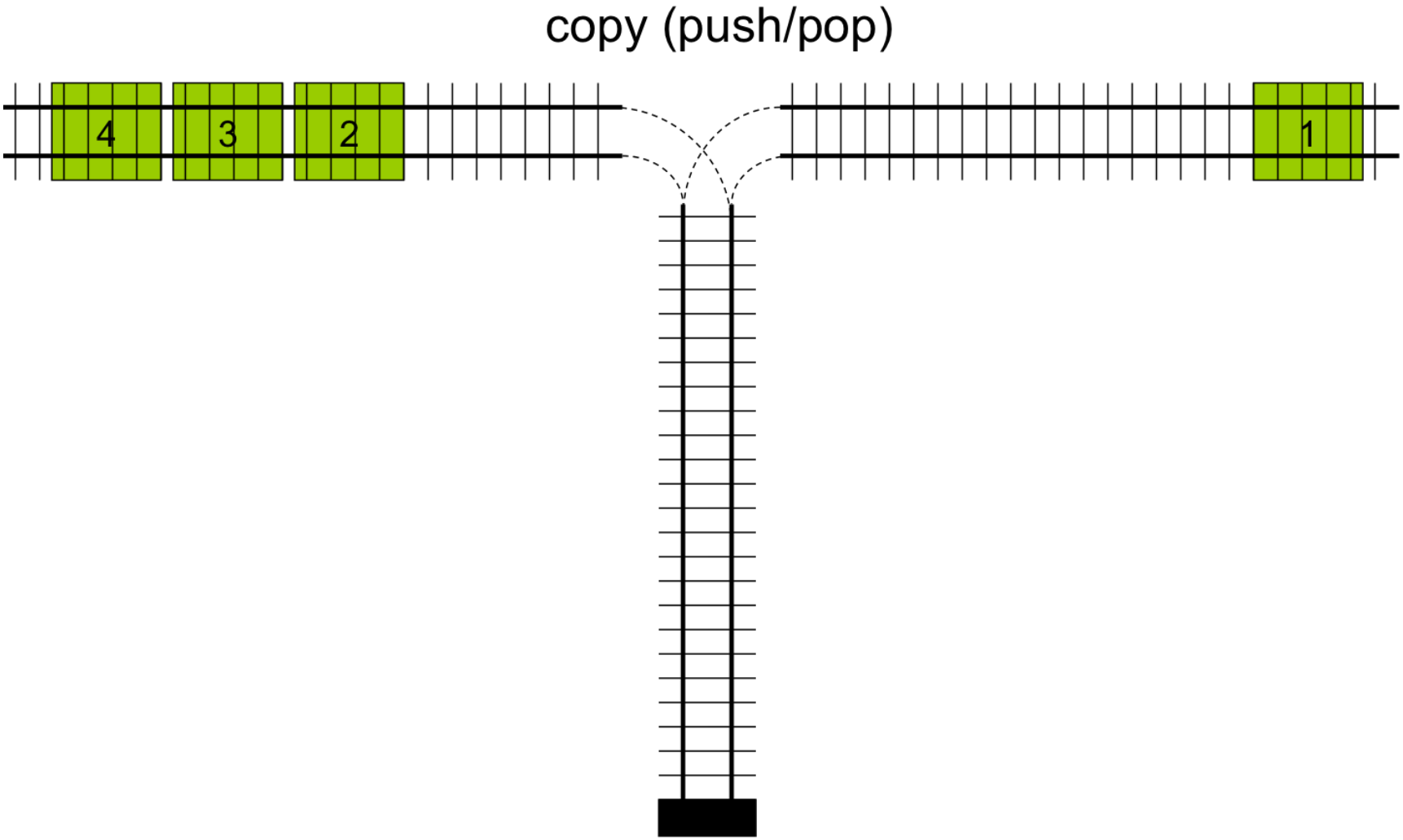
Example: creating permutations



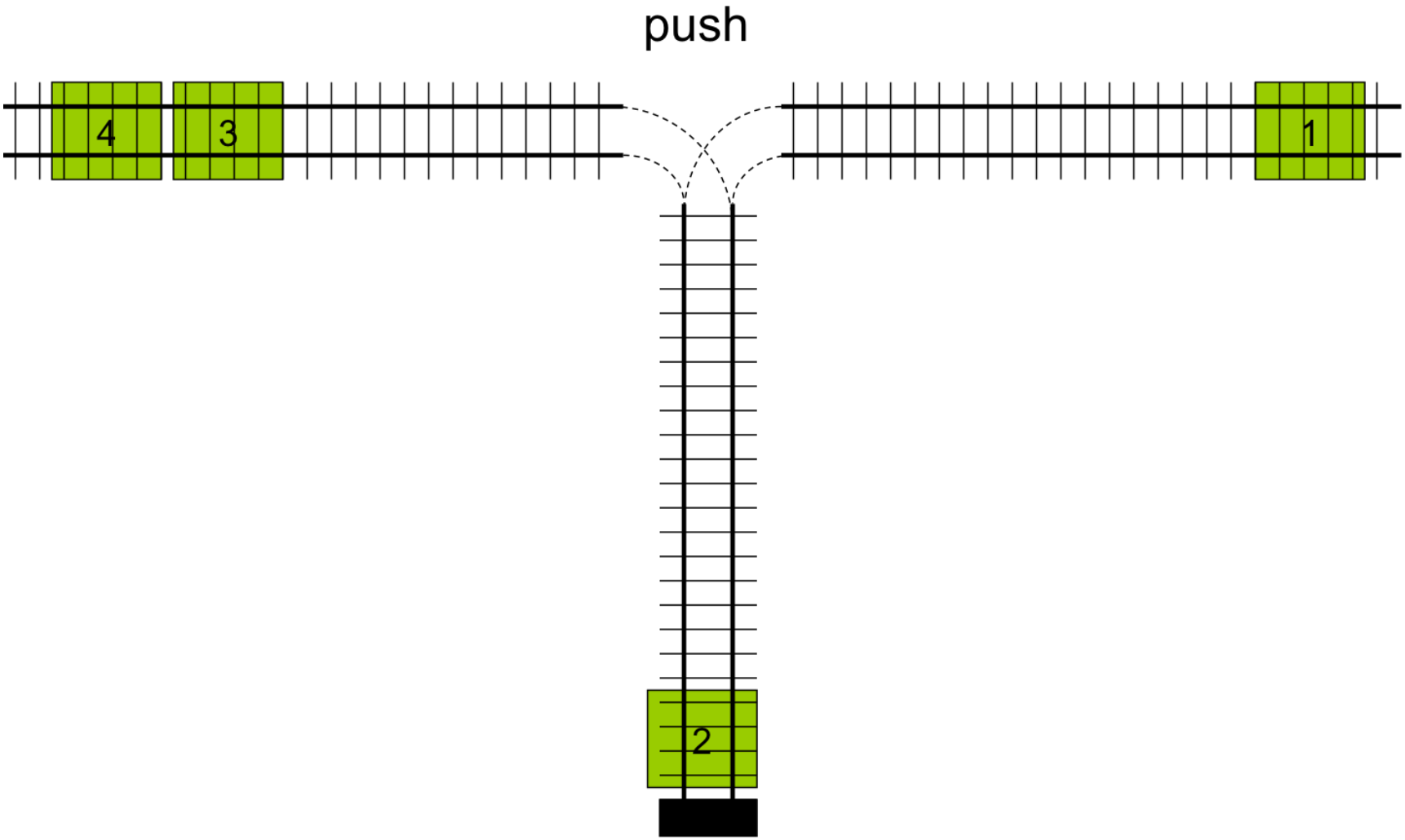
Example: creating permutations



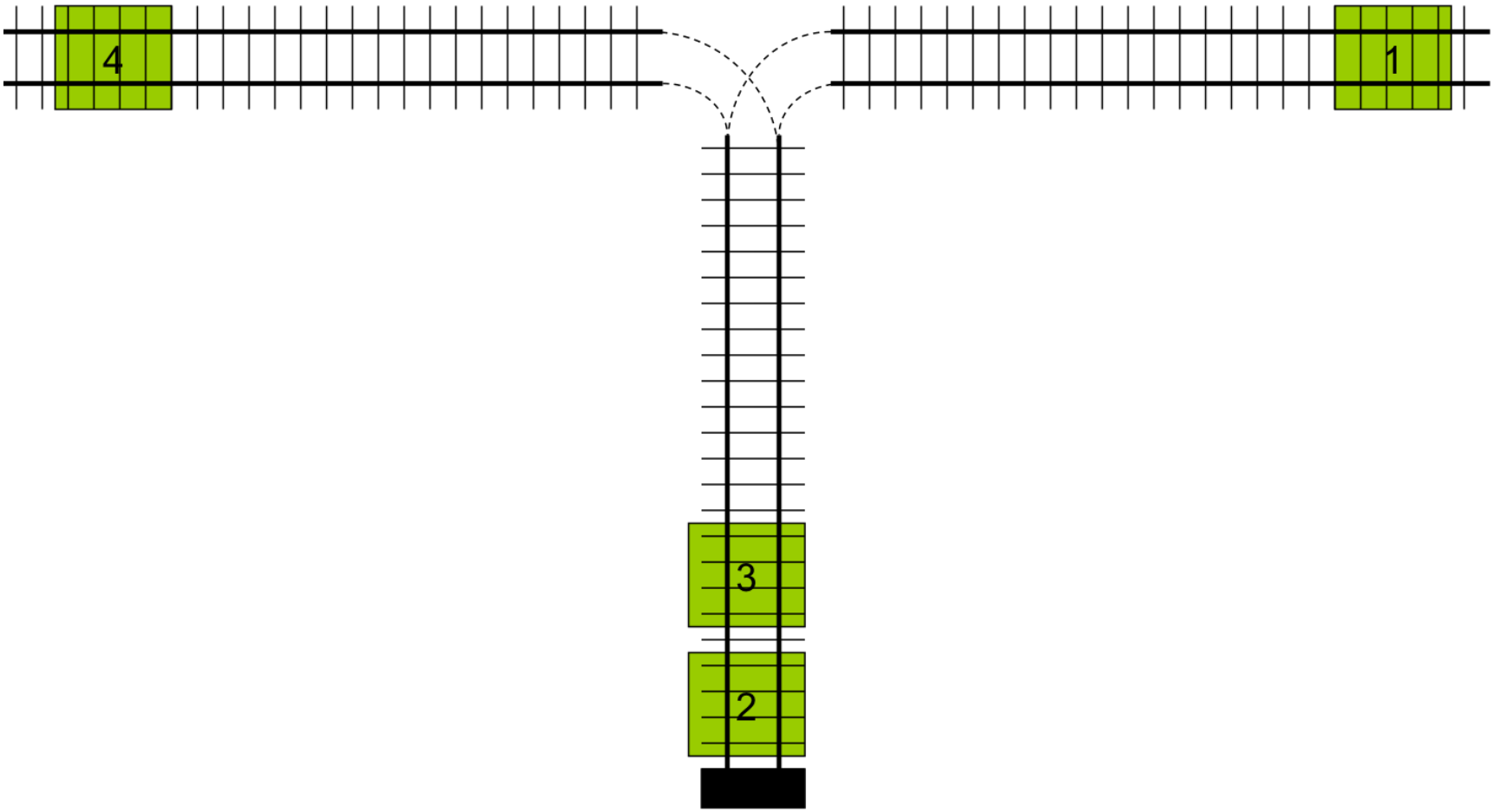
Example: creating permutations



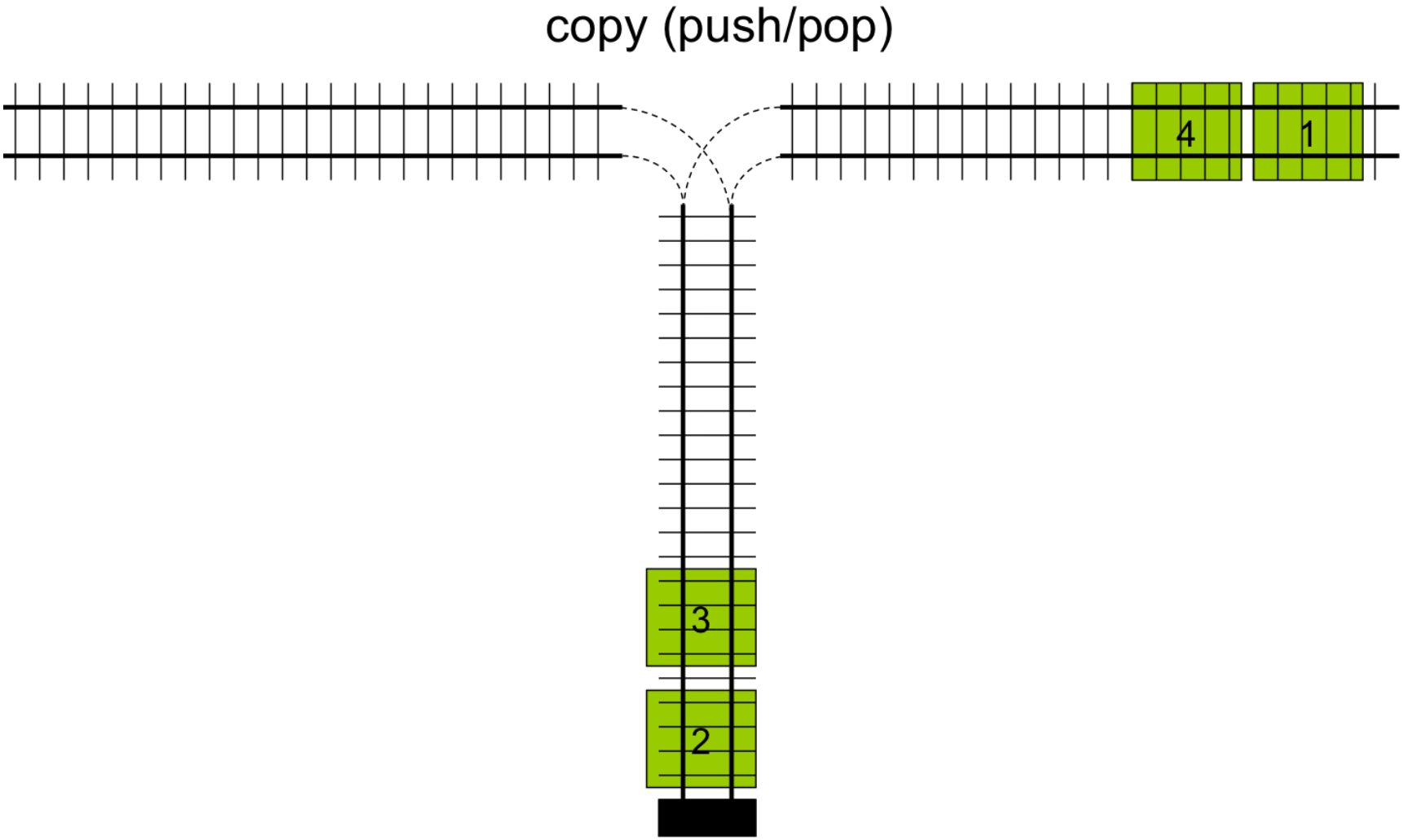
Example: creating permutations



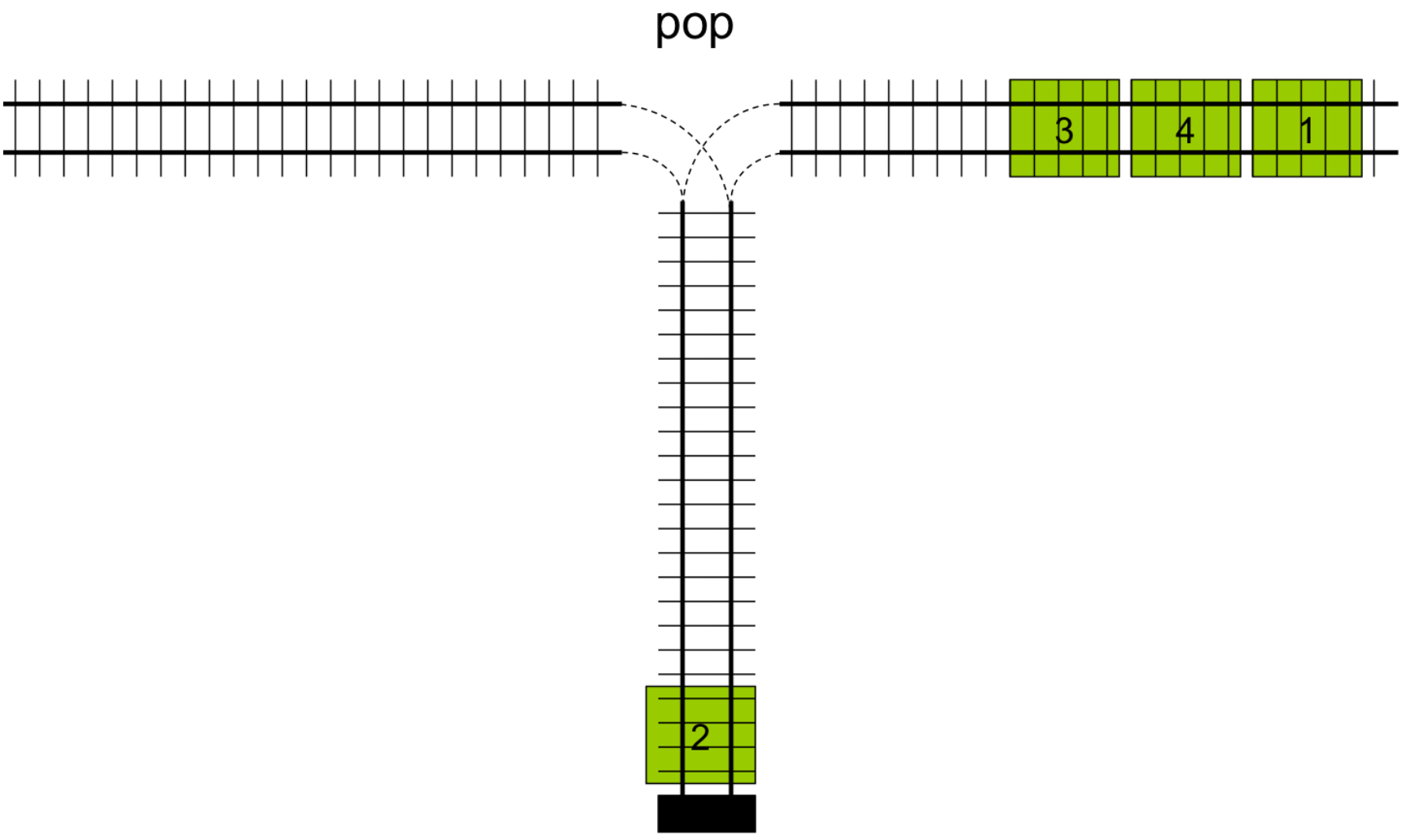
Example: creating permutations



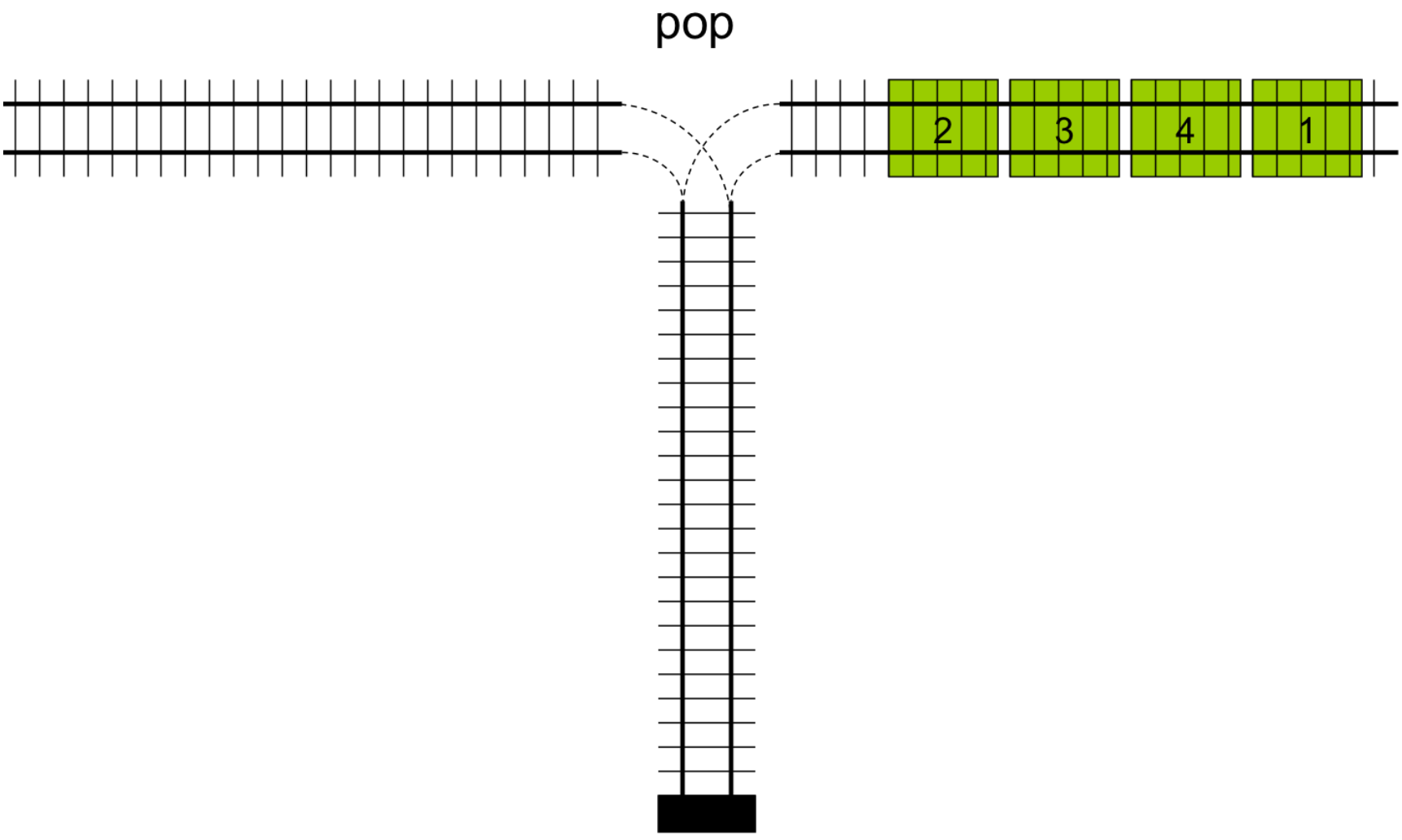
Example: creating permutations



Example: creating permutations



Example: creating permutations



In class exercise: creating permutations

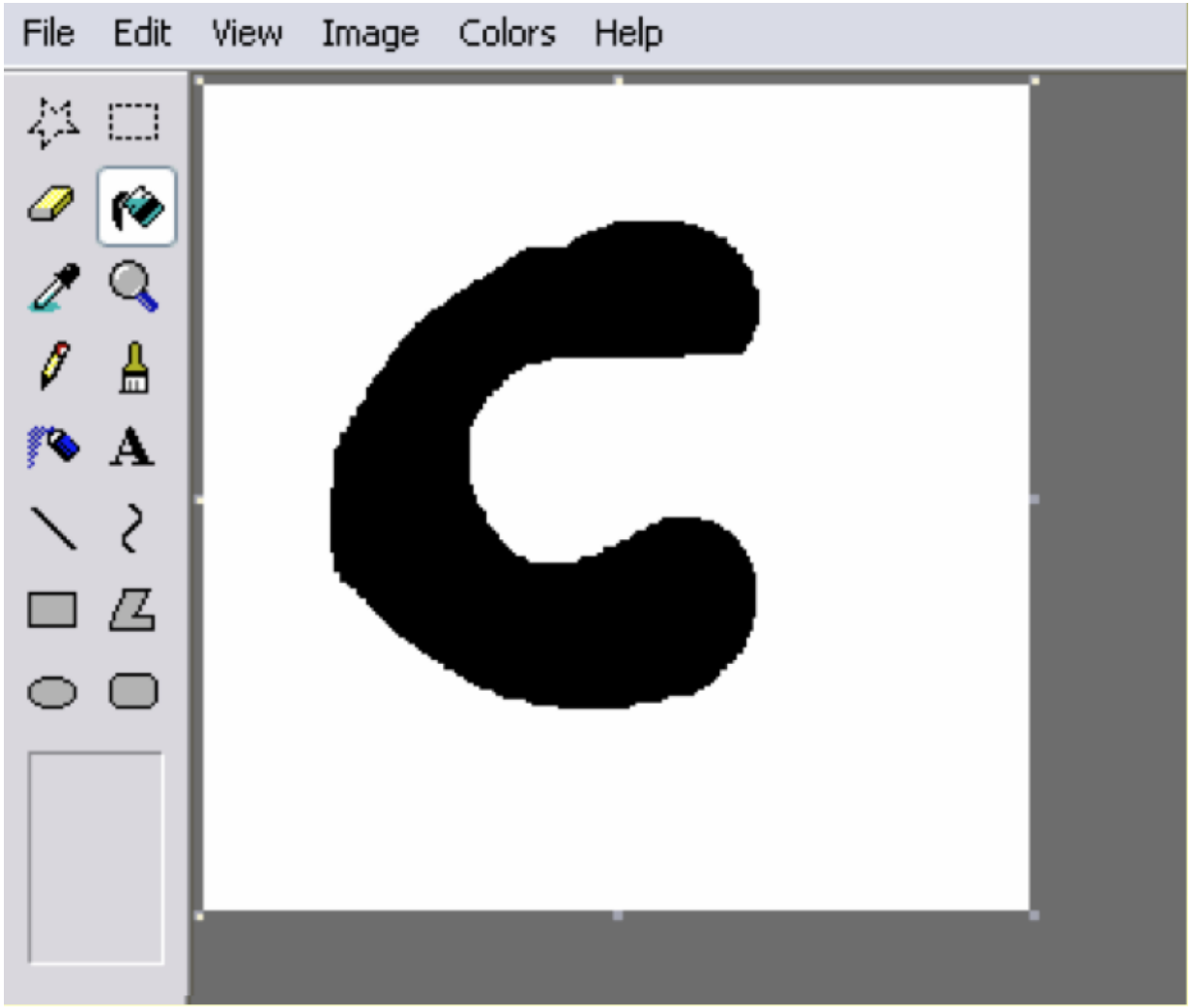
Given an input (left-to-right): 4, 3, 2, 1

Can you create the permutation (left-to-right): 1, 4, 2, 3 ?

What sequence of push/pops would perform the permutation?

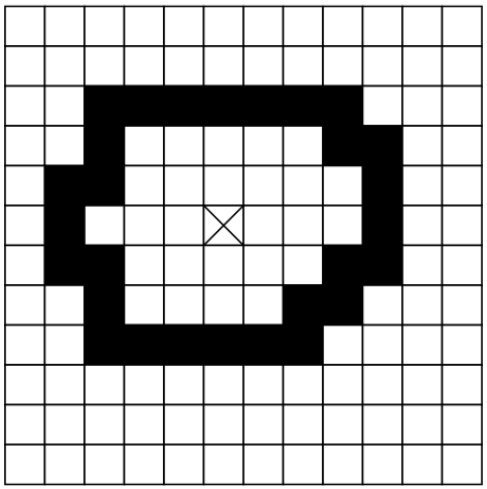
Another real world problem where the stack solution is particularly elegant is region-growing in images.

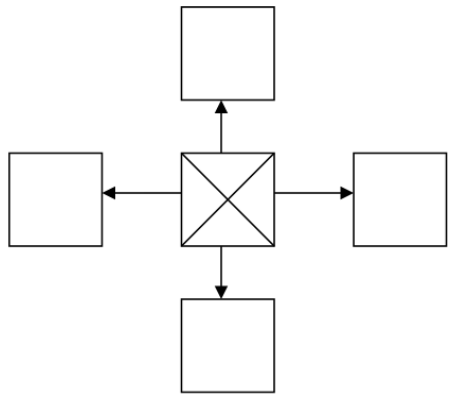
Also called flood-fill.



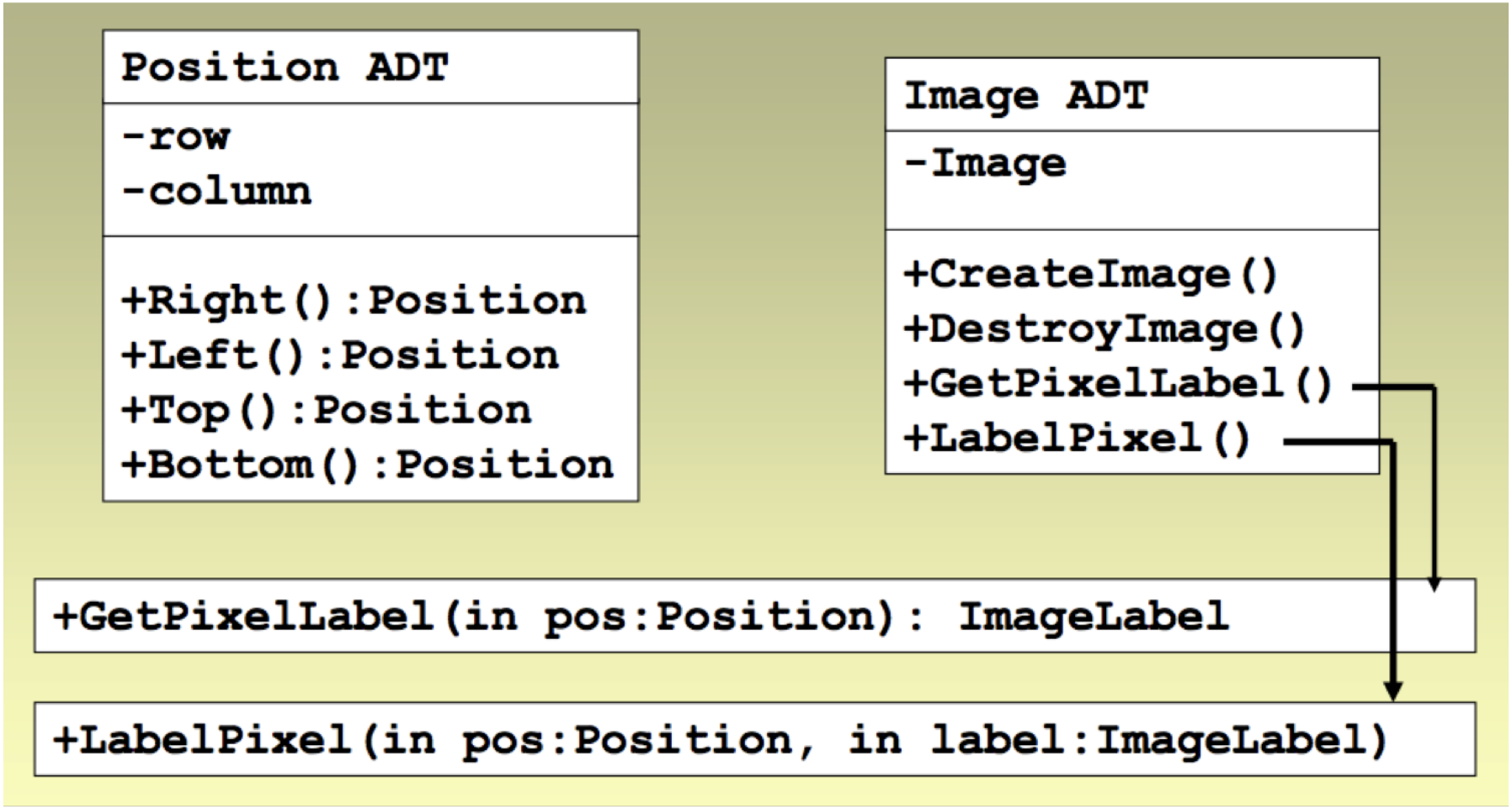
The region growing problem can be described as follows.

Given a two-dimensional array of pixels, and the starting coordinates of a pixel, find all pixels that are similar.





First, lets define two ADT's to describe an Image and a position in the Image.



The similarity can be described as the pixel being considered is the same color as the start pixel.

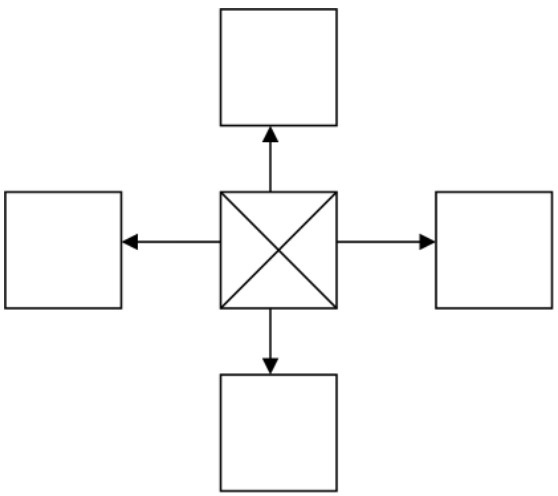
```
Function isSimilar(
    in I:Image,
    in p1:Position,
    in p2:Position):boolean

    if( I.GetPixelLabel(p1) ==
      I.GetPixelLabel(p2) )
            return true
    else
         return false
    endif
endfunction
```

Keeping track of which positions need to be checked for similarity can be done using a stack.

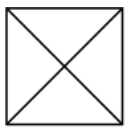
First we define a current pixel we are visiting. Then, we need to check its 4 neighbors.

So, we push all 4 neighbor positions onto a stack,

- if they are similar to current and
- if they are not already in region

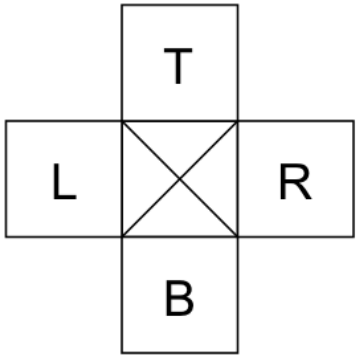


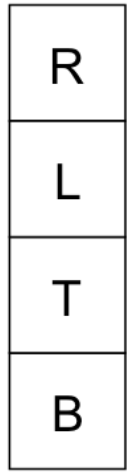
Example



stack (empty)

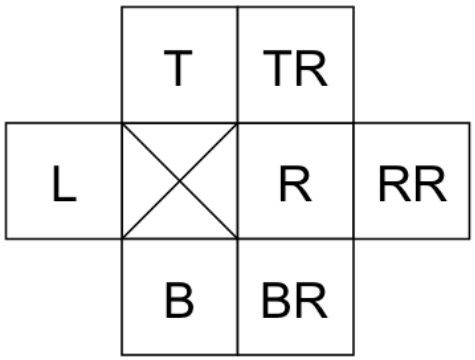
Example

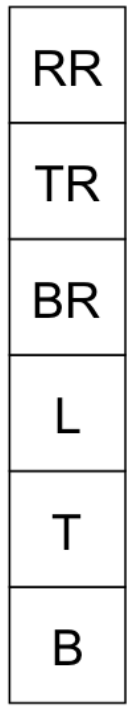




stack (empty)

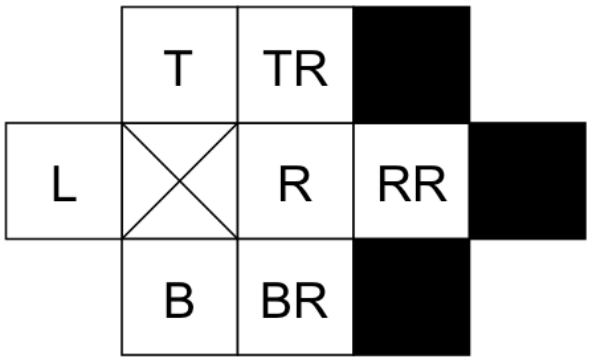
Example

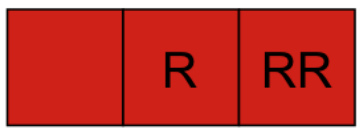


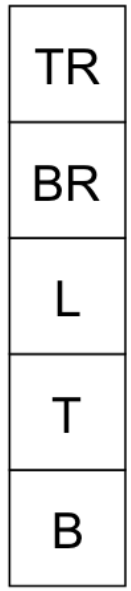


stack (empty)

Example

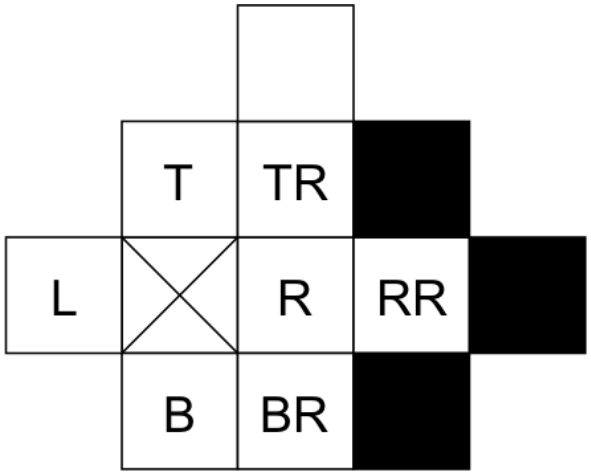


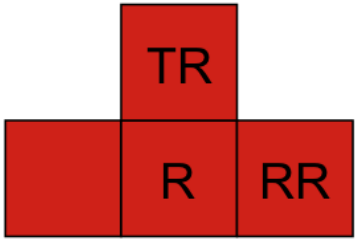


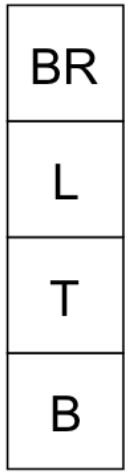


stack (empty)

Example







stack (empty)

This gives the Image Fill algorithm (in pseudocode)

```
function ImageFill(in Image I, in Position s, out Image O)
Stack stack

stack.push(s);

while( not stack.empty() )
    stack.pop(c)
    O.labelPixel(c)

    if( similar(c, c.left()) and !O.GetPixel(c.left()))
        O.LabelPixel(c.left())
        stack.push(c.left())
    endif
    ... similar to right, top, bottom neighbors
endwhile
endfunction
```

Defining an AbstractStack Interface

- create a stack (constructor)
- destroy a stack (destructor)
- empty query (isempty)
- insert (push)
- remove (pop)
- retrieve (top)

See code.

Exercise: Defining Tests for the Stack ADT

See website.

Next Actions and Reminders

- Read CH 7
- No warmup for Fri
- Note: the class meeting on Monday 10/2 is cancelled. A pre-recorded lecture on error handling will be available instead.