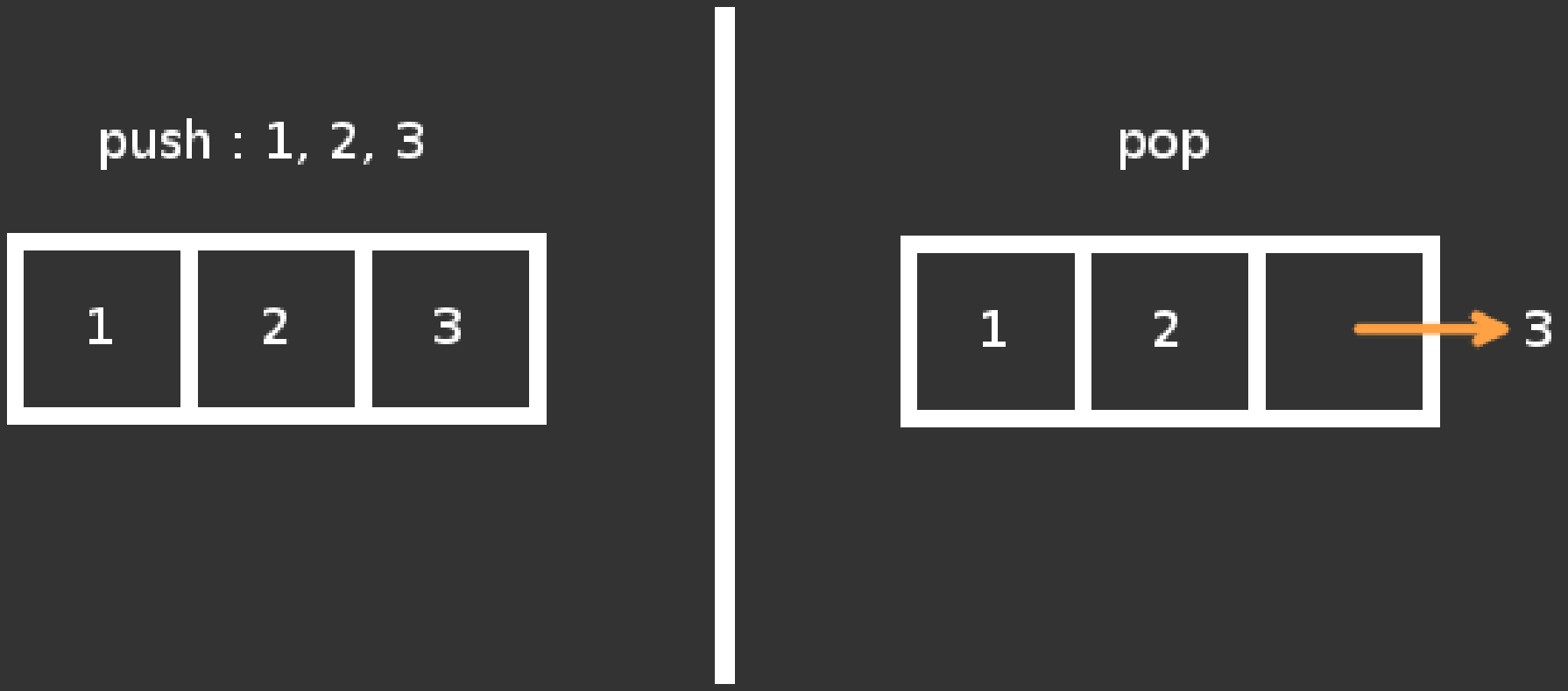
Очередь на 6 стеках

Презентацию подготовил Чистяков Артем.

Что такое стек и очередь?

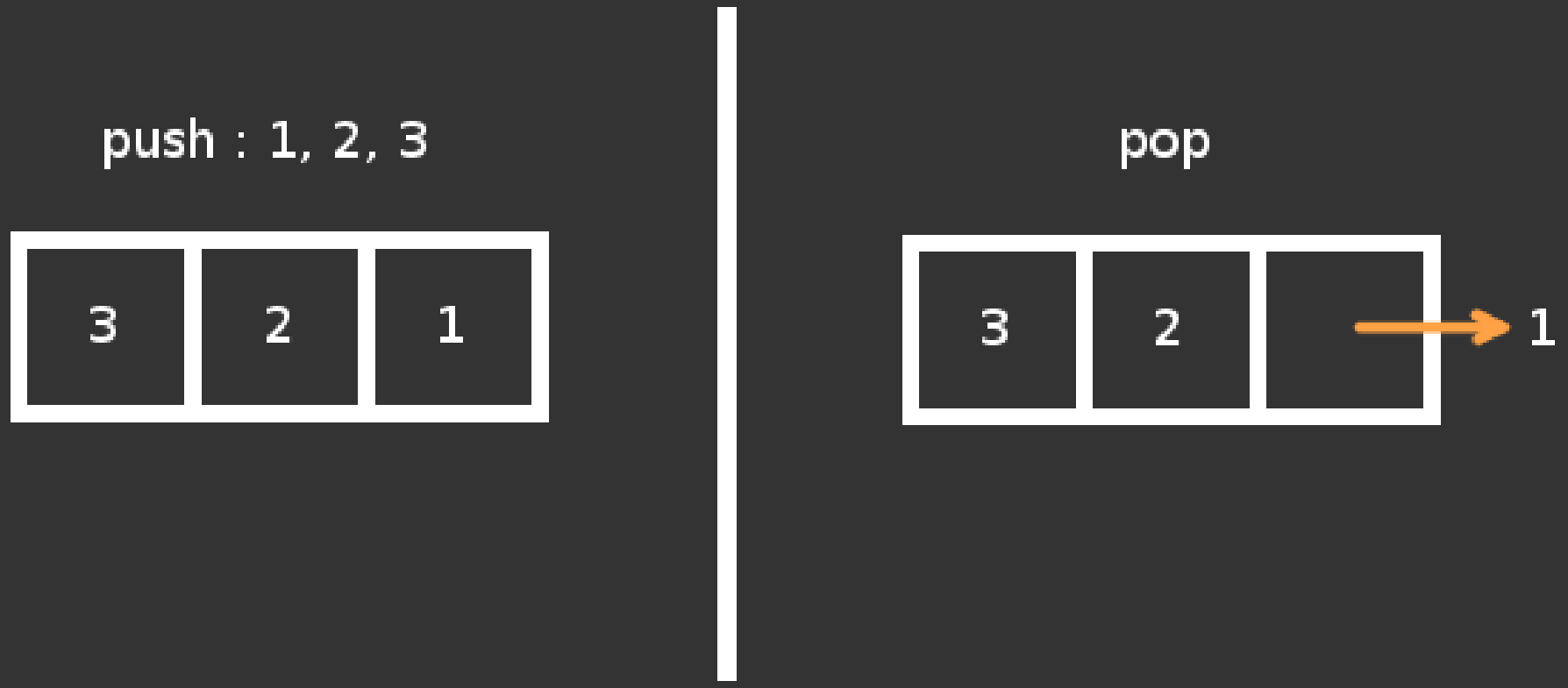
- Стек (stack) – это структура данных, представляющая собой список элементов, организованных по принципу LIFO (last in – first out). Структура поддерживает всего 2 операции: push и pop.

Push добавляет элемент в начало структуры.
Pop возвращает и удаляет элемент из начала этой структуры.


push : 1, 2, 3
1
2
3
pop
1
2
3

- Очередь (queue) – это структура данных, представляющая собой список элементов, организованных по принципу FIFO (first in – first out). Структура поддерживает всего 2 операции: push и pop.

  Push добавляет элемент в конец структуры.
  Pop возвращает и удаляет элемент из начала этой структуры.


push : 1, 2, 3
3
2
1
pop
3
2
1

Постановка задачи

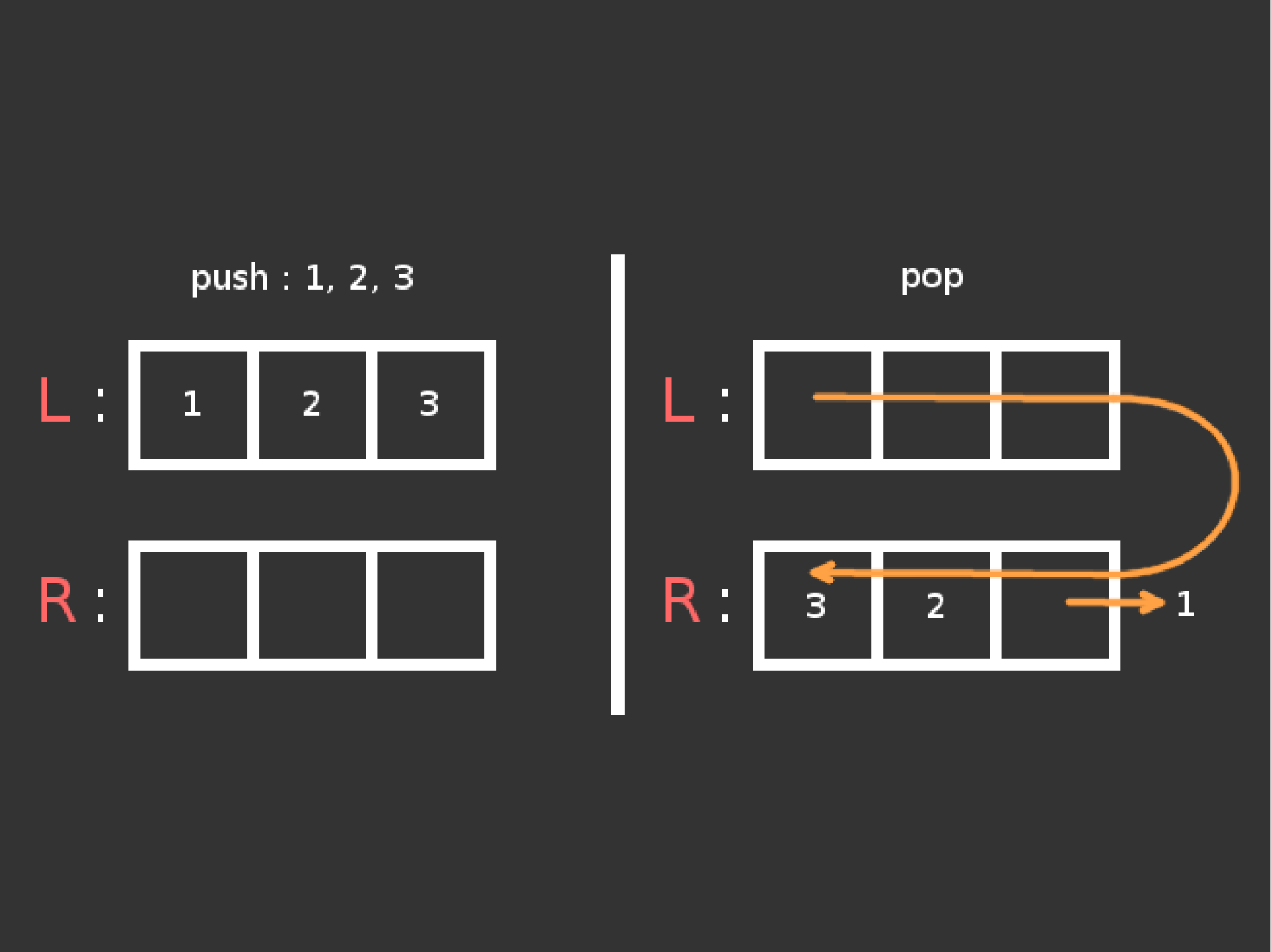
Требуется реализовать очередь, где каждая ее операция будет занимать $O(1)$ времени. Разрешается пользоваться только стеками.

Очередь, реализованную на стеках, относительно легко сделать персистентной.

Реализация на 2 стеках за $O(n)$

Заведем 2 стека: **L** для операций push и **R** для операций pop. Будем всегда добавлять элементы в **L** и возвращать элементы из **R**.

Если во время операции pop стек **R** оказался пустым, то переместим все элементы из стека **L** в **R**, что даст нам “правильный” порядок очереди.


push : 1, 2, 3
L :
1
2
3
R :
pop
L :
R :
3
2
1

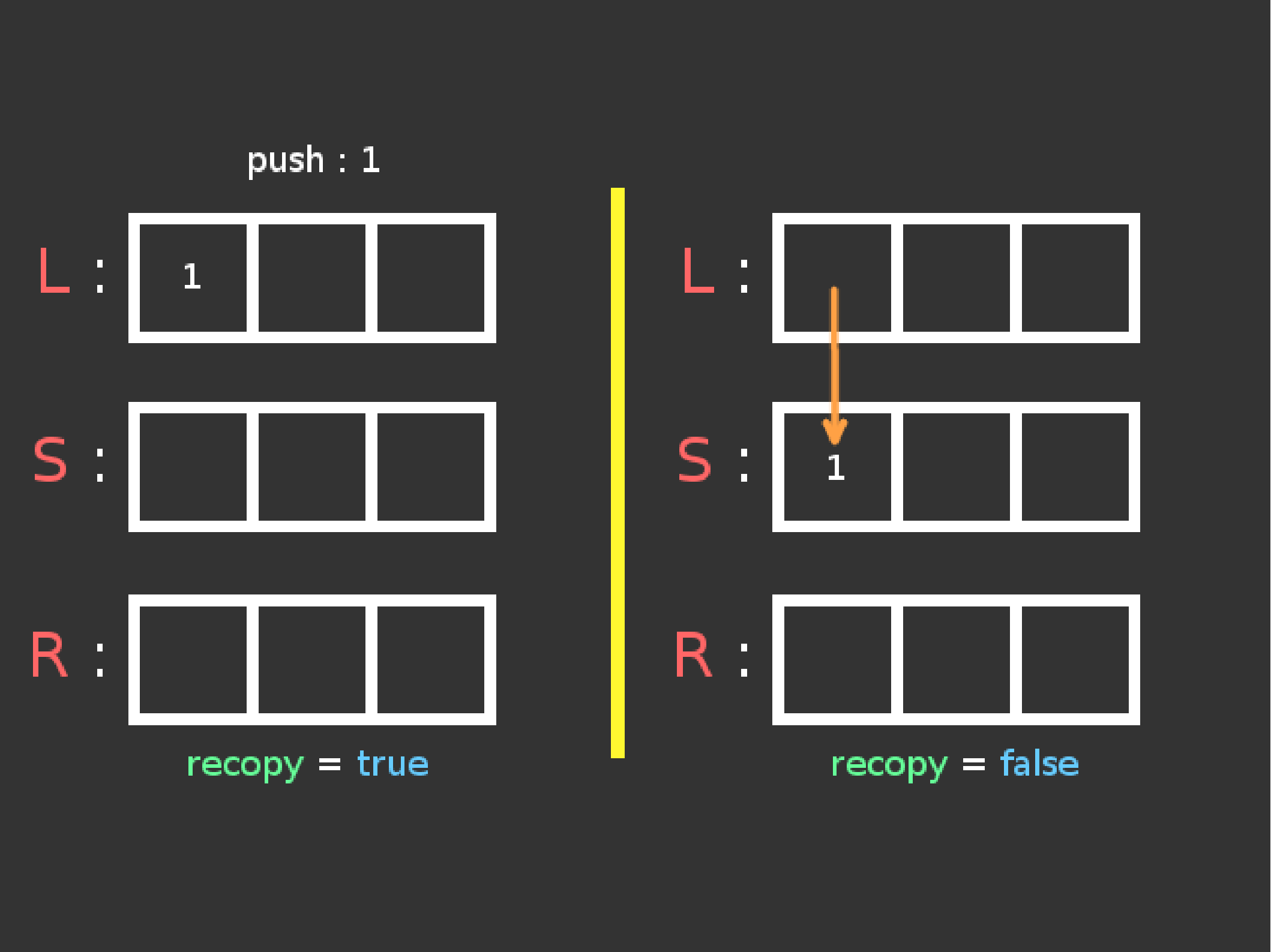
Реализация на 6 стеках за $O(1)$

Аналогично очереди на 2 стеках, заведем стек **L** для операций push и стек **R** для операций pop.

Если во время операции pop стек **R** - пустой, нам понадобится стек **S**, содержащий элементы **L** в правильном для извлечения порядке.

Поэтому, если в какой-то момент алгоритма **L**.size() > **R**.size(), постепенно *переместим* элементы стека **L** в **S**. Будем перемещать константное количество, $O(1)$, элементов на каждой поступающей операции push или pop, пока не опустошим **L**.

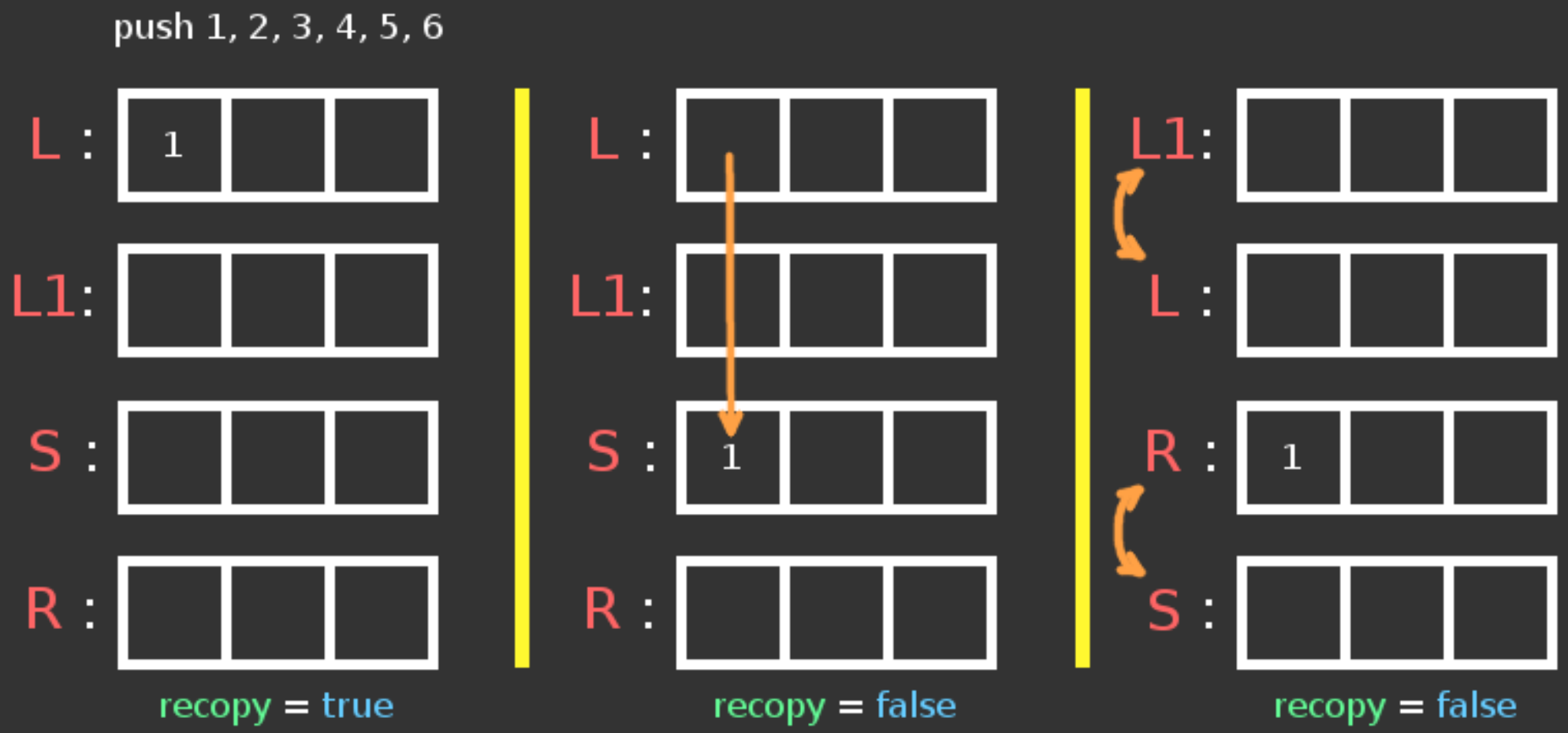
Пусть за такое состояние отвечает переменная recopy.

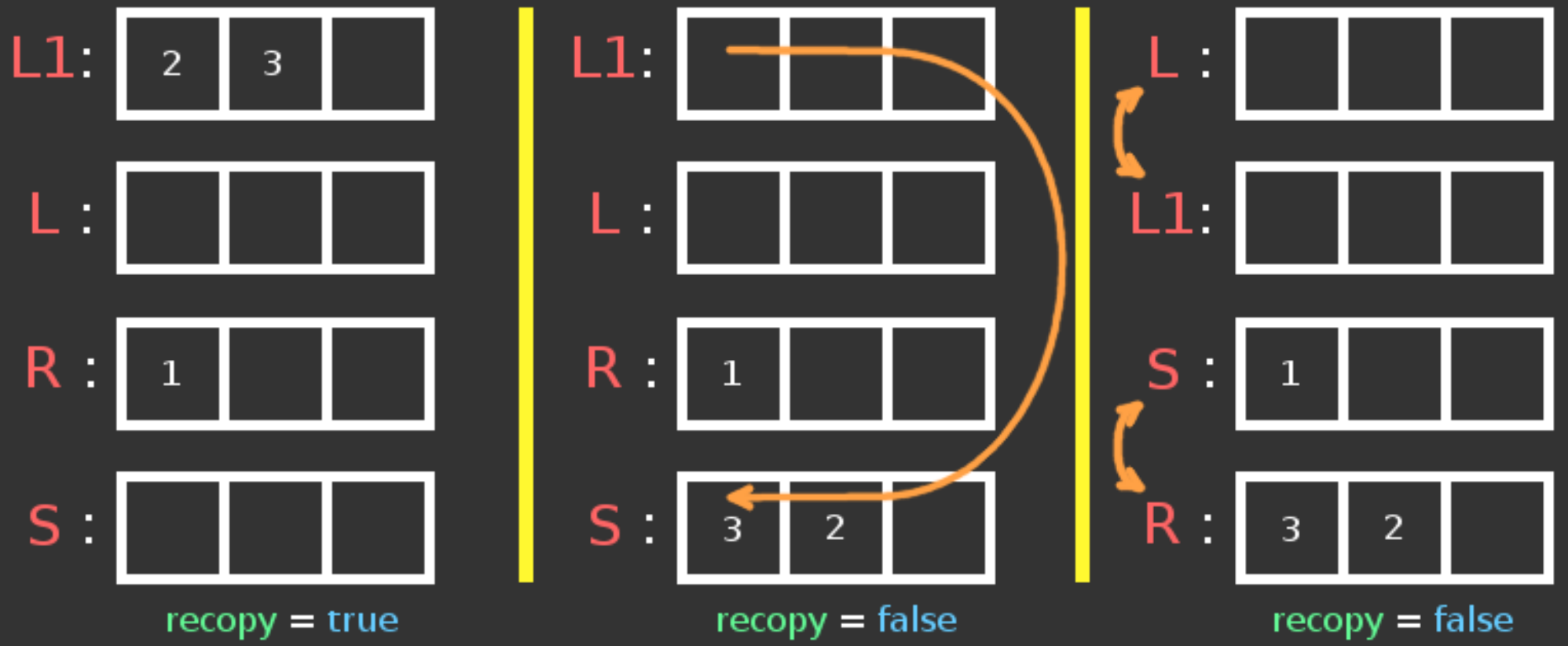

push : 1
L :
1
S :
R :
recopy = true
L :
S :
1
R :
recopy = false

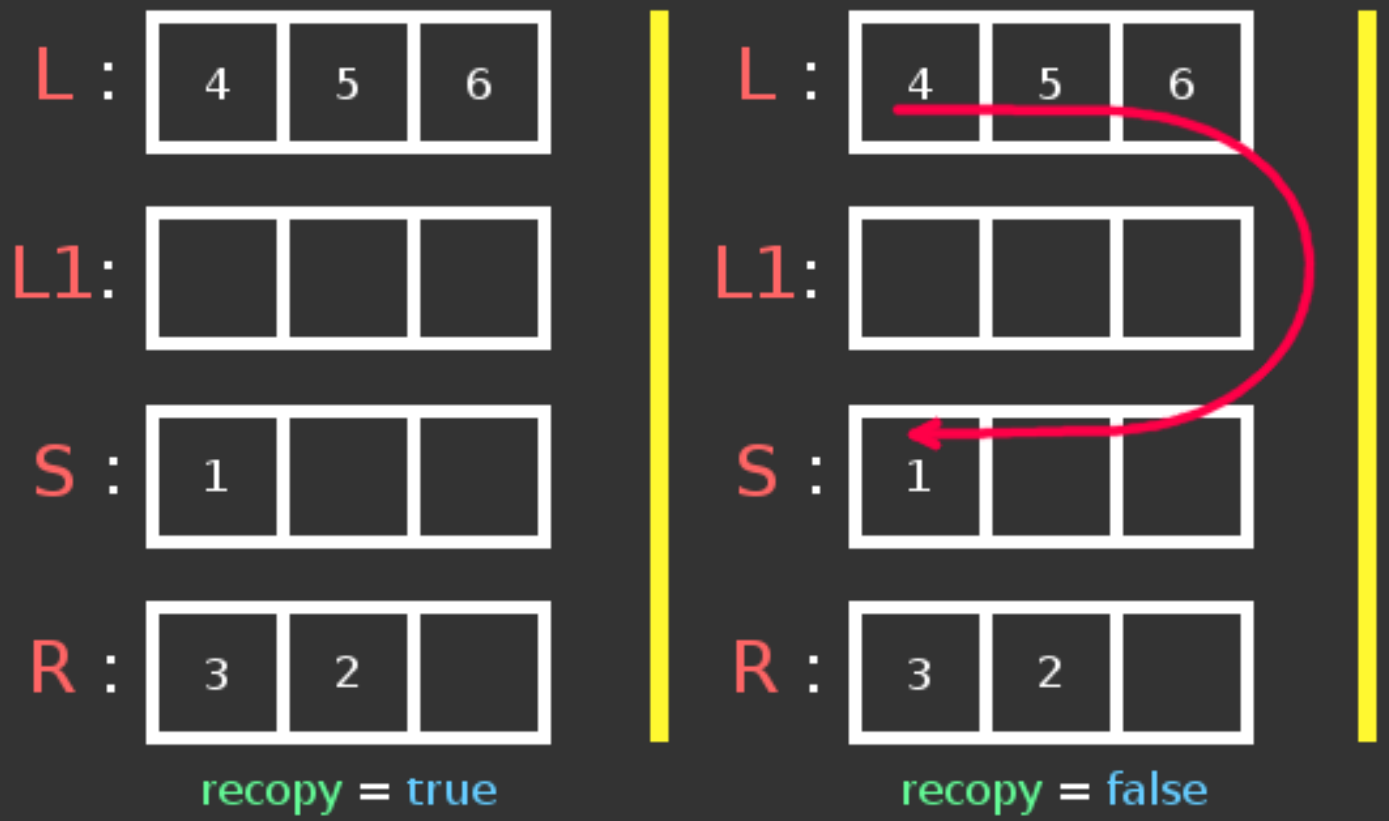
Понятно, что во время перемещения элементов могут поступить операции push и если стек **L** был довольно большим, то он не успел опустошиться, и потерял свою структуру, сложить новые элементы туда нельзя, значит заведем стек **L1**, куда и будем складывать поступающие элементы.

После окончания перемещения поменяем ролями стеки **L** с **L1** и **R** с **S**. Если при операции pop стек **R** оказался пустым, поменяем его со стеком **S**.

Однако, мы получим неприятную вещь: при новой операции push стек **S** может оказаться не пустым, то есть в нем будет лежать начало очереди, и при перемещении в **S** элементов **L** порядок для извлечения будет нарушен.


push 1, 2, 3, 4, 5, 6
L :
1
L1:
S :
R :
recopy = true
L :
L1:
S :
1
R :
recopy = false
L1:
L :
R :
1
S :
recopy = false


L1:
2
3
L :
R :
1
S :
recopy = true
L1:
L :
R :
1
S :
3
2
recopy = false
L :
L1:
S :
1
R :
3
2
recopy = false


L :
4
5
6
L1:
S :
1
R :
3
2
recopy = true
L :
4
5
6
L1:
S :
1
R :
3
2
recopy = false
Error

Для преодоления этой проблемы нам понадобится изменить назначение стеков **R**, **S** и ввести 2 новых: **RC** и **RC1**.

Теперь, при recopy == true, мы будем принудительно извлекать все элементы стека **R** в **S**, далее перемещать элементы **L** в **R** (а не в **S**), а затем обратно элементы стека **S** в **R**. Легко заметить, что теперь стек **R** хранит все нужные елементы в правильном порядке.

*Но что вернуть при операции pop, когда стек **R** занят перемещением?*

Для этого заведем стек **RC** – копию стека **R**, из которого и будем извлекать требуемые элементы, когда **R** занят перемещением.

Поддерживать стек **RC** будем с помощью стека **RC1**, который нужен для дублирования всех элементов, что перемещаются в стек **R**.

По окончанию перемещения стека **S** в стеки **R** и **RC1**, в **RC1** будет находиться правильная последовательность элементов, поэтому для поддержания стека **RC** нам нужно будет поменять его с **RC1**, и не забыть поменять стек **L** с **L1**.

Как было сказано ранее, стек **RC** нужен для извлечения элементов во время занятости стека **R**. Для того чтобы сохранить корректность очереди во время опустошения **RC** (входящая операция pop), нам понадобится переменная toCopy, которая будет обозначать количество не извлеченных элементов в стеке **S** (так как стек **R** = **RC**, то во время перемещения стек **S** станет развернутым **RC**). Извлеченные элементы будут нарастать со дня стека **S**, так что пока toCopy > 0, **S** можно перемещать обратно в стеки **R** и **RC1**.

Если во время операции pop у нас toCopy == 0, это значит, что следует извлечь элемент из стеков **R** и **RC1**.

Осталось только решить проблему с непустым стеком **RC1** (который был **RC** до замены) после завершения перемещения (правильная последовательность уже будет в **R**). Покажем, что мы всегда успеем его опустошить, если будем извлекать из него элемент на каждой операции, когда recopy == false. Также нужно определиться с количеством элементов для перемещения.

Для этого полностью проанализируем алгоритм.

Анализ алгоритма

Пусть на начало перемещения в стеке **R** содержится n элементов, тогда в **L** – n + 1 элемент. Мы корректно можем обработать любое количество операций push, n операций pop и 1 операцию, активирующую перемещение. Таким образом мы гарантированно можем обработать n + 1 операций.

Посмотрим на действия, которые нам предстоят:

- Переместить содержимое **R** в **S** – n действий.
- Переместить содержимое **L** в **R** и **RC1** – n + 1 действий.
- Переместить первые toCopy элементов из **S** в **R** и **RC1**, а остальные выкинуть – n действий.
- Поменять ролями **RC** с **RC1** и **L** с **L1** – 2 действия.

Таким образом, мы получили 3n + 3 действий на n + 1 операций, или 3 ~ $O(1)$ действий на операцию для поддержания корректности очереди. То есть, нам достаточно перемещать по 3 элемента за операции push и pop.

Теперь рассмотрим, как изменились стеки за весь период перемещения. Пусть нам поступило k операций, где x операций pop и k - x операций push. Не забываем про n элементов в **R** и n + 1 элемент в **L**. После перемещения в стеках будет:

- k - x элементов в **L**.
- (k - x) + (n + 1) элементов **R**.

То есть, до следующего перемещения еще n + 2 операций. С другой стороны стек **RC1** содержит n элементов, так что мы успеем его очистить, удаляя по одному элементу, когда recopy == false.

Резюме

Очередь состоит из 6 стеков: **L**, **L1**, **S**, **R**, **RC**, **RC1**. 2 внутренних переменных: recopy, toCopy + переменной copied, показывающей перемещали ли мы элементы из **L** в **R**, чтобы не начать перемещать их из **R** в **S**.

Очередь будет работать в 2 режимах:

- Обычный режим – push в стек **L**; pop из **R**, **RC** и **RC1** для поддержания порядка; проверяем на активацию режима перемещения.
- Режим перемещения – push в **L1**; pop из **RC**, возможно из **R** и **RC1**; совершаем по 3 дополнительных действия: перемещаем **R** в **S**, извлекаем из **L** в **R** и **RC1**, далее перемещаем toCopy элементов из **S** в **R** и **RC1**; проверяем на выключение режима перемещения; меняем ролями **RC** с **RC1** и **L** с **L1**.

- После любой операции в обычном режиме проверяем на активацию режима перемещения, recopy = **L**.size() > **R**.size(), если это так, то toCopy = **R**.size(), copied = false. Совершаем действия для перемещения.

- После любой операции в режиме перемещения следуют проверка на выключение режима, recopy = **S**.empty(), и при выключении, меняем ролями **RC** с **RC1** и **L** с **L1**.

Псевдокод

```
function push(x : T):
  if !recopy
    L.push(x)
    if RC1.size > 0
      RC1.pop()
    checkRecopy()
  else
    L1.push(x)
    checkCommon()
```

```
T pop():
  if !recopy
    T x = R.pop()
    RC.pop()
    if RC1.size > 0
      RC1.pop()
    checkRecopy()
    return x
  else
    T x = RC.pop()
    if toCopy > 0
      toCopy -= 1
    else
      R.pop()
      RC1.pop()
    checkCommon()
    return x
```

```
function checkRecopy():
  recopy = L.size > R.size
  if recopy
    toCopy = R.size
    copied = false
    checkCommon()
```

```
function checkCommon():
  performRecopy()
  // Если мы не все переместили,
     то у нас не пуст стек S
  recopy = S.size > 0
```

```
function performRecopy():
  // Нам достаточно 3 операций на вызов
  int toDo = 3
  // Пытаемся переместить R в S
  while !copied and toDo > 0 and R.size > 0
    S.push(R.pop())
    toDo -= 1
  // Пытаемся переместить L в R и RC1
  while toDo > 0 and L.size > 0
    copied = true
    T x = L.pop()
    R.push(x)
    RC1.push(x)
    toDo -= 1
  // Пытаемся переместить S в R и RC1 с учетом toCopy
  while toDo > 0 and S.size > 0
    T x = S.pop()
    if toCopy > 0
      R.push(x)
      RC1.push(x)
      toCopy -= 1
    toDo -= 1
  // Если все перемещено, то меняем роли L, L1 и RC, RC1
  if S.size == 0
    swap(L, L1)
    swap(RC, RC1)
```