Замена системной рекурсии стеком. Разбор формул при помощи стека - без приоритета и с приоритетом. Бинарные деревья. Бинарные деревья поиска.

Осенний семестр, 2020 год

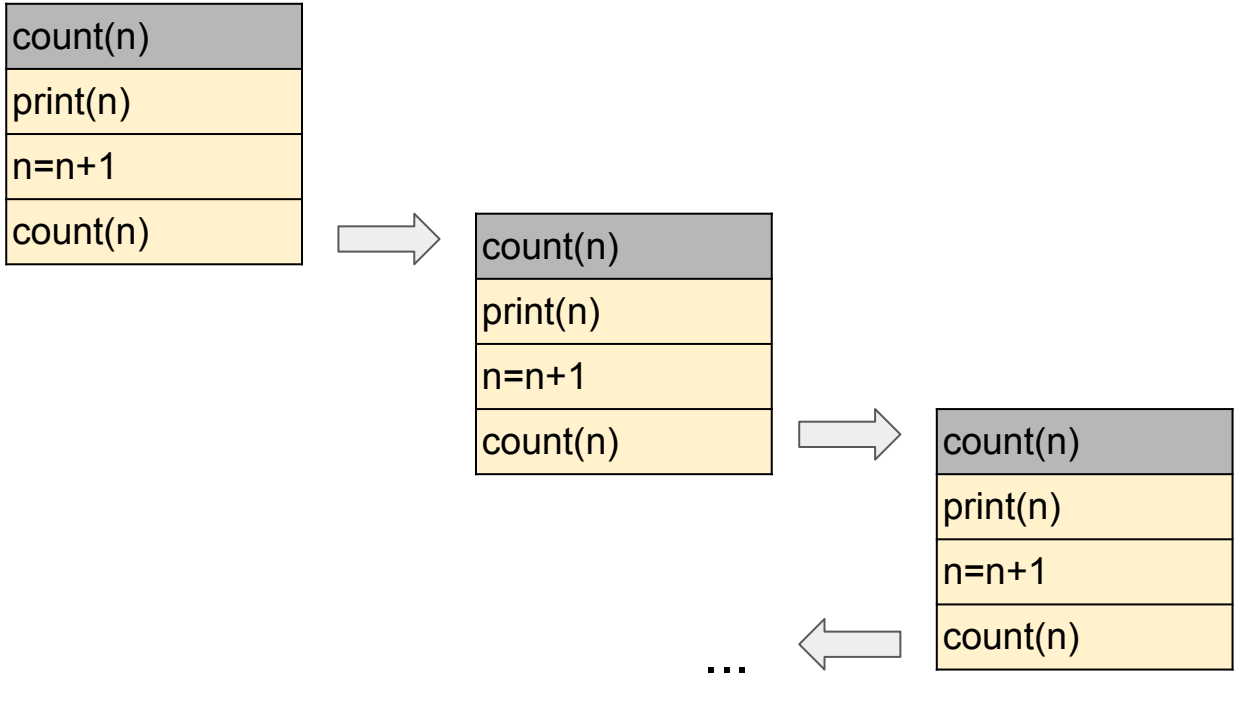
Дианкин Игорь

Замена системной рекурсии стеком.

- Рекурсия крайне удобна при написании кода. Обычно код становится более читаемы и в целом менее громоздким.
- К сожалению, из-за несовершенства мира, а особенно ограниченности вычислительных машин, рекурсия может начинать поглощать неимоверно большое количество памяти.
- От рекурсии принято избавляться, где это возможно. В некоторых случаях, компиляторы автоматически замеют рекурсию циклом, конкретно в случаях хвостовой рекурсии.
- Хвостовая рекурсия - вид рекурсии при котором рекурсивный вызов является последней операцией перед возвратом из функции.

Пример хвостовой рекурсии

```
def count(n):
    print(n)
    n += 1
    count(n)
```



Output:

1
2
3
4
5
6
7
...

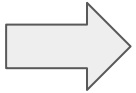
Что такое системный стек (call stack)

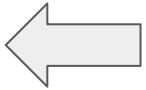
```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```





Call Stack

Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```

Call Stack

main(), A = 5

Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```

Call Stack

help_1(), A = 5
main(), A = 5

Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```

Call Stack

help_2(), A = 5
help_1(), A = 5
main(), A = 5

Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```

Call Stack

help_2(), A = 5, C = 6
help_1(), A = 5
main(), A = 5

Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```

Call Stack

help_2(), A = 5, C = 6
help_1(), A = 5
main(), A = 5

Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```

Call Stack

help_1(), A = 5, B = 6
main(), A = 5

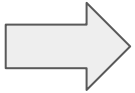
Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```



Call Stack

main(), A = 5, B = 6

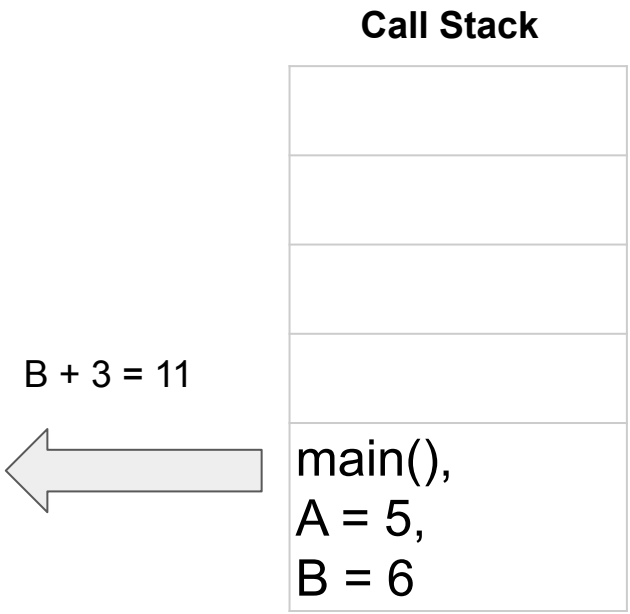
Что такое системный стек (call stack)

```
def main(A):
    B = help_1(A)
    return B + 3

def help_1(B):
    C = help_2(B)
    return C + 2

def help_2(C):
    return C + 1

main(5)
```



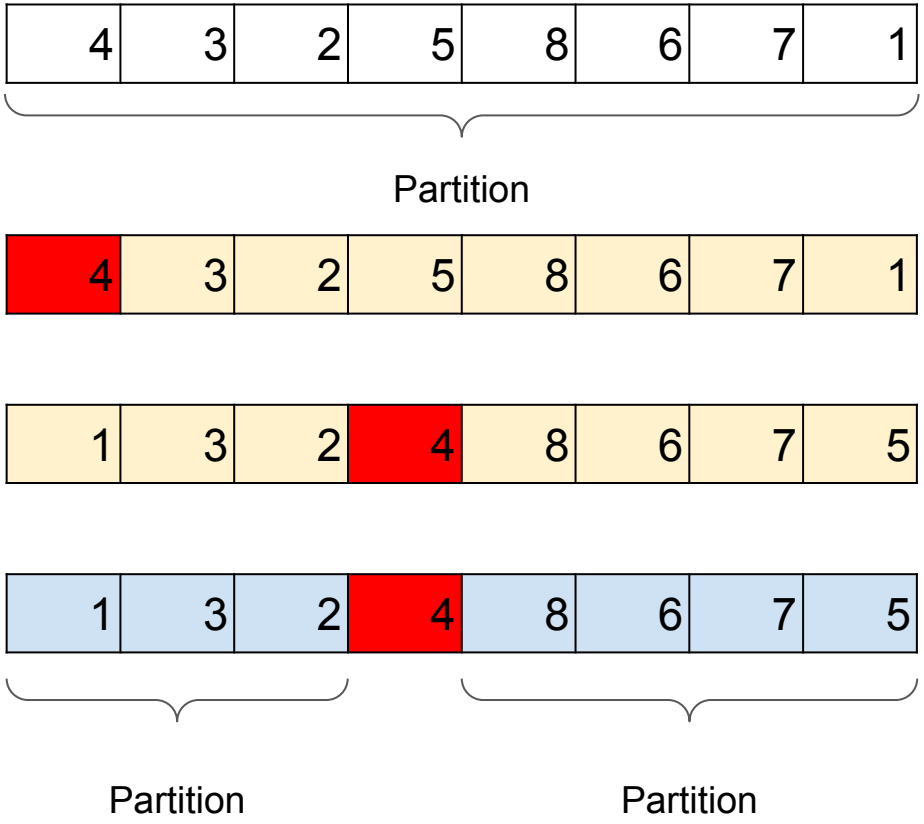
Что это значит? На примере QSort.

- Рекурсия крайне удобна при написании кода. Обычно код становится более читаемы и в целом менее громоздким.
- К сожалению, из-за несовершенства мира, а особенно ограниченности вычислительных машин, рекурсия может начинать поглощать неимоверно большое количество памяти.
- От рекурсии принято избавляться, где это возможно. В некоторых случаях, компиляторы автоматически замеют рекурсию стеком, конкретно в случаях хвостовой рекурсии. В остальных случаях одного подхода нет, и задача становится индивидуальной.
- Хвостовая рекурсия - вид рекурсии при котором рекурсивный вызов является последней операцией перед возвратом из функции.

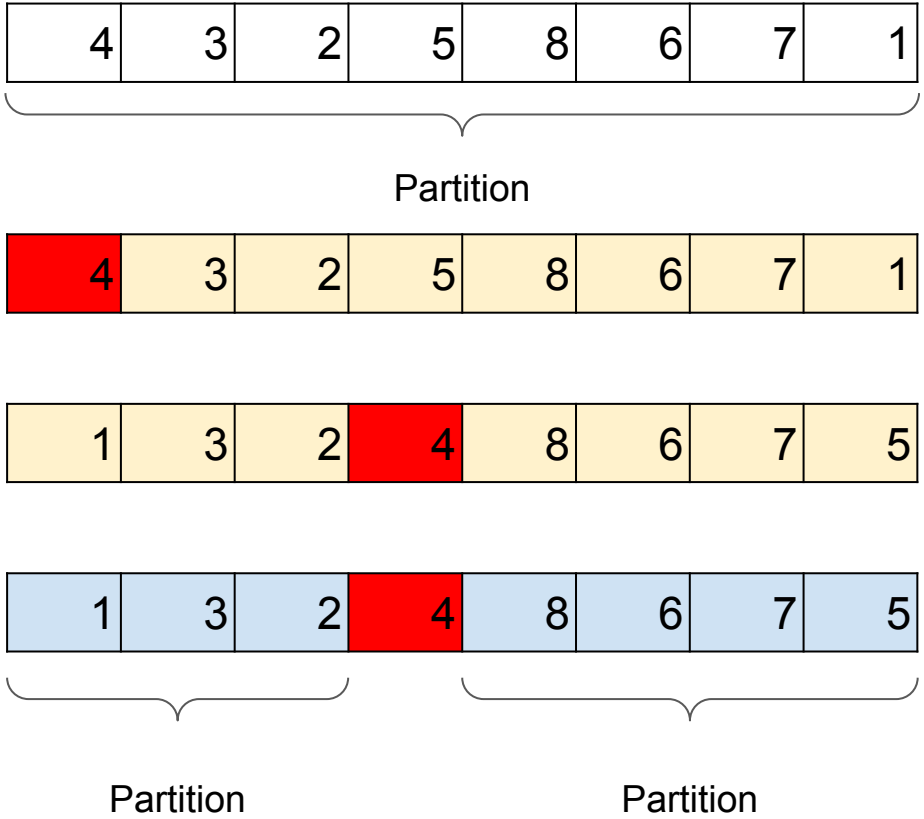
Замена системной рекурсии стеком.

- Пример для QSort

- Рекурсия здесь интуитивно подходит для реализации принципа “Разделяй и Влавствуй”



Как обойти рекурсию и использовать стек?



Как обойти рекурсию и использовать стек?

Инициация

Индексы ячеек

0	1	2	3	4	5	6	7	8

4	3	2	5	8	6	7	1

Push([0, 8])

Operation Stack

0, 8

Как обойти рекурсию и использовать стек?

Цикл пока стек не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8
4	3	2	5	8	6	7	1	

Partition([0, 8])

Operation Stack

Как обойти рекурсию и использовать стек?

Цикл пока стек не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8
4	3	2	5	8	6	7	1	
1	3	2	4	8	6	7	5	
1	3	2	4	8	6	7	5	

Push([0, 3])

Operation Stack

Как обойти рекурсию и использовать стек?

Цикл пока стек не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8
4	3	2	5	8	6	7	1	
1	3	2	4	8	6	7	5	
1	3	2	4	8	6	7	5	

Push([0, 3])

Push([4, 8])

Operation Stack

4, 8
0, 3

Как обойти рекурсию и использовать стек?

Стек не пуст, следующий цикл

Индексы ячеек

0	1	2	3	4	5	6	7	8
1	3	2	4	8	6	7	5	

Partition([4, 8])

Operation Stack

4, 8
0, 3

Как обойти рекурсию и использовать стек?

Стек не пуст, следующий цикл

Индексы ячеек

0	1	2	3	4	5	6	7	8
1	3	2	4	5	6	7	8	

Partition([4, 8])

Operation Stack

0, 3

Как обойти рекурсию и использовать стек?

Стек не пуст, следующий цикл

Индексы ячеек

0	1	2	3	4	5	6	7	8
1	3	2	4	5	6	7	8	

Push([4, 7])

Operation Stack

4, 7
0, 3

Как обойти рекурсию и использовать стек?

Стек не опять не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8
1	3	2	4	5	6	7	8	

Partition([4, 7])

Operation Stack

4, 7
0, 3

Как обойти рекурсию и использовать стек?

Стек не опять не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8
1	3	2	4	5	6	7	8	

Тут и так отсортировано,
поэтому подробно
описывать не будем

Operation Stack

0, 3

Как обойти рекурсию и использовать стек?

Стек все еще не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8
1	3	2	4	5	6	7	8	

Partition([0, 3])

Operation Stack

0, 3

Как обойти рекурсию и использовать стек?

Стек все еще не пуст

Индексы ячеек

0	1	2	3	4	5	6	7	8

1	2	3	4	5	6	7	8

Partition([0, 3])

Operation Stack

Как обойти рекурсию и использовать стек?

Теперь стек пуст.

Индексы ячеек

0	1	2	3	4	5	6	7	8

1	2	3	4	5	6	7	8

Как и в обычной быстрой сортировке, помним что массив из одного элемента отсортирован.

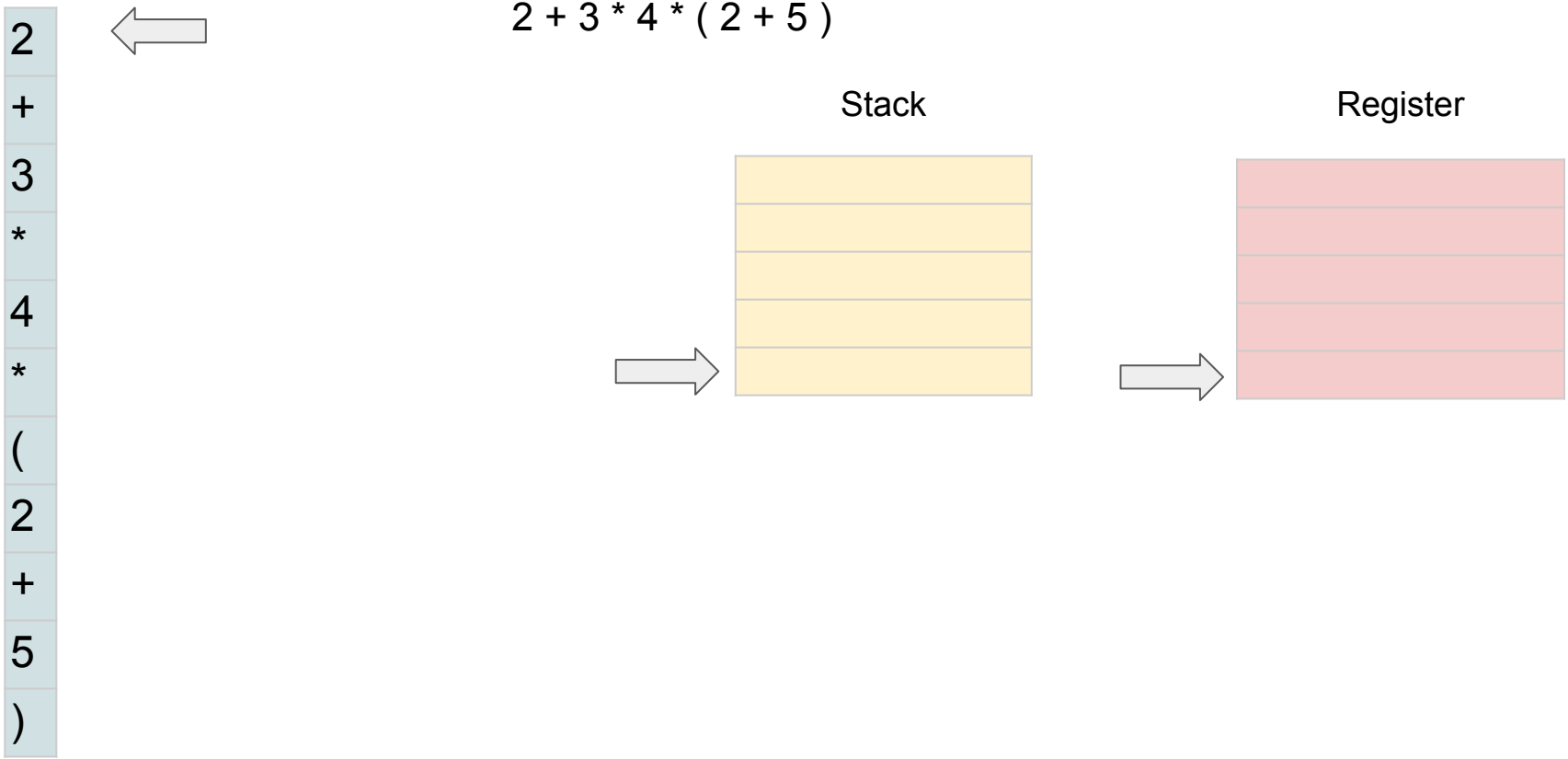
Operation Stack

Разбор формул

- Разбор формулы - задача о том, как правильно, имея математическое выражение в формате строки, перевести его в понятный для совершения операций машиной формат.
- Операция - простейшая функция (+, -, *, /, возведение в степень)
- Операнд - то над чем совершается операция (число, переменная, структура данных)

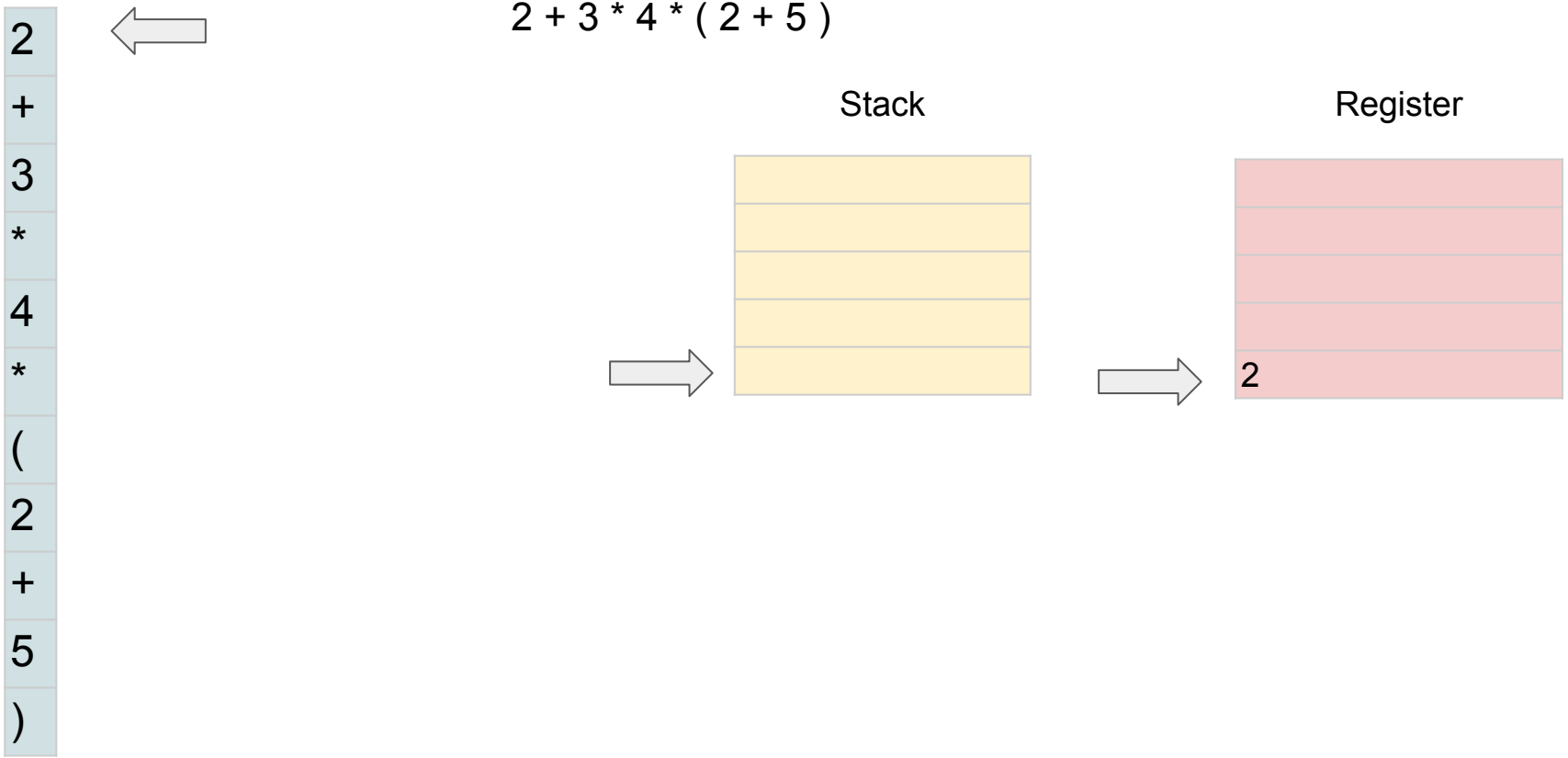
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



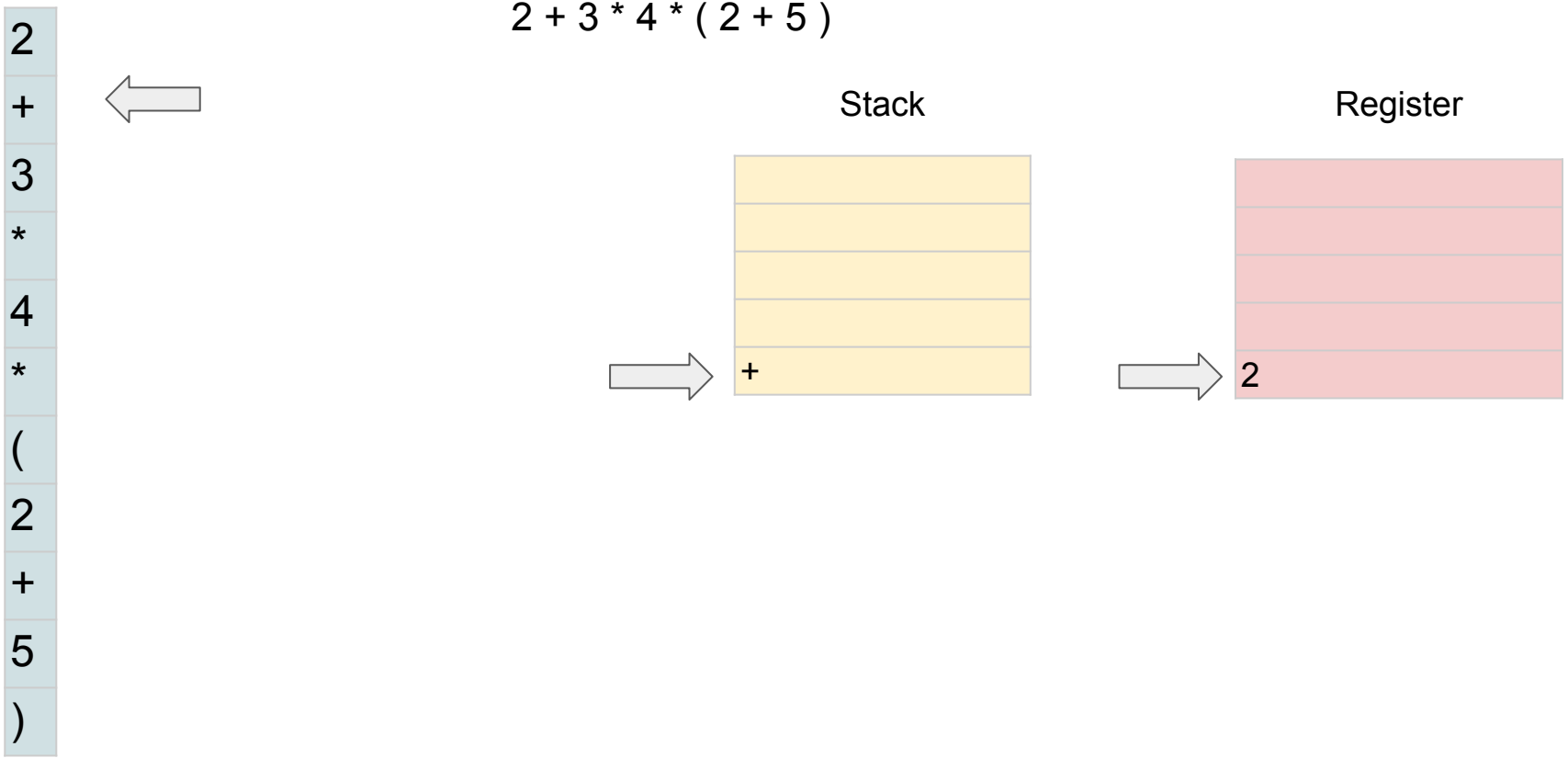
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



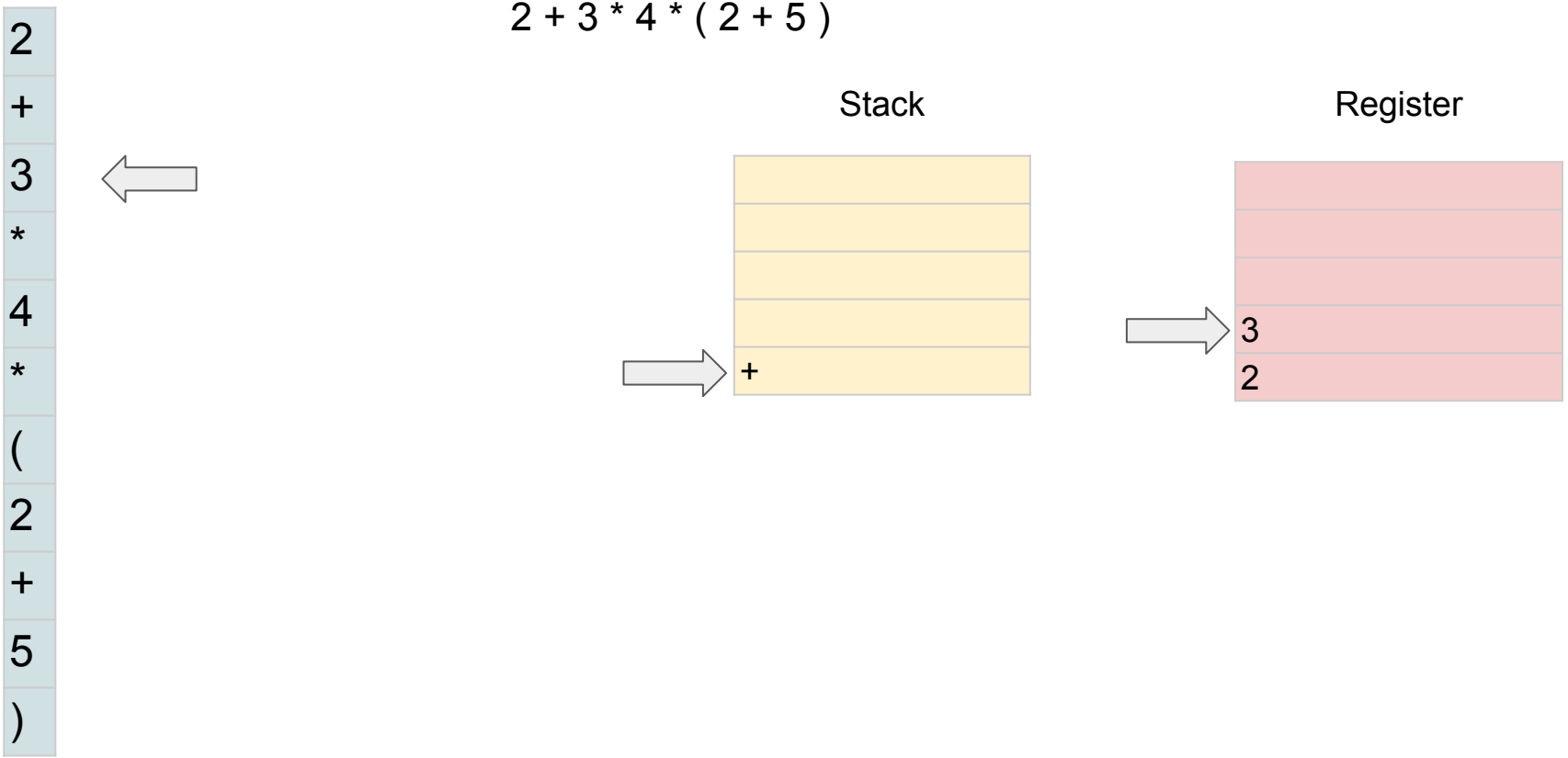
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



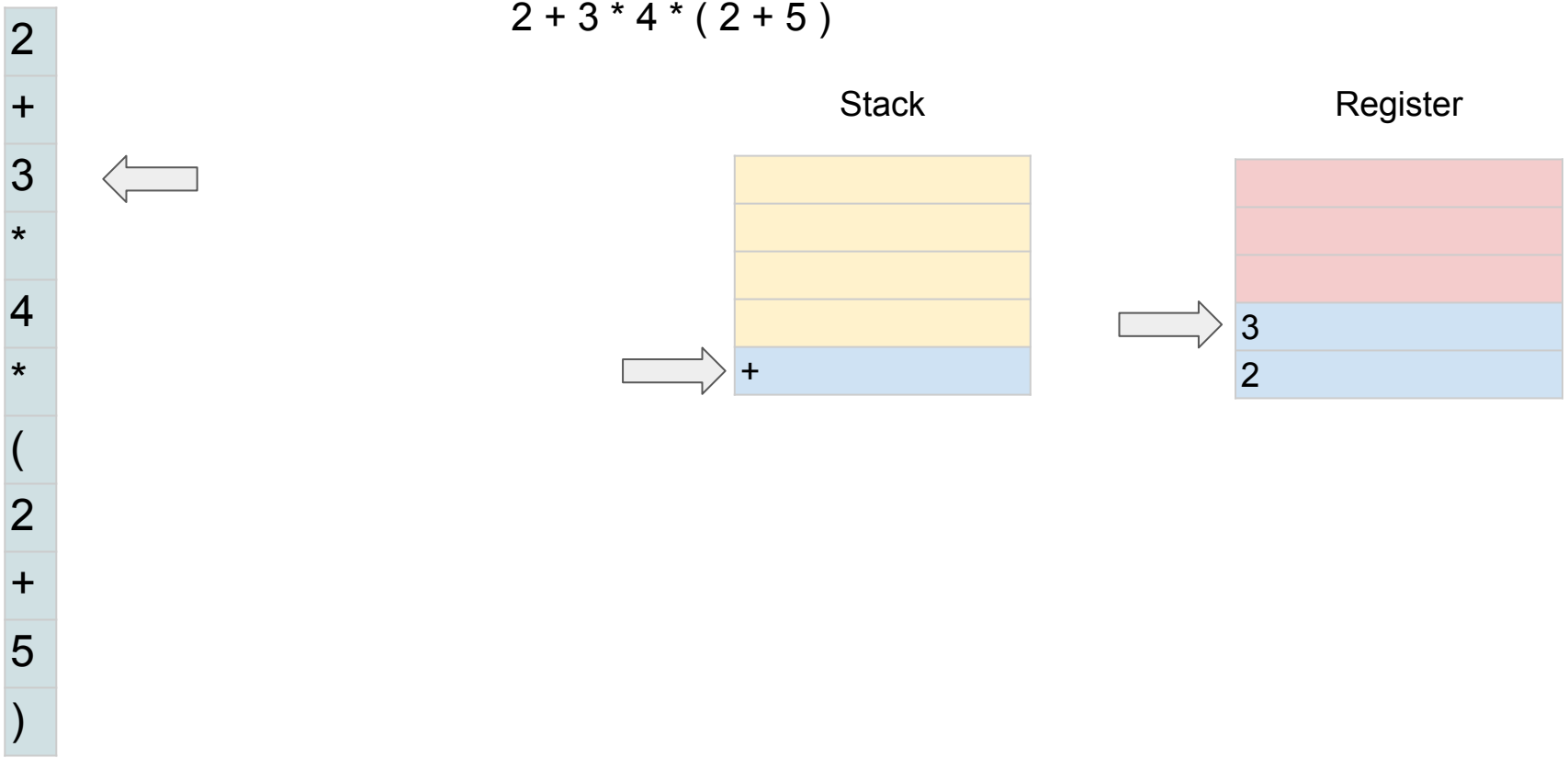
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



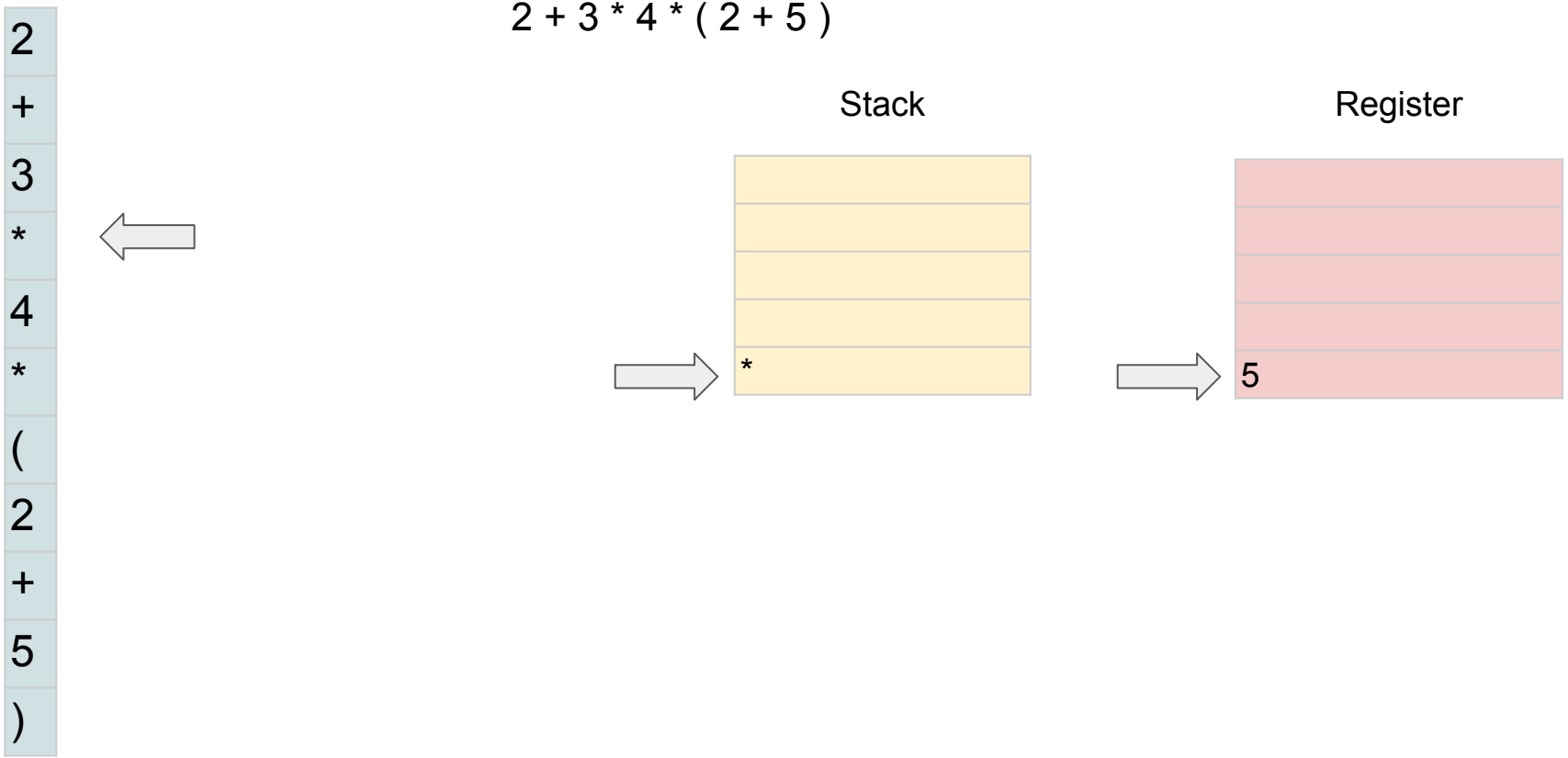
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



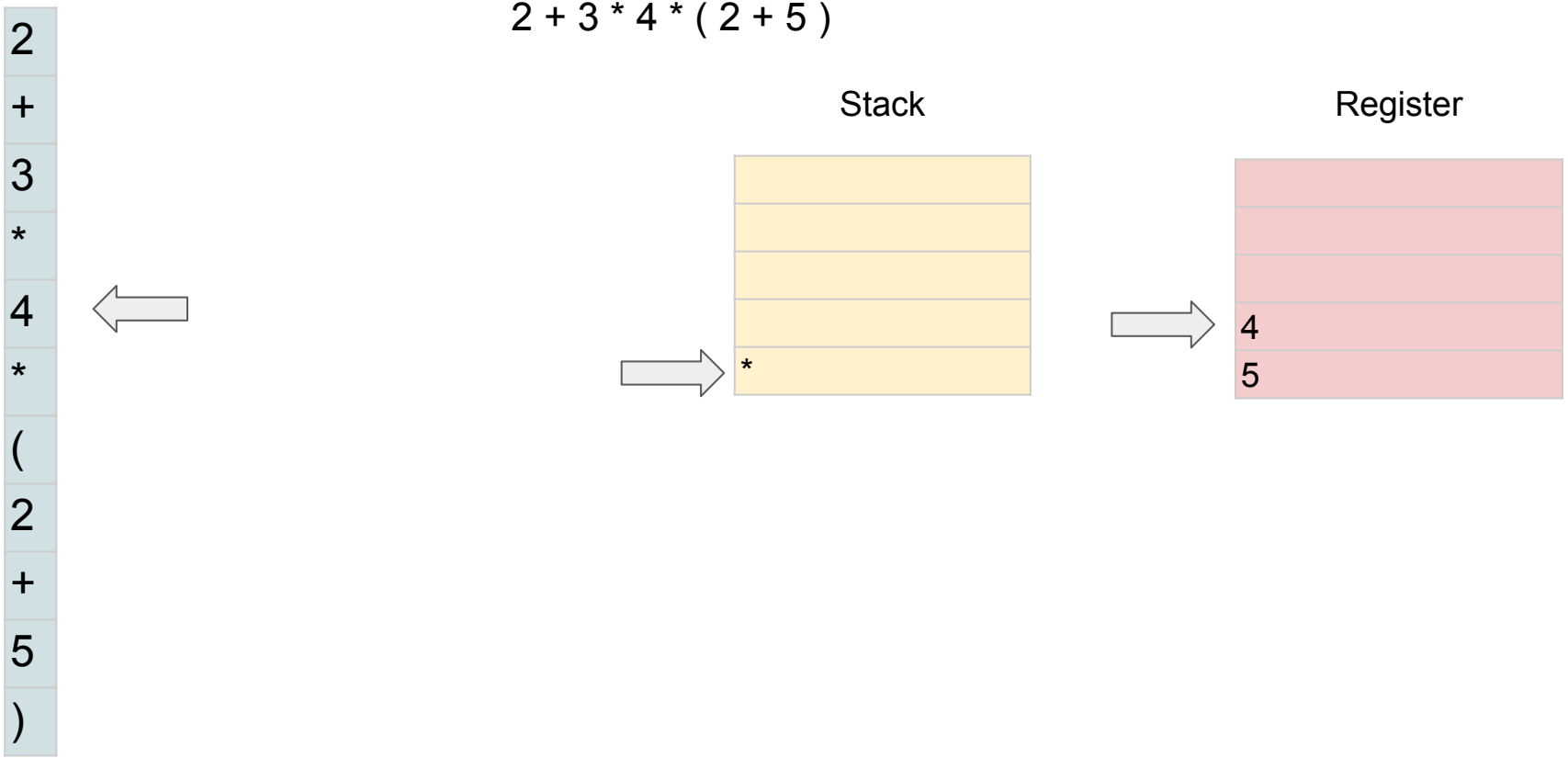
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



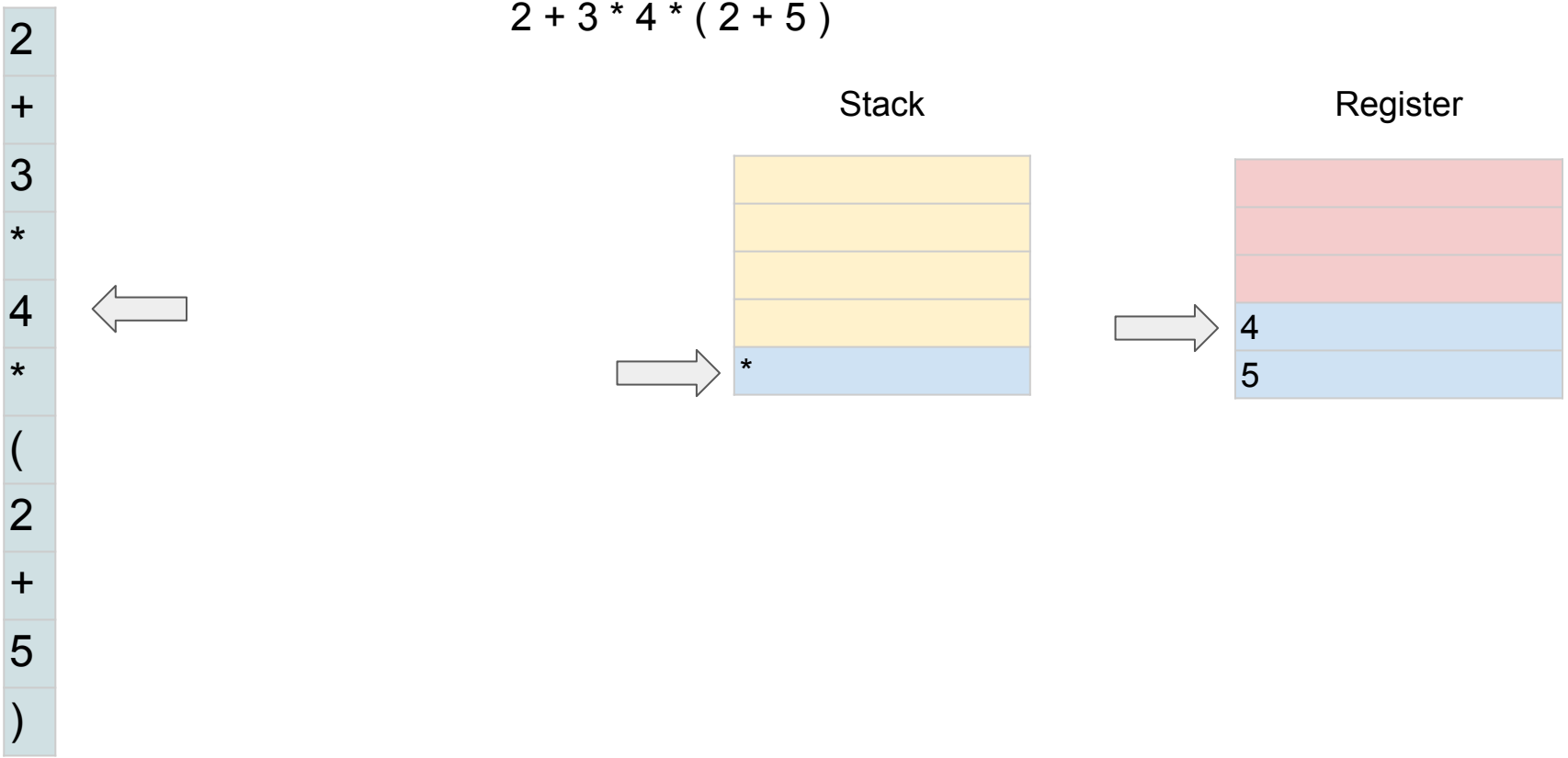
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



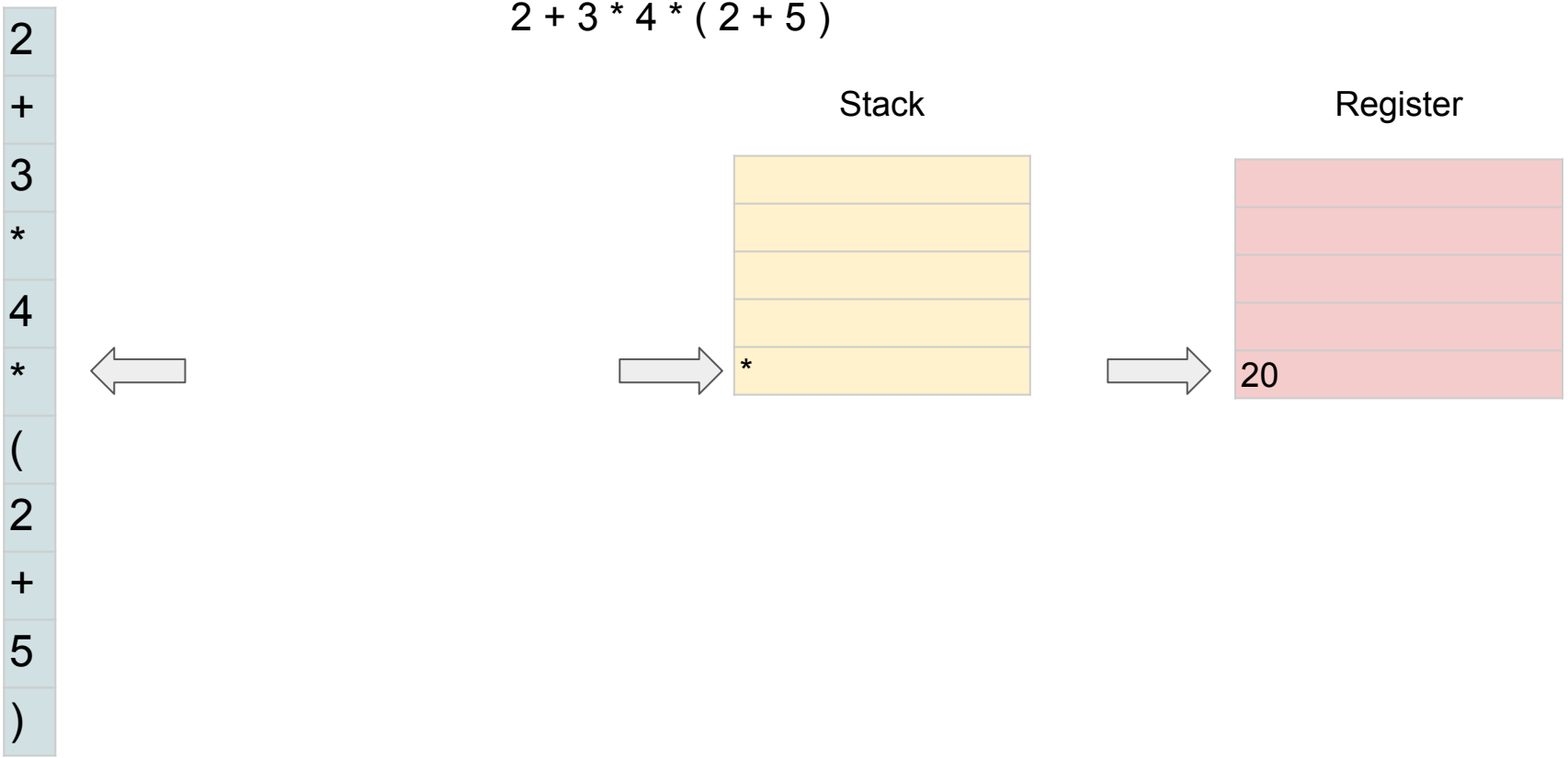
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



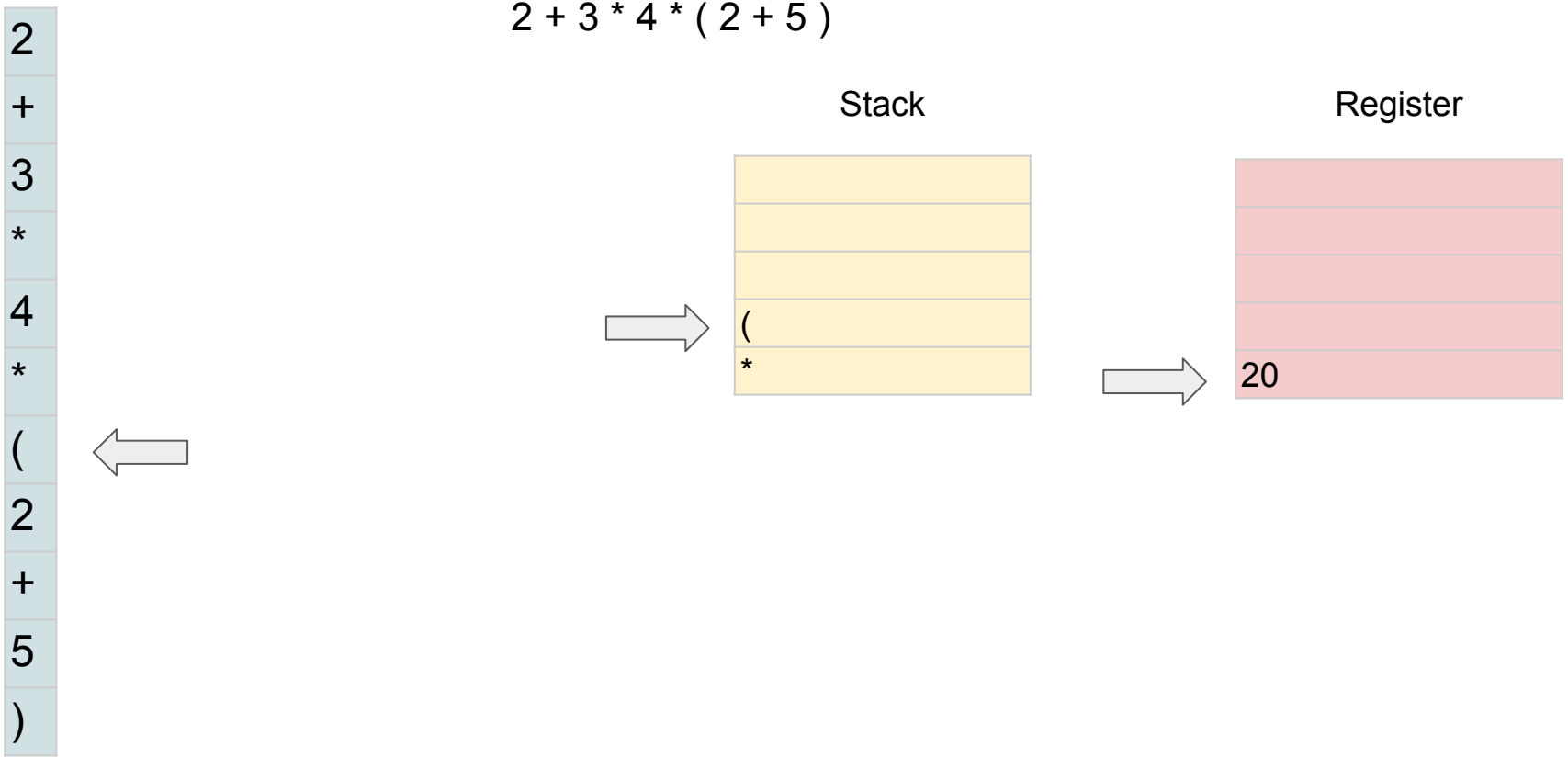
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



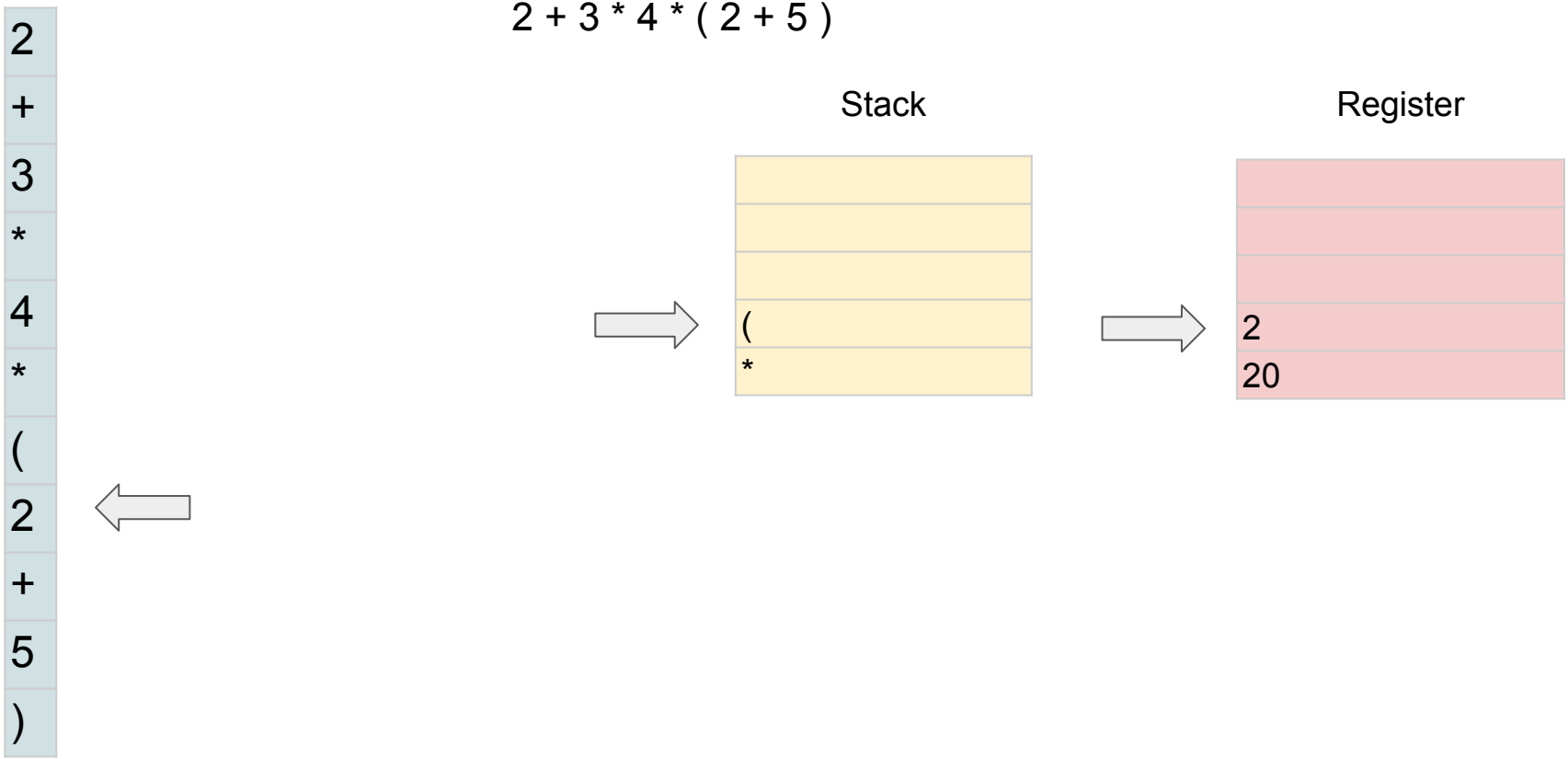
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



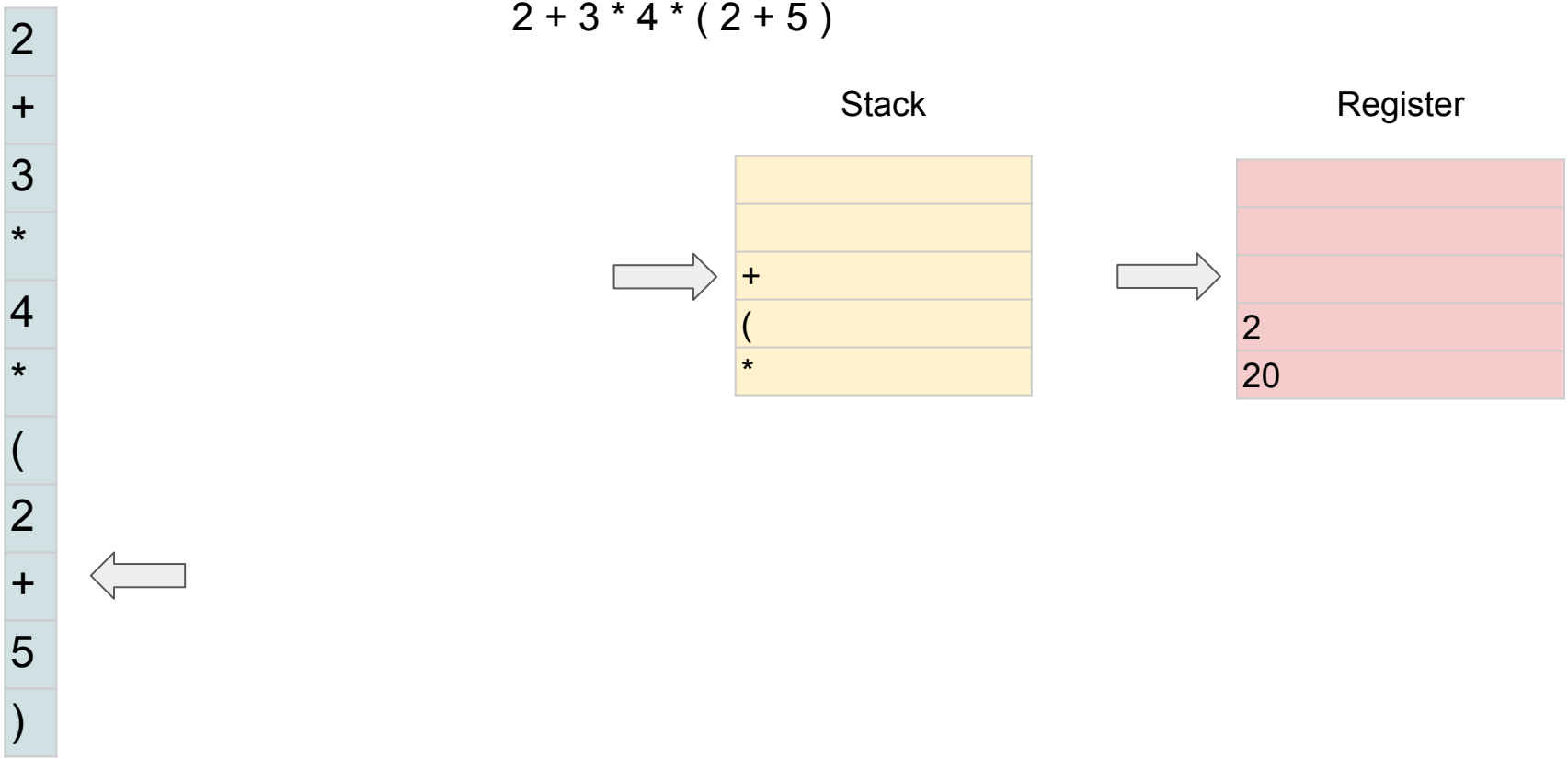
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



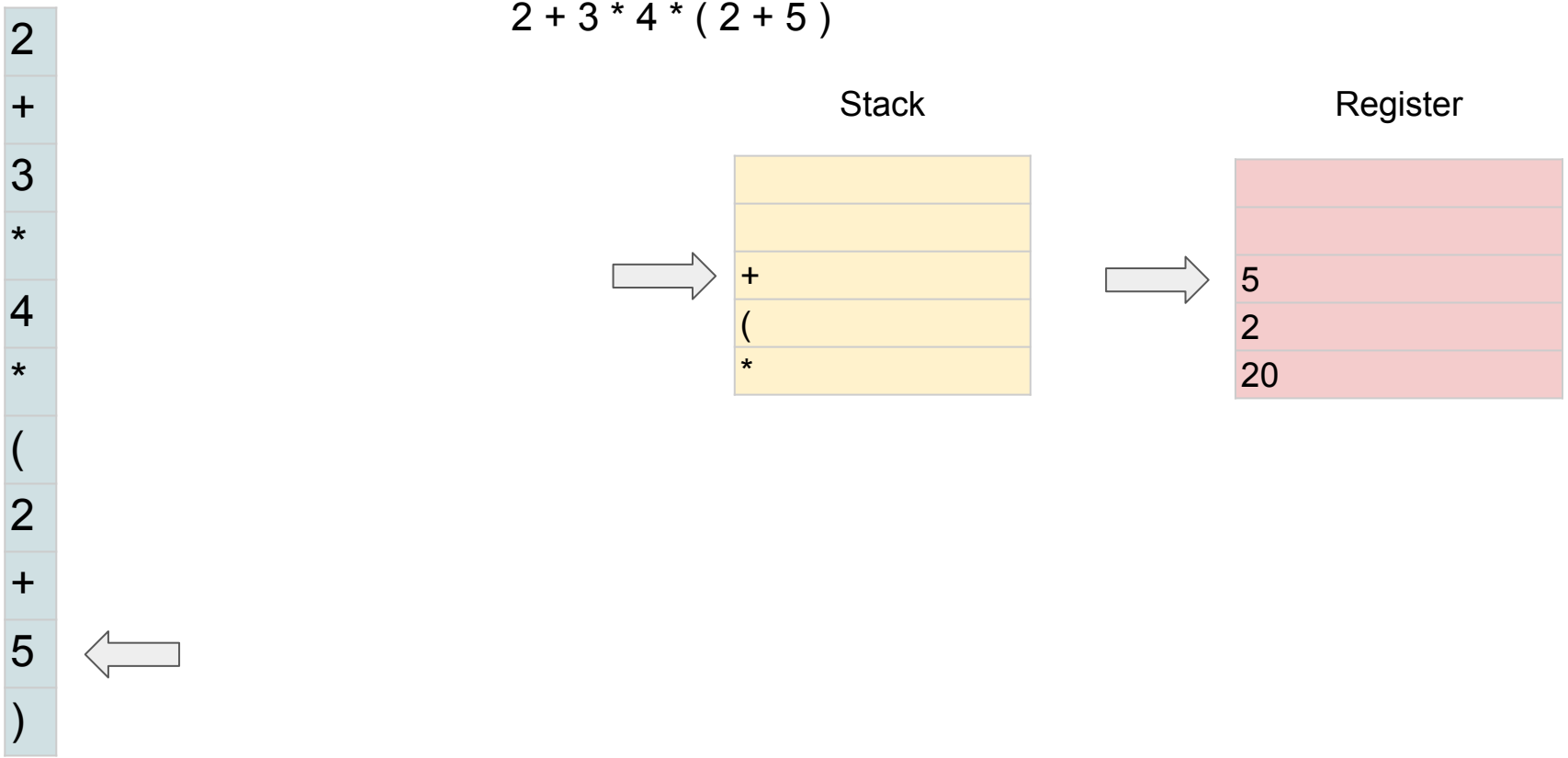
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



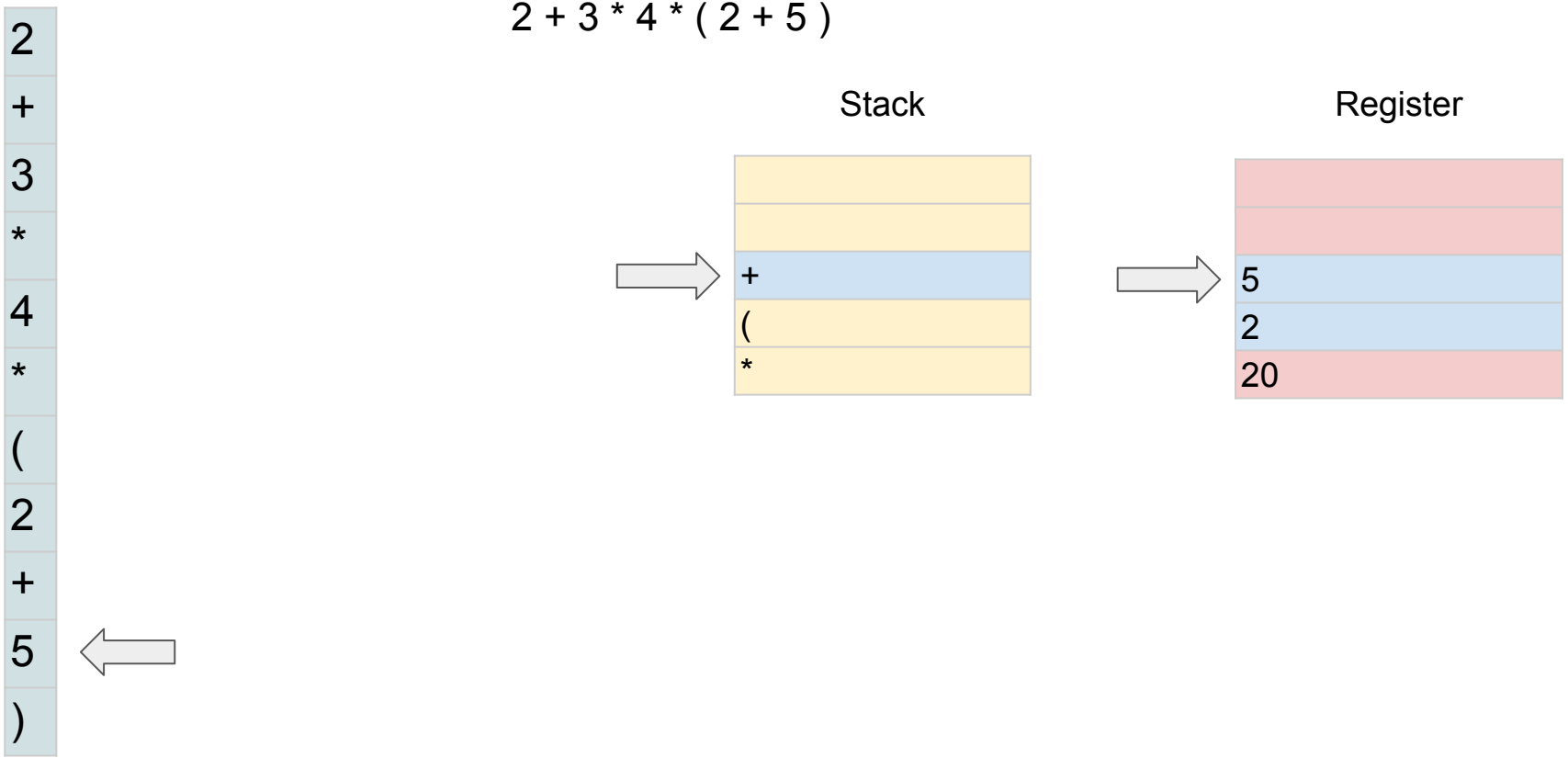
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



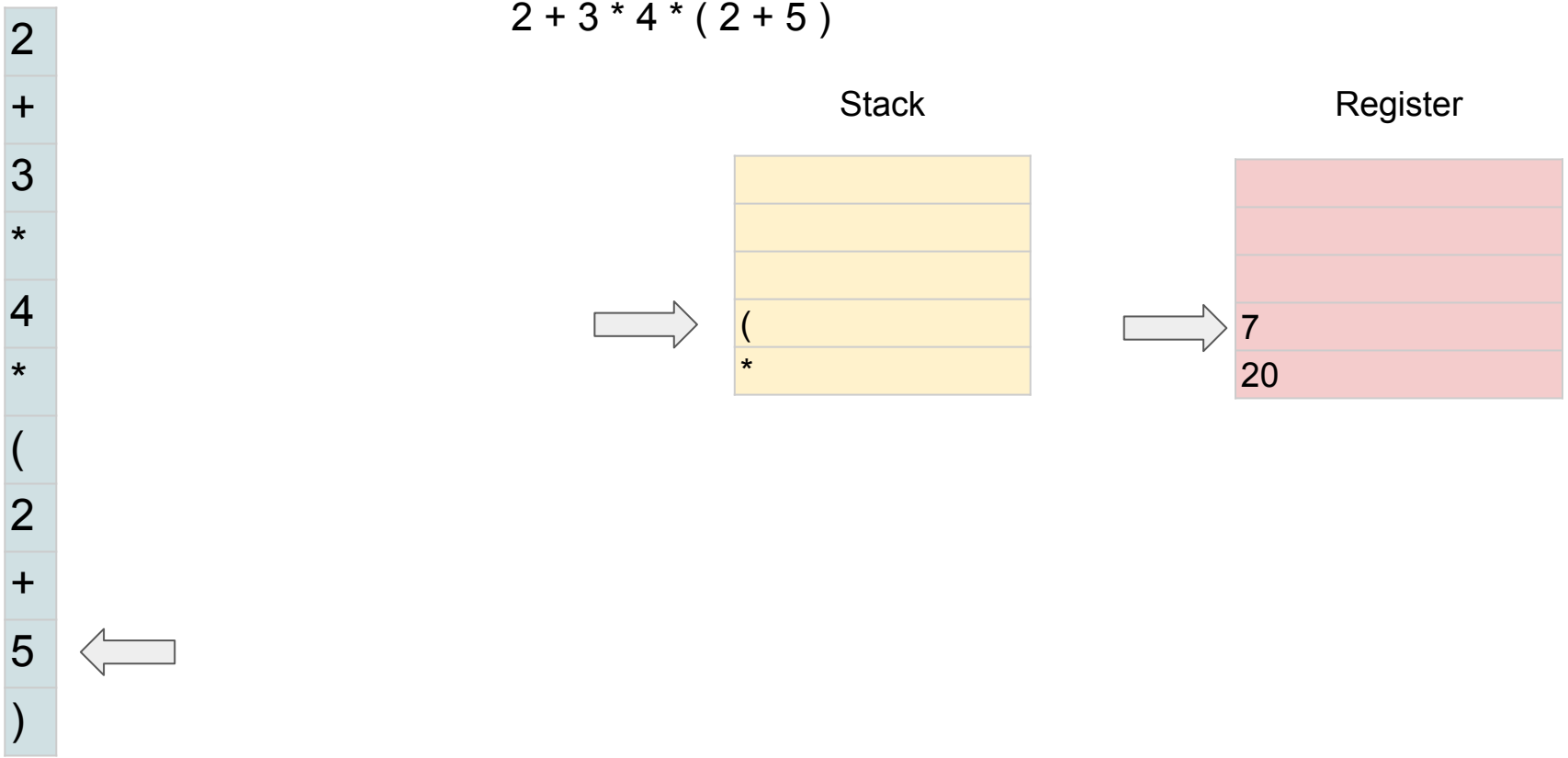
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



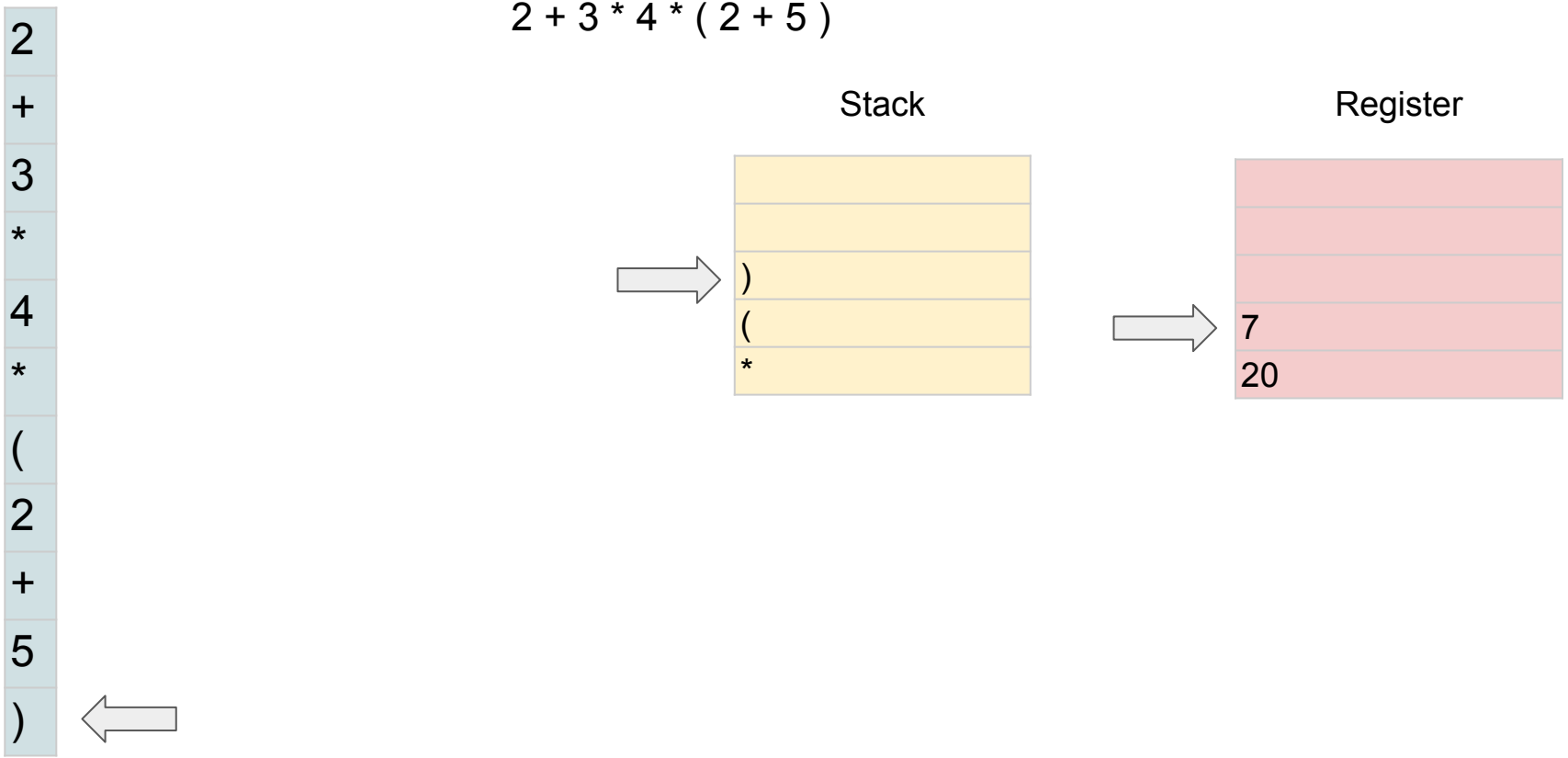
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



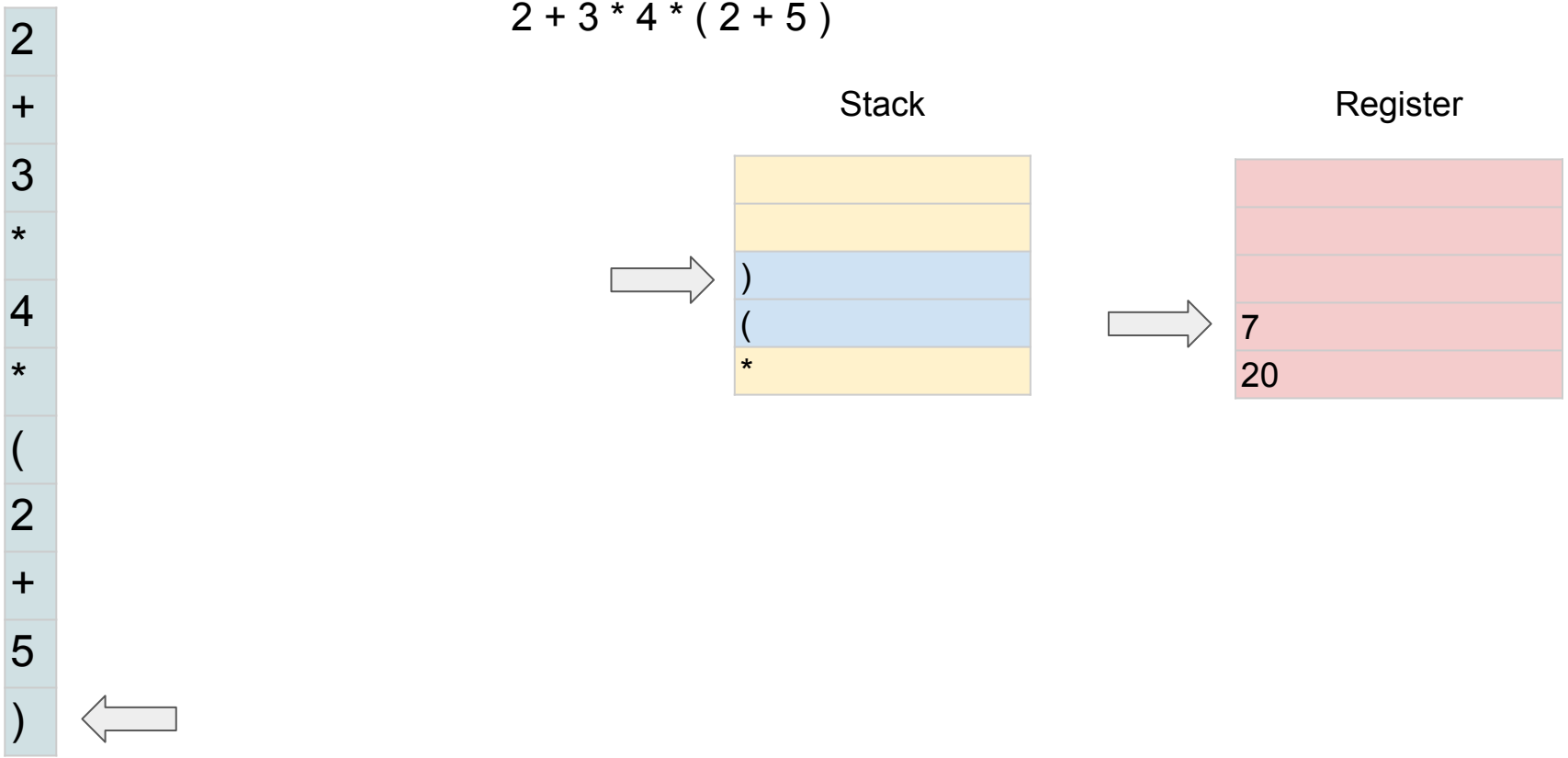
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



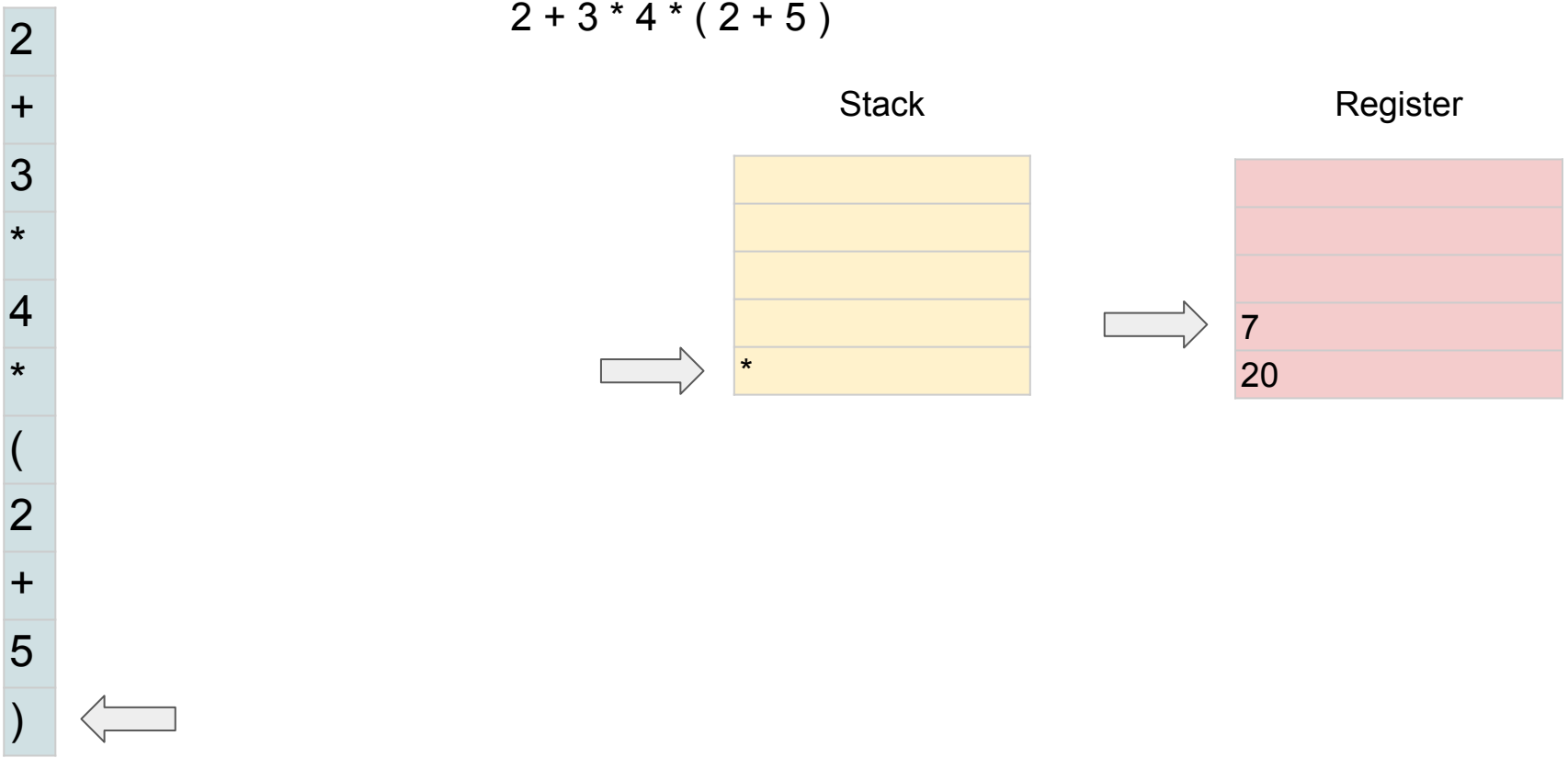
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



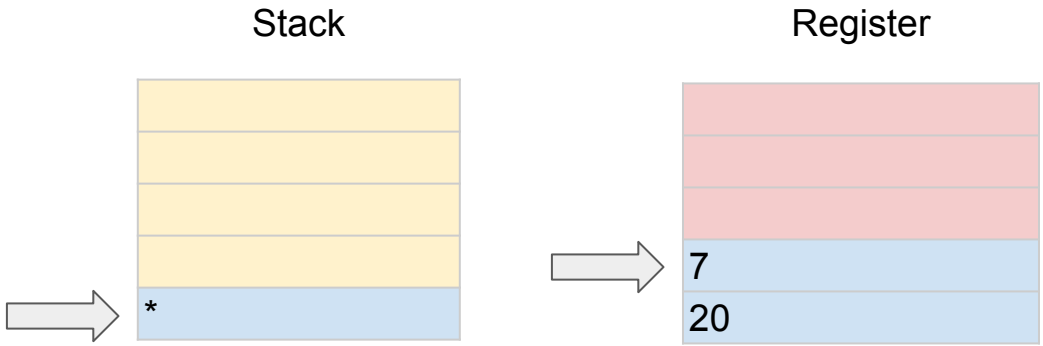
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



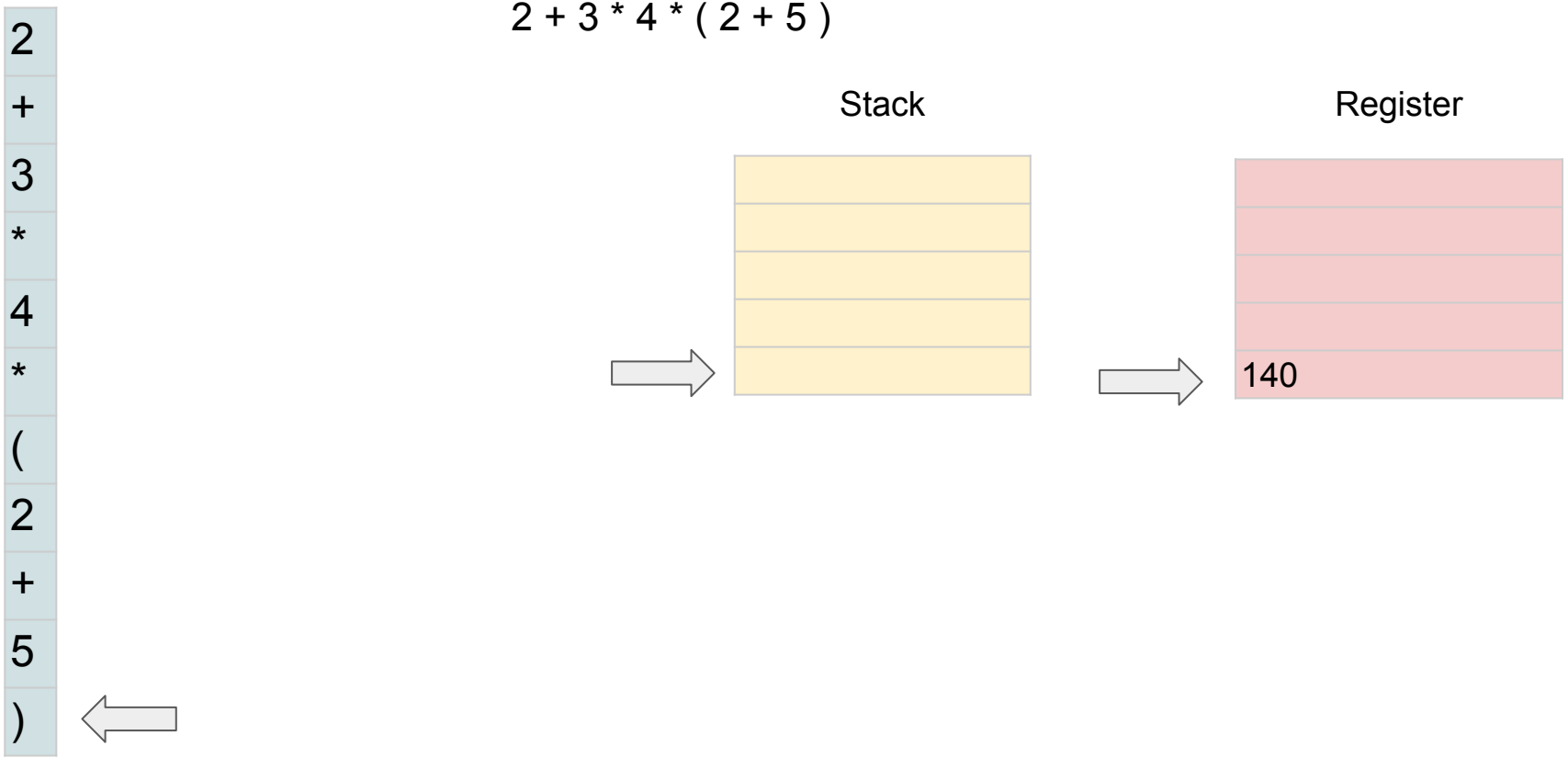
Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



Разбор формул без приоритета

2 + 3 * 4 * (2 + 5)



Разбор формул с приоритетом

- Есть много подходов к введению приоритета операций в разбор формул, мы рассмотрим один из них - алгоритм сортировочной станции (Shunting-yard algorithm).

- Для использования его предварительно формулу нужно привести к обратной польской нотации.

Что такое обратная польская нотация?

- В нашем повседневном пользовании мы для записи формул используем так называемую инфиксную нотацию (*2 + 2*) совершая операцию над двумя аргумента, мы записываем оператор между ними. Можно делать иначе, формулы становятся менее читаемы для человека, но более приятны для парсинга машиной.

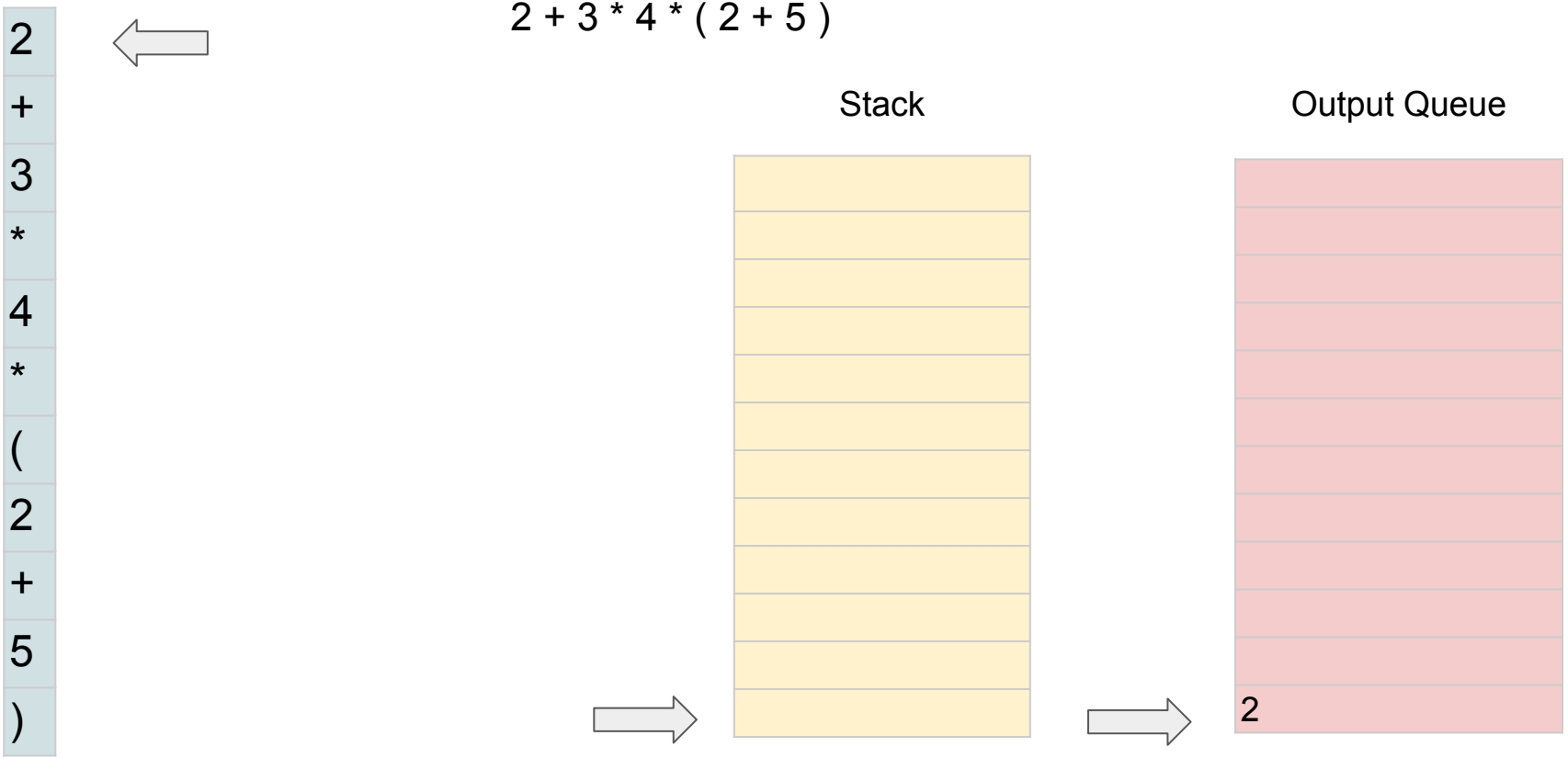
- Префиксная нотация: +22
- Постфиксная нотация: 22+

- Речь сейчас идет строго о способе записи символов. Смысл и результат сложения двух двоек от способа записи не меняется.

- Постфиксную нотацию (22+) принято называть Польской обратной нотацией.

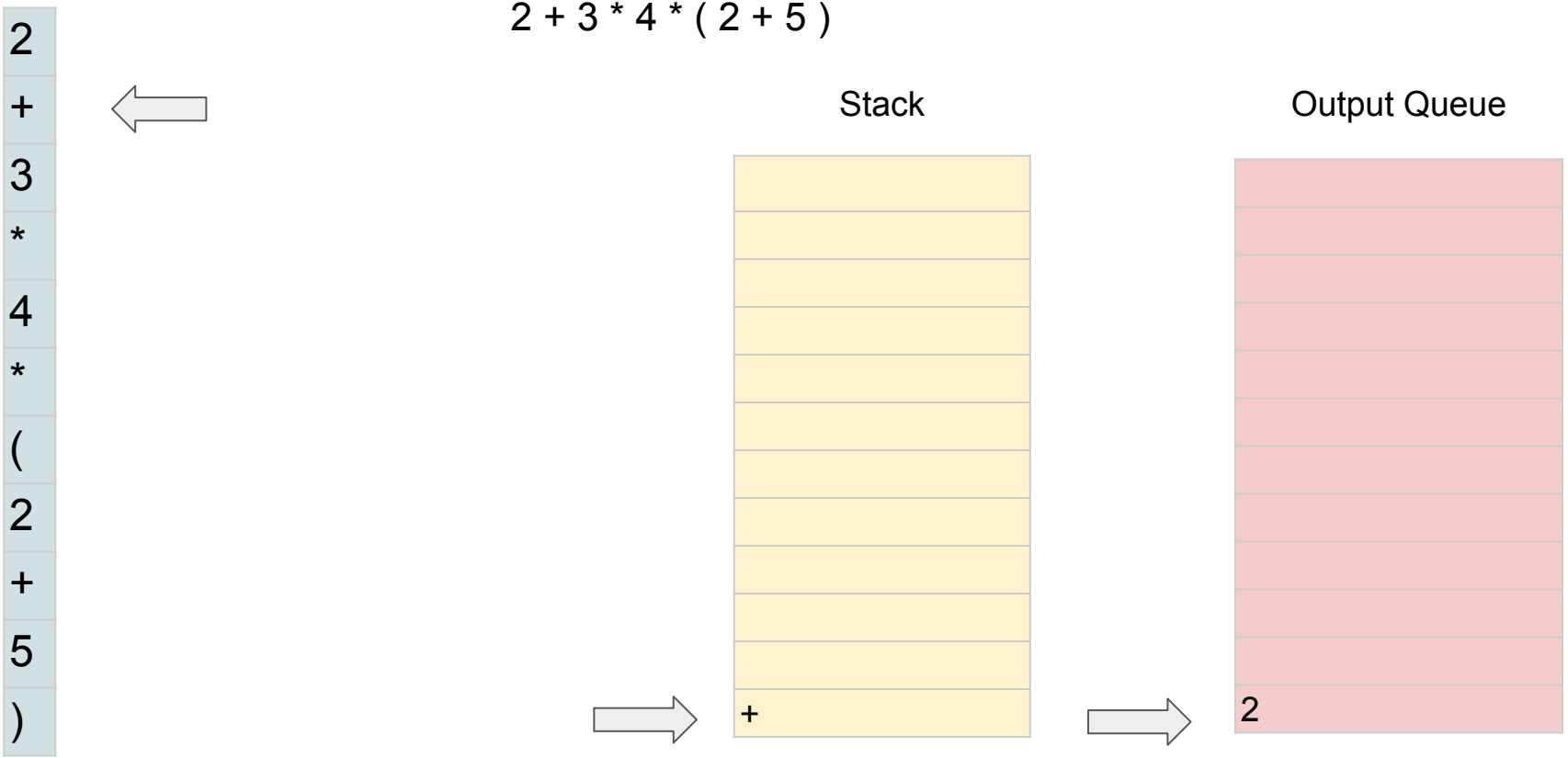
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



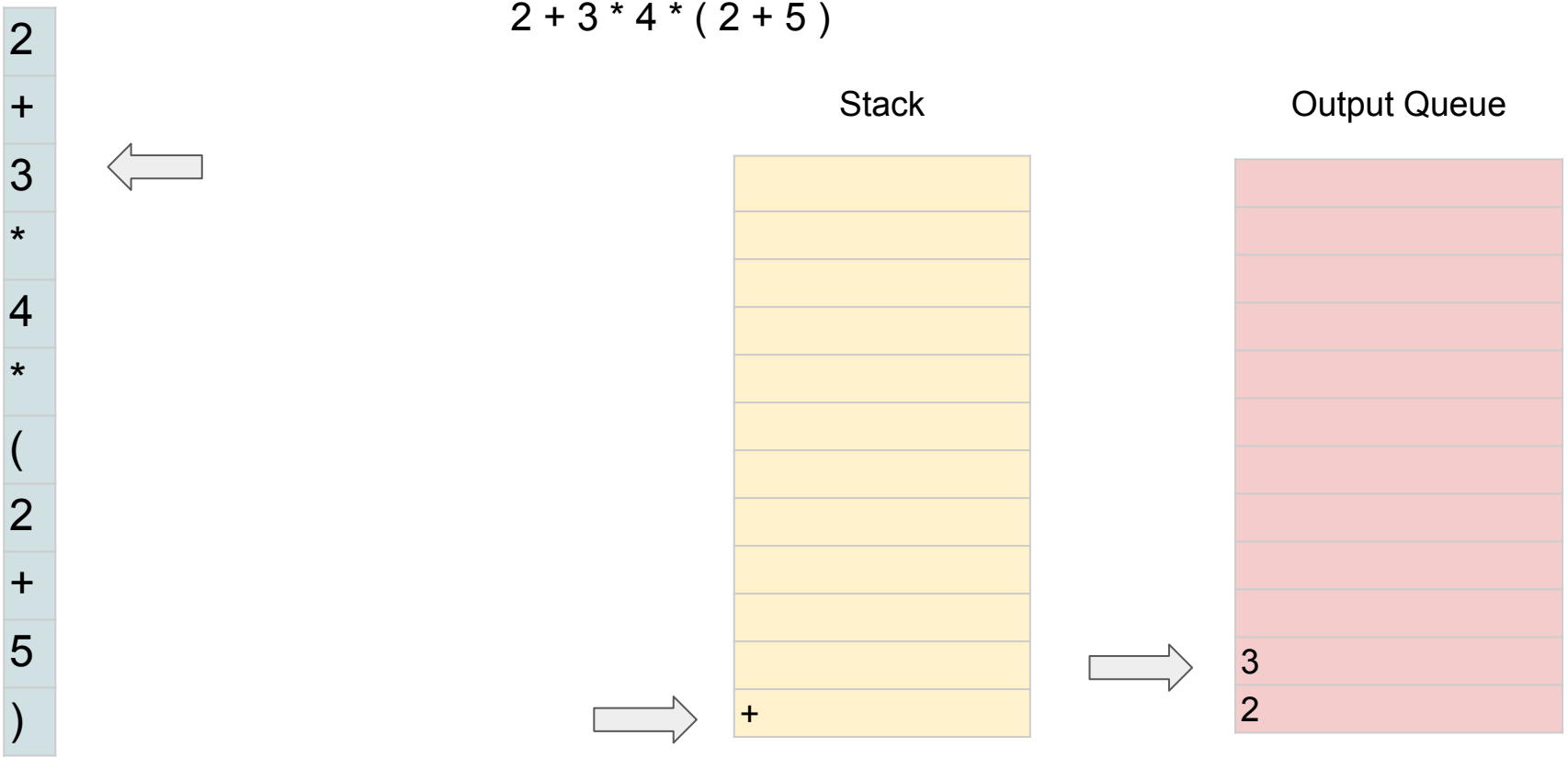
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



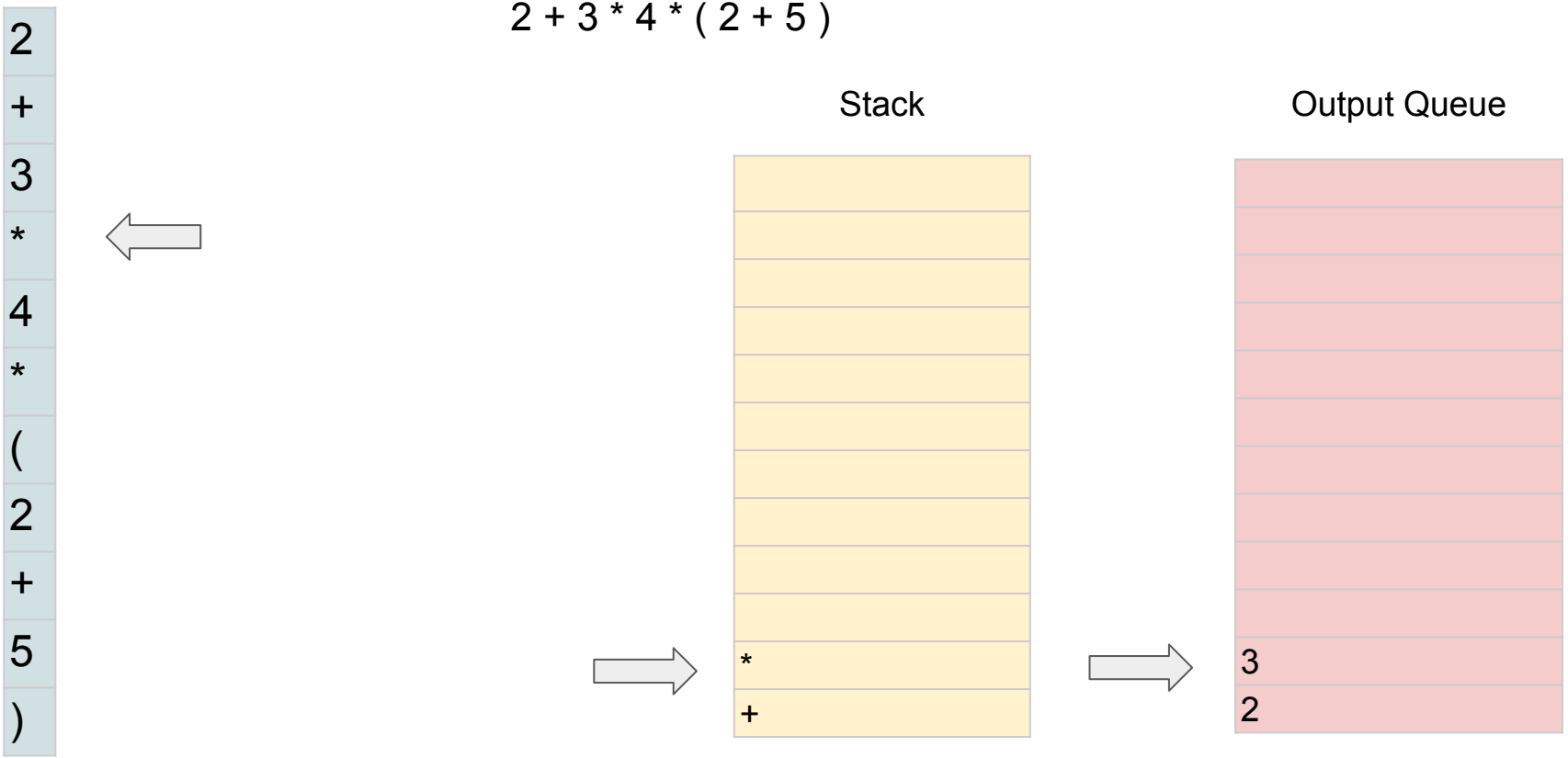
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



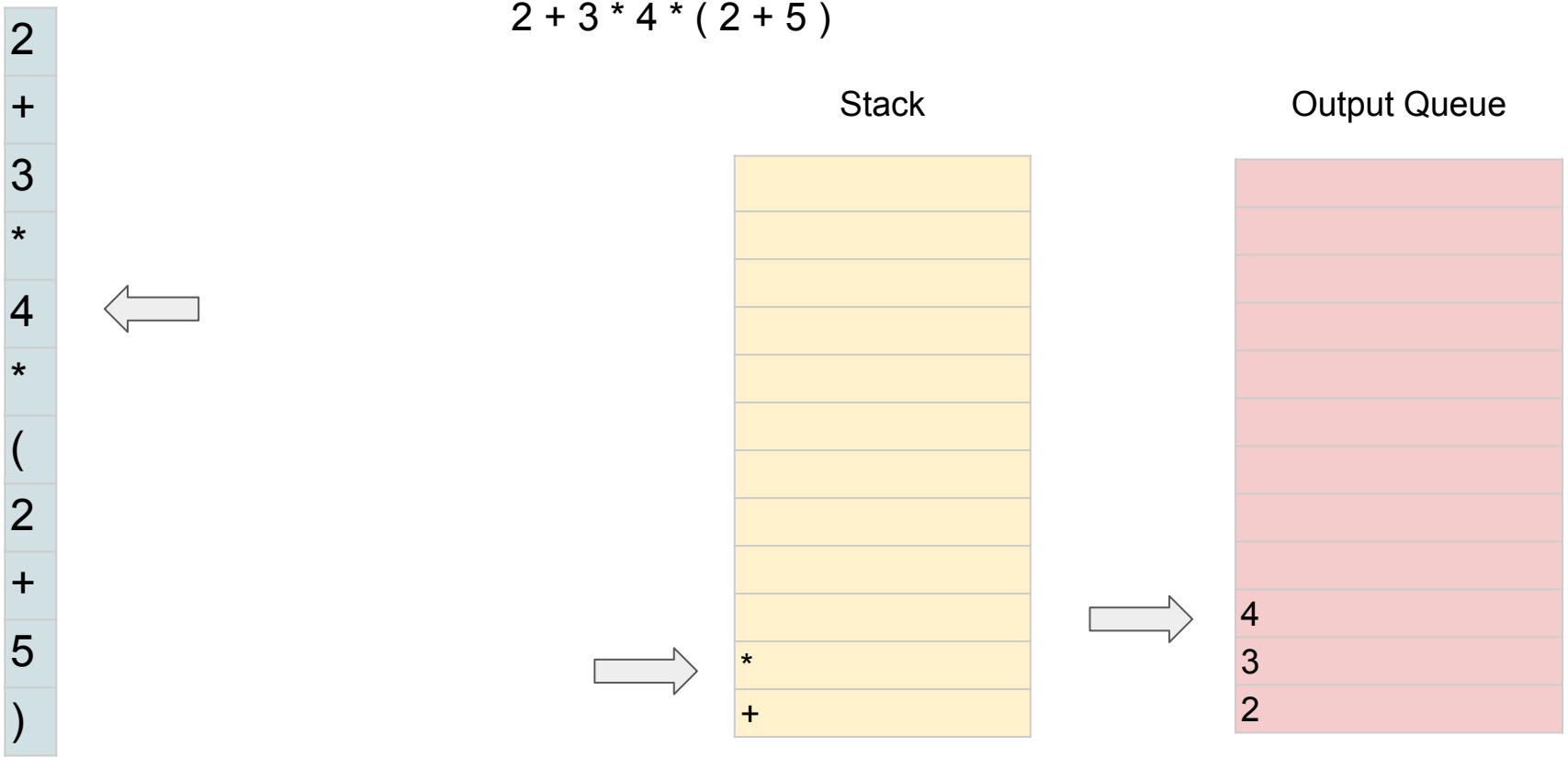
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



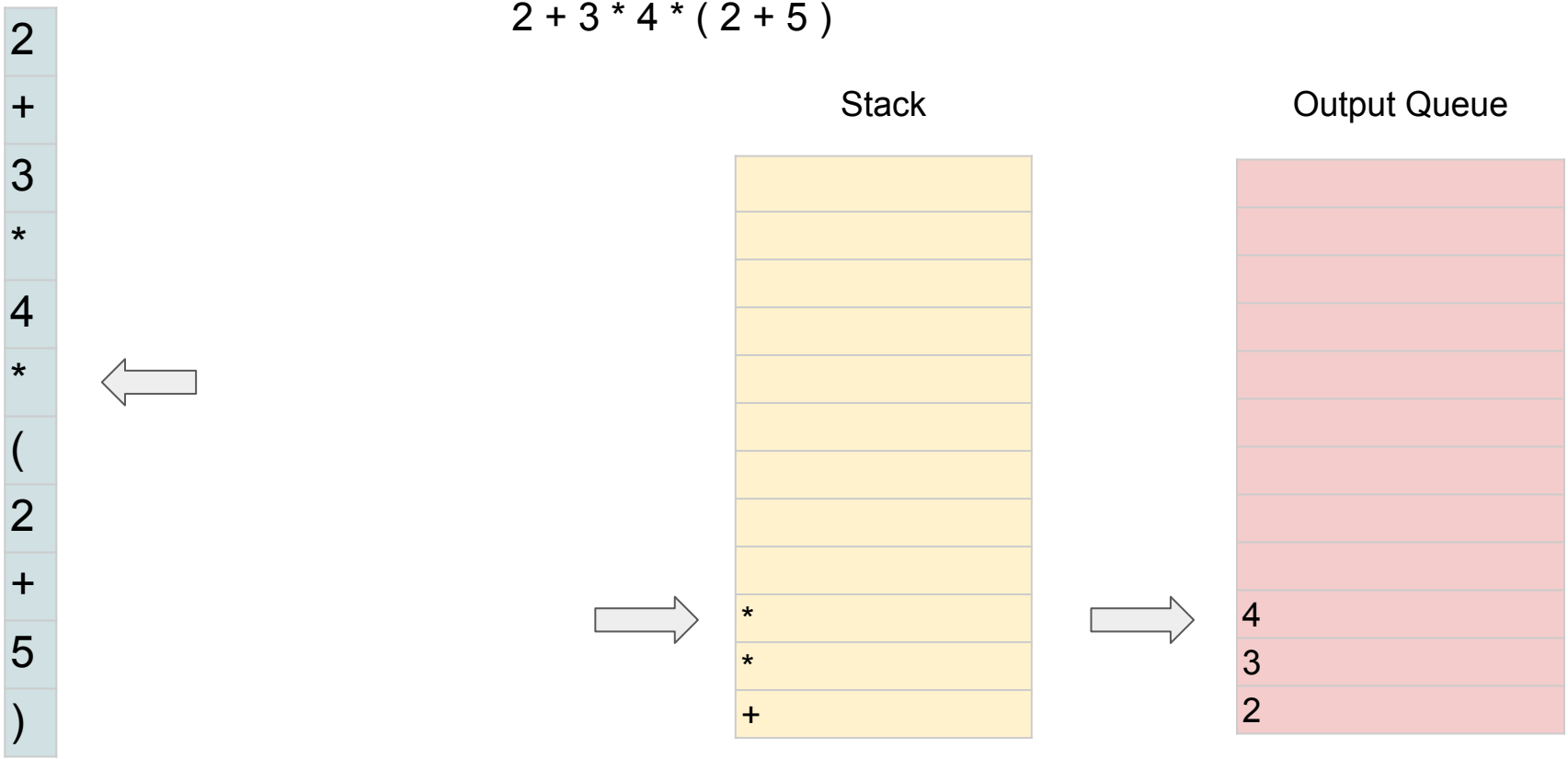
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



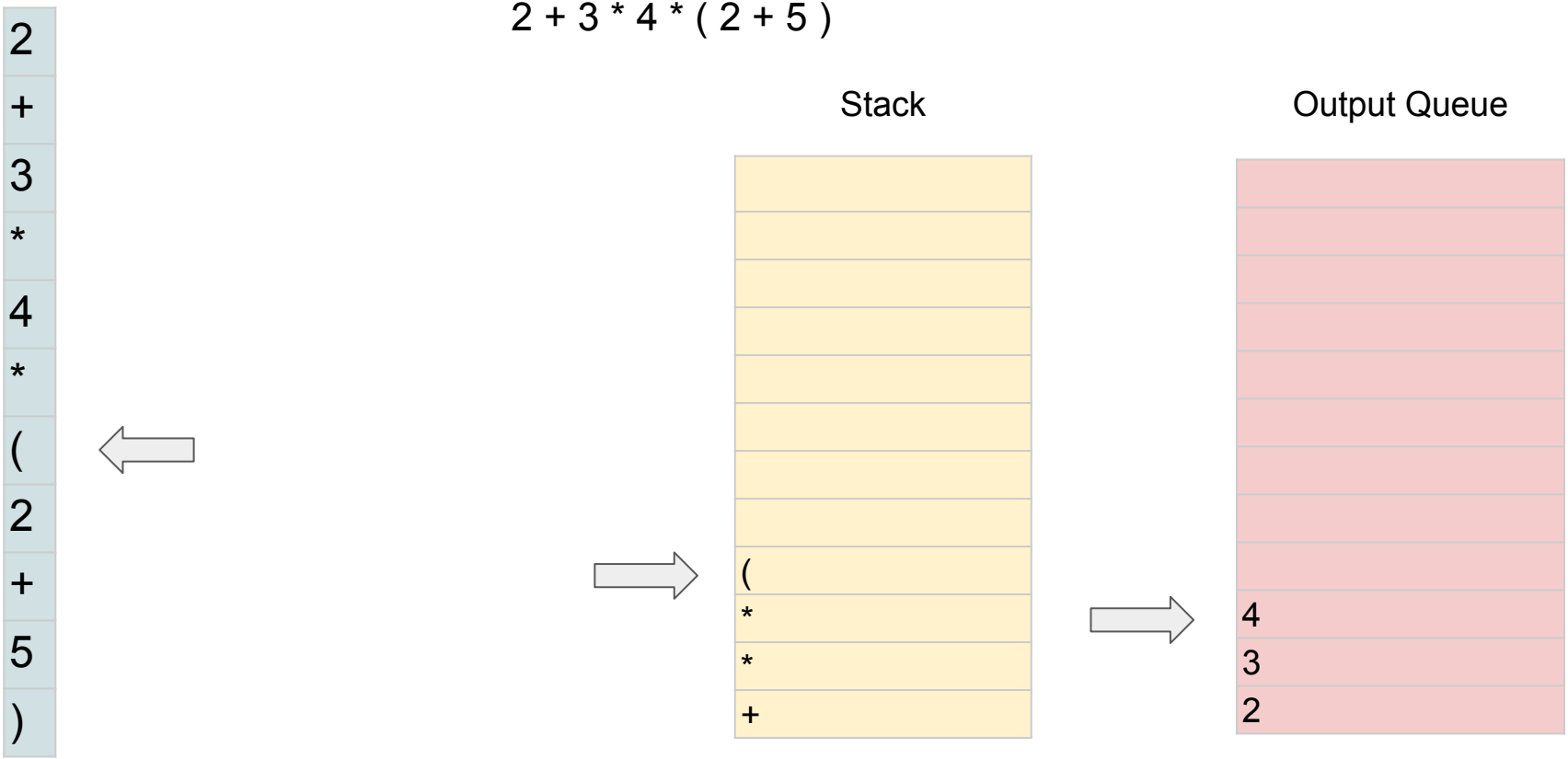
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



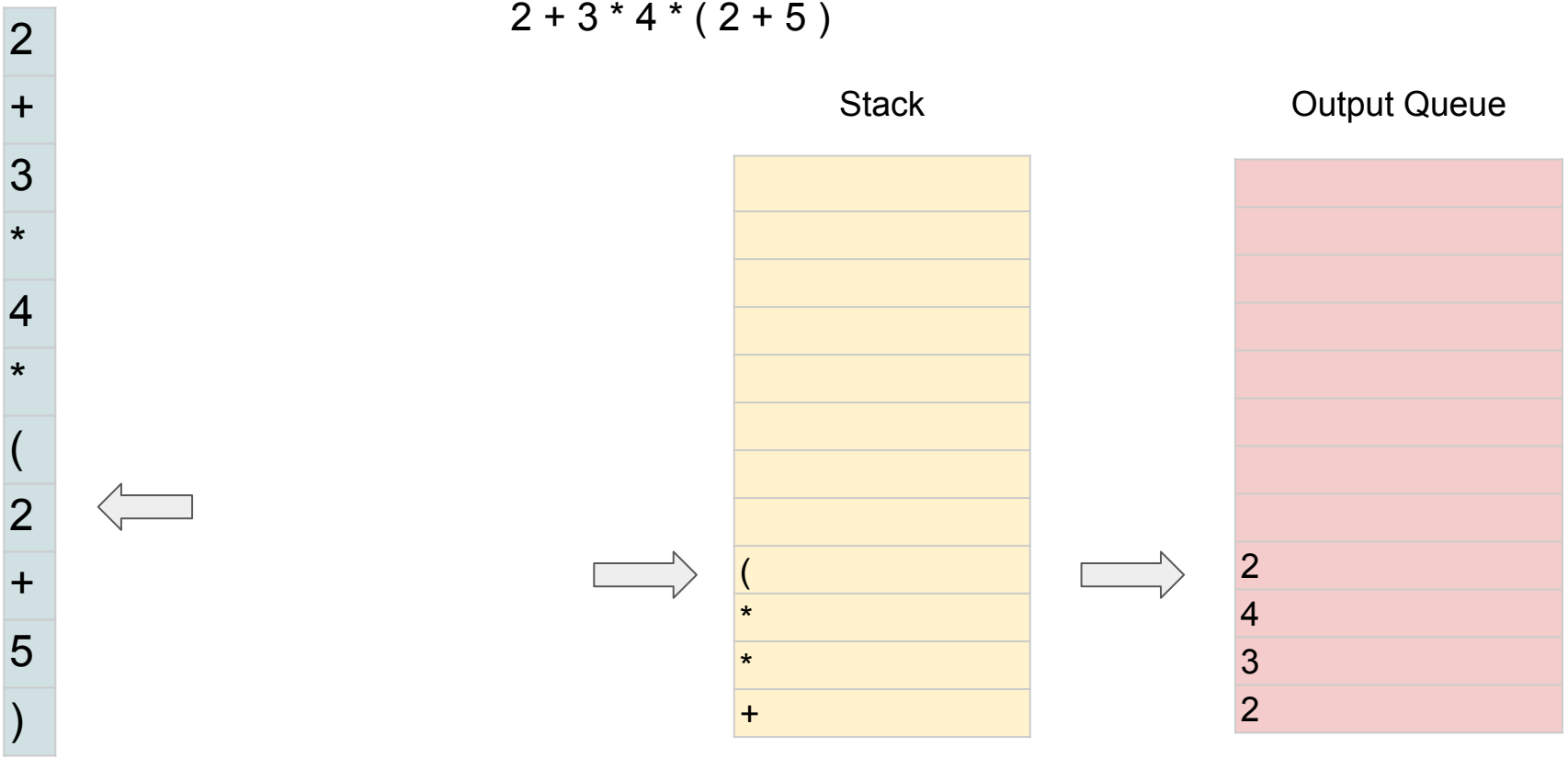
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



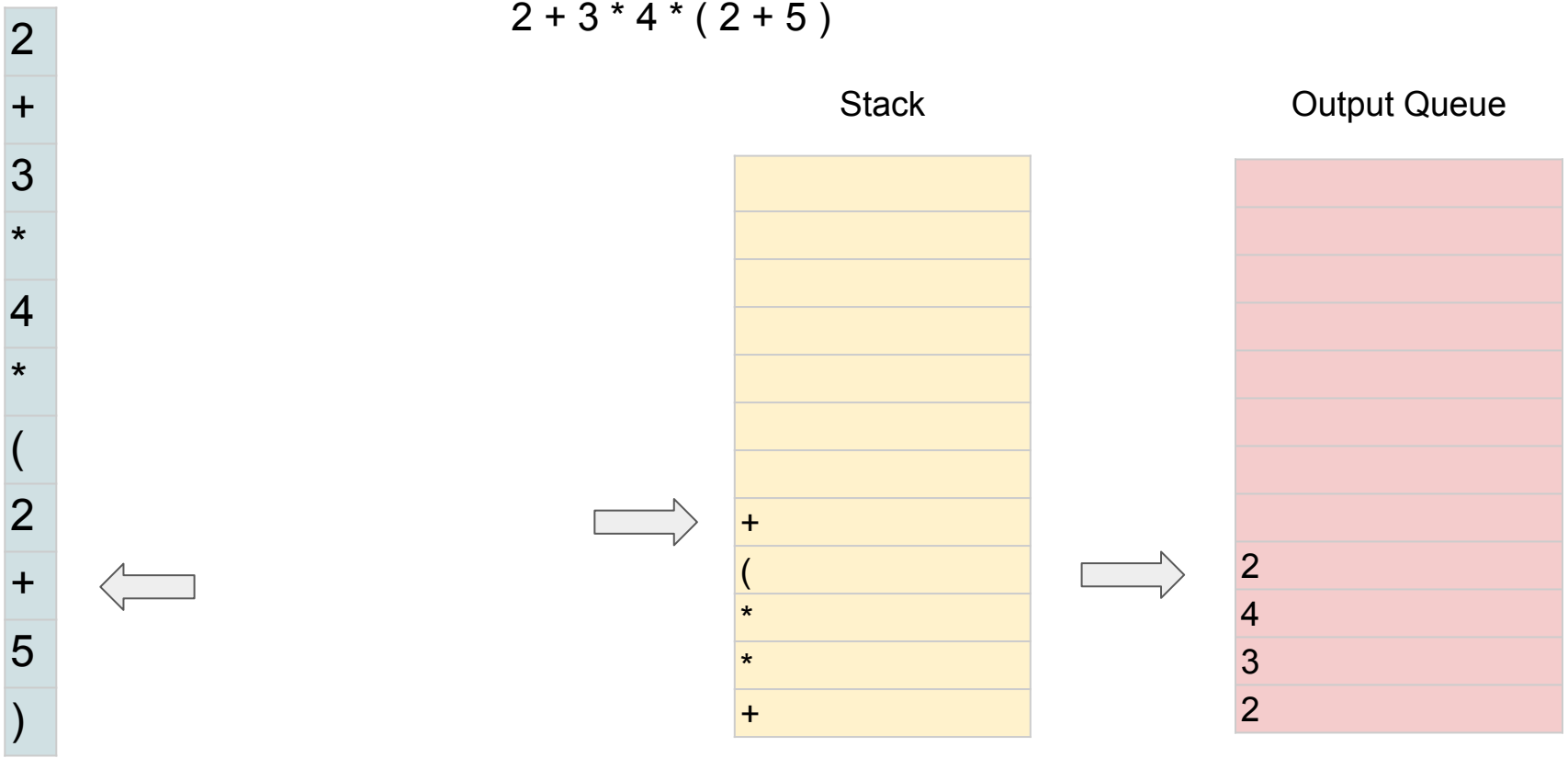
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



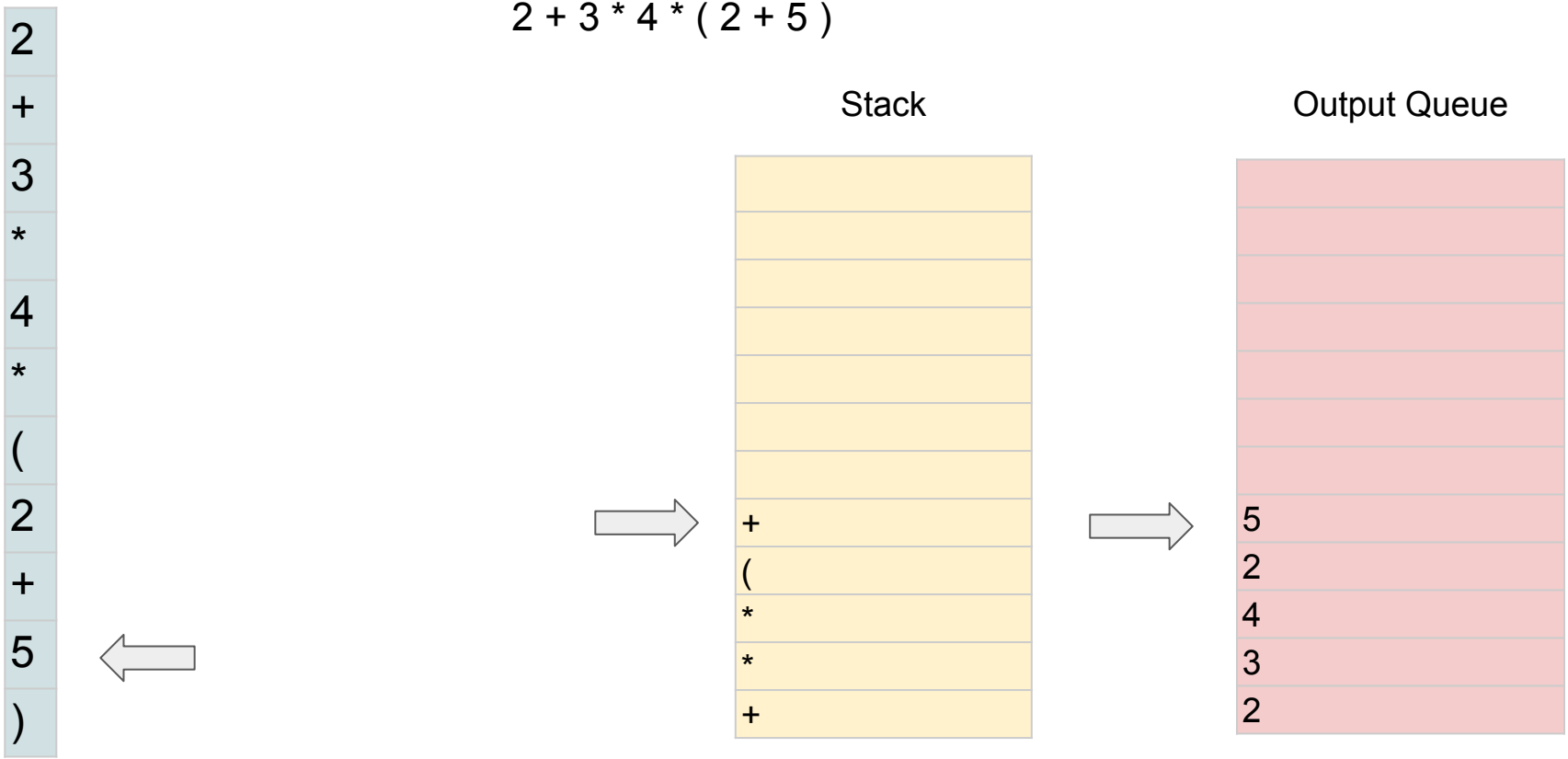
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



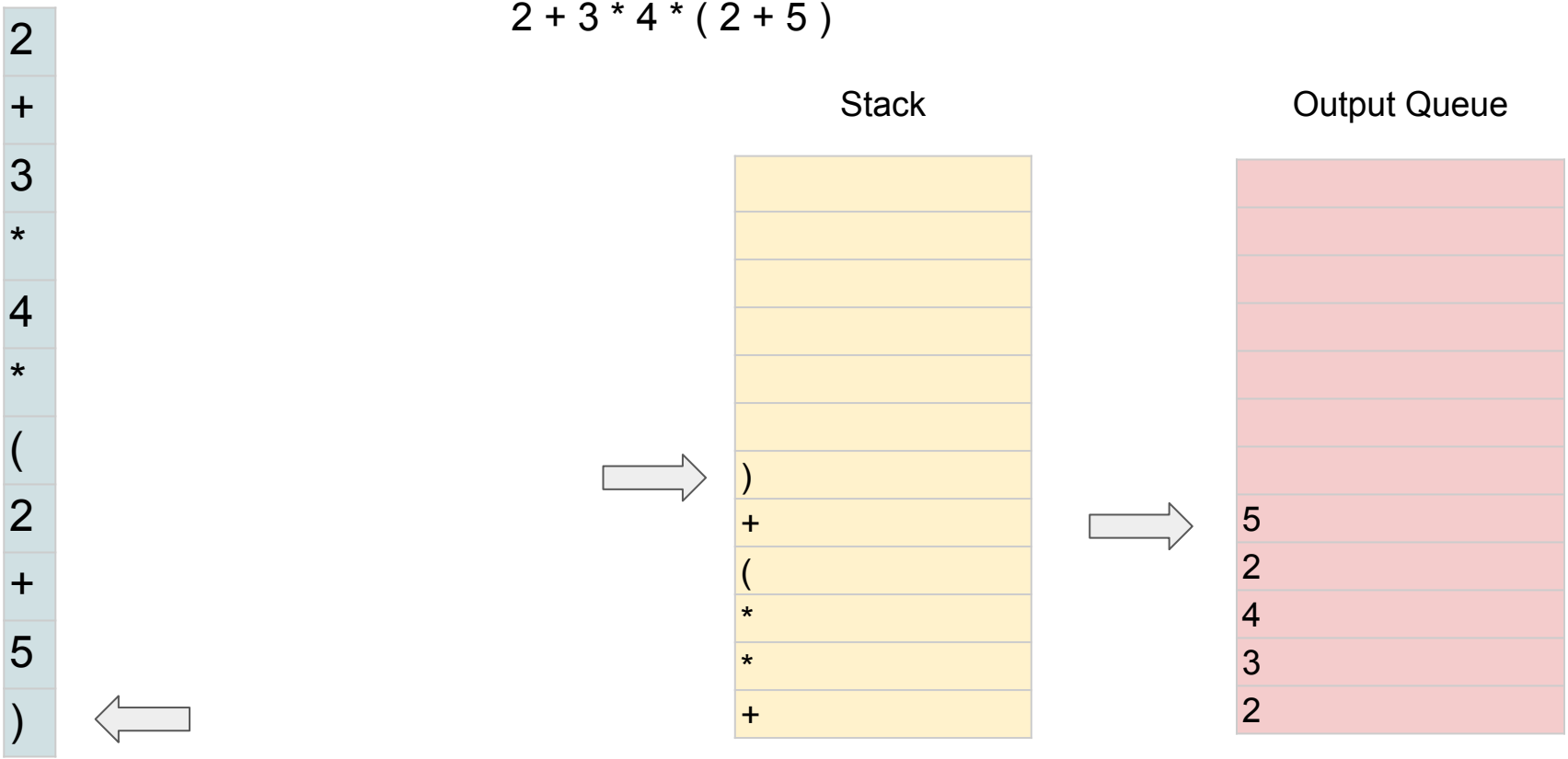
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



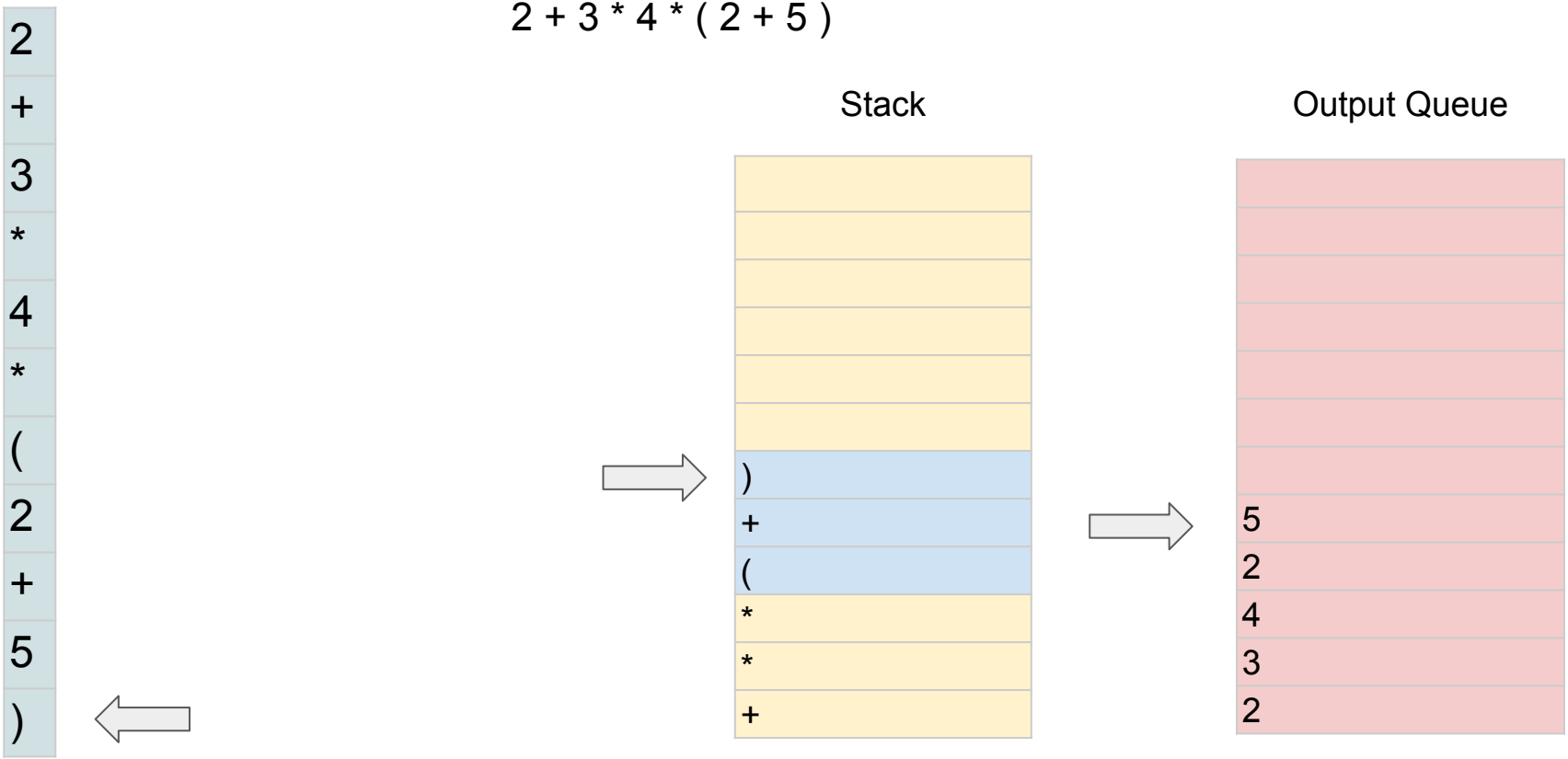
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



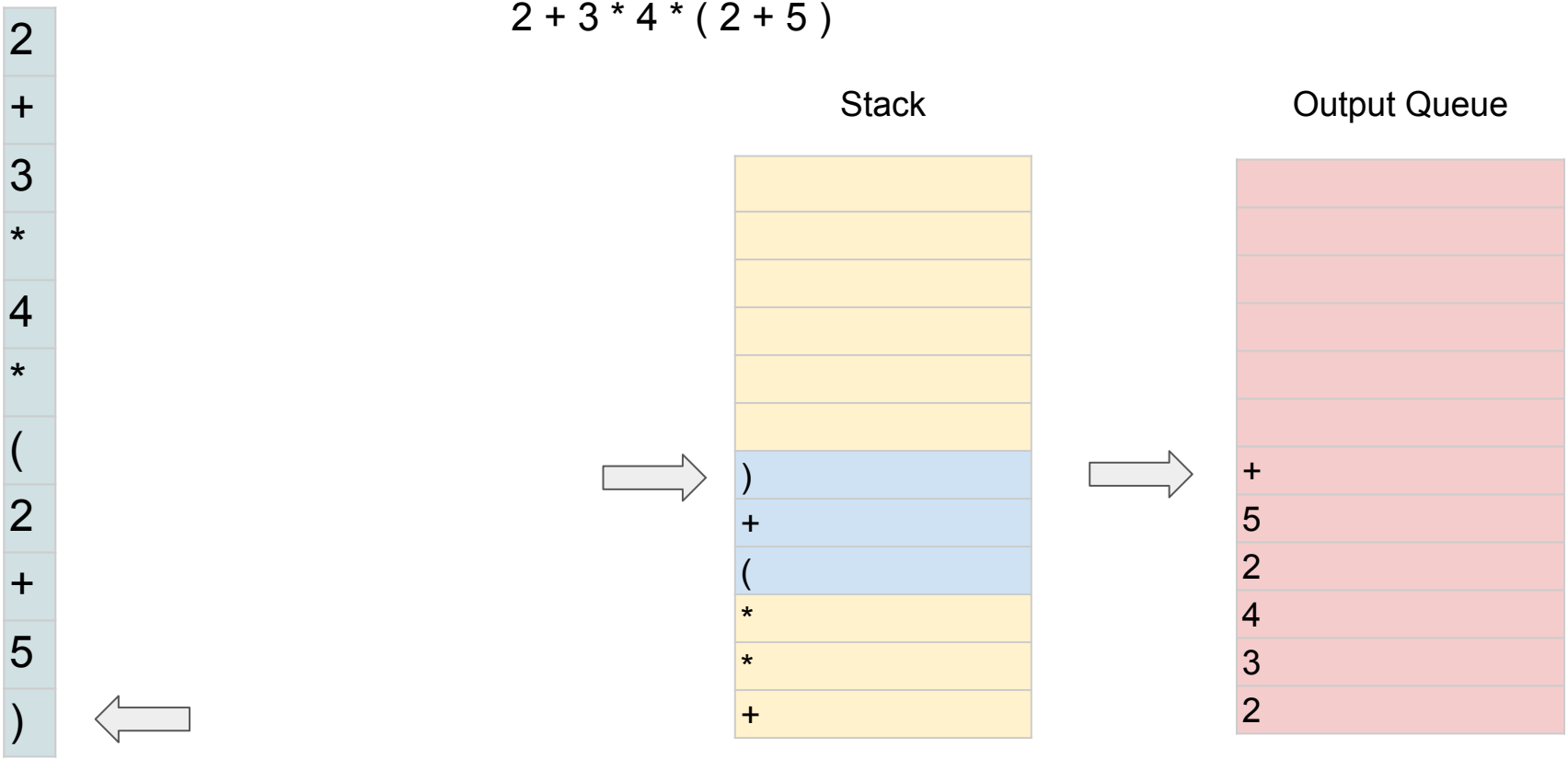
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



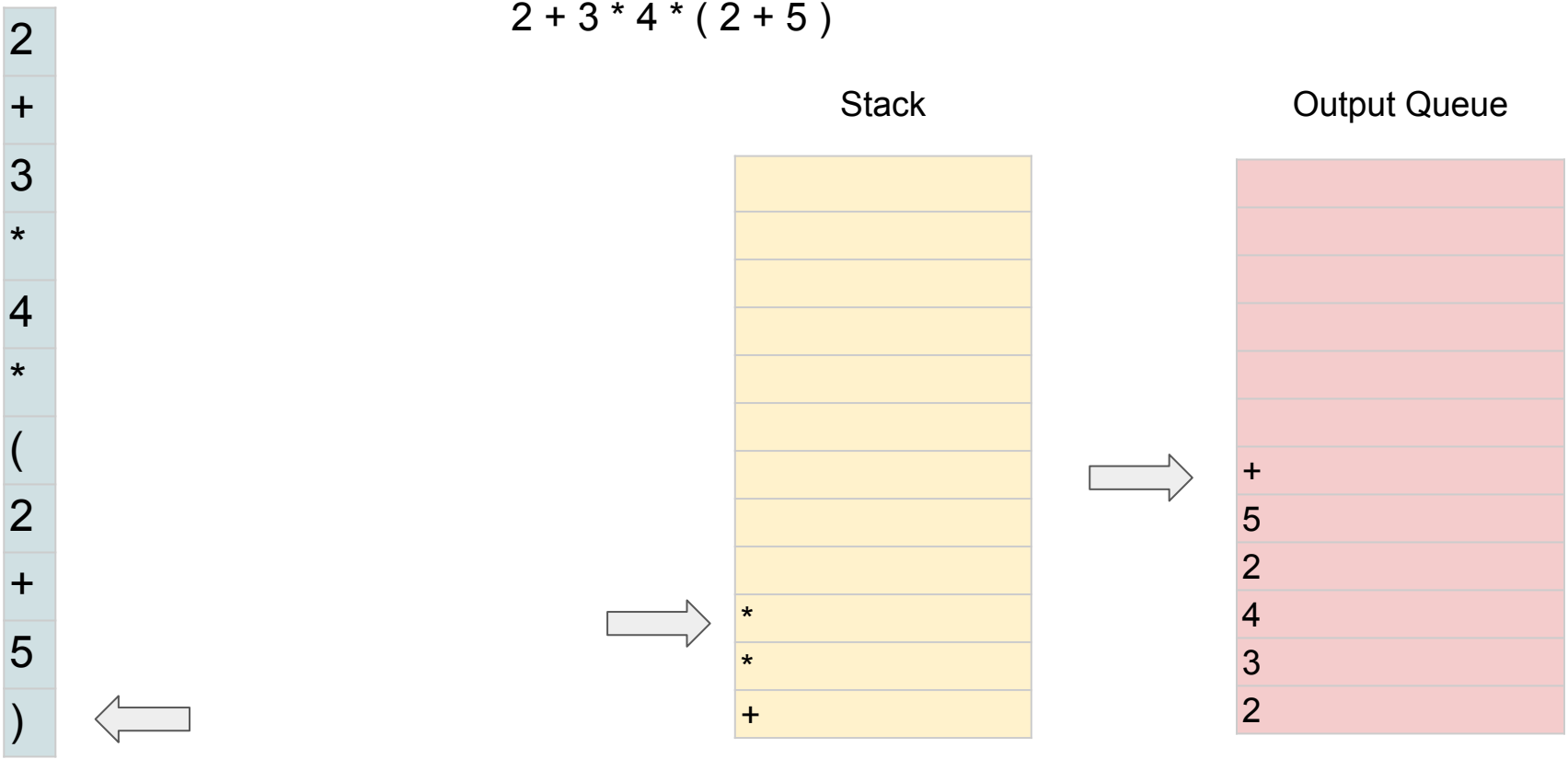
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



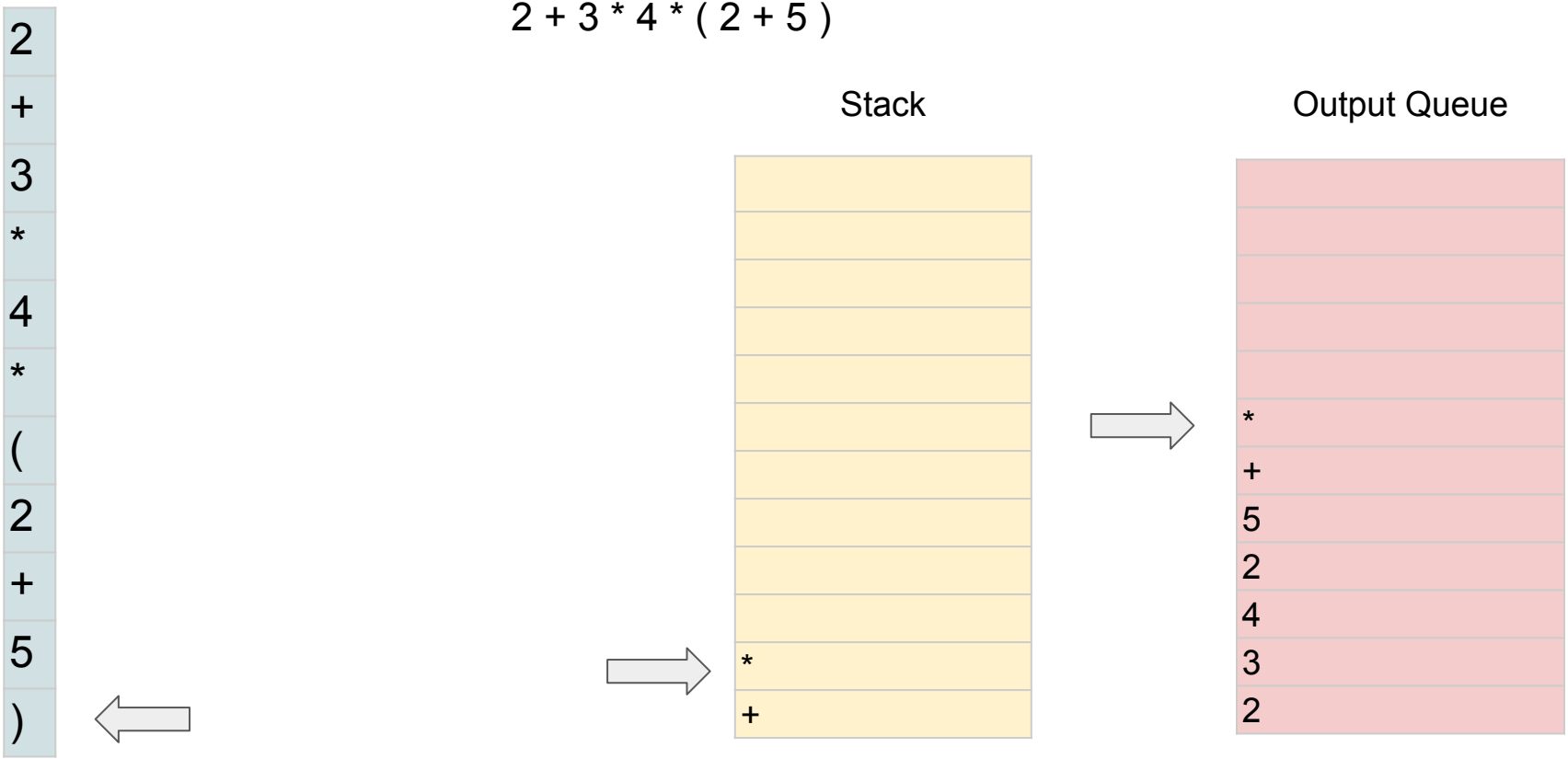
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



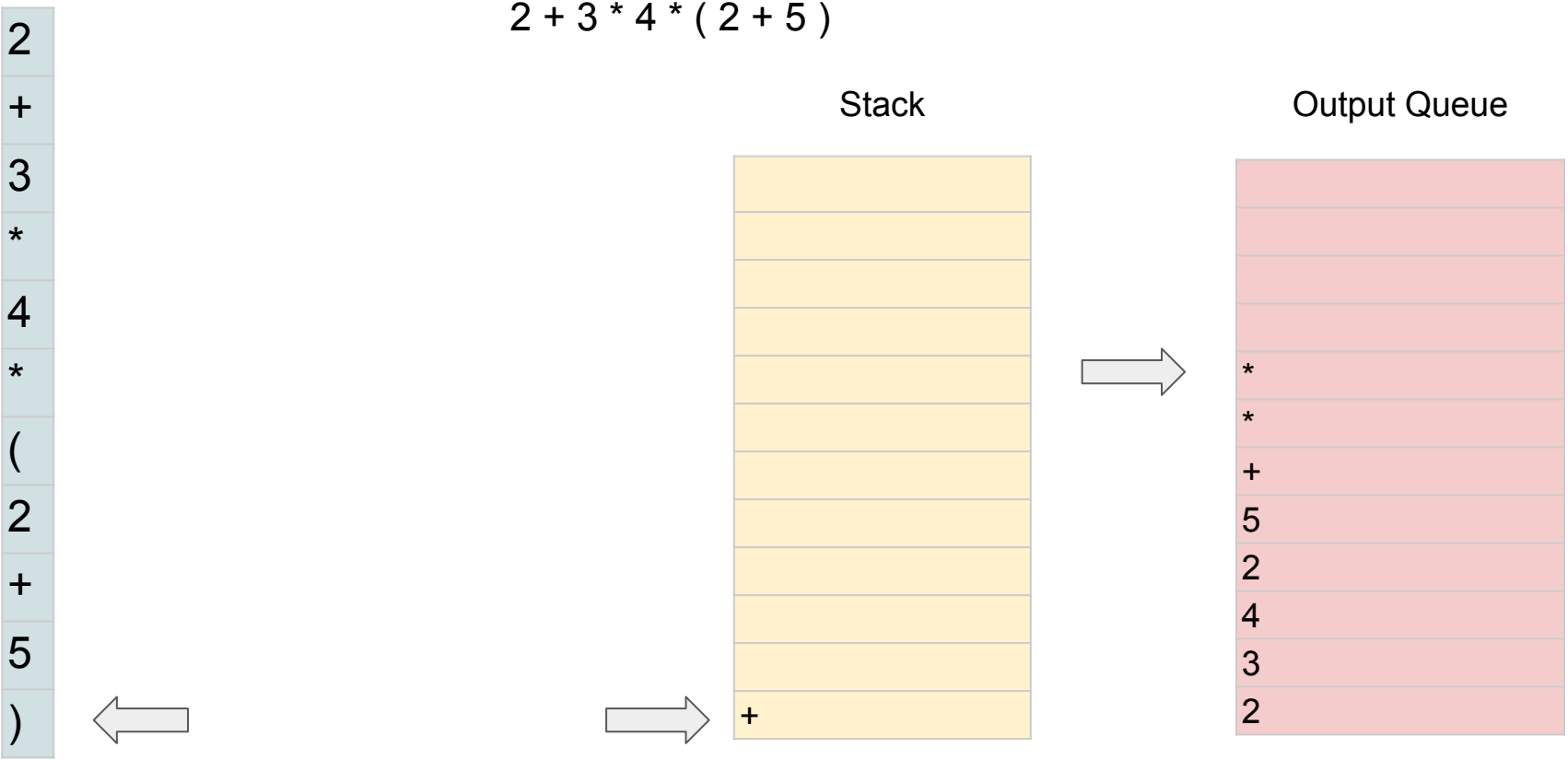
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



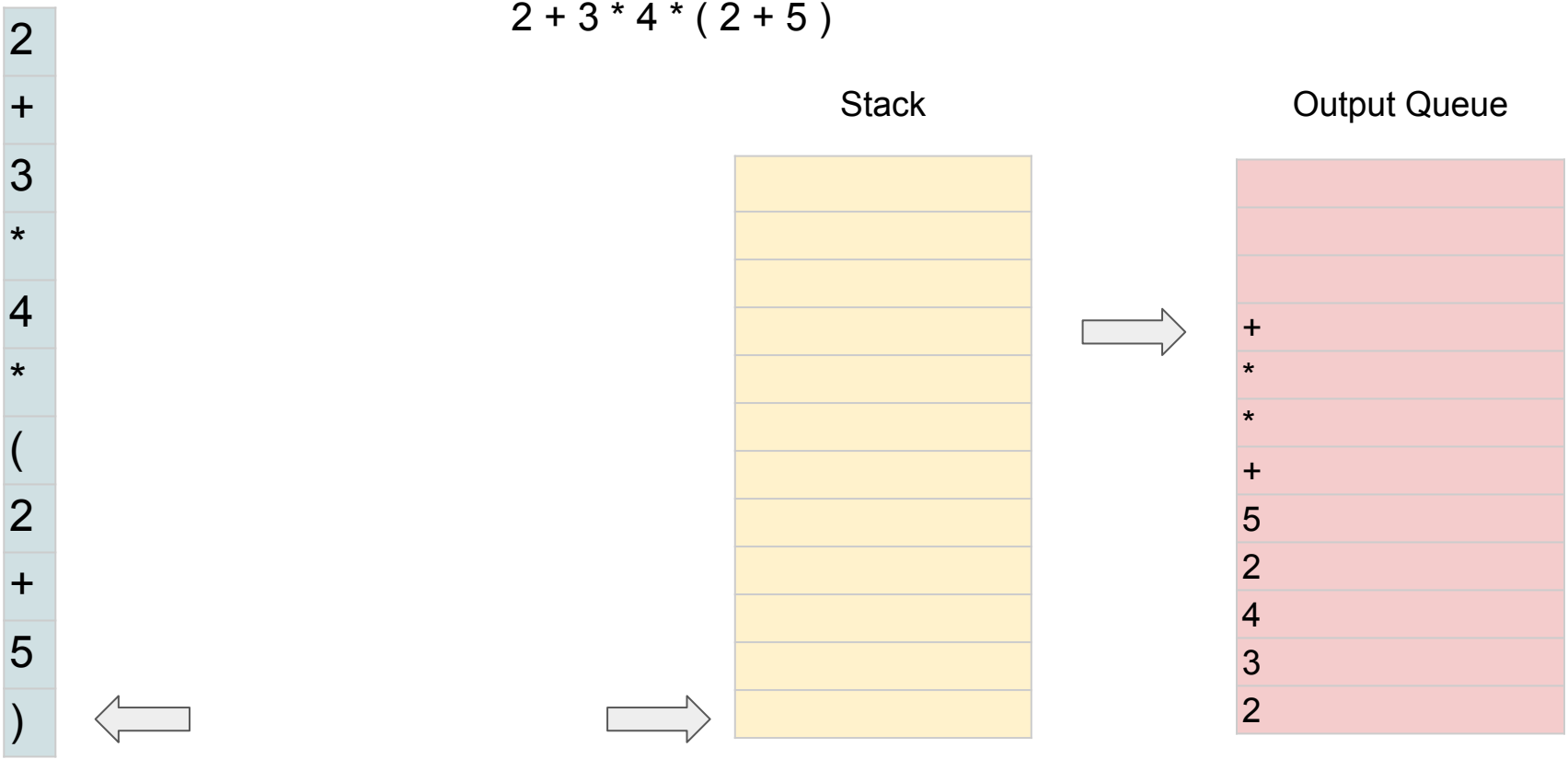
Приведение к обратной польской нотации

2 + 3 * 4 * (2 + 5)



Приведение к обратной польской нотации

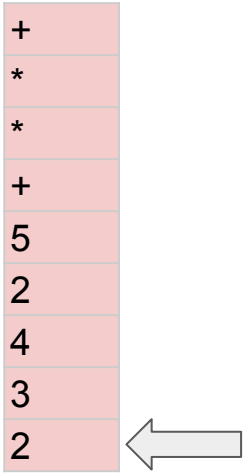
2 + 3 * 4 * (2 + 5)



Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue

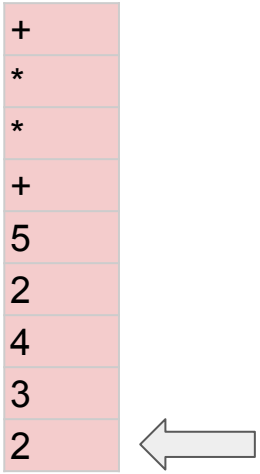


Operation Stack

Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue

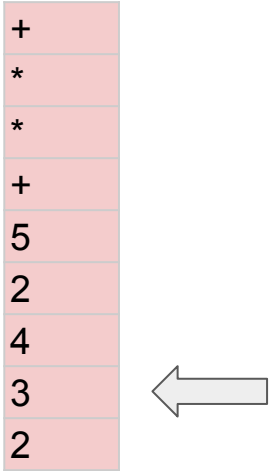


Operation Stack

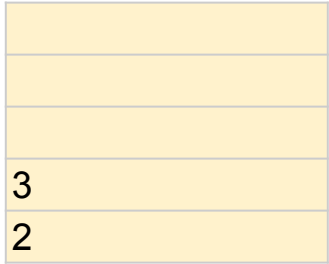
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



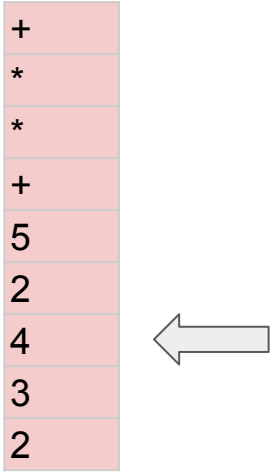
Operation Stack



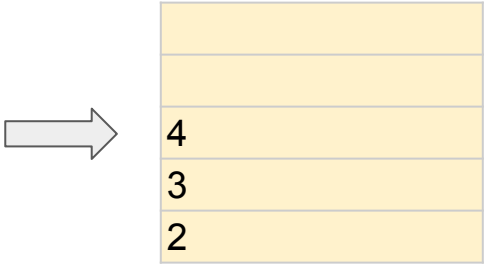
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



Operation Stack



Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue

+
*
*
+
5
2
4
3
2

Operation Stack

2
4
3
2

Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue

+
*
*
+
5
2
4
3
2

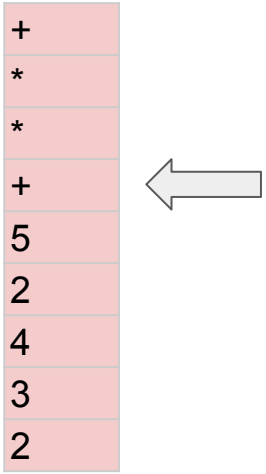
Operation Stack

5
2
4
3
2

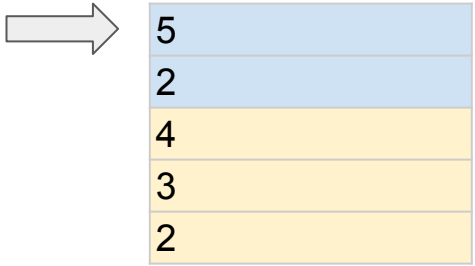
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



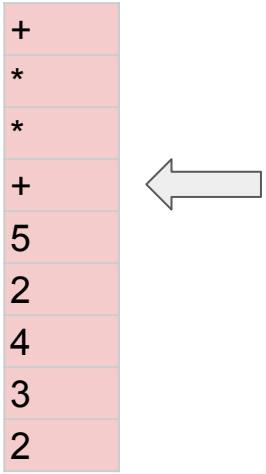
Operation Stack



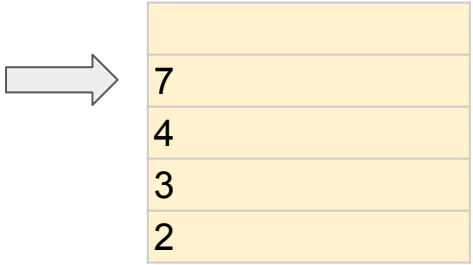
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



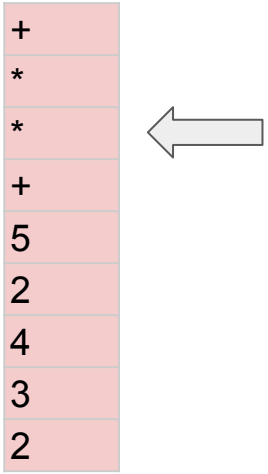
Operation Stack



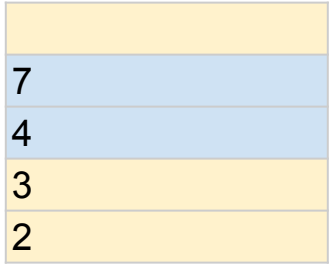
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



Operation Stack



Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue

+
*
*
+
5
2
4
3
2

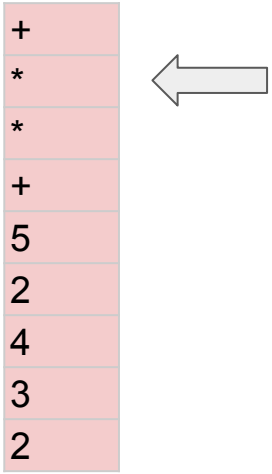
Operation Stack

28
3
2

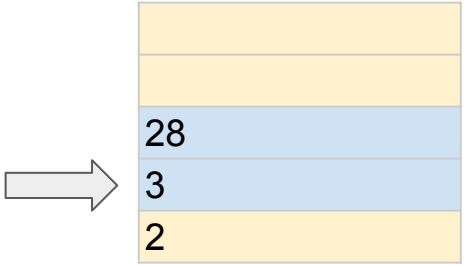
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



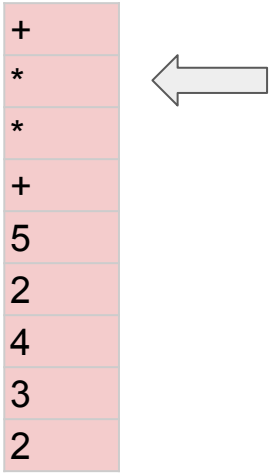
Operation Stack



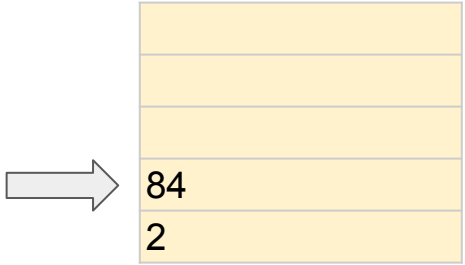
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



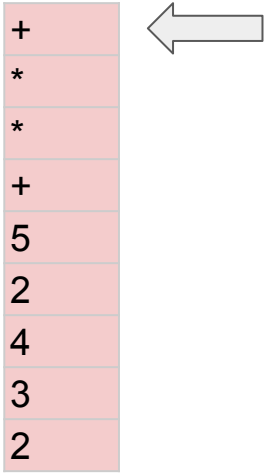
Operation Stack



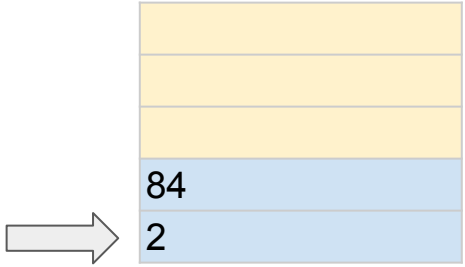
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



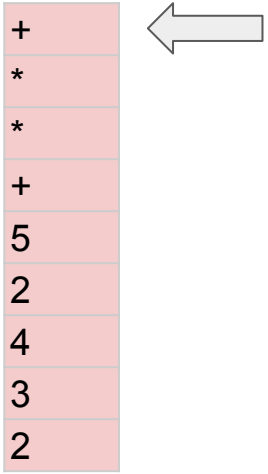
Operation Stack



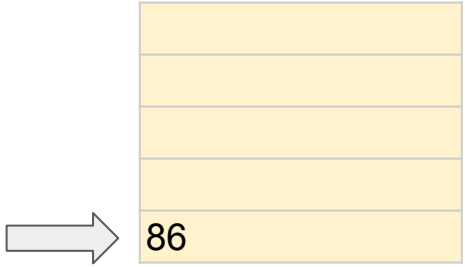
Разбор приведенной формулы

2 + 3 * 4 * (2 + 5)

Output Queue



Operation Stack



А если бы все было немного сложнее?

23*67++

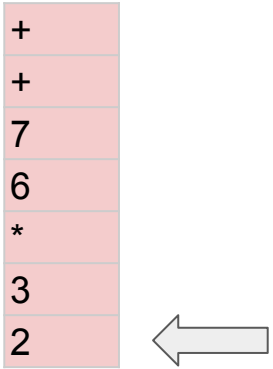
Output Queue

Operation Stack

А если бы все было немного сложнее?

23*67++

Output Queue

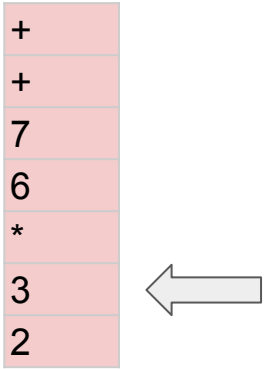


Operation Stack

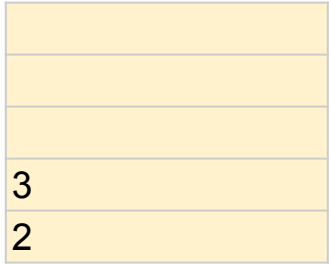
А если бы все было немного сложнее?

23*67++

Output Queue



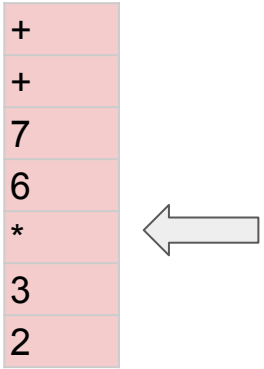
Operation Stack



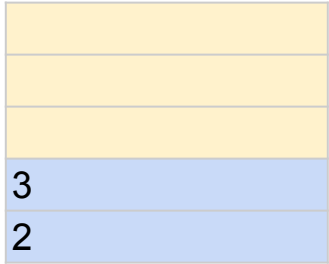
А если бы все было немного сложнее?

23*67++

Output Queue



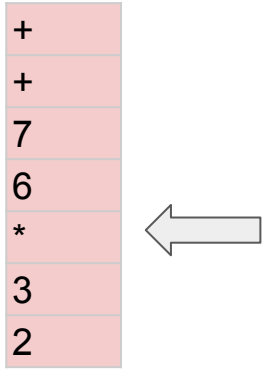
Operation Stack



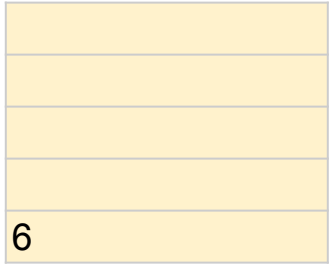
А если бы все было немного сложнее?

23*67++

Output Queue



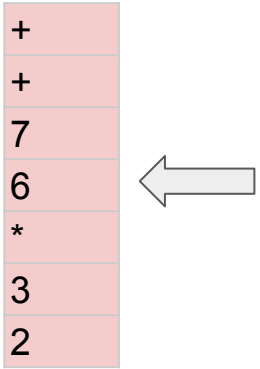
Operation Stack



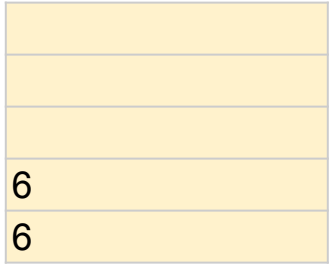
А если бы все было немного сложнее?

23*67++

Output Queue



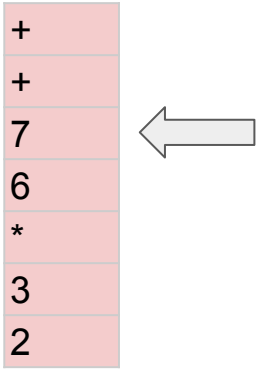
Operation Stack



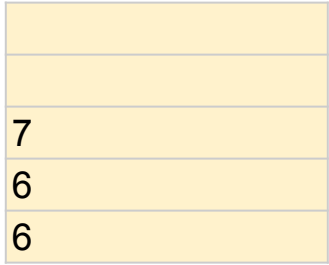
А если бы все было немного сложнее?

23*67++

Output Queue



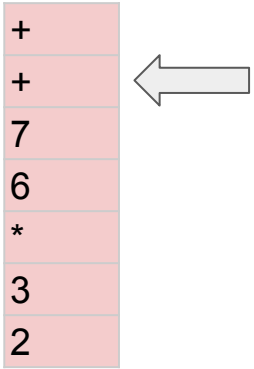
Operation Stack



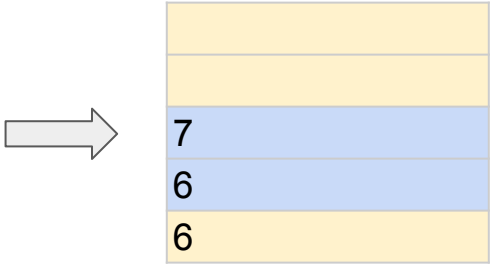
А если бы все было немного сложнее?

23*67++

Output Queue



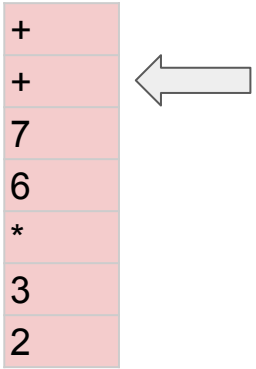
Operation Stack



А если бы все было немного сложнее?

23*67++

Output Queue

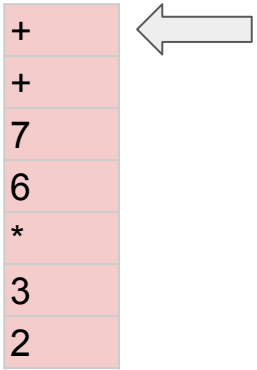


Operation Stack

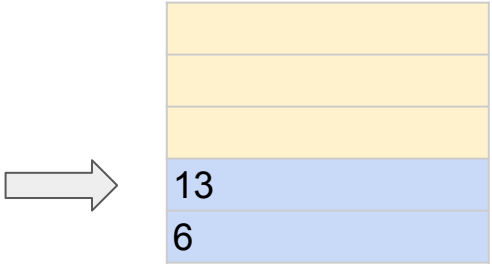
А если бы все было немного сложнее?

23*67++

Output Queue



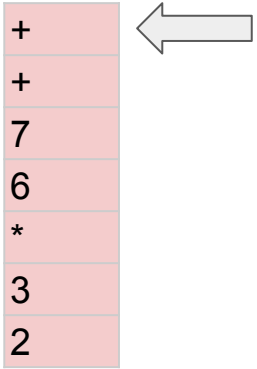
Operation Stack



А если бы все было немного сложнее?

23*67++

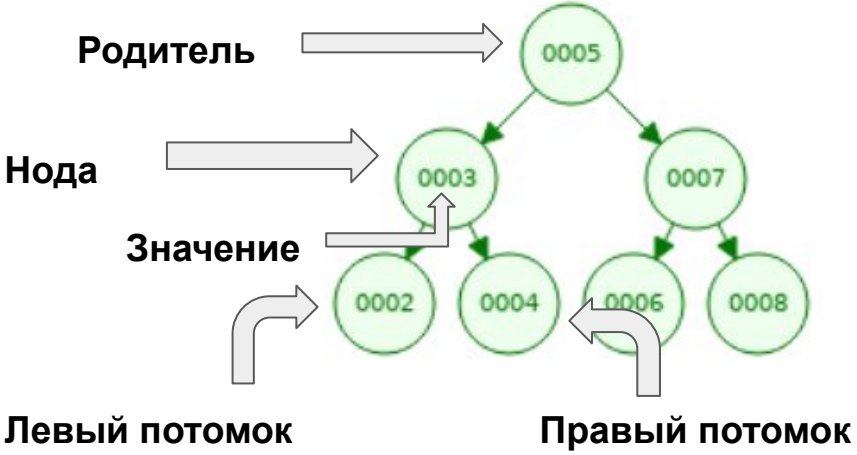
Output Queue



Operation Stack

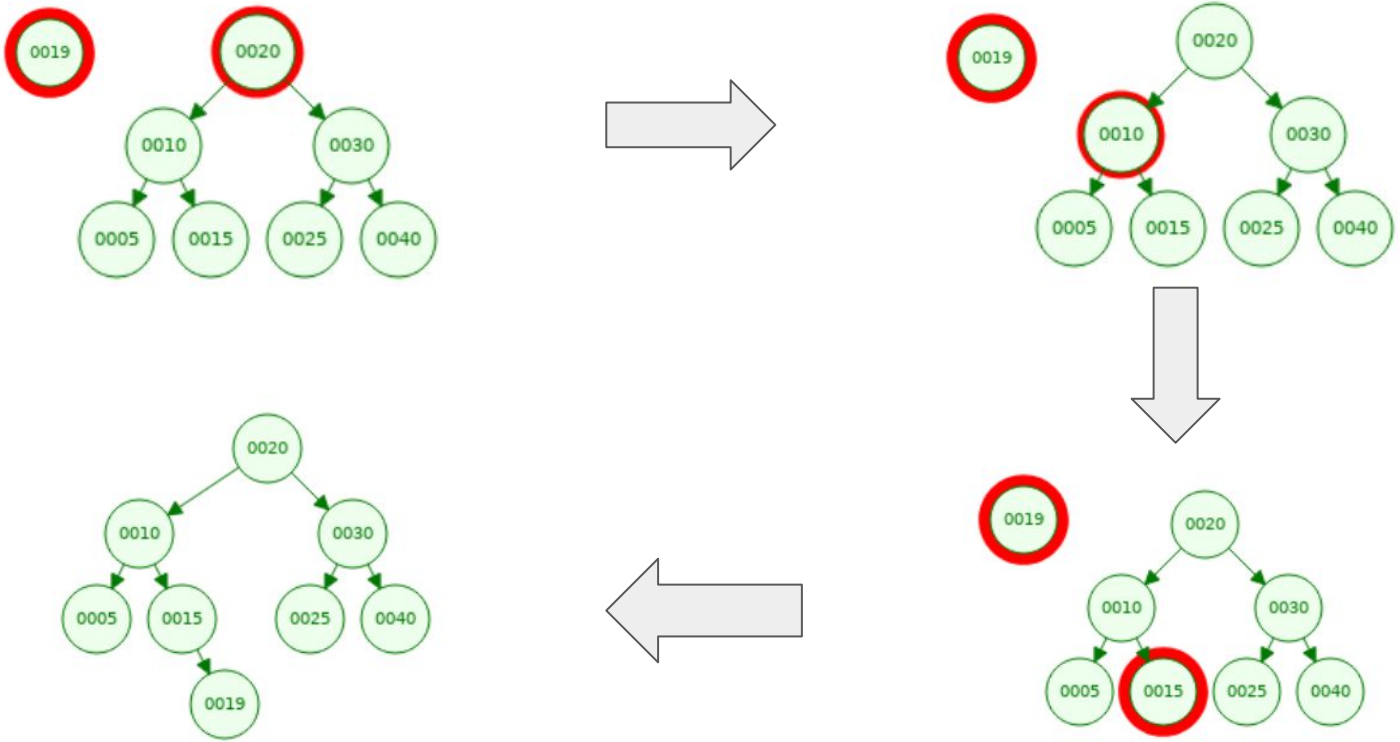
Бинарные деревья.

- Состоит из элементов (нод)
- Каждая нода хранит значение и ссылки на некоторые другие нода.
- Этими другими нодами могут быть родительская нода, левый потомок и правый потомок.



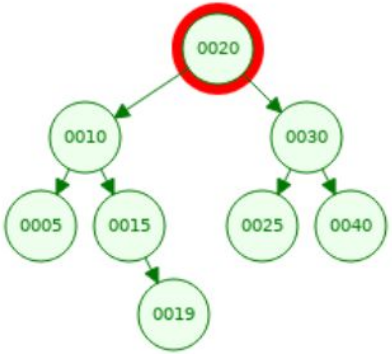
https://www.cs.usfca.edu/~galles/visualization/BST.html

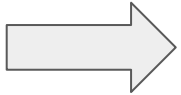
Бинарные деревья поиска. Вставка элемента

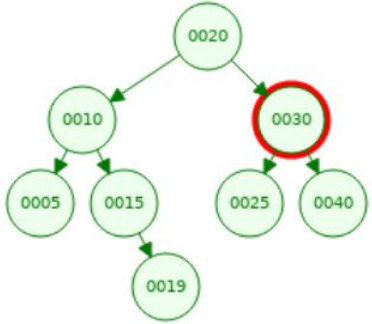


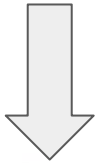
https://www.cs.usfca.edu/~galles/visualization/BST.html

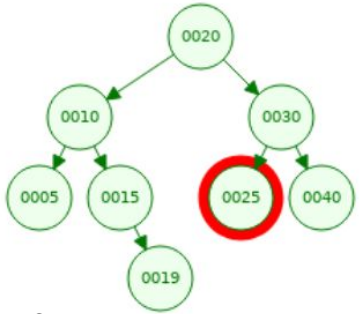
Бинарные деревья поиска. Поиск элемента









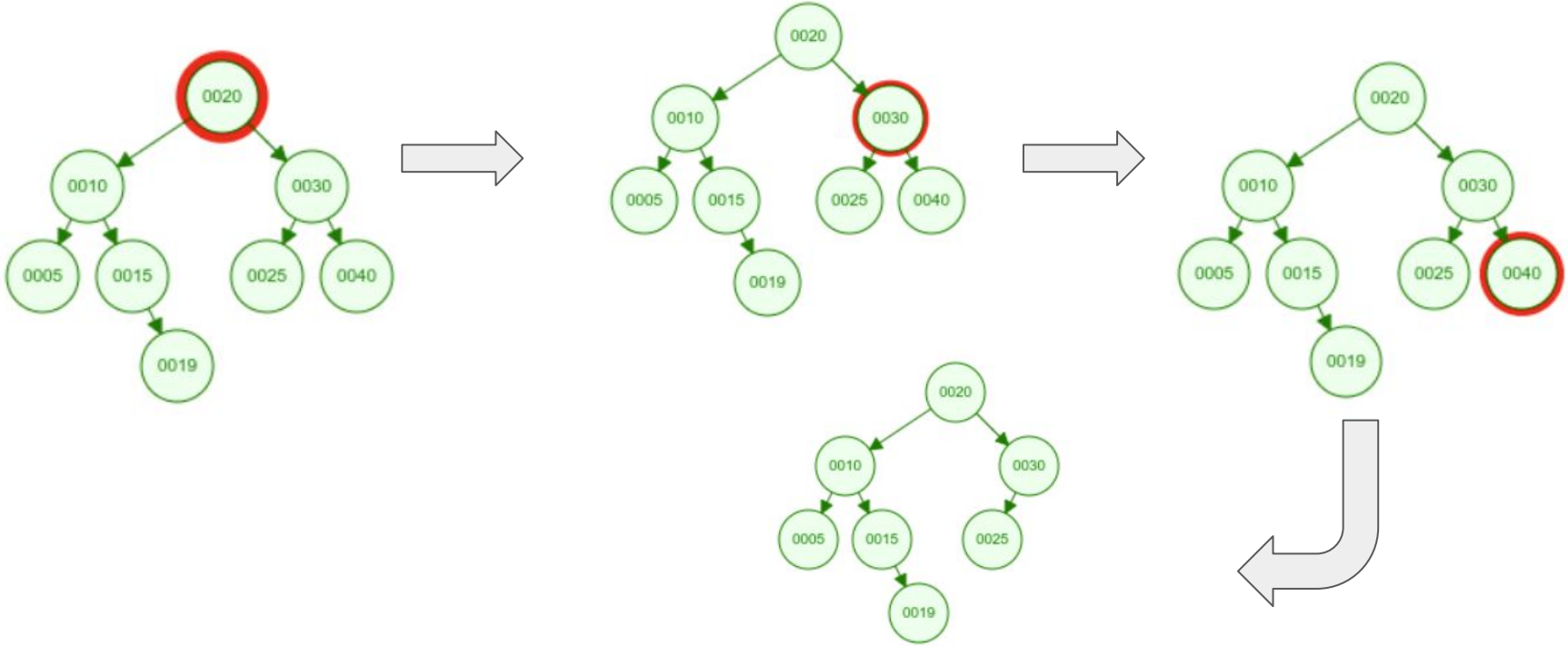


- **Ищем 25**

https://www.cs.usfca.edu/~galles/visualization/BST.html

Бинарные деревья поиска. Удаление элемента.

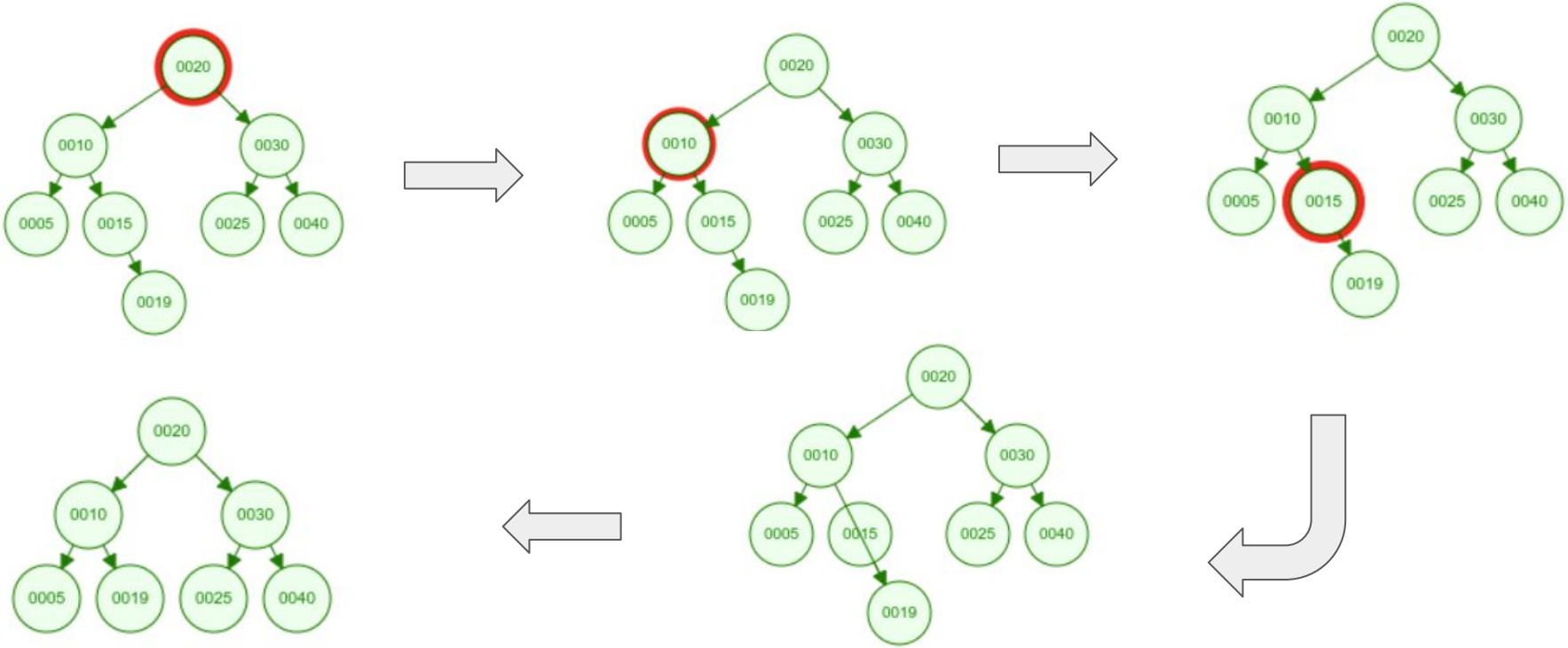
- Случай первый. Когда потомков нет. Удаляем 40.



https://www.cs.usfca.edu/~galles/visualization/BST.html

Бинарные деревья поиска. Удаление элемента.

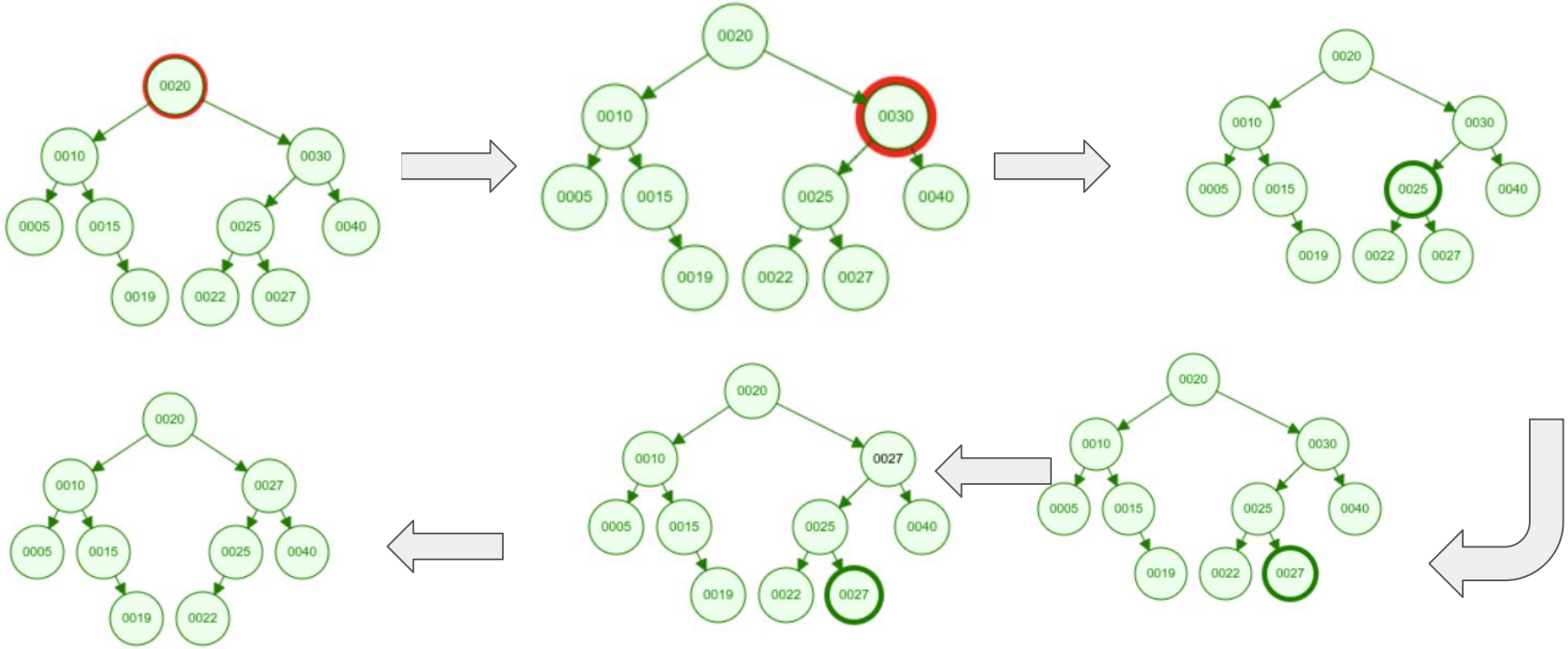
- Случай второй. Когда когда есть один потомок. Удаляем 15.



https://www.cs.usfca.edu/~galles/visualization/BST.html

Бинарные деревья поиска. Удаление элемента.

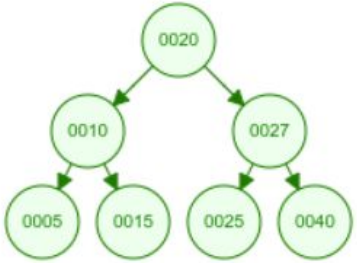
- Случай третий. Когда потомков два. Удаляем 30.



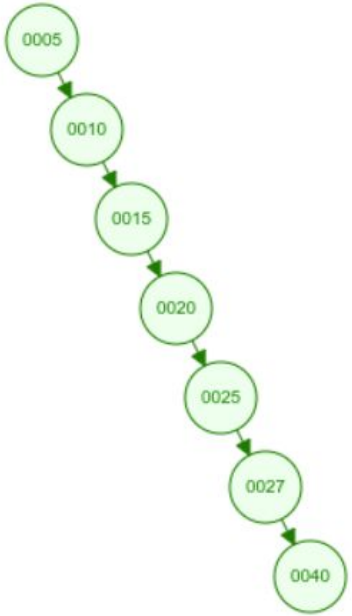
https://www.cs.usfca.edu/~galles/visualization/BST.html

Время работы Бинарного дерева поиска.

	В среднем случае	В худшем случае
Добавление элемента	O(logN)	O(N)
Поиск элемента	O(logN)	O(N)
Удаление элемента	O(logN)	O(N)



Лучший случай



Худший случай

https://www.cs.usfca.edu/~galles/visualization/BST.html