РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

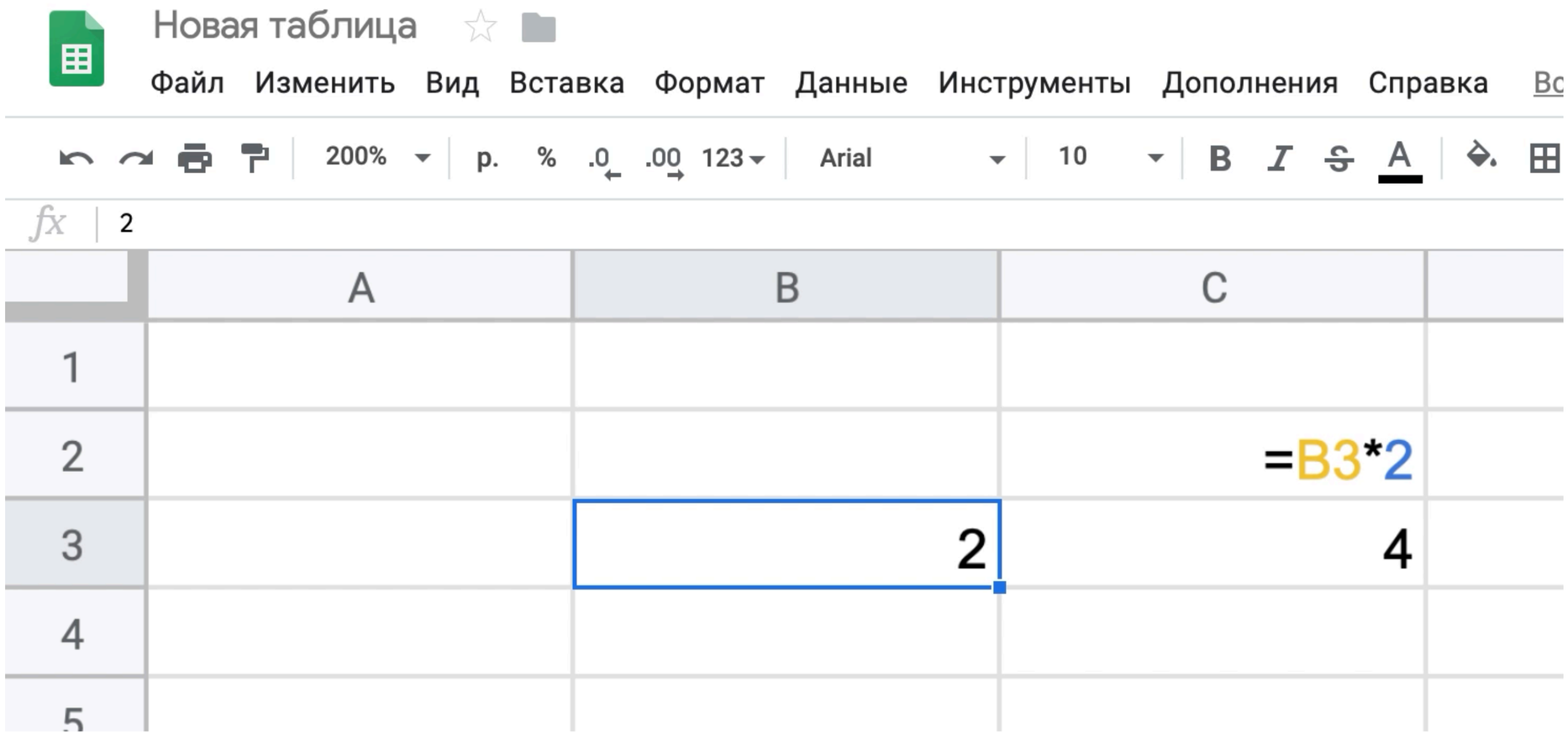
Александр Шмидт

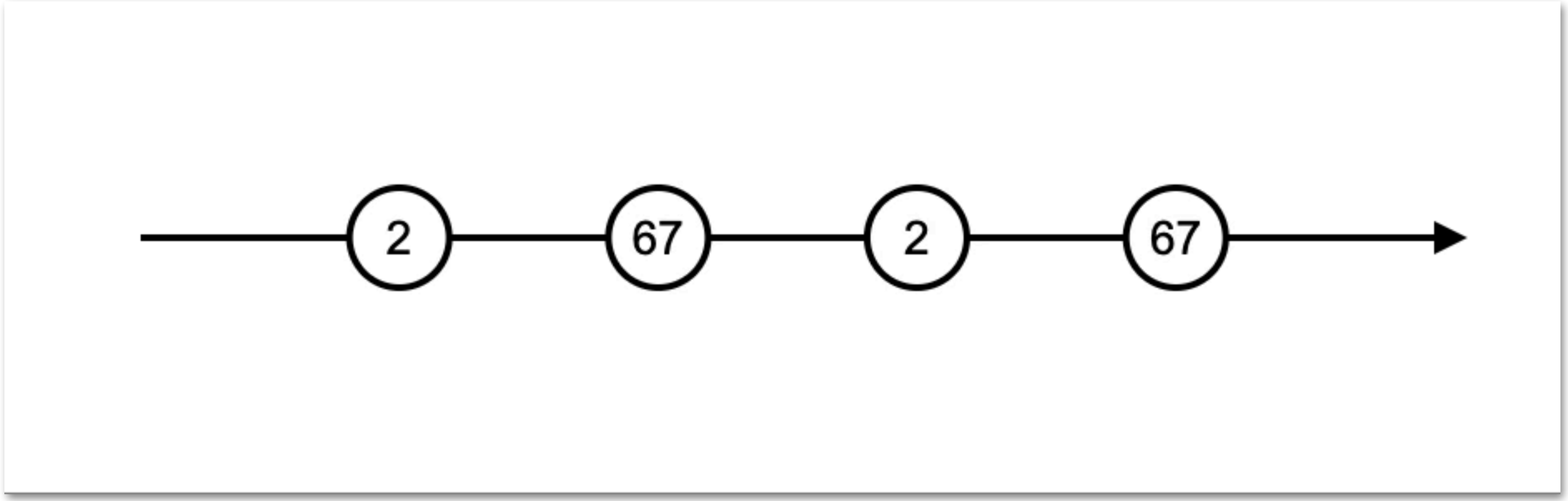
О докладчике

/chupans/CppRussia2019 — примеры

«Реактивное программирование — парадигма программирования, ориентированная на потоки данных и распространение изменений»

— wikipedia.org


Новая таблица
Файл Изменить Вид Вставка Формат Данные Инструменты Дополнения Справка
200% р. % .0 .00 123 Arial 10
fx 2
A B C
1
2 =B3*2
3 2 4
4
5


2
67
2
67

PULL

```
1.template<class T>
2.class Iterator
3.{
4.     T element();
5.     void moveNext();
6.     bool isDone();
7.};
```

PUSH

```
1.template <class T>
2.class IObserver
3.{
4.     void onNext(T);
5.     void onError(Error *);
6.     void onCompleted();
7.}
8.
9.template <class T>
10.class IObservable
11.{
12.     void subscribe(IObserver<T> observer);
13.}
```

RxCpp

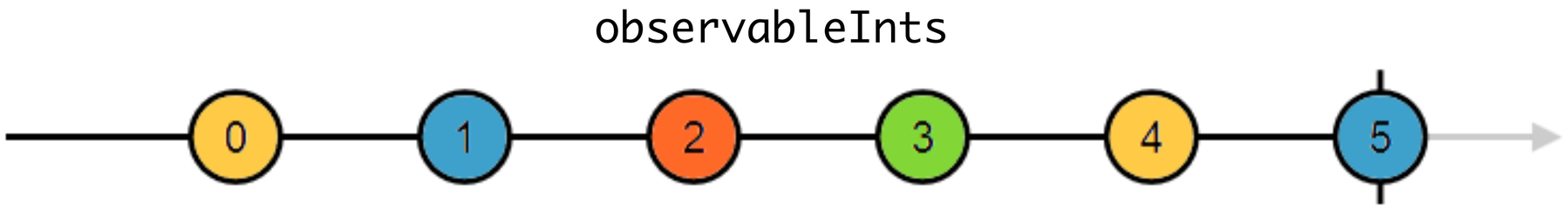
```
1.vector<string> words = { "Hello", ", ", "world", "!" };
2.
3.auto wordsStream = observable<>::iterate(words);
4.
5.wordsStream.subscribe([](string str)
6.{
7.    cout << str;
8.});
```

```
Hello, world!
_
```

std::ranges

```
1.vector<int> ints {0, 1, 2, 3, 4, 5};
2.auto even = [](int i){ return 0 == i % 2; };
3.auto square = [](int i) { return i * i; };
4.auto observableInts = observable::iterate(ints);
5.
6.observableInts.filter(even)
7.               .map(square)
8.               .subscribe([](int i)
9.{
10.  cout << i << ' ';
11.});
```

```
0 4 16
_
```


observableInts
0
1
2
3
4
5

Rx operators

Creation Observables
from
interval
of
timer

Conditional Operators
defaultIfEmpty
every
sequenceEqual

Combination Operators
combineLatest
concat
merge
race
startWith
withLatestFrom
zip

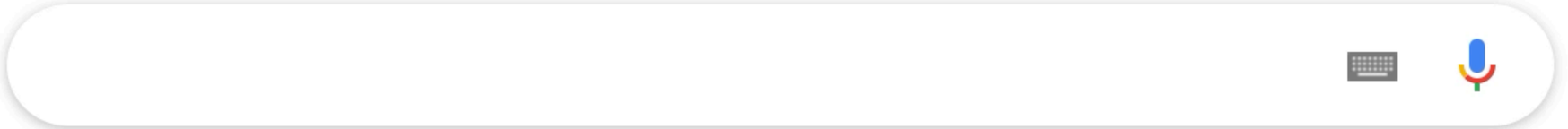
Filtering Operators
debounceTime
debounce
distinct
distinctUntilChanged
elementAt
filter
find
findIndex
first
ignoreElements
last
sample
skip
skipUntil
skipWhile
take
takeLast
takeUntil
takeWhile
throttle
throttleTime

Mathematical Operators
count
max
min
reduce

Transformation Operators
buffer
bufferCount
bufferTime
bufferToggle
bufferWhen
concatMap
concatMapTo
map
mapTo
mergeMap
mergeMapTo
pairwise
pluck
repeat
scan
switchMap
switchMapTo

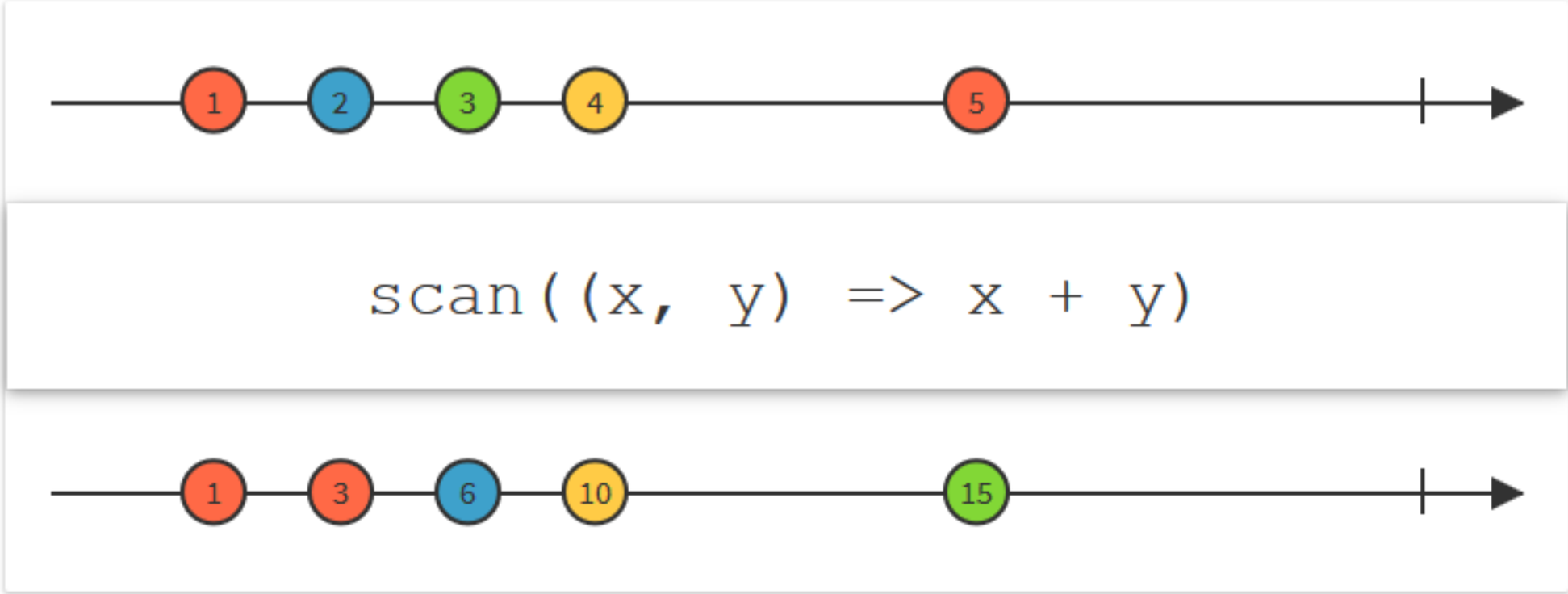
Utility Operators

delay
delayWhen

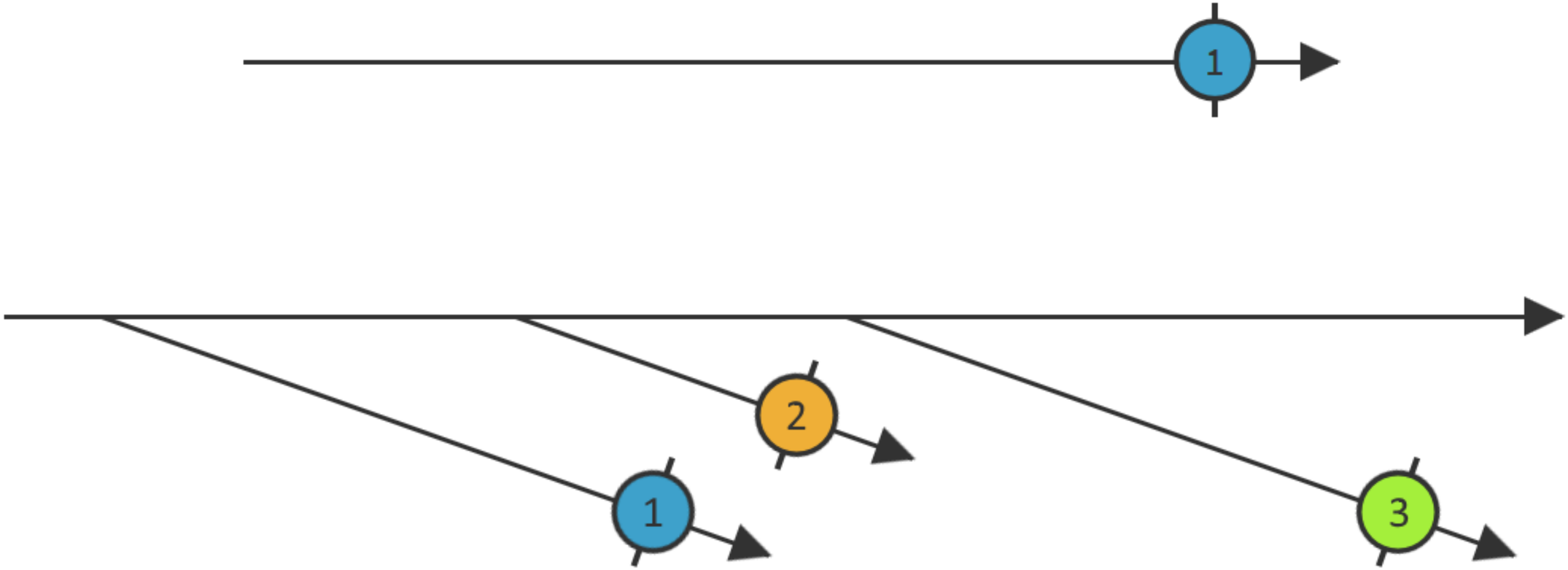


Поиск в Google Мне повезёт!

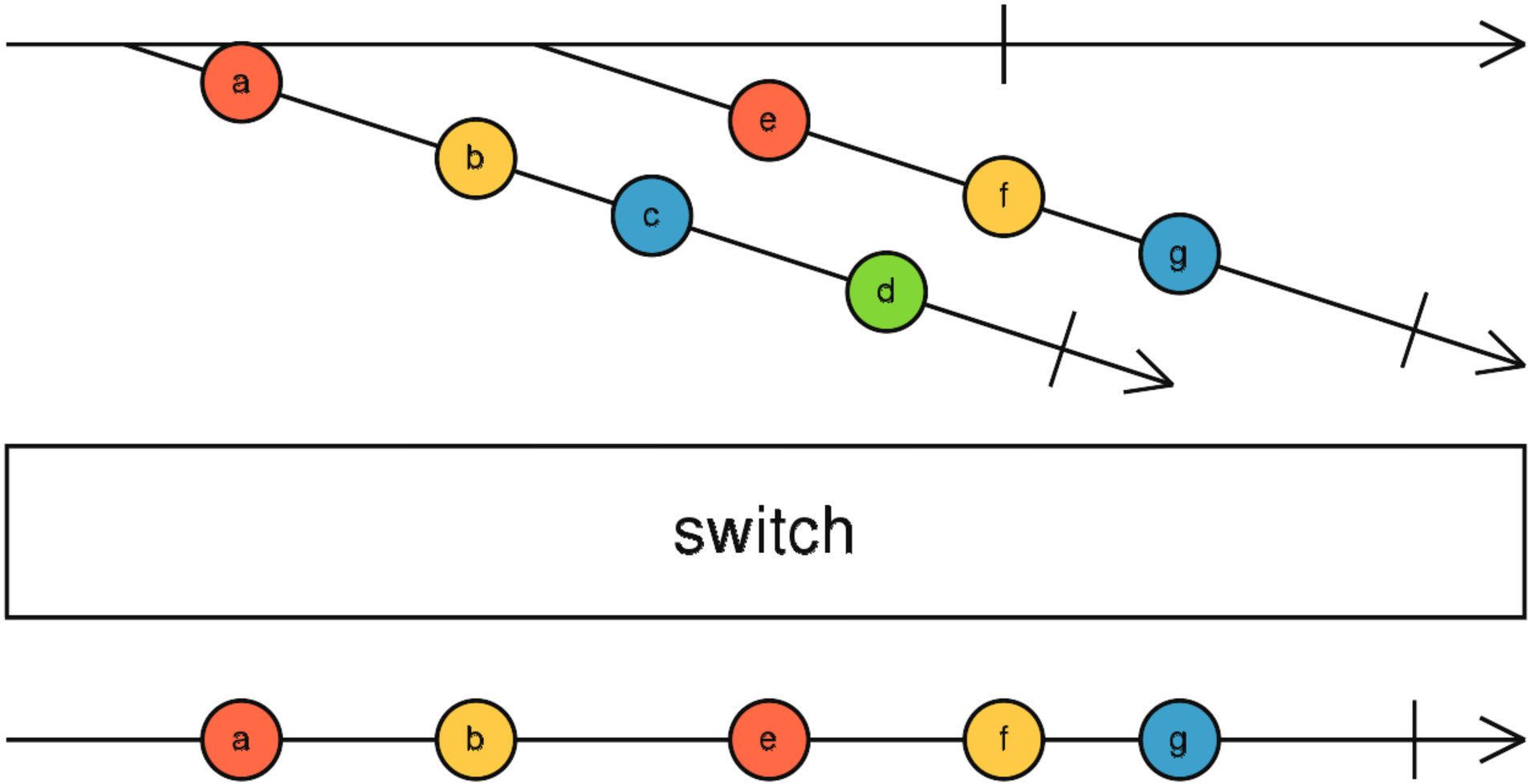
```
1.auto symbols = textEdit.getSymbolsStream();
2.
3.auto strings = symbols.scan(string{}, [](string seed, char symbol)
4.{
5.  return seed + symbol;
6.});
```


1
2
3
4
5
scan((x, y) => x + y)
1
3
6
10
15

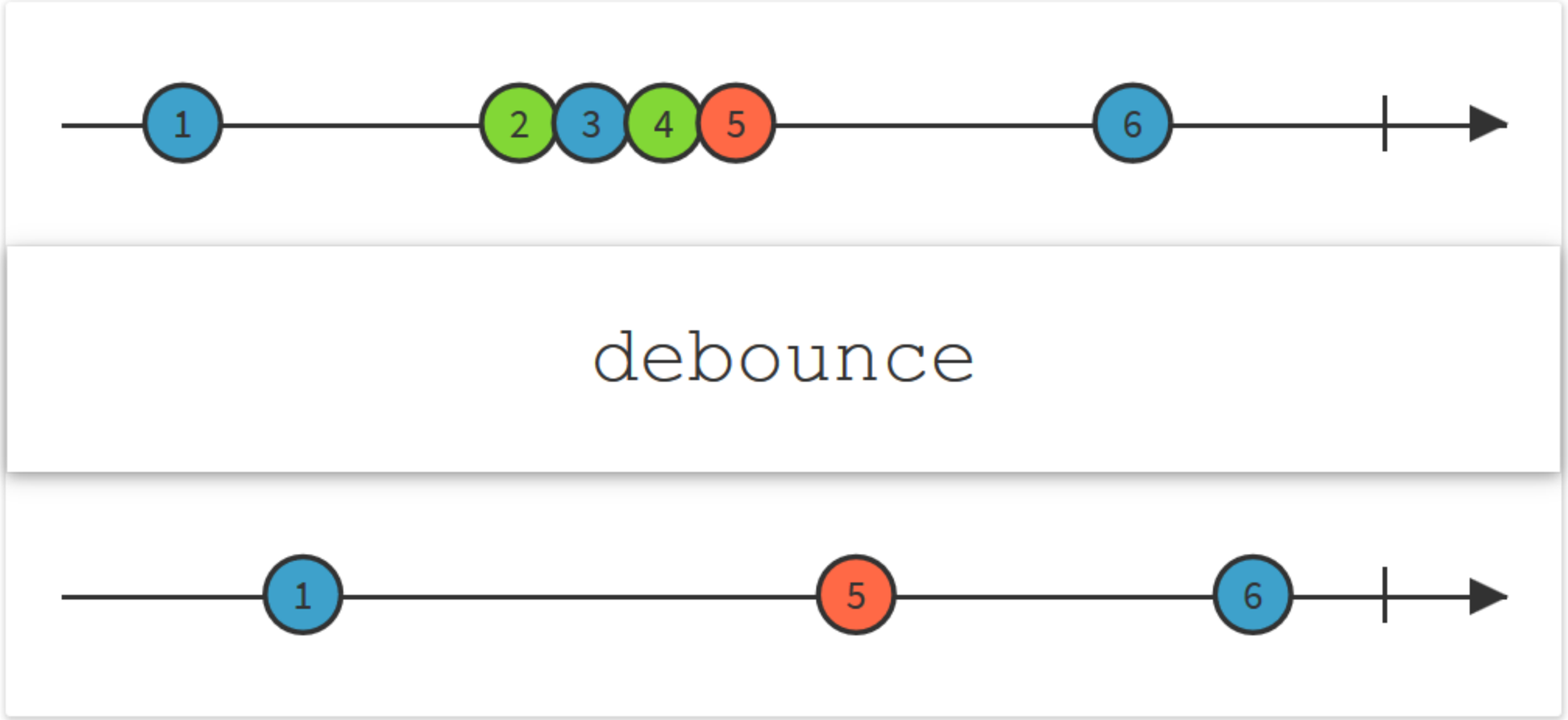
```
1.auto symbols = textEdit.symbolsStream_;
2.
3.auto strings = symbols.scan(string{}, [](string seed, char symbol)
4.{
5.    return seed + symbol;
6.});
7.
8.auto allHints = strings.map([](string input)
9.{
10.    future<string> hint = netClient.getHintFor(input);
11.
```


1
2
1
3

```
1.auto symbols = textEdit.symbolsStream_;
2.
3.auto strings = symbols.scan(string{}, [](string seed, char symbol)
4.{
5.    return seed + symbol;
6.});
7.
8.auto allHints = strings.map([](string input)
9.{
10.    future<string> hint = netClient.getHintFor(input);
11.
12.    return observable::from_future(hint);
13.});
```


a
b
c
d
e
f
g
switch
a
b
e
f
g

```
1.auto symbols = textEdit.symbolsStream_;
2.
3.auto strings = symbols.scan(string{}, [](string seed, char symbol)
4.{
5.    return seed + symbol;
6.});
7.
8.auto allHints = strings.map([](string input)
9.{
10.    future<string> hint = netClient.getHintFor(input);
11.
12.    return observable::from_future(hint);
13.});
14.
15.auto hints = allHints.switch_on_next();
16.
17.hints.subscribe(display, handleError);
```

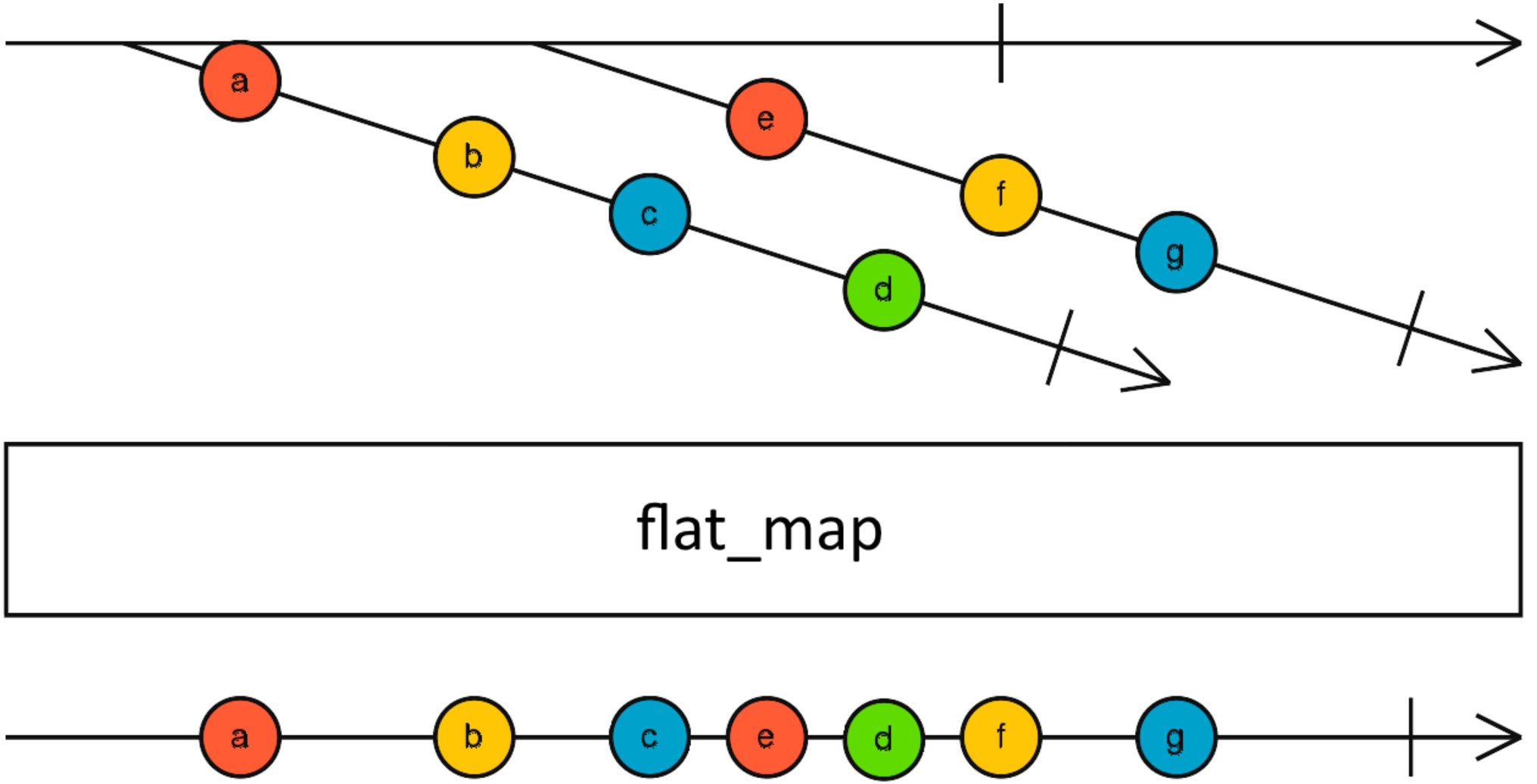

1
2
3
4
5
6
debounce
1
5
6

```
1.auto symbols = textEdit.symbolsStream_;
2.
3.auto strings = symbols.scan(string{}, [](string seed, char symbol)
4.{
5.    return seed + symbol;
6.}).debounce(300ms);
7.
8.auto allHints = strings.map([](string input)
9.{
10.    future<string> hint = netClient.getHintFor(input);
11.
12.    return observable::from_future(hint);
13.});
14.
15.auto hints = allHints.switch_on_next();
16.
17.hints.subscribe(display, handleError);
```

```
1.auto values = observable::create(...);
2.values.observe_on(threadPoolScheduler);
3.values.subscribe_on(guiThreadScheduler);
```

```
1.auto paths = observable<std::path> ... ;
2.
3.paths.map(processFile).observe_on(threadPoolScheduler)
4.      .subscribe(print);
```

```
Processing file 1...
Done with 1
Processing file 2...
Done with 2
Processing file 3...
Done with 3
```


a
b
c
d
e
f
g
flat_map
a
b
c
e
d
f
g

```
1.auto paths = observable<path> ... ;
2.
3.auto parallelPaths = paths.flat_map([](path p)
4.{
5.    future<Data> dataFuture = processor.processFile(p);
6.    auto processedFile = observable::from_future(dataFuture);
7.
8.    return processedFile.subscribe_on(threadPoolScheduler);
9.});
10.
11.parallelPaths.subscribe(print);
```

```
Processing file 1...
Processing file 2...
Processing file 3...
Done with 2
Done with 1
Done with 3
```

```
1.auto randoms = observable::create([](subscriber dest)
2.{
3.  dest.on_next(std::rand() % 100);
4.  dest.on_completed();
5.});
6.
7.randoms.subscribe(print);
8.randoms.subscribe(print);
```

```
22
60
_
```

Observables

COLD

Отдаёт значения по мере необходимости

Примеры:
- Данные из памяти
- Базы данных
- Веб запросы

HOT

Отдаёт значения по мере появления

Примеры:
- Пользовательский ввод
- Данные с датчиков
- Системные сообщения

```
1.auto randoms = observable::create([](subscriber dest)
2.{
3.  dest.on_next(std::rand() % 100);
4.  dest.on_completed();
5.}).publish();
6.
7.randoms.subscribe(print);
8.randoms.subscribe(print);
```

```
40
40
_
```

```
1.observable::interval(500ms, testScheduler)
2.             .delay(500ms, testScheduler)
3.             .take(10)
4.             .subscribe(...);
5.
6.testScheduler.advance_by(10s);
```

Производительность

100’000’000 элементов

	Cold	Hot
For loop	~3 ms	~3 ms
Observable	~800 ms	~3400 ms


Applied Duality
Microsoft
NETFLIX
GitHub
SOUNDCLOUD
Trello
treehouse
SeatGeek
odvoo
Couchbase
futurice
O.C.TANNER
airbnb

Функциональный

Лаконичный

Тестируемый

Со встроенной обработкой ошибок

Q&A

Всем спасибо