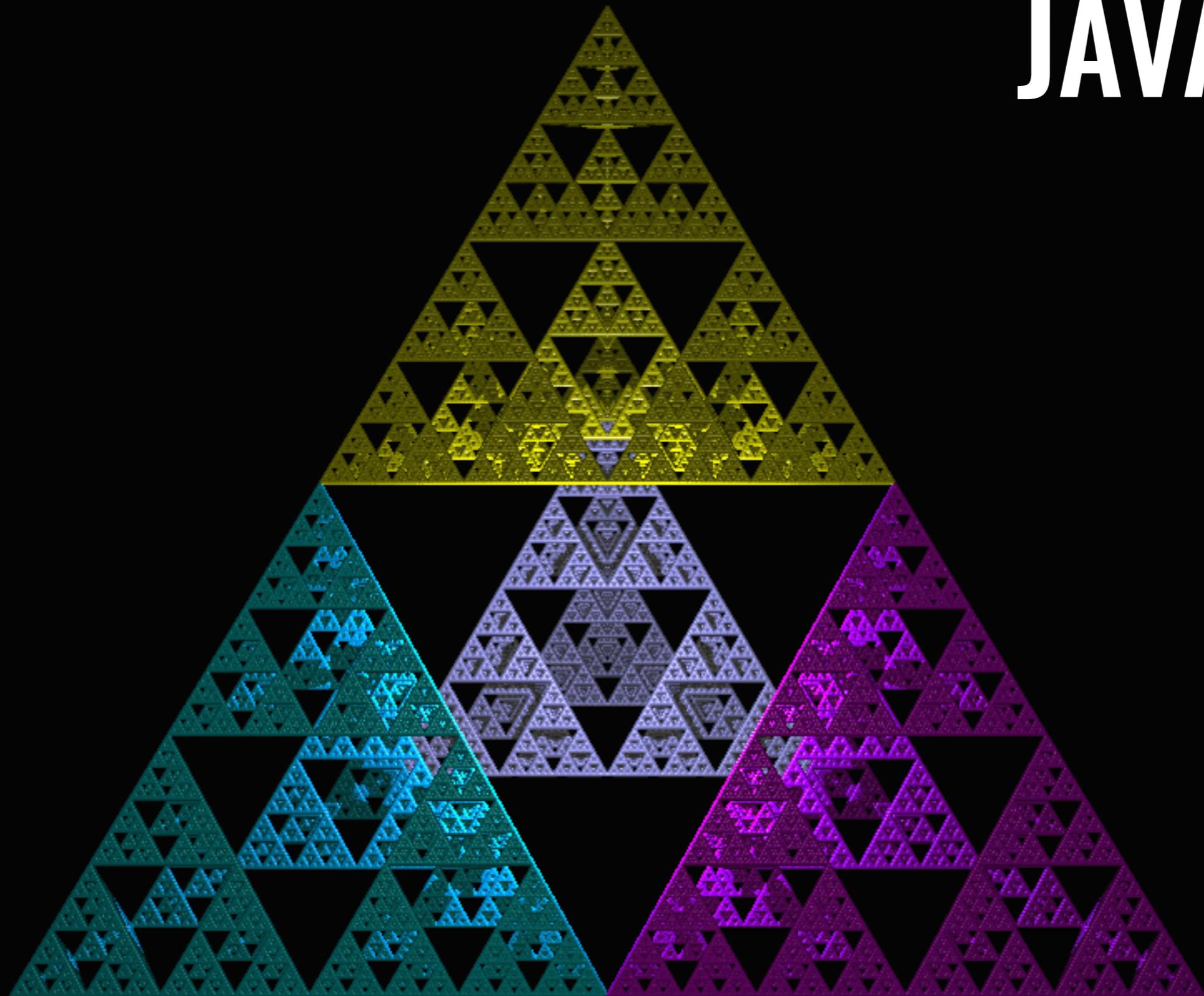


JAVASCRIPT

И

D3

СТРУКТУРЫ ДАННЫХ JAVASCRIPT



Объекты

```
01: var human = {  
02:   firstName: 'John',  
03:   secondName: 'Doe',  
04:   age: 25,  
05:   name: function () {  
06:     return this.firstName + ' ' + this.secondName;  
07:   },  
08:   greet: function () {  
09:     return 'Hello, I am ' + this.name();  
10:   }  
11: }
```

Объекты

```
01: var human = {  
02:   firstName: 'John',  
03:   secondName: 'Doe',  
04:   age: 25,  
05:   name: function () {  
06:     return this.firstName + ' ' + this.secondName;  
07:   },  
08:   greet: function () {  
09:     return 'Hello, I am ' + this.name();  
10:   }  
11: }
```

Ключ

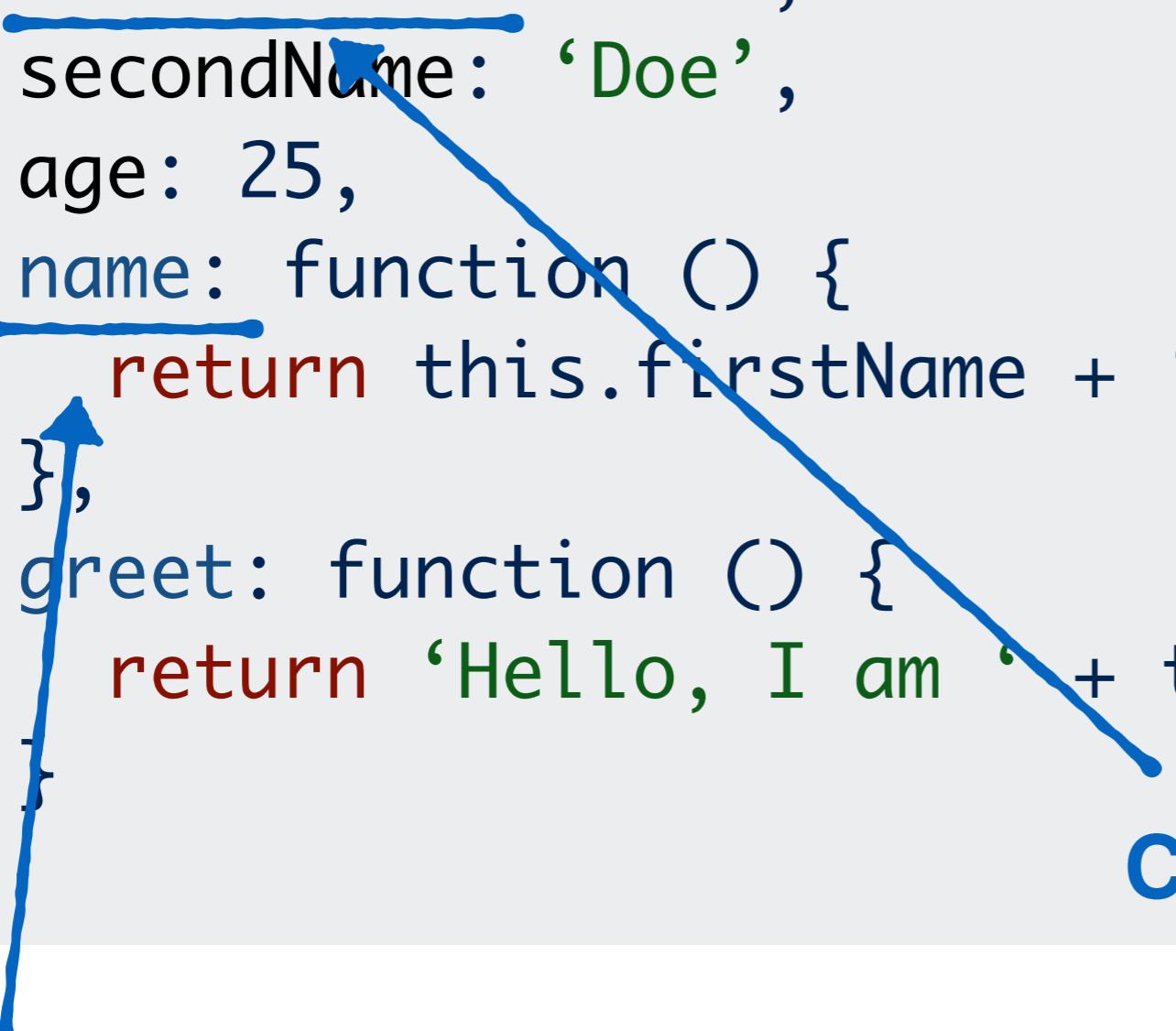
Значение

Объекты

```
01: var object = {  
02:   '4eburek-2016': null, нельзя  
03:  
04:   '4eburek-2016': null, можно  
05: }
```

Объекты

```
01: var human = {  
02:   firstName: 'John',  
03:   secondName: 'Doe',  
04:   age: 25,  
05:   name: function () {  
06:     return this.firstName + ' ' + this.secondName;  
07:   },  
08:   greet: function () {  
09:     return 'Hello, I am ' + this.name();  
10:   }  
11: }
```



Метод

Свойство

Объекты

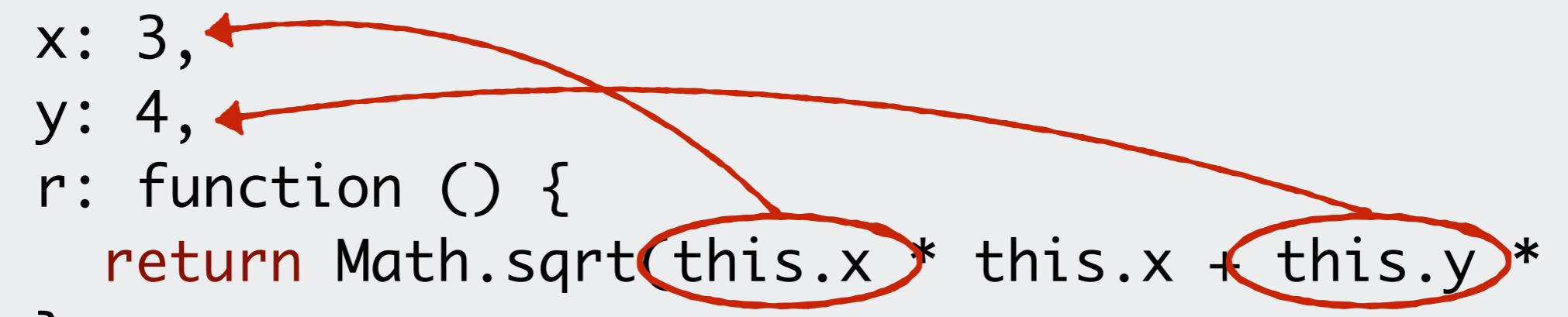
```
01: var human = { ... }  
02:  
03: human.firstName // -> John Doe  
04:  
05: human['firstName']; // -> John Doe  
06:  
07: human.name; // -> function () { ... }  
08:  
09: human.name(); // -> 'John Doe'  
10:  
11: human.firstName = 'Jake';  
12:  
13: human.name(); // -> 'Jake Doe'
```

Объекты: контекст

```
01: var vector = {  
02:   x: 3,  
03:   y: 4,  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5
```

Объекты: контекст

```
01: var vector = {  
02:   x: 3, ←  
03:   y: 4, ←  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5
```



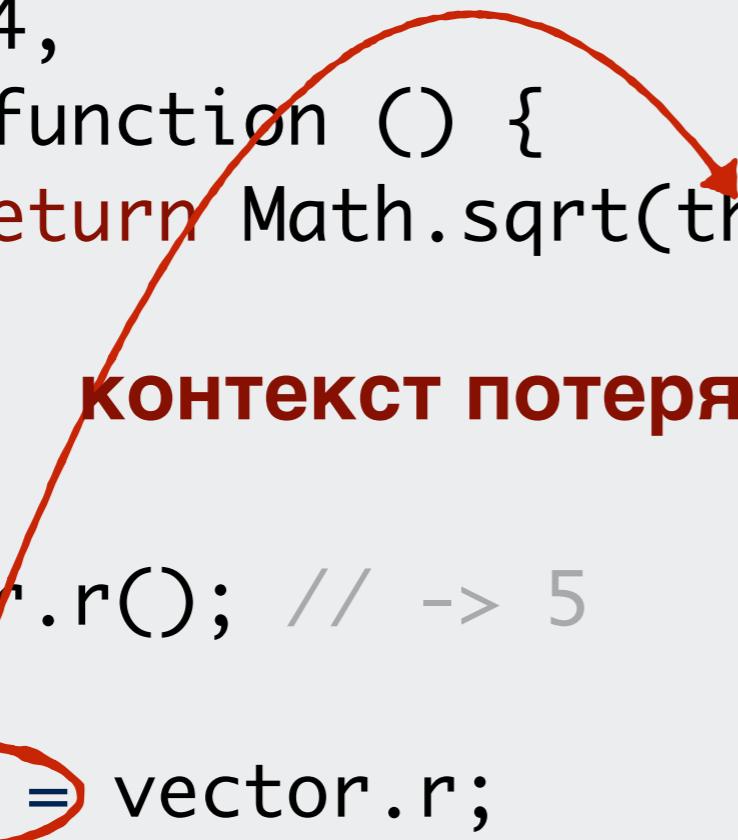
Объекты: контекст

```
01: var vector = {  
02:   x: 3,  
03:   y: 4,  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5  
10:  
11: var r = vector.r;  
12:  
13: r(); // -> NaN
```

Объекты: контекст

```
01: var vector = {  
02:   x: 3,  
03:   y: 4,  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5  
10:  
11: var r = vector.r;  
12:  
13: r(); // -> NaN
```

контекст потерялся



Объекты: контекст

```
01: var vector = {  
02:   x: 3,  
03:   y: 4,  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5  
10:  
11: var r = vector.r;  
12:  
13: r(); // -> NaN
```

Новый контекст

Объекты: контекст

```
01: var vector = {  
02:   x: 3,  
03:   y: 4,  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5  
10:  
11: var r = vector.r;  
12:  
13: r(); // -> NaN
```

Глобальный контекст

Вызов в глобальном контексте

Объекты: контекст

```
01: var vector = {  
02:   x: 3,  
03:   y: 4,  
04:   r: function () {  
05:     return Math.sqrt(this.x * this.x + this.y * this.y);  
06:   }  
07: }  
08:  
09: vector.r(); // -> 5  
10:  
11: var r = vector.r.bind(vector); // связывание контекста  
12:  
13: r(); // -> NaN
```

Массивы

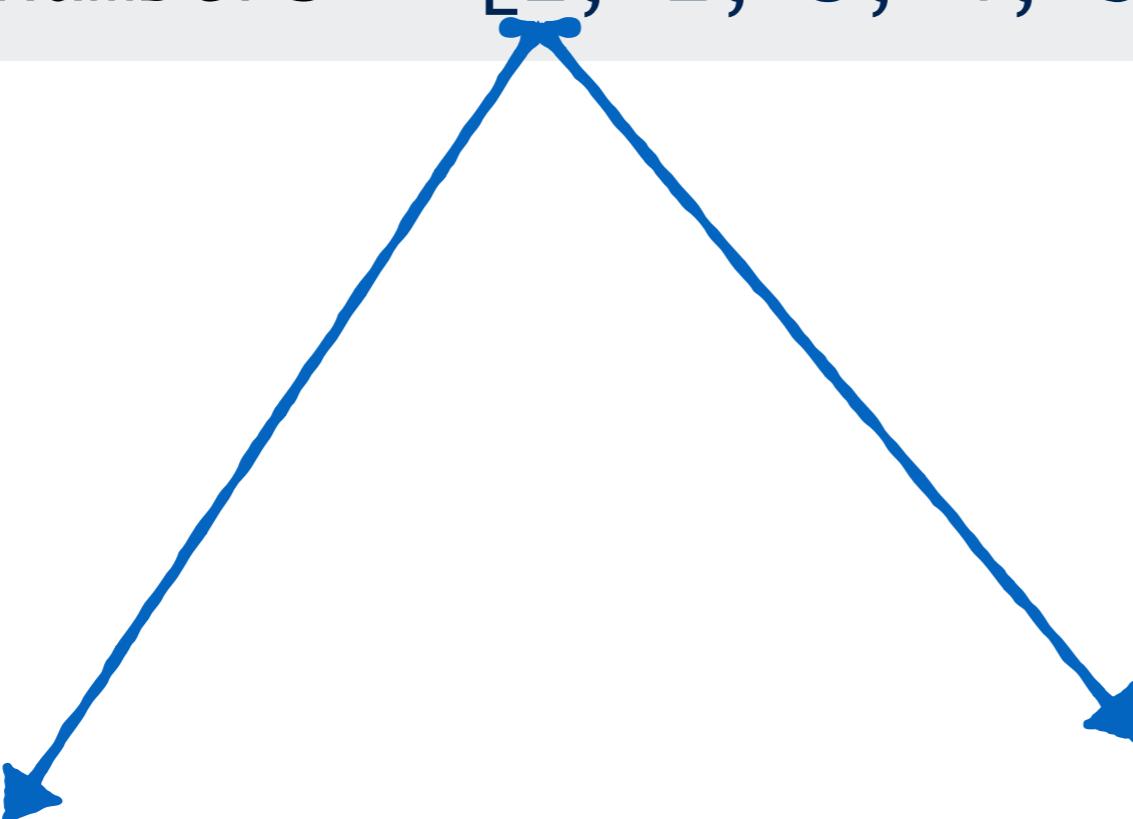
```
01: var numbers = [1, 2, 3, 4, 5];
```

Массивы

```
01: var numbers = [1, 2, 3, 4, 5];
```

Значение: 1

Индекс: 0



Массивы

```
01: var numbers = [1, 2, 3, 4, 5];
02:
03: numbers[0]; // -> 1
04:
05: numbers.length; // -> 5
06:
07: numbers[4] = 'Hello';
08:
09: numbers; // -> [1, 2, 3, 4, 'Hello'];
```

Массивы

вообще-то это объекты

```
01: var numbers = [1, 2, 3, 4, 5];  
02:  
03: var numbers = {  
04:   0: 1,  
05:   1: 2,  
06:   2: 3,  
07:   3: 4,  
08:   4: 5,  
09:   length: 5  
10: }
```

Массивы

вообще-то это объекты

```
01: var numbers = [1, 2, 3, 4, 5];  
02:  
03: var numbers = {  
04:   0: 1,  
05:   1: 2,  
06:   2: 3,  
07:   3: 4,  
08:   4: 5,  
09:   length: 5  
10: }
```

- порядок (все ключи - числа)
- длина (есть ключ length)

Метод	Сигнатура	Описание
push	a.push(a, [b, c, ...])	Вставляет в конец массива указанные аргументы. Возвращает новую длину массива.
pop	a.pop()	Удаляет последний элемент массива. Возвращает этот элемент.
join	a.join([str])	Приводит массив к строке, объединяя элементы массива строкой str (по умолчанию - запятая)
reverse	a.reverse()	Переворачивает порядок массива
sort	a.sort([comprator])	Сортирует массив используя компратор (по-умолчанию лексикографический компратор)
slice	a.slice([from, to])	Возвращает кусок массива от индекса from (по-умолчанию 0), до индекса to (по-умолчанию длина).
splice	a.splice(point,N)	Вырезает в массиве N элементов начиная с элемента с индексом point. Возвращает вырезанный кусок.

Метод	Сигнатура	Описание
<code>shift</code>	<code>a.shift()</code>	Удаляет первый элемент из массива и возвращает его значение.
<code>unshift</code>	<code>a.unshift([a1, a2, ...])</code>	Добавляет один или более элементов в начало массива и возвращает новую длину массива.
<code>indexOf</code>	<code>a.indexOf(element)</code>	Возвращает первый индекс, по которому данный элемент может быть найден в массиве или -1, если такого индекса нет.
<code>lastIndexOf</code>	<code>a.lastIndexOf(element)</code>	Тоже самое, что <code>indexOf</code> , только считает с конца массива
<code>concat</code>	<code>a.concat(elementOrArray)</code>	Возвращает новый массив, состоящий из массива, на котором он был вызван, соединённого с другими массивами и/или значениями, переданными в аргументах.

Работа со структурами данных

- перебор
- фильтрация
- трансформация
- сортировка
- сворачивание

for... in

```
01: var object = { ... };  
02:  
03: var copy = {};  
03:  
03: for (var property in object) {  
04:     copy[property] = object[property];  
05: }
```

for... in

```
01: var object = { ... };  
02:  
03: var copy = {};  
04:  
05: for (var property in object) {  
06:   if (object.hasOwnProperty(property)){  
07:     copy[property] = object[property];  
08:   }  
09: }
```

forEach, map, filter

```
01: var numbers = [1, 2, 3, 4, 5];
02:
03: numbers.forEach(function (number) {
04:   console.log(number);
05: });
06:
07: numbers.map(function (number) {
08:   return number * 2;
09: });
10:
11: numbers.filter(function (number) {
12:   return number > 2;
13: });
```

some, every

```
01: var numbers = [1, 2, 3, 4, 5];
02:
03: numbers.some(function (number) {
04:   return number == 3;
05: }); // -> true
06:
07: numbers.every(function (number) {
08:   return number == 3;
09: }); // -> false
```

reduce

```
01: var numbers = [1, 2, 3, 4, 5];
02:
03: var sum = numbers.reduce(function (sum, number) {
04:   return sum + number;
05: }); // -> 15
06:
07: var max = numbers.reduce(function (max, number) {
08:   return number > max ? number : max;
09: }); // -> 5
```

reduce =

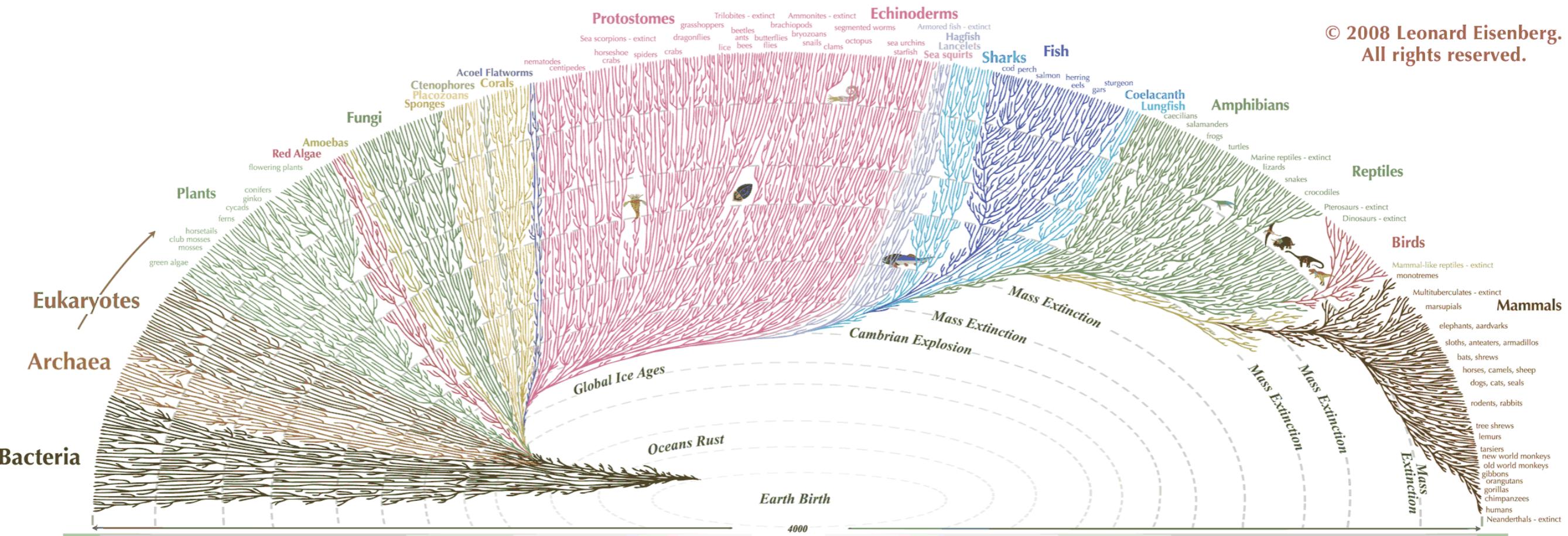


ООП

ну, или типа того



© 2008 Leonard Eisenberg.
All rights reserved.



All the major and many of the minor living branches of life are shown on this diagram, but only a few of those that have gone extinct are shown. Example: Dinosaurs - extinct

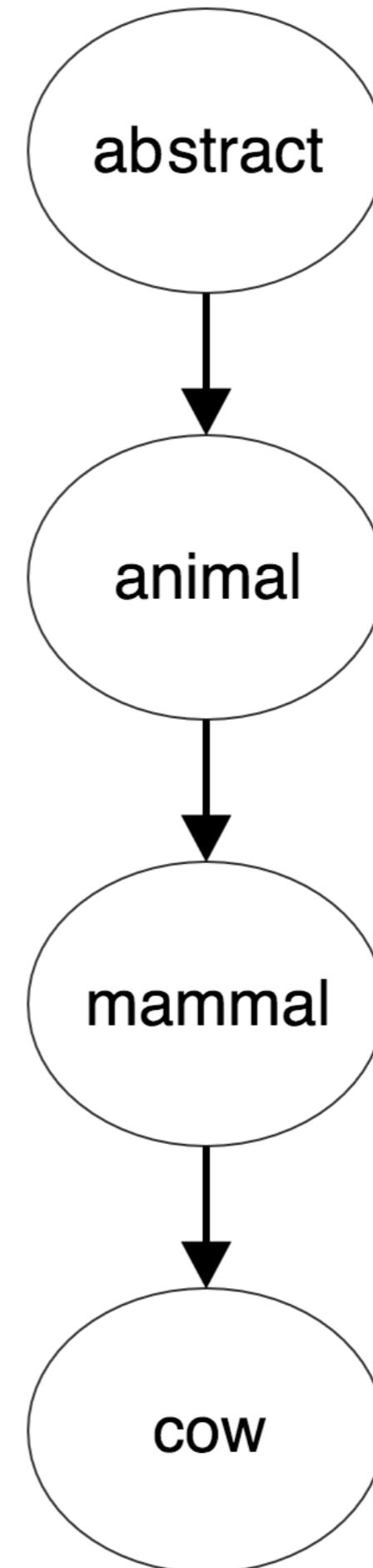
© 2008 Leonard Eisenberg. All rights reserved.
evogeneao.com

Абстрактный класс существ

Класс всех животных

Класс млекопитающих

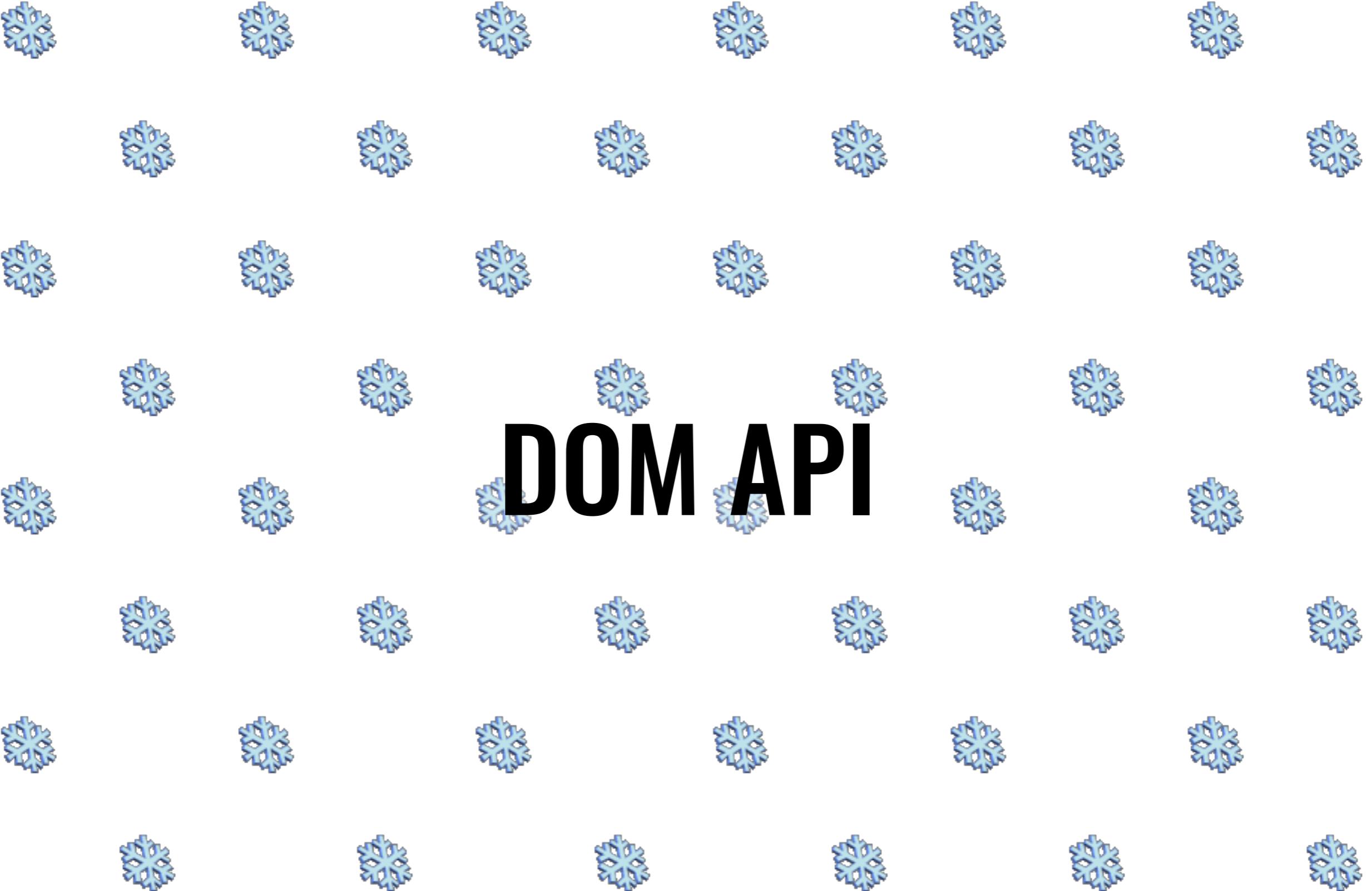
Класс коров



В JavaScript нет классов

deal with it





DOM API

ПОЗВОЛЬТЕ ПРЕПОДАТЬ ВАМ



КОЛДОВСКИЕ ШТУЧКИ!

Все начинается с документа

document

Выборка элементов

```
document.getElementById('myId')
```

```
document.getElementById('myId')
```

Возвращает DOM-элемент у которого
идентификатор равен аргументу функции

```
document.getElementsByTagName('div')
```

Возвращает коллекцию (не массив) ДОМ-элементов название тэгов, которых равно аргументу функции

```
document.getElementsByClassName('link')
```

Возвращает коллекцию (не массив) DOM-элементов, у которых есть класс название которого равно аргументу функции

```
var selector = 'body ul a.active';  
  
document.querySelector(selector);
```

Возвращает первый элементов, соответствующий
данному CSS - селектору

```
var selector = 'body ul a.active';  
  
document.querySelectorAll(selector);
```

Возвращает коллекцию (не массив) всех элементов, соответствующих данному CSS - селектору

```
var selector = 'body ul a.active';  
  
document.querySelectorAll(selector);
```

Возвращает коллекцию (не массив) всех элементов, соответствующих данному CSS - селектору

Список динамический и актуализируется по мере изменения DOM-дерева

Изменение элементов

```
01: var tags = document.getElementsByTagName('body');  
02:  
03: var body = tags[0];  
04:  
05: body.innerHTML = 'Hello, World!';
```



jQuery

- предоставляет упрощенный API взаимодействия с DOM
- берет на себя решение кросс-браузерных проблем
- упрощает работу с AJAX
- имеет встроенные методы анимации
- это действительно хорошая библиотека

jQuery

- предоставляет упрощенный API взаимодействия с DOM
- берет на себя решение кросс-браузерных проблем
- упрощает работу с AJAX
- имеет встроенные методы анимации
- это действительно хорошая библиотека

\$

jQuery

```
01: var $navLinks = $('body ul a.active');
```

jQuery

```
01: var $navLinks = $('body ul a.active');
```

Без jQuery

```
01: var items = document.querySelectorAll('ul li');
```

С jQuery

```
01: var $items = $('ul li');
```

Без jQuery

```
01: var items = document.querySelectorAll('ul li');  
02:  
03: var html = [].map.call(items, function(item){  
04:     return item.innerHTML;  
05: });
```

С jQuery

```
01: var $items = $('ul li');  
02:  
03: var html = $items.map(function(){  
04:     return $(this).html();  
05: });
```

Без jQuery

```
01: var items = document.querySelectorAll('ul li');  
02:  
03: [].forEach.call(items, function(item){  
04:     item.innerHTML = 'new value';  
05: });
```

С jQuery

```
01: var $items = $('ul li');  
02:  
03: $items.html('new value');
```

Без jQuery

```
01: var items = document.querySelectorAll('ul li');  
02:  
03: [].forEach.call(items, function(item){  
04:     item.innerHTML = 'new value';  
05: });
```

боль и унижение

С jQuery

```
01: var $items = $('ul li');  
02:  
03: $items.html('new value');
```

процветание и гармония

jQuery - это только сахар

АСИНХРОННОСТЬ



```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```

```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```

```
> for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0)  
}  
< 35  


---

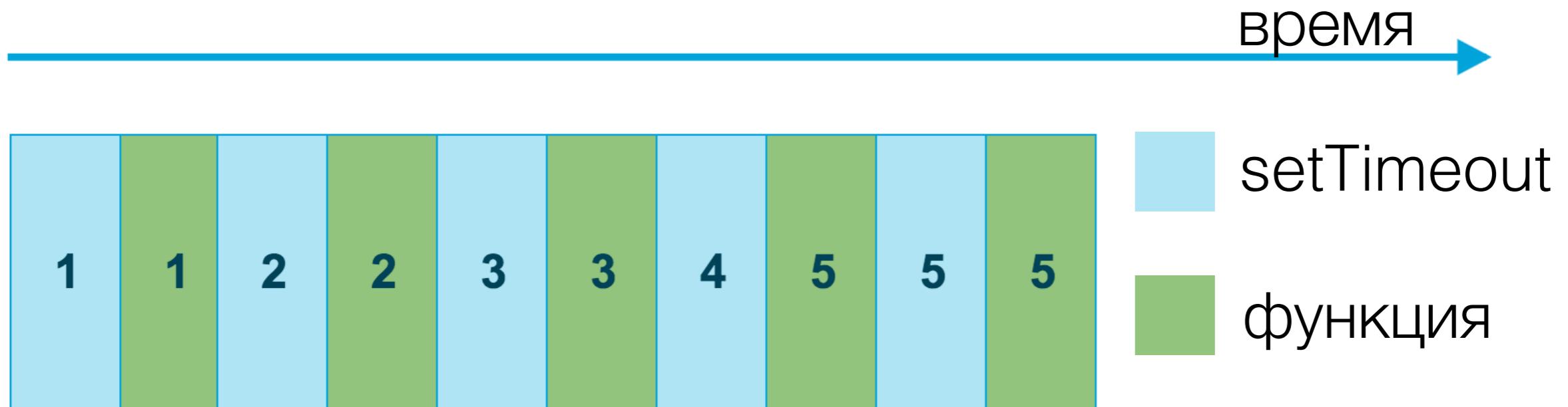
5 5  


---

>
```

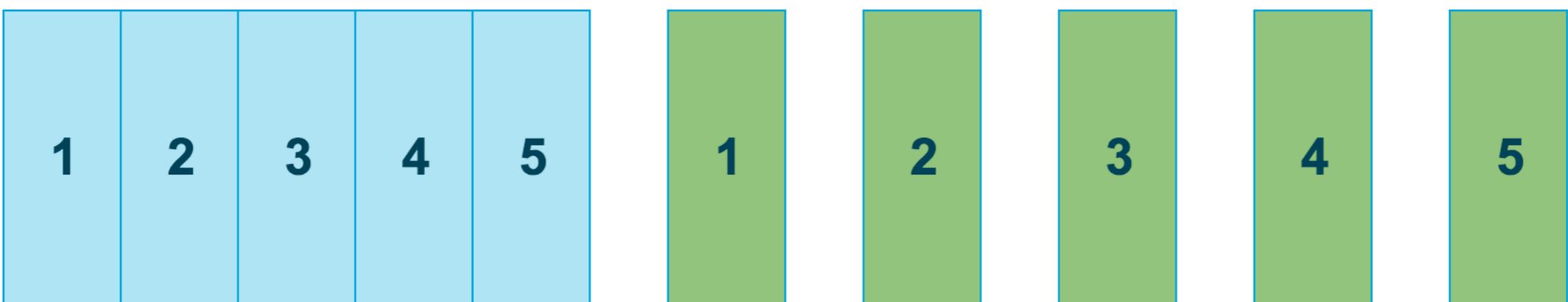


```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```



```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```

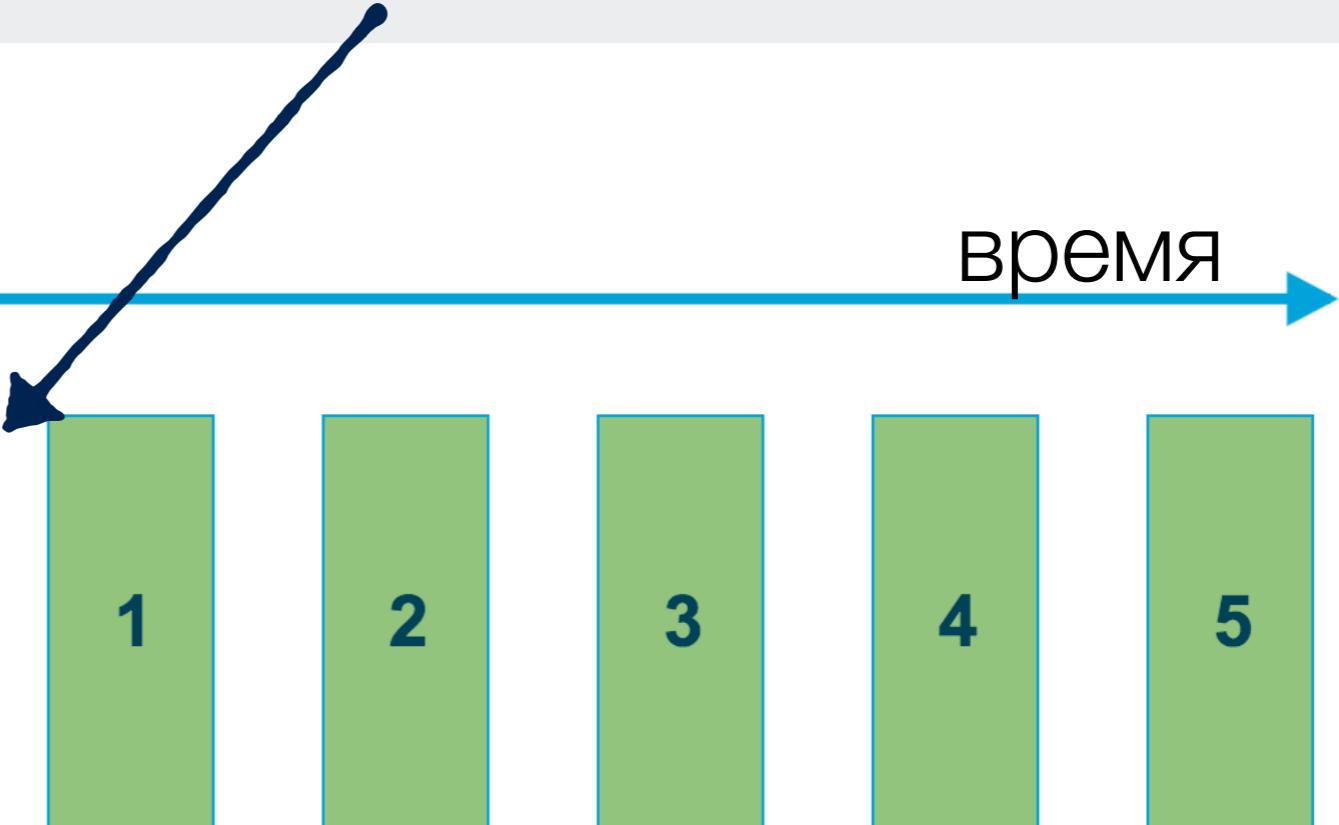
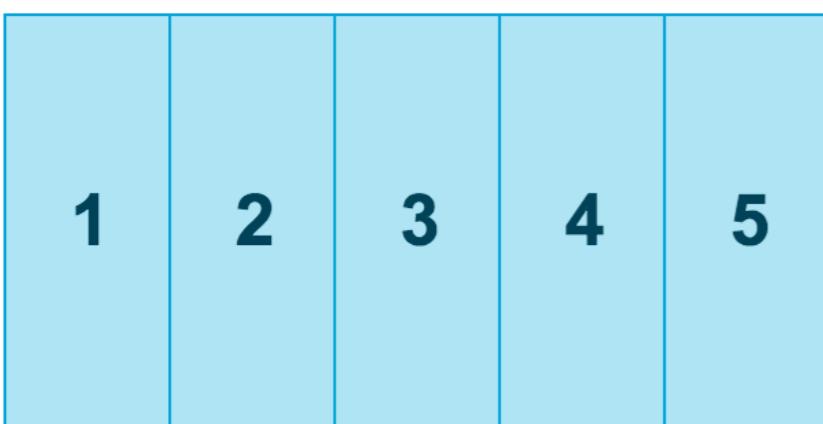
время



```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```

такт event-loop'a

время



```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```

Блокирующее выполнение

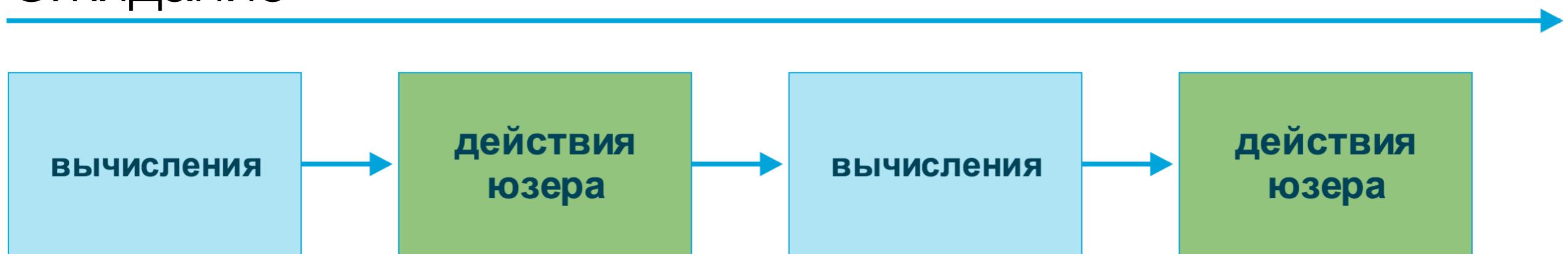
```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(i);  
    }, 0);  
}
```

Асинхронное выполнение

Асинхронность - это боль

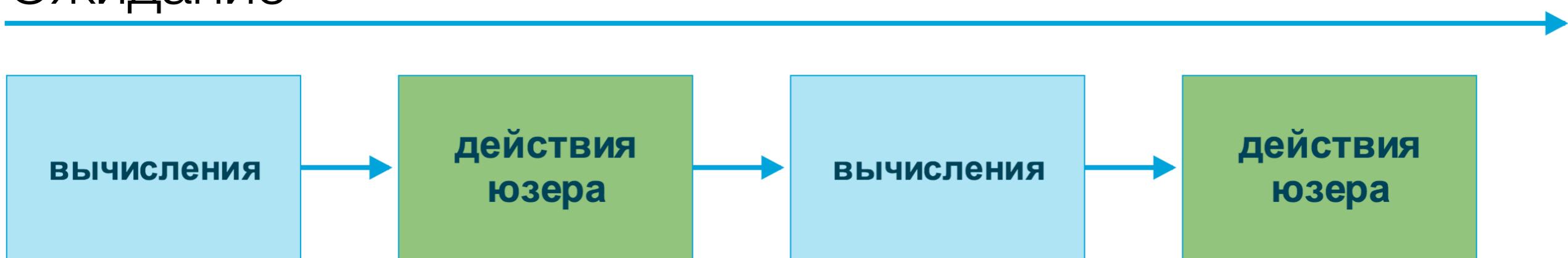
Асинхронность - это боль

Ожидание

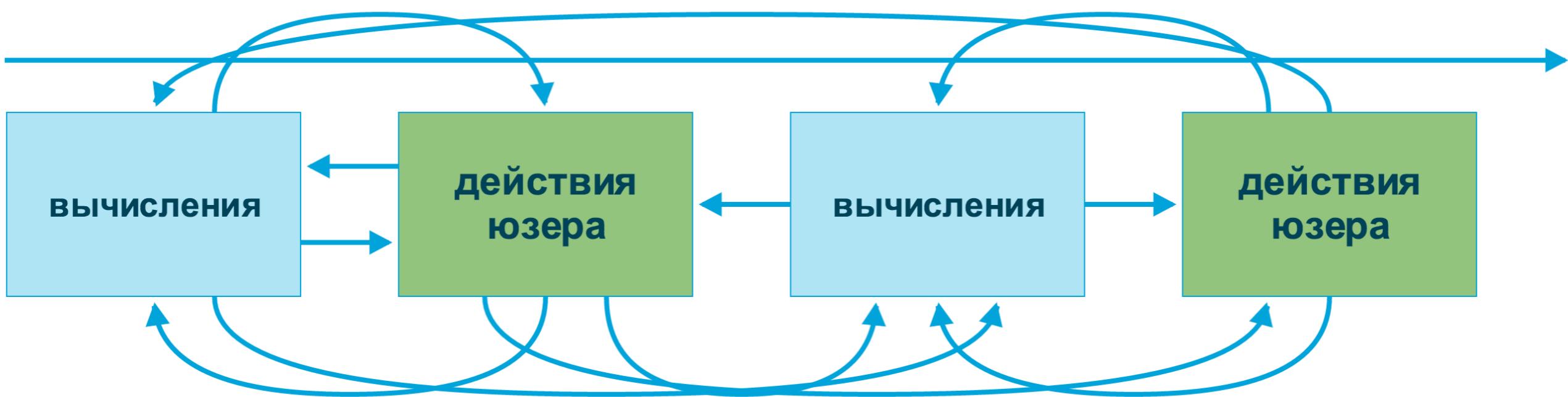


Асинхронность - это боль

Ожидание



Реальность



```
▶ for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(j);  
    }, 0);  
}
```

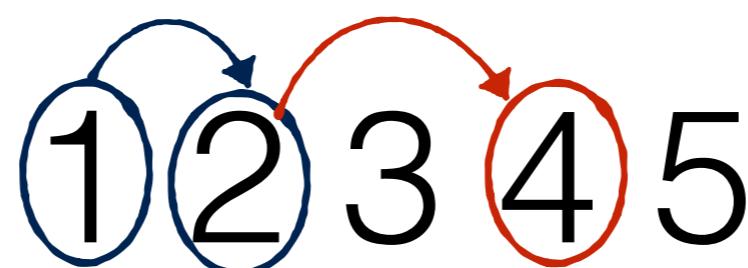
1 2 3 4 5

▶

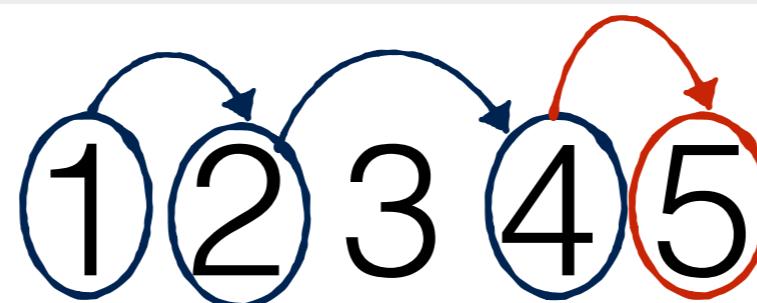
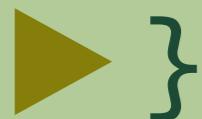
```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(j);  
    }, 0);  
}
```



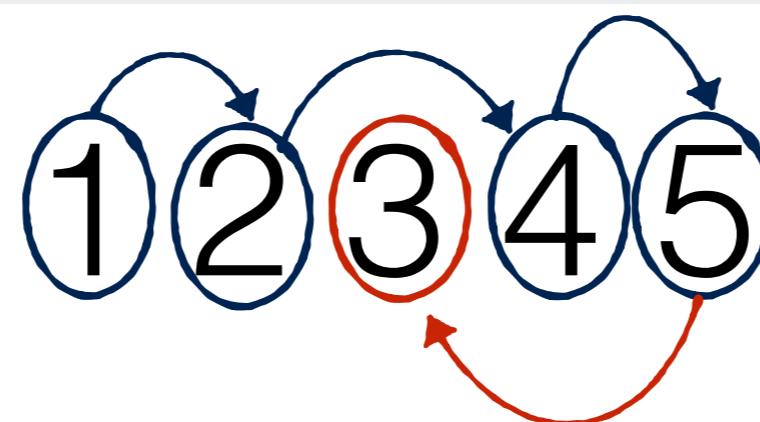
```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(j);  
    }, 0);  
}
```



```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(j);  
    }, 0);  
}
```



```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(j);  
    }, 0);  
}
```



Решение: замыкания

```
for(var i = 0; i < 5; i++){
    setTimeout((function(j){
        return function(){
            return console.log(j);
        };
    })(i), 0);
}
```

Решение: связывание контекста

```
for(var i = 0; i < 5; i++){  
    setTimeout(function(){  
        console.log(this);  
    }.bind(i), 0);  
}
```

Источники асинхронности

- Таймеры
- События
- Асинхронные запросы

СОБЫТИЯ

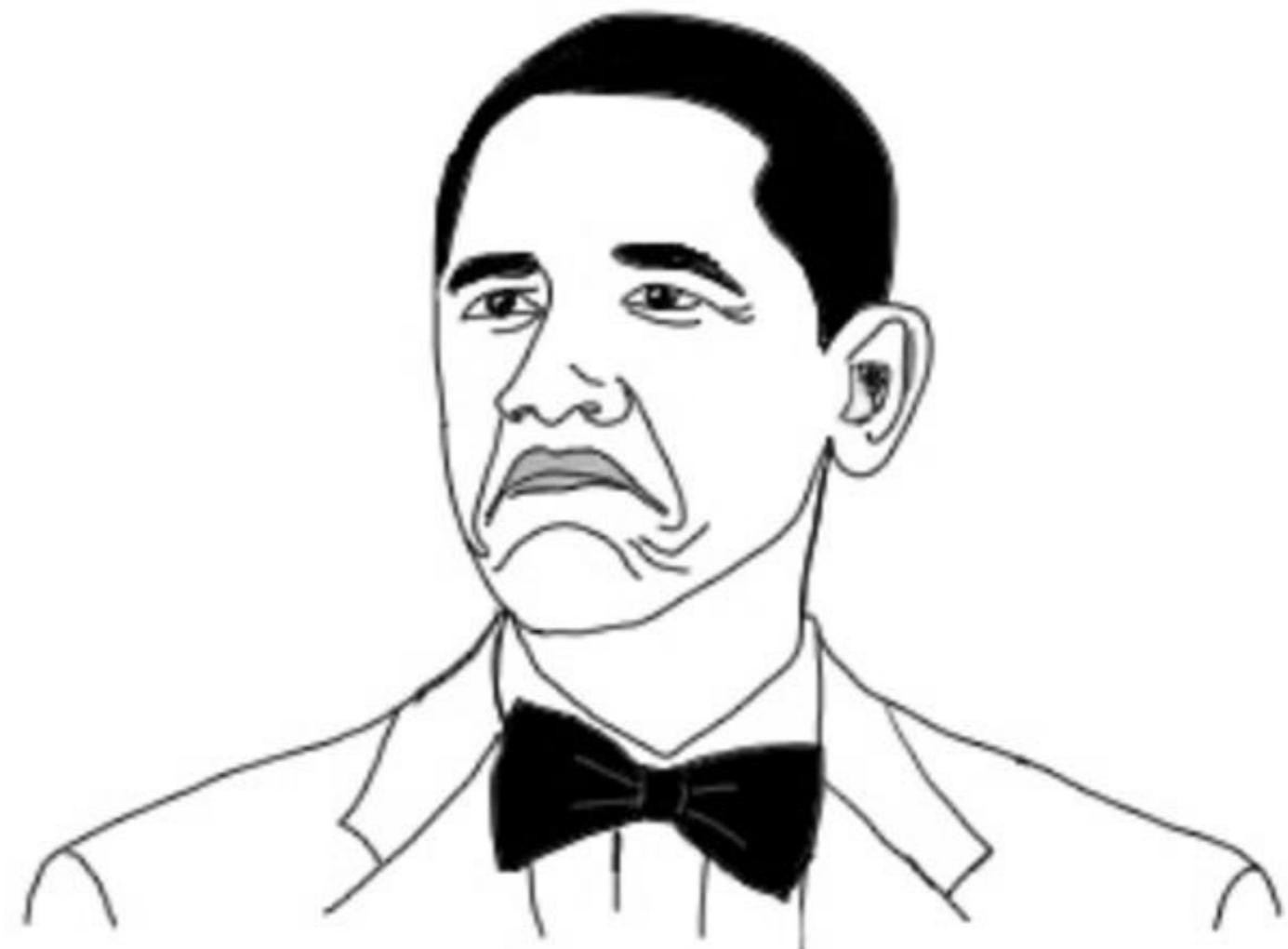


События

- Происходят в разное время (т.е. асинхронно)
- На них можно подписаться
- От них можно отписаться
- Их можно обработать

События

- Происходят в разное время (т.е. асинхронно)
- На них можно подписаться
- От них можно отписаться
- Их можно обработать



NOT BAD

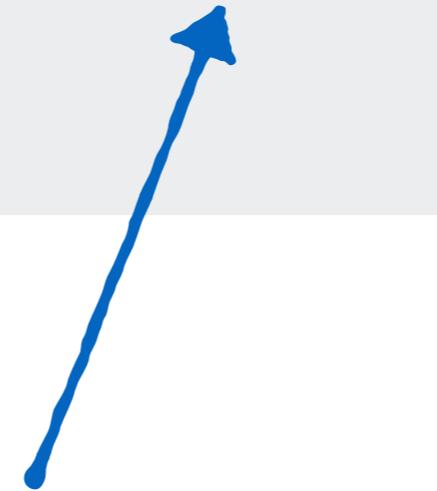
События

```
01: var input = document.getElementById('text');  
02:  
03: input.addEventListener('keyup', function (event) {  
04:   console.log(event);  
05: });
```

Источник события

События

```
01: var input = document.getElementById('text');  
02:  
03: input.addEventListener('keyup', function (event) {  
04:   console.log(event);  
05: });
```

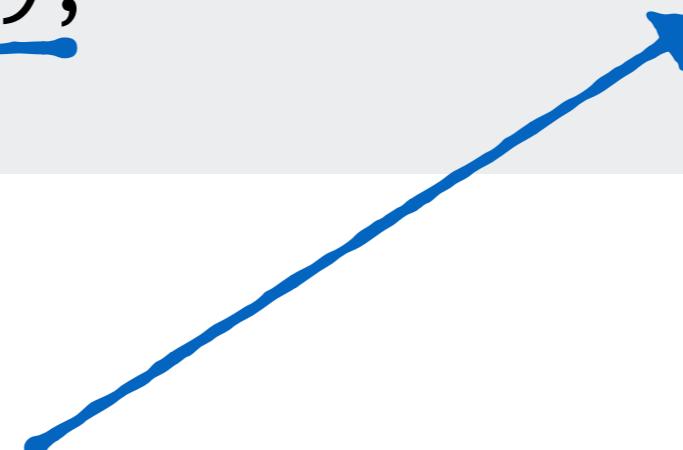


Название события

(когда пользователь отпускает нажатую клавишу)

События

```
01: var input = document.getElementById('text');  
02:  
03: input.addEventListener('keyup', function (event) {  
04:   console.log(event);  
05: });  
});
```



Обработчик события

(выводит в консоль объект события)

Объект события

```
▼ KeyboardEvent {} ⓘ  
  altKey: false  
  bubbles: true  
  cancelBubble: false  
  cancelable: true  
  charCode: 0  
  ctrlKey: false  
  currentTarget: null  
  defaultPrevented: false  
  detail: 0  
  eventPhase: 0  
  keyCode: 68  
  keyIdentifier: "U+0044"  
  keyLocation: 0  
  location: 0  
  metaKey: false  
  ▶ path: Array[5]  
    repeat: false  
    returnValue: true  
    shiftKey: false  
  ▶ srcElement: input#text.classOne.classTwo  
  ▶ target: input#text.classOne.classTwo  
  timeStamp: 1445842296477  
  type: "keyup"  
  ▶ view: Window  
  which: 68  
  ▶ __proto__: KeyboardEvent
```

>

Объект события (ключевые свойства)

```
▼ KeyboardEvent {} ⓘ  
  altKey: false  
  bubbles: true  
  cancelBubble: false ← предотвращение всплытия  
  cancelable: true  
  charCode: 0  
  ctrlKey: false  
  currentTarget: null  
  defaultPrevented: false ← предотвращение поведения по-умолчанию  
  detail: 0  
  eventPhase: 0 ← фаза события  
  keyCode: 68  
  keyIdentifier: "U+0044"  
  keyLocation: 0  
  location: 0  
  metaKey: false  
  ▶ path: Array[5]  
  repeat: false  
  returnValue: true  
  shiftKey: false  
  ▶ srcElement: input#text.classOne.classTwo ← ИСТОЧНИК СОБЫТИЯ  
  ▶ target: input#text.classOne.classTwo ← ИСТОЧНИК СОБЫТИЯ  
  timeStamp: 1445842296477 ← ВРЕМЯ  
  type: "keyup" ← ТИП СОБЫТИЯ  
  ▶ view: Window  
  which: 68  
  ▶ __proto__: KeyboardEvent
```

Код клавиши

>

Фазы событий

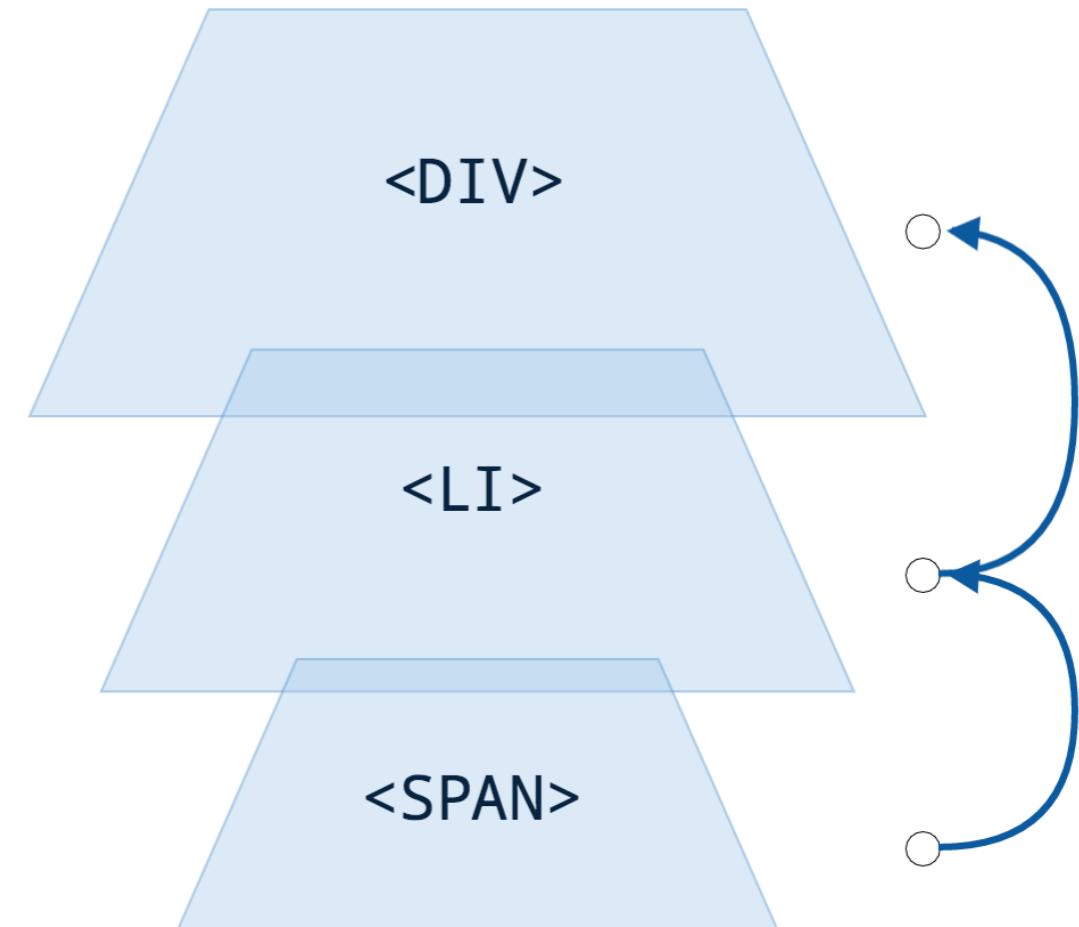
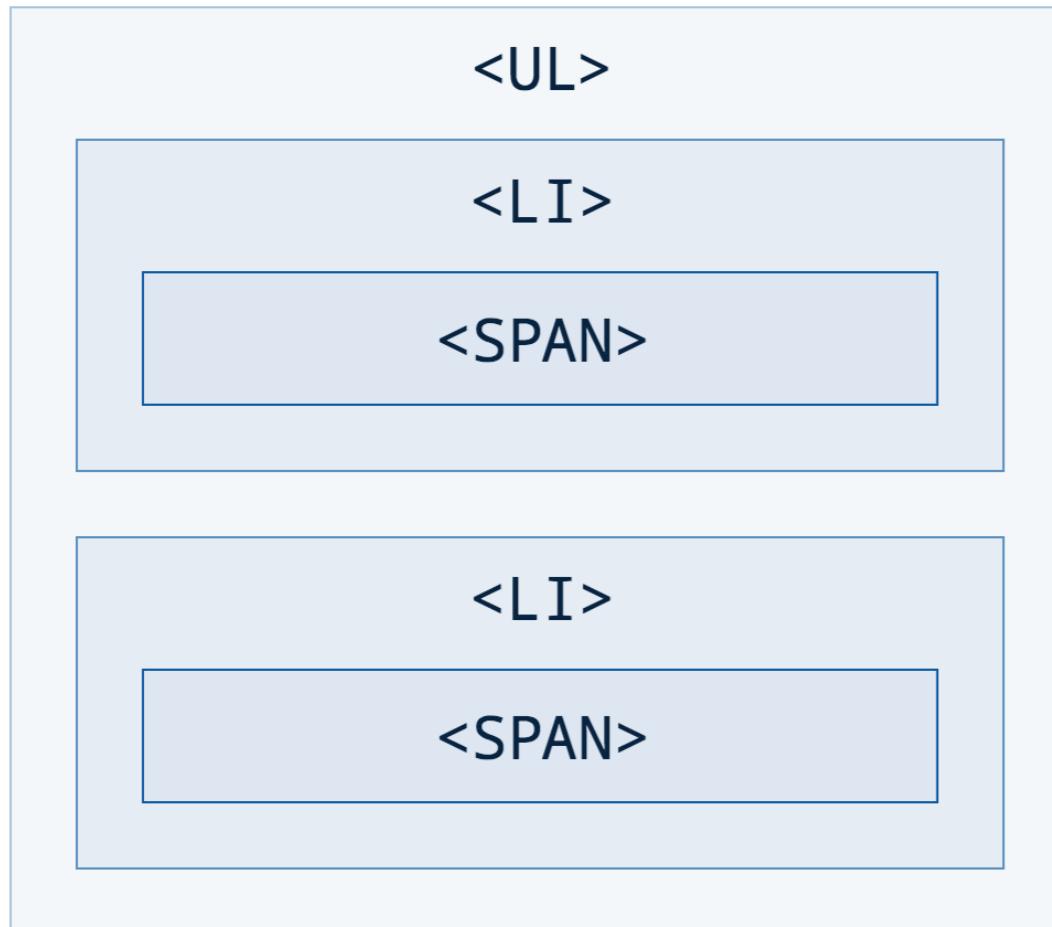
- Всплытие (bubbling stage)

При наступлении события обработчики сначала срабатывают на самом вложенном элементе, затем на его родителе, затем выше и так далее, вверх по цепочке вложенности.

Фазы событий

- Всплытие (bubbling stage)

При наступлении события обработчики сначала срабатывают на самом вложенном элементе, затем на его родителе, затем выше и так далее, вверх по цепочке вложенности.



Фазы событий

- Всплытие (bubbling stage)

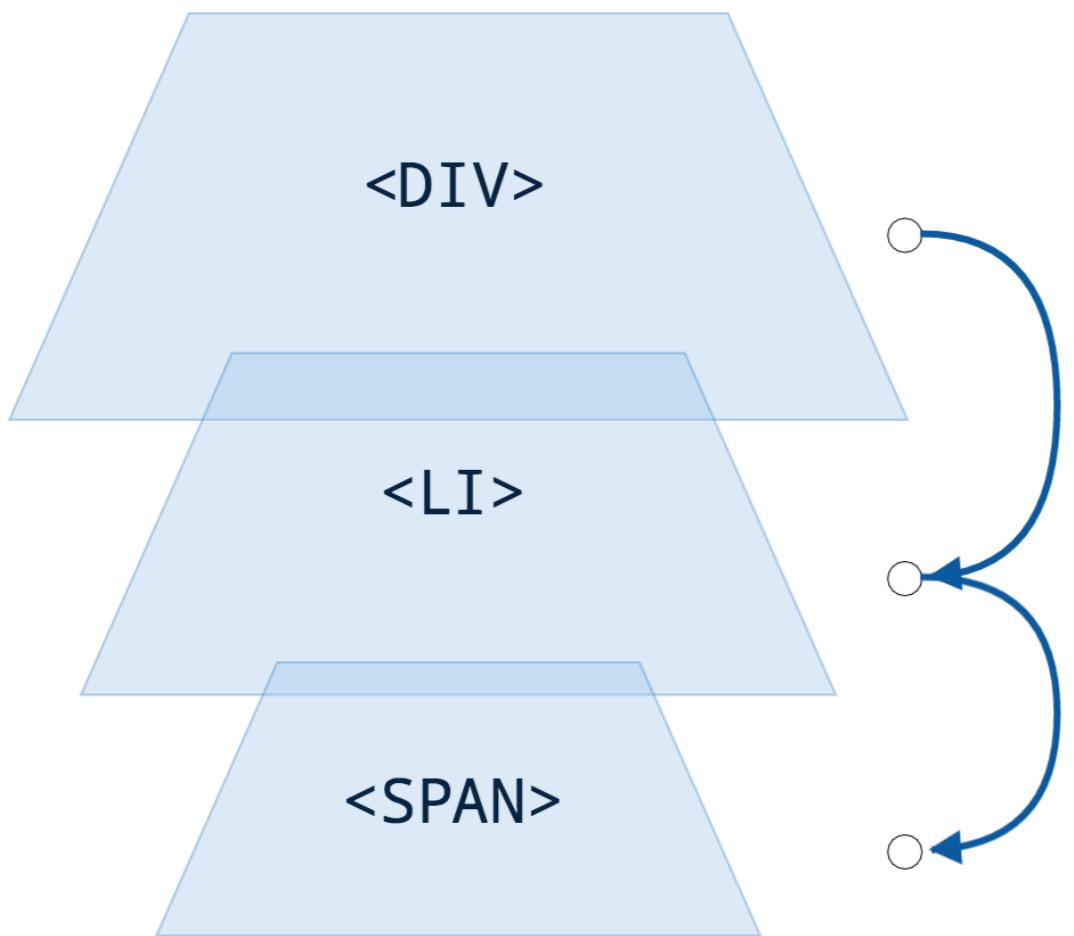
При наступлении события обработчики сначала срабатывают на самом вложенном элементе, затем на его родителе, затем выше и так далее, вверх по цепочке вложенности.

В объекте события свойство target - указывает на тот элемент где события произошло.

Не все события всплывают.

Фазы событий

- Всплытие (bubbling stage)
- Погружение (capturing stage)



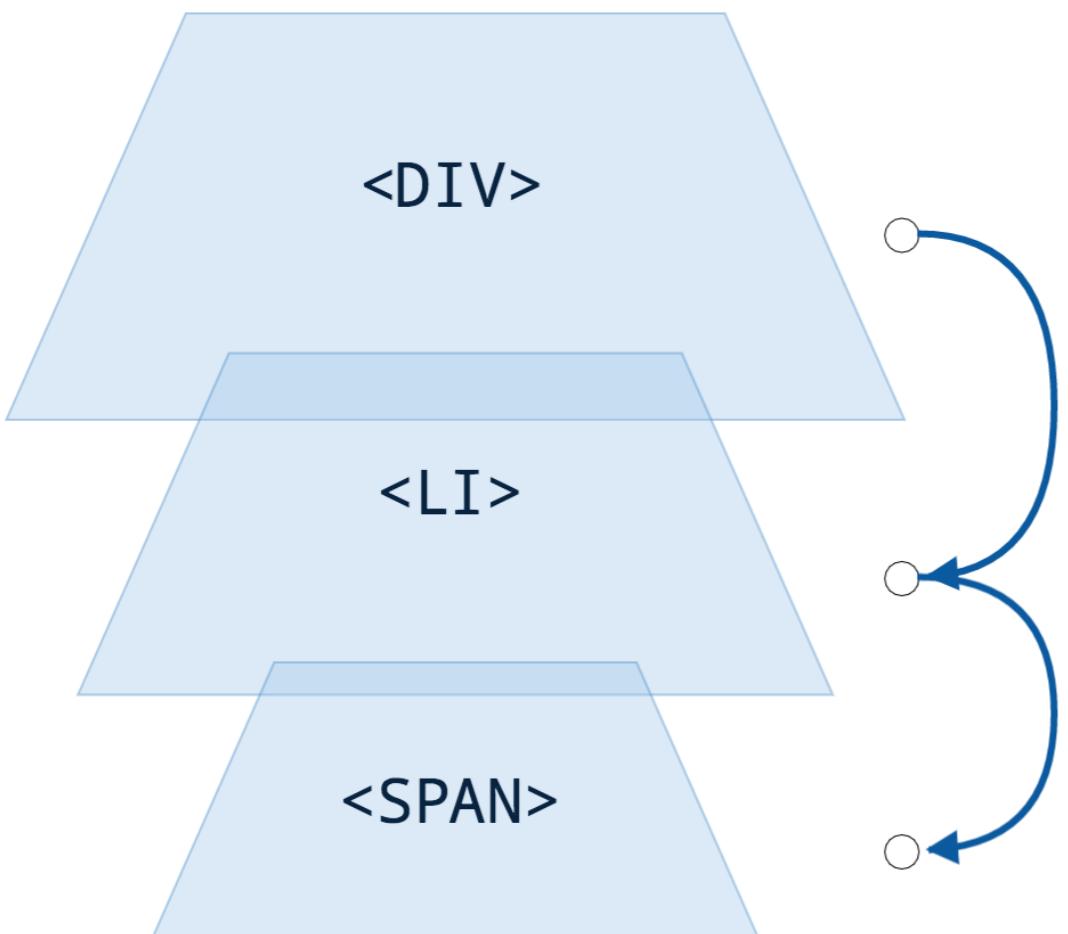
событие распространяется
сверху вниз

Фазы событий

- Всплытие (bubbling stage)
- Погружение (capturing stage)

Чтобы поймать событие на стадии перехвата, нужно использовать третий аргумент `addEventListener`:

- Если аргумент `true`, то событие будет перехвачено по дороге вниз.
- Если аргумент `false`, то событие будет поймано при всплытии.



событие распространяется
сверху вниз

Фазы событий

- Всплытие (bubbling stage)
- Погружение (capturing stage)

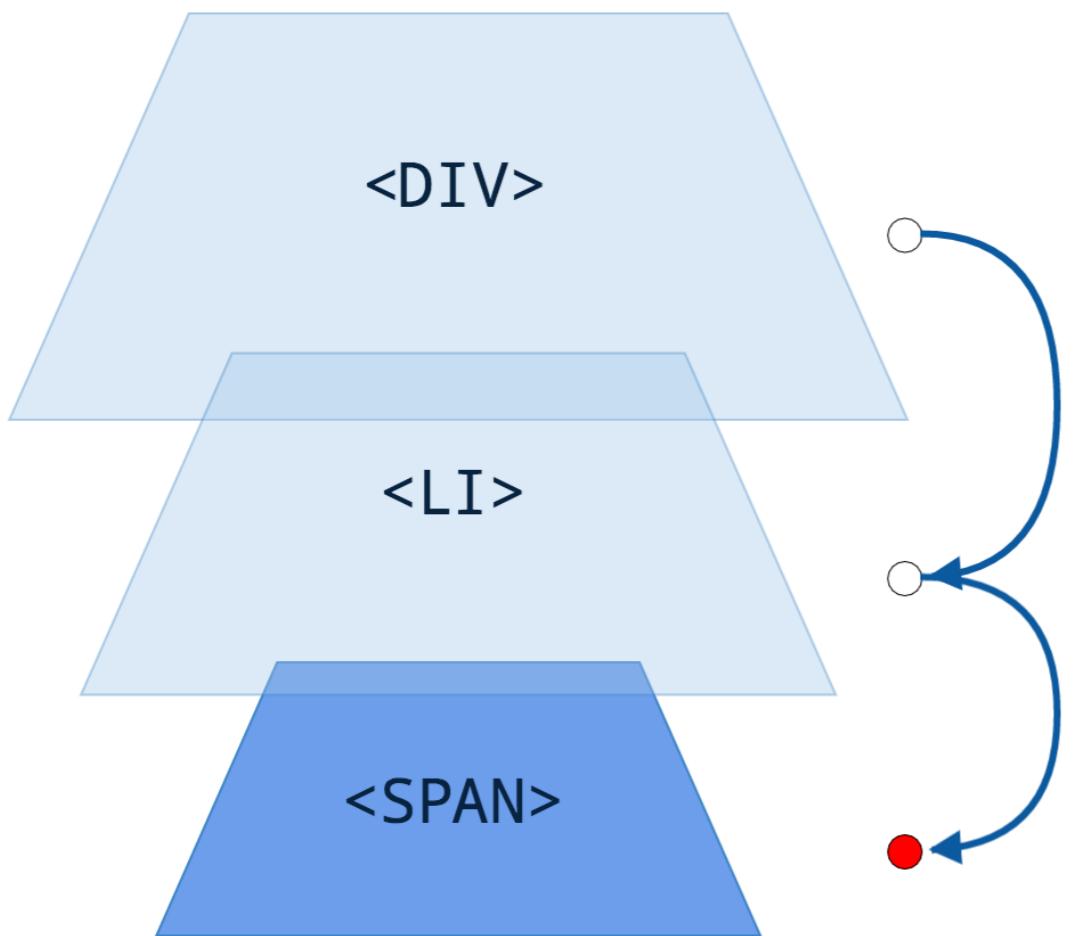
Чтобы поймать событие на стадии перехвата, нужно использовать третий аргумент `addEventListener`:

- Если аргумент `true`, то событие будет перехвачено по дороге вниз.
- Если аргумент `false`, то событие будет поймано при всплытии.

Есть события, которые не всплывают, но которые можно перехватить

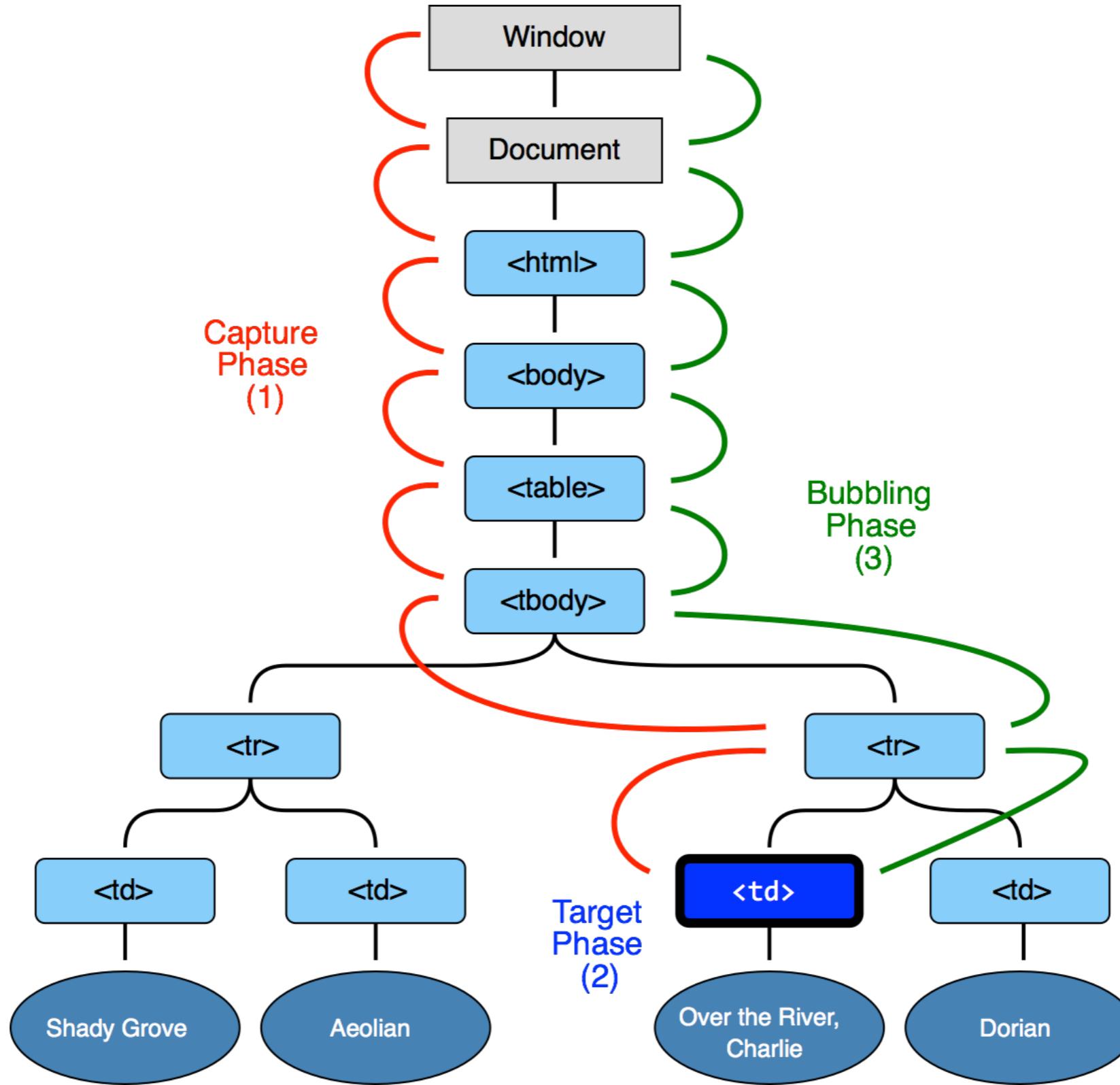
Фазы событий

- Всплытие (bubbling stage)
- Погружение (capturing stage)
- Достижение цели (target stage)



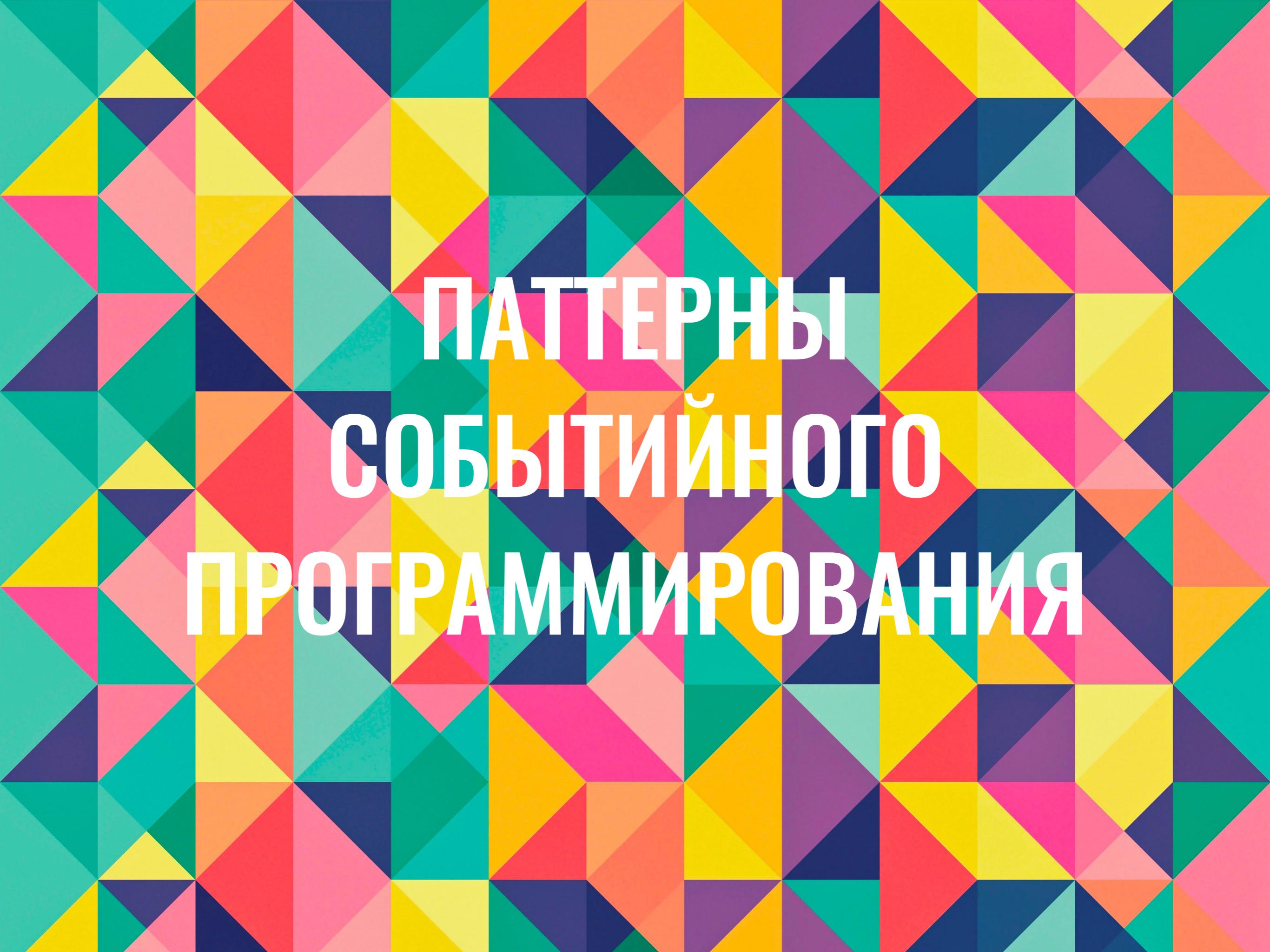
событие достигло цели:
`event.target`

Фазы событий (DOM3 events)



Что является событием

- события ввода - движения курсора мыши, клики, прокрутки, ввод с клавиатуры и других устройств (тачскрин, микрофон и так далее)
- события окна (изменение размера, хэша, закрытие)
- события браузера (загрузка документа, отрисовка элементов)
- события сети (завершение запроса, события веб-сокетов)
- события специальных элементов (video, audio, canvas)
- пользовательские события



ПАТТЕРНЫ СОБЫТИЙНОГО ПРОГРАММИРОВАНИЯ

Pub/Sub

Publisher Subscriber



Pub/Sub

Publisher Subscriber



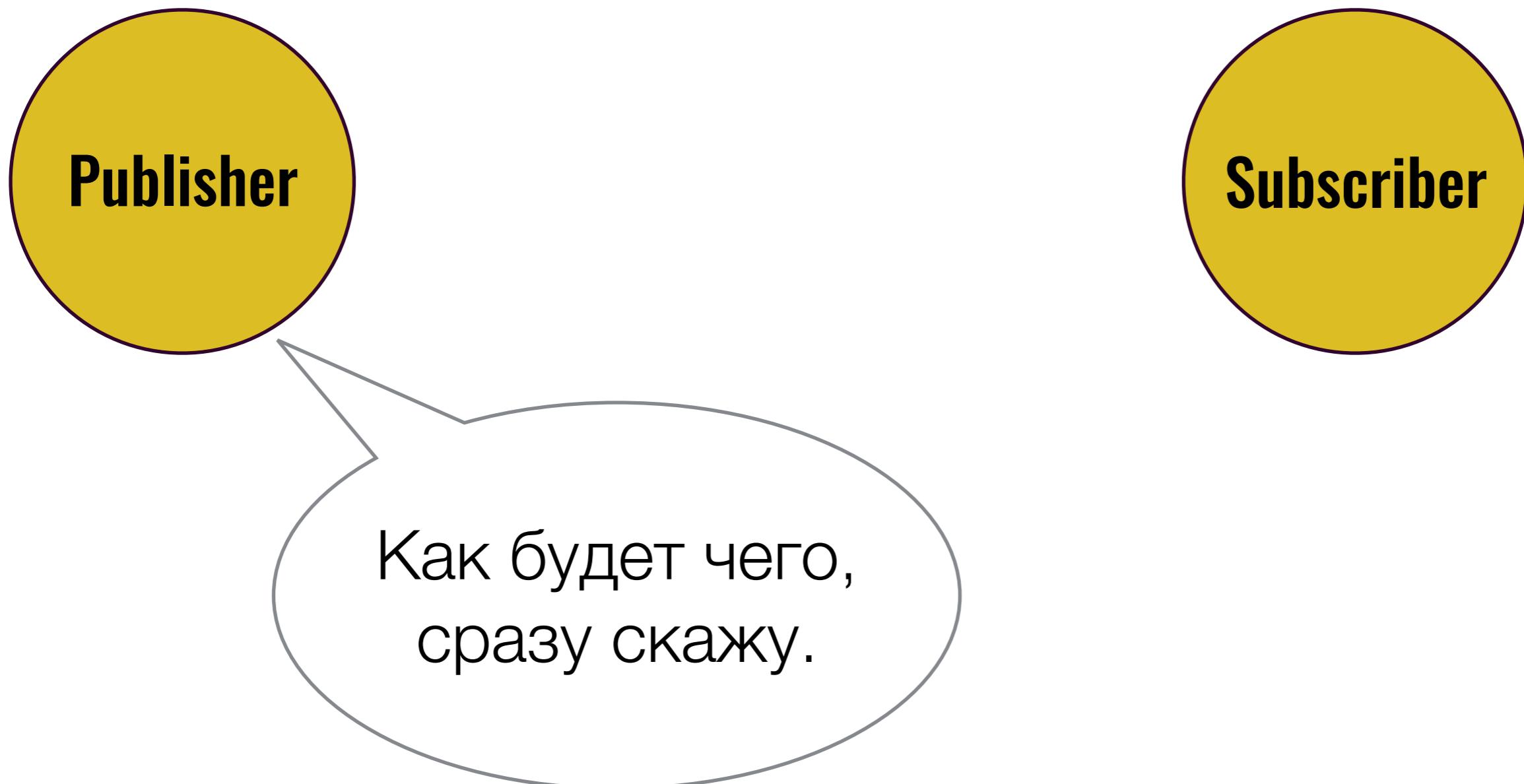
Pub/Sub

Publisher Subscriber



Pub/Sub

Publisher Subscriber



Pub/Sub

Publisher Subscriber



спустя какое-то время



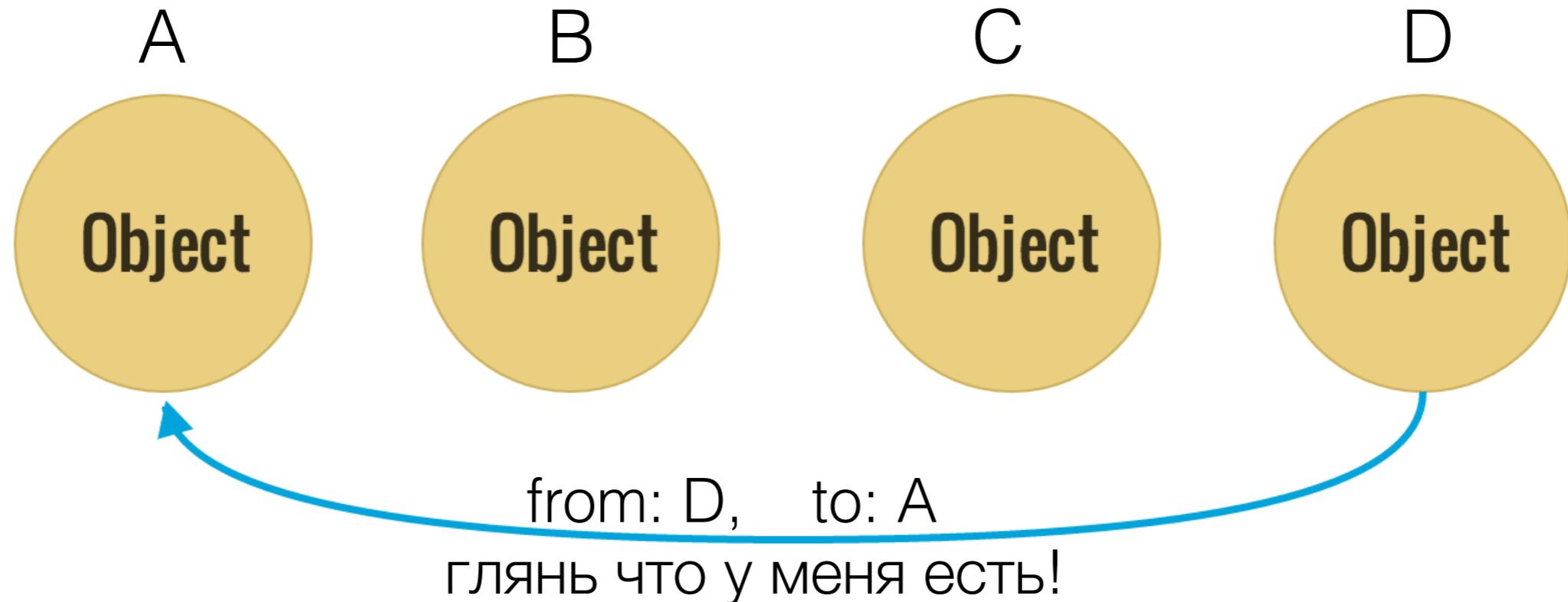
Pub/Sub

Publisher Subscriber



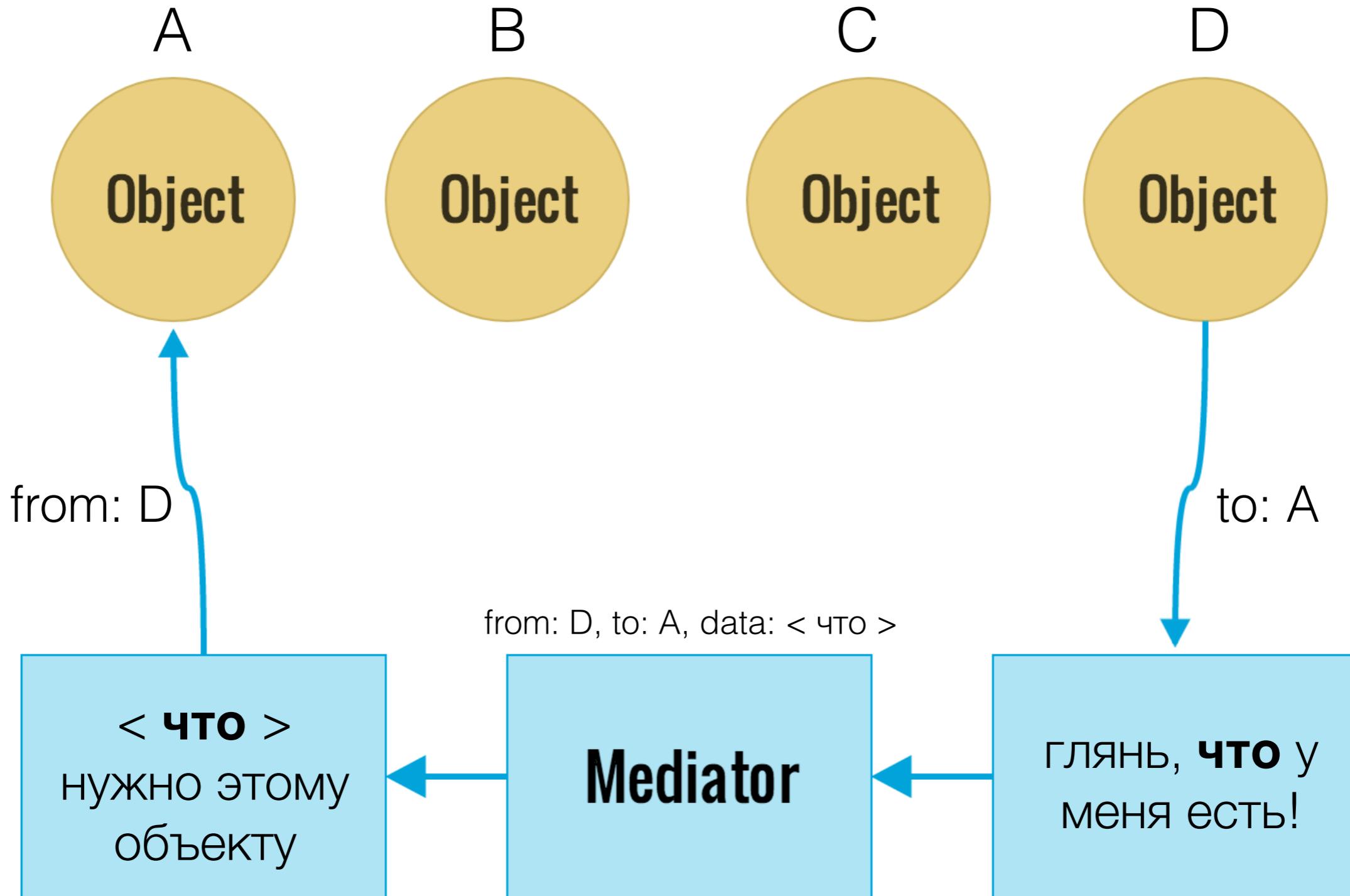
Pub/Sub

Publisher Subscriber



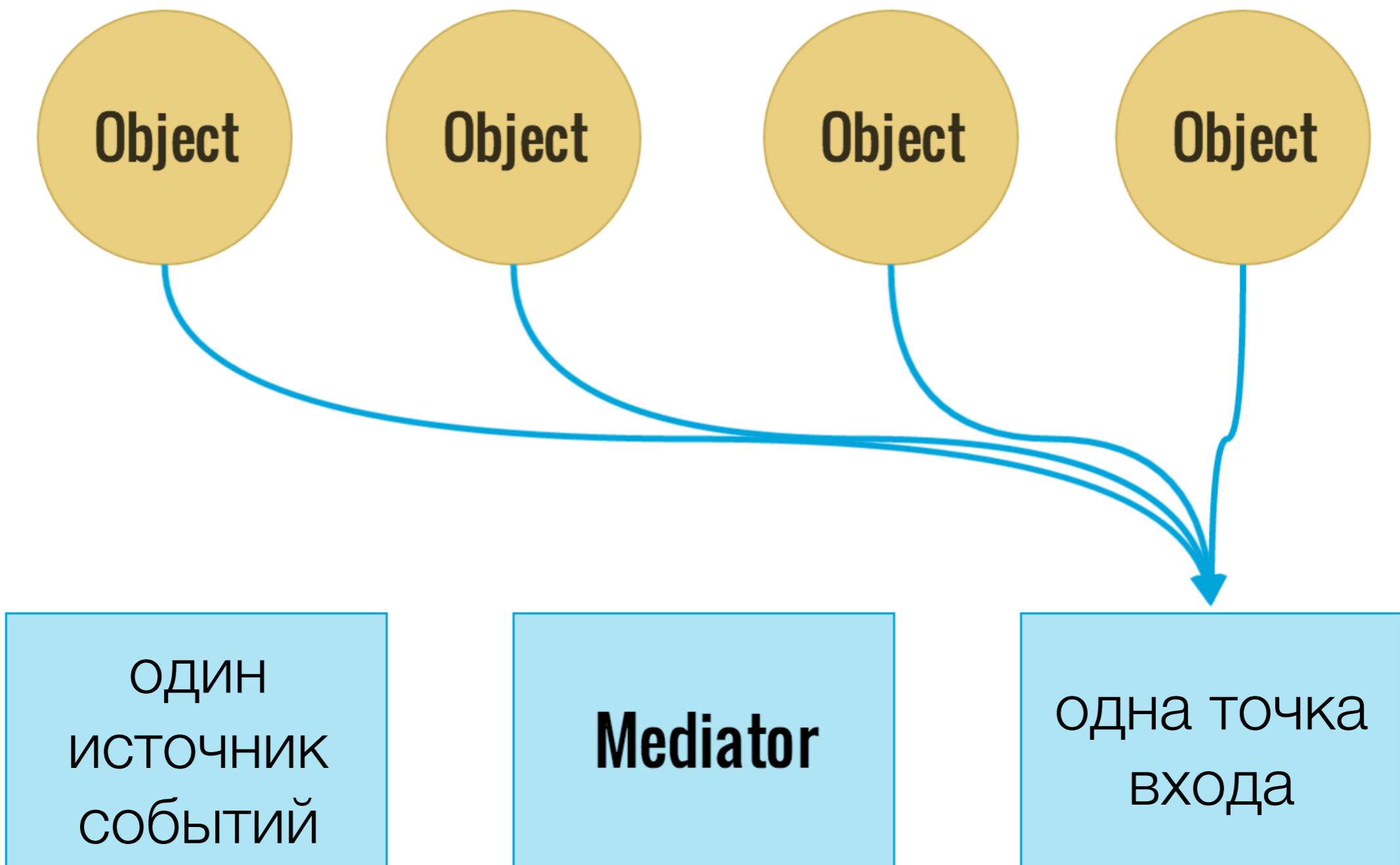
Pub/Sub

Publisher Subscriber



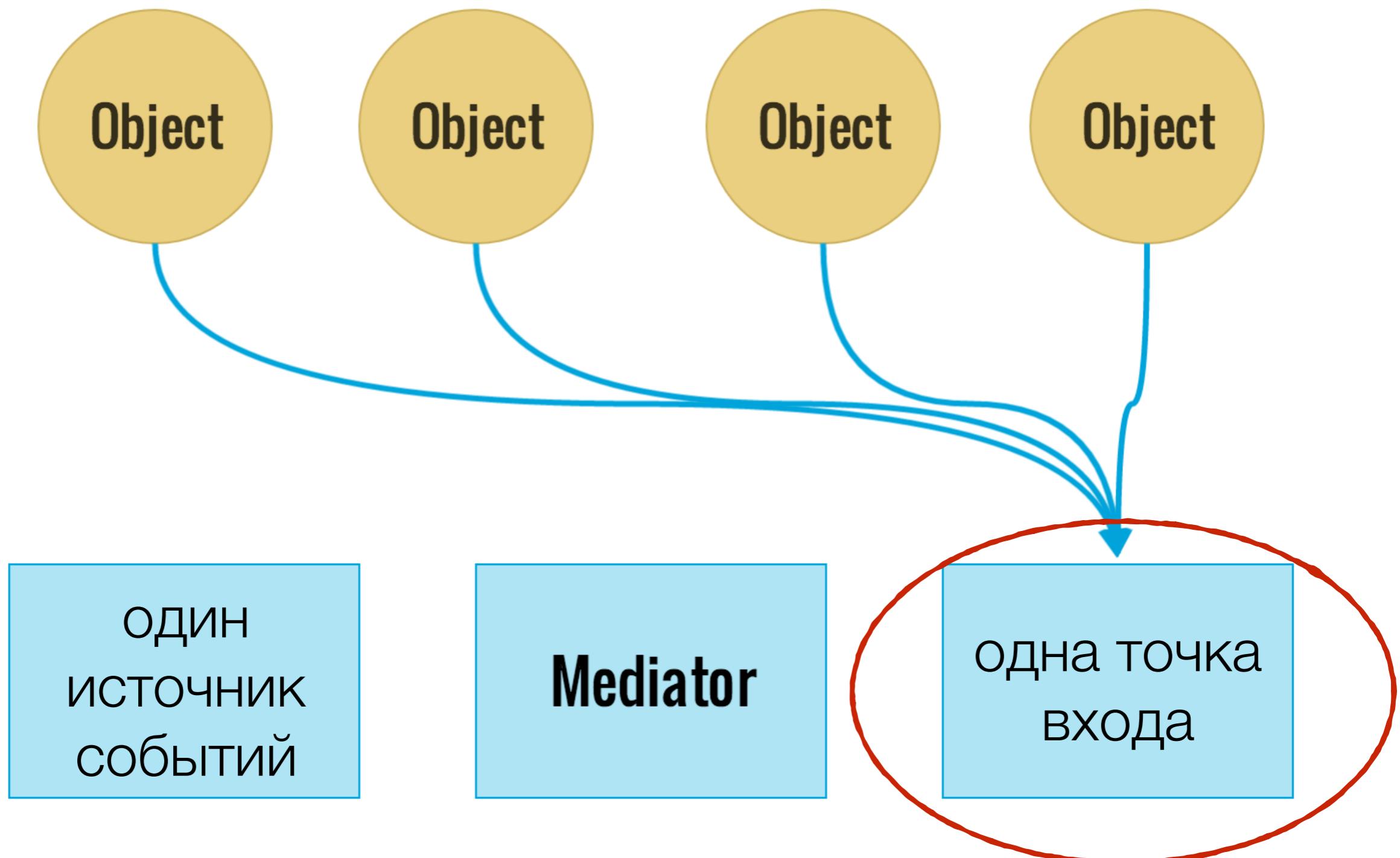
Pub/Sub

Publisher Subscriber



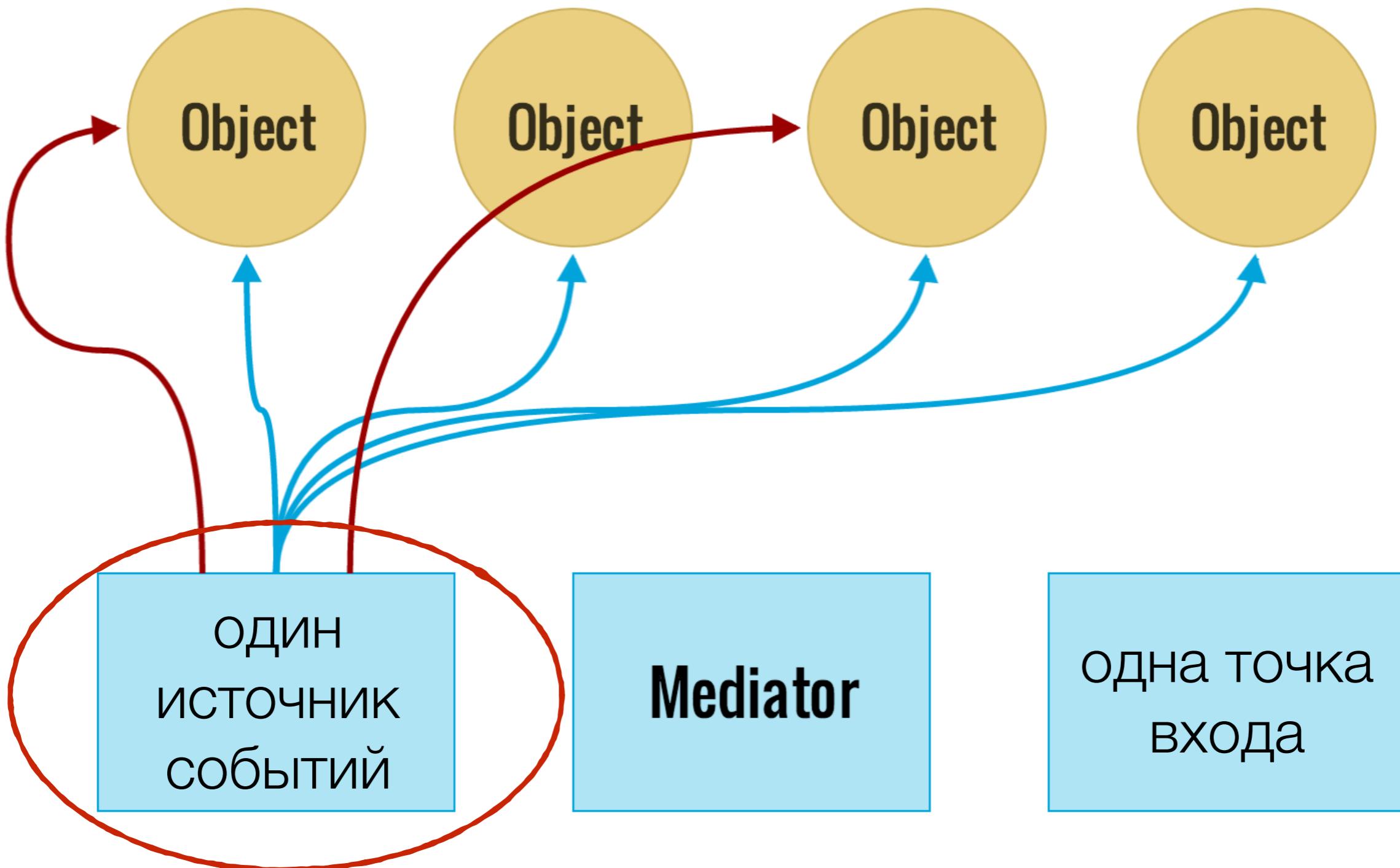
Pub/Sub

Publisher Subscriber



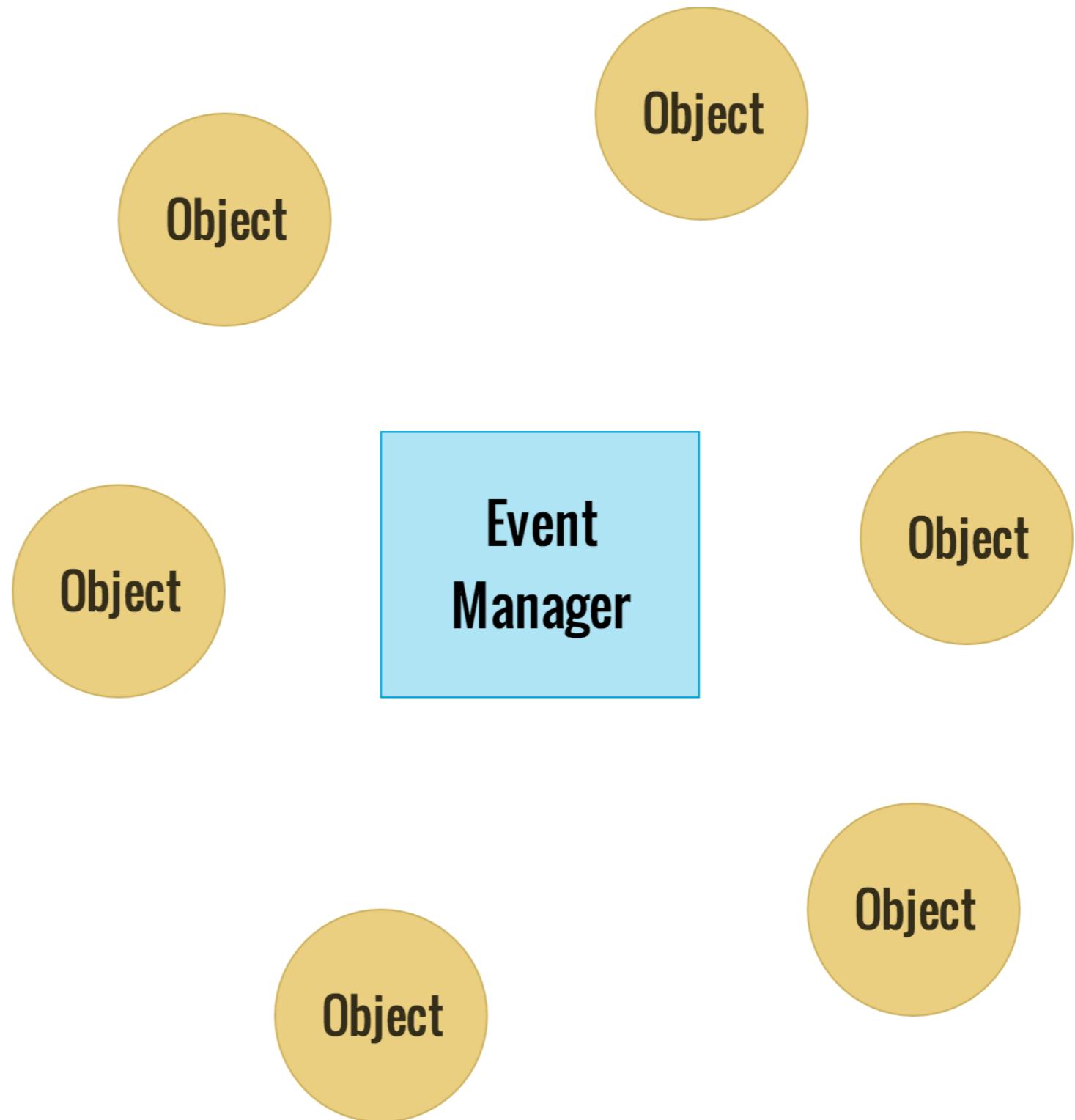
Pub/Sub

Publisher Subscriber



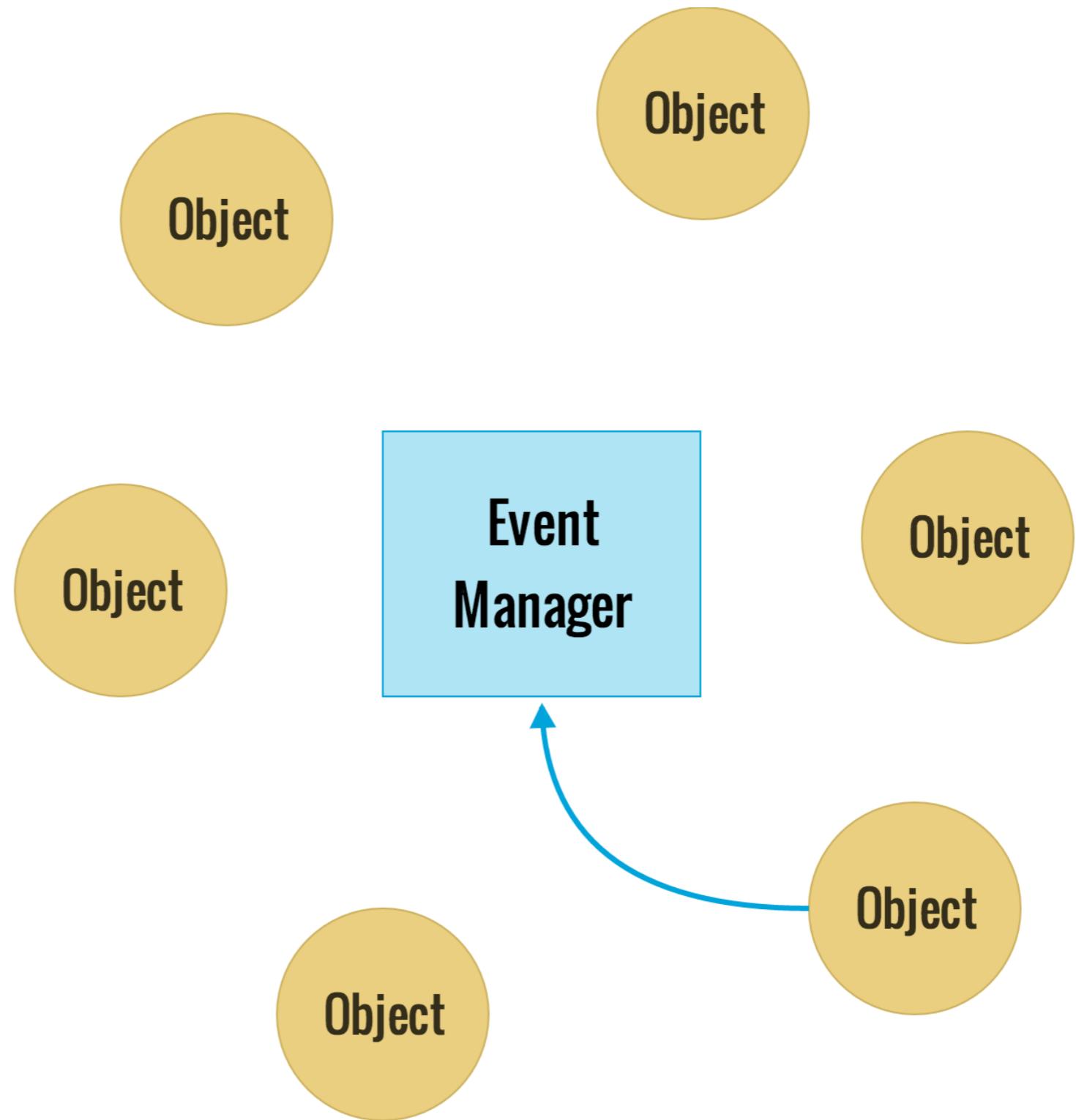
Pub/Sub

Publisher Subscriber



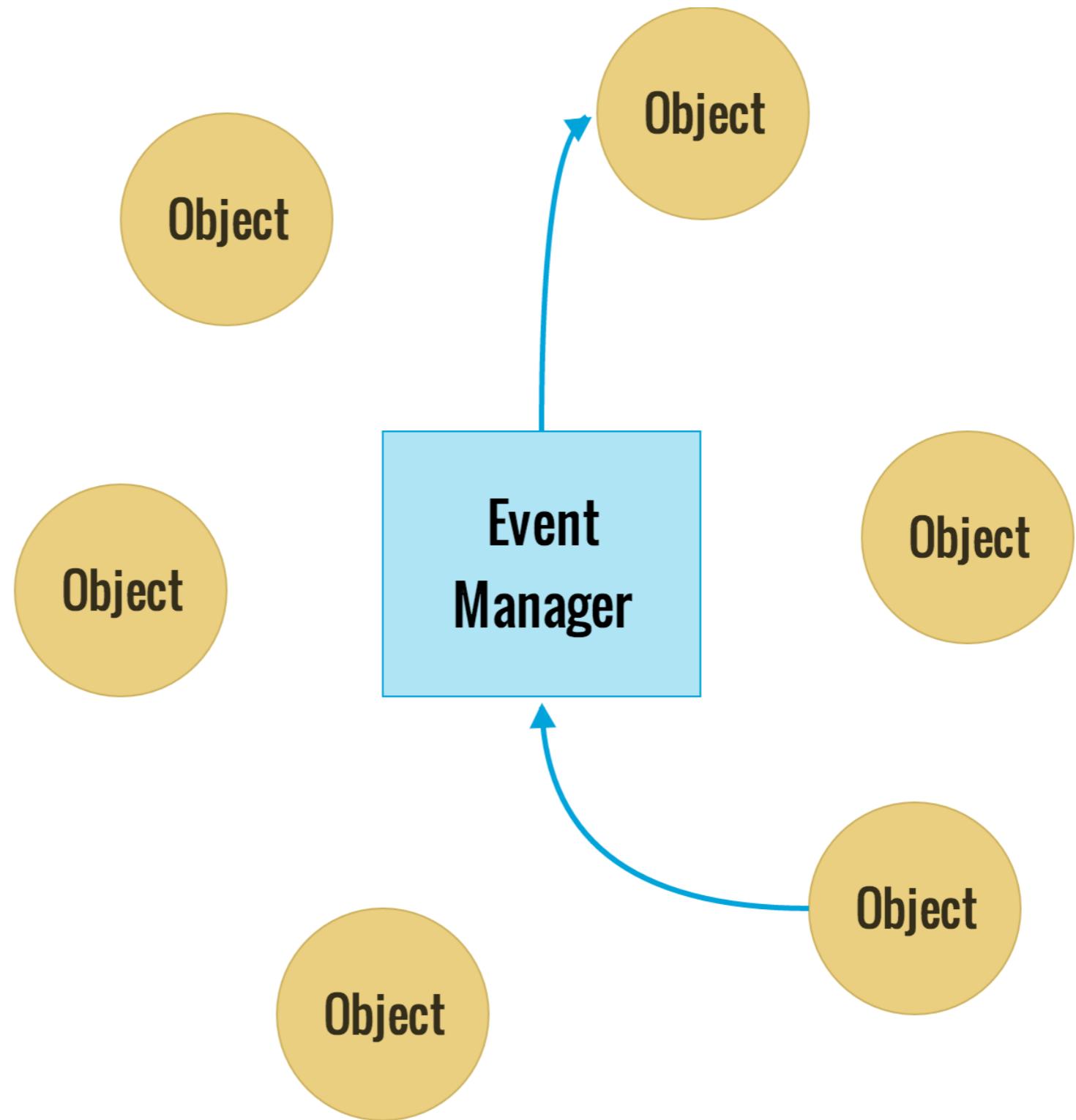
Pub/Sub

Publisher Subscriber



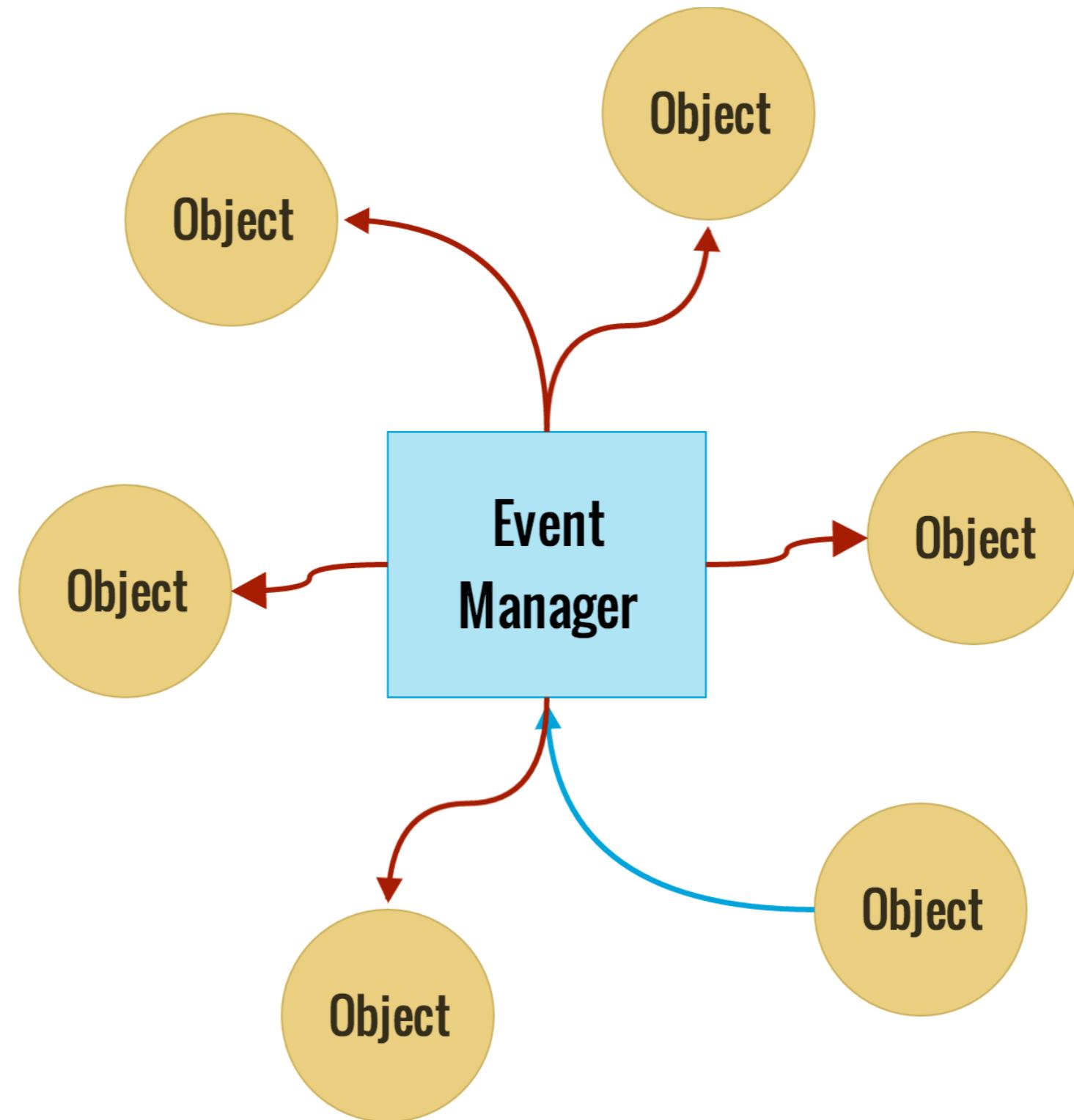
Pub/Sub

Publisher Subscriber



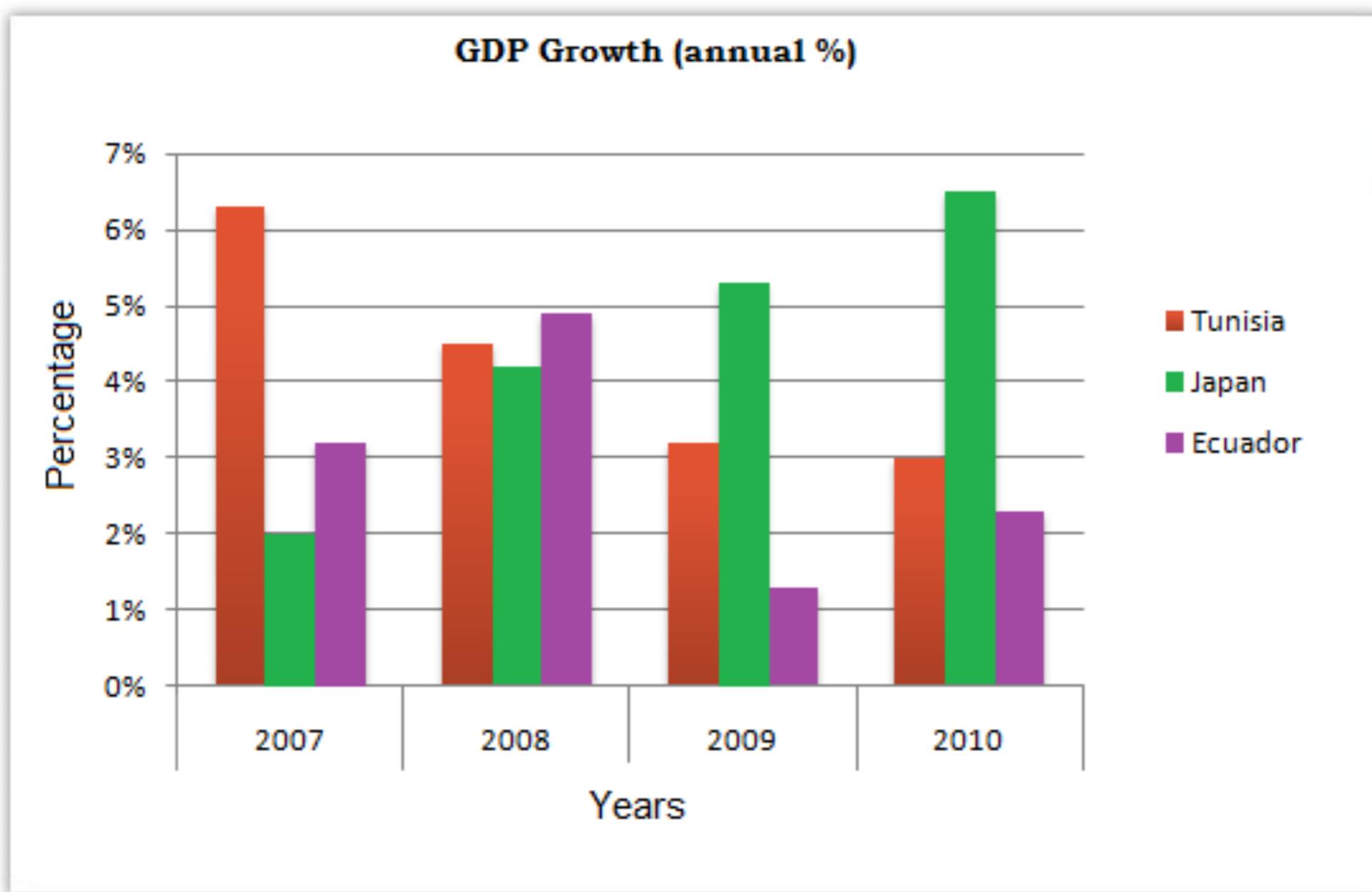
Pub/Sub

Publisher Subscriber



D3





среднестатистическая гистограмма

всего-лишь

HTML

CSS

JS

SVG

DOM

Подключаем

```
01: <!DOCTYPE html>
02:<html>
03:   <head>
04:     <title>D3</title>
05:   </head>
06:   <body>
07:     <script type="text/javascript" src="d3.js">
08:   </body>
09:</html>
```

И наслаждаемся

```
01: var body = d3.select('body');  
01: body.append('p').text('Hello World!')
```

И наслаждаемся

```
01: var body = d3.select('body');  
01: body.append('p').text('Hello World!')
```

Hello World!

а теперь, сделаем что-то полезное

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:     .data(BIG_DATA)
04:       .enter()
05:         .append('div')
06:           .attr('class', 'bar')
07:             .style('width', function(d) {
08:               return d + 'px';
09:             })
10:             .text(function(d) {
11:               return d;
12:             })
13:             .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:     .data(BIG_DATA)
04:       .enter()
05:         .append('div')
06:           .attr('class', 'bar')
07:             .style('width', function(d) {
08:               return d + 'px';
09:             })
10:             .text(function(d) {
11:               return d;
12:             })
13:             .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:     .append('div')
06:     .attr('class', 'bar')
07:     .style('width', function(d) {
08:       return d + 'px';
09:     })
10:     .text(function(d) {
11:       return d;
12:     })
13:   .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

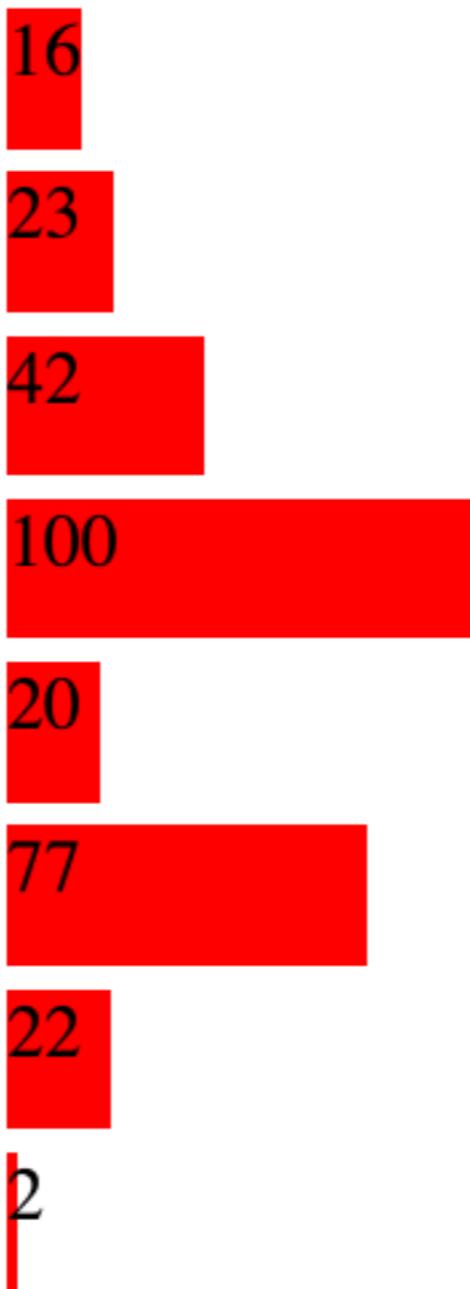
```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

```
01: d3.select('.container')
02:   .selectAll('div')
03:   .data(BIG_DATA)
04:   .enter()
05:   .append('div')
06:   .attr('class', 'bar')
07:   .style('width', function(d) {
08:     return d + 'px';
09:   })
10:   .text(function(d) {
11:     return d;
12:   })
13:   .exit();
```

```
var BIG_DATA = [16, 23, 42, 100, 20, 77, 22, 2];
```



на этом все
спасибо!