Разбираем отладчик JavaScript

(на основе Chrome DevTools)

Алексей Козятинский
ak239spb@gmail.com
twitter.com/ak_239

План

- console.log
- Асинхронные стеки
  - IO’18: Async Step Into
- CPU profiler
- Heap profiler
- IO’18: Eager evaluation
- IO’18: Step Into + Web Worker

IO’18: https://youtu.be/mfuE53x4b3k

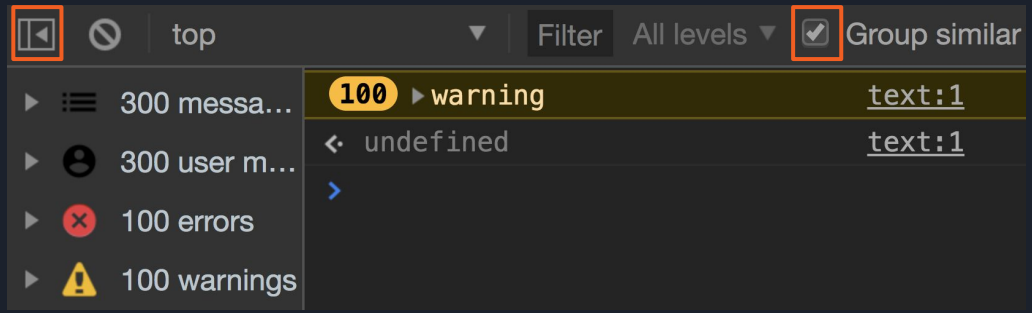
console.log

```
Object.defineProperty(window, 'a', {
  get() {
    console.log('DevTools are here!');
  }
});
console.log(window);
window.a = 42;
```

```
▶ Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}
```

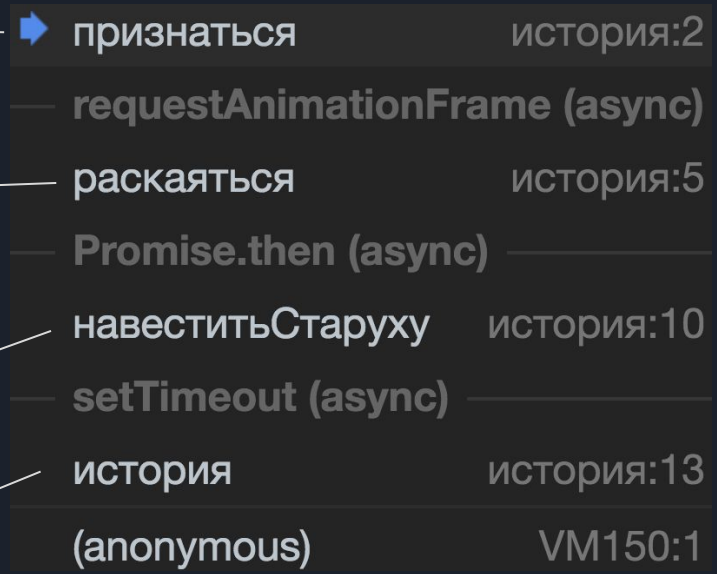
console.log - ваш друг

- console.log(window) - медленный и бессмысленный
  - console.log({a:1, b: 2}) лучше
- console.log игнорирует ваши геттеры
- console.(log|warn|trace|...) - у нас много разных методов и удобный сайдбар для их фильтрации



Асинхронный стек

```
function признаться() {
 debugger;
}
function раскаяться() {
  requestAnimationFrame(признаться);
}
function навеститьСтаруху() {
  применитьТопор()
    .then(применитьТопорЕщеРаз)
    .then(раскаяться);
}
function история() {
  setTimeout(навеститьСтаруху, SOME_TIME);
}
```



Асинхронный вызов

Асинхронный вызов - это любой вызов функции с использованием одного из встроенных примитивов (setTimeout, promise.then, ...), который вызывает вашу функцию в будущем.

```
setTimeout(yourFunction, 1000);

promise.then(yourFunction);

script.addEventListener('load', yourFunction);

...
```

Синхронный стек - это список синхронных вызовов.

Асинхронный стек - это список синхронных и асинхронных вызовов.

Асинхронный вызов + StepInto =❤️

```
1 function foo() {
2   bar();
3 }
4 function bar() {
5   return 42;
6 }
```

```
1 function foo() {
2   bar();
3 }
4 function bar() {
5   return 42;
6 }
```

```
1 function foo() {
2   setTimeout(bar, 1000);
3 }
4 function bar() {
5   return 42;
6 }
```

```
1 function foo() {
2   setTimeout(bar, 1000);
3 }
4 function bar() {
5   return 42;
6 }
```

Асинхронный стек: взгляд изнутри

```
function foo() {
  setTimeout(bar, 1000);
}

function bar() {
  return 42;
}
```

собираем и сохраняем текущий стек

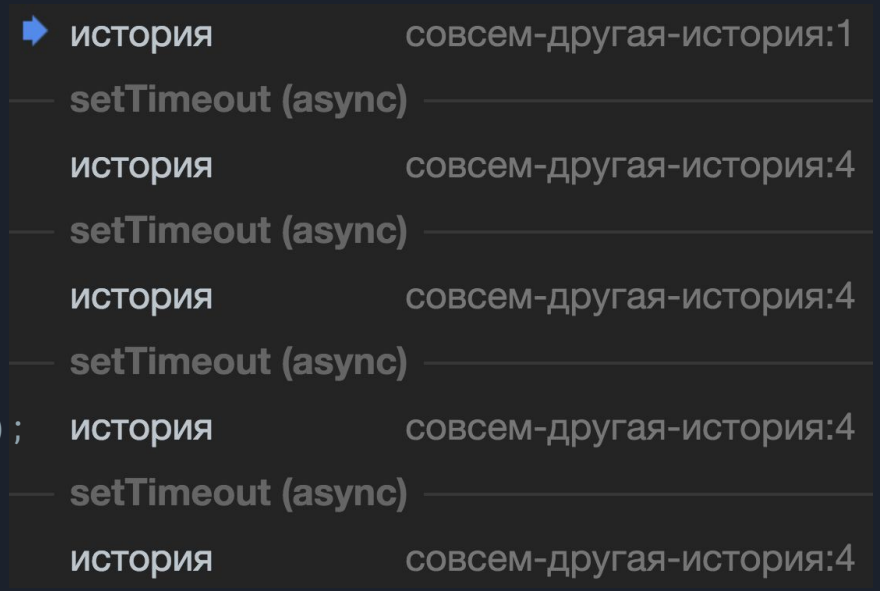
```
foo        async-step-into:2
```

используем собранный стек

```
bar        async-step-into:5
setTimeout (async)
foo        async-step-into:2
```

Асинхронный стек: сложность

```
function история() {
  if (решился())
    навеститьСтаруху();
  setTimeout(история, ОДИН_ДЕНЬ);
}
setTimeout(выигратьВЛотерею, ПЯТНАДЦАТЬ_ЛЕТ);
история();
```



...

Асинхронный стек - ваш друг

- помогает разобраться, откуда текущий синхронный код вызван
- для экономии памяти - мы отбрасываем старые асинхронные стеки
- step-into одинаково хорош с синхронными и асинхронными вызовами

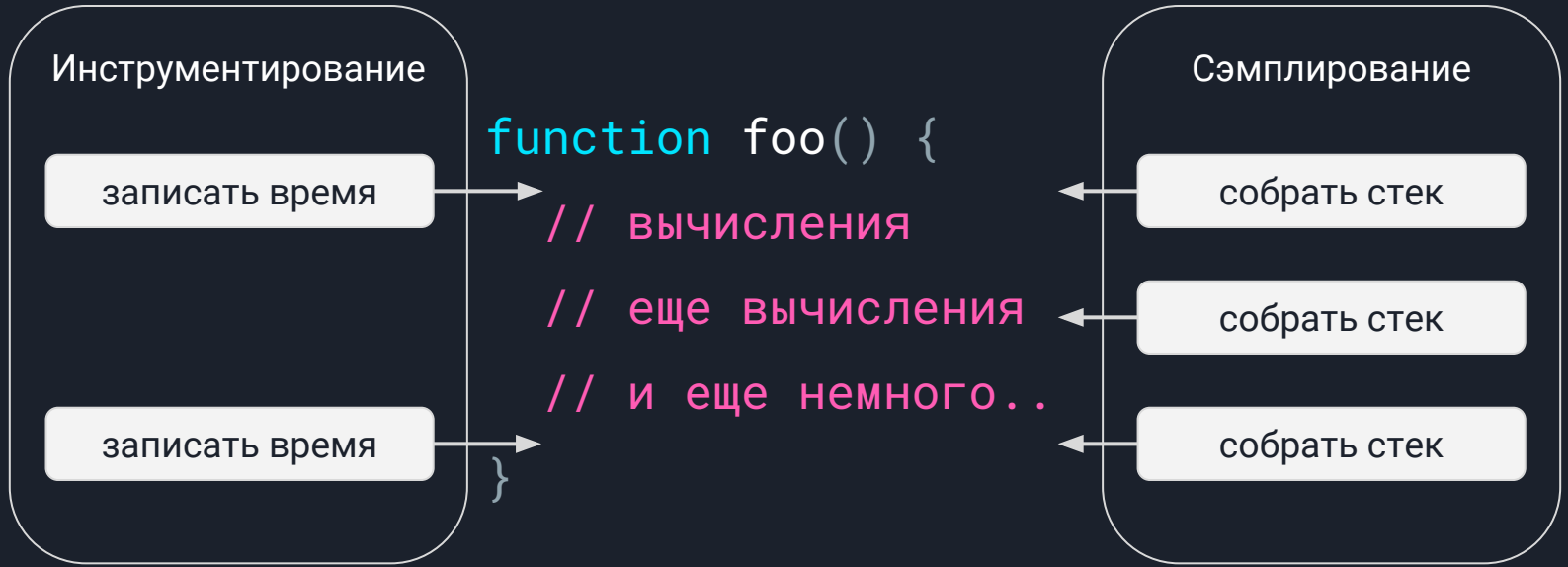
Промежуточные итоги

- console.log и стеки помогают разобраться, что именно происходит в вашей программе

Далее...

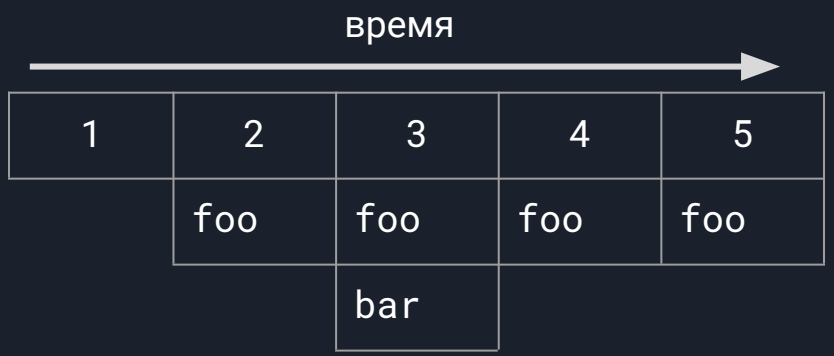
- CPU profiler - почему программа работает медленно
- Heap profiler - почему программа использует так много памяти

Профилирование



Сэмплирование

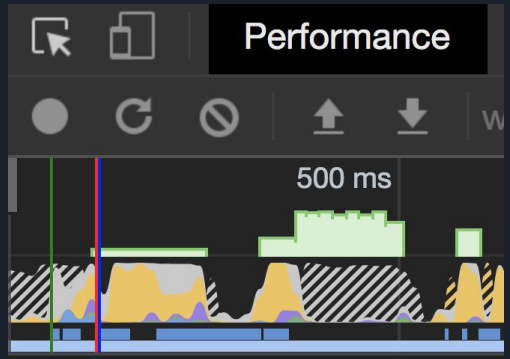
```
function bar() {
  return 2 * 2;
}
function foo() {
  const r = bar();
  r *= 3;
  return r;
}
foo()
```



	self	total
foo	3	4
bar	1	1

CPU profiler - ваш друг

- помогает найти медленные части вашего приложения
- в DevTools для JavaScript - сэмплирование - записанный профиль очень близок к реальной производительности



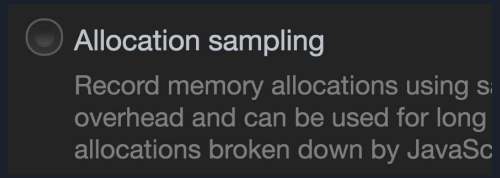
Неожиданное сэмплирование

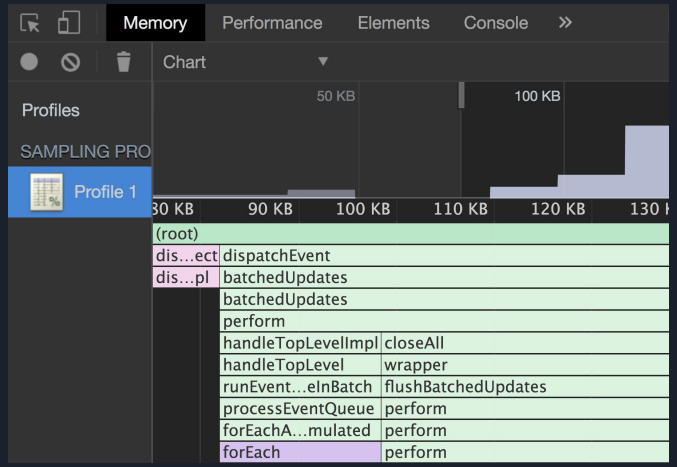
```
function bar() {
  new Int32Array(4);
}
function foo() {
  const r = bar();
  new Int32Array(4);
  new Int32Array(4);
}
foo()
```

	self	total
foo	8	12
bar	4	4

Allocation sampling - ваш друг

- показывает, какая часть вашего приложения выделяет большую часть памяти
- выделение памяти - не катастрофа - пока мы ее вовремя освобождаем

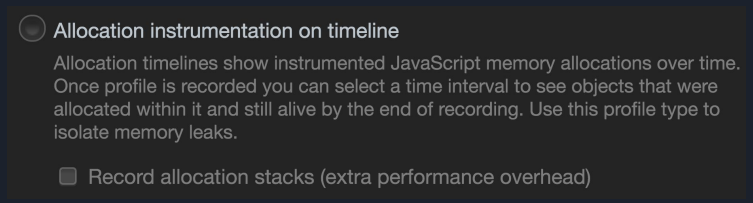


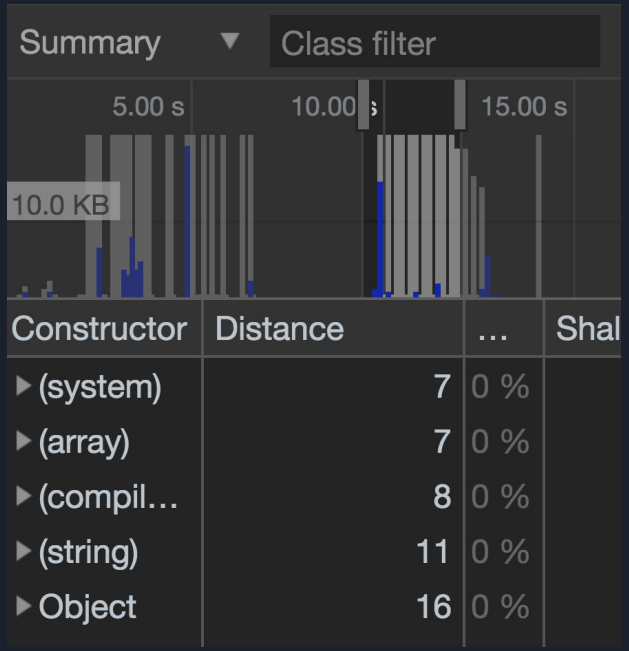


Allocation instrumentation - ваш друг

- позволяет изолировать утечки памяти

Показывает объекты выделенные за интервал времени и все еще живые к окончанию записи.





Промежуточные итоги

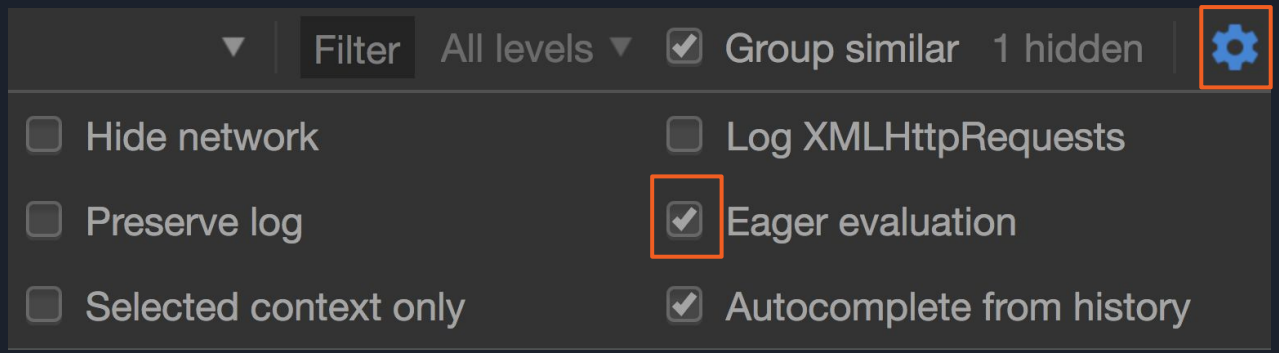
- CPU profiler показывает, сколько занимает вызов каждой функции в вашей программе
- Allocation sampling показывает, сколько памяти выделяет вызов каждой функции в вашей программе
- Allocation instrumentation показывает выделенные и все еще живые объекты

Далее:

- IO’18 Eager Evaluation
- IO’18 Step Into Worker

IO'18 eager evaluation

```
> [1n,2n,3n].map(v => v ** 1n)
< (3) [1n, 2n, 3n]
```



IO'18 eager evaluation

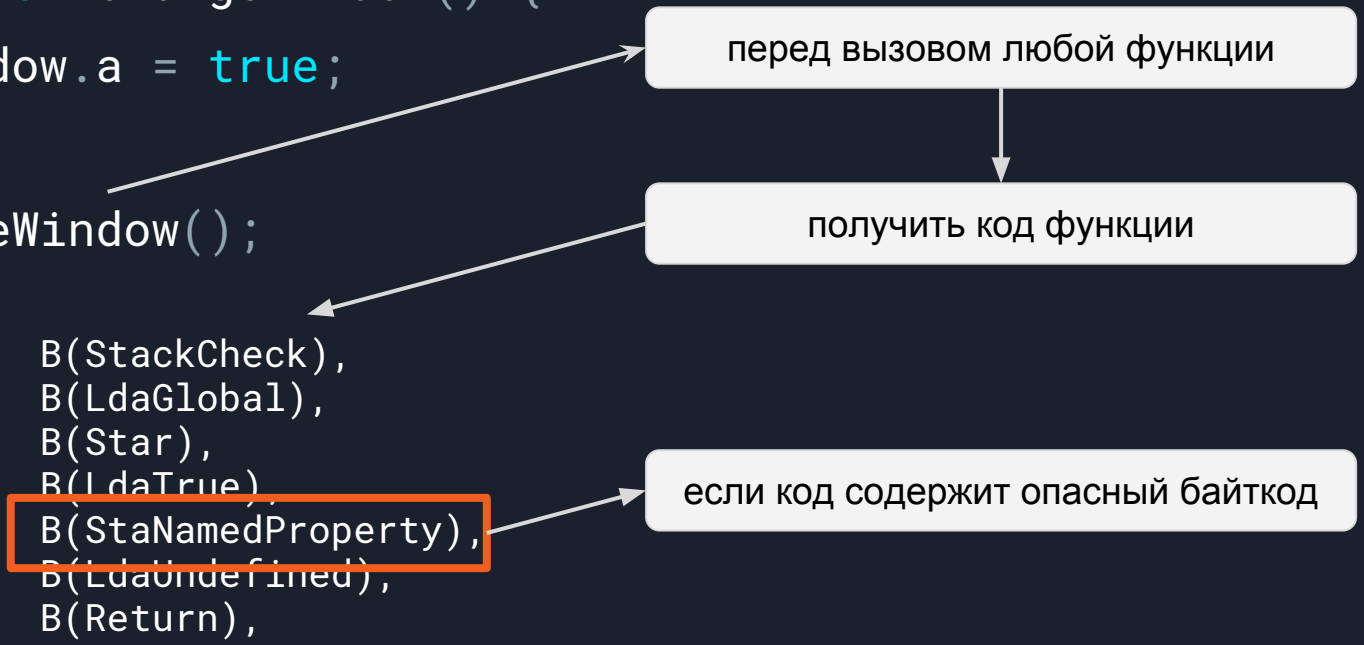
```
> [1n,2n,3n].map(v => v ** 1n)
<· (3) [1n, 2n, 3n]
```

- консоль показывает вам результат выражения на лету
- доступно в последнем Google Chrome Canary

```
while(true){}                  // очень долгий JS
window.location.href = '...' // очень опасный JS
```

Побочные эффекты: начало

```
function changeWindow() {
  window.a = true;
}
changeWindow();
```



Побочные эффекты: начало

```
function changeWindow() {
  window.a = true;
}
changeWindow();
```

перед вызовом любой функции

получить код функции

```
B(StackCheck),
B(LdaGlobal),
B(Star),
B(LdaTrue),
B(StaNamedProperty),
B(LdaUndefined),
B(Return),
```

если код содержит опасный байткод

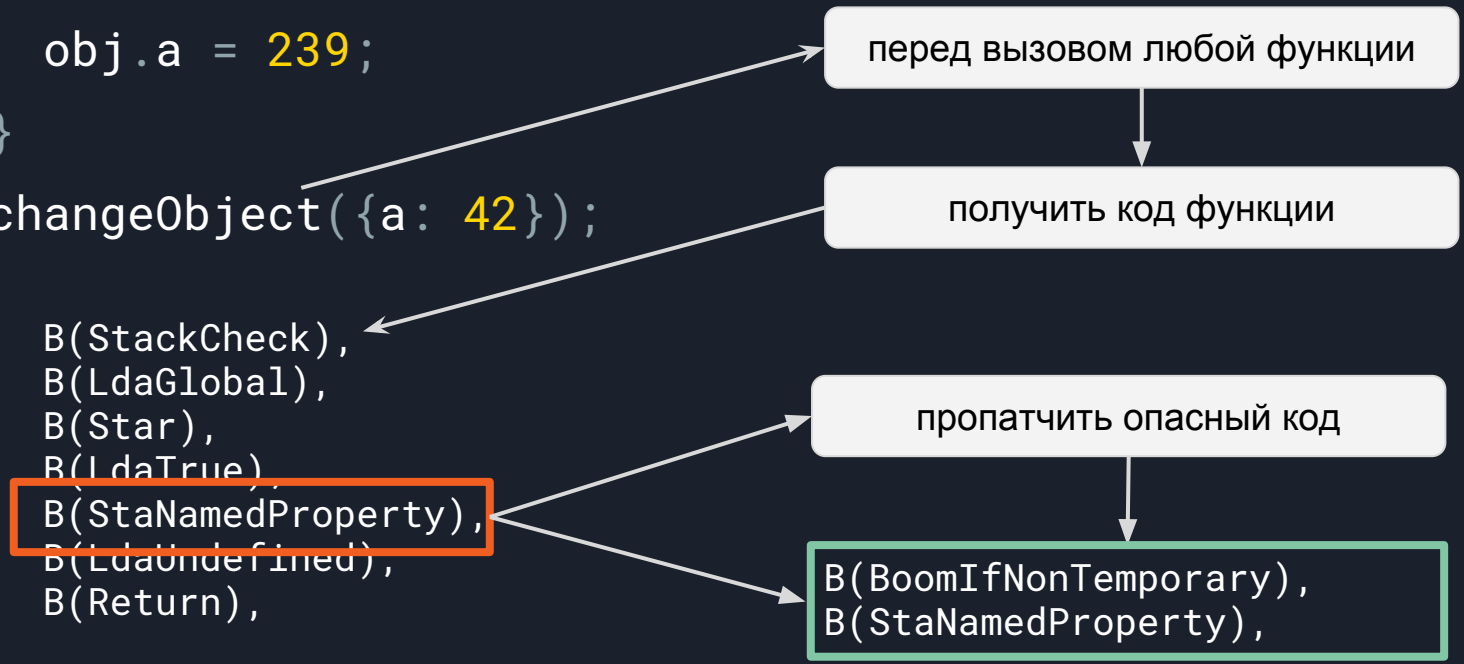
Побочные эффекты: возвращение

```
function storeTime(object) {
  object.time = Date.now();
}
storeTime(window);     // небезопасно
storeTime({time: 0}); // ???
```


TopWalls.com

Побочные эффекты: возвращение

```
function changeObject(obj) {
  obj.a = 239;
}
changeObject({a: 42});
```



Побочные эффекты: недостающий кусок

Как мы помечаем временные объекты?

Временный объект - объект, созданный во время выполнения кода в консоли и никуда не сохраненный.

```
{
  let a = {b: 2};
  a.b = 23;
  window.r = a;
}
```

начать отслеживать все аллокации

новый объект ⇒ временный

безопасный код

опасный код

Побочные эффекты: недостающий кусок

Как мы помечаем временные объекты?

Временный объект - объект, созданный во время выполнения кода в консоли и никуда не сохраненный.

```
{
  let a = {b: 2};
  a.b = 23;
  window.r = a;
}
```

начать отслеживать все аллокации

новый объект ⇒ временный

безопасный код

опасный код

Побочные эффекты: итоги

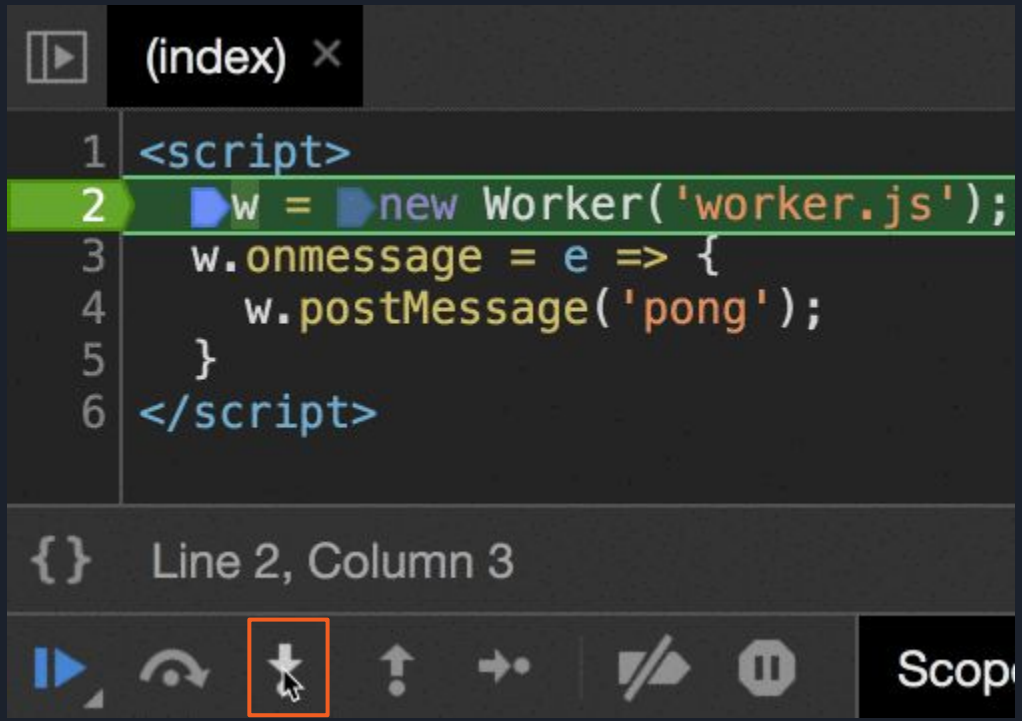
- дополняем все опасные участки кода функции дополнительной проверкой
- проверка разрешает опасные операции на временных объектах
- не просто выполняем ваш код, а гарантируем, что это безопасно

```
new Set([1,2,3]).delete(2)
```

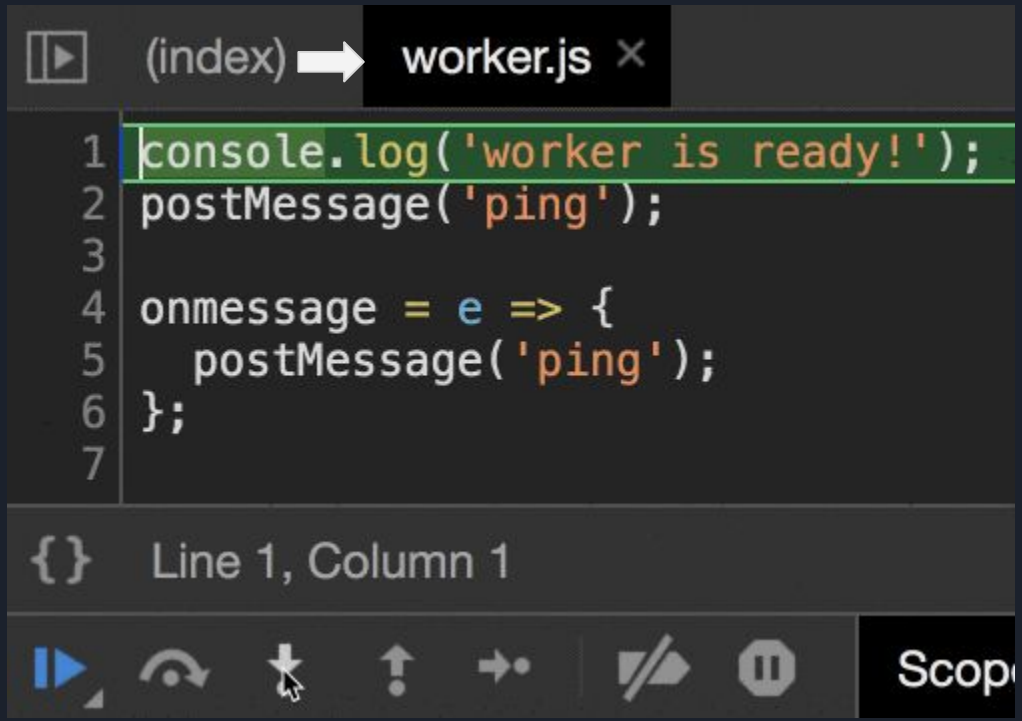
Eager evaluation - ваш друг

- пробуйте разные новые языковые фичи, отлаживайте регулярные выражения, экспериментируйте в консоли
- новый режим работы V8 гарантирует отсутствие побочных эффектов

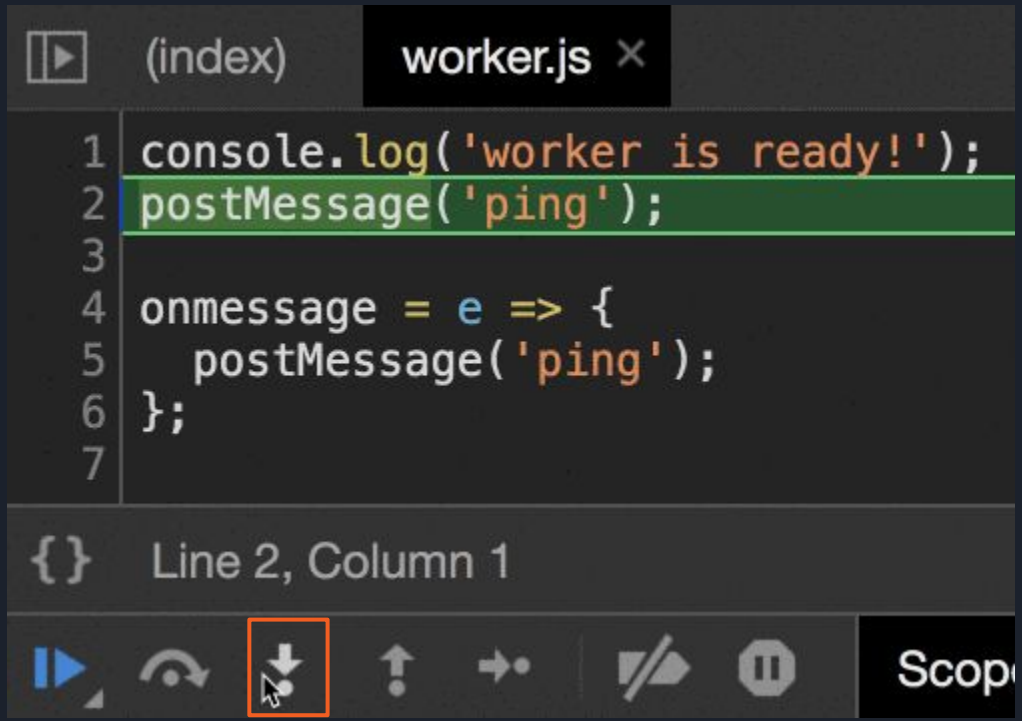
IO'18 step-into & workers = ❤️



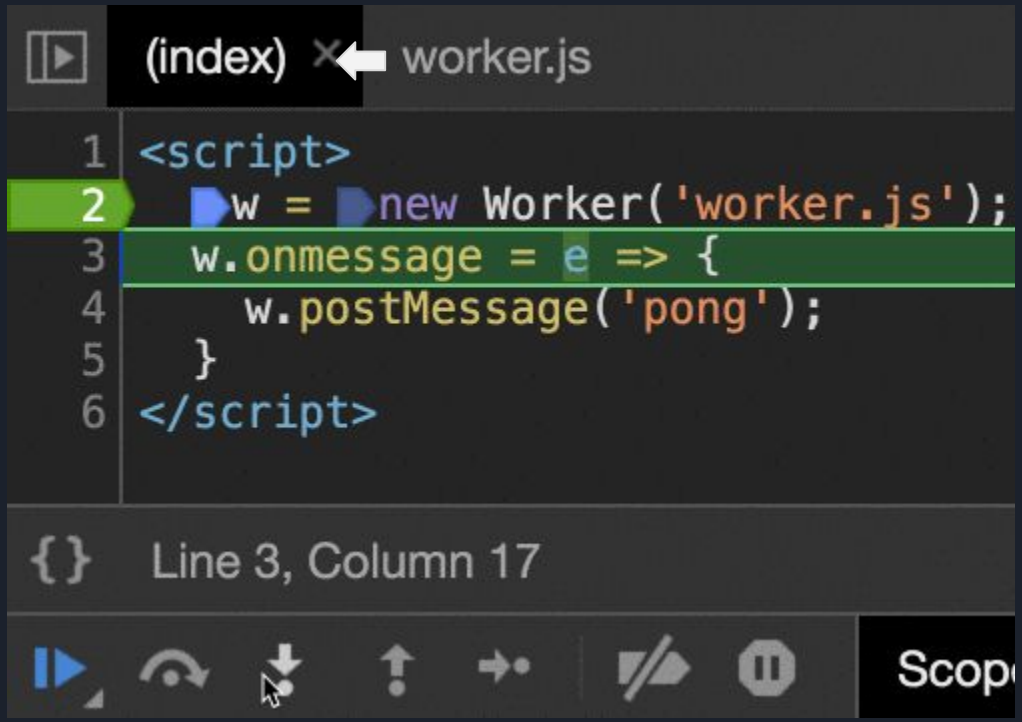
IO'18 step-into & workers = ❤️



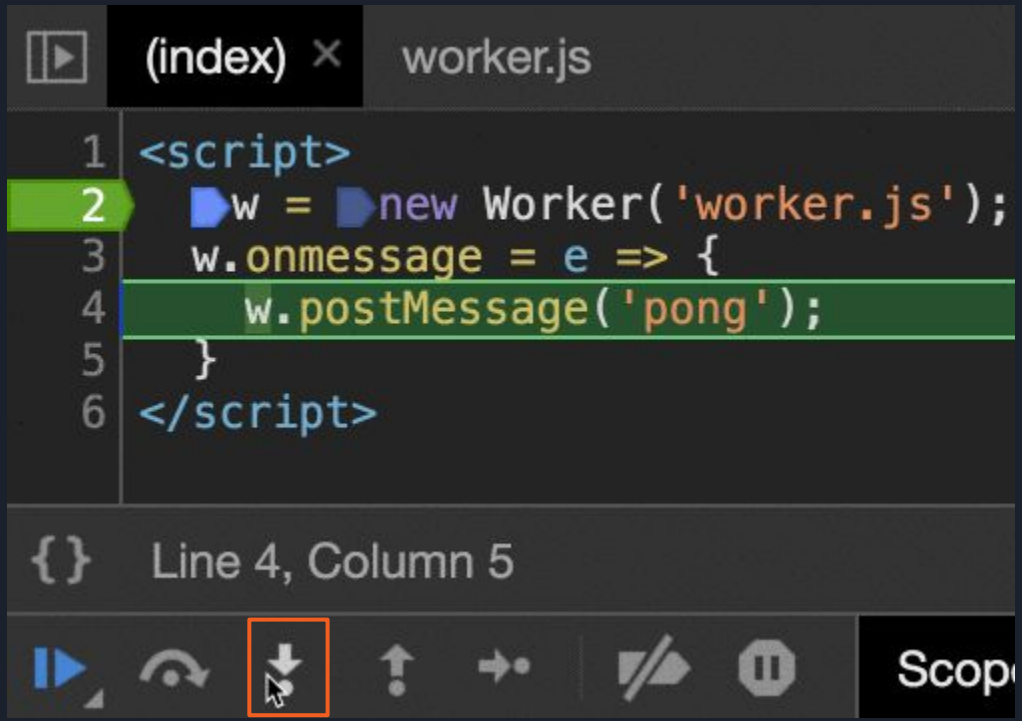
IO'18 step-into & workers = ❤️



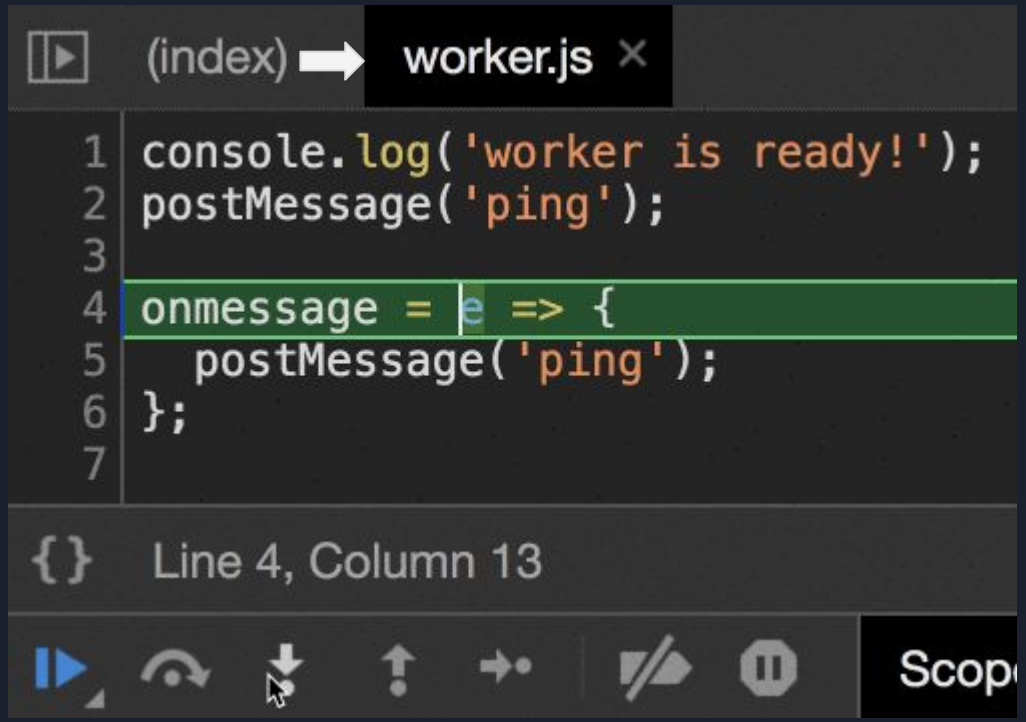
IO'18 step-into & workers = ❤️



IO'18 step-into & workers = ❤️

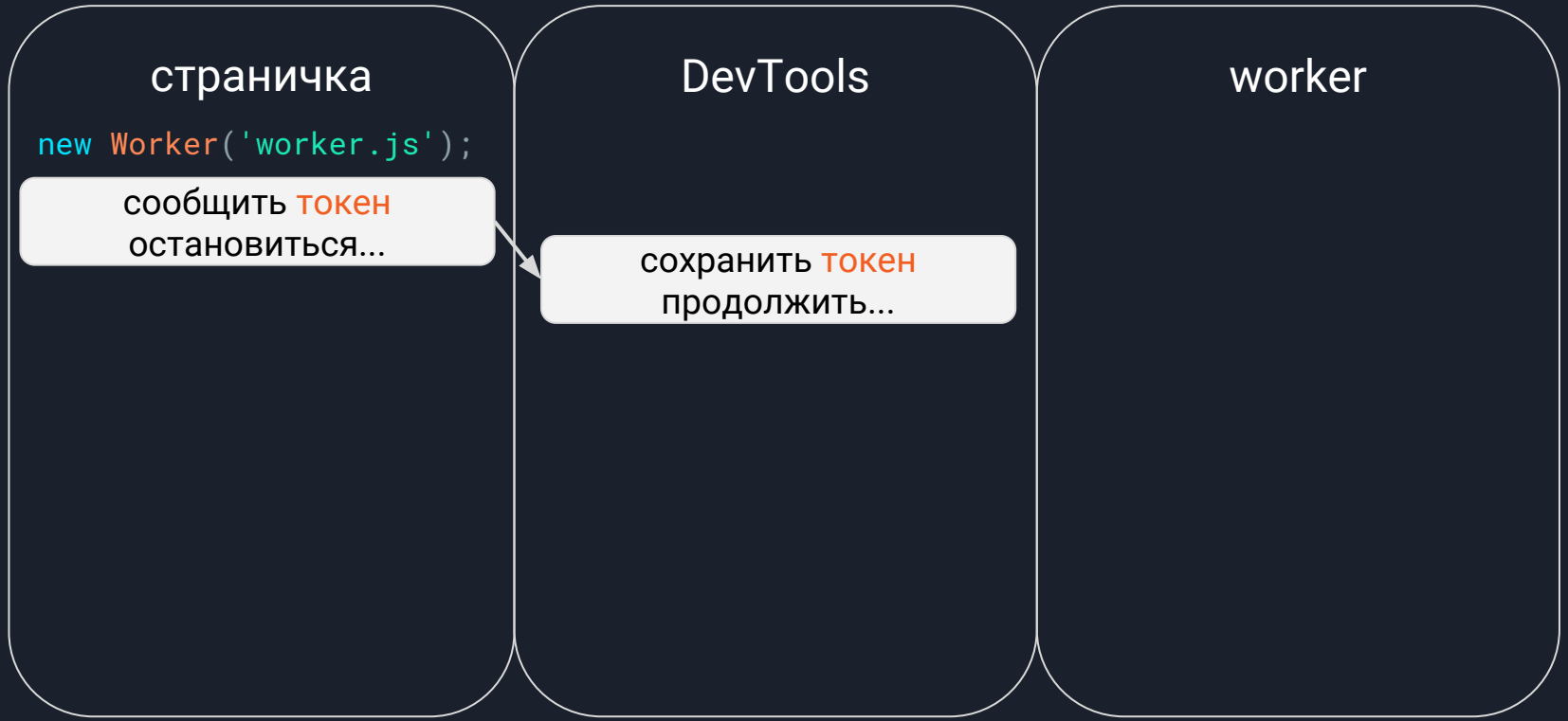


IO'18 step-into & workers = ❤️

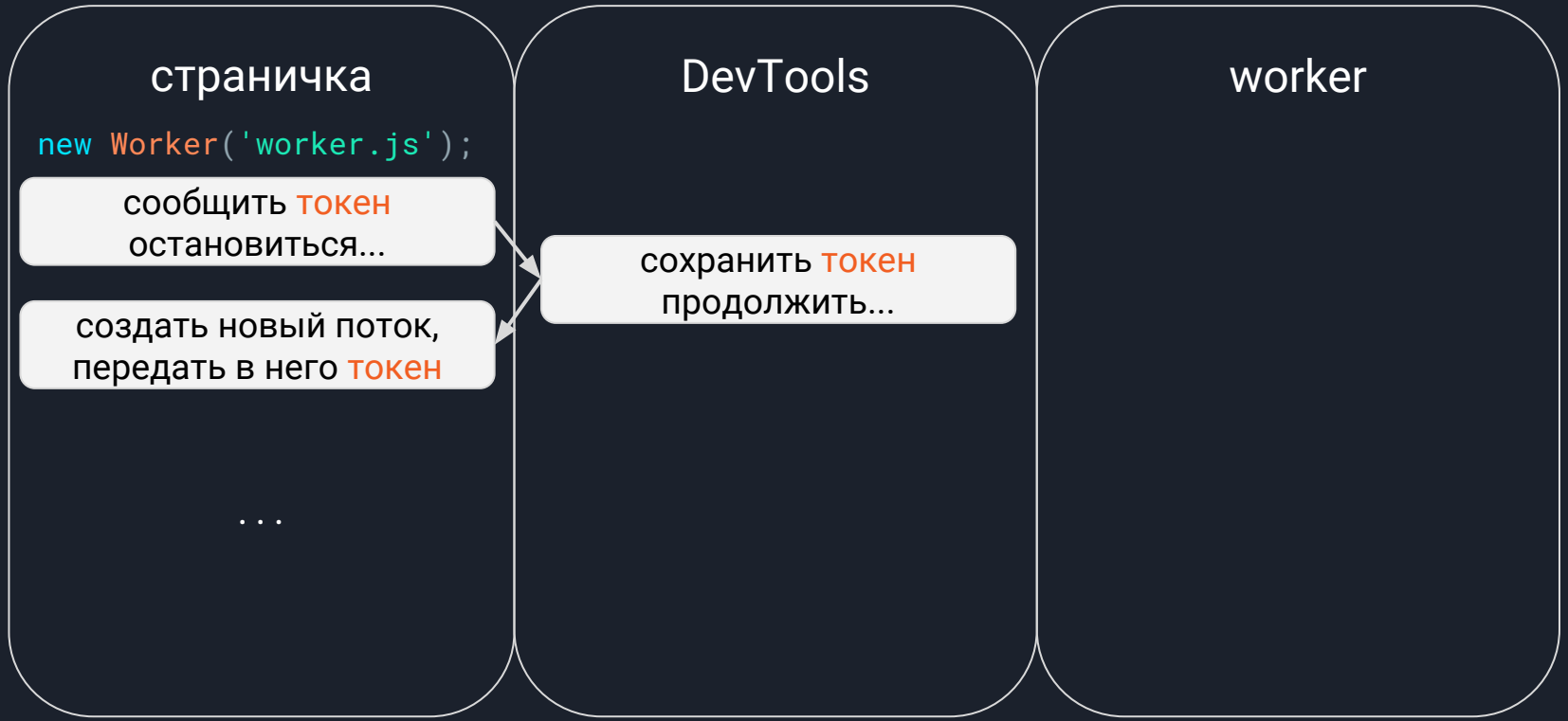


IO'18 step-into & workers: new Worker

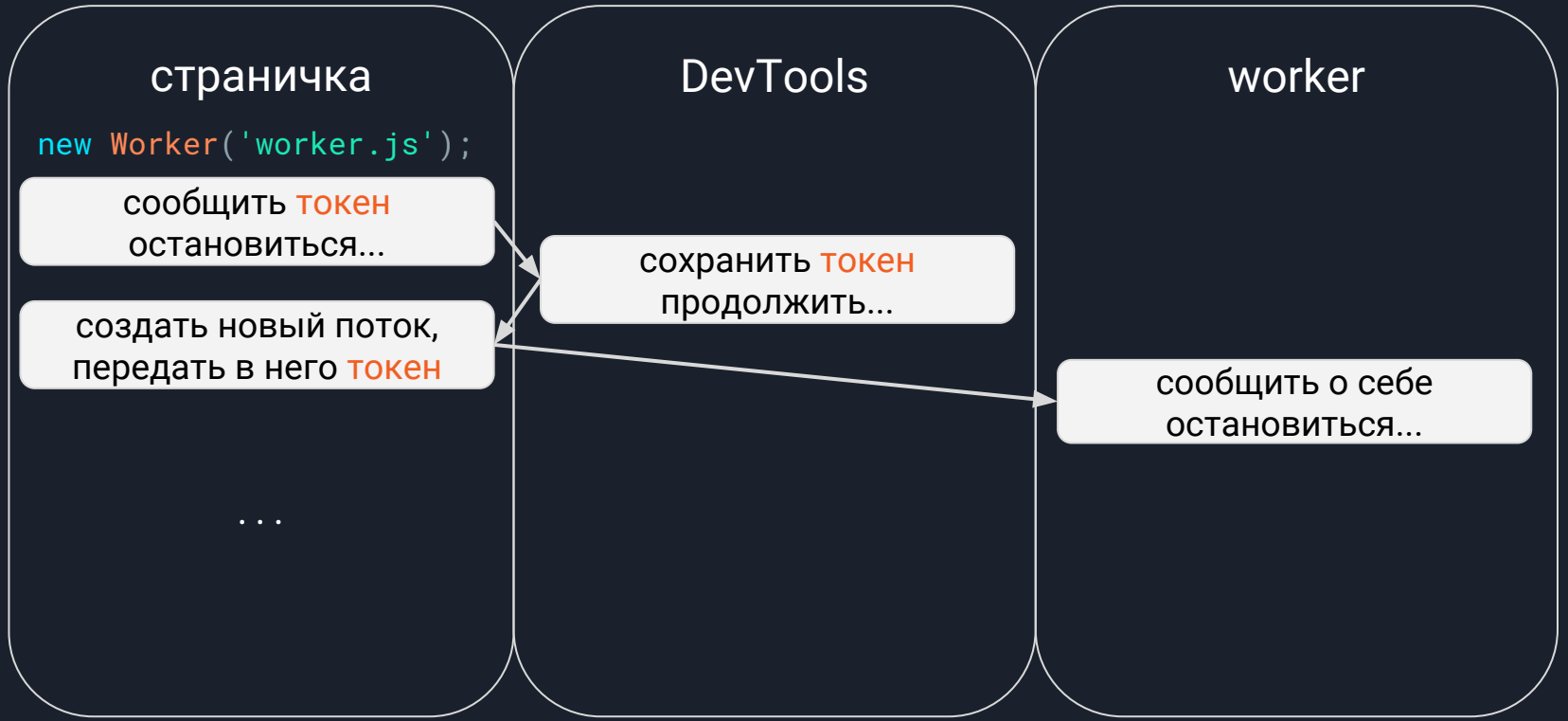
IO'18 step-into & workers: new Worker



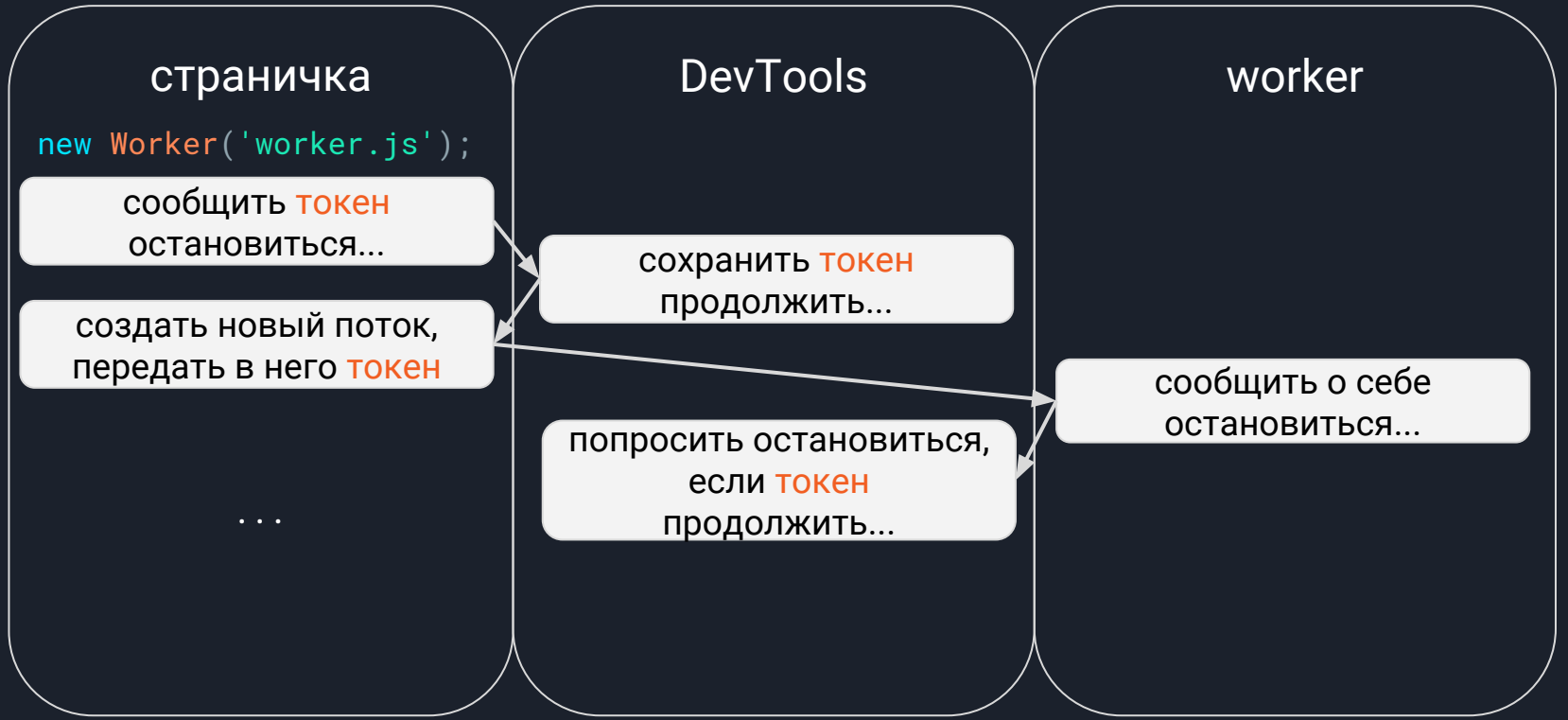
IO'18 step-into & workers: new Worker



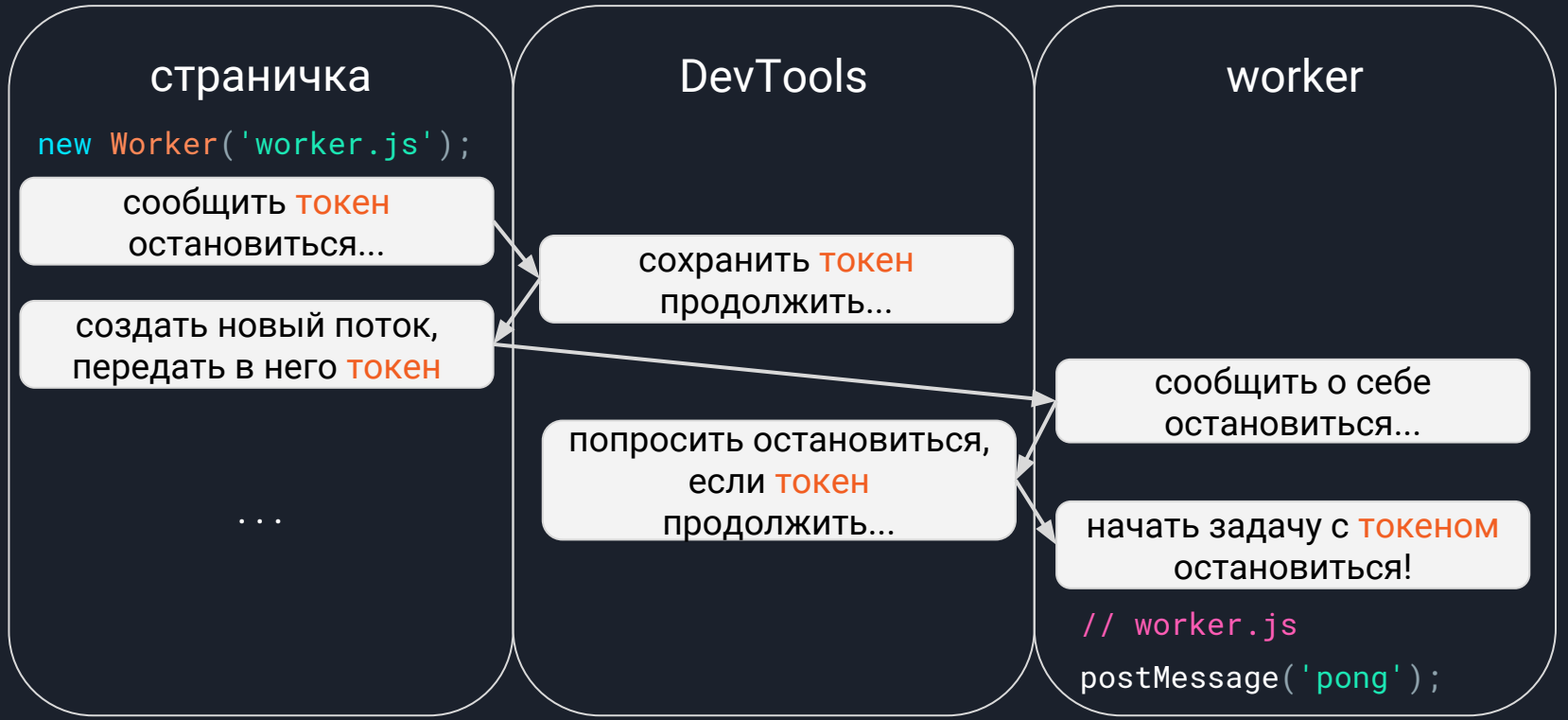
IO'18 step-into & workers: new Worker



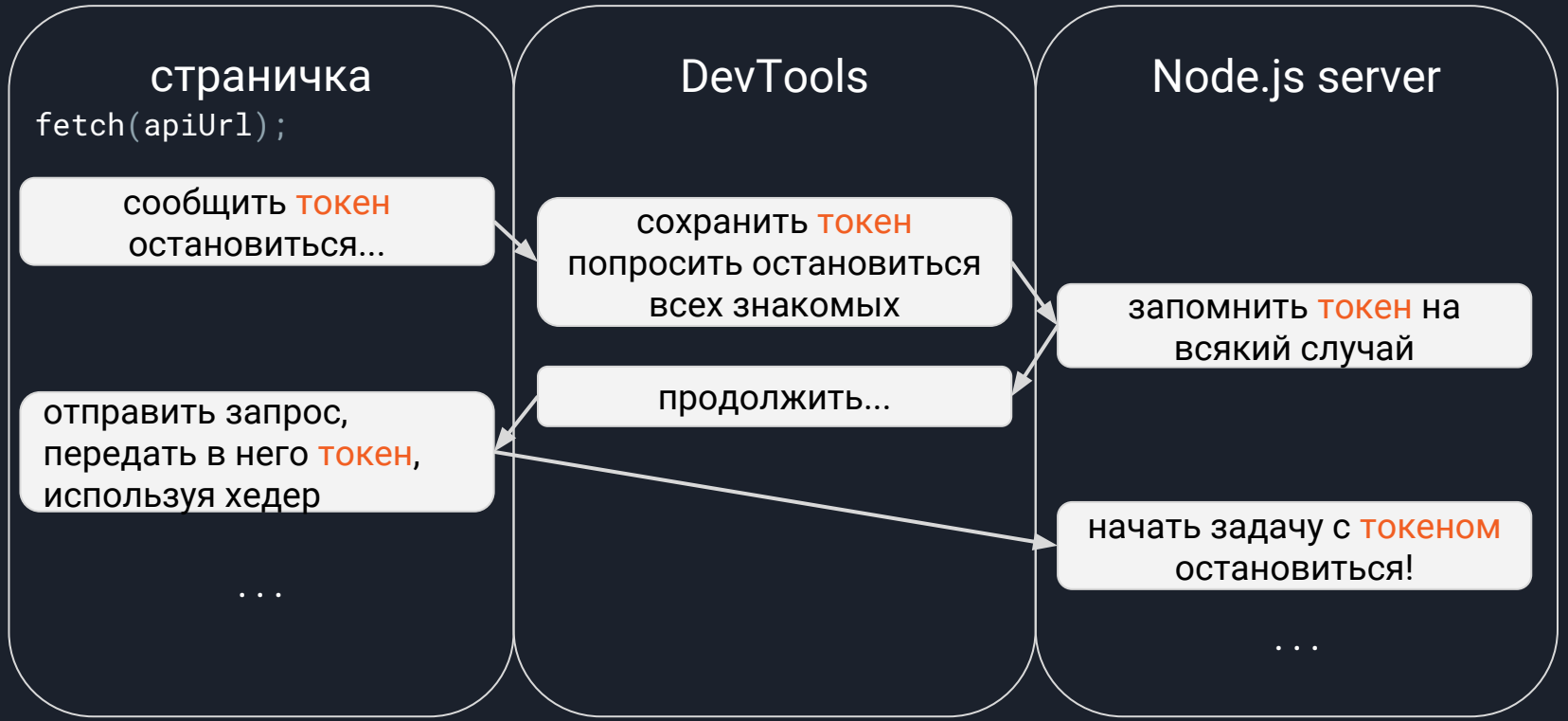
IO'18 step-into & workers: new Worker



IO'18 step-into & workers: new Worker



IO'18 step-into: универсальная схема

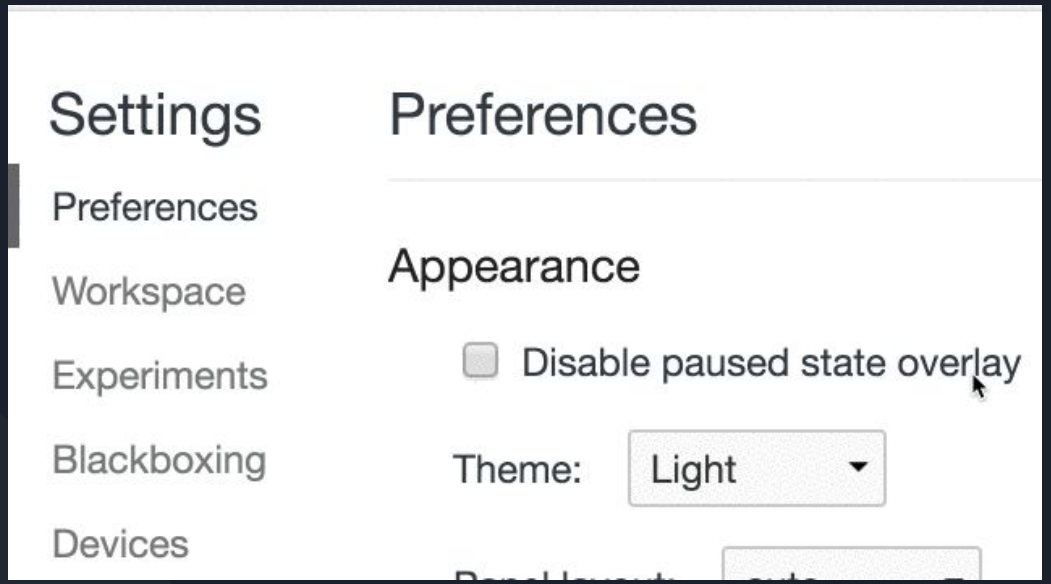


IO'18 step-into worker - ваш друг

- используйте step-into для перехода из воркера в страничку и обратно
- используйте step-into для перехода в скрипт воркера из конструктора
- используйте step-into для асинхронных вызовов

Всем спасибо!

Сцена после титров

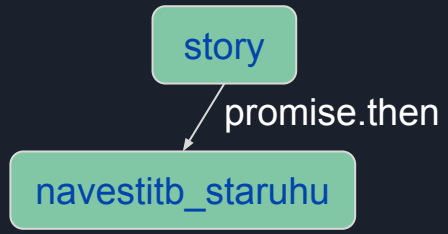


console.assert(true, ...)

```
function division(a,b) {
  console.assert(b !== 0, 'на ноль нельзя');
  return a / b;
}
// ...
var assert = console.assert;
console.assert = function(a, b, c, d) {
  if (a) return;
  assert(a, b, c, d);
}
```

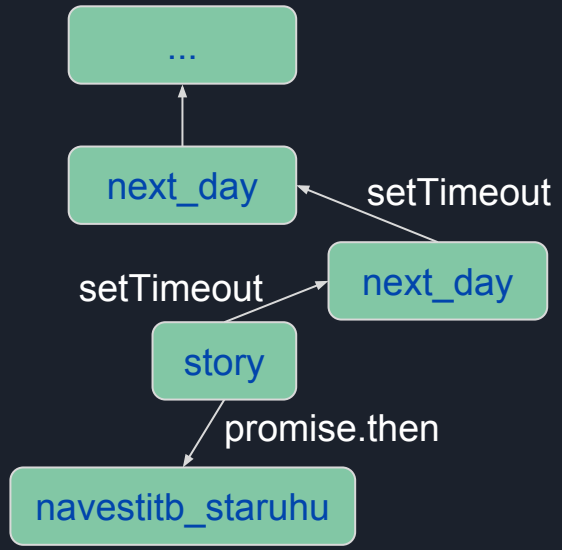
Асинхронные стеки

```
function navestit_staruhu() {
  primenitb_topor();
  setTimeout(arest_Mikoly, SOME_TIME1);
  rodion_raskaylsya.then(priznatbsya);
}
function priznatbsya() {
  setTimeout(uehatb_v_Sibire, SOME_TIME2);
}
function story() {
  rodion_poluchil_pismo.then(navestitbStaruhu);
  function next_day() {
    update_world_state();
    setTimeout(next_day, ONE_DAY);
  }
  next_day();
}
```



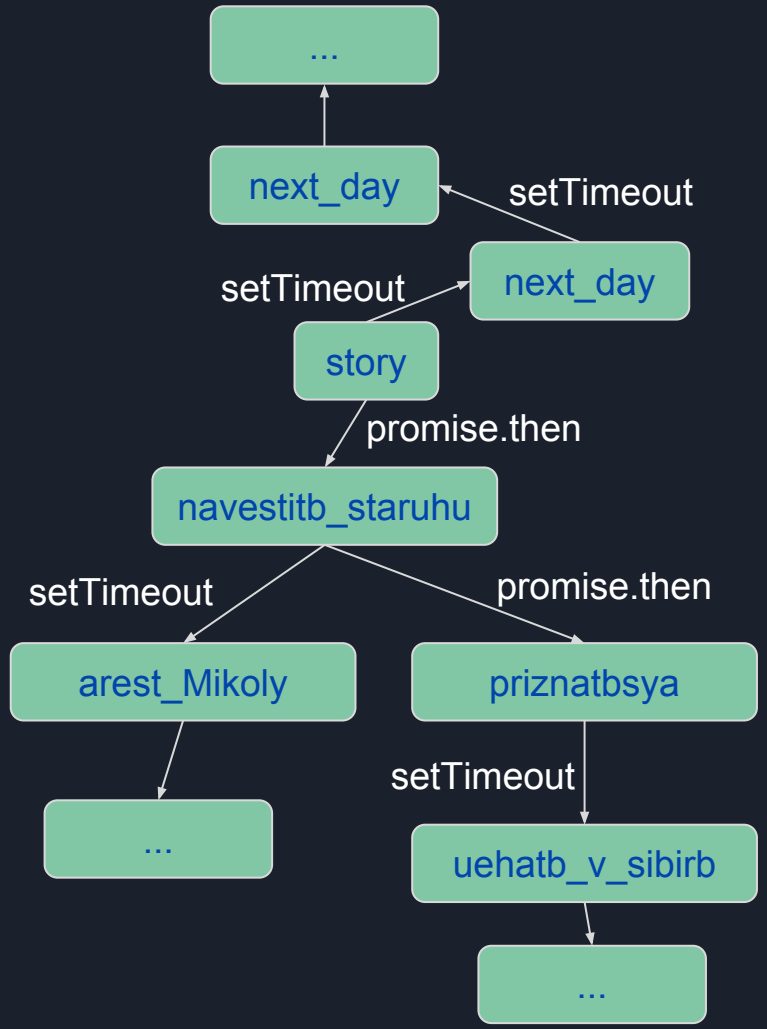
Асинхронные стеки

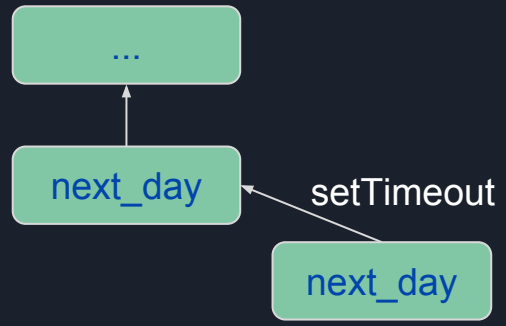
```
function navestit_staruhu() {
  primenitb_topor();
  setTimeout(arest_Mikoly, SOME_TIME1);
  rodion_raskaylsya.then(priznatbsya);
}
function priznatbsya() {
  setTimeout(uehatb_v_Sibire, SOME_TIME2);
}
function story() {
  rodion_poluchil_pismo.then(navestitbStaruhu);
  function next_day() {
    update_world_state();
    setTimeout(next_day, ONE_DAY);
  }
  next_day();
}
```

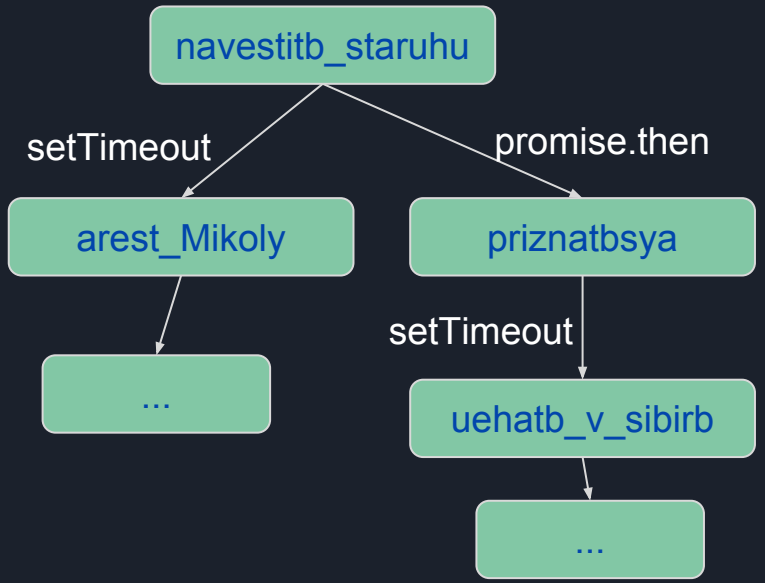


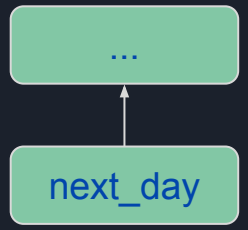
Асинхронные стеки

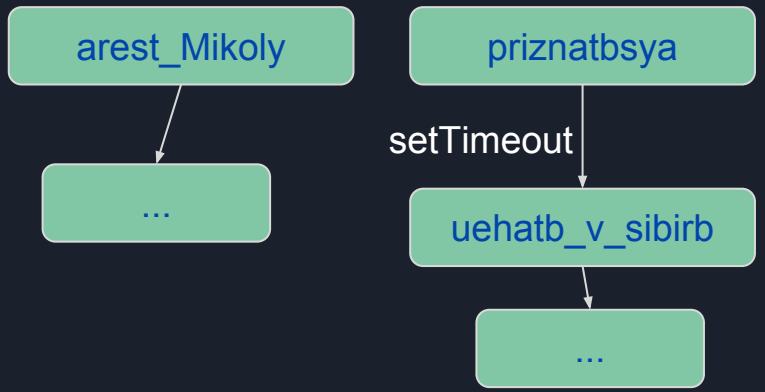
```
function navestit_staruhu() {
  primenitb_topor();
  setTimeout(arest_Mikoly, SOME_TIME1);
  rodion_raskaylsya.then(priznatbsya);
}
function priznatbsya() {
  setTimeout(uehatb_v_Sibire, SOME_TIME2);
}
function story() {
  rodion_poluchil_pismo.then(navestitbStaruhu);
  function next_day() {
    update_world_state();
    setTimeout(next_day, ONE_DAY);
  }
  next_day();
}
```

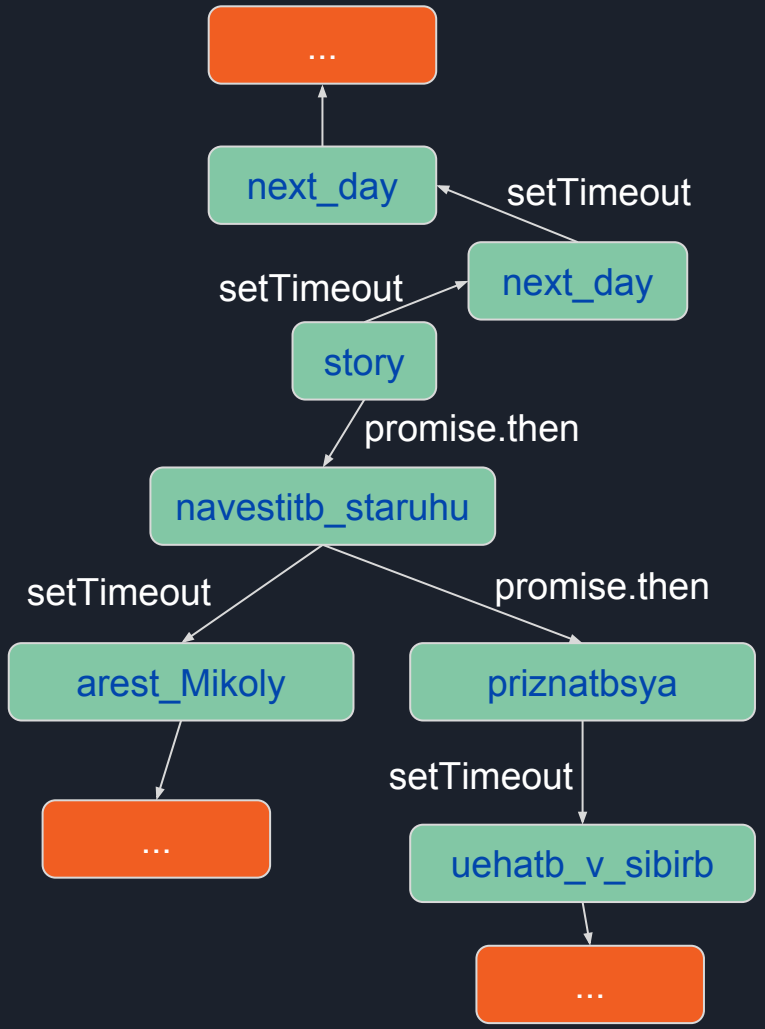


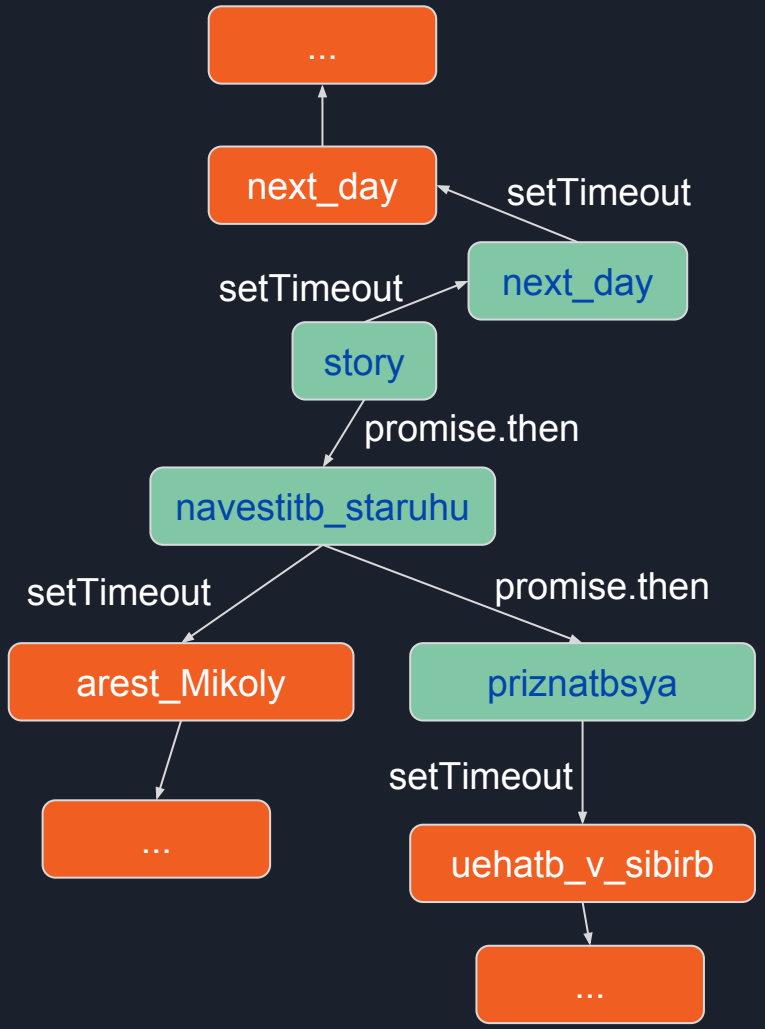

...
next_day
setTimeout
next_day

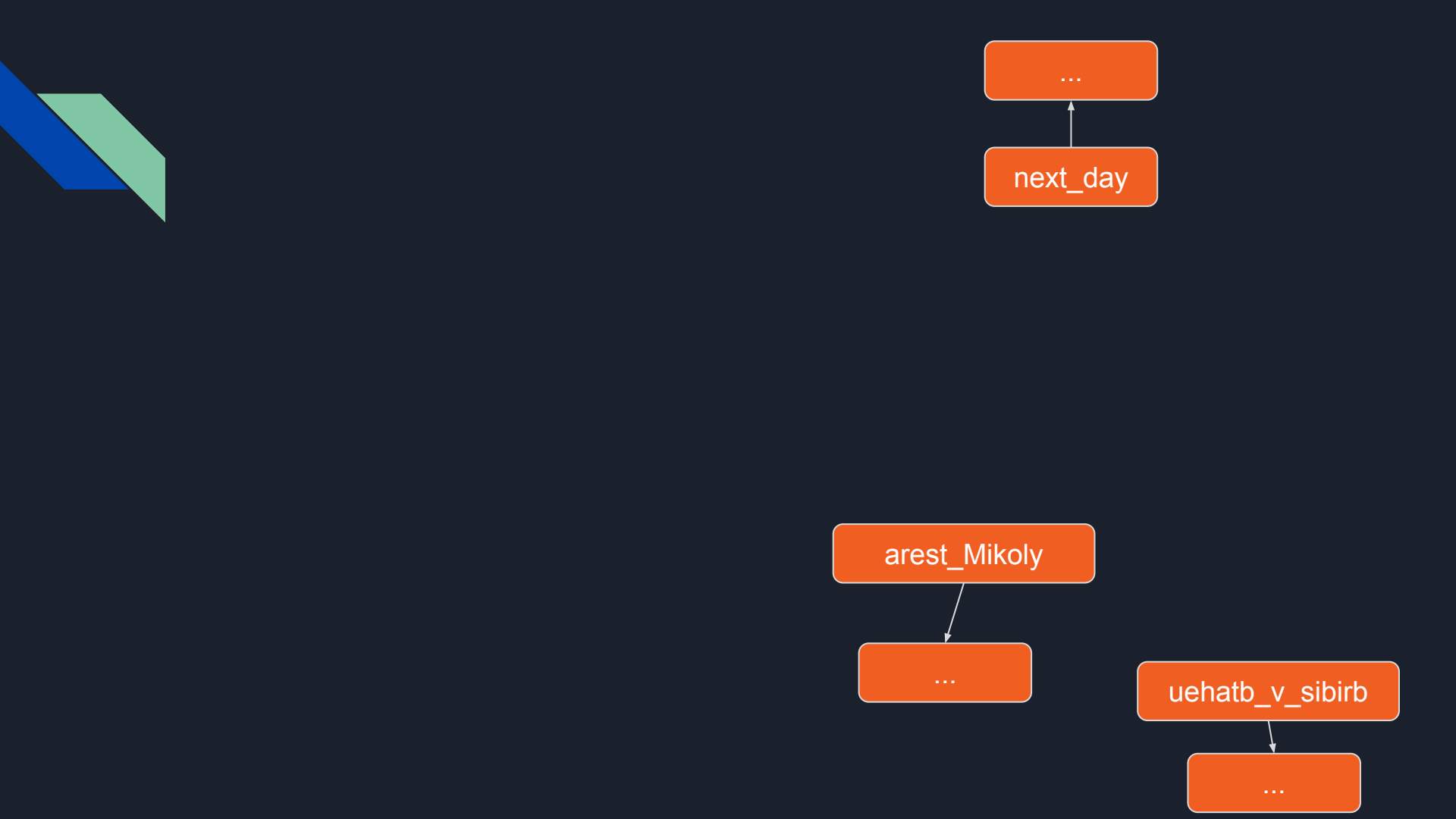

navestitb_staruhu
setTimeout
promise.then
arest_Mikoly
priznatbsya
...
setTimeout
uehatb_v_sibirb
...


...
next_day


arest_Mikoly
...
priznatbsya
setTimeout
uehatb_v_sibirb
...


...
next_day
setTimeout
next_day
setTimeout
story
promise.then
navestitb_staruhu
setTimeout
promise.then
arest_Mikoly
priznatbsya
...
setTimeout
uehatb_v_sibirb
...

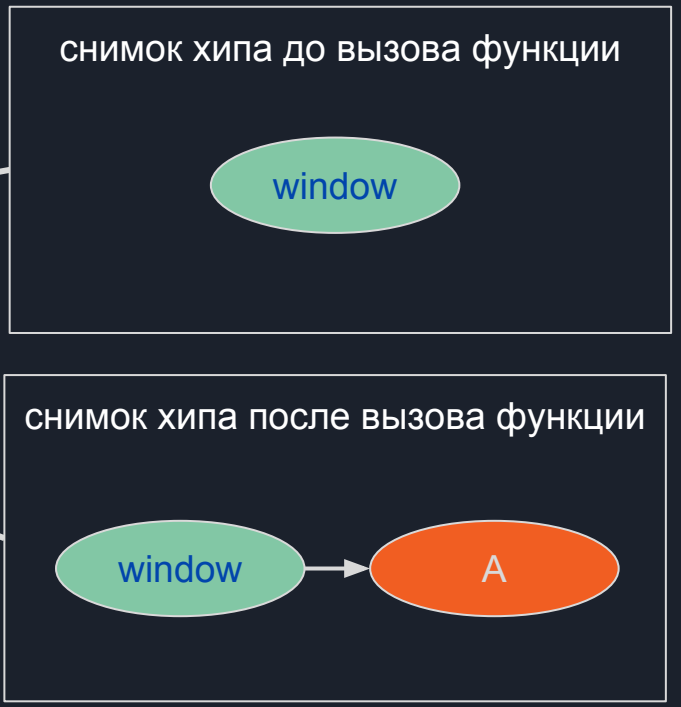

...
next_day
setTimeout
next_day
setTimeout
story
promise.then
navestitb_staruhu
setTimeout
promise.then
arest_Mikoly
priznatbsya
...
setTimeout
uehatb_v_sibirb
...


...
next_day
arest_Mikoly
...
uehatb_v_sibirb
...

Инструментирование VS сэмплирование

- инструментирование
  - всегда включает все функции и содержит очень точную информацию о том, какие функции вызывались
  - неравномерно искажает результат
  - сложней в реализации
- сэмплирование
  - равномерно искажает результат
  - точность ограничена частотой сэмплирования

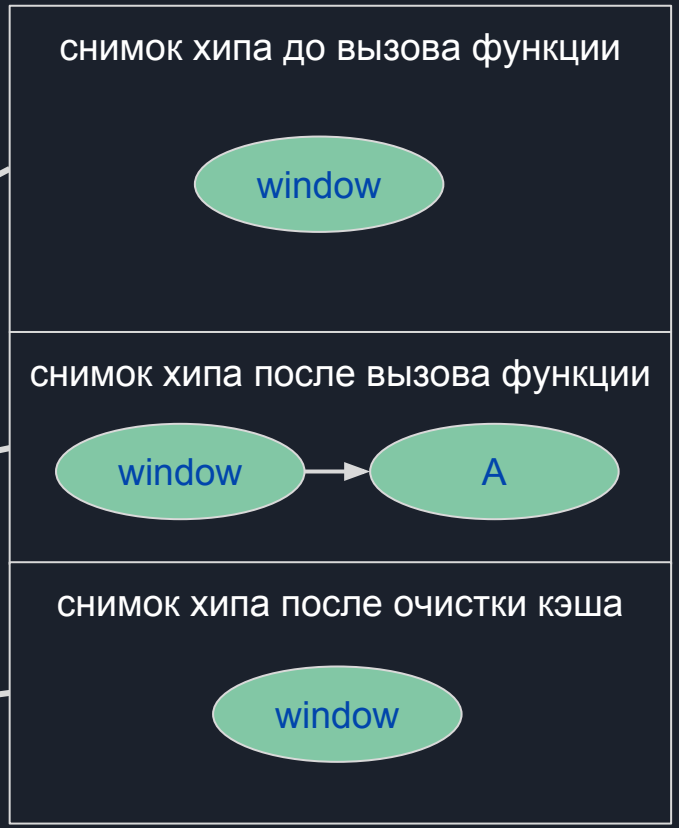
```
function leakA() {
  window.leak = new A();
}
leakA();
```

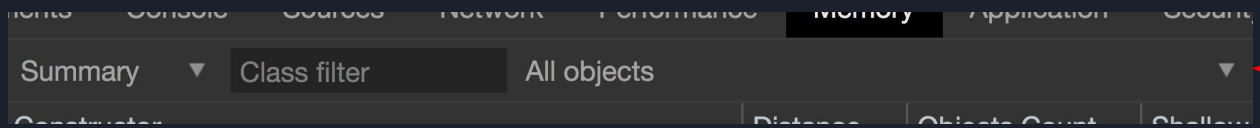


Heap snapshot

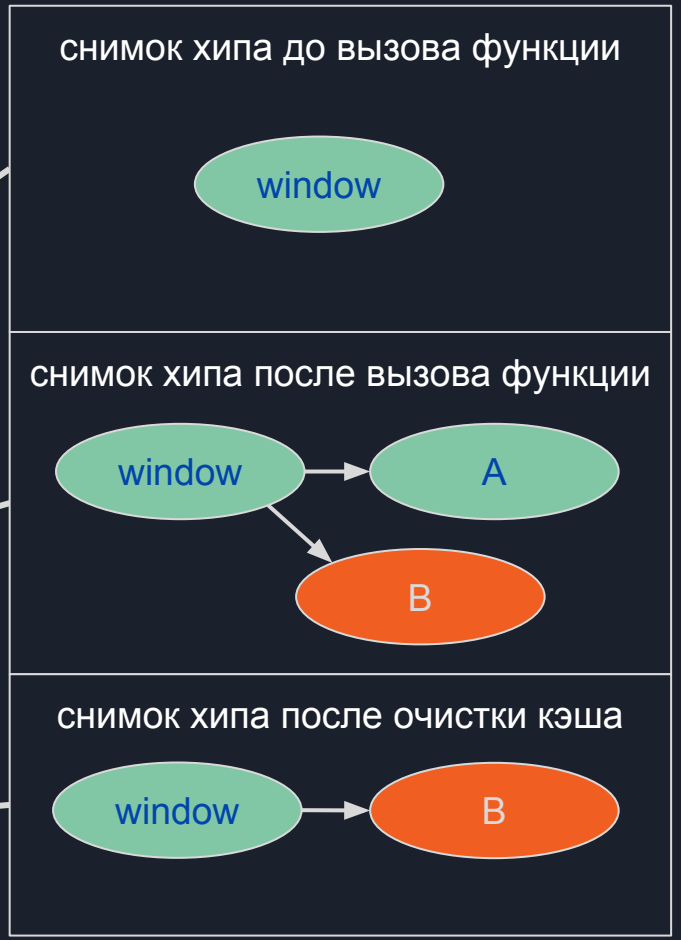
Heap snapshot profiles show me

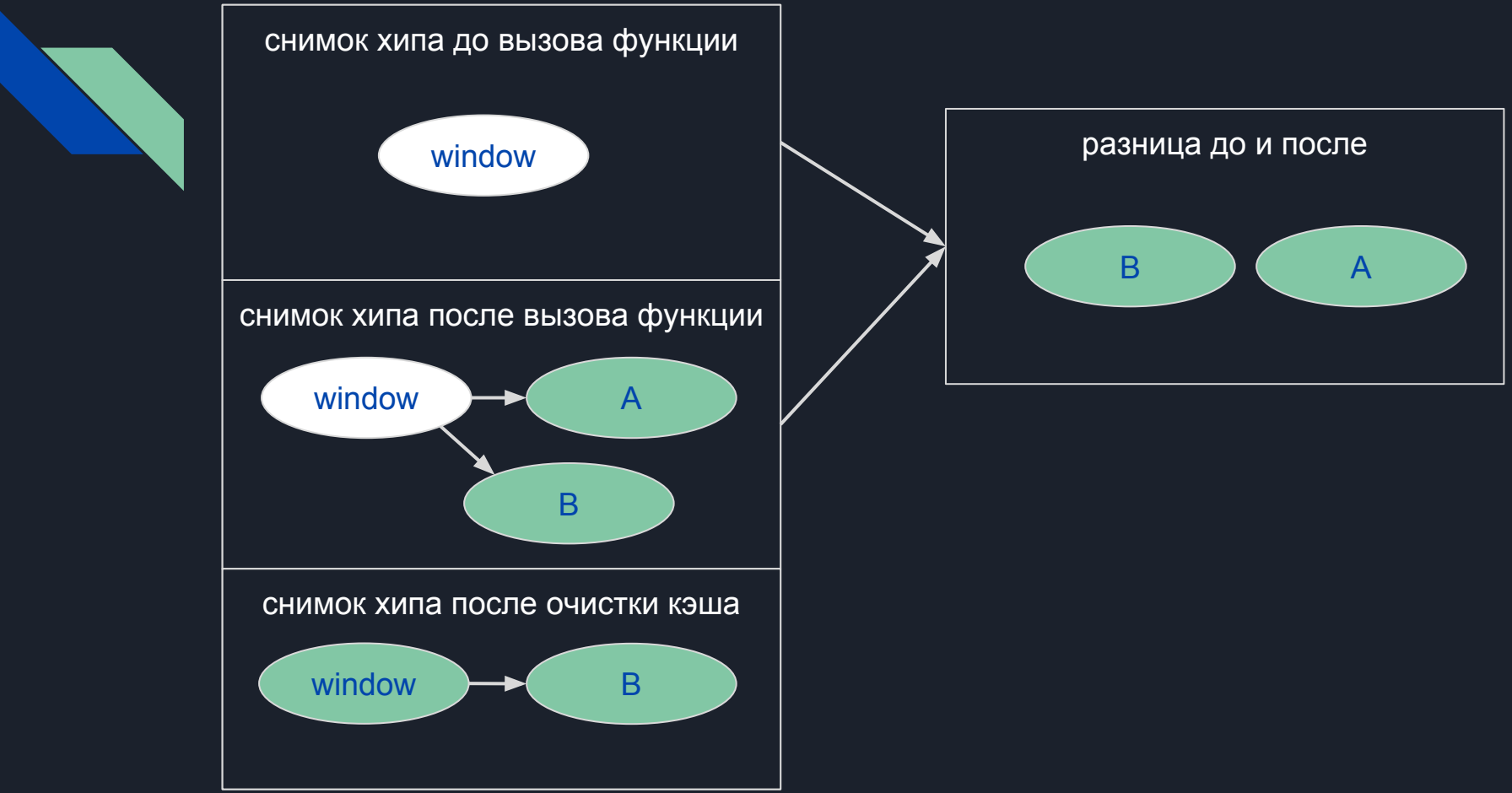
```
function getValue() {
  if (!window.cache)
    window.cache = new A();
  return window.cache;
}
...
getValue();
...
delete window.cache;
```

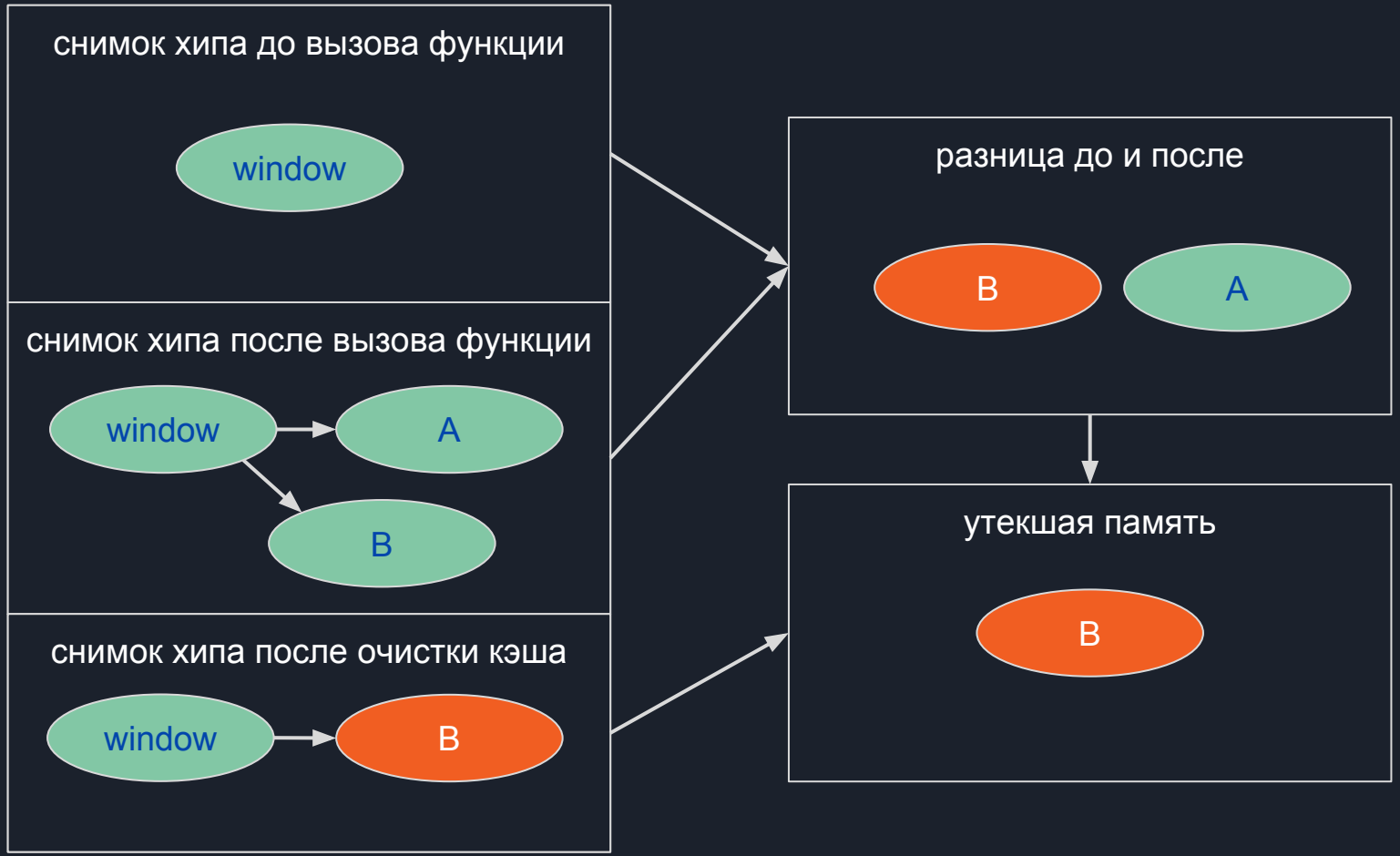




```
function getValue() {
  if (!window.cache)
    window.cache = new A();
  window.leak = new B();
  return window.cache;
}
...
getValue();
...
delete window.cache;
```




снимок хипа до вызова функции
window
снимок хипа после вызова функции
window
A
B
снимок хипа после очистки кэша
window
B
разница до и после
B
A


снимок хипа до вызова функции
window
снимок хипа после вызова функции
window
A
B
снимок хипа после очистки кэша
window
B
разница до и после
B
A
утекшая память
B