Инкрементальные вычисления в UI-фреймворках

Приятно познакомиться



Василий
Алфертьев

Open Solutions_
software development

Telegram: @alfertev2012

GitHub: @alfertev2014

React

TypeScript

Чем ещё владею

- 5+ лет в C++
  - системное программирование
  - Linux
  - UI на Qt

- ~6 лет в Java
  - backend на spring
  - базы данных
  - монолиты, микросервисы...

Чем ещё владею

- 5+ лет в C++
  - системное программирование
  - Linux
  - UI на Qt
- ~6 лет в **Java**
  - backend на spring
  - базы данных
  - монолиты, микросервисы...

- Увлекаюсь
  - дизайном языков программирования
  - best practices и архитектурой ПО
  - computer science
- Тянет разбираться в
  - компиляторах, IDE и инструментах
  - **Устройстве UI-фреймворков и библиотек**

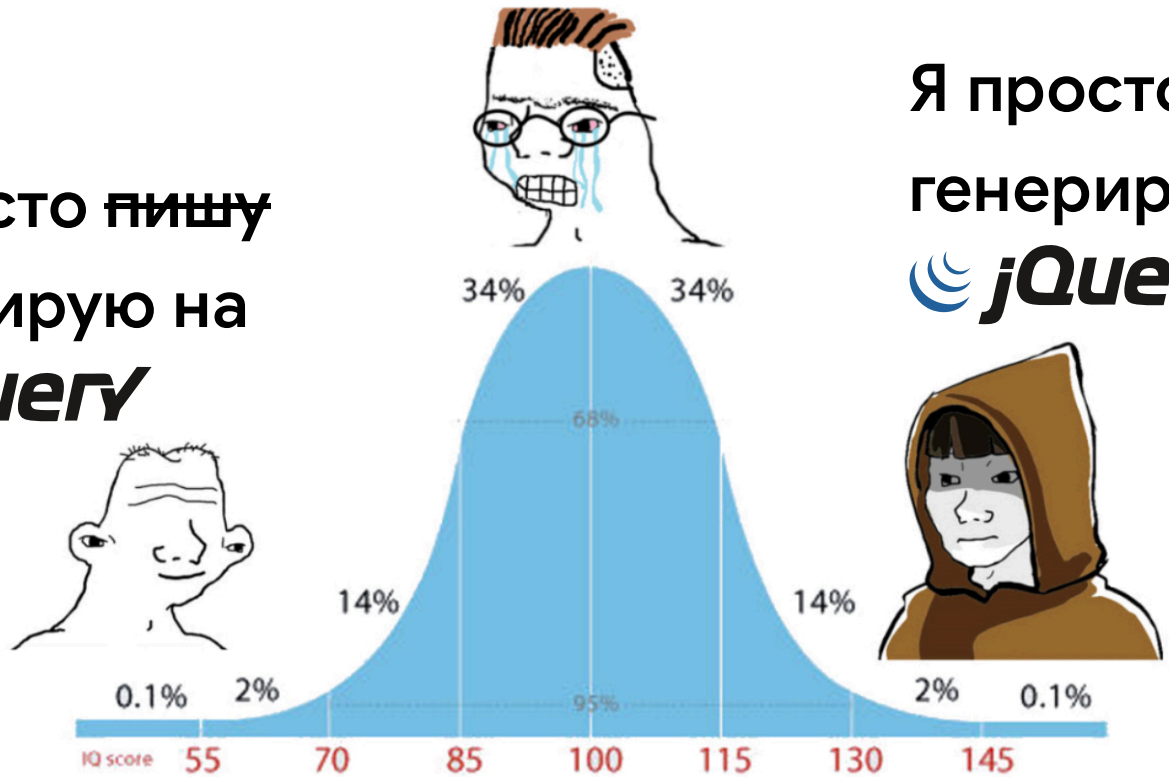
Чем ещё владею

- 5+ лет в C++
  - системное программирование
  - Linux
  - UI на Qt
- ~6 лет в **Java**
  - backend на spring
  - базы данных
  - монолиты, микросервисы...
- Увлекаюсь
  - дизайном языков программирования
  - best practices и архитектурой ПО
  - computer science
- Тянет разбираться в
  - компиляторах, IDE и инструментах
  - **Устройстве UI-фреймворков и библиотек**

UI-фреймворки

Vue лучше, чем React!

Я просто ~~пишу~~ генерирую на jQuery



Я просто ~~пишу~~ генерирую на jQuery


S
A
B
C
D
JS
CSS
TS
Claude
yup
AXIOS
MOCHA
jQuery
XSTATE
Sass
styled components
Jōtai
Gulp

📌 Pin Images



Чего мы хотим от UI-фреймворка?

Чего мы хотим от UI-фреймворка?

Повышение продуктивности при разработке UI

Чего мы хотим от UI-фреймворка?

Повышение продуктивности при разработке UI

- Улучшение Developer Experience в понятной парадигме

Чего мы хотим от UI-фреймворка?

Повышение продуктивности при разработке UI

- Улучшение Developer Experience в понятной парадигме
- Автоматически оптимизированный результат

О чём доклад

О чём доклад

- **Инкрементальные вычисления** как модель исполнения чистых функций

О чём доклад

- **Инкрементальные вычисления** как модель исполнения чистых функций
- "Fine-grained reactivity" для точечного обновления UI

О чём доклад

- **Инкрементальные вычисления** как модель исполнения чистых функций
- "Fine-grained reactivity" для точечного обновления UI
- UI-фреймворк - слой абстракции над JavaScript

О чём доклад

- **Инкрементальные вычисления** как модель исполнения чистых функций
- "Fine-grained reactivity" для точечного обновления UI
- UI-фреймворк - слой абстракции над JavaScript
- Декларативный DSL для UI с семантикой инкрементального исполнения

О чём доклад

- **Инкрементальные вычисления** как модель исполнения чистых функций
- "Fine-grained reactivity" для точечного обновления UI
- UI-фреймворк - слой абстракции над JavaScript
- Декларативный DSL для UI с семантикой инкрементального исполнения
- "Proof-of-concept" UI-фреймворк с инкрементальными вычислениями

Ключевые моменты

Ключевые моменты

🚀 **Реактивность**

Обновление представления
при изменении состояния

Ключевые моменты

🚀 **Реактивность**

Обновление представления
при изменении состояния

📜 **Декларативность**

Структура и поведение UI -
на декларативном DSL

Ключевые моменты

🚀 Реактивность

Обновление представления
при изменении состояния

📜 Декларативность

Структура и поведение UI -
на декларативном DSL

✅ Строгость

Строгая типизация, строгие абстракции,
строгие соглашения и ограничения

Фреймворки и слои абстракции

- Коробочные решения
- Конструкторы сайтов, CMS, NoCode
- Каталоги готовых компонентов, LowCode
- UI-киты, дизайн-системы
- Библиотеки компонентов, утилит и тем
- **Слой абстракции с другой парадигмой**  ...
- Общие Javascript-библиотеки для типовых задач
- Язык JavaScript и его runtime

Фреймворки и слои абстракции

- Коробочные решения
- Конструкторы сайтов, CMS, NoCode
- Каталоги готовых компонентов, LowCode
- UI-киты, дизайн-системы
- Библиотеки компонентов, утилит и тем
- **Слой абстракции с другой парадигмой**  ...
- Общие Javascript-библиотеки для типовых задач
- Язык JavaScript и его runtime

Фреймворки похожи друг на друга

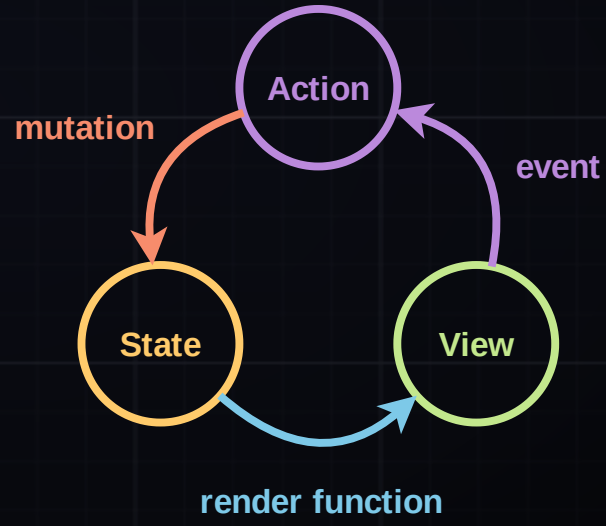


Фреймворки похожи друг на друга

Дерево компонентов



Цикл обновления



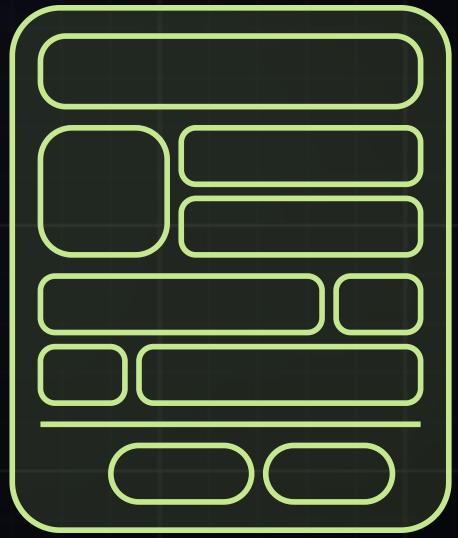
Состояние



Логика



Представление



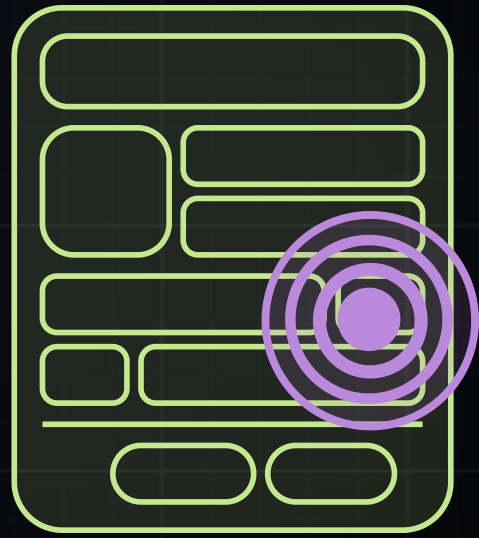
Состояние



Логика



Представление



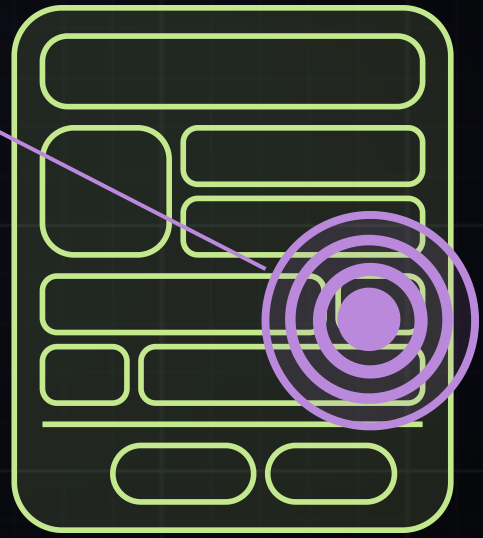
Состояние

Логика

Представление



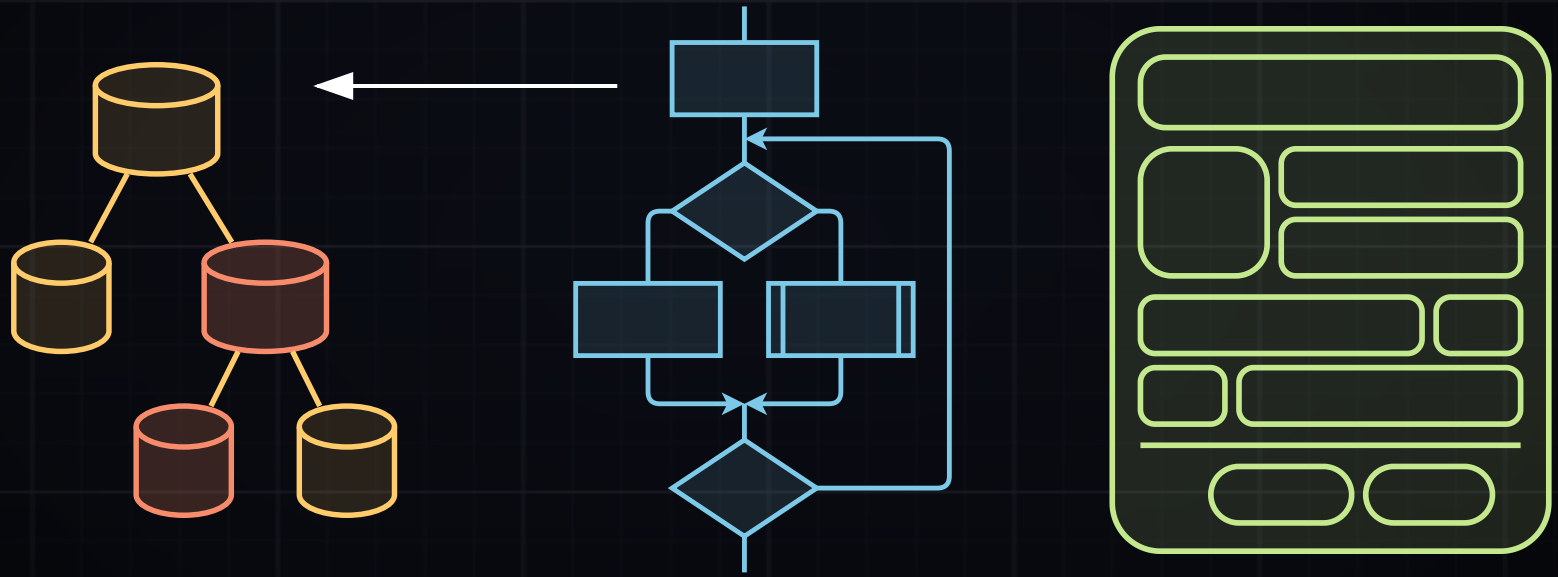




Состояние

Логика

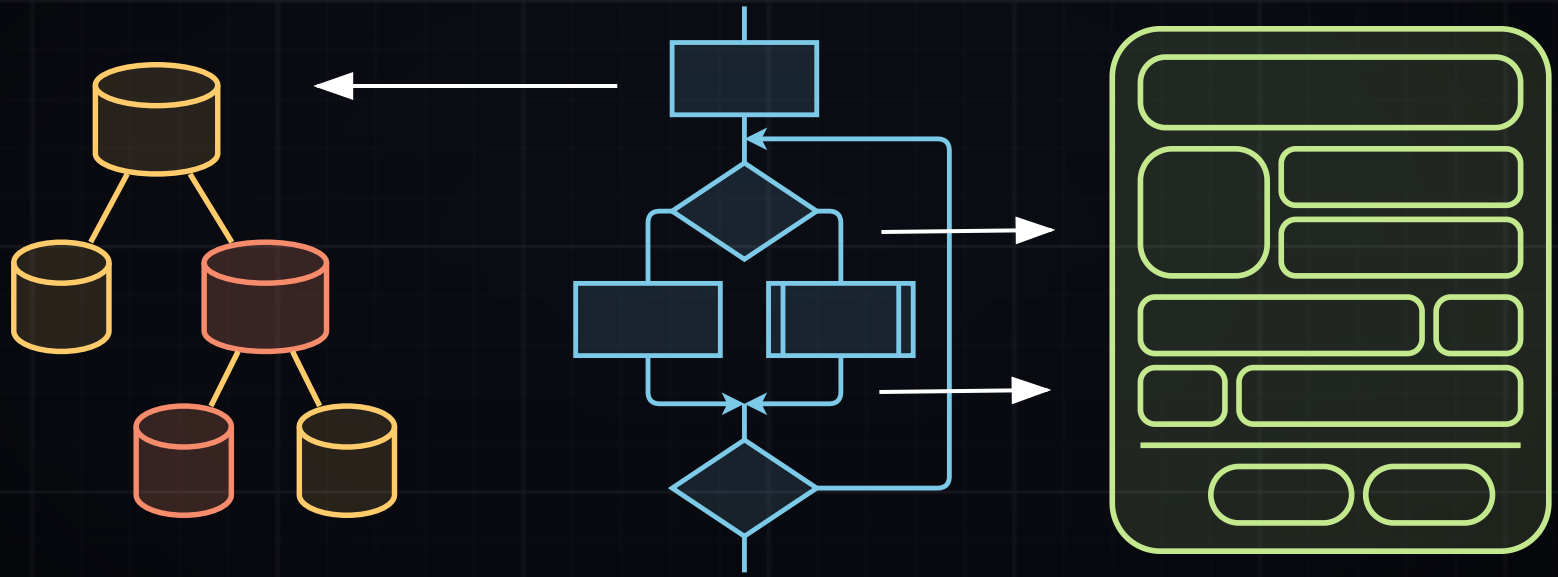
Представление



Состояние

Логика

Представление



Состояние

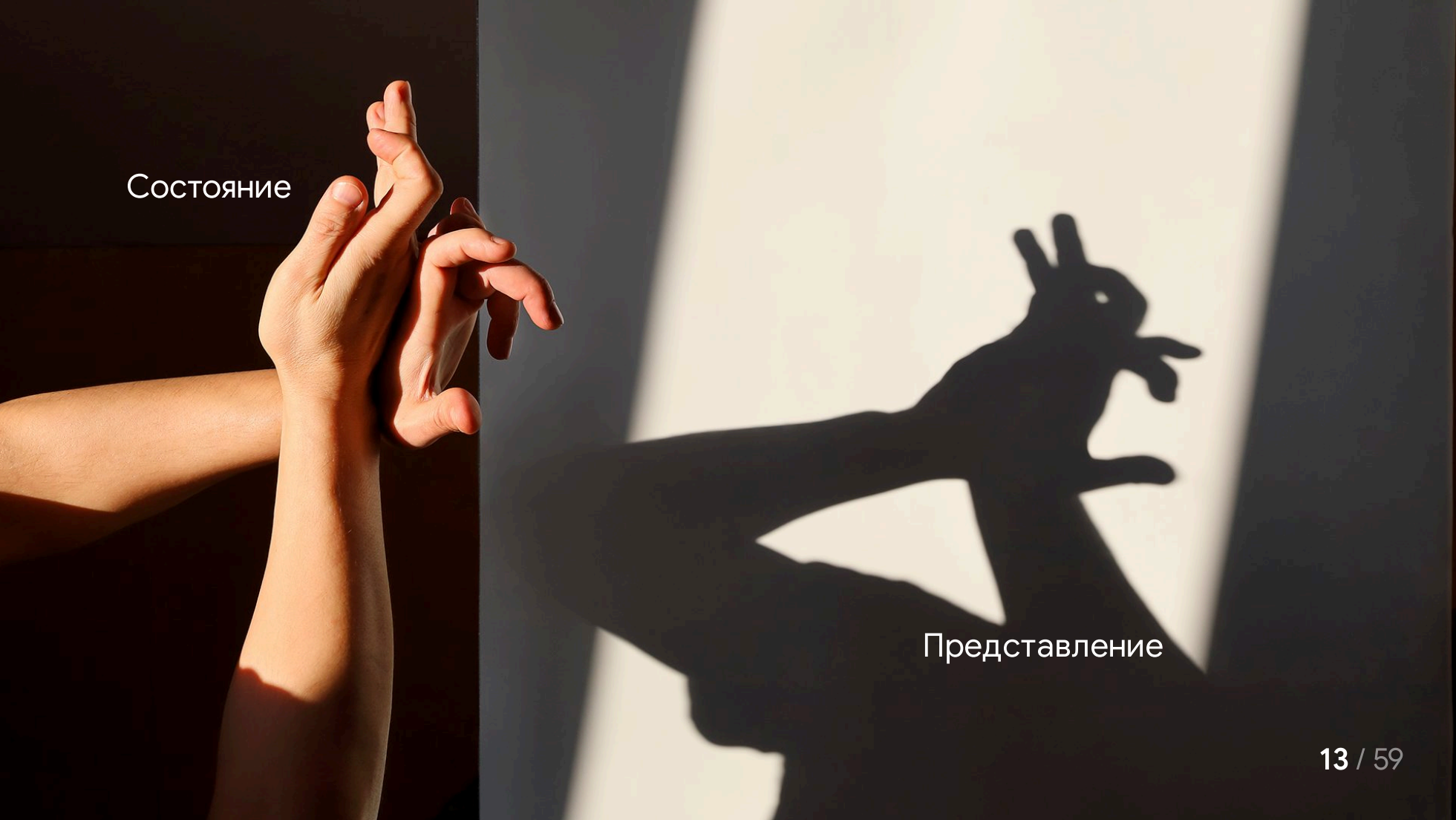
Логика

Представление

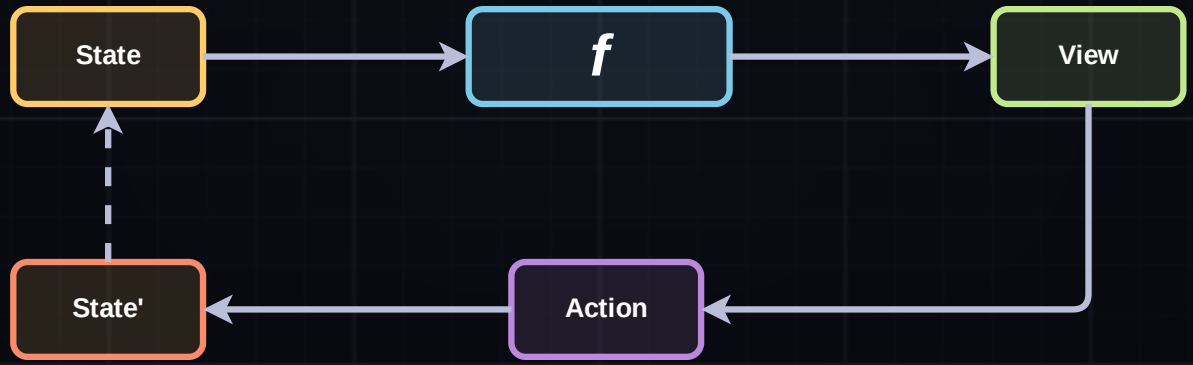




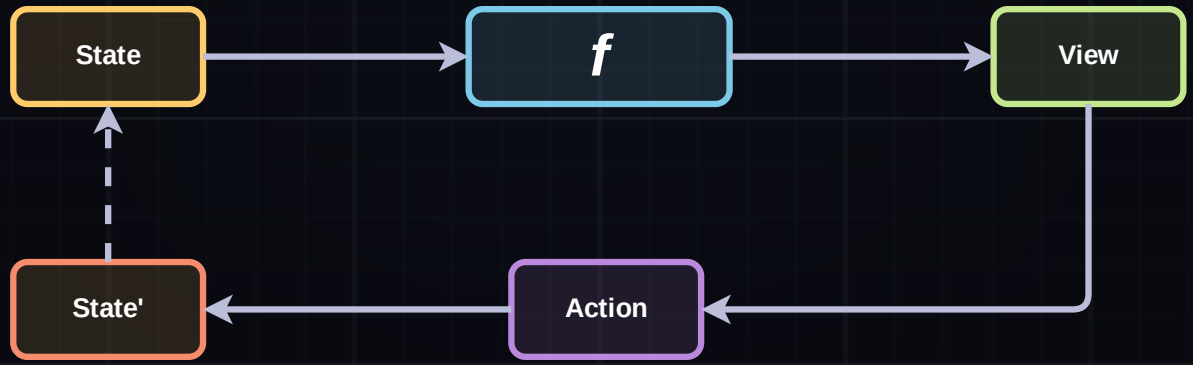



Состояние
Представление

Чистые функции



Чистые функции

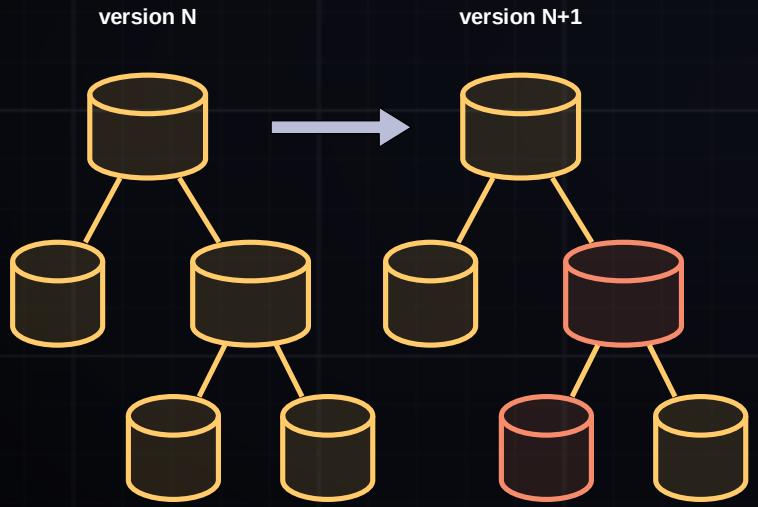


где *f* - чистая функция без побочных эффектов и внутреннего состояния

Операции с данными

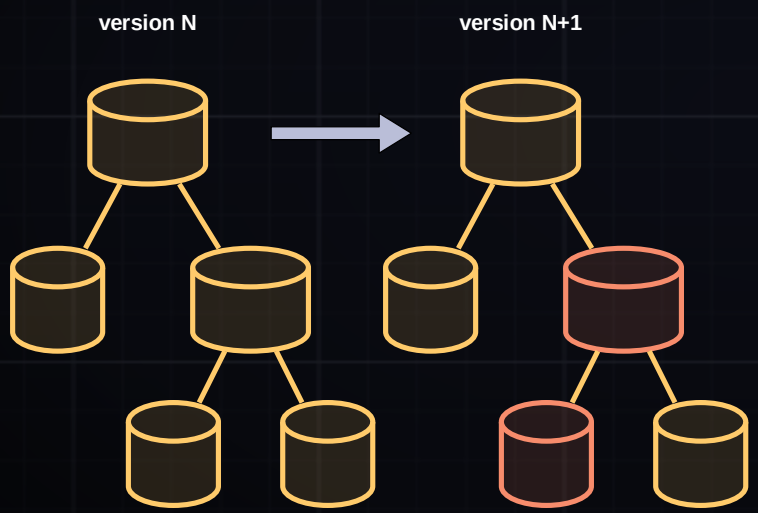
Операции с данными

Immutable data

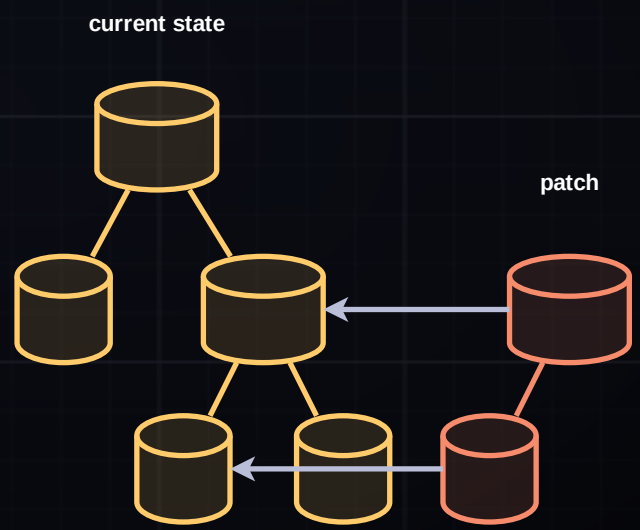


Операции с данными

Immutable data



Mutable data

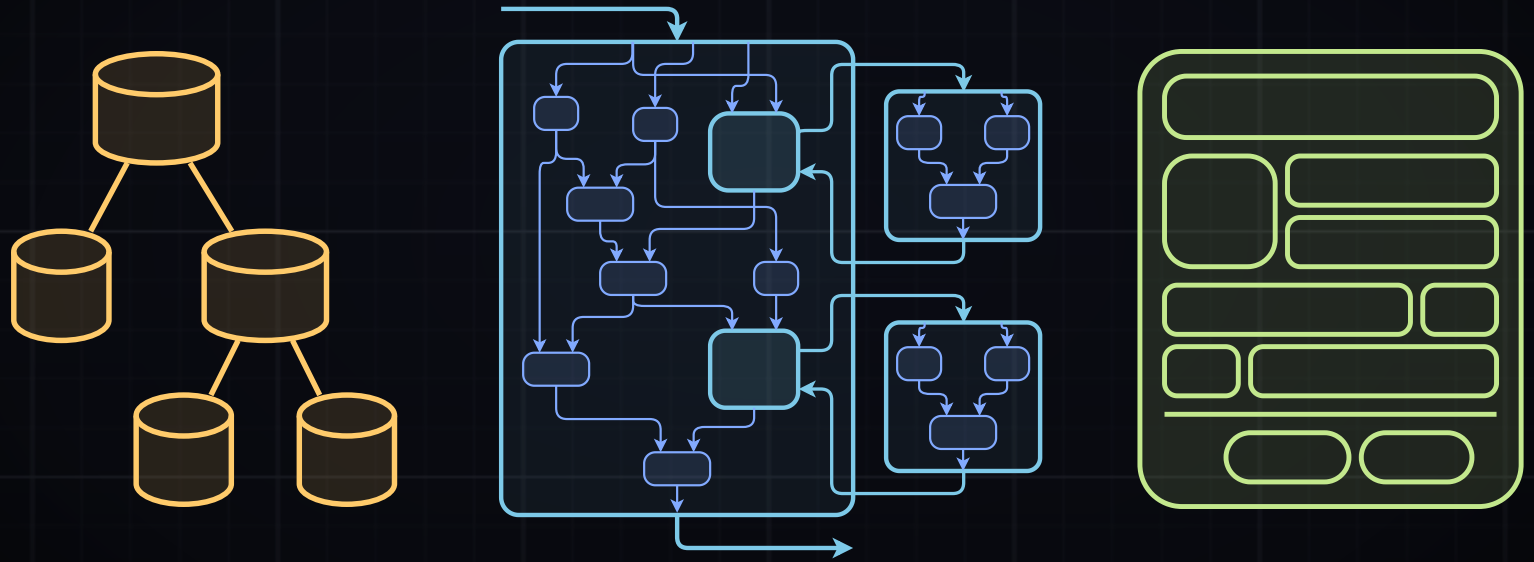


Инкрементальные вычисления

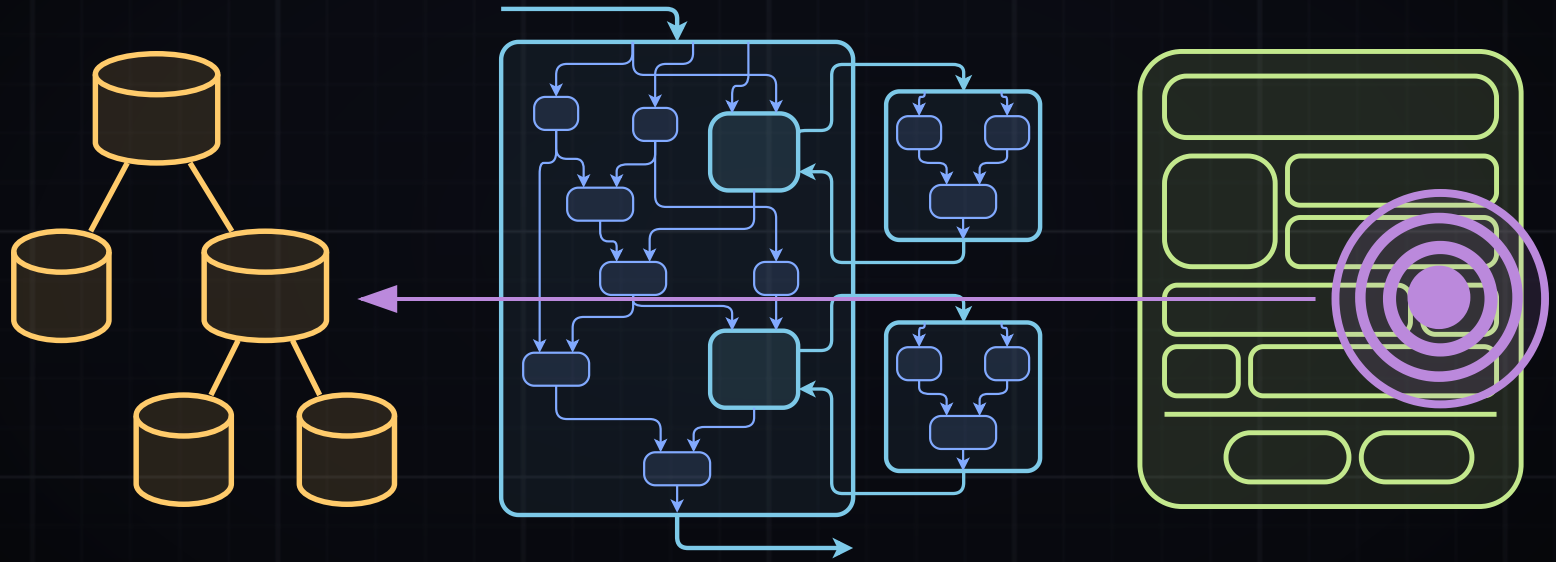
Инкрементальные вычисления



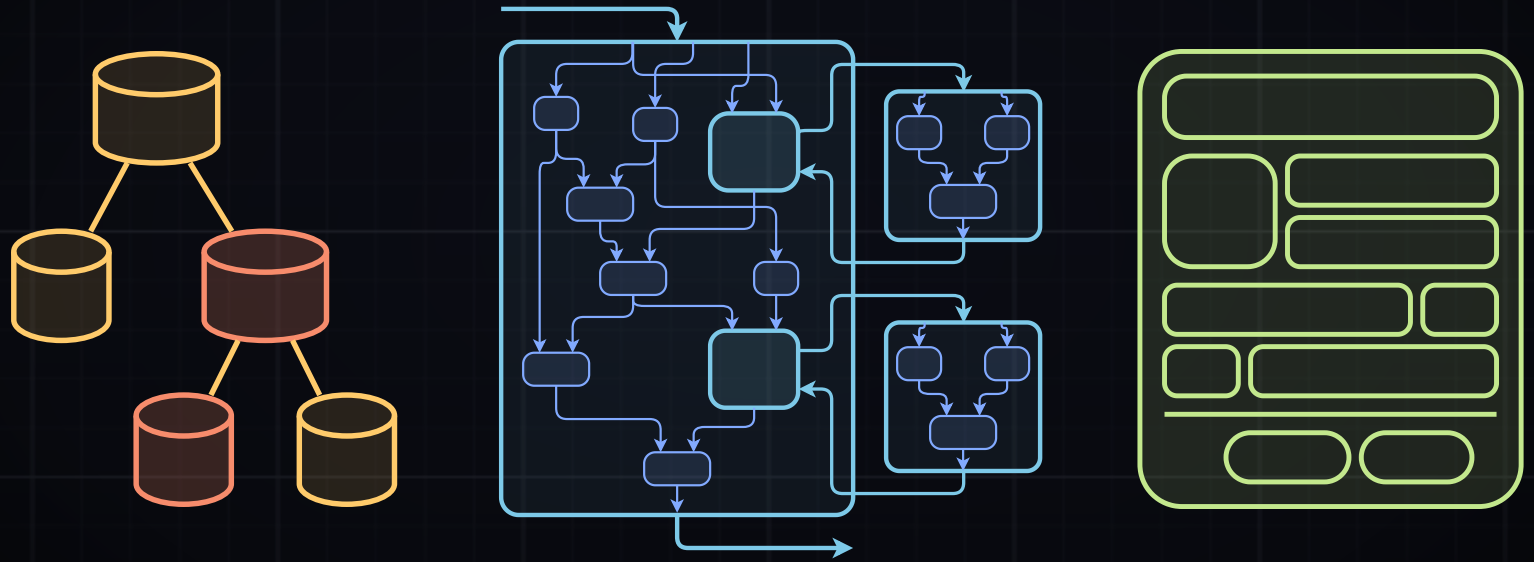
Инкрементальные вычисления



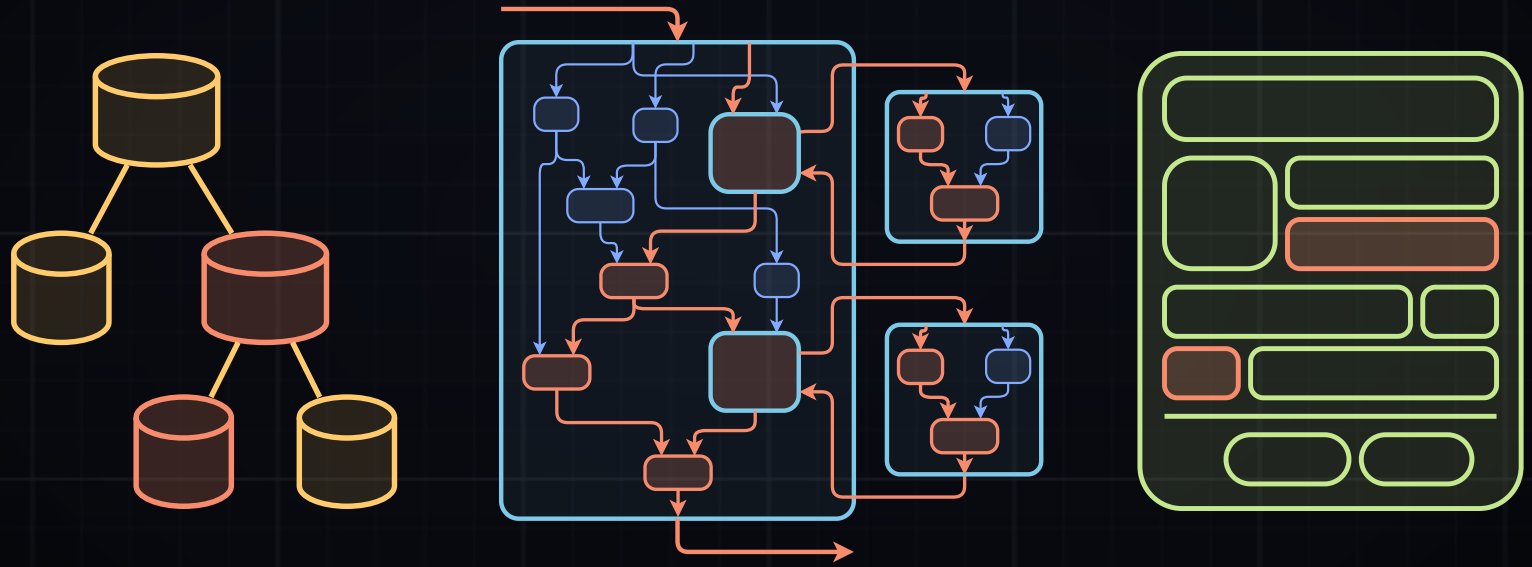
Инкрементальные вычисления



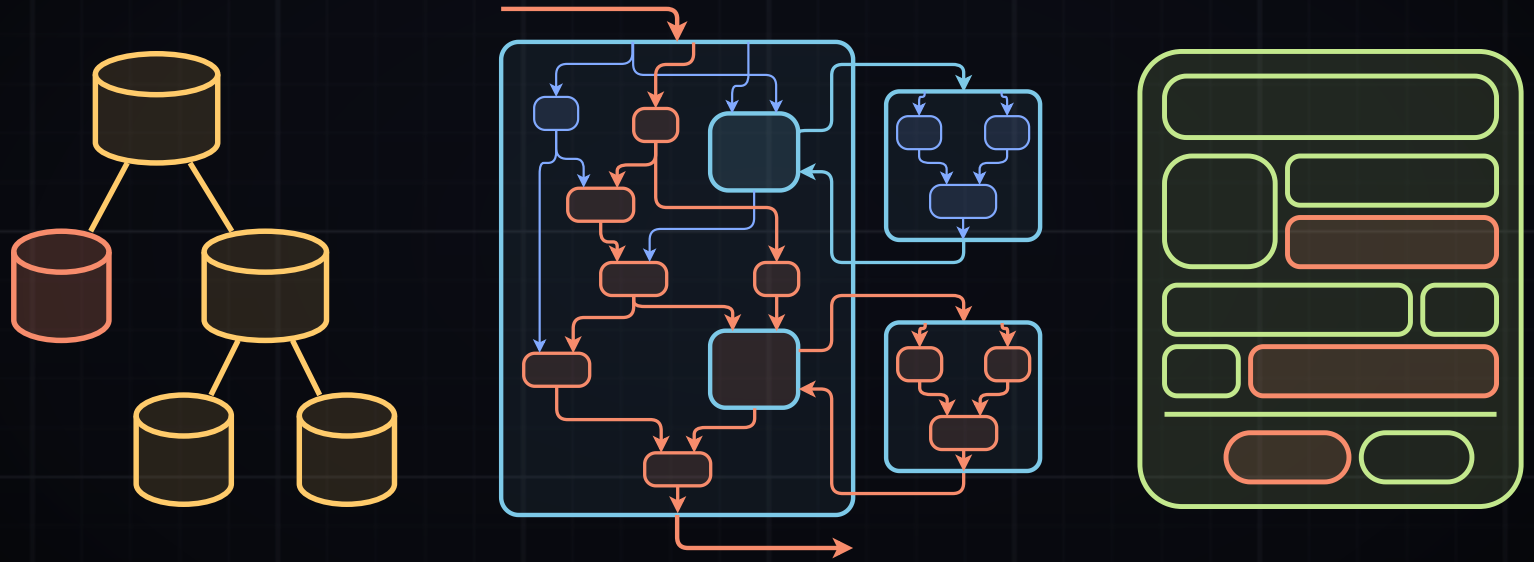
Инкрементальные вычисления



Инкрементальные вычисления



Инкрементальные вычисления



```
let U = 5

let R = 1

const I = U / R   // ⟶ 5

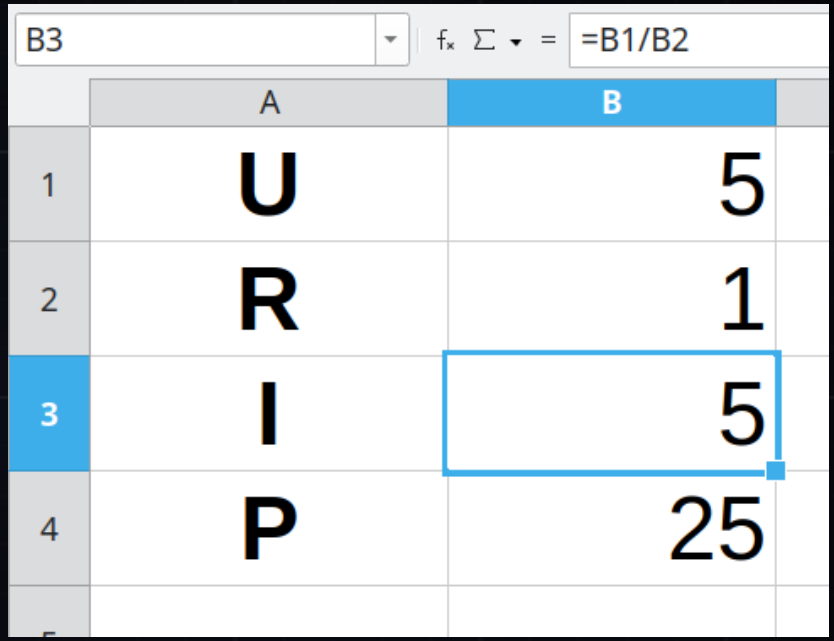
const P = I * U   // ⟶ 25
```

```
let U = 5

let R = 1

const I = U / R   // ⟶ 5

const P = I * U   // ⟶ 25
```

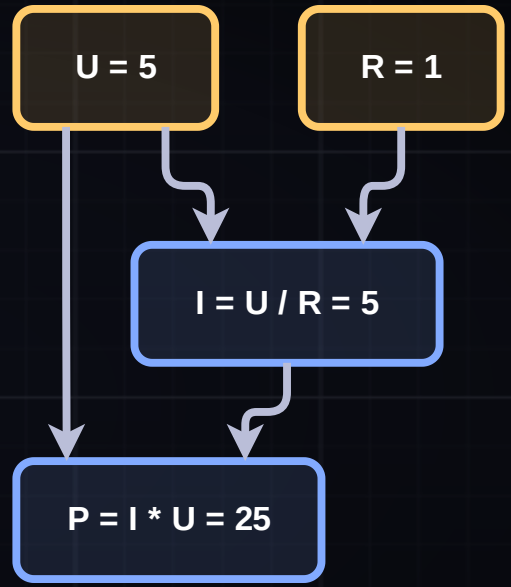


```
let U = 5

let R = 1

const I = U / R   // ⟶ 5

const P = I * U   // ⟶ 25
```

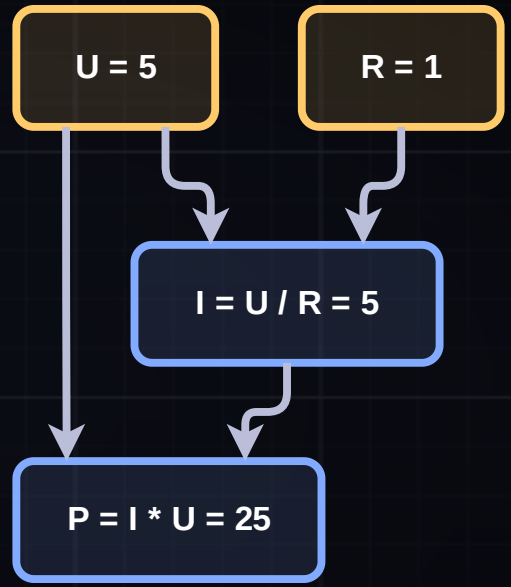


```
let U = 5

let R = 1

const I = U / R   // ⟶ 5

const P = I * U   // ⟶ 25
```

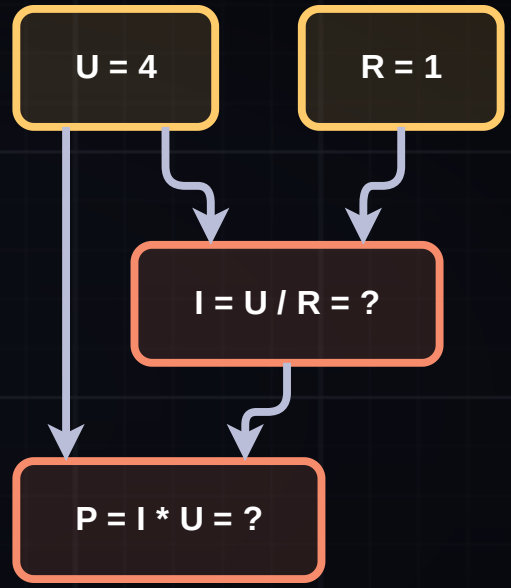


```
let U = 4

let R = 1

const I = U / R   // ⟶ ?

const P = I * U   // ⟶ ?
```

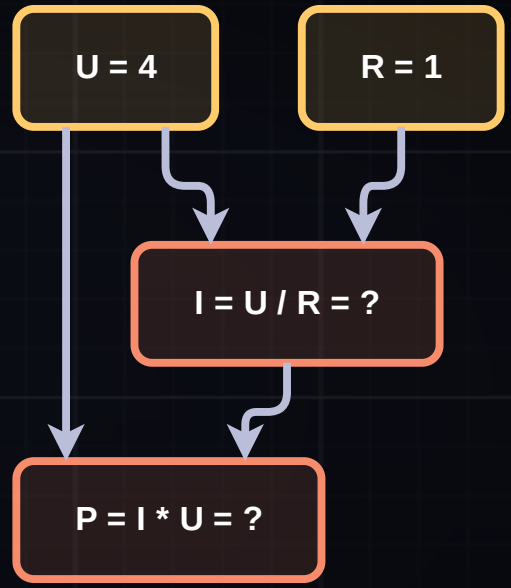


```
let U = 4

let R = 1

const I = U / R   // ⟶ ?

const P = I * U   // ⟶ ?
```

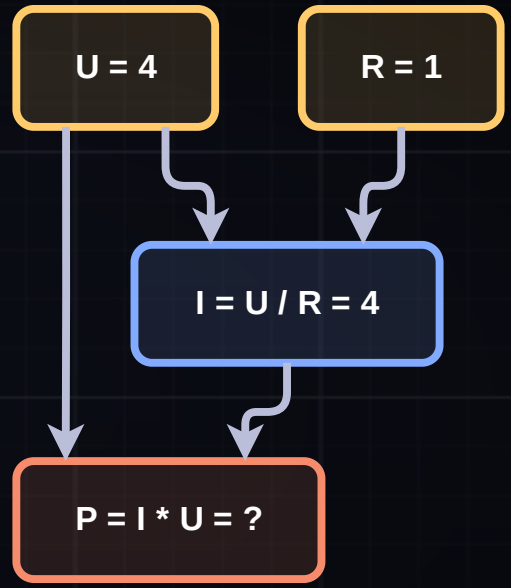


```
let U = 4

let R = 1

const I = U / R   // ⟶ 4

const P = I * U   // ⟶ ?
```

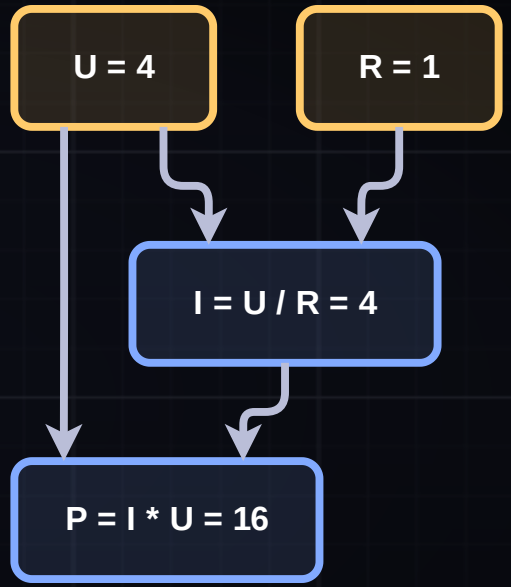


```
let U = 4

let R = 1

const I = U / R   // ⟶ 4

const P = I * U   // ⟶ 16
```

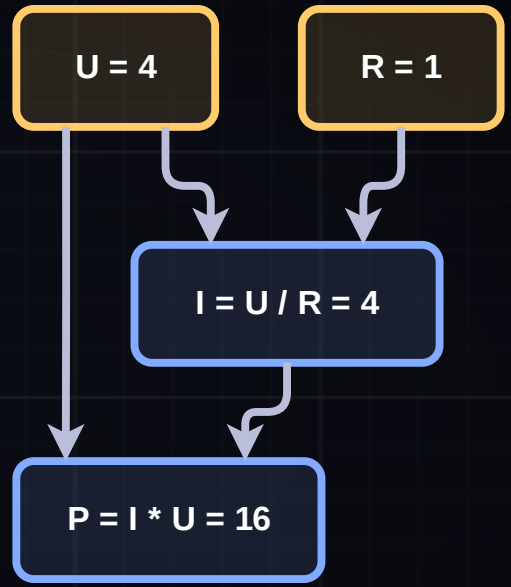


```
let U = 4

let R = 1

const I = U / R   // ⟶ 4

const P = I * U   // ⟶ 16
```

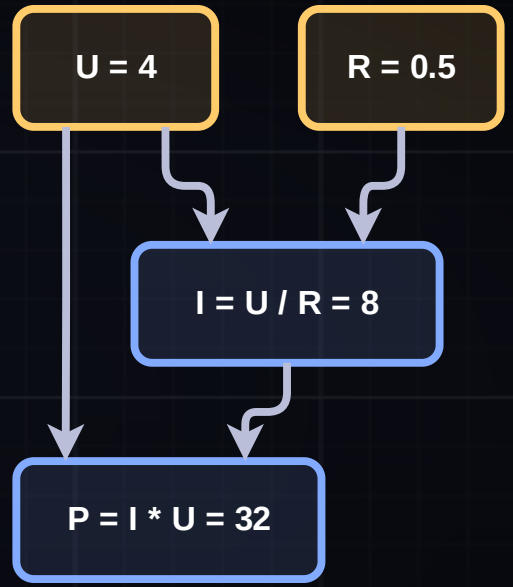


```
let U = 4

let R = 0.5

const I = U / R   // ⟶ 8

const P = I * U   // ⟶ 32
```

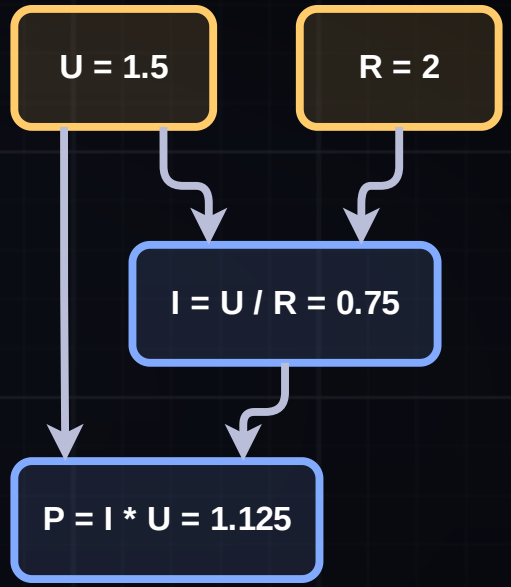


```
let U = 1.5

let R = 2

const I = U / R   // ⟶ 0.75

const P = I * U   // ⟶ 1.125
```



```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

  const amperage = voltage / resistance  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} on:input={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} on:input={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```

```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

  const amperage = voltage / resistance  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} on:input={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} on:input={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```

```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

  const amperage = voltage / resistance  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} on:input={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} on:input={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```

```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

  const amperage = voltage / resistance  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} on:input={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} on:input={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```

```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

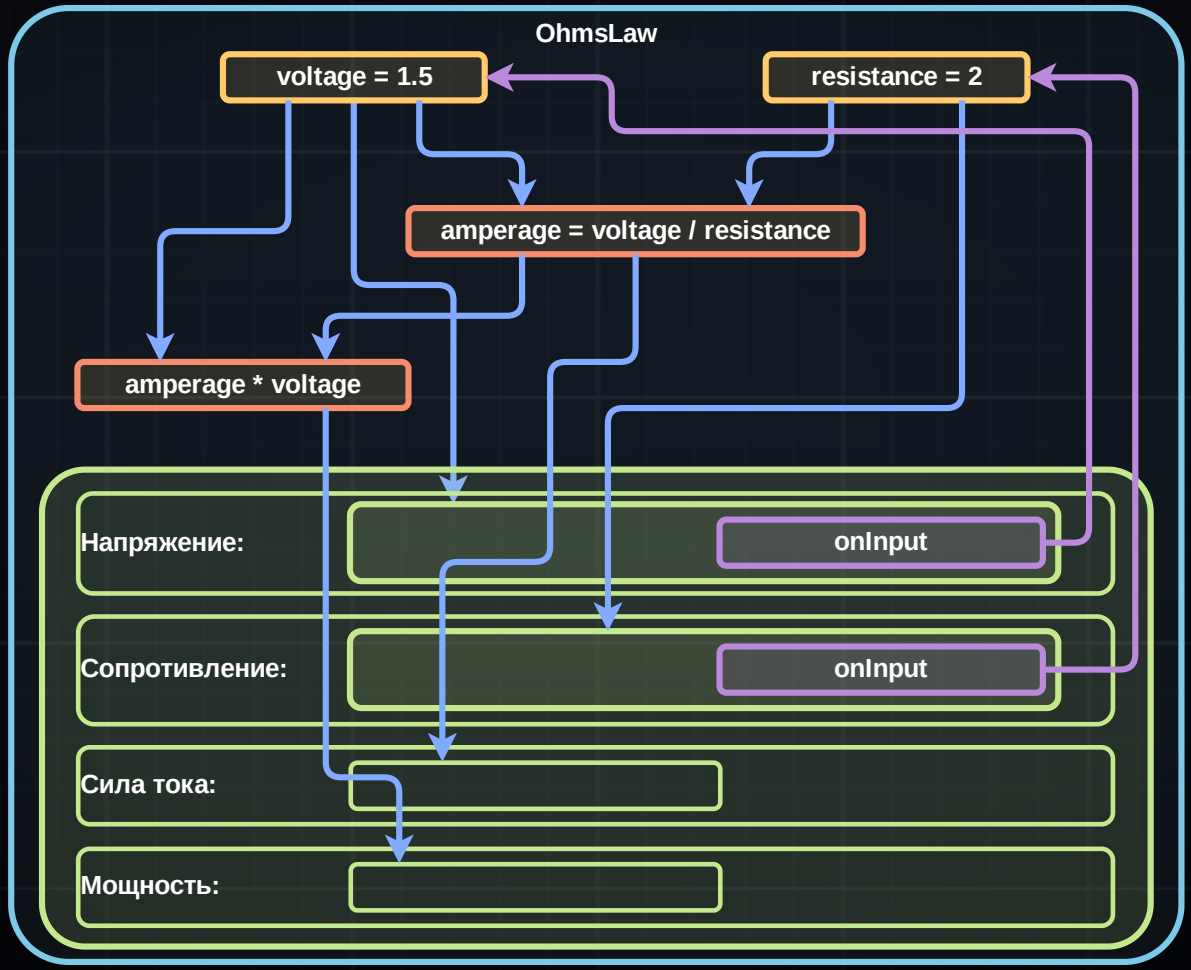
  const amperage = voltage / resistance  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} on:input={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} on:input={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```

```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

  const amperage = voltage / resistance  // I

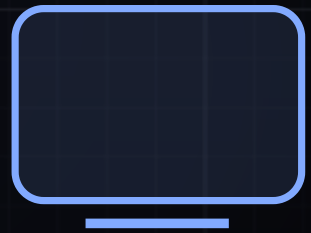
  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} on:input={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} on:input={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```


OhmsLaw
voltage = 1.5
resistance = 2
amperage = voltage / resistance
amperage * voltage
Напряжение:
onInput
Сопротивление:
onInput
Сила тока:
Мощность:

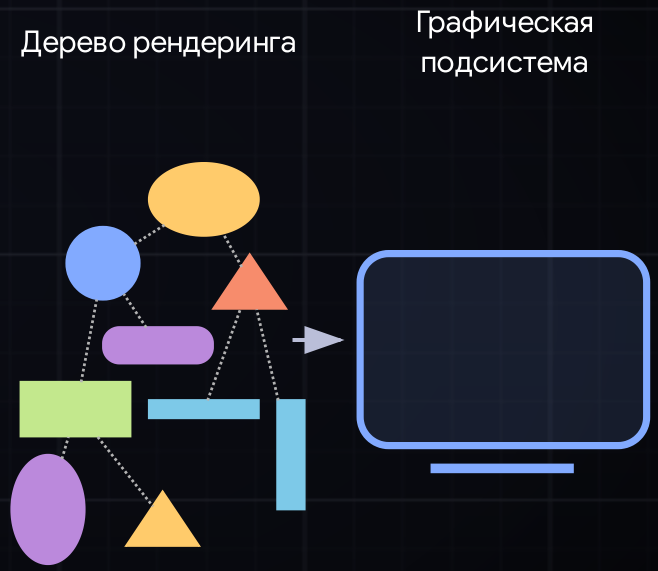
Представление

Представление

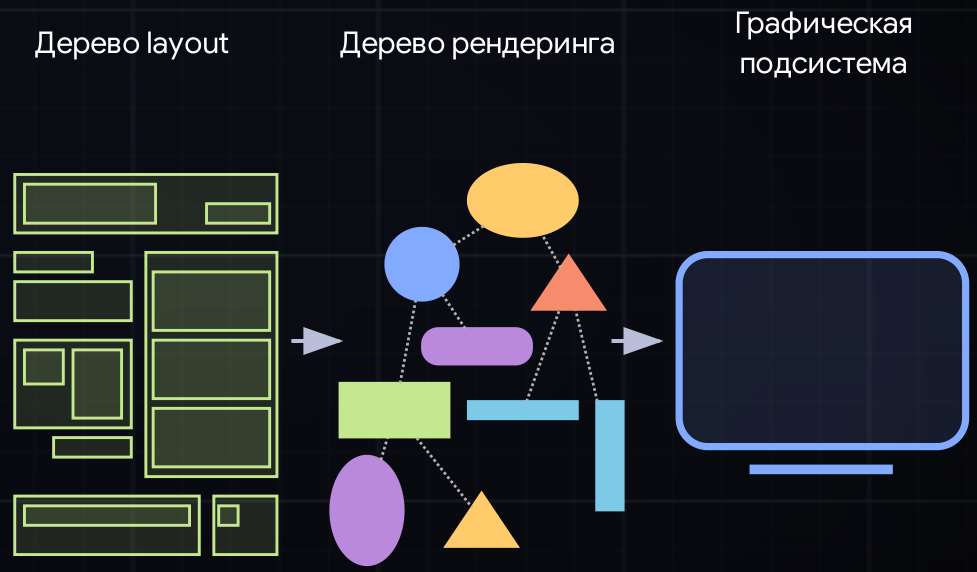
Графическая
подсистема



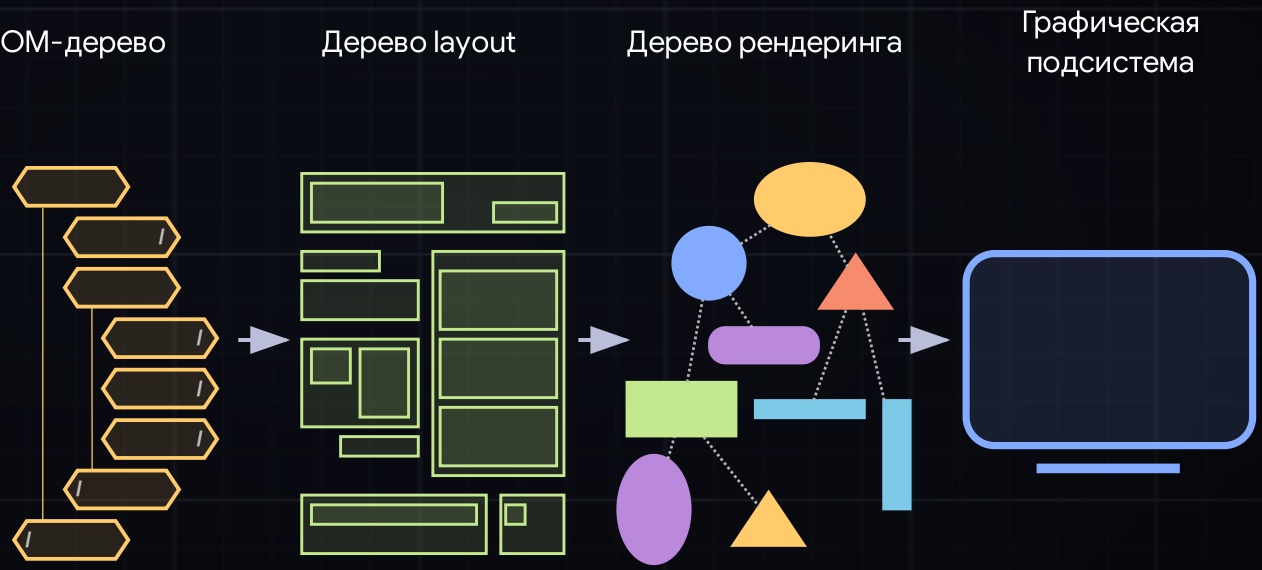
Представление



Представление



Представление



Представление

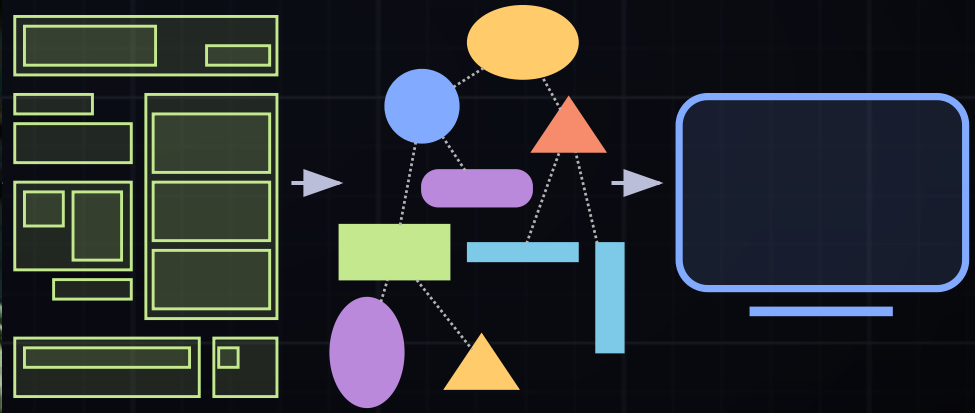
DOM-дерево



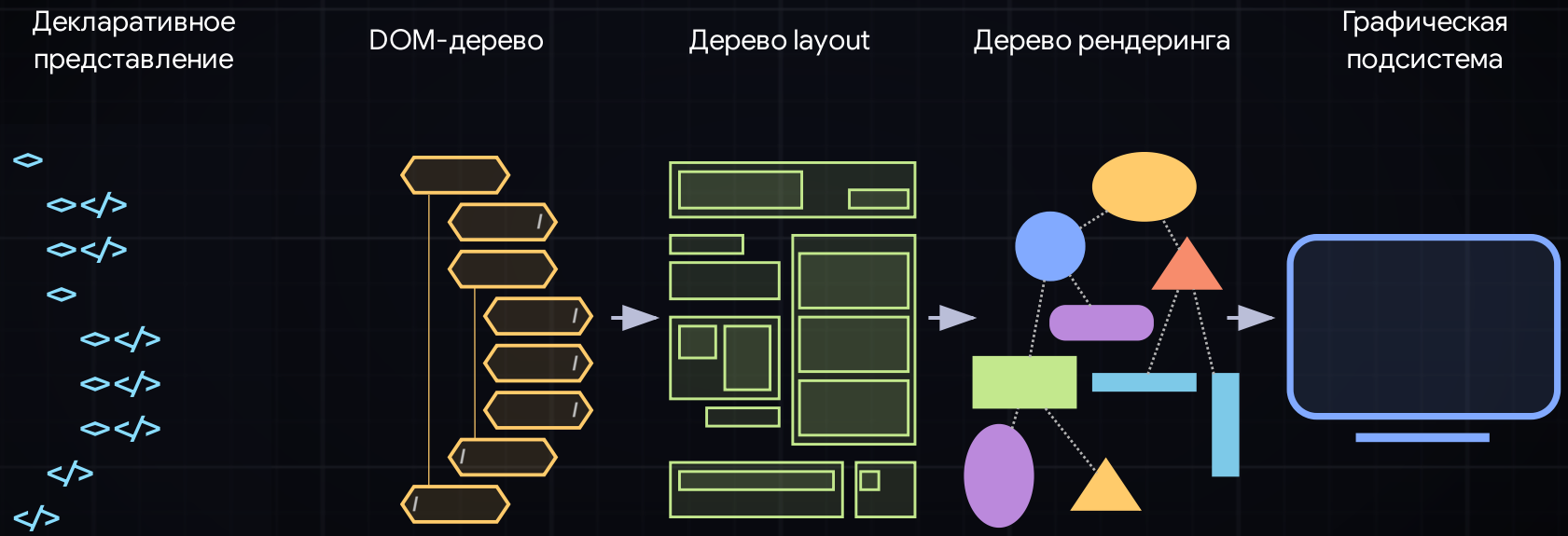
Дерево layout

Дерево рендеринга

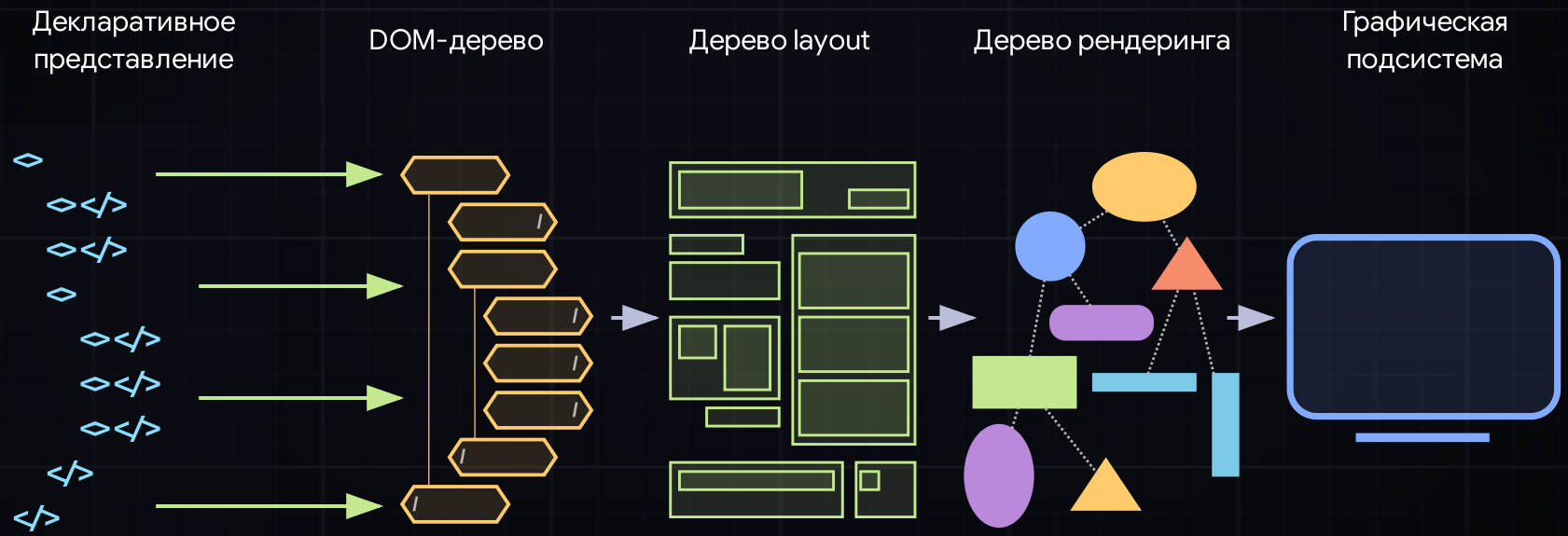
Графическая подсистема



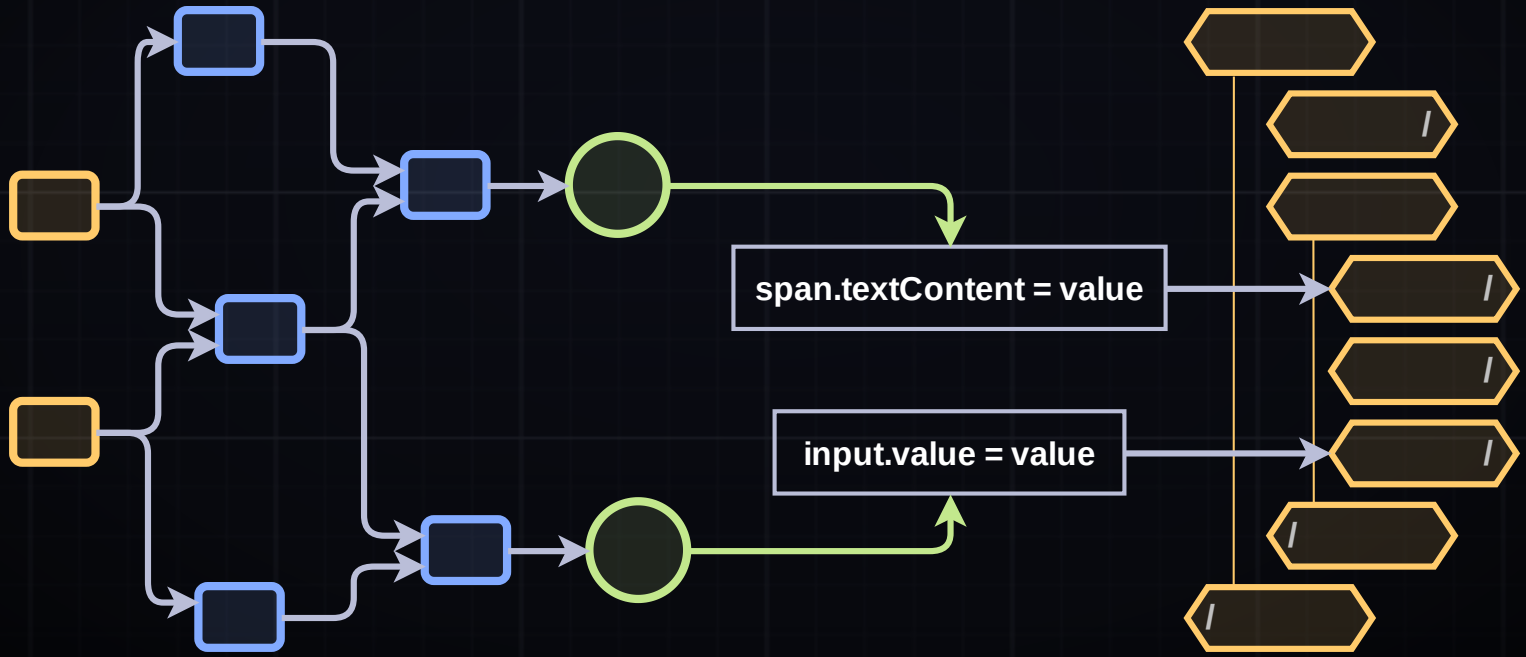
Представление



Представление



Потребители данных (эффекты)



Инкрементальные вычисления

- Минимизация промежуточных вычислений
- Точечное обновление UI
- Код остаётся декларативным

Декларативный TSX

Декларативное программирование

Декларативное программирование

- Код описывает ожидаемый результат, а не способ его получения
  - Код описывает структуру UI, потоки данных, логику приложения в виде **деклараций**, а не последовательности действий

Декларативное программирование

- Код описывает ожидаемый результат, а не способ его получения
  - Код описывает структуру UI, потоки данных, логику приложения в виде **деклараций**, а не последовательности действий
- Domain Specific Language (DSL) вместо универсального JavaScript

Декларативное программирование

- Код описывает ожидаемый результат, а не способ его получения
  - Код описывает структуру UI, потоки данных, логику приложения в виде **деклараций**, а не последовательности действий
- Domain Specific Language (DSL) вместо универсального JavaScript
- Строгие абстракции:
  - Программист сосредоточен на семантике DSL
  - Детали реализации берёт на себя фреймворк

Декларативность в UI-фреймворках

Vue

```vue
<script setup>
import { ref } from 'vue'

const show = ref(true)
const list = ref([1, 2, 3])
</script>

<template>
  <button @click="show = !show">Toggle List</button>
  <button @click="list.push(list.length + 1)">Push</button>
  <button @click="list.pop()">Pop</button>
  <button @click="list.reverse()">Reverse List</button>

  <ul v-if="show && list.length">
    <li v-for="item of list">{{ item }}</li>
  </ul>
  <p v-else-if="list.length">List is not empty, but hidden.</p>
  <p v-else>List is empty.</p>
</template>
```

Svelte

```svelte
<script>
  import Thing from './Thing.svelte';

  let things = $state([
    { id: 1, color: 'darkblue' },
    { id: 2, color: 'indigo' }
  ]);
  function handleClick() {
    things = things.slice(1);
  }
</script>

<button onclick={handleClick}> Remove first thing </button>
<div>
  <div>
    {#each things as thing (thing.id)}
      <Thing current={thing.color} />
    {/each}
  </div>
</div>
```

Декларативный TSX

Декларативный TSX

- Подмножество TypeScript с семантикой инкрементальных вычислений

Декларативный TSX

- Подмножество TypeScript с семантикой инкрементальных вычислений
- Используется как "язык шаблонов" для UI-компонентов

Декларативный TSX

- Подмножество TypeScript с семантикой инкрементальных вычислений
- Используется как "язык шаблонов" для UI-компонентов
- Поддерживает простую логику состояния, трансформации и привязки данных

Декларативный TSX

- Подмножество TypeScript с семантикой инкрементальных вычислений
- Используется как "язык шаблонов" для UI-компонентов
- Поддерживает простую логику состояния, трансформации и привязки данных
- Сложная логика выносится за пределы языка в обычный TypeScript

Декларативный TSX

- Подмножество TypeScript с семантикой инкрементальных вычислений
- Используется как "язык шаблонов" для UI-компонентов
- Поддерживает простую логику состояния, трансформации и привязки данных
- Сложная логика выносится за пределы языка в обычный TypeScript
- Взаимодействие с обычным TypeScript - через "границу" интеграции

	Декларативный TSX	Языки шаблонов
Типизация	Уже типизирован TypeScript-ом	Нужно интегрировать TypeScript (или писать свой typechecker?)
Поддержка IDE	Уже есть	Нужны дополнительные расшинения, начиная с синтаксиса
Гибкость языка	Lexical scopes, замыкания, функции высшего порядка	В разметке свои правила видимости имён, замыканий, передачи аргументов функций и т.п.

"Fine-grained reactivity"

Разновидности реактивности

Разновидности реактивности

Ориентированные на данные

- первичные и производные данные
- граф зависимостей



Разновидности реактивности

Ориентированные на данные

- первичные и производные данные
- граф зависимостей

 MobX  Vue  Solid

Ориентированные на события

- события и потоки данных
- трансформация, фильтрация, буферизация событий и т.п.

 Rx.js

Разновидности реактивности

Ориентированные на данные

- первичные и производные данные
- граф зависимостей

MobX Vue Solid

Ориентированные на события

- события и потоки данных
- трансформация, фильтрация, буферизация событий и т.п.

Rx.js

Гибрид первых двух

- подробный граф из событий, данных и действий с ними

Effector

Разновидности реактивности

Ориентированные на данные

- первичные и производные данные
- граф зависимостей

MobX Vue Solid

Ориентированные на события

- события и потоки данных
- трансформация, фильтрация, буферизация событий и т.п.

Rx.js

Гибрид первых двух

- подробный граф из событий, данных и действий с ними

Effector

Основанные на перевычислении

- rerender частей приложения
- reconcilliation результатов

React Preact

"Гранулярность" реактивности



"Гранулярность" реактивности

Пересчитать всё

"Coarse-grained reactivity"



"Fine-grained reactivity"



Пересчитать инкрементально

"Гранулярность" реактивности

Пересчитать всё	"Coarse-grained reactivity"	"Fine-grained reactivity"	Пересчитать инкрементально
			?

Сигналы (API)

```ts
type Signal<T> = {
  get: () => T
  set: (value: T) => void
}

type Computed<T> = {
  get: () => T
}

const signal = <T>(initValue: T): Signal<T> => { /* ... */ }
const computed = <T>(func: () => T): Computed<T> => { /* ... */ }
```

Декларативный TSX

```
const OhmsLaw = () => {
  let voltage = 1.5   // U
  let resistance = 2  // R

  const amperage = voltage / resistance  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} onInput={e => {
        voltage = e.target.valueAsNumber
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} onInput={e => {
        resistance = e.target.valueAsNumber
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> А</p>
      <p>Мощность: <span>{amperage * voltage}</span> Вт</p>
    </div>
  )
}
```

```
const OhmsLaw = () => {
  const voltage = signal(1.5)   // U
  const resistance = signal(2)  // R

  const amperage = computed(() => voltage.get() / resistance.get())  // I

  return (
    <div>
      <p>Напряжение: <input type="number" value={voltage} onInput={e => {
        voltage.set(e.target.valueAsNumber)
      }}/></p>
      <p>Сопротивление: <input type="number" value={resistance} onInput={e => {
        resistance.set(e.target.valueAsNumber)
      }}/></p>
      <p>Сила тока: <span>{amperage}</span> A</p>
      <p>Мощность: <span>{computed(() => amperage.get() * voltage.get())}</span> Вт</p>
    </div>
  )
}
```

Преобразования кода

```
-let state = initExpression
+const state = signal(currentOf(initExpression))

-const result = computationExpression(state)
+const result = computed(() => computationExpression(currentOf(state)))
```

Преобразования кода

```
-let state = initExpression
+const state = signal(currentOf(initExpression))

-const result = computationExpression(state)
+const result = computed(() => computationExpression(currentOf(state)))
```

где:

```
const currentOf = <T>(s: Computed<T> | T): T => {
  return s instanceof Computed ? s.get() : s
}
```

```
let lightswitch = true

let voltage = 5
let resistance1 = 1
let resistance2 = 1

const power = lightswitch ? voltage * voltage / (resistance1 + resistance2) : 0
```

Декларативный TSX

```
let lightswitch = true

let voltage = 5
let resistance1 = 1
let resistance2 = 1

const power = lightswitch ? voltage * voltage / (resistance1 + resistance2) : 0
```

```
const lightswitch = signal(true)

const voltage = signal(5)
const resistance1 = signal(1)
const resistance2 = signal(1)

const power = computed(() ⇒
  lightswitch.get()
    ? (voltage.get() * voltage.get()) / (resistance1.get() + resistance2.get())
    : 0)
```

```
const lightswitch = signal(true)

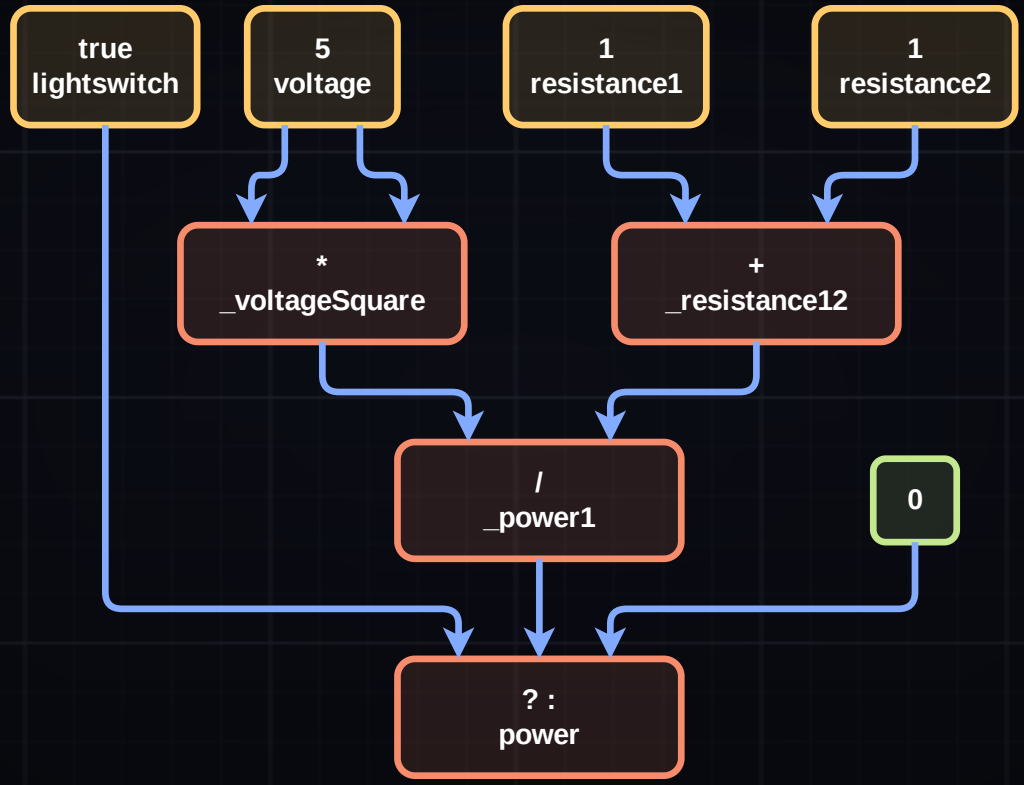
const voltage = signal(5)
const resistance1 = signal(1)
const resistance2 = signal(1)

const _voltageSquare = computed(() => voltage.get() * voltage.get())
const _resistance12 = computed(() => resistance1.get() + resistance2.get())
const power = computed(() =>
  lightswitch.get()
    ? _voltageSquare.get() / _resistance12.get()
    : 0)
```

```
const lightswitch = signal(true)

const voltage = signal(5)
const resistance1 = signal(1)
const resistance2 = signal(1)

const _voltageSquare = computed(() => voltage.get() * voltage.get())
const _resistance12 = computed(() => resistance1.get() + resistance2.get())
const _power1 = computed(() => _voltageSquare.get() / _resistance12.get())
const power = computed(() => lightswitch.get() ? _power1.get() : 0)
```


true
lightswitch
5
voltage
1
resistance1
1
resistance2
*
_voltageSquare
+
_resistance12
/
_power1
0
? :
power

Сигналы: Потеря реактивности

```
const amperage = signal(2)
const resistance = signal(1)

const power = computed(() ⇒ amperage.get() * amperage.get() * resistance.get())
```

Сигналы: Потеря реактивности

```
const amperage = signal(2)
const resistance = signal(1)

const voltage = amperage.get() * resistance.get()  // ⟶ 2

const power = computed(() ⇒ amperage.get() * voltage)  // ⟶ 4
```

Сигналы: Потеря реактивности

```
const amperage = signal(2)
const resistance = signal(1)

const voltage = amperage.get() * resistance.get()  // ⟶ 2

const power = computed(() ⇒ amperage.get() * voltage)  // ⟶ 4

resistance.set(2)

console.log(power.get())  // 4
```

Сигналы: Потеря реактивности

```
const amperage = signal(2)
const resistance = signal(1)

const voltage = computed(() ⇒ amperage.get() * resistance.get())  // ⟶ 2

const power = computed(() ⇒ amperage.get() * voltage.get())  // ⟶ 4

resistance.set(2)

console.log(power.get())  // 8
```

Сигналы: Потеря реактивности

```
const amperage = signal(2)
const resistance = signal(1)

const voltage = computed(() ⇒ amperage.get() * resistance.get())  // ⟶ 2

const power = computed(() ⇒ amperage.get() * voltage.get())  // ⟶ 4

resistance.set(2)

console.log(power.get())  // 8
```

Условный рендеринг

Декларативный TSX

```
const Counter = () => {
  let counter = 0

  return (
    <div>
      <button on:click={() => { counter++ }}>
        Увеличить счётчик
      </button>
      <p>{`Кнопка была нажата ${counter} раз!`}</p>
      <p>Это число{" "}
        <span>{counter % 2 === 0 ? <em>чётное</em> : <strong>нечётное</strong>}!</span>
      </p>
    </div>
  )
}
```

Условный рендеринг

```
const Counter = () => {
  let counter = 0

  return (
    <div>
      <button on:click={() => { counter++ }}>
        Увеличить счётчик
      </button>
      <p>{`Кнопка была нажата ${counter} раз!`}</p>
      <p>Это число{" "}
        <span>{counter % 2 === 0 ? <em>чётное</em> : <strong>нечётное</strong>}!</span>
      </p>
    </div>
  )
}
```

Условный рендеринг

```
const Counter = () => {
  const counter = signal(0)

  return (
    <div>
      <button on:click={() => { counter.set(counter.get() + 1) }}>
        Увеличить счётчик
      </button>
      <p>{`Кнопка была нажата ${counter.get()} раз!`}</p>
      <p>Это число <span>
        {reIf(computed(() => counter % 2 === 0), (
          <em>чётное</em>
        ), (
          <strong>нечётное</strong>
        ))}!</span>
      </p>
    </div>
  )
}
```

Декларативный TSX

```
const f = (a: string, b: number) => {
  const c = a.toUppercase()
  const d = g(b * 2, 42)
  const e = g(100500, b * 3)
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

```
const f = (a: string, b: number) => {
  const c = a.toUppercase()
  const d = g(b * 2, 42)
  const e = g(100500, b * 3)
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

```
const f = (a: string, b: number) => {
  const c = a.toUppercase()
  const d = g(b * 2, 42)
  const e = g(100500, b * 3)
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

```ts
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()
  const d = g(b * 2, 42)
  const e = g(100500, b * 3)
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

Декларативный TSX

```
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()  // c = "THE ANSWER"
  const d = g(b * 2, 42)
  const e = g(100500, b * 3)
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

```
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()  // c = "THE ANSWER"
  const d = g(b * 2, 42)  // g(84, 42) = ?
  const e = g(100500, b * 3)  // g(100500, 126) = ?
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

```
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()  // c = "THE ANSWER"
  const d = g(b * 2, 42)  // g(84, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 126) = ?
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {  // p = 84, p = 42
  const pp = p * p  // pp = 7056
  const qq = Math.sqrt(q)  // q = 6.48074069840786
  return pp + qq  // 7062.480740698408
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

```
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()  // c = "THE ANSWER"
  const d = g(b * 2, 42)  // g(84, 42) = 7062.4807406984 08
  const e = g(100500, b * 3)  // g(100500, 126) = 10100250011.224972
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {  // p = 100500, p = 126
  const pp = p * p  // pp = 10100250000
  const qq = Math.sqrt(q)  // q = 11.224972160321824
  return pp + qq  // 10100250011.224972
}

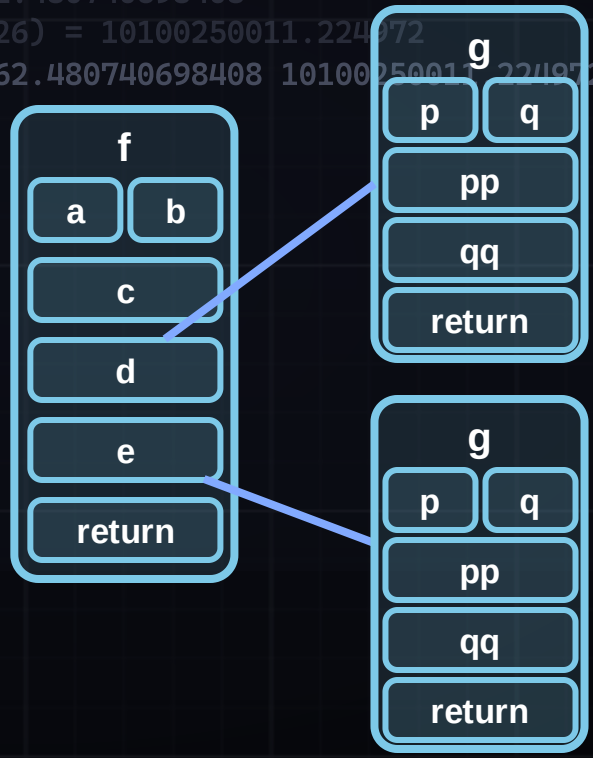
let a = "The Answer"
let b = 42
f(a, b)
```

```ts
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()  // c = "THE ANSWER"
  const d = g(b * 2, 42)  // g(84, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 126) = 10100250011.224972
  return `${c} ${d} ${e}`  // "THE ANSWER 7062.480740698408 10100250011.224972"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "The Answer"
let b = 42
f(a, b)
```

Декларативный TSX

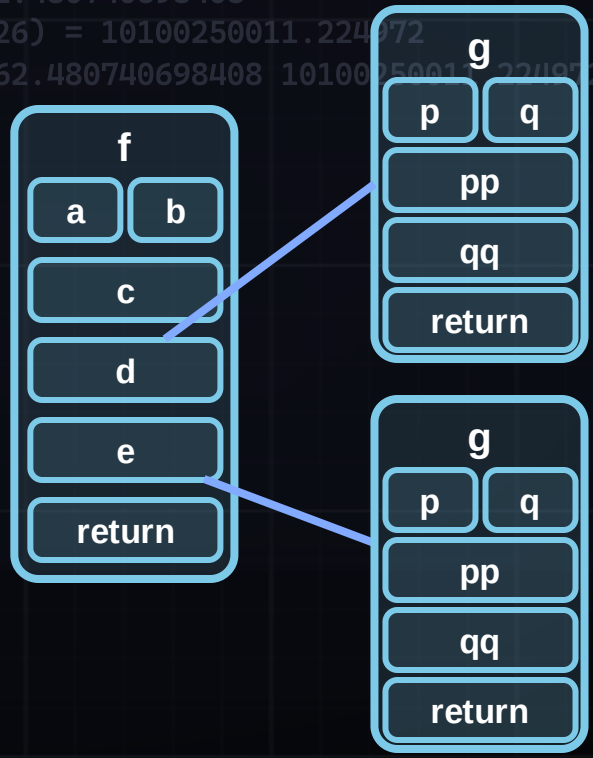


```ts
const f = (a: string, b: number) => {  // a = "The Answer", b = 42
  const c = a.toUppercase()  // c = "THE ANSWER"
  const d = g(b * 2, 42)  // g(84, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 126) = 10100250011.224072
  return `${c} ${d} ${e}`  // "THE ANSWER 7062.480740698408 10100250011.224072"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 42
f(a, b)
```

Декларативный TSX

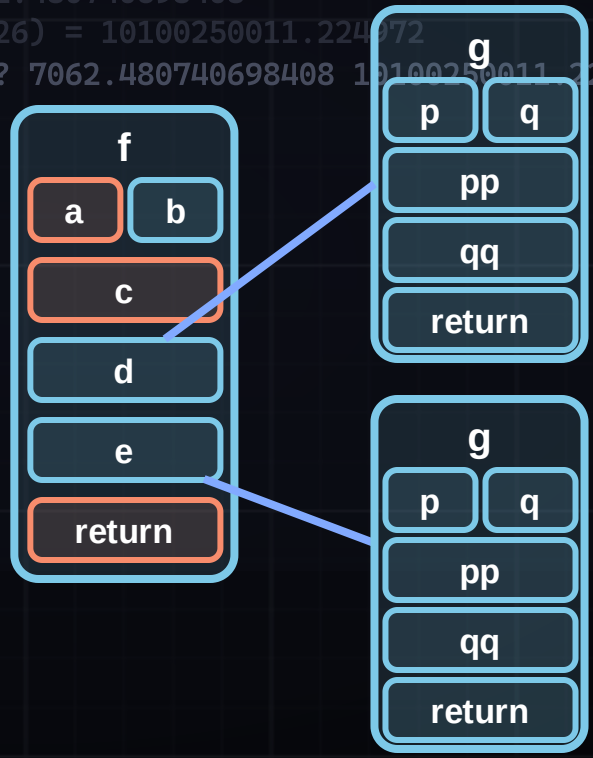


```ts
const f = (a: string, b: number) => {  // a = "Any questions?", b = 42
  const c = a.toUppercase()  // c = "ANY QUESTIONS?"
  const d = g(b * 2, 42)  // g(84, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 126) = 10100250011.224972
  return `${c} ${d} ${e}`  // "ANY QUESTIONS? 7062.480740698408 10100250011.224972"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 42
f(a, b)
```

Декларативный TSX

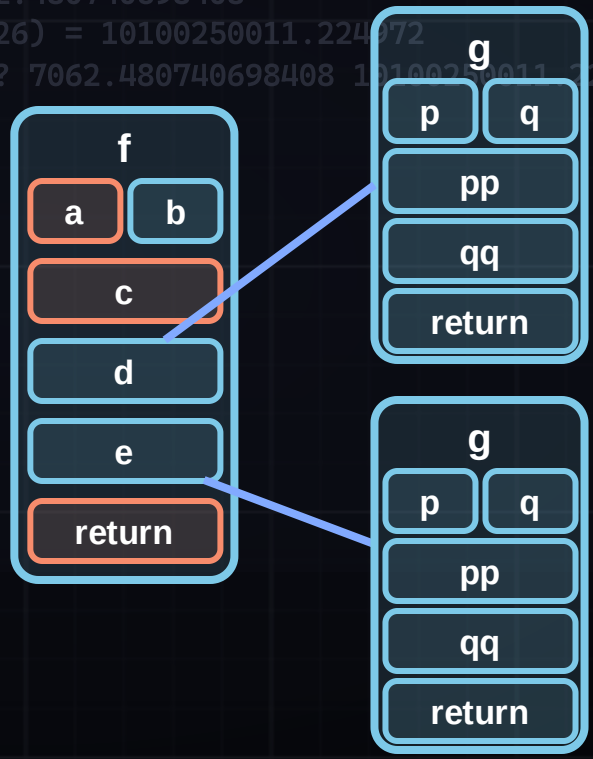


```ts
const f = (a: string, b: number) => {  // a = "Any questions?", b = 42
  const c = a.toUppercase()  // c = "ANY QUESTIONS?"
  const d = g(b * 2, 42)  // g(84, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 126) = 10100250011.224972
  return `${c} ${d} ${e}`  // "ANY QUESTIONS? 7062.480740698408 10100250011.224972"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 43
f(a, b)
```

Декларативный TSX

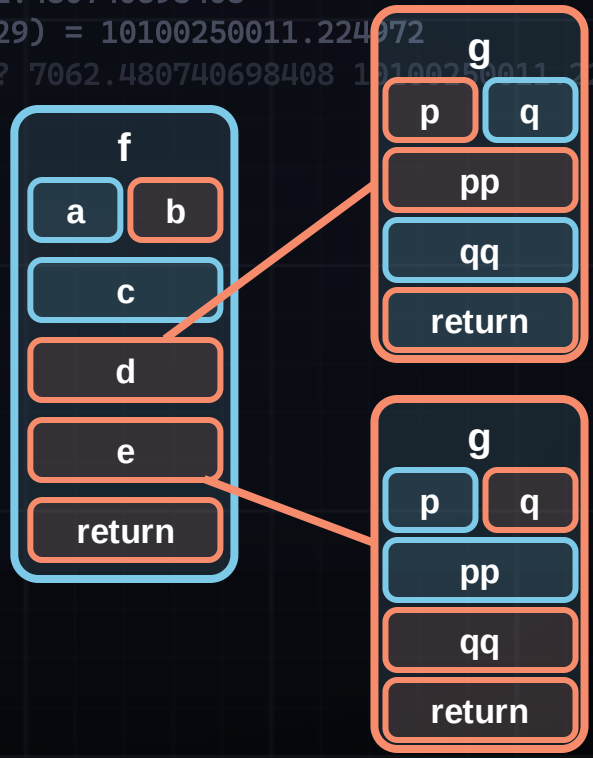


```ts
const f = (a: string, b: number) => {  // a = "Any questions?", b = 43
  const c = a.toUppercase()  // c = "ANY QUESTIONS?"
  const d = g(b * 2, 42)  // g(86, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 129) = 10100250011.224972
  return `${c} ${d} ${e}`  // "ANY QUESTIONS? 7062.480740698408 10100250011.224972"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 43
f(a, b)
```

Декларативный TSX

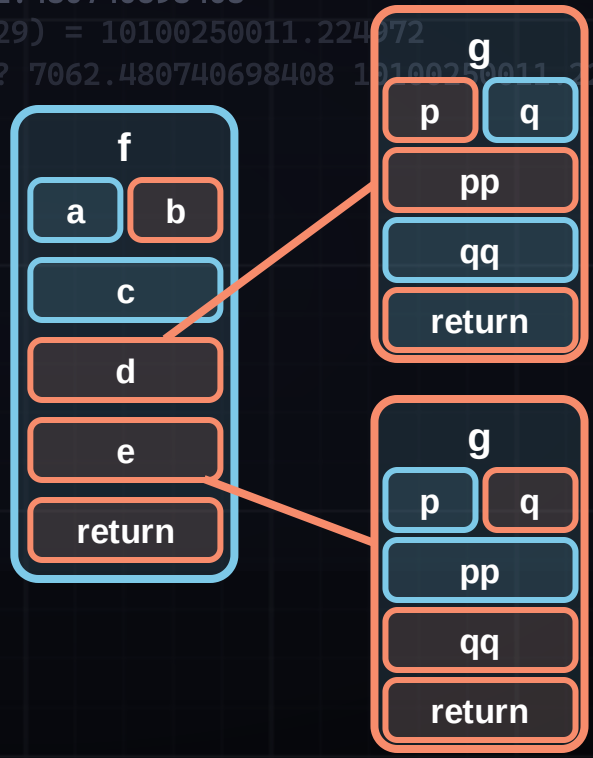


```ts
const f = (a: string, b: number) => {  // a = "Any questions?", b = 43
  const c = a.toUppercase()  // c = "ANY QUESTIONS?"
  const d = g(b * 2, 42)  // g(86, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 129) = 10100250011.224972
  return `${c} ${d} ${e}`  // "ANY QUESTIONS? 7062.480740698408 10100250011.224972"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 43
f(a, b)
```

Декларативный TSX

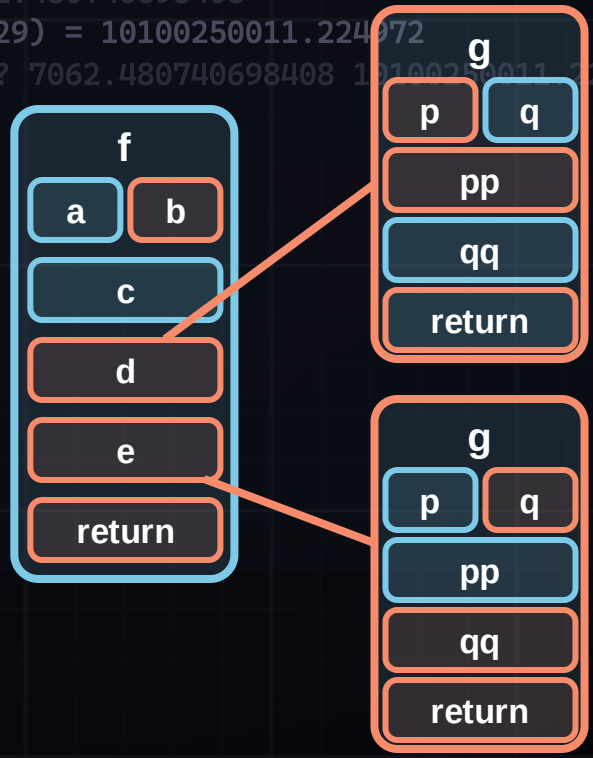


```ts
const f = (a: string, b: number) => {  // a = "Any questions?", b = 43
  const c = a.toUppercase()  // c = "ANY QUESTIONS?"
  const d = g(b * 2, 42)  // g(86, 42) = 7062.480740698408
  const e = g(100500, b * 3)  // g(100500, 129) = 10100250011.224972
  return `${c} ${d} ${e}`  // "ANY QUESTIONS? 7062.480740698408 10100250011.224972"
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 43
f(a, b)
```

Декларативный TSX



Декларативный TSX

```
const f = (a: string, b: number) => {
  const c = a.toUppercase()
  const d = g(b * 2, 42)
  const e = g(100500, b * 3)
  return `${c} ${d} ${e}`
}

const g = (p: number, q: number) => {
  const pp = p * p
  const qq = Math.sqrt(q)
  return pp + qq
}

let a = "Any questions?"
let b = 43
f(a, b)
```

```ts
const f = (a: Computed<string>, b: Computed<number>): Computed<string> => {
  const c = computed(() => a.get().toUppercase())
  const d = g(computed(() => b.get() * 2), 42)
  const e = g(100500, computed(() => b.get() * 3))
  return computed(() => `${c.get()} ${d.get()} ${e.get()}`)
}

const g = (p: Computed<number>, q: Computed<number>): Computed<string> => {
  const pp = computed(() => p.get() * p.get())
  const qq = computed(() => Math.sqrt(q.get()))
  return computed(() => pp.get() + qq.get())
}

const a = signal("Any questions?")
const b = signal(43)
f(a, b)
```

Наблюдения

- Инкрементальные вычисления и сигнальная реактивность похожи
- Дерево вызовов функций <==> Дерево графов сигналов
- Порядок вычислений ± одинаковый
- Минимизация "лишних" перевычислений

Компиляция DSL в сигналы

Компиляция DSL в сигналы

Преимущества

- Выглядит как TypeScript
- Нет boilerplate ".get()", ".set()", "signal()", "computed(() => ...)"
- Есть возможность автоматически оптимизировать

Компиляция DSL в сигналы

Преимущества

- Выглядит как TypeScript
- Нет boilerplate ".get()", ".set()", "signal()", "computed(() => ...)"
- Есть возможность автоматически оптимизировать

Недостатки

- "Магия" порядка исполнения вычислений
- +1 шаг сборки
- Ограничения синтаксиса и используемых API

Декларативный TSX: ограничения

- **Запрещён "лишний" синтаксис:**
  - `class`, `this`, `new`, `delete`, `var`, `function`, **циклы** `for - in`, методы (в т.ч. `get` - и `set` -) и т.д.
- **"Белый список" API:**
  - нет прямого доступа к Web APIs
  - import-ить можно другие модули декларативного TSX
- **Чистота функций:**
  - отсутствие циклов, присваиваний, побочных эффектов
  - Изменения данных - только в специальных контекстах
- **Типизация:**
  - Запрет небезопасных `as`, `any`
  - Все property должны быть readonly, кроме тех, которые можно менять
- ...



"Escape Hatches"

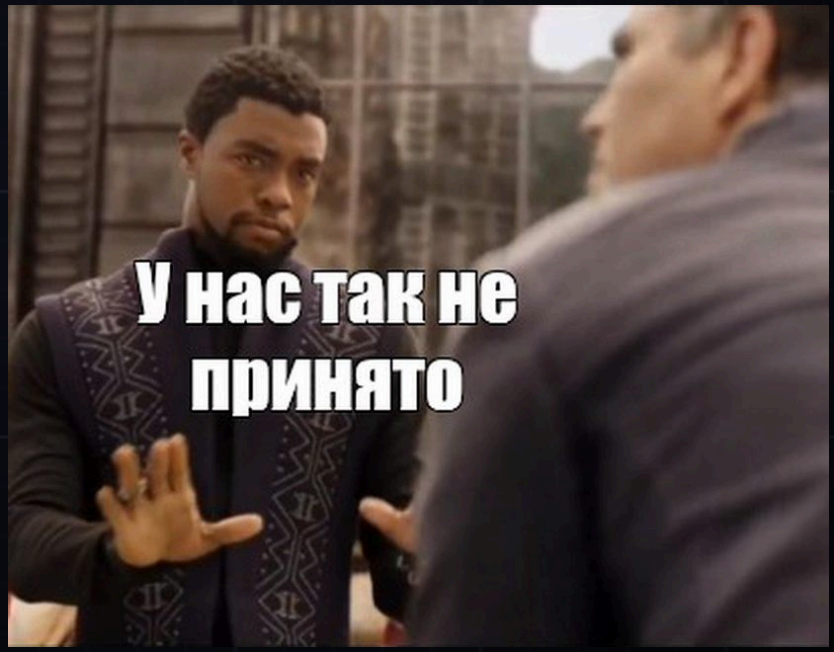
"Где мои ...?"

- `useEffect`
- `useRef`
- `watch`
- `watchEffect`
- `onMounted`
- `onUnmounted`
- и т.п

"Escape Hatches"

"Где мои ...?"

- `useEffect`
- `useRef`
- `watch`
- `watchEffect`
- `onMounted`
- `onUnmounted`
- и т.п





Вложенная реактивность

Объекты как значения

```
let a = 100500.0
let b = "Whatever"

const c = { foo: a, bar: b }
```

Декларативный TSX

Объекты как значения

```
const a = signal(100500.0)
const b = signal("Whatever")

const c = computed(() ⇒ ({ foo: a.get(), bar: b.get() }))
```

Объекты как значения

```
const a = signal(100500.0)
const b = signal("Whatever")

const c = computed(() => ({ foo: a.get(), bar: b.get() }))

const d = computed(() => `${c.get().foo} is ${c.get().bar}!`)
```

Объекты как значения

```
const a = signal(100500.0)
const b = signal("Whatever")

const c = { foo: a, bar: b }

const d = computed(() => `${c.foo.get()} is ${c.bar.get()}!`)
```

Объекты как значения

```
const a = signal(100500.0)
const b = signal("Whatever")

const c = { foo: a, bar: b }

const { foo, bar } = c

const d = computed(() => `${foo.get()} is ${bar.get()}!`)
```

Объекты как значения

```
const a = signal(100500.0)
const b = signal("Whatever")

const c = { foo: a, bar: b }

const { foo, bar } = c

const d = computed(() => `${foo.get()} is ${bar.get()}!`)

const e = computed(() => JSON.stringify(c))   // ???
```

Объекты как значения

```
const a = signal(100500.0)
const b = signal("Whatever")

const c = { foo: a, bar: b }

const { foo, bar } = c

const d = computed(() => `${foo.get()} is ${bar.get()}!`)

const e = computed(() => JSON.stringify(snapshot(c)))
```

Изменяемые объекты

```
let a = 100500.0
let b = "Whatever"

let c = { foo: a, bar: b }
```

Декларативный TSX

Изменяемые объекты

```
let a = 100500.0
let b = "Whatever"

let c: {
  foo: number
  bar: string
} = { foo: a, bar: b }
```

Декларативный TSX

Изменяемые объекты

```
let a = 100500.0
let b = "Whatever"

let c: {
  readonly foo: number
  readonly bar: string
} = { foo: a, bar: b }
```

Декларативный TSX

Реактивные массивы

Декларативный TSX

```
const weekdays: string[] = ["Пн", "Вт", "Ср", "Чт"]
const weekdaysUpper: readonly string[] = weekdays.map(wd => wd.toUppercase())
const weekdaysLower: readonly string[] = weekdays.map(wd => wd.toLowercase())

weekdays.push("Пт", "Сб", "Вс")
weekdays.splice(2, 1)
weekdays.sort()

reconcile(weekdays, ["Вт", "Чт", "Сб"])
```

Реактивные массивы

```
const weekdays: string[] = ["Пн", "Вт", "Ср", "Чт"]
const weekdaysUpper: readonly string[] = weekdays.map(wd => wd.toUppercase())
const weekdaysLower: readonly string[] = weekdays.map(wd => wd.toLowercase())

weekdays.push("Пт", "Сб", "Вс")
weekdays.splice(2, 1)
weekdays.sort()

reconcile(weekdays, ["Вт", "Чт", "Сб"])
```

Декларативный TSX

Реактивные массивы

Декларативный TSX

```
const weekdays: string[] = ["Пн", "Вт", "Ср", "Чт"]
const weekdaysUpper: readonly string[] = weekdays.map(wd => wd.toUppercase())
const weekdaysLower: readonly string[] = weekdays.map(wd => wd.toLowercase())

weekdays.push("Пт", "Сб", "Вс")
weekdays.splice(2, 1)
weekdays.sort()

reconcile(weekdays, ["Вт", "Чт", "Сб"])
```

Реактивные массивы

Декларативный TSX

```
const weekdays: string[] = ["Пн", "Вт", "Ср", "Чт"]
const weekdaysUpper: readonly string[] = weekdays.map(wd => wd.toUppercase())
const weekdaysLower: readonly string[] = weekdays.map(wd => wd.toLowercase())

weekdays.push("Пт", "Сб", "Вс")
weekdays.splice(2, 1)
weekdays.sort()

reconcile(weekdays, ["Вт", "Чт", "Сб"])
```

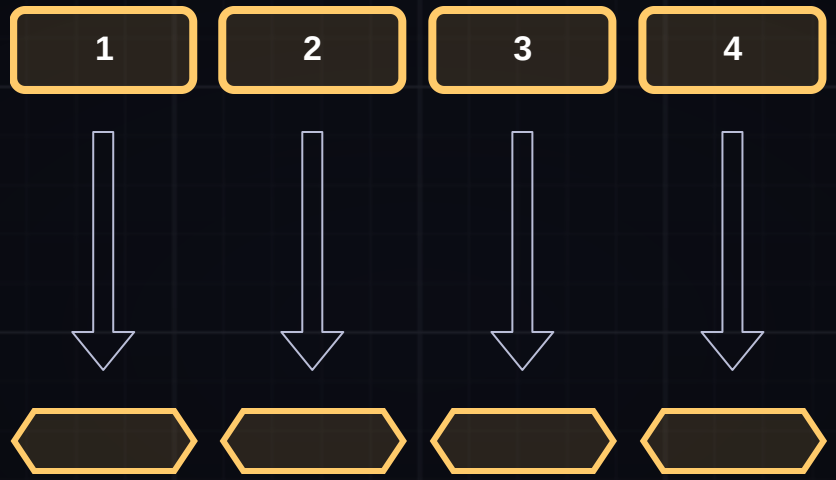
Реактивные массивы

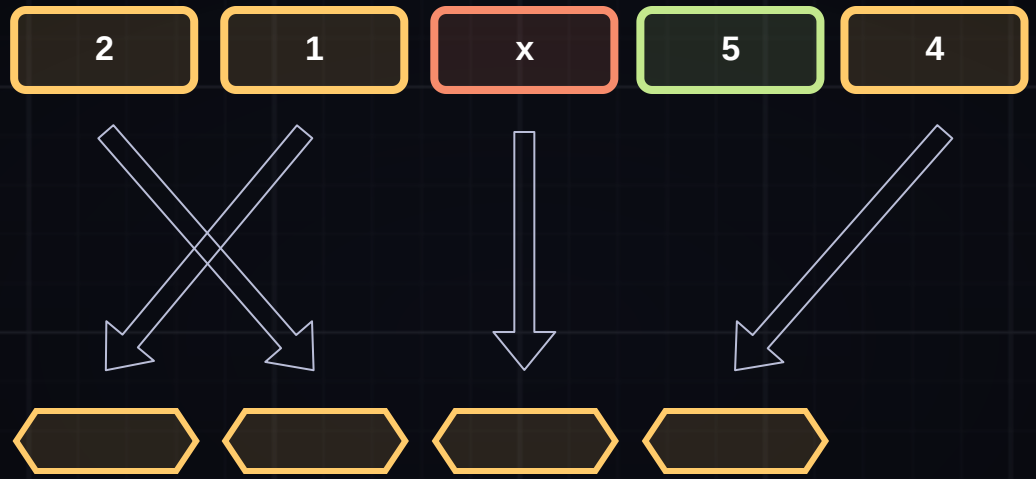
Декларативный TSX

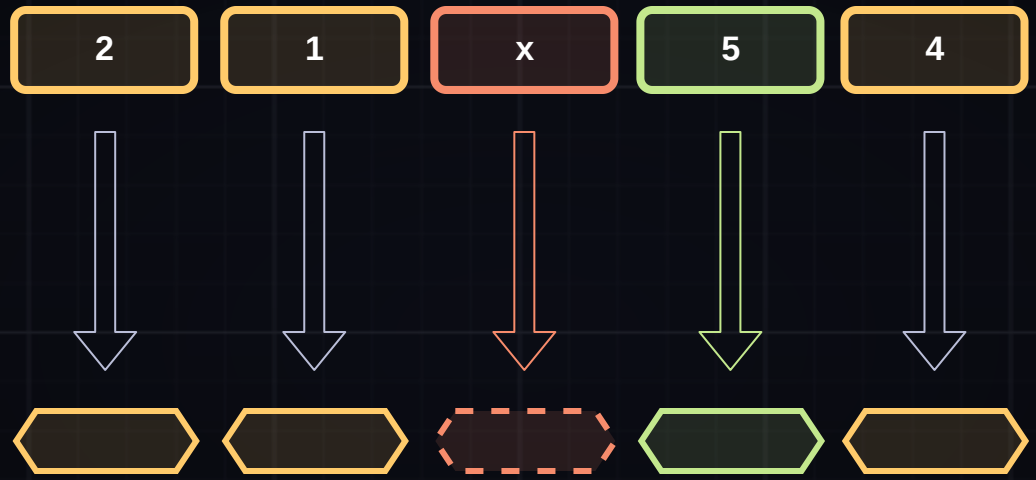
```
const weekdays: string[] = ["Пн", "Вт", "Ср", "Чт"]
const weekdaysUpper: readonly string[] = weekdays.map(wd => wd.toUppercase())
const weekdaysLower: readonly string[] = weekdays.map(wd => wd.toLowercase())

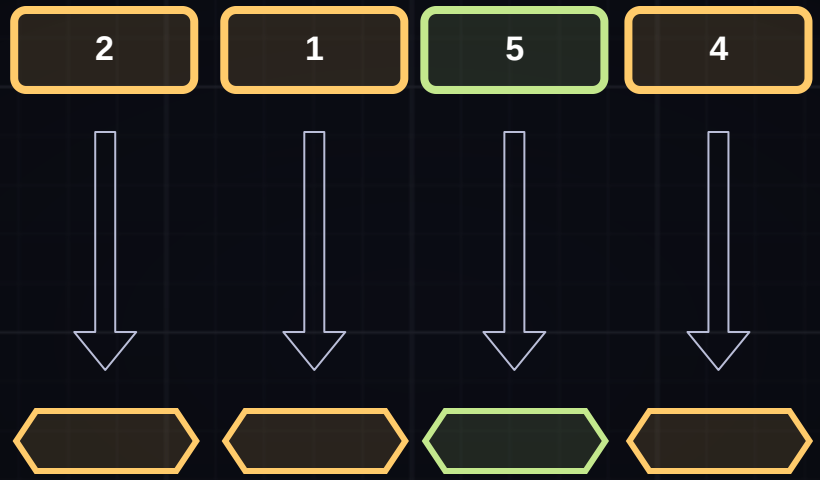
weekdays.push("Пт", "Сб", "Вс")
weekdays.splice(2, 1)
weekdays.sort()

reconcile(weekdays, ["Вт", "Чт", "Сб"])
```


1
2
3
4


2
1
x
5
4


2
1
x
5
4


2
1
5
4

Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input = "" }}>Добавить</button>
    <ul>{todos.filter(({ done }) => !showDone || done).map(({ task, done }) => (
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.indexOf(task), 1) }}>Удалить</button>
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```

Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input
    <ul>{todos.filter(({ done }) => !showDone || done).
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.in
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```



Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input = "" }}>Добавить</button>
    <ul>{todos.filter(({ done }) => !showDone || done).map(({ task, done }) => (
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.indexOf(task), 1) }}>Удалить</button>
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```

Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input = "" }}>Добавить</button>
    <ul>{todos.filter(({ done }) => !showDone || done).map(({ task, done }) => (
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.indexOf(task), 1) }}>Удалить</button>
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```

Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input = "" }}>Добавить</button>
    <ul>{todos.filter(({ done }) => !showDone || done).map(({ task, done }) => (
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.indexOf(task), 1) }}>Удалить</button>
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```

Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input = "" }}>Добавить</button>
    <ul>{todos.filter(({ done }) => !showDone || done).map(({ task, done }) => (
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.indexOf(task), 1) }}>Удалить</button>
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```

Декларативный TSX

```
type Todo = { task: string; done: boolean }
const TodoList = () => {
  const todos: Todo[] = [{ task: "Ничего", done: true }]
  let input: string = ""
  let showDone: boolean = false
  return (<>
    <input type="text" bind:value={input}>
    <button on:click={() => { todos.push(input); input = "" }}>Добавить</button>
    <ul>{todos.filter(({ done }) => !showDone || done).map(({ task, done }) => (
      <li>
        <input type="checkbox" bind:value={done}/>
        <input type="text" bind:value={task}/>
        <button on:click={() => { todos.splice(todos.indexOf(task), 1) }}>Удалить</button>
      </li>
    ))}</ul>
    <button on:click={() => { showOnlyDone = !showOnlyDone }}>
      Показать сделанные
    </button>
  </>)
}
```

Вложенные объекты

```
type Data = {
  foo: { readonly a: string; readonly b: boolean } | null
  bar: {
    readonly c: number
    d: string | number
    readonly e: readoly boolean[]
  }[]
  baz: { readonly kind: "A"; a: string } | { readonly kind: "B"; b: number }
}
```



Древовидные данные

Алгебраические типы данных

"Value-types"

Декларативный TSX: ограничения

- ...
- **Древовидные данные:**
  - Только объекты, массивы и примитивы
  - Копирование объектов, а не передача по ссылке
  - Запрет оператора `[]` для свойств объектов и индексной сигнатуры
  - Запрет сравнения объектов по ссылке (===)
- **Специальный API:**
  - реактивные массивы вместо обычных
  - `snapshot` и `reconcile` для вложенных данных



Что осталось за кадром?

- callbacks, event handlers, render-props (slots)
- Context/Provide-Inject
- Keep-alive - кеширование состояния компонентов
- Асинхронная логика, например, запросы на сервер
- SSR, SSG

Примеры реализации похожих идей

Декларативные DSL:

-  Marko
- Compose - компиляторный плагин для  Kotlin
- "Реактивный CSS"
- QML - декларативный язык для 

State Managers:

- Signia - инкрементально вычисляемые сигналы
- Tanstack DB - live queries

Заключение

Заключение

Заключение

- Инкрементальные вычисления интересны в плане производительности и DX

Заключение

- Инкрементальные вычисления интересны в плане производительности и DX
- Ментальная модель UI - основа декларативного языка UI-фреймворка

Заключение

- Инкрементальные вычисления интересны в плане производительности и DX
- Ментальная модель UI - основа декларативного языка UI-фреймворка
- Многие фреймворки приходят к этой концепции:
  - Те же соглашения и ограничения
  - Те же паттерны и best practices

Спасибо!



Василий Алфертьев

Telegram: @alfertev2012

GitHub: @alfertev2014