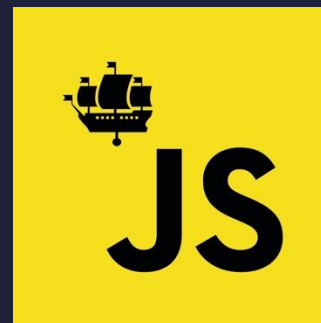


**Василий Беляев**

Frontend Team Lead

# Как мы компоненты оптимизировали, оптимизировали, да выоптимизировали



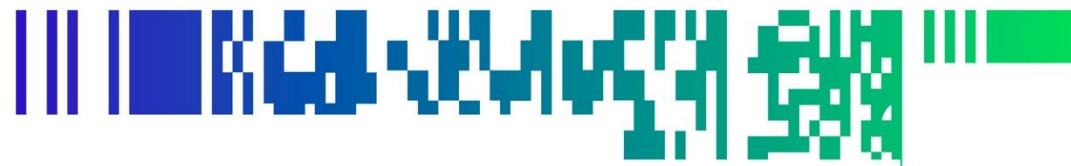
# ВАСИЛИЙ БЕЛЯЕВ

Frontend Team Lead







- В IT уже более 18 лет – начинал с верстальщика сайтов, и умею верстать на таблицах
- Сейчас руковожу группой разработки интерфейсов в «Криптоните» и много думаю про оптимизацию компонентов
- РАБОТАЛ В ТАКИХ КОМПАНИЯХ, КАК «РБК», «REDFOX», «RUSOFT»/«RUCARD».**

# ЧЕМ ЗАНИМАЕМСЯ



## КРИПТОНИТ

-  Мы **разрабатываем** ПО и программно-аппаратные комплексы для хранения, обработки и анализа больших данных с помощью моделей машинного обучения.
-  Занимаемся **исследованиями** в области криптографии, информационной безопасности и больших данных.
-  **Просвещаем** – при нашей поддержке был создан Музей криптографии в Москве
-  **600+ ЧЕЛОВЕК, 39 СОТРУДНИКОВ С УЧЕНОЙ СТЕПЕНЬЮ**

# НАШ СТЕК НА FRONT'e



JavaScript



TypeScript



Vue.js



Pinia



Vite



Vitest



ESlint



Storybook



Stylelint



Playwright



node.js



yarn



# • КТО ЗАНИМАЕТСЯ РАЗРАБОТКОЙ БИБЛИОТЕК?



- А КТО ЗНАЕТ, СКОЛЬКО ВЕСЯТ БАНДЛЫ БИБЛИОТЕК?

ПРЕДИСЛОВИЕ

# ПРОБЛЕМЫ С БИБЛИОТЕКАМИ

# ПРОБЛЕМЫ



 Размер бандла

 Большое количество лишних зависимостей

 Конфликт версий – и такое тоже было!

## СКОРОСТЬ СБОРКИ ПРОЕКТА

- из-за размера бандлов – долго выкачивались пакеты
- из-за большого количества зависимостей – долго происходил деплой проекта



# ПРОБЛЕМЫ В ЦИФРАХ



 68 мб – вес бандлов библиотек

 45 минут – сборка всех библиотек

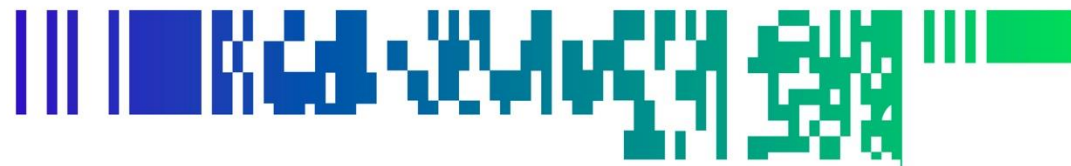
 **12,5 МИНУТ – СРЕДНЕЕ ВРЕМЯ СБОРКИ ПРОЕКТОВ**






ПРЕДИСЛОВИЕ

# НЕМНОГО ТЕРМИНОВ

# ЧТО ЕСТЬ ЧТО?

## Сначала по терминам

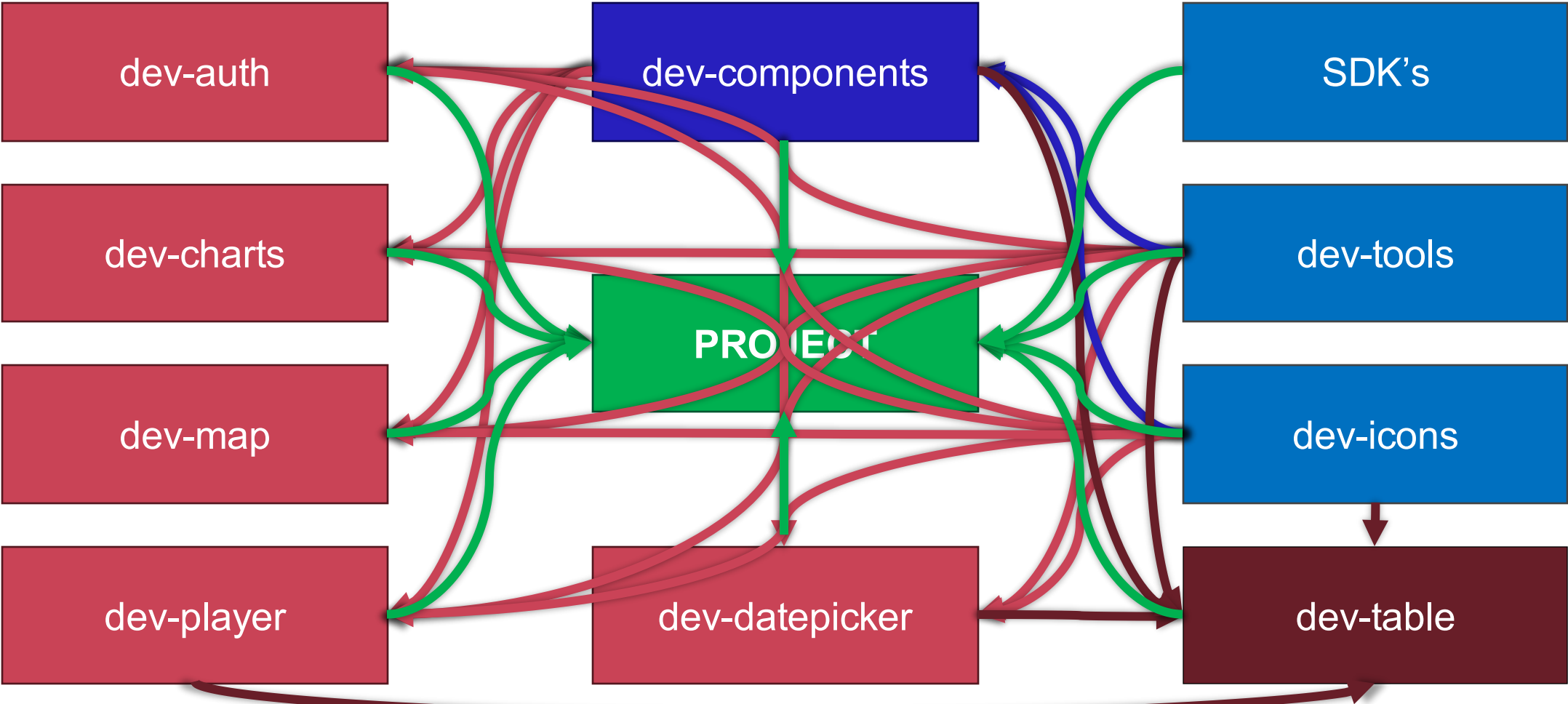


-  **ПРОЕКТ** — продукт, который мы разрабатываем для заказчика или распространения
-  **CORE-БИБЛИОТЕКА** — библиотека, на основе которой строится проект или наша библиотека
-  **БИБЛИОТЕКА** — наша самописная библиотека, которую мы поддерживаем и развиваем
-  **КОМПОНЕНТ** — единица библиотеки, которую можем вызвать и использовать
-  **ЗАВИСИМОСТЬ** — все, что тянется из `npmjs` или `nexus`

ПРОБЛЕМЫ

# ЧТО БЫЛО ИЗНАЧАЛЬНО С КОМПОНЕНТАМИ?

# КАК ВЫГЛЯДИТ ХАОС В БИБЛИОТЕКАХ



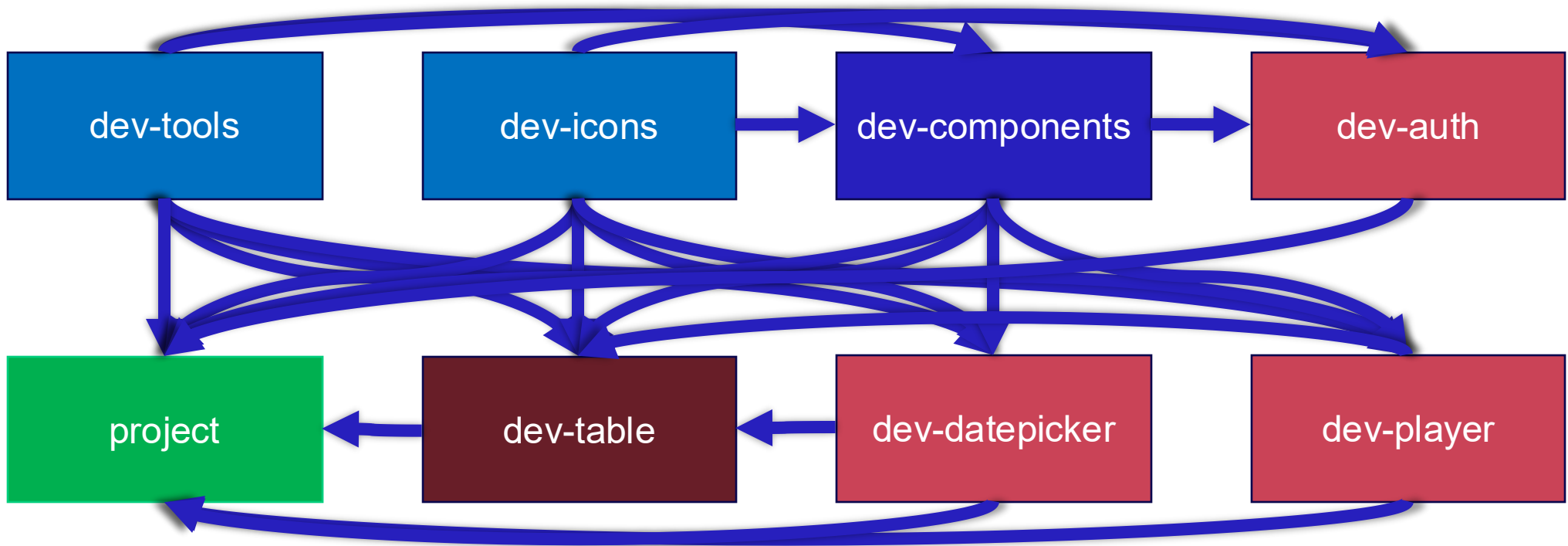
ПРОБЛЕМЫ

# ОТДЕЛЬНЫЙ СЛУЧАЙ

# ЗАВИСИМОСТИ ОТ DEV-TOOLS ДО ПРОЕКТА



→ - Прямые зависимости



РЕШЕНИЯ

# КАК МОЖНО ИСПРАВИТЬ?



# ПУТИ РЕШЕНИЯ ПРОБЛЕМ



 Убираем импорты

 Перебираем зависимости

 Оптимизируем package.json

## ПРОВЕРЯЕМ

- Проверяем работу на проектах
- Радует результат
- Находим новые проблемы

РЕШЕНИЯ

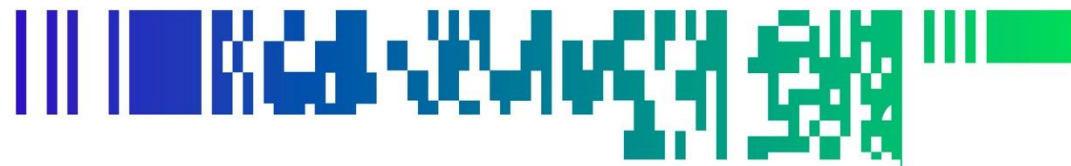
# ПОЙДЕМ ПО ПОРЯДКУ

ИМПОРТЫ

# РАБОТА С ИМПОРТАМИ

# ИМПОРТЫ

## И проблема с ними



Явные импорты компонентов тянут в себе код компонента

1 импорт компонента == 1 копия компонента

Раздувание библиотеки до вселенских размеров

**ИНОГДА СЛУЧАЛИСЬ КОЛЛИЗИИ С АКТУАЛЬНОСТЬЮ  
ОСНОВНОЙ БИБЛИОТЕКИ**

# ЧТО БЫЛО ИЗНАЧАЛЬНО?

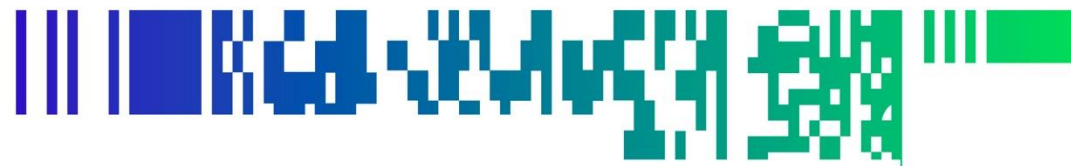


● ● ● ./src/component1/component.vue

```
import Component1 from './component1.vue';  
import Component2 from './component2.vue';  
import Component3 from 'dev-components';
```

//и это все внутри самих компонентов

# КАК ИСПОЛЬЗОВАЛИ КОМПОНЕНТЫ РАНЬШЕ



```
./src/component1/component.vue

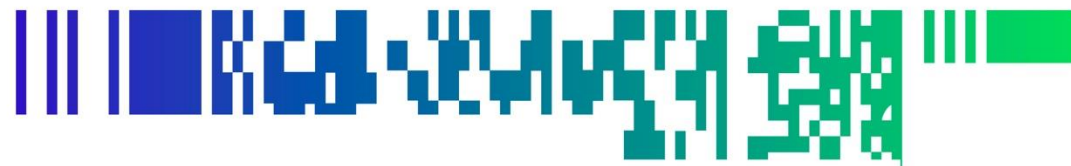
<script setup>
import Component1 from './component1.vue';
import Component2 from './component2.vue';
import Component3 from 'dev-components';
</script>

<template>
  <Component1/>
  <Component2/>
  <Component3/>
</template>
//и это все внутри самих компонентов
```

РЕШЕНИЕ

# ПЛАГИНЫ, КАК РЕШЕНИЕ

# ОТЛИЧИЕ КОМПОНЕНТА ОТ ПЛАГИНА?



## Импорт компонента

- Импортирует код компонента в указанное место
- НЕ проверяет на повтор подключения
- Раздувает библиотеку до больших размеров

## Импорт плагина

- Добавляет ссылку на компонент в библиотеке
- Проверяет наличие дубликата в глобальном scope и повторно не подключает компонент
- **ПОЗВОЛЯЕТ СОКРАТИТЬ РАЗМЕР БИБЛИОТЕКИ ДО АДЕКВАТНЫХ РАЗМЕРОВ**



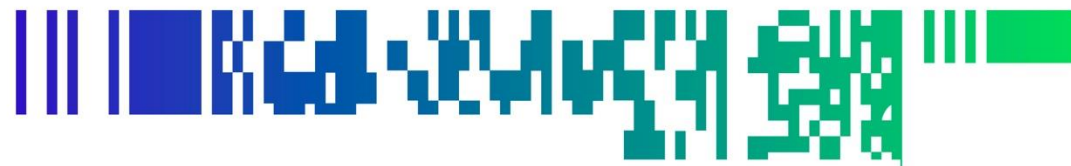
# ЧТО МОЖНО СДЕЛАТЬ



## ВМЕСТО ПРЯМЫХ ИМПОРТОВ СОЗДАЕМ ПЛАГИНЫ

- Компонент подключаем теперь только через созданный плагин
- Проверяем существование компонента в глобальном scope

# ПРИМЕР ПЛАГИНА ДЛЯ КОМПОНЕНТА

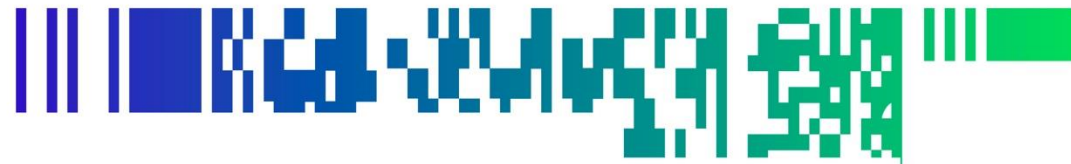


```
./src/index.ts

import DevTable from './DevTable.vue';

export default {
  install: (app) => {
    app.component('DevTable', DevTable);
  },
};

export { default as DevTable } from './DevTable.vue';
```



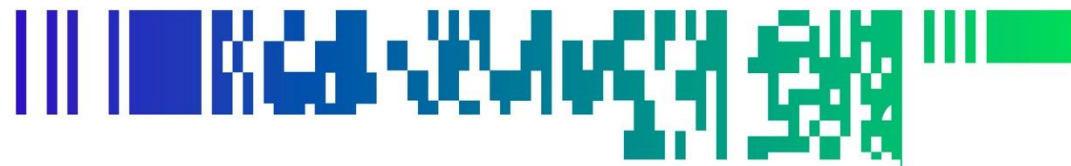
 Вместо прямых импортов создаем плагины

## ПЛАГИНЫ ИМПОРТИРУЕМ В ГЛОБАЛЬНОМ ФАЙЛЕ ПОДКЛЮЧЕНИЯ

- Создаем единый файл подключения компонентов в библиотеке
- Подключаем все библиотеки в глобальный score проекта/библиотеки
- Для подключения используем только плагины

# ЧТО ПОЛУЧИЛОСЬ

С файлом глобального импорта



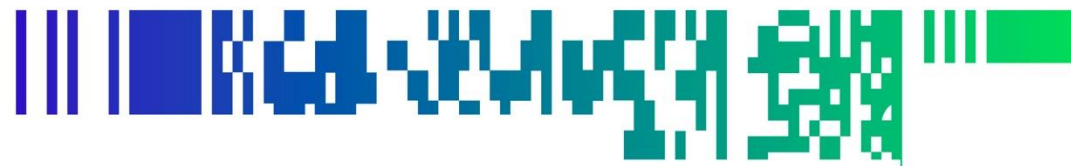
```
./src/index.ts

import DevTablePlugin from '@components';
export default {
  install: (app) {
    app.use(DevTablePlugin);
  },
}
export { cellFactory } from './helpers/cells';
export { DevTable } from '@components';
```



- Вместо прямых импортов создаем плагины
- Плагины импортируем в глобальном файле подключения
- УБИРАЕМ ИМПОРТЫ ИЗ САМИХ КОМПОНЕНТОВ**
  - Проходим все файлы и смотрим на импорты
  - Убираем все импорты вида: `import component from './component.vue'`
  - Подключаем плагины в проекте

# ИСПОЛЬЗОВАНИЕ ИМПОРТОВ



```
./src/index.ts

import DevDatepicker from 'dev-datepicker';
import DevComponents from 'dev-components';
import { createApp } from 'vue';
import DevTablePlugin from '@components';
import App from './App.vue';

(async () => {
  const app = createApp(App);
  app.use(DevComponents, {});
  app.use(DevDatepicker);
  app.use(DevTablePlugin);
  app.mount('#app');
})();
```

# PROFIT

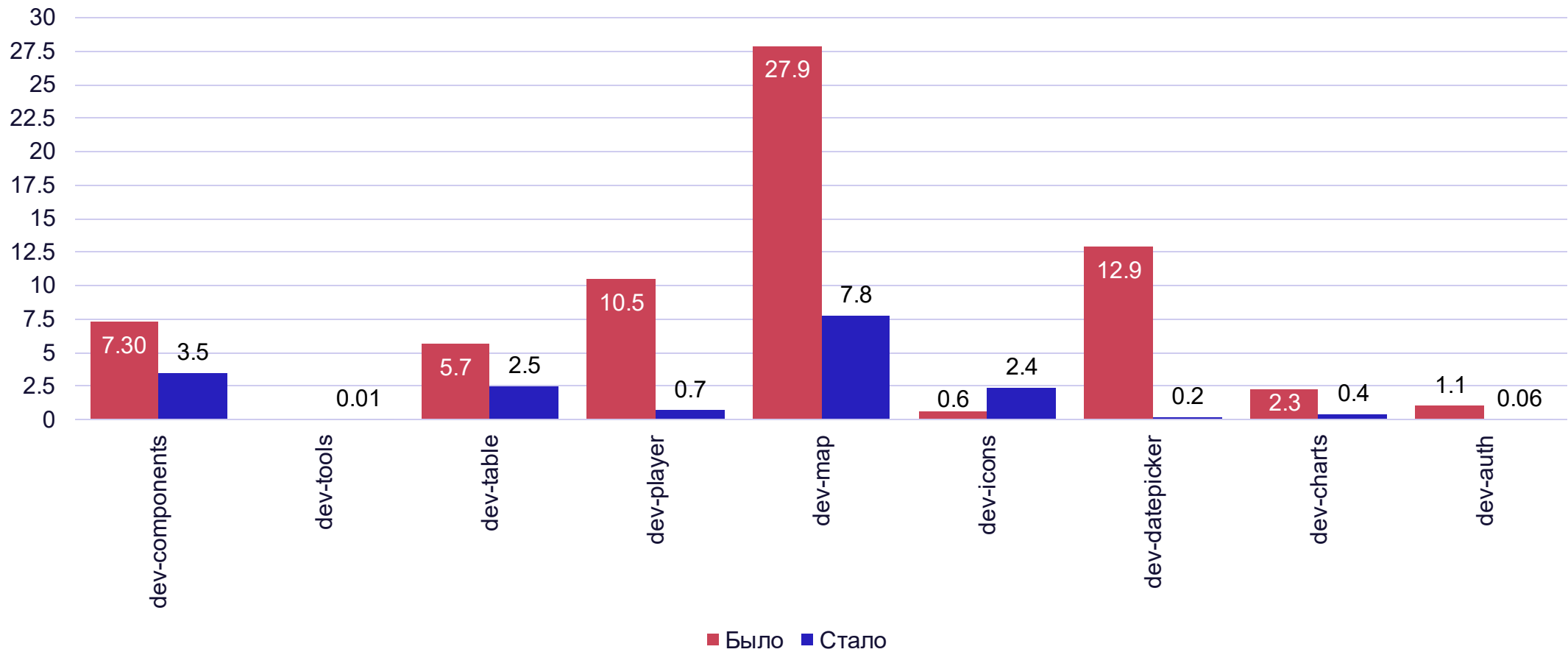


 Убрали прямые импорты в компонентах наших библиотек

 Сократили размер бандлов наших библиотек

# PROFIT

## Размеры бандлов





# PROFIT В ЦИФРАХ



**74%**

Было: 68,3 Мб

**Стало: 17,3 Мб**

И это только начало!

ЗАВИСИМОСТИ

# РАБОТА С ЗАВИСИМОСТЯМИ

# ПРОБЛЕМЫ, КОТОРЫЕ РЕШАЕМ



- На данном этапе мы исправляем скорость сборки
- Беспорядок в зависимостях
- Убираем лишние библиотеки

# КАК РАБОТАЕТ PACKAGE.JSON

В рамках библиотек



 **dependencies**

 **devDependencies**

 **PEER-DEPENDENCIES**

- зависимости, необходимые для работы библиотеки
- служат указателем нужных версий библиотек для корректной работы библиотеки



• КТО ЗНАЕТ ОТЛИЧИЯ УСТАНОВКИ  
ЗАВИСИМОСТЕЙ В NPM и YARN?

# КАК РАБОТАЕТ PACKAGE.JSON

В рамках библиотек



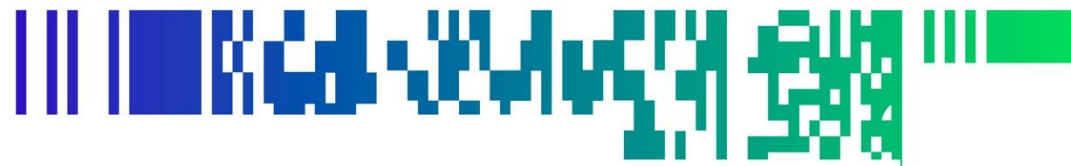
 dependencies

 devDependencies

 PEER-DEPENDENCIES

- зависимости, необходимые для работы библиотеки
- служат указателем нужных версий библиотек для корректной работы библиотеки
- при использовании yarn **не устанавливаются!**
- при использовании npm устанавливаются, начиная с версии v7

# ЧТО ПОЛУЧИЛОСЬ У НАС?



```
./lib/package.json

dependencies: {
  "dev-icons": "^0.1",
  "dev-tools": "^0.1",
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "core-libs": "^1",
  ...
},
"devDependencies": {
  ...
}
```

```
./lib/package.json

"dependencies": {
  "core-libs": "^1",
  ...
},
"devDependencies": {
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "dev-icons": "^0.1",
  "dev-tools": "^0.1"
},
"peerDependencies": {
  ...
  "dev-components": "^0.2",
}
```

# NPM I --PRODUCTION

Как ведет себя флаг --production?



 `npm install --production`

 при использовании этого флага устанавливаются зависимости из секций **dependencies** и **peerDependencies**

 ускоряет разворачивание проекта на dev стенд (окружение для разработки)

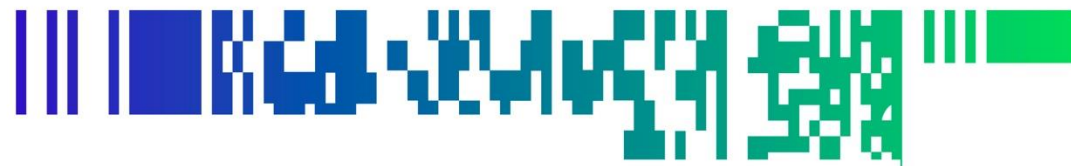
 ускоряет сборку бандла библиотеки

 устанавливает 35 зависимостей



# YARN INSTALL --PROD

Чем отличается от npm?



■ yarn install --prod

■ при использовании этого флага устанавливаются зависимости **только из секции dependencies**

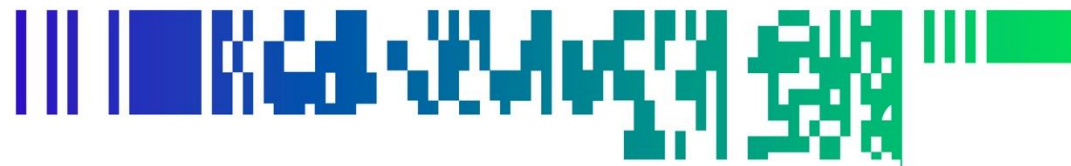
■ еще больше ускоряет разворачивание проекта на dev стенд (окружение для разработки)

■ еще больше ускоряет сборку бандла библиотеки

■ устанавливает 15 зависимостей

# КАК РАБОТАЮТ ЗАВИСИМОСТИ

## Установка в проекте



```
./package.json
{
  ...
  dependencies: {
    "core-lib": "^1.0.0"
  },
  devDependencies: {
    "dev-lib": "^1.0.0"
  },
}
```

```
Terminal
... % yarn install
```

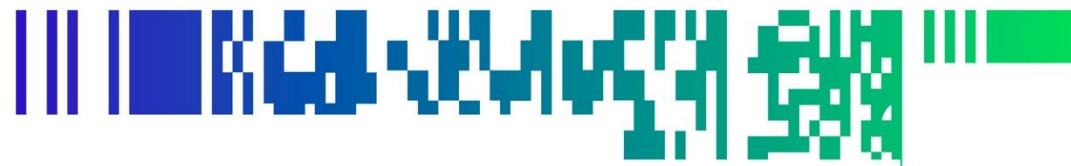
### Результат установки:

```
core-lib@1.0.0
dev-lib@1.0.0
```

```
Terminal
success Saved lockfile.
🌟 Done in 15.69s.
```

# КАК РАБОТАЮТ ЗАВИСИМОСТИ

## Установка в проекте с --prod



```
./package.json
{
  ...
  dependencies: {
    "core-lib": "^1.0.0"
  },
  devDependencies: {
    "dev-lib": "^1.0.0"
  },
}
```

```
Terminal
... % yarn install --prod
```

### Результат установки:

```
core-lib@1.0.0
```

```
Terminal
success Saved lockfile.
🌟 Done in 15.69s.
```

# КАК РАБОТАЮТ ЗАВИСИМОСТИ С peerDependencies



```
lib/package.json
{
  ...
  dependencies: {
    "core-lib-pack": "^1.0.0",
    "dayjs": "^1.0.0"
  },
  devDependencies: {
    "dev-lib-pack": "^1.0.0",
  },
  peerDependencies: {
    "peer-lib": "^1.0.0"
  }
}
```

```
./package.json
{
  ...
  dependencies: {
    "core-lib": "^1.0.0",
    "datepicker": "^1.0.0"
  },
  devDependencies: {
    "dev-lib": "^1.0.0"
  },
}
```

```
Terminal
... % yarn install
```

## Результат установки:

```
core-lib@1.0.0
datepicker@1.0.0
core-lib-pack@1.0.0
dayjs@1.0.0

dev-lib@1.0.0
```

```
Terminal
warning " > dev-lib@1.0.0"
has unmet peer dependency
"peer-lib@>=1.0.0".
```

# КАК РАБОТАЮТ ЗАВИСИМОСТИ

С библиотеками + установка --prod



```
lib/package.json
{
  ...
  dependencies: {
    "core-lib-": "^1.0.0",
    "dayjs": "^1.0.0"
  },
  devDependencies: {
    "dev-lib-pack": "^1.0.0",
    "peer-lib": ">=1.0.0"
  },
  peerDependencies: {
    "peer-lib": ">=1.0.0"
  }
}
```

```
./package.json
{
  ...
  dependencies: {
    "core-lib": "^1.0.0",
    "datepicker": "^1.0.0"
  },
  devDependencies: {
    "dev-lib": "^1.0.0"
  },
}
```

```
Terminal
... % yarn install --prod
```

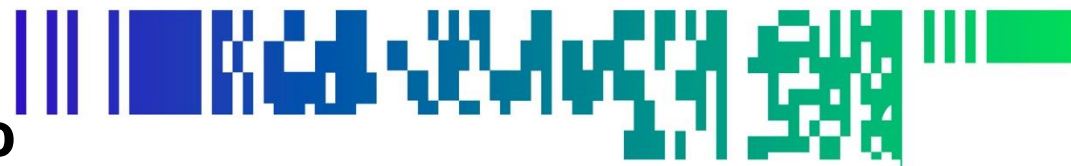
## Результат установки:

```
core-lib@1.0.0
datepicker@1.0.0
core-lib-pack@1.0.0
dayjs@1.0.0
```

```
Terminal
warning " > dev-lib@1.0.0"
has unmet peer dependency
"peer-lib@>=1.0.0".
```

# КАК ИСПРАВИТЬ?

Просто добавить пакет в нужную секцию



```
lib/package.json

{
  ...
  dependencies: {
    "core-lib-": "^1.0.0",
    "dayjs": "^1.0.0"
  },
  devDependencies: {
    "dev-lib-pack": "^1.0.0",
    "peer-lib": ">=1.0.0"
  },
  peerDependencies: {
    "peer-lib": ">=1.0.0"
  }
}
```

```
./package.json

{
  ...
  dependencies: {
    "core-lib": "^1.0.0",
    "datepicker": "^1.0.0"
  },
  devDependencies: {
    "dev-lib": "^1.0.0",
    "peer-lib": ">=1.0.0"
  },
}
```

```
Terminal

... % yarn install --prod
```

## Результат установки:

```
core-lib@1.0.0
datepicker@1.0.0
core-lib-pack@1.0.0
dayjs@1.0.0
```

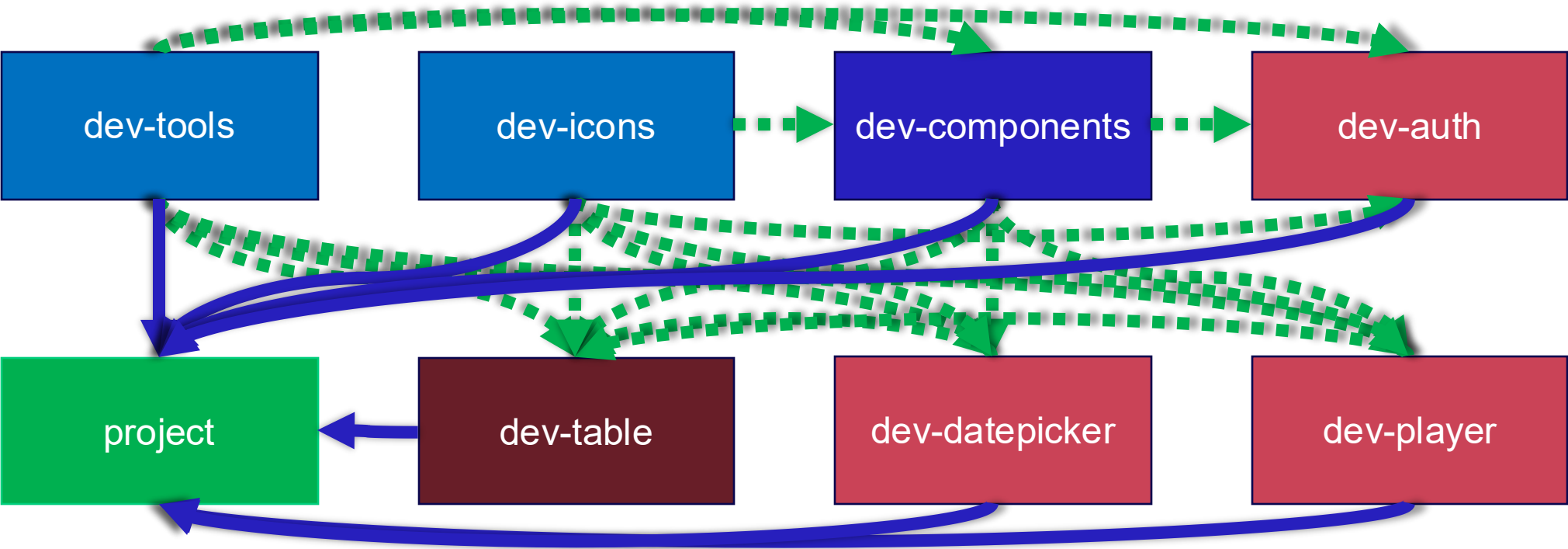
```
Terminal

success Saved lockfile.
🌟 Done in 15.69s.
```

# ЧТО ПОЛУЧИЛОСЬ ПО ЗАВИСИМОСТЯМ



.....➡ - devDeps  
➡ - prodDeps



# В ЧЕМ НАШ ПРОФИТ?

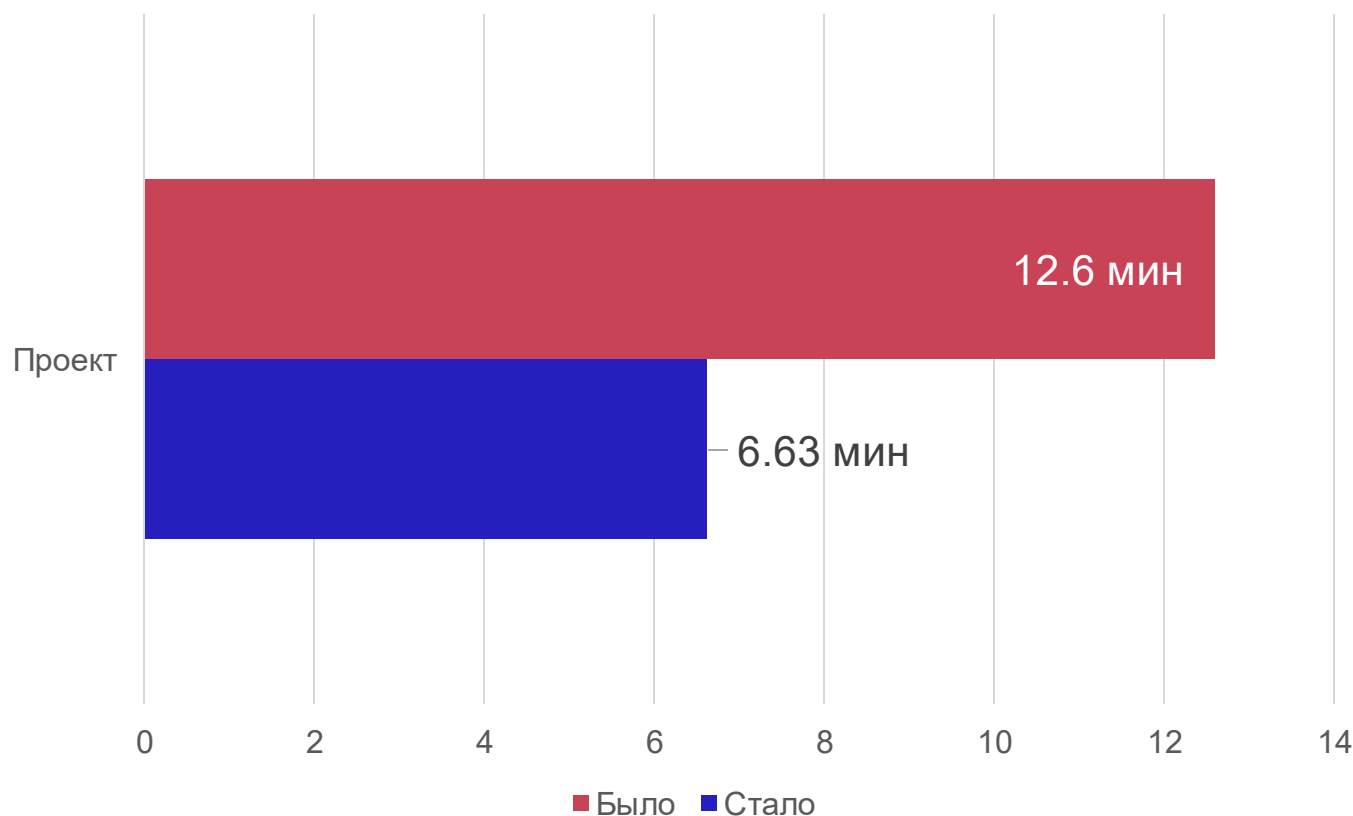
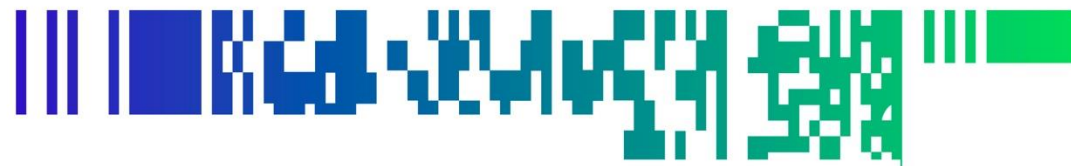


- Мы стали тянуть меньше зависимостей при установке библиотек
- Скорость сборки проекта стала быстрее
- Теперь не библиотеки управляют нужными версиями библиотек, а проекты – поведение стало более предсказуемым с точки зрения работы компонентов в библиотеках



# PROFIT

## Время сборки проектов



# 47.4%

Было: 12 мин 36 сек

**Стало: 6 мин 38 сек**

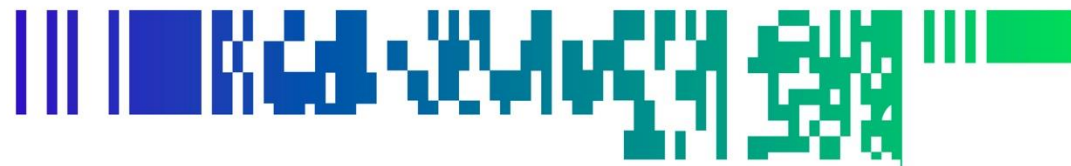
\* По средним замерам  
скорости сборки проектов

WORKFLOW

# ИЗМЕНЕНИЯ В КОМАНДЕ

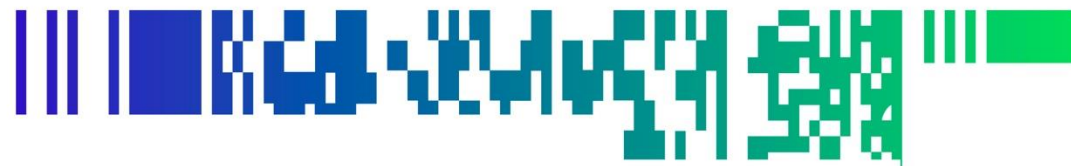
# ИЗМЕНЕНИЯ

К этому этапу



- Мы выросли и нас, frontend-разработчиков стало больше!
- К нам присоединилась еще одна команда frontend-разработчиков
- Проектов стало больше
- Библиотеки стали активнее развиваться
- **СТАЛО СЛОЖНЕЕ СЛЕДИТЬ ЗА ИЗМЕНЕНИЯМИ**

# РОСТ КОМАНДЫ И НОВЫЕ ПРОБЛЕМЫ



 «Не работает ...»

 «Обновили и потеряли совместимость»

 «Что-то поехало в компоненте»

# ЧТО МОЖНО СДЕЛАТЬ?

## Меняем workflow



- Все изменения в компонентах проводим через ui-kit
- Вводим еженедельный синк по компонентам Дизайн + Аналитика + Фронт
- Вводим оповещения об изменениях в компонентах
- **ВВОДИМ РАСПИСАНИЕ ВЫПУСКА ВЕРСИЙ**

# CORE-КОМАНДА

Выделяем отдельно



Кроме изменения workflow мы еще выделяем core-команду:

Кто? – По 1-2 разработчика из каждой группы разработки интерфейсов

Для чего? – Делать кросс-командное review

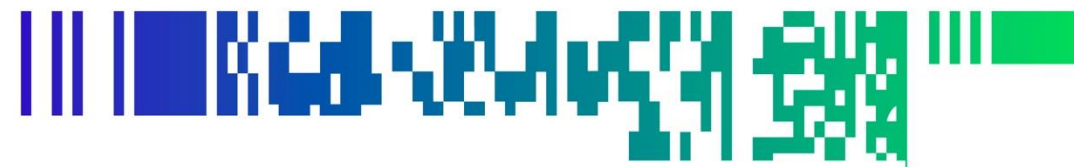
Следить за соблюдением workflow

**ОБСУЖДЕНИЕ РЕАЛИЗАЦИИ КОМПОНЕНТОВ И УЧАСТИЕ В СИНКАХ**

# ЧТО ПОЛУЧИЛОСЬ?

Было

КТО ПОЛОМАЛ  
TEXTINPUT?  
- Вася, это ты  
сделал?



СТАЛО

Новый компонент или фича

Собираем информацию

Идем к UI/UX

Дожидаемся изменений в UI-kit

Заводим задачу и делаем её

Выпускаем новую версию библиотеки

# PROFIT



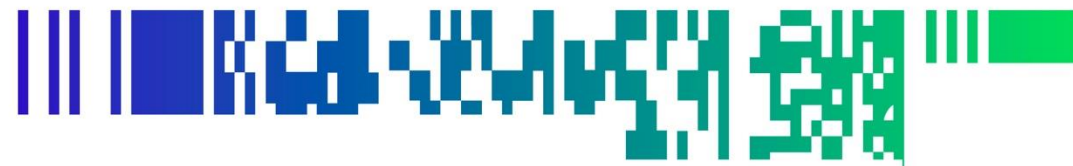
- Сделали процесс по разработке более прозрачным
- Библиотеки стали выпускаться по расписанию
- Сократили количество «неконтролируемых багов» в библиотеках
- Сократили количество внезапных изменений



КОНФЛИКТЫ

# КОНФЛИКТ ВЕРСИЙ

# ЕЩЕ ОДНА ПРОБЛЕМА



- У нас активно стала обновляться и дополняться библиотека иконок, в результате в интерфейсах обнаружались странные проблемы
- В какой-то момент обнаружили странное поведение – в зависимости от проекта и версий библиотек у нас пошло смещение иконок и их неправильное отображение

# ПЕРЕБИРАЕМ ЗАВИСИМОСТИ ЕЩЕ РАЗ



- Оставляем в prod-зависимостях только те зависимости, которые необходимы для работы библиотек (например, `dayjs` для `datepicker`)
- Версии core-библиотек выносим в `peerDependencies`

# ЕЩЕ РАЗ ПРАВИМ PACKAGE.JSON



```
./lib/package.json



"dependencies": {
  "core-lib-personal": "^0.1",
  "core-lib-shared": "^0.1",
  "dev-icons": "^0.1",
  ...
},
"devDependencies": {
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "dev-tools": "^0.1"
},
"peerDependencies": {
  ...
}
```

```
./lib/package.json

"dependencies": {
  "core-lib-personal": "^0.1",
},
"devDependencies": {
  ...
  "core-lib-shared": "^0.1",
  "dev-icons": "^0.1",
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "dev-tools": "^0.1"
},
"peerDependencies": {
  ...
  "dev-components": "^0.2",
}
```

# ПЕРЕСМАТРИВАЕМ ПРОЦЕСС СБОРКИ



-  Убираем флаг --prod для сборки библиотек в зависимостях
-  Проверяем работоспособность на проекте – все огонь, все работает, как надо, и иконки стали отображаться корректно

# МИНУСЫ




- Сборка библиотек замедлилась, но временно (тоже исправимо)
- В проектах нужно тянуть некоторые дополнительные библиотеки
- Иногда бывают merge request'ы, где что-то не нужное пытаются занести в prod (1 раз пропустил такой merge request с иконками в компоненты)
- Как и прежде, проект управляет версиями библиотек, которые ему нужны

# НО И ПЛЮСЫ



 Убираем конфликты версий!

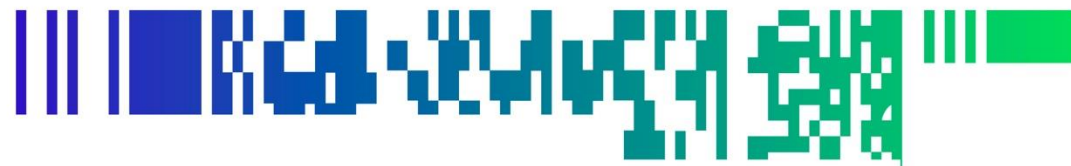
 За счет сокращения количества зависимостей в библиотеках в блоке `dependencies` и сокращения количества версий – ускоряем сборку проектов



WORKFLOW

**НО ЕСТЬ ЕЩЁ,  
ЧТО ИСПРАВЛЯТЬ**



# ОСТАЛИСЬ ЕЩЕ ПРОБЛЕМЫ



-  Периодическая потеря совместимости библиотек
-  Желание затянуть новый функционал в старые версии библиотек

# ВРЕМЕННОЕ РЕШЕНИЕ

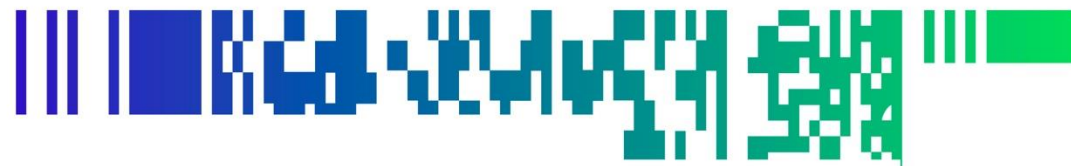


- Вводим срок жизни версии компонентов (поддерживаем только последние 3 версии)
- Пытаемся запретить делать cherry-pick из актуальной версии в старую

ЕЩЕ ОДНА ПРОБЛЕМА

# СОВМЕСТИМОСТЬ ВЕРСИЙ

# ПРОБЛЕМЫ, КОТОРЫЕ ХОТИМ РЕШИТЬ



- При обновлении библиотек иногда происходили изменения в API компонентов, которые ломали работу приложения
- «Зоопарк» версий библиотек в проектах
- От проекта к проекту отличались версии core-библиотек для проектов. И в библиотеках тоже
- Абсолютно разные файлы конфигураций для eslint, typescript и т.д. и для библиотек

# ЧТО ПРИДУМАЛИ?



■ Создать библиотеку с совместимыми версиями наших библиотек (dev-core)

■ **СОЗДАТЬ БИБЛИОТЕКУ С CORE-БИБЛИОТЕКАМИ И КОНФИГАМИ (DEV-CONFIG)**

- Добавляем все core-библиотеки (vue, dayjs и т.д.)
- Добавляем все конфиги (eslint, vite, storybook, typescript)
- Сборочные процессы (gitlab-ci, cliff)

РЕШЕНИЕ СОВМЕСТИМОСТИ НАШИХ  
БИБЛИОТЕК

# DEV-CORE





**• КТО ОБНОВЛЯЛ ЗАВИСИМОСТИ, И В ДРУГИХ БИБЛИОТЕКАХ ОТВАЛИВАЛИСЬ КОМПОНЕНТЫ?**

# ПРОБЛЕМЫ

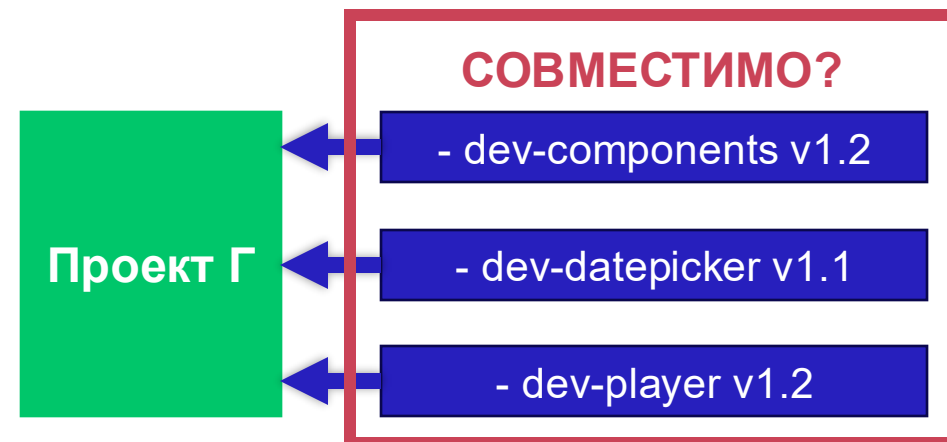
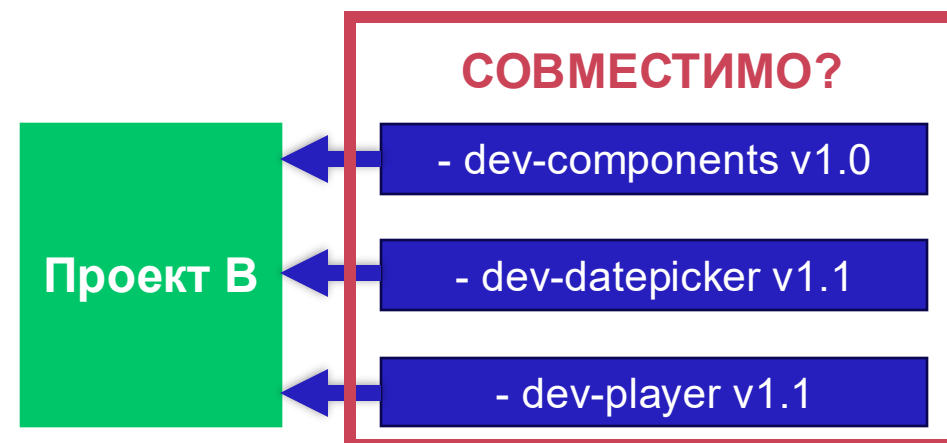
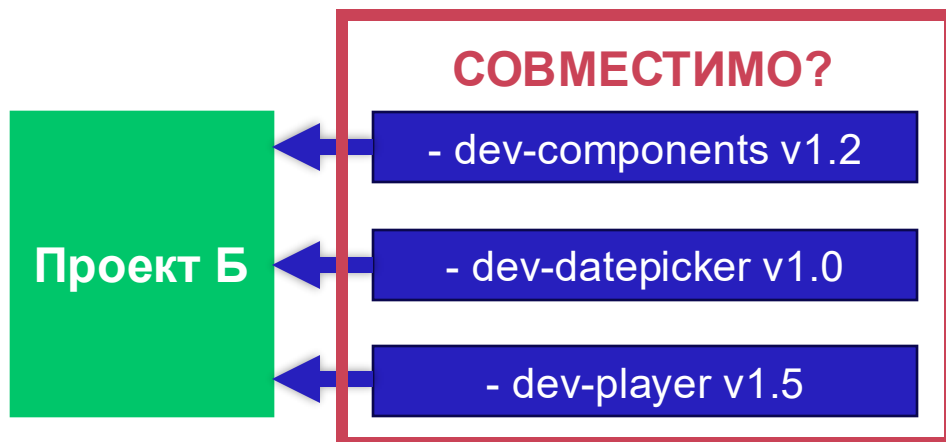
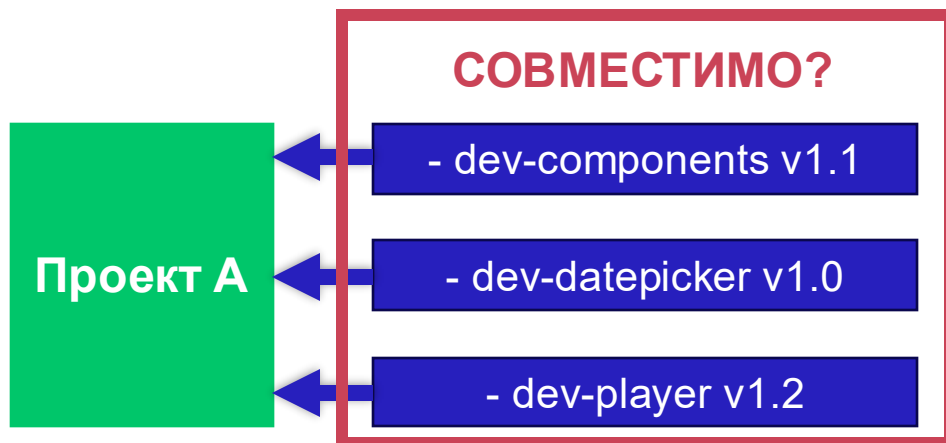
Которые решаем



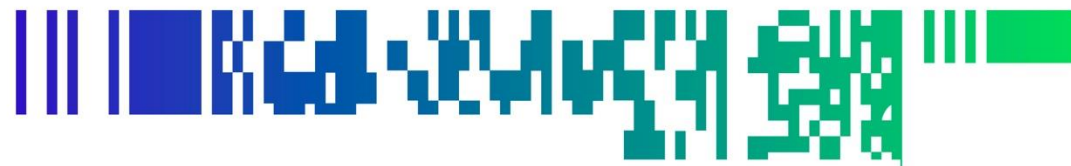
-  Отсутствие прозрачности по совместимости наших версий библиотек
-  Регулярные вопросы при поднятии версий библиотек



# КАК БЫЛО РАНЬШЕ?



# ЧТО ВХОДИТ В DEV-CORE?



 Совместимые версии библиотек

 Сборка и деплой общего storybook'а по компонентам

# МЕНЯЕМ WORKFLOW



■ Теперь после выпуска библиотек создаем merge request в dev-core

■ Прогоняем unit-тесты

■ Смотрим визуально

# МЕНЯЕМ WORKFLOW

Если есть упавшие тесты или баги



 Создаем задачи на исправление





 Ждем исправления

 Выпускаем новые версии

 Обновляем MR

# МЕНЯЕМ WORKFLOW

Если все исправили или все ок

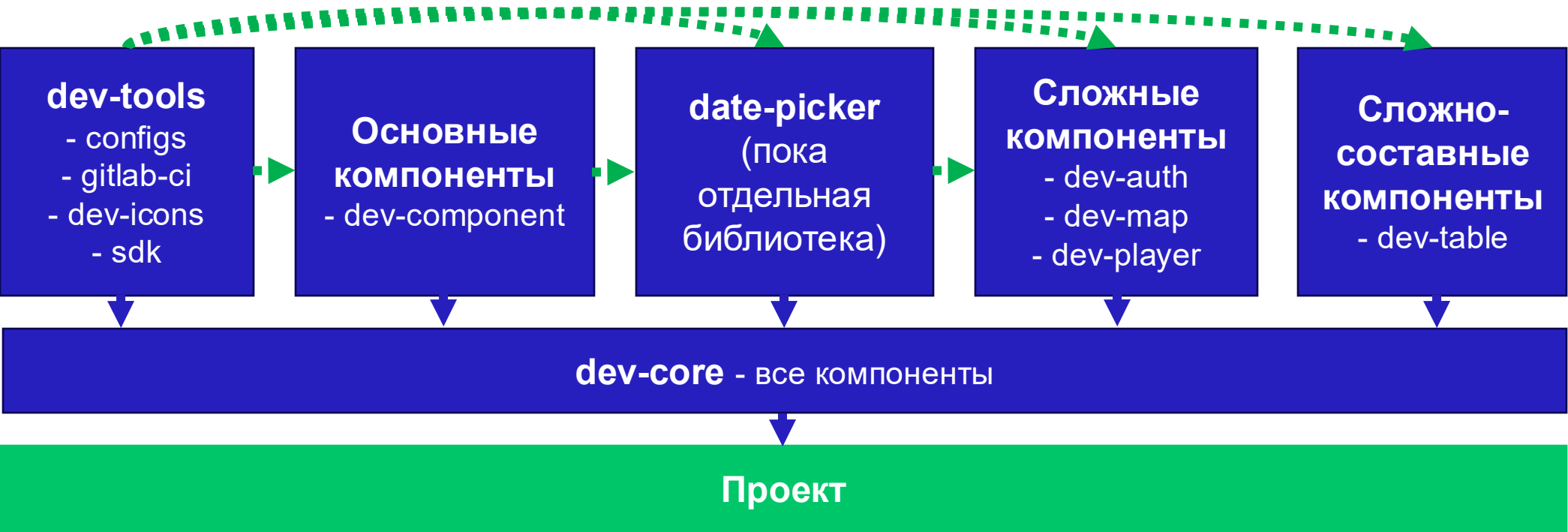
-  Выпускаем dev-core с фиксацией совместимых версий
-  Выпускаем storybook на все библиотеки
-  Выпускаем анонс по dev-core с изменениями в библиотеках
-  Используем в проектах



# ИТОГОВАЯ СХЕМА КОМПОНЕНТОВ



.....➡ - devDeps  
——➡ - prodDeps



# PACKAGE.JSON

## В проектах



```
./project/package.json

"dependencies": {
  "dev-icons": "^0.1",
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "dev-tools": "^0.1",
  "core-libs-shared": "^0.1",
  "core-libs-personal": "^0.1",
  ...
},
"devDependencies": {
  ...
  "dev-libs-shared ": "^0.1",
  "dev-libs-personal ": "^0.1",
},
```

```
./project/package.json

"dependencies": {
  "dev-core": "^0.1",
  "core-libs": "^0.1",
  "core-libs-shared": "^0.1",
  "core-libs-personal": "^0.1",
  ...
},
"devDependencies": {
  ...
  "dev-libs-shared ": "^0.1",
  "dev-libs-personal ": "^0.1",
},
```

# PROFIT

Если все исправили или все ок



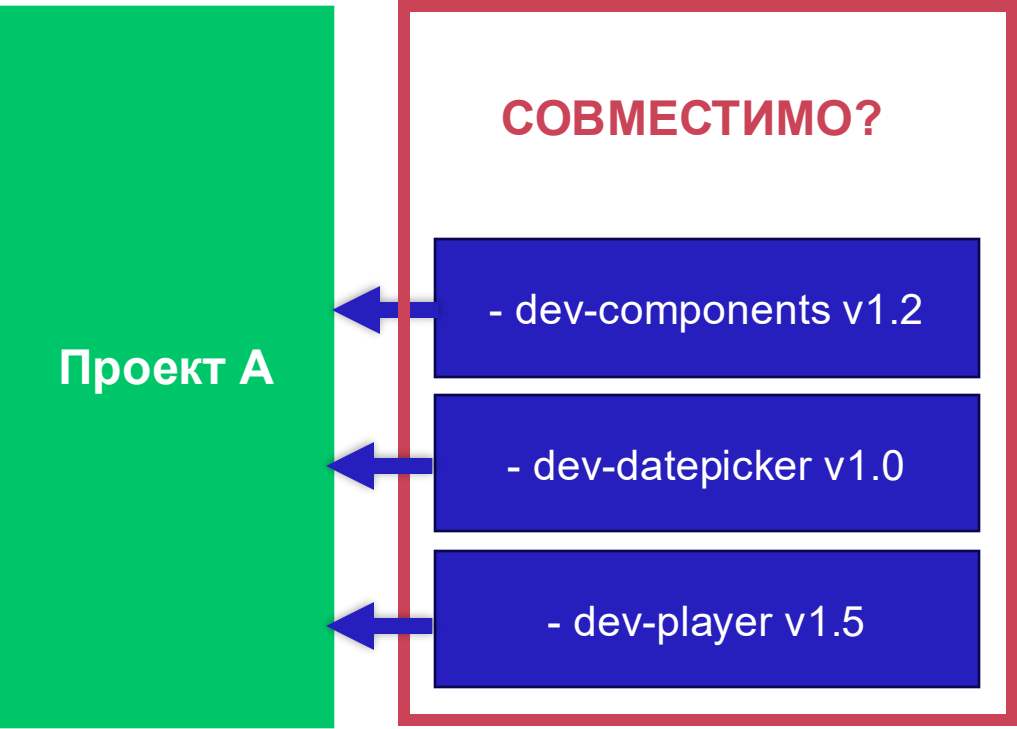
- Теперь у нас в рамках 1 версии dev-core – все библиотеки совместимы
- Убрали ответственность с разработчиков проектов по сопоставлению совместимых версий



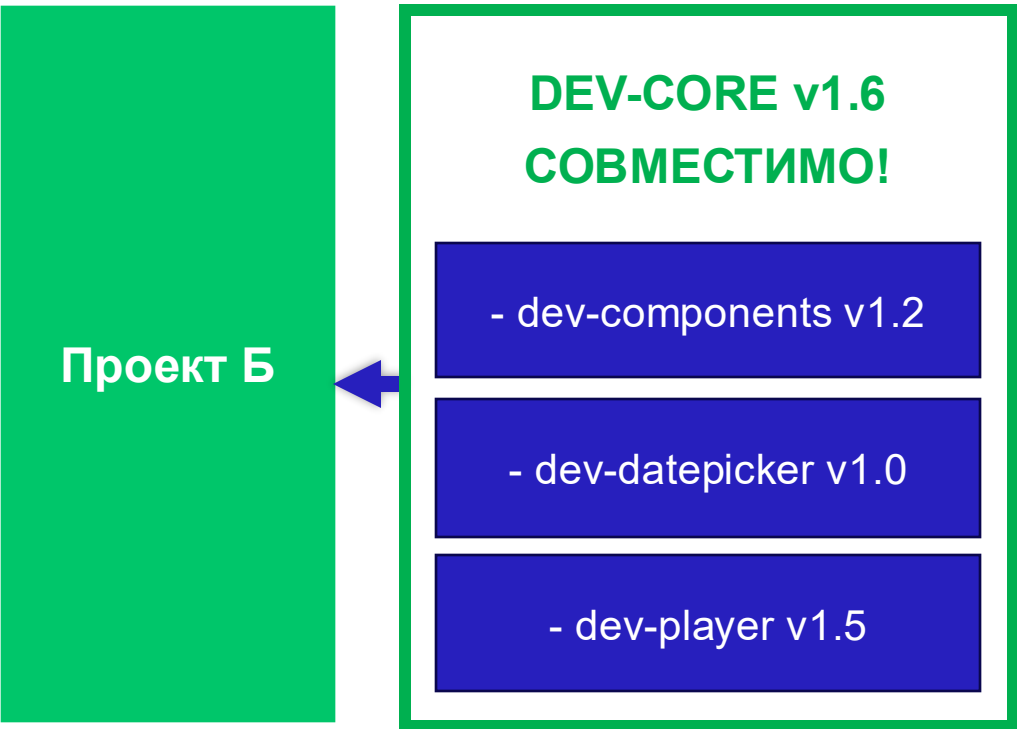
# ЧТО ПОЛУЧИЛОСЬ?



БЫЛО



СТАЛО



НО ЭТО ЕЩЕ НЕ ВСЕ

**НО МЫ НЕ МОГЛИ  
ОСТАНОВИТЬСЯ**

СИНХРОНИЗАЦИЯ CORE-БИБЛИОТЕК

# DEV-CONFIG

# ПРОБЛЕМЫ

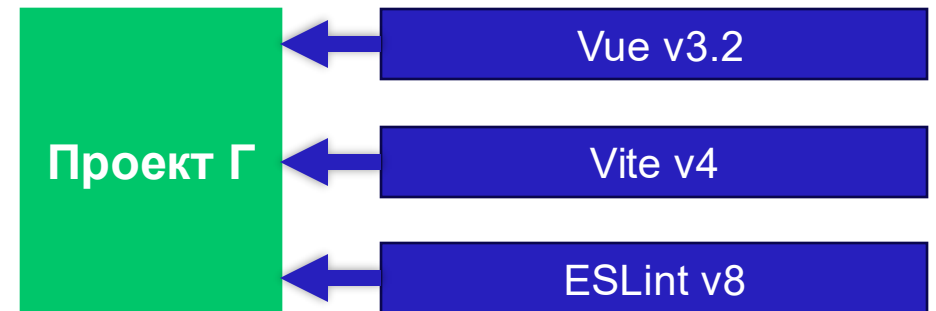
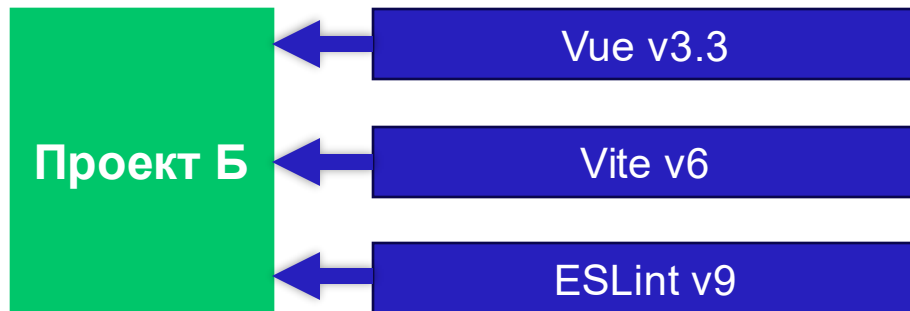
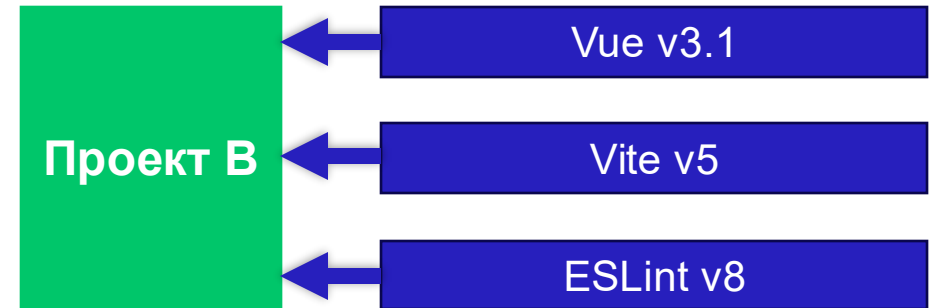
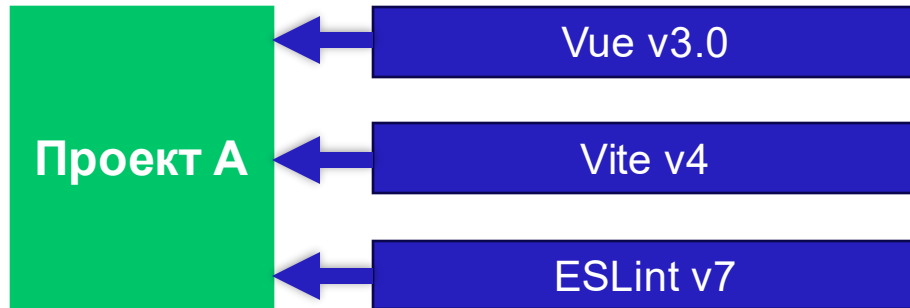
Которые решали



- «Зоопарк» версий библиотек в проектах
- От проекта к проекту отличались версии core-библиотек для проектов. И в библиотеках тоже
- Абсолютно разные файлы конфигов для eslint, typescript и т.д. и для библиотек

# КАК БЫЛО РАНЬШЕ?

Большое количество разных версий

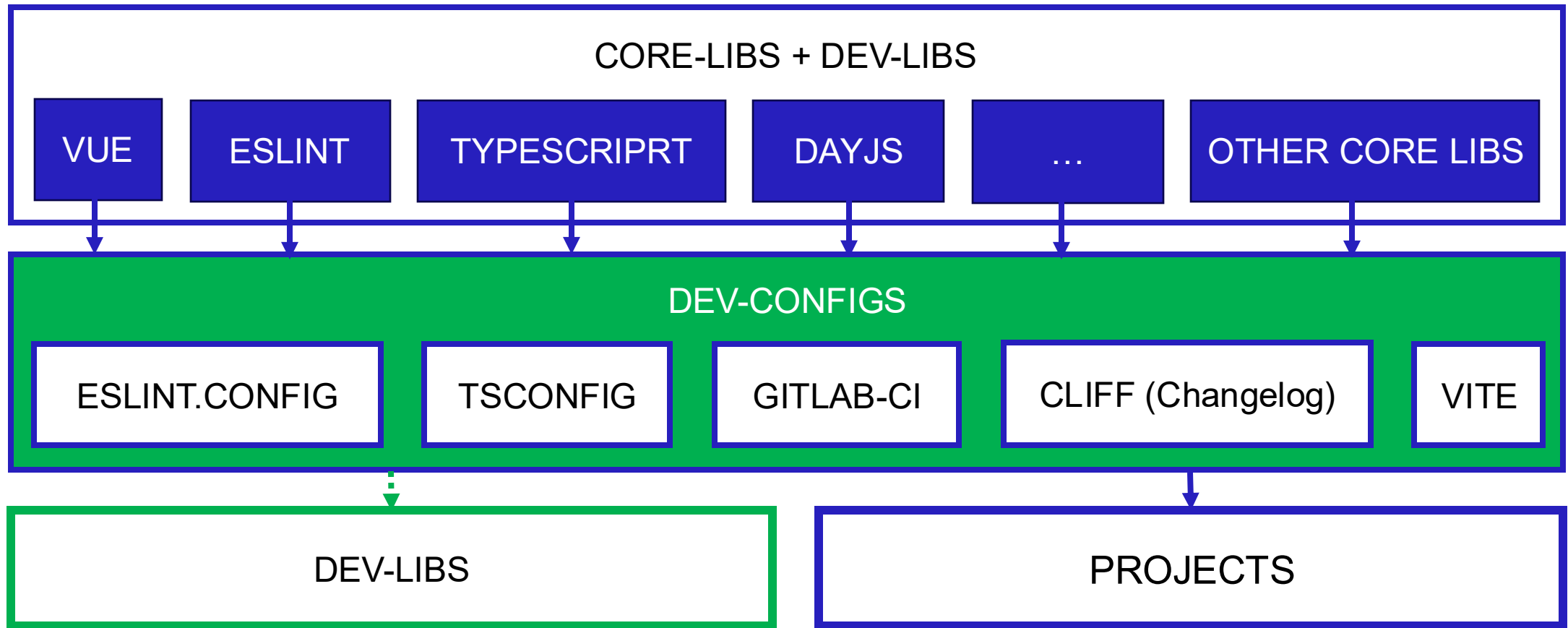


# ЧТО ТАКОЕ DEV-CONFIG И ДЛЯ ЧЕГО?



- dev-config – библиотека, которая управляет основными версиями зависимостей из npmjs для наших библиотек и проектов
- также содержит все конфиги для работы
- делает консистентным использование сборочных процессов

# СХЕМА РАБОТЫ DEV-CONFIG



# ВНОСИМ ИЗМЕНЕНИЯ В БИБЛИОТЕКИ



```
./lib/package.json

"dependencies": {
  "core-libs-personal": "^1.0"
},
"devDependencies": {
  "core-libs-shared": "^0.2",
  ...,
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "dev-icons": "^0.1",
  "dev-tools": "^0.1"
},
"peerDependencies": {
  ...,
  "dev-components": "^0.2",
```

```
./lib/package.json

"dependencies": {
  "core-libs-personal": "^1.0"
},
"devDependencies": {
  "dev-config": "^0.2",
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
  "dev-icons": "^0.1",
  "dev-tools": "^0.1"
},
"peerDependencies": {
  "dev-config": "^0.2",
  "dev-components": "^0.2",
  "dev-datepicker": "^0.2",
```



# ВНОСИМ ИЗМЕНЕНИЯ В DEV-CORE



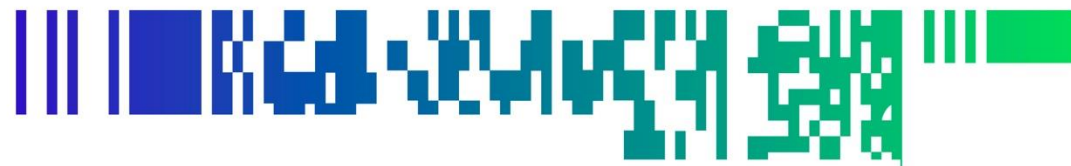
● ● ● ./lib/package.json

```
"dependencies": {  
  "dev-icons": "^0.1"  
  "dev-components": "^0.2",  
  "dev-datepicker": "^0.2",  
  "dev-tools": "^0.1",  
},
```

● ● ● ./lib/package.json

```
"dependencies": {  
  "dev-icons": "^0.1"  
  "dev-components": "^0.2",  
  "dev-datepicker": "^0.2",  
  "dev-tools": "^0.1",  
  "dev-config": "^0.2",  
},
```

# ЧТО ЕЩЕ СДЕЛАЛИ?



- Добавили cache для node\_modules и ускорили установку зависимостей при сборке

# ПЛЮСЫ



 Убрали «зоопарк» зависимостей

 Сделали единую точку для обновления core-библиотек

PROFIT В ГРАФИКАХ И ЦИФРАХ

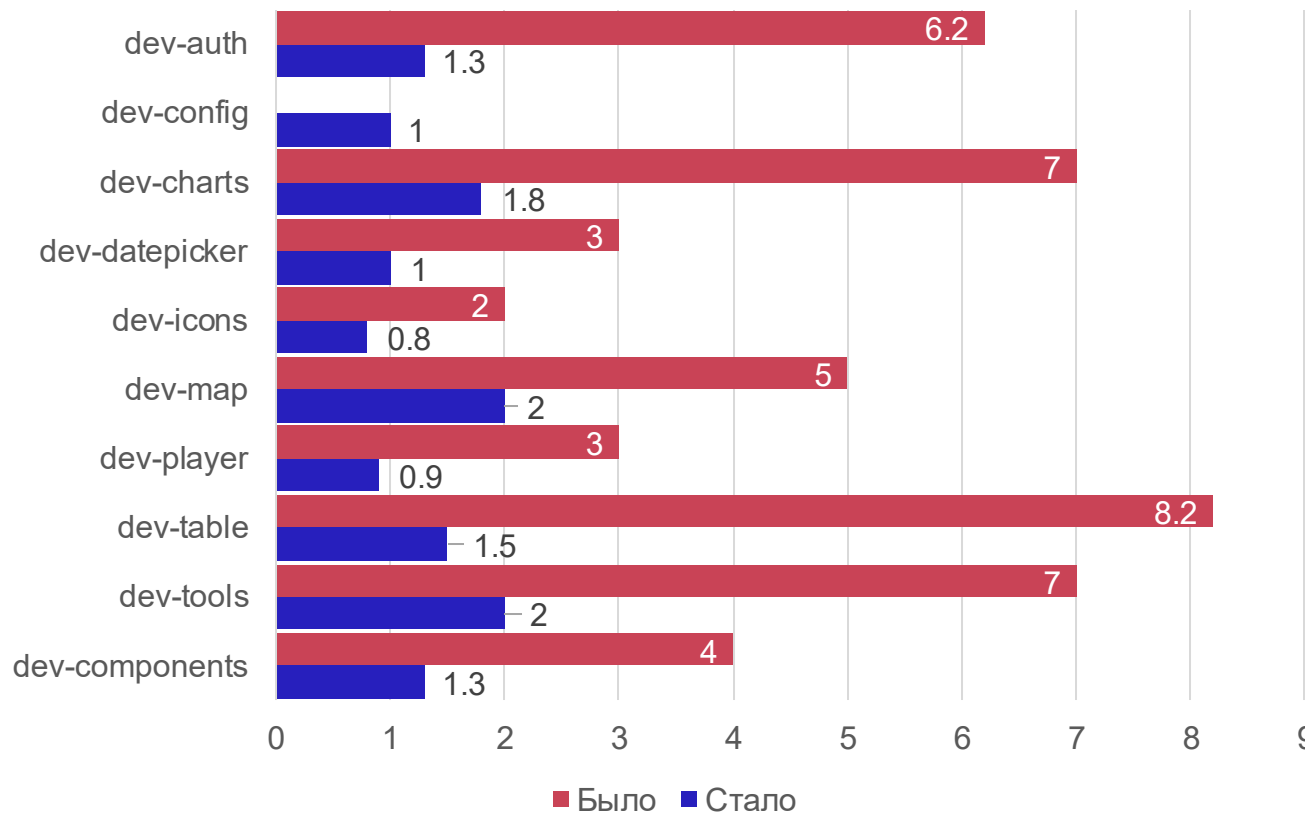
# ТЕКУЩИЙ РЕЗУЛЬТАТ



• ГОТОВЫ УВИДЕТЬ ПРОФИТ В ЦИФРАХ?

# PROFIT

## Время сборки библиотек



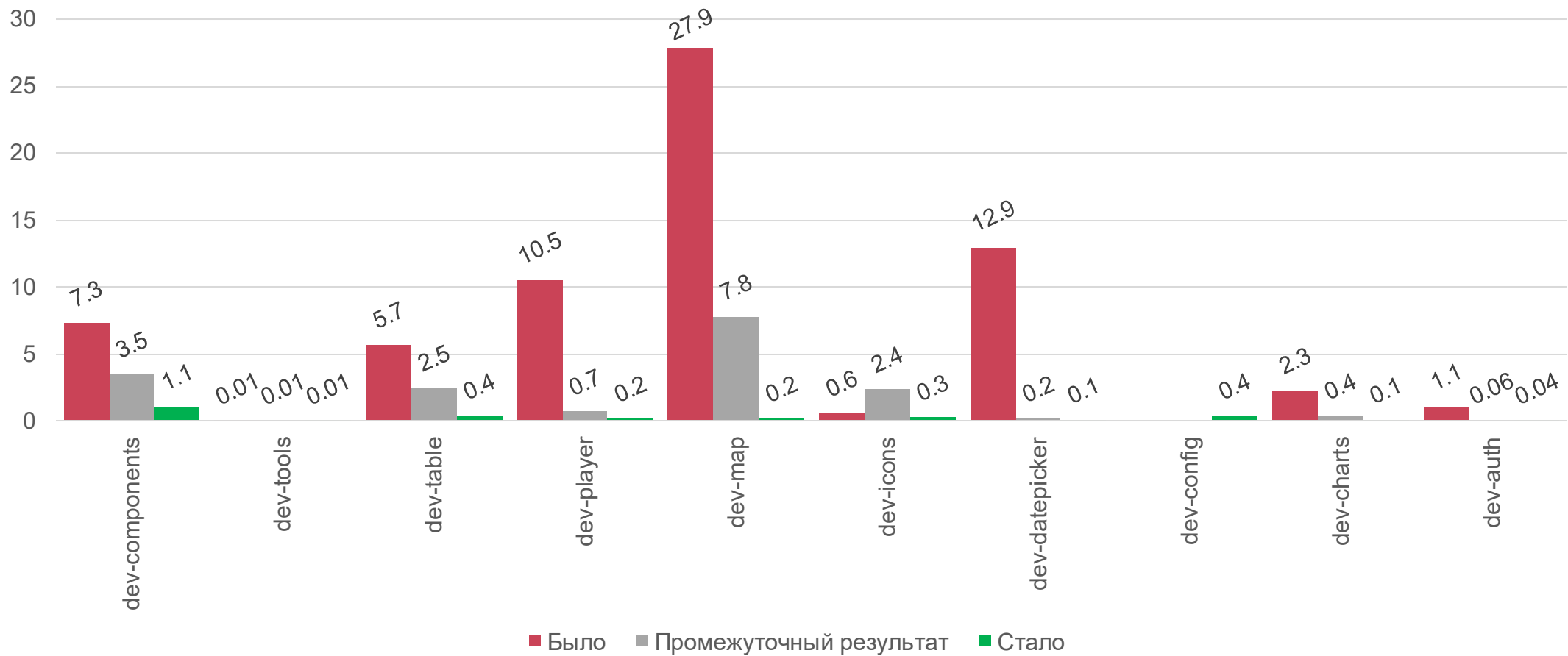
# 70.1%

Было: 45 мин 24 сек

**Стало: 13 мин 36 сек**

# PROFIT

## Размеры бандлов



# PROFIT В ЦИФРАХ



95.8%

Было: 68.3 Мб

Стало: 2.9 Мб



# PROFIT В ЦИФРАХ



## ВРЕМЯ СБОРКИ ПРОЕКТОВ

**47.4%**

Было: 12 МИН 36 сек\*

Стало: 6 мин 38 сек\*

**95.8%**

Было: 68.3 Мб

Стало: 2.9 Мб

\* По средним замерам скорости сборки проектов

# PROFIT В ЦИФРАХ



**ВРЕМЯ СБОРКИ  
ПРОЕКТОВ**

**47.4%**

Было: 12 МИН 36 сек\*  
Стало: 6 мин 38 сек\*

**95.8%**

Было: 68.3 Мб  
Стало: 2.9 Мб

**ВРЕМЯ СБОРКИ  
БИБЛИОТЕК**

**70.1%**

Было: 45 МИН 24 сек  
Стало: 13 мин 36 сек

\* По средним замерам скорости сборки проектов

# ЗАКЛЮЧЕНИЕ

# ЗАКЛЮЧЕНИЕ



- Относитесь к своим библиотекам, как к продуктам
- Не давайте им раздуваться до размеров вселенского масштаба
- И следите за импортами и зависимостями в библиотеках



**СПАСИБО ЗА ВНИМАНИЕ!**



**Василий Беляев**

Frontend Team Lead  
«Криптонит»

**Контакты:**

Канал: [https://t.me/devleader\\_pro](https://t.me/devleader_pro)

Телеграм: <https://t.me/devleader>



@DEVLEADER\_PRO



**kryptonite.ru**