

# FreeWheel在微服务架构下的 前端改造实践

宋一玮

FreeWheel基础架构部主任工程师



# AiCon

全球人工智能与机器学习技术大会

助力人工智能落地

2018.1.13 - 1.14 北京国际会议中心



扫描关注大会官网

# SPEAKER INTRODUCE



## 宋一玮

FreeWheel基础架构部主任工程师

毕业于北京理工大学，曾供职于IBM、Amazon以及一家O2O创业公司，现任FreeWheel基础架构部门主任工程师，负责FreeWheel自有前端框架SparkUI的设计研发和推广。

从最早的ASP、JSF、Flex、Dojo，一直到移动端、Angular，以及现在FreeWheel使用的React.js，从事前端开发已有10年。






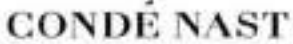










# TABLE OF CONTENTS 大纲

---

- 适用于微服务体系的前端SPA架构
- 自研前端框架SparkUI
- 新旧代码并存的渐进改造

# FreeWheel简介

## FreeWheel

# 适用于微服务体系的前端SPA架构

## FreeWheel前端应用现状

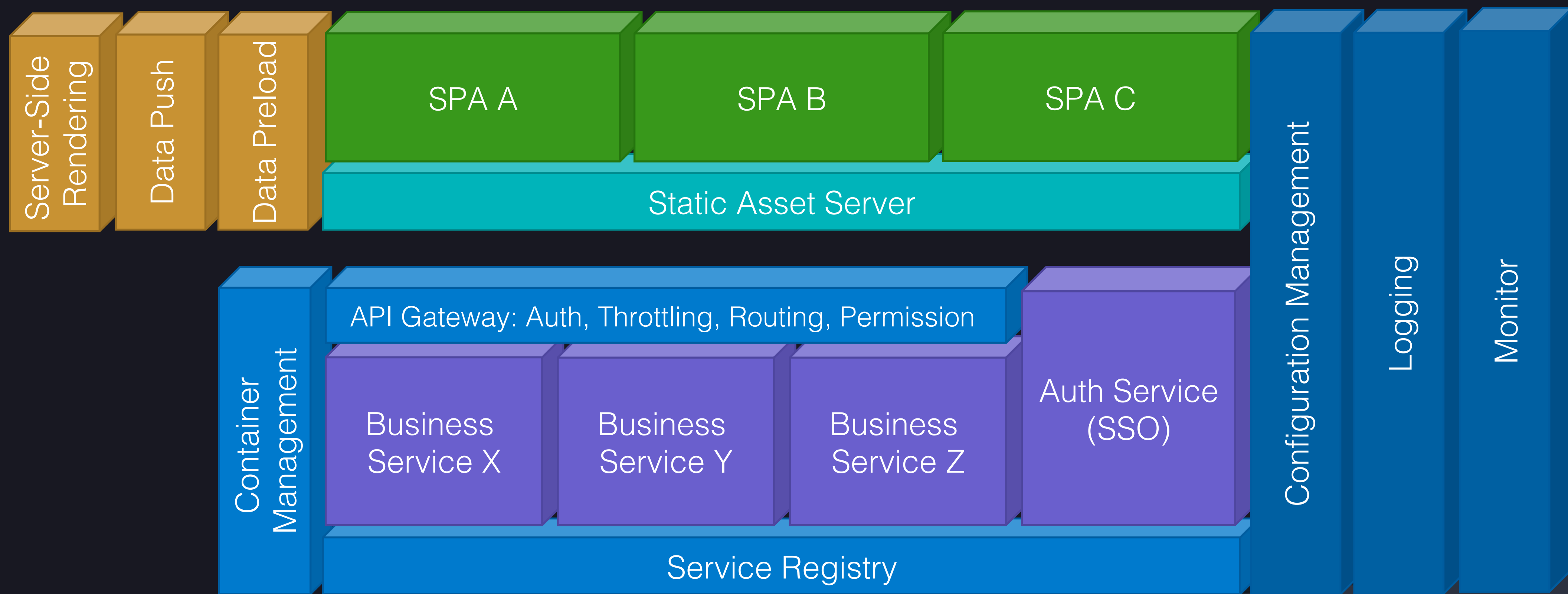
- 持续开发迭代了10年多的RoR Web应用
- 20多个产品模块，1200+页面
- 60余万行代码，其中包含近20万行基于jQuery的传统JS代码
- 7周的发版周期
- 全栈开发工程师以及QA工程师



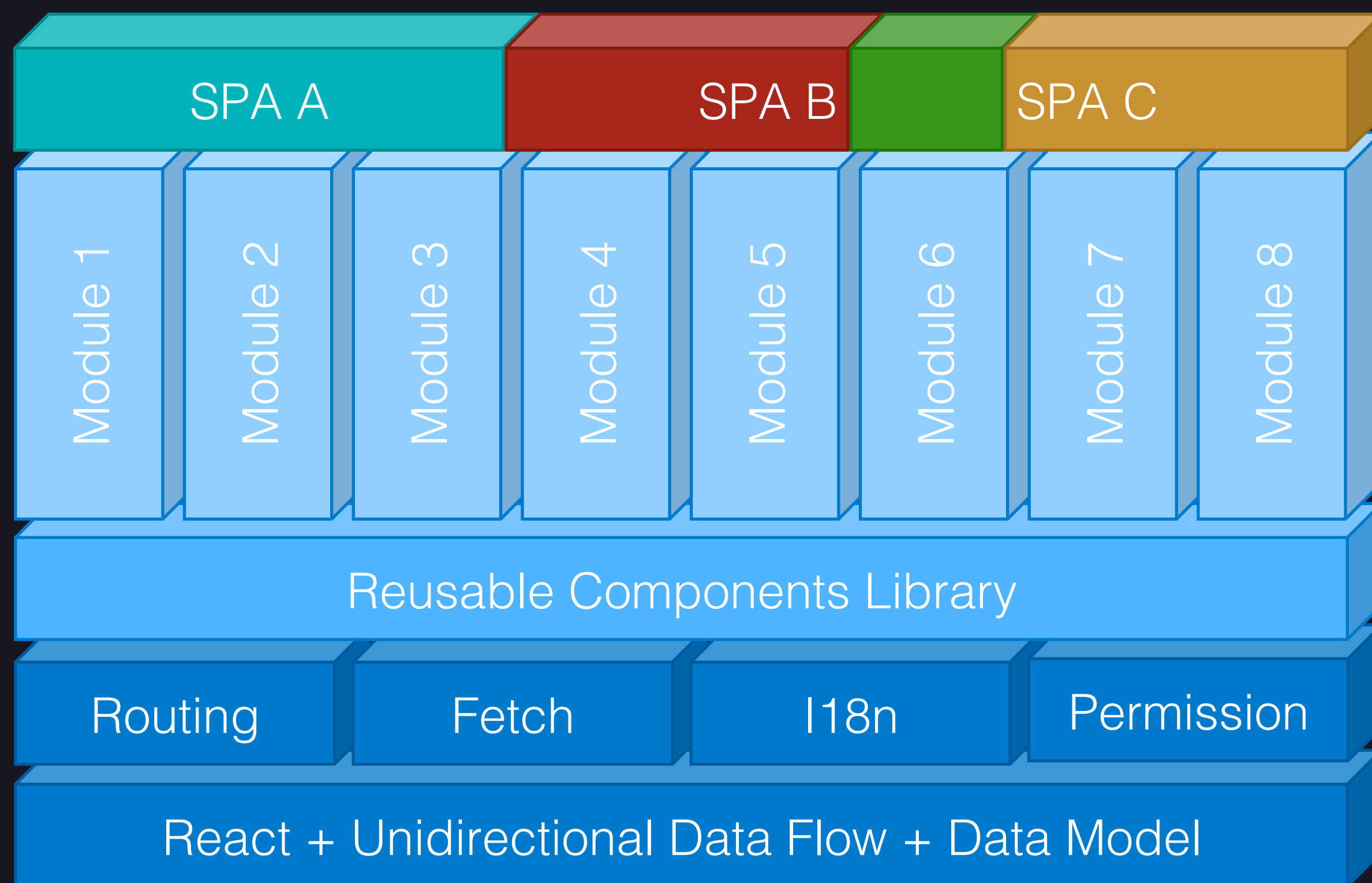
## 改造目标

- 前后端分离：后端微服务+前端SPA
- 产品与技术紧密结合，采用主流的前端体验和技术栈
- 更加完整的前端UT和端到端自动化测试覆盖
- 更加灵活的发版周期
- 全栈工程师

# 适用于微服务体系的前端SPA架构



# 适用于微服务体系的前端SPA架构

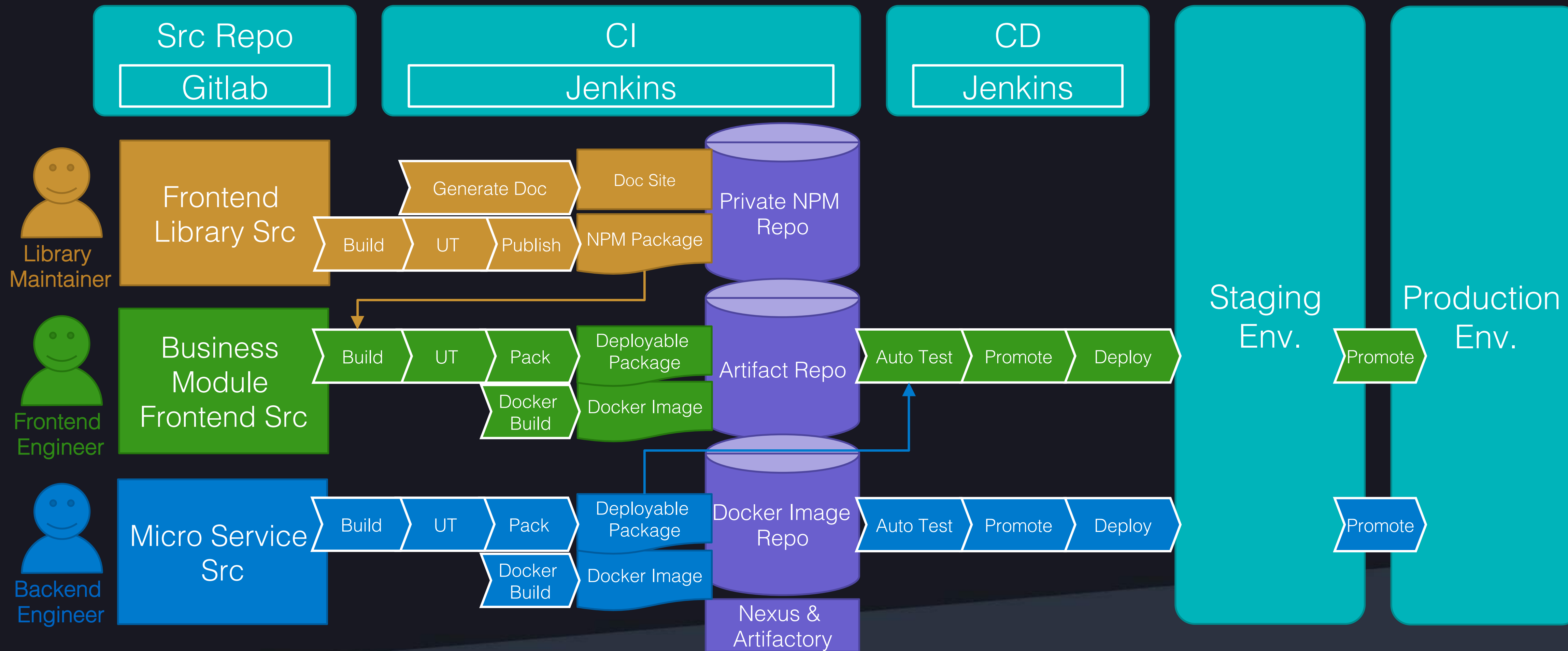




# 前端SPA架构 - 前端技术栈

	原有前端架构	新前端架构
服务器	Rails + Unicorn + Nginx	基于Nginx+CDN的静态资源服务器
SSR服务器端渲染	Rails MVC + ERB/HAML	Node.js + React SSR
浏览器端JS	jQuery jQueryUI Bootstrap Handlebars Underscore	React.js SparkUI Redux Immutable.js React-router Lodash
请求数据	jQuery.ajax + RESTful + JSON	Fetch API + RESTful + JSON
JS组件	基于jQuery定制开发	基于React定制开发
开发语言	Ruby + ES5	ES6/7 + JSX
打包工具	Rails Assets Pipeline	Webpack + Babel + Gulp
UT	Rspec	Mocha + Chai + Sinon + Enzyme
CSS	SCSS	CSS Modules + PostCSS

# 前端SPA架构 - 前端CI/CD



# 前端SPA架构 – 容器化

- 微服务容器化为前端开发测试提供良好的基础
  - 开发：UI-in-Docker工具 ( docker-compose )
  - 测试：DEP ( Dynamic Environment Provisioning ) 平台

# TABLE OF CONTENTS 大纲

---

- 适用于微服务体系的前端SPA架构
- 自研前端框架SparkUI
- 新旧代码并存的渐进改造

# 自研前端框架SparkUI介绍 – 面临挑战

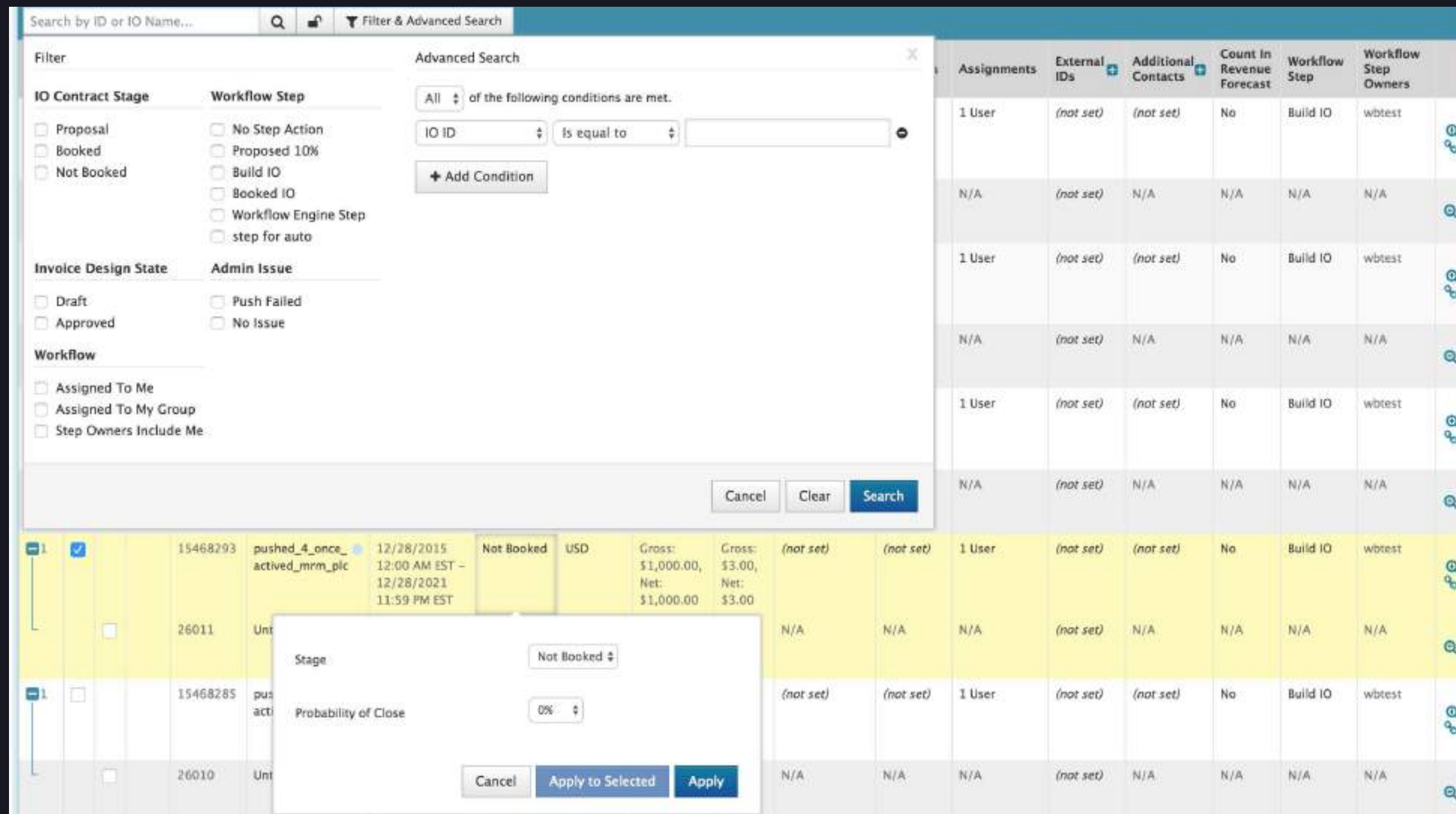
商业2B应用比起传统2C应用主要有如下特征：

- 软件质量标准高
- 复杂的业务模型和逻辑
- 跨功能模块的关联和交互
- 单一客户对需求的影响力强

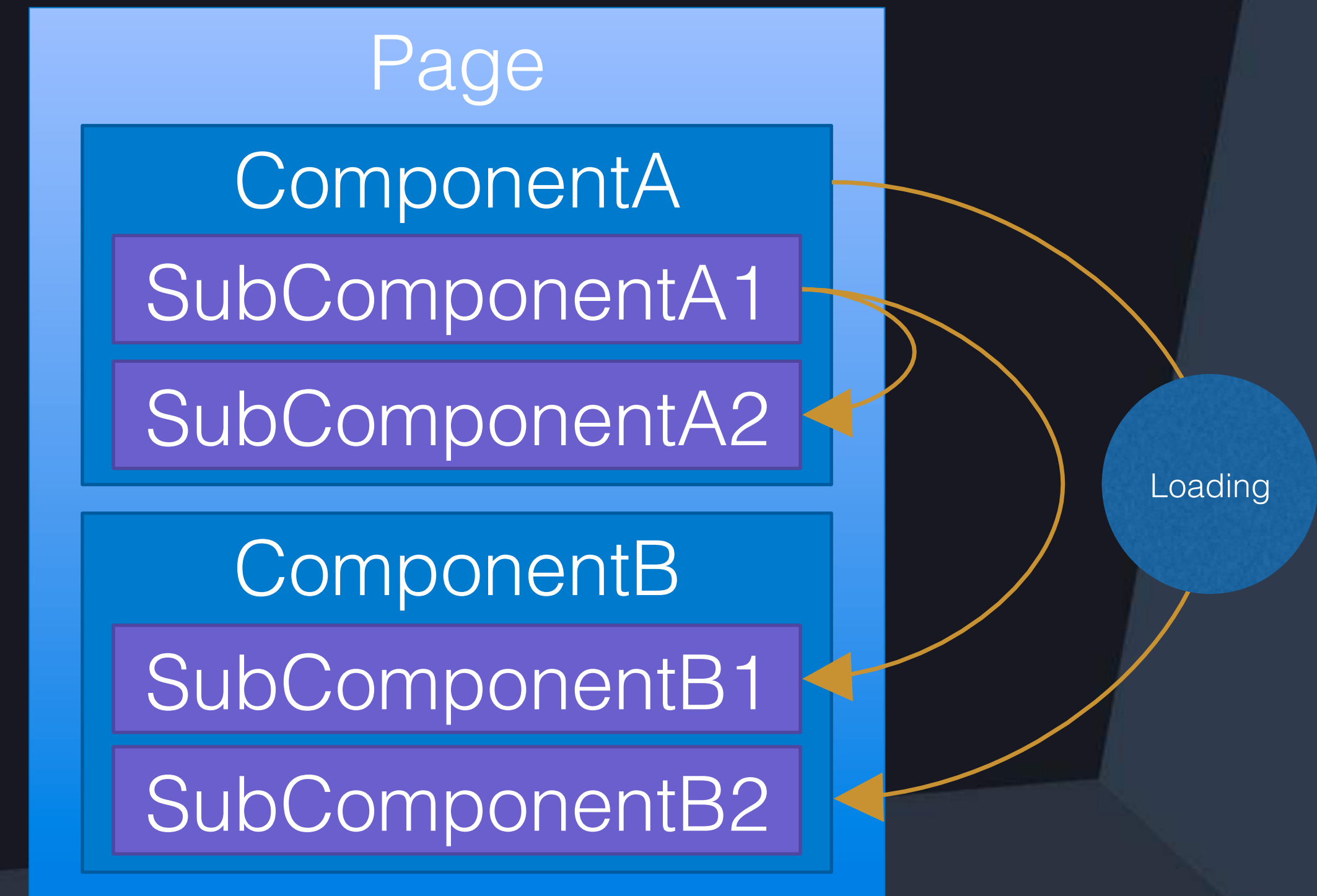


# 自研前端框架SparkUI介绍 - 面临挑战

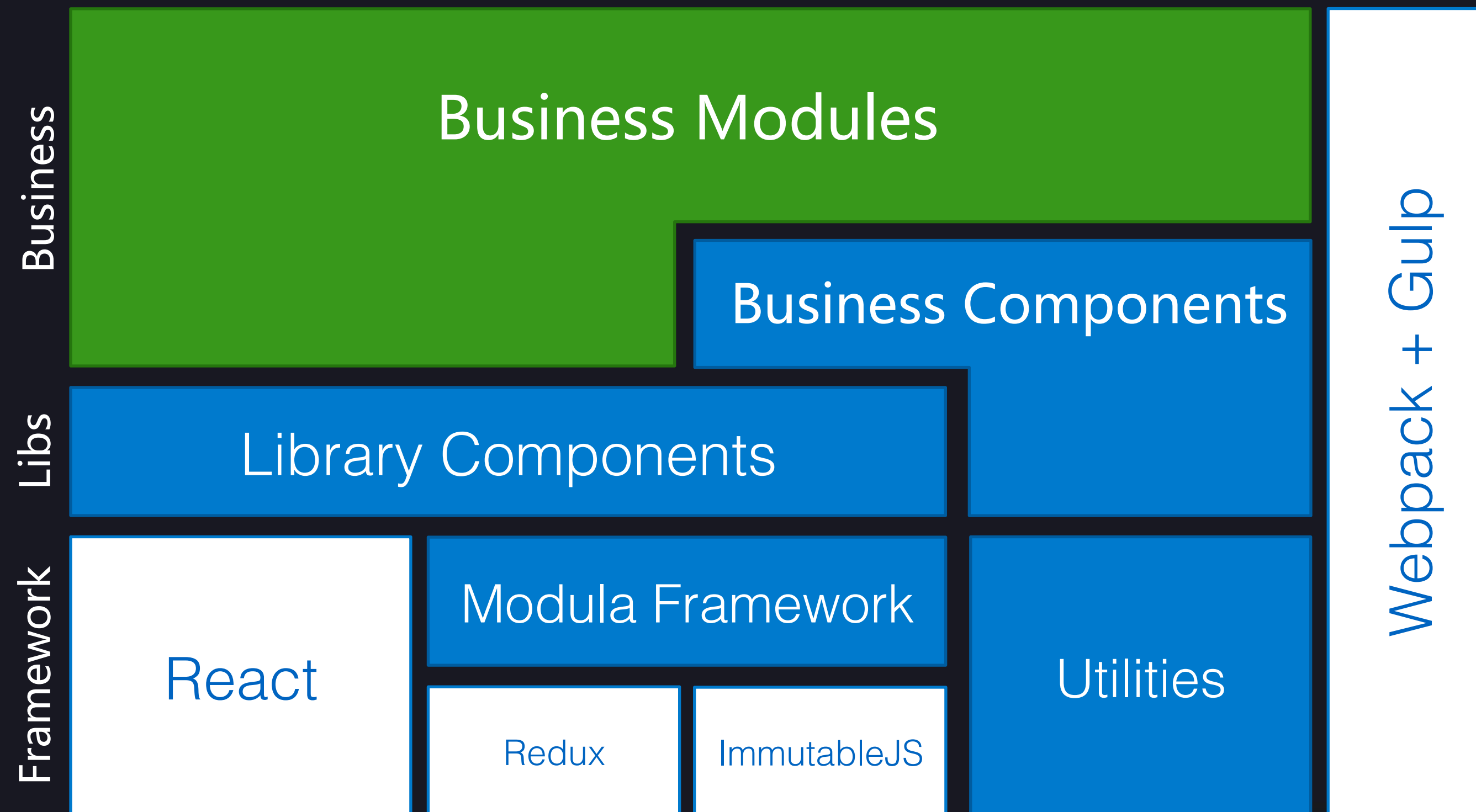
- 复杂的业务模型和逻辑



- 复杂的UI交互



# 自研前端框架SparkUI



\* Code Name: SparkUI

# 自研前端框架SparkUI

- 超过10万行框架及可复用组件代码，其中近4万为UT单元测试，UT覆盖率达到99.82%
- 40个子package，其中包含超过50个各类用途的可重用React组件
- 已有若干业务模块基于SparkUI完成重构或开发新功能



# 自研前端框架SparkUI - 可重用组件

**05 Date Range Picker**

Please select a date range

2017-07-04 - 2017-08-23

Jul 2017 Aug 2017

Last 7 Days  
Last 15 Days  
Most Recent Month  
The Second Quarter

en-US

Currency Field  
\$ 6,162,160

Numeric value: 6162160

Impression Field  
#,###

Numeric value:

Percent Field

Single Select  
Black

Multi Select  
x Turquoise x Blue

Red  
Green  
Black  
Xanadu  
Wine dark  
Really long option label (hover over c

SPARK

- Overview
- Spark Affix
- Spark Calendar
- Spark Debug
- Spark Editor
- Spark Elemental
- Spark Fetch
- Spark Gettext
- Spark Icon
- Spark Link
- Spark Loading
- Spark Messages
- Spark Modula
- Spark Modula Form
- Spark Navigation
- Spark On Click Outside
- Spark Raw Grid
  - 01 Grid Table Simple
  - 02 Grid Table Normal
  - 03 Grid Table Sophistic...
  - 04 Grid Sophisticated
  - 05 Grid Table Hierarchy...
- Spark Search
- Spark Smart Select
- Spark Table
- Spark Tooltip
- Fw Affix

**04 Grid Sophisticated**

NBA Player Database

Search By Player ID or Name 🔍 Filter & Advanced Search

	ID	Name	Age
+2	1	Michael Jordan	23
-2	2	Kobe Bryant	22
	1	Bulls vs Lakers	N/A
	4	Lakers vs Spurs	N/A
+1	3	Lebron James	21
+2	4	Yao Ming	19
+1	5	Tim Duncan	18

Basic

ID	4
Name	4

Select all results across all pages

03 Form In Widget

name (123)

Disable this value?

Name  
name

ID  
123

Cancel Clear Apply

**02 Overlay Loading**

Content is visible with OverlayLoading

It would provide better user experience for cases like reloading or updating

For years, LUMA has mapped the intermediaries of the Digital Media and Marketing ecosystem. These maps have always included Marketers, Agencies and Publishers, albeit in an aggregated way. Today, we map a larger complement to these critical players with the introduction of three new LUMAscapes: Marketers, Agencies and Publishers.

Start Loading End Loading

**01 Tooltip**

Text Tooltip HTML Tooltip Icon Tooltip

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In semper volutpat ultrices. Mauris lobortis lacus vel ullamcorper vestibulum.

Show Code

# React可重用组件设计要点

- 1. 无状态组件 ( Stateless Component ) 优于状态化组件 ( Stateful Component )

```
<LinkGroup
  · label="More"
  · expanded={ model.get('linkGroupExpanded') }
  · onClick={ model.sendLinkGroupToggle }
>
  · { /* <Link>s */ }
</LinkGroup>
```

# React可重用组件设计要点

- 2. 组合组件 ( Composing Components ) 优于具有DSL ( Domain Specific Language ) 属性的单一组件

```
<Tooltip content={ <p> I am a <strong>tooltip</strong></p> }>
| ··<div>My Content</div>
</Tooltip>

// v.s.

<Label tooltip="I am a tooltip" tooltipFontWeight="bold">My Content</Label>
```

# React可重用组件设计要点

- 3. 高阶组件 ( HOC , Higher-Order Components ) 优于混合属性 ( Mixins )

```
const EnhancedCheckbox = recompose.pure(Checkbox);  
  
// v.s.  
  
<Checkbox label="Subscribe to news" { ...subscribe.getCheckboxProps() } />
```

# 自研前端框架SparkUI – 应用状态管理

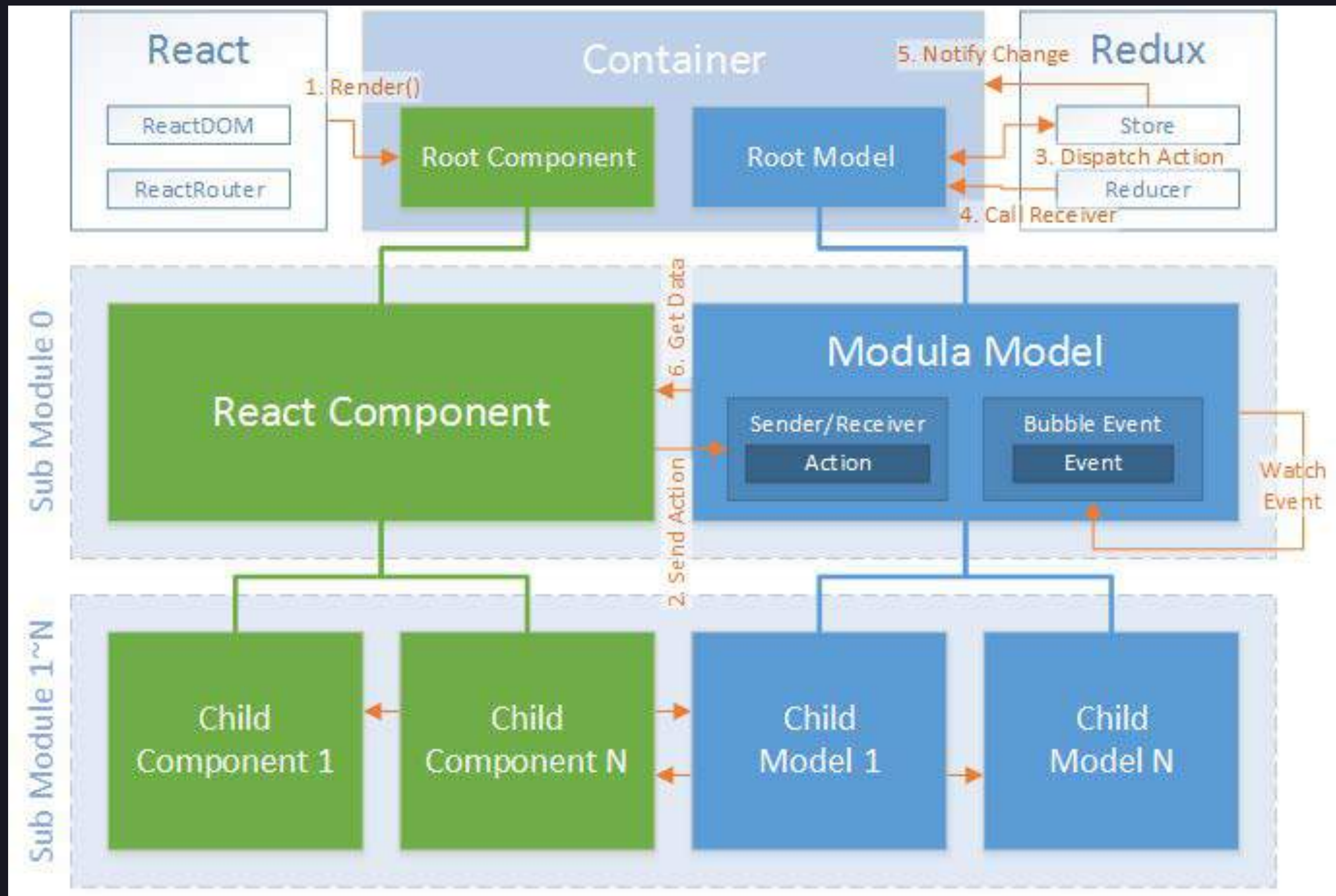
	React Only	Flux	Redux
优点	<ul style="list-style-type: none"><li>• 不依赖其他框架</li></ul>	<ul style="list-style-type: none"><li>• 将state抽取为store, 并以action方式改变store, 单向数据流从React中分离出来</li></ul>	<ul style="list-style-type: none"><li>• 单一store, single source of truth;</li><li>• 生态丰富</li></ul>
缺点	<ul style="list-style-type: none"><li>• 需要将深层组件的state lift up, 导致顶层组件中的state越来越复杂、难以维护</li></ul>	<ul style="list-style-type: none"><li>• 一个应用内存在多个store, 如果在store之间建立关联, 将导致store难于维护;</li><li>• waitFor接口在实际应用中会带来较多问题</li></ul>	<ul style="list-style-type: none"><li>• 对于大型应用, store与业务数据差别较大</li><li>• State tree节点间的关联较难实现</li><li>• 流行的处理Side Effects的方案都不够直接</li></ul>

# 自研前端框架SparkUI – 应用状态管理

## Modula应用状态管理框架设计理念

- **Application State = Initial State + Deltas** , 其中 Delta 是由 Actions 触发的 [ 借鉴Flux, Elm ]
- **Application State** 可以由一棵 **Model Tree** 来描述, 这棵树的每个节点都是一个可以描述有效业务实体的 **Model** [ 借鉴Redux, Elm ]
- 由一个给定的 **Application State** 到另一个State的 **Transition** 可以由 **Model Tree** 提供的 **Reactions** 所描述, 一次成功的 **Action** 到 **Reaction** 的匹配会将 **Model Tree** 演变为下一个状态 [ 原创 ]
- **Side Effect** 是上述 **state transitions** 的结果, 它包含了一个更新的model实例, 以及0至多个 **callback functions** [ 借鉴Elm ]

# 自研前端框架SparkUI - 应用状态管理



```
const EmployeeModel = createModel({
  propTypes: {
    id: PropTypes.string,
    name: PropTypes.string,
    age: PropTypes.number,
    todoList: ImmutablePropTypes.listOf(PropTypes.instanceOf(TodoModel))
  },
  defaults: {
    id: null,
    name: 'Employee',
    age: 0,
    todoList: () => new List()
  },
  sendAddTodo() {
    this.dispatch({ type: ActionTypes.ADD_TODO });
  },
  recvAddTodo() {
    return {
      type: ActionTypes.ADD_TODO,
      update(model, action) {
        const newModel = model.set('todoList', todoList => (
          todoList.add(new TodoModel())
        ));
        return [ newModel ];
      }
    };
  }
});
```

# 自研前端框架SparkUI – 前端路由

- 曾封装并使用react-router
  - 遇到的问题：应用状态分散在react-router的state与Modula Model（Redux state）里，两者经常有同步问题
  - 解决思路：将路由相关的state也合并进Modula Model中
- 前端路由框架spark-router
  - 针对Model配置路由，Component根据Model切换相应界面
  - Functional函数式配置，可测试性良好
  - 支持类似react-router v4中的multi-match功能
  - 可与react-router共存

```
const advertisingRoute =
  redirectRoute(
    [
      { path: '/', redirect: '/campaigns' }
    ],
    contextSwitchRoute(AdvertisingModel, [
      { path: '/campaigns', propName: 'campaigns', route: simpleRoute(CampaignsModel) },
      { path: '/advertisers', propName: 'advertisers', route: simpleRoute(AdvertisersModel) },
      { path: '/agencies', propName: 'agencies', route: booleanRoute }
    ])
  );
```

```
const AdvertisingComponent = ({ model }) => (
  <div>
    <Tab>
      <TabItem href="/advertising/campaigns" active={ model.get('campaigns') }>
        Campaigns
      </TabItem>
      <TabItem href="/advertising/advertisers" active={ model.get('advertisers') }>
        Advertisers
      </TabItem>
      <TabItem href="/advertising/agencies" active={ model.get('agencies') }>
        Agencies
      </TabItem>
    </Tab>
    { model.get('campaigns') && <CampaignsComponent model={ model.get('campaigns') } /> }
    { model.get('advertisers') && <AdvertisersComponent model={ model.get('advertisers') } /> }
    { model.get('agencies') && <AgenciesComponent model={ model.get('agencies') } /> }
  </div>
);
```



# TABLE OF CONTENTS 大纲

---

- 适用于微服务体系的前端SPA架构
- 自研前端框架SparkUI
- 新旧代码并存的渐进改造

# 新旧代码并存的渐进改造



# 新旧代码并存的渐进改造



# 新旧代码并存的渐进改造 – 混合工程结构

- SPA前端代码的源码依旧放在Rails工程Module目录下
- 通过Webpack打包的bundle JS / CSS也按照Module对资源文件的约定 ( convention ) 放在 `modules/my_module/app/assets/javascripts/my_module/compiled` 目录下
- 藉由Rails Asset Pipeline打包进Rails工程发布包进行统一部署
- 仍使用Rails页面模版作为入口

# 新旧代码并存的渐进改造 - 混合工程结构

```
<%= javascript_include_tag "my_module/compiled/my_module" %>
<%= @js_module_alias = "my_module" %>
<div id="spa"></div>
<script>
  (function() {
    var React = require('react');
    var ReactDOM = require('react-dom');

    var AppContainer = require('<%= @js_module_alias %>').AppContainer;

    ReactDOM.render(
      React.createElement(AppContainer),
      document.getElementById('spa')
    );
  })();
</script>
```

- Rails的后端（页面）路由委托给前端

```
scope 'spa' do
  get '/', :to => 'spa#index', :as => 'spa'
  get '*pages', :to => 'spa#index'
end
```

# 新旧代码并存的渐进改造 – 独立组件库工程

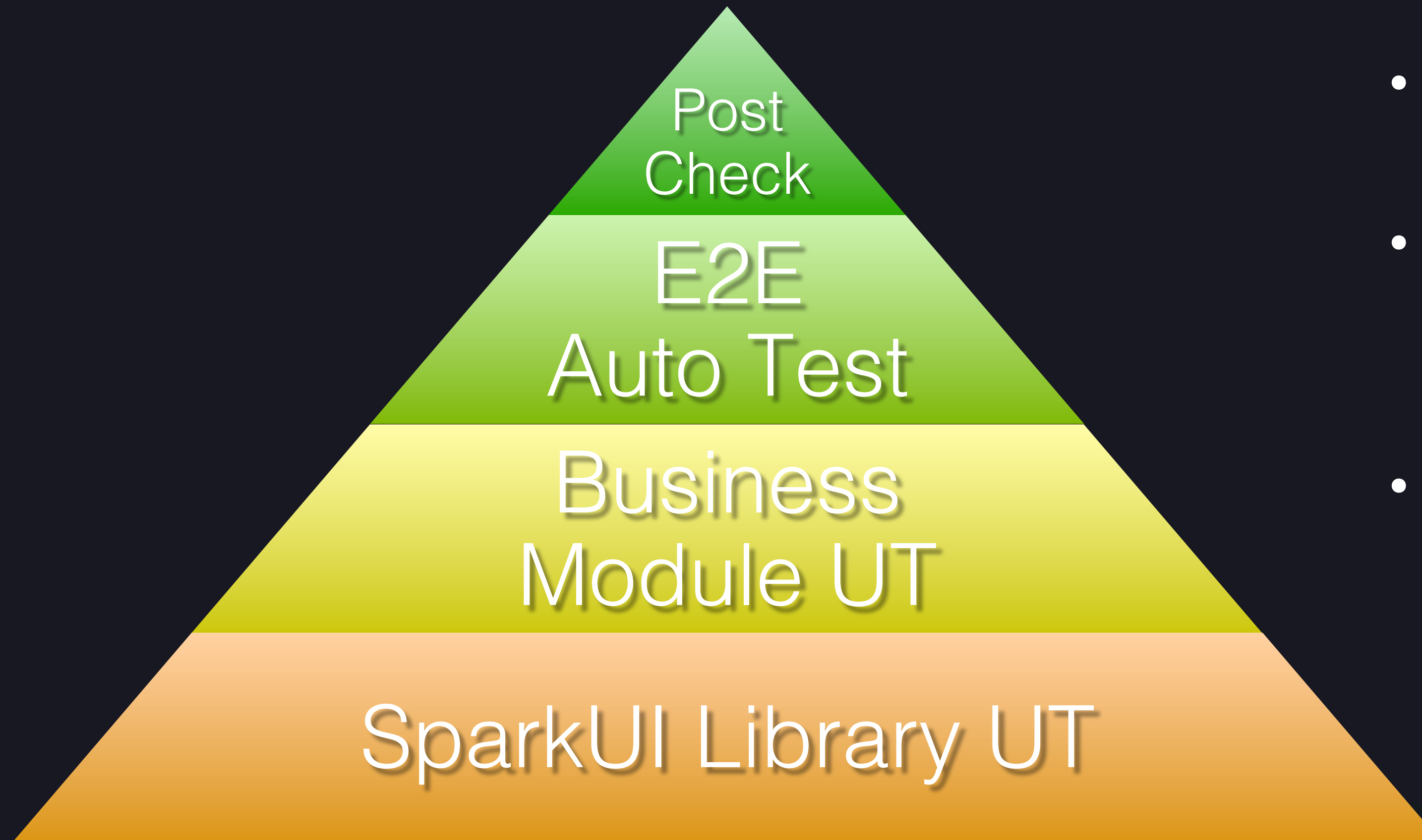
- 将SparkUI剥离为独立纯前端工程；
- 利用Webpack打包并发布到公司Nexus上私有NPM Repository里；
- 其他工程利用npm install安装依赖。

	原有混合工程	独立工程
版本管理	任何对SparkUI的迭代都会直接影响到业务模块	业务模块代码可以更有计划地升级SparkUI版本，在此之前无须反复回归测试
开发效率	SparkUI是纯JS库，Rails工程开发环境给SparkUI开发带来一定负担	不同的发版节奏令SparkUI可以追逐更高的代码质量，目前其源代码已超10万行，单元测试覆盖率高达99.82%
源码权限	任何业务模块开发人员均可修改SparkUI代码，带来潜在代码冲突	独立的代码库可以隐藏部分SparkUI的内部API或工具代码，防止业务模块中滥用
跨工程复用	任何Rails工程之外的工程在利用SparkUI时都会比较繁琐。	作为标准NPM包复用

# 新旧代码并存的渐进改造 – 新老JS代码混用

- 在SparkUI中开发了一个AdapterComponent
- 统一封装基于jQuery的老JS接口，利用componentDidMount完成其初始化
- 严格控制新老代码在运行时的边界

# 新旧代码并存的渐进改造 – 质量保证



- Capybara
- Cucumber + Selenium  
Cypress
- JSDOM  
Mocha + Chai + Sinon  
Enzyme  
Babel-istanbul  
SparkUI Test Utility



# 新旧代码并存的渐进改造 – 灰度上线

- 为改写或重构的模块入口设置切换开关，On则使用新代码，Off则使用老代码
- 统一的上线包，开关值设在CM中
- 提高测试覆盖率，做好回归测试
- 上线后密切关注监控
- 发现线上问题时一键rollback回老代码



# 产品迭代与技术演进的平衡

- 公司上下统一认识，认可技术演进带来的长期利益；
- 定立产品研发计划时为技术演进工作保留一定比例的资源；
- 设立专门团队专注架构演进、基础设施设计开发；
- 持续提高自动化测试覆盖率。

# 广告时间

- SparkUI框架开源进行时！
- 项目及其代码已通过FreeWheel母公司Comcast审查流程
- 正式名称将采用“modulajs”，部分包将于近期开源在Github上
- Comcast开源项目首页：<https://github.com/Comcast>
- 届时请大家多关注，多交流

# 总结

- 微服务体系下，如何设计实施基于SPA的前端架构
- 即将开源的SparkUI，是一套适用于商业SPA应用的前端框架
- 以渐进改造的方式，兼顾产品迭代和技术演进

# THANK YOU

---

如有需求，欢迎至 [ 讲师交流会议室 ] 与我们的讲师进一步交流

