基于 Electron 的跨平台桌面客户端开发实践

王丰

字节跳动 飞书前端团队

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

TGO鲲鹏会

汇聚全球科技领导者的高端社群

全球12大城市

850+高端科技领导者

使命 Mission

为社会输送更多优秀的科技领导者

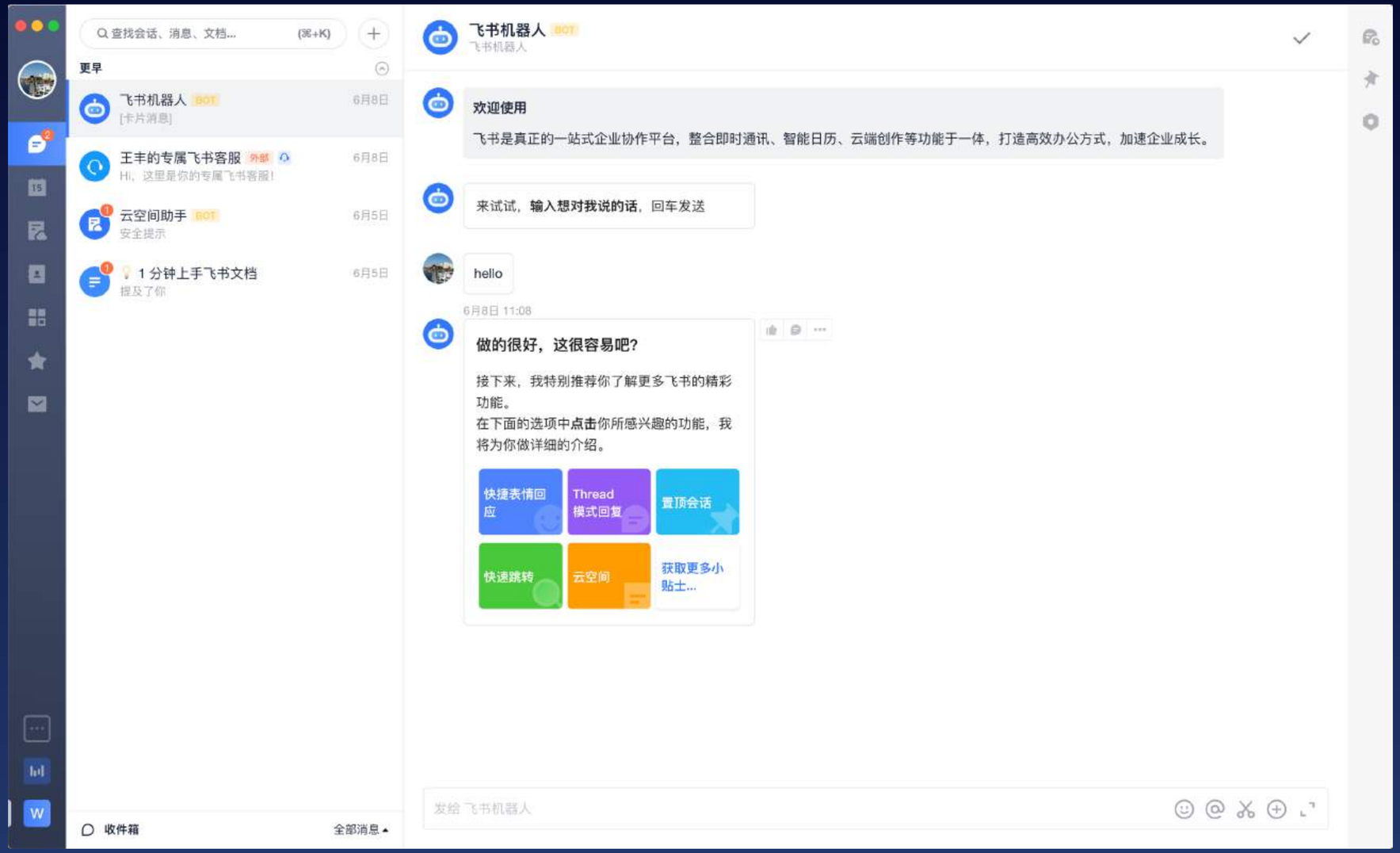
愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

目录

飞书（一站式企业协作平台）



WebApp 的可行性

成熟的参考产品

成熟的开发模式

WebApp 的可行性

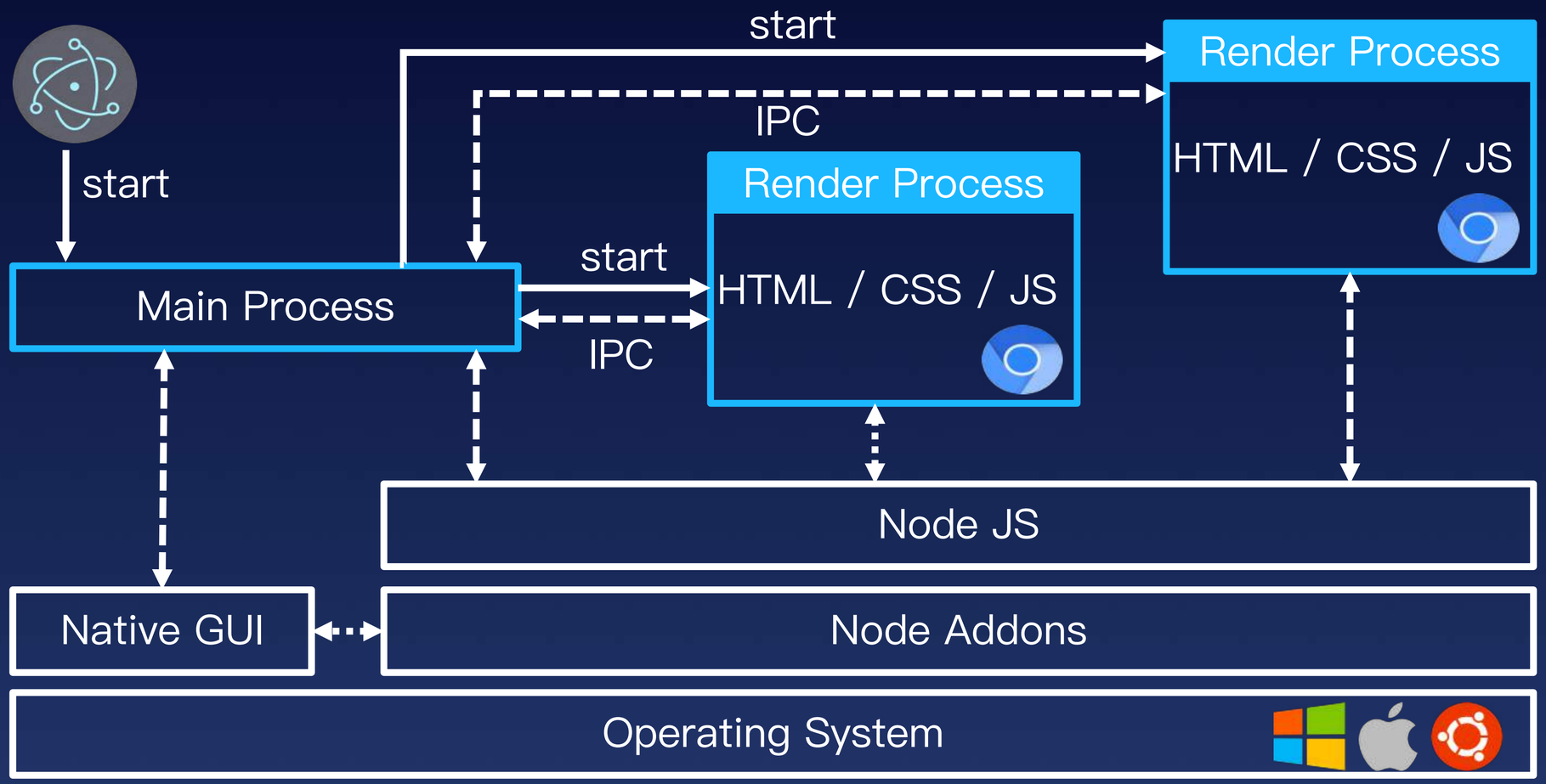
Native Features 如何处理?

基于 Chromium 定制 WebApp Runtime

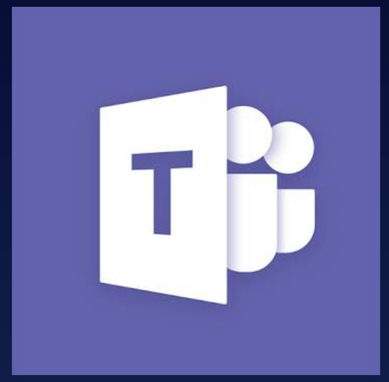
离线使用
消息通知
音频通话
系统菜单
屏幕截图
视频通话
快捷访问
图片粘贴
文件读写

+

基于 Electron 的 WebApp Runtime



Electron 的可行性

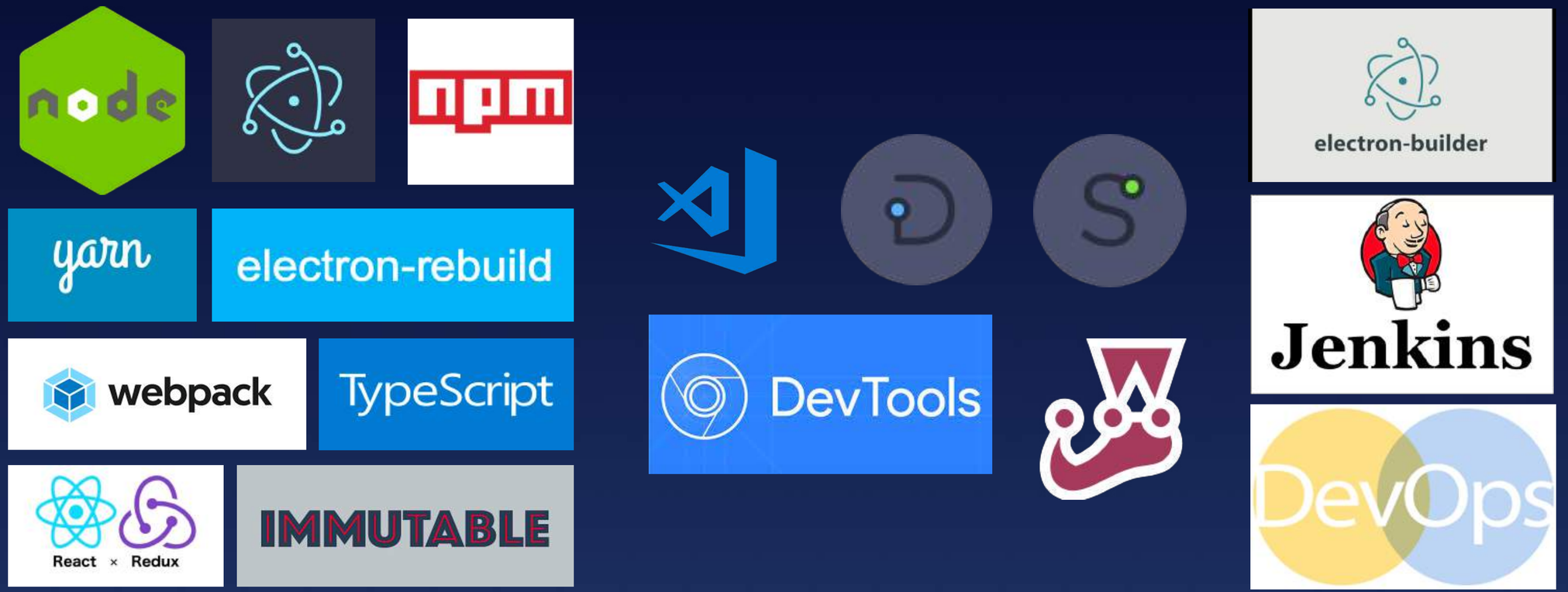


总结（飞书为什么选择 Electron）

1. 飞书主业务功能 WebApp 可行，但缺少 Native 特性
2. Electron 能为 WebApp 定制环境，增加 Native 特性
3. 实践中 VSCode、Slack 等验证了 Electron 的可行性

目录

飞书的工程化建设

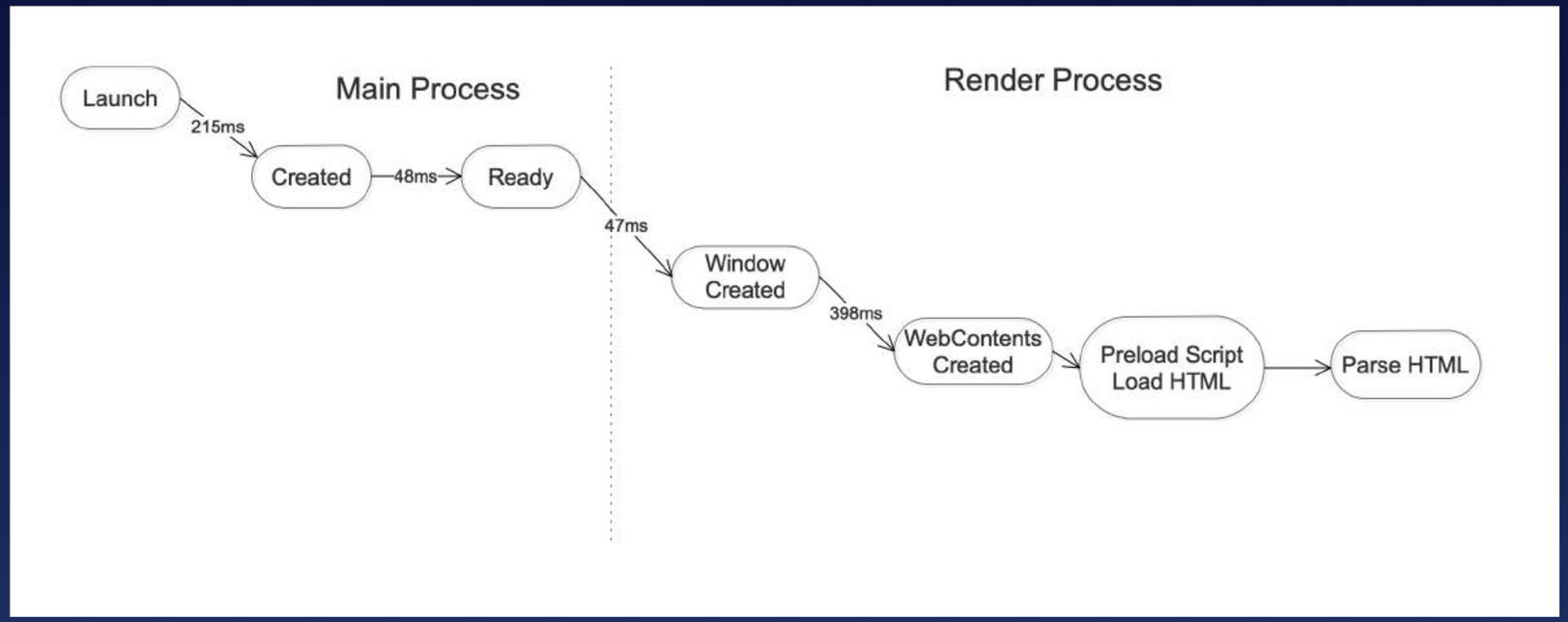

node
npm
yarn
electron-rebuild
webpack
TypeScript
React × Redux
IMMUTABLE
DevTools
electron-builder
Jenkins
DevOps

Electron 相关的技术挑战

快速启动

如何提升 Electron App 的启动速度?

Electron App 的启动流程



1. 存在约1s不可抗耗时
2. 存在 > 400ms 白屏
3. 窗口创建、显示可控
4. preload代码逻辑可控

Electron App 的启动优化

1. 提前创建、加载主窗口
2. 主窗口创建期间，本地化 WebApp 数据
3. Webapo启动界面显示后，再显示窗口
4. 减少 Preload 执行时间
5. 提升 WebApp 启动速度

```
// main process
ipcMain.on('main-window-splash-screen-show', () => {
  mainWindow.show()
})

// main window
splashScreen.show()
requestAnimationFrame(() => {
  ipcRenderer.send('main-window-splash-screen-show')
})
```

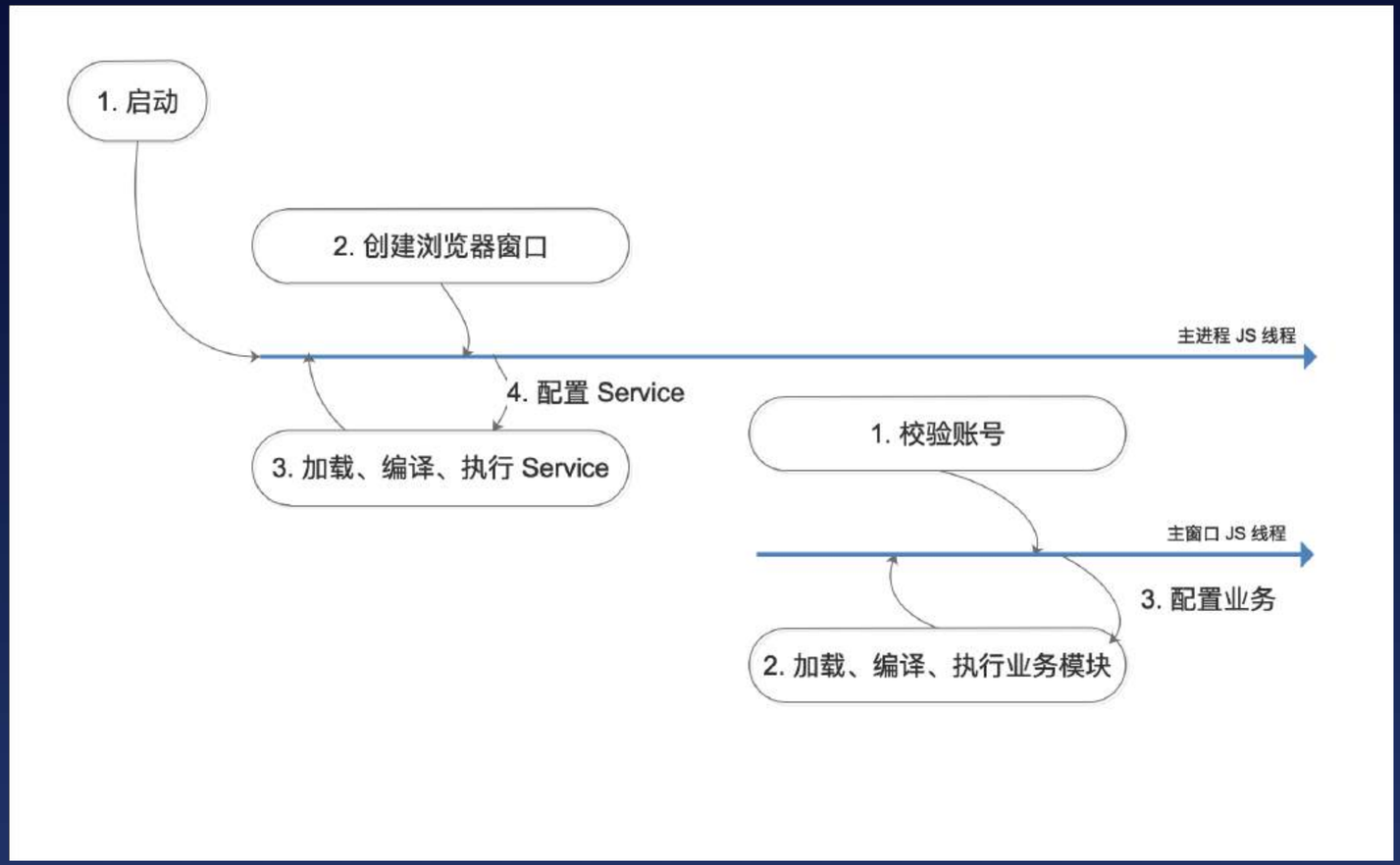
复杂 Electron App 的特点

1. 代码量大，编译执行耗时，需要大量的 CPU 资源
2. 依赖层级多，容易串行等待，CPU 资源利用不充分

复杂 Electron App 启动优化的目标问题

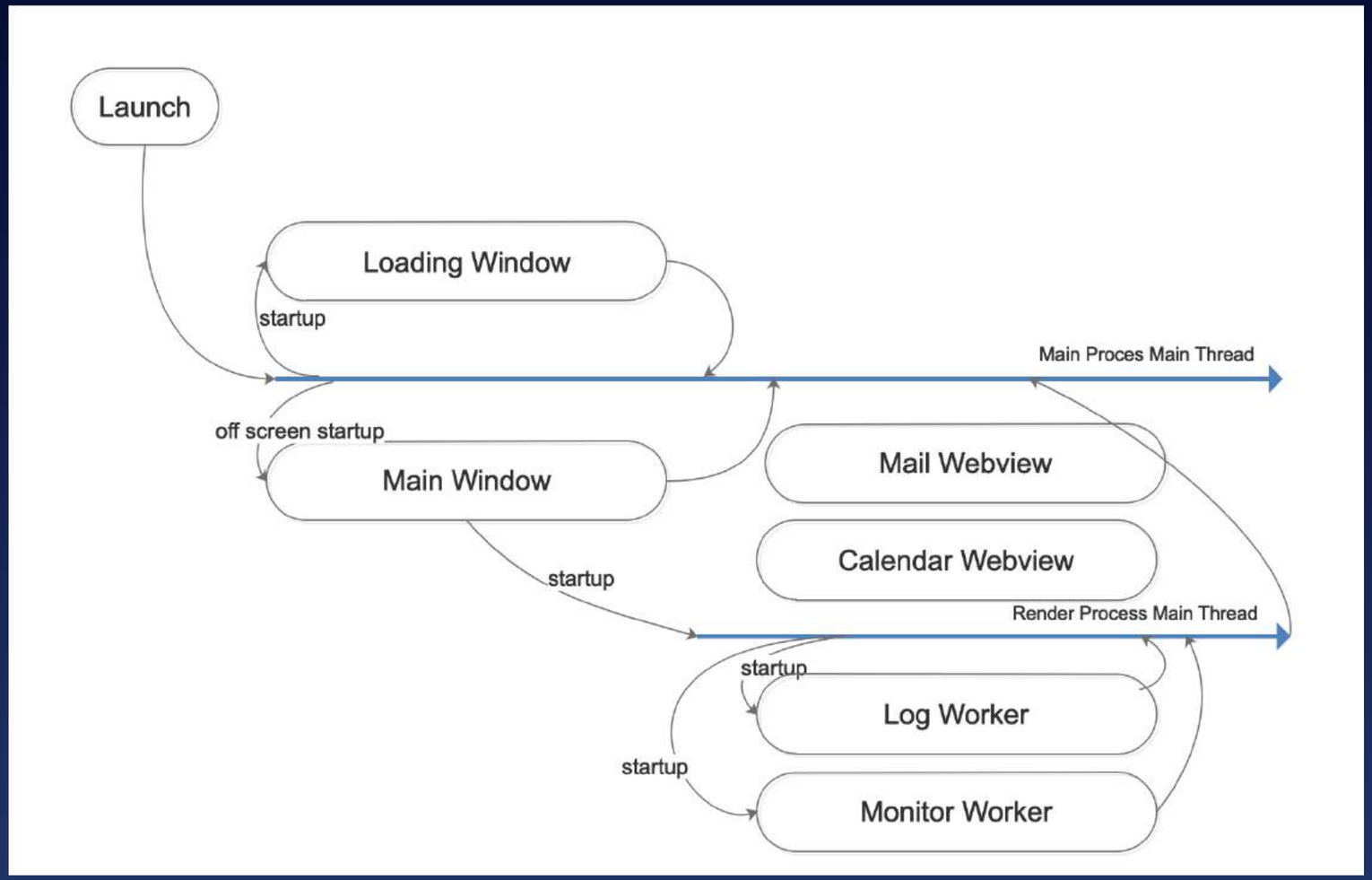
1. 如何充分利用单个线程，消除空闲
2. 如何突破单个线程限制，并行执行
3. 如何找到延迟加载时机，减少执行

如何充分利用单个线程，消除空闲



1. 异步调用期间，尽量多的加载、编译、执行模块
2. 异步调用成功后，初始化模块

如何突破单个线程限制，并行执行



1. BrowserWindow
2. Web Worker
3. Webview
4. BrowserView
5. ChildProcess

如何找到延迟加载时机，减少执行

1. request　Idle　Callback
2. Event: 'browser–window–blur'

WebContents 类组件启动优化

1. 空间换时间
2. 组件预创建
3. 页面预加载
4. 关闭后缓存

总结（快速启动）

1. 浏览器窗口创建、加载前置
2. WebApp 数据预加载
3. 减少 Preload 脚本的耗时
4. 启动界面渲染后再显示窗口
5. 消除空闲、并行启动、减少执行
6. WebContents 类组件使用空间换时间策略

内存管理

如何管理和优化 Electron App 的内存占用?

Electron App 内存占用大引发的问题

1. 操作系统强杀
2. 应用切换卡顿
3. 用户负面评价

Electron App 内存管理的目标问题

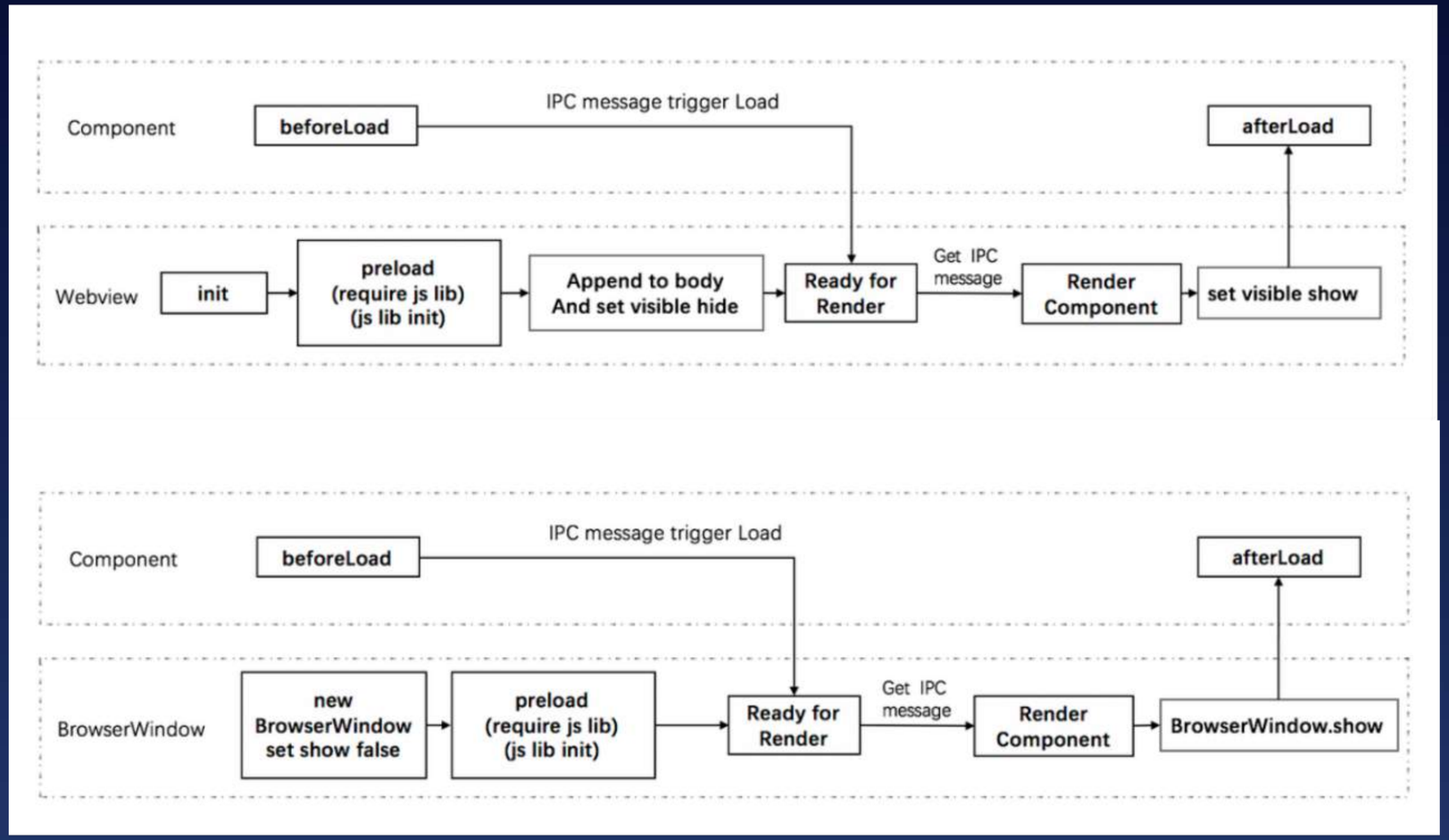
1. Chromium 内存策略激进，且 WebApp 对内存可控性很低
2. 优化 WebContents 类组件时，低性价比的空间换时间策略
3. 业务框架、业务代码中的内存泄漏
4. 缺乏与用户侧一致的内存统计指标

提升 WebApp 对内存的可控性

```
void WebFrame::ClearCache(v8::Isolate* isolate) {
  isolate->IdleNotificationDeadline(0.5);
  blink::WebCache::Clear();
  base::MemoryPressureListener::NotifyMemoryPressure(
      base::MemoryPressureListener::MEMORY_PRESSURE_LEVEL_CRITICAL);
}
```

1. 缓存清理
2. 压力警告

高性价比的空间换时间



1. 预创建，20M 换取 400ms
2. 预加载，40M 换取 1s
3. 缓存，40M 换取 1s

解决内存泄漏

1. EventBus
2. ReactFiber

```
ReactDOM.render(
  <EventBusProvider provider=
    
    {
  </EventBusProvider>, document.ge
);
export default withEventBus
```

统一内存统计指标

```
#if !defined(OS_LINUX)
    uint64_t phys_footprint = 0;
#if defined(OS_MACOSX)
    phys_footprint =
        process_metric.second->metrics->GetTaskVMInfo().
phys_footprint >> 10;
#else
    base::WorkingSetKBytes ws_usage;
    if (process_metric.second->metrics->GetWorkingSetKBytes(&
ws_usage)) {
      phys_footprint = ws_usage.priv;
    }
#endif
    memory_dict.Set("footprint", static_cast<double>(phys_footprint
));
#endif
```

1. app.getAppMetrics
2. process.memoryUsage
3. 以上取值与 TaskManager、ActivityMonitor 相差较大
4. Windows Private Working Set
5. Mac Physical footprint

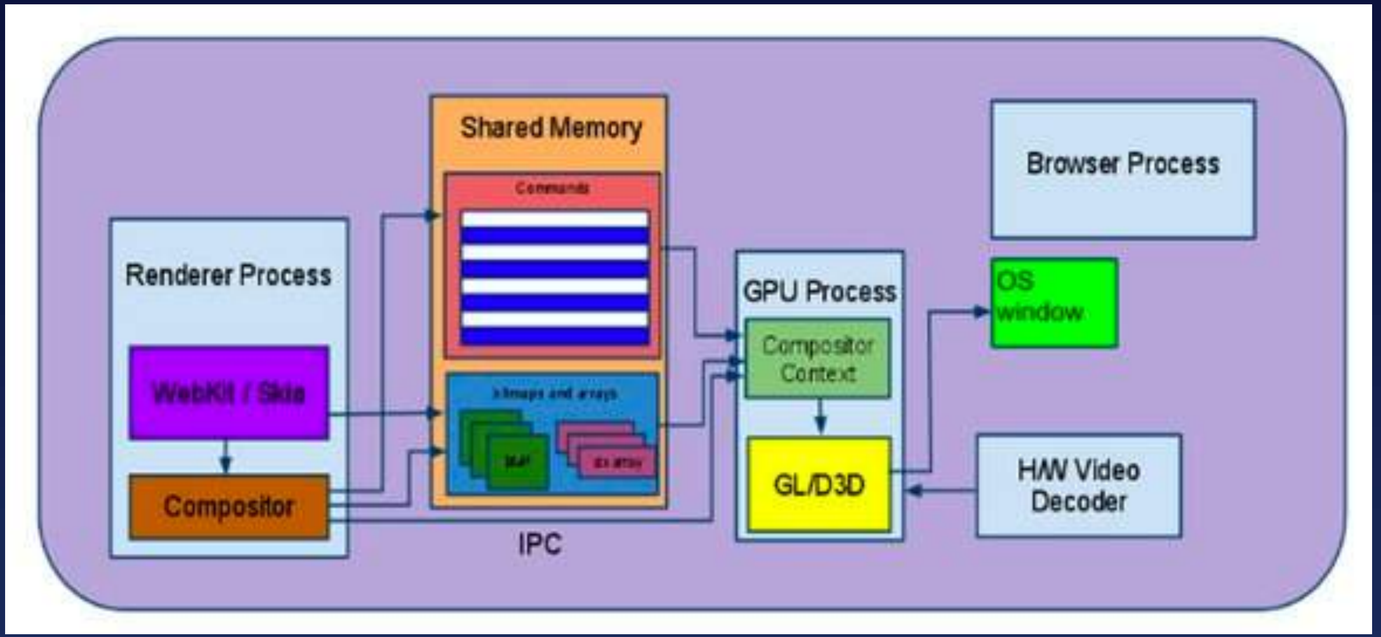
CPU 异常处理

如何处理 Electron App 的 CPU 异常?

Electron App 进程间 CPU 异常会互相影响

1. 主进程异常时，渲染进程发生 UI Block
2. 渲染进程异常时，有可能会影响主进程，引起其他渲染进程 UI Block

Electron App 进程间 CPU 异常互相影响的根源



1. 渲染进程间共用工具进程
2. 渲染进程对工具进程的调用通过主进程做 IPC 转发
3. 主进程忙碌时，IPC 被阻塞，导致渲染被阻塞

解决 Electron App 的进程间 CPU 异常影响

1. 主进程逻辑轻量化，CPU 密集性任务使用子进程处理
2. 允许 backgroundThrottling，降低后台进程与主进程的通信频率
3. 添加进程异常监控，对异常进程进行销毁重建

安全加固

ASAR 反解

```
[illegible] asar extract app.asar app.extract
[illegible]buffer.js:53
    throw new ERR_BUFFER_OUT_OF_BOUNDS();
    ^

RangeError [ERR_BUFFER_OUT_OF_BOUNDS]: Attempt to write outside buffer bounds
    at boundsError [illegible]buffer.js:53:11)
    at Buffer.readUInt32LE [illegible]buffer.js:[illegible])
```

杀软误杀场景

剪切板复制图片

Node-ffi 分配内存

修改开机启动项

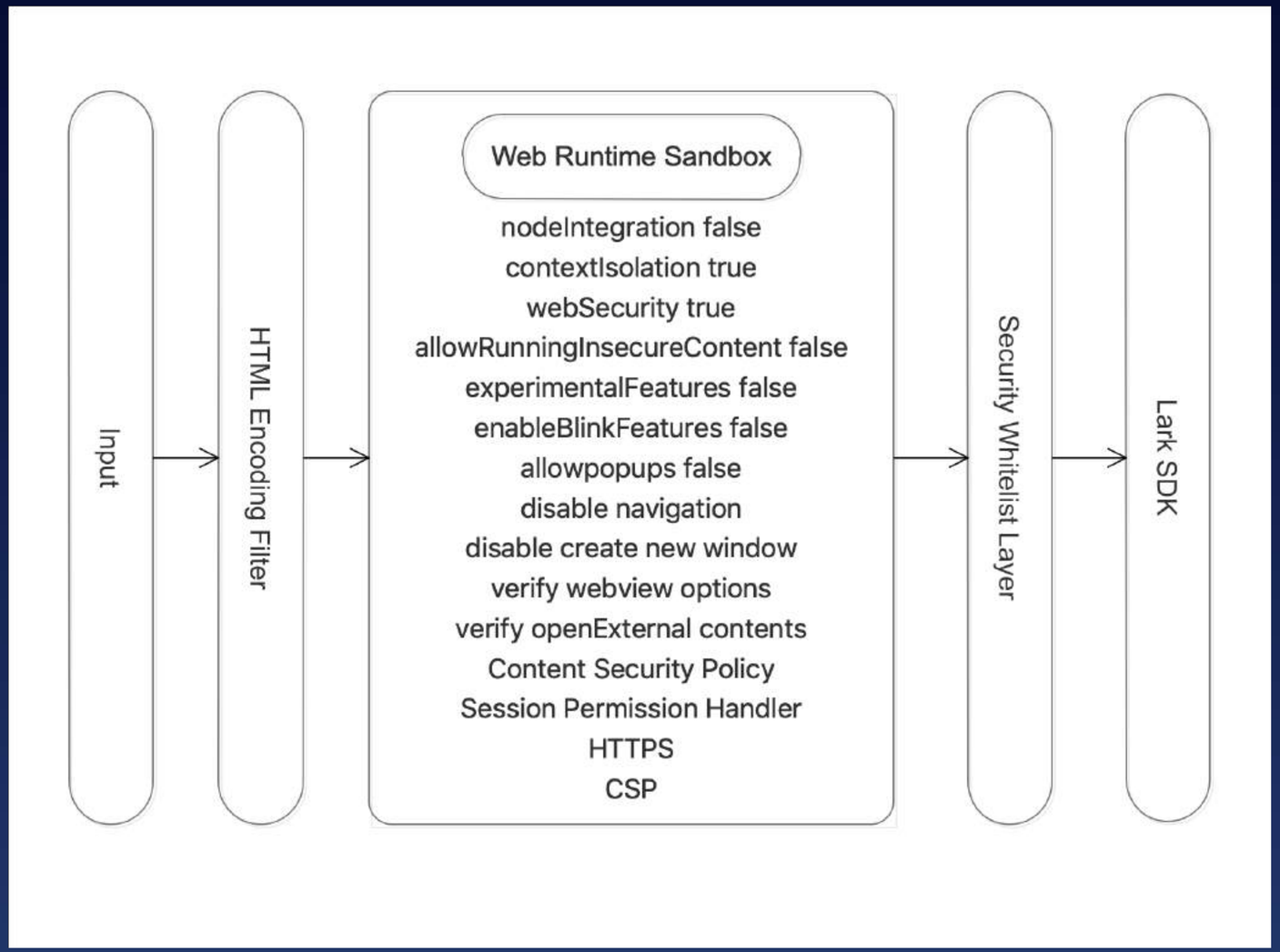
修改 URL 协议关联

解压更新包

杀软异常监控

杀软误杀处理

Web 攻击



1. 安全的 WebContents
2. 安全的用户输入
3. 安全的接口访问

安全调试

1. 线上包保留 DevTools
2. DevTools 调用白名单校验

业务治理

解决应用规模持续增长，引入的稳定性、性能等问题。

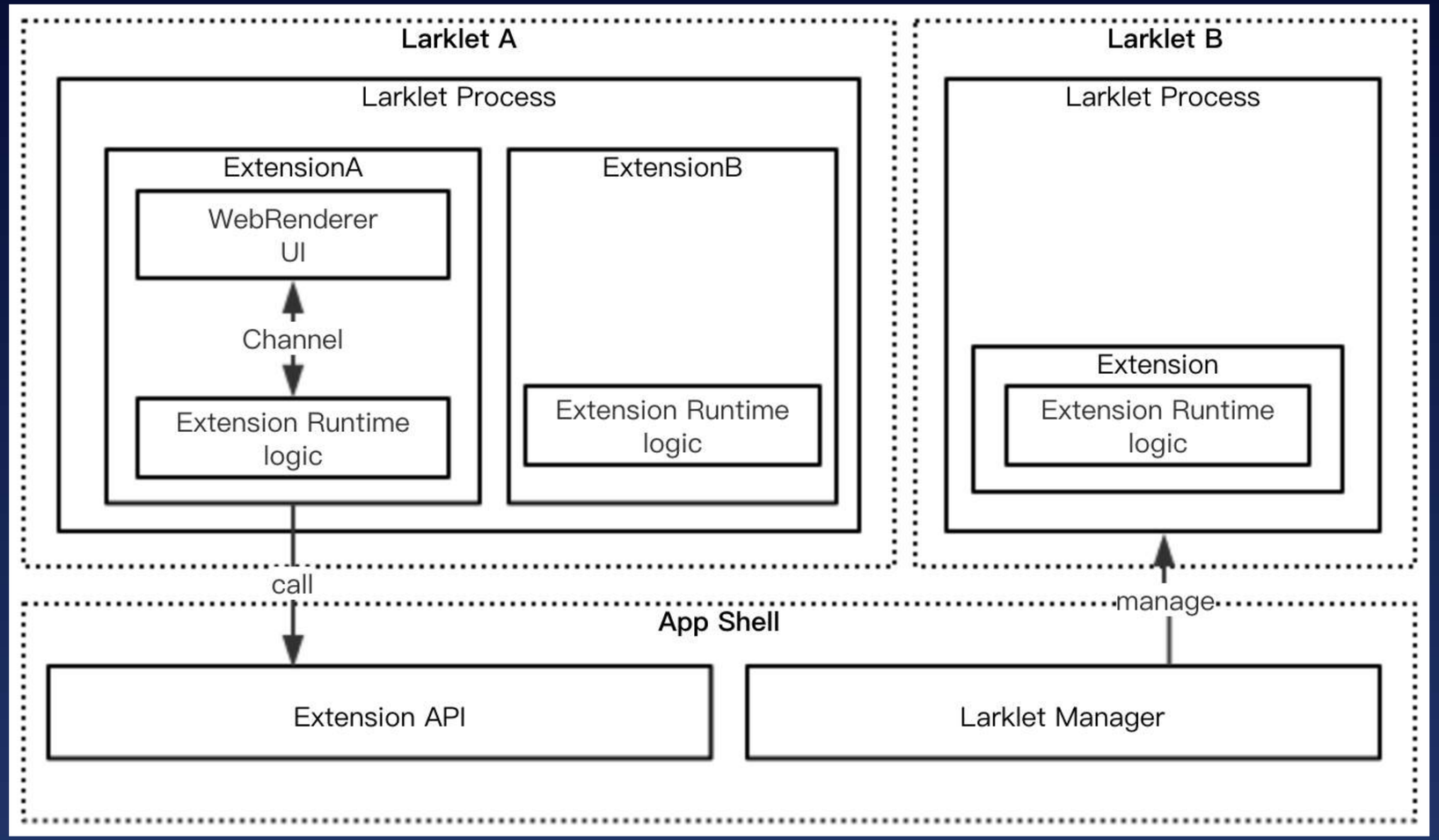
稳定性、性能问题表现

1. 业务间共享线程，资源竞争，业务响应变慢
2. 业务间共享 Context ，不能即用即走，内存越用越大

稳定性、性能问题根源

1. 业务运行环境不隔离，资源没有独立核算
2. WebContents 可控性差，不能满足长期运行场景

解决思路

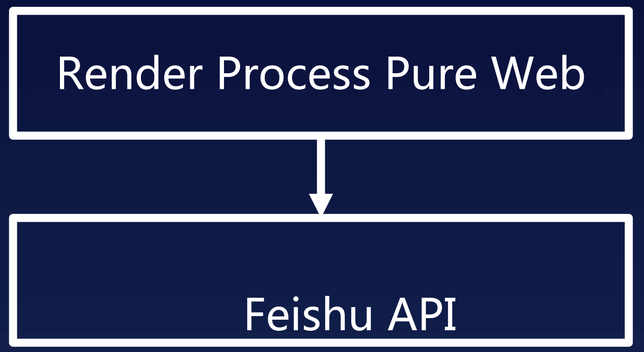


1. 业务间拆分为独立进程
2. 进程内拆分为 Logic 和 UI
3. Logic 运行在 nodejs 环境
4. UI 运行在 WebContents 环境
5. 轻量化 UI，即用即走

目录

Render Process Node Intergration

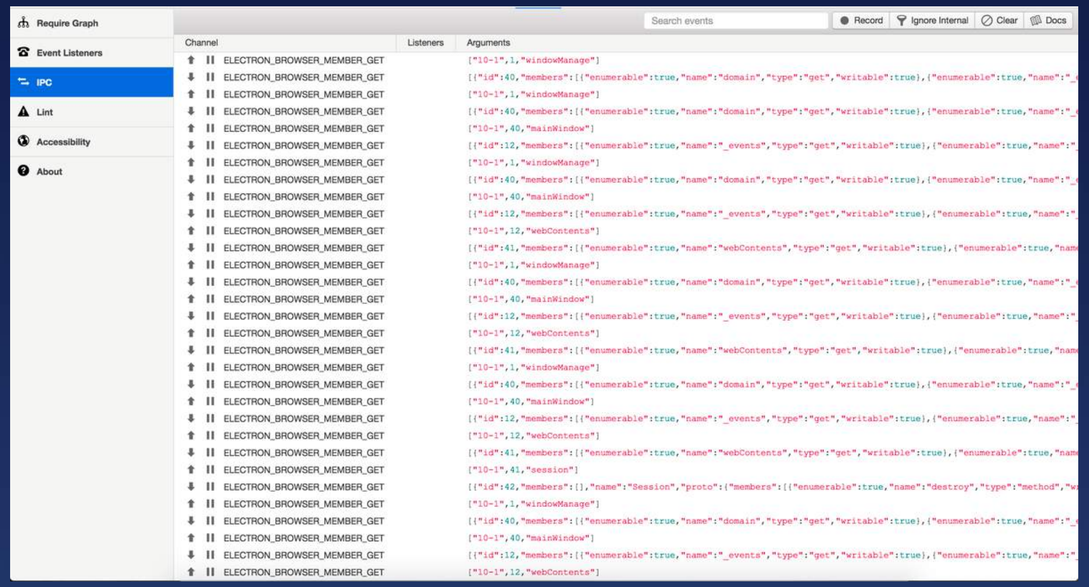
1. 多 WebContents 场景下，代码冗余，资源管理困难
2. 桌面、Web 端复用场景下，维护困难，难以复用



Remote 的性能问题

```
const { remote } = require('electron');

function clearAllCookies() { // cost 39ms
  if (
    !remote.app ||
    !remote.app.windowManage ||
    !remote.app.windowManage.mainWindow ||
    !remote.app.windowManage.mainWindow.webContents ||
    !remote.app.windowManage.mainWindow.webContents ||
    !remote.app.windowManage.mainWindow.webContents.session ||
    !remote.app.windowManage.mainWindow.webContents.session.clearStorageData ||
    typeof remote.app.windowManage.mainWindow.webContents.session.clearStorageData !== 'function'
  ) return;
  remote.app.windowManage.mainWindow.webContents.session.clearStorageData({ storages: ['cookies'] });
}
```



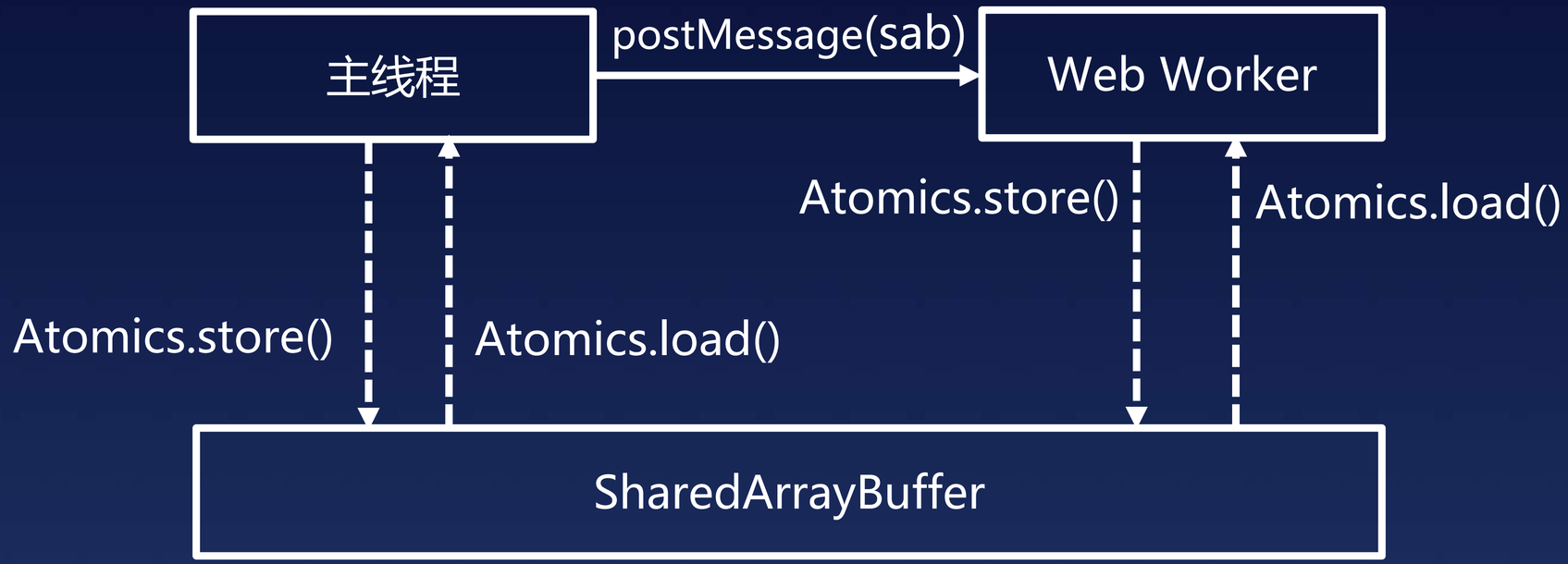
1. Remote 是基于 IPC 的 同步RPC 调动
2. 每次 get、set 都会触发同步 IPC
3. 同步 IPC 会 Block UI

require 的性能问题

1. require 会经过 resolve、load、wrap、runInContext 等操作，使用 require 加载 React、ReactDOM 比使用 Script 加载慢将近一倍
2. ASAR 能改善 require 的性能
3. 任何进程代码都尝试合并打包，提升加载性能

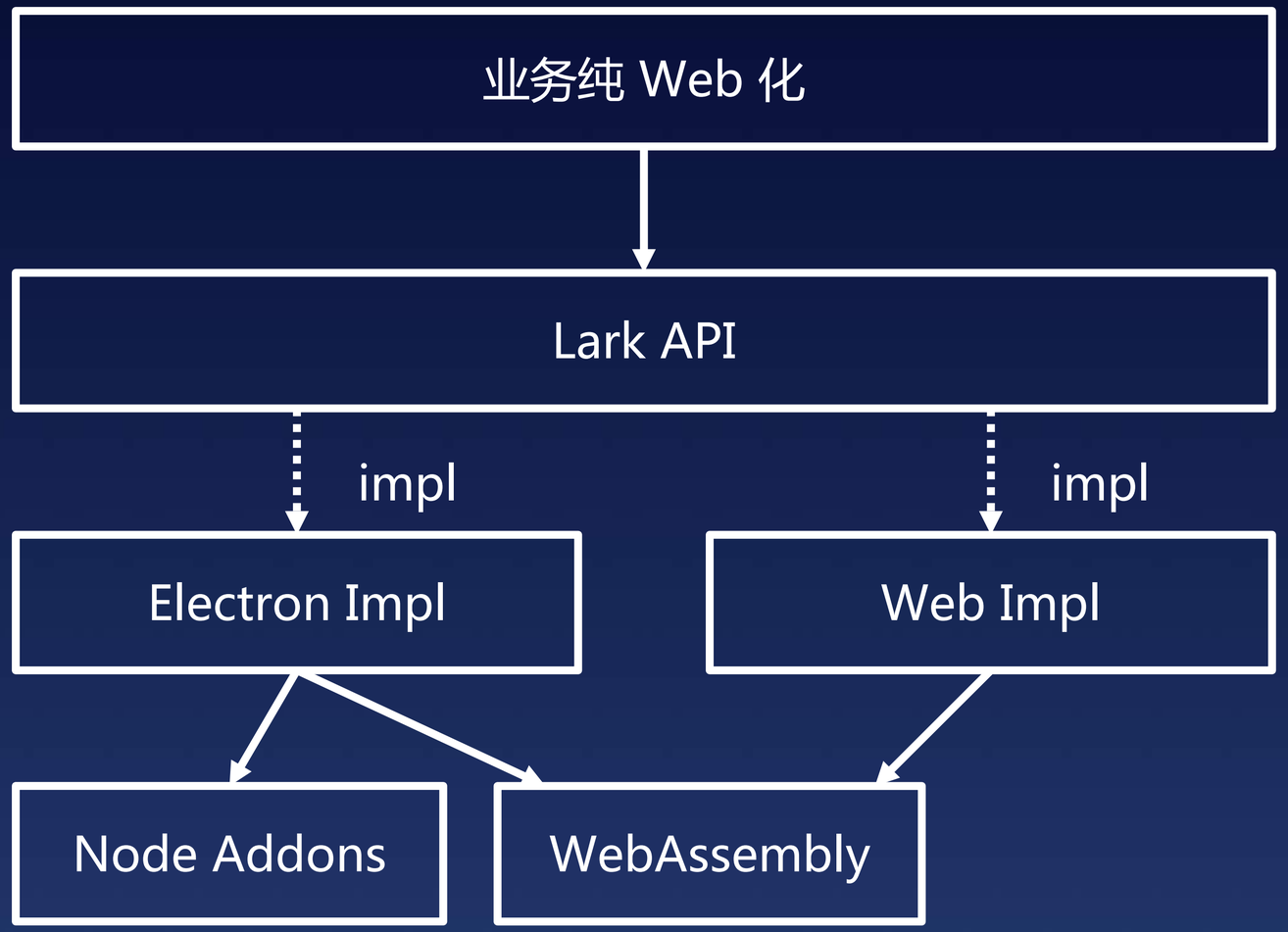
目录

使用 SharedArrayBuffer 实现共享内存




```
const sab = new SharedArrayBuffer(1024);
```

使用 Wasm 提升基础模块 Web 环境复用性



Q&A

Geekbang. 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

THANKS

GMTC
全球大前端技术大会

泡夫子

北京 海淀

扫一扫上面的二维码图案，加我微信