JavaScript in IoT

Yorkie

Rokid Engineer

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

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

自我介绍

- Creator of ShadowNode & tensorflow-nodejs
- Node.js Collaborator
- Rokid Engineer

目录

- What’ s the **IoT**?
- Why **JavaScript** ?
- **Node.js** on Edge Device
- Make an **OS** for JavaScript Developer

What's the IoT

- Service-oriented user interface
- Resource restricted device
- Efficient is greater than elegance

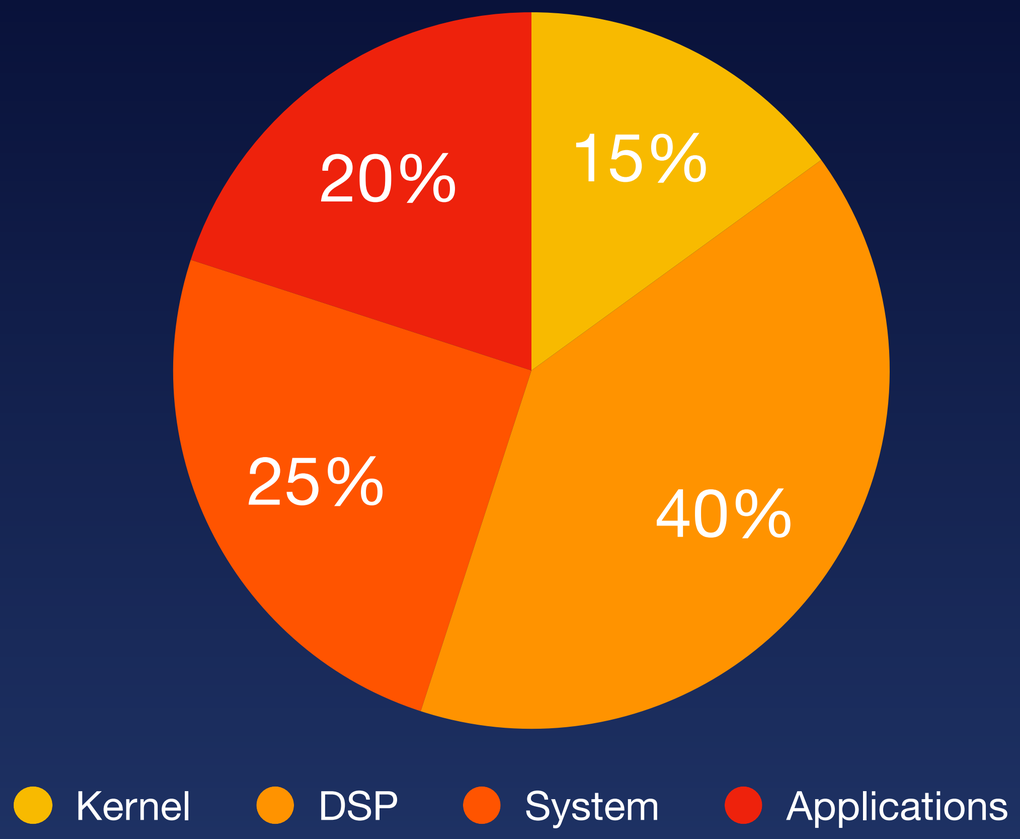
Service-oriented User Interface

- user interface is expanding to every IoT device.
- user interface is adaptive, also known as AUI.
- screen is optional.
- collaboration everywhere.
- in-parallel runtimes.

Resource Restricted Device

	low-end	medium-end	smart phone
RAM / ROM	4MB / 4MB	128MB / 128MB	2GB / 16GB
Processors	1 Core	1 - 4 Core	4 Core+
OS	RTOS	Linux	Android / iOS

Resource Restricted Device



20% * 128MB = 25.6MB

Efficient is greater than Elegance

- trade off everyday.
- efficient workflows on the hand.
- efficiency is elegant at real world.

Why JavaScript

- IoT fragmentation
- JavaScript ecosystems (including Web and Node.js)

IoT Fragmentation

The unified Web — WOT

Popular Programming language — JavaScript

Popular Programming language — JavaScript

**“Any application that can be written in JavaScript,
will eventually be written in JavaScript ?”**

Node.js on Edge Device — ShadowNode

- Brief introduction and history of ShadowNode
- Use **N-API**
- Optimizations on device

ShadowNode: brief introduction

- bio: use Node.js in your end device
  - support Linux and macOS
  - support x86, arm and aarch64
  - core APIs: *assert / buffer / N-API Add-ons / child process / crypto / dns / events / file system / http / https / module / net / os / process / timers / TLS / UDP*
  - extended APIs: *WebSocket / D-Bus / MQTT / Profiler*
  - heap & cpu profiler
- a fork of another awesome project: IoT.js
  - panda-project/jerryscript: memory optimized ECMAScript JITless VM
  - panda-project/libtuv: memory optimized libuv fork

ShadowNode: project tree

- src/
  - src/modues.json, modules configure.
  - src/iotjs.c, bootstrap and entry file.
  - modules/, the C implementation of core modules.
  - js/, the JavaScript part of core modules.
  - napi/, the N-API related files.
- deps/, the dependencies just like Node.js
- cmake/, the build scripts in CMake.
- test/, the test directory
  - testsets.json, defines the run set for unit tests.
  - napi-testsets.json, defines run set for N-API tests.
- tools/
  - tools/build.py, the build starter

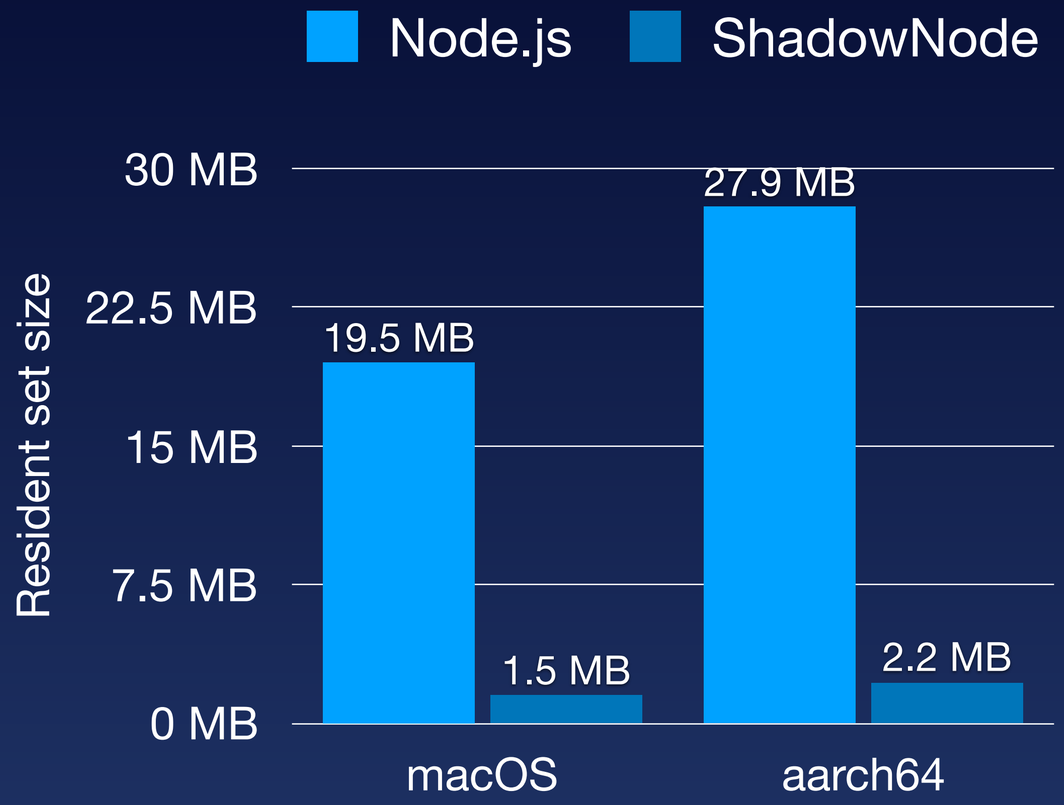
ShadowNode: history

- ***2017.09*** Migrate a Node.js application from android to linux which only owns 256MB RAM at **Rokid**.
- ***2017.10*** Try to do reduction on this existed application includes: tree-shaking, unnecessary dependencies and refactors, but failed always.
- ***2017.11*** The IoT.js project was found, but no TLS, MQTTs, D-Bus and child process, start hacking ShadowNode and supporting the above modules and some of NPM modules.
- ***2018.06*** **N-API** implementation has been released.
- ***2018.08*** Start building a new project which is based on ShadowNode.

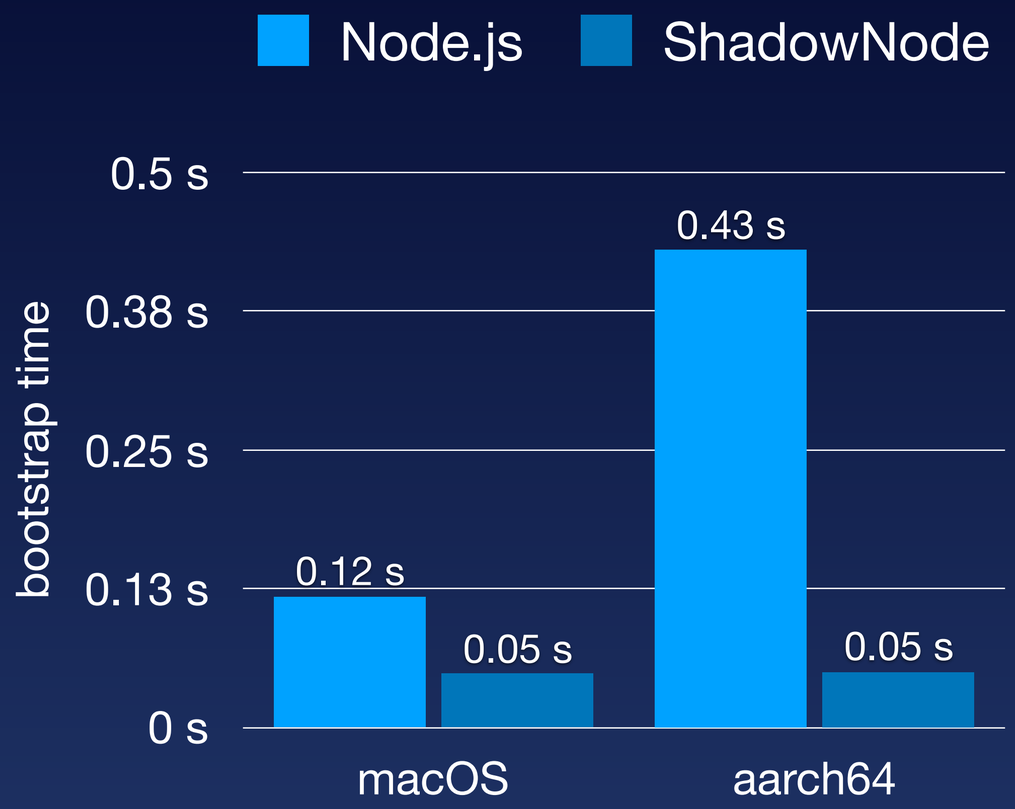
Compare to Node.js

NPM	
Node.js Official	**ShadowNode**
V8	**JerryScript**
libuv	**libtuv**
OpenSSL	**mbedTLS**

Compare to Node.js



Compare to Node.js



Compare to Node.js

- ShadowNode is not going to replace the official Node.js
- ShadowNode shall be the subset of Node.js ecosystem on embedded system
- ShadowNode embraces Node.js community

Use N-API

- how it works seamlessly on Node.js & ShadowNode
- implement `HandleScope` on JerryScript
- try **N-API**
- test **N-API** implementation

N-API ABI Stability

- no re-compiling on different Node.js versions.
- no re-compiling on different JavaScript engine like node-chakracore.
- no re-compiling on the runtime outside of Node.js like ShadowNode.

Comments from Node.js Team

Node.js Collection

Community-curated content for the millions of Node.js users.

Following

340

Since N-API provides a JavaScript engine-independent interface to the language features, its availability makes it possible to drop other JavaScript engines into Node.js. The node-chakracore project is an example of a Node.js version with a different JavaScript engine “under the hood”.

N-API can also serve as an interface to the JavaScript engine outside of Node.js. The ShadowNode project is in the process of implementing N-API using the JerryScript JavaScript engine as its back end. Such efforts make it possible to provide native add-ons for a variety of environments such as constrained IoT devices using a single code base.

HandleScope on V8

```
void Init(Handle<Object> target) {
  HandleScope scope;
  target->Set(String::New(“gc"), FunctionTemplate::New(GC)->GetFunction());
  target->Set(String::New(“pause”),
              FunctionTemplate::New(Pause)->GetFunction());
  target->Set(String::New("resume"),
              FunctionTemplate::New(Resume)->GetFunction());
}

// copy from https://github.com/bnoordhuis/node-profiler
```

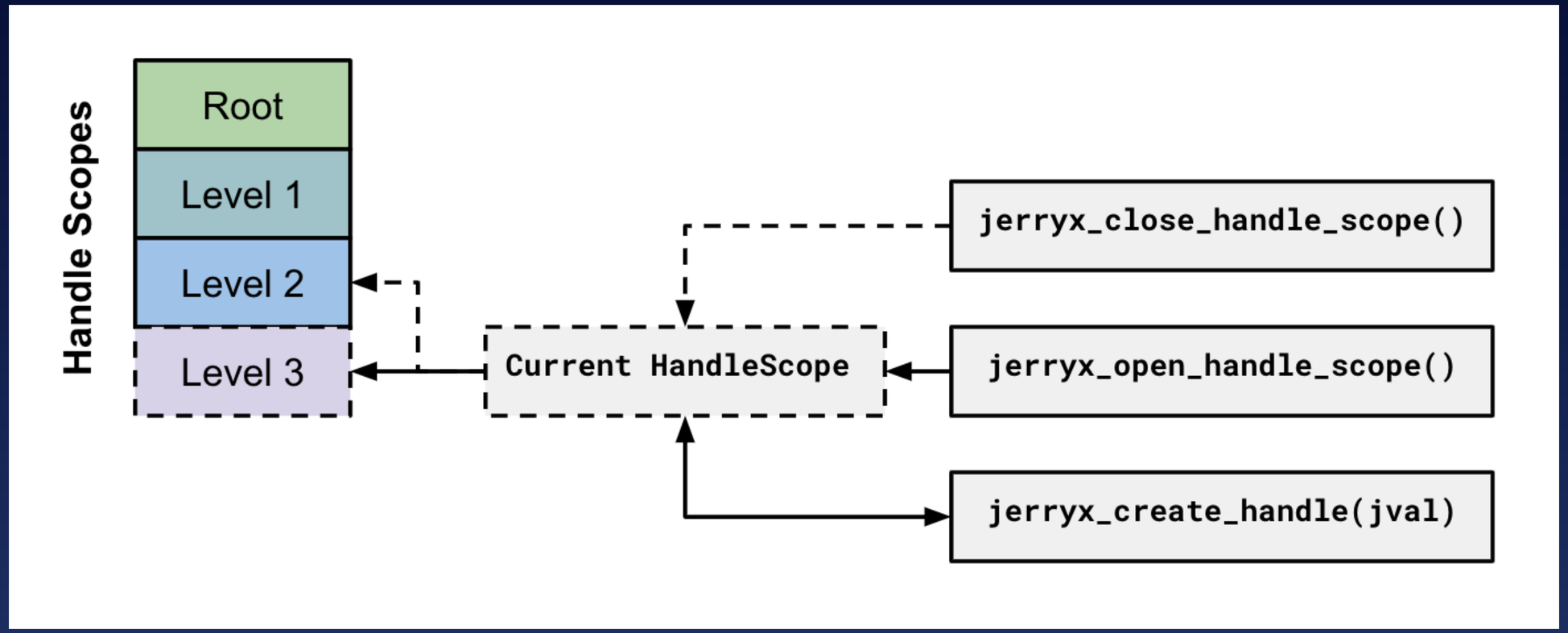
HandleScope on JerryScript

```
JS_FUNCTION(Test) {
  jerry_value_t name = JS_GET_ARG(0, string);
  // ... do something
  jerry_release_value(name);
}
```

HandleScope on JerryScript

```
JS_FUNCTION(Test) {
  jerry_value_t name = JS_GET_ARG(0, string);
  // ... do something
  jerry_release_value(name);
}

JS_FUNCTION(TestWithHandleScope) {
  jerryx_handle_scope scope;
  jerryx_open_handle_scope(&scope);

  jerry_value_t name = jerryx_create_handle(JS_GET_ARG(0, string));
  jerryx_close_handle_scope(scope);
  // name has been released
}
```

HandleScope on JerryScript



Usage threshold of HandleScope

- `HandleScope` is nested, but it commonly is up to a threshold value.
- introduce a pool to pre-allocated handle scopes could avoid most `malloc()` in real world.
- the pool is configurable by **`JERRYX_SCOPE_PRELIST_SIZE(20)`**, it stands for the maximum depth at call stack without `malloc()` at N-API add-on.

Try **N-API**

Test **N-API** implementation

- pull **N-API** tests from nodejs/node repository.
- build add-ons with node-gyp.
- run tests on ShadowNode.
- NAPI Test Suite reuses tests for different **N-API** implementations, but still work in progress.

Optimizations on Device

- re-implement the NPM package in C is always working.
- otherwise
  - introduce **NODE_PRIORITIZED_PATH** to decrease the module path searching on device case.
  - linux ***Copy-On-Write***.

C/C++ advanced NPM package

- JavaScript must be the application-level language at embedded device.
- JavaScript object heap is always expensive than system allocators.
- keep the API to be consistent at JavaScript.
- some packages are optimized in this way: MQTT / WebSocket.

NODE_PRIORITIZED_PATH

- `NODE_PRIORITIZED_PATH=/usr/lib/node_modules node app.js`
- A workaround to improve performance of the Node.js module searching algorithm.
  - most `require` are from the global path, which differs from server-side use cases.
  - no effect on previous algorithm by introducing the new env variable.
- optimized 30% on booting the main service.

Copy On Write

- principle of laziness, do action when it’s really required.
- do copy when the data is really changed.
- linux `fork()` implements COW.

Copy On Write - uv_spawn

```
function spawn (file, args) {
  var pid = fork()
  if (pid === 0) {
    execvp(file, args) // this disables COW
    // starting VM and load script
  }
}

spawn(‘test.js’, [])
```

Copy On Write - fork

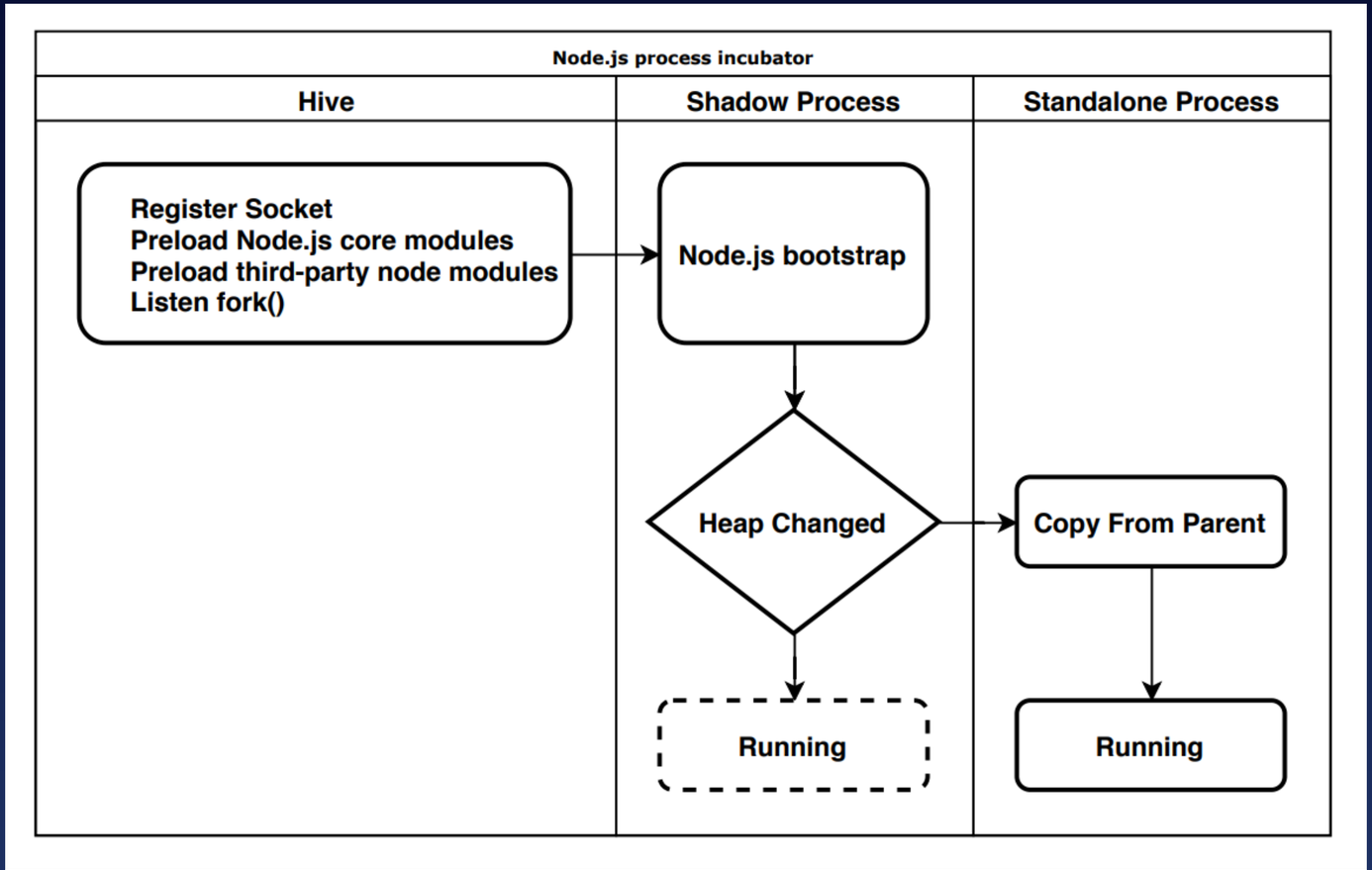
```
var fork = require('linux-sys').fork

// load common modules for children
var player = require('player')
var http = require('http')
var foobar = require('foobar')

// start forking
var pid = fork()
if (pid == 0) {
  // here is the child process
  // use player / http / foobar
}
```

4ms

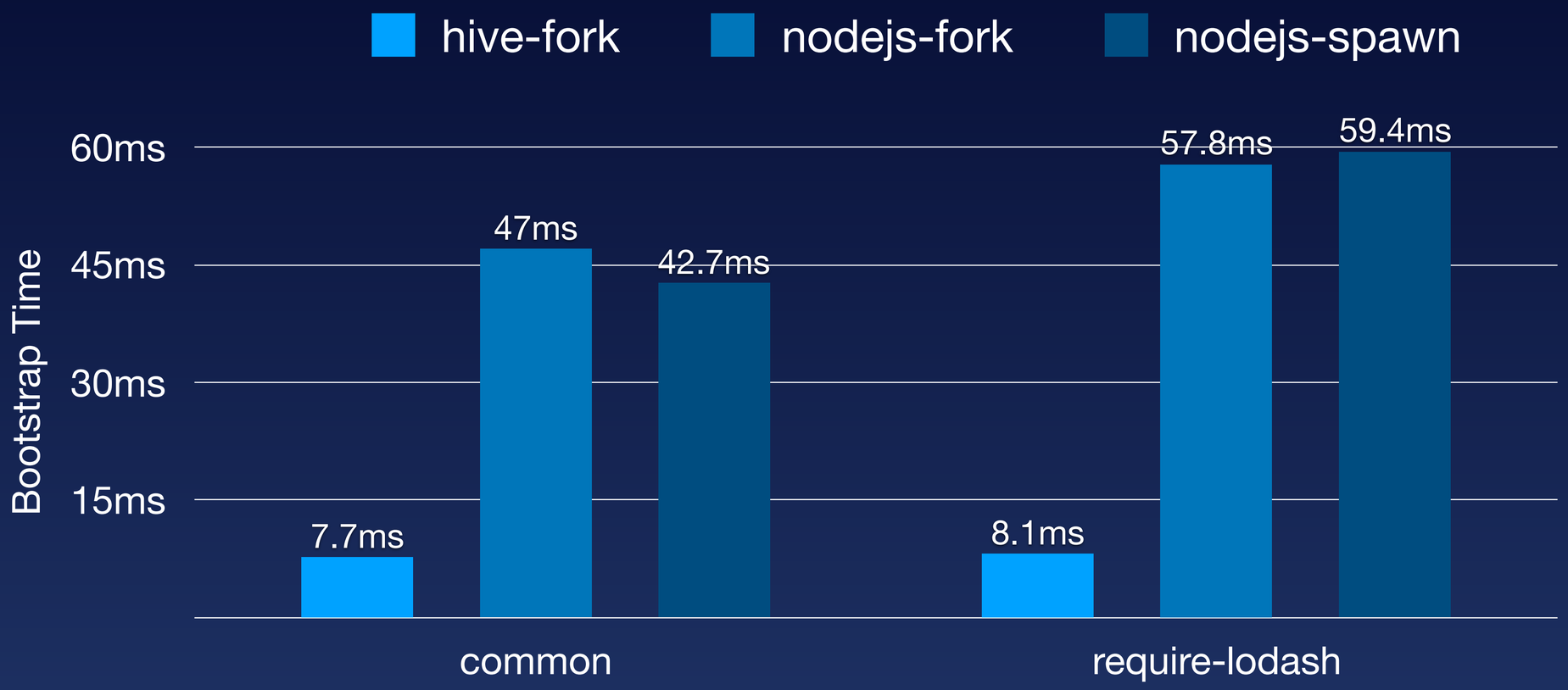
Hive — Node.js process incubator

Zygote	**Hive**
Register socket	**Register socket**
Preload all Java class	**Preload all Node.js core modules**
Preload resource	**Preload third-party node modules**
Listen for `fork()` connection	**Listen for `fork()` connection**

Hive — Node.js process incubator



Hive — Node.js process incubator



Make an OS for JavaScript Developer

- Guide to make an operating system
- Principles of design API
- Boundaries between JavaScript and others

Guide to make an operating system

Source code	**Google Repo**
Kernel	**Linux**
Build Framework	**OpenWRT**
Configuration	**defconfig**

Google Repo

```
repo init
  -u https://github.com/yodaos-project/yodaos
  -m default.xml
  -b master

repo sync
```

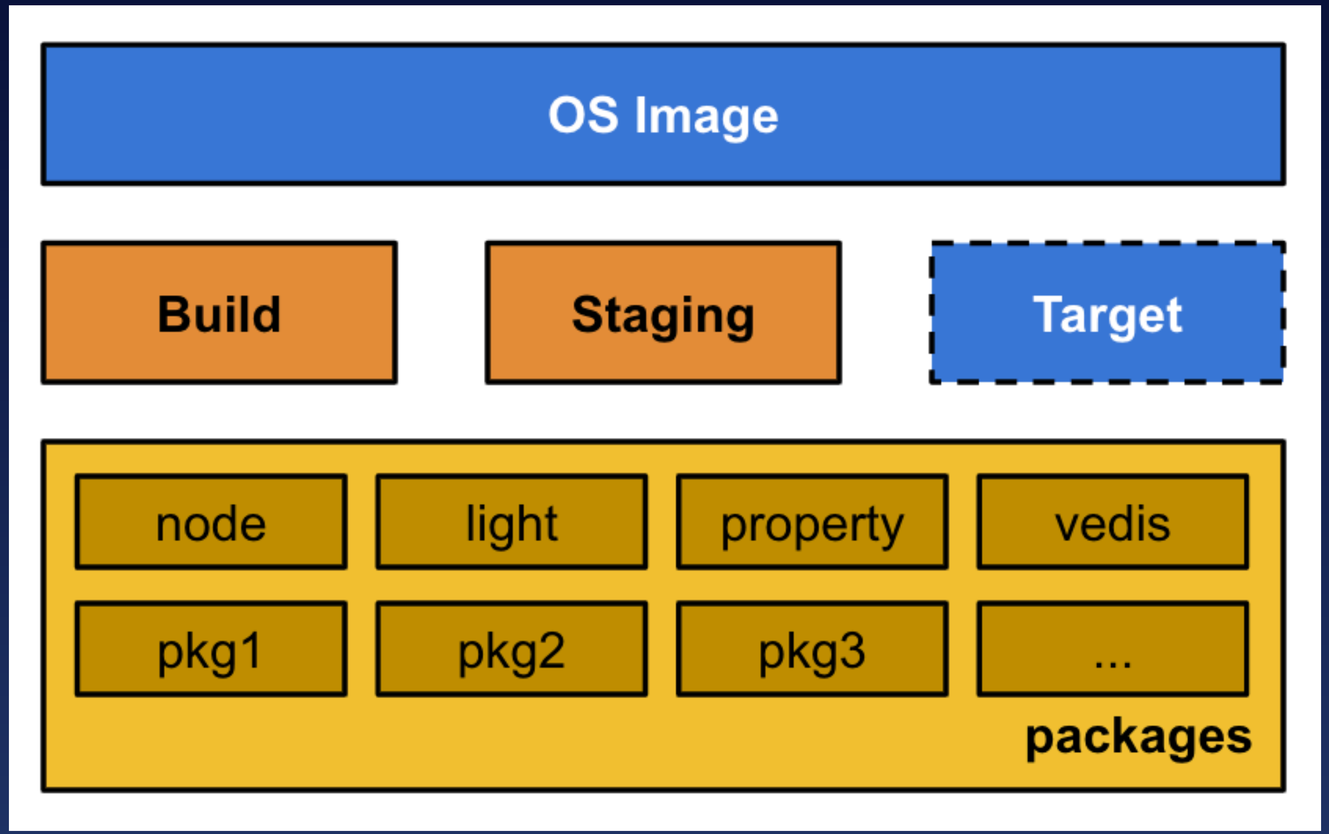
Google Repo

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <remote name="github" fetch="https://github.com" review="https://github.com" />
  <default revision="master" remote="github" sync-c="true" sync-j="2" />
  <include name="manifests/base.xml" />

  <project path="libs/nodejs" name="yodaos-project/ShadowNode" remote="github"
revision=“master" />
  <project path="products/yodaos/rbpi-3b-plus"
    name="yodaos-project/product-raspberry" remote="github" revision=“refs/tags/v1.0" />
</manifest>
```

Kernel — Why Linux?

- provide the Posix-compatible APIs for system programming.
- provide the richness drivers ecosystem for different hardwares.
  - microphone
  - speaker
  - input event
  - motion control
  - light & display
  - ...

OpenWRT - build system for linux distribution



Makefile and defconfig

```
# config target
CONFIG_TARGET_leo=y
CONFIG_TARGET_leo_k18_universal=y
CONFIG_TARGET_leo_k18_universal_LEO_K18_UNIVERSAL=y

# config product
CONFIG_PRODUCT_OS_NAME="yodaos"
CONFIG_PRODUCT_NAME="Rokid-Devkit"
CONFIG_PRODUCT_PATHNAME="rokid/universal"

# config kernel
CONFIG_EXTERNAL_KERNEL_CONFIG=y
CONFIG_EXTERNAL_KERNEL_TREE="$(TOPDIR)/../kernel/k18/4.4"

# config packages
CONFIG_PACKAGE_nodejs=y
CONFIG_PACKAGE_wget=y
CONFIG_PACKAGE_wpa-cli=y
CONFIG_PACKAGE_wpa-supplicant=y
CONFIG_PACKAGE_light=y
```

Makefile and defconfig

```
$ cd ./openwrt
$ cp ./path/to/your/defconfig .config
$ make defconfig
$ make -j32
```

There is an OS for Web community

YODAOS

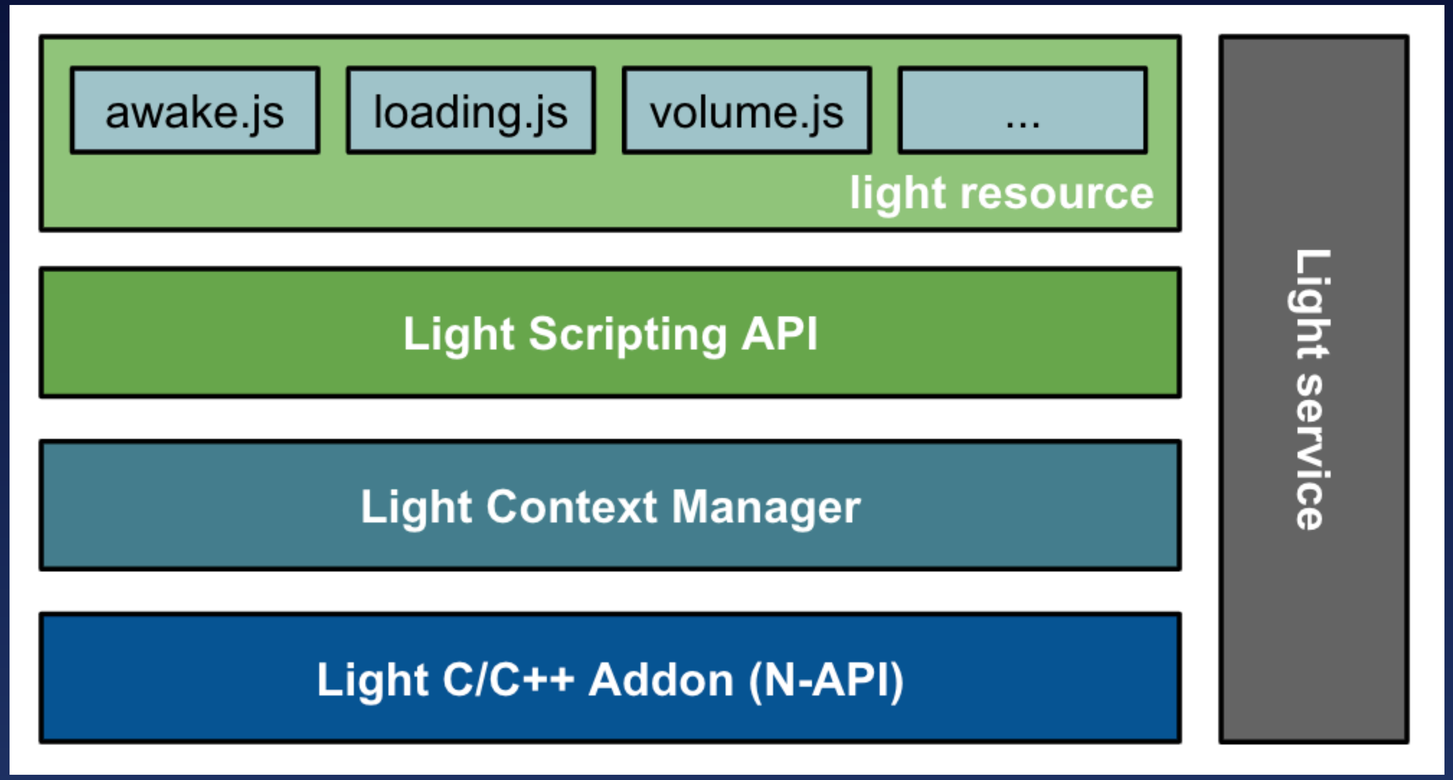
Principles of design API

- consider the messaging ways firstly
- the shape of applications on your OS
- design your “system calls”
- design your application API

Boundaries between JavaScript and others

- JavaScript is application-level
- outside of Node.js

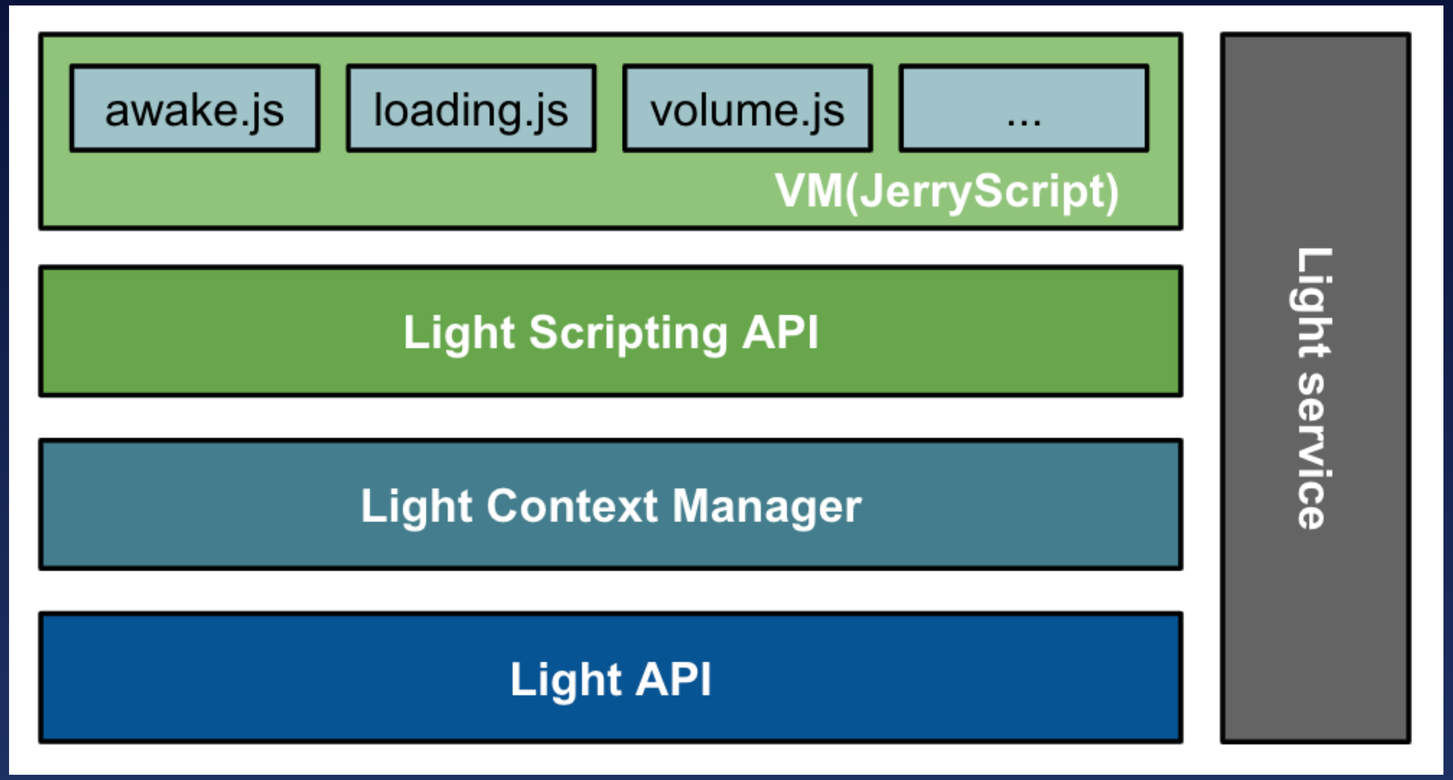
Example: LED Architecture(0)



Example: LED Architecture

```
module.exports = function render (light, args, done) {
  var muted = !!(args && args.muted)
  light.clear()

  if (!muted) {
    light.sound('system://mic_enable.ogg')
    light.render()
    done()
  } else {
    light.sound('system://mic_close_tts.ogg')
    light.fill(255, 0, 0)
    light.render()
  }
}
```

Example: LED Architecture(1)



THANKS

GMTC
全球大前端技术大会