Introduction to IoT.js

September 2015

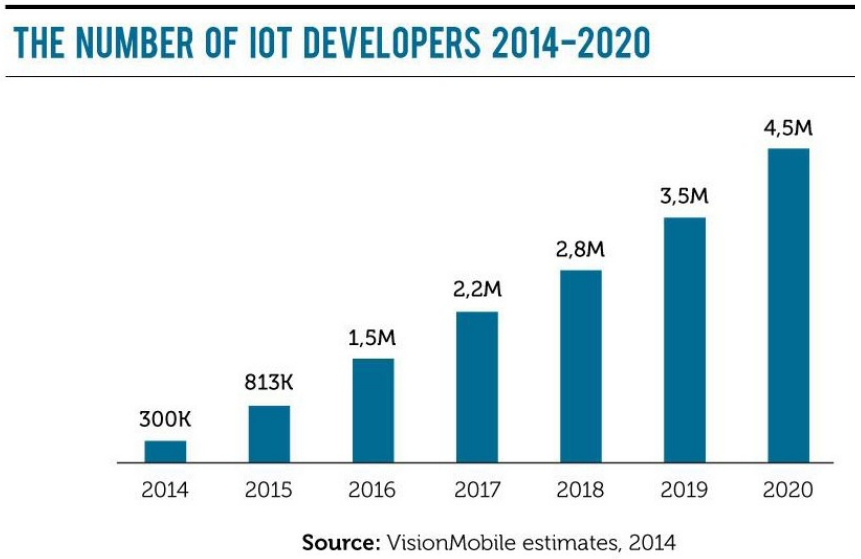
Software Center
Samsung Electronics

World of connected devices

"Number of Connected objects expected to reach 50 billion by 2020", Cisco 2014
"There is expected to be 75 billion connected devices by 2020", Intel 2014

Number of connected devices is continuously growing!

- Becoming cheaper
- Connected to the network
- Sensing data
- Acting on output



Let's create interactive applications
by organizing billions of connected devices!

The Challenge (=The problem of IoT today)

- We expect one hundred billion IoT devices to be deployed within ten years

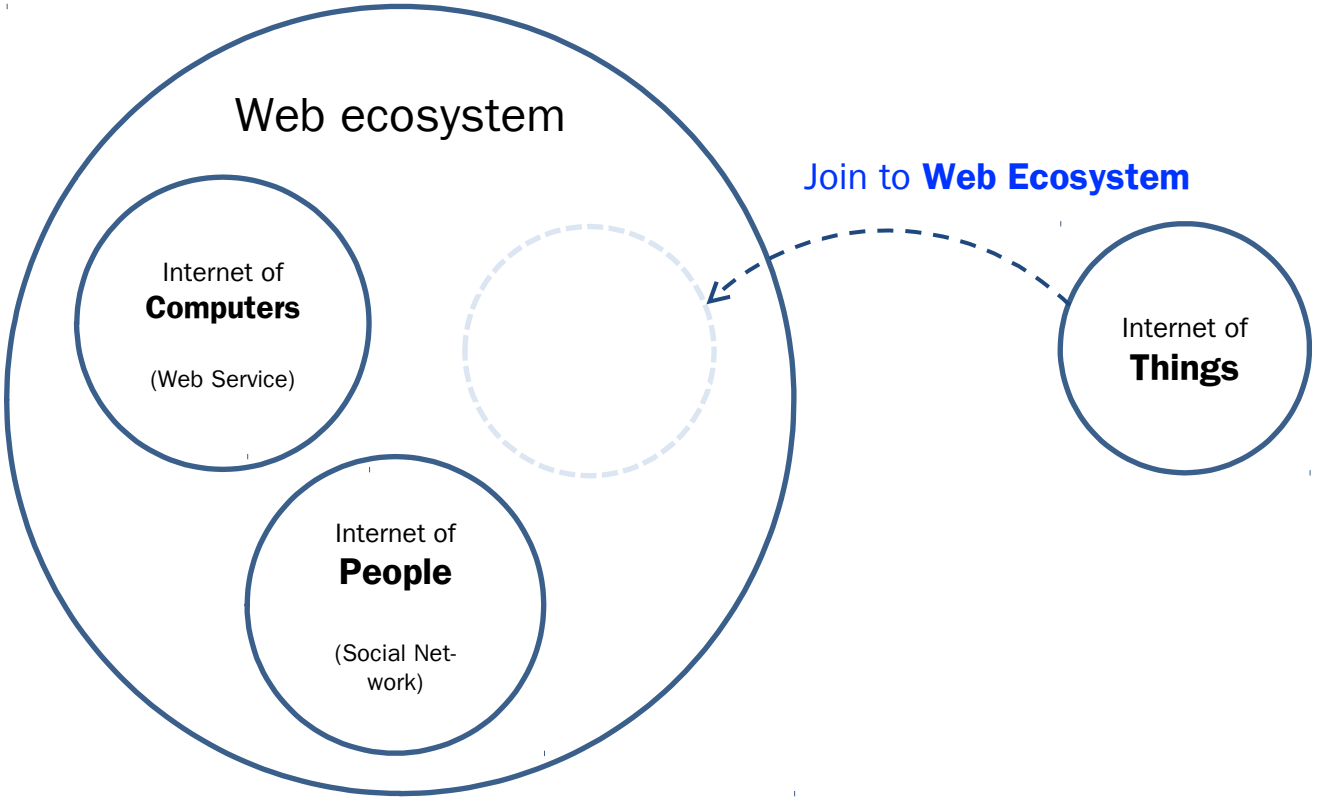
- But, the Internet of Things is currently beset with problems
  - Product silos that don't interoperate with each other
  - Plethora of approaches & incompatible platforms
  - This is blocking the benefits of the network effect

- This is painful for developers
  - Hard to keep track of who is doing what
  - Expensive to learn and port to different platforms
  - Challenging to create services that span domains and platforms

- Platform developers seeking to unlock the commercial potential
  - To reduce development costs for IoT applications and services
  - To fulfil customer demand for services requiring integration with other platforms
  - To grow the size of the overall markets
    - A small share of a huge market is better than a big share of a small market

***Dave Raggett**, An introduction to the Web of things Framework, W3C WOT IG, May 2015*
http://www.w3.org/2015/05/wot-framework.pdf

Easiest way to build up IoT ecosystem



What is IoT.js?

- IoT.js is JavaScript powered IoT application/service platform.
- Simply say downsized version of node.js (http://nodejs.org)
  - Most famous platform in web developer community

- Site Links
  - http://www.iotjs.net
  - https://github.com/Samsung/iotjs

- Demo Video
  - https://youtu.be/FLnT129j64c

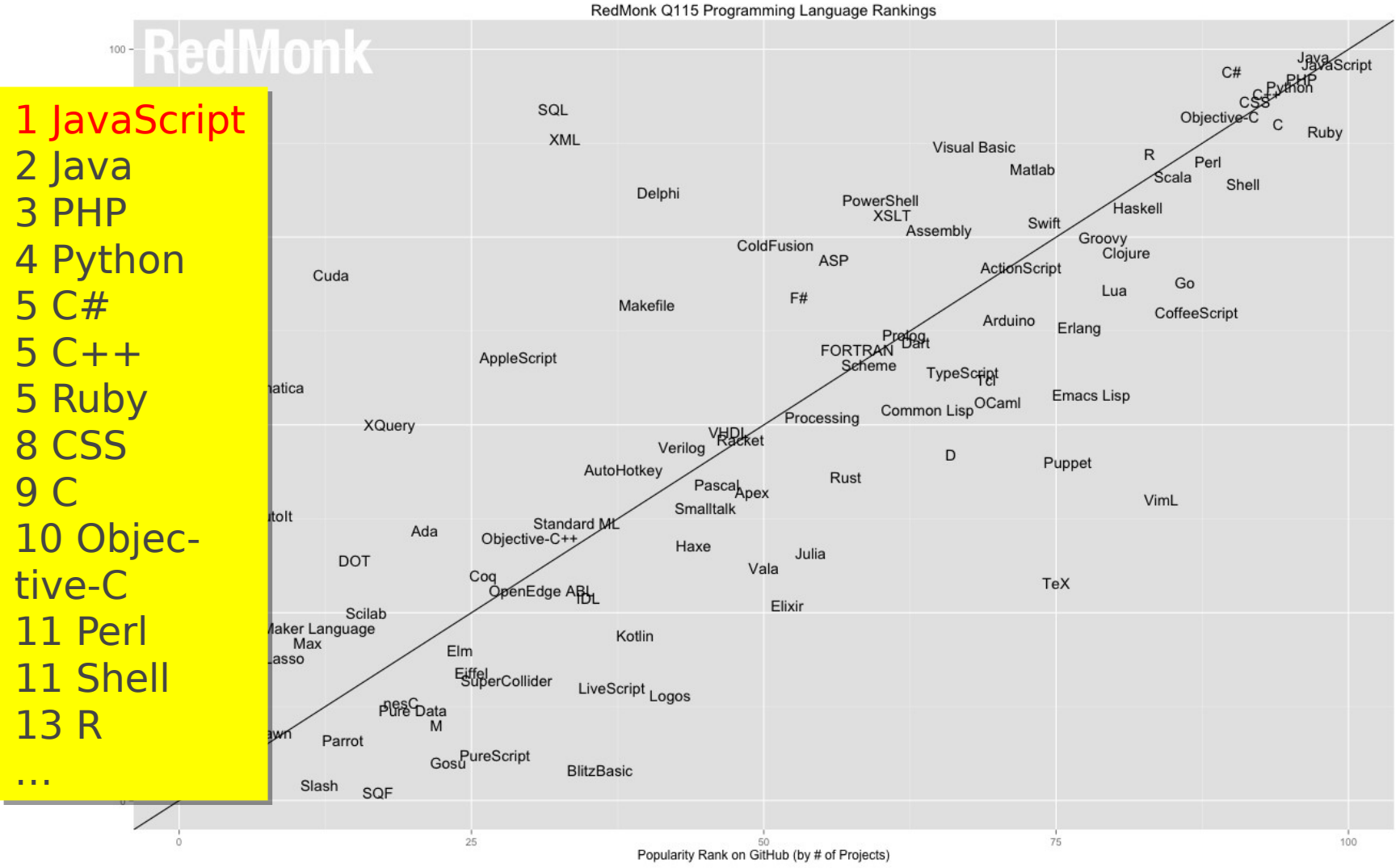
What is this for?

- JavaScript developers become makers
  - World's largest software developer pool today
- Fast prototyping solution for independent IoT developers
  - Makers build projects with less hassle
  - Product designers prototyping IoT applications with easy
  - Developing IoT solutions with JavaScript, such like developing web applications
  - Even production with highly optimized solutions
- Competitive solution for IoT chip vendors
  - Exclusive, essential solution for strengthen the product competitiveness

Why JavaScript?

- No. 1 famous language in the world!

1 JavaScript
2 Java
3 PHP
4 Python
5 C#
5 C++
5 Ruby
8 CSS
9 C
10 Objec-tive-C
11 Perl
11 Shell
13 R
...



https://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/

Why JavaScript?

- JavaScript* is in wide use
  - Large numbers of web developers are familiar with it
  - Well-documented with a strong ecosystem
  - Already standardized and with multiple implementations

- JavaScript is consistent with web programming and portable mobile apps
  - It is used by HTML5, and HTML5 is useful for developing UI “companion apps” for IoT devices

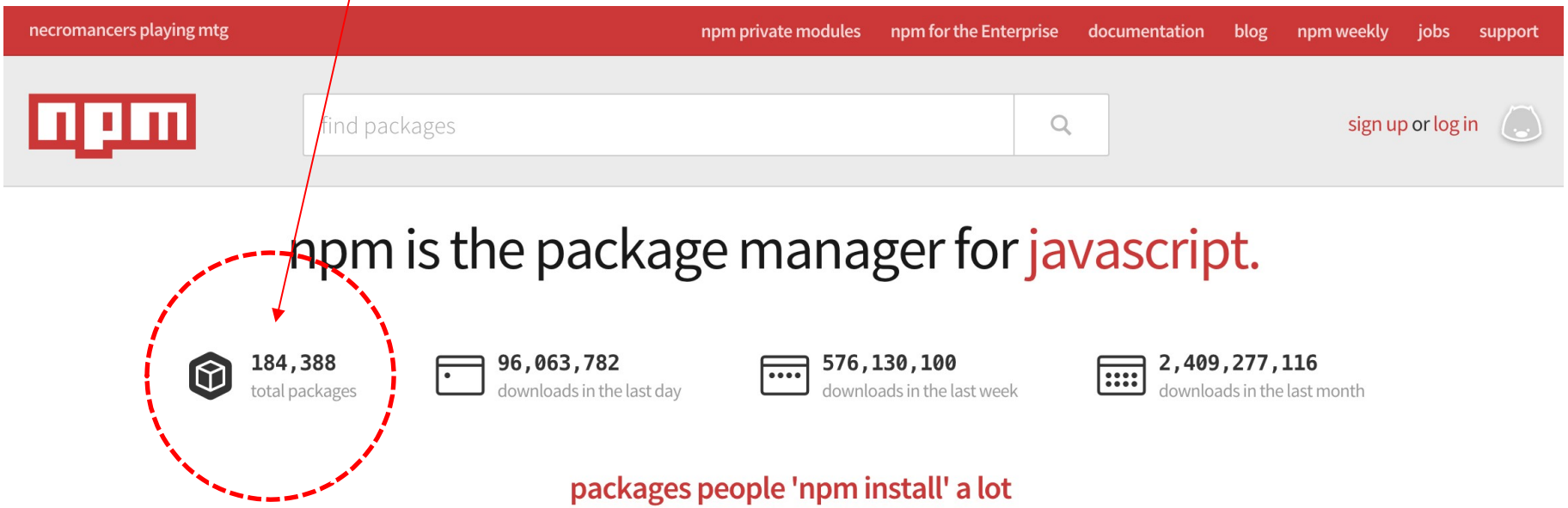
- JavaScript is well-suited to embedded device programming
  - Supports asynchronous function calls and I/O
  - Asynchronous calls are useful for event-driven hardware programming
  - The Node.js* engine in particular has many useful features for both web services and embedded devices

- ***Michel McCool,*** *Programming the Internet of Things Using Node.js, IDF15,*

Why IoT.js?

- Backward compatibility to node.js and it's applications
  - Well known programming model
  - Over a hundred thousand packages available via the npm package manager

http://www.npmjs.org



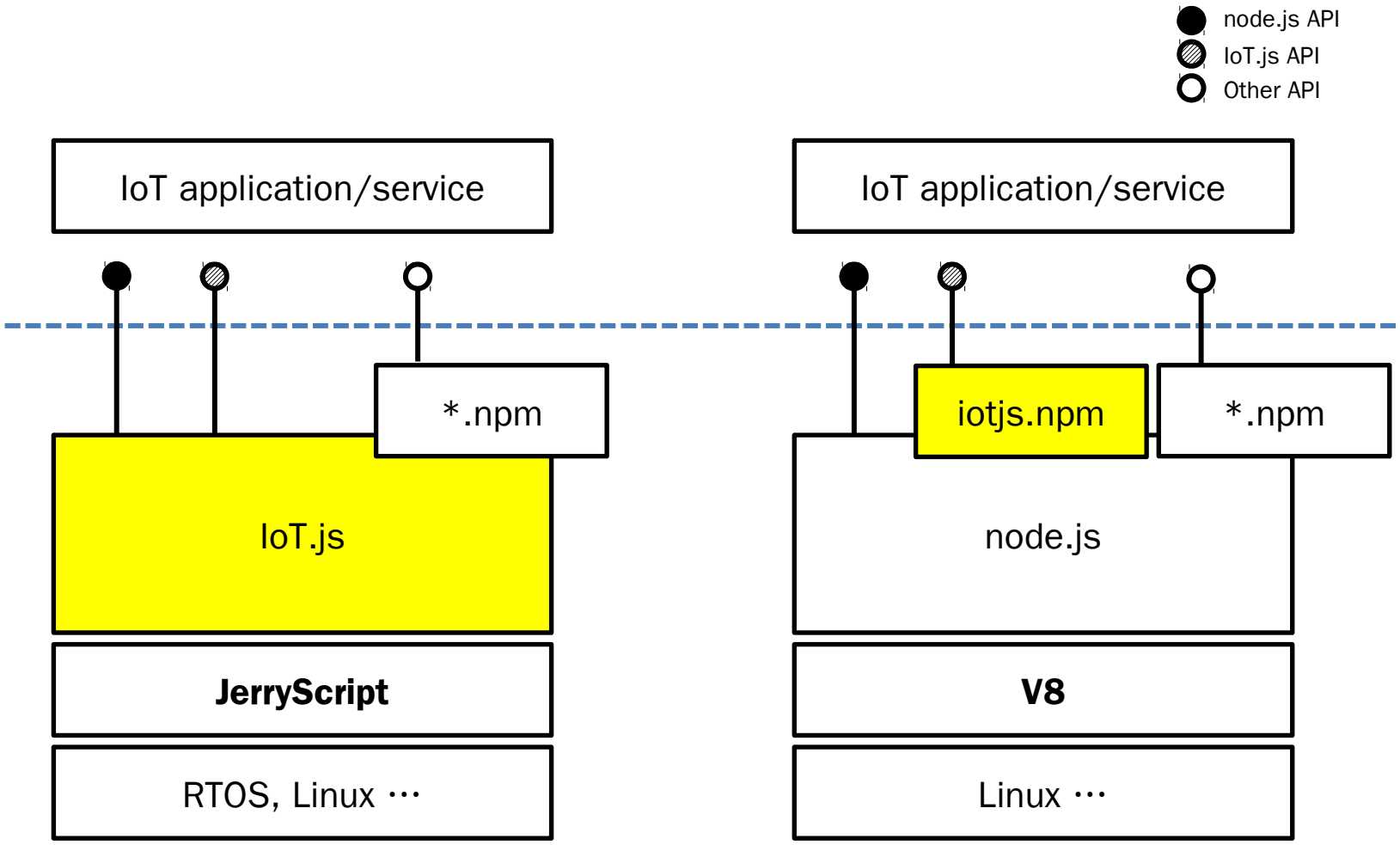
express

Why Node.js?

- Strong ecosystem and package management system
  - Over a hundred thousand packages available via the npm package manager
- Programming model well-suited to embedded devices as well as servers
  - Event-driven asynchronous programming model, support for asynchronous functions
  - Lack of explicit event loop means transparent power state management can be implemented
  - Good support for interfacing to native C libraries
- Community is already using Node.js* for embedded devices and robotics
  - For example: http://nodebots.io/, Firmata, Cylon, JohnnyFive, …
- Web services can (of course!) also be built with Node.js

- ***Michel McCool,*** *Programming the Internet of Things Using Node.js, IDF15,*

IoT.js vs Node.js



IoT.js

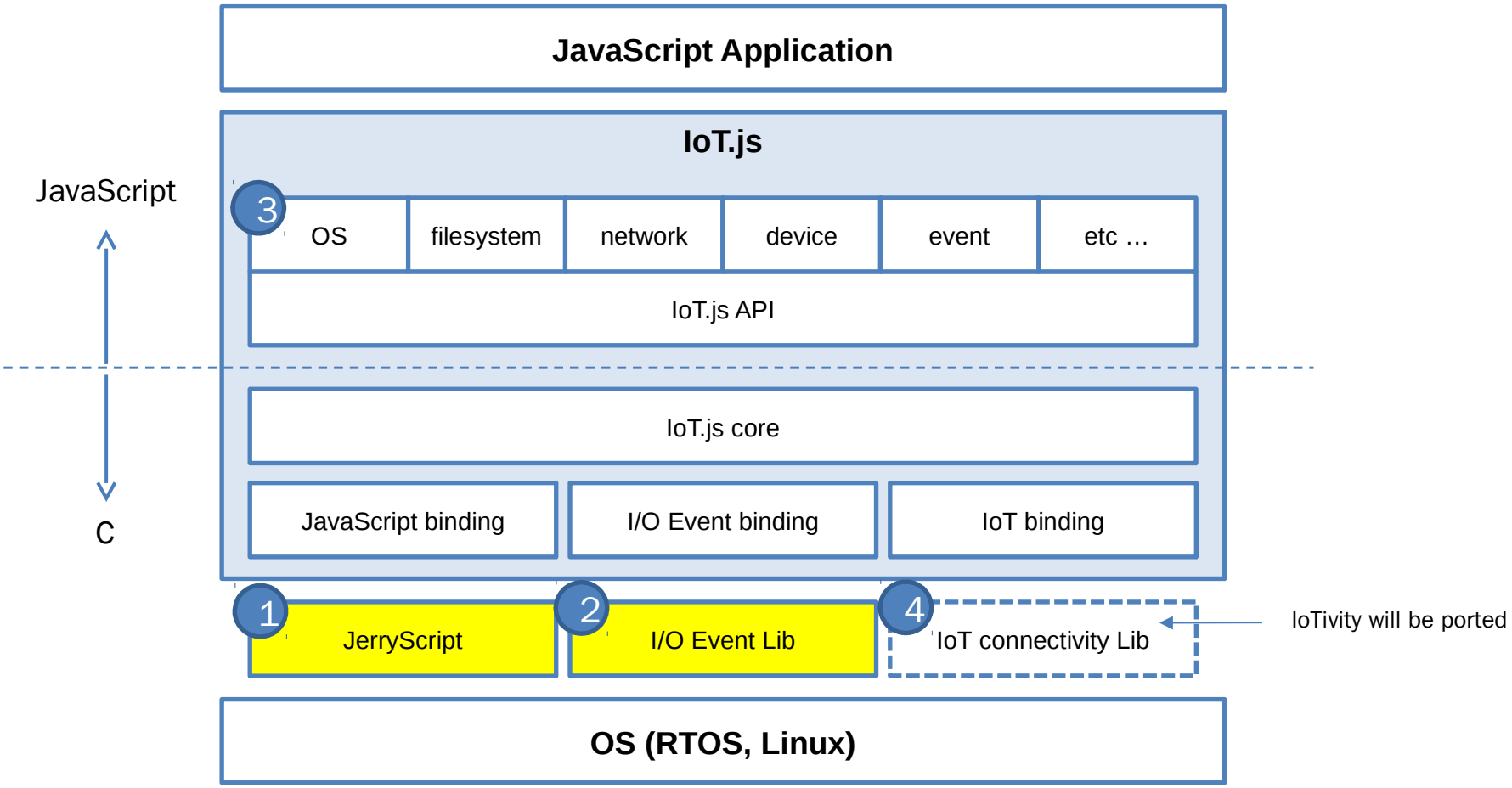
(**256KB** FLASH + **256KB** RAM)

Node.js

(**11MB** FLASH + **8MB** RAM)

Architecture

- 1 JerryScript + 2 Async. I/O event library + 3 Framework + 4 Connectivity



Specs

- Supporting CPU architecture
  - IoT.js code itself does not depend on architecture
  - JerryScript has architecture dependent codes
  - Current: ARMv7l, x86-64. i686

- Memory Requirements
  - 72KB RAM for “Hello world” to console
  - GPIO control requires 128KB RAM
  - Current memory usage is mostly (2/3) for IoT.js built-in objects written in JavaScript
  - Built-in objects can be optimized by rewritten in C.

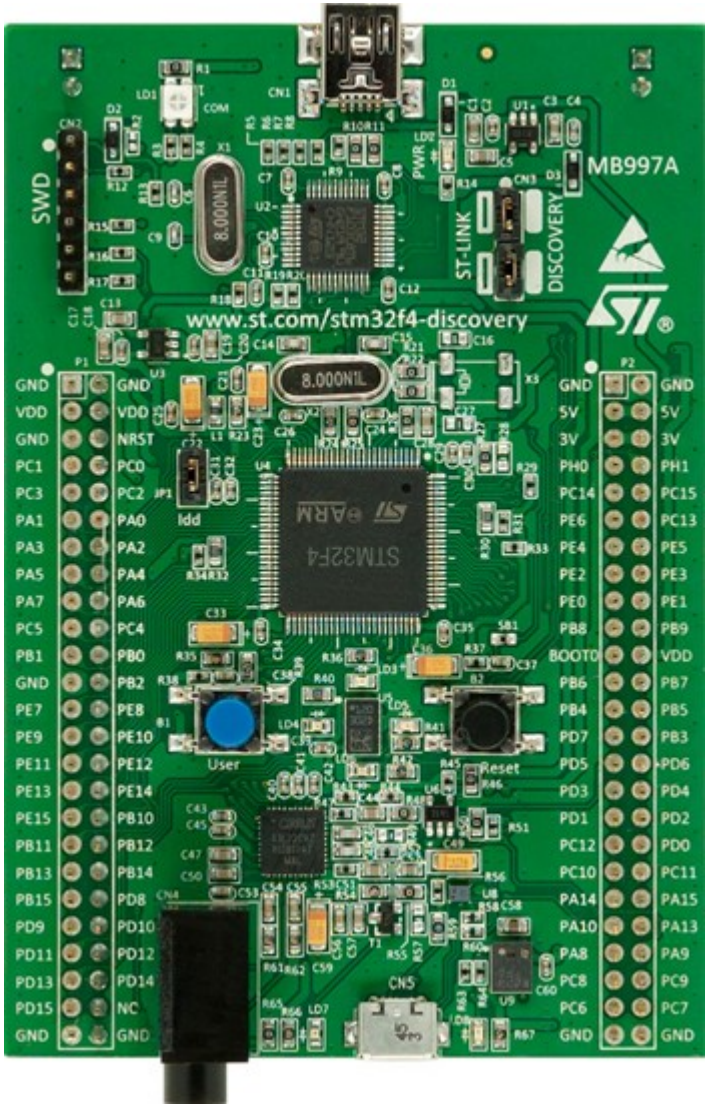
  → Our target is running IoT.js with single IoT application less then 256KB RAM.

- Supporting Target boards
  - STM32F4-Discovery with BB (192K total RAM) with NuttX; our reference board
  - Raspberry Pi 2; for ARM-Linux reference
  - Planning to support ARTIK-1 + nucleus

Reference hardware for developer

- STM32F4 Discovery Board
- Cortex M4F, 168 MHz
  - Freq. is reduced to 100 MHz
- 128KB RAM + 64KB CCM
- 1MB Flash

> make release.mcu_stm32f4-cp

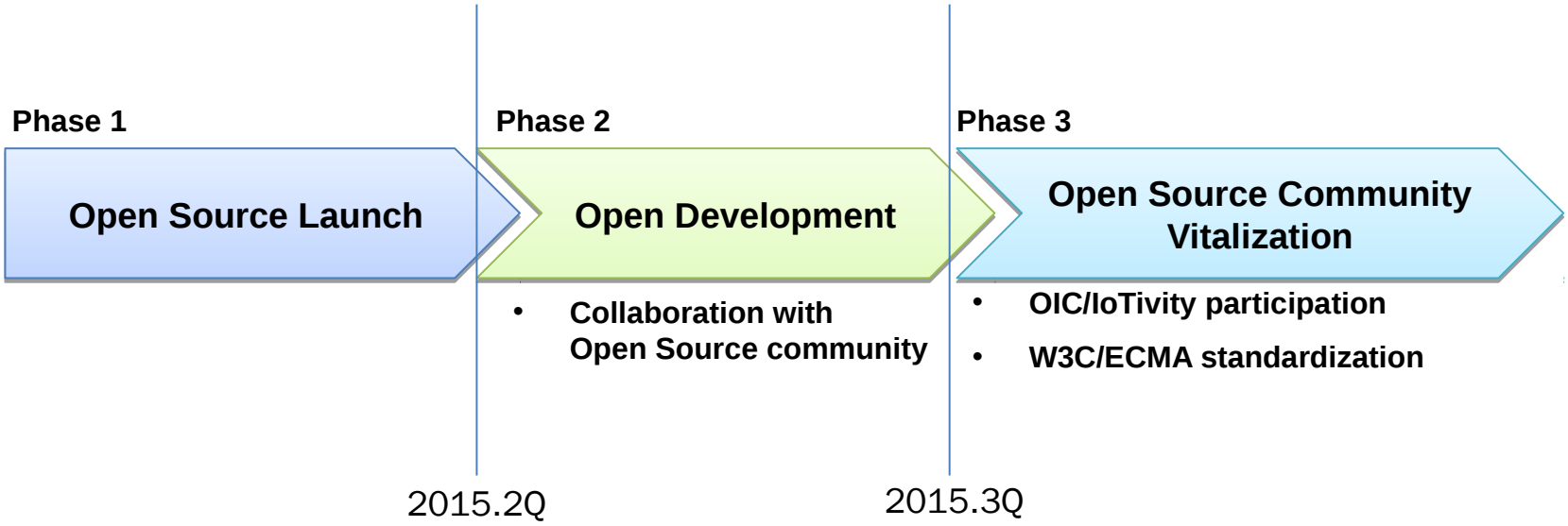


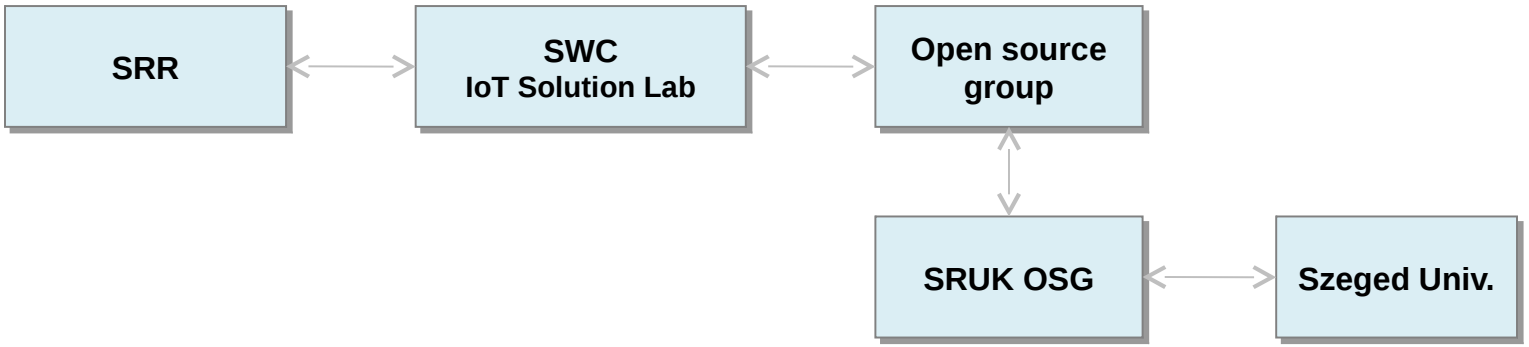
http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419

Key Features

•STM32F407VGT6 microcontroller featuring 32-bit ARM Cortex-M4F core, 1 MB Flash, 192 KB RAM in an LQFP100 package
•On-board ST-LINK/V2 with selection mode switch to use the kit as a standalone ST-LINK/V2 (with SWD connector for programming and debugging)
•Board power supply: through USB bus or from an external 5 V supply voltage
•External application power supply: 3 V and 5 V
•LIS302DL or LIS3DSH ST MEMS 3-axis accelerometer
•MP45DT02, ST MEMS audio sensor, omni-directional digital microphone
•CS43L22, audio DAC with integrated class D speaker driver
•Eight LEDs:
•LD1 (red/green) for USB communication
•LD2 (red) for 3.3 V power on
•Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
•2 USB OTG LEDs LD7 (green) VBus and LD8 (red) over-current
•Two push buttons (user and reset)
•USB OTG FS with micro-AB connector
•Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing

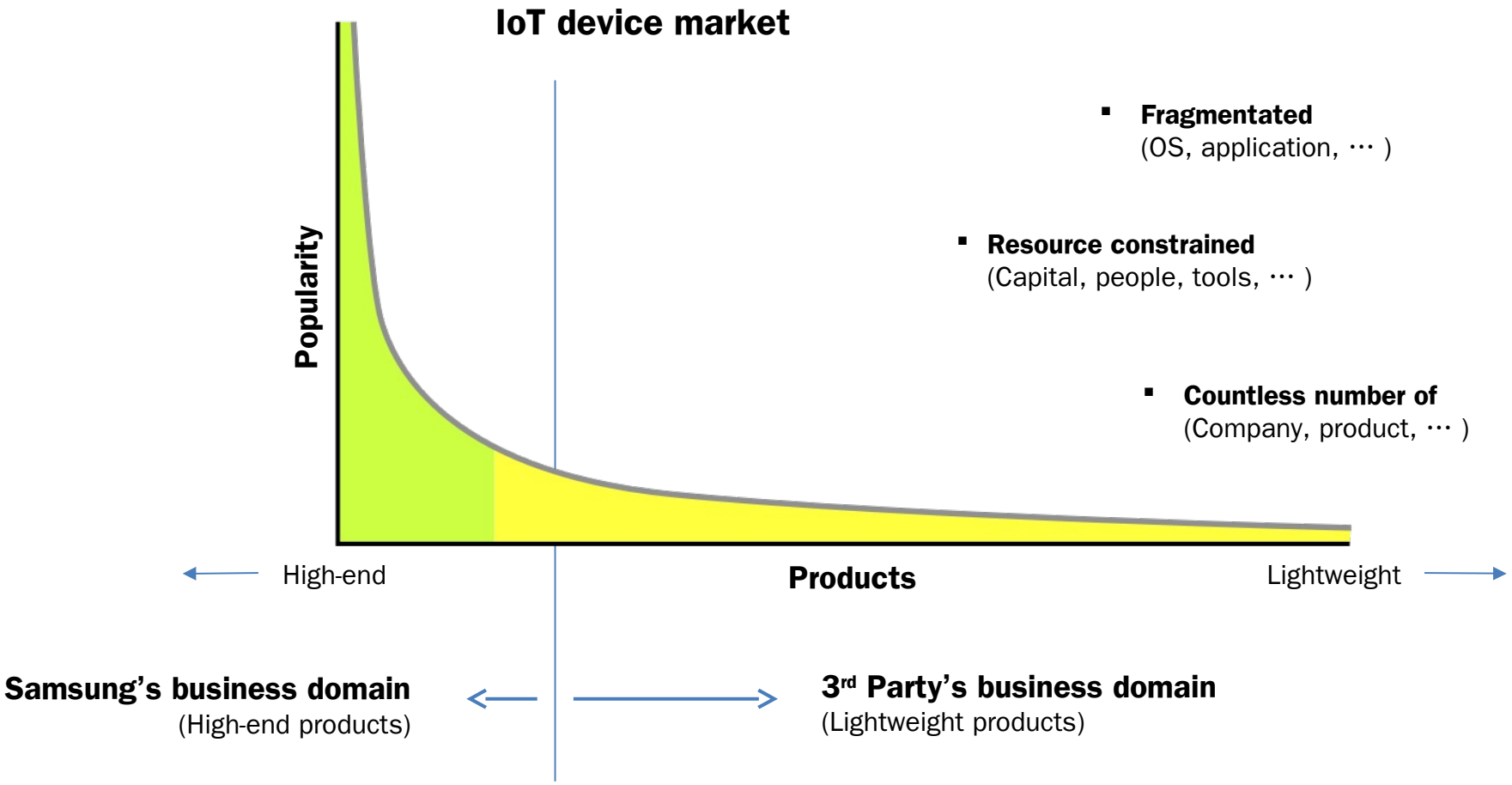
Go open source





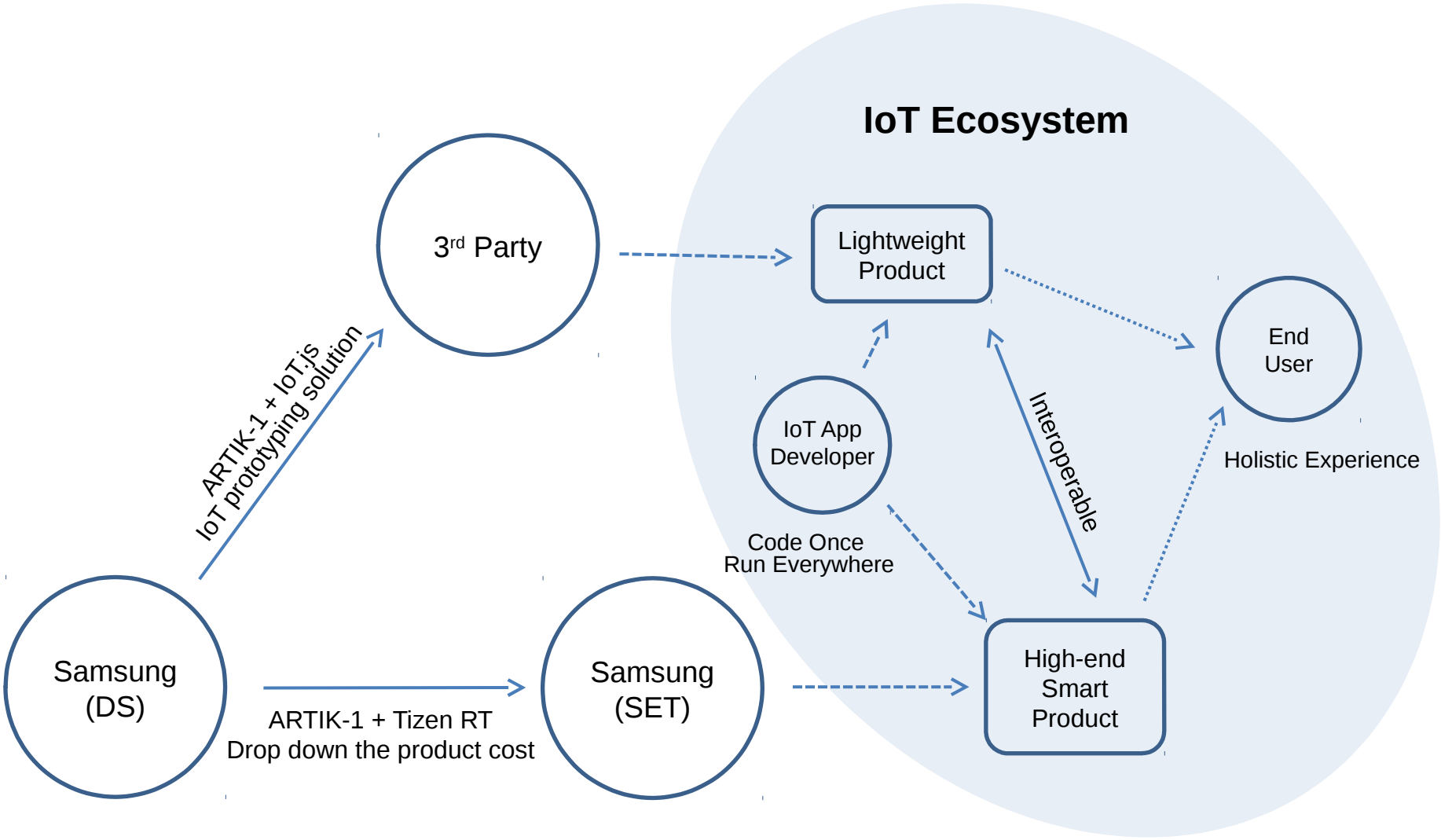
Why in Samsung?

- IoT market is long tail.
- Needs effective solution to interoperate with 3rd party products.

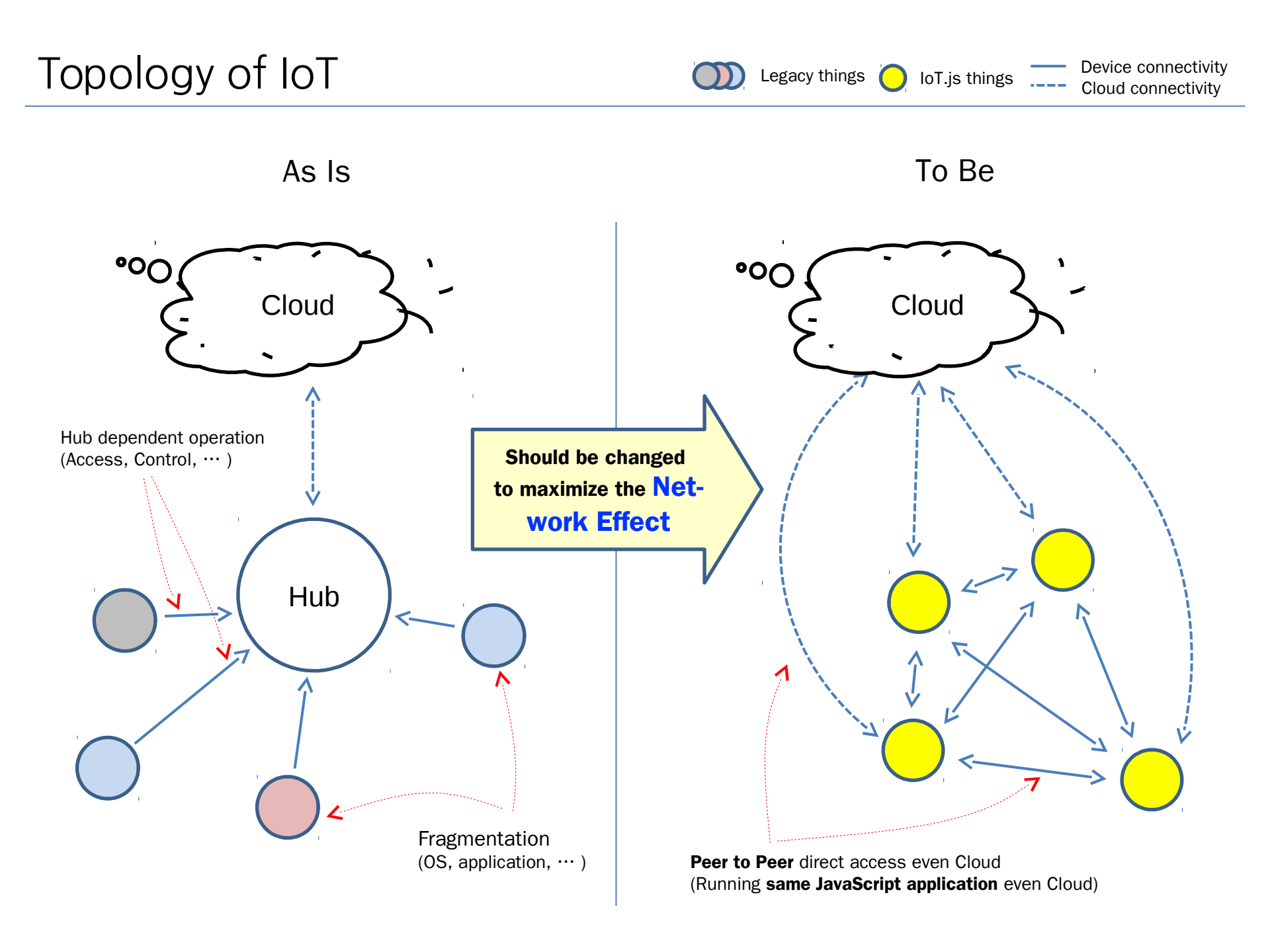


* Image source : https://en.wikipedia.org/wiki/Long_tail

IoT business ecosystem



Topology of IoT



News Scraps

Samsung Pushing IoT.js To Enhance Interoperability

- http://www.androidheadlines.com/2015/07/samsung-pushing-iot-js-enhance-interoperability.html

> Of course, Samsung is not showing us anything final or any products, they are just opening the doors for all kinds of software engineers and inviting them to join this latest movement. *Any friend of the open source idea will rejoice at this news and hopefully many of the gifted developers out there will join Samsung in their endeavor* so we may soon live in an even more comfortable world with intelligent lights, heating, fridges, you name it.

Samsung begins IoT.js development for expanding interoperability to devices

- http://www.sammobile.com/2015/07/21/samsung-begins-iot-js-development-for-expanding-interoperability-to-devices
/

> Looks like Samsung isn't just innovating with their unique hardware, but also with their software. Samsung has recently opened development of a platform called IoT.js which is a platform for the Internet of Things that means to expand the interoperability of lightweight devices.
>
> *This is one of the most exciting projects we've seen from Samsung lately and it could be a total changer.* How do you think it will affect the tech industry once it becomes more widely used?

What's the deal with iot.js and JerryScript

- http://maxogden.com/iotjs-and-jerryscript.html

> The exciting thing about this stuff is that it makes low power hardware more accessible to coders like me who know JS and can install modules from NPM but don't want to deal with C and compiled language tooling and debugging headaches.
>
> *I think a low power Node.js runtime is long overdue and am looking forward to the first stable iot.js release.*

News Scraps

JerryScript & IoT.js: JavaScript for IoT from Samsung

- http://www.infoq.com/news/2015/08/iotjs-jerryscript-samsung

Contacts

If you have any question, please don’t hesitate to contact me.

Sung-Jae Lee / Principal Engineer
IoT Solution Lab., Software Center
SAMSUNG ELECTRONICS Co., Ltd.
sj925.lee@samsung.com

Introduction to JerryScript

(The heart of IoT.js)

July 2015

Software Center
Samsung Electronics

JerryScript

- JerryScript is the lightweight JavaScript engine intended to run on a very constr ained devices such as microcontrollers:
  - Only few kilobytes of RAM available to the engine (<64* KB RAM)
  - Constrained ROM space for the code of the engine (<200 KB ROM)

- Site Links
  - http://www.jerryscript.net
  - https://github.com/Samsung/jerryscript

- Demo Videos
  - https://youtu.be/zhQW6ywO6sM - long version
  - https://youtu.be/JbQvT3hjPec - short version

*Actually, JerryScript core can be run on 8KB memory

JS engine for low memory devices

Subset of JavaScript
- Language leveling based on
  - manual customization
  - target device specification
- **No new syntax** constructs like in
  - TypeScript
  - CoffeeScript
  - WearScript
- Support of built-ins

Access to **peripherals**
- Define **Common Peripherals API**
  - Sensors
  - Actuators
- Vendor chooses what to implement

Requirements:
- Cortex M3/M4/M4F
- Ability to run in a few KB of memory
- **On-device** compilation and execution
- Optional **over-the-air updater**

Target Segment

Segment	Memory Footprint	Binary Size	Configuration
Pico	8KB~16KB	~100KB	Compact JerryScript + Subset Profile
Nano	**16KB~64KB**	**~200KB**	**JerryScript + Full Profile (ES v5.1)**
Micro	64KB~256KB		JerryScript + Bytecode Optimization
Light	256KB~	~300KB	JerryScript with JIT
Full	8MB~	10MB	V8

`2014

`2015

`2016

JerryScript Status

- Embedding C API (in progress, partially supported)
  - http://samsung.github.io/jerryscript/API/
- Designed for devices with constrained resource
  - compact bytecode representation
  - no AST, directly produce bytecode
  - adjustable memory(heap) pool
  - Startup heap size : 2kb(x64 linux)
- Supported Platform
  - Low dependency on platform : own libc library
  - Linux, Nuttx(RTOS), Contiki(RTOS)
- Multi-segment support
  - Compact Profile : limited EMCAScript support
  - Full EMCAScript 5 support (in progress)

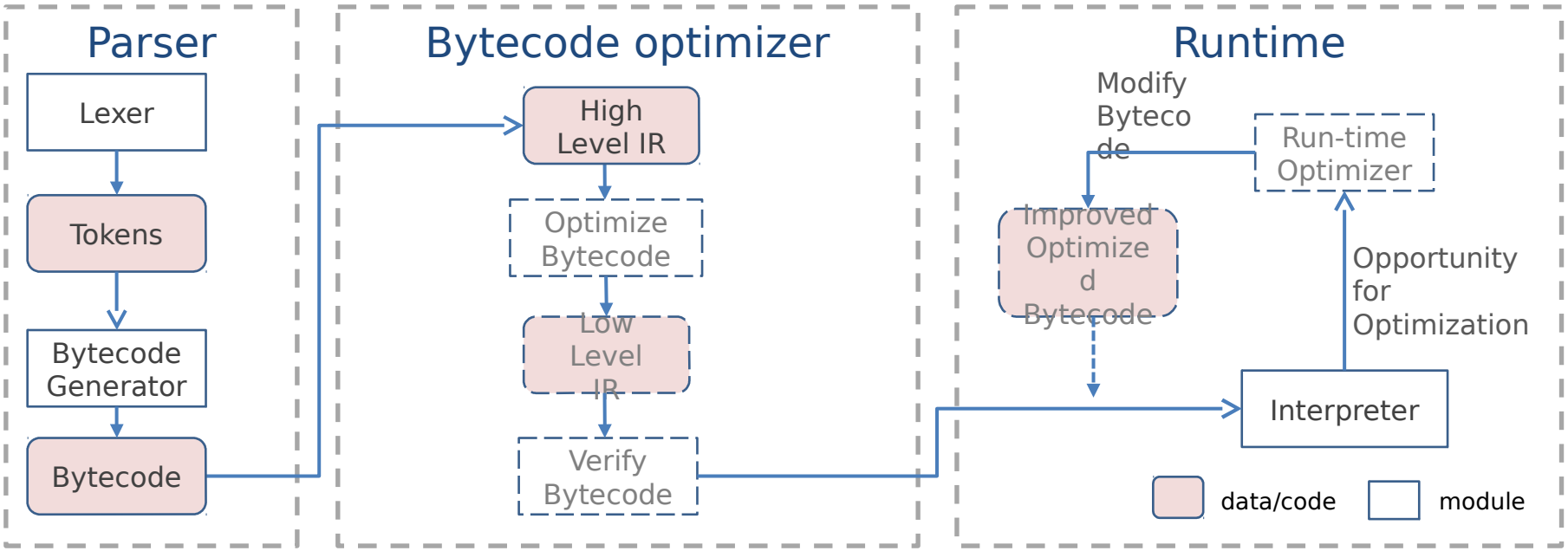
Jerry Engine project goal

- Develop ECMA-262 5.1 compliant small footprint engine:
- Memory consumption can be limited to the specified value
  - this will limit engine's capability to handle memory consuming applications
- Language features can be limited
  - based on chosen memory limit
  - by manual tuning
- Parser should provide full support of ECMA-262 and produce correct byte-code for interpretation
  - optionally develop set of byte-code optimizations
- Define ECMA-262 5.1 subset for Micro Controllers
  - Compact Profile
- Define prototype of Common-API for interaction with peripherals

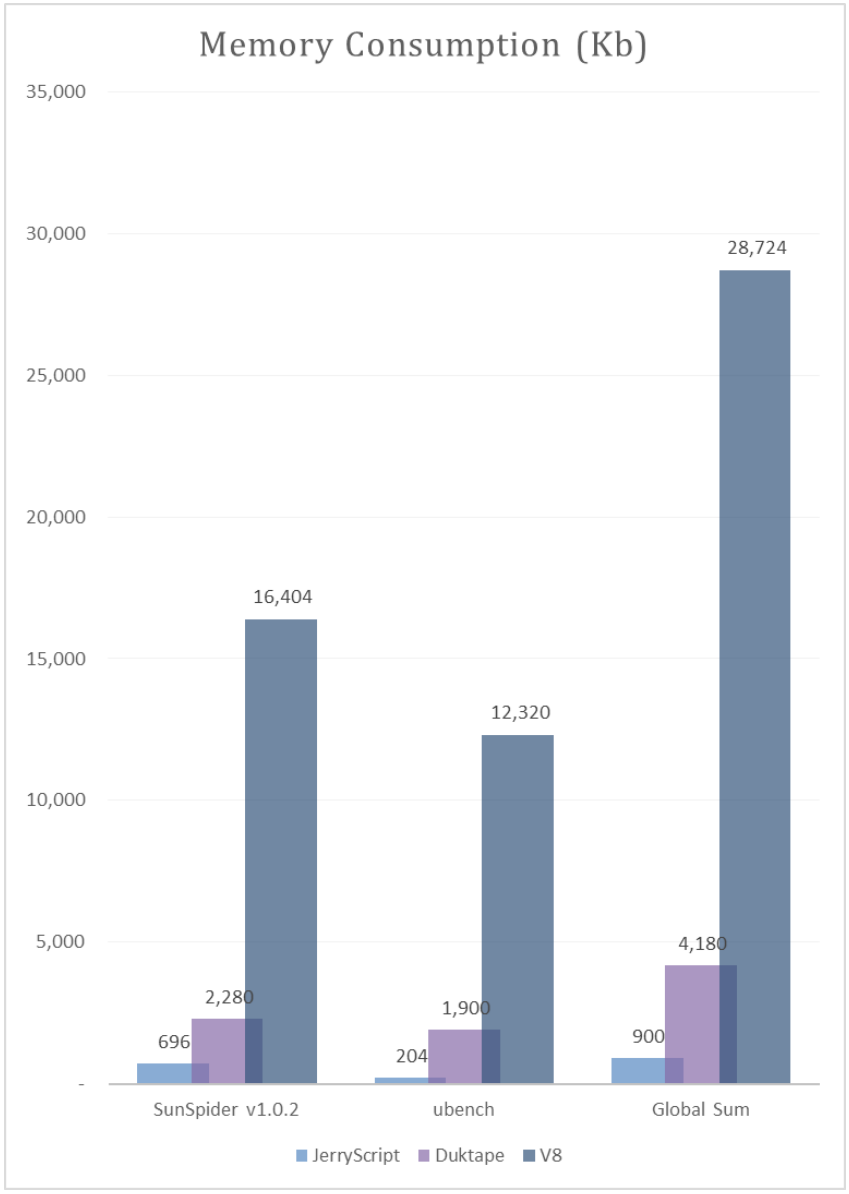
How we provide a small overhead

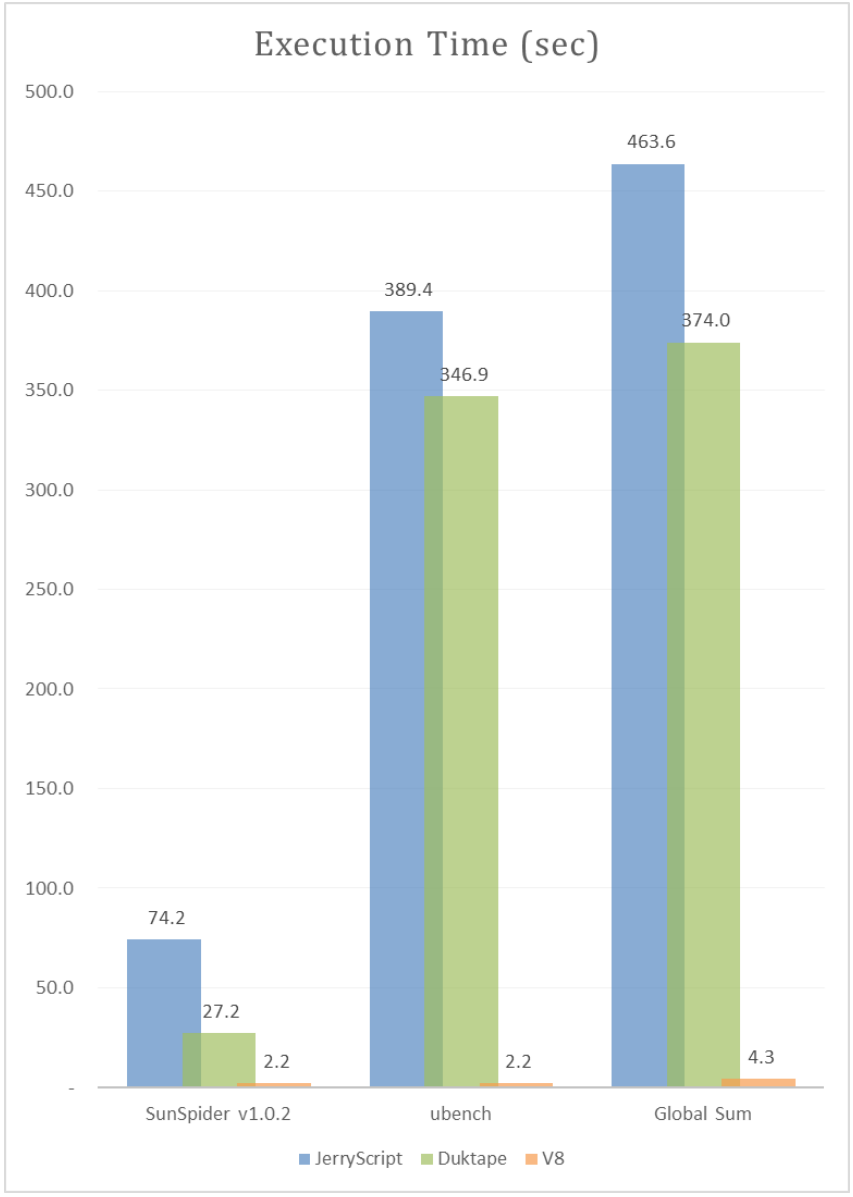
- Jerry is a pure interpreter
  - No overhead for storing compiled code
  - High-level byte-code that simplifies implementation
- Representation of JS objects optimized for size
  - Designed to save memory, not for performance
- Parser doesn’t store AST
  - It produces byte-code directly from the source code
- Parser process code line-by-line
  - It doesn’t store full program in memory

Overall architecture



Performance Comparison (2015.09)





Jerry vs. duktape, V8 benchmark - Average (2015.09)

Benchmark	Memory (Kb)					Performance (sec)				
	JerryScript			Duktape	V8	JerryScript			Duktape	V8
	Old (2015.07)	New (2015.09)	New +Snapshot			Old (2015.07)	New (2015.09)	New +Snapshot		
SunSpider v1.0.2	928	852	696	2,280	16,404	107.9	74.2	74.2	27.2	2.2
ubench	268	276	204	1,900	12,320	400.8	390.9	389.4	346.9	2.2
Global Sum	1,196.0	1,128.0	900.0	4,180.0	28,724.0	508.7	465.1	463.6	374.0	4.3
	1.3	1.3	**1**	**4.6**	**31.9**	117.5	107.4	107.0	86.4	1

```
V8                version 3.14.5.9
JerryScript(old)  abc2b55297eff13538fca2440d127ba79cdcc8b3
JerryScript(new)  311cc65b33a150c1eed055f4a96670922d5478ca
Duktape 1.2.99(v1.2.0-276-g322ccf9)
                  cd2c19761b07e8d675f2079328fbfc9dd9c3b83a
```

* Measured on Raspberry-Pi2

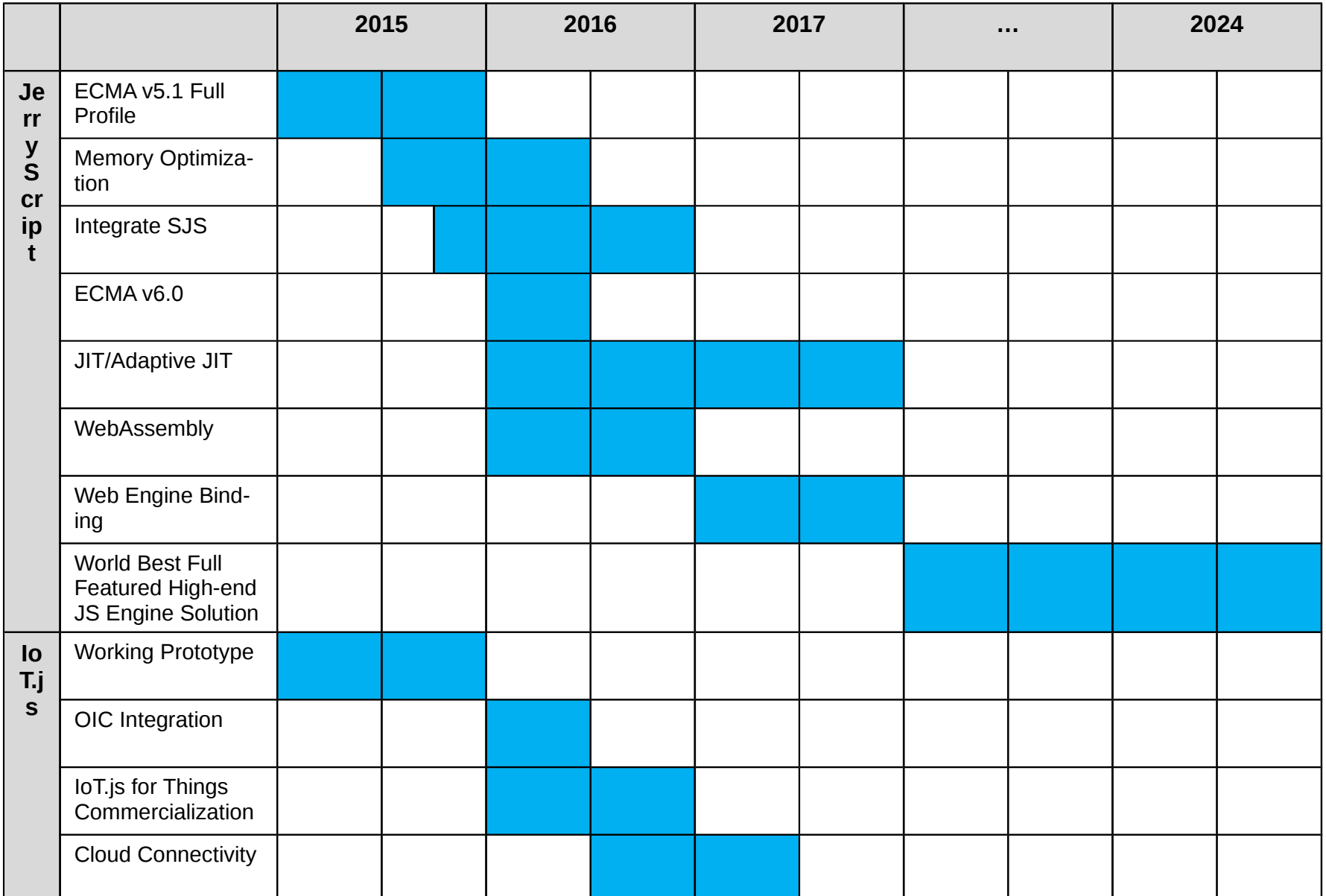
Contacts

If you have any question, please don’t hesitate to contact me.

Sung-Jae Lee / Principal Engineer
IoT Solution Lab., Software Center
SAMSUNG ELECTRONICS Co., Ltd.
sj925.lee@samsung.com

APPENDIX

Roadmap


2015
2016
2017
…
2024
Jerry Script
ECMA v5.1 Full Profile
Memory Optimiza-tion
Integrate SJS
ECMA v6.0
JIT/Adaptive JIT
WebAssembly
Web Engine Bind-ing
World Best Full Featured High-end JS Engine Solution
IoT.js
Working Prototype
OIC Integration
IoT.js for Things Commercialization
Cloud Connectivity

Web of Things Framework

- Expose IoT platforms and devices through the World Wide Web for a Web of Th ings
  - Device abstraction layer to bridge IoT to the Web
- “Things” as proxies for physical and abstract entities
- Modelled in terms of events, properties and actions
  - What events does this thing generate?
    - Someone has just rung the door bell
    - Someone has just inserted a door key
  - What properties does this thing have?
    - Door is open or closed
  - What actions can we invoke on this thing?
    - Unlock the door
  - Thing with on/off property as proxy for a light switch
- With bindings to scripting APIs and protocols
  - Service logic decoupled from underlying communication details

***Dave Raggett**, An introduction to the Web of things Framework, W3C WOT IG, May 2015*
http://www.w3.org/2015/05/wot-framework.pdf

Web of Things Framework

- Standard way to retrieve “thing” descriptions
- Standard format for “thing” descriptions (e.g. JSON-LD)
- Owner, purpose, version, access control, terms & conditions, relationships to ot her things, security best practices, . . .
  - Giving data owners control over who can access their data and for what purposes - co ntract between consumer & supplier
- Semantics and data formats for events, properties & actions
- Properties have discrete values, or smoothly changing values that are interpolat ed between data points, e.g. for robotics
  - Delegating control to where it makes the most sense
  - Clock sync across controllers: 1-10 mS with NTP, and microseconds with IEEE protocols
- Communication patterns
  - Push, pull, pub-sub, and peer to peer
- Bindings to a range of protocols
  - HTTP, Web Sockets, CoAP, MQTT, STOMP, XMPP, WebRTC

***Dave Raggett**, An introduction to the Web of things Framework, W3C WOT IG, May 2015*
http://www.w3.org/2015/05/wot-framework.pdf

Compact Profile Limitations Idea

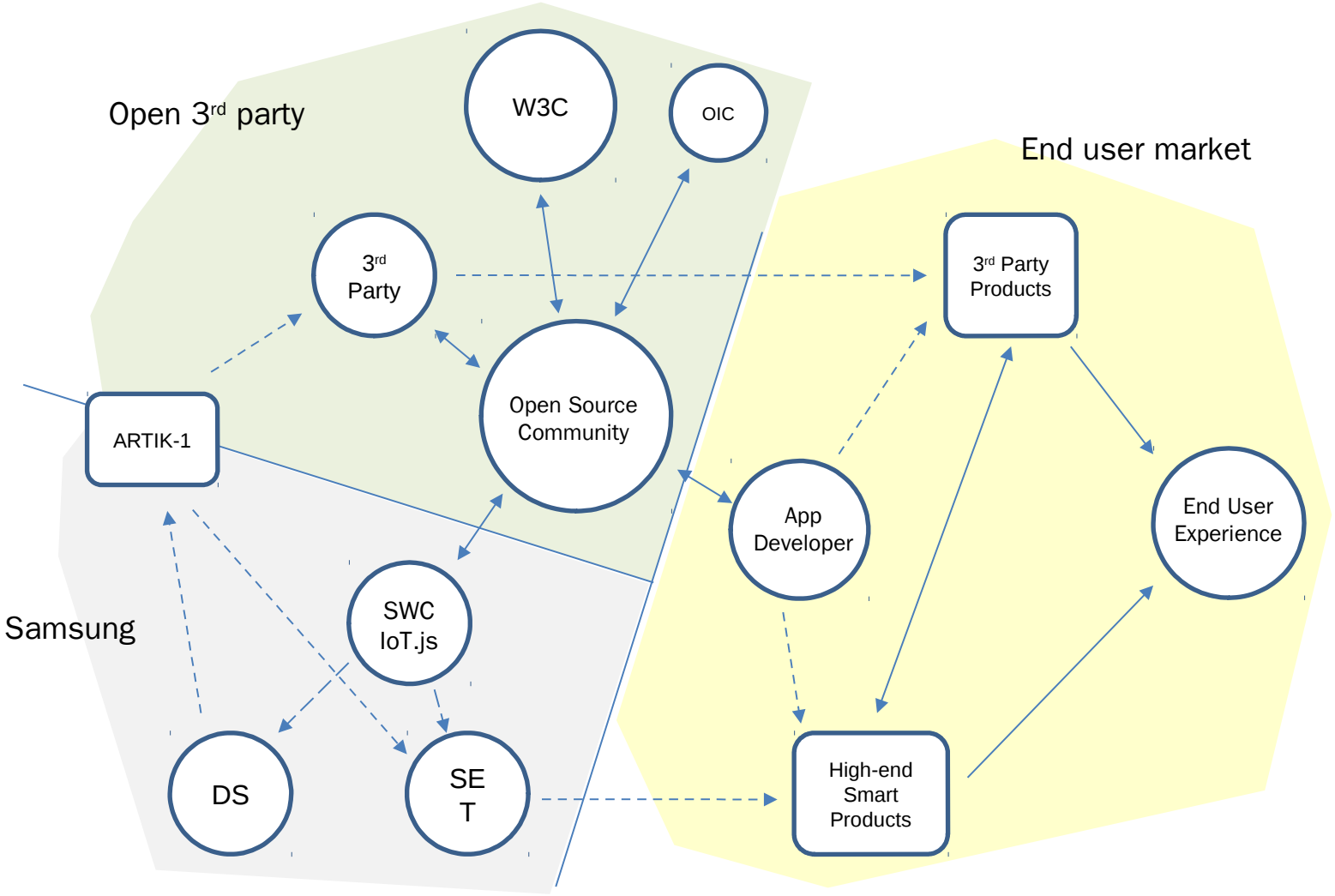
- Prohibit eval() usage, same as Function or new Function
  - eval() requires run time compilation because source text might not appear in program or can be generated at run time.
- Limit Unicode support
  - UTF-16 -> UTF-8
- Cut representation of Number type from 64bit to 32bit/16/8bit
  - double -> float, cortex m4f has only float support
- Immutable strings
  - No mutation or re-creation
- Limit recursion level
- Prohibit addition, deletion or assignment to the properties of built-in objects
  - This limitation is needed to support a more efficient implementation based on static compilation of built-in objects without risking that objects are mutated on shadowed by dynamically added properties
- Do not allow usage of with
  - with makes access to named references inefficient, because the scopes for such access cannot be compiled until run time

Compact Profile Limitations

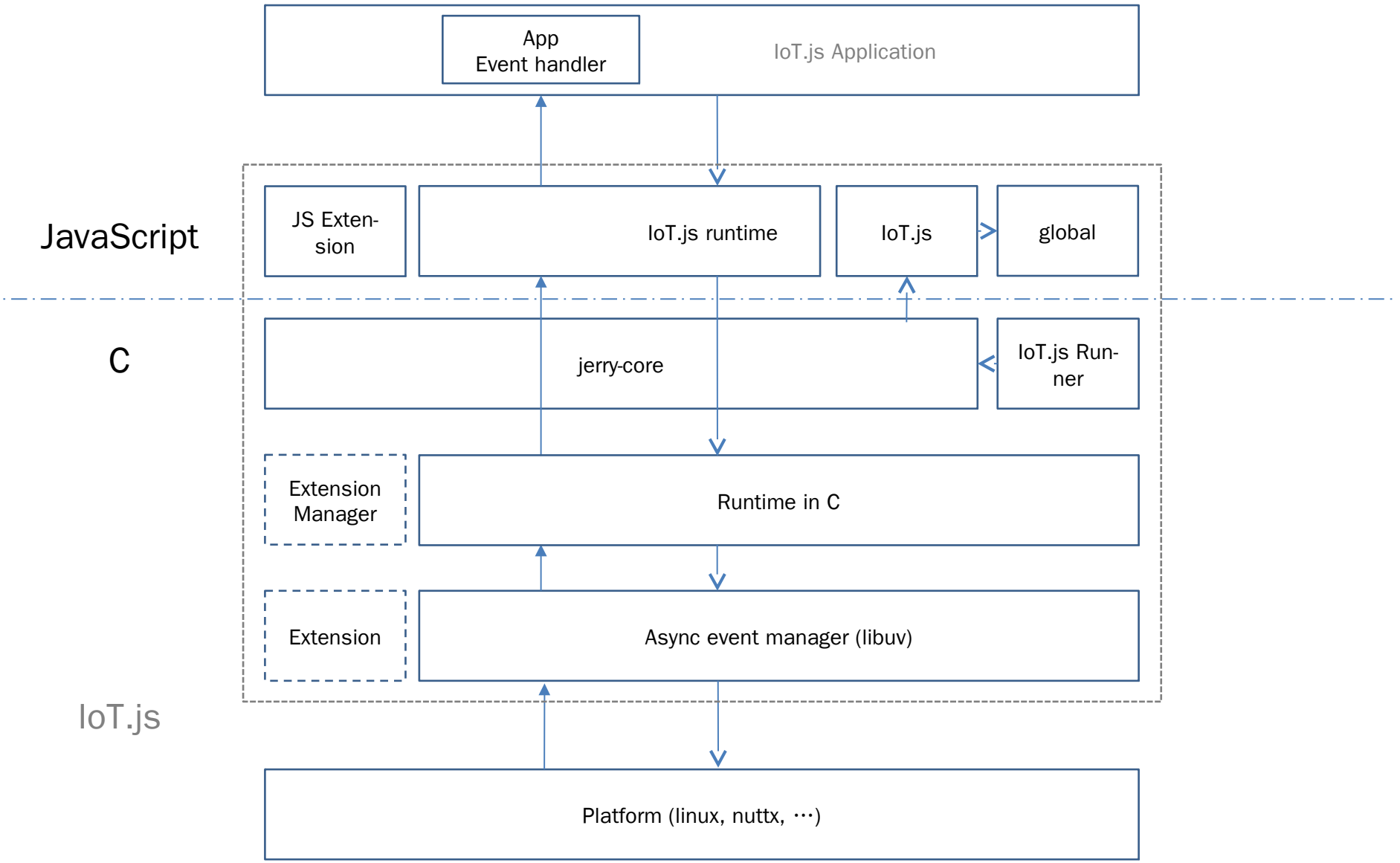
- NOT REQUIRED to support
  - Implicit declaration of variables
  - Dynamic typing
  - Modification of built-in objects
  - All built-in objects
  - Function calls with number of arguments that differs from function declaration
  - Declaration functions inside the function's body
  - Equality operators "==", and "!="
  - "with" statement
  - "eval()" method
  - Accessor properties (e.g. getters and setters)
  - Debugger statements
  - Labelled statements
  - Strict and non-strict execution mode

- COULD support
  - Limit the maximum number of elements in Array and String variables
  - Various number types such as
  - INT8, INT16, INT32, INT64, FLOAT8, FLOAT16, FLOAT32, FLOAT64
  - Switching default encoding between ASCII, UTF-8, UCS-2 and UTF-16

WoT business ecosystem



Architecture



Jerry vs. duktape, V8 benchmark - SunSpider v1.0.2 (2015.09)

Benchmark	Memory (Kb)					Performance (sec)				
	JerryScript			Duktape	V8	JerryScript			Duktape	V8
./tests/sunspider-1.0.2/*	Old (2015.07)	New (2015.09)	New +Snapshot			Old (2015.07)	New (2015.09)	New +Snapshot		
3d-cube.js	172	140	104	220	1,552	6.6	3.7	3.7	1.1	0.2
access-binary-trees.js	92	88	76	240	708	3.4	2.8	2.8	1.3	0.1
access-fannkuch.js	56	40	28	176	1,268	16.1	9.8	9.8	2.0	0.2
access-nbody.js	68	64	44	184	3,112	7.0	4.6	4.6	1.8	0.2
bitops-3bit-bits-in-byte.js	36	36	28	176	268	5.3	3.4	3.3	0.6	0.1
bitops-bits-in-byte.js	36	36	28	176	292	7.9	4.5	4.5	0.9	0.1
bitops-bitwise-and.js	28	32	24	176	640	5.2	4.2	4.1	7.2	0.2
controlflow-recursive.js	244	220	220	216	268	4.5	3.3	3.3	1.3	0.2
math-cordic.js	48	48	32	176	1,456	8.6	4.9	4.9	2.4	0.2
math-partial-sums.js	40	40	28	176	3,068	3.4	2.6	2.6	2.8	0.2
math-spectral-norm.js	52	52	40	176	1,656	4.7	3.2	3.2	1.0	0.2
string-fasta.js	56	56	44	188	2,116	35.3	27.3	27.4	4.7	0.2
Sum	928	852	696	2,280	16,404	107.9	74.2	74.2	27.2	2.2
	1.3	1.2	**1.0**	**3.3**	**23.6**	49.7	34.2	34.1	12.5	1.0

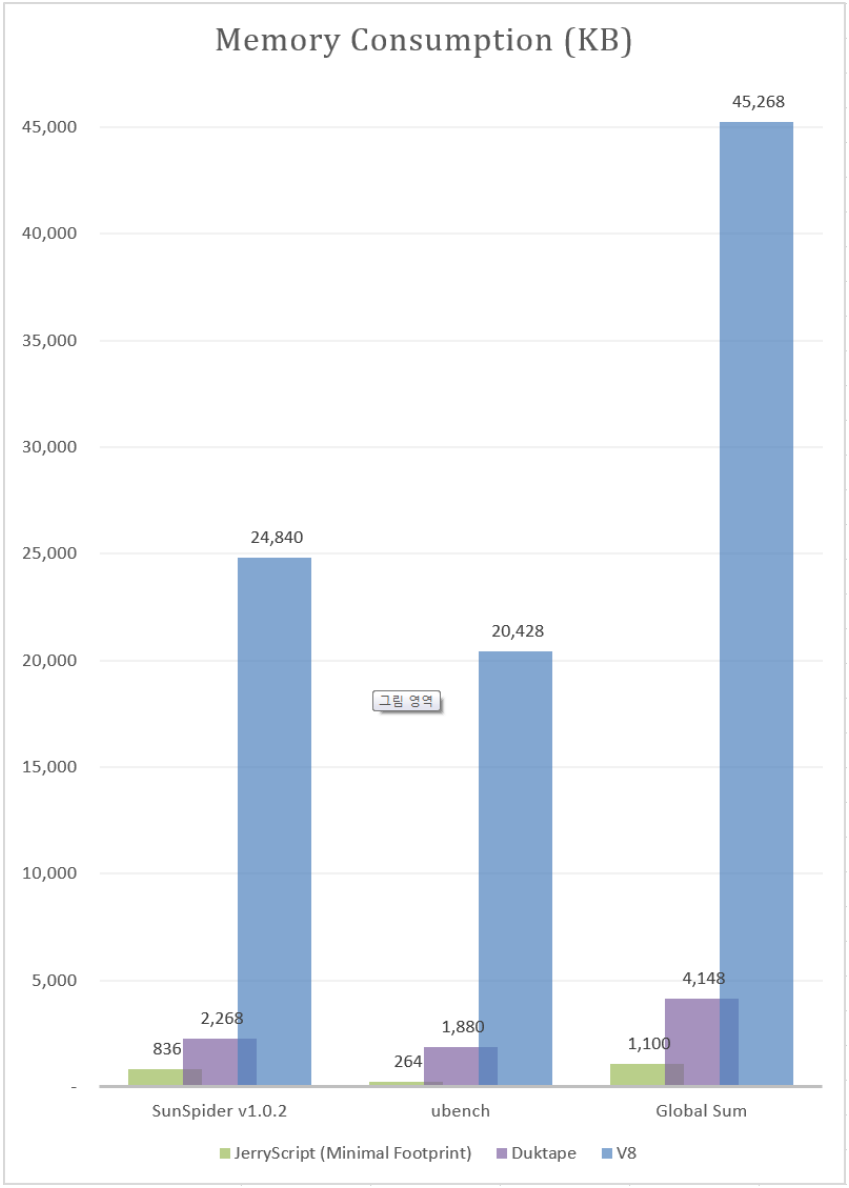
* Measured on Raspberry-Pi2

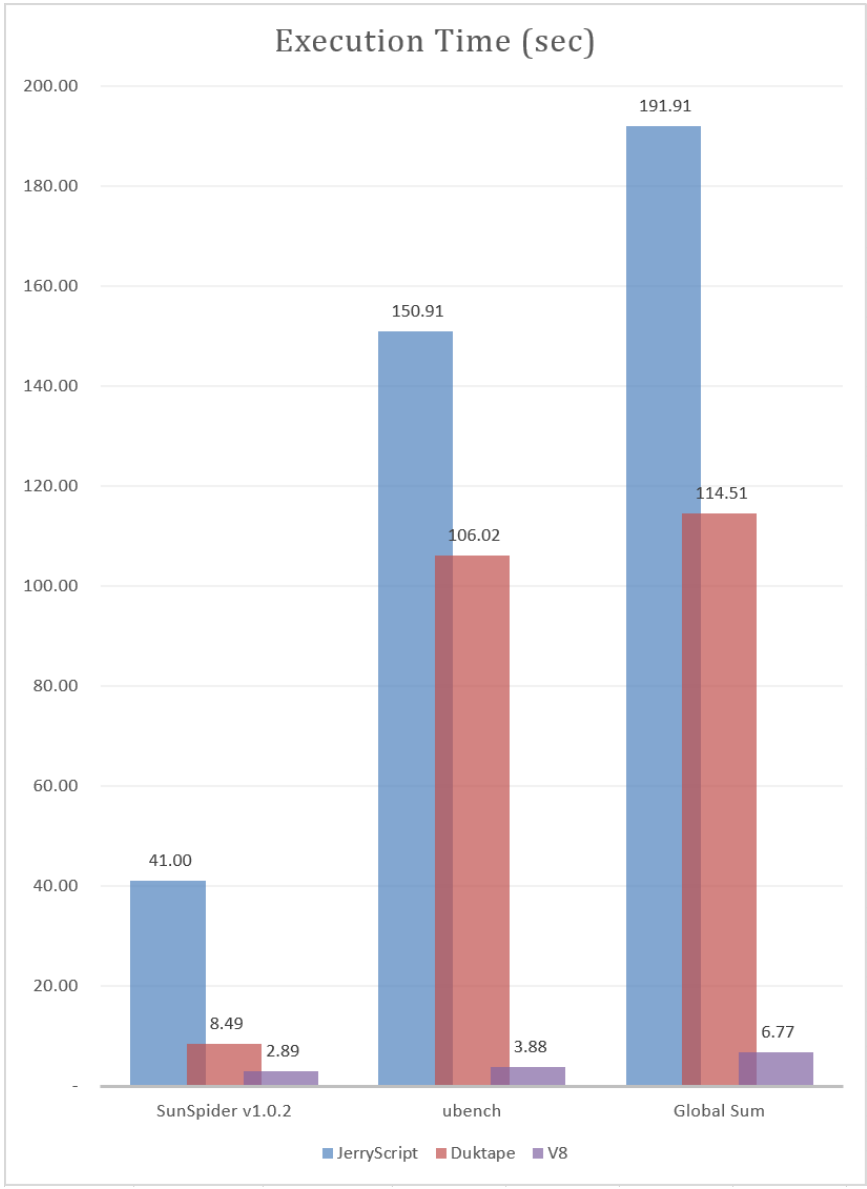
Jerry vs. duktape, V8 benchmark - ubench (2015.09)

Benchmark	Memory (Kb)					Performance (sec)				
	JerryScript			Duktape	V8	JerryScript			Duktape	V8
./tests/ubench/*	Old (2015.07)	New (2015.09)	New +Snapshot			Old (2015.07)	New (2015.09)	New +Snapshot		
function-closure.js	32	32	24	504	2,832	5.3	3.5	3.6	19.7	0.2
function-correct-args.js	32	32	24	176	1,244	87.7	93.9	91.9	37.6	0.2
function-empty.js	32	32	24	176	1,244	31.9	28.9	28.9	42.6	0.2
function-excess-args.js	32	32	24	176	1,244	62.7	66.0	66.2	35.0	0.2
function-missing-args.js	32	32	24	176	1,244	65.0	66.4	66.3	31.6	0.2
function-sum.js	32	32	24	176	1,244	48.6	45.3	45.5	28.7	0.2
loop-empty-resolve.js	24	28	20	172	780	5.0	4.6	4.7	8.0	0.2
loop-empty.js	24	28	20	172	1,244	43.7	39.6	39.5	58.8	0.3
loop-sum.js	28	28	20	172	1,244	50.9	42.8	43.0	84.9	0.4
Sum	268	276	204	1,900	12,320	400.8	390.9	389.4	346.9	2.2
	1.3	1.4	**1.0**	**9.3**	**60.4**	185.7	181.1	180.4	160.7	1.0

* Measured on Raspberry-Pi2

Performance Comparison (2015.07)





Jerry vs. duktape, V8 benchmark - Average (2015.07)

Benchmark	Memory (Kb)					Performance (sec)		
	JerryScript			Duktape	V8	JerryScript (Full)	Duktape	V8
	(Full)	(Compact Profile)	(Minimal Footprint)					
SunSpider v1.0.2	964	940	836	2,268	24,840	41	8	3
ubench	300	300	264	1,880	20,428	151	106	4
Global Sum	1,264	1,240	1,100	4,148	45,268	192	115	7
	1.15	1.13	**1**	**3.8**	**41.2**	28.4	16.9	1

```
V8    b2ed25304e203bbd22d6b09db575980f6aecf30a
Jerry abc2b55297eff13538fca2440d127ba79cdcc8b3
duk   cd2c19761b07e8d675f2079328fbfc9dd9c3b83a
```

* Measured on arndale board, on arm x32 linux

Jerry vs. duktape, V8 benchmark - SunSpider v1.0.2 (2015.07)

Benchmark	Memory (Kb)					Performance (sec)		
./tests/sunspider-1.0.2/*	JerryScript			Duktape	V8	JerryScript (Full)	Duktape	V8
	(Full)	(Compact Profile)	(Minimal Footprint)					
3d-cube.js	172	148	128	224	1,788	2.59	0.38	0.26
access-binary-trees.js	96	100	88	240	1,880	1.36	0.36	0.22
access-fannkuch.js	64	64	52	176	2,164	6.07	0.65	0.23
access-nbody.js	84	84	72	184	2,064	2.79	0.49	0.23
bitops-3bit-bits-in-byte.js	44	44	36	172	2,068	2.24	0.19	0.22
bitops-bits-in-byte.js	40	40	32	172	2,340	3.18	0.29	0.23
bitops-bitwise-and.js	32	32	28	172	1,892	2.01	2.33	0.26
controlflow-recursive.js	208	204	208	216	2,228	1.92	0.37	0.22
math-cordic.js	52	52	48	176	2,164	3.38	0.80	0.23
math-partial-sums.js	48	48	44	172	1,864	1.55	0.96	0.26
math-spectral-norm.js	56	56	44	176	2,296	1.84	0.32	0.24
string-fasta.js	68	68	56	188	2,092	12.07	1.35	0.28
Sum	964	940	836	2,268	24,840	41	8	3
	1.2	1.1	**1.0**	**2.7**	**29.7**	14.2	2.9	1

* Measured on arndale board, on arm x32 linux

Jerry vs. duktape, V8 benchmark - ubench (2015.07)

Benchmark	Memory (Kb)					Performance (sec)		
./tests/ubench/*	JerryScript			Duktape	V8	JerryScript (Full)	Duktape	V8
	(Full)	(Compact Profile)	(Minimal Footprint)					
function-closure.js	32	32	28	504	1,900	1.90	5.73	0.24
function-correct-args.js	36	36	32	172	2,196	33.80	9.73	0.43
function-empty.js	32	32	28	172	2,196	11.93	12.83	0.45
function-excess-args.js	36	36	32	172	2,272	24.25	10.37	0.47
function-missing-args.js	32	32	28	172	2,516	25.37	8.63	0.50
function-sum.js	36	36	32	172	2,324	18.14	8.64	0.38
loop-empty-resolve.js	32	32	28	172	2,624	1.86	2.18	0.22
loop-empty.js	32	32	28	172	2,184	16.23	19.17	0.56
loop-sum.js	32	32	28	172	2,216	17.43	28.76	0.65
Sum	300	300	264	1,880	20,428	151	106	4
	1.1	1.1	**1.0**	**7.1**	**77.4**	38.9	27.3	1

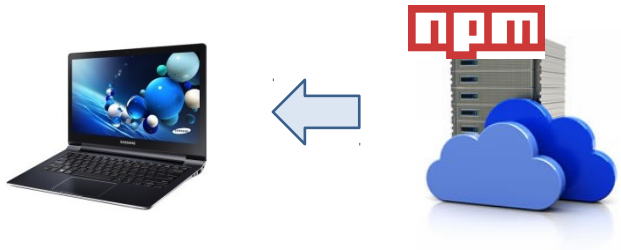
* Measured on arndale board, on arm x32 linux

DEMO

Demo #1 - IoT.js

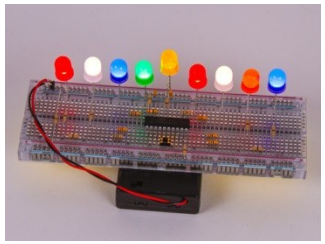
- Scenario
  - Run LED blinking demo led.js on RPi2 with node.js
  - Run same LED blinking demo led.js on RPi2 with iot.js
  - Again run same led.js on STM32F4 board with iot.js
- Video
  - https://youtu.be/FLnT129j64c

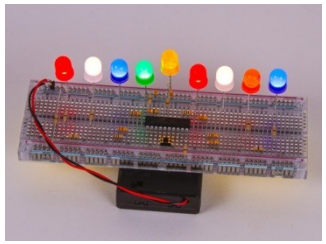
Demo #1 - IoT.js



RTOS on STM32F4

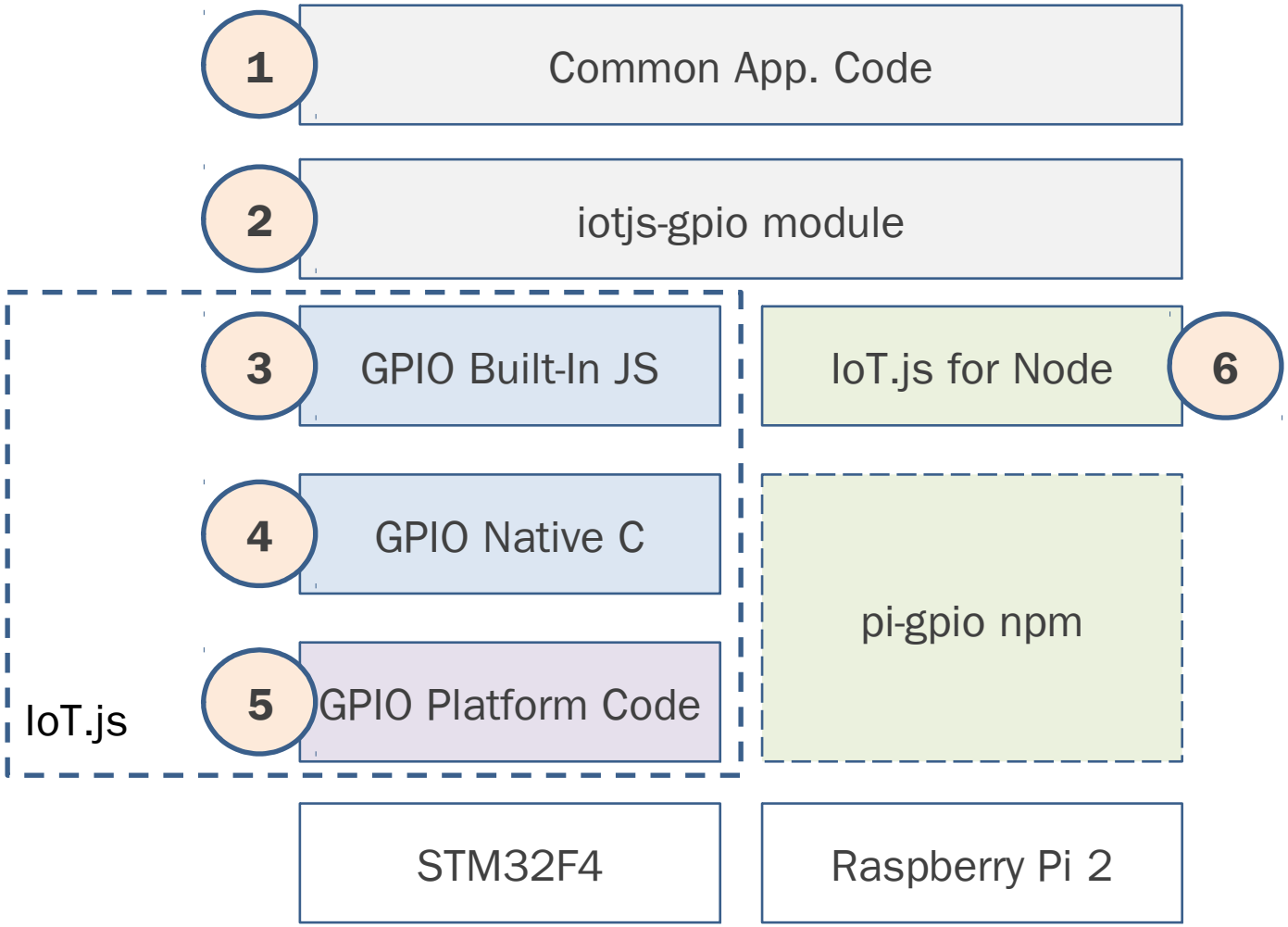
Linux on Raspberry Pi 2





Demo #1 - IoT.js

- Code structure



Sample code

- Common function “run( )” for both iot.js and node.js

1

```
var gpio = require("iotjs-gpio")

function gpio_run() {
  var on = 1;
  var idx = 0;

  console.log("start blinking...");
  intervalId = setInterval(function() {
    var portpin = gpiocfg.map(gpio_pout[idx]);
    var err = gpio.write(portpin, on);
    idx = idx + 1;
    if (idx >= gpio_ocnt) {
      idx = 0;
      on = (on + 1) % 2;
    }
  }, 100);
}
```

Sample code

- iotjs-gpio module to switch iot.js or node.js

2

```
if (process.iotjs) {
  module.exports = require('gpio');
}
else {
  module.exports = require('./node-gpio.js')
}
```

Sample code

- Built-in gpio.js that calls native gpioctl

3

```
var gpioctl = process.binding(process.binding.gpioctl)

GPIO.write = function(portpin, val, callback) {
  var err = gpioctl.writepin(portpin, val);

  if (util.isFunction(callback)) {
    process.nextTick(function() {
      callback(err);
    });
  }
  else {
    return err;
  }
};
```

- Current version is SYNC which emulates ASYNC

Sample code

- Native gpioctl code; Binding JS “writepin()” to C “WritePin()”

4

```
JHANDLER_FUNCTION(WritePin, handler) {
  JObject* jgpioctl = handler.GetThis();
  GpioControl* gpioctrl =
                GpioControl::FromJGpioCtl(*jgpioctl);

  uint32_t portpin = handler.GetArg(0)->GetInt64();
  uint8_t data = (uint8_t)handler.GetArg(1)-
>GetInt32();
  int err = gpioctrl->WritePin(portpin, data);
  …
}

JObject* InitGpioCtl() {
  …
  jgpioctl = new JObject();
  jgpioctl->SetMethod(JSCT("writepin"), WritePin);
  …
}
```

Sample code

- Target platform dependent code; for NuttX/STM32F4

5

```
int GpioControlInst::WritePin(uint32_t portpin,
                              uint8_t data) {
  if (_fd > 0) {
    struct gpioioctl_write_s wdata;
    wdata.port = portpin;
    wdata.data = data;
    return ioctl(_fd, GPIOIOCTL_WRITE,
                 (long unsigned int)&wdata);
  }
  return IOTJS_GPIO_NOTINITED;
}
```

Sample code

- for node.js, for gpio, use “pi-gpio” module


```
var pi_gpio = require('pi-gpio');

GPIO.write = function(port, val, callback) {
  pi_gpio.write(port, val, (callback || noop));
};
```