Samsung Open Source Group

JerryScript

An ultra-lightweight JavaScript engine for the Internet of Things

Tilmann Scheller
Principal Compiler Engineer
t.scheller@samsung.com

Samsung Open Source Group
Samsung Research UK

OpenIoT Summit Europe 2016
Berlin, Germany, October 11 – 13, 2016

Overview

- Introduction
- JerryScript
- JerryScript Internals Overview
- Memory consumption/Performance
- Demo
- Future work
- Summary

Introduction

What is JerryScript?

- A really lightweight JavaScript engine
- Has a base footprint of ~3KB of RAM
- Optimized for microcontrollers
- Originally developed from scratch by Samsung
- JerryScript is an open source project released under the Apache License 2.0
- Actively developed on GitHub

Why JavaScript on microcontrollers?

- There's a huge pool of JavaScript developers
- Opens up the possibility for web developers to easily write software for embedded devices
- Performance overhead of JavaScript less of an issue for control tasks
- Increased productivity, shorter time to market
- Ability to load code dynamically over the network
- Security: Executing JavaScript code is safer than executing arbitrary native code

JerryScript

JerryScript History

- Development started in June 2014
- Released as open source in June 2015
- JerryScript passed 100% of the test262 conformance test suite in August 2015
- Rewritten compact byte code implementation landed in January 2016
- JerryScript 1.0 released in September 2016
- Current focus on usability

JerryScript

- Heavily optimized for a low memory footprint
- Interpreter-only
- Compact object representation
- Compressed pointers
- No AST, directly creating byte code
- Compact byte code heavily optimized for low memory consumption

JerryScript Portability

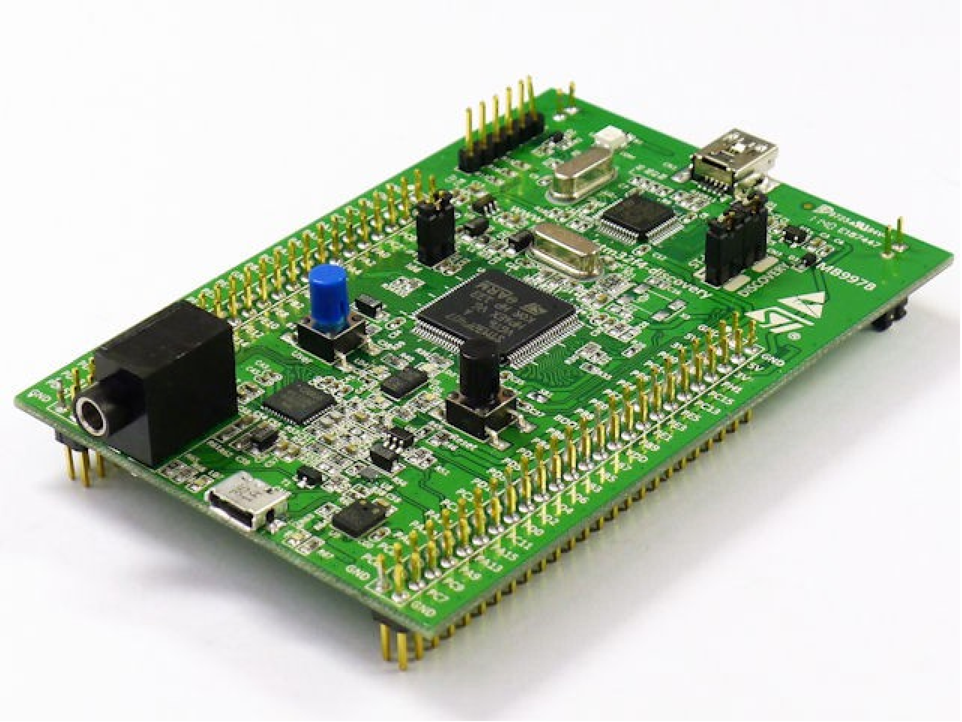
- Extremely portable
- Self-contained
- Small C library
- Can run bare-metal
- Supports the STM32F4, Arduino 101, FRDM-K64F, ESP8266 (experimental) boards
- OS support: NuttX, Zephyr, mbed OS, RIOT
- Runs on Linux/macOS as well

JerryScript

- Written in C99
- About 84KLOC
- Code size 156KB when compiled with GCC in LTO mode for ARM Thumb-2
- Implements the entire ECMAScript 5.1 standard, passes 100% of the test262 conformance test suite
- C API for embedding JerryScript
- Byte code snapshot feature

Target hardware

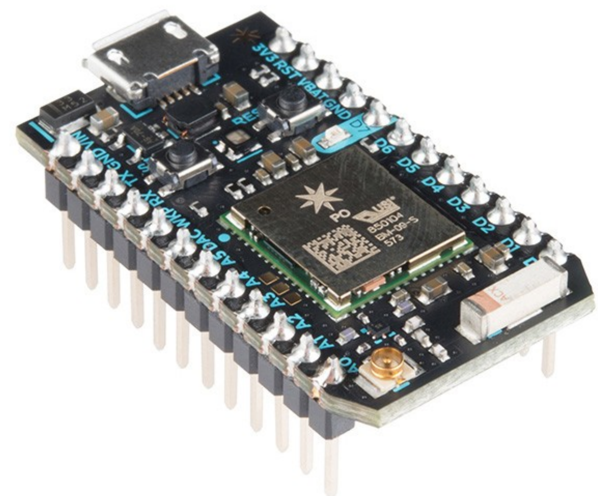
- STM32F4 developer board
- Cortex-M4F clocked at 168 MHz
- 192KB of RAM
- 1MB of flash memory



SAMSUNG
Open Source Group

Target hardware

- Particle Photon board
- Cortex-M3 clocked at 120 MHz
- 128KB of RAM
- 1MB of flash memory
- Wi-Fi integrated
- Small footprint (37mm x 20mm)



JerryScript C API

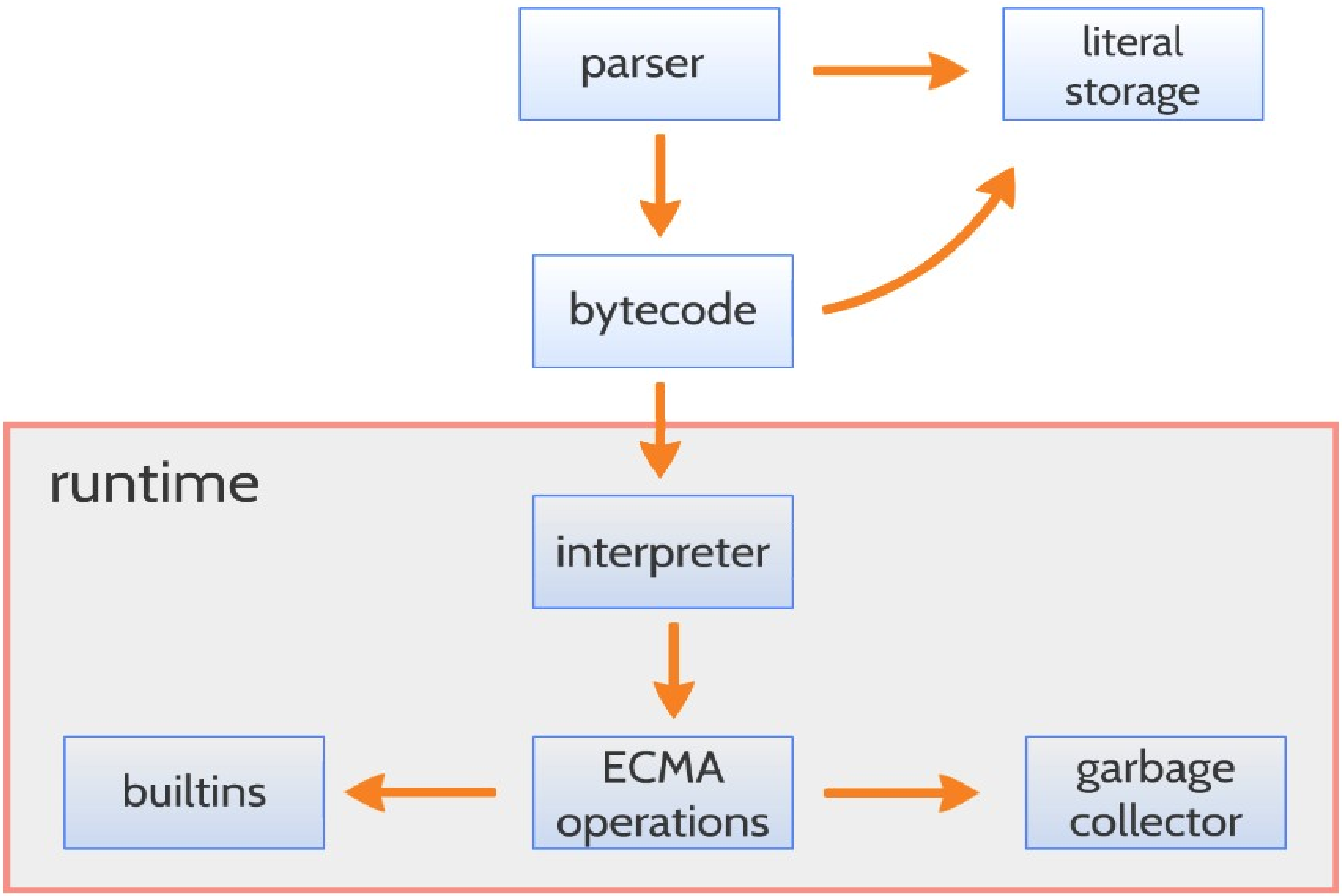

```
#include <string.h>
#include "jerry.h"

int
main (int argc, char * argv[]) {
  char script [] = "print ('Hello, World!');";

  jerry_completion_code_t code = jerry_run_simple (script,
                                                   strlen (script),
                                                   JERRY_FLAG_EMPTY);
}
```

JerryScript C API


```
int
main (int argc, char * argv[]) {
  char script1 [] = "var s = 'Hello, World!';";
  char script2 [] = "print (s);";

  // Initialize engine
  jerry_init (JERRY_FLAG_EMPTY);

  jerry_api_value_t eval_ret;

  // Evaluate script1
  jerry_api_eval (script1, strlen (script1),
                  false, false, &eval_ret);
  // Free JavaScript value, returned by eval
  jerry_api_release_value (&eval_ret);

  // Evaluate script2
  jerry_api_eval (script2, strlen (script2),
                  false, false, &eval_ret);
  // Free JavaScript value, returned by eval
  jerry_api_release_value (&eval_ret);

  // Cleanup engine
  jerry_cleanup ();
}
```

JerryScript Internals Overview

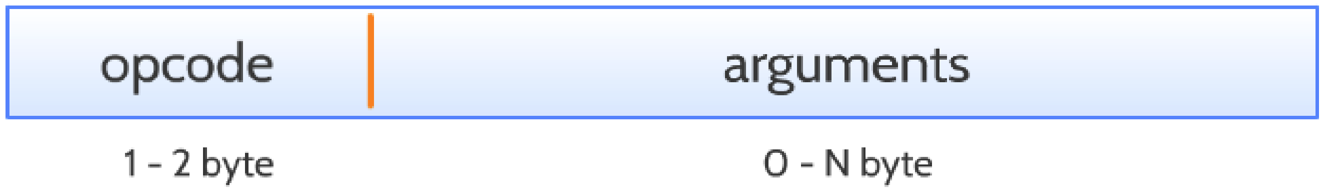
High-Level Design Overview



Parser Overview

- Optimized for low memory consumption
  - E.g. only 41KB of memory is required to parse the 95KB of concatenated IoT.js source code
    - 12.5KB byte code, 10KB literal references, 12.2KB literal storage data, 7KB for parser temporaries
- Generates byte code directly
  - No intermediate representation (e.g. AST)
- Recursive descent parser
  - The parser uses a byte array for the parser stack instead of calling functions recursively

Compact Byte Code (CBC)



- CBC is a variable-length byte code
- Currently 306 opcodes are defined
  - Majority of the opcodes are variants of the same operation
- E.g. “this.name” is a frequent expression in JavaScript so an opcode is defined to resolve this expression
  - Usually this operation is constructed from multiple opcodes: op_load_this, op_load_name, op_resolve
  - Other examples: “a.b(c,d)” or “i++”

Compact Byte Code Interpreter

- The interpreter is a combination of a register and stack machine
  - The stack is used to compute temporary values
  - The registers are used to store local variables
- Byte code decompression
  - Byte code instructions are decoded into a maximum of three atomic instructions and those instructions are executed by the interpreter

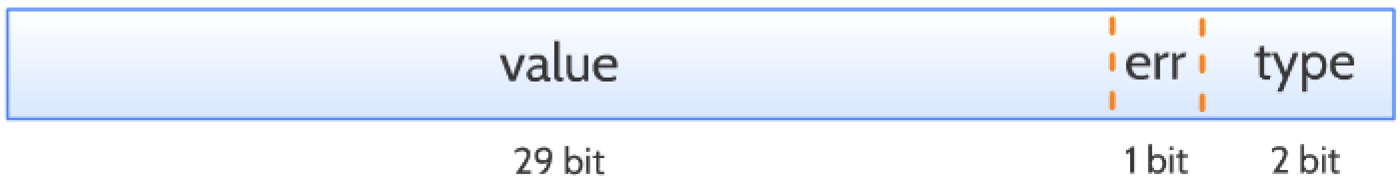
Compressed Pointers

- Compressed pointers are 16-bit values, which represent 8 byte aligned addresses on the JerryScript heap
  - Saves 50% of memory on 32-bit systems
- The JerryScript heap is a linear memory space with a maximum size of 512KB (equals to UINT16_MAX * 8)
  - UINT16_MAX is 65535
- Pointer compression can also be turned off to enable a maximum heap size of 4GB

Value Representation

Samsung Open Source Group

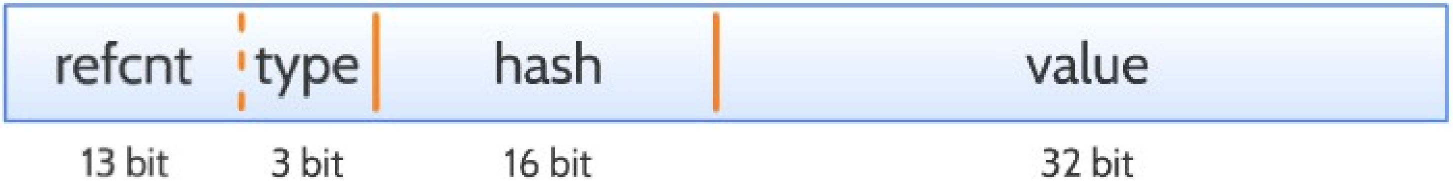
- JavaScript is a dynamically typed language
  - All values carry type information as well
- ECMAScript values in JerryScript are 32-bit wide
  - They can be primitive values (true, null, undefined, …) or pointers to numbers, strings or objects
- On 32-bit systems, 29 bits are enough to directly store any 8 byte aligned 32-bit pointer



String Representation

Samsung Open Source Group

- String descriptor is 8 bytes long
- Several string types are supported in JerryScript besides the usual character array
  - Short strings: Stored in the 32-bit value field
  - Magic (frequently used) string indices



Number Representation

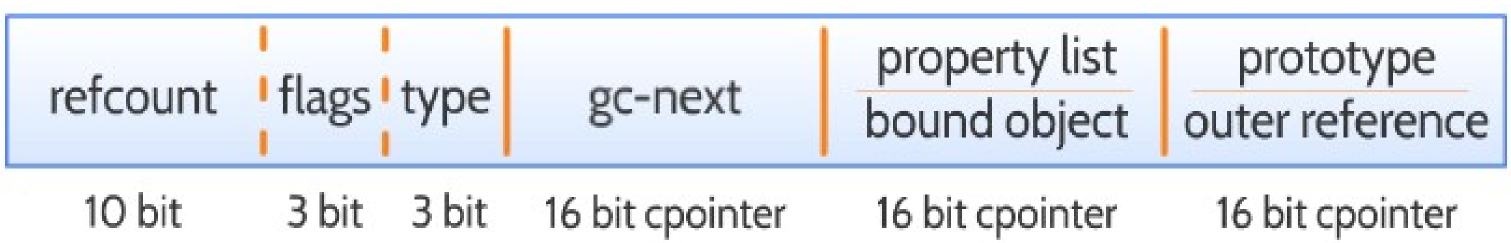
- Numbers are double precision values by default
- Optional mode for single precision values
  - Single precision numbers do not satisfy the ECMAScript requirements but can be computed faster, trading precision for performance

IEEE 754 number

32/64 bit

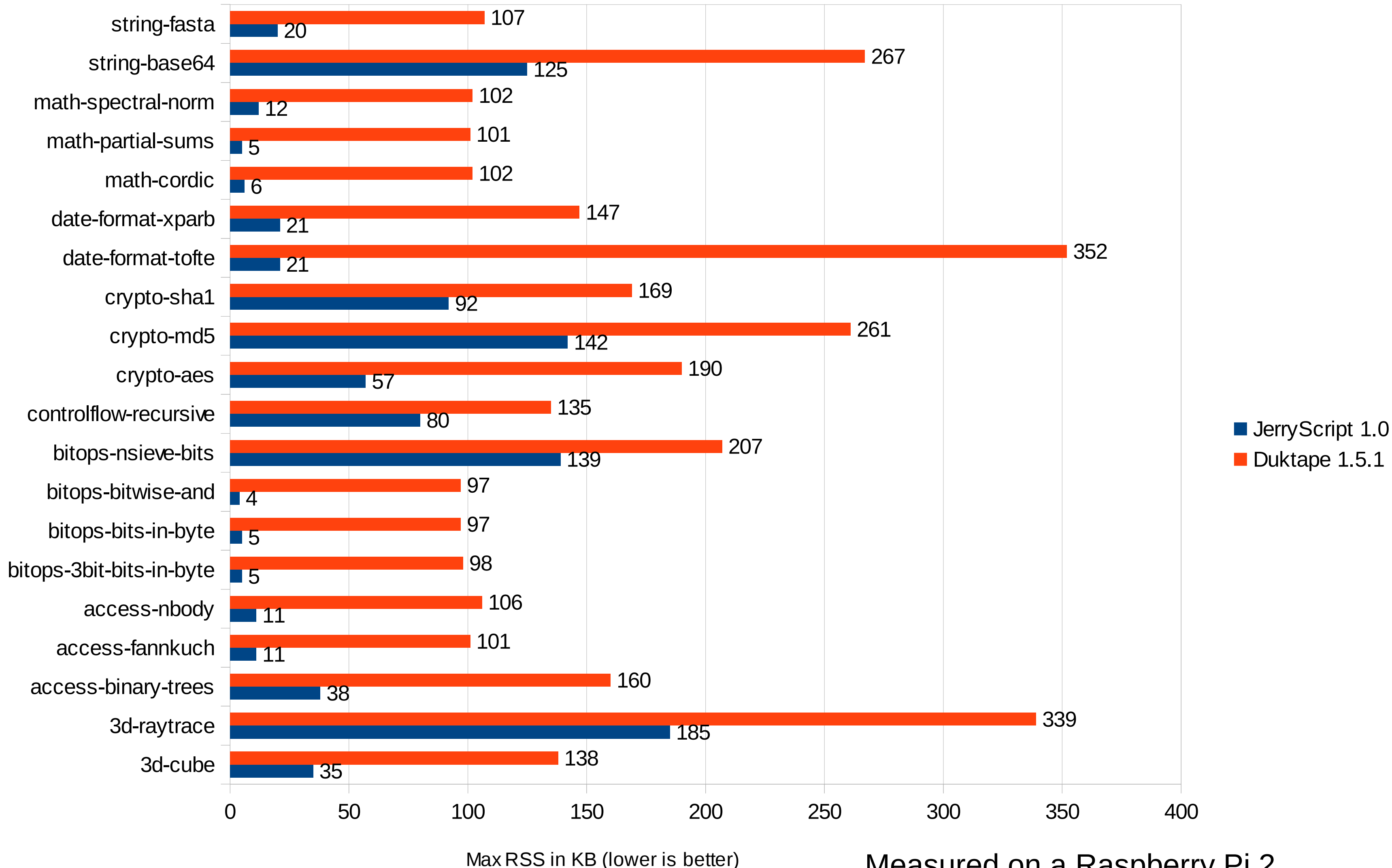
Object Representation

- Garbage collector can visit all existing objects
- Objects have a property list
  - Named data, named accessor properties
  - Internal properties
- Functions are objects in JavaScript



Memory consumption/ Performance

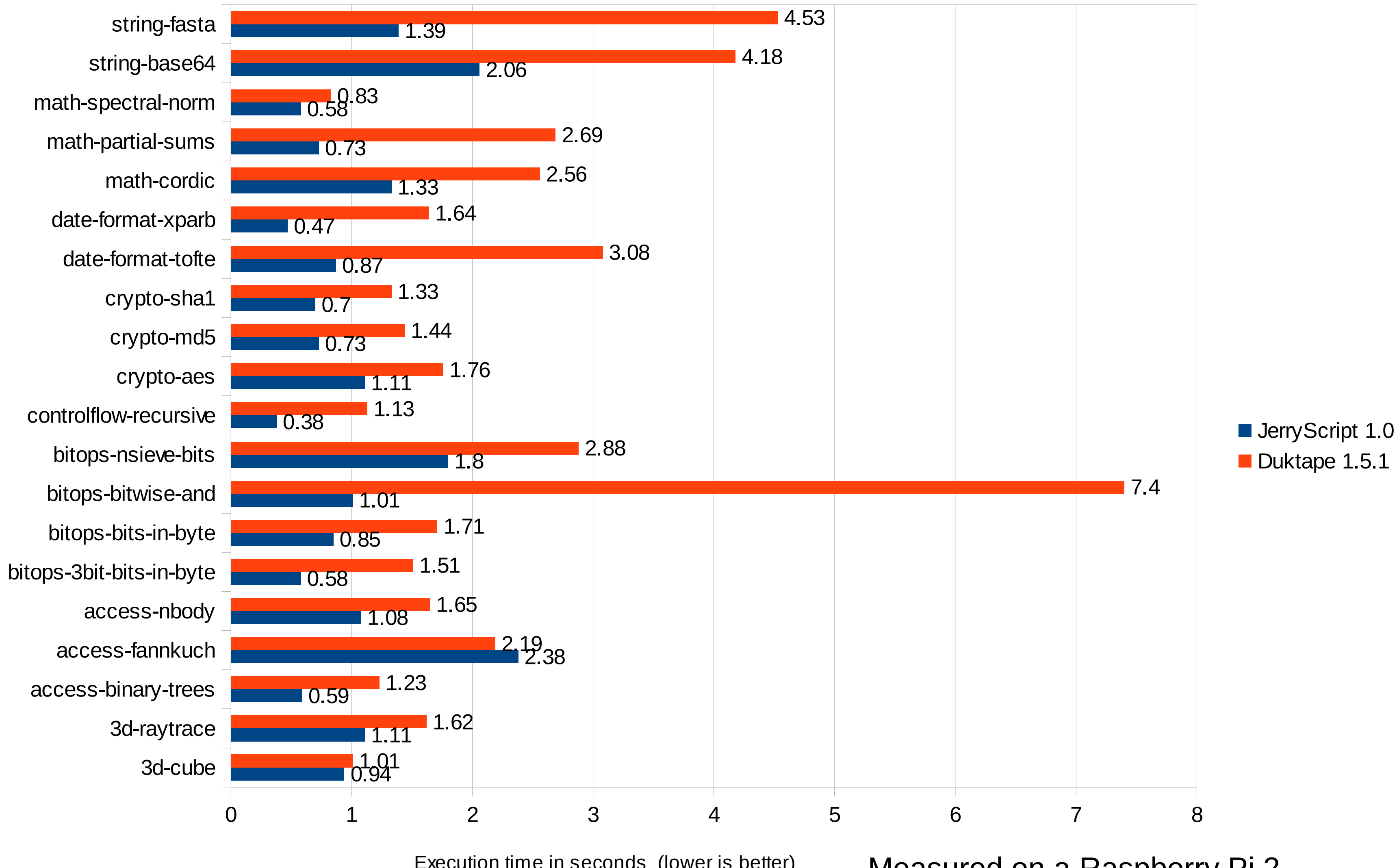
SunSpider 1.0.2 - Memory consumption



Measured on a Raspberry Pi 2

SunSpider 1.0.2 - Performance

SAMSUNG
Open Source Group



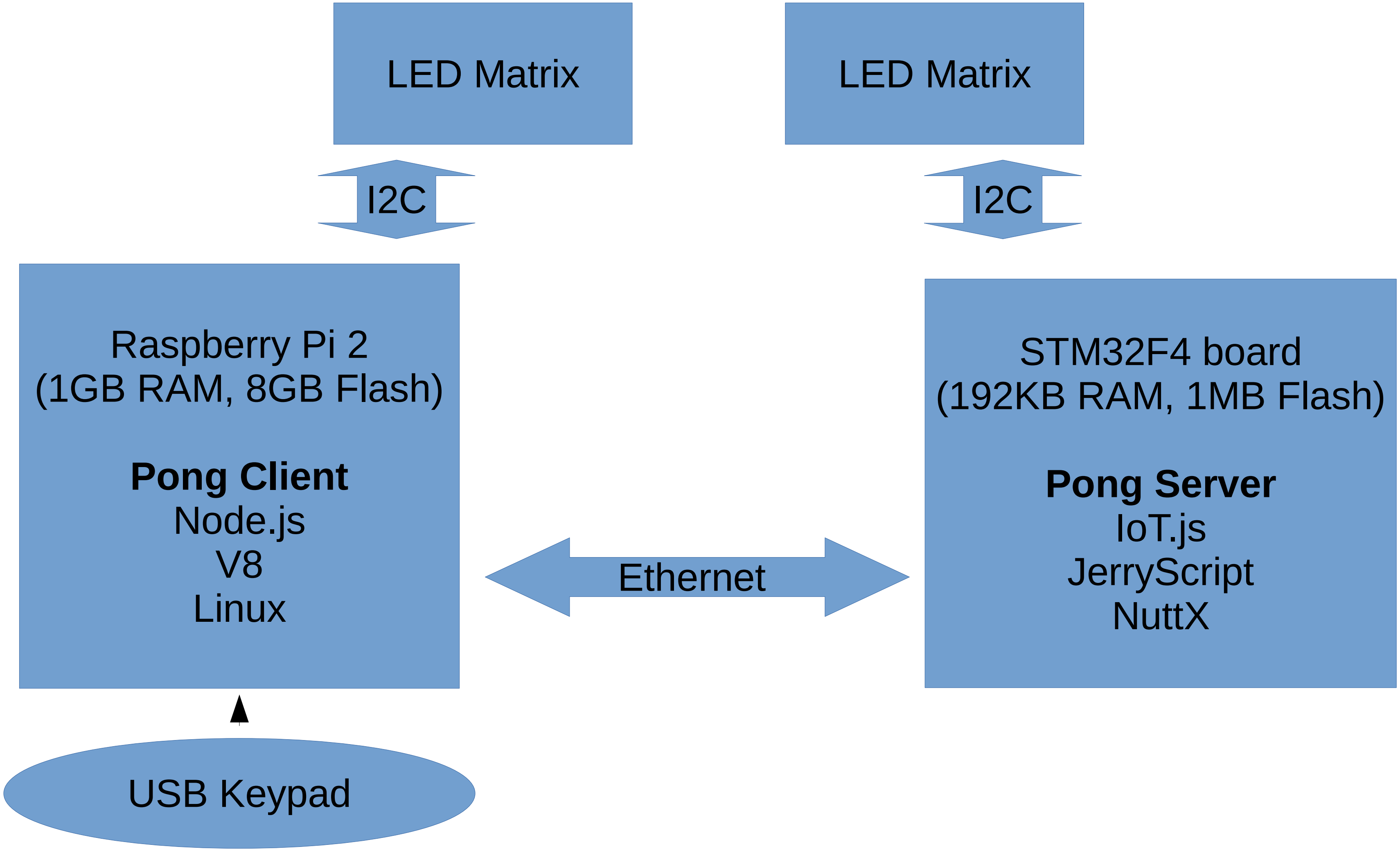
Measured on a Raspberry Pi 2

Demo

Pong Demo

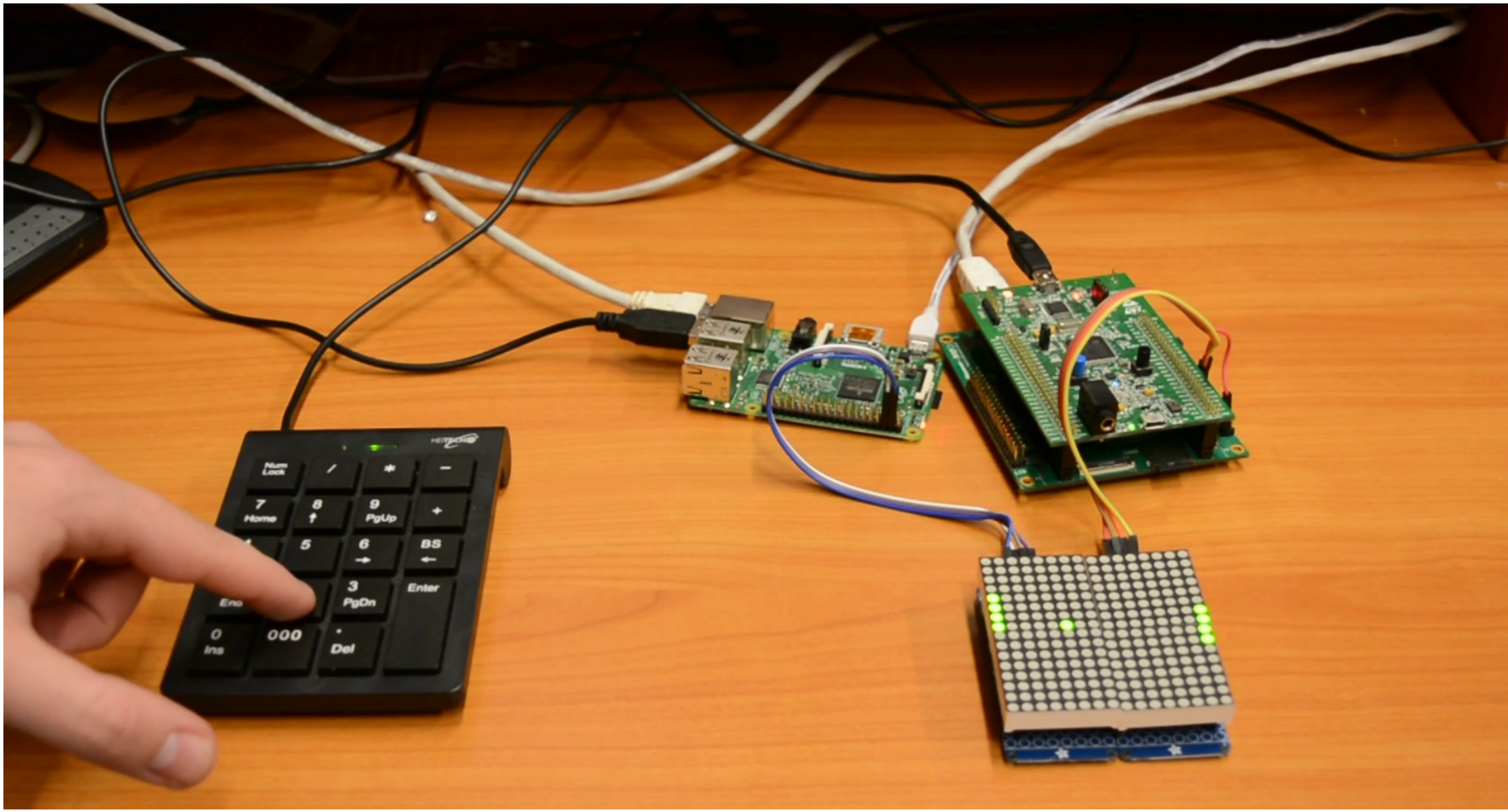
- Implementation of the classic Pong game
- Display shared across two devices
- Each device drives one LED matrix
- Implemented as a Node.js module
- "AI" oppenent running on the microcontroller

Pong Demo


LED Matrix
LED Matrix
I2C
I2C
Raspberry Pi 2
(1GB RAM, 8GB Flash)
Pong Client
Node.js
V8
Linux
Ethernet
STM32F4 board
(192KB RAM, 1MB Flash)
Pong Server
IoT.js
JerryScript
NuttX
USB Keypad

SAMSUNG
Open Source Group

Pong Demo



JerryScript 6LoWPAN Demo

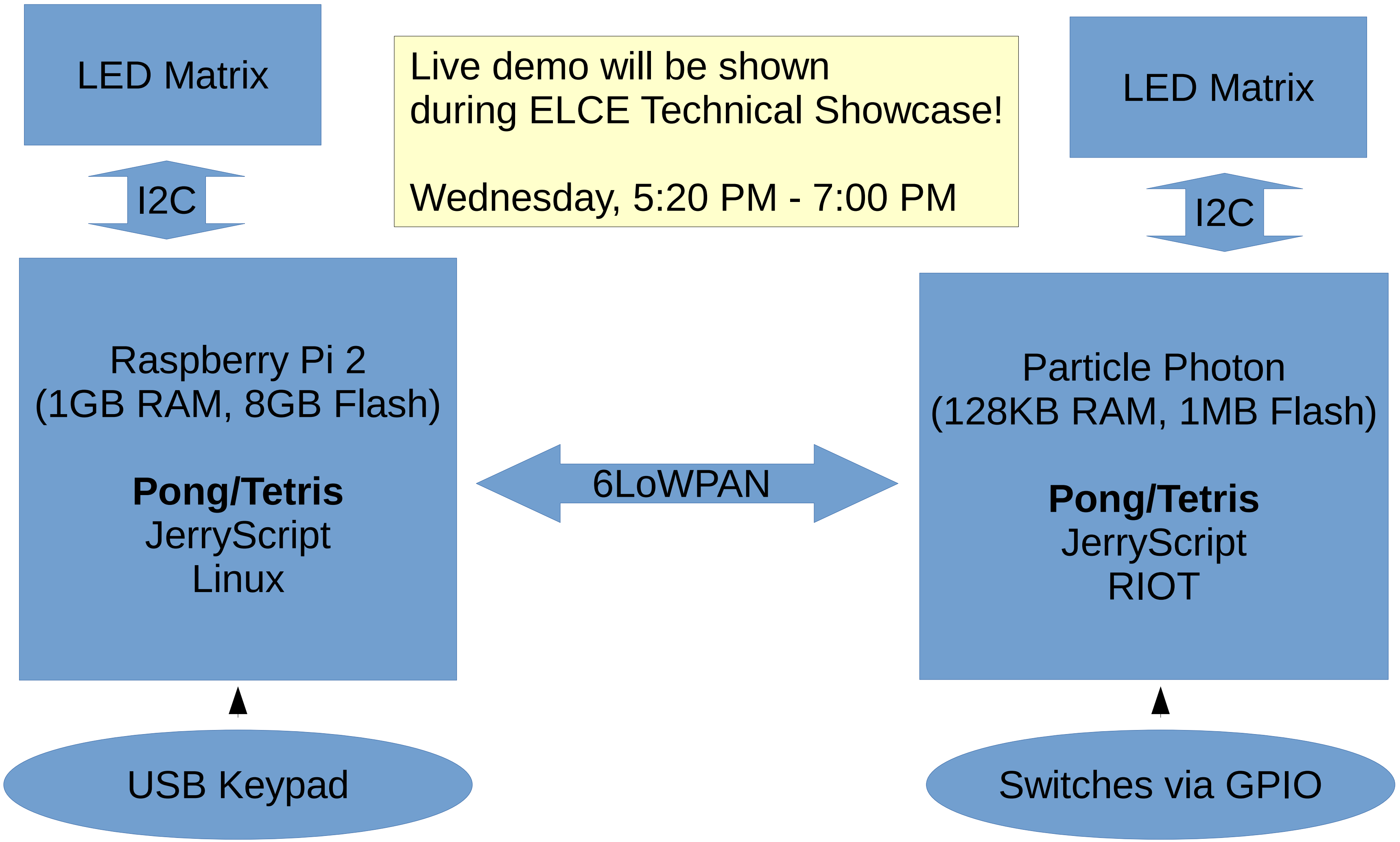
- Multiplayer implementation of the classic Pong/Tetris game
- Each device drives one LED matrix as display
- Game implemented in JavaScript
- Running on Photon boards
- Low-power wireless communication via 6LoWPAN

JerryScript 6LoWPAN Demo

Samsung Open Source Group

Live demo will be shown
during ELCE Technical Showcase!

Wednesday, 5:20 PM - 7:00 PM



Future work

Future work

- Further performance and memory optimizations
- Debugging support
- Memory profiling
- Selected ES6 features
- Support for more boards

Summary

Summary

- Significantly lowers barrier of entry for JavaScript development targeting heavily constrained embedded devices
- Speeds up development
- Active community
- More information on http://jerryscript.net
- Looking for bug reports and feedback

Thank you.

SAMSUNG
Open Source Group

Contact Information:

Tilmann Scheller
t.scheller@samsung.com

Samsung Open Source Group
Samsung Research UK