DOTNEXT

Raspberry PI and .NET Core on Linux: the fast track to IoT

Microsoft Most Valuable Professional

Raffaele Rialdi - Senior Software Architect

@raffaeler
raffaeler@vevy.com

Who am I?

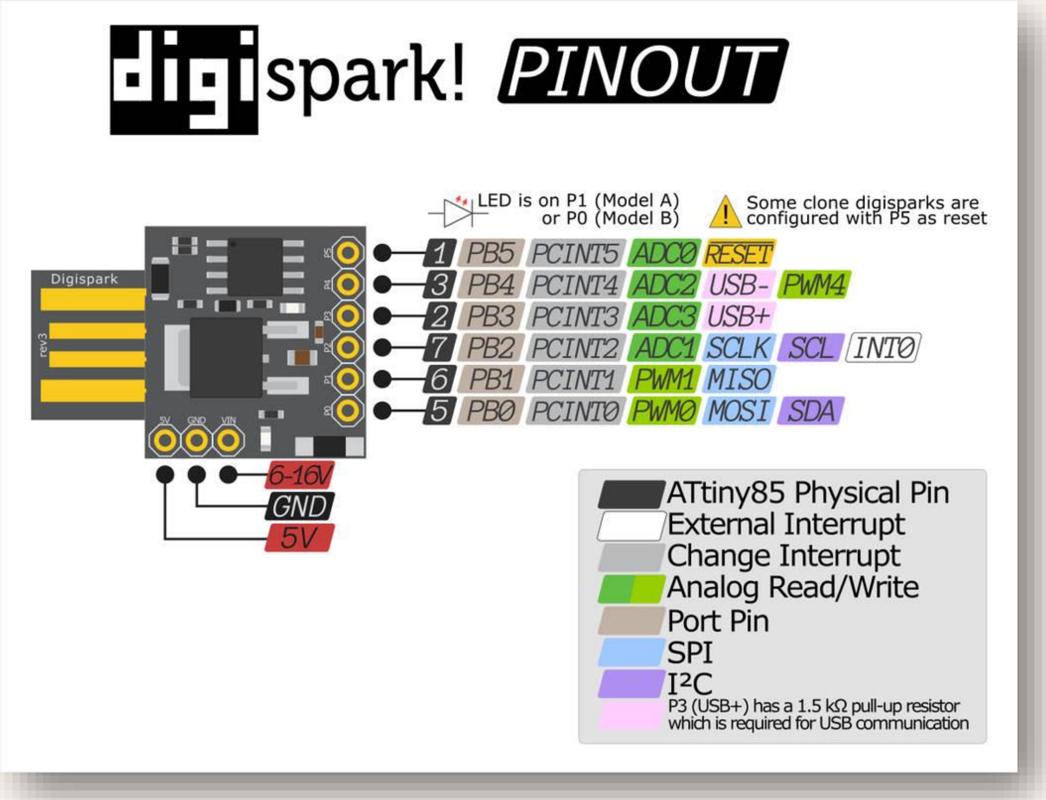
- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
  - @raffaeler also known as "Raf"
- Consultant in many industries
  - Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe (development and security)
  - Italy, Romania, Bulgaria, Russia (Moscow, St Petersburg and Novosibirsk), USA, …
- And proud member of the great Microsoft MVP family since 2003

La sessione non è:
..... un tutorial su come hackerare sistemi
..... una dimostrazione sulle vulnerabilità di Windows
..... adatta ai deboli di cuore

Agenda

- IoT: computers or microcontrollers, that is the question!
- What can we do with the Raspberry PI and .NET Core
- Driving physical sensors/devices from the Raspberry PI
- The new goodies inside .NET Core 3.0 and C# 7.x (very useful on the RPi)
- Publishing the App
- Interoperability with C/C++ code
- Code, code, code!

The "Tiny85" microcontroller (Arduino)

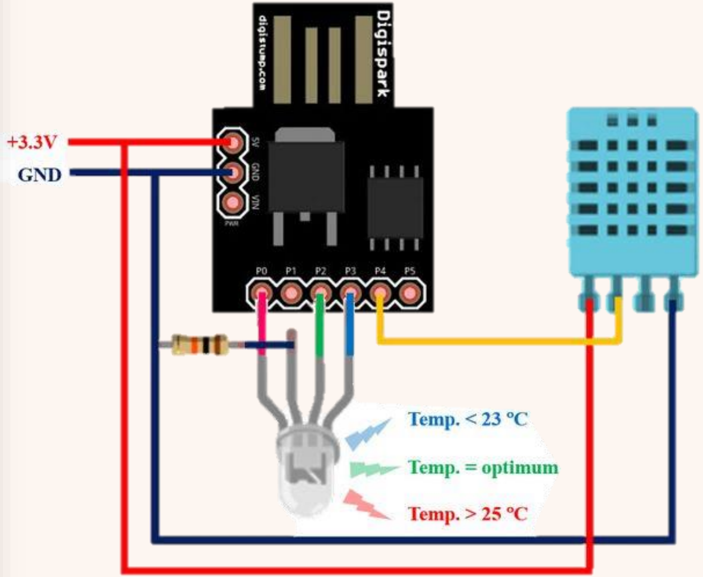


LED is on P1 (Model A) or P0 (Model B)

Some clone digisparks are configured with P5 as reset

1	PB5	PCINT5	ADC0	RESET		
3	PB4	PCINT4	ADC2	USB-	PWM4	
2	PB3	PCINT3	ADC3	USB+		
7	PB2	PCINT2	ADC1	SCLK	SCL	INT0
6	PB1	PCINT1	PWM1	MISO		
5	PB0	PCINT0	PWM0	MOSI	SDA	

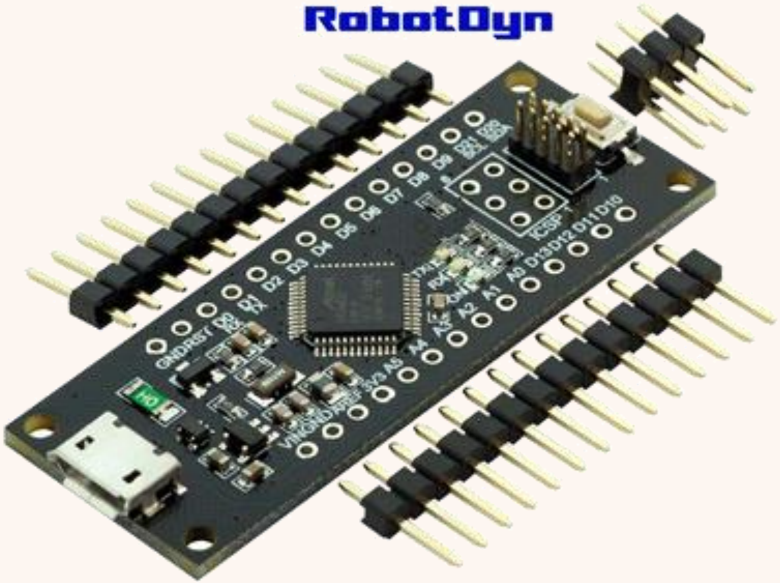
- ATtiny85 Physical Pin
- External Interrupt
- Change Interrupt
- Analog Read/Write
- Port Pin
- SPI
- I²C
- P3 (USB+) has a 1.5 kΩ pull-up resistor which is required for USB communication



ATMEL SAMD21: the big, "fat" microcontroller (Arduino)

- 48MHz, 256K Flash, 32K RAM
- 12 Channel DMA
- 8 hardware timers + comparators
- RTC, watchdog
- USB2.0 (8 endpoints)
- 6 serial ports (USART, SPI, I2C)
- I2S Sound port
- 10 bit DAC, comparators, 20 channel ADC
- Touch controller
- 52 I/O pins

- Still, no operating system and real-time



Microcontrollers vs Full computers

Microcontrollers

- Single-chip, no operating system
- Very cheap
- Rich of on-board peripherals
- Real-time processing
- Data acquisition on reboot is a good strategy to avoid bugs
- Secure protocols are hard to implement (low resources)

Computers / Embedded boards

- Full Operating Systems
- Popular OSes are not real-time
- Data is acquired on polling or hardware interrupt requests
- Rebooting is slow
- Require frequent security updates
- Secure protocol stacks are tested and maintained (TLS, crypto, …)

.NET Core on the Raspberry PI, on Linux (Raspbian)

- You can use all the .NET Core power, no exceptions

- Three .NET Core options
  - Install the .NET Core SDK
  - Install the .NET Core Runtime
  - Do not install anything and use xcopy deployment

- With .NET Core 3.0 you can start using C# 8

- You can remote debug the application or going deep with LLDB + SOS.DLL

Getting started with the Raspberry PI

- Device information: https://www.raspberrypi.org/
  - Required Hardware: RPi 2 to RPi 3B+
    - RPi Zero cannot run .NET Core (yet) because of the ARMv6 CPU

- NetCore for Linux-ARM
  - Available as SDK and Runtime

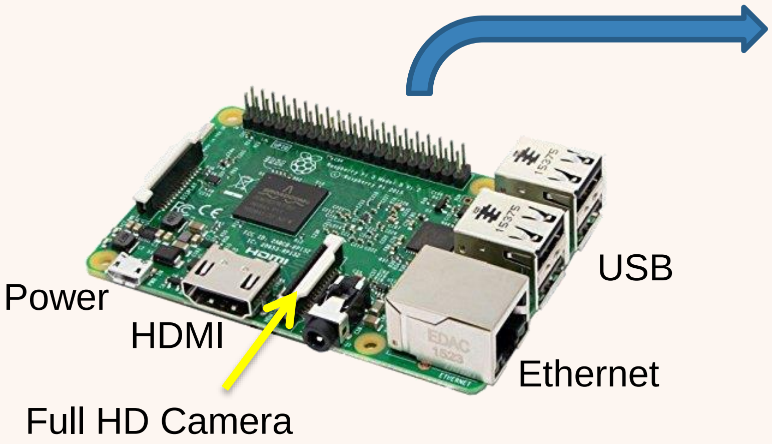
- One repository with all the info you need:
  - https://github.com/raffaeler/raspberrypi
  - Tutorials, materials, resources, GPIO pin maps and more

Useful tools

- SSH client
  - You need to enable SSHD on the device via raspi-config utility
  - Get a SSH client for Windows (Putty, Bitvise, …) to use the terminal
  - Get an SCP client (WinSCP, Bitvise, …) to copy files from/to the device

- DeployTool by Raf (me)
  - A tool to ease deployment to a Linux machine (Continuous Deployment)
  - https://github.com/raffaeler/DeployTool

Raspberry Pi Peripherals



- ARM Cortex A53 - 4 Cores – 1.4GHz - 1GB RAM
- GPU Broadcom VideoCore IV
- Ethernet 1GBit – Wifi 2.4/5GHz – BT4.2 / BLE
- GPIO 40 pins – I2C – 3xSPI – UART – 2 x PWM

BCM	WPi			Pins			WPi	BCM
-	-	3.3V	01	● ●	02	5V	-	-
2	8	SDA.1	03	● ●	04	5V	-	-
3	9	SCL.1	05	● ●	06	0V	-	-
4	7	GPIO.7	07	● ●	08	TxD	15	14
-	-	0V	09	● ●	10	RxD	16	15
17	0	GPIO.0	11	● ●	12	GPIO.1	1	18
27	2	GPIO.2	13	● ●	14	0V	-	-
22	3	GPIO.3	15	● ●	16	GPIO.4	4	23
-	-	3.3V	17	● ●	18	GPIO.5	5	24
10	12	MOSI	19	● ●	20	0V	-	-
9	13	MISO	21	● ●	22	GPIO.6	6	25
11	14	SCLK	23	● ●	24	CE0	10	8
-	-	0V	25	● ●	26	CE1	11	7
0	30	SDA.0	27	● ●	28	SCL.0	31	1
5	21	GPIO.21	29	● ●	30	0V	-	-
6	22	GPIO.22	31	● ●	32	GPIO.26	26	12
13	23	GPIO.23	33	● ●	34	0V	-	-
19	24	GPIO.24	35	● ●	36	GPIO.27	27	16
26	25	GPIO.25	37	● ●	38	GPIO.28	28	20
-	-	0V	39	● ●	40	GPIO.29	29	21

http://github.com/raffaeler

Introducing the new System.Device namespace

- A new Microsoft library to control physical devices
  - System.Device.Gpio
  - System.Device.I2c
  - System.Device.Pwm

Controlling the peripherals

  - IoT.Device.Bindings

High-level device management

https://github.com/dotnet/iot/tree/master/src/devices

- Published on GitHub: http://github.com/dotnet/iot (still experimental)

Creating a Console App

- Use the default template for a NetCore 2.1 Console app
- Three local peripherals "netstandard" libraries currently available:
  - IoT library: http://github.com/dotnet/iot (currently a pre-release version)
  - Unosquare.Raspberry.IO by Unosquare Labs
  - Pi.IO by Peter Marcu
- Any ARM specific resource requires RuntimeIdentifier in the csproj

```
<PropertyGroup>
    <OutputType>Exe</OutputType>
    <RuntimeIdentifier>linux-arm</RuntimeIdentifier>
    <TargetFramework>netcoreapp2.0</TargetFramework>
</PropertyGroup>
```

Deploying the App

- Creating the publishing binaries:

  `dotnet publish -c Release -r linux-arm --self-contained=false`

  - `--self-contained=true` includes everything needed to run (no runtime needed)
    - `-p:PublishReadyToRun=true` compiles into native code (ARM assembler)
  - `-p:PublishSingleFile=true` compiles into a "fat" single file containing all

- ReadyToRun requires the same operating system (Linux)
  - Can be run from WSL (Windows Subsystem for Linux) or directly on the RPi

- On the Raspberry PI set the execution attribute: `chmod +x myapp`
  - Run it: `./myapp`

Demo

Deploying a basic application

Continuous Deployment (CD)

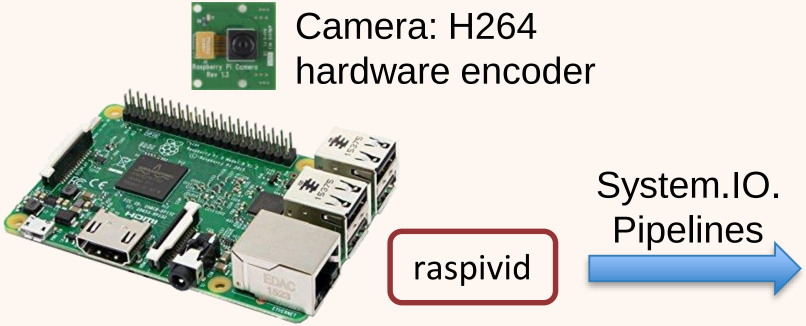
SSHDeploy is a tool created by me (Raf)

- https://github.com/raffaeler/DeployTool

1. Write the configuration file
2. Run "dotnet-deploy interact"
3. Use the menu to run the config

The new version is currently on a different branch

Going deeper: step 0

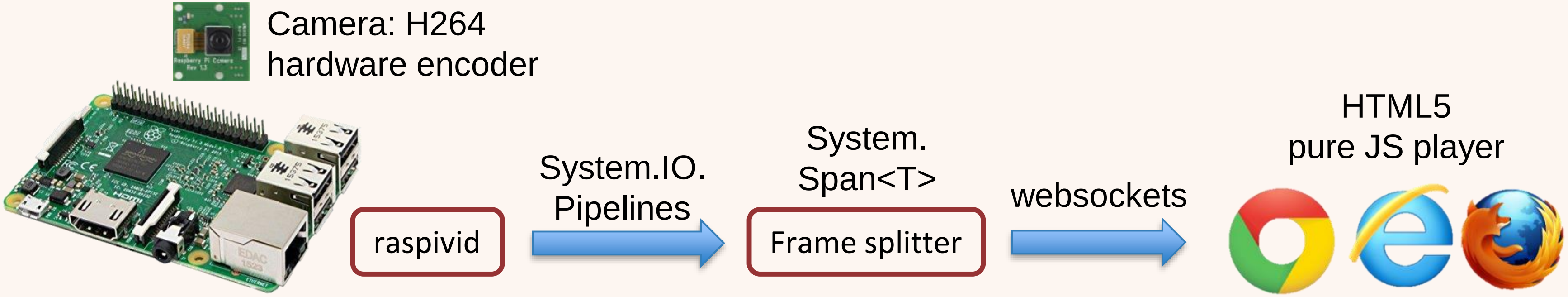


Camera: H264 hardware encoder

raspivid

System.IO.Pipelines

Going deeper: step 0



Camera: H264
hardware encoder

raspivid

System.IO.
Pipelines

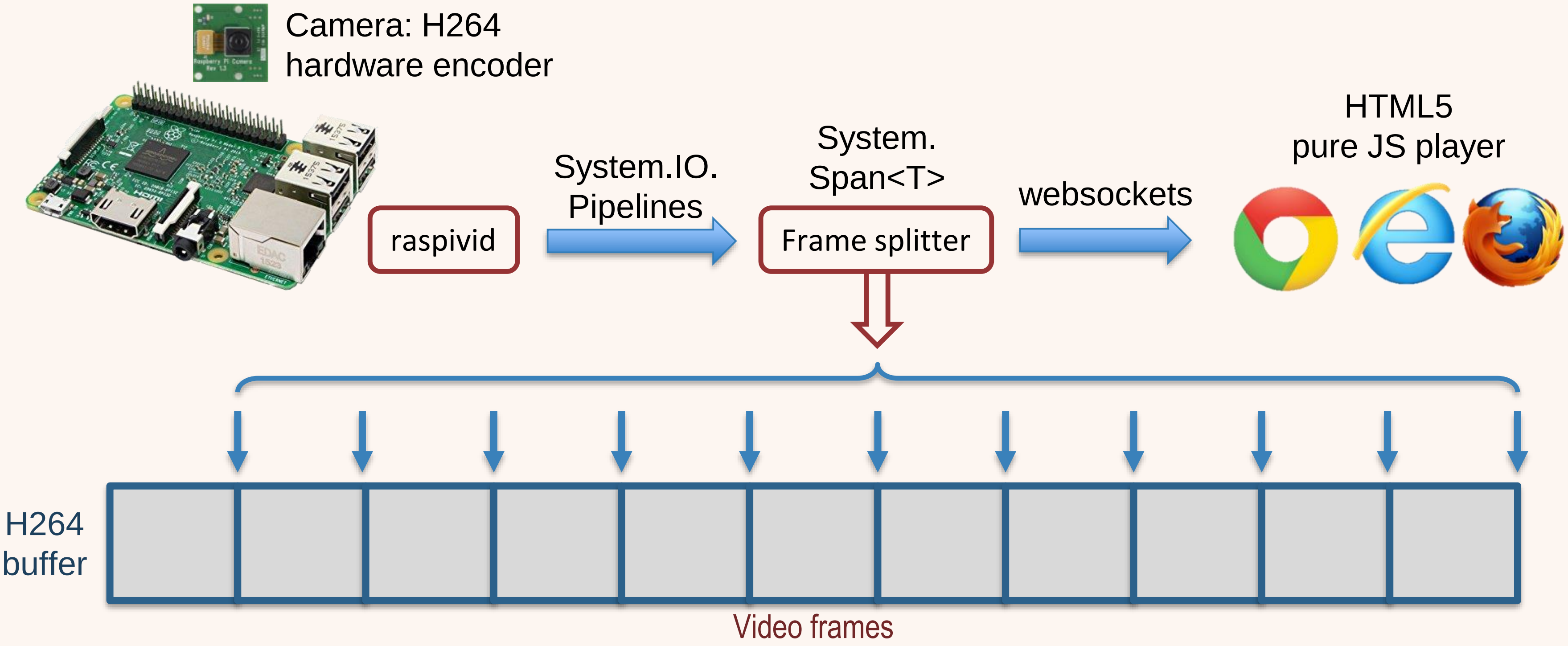
System.
Span<T>

Frame splitter

websockets

HTML5
pure JS player

Going deeper: step 0



Camera: H264 hardware encoder

raspivid

System.IO. Pipelines

System. Span<T>

Frame splitter

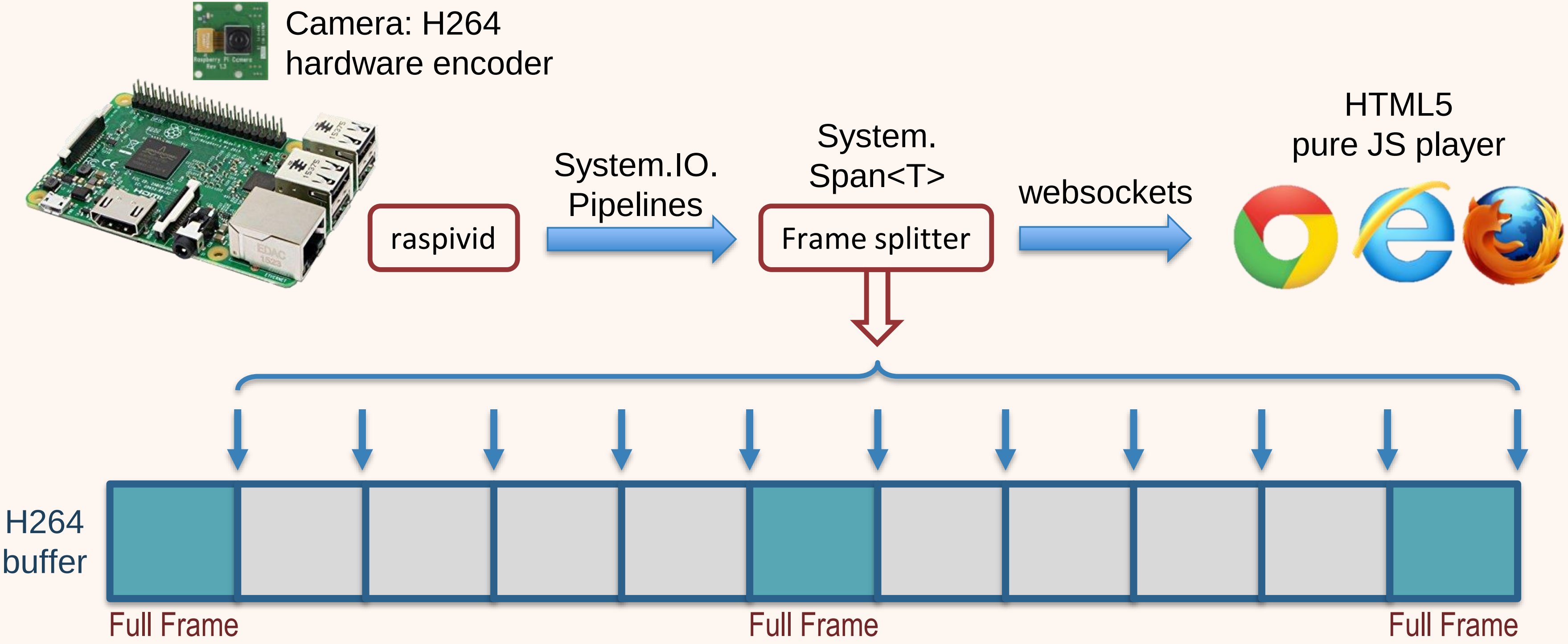
websockets

HTML5 pure JS player

H264 buffer

Video frames

Going deeper: step 0



Camera: H264
hardware encoder

raspivid

System.IO.
Pipelines

System.
Span<T>

Frame splitter

websockets

HTML5
pure JS player

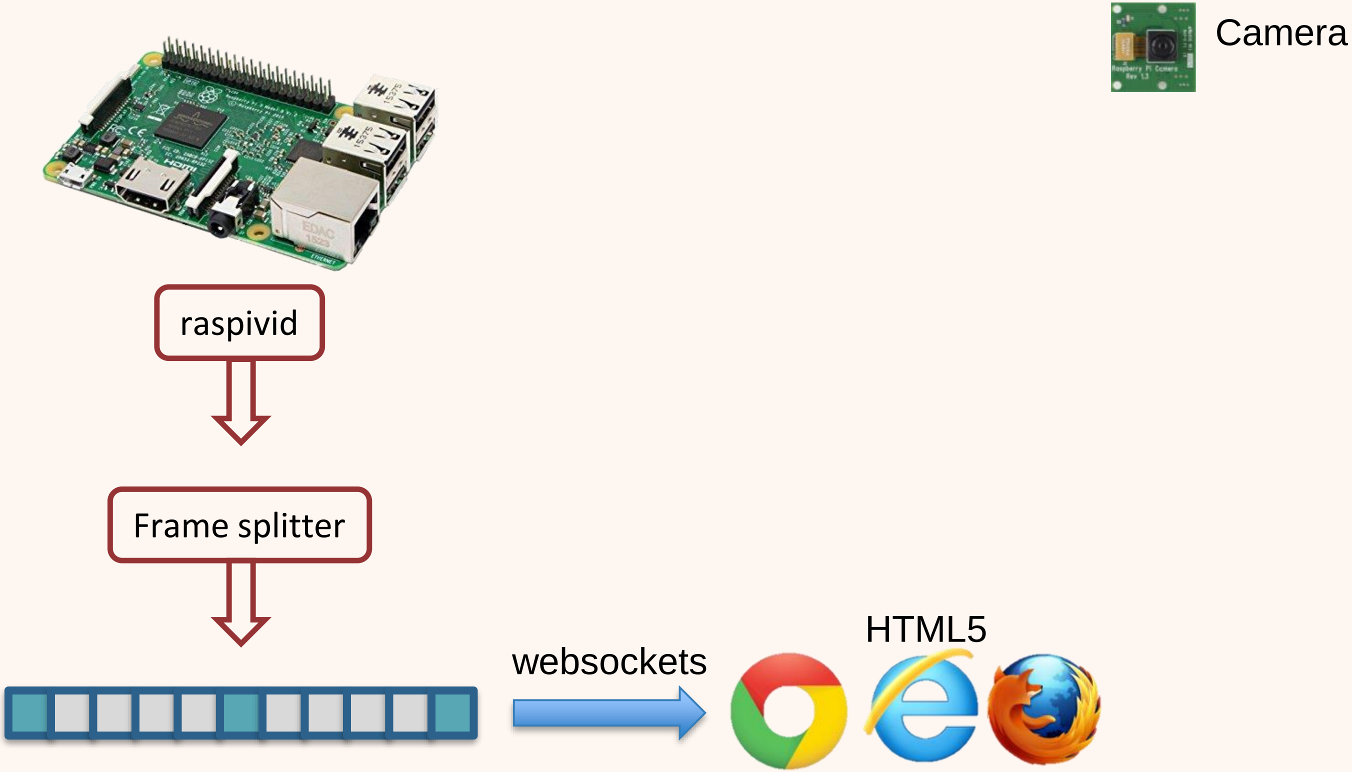
H264
buffer

Full Frame

Full Frame

Full Frame

Going deeper: step 1



Camera

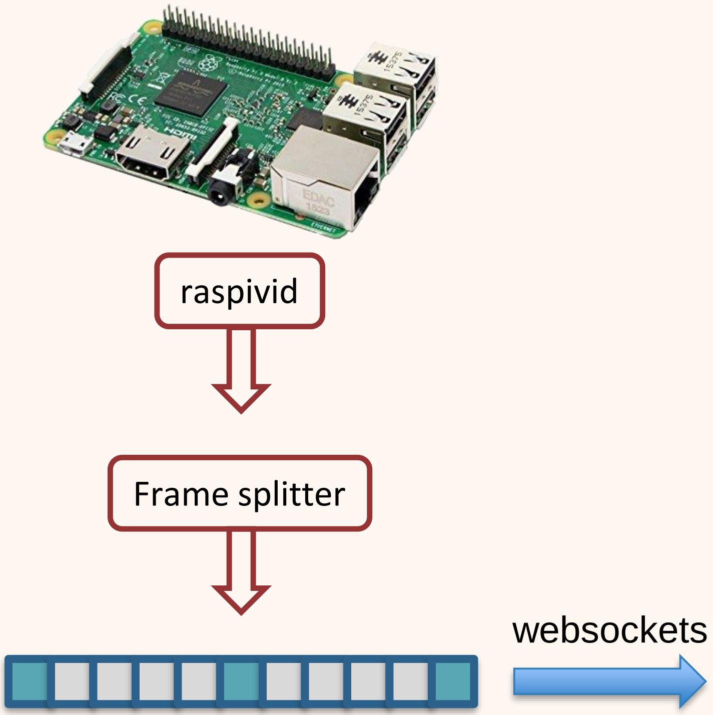
raspivid

Frame splitter

websockets

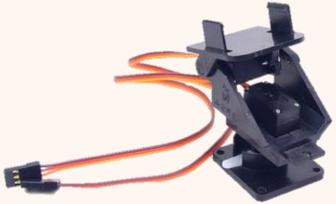
HTML5

Going deeper: step 1



HTML5

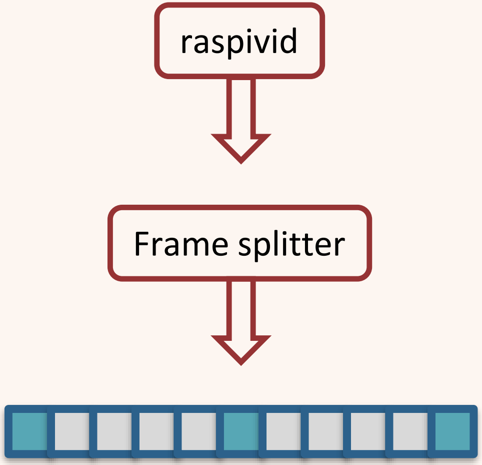
Camera



2-axis servo
PWM

PWM = Pulse Width Modulation
used to driver SG90 Servo Motors

Going deeper: step 1

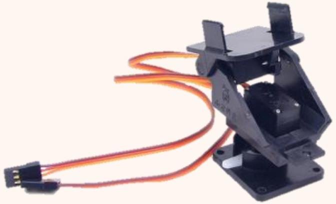


raspivid

Frame splitter

I2C controlled
Microchip
gesture sensor

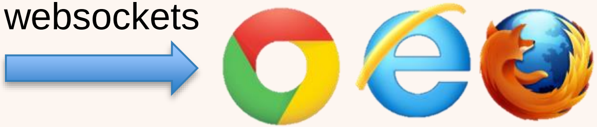
Camera



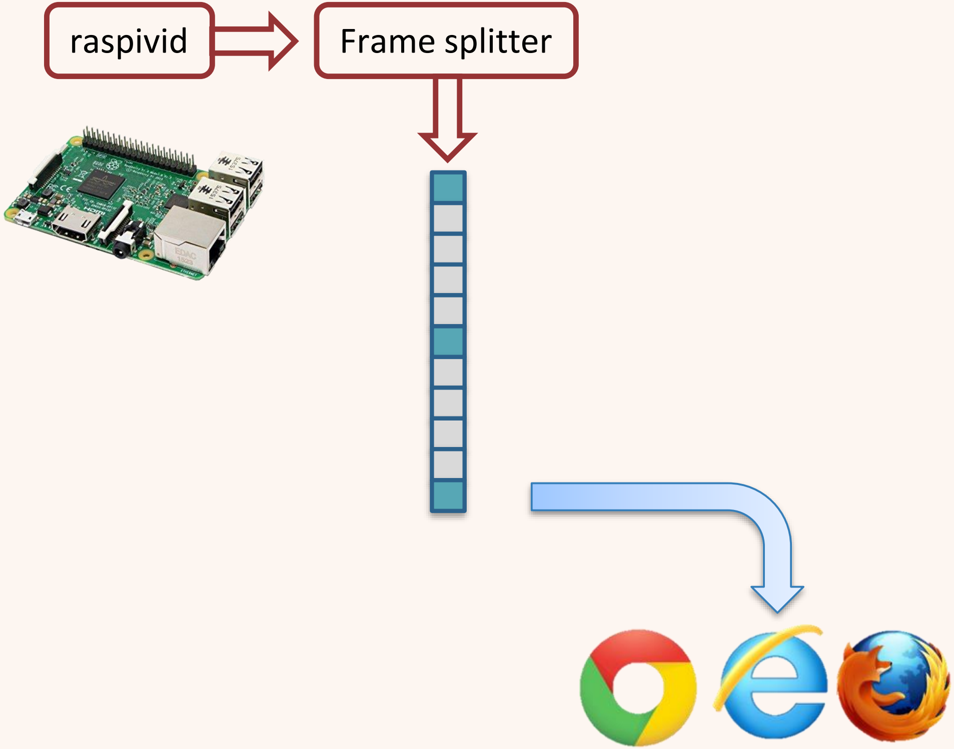
2-axis servo
PWM

websockets

HTML5



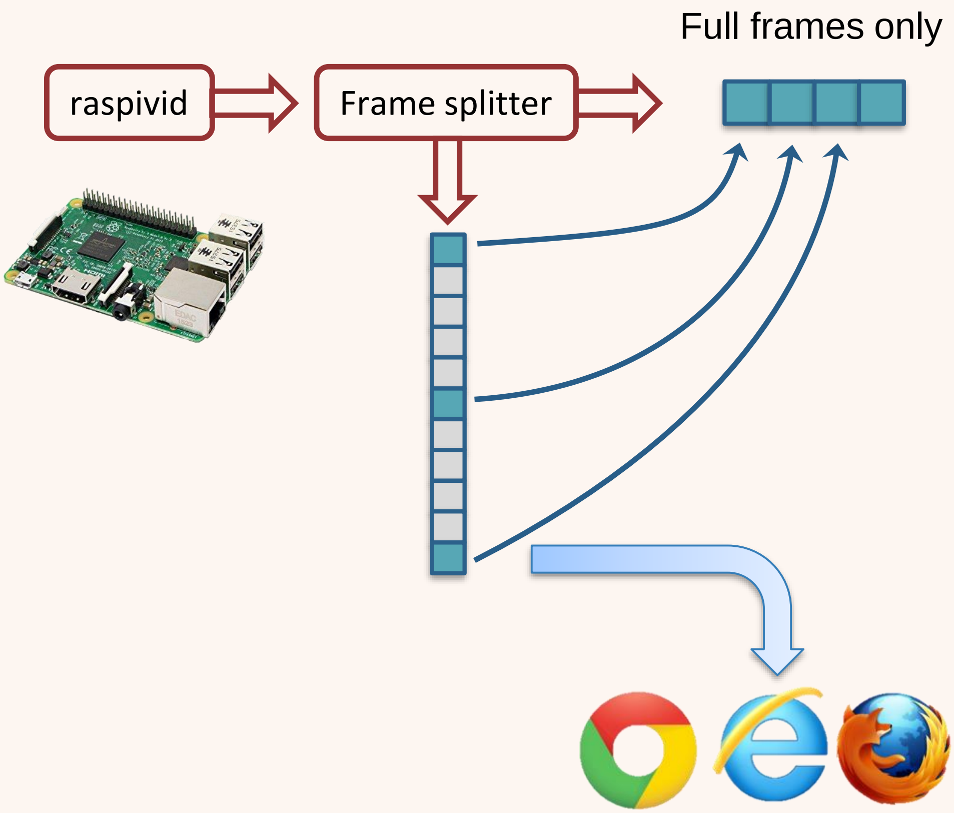
Going deeper: step 2



raspivid

Frame splitter

Going deeper: step 2

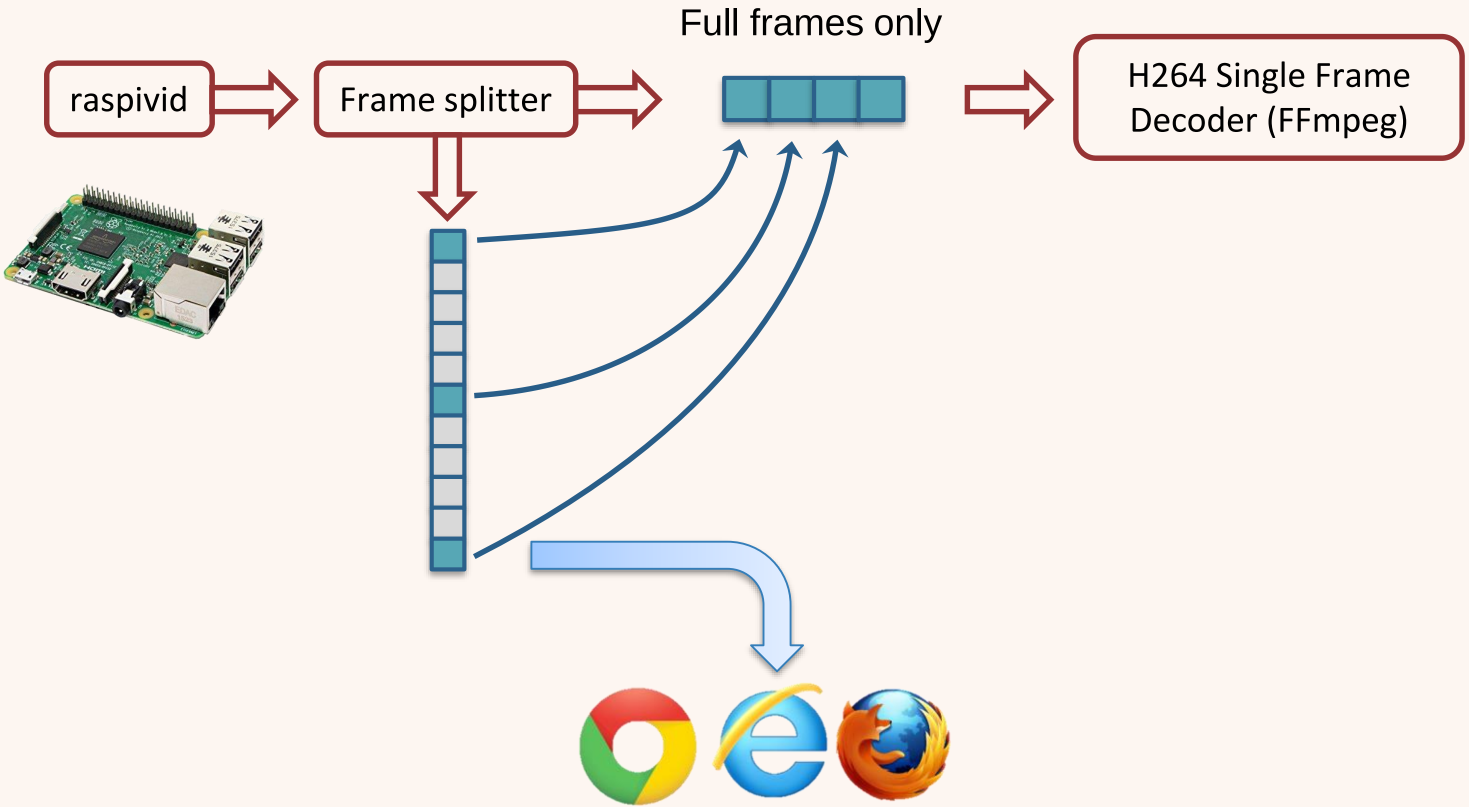


Full frames only

raspivid

Frame splitter

Going deeper: step 2



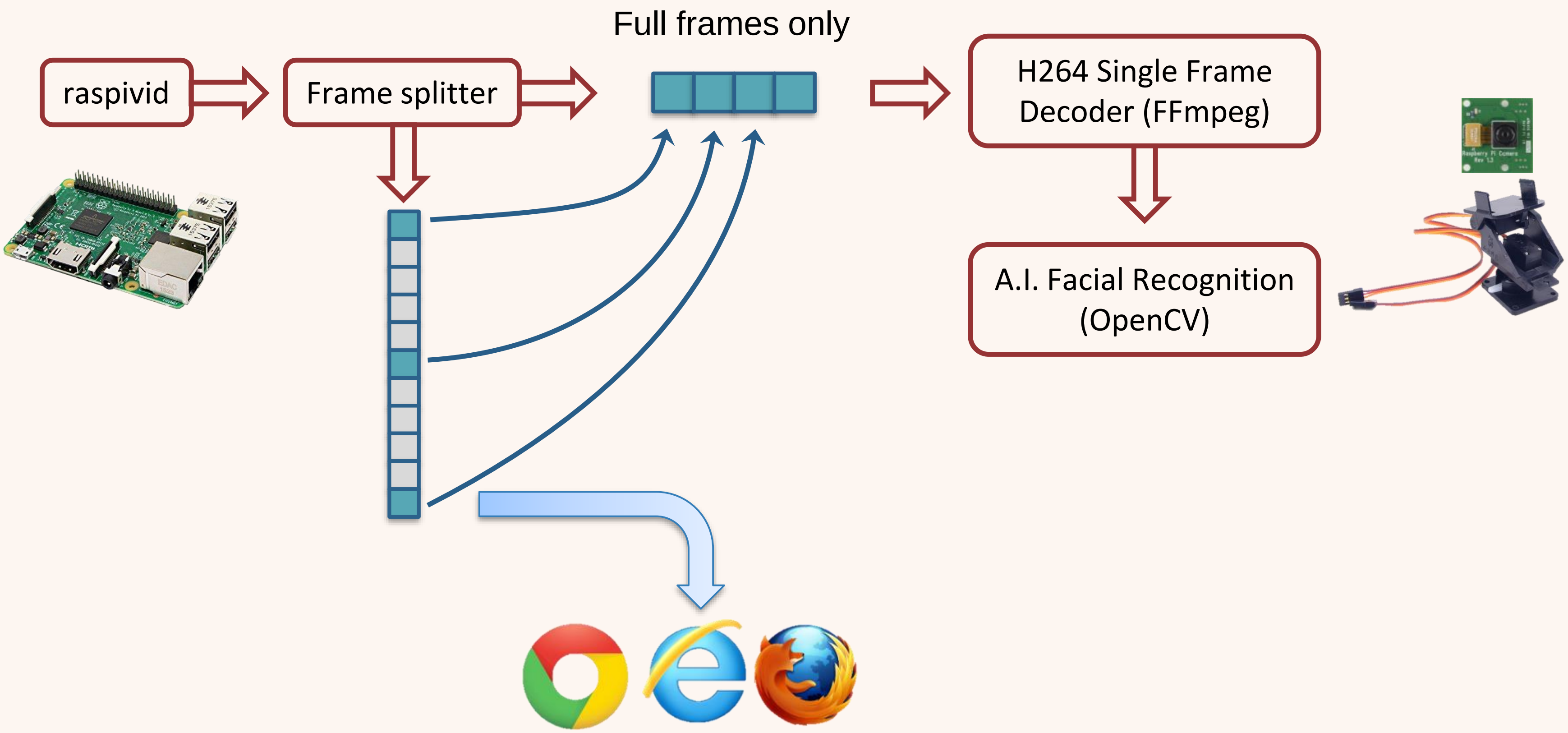
Full frames only

raspivid

Frame splitter

H264 Single Frame Decoder (FFmpeg)

Going deeper: step 2



raspivid

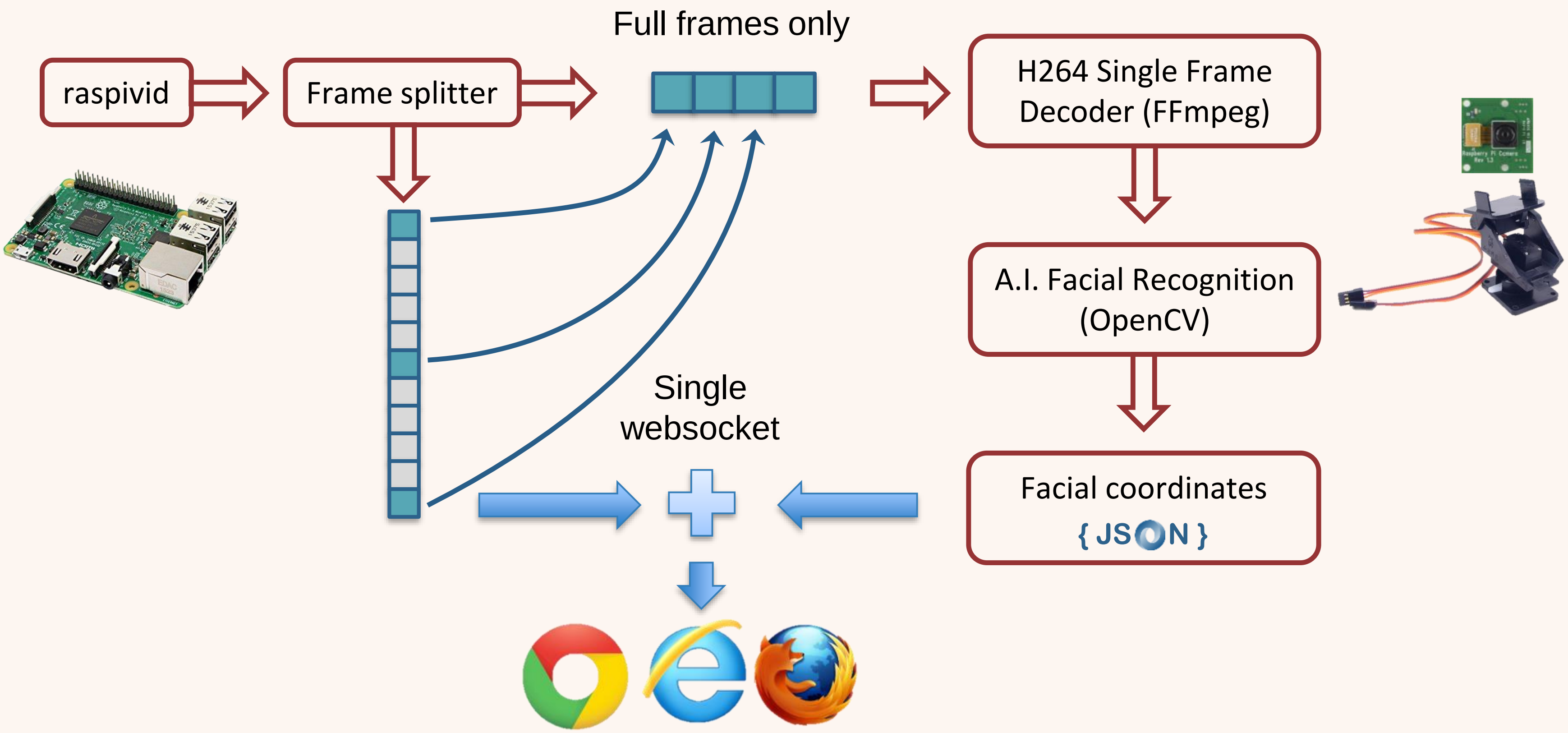
Frame splitter

Full frames only

H264 Single Frame Decoder (FFmpeg)

A.I. Facial Recognition (OpenCV)

Going deeper: step 2



raspivid

Frame splitter

Full frames only

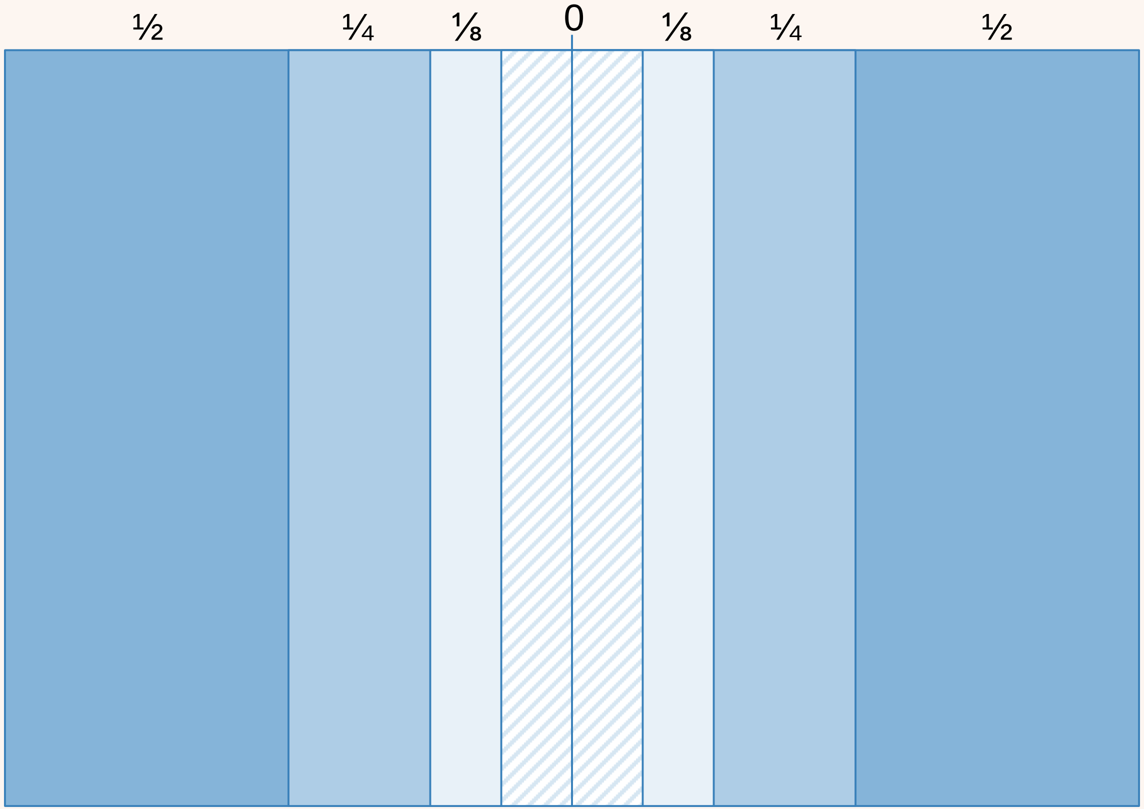
H264 Single Frame Decoder (FFmpeg)

A.I. Facial Recognition (OpenCV)

Single websocket

Facial coordinates { JSON }

Algorithm to move the servo (x axis)



Compiling OpenCV, FFmpeg, H264, Boost, …

- Raspberry PI is not powerful enough
  - GCC goes out of memory
  - Compile times may take days!
- Docker to the rescue ☺
  - Configure a powerful Debian docker container
  - Add (if required) the Raspbian repositories
  - Add all the required developer tools and packages
  - Compile the native library
  - Copy all the files on the Raspberry
  - The symbolic links must be re-created locally

Docker on the Raspberry PI

- Installation
  - curl –sSL https://get.docker.com | sh
  - sudo usermod -aG docker pi

- VS 2019 and VS Code have the same great integration of Windows

- Of course, you can also use the CLI
  - docker build –t myApp:tag –f dockerfile .
  - docker run –p 5000:5000 myApp

Demo @
Discussion Zone

Takeaways

- Check out the material on GitHub
  - https://github.com/raffaeler/raspberrypi
  - All the docs you need to put your hands on the Raspberry PI with .NET Core

- Start using .NET Core on Linux
  - Using WSL (WSL 2 is coming very soon!)
  - Using Linux Docker containers
  - Deploying on the RPi

- Continue the conversation later today, on Github or on Twitter @raffaeler

Questions?

```
C:\app1>dotnet --help
.NET Command Line Tools
Usage: dotnet [common-options] [command] [arguments]

Common Options (passed before the command):
  -v|--verbose  Enable verbose output
  --version     Display .NET CLI Version Number
  --info        Display .NET CLI Info

Common Commands:
  new
  restore Restore
  build
  publish Publishes a .NET project for deployment (including the runtime)
  run Compiles and immediately
  test Runs unit
  pack Creates a NuGet package
```

Questions @ booth outside this room

- Interfacing sensors
- Publishing
- Debugging and crash dump analysis
- Running the app as a service
- Interoperability with native code
- …

Thank you!