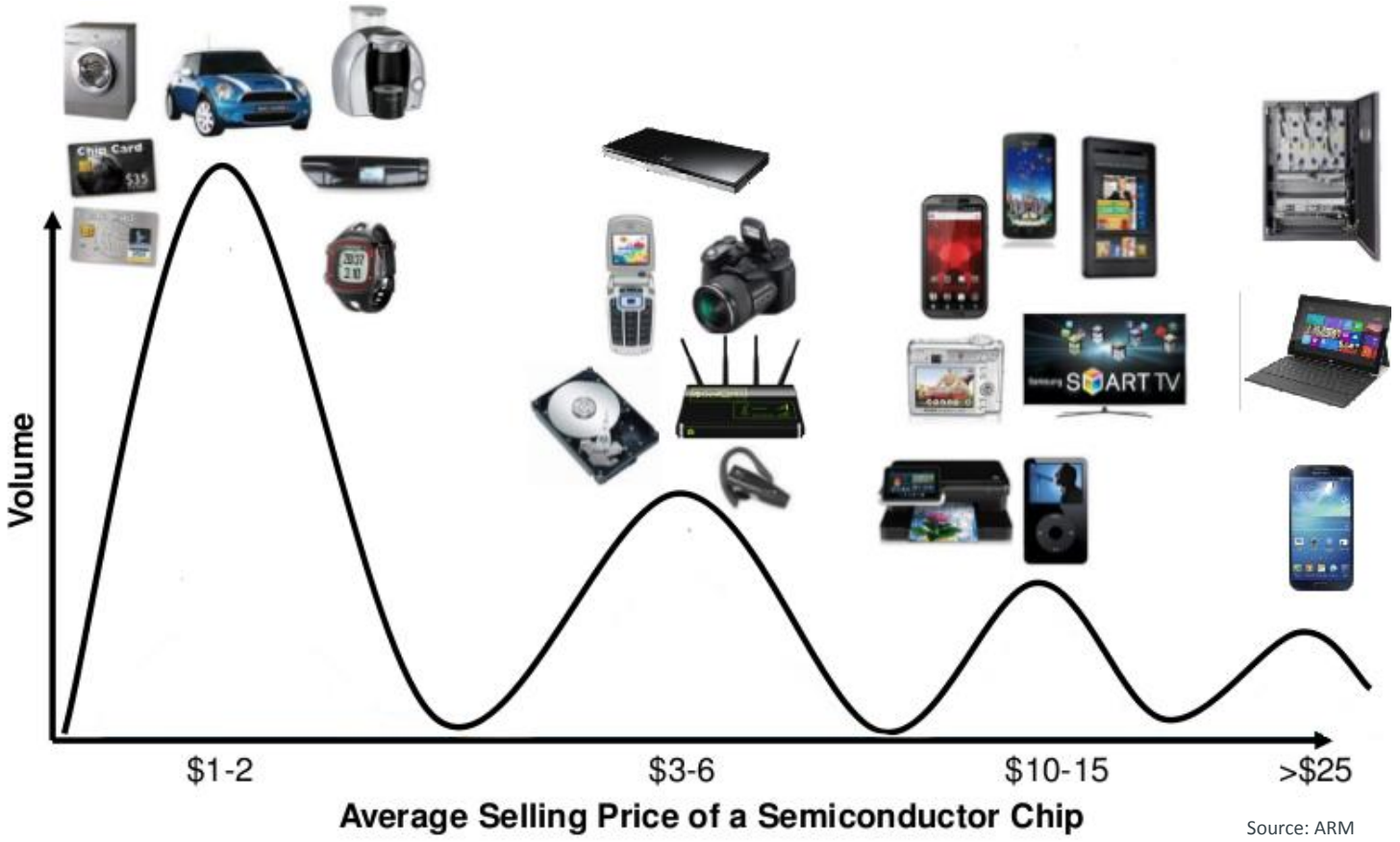
Java application platforms for design-to-cost embedded systems

Régis Latawiec, coo

IS2T

www.is2t.com

Embedded Processing Market Share

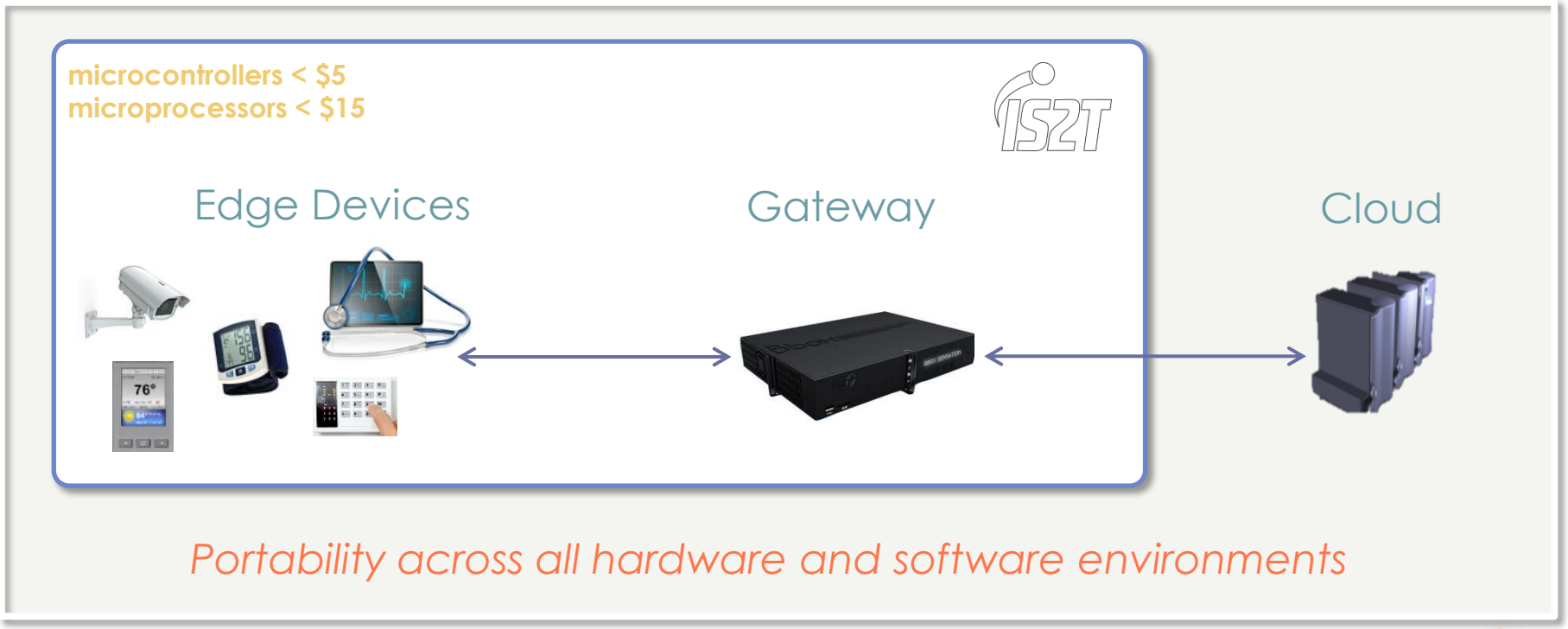


Source: ARM

IS2T - Solutions for Embedded Innovations

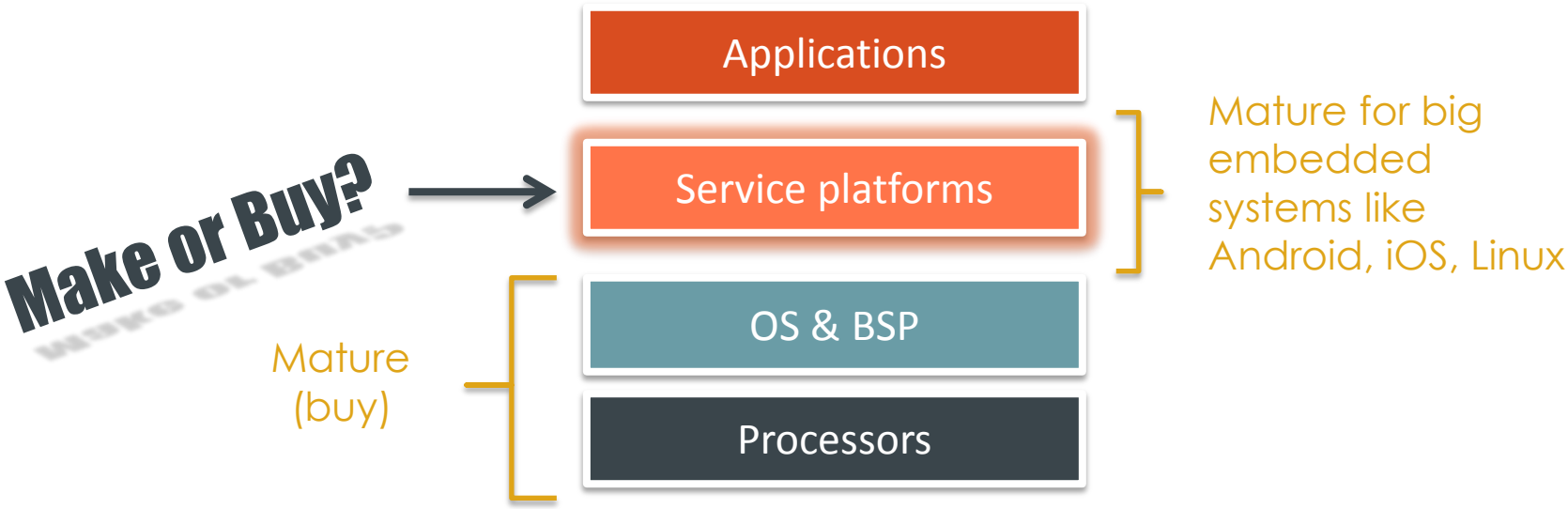
Develop software applications and leverage innovations

at low Total Cost of Ownership.

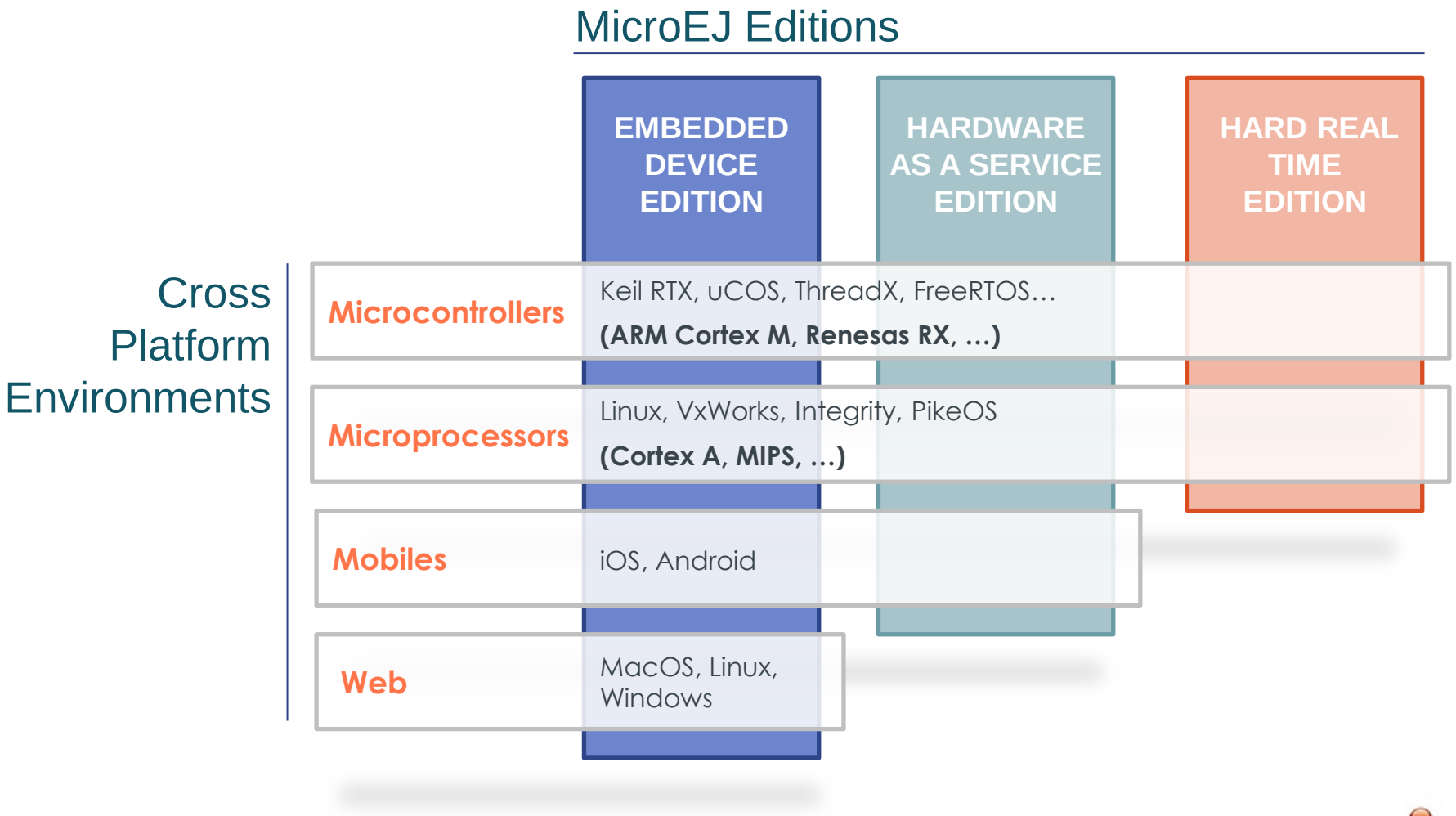


Embedded Market Maturity

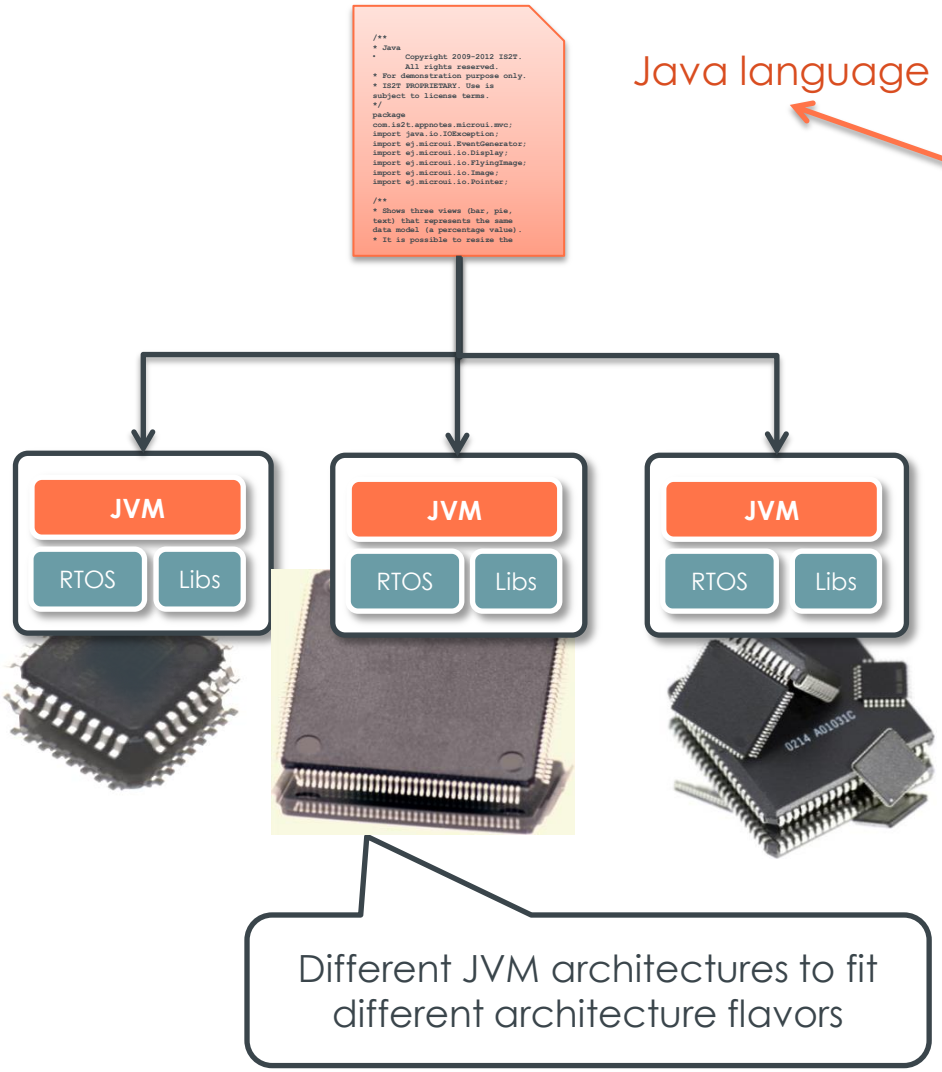
- Like servers, workstations and smartphones…
- … cost constrained embedded systems now look at 3rd party "platform" procurement



MicroEJ Platforms



Java Platforms Concept for Embedded systems



Optimized MicroEJ® VMs

32/64-bit CPU

Optimized implementation

32-bit MCU/MPU

Java

- Code quality
- Productivity
- Reliability
- Portability
- Scalability
- Maintainability
- Code density

Java

- High speed execution
- Determinism (HRT)
- Tiny footprint
- Interface C/asm
- Low power, etc.

Embedded Java Platform Example

life.augmented

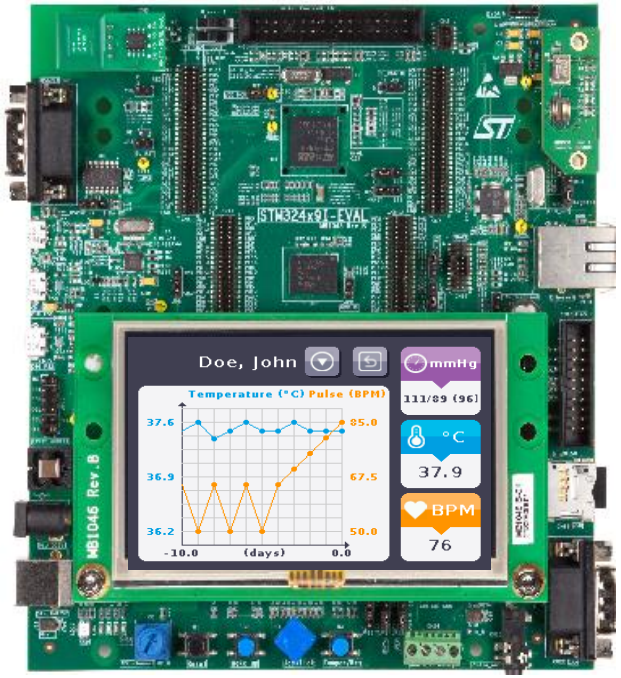
- STM32F2x (Cortex-M3) – 120MHz
- 16-bit col. QVGA LCD, Touch
- APIs: B-ON, MicroUI, MWT, SNI
- Boot time (reset to `main(String[] args)`): 2ms

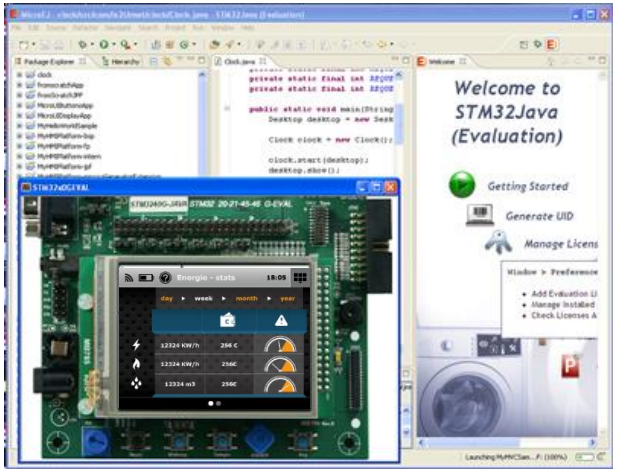
Application Memory Requirements

	Flash (ROM)	422KB	RAM	42KB
Java needs	Virtual Machine (runtime & GC)	28KB	Virtual Machine	1KB
	Libraries (graphics, com, float…)	132KB	Native Stack	28KB
	Graphical resources (images)	228KB		
	Application	34KB	Application	13KB

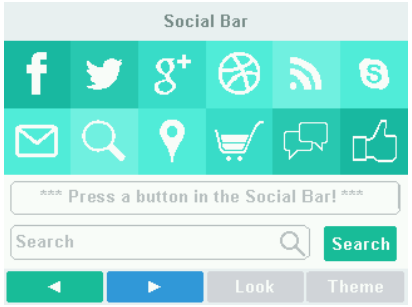
IS2T

GUI Examples on STM32 MCUs





Eclipse IDE

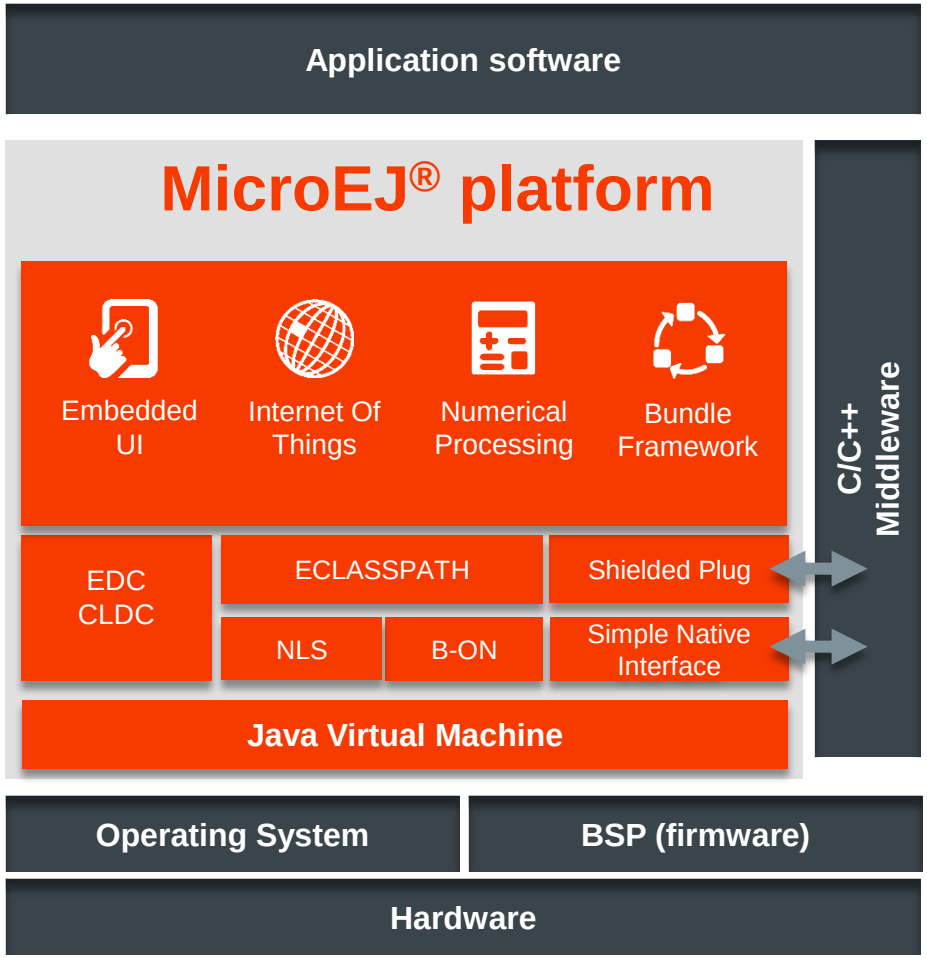


EMBED JAVA TO A LEGACY C BASED APPLICATION

MicroEJ Embedded Devices Edition

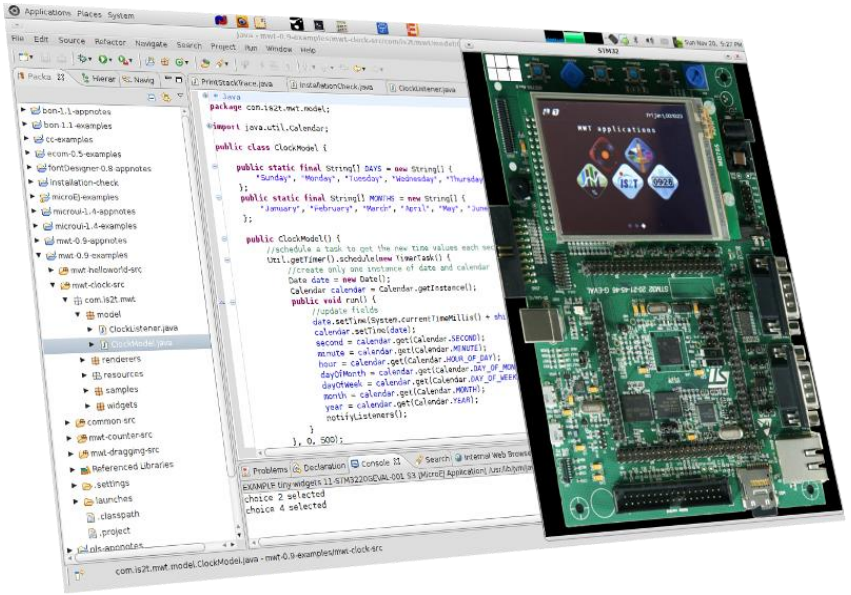
What is a MicroEJ Embedded Platform?

- Dual Java Platform
  - » Embedded platform (EmbJPF)
  - » Simulated platform (SimJPF)
- Integration with legacy
  - » RTOS if any
  - » Firmware & Driver
- General purpose
  - » CLDC/EDC, BON, NLS
- Special packs
  - » UI, IoT, Num, SOA



MicroEJ SDK

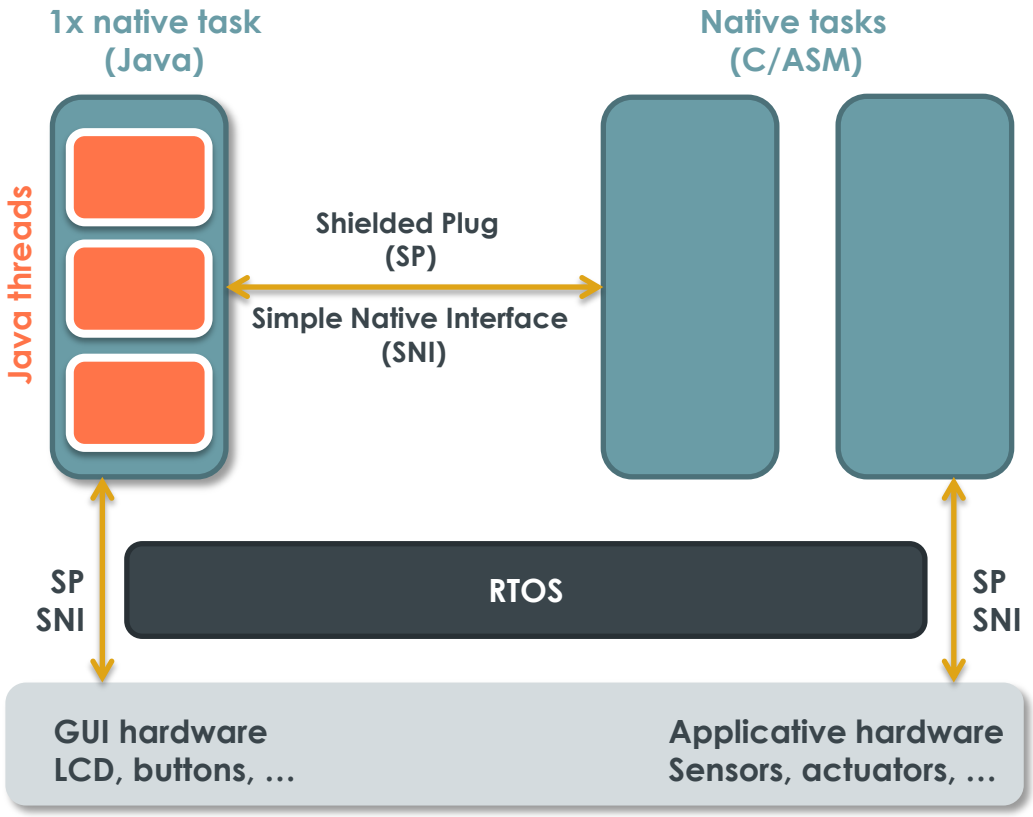
- Java platform design
  - » Integrate to your RTOS
  - » Interface to your C code
  - » Supports ARM-Keil, GNU, IAR, GreenHills, Windriver



- Java application design
  - » Java project editor
  - » Simulate to prototype and debug
  - » Analyze memory usage
  - » Deploy

Easy RTOS Integration (Green Thread)

- Multi-threaded Java execution environment within a single RTOS task



RTOS Examples

- µC/OS, ThreadX, RTX
  FreeRTOS
- Linux, Integrity, VxWorks
- Your RTOS!

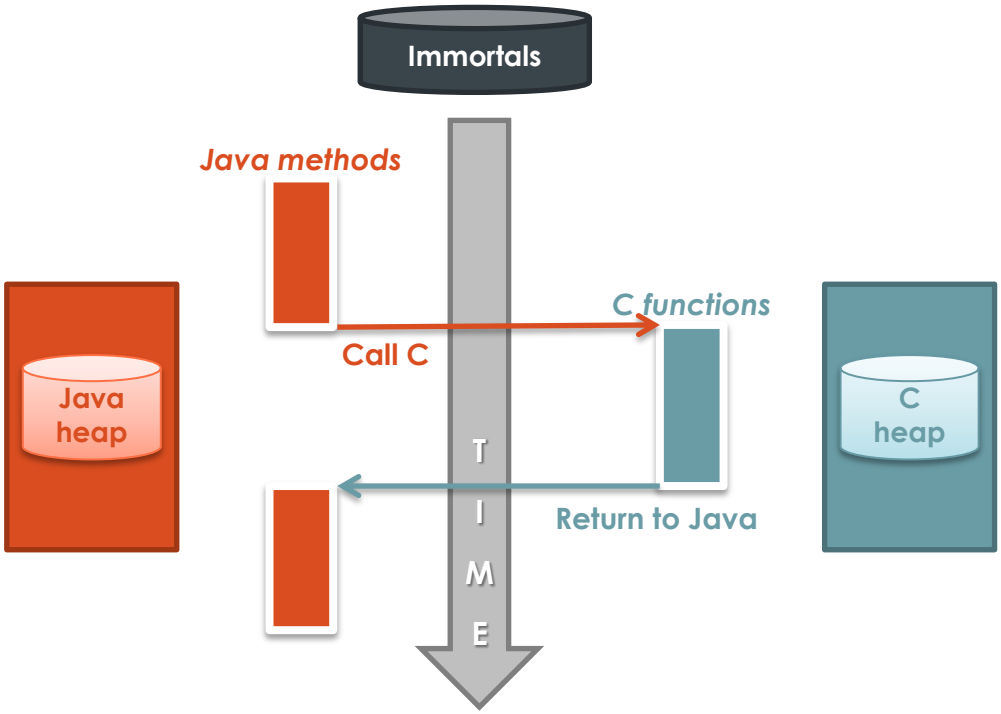
Easy RTOS Integration (Green Thread)

- Same Java thread scheduling policy for all RTOS
  - » Portability improved
    Not only at binary level, but also scheduling level
- Easy control of CPU resource usage for Java world
  - » Java RTOS task priority setting for Java world
  - » CPU resource allocation irrespective of the number of threads
- Java threads & native Tasks synchronization means
  - » Allows synchronous and asynchronous Java / native calls

Easy Java → C Interface (Calls 1/2)

- SNI (ESR 012) : Simple Native Interface
- Call Java world → C/asm
- Arguments: base types (int, float, double , char)

www.e-s-r.net



Easy Java → C Interface (Calls 2/2)

- Easy mapping using naming convention

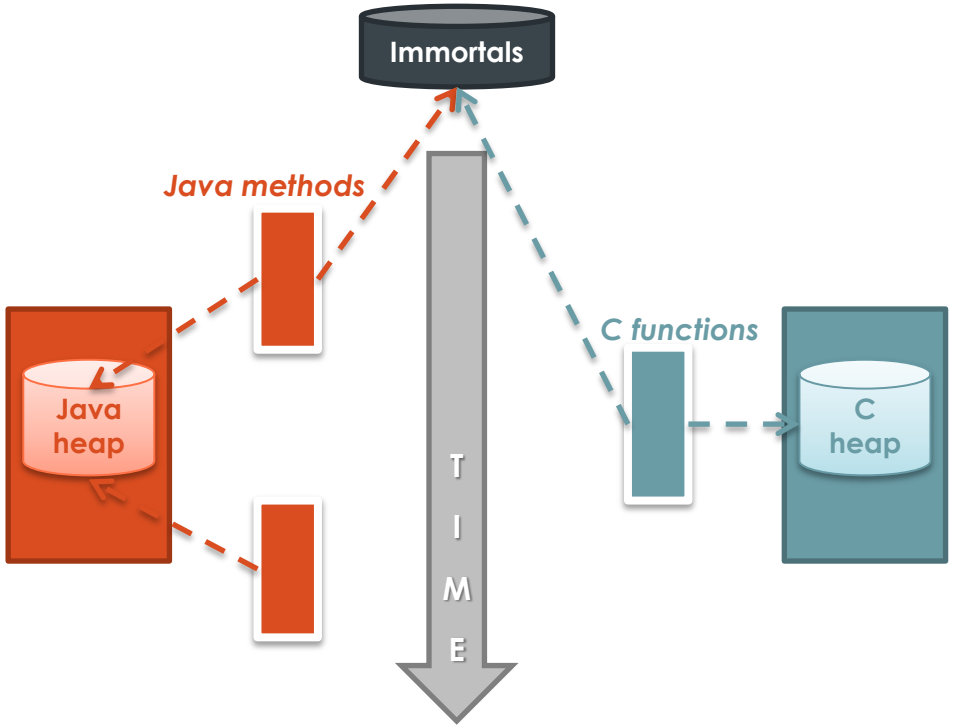
```
package GPIO;

public class Main {
    public static native void toggle();

    public static void main(String[] a) throws InterruptedException
{
        while(true){
            toggle();
            Thread.sleep(10);
        }
    }
}
```

```
#include <sni.h>
#include "gpio.h"

void Java_GPIO_Main_toggle(){
    GPIOE->ODR ^= GPIO_Pin_2 ;
}
```

Easy Java ↔ C Interface (Data 1/2)

- SNI (ESR 012): Simple Native Interface
- Share arrays of base types
- Zero copy buffers and compatible with DMA systems

www.e-s-r.net



Java → C Interface

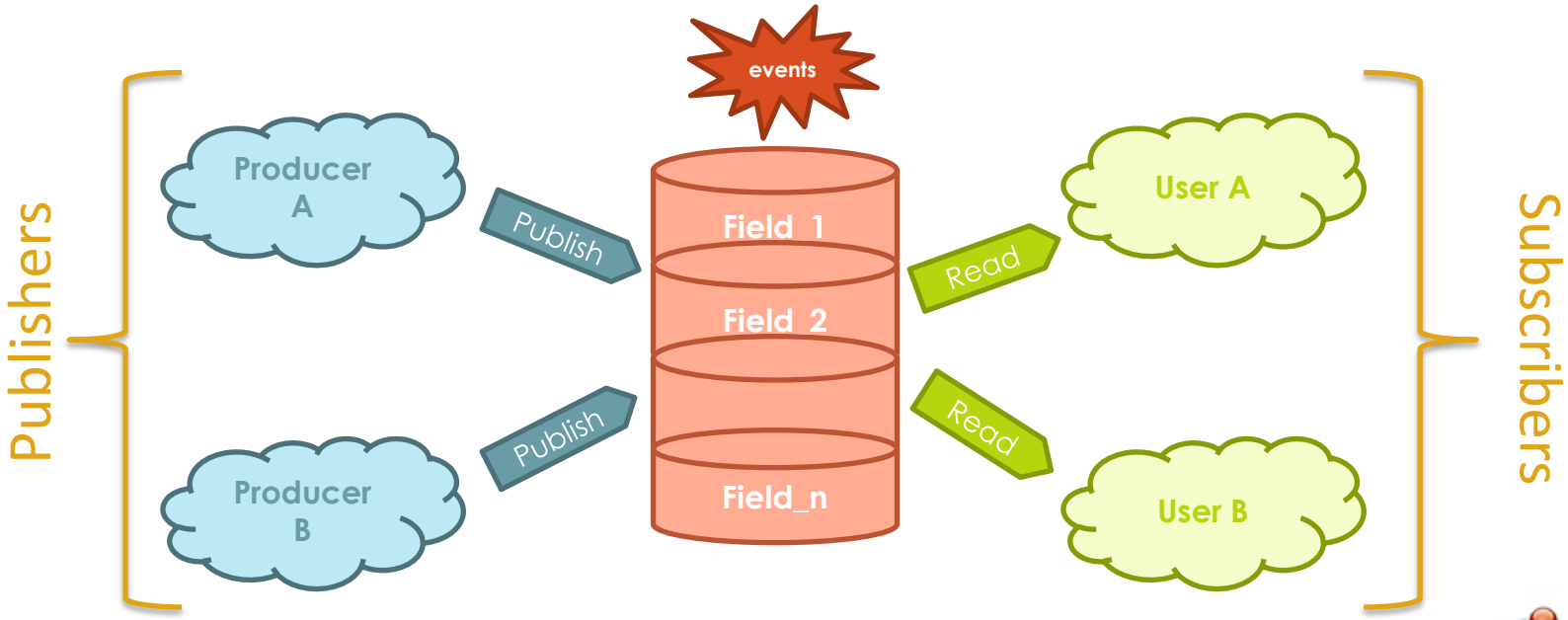
- Immortals are used to share data memory between Java and C

```
package com.corp.examples;
public class Hello {

   static int[] array = (int[])Immortals.setImmortal(new int[50]);
   public static native int getData(int[] array);

   public static void main(String[] args){
      int nb = getData(array);
   }
}
```

```
#include <sni.h>

jint Java_com_corp_examples_Hello_getData(jint* array){
    array[0] = 0xBEEF;
    return 1 ;
}
```

Shielded Plug for Safe & Easy C Integration

- Communication between two separated worlds (Java & native like C/asm)
- Pooling or notification event types
- Spatial & temporal decoupling
- Ideal to add Java tasks on top of a legacy C program



Shielded Plug Java Read Example

```
<shieldedPlug>
          <database name="Forecast" id="0" immutable="true" version="1.0.0">
          <block id="0" name="WIND" length="8" maxTasks="1"/>
          <block id="1" name="TEMP" length="4" maxTasks="1"/>
          <block id="2" name="THERMOSTAT" length="4" maxTasks="1"/>
          </database>
</shieldedPlug>
```

```
public class Wind {
          public int speed; //in ms [0..]
          public int direction; //in degree [0..360]
}
```

```
public class WindReader implements SPReader {
          private static final int SPEED = 0;
          private static final int DIRECTION = 4;
          public Object readObject(ShieldedPlug database, int blockID) throws
                                                                    EmptyBlockException {
                    Wind w = new Wind();
                    byte[] data = new byte[database.getLength(blockID)];
                    database.read(blockID, data);
                    w.speed = ByteArray.readInt(data, SPEED);
                    w.direction = ByteArray.readInt(data, DIRECTION);
                    return w;
          }
}
```

Shielded Plug C Publish Example

```
#include <sp.h>

struct Wind {
        int32_t speed;
        int32_t direction;
};

void windPublication(){
        struct Wind w;
        ShieldedPlug database = SP_getDatabase(Forecast_ID);
        w.speed = speed();
        w.direction = direction();
        SP_write(database, Forecast_WIND, &w);
}
```

Extend the Simulation Platform

- Why building your simulator?
  - » Prototype before having hardware available
- Build your virtual device for UI
  - » Front Panel Designer (buttons, LCD display, LEDs, etc.)
- Build your peripheral extensions (mocks)
  - » Software mocks in Java or C connected to the simulation engine
  - » Hardware mocks over workstation communication interfaces

Extend the Simulation Platform

```
public void runTest(Display display,
String message) {
  MessageViewable viewable = new
MessageViewable(display);
  viewable.init("Hello, world!);
  viewable.show();
 }
```

Application

```
public static void blink() {
  while (true) {
    toggle();
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
  }
}
public static final native boolean
toggle();
```

Simulated Java Platform

Mock Front Panel

JVM (S3)

Native Interface

Mock Interface

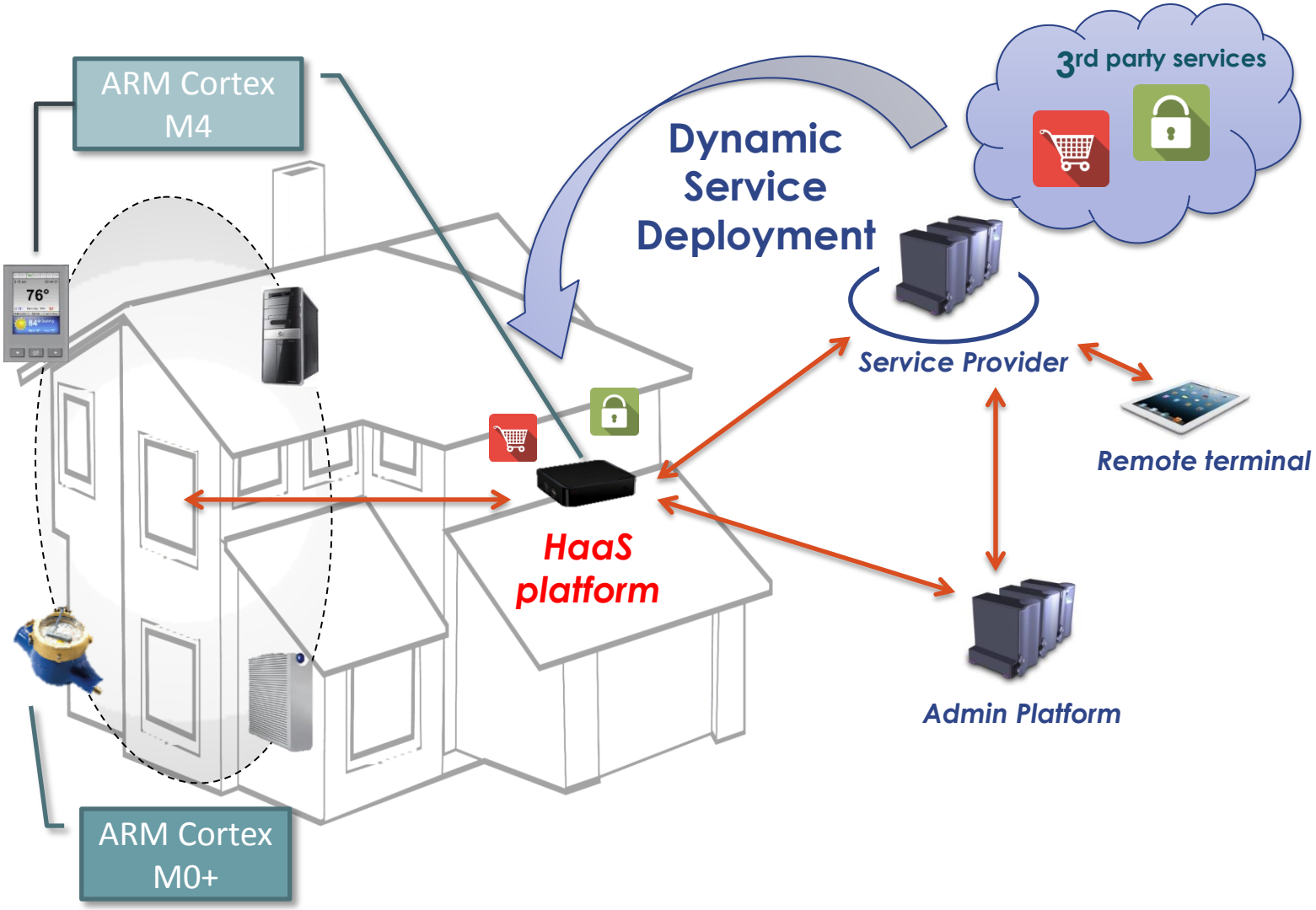
Mock Custom (LED)

```
public class LED {
  private boolean state;

  public static boolean
toggle() {
    state = !state;
    if (state) {

drawImage("ledOn.png");
    } else {

drawImage("ledOff.png");
    }
  return state;
  }
}
```

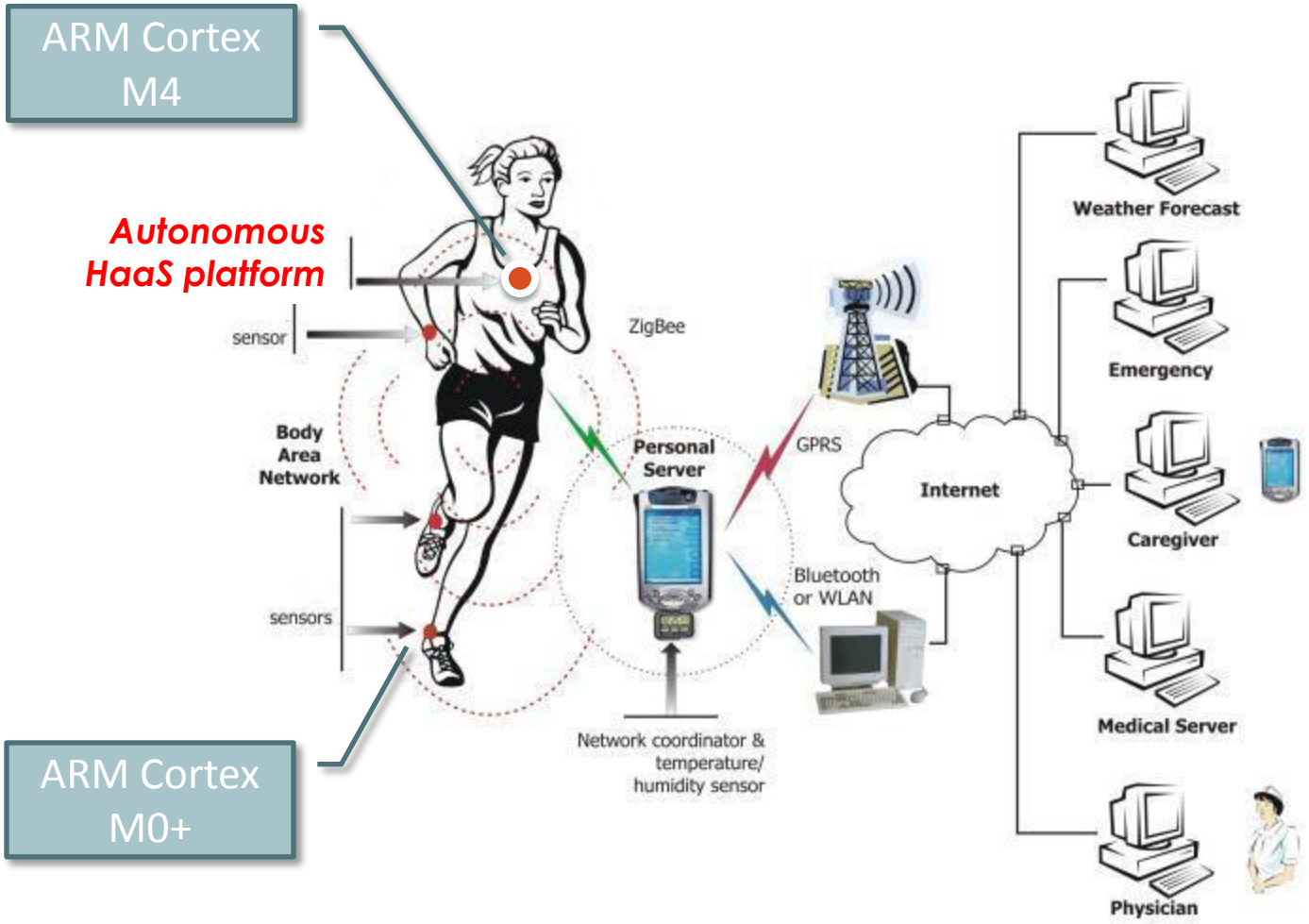
APPLICATION PLATFORMS FOR SMART OBJECTS (IOT)

MicroEJ - Hardware as a Service

HaaS for Home Energy Management



HaaS for Wearable Electronics



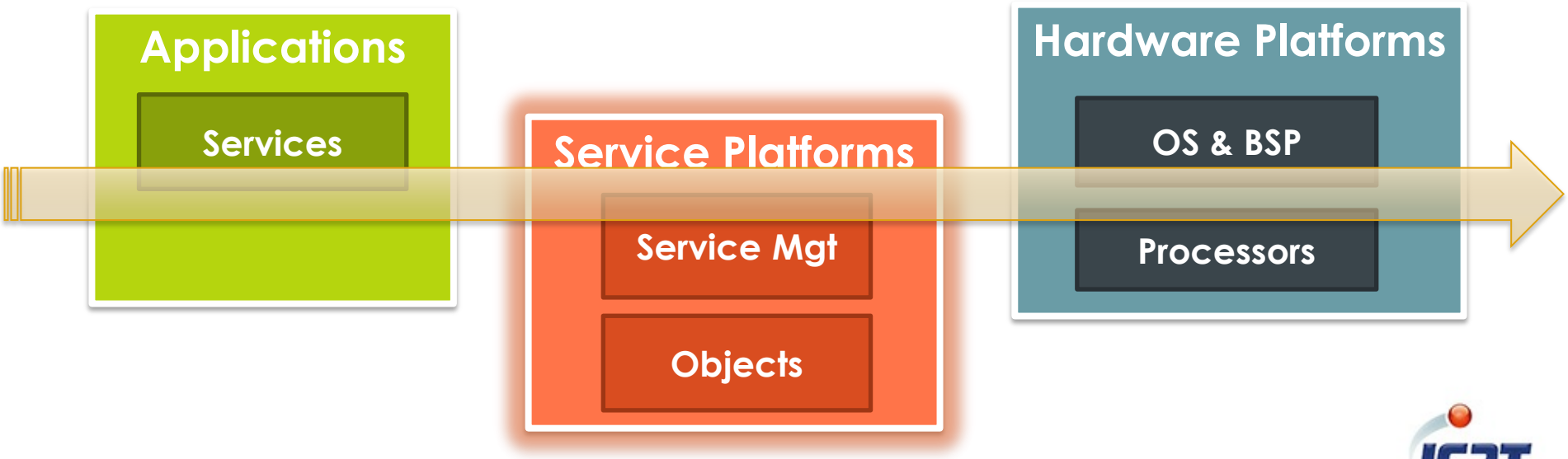
IoT Market Challenges

- Energy efficiency
  - » No bloatware!
- Cost Effectiveness
  - » Small execution environments
- Rich Eco-Systems
  - » More software enablers for innovative business models
- Reliability
  - » Data integrity, service management
- Security
  - » Virtualization, resource management

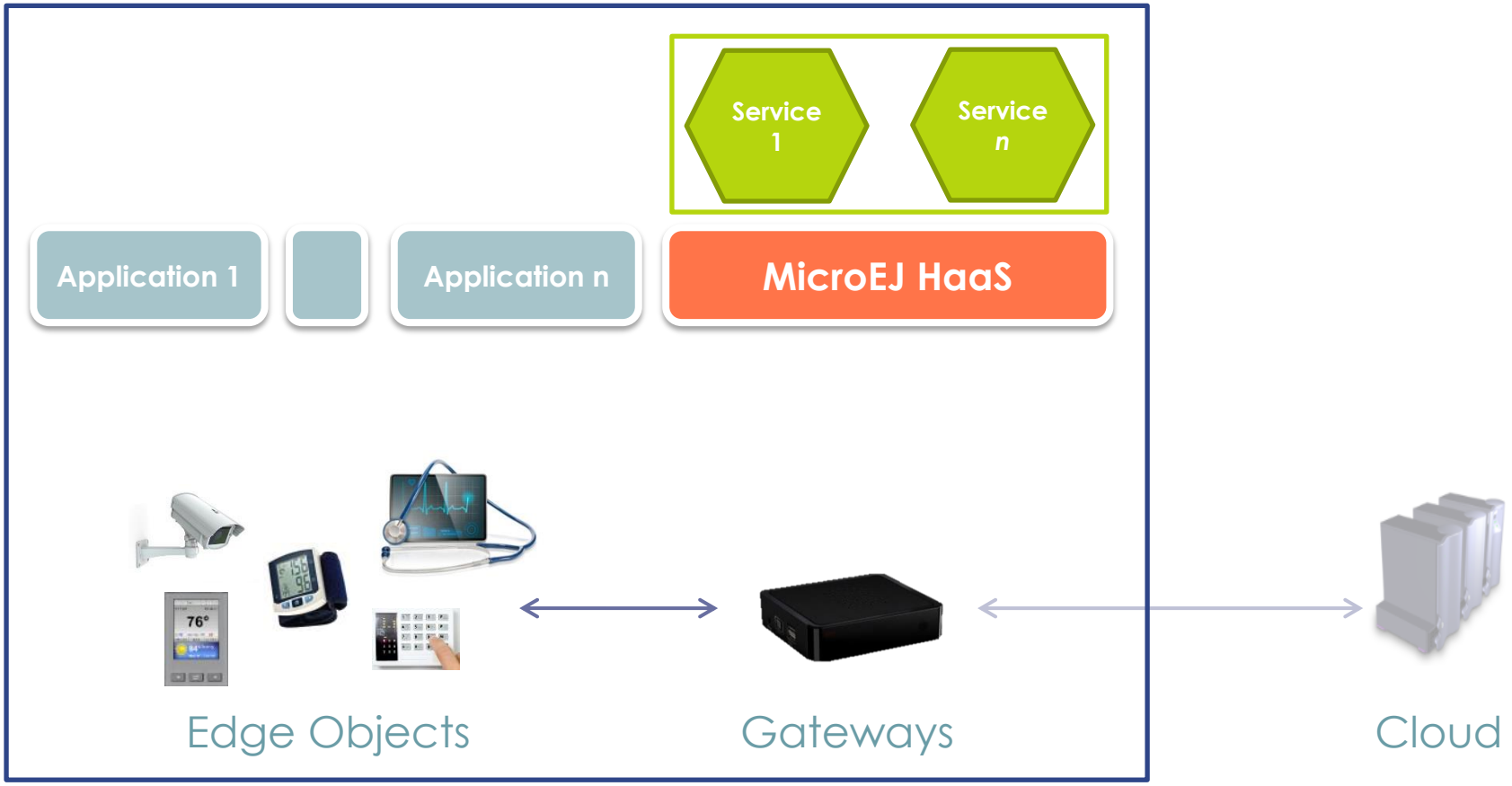
Solution Alignment

- Various topologies for gateways and edge devices
- Time-to-Market can not wait for specific system availability

⌦ *Need unified and portable execution environments*



HaaS Platform Overview

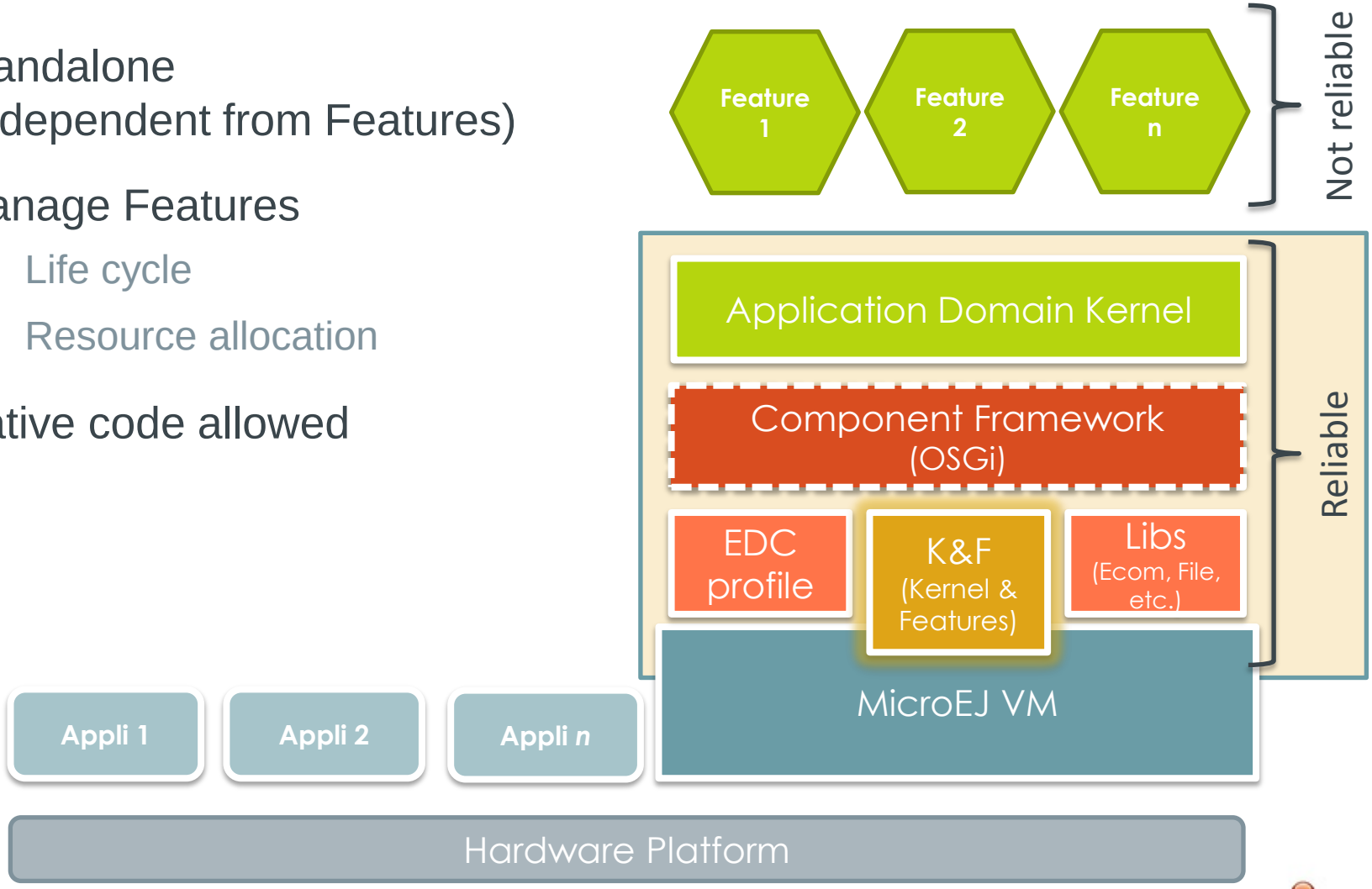


New Capabilities

- Let marketing try new ideas
  - » Try new services fast
- Share your platform with your Eco-System
  - » Provide an open platform with safe isolation capability
- Let your customer choose a product configuration
  - » In the field dynamic service deployment and activation
- Keep using your legacy device base
  - » Use ubiquitous technology with low constraints on hardware

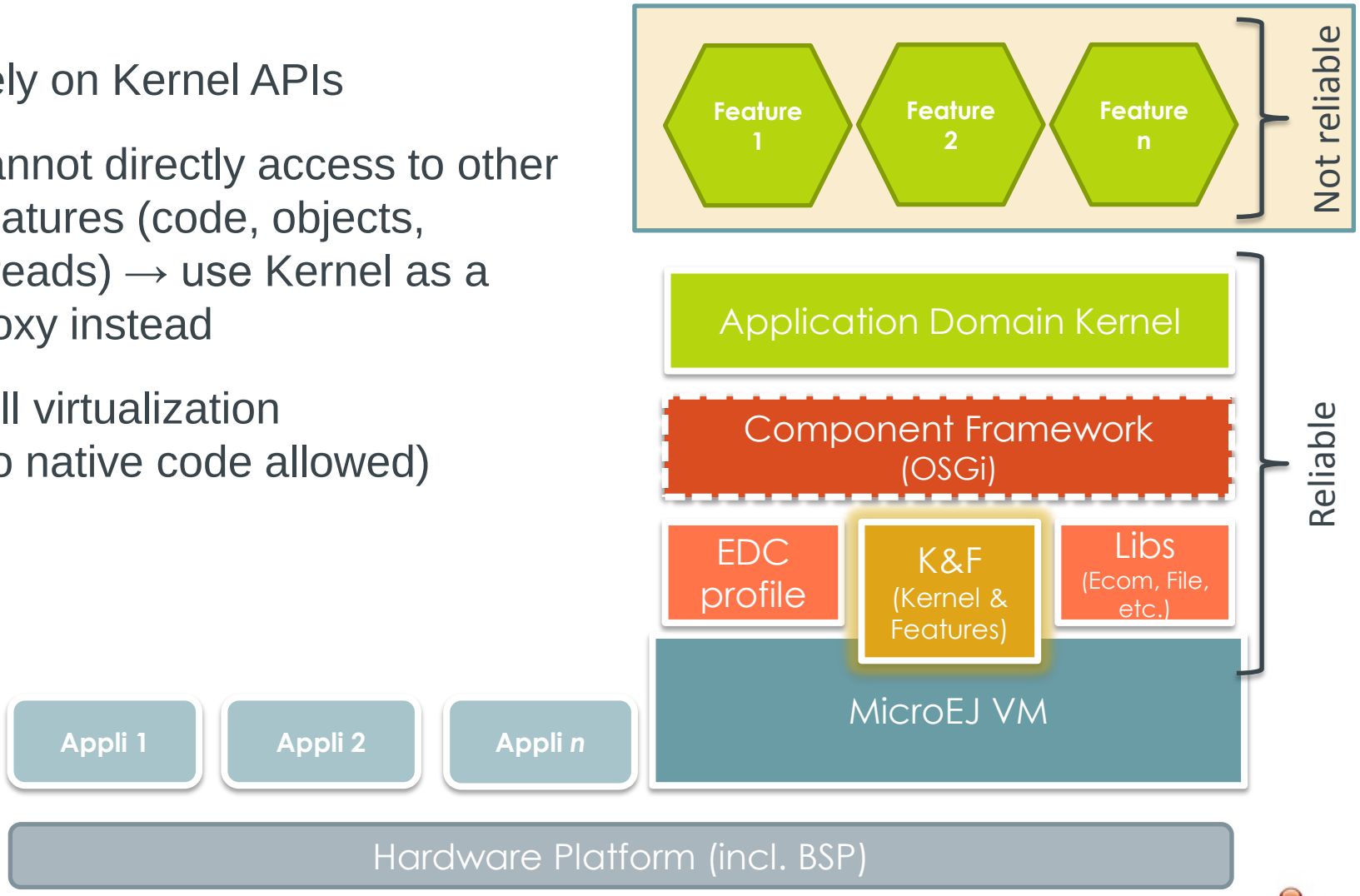
MicroEJ® Haas Architecture – Kernel

- Standalone (independent from Features)
- Manage Features
  - » Life cycle
  - » Resource allocation
- Native code allowed



MicroEJ® HaaS Architecture – Features

- Rely on Kernel APIs
- Cannot directly access to other Features (code, objects, threads) → use Kernel as a proxy instead
- Full virtualization (no native code allowed)



K&F Key Features

- Low consumption & OS agnostic
  - » Kernel & Features: ~20KBytes
  - » Run the same on any RTOS
- Ressources management
  - » CPU and memory allocations
  - » All I/O : file system, TCP/IP, UART, USB, etc.
- Stable & Secure
  - » Kill of a Feature (group of bundles) feasible at any time
    - Threads + objects + code
  - » No back door

K&F and OSGi

- Bundle life cycle management
  - » Load/unload, enable/disable
- Resource management
  - » Bundles cannot access to larger CPU and memory resources than required
  - » Bundles cannot access to physical resource when not allowed to
- Isolation
  - » Bundles is isolated from each others and interface according to the rules defined by the Kernel
- Stable & Secure
  - » Unload has no impact on other Bundles
  - » No stale reference, no zombie threads, etc.

THANK YOU!

More information: www.is2t.com

Evaluation kits: is2t.microej.com