indes
The choice of professionals

Ontwikkelen, Debuggen en Monitoring van IoT / Cloud Device Firmware

Gerard Fianen

INDES Integrated Development Solutions BV

DESIGN AUTOMATION & EMBEDDED SYSTEMS

8 OKT
VAN DER VALK HOTEL
EINDHOVEN

D&E event 2019

Agenda

Security

Remote Update

Remote troubleshooting / IoT Device monitoring

IoT Security - Legal

National Cyber Security Centre
a part of GCHQ

Europe (GDPR) ,
– In effect with fines of €20M or 4% global

The Netherlands (AVG)

Department for Digital, Culture Media & Sport

UK Government (DCMS)
- Legislated enforcement within 3 years with GDPR type fines
- Published Code of Practice for Consumer IoT Security
- 13 Codes of Practice

enisa
European Network and Information Security Agency

ETSI
– Reiterating UK DCMS guidelines

ETSI

EU (ENISA)
– >150 baseline recommendations

Best Practices guides

IoT Security Foundation www.iotsecurityfoundation.org
– Initially UK-centric, now the global forum for IoT Security

https://www.iotsecurityfoundation.org/best-practice-guidelines/

C: Device Secure Boot

The integrity of a device depends critically on executing a trusted boot sequence. A staged boot sequence, where every stage is checked for validity before initialising, minimises the risk of rogue code being run at boot time. Having a fully assured first boot stage is vital to ensuring the subsequent stages can be trusted.

1. Make sure the ROM-based secure boot function is always used. Use a multi-stage bootloader initiated by a minimal amount of read-only code (typically stored in one-time programmable memory).
2. Use a hardware-based tamper-resistant capability (e.g. a microcontroller security subsystem, Secure Access Module (SAM) or Trusted Platform Module (TPM)) to store crucial data items and run the trusted authentication/cryptographic functions required for the boot process. Its limited secure storage capacity must hold the read-only first stage of the bootloader and all other data required to verify the authenticity of firmware.
3. Check each stage of boot code is valid and trusted immediately before running that code. Validating code immediately before its use can reduce the risk of TOCTOU attacks (Time of Check to Time of Use).
4. At each stage of the boot sequence, wherever possible, check that only the expected hardware is present and matches the stage's configuration parameters.
5. Do not boot the next stage of device functionality until the previous stage has been successfully booted.
6. Ensure failures at any stage of the boot sequence fail gracefully into a secure state, to ensure no unauthorised access is gained to underlying systems, code or data (for example, via a uboot prompt). Any code run must have been previously authenticated.

Further discussion on secure booting can be found here.

Resources on how to boot securely are listed below:

- Securing the IoT: Part 1
- Securing the IoT: Part 2
- TOCTOU attacks

Over 100 members and growing…

IoT Security by design

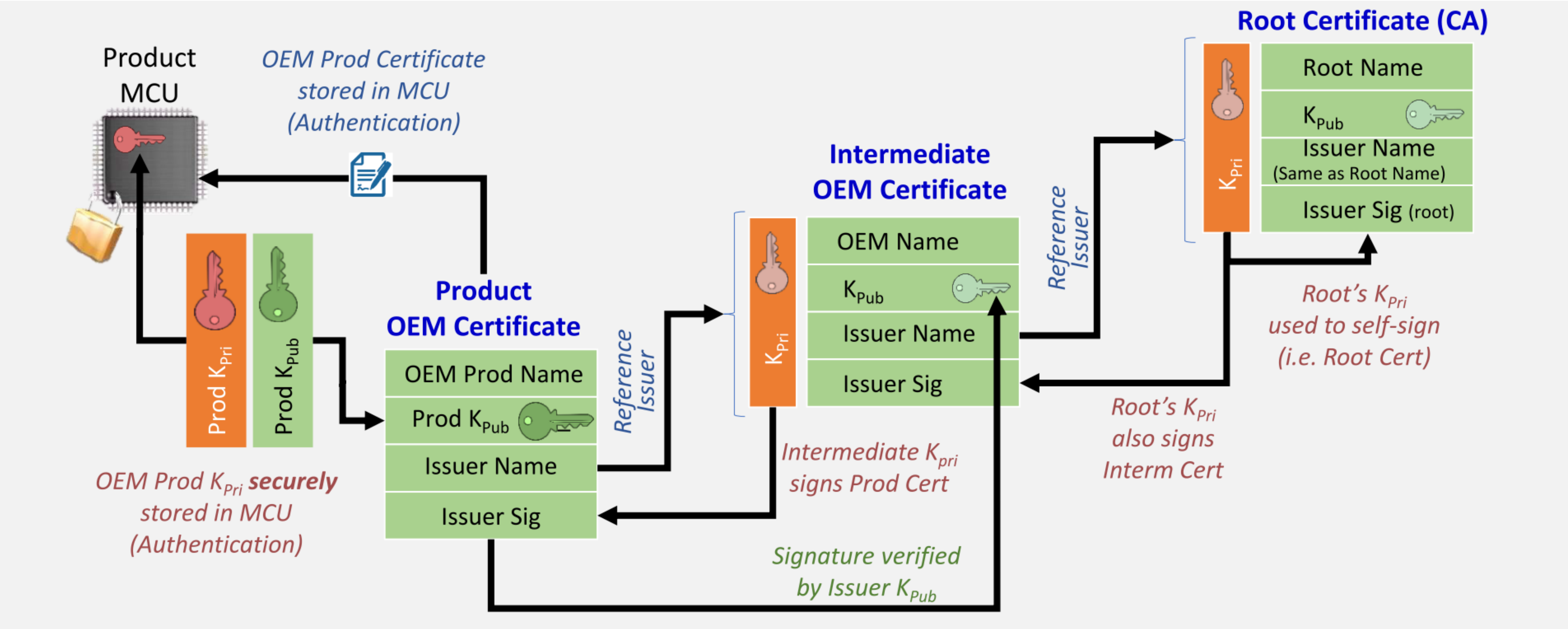
It all starts with the requirements !

1) No default passwords ✓
2) Implement a vulnerability disclosure policy
3) Keep software updated ✓
4) Securely store credentials and security-sensitive data ✓
5) Communicate securely ✓
6) Minimise exposed attack surfaces ✓
7) Ensure software integrity ✓
8) Ensure that personal data is protected ✓
9) Make systems resilient to outages ✓
10) Monitor system telemetry data
11) Make it easy for consumers to delete personal data ✓
12) Make installation and maintenance of devices easy ✓
13) Validate input data ✓
14) Keep verifying for vulnerabilities

The right tools simplify Secure by Design

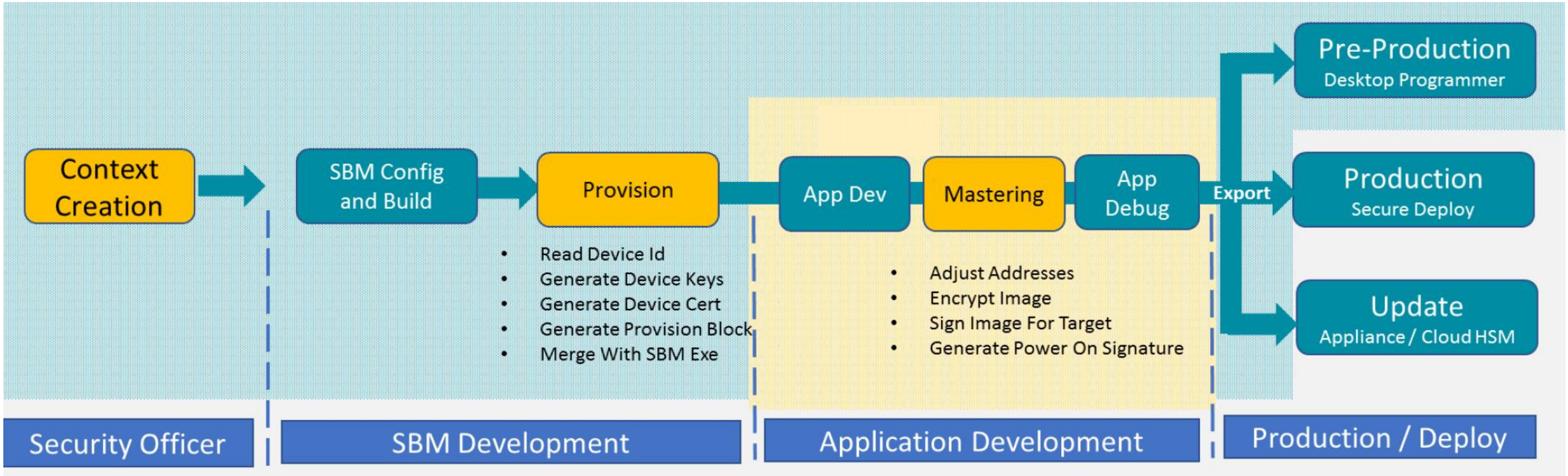
IoT Security by design

indes
The choice of professionals



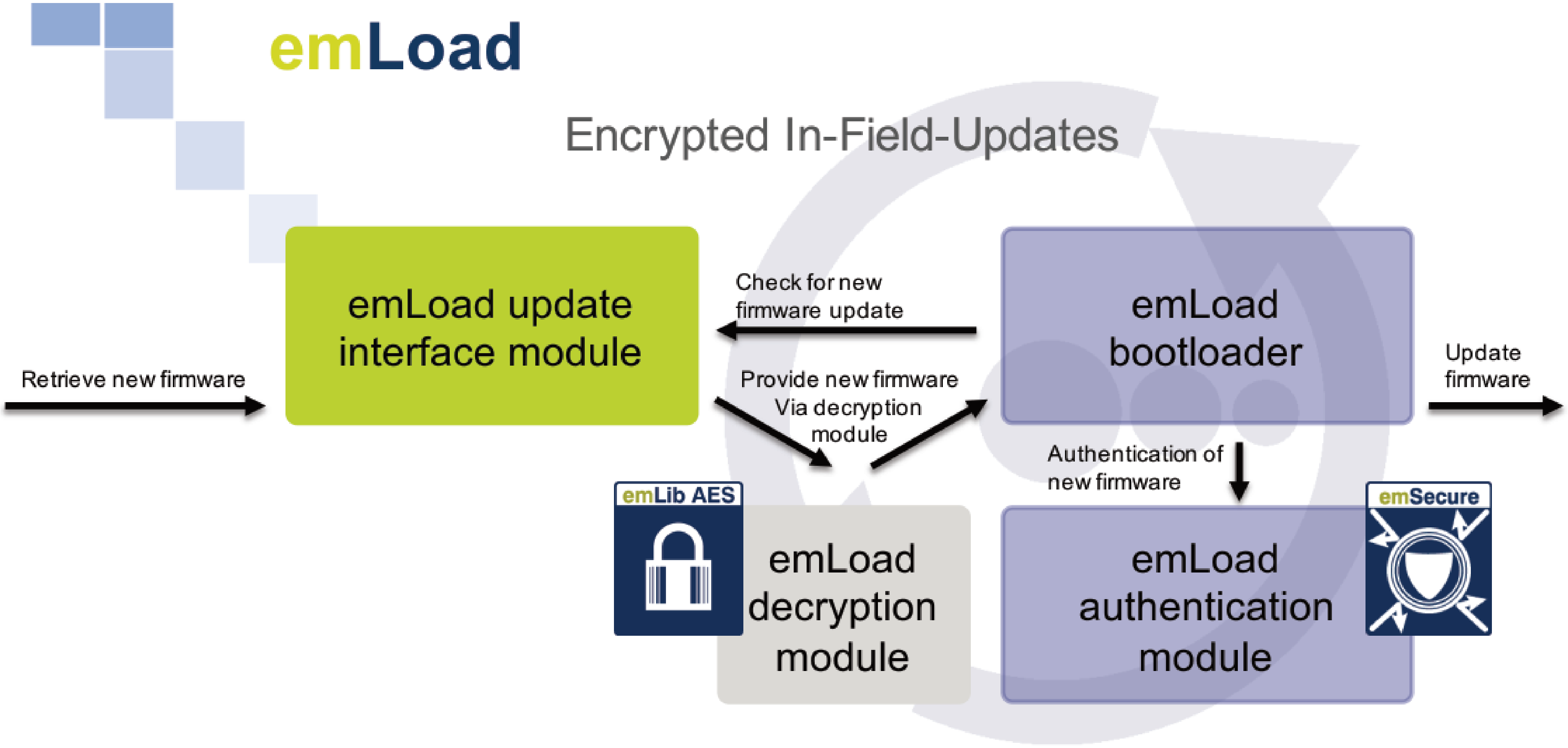
8 OKT
VAN DER VALK HOTEL EINDHOVEN
D&E event 2019

Security in the design-flow

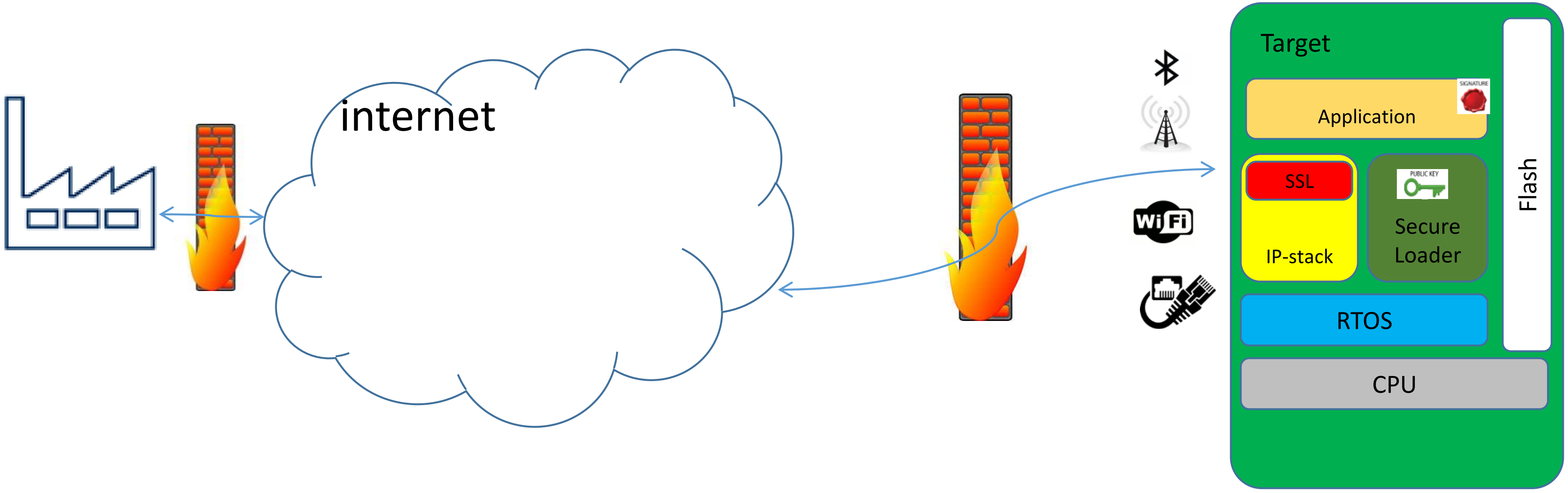


Ideally it should be possible for one person to cover the whole process

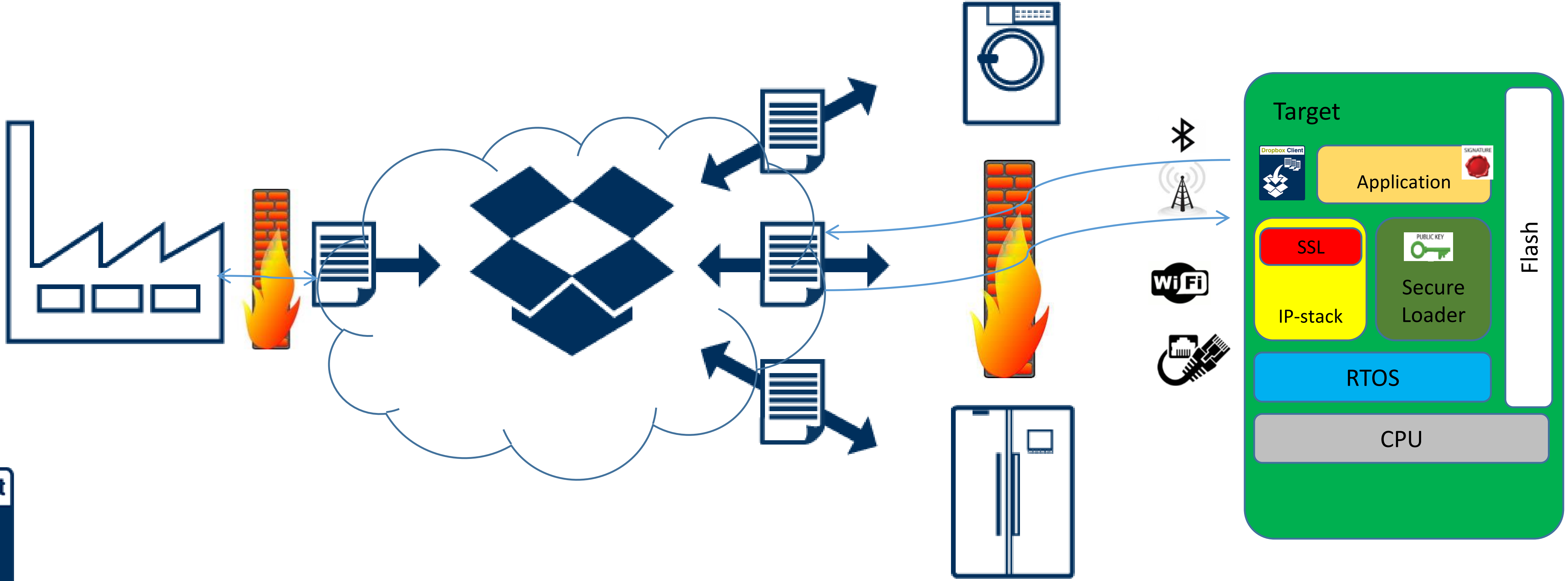
Secure bootloader



Transfer channel ..

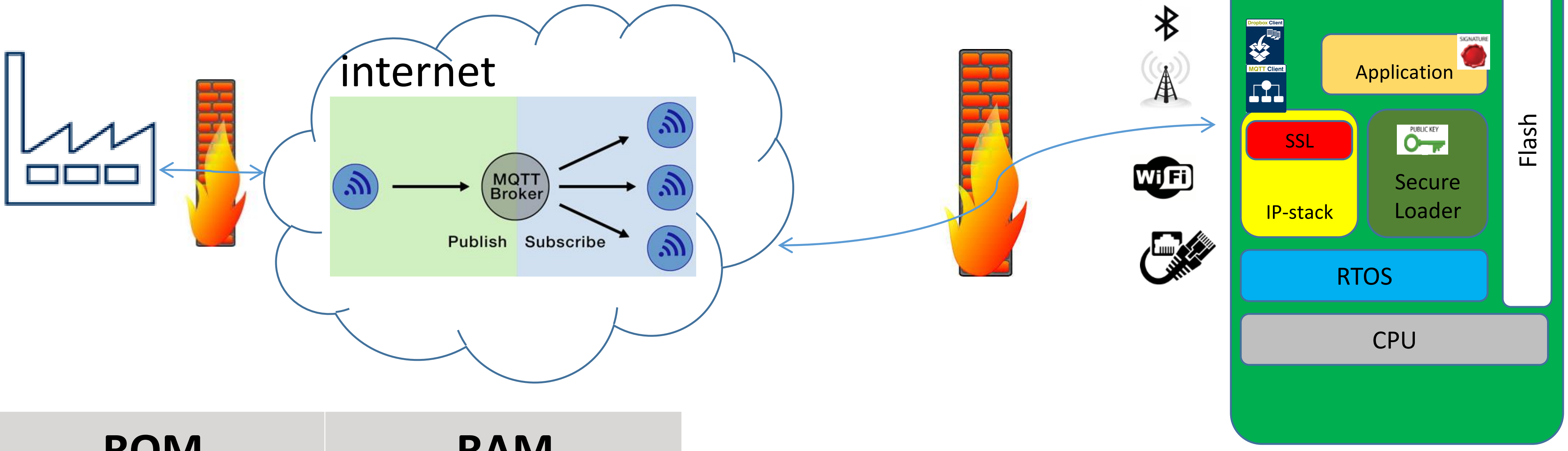


Transfer channel ..



Transfer channel ..

To get messages a from an MQTT broker a subscriber establishes a connection to the broker. The broker checks if a publisher has sent a message for the subscribed topic and if so, sends it to the subscriber. The advantage of this approach is that publisher and subscriber do not need to know each other and that they do not need to run at the same time. All they need to know is the IP address of the broker.

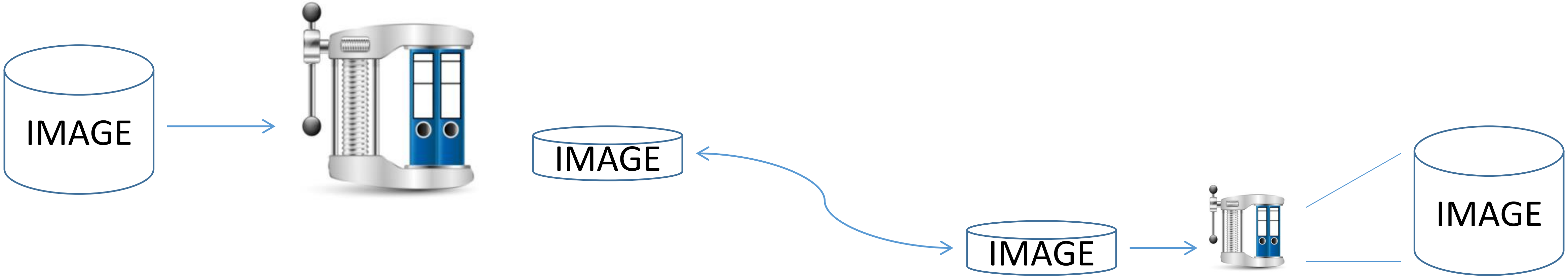


ROM	RAM
Appr. 2.5 kBytes	Appr. 60 bytes

Low bandwith, small memory

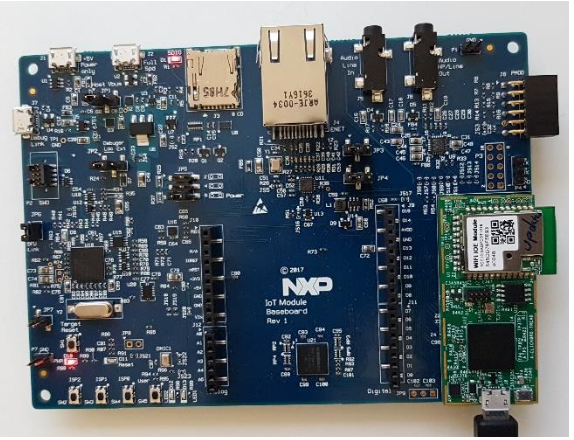
- Embedded "unzip"

Configuration bitstreams to program FPGA and CPLD devices.
Permanent files for embedded web server static content.
Upgrading firmware using a compressed image.
Storing user interface messages for multiple languages.



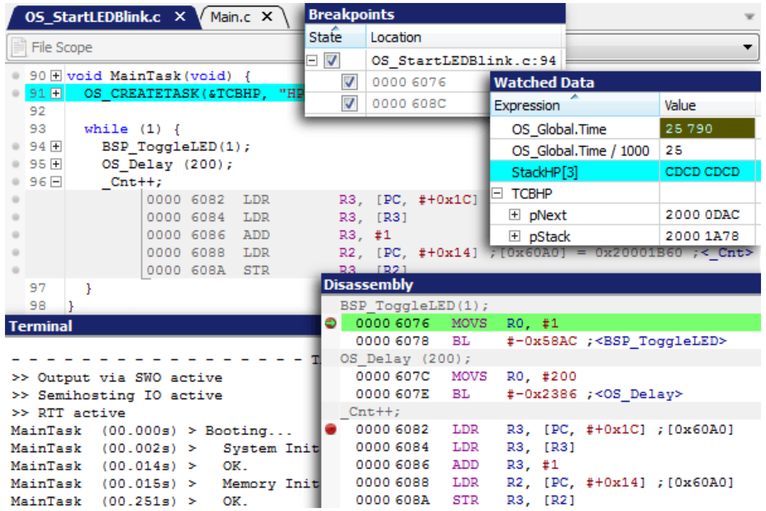
- Highly efficient compression
- (very) Small decompressor ROM footprint
- Fixed decompressor RAM use, chosen when compressing
- Wide range of codecs to choose from
- Automatic selection of best codec for each file

Real-Time (streaming) Trace



SWD / ETM

USB / TCP-IP



Instruction Trace

```
:ime               info
 184140            000100fc -> write r31
 184141  100a8       IncrementCounterBy1   add              %r0,%
 184141            0000002e -> write r0
 184142  100ac       IncrementCounterBy1+0x04  extb
 184142            0000002e -> write r0
 184143  100b0       IncrementCounterBy1+0x08  j_s
 184144  100fc       main+0x30        stb              %r0,[%r1]
 184144                   2e -> write mem [0x011000]
 184145  10100       main+0x34        ldb              %r0,[%r2]
 184145                   5a <- read  mem [0x011001] -> write
 184146            0000005a -> write r0
 184147  10104       main+0x38        bl_s             IncrementCc
 184147            00010106 -> write r31
 184148  100b4       IncrementCounterBy2   add              %r0,%
 184148            0000005c -> write r0
 184149  100b8       IncrementCounterBy2+0x04  extb
 184149            0000005c -> write r0
 184150  100bc       IncrementCounterBy2+0x08  j_s
 184151  10106       main+0x3a        stb              %r0,[%r2]
```

Printf / Application logging

```
Svc] Starting key provisioning...
Svc] Write root certificate...
Svc] Write device private key...
 Svc] Write device certificate...
 Svc] Key provisioning done...
 Svc] Starting WiFi...
mr Svc] WiFi module initialized.
WS-MAIN] WiFi connected to AP AndroidAP.
AWS-MAIN] Attempt to Get IP.
AWS-MAIN] IP Address acquired 192.168.0.51
AWS-LED] [Shadow 0] MQTT: Creation of dedicated MQT
AWS-LED] Sending command to MQTT task.
MQTT] Received message 10000 from queue.
MQTT] Looked up a7sw0r7rvpirn.iot.us-east-1.amazona
[MQTT] MQTT Connect was accepted. Connection establ
[MQTT] Notifying task.
[AWS-LED] Command sent to MQTT task passed.
```

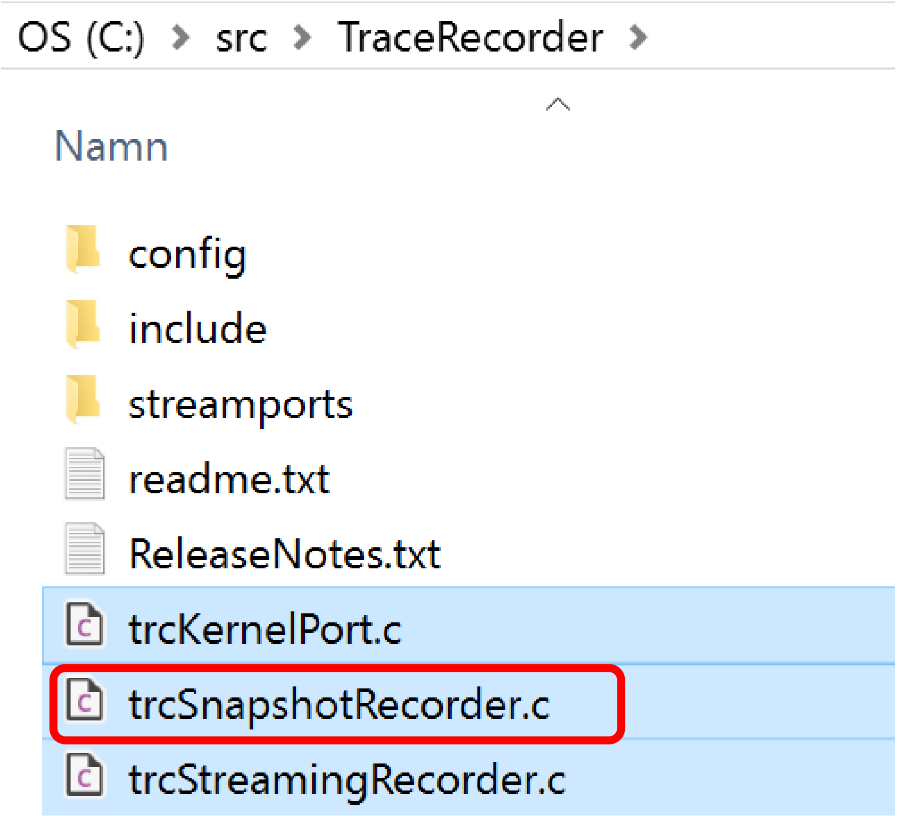
Bringing Trace to another level : Event Trace

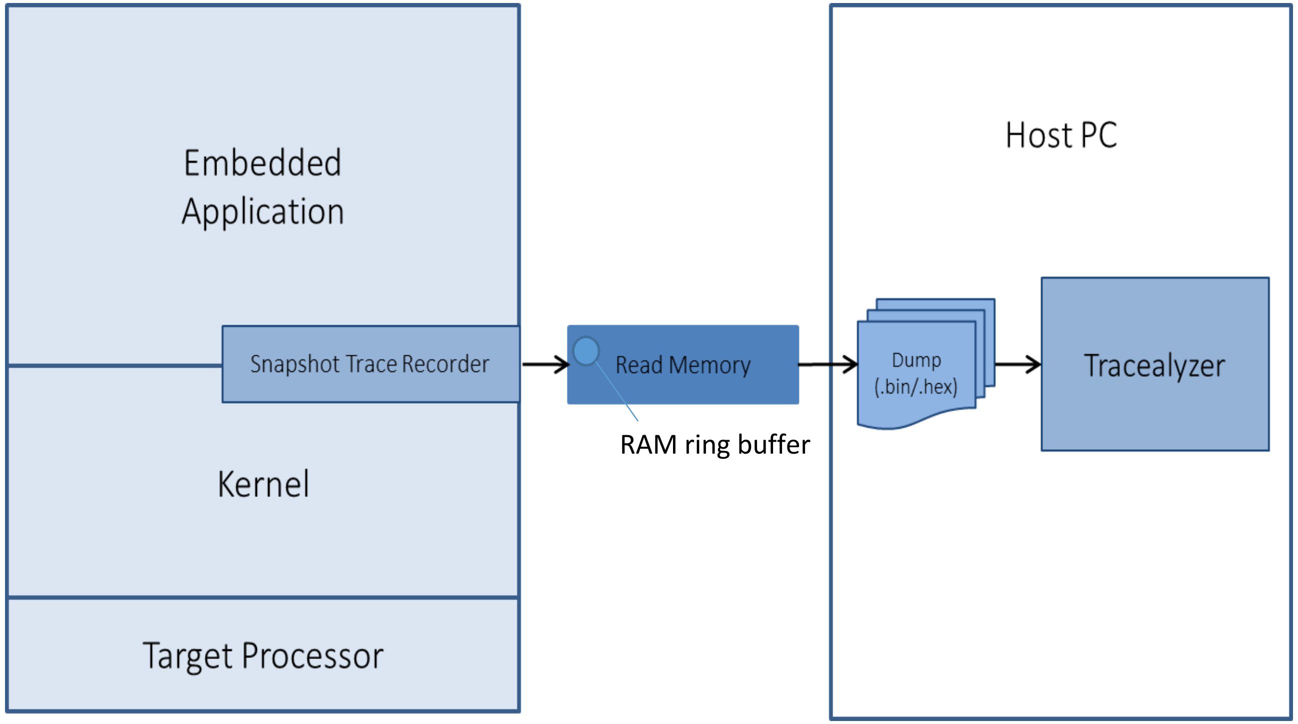
indes
The choice of professionals

```
09.319]    Context switch on CPU 0 to Control
09.330]    xQueueReceive(CtrlDataQueue, 100) retur
10.253]    OS Ticks: 8109
11.253]    OS Ticks: 8110
11.270]    Context switch on CPU 0 to Pos_ADC_ISR
11.281]    xQueueSendFromISR(CtrlDataQueue)
11.290]    Context switch on CPU 0 to Control
11.868]    xQueueSend(MotorQueue)
11.878]    Actor Ready: Motor
11.889]    Context switch on CPU 0 to Motor
11.900]    xQueueReceive(MotorQueue, 10) returns s
11.934]    xQueueReceive(MotorQueue, 10) blocks
11.954]    Context switch on CPU 0 to Control
11.965]    xQueueReceive(CtrlCmdQueue, 0) timeout/
11.977]    xQueueReceive(CtrlDataQueue, 100)
11.990]    xQueueReceive(CtrlCmdQueue, 0) timeout/
```

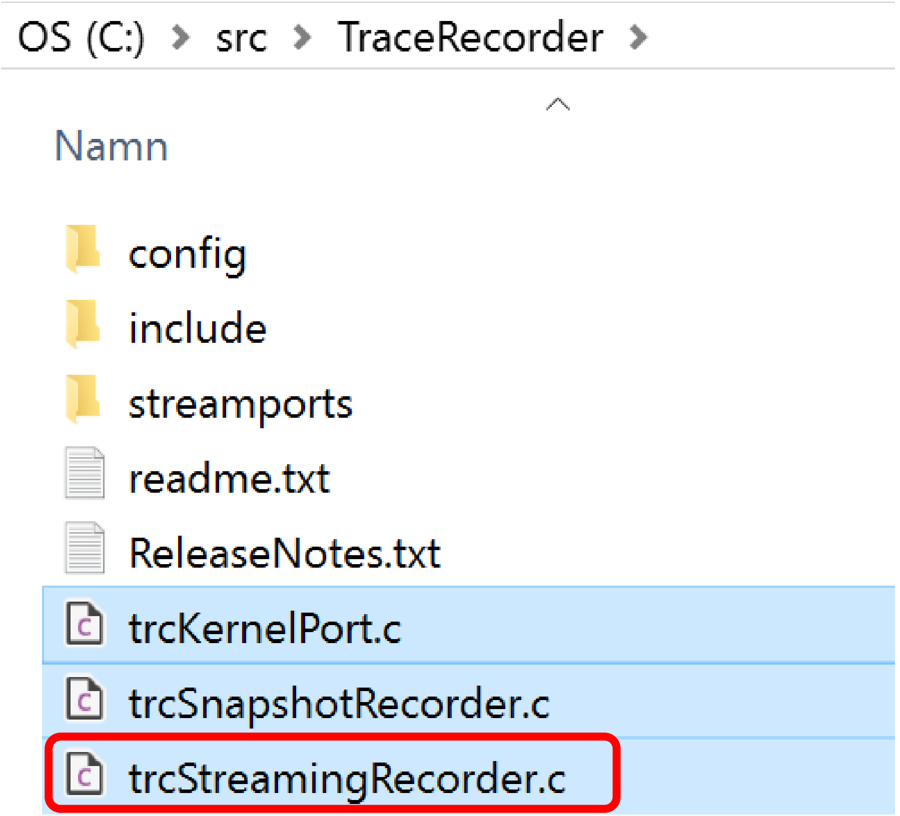
	Instruction Trace	**Event Trace**	**Application Logging**
Producer	Processor core	Software (API or Kernel)	Software (application)
Abstraction Level	Low	Medium	High
Overhead	None	Some	More
System Requirements	High	Low	Low
Flexibility	Low	High	High

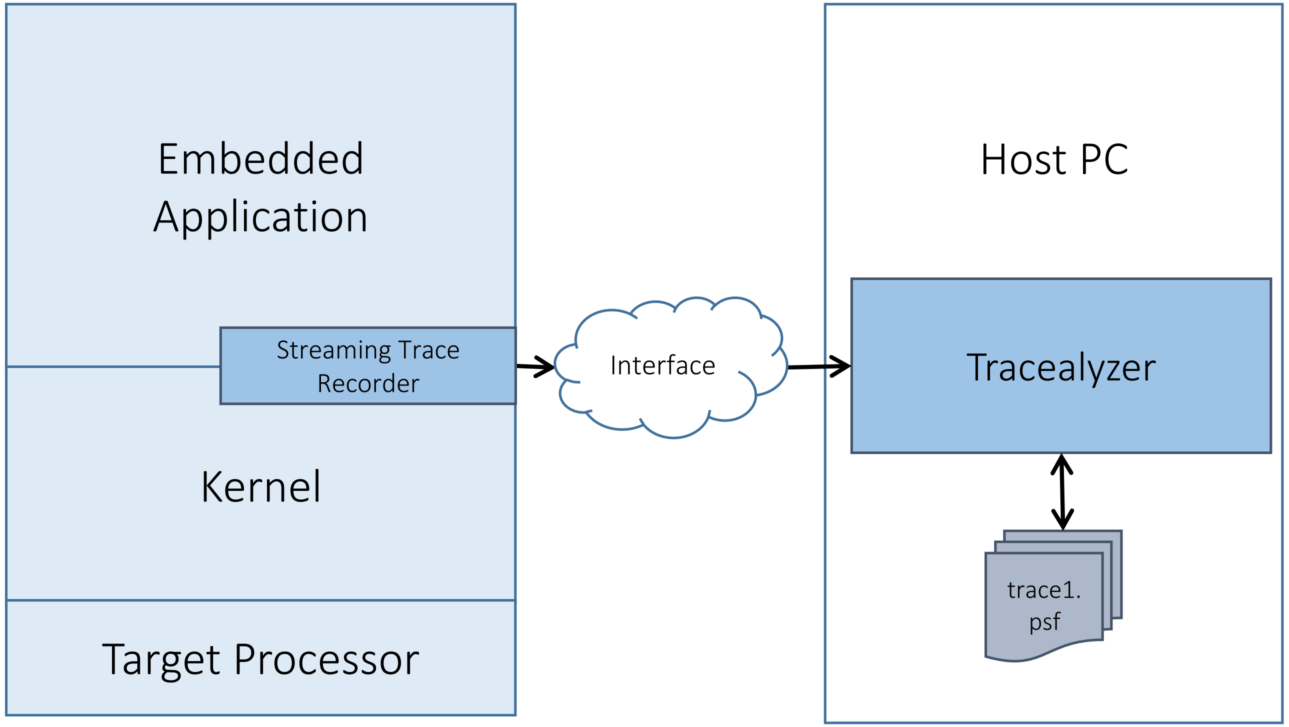
Example : FreeRTOS implementation





Example : FreeRTOS implementation





Event Visualization - 1

indes
The choice of professionals

What can be studied? Some examples:

- Multi-threading and timing
  - Context switches, internal kernel events
  - Execution time, response time, periodicity…
- API calls (OS, Middleware stacks, Drivers…)
  - Call sequences and timing
  - Parameters and return values
  - Blocking and timeouts
  - Object dependencies
- Application logging
  - Debug messages, variable values…
- Time between important events
- State changes over time

Event Visualization -2

indes
The choice of professionals

RTOS Analytics

- Service Call Block Time
- Object Utilization
- Message Receive Time
- Priority Changes
- Event Intensity (three types)

CPU Usage

- CPU Load Graph
- Actor Statistics Report
- Actor Instance Graphs
- Interval Coverage Graph

Application Analytics

- Communication Flow
- State Machine Graph
- User Events Signal Plot
- I/O Intensity and I/O Plot

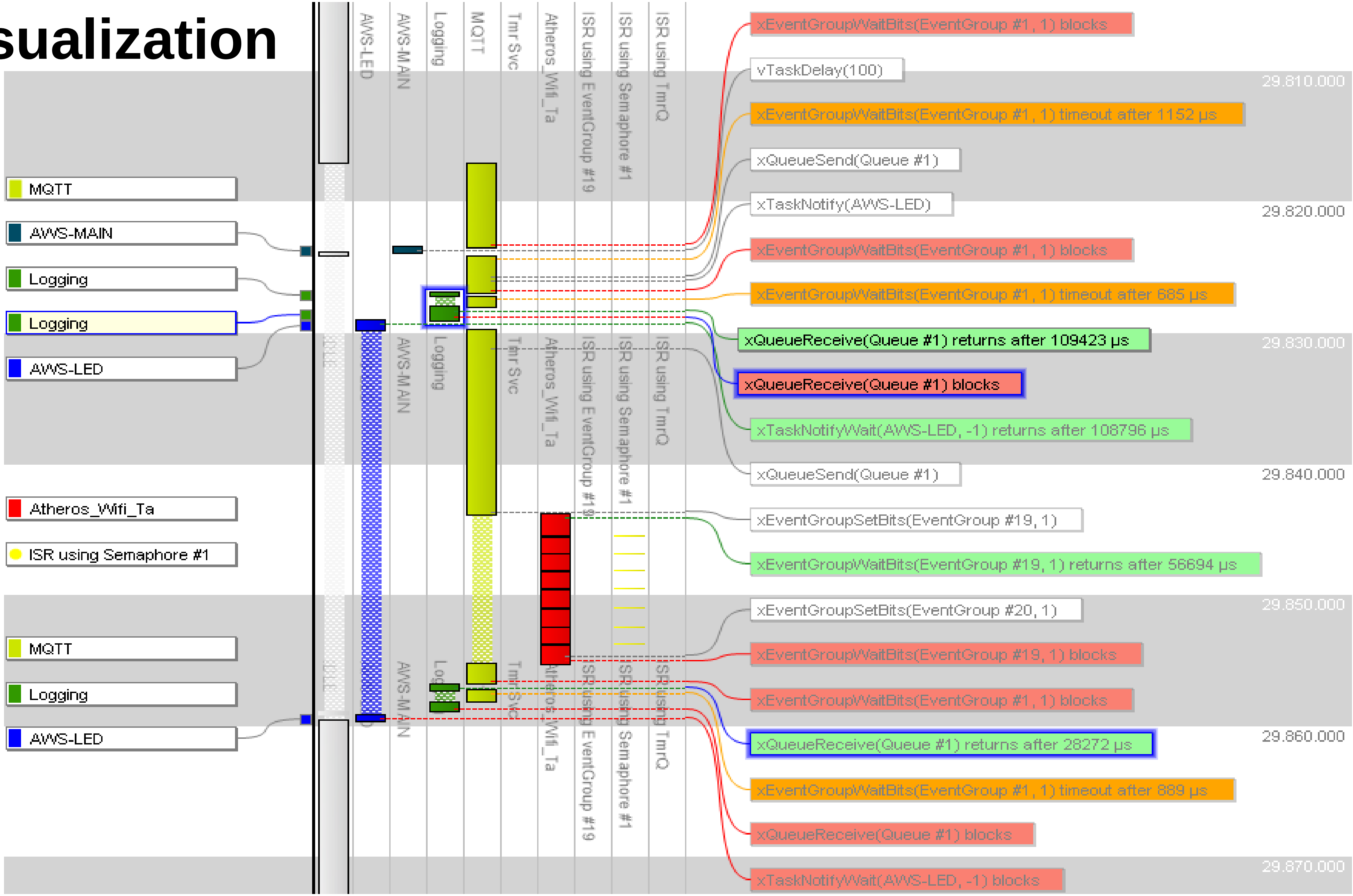
Memory Usage

- Stack Usage
- Memory Heap Utilization

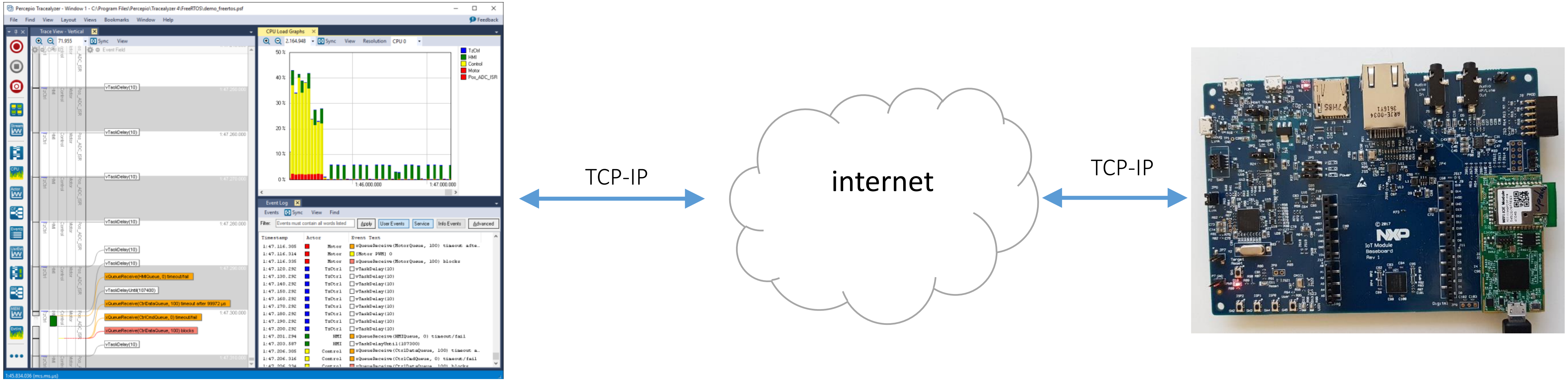
All views are **interconnected** in clever ways, so you can click on a data point in one view and see the corresponding location in another related view.

percepio
SENSING SOFTWARE

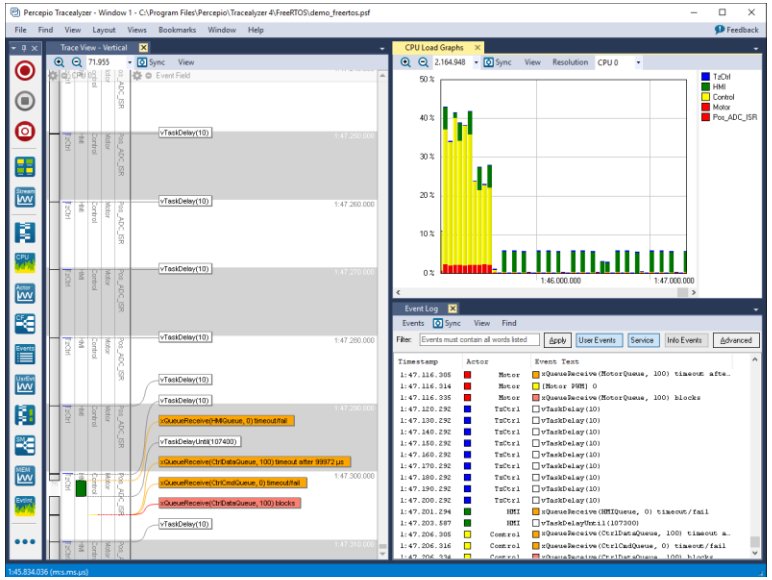
Event Visualization



Remote "streaming" Trace

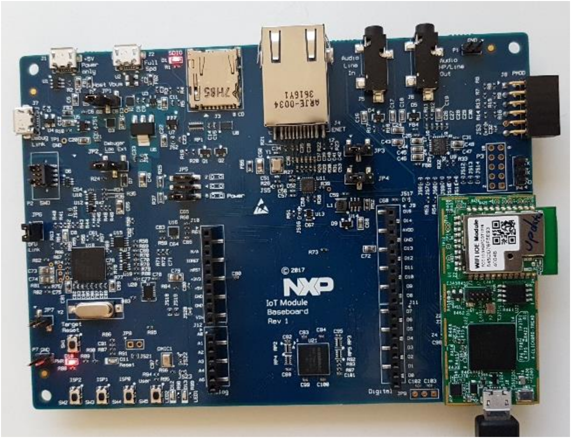


Remote Device Monitoring - 1



AWS SDK

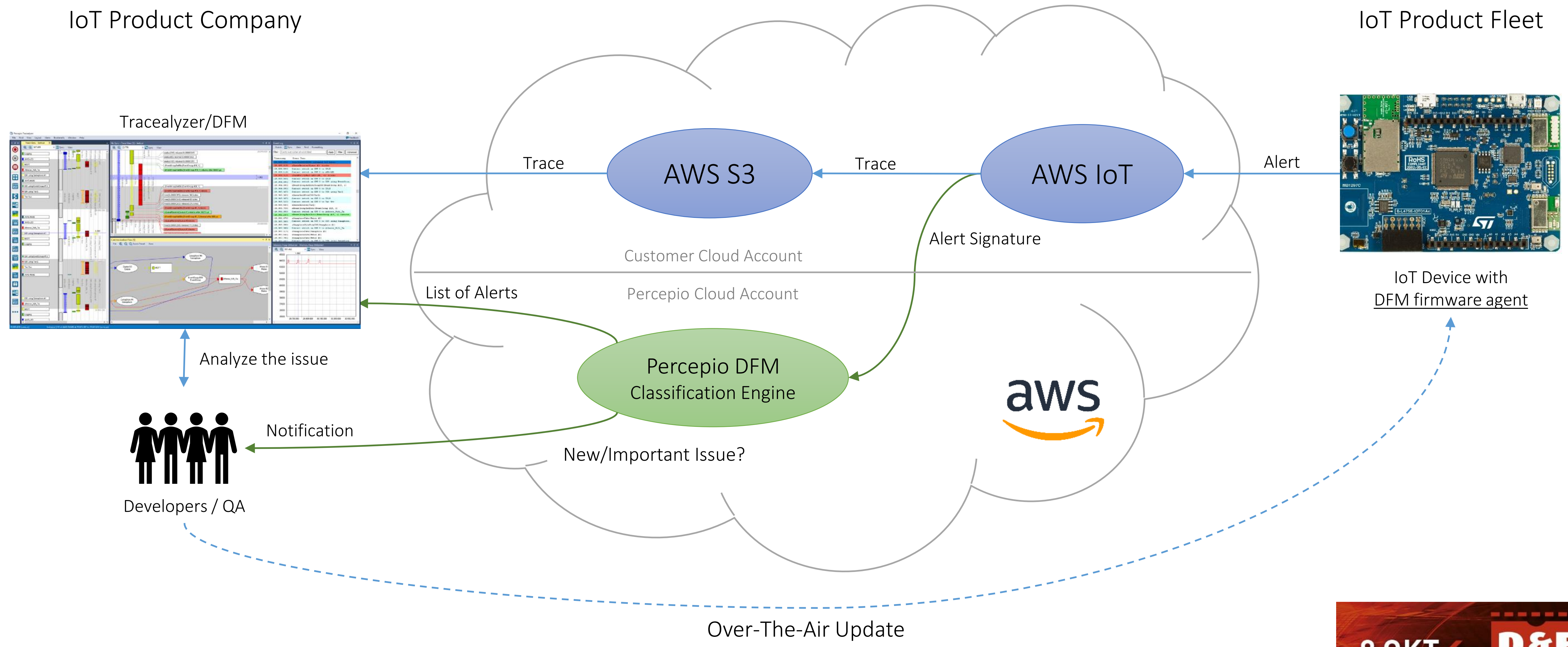
AWS IoT Core



- Alerts generated by e.g. asserts and fault handlers
- Compact trace format
  – less than 5 KB needed
  for350 ms trace, during a busy period, with many details
- Post-Mortem Debug: Data can survive a crash – uploaded after restart

Remote Device Monitoring - 2

indes
The choice of professionals



INDES –
Integrated Development Solutions BV

Voor vragen : wij staan op stand 23

www.indes.com info@indes.com gerard@indes.com

8 OKT
VAN DER VALK HOTEL
EINDHOVEN
D&E event 2019