IoT
Online
Conference

percepio®

www.iotonlineconference.com

Continuous Software Improvement with IoT Device Monitoring and Visual Trace Diagnostics

Johan **Kraft**

AGENDA

1 Introduction – Challenges in System Testing

2 IoT Device Monitoring – Concept and Benefits

3 Hands-on Demo

THE SPEAKER

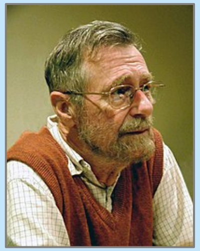
Dr. Johan Kraft

→ CEO, CTO and founder, Percepio AB

Focus: embedded software tracing and visualization for simplified development

Original developer of Percepio’s first product for visual trace diagnostics, Tracealyzer, and the founder of the company. Background in applied academic research in collaboration with industry, focused on embedded software timing analysis, and embedded software development at ABB Robotics. PhD in computer science.

SOFTWARE TESTING – NO GUARANTEE



> “Testing only shows the presence, not absence of bugs.”
> Edsger W. Dijkstra, 1930-2002

According to software engineering research on 13,500 projects *

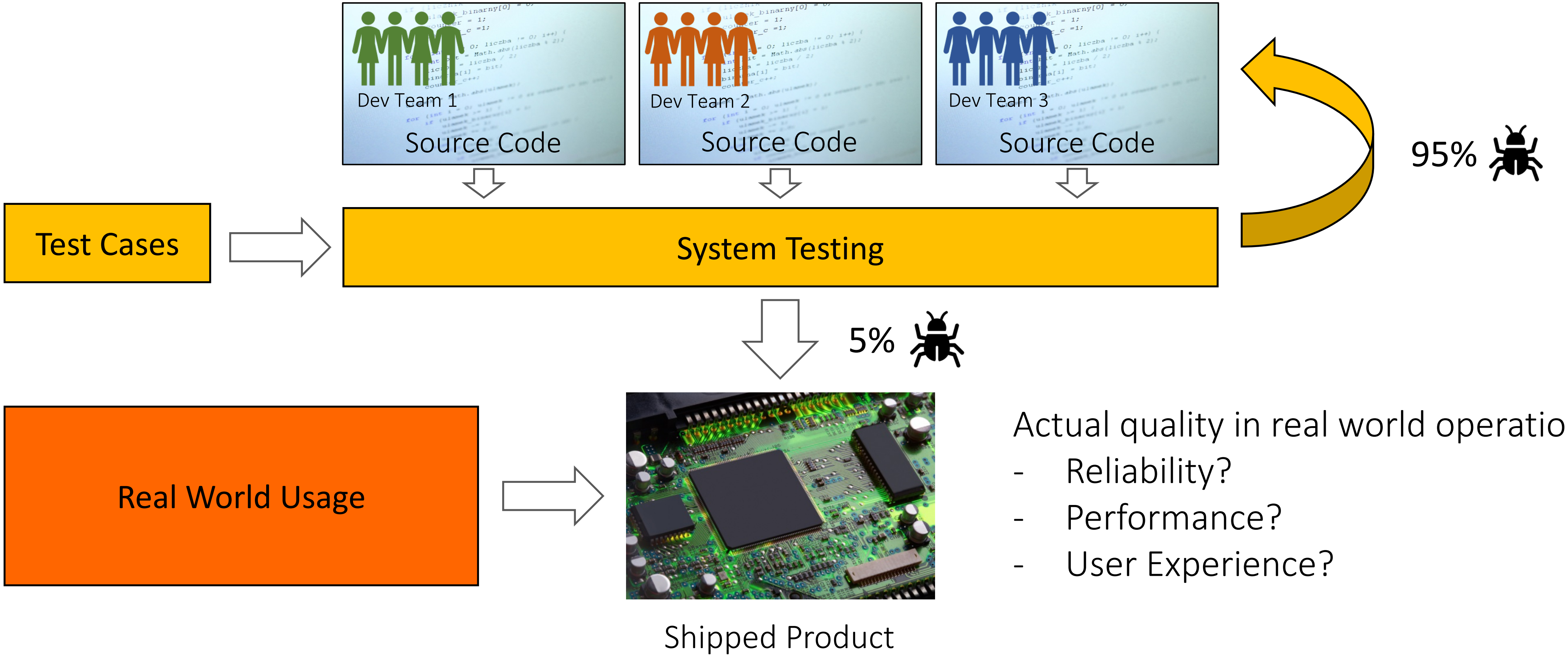
- Defect removal rate: 95%. Remaining 5% end up in shipped products
- Defect introduction rate: 50-100 bugs per 1000 lines of code (KLOC)
- 20% of missed bugs classified as “major defects”

For a 50 KLOC product, 25 - 50 major defects will likely hit the customers

Typically elusive bugs, related to specific circumstances

*Research report summary by Jack Gannsle, https://www.embedded.com/latent-defects/

CHALLENGES OF SYSTEM TESTING



Dev Team 1

Source Code

Dev Team 2

Source Code

Dev Team 3

Source Code

Test Cases

System Testing

95%

5%

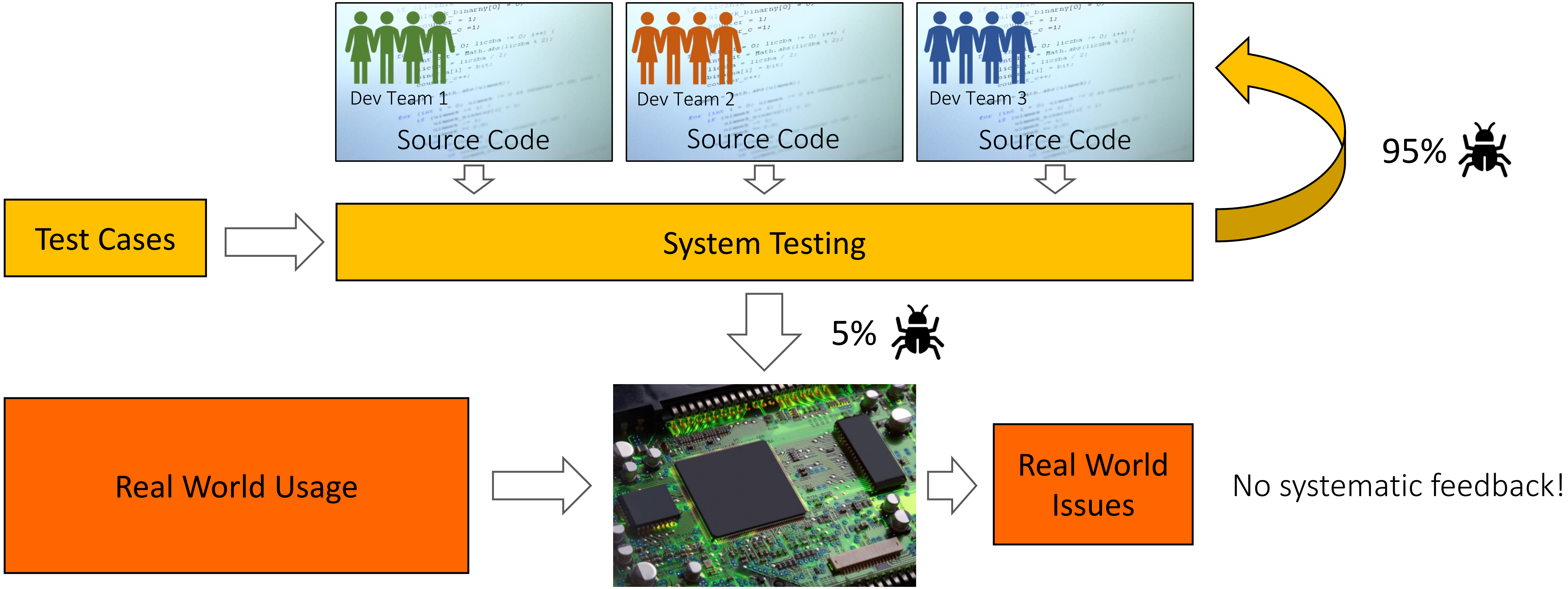
Real World Usage

Shipped Product

Actual quality in real world operation?

- Reliability?
- Performance?
- User Experience?

HOW DEAL WITH REAL-WORLD ISSUES?



Dev Team 1

Source Code

Dev Team 2

Source Code

Dev Team 3

Source Code

Test Cases

System Testing

95% 🐞

5% 🐞

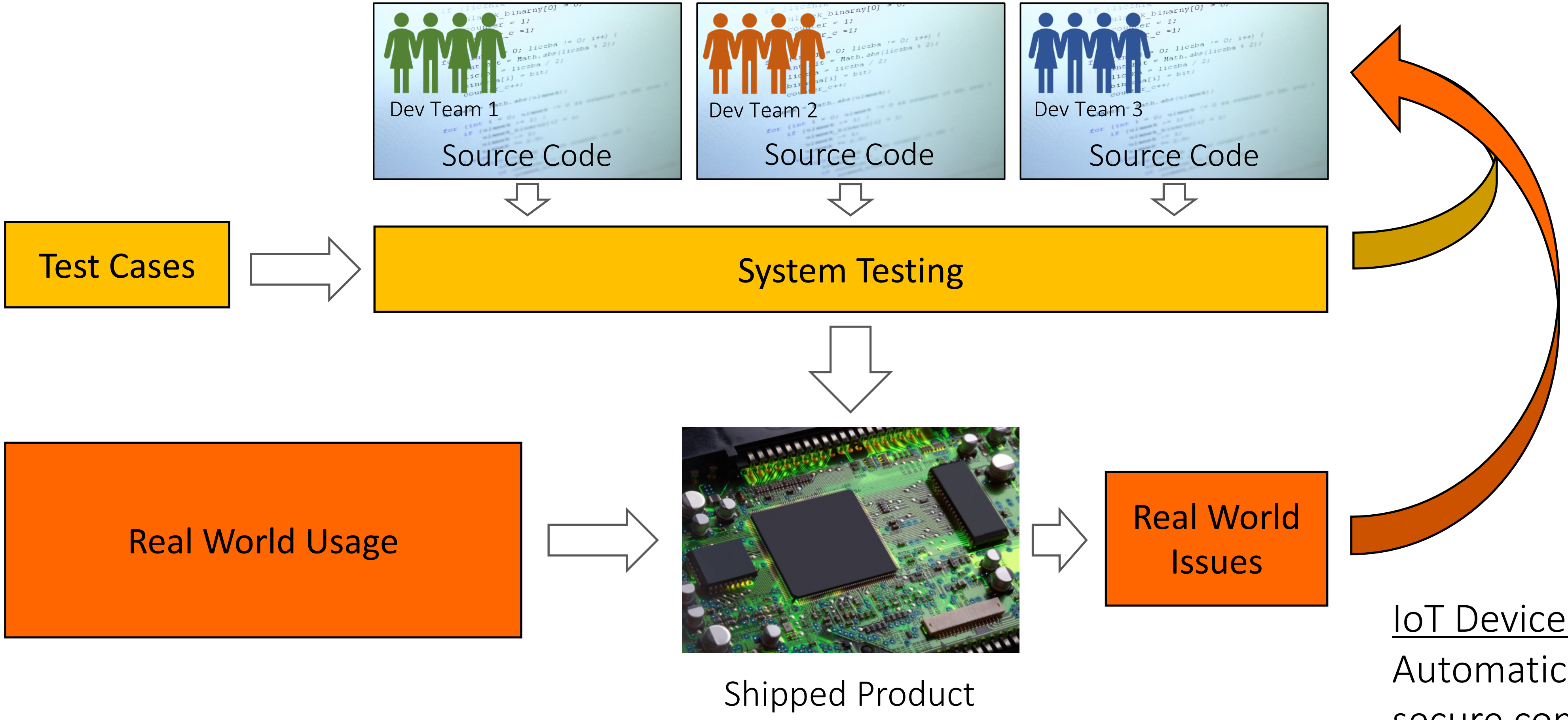
Real World Usage

Shipped Product

Real World Issues

No systematic feedback!

CLOSE THE LOOP - IOT DEVICE MONITORING



Dev Team 1

Source Code

Dev Team 2

Source Code

Dev Team 3

Source Code

Test Cases

System Testing

Real World Usage

Shipped Product

Real World Issues

<u>IoT Device Monitoring</u>
Automatic alerts via
secure connectivity

IOT DEVICE MONITORING - BENEFITS

<u>Improve Product Sales</u>
Catch remaining bugs ASAP and improve product reviews, customer satisfaction and brand loyalty.

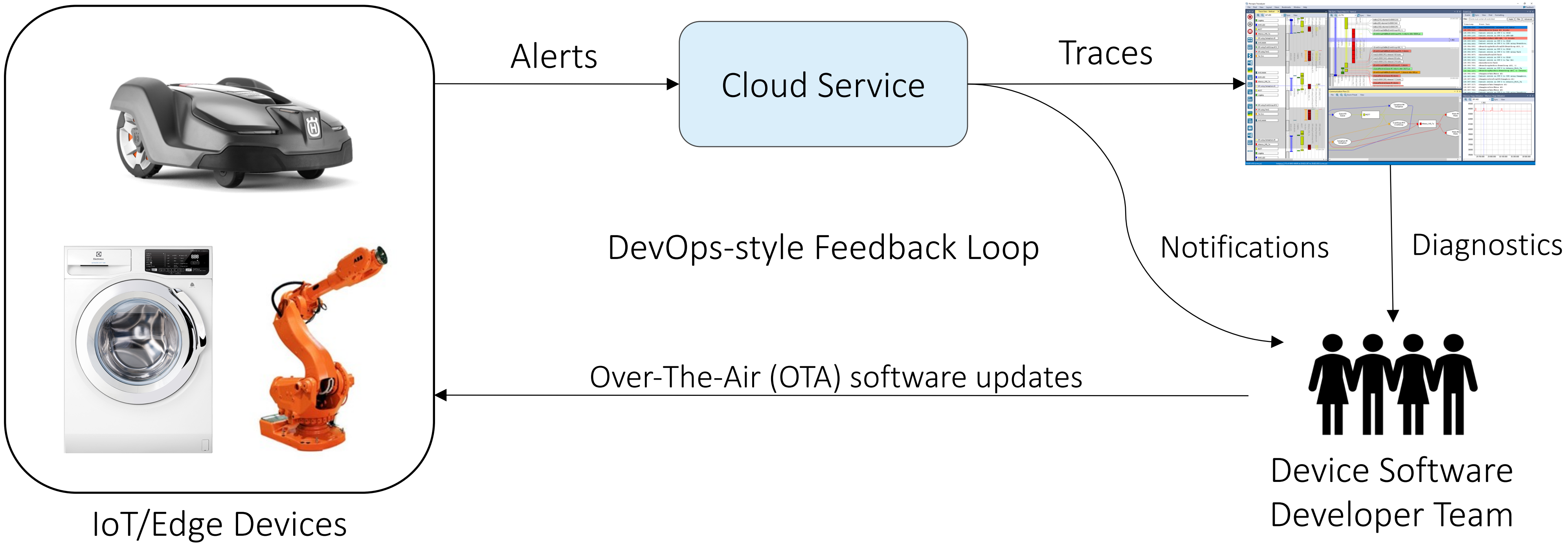
<u>Reduce Risks and Costs</u>
Improve reliability to reduce liability risks as well as customer support and return costs.

Minimize debugging costs for customer issues caused by missed bugs.

<u>Beat the Competition</u>
Get continuous feedback on product performance from real-world usage and discover important improvements.

Product performance?
Battery life?
User experience?

A CLOUD-CONNECTED “FLIGHT RECORDER”



Alerts

Cloud Service

Traces

DevOps-style Feedback Loop

Notifications

Diagnostics

Over-The-Air (OTA) software updates

IoT/Edge Devices

Device Software Developer Team

Know about real-world issues within seconds

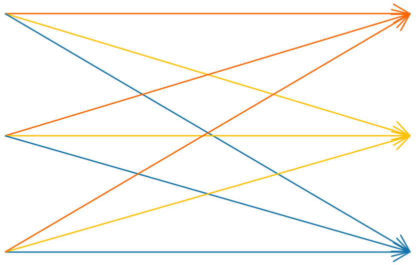
Speed up resolution with detailed diagnostics

Take full advantage of OTA updates for continuous software improvement

IOT DEVICE MONITORING - USE CASES

Phases

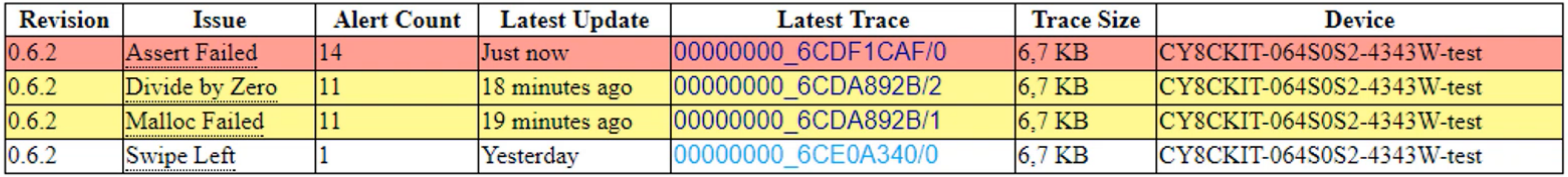
- System test
- Field testing
- Deployment



Concerns

- Errors, due to software bugs or hardware issues
- Warnings, e.g. ”stack usage exceeded 95%”
- Product performance
  - User experience, e.g. startup time
  - Feature usage
  - Battery life
  - Algorithm performance

ISSUES, ALERTS AND SYMPTOMS



Revision	Issue	Alert Count	Latest Update	Latest Trace	Trace Size	Device
0.6.2	Assert Failed	14	Just now	00000000_6CDF1CAF/0	6,7 KB	CY8CKIT-064S0S2-4343W-test
0.6.2	Divide by Zero	11	18 minutes ago	00000000_6CDA892B/2	6,7 KB	CY8CKIT-064S0S2-4343W-test
0.6.2	Malloc Failed	11	19 minutes ago	00000000_6CDA892B/1	6,7 KB	CY8CKIT-064S0S2-4343W-test
0.6.2	Swipe Left	1	Yesterday	00000000_6CE0A340/0	6,7 KB	CY8CKIT-064S0S2-4343W-test

Issue: A unique combination of Alert type and Symptoms

Alert: An individual report, with a numeric type code (e.g. 42: Malloc Failed)

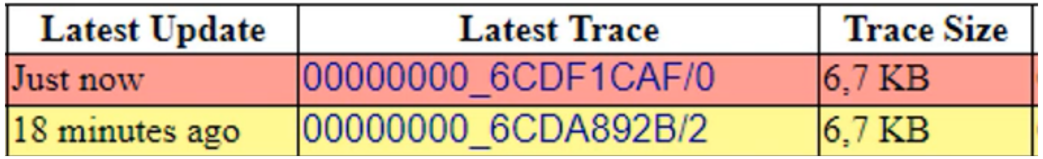
- Software errors, due to e.g. failed sanity checks, error handlers or hardware fault handlers.
- Proactive warnings (e.g. “memory usage at 90%”)
- Periodic reporting of device metrics (e.g. startup time, features used)

Symptom: Additional diagnostic information (optional and configurable)

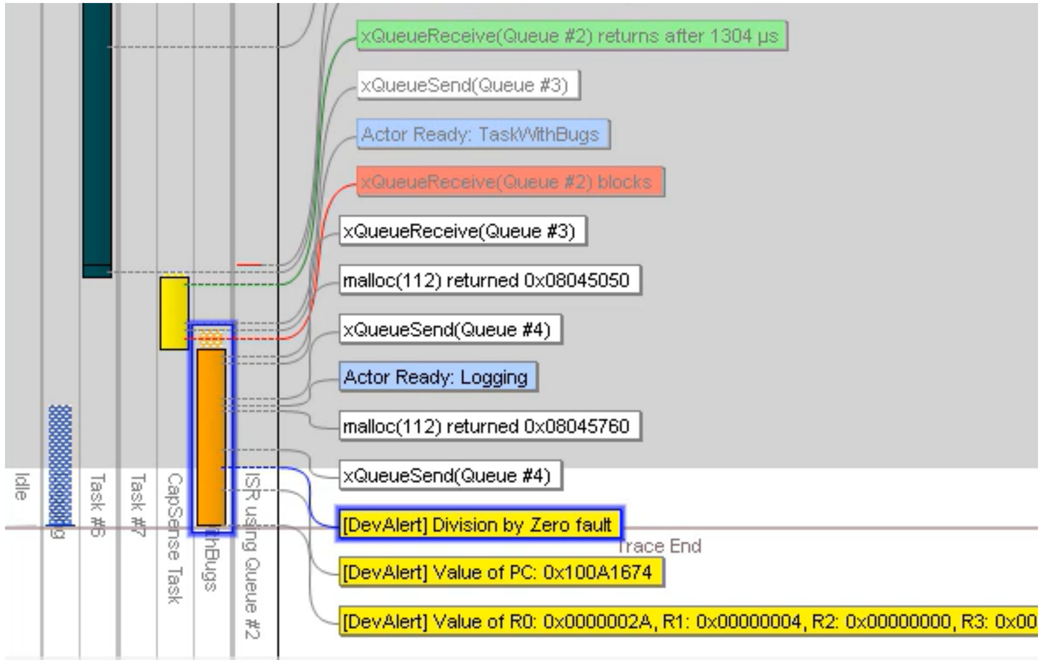
- Program counter, stack pointer
- Selected registers and variables
- Function parameters

COMBINE WITH VISUAL TRACE DIAGNOSTICS

- Alerts include software traces
- See what happened just before the alert
  - Recent software execution, API calls, etc.
  - Recent inputs (e.g. sensor values)
  - Software logging messages
- Provides context – important for debugging

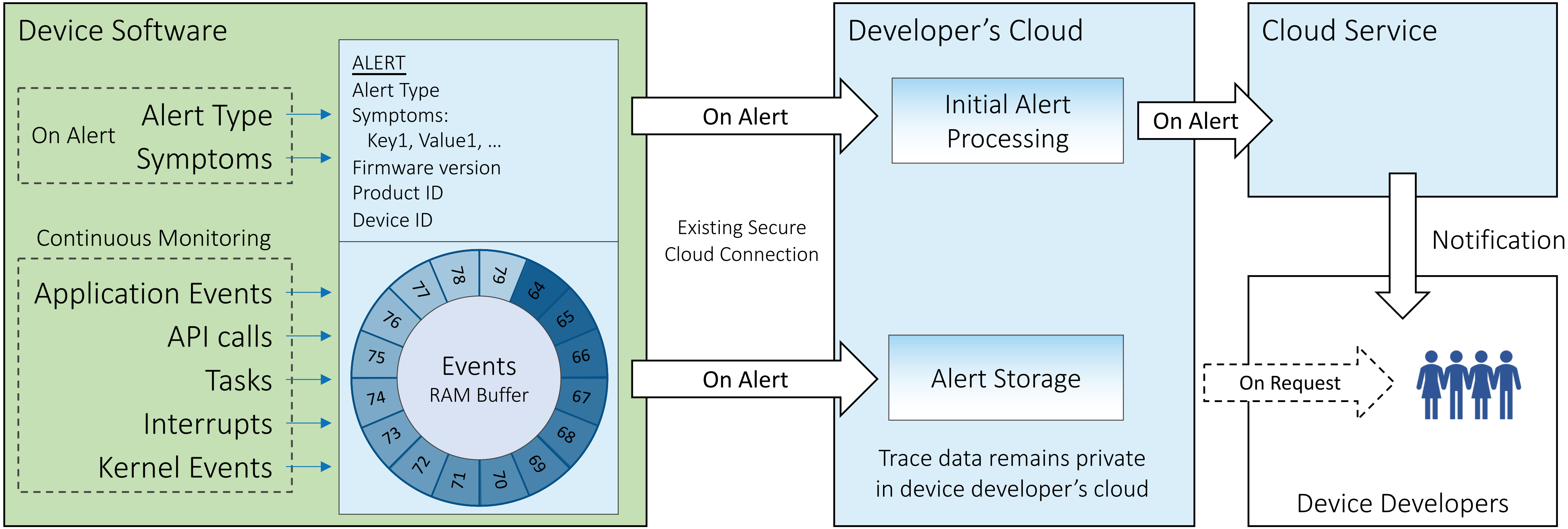


Latest Update	Latest Trace	Trace Size
Just now	00000000_6CDF1CAF/0	6,7 KB
18 minutes ago	00000000_6CDA892B/2	6,7 KB

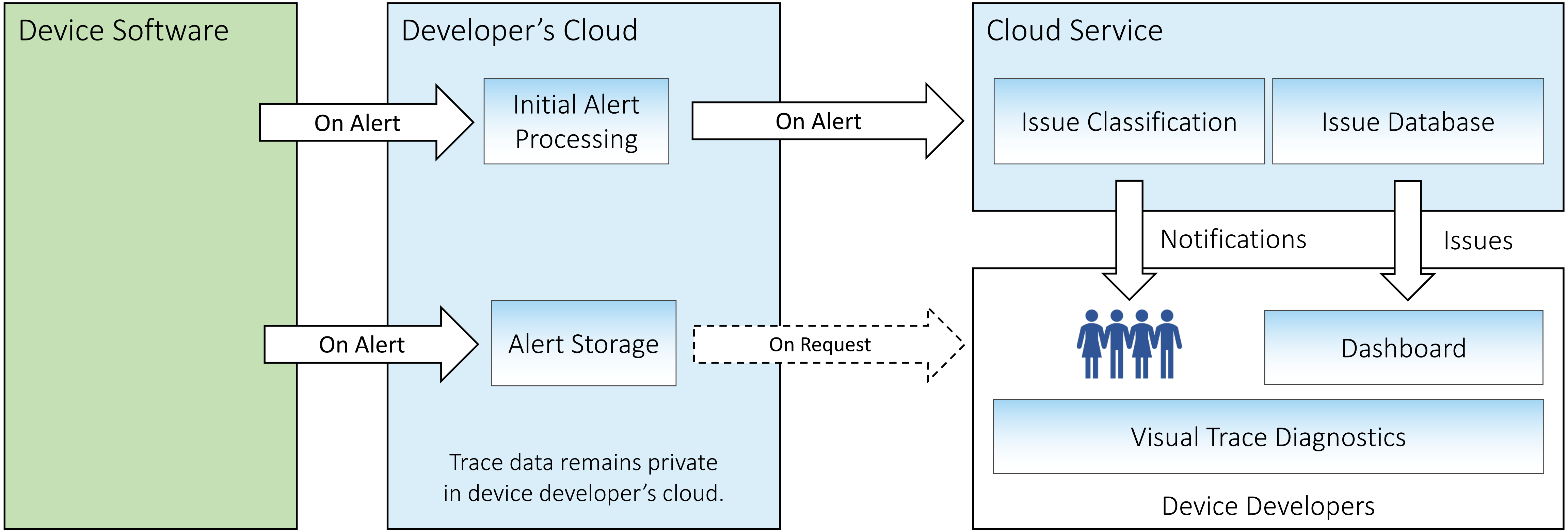


xQueueReceive(Queue #2) returns after 1304 µs
xQueueSend(Queue #3)
Actor Ready: TaskWithBugs
xQueueReceive(Queue #2) blocks
xQueueReceive(Queue #3)
malloc(112) returned 0x08045050
xQueueSend(Queue #4)
Actor Ready: Logging
malloc(112) returned 0x08045760
xQueueSend(Queue #4)
[DevAlert] Division by Zero fault
[DevAlert] Value of PC: 0x100A1674
[DevAlert] Value of R0: 0x0000002A, R1: 0x00000004, R2: 0x00000000, R3: 0x00

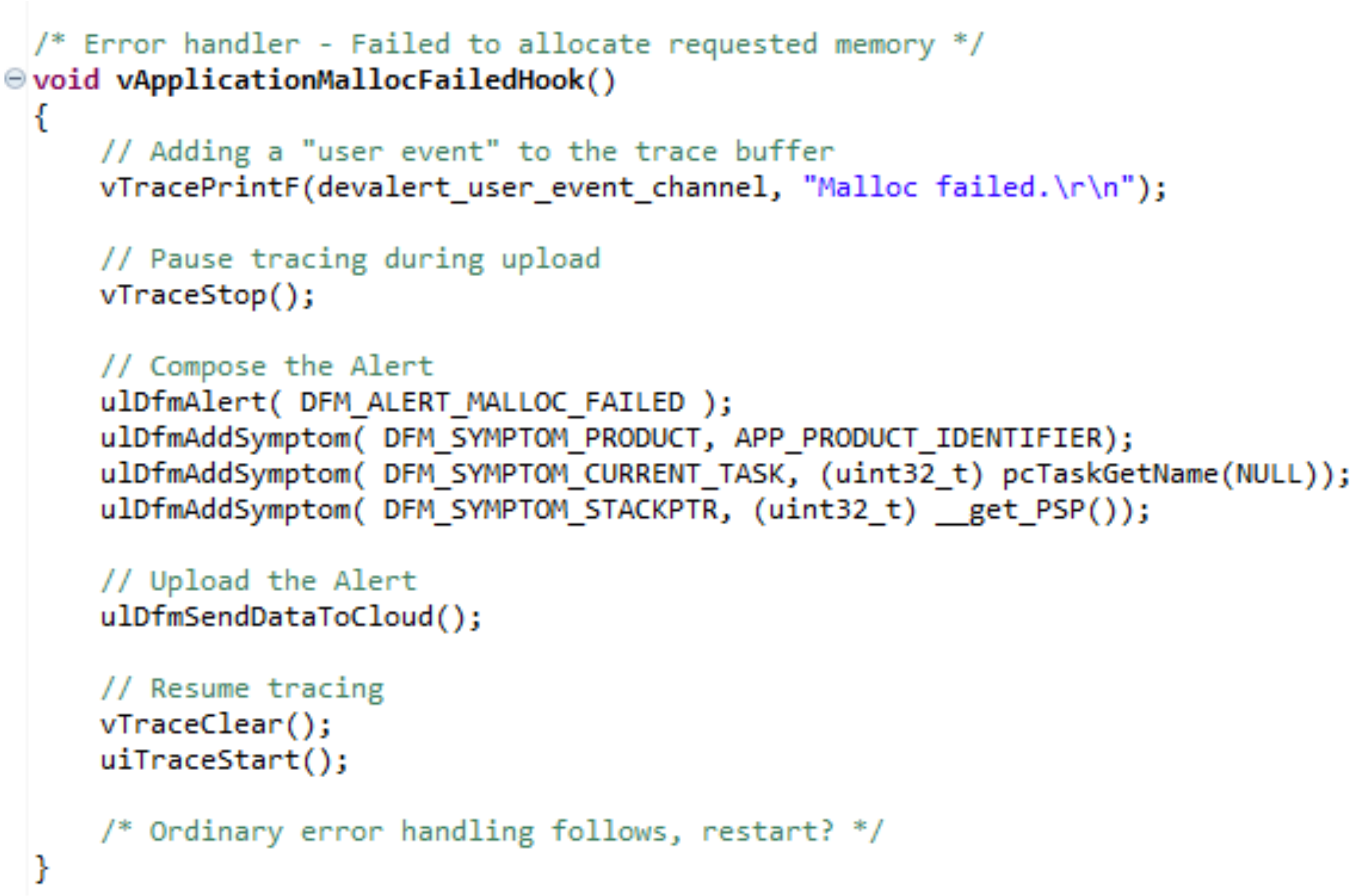
DATA FLOW – FROM DEVICE TO DEVELOPER



DATA FLOW – FROM DEVICE TO DEVELOPER



GENERATING ALERTS



```
/* Error handler - Failed to allocate requested memory */
void vApplicationMallocFailedHook()
{
    // Adding a "user event" to the trace buffer
    vTracePrintF(devalert_user_event_channel, "Malloc failed.\r\n");

    // Pause tracing during upload
    vTraceStop();

    // Compose the Alert
    ulDfmAlert( DFM_ALERT_MALLOC_FAILED );
    ulDfmAddSymptom( DFM_SYMPTOM_PRODUCT, APP_PRODUCT_IDENTIFIER);
    ulDfmAddSymptom( DFM_SYMPTOM_CURRENT_TASK, (uint32_t) pcTaskGetName(NULL));
    ulDfmAddSymptom( DFM_SYMPTOM_STACKPTR, (uint32_t) __get_PSP());

    // Upload the Alert
    ulDfmSendDataToCloud();

    // Resume tracing
    vTraceClear();
    uiTraceStart();

    /* Ordinary error handling follows, restart? */
}
```

DEMO - PERCEPIO DEVALERT

Cypress PSoC 64 MCU (Arm Cortex M4)

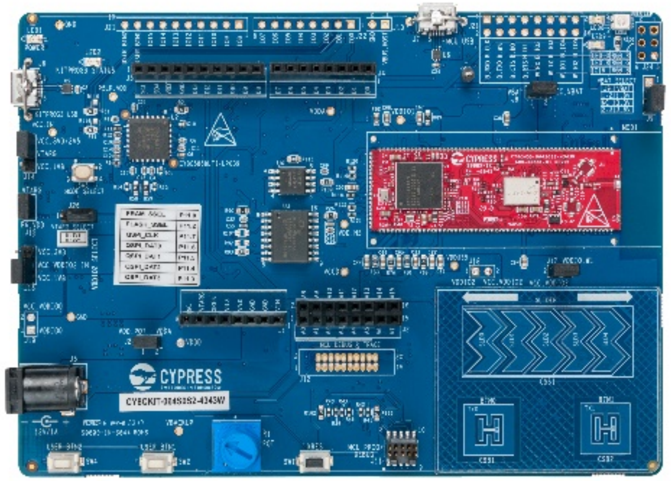
FreeRTOS

MQTT over Wi-Fi to AWS IoT Core

Serial console via USB debug cable

DevAlert Firmware Monitor (DFM)

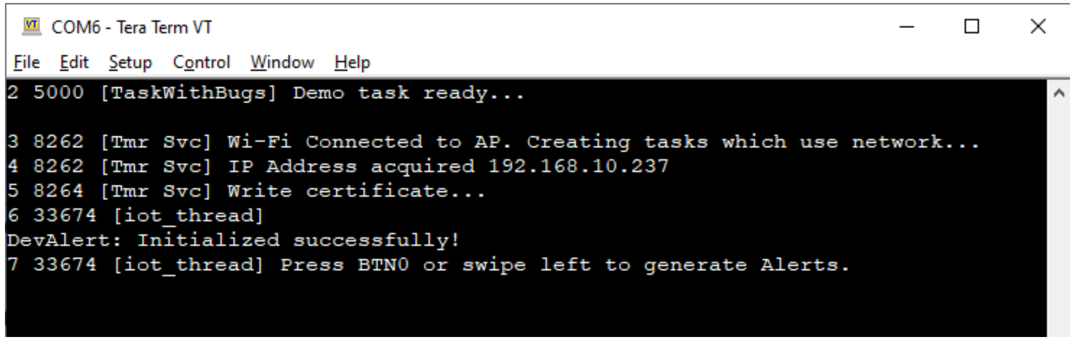
Buttons trigger real errors, causing alerts



aws qualified device FreeRTOS

Wi-Fi® Ready

devAlert™



```
2 5000 [TaskWithBugs] Demo task ready...

3 8262 [Tmr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
4 8262 [Tmr Svc] IP Address acquired 192.168.10.237
5 8264 [Tmr Svc] Write certificate...
6 33674 [iot_thread]
DevAlert: Initialized successfully!
7 33674 [iot_thread] Press BTN0 or swipe left to generate Alerts.
```

Demo project available at github.com/percepio

SECURITY

#1: No new attack surfaces - just as secure as the existing application

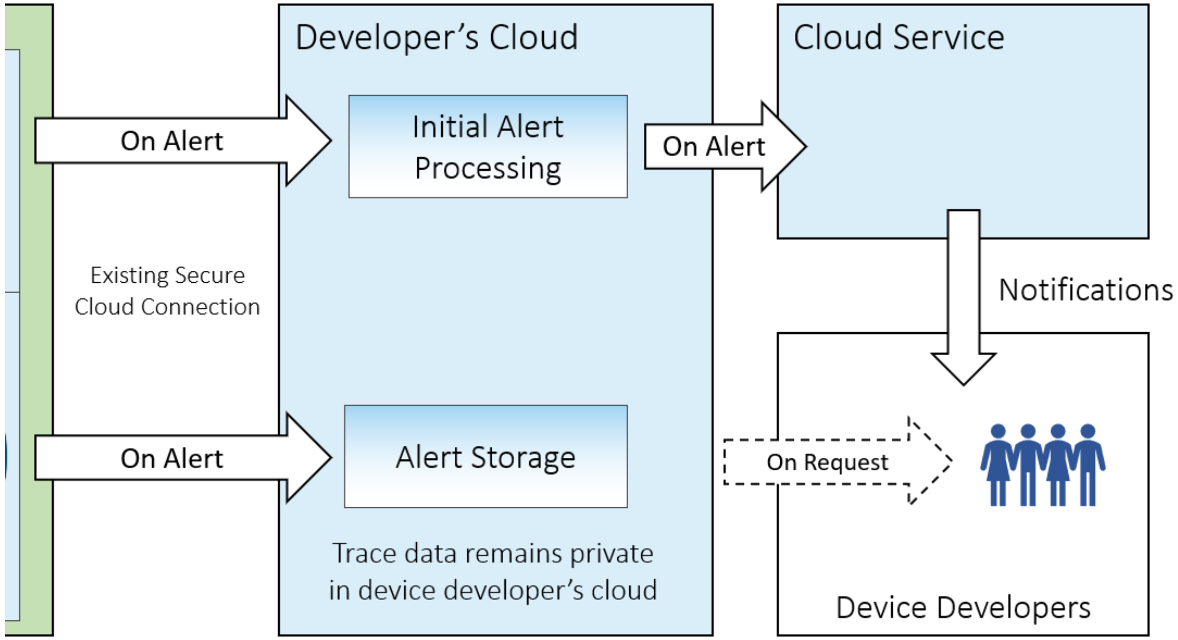
- Alerts are uploaded in the same way as other application data, directly to the cloud account of the device. The trace data remains there.

- The device agent doesn’t listen for incoming data

#2: Cloud processing is divided into two separate domains

#3: The alert signature is mostly numeric key-value pairs, without standardized meaning

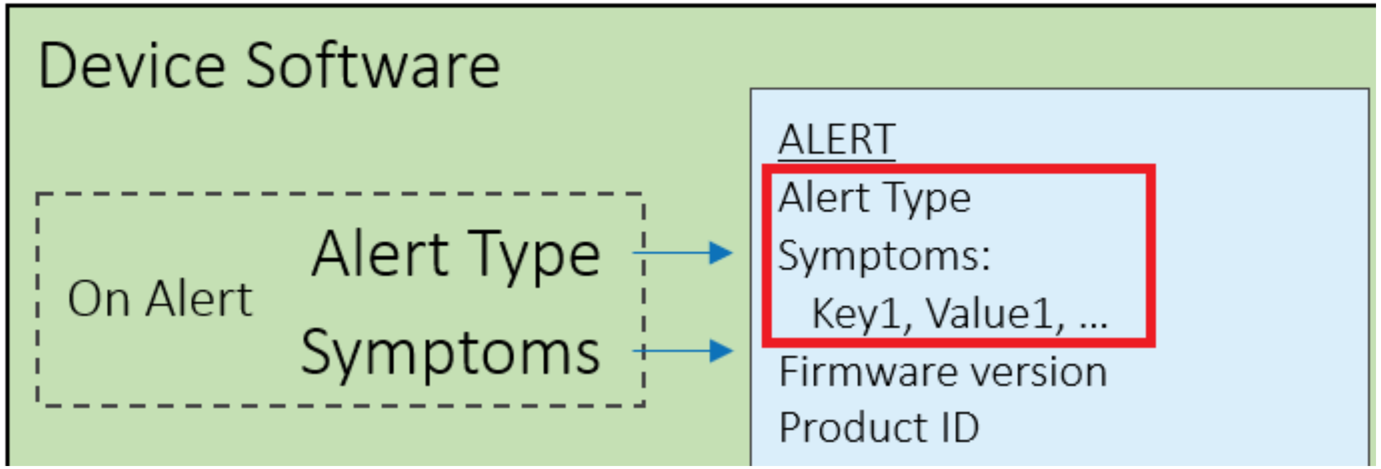
SECURITY



#2: Cloud processing is divided into two separate domains

- Developer's cloud account: Receives the full alerts directly from the device and stores them
- Cloud service: Receives only the alert signatures, indirectly

SECURITY



#3: The alert signature is mostly numeric key-value pairs, without standardized meaning

- Defined in the cloud service, individually per product and alert
- Meaningless on its own

PRIVACY, GDPR

This is about monitoring DEVICES, not people. GDPR only applies to personal data.

The software tracing allows for logging anything in the device software, in theory also personal data. But…

- Requires explicit logging by the device developer
- Not provided to the central cloud service - only available to the device developers
- No different from traditional application logs…

Don’t log any information that can be connected to a person!

- For the “device ID”, use an anonymous but unique code (e.g. a GUID)
- Avoid logging personal data, including the IP address of the device

OPERATIONAL COSTS

Large fleet with many alerts - 1 million devices, 1 alert per week (5 KB)

- 52 000 000 alerts / year
- 261 GB / year

Cost example, using AWS

Alert upload:

- No AWS charge from data transfer when using AWS Basic Ingest
- Triggers a script for each alert (52 million times)
- Cost: $218/year

Alert storage (S3):

- Cost: $300/year, assuming all 261 GB is kept

Total cost: $518/year, plus OTA update costs

OTA update cost, 1 000 000 devices using AWS IoT Core

- Cost: $1,875 per update - But what is the alternative cost of letting bugs remain unfixed?

RUNTIME OVERHEAD

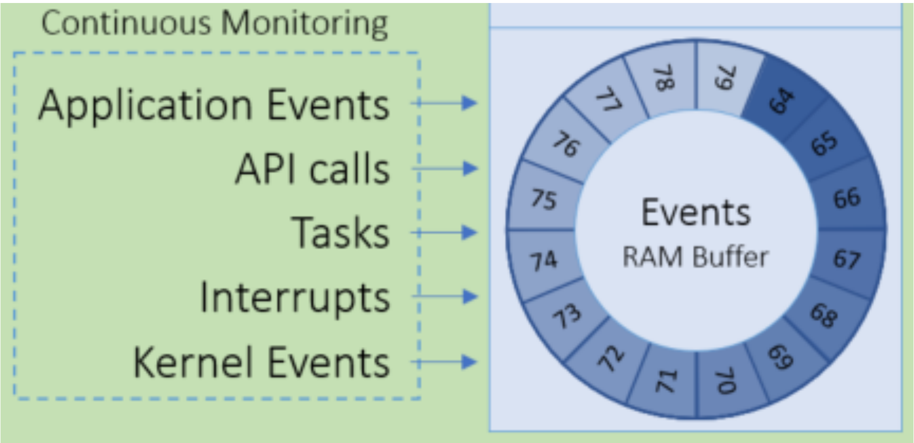
RAM usage: Typically 1-10 KB for trace buffer (configurable)

Flash usage: 10-20 KB needed for trace recorder library

CPU usage:

- Microseconds per event
- Depends on application (event rate, amount of logging)
- Some penalty, but typically not noticeable (a few percent)

No “probe effect” since always on



SUMMARY

Enables a DevOps-style feedback loop from devices to developers

Take full advantage of OTA updates to improve product quality and performance

Visual trace diagnostics simplifies analysis and debugging

Just as secure as the IoT application platform

Some operational costs, but much smaller than the alternative costs

Some runtime overhead, but no “probe effect” since always on

LEARNING MORE

https://percepio.com/devalert

info@percepio.com

THANK YOU

IoT
Online
Conference

www.iotonlineconference.com

IoT
Online
Conference

www.iotonlineconference.com