ECEN 5053-002

Developing the Industrial Internet of Things

Week 14 - Lecture

Debugging Deeply Embedded Systems

Dave Sluiter - Spring 2018

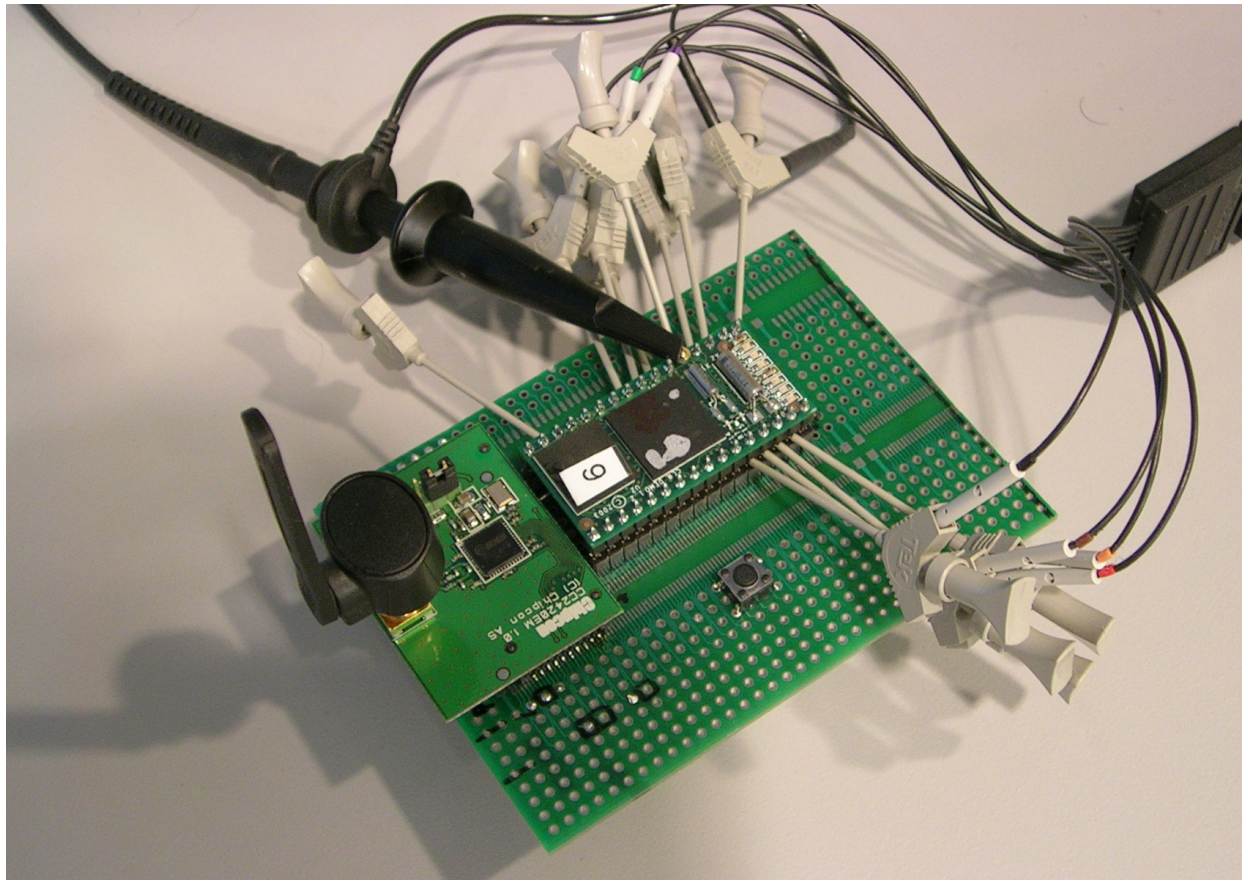
Material

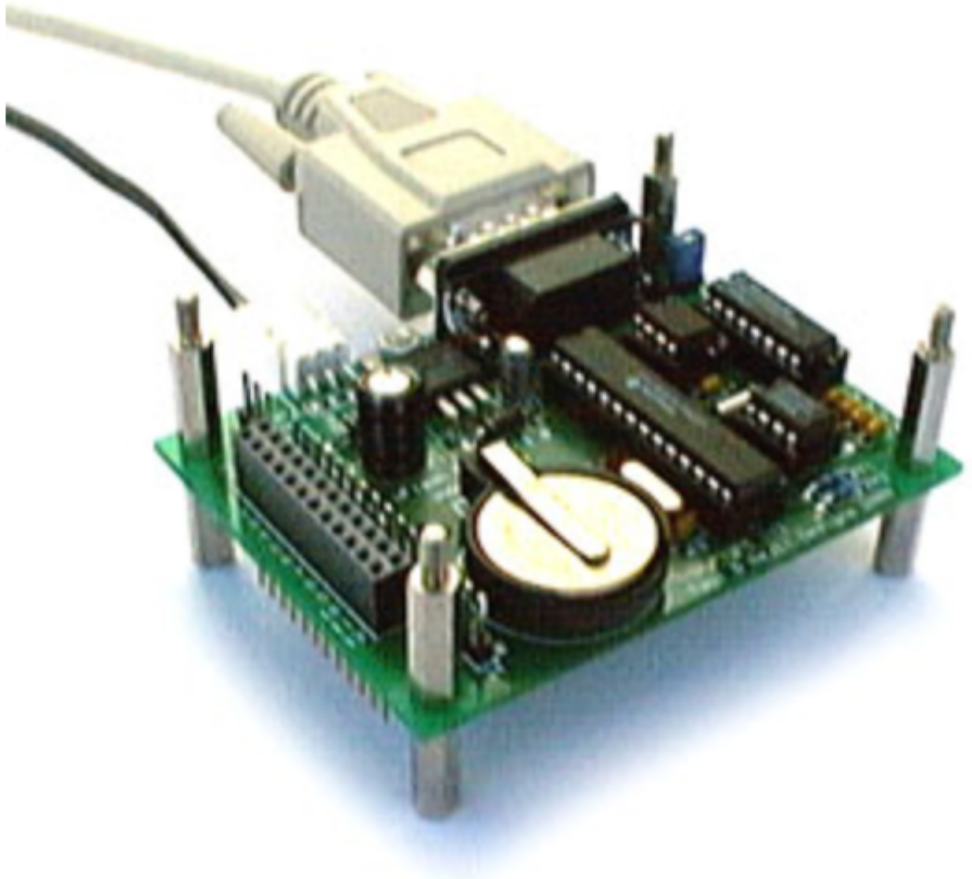
- Debugging deeply embedded systems

Learning Outcomes

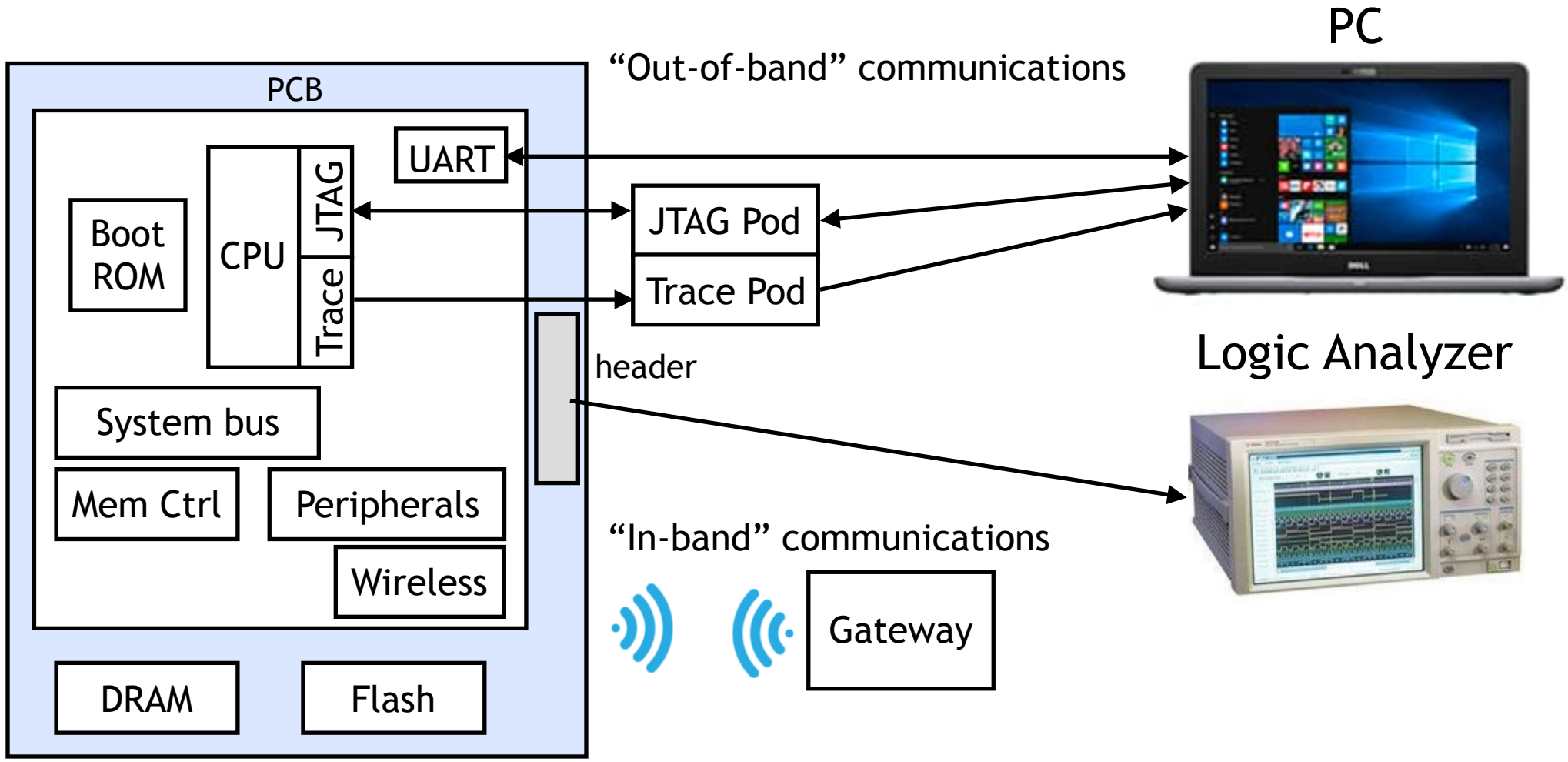
- Students will learn about techniques for debugging deeply embedded systems

Typical Debug Environment





Easy Access in the Lab/Desk



What happens if

- Your system is embedded in another system?
- And that system is embedded in another system?
- And that system is in:
  - Another city?
  - Another country?
  - Not even on the planet?
- Uh oh…

Relax

- Typically your customer and your customer's customer will specify a set of requirements for remote monitoring and access to an embedded system.
- These requirements define the use cases that need to be supported
  - Designed-in from the beginning

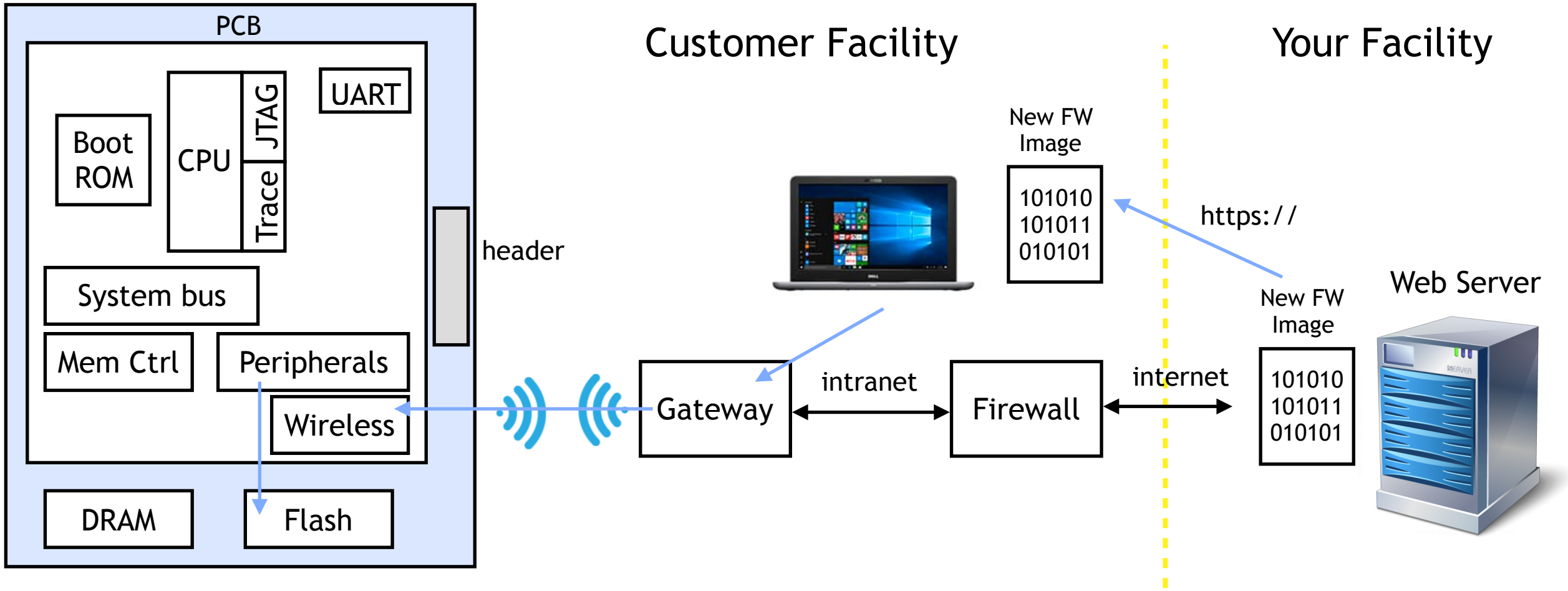
Requirement Types

- Software/firmware updates
  - Push for secure updates, sometimes customers do not understand the threats. Educate your customers.
- In-service monitoring
  - Realtime metrics monitoring: performance, anomalous behavior, error conditions
- Out-of-service monitoring
  - RMA (Returned Material Authorization, i.e. a returned product), customer wants root-cause analysis performed

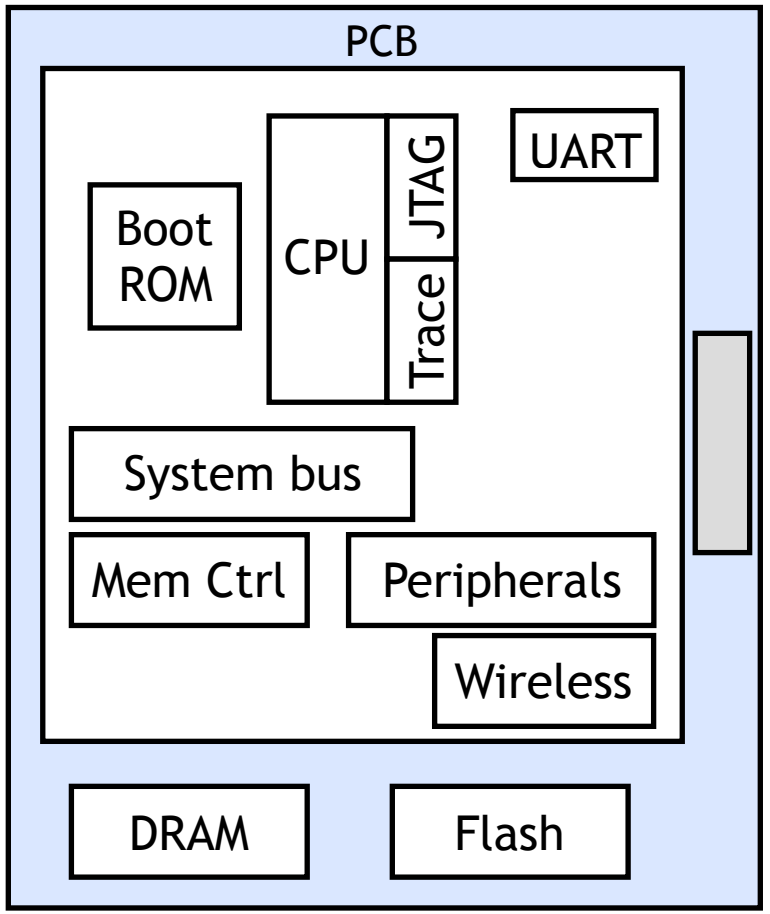
In-service Access Types

- Direct, real-time to you
  - Communication link provided by a customer's firewall
- Indirect
  - Debug ports brought out to final product's interface
    - A trained customer service technician can run tests and collect data
  - Diagnostic reports created by your system, extracted by your customer and emailed/uploaded to you

Firmware Updates in the Field



Direct Access in the Field

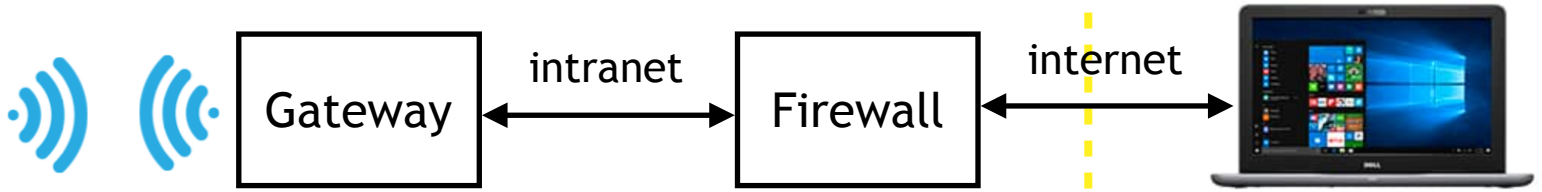


header

Customer Facility

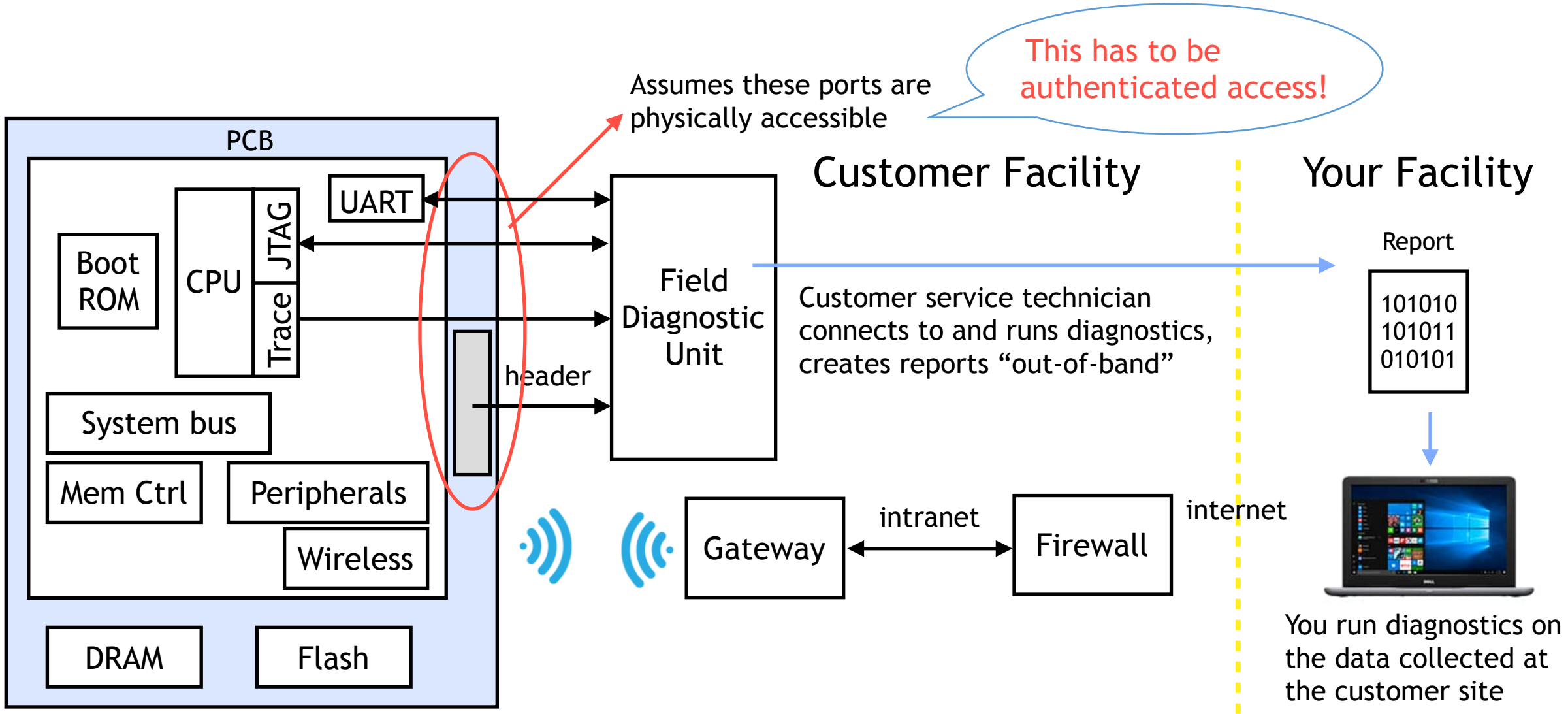
Your Facility

End customer may open up a certain port number and protocol. Hopefully this would be TLS using a CA

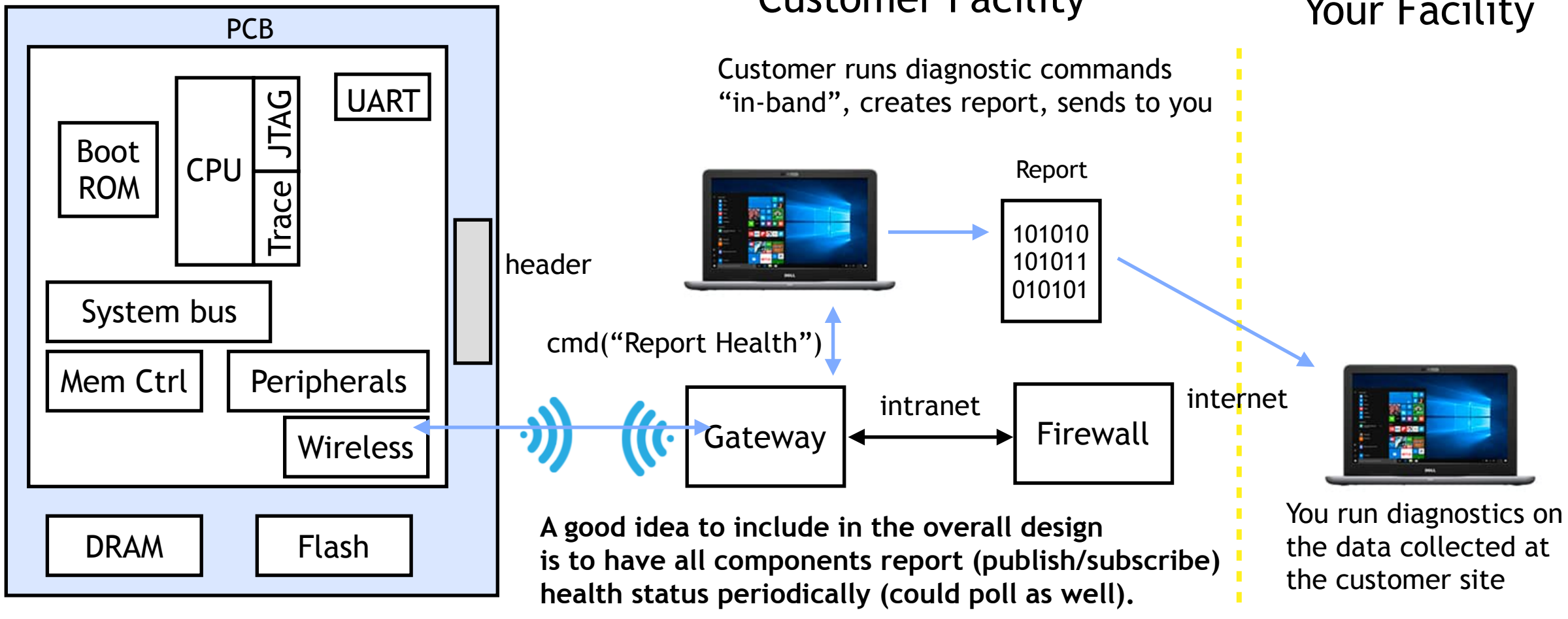


You run remote diagnostics, periodically or continuously. **Black & Veatch** does this for their utility companies.

Indirect Access in the Field



Indirect Access in the Field



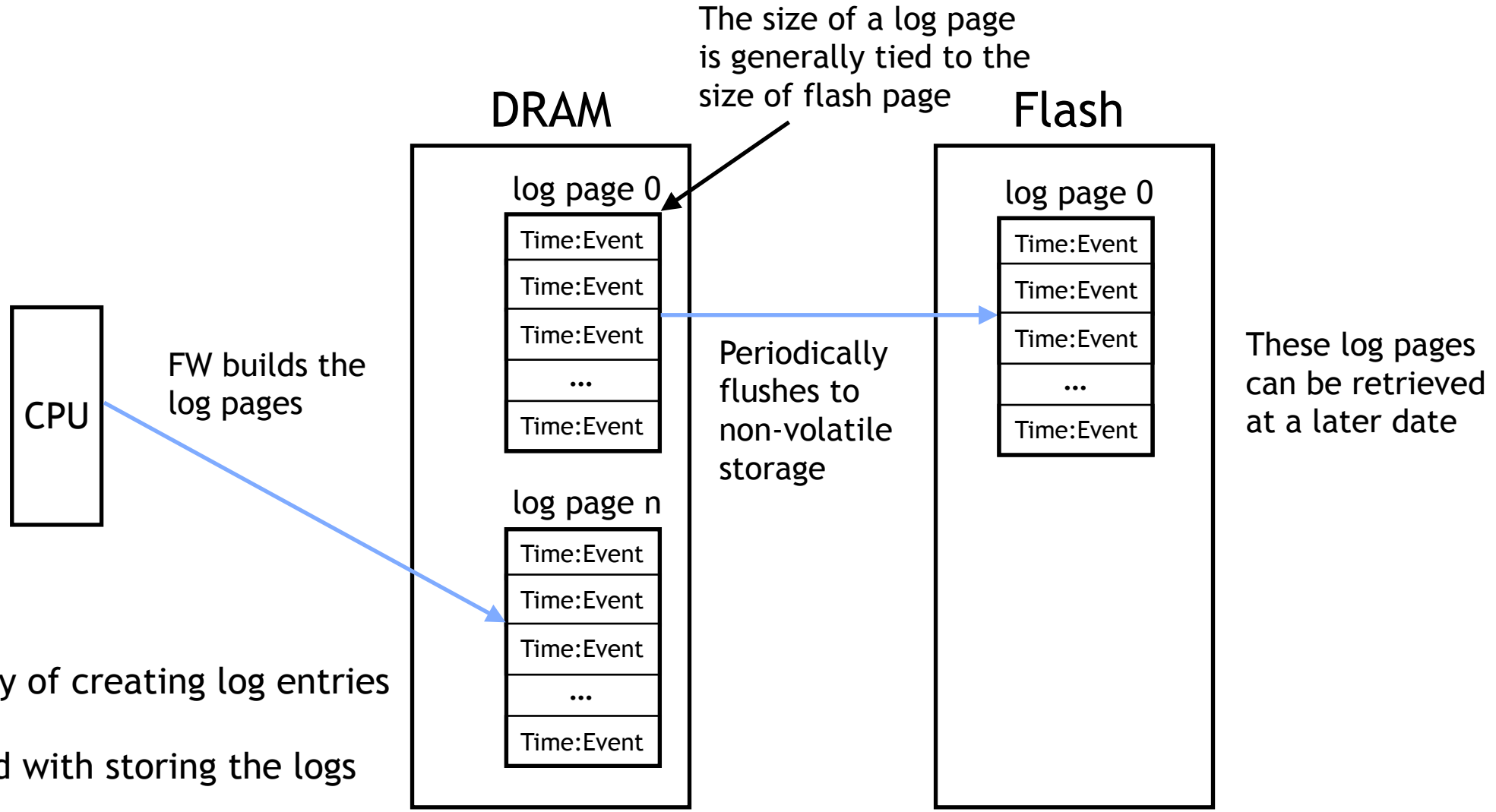
Out-of-service Monitoring

- This really isn't "monitoring" per se
- A unit has failed or has become unresponsive
  - Sometimes a unit can recover by power cycling or resetting
- Customer returns the unit (RMA) and wants to know the root cause of the failure
- Shipping a new unit to the customer is easy
  - But costs money
- How do you determine the root cause of the failure?
- One technique that can help are **log pages**

What's a log page?

- A log page is a **list of critical events** that have occurred over time
- What are critical events?
  - Illegal command received
  - Parity/ECC errors on data busses and SRAM/DRAM
  - Temperature excursions
  - Invalid FW imaged loaded
  - Communication errors / retry / timeouts
  - Illegal instruction executed
  - Depending on your system/product, there may be many more critical events
- If possible include a timestamp with each event
- Due to space/cost constraints, it is typically not practical to store every event. So a tradeoff has to be made. Again, this goes back to customer requirements.

Constructing a Log Page



The primary tradeoff is:

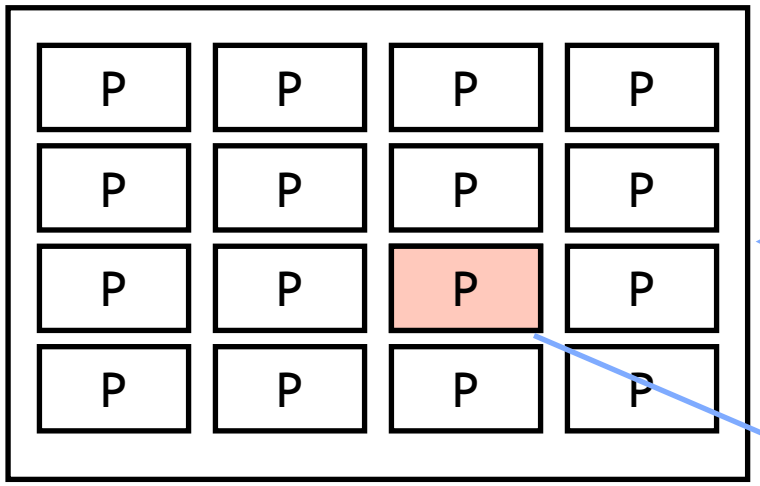
* Granularity/frequency of creating log entries
  vs.
* Space/cost associated with storing the logs

Root Cause

- If you've done a "good job" designing the log page system it "should" capture the cause of the failure
- Then you report that reason back to your customer
- Be aware, some failures are very difficult to root-cause
  - Some are never determined
- A step beyond the notions outlined here can sometimes proceed to disassembly of individual components. Looking for cracked PCB traces or solder joints, metal migration inside of a chip etc.

Revisiting Product Validation

Rack of your products being tested.
RDT - Reliability Demonstration Testing



Testing System

A unit reports a failure.
You check the log pages.
The log page doesn't explain the failure:
the PC fetched an illegal instruction -or- the code that is executing doesn't make any sense.
How did the PC get to this memory location?

Revisiting Product Validation

- You have a couple choices if you're restricting your analysis to in-form-factor products
  - Creating a new FW image that has **asserts** coded into it
    - Asserts are additional error checking code that would not normally be part of a product distribution
    - Downside is that the presence of the asserts changes the timing of execution. You may not uncover how the PC got to where it was
  - Revert back to the "Easy" fully instrumented system with the production FW that failed
    - Debugger and Logic Analyzer
- Re-run the validation tests. Hopefully you will figure out how the code got to where it was

The Importance of Trace

- Next class we will see a demo of TRACE32 from Lauterbach

End