Securing BareMetal Hardware at Scale

Paul McMillan and Matt King

Intro

- Paul and Matt are responsible for platform and infrastructure security at Oracle Cloud Infrastructure

- No discussion of specific products, features, or vulnerabilities

This talk isn’t about

- Virtual Machines
- Runtime Security
- Laptops/Desktops (we assume a secured datacenter)
- TCG
- Defending against hardware implants or trojans
- Malicious vendors

BareMetal Servers

- *No Hypervisors or VMs*
- Customers run their own kernel
  - Allows low-level access to hardware devices
  - And firmware interfaces

Problem: Customers and Firmware

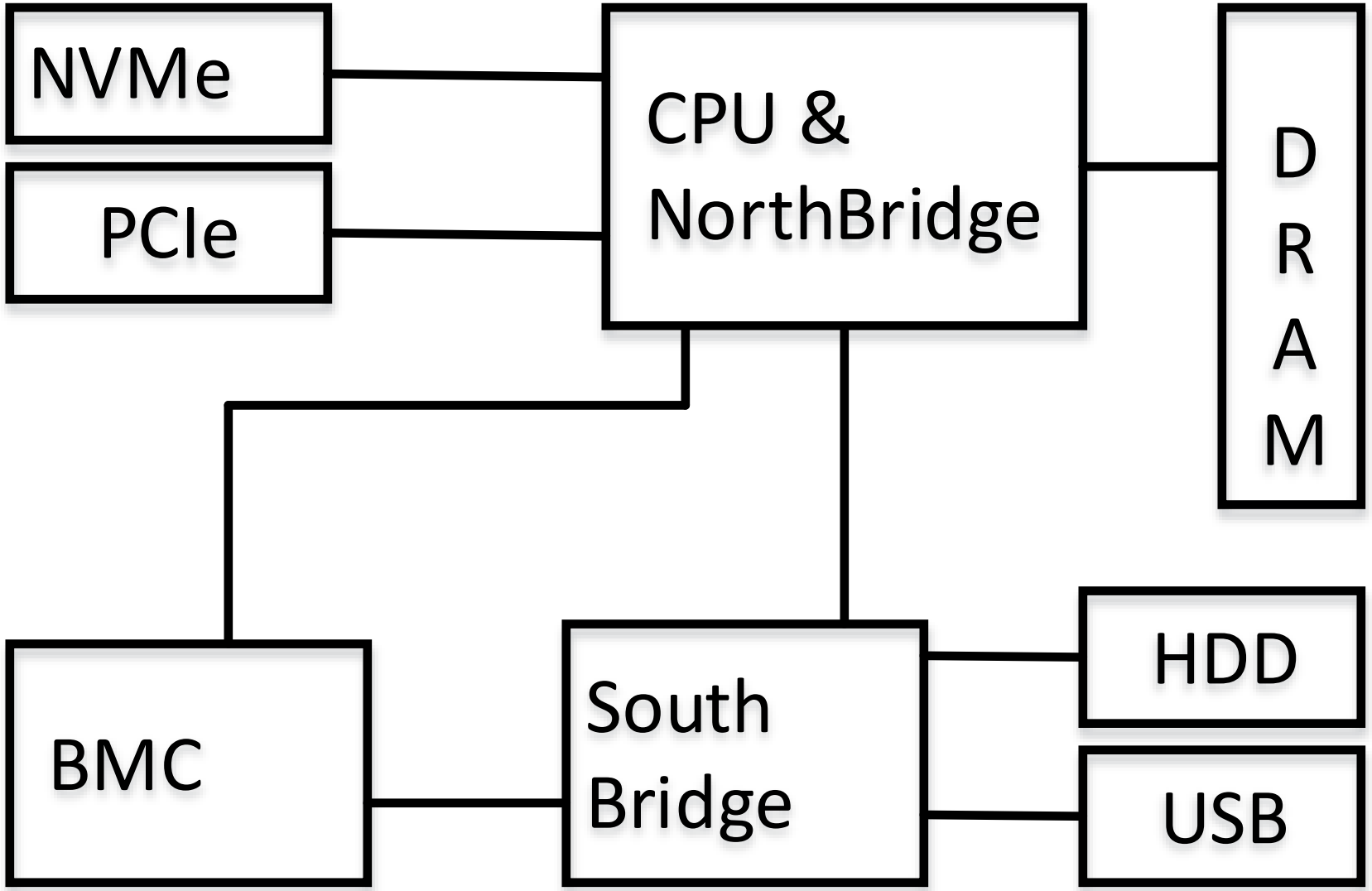
- Systems have a wide range of mutable code in non-volatile storage
- Customers can run firmware update utilities
- This can lead to:
  - Inconsistent versions across the fleet
  - Installation of known buggy firmware
  - Malicious firmware

Goal: Give Customers “Pristine” systems

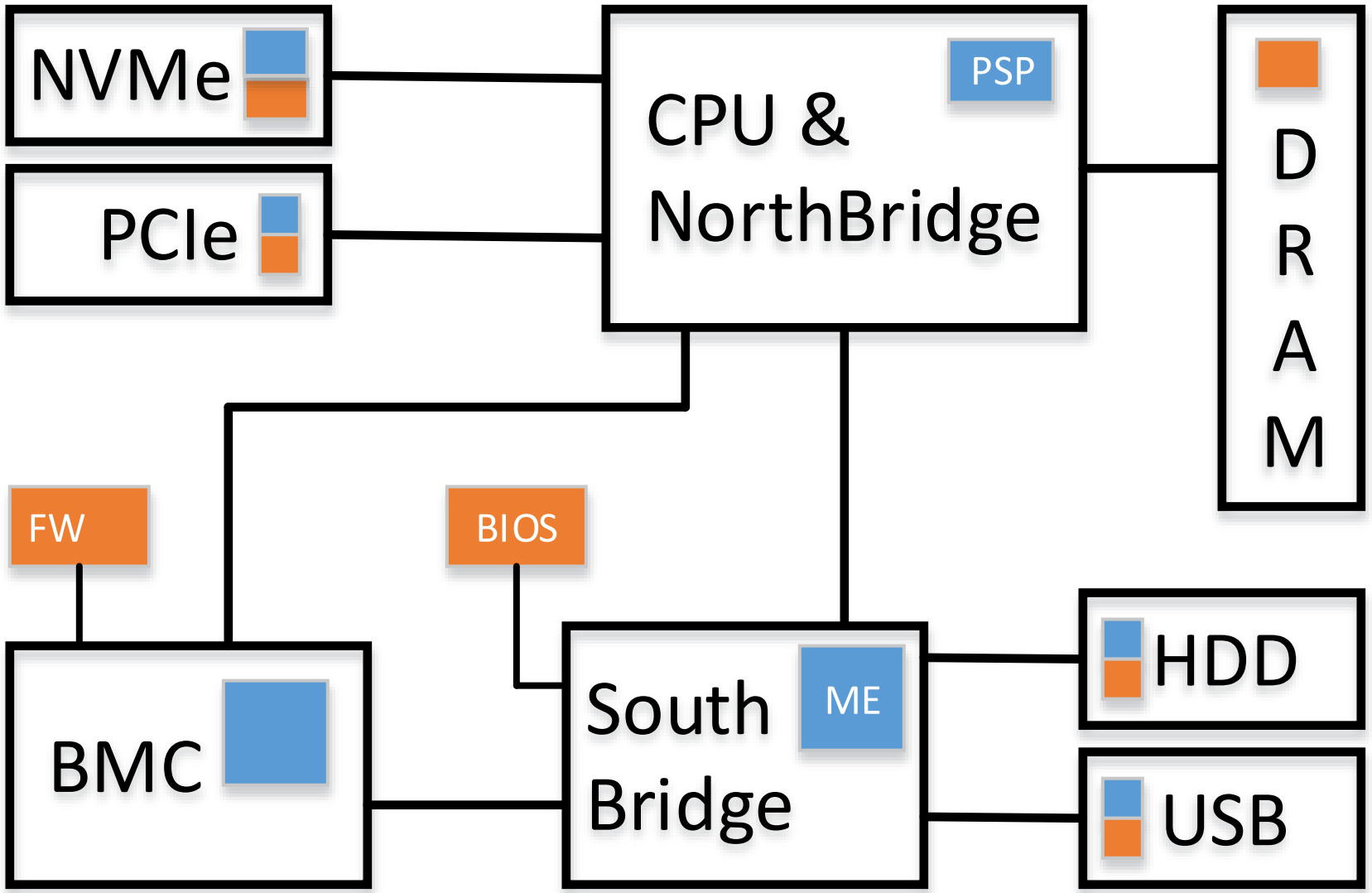
- No dependencies on previous (mis)use
- Automatable remanufacturing process
- Security in the face of bugs

Background

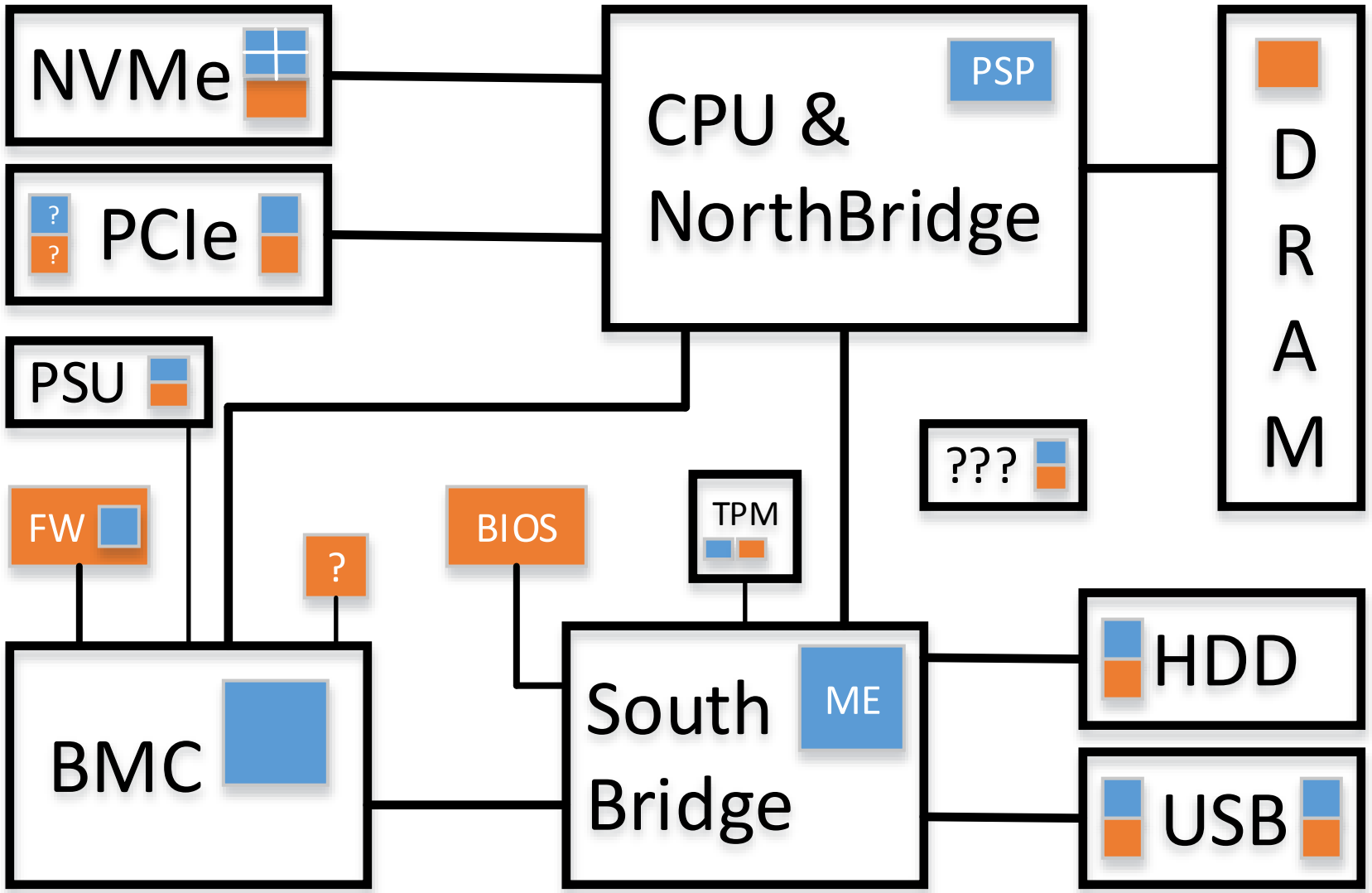
Server Platform



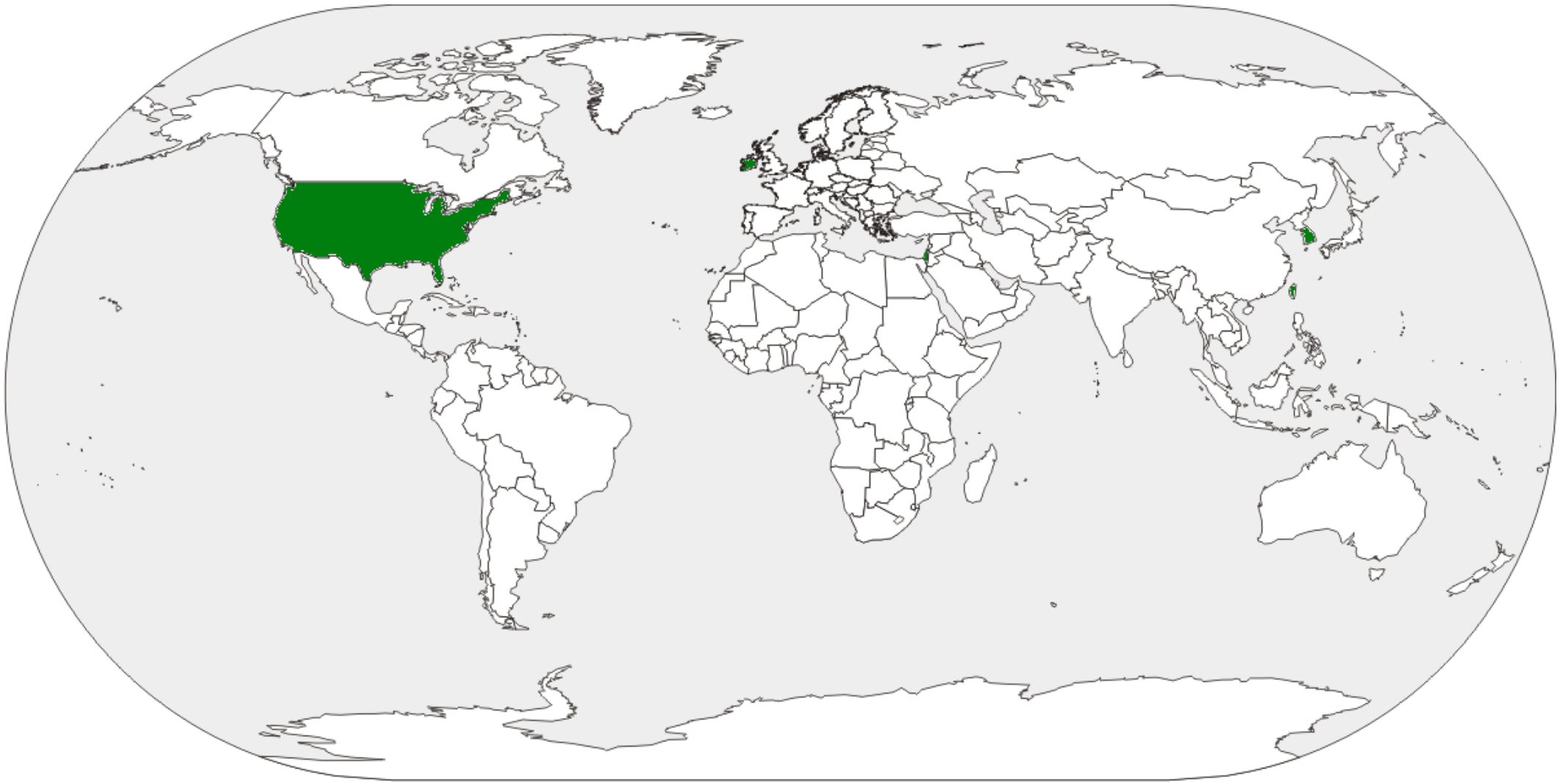
Server Platform



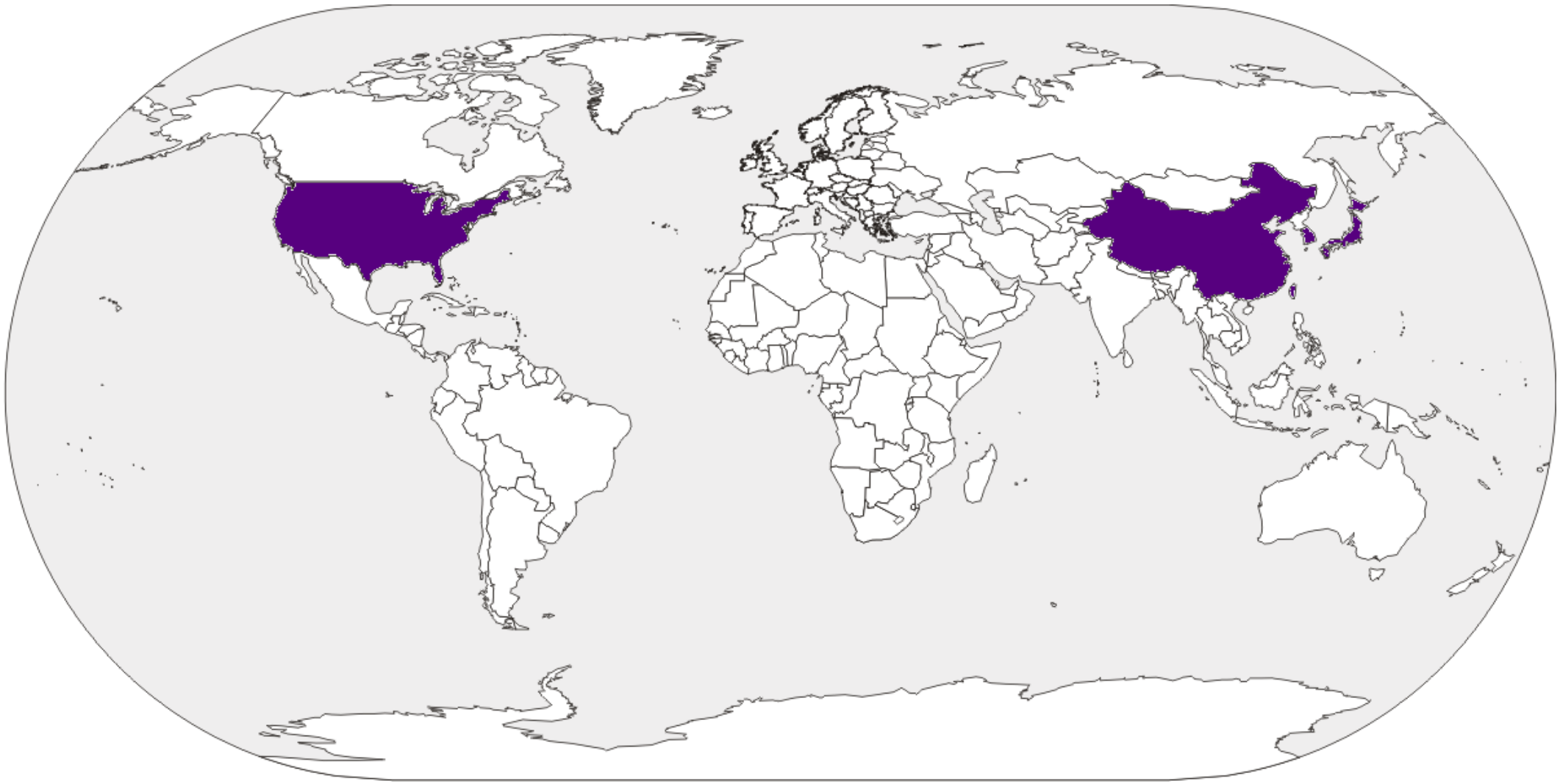
Server Platform



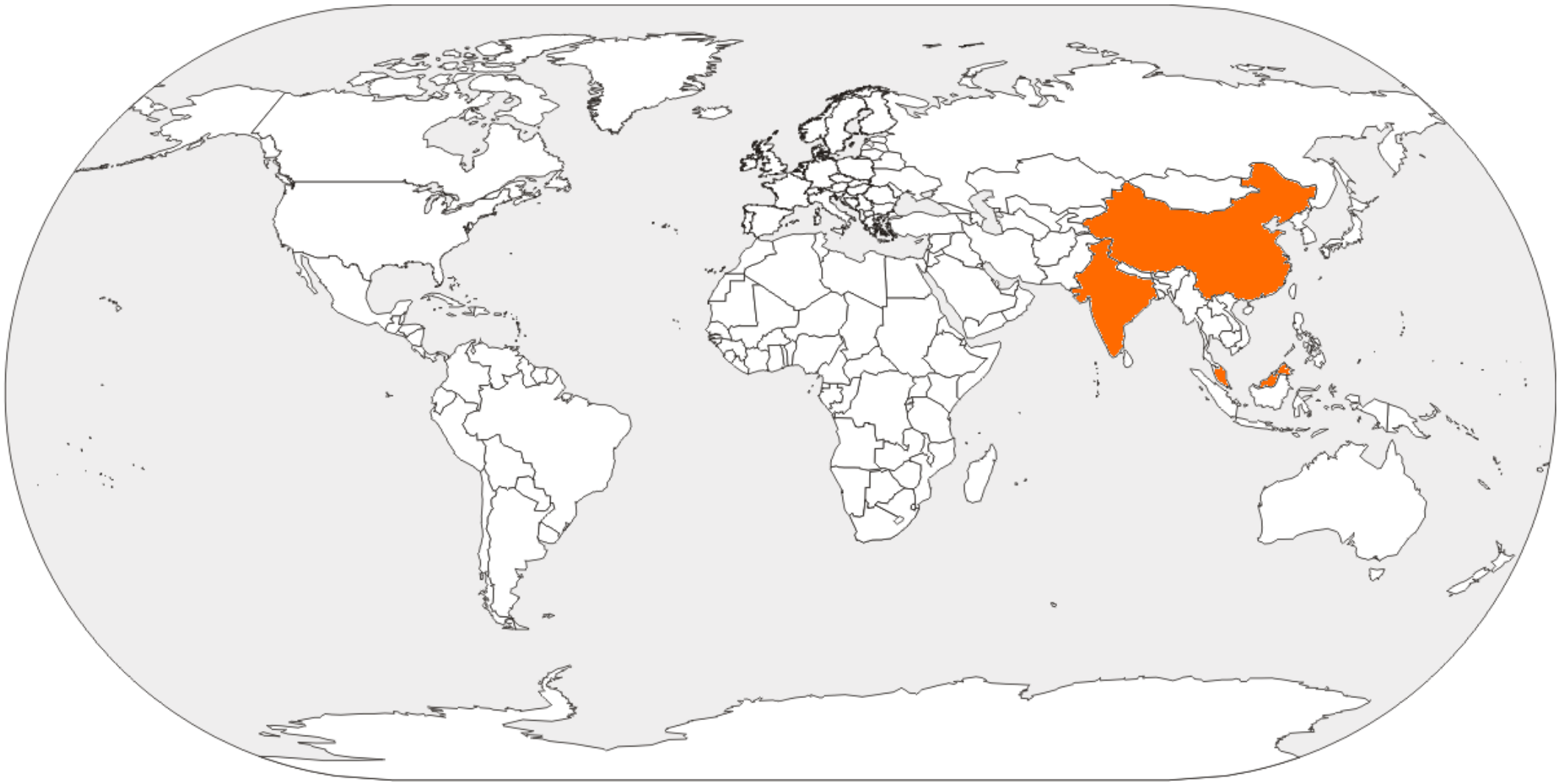
Supply Chain – Component Manufacturing



Supply Chain – PCB Manufacturing



Supply Chain – Device Assembly



Supply Chain – Destinations

Supply Chain

- Hard to ensure systems arrive with desired Firmware
- Devices sometimes arrive non-functional
  - Bugs
  - Failures
  - Bitrot
  - etc.

- *How do we verify firmware in the Datacenter?*

Current Solutions

Signed Firmware

- Vendor signature over device firmware
  - Blocks unintended code from running
- Ensures firmware matches device
- Widely implemented
- Recommended by NIST to protect firmware

Signing Limitations

- Signatures are not an indication of quality
- Updates are validated by running firmware
- No provisions for remediation
- No indication if running old (vulnerable) version
- Must wait for vendors to generate and sign patched versions

Secure Boot

- Extends signature checking to device boot
  - ROM checks boot loader, which checks kernel, etc.
- Prevents runtime bugs from gaining persistence
- Common among restricted-use devices
  - Examples: game consoles, carrier-locked phones

Secure Boot Limitations

- Doesn’t solve inherent issues with code signing
- Unable to sign configuration data
- No revocation mechanism

Measurement

- Device reports its own status
  - Frequently signed by a private key to prove authenticity
- Proves exactly what code was loaded
- Examples: Google Titan, TPMs

Measurement Limitations

- Few devices support secure measurement
- Measurements often unstable
  - Firmware updates
  - Configuration changes
  - Device Identifiers: serial numbers, MAC address

Our Challenge

- Signing is insufficient
- Most devices do not offer measurement
- *So what now?*

Recovery

Hardware Engineering Challenges

- Firmware developers need to update firmware when:
  - There is no firmware
  - The update routines don’t work (or exist)
  - Firmware hangs
  - Signature checking is broken
  - Hardware features don’t work

Hardware Engineering Solutions

- Nearly all hardware has a recovery mechanism
- Does not depend on current firmware
  - Built in to ROM
  - JTAG
  - Serial Port
  - Other proprietary methods

- *Can we use these for security too?*

Recovering Firmware for Assurance

- Apply updates to mutable firmware without executing unknown code

- Use hardware mechanisms that operate regardless of current device state
- Bypass or halt execution of updatable code

Example Recovery Process

- Enter recovery mode
  - Device waits for new firmware instead of booting
- Supply known boot loader to device
- Reboot device
  - Now it’s running the known boot loader
- Do normal firmware load to finish updating

Operationalizing Recovery

- Custom-built hardware device to drive recovery interfaces
- Connectivity to devices and management interface

- NO RUN-TIME UPDATEABLE FIRMWARE!

Custom Hardware

Show and Tell

Limitations

- Not general purpose, customized for each device
  - We have a large, homogenous fleet
- Requires vendor support
  - Needs schematics, letter of volatility, tooling, etc…
- May overwrite logged data
  - No diagnostic info for failure analysis
- Cabling sucks

Limitations (cont)

- Reverse engineering is expensive and slow
- Often requires full device reboots
  - Challenging on multi-tenant systems
- Impacts component selection
- Write cycle limits on non-volatile storage
- If you can do it, so can attackers!
  - Secure boot helps

Future Work

In-band Recovery

- Recover without the extra hardware
- Recover devices without host reboot
  - Useful for PCI pass-through to guest VM
- More reliable
- Easier to standardize

Detection

- Better device-level Identification/Attestation
- Firmware update tooling doesn’t report previous state of device
- Standard methods to chain device firmware into trusted boot

Runtime Integrity

- LOL!
- Requires substantial changes in firmware and hardware development
- If you have ideas, we'd love to hear them.

Conclusions

- Gaining assurance of running firmware is hard but possible
- Common requests are more likely to become reality
  - Pester your vendors (this really works!)
- We are moving towards a world where device firmware can be verified

Q&A

- Paul McMillan
  - @PaulM
  - Paul@McMillan.ws
- Matt King
  - @syncsrc
  - jtag@syncsrc.org