Jailbreaking an Electric Vehicle in 2023

WHAT IT MEANS TO HOTWIRE TESLA'S X86-BASED SEAT HEATER

Christian Werling
Niclas Kühnapfel
Hans Niklas Jacob
TU Berlin

Oleg Drokin
Independent




Tesla’s Infotainment Now AMD-Powered

TESMANIAN

Model S Model 3 Model X Model Y Mach-E Gift Card SALE News

Europe

Tesla to Soon Start Delivering Model 3 & Y with AMD Ryzen Chips to Europe, Parts Catalog Hints

Dec 6, 2021 By Eva Fox 3 Comments

Our Previous AMD Research

faulTPM: Exposing AMD fTPMs' Deepest Secrets

Hans Niklas Jacob*, Christian Werling*, Robert Buhren, Jean-Pierre Seifert†
Technische Universität Berlin – SecT
†*also: Fraunhofer SIT*
{ *hnj, cwerling, robert.buhren, jpseifert* }*@sect.tu-berlin.de*

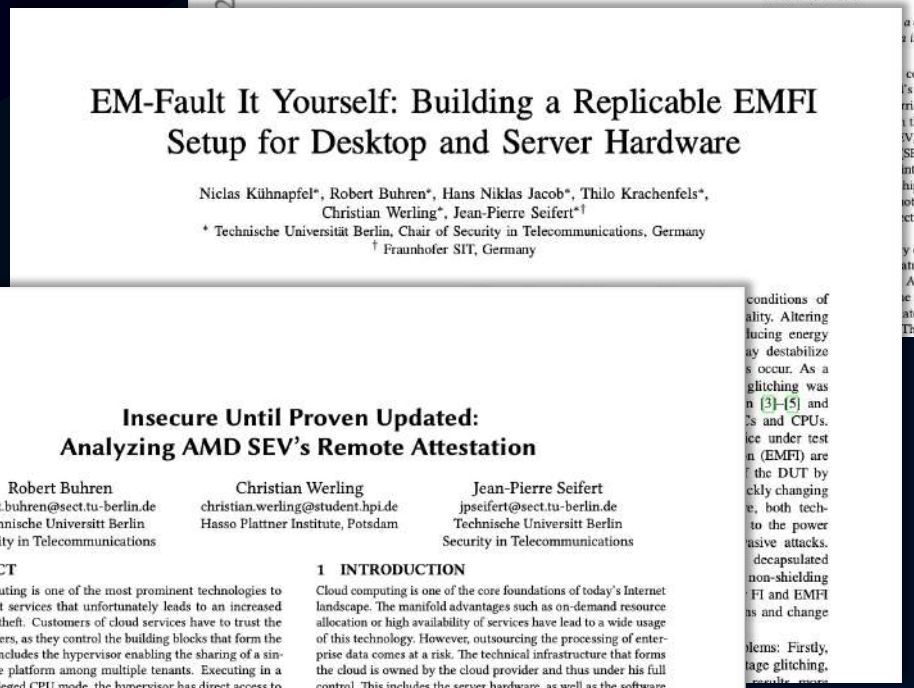
One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization

Robert Buhren
robert.buhren@sect.tu-berlin.de
Technische Universität Berlin - SECT

Hans Niklas Jacob
hnj@sect.tu-berlin.de
Technische Universität Berlin - SECT

Thilo Krachenfels
tkrachenfels@sect.tu-berlin.de
Technische Universität Berlin - SECT

Jean-Pierre Seifert
jpseifert@sect.tu-berlin.de
Technische Universität Berlin - SECT
Fraunhofer SIT



EM-Fault It Yourself: Building a Replicable EMFI Setup for Desktop and Server Hardware

Niclas Kühnapfel*, Robert Buhren*, Hans Niklas Jacob*, Thilo Krachenfels*,
Christian Werling*, Jean-Pierre Seifert*†
* Technische Universität Berlin, Chair of Security in Telecommunications, Germany
† Fraunhofer SIT, Germany

Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation

Robert Buhren
robert.buhren@sect.tu-berlin.de
Technische Universitt Berlin
Security in Telecommunications

Christian Werling
christian.werling@student.hpi.de
Hasso Plattner Institute, Potsdam

Jean-Pierre Seifert
jpseifert@sect.tu-berlin.de
Technische Universitt Berlin
Security in Telecommunications

ABSTRACT

Cloud computing is one of the most prominent technologies to host Internet services that unfortunately leads to an increased risk of data theft. Customers of cloud services have to trust the cloud providers, as they control the building blocks that form the cloud. This includes the hypervisor enabling the sharing of a single hardware platform among multiple tenants. Executing in a higher-privileged CPU mode, the hypervisor has direct access to the memory of virtual machines. While data at rest can be protected using well-known disk encryption methods, data residing in main memory is still threatened by a potentially malicious cloud provider.

AMD Secure Encrypted Virtualization (SEV) claims a new level of protection in such cloud scenarios. AMD SEV encrypts the main memory of virtual machines with VM-specific keys, thereby denying the higher-privileged hypervisor access to a guest's memory.

1 INTRODUCTION

Cloud computing is one of the core foundations of today's Internet landscape. The manifold advantages such as on-demand resource allocation or high availability of services have lead to a wide usage of this technology. However, outsourcing the processing of enterprise data comes at a risk. The technical infrastructure that forms the cloud is owned by the cloud provider and thus under his full control. This includes the server hardware, as well as the software components that allow the co-location of multiple virtual machines on a single host.

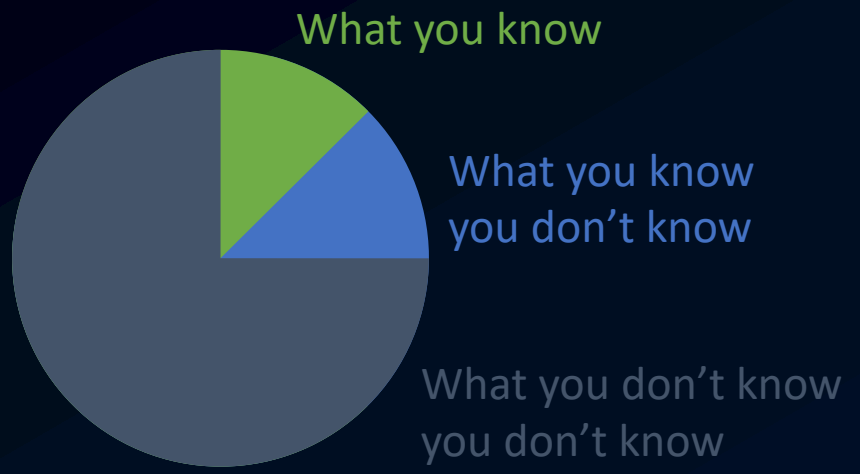
Therefore security concerns impede the deployment of confidential data and applications in cloud scenarios [14, 19]. The potential threats range from misconfiguration of software components over cloud provider admin access to foreign government access [8].

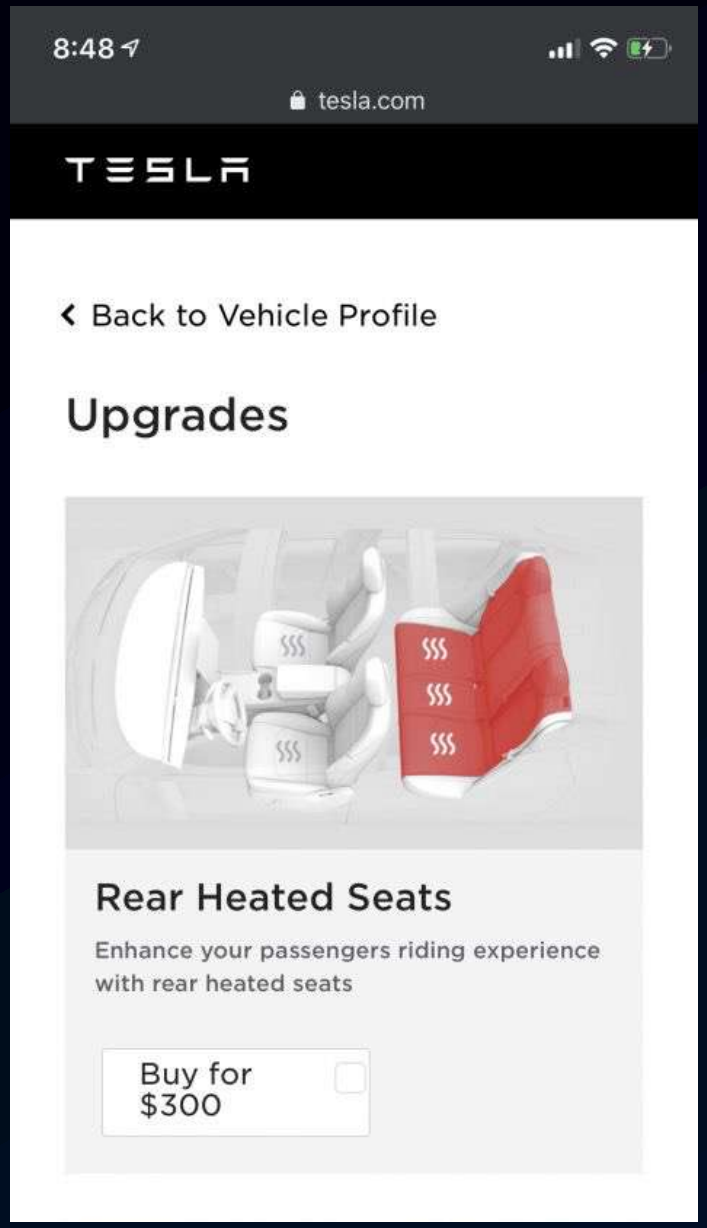
To counter these threats, the research community, as well as industry, proposed new approaches to allow secure cloud computing

Why Jailbreak a Car?

Many reasons:

- to ”look around” (curiosity)
- to replace its software
- **to activate soft-locked features**

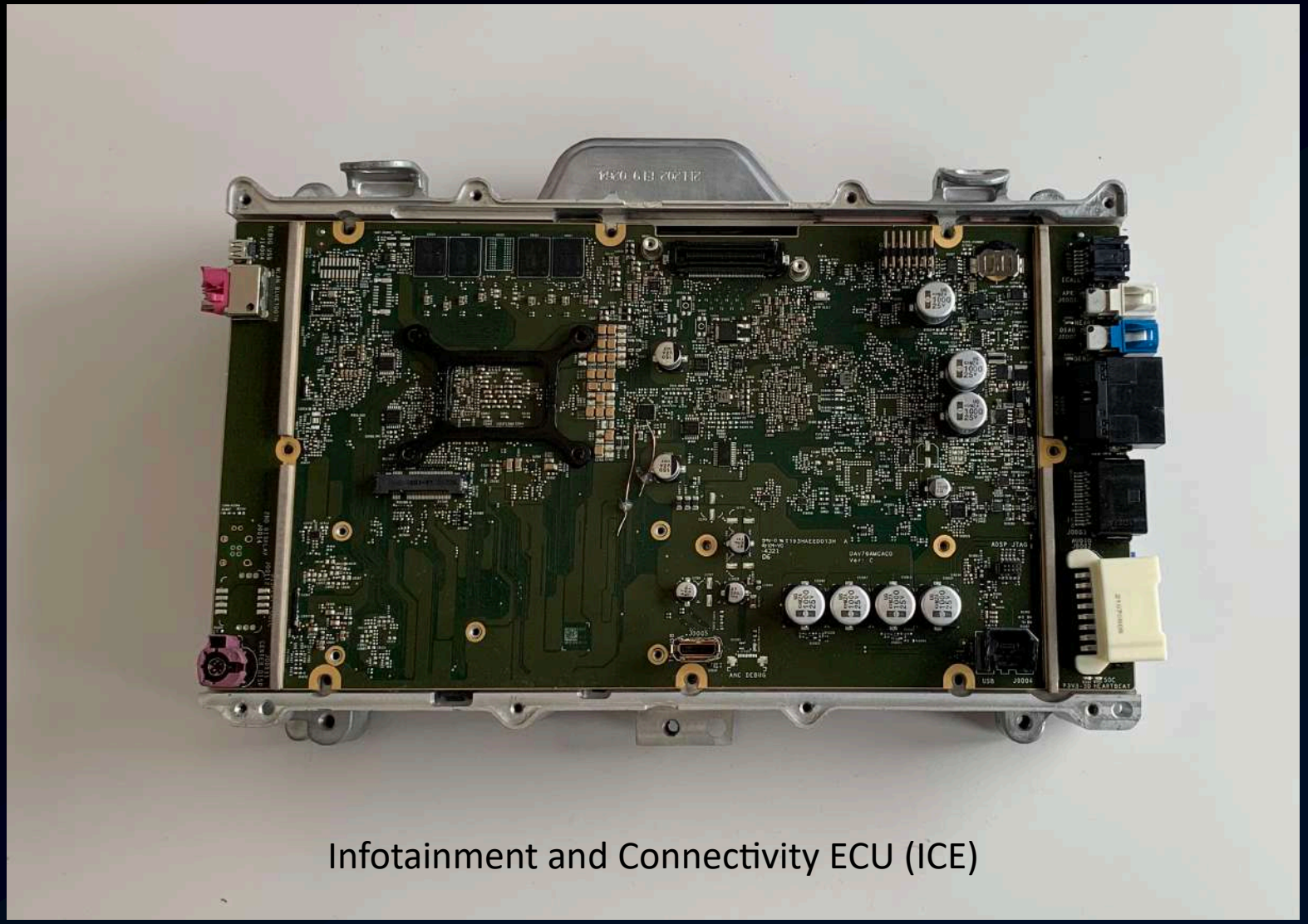



8:48
tesla.com
TESLA
Back to Vehicle Profile
Upgrades
Rear Heated Seats
Enhance your passengers riding experience with rear heated seats
Buy for $300

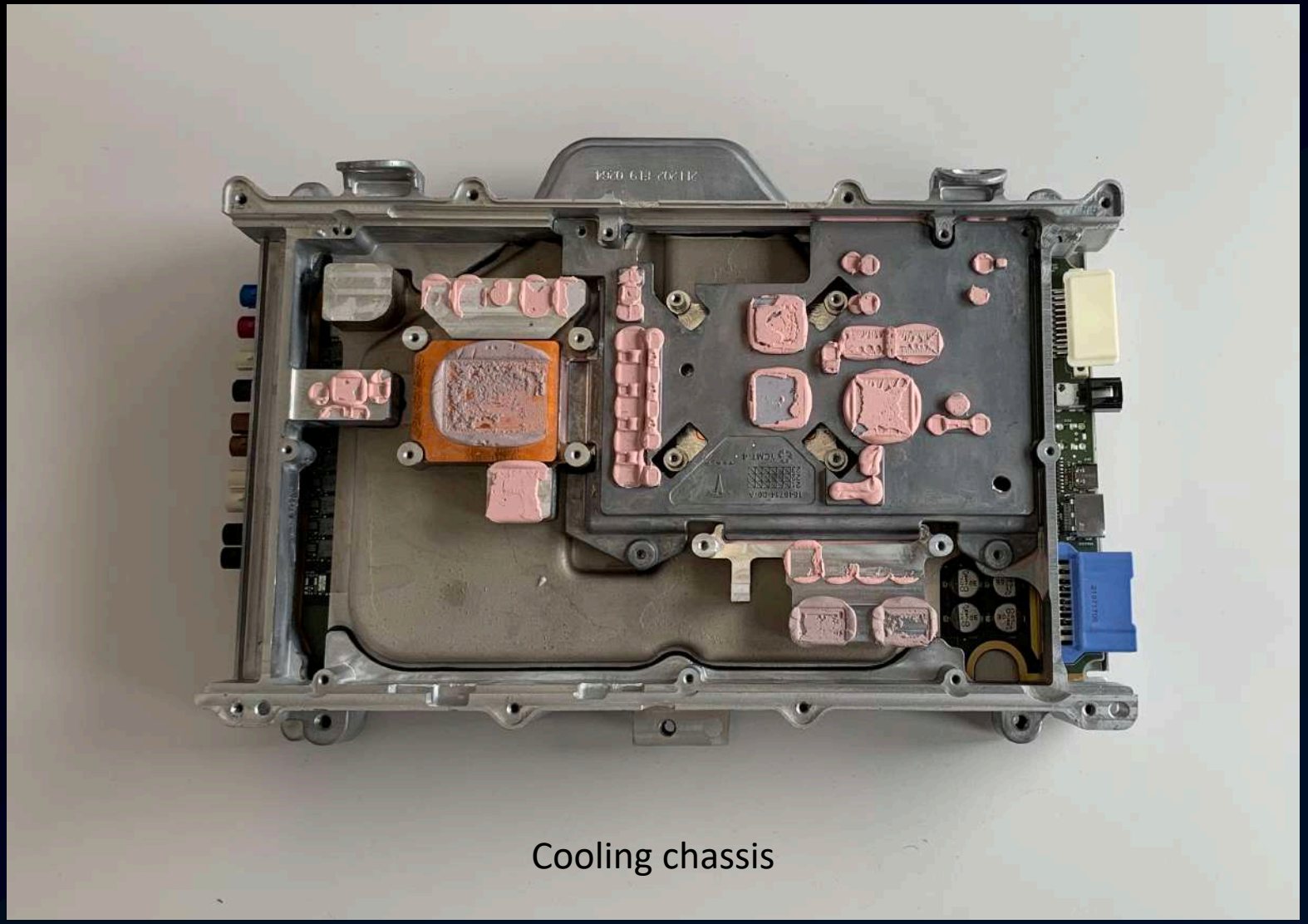
Outline

1. Analyzing Boot and Firmware Security
2. Hotwiring the Infotainment system
3. Extracting Secrets from the Tesla

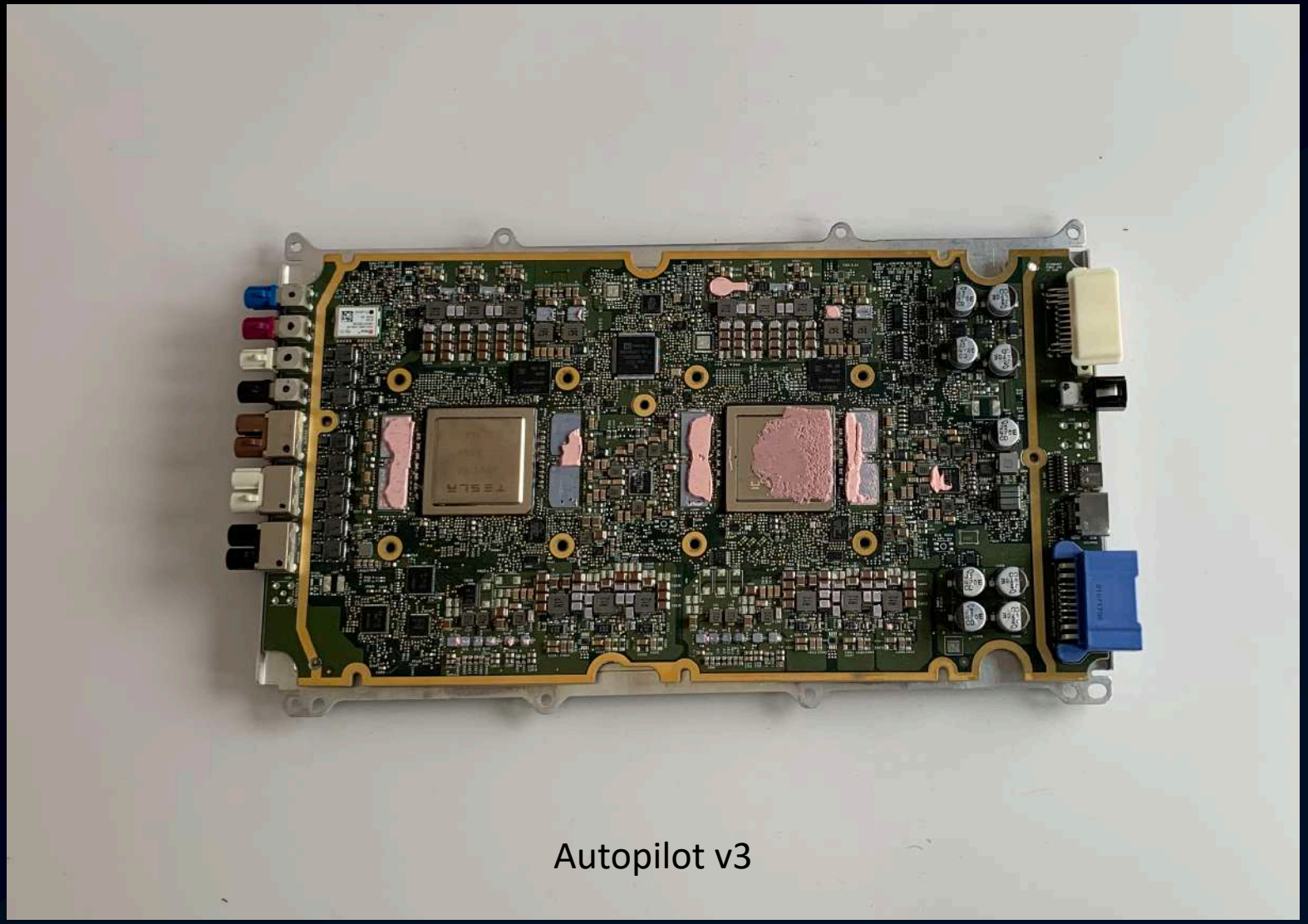
Model 3 Car Computer



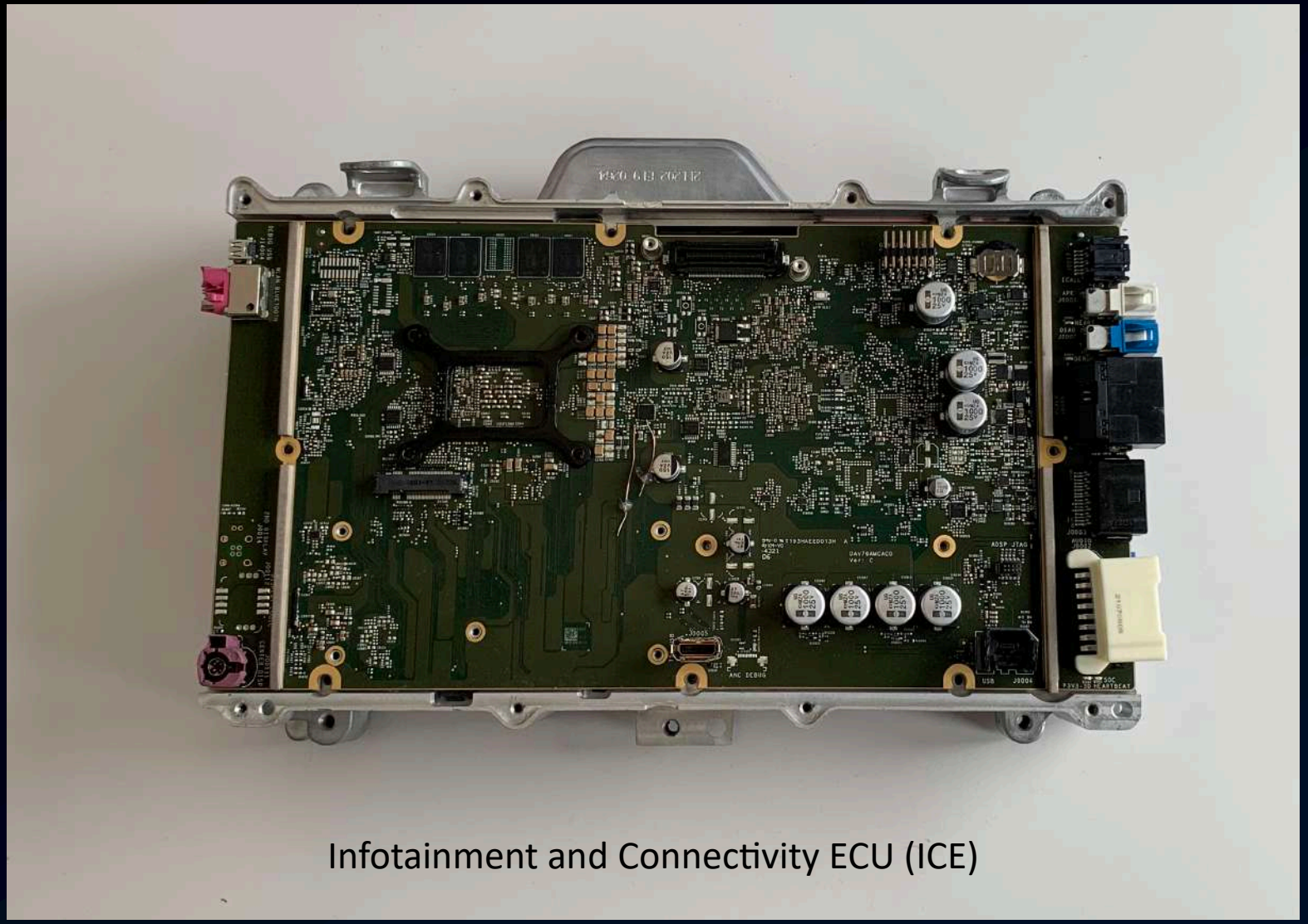
Infotainment and Connectivity ECU (ICE)



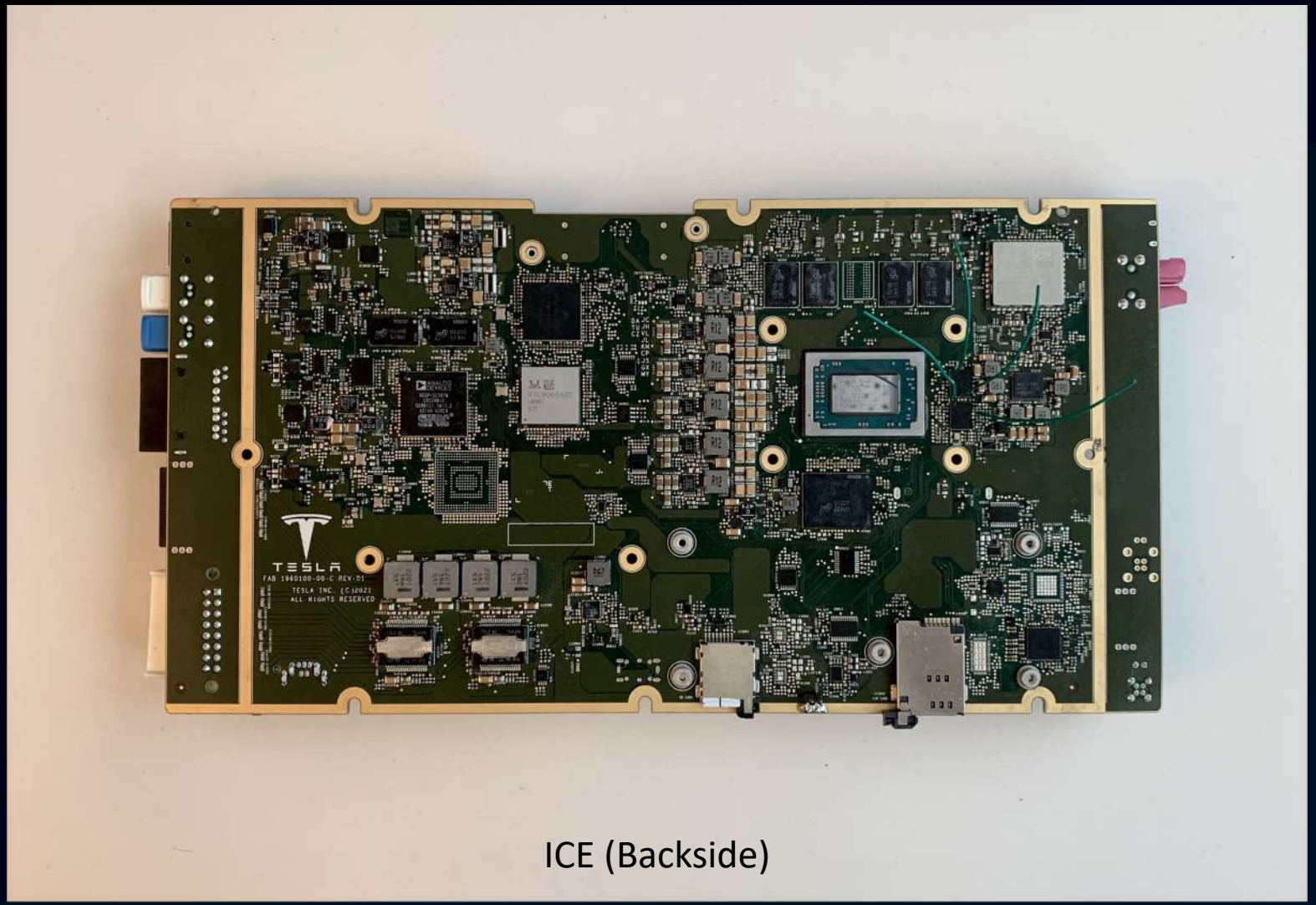
Cooling chassis



Autopilot v3

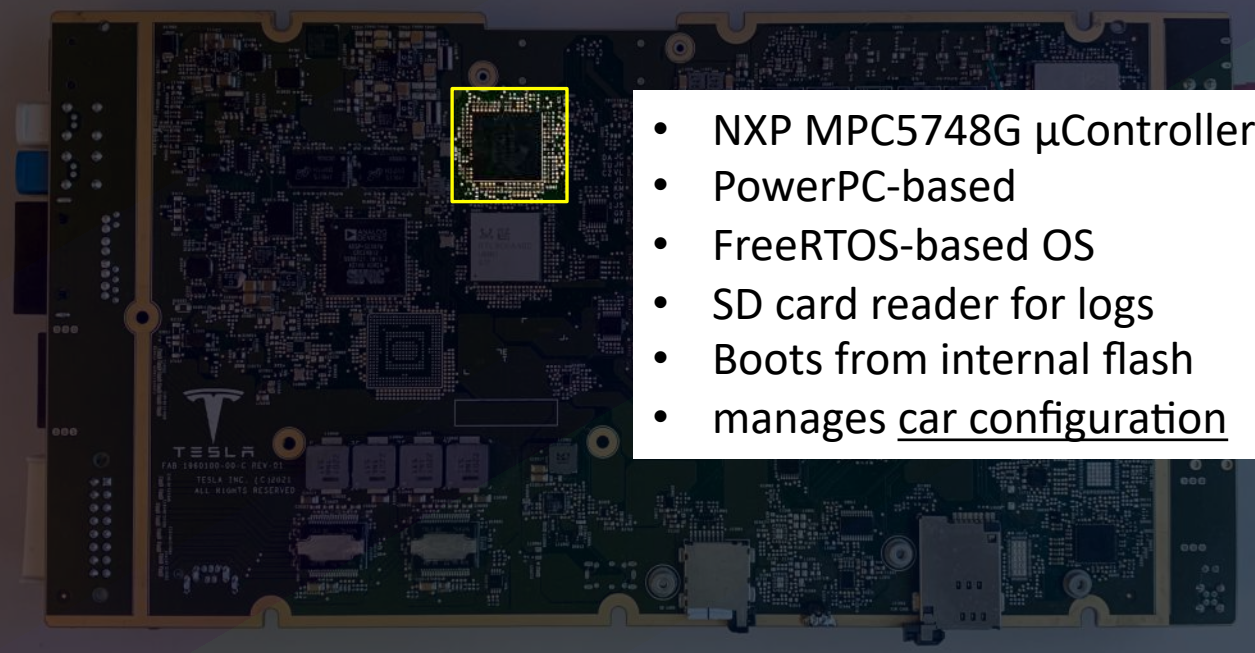


Infotainment and Connectivity ECU (ICE)



ICE (Backside)

Gateway



- NXP MPC5748G µController
- PowerPC-based
- FreeRTOS-based OS
- SD card reader for logs
- Boots from internal flash
- manages car configuration

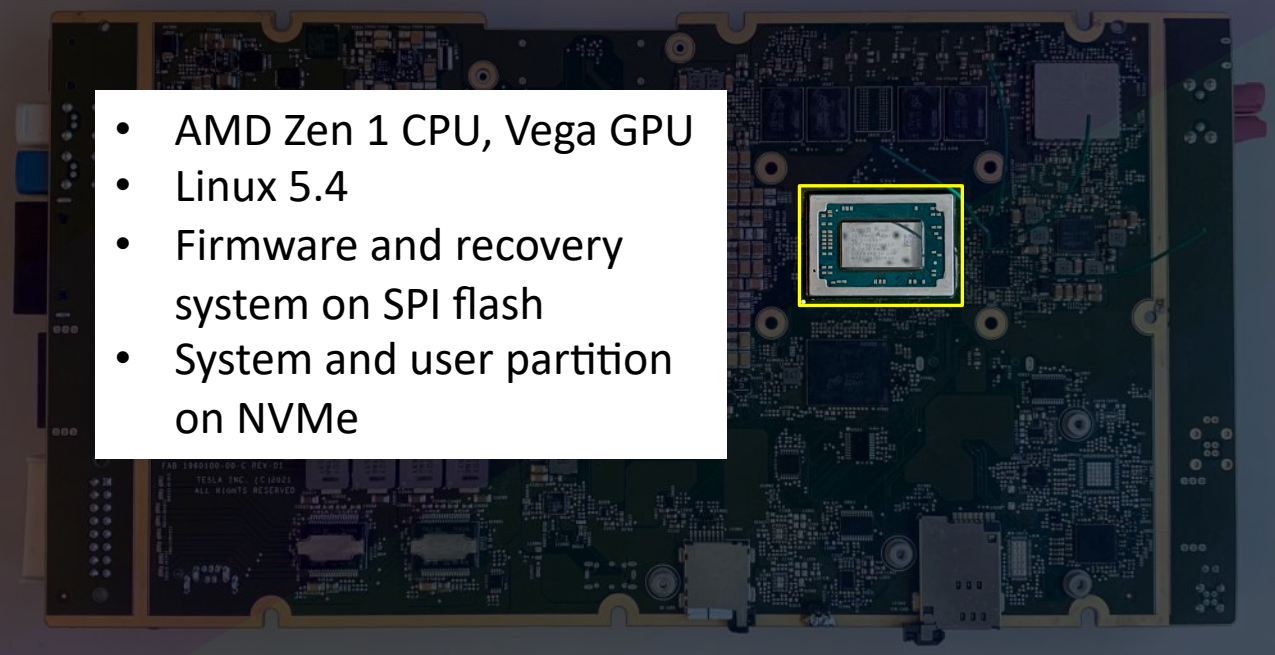
ICE (Backside)

Car configuration

- Stored and managed by the Gateway
- Lists (paid) hardware and software features
  - Car performance
  - Battery capacity (for software-locked batteries)
  - Level of Autopilot: (Enhanced) Autopilot, Full Self-Driving capability
  - Car region
  - Rear seat heaters

Infotainment APU

- AMD Zen 1 CPU, Vega GPU
- Linux 5.4
- Firmware and recovery system on SPI flash
- System and user partition on NVMe



ICE (Backside)

Previous Tesla Hacking

- Threat model: *Outsider* who is remote or in physical proximity
- Goal: Access/control car
- Software-based vulnerabilities: Can be fixed by Tesla over-the-air

Platform Threats from the *Inside*

- Threat model: Insider who already has **digital and physical access to the car**
- Goal: Tweak car beyond normal flows
  - activate **soft-locked features** without paying
  - lift repair and regulation restrictions
- Insider not limited to software-based attacks

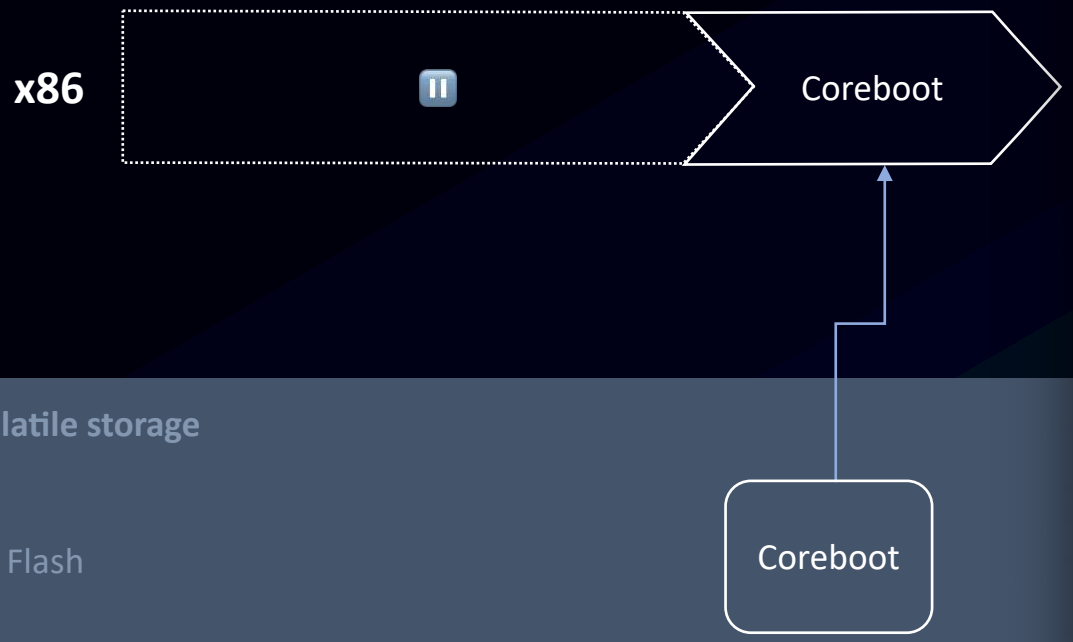
Verified Boot

Non-volatile storage

SPI Flash

NVMe

Verified Boot

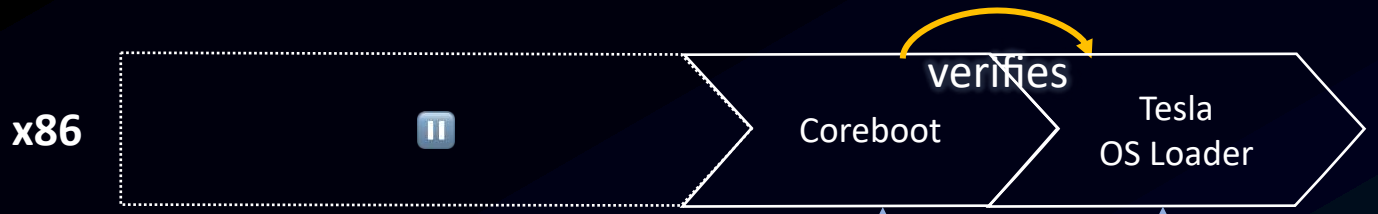


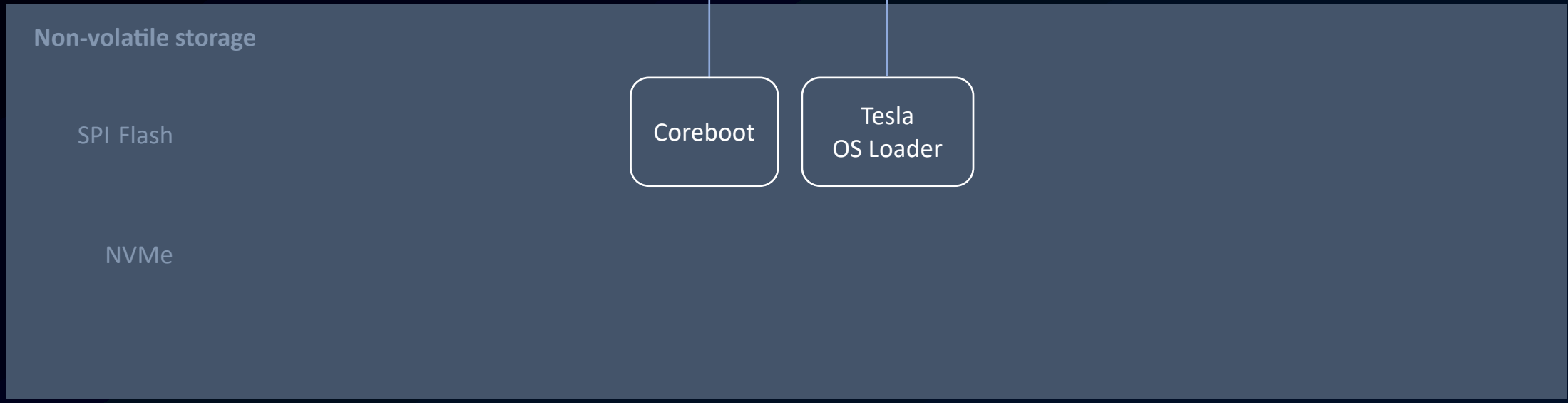

```
cwerling@MackBockProPro:~/Git/tesla-hacking/logs
coreboot-archive/develop/2021.44.25.2-8836-gb025c688348a
Thu Jan 13 14:46:27 UTC 2022 bootblock starting (log leve
l: 5)...
PMxC0 STATUS: 0x800 BIT11


coreboot-archive/develop/2021.44.25.2-8836-gb025c688348a
Thu Jan 13 14:46:27 UTC 2022 romstage starting (log level
: 5)...
POST: 0x41
POST: 0x42
POST: 0x43
POST: 0x34
POST: 0x36
POST: 0x92
POST: 0x98
SF size 0x2000000 does not correspond to CONFIG_ROM_SIZE
0x1000000!!
POST: 0x44


coreboot-archive/develop/2021.44.25.2-8836-gb025c688348a
Thu Jan 13 14:46:27 UTC 2022 ramstage starting (log level
: 5)...
POST: 0x39
POST: 0x80
POST: 0x70
POST: 0x71
Board name: Spinach
```

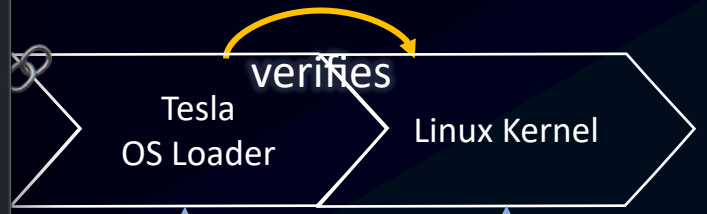
Verified Boot

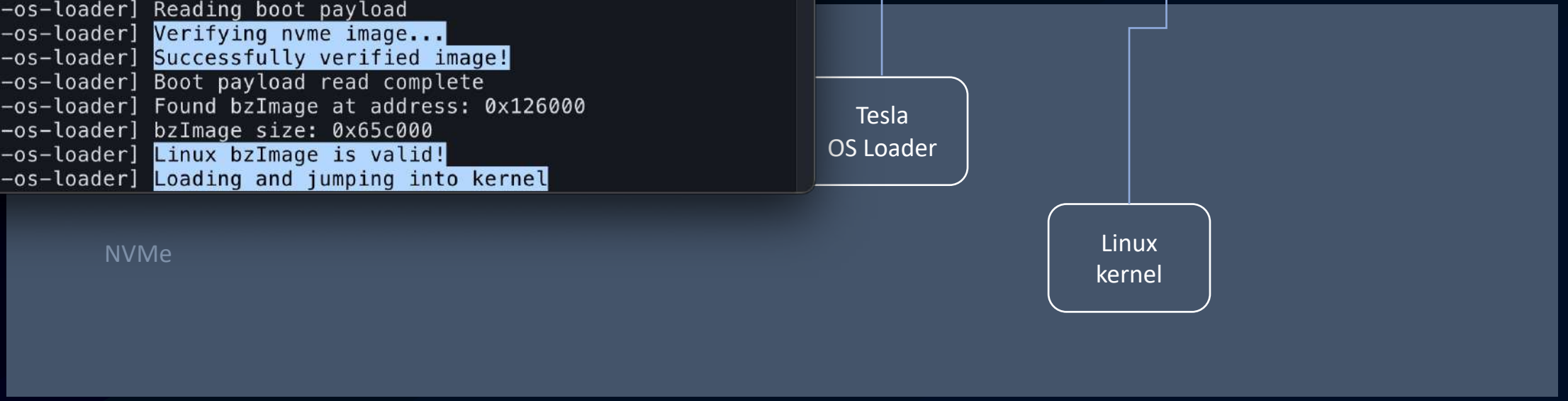




cwerling@MackBockProPro:~/Git/tesla-hacking/logs ⌥⌘1

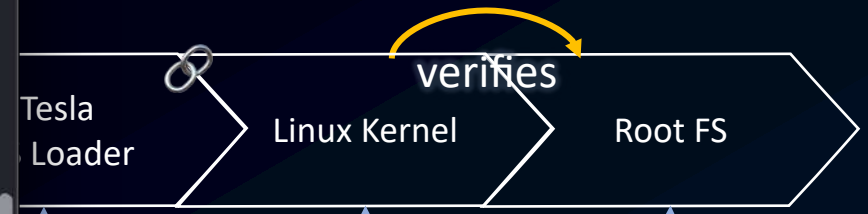
```
[tesla-os-loader] v7
[tesla-os-loader] starting
[tesla-os-loader] Booted from SPI Bank B
[tesla-os-loader] Found NVMe
[tesla-os-loader] BAR address: 0xD0200000
[tesla-os-loader] Attempting to boot online image from the NVMe
[tesla-os-loader] initialized NVMe controller access
[tesla-os-loader] NVMe with boot partition support detected OK
[tesla-os-loader] Reading header from NVMe boot partition ID: 0x1
[tesla-os-loader] Reading container header
[tesla-os-loader] Boot partition header read complete
[tesla-os-loader] Verifying header...
vb2_secdata_kernel_set: secdata_kernel flags updated from 0x0 to 0x6
vb2_verify_keyblock: Checking keyblock signature...
vb2_verify_kernel_preamble: Verifying kernel preamble.
[tesla-os-loader] Successfully verified boot partition header!
[tesla-os-loader] container_type: Vboot bzimage
[tesla-os-loader] image_version: Version 1
[tesla-os-loader] header_crc32: 0x43524254
[tesla-os-loader] Reading boot payload
[tesla-os-loader] Verifying nvme image...
[tesla-os-loader] Successfully verified image!
[tesla-os-loader] Boot payload read complete
[tesla-os-loader] Found bzImage at address: 0x126000
[tesla-os-loader] bzImage size: 0x65c000
[tesla-os-loader] Linux bzImage is valid!
[tesla-os-loader] Loading and jumping into kernel
```

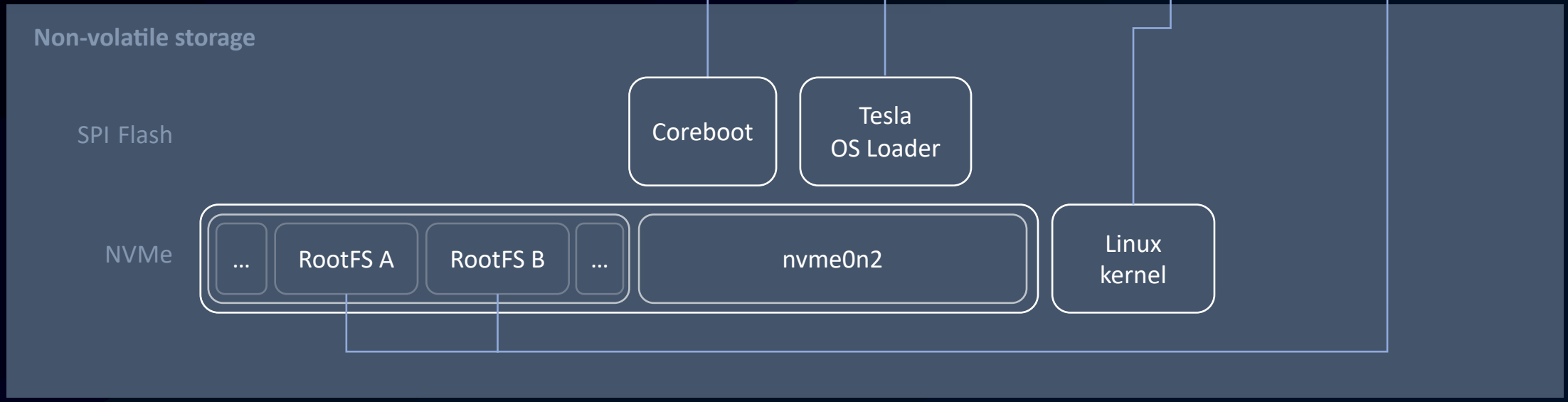




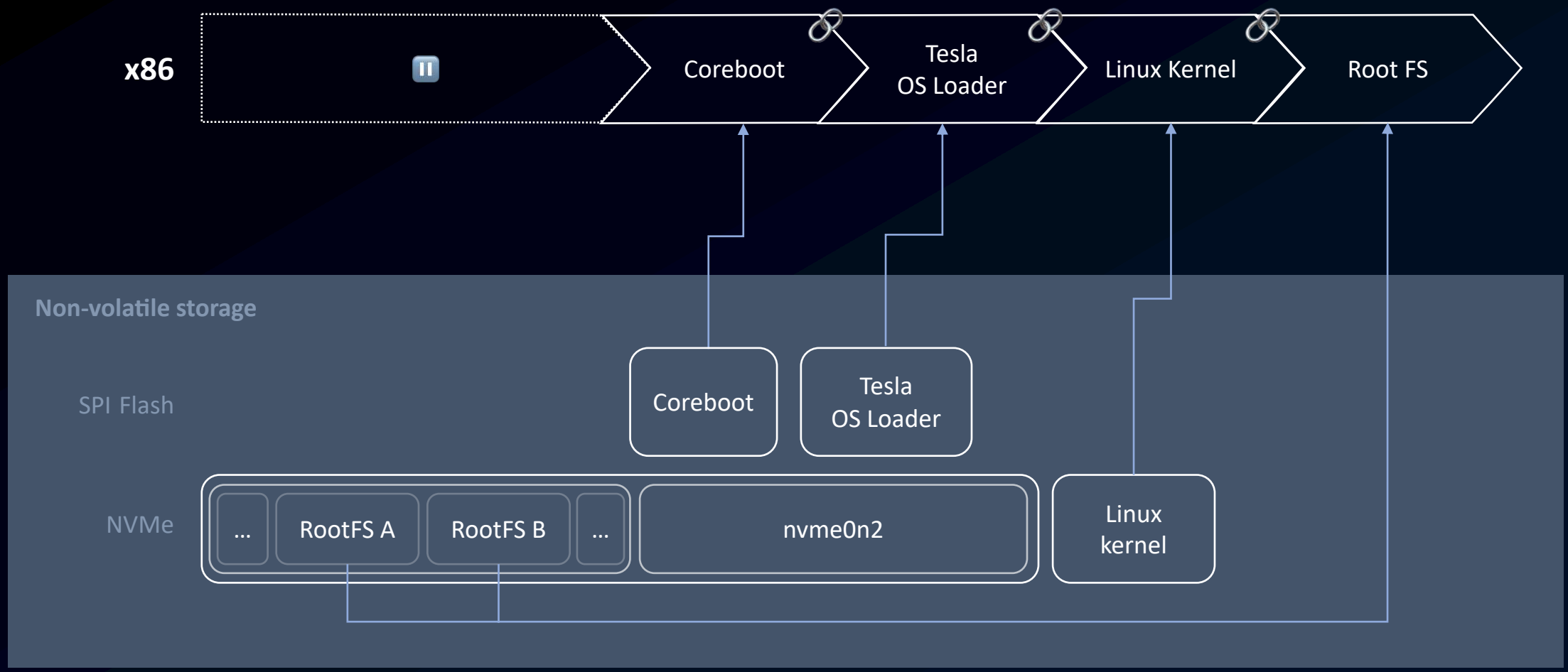
cwerling@MackBockProPro:~/Git/tesla-hacking/logs ⌥⌘1

```
[    6.284733] Run /init as init process
[    6.289009] verity-init: info: verity-init booting
[    6.293867] verity-init: info: root block device: /dev/nvme0n1p3
[    6.299906] verity-init: info: customer_key_lock 1
[    6.304727] verity-init: info: bios_key_rev_id 0x0001
[    6.309791] verity-init: info: creating root verity device
[    6.318866] device-mapper: verity: sha256 using implementation "sha2
56-ni"
[    6.326004] device-mapper: table: 253:0: adding target device (start
 sect 0 len 2783320) caused an alignment inconsistency
[    6.337075] device-mapper: table: 253:0: adding target device (start
 sect 0 len 2783320) caused an alignment inconsistency
```





Verified Boot

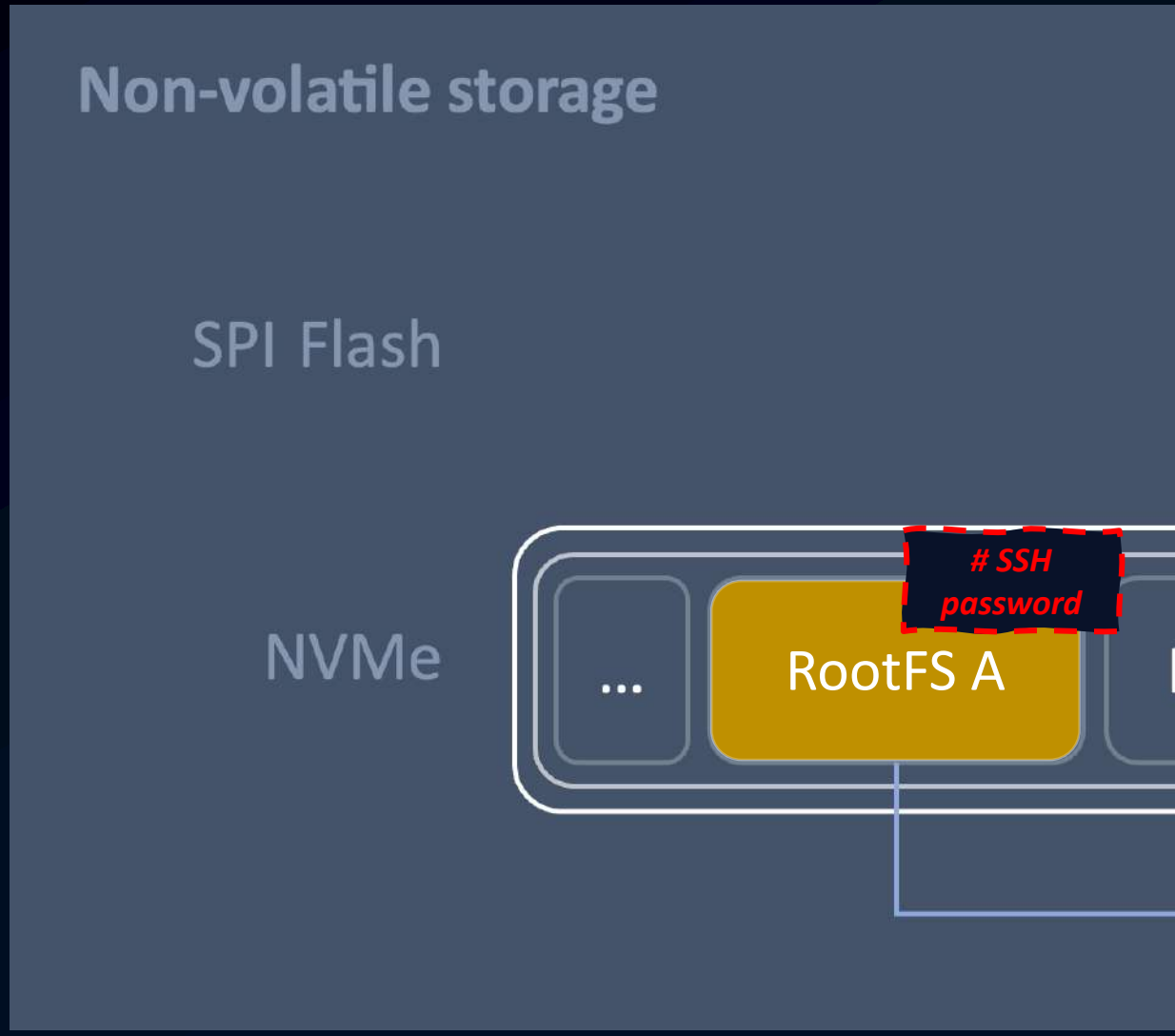


How to get a **root shell**

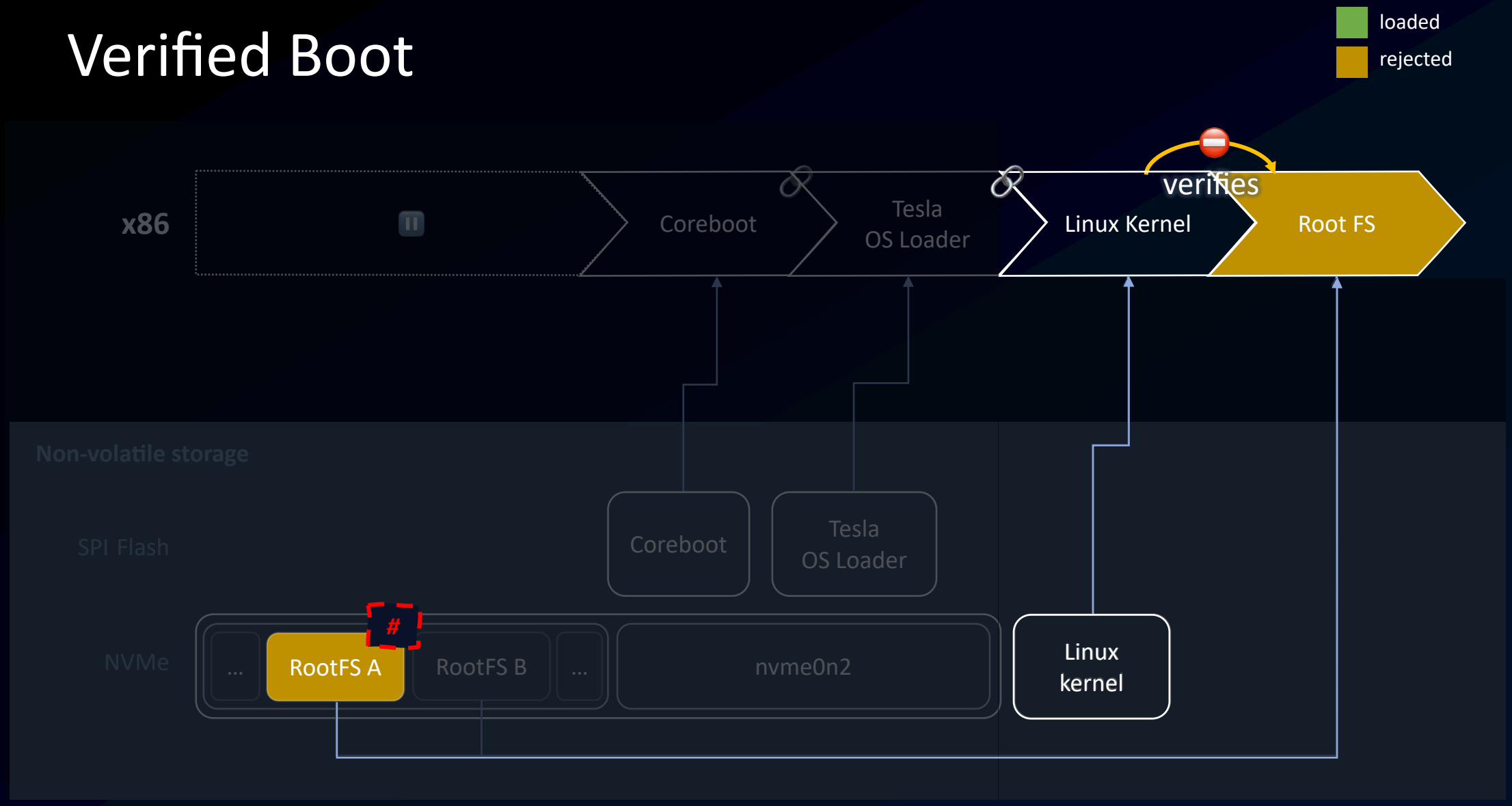
Many options:

- Spawn serial shell on boot
- Add SSH key to `authorized_keys` file
- Add known SSH password

They all require **changes** to the Root file system

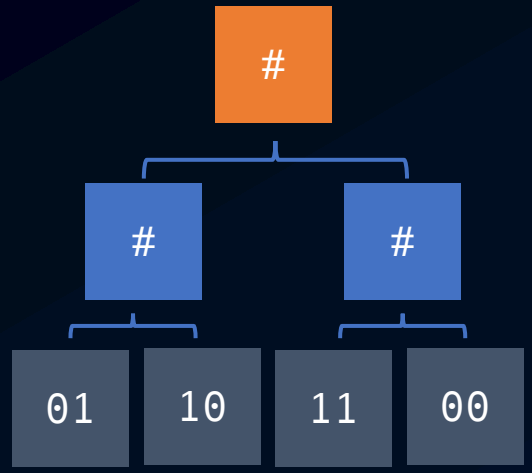


Verified Boot

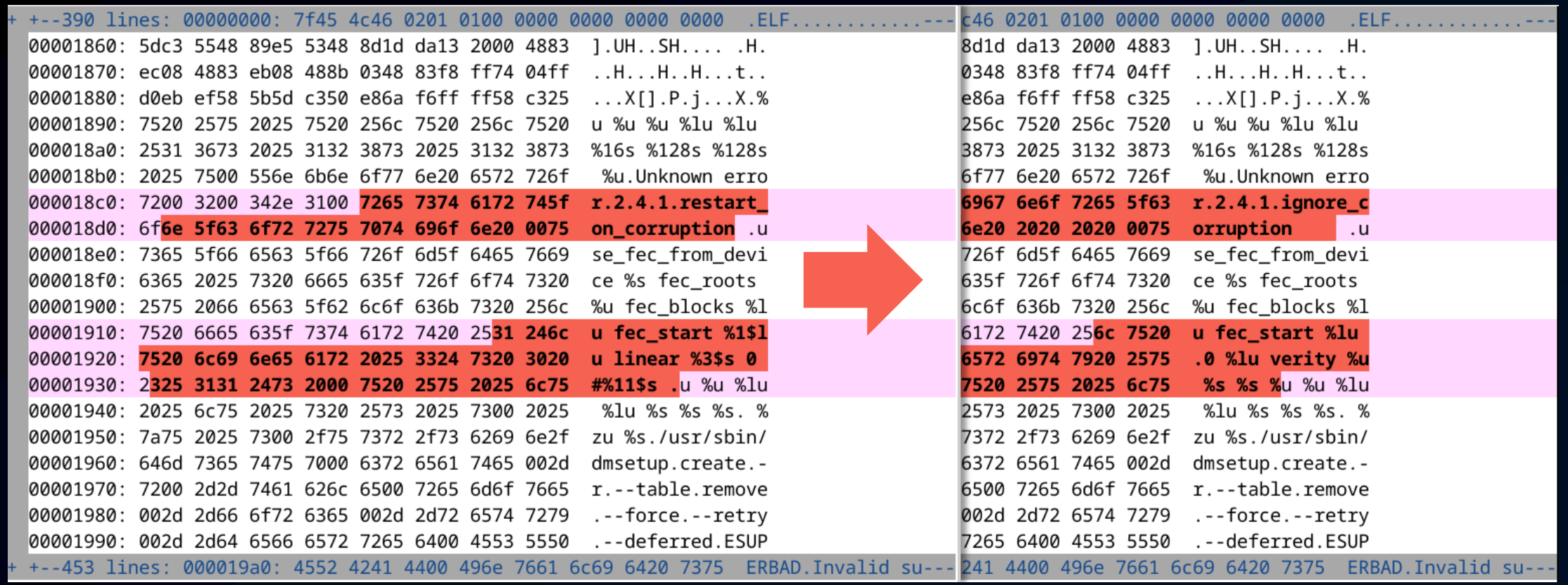


dm-verity

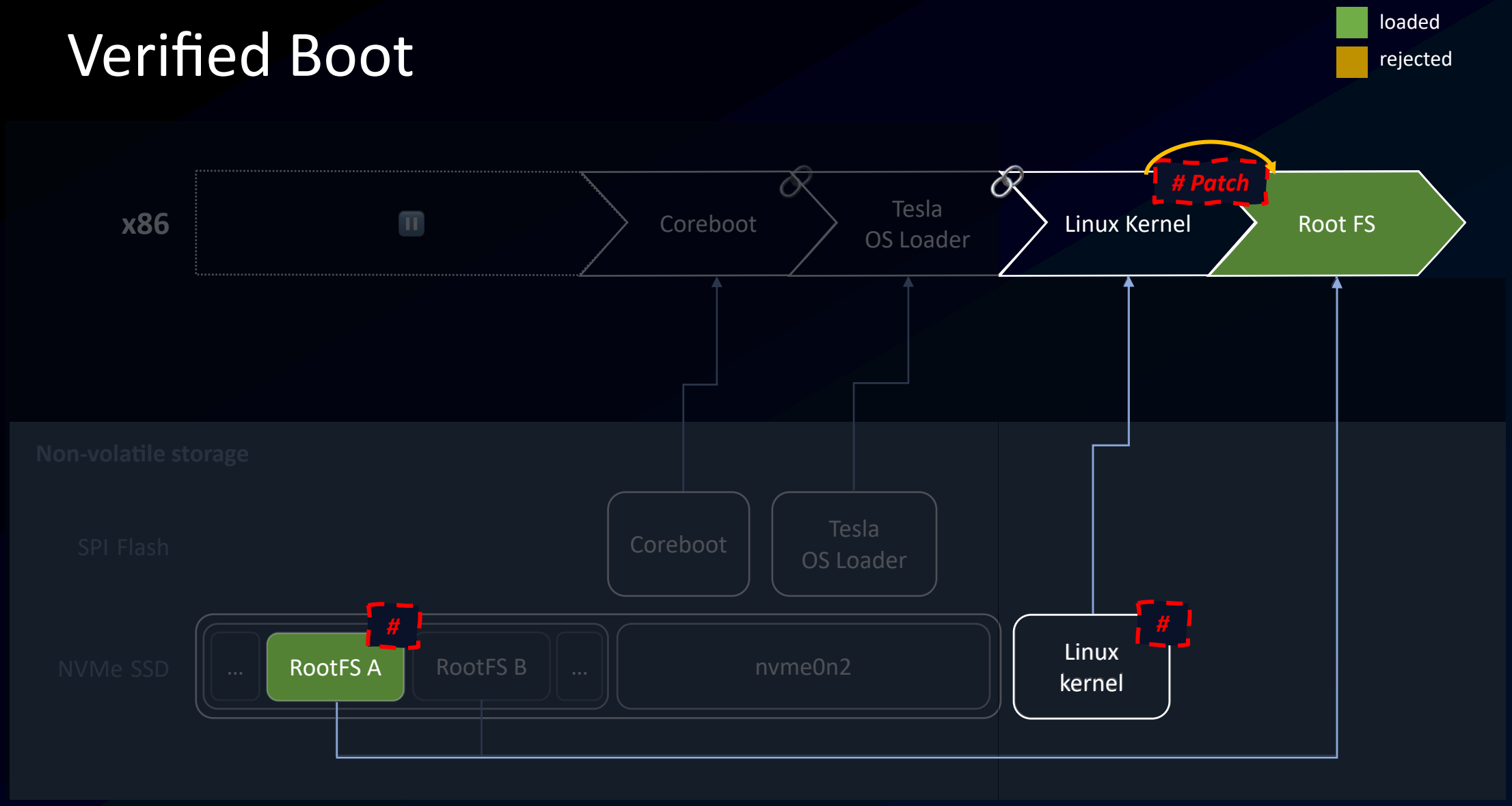
- Integrity checking of block devices
  - When a block is read into memory, it’s hashed in parallel
- Merkle tree used to efficiently store and verify hashes of individual block
  - Trusted root file system represented by root hash
  - Intermediate hashes stored alongside data



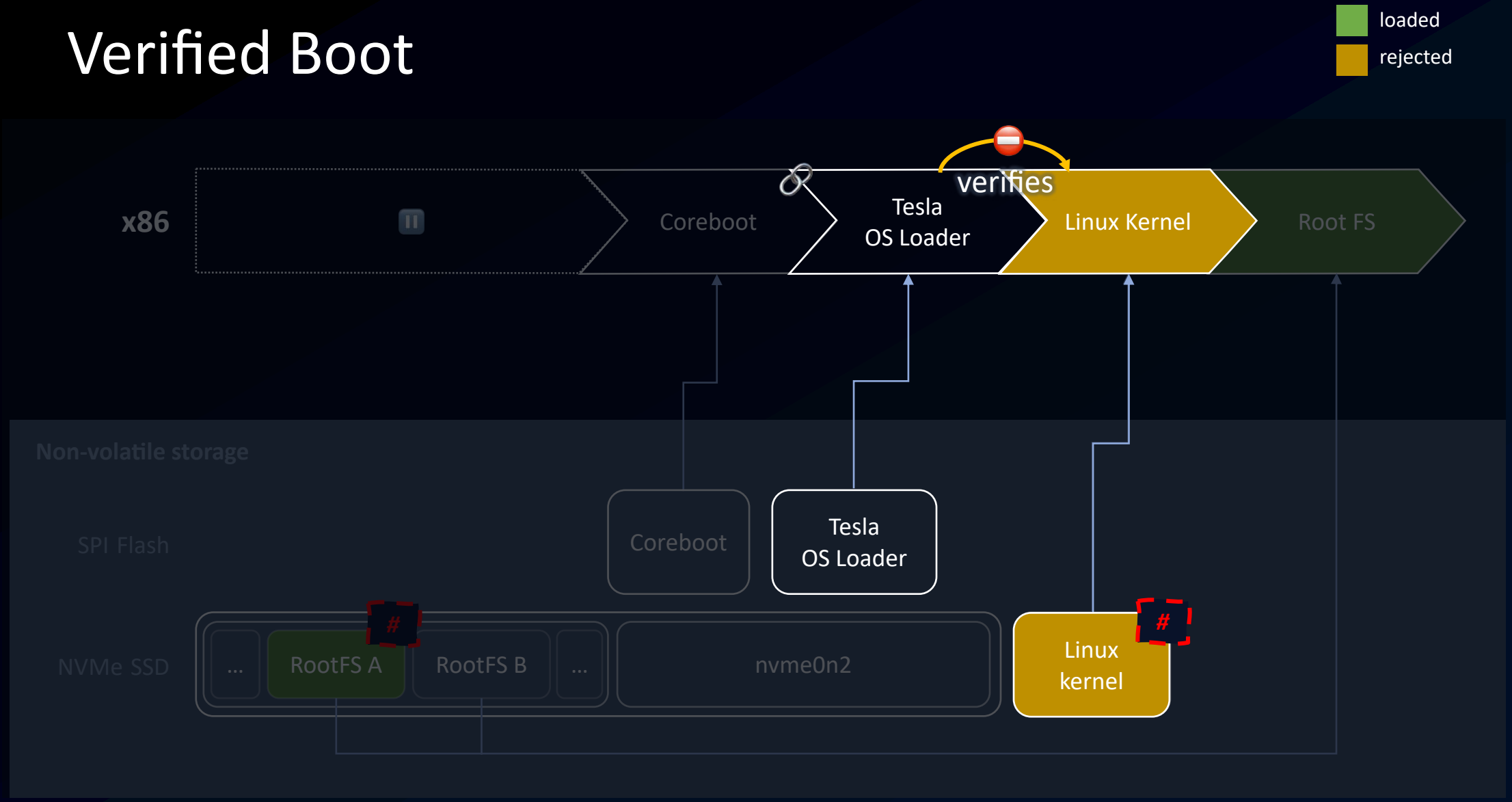
dm-verity *# Patch*



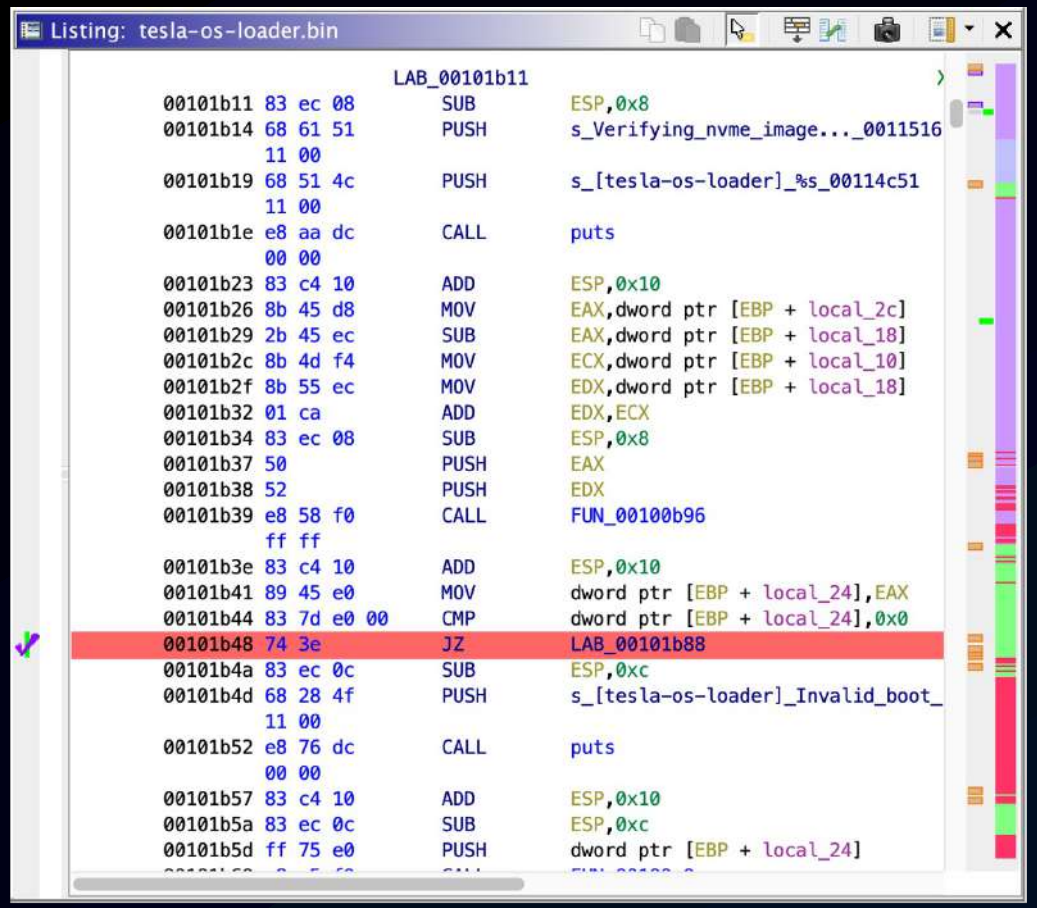
Verified Boot

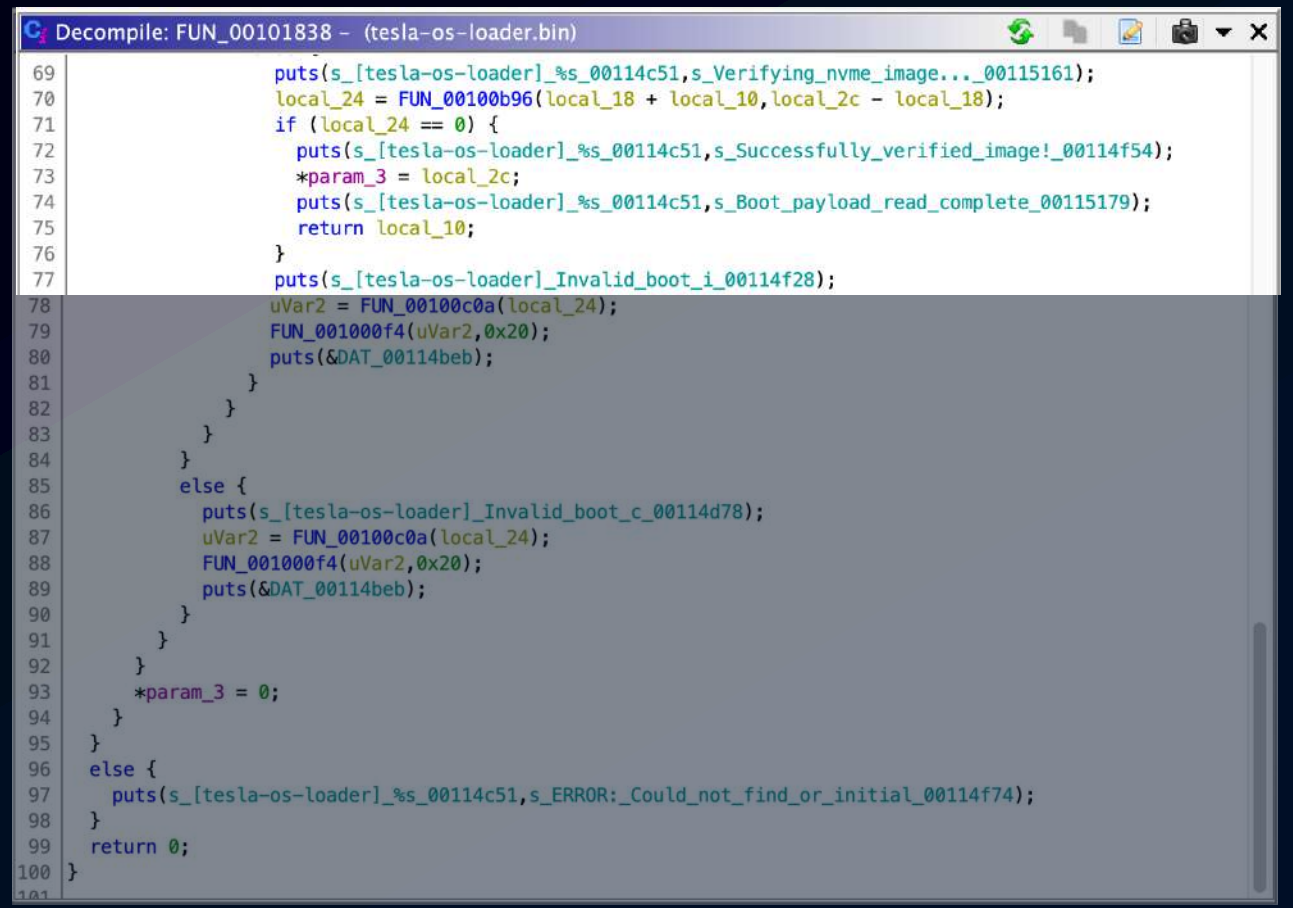


Verified Boot

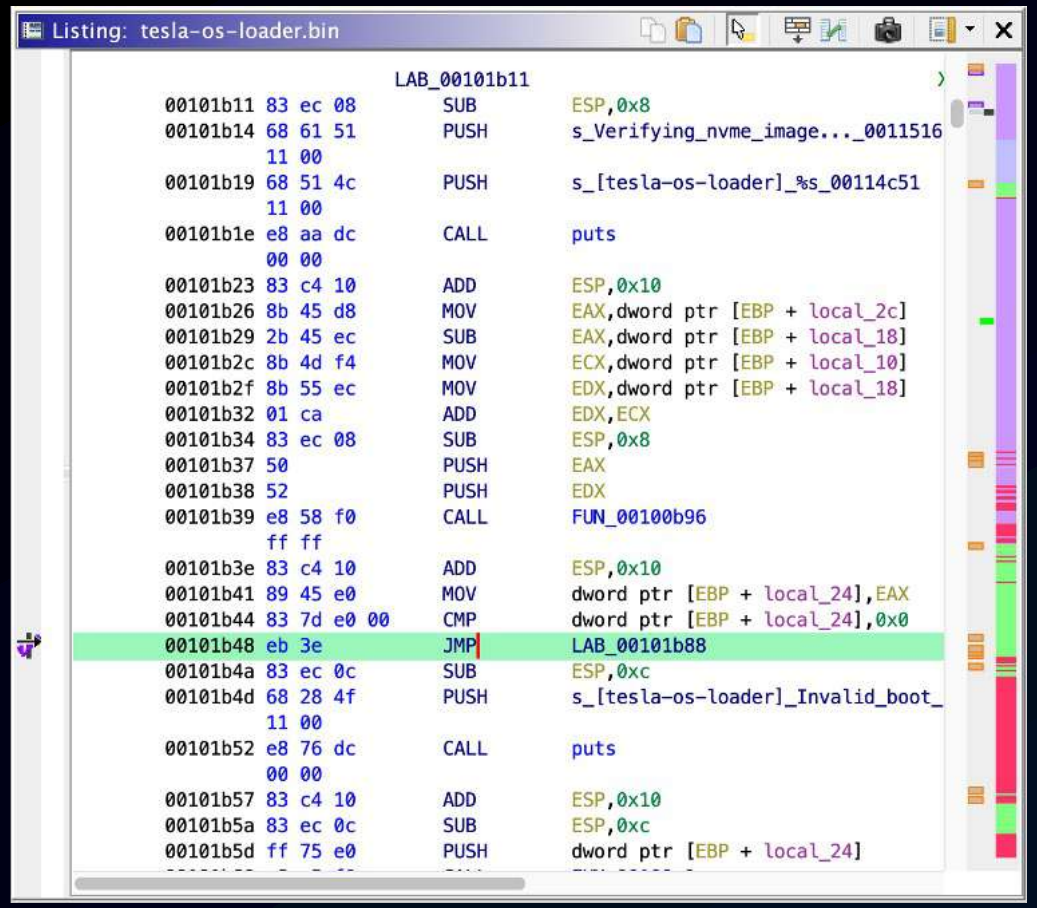


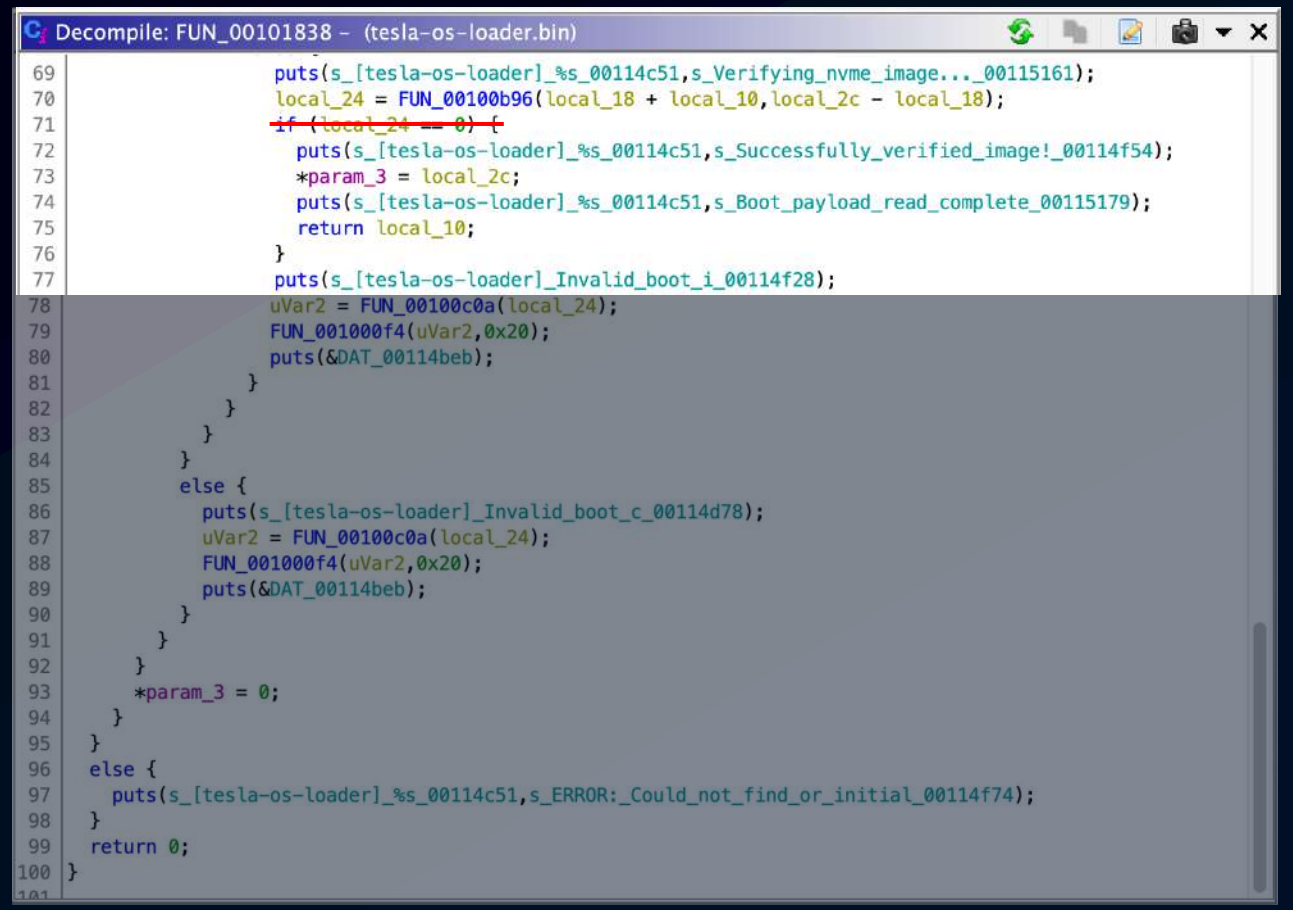
Tesla OS Loader **# Patch**



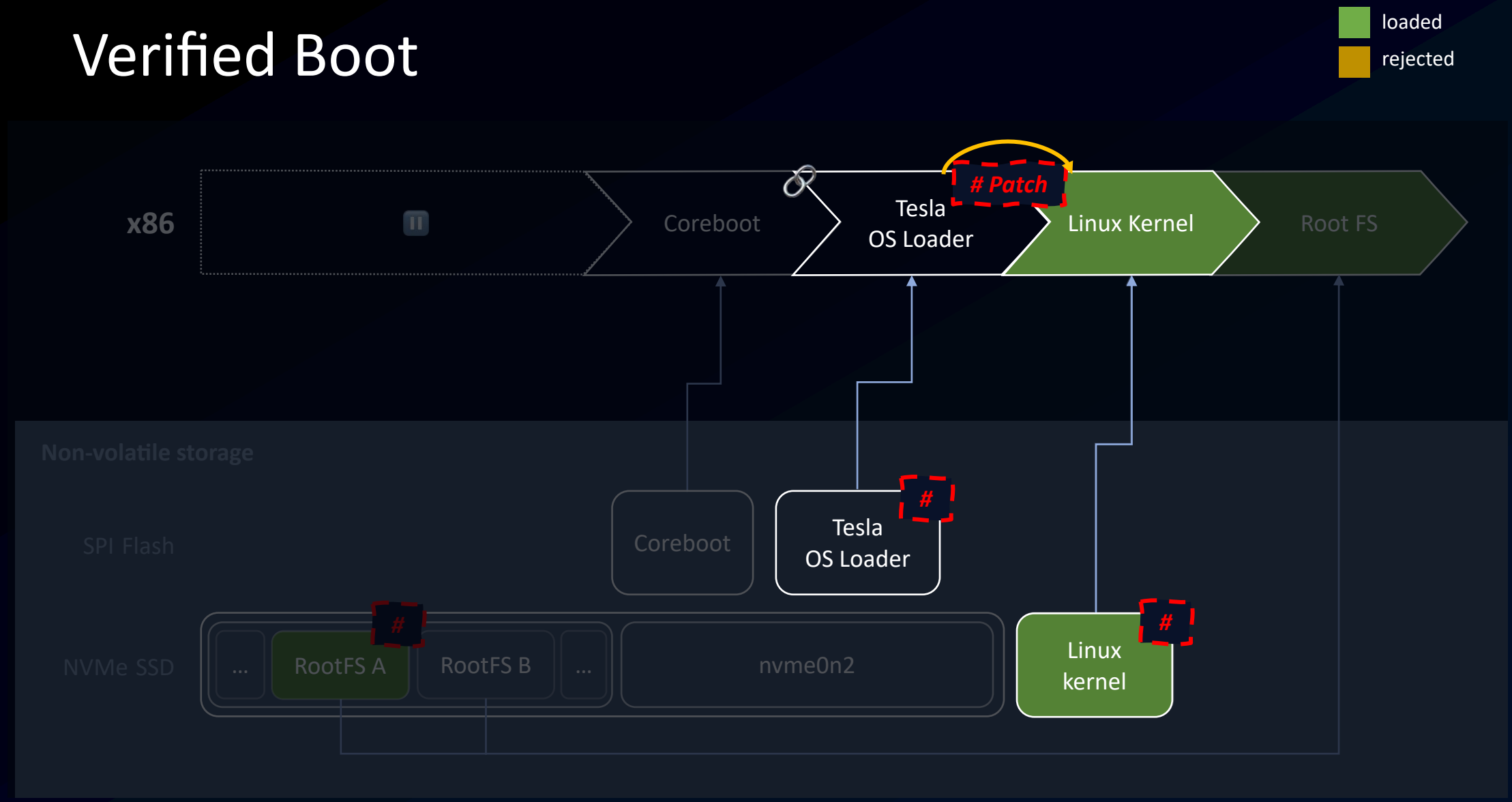


Tesla OS Loader **# Patch**

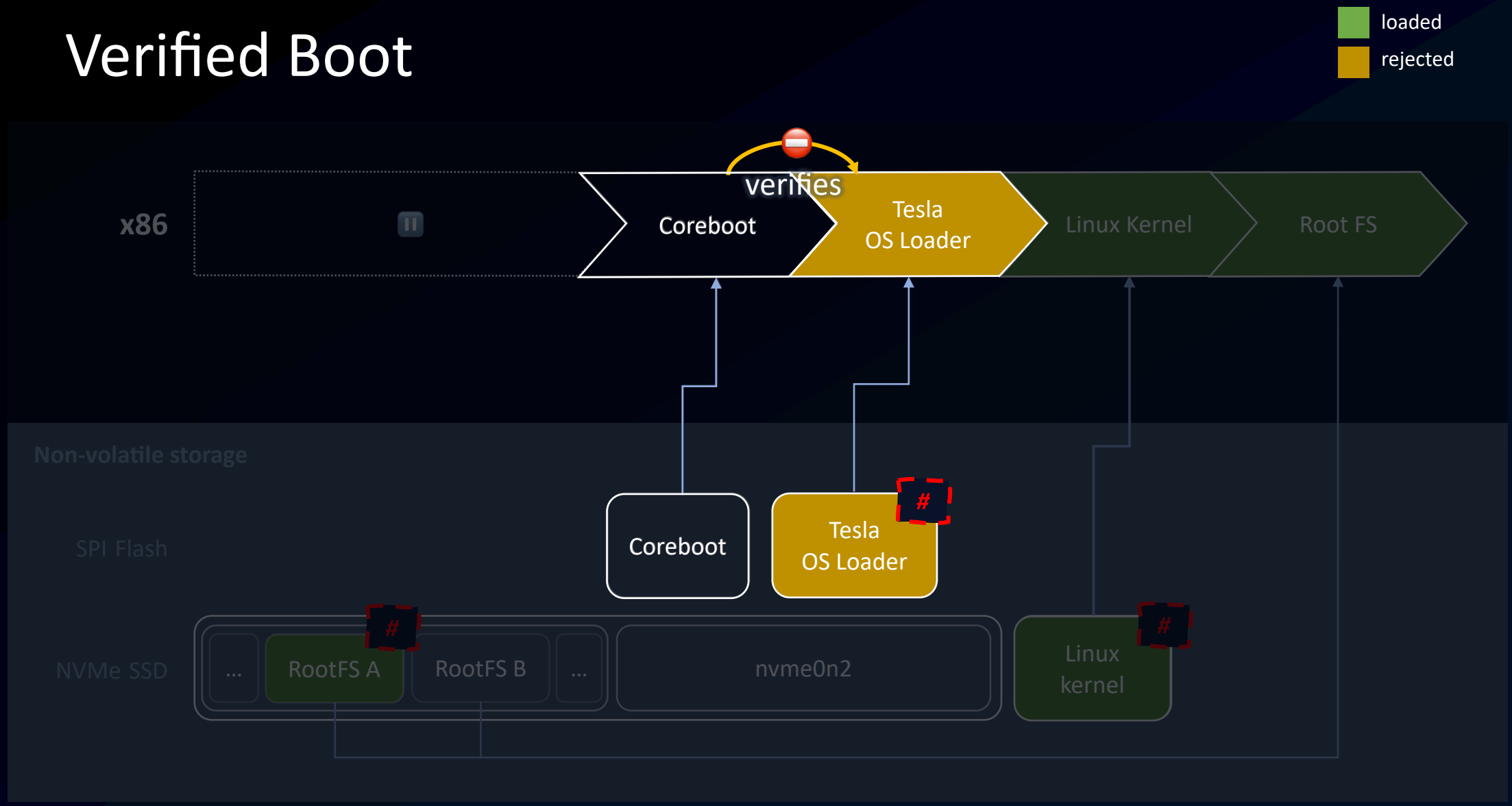




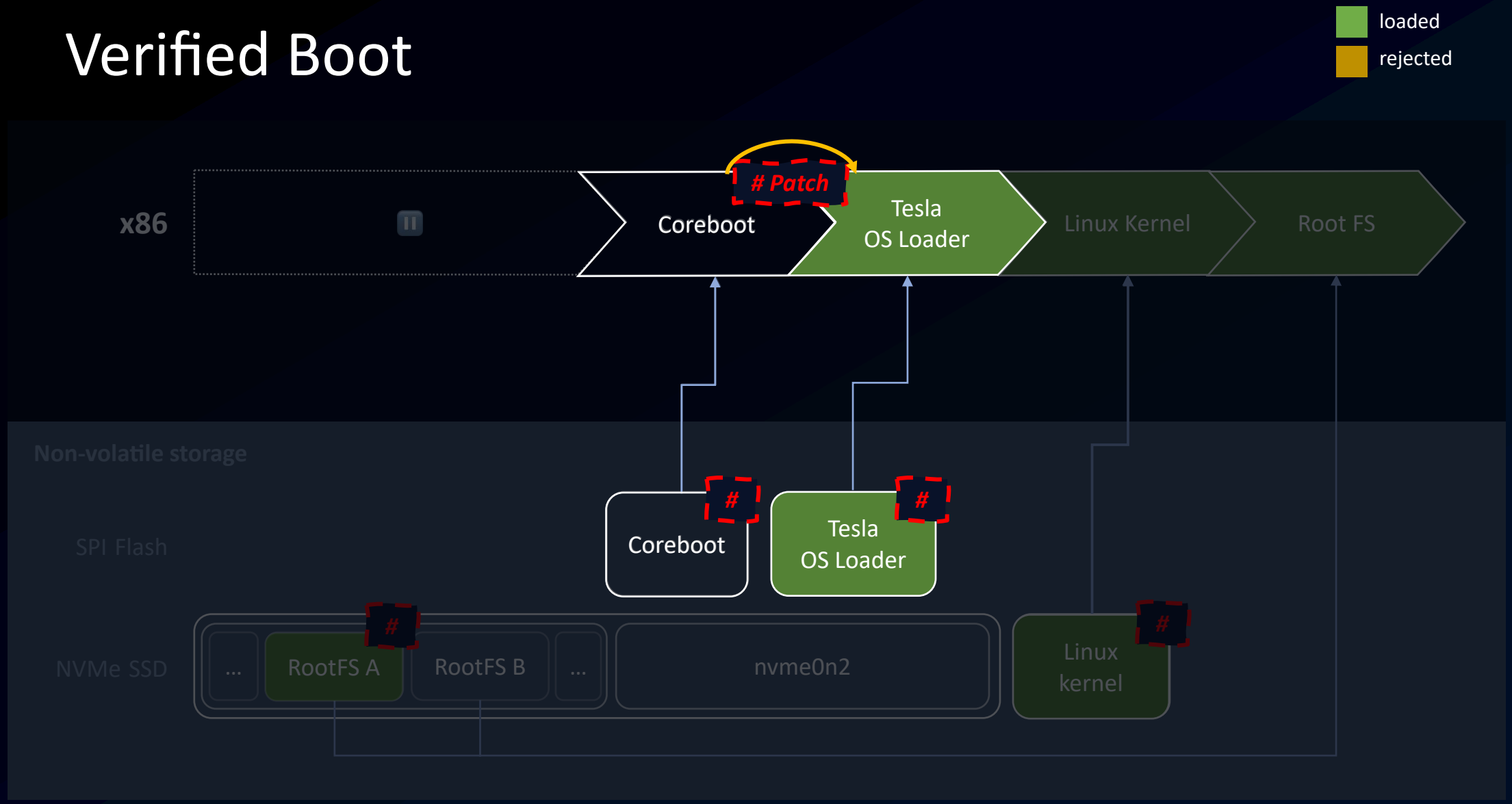
Verified Boot



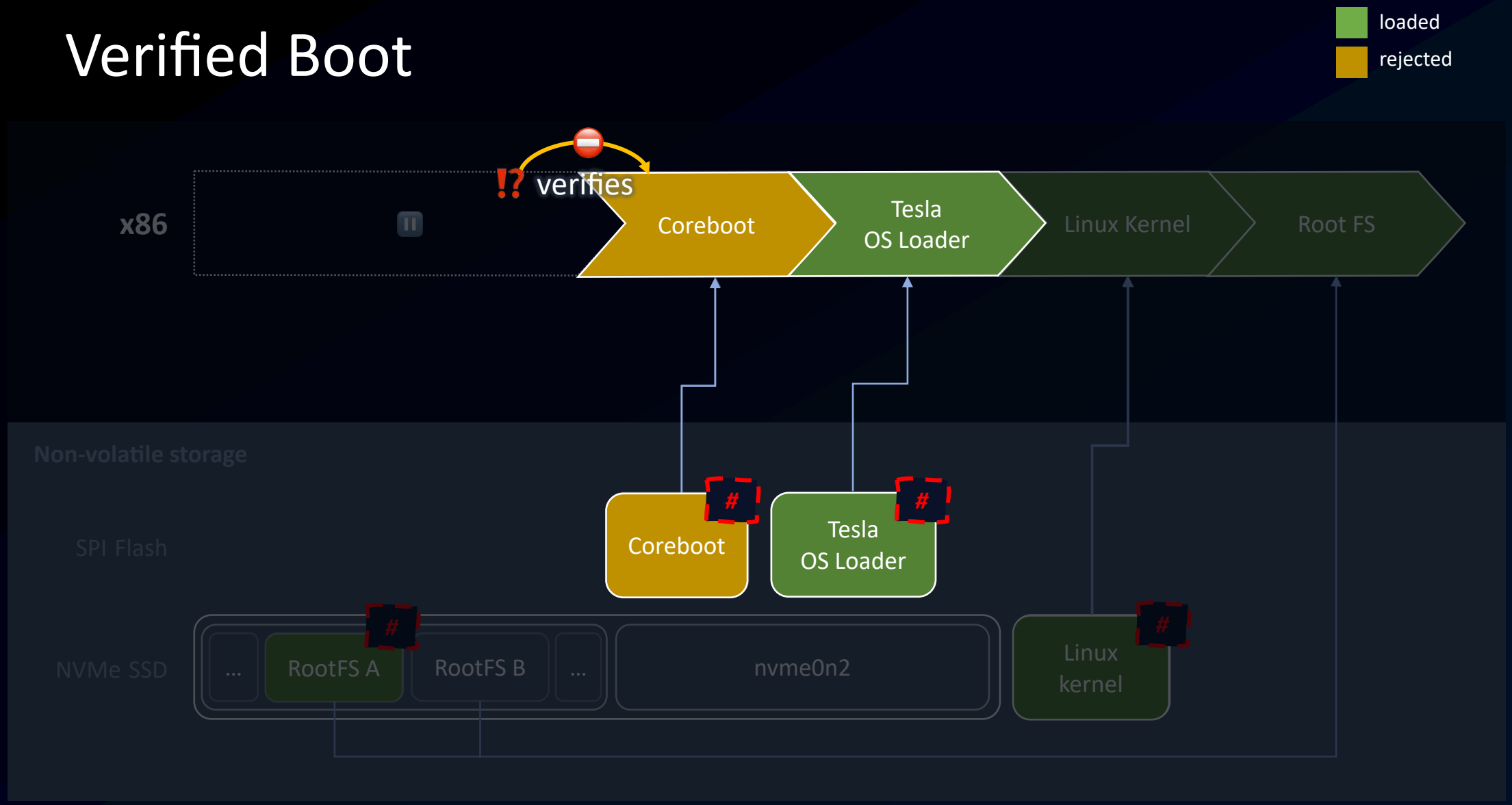
Verified Boot



Verified Boot

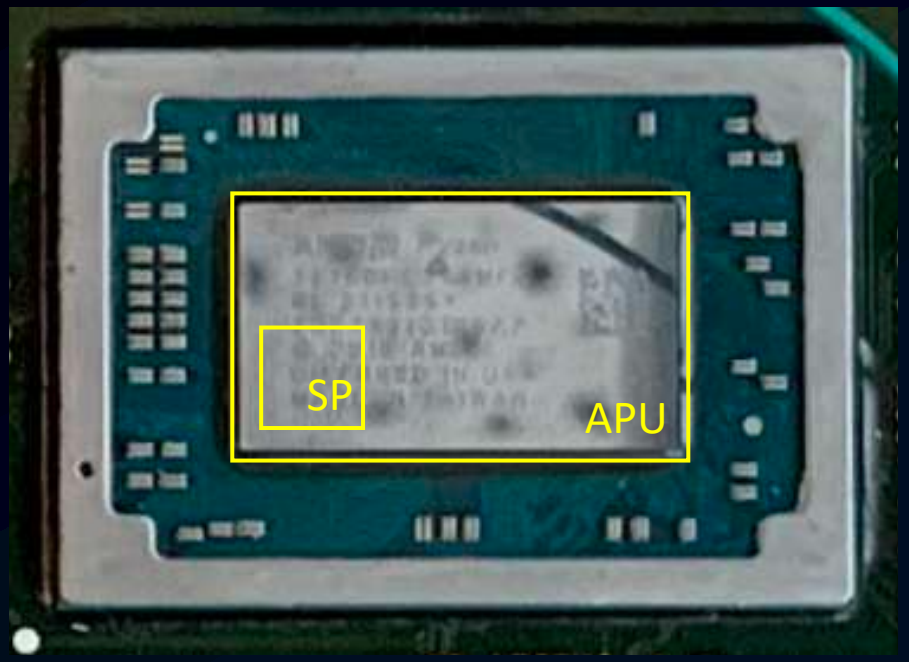


Verified Boot

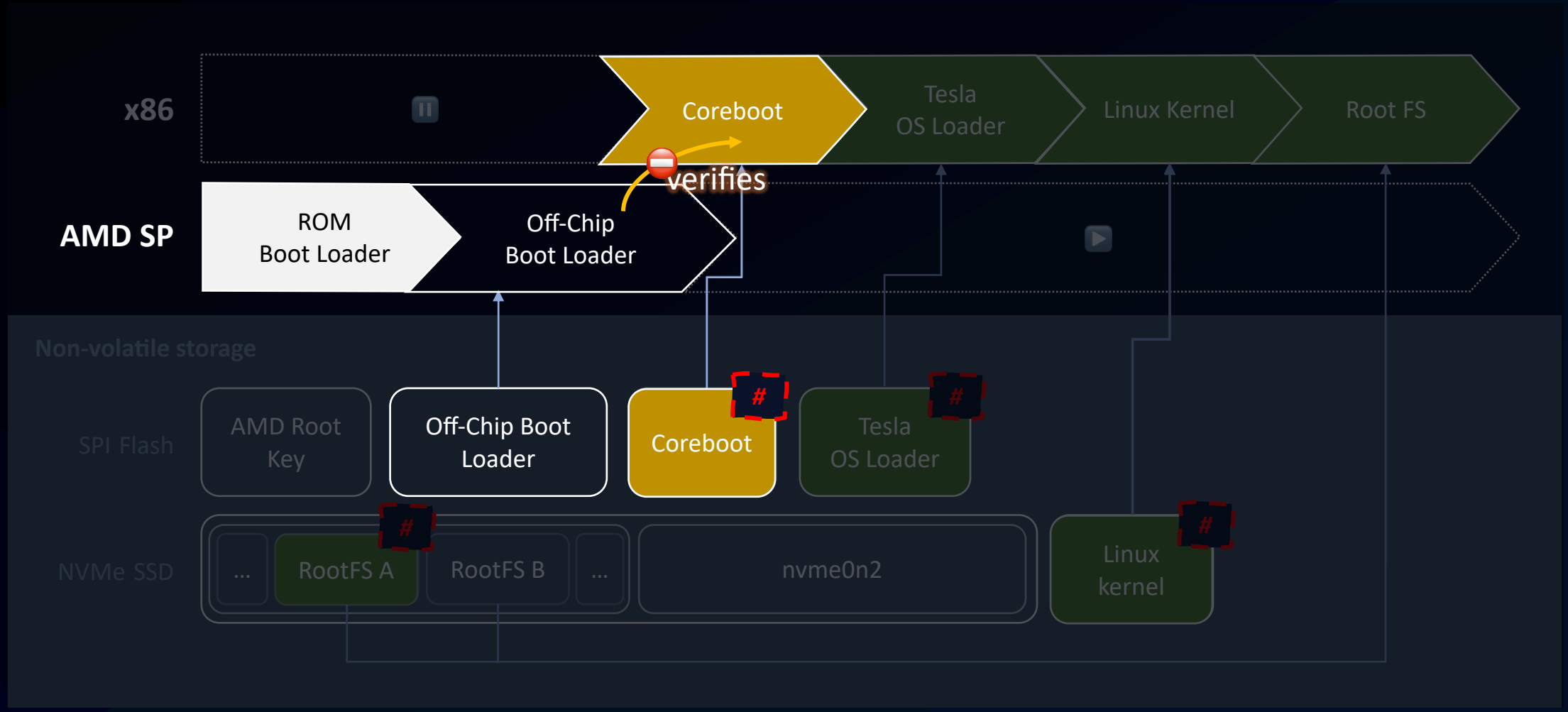


AMD Secure Processor

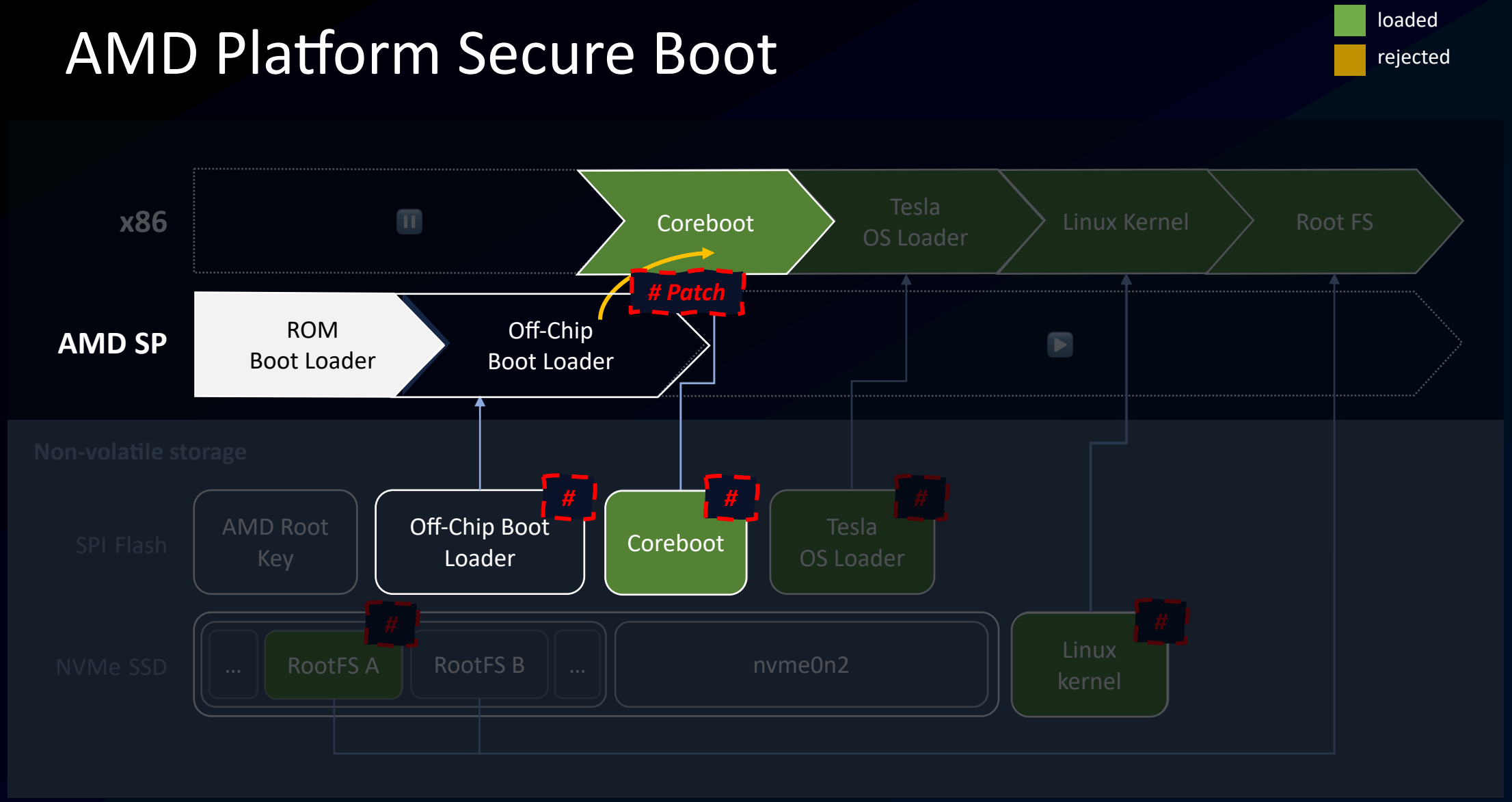
- ARMv7 µController
- Integrated into CPU SoC
- Highly privileged
- Variety of responsibilities
  - Hardware root of trust
  - Firmware TPM (fTPM) for key management and more
  - (On EPYC Servers) Secure Encrypted Virtualization



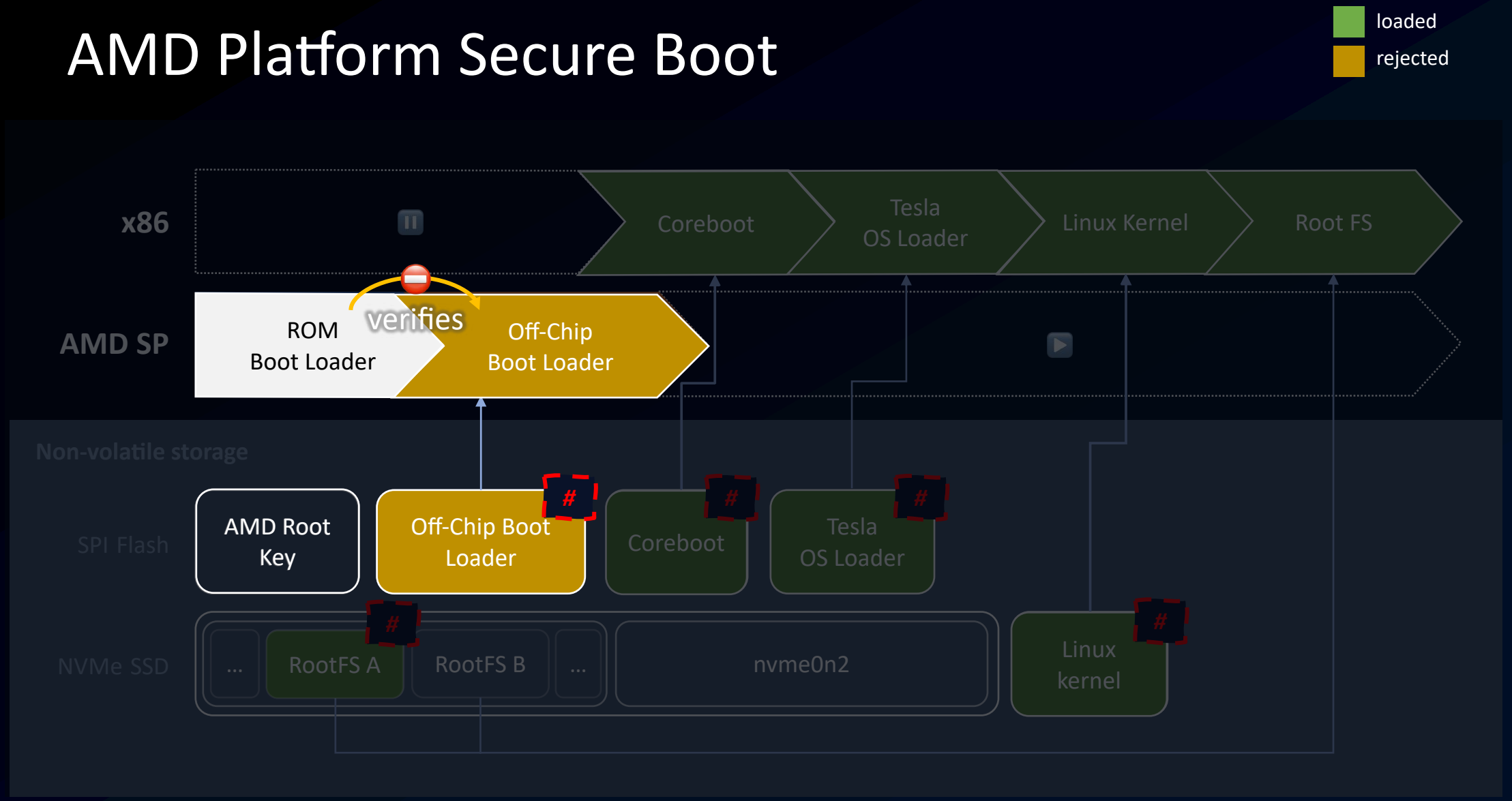
AMD Platform Secure Boot



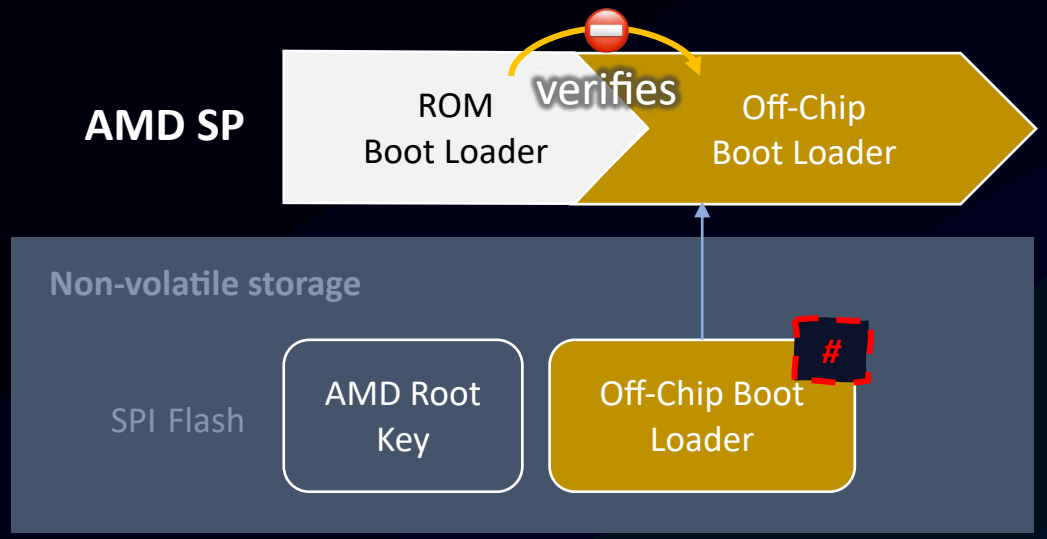
AMD Platform Secure Boot



AMD Platform Secure Boot



Previous AMD SP Vulnerabilities



- 2019: Off-Chip Boot Loader Buffer overflow
  - Arbitrary Code Execution ✅
  - **Fixed via firmware updates**
- 2020: ROM Boot Loader Buffer overflow
  - Arbitrary Code Execution ✅
  - Not fixable (ROM)
  - Fixed in new generations (>= Zen 2)
  - **Fixes backported to Tesla's Zen 1 APU**

Tesla’s Security Evolution

2014

- Open X servers
- Hardcoded passwords
- Diagnostic Ethernet: root
- No code signing

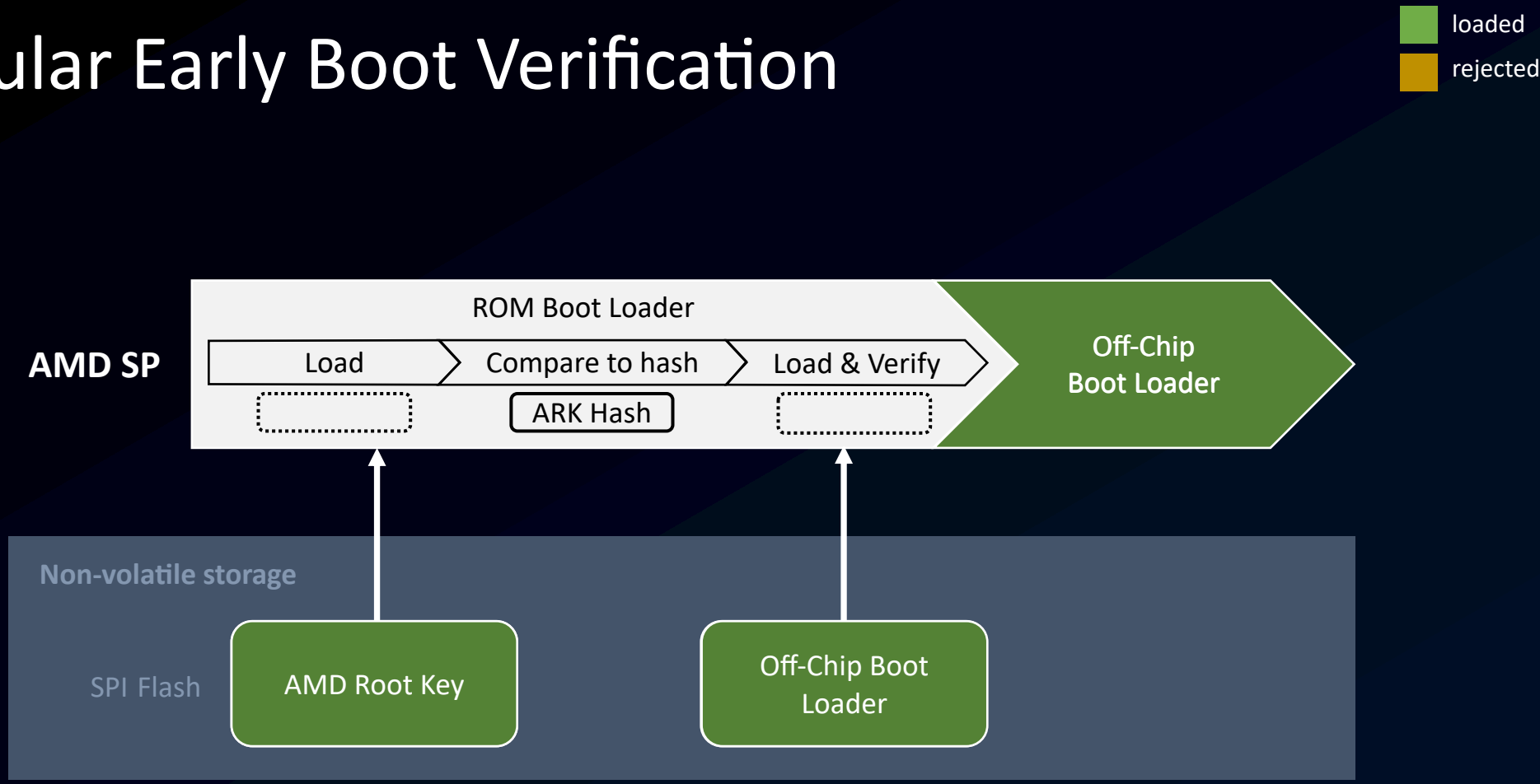
2023

- Firmware and OS signing
- Chain of trust during boot
- **Root of trust in AMD SoC**

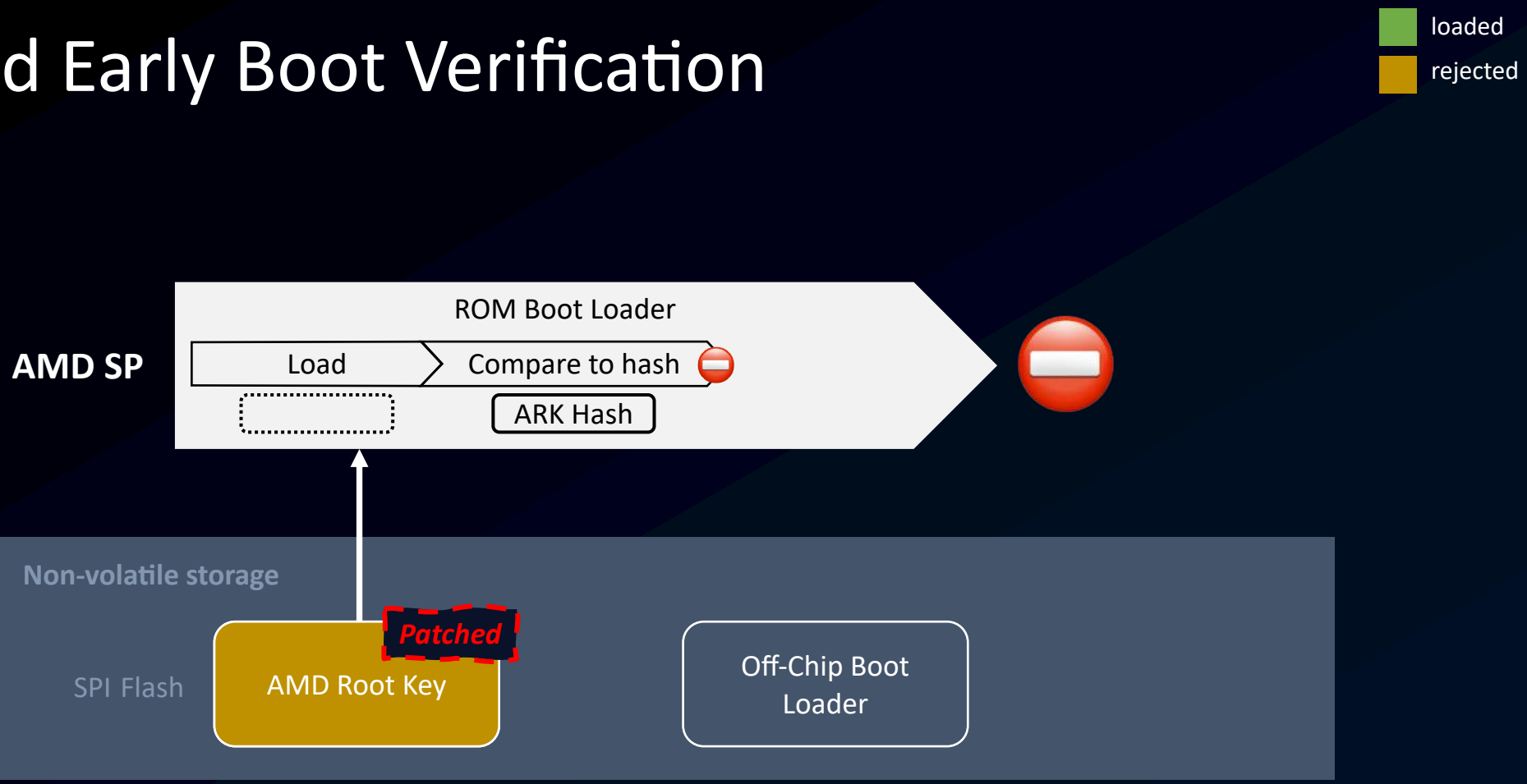
Outline

1. Analyzing Boot and Firmware Security
2. Hotwiring the Infotainment system
3. Extracting Secrets from the Tesla

Regular Early Boot Verification



Failed Early Boot Verification



Fault Injection Attacks

Induce fault by altering the IC's environment:

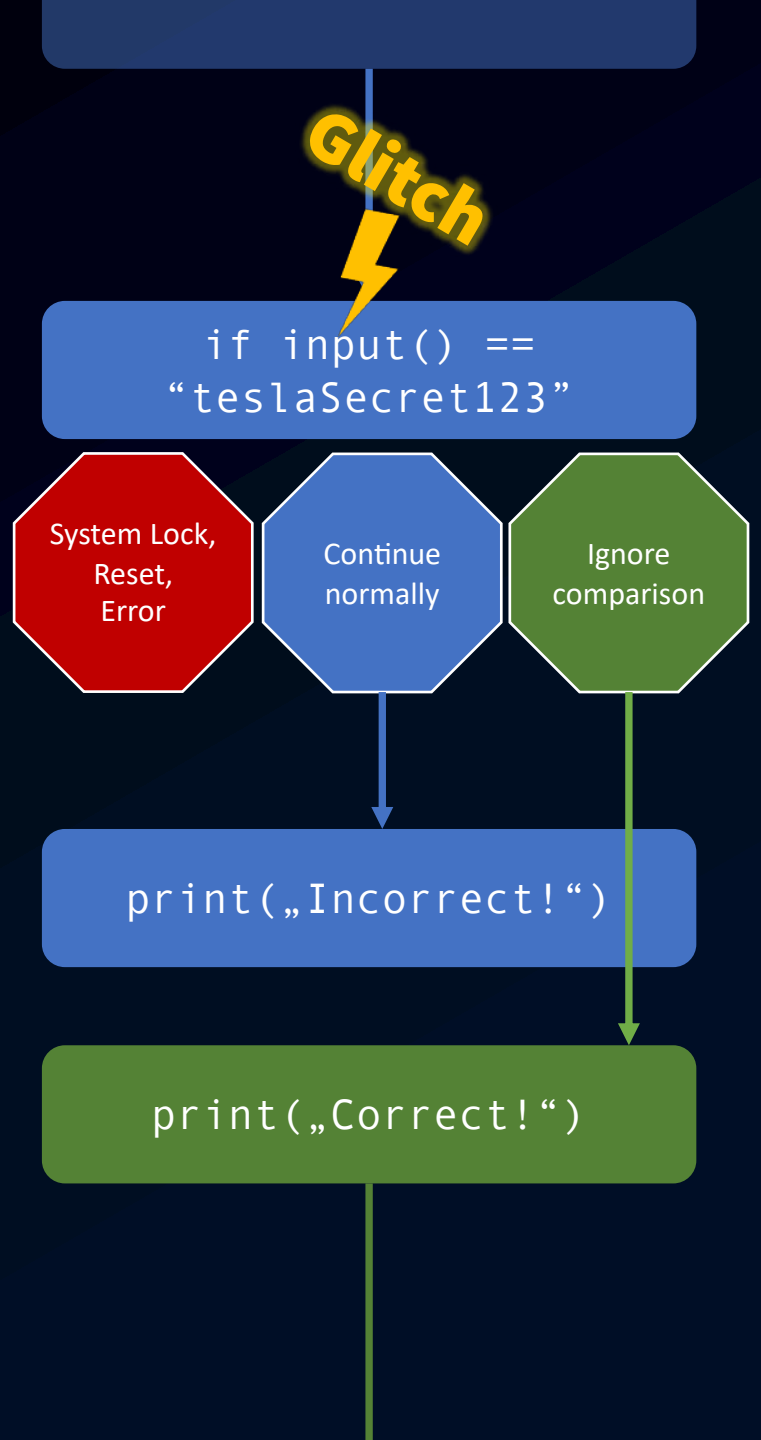
- Laser, electromagnetic-radiation, clock, supply voltage

Voltage Glitching:

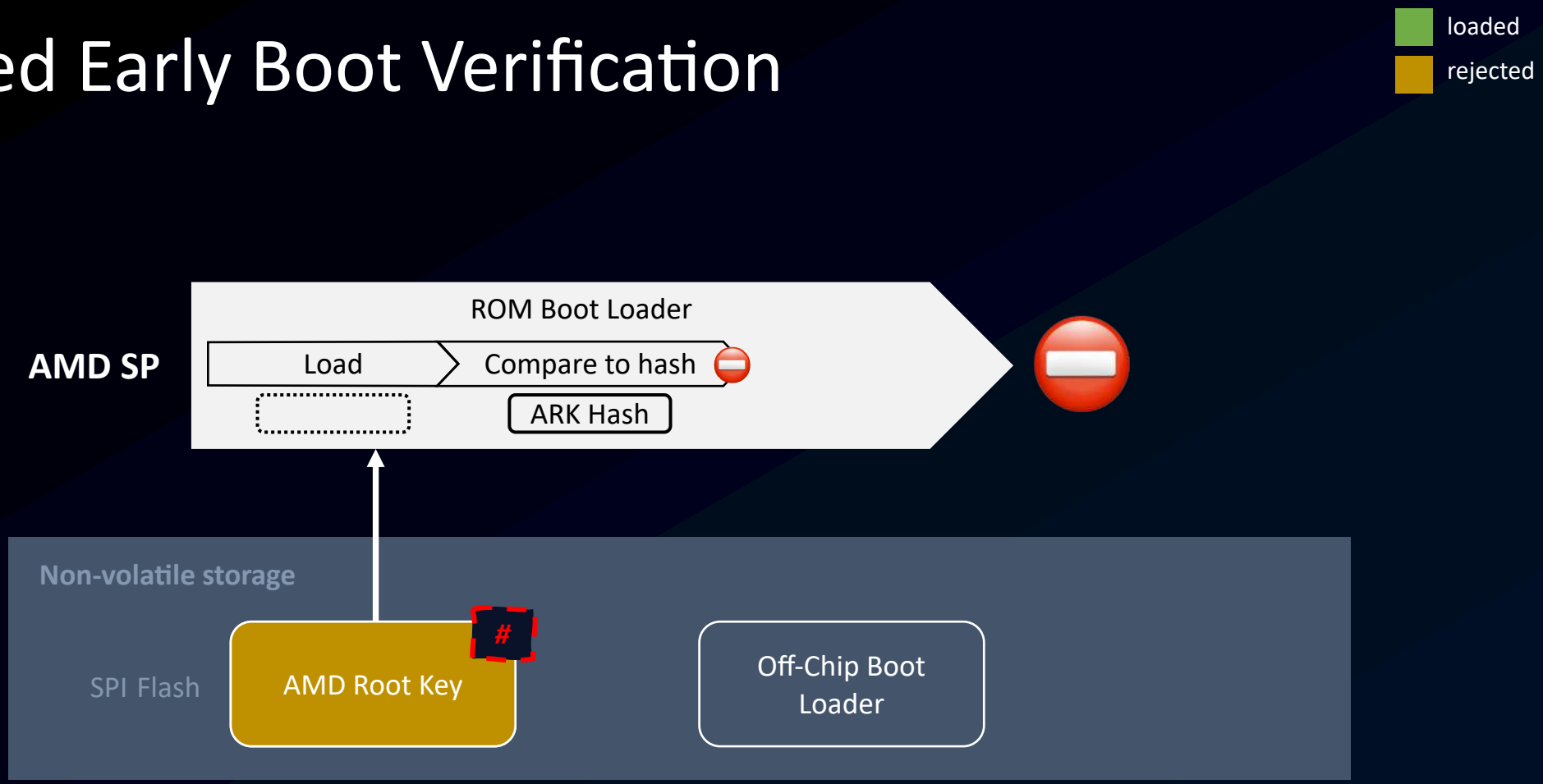
- Lowering voltage shortly

Key Challenges

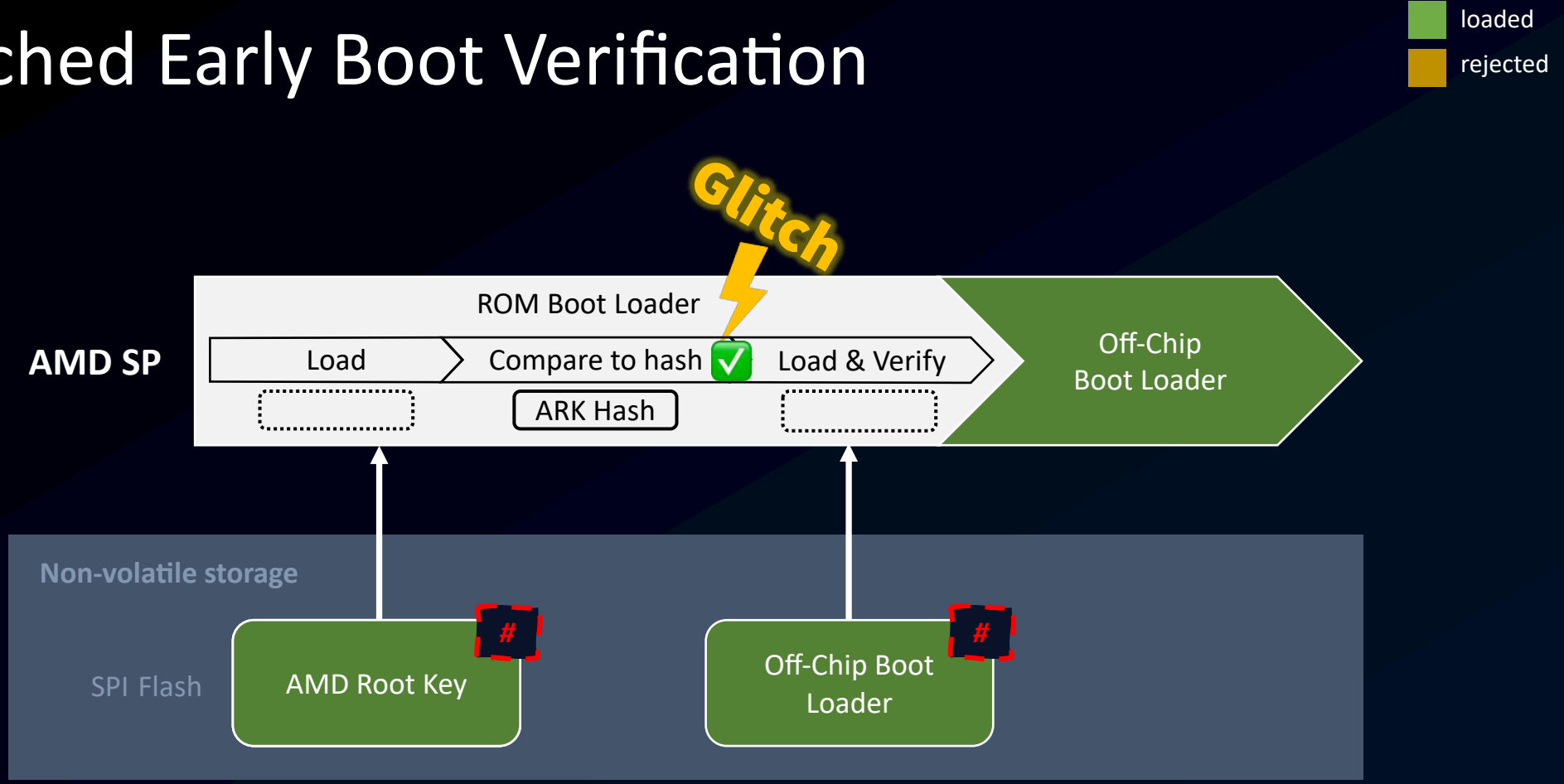
- **Most faults are “useless”**
- **Trigger:**
  Figure out when targeted check happens
- **Parameters:**
  Voltage drop steepness, width, minimum
- **Reset/Success:**
  Identify failed attacks and retry as fast as possible



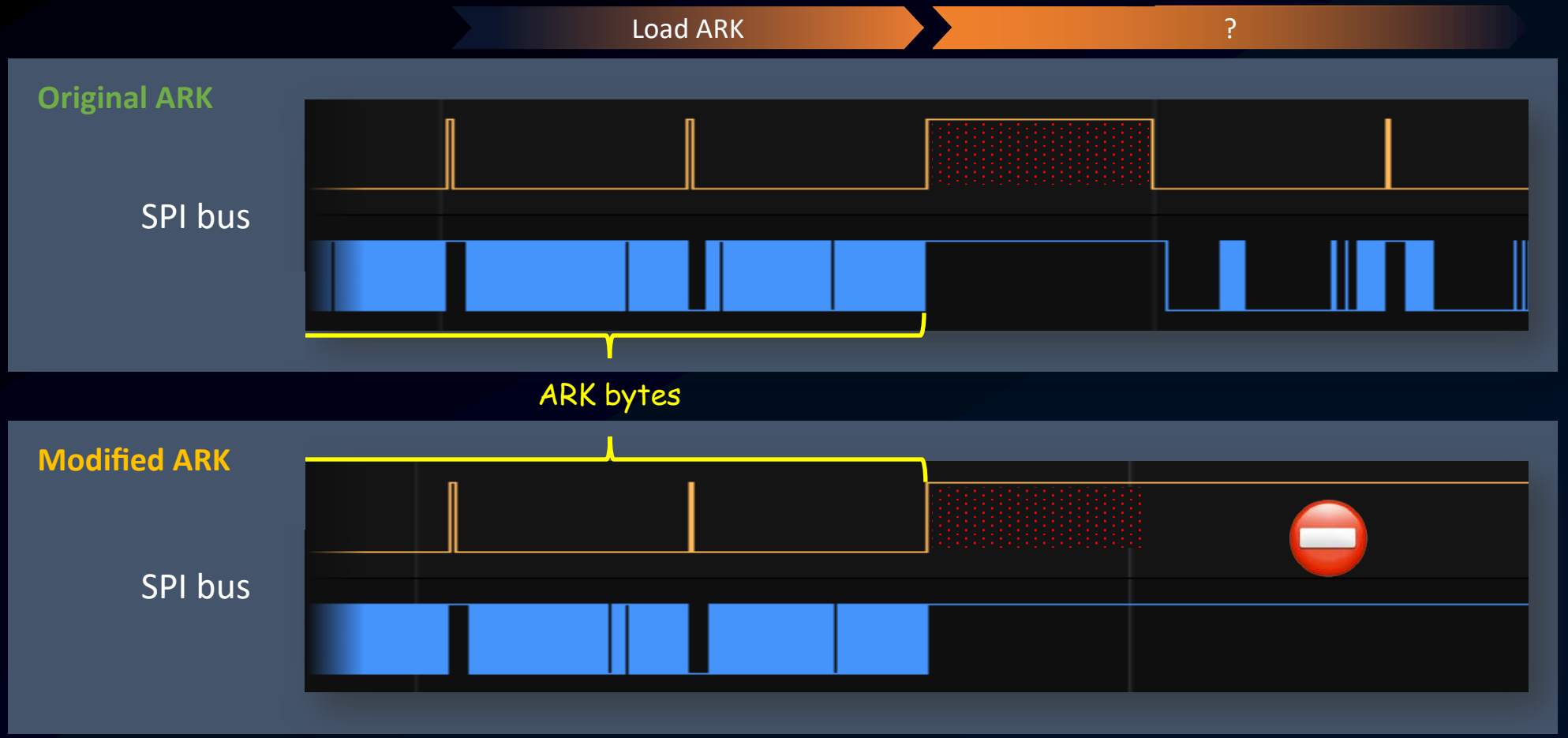
Failed Early Boot Verification



Glitched Early Boot Verification

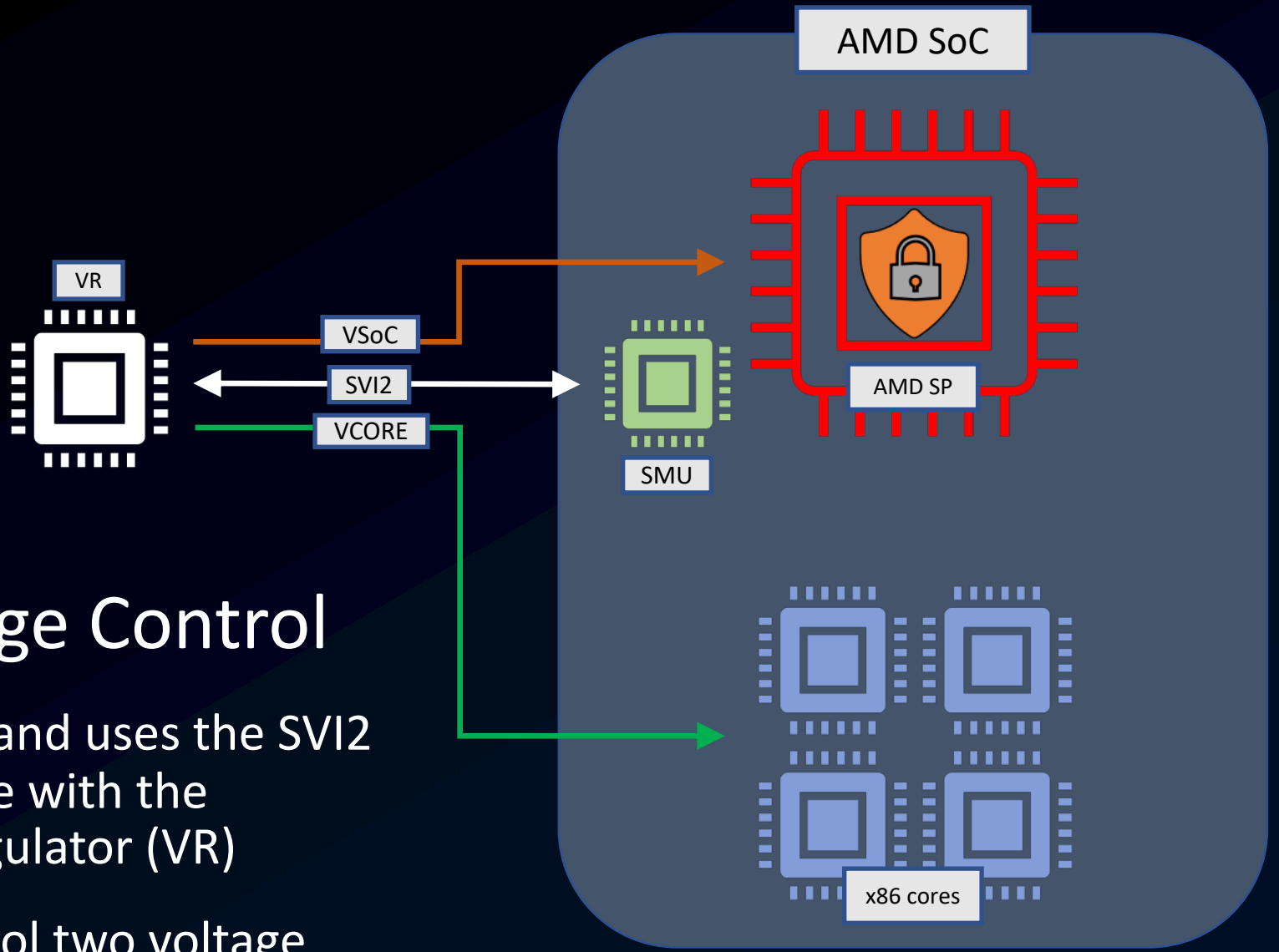


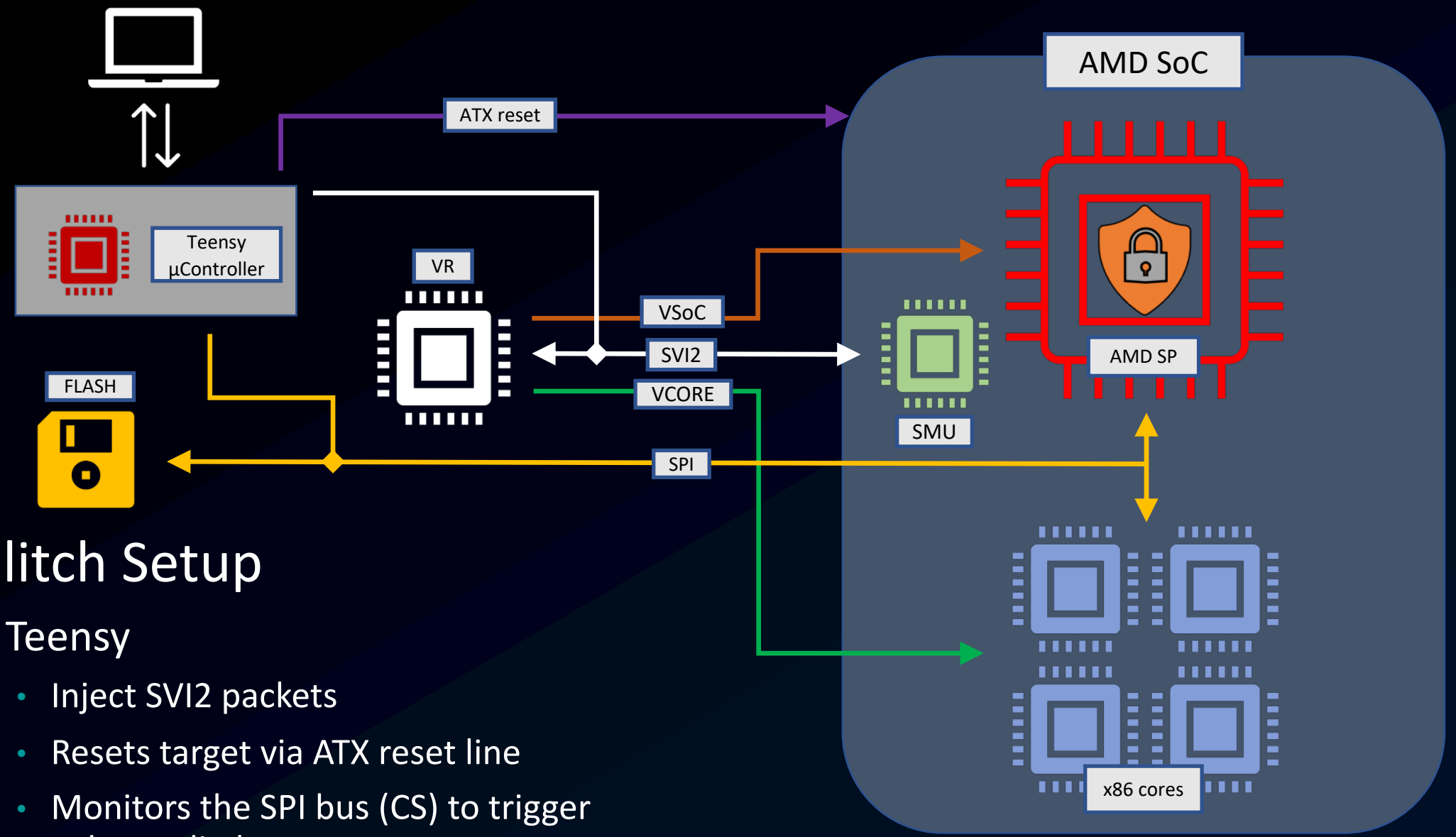
Finding the ARK Verification Time Window



Dynamic Voltage Control

- SMU monitors SoC and uses the SVI2 bus to communicate with the external voltage regulator (VR)
- SVI2 allows to control two voltage domains per VR





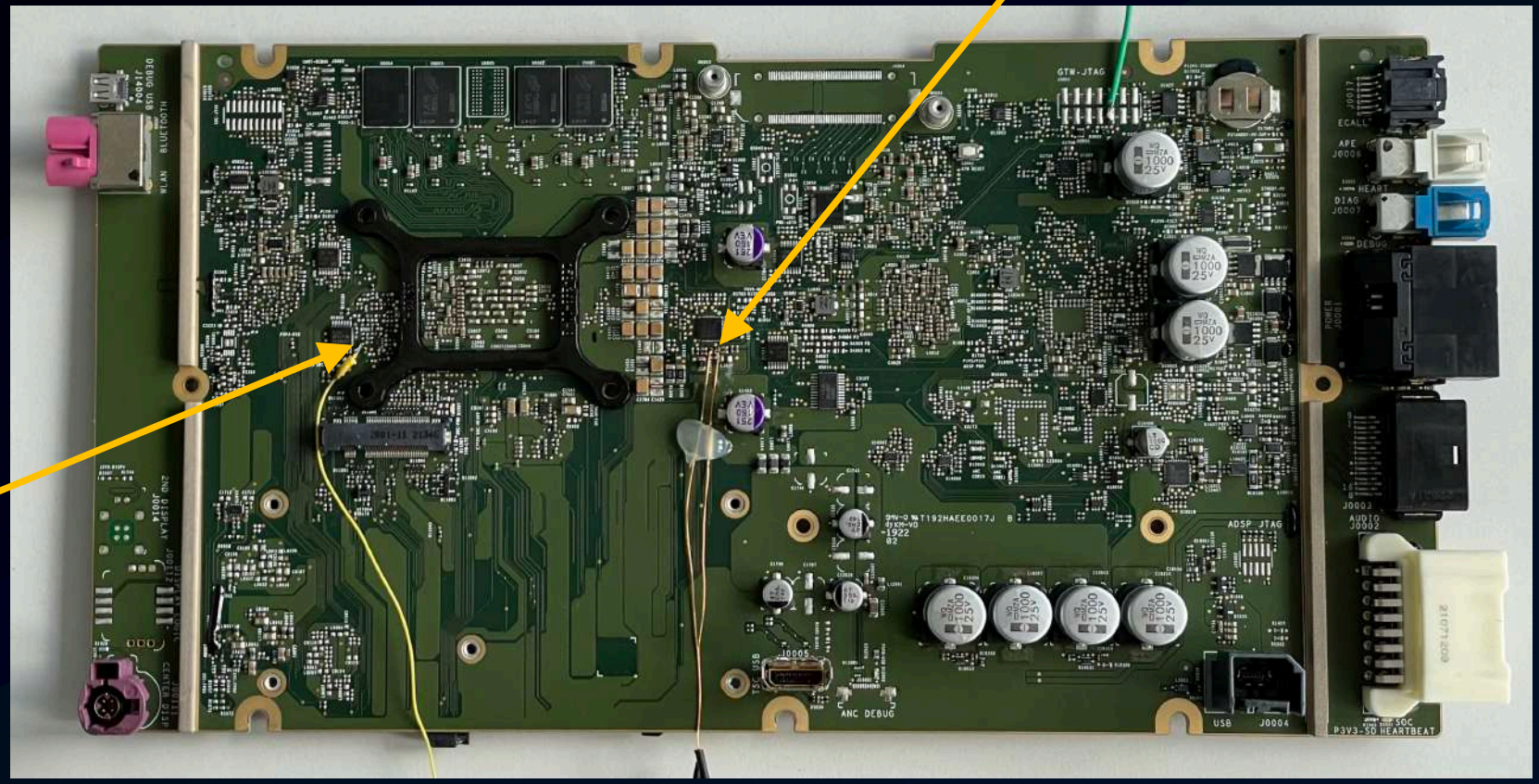
Glitch Setup

- Teensy
  - Inject SVI2 packets
  - Resets target via ATX reset line
  - Monitors the SPI bus (CS) to trigger voltage glitch
- External PC controls glitch parameters

Voltage Glitch Wiring

SVI2 bus (SVD + SVC)

SPI chip-select



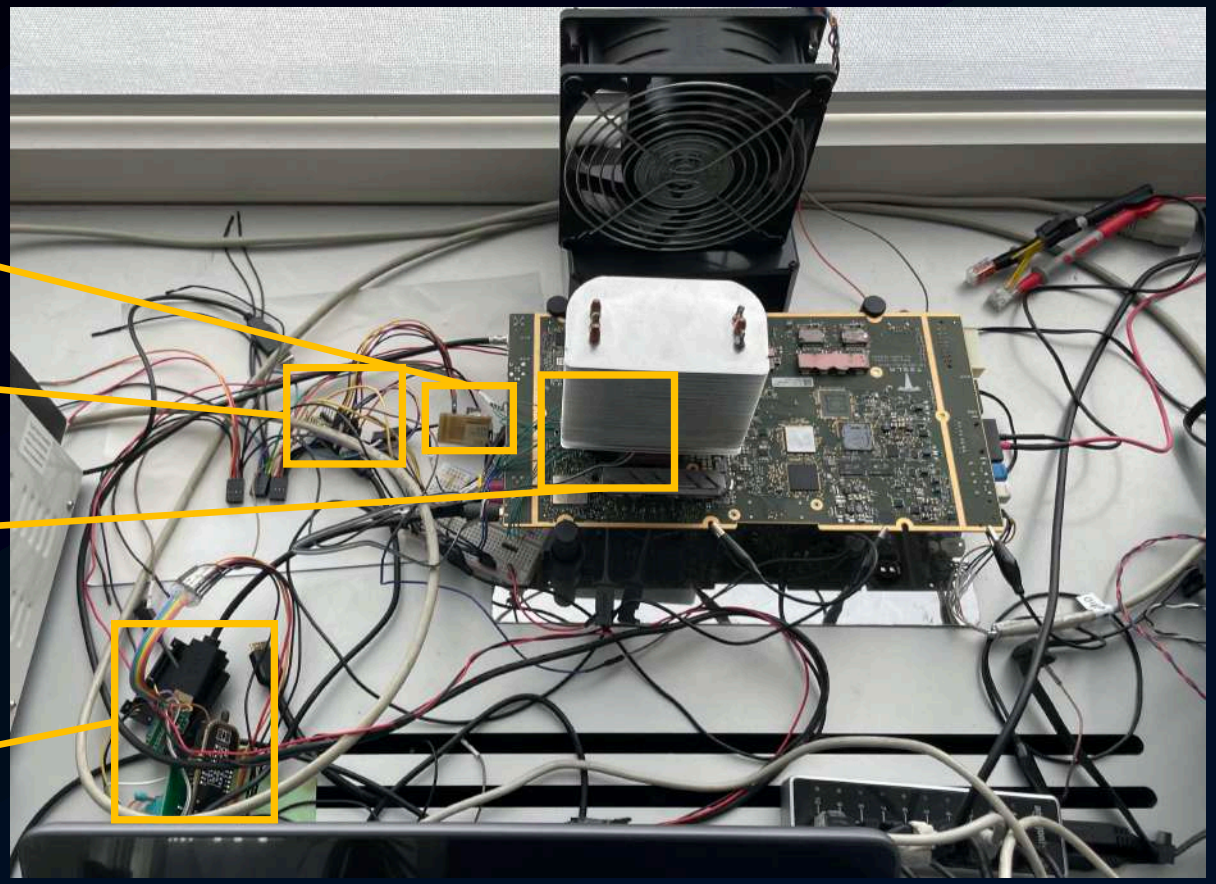
Glitch Setup in Reality

SVI2 bus

Teensy
µController

SPI bus

ATX reset
SPI programmer
Serial output



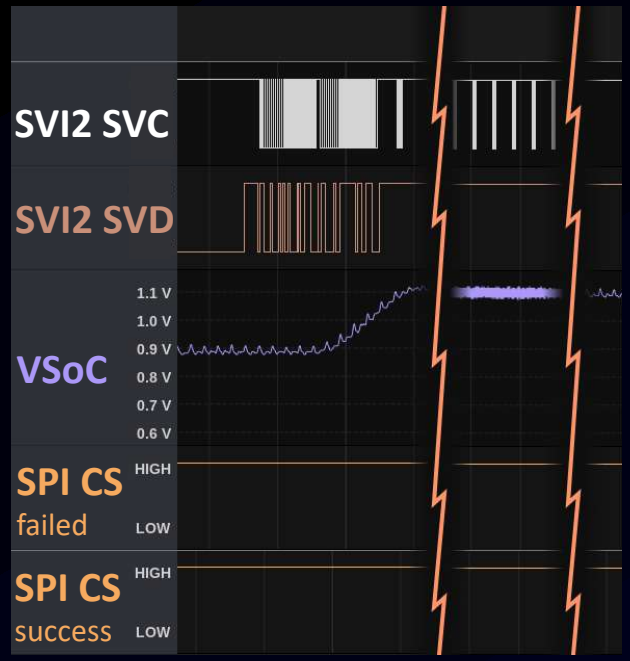
Voltage Glitch Steps

- SVI2 SVC: bus clock
- SVI2 SVD: bus data
- VSoC: target’s voltage
- SPI CS: chip-select signal

Voltage Glitch Steps

- SoC sets initial voltage
- SVD rising edge triggers attack logic
- VSoC rises

Voltage Glitch Steps

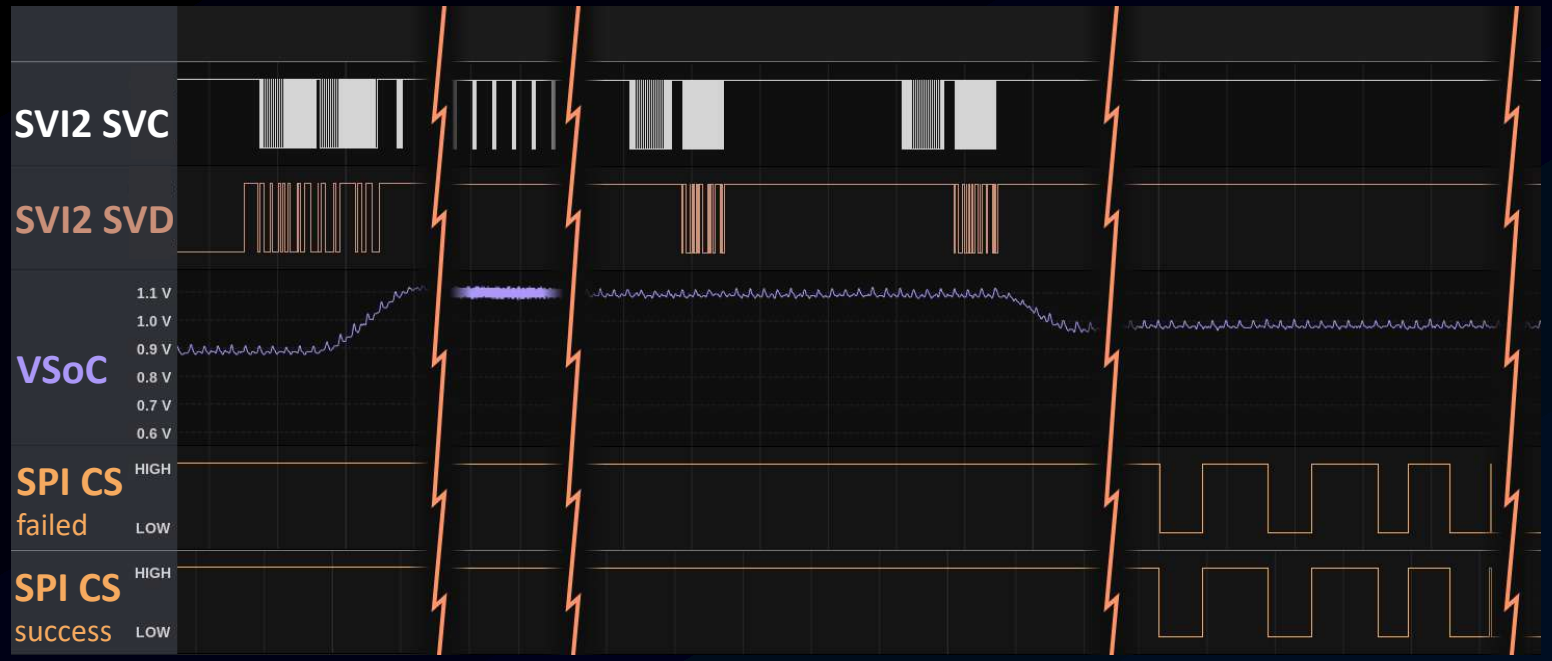


- VR sends telemetry packets
- VSoC stable

Voltage Glitch Steps

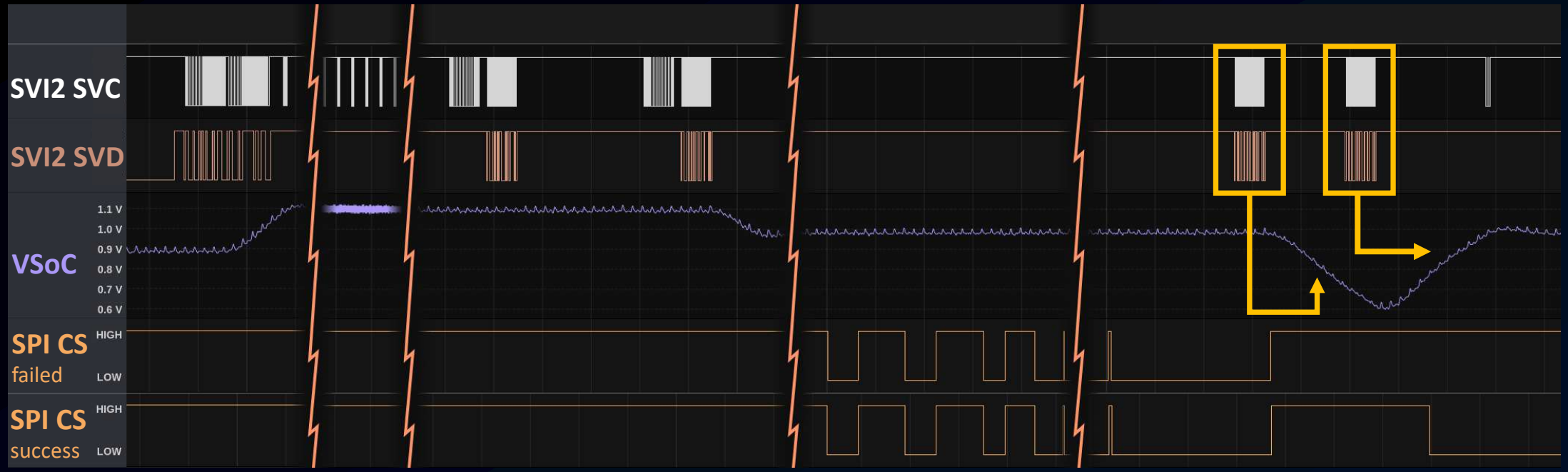
- Teensy injects SVI2 packets
- Disable telemetry to avoid collisions
- VSoC is adjusted

Voltage Glitch Steps



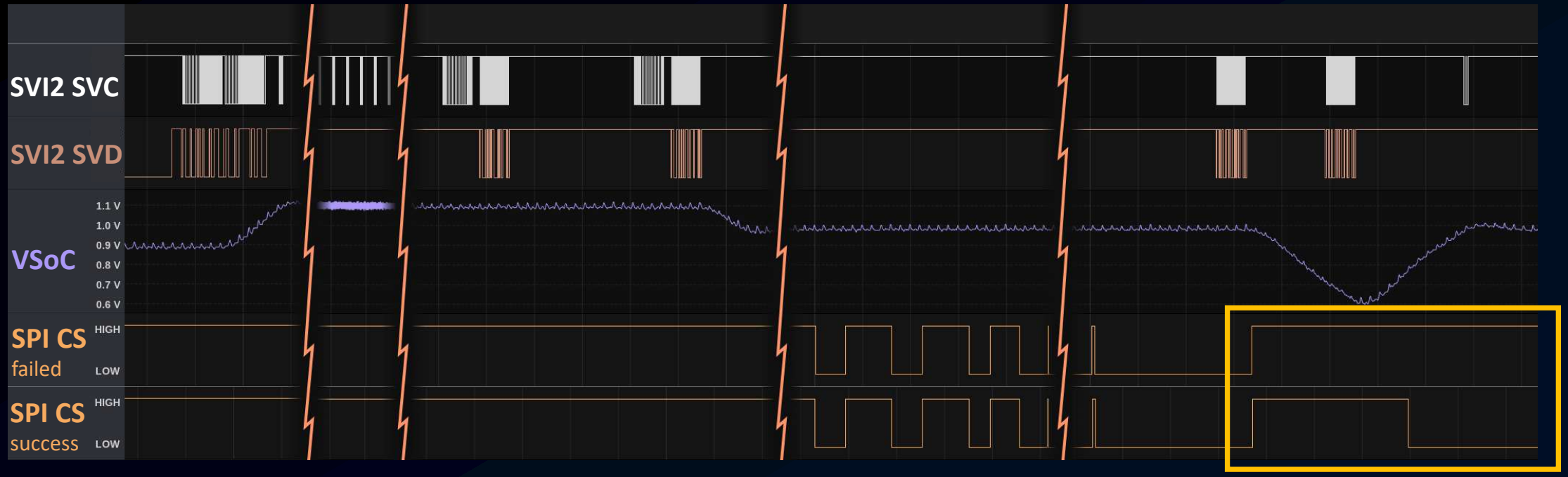
- Teensy starts counting CS edges to trigger glitch on time
- CS becomes active → AMD SP loads data

Voltage Glitch Steps



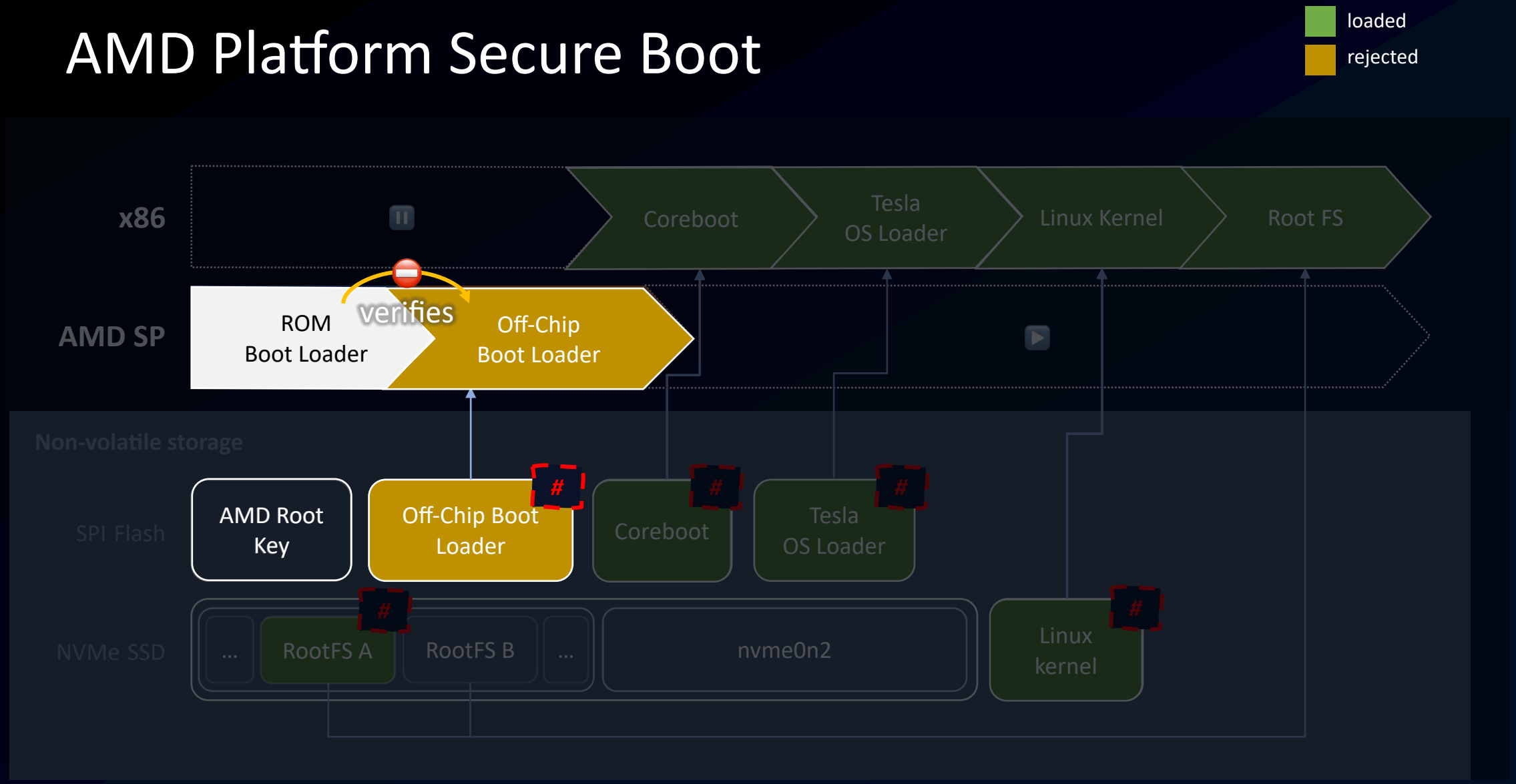
- Teensy injects two SVI2 packets to create voltage disturbance
- Voltage drop on VSoC (glitch)

Voltage Glitch Steps

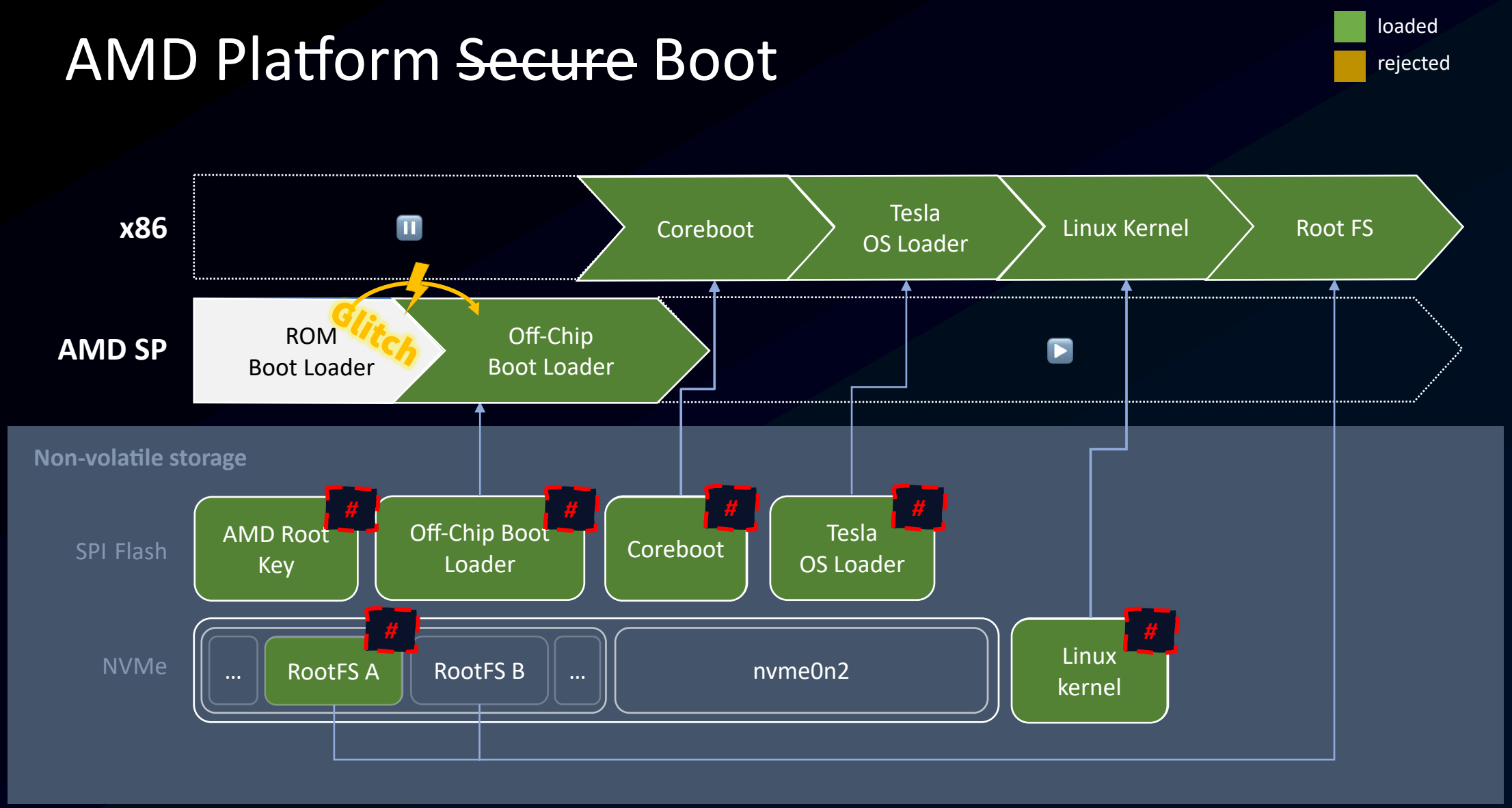


- Teensy monitors CS to detect success
- Teensy resets target on fail
- CS inactive (high) → failed attempt
- CS active (low) → successful attempt

AMD Platform Secure Boot



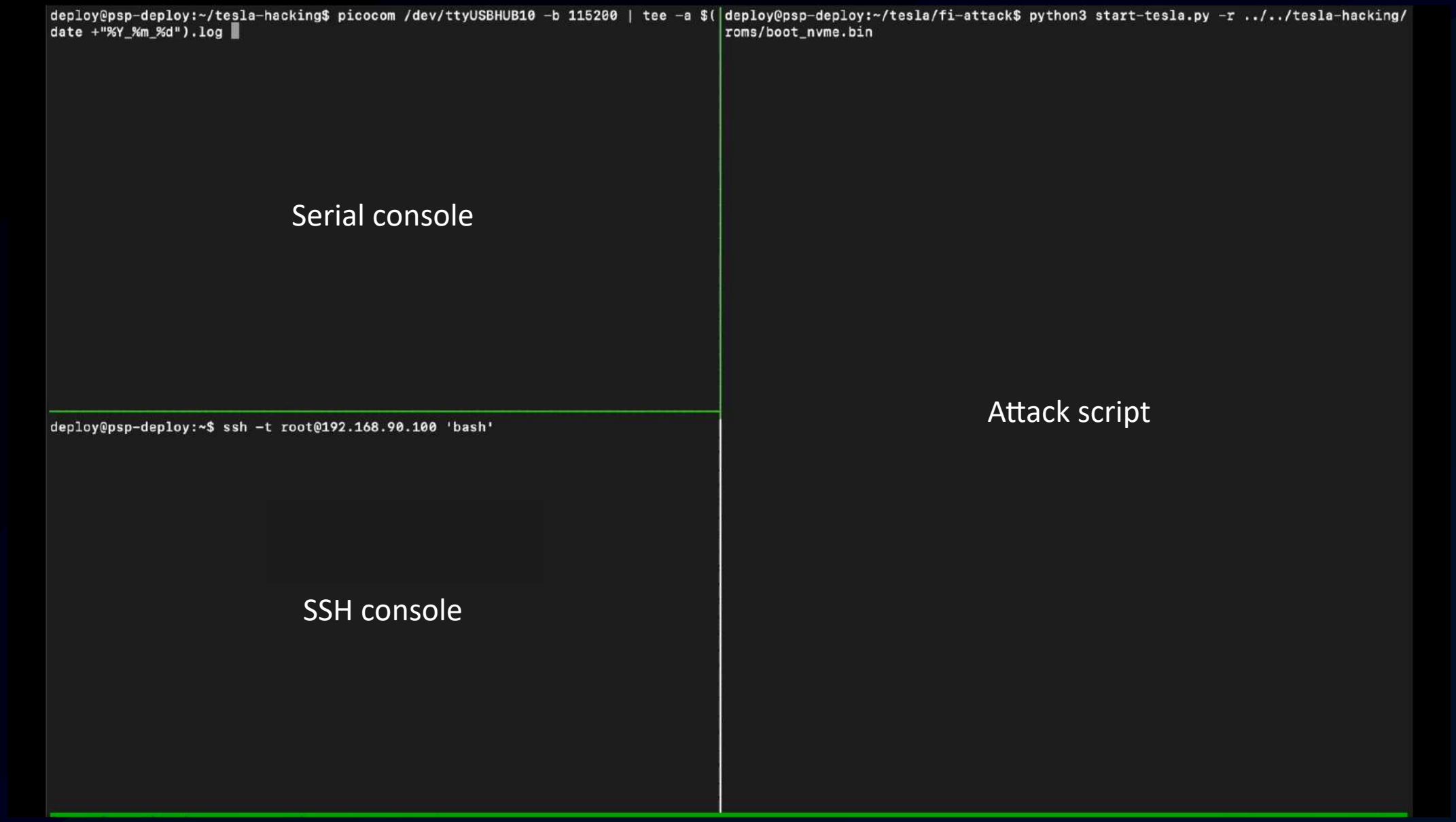
AMD Platform ~~Secure~~ Boot



Trying to Activate the Rear Seat Heaters

Finding their Configuration ID

```
{
  "accessId": 13,
  "codeKey": "rearSeatHeaters",
  "content": {
    "enums": [
      {
        "codeKey": "NONE",
        "description": "None",
        "value": 0
      },
      {
        "codeKey": "KONGSBERG_LOW_POWER",
        "description": "Kongsberg low-power heaters",
        "value": 1
      }
    ]
  },
  "description": "Type of rear seat heaters installed",
  "products": [
    "Model3",
    "ModelY"
  ]
}
```


deploy@psp-deploy:~/tesla-hacking$ picocom /dev/ttyUSBHUB10 -b 115200 | tee -a $(
date +"%Y_%m_%d").log
Serial console
deploy@psp-deploy:~/tesla/fi-attack$ python3 start-tesla.py -r ../../tesla-hacking/
roms/boot_nvme.bin
Attack script
deploy@psp-deploy:~$ ssh -t root@192.168.90.100 'bash'
SSH console

~~Trying to~~ Activate the Rear Seat Heaters

What About Persistence?

- Sorry, voltage glitch is not persistent
  - Need to glitch on every Infotainment boot
  - But the car configuration survives regular infotainment (re)boot
  - And Infotainment supposedly doesn't reboot very often
- Glitching could be made even smoother by a mod chip/PCB
  - Implementation detail … 🙃
  - *We leave this as an exercise to the interested audience*

Secure Configuration Items

- Demo possible since the rear seat heaters were an “insecure configuration item” in our Gateway firmware version
  - ”Secure configuration items” can only be changed with a valid signature
- “**Rear seat heaters were upgraded to be a signed configuration starting in the 2022.44 release**”, Tesla told us
- So being root on the Infotainment is not sufficient
  - Software or hardware vulnerability in Gateway necessary

ZERO DAY INITIATIVE

PRIVACY WHO WE ARE HOW IT WORKS BLOG ADVISORIES LOG IN | SIGN UP

ADVISORY DETAILS

July 18th, 2023

(Pwn2Own) Tesla Model 3 Gateway Firmware Signature Validation Bypass Vulnerability

ZDI-23-972
ZDI-CAN-20734

CVE ID	CVE-2023-32156
CVSS SCORE	9.0, (AV:A/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H)
AFFECTED VENDORS	Tesla
AFFECTED PRODUCTS	Model 3
VULNERABILITY DETAILS	This vulnerability allows network-adjacent attackers to execute arbitrary code on affected Tesla Model 3 vehicles. An attacker must first obtain the ability to execute privileged code on the Tesla infotainment system in order to exploit this vulnerability. The specific flaw exists within the handling of firmware updates. The issue results from improper error-handling during the update process. An attacker can leverage this vulnerability to execute code in the context of Tesla's Gateway ECU.
ADDITIONAL DETAILS	Fixed in 2023.12 firmware release.

Outline

1. Analyzing Boot and Firmware Security
2. Hotwiring the Infotainment system
3. Extracting Secrets from the Tesla

What secrets are there on the Tesla?

CAR CREDENTIALS

- Authenticates car against Tesla servers (Tesla's car VPN)
  - Firmware updates
  - Car configuration
- Bound to Vehicle Identification Number (VIN)
- Used to remotely (de-)authorize services

USER DATA

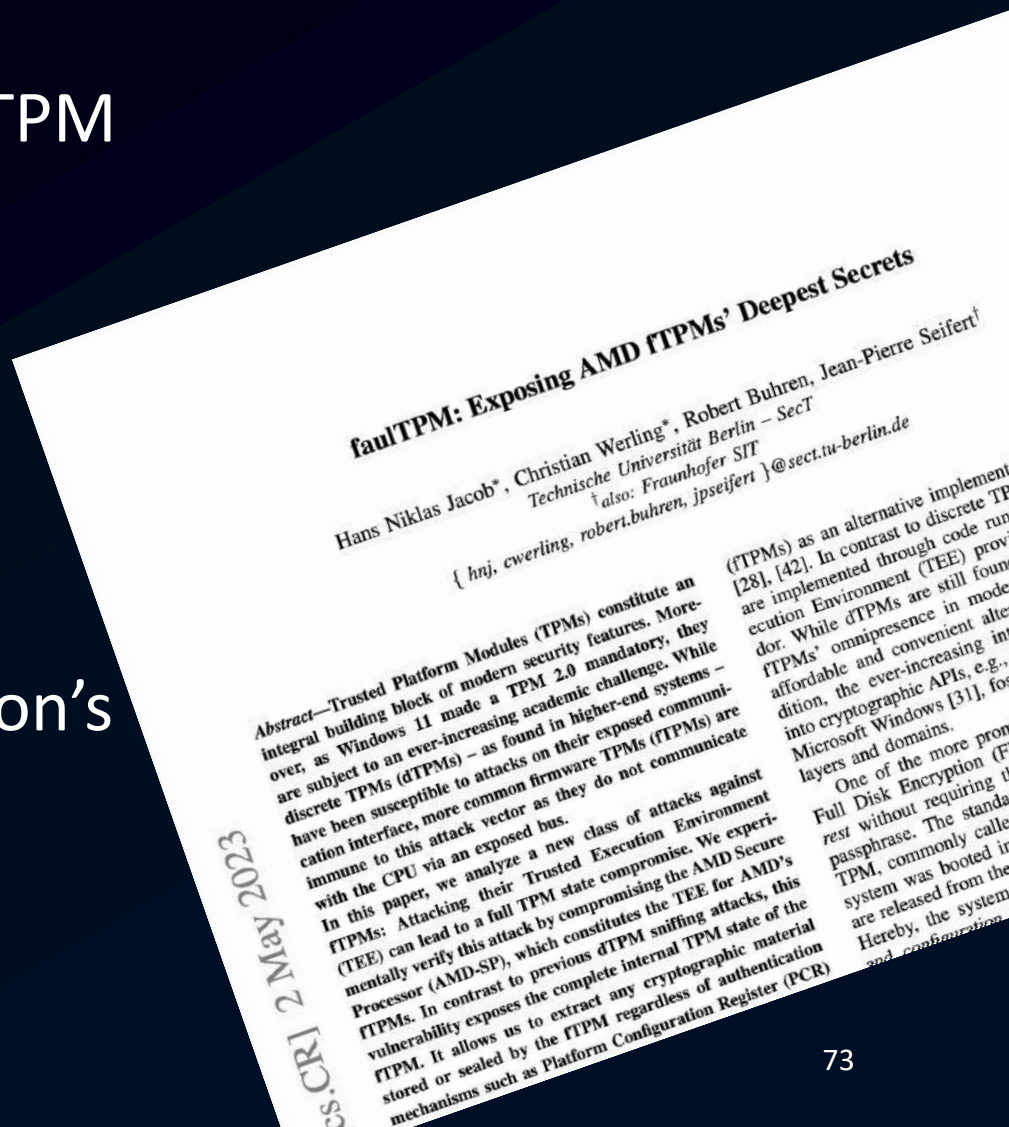
- Phones connected via Bluetooth
  - Contacts, calendar, call logs ...
- Locations visited
- WiFi passwords
- Spotify and Gmail session cookies

How are these secrets secured?

- Everything used to be cleartext
  - Car Credentials on SD card, on storage
  - User data on cleartext storage partition
- Now there is TPM-based security
  - Car Creds sealed in TPM
  - User data partition encrypted, key sealed in TPM

What we extracted

- We wrote a paper on attacking AMD's fTPM
  - Extracting the TPM's internal state
  - ***Unsealing arbitrary TPM objects***
- We extracted the car credentials
  → giving us access to Tesla's server endpoints meant for cars
- We extracted the encrypted user partition's disk encryption keys
  → we have access to user data


faulTPM: Exposing AMD fTPMs' Deepest Secrets

Hans Niklas Jacob*, Christian Werling*, Robert Buhren, Jean-Pierre Seifert†

Technische Universität Berlin – SecT

†also: Fraunhofer SIT

{hnj, cwerling, robert.buhren, jpseifert}@sect.tu-berlin.de

Abstract—Trusted Platform Modules (TPMs) constitute an integral building block of modern security features. Moreover, as Windows 11 made a TPM 2.0 mandatory, they are subject to an ever-increasing academic challenge. While discrete TPMs (dTPMs) – as found in higher-end systems – have been susceptible to attacks on their exposed communication interface, more common firmware TPMs (fTPMs) are immune to this attack vector as they do not communicate with the CPU via an exposed bus.

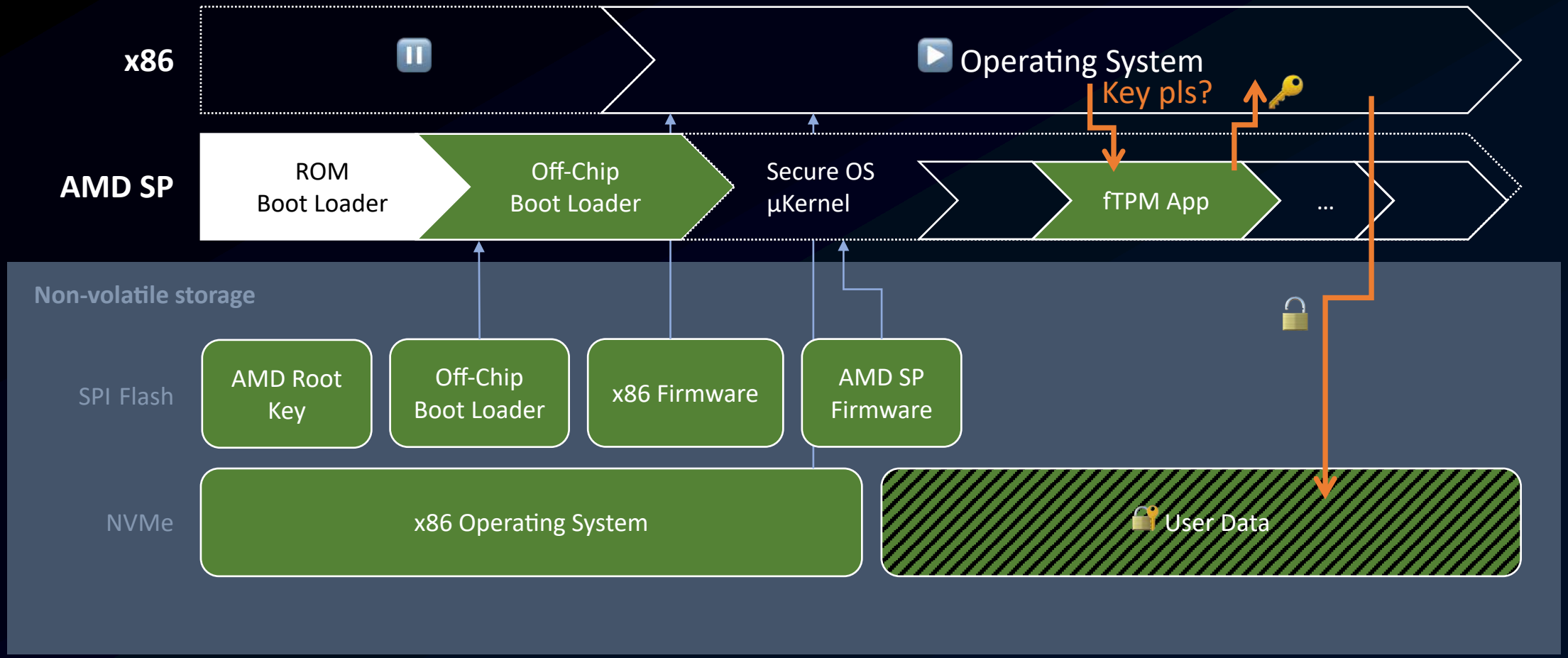
In this paper, we analyze a new class of attacks against fTPMs: Attacking their Trusted Execution Environment (TEE) can lead to a full TPM state compromise. We experimentally verify this attack by compromising the AMD Secure Processor (AMD-SP), which constitutes the TEE for AMD's fTPMs. In contrast to previous dTPM sniffing attacks, this vulnerability exposes the complete internal TPM state of the fTPM. It allows us to extract any cryptographic material stored or sealed by the fTPM regardless of authentication mechanisms such as Platform Configuration Register (PCR)

(fTPMs) as an alternative implement
[28], [42]. In contrast to discrete TP
are implemented through code run
ecution Environment (TEE) provi
dor. While dTPMs are still foun
fTPMs' omnipresence in mode
affordable and convenient alte
dition, the ever-increasing int
into cryptographic APIs, e.g.,
Microsoft Windows [31], fo
layers and domains.

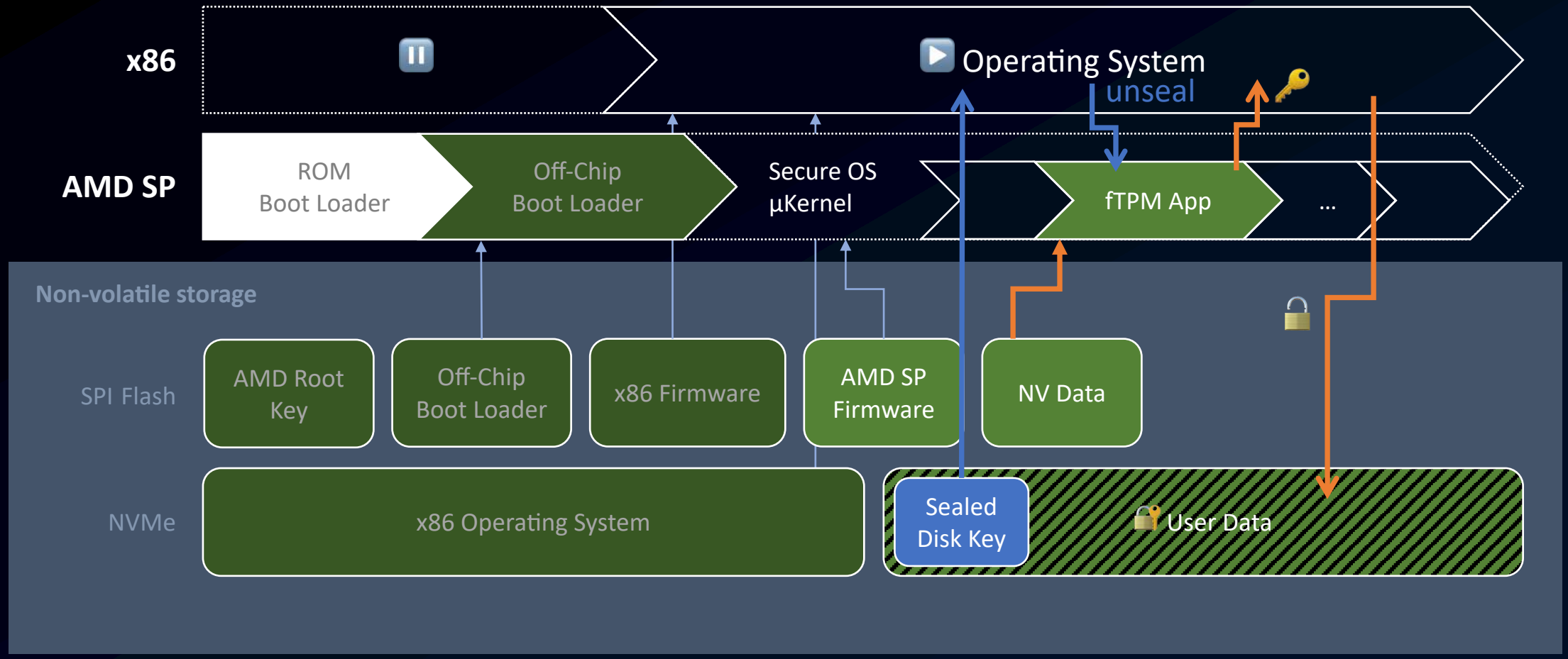
One of the more pron
Full Disk Encryption (F
rest without requiring t
passphrase. The standa
TPM, commonly calle
system was booted in
are released from the
Hereby, the system

cs.CR] 2 May 2023

Where in the boot is the fTPM?



Where in the boot is the fTPM?

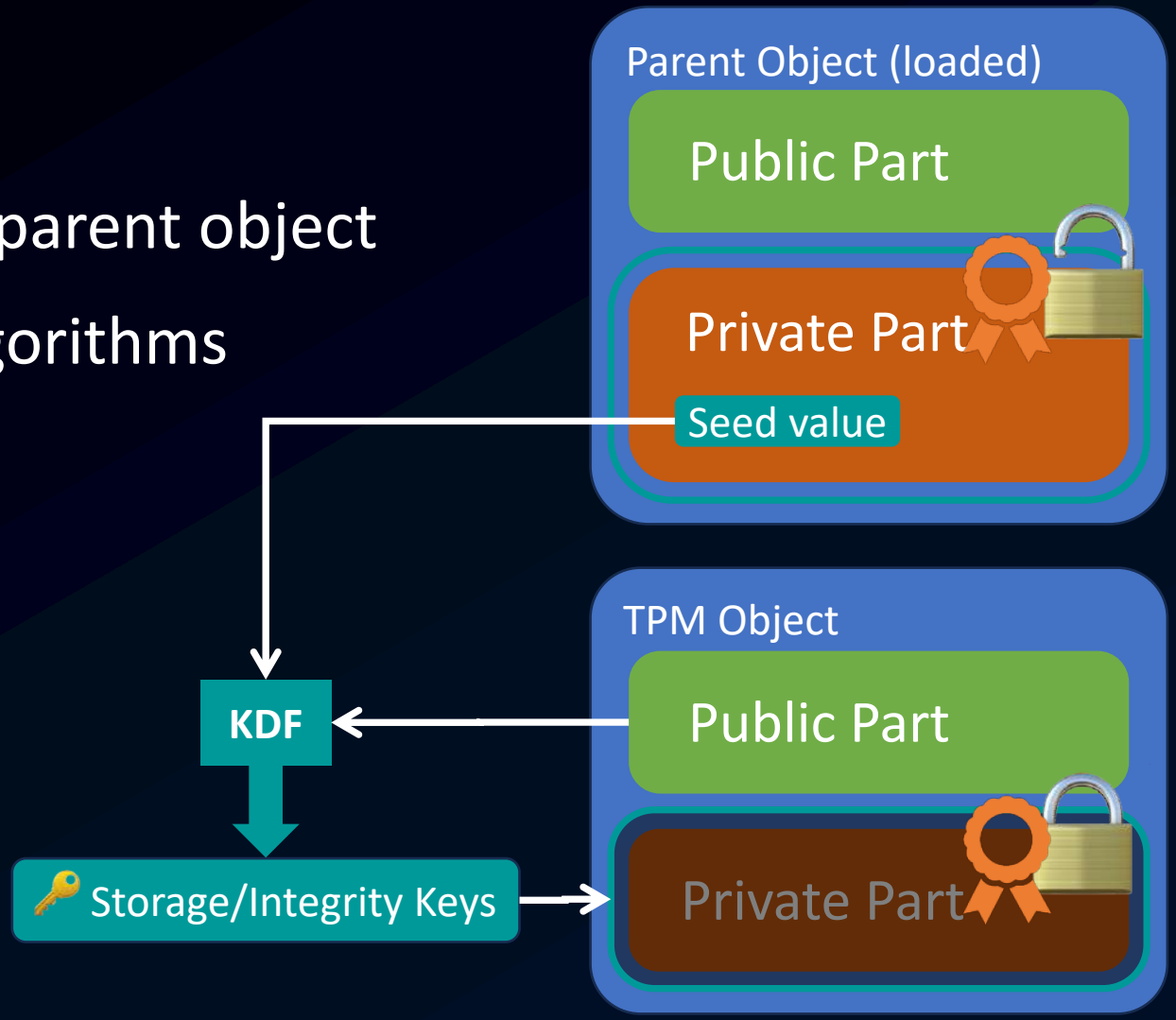


TPM Objects

- Public Part
  - Metadata
    - Which algorithm (AES, RSA, ECC, ...)
    - When and how can the object be used (policy)
  - Public key (if asymmetric algo.)
- Private Part
  - (Private) key
  - Auth value (for user input policy)
  - Seed value
  - Encrypted, integrity-protected

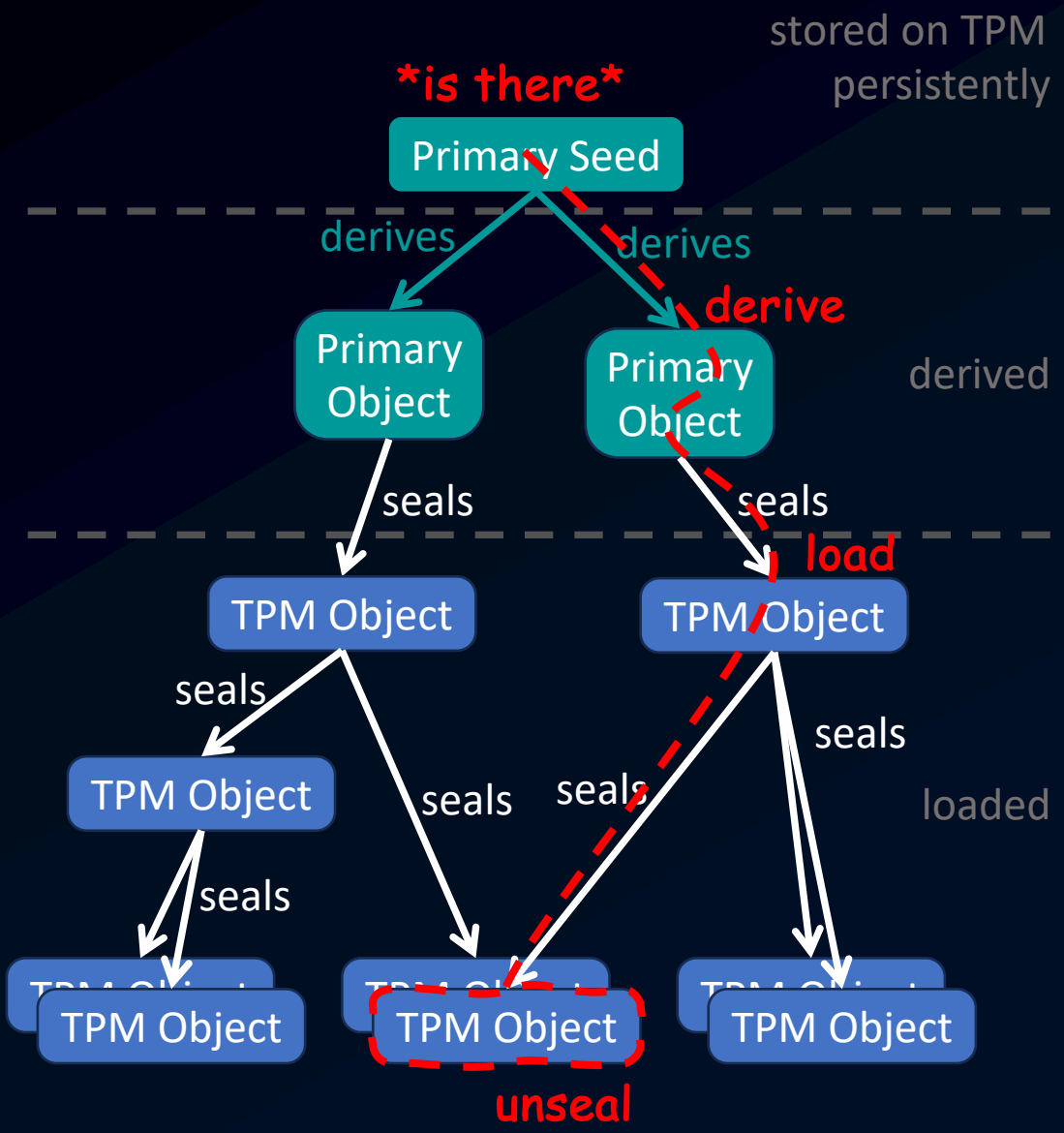
TPM Object Sealing

- Objects are sealed using a parent object
- TPM Spec. gives sealing algorithms



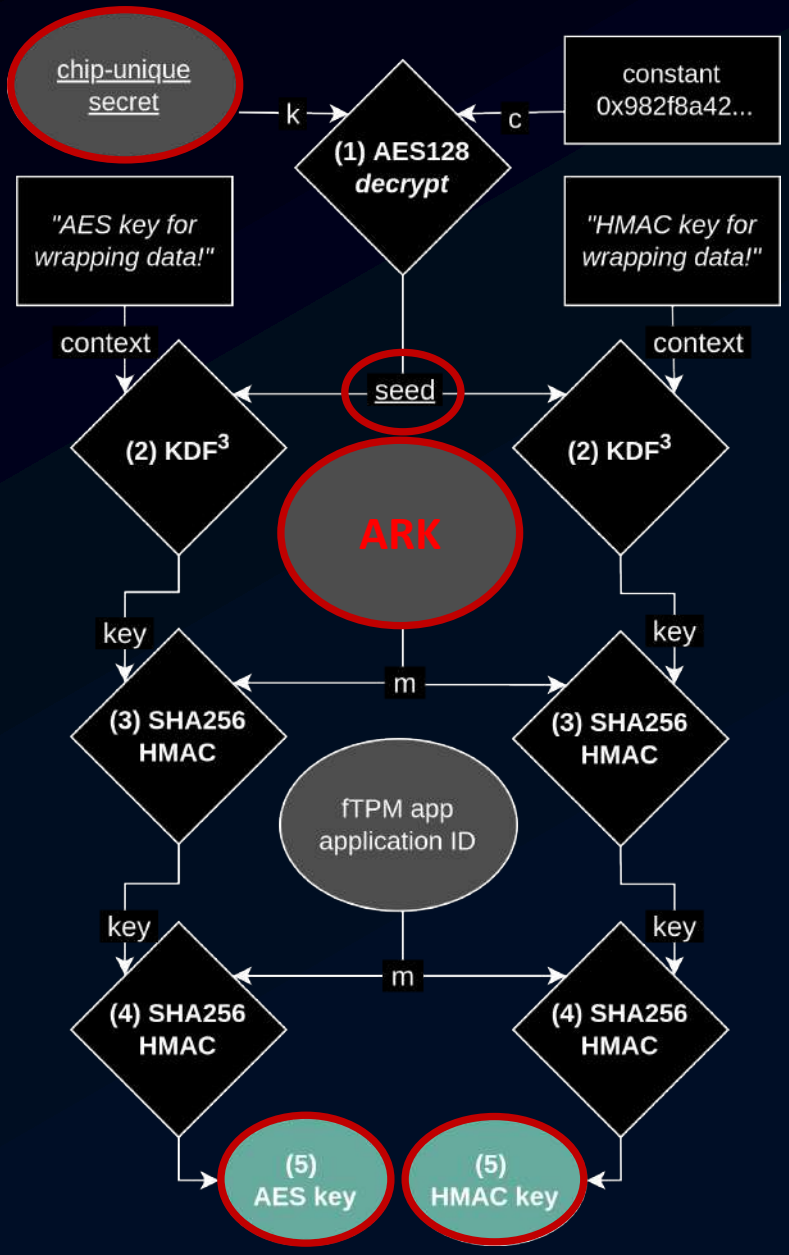
TPM Object Hierarchies

- TPM objects form a forest (multiple trees)
- Roots: Primary objects
  - Derived from one of three primary seeds
- Need to walk hierarchy to unseal/load object

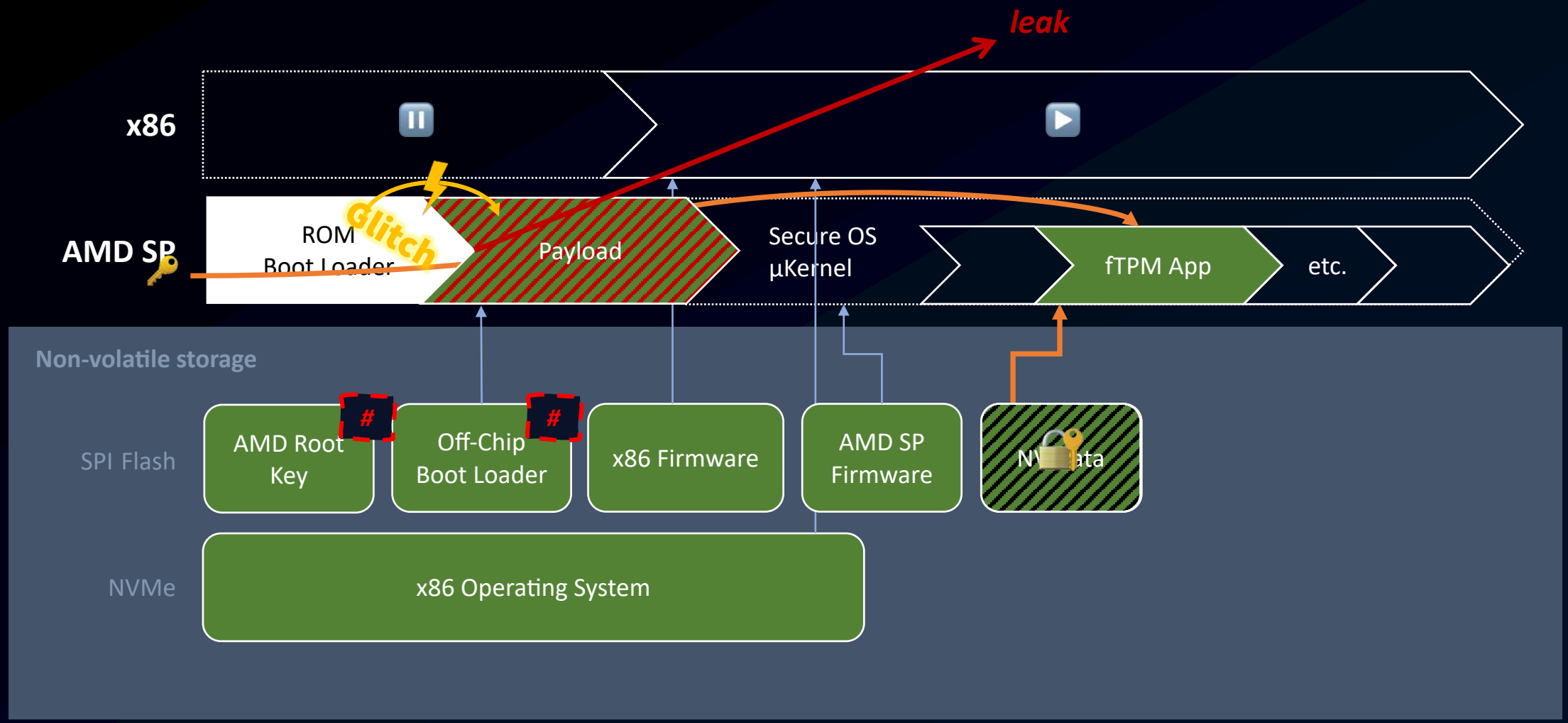


The Non-Volatile fTPM Data

- On SPI flash chip
  - Primary seeds, persistent counters, etc.
- Encrypted and integrity-protected
- We reverse-engineered the key derivation
- **Chip-unique secret** locked in CCP storage
  - Can only be used as AES key
- But we can extract intermediate value

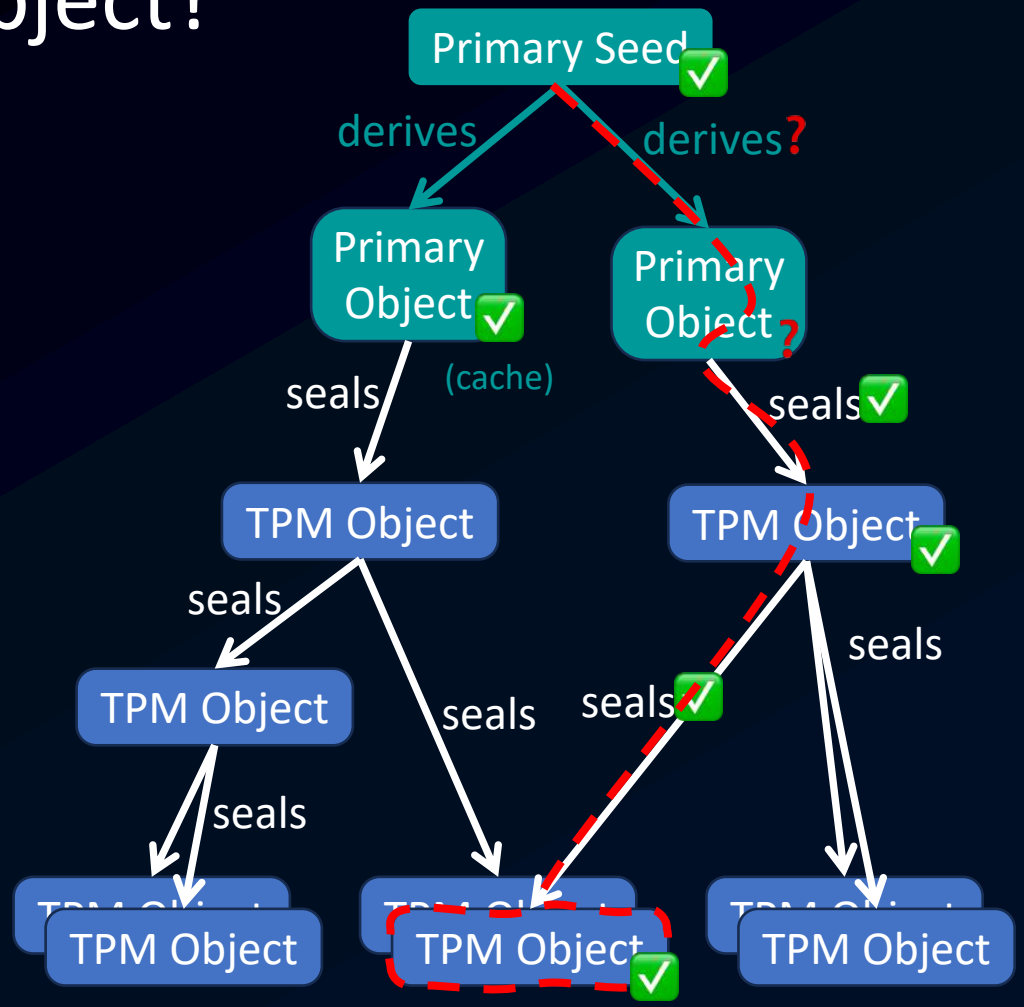


Where in the boot is the fTPM?



How do we unseal a TPM Object?

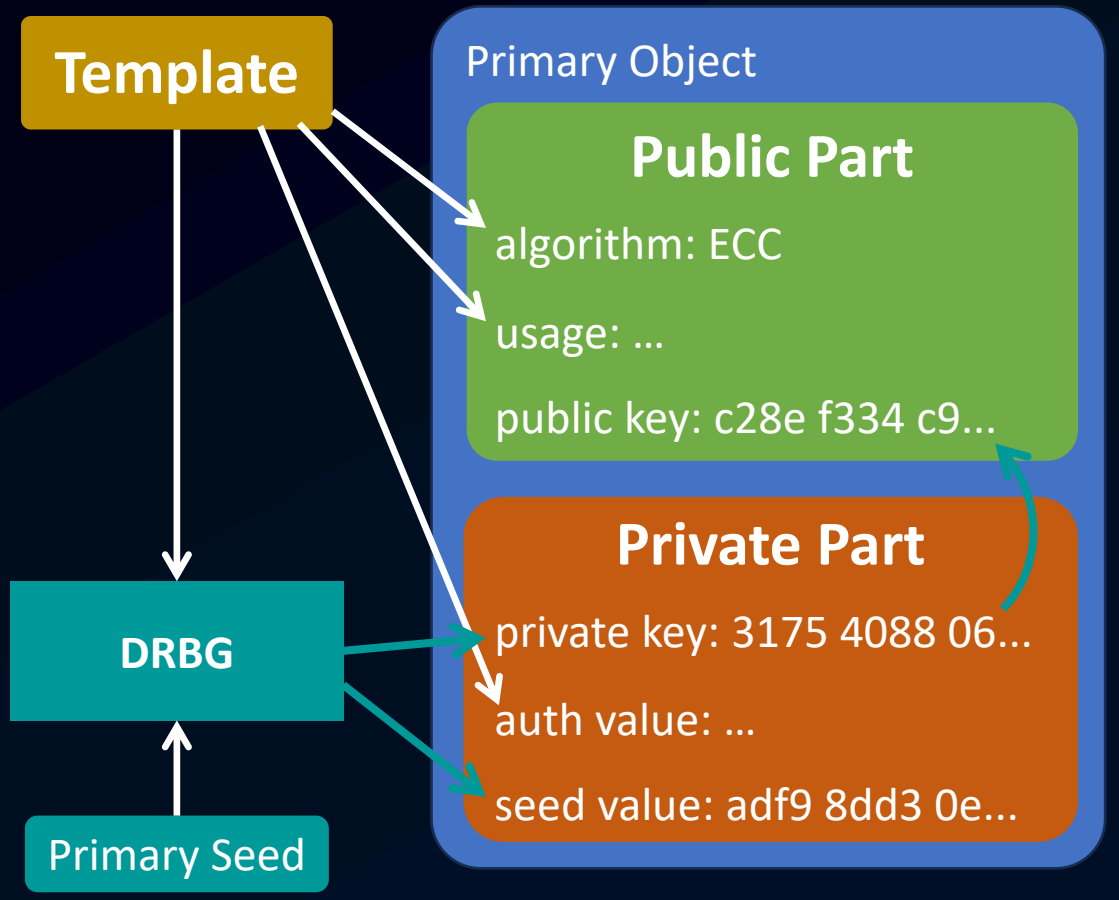
- TPM objects are stored externally
- Sealing is defined in TPM spec.
- Primary objects:
  - Some are cached in NV data (see faulTPM)
  - Seeds should be in NV data
  - Derivation only loosely specified!



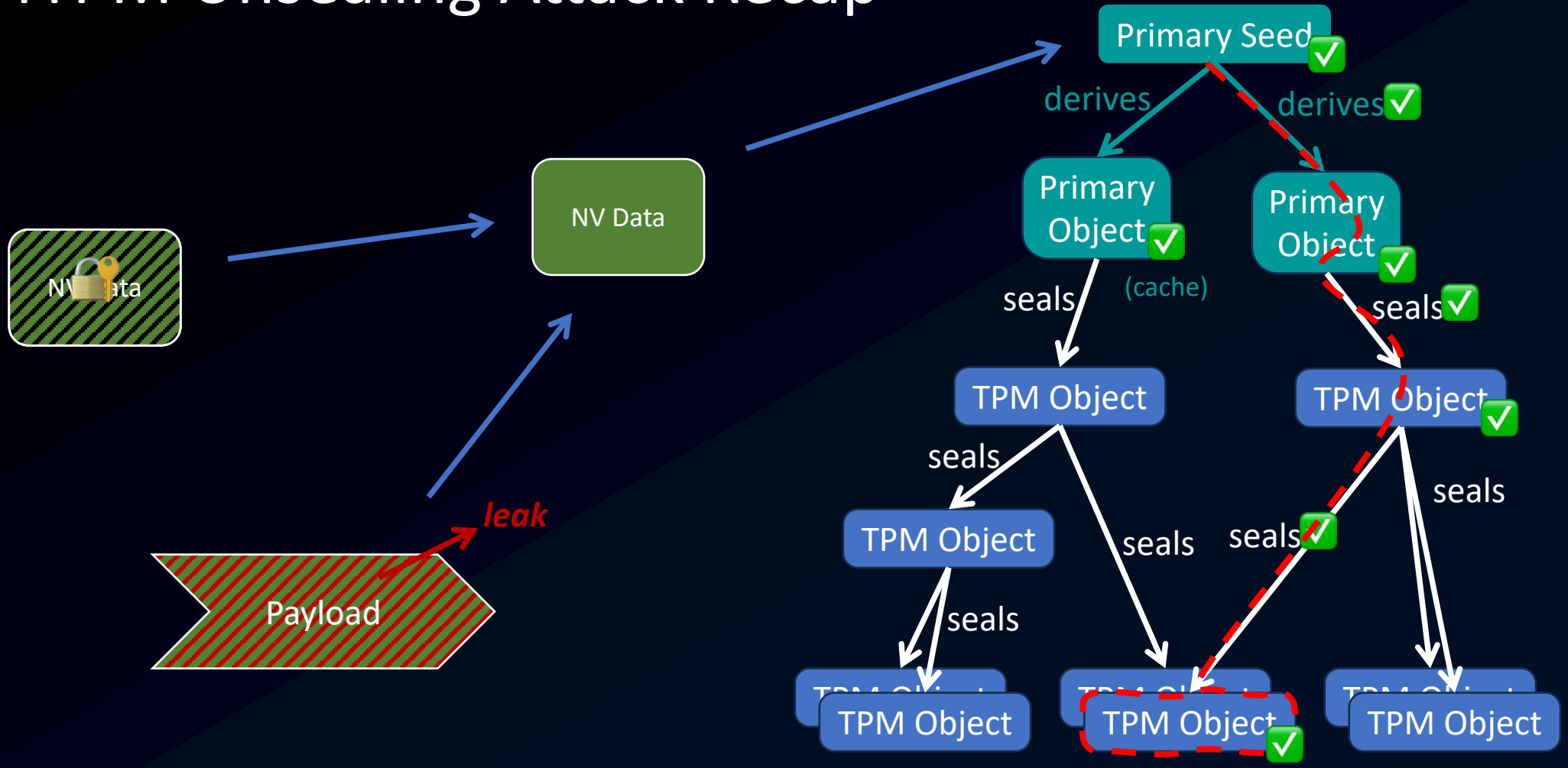
Primary Object Derivation

- Most fields come from the input “template”
  - Metadata, Authorization, …
- Other fields are derived from a **deterministic random bit generator (DRBG)**
  - Seeded with template and seed
  - **Algorithm not specified by spec.**

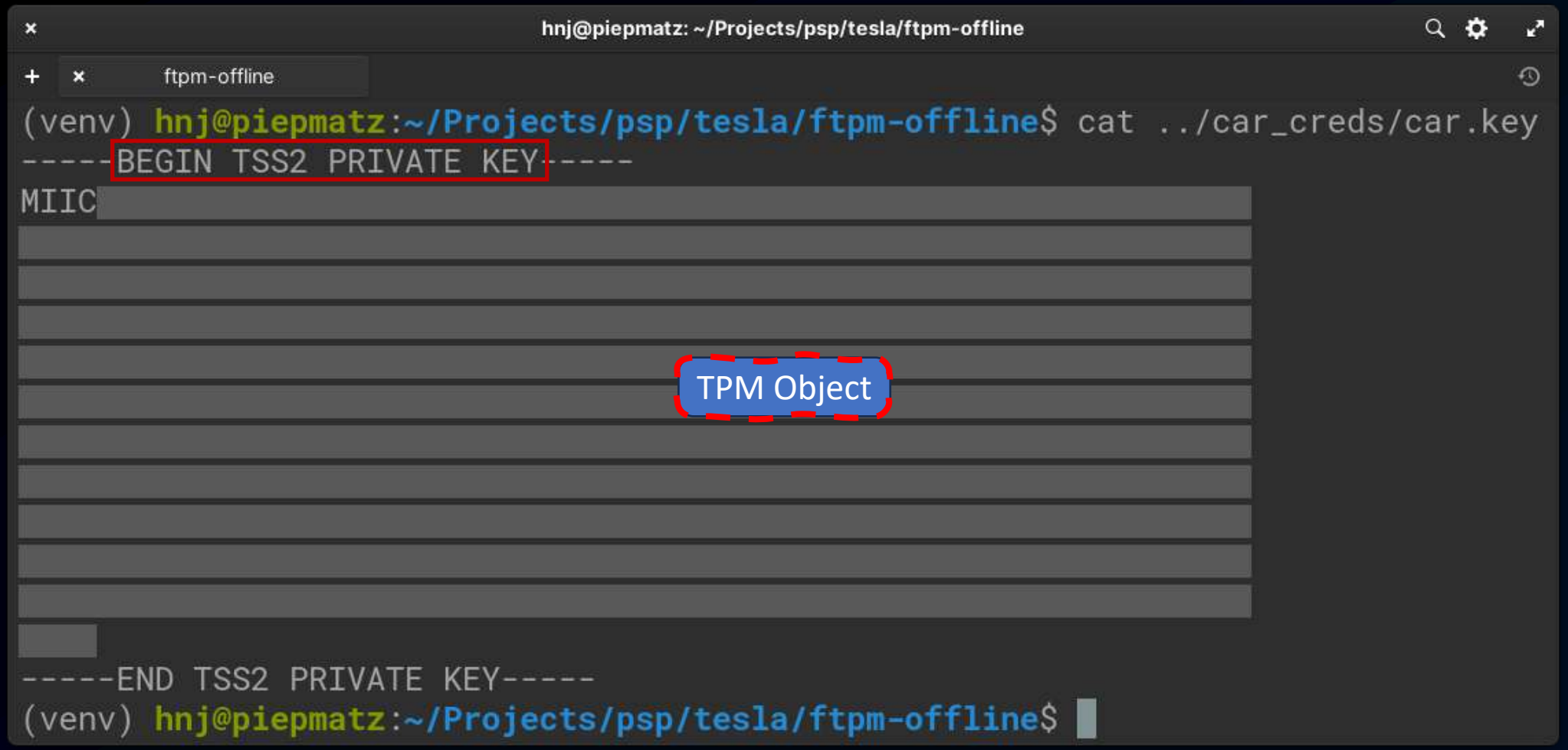
    → **reverse engineering**



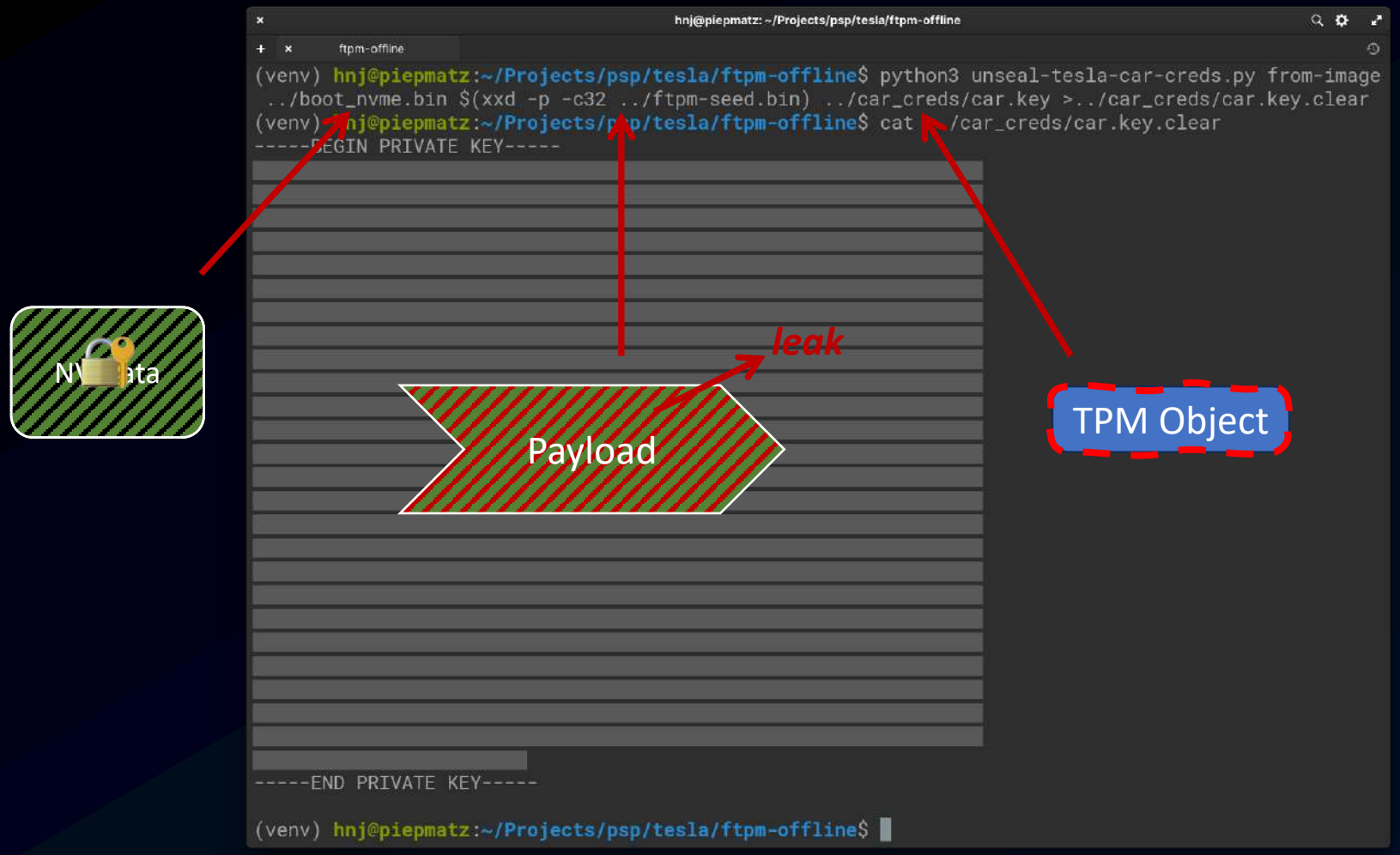
fTPM Unsealing Attack Recap



Finding the Car Credentials



Unsealing the Car Credentials

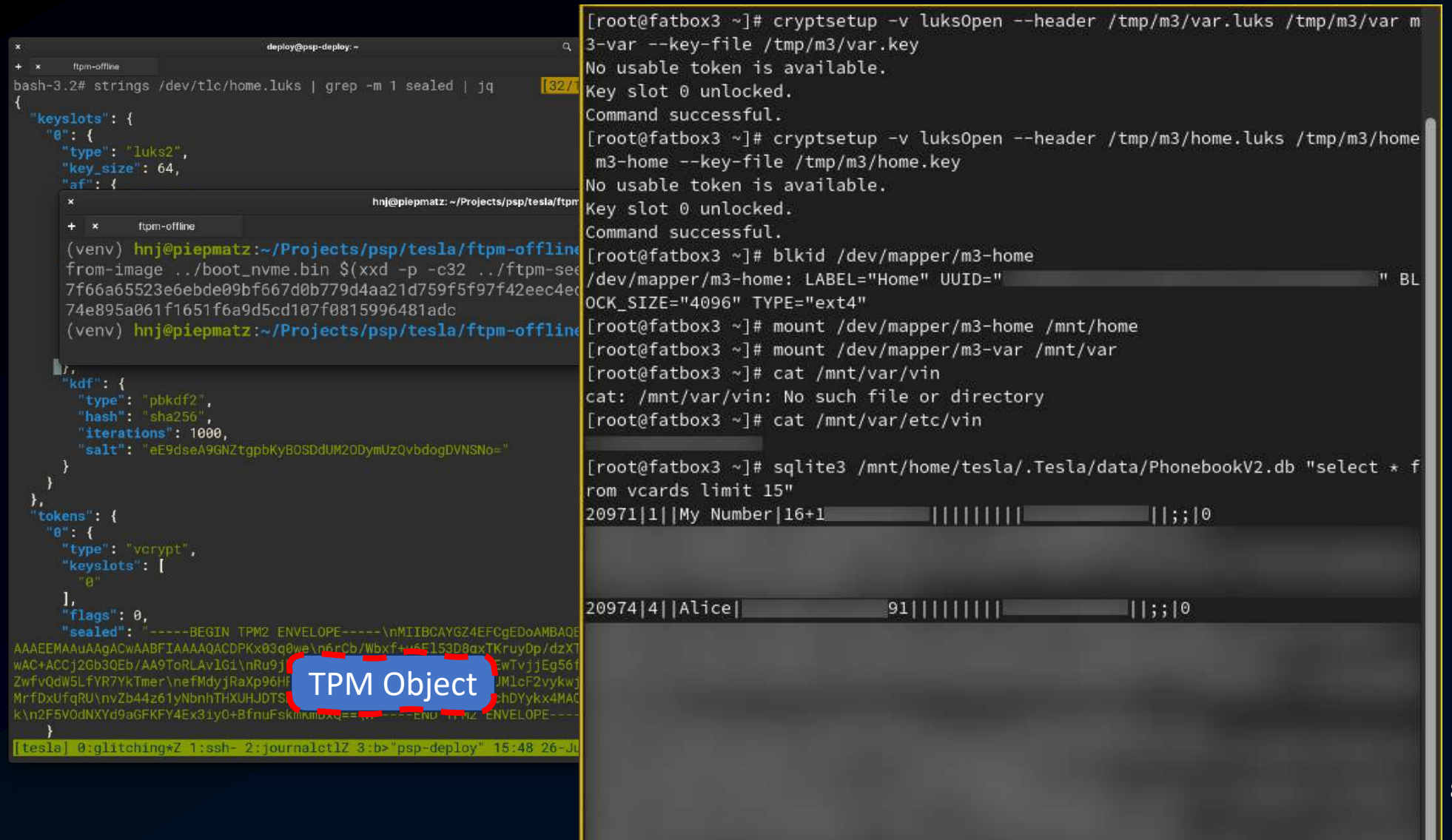


Using the Car Credentials

```
hnj@piepmatz: ~/Projects/psp/tesla/ftpm-offline
ftpm-offline
(venv) hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline$ echo -e "GET /mothership/vehicles/[redacted]/ HTTP/1.0\r\n"
| openssl s_client -connect api-prd.vn.tesla.services:443 -cert ../car_creds/car.crt -verify_quiet -quiet -ign_eof -nocomm
ands -key ../car_creds/car.key.clear
depth=0 CN = api-prd.vn.tesla.services, OU = Tesla Motors, O = Tesla, L = Palo Alto, ST = California, C = US
verify error:num=20:unable to get local issuer certificate
depth=0 CN = api-prd.vn.tesla.services, OU = Tesla Motors, O = Tesla, L = Palo Alto, ST = California, C = US
verify error:num=21:unable to verify the first certificate
HTTP/1.1 200 OK
Date: Wed, 26 Jul 2023 [redacted] GMT
Content-Type: application/json; charset=utf-8
Connection: close
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none
Referrer-Policy: strict-origin-when-cross-origin
X-TXID: [redacted]
ETag: [redacted]
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: [redacted]
X-Runtime: [redacted]
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy: default-src 'none'

{"id":[redacted],"vin":"[redacted]","nickname":"[redacted]","last_seen":[redacted],"created_at":[redacted]
,"current_version":"develop/2023.20.[redacted]","current_version_time":null,"active":true,"cell_number":null,"countr
y":"US","backseat_token":null,"backseat_token_updated_at":null,"radio_config":null,"service_possession":false,"hermes_capa
ble":true,"factory_gated":true,"delivered":true,"model":"3","use_country":null,"service_state":null,"connection_id":null,"
connection_region":"aws:us-west-2","birthplace":"fremont-factory","do_not_disturb_until":null,"device_type":"vehicle","is_
customer":true,"state":"asleep","odin_grablogs":false,"type":"Vehicle"}
(venv) hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline$
```

Extracting the Disk Encryption Keys



Outline

1. Analyzing Boot and Firmware Security
2. Hotwiring the Infotainment system
3. Extracting Secrets from the Tesla

Summary

1. We reverse-engineered Tesla's boot security
   - Tesla sets a good example of how it should be done
2. We still rooted the system through voltage glitching
   - This allows to activate some soft-locked features without paying
   - Not software-patchable by anyone
3. We extracted hardware-bound secrets from the TPM using the same attack
   - This can ease independent repairs

Key Takeaways

1. Soft-locking hardware features increases hacking incentives
2. Using battle-tested open-source software like Coreboot and Linux provides a good level of software security
3. But: Consider *hardware* attacks in your threat model, too

Responsible Disclosure(s)

- 2021: Informed AMD about voltage glitching susceptibility
- 2022: Shared faulTPM attack with AMD (based on glitching)
- 2023: Informed Tesla about “AMD jailbreak”
  - Tesla was ‘relieved’ that a single glitch did not yield persistence
  - Did not comment the car_creds extraction

Jailbreaking an Electric Vehicle in 2023

WHAT IT MEANS TO HOTWIRE TESLA'S X86-BASED SEAT HEATER

Christian Werling — cwerling@sect.tu-berlin.de
Niclas Kühnapfel — kuehnapfel@tu-berlin.de
Hans Niklas Jacob — hnj@sect.tu-berlin.de




Oleg Drokin — drokin@linuxhacker.ru

All code available at:
github.com/PSPReverse