DIY: Secure Embedded Projects using Trust

Teddy Reed && David Anthony

Why

- Fascination with embedded systems and devices
- Popularity of Secure Boot, UEFI, and Trusted Computing
- Lack of TPM availability
    - There are great Linux drivers in tpmdd, unfortunately the devices cannot be purchased without an NDA or cannot interface easily with embedded systems
- Hope to inspire community

What

- Short introduction to Trusted Computing focusing on features appealing to embedded developers
  - Compare criticisms to creativity
- UEFI, Linux, and U-Boot drivers for your TPM
- Secure Boot example using a TPM for U-Boot
- More examples, configuration tutorials, documentation and getting-started “kits”

Part 1: TPM

Trusted Platform Module

"A facial recognition system which doesn't recognize you if you change your shirt" - Ariel Segall

Secure, Trusted, Verified Boot

Software Integrity

(Local and Remote)

Your Imagination

Building Blocks

Protected Storage

Apply access control to storage based on logical or physical machine state

Non-Removable Private Keys

Allow portable-encrypted private keys, constrain use to a unique platform

Measurement Registers

Track platform execution and apply access control to execution measurement

Hashing, RNG, Key Generation

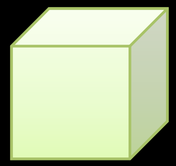
Common crypto functions available to commodity hardware in memory-absent environments

A measurement register, or Platform Configuration Register (PCR), each 160-bit wide, can ONLY be extended, read, or reset

Building Blocks

Track platform execution and apply access control to execution measurement

Measurement Registers



A measurement register, or Platform Configuration Register (PCR), each 160-bit wide, can ONLY be extended, read, or reset

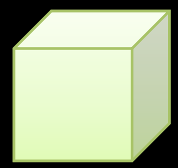
PCR_Extend(n, hash):

PCR(n) := SHA1(PCR(n) + hash)

Building Blocks

Track platform execution and apply access control to execution measurement

Measurement Registers



Asymmetric Key Cryptography

Building Blocks

Software Support

Trusted Computing Terminology

- Ownership
- Key types
- Binding, Sealing
- Attestation
  Appraisal
- Measurement

Trusted Computing Terminology

- Ownership
- Key types
- Binding, Sealing
- Attestation
  Appraisal
- Measurement

“Take Ownership” -
Assigns an owner to the TPM, setting the owner password and creating a “Storage Root Key” (SRK)

Clearable, Repeatable

Trusted Computing Terminology

- Ownership
- Key types
- Binding, Sealing
- Attestation Appraisal
- Measurement

Endorsement (TPM Identity)

SRK - Root of key hierarchy

transitive parent key

Attestation Identity

Signing Keys

...more!

Trusted Computing Terminology

- Ownership
- Key types
- Binding, Sealing
- Attestation Appraisal
- Measurement

Binding - Data encryption with the TPM Endorsement Key

Sealing - Data encryption with the additional property of PCR values at the time of encryption

Quoting - Like sealing, but produces a signature

Trusted Computing Terminology

- Ownership
- Key types
- Binding, Sealing
- Attestation
  Appraisal
- Measurement

Attestation - Vouching for the accuracy of information

Appraisal - Assessing the information using a previously defined state

Trusted Computing Terminology

- Ownership
- Key types
- Binding, Sealing
- Attestation Appraisal
- Measurement

Static Root of Trust

Dynamic Root of Trust

Cumulative hashes of executables, libraries, scripts, etc.

Trusted Computing Terminology

Ariel Segall's - Intro to Trusted Computing 101

http://goo.gl/oh21v

Trusted Computing Terminology

Ariel Segall's - Intro to Trusted Computing 101

http://goo.gl/oh21v

Trust us

Criticisms

Critique	Creativity
Remote Attestation Abuse and Service Constraints	Distributed Attestation Services
Manufacturer Trust	Ignorance, EK-less
Privacy	Key-use Awareness and DAA

Criticisms

Critique	Creativity
Remote Attestation Abuse and Service Constraints	**Distributed Attestation Services**
Manufacturer Trust	EK-less TPM
Privacy	Key-use Awareness and DAA

Criticisms

Critique	Creativity
Remote Attestation Abuse and Service Constraints	Distributed Attestation Services
Manufacturer Trust	**EK-less TPM**
Privacy	Key-use Awareness and DAA

Criticisms

Critique	Creativity
Remote Attestation Abuse and Service Constraints	Distributed Attestation Services
Manufacturer Trust	EK-less TPM
Privacy	**Key-use Awareness and DAA**

Booting securely in the non-embedded world

Booting securely in the non-embedded world

OEM Custom

Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader

OEM Custom

Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader

#

OEM Custom

Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader

OEM Custom

Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader

OEM Custom

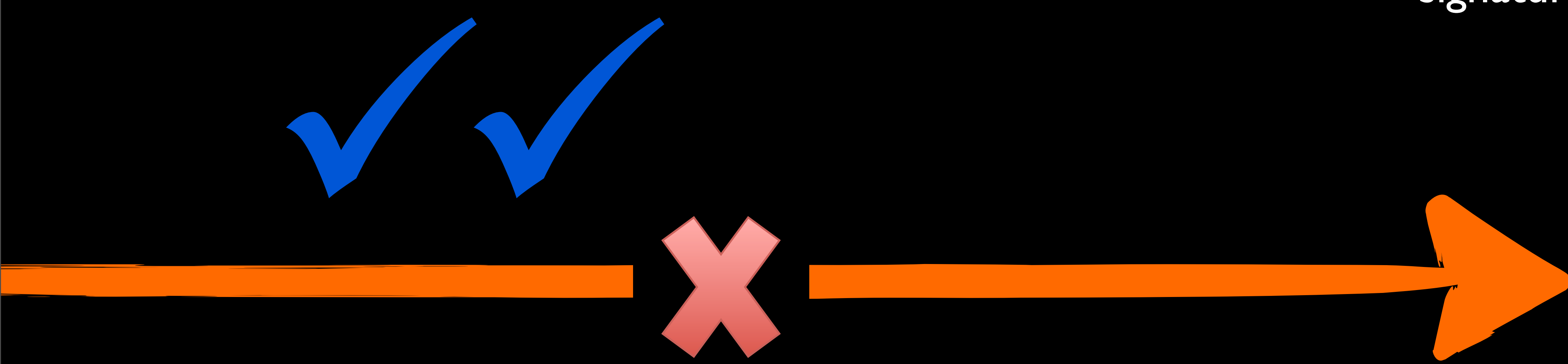
Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader



OEM Custom

Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader

OEM Custom

Use ROM or pre-BIOS code to verify firmware signatures (using known or custom signature verification algorithms

UEFI

Check UEFI application, driver, and bootloader signatures against a user or OEM-controlled certificate store

Trusted Grub
TBOOT
TXT: DRTM
Anti-EM

Check kernel, ram disk, and additional OS boot data signatures within the boot loader

Recap: Measurement

- Fancy word for secured-logging
- Systems and designers can implement a "static" or "dynamic" root ...of trust measurement
- Struggle to add support for measurement
- We missed some implementations, please don't be mad :'(

Part 2: TPM on **your** embedded device

BeagleBone Revision A5, A6

JTAG Emulator (XDS100v2), USB Power, USB Ethernet, UART0 (Serial)

Using 1 Micro USB!

BeagleBone Revision A5, A6

Out of the 96 pins (most with 7 configuration modes) almost every interface on the board is easily exposed to your creativity

JTAG Emulator (XDS100v2), USB Power, USB Ethernet, UART0 (Serial)

Using 1 Micro USB!

BeagleBone Revision A5, A6

Out of the 96 pins (most with 7 configuration modes) almost every interface on the board is easily exposed to your creativity

Many supported Linux distributions, great documentation for assembling your own, and compiling your own kernel (even community support for 3.7/3.8)

JTAG Emulator (XDS100v2), USB Power, USB Ethernet, UART0 (Serial)

Using 1 Micro USB!

TPS6517B

5/3.3/1.8V

Ethernet

USB

MMC0

AM3359

256M DDR

CAN

TPS6517B

EEPROM

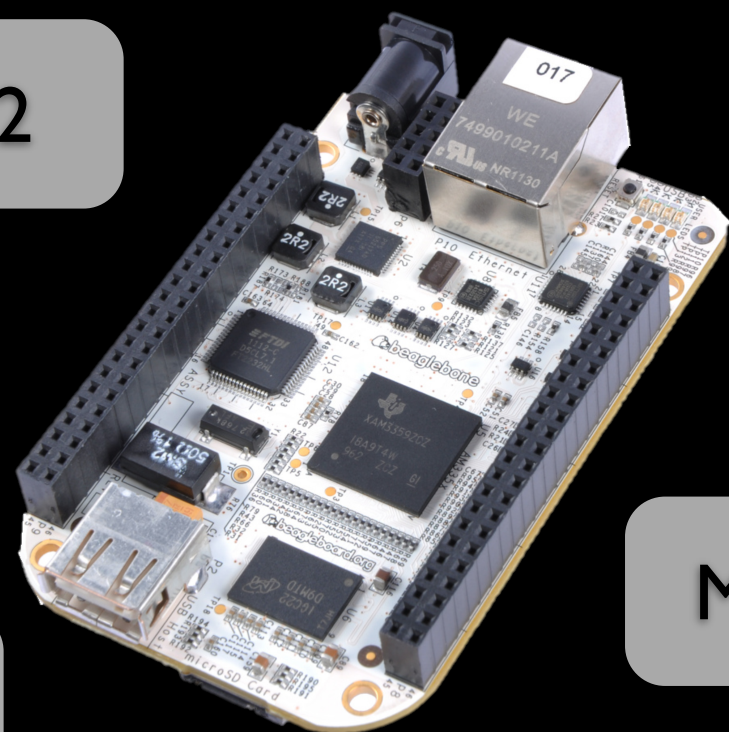
5/3.3/1.8V

Ethernet

I2C 1

I2C 2

UART x4



GPMC

SPI

MMC1

MMC2

USB

MMC0

Timer x4

AM3359

256M DDR

Battery Charger

CAN

TPS65217B

EEPROM

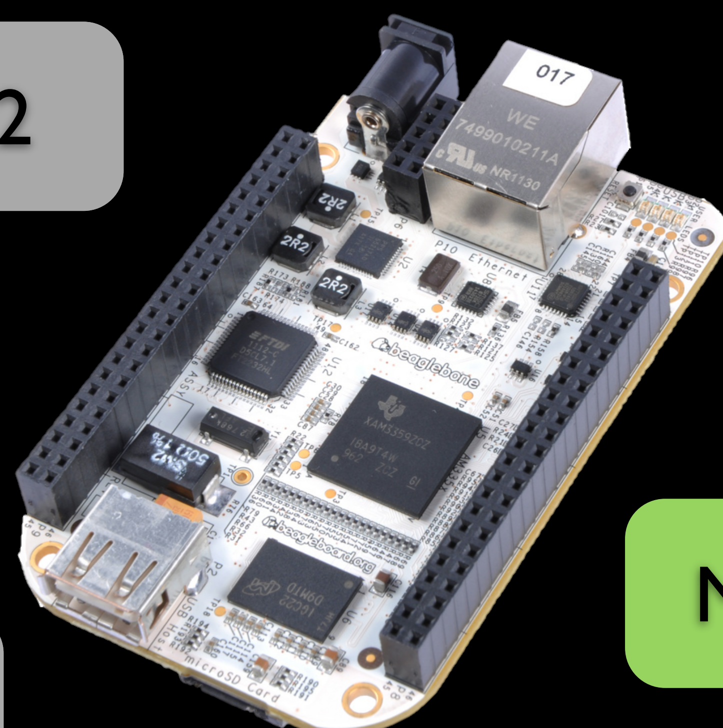
5/3.3/1.8V

Ethernet

I2C 1

I2C 2

UART x4



GPMC

SPI

MMC1

MMC2

USB

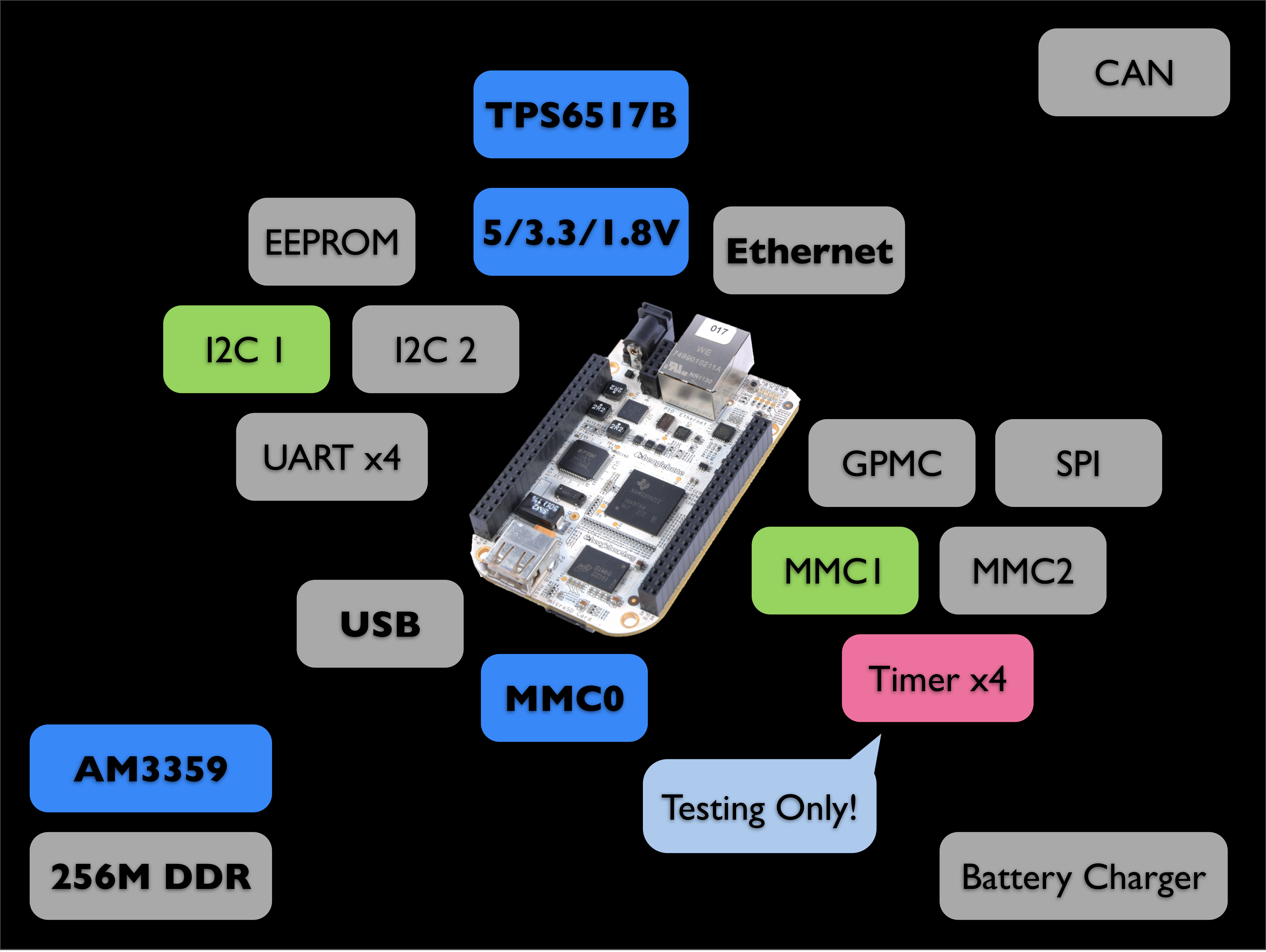
MMC0

Timer x4

AM3359

256M DDR

Battery Charger


CAN
TPS65I7B
5/3.3/I.8V
EEPROM
Ethernet
I2C I
I2C 2
UART x4
GPMC
SPI
MMCI
MMC2
USB
MMC0
Timer x4
AM3359
Testing Only!
256M DDR
Battery Charger

CAN

TPS6517B

TPM

EEPROM

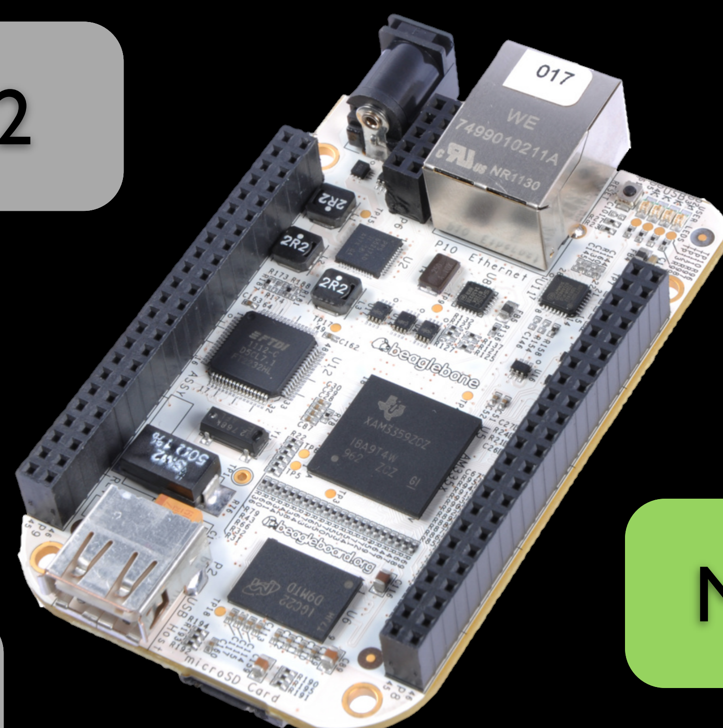
5/3.3/1.8V

Ethernet

I2C 1

I2C 2

UART x4



GPMC

SPI

MMC1

MMC2

USB

MMC0

Timer x4

AM3359

256M DDR

Battery Charger

I2C1_SDA

I2C1_SCLK

Not so exciting here, we use BeagleBone's I2C1 bus because it is reserved for non-cape components

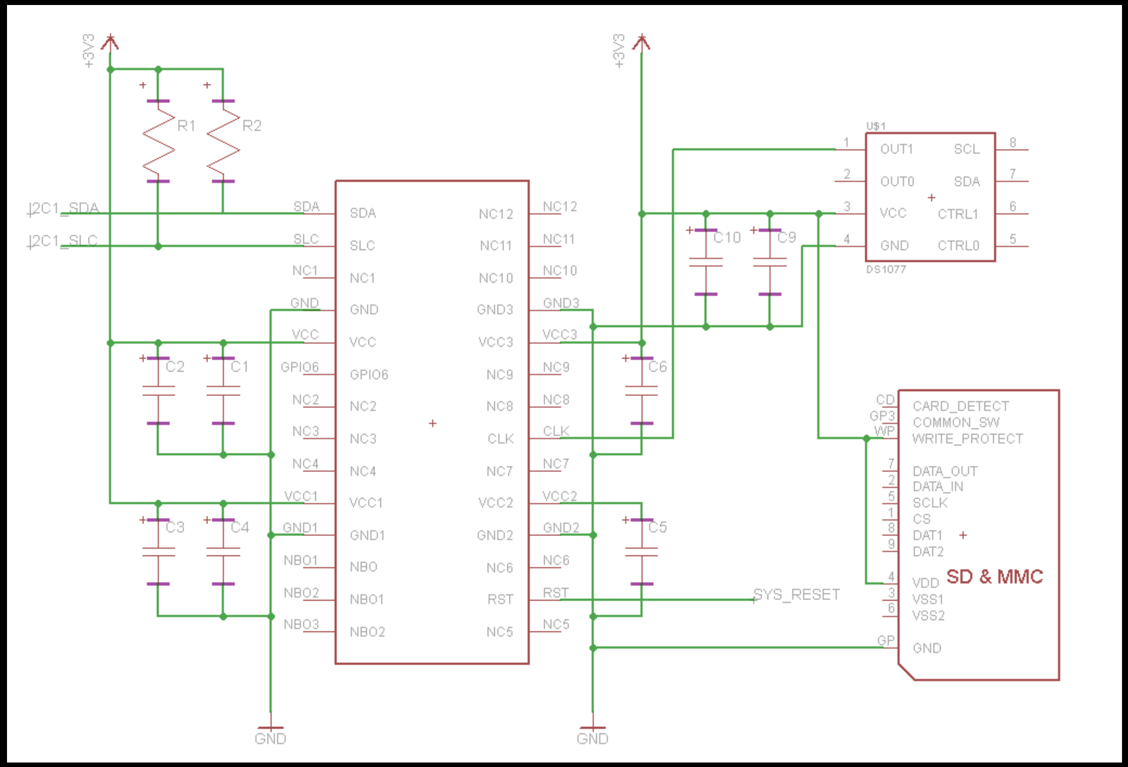
SYS_RESETn

SYS_RESETn is used by the CPU for a soft or hard reset. The AM3359 will pull this line during a soft reset (with a variable frequency), and the hardware will pull it to force a hard reset

CLK

An separate external clock assures no software control by the system

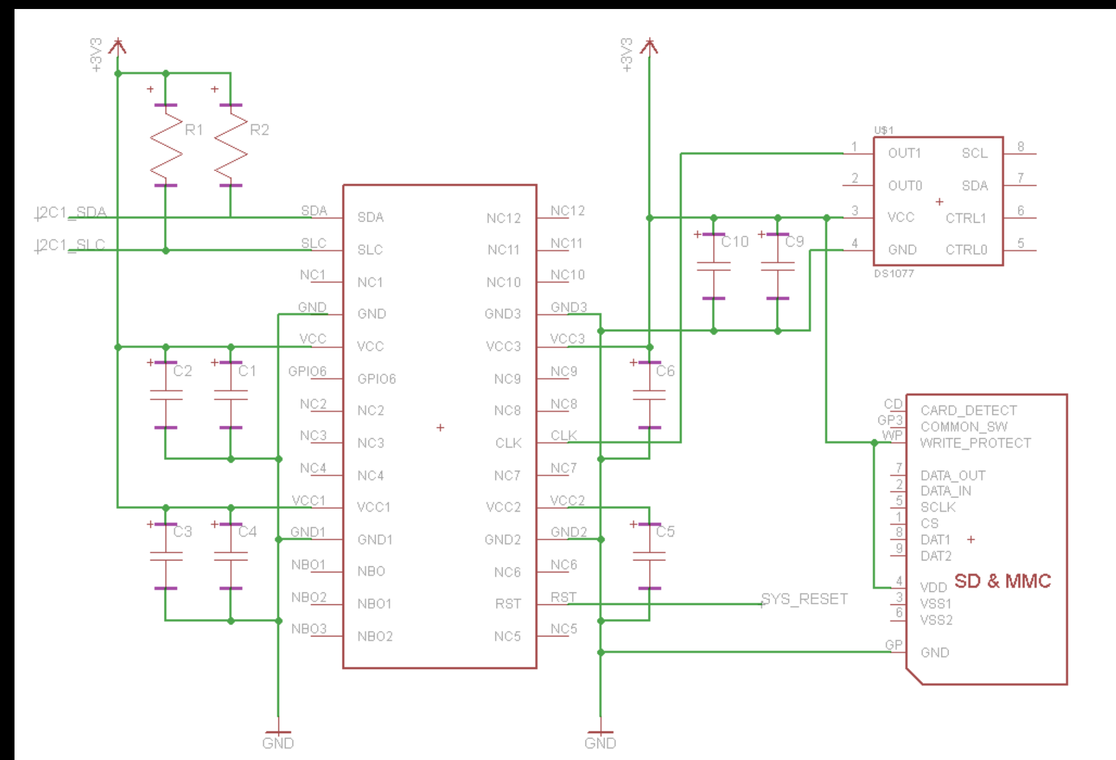
Configuration Schematic:



Software:

- U-Boot/Linux TPM driver (branches for each):
  http://github.com/theopolis/tpm-i2c-atmel
- UEFI I2C TPM SecurityPkg:
  http://github.com/theopolis/SecurityPkg

Configuration Schematic:



What you can't read that?

Software:

- U-Boot/Linux TPM driver (branches for each):
  http://github.com/theopolis/tpm-i2c-atmel
- UEFI I2C TPM SecurityPkg:
  http://github.com/theopolis/SecurityPkg

TPM Manufacturers

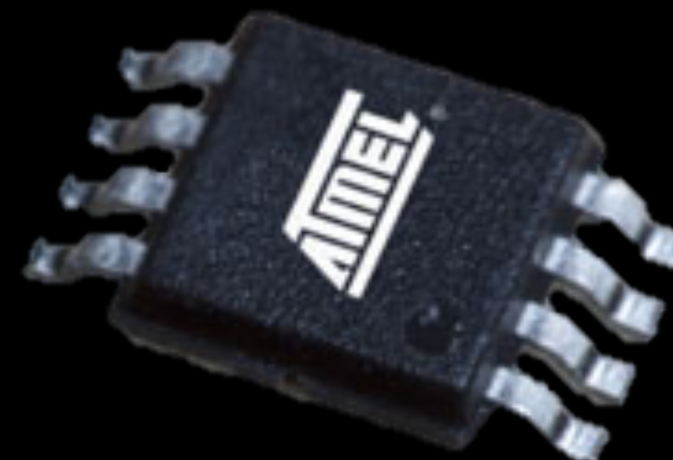
- Atmel
- Broadcom
- Infineon
- Intel
- ITE
- Nuvoton (?)
- Sinosun
- STMicro
- Toshiba
- *Software

Acquiring a TPM

- Atmel AT97SC3204[T]
- $6.30 - $6.50
- DigiKey, Mouser, AVNET Express
- Option for purchasing EK-less TPM

Board

Alternate Storage



TPM

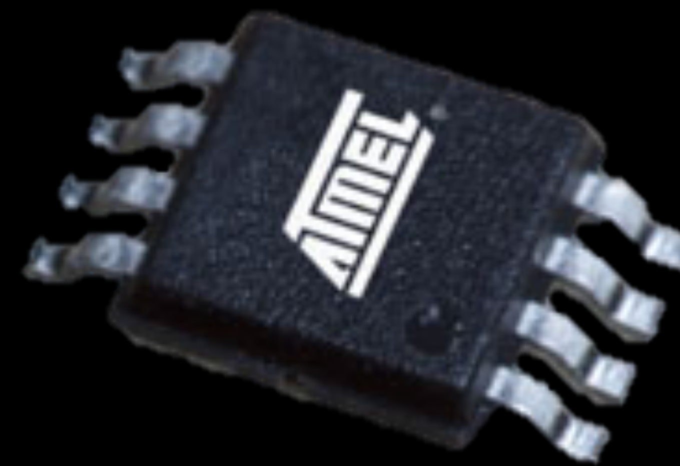
(

| 

)

33MHz Clock

Board



TPM

Alternate Storage

(

| 

)

33MHz Clock

Create a SRTM on the BeagleBone

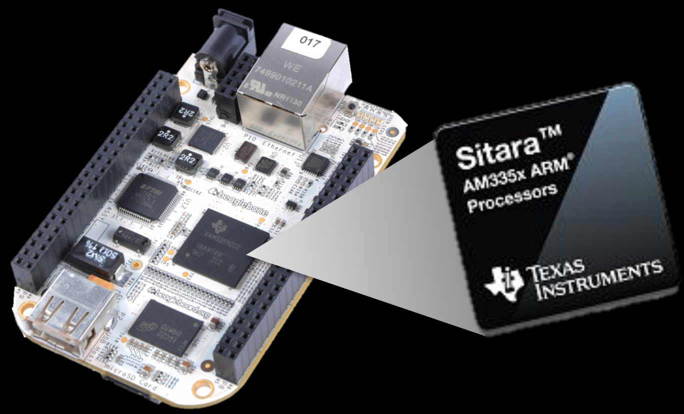
Potential for Error

- A static root of trust measurement implies a set of routines secured from any software attack possible

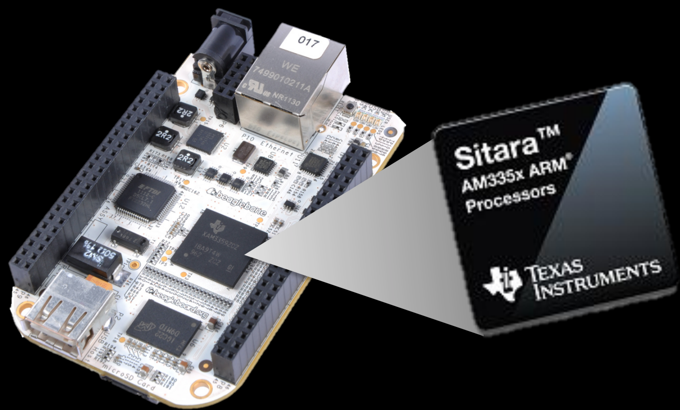
MMC0's write-protect pin (P8-42) is multiplexed with others. An SRTM using MMC0 violates the above statement as an attacker can change the MUX setting for the pin, thus disabling the write protecting and changing our initialization routines

Options

- The BeagleBone exposes the AM3359 boot configuration pins, configure them for a default boot of MMC1, and control the WP pin externally
- Similar, but use USB or SPI to retrieve the code
- Permanently disable writing to the SD card in MMC0 using a PROGRAM_CSD command CMD27 with bit 13 set



ROM Code reads a boot config from pins pulled high or low to determine a boot device then reads and executes a loader from device



ROM Code reads a boot config from pins pulled high or low to determine a boot device then reads and executes a loader from device

The default boot device is MMC1, using partition 1 and a FAT a file called MLO (x-loader) is executed

By pulling WP high, we prevent SW modifications to this media*

ROM Code reads a boot config from pins pulled high or low to determine a boot device then reads and executes a loader from device

The default boot device is MMC1, using partition 1 and a FAT a file called MLO (x-loader) is executed

By pulling WP high, we prevent SW modifications to this media*

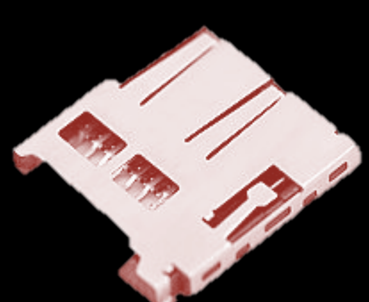
The MLO is called a second-phase loader (SPL), the first phase is the ROM code, and is where we initialize the SRTM

ROM Code reads a boot config from pins pulled high or low to determine a boot device then reads and executes a loader from device

The default boot device is MMC1, using partition 1 and a FAT a file called MLO (x-loader) is executed

By pulling WP high, we prevent SW modifications to this media*

The MLO is called a second-phase loader (SPL), the first phase is the ROM code, and is where we initialize the SRTM

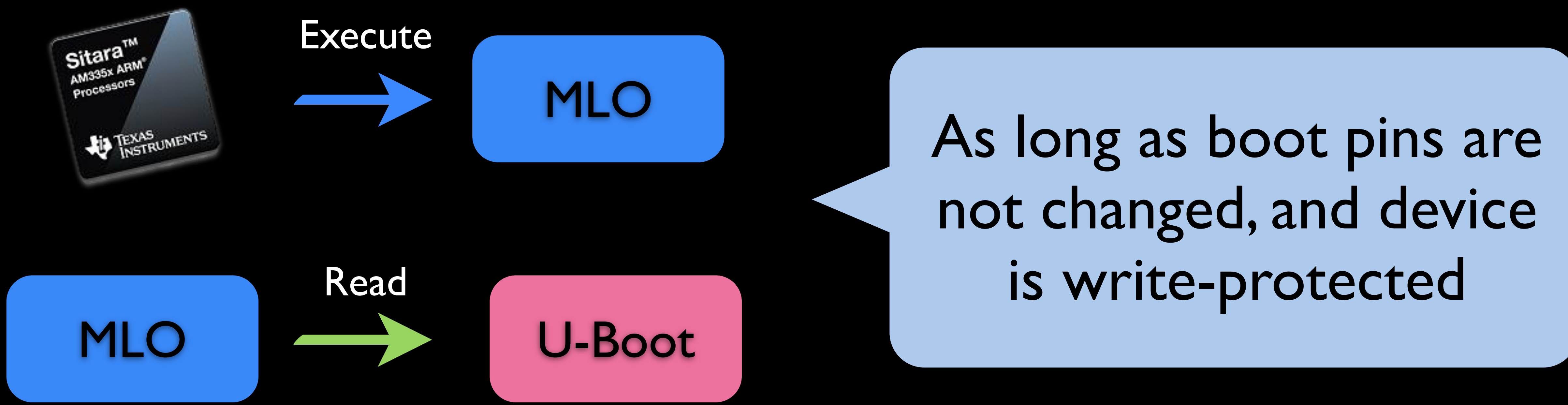


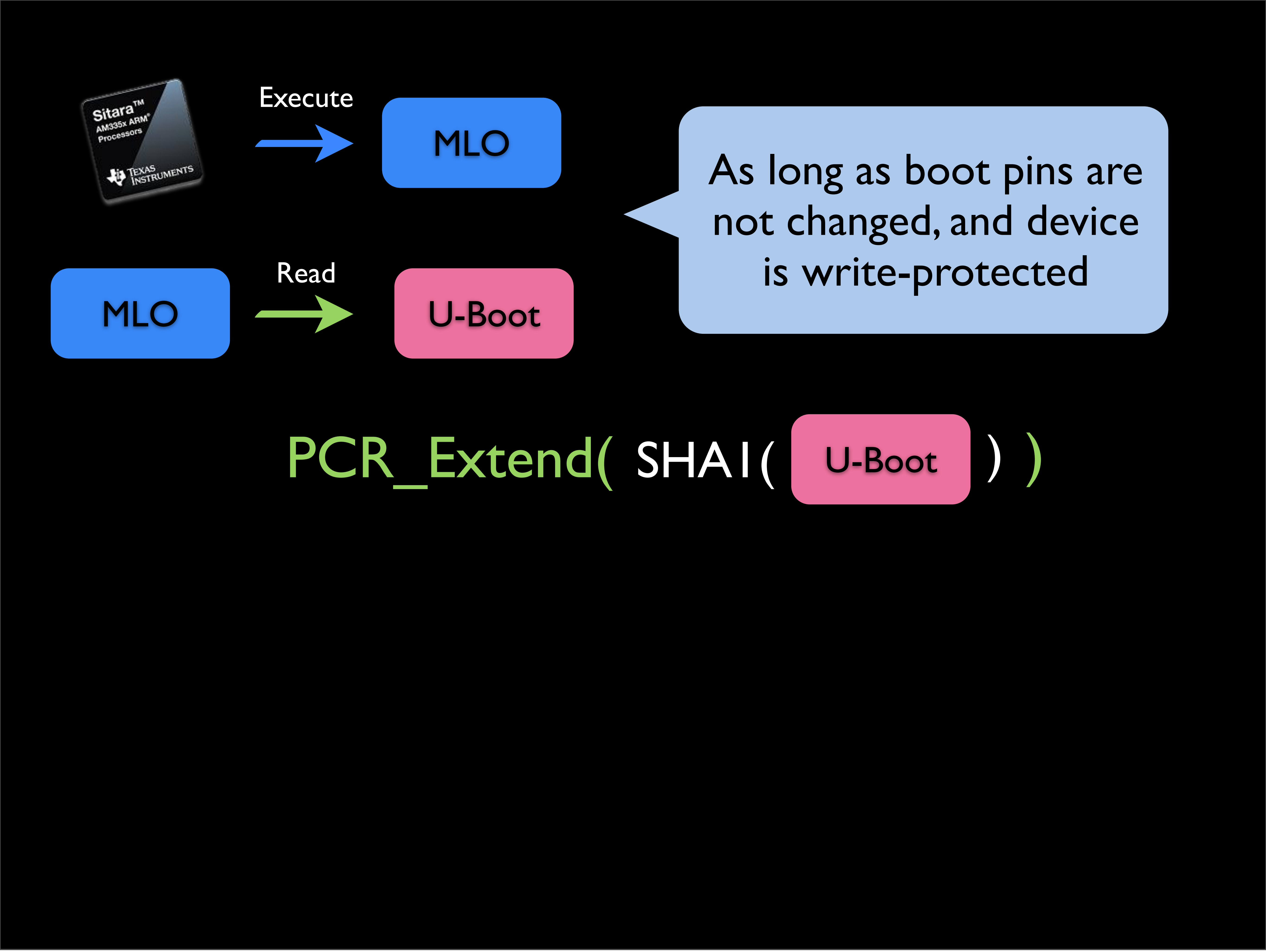
The SPL reads and measures U-Boot or UEFI from an alternate device (e.g., MMC0)

The measurement chain continues into R/W storage

Use the SRTM for a Secure Boot

Implemented with Hashing, Sealing, and Unsealing


Sitara™
AM335x ARM®
Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
As long as boot pins are not changed, and device is write-protected


Sitara™ AM335x ARM® Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
As long as boot pins are not changed, and device is write-protected
PCR_Extend(SHA1(U-Boot))

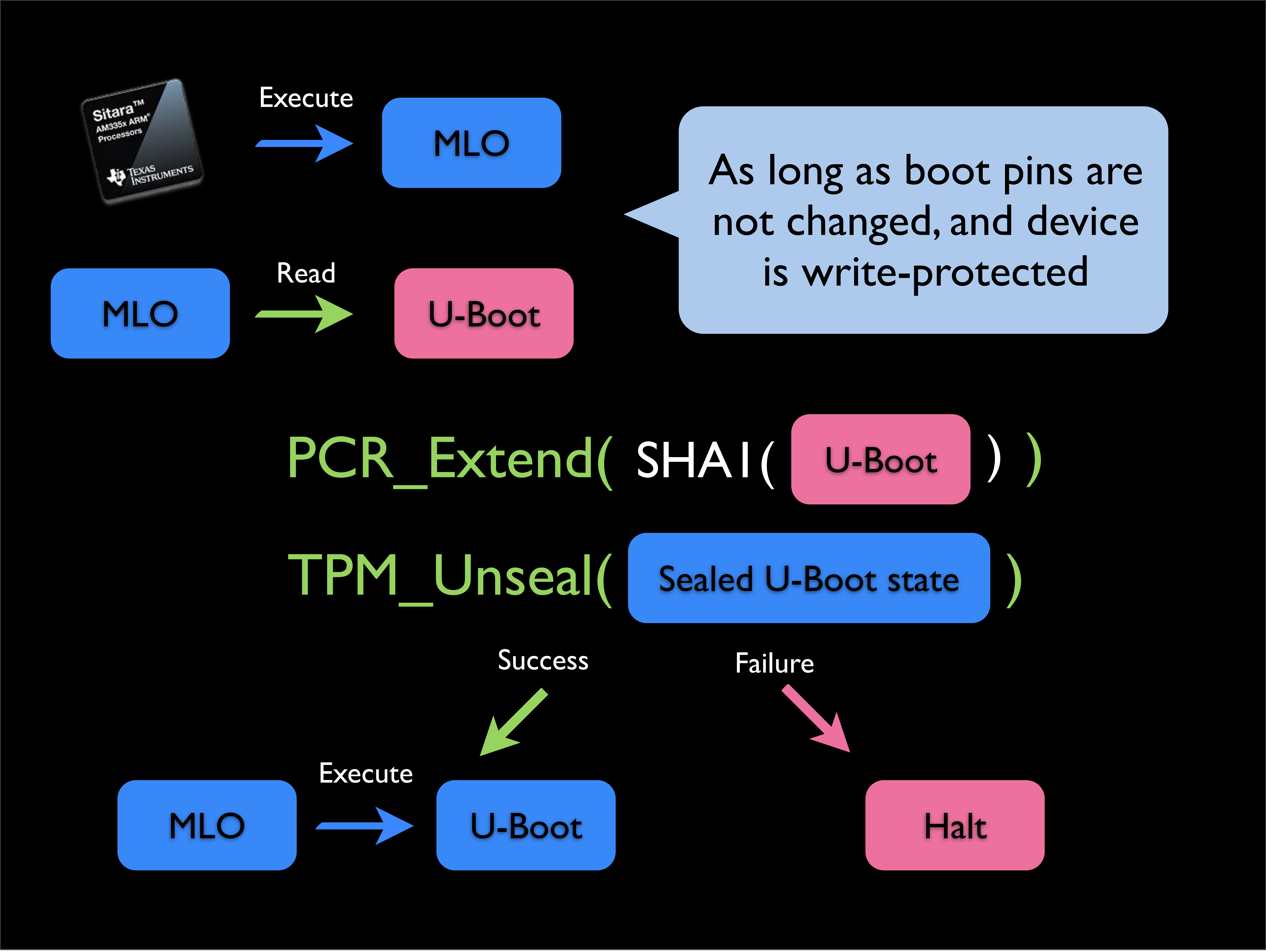
Execute → MLO

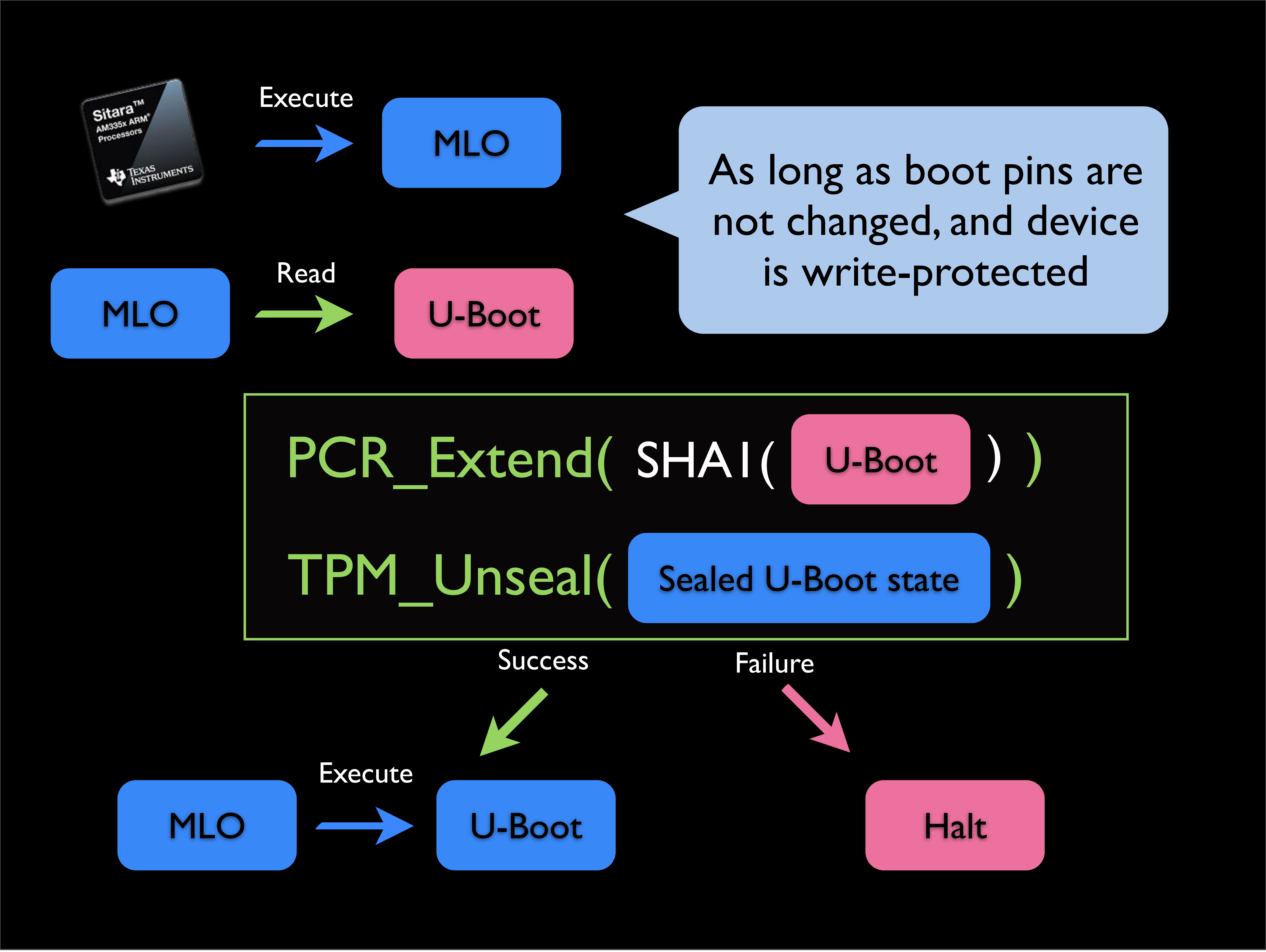
MLO → Read → U-Boot

As long as boot pins are not changed, and device is write-protected

PCR_Extend(SHA1(U-Boot))

TPM_Unseal(Sealed U-Boot state)


Sitara™
AM335x ARM® Processors
Texas Instruments
Execute
MLO
Read
MLO
U-Boot
As long as boot pins are not changed, and device is write-protected
PCR_Extend(SHA1(U-Boot))
TPM_Unseal(Sealed U-Boot state)
Success
Failure
Execute
MLO
U-Boot
Halt


Sitara™
AM335x ARM® Processors
Texas Instruments
Execute
MLO
As long as boot pins are not changed, and device is write-protected
MLO
Read
U-Boot
PCR_Extend(SHA1(U-Boot))
TPM_Unseal(Sealed U-Boot state)
Success
Failure
MLO
Execute
U-Boot
Halt

MLO

1. Initialize TPM: Startup, Selfcheck
2. Verify TPM Configuration

(libSboot, libTLCL, TPM driver)

3. Read U-Boot
4. Extend a PCR with U-Boot hash

5. Read Sealed U-Boot blob
6. Unseal U-Boot blob

Ok, so before we can secure boot, we must Seal a blob for U-Boot

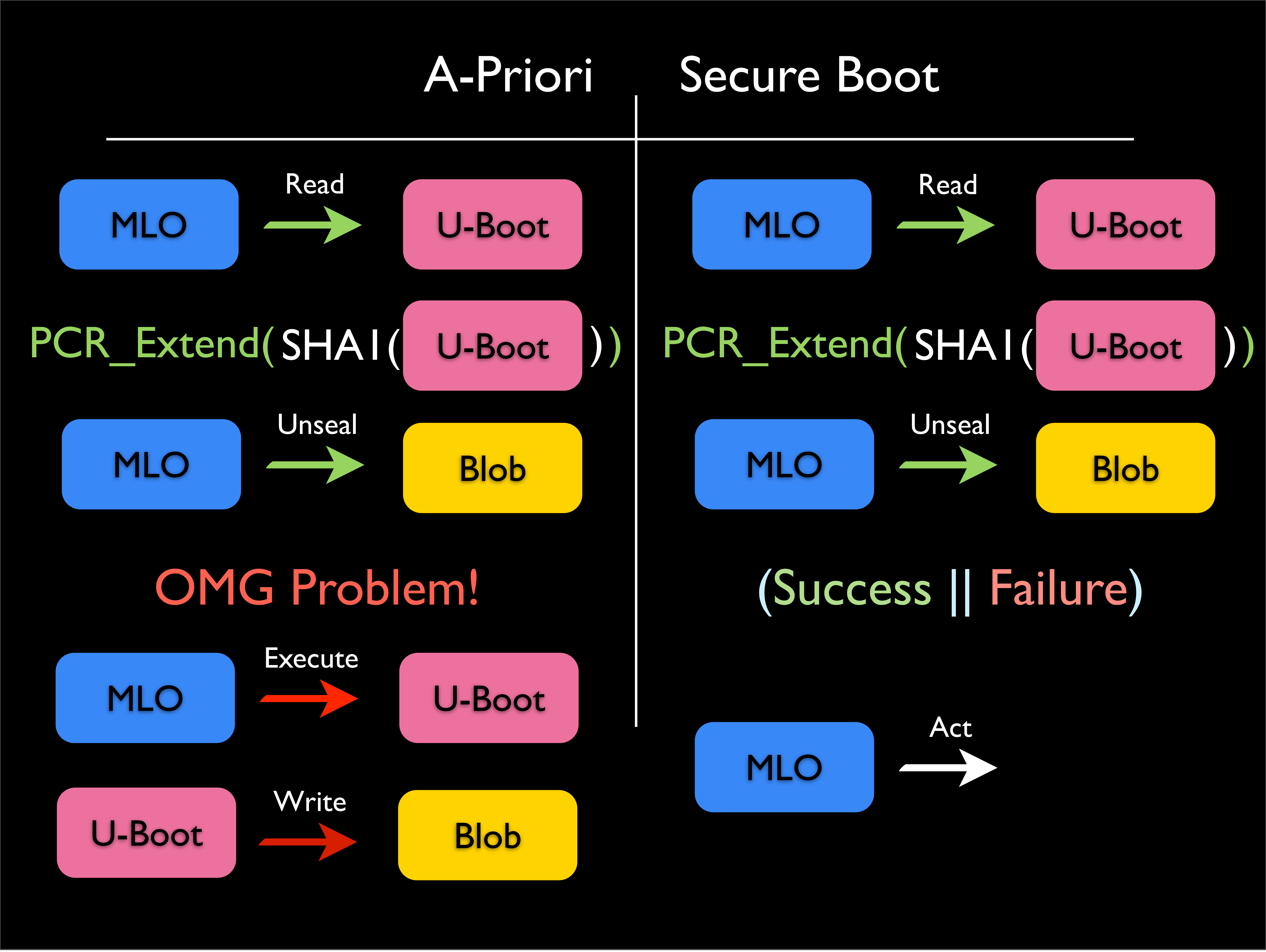
(Where U-Boot is what MLO will eventually execute)

But one more thing...

During the Secure Boot: the second phase loader, called MLO, our SRTM, is verifying that the U-Boot it just read is the expected U-Boot by using the Extended PCR to Seal

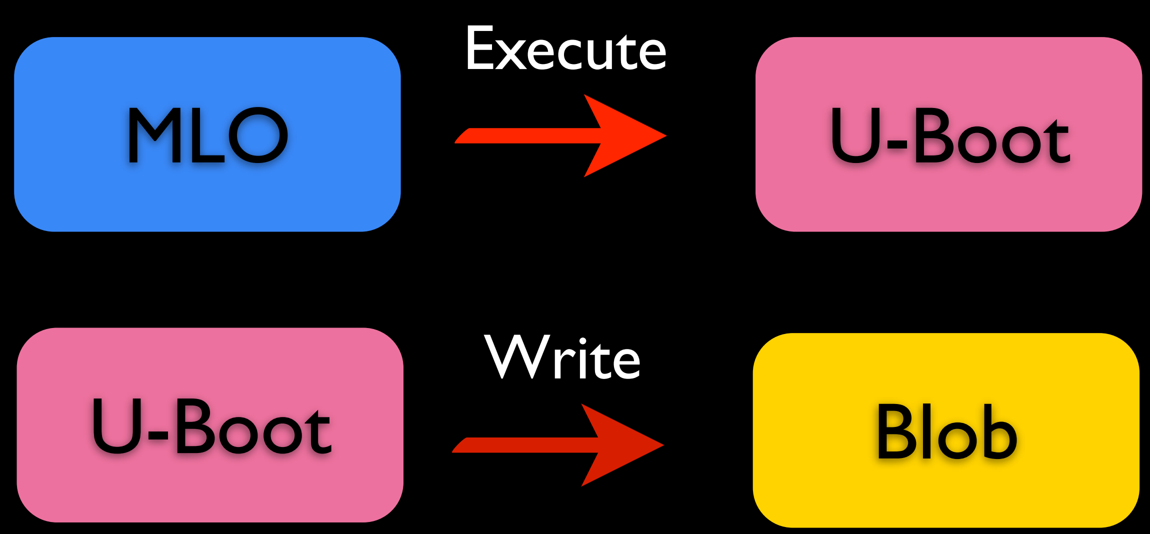
Remember, we enforce state by Sealing to PCRs

This means we must Seal while the PCR is correctly Extended


A-Priori
Secure Boot
MLO
Read
U-Boot
PCR_Extend(SHA1(U-Boot))
MLO
Unseal
Blob
OMG Problem!
MLO
Execute
U-Boot
U-Boot
Write
Blob
MLO
Read
U-Boot
PCR_Extend(SHA1(U-Boot))
MLO
Unseal
Blob
(Success || Failure)
MLO
Act

Compile MLO once to allow U-Boot to execute without verification, then a second time with verification enforced

OMG Problem!



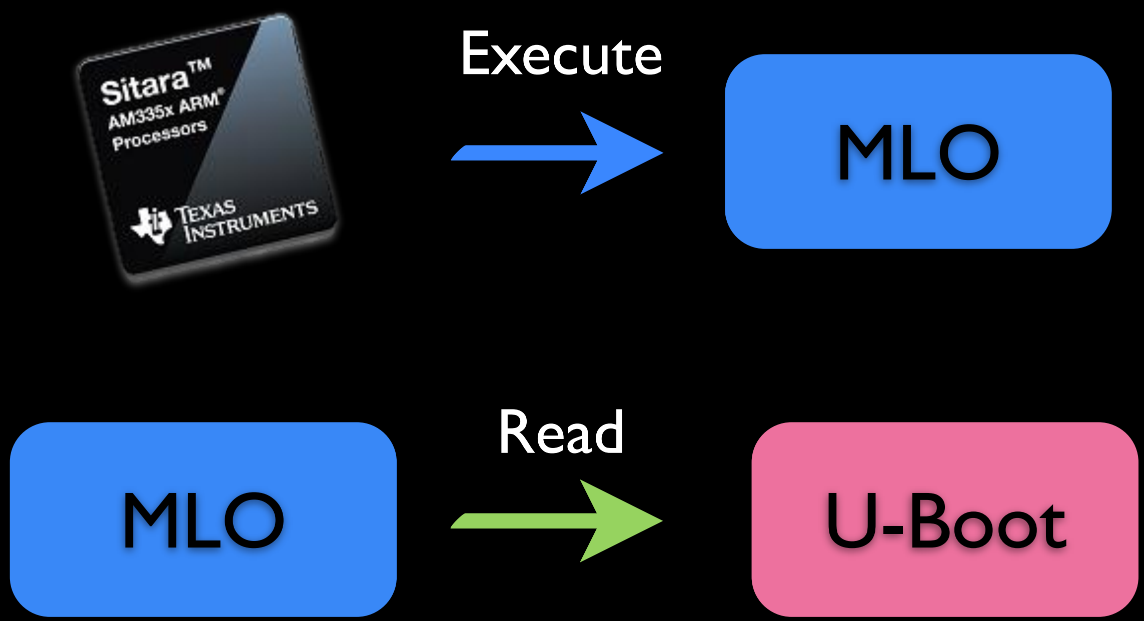
Also: Prevent arbitrary writes using access control on blob storage, in this example we use Physical Presence to enable reading and writing

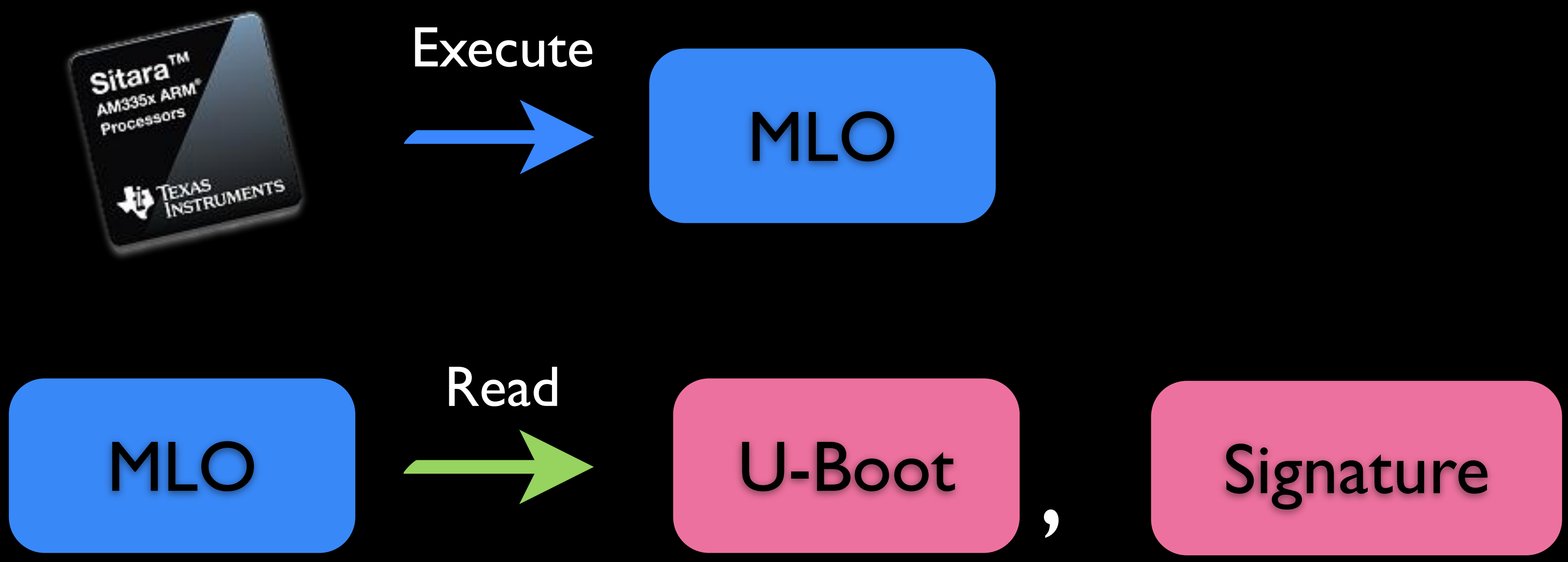
If MLO is enforcing a Secure Boot, changing the U-Boot binary is not possible, even for an expected patch

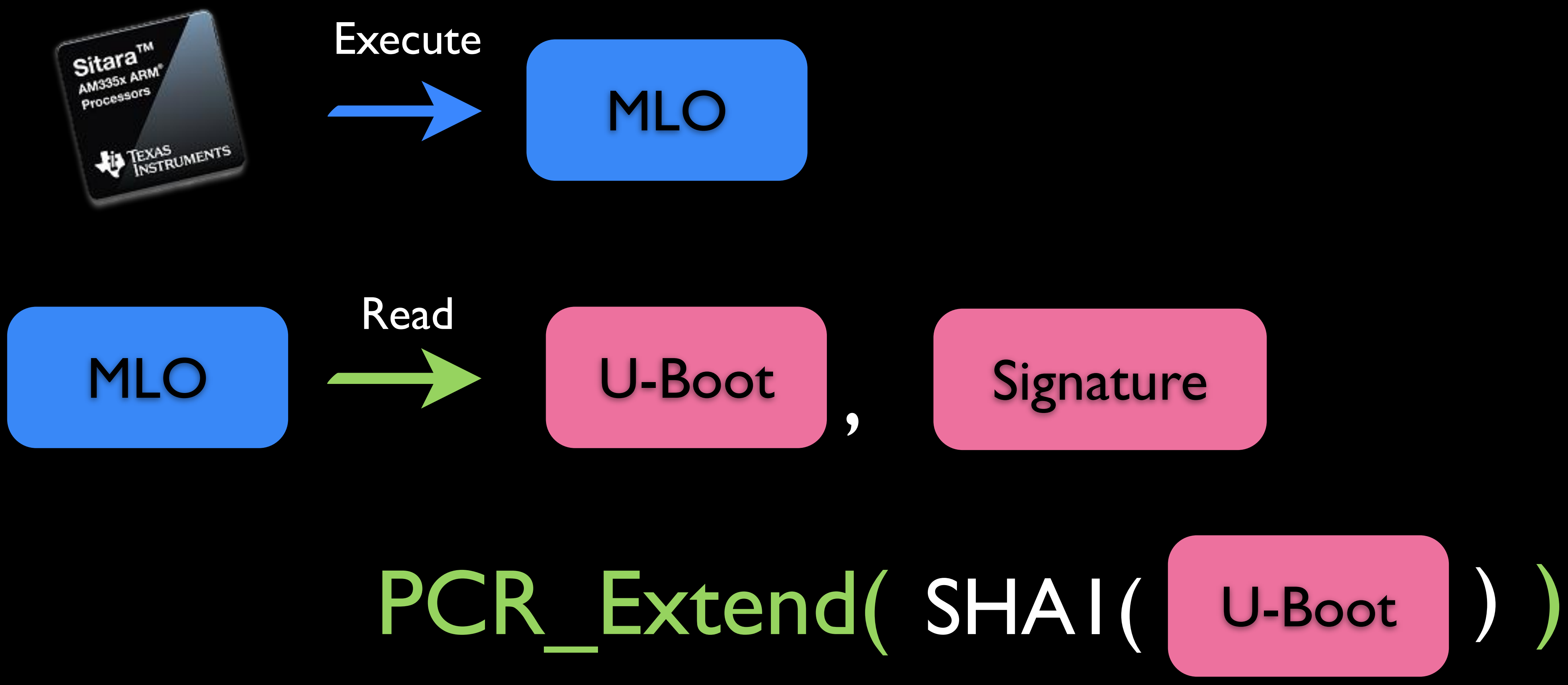
Aside: We use the TPM's NVRAM to store blobs for agnostic storage support and to protect the blob from arbitrary writes

Use the SRTM for a Secure Boot

Implemented with Signatures


Sitara™
AM335x ARM®
Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot


Sitara™
AM335x ARM®
Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
,
Signature

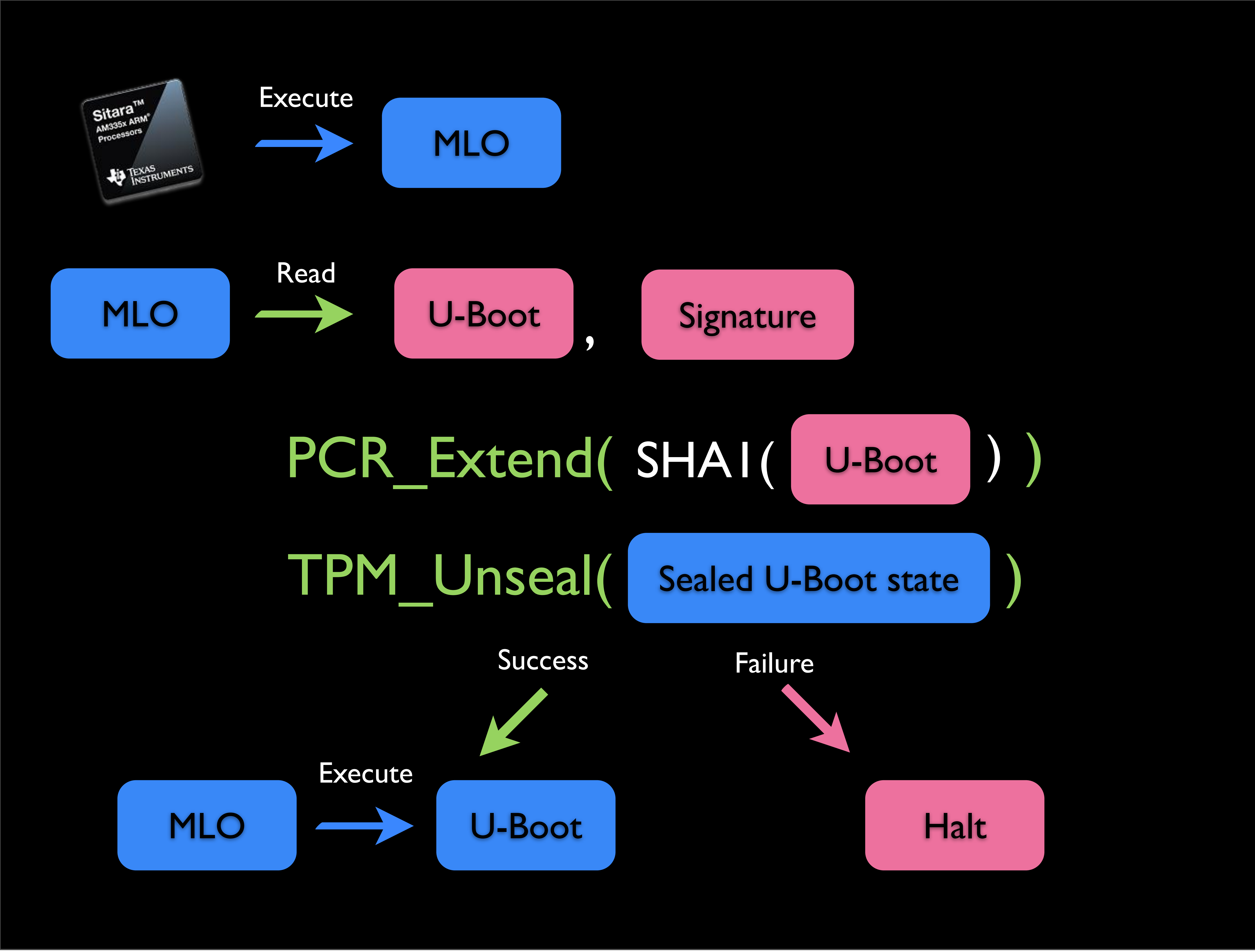

Sitara™ AM335x ARM® Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
,
Signature
PCR_Extend(SHA1(U-Boot))

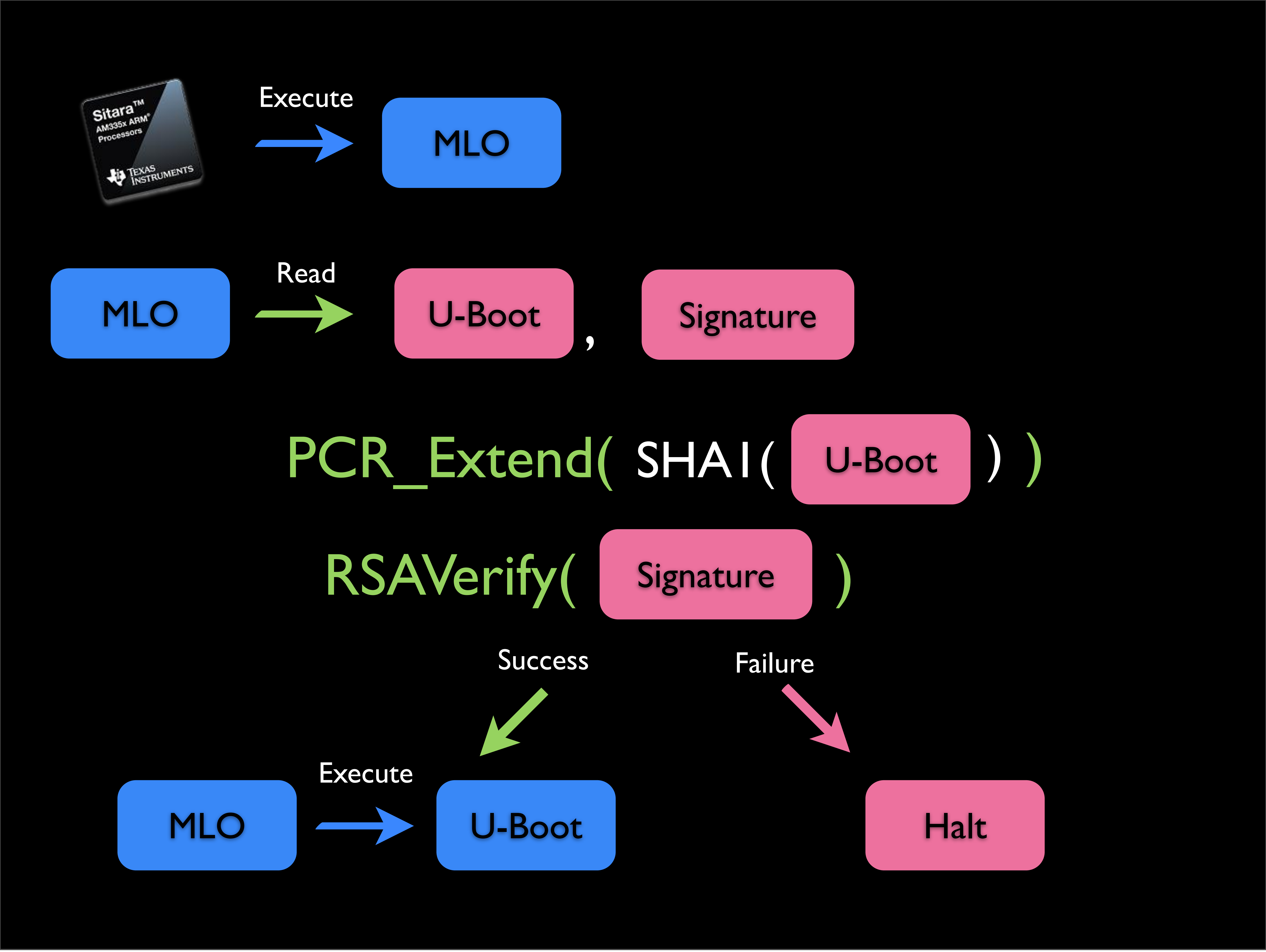
Execute → MLO

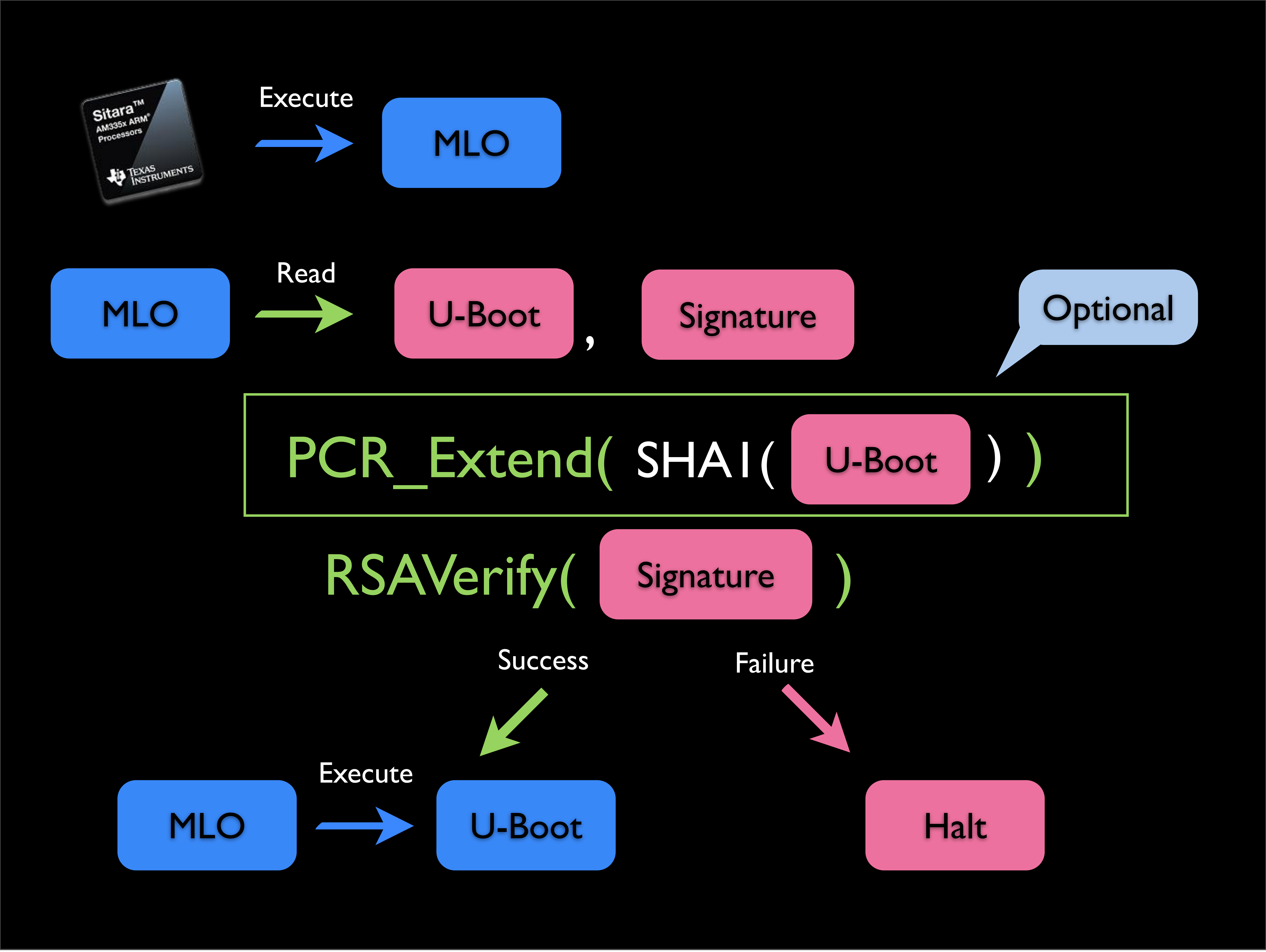
MLO → Read → U-Boot , Signature

PCR_Extend(SHA1(U-Boot))

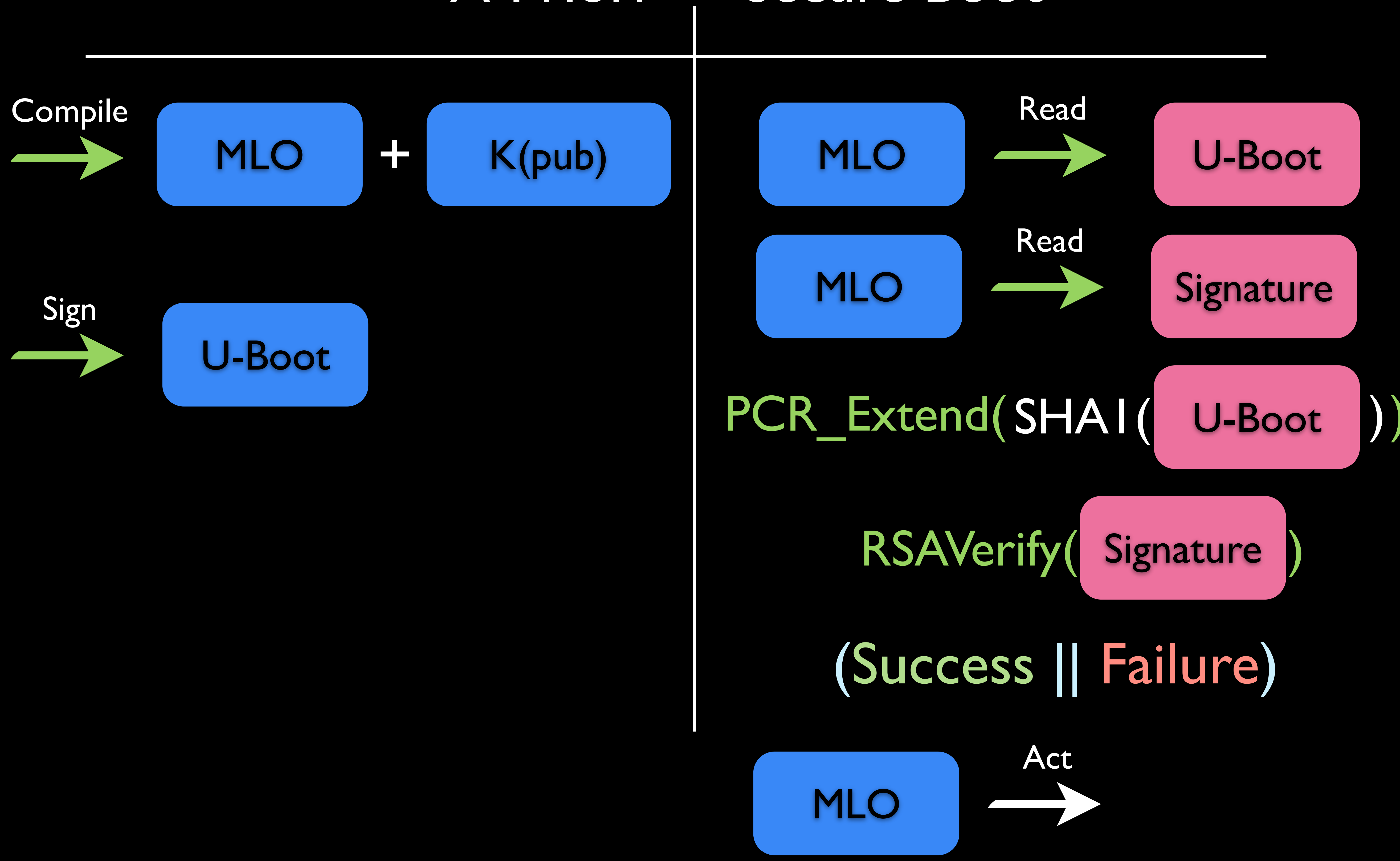
TPM_Unseal(Sealed U-Boot state)


Sitara™
AM335x ARM®
Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
,
Signature
PCR_Extend(SHA1(U-Boot))
TPM_Unseal(Sealed U-Boot state)
Success
Failure
MLO
Execute
U-Boot
Halt


Sitara™
AM335x ARM® Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
,
Signature
PCR_Extend(SHA1(U-Boot))
RSAVerify(Signature)
Success
Failure
MLO
Execute
U-Boot
Halt


Sitara™ AM335x ARM® Processors
Texas Instruments
Execute
MLO
MLO
Read
U-Boot
,
Signature
Optional
PCR_Extend(SHA1(U-Boot))
RSAVerify(Signature)
Success
Failure
MLO
Execute
U-Boot
Halt

A-Priori Secure Boot



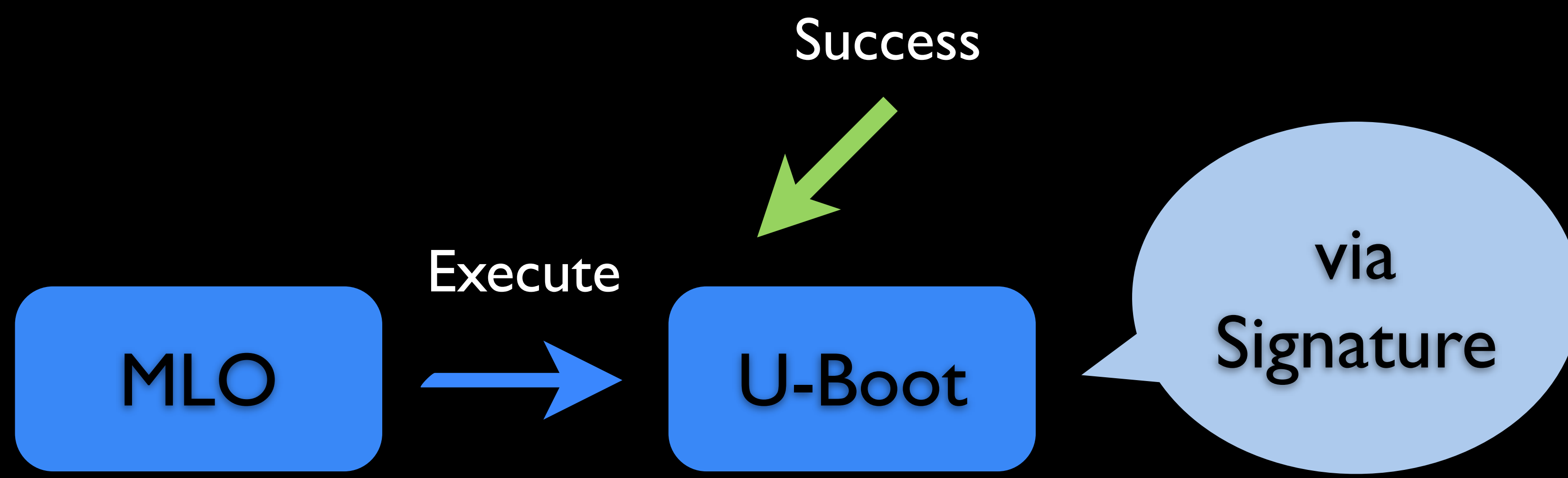
Expected updates to U-Boot will contain a valid signature and not require any change in Secure Boot enforcement

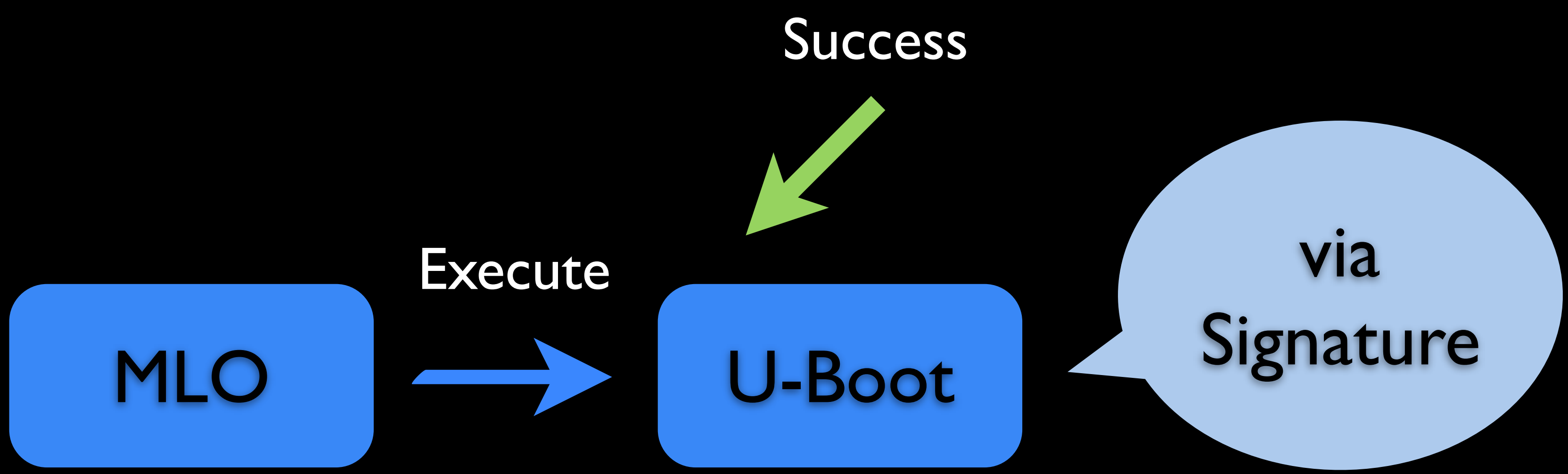
Note: A SRTM using signatures (certificates) does not require a TPM

Well, it really is not a RTM is you are only verifying signatures, it is missing the 'secure-logging' block

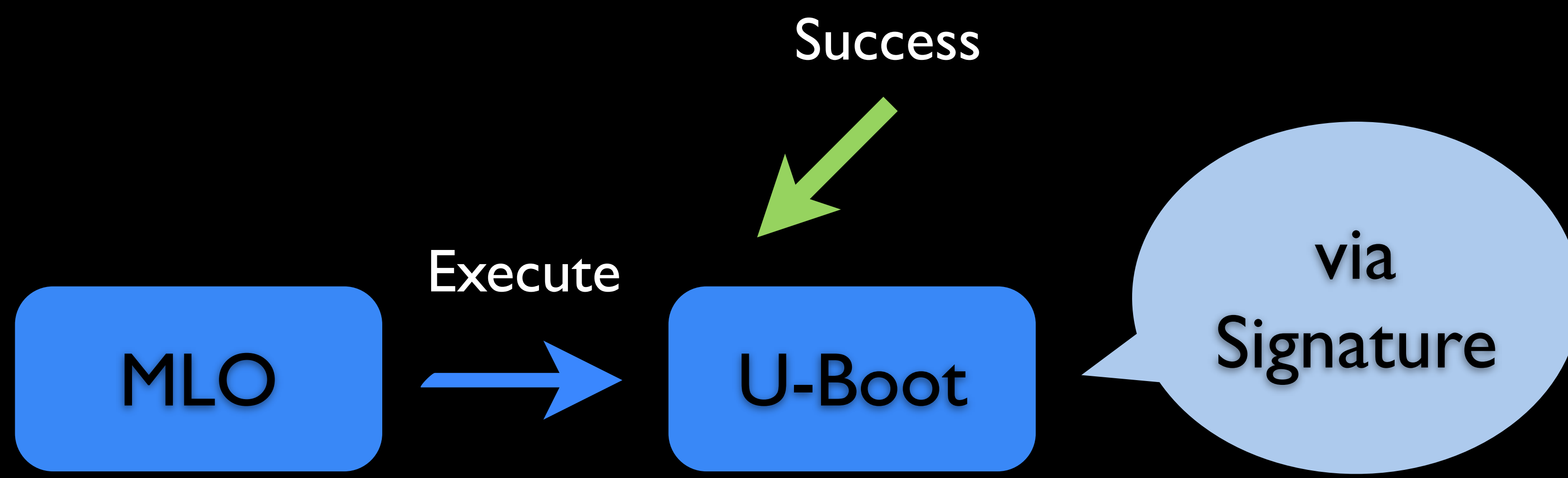
Use the SRTM for a Secure Boot

Implemented with Hashing, Sealing, Unsealing and Signatures


Success
Execute
MLO
U-Boot
via
Signature

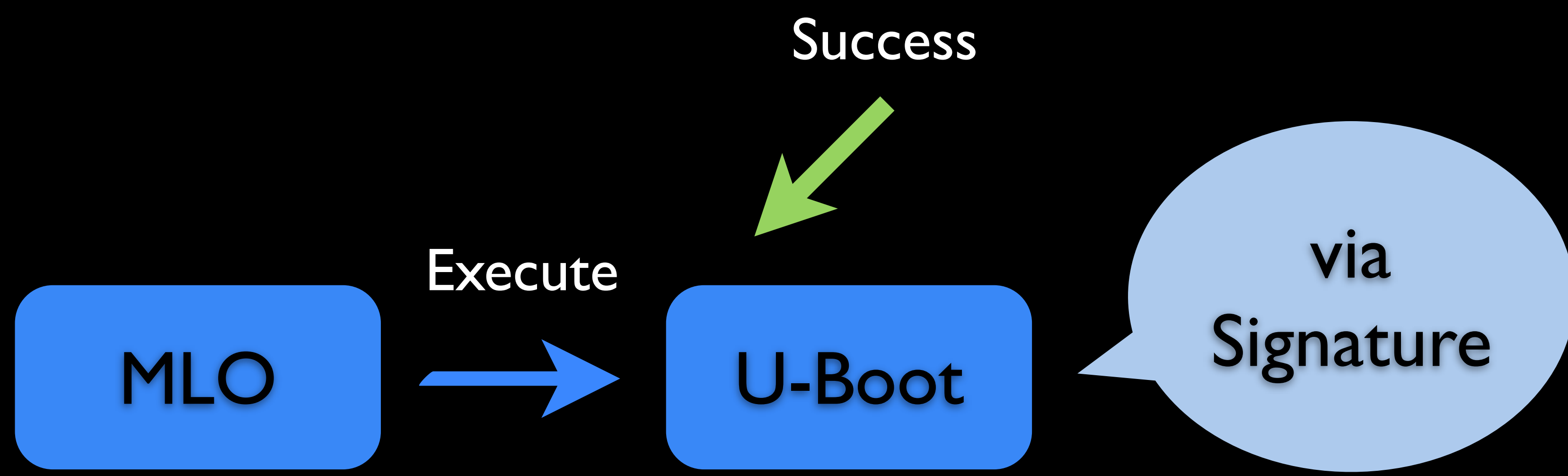


U-Boot #> _



U-Boot #> _

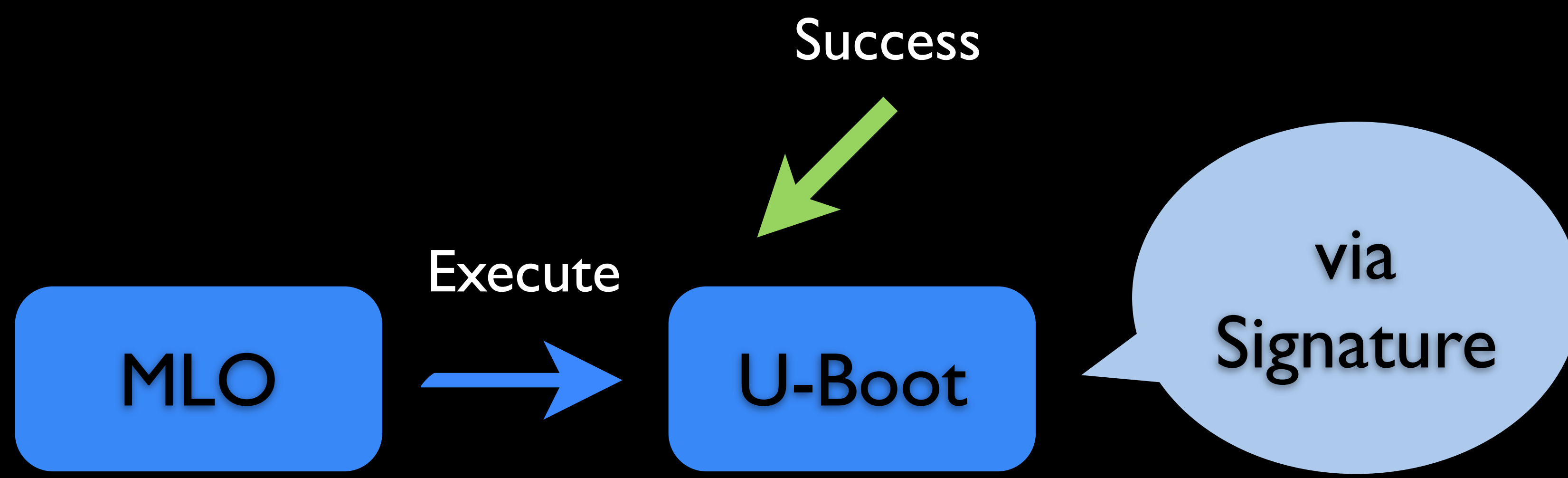
U-Boot #> fatload mmc 0 code.bin 80008000 *



U-Boot #> _

U-Boot #> fatload mmc 0 code.bin 80008000 *

U-Boot #> envset bootargs root=/dev/nfs rw
nfsroot=172.17.77.175:/export/rootfs

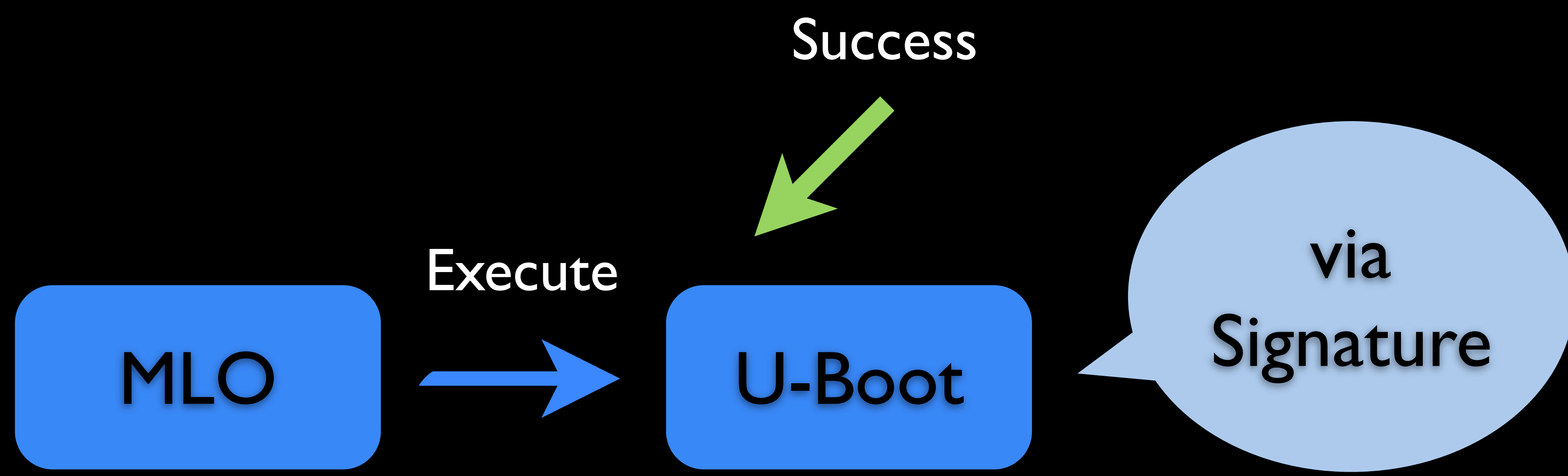


U-Boot #> _

U-Boot #> fatload mmc 0 code.bin 80008000 *

U-Boot #> envset bootargs root=/dev/nfs rw nfsroot=172.17.77.175:/export/rootfs

PCR_Extend(SHA1(CMD, ENV))



U-Boot #> _

U-Boot #> fatload mmc 0 code.bin 80008000 *

U-Boot #> envset bootargs root=/dev/nfs rw
nfsroot=172.17.77.175:/export/rootfs

PCR_Extend(SHA1(CMD, ENV))

For every command, and again for env modifications

Finally, repeat the process for the kernel, ramdisk, and flattened device tree using a separate sealed blob, or appropriate signatures

There are other ways to execute code in U-Boot, we aim to protect any path leading to execution of a kernel from U-Boot

Assure measurement before any possible JMP

libSboot

- Simple example of a Secured Boot
- Implemented in U-Boot
- Modeled loosely after Chromium's vboot
- Many more features coming
  http://github.com/theopolis/u-boot-sboot

Continuing Measurement

Linux Integrity Measurement Architecture

Appraisal

Reporting

Kernel

Integrity Log

```
/sys/kernel/security/ima/ascii_runtime_measurements
```

```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a ld.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f7 libselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d0 libc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c35 libpcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc7f libdl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34592 config
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a8f2 libsepol.so.1
```

PCR, SHA1(file + name), Subsystem, SHA1(content), hint

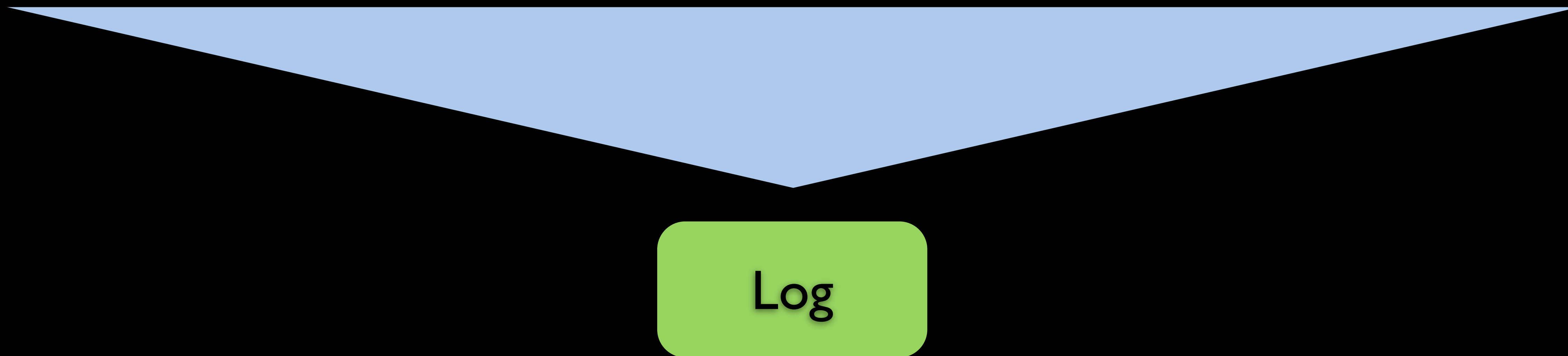
Integrity Log

/sys/kernel/security/ima/ascii_runtime_measurements

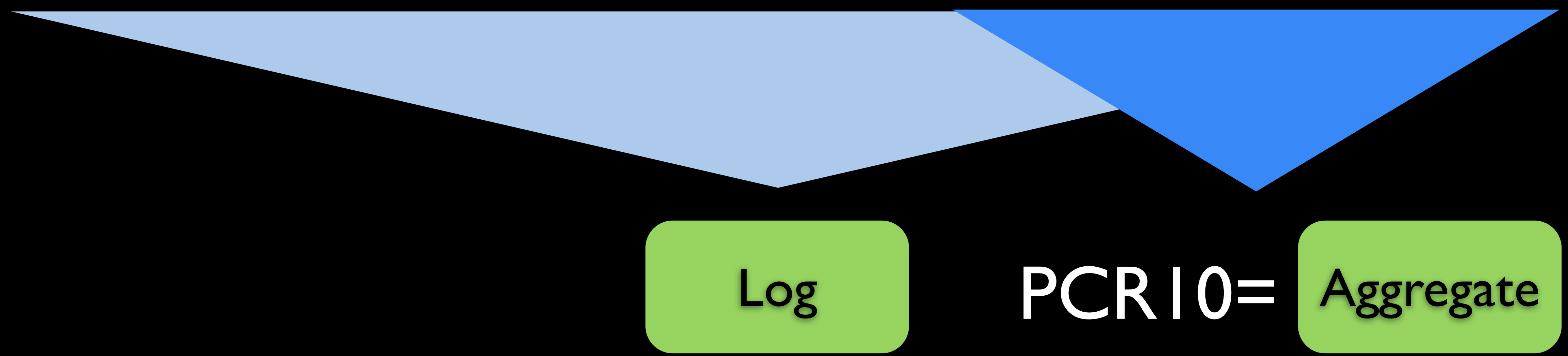
```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a d.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f ibselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d bc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c pcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc dl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34 fig
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a epol.so.1
```

10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima
8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate

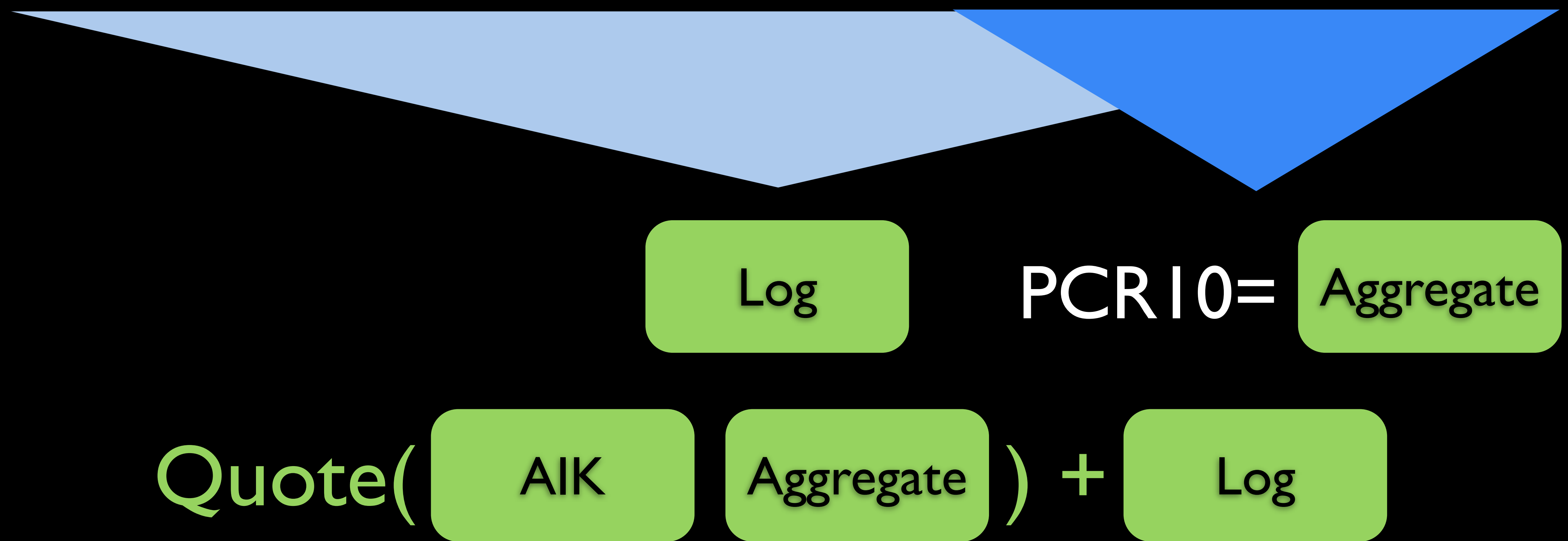
```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a ld.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f7 libselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d0 libc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c35 libpcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc7f libdl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34592 config
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a8f2 libsepol.so.1
```



```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a ld.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f7 libselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d0 libc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c35 libpcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc7f libdl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34592 config
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a8f2 libsepol.so.1
```



```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a ld.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f7 libselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d0 libc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c35 libpcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc7f libdl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34592 config
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a8f2 libsepol.so.1
```



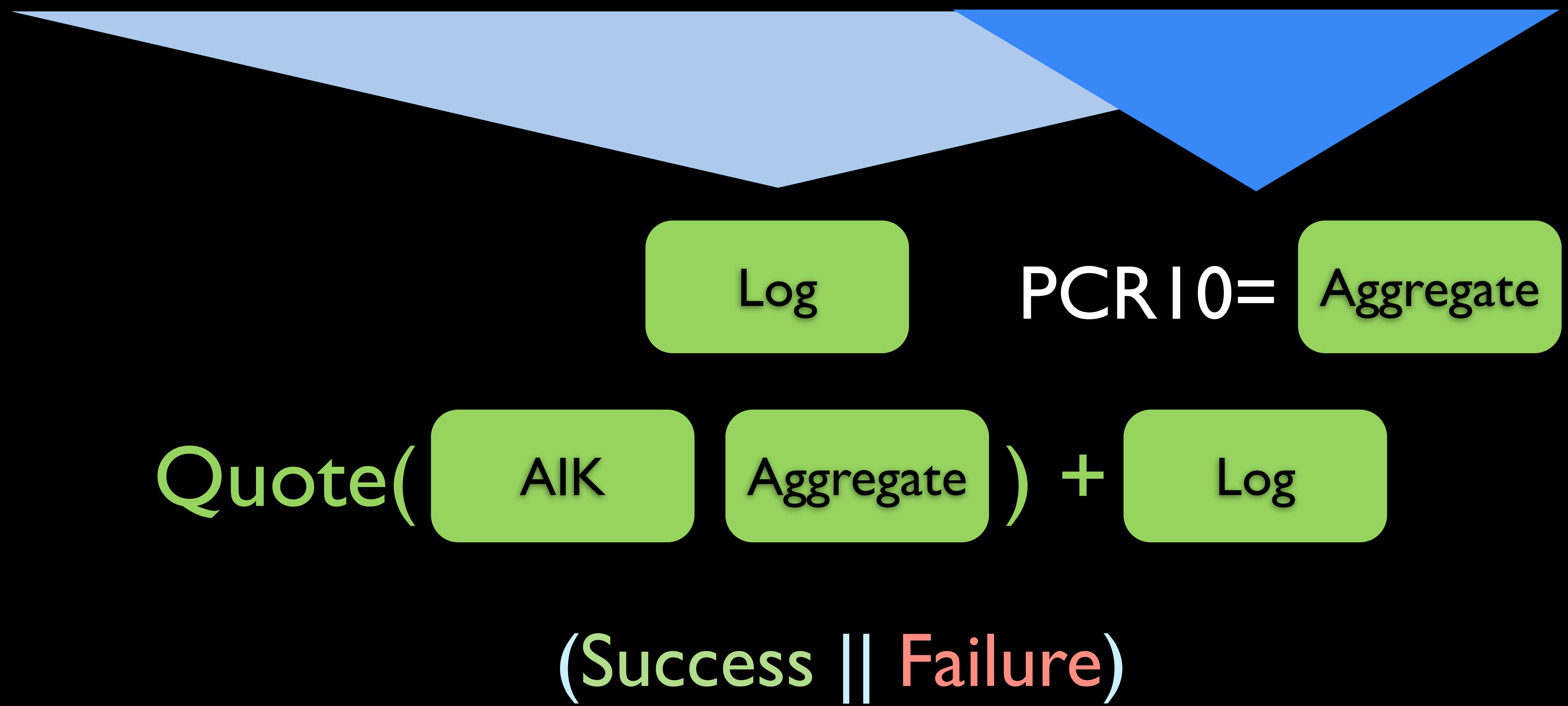
```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a ld.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f7 libselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d0 libc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c35 libpcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc7f libdl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34592 config
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a8f2 libsepol.so.1
```

Log PCR10= Aggregate

Quote(AIK Aggregate) + Log

(Success || Failure)

```
10 3772aaa767c90b2361cef5f56b2ef1bd4efbd349 ima 8b3f2772dec8248c25ef12ed130a7c52986f4a65 boot_aggregate
10 dc99efa590c706a43792618dde88c590a6942ec7 ima fe932380326d7c51d17bac45f5d1c9f576d19f6c /sbin/init
10 fcaa7505fae70096cb9b6a8ec06ec6400b756aa2 ima 0ddd922ae7f5a6dcf788438db1fe47e9a0641e6d ld-2.15.so
10 501975777299919e49aac14c262d6388eae38e79 ima 8d848950517879e0dd77dc9602cad294b454b05a ld.so.cache
10 195830b88844db79ff994c57022e94da416c486c ima 28c4c3a750f5679b9092b2bb2f98c5f745e422f7 libselinux.so.1
10 770cd9400624a5678da388545df1297e182ccd10 ima 03db374e3cedeaf987db096a034bccb5c5bcf3d0 libc-2.15.so
10 82d48ec5fc4344a18a9d17ec1bf1bd8511f99fe6 ima e801e50a5f3ce7acc6e39b1133bce04120c46c35 libpcre.so.1.0.1
10 81ee4b0bbf4f5b464135e3e3d79b2777bceaa236 ima 869231d2fe1afe45ab284adc0efe5a237509bc7f libdl-2.15.so
10 67f5923749dfa266721ee0d6ad038102297c1170 ima e5f8003967fd31f295a115e1d682dd0169b34592 config
10 24894f13a9def8dd2f18838f04fde4becc184fc3 ima 032663452ea268aa1528bd466dda3738bb59a8f2 libsepol.so.1
```



We can pre-computed possible valid logs

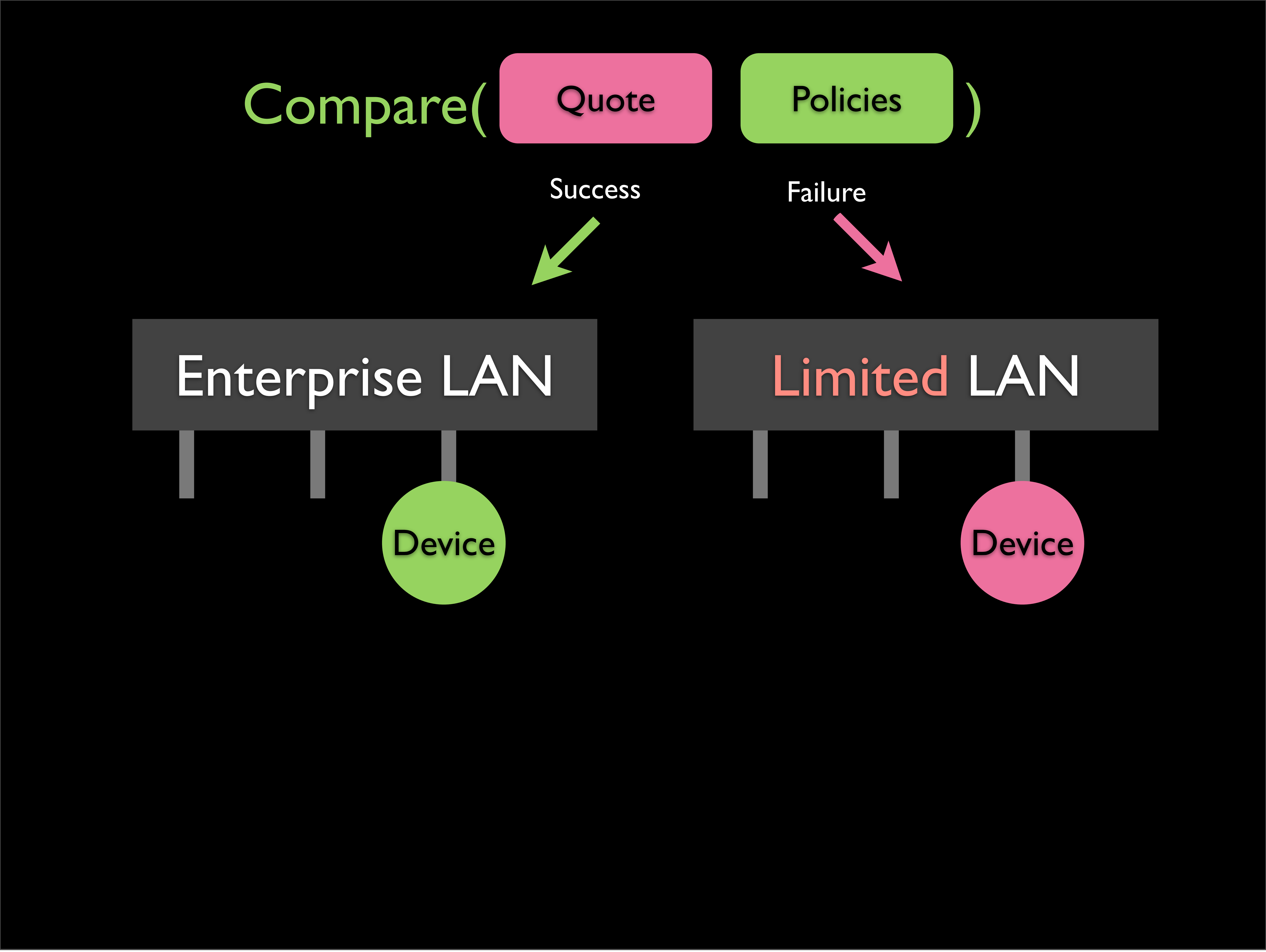
IMA calculates boot aggregate

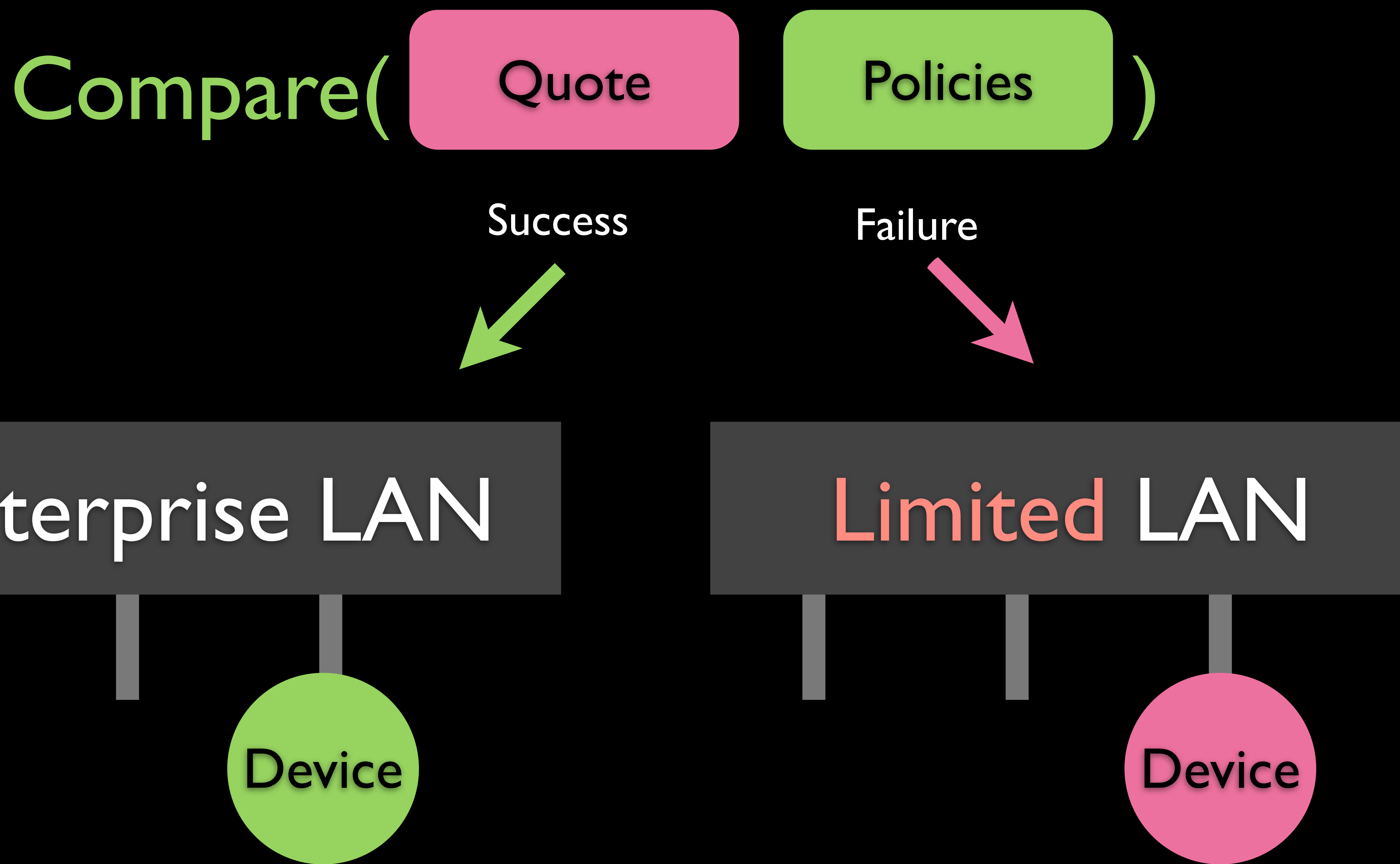
IMA measures each subsequent executable and mmap

OpenPTS quotes and sends run log to trusted third party for appraisal

StrongSwan, Trusted Network Connect Standards, and Network Endpoint Assessment protocols make network access policy decisions based on appraisal

Compare(Quote Policies)


Compare(Quote Policies)
Success
Failure
Enterprise LAN
Device
Limited LAN
Device



We can make local policy decisions too!

IMA only measures by default

With Linux 3.7, IMA Appraisal extensions are included:

```
(a)IMA-Appraisal-Signature-Extension
(b)IMA-Appraisal-Directory-Extension
```

For all Files:

security.ima := Hash(File{i})

(a) Sign(Hash(File{i}))

(b) HMAC(File{i}'s metadata++)

Wait, where did you get that key?

We need an HMAC to protect metadata, because we make expected changes

The HMAC is protecting against offline attacks

So..., where did you get that key?

Linux Trusted and Encrypted Keys!

Use the TPM to seal symmetric keys to state*

Linux Encryption Keys can be used without a TPM

Linux uses Trusted Keys and the TPM to allow key use when an expected state is measured

Offline retrieval of the Trusted Key is not possible unless the SRTM is bypassed

These keys can be used in other creative ways such as device identity or network data encryption

Part 3: Gaps, Ideas and You

Securing your Embedded Devices: Booting

- A Secured Boot can be used to maintain expected boot options (the embedded bootstrap does not change often while in production)
- **User programmable key stores allow the device owner to decide what firmware/ kernel/etc they want to accept**

Securing your Embedded Devices: Measurement

- Measurement may continue past booting, into the Operating System execution. While measurement will not protect against runtime attacks, it can enforce expected state
- **Expected OS executables and libraries can be pre-processed, along with user-defined update signatures**

Securing your Embedded Devices: Attestation

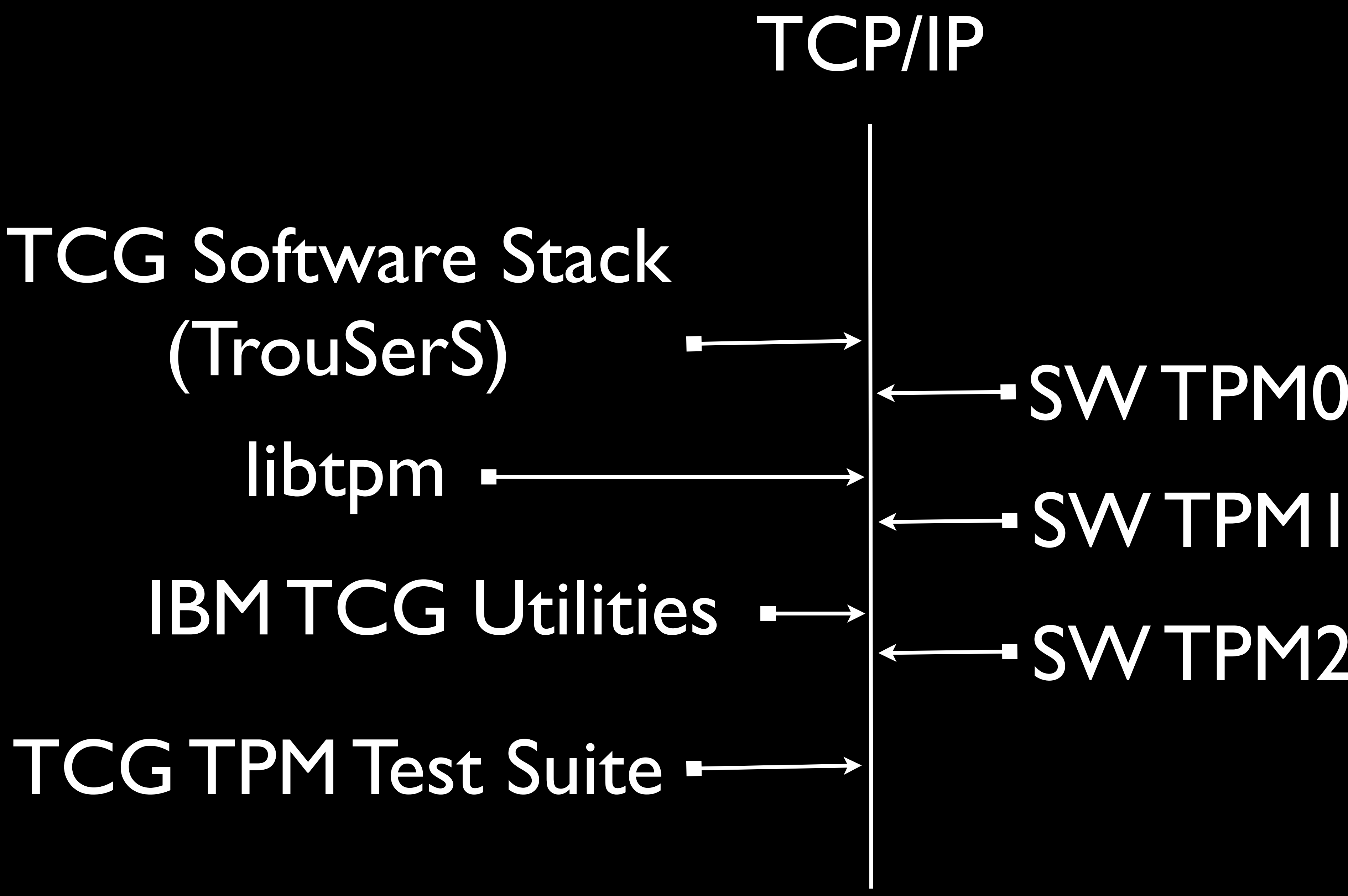
- Anonymous, and Identity-based Attestation allows remote services and protocols to enforce state policy
- **Distributed key infrastructures and trusted parties allow users to attest themselves remotely (remote services can enforce user-defined policys)**

I'm not sure...
I want to test

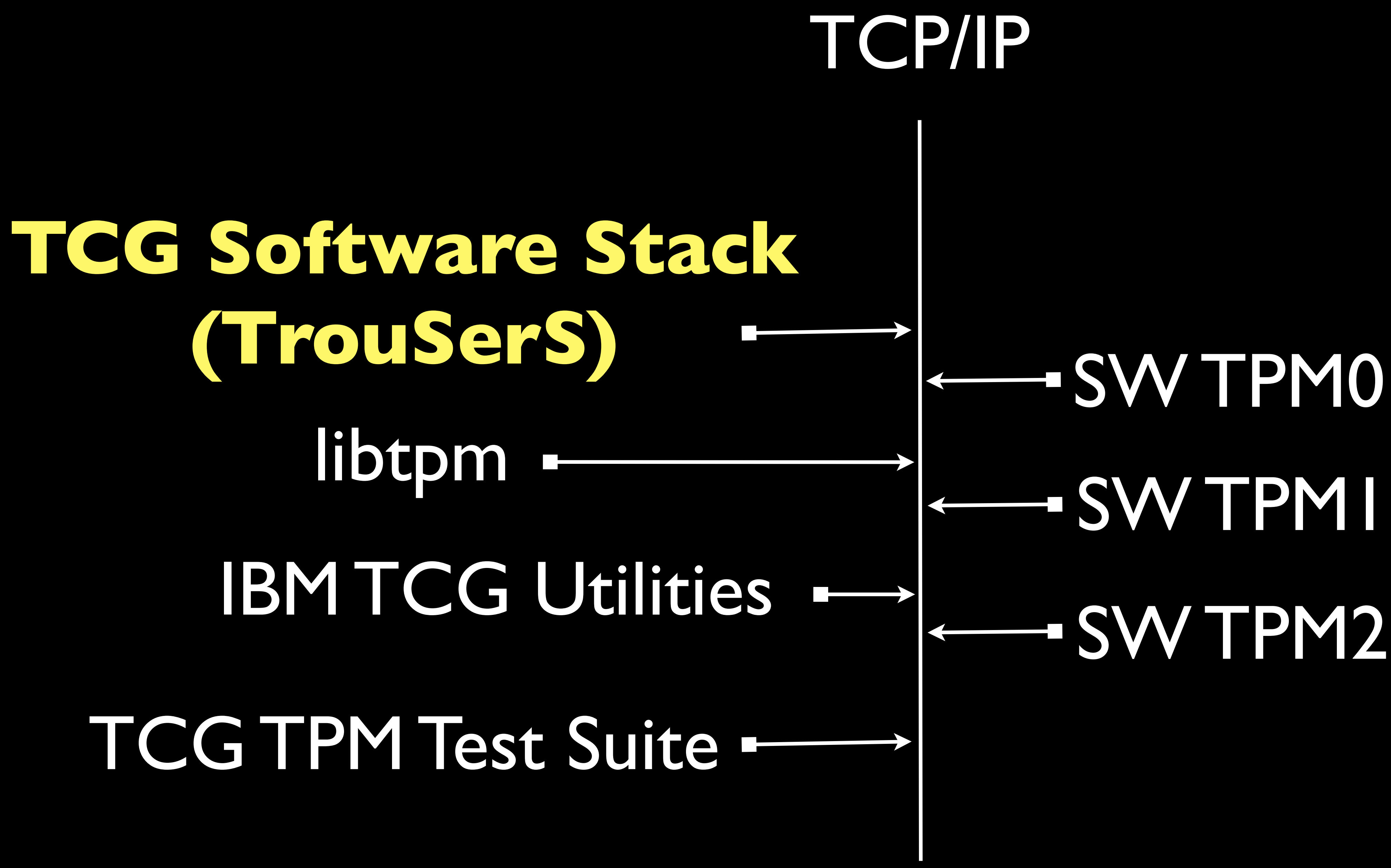
vTPM and XEN

IBM Software TPM

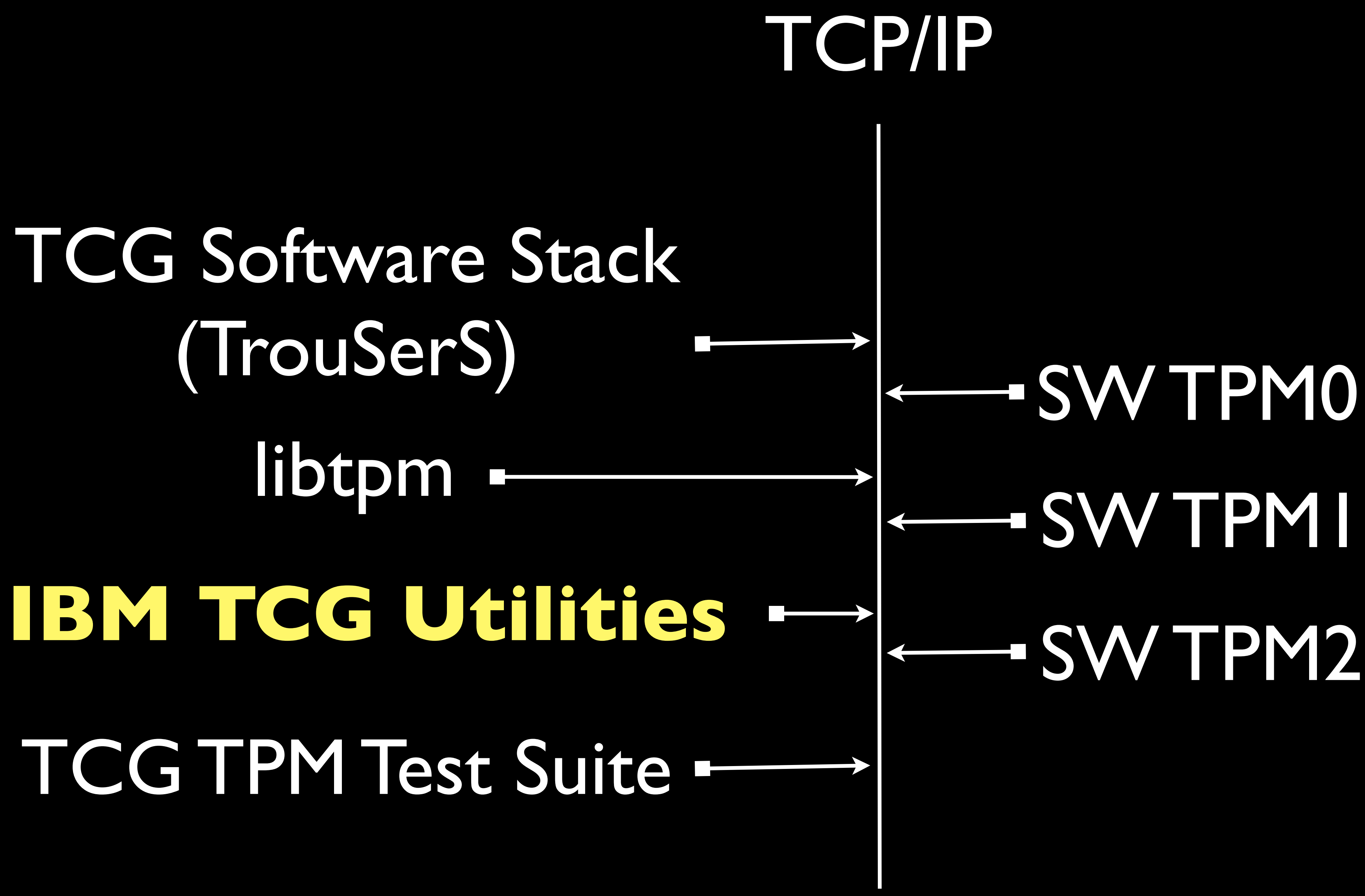
IBM Software TPM



IBM Software TPM



IBM Software TPM



Maybe TC/TPM is an overkill

- Atmel ATSHA204 (newer version of AT88SA102S) enables identification with protected memory
- Allows secure storage for private keys and additional sensitive data
- Does not include crypto functions

Presentation Recap

- Trust criticisms are real but we should be able to offer creative advantages
- Trusted Computing hardware and concepts are available for embedded development
- IMA, OpenPTS, StrongSwan's NEA are already available, we present an example Secure Boot for U-Boot
- More OSS capabilities are needed

TPM Kits

- An Atmel AT97SC3204T (I2C TPM)
- 28 Pin SSOP breakout
- Maxim DS1077LZ-66+ OSC
- 8 Pin SOIC breakout


DIY: Using Trust to Secure Embedded Projects
SHMOOCON IX 2013
http://prosauce.org/shmoo
ET-PCB SSOP28
WWW.ETT.CO.TH
SSOP-28

Questions

???