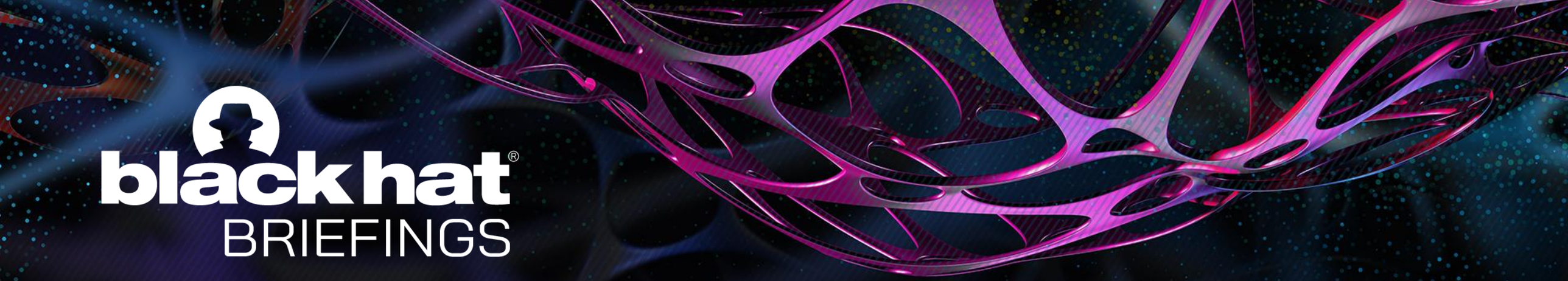


How to Secure Unique Ecosystem Shipping 1 Billion+ Cores?

Adam ‘pi3’ Zabrocki, Marko Mitic

/usr/bin/whoarewe

Private contact:

http://pi3.com.pl
pi3@pi3.com.pl
Twitter: @Adam_pi3

Adam ‘pi3’ Zabrocki:

- NVIDIA (currently – Director of Offensive Security)
  - Leading Offensive Security Research efforts
  - RISC-V (Vice-Chair of J-ext, author: PM, HW CFI, MTE, more)
  - Security architect for GPU and next-gen NVIDIA products

- Phrack author
- Bughunter (Hyper-V, KVM, RISC-V ISA, Intel uCode, Linux kernel, FreeBSD, OpenSSH, Apache, gcc SSP/ProPolice, more) – CVEs
- Creator and a developer of Linux Kernel Runtime Guard (LKRG)
- Speaker at BlackHat, DEF CON, BSides, Confidence, Open-Source Tech more
- The Pwnie Awards nominee (x2)

Private contact:

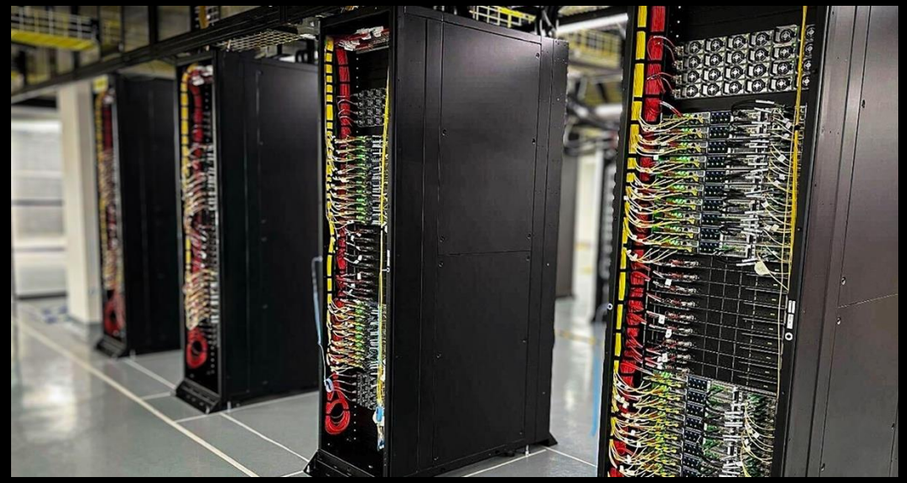
markomitic.net
linkedin.com/markomitic
Twitter: @markomitic

Marko Mitic

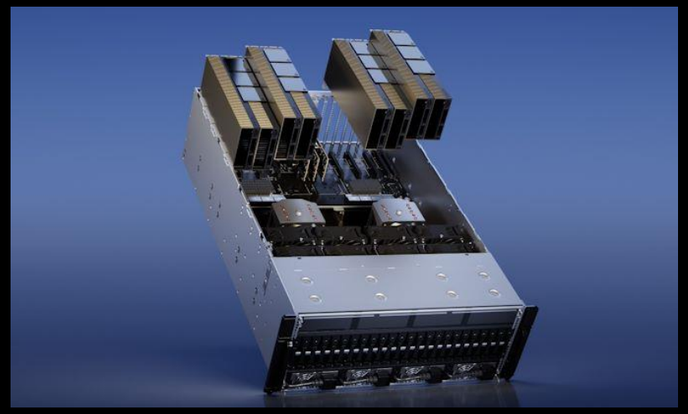
- Software Security Architect & System Software Manager at NVIDIA
- Leads NVIDIA’s Core RISC-V team
- GPU Product Security & Risk Officer, PSIRT lead

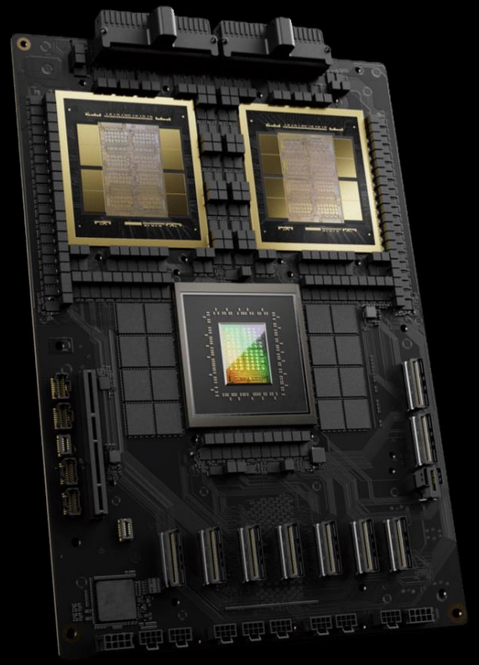
NVIDIA

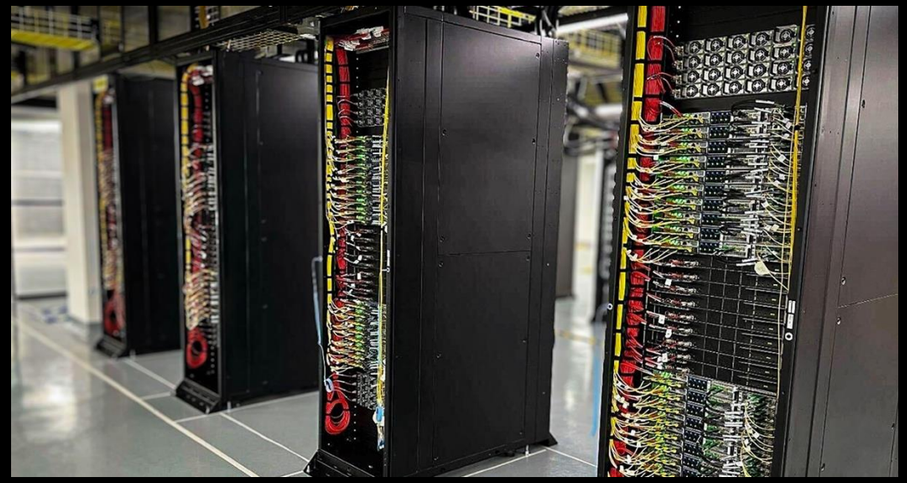
Why this talk?



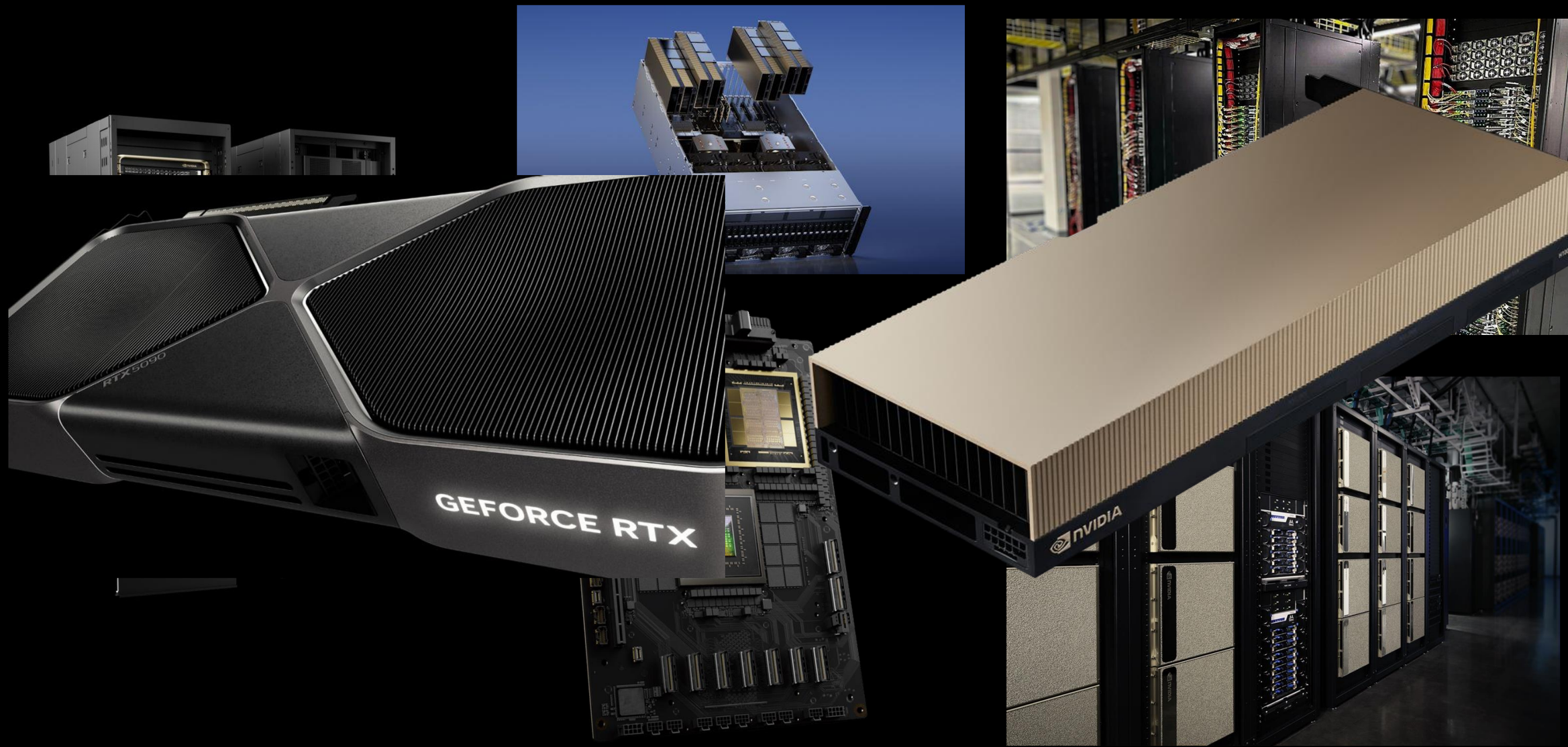
Why this talk?



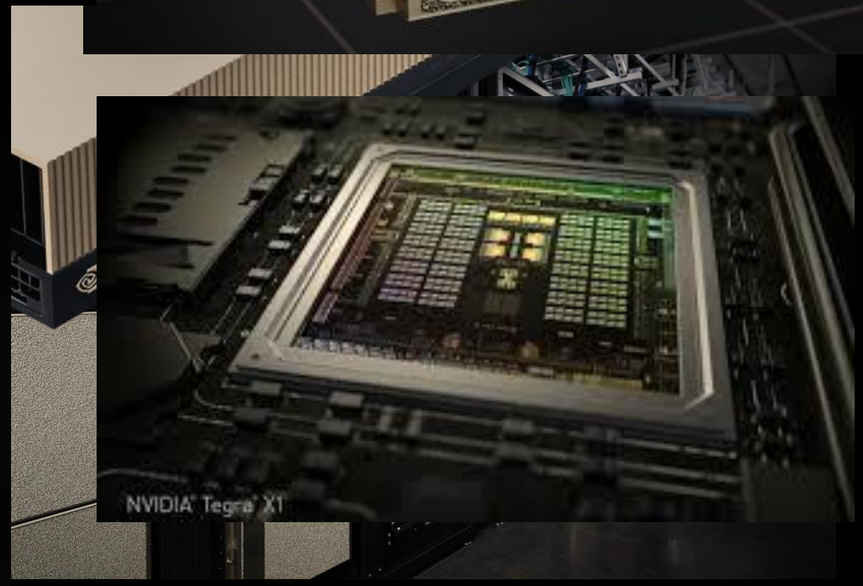




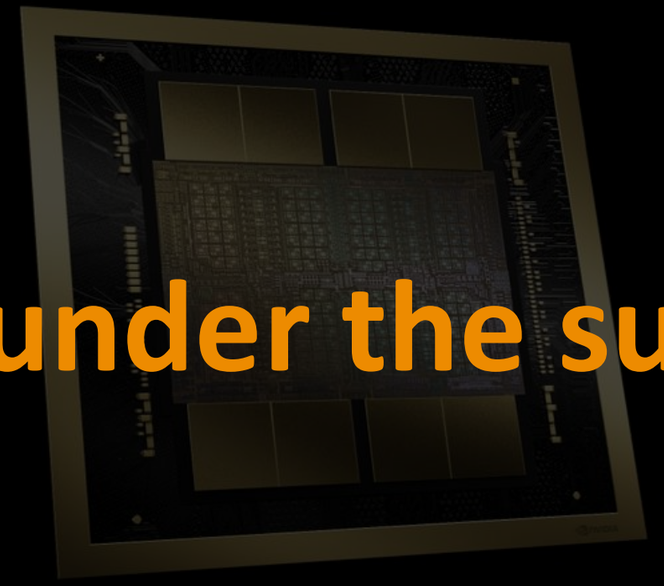
Why this talk?



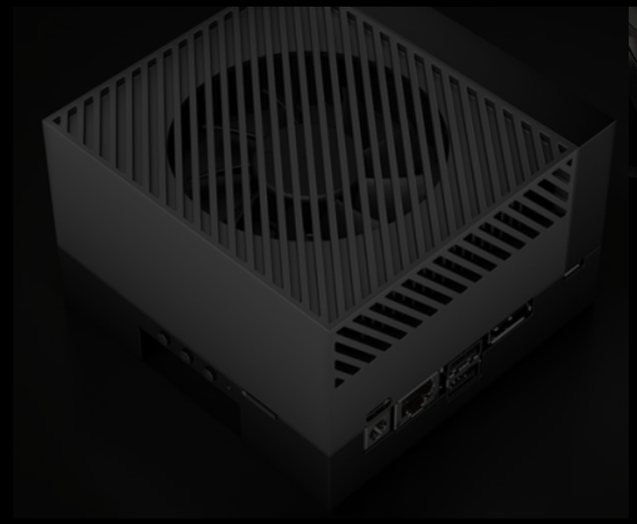
Why this talk?

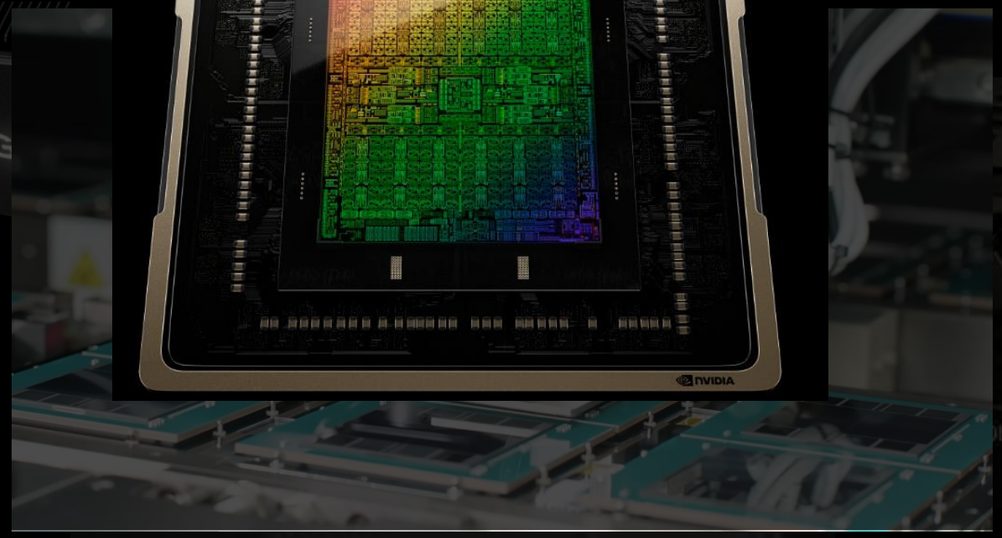


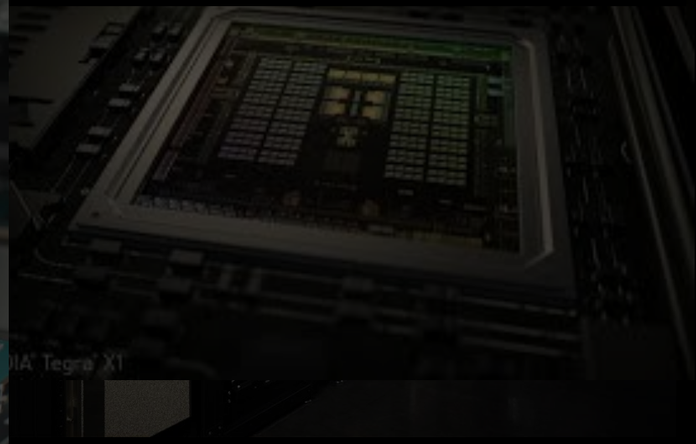
Why this talk?



“There is nothing hidden under the sun”







Why this talk?

- Each NVIDIA chipset may include ~10-50 microcontrollers (MCUs)
  - Function Level Controllers (e.g., Codecs, Memory Controllers, Chip2Chip Interfaces, more)
  - Chip/System Level Control (e.g., Resource Management, PMU, Security, more)
  - Data Processing including packet routing in networking

Why this talk?

- Each NVIDIA chipset may include ~10-50 microcontrollers (MCUs)
  - Function Level Controllers (e.g., Codecs, Memory Controllers, Chip2Chip Interfaces, more)
  - Chip/System Level Control (e.g., Resource Management, PMU, Security, more)
  - Data Processing including packet routing in networking

- Legacy Falcon (internal proprietary RISC ISA) were difficult to scale
  - Sufficient at that time… requirements and expectation changed
  - Security layer needed to be updated to fulfill modern and future(!) expectations

Why this talk?

- Each NVIDIA chipset may include ~10-50 microcontrollers (MCUs)
  - Function Level Controllers (e.g., Codecs, Memory Controllers, Chip2Chip Interfaces, more)
  - Chip/System Level Control (e.g., Resource Management, PMU, Security, more)
  - Data Processing including packet routing in networking
- Legacy Falcon (internal proprietary RISC ISA) were difficult to scale
  - Sufficient at that time… requirements and expectation changed
  - Security layer needed to be updated to fulfill modern and future(!) expectations
- NVIDIA chip must meet the demand
  - Not only AI workloads is booming – NVIDIA processors are crucial
  - Opportunity to redesign the ecosystem
    - In secure manner that will be scalable in the future!

Why RISC-V

Why RISC-V

- Retire proprietary Falcon architecture

Why RISC-V

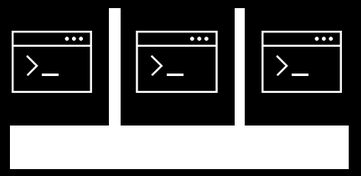
- Retire proprietary Falcon architecture
- Performance

Why RISC-V

- Retire proprietary Falcon architecture
- Performance
- Enable fast & flexible HW/SW co-design, custom extensions

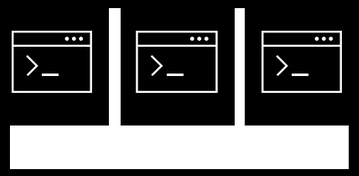
Why RISC-V

- Retire proprietary Falcon architecture
- Performance
- Enable fast & flexible HW/SW co-design, custom extensions
- Layered security isolation primitives



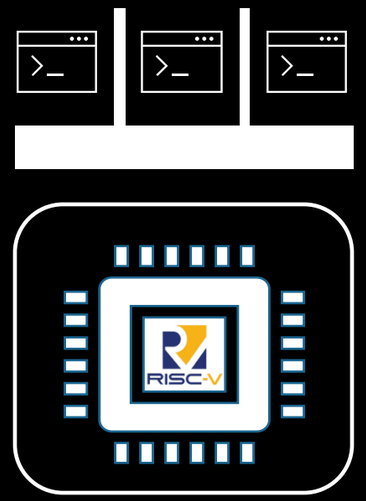
Why RISC-V

- Retire proprietary Falcon architecture
- Performance
- Enable fast & flexible HW/SW co-design, custom extensions
- **Layered security isolation primitives**

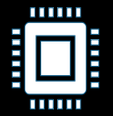


Why RISC-V

- Retire proprietary Falcon architecture
- Performance
- Enable fast & flexible HW/SW co-design, custom extensions
- **Layered security isolation primitives**
- Common configurable foundation for all MCUs across all products



Why RISC-V



Retire proprietary Falcon architecture

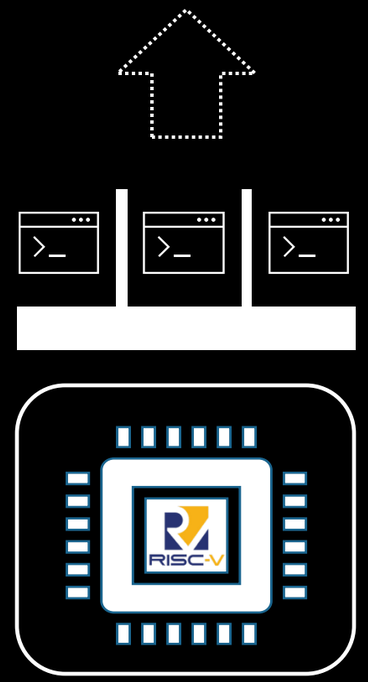
Performance

Enable fast & flexible HW/SW co-design, custom extensions

Layered security isolation primitives

Common configurable foundation for all MCUs across all products

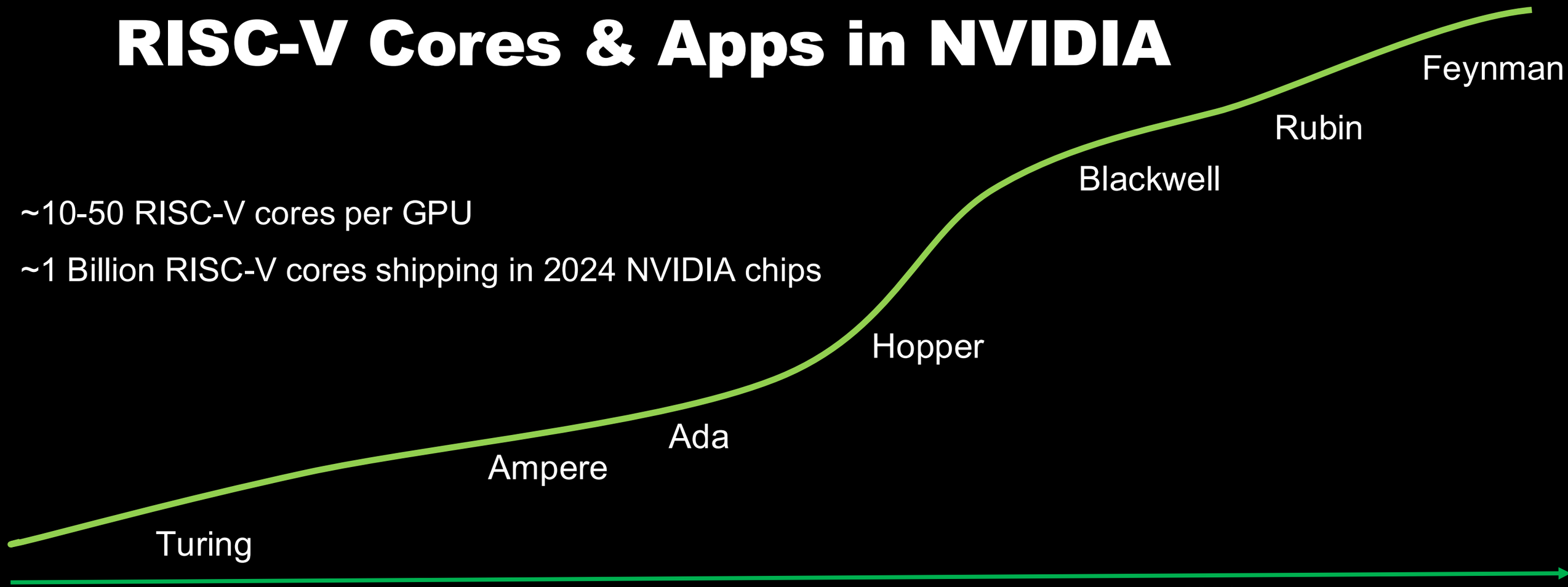
Scale up and out and build only for what is needed



RISC-V Cores & Apps in NVIDIA

~10-50 RISC-V cores per GPU

~1 Billion RISC-V cores shipping in 2024 NVIDIA chips



From Silicon to Software: Foundational elements for Secure Execution

From Silicon to Software: Foundational elements for Secure Execution

- Memory Protection and Isolation
- Hardware Security Mitigations
- BootROM
- TEE
- (TEE) Operating Systems
- Formal Verification
- OSR
- …

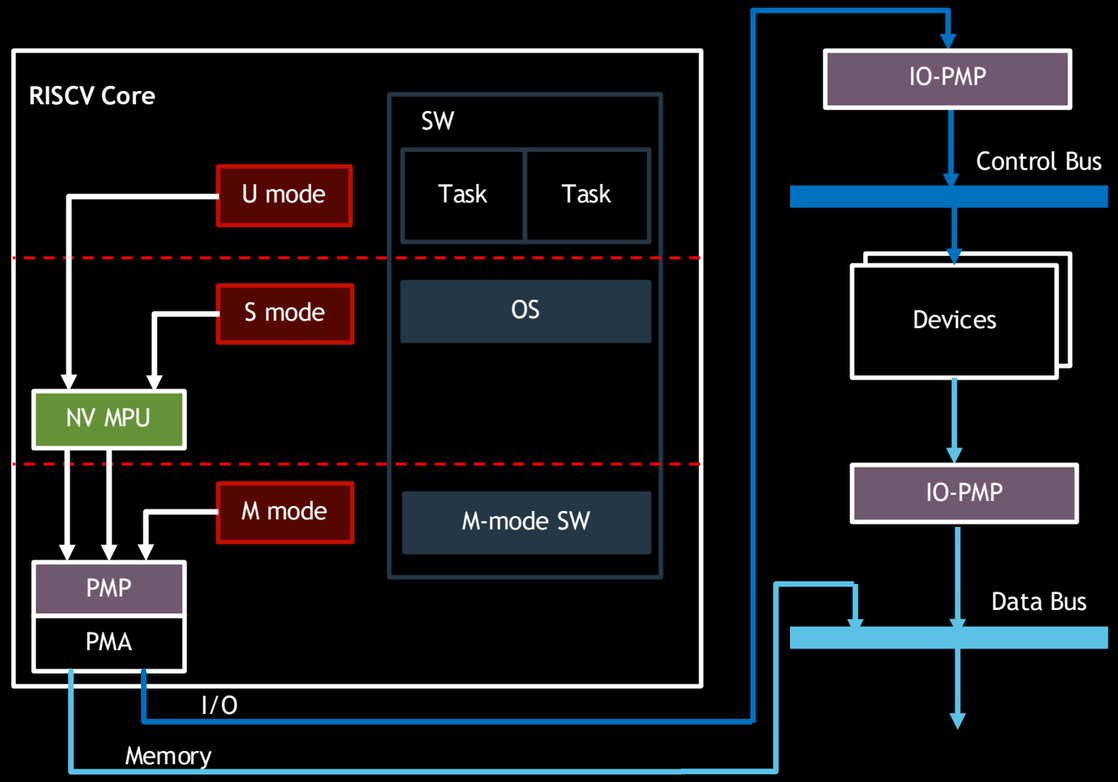
- HW Root of Trust
- Secure Storage
- Crypto Accelerator
- Tamper Detection
- Crypto Libraries
- Key Management
- Secure Software Development Lifecycle
- …

From Silicon to Software: Foundational elements for Secure Execution

- Memory Protection and Isolation
- Hardware Security Mitigations
- BootROM
- TEE
- (TEE) Operating Systems
- Formal Verification
- OSR
- …

- HW Root of Trust
- Secure Storage
- Crypto Accelerator
- Tamper Detection
- Crypto Libraries
- Key Management
- Secure Software Development Lifecycle
- …

RISC-V Intro – Privilege Modes

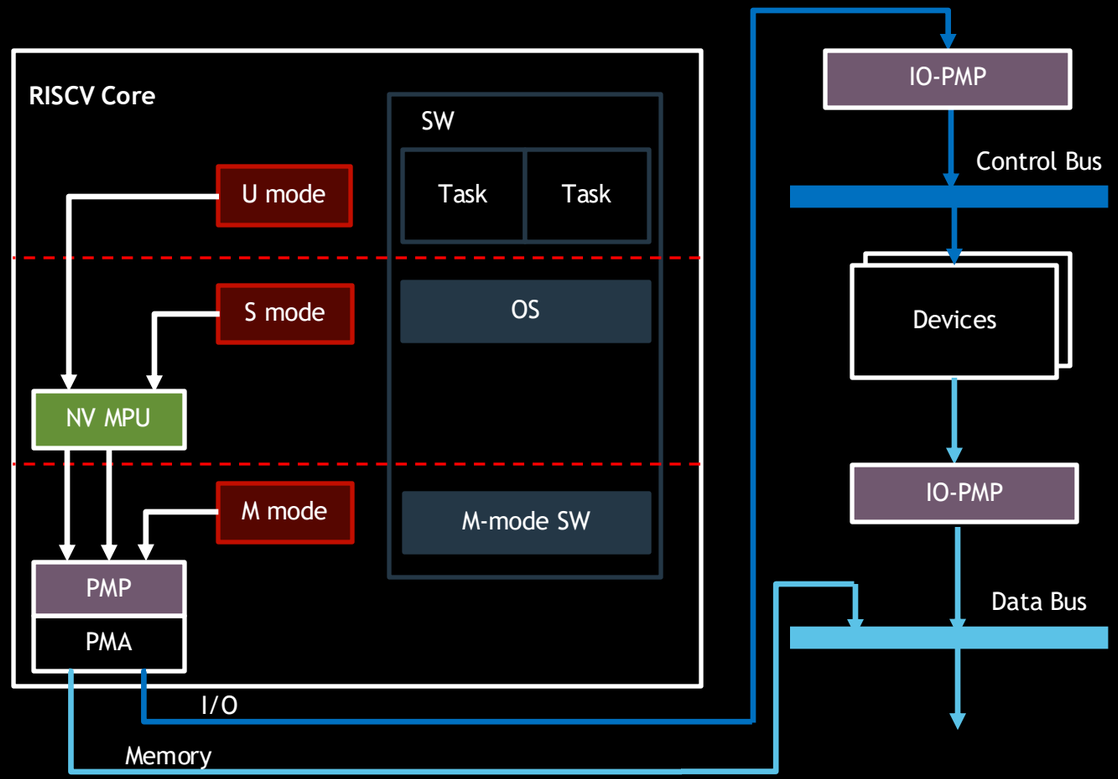


RISC-V Modes		
Level	Name	Abbr.
0	User/Application	U
1	Supervisor	S
2	Hypervisor	HS
3	Machine	M

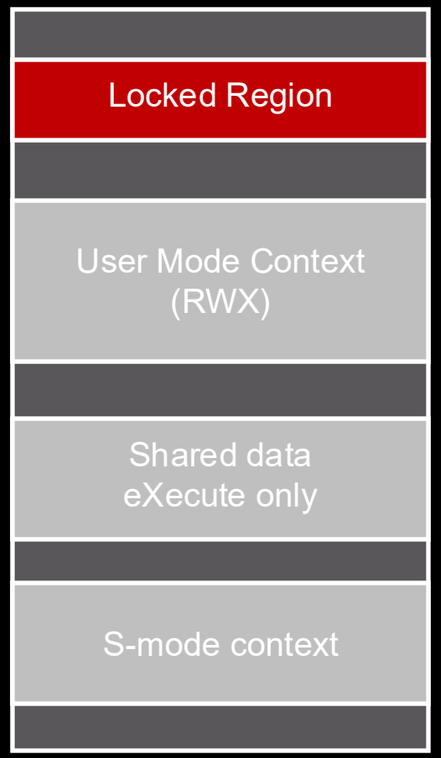
Supported Combinations of Modes	
1	M
2	M. U
3	M, S, U
4	M, HS, (V)S, (V)U

- **Each mode has Control and Status Registers (CSRs)**

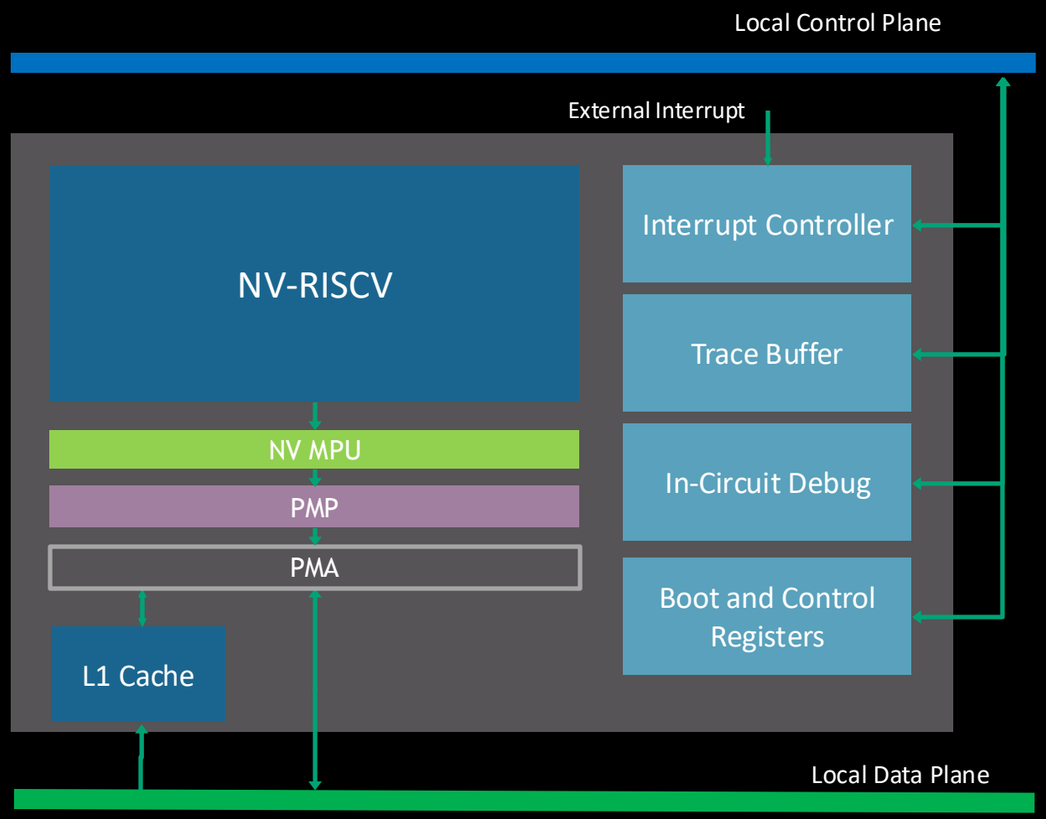
RISC-V Intro – Memory Protection



- PMP – Physical Memory Protection
- IO-PMP – PMP for I/O and devices
- Ability to lock a region until reset



NVRISC-V



NV-RISCV32	NV-RISCV64	NV-RVV
RV32I-MU Multiplication Compression Float	RV64I-MSU Multiplication Compression Float Bit manipulation Atomics	RV32I-MU Multiplication Compression Float Vector
In Order Single Issue 1.8 CM/MHz 1.8 GHz	Out of Order Dual Issue 5 CM/MHz 2 GHz SMP	NV-RISCV32 + vector extension (1024-bit)

Examples of NVIDIA custom security extensions
Secure Debug with ICD
ROM memory protection extension
DCLS
ePMP (draft 0.7)
TBI/PM (Draft 0.7)
Secure I/0 (Exception on bus error)
Halt extension (via CSR)
NV Priv. level extension (via CSR)

Peregrine

NVRISCV + Peripheral devices

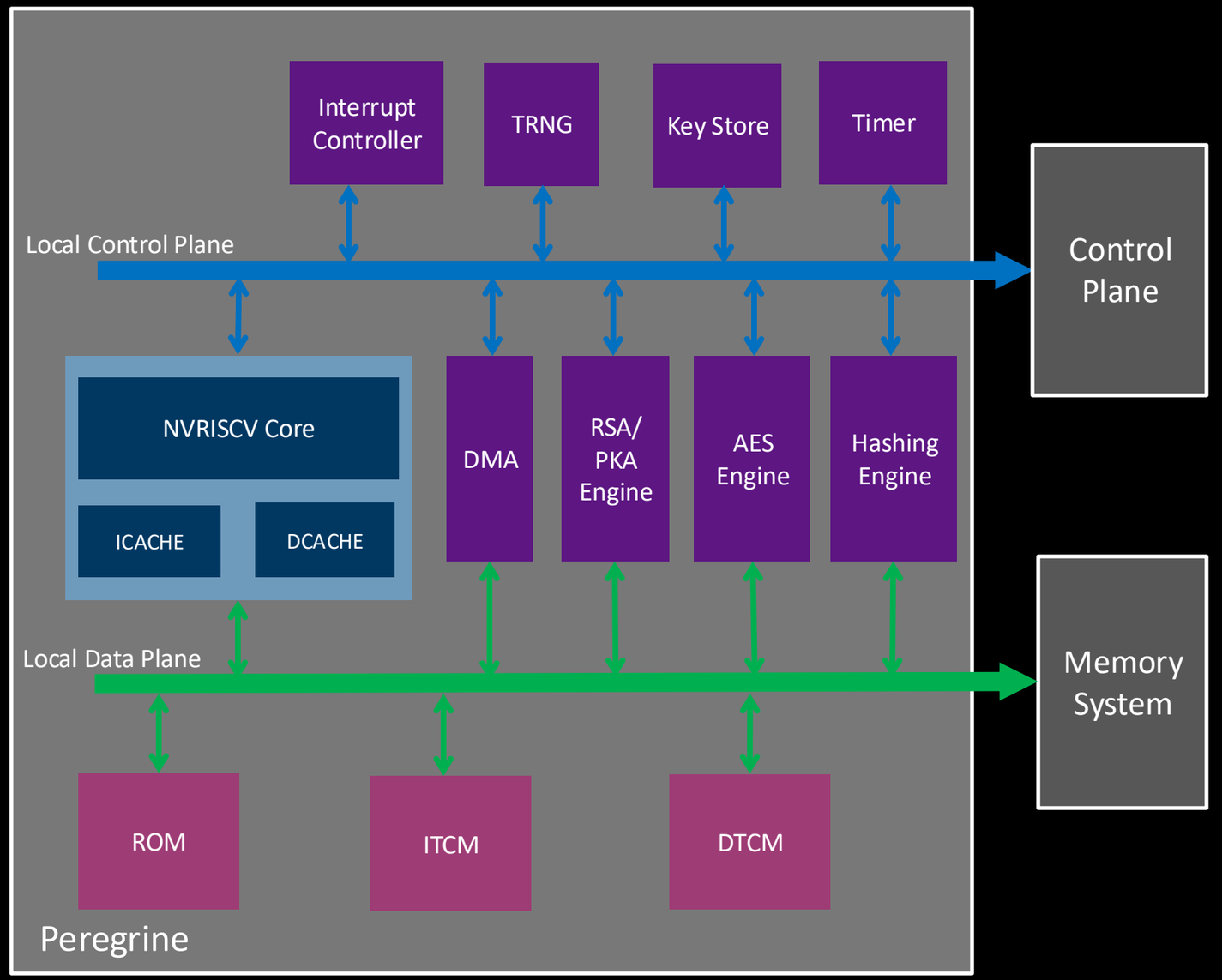
Single and multi-core MCUs

RISC-V extensions may be present or not

Configurable peripherals (crypto, channels)

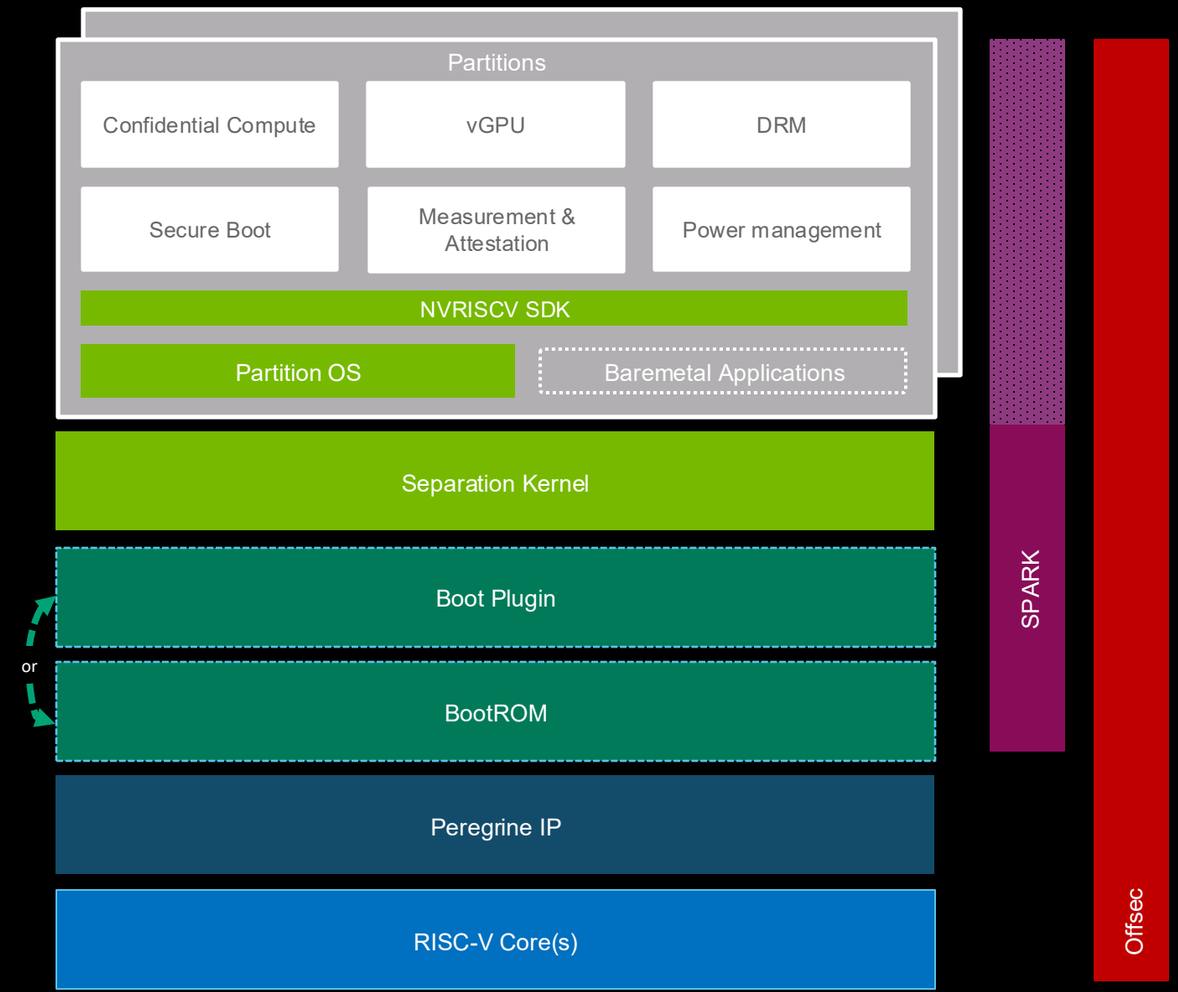
Cache and TCM sizes parameterized

Optional DCLS (Dual Core Lock Step)



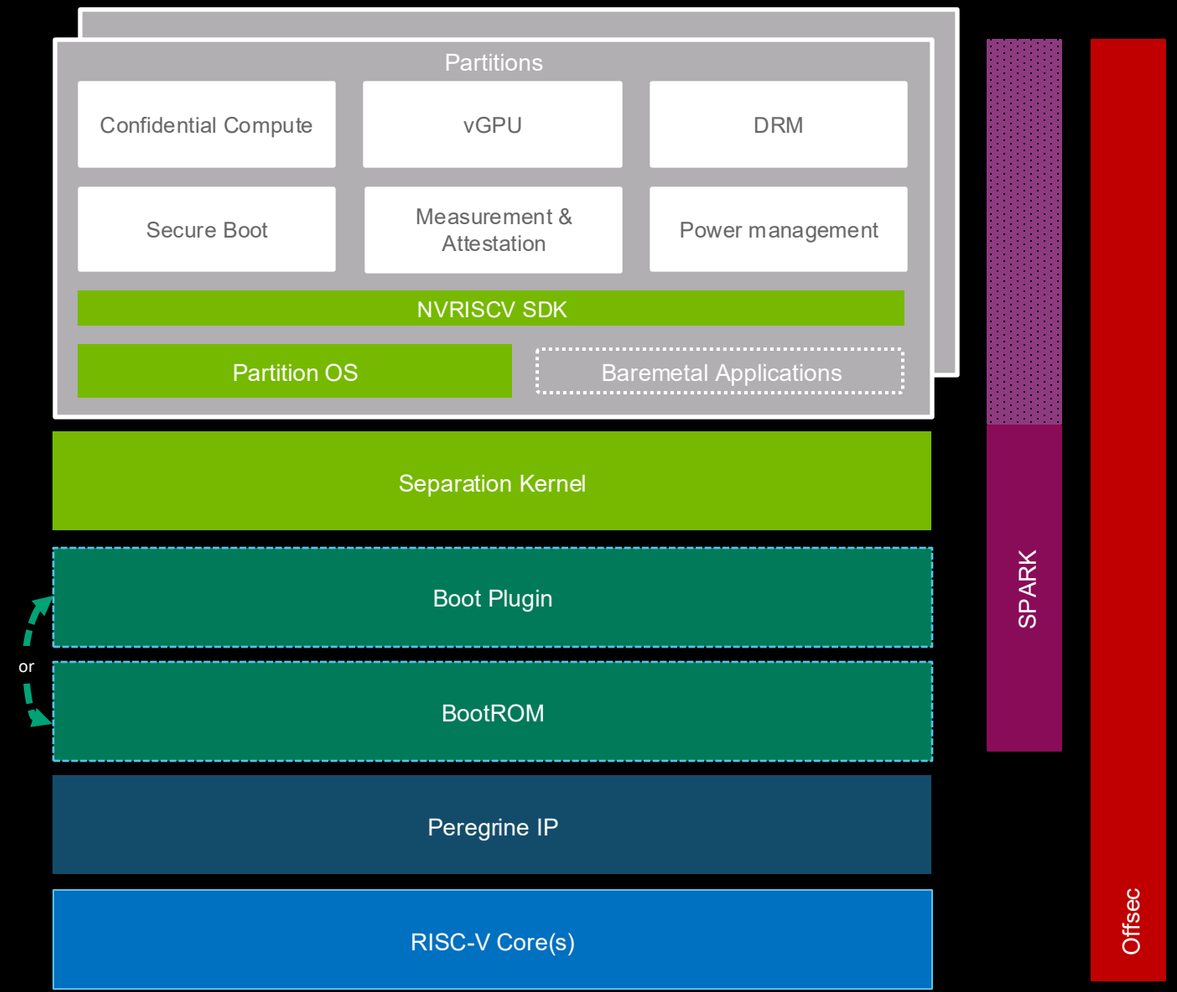
One Core Strategy – Peregrine Ecosystem

- Unified embedded HW and SW across all NVIDIA products



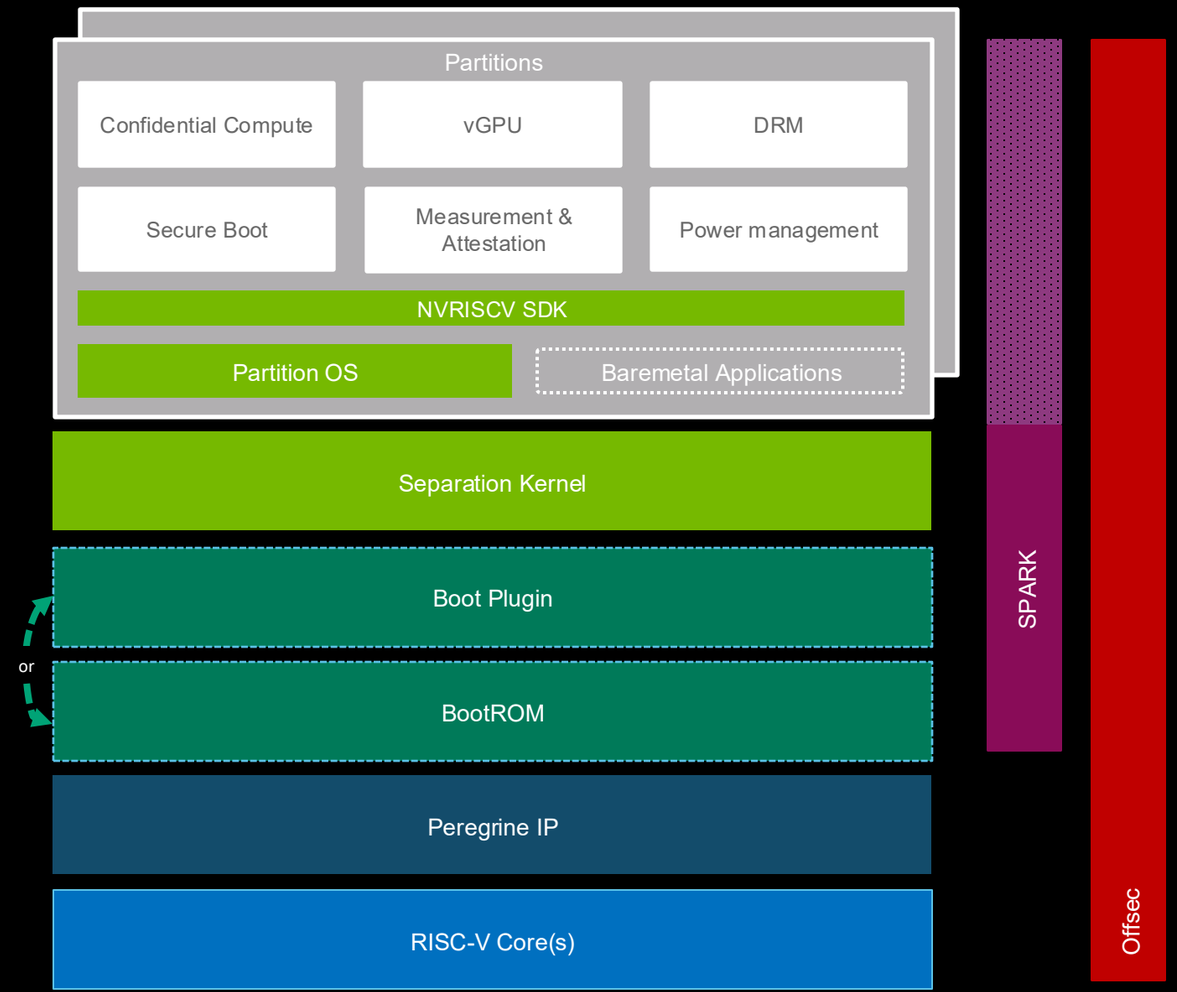
One Core Strategy – Peregrine Ecosystem

- Unified embedded HW and SW across all NVIDIA products
- Configurable architecture, easily adapted to different products, features and deployments
- Uniform attack mitigations; In-depth offensive security efforts investments



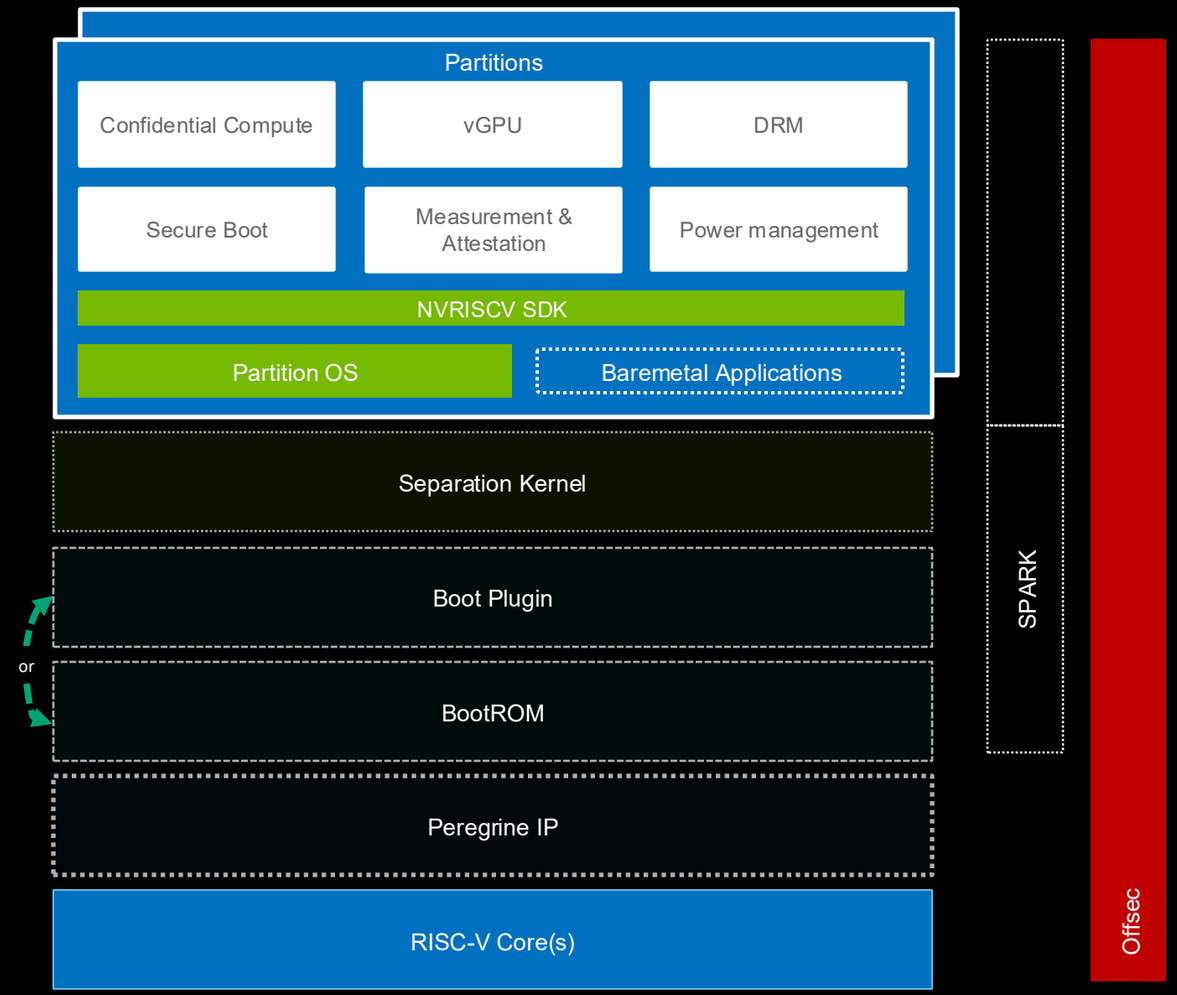
One Core Strategy – Peregrine Ecosystem

- Unified embedded HW and SW across all NVIDIA products
- Configurable architecture, easily adapted to different products, features and deployments
- Uniform attack mitigations; In-depth offensive security efforts investments
- Partition architecture is the foundation for running mixed-criticality applications on NVRISCV
- Peregrine/NVIRSCV architecture foundation for GPU SW Security



One Core Strategy – Peregrine Ecosystem

- Architectural flexibility: Great for innovation, but there are still challenges



RISC-V challenges

- Open-source and flexibility
  - Despite undeniable advantages, there are drawbacks:
    - Fragmentation
      - Remediations: Profiles, RISC-V Foundation
    - Not as mature SW ecosystem
      - Remediations: RISC-V Dev Partners, Extension TG/SIG
- Profiles
  - Addresses fragmentation but they may NOT be mutually compatible
    - E.g., RVB23 != RVA23
    - RVA profiles are trying to be backward compatible but there are caveats
- “Custom” extensions might be costly (contribute to RISC-V!)
  - Your custom (private) extension may become incompatible with the officially ratified one
    - New extension may solve your problem in a better (or not) way
  - Custom HW means custom SW support

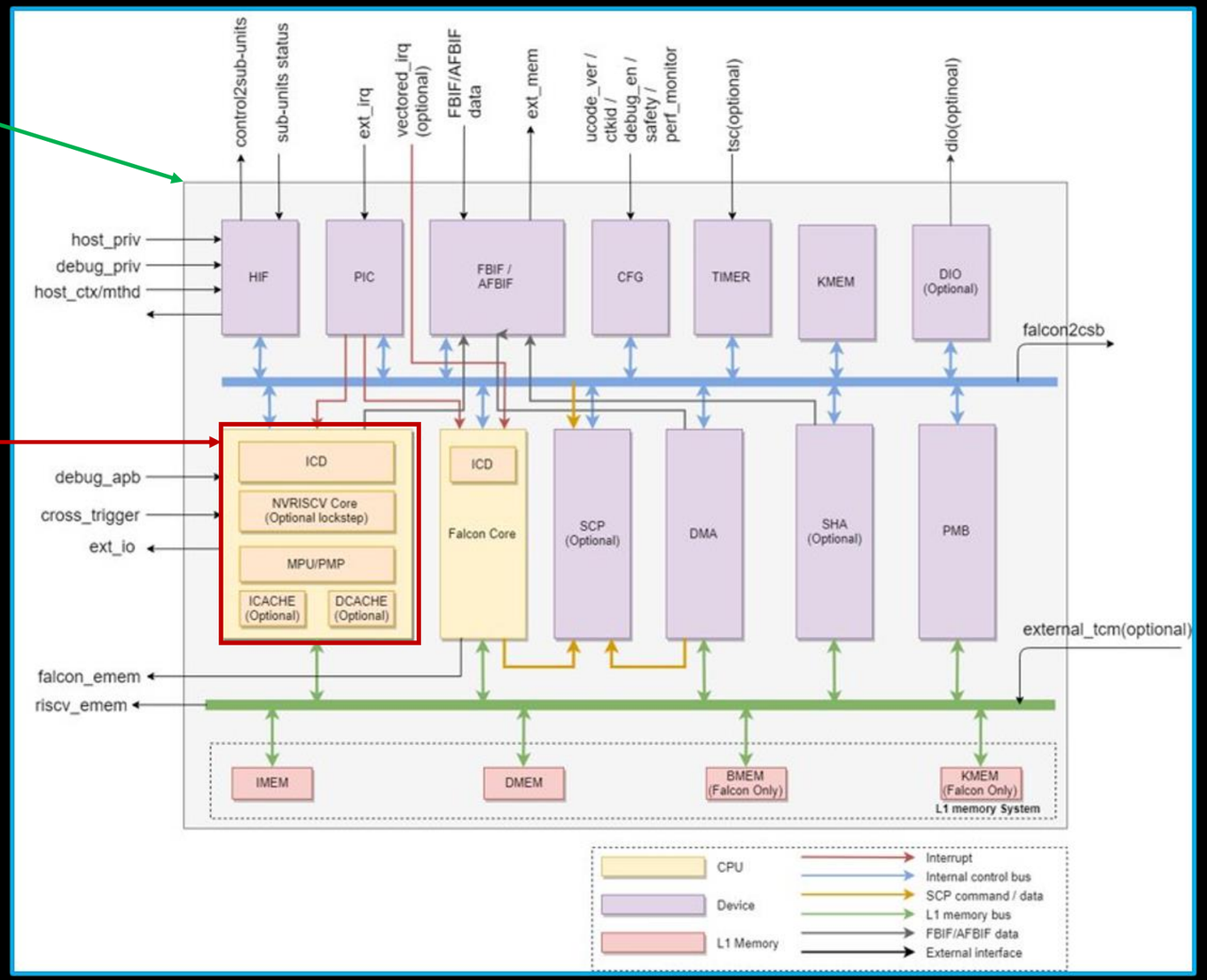
NVRISC-V

Peregrine chiplet (packet)

NVRISC-V

How to secure this new execution environment?

Learn from the past (e.g., No ASLR).
Peregrine must consider inner-”outside” peregrines



NVRISC-V

- How to effectively find software vulnerabilities (the BIGGEST attack surface)?
  - NVIDIA Offensive Security Research (OSR)
    - Manual Vulnerability Research is a *must have* but not a sufficient neither a scalable solution
    - Automatic vulnerability detection (fuzzing) is a crucial piece – how to increase the effectiveness?
      - Address Sanitizers and instrumentation (code-coverage) can help but…

NVRISC-V

- How to effectively find software vulnerabilities (the BIGGEST attack surface)?
  - NVIDIA Offensive Security Research (OSR)
    - Manual Vulnerability Research is a *must have* but not a sufficient neither a scalable solution
    - Automatic vulnerability detection (fuzzing) is a crucial piece – how to increase the effectiveness?
      - Address Sanitizers and instrumentation (code-coverage) can help but… RISC-V did not support that (not at that time) :(

NVRISC-V

- How to effectively find software vulnerabilities (the BIGGEST attack surface)?
  - NVIDIA Offensive Security Research (OSR)
    - Manual Vulnerability Research is a *must have* but not a sufficient neither a scalable solution
    - Automatic vulnerability detection (fuzzing) is a crucial piece – how to increase the effectiveness?
      - Address Sanitizers and instrumentation (code-coverage) can help but… RISC-V did not support that (not at that time) :(

- RISC-V Pointer Masking (PM) extension
  - NVIDIA aimed to add HWASAN (and later MTE) to its RISC-V ecosystem
    - Including M-mode (unusual), S-mode and U-mode support
    - Bare mode support (unusual)
    - With and without OS layer support

NVRISC-V

- How to effectively find software vulnerabilities (the BIGGEST attack surface)?
  - NVIDIA Offensive Security Research (OSR)
    - Manual Vulnerability Research is a *must have* but not a sufficient neither a scalable solution
    - Automatic vulnerability detection (fuzzing) is a crucial piece – how to increase the effectiveness?
      - Address Sanitizers and instrumentation (code-coverage) can help but… RISC-V did not support that (not at that time) :(

- RISC-V Pointer Masking (PM) extension
  - NVIDIA aimed to add HWASAN (and later MTE) to its RISC-V ecosystem
    - Including M-mode (unusual), S-mode and U-mode support
    - Bare mode support (unusual)
    - With and without OS layer support
  - We developed a custom extension and brought it to the RISC-V International (TEE group)

NVRISC-V

- How to effectively find software vulnerabilities (the BIGGEST attack surface)?
  - NVIDIA Offensive Security Research (OSR)
    - Manual Vulnerability Research is a *must have* but not a sufficient neither a scalable solution
    - Automatic vulnerability detection (fuzzing) is a crucial piece – how to increase the effectiveness?
      - Address Sanitizers and instrumentation (code-coverage) can help but… RISC-V did not support that (not at that time) :(

- RISC-V Pointer Masking (PM) extension
  - NVIDIA aimed to add HWASAN (and later MTE) to its RISC-V ecosystem
    - Including M-mode (unusual), S-mode and U-mode support
    - Bare mode support (unusual)
    - With and without OS layer support
  - We developed a custom extension and brought it to the RISC-V International (TEE group)
  - Independently, Google was working on own “Pointer Masking”

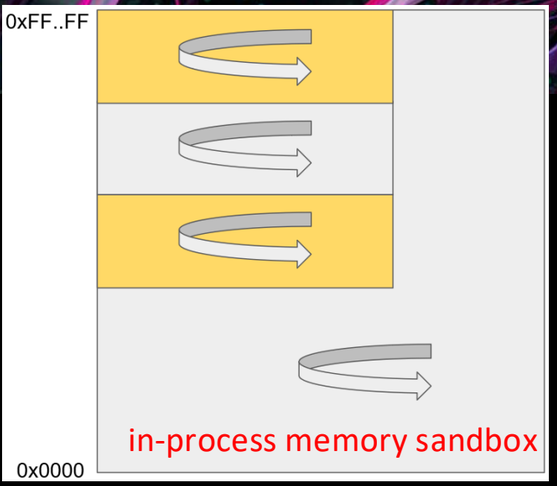
NVRISC-V

- How to effectively find software vulnerabilities (the BIGGEST attack surface)?
  - NVIDIA Offensive Security Research (OSR)
    - Manual Vulnerability Research is a *must have* but not a sufficient neither a scalable solution
    - Automatic vulnerability detection (fuzzing) is a crucial piece – how to increase the effectiveness?
      - Address Sanitizers and instrumentation (code-coverage) can help but… RISC-V did not support that (not at that time) :(

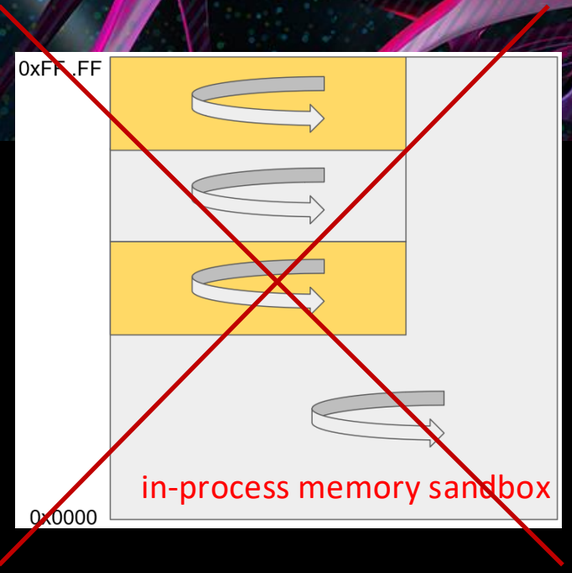
- RISC-V Pointer Masking (PM) extension
  - NVIDIA aimed to add HWASAN (and later MTE) to its RISC-V ecosystem
    - Including M-mode (unusual), S-mode and U-mode support
    - Bare mode support (unusual)
    - With and without OS layer support
  - We developed a custom extension and brought it to the RISC-V International (TEE group)
  - Independently, Google was working on own “Pointer Masking”
  - We decided to unite our use-cases and promote a single standard for all.

NVRISC-V

- RISC-V Pointer Masking extension
  - Serves as a framework
  - PM supported multiple use-cases:
    - HWASAN (later a base for HW MTE)
    - Pointer Authentication (PAC)
    - HW Memory Sandboxing (PM introduced 2 CSRs: actual_address = (requested_address & ~mpmmask) | mpmbase

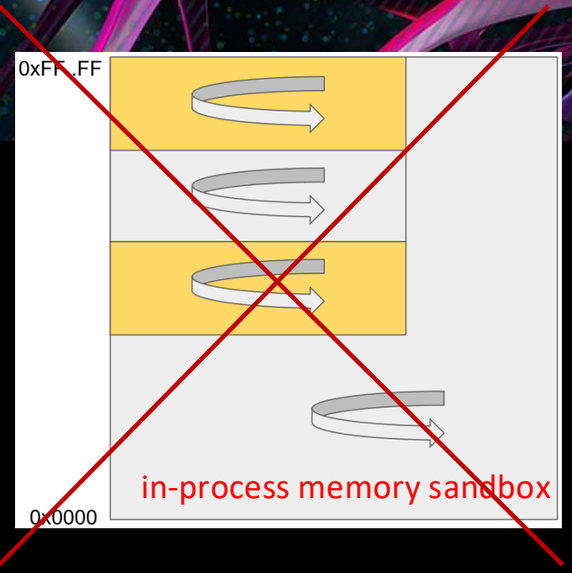


NVRISC-V



- RISC-V Pointer Masking extension
  - Serves as a framework
  - PM supported multiple use-cases:
    - HWASAN (later a base for HW MTE)
    - ~~Pointer Authentication (PAC)~~
    - ~~HW Memory Sandboxing (PM introduced 2 CSRs: actual_address = (requested_address & ~mpmmask) | mpmbase~~
  - Ratified version – HWASAN only + ISA integration
    - Current equation for VA:
      transformed_effective_address = {{PMLEN{effective_address[XLEN-PMLEN-1]}}, effective_address[XLEN-PMLEN-1:0]}
    - Current equation for PA:
      transformed_effective_address = {{PMLEN{0}}, effective_address[XLEN-PMLEN-1:0]}
    - No new CSRs, PMLEN in *envcfg (2 bits, **supports top 7 or 16 bits of masking**)

NVRISC-V



- RISC-V Pointer Masking extension
  - Serves as a framework
  - PM supported multiple use-cases:
    - HWASAN (later a base for HW MTE)
    - ~~Pointer Authentication (PAC)~~
    - ~~HW Memory Sandboxing (PM introduced 2 CSRs: actual_address = (requested_address & ~mpmmask) | mpmbase~~
  - Ratified version – HWASAN only + ISA integration
    - Current equation for VA:
      transformed_effective_address = {{PMLEN{effective_address[XLEN-PMLEN-1]}}, effective_address[XLEN-PMLEN-1:0]}
    - Current equation for PA:
      transformed_effective_address = {{PMLEN{0}}, effective_address[XLEN-PMLEN-1:0]}
    - No new CSRs, PMLEN in *envcfg (2 bits, **supports top 7 or 16 bits of masking**)
  - RISC-V included PM as part of the profiles!
    - RVA23:
      - Supm, Ssnpm – mandatory for RVA23S64
      - Sspm – optional for RVA23S64
    - RVB23:
      - Supm – optional for RVB23U64
      - Ssnpm and Sspm – optional for RVB23S64

NVRISC-V

- RISC-V Pointer Masking extension
  - 4+ years of work
  - Pointer Masking – umbrella for 5 extensions
    - Split per priv-level:
      - Ssnpm – A supervisor-level extension for the next lower privilege
      - Smnpm – A machine-level extension for the next lower privilege
      - Smmpm – A machine-level extension for M-mode
    - Additionally, 2 extensions describing an execution environment – no bearing on HW implementations.
      - Sspm – PM support available in supervisor mode
      - Supm – PM support available in user mode
  - SW ecosystem got support for it
    - LLVM/GCC compilers, binutils, Linux kernel, Qemu, SPIKE, SAIL and more
  - We added HWASAN support for NVIDIA SW ecosystems
    - Fuzzing GSP firmware (under Partition OS)
    - Preparing to fuzz bare-metal microcode
    - More in progress
    - Fuzzing RM
    - Preparing to fuzz firmware under RTOS

NVRISC-V

- RISC-V Pointer Masking extension
  - 4+ years of work
  - Pointer Masking – umbrella for 5 extensions
    - Split per
      - Ssn
      - Smr
      - Smr
    - Addition
      - Sspr
      - Sup
  - SW ecosy
    - LLVM/GCC compilers, binutils, Linux kernel, Qemu, SPIKE, SAIL and more
  - We added HWASAN support for NVIDIA SW ecosystems
    - Fuzzing GSP firmware (under Partition OS)
    - Preparing to fuzz bare-metal microcode
    - More in progress
    - Fuzzing RM
    - Preparing to fuzz firmware under RTOS

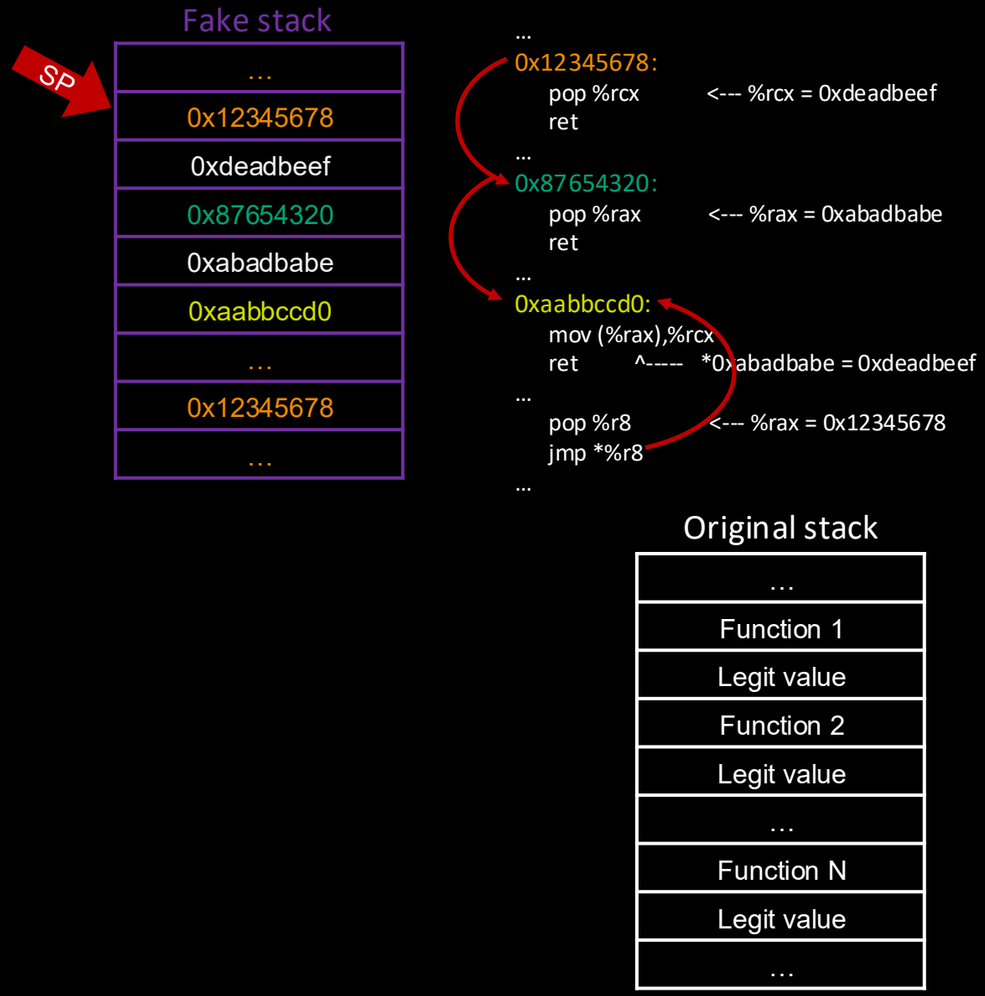
Contributors

Authors: Adam Zabrocki, Martin Maas, Lee Campbell, RISC-V Runtime Integrity and J Extension Task Groups

Contributors: Krste Asanovic, Jecel Assumpcao, James Ball, Alexey Baturo, Allen Baum, Andrew Bresticker, Paul Donahue, Greg Favor, Andy Glew, Deepak Gupta, John Hauser, Samuel Holland, John Ingalls, James Kenney, Earl Killian, Christos Kotselidis, Dean Liberty, Philip Reames, Ian Rogers, Josh Scheid, Kostya Serebryany, Ved Shanbhogue, Boris Shingarov, Andrew Waterman, David Weaver, Foivos Zakkak, Members of the Runtime Integrity and J Extension Task Groups

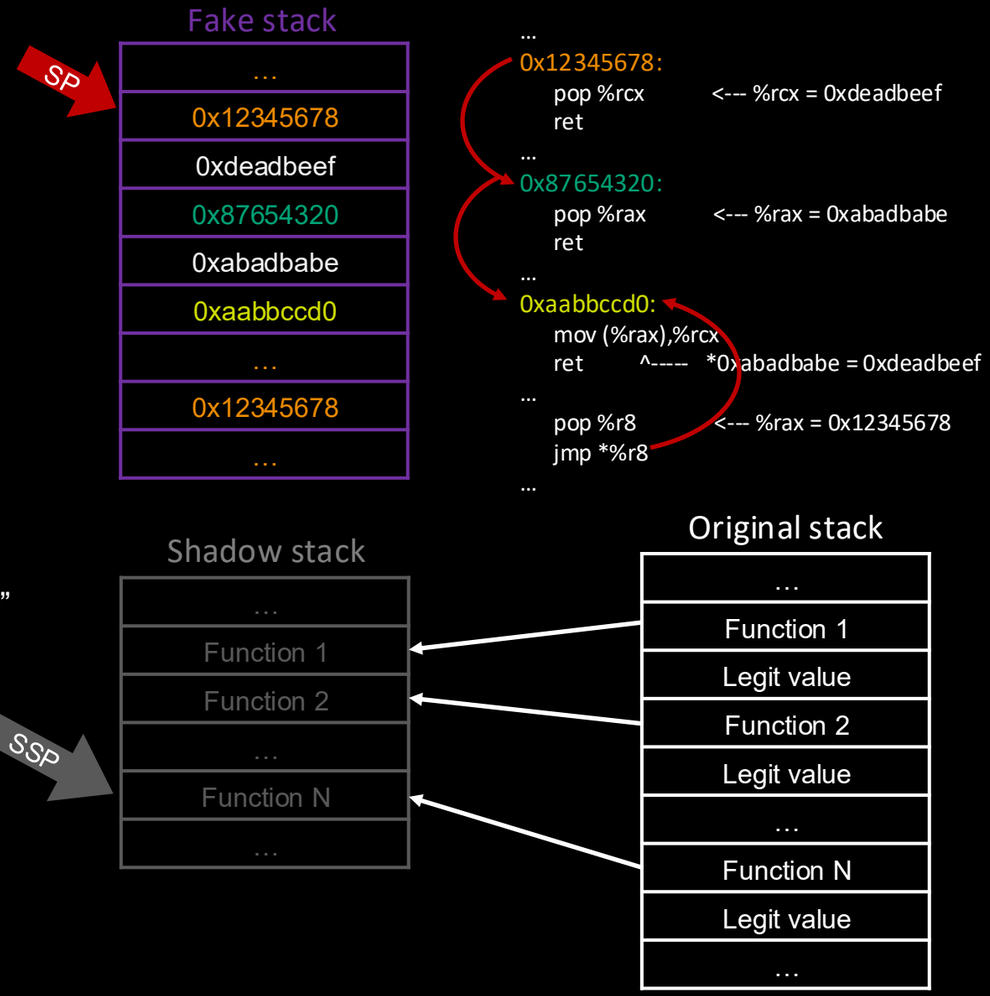
NVRISC-V

- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)



NVRISC-V

- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)
  - CFI is actually 2 sub-extensions
    - Zicfiss – Control Flow Integrity Shadow Stack
      - Enforces backward-edge control flow integrity
      - Creates a new region “shadow stack” which keeps a copy of RA only
      - New reg (SSP) and instructions for “shadow stack” management
      - Preserves the original stack ABI
      - Before return, the RA is verified against the “shadow stack” copy
        - Function can only return to its original caller
        - If verification failed, SW-check exception is raised – “Shadow Stack Fault (code=3)”

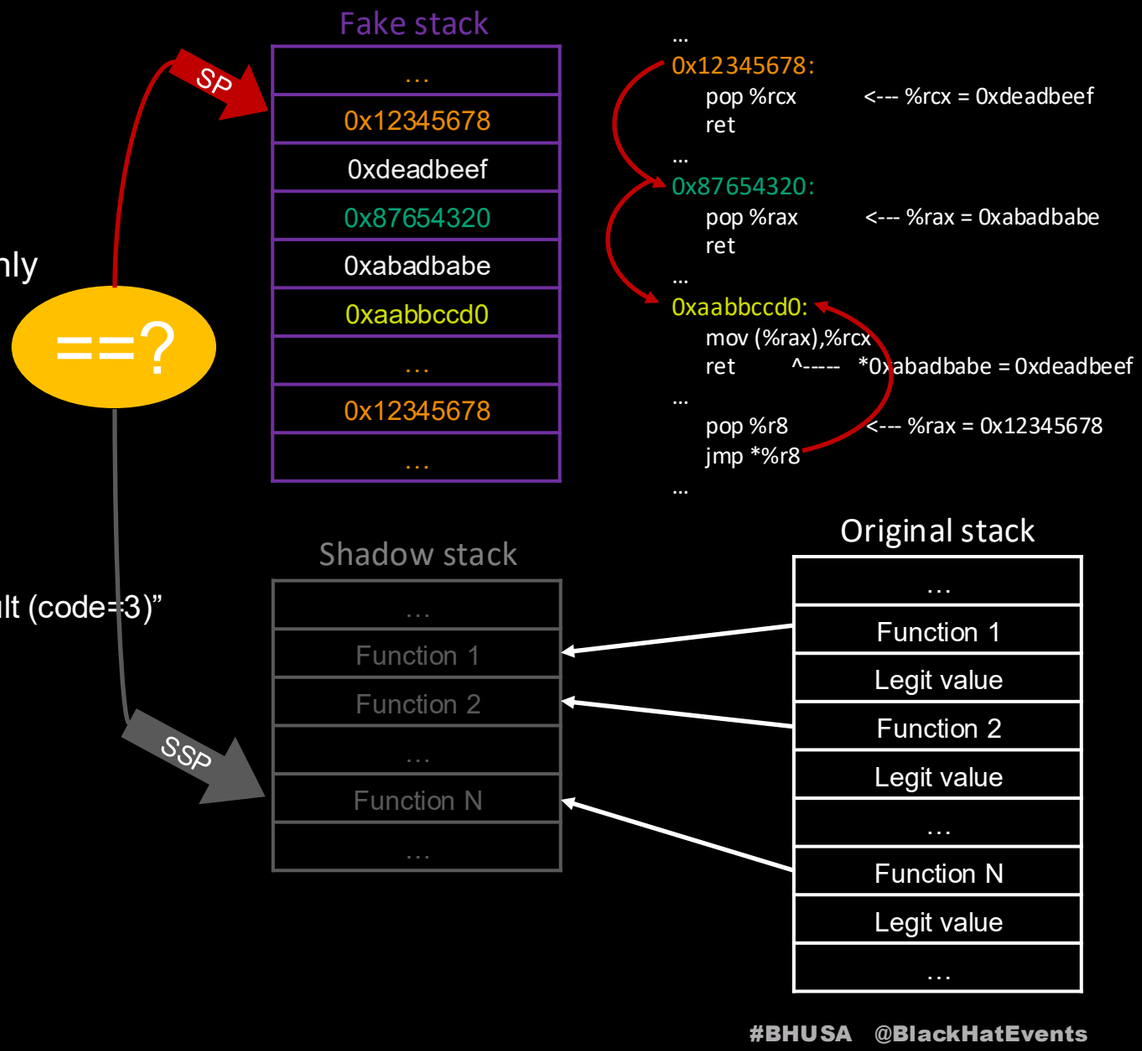



```
function_entry:
    addi sp,sp,-8  # push link register x1
    sd x1,(sp)     # on regular stack
    sspush x1      # push link register x1 on shadow stack
     :
    ld x1,(sp)     # pop link register x1 from regular stack
    addi sp,sp,8
    sspopchk x1    # fault if x1 not equal to shadow return address
    ret
```

NVRISC-V

- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)
  - CFI is actually 2 sub-extensions
    - Zicfiss – Control Flow Integrity Shadow Stack
      - Enforces backward-edge control flow integrity
      - Creates a new region “shadow stack” which keeps a copy of RA only
      - New reg (SSP) and instructions for “shadow stack” management
      - Preserves the original stack ABI
      - Before return, the RA is verified against the “shadow stack” copy
        - Function can only return to its original caller
        - If verification failed, SW-check exception is raised – “Shadow Stack Fault (code=3)”

```
function_entry:
    addi sp,sp,-8  # push link register x1
    sd x1,(sp)     # on regular stack
    sspush x1      # push link register x1 on shadow stack
     :
    ld x1,(sp)     # pop link register x1 from regular stack
    addi sp,sp,8
    sspopchk x1    # fault if x1 not equal to shadow return address
    ret
```

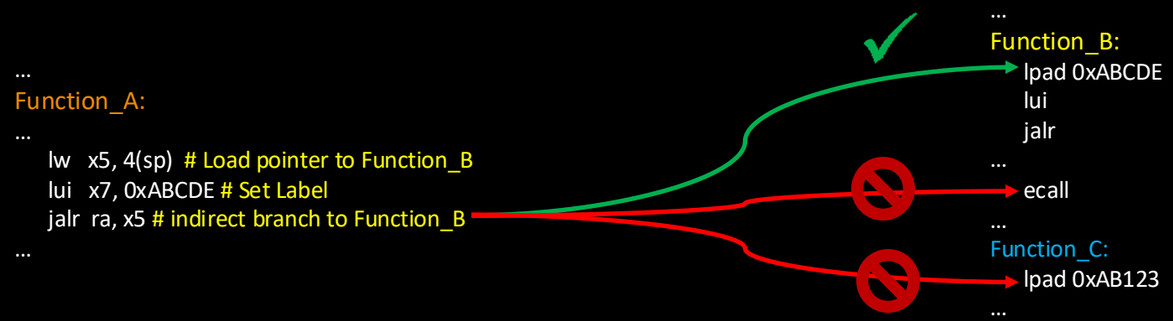


NVRISC-V

- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)
  - CFI is actually 2 sub-extensions
    - Zicfilp – Control Flow Integrity Landing Pads
      - Enforces forward-edge control flow integrity
      - Indirect branch *must* be a landing pad instruction (LPAD)
        - 20-bit encoded label instruction
      - Each hart maintains an expected landing pad (ELP) state
      - If ELP == LP_EXPECTED a SW exception is raised if
        - PC of next instruction is not 4-bytes aligned or is not an LPAD
        - A label does not match the expected landing pad label in bits 31:12 of the x7 register
        - If verification failed, SW-check exception is raised – “Landing Pad Fault (code=2)”

NVRISC-V

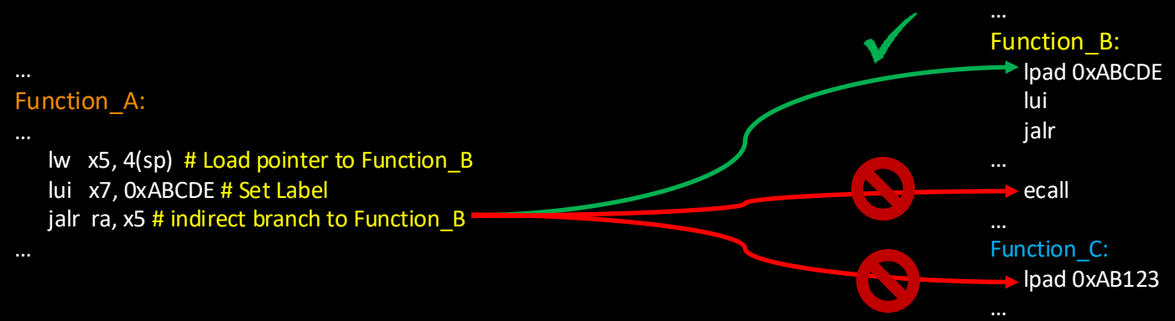
- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)
  - CFI is actually 2 sub-extensions
    - Zicfilp – Control Flow Integrity Landing Pads
      - Enforces forward-edge control flow integrity
      - Indirect branch *must* be a landing pad instruction (LPAD)
        - 20-bit encoded label instruction
      - Each hart maintains an expected landing pad (ELP) state
      - If ELP == LP_EXPECTED a SW exception is raised if
        - PC of next instruction is not 4-bytes aligned or is not an LPAD
        - A label does not match the expected landing pad label in bits 31:12 of the x7 register
        - If verification failed, SW-check exception is raised – “Landing Pad Fault (code=2)”



NVRISC-V

- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)
  - CFI is actually 2 sub-extensions
    - Zicfilp – Control Flow Integrity Landing Pads
      - Enforces forward-edge control flow integrity
      - Indirect branch *must* be a landing pad instruction (LPAD)
        - 20-bit encoded label instruction
      - Each hart maintains an expected landing pad (ELP) state
      - If ELP == LP_EXPECTED a SW exception is raised if
        - PC of next instruction is not 4-bytes aligned or is not an LPAD
        - A label does not match the expected landing pad label in bits 31:12 of the x7 register
        - If verification failed, SW-check exception is raised – “Landing Pad Fault (code=2)”

- CFI as part of the RISC-V profiles
- SW ecosystem got support for it
  - LLVM/GCC, binutils, Linux kernel, Qemu, more
- We are adding SW support for CFI
  - We are committed to bringing CFI support for HW and SW in “Rubin” chips (GR20x)
  - We are considering adding “Zicfiss” to M-mode



NVRISC-V

- RISC-V Control Flow Integrity (CFI) extension
  - CFI tries to protect against code reuse attacks (e.g., ret2libc, ROP, COP/JOP, etc)
  - CFI is actually 2 sub-extensions
    - Zicfilp – Contro
      - Enforces fo
      - Indirect bra
        - 20-bit en
      - Each hart m
      - If ELP == L
        - PC of ne
        - A label does not match the expected landing pad label in bits 31:12 of the x7 register
        - If verification failed, SW-check exception is raised – “Landing Pad Fault (code=2)”

Contributors

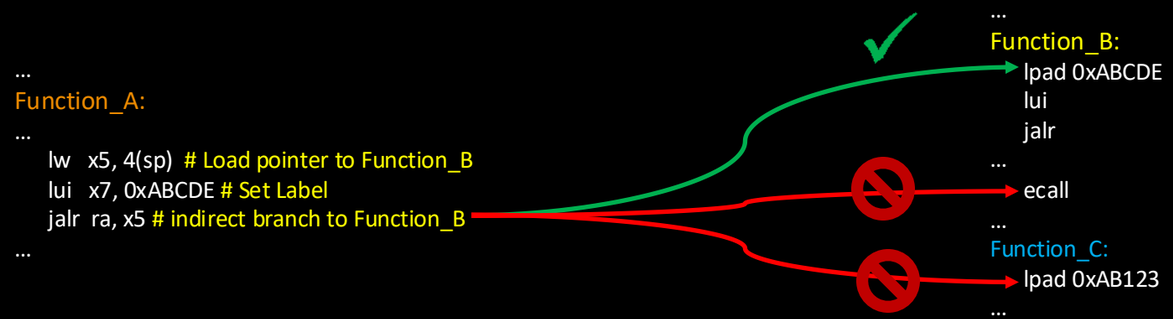
This RISC-V specification has been contributed to directly or indirectly by (in alphabetical order):

Adam Zabrocki, Andrew Waterman, Antoine Linarès, Argyro Palli, Dean Liberty, Deepak Gupta, Eckhard Delfs, George Christou, Greg Favor, Greg McGary, Henry Hsieh, Johan Klockars, John Hauser, John Ingalls, Kip Walker, Kito Cheng, Lasse Collin, Liu Zhiwei, Mark Hill, Nick Kossifidis, Phillip Reames, Rui Ueyama, Sami Tolvanen, Sotiris Ioannidis, Stefan O'Rear, Thurston Dang, Tsukasa OI, Vedvyas Shanbhogue

SC-V profiles

upport for it
Linux kernel, Qemu, more

upport for CFI
o bringing CFI support for
n” chips (GR20x)
dding “Zicfiss” to M-mode

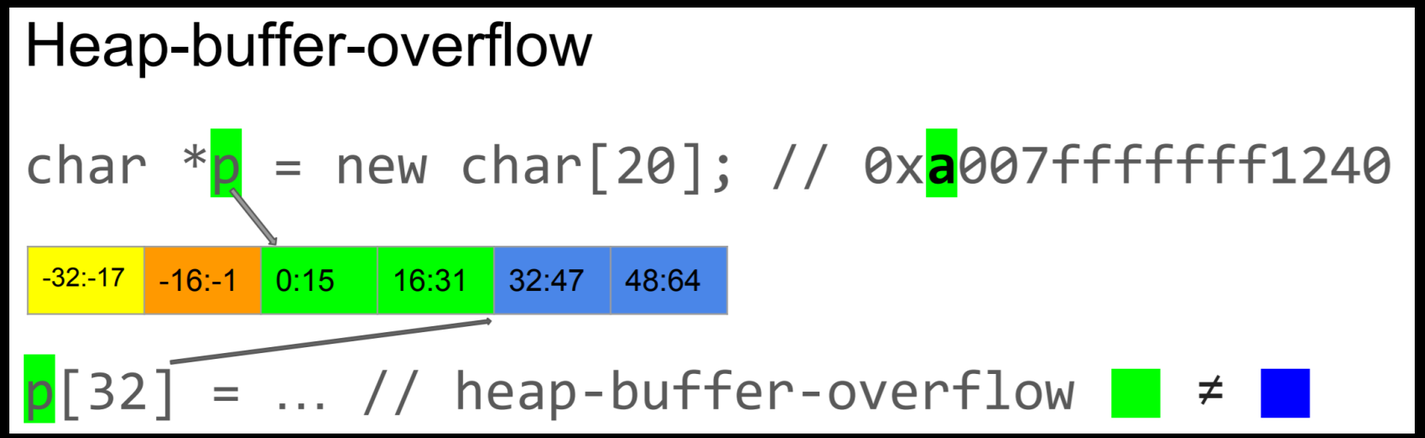


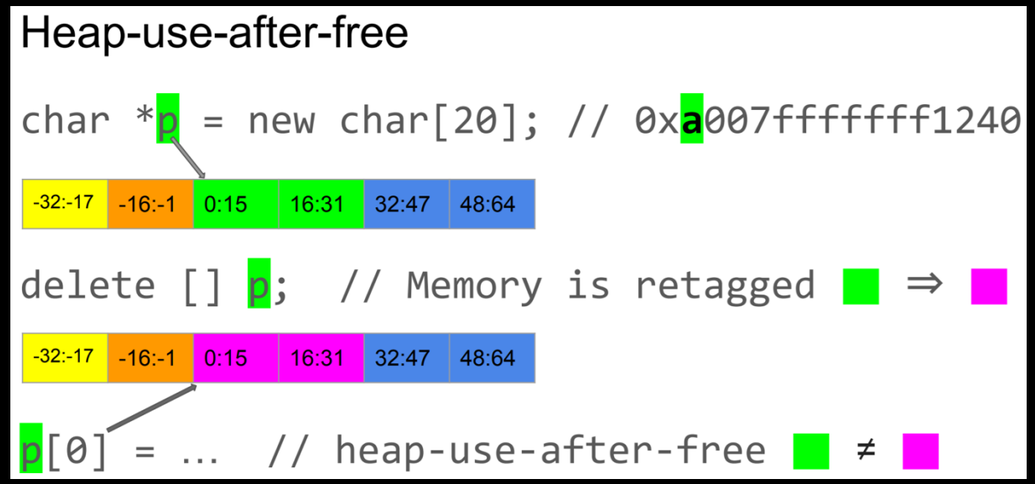
NVRISC-V

- (NV)RISC-V extensions – what's next?
  - RISC-V Memory Tagging extension (MTE)
  - RISC-V CFI M-mode Shadow Stack sub-extension
  - RISC-V Hardware Fault Isolation (HFI)

NVRISC-V

- (NV)RISC-V extensions – what's next?
  - RISC-V Memory Tagging extension (MTE)
    - Hardware-assisted Memory Tagging – addresses performance issues with HWASAN
    - We are actively contributing to RISC-V MTE. Beta spec released in June 2025.
    - NVRISC-V ecosystem (HW and SW) support when ratified





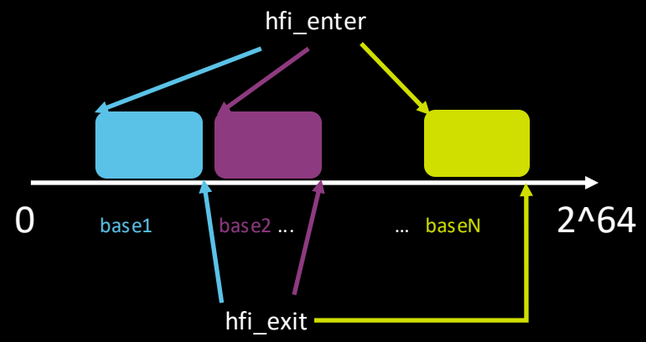
  - RISC-V CFI M-mode Shadow Stack sub-extension
  - RISC-V Hardware Fault Isolation (HFI)

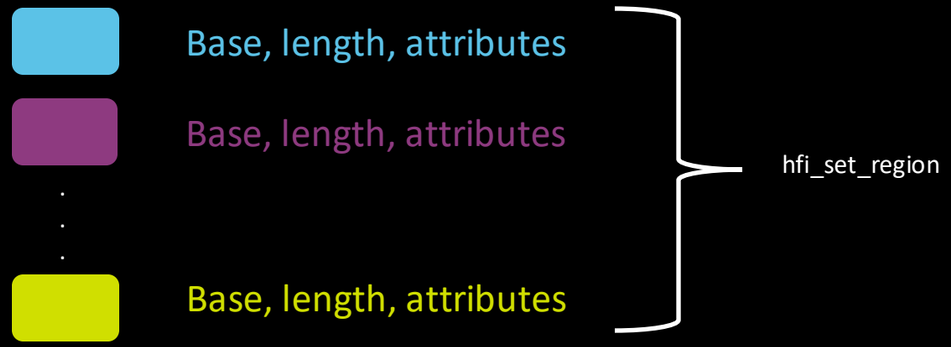
NVRISC-V

- (NV)RISC-V extensions – what's next?
  - RISC-V Memory Tagging extension (MTE)
  - RISC-V CFI M-mode Shadow Stack sub-extension
    - CFI Shadow Stack (Zicfiss) is not defined for M-mode
      - NVIDIA and RISC-V are working on Zicfiss for M-mode to enhance protection of the critical M-mode SW.
    - Landing Pads (Zicfilp) is defined for all modes (include M-mode) already
  - RISC-V Hardware Fault Isolation (HFI)

NVRISC-V

- (NV)RISC-V extensions – what's next?
  - RISC-V Memory Tagging extension (MTE)
  - RISC-V CFI M-mode Shadow Stack sub-extension
  - RISC-V Hardware Fault Isolation (HFI)
    - Addresses in-process Memory Sandbox
      - No TG yet
    - We are evaluating benefits of bringing HFI sandbox to our SW ecosystem (Partition OS, Separation Kernel)
    - HFI introduce a user-mode concept of memory regions. Any access "outside" of the predefined region generates a trap.





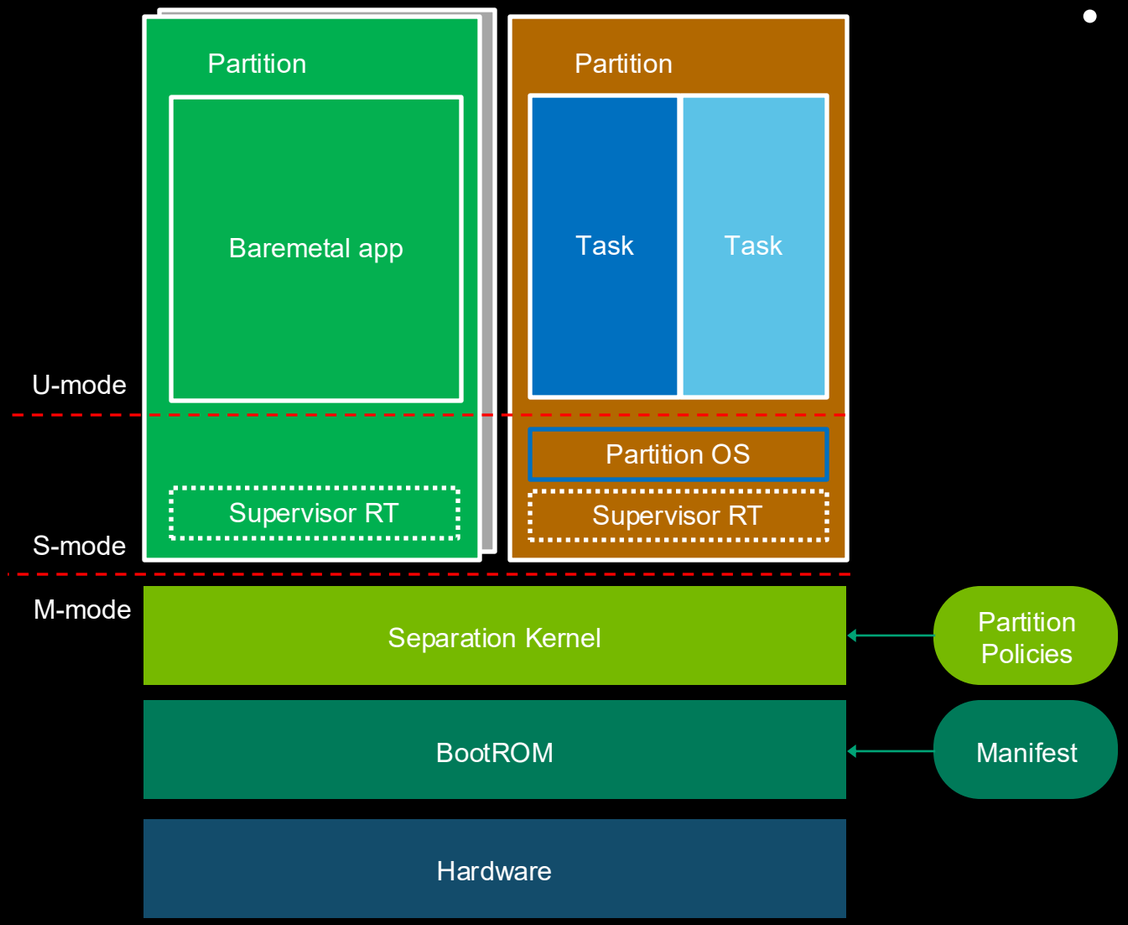
NVRISC-V

- (NV)RISC-V extensions – what's next?
  - RISC-V Memory Tagging extension (MTE)
  - RISC-V CFI M-mode Shadow Stack sub-extension
  - RISC-V Hardware Fault Isolation (HFI)

- Additional areas of interest:
  - Post Quantum Cryptography (PQC)
  - Side-channel protection / hardening
  - CHERI
  - Enhanced Hardware Fault Injection protection

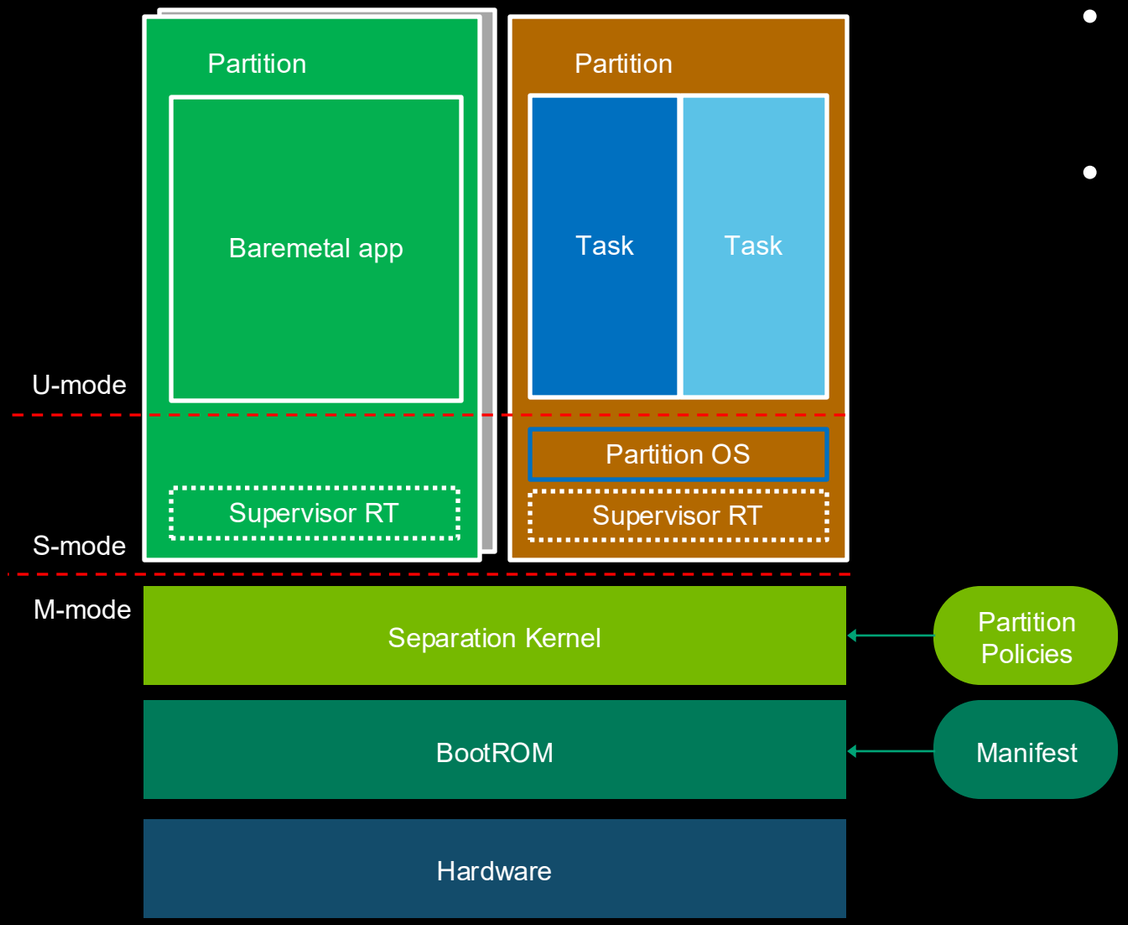
Building the Secure Software Foundation on RISC-V

Peregrine / NVRISCV Multi-Partition Software Architecture

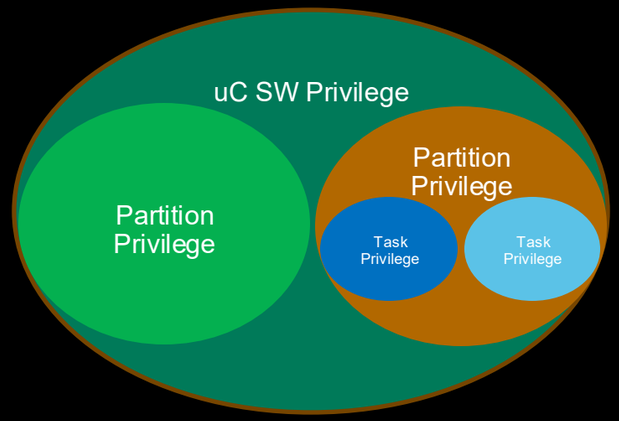


- Multiple Independent Levels of Security/Safety (MILS) architecture

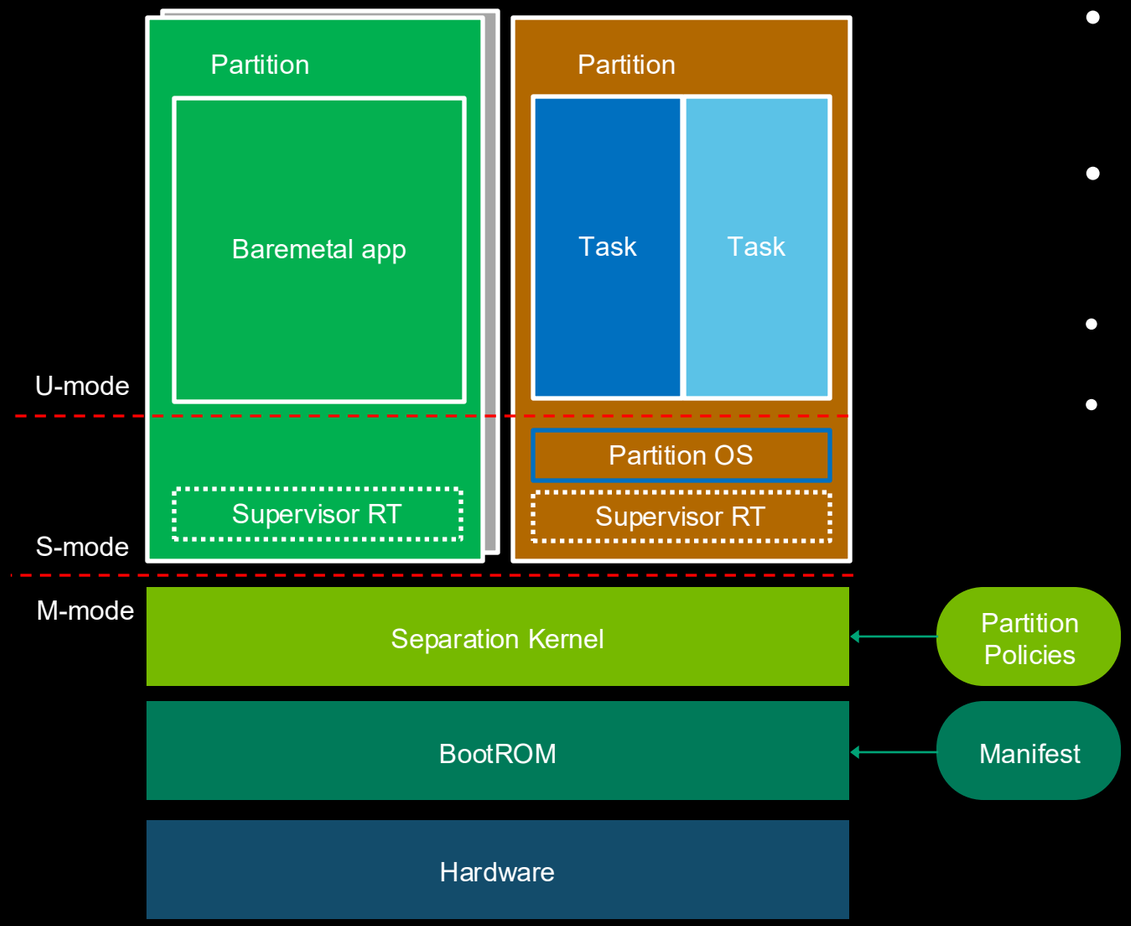
Peregrine / NVRISCV Multi-Partition Software Architecture



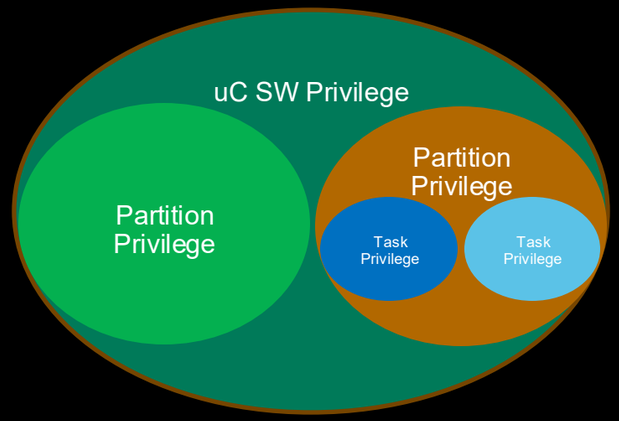
- Multiple Independent Levels of Security/Safety (MILS) architecture
- Fine-grained access control to HW defined by manifest and partition policies



Peregrine / NVRISCV Multi-Partition Software Architecture

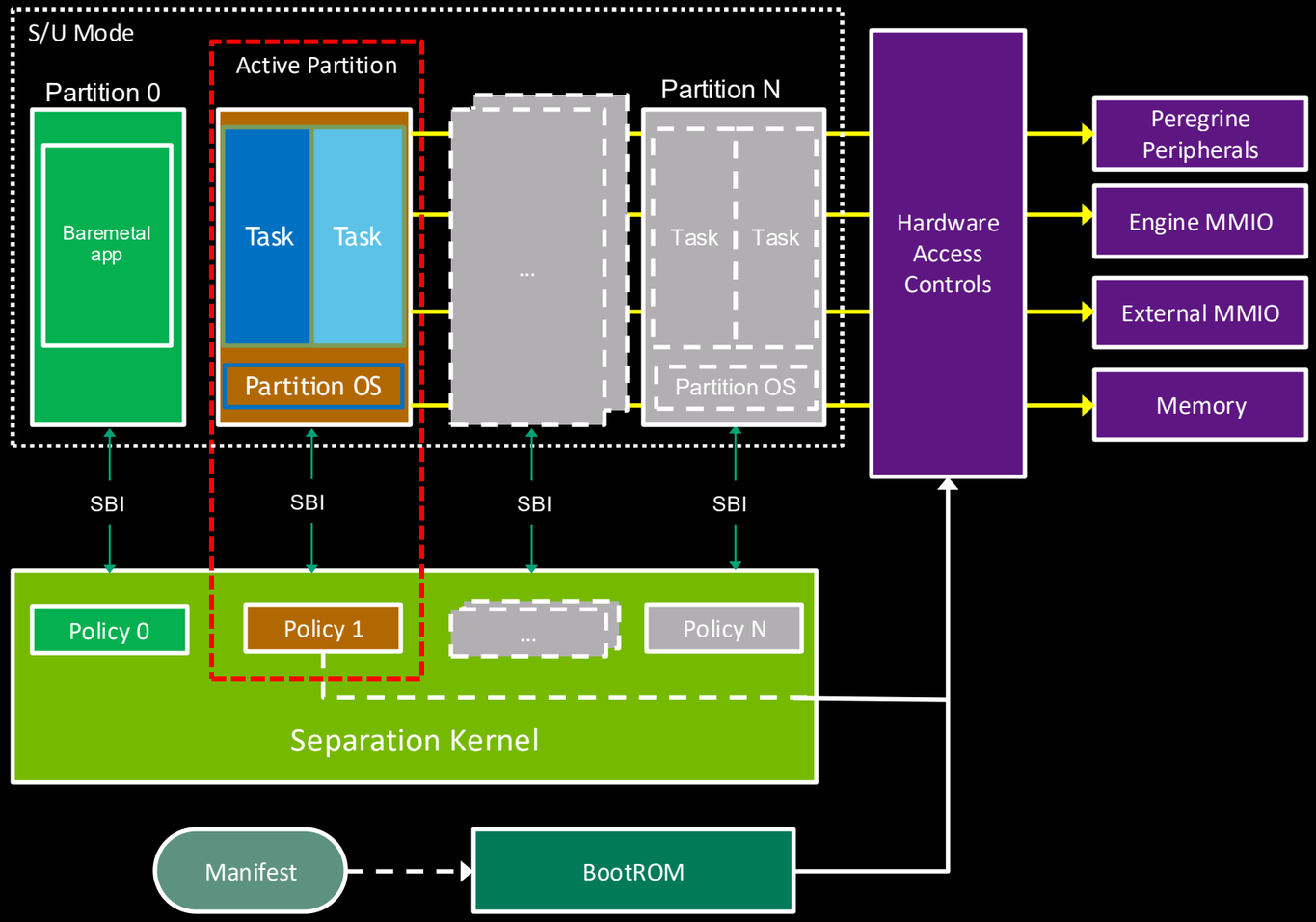


- Multiple Independent Levels of Security/Safety (MILS) architecture
- Fine-grained access control to HW defined by manifest and partition policies
- Partition is defined by partition configurations – partition policies
- Manifest and policies are signed static configuration sets



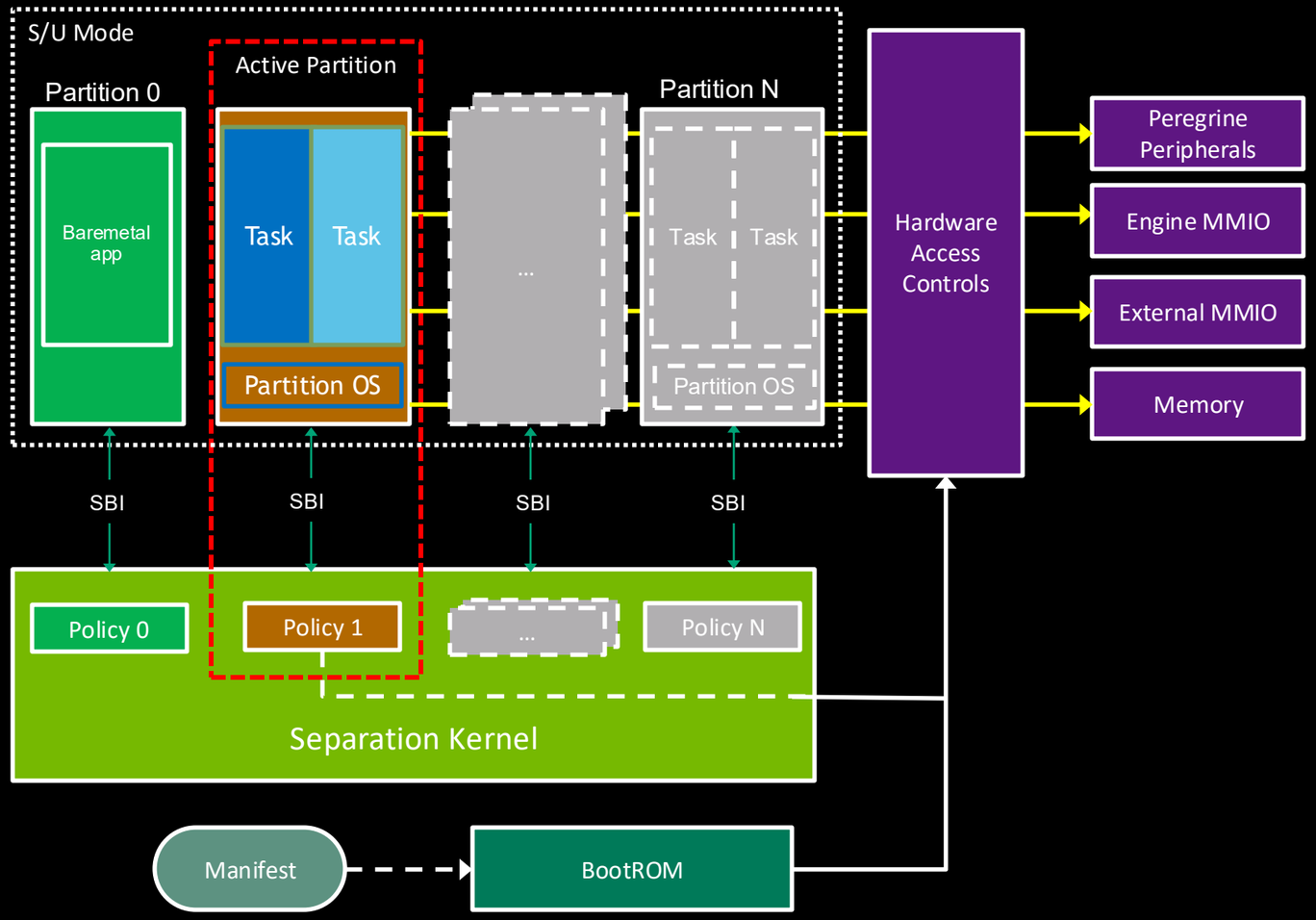
Foundation for running mixed-criticality applications

- All information flow in/out partitions is access controlled



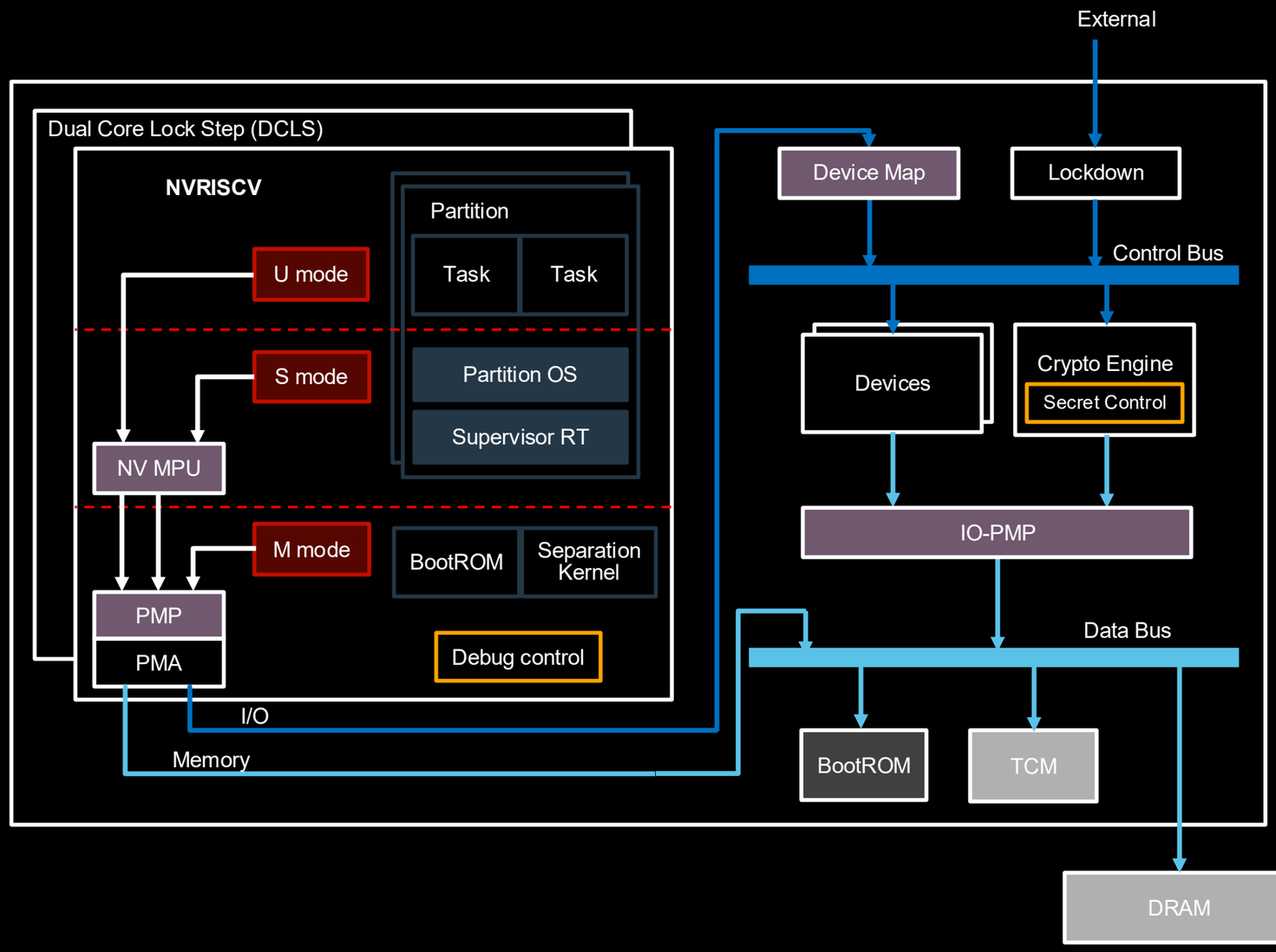
Foundation for running mixed-criticality applications

- All information flow in/out partitions is access controlled
- Separation Kernel (not a Hypervisor):
- Controls what HW is exposed to partition
- Does not abstract HW
- Small and formally verified to be free of runtime errors



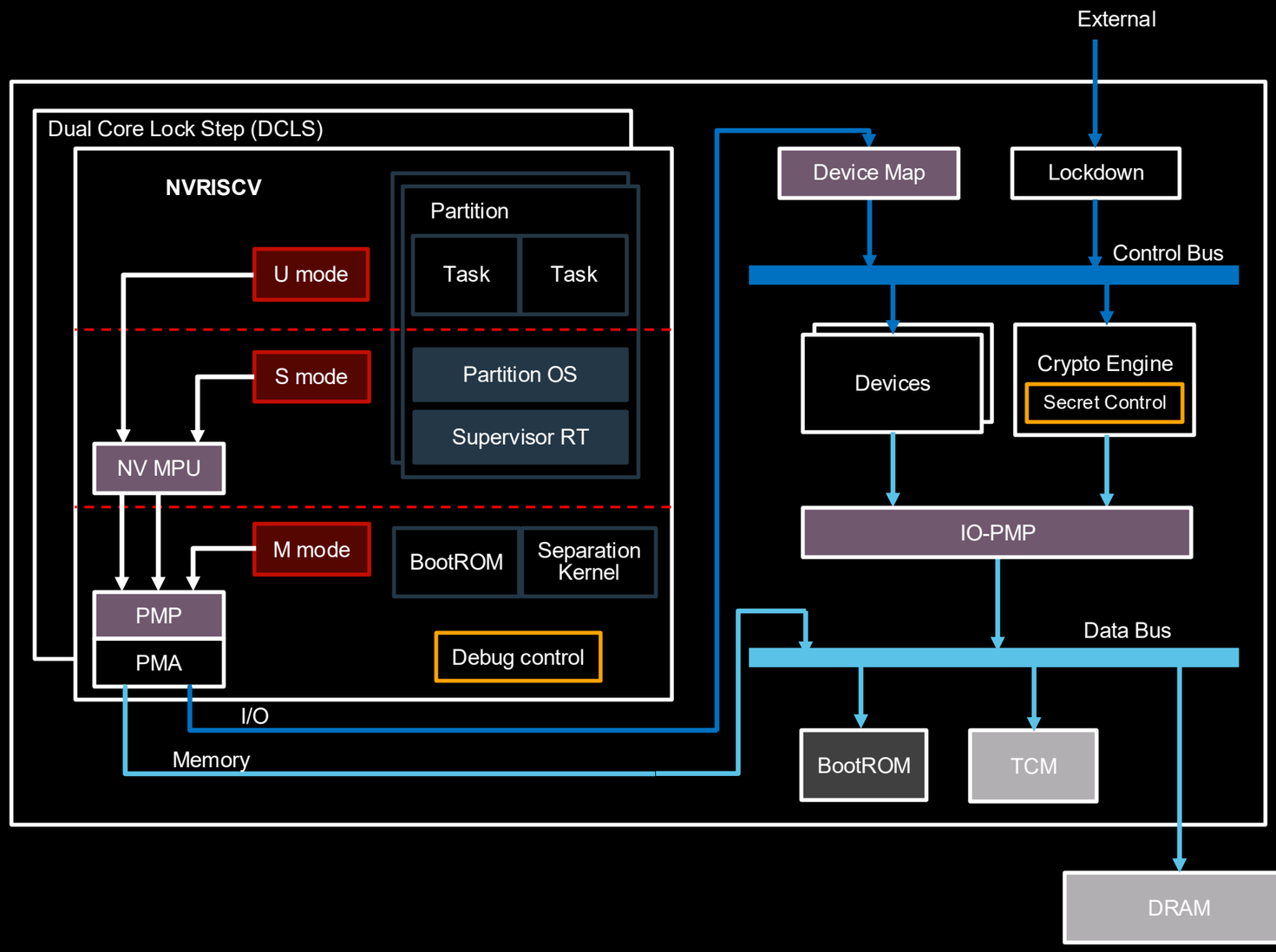
NVIDIA's custom RISC-V extensions to enforce secure boot

- Immutable BootROM



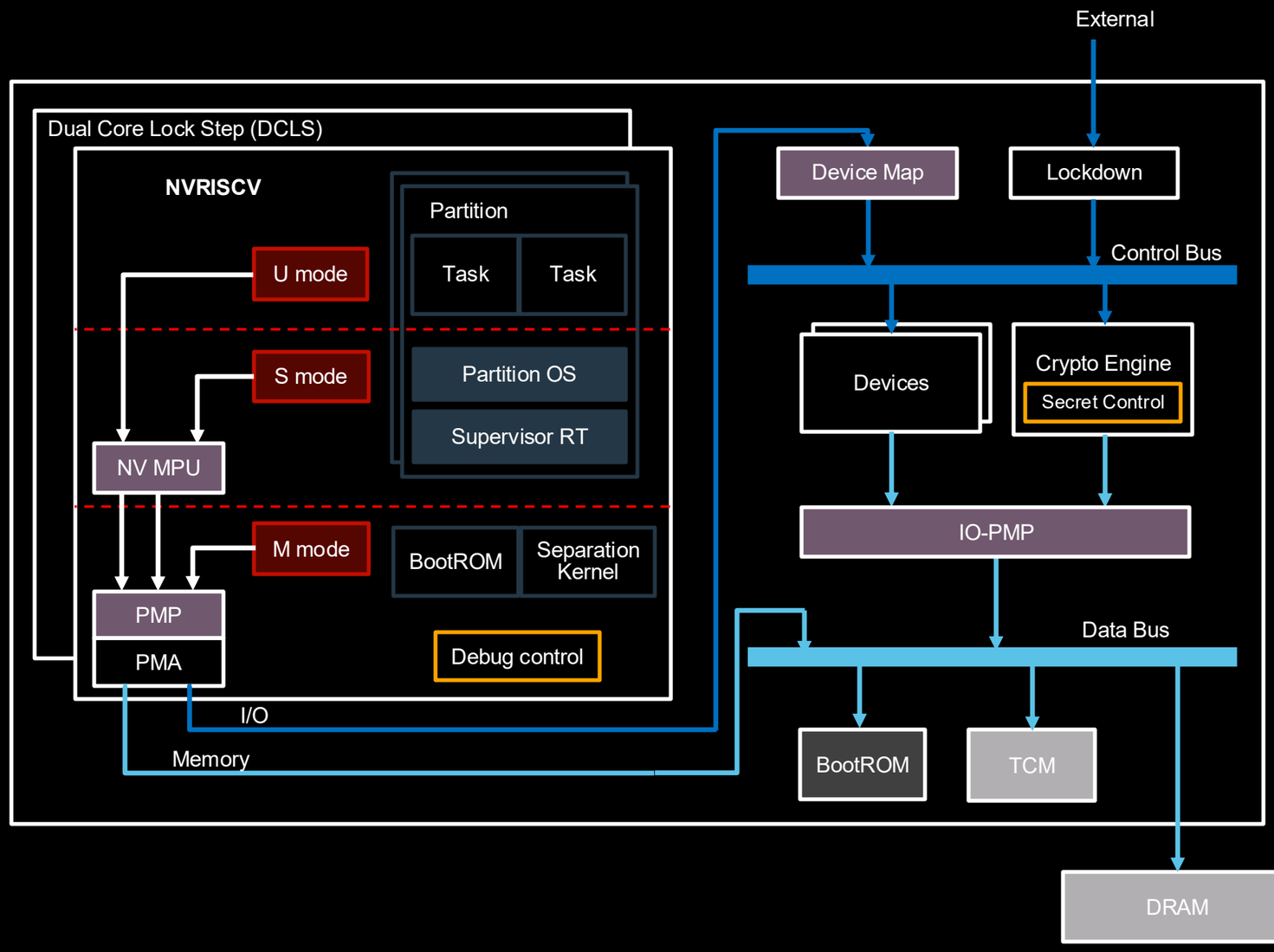
NVIDIA's custom RISC-V extensions to enforce secure boot

- Immutable BootROM
- mromprot (NV extension), XOM



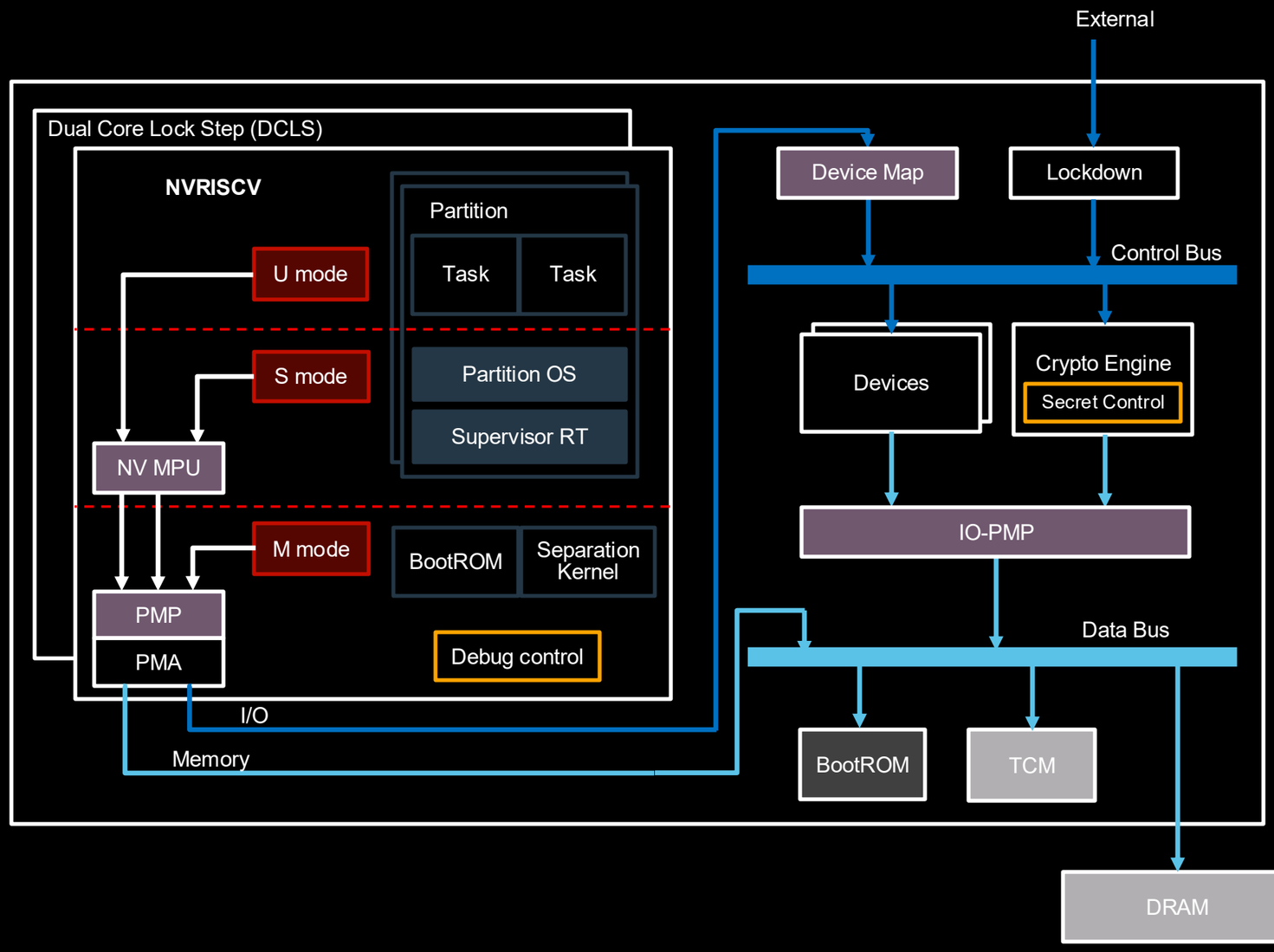
NVIDIA's custom RISC-V extensions to enforce secure boot

- Immutable BootROM
- mromprot (NV extension), XOM
- No return address spill on stack



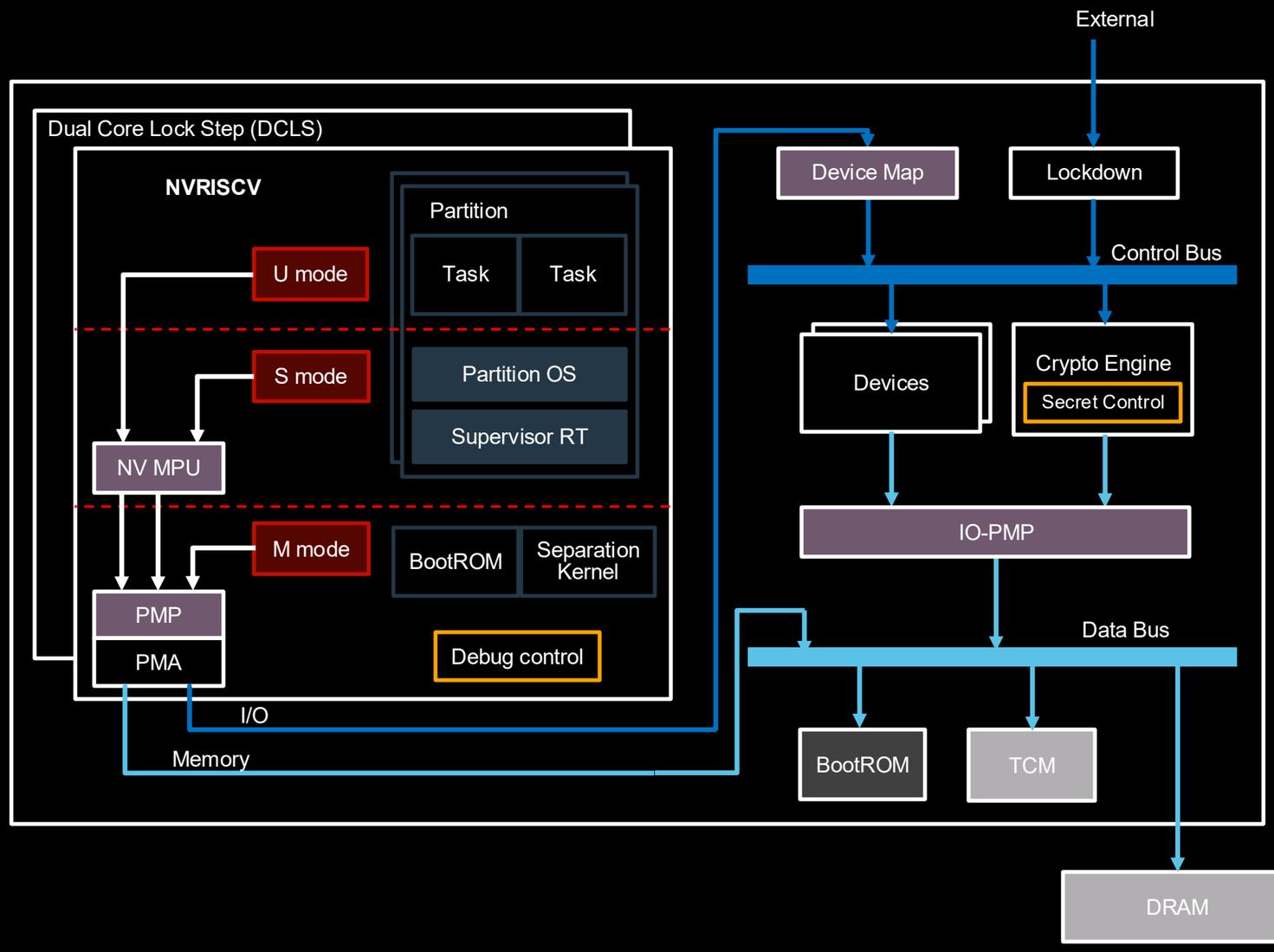
NVIDIA's custom RISC-V extensions to enforce secure boot

- Immutable BootROM
- mromprot (NV extension), XOM
- No return address spill on stack
- External MMIO Lockdown



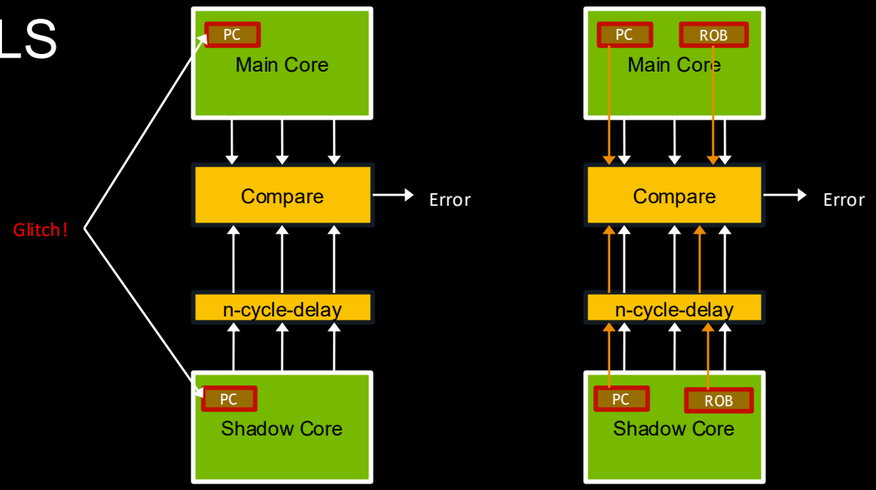
NVIDIA's custom RISC-V extensions to enforce secure boot

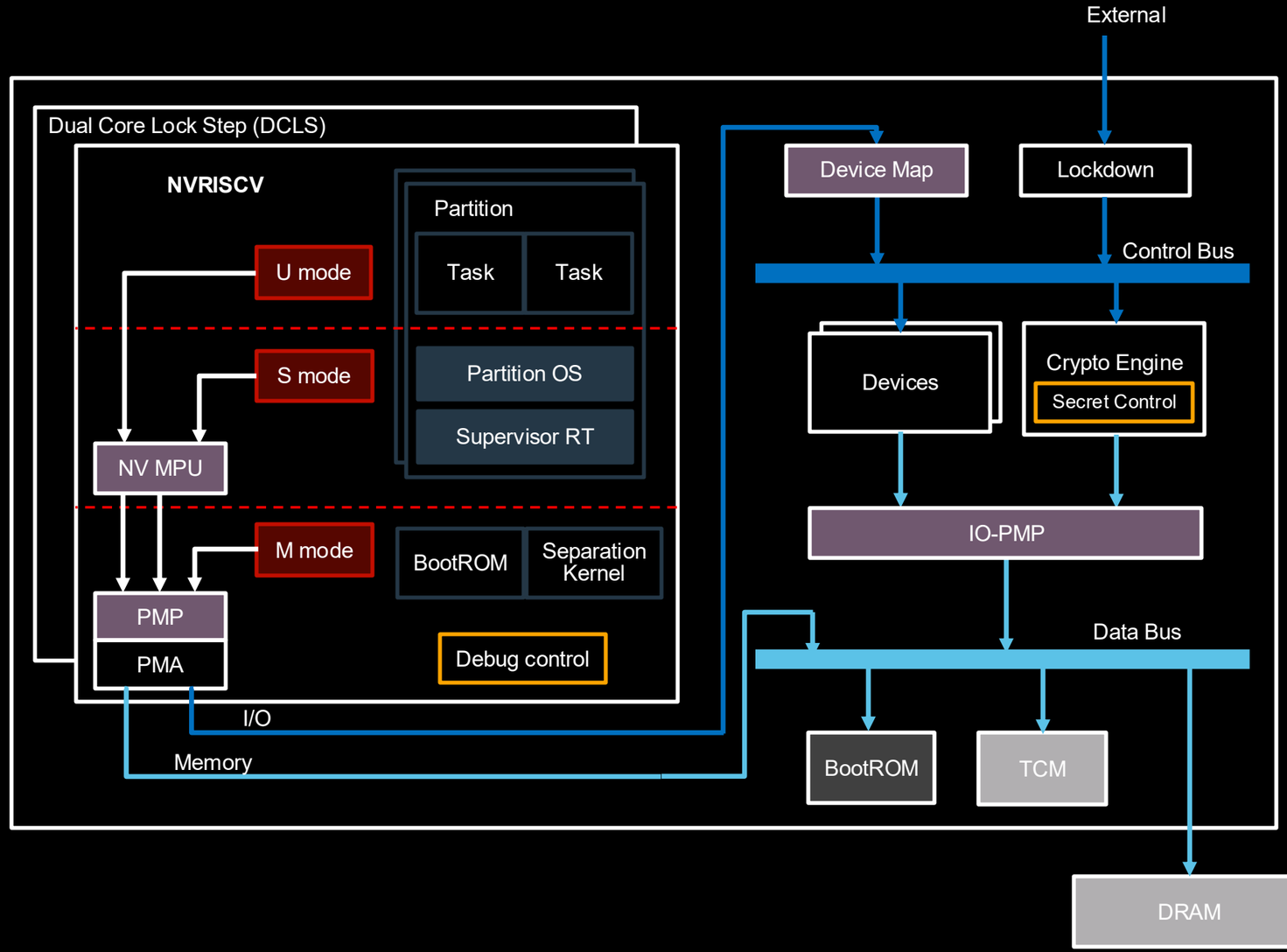
- Immutable BootROM
- mromprot (NV extension), XOM
- No return address spill on stack
- External MMIO Lockdown
- DEF CON 29: "*Glitching RISC-V chips: MTVEC corruption for hardening ISA*"



NVIDIA's custom RISC-V extensions to enforce secure boot

- Immutable BootROM
- mromprot (NV extension), XOM
- No return address spill on stack
- External MMIO Lockdown
- DEF CON 29: "*Glitching RISC-V chips: MTVEC corruption for hardening ISA*"
- DCLS





Language-based security: formally verified components

Language-based security: formally verified components

- Tests can only prove bugs exist, not that they don’t

Language-based security: formally verified components

- Tests can only prove bugs exist, not that they don’t
- SPARK uses contracts and formal verification to prove whole classes of bugs cannot happen

Language-based security: formally verified components

- Tests can only prove bugs exist, not that they don't
- SPARK uses contracts and formal verification to prove whole classes of bugs cannot happen

```
Procedure Do_Operation(X : in out Integer; Y : in out Integer; V : in Integer)
  Precondition:
    V > 0
    X >= V

  Postcondition:
    X = X'Old - V
    Y = Y'Old + V
```

Language-based security: formally verified components

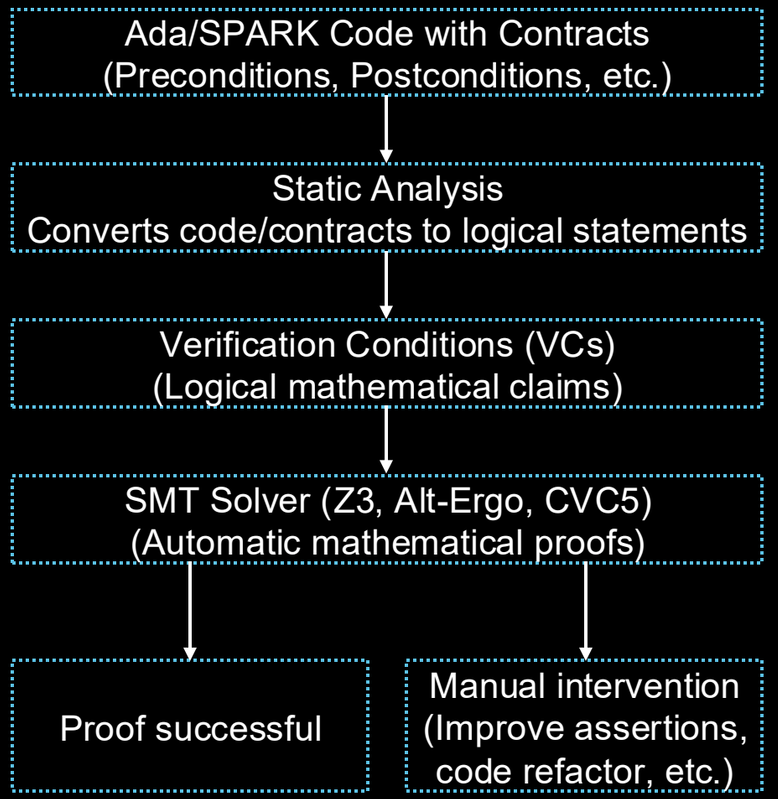
- Tests can only prove bugs exist, not that they don't
- SPARK uses contracts and formal verification to prove whole classes of bugs cannot happen

```
Procedure Do_Operation(X : in out Integer; Y : in out Integer; V : in Integer)
  Precondition:
    V > 0
    X >= V

  Postcondition:
    X = X'Old - V
    Y = Y'Old + V



  begin
     X := X - V;
     Y := Y + V;
  end Do_Operation;
```

Language-based security: formally verified components

- Tests can only prove bugs exist, not that they don't
- SPARK uses contracts and formal verification to prove whole classes of bugs cannot happen

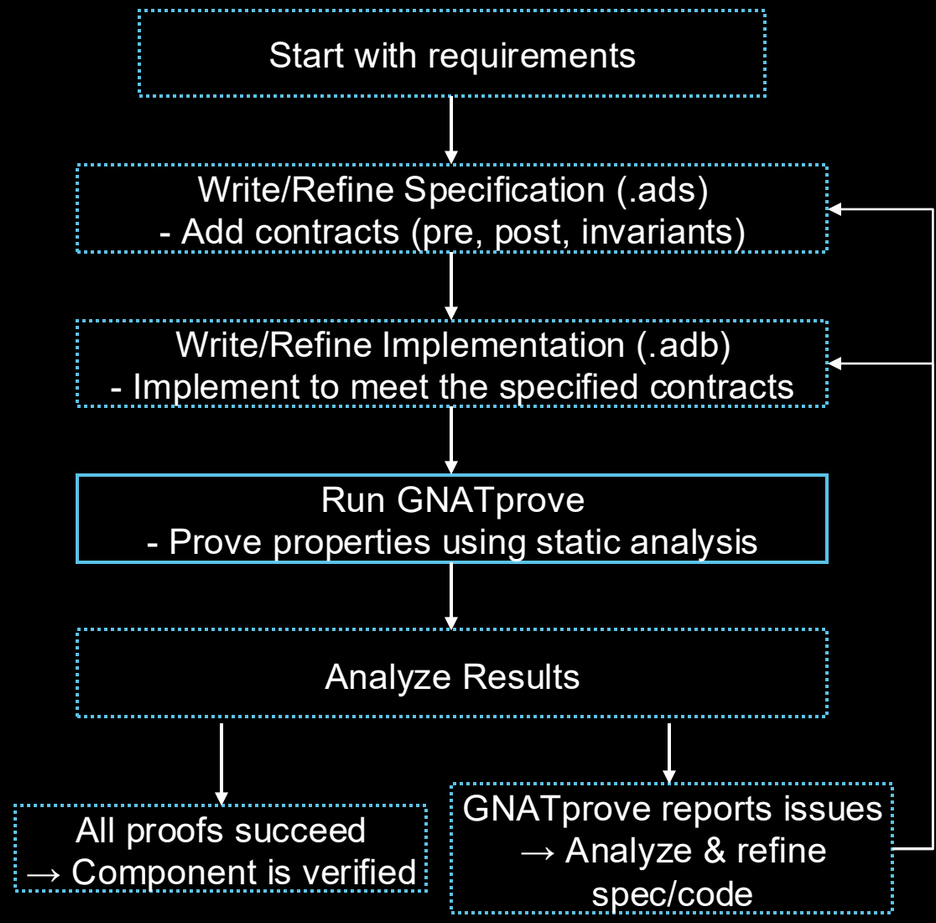
```
Procedure Do_Operation(X : in out Integer; Y : in out Integer; V : in Integer)
  Precondition:
    V > 0
    X >= V

  Postcondition:
    X = X'Old - V
    Y = Y'Old + V



  begin
     X := X - V;
     Y := Y + V;
  end Do_Operation;
```



Language-based security: formally verified components

- Tests can only prove bugs exist, not that they don't
- SPARK uses contracts and formal verification to prove whole classes of bugs cannot happen

```
Procedure Do_Operation(X : in out Integer; Y : in out Integer; V : in Integer)
  Precondition:
    V > 0
    X >= V

  Postcondition:
    X = X'Old - V
    Y = Y'Old + V


  begin
     X := X - V;
     Y := Y + V;
  end Do_Operation;
```



Language-based security: formally verified components

- Tests can only prove bugs exist, not that they don't
- SPARK uses contracts and formal verification to prove whole classes of bugs cannot happen

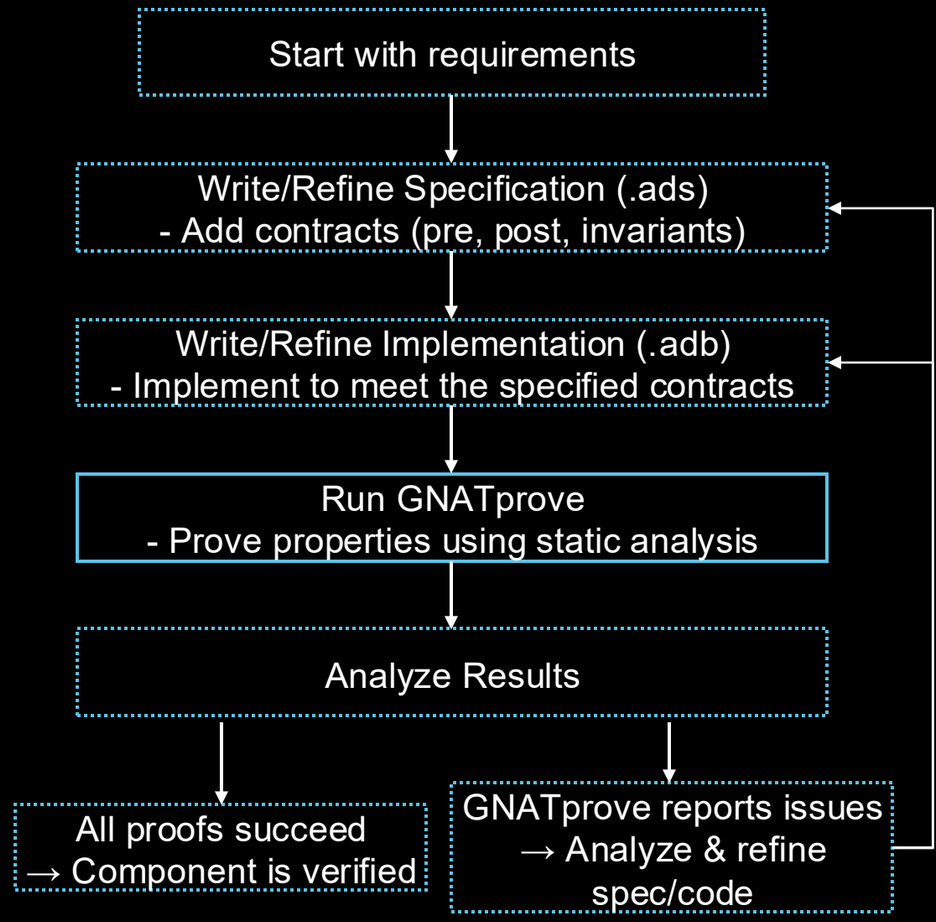
```
Procedure Do_Operation(X : in out Integer; Y : in out Integer; V : in Integer)
  Precondition:
    V > 0
    X >= V

  Postcondition:
    X = X'Old - V
    Y = Y'Old + V

  begin
     X := X - V;
     Y := Y + V;
  end Do_Operation;
```

Proven Procedure → **Tested Procedure**

Preconditions are proven
Postconditions are tested

Tested Procedure → **Proven Procedure**

Preconditions are tested
Postconditions are proven



Language-based security: formally verified components

- Why not do all this with C, Ada, Rust..?

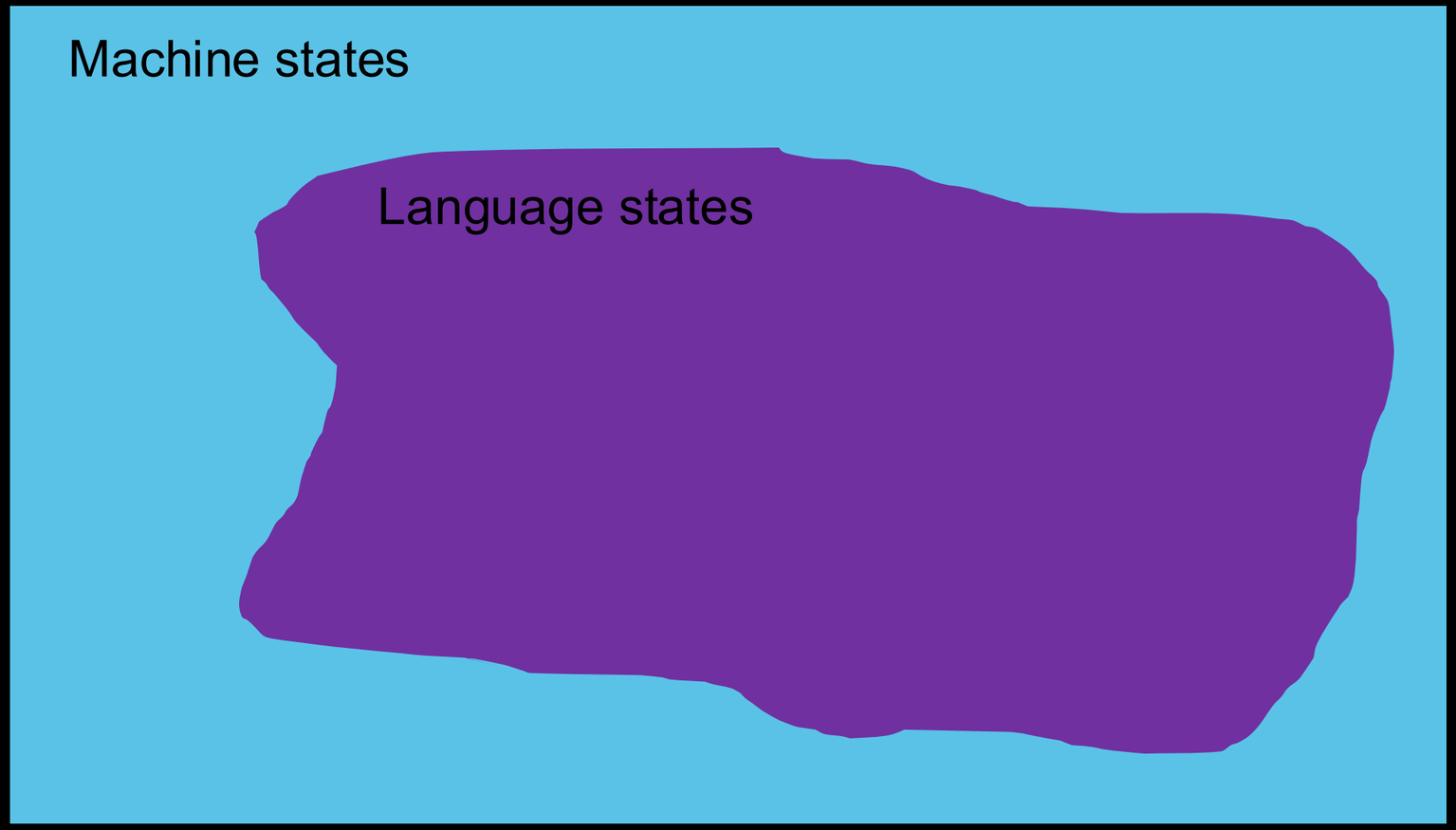
*What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components

Machine states

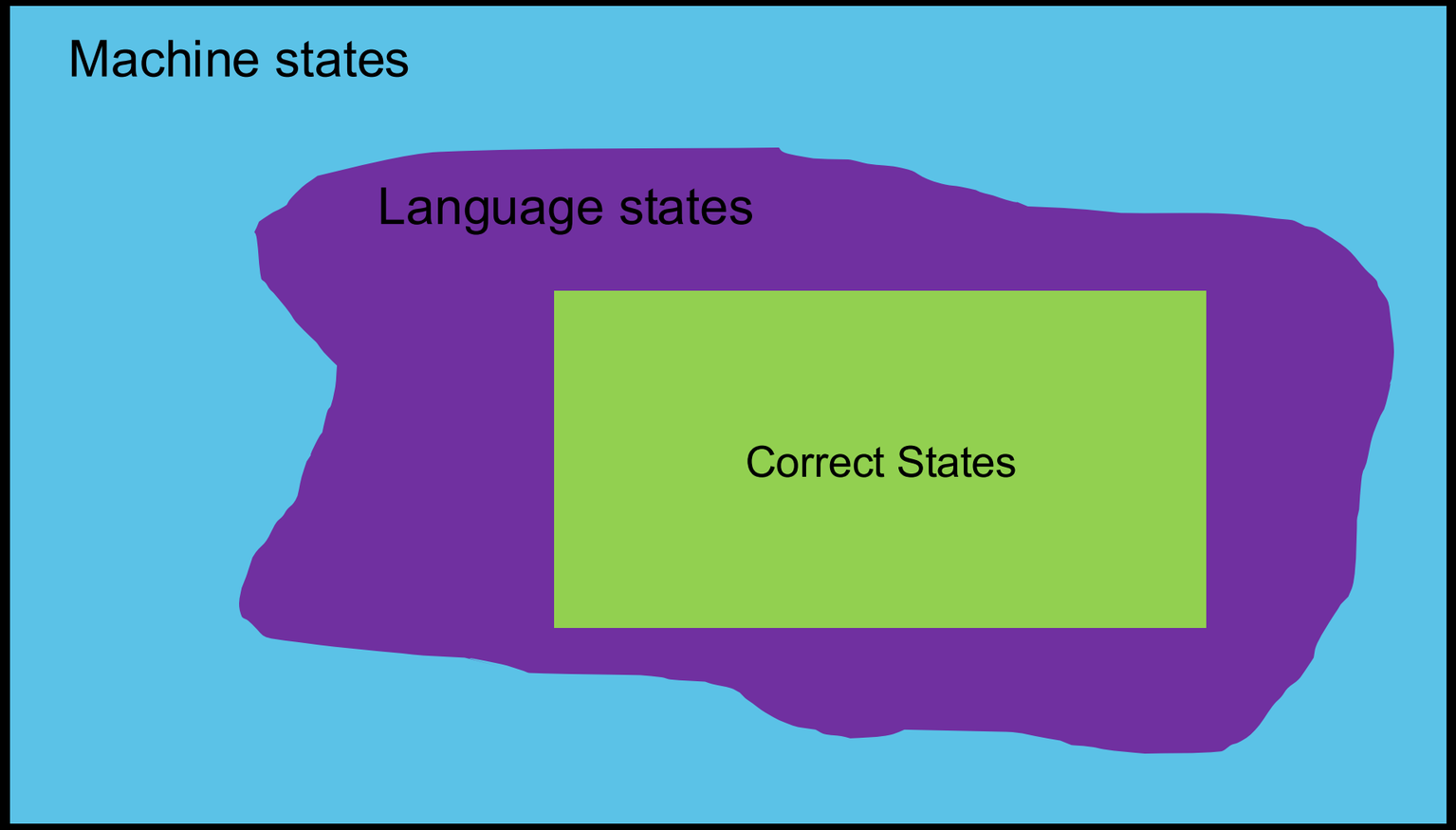
*What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components



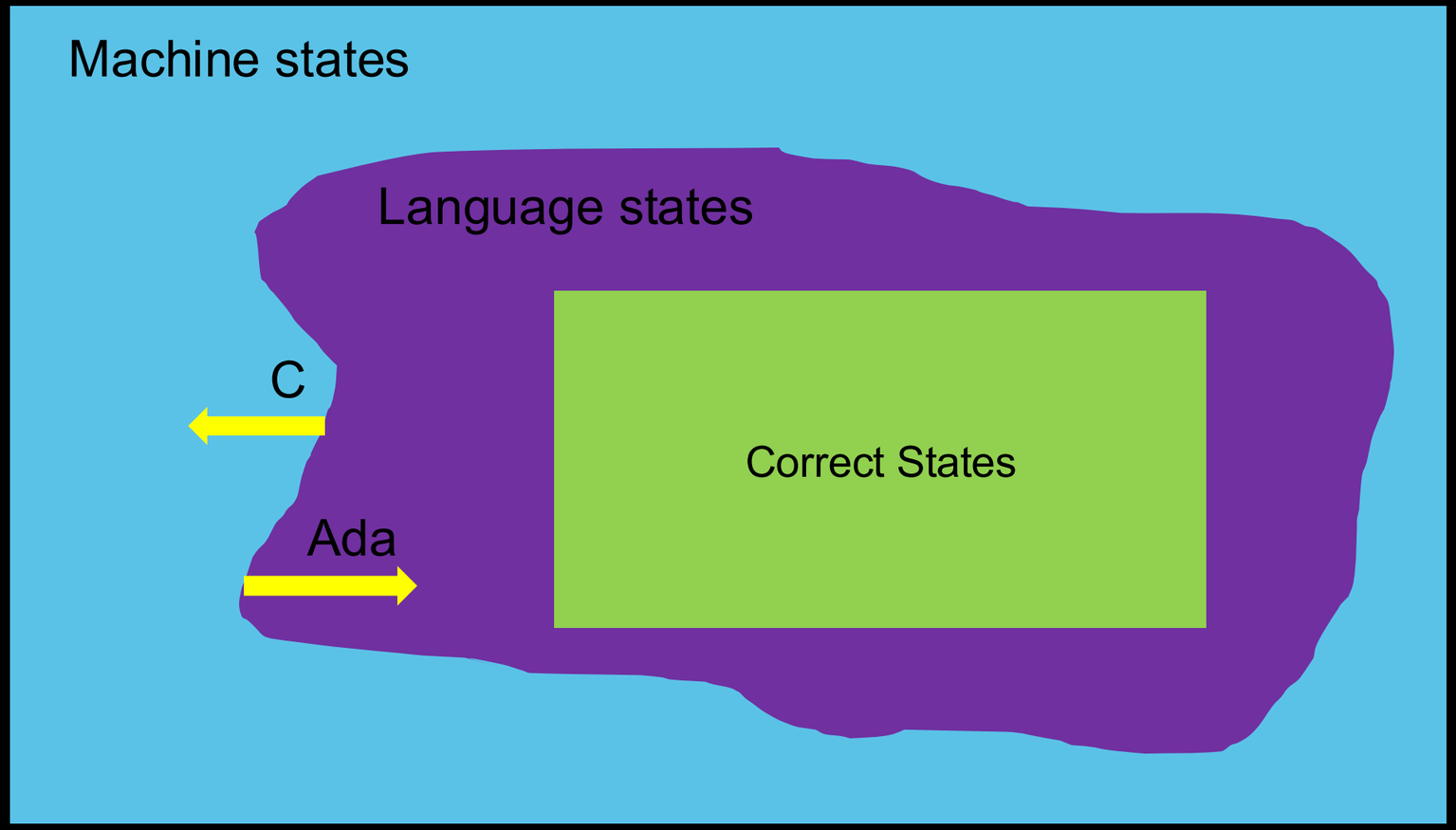
*What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components



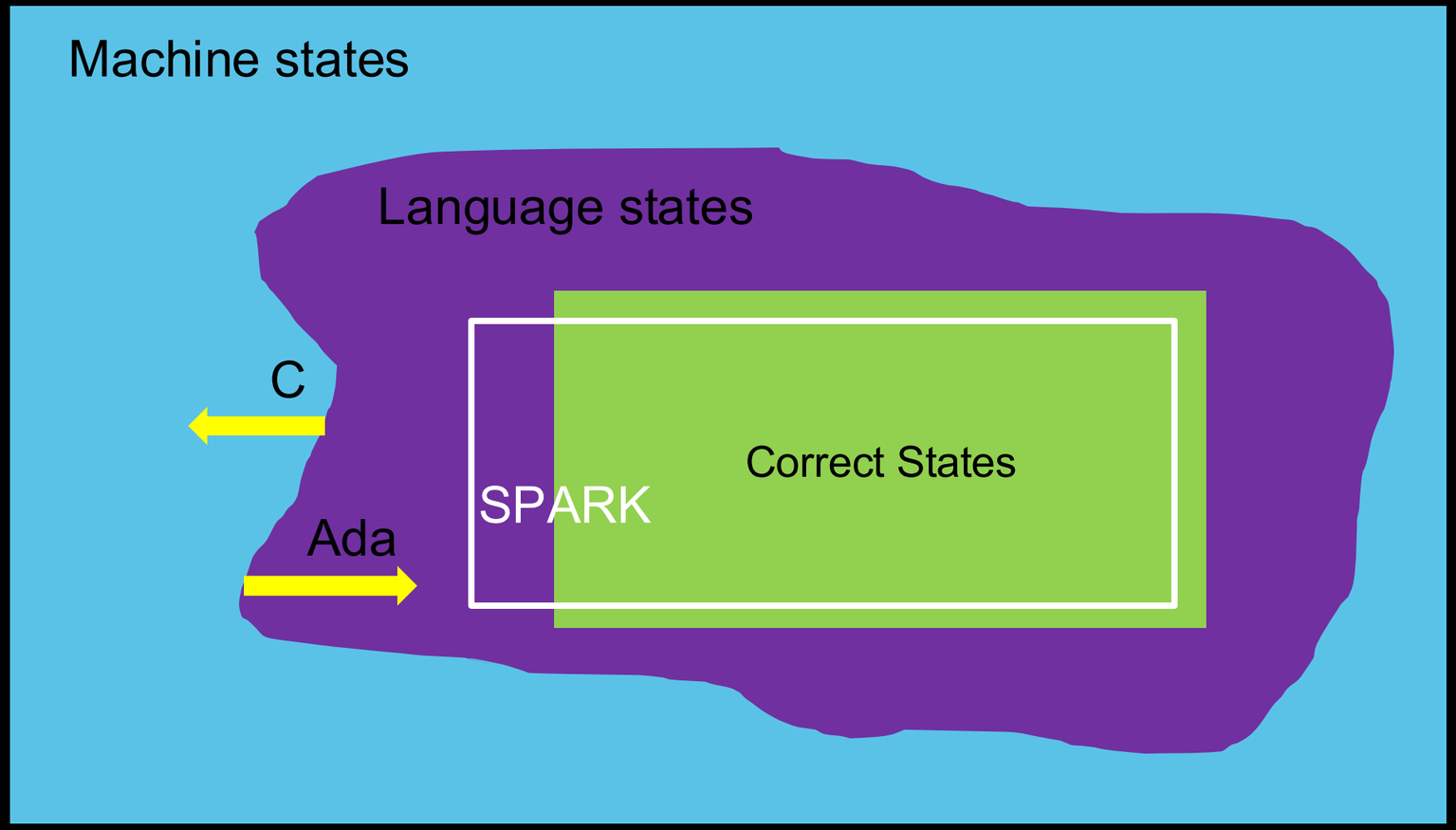
What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components



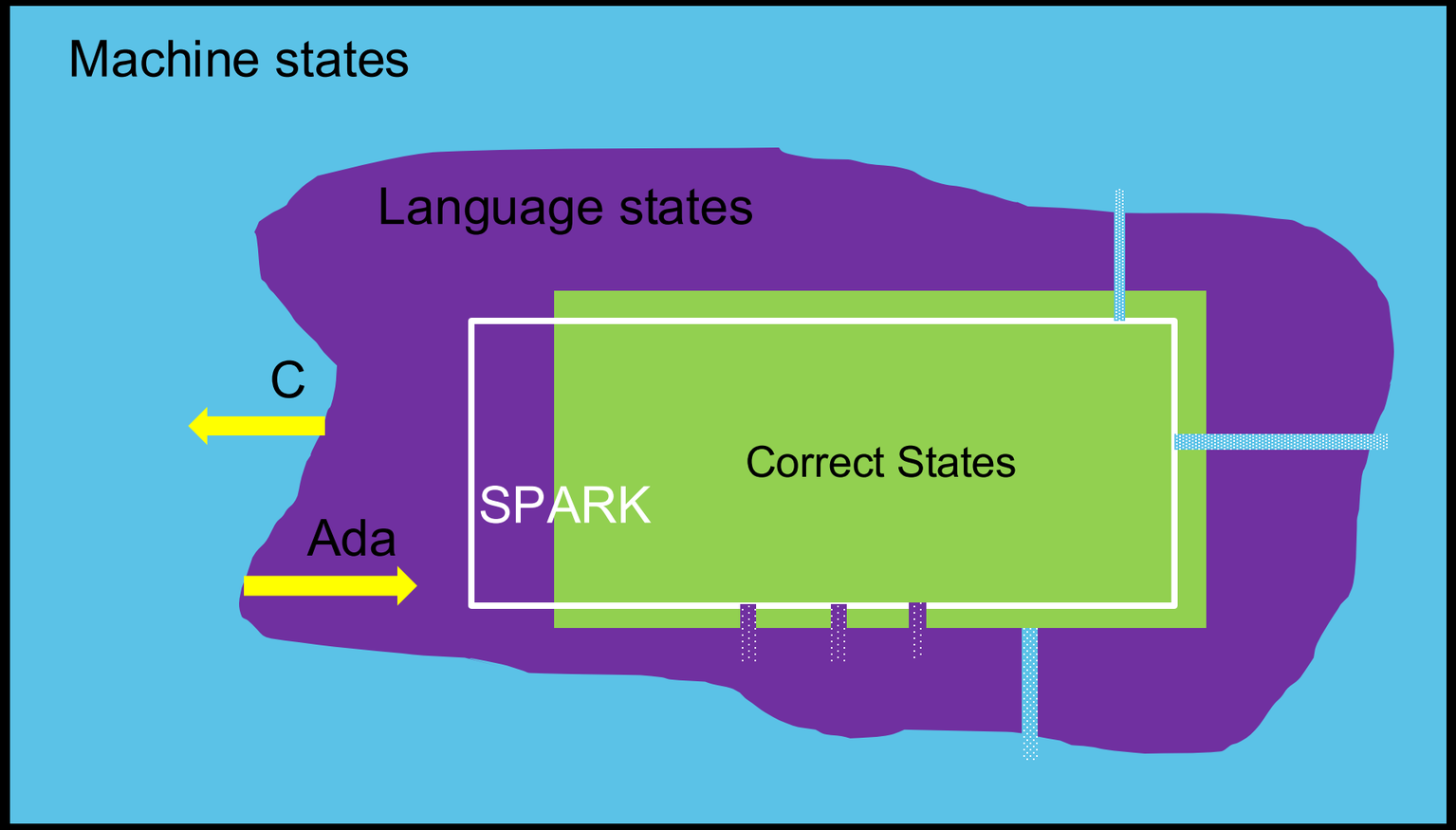
What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components



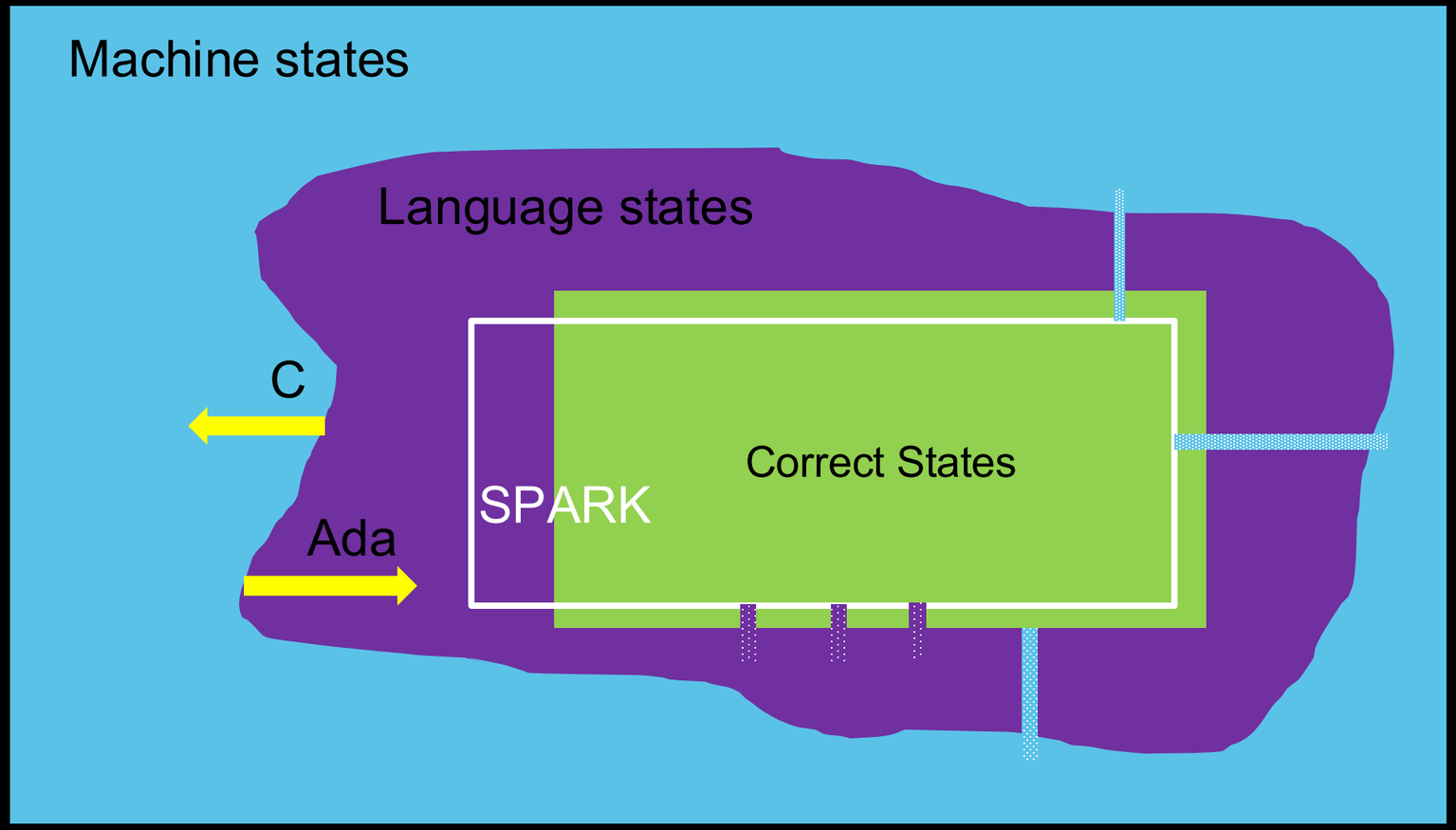
What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components



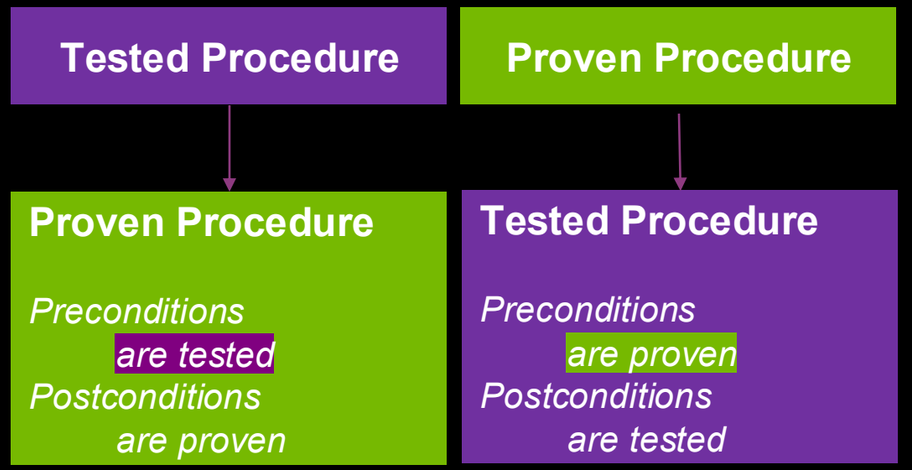
What is Safety-Critical Software, and How Can Ada and SPARK Help?

Language-based security: formally verified components



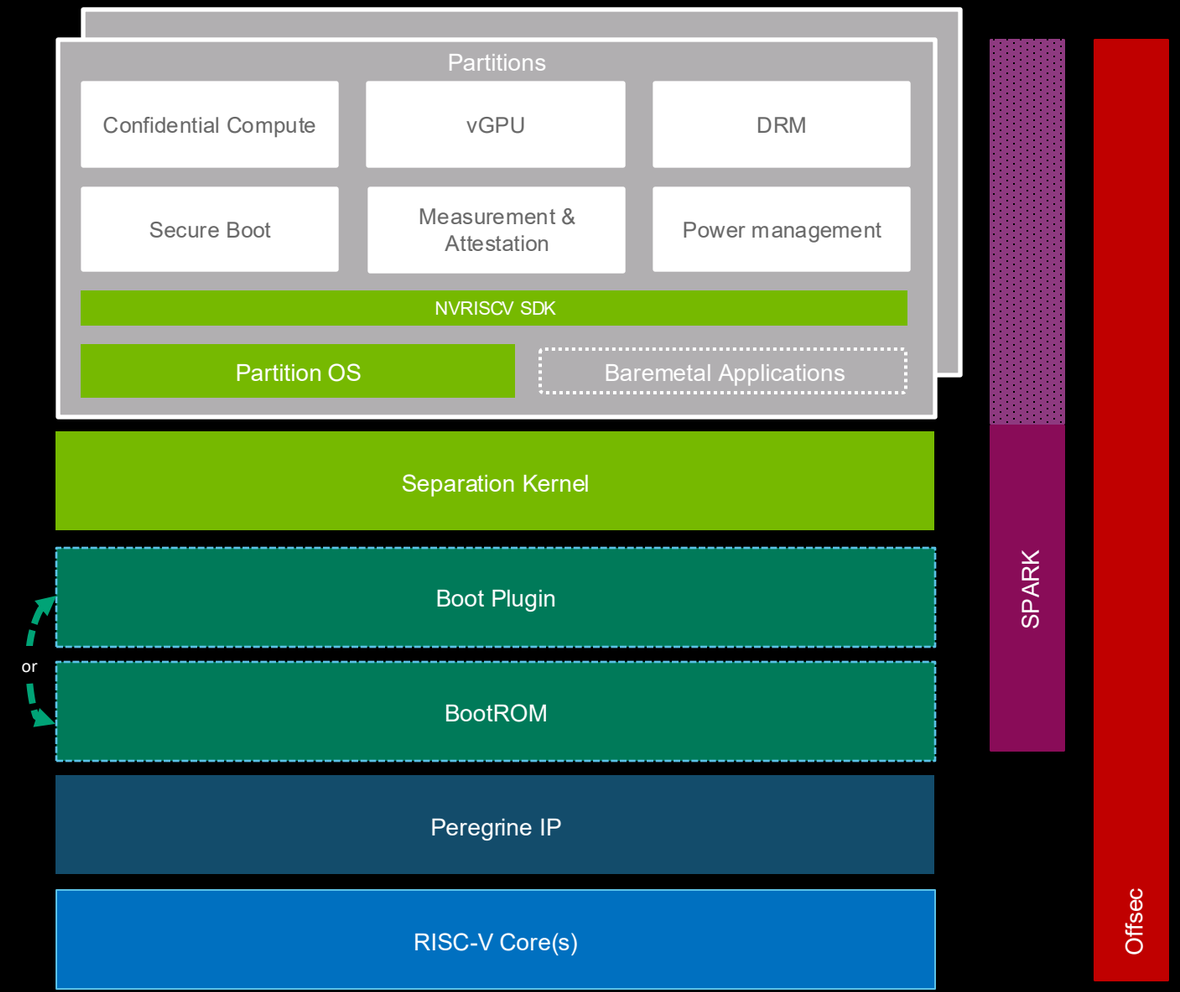
What is Safety-Critical Software, and How Can Ada and SPARK Help?

```
Procedure Do_Operation(X : in out Integer; Y :
in out Integer; V : in Integer)
  Precondition:
    V > 0
    X >= V

  Postcondition:
    X = X'Old - V
    Y = Y'Old + V
```



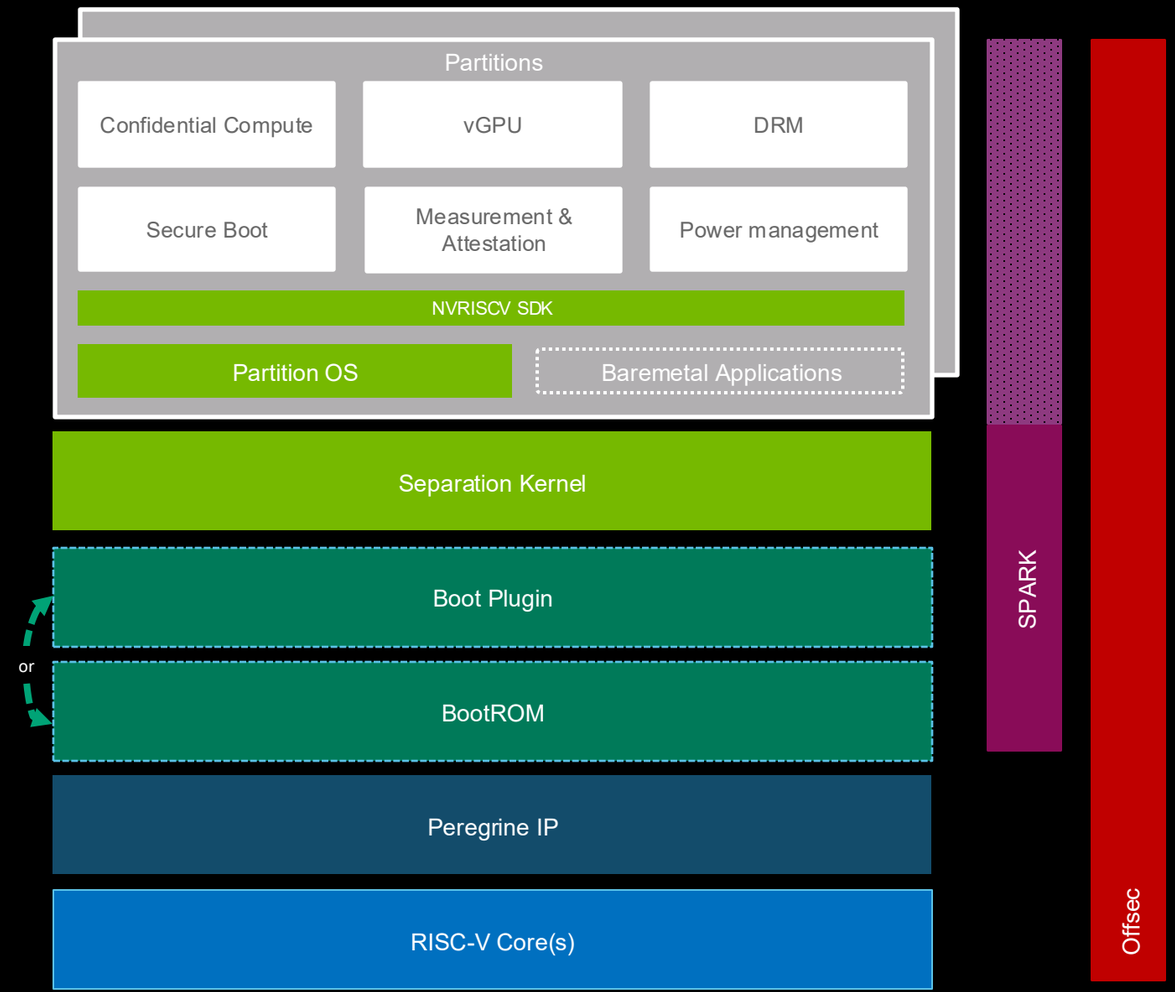
Foundation for running mixed-criticality applications

- Partitions are isolated execution environments where applications run
- Core SW formally verified to be free of runtime errors (AoRTE)



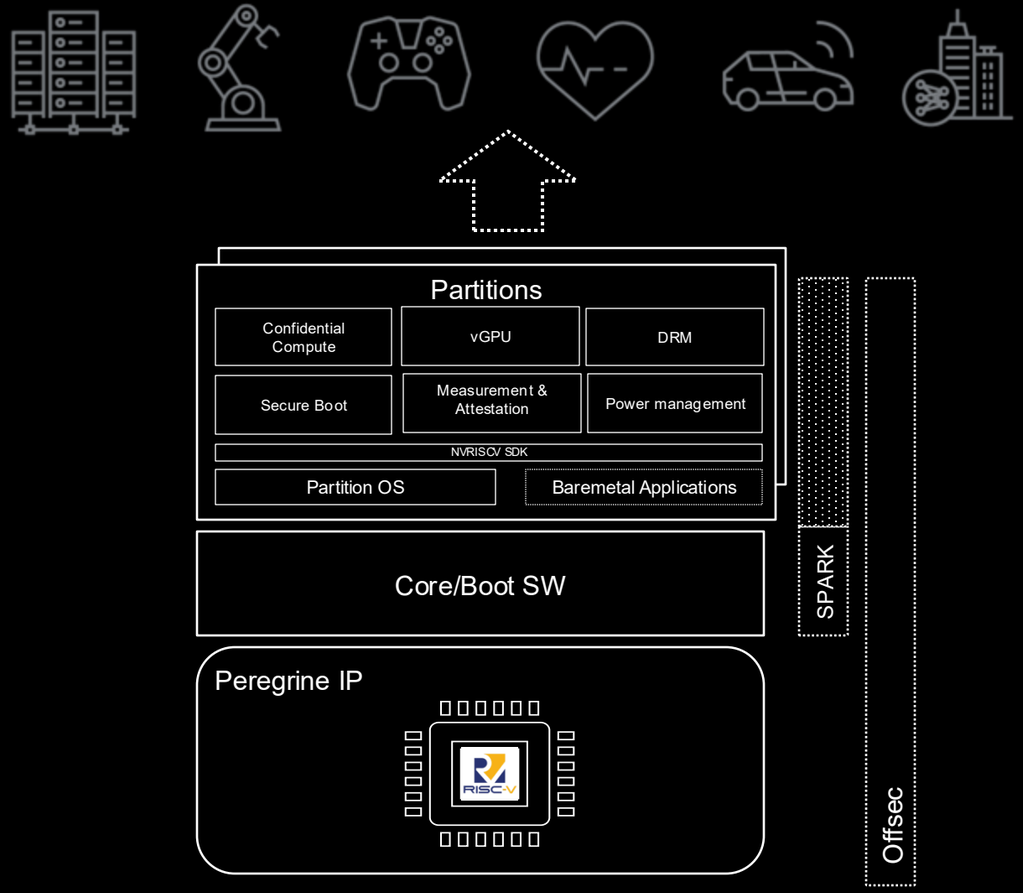
Foundation for running mixed-criticality applications

- Partitions are isolated execution environments where applications run
- Core SW formally verified to be free of runtime errors (AoRTE)
- Hardware never speculates past privilege mode switch
- Hardware never speculates past CSR read
- Speculative D cache refill is disabled
- Branch predictor partitioned between privilege modes



Practical takeaways from designing and deploying a billion-core secure system

- Think Holistically
- HW/SW Co-Design is a must
- Standardize When You Can, Innovate When You Must
- Memory Safety is a Hardware Problem Too
- No Silver Bullets – Layered Defense is Essential



Lessons learned

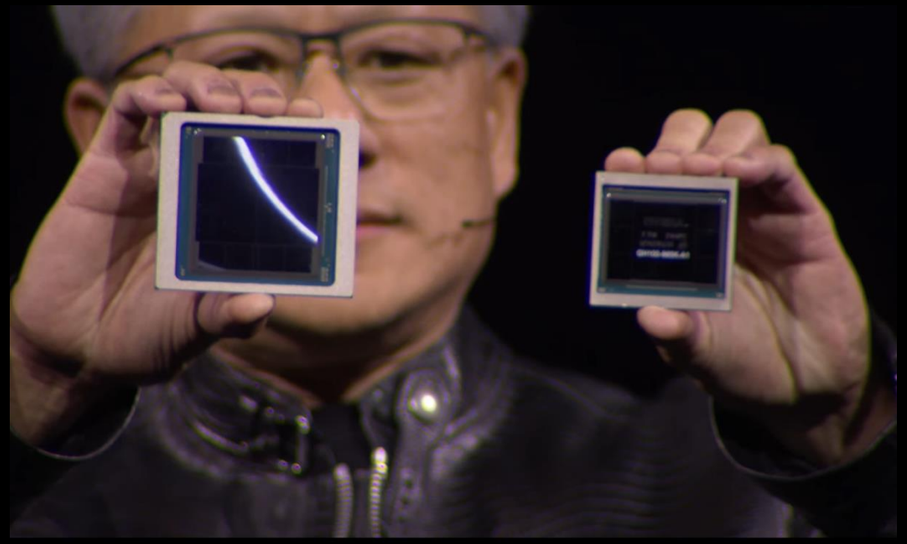
- Hardware extensions are not enough
  - The BIGGEST attack surface is <u>software</u>
    - HW and SW must cooperate to create a secure ecosystem (HW CFI, MTE, HFI, more)

Lessons learned

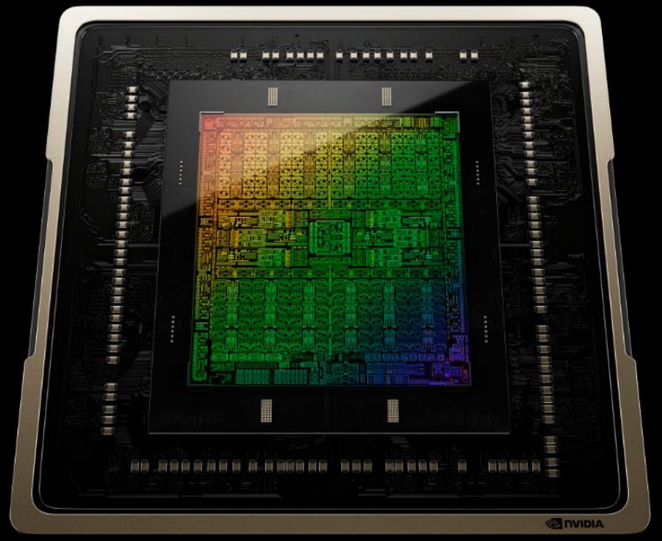
- Hardware extensions are not enough
  - The BIGGEST attack surface is software
    - HW and SW must cooperate to create a secure ecosystem (HW CFI, MTE, HFI, more)
  - Formally verified languages (like Ada/SPARK) and memory safe languages (like Rust) are great!
    - Significant security ROI but costs are substantial
      - It is likely to have “hybrid” software for a while
    - Non-memory safety vulnerabilities still exist and affect both type of languages
      - DefCon 30: Adam Zabrocki, Alex Tereshkin - Exploitation in the era of Formal Verification https://www.youtube.com/watch?v=TclaZ9LW1WE

Lessons learned

- Hardware extensions are not enough
  - The BIGGEST attack surface is software
    - HW and SW must cooperate to create a secure ecosystem (HW CFI, MTE, HFI, more)
  - Formally verified languages (like Ada/SPARK) and memory safe languages (like Rust) are great!
    - Significant security ROI but costs are substantial
      - It is likely to have “hybrid” software for a while
    - Non-memory safety vulnerabilities still exist and affect both type of languages
      - DefCon 30: Adam Zabrocki, Alex Tereshkin - Exploitation in the era of Formal Verification https://www.youtube.com/watch?v=TclaZ9LW1WE
  - Creating innovative ecosystems demands a forward-thinking mindset:
    - Flexibility should support adaptation to ecosystem evolution forecasting in both HW and SW
      - Something which is not a problem today, can be a critical vulnerability tomorrow (e.g., side channels)
      - Being part of various initiatives/organizations is important
        - It helps identify industry trends and make informed predictions, even if the signals aren't always obvious.
    - Scalability, flexibility, performance, reliability and security should be considered collectively, not separately.
    - Hybrid attacks (not just pure SW or pure HW) are likely to be rising (Rowhammer, speculative execution, etc.)



Q&A



Private contact:

http://pi3.com.pl
pi3@pi3.com.pl
Twitter: @Adam_pi3

Adam ‘pi3’ Zabrocki

NVIDIA

Private contact:

markomitic.net
linkedin.com/markomitic
Twitter: @markomitic

Marko Mitic