BEDROCK
Systems Inc

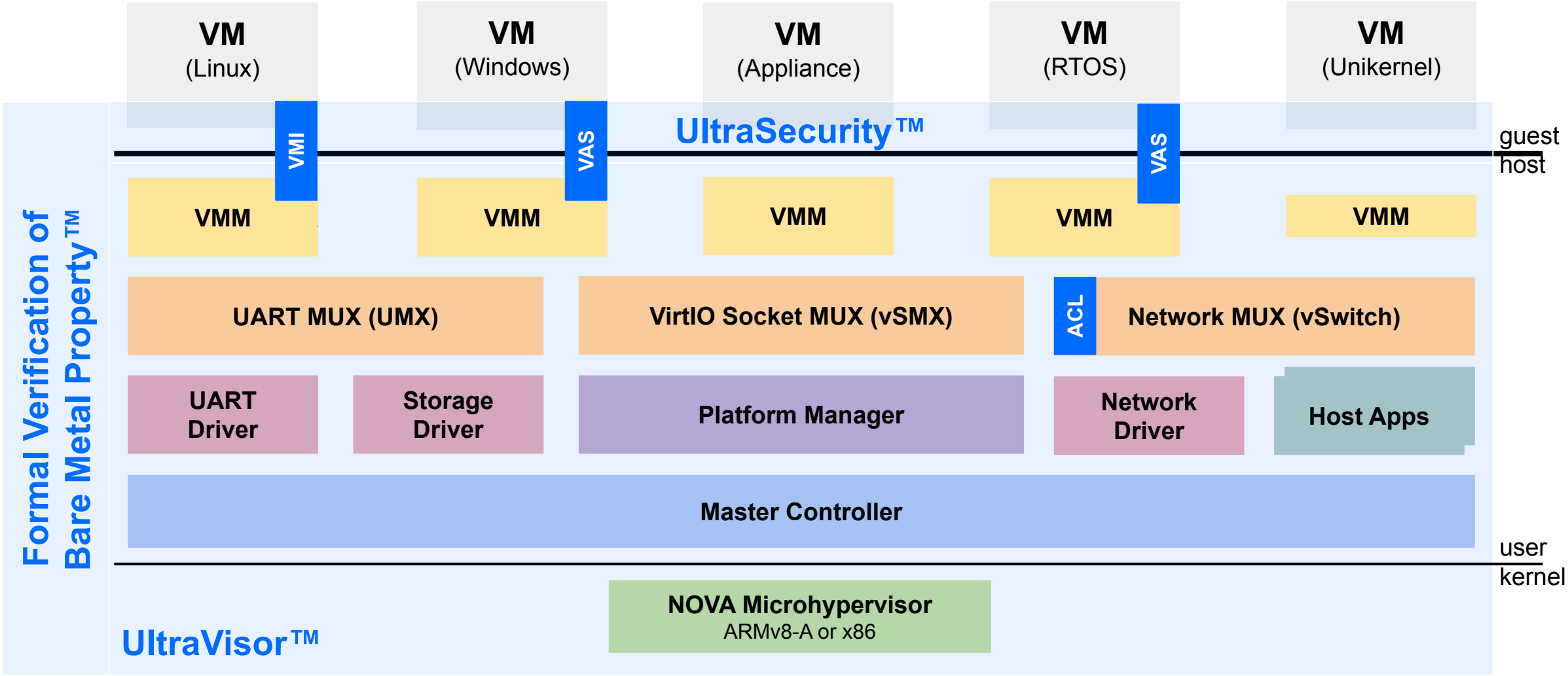
NOVA Microhypervisor Feature Update

Udo Steinberg

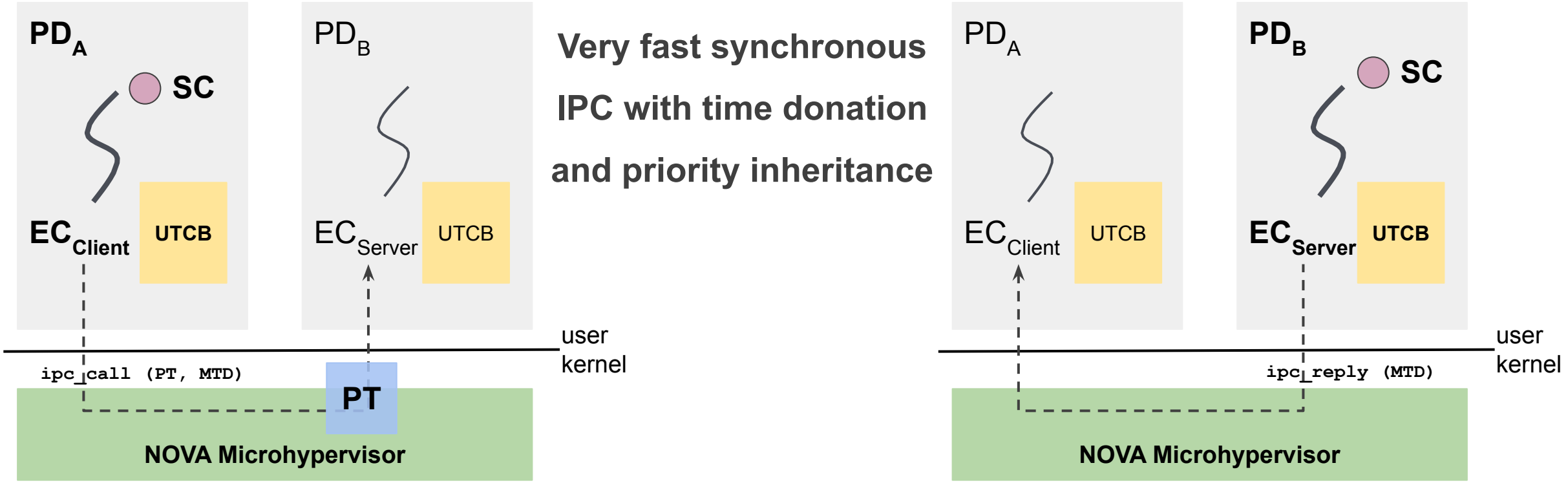
Agenda

- Architecture Overview
- NOVA Building Blocks
- Recent Innovations
- ARM/x86 Code Unification
- Advanced Security Features (x86)
- Performance
- Q&A

Architecture Overview

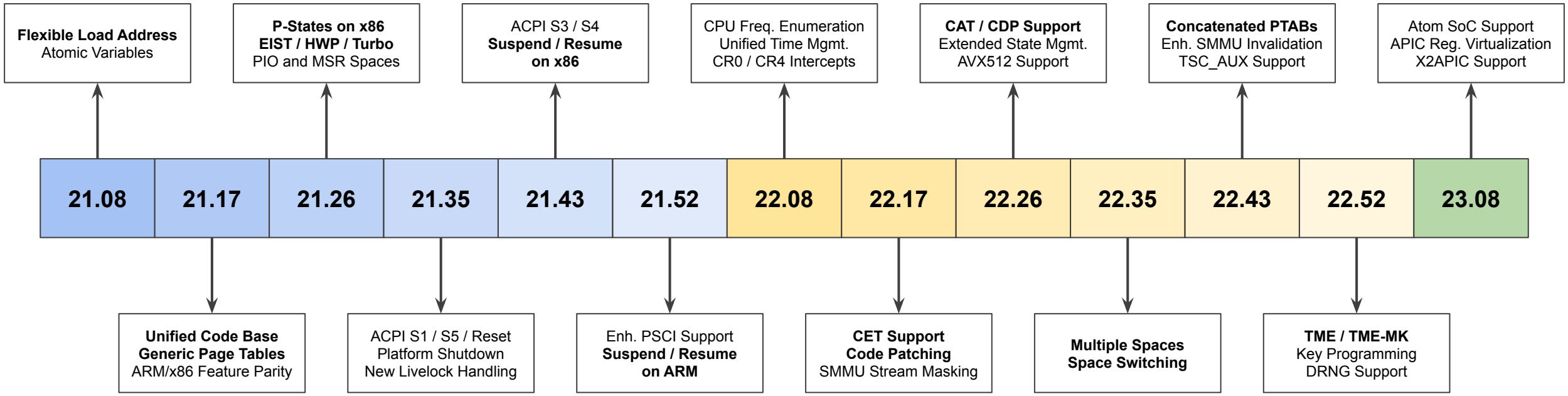


NOVA Microhypervisor Building Blocks



- Protection Domains, Execution+Scheduling Contexts, Portals, Semaphores
- Hypercall interface uses capability-based access control for all operations

NOVA Innovation Timeline



Approximately 2 months between releases

NOVA Design Goals

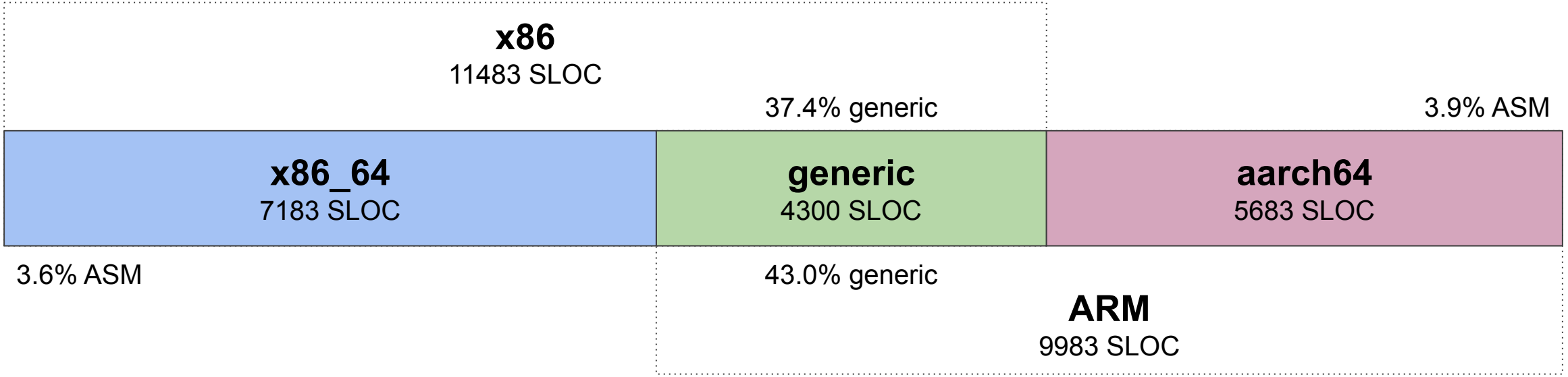
- Provide the same (or similar) functionality across all architectures
- Generic API that abstracts from architectural differences as much as possible
- Simple Build Infrastructure
  - `make ARCH= BOARD=`
- Standardized Boot Process and Resource Enumeration
  - Use Multiboot v2/v1, UEFI, ACPI, PSCI
- Formal Verification of highly concurrent C++ code and weakly-ordered memory
- Modern, Small, Fast: Best-in-Class Security and Performance

Code Structure for Multiple Architectures

- Functions can be categorized as
  - Same API / Same Implementation
    - can share source/header/spec file
  - Same API / Different Implementation
    - can potentially share header/spec file
  - Different API and Implementation
    - cannot share anything
- Architecture-specific files will override generic files with the same name

- **NOVA**
  - `Makefile`
  - `build-aarch64`
  - `build-x86_64`
  - `doc`
  - **`inc`** ← ***generic***
    - **`aarch64`** ← *specific*
    - **`x86_64`** ← *specific*
  - **`src`** ← ***generic***
    - **`aarch64`** ← *specific*
    - **`x86_64`** ← *specific*

Unified Code Base



NOVA x86 Binary (ACPI)

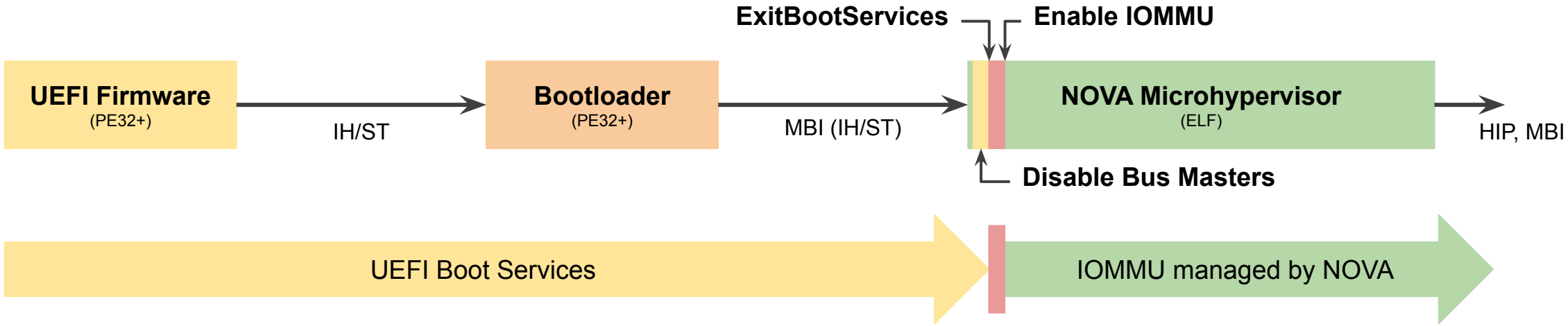
- 68808 Bytes Code
- 2464 Bytes Data

NOVA ARM Binary (ACPI)

- 66916 Bytes Code
- 300 Bytes Data

SLOC based on **release-23.08.0**, binary sizes based on **gcc-12.2.0** build. Other versions will produce different numbers.

Protecting against Boot-Time DMA Attacks



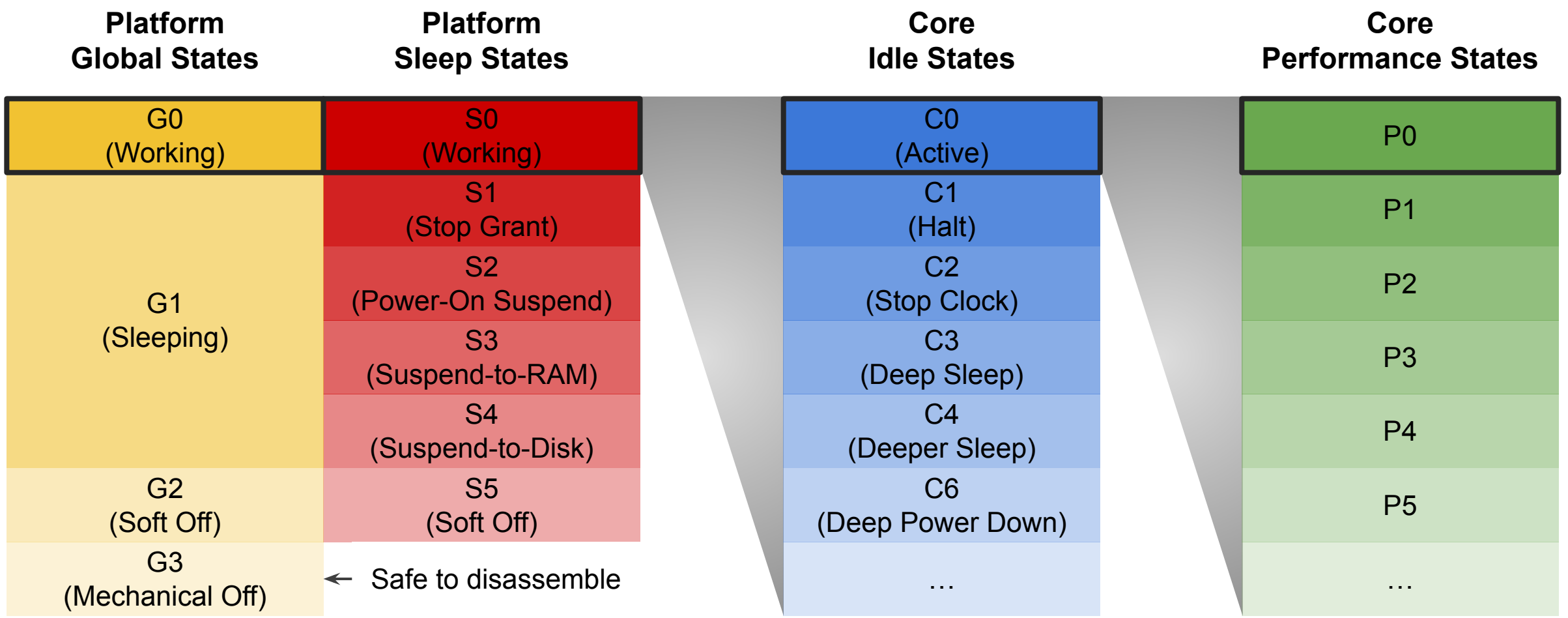
- Firmware owns platform hardware prior to UEFI ExitBootServices
- ExitBootServices drops DMA protections for Legacy OS Support 😠
- Window of opportunity for DMA attack before NOVA can enable IOMMU

⇒ NOVA disables bus masters and manages ExitBootServices flow

Flexible Load Address

- No physical load-address range works across all platforms/architectures
  - Some platforms have RAM starting at 0, some have MMIO starting at 0
  - Bootloader may want to move image to end of memory
- Load address now flexible (can move NOVA up/down by multiples of 2 MiB)
  - No ELF relocation support needed
  - Init section mapped 1:1 and uses position-independent (mode-independent) code
  - Runtime section mapped V⇒P via paging and uses virtual memory
- Multiboot protocol deficiencies currently limit load address to < 4 GiB
  - Many multiboot structures defined as 32-bit and nothing defined for aarch64 😠

Power Management: Overview of ACPI States



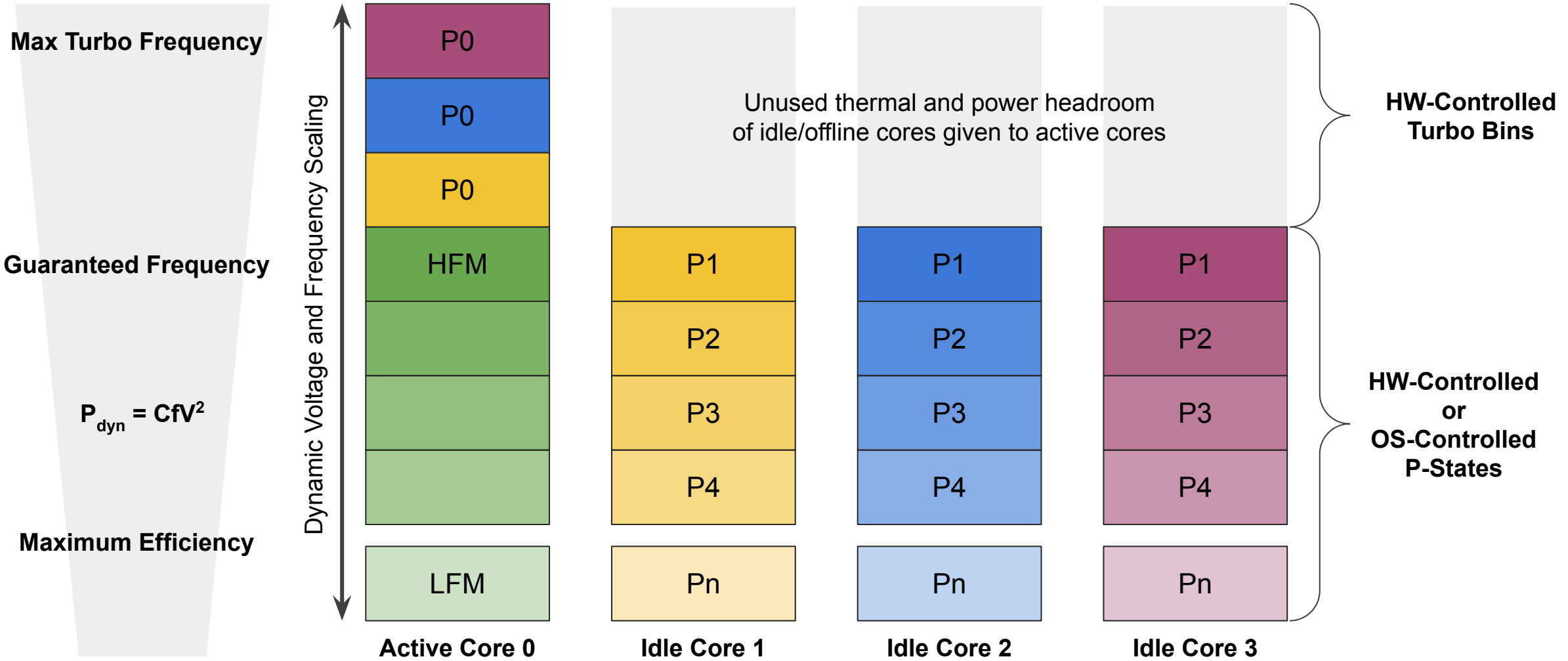
Suspend/Resume: Possible Approaches

- Either save/restore entire register set of all devices
  - Significant amount of state to manage
  - Does not work for devices with hidden “internal state” or complex state machines

  ⇒ Possibly suitable for generic devices managed by some other component
- Or save high-level configuration and reinitialize devices based on that
  - Less information to maintain ⇒ faster save/restore
  - Resume similar to first initialization ⇒ using same code paths as initial boot

  ⇒ Approach for all devices managed by the NOVA microhypervisor

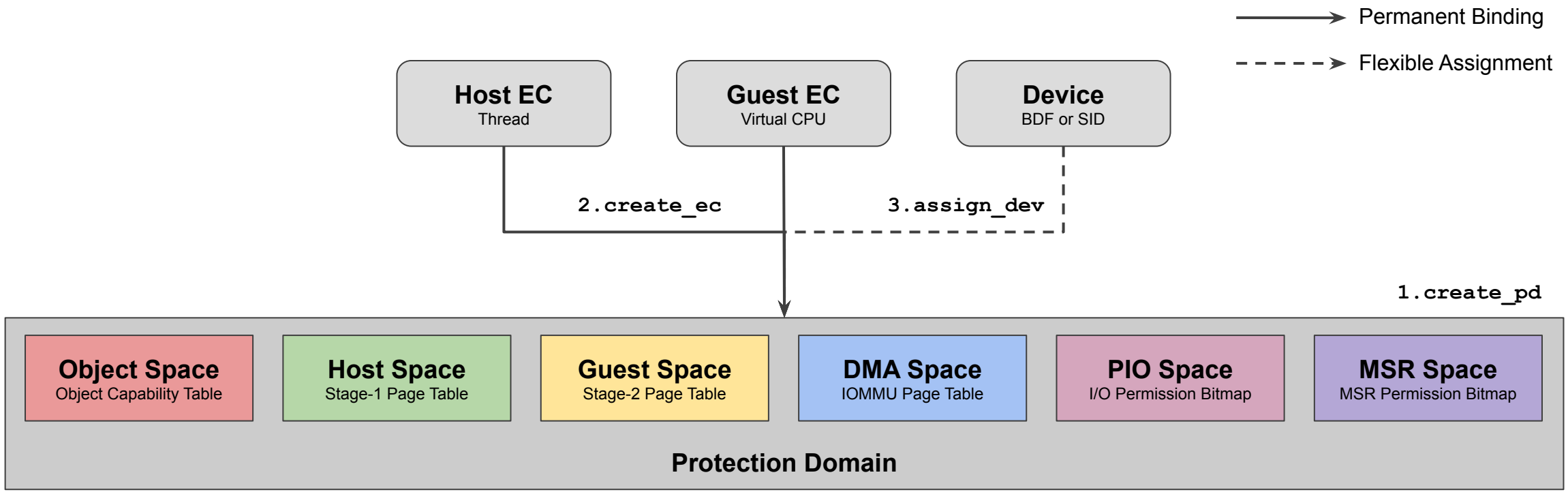
Performance: ACPI P-States on x86



Power Management: Feature Comparison

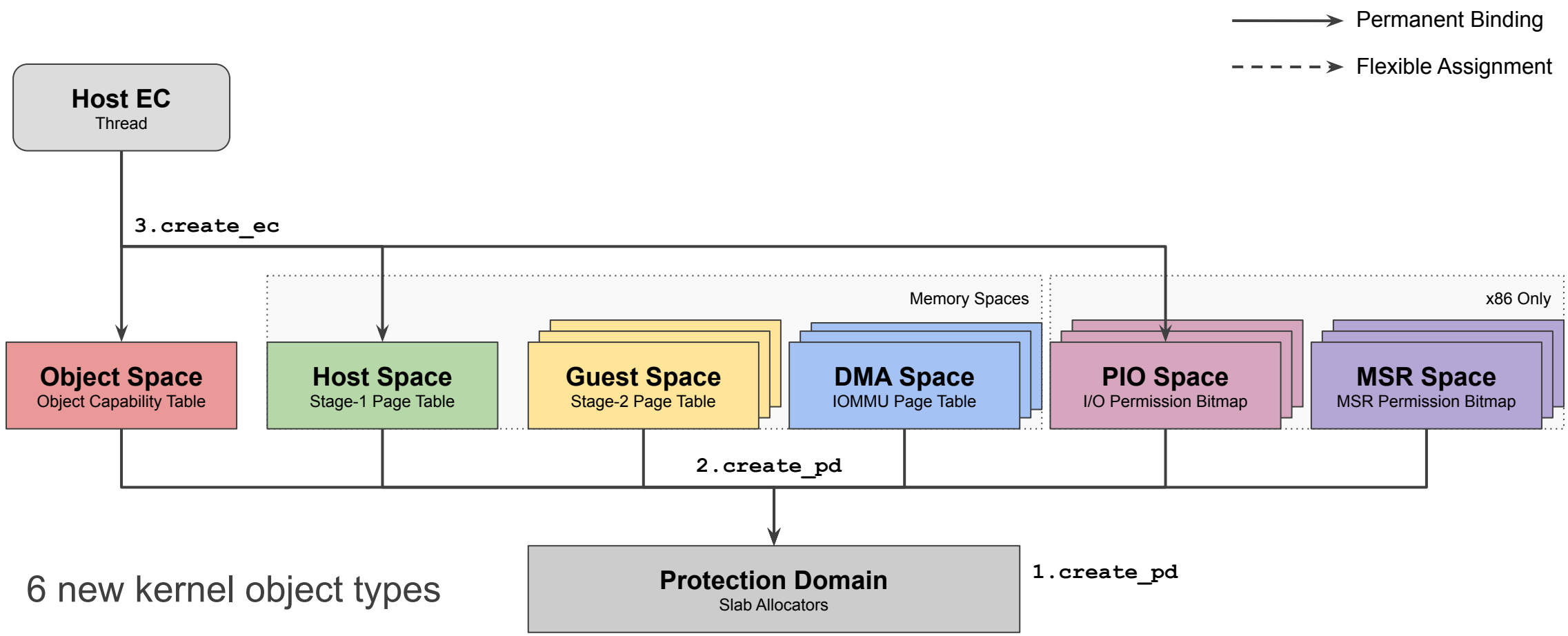
	x86_64	aarch64
P-State Support	EIST (older) and HWP (SKL+)	Not Yet
S1	Supported	N/A
S2/S3	Supported	Supported (PSCI 1.0+ Optional)
S4/S5	Supported	Supported (PSCI 0.2+ Mandatory)
Platform Reset	Supported	Supported (PSCI 0.2+ Mandatory)
S0ix (Low-Power Idle)	Not Yet	
Hypercall `ctrl_hw`	Returns `BAD_FTR` for unsupported functionality	
Hypercall `assign_dev`	Device assignment preserved over Suspend/Resume	
Hypercall `assign_int`	Interrupt configuration preserved over Suspend/Resume	

Past: Singleton Spaces built into Protection Domain



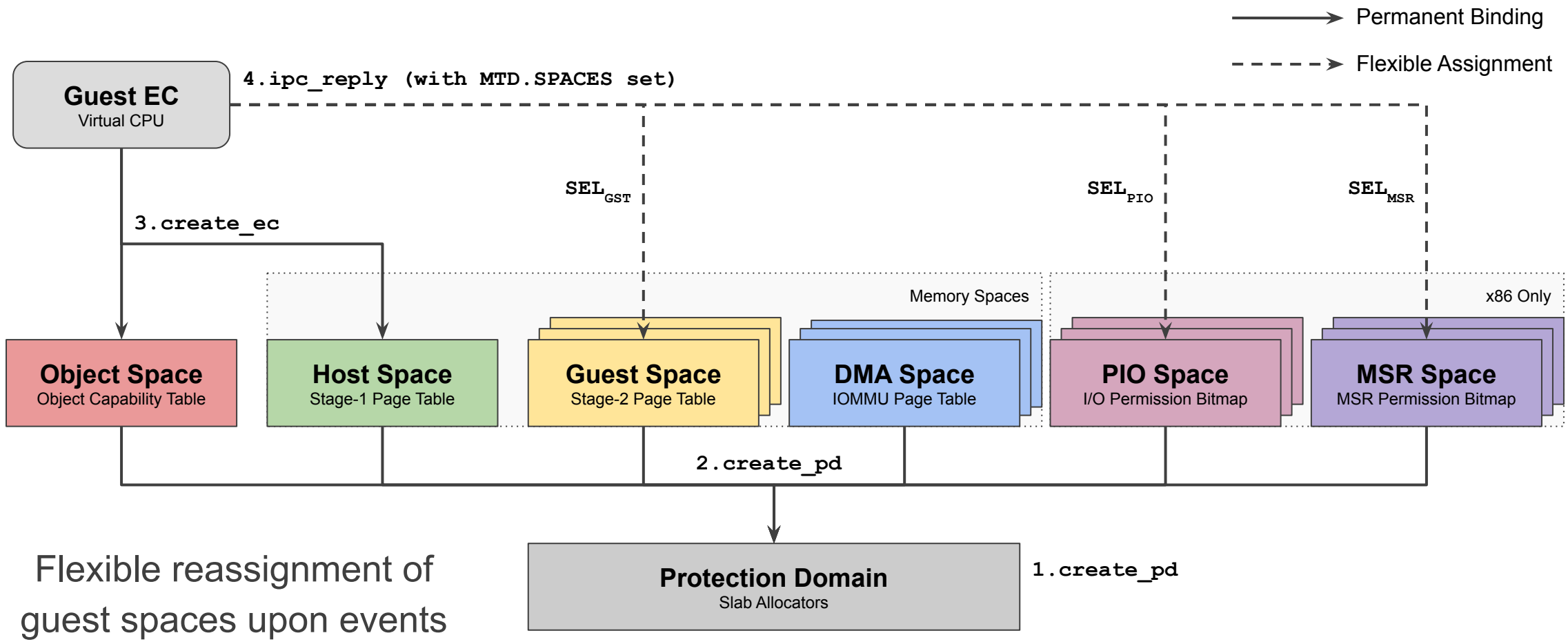
- Suboptimal for Nested Virtualization, SMMU Virtualization, Advanced VMI

Multiple Spaces separated from Protection Domain



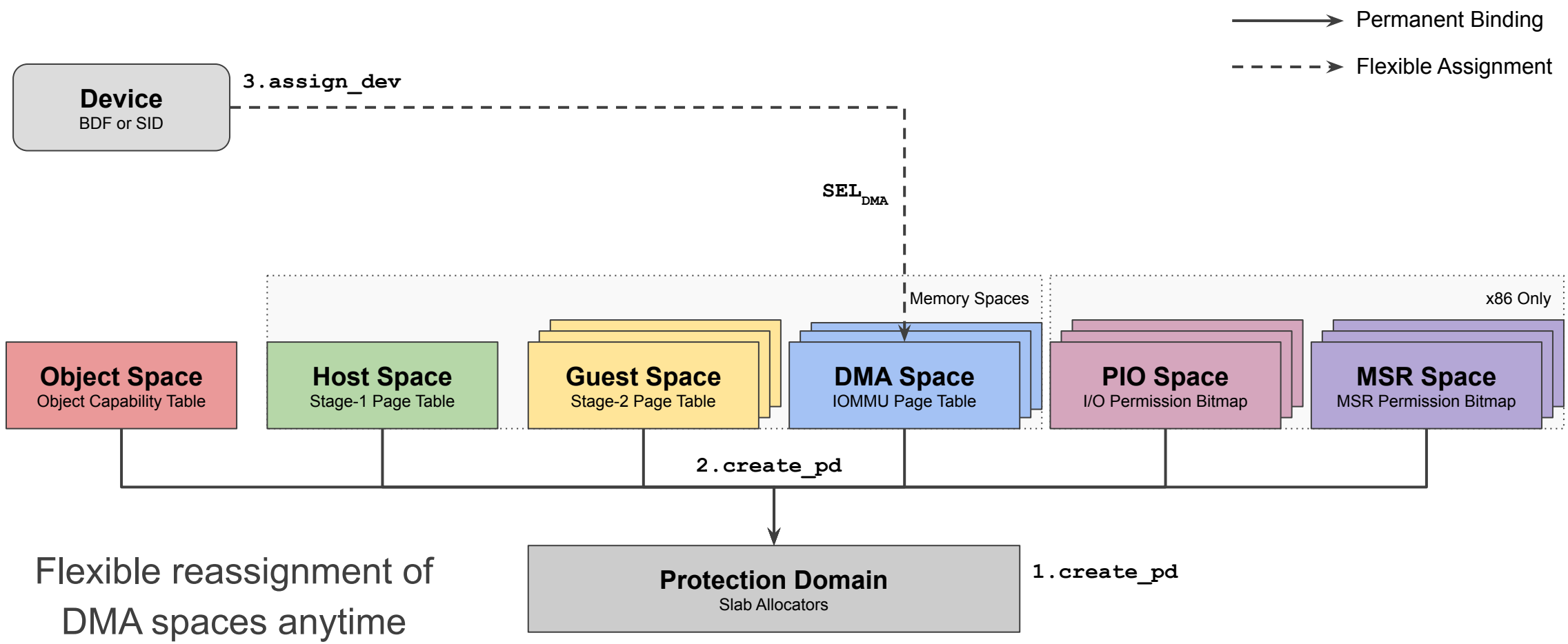
6 new kernel object types

Multiple Spaces separated from Protection Domain



Flexible reassignment of guest spaces upon events

Multiple Spaces separated from Protection Domain

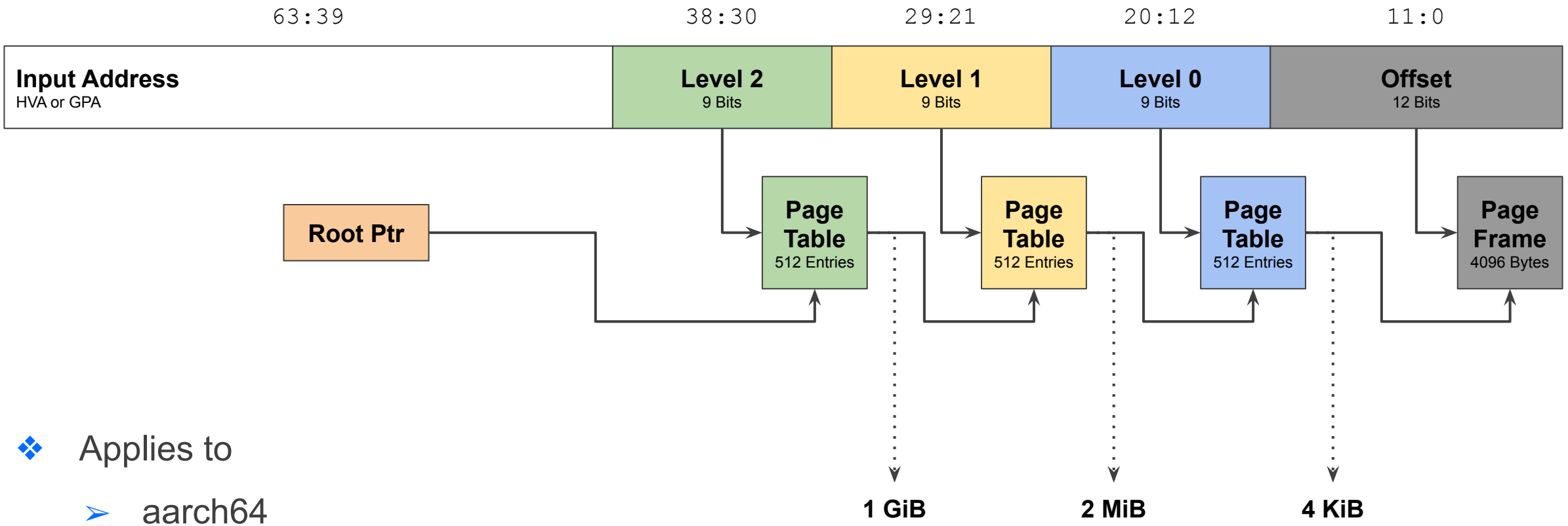


Flexible reassignment of DMA spaces anytime

Generic Page Tables

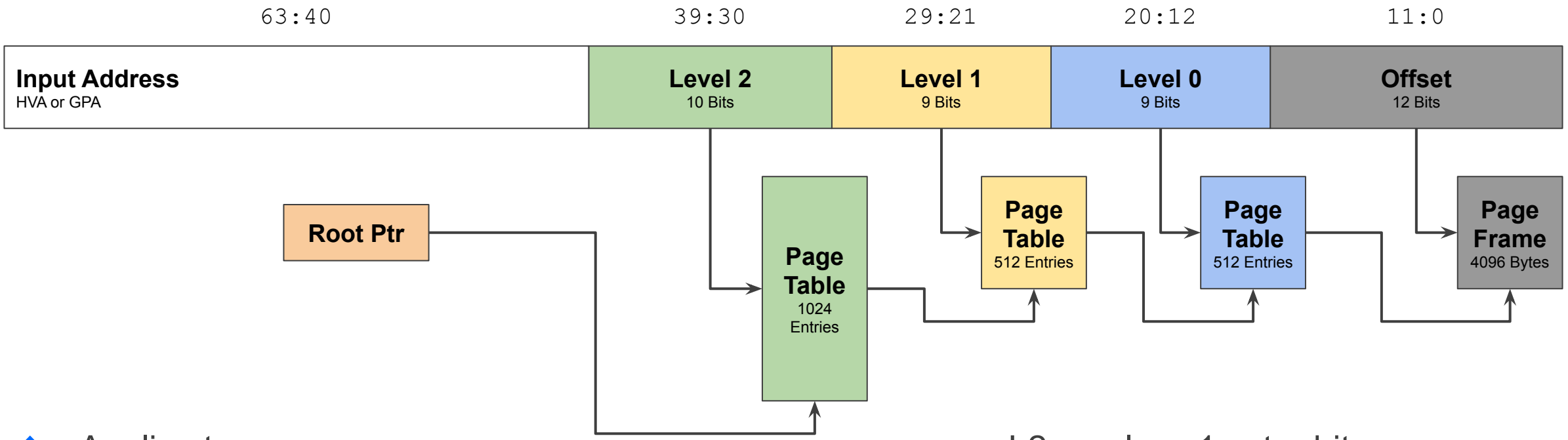
- NOVA manages 3 Page-Table Types for each Architecture
  - Host (Stage-1), Guest (Stage-2), DMA (IOMMU/SMMU)
  - These correspond to the 3 memory spaces
- Architecture-Independent Template Base Class
  - Lock-less page-table traversal, (de)allocation and PTE updates
  - Substitution of large pages with page tables and vice versa
- Type-Specific Subclasses (CRTP)
  - Encode access permissions and memory attributes (cacheability, shareability, key index)
  - Ensure non-coherent agents observe updates in the correct order

Page Tables: 39-bit Address Space (3-Level)



- Applies to
  - aarch64
  - x86_64

Page Tables: 40-bit Address Space (Concatenated)

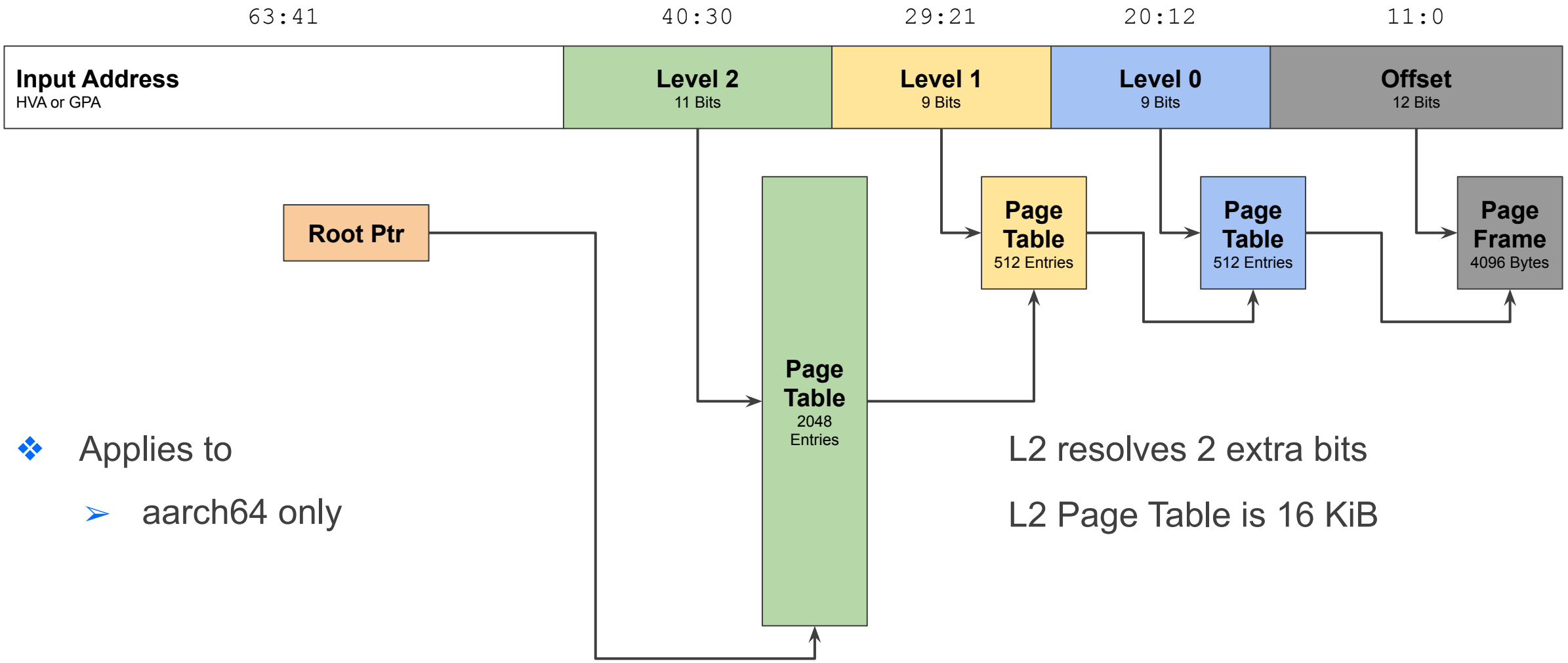


- Applies to
  - aarch64 only

L2 resolves 1 extra bit

L2 Page Table is 8 KiB

Page Tables: 41-bit Address Space (Concatenated)

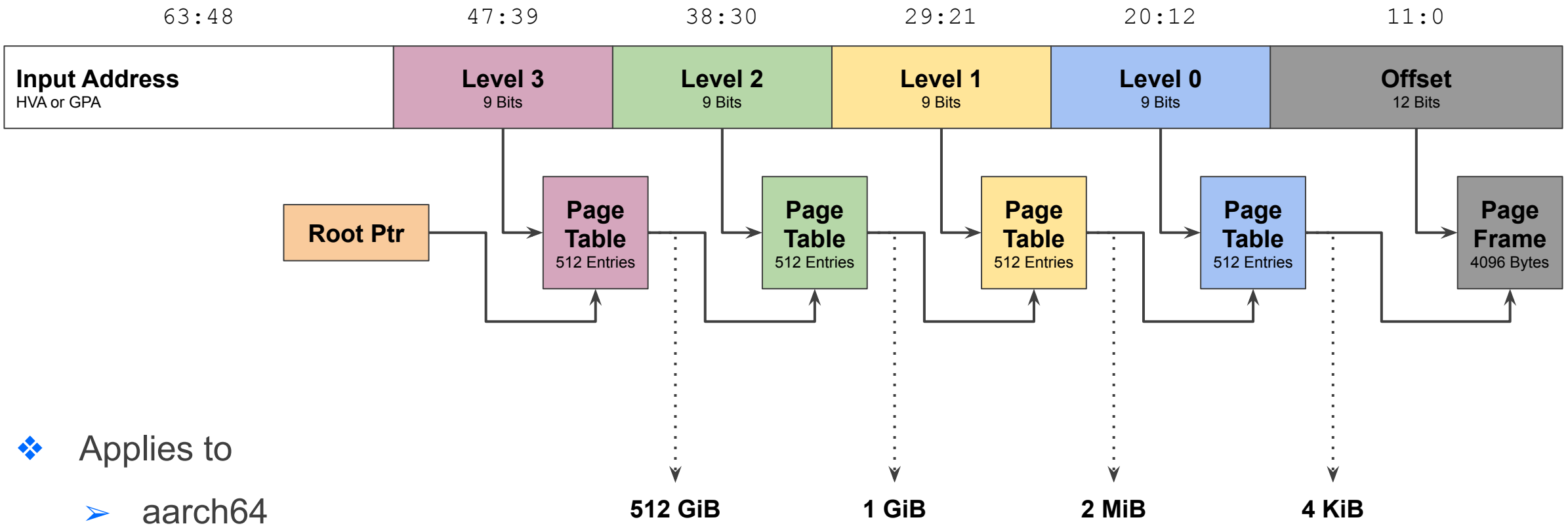


- Applies to
  - aarch64 only

L2 resolves 2 extra bits

L2 Page Table is 16 KiB

Page Tables: 48-bit Address Space (4-Level)



- Applies to
  - aarch64
  - x86_64

Page Tables: Level-Indexing Configuration

x86_64

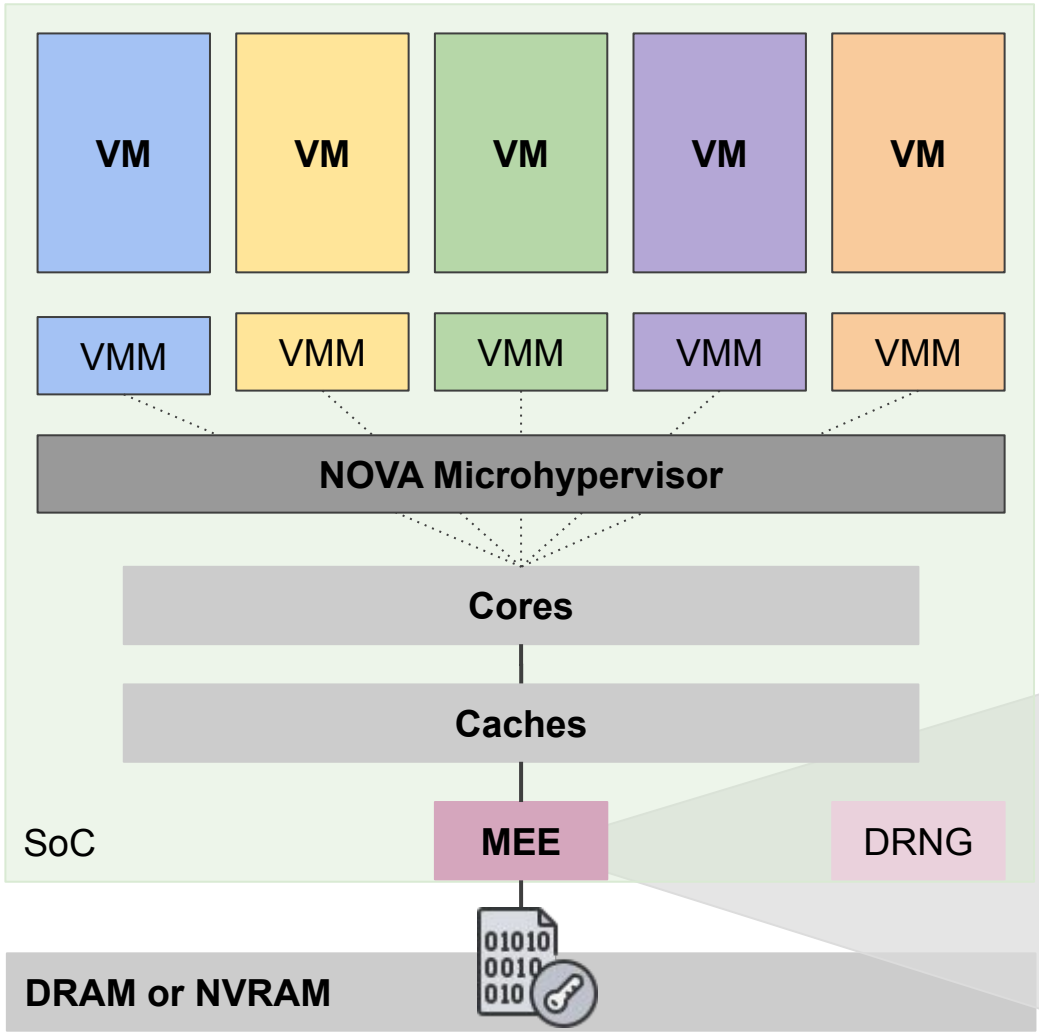
Bits	L4	L3	L2	L1	L0	OFF
57	9	9	9	9	9	12
48		9	9	9	9	
39			9	9	9	

aarch64

Bits	L3	L2	L1	L0	OFF
52	4+9	9	9	9	12
48	9	9	9	9	
44	5	9	9	9	
42		3+9	9	9	
40		1+9	9	9	
36		6	9	9	
32			2+9	9	

- NOVA derives from input bits
  - Uniform index bits per level
  - Non-uniform concatenation at top level
  - Number of required page-table levels

Multi-Key Total Memory Encryption (TME-MK)



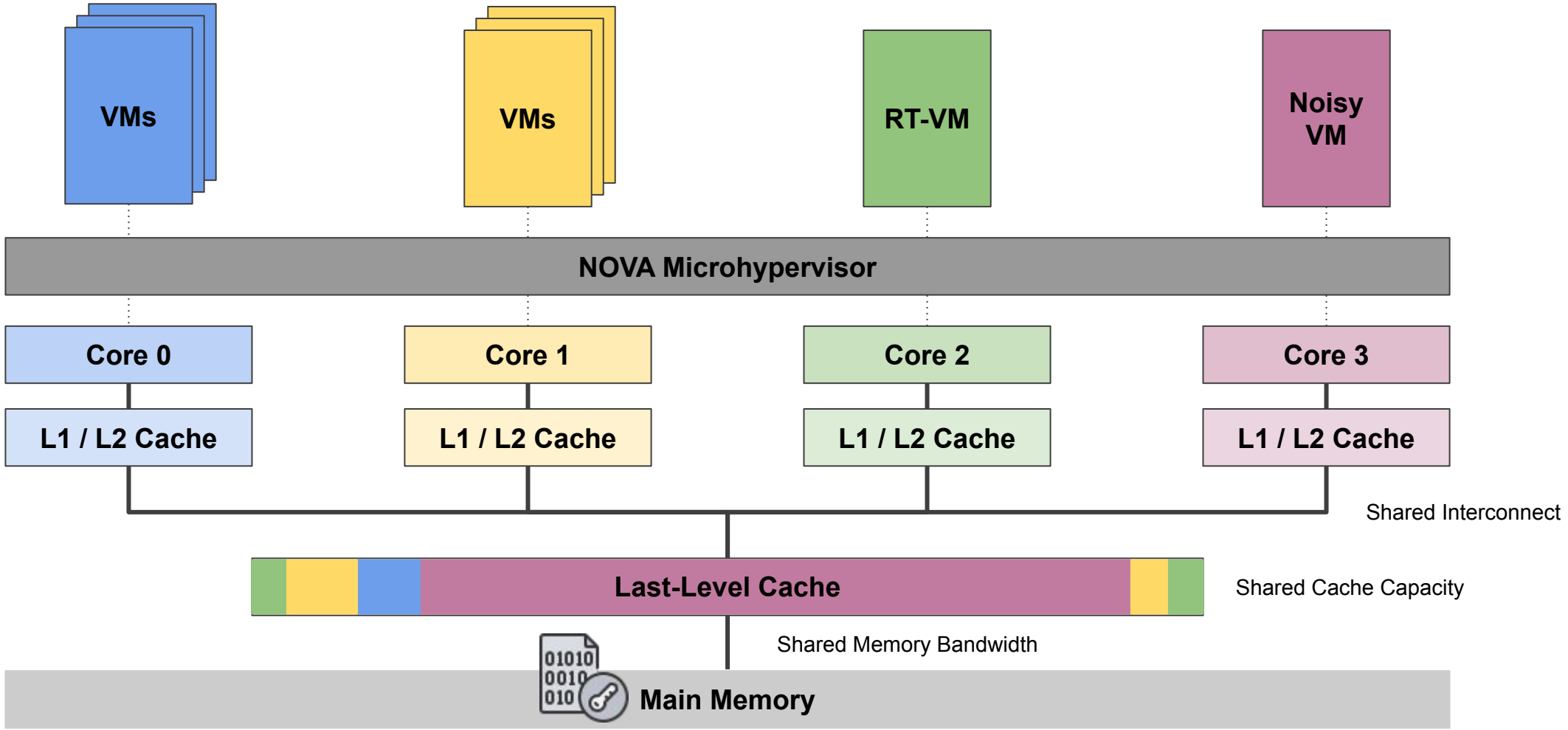
- KeyID per page encoded in PTE
- Stealing upper physical bits

Unused	KeyID	Physical Address	Attributes
Unused	KeyID	Physical Address	Attributes
Unused	KeyID	Physical Address	Attributes
Unused	KeyID	Physical Address	Attributes

Key0	FW TME Key
Key1	AES-XTS-128
Key2	AES-XTS-256
Key3	AES-XTS-256
Key4	AES-XTS-128
Key5	AES-XTS-128

- Key Programming
  - random/tenant
  - DRNG entropy

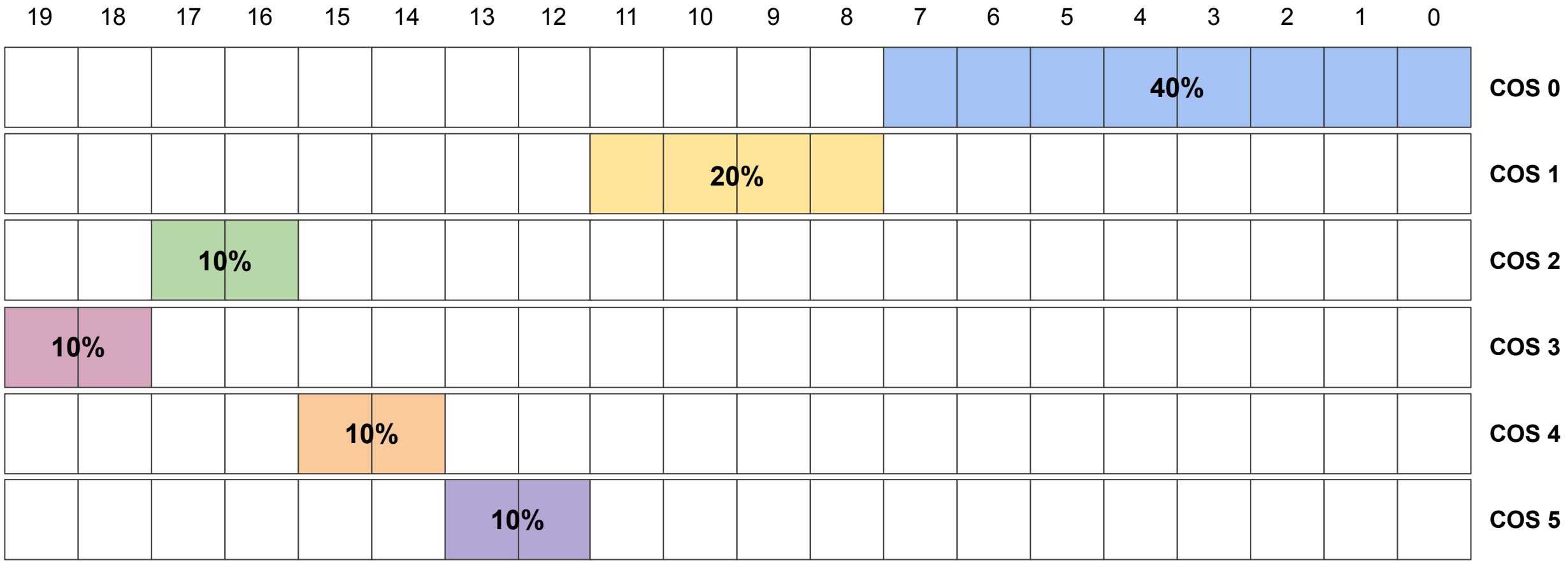
Protecting against “Noisy Neighbor” Domains



Cache Allocation Technology (CAT)

- Limited Number of Classes of Service (COS)
  - Hypervisor assigns COS to all cache-sharing entities (VMs/Processes/Threads)
- Capacity Bitmask per COS
  - Specifies into which ways of the cache that COS can allocate
  - Can be defined per CPU and separately for L2/L3 caches
  - Bitmasks must be contiguous, but can overlap
  - Active COS per CPU set in IA32_PQR_ASSOC MSR
- Improves Predictability (WCET) and Cache Side-Channel Resistance

Cache Allocation Technology (CAT): Example

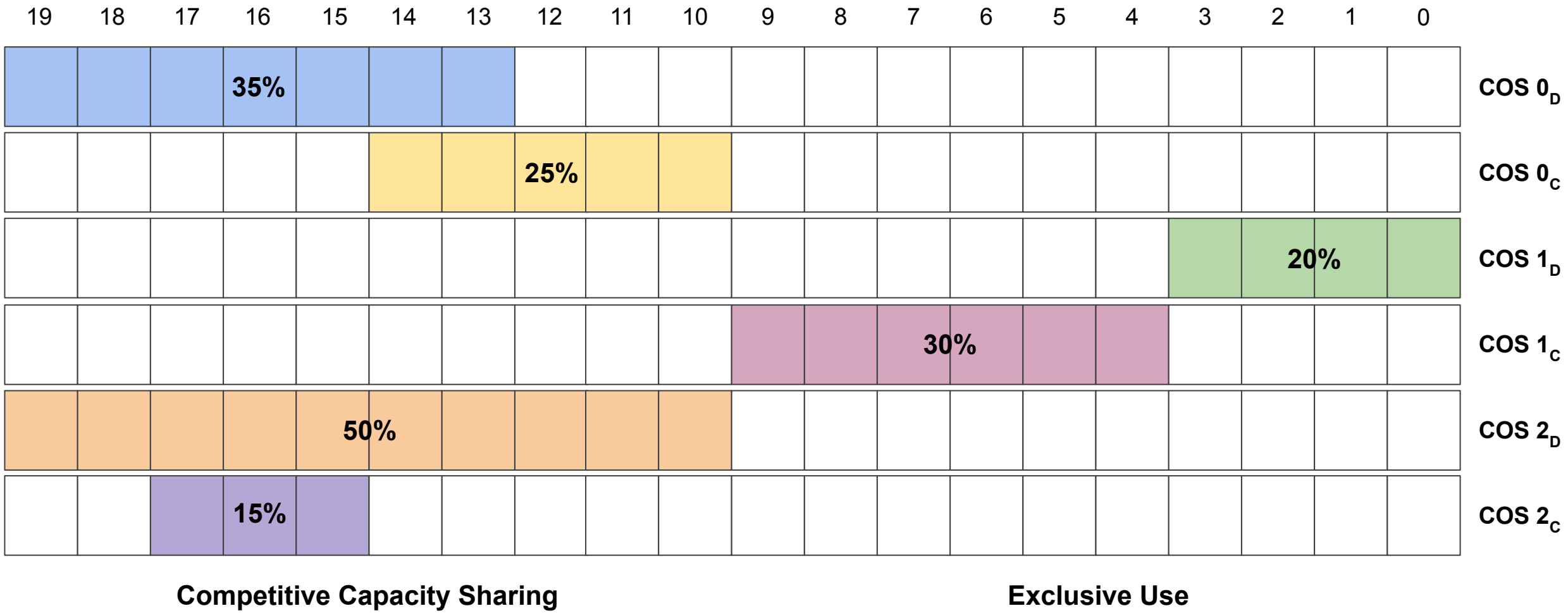


Full Isolation / No Capacity Sharing

Code and Data Prioritization (CDP)

- Capacity bitmasks redefined as pairs
  - Even bitmask number ⇒ data
  - Odd bitmask number ⇒ code
- More fine-grain control over how the cache is being used
- Total number of classes of service (COS) cut in half
  - 6 COS for CAT ⇒ 3 COS for CDP
- NOVA API enforces selection of CAT vs. CDP mode for L2/L3 cache
  - QoS configuration required before capacity bitmasks can be programmed
  - Mode cannot be changed subsequently

Code and Data Prioritization (CDP): Example



Class of Service (COS)

- Option 1: Assign COS to Protection Domain (PD)
  - For PDs that span multiple cores, COS settings must be consistent across cores
- Option 2: Assign COS to Execution Context (EC)
  - Expensive to set new COS during each context switch (WRMSR)
- In NOVA: COS is a Scheduling Context (SC) attribute
  - Only need to context-switch COS during scheduler invocations
  - Extends the call-chain time/priority donation model with COS donation
  - Servers use the cache capacity allocated to their respective clients
  - Additional benefit: COS settings can be configured differently on each core

Code Integrity Protection

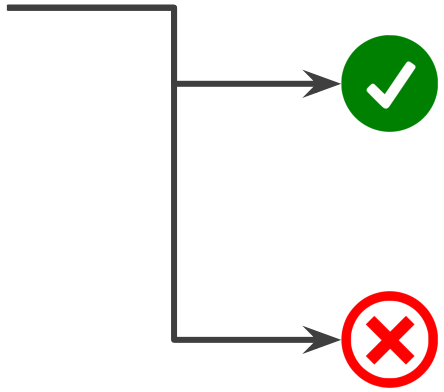
- Long history of paging features raising the bar for code injection attacks
  - Non-writable code / Non-executable stack (W^X)
  - Supervisor Mode Execution Prevention (SMEP)
  - Supervisor Mode Access Prevention (SMAP)
  - Mode-Based Execution Control (MBEC) for Stage-2 with XU/XS permission bits
- Code snippets (gadgets) in existing code could still be chained together
  - Control-Flow Hijacking: COP / JOP / ROP attacks
  - Instruction length is fixed on ARM but varies on x86

Control-Flow Enforcement Technology (CET)

- Protects integrity of control-flow graph using x86 hardware features
- Indirect Branch Tracking (Forward-Edge) `make ARCH=x86_64 CFP=branch`
  - Used with indirect JMP / CALL instructions
  - Valid branch targets must be marked with ENDBR instruction
  - Requires compiler support (available since gcc-8)
- Shadow Stacks (Backward-Edge) `make ARCH=x86_64 CFP=return`
  - Used with CALL / RET instructions
  - Second stack used exclusively for return addresses
  - Can only be written by control-transfer and shadow-stack-management instructions

CET Indirect Branch Tracking

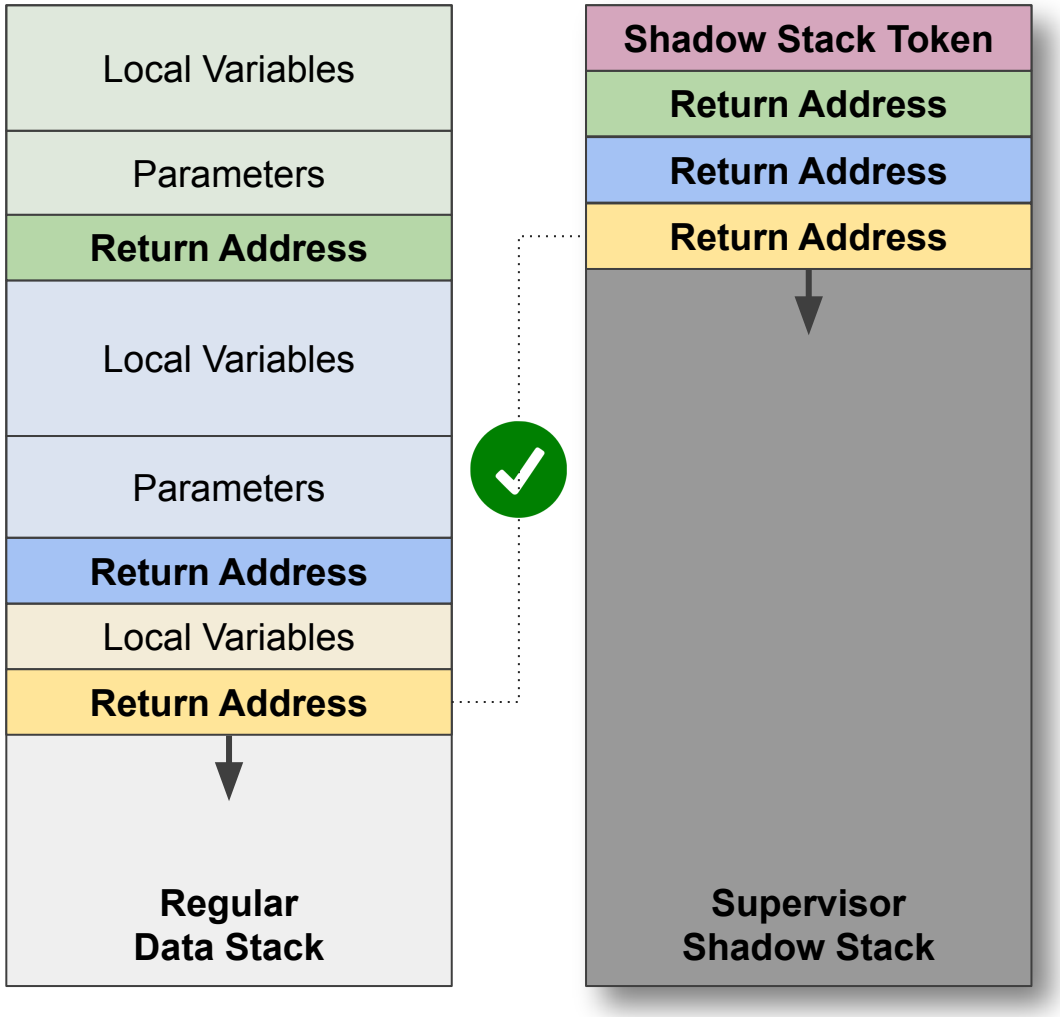
```
call *0x30(%rbx)
```



```
ffffffff80003a60 <Buddy::free(void*)>:
ffffffff80003a60:       endbr64
ffffffff80003a64:       test   %rdi,%rdi
ffffffff80003a67:       je     ffffffff80003a84
ffffffff80003a69:       sub    0xf1e8(%rip),%rdi
ffffffff80003a70:       shr    $0xc,%rdi
ffffffff80003a74:       imul   $0x18,%rdi,%rdi
ffffffff80003a78:       add    0xf1d1(%rip),%rdi
ffffffff80003a7f:       jmp    ffffffff80003962
ffffffff80003a84:       ret
```

- CALL / JMP Instruction
  - Next instruction must be ENDBR
  - #CP exception otherwise

CET Supervisor Shadow Stacks



- CALL instruction
  - Pushes return address onto both stacks
- RET instruction
  - Pops return address from both stacks
  - #CP exception if addresses not equal
- Shadow Stack Management
  - Busy bit in token prevents multi-activation
  - NOVA must unwind supervisor shadow stack during context switches

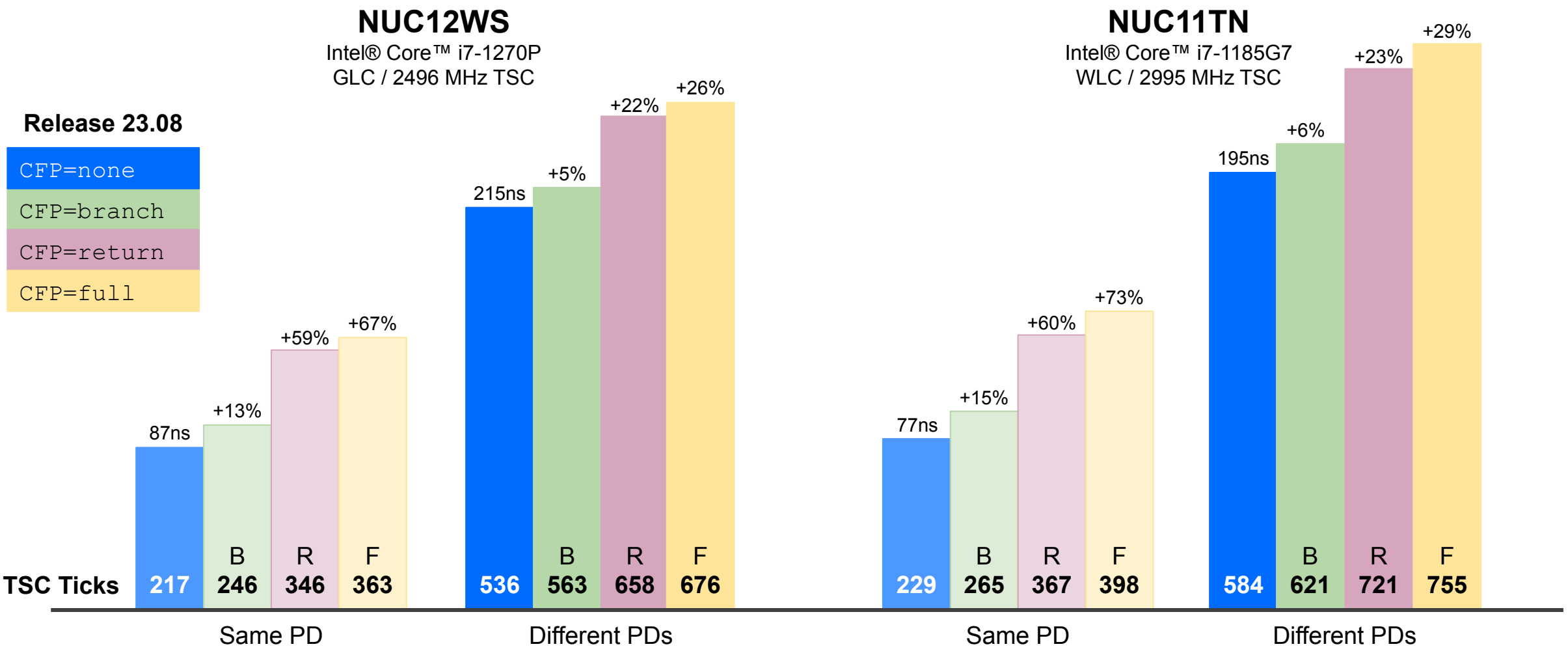
Managing the ISA Evolution

- Modern CPU features require new instructions that fault on older CPUs
  - Example: Shadow Stack Management Instructions
- Newer CPUs provide instructions that do more/optimized work
  - Example: XSAVE / XSAVEC / XSAVEOPT / XSAVES (Extended State Management)
- Code using newer instructions will fault on older CPUs
- Possible options
  - Either compile different binaries with/without those instructions (compile-time)
  - Or select the most suitable instruction (each time) at run time (run-time)
  - Or install the most suitable instruction (once) at boot time (boot-time)

Boot-Time Code Patching

- Install the optimal instruction (sequence) once at boot time
  - `asm ("xsaves %0" : "=m" (fpu_state) …);` ⇒ `asm ("xsave %0" : "=m" (fpu_state) …);`
  - `asm ("setssbsy");` ⇒ `asm ("nop");`
- Early feature checking to determine which patches to apply
  - Patches only low-level assembler instructions (newer CPUs ⇒ less patching)
  - No need for patching any high-level C++ code ⇒ compiles to innocuous instructions
  - No need for repeated run-time feature tests ⇒ no extra overhead
- One generic binary that works across a wide range of hardware platforms
  - Automatically adjusts to supported platform features

Round-Trip IPC Performance (with CET)



Questions and Discussion

The NOVA microhypervisor is licensed under GPLv2

Releases: https://github.com/udosteinberg/NOVA/tags

More Information: bedrocksystems.com and hypervisor.org