Advanced Compilers and Architectures

ARM MMU Overview

Mattias Holm, M.Sc.
holm@liacs.nl

Universiteit Leiden

Leiden University. The university to discover.

Outline

- Memory Management Overview
- ARM Overview
- ARM MMU Specifics

VM and MMU

Memory Management

- Needed to protect applications from each other.
- Necessary if an application requests more memory than is physically available.

Memory Management

- Two Major Technologies:
  - Segmentation
  - Paging

Segmentation

- Processor tracks start and length attributes for memory segments using registers.
- Each segment has access attributes:
  - Read, write, execute et.c.
- Violation of attributes results in a segment violation or segfault (UNIX SIGSEGV).

Paging

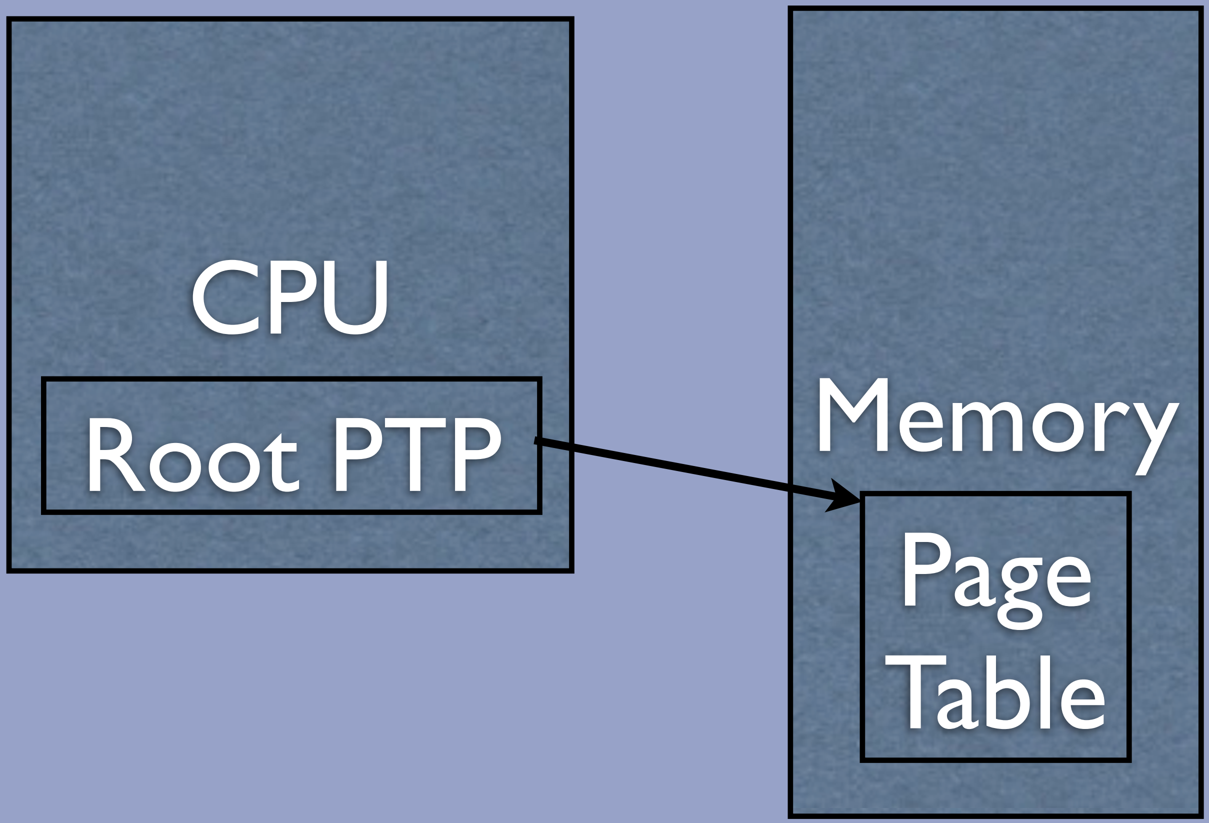
- Introduced to allow memory to be swapped out to disk.
- Memory divided into pages of fixed size (usually 4 KiB).
- Memory pages specified using page tables.
- Pages have access attributes like segments.
- Has mostly replaced segmentation.

Paging

- Page Table Pointers (PTPs) identify page tables.
- Page Table Entries (PTEs) map virtual to physical address and track page attributes.
- Translation Lookaside Buffer (TLB) caches PTEs for quick access.
- If an entry is not in the TLB, memory system will do a page table walk.

Paging

- Table Walk
  - Processor performs a load or store to an address that is not in the TLB (assume address is 0x10201234).
  - Processor uses the page table pointer (stored in a special register) to find the page table.



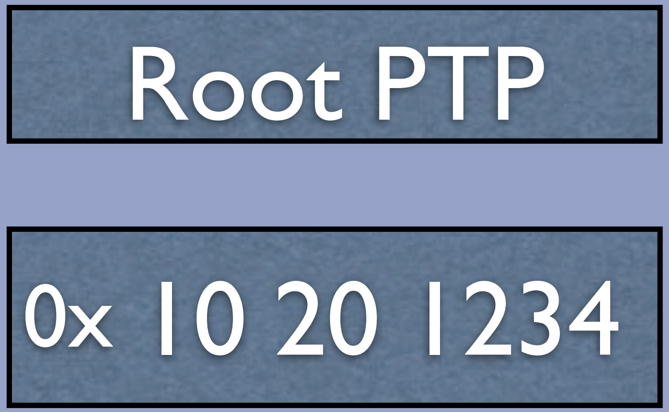
Paging

Root PTP

0x 10 20 1234

Paging

- Table Walk



Paging

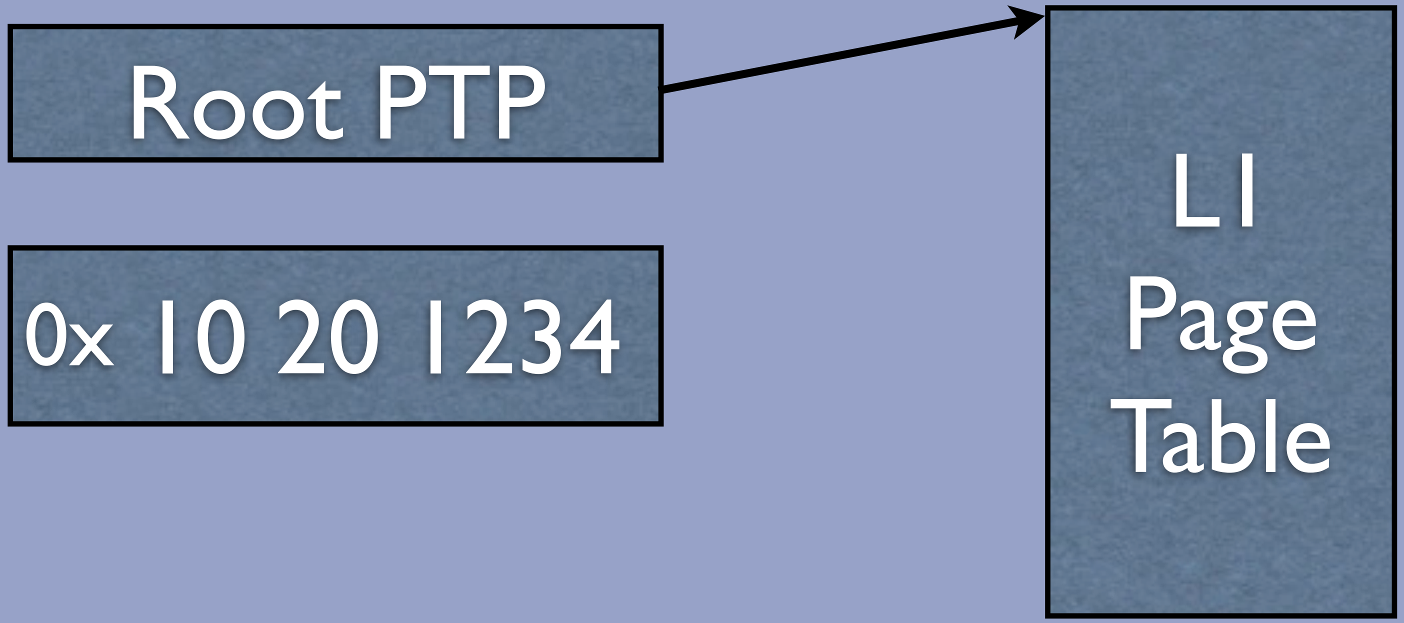
- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.

Root PTP

0x 10 20 1234

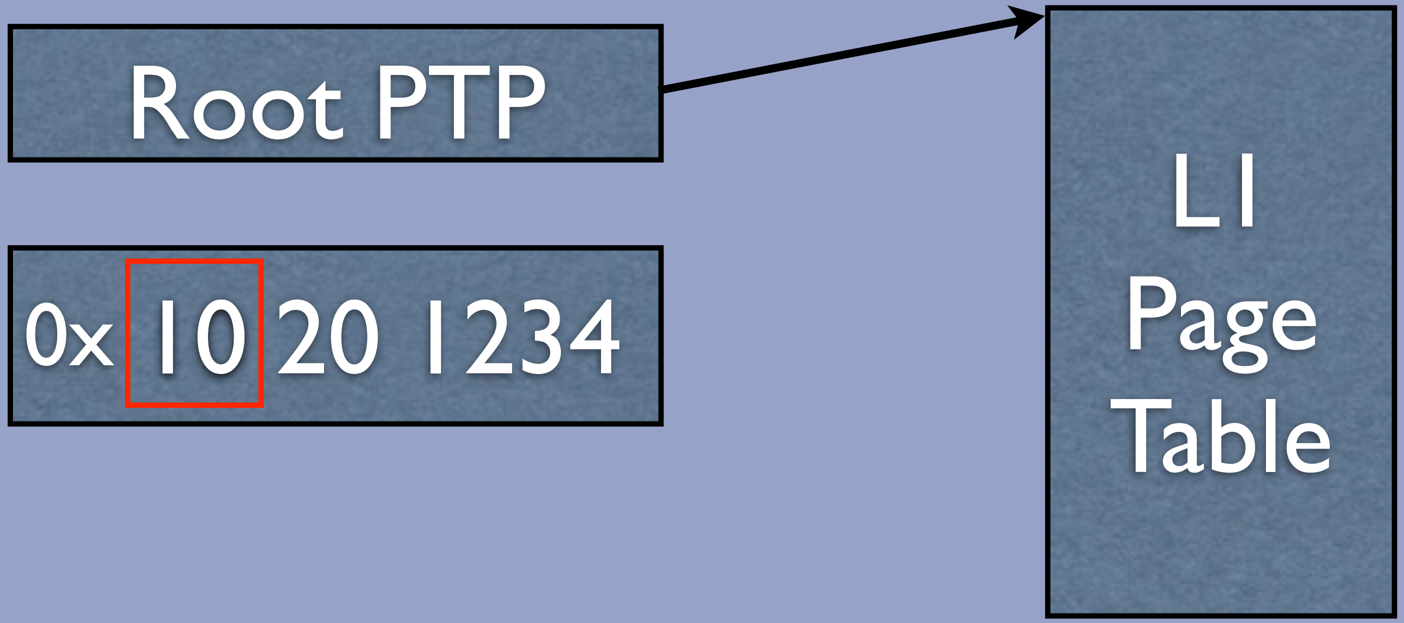
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.



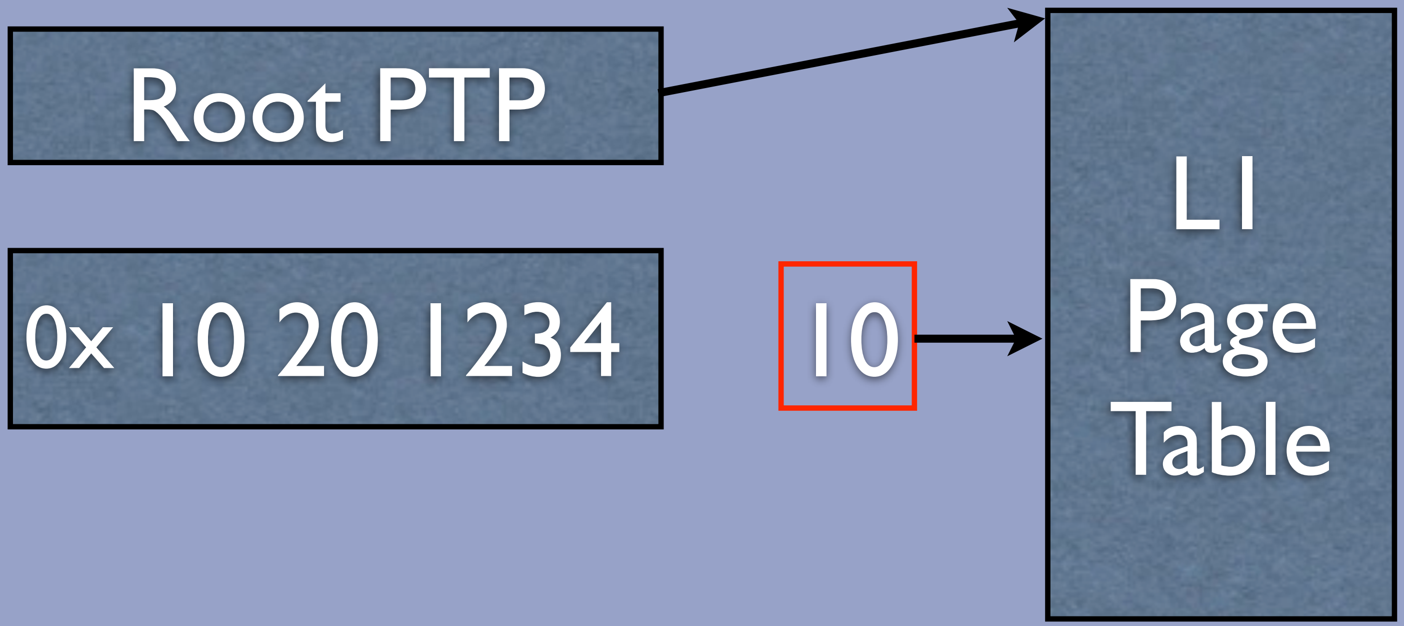
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.



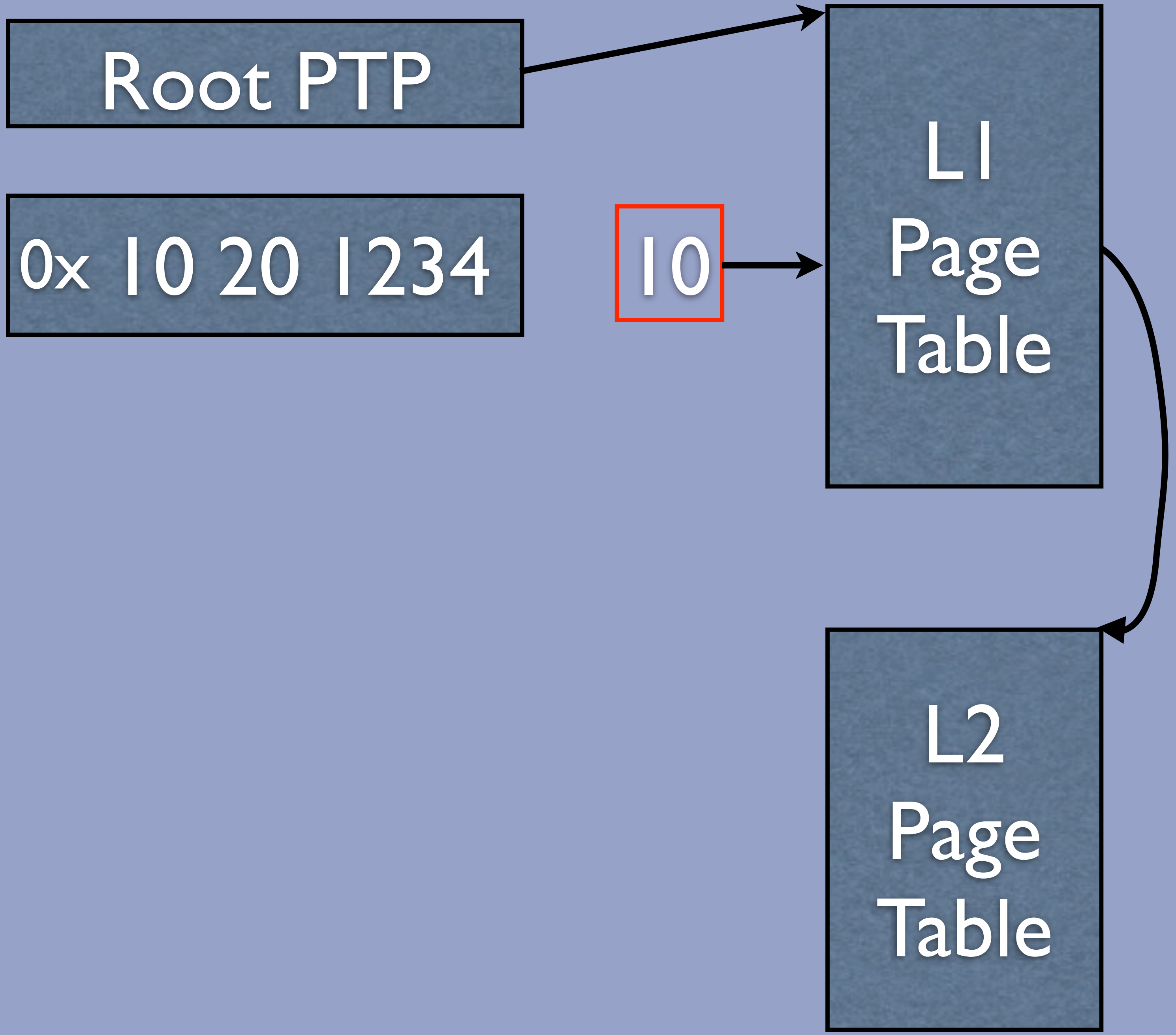
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.



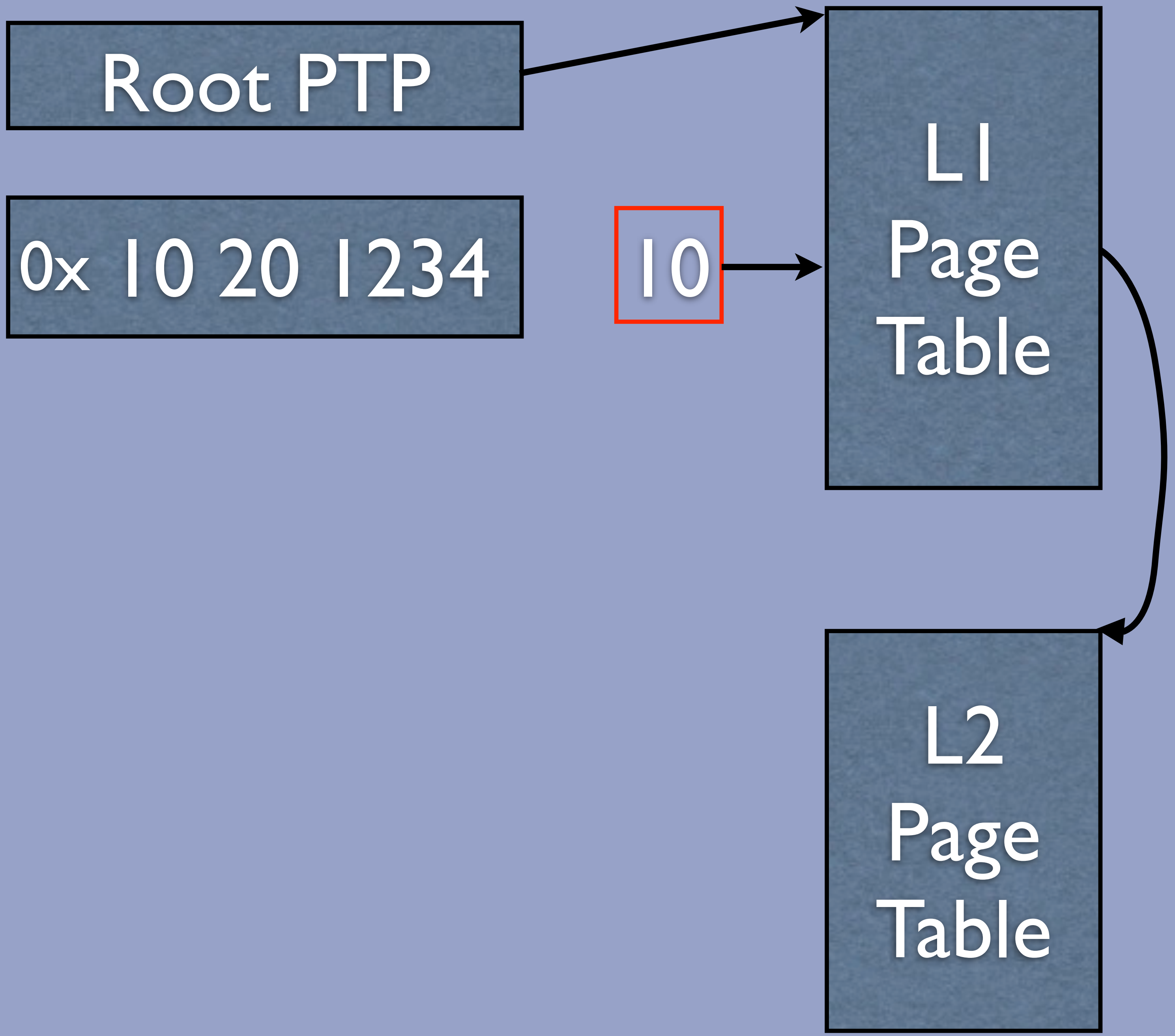
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.



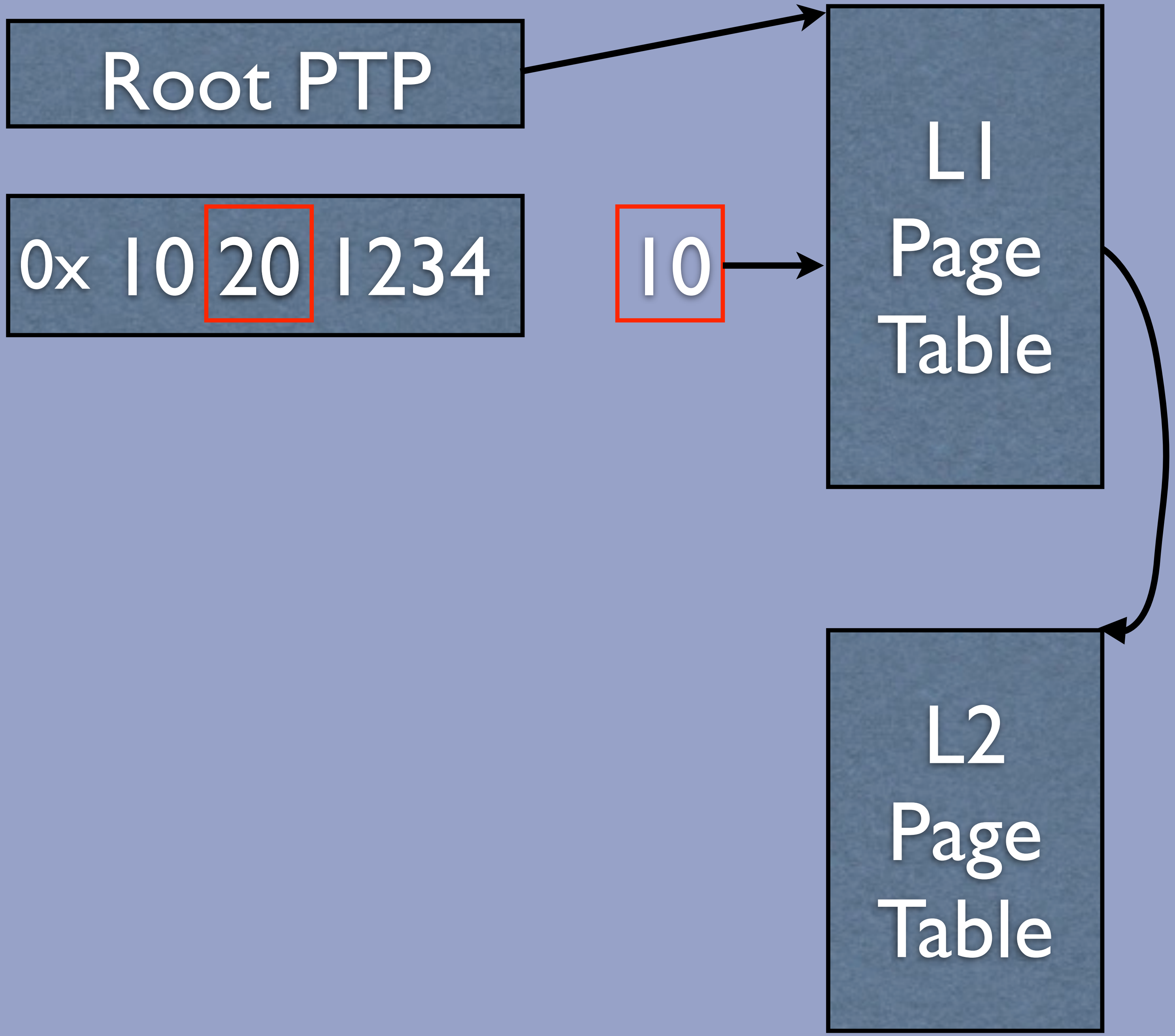
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.



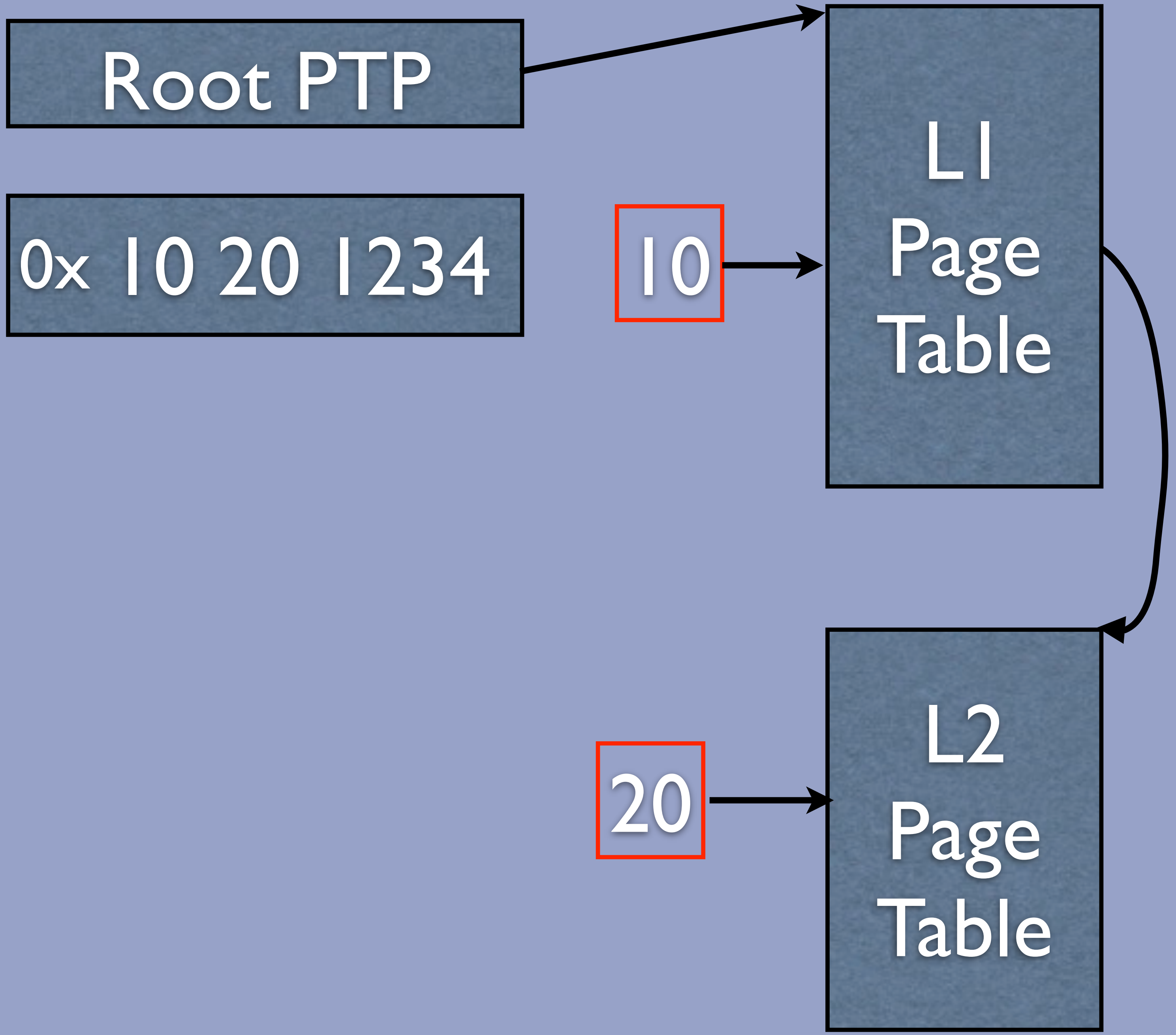
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.



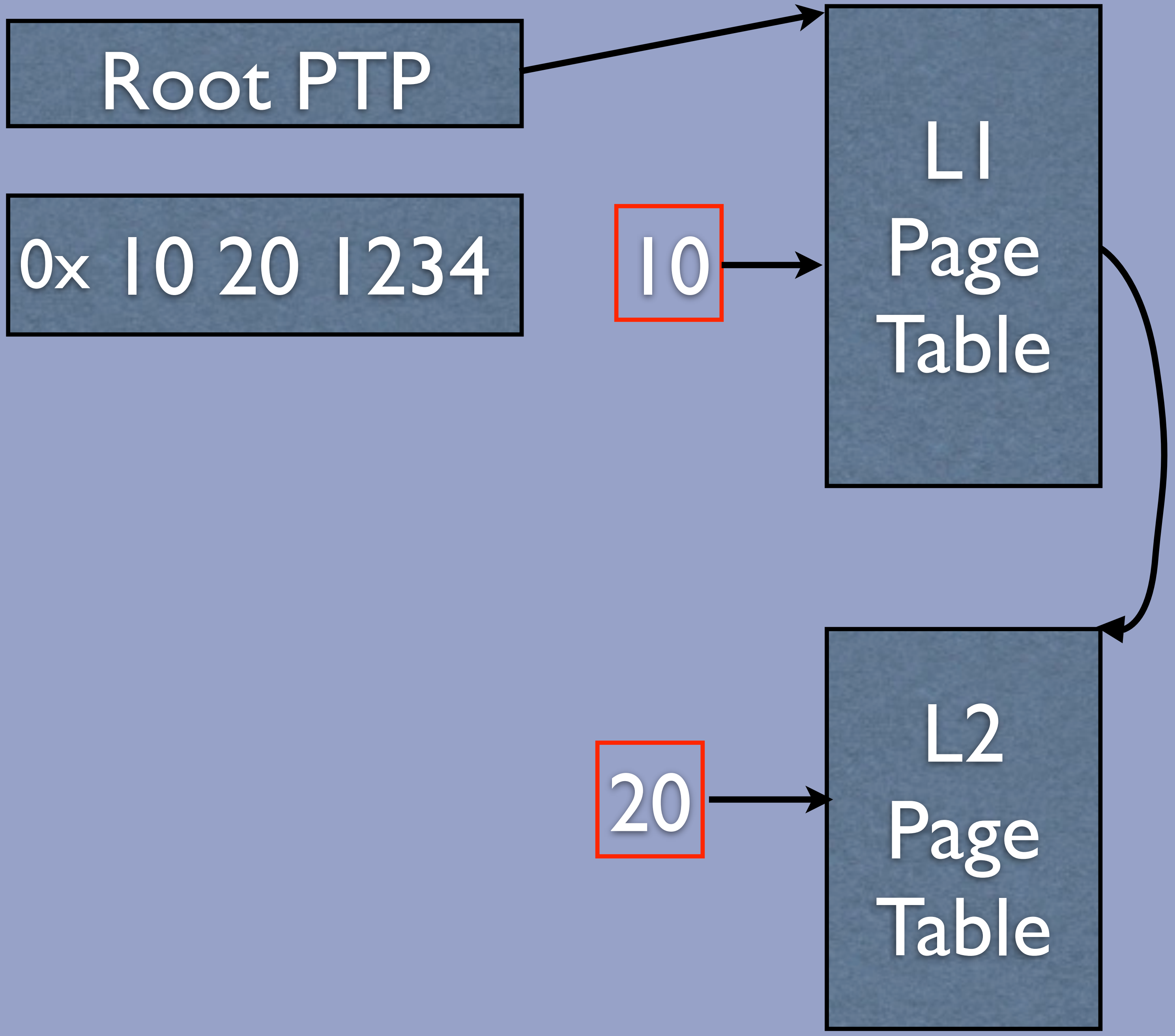
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.



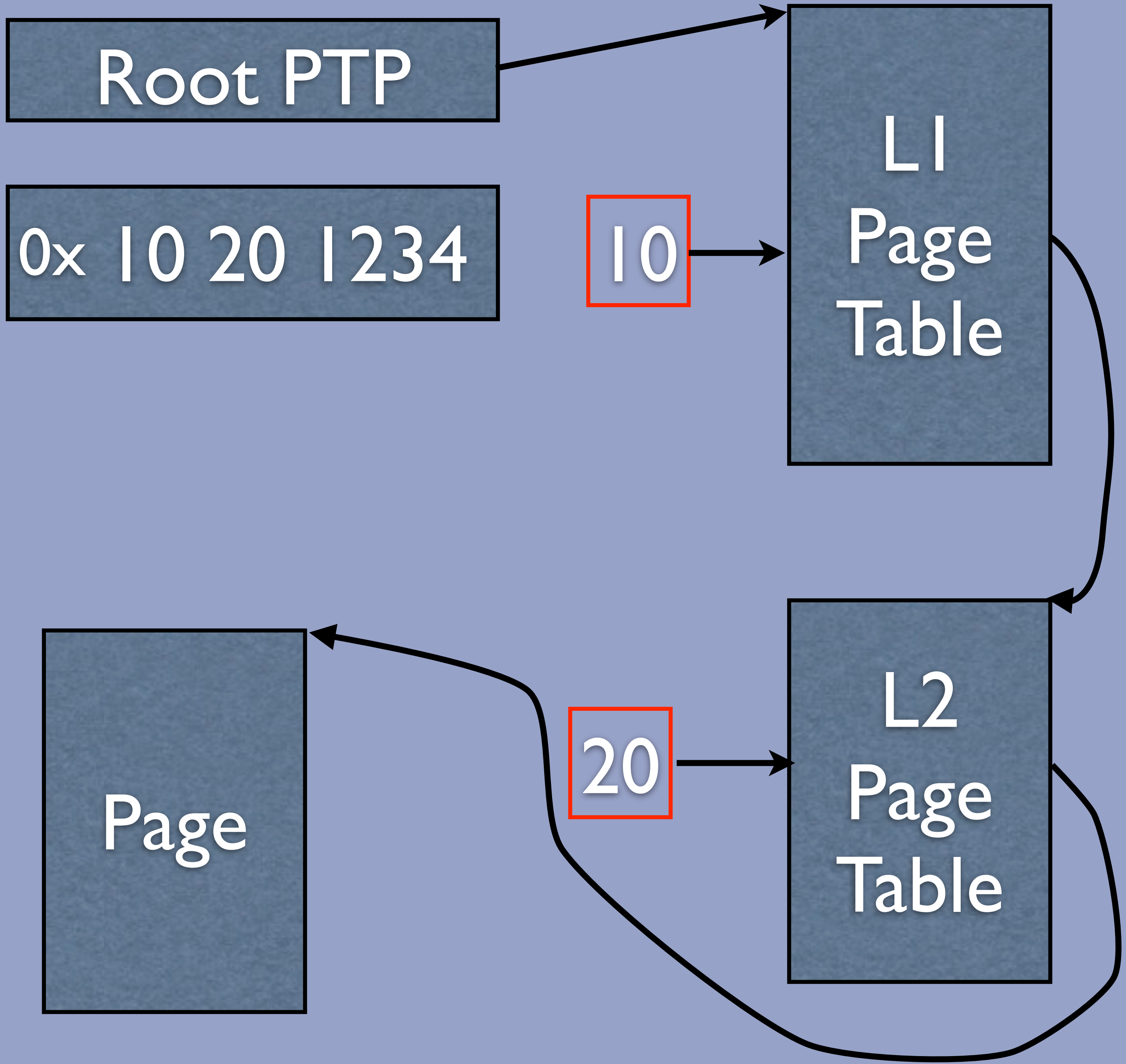
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.



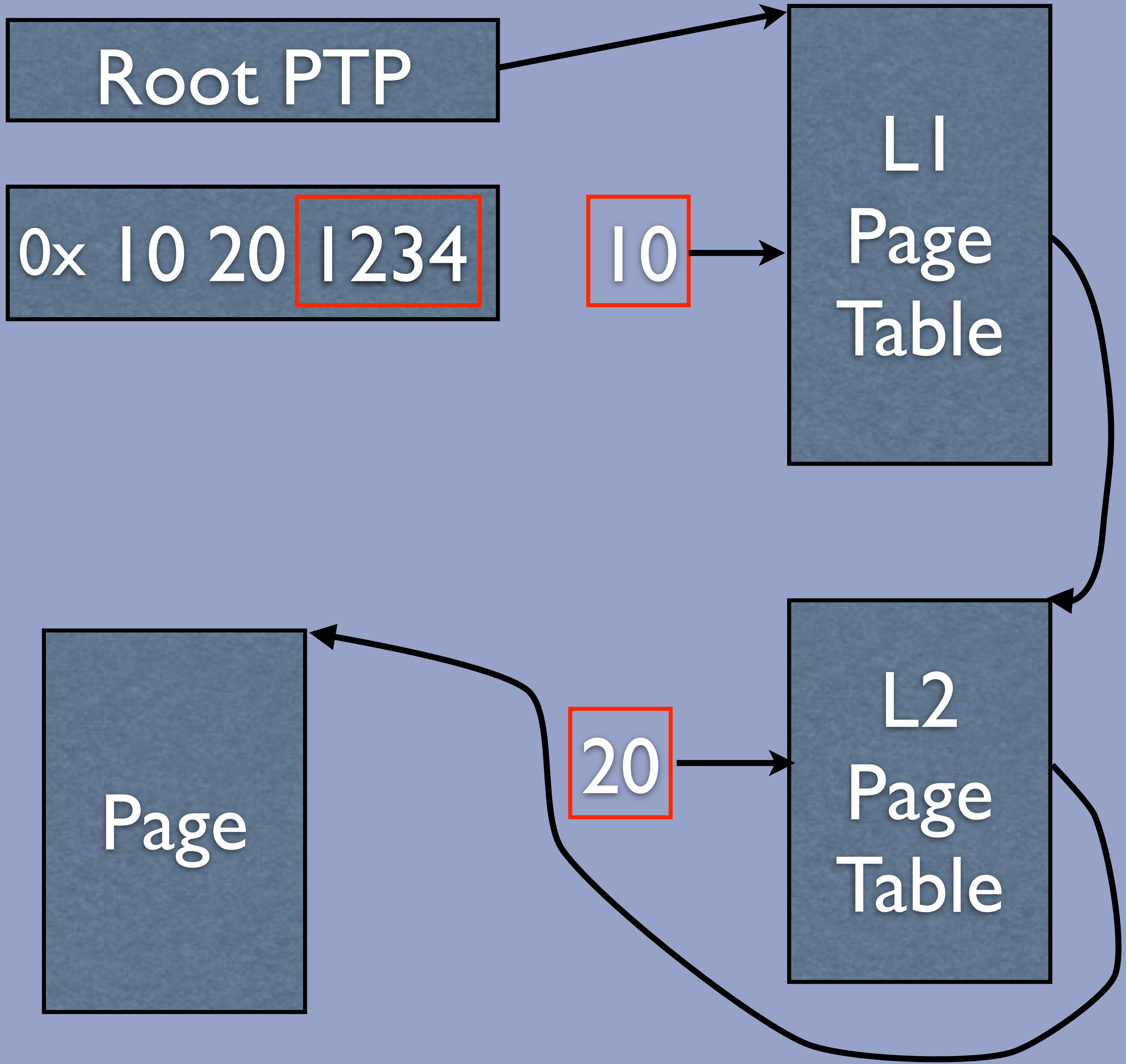
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.



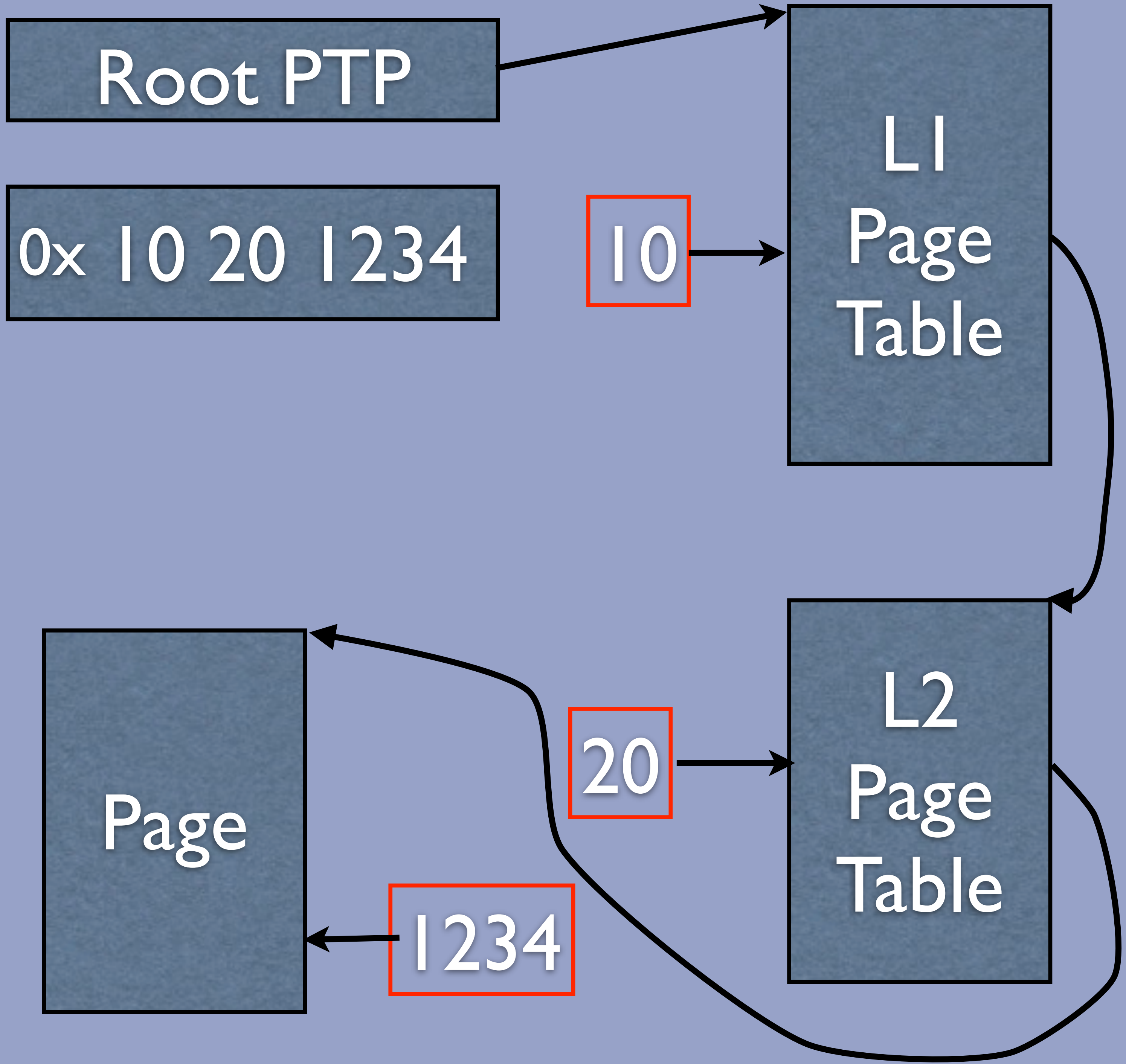
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.



Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.



Page Allocation

- How do we allocate virtual memory?
- How do we allocate physical memory?

Allocating virtual addresses

- Each process has its own VM table.
- Process associated with a sorted linked list that track the allocated VM blocks.
- Can place allocated page list in a balancing tree for faster search.

Allocating physical addresses

- Kernel needs to track which physical pages has been allocated.
- Needs a list of free pages.
- Linux has lists of 2^n sized blocks that are free. If a block of power *a* is requested, but does not exist, split block of power *a+1*.

Allocating space for the VM structures

- Cannot use the kernel's generic VM code to allocate space for VM structures (*turtles all the way down*).
- Break the chain of recursion by special casing the VM structure allocation.
- Steal one physical page and use this for storing VM structures, and to describe itself.

ARM

ARM Overview

- ARM Developed by Acorn Computers Ltd in the UK, ARM 1 released 1985.
- Acorn + Apple worked on new chip design for the Newton PDA, ARM 6 released in 1994.
- Processors are licensed, not manufactured.

ARM Overview

- Used in around 98% of all mobile phones.
- Has around 90% of the embedded processor market.
- Very power efficient.

ARM Overview

Family	Architecture	Core	Chips
ARM1	ARMv1	ARM1	ARM1
ARM6	ARMv3	ARM60, ARM600, ARM610	ARM60, ARM600, ARM610
Cortex-A	ARMv7-A	Cortex-A8, Cortex-A9	OMAP3xxx, Apple A4

ARM Characteristics

- 32 bit architecture
- Load-store architecture (RISC)
- Normally little-endian, but may vary between processors
- 16 GPRs (r15 = pc)
- Multiple ISAs (ARM, Thumb, Jazelle)

Modes and Registers

- ARM processor banks registers, depending on mode.
- USR: User applications
- SYS: System mode, with access to USR registers.
- SVC, ABT, UND, IRQ: banks r13-r14
- FIQ: banks r8-r14

USR	SYS	SVC	ABT	UND	IRQ	FIQ
r0						
r1						
r2						
r3						
r4						
r5						
r6						
r7						
r8						r8_fiq
r9						r9_fiq
r10						r10_fiq
r11						r11_fiq
r12						r12_fiq
r13 (sp)		r13_svc	r13_abt	r13_und	r13_irq	r13_fiq
r14 (lr)		r14_svc	r14_abt	r14_und	r14_irq	r14_fiq
r15 (pc)						
cpsr						
		spsr_svc	spsr_abt	spsr_und	spsr_irq	spsr_fiq

ARMv7

- ARMv7 comes in 3 variants:
  - ARMv7-A with MMU (paging).
  - ARMv7-R for hard realtime applications with MPU (segmentation).
  - ARMv7-M micro-controller version, no memory protection.

ARMv7-A/R

- VMSA - Virtual Memory System Architecture
- PMSA - Protected Memory System Architecture
- Our kernel runs on ARMv7-A (Cortex-A8)

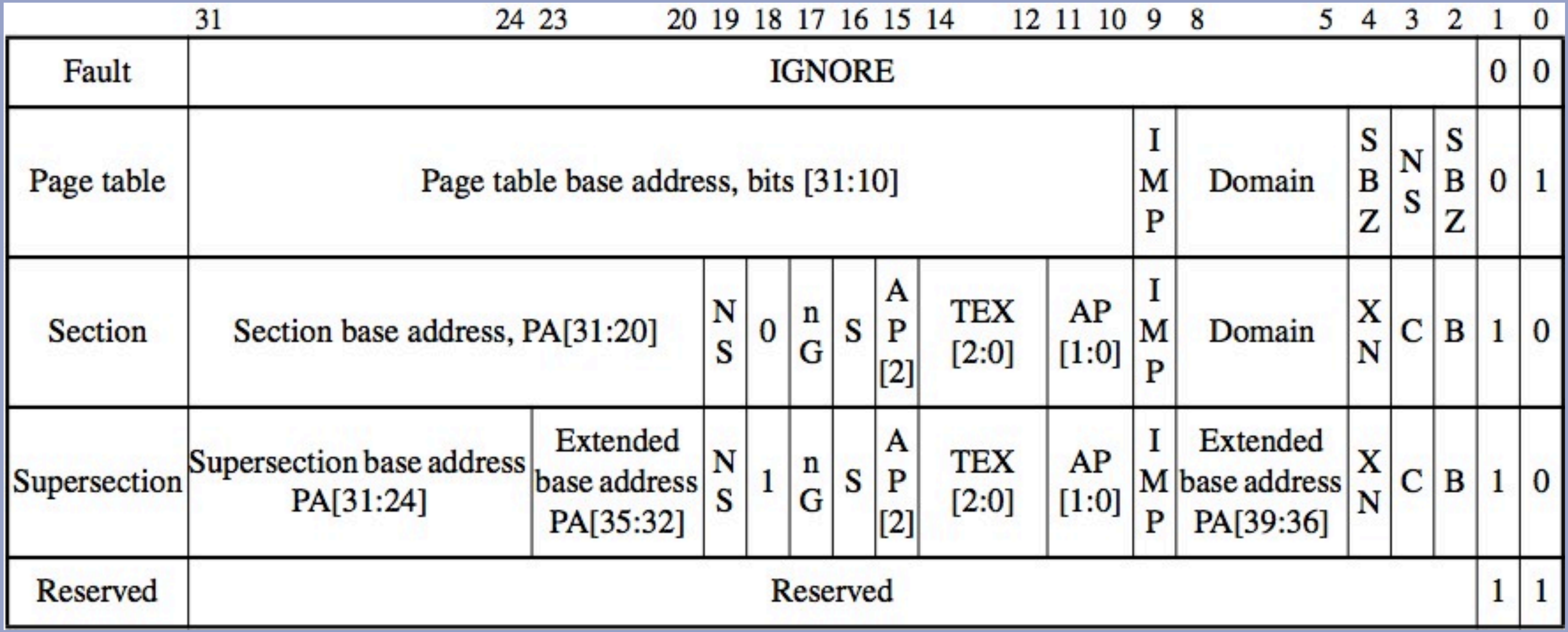
ARMv7-A MMU

ARMv7-A MMU

- Control using coprocessor 15 and mrc + mcr instructions.
- 2 level page tables.
- Page sizes: 4 KiB, 64 KiB, 1 MiB and 16 MiB
- Permissions for supervisor and user (read / write).
- No-Execute (NX) bit

ARMv7-A MMU

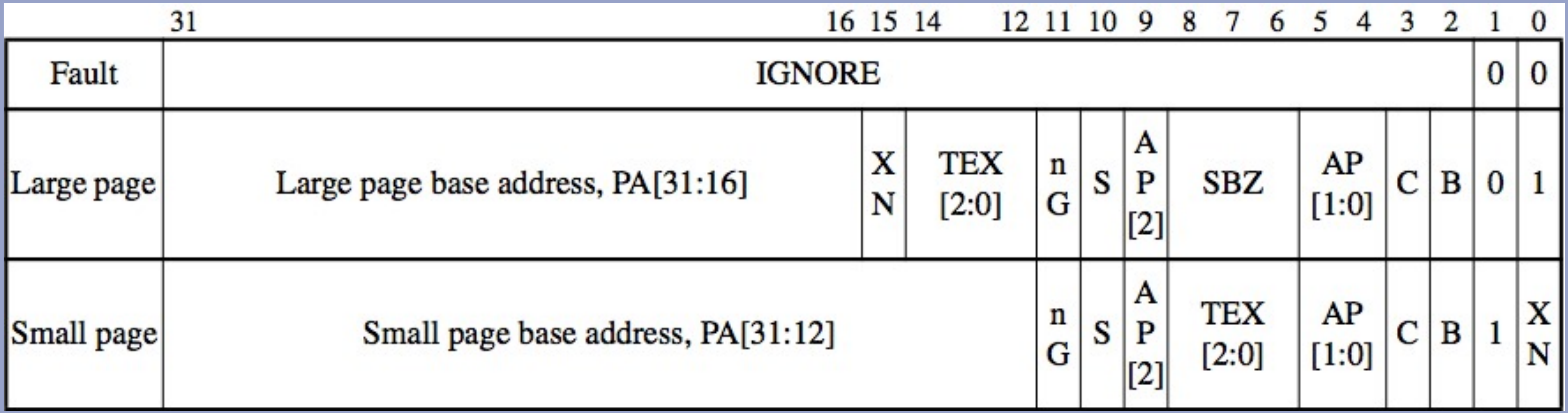
Level 1 Table Entries



	31 24	23 20	19	18	17	16	15	14 12	11 10	9	8 5	4	3	2	1	0
Fault	IGNORE														0	0
Page table	Page table base address, bits [31:10]									IMP	Domain	SBZ	NS	SBZ	0	1
Section	Section base address, PA[31:20]		NS	0	nG	S	AP[2]	TEX [2:0]	AP [1:0]	IMP	Domain	XN	C	B	1	0
Supersection	Supersection base address PA[31:24]	Extended base address PA[35:32]	NS	1	nG	S	AP[2]	TEX [2:0]	AP [1:0]	IMP	Extended base address PA[39:36]	XN	C	B	1	0
Reserved	Reserved														1	1

ARMv7-A MMU

Level 2 Table Entries



	31	16	15	14	12	11	10	9	8	7	6	5	4	3	2	1	0
Fault	IGNORE															0	0
Large page	Large page base address, PA[31:16]		XN	TEX [2:0]		nG	S	AP [2]	SBZ			AP [1:0]		C	B	0	1
Small page	Small page base address, PA[31:12]					nG	S	AP [2]	TEX [2:0]			AP [1:0]		C	B	1	XN

ARMv7-A MMU

- More page attributes:
  - Global
  - Memory region attributes
    - Cacheable, Bufferable, TEX
  - Shareable
  - Domain

ARMv7-A MMU

TEX[2:0]	C	B	Description
000	0	0	Strongly ordered
000	0	1	Shareable device
000	1	0	Outer and inner write-through, no write-allocate
000	1	1	Outer and inner write-back, no write-allocate
001	0	0	Outer and inner non-cacheable
001	0	1	Reserved
001	1	0	IMPLEMENTATION DEFINED
001	1	1	Outer and inner write-back, write-allocate
010	0	0	Non-shareable device
010	0	1	Reserved
010	1	-	Reserved
011	-	-	Reserved
1BB	A	A	Cacheable memory; outer = AA, inner = BB

ARMv7-A MMU

AA/BB	Attribute
00	Non-cacheable
01	Write-back, write-allocate
10	Write-through, no write-allocate
11	Write-back, no write-allocate

ARMv7-A MMU

- TEX remapping can be used to change TEX, C and B bits to an attribute index.
- Useful for operating systems to define a set of logical memory types using PRRR and NMRR registers.
  - TEX[0]:C:B = 0 → Device memory
  - TEX[0]:C:B = 1 → Normal memory

ARMv7-A MMU

- Large pages do not decrease table sizes.
- Sections and super-sections reduce the need for L2 table blocks and the penalty for walking the full table.
- Large pages, sections and super-sections increase the memory covered by the TLB.

ARMv7-A MMU

- Two root pointers, with configurable address coverage.
  - TTBR0: Recommended for user applications (non-global)
  - TTBR1: Recommended for system (global)

ARMv7-A TLBs

- The TLB caches the PTEs.
- PTE in TLB is if it is non global bound to an ASID which must be synced to the root PTP.
    - No TLB flush necessary on context switch as ASID will change.

Q&A