# 18-447 Lecture 19: Survey of Commercial VM Arch + a Decomposition of Meltdown

James C. Hoe
Department of ECE
Carnegie Mellon University

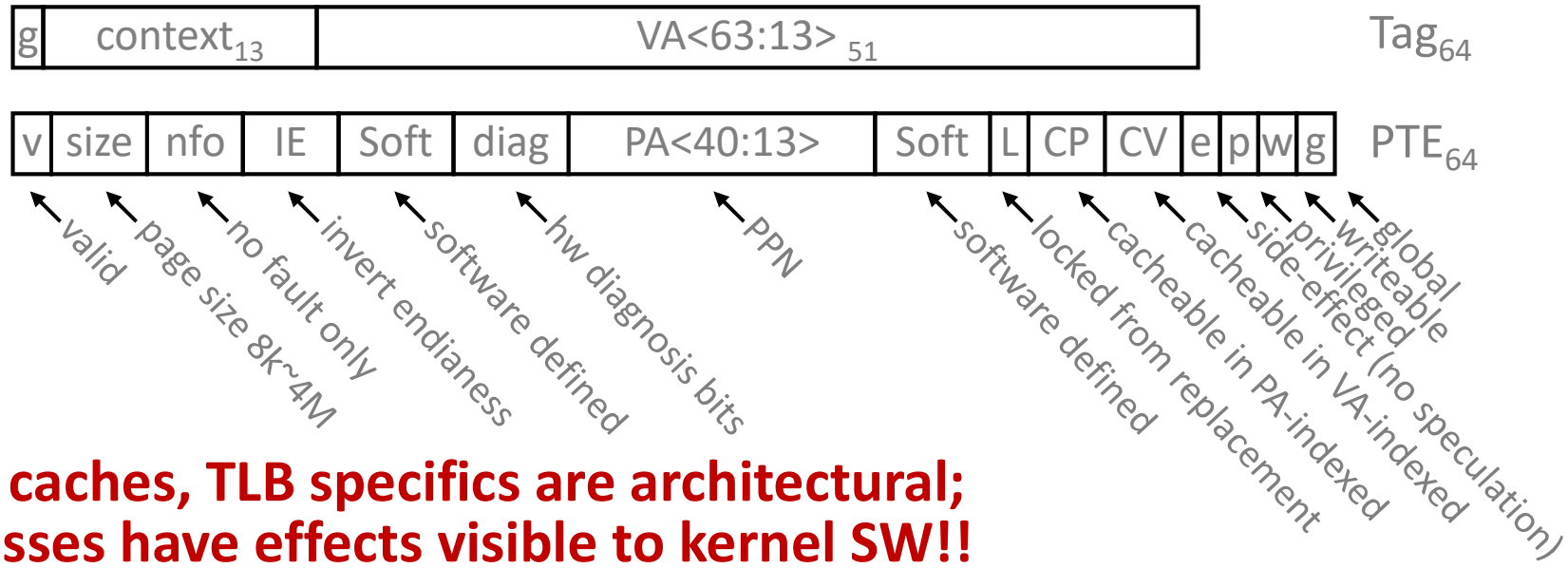
# Housekeeping

- Your goal today
  - see the many realizations of "VM", focusing on deviation from textbook-conceptual norms
  - put everything in 447 together in Meltdown
- Notices
  - Handout #15: Lab 4, due week 14
  - Handout #16: HW 5 solutions
  - Midterm 2, Wed, 4/6
- Readings
  - Synthesis Lecture: Architectural and Operating System Support for Virtual Memory (optional)
  - start on P&H Ch 6

# B. Jacob and T. Mudge, *Virtual Memory in Contemporary Processors*, IEEE Micro, 1998

# SPARC V9 PTE/TLB Entry

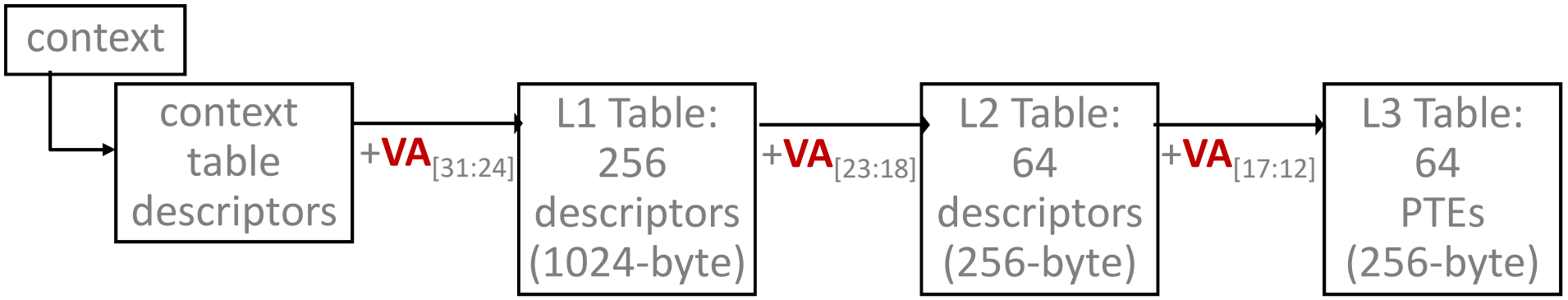
- 64-bit **VA** + context ID
  - implementation can choose not to map high-order bits (require sign extension in unmapped bits)
  - e.g., UltraSPARC 1 mapped only lower 44 bits
- **PA** space size set by implementation, $2^{28}$ max pgs
- 64 entry fully associative I-TLB and D-TLB




**Unlike caches, TLB specifics are architectural;**
**TLB misses have effects visible to kernel SW!!**

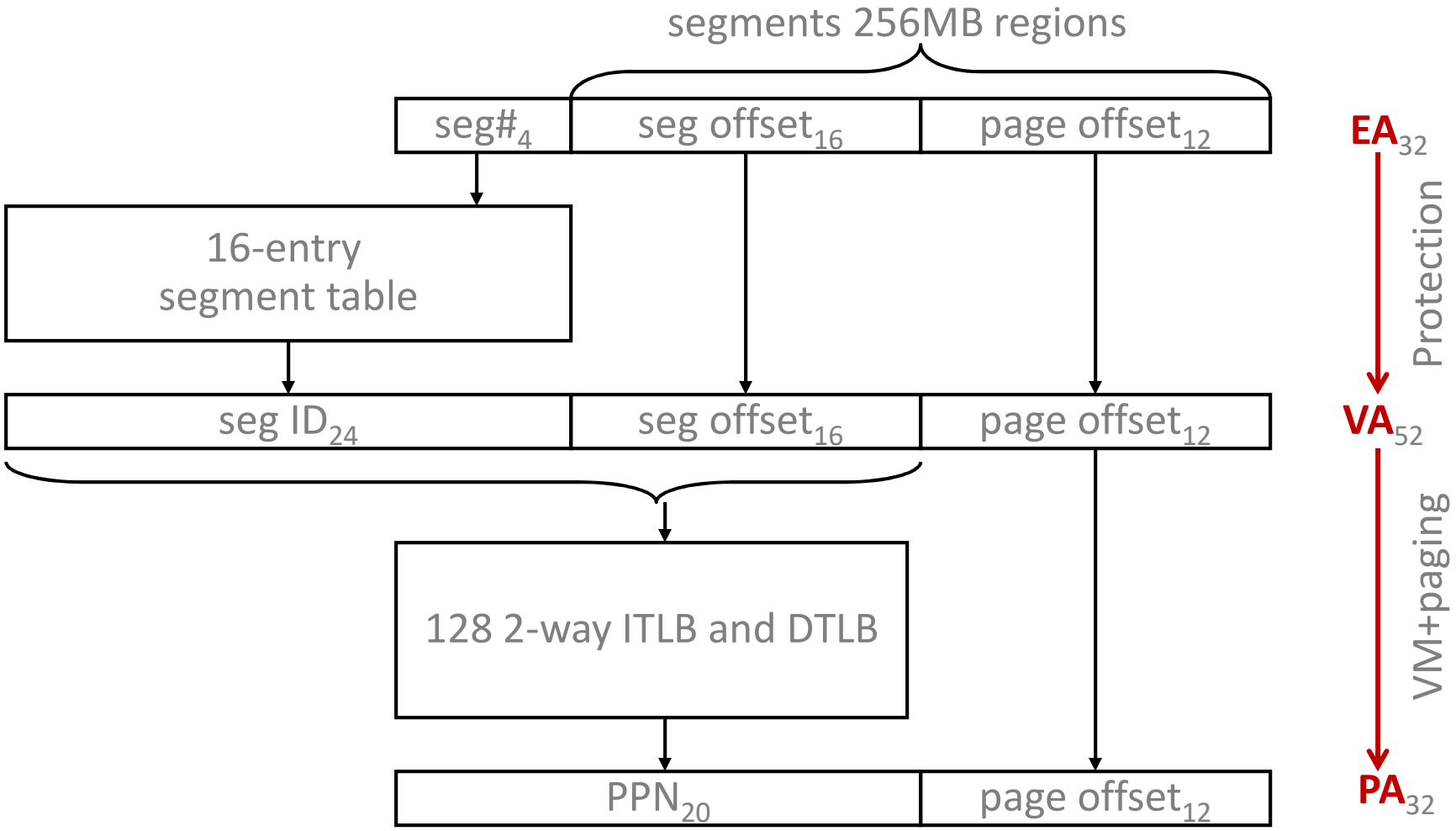
# SPARC TLB Miss Handling

- 32-bit V8 used a 3-level hierarchical page table for HW MMU page-table walk




- 64-bit V9 switched to Translation Storage Buffer
  - a software managed, in-DRAM direct-mapped "cache" of PTEs (think hashed pg table or SW TLB)
  - HW assisted address generation on a TLB miss
  - TLB miss handler (SW) searches TSB. If TSB misses, a slower TSB-miss handler takes over
  - OS can use any page table structure after TSB

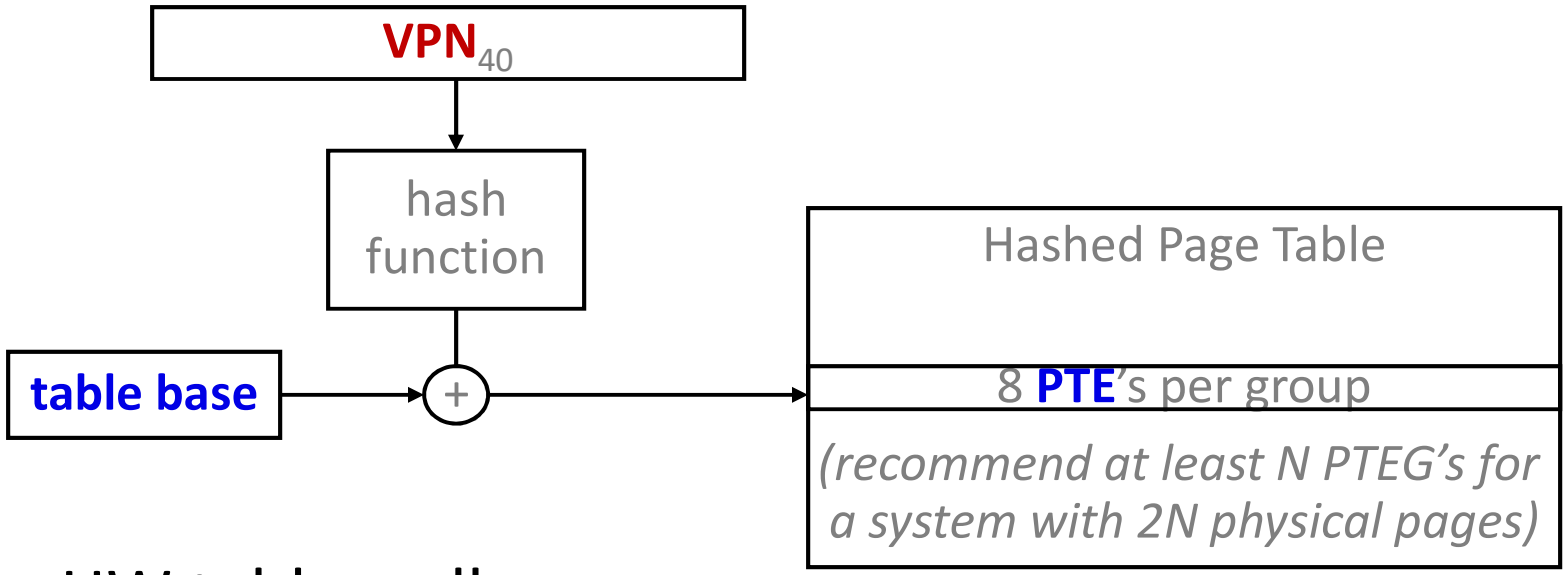
# IBM PowerPC (32-bit)




64-bit PowerPC = 64-bit **EA** -> 80-bit **VA** → 64-bit **PA**

How many segments in 64-bit **EA**?

# IBM PowerPC Hashed Page Table



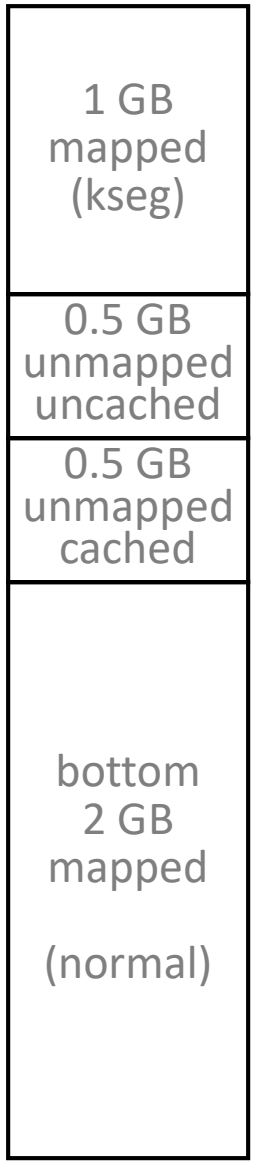

- HW table walk
  - **VPN** hashes into a PTE group (**PTEG**) of 8
  - 8 **PTE**s searched sequentially for tag match
  - if not found in first **PTEG** search a second **PTEG**
  - if not found in 2nd **PTEG**, trap to software handler
- Hashed table structure also used for 64-bit **EA→VA**

# MIPS R10K

- 64-bit **VA**
  - top 2 bits set kernel/supervisor/user mode
  - additional bits set cache and translation behavior
  - bit 61-40 not translate at all

    (holes and repeats in the **VA**??)
- 8-bit **ASID** (address space ID) distinguishes between processes
- 40-bit **PA**
- Translation -

  "64"-bit **VA** and 8-bit **ASID** → 40-bit **PA**




simplified example from 32-bit VA in R2000/3000

# MIPS TLB

- 64-entry fully associative unified TLB
- Each entry maps 2 consecutive **VPN**s to independent respective **PPN**s
- Software TLB-miss handling *(exotic at the time)*
  - 7-instruction page table walk in the best case
  - TLB Write Random: chooses a random entry for TLB replacement
  - OS can exclude low TLB entries from replacement (some translations must not miss)
- TLB entry
  - **N**: noncacheable
  - **V**: valid

| $VPN_{20}$ | $ASID_6$ | $0_6$ |
|---|---|---|
| $PPN_{20}$ | **ndvg** | $0_8$ |

R2000

**D**: dirty **(write-enable!!)**

**G**: ignore **ASID**

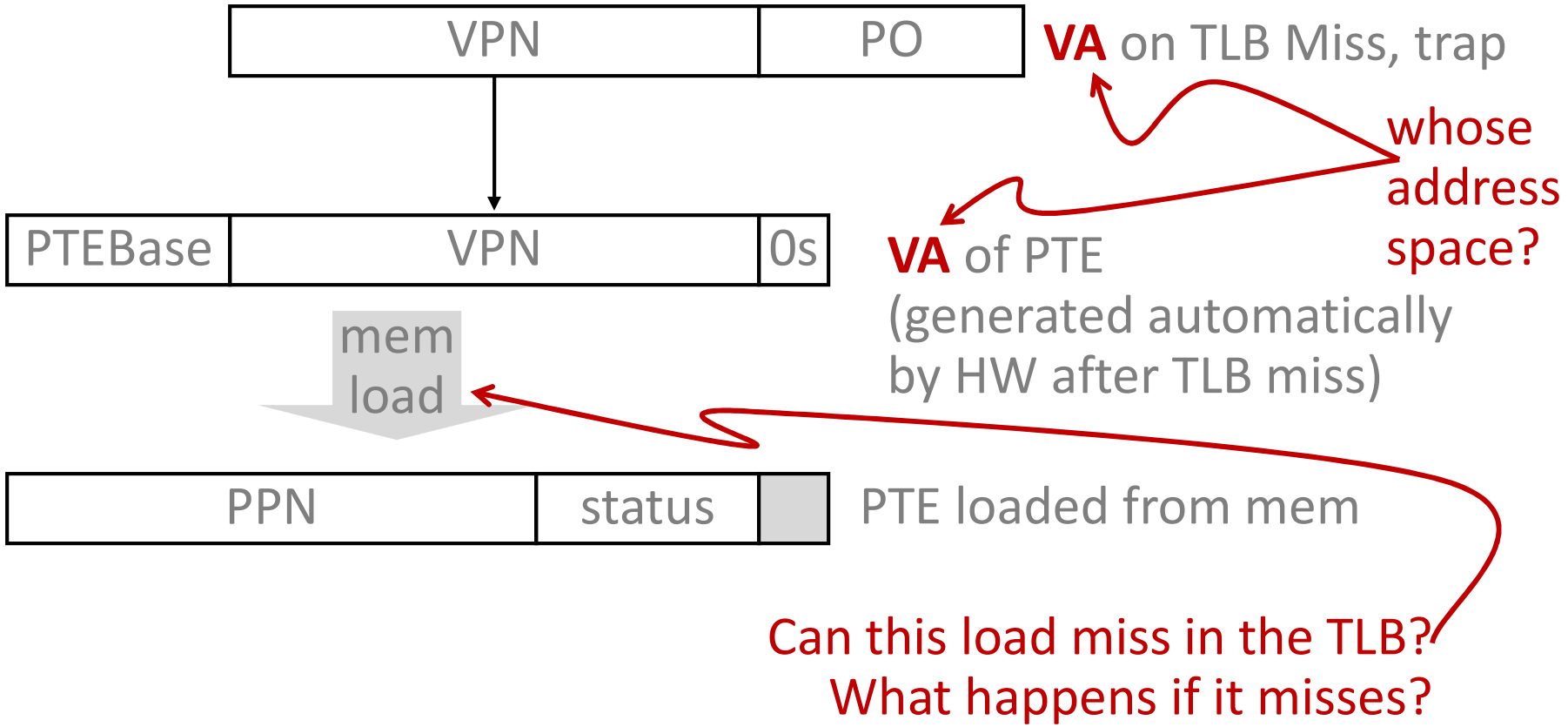
# MIPS Bottom-Up Hierarchical Table

- TLB miss vectors to a SW handler
  - page table organization is not hardcoded in ISA
  - ISA favors a chosen reference page table scheme by providing "optional" hardware assistance

- Bottom-Up Table
  - start with 2-level hierarchical table **(32-bit case)**
  - allocate all L2 tables for all **VA** pages (empty or not) linearly in the *mapped kseg* space
  - **VPN** is index into this linear table in **VA**

This table scales with VA size!! Is this okay?

# Bottom-Up Table Walk




*notice translation also eats up TLB entries!*

# User TLB Miss Handling

```
mfc0 k0,tlbcxt      # move the contents of TLB
                    #       context register into k0
mfc0 k1,epc         # move PC of faulting memory
                    #       instruction into k1
lw k0,0(k0)         # load thru address that was
                    #       in TLB context register
mtc0 k0,entry_lo    # move the loaded value (a PTE)
                    #       into the EntryLo register
tlbwr               # write PTE into the TLB
                    #       at a random slot number
j k1                # jump to PC of faulting
                    #       load instruction to retry
rfe                 # restore privilege (in delay slot)
```

# HP PA-RISC: PID and AID

- 2-level: 64b **EA**→96b **VA** (global)→64b **PA**
- Variable sized segmented **EA**→**VA** translation
- Rights-based access control
  - user controls segment registers (user can generate any **VA** it wants!!)

    *in contrast, everyone else controls translation to control what **VA** can be reached from a process*
  - each virtual page has an access ID (**AID**) assigned by OS
  - each process has 8 active protection IDs (**PID**s) in privileged HW registers controlled by OS
  - a process can access a page only if one of the 8 **PIDs** matches the page's **AID**

# Intel 80386

- Two-level address translation:

  segmented **EA** → global **VA** → **PA**

- User-private 48-bit **EA**
  - 16-bit **SN** (implicit) + 32-bit **SO**
  - 6 user-controlled registers hold active **SN**s; selected according to usage: code, data, stack, etc
- Global 32-bit **VA**
  - 20-bit **VPN** + 12-bit **PO**
- An implementation defined paged **PA** space

What is very odd about this?

# Living with the mistake

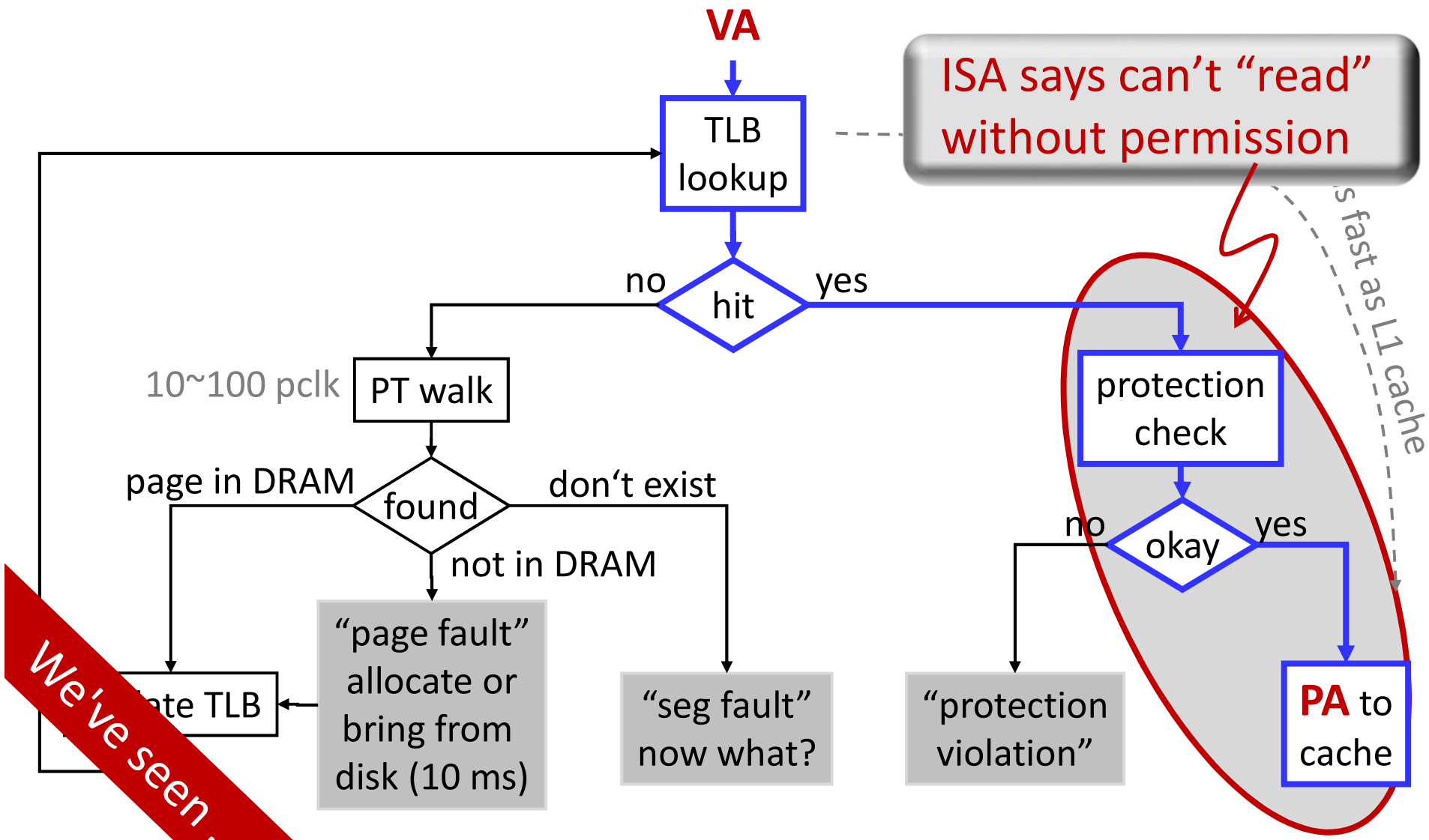
- 32-bit global **VA** too small to share by processes
  - per-process **EA** space oddly bigger than **VA** space
  - until 1990, no one cared *DOS and Windows*
- Later multitasking OS ignore segment protection
  - time-multiplex *****global***** **VA** space for use by 1 process at a time
  - code, data, stack segments always map to entire **VA** space, $0\sim(2^{32}-1)$
  - set MMU to use a different table on context switch
  - BUT! TLB for **VA** translation doesn't have **ASID**; must flush TLB on context switch

*Later IA32e added* **PCID** *to TLB as fix*

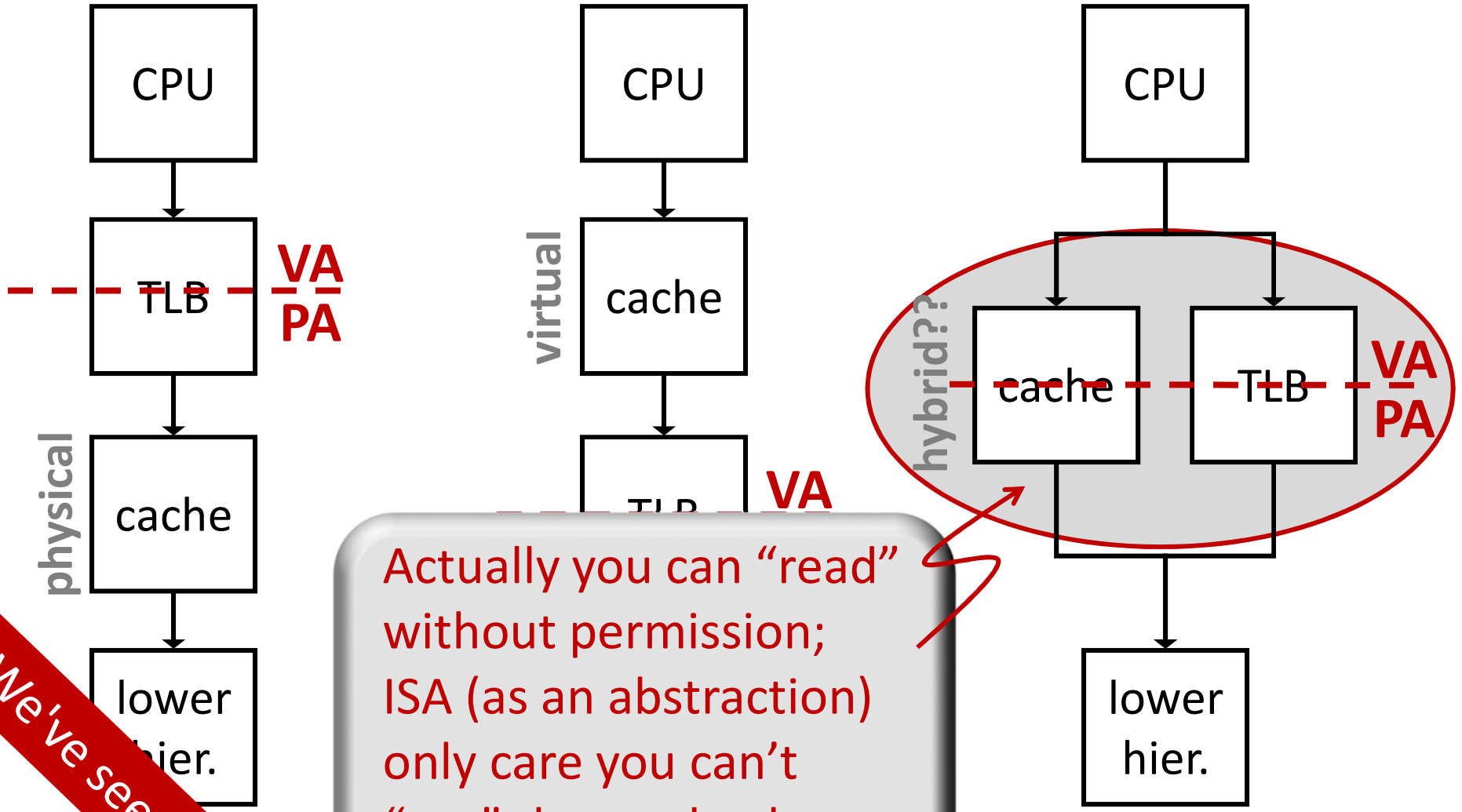
# Meltdown in 18-447 Terms

***How to "know" the value at a memory location without permission to read it?***

# VA to PA Translation Flow Chart

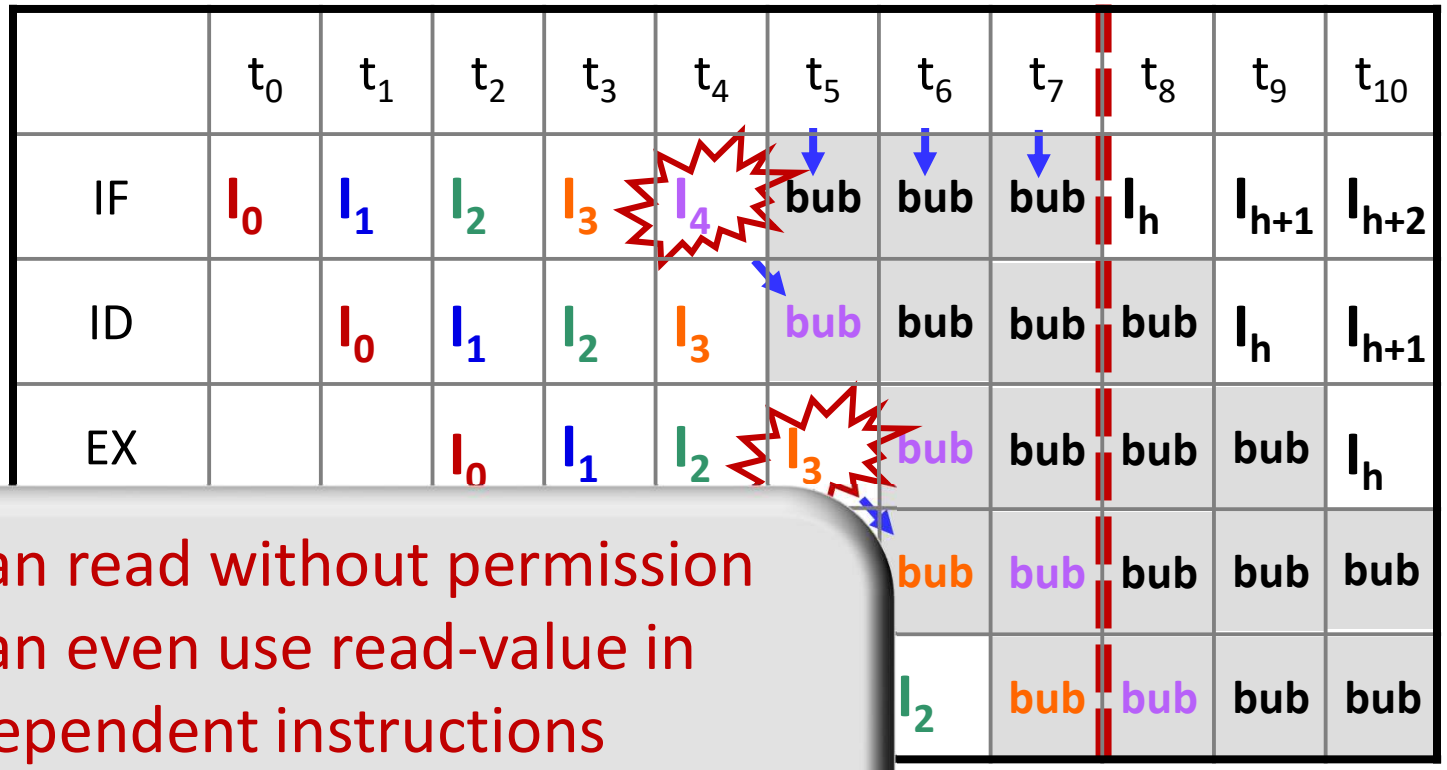


# How should VM and Cache Interact?



# "Flushing" a Pipeline

privileged mode



| | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| IF | $I_0$ | $I_1$ | $I_2$ | $I_3$ | $I_4$ | bub | bub | bub | $I_h$ | $I_{h+1}$ | $I_{h+2}$ |
| ID | | $I_0$ | $I_1$ | $I_2$ | $I_3$ | bub | bub | bub | bub | $I_h$ | $I_{h+1}$ |
| EX | | | $I_0$ | $I_1$ | $I_2$ | $I_3$ | bub | bub | bub | bub | $I_h$ |
| | | | | | | | bub | bub | bub | bub | bub |
| | | | | | | | $I_2$ | bub | bub | bub | bub |

- can read without permission
- can even use read-value in dependent instructions
- as long as at the end can't "see" any of it

***100s of speculative instructions in flight in modern OOO CPUs***

rain older inst

g inst. is oldest

until pipeline is empt

er to be safe than fast

We've seen …

# Key Idea 3: Inter-Model Compatibility

"a valid program whose logic will not depend implicitly upon time of execution and which runs upon configuration A, will alsc
latter includes at least th
required I/O devices …."

**a fundamental tenet that ISA does not care about time**

- Invalid programs not constrained to yield same result
  - "invalid"==violating architecture manual
  - "exceptions" are architecturally defined
- The King of Binary Compatibility: Intel x86, IBM 360
  - stable software base and ecosystem
  - performance scalability

We've seen …

[Amdahl, Blaauw and Brooks, 1964]

# What cache is in your computer?

- How to figure out what cache configuration is in your co
  - capa
  - number of levels

Cache invisible architecturally, but performance "side-effect" easily detectable using timer

- The presence or lack of a cache should not be detectable by functional behavior of software
- But you could tell if you measured execution time to infer the number of cache misses

We've seen …

Infer read-value without "seeing" by running code to cause hit/miss based on unseen value

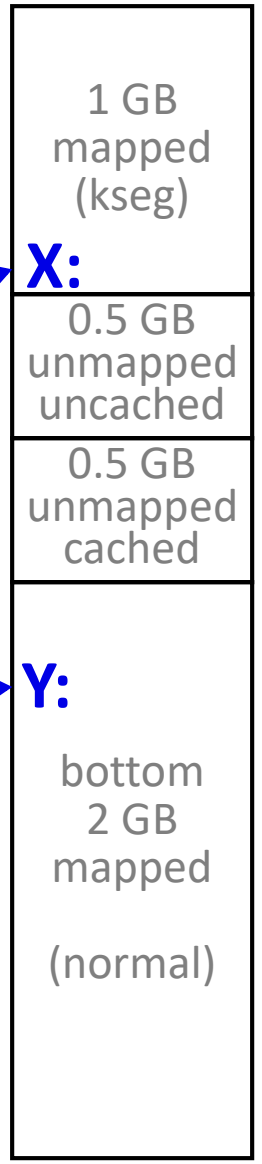
# MIPS R10K

- 64

Read addr **Y**+C, **Y**+2C, **Y**+3C . . . so addr **Y** is not in cache; then attempt to execute:

```
I1: lw t0, 0(r"X")
I2: andi t0, t0, 0x1
I3: sli t0, t0, "log2(blocksize)"
I4: add t0, t0, r"Y"
I5: lw x0, 0(t0)
```

$I_1$ is an exception so $I_1$~$I_5$ not observed architecturally; nevertheless addr **Y** is cached if LSB of mem[**X**] is 0

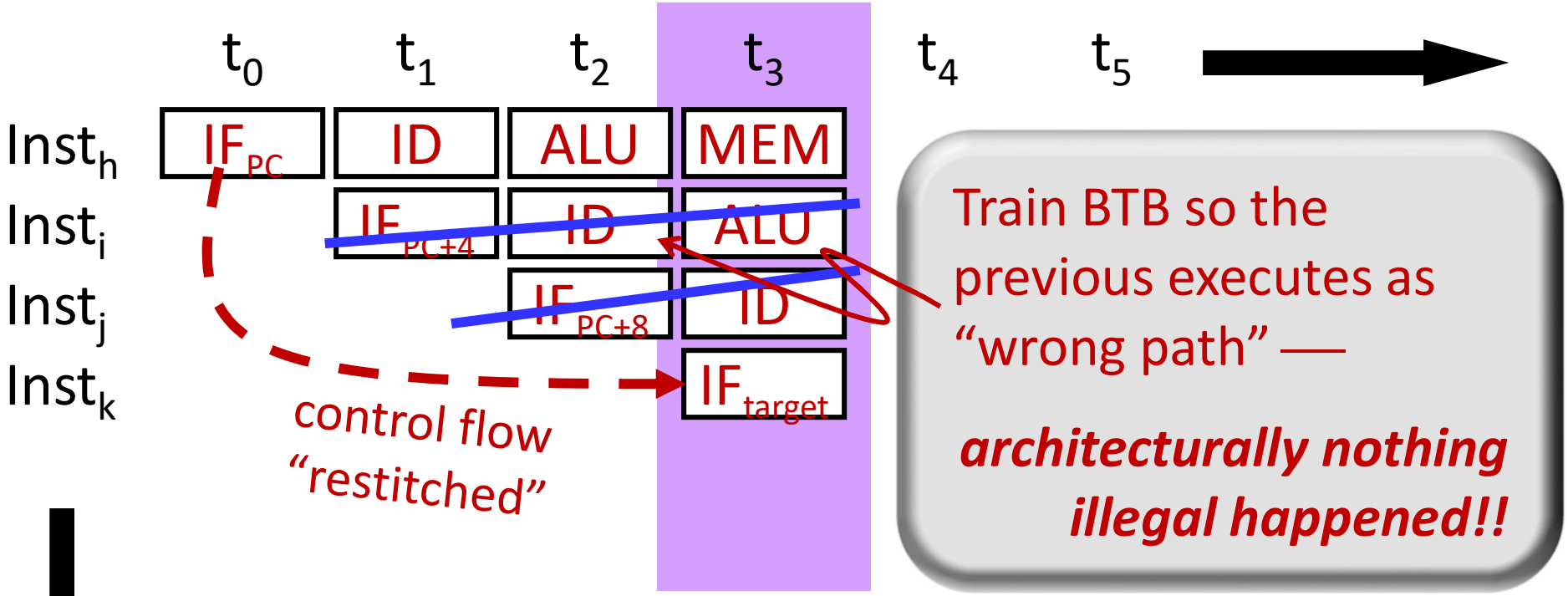



simplified example from 32-bit VA in R2000/3000

We've seen ...

-bit **VA** and 8-bit **ASID** ⇒ 40-bit **PA**

# Control Speculation: PC+4




We've seen ...

Insth is ... branch

When insth branch resolves
- branch target (Instk) is fetched
- flush instructions fetched since insth ("wrong-path")

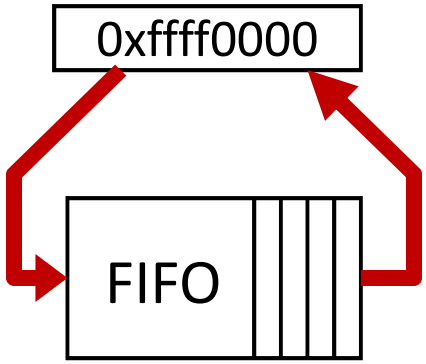
# Idempotency and Side-effects

- Meltdown vulnerability not a bug but an ISA-allowed simplification—*no fast kill after exception as with BP miss*
- Same issue doesn't arise with MMIO—ISA disallows spurious read if PTE says "uncacheable" or "side-effect"

***Not a "bug" but something is very wrong!!!***

***How to fix this . . . .***

- LW/SW to mmap locations can have side-effects
  - reading/writing mmap location can imply commands and other state changes
  - consider a FIFO example
    - SW to 0xffff0000 pushes value
    - LW from 0xffff0000 returns popped value




We've seen …

What happens if 0xffff0000 is cached?