EPFL

Modularity through Memory Virtualization

Prof. George Candea
School of Computer & Communication Sciences

Objectives

- Understand memory virtualization
  - *the coolest form of modularization we have in operating systems*
- Back-of-the-envelope calculations
  - *sanity-check a design before writing any code*

Outline

- Names and Page Tables
  - *PT = directory for mapping memory names (VA) to memory locations (PA)*
  - *Specific example: Intel Skylake microarchitecture*
- Caching
  - *Generalities*
  - *Caching in name translation*
- Key reference values for system design

Examples of Names

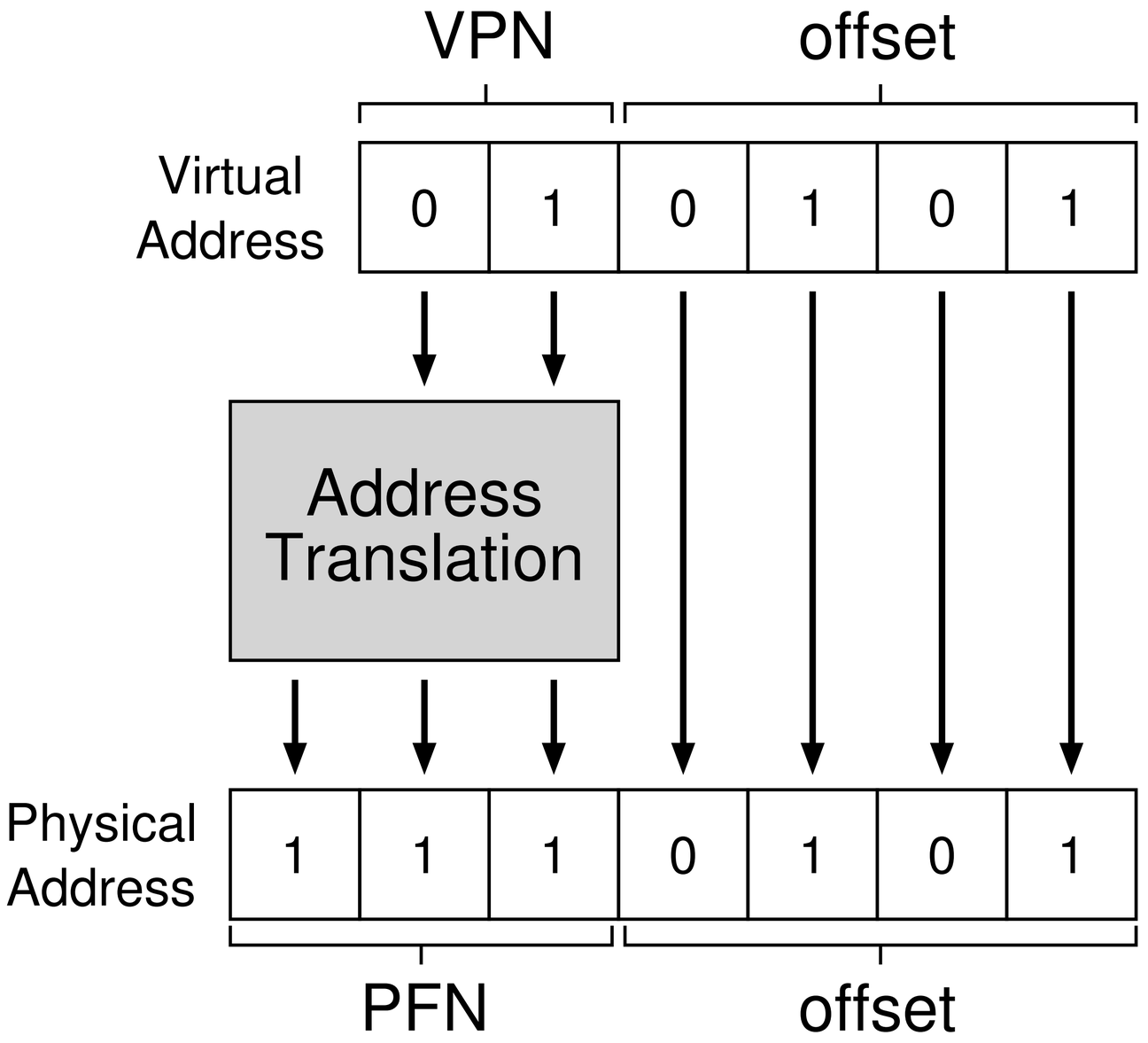
- Memory address
  - *names a memory location*
- Pointer
  - `void*` names the location of a memory word
  - `int*` names the location of an integer
- Virtual memory address
  - *names (from userspace) a physical memory address*

Names & Page Tables

Names

- Scope
  - *Private: unique within a context (e.g., a private IP address)*
  - *Global: unique across contexts (e.g., a global IP address)*
- Structure
  - *Hierarchical: name relationship implies object relationship (e.g., two IP addresses sharing the same prefix)*
  - *Flat: name relationship implies nothing (e.g., content IDs in Peer-to-Peer networks)*
- Naming system
  - *Directories of name→value mappings, support name lookups and updates*

Indirection



Virtual namespace

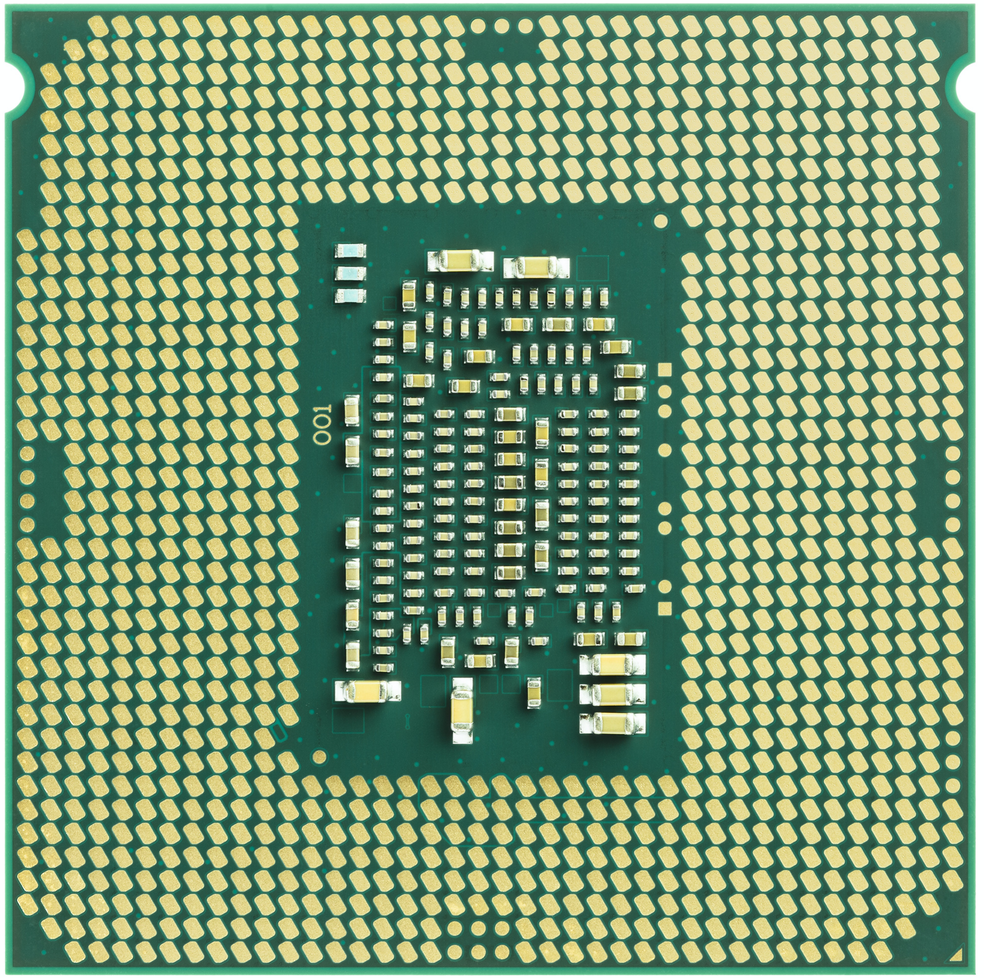
Name translation controls visibility

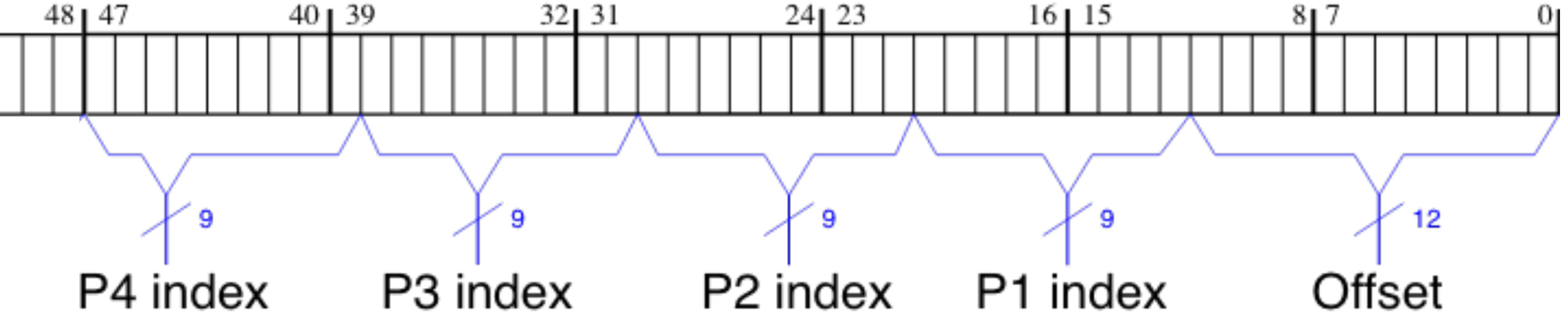
"if you can't name it, you can't use it"

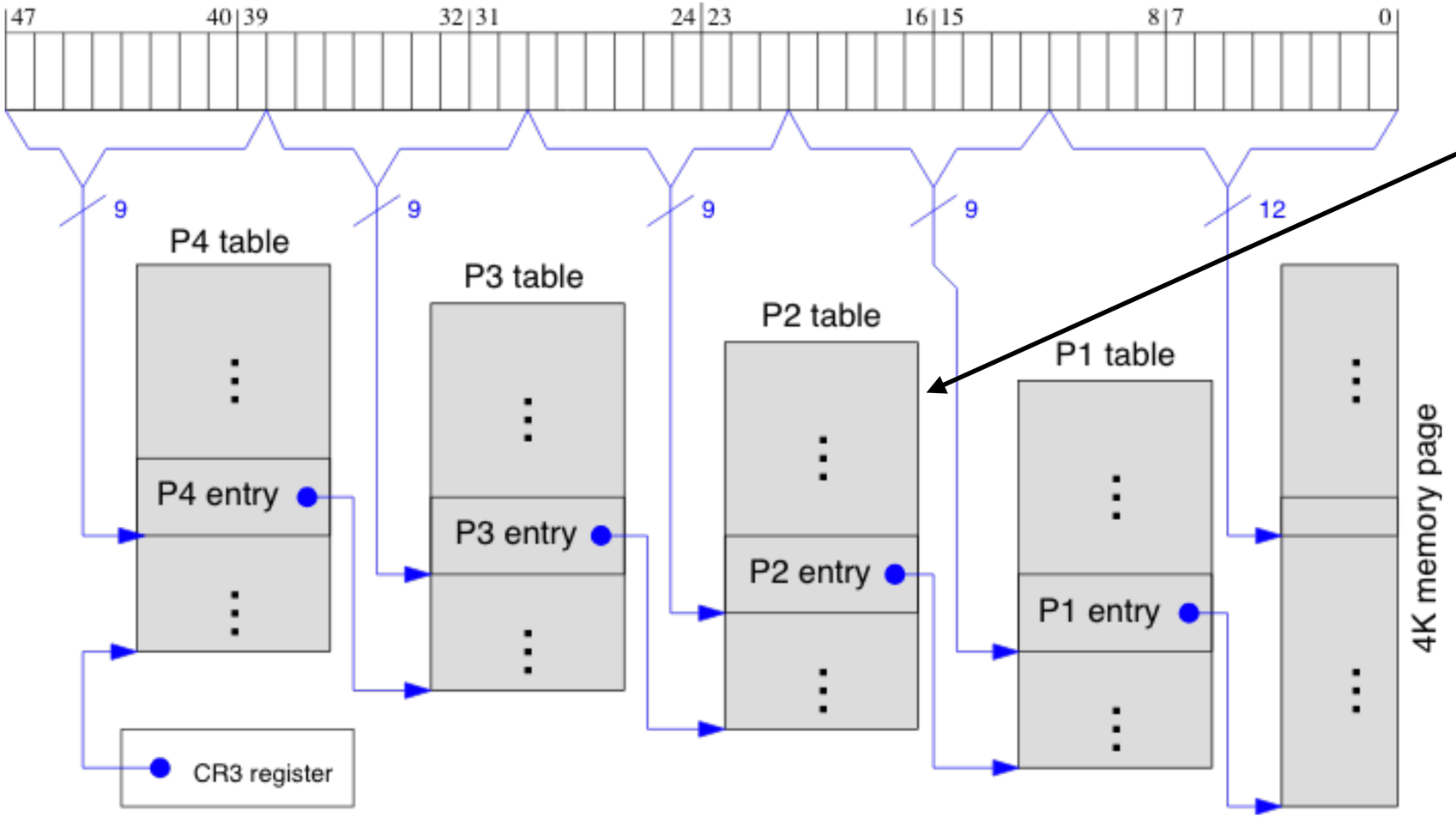
Physical namespace

Diagram courtesy of http://pages.cs.wisc.edu/~remzi/OSTEP


INTEL® CORE™ i7
i7-6700K
SR2L0 4.00GHZ
X611A978 e4





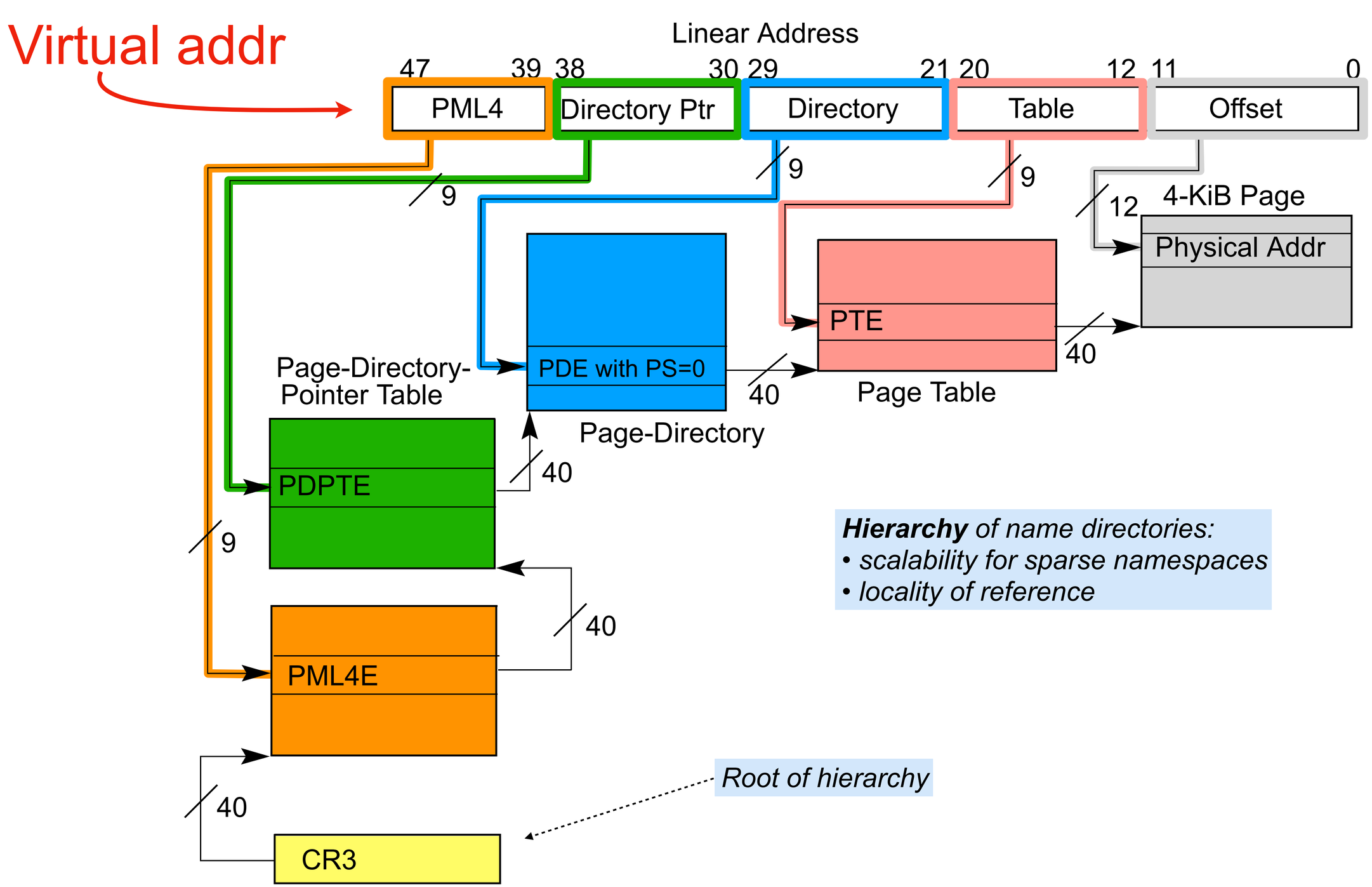


2^9=512 entries
8 bytes / entry

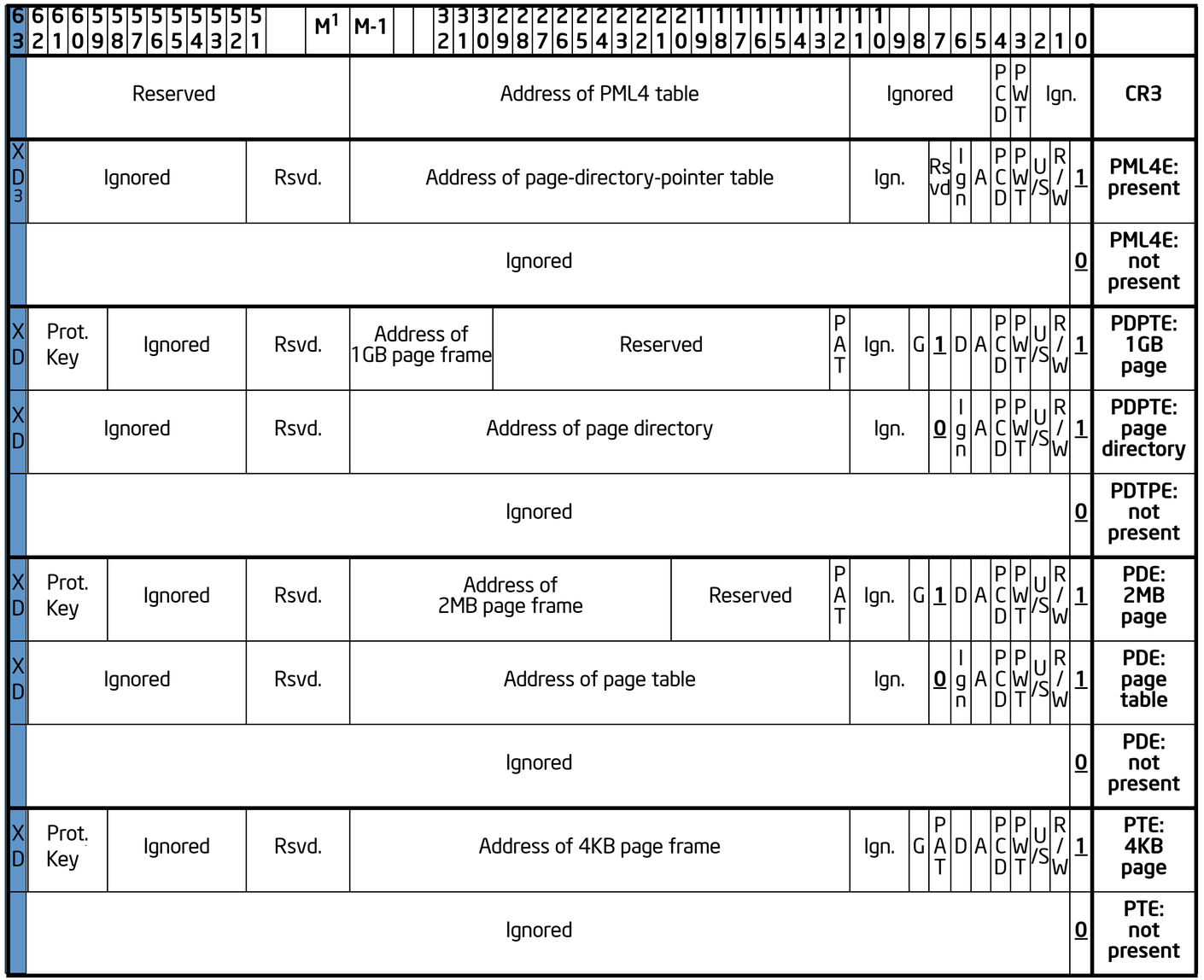
Hierarchy *of name directories:*
- *scalability for sparse namespaces*
- *locality of reference*

HW walks page tables
OS updates them

Diagrams courtesy of https://os.phil-opp.com/page-tables/


Virtual addr
Linear Address
47
39
38
30
29
21
20
12
11
0
PML4
Directory Ptr
Directory
Table
Offset
9
9
9
12
4-KiB Page
Physical Addr
PTE
PDE with PS=0
Page-Directory-
Pointer Table
40
40
Page Table
Page-Directory
PDPTE
40
9
PML4E
40
40
CR3
Root of hierarchy
Hierarchy of name directories:
• scalability for sparse namespaces
• locality of reference

VA bit index



CR3 (root)

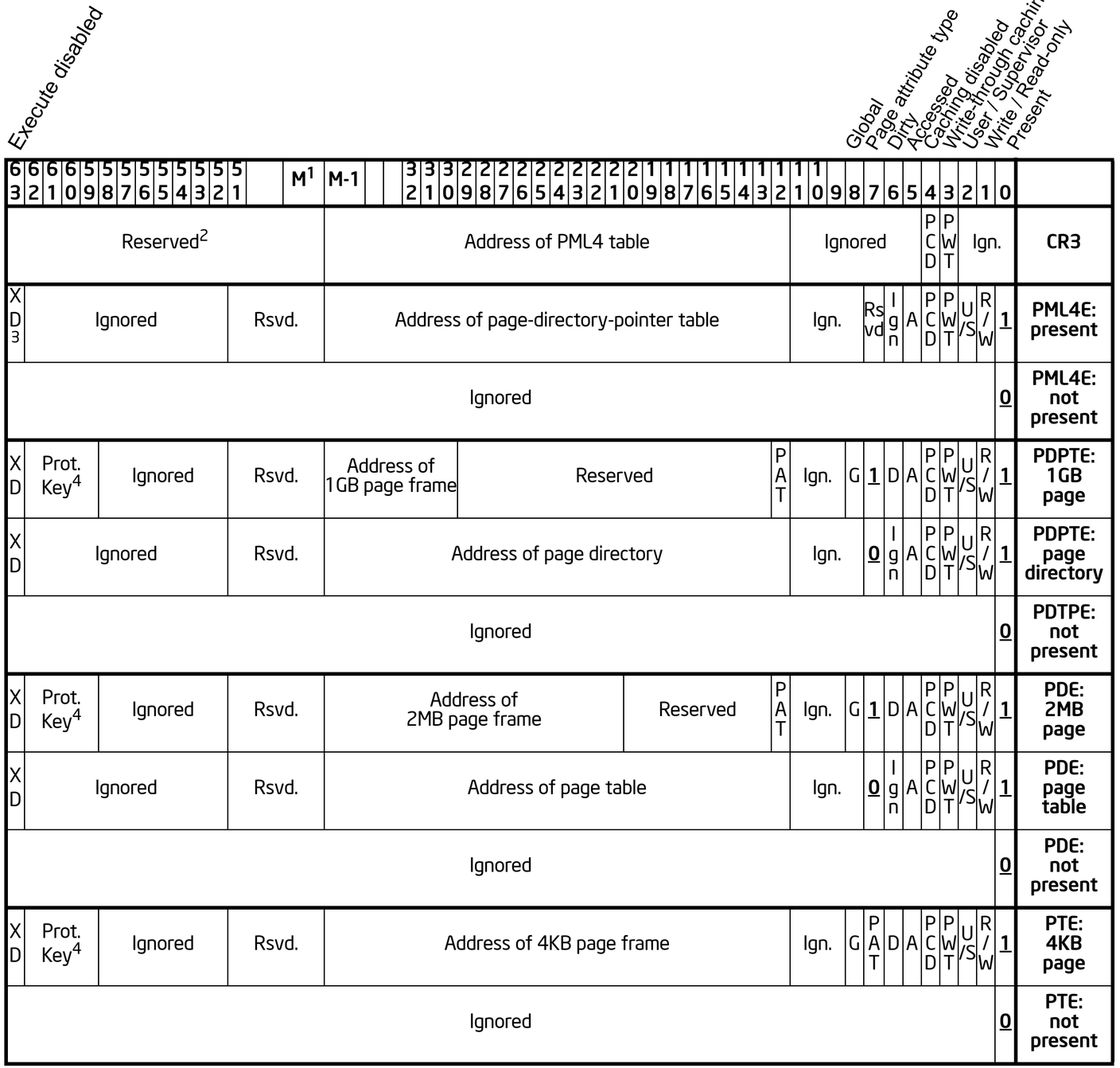
Top-level page table

2nd-level page table

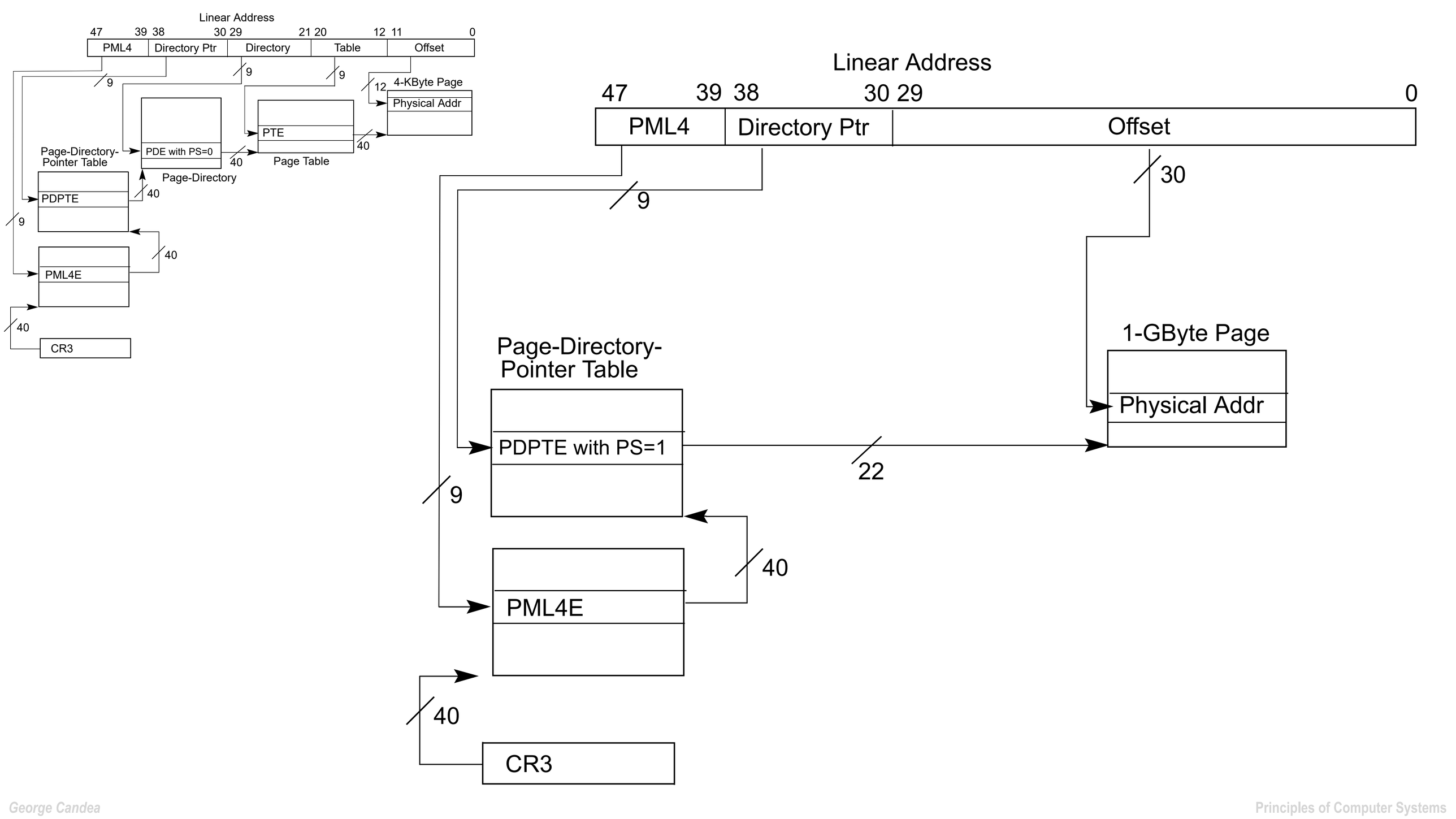
3rd-level page table

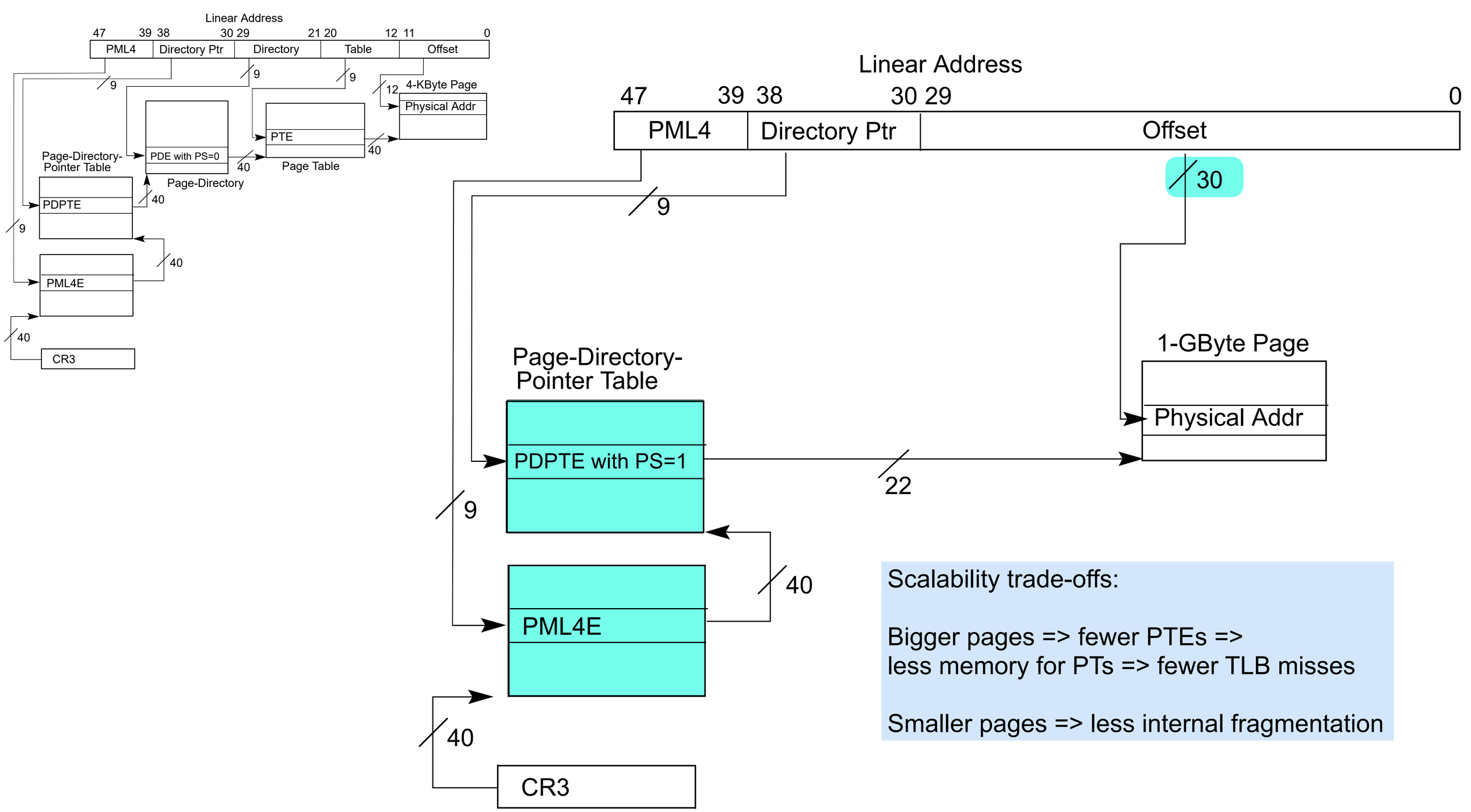
Last-level page table

Intel® 64 and IA-32 Architectures Software Developer's Manual . Volume 3: System Programming Guide, May 2018



Intel® 64 and IA-32 Architectures Software Developer's Manual .
Volume 3: System Programming Guide, May 2018

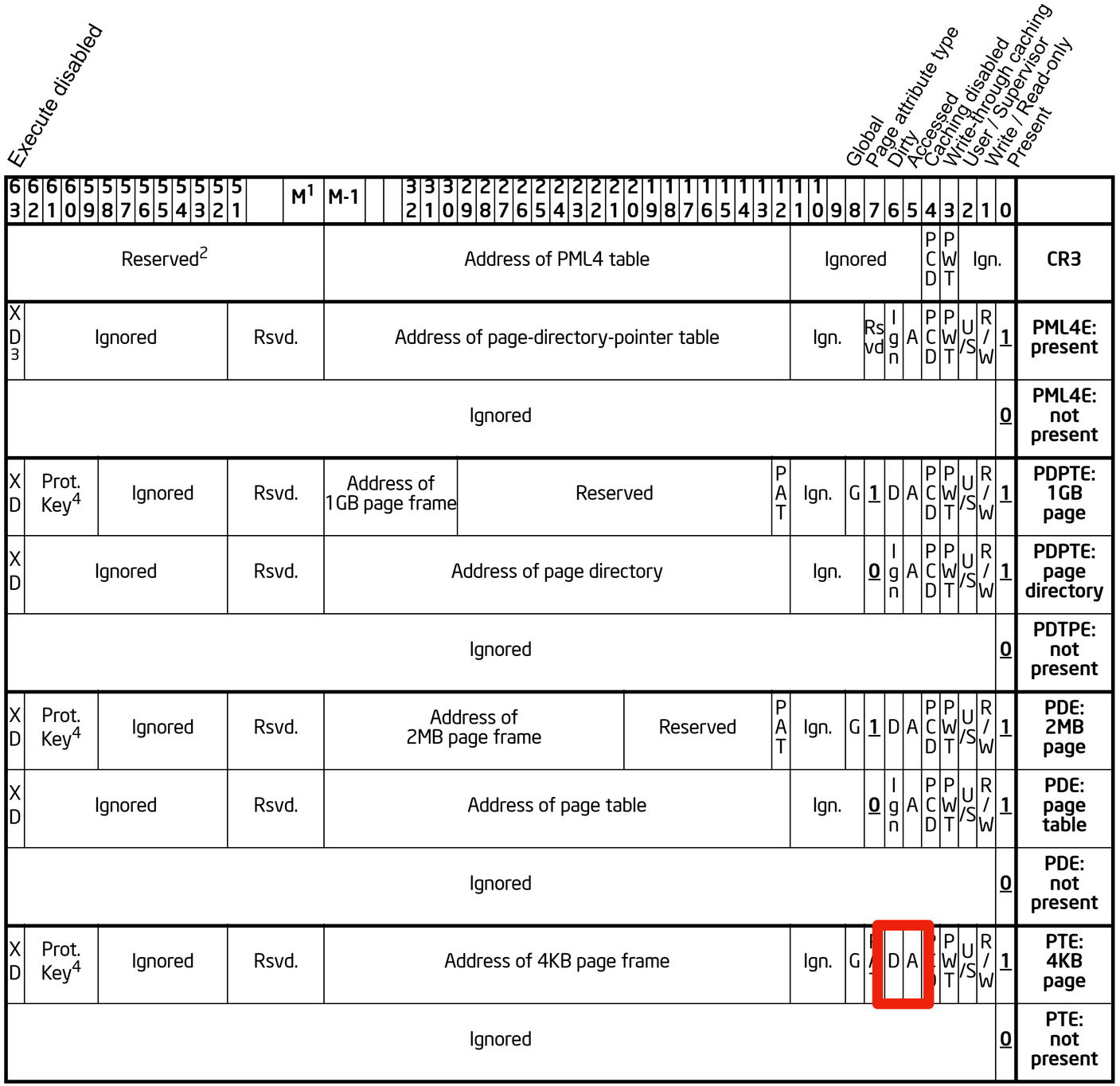

Linear Address
47
39
38
30
29
21
20
12
11
0
PML4
Directory Ptr
Directory
Table
Offset
9
9
9
12
4-KByte Page
Physical Addr
PTE
40
Page Table
PDE with PS=0
40
Page-Directory
Page-Directory-
Pointer Table
PDPTE
40
9
40
PML4E
40
CR3
Linear Address
47
39
38
30
29
0
PML4
Directory Ptr
Offset
9
30
1-GByte Page
Physical Addr
Page-Directory-
Pointer Table
PDPTE with PS=1
22
9
40
PML4E
40
CR3



Scalability trade-offs:

Bigger pages => fewer PTEs =>
less memory for PTs => fewer TLB misses

Smaller pages => less internal fragmentation



4 x 50 ns table lookup +
1 x 50 ns access =
250 ns =
~750 instructions @ 3 GHz clock

I could be doing a lot of useful work while waiting to read memory !

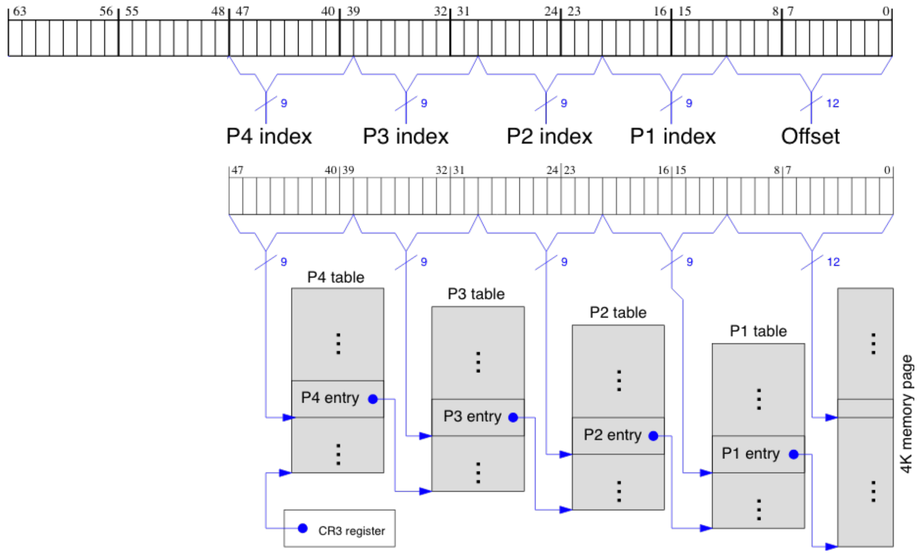
What to do ?

Intel® 64 and IA-32 Architectures Software Developer's Manual . Volume 3: System Programming Guide, May 2018

Caching Generalities

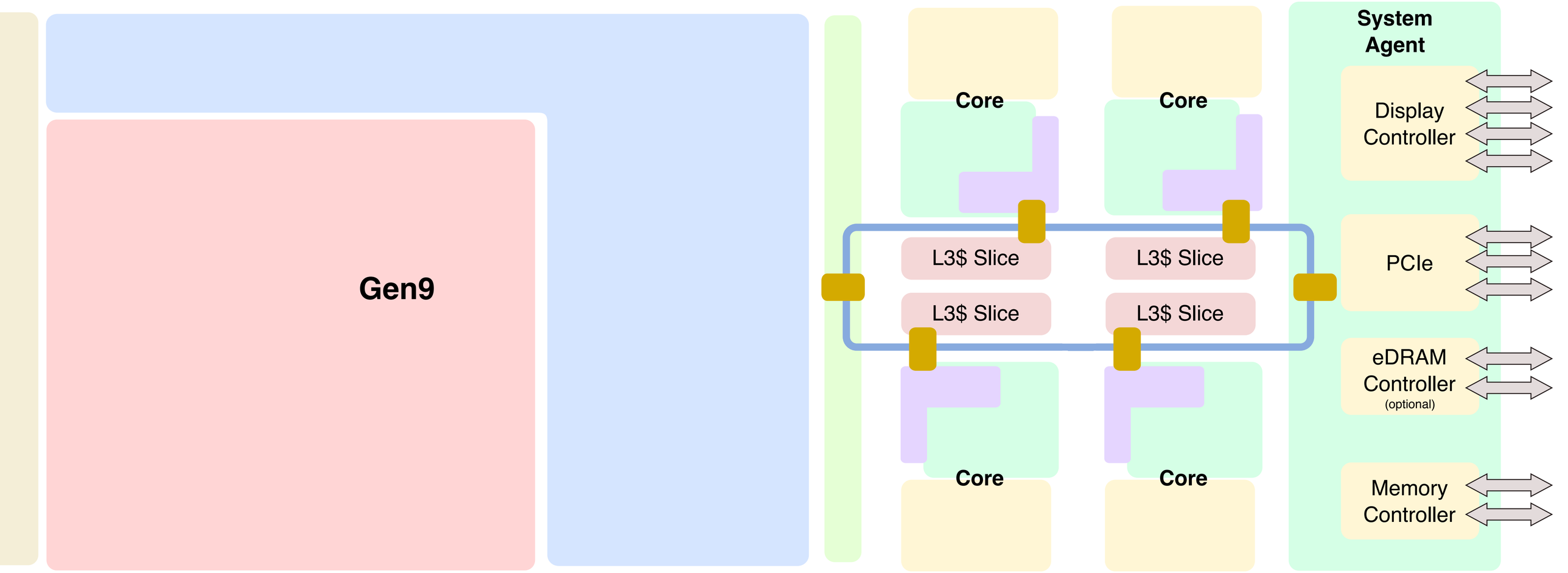
Recap of Key Concepts in Caching

- Locality
  - *temporal (reuse within short amount of time)*
  - *spatial (use shortly physically nearby data)*
- Replacement algorithm
  - *LRU, Time-aware LRU in CDNs, Least-frequent LRU in CDNs, MRU in scanning big data*
- Write-through vs. write-back
- Working set
- Direct-mapped vs. partially associative vs. fully associative
- Speed vs. size (and thus cost) trade-off



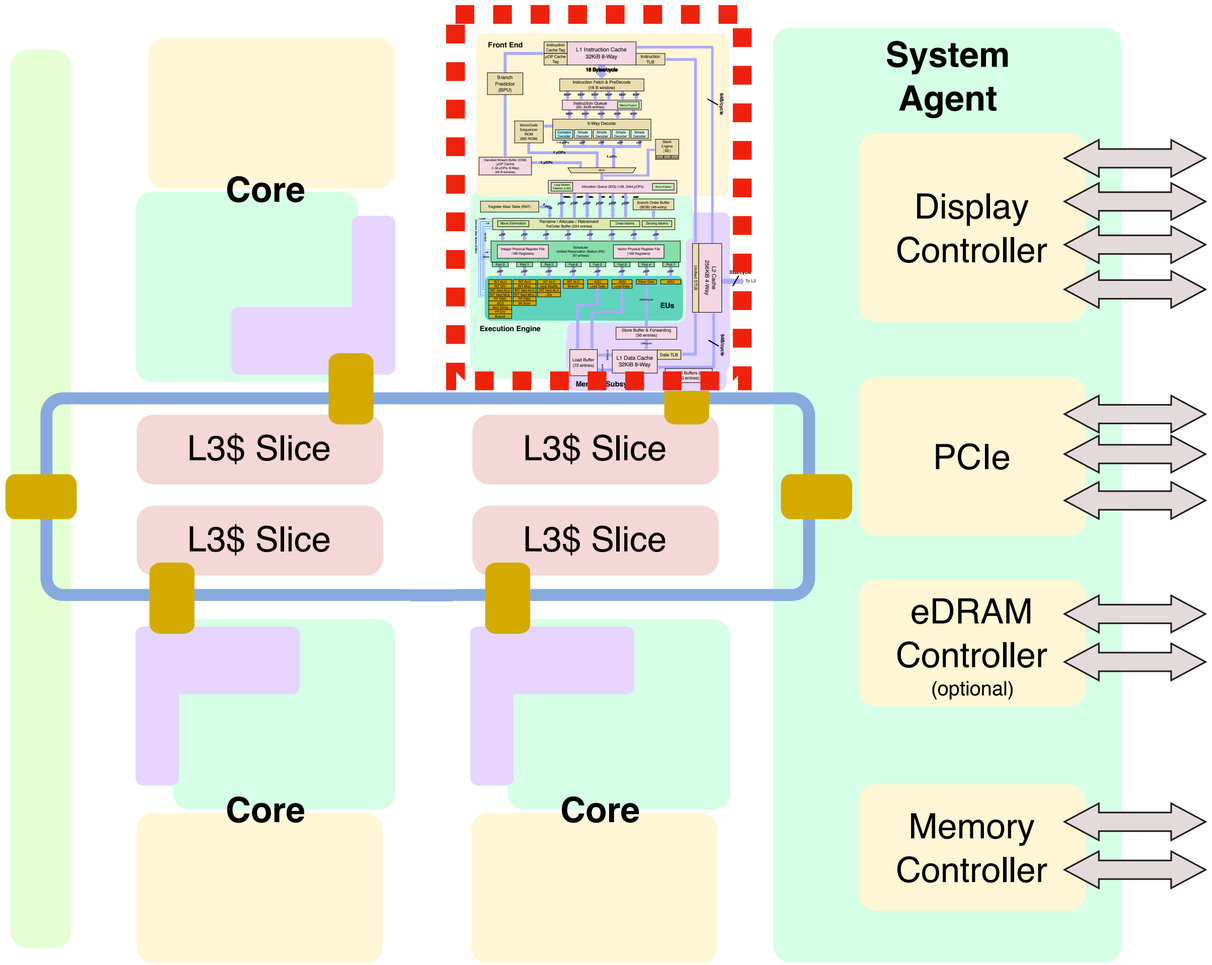
Caching & TLBs

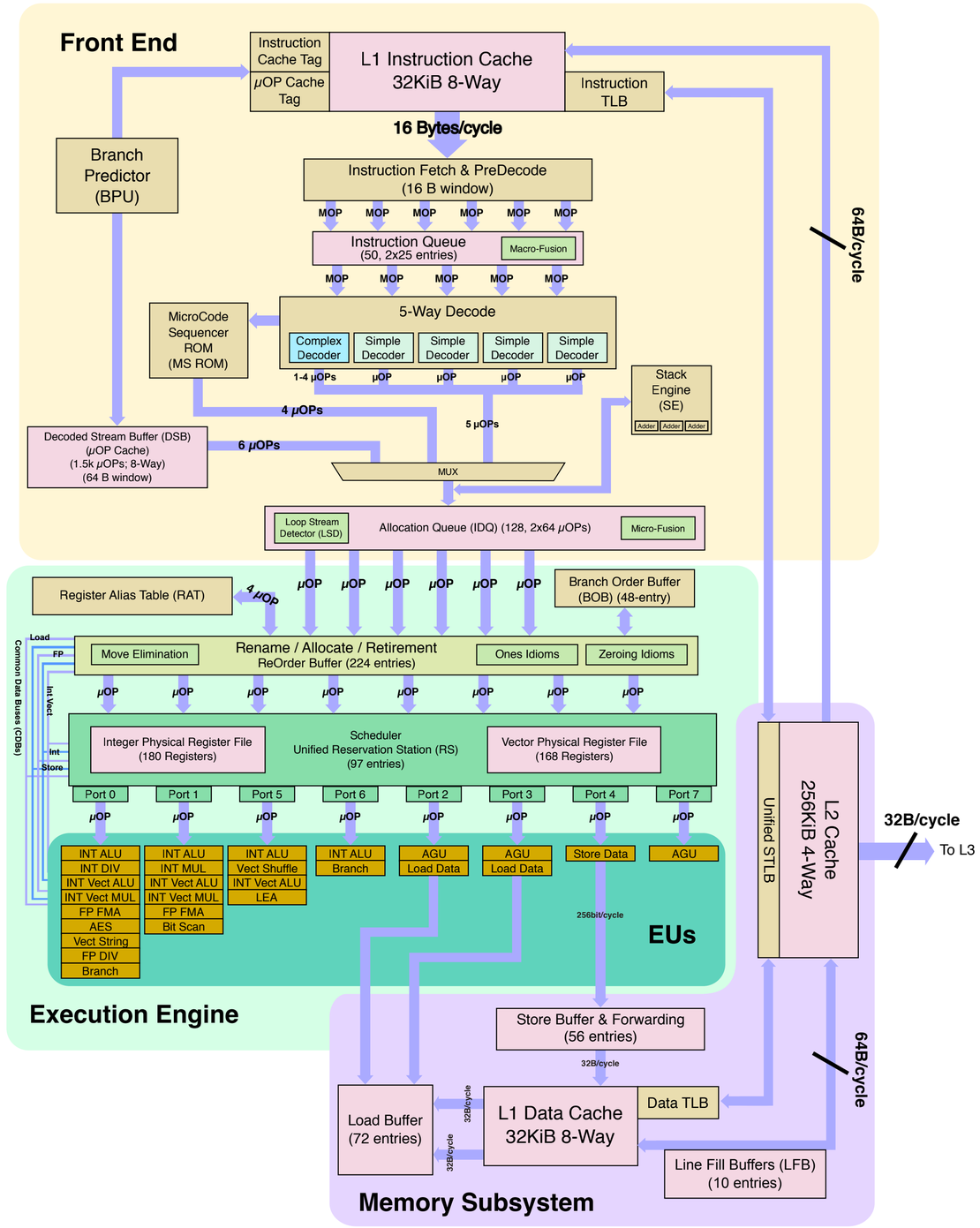
Intel Skylake Microarchitecture

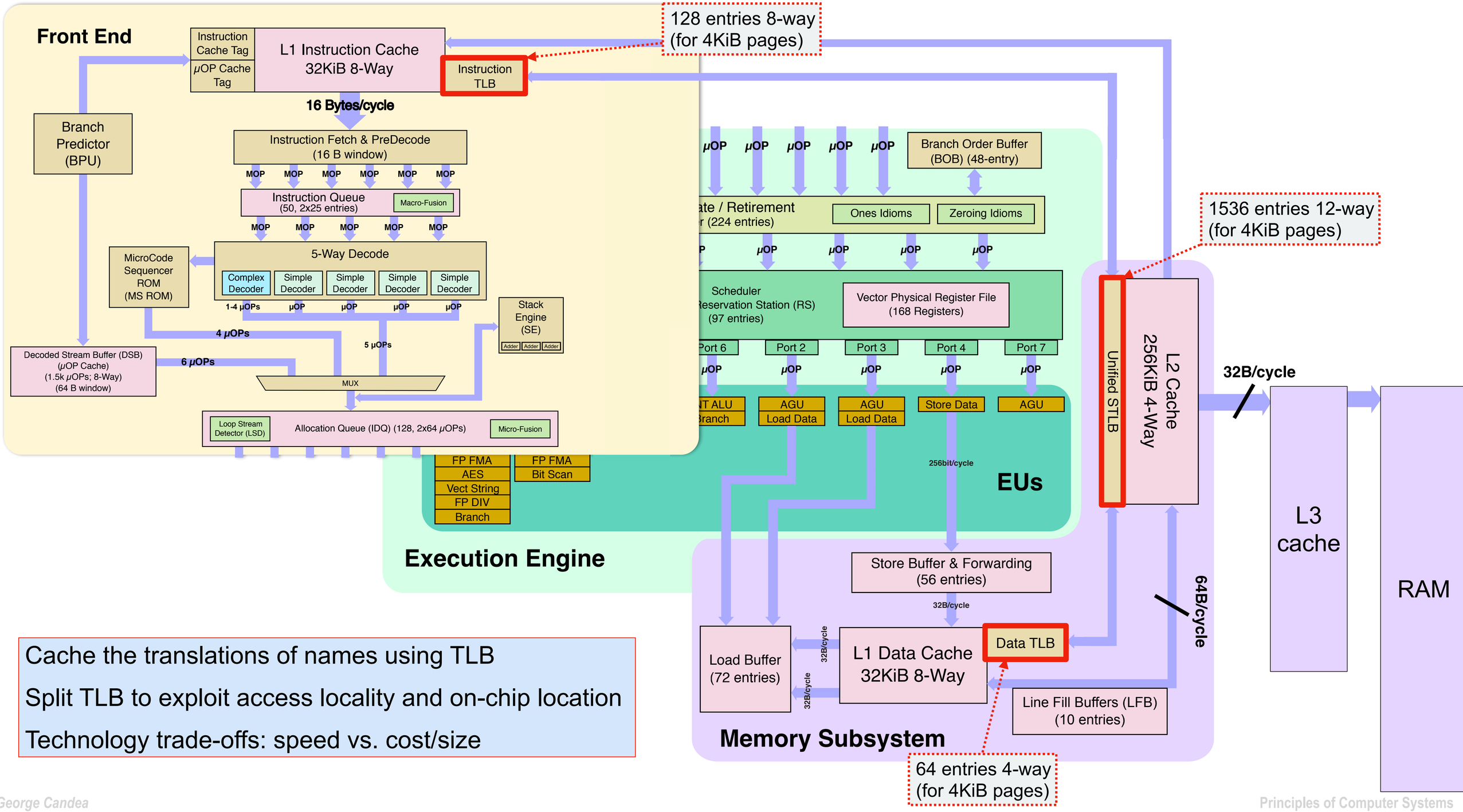


Intel Skylake Microarchitecture

Gen9



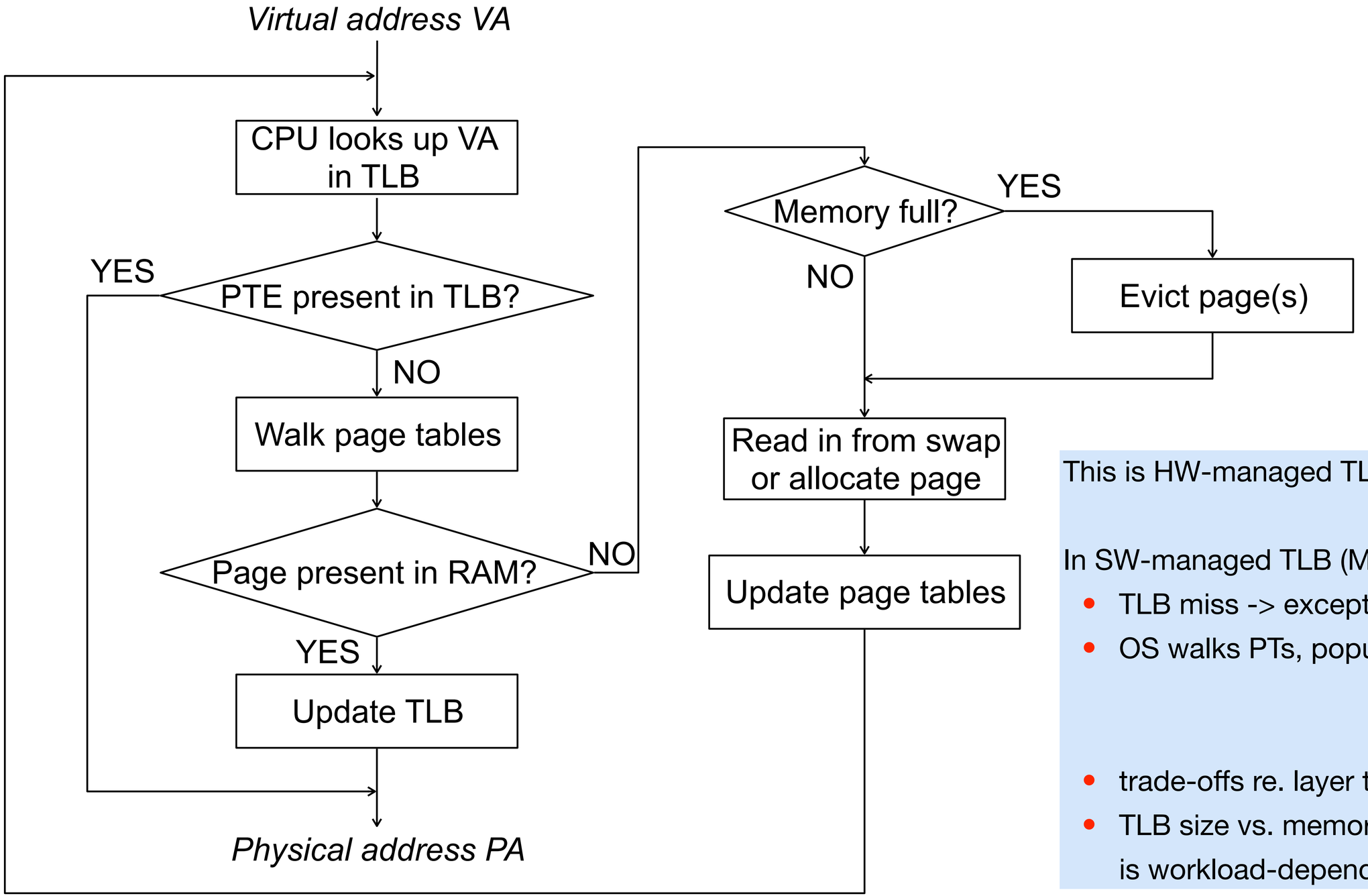

Front End
Instruction Cache Tag
μOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50, 2x25 entries)
Macro-Fusion
64B/cycle
MicroCode Sequencer ROM (MS ROM)
5-Way Decode
Complex Decoder
Simple Decoder
1-4 μOPs
μOP
Stack Engine (SE)
Adder
4 μOPs
5 μOPs
Decoded Stream Buffer (DSB) (μOP Cache) (1.5k μOPs; 8-Way) (64 B window)
6 μOPs
MUX
Loop Stream Detector (LSD)
Allocation Queue (IDQ) (128, 2x64 μOPs)
Micro-Fusion
Register Alias Table (RAT)
4 μOPs
Branch Order Buffer (BOB) (48-entry)
Load
FP
Int Vect
Int
Store
Common Data Buses (CDBs)
Move Elimination
Rename / Allocate / Retirement
ReOrder Buffer (224 entries)
Ones Idioms
Zeroing Idioms
Integer Physical Register File (180 Registers)
Scheduler
Unified Reservation Station (RS)
(97 entries)
Vector Physical Register File (168 Registers)
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
INT ALU
INT DIV
INT Vect ALU
INT Vect MUL
FP FMA
AES
Vect String
FP DIV
Branch
INT MUL
Bit Scan
Vect Shuffle
LEA
AGU
Load Data
Store Data
256bit/cycle
EUs
Execution Engine
Unified STLB
L2 Cache 256KiB 4-Way
32B/cycle
To L3
Store Buffer & Forwarding (56 entries)
Load Buffer (72 entries)
L1 Data Cache 32KiB 8-Way
Data TLB
Line Fill Buffers (LFB) (10 entries)
Memory Subsystem



Cache the translations of names using TLB

Split TLB to exploit access locality and on-chip location

Technology trade-offs: speed vs. cost/size



This is HW-managed TLB !

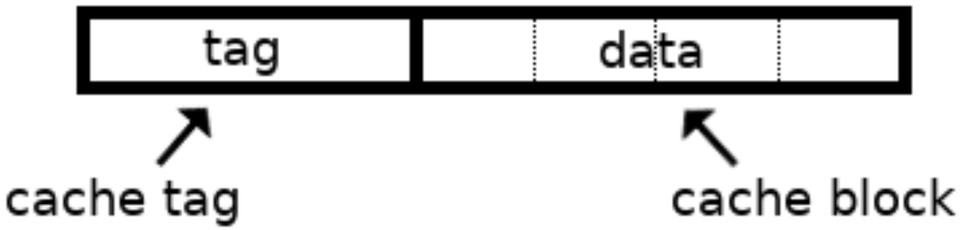
In SW-managed TLB (MIPS, SPARC, …)

- TLB miss -> exception to OS
- OS walks PTs, populates TLB

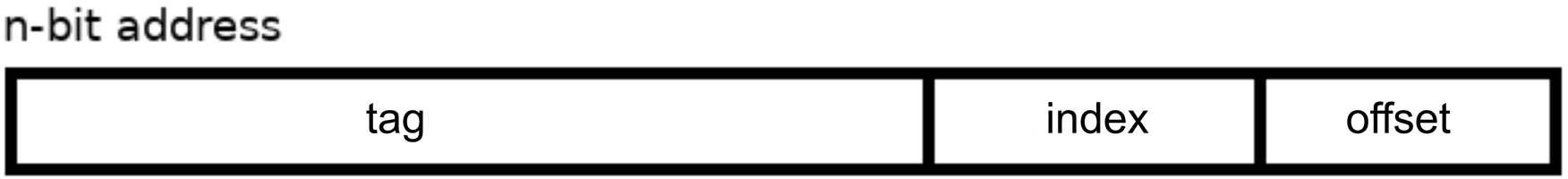
- trade-offs re. layer to delegate to
- TLB size vs. memory size trade-off is workload-dependent => hard!

Cache structure

Cache entry (line)



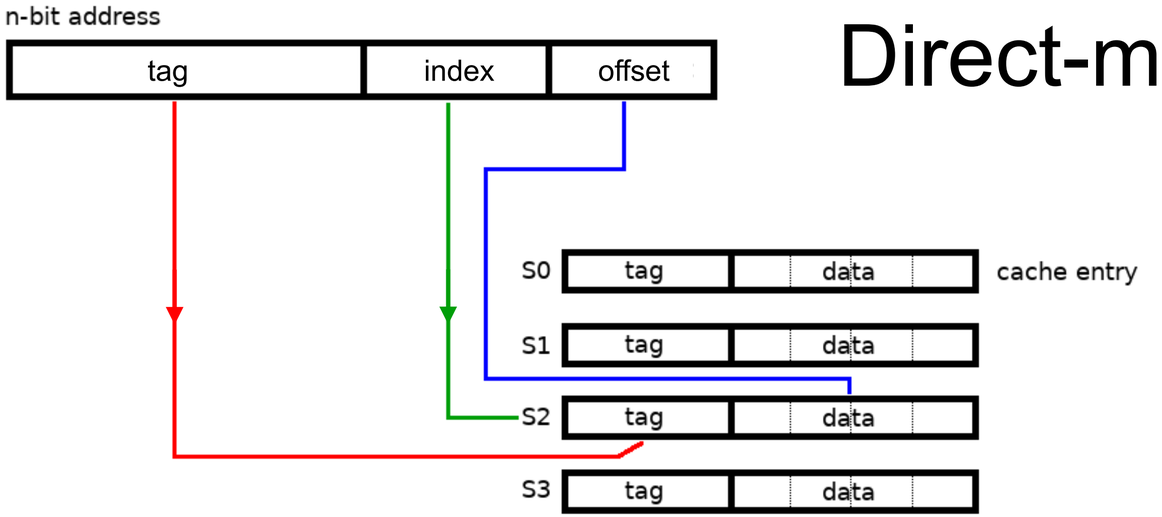
Address (cache lookup key)



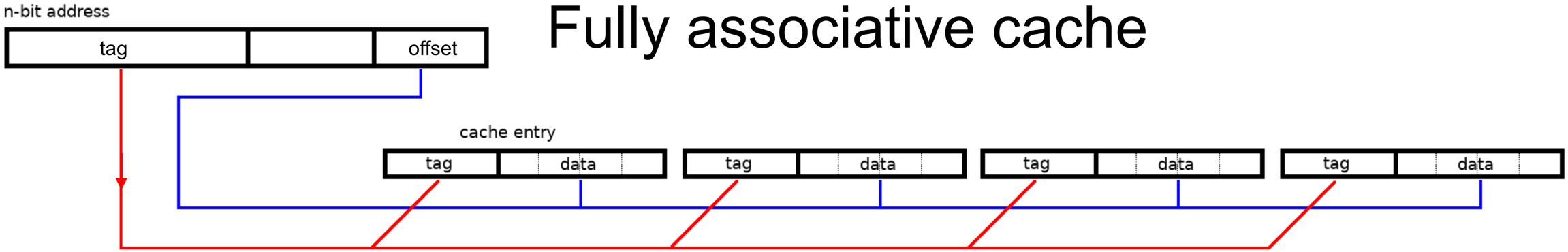
https://www.allaboutcircuits.com/technical-articles/principles-of-cache-design/

Cache structure

Direct-mapped cache

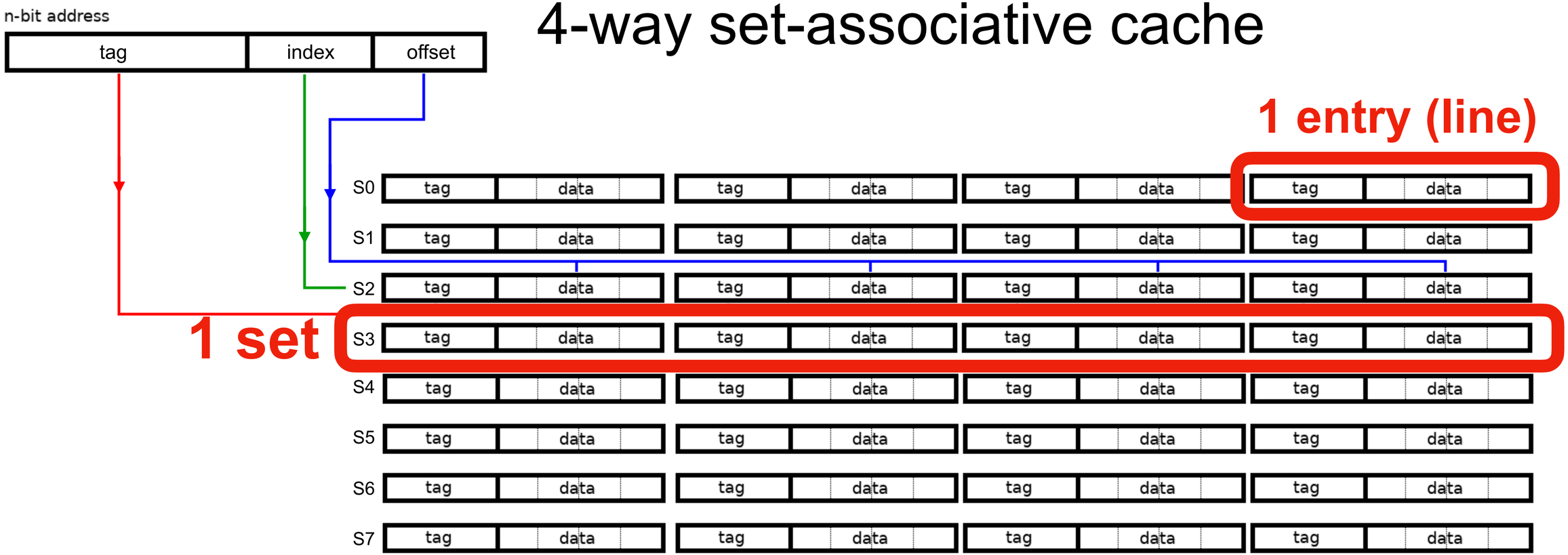


Fully associative cache



https://www.allaboutcircuits.com/technical-articles/principles-of-cache-design/

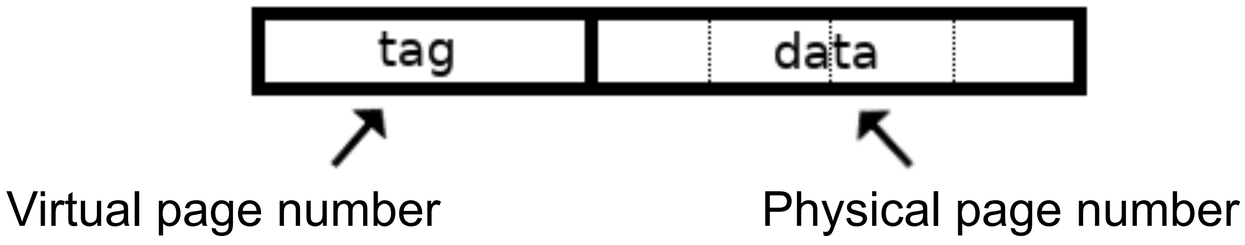
Cache structure



https://www.allaboutcircuits.com/technical-articles/principles-of-cache-design/

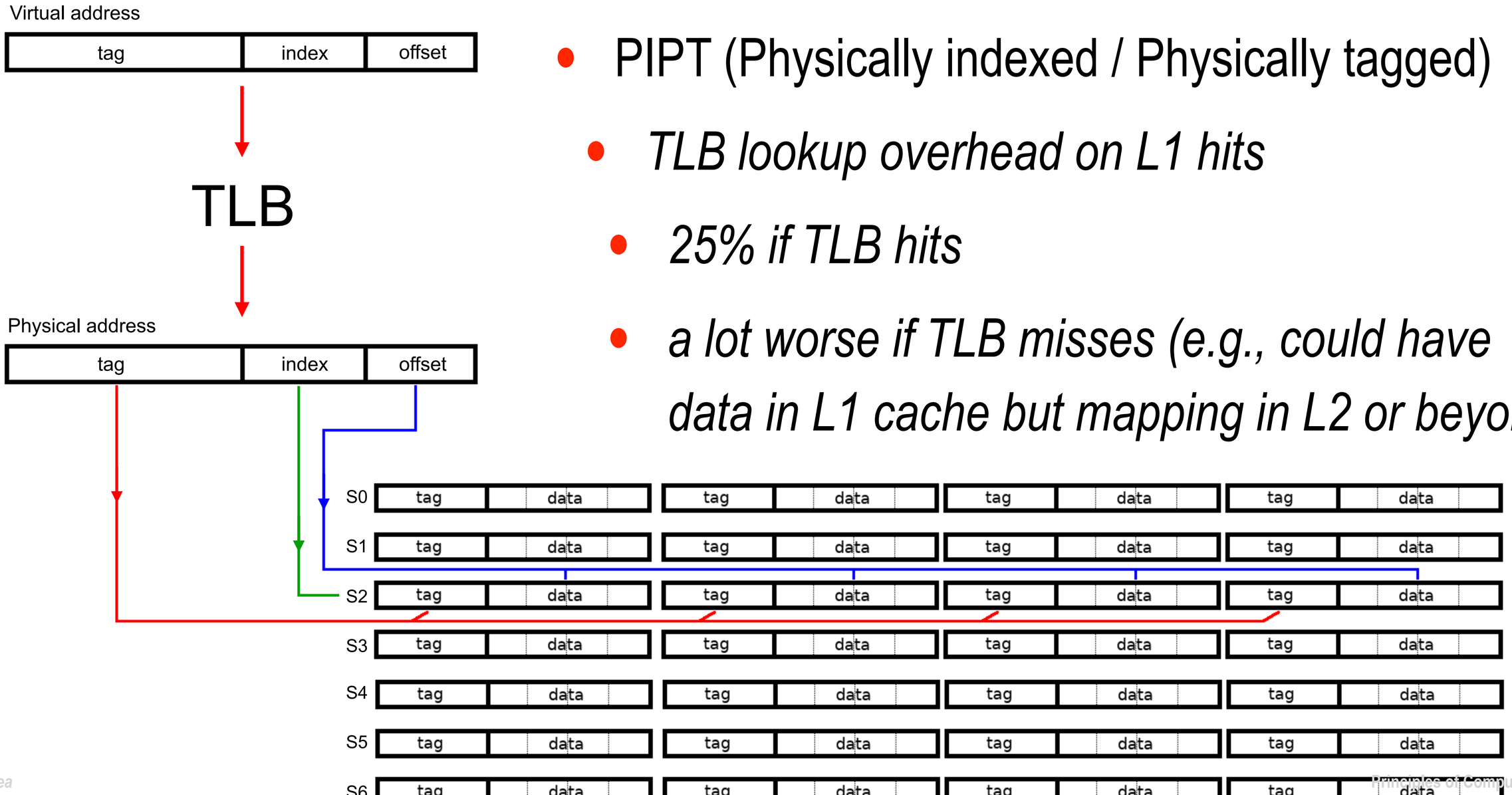
Overhead of memory-location name resolution

TLB cache line



- Skylake (4 KiB pages)
  - *L1 i-TLB: 16 sets * 8-way set-associative = 128 entries*
  - *L1 d-TLB: 16 sets * 4-way set associative = 64 entries*
  - *L2 shared TLB: 256 sets * 12-way set associative = 1536 entries*

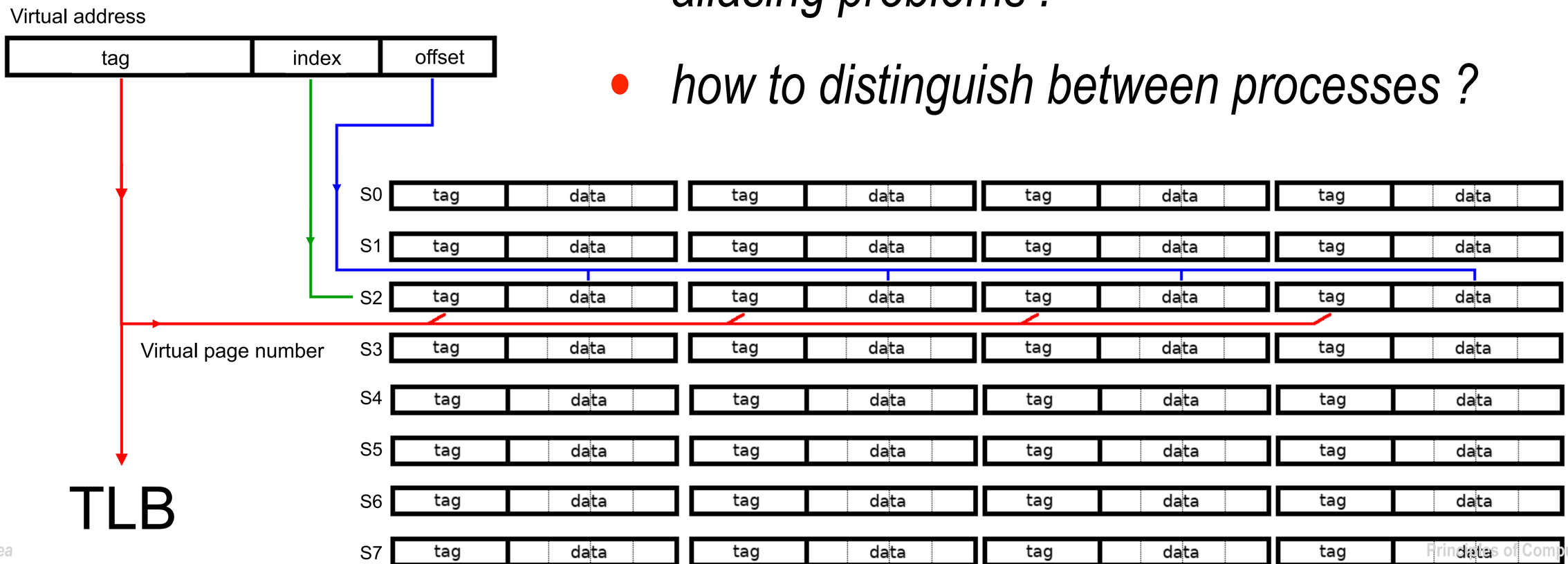
PIPT L1 d/i-cache



- PIPT (Physically indexed / Physically tagged)
  - *TLB lookup overhead on L1 hits*
    - *25% if TLB hits*
    - *a lot worse if TLB misses (e.g., could have data in L1 cache but mapping in L2 or beyond)*

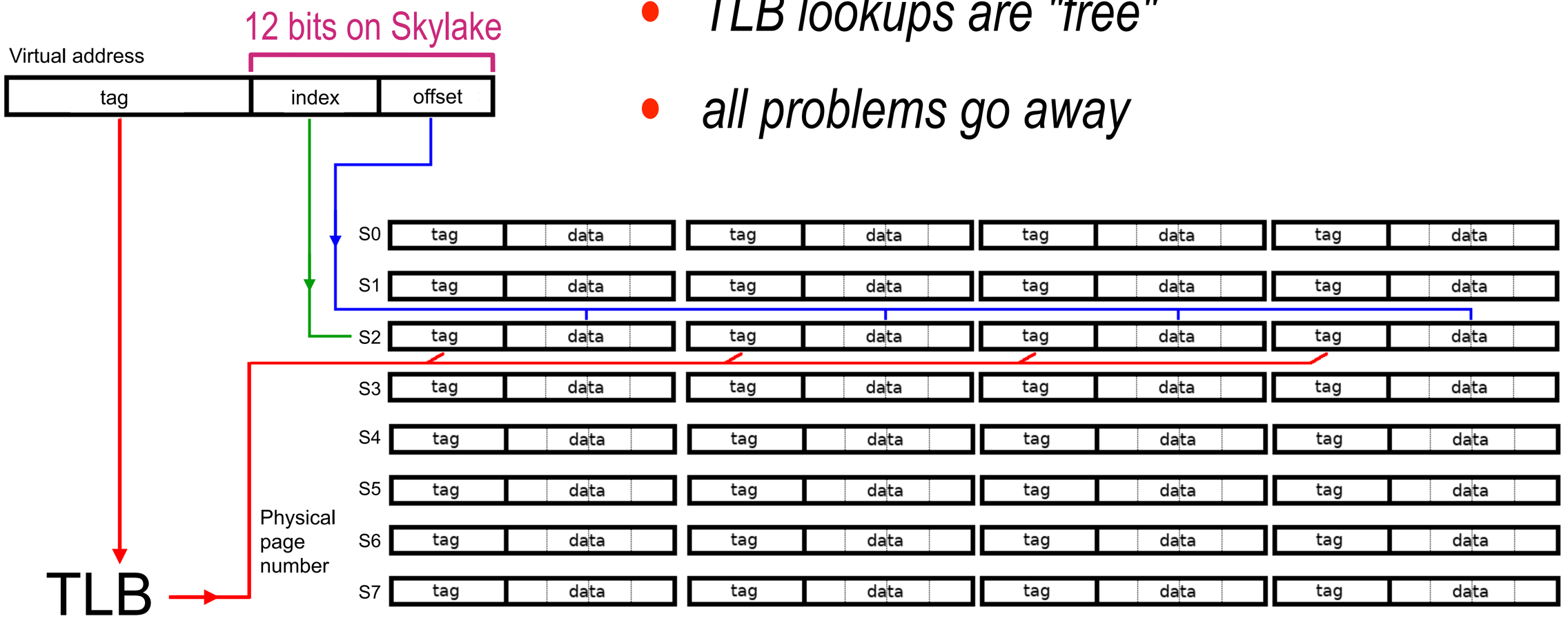
VIVT L1 d/i-cache

- VIVT (Virtually indexed / Virtually tagged)
  - *TLB lookups become free*
  - *aliasing problems !*
  - *how to distinguish between processes ?*



VIPT L1 d/i-cache

- VIPT (Virtually indexed / Physically tagged)
  - *TLB lookups are "free"*
  - *all problems go away*



Zero-cost VA-to-PA resolution

This slide is not about the TLB but about the memory cache (in particular L1, since for L2/L3 the TLB overhead is negligible)

- Types of main memory caches
  - *PIPT (Physically indexed / Physically tagged)*
    - TLB lookup introduces 25% overhead on L1 hits
  - *VIVT (Virtually indexed / Virtually tagged)*
    - Aliasing problems + how to distinguish between processes?
  - *VIPT (Virtually indexed / Physically tagged)*
    - Parallel lookup in TLB and L1 => mask entirely the TLB's (<1 cycle) as part of L1's 3-4 cycles
  - ~~*PIVT (Physically-indexed / Virtually-tagged)*~~
- Caches on x86
  - *L1 is VIPT*
  - *L2 and L3 are PIPT*

Summary

- MMU maps memory VAs (names) to PAs (values)
- Space scalability
  - *use pages and offsets as a way to avoid looking up every single location*
  - *name space is sparse => use hierarchical, multi-level page tables (directories)*
  - *reduce number name lookups by aggregating locations (huge pages)*
- Performance scalability
  - *cache mappings in TLBs*
  - *mask TLB lookup with VIPT memory caches (in the common case)*
- Make common case fast, make rare case merely correct

Reference Values for System Design

Reference Values (absolute time)

- L1 / L2 / main RAM reference ~ 1 / 4 / 40-50 ns
- DRAM bandwidth ~200 GB/sec
- Flash SSD random access 50-100 microsec
- Flash SSD bandwidth 5-10 GB/sec
- Seek on enterprise-grade HDD ~4 millisec
- Bandwidth of enterprise-grade HDD ~150 MB/sec
- Mutex lock or unlock ~ 15-100 ns *(depending on whether share cache line, are contended, are cache-aligned ...)*
- Packet RTT within a datacenter < 10 microsec (ignoring software stacks)
- Bandwidth between two hosts in a data center 10-400 Gbps
- Compress 1 KB of data (with Snappy/LZ77) ~ 2 microsec
- Packet roundtrip CA -> Amsterdam -> CA = 150 millisec (*note: speed of light in fiber ~5 msec/1,000 km)*

Reference Values (CPU cycles)

- Register-register ADD/OR/etc. < 1 CPU cycle *(in an OO, pipelined processor)*
- Register write ~1 cycle
- Correctly / incorrectly predicted "if" branch = 1 cycle / 10-20 cycles
- L1 / L2 / L3 / main RAM read = ~4 cycles / ~2 cycles / ~30-40 cycles / ~100-120 cycles
- TLB hit = 0.5 - 1 cycle
- CAS = 15-30 cycles (increases with the number of cores !)
- C function direct / indirect call = 10-20 cycles / 20-50 cycles
- Kernel crossing round-trip = 1,000 cycles
- Thread context switch (direct costs) = 2,000 cycles
- On NUMA, different-socket mem hierarchy access is 3 - 10x that of non-NUMA