CS211
Advanced Computer Architecture

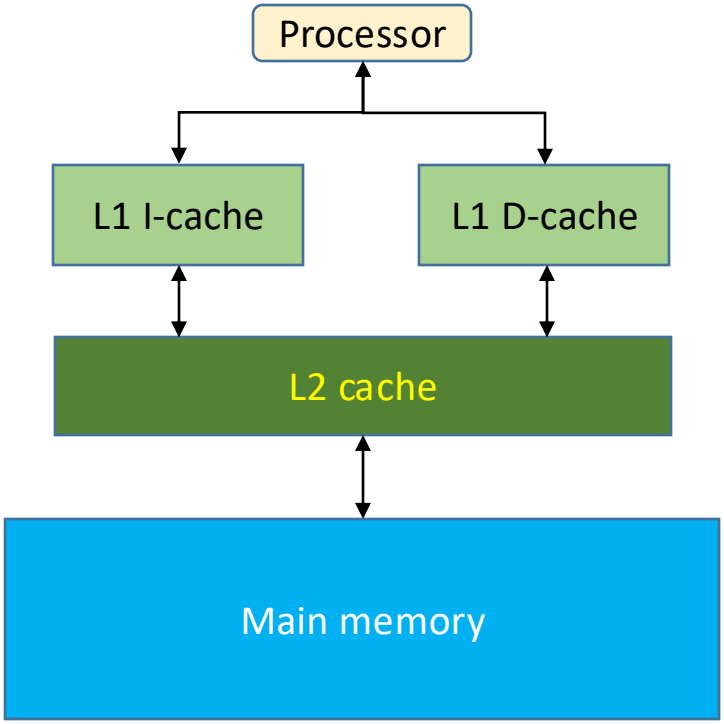
L07 Memory II

Chundong Wang

October 15th, 2025

Case Study: ARM Cortex-A53 Cache Systems

- L1 I-Cache is 8KB to 64 KB, has 64B cache lines, is 2-way set associative, and has a 128-bit read interface to L2
- L1 D-Cache is 8KB to 64 KB, has 64B cache lines, is 4-way set associative, has a 128-bit read interface to L2, and a 256-bit write interface to L2
- L2 Cache is 128KB to 2 MB, has 64B cache lines, and is 16-way set associative
- Both the L1 D cache and L2 use a write-back policy defaulting to allocate on write.
- LRU approximation in all the caches

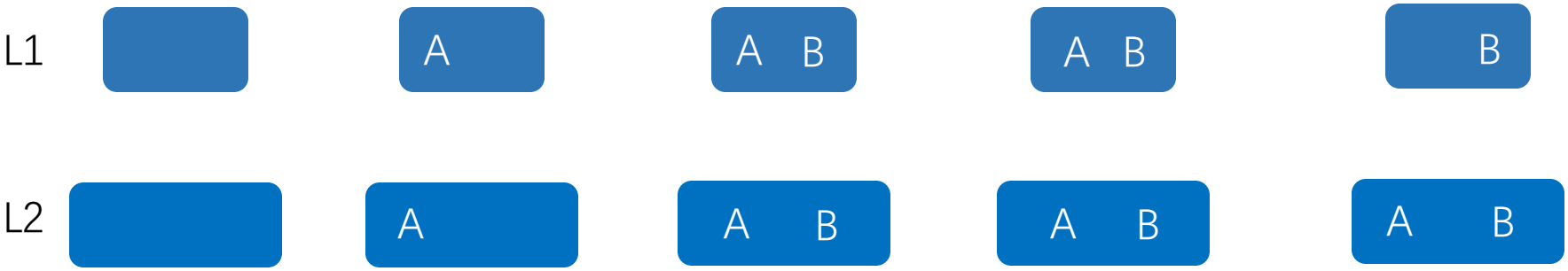


Case Study: Intel Core i7 6700

- L1 I-Cache is 32KB, has 64B cache lines, is 8-way set associative
- L1 D-Cache 32KB, has 64B cache lines, is 8-way set associative
- L1 I-Cache and D-Cache have Pseudo-LRU replacement
- L2 Cache is 256KB, has 64B cache lines, is 4-way set associative
- L2 Cache has Pseudo-LRU replacement
- L3 Cache is 8MB, 2MB per core, has 64B cache lines, is 16-way set associative
- L3 Cache has Pseudo-LRU replacement but with an ordered selection algorithm
  - The block replaced is always the lowest numbered way whose access bit is off

Cache Inclusion Policy

Inclusive



Initial state

Read A miss; load A into L1 and L2

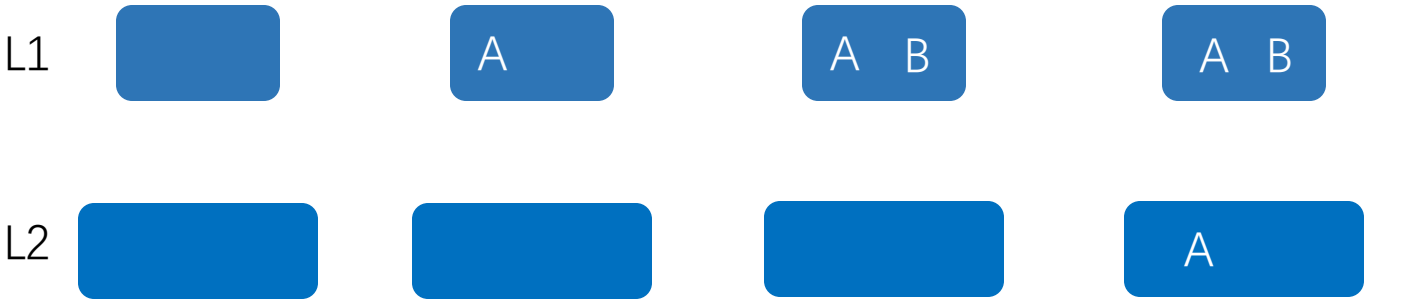
Read B miss; load B into L1 and L2

Evict A from L1 due to cache replacement

Evict B from L2 due to cache replacement

Back invalidation

Exclusive



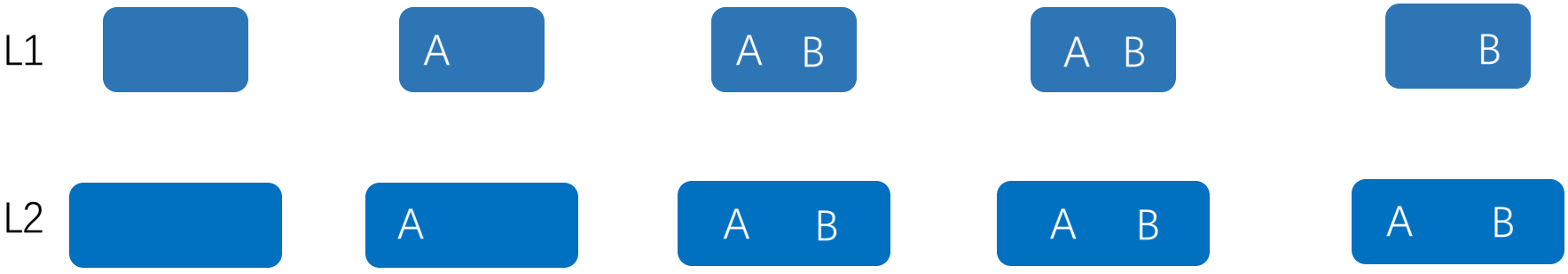
Initial state

Read A miss; load A into L1

Read B miss; load B into L1

Evict A from L1 due to cache replacement and place in L2

Non-inclusive



Initial state

Read A miss; load A into L1 and L2

Read B miss; load B into L1 and L2

Evict A from L1 due to cache replacement

Evict B from L2 due to cache replacement

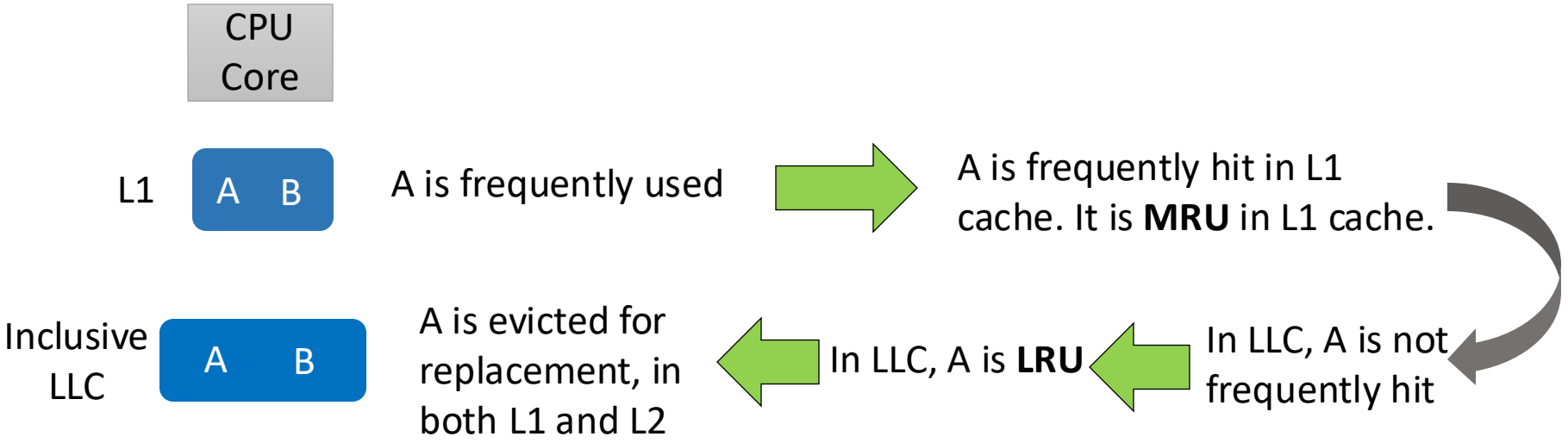
Cache Inclusion Policy

- Multi-level caches are designed depending upon if data in one cache level are also in other cache levels
- Inclusive Policy
  - Same data in all levels
- Exclusive Policy
  - Data in only one cache
- Exclusive policy increases effective amount of caching, but:
  - If data in L2 but not L1, then block is moved from L2 to L1
  - If this causes an eviction from L1, then victim cache block moved to L2
- Non-inclusive policy is a blend of inclusive and exclusive policies

Inclusive, or not?

- Inclusive cache eases coherence
  - Updating a cache block in L1 entails an update in inclusive LLC.
  - A non-inclusive LLC, say L2 cache, which needs to evict a block, **must** ask L1 cache if it has the block, because such information is not present in LLC.
- Non-inclusive cache yields higher performance though, why?
  - No back invalidation
  - More data can be cached

'*Sneaky*' LRU for Inclusive Cache



As a result, MRU block that should be retained might be evicted, which causes performance penalty.

What if LLC is non-inclusive?

Link: https://doi.org/10.1109/MICRO.2010.52

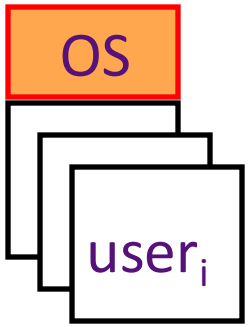
Main Memory

Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy

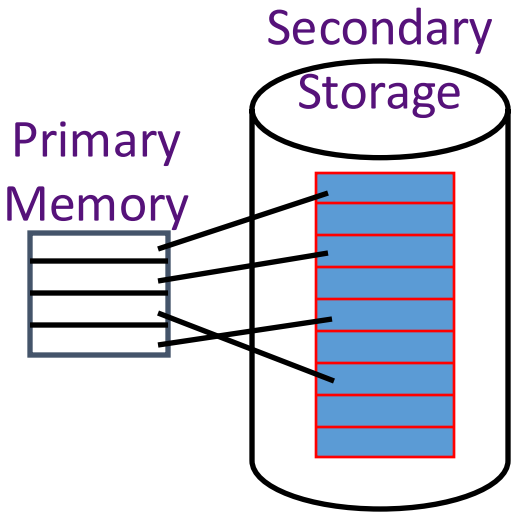
several users, each with their private address space and one or more shared address spaces



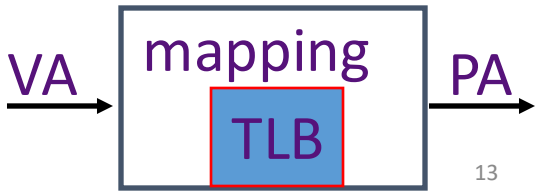
Demand Paging

Provides the ability to run programs larger than the primary memory

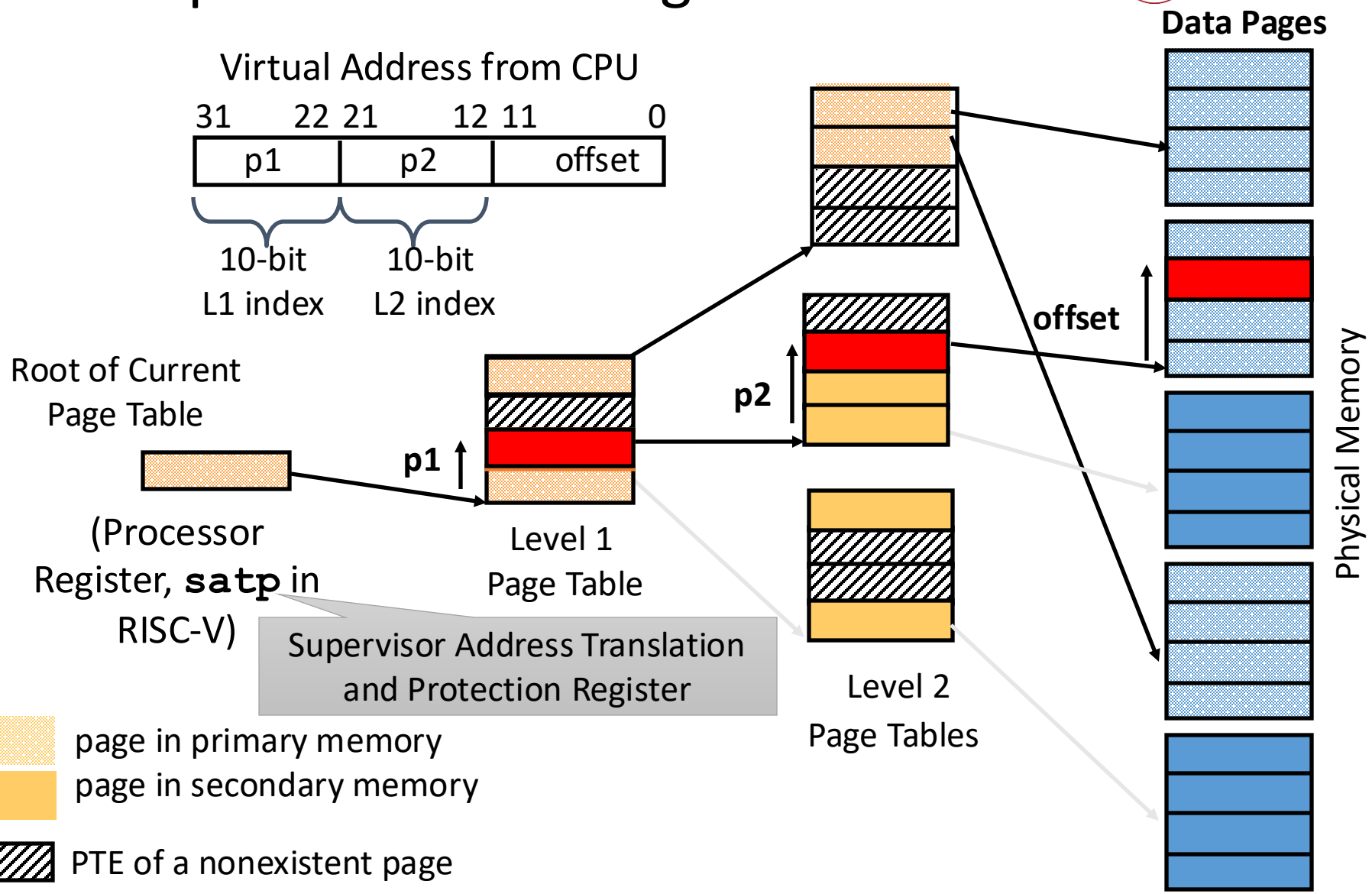
Hides differences in machine configurations



The price is address translation on each memory reference

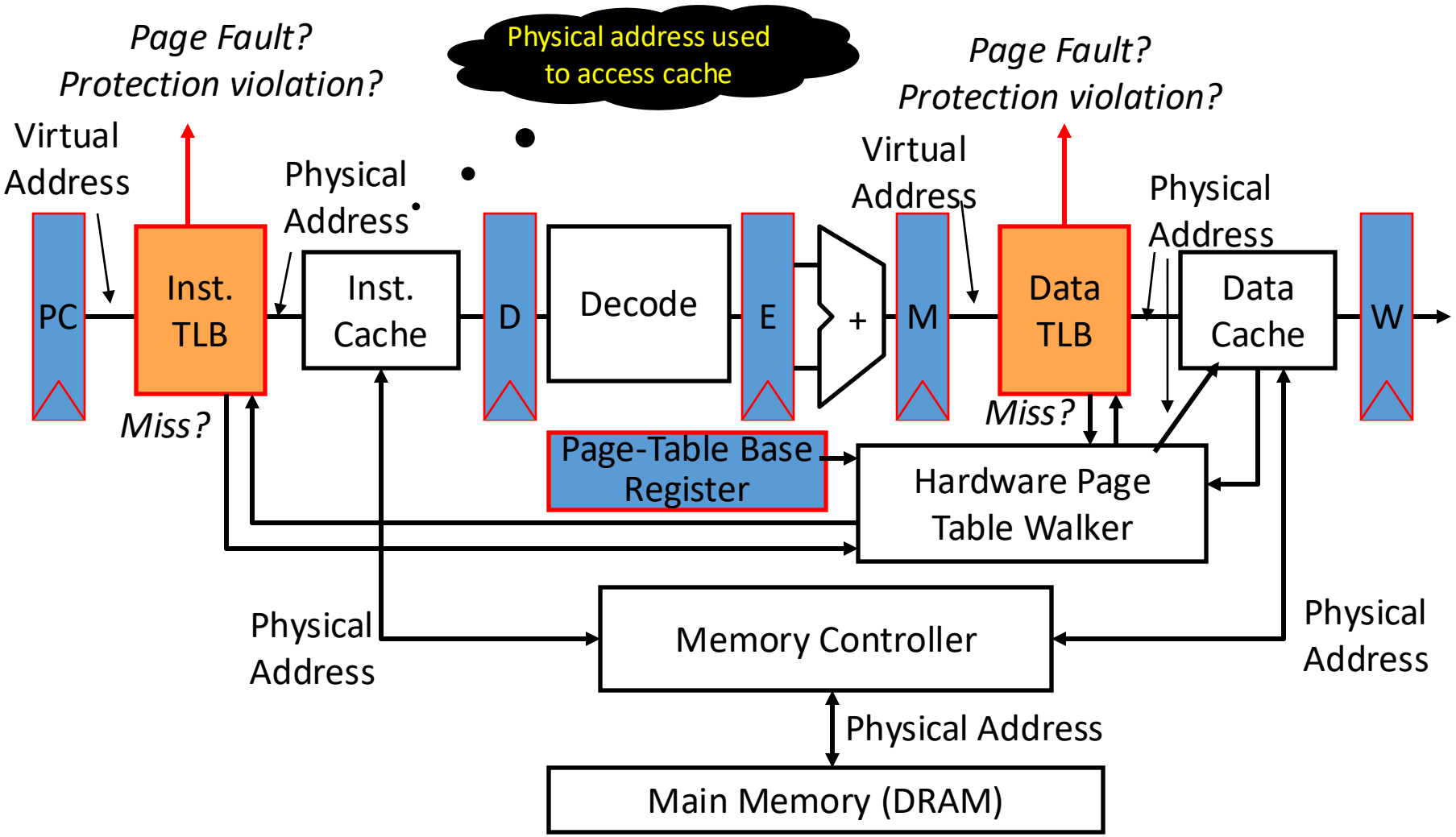


Recap: Hierarchical Page Table



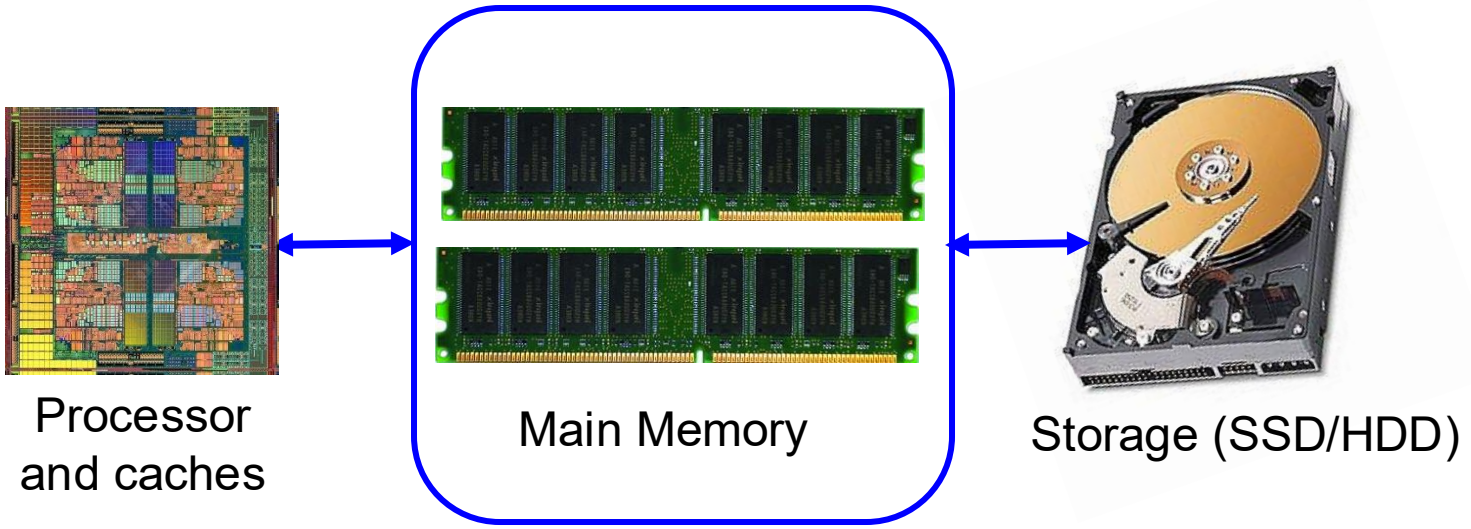
RISC-V (RV32) Sv32 Virtual Memory Scheme

Page-Based Virtual-Memory Machine
(Hardware Page-Table Walk)

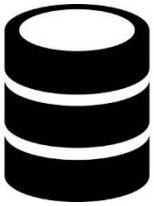


- Assumes page tables held in untranslated physical memory

The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

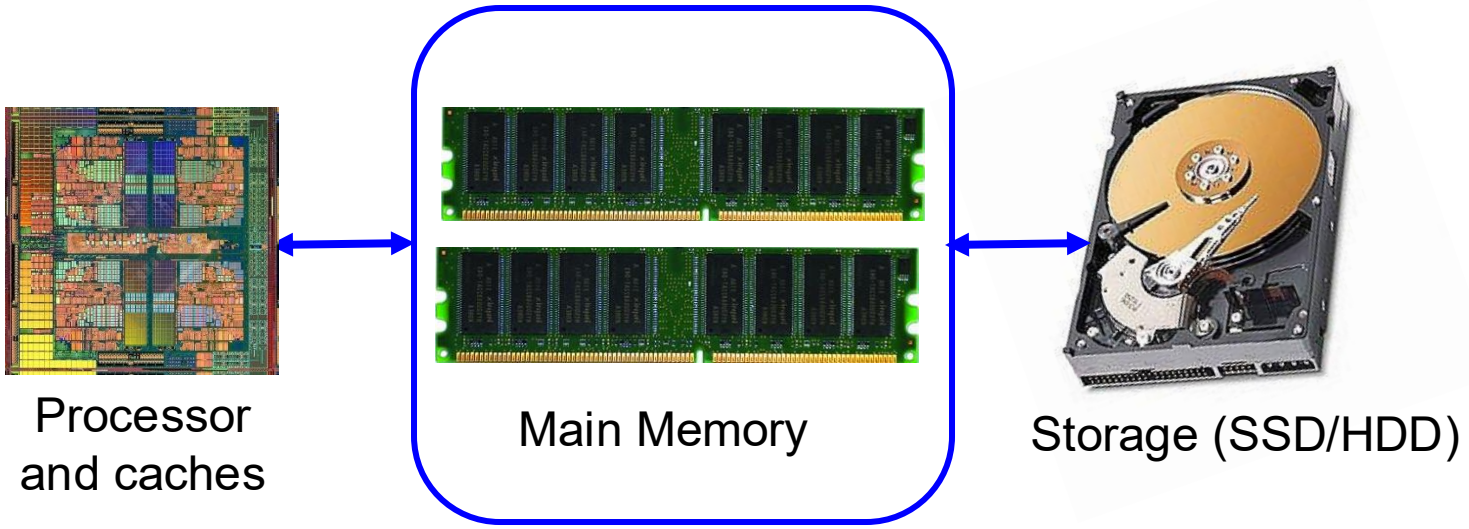

In-memory database


Social networks

In-memory analytics


Data centers

The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Main memory system must scale (in *size*, *technology*, *efficiency*, *cost*, and *management algorithms*) to maintain performance growth and technology scaling benefits

Main Memory

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- How Can We Do Better?

Major Trends Affecting Main Memory (II)

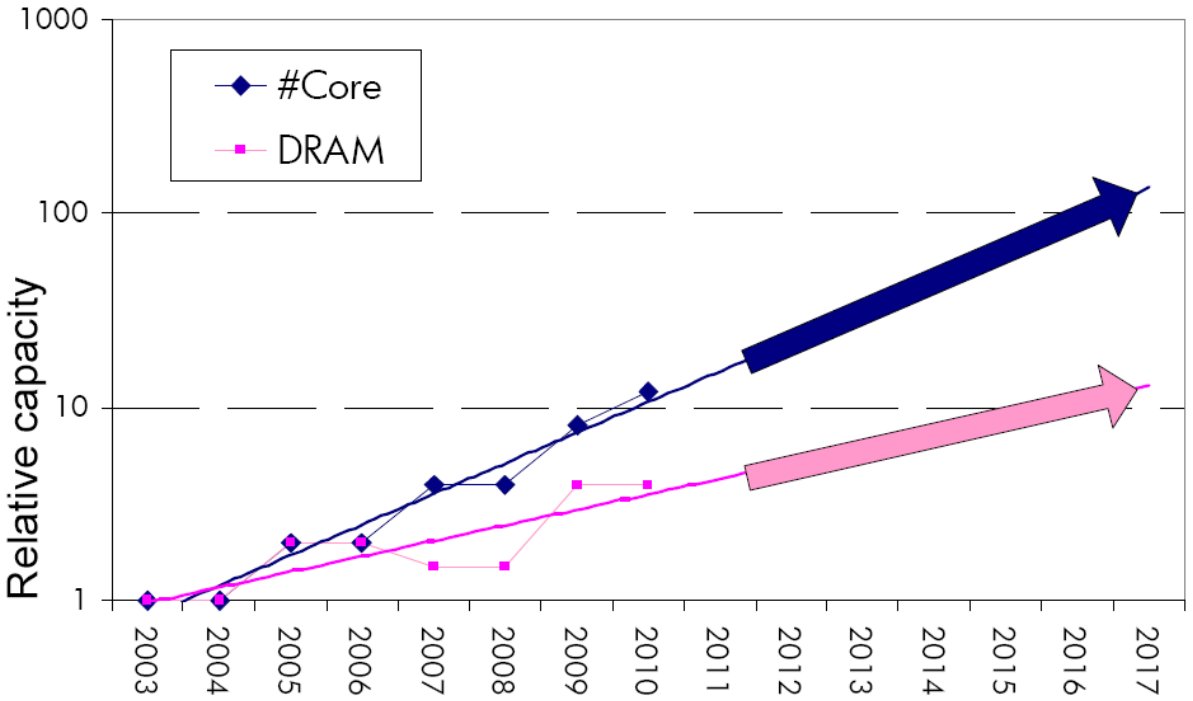
- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile, heterogeneity

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending

Example: The Memory Capacity Gap

Core count doubling ~ every 2 years
DRAM DIMM capacity doubling ~ every 3 years



- *Memory capacity per core* expected to drop by 30% every two years
- Trends worse for *memory bandwidth per core*!

Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
  - DRAM consumes power even when not used (periodic refresh)

- DRAM technology scaling is ending

Major Trends Affecting Main Memory (IV)

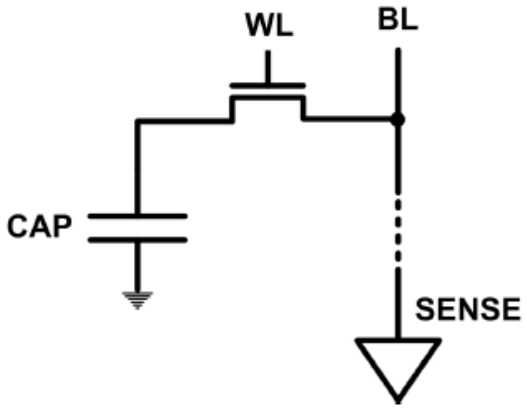
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

Main Memory

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- How Can We Do Better?

The DRAM Scaling Problem

- DRAM stores charge in a capacitor (charge-based memory)
  - Capacitor must be large enough for reliable sensing
  - Access transistor should be large enough for low leakage and high retention time
  - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]



- DRAM capacity, cost, and energy/power hard to scale

Solution 1: Fix DRAM

- Overcome DRAM shortcomings with
  - System-DRAM co-design
  - Novel DRAM architectures, interface, functions
  - Better waste management (efficient utilization)

- Key issues to tackle
  - Enable reliability at low cost
  - Reduce energy
  - Improve latency and bandwidth
  - Reduce waste (capacity, bandwidth, latency)
  - Enable computation close to data

Solution 1: Fix DRAM

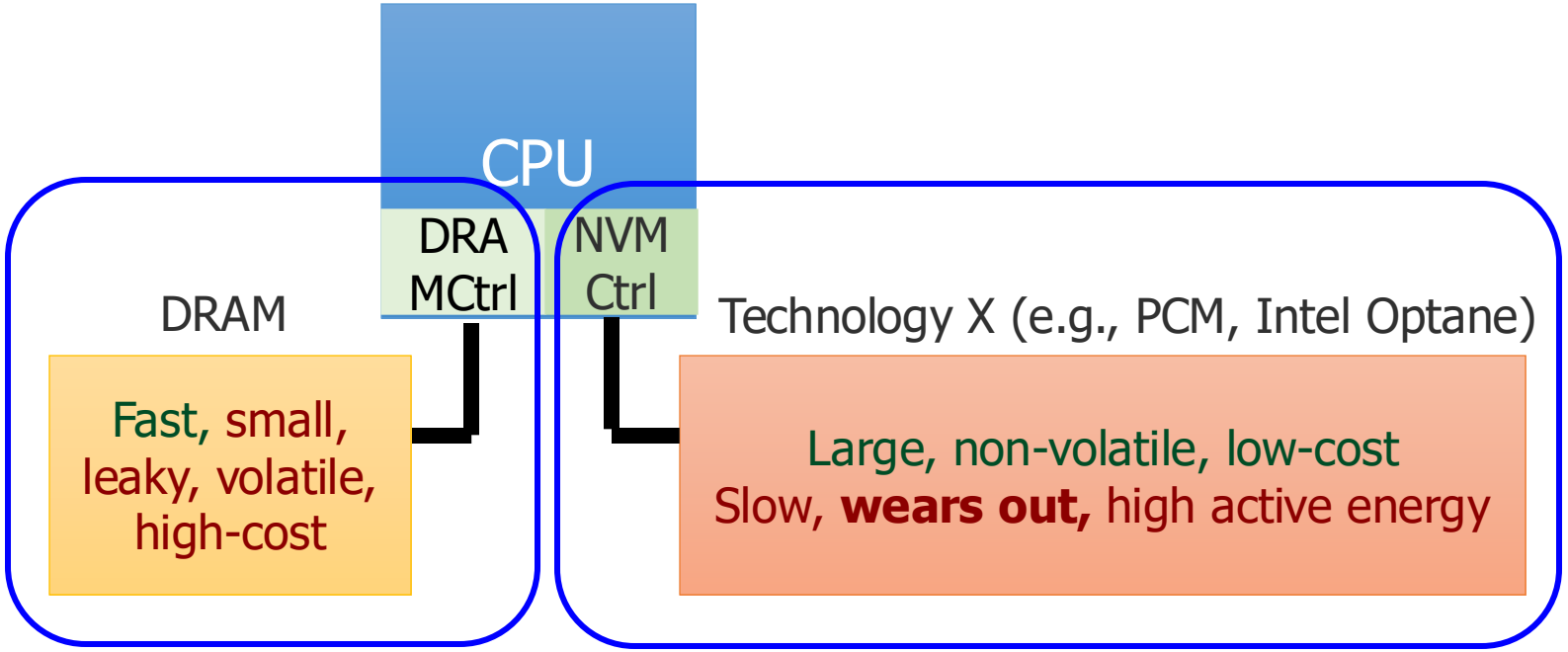
- Liu+, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," ISCA 2013.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.
- Pekhimenko+, "Linearly Compressed Pages: A Main Memory Compression Framework," MICRO 2013.
- Chang+, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," HPCA 2014.
- Khan+, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS 2014.
- Luo+, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost," DSN 2014.
- Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.
- Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.
- Qureshi+, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," DSN 2015.
- Meza+, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," DSN 2015.
- Kim+, "Ramulator: A Fast and Extensible DRAM Simulator," IEEE CAL 2015.
- Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM," IEEE CAL 2015.
- Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," ISCA 2015.
- Ahn+ "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," ISCA 2015.
- Avoid DRAM:
  - Seshadri+, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," PACT 2012.
  - Pekhimenko+, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," PACT 2012.
  - Seshadri+, "The Dirty-Block Index," ISCA 2014.
  - Pekhimenko+, "Exploiting Compressed Block Size as an Indicator of Future Reuse," HPCA 2015.
  - Vijaykumar+, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," ISCA 2015.

Solution 2: Emerging Memory Technologies

- Some emerging resistive memory technologies seem more scalable than DRAM (and they are non-volatile)
- Example 1: Phase Change Memory
  - Expected to scale to 9nm (2022 [ITRS])
  - Expected to be denser than DRAM: can store multiple bits/cell
- Example 2: Intel Optane DC Memory
  - ~~Commercially available~~, in terabytes
  - **Ceased recently**
- But, emerging technologies have shortcomings as well
  - Can they be enabled to replace/augment/surpass DRAM?

- Lee+, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA'09, CACM'10, Micro'10.
- Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters 2012.
- Yoon, Meza+, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012.
- Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.
- Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.
- Lu+, "Loose Ordering Consistency for Persistent Memory," ICCD 2014.
- Zhao+, "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," MICRO 2014.
- Yoon, Meza+, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," ACM TACO 2014.
- … …

Solution 3: Hybrid Memory Systems



Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Yoon, Meza et al., "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

… …

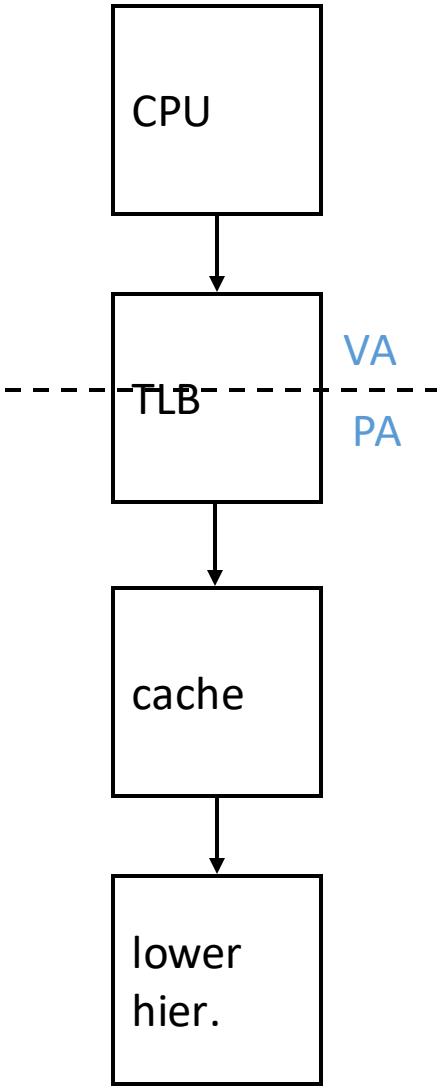
Some Promising Directions

- New memory architectures
  - Rethinking DRAM
  - **A lot of hope in fixing DRAM**

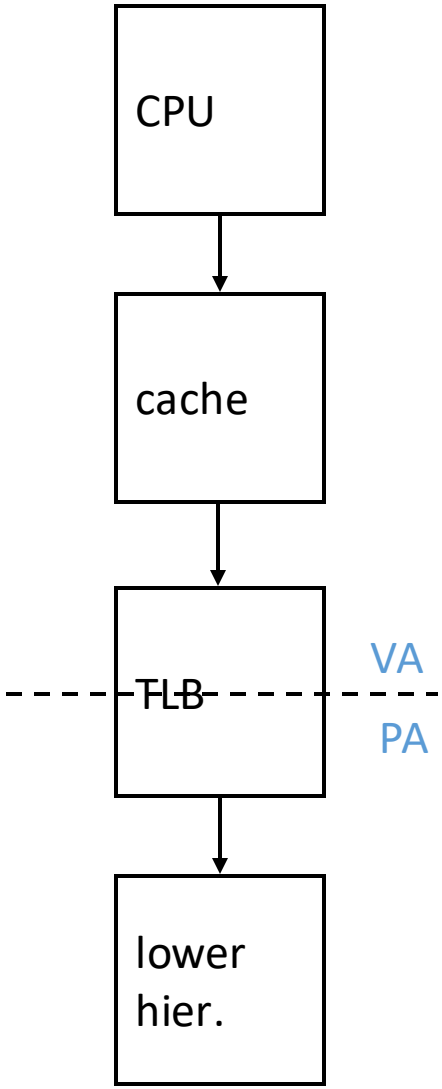
- Enabling emerging NVM technologies
  - Hybrid memory systems
  - Single-level memory and storage
  - **A lot of hope in hybrid memory systems and single-level stores**

Virtual Memory and Cache Interaction

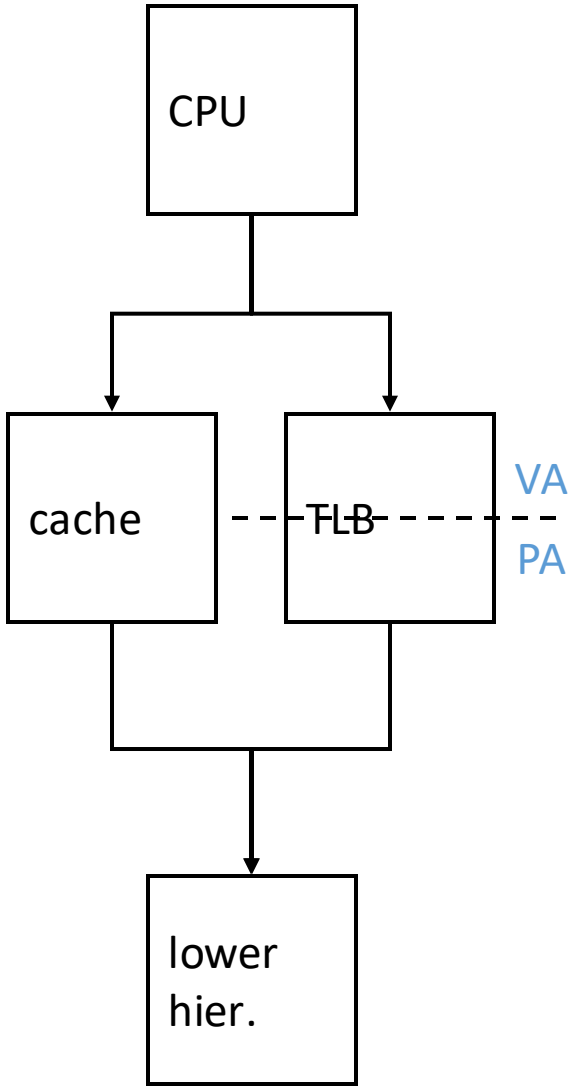
Cache-VM Interaction



physical cache

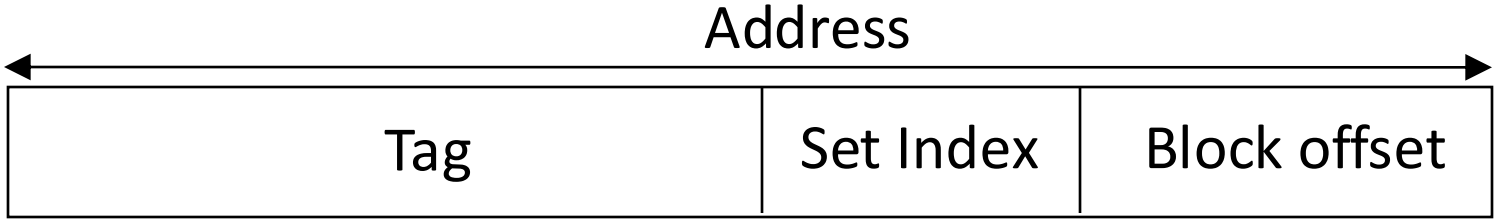


virtual (L1) cache



virtual-physical cache

Address Translation and Caching



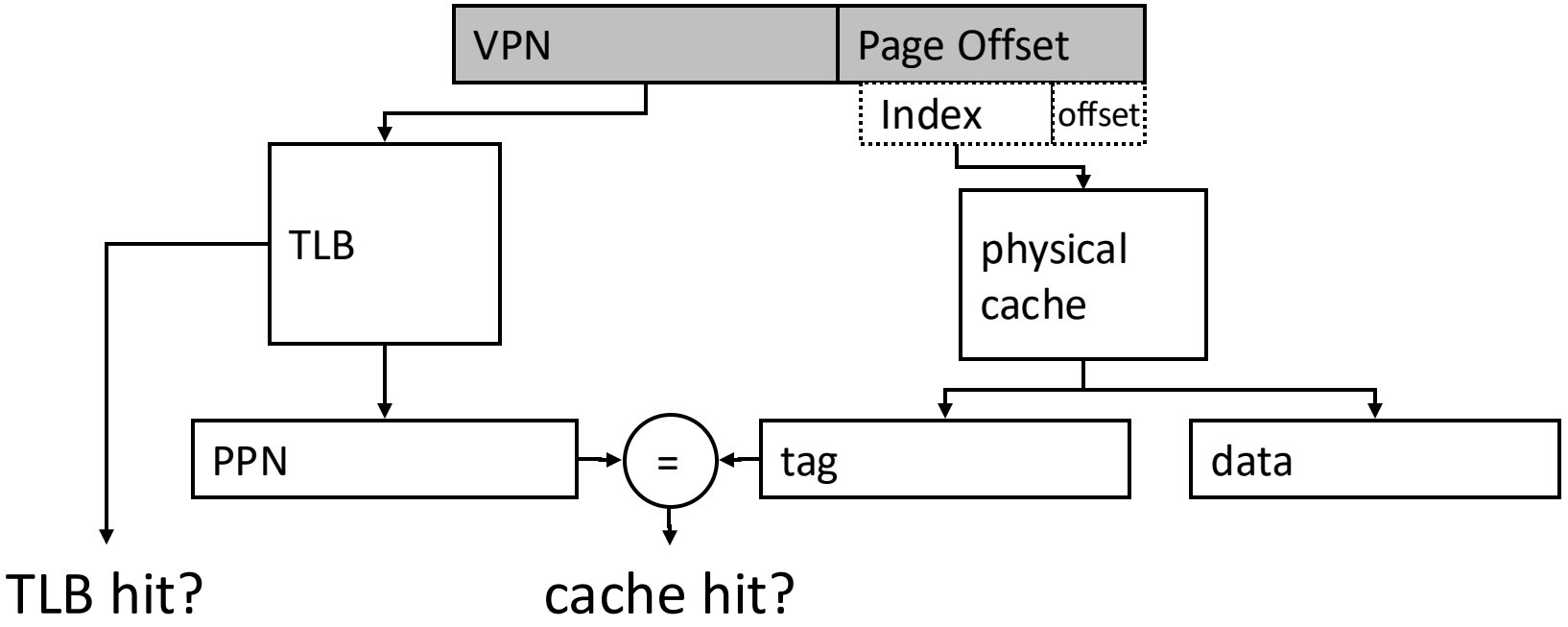
- When do we do the address translation?
  - Before or after accessing the L1 cache?
- In other words, is the cache virtually addressed or physically addressed?
  - Virtual versus physical cache
- What are the issues with a virtually addressed cache?
- Synonym problem:
  - Two different virtual addresses can map to the same physical address → same physical address can be present in multiple locations in the cache → can lead to inconsistency in data

Homonyms and Synonyms

- Homonym: Same VA can map to two different PAs
  - Why?
    - VA is in different processes
- Synonym: Different VAs can map to the same PA
  - Why?
    - Different pages can share the same physical frame within or across processes
    - Reasons: shared libraries, shared data, copy-on-write pages within the same process, …
- Do homonyms and synonyms create problems when we have a cache?
  - Is the cache virtually or physically addressed?

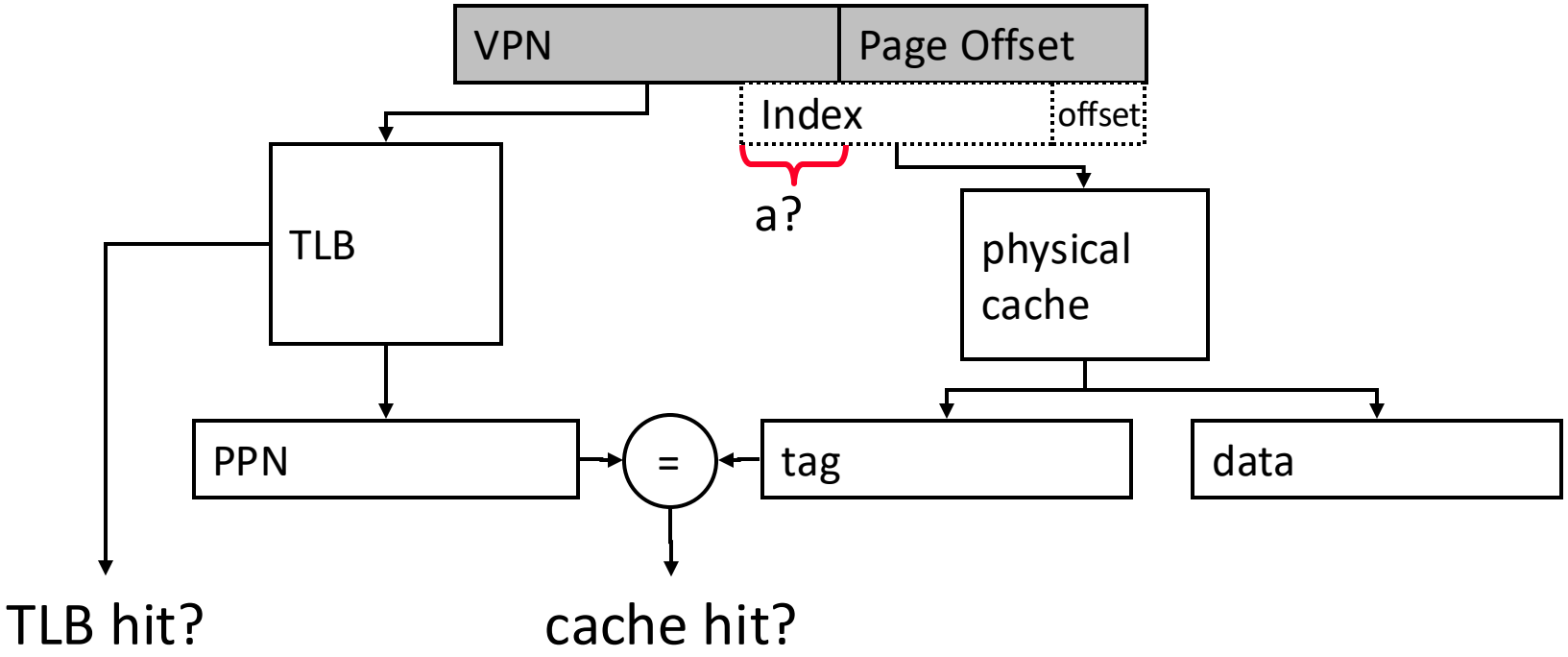
Virtually-Indexed Physically-Tagged

- If C≤(page_size × associativity), the cache index bits come only from page offset (same in VA and PA)
- If both cache and TLB are on chip
  - index both arrays concurrently using VA bits
  - check cache tag (physical) against TLB output at the end



Virtually-Indexed Physically-Tagged

- If C>(page_size × associativity), the cache index bits include VPN ⇒ Synonyms can cause problems
  - Different VAs mapped to the same physical address
  - The same physical address can exist in two locations
- Solutions?



Some Solutions to the Synonym Problem

- Limit cache size to (page size × associativity)
  - get index from page offset
- On a write to a block, search all possible indices that can contain the same physical block, and update/invalidate
  - Used in Alpha 21264, MIPS R10K
- Restrict page placement in OS
  - make sure index(VA) = index(PA)
  - Called page coloring
  - Used in many SPARC processors

PIPT and VIVT

- Physically indexed, physically tagged (PIPT) caches use the physical address for both the index and the tag.
  - Simple to implement but slow, as the physical address must be looked up (which could involve a TLB miss and access to main memory) before that address can be looked up in the cache.
- Virtually indexed, virtually tagged (VIVT) caches use the virtual address for both the index and the tag.
  - Potentially much faster lookups.
  - Problems when several different virtual addresses may refer to the same physical address
    - Addresses would be cached separately despite referring to the same memory, causing coherency problems.
  - Additionally, there is a problem that virtual-to-physical mappings can change, which would require clearing cache blocks
- Can we have PIVT?

Conclusion

- Case studies for cache
- Cache inclusion
- Main memory
- Interaction between cache and memory

Acknowledgements

- These slides contain materials developed and copyright by:
  - Prof. Onur Mutlu (ETH Zurich)
  - Prof. Zhi Wang (FSU)
  - Prof. Jason Tang (UMBC)
  - Prof. Krste Asanović (UC Berkeley)