TLB Design and Management Techniques

Sparsh Mittal

IIT Hyderabad, India

TLB = Translation Lookaside Buffer

Acronyms

- VA/PA = virtual/physical address
- HW/SW = hardware/software
- ITLB/DTLB = instruction/data TLB
- PTE = page table entry
- PTW = page table walker
- Reg = register
- VM = virtual memory
- ASID = address space identifier
- MMU = memory management unit

A Background on TLB

- To leverage memory access locality in VA to PA translation, processors with page-based virtual memory use TLB
- TLB is a cache for page table
- TLB caches VA to PA translations

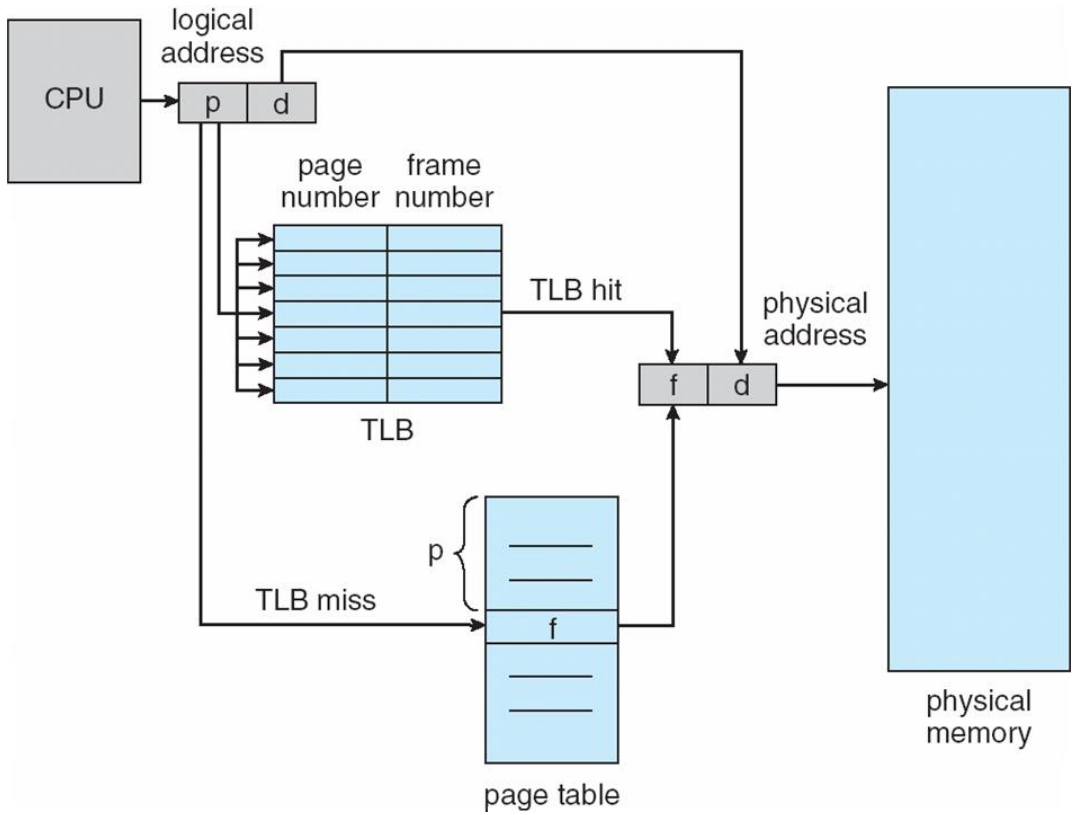
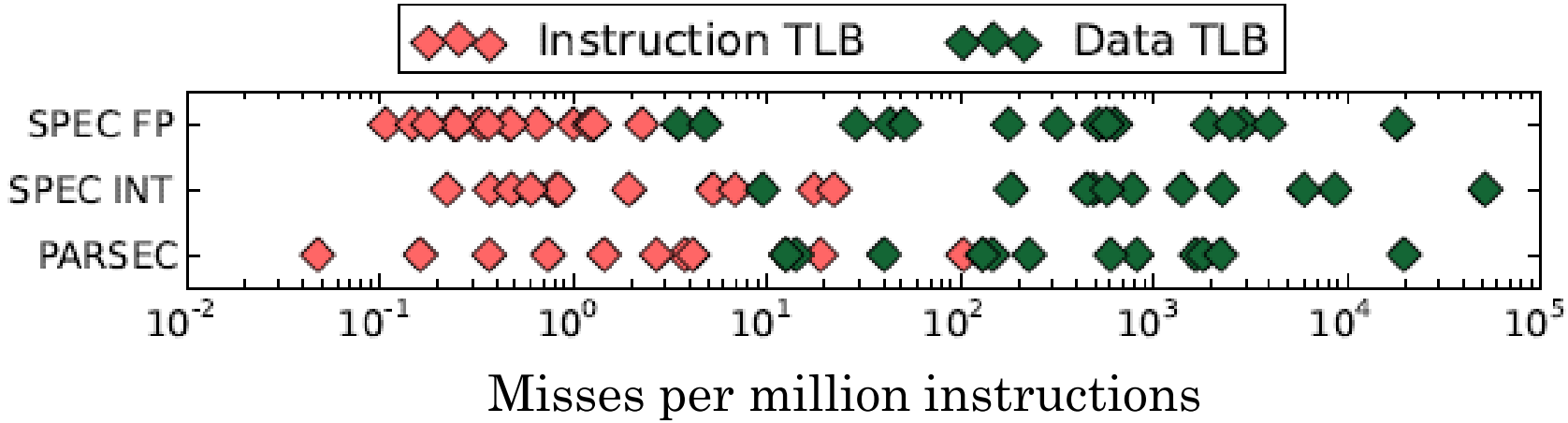


Figure credit: Silberschatz, Galvin and Gagne

Data/instruction TLB and L1/L2 TLB

- DTLB and ITLB may be unified or separate
- DTLB miss-rate >> ITLB miss-rate



- TLB can be single or multi-level, similar to cache.
- First level = L1 TLB, second level = L2 TLB

Lustig et al. TACO'13

Motivation for intelligent management of TLB

- On a TLB miss, page table needs to be accessed ➔ costly (e.g., 4 memory accesses in a 4-level page table)
- TLB miss handling may take up to 50% of execution time
- Need to choose TLB parameters carefully: TLB size, associativity, number of levels (one or two), superpage or multiple page-sizes, etc.

Useful terms and concepts

Useful terms

- **TLB Coverage** (or reach or mapping size): sum of memory mapped by all TLB entries
- E.g., page size = 4KB, # entries = 1024 ➔ coverage = 1024*4KB
- **Superpage:**
  - A VM page with size = 2^k times system page size and which maps to contiguous physical pages
  - Increases coverage by using one entry for multiple virtual pages
  - Useful for mapping large objects, e.g., big arrays, kernel data
  - Helpful in increasing working set size of applications

TLB Shootdown and HW/SW-managed TLB

- **TLB shootdown:** Invalidating those TLB entries whose translation mapping has changed on a change in VA-to-PA mapping (e.g., due to page swaps).

- **HW and SW-managed TLB:**
- TLB hit/miss determination is always done in hardware.
- TLB miss handling may be done in HW or SW

TLB in real processors

TLB architecture in Older Processors

Processor	Entries	Assoc.	Year
21064	32I + 32D	fully	1992
Super Sparc	64U	fully	1992
R4000	96U	fully	1992
PA-7100	120U	fully	1992
PowerPC 601	256U	2 way	1993
R8000 (TFP)	384U	3 way	1994
PowerPC 620	64I + 64D	fully	1995
21164	48I + 64D	fully	1995
R10000	8I + 64D	fully	1996
Strong ARM	32I+ 32D	fully	1996

I= instruction, D = data, U = unified

Juan et al. ISLPED'97

D-TLB Architecture in Recent Processors

Processor	L1 TLB Configuration	L2 TLB Configuration
Intel Haswell [5]	4-way SA split L1 TLBs: 64-entry (4KB), 32-entry (2MB) and 4-entry (1GB)	8-way SA 1024-entry (4KB and 2MB)
AMD 12h family [6]	48-entry FA TLB (all page sizes)	4-way SA 1024-entry TLB (4KB) 2-way SA 128-entry TLB (2MB) 8-way SA 16-entry TLB (1GB)
Sparc T4 [3]	128-entry FA TLB (all page sizes)	
UltraSparc III	2-way SA 512-entry TLB (8KB) 16-entry FA TLB (superpages and locked 8KB)	

Papadopoulou et al. HPCA'15

Private, per-core, data TLB hierarchy in 3 recent Intel processors

L1 DTLBs

Sandy Bridge / Haswell / Broadwell		
Page-size	Entries	Assoc.
4 KB	64	4-way
2 MB	32	4-way
1 GB	4	fully

L2 DTLBs

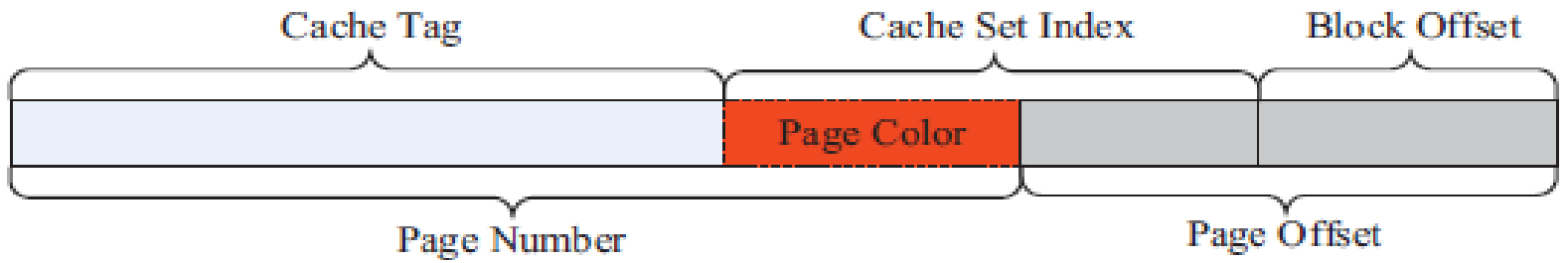
Sandy Bridge			Haswell			Broadwell		
Page-size	Entries	Assoc.	Page-size	Entries	Assoc.	Page-size	Entries	Assoc.
4 KB	512	4-way	4 KB/2 MB	1024	8-way	4 KB/2 MB	1536	n/a
2 MB	—							
1 GB	—		1 GB	—		1 GB	16	n/a

Karakostas et al. HPCA'16

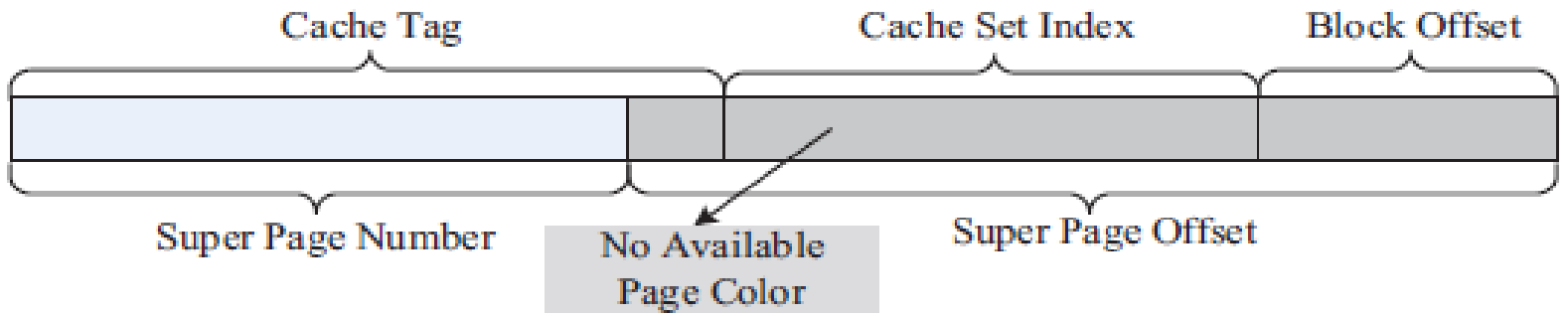
Challenges in managing TLBs

Challenges in using superpage (1/2)

- Large pages increase TLB reach and reduce TLB misses
- They waste memory when footprint is small
- Preclude use of page coloring



(a) Page coloring with normal page address



(b) Page coloring with super page address

Chen et al. ICCD'13

Challenges in using superpage (2/2)

- Require non-trivial changes in OS operation and memory management
- OS tracks memory usage on page granularity
- => a change in even one byte requires writing back the whole page to secondary storage on modification to a memory mapped file
- => huge IO traffic

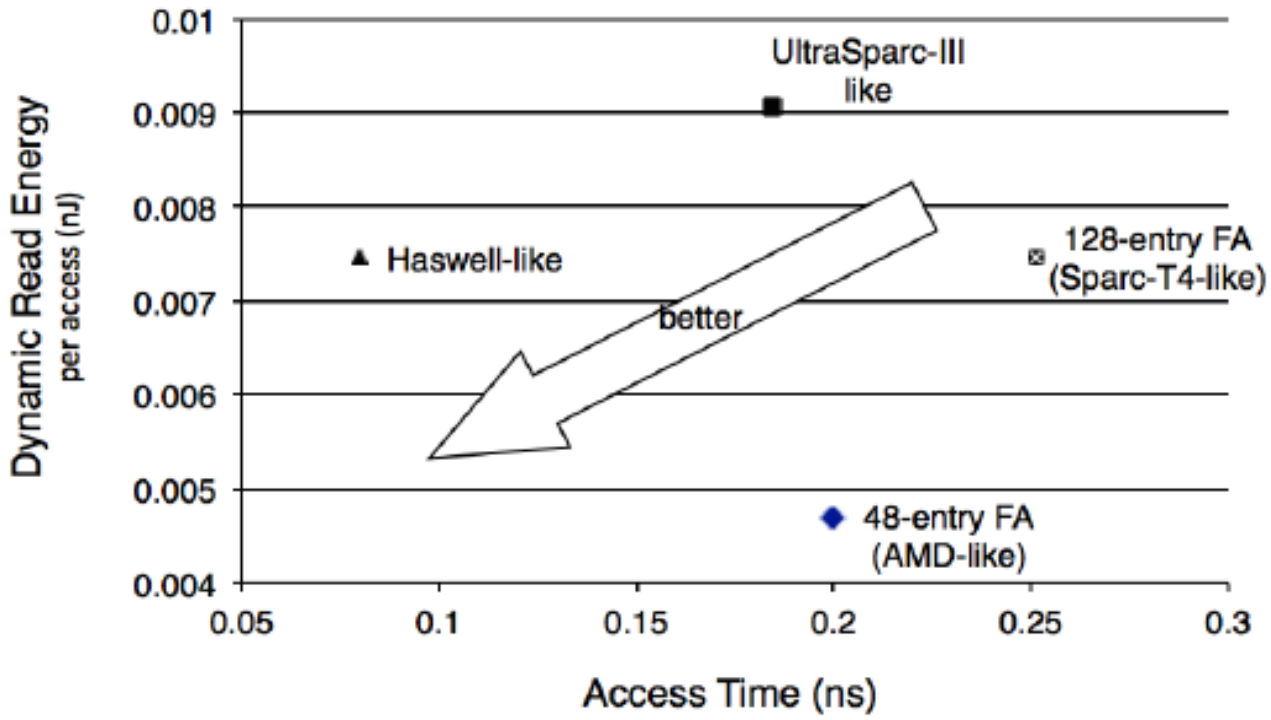
Minimizing TLB flushing overhead

- TLB flushing required on context switch in multitasking systems and virtualized platforms with consolidated workloads
- To avoid flushing TLB, VM and/or process-specific tags need to be used with each TLB entry
- However, this makes TLB a shared resource => contention

TLB design tradeoffs

- Parameters such as TLB size, associativity and number of levels need to be carefully selected
  - Many conflicting goals: low latency, energy, miss-rate, etc.

Access time and Dynamic Energy Trade-Offs



Papadopoulou et al. HPCA'15

Key Ideas of TLB Management Techniques

Key Ideas of TLB Management Techniques (1/2)

- **Reduce TLB accesses by**
  - using virtual caches
- **Reduce TLB miss-rate by**
  - increasing TLB reach
  - prefetching
  - software caching
  - TLB partitioning
  - increasing spatial/temporal locality by reducing page transitions
  - inserting compiler-hints in binary to optimize TLB replacement decisions

Key Ideas of TLB Management Techniques (2/2)

- **Reduce TLB leakage energy by**
  - TLB reconfiguration
  - Designing TLB with non-volatile memory
- **Reduce TLB dynamic energy by**
  - lowering TLB associativity
  - reducing number of ways consulted
- **Reducing flushing overhead**
  - On context switch, save only those entries which are expected to be used in near future
  - Use process-ID as tags

TLB Architectures for Multicore Processors

Private v/s shared TLB Designs

- In multicore processors
  - **Private per-core TLBs** reduce access latency, scales well to large core-count
  - **Shared TLB** provides capacity benefits =>reduces TLB misses

- To bring their benefits together, different techniques have been proposed

Inter-core prefetching (1/2)

- Different threads work on similar data/instructions => TLB misses from different cores show significant correlation
- To avoid them, on each TLB miss, a core can fill its own TLB and also place this translation in prefetch buffers of other cores
- (see figure on next slide)

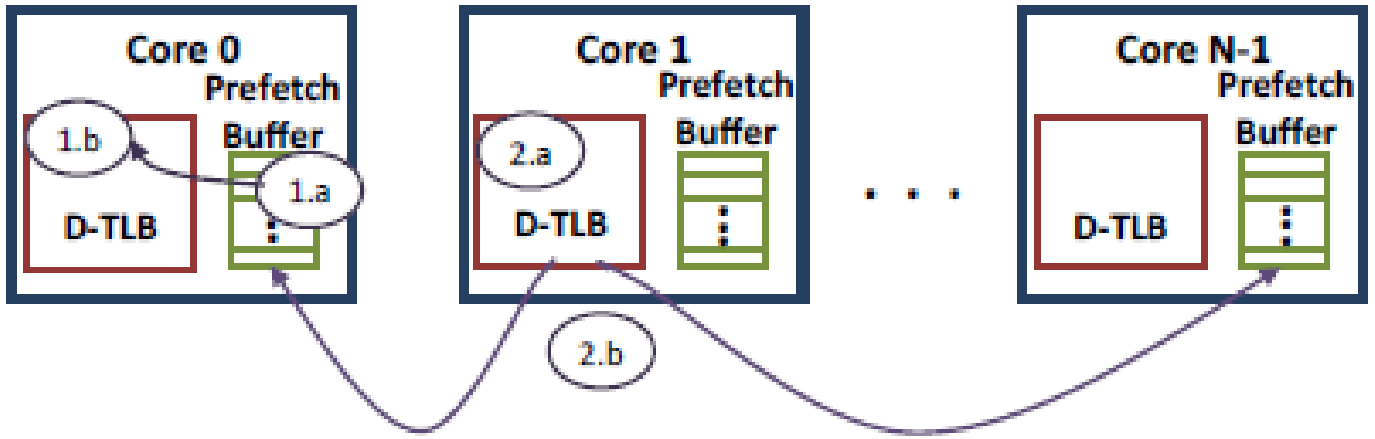
Bhattacharjee et al. PACT'09, Lustig et al. TACO'13

Inter-core prefetching (2/2)

- **Case 1:**

(1a): DTLB miss but prefetch buffer (PB) hit on core 0

(1b): Remove entry from core 0's PB and add to its DTLB



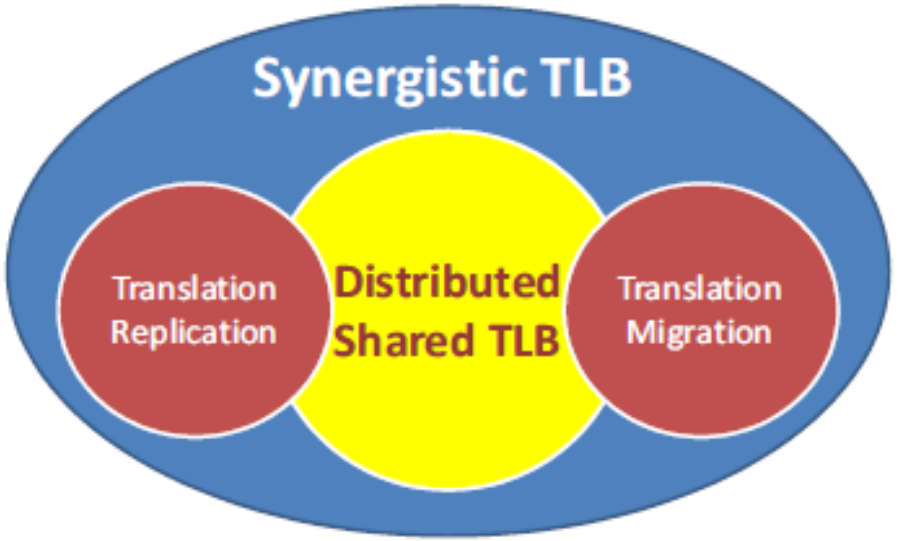
- **Case 2:**

(2a): miss in both DTLB and PB of core 1 => fetch translation from page table into DTLB

(2b): Also push this translation to PBs of other cores

Synergistic TLB design

- Use per-core private TLBs
- Allow victim entries from one TLB to be stored in another TLB for emulating a distributed-shared TLB



- To change remote TLB hits into local TLB hits, perform
  1. replication and/or
  2. migration

  of entries from one TLB to another

Srikantaiah et al. MICRO'10

Improving TLB Energy Efficiency

Motivation for TLB power management

- TLB power consumption is high

- TLB frequently accessed => it is a thermal hotspot, e.g.,
- Power density (in nW/mm²)
  - ITLB: 7.8
  - DL1 : 0.67
  - IL1 : 0.98

- => TLB power management is crucial

Kadayif et al. TODAES'05

Reusing recent translations (1/2)

- **Observation:** Instruction stream has high locality.
- **Technique:** Reuse translation to current page => access to TLB will be required only when another page is accessed => accesses to TLB reduced

- **Observation:** In multi-ported TLB, many accesses occurring to TLB in same or consecutive cycles are to same page or even same cache block
- **Intra-cycle compaction:** Compare TLB accesses within same cycle and then, only send unique VPNs to TLB
- **Inter-cycle compaction:** Store TLB translations in a cycle and reuse in next cycle to avoid TLB accesses

Ballapuram et al. ISLPED'05

Reusing recent translations (2/2)

(a) VAs at four ports of TLB

Cycle				
J	0x8395BA11	0x8395BAEE	0x8395BA0F	0xEEFFDDAA
J+1	0x8395BAF1	0x8395BCED	0x87241956	-----

(b) Corresponding VPNs

Cycle				
J	0x8395B	0x8395B	0x8395B	0xEEFFD
J+1	0x8395B	0x8395B	0x87241	-----

(c) VPNs after intra-cycle compaction [(b)⟶(c)]

Cycle				
J	0x8395B	-----	-----	0xEEFFD
J+1	0x8395B	-----	0x87241	-----

(d) VPNs after inter-cycle compaction [(b)⟶(d)]

Cycle				
J	0x8395B	0x8395B	0x8395B	0xEEFFD
J+1	-----	-----	0x87241	-----

Ballapuram et al. ISLPED'05

Compiler-based techniques

Challenge: Translation-reuse is ineffective if successive memory accesses are to different pages

Solution: Perform compiler-management to reduce page-transitions for both data and instruction

DTLB optimization techniques

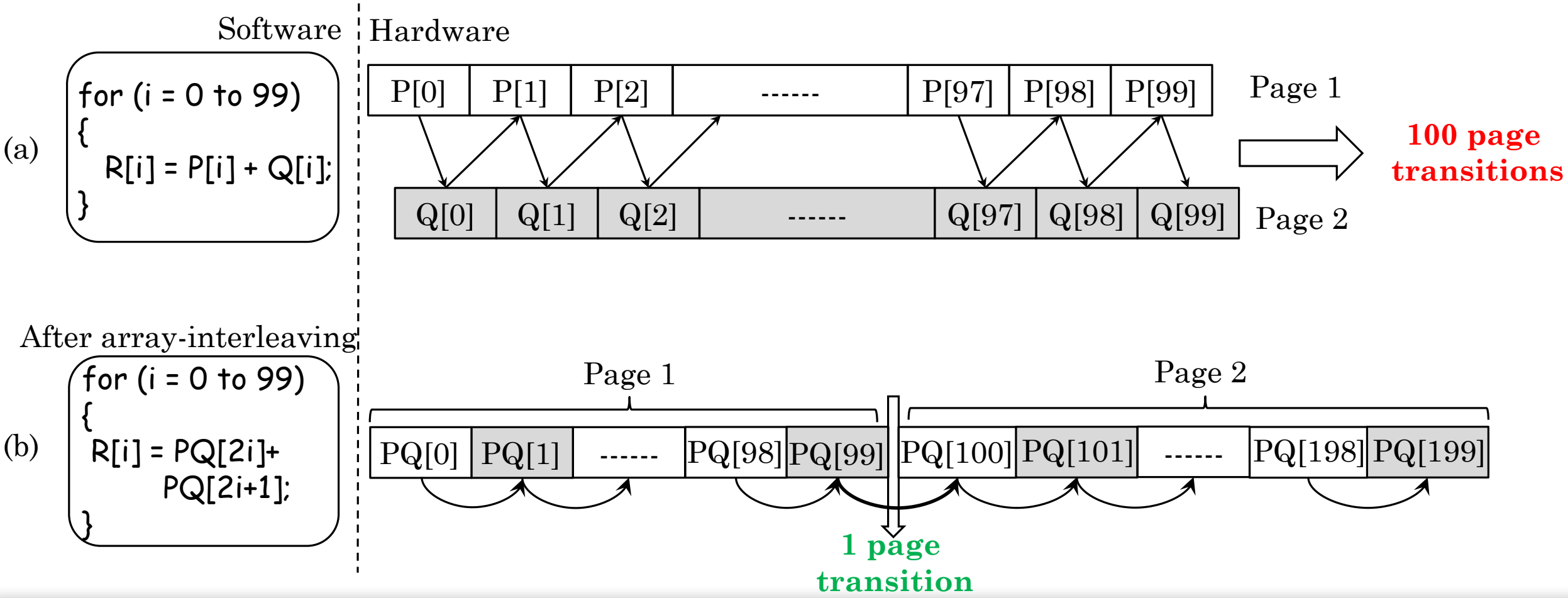
1. Array interleaving
2. Instruction scheduling
3. Operand reordering

We will discuss them in coming slides

Jeyapaul et al. IJPP'10

Array-interleaving

- For innermost loop of nested loops, perform array interleaving between two arrays if they are of the same size and have same reference functions



Loop-unrolling, instruction scheduling and operand reordering

- **Instruction scheduling:** aggregates instructions accessing same pages
- **Operand reordering:** changes memory access pattern by reordering operands

ITLB optimization techniques

- **Observation:** Successive instructions are from different pages when
  1. loop block of a program crosses page limit and
  2. callee and caller function are on different pages.

- **Technique:** Use code-placement strategy
- Reallocate function blocks of a program to minimize page switches
- e.g., function position limits are used over a single page to ensure placing a function in same page

Jeyapaul et al. IJPP'10

Using privilege-level and memory-region information

- **Observation:** A user-mode instruction fetch cannot hit in a block storing kernel-mode code
- **Technique:** Store this privilege-level info with each block of IL1 and ITLB
- On a user-mode instruction fetch, only ways storing user-mode code are consulted.

- **Observation:** An access to heap data cannot hit in a block storing stack data and vice versa
- **Technique:** By storing this info in each DL1 cache block, accesses to those ways are avoided which are guaranteed to lead to a miss

Fang et al. JSA'13

Dead-page based management

- **Observation:** In managed languages e.g., Java, garbage collector reclaims allocated memory only during garbage collection
- => Some pages become dead, i.e., most of their objects are dead. These pages.
  - are rarely accessed
  - have poor spatial/temporal locality
  - responsible for most of TLB misses
- **Leveraging dead-page information:**

1. During eviction, preferentially evict dead pages
2. In future, bypass these pages from TLB

Tong et al. ICCD'14

TLB reconfiguration in processors with multiple page sizes

- Some processors use two-level TLB design with different L1 TLBs for every page size

- **Challenge:** Due to accesses to multiple L1 TLBs and page walks for small page size (4KB), large dynamic energy is wasted
- **Observation:** Contribution of all L1 TLBs to hits is not same => accessing all L1 TLBs may not boost performance
- **Technique:** Perform TLB way-reconfiguration
  - Disable L1 TLB ways if their utility is small

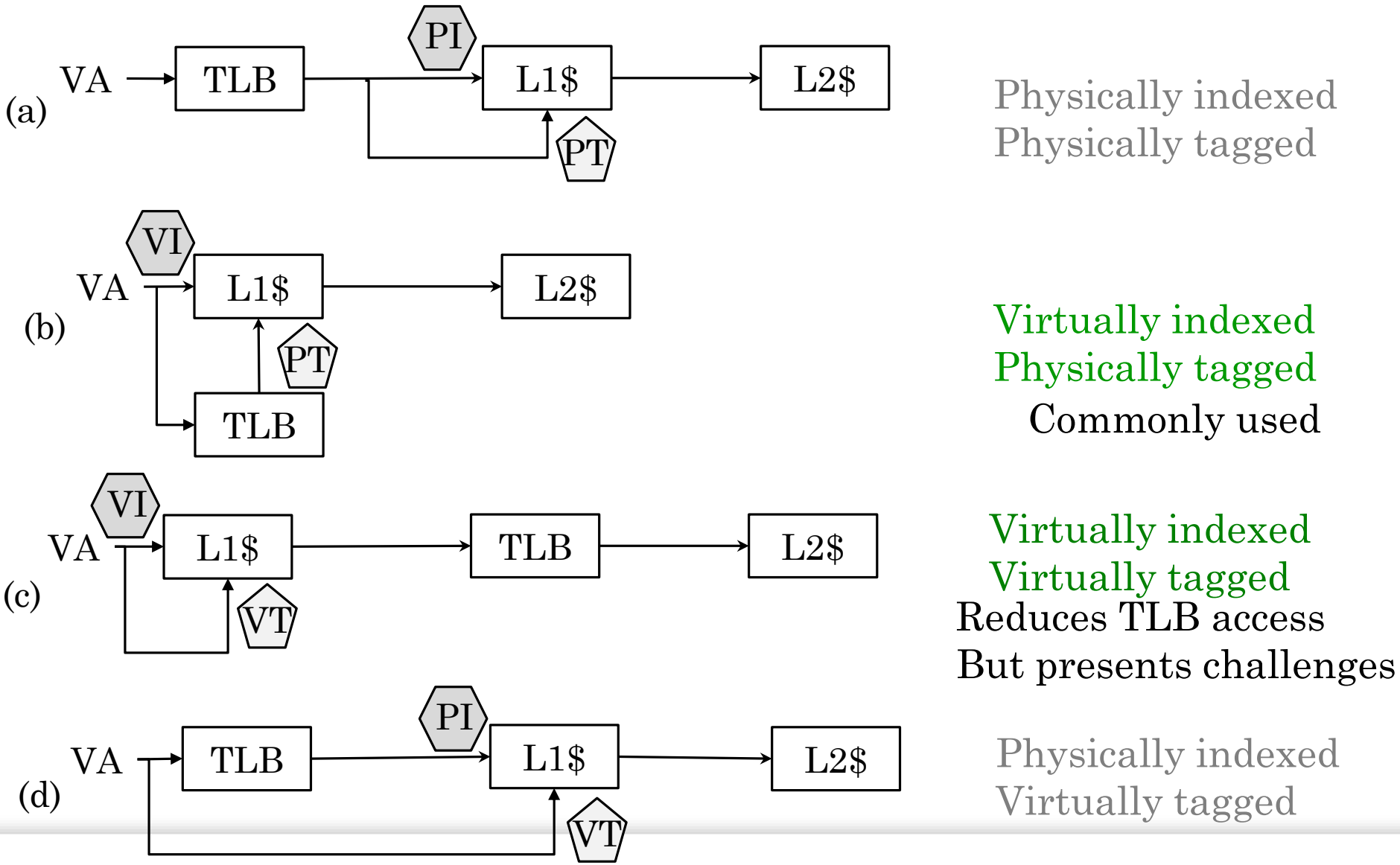
Karakostas et al. HPCA'16

Techniques for Virtual Cache

4 possible cache/TLB organizations

- Depending on whether index and tag of L1 cache is derived from virtual or physical address, we can have 4 possible organizations:
  - PI-PT cache (physically-indexed, physically-tagged)
  - VI-PT cache (virtually-indexed, physically-tagged)
  - VI-VT cache (virtually-indexed, virtually-tagged) (also called **virtual cache**)
  - PI-VT cache (physically-indexed, virtually-tagged)

4 possible cache/TLB organizations



(a) Physically indexed
Physically tagged

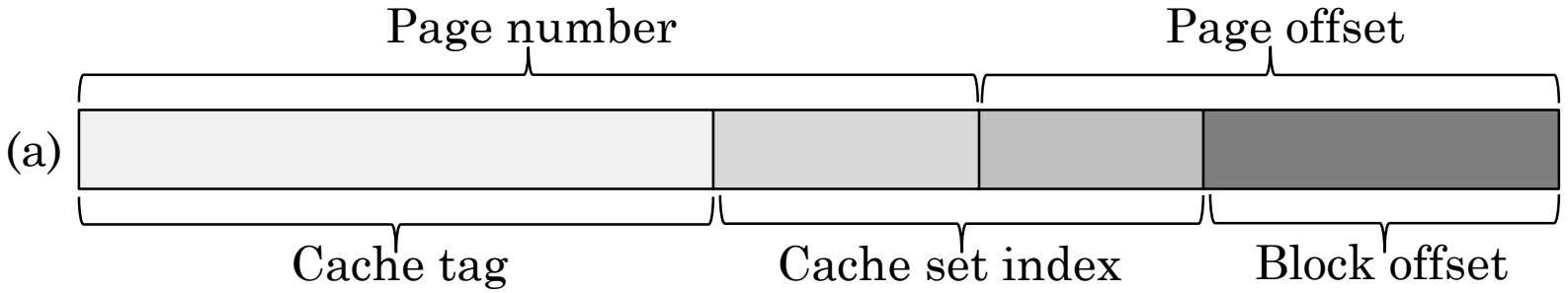
(b) Virtually indexed
Physically tagged
Commonly used

(c) Virtually indexed
Virtually tagged
Reduces TLB access
But presents challenges

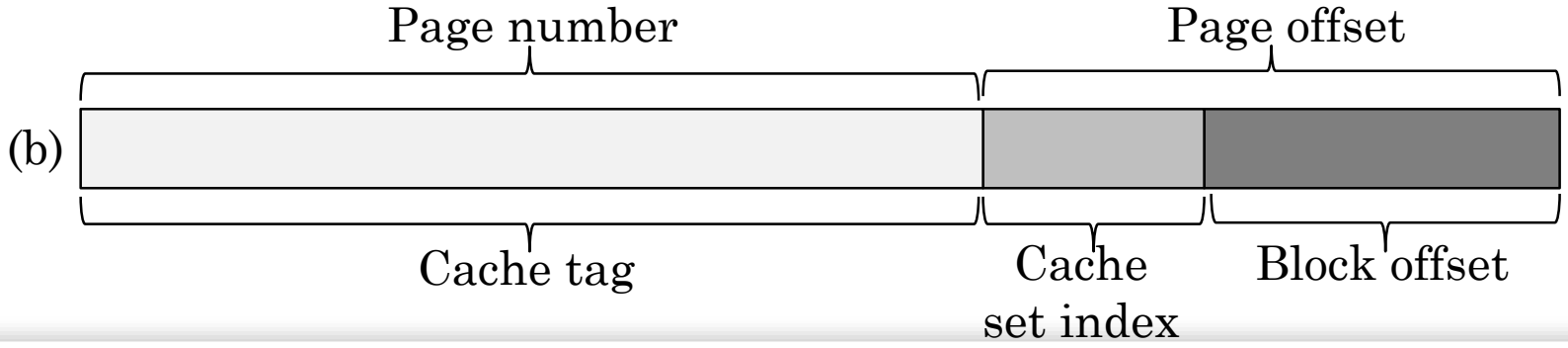
(d) Physically indexed
Virtually tagged

Physical address subdivision

Address subdivision in **general case**



Special case: all bits of cache set come from page offset
=> Cache set location of an address is independent of VA-to-PA translation



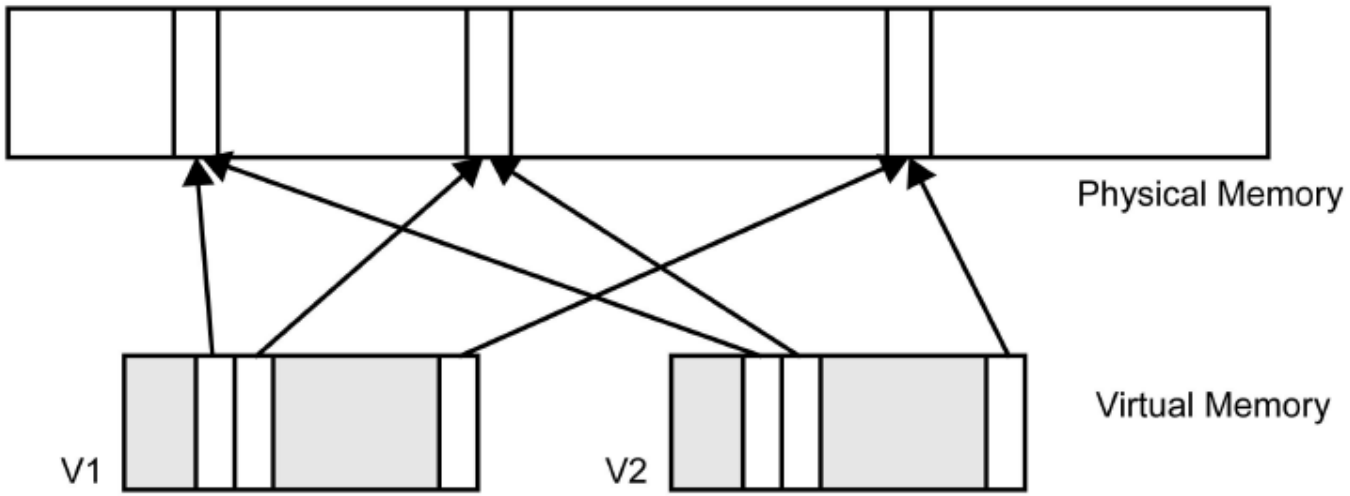
TLB access not required for obtaining index

Useful term: Homonyms

- **Homonyms:** Homonyms occur when a VA refers to multiple physical locations in different address spaces.
- Challenge: If not disambiguated, incorrect data may be returned.
- Solution: Use application-specific ID (ASID) to remove homonyms

Useful term: Synonyms

- **Synonyms:** Multiple different VAs mapping to same PA



- Challenge: These synonyms can reside in multiple sets in cache under different VAs. If one synonym of a block is modified, access to other synonyms with different VAs may return stale data
- Solution: (discussed in next slides)

Qiu et al. TC'08

Observations which simplify virtual cache designs

- In most apps, very few pages have synonyms
- These synonyms are accessed infrequently
- Most synonym pages only see reads => they cannot generate read/write inconsistency

Ideas for designing virtual cache

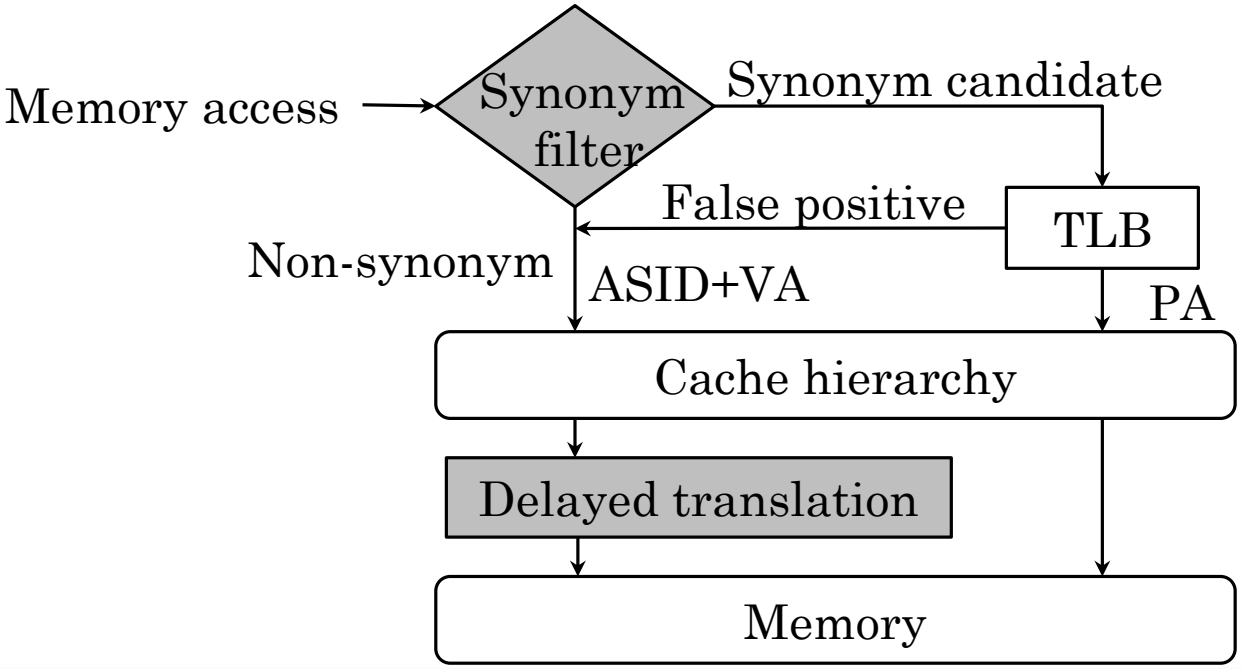
1. Remap accesses directed towards different VAs to a single 'primary' VA
   - Primary VA is found based on the first touch

Primary VA	Secondary VA	Physical Address
W	--	P1
X	X1, X2, X3	P2
Y	--	P3
Z	Z1	P4

See survey paper for details

Ideas for designing virtual cache

2. Use hybrid virtual cache designs where PA is used for addresses with synonyms and VA is used for non-synonym addresses
   - Can default to physical caching if required



ASID = app-specific ID

Park et al. ISCA'16

References

- Sparsh Mittal, “A Survey of Techniques for Architecting TLBs”, Concurrency and Computation, 2017 (link)