Virtual Memory

Works best when you don’t need it

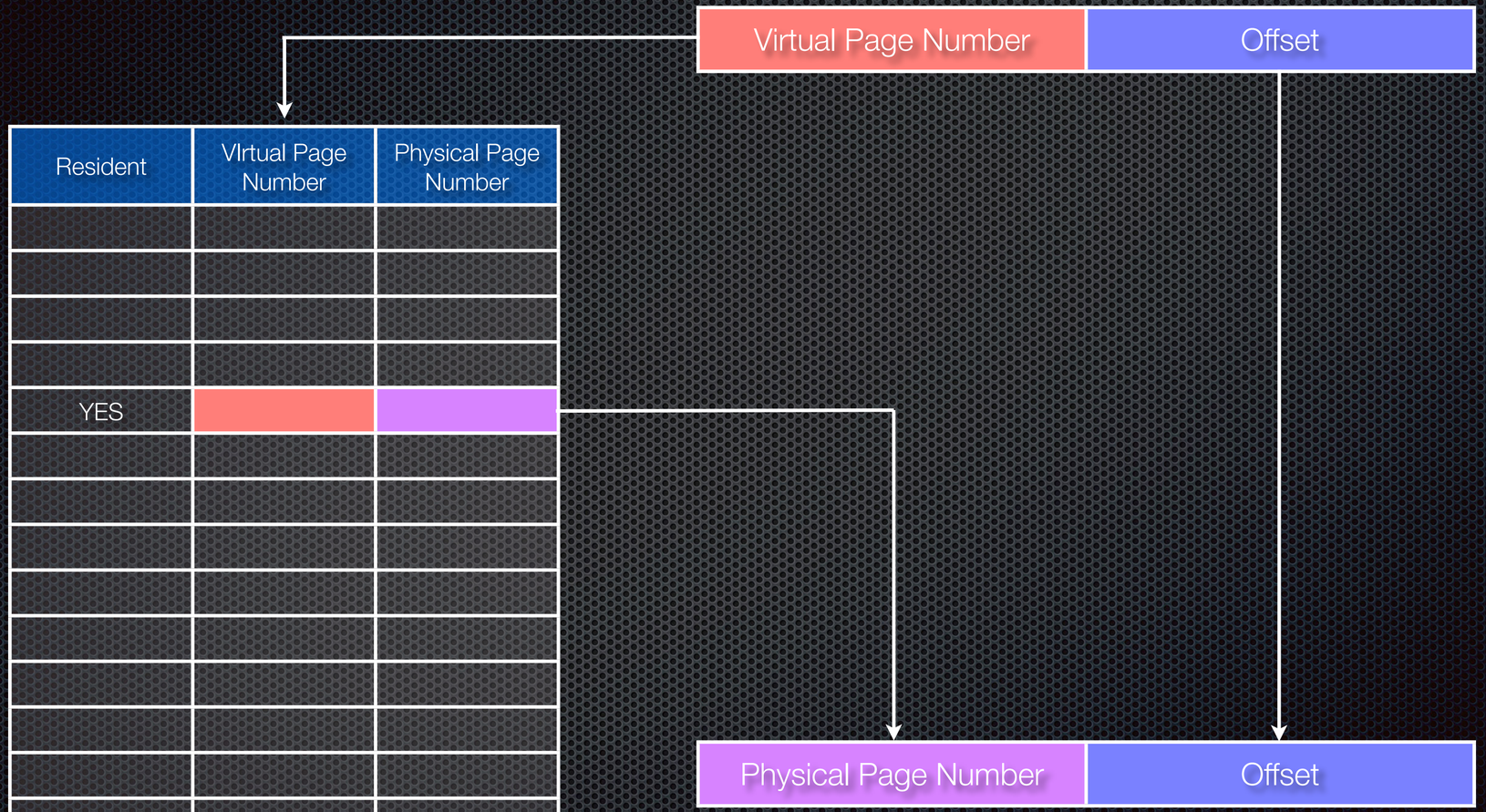
Historically

- Motivated by expensive leased mainframes
  - Large fraction of CPU time stalled on I/O
- A way to maximize utilization
- Supports multitasking and protection

Paging

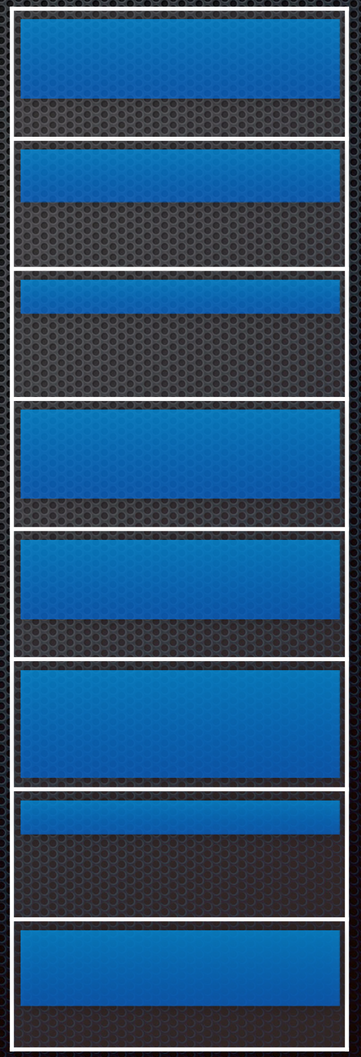
- Simple mapping
- Fixed-size units of memory (e.g., 4K)
- Page table indexed by virtual page number
- Returns physical page number
- Extended to support non-resident pages

Page Table is Simple Lookup



Internal Fragmentation

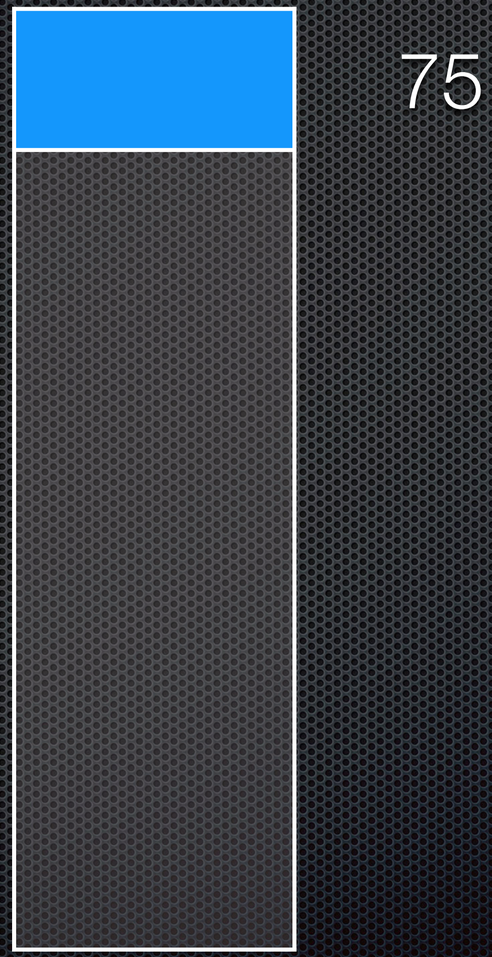
- Code and data don’t fit precisely into pages
- Left-over space is waste
- This was a big deal when machines had 64K (16 pages) of memory, in the 1960’s
- How can we scavenge that waste?



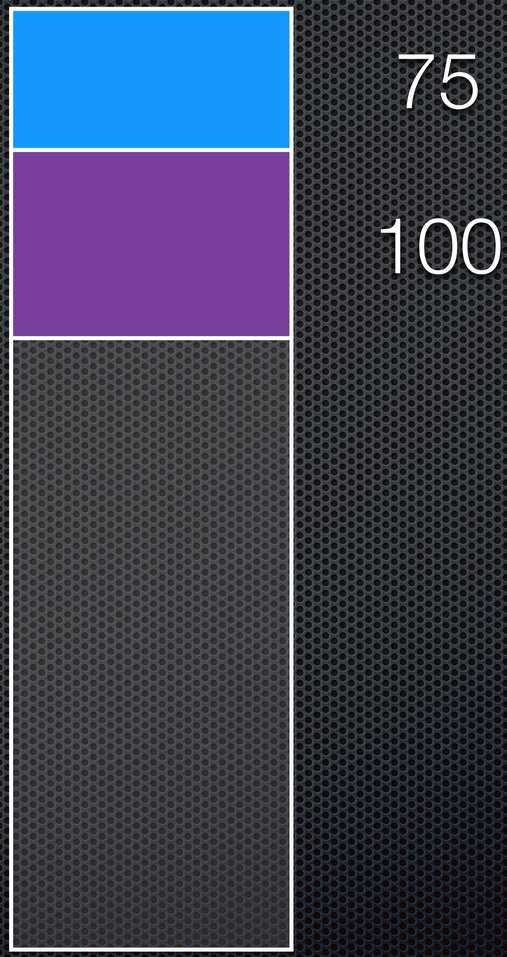
Segmentation

- Make memory units variable in size
- More complex mapping
- Relocation pointer added to address, with result compared to an upper bound
- No internal fragmentation (for statically allocated applications)

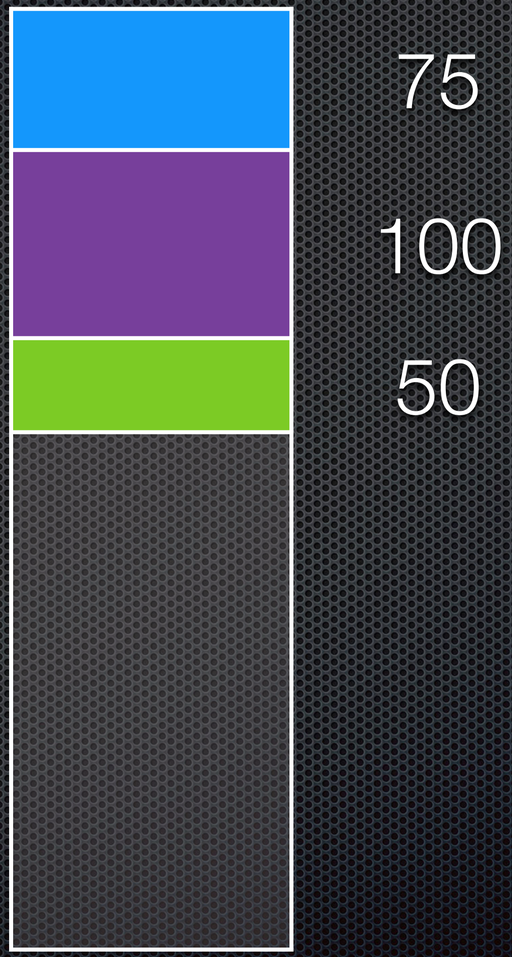
External Fragmentation



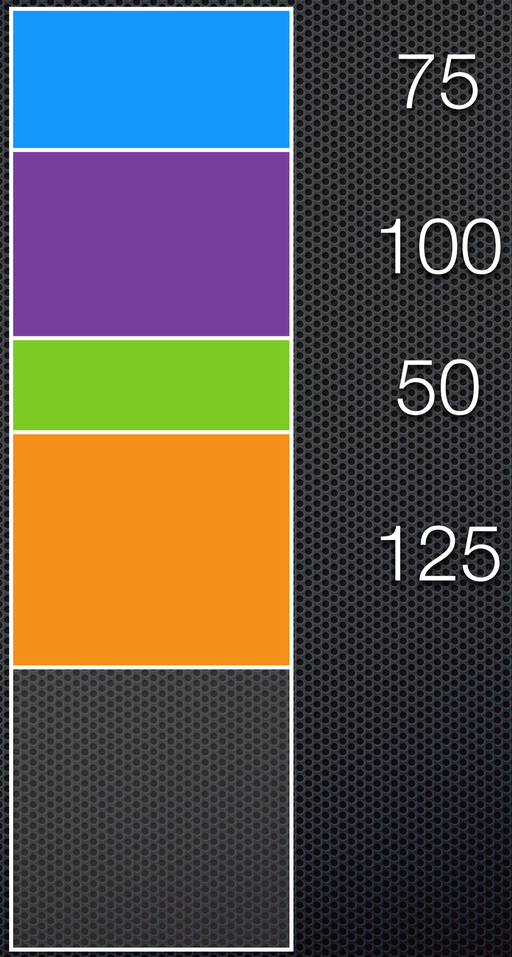
External Fragmentation



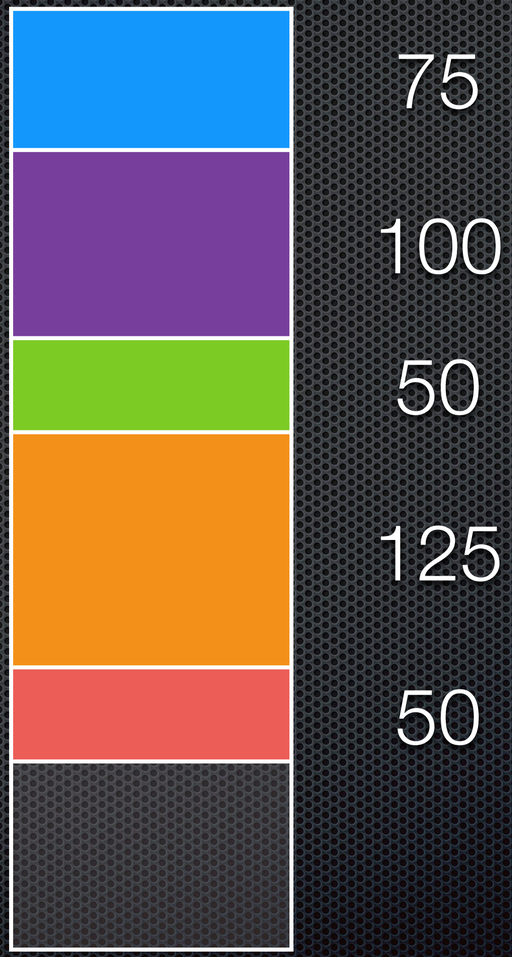
External Fragmentation



External Fragmentation

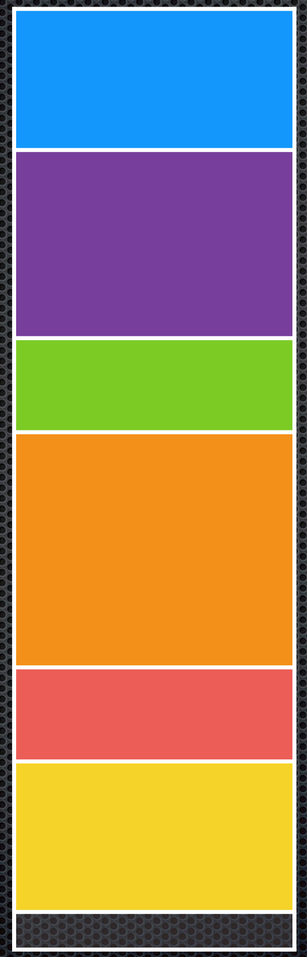


External Fragmentation



External Fragmentation

Next some deallocations



75

100

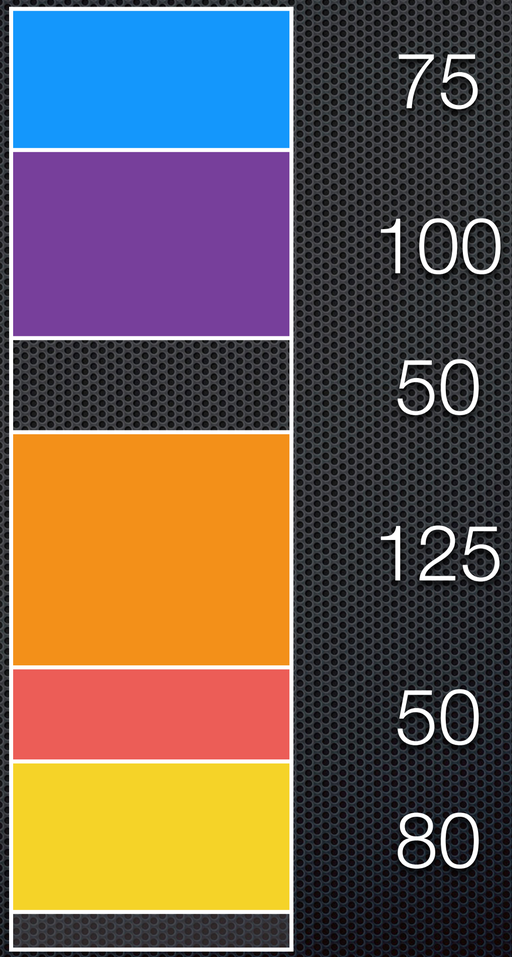
50

125

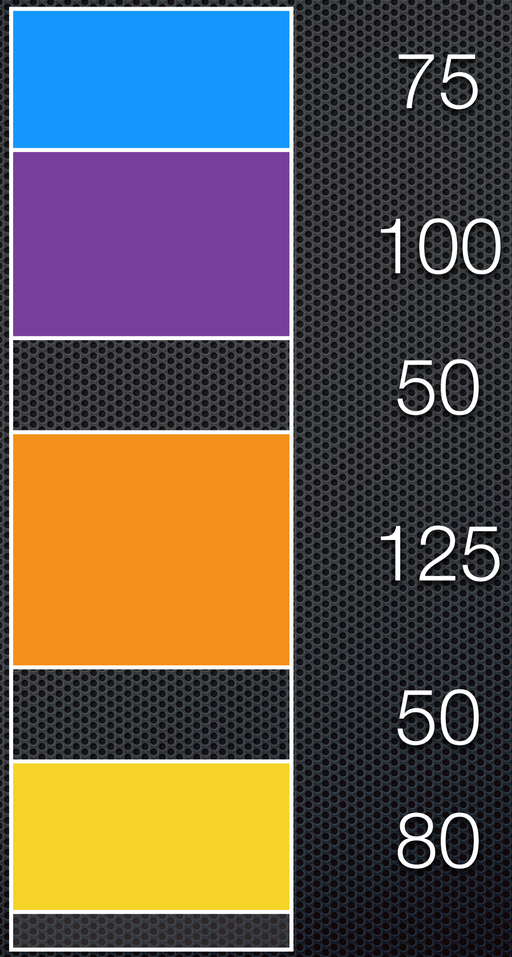
50

80

External Fragmentation



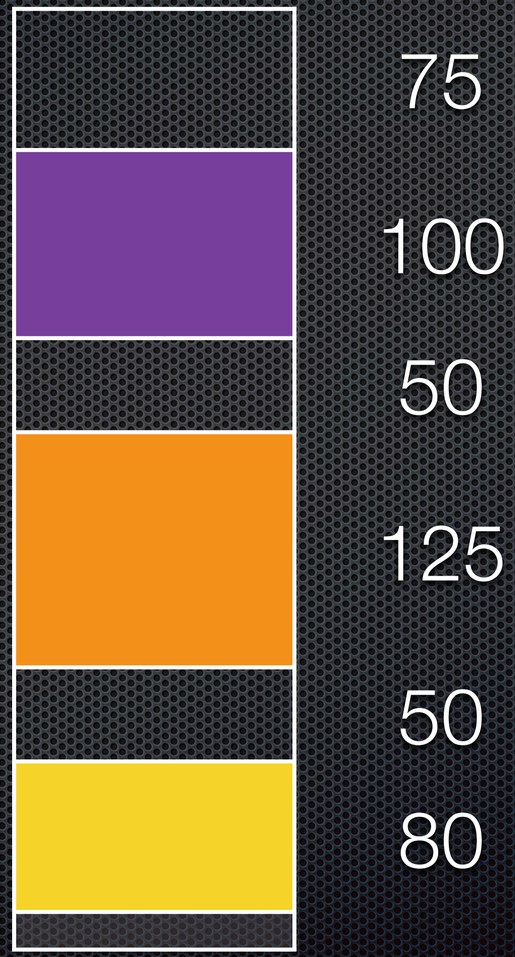
External Fragmentation



External Fragmentation

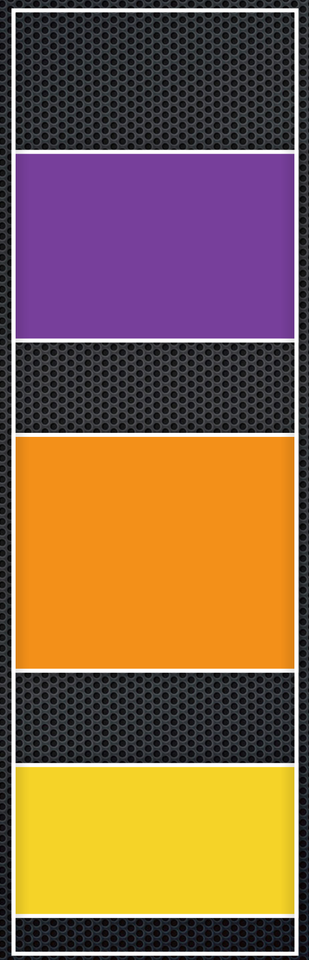
Next try to allocate 135

195 is available but not in one place



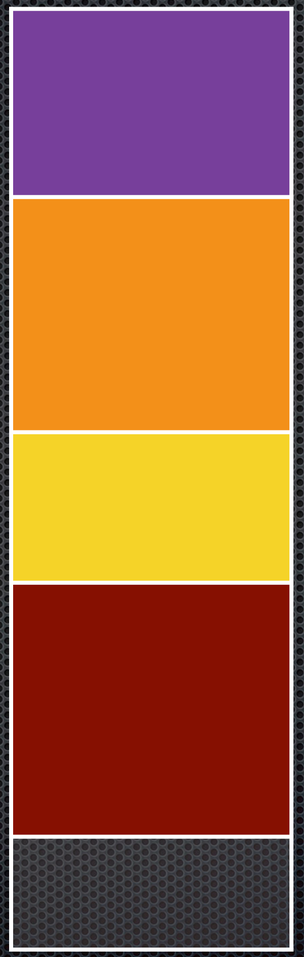
Compaction

Now there is enough space to allocate 135



195

Compaction



135

Compaction

- Time consuming process
- Can cause noticeable pauses
- Many variations on incremental compaction
- Even without compaction, segment swap time can be long

Paged Segmentation

- Break segments into pages
- Avoids external fragmentation
- Allows partial swapping of segments
- Another hierarchy of tables

Return to Paging

- Fixed size avoids external fragmentation and compaction
- Matched well with traditional disk layout
- But as memory grew to GB, page tables became huge
  - When page tables can’t be in cache, translations are very slow

Superpages

- Have a small number of fixed page sizes instead of just one
- e.g., 4K and 2M
- Reduces page table size
- Increases complexity of management policy and mapping function

Hierarchical Tables

- Rather than a flat page table, have multiple levels of tables
- Creates a tree-structured table (radix page table)
  - Part of VPN selects entry in table that points to next table
  - Next part of VPN selects entry in that table, etc. (Intel has 4 levels)
- Only need to fill in portions of the tree that are active
- Translation can take many accesses (four for non-virtualized accesses)

Larger Address Spaces

- Page tables grow huge, or deep in hierarchy
- Many pages are not allocated -- sparse use of tables
- Can invert the tables
  - Hash on virtual address to a PPN entry or a linked list of physical addresses with their virtual translations (collision chain) -- search list
  - Works well if density is low

Larger Address Spaces

- Page tables grow huge, or deep in hierarchy
- Many pages are not allocated -- sparse use of tables
- Can invert the tables
  - Hash on virtual address to a linked list of physical addresses with their virtual translations -- search list
  - Inverted tables tend to be large. Many variations.

Translation Lookaside Buffer

Speeding Virtual Memory Translations with a Small Fully Associative Cache

Simple Example

- 16-bit virtual and physical addresses
- Word addressing
- 4K word pages
- 16 virtual and physical pages
- TLB is 4-entry, fully associative, LRU

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	2	3
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	2	3
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Example

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
I		
I		
I		
I		

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens?

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	1	E
I		
I		
I		

Miss

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens?

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	1	E
V	2	3
I		
I		

Miss

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens?

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	1	E
V	2	3
V	3	5
I		

Miss

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens?

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	1	E
V	2	3
V	3	5
V	9	8

Miss

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens?

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	A	F
V	2	3
V	3	5
V	9	8

Miss

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens for the next three?

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	A	F
V	1	E
V	2	3
V	3	5

Miss
Miss
Miss

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens for the next four?

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	A	F
V	1	E
V	2	3
V	3	5

Hit 3
Hit 4
Hit 1
Hit 2

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

What happens?

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

Memory

PPN	Page
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

16-bit Virtual Address

Translation Lookaside Buffer

V/I	Tag VPN	PPN
V	A	F
V	1	E
V	9	8
V	3	5

1, 2, 3, 9, A, 1, 2, 3, 2, 3, A, 1, 9

Page Table

VPN	PPN
0	1
1	E
2	3
3	5
4	9
5	C
6	7
7	6
8	2
9	8
A	F
B	D
C	4
D	0
E	A
F	B

TLB and Multiprocessing

- TLB is augmented with a process tag
- Avoids need to purge TLB on context switch
- Shared data has its translation aliased and replicated

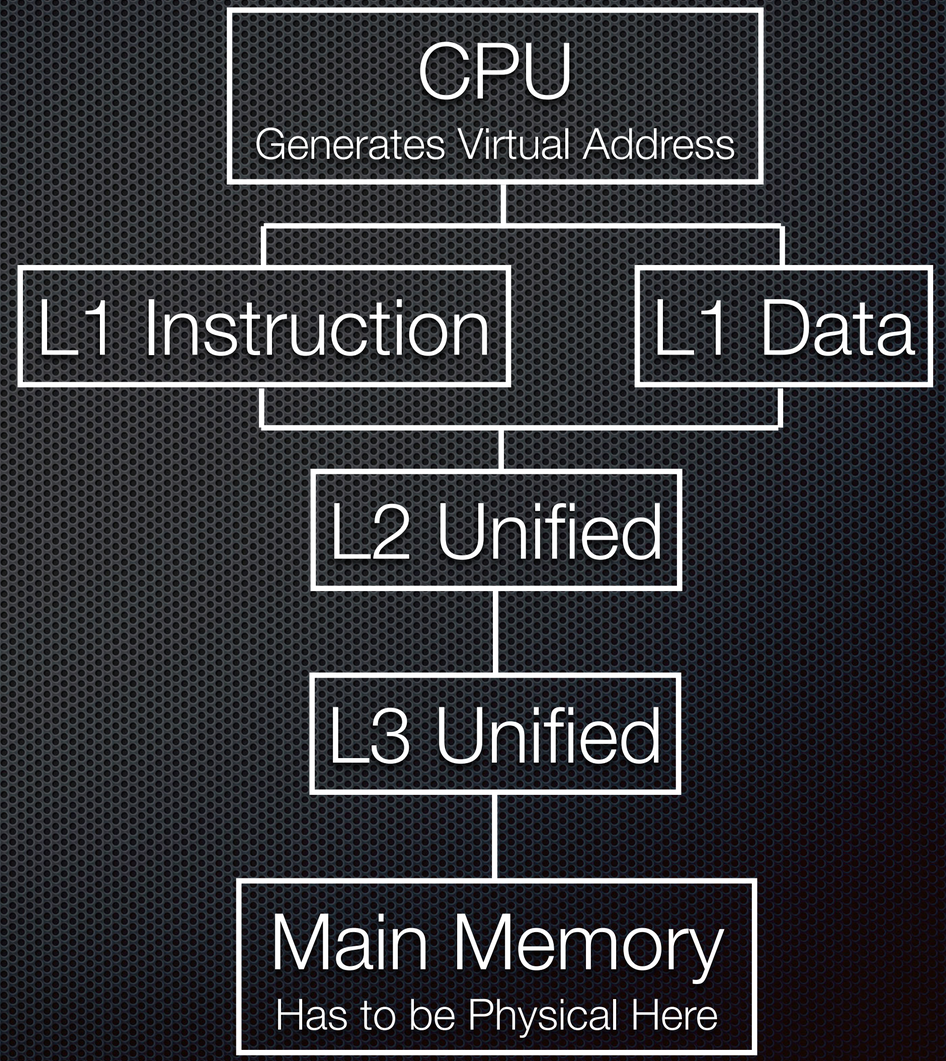
TLB Limitations

- Full associativity is power hungry and slow
- TLB has to be small (e.g., 128 - 512 entries), often banked
- Works well with small working set, but poor for more dynamic sets (e.g., some garbage collected heap organizations)
- Has to interact with caches

Where to Translate?

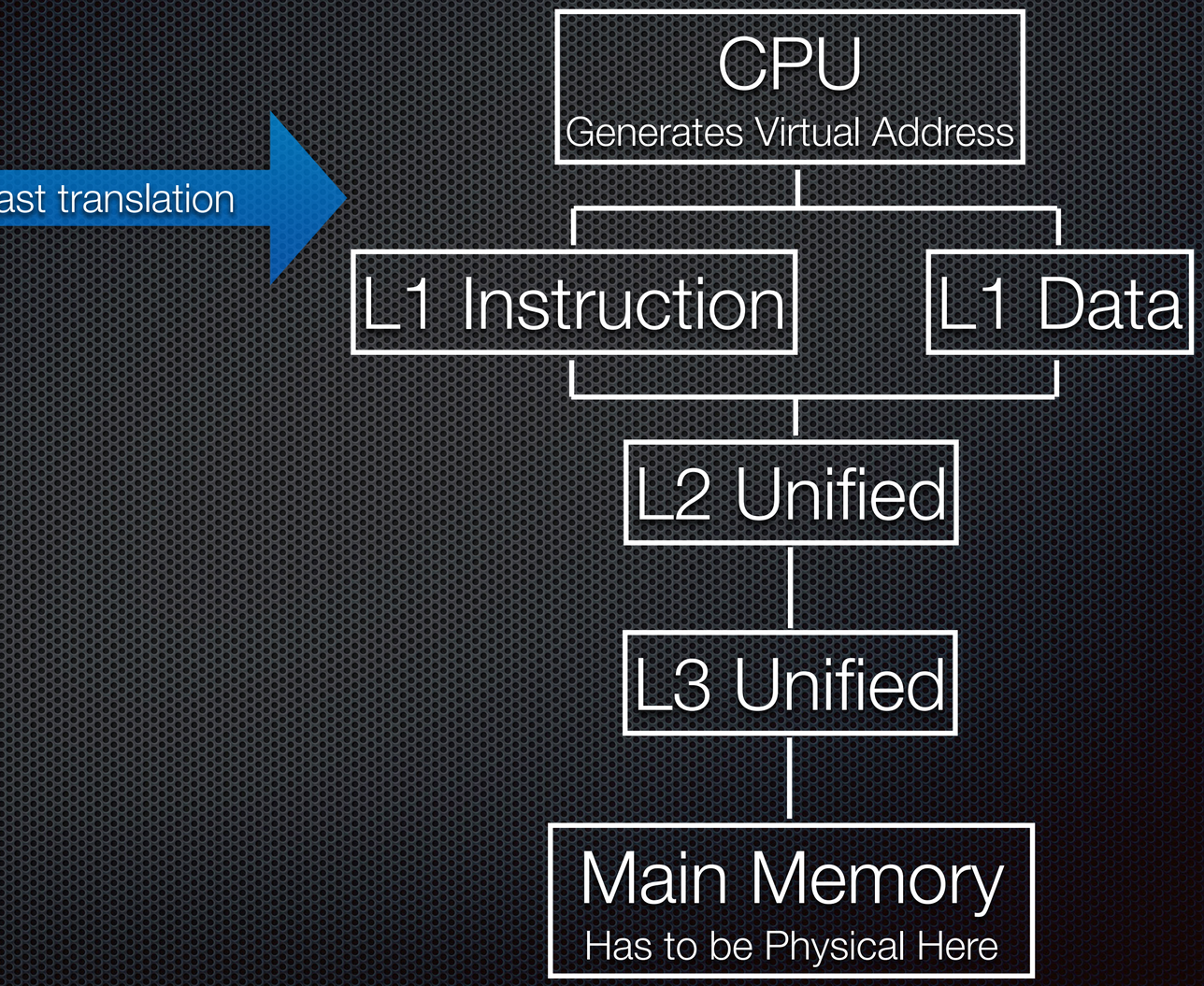
Virtual to physical has to happen somewhere

Options

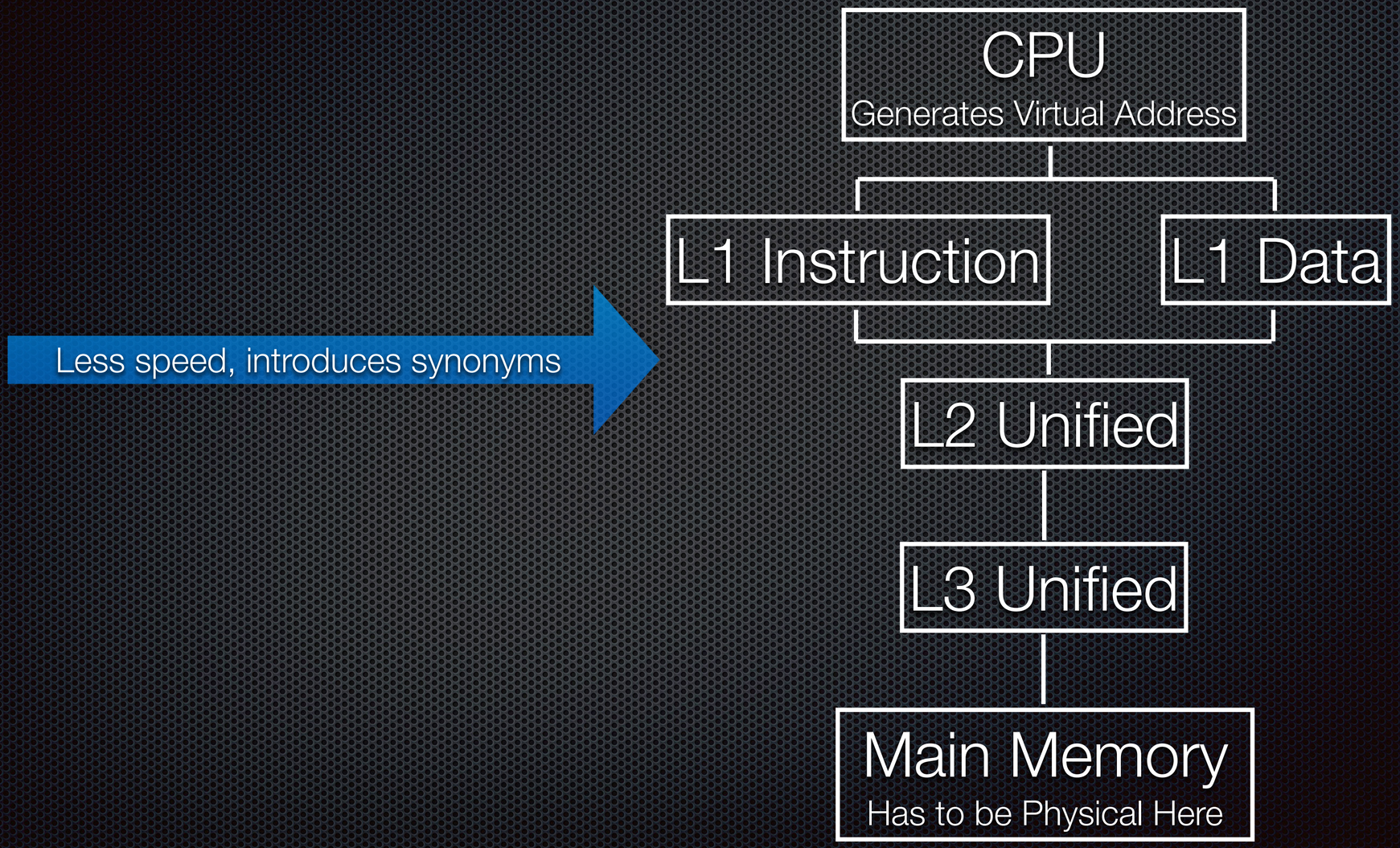

CPU
Generates Virtual Address
L1 Instruction
L1 Data
L2 Unified
L3 Unified
Main Memory
Has to be Physical Here

Options

All physical, requires fast translation



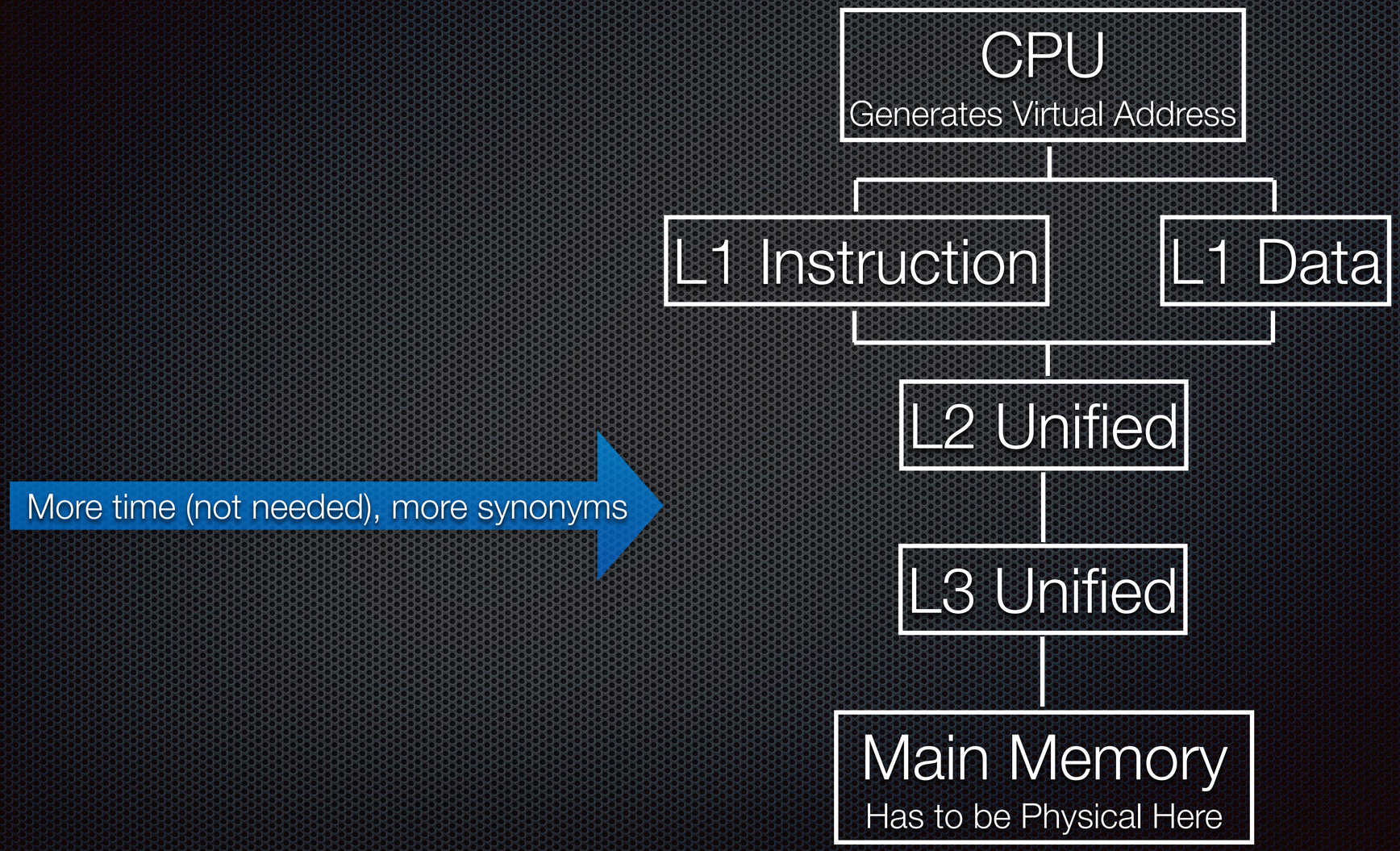
Options


CPU
Generates Virtual Address
L1 Instruction
L1 Data
Less speed, introduces synonyms
L2 Unified
L3 Unified
Main Memory
Has to be Physical Here

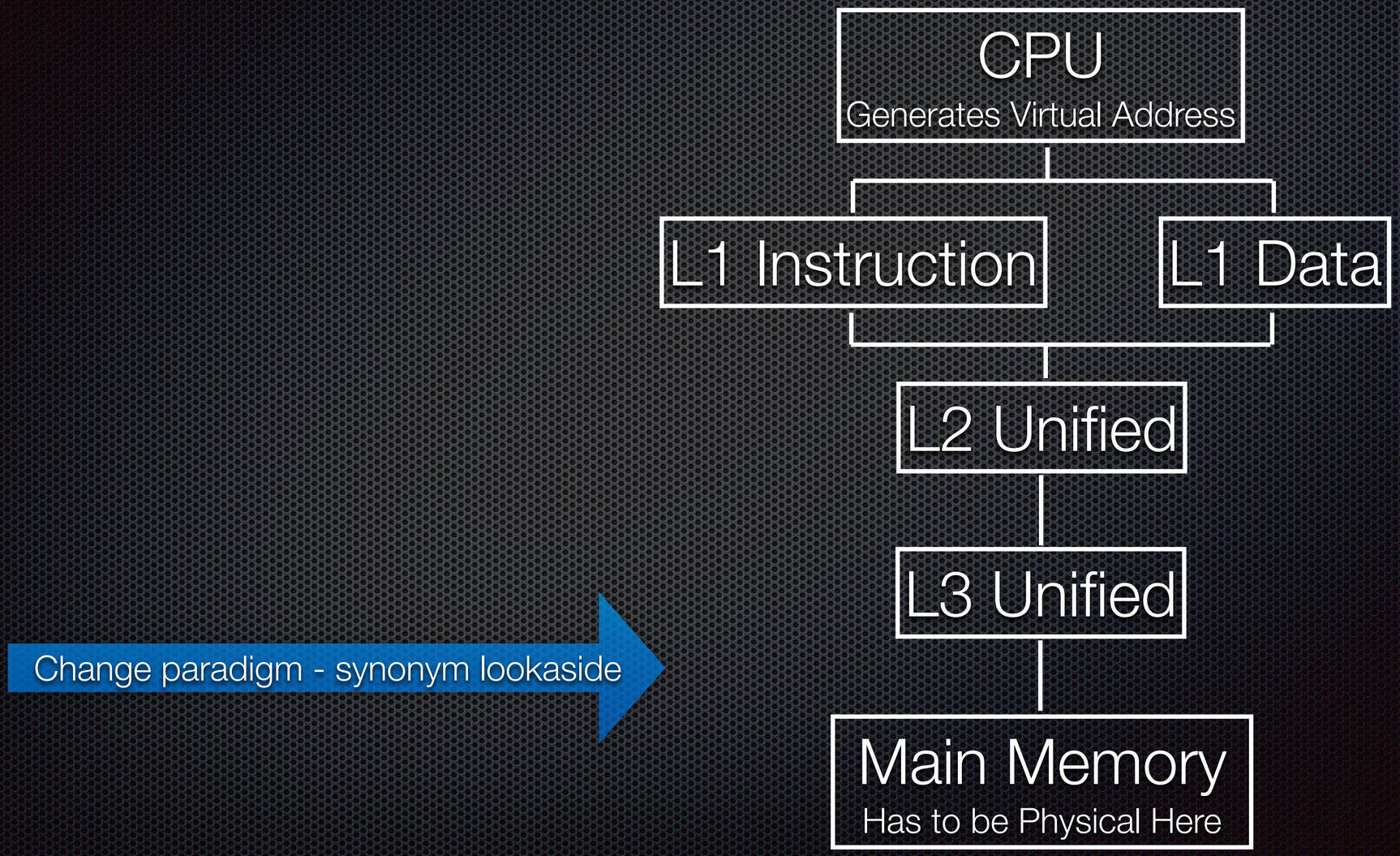
Synonyms

- Multiple processes can share physical addresses
- Each process has its own virtual address space
  - A physical address can have multiple virtual addresses
- A virtual cache can end up holding copies of a value with different addresses
- If one of those values gets changed, the other copies must be invalidated
- Requires that highest physically-mapped cache has pointers to synonyms

Options


CPU
Generates Virtual Address
L1 Instruction
L1 Data
L2 Unified
L3 Unified
Main Memory
Has to be Physical Here
More time (not needed), more synonyms

Options


CPU
Generates Virtual Address
L1 Instruction
L1 Data
L2 Unified
L3 Unified
Main Memory
Has to be Physical Here
Change paradigm - synonym lookaside

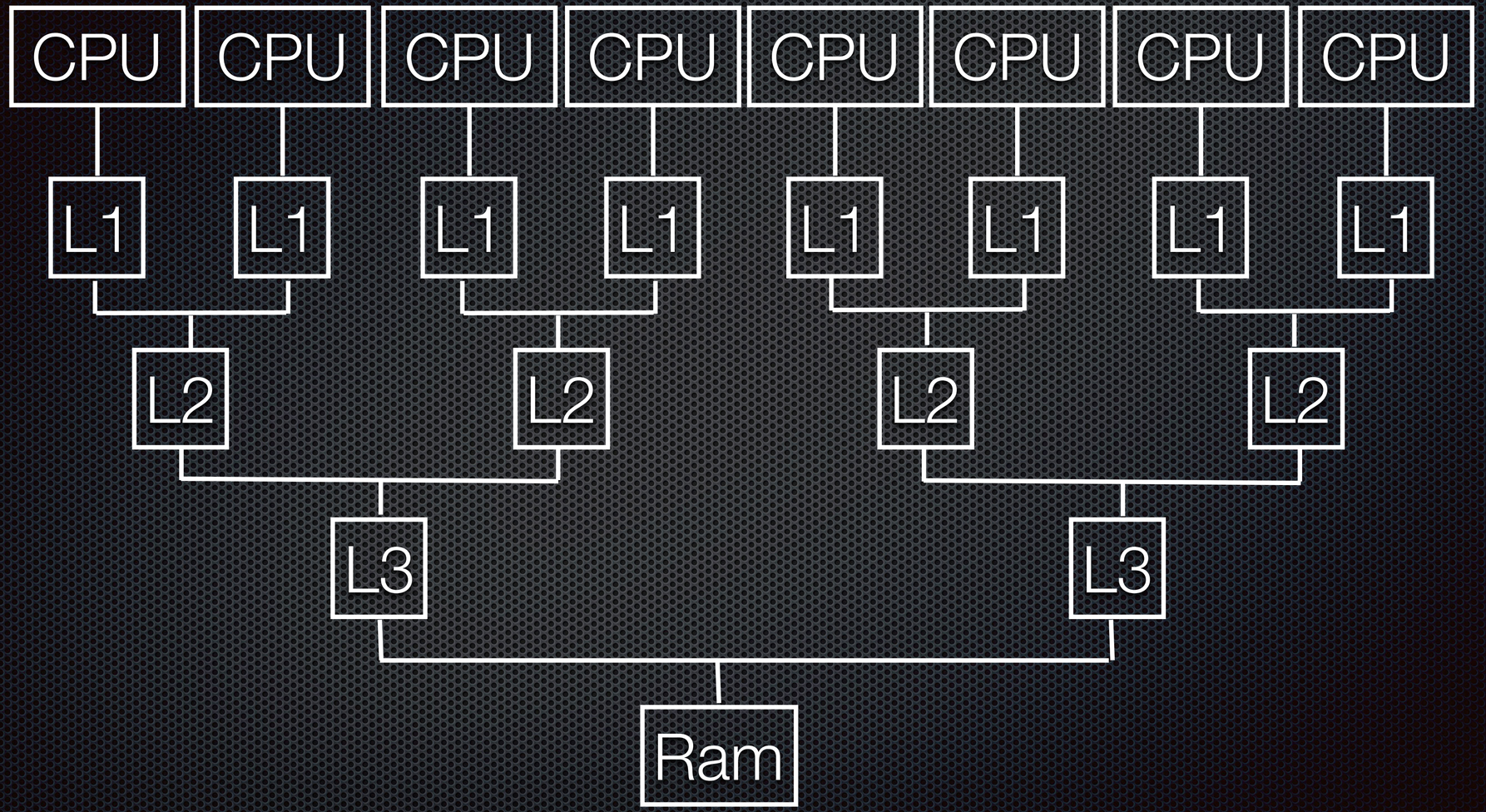
Typical

- Place translation and TLB between L1 and L2
  - TLB can be larger (more effective), but more complex cache structures to resolve synonyms and handle invalidation
- Translate above L1
  - Avoids synonyms, but TLB must be smaller to be faster (less effective)
- Synonyms are less common than translations

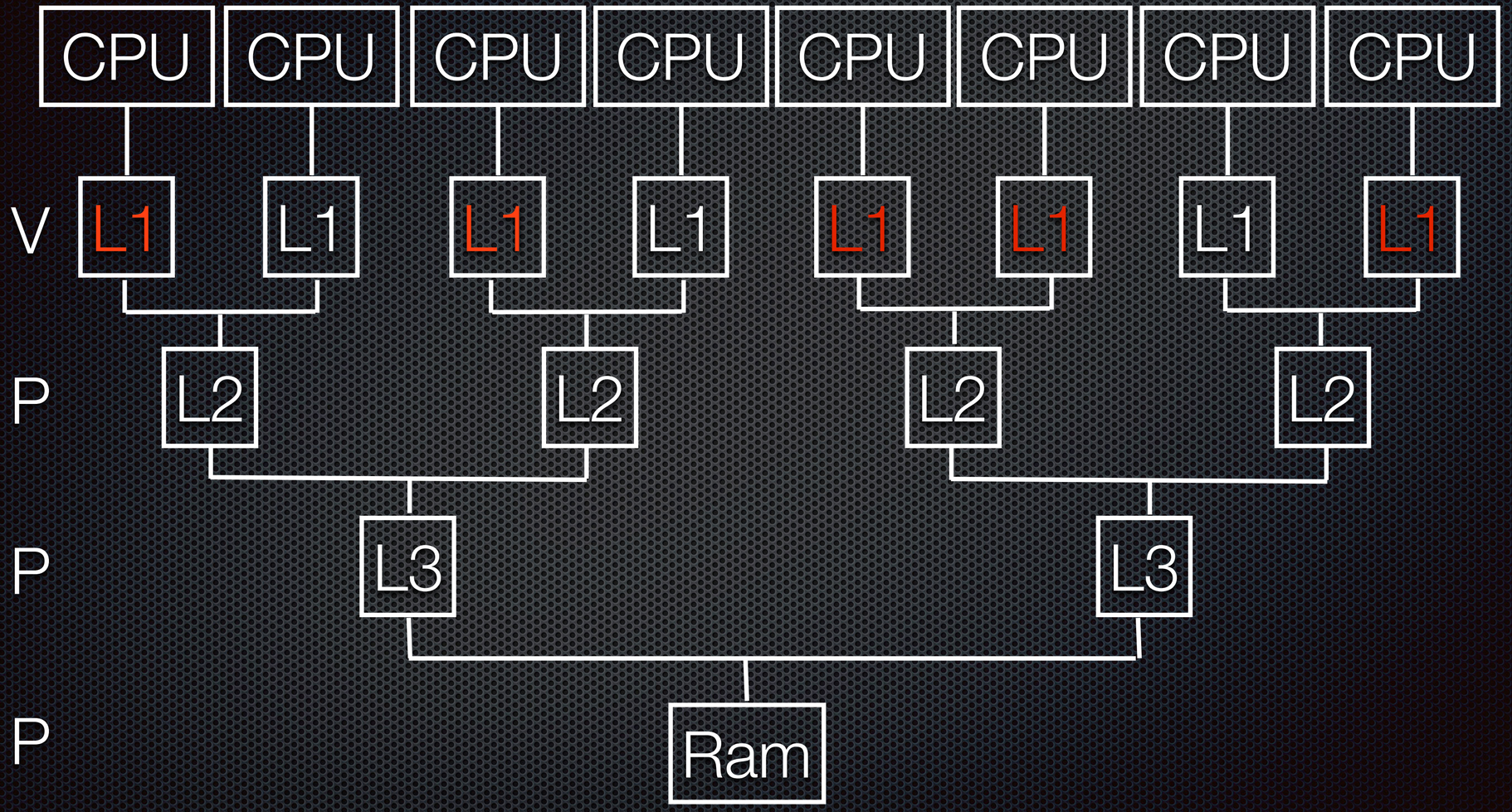
Typical

- Place translation and TLB between L1 and L2
  - TLB can be larger (more effective), but more complex cache structures to resolve synonyms
- Translate above L1
  - Avoids synonyms, but TLB must be smaller to be faster (less effective)
- Synonyms are less common than translations....but....

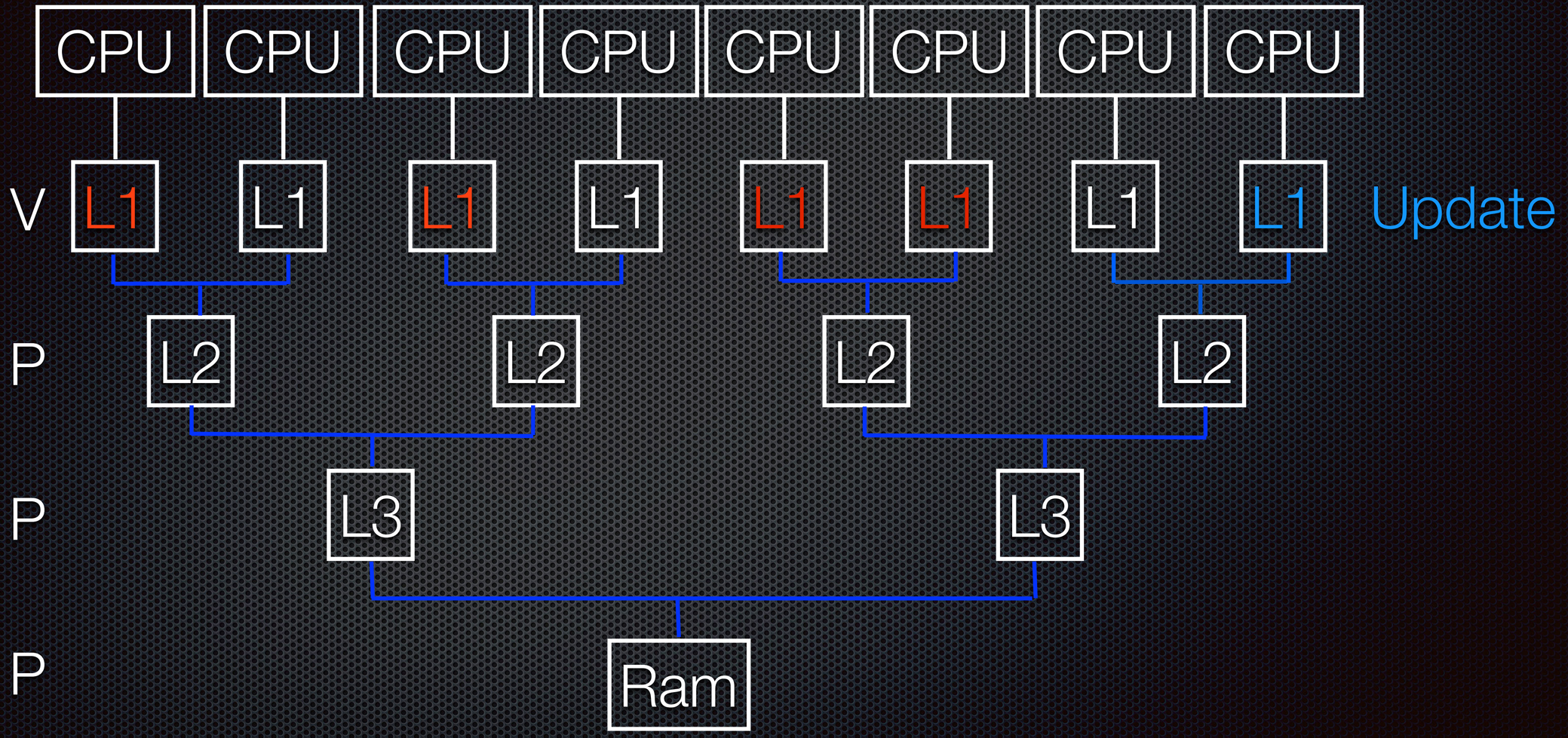
Multicore


CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
L1
L1
L1
L1
L1
L1
L1
L1
L2
L2
L2
L2
L3
L3
Ram

Synonyms


CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
V
L1
L1
L1
L1
L1
L1
L1
L1
P
L2
L2
L2
L2
P
L3
L3
P
Ram

Coherence Traffic



Every L2 has to reverse map to invalidate all synonyms

All Physical Caches

- With enough shared memory cores, synonym handling can produce significant overhead
- Some parallel applications have few synonyms, and sometimes OS scheduling can avoid high traffic through placement
- But for applications like databases, it is better to avoid them
  - Requires very fast translation and sophisticated but fast TLB above L1

Michel Dubois IEEE TC 08

The Synonym Lookaside Buffer: A Solution to the Synonym Problem in Virtual Caches

(Almost) Never Translate

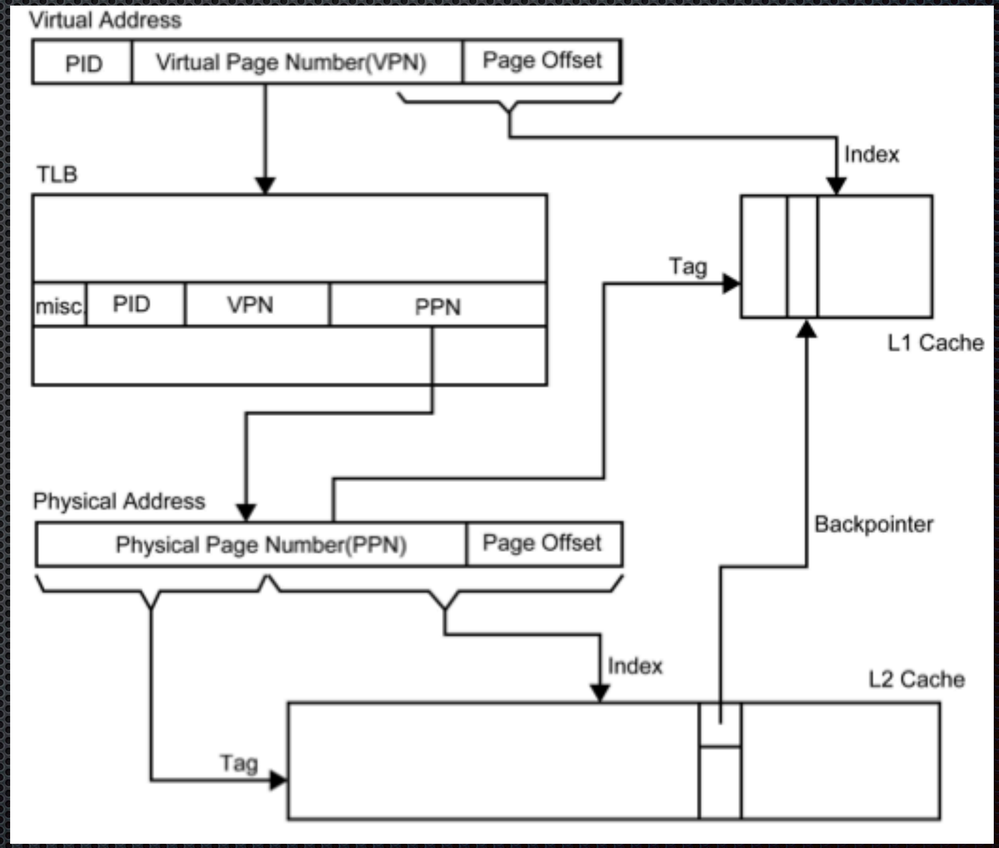
- As number of cores and amount of sharing increases, more time is spent resolving synonyms
- But TLB is on the fast path
- So stay virtual, and explicitly resolve synonyms (SLB)
- Scales with actual sharing, rather than address space

A Closer Look At Synonyms

- Multiple virtual addresses mapped to same physical
- Only one per page
- Must be at same address in different pages
- Used for sharing kernel, libraries, data, etc.
- Supports copy on write, to avoid duplicating read-only pages until one owner makes a change
- Allows message passing by remapping buffers

Virtual L1 Physical L2 Cache

Miss in L1 that hits in L2 and indicates a synonym -- L2 backpointer points to place in L1, and causes a short miss (internal copy) with L2 update

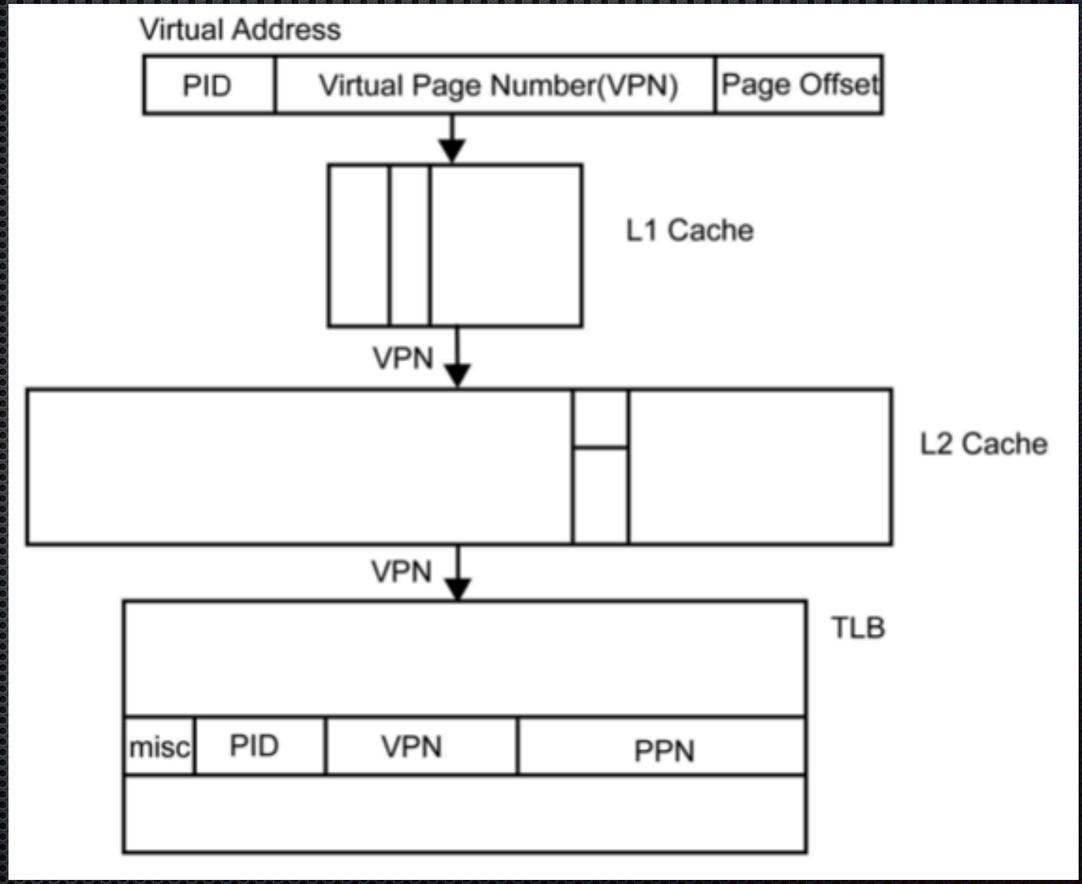


Multicore

- TLBs in different cores contain duplicate entries
- Copies can be inconsistent, need to be shot down
- Complicated by variable-size superpages

Virtual L1 and L2

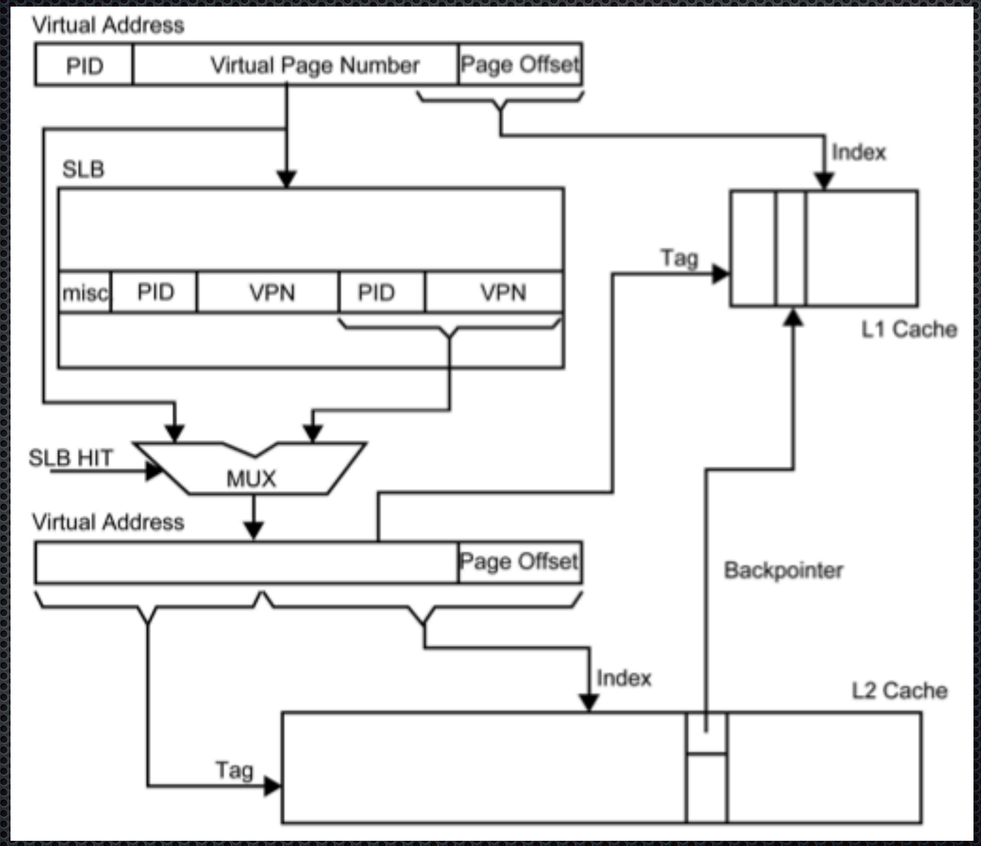
- TLB accessed only on L2 miss
- Could be moved even lower



Synonym Lookaside Buffer

- One per core, checked in parallel with L1 access
- Primary virtual address acts like physical address
- Secondary addresses translated to primaries
- SLB is like TLB, but maps secondary to primary VAs
- L1 and lower levels are tagged with primary addresses

SLB Organization



SLB Operation

- SLB hits only for secondary addresses -> primary
- SLB miss checks L1 cache in parallel
  - L1 hit gets primary, L1 miss signals secondary
  - Address goes to L2 -- hit there is primary, and backpointer causes short miss
  - L2 miss checks lower levels
    - Hits above TLB are primary, so no SLB change
    - TLB resolves synonyms, may trigger SLB update

SLB Actions

SLB Hit	L1 Hit	TLB Address	Action
Yes	Yes	Primary	Not possible
Yes	Yes	Secondary	L1 Hit. Value is returned from L1 after translation in SLB
Yes	No	Primary	Not possible
Yes	No	Secondary	L1 Miss. Value is returned from lower level caches/memories
No	Yes	Primary	L1 Hit. Value is returned from L1
No	Yes	Secondary	Not possible
No	No	Primary	L1 Miss. Value is returned from lower level cache/memories
No	No	Secondary	Trap processor. Refill SLB. Retry.

TLB shootdown avoided because caches above TLB are all virtual. Remapping of virtual pages causes overhead, especially if remapping primary page

An All Virtual Memory

- Each page gets a unique ID (primary virtual address)
- Synonyms have secondary virtual addresses
- Page tables need to keep track
- Allocate pri first allocation is temporar

PRIMARY VIRTUAL ADDRESSES	SECONDARY VIRTUAL ADDRESSES	PHYSICAL ADDRESSES
W	--	P1
X	X1, X2, X3	P2
Y	--	P3
Z	Z1	P4

TLB vs. SLB misses

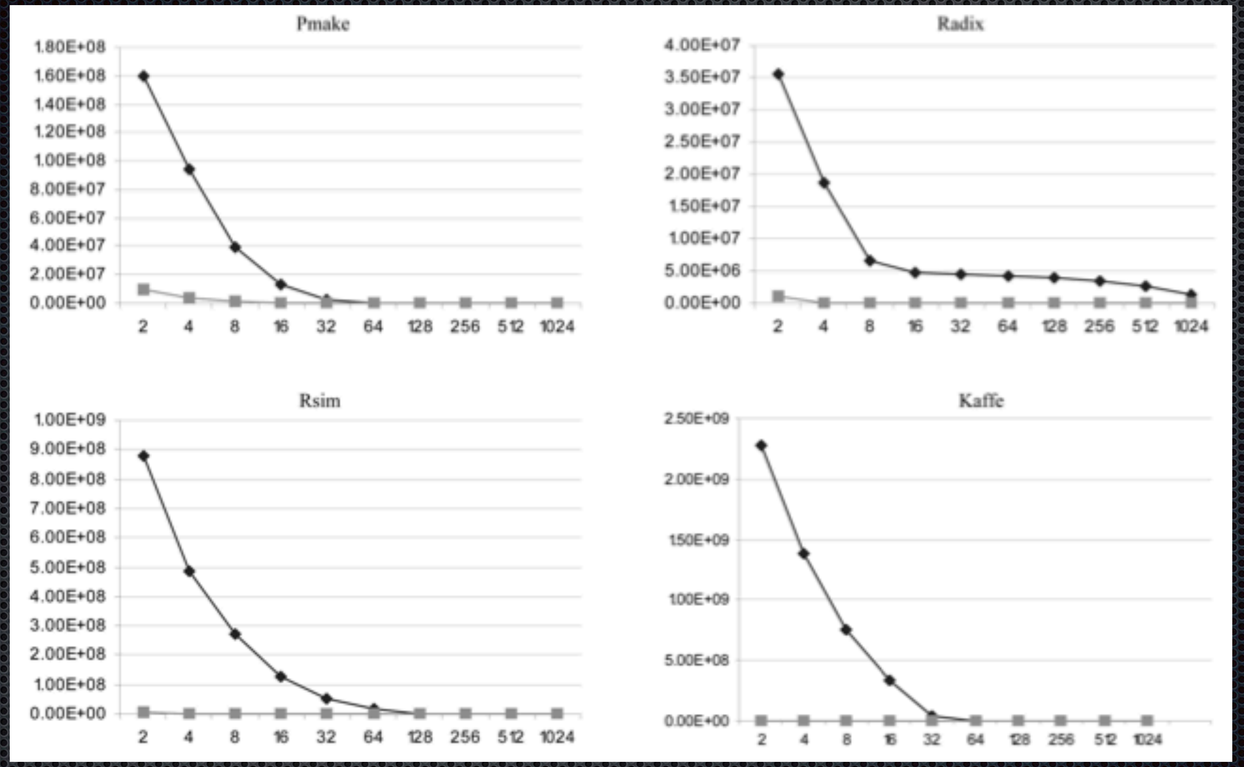
Benchmarks	TLB(32entries)	SLB(16entries)	Ratio
Pmake	2419455	241011	10
Radix	4170193	5246	795
Rsim	45285839	2738	16540
TPC-C	178191107	58404	3051
Kaffe	33613434	4665	7205

SLB Size

- Most hits are ca

Benchmarks	hit(16)	hit(>16)	miss
Pmake	99.5627	0.4097	0.0275
Radix	99.9697	0.0036	0.0267
Rsim	99.9956	0.0006	0.0038
TPC-C	99.9873	0.0053	0.0074
Kaffe	99.9831	0.0034	0.0136

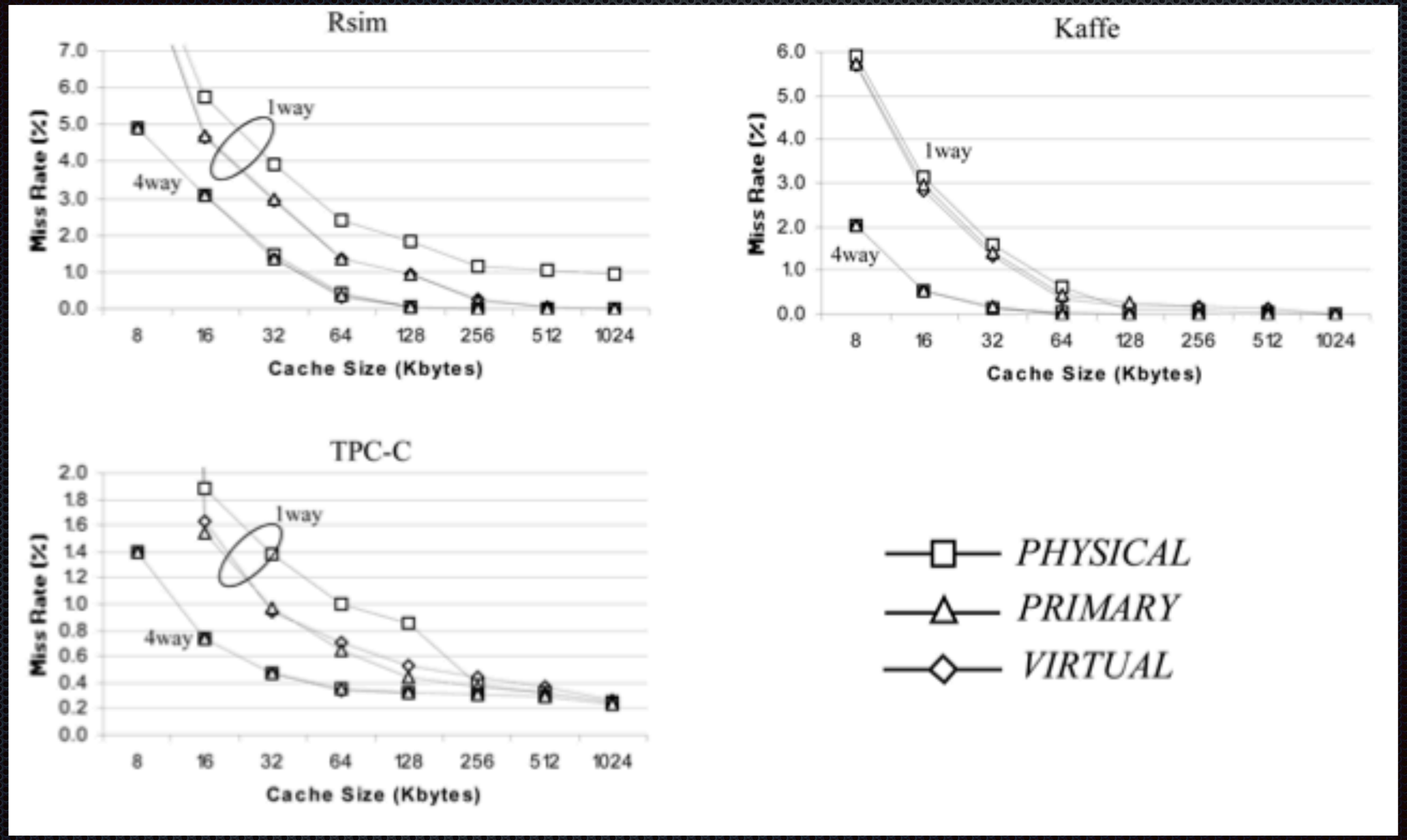
TLB / SLB Misses by Size



Fixed data set size

L1 Miss Rate

Ovals indicate difference traced to conflict misses



Discussion

Park, ISCA16

Efficient Synonym Filtering and Scalable Delayed Translation for Hybrid Virtual Caching

System Overview

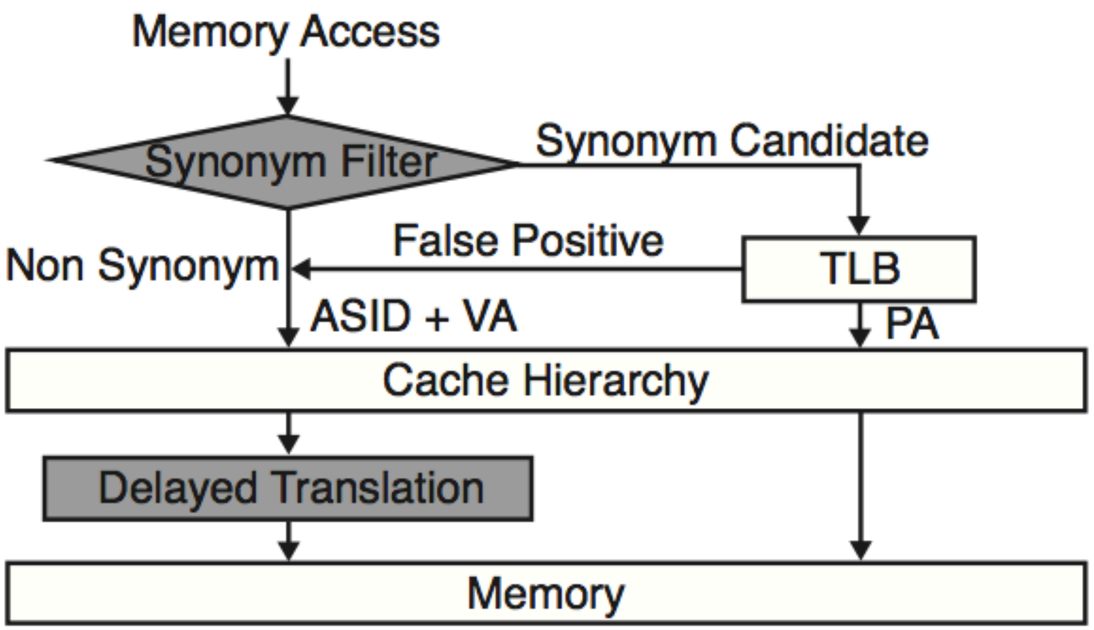


Figure 1. A synonym filter detects synonym candidates and allows non-synonym addresses to bypass translation until LLC miss. The components in gray are newly added by this work.

Sharing is Rare

	Shared area	Shared access
ferret	0.004543%	0.849976%
postgres	66.753033%	15.788043%
SpecJBB	0.328539%	0.023681%
firefox	0.754580%	0.001840%
apache	9.402985%	0.444037%
SPECCPU	0%	0%
Remaining Parsec	0%	0%

Hybrid Cache

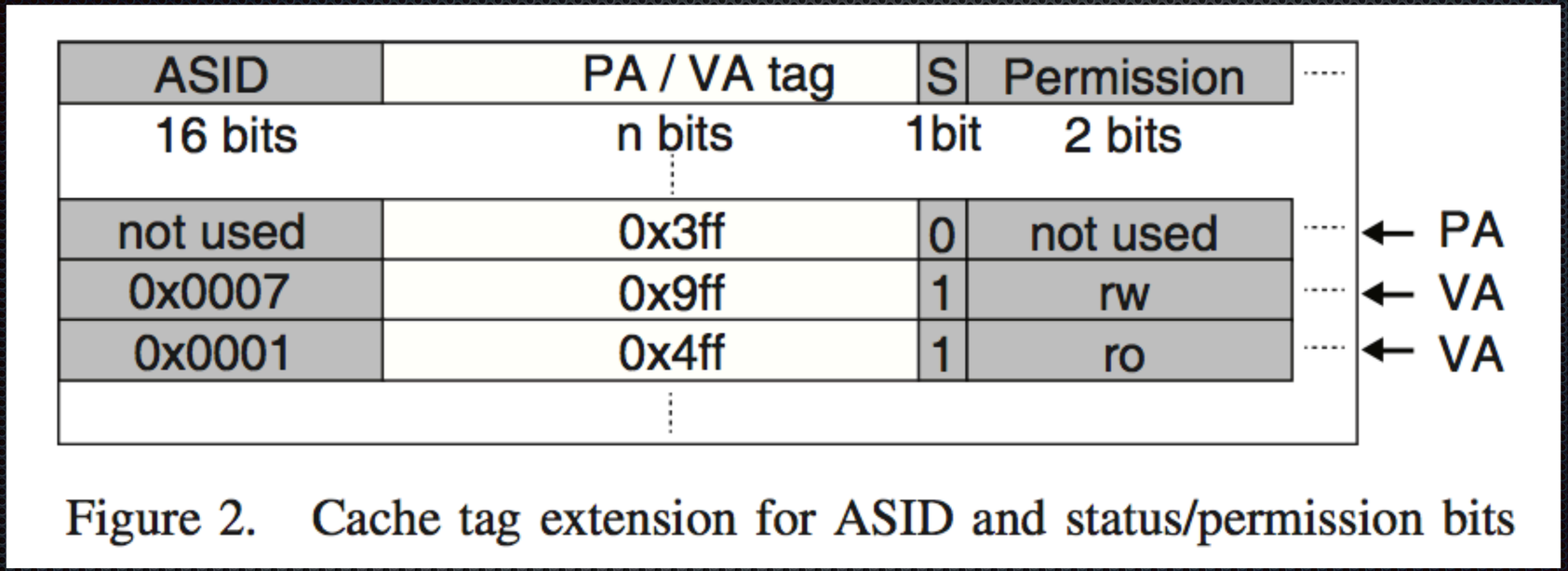


Figure 2. Cache tag extension for ASID and status/permission bits

S tag indicates physical (synonym) or virtual (non-synonym)

Operation

- Synonym filter is checked
- Non-synonyms access L1 cache by virtual address
  - A miss at L1 can result in a translation via a TLB
- Filter and cache can be accessed simultaneously
- Synonym candidates trigger TLB lookup
  - If found, L1 is accessed by physical address
  - If non-synonym, L1 virtual address is used
  - If a miss, translation is done

Synonym Filter

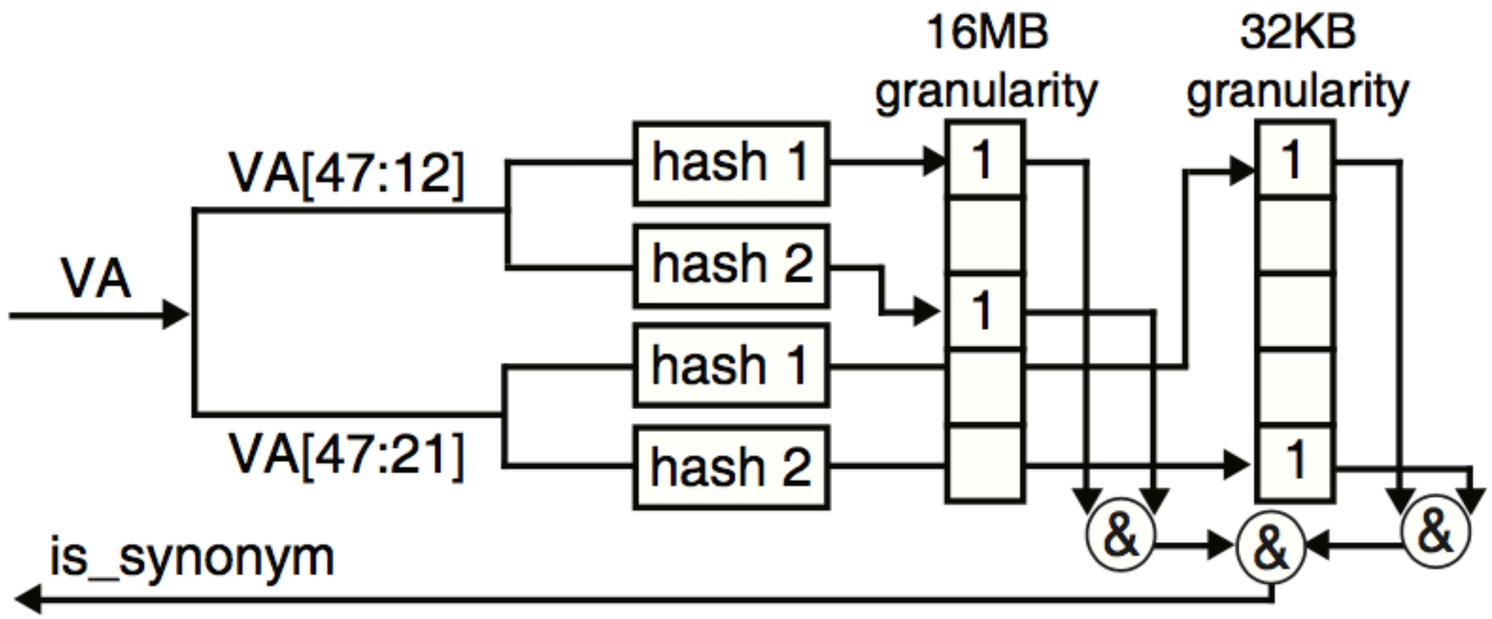


Figure 3. A synonym filter is composed of two Bloom filters each with different granularities. Each filter uses two hash functions. The synonym filter only returns true, thus a synonym candidate when all four bits are set to one.

Effect on TLB

Table II
FALSE POSITIVE RATES, TLB ACCESS AND MISS REDUCTION

	false positive rates	TLB access reduction	total TLB miss reduction
ferret	0.000756%	99.1%	20.4%
postgres	0.427410%	83.7%	-6.1%
SpecJBB	0.000019%	99.9%	42.6%
firefox	0.521944%	99.4%	63.2%
apache	0.000000%	99.5%	69.7%

Application Variation

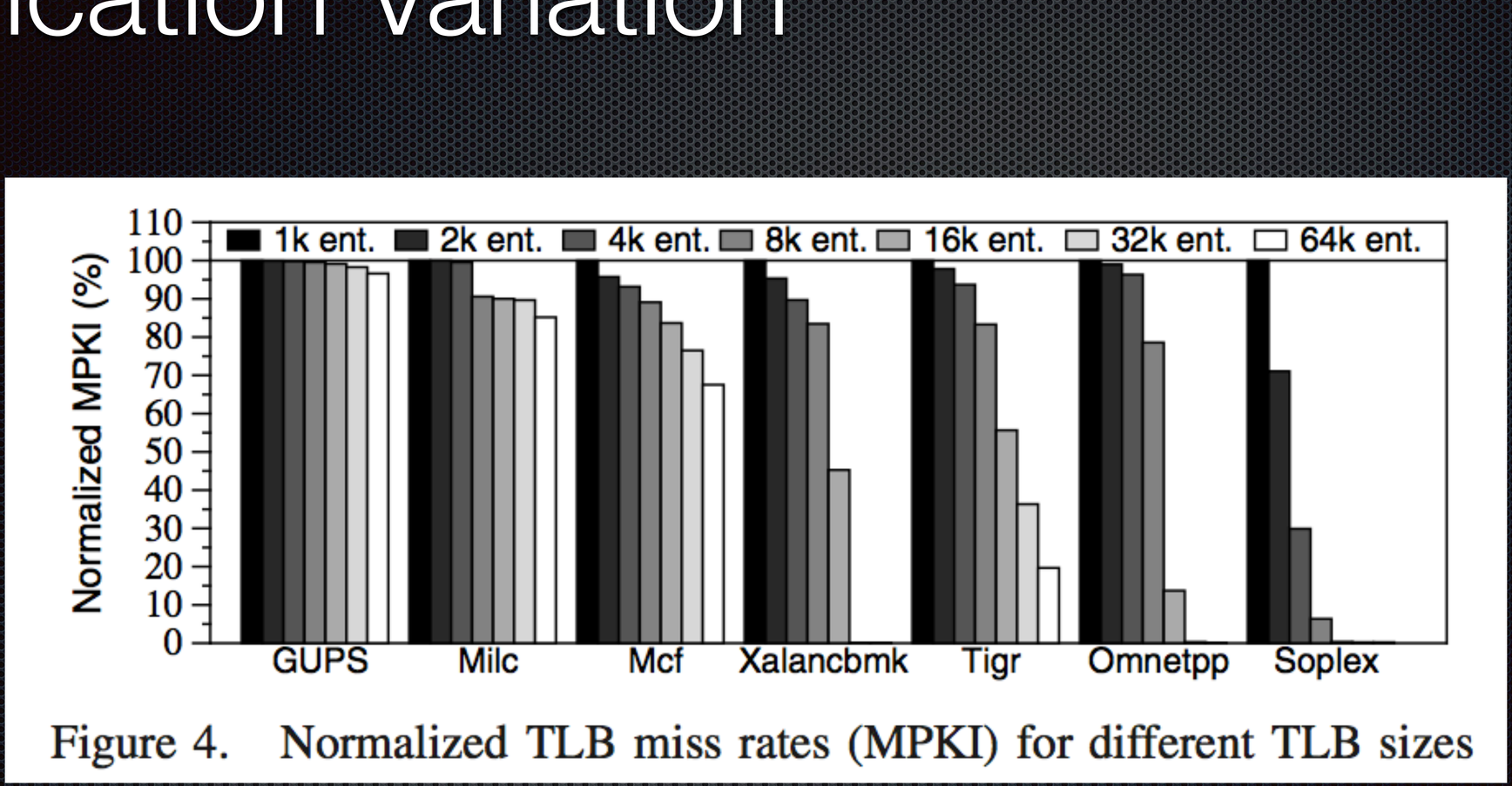


Figure 4. Normalized TLB miss rates (MPKI) for different TLB sizes

Applications with heavy sharing don’t benefit

Fewer Segments than Pages

Table III
MAXIMUM NUMBER OF SEGMENTS IN USE BY EACH APPLICATION, RMM MPKI, AND MEMORY UTILIZATION

Bench.	RMM [12]	Reproduced	MPKI	Usage(%)
astar	33	16	0	96.5
mcf	28	4	0	72.5
omnetpp	27	106	0.02	99
cactus.	70	56	0	83.2
GemsFD.	61	143	0	100
xalancbmk	N/A	244	0.13	96.5
canneal	46	22	0	84.7
stream.	32	16	0	24.7
mummer	61	3	0	100
tigr	167	90	22.93	99.5
memcached	86	839	0.08	100
NPB:CG	95	5	0	100
gups	62	7	0	99.9

Misses Per Kilo-Instructions

Many-Segment Architecture

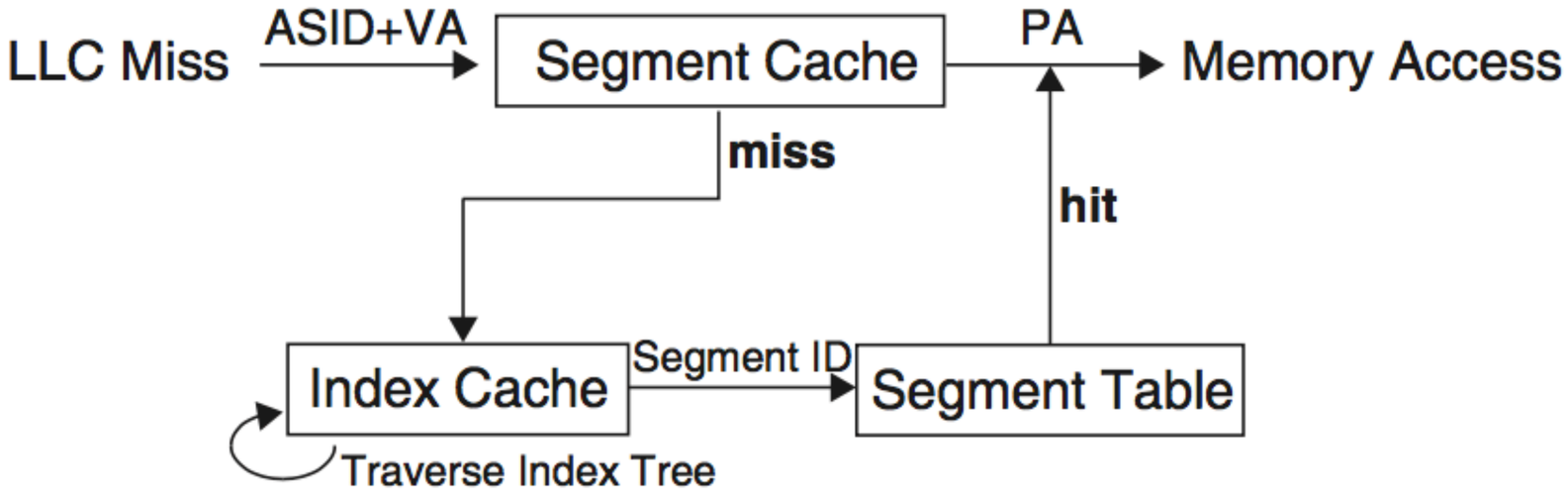


Figure 5. Scalable delayed translation: The overall translation flow from ASID+VA to PA

Extended Segment Table

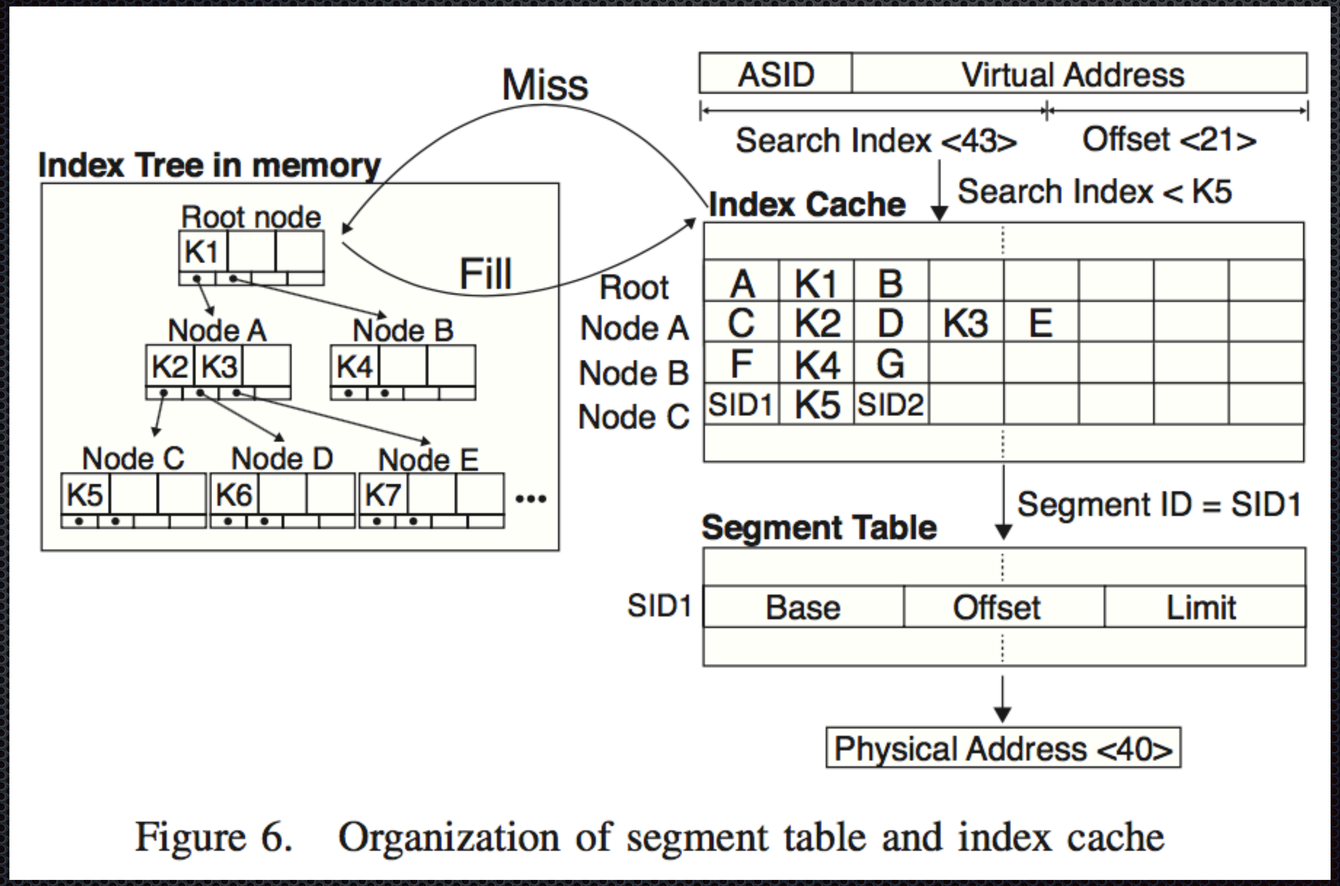


Figure 6. Organization of segment table and index cache

Caches part of the OS's B-Tree for Indexes

Index Cache Size Study

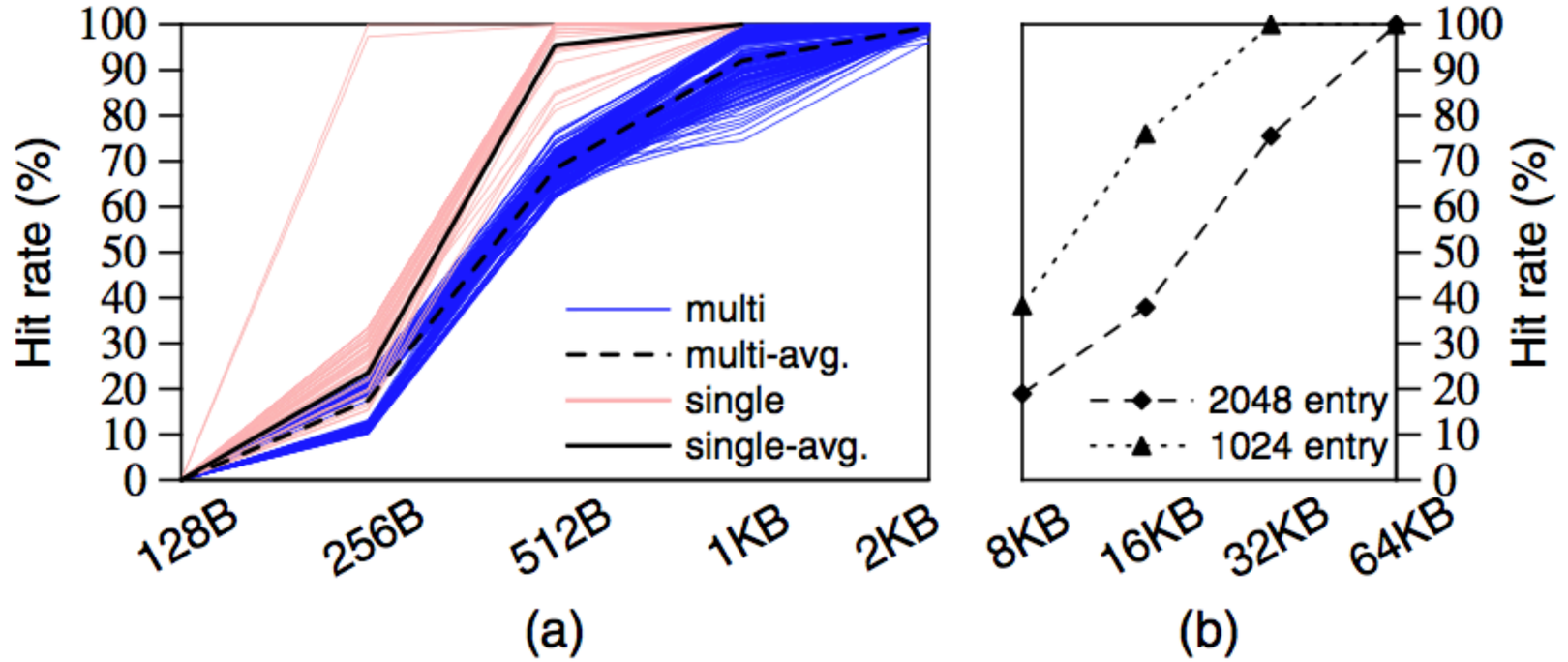


Figure 7. Index cache size sensitivity study. (a) shows index cache hit rate for actual workloads. (b) shows synthetic worst case benchmark.

Performance

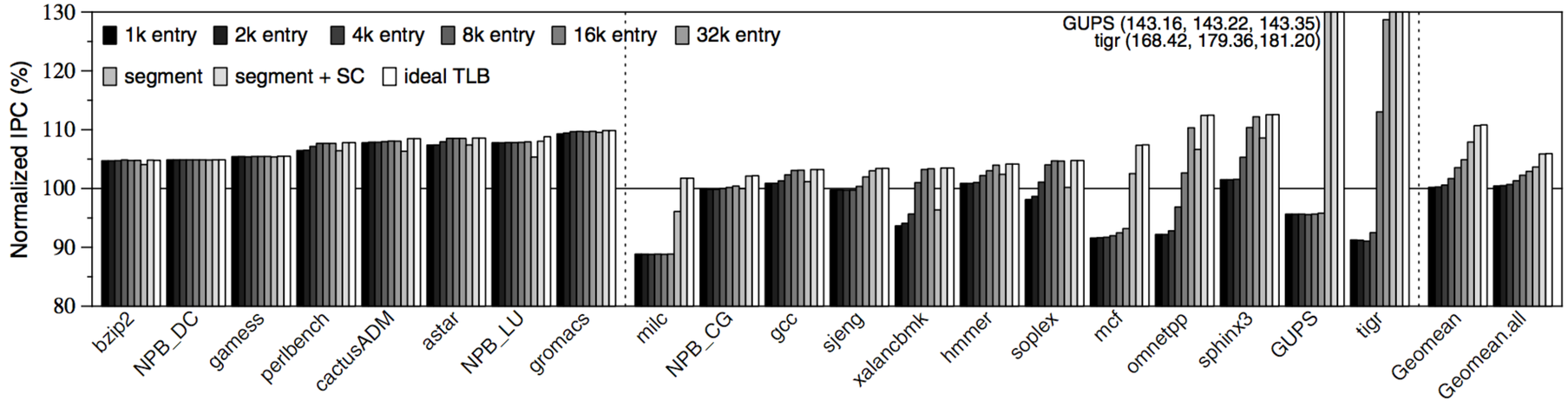


Figure 9. Normalized performance of our proposed system to the baseline system. Workloads on the left are workloads which benefit from delayed translation. The results in the center show workloads in which fixed granularity delayed translation causes overhead but improves as more delayed TLB entries are provided.

Using MARSSx86 and DRAMSim2

Power

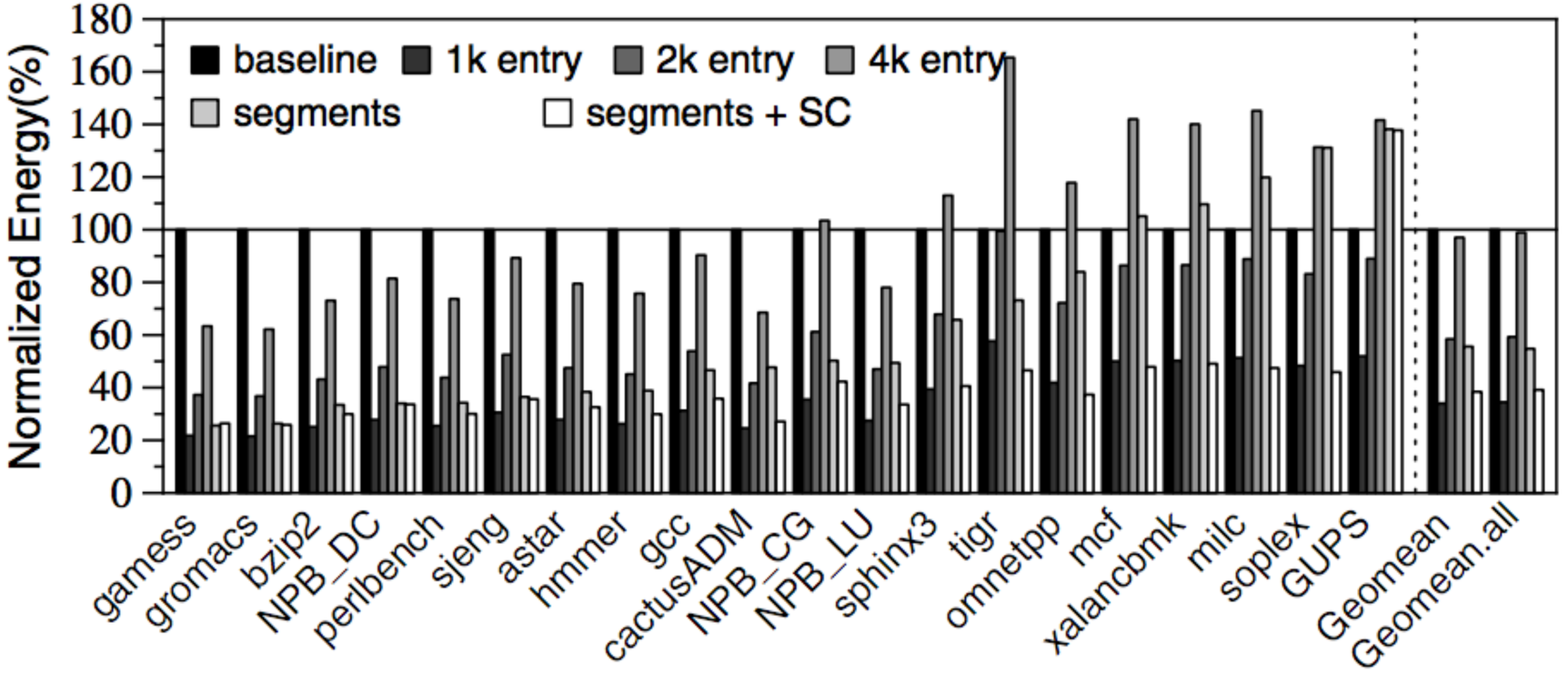


Figure 11. Normalized power consumption of delayed translation over baseline system (power consumption by translation components)

Discussion

Yaniv - Sigmetrics 16

Hash, Don’t Cache (the Page Table)

Overview

- Prior work claimed that Radix Page Tables with Page Walk Caches outperform Hashed Page Tables
- That model was based on the Itanium hashing scheme, which was suboptimal
- An optimized hashing scheme can give better performance
- The difference grows with virtualization
- Radix tables won’t scale as well as hashing in the future

Radix Page Table Structure

9-bit portions of VPN each select a table entry

Each table entry points
to the next table in the tree

CR3 points to top level table

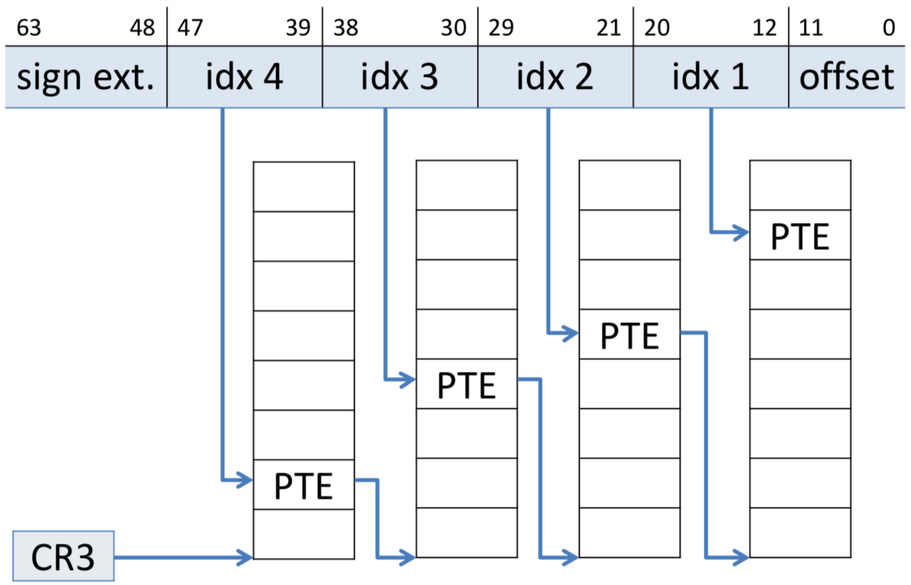


Figure 1: Bare-metal radix page table walk.

63 52	51 12	11 0
0's padding	PPN	attributes

Table 1: Radix PTE structure.

Virtualization Creates a 2D Table

Each step of the guest walk causes a host walk

The guest OS has to walk its table

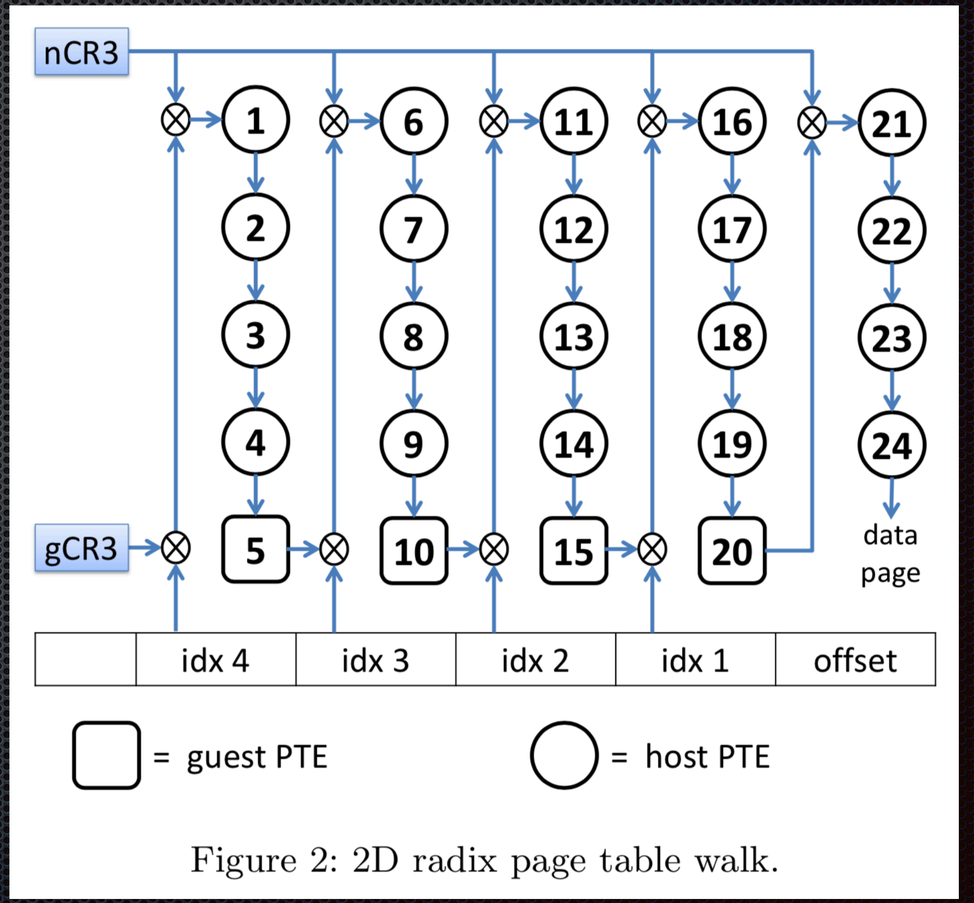


Figure 2: 2D radix page table walk.

Costs 24 memory accesses

Cache the walk

Use the lowest level cached walk portion available

Apply caching in two dimensions

If lucky, then most of walk is avoided

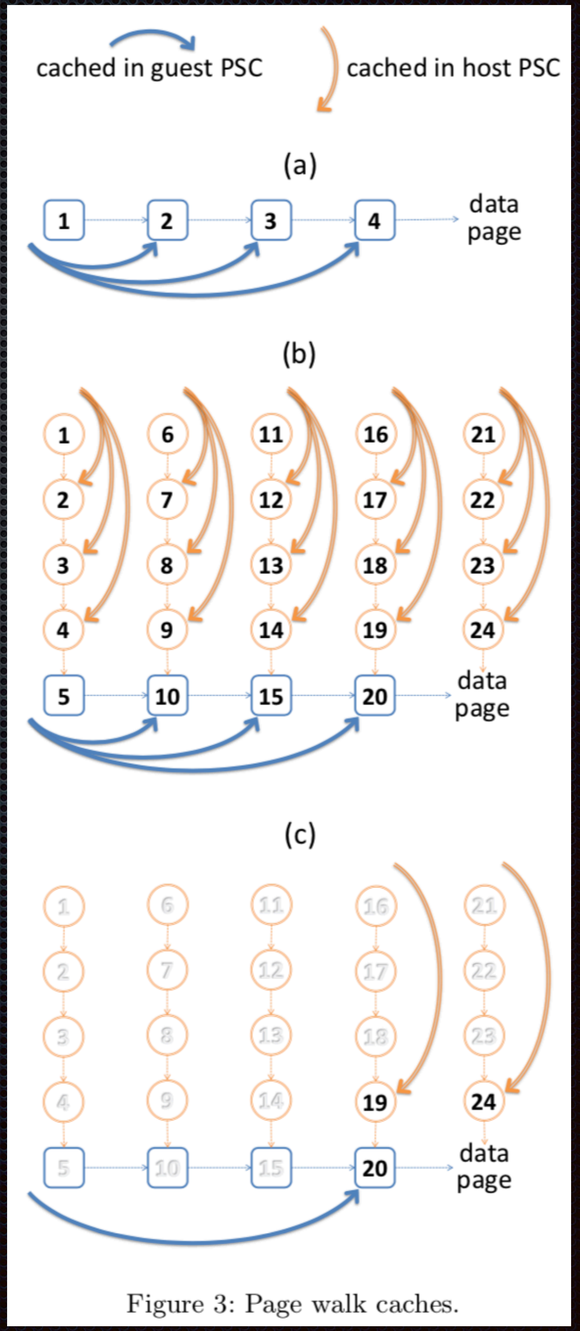


Figure 3: Page walk caches.

Other Radix Benefits

- Entries are grouped so spatially local translations are effectively prefetched
- The indexing scheme is very simple
- The tables are compact

Hashing Itanium Style

VPN hashes into table

Assumes a collision

Points to chain table

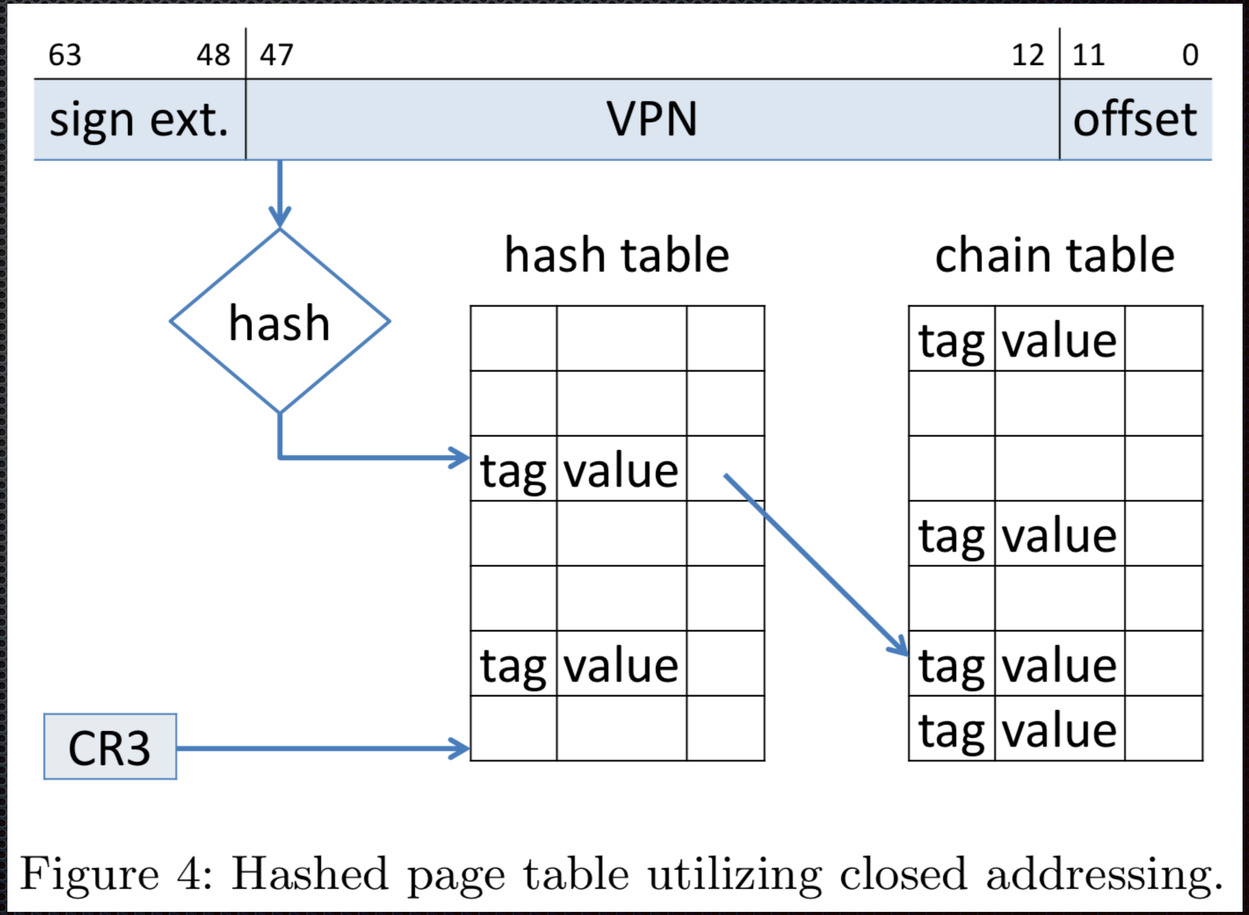


Figure 4: Hashed page table utilizing closed addressing.

Hashing with Open Addressing

Assume collisions are rare

Just go to next slot

Search until empty slot found

Avoids chain tables

Reduces PTE size

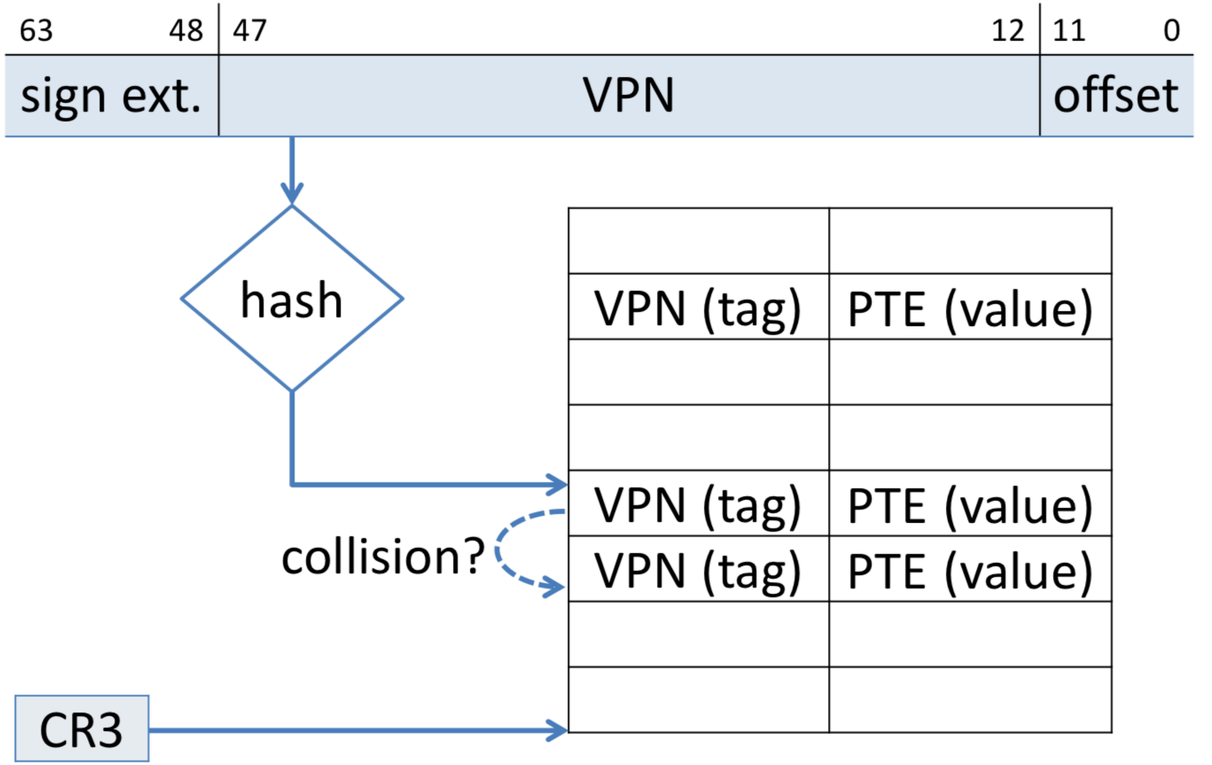


Figure 5: Hashed page table utilizing open addressing.

More Optimizations

- Reorganize to cluster entries
  - Achieves locality advantage
- Scavenge bit from entries to form shared tag, enabling twice as many entries

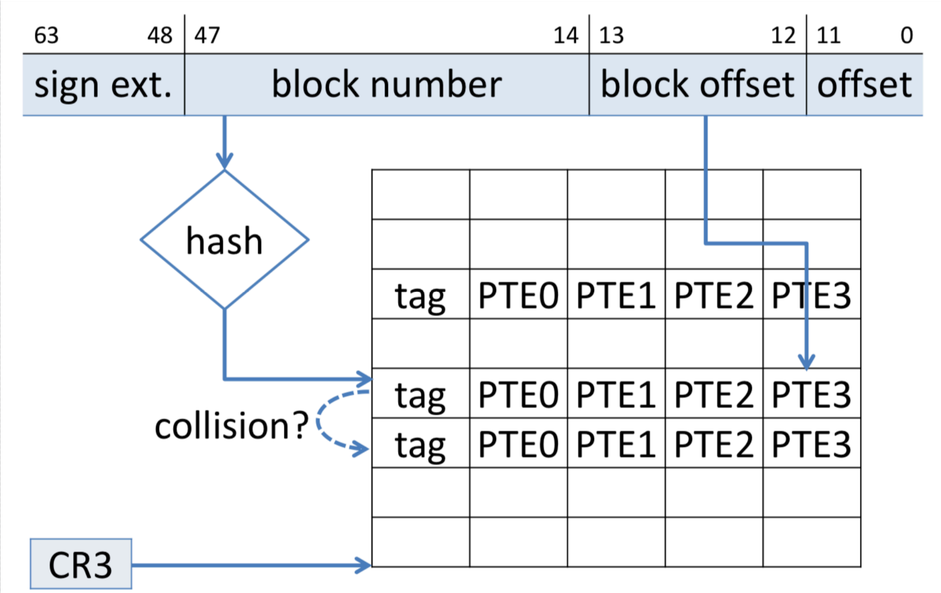


Figure 7: Clustered page table walk.

8 B	7 B	7 B		7 B	7 B
tag	PTE0	PTE1	. . .	PTE6	PTE7

Table 2: Compact cluster of PTEs.

Greatly Reduces 2D Walks

- Guest and host benefit from different clustering factors
- Need to minimize number of pages for guest table
- Increase clustering to reduce cache footprint

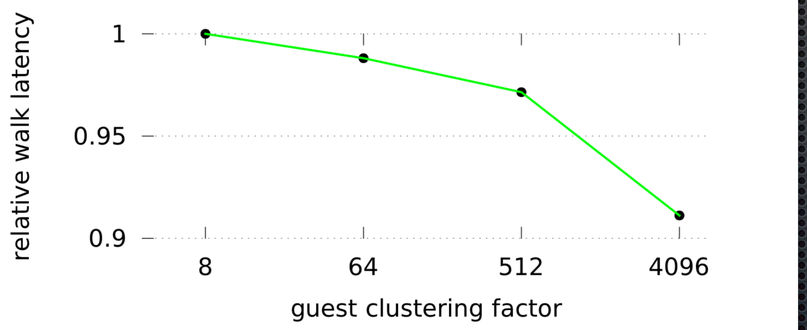


Figure 9: The walk latency of 2D hashed page table decreases as the guest clustering increases.

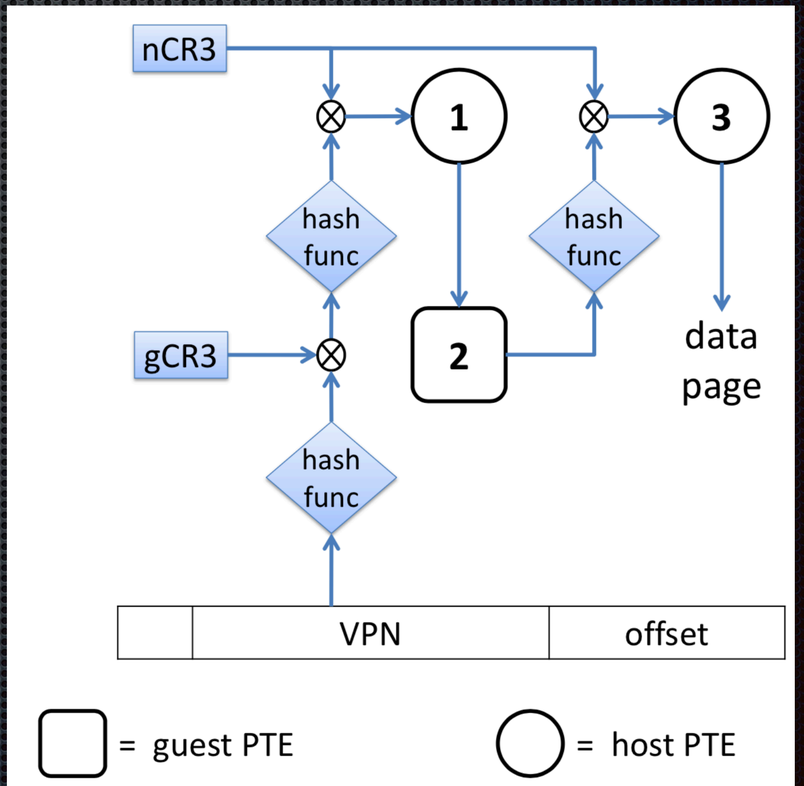


Figure 8: 2D hashed page table walk.

Performance

- Works well for TLB intensive workloads
- Marginal effectiveness for low stress workloads
- Never worse for these tests
- Simulation based on traces and timing estimates
- No use of parallel workloads

benchmark	*bare-metal*		*virtualized*	
	perfect PWCs	*hashed paging*	*perfect PWCs*	*hashed paging*
SPEC mcf	-4%	-6%	-14%	-17%
SPEC cactusADM	-7%	-27%	-7%	-32%
SPEC xalancbmk	-1%	-2%	-7%	-9%
graph500 4GB	-1%	-1%	-6%	-6%
graph500 8GB	-2%	-1%	-8%	-7%
graph500 16GB	-2%	-2%	-10%	-8%
GUPS 2GB	-6%	-5%	-19%	-17%
GUPS 8GB	-11%	-8%	-30%	-24%
GUPS 32GB	-19%	-15%	-35%	-29%

Table 4: The runtime improvement achieved when utilizing (1) ideal PWCs and (2) hashed paging, as compared to radix paging with real PWCs. The improvement provided by hashed paging is comparable to that of ideal PWCs. The exception is cactusADM, for which hashed paging offers a much greater improvement due to radix caching pathologies.

Discussion