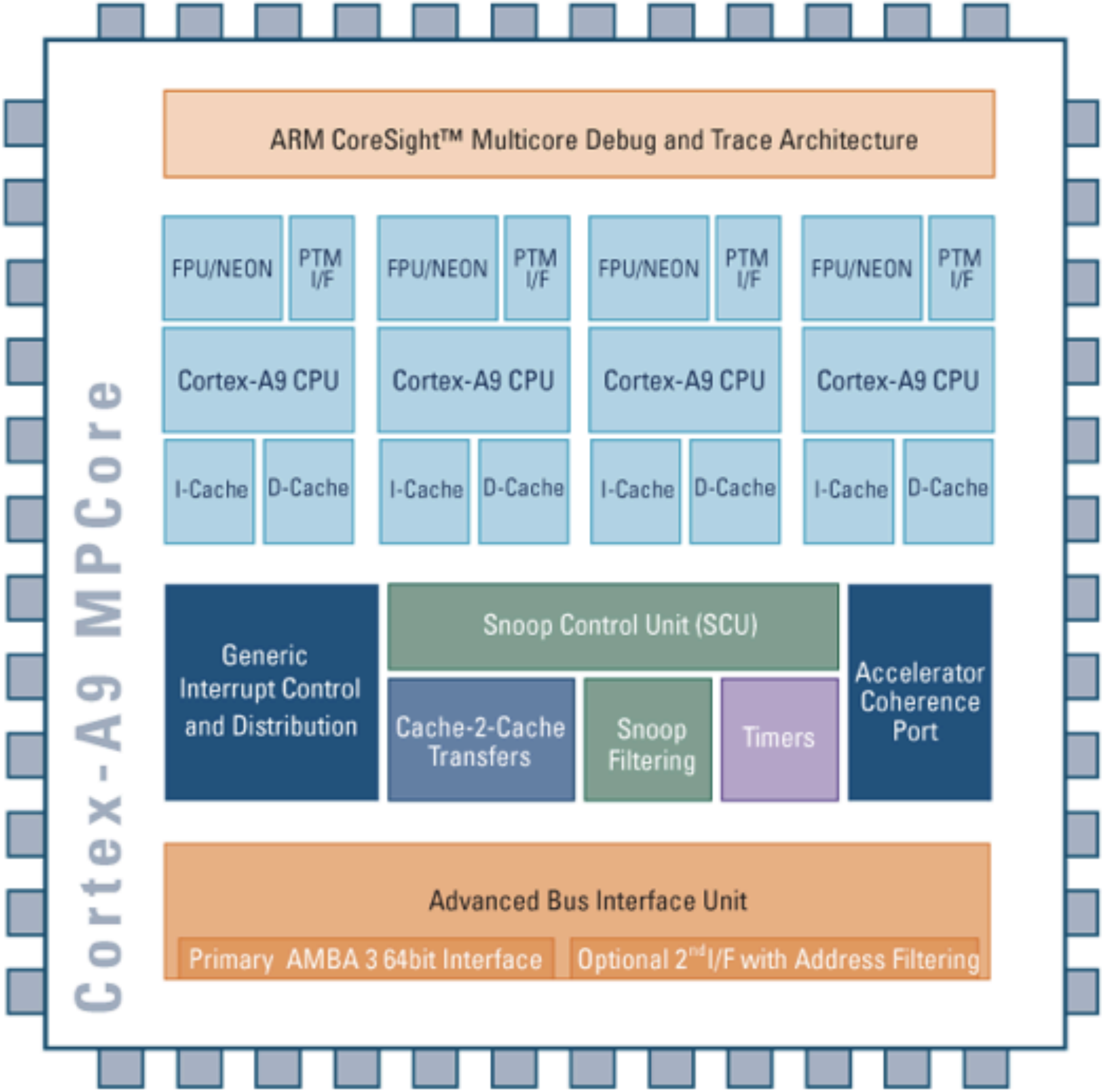
Caching and Virtual Memory

Last Time

- Flexible Address Translation
  - Segmentation + paged translation
  - Multi-level paged translation
  - Hashing
- Efficient Address Translation
  - Translation Lookaside Buffers (TLBs)
  - *Virtually addressed and physically addressed caches*

Main Points

- Cache concept
  - Hardware vs. software caches
- When caches work and when they don't
  - Spatial/temporal locality vs. Zipf workloads
- Cache replacement policies


Cortex-A9 MPCore
ARM CoreSight™ Multicore Debug and Trace Architecture
FPU/NEON
PTM I/F
FPU/NEON
PTM I/F
FPU/NEON
PTM I/F
FPU/NEON
PTM I/F
Cortex-A9 CPU
Cortex-A9 CPU
Cortex-A9 CPU
Cortex-A9 CPU
I-Cache
D-Cache
I-Cache
D-Cache
I-Cache
D-Cache
I-Cache
D-Cache
Generic Interrupt Control and Distribution
Snoop Control Unit (SCU)
Cache-2-Cache Transfers
Snoop Filtering
Timers
Accelerator Coherence Port
Advanced Bus Interface Unit
Primary AMBA 3 64bit Interface
Optional 2nd I/F with Address Filtering

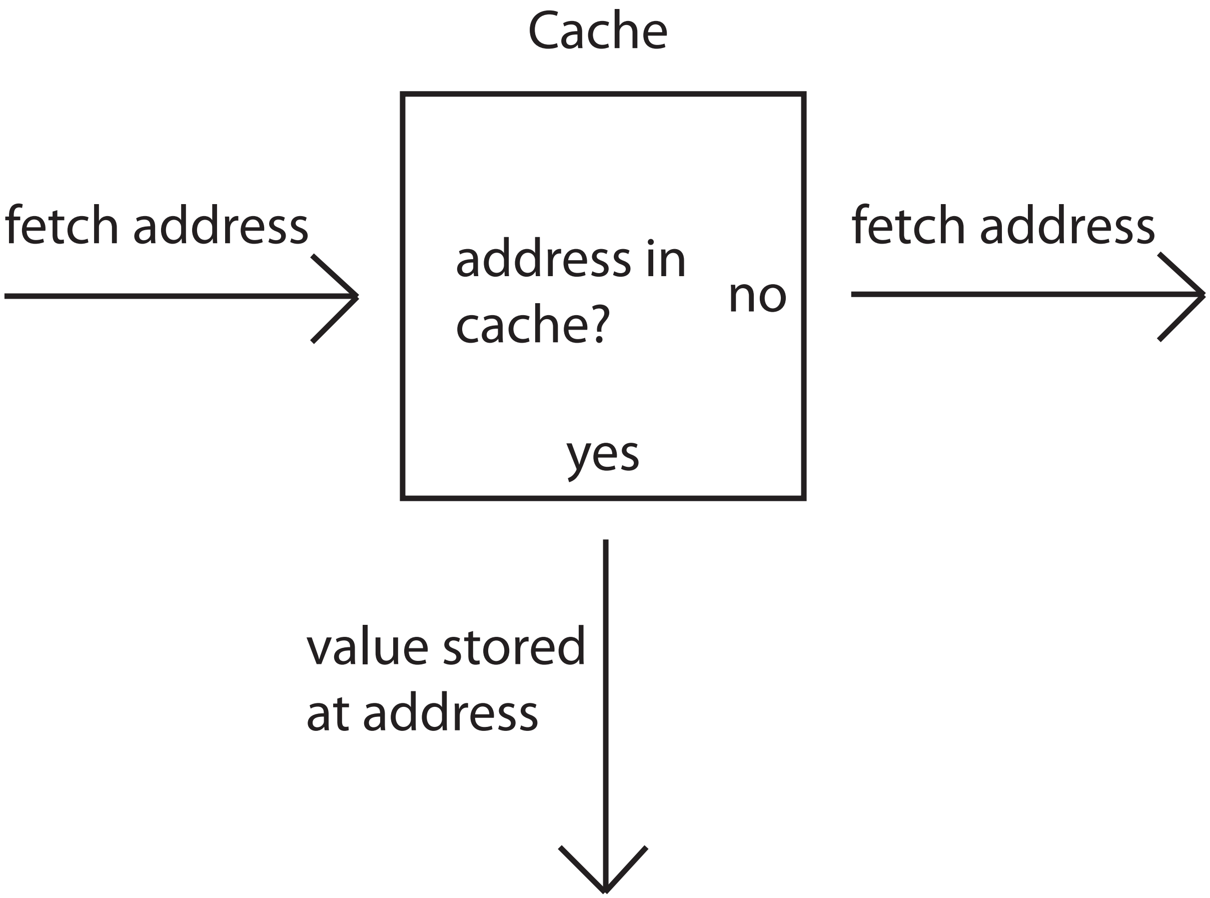
Multicore and Hyperthreading

- Modern CPU has several functional units
  - Instruction decode
  - Arithmetic/branch
  - Floating point
  - Instruction/data cache
  - TLB
- Multicore: replicate functional units (i7: 4)
  - Share second/third level cache, second level TLB
- Hyperthreading: logical processors that share functional units (i7: 2)
  - Better functional unit utilization during memory stalls
- No difference from the OS/programmer perspective
  - Except for performance, affinity, …

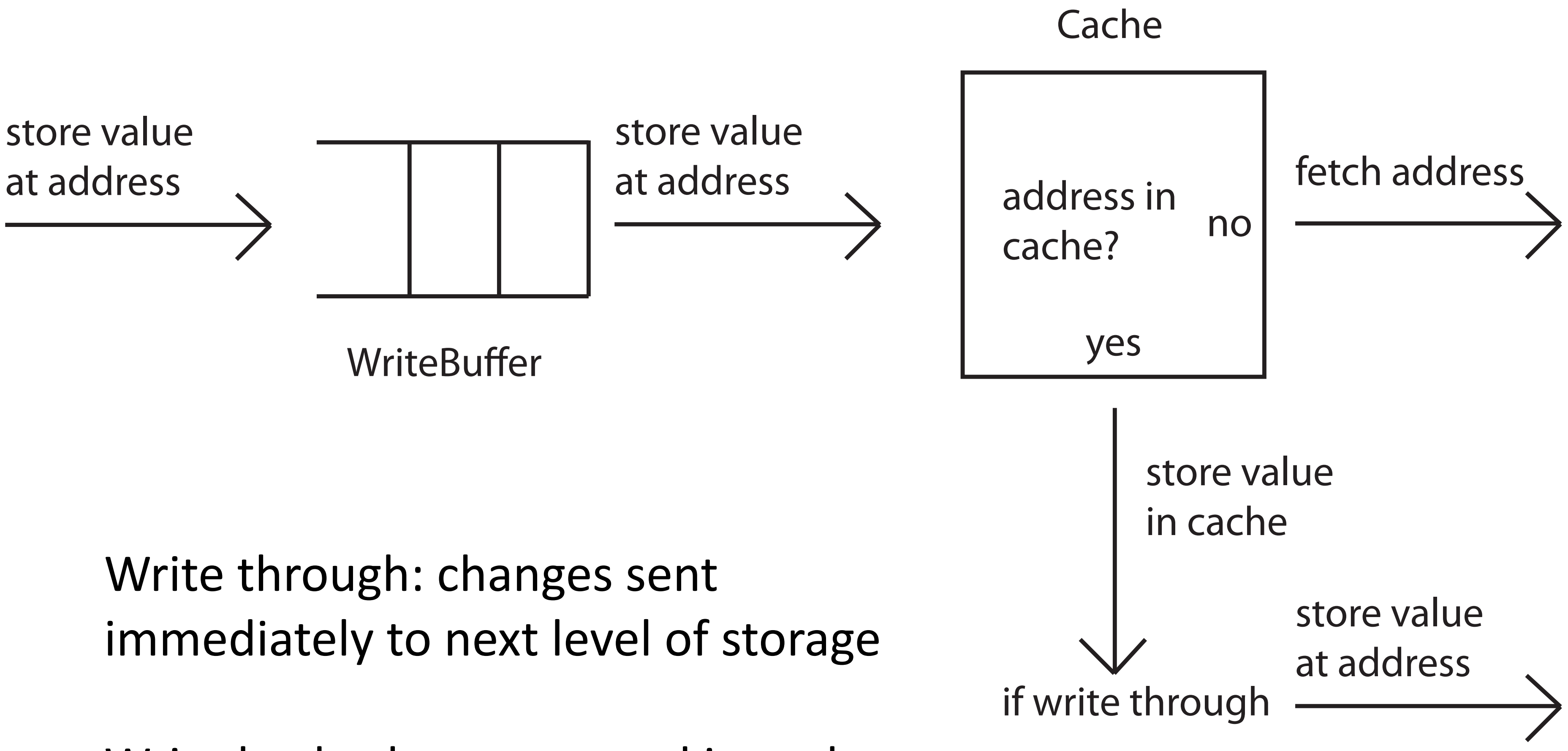
Definitions

- Cache
  - Copy of data that is faster to access than the original
  - Hit: if cache has copy
  - Miss: if cache does not have copy
- Cache block
  - Unit of cache storage (multiple memory locations)
- Temporal locality
  - Programs tend to reference the same memory locations multiple times
  - Example: instructions in a loop
- Spatial locality
  - Programs tend to reference nearby locations
  - Example: data in a loop

Cache Concept (Read)



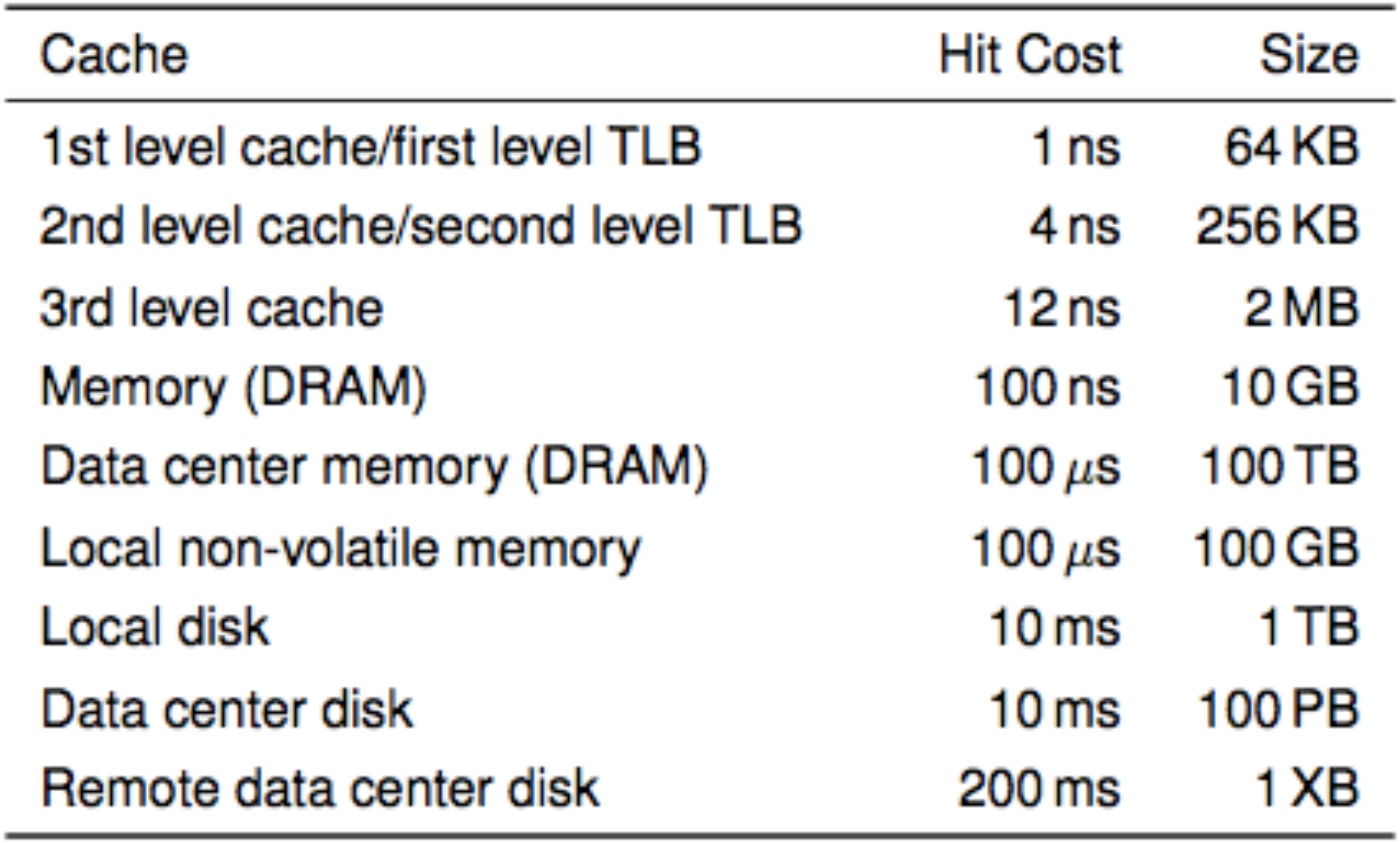
Cache Concept (Write)



Write through: changes sent immediately to next level of storage

Write back: changes stored in cache until cache block is replaced

Memory Hierarchy



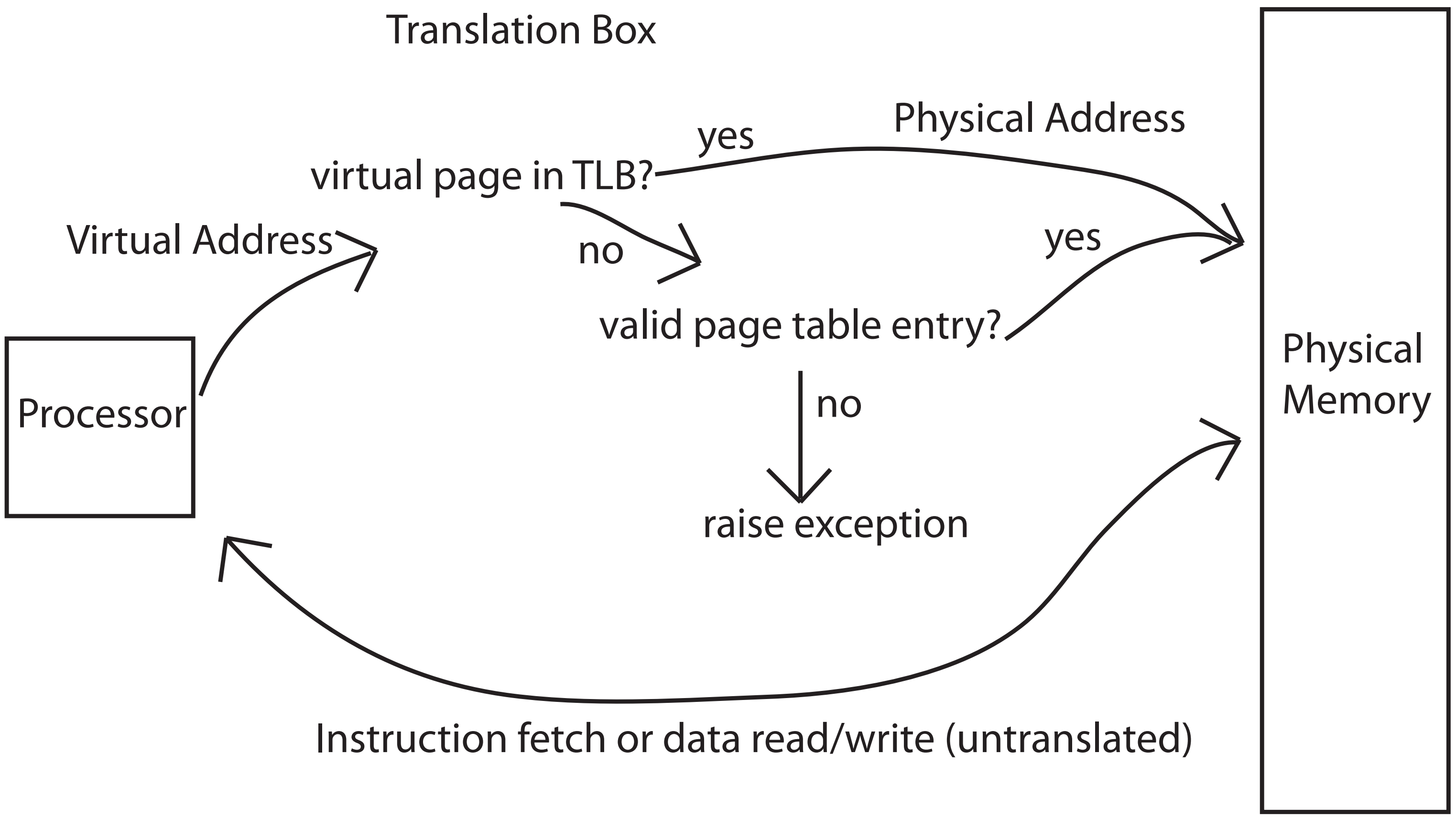
Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

Hardware Design Principle

The bigger the memory, the slower the memory

Address Translation with TLB



Translation Box

Processor

Virtual Address

virtual page in TLB?

yes

Physical Address

no

valid page table entry?

yes

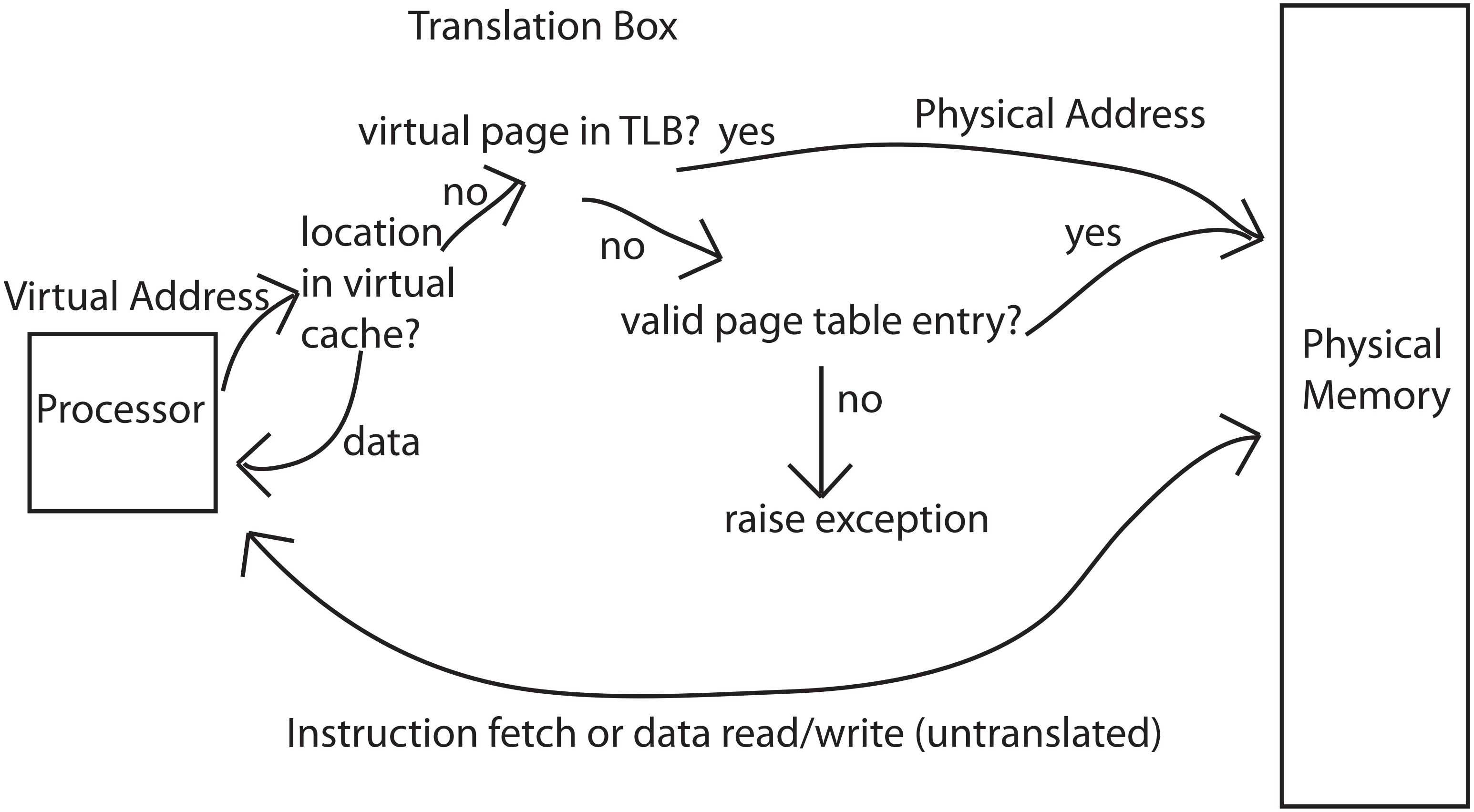
no

raise exception

Physical Memory

Instruction fetch or data read/write (untranslated)

Virtually Addressed Caches



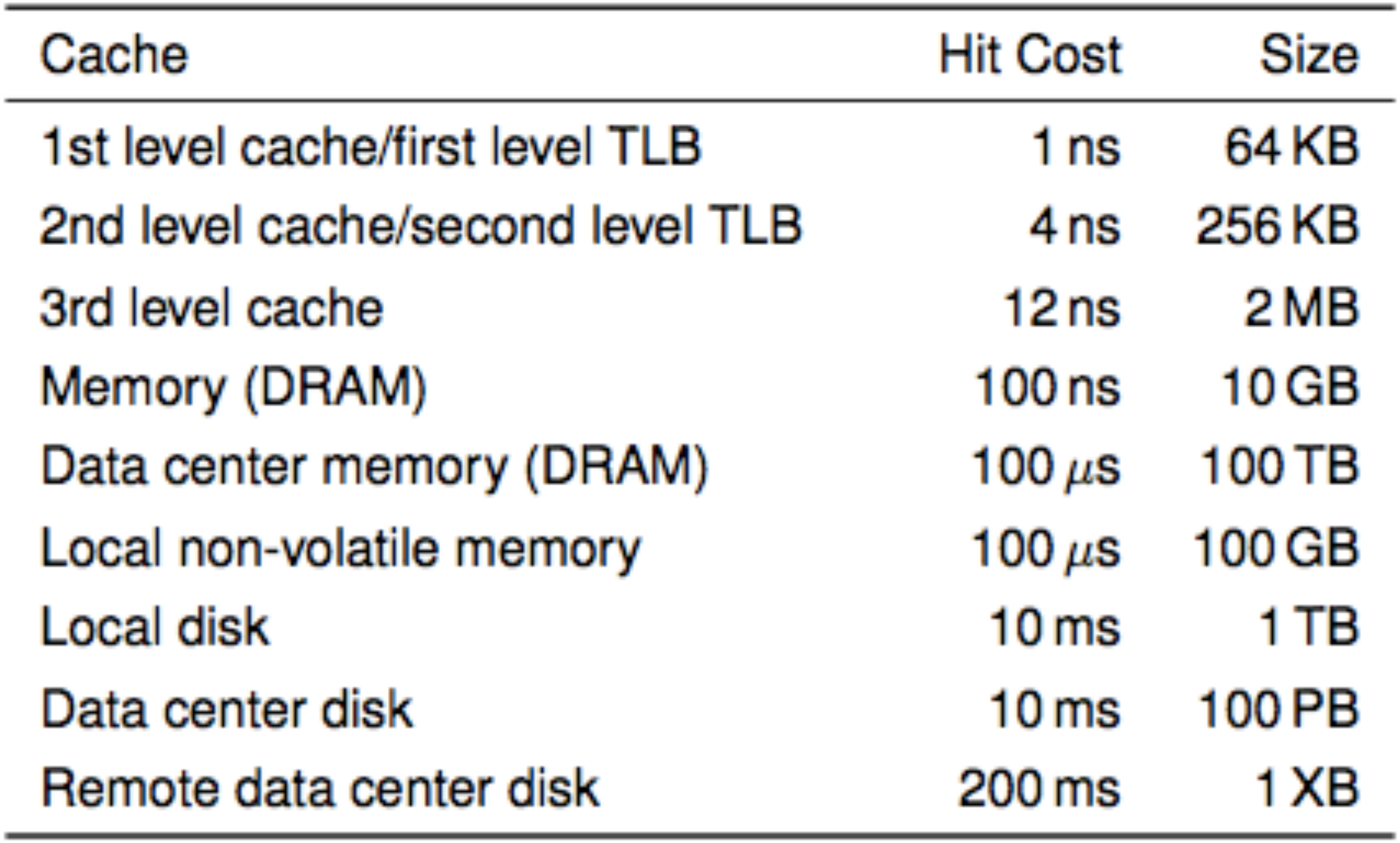
Questions

- With a virtual cache, what do we need to do on a context switch?
- What if the virtual cache > page size?
  - Page size: 4KB (x86)
  - First level cache size: 64KB (i7)
  - Cache block size: 32 bytes

Aliasing

- Alias: two (or more) virtual cache entries that refer to the same physical memory
  - What if we modify one alias and then context switch?
- Typical solution
  - On a write, lookup virtual cache and TLB in parallel
  - Physical address from TLB used to check for aliases

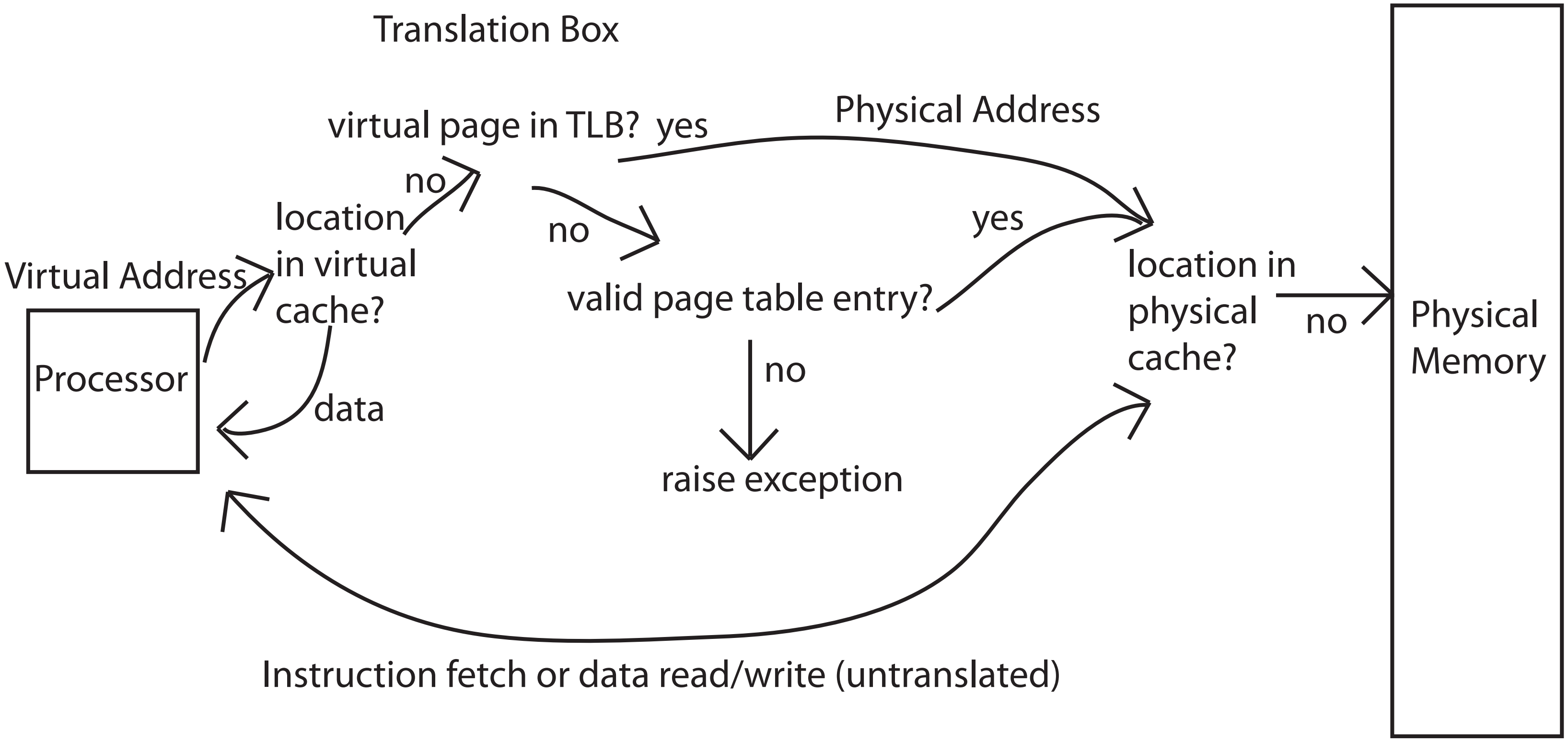
Memory Hierarchy



Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

Translation on a Modern Processor



Translation Box

virtual page in TLB? yes

Physical Address

no

location in virtual cache?

no

Virtual Address

valid page table entry?

yes

location in physical cache?

no

Physical Memory

Processor

data

no

raise exception

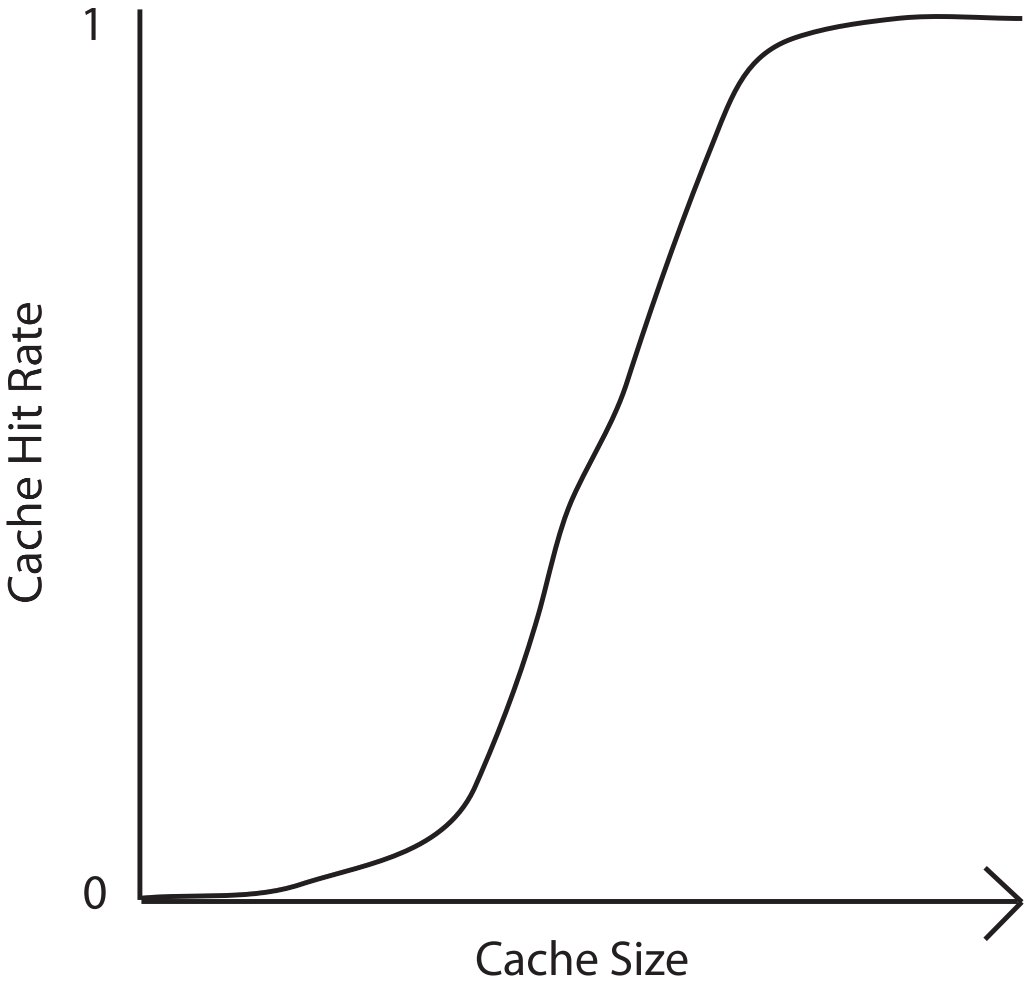
Instruction fetch or data read/write (untranslated)

Question

- What is the cost of a first level TLB miss?
  - – Second level TLB lookup
- What is the cost of a second level TLB miss?
  - – x86: 2-4 level page table walk
- How expensive is a 4-level page table walk on a modern processor?

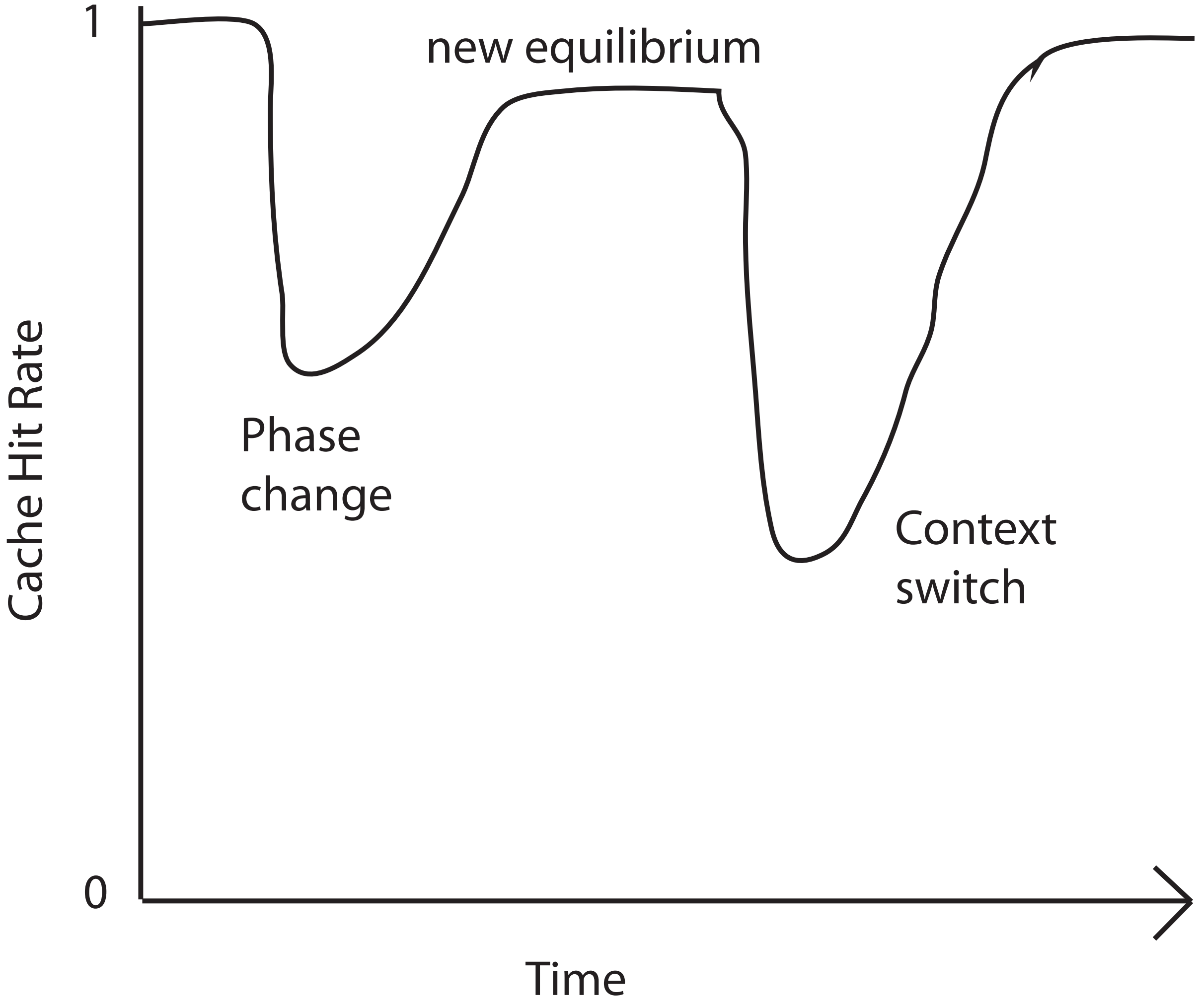
Working Set Model

- Working Set: set of memory locations that need to be cached for reasonable cache hit rate
- Thrashing: when system has too small a cache



Phase Change Behavior

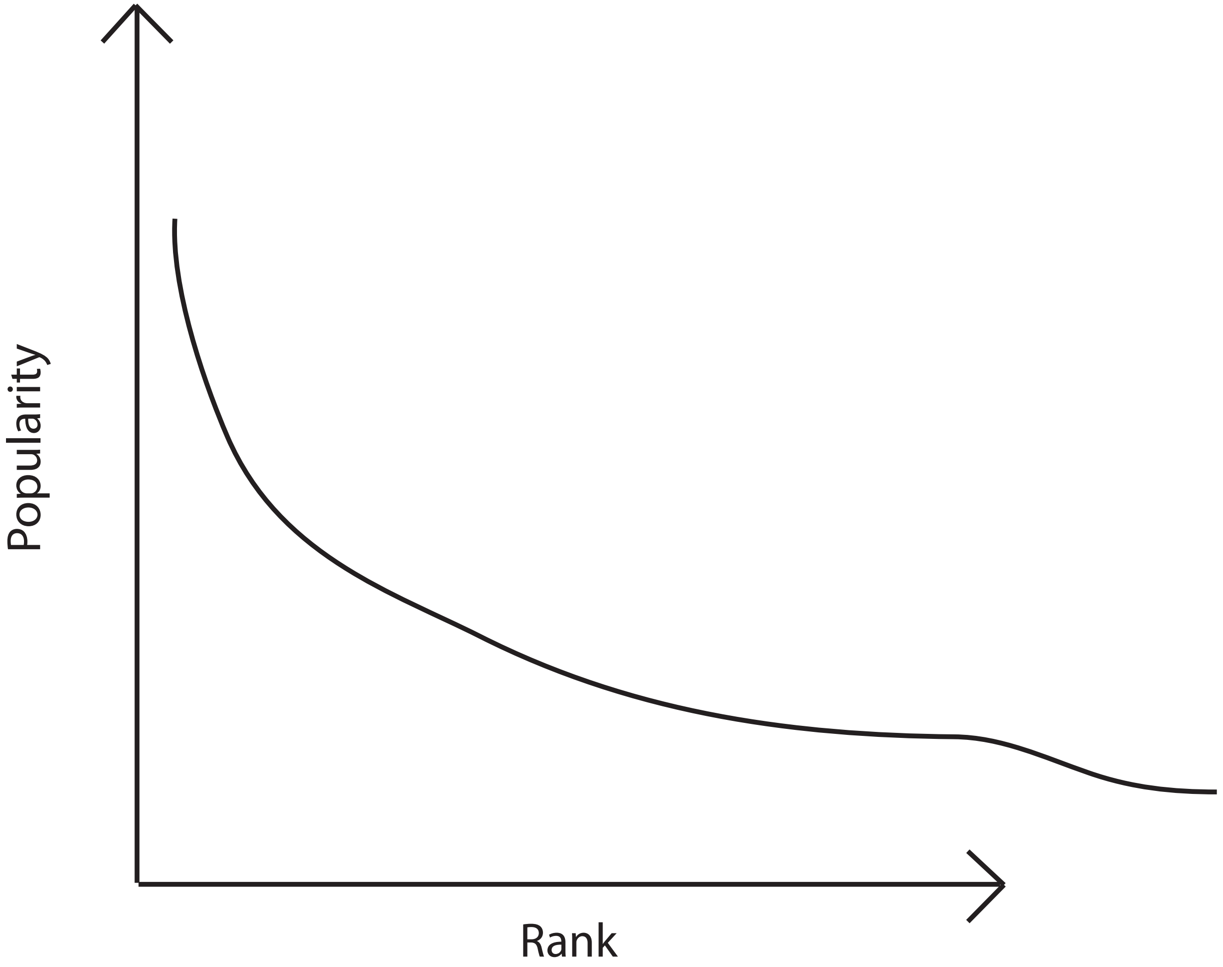
- Programs can change their working set
- Context switches also change working set



Zipf Distribution

- Caching behavior of many systems are not well characterized by the working set model
- An alternative is the Zipf distribution
  - Popularity ~ 1/k^c, for kth most popular item, 1 < c < 2

Zipf Distribution

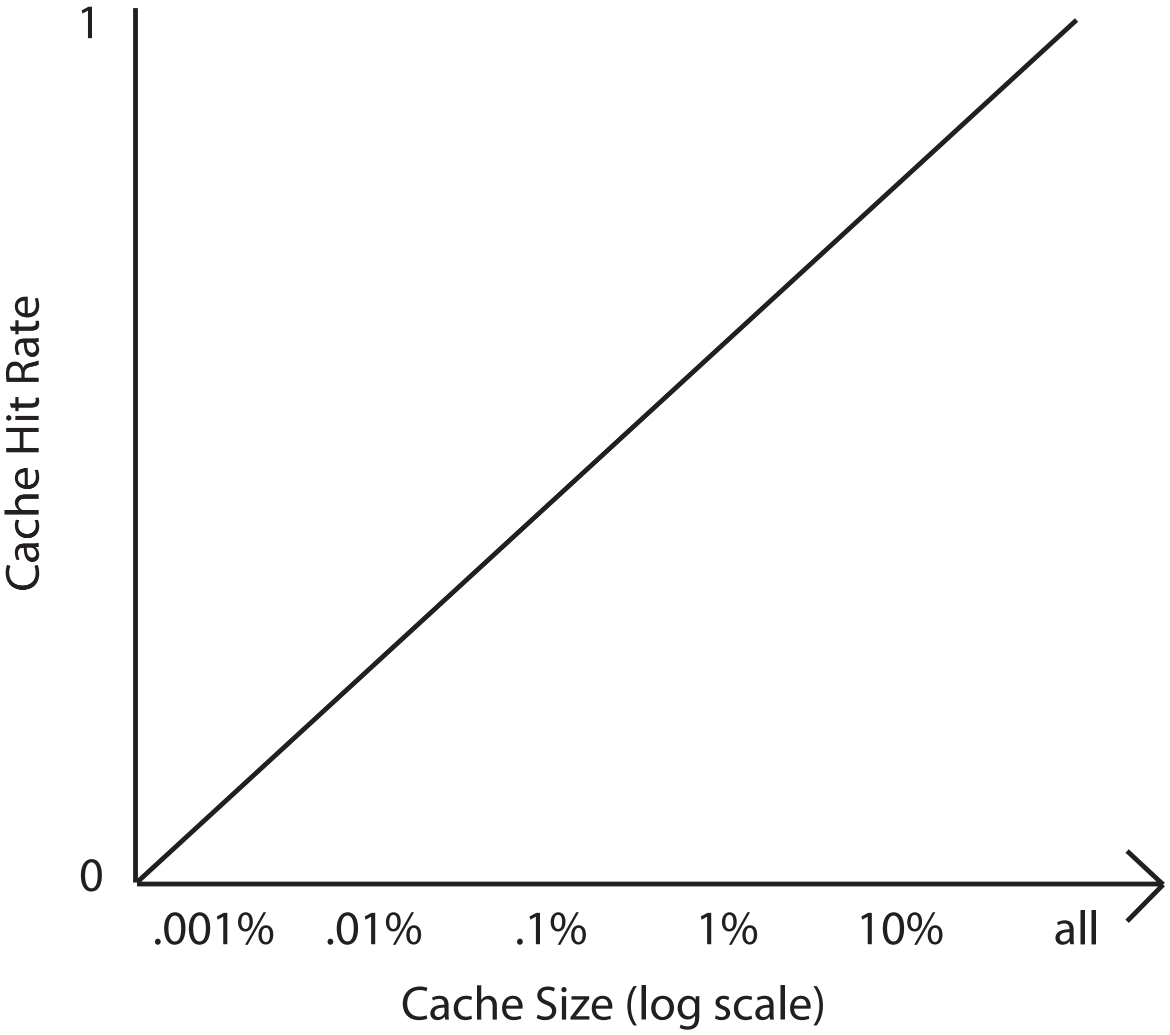


Zipf Examples

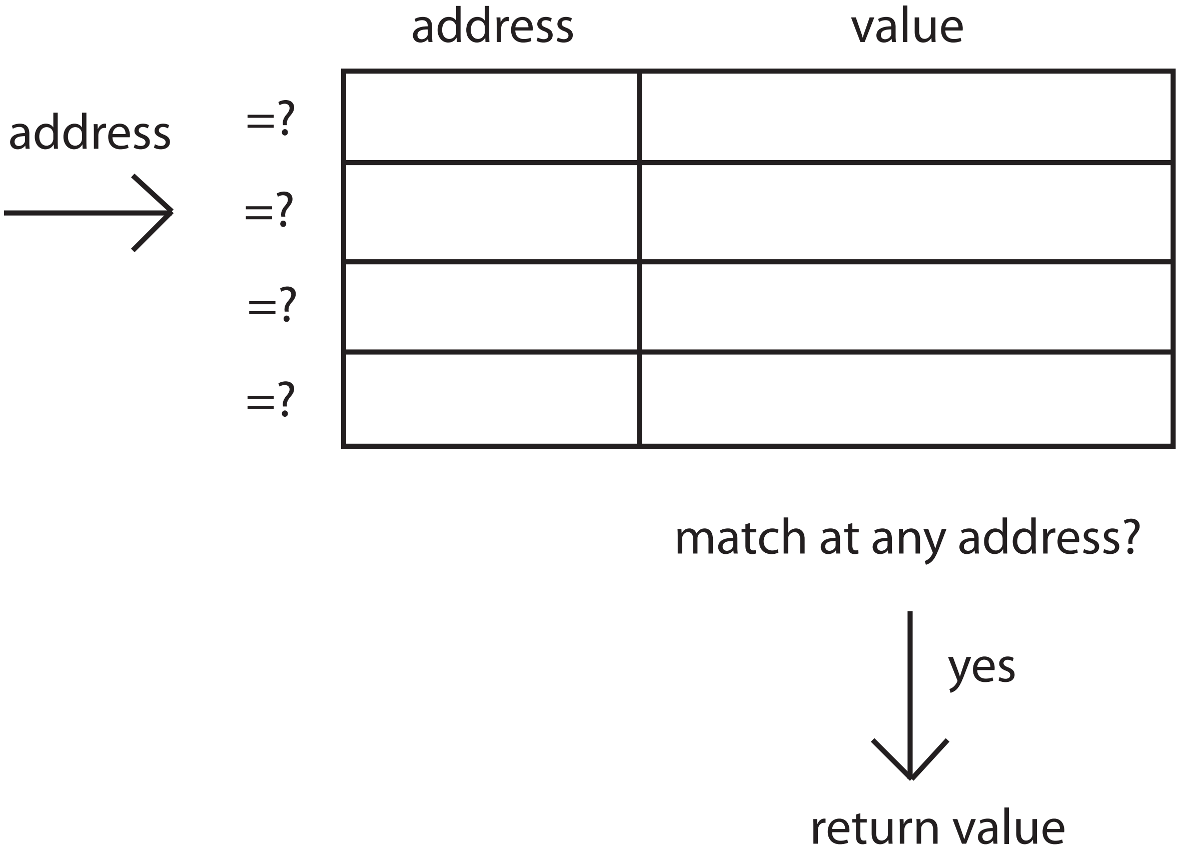
- Web pages
- Movies
- Library books
- Words in text
- Salaries
- City population
- …

Common thread: popularity is self-reinforcing

Zipf and Caching



Cache Lookup: Fully Associative



address

value

address

=?

=?

=?

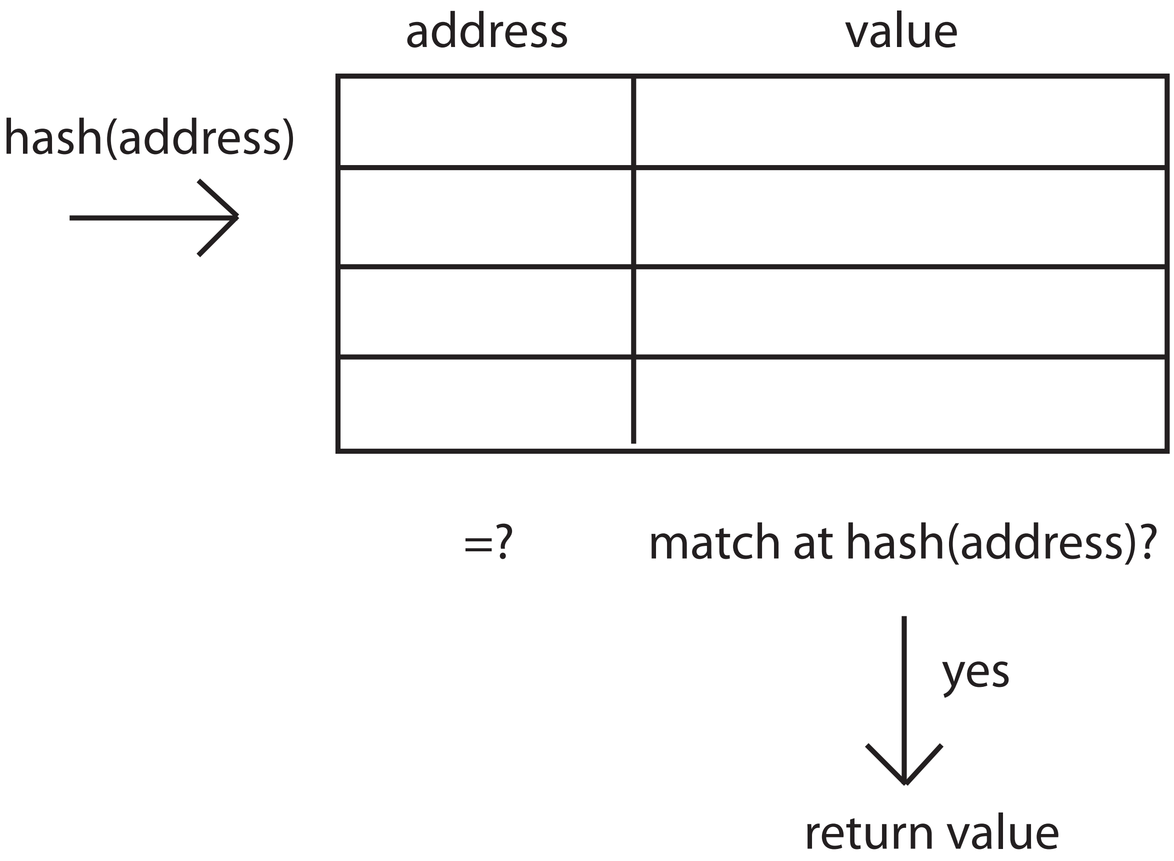
=?

match at any address?

yes

return value

Cache Lookup: Direct Mapped



address

value

hash(address)

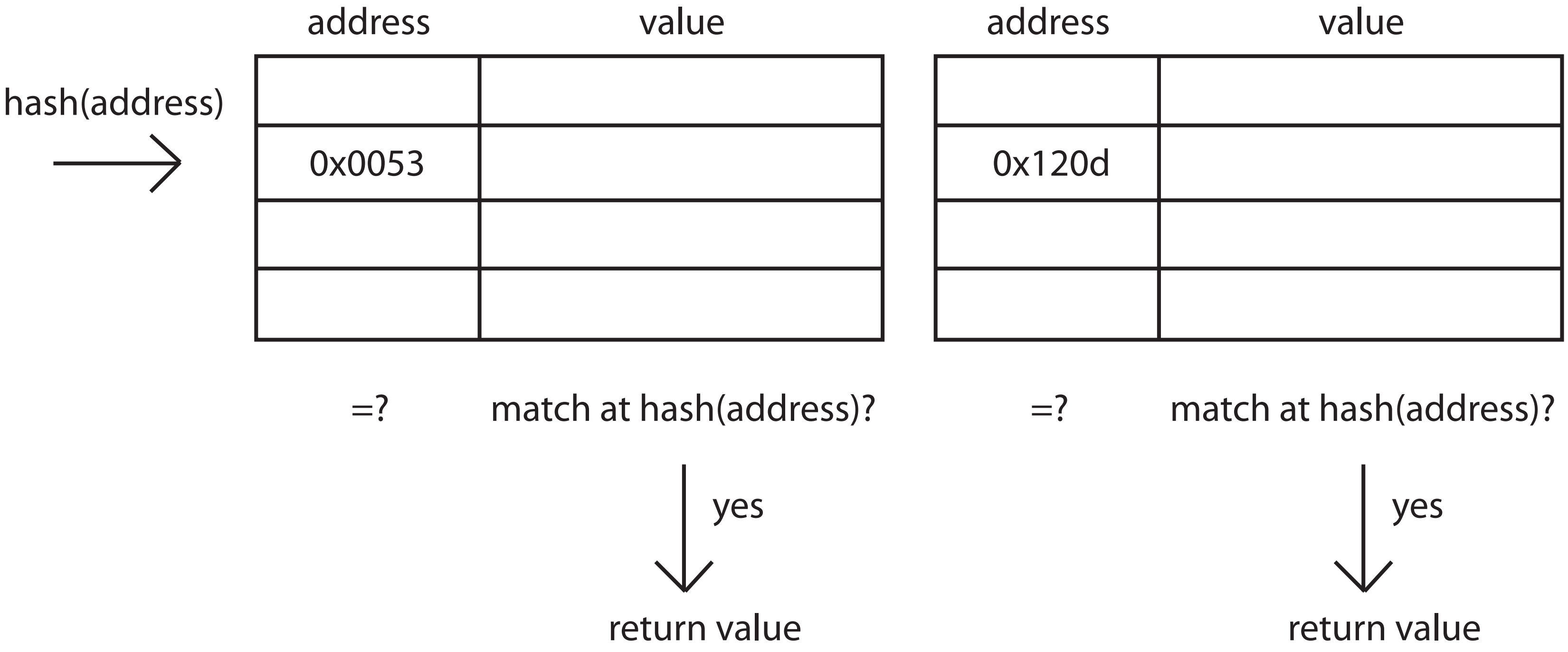
=?

match at hash(address)?

yes

return value

Cache Lookup: Set Associative



address

value

address

value

hash(address)

0x0053

0x120d

=?

match at hash(address)?

=?

match at hash(address)?

yes

return value

yes

return value

Page Coloring

- What happens when cache size >> page size?
  - Direct mapped or set associative
  - Multiple pages map to the same cache line
- OS page assignment matters!
  - Example: 8MB cache, 4KB pages
  - 1 of every 2K pages lands in same place in cache
- What should the OS do?

Cache Replacement Policy

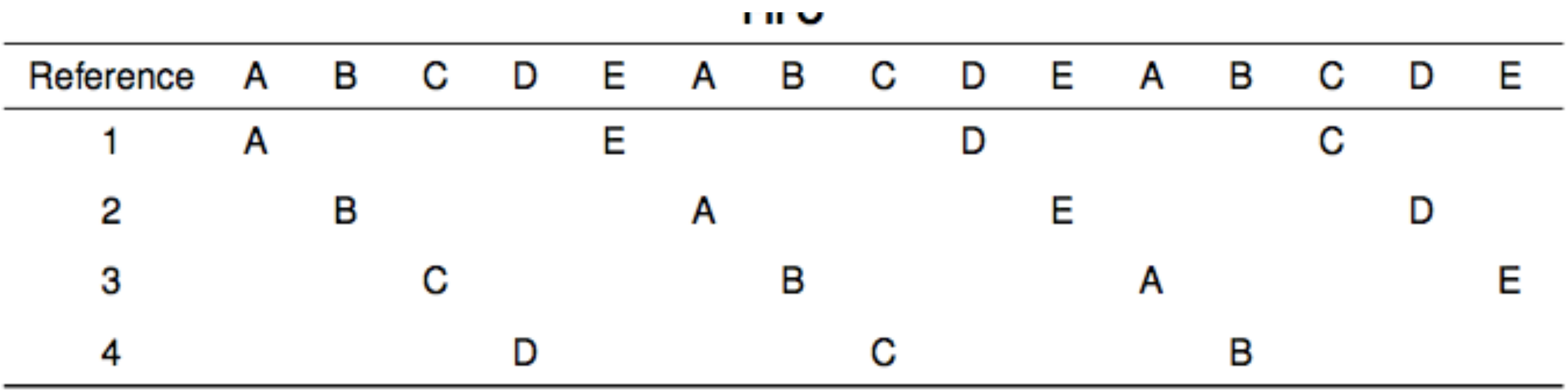
- On a cache miss, how do we choose which entry to replace?
  - Assuming the new entry is more likely to be used in the near future
  - In direct mapped caches, not an issue!

- Policy goal: reduce cache misses
  - Improve expected case performance
  - Also: reduce likelihood of very poor performance

A Simple Policy

- Random?
  - Replace a random entry
- FIFO?
  - Replace the entry that has been in the cache the longest time
  - What could go wrong?

FIFO in Action



FIFO

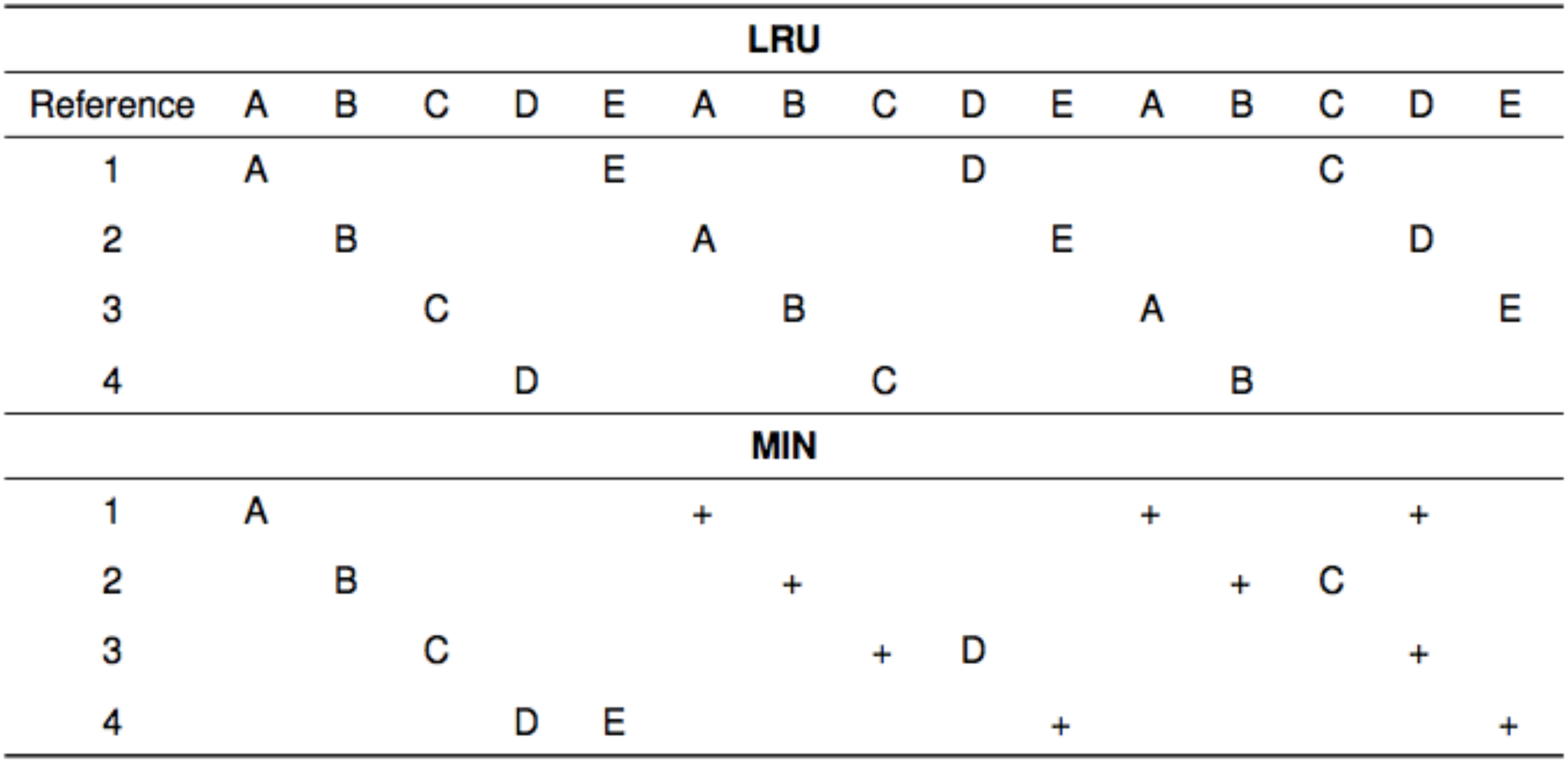
Reference	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
1	A				E				D				C		
2		B				A				E				D	
3			C				B				A				E
4				D				C				B			

Worst case for FIFO is if program strides through memory that is larger than the cache

MIN, LRU, LFU

- MIN
  - Replace the cache entry that will not be used for the longest time into the future
  - Optimality proof based on exchange: if evict an entry used sooner, that will trigger an earlier cache miss
- Least Recently Used (LRU)
  - Replace the cache entry that has not been used for the longest time in the past
  - Approximation of MIN
- Least Frequently Used (LFU)
  - Replace the cache entry used the least often (in the recent past)

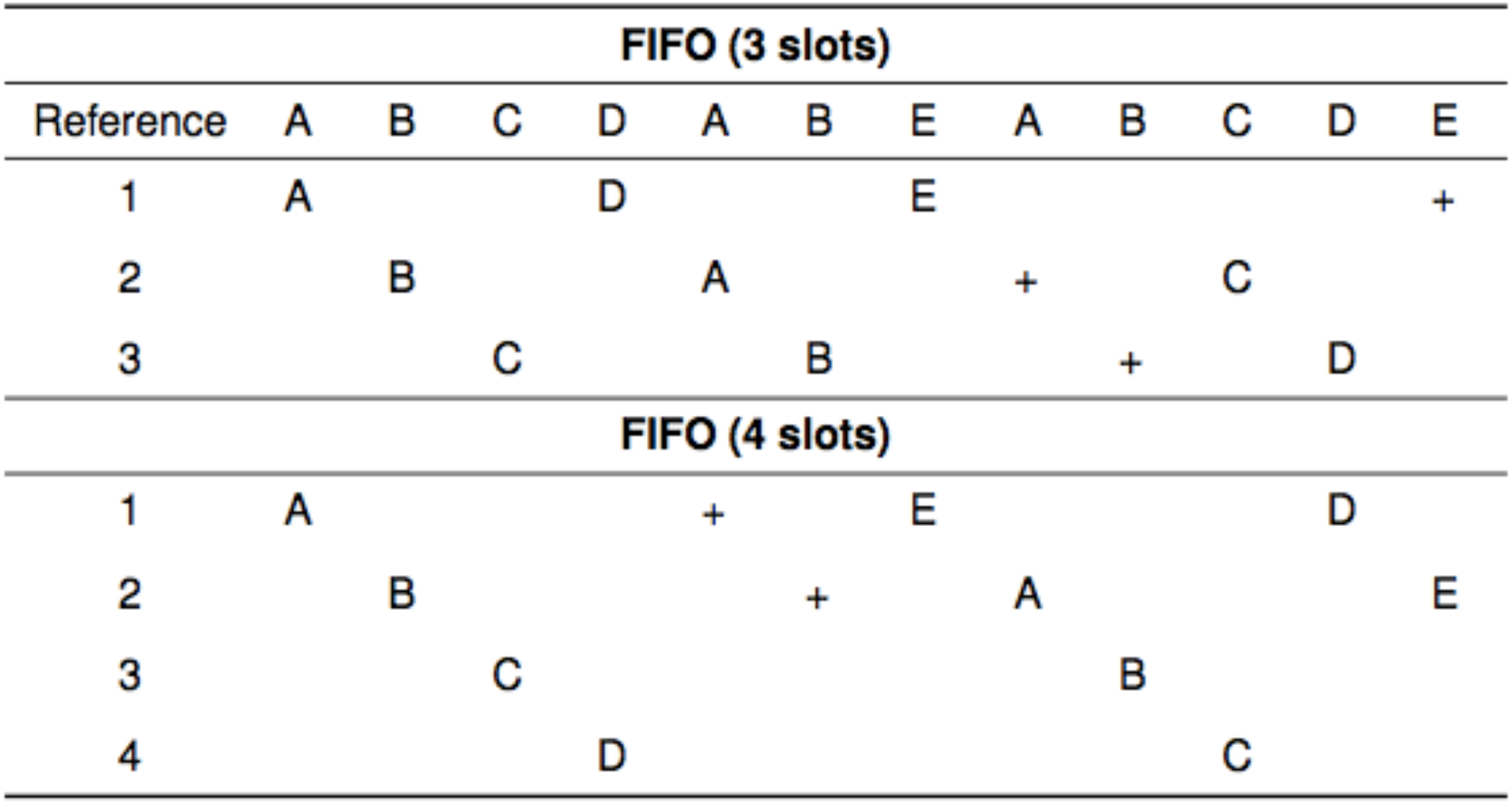
LRU/MIN for Sequential Scan



							LRU								
Reference	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
1	A				E				D				C		
2		B				A				E				D	
3			C				B				A				E
4				D				C				B			
							MIN								
1	A					+					+			+	
2		B					+					+	C		
3			C					+	D					+	
4				D	E					+					+

							LRU								
Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+				+			+	
2		B			+								+		
3				C					E			+			
4						D		+		+					C
							FIFO								
1	A		+				+		E						
2		B			+						A			+	
3				C								+	B		
4						D		+		+					C
							MIN								
1	A		+				+				+			+	
2		B			+								+		C
3				C					E			+			
4						D		+		+					

Belady’s Anomaly



FIFO (3 slots)												
Reference	A	B	C	D	A	B	E	A	B	C	D	E
1	A			D			E					+
2		B			A			+		C		
3			C			B			+		D	
FIFO (4 slots)												
1	A				+		E				D	
2		B				+		A				E
3			C						B			
4				D						C		