Virtual Memory

P $\xrightarrow{\text{fork}}$ C

ระหว่างที่ share

wocopy
Set Readonly

segment	B	Bo	Access			segment	B	Bo	Access
Code	B_0	A_0	R	R		Code	B_0	A_0	
data	B_1	A_1	R/W	R		data	B_1	A_1	
heap	B_2	A_2	R/W	R		heap	B_2	A_2	
stack	B_3	A_3	R/W	R		stack	B_3	A_3	
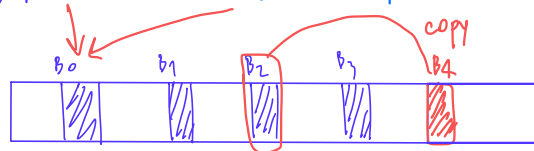
B_0 b_1 B_2 B_3 B_4 copy

COPY ON WRITE
① ช่วยประหยัด RAM
② พร้อมทำงานเร็วขึ้น (ready เร็ว)
● ถ้าไม่มีการ write(เพิ่ม) ก็ยังคง share ต่อไป

Fill on Reference

ถ้าไม่มีการเรียกใช้ function ก็ไม่มีการโหลดเพิ่ม   memory   mapped file   แต่ละ block เข้ากับ frame ของตัว RAM

zero copy I/O
เอาโปรแกรม(page, frame)
ที่ไม่ได้ใช้ Copy เก็บ
(background)

Logical

P_0	main() {
P_1	fn(); }
P_2	fn1(); }
P_3	fn2(); }
P_4	fnn(); }

Physical

main()		}	Z

โหลดเฉพาะที่ใช้   ↑ load
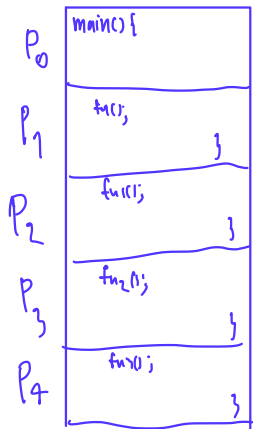
ผลที่ได้
โปรแกรมพร้อมทำงานเร็วขึ้น
ใช้ RAM น้อยลง / ประหยัดหน่วยความจำ

Storage (hardisk)

U
K
Buffer
file   mem mapp file

μP → Cache → RAM → memory Mapped
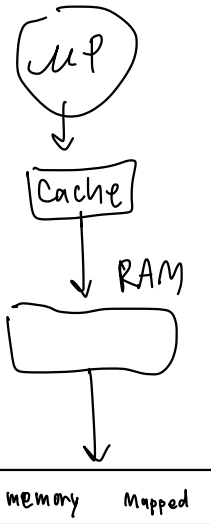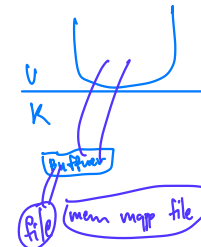
Definitions

- 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 แบ่งเป็นเป็นชุด block
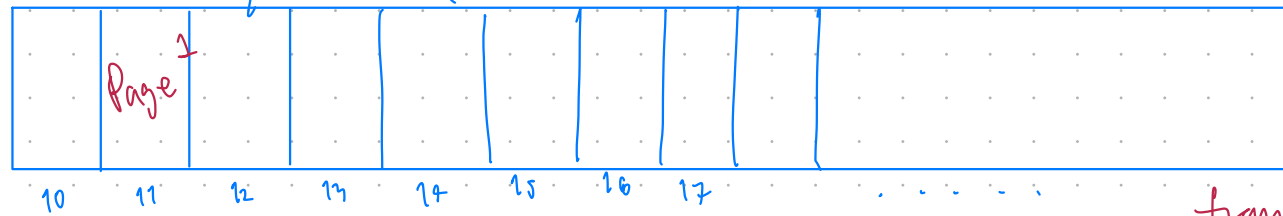 - Unit of cache storage (multiple memory locations)
- Temporal locality การใช้ข้อมูลซ้ำๆในลักษณะ loop
  - Programs tend to reference the same memory locations multiple times ลักษณะการทำงานปกติของ คารประ มวล ผลโปรแกรม ยกเว้นสำหร Call function
  - Example: instructions in a loop
- Spatial locality บริเวณ เดียวกัน
  - Programs tend to reference nearby locations
  - Example: data in a loop กับ Array
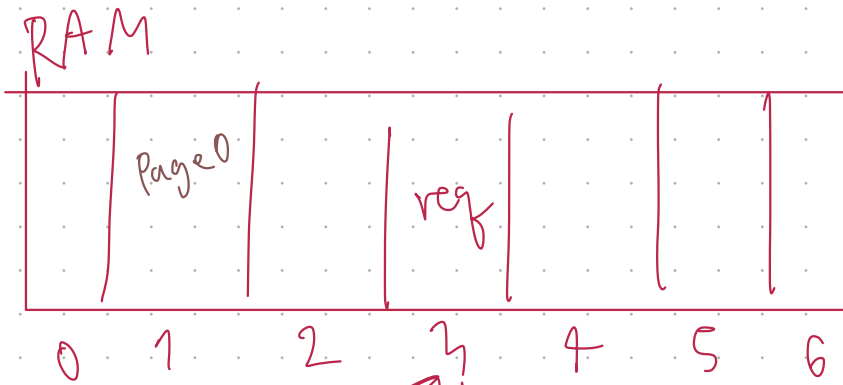
RAM

| | | Page 0 | | | reg | | | | | |

0   1   2   3   4   5   6

Memory   mapped   file   Storage

| | Page 1 | | | | | | | | | |

10   11   12   13   14   15   16   17   . . . . .

frame	access	IISN
Row/Entry 1	R	
n	invalid ไม่อยู่ใน RAM อยู่ใน

frame	access
1	R
Row/Entry 3	R/W

Cache Concept (Read)

Cache

Fetch Address

Address In Cache? (Miss)

No → Fetch Address

Hit
Yes

Store Value in Cache

processor request with cache
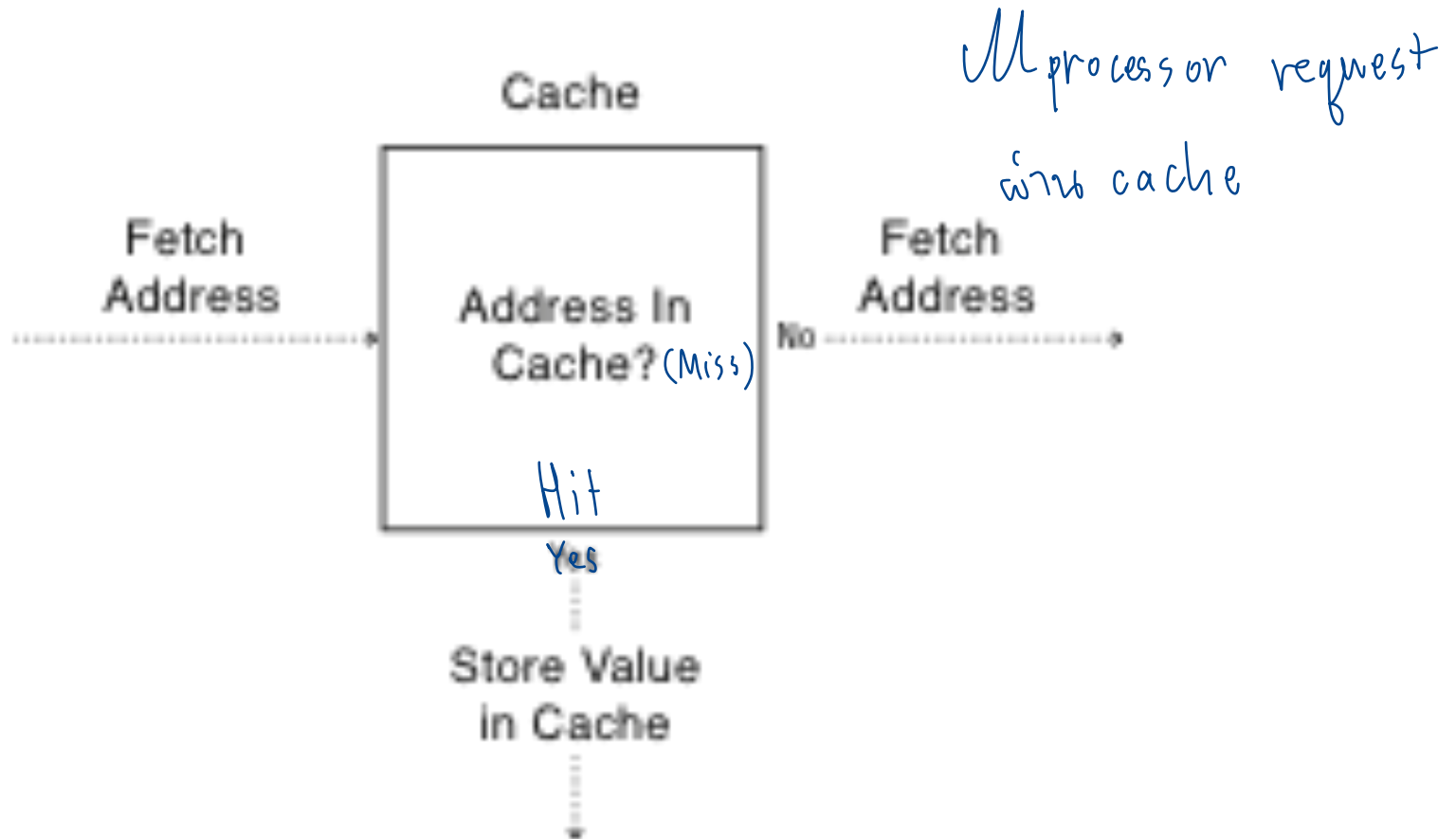
Cache Concept (Write)

write through

Cache

Store Value at Address → WriteBuffer → Store Value at Address → Address In Cache? → No → Fetch Address

Yes

Store Value in Cache

If Write Through → Store Value at Address
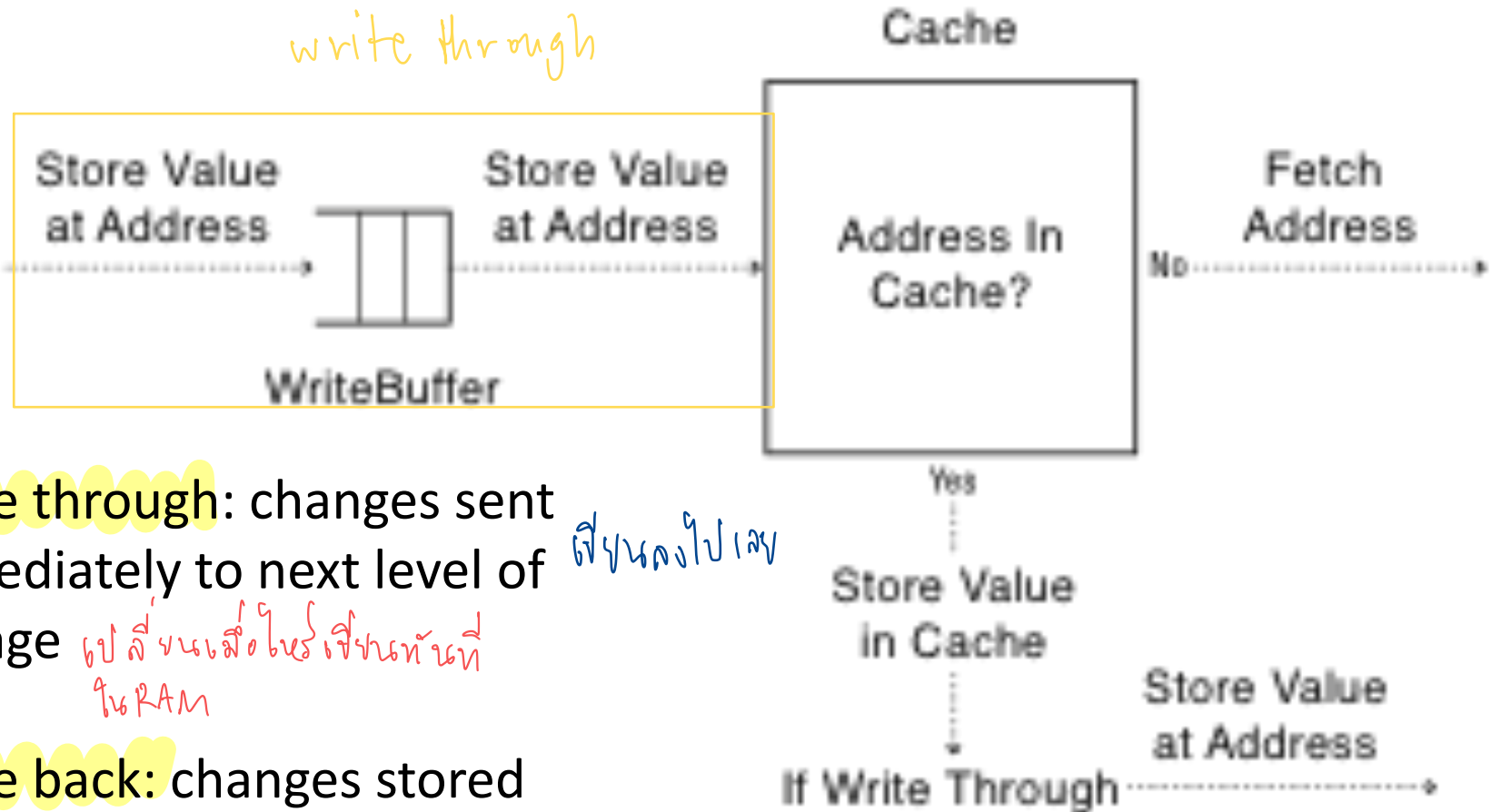
Write through: changes sent immediately to next level of storage เปลี่ยนเมื่อไหร่เขียนทันที ในRAM เขียนลงไปเลย

Write back: changes stored in cache until cache block is replaced เขียนใน cache พอเต็มค่อย เอาออกไปเก็บใน RAM (update ใน RAM) ทำ I/O กับ RAM ให้น้อยที่สุด

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

Cache hit rate
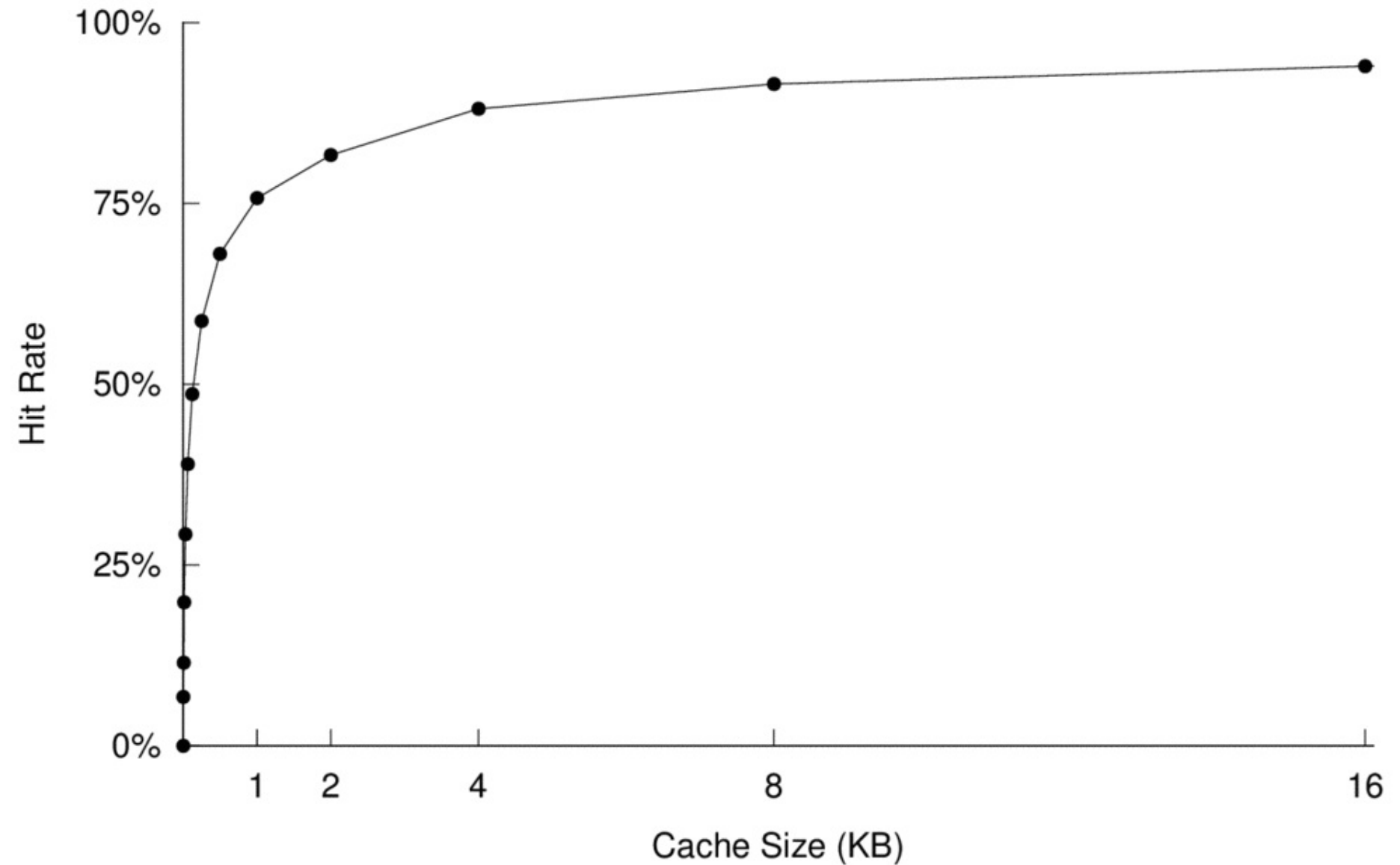
Example of cache hit rate over time
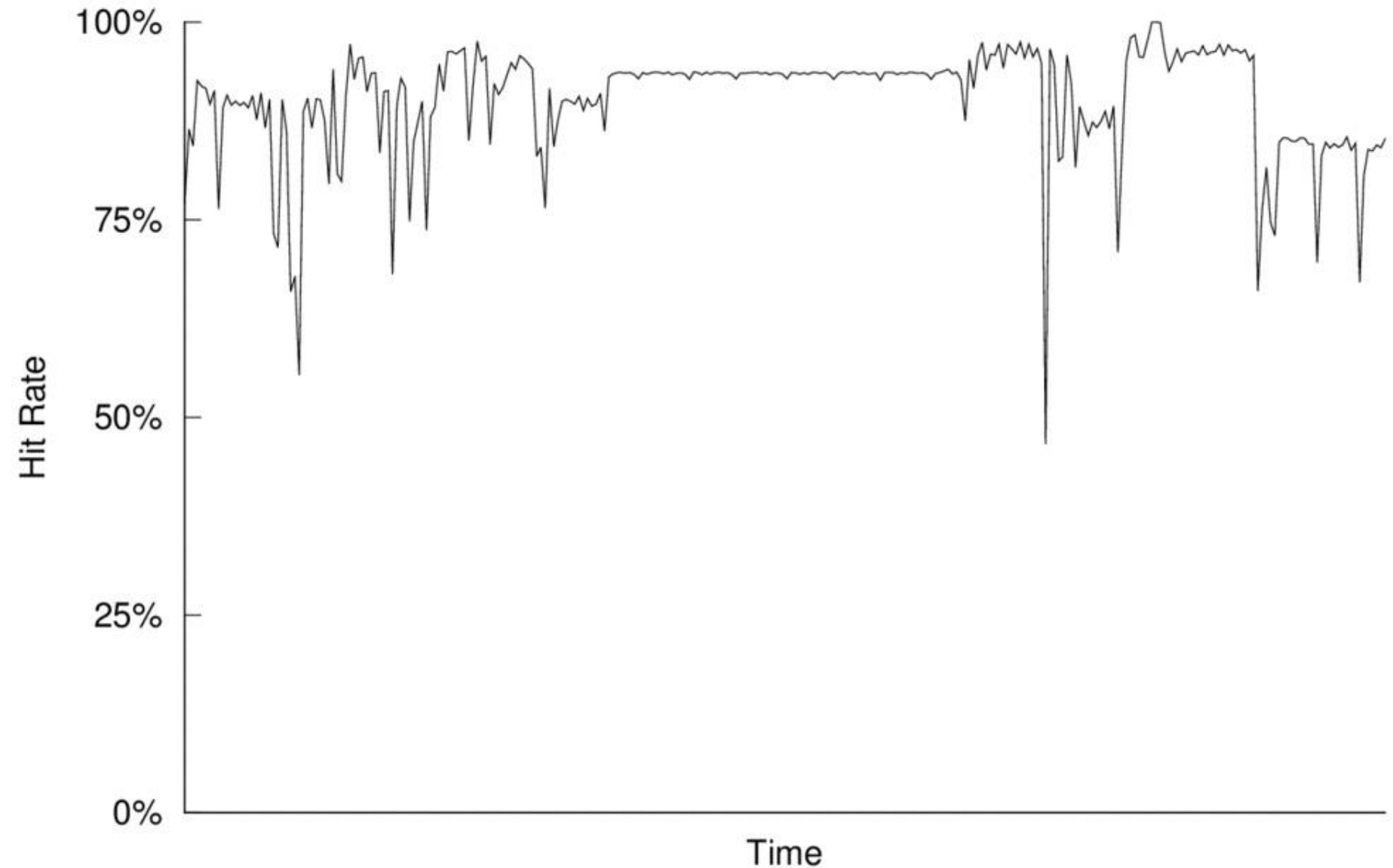
Main Points

- Can we provide the illusion of near infinite memory in limited physical memory?
  - Demand-paged virtual memory
  - Memory-mapped files
- How do we choose which page to replace?
  - FIFO, MIN, LRU, LFU, Clock
- What types of workloads does caching work for, and how well?
  - Spatial/temporal locality vs. Zipf workloads

Hardware address translation is a power tool

- Kernel trap on read/write to selected addresses

  ตัวที่ถูก write จะ copy ออกไปใน free space
  - Copy on write
  - Fill on reference

  } translation ทำให้เกิด

  ใน RAM   LOAD ช่องข้อมูลที่ Program ทำการเรียกใช้

  - Demand paged virtual memory

  - Memory mapped files

  - Modified bit emulation

  - Use bit emulation

Demand Paging (Before)
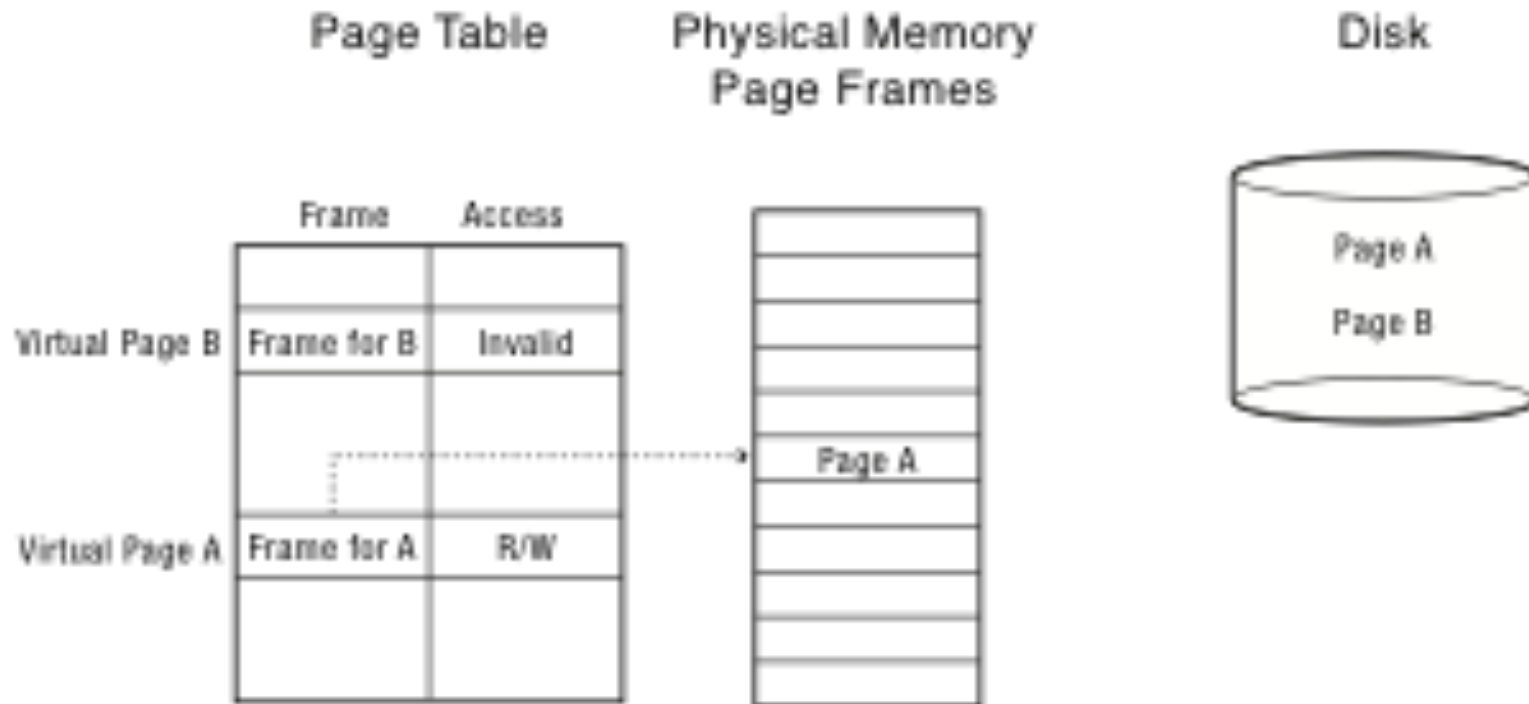
Demand Paging (After)
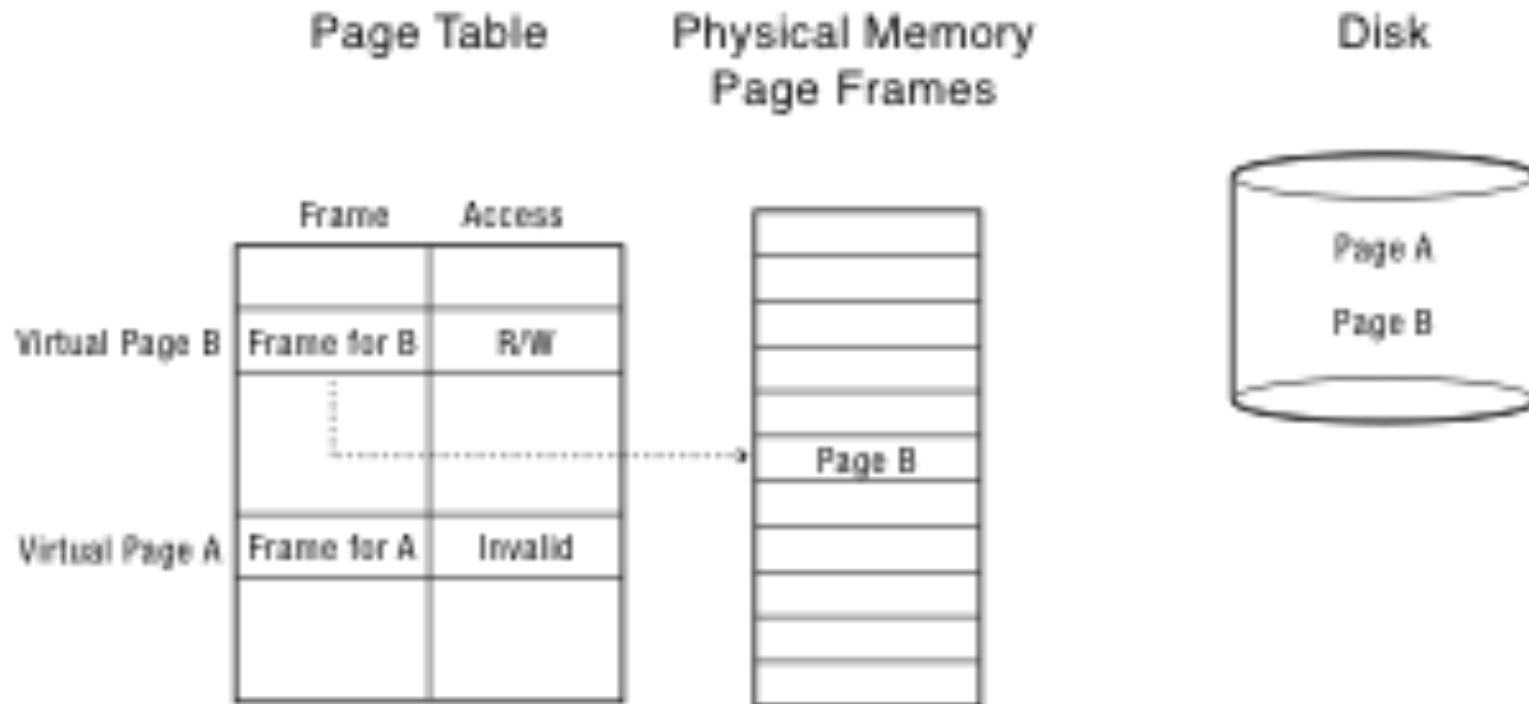
Demand Paging

1. TLB miss
2. Page table walk
3. Page fault (page invalid in page table)
4. Trap to kernel
5. Convert virtual address to file + offset
6. Allocate page frame
   - Evict page if needed
7. Initiate disk block read into page frame
8. Disk interrupt when DMA complete
9. Mark page as valid
10. Resume process at faulting instruction
11. TLB miss
12. Page table walk to fetch translation
13. Execute instruction

Allocating a Page Frame

- Select old page to evict
- Find all page table entries that refer to old page
  - If page frame is shared
- Set each page table entry to invalid
- Remove any TLB entries
  - Copies of now invalid page table entry
- Write changes on page back to disk, if necessary

How do we know if page has been modified?

- Every page table entry has some bookkeeping
  - Has page been modified?
    - Set by hardware on store instruction
    - In both TLB and page table entry
  - Has page been recently used?
    - Set by hardware on in page table entry on every TLB miss
- Bookkeeping bits can be reset by the OS kernel
  - When changes to page are flushed to disk
  - To track whether page is recently used

Keeping Track of Page Modifications (Before)

TLB		
Frame	Access	Dirty
	R/W	No

	Frame	Access	Dirty
Virtual Page B	Frame for B	Invalid	
Virtual Page A	Frame for A	R/W	No

Page Table

Physical Memory Page Frames

Page A

Disk

Old Page A

Old Page B
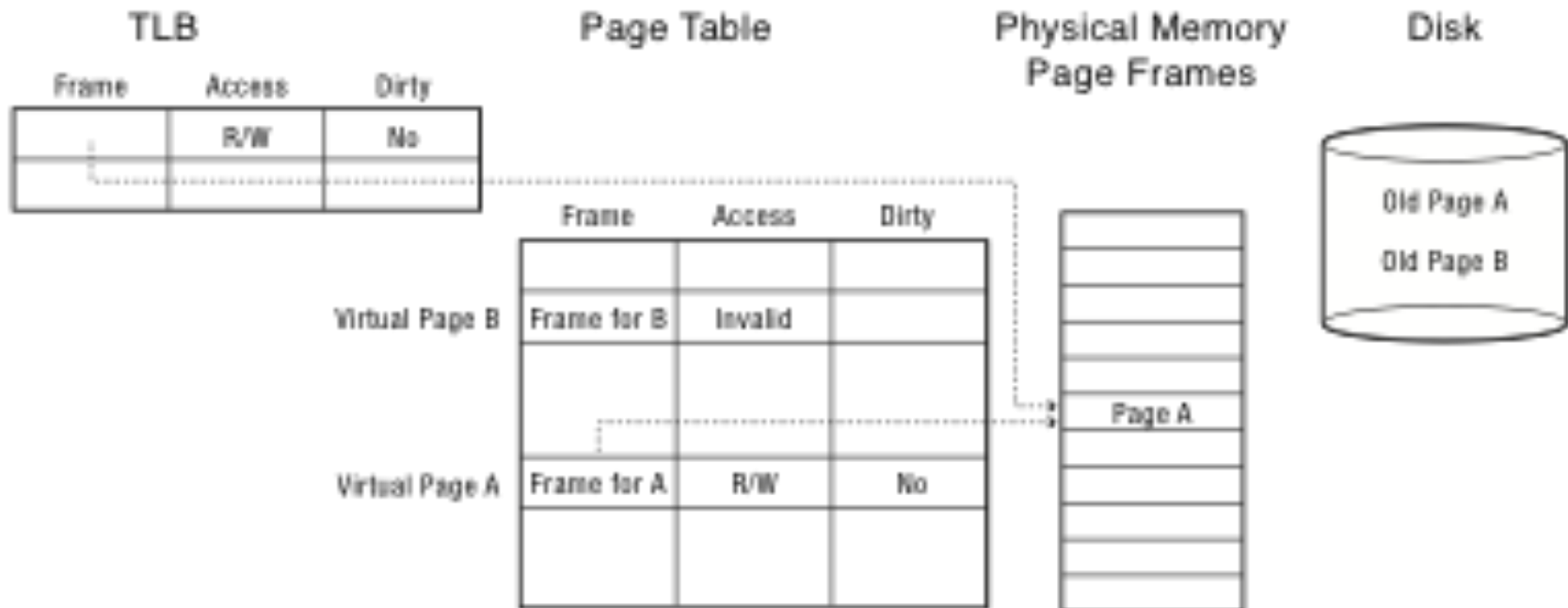
Keeping Track of Page Modifications (After)
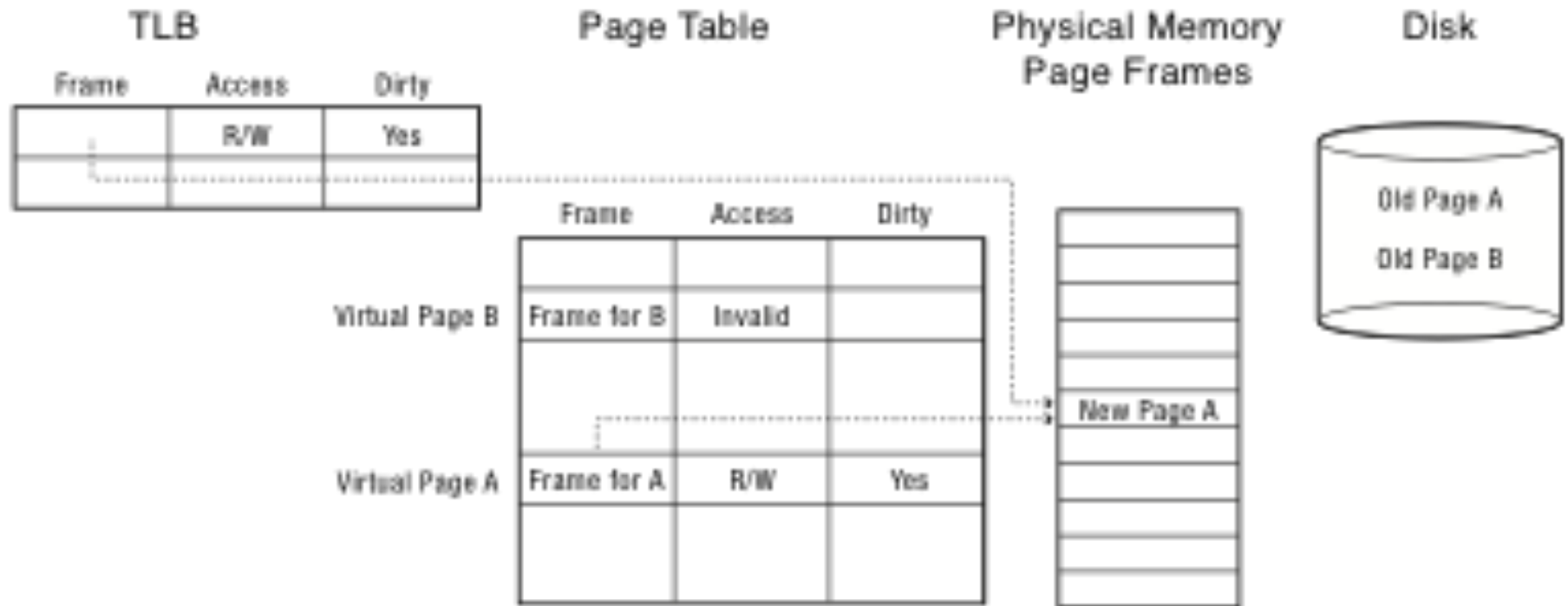
Virtual or Physical Dirty/Use Bits

- Most machines keep dirty/use bits in the page table entry
- Physical page is
  - Modified if *any* page table entry that points to it is modified
  - Recently used if *any* page table entry that points to it is recently used
- On MIPS, simpler to keep dirty/use bits in the core map
  - Core map: map of physical page frames

Models for Application File I/O

- Explicit read/write system calls
  - Data copied to user process using system call
  - Application operates on data
  - Data copied back to kernel using system call
- Memory-mapped files
  - Open file as a memory segment
  - Program uses load/store instructions on segment memory, implicitly operating on the file
  - Page fault if portion of file is not yet in memory
  - Kernel brings missing blocks into memory, restarts process

Advantages to Memory-mapped Files

- Programming simplicity, esp for large files
  - Operate directly on file, instead of copy in/copy out
- Zero-copy I/O
  - Data brought from disk directly into page frame
- Pipelining
  - Process can start working before all the pages are populated
- Interprocess communication
  - Shared memory segment vs. temporary file

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

	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
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

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

MIN

Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
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)

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

Recap

- MIN is optimal
  - replace the page or cache entry that will be used farthest into the future

- LRU is an approximation of MIN
  - For programs that exhibit spatial and temporal locality