Discussion 6

Paging, Caches

10/23/24

Staff

Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					Project 2 Design Due	
Homework 3 Due						
	Midterm 2 (7:00 pm - 9:00 pm)					

Paging

Page Tables

Allocate memory in fixed-size chunks called **pages**.

Page table is indexed by **virtual page number (VPN)**.

- Each entry is called the **page table entry (PTE)** which contains the **physical page number (PPN)** and metadata.
- Page tables also reside in memory.
- Typically (but not always) a page table will fit into a single page.
- Each process has its own page table. **Page table base register** (PTBR) contains the starting address of the page table for that process (e.g. CR3 register for x86).
- Metadata stores bits such as valid, dirty, writable, and more.

Virtual Address	
Virtual Page Number (VPN)	Offset

Page Table	
PPN	Metadata

Physical Address	
Physical Page Number (PPN)	Offset

Page Tables

Assumes **byte addressable** system where each bit addresses one byte.

\# virtual address bits = # virtual page number bits + # offset bits

\# physical address bits = # physical page number bits + # offset bits

\# virtual pages = $2^{\text{\# virtual page number bits}}$

\# physical pages = $2^{\text{\# physical page number bits}}$

page size (in bytes) = $2^{\text{\# offset bits}}$

virtual memory size (in bytes) = # virtual pages $\times$ page size (in bytes)

physical memory size (in bytes) = # physical pages $\times$ page size (in bytes)

\# page table entries = # virtual pages

\# page table entry bits = # physical page number bits + # metadata bits

page table entry size (in bytes) = # page table entry bits $\div$ 8

page table size (in bytes) = # page table entries $\times$ page table entry size (in bytes)

Virtual Address	
Virtual Page Number (VPN)	Offset

Page Table	
PPN	Metadata

Physical Address	
Physical Page Number (PPN)	Offset

Translation Lookaside Buffer (TLB)

Translation lookaside buffer (TLB) is a small *hardware* table containing results of recent address translations.
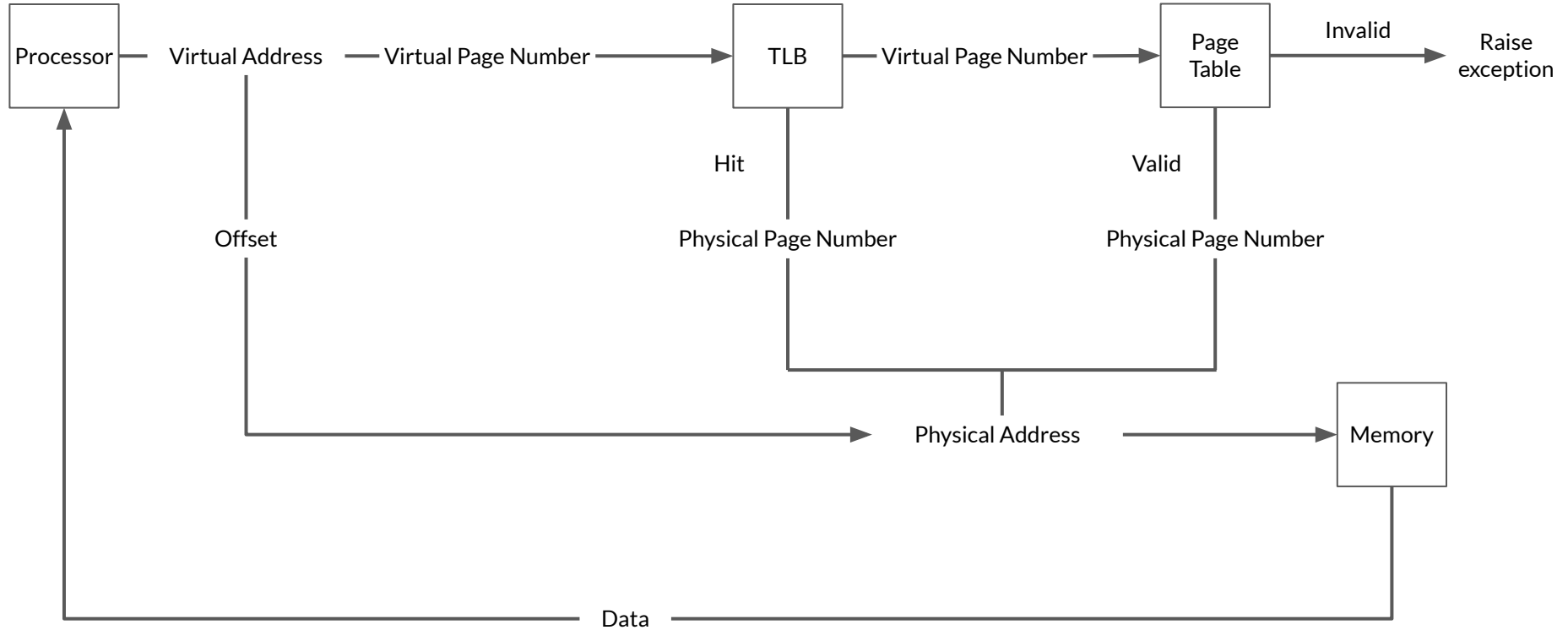
- Can be thought of as a cache, generally fully associative.
- Virtual page number is the tag.
- Stores address translations *not the actual data* (i.e. cache for page table).
- Generally part of the MMU.

Virtual Address	
Virtual Page Number (VPN)	Offset

Translation Lookaside Buffer

Tag	Data	
VPN	PPN	Metadata

Physical Address	
Physical Page Number (PPN)	Offset

Translation Lookaside Buffer (TLB)

Processor → Virtual Address → Virtual Page Number → TLB → Virtual Page Number → Page Table → Invalid → Raise exception

TLB → Hit → Physical Page Number

Page Table → Valid → Physical Page Number

Virtual Address → Offset

Physical Page Number + Physical Page Number → Physical Address → Memory

Memory → Data → Processor

Demand Paging

Working set is the subset of the address space used during execution.

Demand paging is the idea of storing certain working sets on disks and bringing them into memory as necessary.

- Allows for bigger virtual address space than physical memory can provide.
- **Resident set** is the subset of the address space being held in memory.
- **Thrashing** happens when memory is too small for working set and replacements need to frequently happen.

Page fault occurs when a program attempts to access virtual address not currently located in physical memory.

1. MMU traps to the kernel and saves current state information.
2. Kernel finds out which virtual page was needed (e.g. CR2 in x86).
3. Check if virtual address is valid. Use page replacement policy to remove a page from physical memory and write it to disk.
4. Bring in the page corresponding to the virtual address from disk (e.g. swap map in Linux) and update the page table entry.
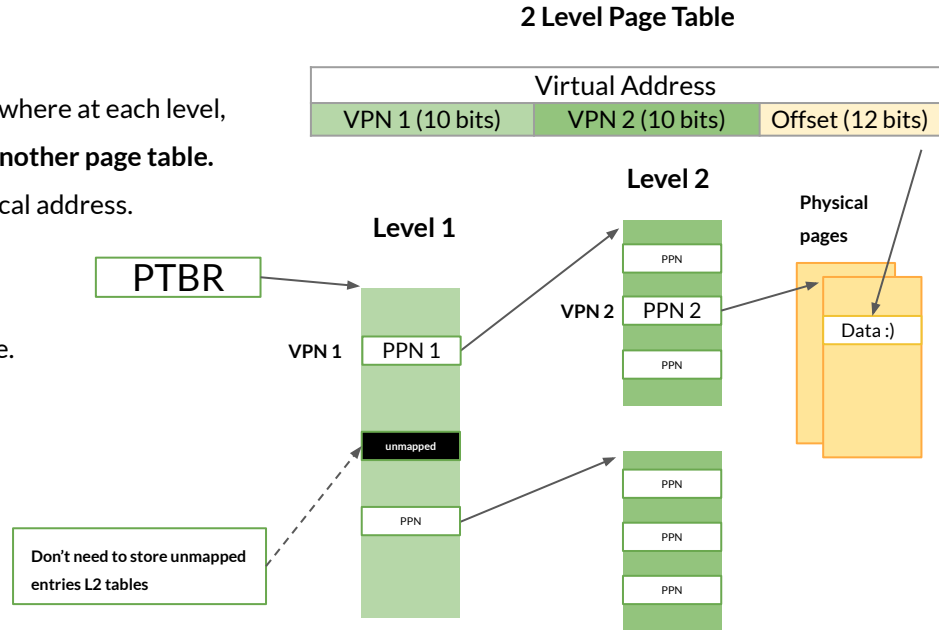5. Restore saved state from (1) and resume execution.
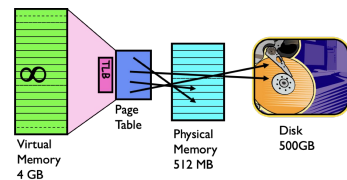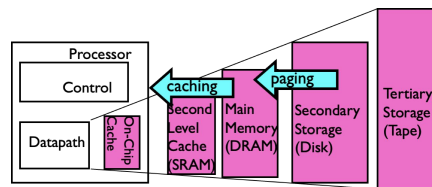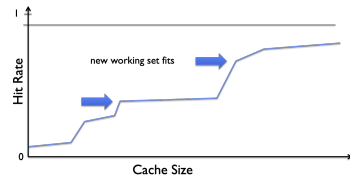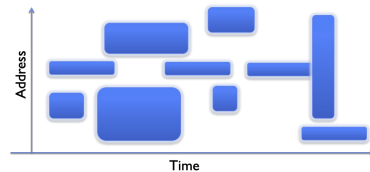
Multi level page tables

What is the issue with using 1 page table per process?

- If our address space is really large, we end up with a very large page table.
  - We may not use all pages all the time → can we instead store a working set of page table in memory?
- If our address space is really **sparse**, we end up with not just a large page table, but also many unused entries.
- Does the entire page table need to be in memory?

Solution: Multi-level paging

1. In a multi-level page table system, we use a **tree** of page tables, where at each level, excluding the bottom, the corresponding page table maps to **another page table.**
2. The **bottom level page table** provides a translation to the physical address.
3. We don't need to use every intermediary level page table
   a. If not needed, they can reside on disk.
4. **PTEs** now point either to next-level page table, or to actual page.
5. Page tables need to be contiguous.
   a. 32-bit address space:
   b. 10b-10b-12b config, page tables **exactly 1 page** in size.
6. Page table lookup *per level* → expensive!
   a. But… we can cache translations with the TLB!

2 Level Page Table

Virtual Address		
VPN 1 (10 bits)	VPN 2 (10 bits)	Offset (12 bits)

PTBR

Level 1

Level 2

Physical pages

VPN 1 — PPN 1

unmapped

PPN

VPN 2 — PPN 2

PPN

PPN

Data :)

PPN

PPN

PPN

Don't need to store unmapped entries L2 tables

Concept Check

1. True or False: The page table base pointer contains the virtual address of the page table.

2. True or False: If a user process accesses an address that generates a page fault, the OS will terminate this process for access violation.

3. What are some advantages of having a larger page size? What about its disadvantages?

Concept Check

1. True or False: The page table base pointer contains the virtual address of the page table.

    False. The CR3 register contains its physical address. If it were virtually addressed, it would have to go through address translation, but then you run into the problem of having to know where the page table is located in physical memory.

2. True or False: If a user process accesses an address that generates a page fault, the OS will terminate this process for access violation.

3. What are some advantages of having a larger page size? What about its disadvantages?

Concept Check

1. True or False: The page table base pointer contains the virtual address of the page table.

   False. The CR3 register contains its physical address. If it were virtually addressed, it would have to go through address translation, but then you run into the problem of having to know where the page table is located in physical memory.

2. True or False: If a user process accesses an address that generates a page fault, the OS will terminate this process for access violation.

   False. A page fault can mean many things: this memory can be just unmapped or previously mapped but not resident in main memory at the time of access. The OS can handle such cases appropriately and map a new page in the user address space.

3. What are some advantages of having a larger page size? What about its disadvantages?

Concept Check

1. True or False: The page table base pointer contains the virtual address of the page table.
   False. The CR3 register contains its physical address. If it were virtually addressed, it would have to go through address translation, but then you run into the problem of having to know where the page table is located in physical memory.

2. True or False: If a user process accesses an address that generates a page fault, the OS will terminate this process for access violation.
   False. A page fault can mean many things: this memory can be just unmapped or previously mapped but not resident in main memory at the time of access. The OS can handle such cases appropriately and map a new page in the user address space.

3. What are some advantages of having a larger page size? What about its disadvantages?
   Having a larger page size would decrease the number of page table entries and thus the size of the page table. It could also potentially improve TLB performance as a result. Its main disadvantage is internal fragmentation.

Little Page Translation

The following table shows the 4 entries in the page table. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

	Valid	PPN
0	0	7
1	1	9
2	0	3
3	1	2

If there are 1024 bytes per page, what is the physical address corresponding to the virtual address 0xF74?

Little Page Translation

The following table shows the 4 entries in the page table. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

	Valid	PPN
0	0	7
1	1	9
2	0	3
3	1	2

If there are 1024 bytes per page, what is the physical address corresponding to the virtual address 0xF74?

 # offset bits = log$_2$(page size)

= log$_2$(1024)

= 10

Little Page Translation

The following table shows the 4 entries in the page table. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

	Valid	PPN
0	0	7
1	1	9
2	0	3
3	1	2

If there are 1024 bytes per page, what is the physical address corresponding to the virtual address 0xF74?

offset bits = $\log_2$(page size)

          = $\log_2$(1024)

          = 10

Virtual Address											
0x	F				7				4		
0b	1	1	1	1	0	1	1	1	0	1	0	0
	VPN		Offset								

Little Page Translation

The following table shows the 4 entries in the page table. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

	Valid	PPN
0	0	7
1	1	9
2	0	3
3	1	2

If there are 1024 bytes per page, what is the physical address corresponding to the virtual address 0xF74?

offset bits = $\log_2$(page size)

= $\log_2$(1024)

= 10

Virtual Address											
0x	F				7				4		
0b	1	1	1	1	0	1	1	1	0	1	0	0
	VPN				Offset						

Physical Address											
0x											
0b	1	0									
	PPN			Offset							

Little Page Translation

The following table shows the 4 entries in the page table. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

	Valid	PPN
0	0	7
1	1	9
2	0	3
3	1	2

If there are 1024 bytes per page, what is the physical address corresponding to the virtual address 0xF74?

offset bits = $\log_2$(page size)

$\quad\quad$ = $\log_2$(1024)

$\quad\quad$ = 10

Virtual Address											
0x	F				7				4		
0b	1	1	1	1	0	1	1	1	0	1	0	0
	VPN				Offset						

Physical Address											
0x	B				7				4		
0b	1	0	1	1	0	1	1	1	0	1	0	0
	PPN				Offset						

Demand Pages

An up-and-coming big data startup has just hired you to help design their new memory system for a byte- addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. Suppose you know that there will only be 4 processes running at the same time, each with a Resident Set Size (RSS) of 512MB and a working set size of 256KB. What is the minimum amount of TLB entries that your system would need to support to be able to map/cache the working set size for one process? What happens if you have more entries? What about if you have fewer entries?

2. Suppose you run some benchmarks on the system and you see that the system is utilizing over 99% of its paging disk IO capacity, but only 10% of its CPU. What is a combination of the of disk space and memory size that can cause this to occur? Assume you have TLB entries equal to the answer from the previous part.

3. Out of increasing the size of the TLB, adding more disk space, and adding more memory, which one would lead to the largest performance increase and why?

Demand Pages

An up-and-coming big data startup has just hired you to help design their new memory system for a byte-addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. Suppose you know that there will only be 4 processes running at the same time, each with a Resident Set Size (RSS) of 512MB and a working set size of 256KB. What is the minimum amount of TLB entries that your system would need to support to be able to map/cache the working set size for one process? What happens if you have more entries? What about if you have fewer entries?

    Working set size 256 KB = 64 pages → TLB needs 64 entries.

    More entries → performance increases since process has changing working sets.

    Fewer entries → performance suffers since it can't easily translate addresses in the working set.

2. Suppose you run some benchmarks on the system and you see that the system is utilizing over 99% of its paging disk IO capacity, but only 10% of its CPU. What is a combination of the of disk space and memory size that can cause this to occur? Assume you have TLB entries equal to the answer from the previous part.

3. Out of increasing the size of the TLB, adding more disk space, and adding more memory, which one would lead to the largest performance increase and why?

Demand Pages

An up-and-coming big data startup has just hired you to help design their new memory system for a byte- addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. Suppose you know that there will only be 4 processes running at the same time, each with a Resident Set Size (RSS) of 512MB and a working set size of 256KB. What is the minimum amount of TLB entries that your system would need to support to be able to map/cache the working set size for one process? What happens if you have more entries? What about if you have fewer entries?

   Working set size 256 KB = 64 pages → TLB needs 64 entries.

   More entries → performance increases since process has changing working sets.

   Fewer entries → performance suffers since it can't easily translate addresses in the working set.

2. Suppose you run some benchmarks on the system and you see that the system is utilizing over 99% of its paging disk IO capacity, but only 10% of its CPU. What is a combination of the of disk space and memory size that can cause this to occur? Assume you have TLB entries equal to the answer from the previous part.

   System is thrashing → not enough memory to fit the working-set in memory without page faulting .

   Regardless of disk space or TLB size, demand paging will take place as long as physical memory size is < 4 × 512 MB = 2 GB.

3. Out of increasing the size of the TLB, adding more disk space, and adding more memory, which one would lead to the largest performance increase and why?

Demand Pages

An up-and-coming big data startup has just hired you to help design their new memory system for a byte- addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. Suppose you know that there will only be 4 processes running at the same time, each with a Resident Set Size (RSS) of 512MB and a working set size of 256KB. What is the minimum amount of TLB entries that your system would need to support to be able to map/cache the working set size for one process? What happens if you have more entries? What about if you have fewer entries?

   Working set size 256 KB = 64 pages → TLB needs 64 entries.

   More entries → performance increases since process has changing working sets.

   Less entries → performance suffers since it can't easily translate addresses in the working set.

2. Suppose you run some benchmarks on the system and you see that the system is utilizing over 99% of its paging disk IO capacity, but only 10% of its CPU. What is a combination of the of disk space and memory size that can cause this to occur? Assume you have TLB entries equal to the answer from the previous part.

   System is thrashing → not enough memory to fit the working-set in memory without page faulting .

   Regardless of disk space or TLB size, demand paging will take place as long as physical memory size is < 4 ⨉ 512 MB = 2 GB.

3. Out of increasing the size of the TLB, adding more disk space, and adding more memory, which one would lead to the largest performance increase and why?

   Add more memory to avoid demand paging.

Caches

Caches

Cache contains a copy of data that can be accessed more quickly than the original.

- Exploits different types of locality to quickly access data.
- **Temporal locality** is when programs tend to reference same data that was recently accessed.
- **Spatial locality** is when programs tend to reference data near other data that was recently accessed.

Average memory access time (AMAT) = Hit Rate $\times$ Hit Time + Miss Rate $\times$ Miss Time

Miss Time = (Cache Access Time + Miss Penalty)

Replacement Policies

First in First Out (FIFO) evicts page that has been in memory the longest.

- Somewhat fair since each page spends a roughly equal amount of time before eviction.
- Worst performance for cycling through an array bigger than cache.

Minimum (MIN) replaces pages used farthest in the future.

- Provably optimal for minimizing misses.
- Requires knowledge of the future.

Least recently used (LRU) evicts pages that hasn't been used for the longest time.

- Exploits temporal locality to predict the future.
- Costly implementation (e.g. linked lists), especially in hardware.
- Approximates MIN but not always optimal.

Belady's anomaly states that increasing cache capacity can actually *increase* the miss rate.

- Applies to algorithms that don't have the **stack property** which states that the set of pages with cache size n ⊆ set of pages with cache size n + 1.
- FIFO is not a stack algorithm, MIN and LRU are.

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

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

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

FIFO

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

LRU

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

MIN

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

FIFO

	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		

LRU

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

MIN

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

Translation Trivia

1. Consider a machine with a physical memory of 8 GiB, a page size of 8 KiB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page?

2. List the fields of a page table entry (PTE) in your scheme.

Translation Trivia

1. Consider a machine with a physical memory of 8 GiB, a page size of 8 KiB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page?

bytes that can be mapped using a page table = # pages pointed to by page table $\times$ page size

$$= \text{(size of page table} \div \text{size of PTE)} \times \text{page size}$$

$$= 2^{13} \div 2^2 \times 2^{13}$$

$$= 2^{24} \text{ bytes}$$

Byte addressable means each byte is addressed by one bit in the virtual address.

One level $\rightarrow 2^{24}$ bytes

Two levels $\rightarrow 2^{35}$ bytes

Three levels $\rightarrow 2^{46}$ bytes

2. List the fields of a page table entry (PTE) in your scheme.

Translation Trivia

1. Consider a machine with a physical memory of 8 GiB, a page size of 8 KiB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page?

bytes that can be mapped using a page table = # pages pointed to by page table $\times$ page size

$$= (\text{size of page table} \div \text{size of PTE}) \times \text{page size}$$

$$= 2^{13} \div 2^2 \times 2^{13}$$

$$= 2^{24} \text{ bytes}$$

Byte addressable means each byte is addressed by one bit in the virtual address.

One level $\rightarrow 2^{24}$ bytes

Two levels $\rightarrow 2^{35}$ bytes

Three levels $\rightarrow 2^{46}$ bytes

2. List the fields of a page table entry (PTE) in your scheme.

PPN bits = $\log_2$(# physical pages)

$$= \log_2(\text{physical memory size} \div \text{page size})$$

$$= \log_2(2^{33} \div 2^{13})$$

$$= 20 \text{ bits}$$

metadata bits = # PTE bits - # PPN bits = 12 bits

Translation Trivia

3.   Without a cache or TLB, how many memory operations are required to read or write a single 32-bit word?

4.   With a TLB, how many memory operations can this be reduced to? Best-case scenario? Worst-case scenario?

Translation Trivia

3. Without a cache or TLB, how many memory operations are required to read or write a single 32-bit word?

    3 page table lookups + 1 data access = 4

4. With a TLB, how many memory operations can this be reduced to? Best-case scenario? Worst-case scenario?

Translation Trivia

3. Without a cache or TLB, how many memory operations are required to read or write a single 32-bit word?

   3 page table lookups + 1 data access = 4

4. With a TLB, how many memory operations can this be reduced to? Best-case scenario? Worst-case scenario?

   TLB access does not count as memory operation.

   Best case: 1 data access

   Worst case: 3 page table lookups + 1 data access = 4

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

$$0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.$$

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

\# offset bits = $\log_2$(page size)

$\quad = \log_2(1024)$

$\quad = 10$

TLB hits	0
Page Faults	0

Page Table		
	Valid	PPN
0	0	NULL
1	1	2
2	0	NULL
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

TLB		
Tag	PPN	Valid
NULL	NULL	0
NULL	NULL	0
NULL	NULL	0
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

      = $\log_2$(1024)

      = 10

TLB hits	0
Page Faults	0

Page Table		
	Valid	PPN
0	0	NULL
1	1	2
2	0	NULL
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address											
0x		2			9				4			
0b	0	0	0	1	0	1	0	0	1	0	1	0	0
	VPN			Offset								

TLB		
Tag	PPN	Valid
NULL	NULL	0
NULL	NULL	0
NULL	NULL	0
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

<p style="text-align:center">**0x294**, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.</p>

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

     = $\log_2$(1024)

     = 10

TLB hits	0
Page Faults	1

Page Table		
	Valid	PPN
0	0	NULL
1	1	2
2	0	NULL
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address											
0x		2			9				4			
0b	0	0	0	1	0	1	0	0	1	0	1	0	0
	VPN						Offset					

TLB			
Tag	PPN	Valid	
NULL	NULL	0	
NULL	NULL	0	
NULL	NULL	0	
NULL	NULL	0	

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

<center>0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.</center>

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

TLB hits	0
Page Faults	1

offset bits = $\log_2$(page size)

    = $\log_2$(1024)

    = 10

	Page Table	
	Valid	PPN
0	1	1
1	1	2
2	0	NULL
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

0x		2			9			4		
0b	0	0	0	1	0	1	0	0	1	0	1	0	0
	VPN				Offset					

	TLB		
Tag	PPN	Valid	
0	1	1	
NULL	NULL	0	
NULL	NULL	0	
NULL	NULL	0	

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, **0xA76**, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

TLB hits	0
Page Faults	1

offset bits = $\log_2$(page size)

      = $\log_2$(1024)

      = 10

Page Table		
	Valid	PPN
0	1	1
1	1	2
2	0	NULL
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address		
0x	A	7	6
0b	0 1 0 1 0 0 1 1 1 0 1 1 0		
	VPN	Offset	

TLB		
Tag	PPN	Valid
0	1	1
NULL	NULL	0
NULL	NULL	0
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, **0xA76**, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

      = $\log_2$(1024)

      = 10

TLB hits	0
Page Faults	2

Page Table		
	Valid	PPN
0	1	1
1	1	2
2	0	NULL
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

		Virtual Address											
0x			A			7				6			
0b	0	1	0	1	0	0	1	1	1	0	1	1	0
	VPN				Offset								

TLB		
Tag	PPN	Valid
0	1	1
NULL	NULL	0
NULL	NULL	0
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

<div align="center">

0x294, **0xA76**, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

</div>

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

     = $\log_2$(1024)

     = 10

TLB hits	0
Page Faults	2

Page Table

	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

		A				7				6		
0x												
0b	0	1	0	1	0	0	1	1	1	0	1	1	0
	VPN			Offset									

TLB

Tag	PPN	Valid
0	1	1
2	3	1
NULL	NULL	0
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

    0x294, 0xA76, **0x5A4**, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

    How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

    = $\log_2$(1024)

    = 10

TLB hits	0
Page Faults	2

	Page Table	
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address												
0x		5			A				4				
0b	0	0	1	0	1	1	0	1	0	0	1	0	0
	VPN				Offset								

TLB		
Tag	PPN	Valid
0	1	1
2	3	1
NULL	NULL	0
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, **0x5A4**, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

TLB hits	0
Page Faults	2

\# offset bits = $\log_2$(page size)

$\qquad$ = $\log_2$(1024)

$\qquad$ = 10

	Page Table	
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

		Virtual Address											
0x			5			A				4			
0b	0	0	1	0	1	1	0	1	0	0	1	0	0
	VPN				Offset								

	TLB	
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, **0x923**, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

    = $\log_2$(1024)

    = 10

TLB hits	0
Page Faults	2

Page Table

	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

		Virtual Address											
0x		9			2			3					
0b	0	1	0	0	1	0	0	1	0	0	0	1	1
	VPN			Offset									

TLB

Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, **0x923**, 0xCFF, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

    = $\log_2$(1024)

    = 10

TLB hits	1
Page Faults	2

Page Table

	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

0x	9	2	3

0b	0	1	0	0	1	0	0	1	0	0	0	1	1
	VPN			Offset									

TLB

Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, **0xCFF**, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

    = $\log_2$(1024)

    = 10

TLB hits	1
Page Faults	2

Page Table

	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

		Virtual Address		
0x		C	F	F
0b	0	1 1 0 0	1 1 1 1 1 1 1 1
		VPN	Offset	

TLB

Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

    0x294, 0xA76, 0x5A4, 0x923, **0xCFF**, 0xA12, 0xF9F, 0x392, 0x341.

    How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

       = $\log_2$(1024)

       = 10

TLB hits	1
Page Faults	3

	Page Table	
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	0	4
4	0	5
5	1	6
6	1	7
7	0	NULL

		Virtual Address										
0x		C			F				F			
0b	0	1	1	0	0	1	1	1	1	1	1	1	1
	VPN			Offset								

TLB		
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
NULL	NULL	0

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, **0xCFF**, 0xA12, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

TLB hits	1
Page Faults	3

offset bits = $\log_2$(page size)

      = $\log_2$(1024)

      = 10

	Page Table	
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

		C			F			F		
0x										
0b	0	1	1	0	0	1	1	1	1	1	1	1	1
	VPN			Offset							

	TLB	
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, **0xA12**, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

        = $\log_2$(1024)

        = 10

TLB hits	1
Page Faults	3

Page Table	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

		Virtual Address		
0x		A	1	2
0b	0 1 0 1	0 0 0 0	1 0 0 1	0
	VPN		Offset	

TLB	Tag	PPN	Valid
	0	1	1
	2	3	1
	1	2	1
	3	4	1

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, **0xA12**, 0xF9F, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)
    = $\log_2$(1024)
    = 10

TLB hits	2
Page Faults	3

	Page Table	
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

0x		A			1			2		
0b	0	1	0	1	0	0	0	0	1	0	0	1	0

VPN — Offset

	TLB		
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, **0xF9F**, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

$\quad = \log_2(1024)$

$\quad = 10$

TLB hits	2
Page Faults	3

Page Table		
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

0x		F		9		F	
0b	0	1	1	1	1	1	0	0	1	1	1	1	1
	VPN			Offset			

TLB		
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, **0xF9F**, 0x392, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

   = $\log_2$(1024)

   = 10

TLB hits	3
Page Faults	3

Page Table		
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address		
0x	F	9	F
0b 0	1 1	1 1 1 0 0 1	1 1 1 1
	VPN	Offset	

TLB		
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

<div align="center">

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, **0x392**, 0x341.

</div>

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

      = $\log_2$(1024)

      = 10

TLB hits	3
Page Faults	3

Page Table

	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

		3		9		2	
0x							

0b	0	0	0	1	1	1	0	0	1	0	0	1	0
	VPN			Offset								

TLB

Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, **0x392**, 0x341.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

      = $\log_2$(1024)

      = 10

TLB hits	4
Page Faults	3

Page Table		
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address		
0x	3	9	2
0b	0 0 0	1 1 1 0 0 1	0 0 1 0
	VPN	Offset	

TLB		
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, **0x341**.

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

  = $\log_2$(1024)

  = 10

TLB hits	4
Page Faults	3

Page Table		
	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

	Virtual Address		
0x	3	4	1
0b	0 0 0 1 1	0 1 0 0 0	0 0 1
	VPN	Offset	

TLB		
Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

Translation Trivia

5. Consider a machine with a page size of 1024 bytes. There are 8KB of physical memory and 8KB of virtual memory. The TLB is a fully associative cache with space for 4 entries that is currently empty. Assume that the physical page number is always one more than the virtual page number. This is a sequence of memory address accesses for a program we are writing:

<div align="center">

0x294, 0xA76, 0x5A4, 0x923, 0xCFF, 0xA12, 0xF9F, 0x392, **0x341**.

</div>

How many TLB hits and page faults are there? What are the contents of the TLB at the end of the sequence?

offset bits = $\log_2$(page size)

       = $\log_2$(1024)

       = 10

TLB hits	5
Page Faults	3

Page Table

	Valid	PPN
0	1	1
1	1	2
2	1	3
3	1	4
4	0	5
5	1	6
6	1	7
7	0	NULL

Virtual Address

			3			4			1		
0x

0b: 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1

VPN | Offset

TLB

Tag	PPN	Valid
0	1	1
2	3	1
1	2	1
3	4	1

AMAT Calculations

Assume you are building a memory scheme with single level page tables. Each main memory access takes 50 ns and each TLB access takes 10 ns.

1. Assuming no page faults (i.e. all virtual memory is resident,) what TLB hit rate is required for an AMAT of 61 ns?

2. Assuming a TLB hit rate of 50%, how does the AMAT of this scenario compare to no TLB?

3. To improve your system, you add a two level paging scheme and a cache. The cache has a 90% hit rate with a lookup time of 20 ns. Additionally, the TLB hit rate is now improved to 95%. What is the average time to read a location from memory?

AMAT Calculations

Assume you are building a memory scheme with single level page tables. Each main memory access takes 50 ns and each TLB access takes 10 ns.

1. Assuming no page faults (i.e. all virtual memory is resident,) what TLB hit rate is required for an AMAT of 61 ns?

    (10+50) × hit rate + (1 - hit rate) × (10 + 50 + 50) = 61.

    Solving for hit rate gives 98%.

2. Assuming a TLB hit rate of 50%, how does the AMAT of this scenario compare to no TLB?

3. To improve your system, you add a two level paging scheme and a cache. The cache has a 90% hit rate with a lookup time of 20 ns. Additionally, the TLB hit rate is now improved to 95%. What is the average time to read a location from memory?

AMAT Calculations

Assume you are building a memory scheme with single level page tables. Each main memory access takes 50 ns and each TLB access takes 10 ns.

1. Assuming no page faults (i.e. all virtual memory is resident,) what TLB hit rate is required for an AMAT of 61 ns?

   $(10+50) \times$ hit rate $+ (1$ - hit rate$) \times (10 + 50 + 50) = 61$.

   Solving for hit rate gives 98%.

2. Assuming a TLB hit rate of 50%, how does the AMAT of this scenario compare to no TLB?

   With TLB: AMAT $= (10 + 50) \times 0.5 + (1 - 0.5) \times (10 + 50 + 50) = 85$ ns

   Without TLB: AMAT $= 50 + 50 = 100$ ns

3. To improve your system, you add a two level paging scheme and a cache. The cache has a 90% hit rate with a lookup time of 20 ns. Additionally, the TLB hit rate is now improved to 95%. What is the average time to read a location from memory?

AMAT Calculations

Assume you are building a memory scheme with single level page tables. Each main memory access takes 50 ns and each TLB access takes 10 ns.

1. Assuming no page faults (i.e. all virtual memory is resident,) what TLB hit rate is required for an AMAT of 61 ns?

   (10+50) $\times$ hit rate + (1 - hit rate) $\times$ (10 + 50 + 50) = 61.

   Solving for hit rate gives 98%.

2. Assuming a TLB hit rate of 50%, how does the AMAT of this scenario compare to no TLB?

   With TLB: AMAT = (10 + 50) $\times$ 0.5 + (1 - 0.5) $\times$ (10 + 50 + 50) = 85 ns

   Without TLB: AMAT = 50 + 50 = 100 ns

3. To improve your system, you add a two level paging scheme and a cache. The cache has a 90% hit rate with a lookup time of 20 ns. Additionally, the TLB hit rate is now improved to 95%. What is the average time to read a location from memory?

   AMAT = 0.9 $\times$ 20 + 0.1 $\times$ (20 + 50) = 25 ns

   Page table stored in memory $\rightarrow$ 25 $\times$ 3 = 75 ns

   Need to incorporate TLB cost as well $\rightarrow$ 0.95 $\times$ (10 + 25) + 0.05 $\times$ (10 + 75) = 37.5 ns

Associativity Analysis

A big data startup has just hired you to help design their new memory system for a byte-addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. First, you create a direct mapped cache and a fully associative cache of the same size that uses an LRU replacement policy. You run a few tests and realize that the fully associative cache performs much worse than the direct mapped cache does. What's a possible access pattern that could cause this to happen?

2. Instead, your boss tells you to build a 8KB 2-way set associative cache with 64 byte cache blocks. How would you split a given virtual address into its tag, index, and offset numbers?

Associativity Analysis

A big data startup has just hired you to help design their new memory system for a byte-addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. First, you create a direct mapped cache and a fully associative cache of the same size that uses an LRU replacement policy. You run a few tests and realize that the fully associative cache performs much worse than the direct mapped cache does. What's a possible access pattern that could cause this to happen?

   Let's say each cache held X amount of blocks. An access pattern would be to repeatedly iterate over X + 1 consecutive blocks, which would cause everything in the fully associative cache to miss every time.

2. Instead, your boss tells you to build a 8KB 2-way set associative cache with 64 byte cache blocks. How would you split a given virtual address into its tag, index, and offset numbers?

Associativity Analysis

A big data startup has just hired you to help design their new memory system for a byte-addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

1. First, you create a direct mapped cache and a fully associative cache of the same size that uses an LRU replacement policy. You run a few tests and realize that the fully associative cache performs much worse than the direct mapped cache does. What's a possible access pattern that could cause this to happen?
    Let's say each cache held X amount of blocks. An access pattern would be to repeatedly iterate over X + 1 consecutive blocks, which would cause everything in the fully associative cache to miss every time.

2. Instead, your boss tells you to build a 8KB 2-way set associative cache with 64 byte cache blocks. How would you split a given virtual address into its tag, index, and offset numbers?
    # offset bits = $\log_2$ (block size) = 6
    # set bits = $\log_2$ (# sets) = $\log_2$((cache size ÷ associativity) ÷ # blocks) = $\log_2(2^{13-1} ÷ 2^6)$ = 6
    # tag bits = # virtual address bits - # set bits - # offset bits = 32 - 6 - 6 = 20

Replacement Roulette

Assume your program has the following memory access pattern: A, B, C, D, A, B, D, C, B, A.

1.   How many misses will you get with FIFO?

2.   How many misses will you get with LRU?

3.   How many misses will you get with MIN?

Replacement Roulette

Assume your program has the following memory access pattern: A, B, C, D, A, B, D, C, B, A.

1. How many misses will you get with FIFO?

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

2. How many misses will you get with LRU?

3. How many misses will you get with MIN?

Replacement Roulette

Assume your program has the following memory access pattern: A, B, C, D, A, B, D, C, B, A.

1. How many misses will you get with FIFO?

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

2. How many misses will you get with LRU?

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

3. How many misses will you get with MIN?

Replacement Roulette

Assume your program has the following memory access pattern: A, B, C, D, A, B, D, C, B, A.

1. How many misses will you get with FIFO?

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

2. How many misses will you get with LRU?

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

3. How many misses will you get with MIN?

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

Replacement Roulette

Assume your program has the following memory access pattern: A, B, C, D, A, B, D, C, B, A.

4. If we increase the cache size, are we always guaranteed to get better cache performance? Explain for FIFO, LRU, and MIN.

Replacement Roulette

Assume your program has the following memory access pattern: A, B, C, D, A, B, D, C, B, A.

4.   If we increase the cache size, are we always guaranteed to get better cache performance? Explain for FIFO, LRU, and MIN.

FIFO suffers from Belady's anomaly, so there isn't a guarantee.

LRU and MIN are stack algorithms → contents of a cache with size S is always a subset of the contents of a cache with size S + 1.

On the Clock

1. Suppose that we have a 10-10-12 virtual address split using a two-level paging scheme. Assume that the physical address is 32-bit as well and PTE is 4 bytes. Show the format of a page table entry (PTE) complete with bits required to support the clock algorithm.

On the Clock

1. Suppose that we have a 10-10-12 virtual address split using a two-level paging scheme. Assume that the physical address is 32-bit as well and PTE is 4 bytes. Show the format of a page table entry (PTE) complete with bits required to support the clock algorithm.

   # physical page number bits = # physical address bits - # offset bits

   $$= 32 - 12$$

   $$= 20$$

   Have 12 bits remaining for clock algorithm, need dirty, use, writable, and valid bits.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

1. Set use bit = 1 (don't advance clock hand).
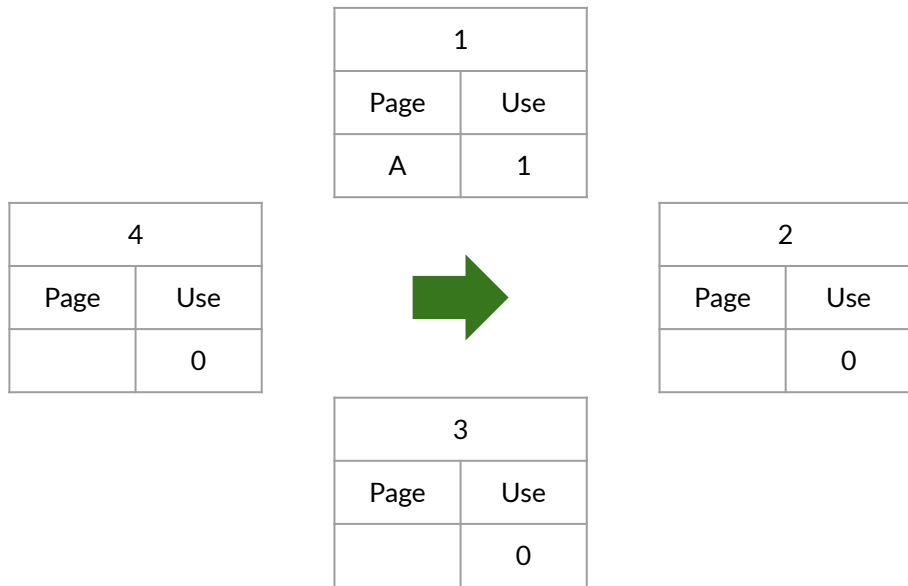
Miss

1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
	0

4	
Page	Use
	0

2	
Page	Use
	0

3	
Page	Use
	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

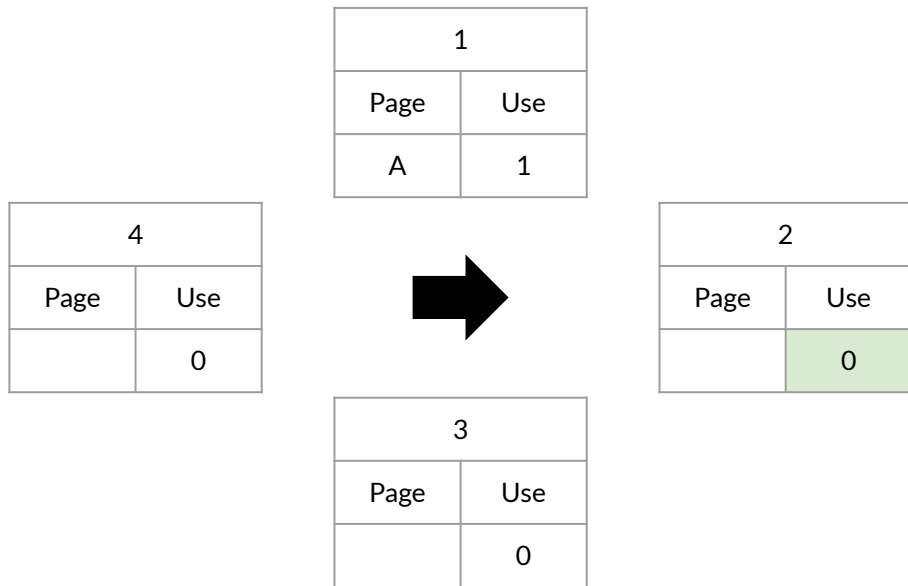Hit

1. Set use bit = 1 (don't advance clock hand).

Miss

1. **Check use bit.**

    0 → evict entry (if necessary), bring in page, set use bit = 1.

    1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1	
Page	Use
	0

4	
Page	Use
	0

2	
Page	Use
	0

3	
Page	Use
	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
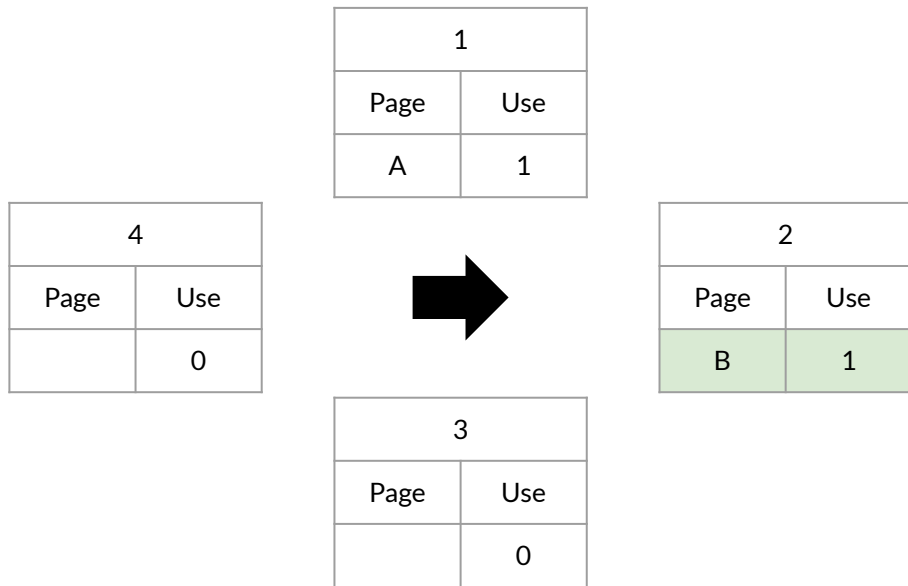
1. Set use bit = 1 (don't advance clock hand).

Miss

1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1	
Page	Use
A	1

4	
Page	Use
	0

2	
Page	Use
	0

3	
Page	Use
	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
	0

➡

2	
Page	Use
	0

3	
Page	Use
	0

Hit

1. Set use bit = 1 (don't advance clock hand).
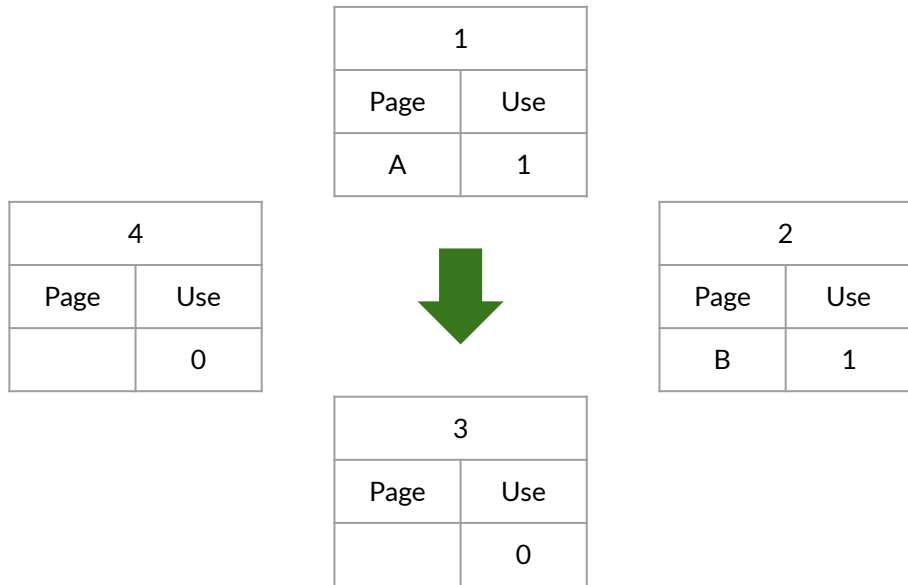
Miss

1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, **B**, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
	0

2	
Page	Use
	0

3	
Page	Use
	0

Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, **B**, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
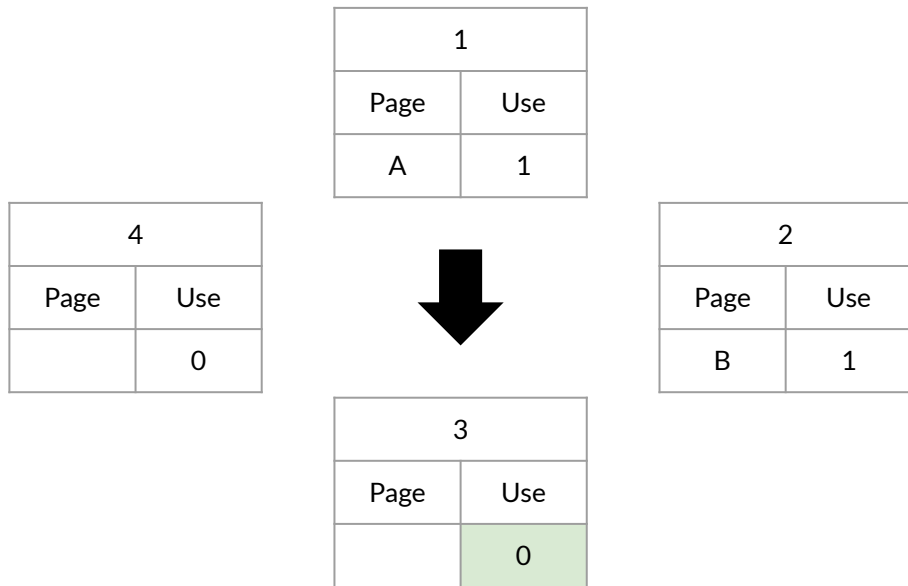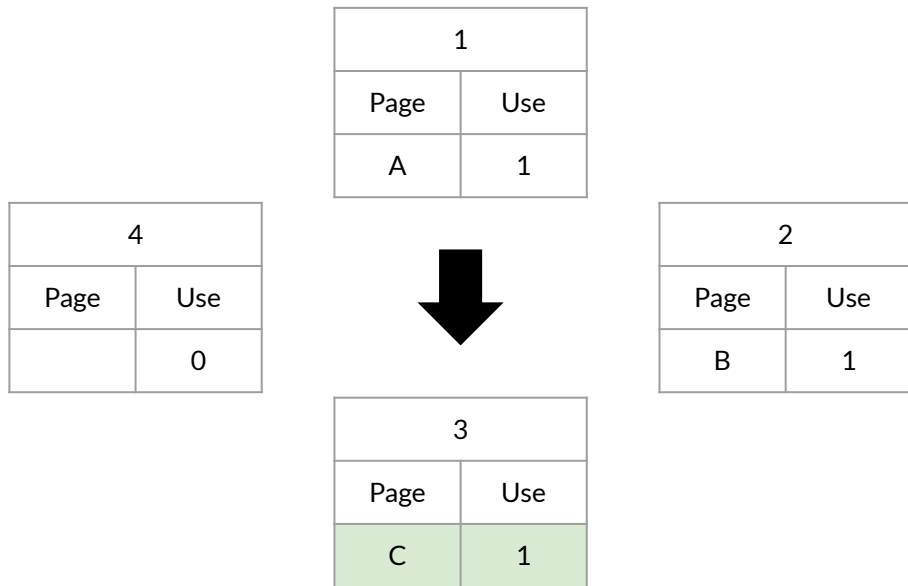1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	1

4	
Page	Use
	0

2	
Page	Use
B	1

3	
Page	Use
	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, **B**, C, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
	0

2	
Page	Use
B	1

3	
Page	Use
	0

Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, **C**, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

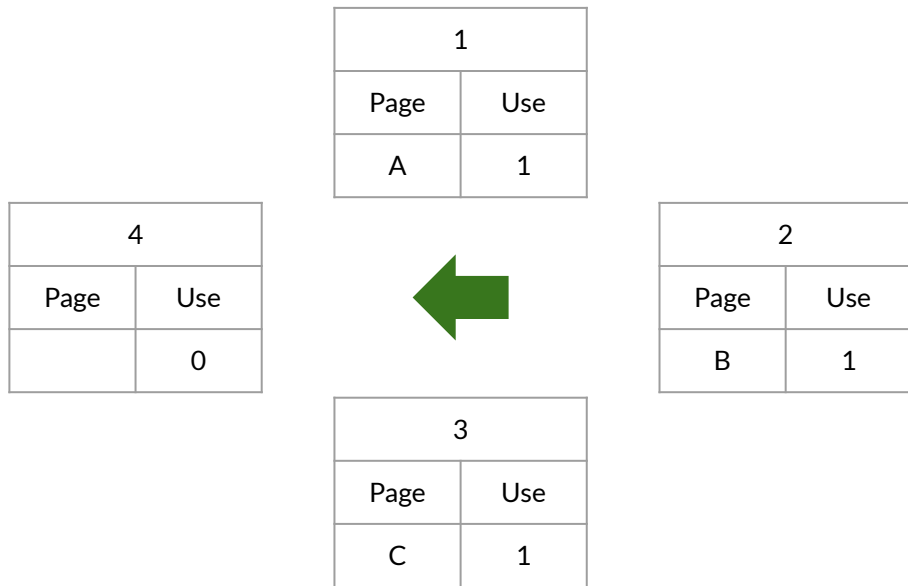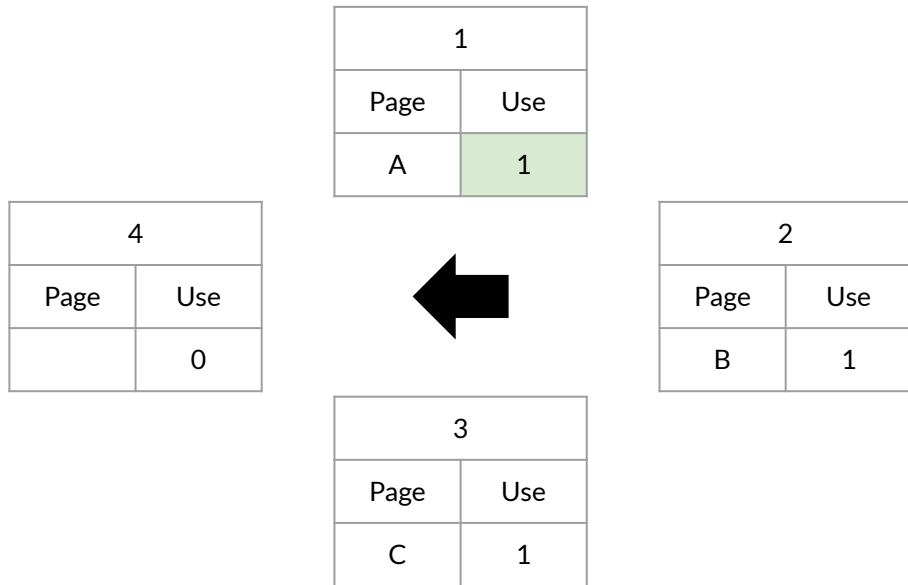Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1

Page	Use
A	1

4

Page	Use
	0

2

Page	Use
B	1

3

Page	Use
	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, **C**, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?
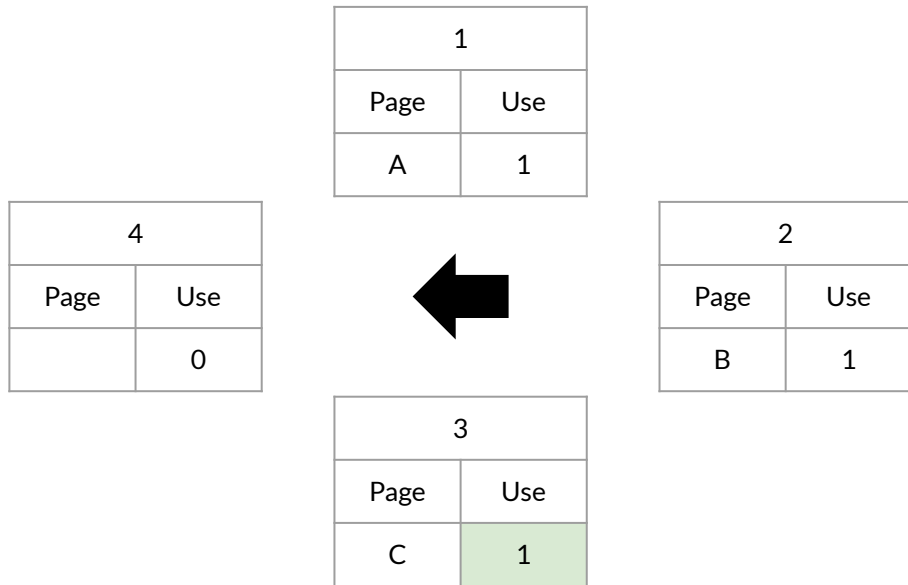
Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1

Page	Use
A	1

4

Page	Use
	0

2

Page	Use
B	1

3

Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, **C**, A, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
	0

2	
Page	Use
B	1

3	
Page	Use
C	1

Hit
1. Set use bit = 1 (don't advance clock hand).
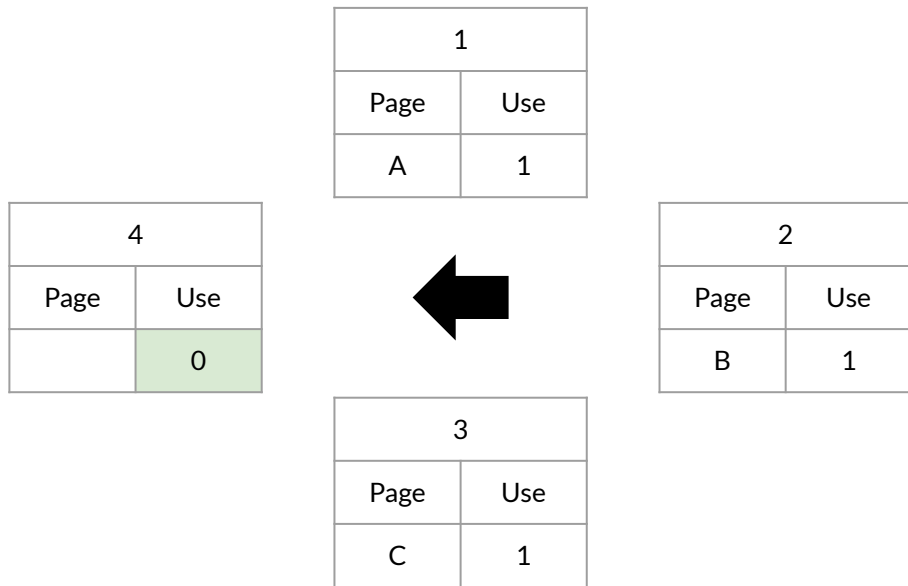
Miss
1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, **A**, C, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

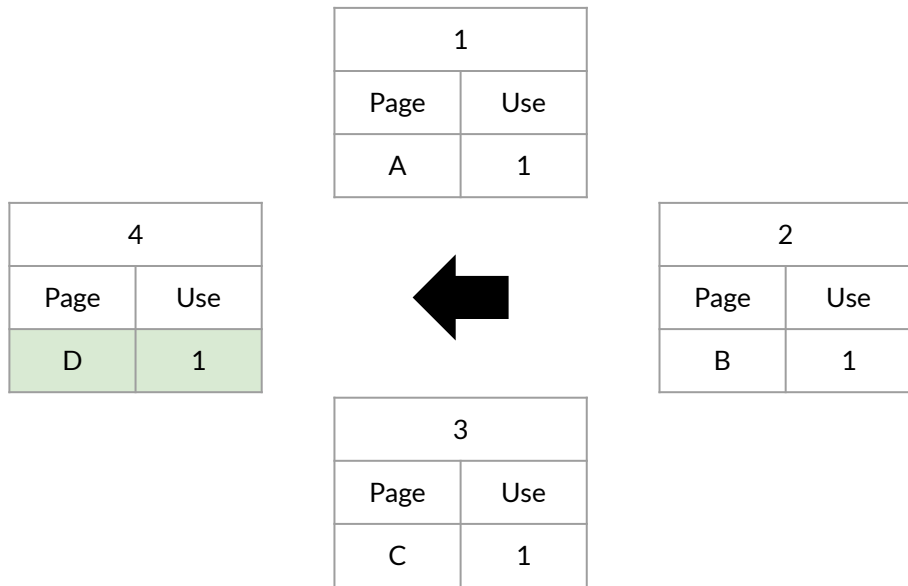1. **Set use bit = 1 (don't advance clock hand).**

Miss

1. Check use bit.

    0 → evict entry (if necessary), bring in page, set use bit = 1.

    1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1	
Page	Use
A	1

4	
Page	Use
	0

2	
Page	Use
B	1

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, **C**, D, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

1. **Set use bit = 1 (don't advance clock hand).**

Miss

1. Check use bit.
   $0 \rightarrow$ evict entry (if necessary), bring in page, set use bit = 1.
   $1 \rightarrow$ set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	1

2	
Page	Use
B	1

4	
Page	Use
	0

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, **D**, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?
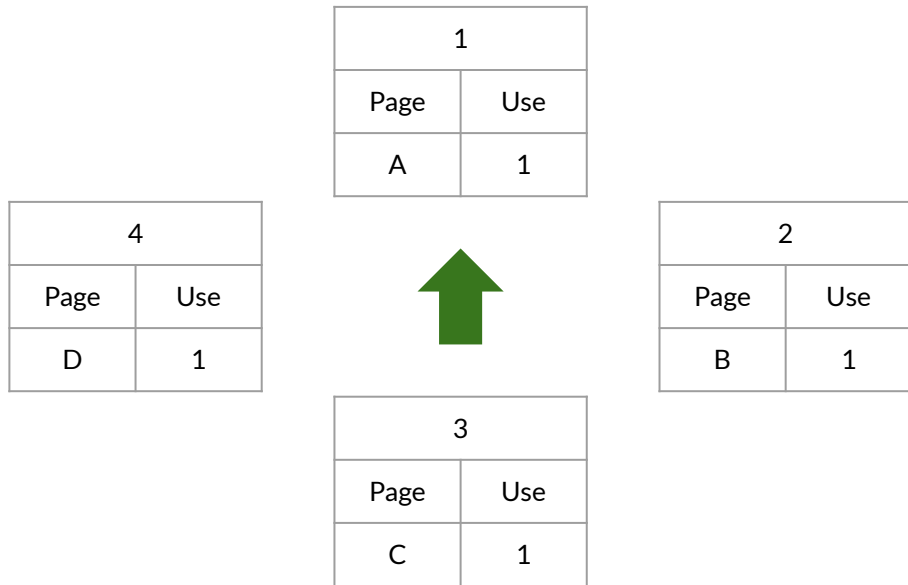
Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	1

2	
Page	Use
B	1

3	
Page	Use
C	1

4	
Page	Use
	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, **D**, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

Hit
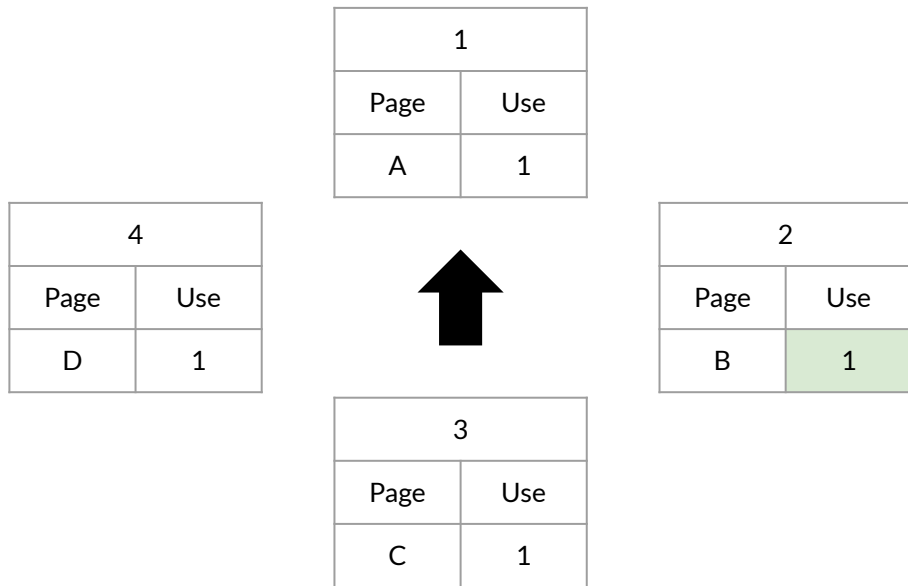1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, **D**, B, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. Check use bit.
    0 → evict entry (if necessary), bring in page, set use bit = 1.
    1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, **B**, D, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	1

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

Hit

1. **Set use bit = 1 (don't advance clock hand).**
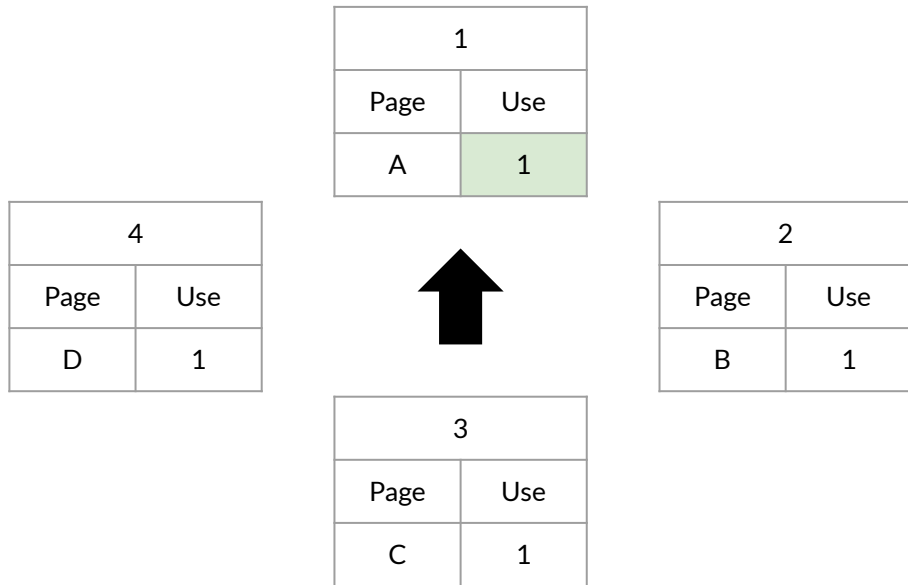
Miss

1. Check use bit.

   0 → evict entry (if necessary), bring in page, set use bit = 1.

   1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, **D**, A, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

1. **Set use bit = 1 (don't advance clock hand).**

Miss

1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	1

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, **A**, E, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

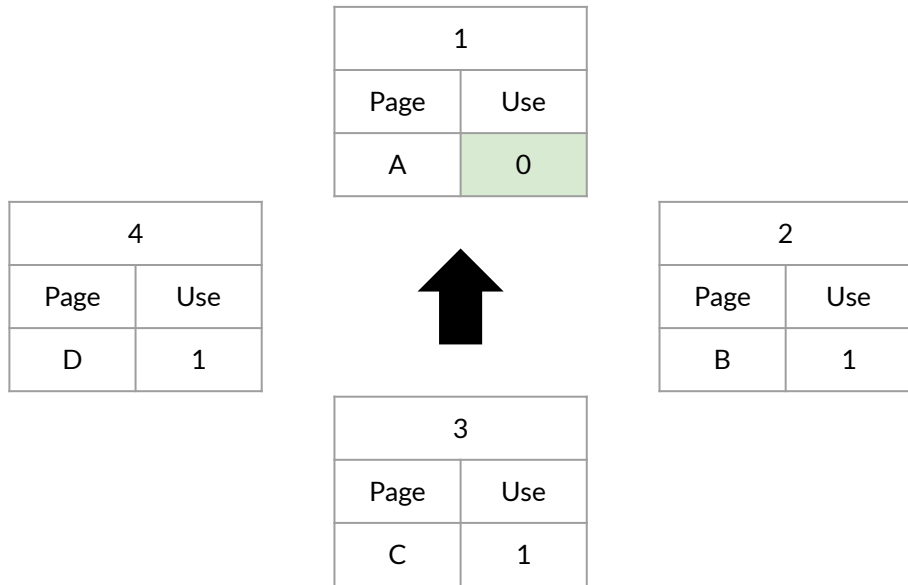1. **Set use bit = 1 (don't advance clock hand).**

Miss

1. Check use bit.

   0 → evict entry (if necessary), bring in page, set use bit = 1.

   1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1	
Page	Use
A	1

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
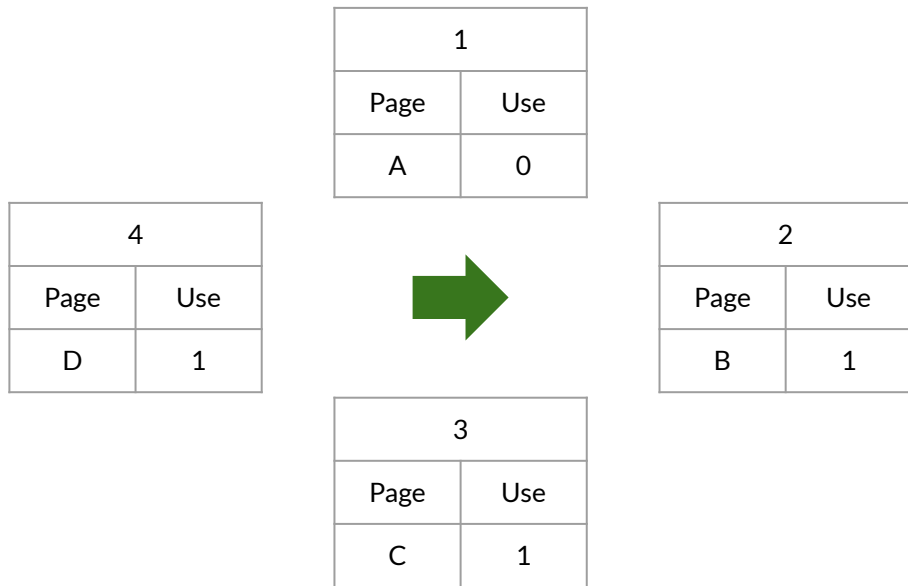1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**

0 → evict entry (if necessary), bring in page, set use bit = 1.

1 → set use bit = 0.

2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	1

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?
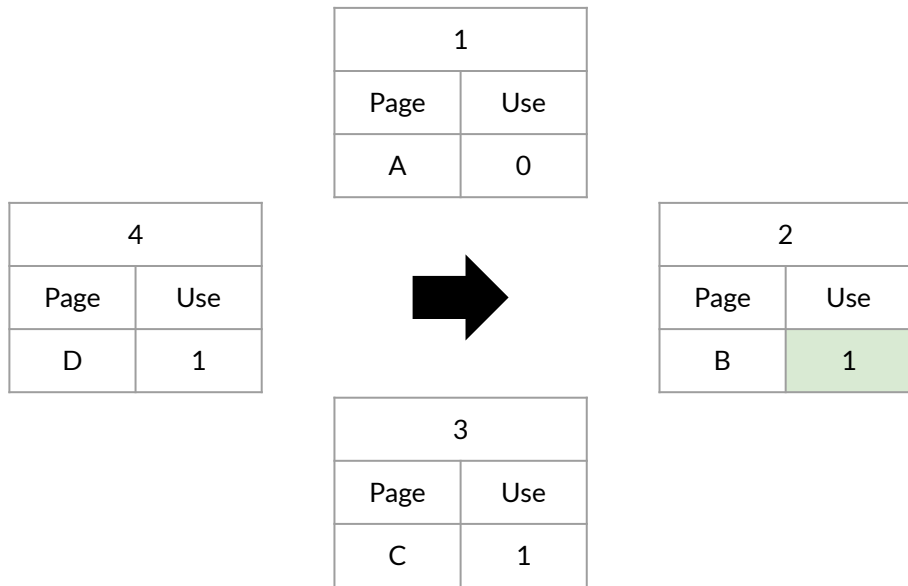
Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

    A, B, C, A, C, D, B, D, A, **E**, B, F

    What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

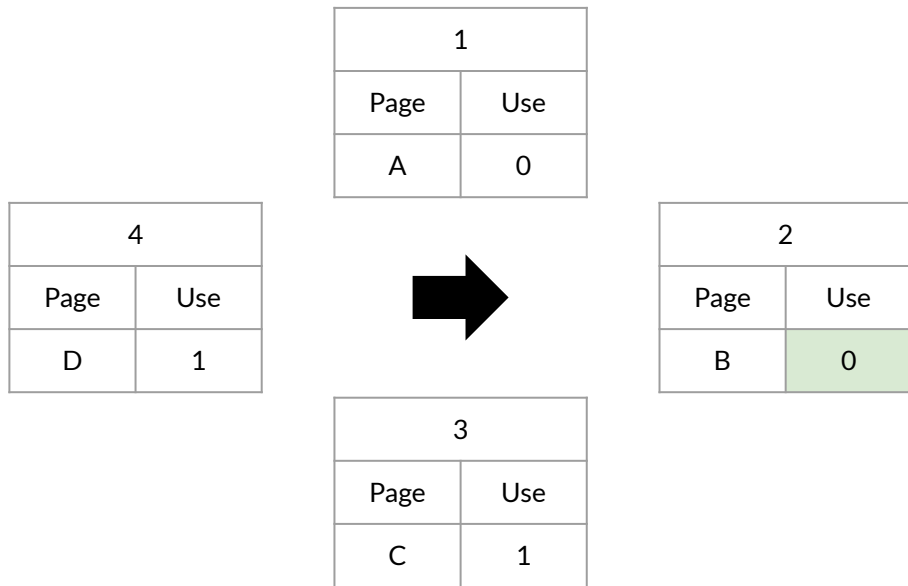1. Set use bit = 1 (don't advance clock hand).

Miss

1. Check use bit.

    0 → evict entry (if necessary), bring in page, set use bit = 1.

    1 → set use bit = 0.

2. **Advance clock hand.**

3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

3	
Page	Use
C	1

2	
Page	Use
B	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
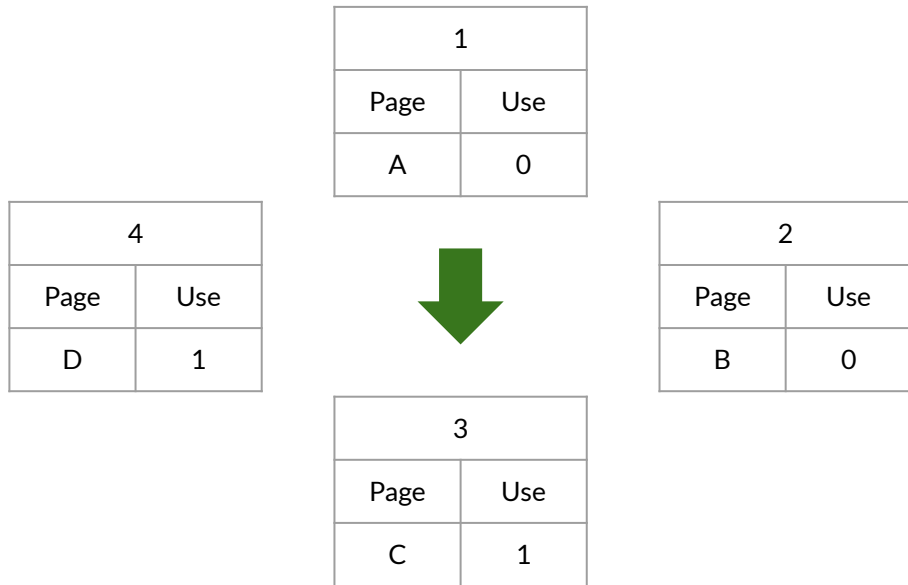 1. Set use bit = 1 (don't advance clock hand).

Miss
  1. **Check use bit.**
     0 → evict entry (if necessary), bring in page, set use bit = 1.
     1 → set use bit = 0.
  2. Advance clock hand.
  3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	1

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

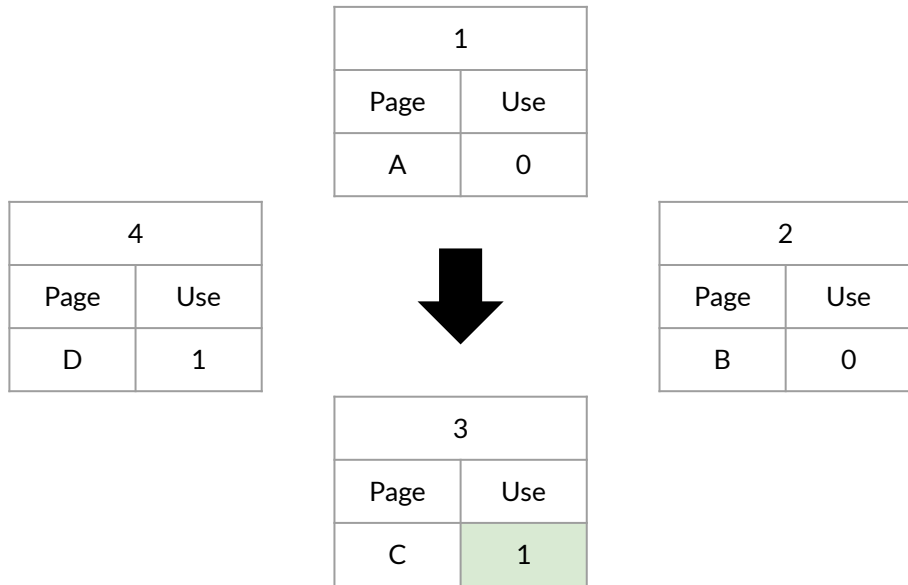1. Set use bit = 1 (don't advance clock hand).

Miss

1. **Check use bit.**

   0 → evict entry (if necessary), bring in page, set use bit = 1.

   1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	0

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	0

3	
Page	Use
C	1

Hit

1. Set use bit = 1 (don't advance clock hand).

Miss

1. Check use bit.

   0 → evict entry (if necessary), bring in page, set use bit = 1.

   1 → set use bit = 0.

2. **Advance clock hand.**

3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
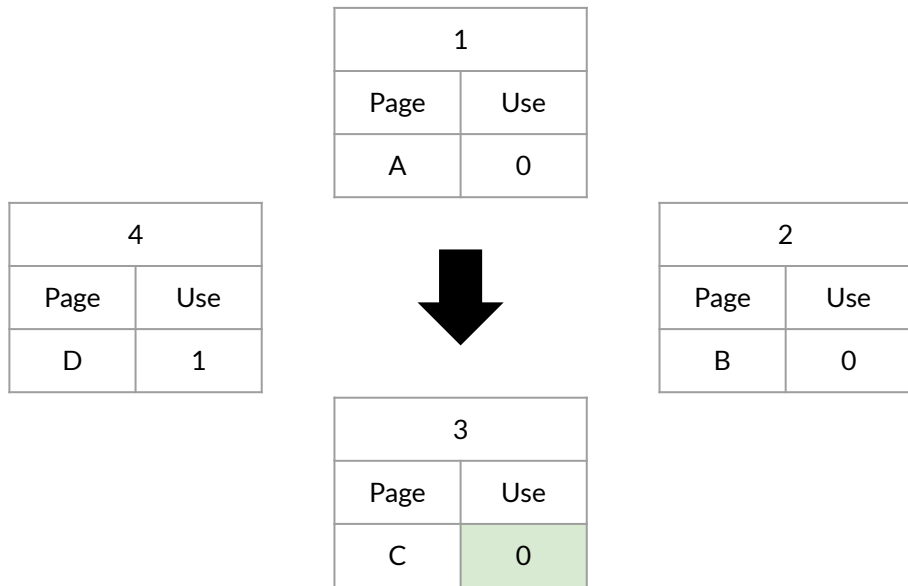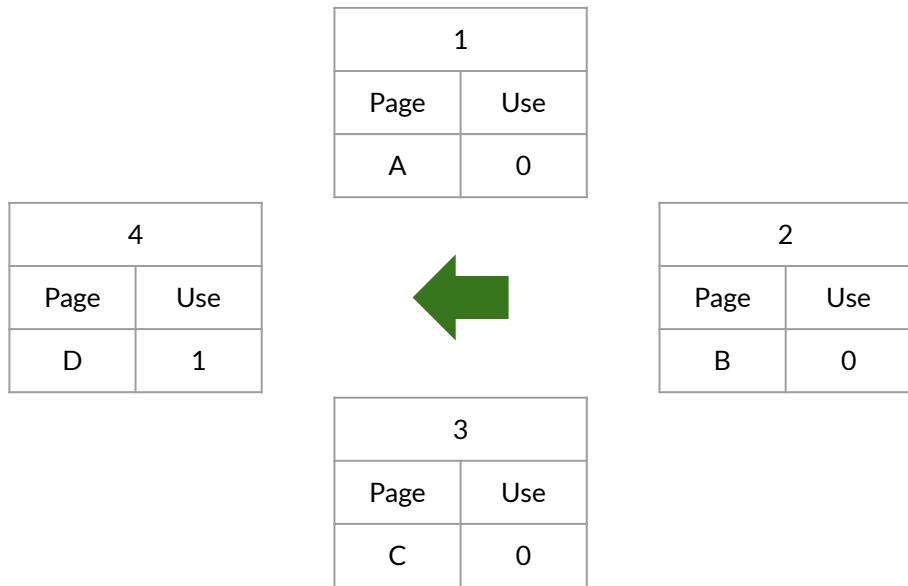1. Set use bit = 1 (don't advance clock hand).
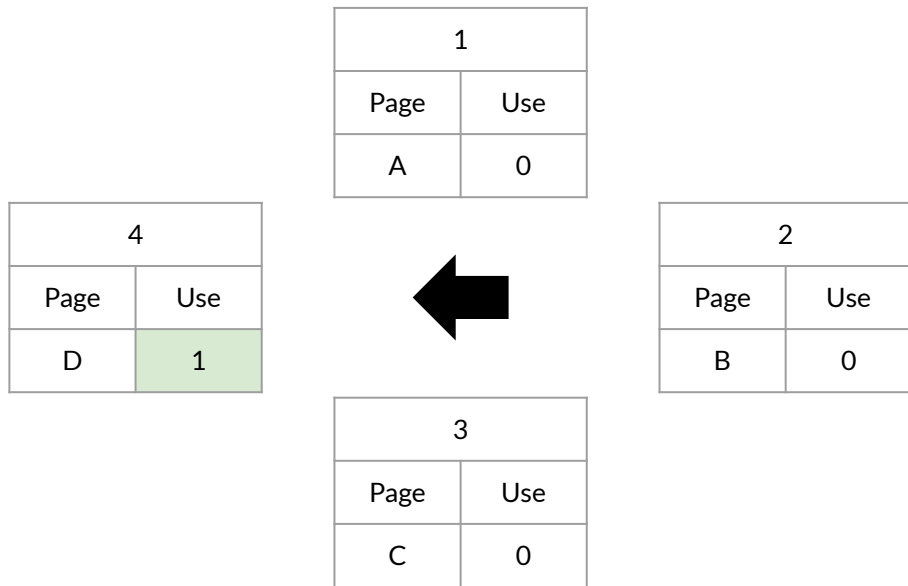
Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	0

3	
Page	Use
C	1

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

1. Set use bit = 1 (don't advance clock hand).

Miss

1. **Check use bit.**
   $0 \rightarrow$ evict entry (if necessary), bring in page, set use bit = 1.
   $1 \rightarrow$ set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	0

3	
Page	Use
C	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	0

3	
Page	Use
C	0

Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
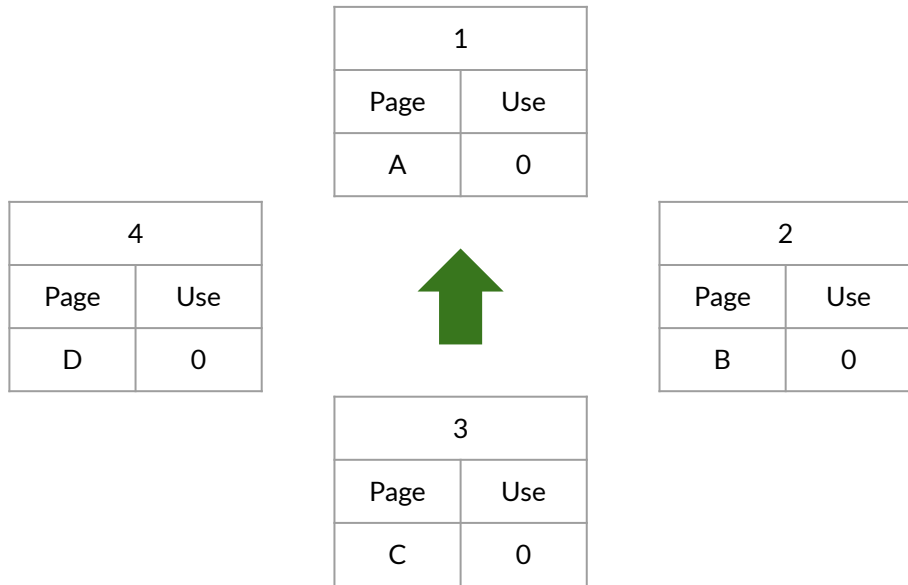1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
A	0

4	
Page	Use
D	1

2	
Page	Use
B	0

3	
Page	Use
C	0

On the Clock

2.   Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
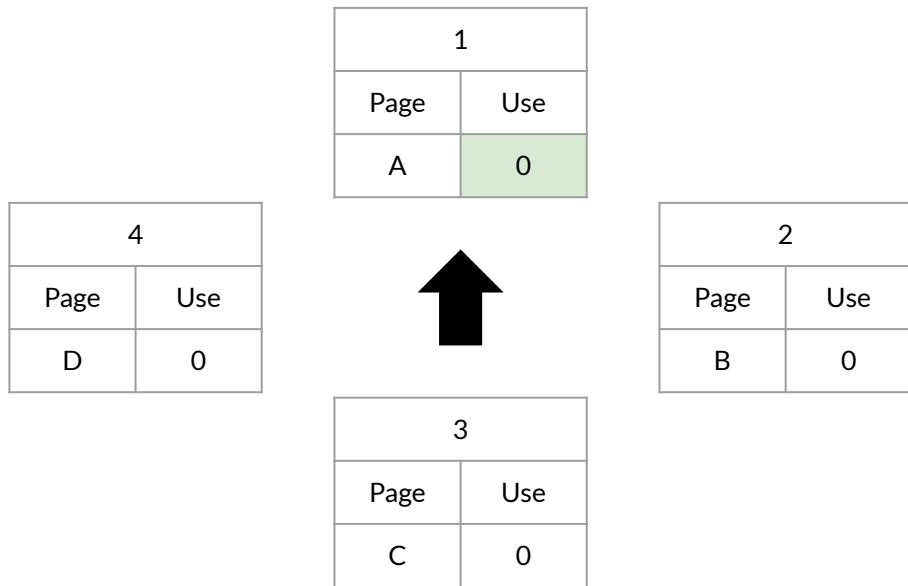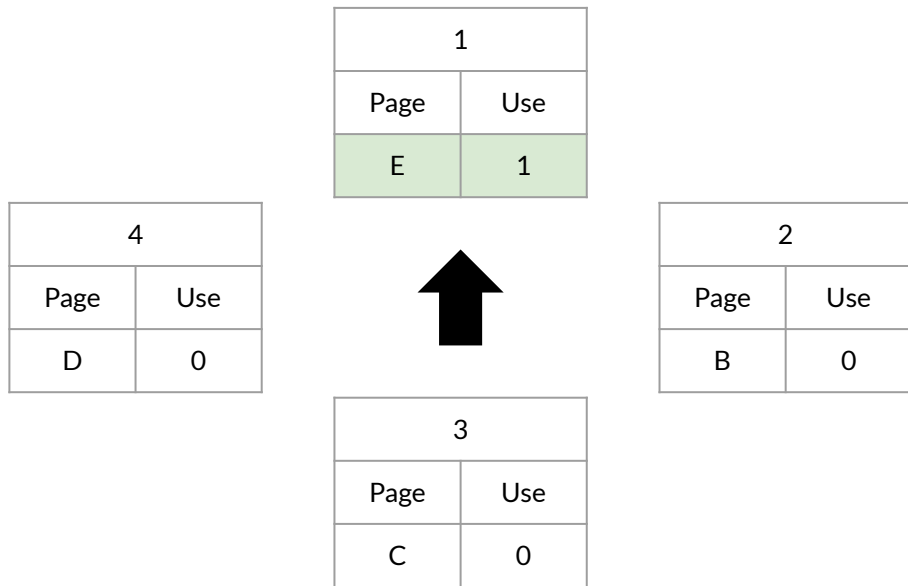1.   Set use bit = 1 (don't advance clock hand).

Miss
1.   Check use bit.
     0 → evict entry (if necessary), bring in page, set use bit = 1.
     1 → set use bit = 0.
2.   **Advance clock hand.**
3.   Repeat if necessary.

1
Page	Use
A	0

4
Page	Use
D	0

2
Page	Use
B	0

3
Page	Use
C	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
A	0

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
C	0

Hit
1. Set use bit = 1 (don't advance clock hand).
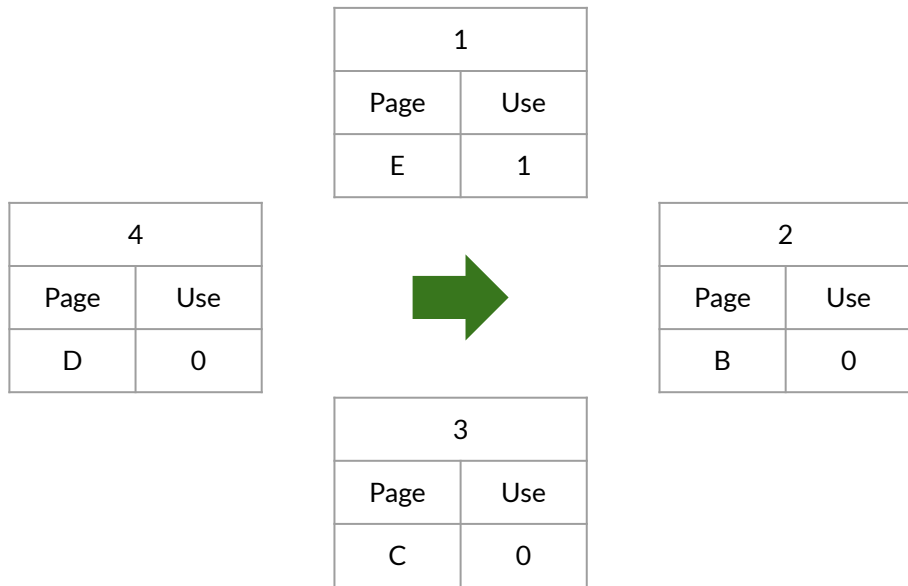
Miss
1. **Check use bit.**
    0 → evict entry (if necessary), bring in page, set use bit = 1.
    1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
C	0

Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, **E**, B, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
1. Set use bit = 1 (don't advance clock hand).
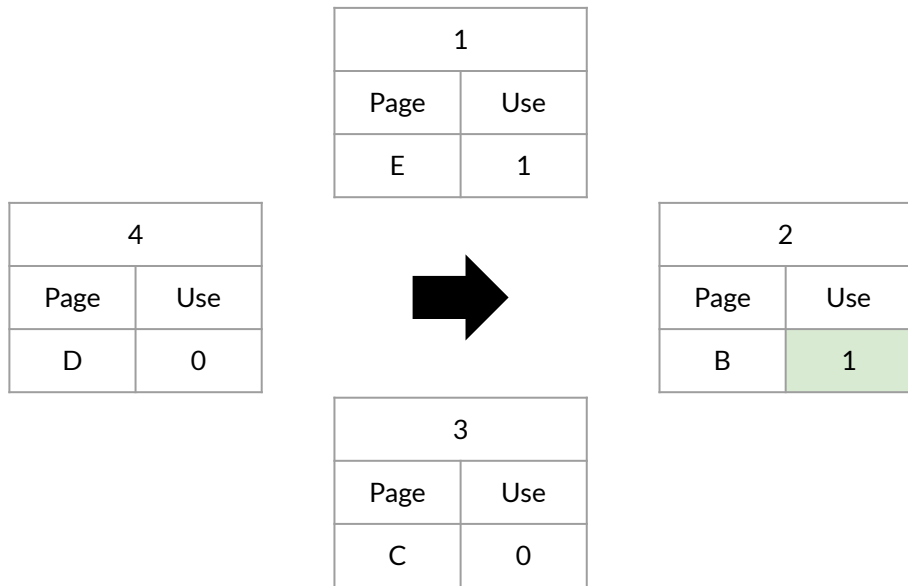
Miss
1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
C	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, **B**, F

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	1

3	
Page	Use
C	0

Hit

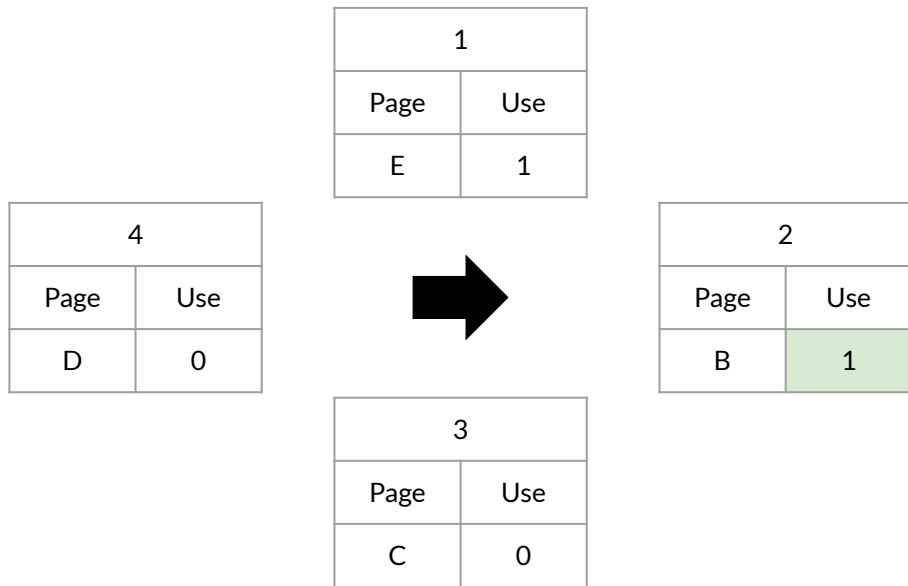1. **Set use bit = 1 (don't advance clock hand).**

Miss

1. Check use bit.

   0 → evict entry (if necessary), bring in page, set use bit = 1.

   1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

    A, B, C, A, C, D, B, D, A, E, B, **F**

    What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

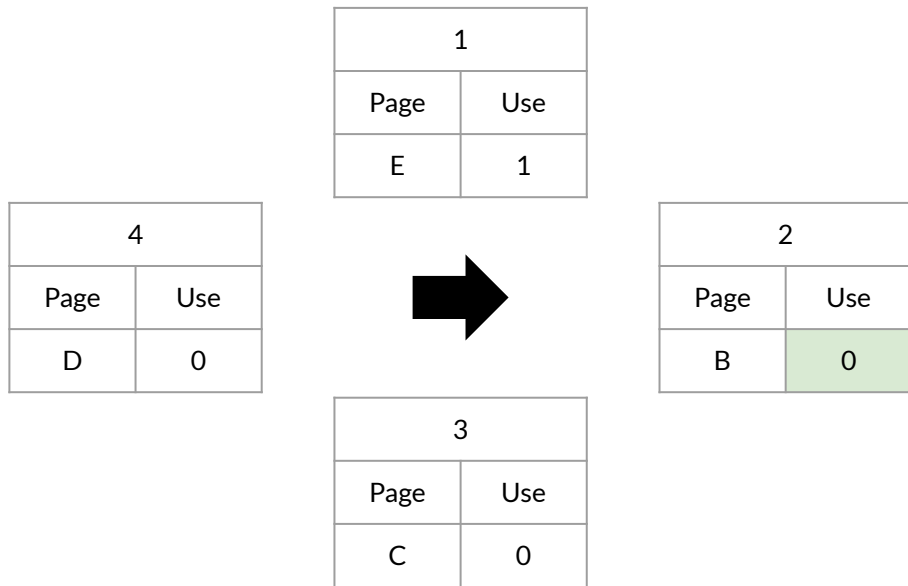1. Set use bit = 1 (don't advance clock hand).

Miss

1. **Check use bit.**

    0 → evict entry (if necessary), bring in page, set use bit = 1.

    1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	1

3	
Page	Use
C	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, **F**

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
1. Set use bit = 1 (don't advance clock hand).
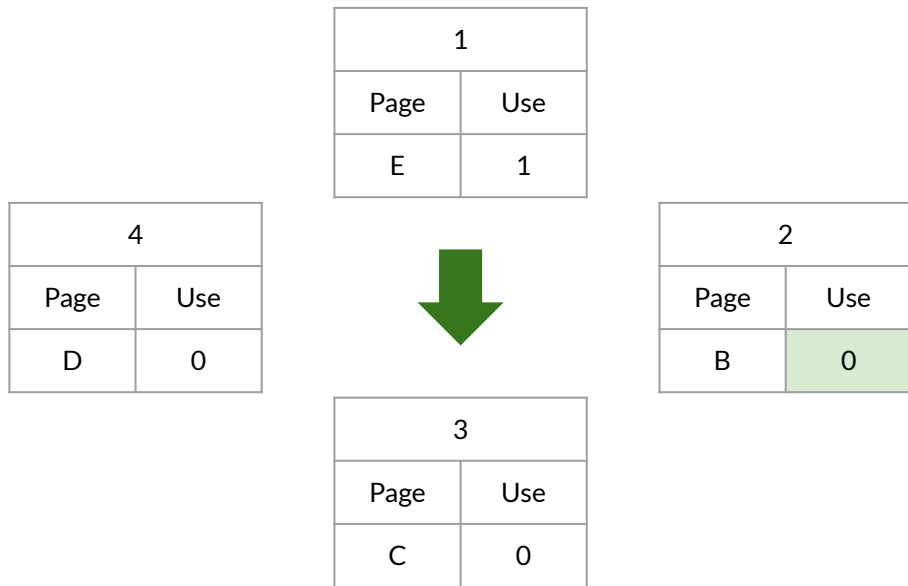
Miss
1. **Check use bit.**
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. Advance clock hand.
3. Repeat if necessary.

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
C	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, **F**

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit
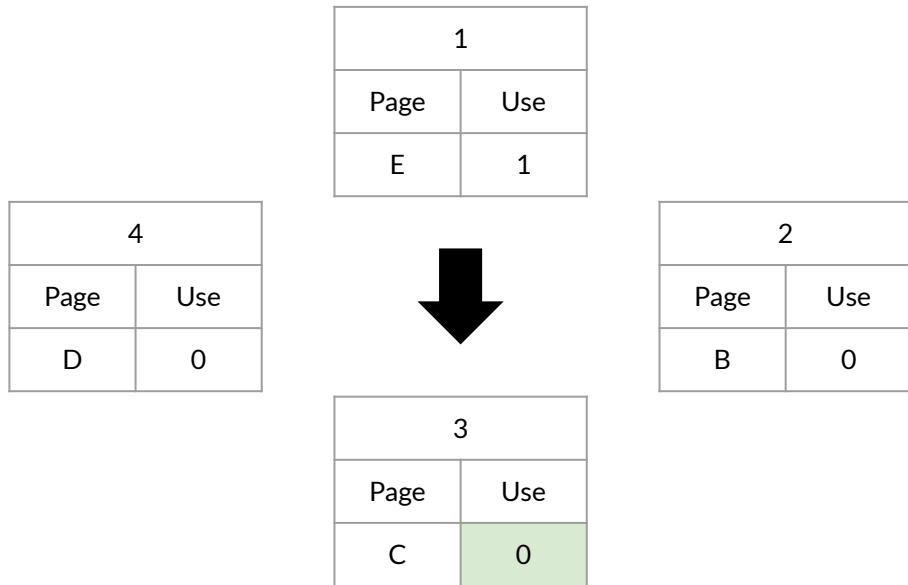1. Set use bit = 1 (don't advance clock hand).

Miss
1. Check use bit.
    0 → evict entry (if necessary), bring in page, set use bit = 1.
    1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
C	0

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, **F**

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

Hit

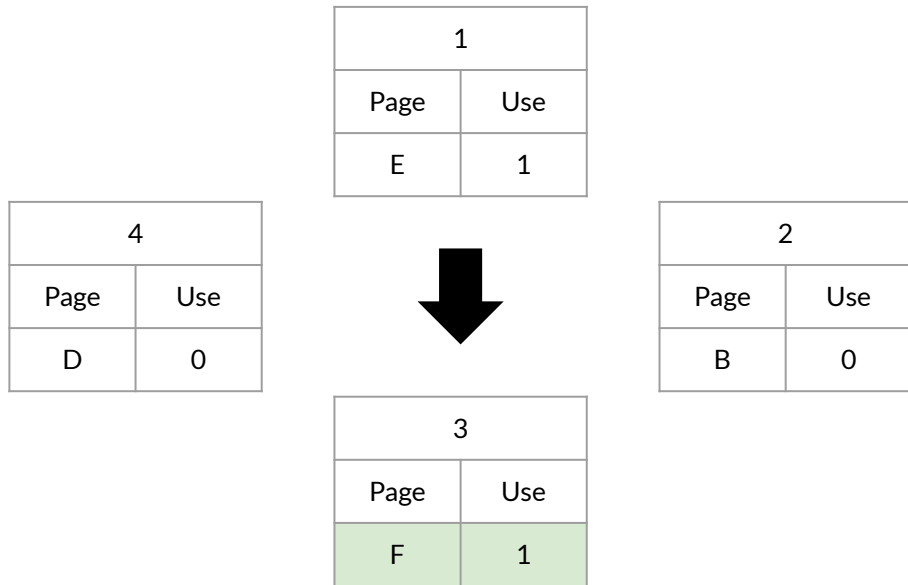1. Set use bit = 1 (don't advance clock hand).

Miss

1. **Check use bit.**

    0 → evict entry (if necessary), bring in page, set use bit = 1.

    1 → set use bit = 0.

2. Advance clock hand.

3. Repeat if necessary.

1

Page	Use
E	1

4

Page	Use
D	0

2

Page	Use
B	0

3

Page	Use
C	0

On the Clock

2.   Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, **F**

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
F	1

Hit
1.   Set use bit = 1 (don't advance clock hand).
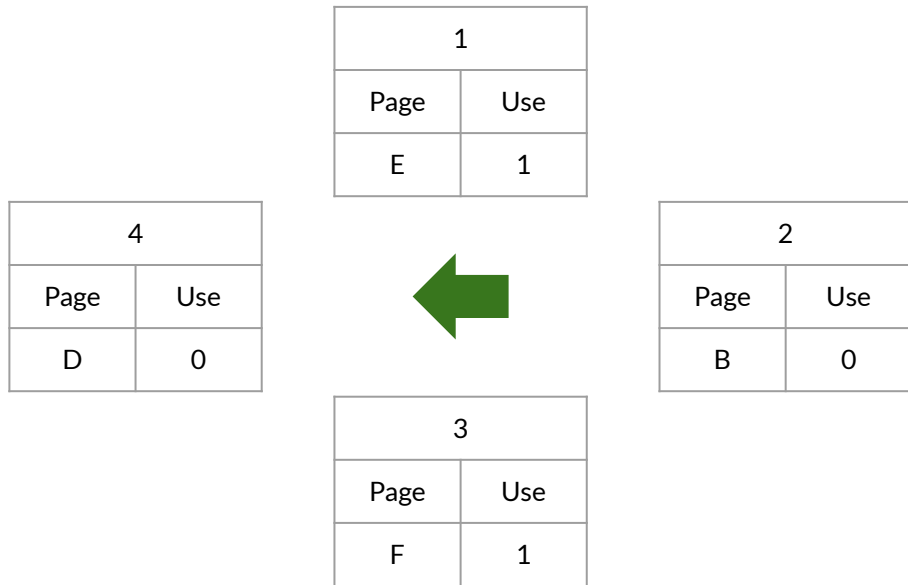
Miss
1.   **Check use bit.**
     0 → evict entry (if necessary), bring in page, set use bit = 1.
     1 → set use bit = 0.
2.   Advance clock hand.
3.   Repeat if necessary.

On the Clock

2. Assume that physical memory can hold at most four pages. The program you run has the following memory access pattern

A, B, C, A, C, D, B, D, A, E, B, **F**

What pages remain in memory at the end of the following sequence of page table operations? What are the use bits set to for each of these pages?

1	
Page	Use
E	1

4	
Page	Use
D	0

2	
Page	Use
B	0

3	
Page	Use
F	1

Hit
1. Set use bit = 1 (don't advance clock hand).

Miss
1. Check use bit.
   0 → evict entry (if necessary), bring in page, set use bit = 1.
   1 → set use bit = 0.
2. **Advance clock hand.**
3. Repeat if necessary.