CS111, Lecture 25
The Clock Algorithm

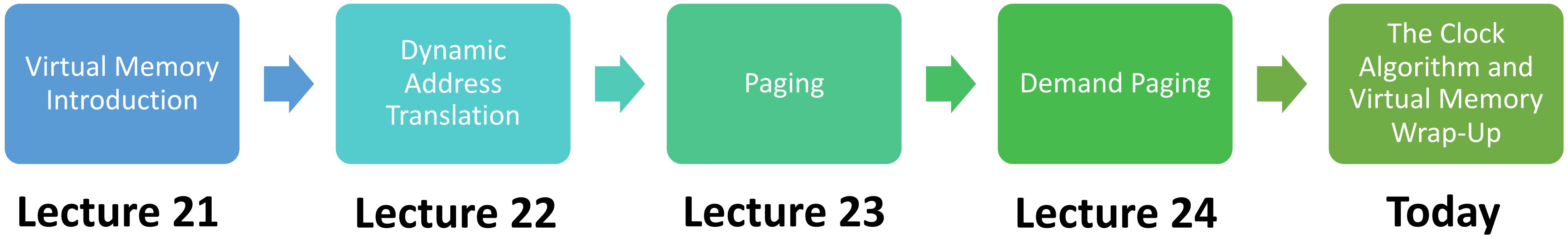
😷 masks recommended

Announcements

- assign6 Released!
- assign6 YEAH hours TODAY 3:30-4:20PM in 160-315
- Almost done grading assign4!
- In the process of reviewing midterm regrade requests

Topic 4: Virtual Memory - How can one set of memory be shared among several processes? How can the operating system manage access to a limited amount of system memory?

CS111 Topic 4: Virtual Memory


Virtual Memory Introduction
Dynamic Address Translation
Paging
Demand Paging
The Clock Algorithm and Virtual Memory Wrap-Up
Lecture 21
Lecture 22
Lecture 23
Lecture 24
Today

assign6: implement *demand paging* system to translate addresses and load/store memory contents for programs as needed.

Learning Goals

- Learn about tradeoffs in approaches for choosing pages to kick out of memory
- Walk through the implementation of the clock algorithm, one algorithm for choosing which page to throw out

Plan For Today

- **Recap:** Demand Paging
- Page Replacement Policies
- The Clock Algorithm
- Virtual Memory summary

Plan For Today

- **Recap: Demand Paging**
- Page Replacement Policies
- The Clock Algorithm
- Virtual Memory summary

Demand Paging

Thought: if memory is in high demand, we could fill up all of memory. If a process wants another page, we may not have any more. What do we do?

Idea #1: process just can't have any more memory (not ideal)

Idea #2: let's "borrow" a used physical page – we'll store its existing contents on disk, and then use the page for this new data.   If the old contents are referenced later, we'll load them back into a physical page.

Overall goal: make physical memory look larger than it is.

Demand Paging

If we need another page but memory is full:

1. Pick a page to kick out
2. Write it to disk
3. Mark the old page map entry as not present
4. Update the new page map entry to be present and map to this physical page

Demand Paging

If the program accesses a page that was swapped to disk:

1. Triggers a page fault (not-present page accessed)
2. We see disk swap contains data for this page
3. Get a new physical page (perhaps kicking out another one)
4. Load the data from disk into that page
5. Update the page map with this new mapping

Disk Swap

We don’t always need to write a swapped-out page to disk – e.g. read-only code pages can always be loaded from executable. And we may have initial data for a page that wasn’t previously swapped out.

There are three categories of pages for swapping to disk:

1. **Read-only code pages**: read from executable when needed
2. **Initialized data pages:** on first access, read from executable.   Once loaded, save to swap file since contents may have changed.
3. **Uninitialized data pages:** e.g. stack, heap – on first access, just clear memory to all zeros. Save to swap file as needed.

Disk Swap

We don’t always need to write a swapped-out page to disk – e.g. read-only code pages can always be loaded from executable. And we may have initial data for a page that wasn’t previously swapped out.

On assign6:

- You’ll only write to disk if a page is “dirty” (modified). Page maps contain a dirty bit that is set whenever a page is modified.
- A page may have contents on disk from the executable or from a previous swap – you’ll read into memory in both cases.

Page Fetching

When should we bring pages into memory?

- Most modern OSes start with no pages loaded, load pages when referenced (“demand fetching”).
- Alternative: *prefetching* - try to predict when pages will be needed and load them ahead of time (requires predicting the future…)
  - One approach: could read many pages on a page fault instead of just 1

Which pages should we throw out of memory if we need more space?

Plan For Today

- **Recap:** Demand Paging
- **Page Replacement Policies**
- The Clock Algorithm
- Virtual Memory summary

Page Replacement

If we need another physical page but all memory is used, which page should we throw out? How do we pick?

- Random? (works surprisingly well!)
- FIFO? (throw out page that's been in memory the longest) – fairness
- Would be nice if we could pick page whose next access is farthest in the future, but we'd need to predict the future…
- LRU (least-recently-used)? Replace page that was accessed the longest time ago.

Page Replacement

If we need another physical page but all memory is used, which page should we throw out? How do we pick?

- Random? (works surprisingly well!)
- FIFO? (throw out page that's been in memory the longest) – fairness
- Would be nice if we could pick page whose next access is farthest in the future, but we'd need to predict the future…
- **LRU (least-recently-used)? Replace page that was accessed the longest time ago.**

Least-Recently-Used

How could we know which page was the least-recently used?

- Store clock time for each page on each reference?
- Scan all pages to find oldest one?

Alternative: just find an old page, not necessarily the oldest.

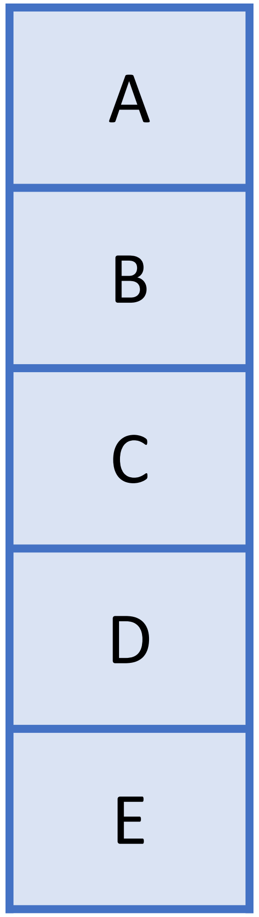
The **clock algorithm** is one implementation of this idea.

Clock algorithm key idea: rotate through pages until we find one that hasn't been referenced since the *last time* we checked it. ("second chance algorithm")

Plan For Today

- **Recap:** Demand Paging
- Page Replacement Policies
- **The Clock Algorithm**
- Virtual Memory summary

Clock Algorithm



Physical Pages

Let's say the program requests mapping page 5, but we have no more physical pages. This triggers the clock algorithm.

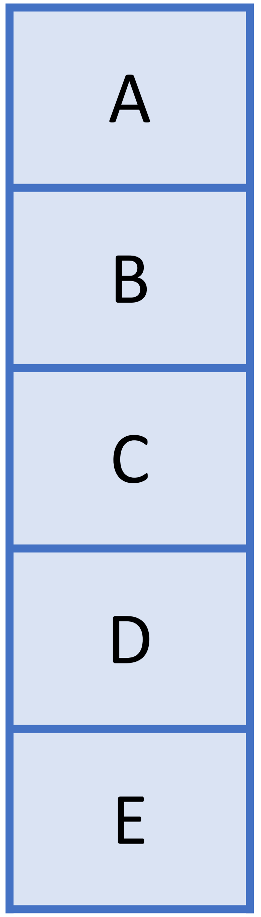
The reference bit is set to 1 whenever that page is read or written.

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	1	1
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	1
1	B	0	1	1
0	A	0	1	1

Page Map

Clock Algorithm



Physical Pages

Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.

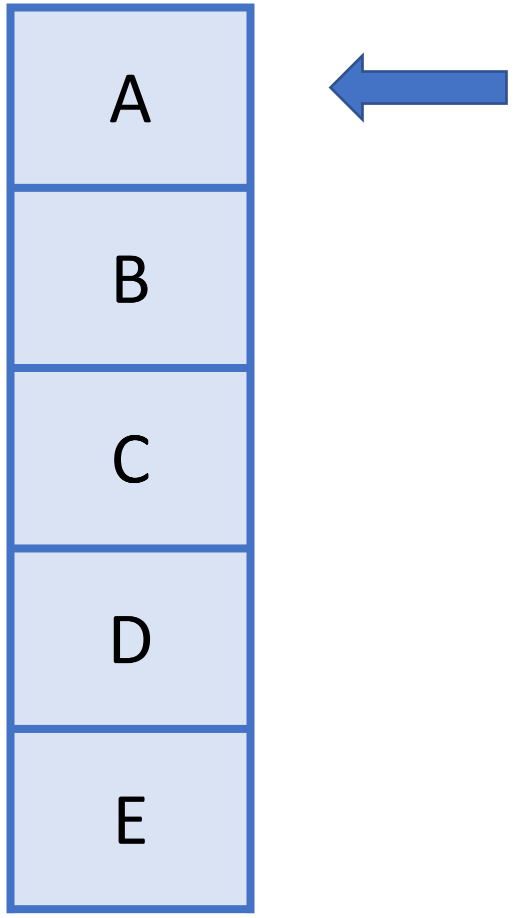
"We'll leave this page for now – but if we come back and it's still unused, we'll kick it out."

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	1	1
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	1
1	B	0	1	1
0	A	0	1	1

Page Map

Clock Algorithm



Physical Pages

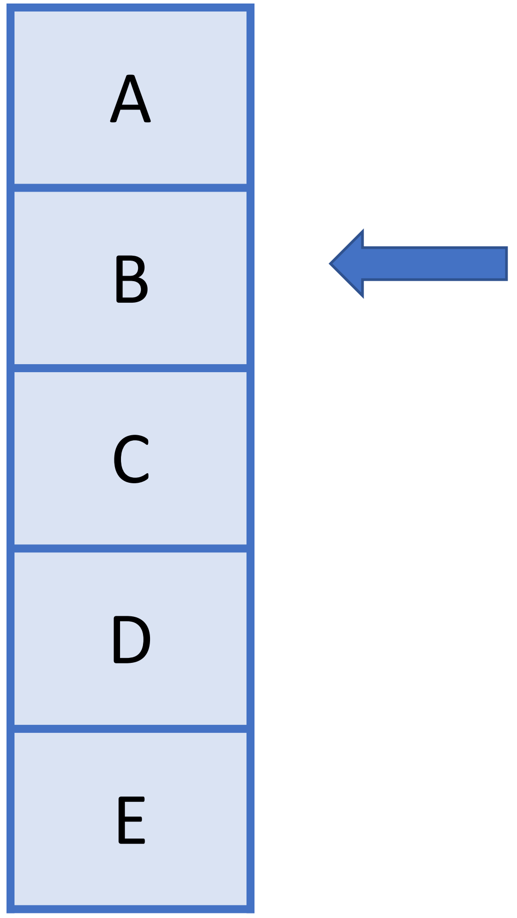
Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.

“reference” bit

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	1	1
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	1
1	B	0	1	1
0	A	0	1	0

Page Map

Clock Algorithm



Physical Pages

Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.

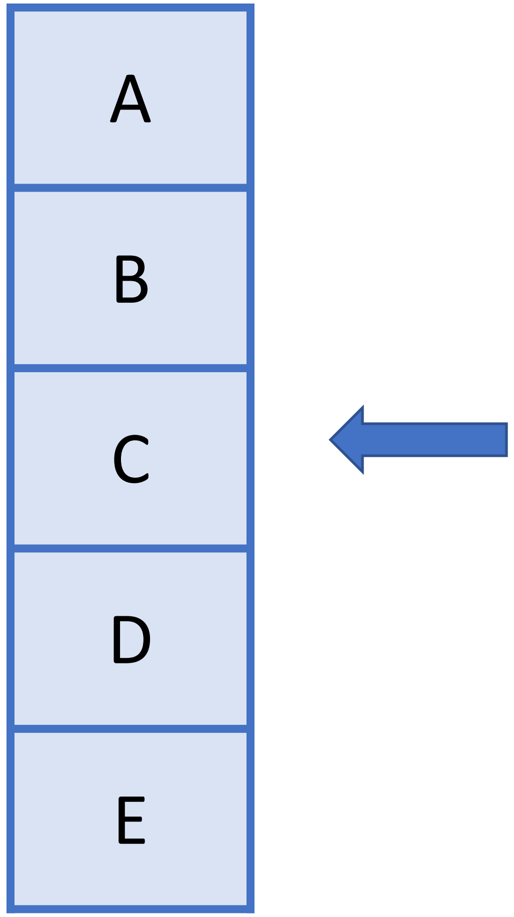
"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	1	1
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	1
1	B	0	1	0
0	A	0	1	0

Page Map

Clock Algorithm

Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.



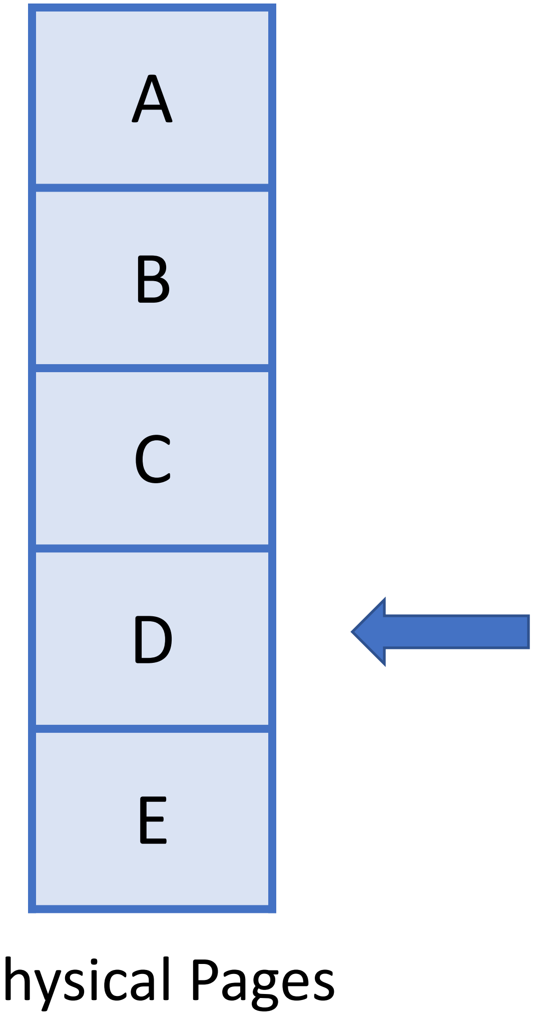
Physical Pages

“reference” bit

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	1	1
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	1	0

Page Map

Clock Algorithm



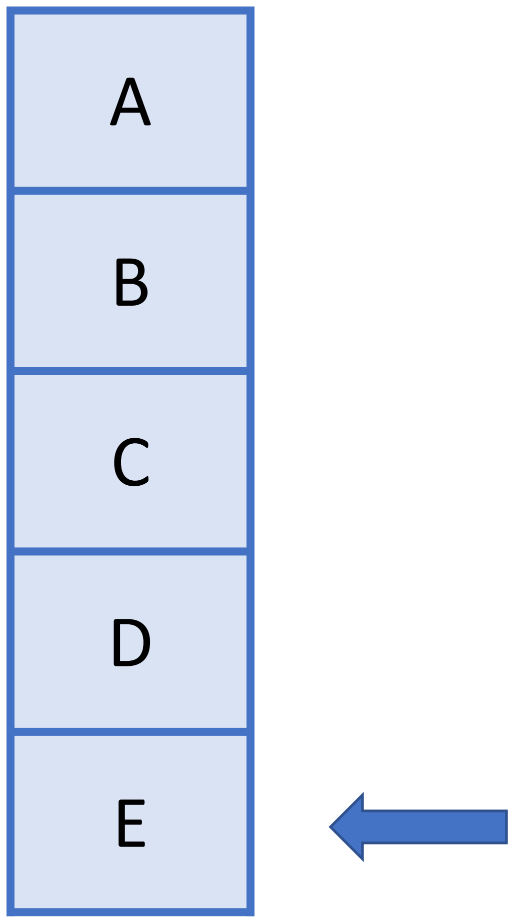
Was this page accessed recently (reference = 1)?
If so, set reference = 0 and continue.

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	1	0
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	1	0

Page Map

Clock Algorithm

Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.



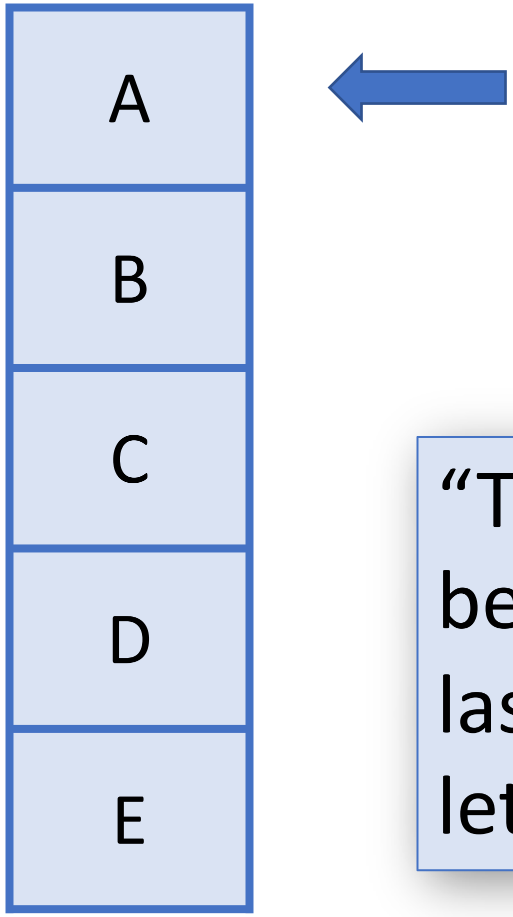
Physical Pages

“reference” bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	1	0

Page Map

Clock Algorithm



Physical Pages

Was this page accessed recently (reference = 1)? If not, this is the one we should remove.

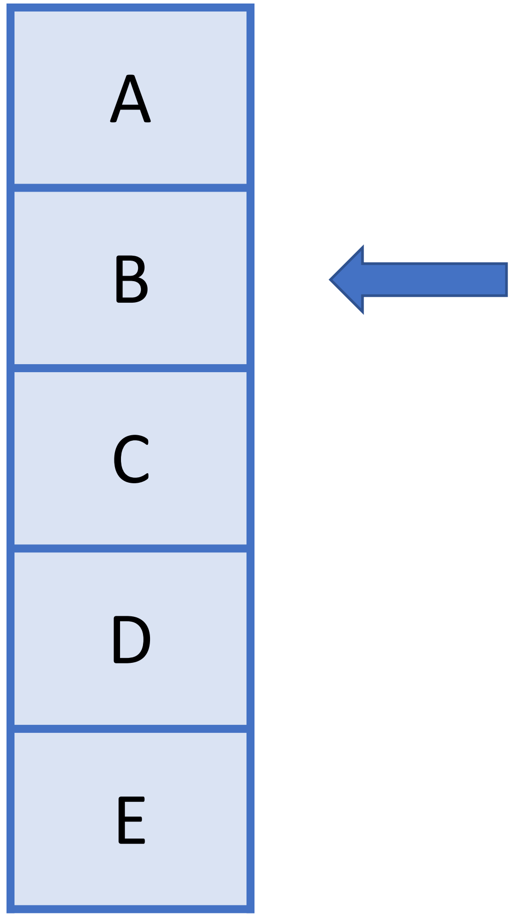
"This page hasn't been used since the last time I checked – let's remove it."

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	X	X	0	X
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	1	🟨 0

Page Map

Clock Algorithm



Physical Pages

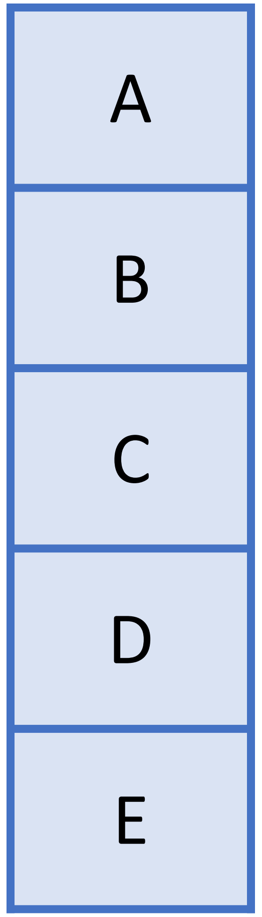
Now the clock algorithm stops, and **we remember the position of the hand for next time it runs.**

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	A	1	1	1
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	0	0

Page Map

Clock Algorithm



Physical Pages

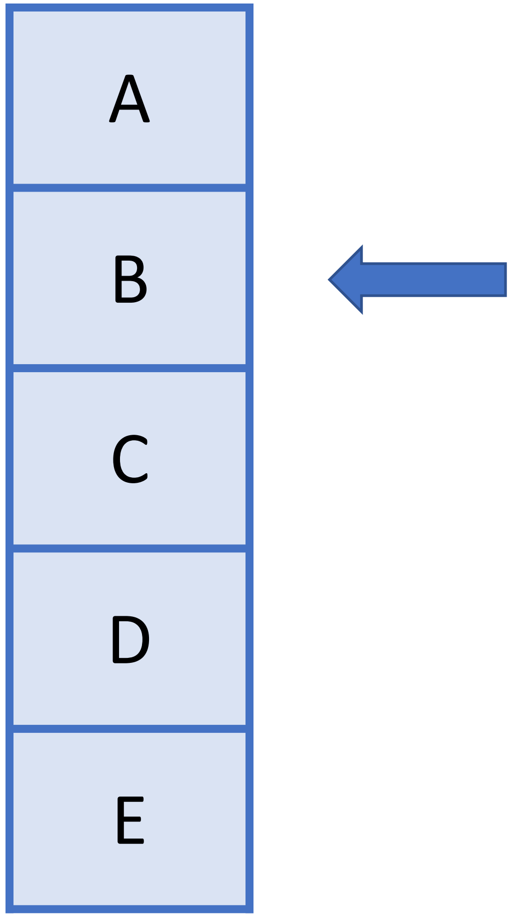
Let's say the program now requests mapping page 4. Some memory accesses have also happened.

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	A	1	1	1
4	X	X	0	X
3	X	X	0	X
2	C	1	1	1
1	B	0	1	1
0	A	0	0	0

Page Map

Clock Algorithm



Physical Pages

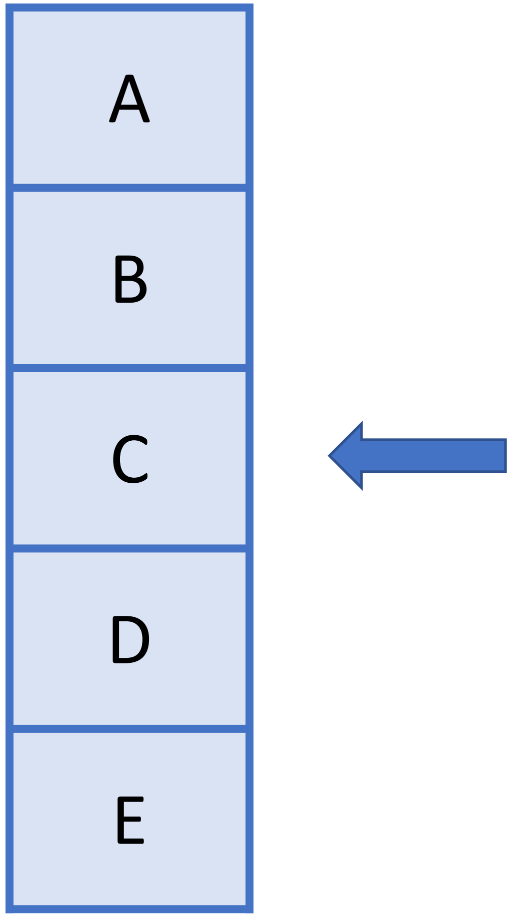
Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	A	1	1	1
4	X	X	0	X
3	X	X	0	X
2	C	1	1	1
1	B	0	1	0
0	A	0	0	0

Page Map

Clock Algorithm



Physical Pages

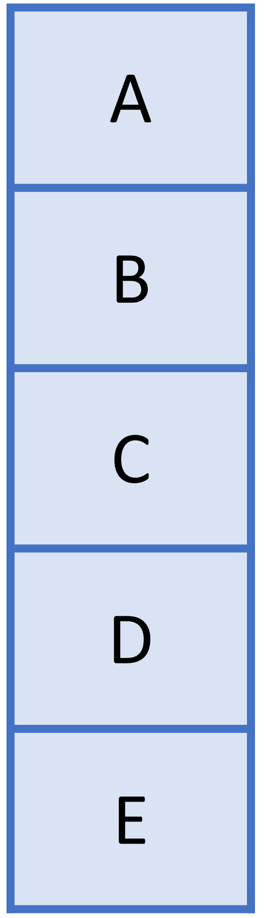
Was this page accessed recently (reference = 1)? If so, set reference = 0 and continue.

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	A	1	1	1
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	0	0

Page Map

Clock Algorithm



Physical Pages

Was this page accessed recently (reference = 1)? If not, this is the one we should remove.

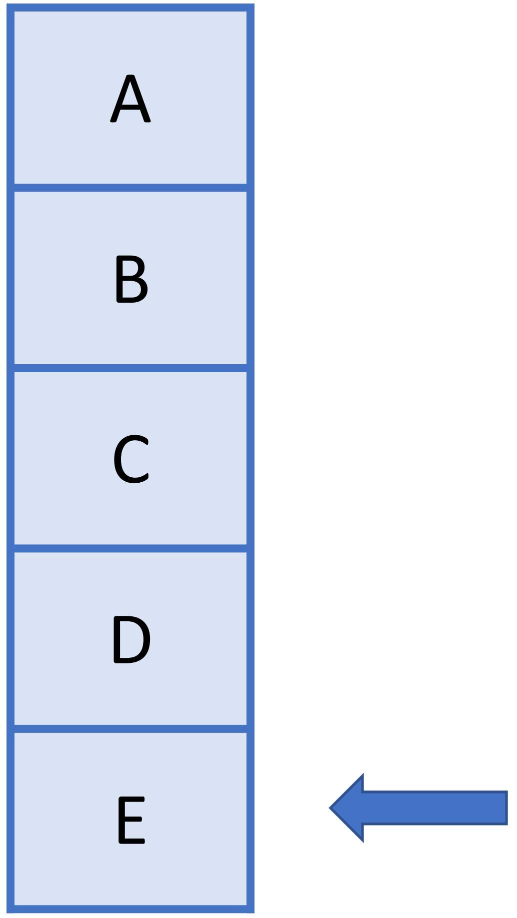
"This page hasn't been used since the last time I checked – let's remove it."

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	1	0
5	A	1	1	1
4	X	X	0	X
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	0	0

Page Map

Clock Algorithm



Physical Pages

Now the clock algorithm stops, and **we remember the position of the hand for next time it runs.**

“reference” bit

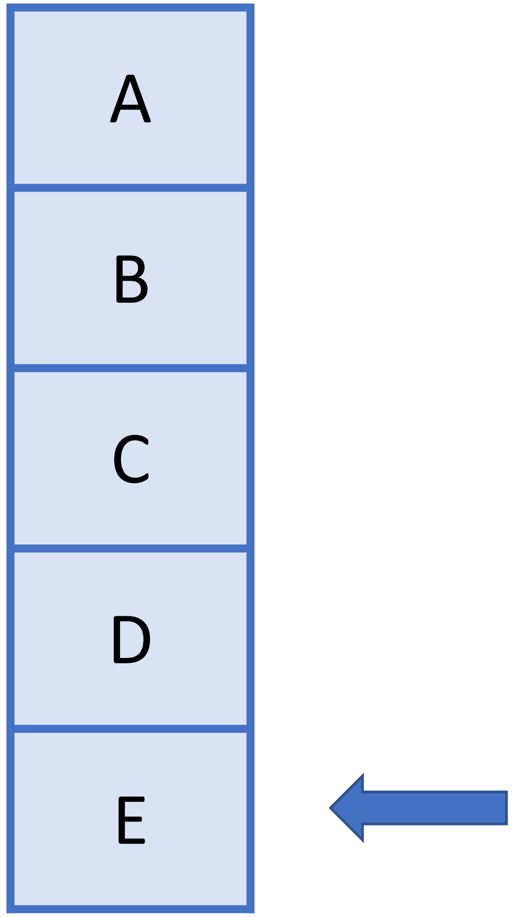
	Physical page #	WR?	PR?	R
7	E	1	1	0
6	D	1	0	0
5	A	1	1	1
4	D	1	1	1
3	X	X	0	X
2	C	1	1	0
1	B	0	1	0
0	A	0	0	0

Page Map

Clock Algorithm

- We add a *reference* bit: set whenever a page is read or written
- When physical memory is full and we need to choose a page to remove, run the clock algorithm.
- Clock hand "sweeps" over pages, rotating back to start if reaching the end.
- Every time the hand visits a page, we ask: "Has this page been referenced since the last time the clock hand swept over it?"
  - **If YES (reference = 1):** mark it as not referenced, and advance clock hand
  - **If NO (reference = 0):** choose it for removal, advance clock hand, stop clock algorithm
- The clock hand position is saved for the next time the algorithm runs
- "Second chance" algorithm

Clock Algorithm



Physical Pages

Some time has passed, pages were referenced, and we now need a new page. Which page will the clock algorithm choose to reuse this time?

Respond on PollEv:
pollev.com/cs111 or text CS111 to 22333 once to join.

"reference" bit

	Physical page #	WR?	PR?	R
7	E	1	1	1
6	D	1	0	0
5	A	1	1	1
4	D	1	1	1
3	X	X	0	X
2	C	1	1	0
1	B	0	1	1
0	A	0	0	0

Page Map

When poll is active, respond at **pollev.com/cs111**

Text **CS111** to **22333** once to join

Which page will be reused next?

- Page A
- Page B
- Page C
- Page D
- Page E

Page Replacement

How does page replacement work if there are multiple processes running?

- *Per-process replacement:* each process has separate pool of physical pages, and a page fault in a process can only replace one of its own pages. But how many physical pages should each process get?
- *Global replacement* (most common): all pages from all processes in single replacement pool. A page fault in one process can kick out a page in another process.

Plan For Today

- **Recap:** Demand Paging
- Page Replacement Policies
- The Clock Algorithm
- **Virtual Memory summary**

Virtual Memory

- Virtual memory is an example of "OS magic" – very powerful mechanism
- Virtualization: making one thing look like another – separation between appearance and reality
- OS can manage physical memory how it wants (e.g. swap to disk), invisible to user programs

Goals:

- **Multitasking** – allow multiple processes to be memory-resident at once
- **Transparency** – no process should need to know memory is shared. Each must run regardless of the number and/or locations of processes in memory.
- **Isolation** – processes must not be able to corrupt each other
- **Efficiency** (both of CPU and memory) – shouldn't be degraded badly by sharing

CS111 Topic 4: Virtual Memory

Virtual Memory - *How can one set of memory be shared among several processes? How can the operating system manage access to a limited amount of system memory?*

Why is answering this question important?

- We can understand one of the most “magical” responsibilities of OSes – making one set of memory appear as several!
- Exposes challenges of allowing multiple processes share memory while remaining isolated
- Allows us to understand exactly what happens when a program accesses a memory address

assign6: implement *paging/demand paging* system to translate addresses and load/store memory contents for programs as needed.

Recap

- Recap: Demand Paging
- Page Replacement Policies
- The Clock Algorithm
- Virtual Memory summary

Lecture 25 takeaway: There are many different policies to choose a page to kick out when memory is full. The clock algorithm is one approximation of LRU to pick an old page to remove.