use/access bit for each page set by hw

-> find less recently used page by looking at access bit
-> access bit is 1 => clear to 0
-> clock hand moves after each run
-> only cares about access bit

when evicting a page that’s dirty, needs to write it to swap

when evicting a page that’s clean (dirty bit == 0), no need to write to swap
### Second Chance / Enhanced Clock

<table>
<thead>
<tr>
<th>Access Bit</th>
<th>Dirty bit</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>evict the page</td>
</tr>
<tr>
<td>0</td>
<td>1 ⇒ 0</td>
<td>clear the dirty bit, track cleared dirty page, move on</td>
</tr>
<tr>
<td>1 ⇒ 0</td>
<td>0</td>
<td>clear the access bit, move on</td>
</tr>
<tr>
<td>1 ⇒ 0</td>
<td>1</td>
<td>clear the access bit, move on</td>
</tr>
</tbody>
</table>
Storage Devices

persistent, not byte addressable, block level access

Hard Disk / Spinning Disk
  -> sector: 512 bytes
  -> 10-20$TB

Solid State Drive
  -> block size 4 KB
  -> 3-5x cost of HDD

TB in capacity
Much cheaper than DRAM
disk read steps:

kernel sends the request to disk controller (ide.c)

disk finds the right platter & surface moves arm to track containing the sector

waits for desired sector to rotate under the head & reads the sector

data is transferred back to the host
Disk Performance

total time = seek time + rotation time + transfer time
                 (arm to track)   (sector under disk head)  (data transfer)

1). Seek time: 1-20 ms depending on how far to seek (let's say 10 ms on average)

2). Rotation time: specified as RPM, e.g., 7200 RPM = 120 RPS = 0.12 RPMs
   (assume it takes half a rotation for the desired sector to be in the right place)
   \[
   \frac{1}{0.12} = 8.3 \text{ ms per rotation} \\
   \frac{1}{2} \times 8.3 = 4 \text{ ms} \\n   \]
   average rotation time = half of full rotation = 4 ms

3). Transfer time: specified as disk bandwidth.

Example: read 1 sector, seek time 10 ms, 7200 RPM, bandwidth 120 MiB/s

\[
\text{total time} = 10 \text{ ms (seek)} + 4 \text{ ms (rotation)} + 0.004 \text{ ms (transfer time of 512 bytes)} \\
= 14.004 \text{ ms}
\]
Reading 10 sectors

\[ \text{10 consecutive reads: } 1 \text{ seek + 1 rotation + transfer time of 5120 bytes (sequential)} \]
\[ 10 + 4 + 0.04 \text{ ms} = 14.04 \text{ ms} \]

\[ \text{10 random reads/writes: } 10 \text{ seek + 10 rotation + transfer time of 5120 bytes} \]
\[ 14.004 \times 10 = 140.04 \text{ ms} \]

**Metrics:** IOPS (I/O operations per second)

\[ \frac{\# \text{ of I/O operations}}{\text{total time (s)}} \]

\[ \text{10 sequential reads: } \frac{10}{0.01404} = 712 \text{ IOPS} \]

\[ \text{10 random reads: } \frac{10}{0.1404} = 71.2 \text{ IOPS} \]
SSD.

- no moving parts
- parallel accesses
- units: page (2-4kB), erase (1-8MB)

Operations:
- read a page
- erase an entire block (set all bits to 1s)
- program a page (can only program an empty page)
To do in place update on SSD:

1. Copy existing pages elsewhere.
2. Random block with empty pages.
3. Erase entire block.
4. Update page with new content.

- Doing this causes a large \# of writes & repeated writes to updated pages.
- SSD pages have limited write cycles (10-100k).
- Wear leveling: spread writes across blocks/pages to reduce repeated writes to a single page.

**N.D.:**

$e\rightarrow s$

Data stored at *block* \( \rightarrow \) *page* \( \rightarrow \)

Logical address \( \downarrow \)

Actual block/page

Copy back

In place update completed!