Outline:
1. Memory hierarchy, overview
2. Buffer replacement policy review: LRU (Clock)
3. LRU implementation using linked list based queue? Clock implementation with circular array
4. Sequential flooding, how LRU fails, and how LRU-2 and MRU would work
5. Q&A for assignment 1

Memory hierarchy overview
1. CPU Cache, in the order of MBs, nanosecond access speed, volatile and expensive
2. Main memory, in the order of GBs, 100 nanosecond access speed, volatile, relatively cheaper
3. Secondary storage (Hard drive or SSD), in the order of TBs, millisecond access speed, non-volatile, cheap
4. Tertiary storage (tape drives), in the order of PBs, seconds or minutes, non-volatile, very cheap

Buffer replacement policy
- In order to provide efficient access to disk pages, every DBMS implements a large shared buffer pool in its own memory space
- Buffer pools is a set of frames; each frame has the same size as a block/page in database on disk
- Mapping between buffer frames and disk pages
- Why don’t use OS memory management unit? - database has specific knowledge about its data access pattern that OS doesn’t know about. So database can optimize the buffer replacement policy based on the data access pattern.
LRU Implementation

Buffer size \( \leq \text{size(Hashtable)} \)

1. Update to null
2. Page 2
3. Page 3
4. Page 4
5. Page 5

- The hash table is allocated when db starts.
- Bookkeeping
  - Actual buffer usage

```
pageID  frameID
1       1
2.      2
3.      3
4.      4
5.      1
```
Clock implementation

Init array.

<table>
<thead>
<tr>
<th>Page ID</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc}
ID:1 & ID:2 & ID:3 \\
PD:1 & PD:2 & PD:3 \\
REF:1 & REF:2 & REF:3 \\
\end{array}
\]

if CURRENT = 4
CURRENT = 0

current = 0

t he hash table is allocated when db starts.