README

Frank (1009990734)
Wu Hung Mao (1008134207)

1 NOV 2025

FIFO vs LRU

FIFO hits count > LRU hits count

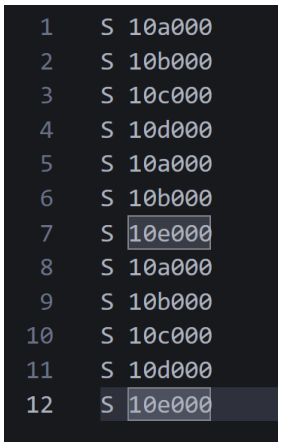

Figure 1: Sequence of page accesses where FIFO hits more than LRU

For the above traces, assume memory size is 3 frames. FIFO has a hit count of 3 while LRU has a hit count of 2, hit rate and miss rate are present in Figure 2.

```
wuhungma@dh2026pc03:~/CSC369/group_3376/A3$ ./sim -f ./traceprogs/FIFO_better_than_LRU.ref -m 3 -s 10000 -a fifo

Hit count: 3
Miss count: 9
Clean evictions: 0
Dirty evictions: 6
Total references : 12
Hit rate: 25.0000
Miss rate: 75.0000
wuhungma@dh2026pc03:~/CSC369/group_3376/A3$ ./sim -f ./traceprogs/FIFO_better_than_LRU.ref -m 3 -s 10000 -a lru

Hit count: 2
Miss count: 10
Clean evictions: 0
Dirty evictions: 7
Total references : 12
Hit rate: 16.6667
Miss rate: 83.3333
```

Figure 2: Hit rate, miss rate, hit count, and miss count for page access where FIFO is better than LRU. FIFO results are listed first, with LRU results listed second.

The reason why FIFO is performing better is because in this specific pattern, when 10c000 arrives, FIFO keep the page 10e000 that will be reused later. LRU evicts page 10e000 because it believes the least recently access page 10e000 will not be reused, but it got reused. So, if there is a memory access pattern such that LRU has to evict least recently used page which is accessed right after, then FIFO might have a higher hit count. However, it requires knowing the memory size, so I don't think it happens a lot in real life.

LRU hits count > FIFO hits count

```
1   S 10a000
2   S 10b000
3   S 10c000
4   S 10a000
5   S 10d000
6   S 10b000
7   S 10a000
8   S 10b000
9   S 10c000
```

Figure 3: Sequence of page accesses where LRU hits more than FIFO

The hit count and miss count and their rates are shown in Figure 4.

```
wuhungma@dh2026pc03:~/CSC369/group_3376/A3$ ./sim -f ./traceprogs/LRU_better_than_FIFO.ref -m 3 -s 10000 -a lru

Hit count: 3
Miss count: 6
Clean evictions: 0
Dirty evictions: 3
Total references : 9
Hit rate: 33.3333
Miss rate: 66.6667
wuhungma@dh2026pc03:~/CSC369/group_3376/A3$ ./sim -f ./traceprogs/LRU_better_than_FIFO.ref -m 3 -s 10000 -a fifo

Hit count: 2
Miss count: 7
Clean evictions: 0
Dirty evictions: 4
Total references : 9
Hit rate: 22.2222
Miss rate: 77.7778
```

Figure 4: Hit rate, miss rate, hit count, and miss count for page access where LRU is better than FIFO. LRU results are listed first, with FIFO results listed second.

When LRU and FIFO encounters page 10d000, LRU keeps recently used page 10a000, while FIFO evict the that page. This causes LRU to hit one additional time because 10a000 is accessed again soon after.
The memory pattern which LRU performs better is when we access same page a lot of times, which allows LRU to exploit temporal locality. FIFO evicts page regardless how frequently the page is accessed, hence, it could be evicting a page that is frequently accessed but have to be evicted just because it is old enough in the RAM.

LRU vs CLOCK

Assume for both replacement policies that the memory size is 3.

LRU performing better than CLOCK:

Figure 5 shows the results of using LRU and CLOCK for the following page accesses:

S 10a000
S 10b000
S 10c000
S 10d000
S 10b000
S 10e000
S 10c000
S 10b000

```
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_better_than_CLOCK.ref -m 3 -s 10000 -a lru

Hit count: 2
Miss count: 6
Clean evictions: 0
Dirty evictions: 3
Total references : 8
Hit rate: 25.0000
Miss rate: 75.0000
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_better_than_CLOCK.ref -m 3 -s 10000 -a cloc

Hit count: 1
Miss count: 7
Clean evictions: 0
Dirty evictions: 4
Total references : 8
Hit rate: 12.5000
Miss rate: 87.5000
```

Figure 5: Hit rate, miss rate, hit count, and miss count for page access where LRU is better than CLOCK. LRU results are listed first, with CLOCK results listed second.

The first three accesses fill up memory. The access to 10d000 clears the references bits to the other pages in CLOCK. With the next access to 10b000, the LRU and CLOCK begin to differ. LRU sees 10b000 as the most recent access, whereas CLOCK would only set its reference bit to 1. CLOCK would not know if 10b000 is the absolute most-recently used page, only that it should not be the least-recently used page. Hence when an access is made to 10e000 and 10c000, LRU would evict the true least-recently used page, while CLOCK would evict an approximation of the least-recently used page (which happens to be 10b000). After these evictions, the memory of both policies are now different, so an access to 10b000 would hit in LRU but not in CLOCK.

CLOCK performing better than LRU:

Figure 6 shows the results of using LRU and CLOCK for the following page accesses. Notice how only the last access was changed:

S 10a000
S 10b000
S 10c000
S 10d000
S 10b000
S 10e000
S 10c000
S 10d000

```
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_worse_than_CLOCK.ref -m 3 -s 10000 -a lru

Hit count: 1
Miss count: 7
Clean evictions: 0
Dirty evictions: 4
Total references : 8
Hit rate: 12.5000
Miss rate: 87.5000
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_worse_than_CLOCK.ref -m 3 -s 10000 -a clock

Hit count: 2
Miss count: 6
Clean evictions: 0
Dirty evictions: 3
Total references : 8
Hit rate: 25.0000
Miss rate: 75.0000
```

Figure 6: Hit rate, miss rate, hit count, and miss count for page access where CLOCK is better than LRU. LRU results are listed first, with CLOCK results listed second.

The page accesses are similar to the earlier case where LRU performed better, making CLOCK miscalculate the true least-recently used page. However, in this case CLOCK performed better than LRU because the final memory access $10d000$ was to the page that was not used recently enough for LRU to keep, while still being used recently enough for CLOCK to contain. LRU has a tighter constraint on the true recent accesses. In comparison, CLOCK does not hold the exact last 3 (our memory size) accessed pages, but is close to it.

LRU vs ARC

Assume for both replacement policies that the memory size is 3. Additionally, assume that ARC has an initial p value of 0.

LRU performing better than ARC:

Figure 7 shows the results of using LRU and ARC for the following page accesses:

S $10a000$
S $10b000$
S $10c000$
S $10a000$
S $10b000$
S $10d000$
S $10e000$
S $10f000$
S $10d000$
S $10e000$

```
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_better_than_ARC.ref -m 3 -s 10000 -a lru

Hit count: 4
Miss count: 6
Clean evictions: 0
Dirty evictions: 3
Total references : 10
Hit rate: 40.0000
Miss rate: 60.0000
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_better_than_ARC.ref -m 3 -s 10000 -a arc

Hit count: 2
Miss count: 8
Clean evictions: 0
Dirty evictions: 5
Total references : 10
Hit rate: 20.0000
Miss rate: 80.0000
```

Figure 7: Hit rate, miss rate, hit count, and miss count for page access where LRU is better than ARC. LRU results are listed first, with ARC results listed second.

LRU performs better than ARC because on the second access to pages $10a000$ and $10b000$, ARC's adaptations overestimate their frequency, moving them into its $T2$ list. For the next accesses to $10d000$, $10e000$, and $10f000$, LRU completely removes the $10a000$ and $10b000$ pages from memory. In contrast, since the value of ARC's p is initially 0, ARC favours their frequency and instead chooses to evict the less frequent $10d000$ and $10e000$. Hence, when accesses to $10d000$ and $10e000$ are made again, LRU hits both while ARC misses. Arc finds the missed pages in its $B1$ ghost list, causing ARC to adapt in favour of recency over frequency. ARC would therefore begin to behave more like LRU until certain sequences of page accesses force ARC to adapt to frequency again.

ARC performing better than LRU:

Figure 8 shows the results of using LRU and ARC for the following page accesses:

S $10a000$
S $10b000$
S $10c000$
S $1fff000$
S $10d000$
S $10e000$
S $10f000$
S $1fff000$

```
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_worse_than_ARC.ref -m 3 -s 10000 -a lru

Hit count: 1
Miss count: 12
Clean evictions: 0
Dirty evictions: 9
Total references : 13
Hit rate: 7.6923
Miss rate: 92.3077
bifrank@dh2026pc16:~/csc369_group/A3/testing$ ./sim -f ./traceprogs/LRU_worse_than_ARC.ref -m 3 -s 10000 -a arc

Hit count: 3
Miss count: 10
Clean evictions: 0
Dirty evictions: 7
Total references : 13
Hit rate: 23.0769
Miss rate: 76.9231
```

Figure 8: Hit rate, miss rate, hit count, and miss count for page access where ARC is better than LRU. LRU results are listed first, with ARC results listed second.

ARC performs better than LRU because the accesses to $1fff000$ are frequent but not recent. LRU favours pages with more recent accesses, whereas ARC adapts to the frequency of $1fff000$. Because of this, ARC will hit on the next access to $1fff000$, whereas LRU had evicted it in favour of more recent pages.

Recommendations

FIFO

Assuming the developer knows the size of physical memory and size of each page, he should avoid writing the program which each memory access jumps across different page. For FIFO, the developer needs to avoid having a frequently accessed page getting evicted often(relying on temporal locality) because many new pages come into cache. Also, the developer should keep belady's anomaly in mind, which means that the program might not have more hit just by simply increasing memory size. FIFO should be used when the pages are accessed near-sequentially.

LRU

LRU should be used when pages are accessed near consecutively, with strong temporal locality. For example, a developer could implement matrix multiplication such that values that are close to each other in memory can be multiplied first, instead of relying on a triple for loop for multiplication. This way LRU won't be forced to evict a page that will be needed in the future loop(iteration).

CLOCK

CLOCK is an approximation of LRU, keeping pages that accessed were recently but may not contain the top most recent pages. This is beneficial to CLOCK if the same pages are near-consecutively accessed with a short run of other page accesses in between, where the run length is slightly larger than memory size. In this case, the accuracy of LRU hinders itself because that page was not recent enough for LRU to keep, but is recent enough for CLOCK to coincidentally contain.

However, the main benefits of CLOCK are its cheaper operations and less overhead compared to LRU. CLOCK does not manage a large data structure to track and compare the recency between pages, therefore saving space and making it a better choice for large memory sizes than LRU.

ARC

ARC should be used when pages are accessed with long spans of recency, frequency, or both. This is because ARC can adapt to the nature of page accesses, making it perform nearly as well as LRU or CLOCK for pages that are accessed recently, while also being able to perform well if pages are suddenly accessed frequently but not recently. However, ARC should not be used when page accesses rapidly change between recency and frequency. This is because ARC will take too long to adapt between policies, not only implementing policies when it is too late, but possibly replacing its previous, more-effective policy for the next sequence.