CS 1550

Project 5 – Virtual Memory Simulator

Trace File

I – Instruction fetch from RAM

L – Load data

S – Store data

M – 1st Load then Store

```
I  0023C790,2
I  0023C792,5
 S BE80199C,4
I  0025242B,3
 L BE801950,4
I  0023D476,7
 M 0025747C,1
I  0023DC20,2
 L 00254962,1
 L BE801FB3,1
I  00252305,1
 L 00254AEB,1
 S 00257998,1
```

Example – Assumptions

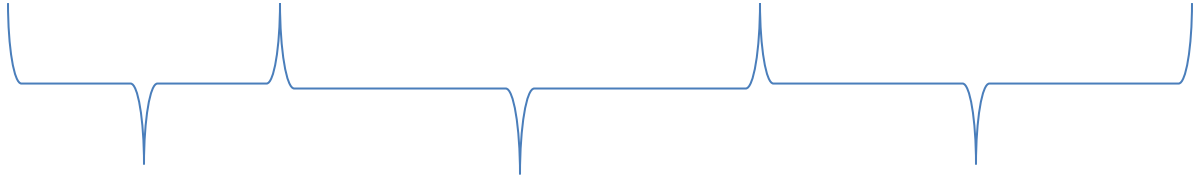
- Number of frames = 3
- Page Size = 4 KB
- Address = 32 bits
- Number of entries in Root of the page table = 256 = 2^8

IMPORTANT!!

These are my assumptions for the examples in these slides. For your implementation, please follow the specifications given in the project description

Example – Address Breakdown

32-bit Address = 00101010 101010101 010101010 010101010



8 bits

Index to Root Page Table

This is because root has 256 = 2^8 entries

12 bits

Index to leaf page table

So, each leaf page table has 2^11 entries.

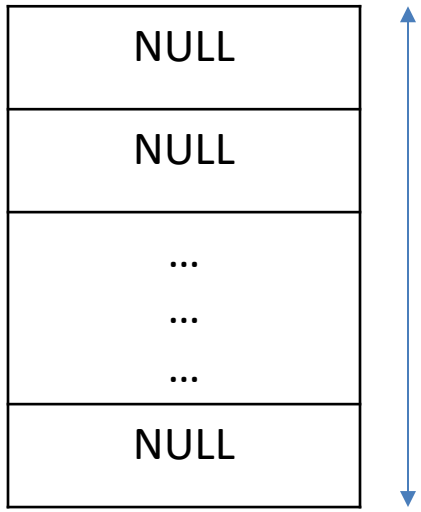
12 bits

Byte Offset

This is because Page Size = 4 KB = 2 ^ 12 Bytes

For this assumption, Consider the following example

Root Page Table

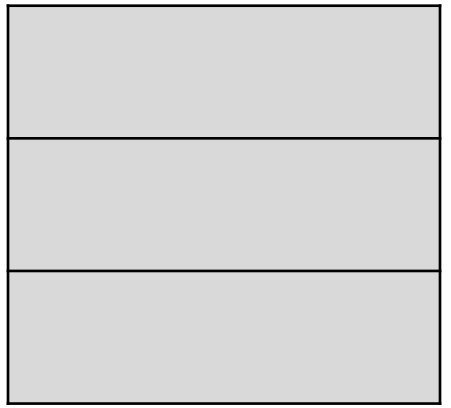


256 entries

Trace File

```
I 0000abcd, 1
  M 00001234, 1
  S 00002345, 1
I 0000abcd, 1
  L 00003456, 1
```

Physical Memory



3 frames

For this assumption, Consider the following example

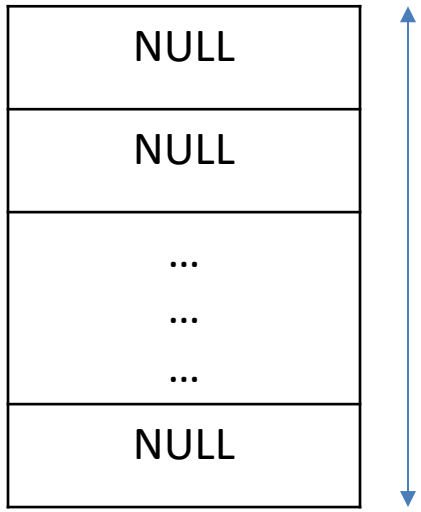
Root PT index = 00

Leaf PT Index = 000

Trace File

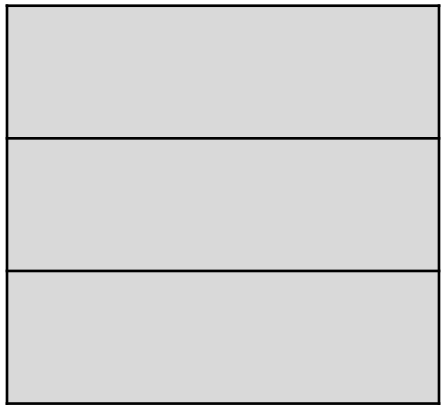
I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

Root Page Table



256 entries

Physical Memory



3 frames

For this assumption, Consider the following example

Root PT index = 00

Index in Leaf PT = 000

Leaf PT 0

V	D	R
		… … …
		.

2^12 entries

Root PT

00

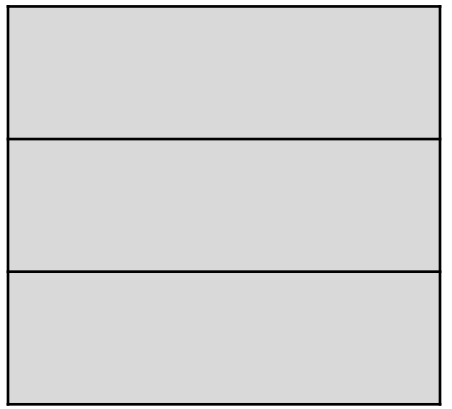
NULL
… … …
NULL

000

Trace File

I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

Physical Memory



PAGE FAULT!!

What will be the initial values of valid, dirty and reference bits in Leaf PT 0??

For this assumption, Consider the following example

Root PT index = 00

Index in Leaf PT = 000

Root PT

00

NULL
… … …
NULL

000

Leaf PT 0

V	D	R
1	0	1
		… … …
		.

Trace File

I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

Physical Memory

00000

For this assumption, Consider the following example

Root PT index = 00

Index in Leaf PT = 001

Root PT

00

NULL
… … …
NULL

Leaf PT 0

001

V	D	R
1	0	1
		… … …
		.

Trace File

I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

Physical Memory

00000

PAGE FAULT!!

For this assumption, Consider the following example

Root PT index = 00

Index in Leaf PT = 001

Root PT

NULL
… … …
NULL

00

Leaf PT 0

V	D	R
1	0	1
1	0	1
		… … …
		.

001

M = 1st Load

Trace File

I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

Physical Memory

00000

PAGE FAULT!!

For this assumption, Consider the following example

Root PT index = 00

Index in Leaf PT = 001

Root PT

NULL
… … …
NULL

00

Leaf PT 0

V	D	R
1	0	1
1	0	1
		… … …
		.

001

Trace File

I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

Physical Memory

00000
00001

M = 1st Load

PAGE FAULT!!

For this assumption, Consider the following example

Root PT index = 00

Index in Leaf PT = 001

Root PT

00

NULL
… … …
NULL

Leaf PT 0

001

V	D	R
1	0	1
1	1	1
		… … …
		.

Dirty Bit Set because it is a store

Trace File

I 00000bcd, 1
M 00001234, 1
S 00002345, 1
I 00000bcd, 1
L 00003456, 1

M = Then Store

Physical Memory

00000
00001

PAGE HIT!!

That’s how you will modify the page table and the physical memory

Page Replacement Algorithms

Example – Assumptions

- Number of physical frames = 3
- Page Size = 4 KB
- Address = 32 bits
- Number of entries in Root of the page table = 256 = 2^8

IMPORTANT!!

These are my assumptions for the examples in these slides. For your implementation, please follow the specifications given in the project description

Project - Virtual Memory Simulator

- Number of physical frames = 3
- Page Size = 4 KB = 2^12 B
- Address = 32 bits
- Number of entries in Root of the page table = 256 = 2^8

Physical Memory

```
S    3856bbe0    0
L    190afc20    0
L    15216f00    0
L    190a7c20    0
L    190a7c28    0
L    190a7c28    0
L    190aff38    0
```

Shift Operations

- Left shift (`x << n`)
  - Fill with `0`s on right
- Right shift (`x >> n`)
  - Logical shift
    - Unsigned values
    - Fill with 0s on left
  - Arithmetic shift
    - Signed values
    - Replicate most significant bit on left

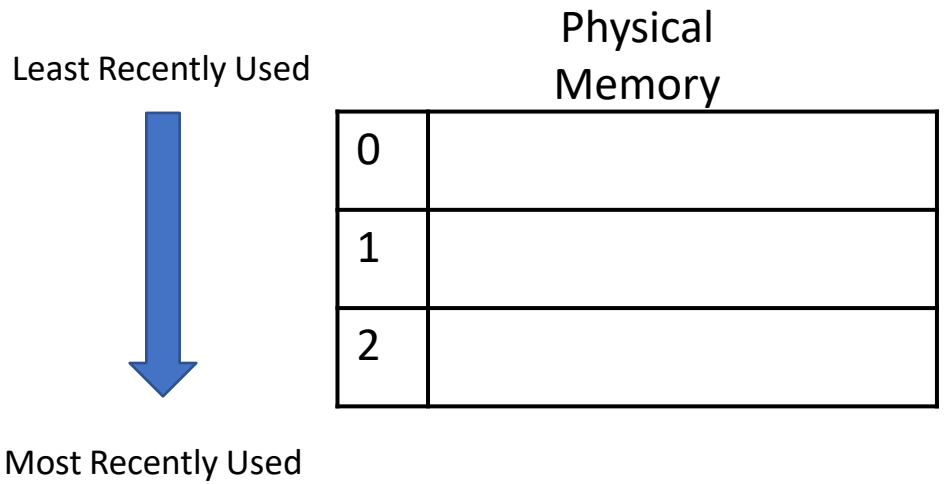
	x	0010 0010
	x<<3	0001 0000
logical:	x>>2	0000 1000
arithmetic:	x>>2	0000 1000

	x	1010 0010
	x<<3	0001 0000
logical:	x>>2	0010 1000
arithmetic:	x>>2	1110 1000

- Notes:
  - Shifts by `n<0` or `n≥w` (`w` is bit width of `x`) are *undefined*
  - **C:** behavior of `>>` is determined by compiler
    - In gcc / Clang, depends on data type of `x` (signed/unsigned)
  - **Java:** logical shift is `>>>` and arithmetic shift is `>>`

LRU

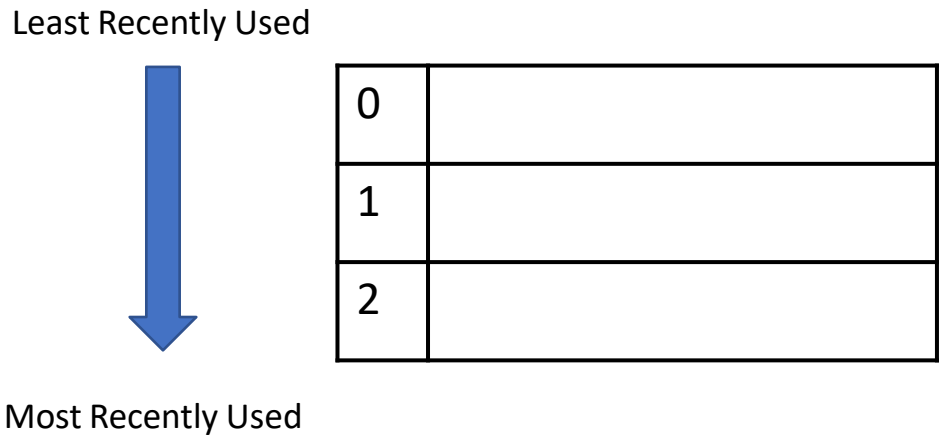
- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)

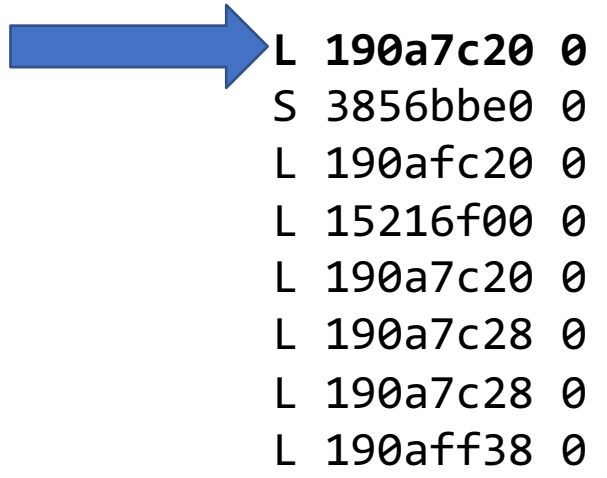



```
L 190a7c20 0
S 3856bbe0 0
L 190afc20 0
L 15216f00 0
L 190a7c20 0
L 190a7c28 0
L 190a7c28 0
L 190aff38 0
```

LRU

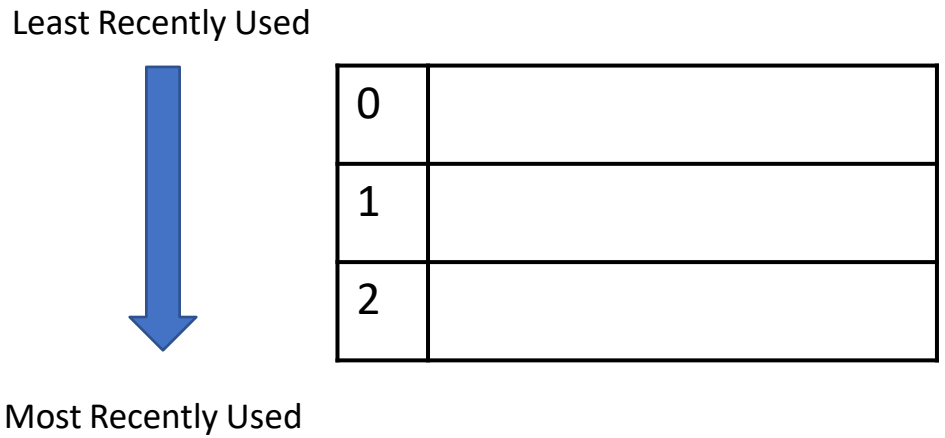
- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)





LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)



Pagefault since it is not in the physical memory

```
L 190a7c20 0
S 3856bbe0 0
L 190afc20 0
L 15216f00 0
L 190a7c20 0
L 190a7c28 0
L 190a7c28 0
L 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)

Pagefault since it is not in the physical memory

Least Recently used

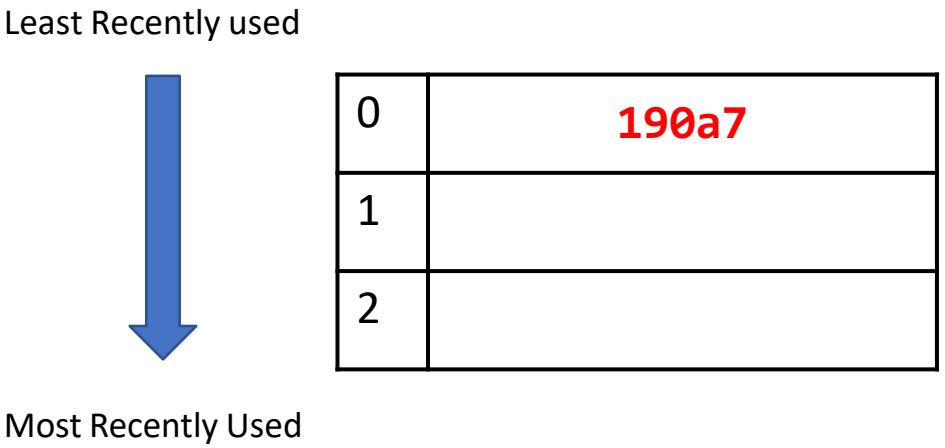
0	**190a7**
1	
2	

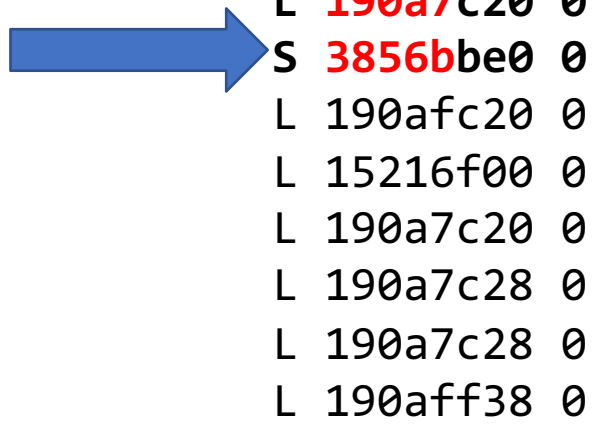
Most Recently Used

```
L 190a7c20 0
S 3856bbe0 0
L 190afc20 0
L 15216f00 0
L 190a7c20 0
L 190a7c28 0
L 190a7c28 0
L 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)





LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)

Pagefault since it is not in the physical memory

Least Recently used

0	**190a7**
1	
2	

Most Recently Used

```
L 190a7c20 0
S 3856bbe0 0
L 190afc20 0
L 15216f00 0
L 190a7c20 0
L 190a7c28 0
L 190a7c28 0
L 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)

Pagefault since it is not in the page table

Least Recently used

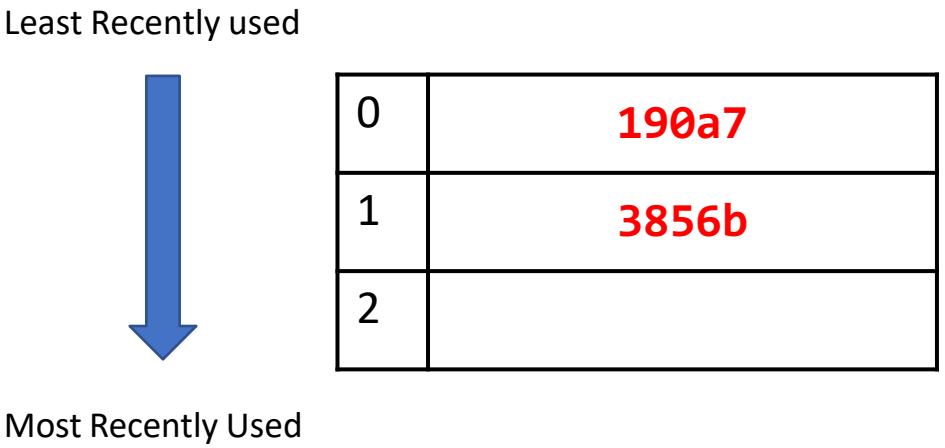
0	**190a7**
1	**3856b**
2	

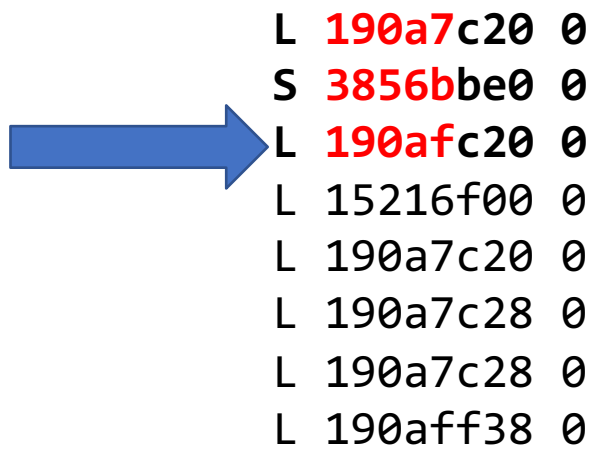
Most Recently Used

```
L 190a7c20 0
S 3856bbe0 0
L 190afc20 0
L 15216f00 0
L 190a7c20 0
L 190a7c28 0
L 190a7c28 0
L 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)





LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume Least Recently Used(LRU)

Pagefault since it is not in the physical memory

Least Recently used

0	**190a7**
1	**3856b**
2	**190af**

Most Recently Used

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

l = load
s = store

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume Least Recently Used(LRU)

Least Recently used

0	**190a7**
1	**3856b**
2	**190af**

Most Recently Used

We need to evict someone!!

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume **Least Recently Used(LRU)**

Least Recently used

0	190a7
1	3856b
2	190af

Most Recently Used

We need to evict someone!!

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume **Least Recently Used(LRU)**

Least Recently used

0	190a7
1	3856b
2	190af

Most Recently Used

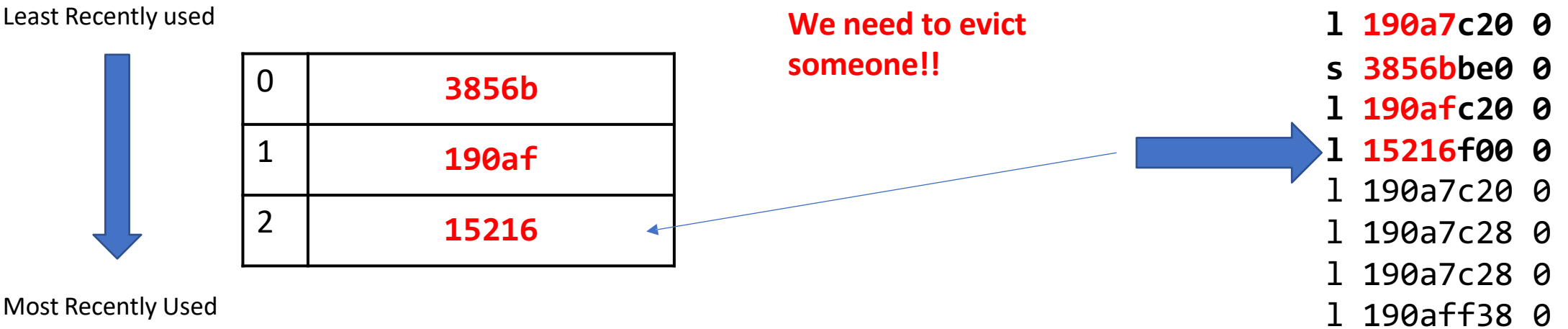
We need to evict someone!!

Evict the LRU page

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume **Least Recently Used(LRU)**



Pagefault again

What type of data structure can you use for your physical memory?

LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Least Recently used

0	190af
1	15216
2	190a7

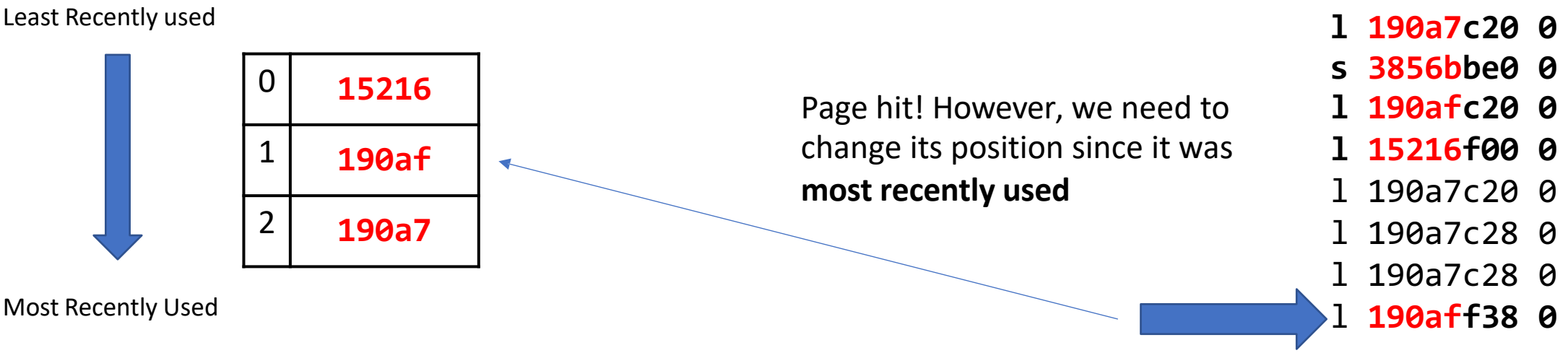
Most Recently Used

Assume we skip to page 190af

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
~~l 190a7c20 0~~
~~l 190a7c28 0~~
~~l 190a7c28 0~~
l 190aff38 0
```

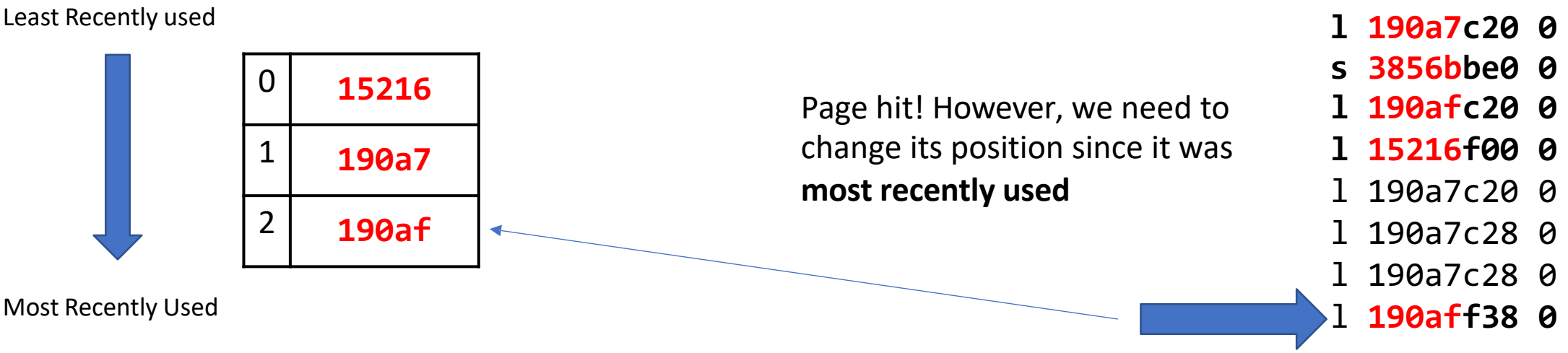
LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames



LRU

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames



Dirty Bit

- **Dirty bit**
  - Stores will set the target's dirty bit, while Loads won't change the dirty bit.
  - Upon eviction, dirty bit decides whether writing page to disk is necessary.

0	190a7	0
1	3856b	1
2		

Set dirty bit for a store

If we evict 3856b whose dirty bit is set, we need to write (save) the updated page data to disk.

If we evict 190a7 whose dirty bit is not set, there's no need to write disk because no data changes.

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

OPT

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames
  - Assume OPT

To find the optimal victim, you'll need to know all future memory accesses

0	
1	
2	

```
L 190a7c20 0
S 3856bbe0 0
L 190afc20 0
L 15216f00 0
L 190a7c20 0
L 190a7c28 0
L 190a7c28 0
L 190aff38 0
```

OPT

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume OPT

0	**190a7**
1	**3856b**
2	**190af**

We need to evict someone!!

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

OPT

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume OPT

0	**190a7**
1	**3856b**
2	**190af**

We need to evict someone!!

Page 190a7 will be used later!

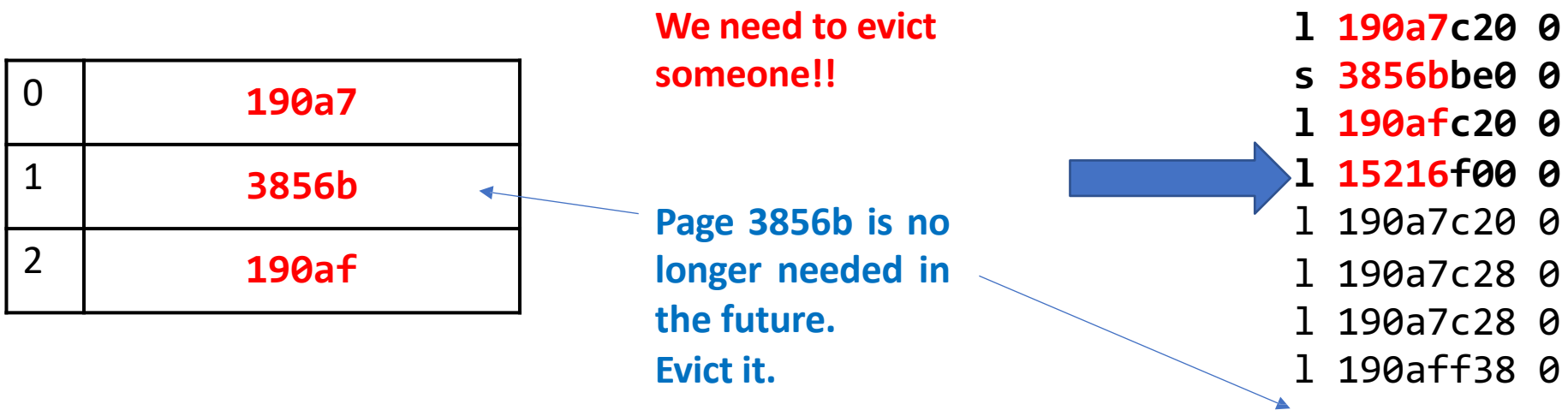
```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

OPT

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume OPT



OPT

- Assume 12KB physical memory, each page is 4KB, so there're 3 page frames

Pagefault again

  - Assume OPT

0	**190a7**
1	**3856b**
2	**190af**

What if there is a tie: multiple pages no longer needed in the future?

--Break the tie by evicting any arbitrary of those pages

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

OPT

- Implementing OPT in a naïve fashion will lead to unacceptable performance caused by searching across trace
- It should not take more than **5 minutes** to run your program

OPT

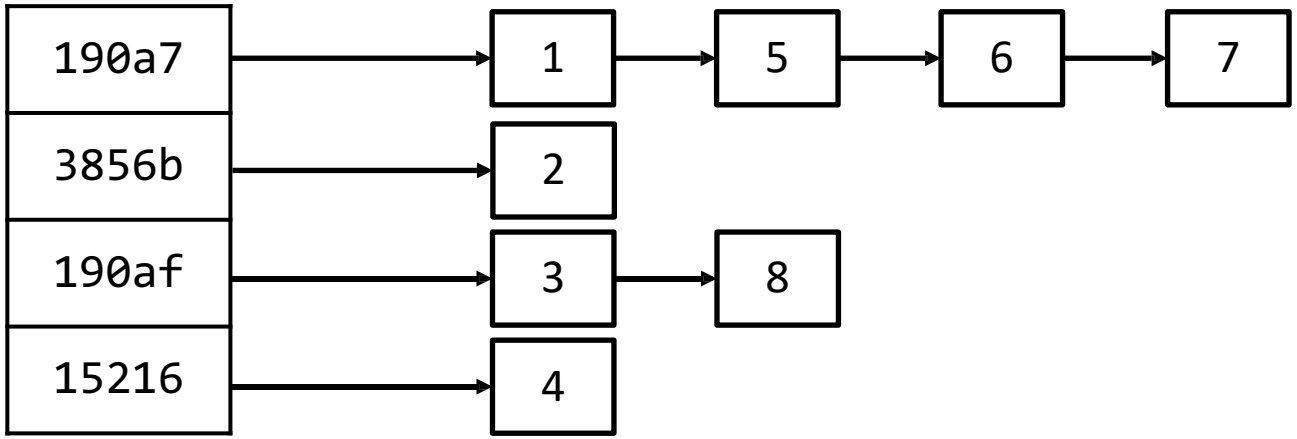
- Implementing OPT in a naïve fashion will lead to unacceptable performance caused by searching across trace
- It should not take more than **5 minutes** to run your program
- **How to be more efficient on searching**
  - E.g., use Hash table with Page Number as Key. If you're using C, here's one example of hash based on the Horner's method[1]

```
int hash(char  *v,  int M)
{
        int h,  a  =  117;
        for (h  =  0;  *v  !=  '\0';  v++)
        h =  (a*h  +  *v) % M;
        return h;
}
```

[1] **Horner's method:** https://www.cs.iupui.edu/~cs240/summer10/slides/t99Hashing.pdf

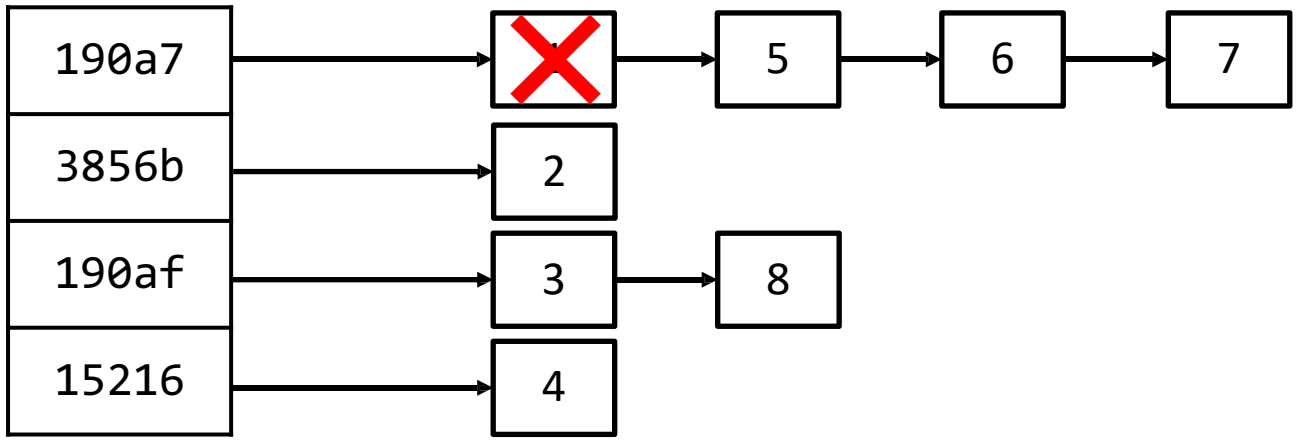
OPT

- Hash Example




```
1) l 190a7c20 0
2) s 3856bbe0 0
3) l 190afc20 0
4) l 15216f00 0
5) l 190a7c20 0
6) l 190a7c28 0
7) l 190a7c28 0
8) l 190aff38 0
```

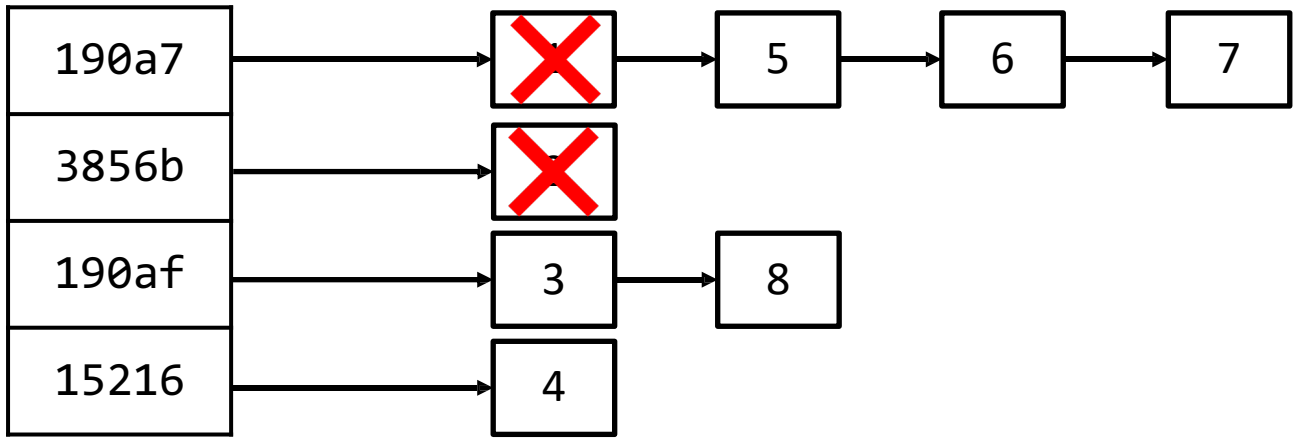
OPT



0	**190a7**
1	
2	

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

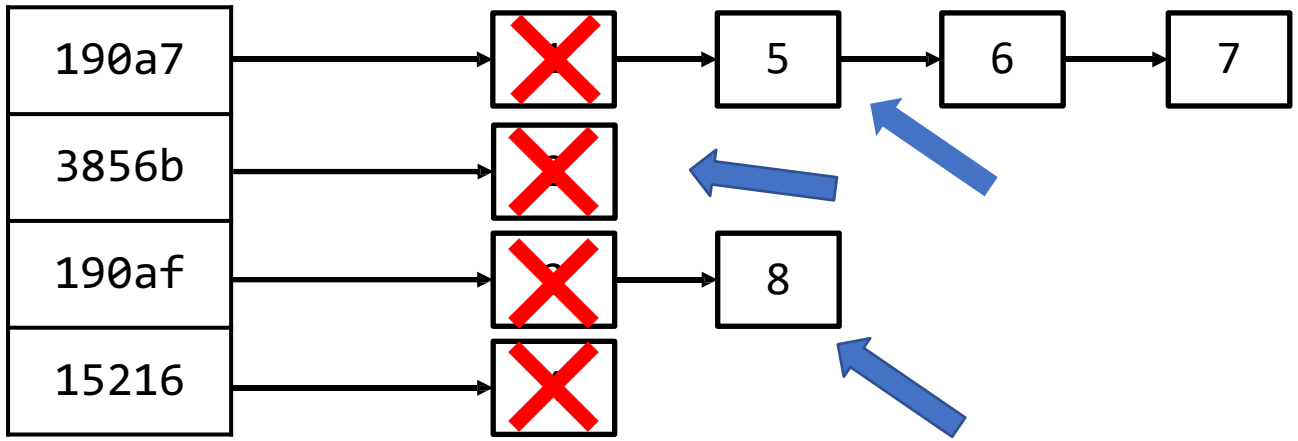
OPT



0	**190a7**
1	**3856b**
2	

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

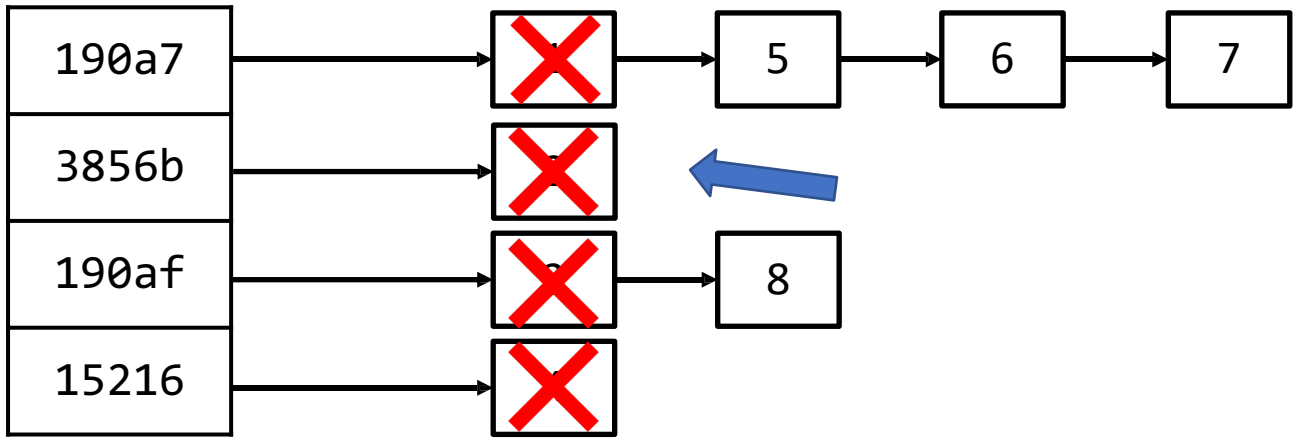
OPT



0	**190a7**
1	**190af**
2	**15216**

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

OPT



0	**190a7**
1	**190af**
2	**15216**

```
l 190a7c20 0
s 3856bbe0 0
l 190afc20 0
l 15216f00 0
l 190a7c20 0
l 190a7c28 0
l 190a7c28 0
l 190aff38 0
```

Virtual Memory Simulator

- No need to use qemu
- You will write the simulator from scratch with Java, C/C++, Java, or Python
- Read from memory traces text files
- Statistics: count the number of events (pagefaults, page evictions etc.)

Statistics

```
Algorithm: %s
Number of frames:       %d
Total memory accesses:  %d
Total page faults:      %d
Total writes to disk:   %d
Number of page table leaves:    %d
Total size of page table:       %d
```

Write Up

For each of your three algorithms, describe in a document the resulting page fault statistics for 8, 16, 32, and 64 frames. Use this information to determine which algorithm you think might be most appropriate for use in an actual operating system. Use OPT as the baseline for your comparisons.