R

بسم الله الرحمن الرحیم

سیستم عامل

جلسه شانزدهم – الگوریتم‌های جایگزینی صفحه (۲)

جلسه‌ی گذشته

Page Replacement

- Assume a normal page table (e.g., BLITZ)
- User-program is executing
- A *PageInvalidFault* occurs!
  - *The page needed is not in memory*
- Select some frame and remove the page in it
  - *If it has been modified, it must be written back to disk*
    - the "dirty" bit in its page table entry tells us if this is necessary
- Figure out which page was needed from the faulting addr
- Read the needed page into this frame
- Restart the interrupted process by retrying the same instruction

Page Replacement Algorithms

- Which frame to replace?
- *Algorithms?*
  - *ورودی: لیست صفحه‌هایی که در حافظه هستند و درخواست‌های مربوط به گرفتن صفحه*
  - *خروجی: کدام صفحه را از حافظه خارج کنیم*
  - *هدف: کاهش تعداد page faultها*

The optimal page replacement algorithm

- **Idea:** Given all the data, how to find the optimal page replacement?

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a										
	1	b										
	2	c										
	3	d										
Page faults												

LFD

- Replace the page that will not be needed for the longest
- Example:

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a						
	1	b	b	b	b	b						
	2	c	c	c	c	c						
	3	d	d	d	d	d						
Page faults							X					

LFD

- Replace the page that will not be needed for the longest
- Example:

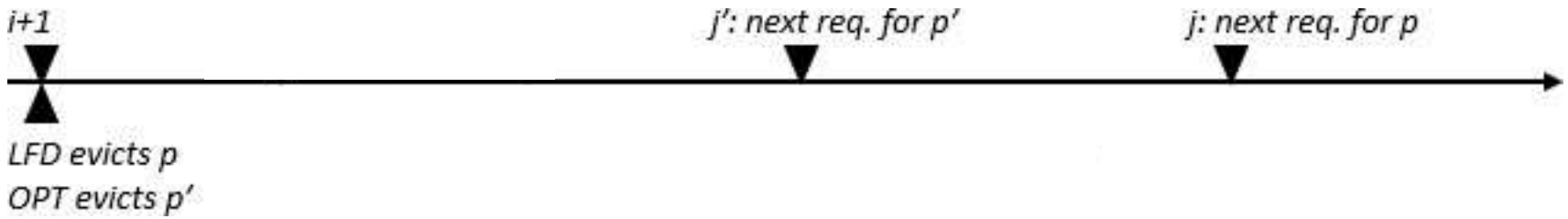
Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a	
	1	b	b	b	b	b	b	b	b	b	b	
	2	c	c	c	c	c	c	c	c	c	c	
	3	d	d	d	d	d	e	e	e	e	e	
Page faults							X				X	

Is LFD Optimal?

- **OPT:** Optimum with longest prefix equal to LFD
  - *First non-equal poistion: i+1*
- Case 1) i+1 is not a page fault.
  - *Is it possible?*
- Case 2) i+1 is page fault
  - *LFD evicts p*
  - *OPT evicts p'*

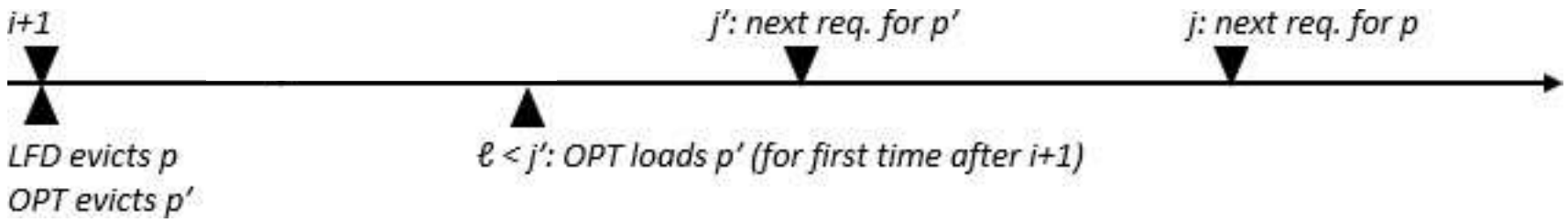
Is LFD Optimal?

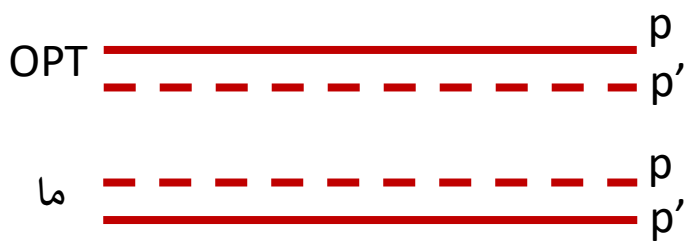
- **OPT:** Optimum with longest prefix equal to LFD
  - *First non-equal poistion: i+1*
- Case 2) i+1 is page fault
  - *LFD evicts p*
  - *OPT evicts p'*



Is LFD Optimal?

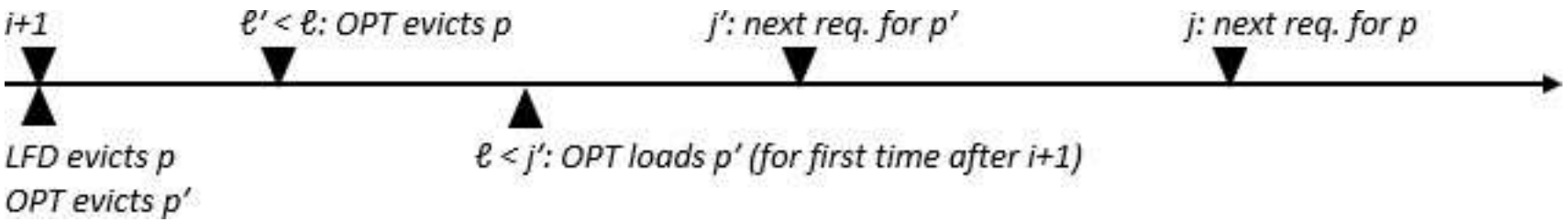
- Proof (by contradiction):
  - **OPT**: Optimum with longest prefix equal to LFD Case
  - 2) i+1 is page fault
    - Case 2-A) OPT keeps p until l
    - Case 2-B) OPT evicts p at l' < l

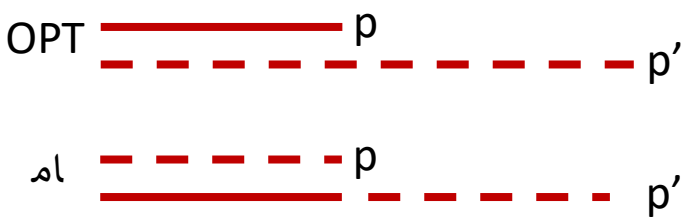




Is LFD Optimal?

- Proof (by contradiction):
  - **OPT**: Optimum with longest prefix equal to LFD Case
  - 2) i+1 is page fault
    - Case 2-A) OPT keeps p until l
    - Case 2-B) OPT evicts p at l' < l





LFD = OPT

Optimal Page Replacement

- Idea:
  - *Select the page that will not be needed for the longest time*
- Problem:
  - *Can't know the future of a program*
  - *Can't know when a given page will be needed next*
  - *The optimal algorithm is unrealizable*

Optimal Page Replacement

- However:
  - *We can use it as a control case for simulation studies*
    - Run the program once
    - Generate a log of all memory references
    - Do we need all of them?
    - Use the log to simulate various page replacement algorithms
    - Can compare others to “optimal” algorithm

جلسه‌ی جدید

تعداد PAGE FAULTها

چه پارامترهایی روی تعداد پیج‌فالت‌ها موثر هستند؟

- الگوریتم جایگزینی صفحه
- اندازه‌ی تعداد frameهای قابل قرارگیری در حافظه
  - *این ممکنه global برای کل پردازه‌ها مشترک باشه*
  - *ممکنه local به ازای هر پردازه باشه.*
- ؟؟

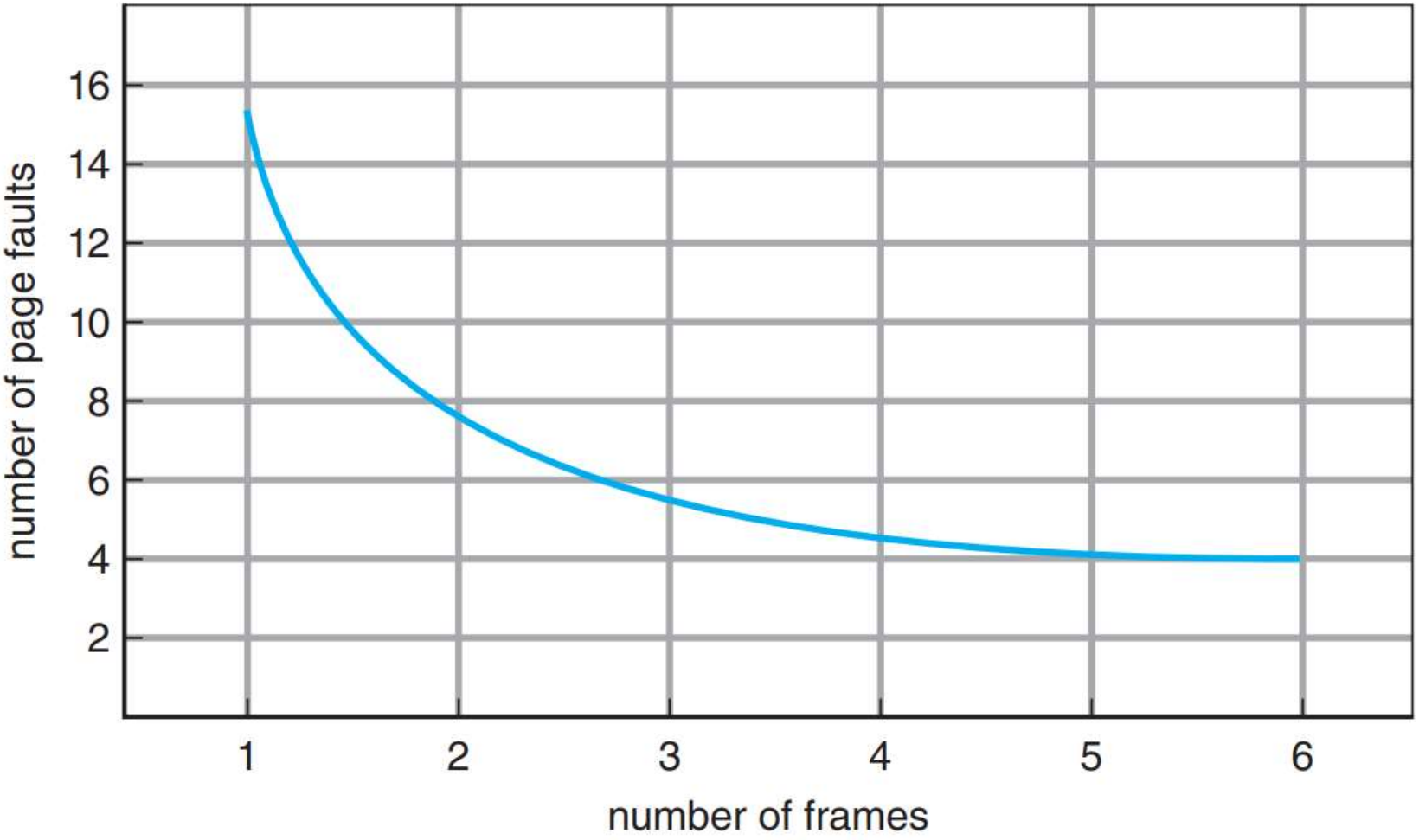


Figure 10.11 Graph of page faults versus number of frames.

FIFO ALGORITHM

FIFO Algorithm

- Always replace the oldest page …
  - *Replace the page that has been in memory for the longest time*

FIFO Algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		**a**	**a**	**a**						
	1	b				**b**						
	2	c	**c**	**c**	**c**	**c**						
	3	d			**d**	**d**						
Page faults							**X**					

FIFO Algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		**a**	**a**	**a**	**a**	**a**	**a**	**a**		
	1	b				**b**	**b**	**b**	**b**	**b**		
	2	c	**c**	**c**	**c**	**c**	**e**	**e**	**e**	**e**		
	3	d			**d**	**d**	**d**	**d**	**d**	**d**		
Page faults							**X**				**X**	

FIFO Algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		**a**	**a**	**a**	**a**	**a**	**a**	**a**	**c**	
	1	b				**b**	**b**	**b**	**b**	**b**	**b**	
	2	c	**c**	**c**	**c**	**c**	**e**	**e**	**e**	**e**	**e**	
	3	d			**d**	**d**	**d**	**d**	**d**	**d**	**d**	
Page faults							**X**				**X**	**X**

FIFO Algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 frames:

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	a
Page Frames	0	a		**a**	**a**	**a**	**a**	**a**	**a**	**a**	**c**	**c**
	1	b				**b**	**b**	**b**	**b**	**b**	**b**	**b**
	2	c	**c**	**c**	**c**	**c**	**e**	**e**	**e**	**e**	**e**	**e**
	3	d			**d**	**d**	**d**	**d**	**d**	**d**	**d**	**a**
Page faults							**X**				**X**	**X**

FIFO Algorithm

- Always replace the oldest page.
  - *Replace the page that has been in memory for the longest time*
- Implementation
  - *Maintain a linked list of all pages in memory*
  - *Keep it in order of when they came into memory*
  - *The page at the tail of the list is oldest*
  - *Add new page to head of list*

FIFO Algorithm

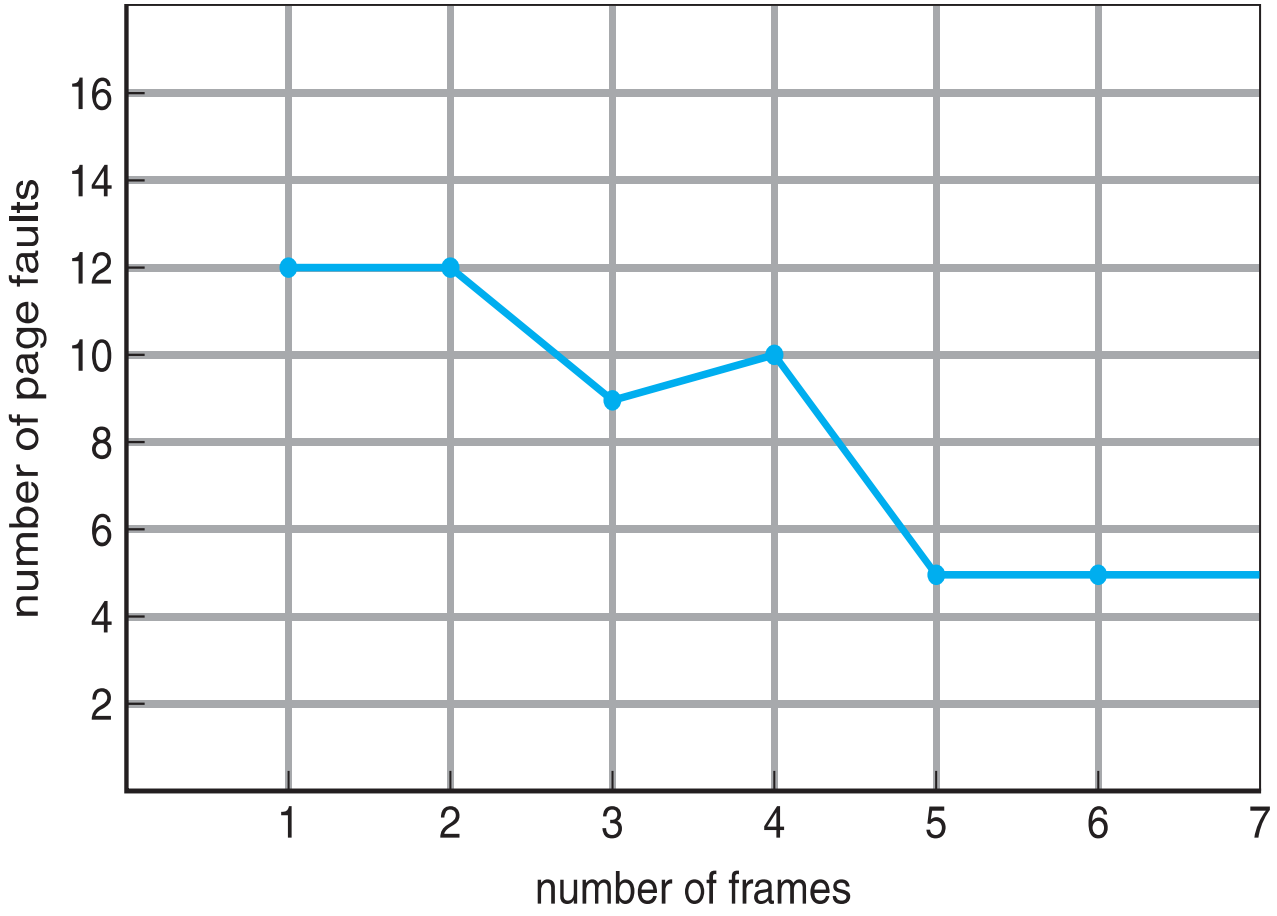
- Disadvantage?
  - *The oldest page may be needed again soon*
  - *Some page may be important throughout execution*
  - *It will get old, but replacing it will cause an immediate page fault*

سوال؟

- در الگوریتم FIFO، اگر تعداد frameها رو زیاد کنیم، اوضاع تعداد page faultها چطور می‌شود؟
  - *لزوما بهتر می‌شود؟!*
  - *متاسفانه لزوما نه!!*

Belady's Anomaly

- Adding more frames can cause more page faults!



NRU ALGORITHM

How Can We Do Better?

- Need an approximation of how likely each frame is to be accessed in the future
  - *If we base this on past behavior we need a way to track past behavior*
  - *Tracking memory accesses requires hardware support to be efficient*

مسئله: یکی از فریم‌هایی که اخیرا استفاده نشده رو بریزیم دور

- چطوری انتخاب کنیم کدوم فریم؟
- علی‌الحساب فرض کنید نمی‌خوایم داده‌ساختار جدیدی نگه داریم...
- مثلا از اطلاعات توی page table استفاده کنیم.
- چه اطلاعاتی داریم؟
- Reference bit
- Dirty bit

Referenced and Dirty Bits

- Each page table entry (and TLB entry!) has a
  - *Referenced bit - set by TLB when page read / written*
  - *Dirty / modified bit - set when page is written*
  - *If TLB entry for this page is valid, it has the most up to date version of these bits for the page*
  - *OS must copy them into the page table entry during fault handling*
- Idea: use the information contained in these bits to drive the page replacement algorithm

Referenced and Dirty Bits

- Some hardware does not have support for the dirty bit
- Instead, memory protection can be used to emulate it
- Idea:
  - *Software sets the protection bits for all pages to "read only"*
  - *When program tries to update the page...*
    - A trap occurs
    - Software sets the Dirty Bit in the page table and clears the ReadOnly bit
    - Resumes execution of the program

Not Recently Used Algorithm

- Uses the Referenced Bit and the Dirty Bit
- Initially, all pages have
  - *Referenced Bit = 0*
  - *Dirty Bit = 0*
- Periodically... (e.g. whenever a timer interrupt occurs)
  - *Clear the Referenced Bit*
  - *Referenced bit now indicates “recent” access*

Not Recently Used Algorithm

- When a page fault occurs...
- Categorize each page...
  - *Class 1:* *Referenced = 0* *Dirty = 0*
  - *Class 2:* *Referenced = 0* *Dirty = 1*
  - *Class 3:* *Referenced = 1* *Dirty = 0*
  - *Class 4:* *Referenced = 1* *Dirty = 1*
- Choose a victim page from …
  - *class 1 … why?*
- If none, choose a page from …
  - *class 2 … why?*
- If none, choose a page from …
  - *class 3 … why?*
- If none, choose a page from …
  - *class 4 … why?*

مسئله: یکی از فریم‌هایی که اخیرا استفاده نشده رو بریزیم دور

- چطوری انتخاب کنیم کدوم فریم؟
- علی‌الحساب فرض کنید نمی‌خوایم داده‌ساختار جدیدی نگه داریم...
- مثلا از اطلاعات توی page table استفاده کنیم.
- راه دیگه‌ای هم داریم؟
- یه طوری می‌خواهیم اونی که اخیرا reference نشده رو بندازیم بیرون...

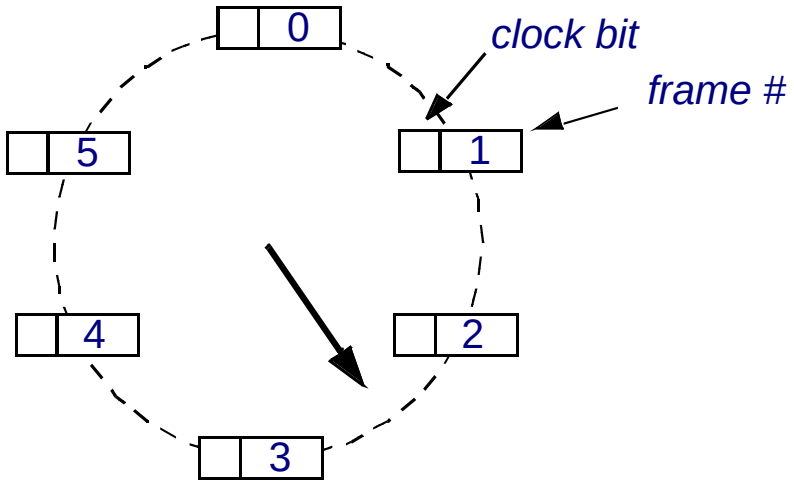
SECOND CHANCE ALGORITHM

Second Chance Algorithm

- An implementation of NRU based on FIFO
- Pages kept in a linked list (oldest at the front)
- Look at the oldest page
  - *If its "referenced bit" is 0...*
    - Select it for replacement
  - *Else*
    - It was used recently; don't want to replace it
    - Clear its "referenced bit"
    - Move it to the end of the list
  - *Repeat*
- What if every page was used in last clock tick?

Implementation of Second Chance

- Maintain a circular list of pages in memory
- Set a bit for the page when a page is referenced
- Search list looking for a victim page that does not have the referenced bit set
  - *If the bit is set, clear it and move on to the next page*
  - *Replaces pages that haven't been referenced for one complete clock revolution*



Enhanced Second Chance algorithm

- همزمان از هر دو ایده‌ی قبلی استفاده کنیم...
- در ۴ گروه براساس reference bit و dirty bit در نظر بگیریم.
- حالا second chance رو براساس گروه‌بندی قبلی انجام بدیم...

LRU

Least Recently Used Algorithm

- A refinement of NRU that orders how recently a page was used
  - *Keep track of when a page is used*
  - *Replace the page that has been used least recently*

Least Recently Used Algorithm

- Replace the page that hasn't been referenced in the longest time

Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page Frames	0	a	a	a	a	a	a	a	a	a	a	a
	1	b	b	b	b	b	b	b	b	b	b	b
	2	c	c	c	c	c	e	e	e	e	e	d
	3	d	d	d	d	d	d	d	d	d	c	c
Page faults							x				x	x

Least Recently Used Algorithm

- But how can we implement LRU?

Least Recently Used Algorithm

- But how can we implement LRU?

- Idea #1:
  - *Keep a linked list of all pages*
  - *On every memory reference, Move that page to the front of the list*
  - *The page at the tail of the list is replaced*

Least Recently Used Algorithm

- But how can we implement LRU?
  - *… without requiring every access to be recorded?*
- Idea #2:
  - *MMU (hardware) maintains a counter*
  - *Incremented on every clock cycle*
  - *Every time a page table entry is used*
    - MMU writes the value to the page table entry
    - This *timestamp* value is the *time-of-last-use*
  - *When a page fault occurs*
    - OS looks through the page table
    - Identifies the entry with the oldest timestamp

Least Recently Used Algorithm

- What if we don't have hardware support for a counter?
- Idea #3:
  - *Maintain a counter in software*
  - *One every timer interrupt...*
    - Increment counter
    - Run through the page table
    - For every entry that has "ReferencedBit" = 1
      - ** Update its timestamp*
      - ** Clear the ReferencedBit*
  - *Approximates LRU*
  - *If several have oldest time, choose one arbitrarily*

NFU

Not Frequently Used Algorithm

- Bases decision of frequency of use rather than recency
- Associate a counter with each page
- On every clock interrupt, the OS looks at each page.
  - *If the reference bit is set increment that page's counter & clear the bit*
- The counter approximates how often the page is used
- For replacement, choose the page with lowest counter

Not Frequently Used Algorithm

- *Problem:*
  - *Some page may be heavily used*
    - Its counter is large
  - *The program's behavior changes*
    - Now, this page is not used ever again (or only rarely)
  - *This algorithm never forgets!*
    - *This page will never be chosen for replacement!*
  - *We may want to combine frequency and recency somehow*

NFU With Aging

- Associate a counter with each page
- On every clock tick, the OS looks at each page.
  - *Shift the counter right 1 bit (divide its value by 2)*
  - *If the reference bit is set...*
    - Set the most-significant bit
    - Clear the Referenced Bit

      - T *100000 = 32*
      - T^1 *010000 = 16*
      - T^2 *001000 = 8*
      - T^3 *000100 = 4*
      - T^4 *100010 = 34*
      - T_5