R

بسم الله الرحمن الرحیم

سیستم عامل

جلسه نوزدهم – الگوریتم‌های جایگزینی صفحه (۴)

جلسه‌ی گذشته

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*

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*

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?*

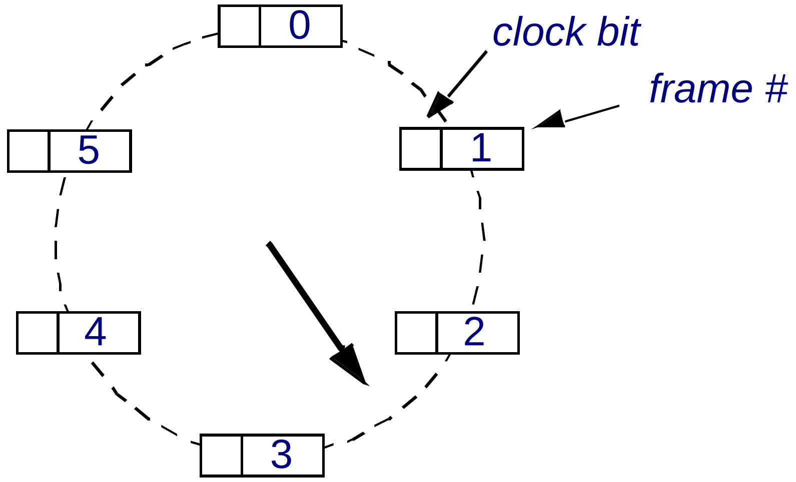
NRU

- *Class 1:* *Referenced = 0 Dirty = 0*
- *Class 2:* *Referenced = 0 Dirty = 1*
- *Class 3:* *Referenced = 1 Dirty = 0*
- *Class 4:* *Referenced = 1 Dirty = 1*

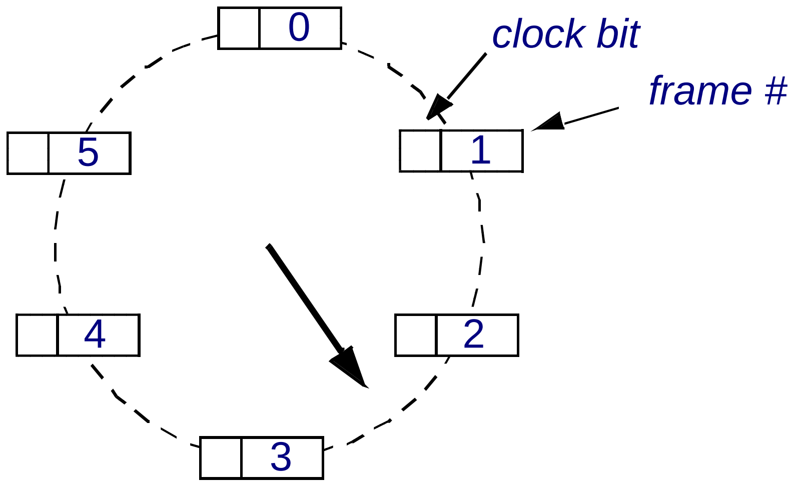
Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	a	d	b	e	b	a	b	c	d
Page	0	a										
Frames	1	b										
	2	c										
	3	d										
Page faults												

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*
- Second Chance with circular queue also referred as Clock Algorithm



Second Chance



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

Page faults

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	0	a	a	a	a	a	a	a	a	a	a	a
Frames	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?

- 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

C = ۲

۱۰

۱۱

- 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*

	RB	C
A	0	۲
B	0	۲
C	0	۱
D	0	۰

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

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

010001
110001

- T_1 *100000 = 32*
- T_2 *010000 = 16*
- T_3 *001000 = 8*
- T_4 *000100 = 4*
- T_5 *100010 = 34*

جلسه‌ی جدید

FRAME ALLOCATION

Local vs. Global Allocation

- Assume we have
  - *5,000 frames in memory and 10 processes*
- Idea: Give each process 500 frames
- Is this fair?
  - *Small processes do not need all those pages*
  - *Large processes may benefit from even more frames*
- Idea:
  - *Look at the size of each process (… but how?)*
  - *Give them a pro-rated number of frames with some minimum*

Local vs. Global Replacement

- Assume several processes: A, B, C, ...
- Some process gets a page fault (say, process A)
- Choose a page to replace.
- *Local page replacement*
  - *Only choose one of A's pages*
- *Global page replacement*
  - *Choose any page*

Local vs. Global Replacement

Original	Age	Local	Global
A0	10	A0	A0
A1	7	A1	A1
A2	5	A2	A2
A3	4	A3	A3
A4	6	A4	A4
A5	3	A6	A5
B0	9	B0	B0
B1	4	B1	B1
B2	6	B2	B2
B3	2	B3	A6
B4	5	B4	B4
B5	6	B5	B5
B6	12	B6	B6
C1	3	C1	C1
C2	5	C2	C2
C3	6	C3	C3

PROACTIVE REPLACEMENT

Proactive Replacement

- Replacing victim frame on each page fault typically requires two disk accesses per page fault
- Alternative → the O.S. can keep several pages free in anticipation of upcoming page faults
- Free List (Inactive list): List of frames that ready for replacement

Proactive Replacement

- Page Faults:
  - *Major Page Fault:*
    - a page is referenced and the page is not in memory.
  - *Minor Page Fault:*
    - process does not have a logical mapping to a page, yet that page is in memory
- Minor Page faults:
  - *process may reference a shared library that is in memory, but the process does not have a mapping to it in its page table*
  - *page is reclaimed from a process and placed on the free-frame list, but the page has not yet been zeroed out and allocated to another process*

Proactive Page Replacement

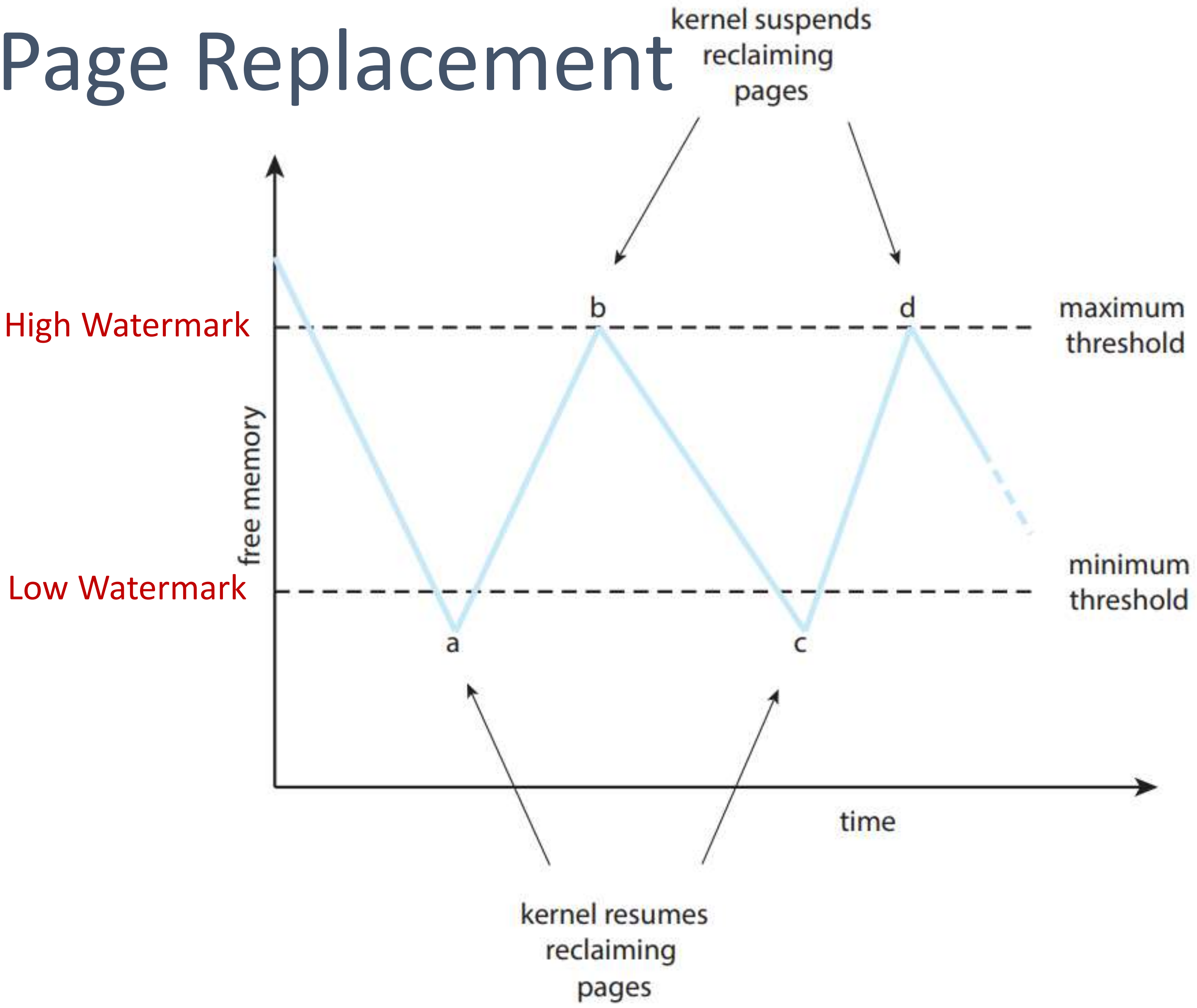


Figure 10.18 Reclaiming pages.

PAGE REPLACEMENT IN LINUX

UNIX Page Replacement

- Some Proactive Page Replacement
- Enable Swapping daemon (kwapd) on low watermark

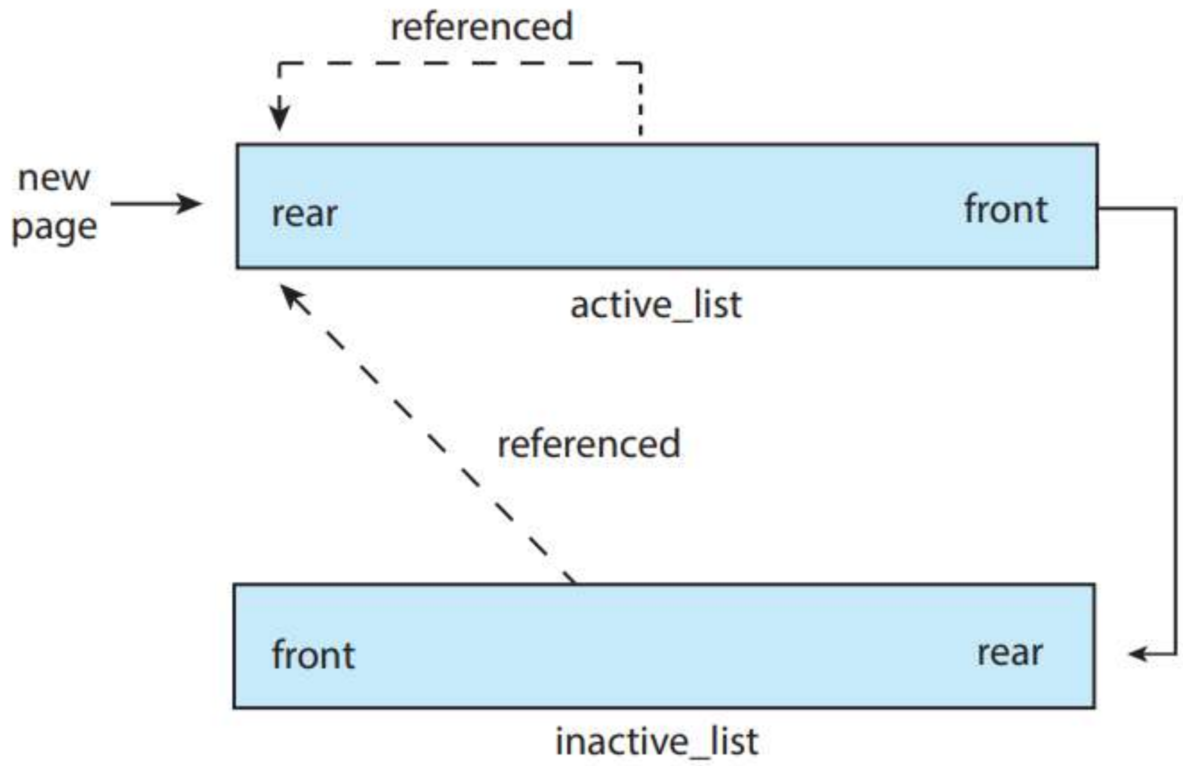


Figure 10.29 The Linux `active_list` and `inactive_list` structures.

UNIX Page Replacement

- Second Chance (Clock) Page replacement used as Approximation of LRU

WORKING SET

The Working Set

- *Locality of reference*
  - *Processes tend to use only a small fraction of their pages*
- *Working Set*
  - *The set of pages a process needs*
  - *If working set is in memory, no page faults*
  - *What if you can't get working set into memory?*

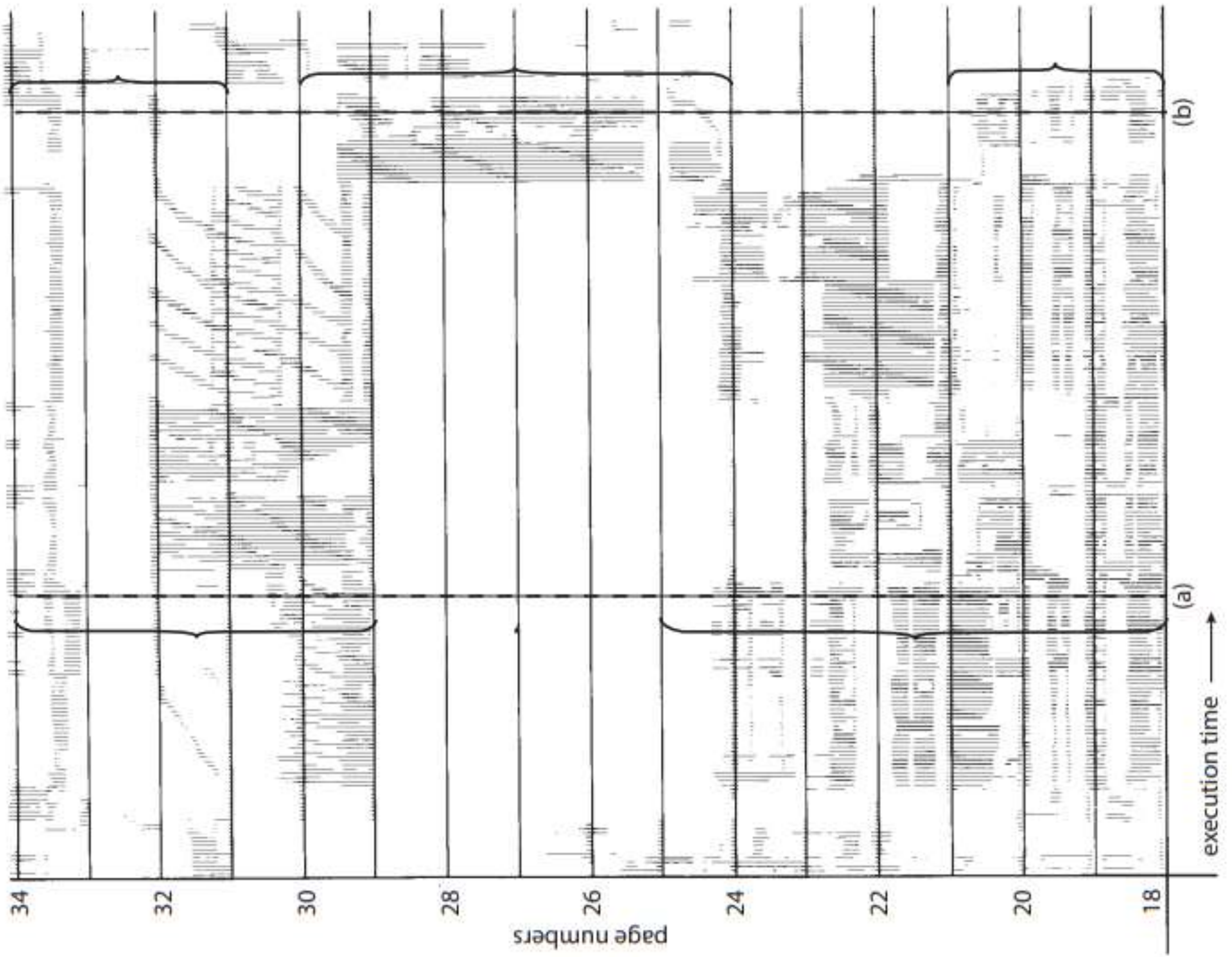


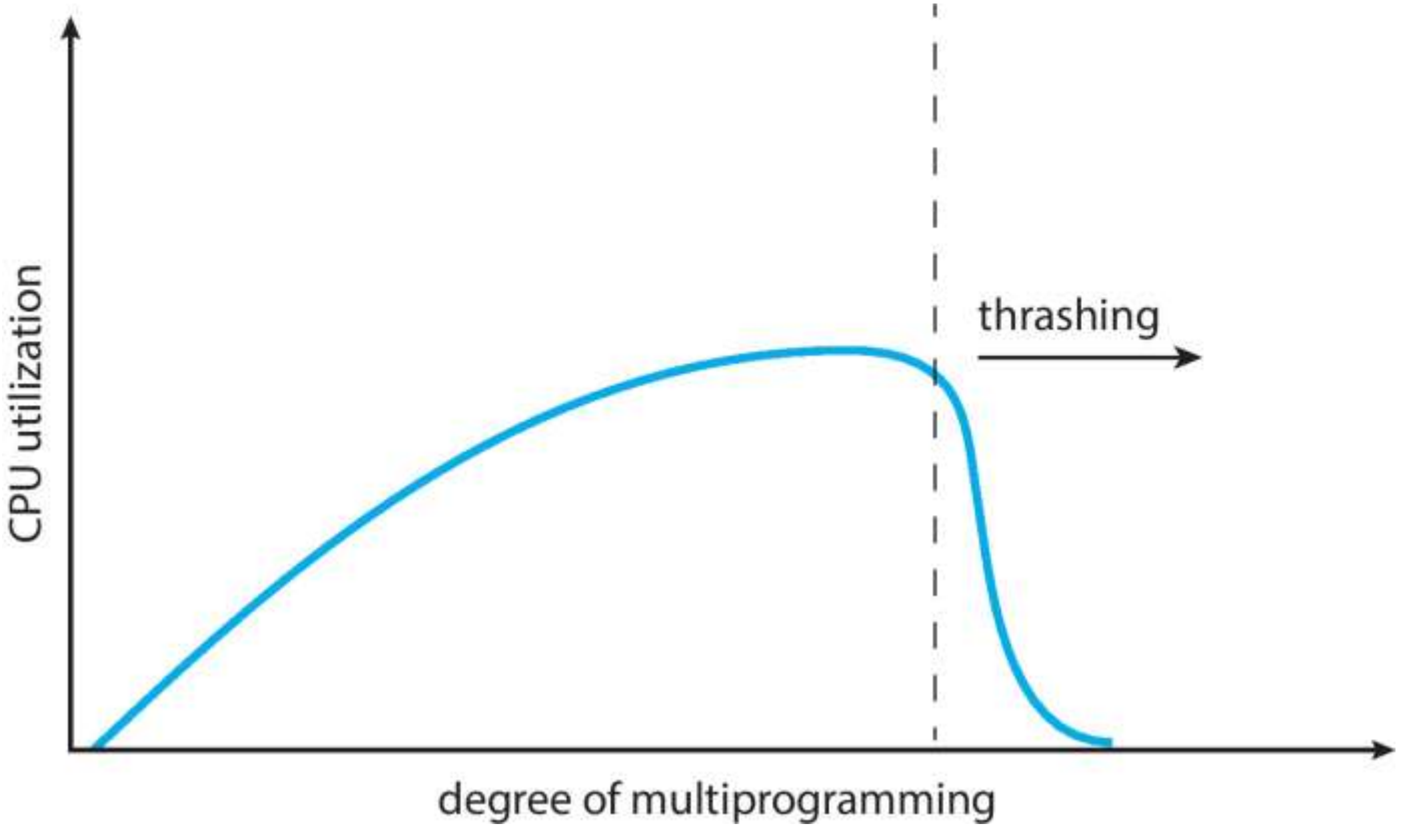
Figure 10.21 Locality in a memory-reference pattern.

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - *Page fault to get page*
  - *Replace existing frame*
  - *But quickly need replaced frame back*
  - *This leads to:*
    - Low CPU utilization
    - Operating system thinking that it needs to increase the degree of multiprogramming
    - Another process added to the system

Thrashing (Cont.)

- **Thrashing**. A process is busy swapping pages in and out



Working Set Algorithm

- *Based on prepaging (prefetching)*
  - *Load pages before they are needed*
- *Main idea:*
  - *Try to identify the process's working set based on time*
  - *Keep track of each page's time since last access*
  - *Assume working set valid for T time units*
  - *Replace pages older than T*

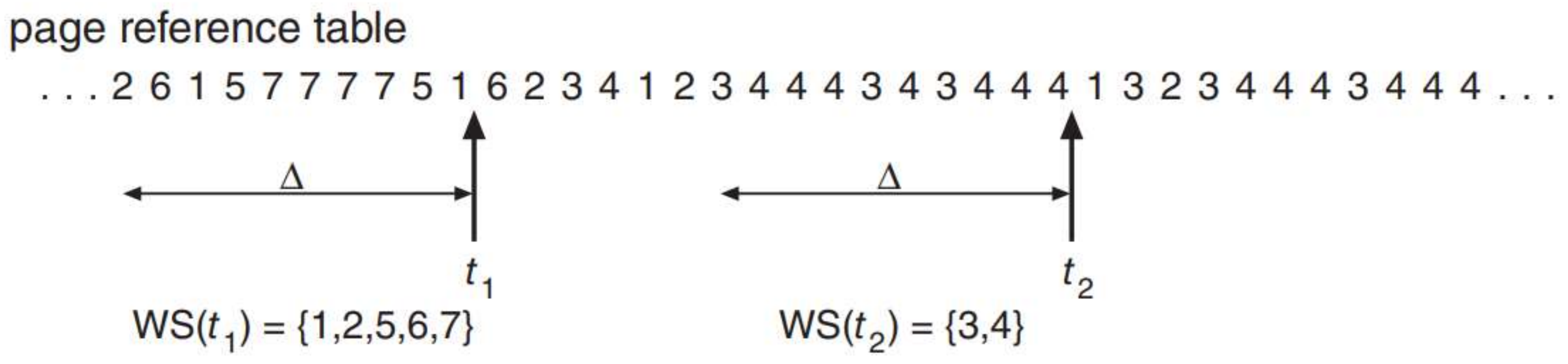
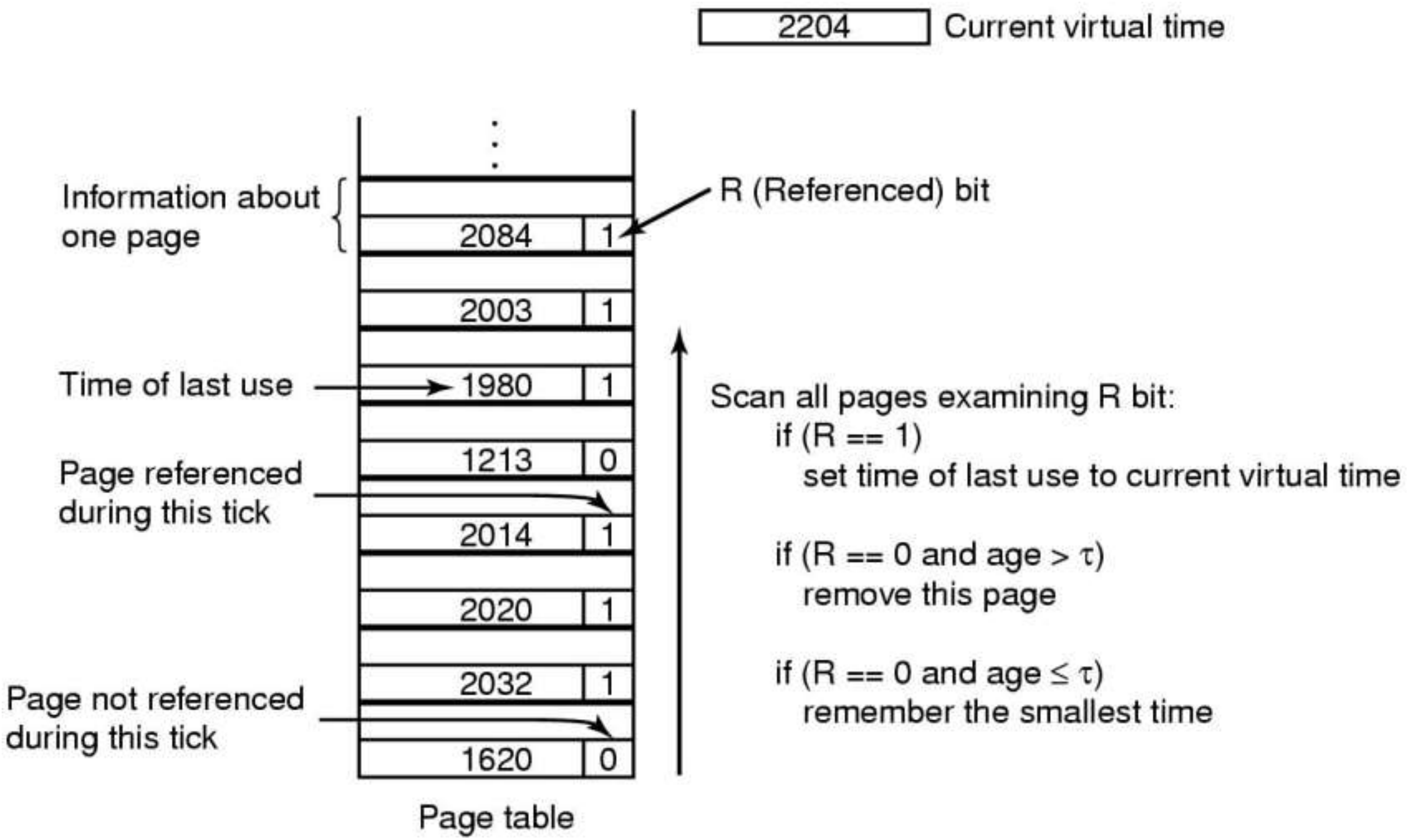


Figure 10.22 Working-set model.

Working Set Algorithm

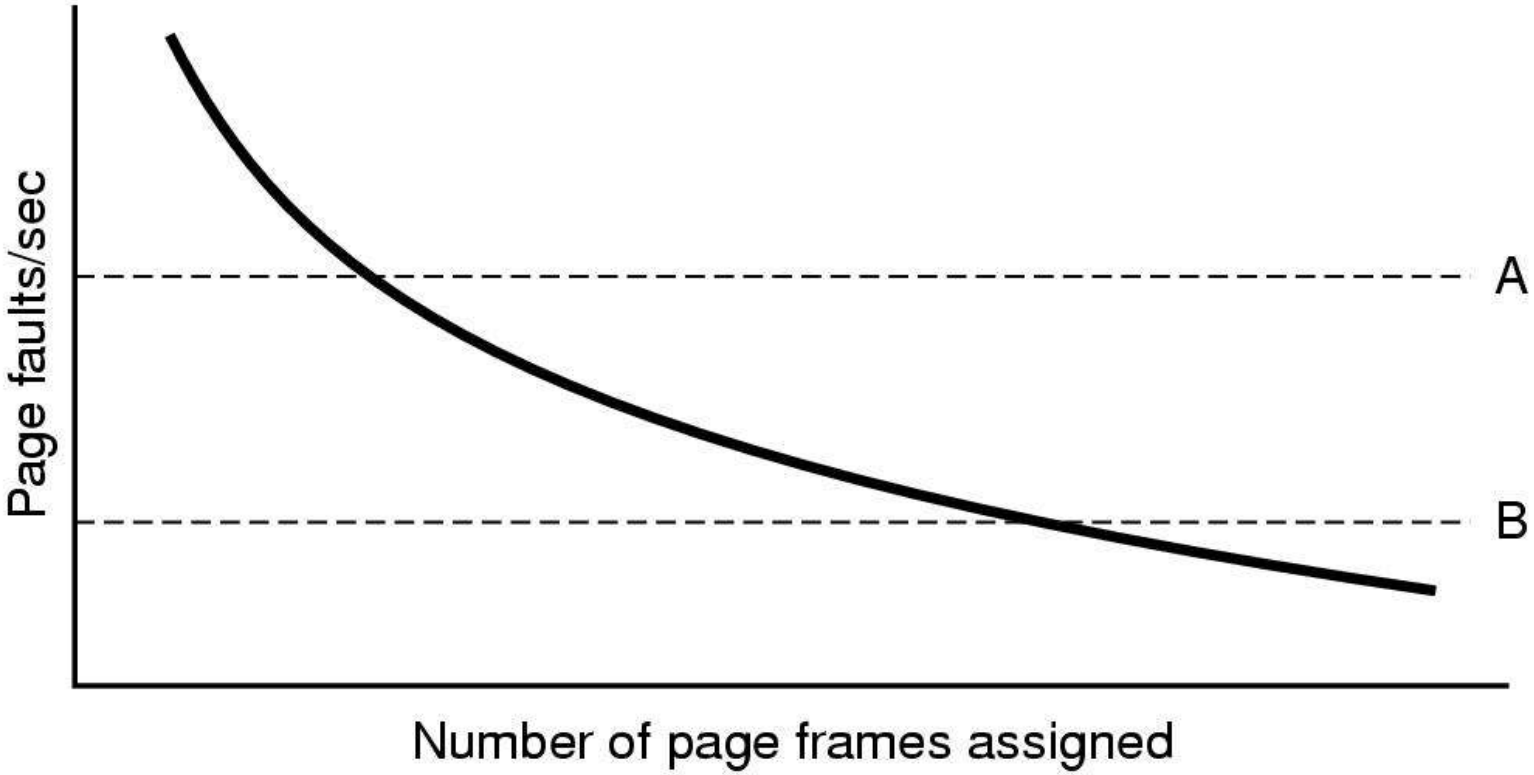
- *Current Virtual Time*
  - *Only consider how much CPU time this process has seen*
- *Implementation*
  - *On each clock tick, look at each page*
  - *Was it referenced since the last check?*
    - Yes: make a note of Current Virtual Time
  - *If a page has not been used in the last T msec,*
    - Assume it is not in the working set!
    - Evict it
    - Write it out if it is dirty

Working Set Algorithm

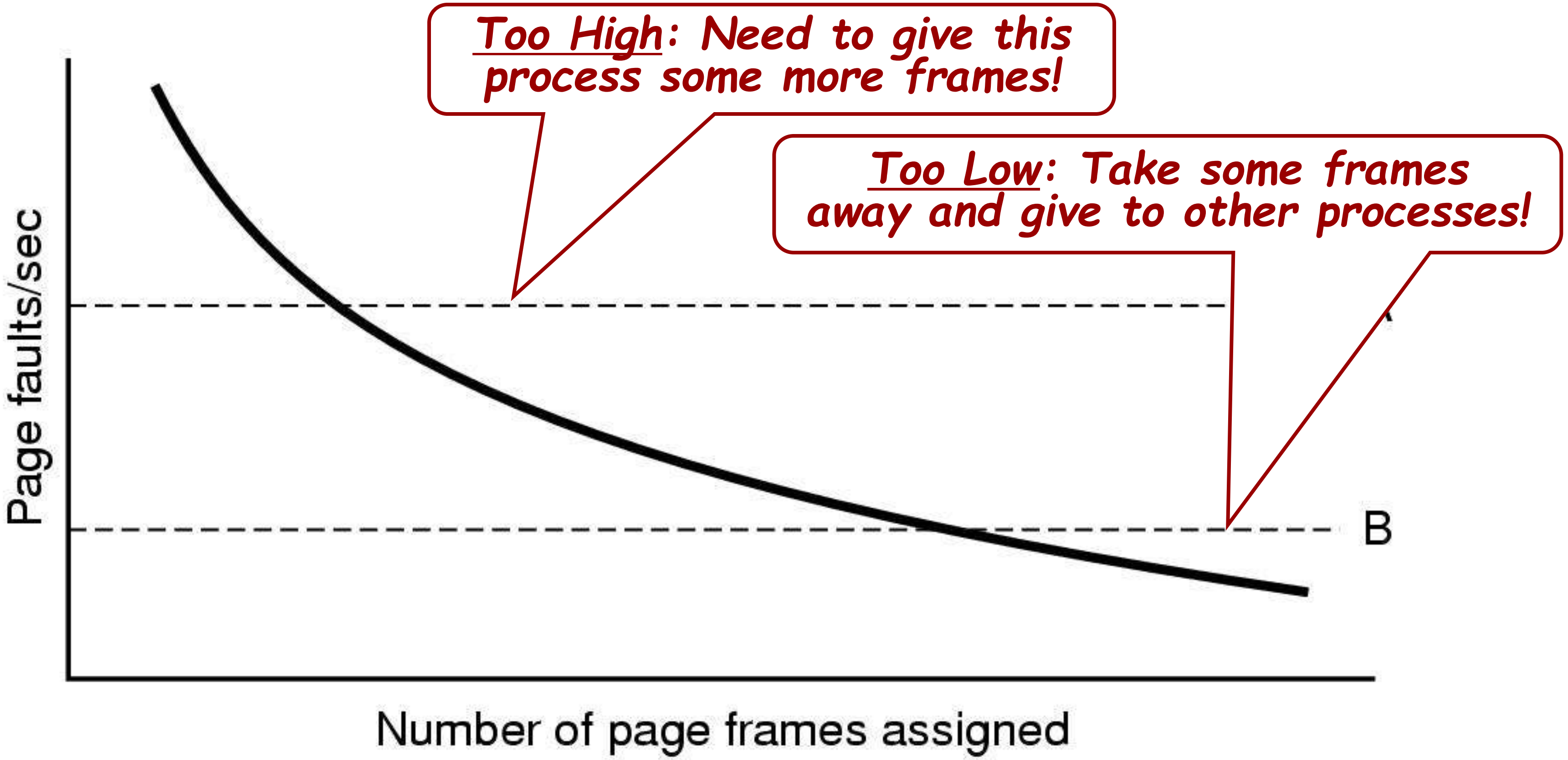


Page Fault Frequency

- *If T is too small, page fault frequency will be high*
- *If you make it bigger page fault frequency will decline*



Page Fault Frequency



Page Fault Frequency

- Measure the page fault frequency of each process
- Count the number of faults every second
- May want to consider the past few seconds as well
- Aging:
  - *Keep a running value*
  - *Every second*
    - Count number of page faults
    - Divide running value by 2
    - Add in the count for this second

WHICH ALGORITHM IS BEST?

Modeling Algorithm Performance

- Run a program
  - *Look at all memory references*
  - *Don't need all this data*
  - *Look at which pages are accessed*
  - *000000122233330011444400112344 4*
  - *Eliminate duplicates*
  - *012301401234*

- This defines the *Reference String*
  - *Use this to evaluate different algorithms*
  - *Count page faults given the same reference string*

Load Control

- Assume:
  - *The best page replacement algorithm*
  - *Optimal global allocation of page frames*
- *Thrashing is still possible!*
  - *Too many page faults!*
  - *No useful work is getting done!*
  - *Demand for frames is too great!*
- Solution:
  - *Get rid of some processes (temporarily swap them out)*
  - *Two-level scheduling (swapping with paging)*

PAGE FAULT AND PROGRAMMING

Program Structure

- Program structure
  - *int[128,128] data;*
  - *Each row is stored in one page*
  - *Program 1*

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

  - *Program 2*

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 page faults