Virtual Memory: Policies (Part II)

- Recap: Page Replacement
- Policy: Resident Set Management
  - The Working Set Model
- Policy: Page Caching
- Case Study: Solaris
- Interlude: Software Implementation of Page Table Entries.
- Policies beyond Page Replacement and Caching

Recap: Page Replacement

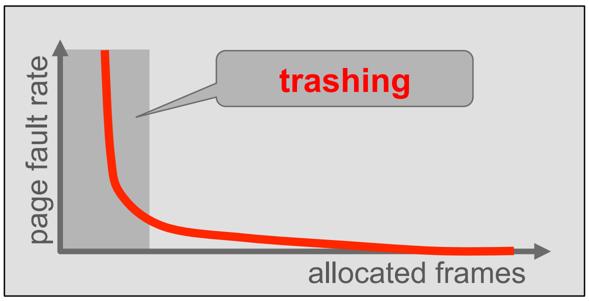
"The chief problem in memory management is not to decide which pages to load, but which pages to *remove.* For, if the page with the least likelihood of being reused in the immediate future is retired to auxiliary memory, the best choice has been made."

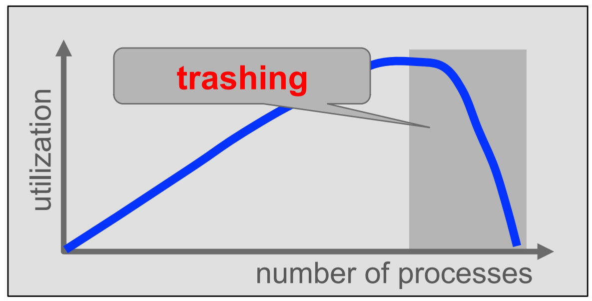
Peter J. Denning, "The Working Set Model for Program Behavior". Communications of the ACM, Vol 11, No. 5, May 1968.

Resident Set Management

Question: How many pages should we keep in memory for this process?

Observation 1: Each process requires a certain minimum set of pages to be resident in memory to run efficiently.

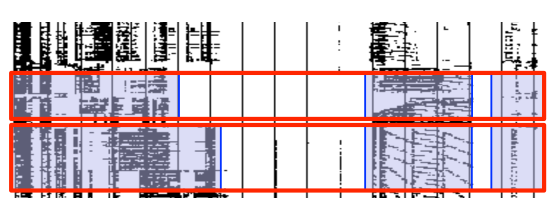




Resident Set Management

Question: How many pages should we keep in memory for this process?

Observation 2: The size of this set changes dynamically as a process executes.



Resident Set Management

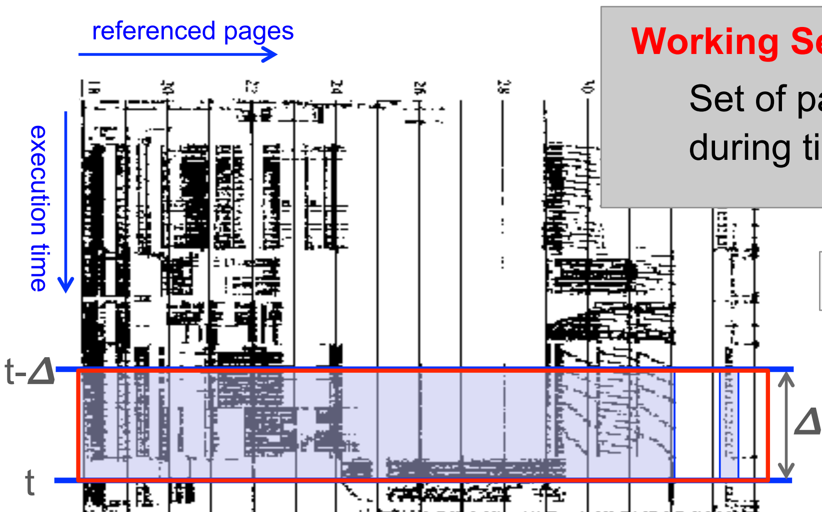
Question: How many pages should we keep in memory for this process?

Observation 1: Each process requires a certain minimum set of pages to be resident in memory to run efficiently.

Observation 2: The size of this set changes dynamically as a process executes.

This leads to algorithms that attempt to maintain an optimal resident set for each active program. (Page replacement with variable number of frames.)

The Working Set Model



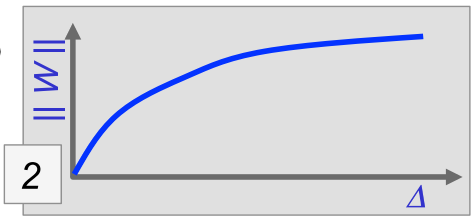
Working Set $W(t,\Delta)$:

Set of pages referenced by process during time interval $[t-\Delta, t]$.

Characteristics of $W(t,\Delta)$:

1 $$||W(t,1)|| = 1$$

$$1 \leq ||W(t,\Delta)|| \leq min(\Delta, N)$$

2 

3 $W(t, \Delta)$ is good predictor for $W(t+d, \Delta)$.

The Working Set Model Algorithm

Working Set *W(t,Δ)* : Set of pages referenced by process during time interval *(t-Δ, t)*.

The memory management strategy follows two rules:

Rule 1: At each reference, the current working set is determined and only those pages belonging to the working set are retained in memory.

Rule 2: A program may run only if its entire current working set is in memory.

Underlying Assumption: Size of working set remains constant over small time intervals.

Working Set Model Algorithm

time					1	2	3	4	5	6	7	8	9	10
reference string		e	d	a	c	c	d	b	c	e	c	e	a	d
working set W(t, Δ), Δ = 4		e	d e	a d e	a c d e	a c d	a c d	b c d	b c d	b c d e	b c e	c e	a c e	a c d e

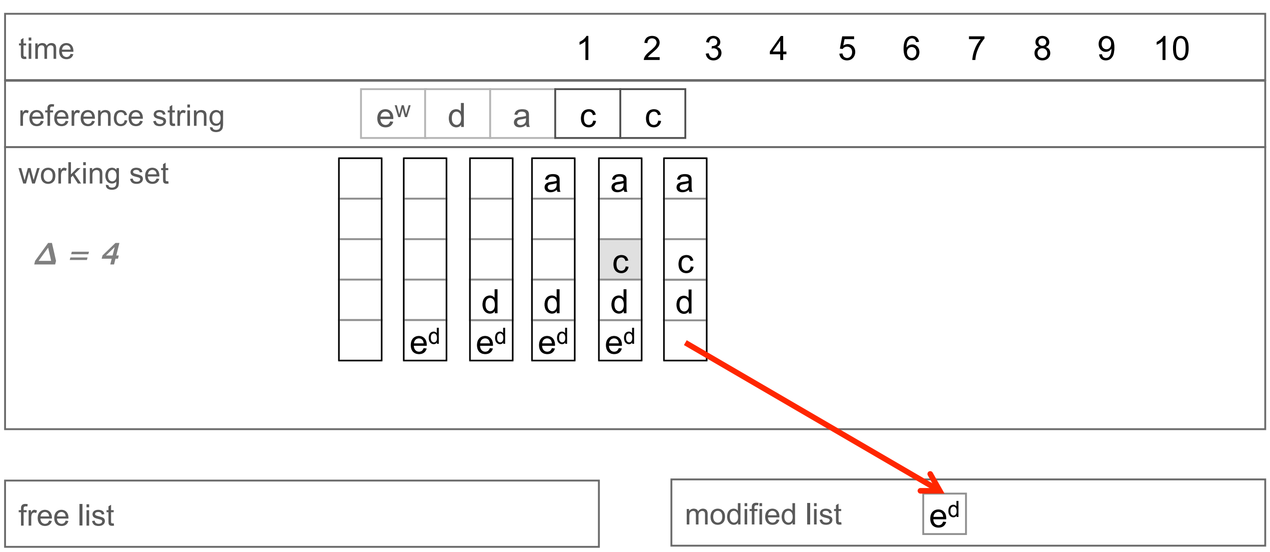
Problems:

- Difficulty in keeping track of working set.
- Estimation of appropriate window size Δ.
  - too small -> page out useful pages; too big -> waste memory

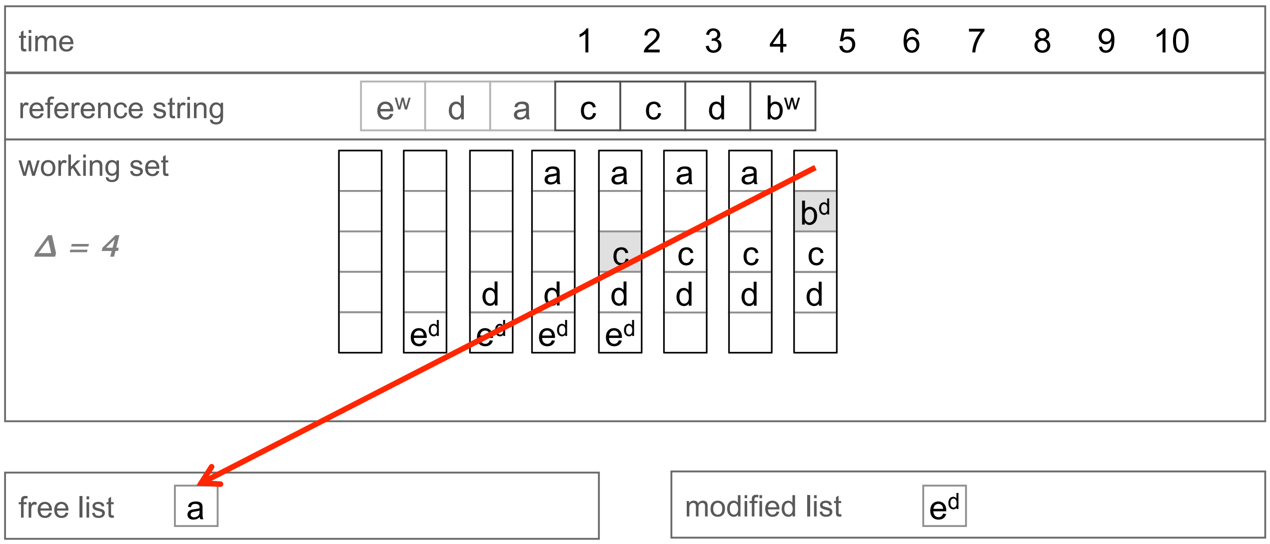
Improve Paging Performance: Page Buffering

- Victim frames are not overwritten directly, but are removed from page table of process, and put into:
  - **free** frame list (clean frames)
  - **modified** frame list (modified frames)
- Victims are picked from the **free** frame list in FIFO order.
- If referenced page is in free or modified list, simply reclaim it.
- Periodically (or when running out of free frames) write **modified** frame list to disk and add now clean frames to free list.

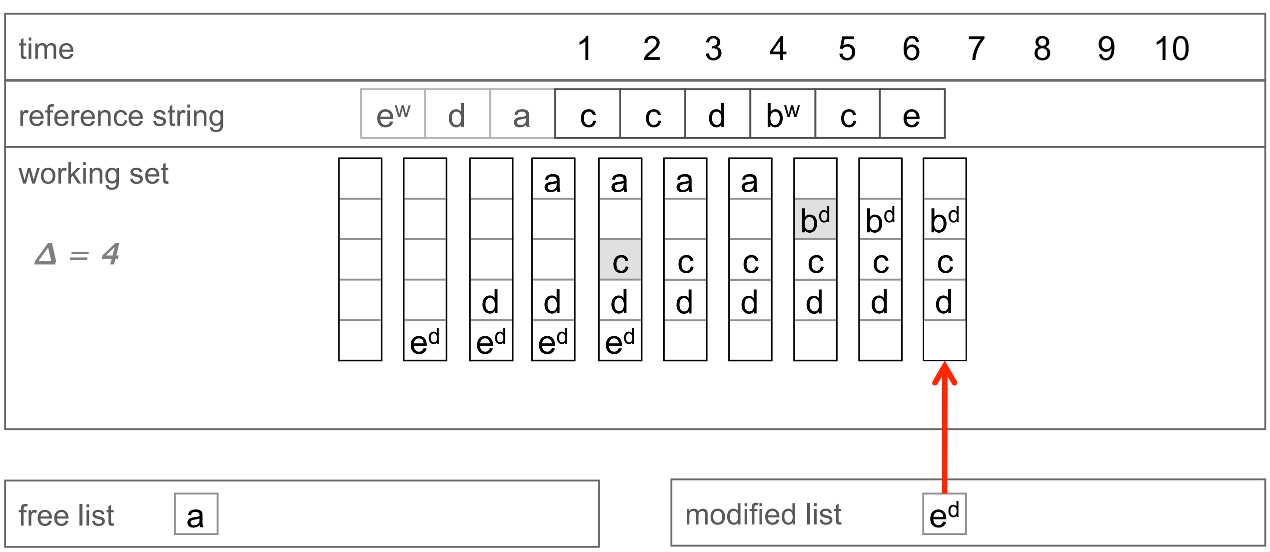
Working Set with Page Caching



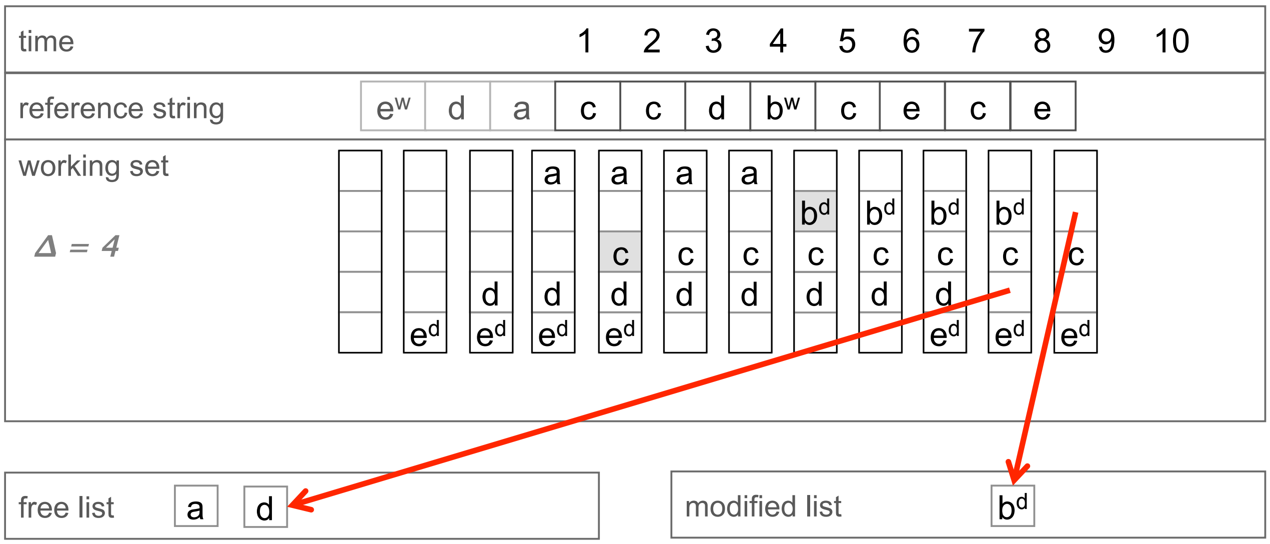
Working Set with Page Caching



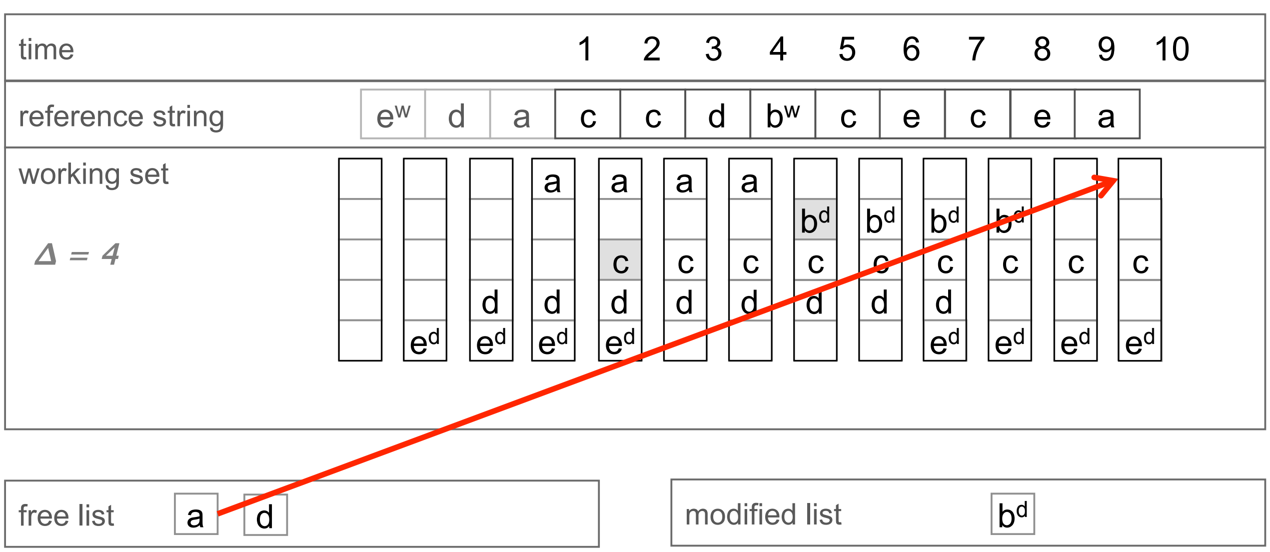
Working Set with Page Caching



Working Set with Page Caching



Working Set with Page Caching



Working Set with Page Caching

time				1	2	3	4	5	6	7	8	9	10
reference string	e^w	d	a	c	c	d	b^w	c	e	c	e	a	
working set				a	a	a	a						a
								b^d	b^d	b^d	b^d		
$\Delta = 4$					c	c	c	c	c	c	c	c	c
			d	d	d	d	d	d	d	d			
		e^d	e^d	e^d	e^d					e^d	e^d	e^d	e^d

OS needs memory!

free list ~~d~~

modified list b^d

Working Set with Page Caching

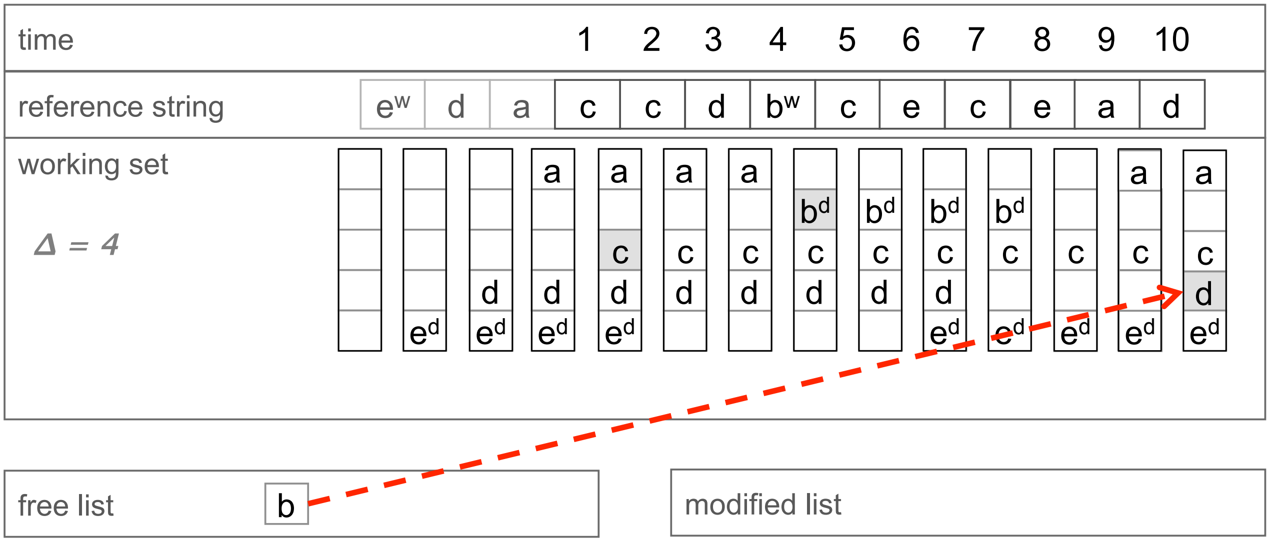
time				1	2	3	4	5	6	7	8	9	10
reference string	e^w	d	a	c	c	d	b^w	c	e	c	e	a	
working set				a	a	a	a						a
								b^d	b^d	b^d	b^d		
$\Delta = 4$					c	c	c	c	c	c	c	c	c
			d	d	d	d	d	d	d	d			
		e^d	e^d	e^d	e^d					e^d	e^d	e^d	e^d

Page out mod. list!

free list

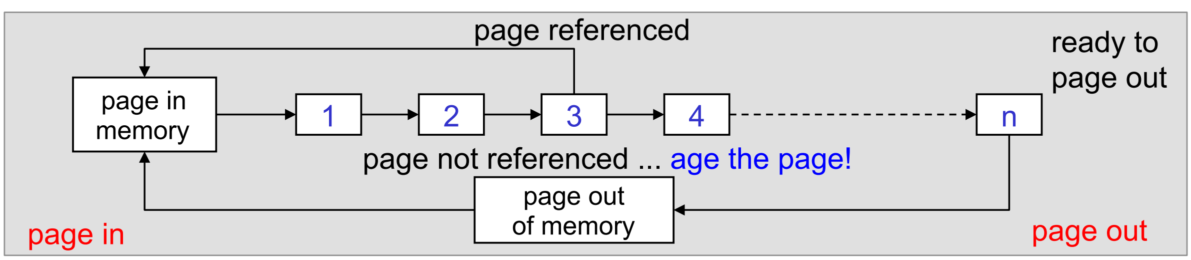
modified list b^d

Working Set with Page Caching



Case Study: Page Buffering in Solaris

- Kernel process (e.g., **pageout** in Solaris) pages out memory frames that are no longer part of a working set of a process, using use bits.
- Periodically increments age register in valid pages.



- **Page stealer** wakes up when available free memory is below *low-water mark*. Pages out frames until available free memory exceeds *high-water mark*.
- Page stealer collects frames and pages them out in a single run. Until then, frames are still available for reference.

Demand Paging on Less-Sophisticated Hardware

Observation1: Demand paging requires *valid* bit that is managed in hardware.

Observation2 : Demand paging most efficient if hardware sets the *use* and *dirty* bits and causes a protection fault when a process writes a page whose *copy_on_write* bit is set.

Other systems require other support, such as *age register* in Solaris.

Fun Fact: We can duplicate *valid* bit by a *software-valid* bit and have the kernel turn off the *valid* bit. The other bits can then be simulated in software.

Demand Paging on Less-Sophisticated Hardware

Example: Implementing ***Use*** Bit in Software**:**

- Maintain a virtual page table entry for each page in software.
- Virtual page table entry has *software-valid* and *software-use* bit.
- For a valid, non-used page, set *valid* bit to 0, *software-valid* bit to 1, and *software-use* bit to 0.
- Reference to page => page fault, because *valid* bit is 0. Page fault handler checks *software-valid* bit.
- If set, kernel knows that page is really valid and it sets *software-use* bit.

Hardware Valid	*Software Valid*	*Software Use*
0	1	0

before referencing valid page

Hardware Valid	*Software Valid*	*Software Use*
1	1	1

after referencing valid page

Demand Paging on Less-Sophisticated Hardware

Example: Implementing ***Aging Register*** in Software:

- Maintain a virtual page table entry for each page in software.
- Virtual page table entry has *software-valid* and *aging-register*.
- Page is loaded: set *valid* bit to 0, *software-valid* bit to 1, and *aging-register* bit to 0.
- Page is referenced: page fault because *valid* bit is 0. Page fault handler then checks *software-valid* bit.
- If set, kernel knows that page is valid and sets *aging-register* to 0.
- Then set *valid* bit to 1 to prevent further, unnecessary faults.
- Periodically, increment *aging-registers* and set *valid* bit to 0 for all pages with *software-valid* bit set.

OS Policies for Virtual Memory

- **Fetch Policy**
  - How/when to get pages into physical memory?
  - demand paging *vs*. prepaging.
- **Placement Policy**
  - Where in physical memory to put pages?
  - Only relevant in NUMA machines.
- **Replacement Policy**
  - Physical memory is full. Which frame to page out?
- **Resident Set Management Policy**
  - How many frames to allocate to process?
  - Replace someone else's frame?
- **Cleaning Policy**
  - When to write a modified page to disk.
- **Load Control**

Virtual Memory: Policies (Part II)

- Resident Set Management & Working Set Model
- Page Caching
- Case Study: Solaris
- Interlude: Software Implementation of Page Table Entries.
- Policies beyond Page Replacement and Caching