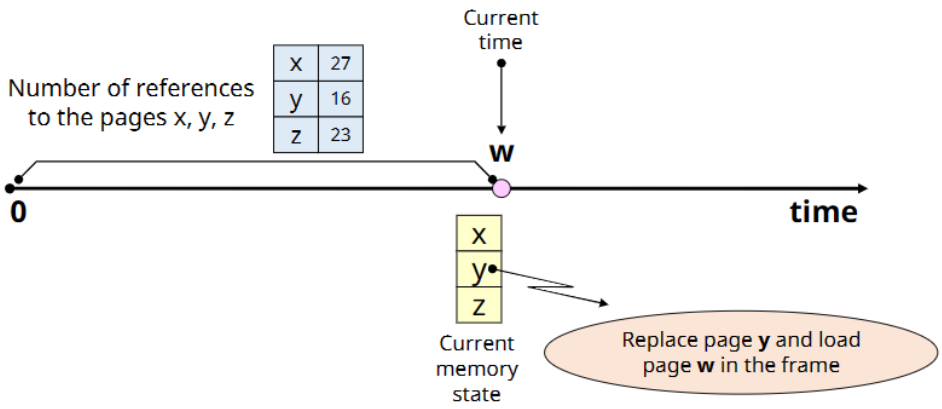
10-2. Virtual Memory Management 4-6

Replacement Strategies: Fixed Allocation

- LRU: 시간 기록 오버헤드가 있음

LFU (Least Frequently Used) Algorithm

> 가장 참조 횟수가 적은 page를 교체
>
> - Tie-reaking rule이 필요 (cf. LRU)



y가 가장 적게 참조됨

- Page 참조 시마다 **참조 횟수**를 누적시켜야 함
- Locality 활용: LRU 대비 적은 오버헤드
- **단점**
  - 최근 적재된, 참조될 가능성이 높은 page가 교체될 가능성이 있음
  - 참조 횟수 누적 오버헤드 존재

[예시]

- **Example**
  - Four-page frames for the process, initially empty
  - ω = 1 2 6 1 4 5 1 2 1 4 5 6 4 5

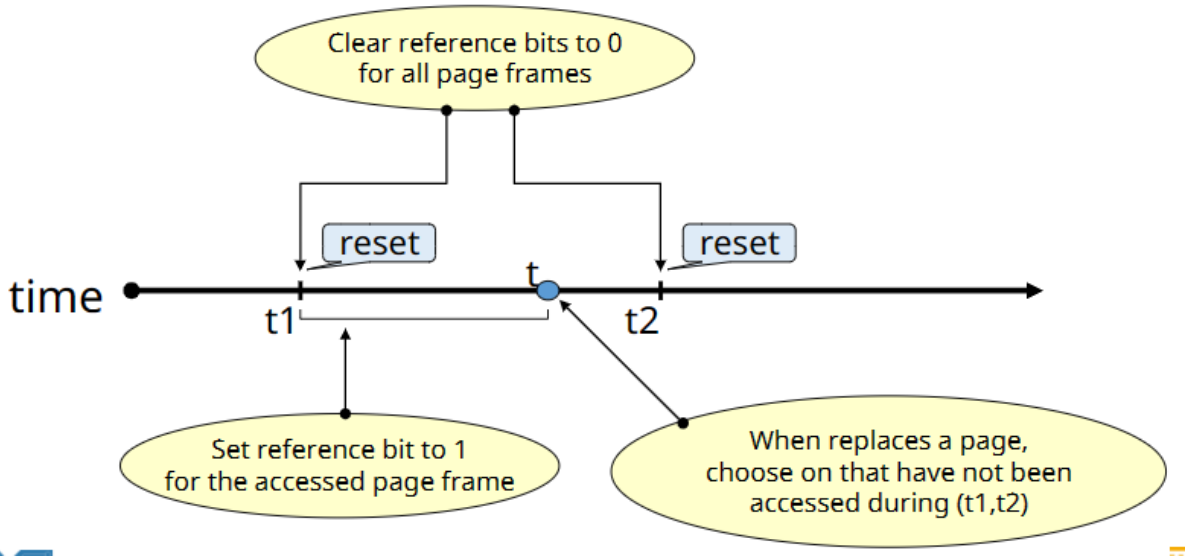
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ref. string	1	2	6	1	4	5	1	2	1	4	5	6	4	5
Memory state	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		2	2	2	2	5	5	5	5	5	5	5	5	5
			6	6	6	6	6	2	2	2	2	6	6	6
					4	4	4	4	4	4	4	4	4	4
Page fault	F	F	F		F	F		F				F		

  - **Number of page faults =**

- number of page faults: 7

NUR (Not Used Recently) Algorithm

> LRU approximation scheme
>
> - LRU보다 적은 오버헤드(횟수 누적)로 비슷한 성능 달성 목적

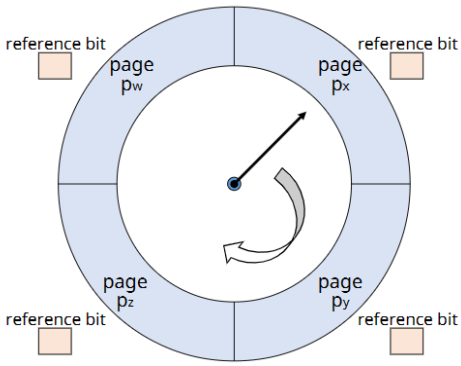


- **Bit vector 사용**
  - reference bit vector (r), update bit vector (m)
- **교체 순서**

1. (r, m) = (0,0): 참조 X, 갱신 X
2. (r,m) = (0,1)
3. (r,m) = (1,0)
4. (r,m) = (1,1): 참조 O, 갱신 O (메모리에서 내려올 때 write back 필요)

Clock Algorithm

> Page frame들을 순차적으로 가리키는 pointer(시계바늘)를 사용해 교체될 page 결정
>
> - Reference bit 사용: 주기적인 초기화 없음



시계침이 돌아가면서 바꿀 수 있는지 여부 확인

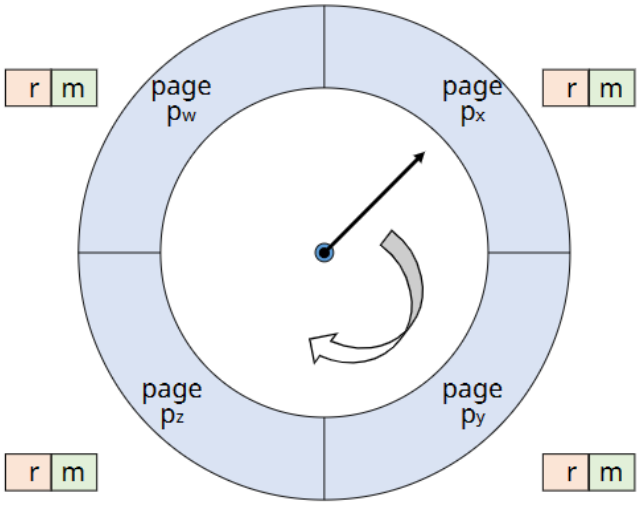
- NUR 실제 적용 예: IBM VM/370 OS
- **Pointer를 돌리면서 교체 page 결정**
  - 현재 가리키고 있는 page의 reference bit(r) 확인
  - r = 0인 경우, 교체 page로 결정
  - r = 1인 경우, reference bit 초기화 후 pointer 이동
    - 시침이 돌아가는 시점이 초기화 시점
- 먼저 적재된 page가 교체될 가능성이 높음
  - FIFO와 유사
- Reference bit를 사용하여 교체 페이지 결정
  - LRU (or NUR)과 유사 (locality 반영)

[예시]

- **Example**
  - 4 page frames for the process, initially it has a, b, c, d
  - ω = c a d b e b a b c d

Time		0	1	2	3	4	5	6	7	8	9	10
Ref. string			c	a	d	b	e	b	a	b	c	d
Memory state	frame 0		→a/1	→a/1	→a/1	→a/1	**e/1**	e/1	e/1	e/1	→e/1	**d/1**
	frame 1		b/1	b/1	b/1	b/1	→b/**0**	→b/**1**	b/0	b/1	b/1	→b/**0**
	frame 2		c/1	c/1	c/1	c/1	c/**0**	c/0	**a/1**	a/1	a/1	a/**0**
	frame 3		d/1	d/1	d/1	d/1	d/**0**	d/0	→ d/0	→ d/0	**c/1**	c/**0**
Page fault Pclock (loaded page) Qclock (displaced page)							F e a		F a c		F c d	F d e

Second Chance Algorithm



- Clock algorithm과 유사
- **Update bit(m)도 함께 고려함**
  - 현재 가리키고 있는 page의 (r,m) 확인
  - (0,0): 교체 page로 결정
  - (0,1): ➡️ (0,0), write-back (cleaning) list에 추가 후 이동
  - (1,0): ➡️ (0,0) 후 이동
  - (1,1): ➡️ (0,1) 후 이동

[예시]

- **Example**
  - Four-page frames for the process, initially it has a, b, c, d
  - ω = c a^W d b^W e b a^W b c d

Time		0	1	2	3	4	5	6	7	8	9	10
Ref. string			c	a^W	d	b^W	e	b	a^W	b	c	d
Memory state	frame 0	→a/10	→a/10	→a/1**1**	→a/11	→a/11	a/**00**	a/00	a/**11**	a/11	→a/11	a/**00**
	frame 1	b/10	b/10	b/10	b/10	b/1**1**	b/**00**	b/**1**0	b/10	b/10	b/10	**d/10**
	frame 2	c/10	c/10	c/10	c/10	c/10	**e/10**	e/10	e/10	e/10	e/10	→e/**0**0
	frame 3	d/10	d/10	d/10	d/10	d/10	→d/**00**	→d/00	→ d/00	→ d/00	**c/10**	c/**0**0
Page fault P2nd-chance Q2nd-chance							F e c				F c d	F d b

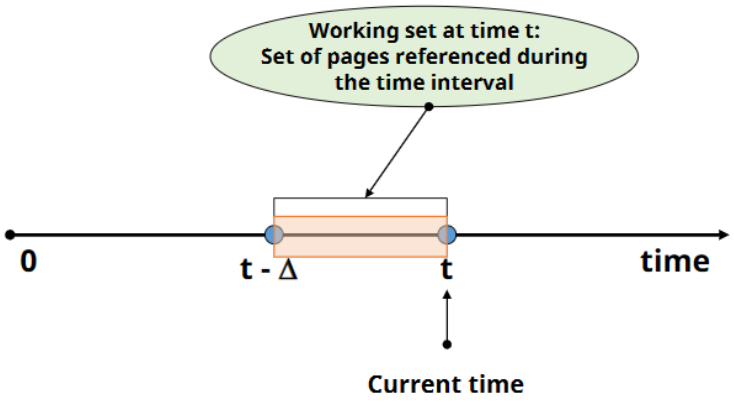
w: write back operation

Other Algorithms

- Additional reference bits algorithm
  - LRU approximation
  - 여러 개의 reference bit를 가짐
    - 각 time-interval에 대한 참조 여부 기록
    - History register for each page
- MRU (Most Recently Used) Algorithm
  - LRU와 정반대 기법
- MFU (Most Frequently Used) Algorithm
  - LFU와 정반대 기법

Replacement Strategies: Variable Allocation

WS (Working Set) Algorithm



주황색 영역: window

- 1968년 Denning이 제안

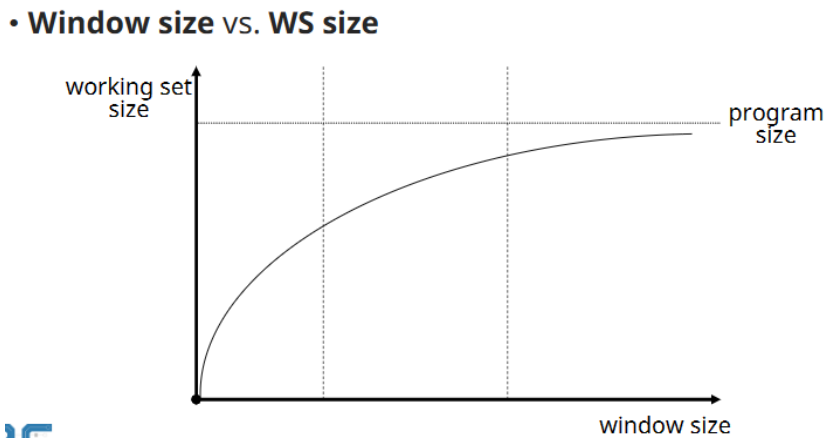
[Working Set]

- Process가 특정 시점에 자주 참조(locality)하는 page들의 집합
- 최근 일정 시간 동안(Δ) 참조된 page들의 집합
  - 시간에 따라 변함
- $W(t, \Delta)$
  - the working set of a process at time t
  - time interval $[t - \Delta, t]$ 동안 참조된 page들의 집합
  - Δ: window size, system parameter

[Working set memory management]

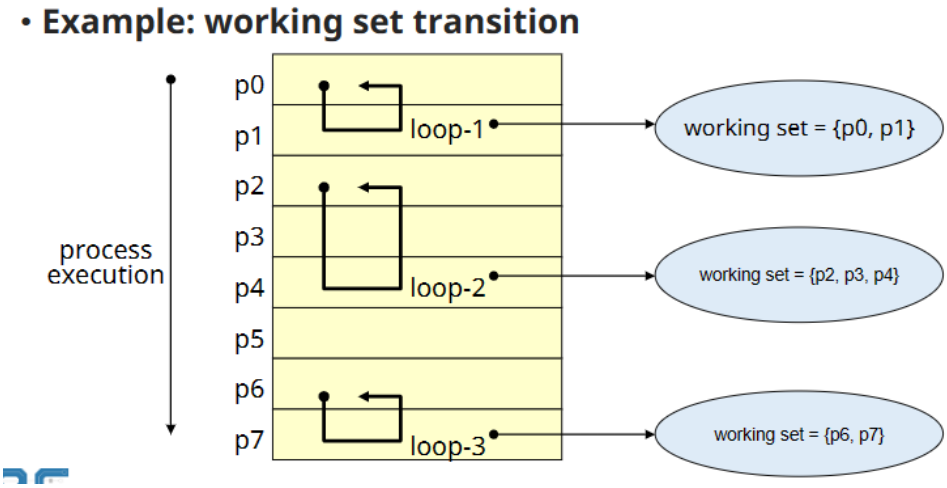
- Locality 에 기반을 둠
- Working set을 메모리에 항상 유지
  - page fault rate (thrashing) 감소
  - 시스템 성능 향상
- Window size(Δ)는 고정
  - Memory allocation은 가변
    - MA가 고정 and Δ가 가변? ➡️ LRU Algorithm
  - Δ값이 성능을 결정 짓는 중요한 요소

[Window size vs WS size]



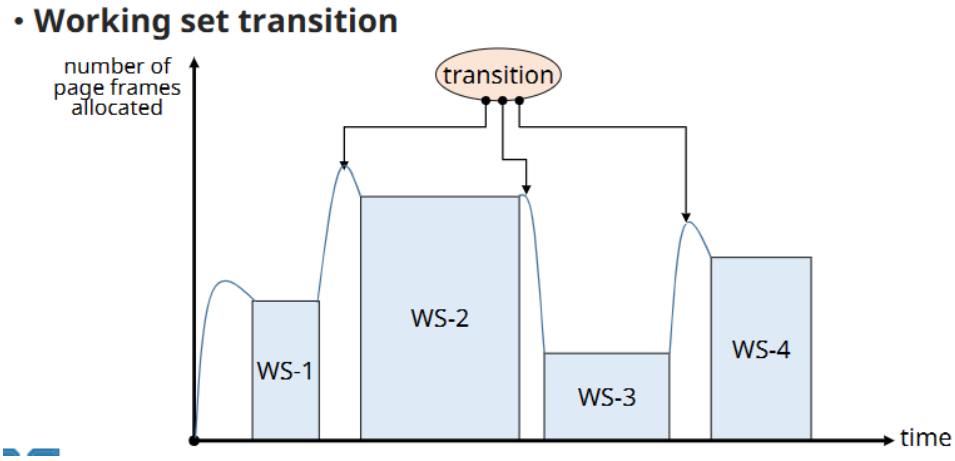
- Window size: Δ (어느 구간을 볼 것인가)
  WS Size: Memory Allocation $M\Delta$(메모리에 유지되는 size)
- locality를 갖고 있어서 초반에는 window size 늘어나면 ws size도 급격하게 증가.
  locality G 벗어나면 더 이상 효과 X

[Example: working set transition]



- loop 돌 때 working set 변화 예측 가능 (ws size: 2 ⇒ 3 ⇒ 2)
- loop 전환할 때는 working set이 증가할 것

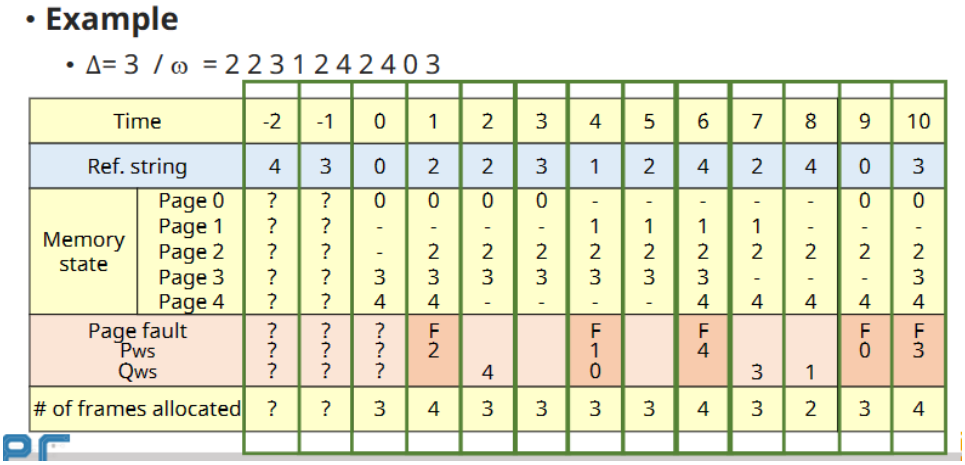
[Working set transition]



- loop에서 loop, ws에서 ws 전환할 때 일시적으로 $M\Delta$가 증가

[Example]

- $\Delta = 3$, number of pages = 5 (0,1,2,3,4)
- Initially pages {0,3,4} in the memory at time 0



Time		-2	-1	0	1	2	3	4	5	6	7	8	9	10
Ref. string		4	3	0	2	2	3	1	2	4	2	4	0	3
Memory state	Page 0	?	?	0	0	0	0	-	-	-	-	-	0	0
	Page 1	?	?	-	-	-	-	1	1	1	1	-	-	-
	Page 2	?	?	-	2	2	2	2	2	2	2	2	-	2
	Page 3	?	?	-	3	3	3	3	3	3	-	-	-	3
	Page 4	?	?	4	4	-	-	-	-	4	4	4	4	4
Page fault Pws Qws		? ? ?	? ? ?	? ? ?	F 2	 4		F 1 0		F 4	 3	 1	F 0	F 3
# of frames allocated		?	?	3	4	3	3	3	3	4	3	2	3	4

[0,2,3,4] ws: 4, [3,0,2] ws: 3, ….

- page fault: 5번 발생 ⇒ 성능이 좋은가?

[성능 평가]

- Page fault 수 외 다른 지표도 함께 봐야 함 (할당하는 page 수가 달라짐)
  - 평균 할당 page frame 수, page fault 비용, page 유지 비용 등
- Example
  - time interval [1,10]
    - # of page fault = 5

        - 평균 할당 page frame 수 = 3.2
        - cost(page fault) = 50, cost(page 유지)=10
        - cost = 50 * 5 + 3.2*10*10 = 570
    - 평가
        - 평균 3.2개의 page frame을 할당 받은 상태에서 5번의 page fault 발생

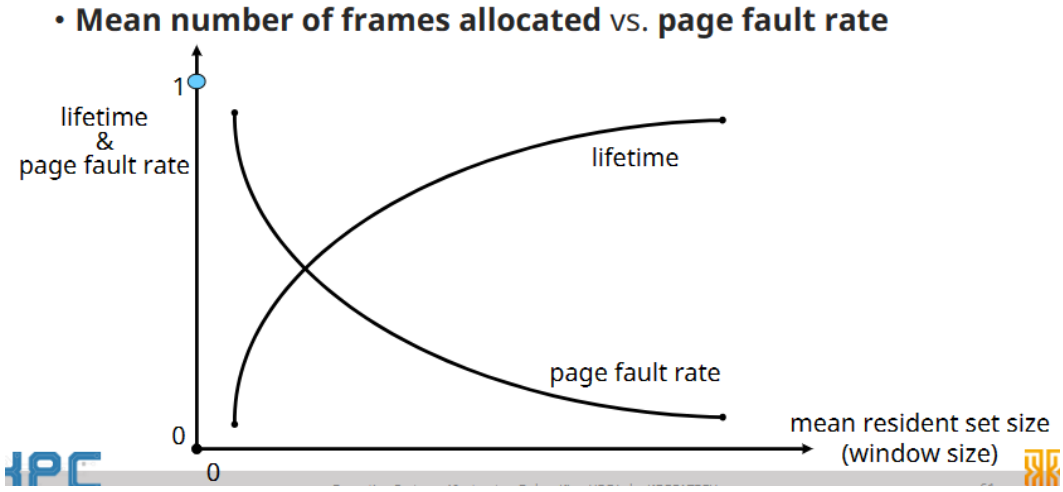
[특성]

- 적재되는 page가 없더라도, 메모리를 반납하는 page가 있을 수 있음
- 새로 적재되는 page가 있을더라도, 교체되는 page가 없을 수 있음

[단점]

- Working set management overhead
    - Residence set(상주 집합)을 page fault가 없더라도, 지속적으로 관리해야 해서 발생하는 오버헤드 존재

[Mean number of frames allocated vs. page fault rate]



- working set 커지면 page의 lifetime 증가, page 유지 비용 증가, page fault rate 감소
    - working set 사이즈를 적당히 유지해야 함

PFF (Page Fault Frequency) Algorithm

- **Residence set size를 page fault rate에 따라 결정**
    - Low page fault rate (long inter-fault time, IFT)
        - Process에게 할당된 page frame 수 감소

    - High page fault rate (short inter-fault time)
        - Process에게 할당된 page frame 수 증가
- **Resident set 갱신 및 메모리 할당**

  ➡️ WS algorithm에서 계속 ws를 유지해야 하는 오버헤드를 개선하기 위해
    - Page fault가 발생 시에만 수행
    - Low overhead
- **Criteria for page fault rate**
    - $IFT > \tau$: low page fault rate
    - $IFT < \tau$: high page fault rate
    - τ: threshold value
        - page fault 사이 시간이 짧은지 긴 지 parameter로 정한 것
        - System parameter

[Algorithm]

1. Page fault 발생 시, IFT (inter-fault time) 계산
    - $IFT = t_c - t_{c-1}$
        - t_{c-1}: time fo previous page fault
        - t_c: time of current page fault
2. $IFT > \tau$ (low page fault rate)
    - Residence set ⬅️ 메모리에서 $(t_{c-1}, t_c]$ 동안 참조된 page들만 유지
        - t_{c-1}에 참조된 page 유지 X
    - 나머지 page들은 메모리에서 내림
        - 메모리 할당 (# of page frames) 유지 or 감소
3. $IFT \leq \tau$ (high page fault rate): 메모리 할당 증가
    - 기존 page들 유지
    - + 현재 참조된 page를 추가 적재
        - 메모리 할당(# of page frames) 증가

[예제]

- $\tau = 2$, number of pages = 5 (0,1,2,3,4)

- intially pages {0,3,4} in the memory at time 0

• **Example**

• τ=2 / ω = 2 2 3 1 2 4 2 4 0 3

Time		-2	-1	0	1	2	3	4	5	6	7	8	9	10
Ref. string		4	3	0	2	2	3	1	2	4	2	4	0	3
Memory state	Page 0	-	-	0	0	0	0	-	-	-	-	-	0	0
	Page 1	-	-	-	-	-	-	1	1	1	1	1	-	-
	Page 2	-	-	-	2	2	2	2	2	2	2	2	2	2
	Page 3		3	3	3	3	3	3	3	3	3	3	-	3
	Page 4	4	4	4	4	4	4	-	-	4	4	4	4	4
Page fault P_{PFF} Q_{PFF}				F 0	F 2			F 1 0,4		F 4			F 0 1,3	F 3
# of frames allocated		-	-	3	4	4	4	3	3	4	4	4	3	4

첫번째 IFT 계산: 4-1=3 > 2, MA 줄임
두번째 IFT 계산: 6-4=2 ≥ 2, MA 증가

[성능 평가]

- Page fault 수 외 다른 지표도 함께 봐야 함
- Example
  - time interval [1,10]
    - # of page fault = 5
    - 평균 할당 page frame 수 = 3.7
  - 평균 3.7개의 page frame을 할당 받은 상태에서 5번의 page fault 발생(WS보다 많은 page fault)

[특징]

- 메모리 상태 변화가 page fault 발생 시에만 발생함
  - low overhead

VMIN (Variable MIN) Algorithm

- Variable allocation 기반 교체 기법 중 optimal algorithm
  - 평균 메모리 할당량과 page fault 발생 횟수 모두 고려했을 때의 optimal
- 실현 불가능한 기법 (unrealizable). 하지만 평가 기준으로 삼을 수 있음
  - Page reference string을 미리 알고 있어야 함

- **기법**
  - $[t, t+\Delta]$을 고려해서 교체할 page 선택

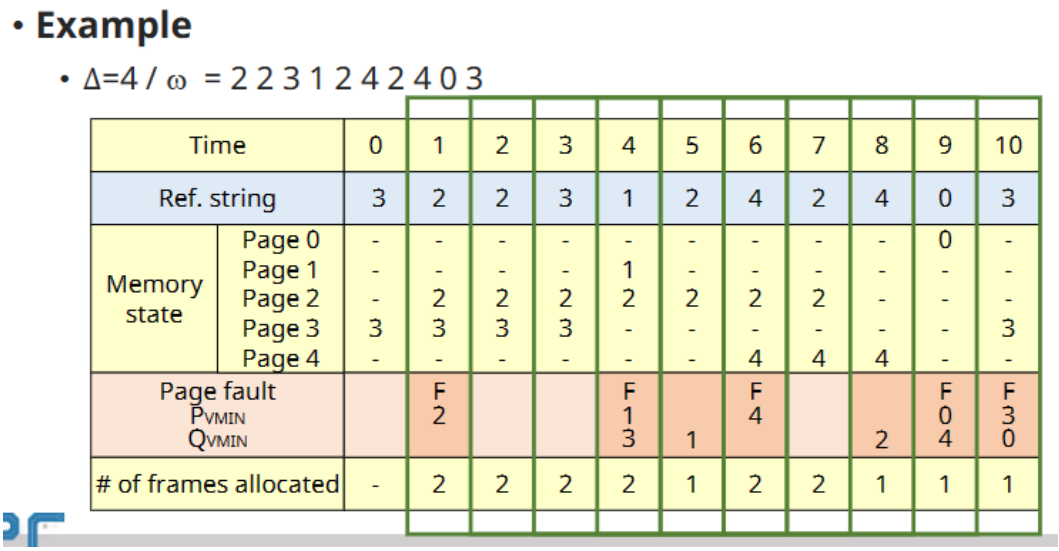
[Algorithm]

Page r이 t시간에 참조되면, page r이 $(t, t+\Delta]$ 사이에 다시 참조되는지 확인

- 참조된다면, page r 유지
- 참조 안 된다면, page r 메모리에서 내림

[Example]

- number of pages = 5 (0,1,2,3,4)



Time		0	1	2	3	4	5	6	7	8	9	10
Ref. string		3	2	2	3	1	2	4	2	4	0	3
Memory state	Page 0	-	-	-	-	-	-	-	-	-	0	-
	Page 1	-	-	-	-	1	-	-	-	-	-	-
	Page 2	-	2	2	2	2	2	2	2	-	-	-
	Page 3	3	3	3	3	-	-	-	-	-	-	3
	Page 4	-	-	-	-	-	-	4	4	4	-	-
Page fault PVMIN QVMIN			F 2			F 1 3	 1	F 4		 2	F 0 4	F 3 0
# of frames allocated		-	2	2	2	2	1	2	2	1	1	1

1~4: 3 보니까 유지, 4~7: 3을 참조하지 않으니까 내림
평균 MA 줄어든 것 볼 수 있음

[성능 평가]

- Page fault 수 외 다른 지료도 함께 봐야함
- Example
  - time interval [1, 10]
    - # of page fault = 5
    - 평균 할당 page frame 수 = 1.6
  - 평가
    - 평균 1.6개의 page frame을 할당 받은 상태에서 5번의 page fault 발생

[최적 성능을 위한 Δ값은?]

최적 성능을 알기 위해서는 비용을 알아야 함!

- $\Delta = \frac{R}{U}$
  - U: 한 번의 참조 시간 동안 page를 메모리에 유지하는 비용
  - R: page fault 발생 시 처리 비용
- $R > \Delta * U$ (Δ가 작으면), 처리 비용 > page 유지 비용
  - Δ를 늘려주는 것이 좋음
- $R < \Delta * U$ (Δ가 크면), page fault 처리 비용 < 유지 비용
  - Δ를 줄이고 page fault를 발생시키는 게 나음

Other Considerations

- Page size
- Program restructuring
- TLB reach

Page Size

- 시스템 특성에 따라 다름
  - No best answer!
  - 점점 커지는 경향
- 일반적인 page size
  - $2^7(128)\text{bytes} \sim 2^{22}(4M)\text{bytes}$

Small Page Size	Large Page Size
large page table / # of PF	small page table / # of PF
high overhead (kernel)	low overhead (kernel)
내부 단편화 감소	내부 단편화 증가
I/O 시간 증가	I/O 시간 감소
locality 향상	locality 저하
page fault 증가	page fault 감소

[HW 발전 경향]

- CPU ⬆️, Memory Size ⬆️

    - 상대적인 page fault 처리 비용 ⬆️

Program Restructuring

- 가상 메모리 시스템의 특성에 맞도록 프로그램을 재구성
- 사용자가 가상 메모리 관리 기법(예. paging system)에 대해 이해하고 있다면, 프로그램의 구조를 변경하여 성능을 높일 수 있음