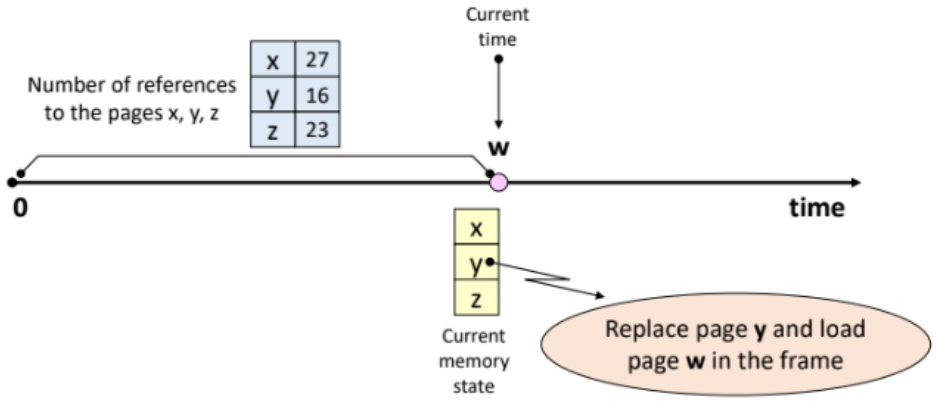
8주차 Virtual Memory Management pt.2

LFU (Least Frequently Used) Algorithm

- 가장 참조 횟수가 적은 Page를 교체
  - Tie-breaking rule : LRU
- Page 참조 시 마다, 참조 횟수를 누적 시켜야 함
- Locality 활용
  - LRU 대비 적은 overhead ⇒ LRU는 시간을 기록해야 하지만 LFU는 (참조 횟수++)만 해주면 됨
- 단점
  - 최근 적재된 참조될 가능성이 높은 page가 교체 될 가능성이 있음
  - 참조 횟수 누적 overhead



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

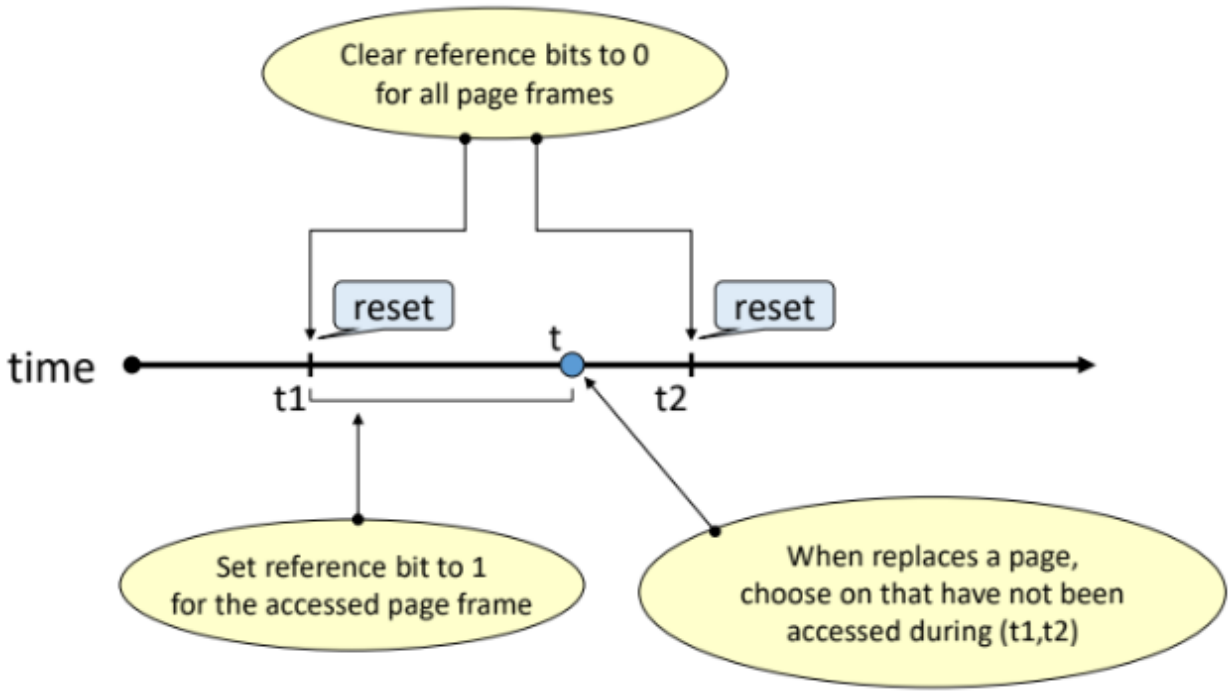
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ref. string	1	2	6	1	4	5	1	2	1	4	5	6	4	5
Memory state	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		2	2	2	2	5	5	5	5	5	5	5	5	5
			6	6	6	6	6	2	2	2	2	6	6	6
					4	4	4	4	4	4	4	4	4	4
Page fault	F	F	F		F	F		F				F		

  - **Number of page faults** = 7

NUR (Not Used Recently) Algorithm

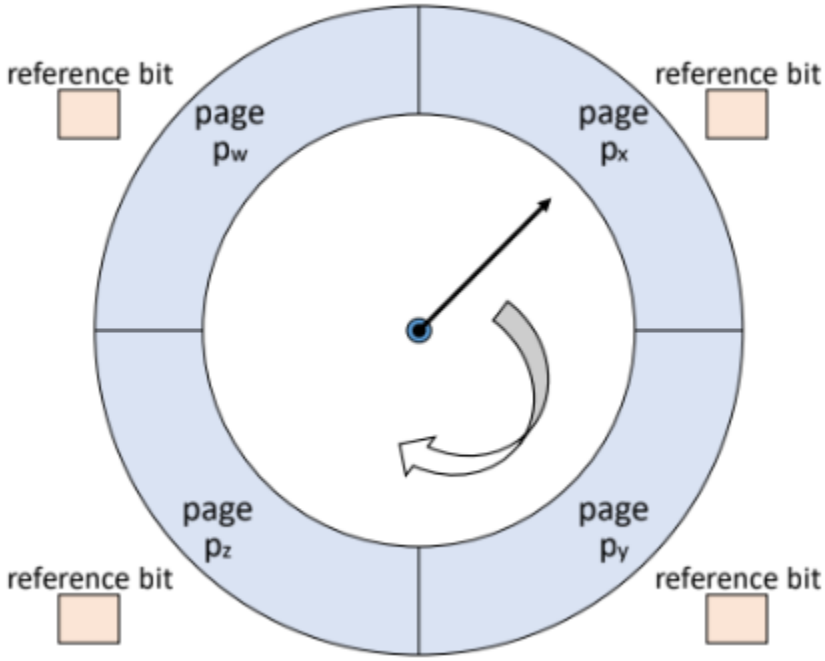
- 최근에 사용되지 않은 Page 교체
- LRU approximation scheme
  - LRU보다 적은 overhead로 비슷한 성능 달성 목적
- Bit vector 사용
  - Reference bit vector (r), Update bit vector (m)
- 교체 순서
  - 최근 참조되지 않은 것부터
  - update bit가 1이면 메모리에서 나올 때 0으로 바꿔주는 write-back을 해줘야 하므로 이걸 피하기 위해 update bit가 0인 것부터 교체

1. (r, m) = (0, 0)
2. (r, m) = (0, 1)
3. (r, m) = (1, 0)
4. (r, m) = (1, 1)



Clock Algorithm

- NUR이 이론적이라면 Clock Algorithm은 그걸 실제로 적용한 것
- IBM VM/370 OS
- **Reference bit 사용함**
  - 주기적인 초기화 없음
  - 시침이 지나가는 시점이 초기화 시점
- Page frame 들을 순차적으로 가리키는 pointer(시계바늘)를 사용하여 교체될 page 결정



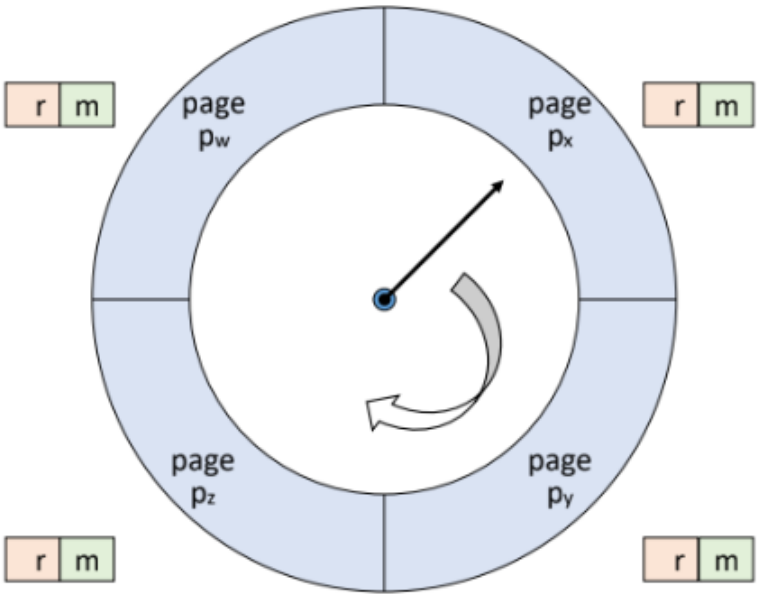
- Pointer를 돌리면서 교체 page 결정
  - 현재 가리키고 있는 page의 reference bit(r) 확인
  - r = 0 인 경우, 교체 page로 결정
  - r = 1 인 경우, reference bit 초기화 후 pointer 이동
- 먼저 적재된 page가 교체될 가능성이 높음
  - FIFO와 유사
- Reference bit를 사용하여 교체 페이지 결정
  - LRU (or NUR)과 유사
  - Locality 반영

- **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



- Example
  - 4 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/**11**	→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/**11**	b/**00**	b/**10**	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

Other Algorithms

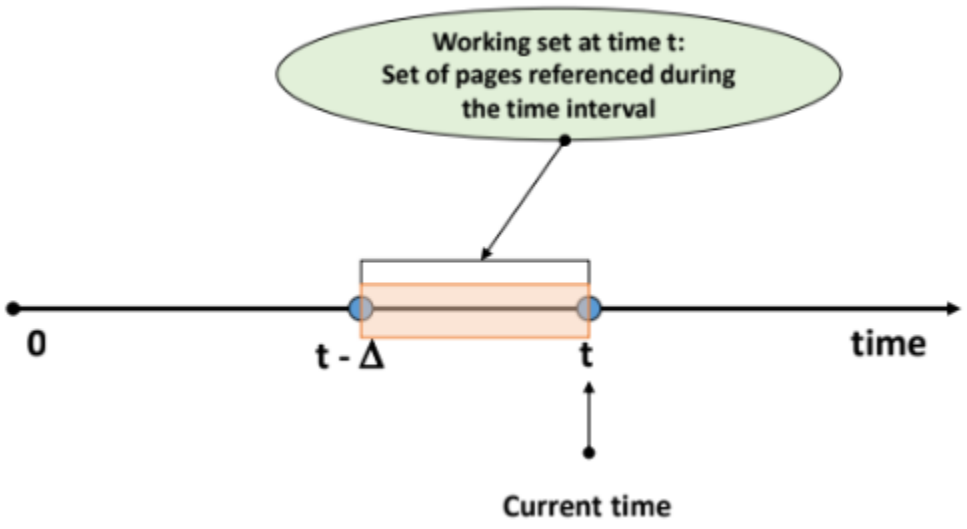
- Additional-reference-bits algorithm
  - LRU approximation
  - 여러 개의 reference bit를 가짐
    - 각 time-interval에 대한 참조 여부 기록
    - History register for each page
- MRU (Most Recently Used) algorithm

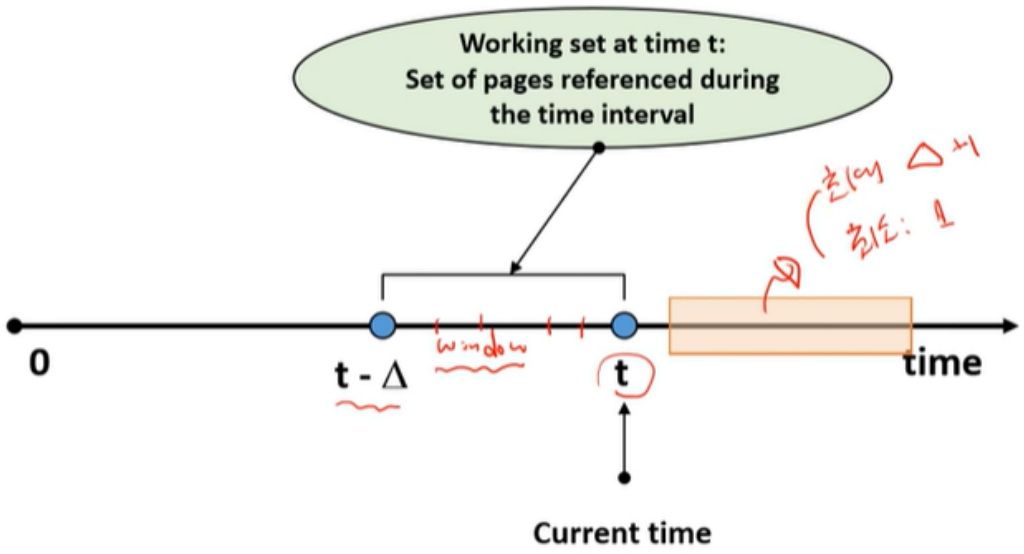
    - LRU와 정반대 기법
- MFU (Most Frequently Used) algorithm
    - LFU와 정반대 기법

Variable allocation

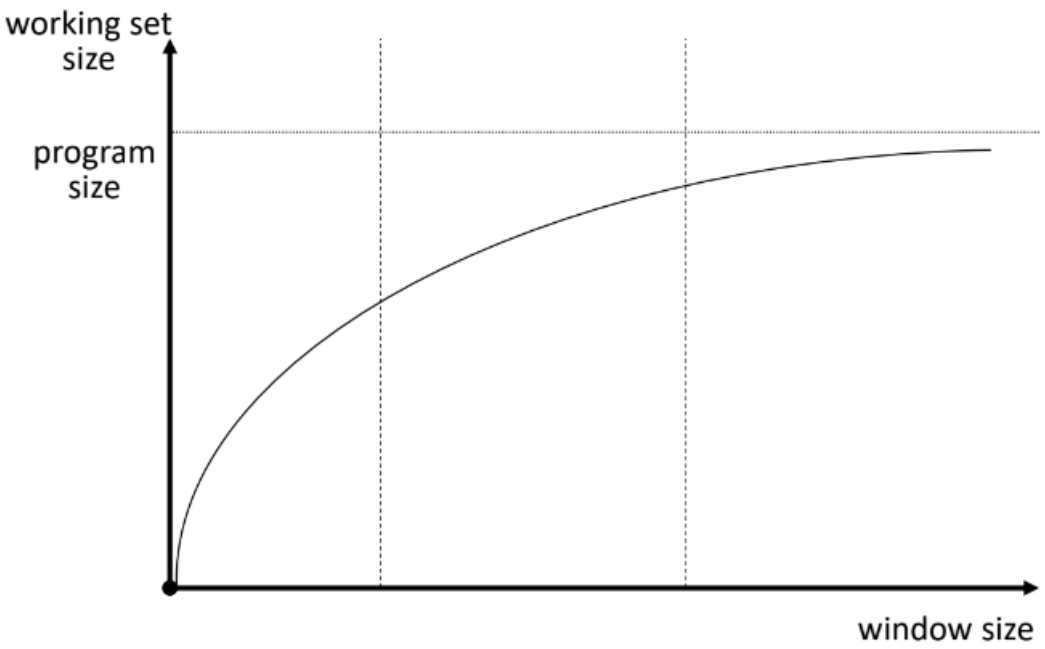
Working Set (WS) algorithm

- 1968년 Denning이 제안
- Working set
    - Process가 특정 시점에 자주 참조하는 page들의 집합 ⇒ locality 반영됨
    - 최근 일정시간 동안(Δ) 참조된 page들의 집합
    - 시간에 따라 변함
    - W(t, Δ)
        - The working set of a process at time t
        - Time interval [t- Δ, t] 동안 참조된 pages들의 집합
        - Δ: window size, system parameter



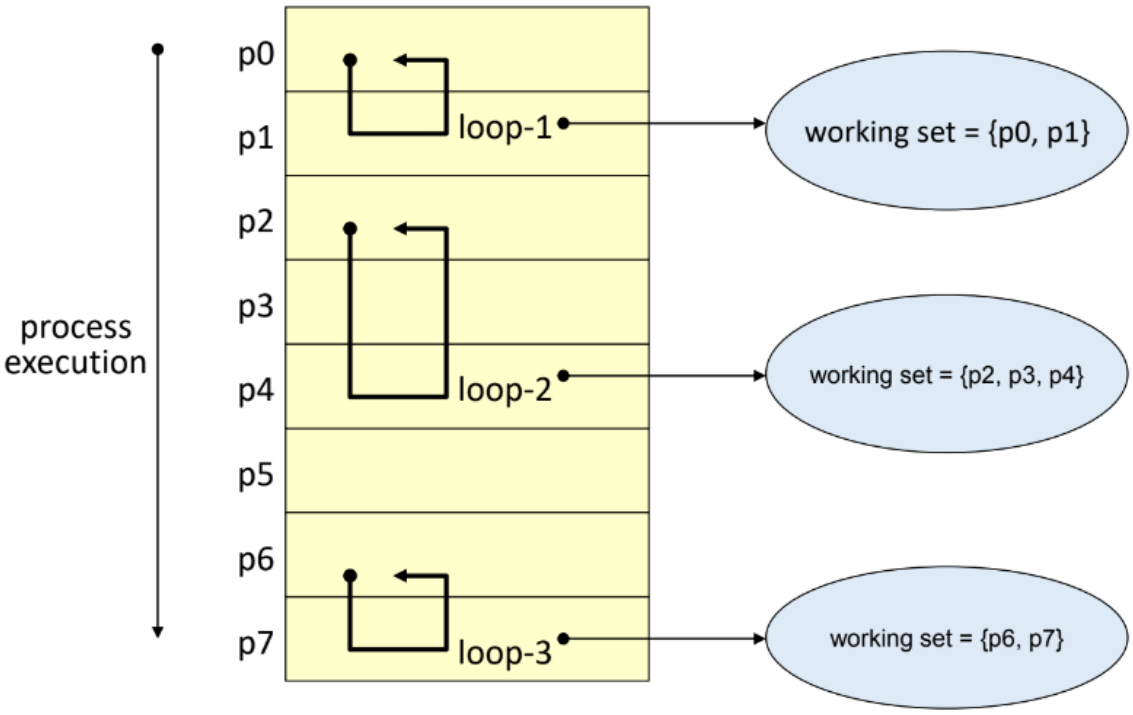


- Working set memory management
  - Locality에 기반을 둠
  - Working set을 메모리에 항상 유지
    - Page fault rate (thrashing) 감소
    - 시스템 성능 향상
  - Window size(Δ)는 고정
    - Memory allocation은 가변
      - MA가 고정 and Δ가 가변 = LRU
    - Δ 값이 성능을 결정 짓는 중요한 요소
- Window size vs. WS size

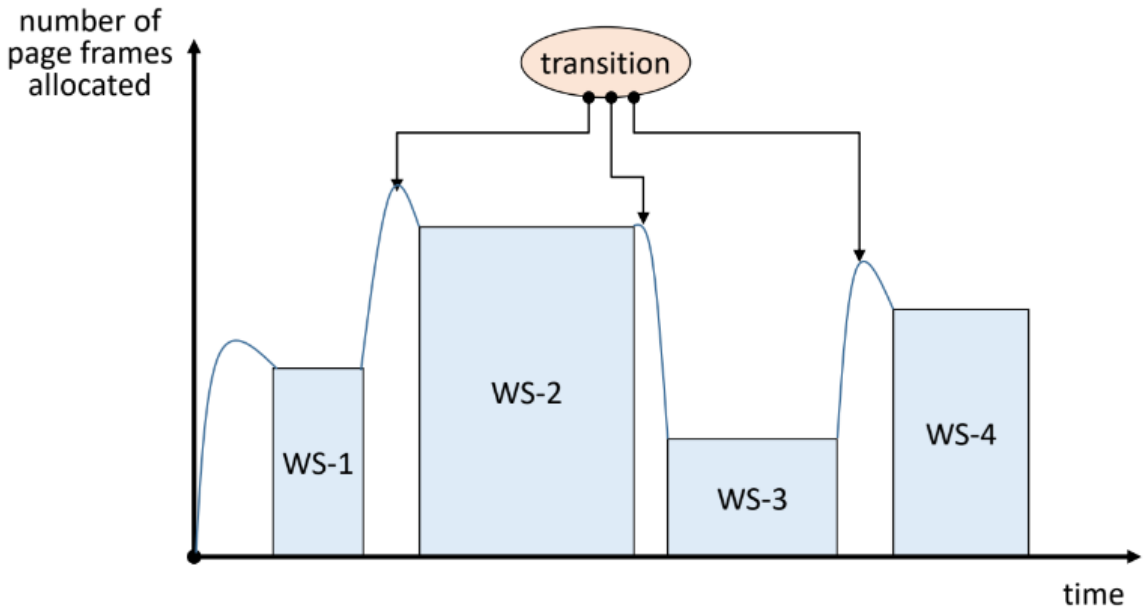


- 최대 WS size는 program size와 유사
- locality 때문에 초반엔 window size를 조금만 늘려도 WS size가 확 늘어남
- 그러나 locality 를 벗어나게 늘리면 증가폭이 감소함

- Example : working set transition



- Working set transition



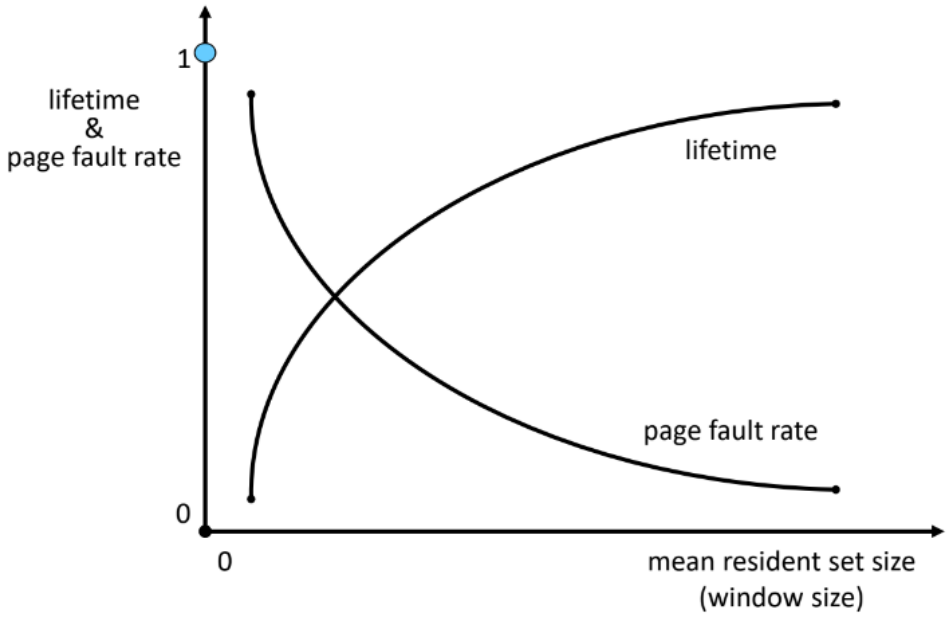
  - loop-1에서 loop-2로 전환될 때 WS size : loop-1에 해당하는 부분만 보다가 전환하는 시점에는 loop-1과 loop-2에 해당하는 부분을 모두 봐야하기 때문에 일시적으로 증가하는 현상 발생

- Example

  - Δ= 3, number of pages = 5 (0, 1, 2, 3, 4)
  - Initially pages {0, 3, 4} in the memory at time 0
  - 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	-	-	-
	Page 2	?	?	-	2	2	2	2	2	2	2	2	2	2
	Page 3	?	?	3	3	3	3	3	3	3	-	-	-	3
	Page 4	?	?	4	4	-	-	-	-	4	4	4	4	4
Page fault		?	?	?	F			F		F			F	F
Pws		?	?	?	2			1		4			0	3
Qws		?	?	?		4		0			3	1		
# of frames allocated		?	?	3	4	3	3	3	3	4	3	2	3	4

    - # of frames allocated = WS size
- 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 = 3.2
        - 평가
            - 평균 3.2개의 page frame을 할당 받은 상태에서 5번의 page fault 발생
            - page fault 처리 비용, page frame 유지 비용 등을 알아야 평가 가능
- 특성
    - 적재되는 page가 없더라도, 메모리를 반납하는 page가 있을 수 있음
    - 새로 적재되는 page가 있더라도, 교체되는 page가 없을 수 있음
- 단점
    - Working set management overhead
    - Residence set (상주 집합)을 Page fault가 없더라도 지속적으로 관리해야 함
- Page Fault Frequency (PFF) algorithm
- Mean number of frames allocated vs. page fault rate



- lifetime이 늘어남 ⇒ page 유지 비용이 늘어남

Page Fault Frequency (PFF) algorithm

- Working Set 알고리즘은 page fault가 일어나지 않아도 계속 지켜봐야하는 오버헤드가 있다
  ⇒ Page fault가 발생하면 관리
- Residence set size를 page fault rate에 따라 결정 ⇒ page fault가 얼마나 자주 일어나는지에 따라 메모리에 몇 개를 유지할 지 결정
  - Low page fault rate (long inter-fault time)
    - Process에게 할당된 PF 수를 감소
  - High page fault rate (short inter-fault time)
    - Process에게 할당된 PF 수를 증가
- Resident set 갱신 및 메모리 할당
  - Page fault가 발생 시에만 수행
  - Low overhead
- Criteria for page fault rate
  - IFT(inter-fault time) > τ : Low page fault rate

- IFT < τ : High page fault rate
- τ : threshold value,
  - 기준점, 이것보다 inter-fault time(page fault가 발생하고 다음 page fault가 발생할 때 까지의 시간)이 길면 page fault rate가 낮다.
  - System parameter

- Algorithm

1. Page fault 발생 시, IFT 계산

   - IFT = $t_c - t_{c-1}$
     - t_{c-1} : time of previous page fault
     - t_c : time of current page fault

2. IFT > τ (Low page fault rate)

   - Residence set ← $(t_{c-1}, t_c]$ 동안 참조 된 page들 만 유지
   - 나머지 page들은 메모리에서 내림
     - 메모리 할당(#of page frames) 유지 or 감소

3. IFT ≤ τ (High page fault rate)

   - 기존 pages들 유지
   - + 현재 참조된 page를 추가 적재
     - 메모리 할당(# of page frames) 증가

- Example
  - τ=2, number of pages = 5 (0, 1, 2, 3, 4)
  - Initially pages {0, 3, 4} in the memory at time 0
  - 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_{FF} Q_{FF}				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

- 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
    - 평가
      - 평균 3.7개의 page frame을 할당 받은 상태에서 5번의 page fault 발생
- 평가
  - 메모리 상태 변화가 page fault 발생 시에만 변함
    - Low overhead

Variable MIN (VMIN) algorithm

- Variable allocation 기반 교체 기법 중 optimal algorithm
  - 평균 메모리 할당량과 page fault 발생 횟수 모두 고려했을 때의 Optimal
- 실현 불가능한 기법 (Unrealizable)
  - Page reference string을 미리 알고 있어야 함
- 기법

- [t,t + Δ] 을 고려해서 교체할 page 선택

- Algorithm
  - Page r이 t 시간에 참조되면, page r이 (t,t + Δ] 사이에 다시 참조되는지 확인
  - 참조된다면 page r을 유지
  - 참조되지 않으면 page r을 메모리에서 내림
- Example
  - Δ=4, number of pages = 5 (0, 1, 2, 3, 4)
  - ω = 2 2 3 1 2 4 2 4 0 3

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 P_{VMIN} Q_{VMIN}			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

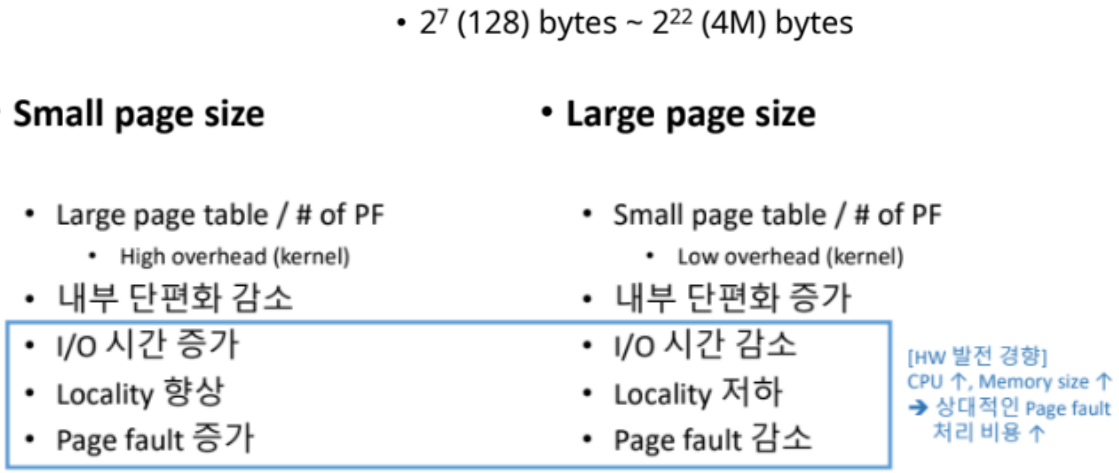
- 성능 평가
  - Page fault 수 외 다른 지표도 함께 봐야 함
  - Example
    - Time interval [1, 10]
      - # of page fault = 5
      - 평균 할당 page frame 수 = 1.6
    - 평가
      - 평균 1.6개의 page frame을 할당 받은 상태에서 5번의 page fault 발생
- 최적 성능을 위한 Δ값은?

- Δ = R / U
  - U : 한 번의 참조 시간 동안 page를 메모리에 유지하는 비용
  - R : page fault 발생 시 처리 비용
- R > Δ * U, (Δ 가 작으면)
  - 처리 비용 > page 유지비용
- R < Δ * U, (Δ 가 크면)
  - 처리 비용 < page 유지비용

Other Considerations

Page Size

- 시스템 특성에 따라 다름
  - No best answer!
  - 점점 커지는 경향
- 일반적인 page size



Program Restructuring

- 가상 메모리 시스템의 특성에 맞도록 프로그램을 재구성

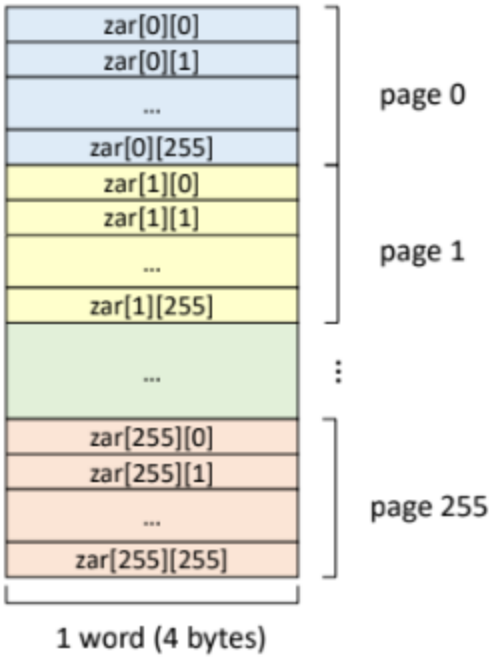
- 사용자가 가상 메모리 관리 기법 (ex. paging system)에 대해 이해하고 있다면, 프로그램의 구조를 변경하여 성능을 높일 수 있음
- Example
  - Paging system, Page size = 1KB
  - sizeof(int) = 4 bytes

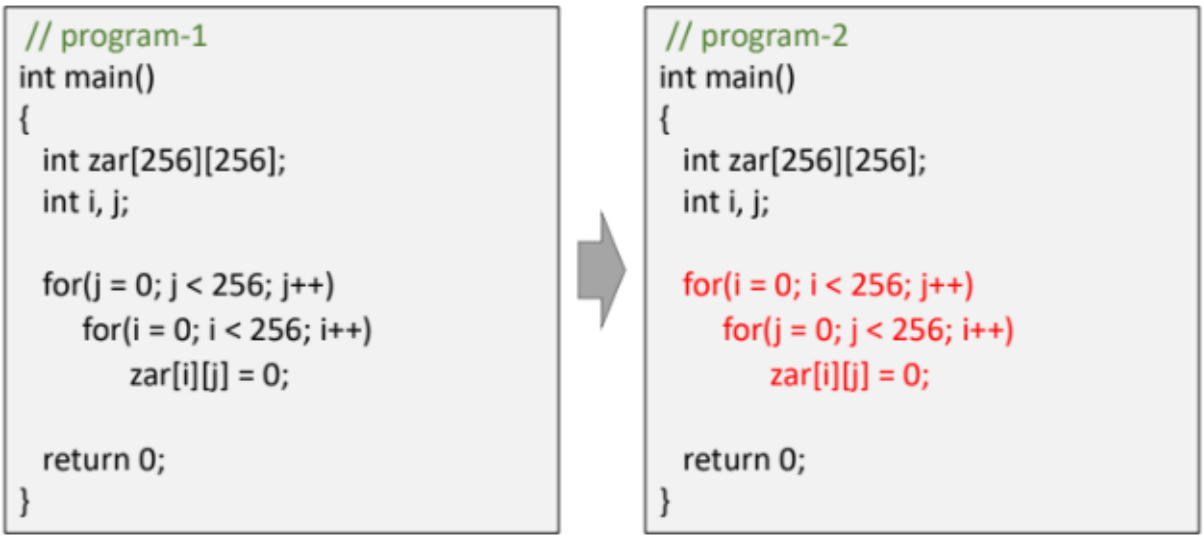
```
// program-1
int main()
{
  int zar[256][256];
  int i, j;

  for(j = 0; j < 256; j++)
      for(i = 0; i < 256; i++)
          zar[i][j] = 0;

  return 0;
}
```

- Row-major order for array





```
// program-1
int main()
{
  int zar[256][256];
  int i, j;

  for(j = 0; j < 256; j++)
      for(i = 0; i < 256; i++)
          zar[i][j] = 0;

  return 0;
}
```

```
// program-2
int main()
{
  int zar[256][256];
  int i, j;

  for(i = 0; i < 256; j++)
      for(j = 0; j < 256; i++)
          zar[i][j] = 0;

  return 0;
}
```

- program-2처럼 하면 page fault를 줄일 수 있음

TLB Reach

- TLB (Translation Lookaside Buffer) : 가상 메모리 주소를 물리적인 주소로 변환하는 속도를 높이기 위해 사용되는 캐시
- TLB를 통해 접근할 수 있는 메모리의 양
  - (The number of entries) * (the page size)
- TLB의 hit ratio를 높이려면
  - TLB의 크기 증가 ⇒ Expensive
  - Page 크기 증가 or 다양한 page size 지원
    - OS의 지원이 필요
      - 최근 OS의 발전 경향