Chapter 8-2

가상 메모리 관리

Virtual memory management

Virtual Memory Management

- 가상 메모리 (기억장치)
  - Non-continuous allocation
    - 사용자 프로그램을 block으로 분할하여 적재/실행
  - Paging/Segmentation system

- 가상 메모리 관리의 목적
  - 가상 메모리 시스템 성능 최적화
    - Cost model
    - 다양한 최적화 기법

Cost Model for Virtual Mem. Sys.

- Page fault frequency (발생 빈도)
- Page fault rate (발생률)

- Page fault rate를 최소화 할 수 있도록 전략들을 설계해야 함
  - Context switch 및 Kernel 개입을 최소화
  - 시스템 성능 향상

Cost Model for Virtual Mem. Sys.

- Page reference string (d)
  - 프로세스의 수행 중 참조한 페이지 번호 순서
  - $\omega = r_1 r_2 \cdots r_k \cdots r_T$
    - r_i = 페이지 번호, $r_i \in \{0, 1, 2, \cdots, N-1\}$
    - N : 프로세스의 페이지 수 (0 ~ N-1)

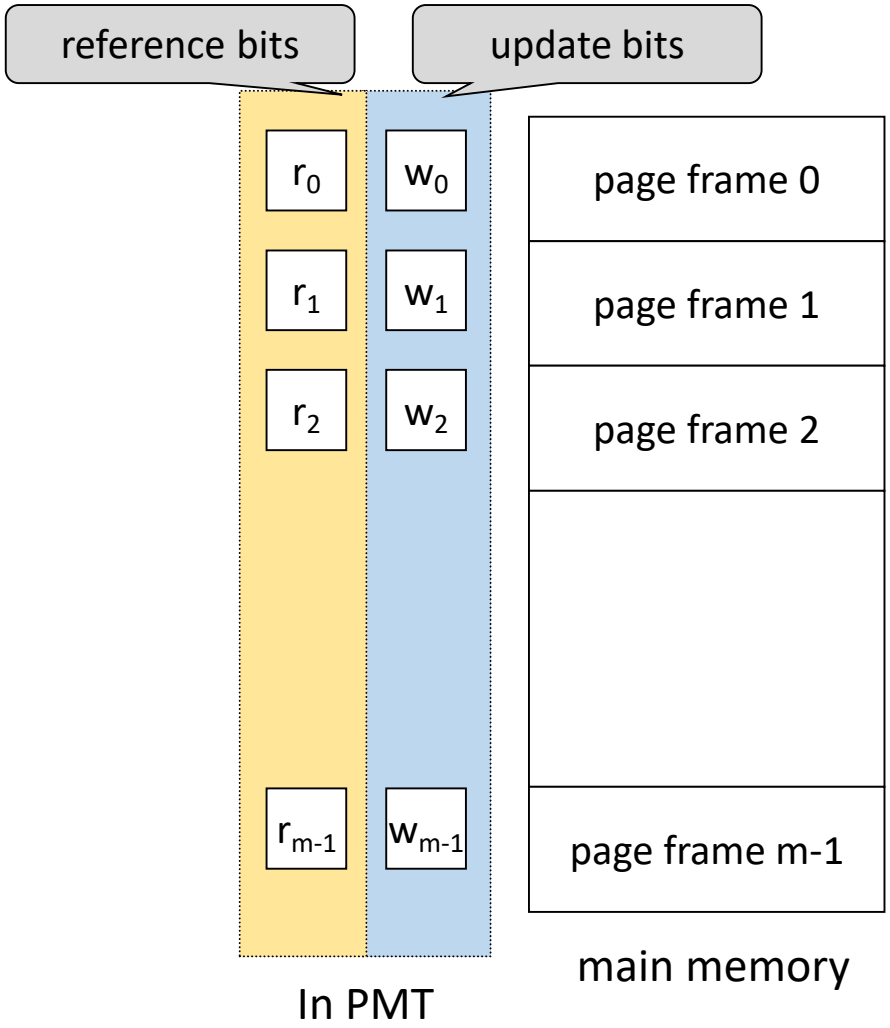
- Page fault rate = F(ω)
  - $F(\omega) = \frac{Num.of\ page\ faults\ during\ \omega}{|\omega|}$

Hardware Components

- Address translation device (주소 사상 장치)
  - 주소 사상을 효율적으로 수행하기 위해 사용
    - E.g., TLB (associated memories), Dedicated page-table register,
      Cache memories

- Bit Vectors
  - Page 사용 상황에 대한 정보를 기록하는 비트들
  - **Reference bits (used bit)**
    - 참조 비트
  - **Update bits (modified bits, write bits, dirty bits)**
    - 갱신 비트

Bit Vectors

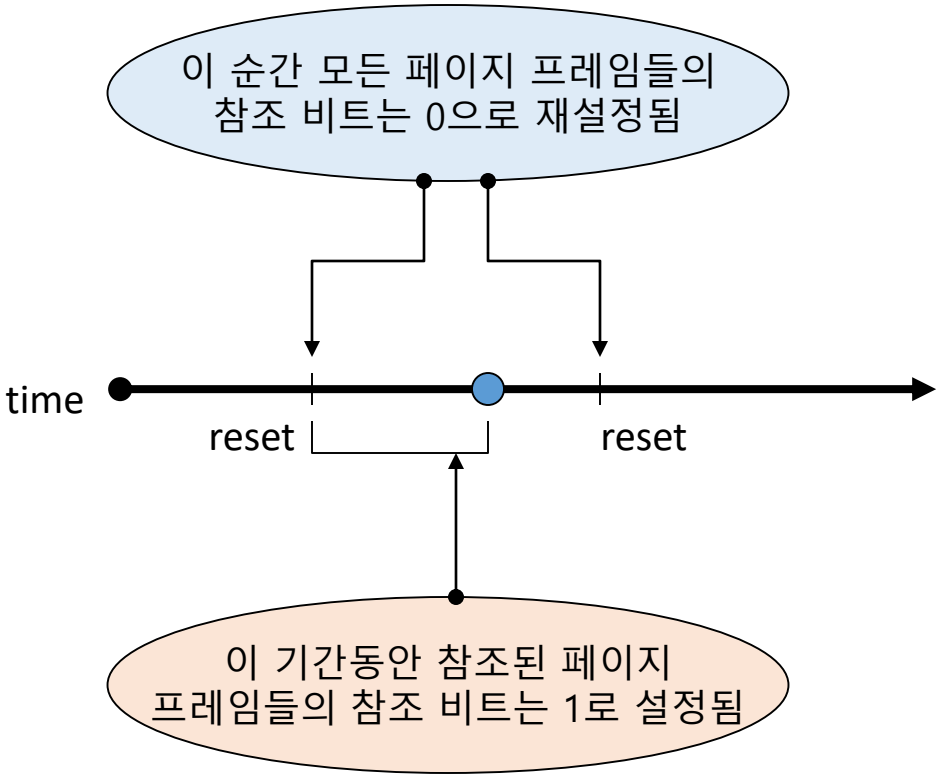

reference bits
update bits
r_0
w_0
r_1
w_1
r_2
w_2
r_{m-1}
w_{m-1}
page frame 0
page frame 1
page frame 2
page frame m-1
In PMT
main memory

Bit Vectors

- **Reference bit vector**
  - 메모리에 적재된 각각의 page가 최근에 참조 되었는지를 표시
  - 운영
    ① 프로세스에 의해 참조되면 해당 page의 Ref. bit를 1로 설정
    ② 주기적으로 모든 reference bit를 0으로 초기화
  - Reference bit를 확인함으로서 최근에 참조된 page들을 확인 가능

Bit Vectors

- Reference bit vector



Bit Vectors

- Update bit vector
  - Page가 메모리에 적재된 후, 프로세스에 의해 수정 되었는지를 표시
  - 주기적 초기화 없음
  - Update bit = 1
    - 해당 page의 (Main memory 상 내용) ≠ (Swap device의 내용)
    - 해당 page에 대한 **Write-back (to swap device)**이 필요

Outline

- Virtual memory management
- Cost model
- Hardware components
- Software components
- Page replacement schemes
  - FA-based schemes
  - VA-based schemes
- Other considerations

Software Components

- 가상 메모리 성능 향상을 위한 관리 기법들
  - Allocation strategies (할당 기법)
  - Fetch strategies
  - Placement strategies (배치 기법)
  - Replacement strategies (교체 기법)
  - Cleaning strategies (정리 기법)
  - Load control strategies (부하 조절 기법)

Allocation Strategies

- 각 프로세스에게 메모리를 얼마 만큼 줄 것인가?
  - **Fixed allocation (고정 할당)**
    - 프로세스의 실행 동안 고정된 크기의 메모리 할당
  - **Variable allocation (가변 할당)**
    - 프로세스의 실행 동한 할당하는 메모리의 크기가 유동적

- 고려사항
  - 프로세스 실행에 필요한 메모리 양을 예측해야 함
  - **너무 큰 메모리 할당 (Too much allocation),**
    - 메모리가 낭비 됨
  - **너무 적은 메모리 할당 (Too small allocation),**
    - Page fault rate ↑
    - 시스템 성능 저하

Fetch Strategies

- **특정 page를 메모리에 언제 적재할 것인가?**
  - **Demand fetch (demand paging)**
    - 프로세스가 참조하는 페이지들만 적재
    - Page fault overhead
  - **Anticipatory fetch (pre-paging)**
    - 참조될 가능이 높은 page 예측
    - 가까운 미래에 참조될 가능성이 높은 page를 미리 적재
    - 예측 성공 시, page fault overhead가 없음
    - Prediction overhead (Kernel의 개입), Hit ratio에 민감함

- **실제 대부분의 시스템은 Demand fetch 기법 사용**
  - 일발적으로 준수한 성능을 보여 줌
  - Anticipatory fetch
    - Prediction overhead, 잘못된 예측 시 자원 낭비가 큼

Placement Strategies

- Page/segment를 어디에 적재할 것인가?
- Paging system에는 불필요

- Segmentation system에서의 배치 기법
  - First-fit
  - Best-fit
  - Worst-fit
  - Next-fit

Replacement Strategies

- 새로운 page를 어떤 page와 교체 할 것인가? (빈 page frame이 없는 경우)
  - Fixed allocation을 위한 교체 기법
    - MIN(OPT, B0) algorithm
    - Random algorithm
    - FIFO(First In First Out) algorithm
    - LRU(Least Recently Used) algorithm
    - LFU(Least Frequently Used) algorithm
    - NUR(Not Used Recently) algorithm
    - Clock algorithm
    - Second chance algorithm
  - Variable allocation을 위한 교체 기법
    - VMIN(Variable MIN) algorithm
    - WS(Working Set) algorithm
    - PFF(Page Fault Frequency) algorithm

Cleaning Strategies

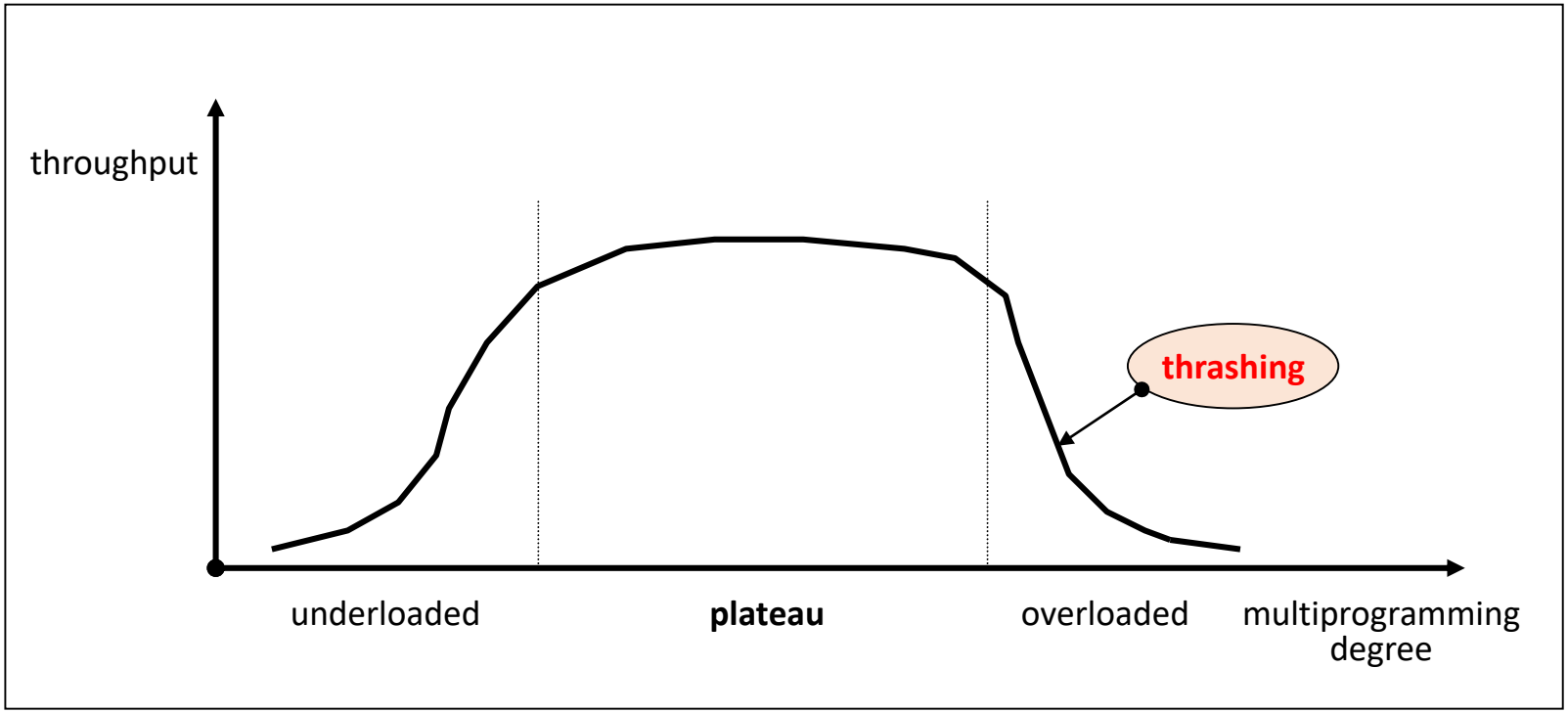
- **변경 된 page를 <u>언제 write-back 할 것인가</u>?**
  - 변경된 내용을 swap device에 반영
  - **Demand cleaning**
    - 해당 page에 메모리에서 내려올 때 write-back
  - **Anticipatory cleaning (pre-cleaning)**
    - 더 이상 변경될 가능성이 없다고 판단 할 때, 미리 write-back
    - Page 교체 시 발생하는 write-back 시간 절약
    - Write-back 이후, page 내용이 수정되면, overhead!

- **실제 대부분의 시스템은 Demand cleaning 기법 사용**
  - 일발적으로 준수한 성능을 보여 줌
  - Anticipatory cleaning
    - Prediction overhead, 잘못된 예측 시 자원 낭비가 큼

Load Control Strategies

- 시스템의 multi-programming degree 조절
  - Allocation strategies와 연계 됨
- 적정 수준의 multi-programming degree를 유지 해야 함
  - Plateau(고원) 영역으로 유지
  - 저부하 상태 (Under-loaded),
    - 시스템 자원 낭비, 성능 저하
  - 고부하 상태 (Over-loaded),
    - 자원에 대한 경쟁 심화, 성능 저하
    - **Thrashing(스레싱)** 현상 발생
      - 과도한 page fault가 발생하는 형상

Load Control Strategies



Software Components

- 가상 메모리 성능 향상을 위한 관리 기법들
  - **Allocation strategies (할당 기법)**
  - Fetch strategies (페치 기법)
  - Placement strategies (배치 기법)
  - **Replacement strategies (교체 기법)**
  - Cleaning strategies (정리 기법)
  - Load control strategies (부하 조절 기법)

Locality

- 프로세스가 프로그램/데이터의 특정 영역을 집중적으로 참조하는 현상
- 원인
  - Loop structure in program
  - Array, structure 등의 데이터 구조

- 공간적 지역성 (Spatial locality)
  - 참조한 영역과 인접한 영역을 참조하는 특성

- 시간적 지역성 (Temporal locality)
  - 한 번 참조한 영역을 곧 다시 참조하는 특성

Locality (Example)

- 가정
  - Paging system
  - Page size = 1000 words
  - Machine instruction size = 1 word
  - 주소 지정은 word 단위로 이루어짐
  - 프로그램은 4번 page에 continuous allocation 됨
  - n = 1000

```
...
for ← 1 to n do
    A[i] ← B[i] + C[i];
endfor
...
```

Locality (Example)

Memory address	Machine code
4000	(R1) ← ONE
4001	(R2) ← n
4002	compare R1, R2
4003	branch greater 4009
4004	(R3) ← B(R1)
4005	(R3) ← (R3) + C(R1)
4006	A(R1) ← (R3)
4007	(R1) ← (R1) + 1
4008	branch 4002
...	...
6000-6999	storage for array A
7000-7999	storage for array B
8000-8999	storage for array C
9000	storage for ONE
9001	storage for n

```
...
for ← 1 to n do
    A[i] ← B[i] + C[i];
endfor
...
```

- $\omega = 494944(47484644)^{1000}$
- 11000번의 메모리 참조 중 5개의 page만을 집중적으로 접근하게 됨

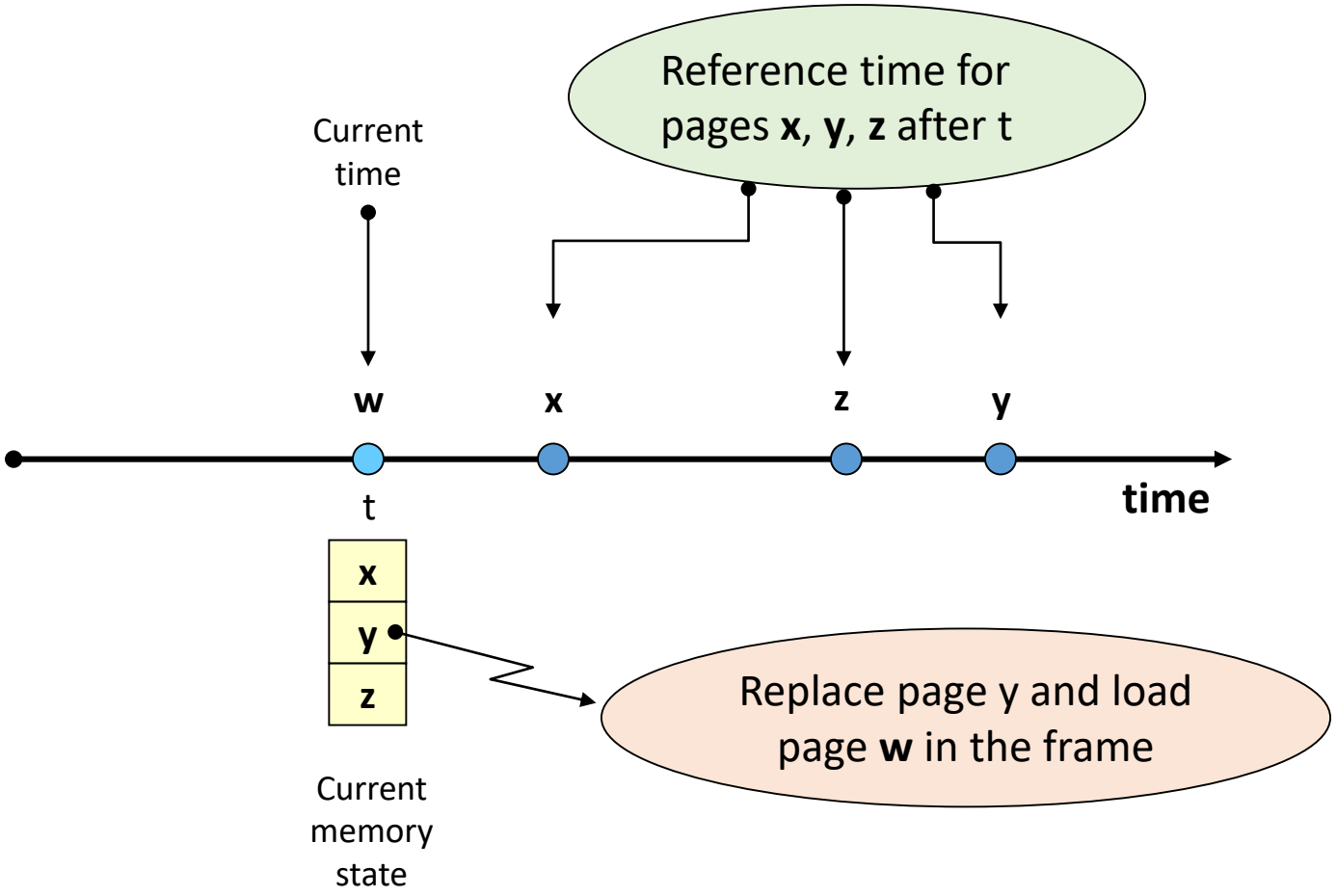
Replacement Strategies

- **Fixed allocation**
  - MIN(OPT, B0) algorithm
  - Random algorithm
  - FIFO(First In First Out) algorithm
  - LRU(Least Recently Used) algorithm
  - LFU(Least Frequently Used) algorithm
  - NUR(Not Used Recently) algorithm
  - Clock algorithm
  - Second chance algorithm
- **Variable allocation**
  - WS(Working Set) algorithm
  - PFF(Page Fault Frequency) algorithm
  - VMIN(Variable MIN) algorithm

Min Algorithm (OPT algorithm)

- 1966년 Belady에 의해 제시
- Minimize page fault frequency (proved)
  - Optimal solution
- 기법
  - 앞으로 가장 오랫동안 참조되지 않을 page 교체
    - Tie-breaking rule : page 번호가 가장 큰/작은 페이지 교체
- 실현 불가능한 기법 (Unrealizable)
  - Page reference string을 미리 알고 있어야 함
- 교체 기법의 성능 평가 도구로 사용 됨

Min Algorithm (OPT algorithm)



Min Algorithm (OPT algorithm)

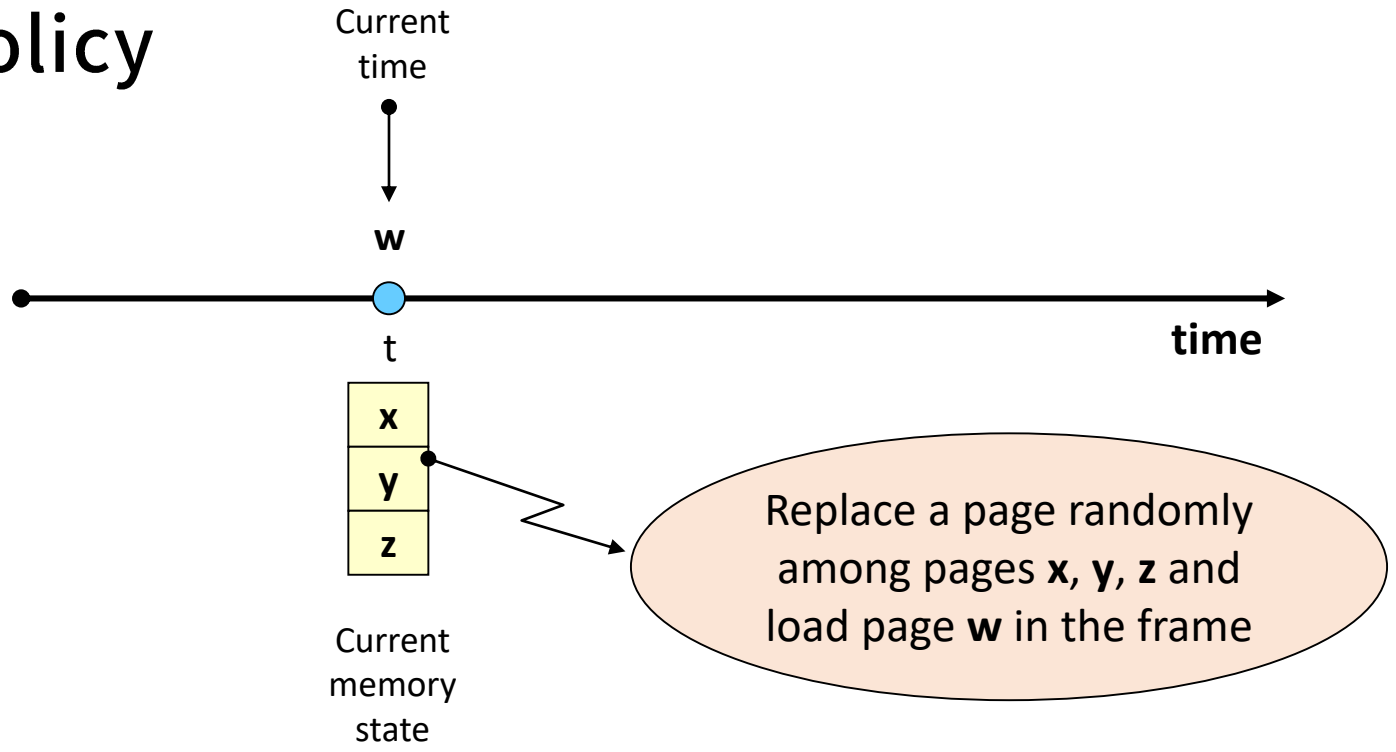
- Example
  - 4 page frames for the process, initially empty
  - ω = 1 2 6 1 4 5 1 2 1 4 5 6 4 5

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	6	6	6
		2	2	2	2	2	2	2	2	2	2	2	2	2
			6	6	6	5	5	5	5	5	5	5	5	5
					4	4	4	4	4	4	4	4	4	4
Page fault	F	F	F		F	F						F		

- **Number of page faults =**

Random Algorithm

- 무작위로 교체할 page 선택
- Low overhead
- No policy

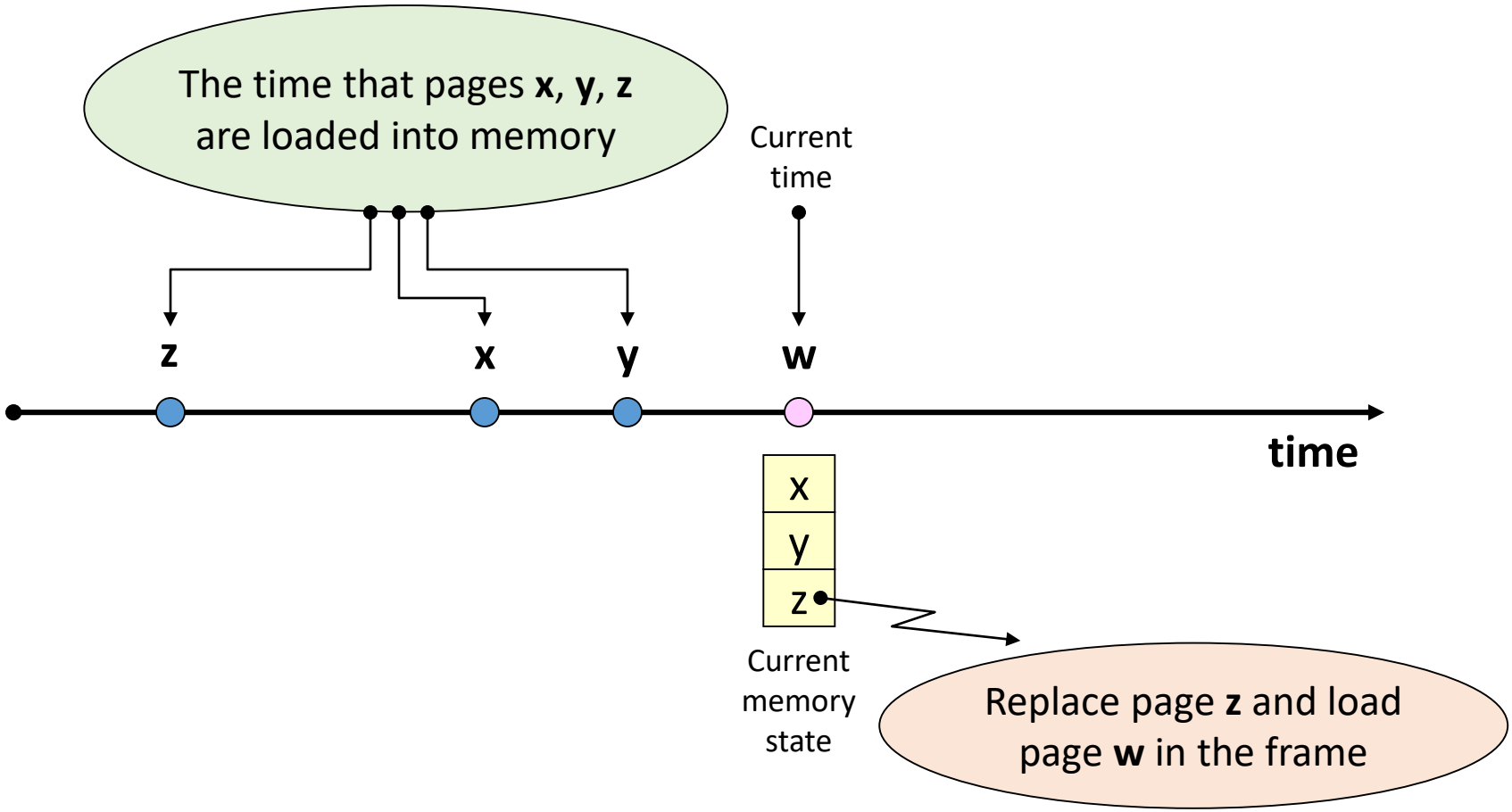


FIFO Algorithm

- First In First Out
  - 가장 오래된 page를 교체
- Page가 적재 된 시간을 기억하고 있어야 함

- 자주 사용되는 page가 교체 될 가능성이 높음
  - Locality에 대한 고려가 없음
- FIFO anomaly (Belady's anomaly)
  - FIFO 알고리즘의 경우,
    더 많은 page frame을 할당 받음에도 불구하고 page fault의 수가 증가하는 경우가 있음

FIFO Algorithm



FIFO Algorithm

- 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	5	5	5	5	5	5	5	4	4
		2	2	2	2	2	1	1	1	1	1	1	1	5
			6	6	6	6	6	2	2	2	2	2	2	2
					4	4	4	4	4	4	4	6	6	6
Page fault	F	F	F		F	F	F	F				F	F	F

- **Number of page faults =**

FIFO Algorithm

- Example (FIFO Anomaly)
  - ω = 1 2 3 4 1 2 5 1 2 3 4 5

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

- **Number of page faults =**

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

- **Number of page faults =**

FIFO Algorithm

- Example (FIFO Anomaly)
  - ω = 1 2 3 4 1 2 5 1 2 3 4 5

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

- **Number of page faults = 9**

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

- **Number of page faults = 10**

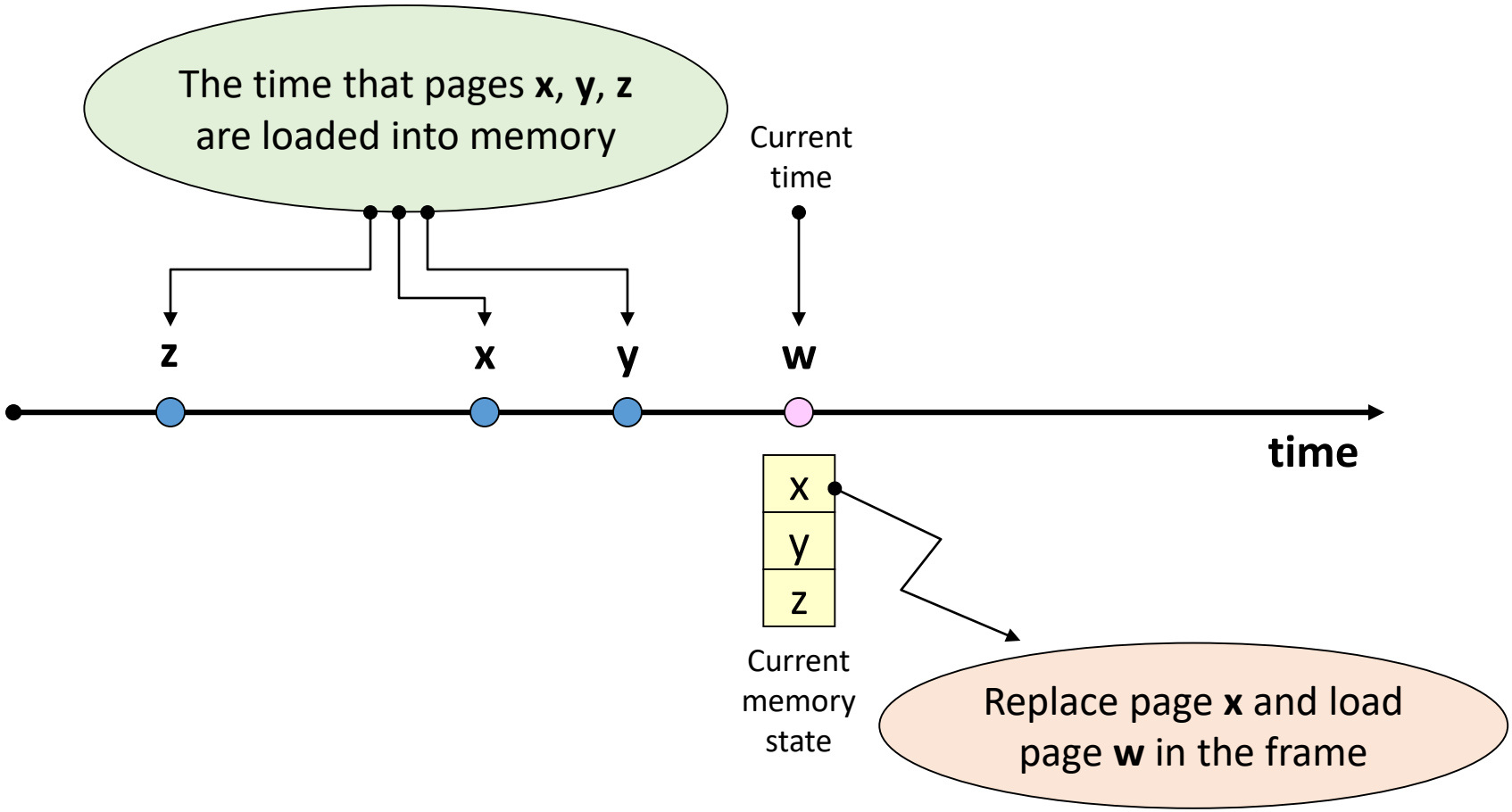
LRU (Least Recently Used) Algorithm

- 가장 오랫동안 참조되지 않은 page를 교체
- Page 참조 시 마다 시간을 기록해야 함

- Locality에 기반을 둔 교체 기법
- MIN algorithm에 근접한 성능을 보여줌

- 실제로 가장 많이 활용되는 기법

LRU (Least Recently Used) Algorithm



LRU (Least Recently Used) Algorithm

- 단점
  - 참조 시 마다 시간을 기록해야 함 (Overhead)
    - 간소화된 정보 수집으로 해소 가능
      - 예) 정확한 시간 대신, 순서만 기록
  - Loop 실행에 필요한 크기보다 작은 수의 page frame이 할당 된 경우, page fault 수가 급격히 증가함
    - 예) loop를 위한 |Ref. string| = 4 / 할당된 page frame 이 3개
    - Allocation 기법에서 해결 해야 함

LRU (Least Recently Used) Algorithm

- 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		

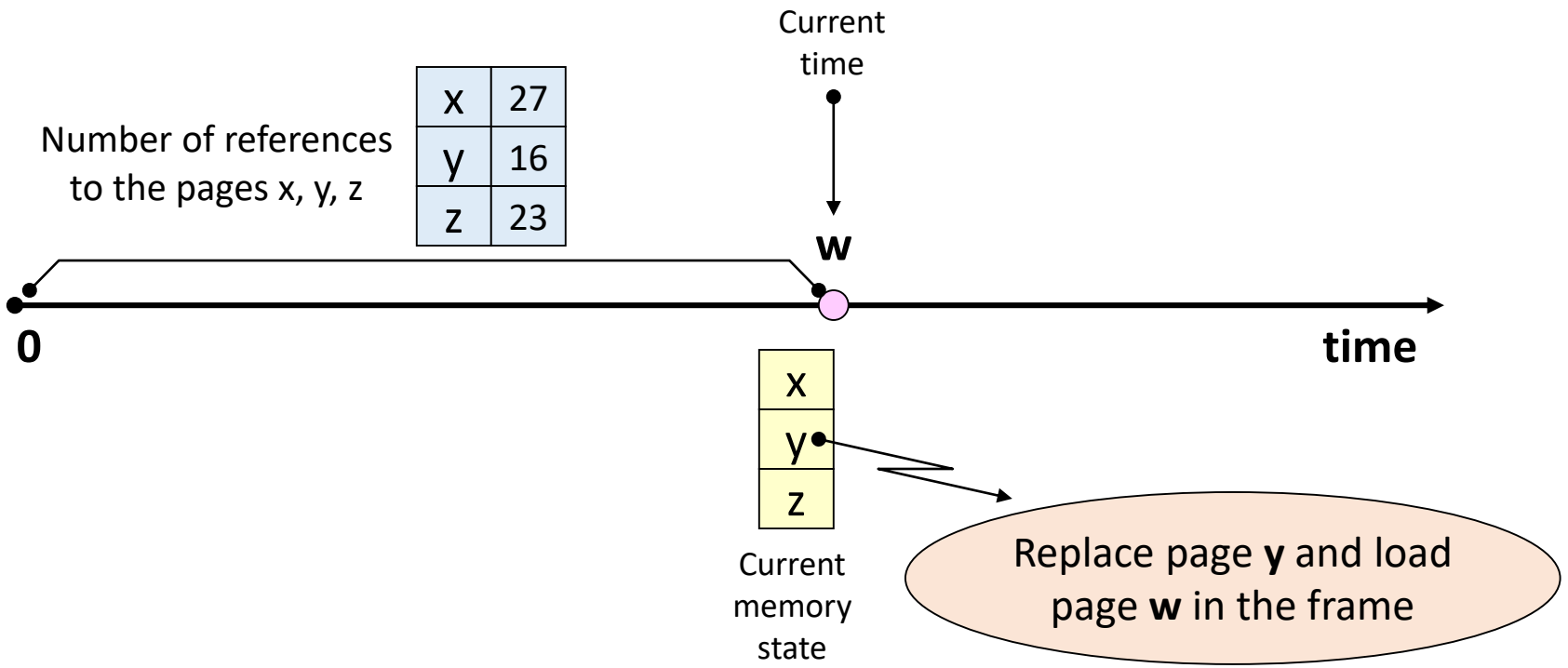
- **Number of page faults =**

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
- 단점
  - 최근 적재된 참조될 가능성인 높은 page가 교체 될 가능성이 있음
  - 참조 횟수 누적 overhead

LFU (Least Frequently Used) Algorithm



LFU (Least Frequently Used) Algorithm

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

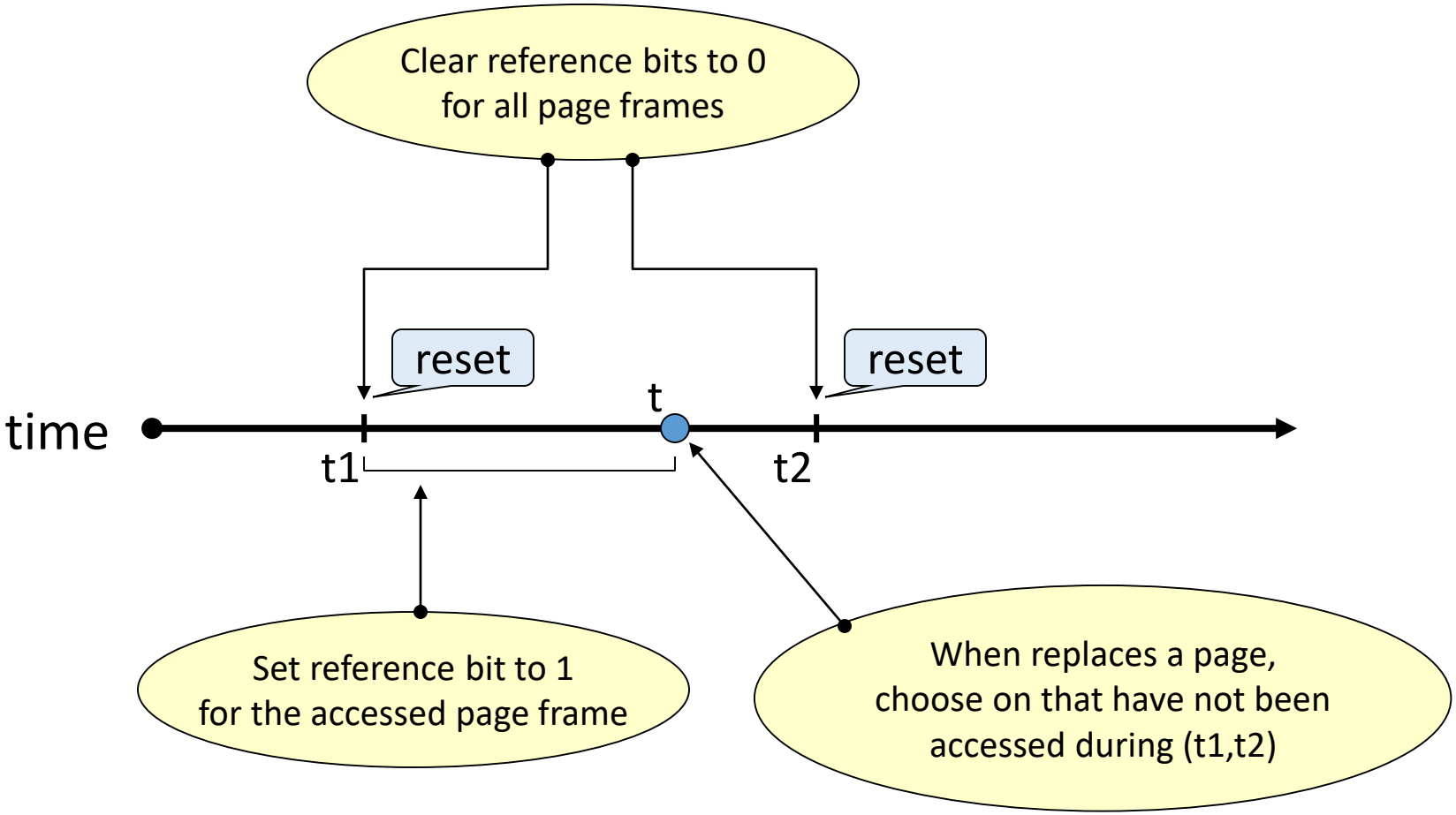
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ref. string	1	2	6	1	4	5	1	2	1	4	5	6	4	5
Memory state	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		2	2	2	2	5	5	5	5	5	5	5	5	5
			6	6	6	6	6	2	2	2	2	6	6	6
					4	4	4	4	4	4	4	4	4	4
Page fault	F	F	F		F	F		F				F		

- **Number of page faults =**

NUR (Not Used Recently) Algorithm

- LRU approximation scheme
  - LRU보다 적은 overhead로 비슷한 성능 달성 목적
- Bit vector 사용
  - Reference bit vector (r), Update bit vector (m)
- 교체 순서
  ① (r, m) = (0,0)
  ② (r, m) = (0,1)
  ③ (r, m) = (1,0)
  ④ (r, m) = (1,1)

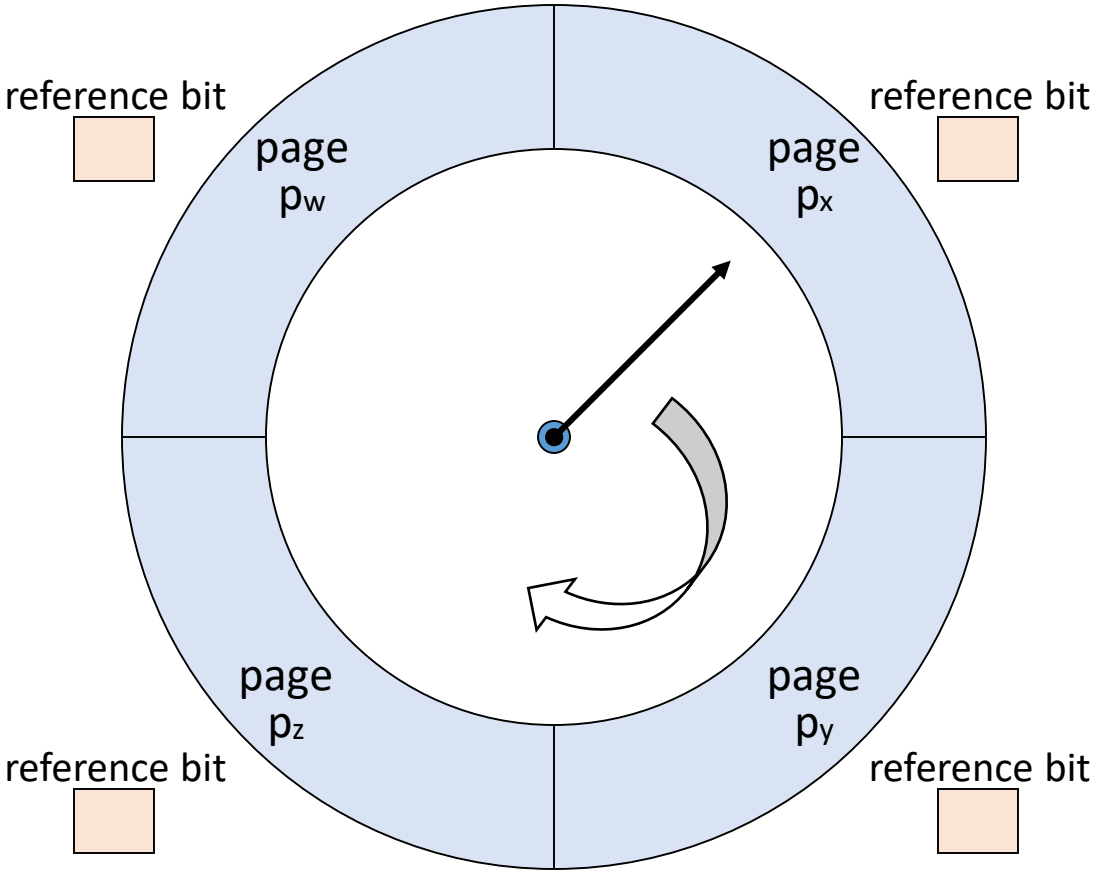
NUR (Not Used Recently) Algorithm



Clock Algorithm

- IBM VM/370 OS
- Reference bit 사용함
  - 주기적인 초기화 없음
- Page frame들을 순차적으로 가리키는 pointer (시계바늘)를 사용하여 교체될 page 결정

Clock Algorithm



Clock Algorithm

- 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

Clock Algorithm

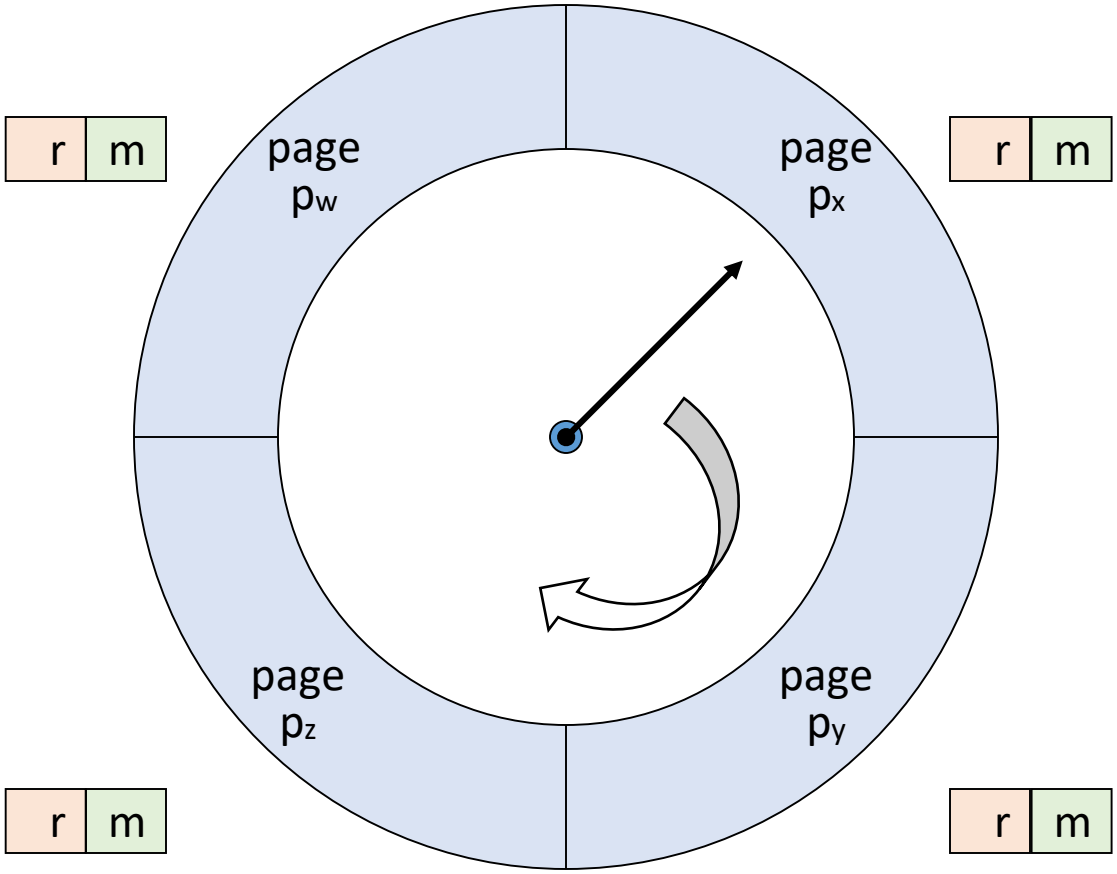
- Example
  - 4 page frames for the process, initially it has a, b, c, d
  - ω = c a d b e b a b c d

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							F		F		F	F
Pclock (loaded page)							e		a		c	d
Qclock (displaced page)							a		c		d	e

Second Chance Algorithm

- Clock algorithm과 유사
- Update bit (m)도 함께 고려 함
  - 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

Second Chance Algorithm



Second Chance Algorithm

- 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

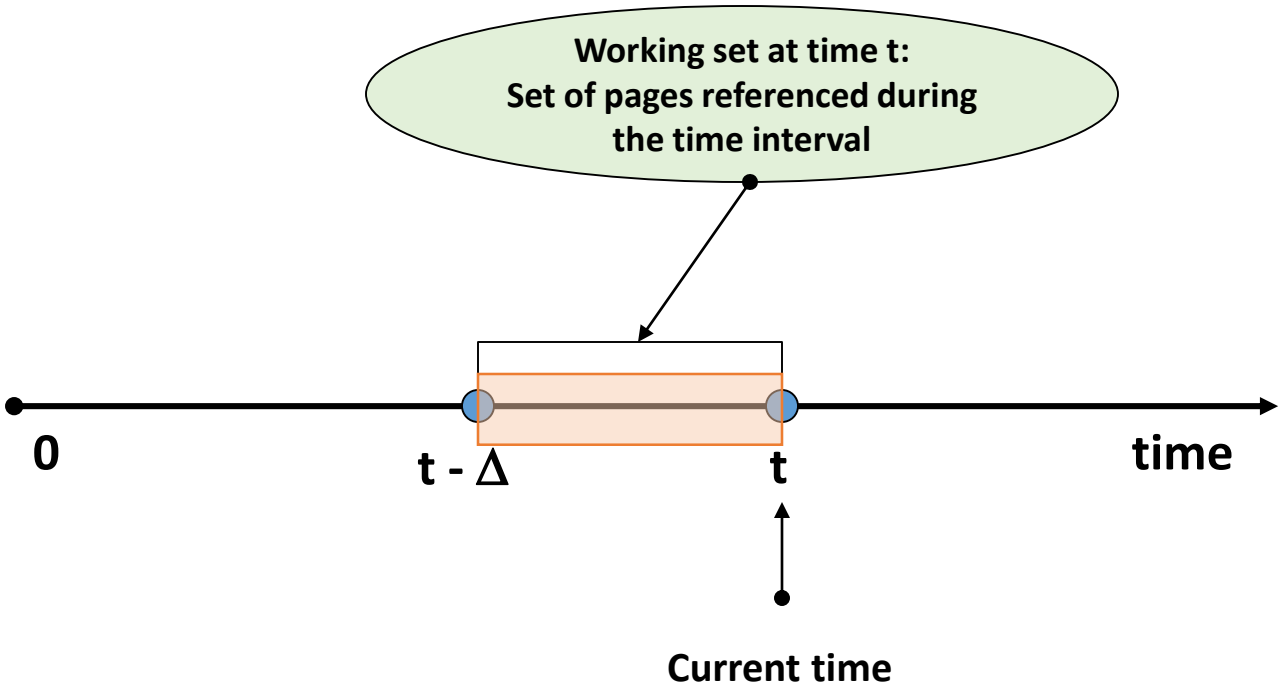
Replacement Strategies

- **Fixed allocation**
  - MIN(OPT, B0) algorithm
  - Random algorithm
  - FIFO(First In First Out) algorithm
  - LRU(Least Recently Used) algorithm
  - LFU(Least Frequently Used) algorithm
  - NUR(Not Used Recently) algorithm
  - Clock algorithm
  - Second chance algorithm
- **Variable allocation**
  - WS(Working Set) algorithm
  - PFF(Page Fault Frequency) algorithm
  - VMIN(Variable MIN) algorithm

Working Set (WS) algorithm

- 1968년 Denning이 제안
- Working set
  - Process가 특정 시점에 자주 참조하는 page들의 집합
  - 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 (WS) algorithm

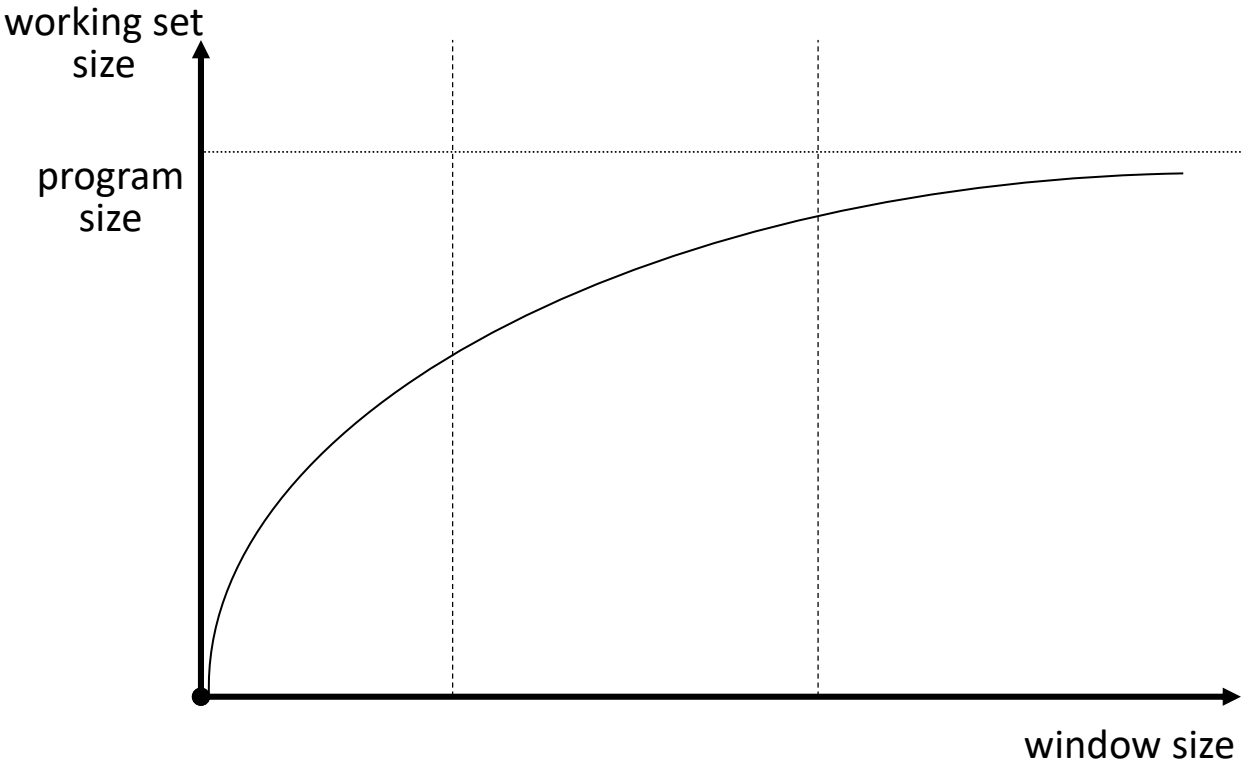


Working Set (WS) algorithm

- 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
      - MA 가 고정 and Δ가 가변 = ?
    - Δ 값이 성능을 결정 짓는 중요한 요소

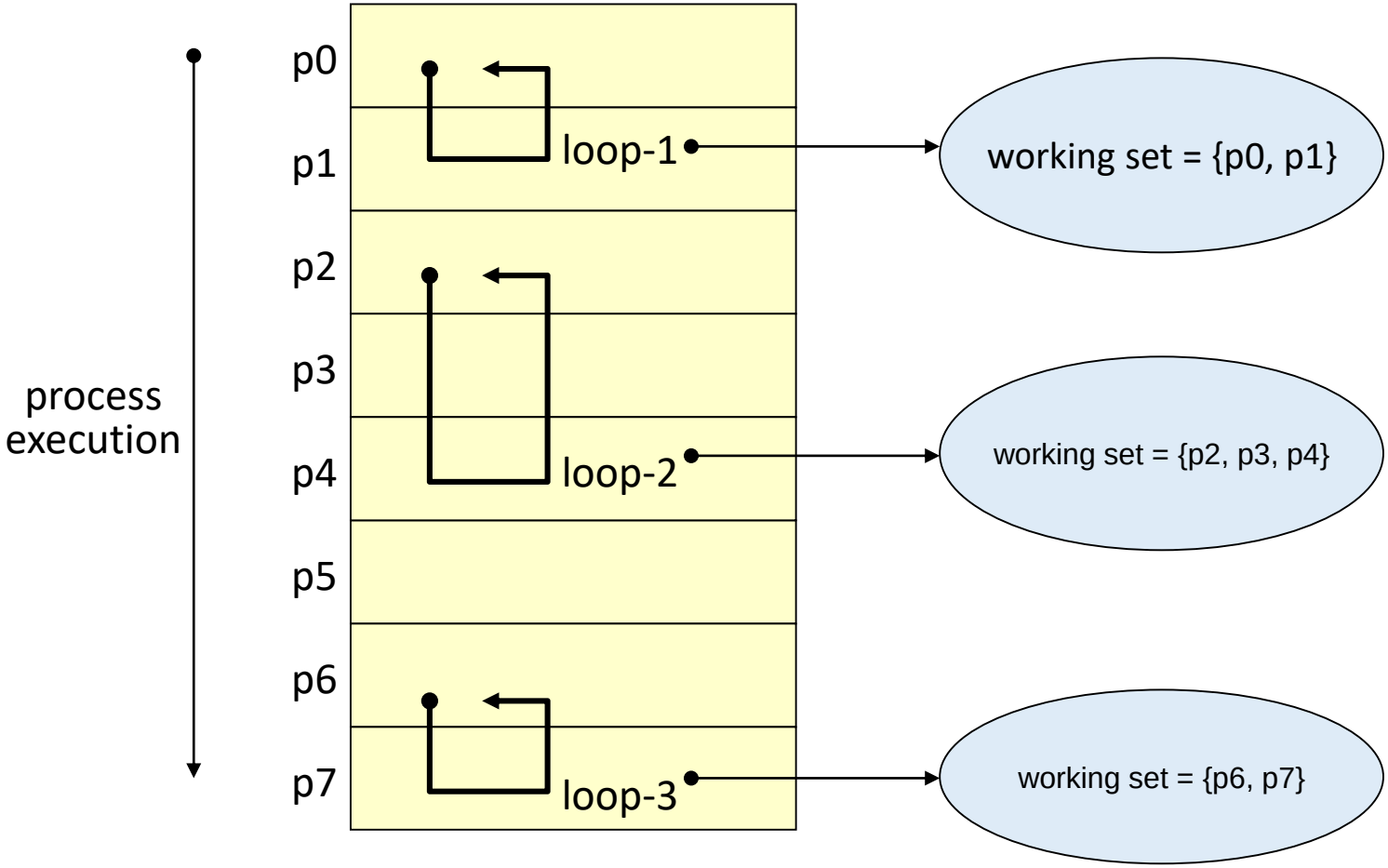
Working Set (WS) algorithm

- Window size vs. WS size



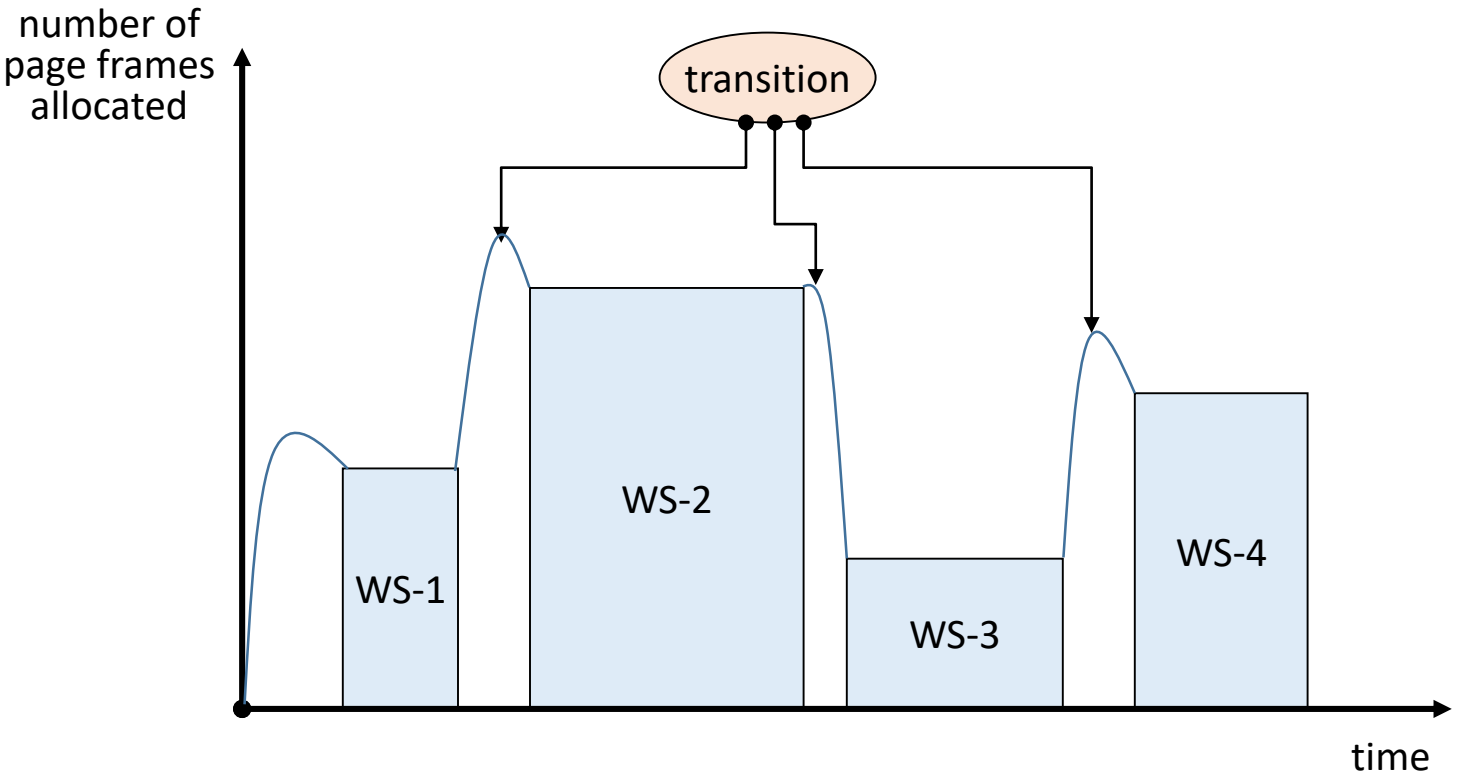
Working Set (WS) algorithm

- Example: working set transition



Working Set (WS) algorithm

- Working set transition



Working Set (ws) algorithm

- 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

Working Set (WS) algorithm

- 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

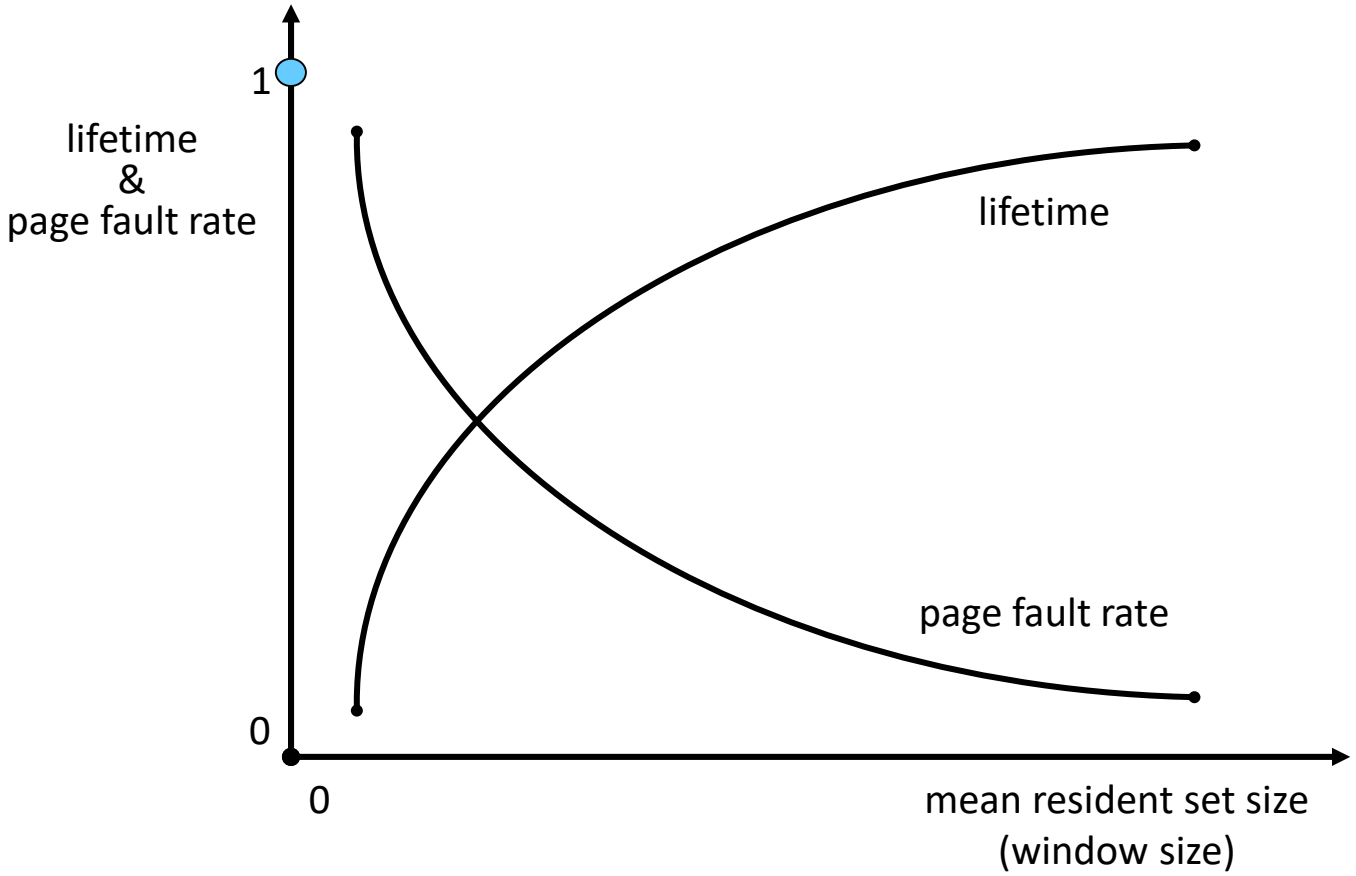
Working Set (WS) algorithm

- 특성
  - 적재 되는 page가 없더라도, 메모리를 반납하는 page가 있을 수 있음
  - 새로 적재되는 page가 있을 더라도, 교체 되는 page가 없을 수 있음

- 단점
  - Working set management overhead
  - Residence set (상주 집합)을 Page fault가 없더라도, 지속적으로 관리해야 함

Working Set (WS) algorithm

- **Mean number of frames allocated** vs. **page fault rate**



Page Fault Frequency (PFF) algorithm

- Residence set size를 page fault rate에 따라 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 발생시에만 수행
  - Low overhead

Page Fault Frequency (PFF) algorithm

- Criteria for page fault rate
  - IFT > τ : Low page fault rate
  - IFT < τ : High page fault rate

  - τ : threshold value
    - System parameter

Page Fault Frequency (PFF) algorithm

- 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 $> \tau$ (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 $\leq \tau$ (High page fault rate)
     - 기존 pages들 유지
     - + 현재 참조된 page를 추가 적재
       - 메모리 할당(# of page frames) 증가

Page Fault Frequency (PFF) algorithm

- 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_{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

Page Fault Frequency (PFF) algorithm

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

Variable MIN (VMIN) algorithm

- Algorithm
  - Page r이 t 시간에 참조 되면,
    page r이 $(t, t + \Delta]$ 사이에 다시 참조되는지 확인
  - 참조 된다면, page r을 유지
  - 참조 안 된다면, page r을 메모리에서 내림

Variable MIN (VMIN) algorithm

- 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

Variable MIN (VMIN) algorithm

- 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

Variable MIN (VMIN) algorithm

- 최적 성능을 위한 Δ값은?
  - $\Delta = \frac{R}{U}$
    - U: 한번의 참조 시간 동안 page를 메모리에 유지하는 비용
    - R: page fault 발생 시 처리 비용
  - $R > \Delta * U$, (Δ 가 작으면)
    - 처리 비용 > page 유지 비용
  - $R < \Delta * U$, (Δ 가 크면)
    - page fault 처리 비용 < 유지 비용

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) bytes ~ 2^{22} (4M) bytes
- **Small page size**
  - Large page table / # of PF
    - High overhead (kernel)
  - 내부 단편화 감소
  - I/O 시간 증가
  - Locality 향상
  - Page fault 증가
- **Large page size**
  - Small page table / # of PF
    - Low overhead (kernel)
  - 내부 단편화 증가
  - I/O 시간 감소
  - Locality 저하
  - 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

Program Restructuring

- Example
  - Paging system, Page size = 1 KB
  - sizeof(int) = 4 bytes

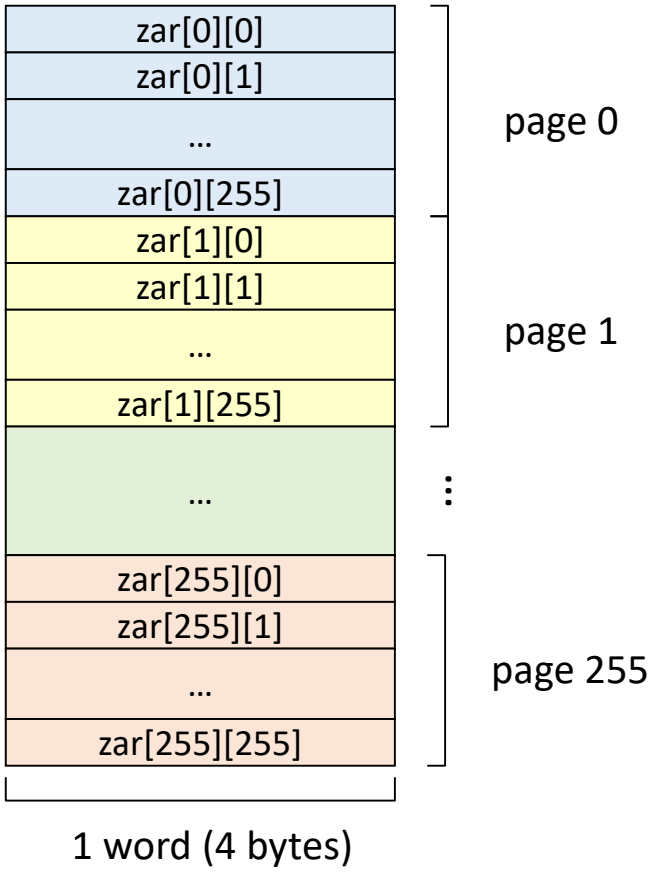
```
 // program-1
int main()
{
   int zar[256][256];
   int i, j;

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

   return 0;
}
```

Program Restructuring

- Row-major order for arrary

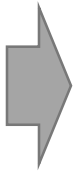


Program Restructuring

```
// 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;
}
```

TLB Reach

- TLB를 통해 접근 할 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

Summary

- Virtual memory management
- Cost model
- Hardware components
- Software components
- Page replacement schemes
  - FA-based schemes
  - VA-based schemes
- Other considerations