

# 10-1. Virtual memory management 1-3

## Virtual Memory Management

- 가상 메모리 (기억 장치)
  - Non-continuous allocation 시스템
    - 사용자 프로그램을 블록으로 분할하여 필요한 것만 적재/실행
  - Paging/Segmentation System + Hybrid System
- 가상 메모리 관리 목적
  - 가상 메모리 시스템 성능 최적화
    - Cost Model: 성능 지표
    - 다양한 최적화 기법

## Cost Model for Virtual Memory System

### [Cost Model]

- Page fault frequency (발생 빈도)
  - Page fault rate (발생률)
- ➡ Page fault rate를 최소화할 수 있도록 전략을 설계해야 함
- Context switch 및 kernel 개입 최소화. (발생 시 오버헤드 큼)
  - 시스템 성능 향상

### [용어]

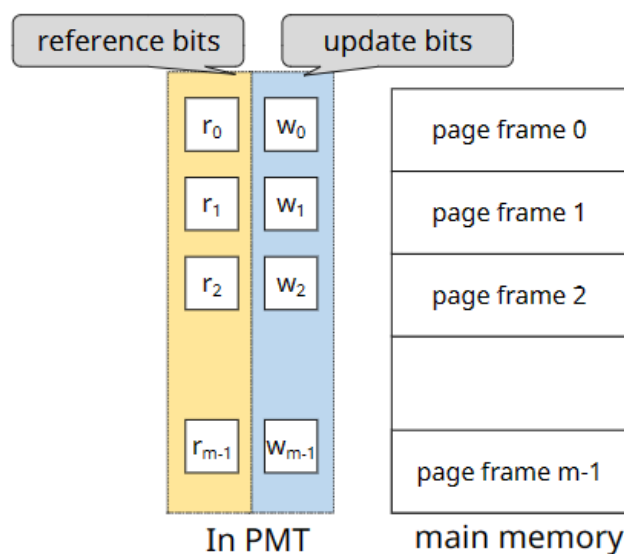
- Page reference string  $d$ 
  - 프로세스의 수행 중 참조한 페이지 번호와 순서
  - $\omega = r_1 r_2 \dots r_k \dots r_T$ 
    - $r_i$ : 페이지 번호,  $r_i \in 0, 1, 2, \dots, N-1$
    - $N$ : 프로세스의 페이지 수 ( $0 \sim N-1$ )

- **Page fault rate =  $F(\omega)$** 
  - $F(\omega) = \frac{\text{Num of page faults during } \omega}{|\omega|}$ 
    - $|\omega|$ : 전체 참조한 페이지 수

## Hardware Components

| 가상 메모리를 실행하기 위해 필요한 HW 요소

- **Address translation device (주소 사상 장치)**
  - 주소 사상을 효율적으로 수행하기 위해 사용
  - eg. TLB (associated memories), dedicated page-table register, cache memories
- **Bit Vectors**

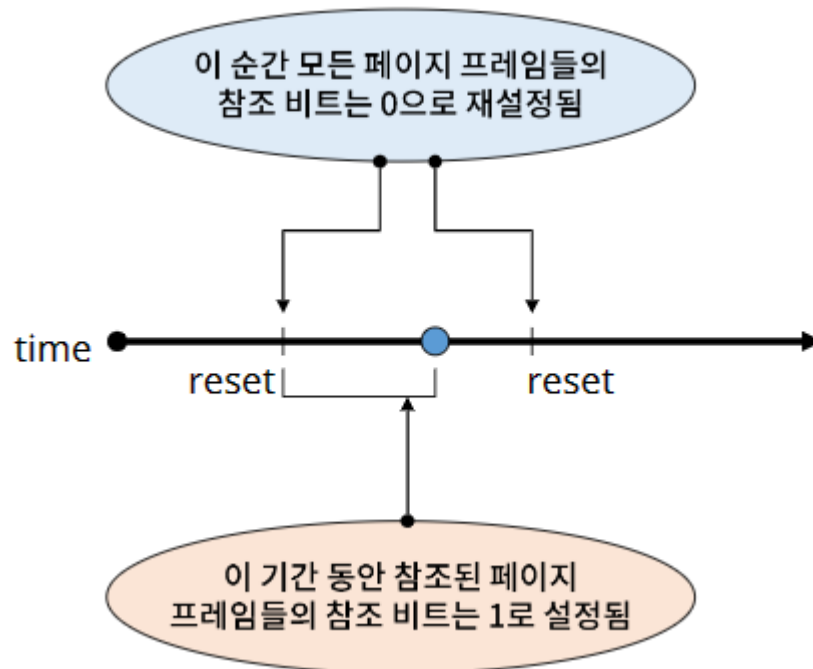


- Page 사용 상황에 대한 정보를 기록하는 비트들
- Reference bits (used bit): 참조 비트. 페이지가 참조되었는가
- Update bits (modified bits, write bits, dirty bits): 갱신 비트. 페이지가 갱신되었는가

## Bit Vectors

### Reference Bit Vector

메모리에 적재된 각각의 page가 최근에 참조되었는지 표시



- 운영
  1. 프로세스에 의해 참조되면 해당 page의 reference bit를 1로 설정
  2. 주기적으로 모든 reference bit를 0으로 초기화
- Reference bit를 확인함으로써 최근에 참조된 page들을 확인 가능
  - cf. locality

## Update Bit Vector

Page가 메모리에 적재된 후, 프로세스에 의해 수정되었는지를 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 필요하다는 뜻
    - cf. 데이터 무결성

## 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 얼마만큼 줄 것인가?
  - Fixed allocation (고정 할당)
    - 프로세스 실행 동안 고정된 크기의 메모리 할당
  - Variable allocation (가변 할당)
    - 프로세스 실행 동안 할당하는 메모리의 크기가 유동적
- 고려 사항
  - 프로세스 실행에 필요한 메모리 양을 예측해야 함
  - 너무 큰 메모리 할당 (Too much allocation)
    - 메모리가 낭비됨
  - 너무 적은 메모리 할당 (Too small allocation)
    - Page fault rate 
    - 시스템 성능 저하

## Fetch Strategies

- 특정 page를 메모리에 언제 적재(fetch)할 것인가?
  - Demand fetch (demand paging)
    - 프로세스가 참조하는 페이지들만 적재
    - Page fault overhead
  - Anticipatory fetch (pre-paging/pre-fetch)
    - 참조될 가능성이 높은 page 예측

- 가까운 미래에 참조될 가능성이 높은 page를 미리 적재
- 예측 성공 시, page fault overhead가 없음. 안 되면 오버헤드 큼
- Prediction overhead (Kernel 개입), hit ratio에 민감함
- **실제 대부분의 시스템은 Demand Fetch 기법 사용**
  - 일반적으로 준수한 성능을 보여줌
  - Anticipatory fetch는 prediction overhead + 잘못된 예측 시 자원 낭비가 큼

## Placement Strategies

- **Page/segment를 어디에 적재할 것인가?**
  - Paging system에는 불필요. 메모리 분할 크기가 일정하기 때문
- **Segmentation system에서의 배치 기법**
  - First-fit
  - Best-fit
  - Worst-fit
  - Next-fit

## Replacement Strategies

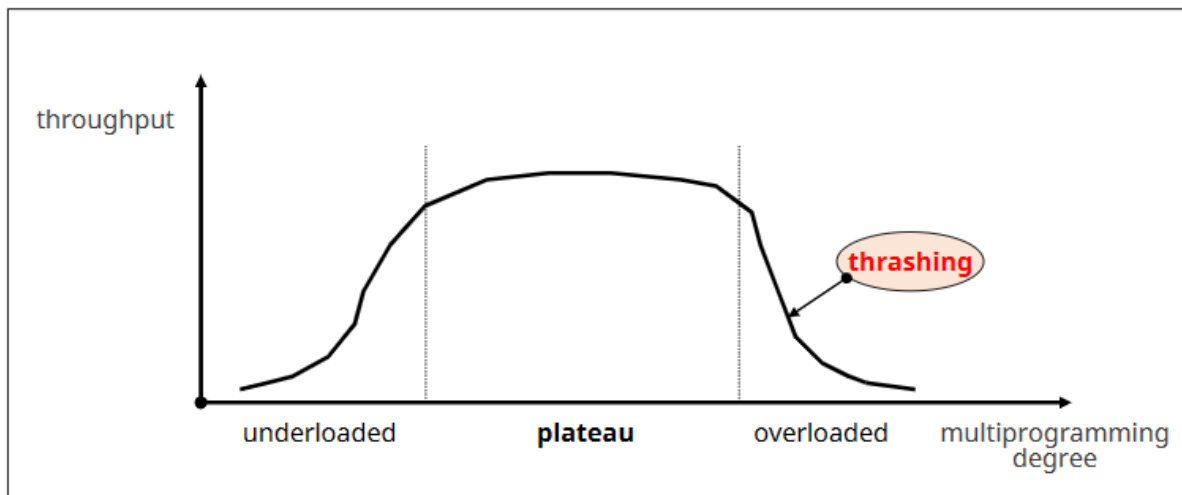
- **새로운 page를 어떤 page와 교체할 것인가 (빈 page frame이 없는 경우)**
  - Fixed allocation을 위한 교체 기법
  - Variable allocation을 위한 교체 기법

## Cleaning Strategies

- **변경된 page를 언제 write-back 할 것인가?**
  - 변경된 내용(dirty bit =1인 것)을 swap device에 반영
  - **Demand cleaning**
    - 해당 page에 메모리에서 내려올 때 write-back
  - **Anticipatory cleaning(pre-cleaning)**
    - 더 이상 변경될 가능성이 없다고 판단할 때, 미리 write-back
    - Page 교체 시 발생하는 write-back 시간 절약
    - Write-back 이후, page 내용이 수정되면 오버헤드 발생!

- 실제 대부분의 시스템은 Demand Cleaning 기법 사용
  - 일반적으로 준수한 성능을 보여줌
  - Anticipatory cleaning은 prediction overhead + 잘못된 예측 시 자원 낭비가 큼

## Load Control Strategies



- 시스템의 multi-programming degree (시스템에 들어온 프로그램 수) 조절
  - Allocation strategies와 연계됨
- 적정 수준의 multi-programming degree를 유지해야 함
  - Plateau(고원) 영역으로 유지
  - 저부하 상태(Underloaded)
    - 시스템 자원 낭비, 성능 저하
  - 고부하 상태(Overloaded)
    - 자원에 대한 경쟁 심화, 성능 저하
    - Thrashing(스레싱) 현상 발생
      - 과도한 page fault가 발행하는 현상

## Locality

프로세스가 프로그램/데이터의 특정 영역을 집중적으로 참조하는 현상

- 원인

- Loop structure in program
- Array, structure 등의 데이터 구조
- 종류
  - 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 word
- Machine instruction size = 1 word
- 주소 지정은 word 단위로 이루어짐
- 프로그램은 4번 page에 continuous allocation됨
- $n = 1000$

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
...

```

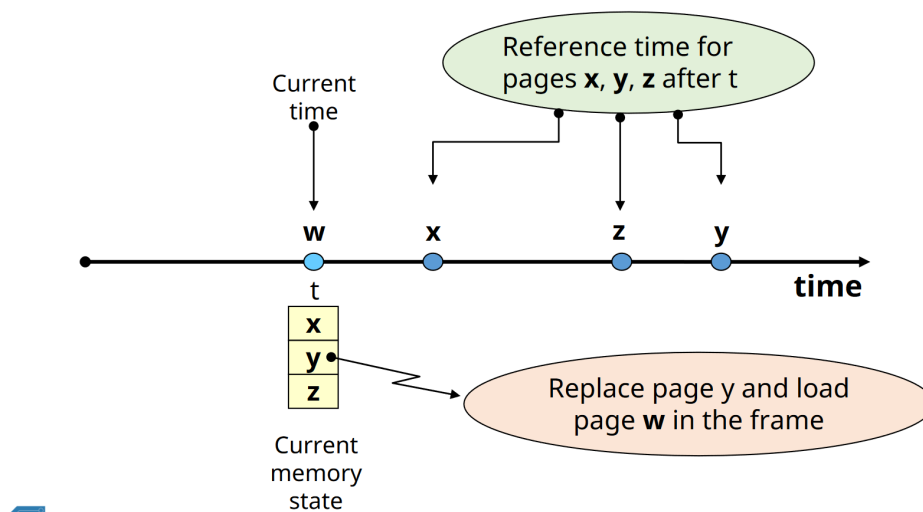
### [실행]

- $\omega = 494944(474846444)^{1000}$
- 지역성: 11000번의 메모리 참조 중 5개의 page만 집중적으로 접근하게 됨
- 지역성을 고려하면 성능을 향상시킬 수 있음

# Replacement Strategies

- **Fixed Allocation:** 정해진 수의 page frame을 프로그램에 할당
  - MIN(OPT, B0) algorithm
  - Random algorithm
  - FIFO 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)



y를 교체하게 될 것. 제일 오랫동안 접근 안 되니까

- 1966년 Beadly에 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 string( $\omega$ )을 미리 알고 있어야 함

- 교체 기법의 성능 평가 도구로 사용됨

[예시]

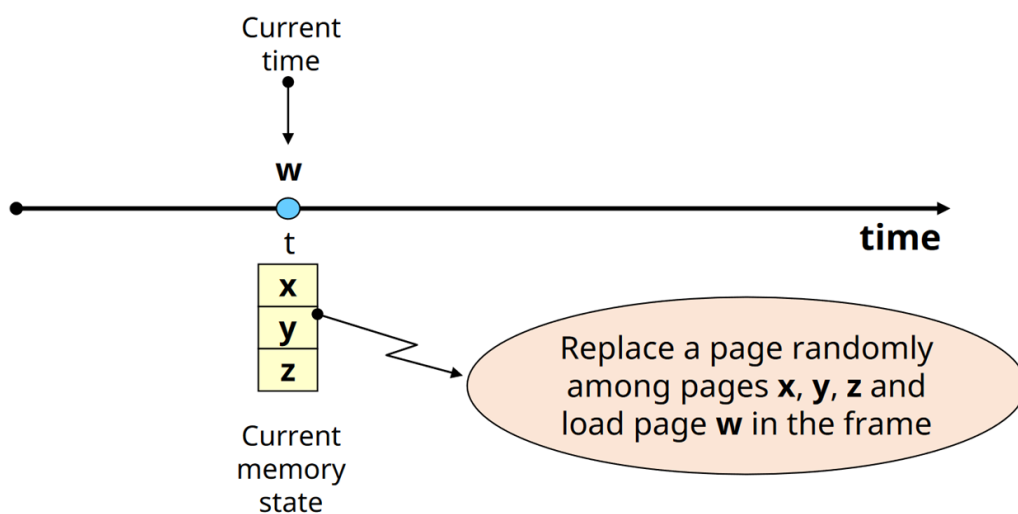
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ref. string	1	2	6	1	4	5	1	2	1	4	5	6	4	5
Memory state	1	1	1	1	1	1	1	1	1	1	1	6	6	6
		2	2	2	2	2	2	2	2	2	2	2	2	2
			6	6	6	5	5	5	5	5	5	5	5	5
					4	4	4	4	4	4	4	4	4	4
Page fault	F	F	F		F	F						F		

- Number of page faults =

tie breaking rule: 페이지 번호 작은 것부터 교체

- Number of page faults = 6

## Random Algorithm



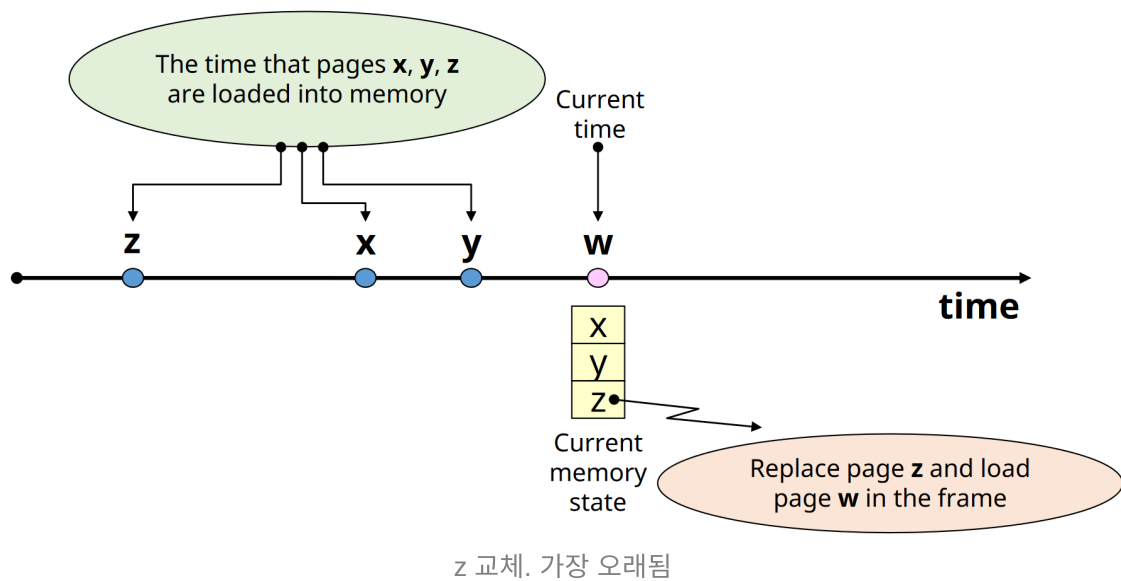
- 무작위로 교체할 page 선택

- Low overhead
- No policy

## FIFO Algorithm

**First In First Out. 가장 오래된 page를 교체**

- Page가 적재된 시간을 기억하고 있어야 함



- 자주 사용되는 page가 교체될 가능성이 높음

- Locality에 대한 고려가 없음
- 가장 먼저 올라왔다는 건 중요하거나 자주 사용된다는 뜻일 수 있는데, 이런 페이지가 FIFO에서는 가장 오래됐다고 교체됨

- FIFO anomaly (Belady's anomaly)

- FIFO 알고리즘의 경우, 더 많은 page frame을 할당 받음에도 불구하고 page fault의 수가 증가하는 경우가 있음

[예시]

- 4 page frames for the process, initially empty
- $\omega = 1\ 2\ 6\ 1\ 4\ 5\ 1\ 2\ 1\ 4\ 5\ 6\ 4\ 5$

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 =

- Number of page faults = 10

[예시2: FIFO Anomaly]

### • Example (FIFO Anomaly)

- $\omega = 1\ 2\ 3\ 4\ 1\ 2\ 5\ 1\ 2\ 3\ 4\ 5$

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

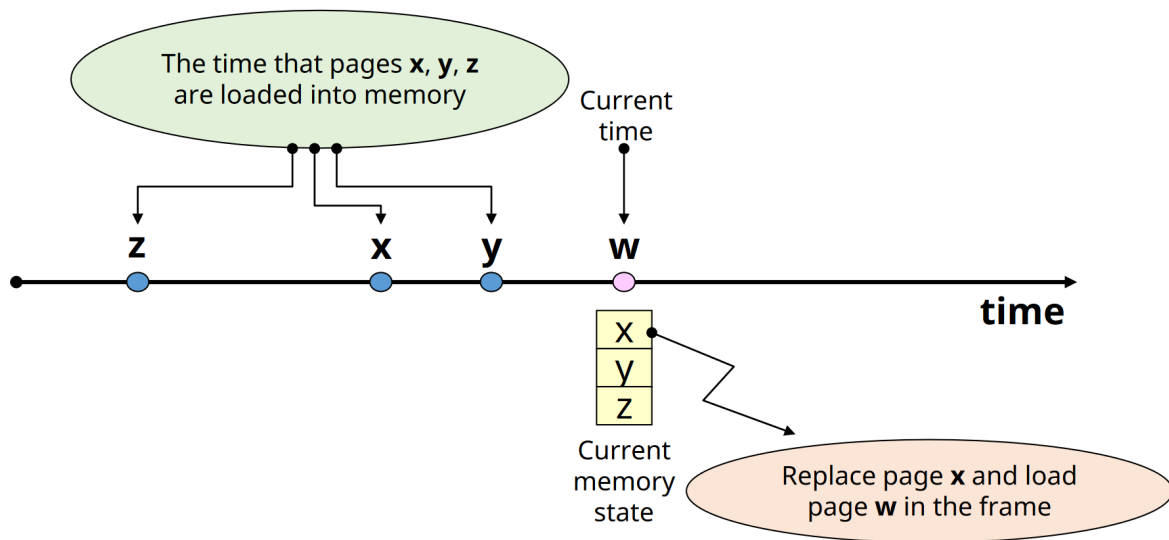
- 자원 증가 but 성능 저하
  - Page fault를 줄이기 위해 page frame을 더 줬는데도 더 많은 page fault가 발생  
⇒ locality를 고려하지 않았기 때문

## LRU(Least Recently Used) Algorithm

가장 오랫동안 참조되지 않은 page를 교체

- Locality에 기반을 둔 교체 기법

- Page 참조 시마다 시간을 기록해야 함



- MIN algorithm에 근접한 성능을 보여줌
- 실제로 가장 많이 활용되는 기법

#### [단점]

- 참조 시마다 시간을 기록해야 함 (오버헤드 발생)
  - 간소화된 정보 수집으로 해소 가능 (ex. 정확한 시간 대신 순서만 기록)
- Loop 실행에 필요한 크기보다 작은 수의 page frame이 할당된 경우, page fault 수가 급격히 증가함
  - ex. loop를 위한 ( $|\text{ref. string}| = \frac{4}{\text{할당된 page frame}}$ )이 3개
  - Allocation 기법에서 해결해야 함
    - loop 실행에 필요한 만큼 page frame을 할당하는 것이 좋음

#### [예시]

- $\omega = 1\ 2\ 6\ 1\ 4\ 5\ 1\ 2\ 1\ 4\ 5\ 6\ 4\ 5$

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
  - 참조한 시간을 고려해야 한다는 단점 존재