12. Virtual Memory

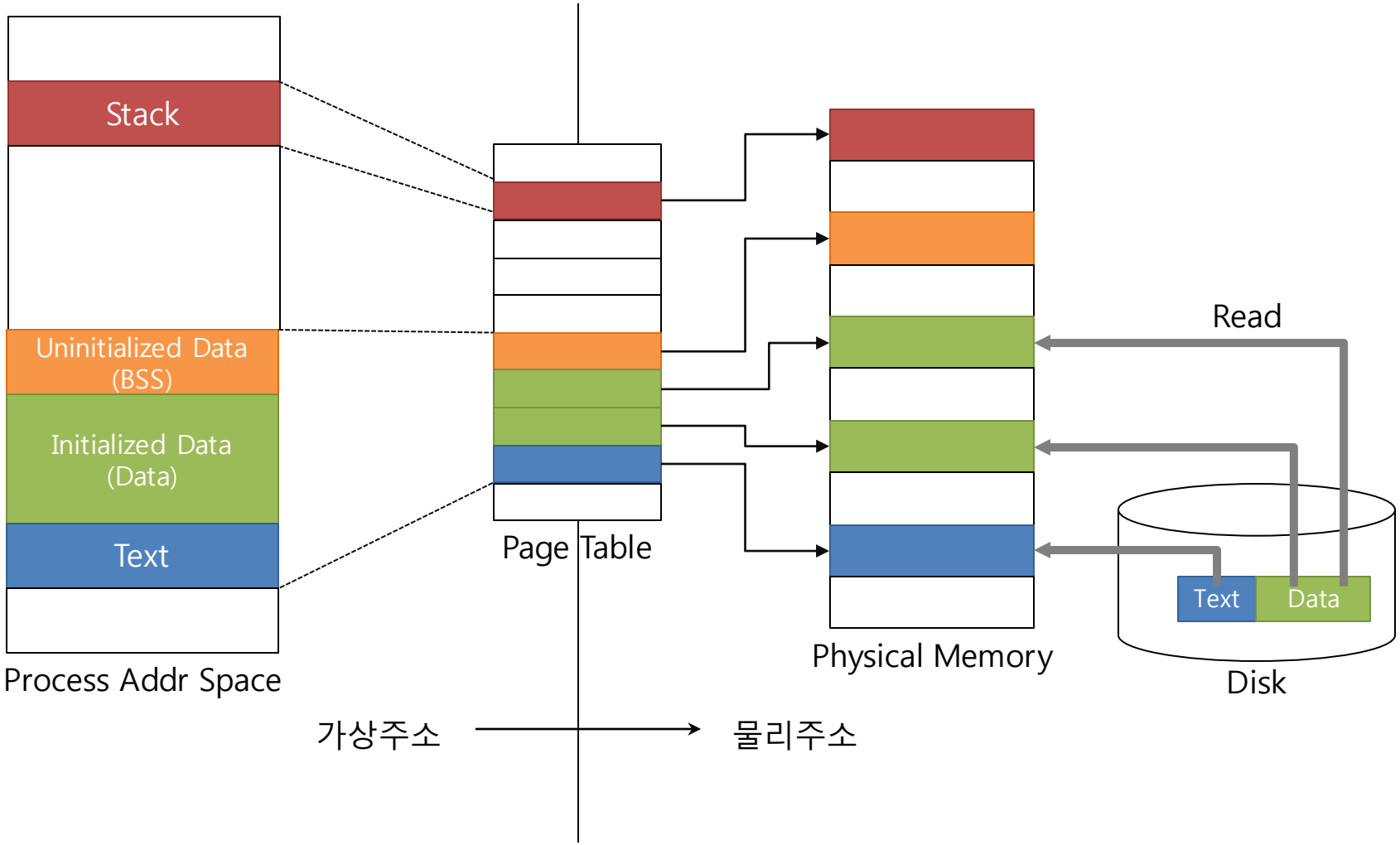
핀토스의 메모리 개요

- 현재 주소공간은 4개의 세그먼트로 구성
  - Stack
  - Initialized data
  - Uninitialized data
  - Code
  - **Heap은 현재 없음 !**

- 프로세스의 메모리 탑재 과정
  - 각 세그멘트(Stack, Data, BSS, Code)가 물리페이지에 탑재
  - 페이지 테이블 초기화

Pintos의 프로세스 주소 공간

- 과제 수행 전 pintos memory layout

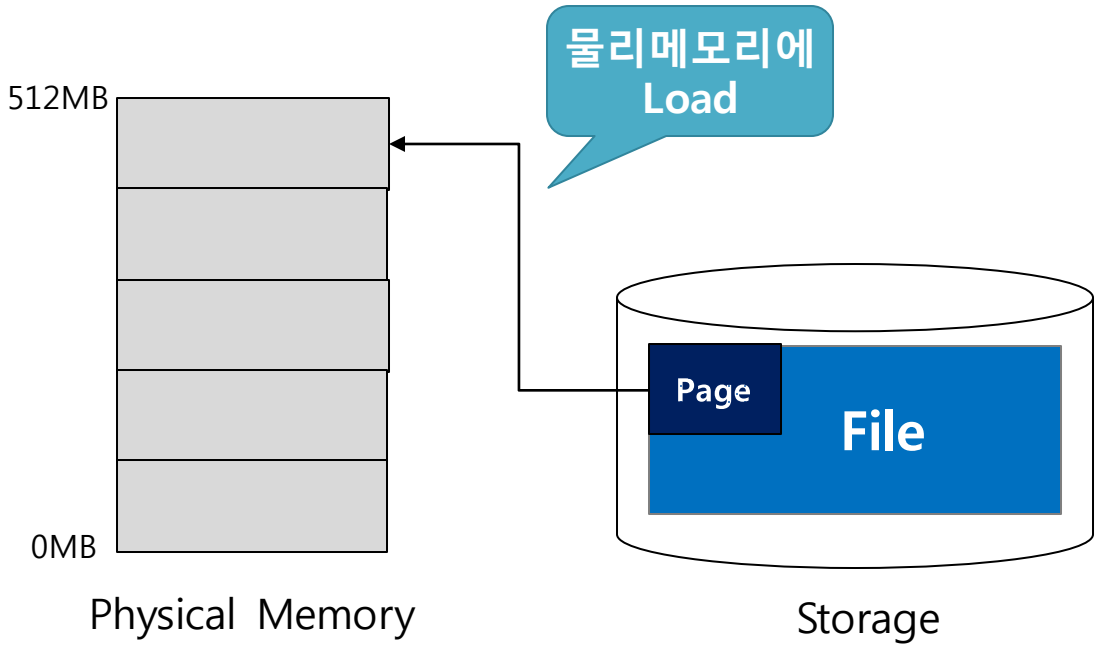


한계

- Swap을 사용할 수 없음
- Demand paging사용 불가
- Virtual memory가 구현되어 있지 않음!!!
- Pintos에서 virtual memory를 구현해 봅시다!!!

기본 개념:
Demand Paging

요구 페이징 (Demand Paging) 이란?

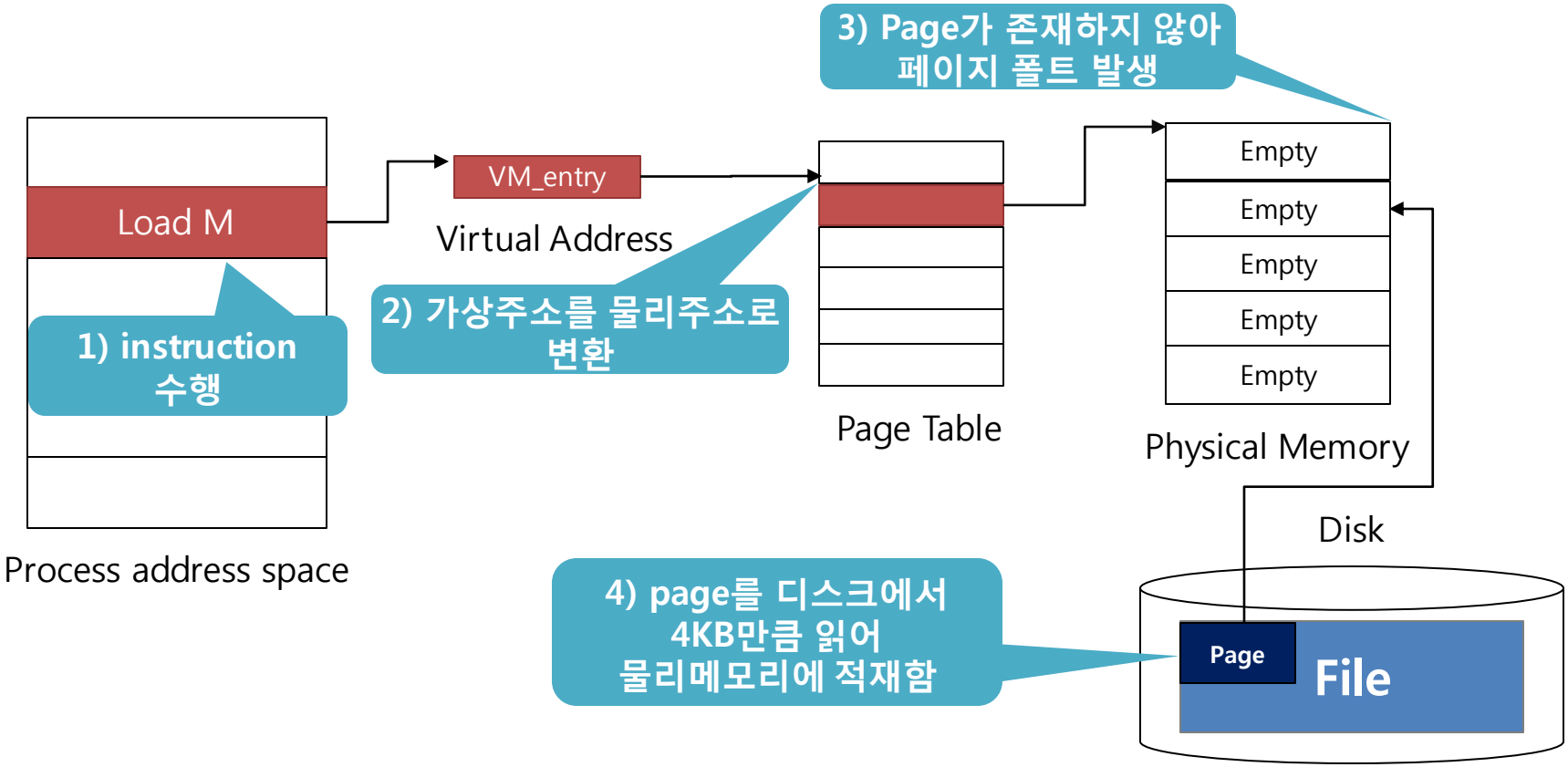


- 요구된 Page들만 물리메모리에 load 하는 기법

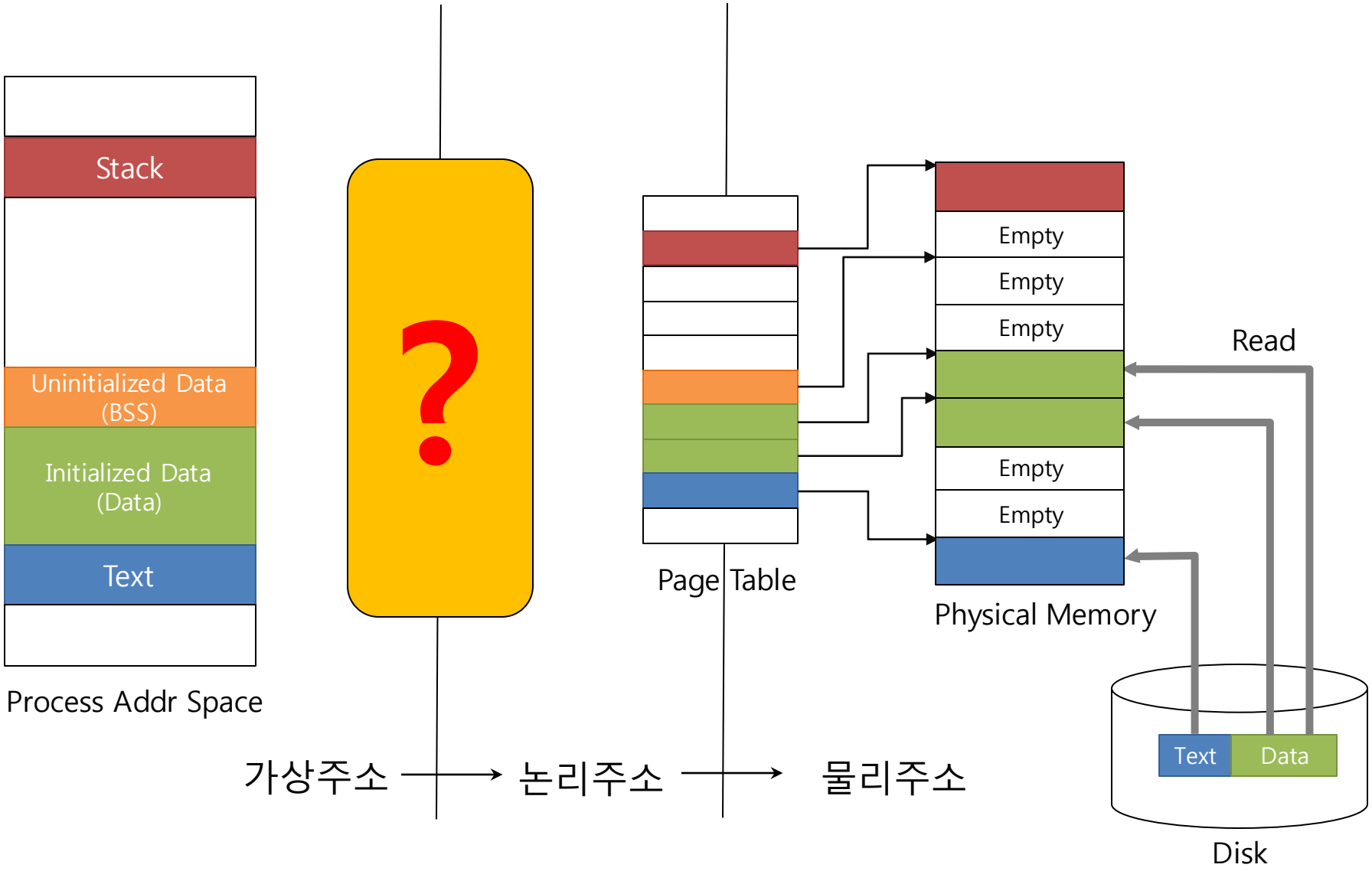
요구 페이징 (Demand Paging) 과정 (Cont.)

1. Instruction 실행
2. 가상주소로부터 가상 페이지 번호 추출
3. 페이지 테이블 참조
4. 페이지 테이블에 물리 페이지 부재시 페이지 폴트 발생
5. 페이지 폴트 발생시 페이지 프레임 할당하고 페이지 테이블 갱신
6. 해당 페이지를 디스크에서 페이지 프레임에 탑재

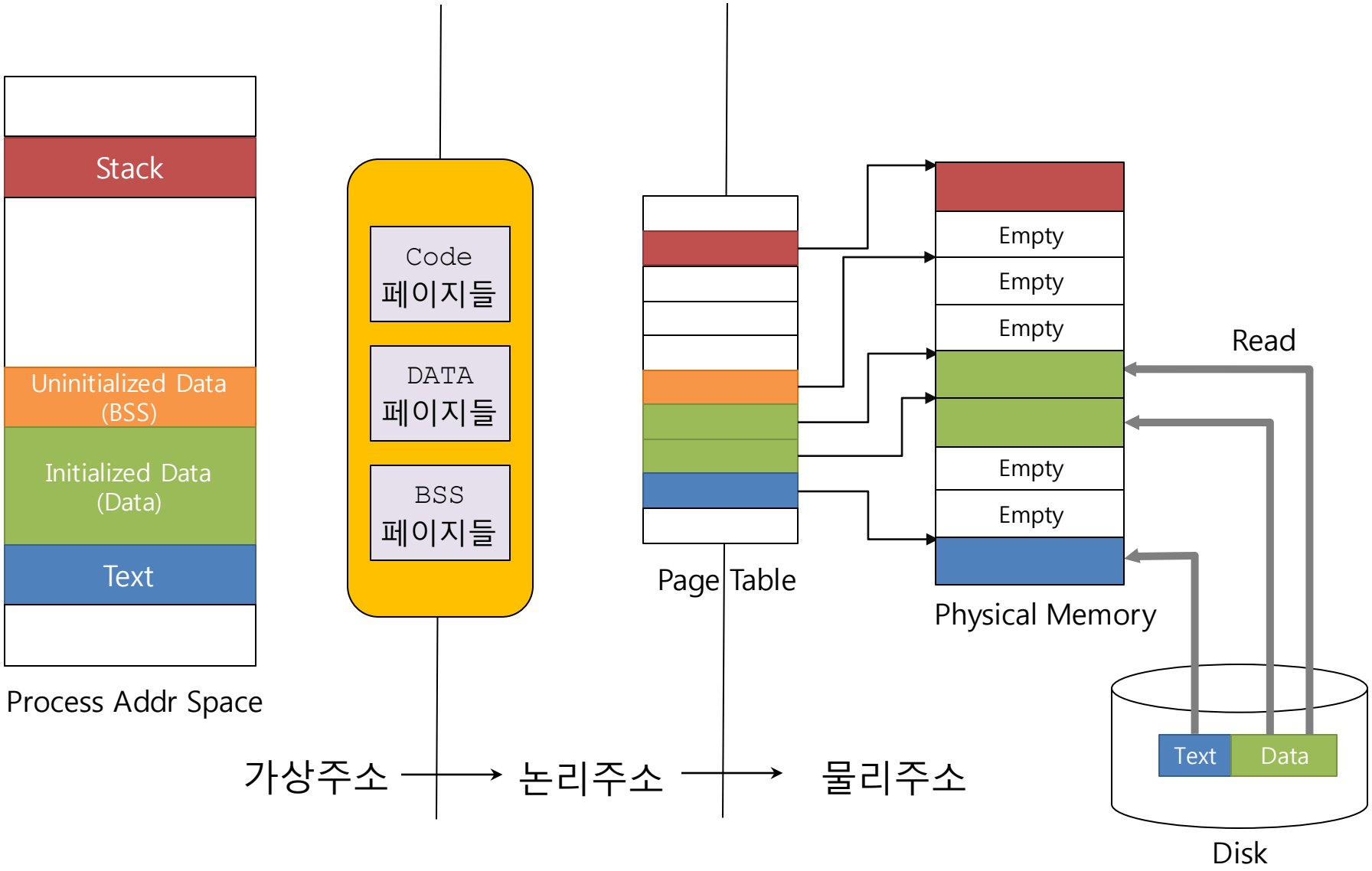
요구 페이징 (Demand Paging) 과정



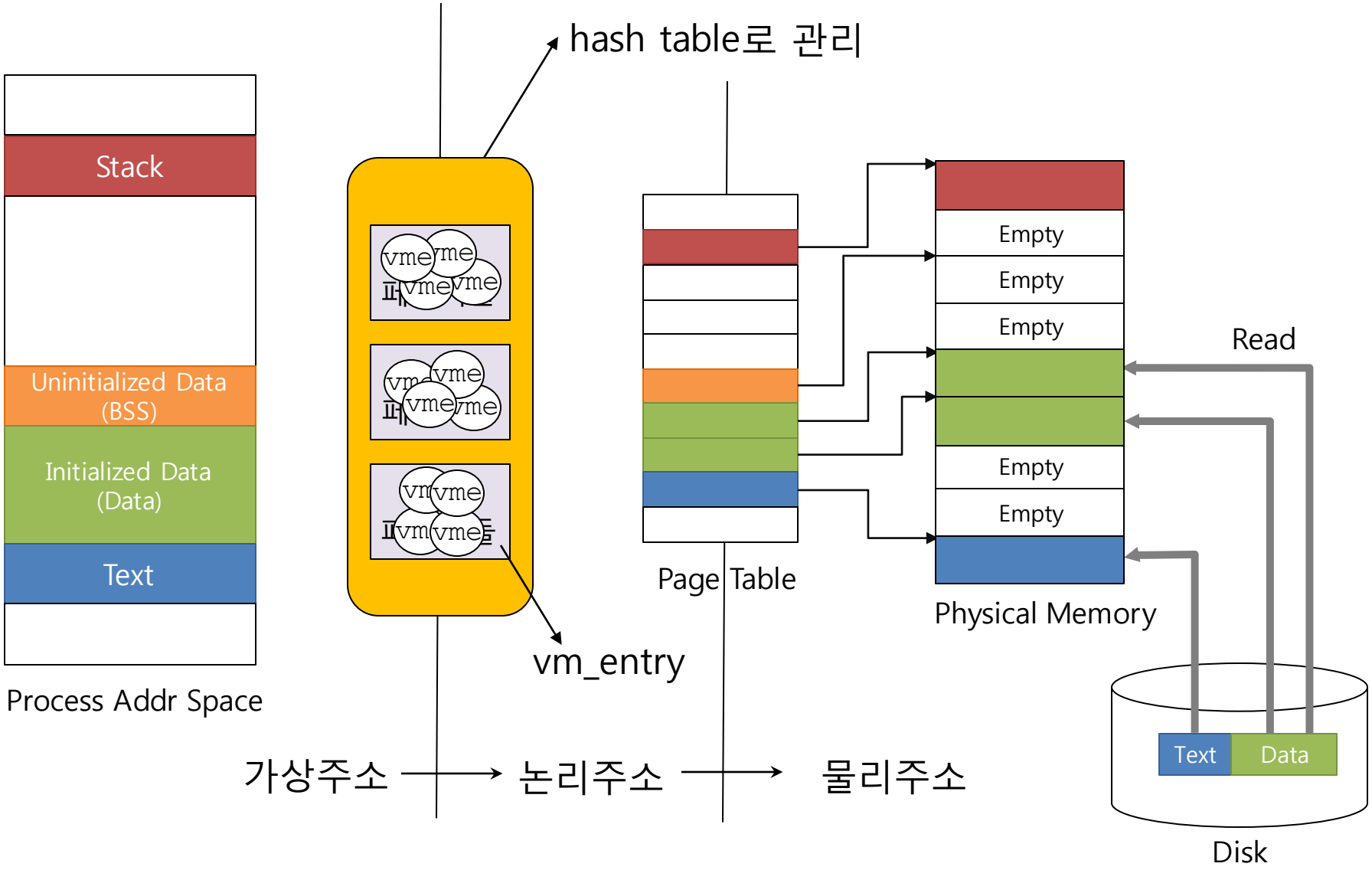
할일 1: 가상주소 공간의 구현



할일 1: 가상주소 공간의 구현 (Cont.)



할일 1: 가상주소 공간의 구현 (Cont.)



할일 2: 가상주소 공간에서의 페이지

- 가상주소 페이지 별 자료구조 정의
- vm_entry
  - 페이지당 하나
  - 각 페이지의 파일 포인터, 오프셋, 크기를 저장
  - 프로그램 초기 탑재시 가상 주소공간 각 페이지에 vm_entry 할당
  - 프로그램 실행시
    - 페이지 테이블 탐색
    - 페이지폴트 시, 가상주소에 해당하는 vm_entry를 탐색
      - vm_entry에 없는 가상 주소는 Segmentaion fault
      - vm_entry가 존재할 경우,
        - 페이지 프레임 할당
        - vm_entry에 있는 파일포인터, 읽기 시작할 오프셋, 읽어야 할 크기 등을 참조해서 페이지 로드
        - 페이지 테이블 갱신

할일 3: 가상주소 공간의 초기화

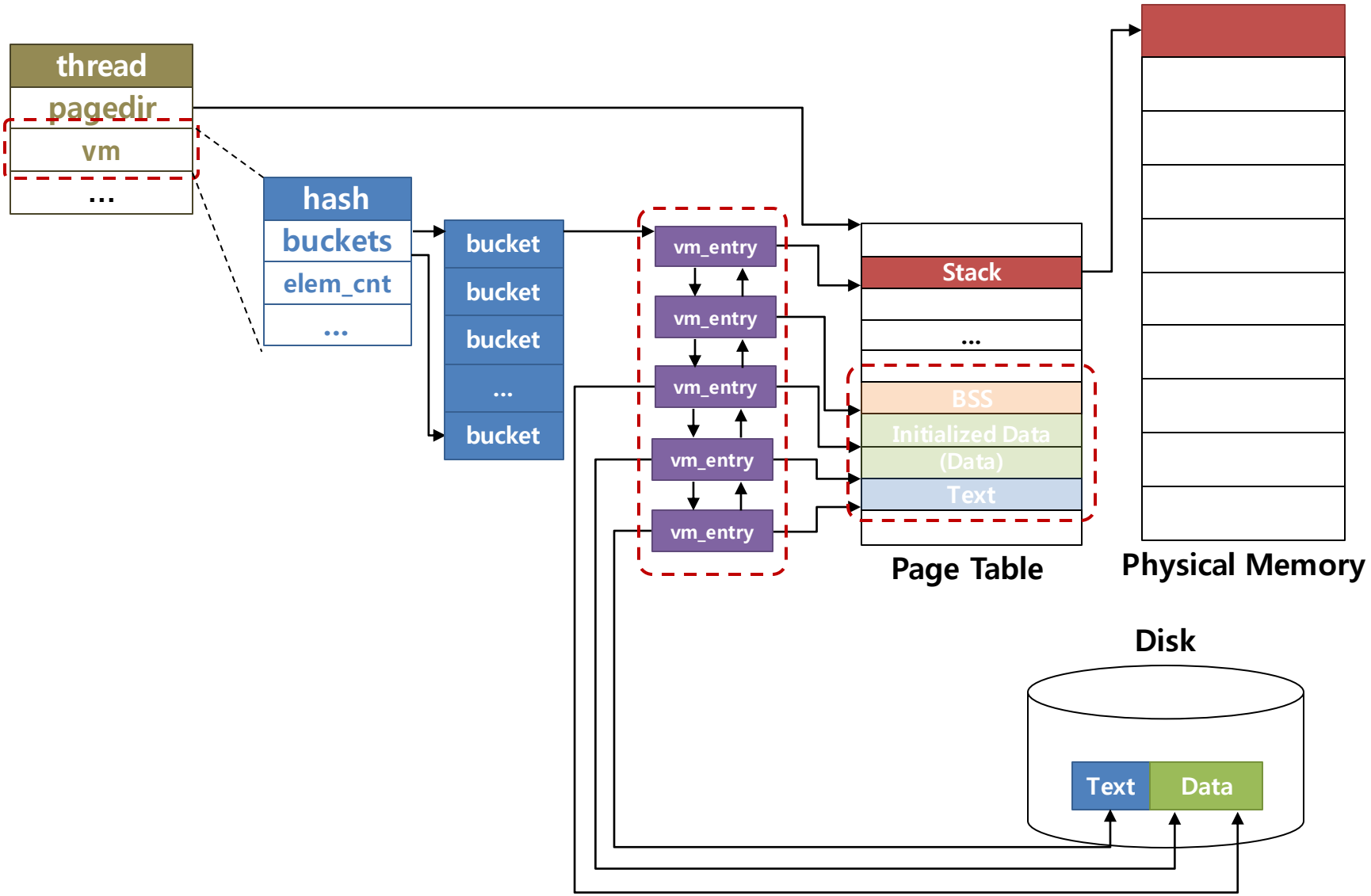
- 기존의 핀토스의 가상주소 공간 초기화 과정
  - ELF 이미지 각 페이지를 물리메모리로 읽어들임.
    - load_segment()로 Data, Code 세그멘트 읽음
    - setup_stack()로 Stack에 물리페이지 할당

- 수정된 핀토스의 가상주소 공간 초기화 과정
  - 디스크 이미지의 세그멘트를 전부 올리는 것은 물리 메모리의 낭비를 초래
  - 물리메모리 할당 대신, 가상 페이지마다 vm_entry를 통해 적재할 정보들만 관리

할일 4: 요구페이징을 위한 페이지 폴트 수정

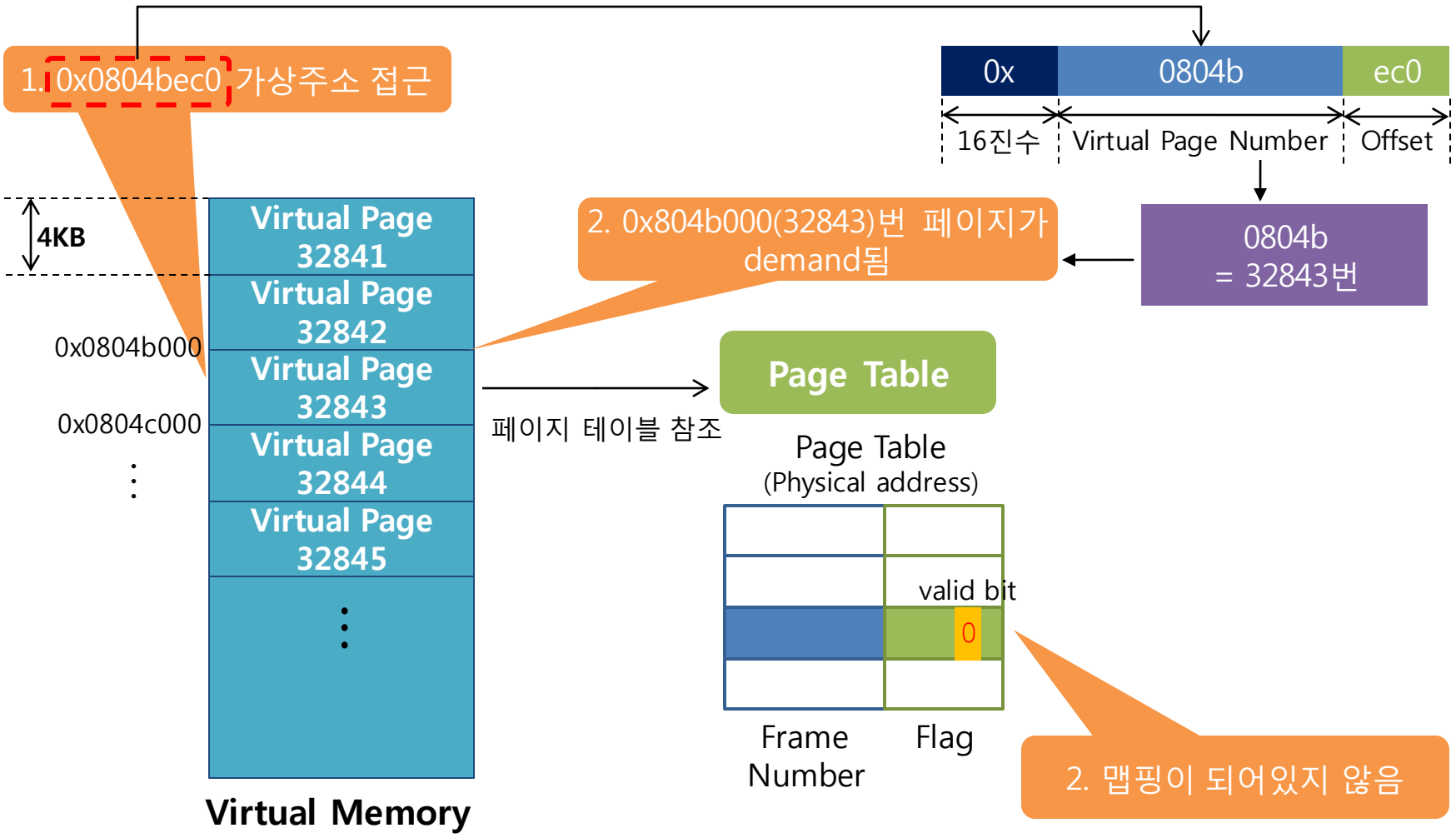
- Demand Paging 구현
  - 프로세스가 요청한 페이지들에 대해서만 물리페이지를 할당해주는 기법
    - 현재 Pintos는 프로그램의 모든 세그멘트에 대해 물리페이지 할당
      - 페이지 폴트 발생 시, 강제 종료(kill)
    - Demand Paging을 위해 요청한 페이지에 대해서만 물리페이지 할당을 수행
      - 페이지 폴트 발생 시, 해당 vm_entry의 존재 유무 확인
      - vm_entry의 가상주소에 해당하는 물리페이지 할당
      - vm_entry의 정보를 참조하여, 디스크에 저장되어 있는 실제 데이터 로드

구현 후 Pintos 가상메모리

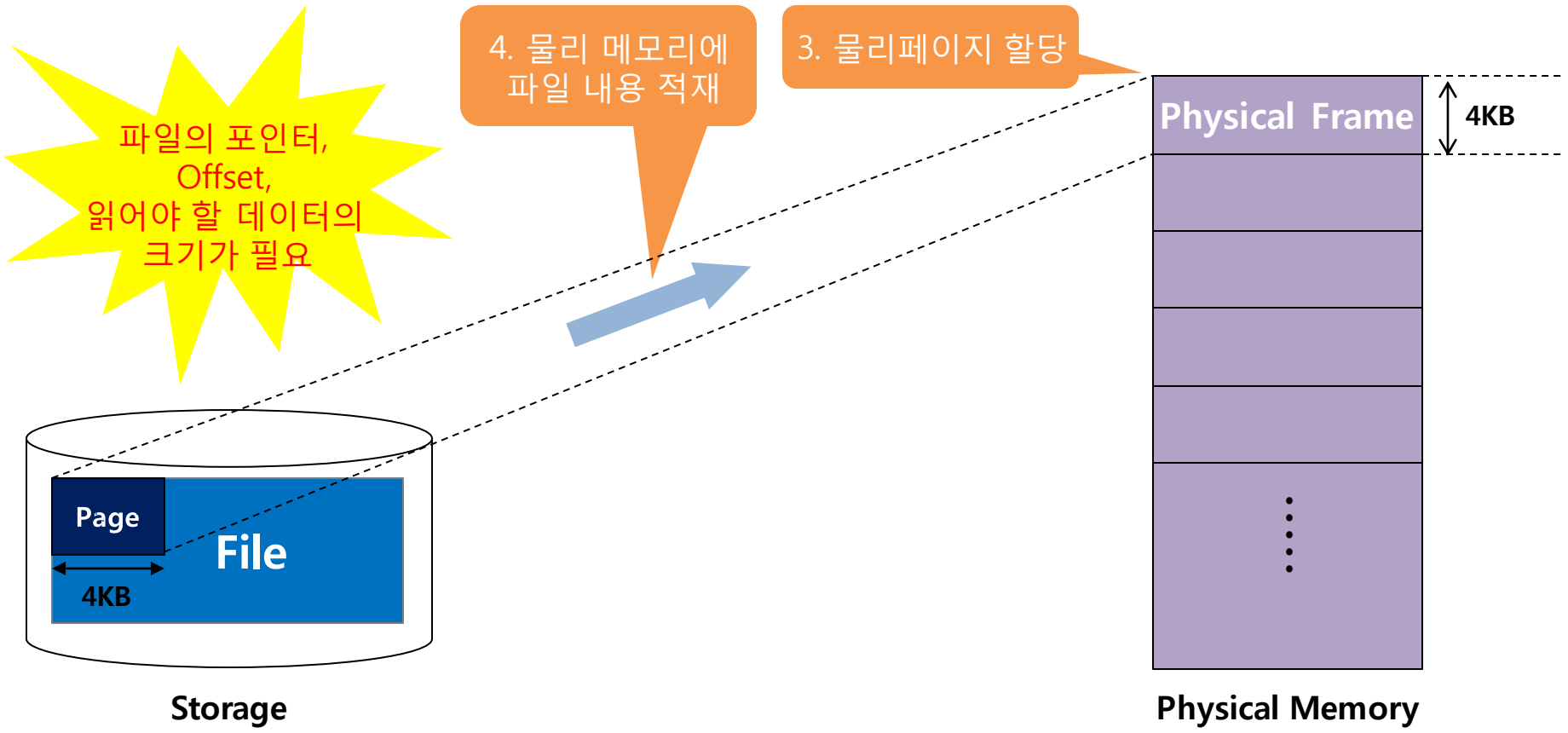


vm_entry

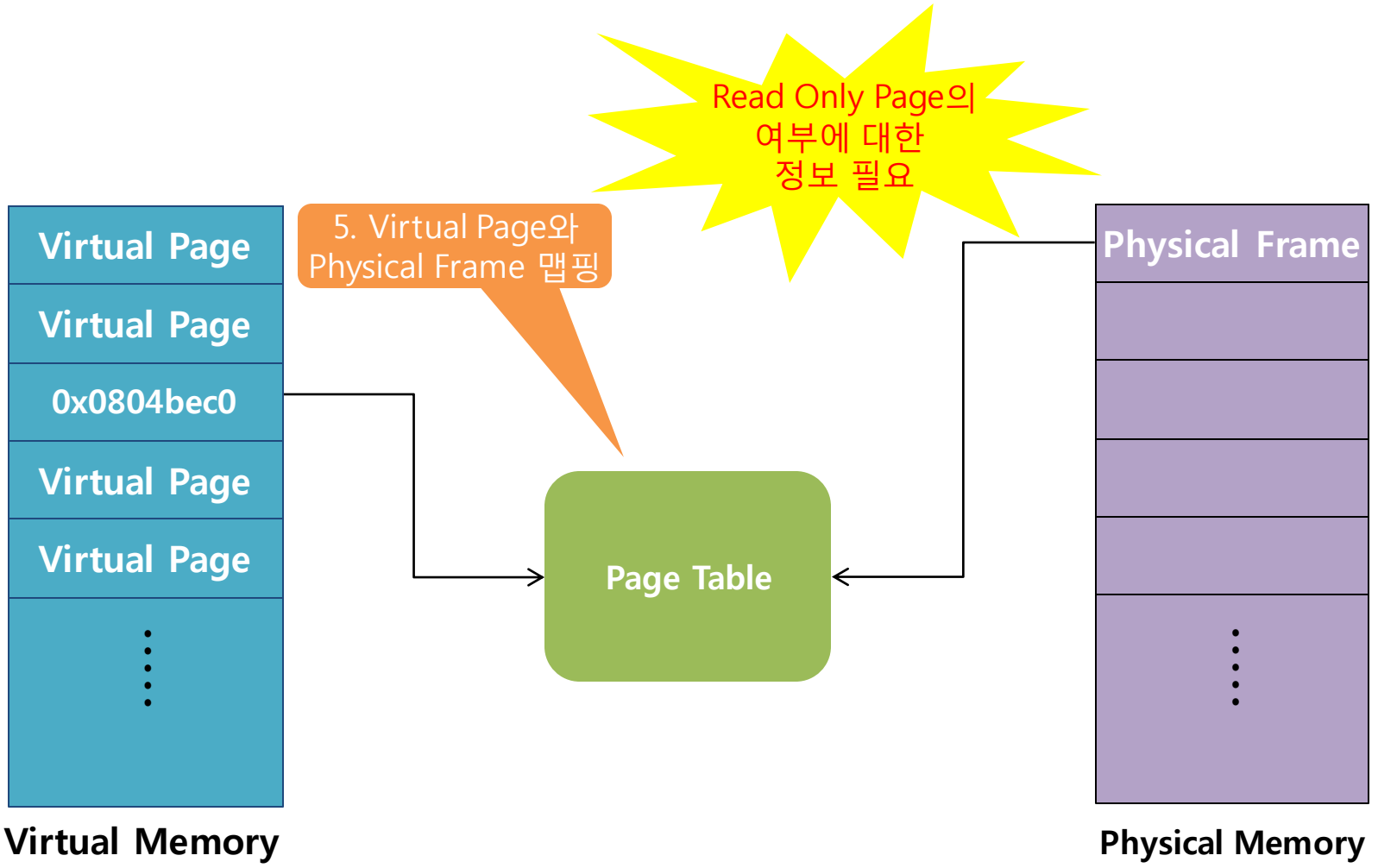
vm_entry 등장의 Motivation



vm_entry 등장의 Motivation (Cont.)

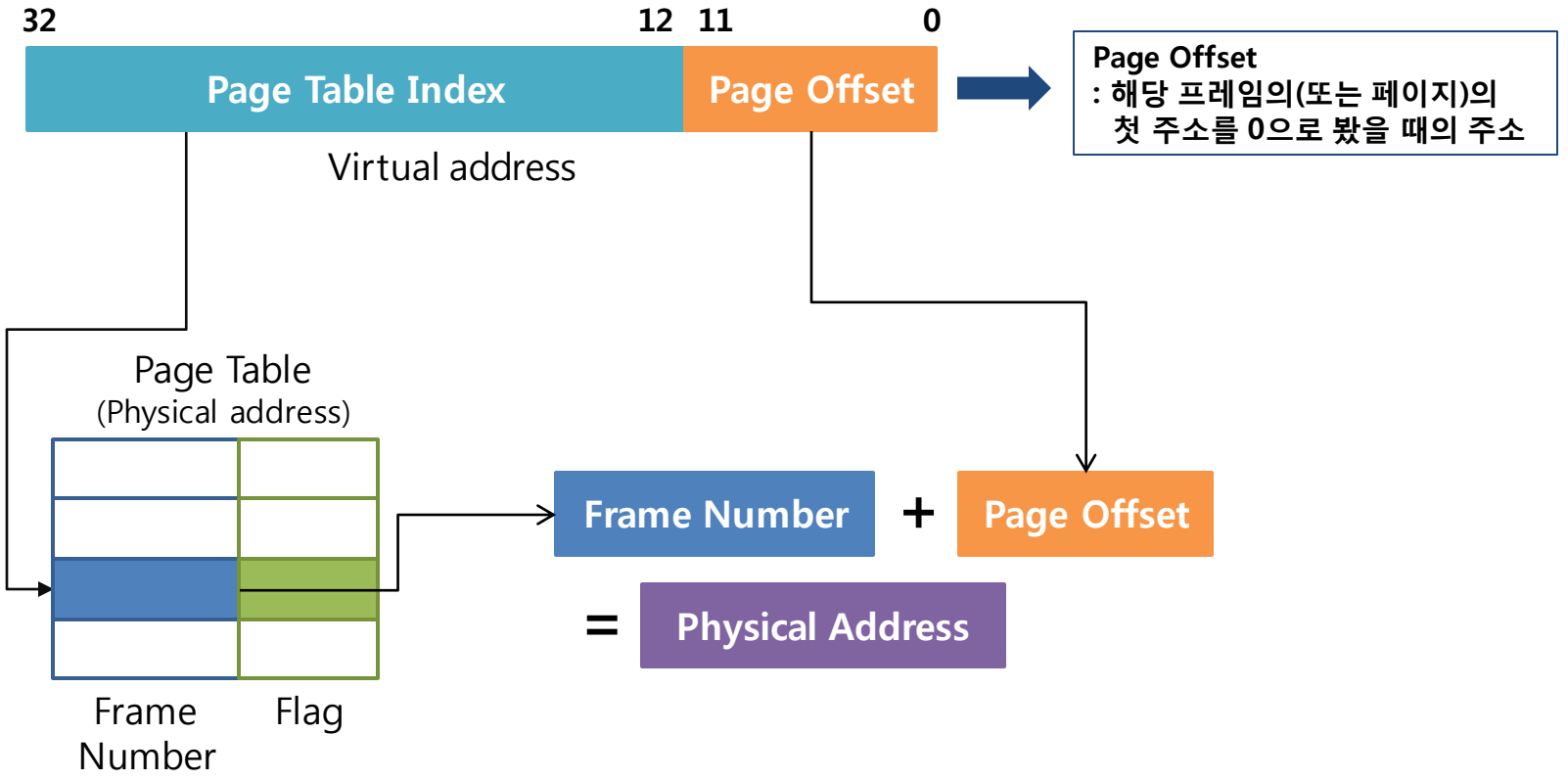


vm_entry 등장의 Motivation (Cont.)

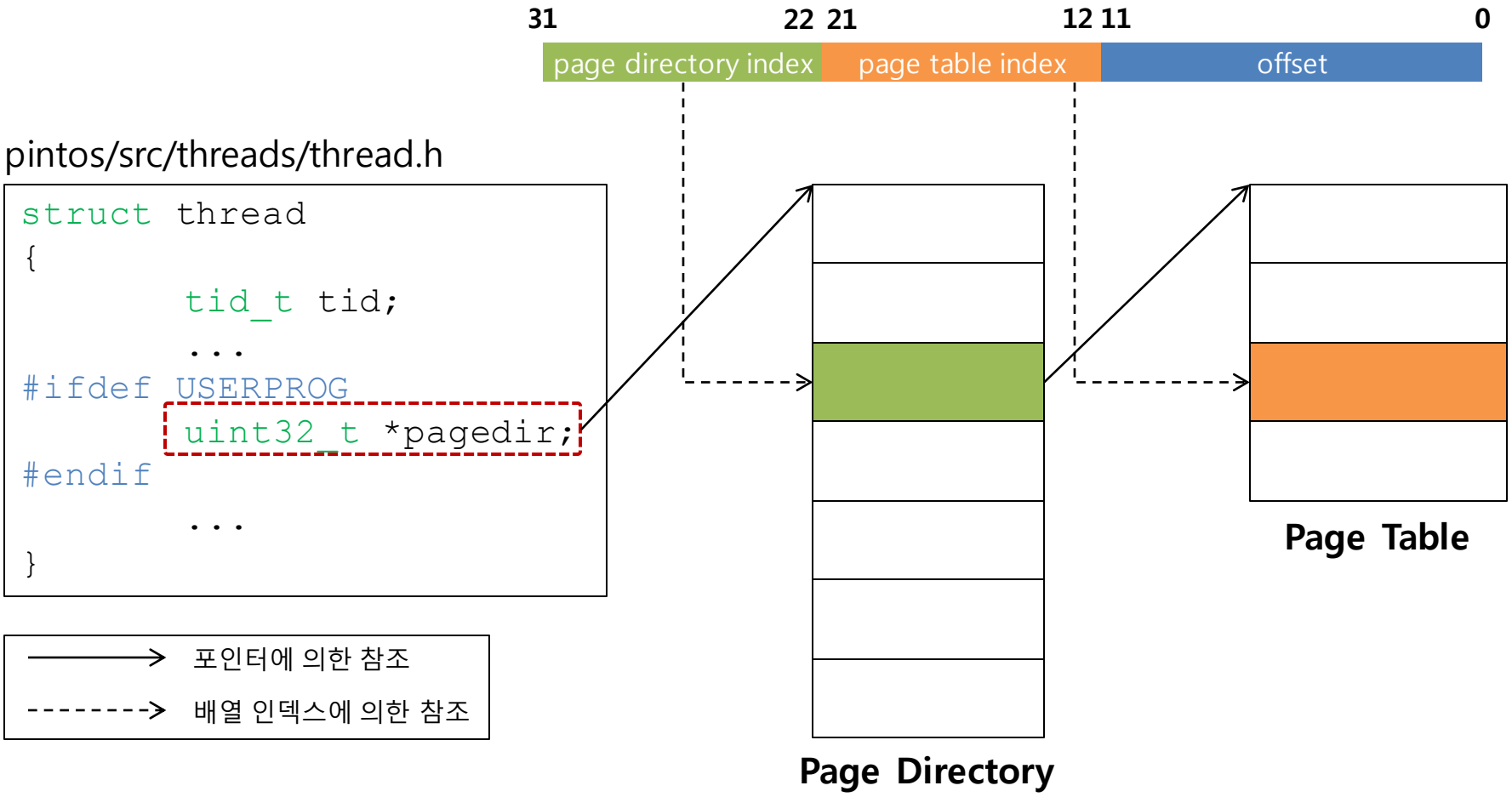


물리페이지의 할당 시에 파일의 Offset에 대한 정보를 읽을 필요가 있음

가상 주소 변환(Translation) 과정



Pintos의 페이지 테이블 구조 (2단계 페이지테이블 구조사용)



vm_entry 의 타입

- 가상 주소 페이지를 3가지 타입으로 분류
  - `vm_entry`의 `type` 필드에 가상 주소의 타입을 저장
  - `VM_BIN`: 바이너리 파일로 부터 데이터를 로드
  - `VM_FILE`: 매핑된 파일로 부터 데이터를 로드
    - Memory Mapped File 과제에서 다룰 예정
  - `VM_ANON`: 스왑영역으로 부터 데이터를 로드
    - Swapping 과제에 다룰 예정

pintos/src/vm/page.h

```
#define VM_BIN          0
#define VM_FILE         1
#define VM_ANON         2
```

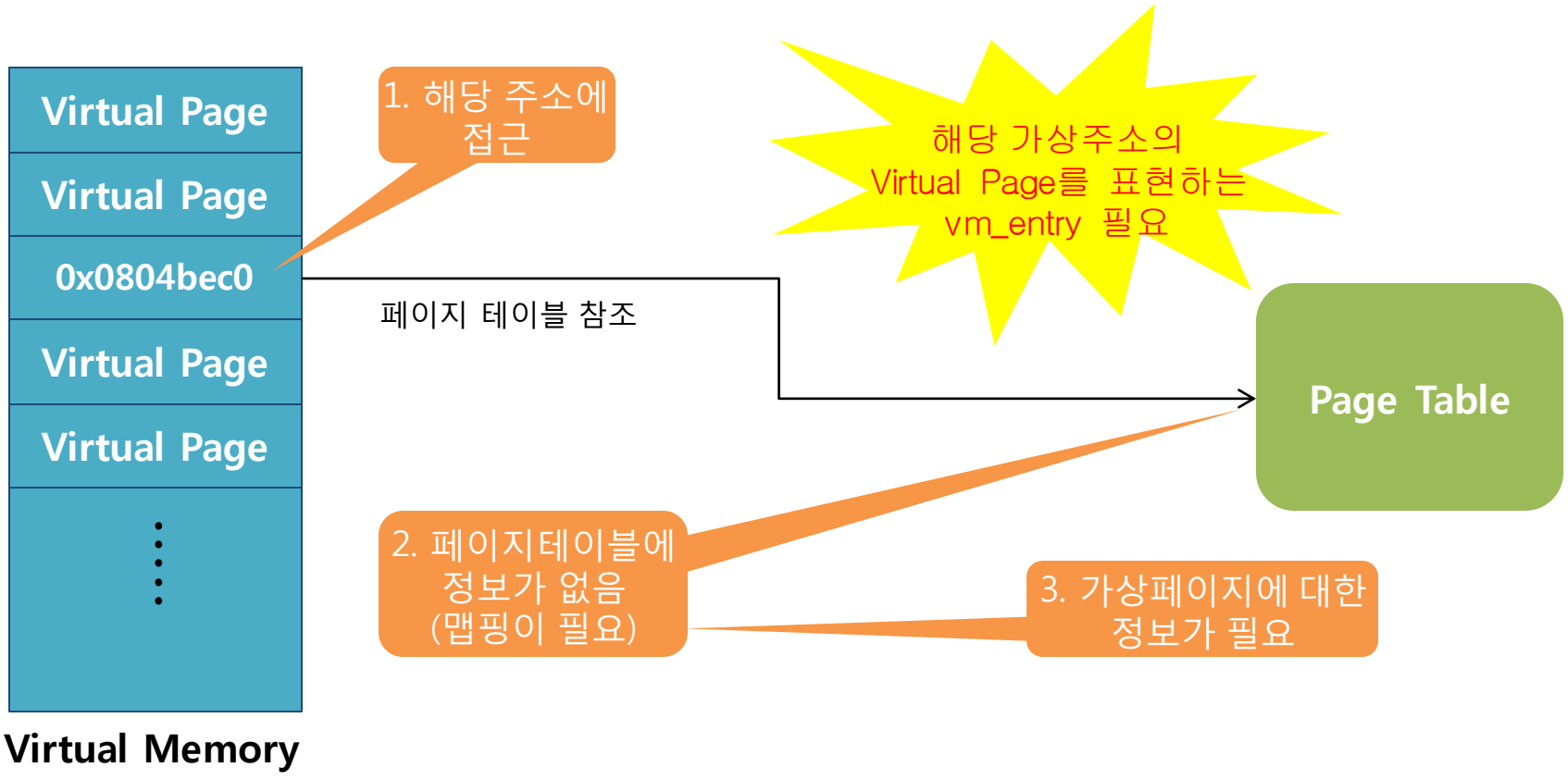
vm_entry 자료구조 정의

pintos/src/vm/page.h

```
struct vm_entry{
    uint8_t type;                   /* VM_BIN, VM_FILE, VM_ANON의 타입 */
    void *vaddr;                    /* vm_entry가 관리하는 가상페이지 번호 */
    bool writable;                  /* True일 경우 해당 주소에 write 가능
                                       False일 경우 해당 주소에 write 불가능 */
    bool is_loaded;                 /* 물리메모리의 탑재 여부를 알려주는 플래그 */
    struct file* file;              /* 가상주소와 맵핑된 파일 */

    /* Memory Mapped File 에서 다룰 예정 */
    struct list_elem mmap_elem; /* mmap 리스트 element */

    size_t offset;                  /* 읽어야 할 파일 오프셋 */
    size_t read_bytes;              /* 가상페이지에 쓰여져 있는 데이터 크기 */
    size_t zero_bytes;              /* 0으로 채울 남은 페이지의 바이트 */

    /* Swapping 과제에서 다룰 예정 */
    size_t swap_slot;               /* 스왑 슬롯 */

    /* 'vm_entry들을 위한 자료구조' 부분에서 다룰 예정 */
    struct hash_elem elem;      /* 해시 테이블 Element */
}
```

vm_entry들을 관리하기 위한 자료구조

vm_entry 관리 Motivation



즉 페이지 폴트가 일어날 때마다
가상 주소에 해당하는 vm_entry를 탐색해야 함

vm_entry 들의 관리

- 메모리 접근시 해당 주소의 가상페이지를 표현하는 vm_entry 탐색
  - vm_entry들은 탐색이 가능하도록 묶어서 관리되어야 함
- 탐색이 빠른 해시로 vm_entry관리
  - vaddr으로 해시 값 추출

Pintos에서의 해시 테이블

- Pintos는 체이닝 해시 테이블을 제공
  - 체이닝 해시 테이블
    - 해시 값이 충돌할 경우, 충돌한 해시 값의 element들을 리스트로 관리
  - src/lib/kernel/hash.* 에 자료구조와 해시테이블을 관리하는 함수 정의

Pintos가 제공하는 해시테이블 인터페이스

- `bool hash_init (struct hash *h ,hash_hash_func *, hash_less_func *, void *aux)`
  - 해시 테이블을 초기화 해주는 함수
  - h : 초기화 할 Hash table
  - hash_hash_func : 해시값을 구해주는 함수의 포인터
  - hash_less_func : 해시 element 들의 크기를 비교해주는 함수의 포인터
    - hash_find()에서 사용
- `void hash_destroy (struct hash *, hash_action_func *)`
  - 해시 테이블을 삭제하는 함수
  - hash_action_func : hash bucket의 entry를 삭제 해주는 함수.
- `struct hash_elem *hash_insert (struct hash *, struct hash_elem *)`
  - 해시 테이블에 Element 삽입
- `struct hash_elem *hash_delete (struct hash *, struct hash_elem *)`
  - 해시 테이블에서 Element 제거
- `struct hash_elem *hash_find (struct hash *, struct hash_elem *)`
  - 해시 테이블에서 Element 검색

해시테이블을 이용해서 구현해야 할 부분

- thread 구조체에 해시 테이블 자료구조 추가
- 프로세스 생성시
  - 해시 테이블 초기화
  - vm_entry들을 해시 테이블에 추가
- 프로세스 실행 중
  - 페이지 폴트가 발생 시, vm_entry를 해시 테이블에서 탐색
- 프로세스 종료시
  - 해시테이블의 버킷리스트와 vm_entry들 제거

thread 구조체에 해시 테이블 자료구조 추가

`struct thread`

프로세스마다 가상주소 공간이 할당되므로, 가상페이지들을 관리할 수 있는 자료구조인 해시테이블 정의

pintos/src/threads/thread.h

```
struct thread{
    /* Owned by thread.c. */
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    ...
    /* Owned by thread.c. */
    unsigned magic;                     /* Detects stack overflow. */

    struct hash vm;   /* 스레드가 가진 가상 주소 공간을 관리하는 해시테이블 */
}
```

해시 테이블 초기화 함수 구현

- `void vm_init (struct hash *vm)`
  - `hash_init()` 함수를 사용하여 해시 테이블 초기화
    - hash_init()함수 설명 참조
- `static unsigned vm_hash_func (const struct hash_elem *e, void *aux)`
  - `vm_entry`의 `vaddr`을 인자값으로 `hash_int()` 함수를 사용하여 해시 값 반환
- `static bool vm_less_func (const struct hash_elem *a, const struct hash_elem *b, void *aux)`
  - 입력된 두 hash_elem의 vaddr 비교
    - a의 vaddr이 b보다 작을 시 true 반환
    - a의 vaddr이 b보다 클 시 false 반환

해시 테이블 초기화 및 제거 및 해시 함수 구현 (Cont.)

pintos/src/vm/page.c

```
void vm_init (struct hash *vm)
{
        /* hash_init()으로 해시테이블 초기화 */
        /* 인자로 해시 테이블과 vm_hash_func과 vm_less_func 사용 */
}
```

pintos/src/vm/page.c

```
static unsigned vm_hash_func (const struct hash_elem *e,void *aux)
{
        /* hash_entry()로 element에 대한 vm_entry 구조체 검색 */
        /* hash_int()를 이용해서 vm_entry의 멤버 vaddr에 대한 해시값을
        구하고 반환 */
}
```

pintos/src/vm/page.c

```
static bool vm_less_func (const struct hash_elem *a, const struct
hash_elem *b)
{
        /* hash_entry()로 각각의 element에 대한 vm_entry 구조체를 얻은
        후 vaddr 비교 (b가 크다면 true, a가 크다면 false */
}
```

해시 테이블 초기화 코드 추가

pintos/src/userprog/process.c

```
static void start_process (void *file_name_)
{
        ...

        /* vm_init()함수를 이용해서 해시테이블 초기화 */

        /* Initialize interrupt frame and load executable */
        memset (&if_, 0, sizeof if_);
        ...
}
```

해시 테이블에 element를 삽입 및 제거 함수 구현

- `bool insert_vme (struct hash *vm, struct vm_entry *vme)`
  - hash_insert() 함수를 이용하여 vm_entry를 해시 테이블에 삽입
  - 삽입 성공 시 true 반환
  - 실패 시 false 반환
- `bool delete_vme (struct hash *vm, struct vm_entry *vme)`
  - hash_delete() 함수를 이용하여 vm_entry를 해시 테이블에서 제거

해시 테이블에 element를 삽입 및 제거 함수 구현 (Cont.)

pintos/src/vm/page.c

```
bool insert_vme (struct hash *vm, struct vm_entry *vme)
{
        /* hash_insert()함수 사용 */
}
```

pintos/src/vm/page.c

```
bool delete_vme (struct hash *vm, struct vm_entry *vme)
{
        /* hash_delete()함수 사용 */
}
```

'가상 주소공간 초기화' 부분에서 사용 예정

해시 테이블 내 vm_entry 검색 함수 구현

- `struct vm_entry *find_vme (void *vaddr)`
  - 인자로 받은 vaddr에 해당하는 vm_entry를 검색 후 반환
    - 가상 메모리 주소에 해당하는 페이지 번호 추출 (pg_round_down())
    - hash_find() 함수를 이용하여 vm_entry 검색 후 반환

해시테이블 인터페이스 구현

pintos/src/vm/page.c

```
struct vm_entry *find_vme (void *vaddr)
{
        /* pg_round_down()으로 vaddr의 페이지 번호를 얻음 */
        /* hash_find() 함수를 사용해서 hash_elem 구조체 얻음 */
        /* 만약 존재하지 않는다면 NULL 리턴 */
        /* hash_entry()로 해당 hash_elem의 vm_entry 구조체 리턴 */
}
```

'요구 페이징 구현' 부분에서 사용 예정

해시 테이블 제거 함수 구현

- `void vm_destroy (struct hash *vm)`
  - `hash_destroy()` 함수를 사용하여 해시 테이블의 버킷리스트와 vm_entry들을 제거

pintos/src/vm/page.c

```
void vm_destroy (struct hash *vm)
{
    /* hash_destroy()으로 해시테이블의 버킷리스트와 vm_entry들을 제거 */
}
```

process_exit() 함수 수정

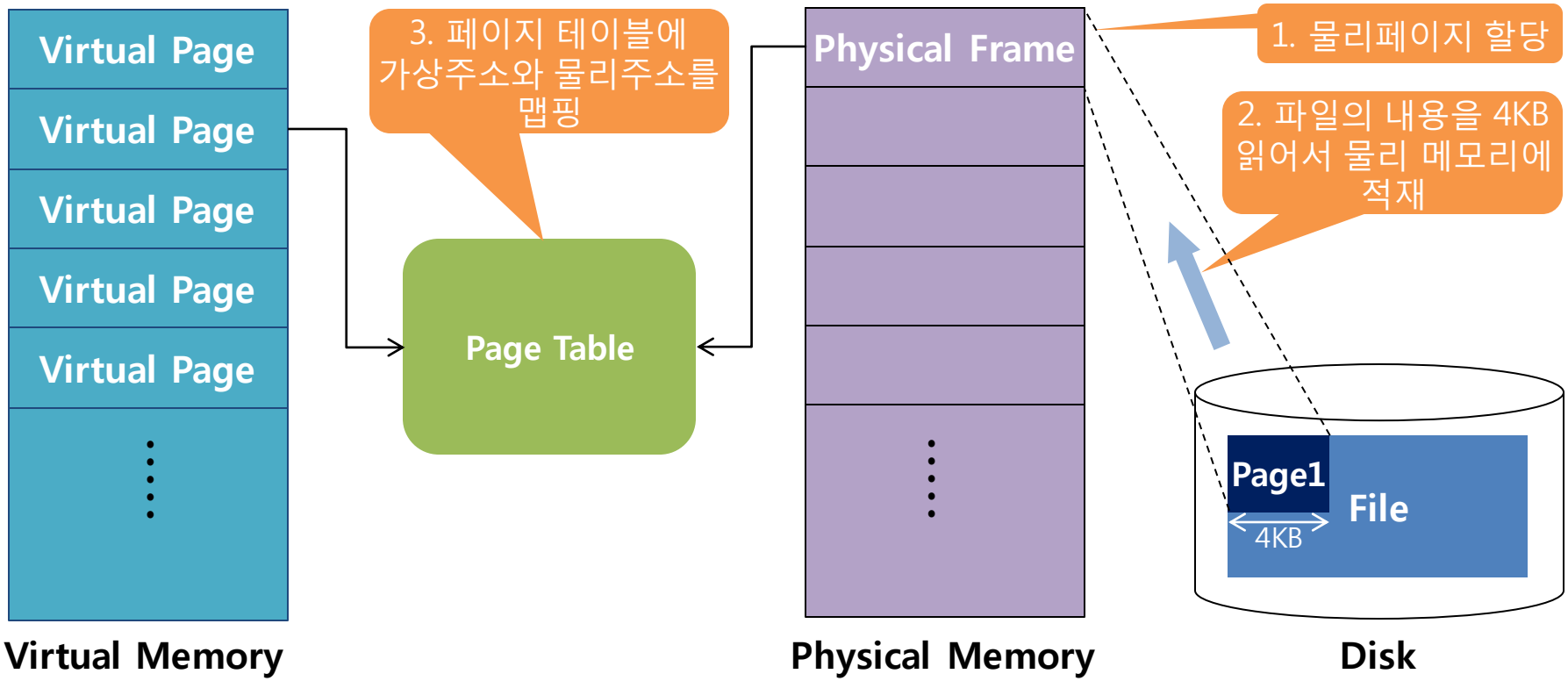
- 프로세스 종료 시 vm_entry들을 제거하도록 코드 수정

pintos/src/userporg/process.c

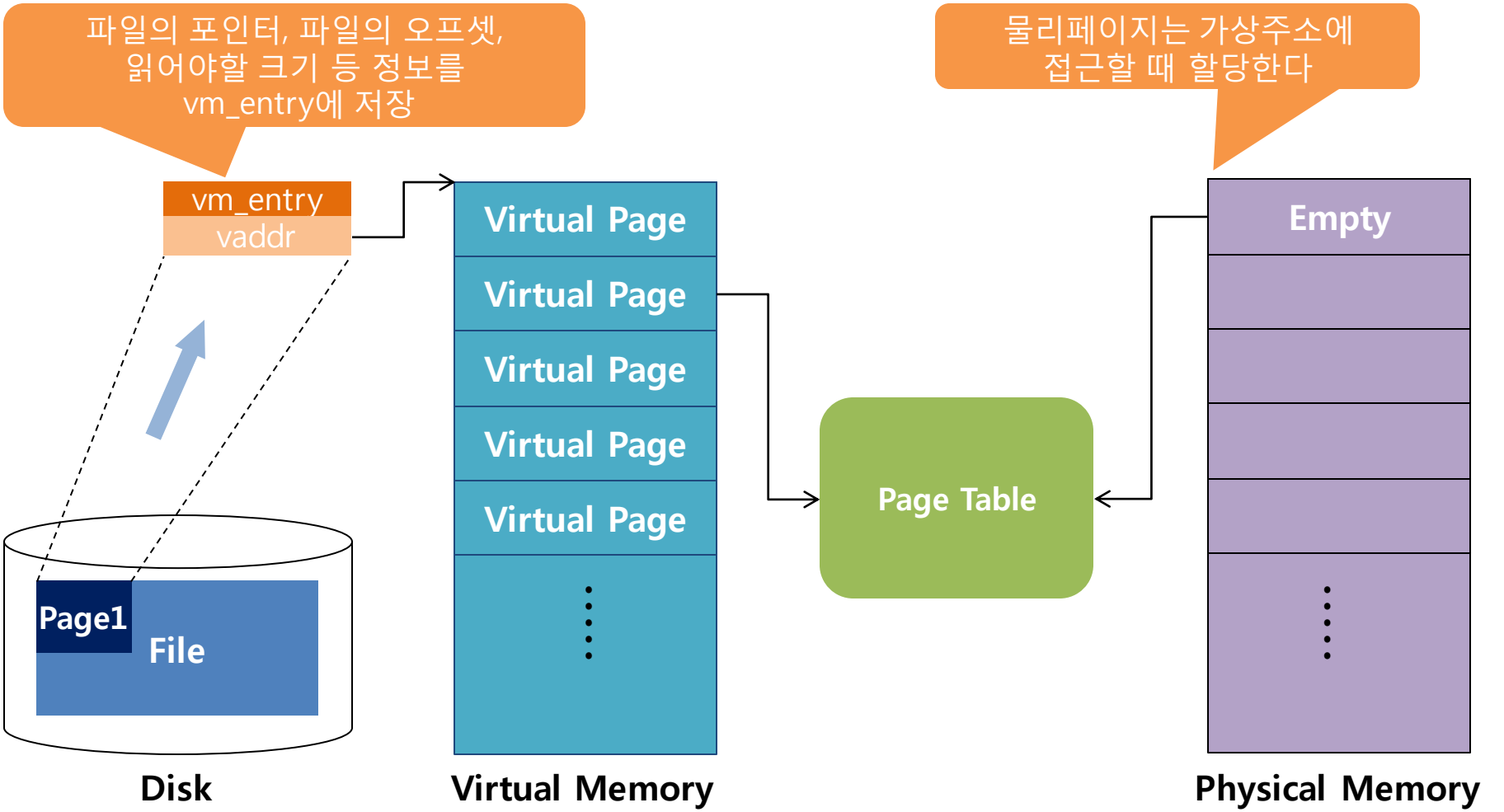
```
void process_exit (void){
    struct thread *cur = thread_current();
    uint32_t *pd;
    ...
    palloc_free_page(cur -> fd);
    /* vm_entry들을 제거하는 함수 추가 */
    pd = cur->pagedir;
    ...
}
```

가상 주소공간 초기화

가상 주소공간 초기화 구현 전 Pintos



가상 주소공간 초기화 구현 후 Pintos



가상주소 접근 시, 물리페이지가 맵핑되어 있지 않다면 해당 가상 주소에 해당하는 vm_entry 탐색 후 vm_entry 정보들을 참조하여 디스크의 데이터를 읽어 물리프레임에 적재

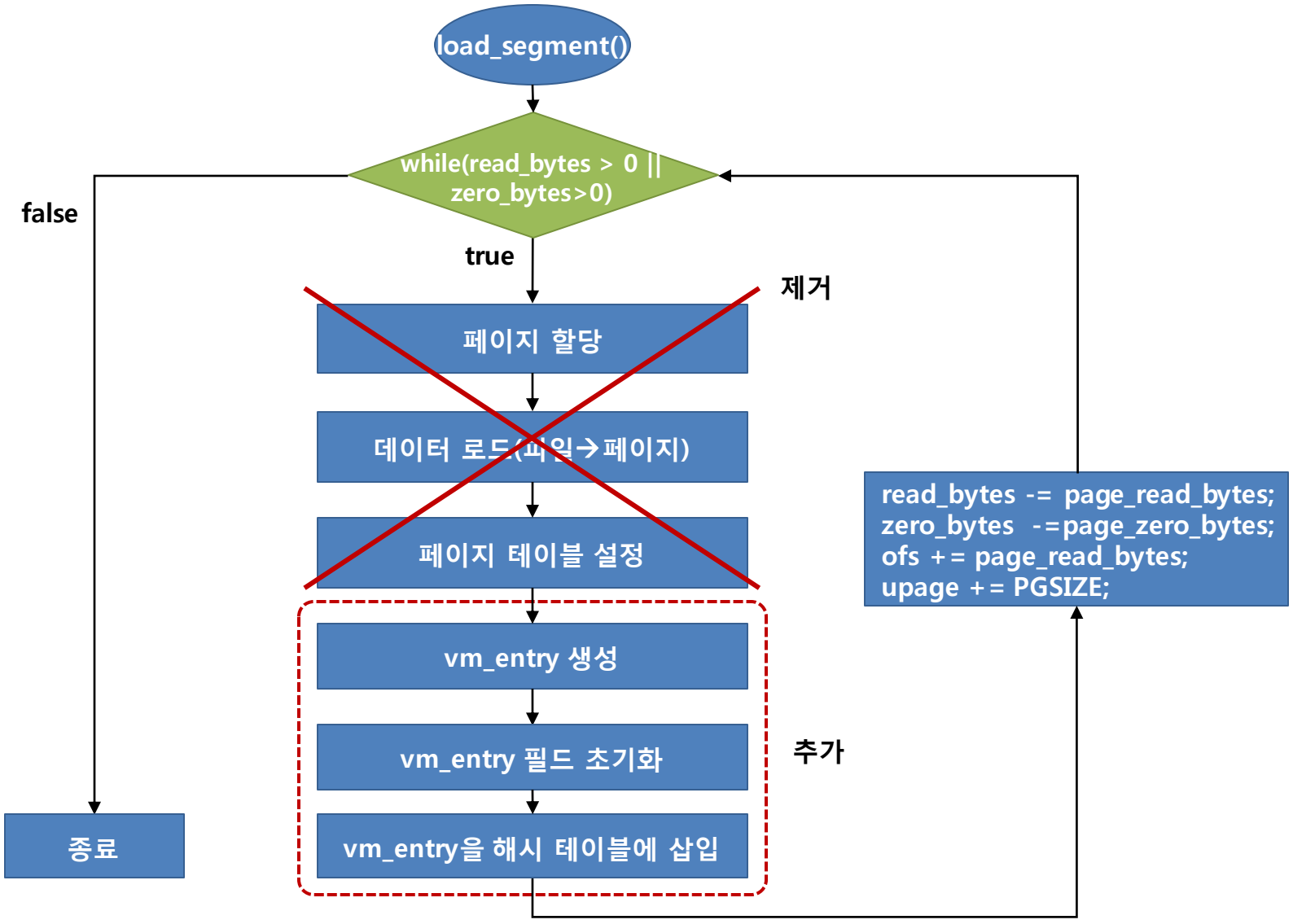
주소 공간 초기화 관련 함수 수정 (ELF 세그먼트)

- pintos/src/userprog/process.c

```
static bool load_segment(struct file *file, off_t ofs,
uint8_t *upage, uint32_t read_bytes, uint32_t zero_bytes,
bool writable)
```

  - ELF포맷 파일의 세그먼트를 프로세스 가상주소공간에 탑재하는 함수이다.
  - 이 함수에 프로세스 가상메모리 관련 자료구조를 초기화하는 기능을 추가한다.
    - 프로세스 가상주소공간에 메모리를 탑재하는 부분을 제거하고, vm_entry 구조체의 할당, 필드값 초기화, 해시 테이블 삽입을 추가한다.

load_segment() 함수 수정



load_segment() 함수 수정 (Cont.)

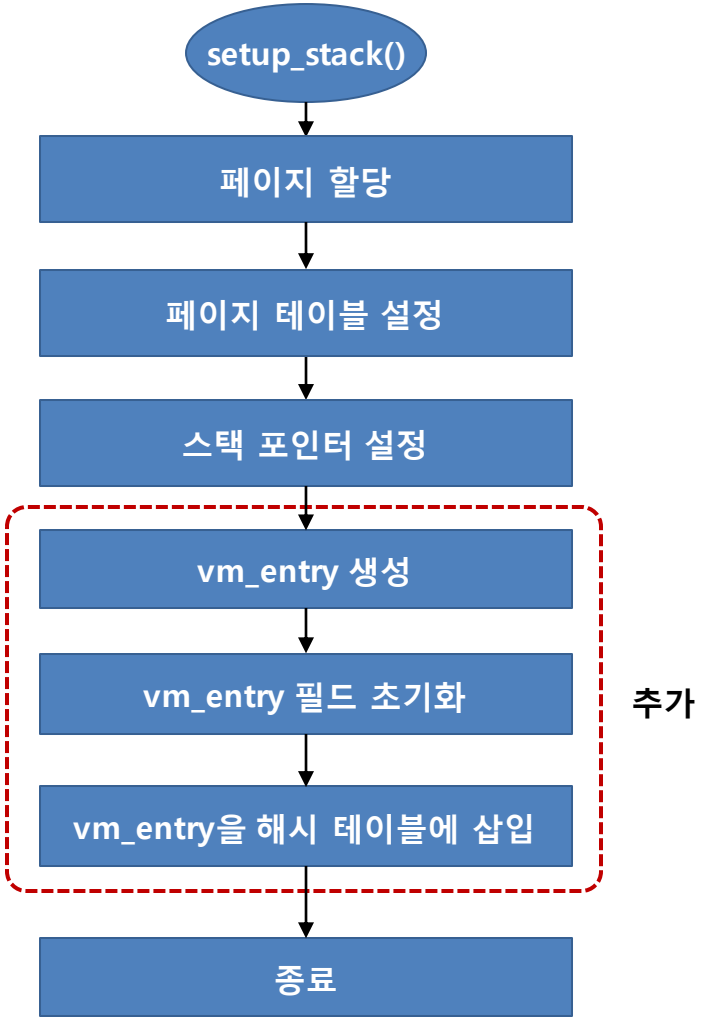
pintos/src/userprog/process.c

```
static bool load_segment (struct file *file, off_t ofs, uint8_t *upage,
                  uint32_t read_bytes, uint32_t zero_bytes, bool writable)
{
        ...
        while (read_bytes > 0 || zero_bytes > 0)
        {
                size_t page_read_byters = read_bytes < PGSIZE
                                                ? read_bytes : PGSIZE;
                size_t page_zero_bytes = PGSIZE - page_read_bytes;

                .....

                /* vm_entry 생성 (malloc 사용) */
                /* vm_entry 멤버들 설정, 가상페이지가 요구될 때 읽어야할 파일의 오프
                        셋과 사이즈, 마지막에 패딩할 제로 바이트 등등 */
                /* insert_vme() 함수를 사용해서 생성한 vm_entry를 해시테이블에 추
                        가 */

                read_bytes -= page_read_bytes;
                zero_bytes -= page_zero_bytes;
                ofs += page_read_bytes;
                upage += PGSIZE;
        }
```

물리 페이지를 할당하고 맵핑하는 부분 삭제

스택 초기화 함수 수정

- 기존
  - 단일 페이지 할당
  - 페이지 테이블 설정
  - 스택 포인터 설정(esp)
- 추가할 부분
  - 4KB 스택의 `vm_entry` 생성
  - 생성한 `vm_entry` 필드값 초기화
  - `vm` 해시 테이블에 삽입



setup_stack() 함수 수정

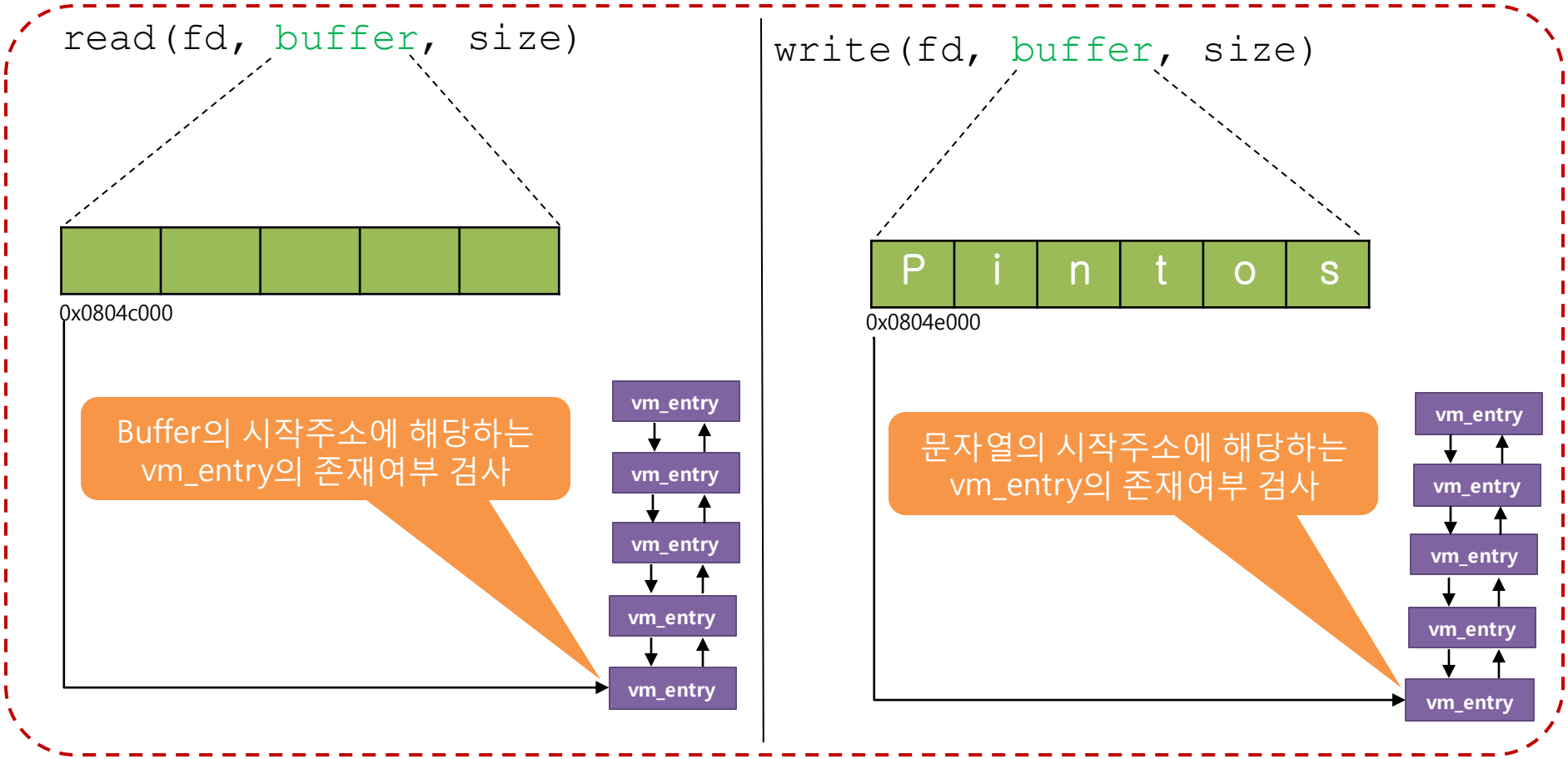
pintos/src/vm/page.c

```
static bool setup_stack (void **esp)
{
        ...
        if (kpage != NULL)
        {
                ...
        }
        /* vm_entry 생성 */
        /* vm_entry 멤버들 설정 */
        /* insert_vme() 함수로 해시테이블에 추가 */

        ...
}
```

가상주소 유효성 검사

주소 유효성 검사란?

- 가상주소에 해당하는 vm_entry가 존재하는지 검사
  - 시스템 콜(read/write) 사용 시 인자로 주어지는 문자열이나 Buffer의 주소에 해당하는 vm_entry가 존재하는지 검사



check_address() 함수 수정

- 기존 check_address() 함수는 esp에 대한 유저 메모리 영역 체크
- vm_entry를 사용하여 유효성 검사 작업을 수행하도록 코드 수정
- vm_entry를 반환하도록 코드 수정

pintos/src/userporg/syscall.c

```
struct vm_entry * check_address (void* addr, void* esp /*Unused*/)
{

        if(addr < (void *)0x08048000 || addr >= (void *)0xc0000000)
        {
                exit(-1);
        }
        /*addr이 vm_entry에 존재하면 vm_entry를 반환하도록 코드 작성 */
        /*find_vme() 사용*/
}
```

check_valid_buffer() 함수 구현

```
void check_valid_buffer (void* buffer, unsigned size,
                         void* esp, bool to_write)
```

- Buffer를 사용하는 read() system call의 경우 buffer의 주소가 유효한 가상주소인지 아닌지 검사할 필요성이 있음
- Buffer의 유효성을 검사하는 함수
- Check_valid_buffer 구현시 check_address()함수를 사용
- to_write 변수는 buffer에 내용을 쓸 수 있는지 없는지 검사하는 변수

pintos/src/userporg/syscall.c

```
case SYS_READ :
        ...
        /* 기존 check_address(/*인자삽입*/)함수는 삭제*/
        /*check_valid_buffer(/*인자삽입*/)함수 구현*/
        …
        break;
```

check_valid_string() 함수 구현

```
void check_valid_string (const void* str, void* esp)
```

- System call에서 사용할 인자의 문자열의 주소값이 유효한 가상주소인지 아닌지 검사하는 함수
- check_valid_string 구현시 check_address()함수를 사용

Characteristic	Read()	Write()
유효성	검사함	검사함
to_write	사용함	사용하지 않음
사용 함수	check_valid_buffer	check_valid_string

check_valid_buffer() 함수 추가

pintos/src/vm/page.c

```
void check_valid_buffer (void *buffer, unsigned size, void *esp,
                                    bool to_write)
{
        /* 인자로 받은 buffer부터 buffer + size까지의 크기가 한 페이지의
        크기를 넘을 수도 있음 */

        /* check_address를 이용해서 주소의 유저영역 여부를 검사함과 동시
        에 vm_entry 구조체를 얻음 */

        /* 해당 주소에 대한 vm_entry 존재여부와 vm_entry의 writable 멤
        버가 true인지 검사 */

        /* 위 내용을 buffer 부터 buffer + size까지의 주소에 포함되는
        vm_entry들에 대해 적용 */
}
```

check_valid_string() 함수 추가

pintos/src/vm/page.c

```
void check_valid_string (const void *str, void *esp)
{
        /* str에 대한 vm_entry의 존재 여부를 확인*/
        /* check_address()사용*/
}
```

sycall_handler() 함수 수정

- 함수 내 check_address()함수의 인자값 변경
  - check_address()함수의 두 번째 인자값은 f->esp
- 시스템 콜 호출 시 인자값의 유효성 검사를 하도록 코드 수정

pintos/src/userporg/syscall.c

```
static void syscall_handler(struct intr_frame *f UNUSED){
    ...
    check_address(esp,esp); /* 인자값 변경 */
    ...
    switch(syscall_n){
        ...
        case SYS_EXEC :
        get_argument(esp , arg , 1);
        /* buffer 사용 유무를 고려하여 유효성 검사를 하도록 코드 추가 */
        f -> eax = exec((const char *)arg[0]);
        break;
        ...
```

sycall_handler() 함수 수정

pintos/src/userporg/syscall.c

```
...
case SYS_OPEN :
get_argument(sp, arg , 1);
/* buffer 사용 유무를 고려하여 유효성 검사를 하도록 코드 추가 */
f -> eax = open((const char *)arg[0]);
break;
...
case SYS_READ :
get_argument(sp, arg , 3);
/* buffer 사용 유무를 고려하여 유효성 검사를 하도록 코드 추가 */
f -> eax = read(arg[0] , (void *)arg[1] , (unsigned)arg[2]);
break;
...
```

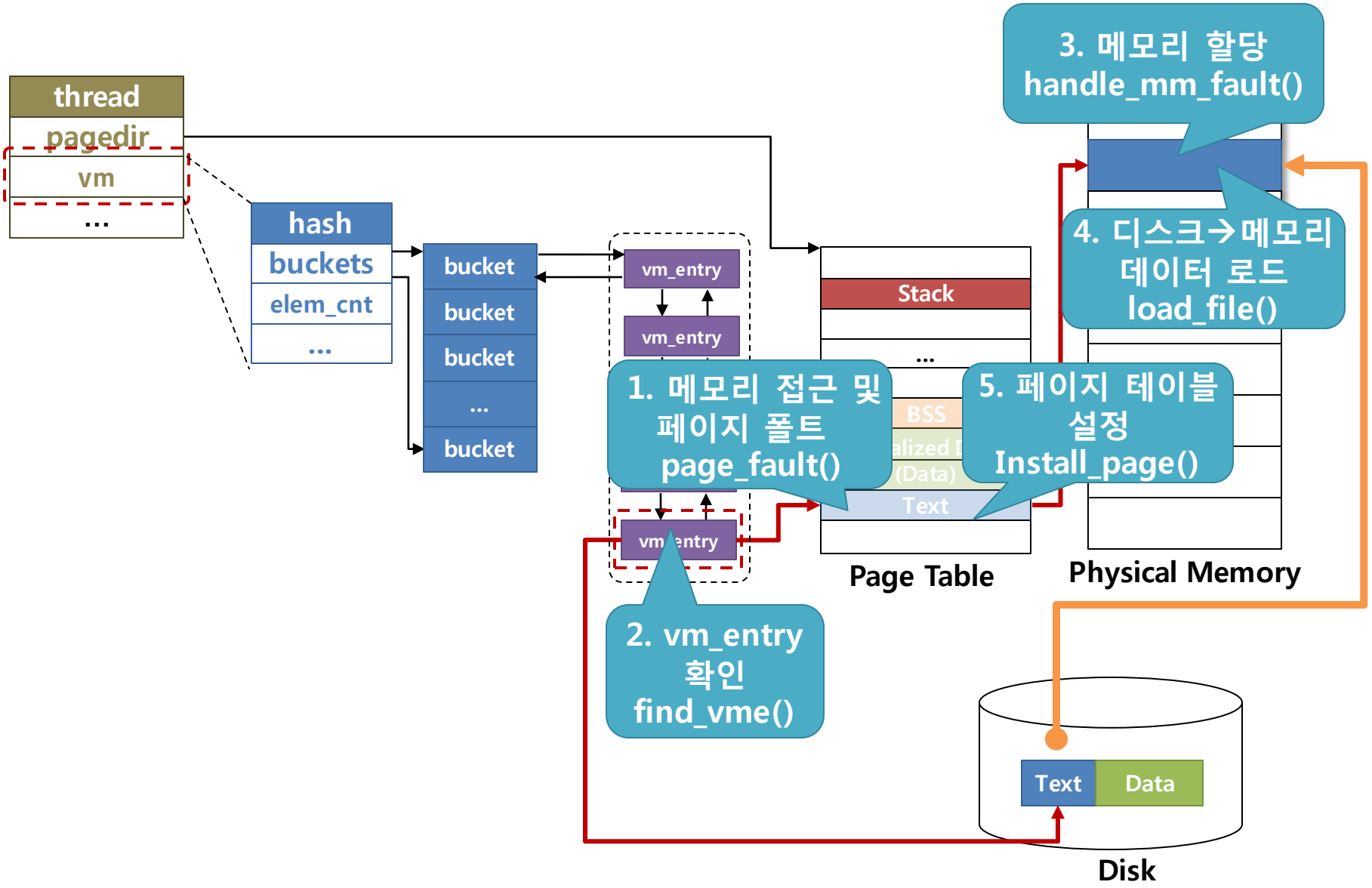
sycall_handler() 함수 수정

pintos/src/userporg/syscall.c

```
            ...
            case SYS_WRITE :
            get_argument(sp, arg , 3);
            /* buffer 사용 유무를 고려하여 유효성 검사를 하도록 코드 추가 */
            f -> eax = write(arg[0] , (const void *)arg[1] ,
                             (unsigned)arg[2]);

            break;
            ...
            default :
            thread_exit ();
        }
    }
```

요구 페이징 구현

요구 페이징 구현 순서


thread
pagedir
vm
...
hash
buckets
elem_cnt
...
bucket
bucket
bucket
...
bucket
vm_entry
vm_entry
vm_entry
Stack
...
BSS
(Data)
Text
Page Table
Physical Memory
3. 메모리 할당
handle_mm_fault()
4. 디스크→메모리
데이터 로드
load_file()
1. 메모리 접근 및
페이지 폴트
page_fault()
5. 페이지 테이블
설정
Install_page()
2. vm_entry
확인
find_vme()
Text
Data
Disk

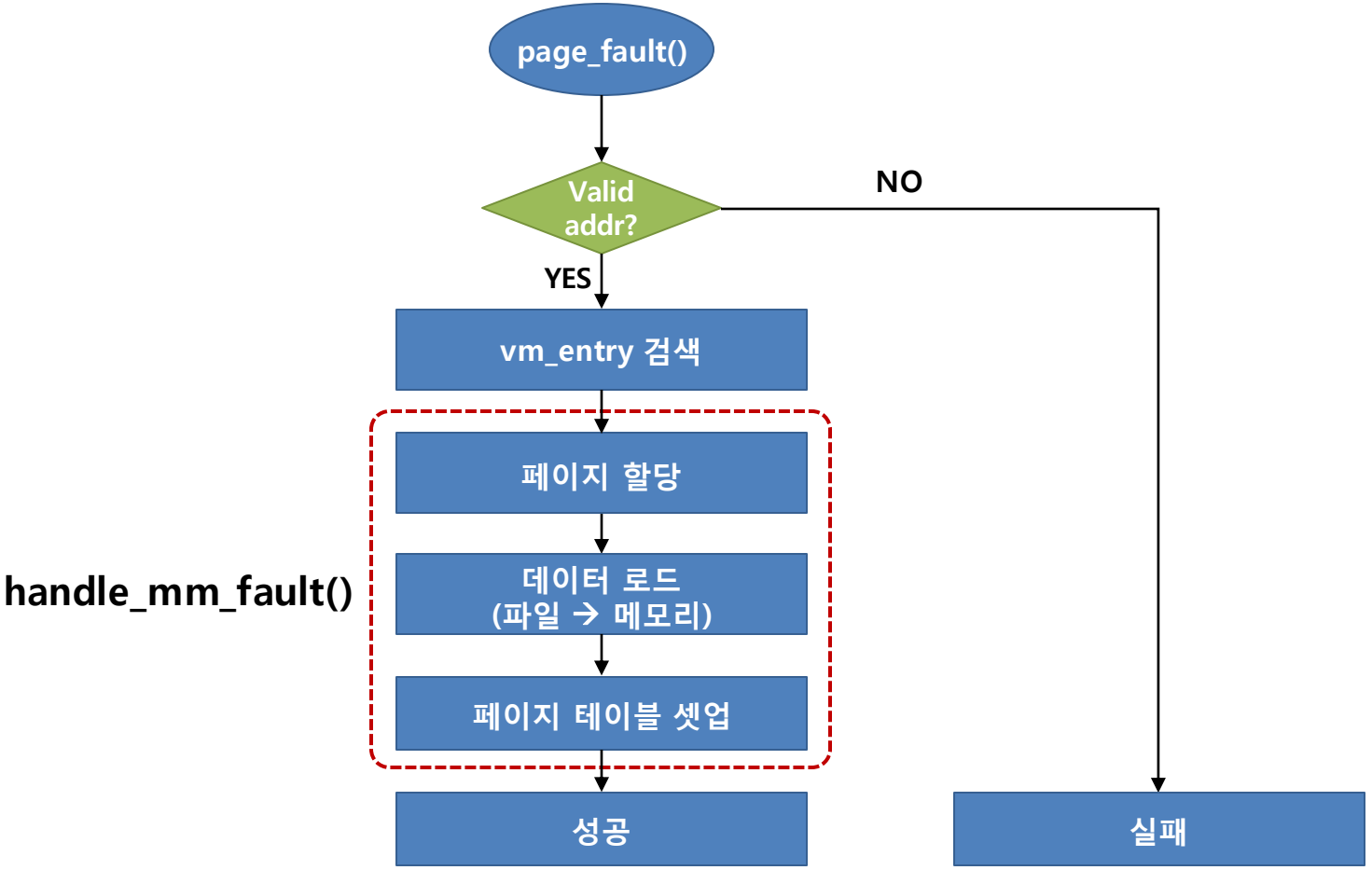
할일 1: 페이지 폴트 처리(Cont.)

Pintos는 페이지 폴트 발생시 처리를 위해 page_fault()함수 존재

- pintos/src/userprog/exception.c
  - `static void page_fault (struct intr_frame *f)`
  - 현재의 pintos의 page_fault 처리는 permission, 주소 유효성 검사 후 오류발생시 무조건 "segmentation fault"를 발생시키고 kill(-1)을 하여 종료
  - kill(-1)처리 관련 코드 삭제
  - fault_addr의 유효성 검사
  - 페이지폴트 핸들러 함수 호출
    - `handle_mm_fault(struct vm_entry *vme)`

할일1: 페이지 폴트 처리(Cont.)

- 페이지폴트 처리 알고리즘



할일 1: 페이지 폴트 처리(Cont.)

- vm_entry 검색 후 페이지를 할당하도록 코드 수정

pintos/src/userporg/exception.c

```
static void page_fault (struct intr_frame *f){
    ...
    /* Determine cause. */
    not_present = (f->error_code & PF_P) == 0;
    write = (f->error_code & PF_W) != 0;
    user = (f->error_code & PF_U) != 0;

    exit(-1);
    /* To implement virtual memory, delete the rest of the function
     body, and replace it with code that brings in the page to
     which fault_addr refers. */
    printf ("Page fault at %p: %s error %s page in %s context.\n",
            fault_addr,
            not_present ? "not present" : "rights violation",
            write ? "writing" : "reading",
            user ? "user" : "kernel");
    kill (f);
}
```

↳ 삭제 & 코드 구현

할일1: 페이지 폴트 처리(Cont.)

- vm_entry검색 후 페이지를 할당하도록 코드 수정

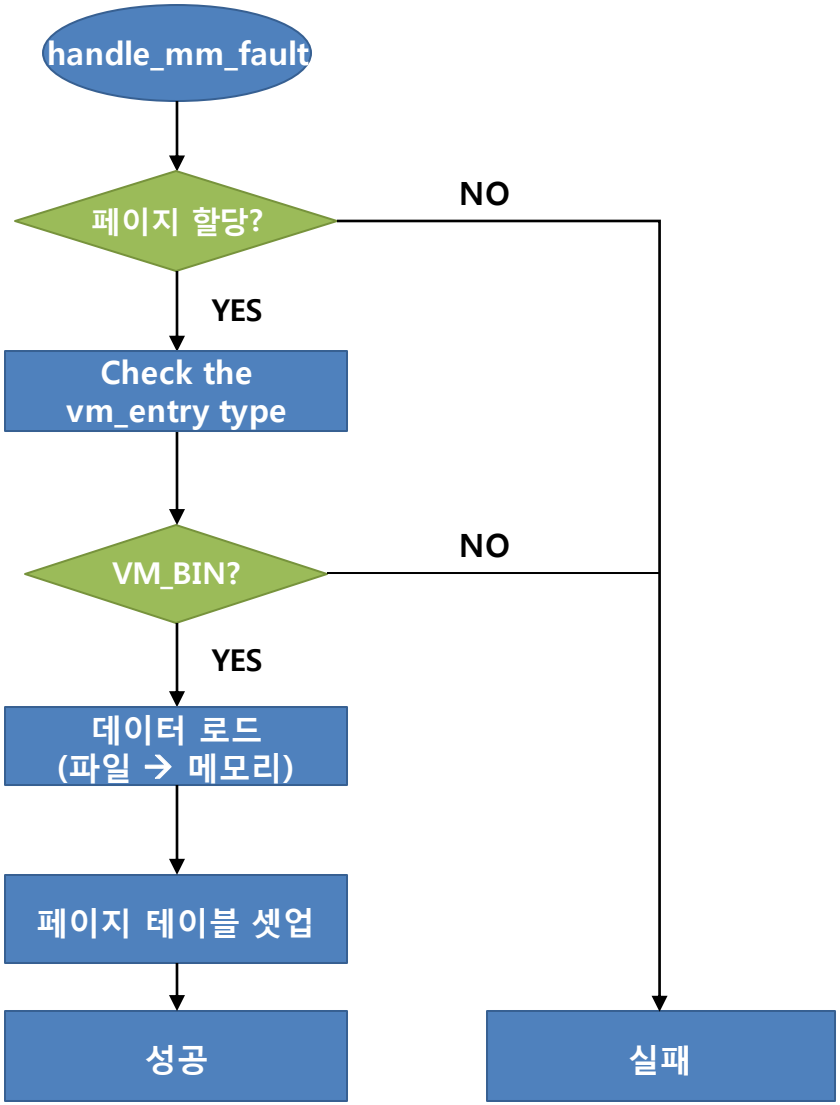
pintos/src/userporg/exception.c

```
static void page_fault (struct intr_frame *f)
{
        void *fault_addr; /*page_fault가 발생한 가상주소*/
        ...
        /* Determine cause. */
        not_present = (f->error_code & PF_P) == 0;
        write = (f->error_code & PF_W) != 0;
        user = (f->error_code & PF_U) != 0;
        /* read only 페이지에 대한 접근이 아닐 경우 (not_present 참조)*/
        /* 페이지 폴트가 일어난 주소에 대한 vm_entry 구조체 탐색 */
        /* vm_entry를 인자로 넘겨주며 handle_mm_fault() 호출 */

        /* 제대로 파일이 물리 메모리에 로드 되고 맵핑  됬는지 검사 */

        ...
}
```

할일2: 페이지폴트 핸들러 구현

- pintos/src/userprog/process.c
  - `bool handle_mm_fault(struct vm_entry *vme)`
    - `handle_mm_fault`는 페이지 폴트 발생시 핸들링을 위해 호출 되는 함수
    - 페이지 폴트 발생시 물리페이지를 할당
    - `Disk`에 있는 `file`을 물리페이지로 `load`
      - `load_file (void* kaddr, struct vm_entry *vme)`를 사용
    - 물리메모리에 적재가 완료되면 가상주소와 물리주소를 페이지테이블로 맵핑
      - `static bool install_page(void *upage, void *kpage,bool writable)`를 사용

할일2: 페이지폴트 핸들러 구현(Cont.)



할일2: 페이지폴트 핸들러 구현(Cont.)

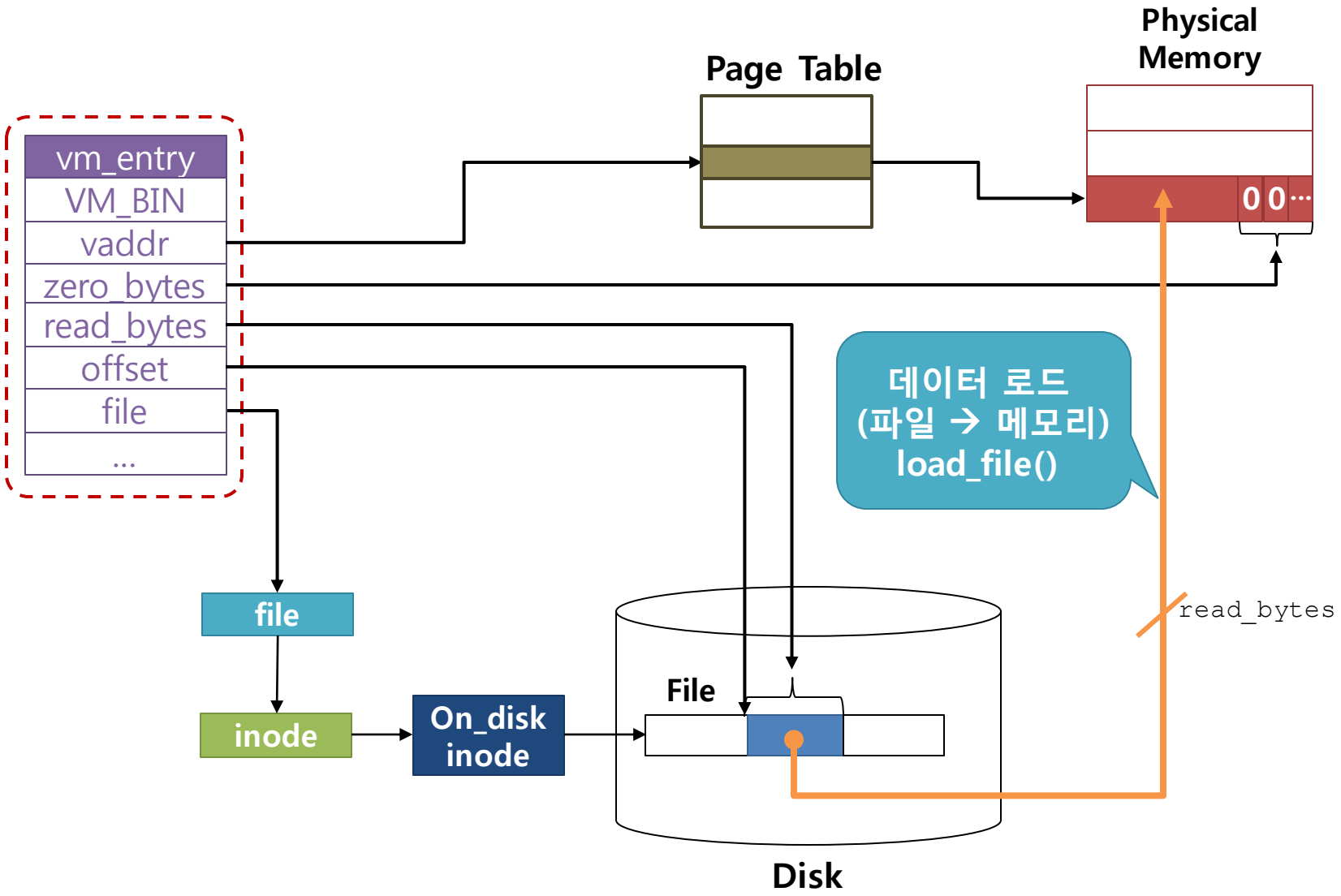
pintos/src/userprog/process.c

```
bool handle_mm_fault (struct vm_entry *vme)
{
        /* palloc_get_page()를 이용해서 물리메모리 할당 */
        /* switch문으로 vm_entry의 타입별 처리 (VM_BIN외의 나머지 타입은 mmf
        와 swapping에서 다룸*/
        /* VM_BIN일 경우 load_file()함수를 이용해서 물리메모리에 로드 */
        /* install_page를 이용해서 물리페이지와 가상페이지 맵핑 */
        /* 로드 성공 여부 반환 */
}
```

할일3: 물리메모리에 파일 쓰기

- 물리메모리 할당 완료 후 실제 디스크의 파일을 물리페이지로 load
- Pintos/src/vm/page.c
  - `bool load_file (void* kaddr, struct vm_entry *vme)`
    - Disk에 존재하는 page를 물리 메모리로 load하는 함수
    - vme의 <파일,offset>으로 한 페이지를 kaddr로 읽어 들이는 함수를 구현
    - `file_read_at()` 함수 또는 `file_read()` + `file_seek()`함수 이용
    - 4KB를 전부 write하지 못했다면 나머지를 0으로 채움

할일3: 물리메모리에 파일 페이지 탑재



할일3: 물리메모리에 파일 쓰기(Cont.)

pintos/src/vm/page.c

```
bool load_file (void *kaddr, struct vm_entry *vme)
{
        /*Using file_read() + file_seek() */
        /* 오프셋을 vm_entry에 해당하는 오프셋으로 설정(file_seek()) */
        /* file_read로 물리페이지에 read_bytes만큼 데이터를 씀*/
        /* zero_bytes만큼 남는 부분을 '0'으로 패딩 */
        /* file_read 여부 반환 */
}
```

pintos/src/vm/page.c

```
bool load_file (void *kaddr, struct vm_entry *vme)
{
        /*Using file_read_at()*/
        /* file_read_at으로 물리페이지에 read_bytes만큼 데이터를 씀*/
        /* file_read_at 여부 반환 */
        /* zero_bytes만큼 남는 부분을'0'으로 패딩 */
        /*정상적으로 file을 메모리에 loading 하면 true 리턴*/

}
```

요구 페이징 구현 관련 함수 추가 및 수정

- pintos/src/userprog/exception.c

```
static void page_fault (struct intr_frame *f)
    /*현재는 page fault가 발생하면 kill(-1)을 하여 종료함*/
    /*kill(-1)처리 관련 코드 삭제*/
    /*vm_entry 검색 후 페이지를 할당하도록 코드 수정, handle_mm_fault() 이용 */
```

- pintos/src/vm/page.c

```
bool load_file (void* kaddr, struct vm_entry *vme)
    /*Disk에 존재하는 page를 물리 메모리로 load하는 함수 */
    /* vme의 <파일,offset>로부터 한 페이지를 kaddr로 읽어 들이는 함수를 구현 */
    /* file_read_at() 함수 또는 file_read() + file_seek()함수 이용 */
```

- pintos/src/userprog/process.c

```
bool handle_mm_fault(struct vm_entry *vme)
    /*handle_mm_fault는 페이지 폴트 발생시 핸들링을 위해 호출 되는 함수 */
    /* 페이지 폴트 발생시 물리페이지를 할당하는 함수를 구현 */
```

수정 사항

- 수정 파일
  - pintos/src/userprog/process.c
  - pintos/src/userprog/exception.c
  - pintos/src/userprog/syscall.*
  - pintos/src/threads/thread.h
  - pintos/Makefile.build
  - pintos/tests/Make.tests
- 추가 파일
  - pintos/src/vm/page.h
  - pintos/src/vm/page.c

수정 파일

- Makefile.build 수정
  - 추가한 page파일을 사용하기 위해 코드 추가

pintos/Makefile.build

```
...
userprog_SRC += userprog/tss.c          # TSS management.

# No virtual memory code yet.
#vm_SRC = vm/file.c                    # Some file.
vm_SRC = vm/page.c

# Filesystem code.
filesys_SRC  = filesys/filesys.c       # Filesystem core.
filesys_SRC += filesys/free-map.c      # Free sector
bitmap.
filesys_SRC += filesys/file.c          # Files.
filesys_SRC += filesys/directory.c     # Directories.
filesys_SRC += filesys/inode.c         # File headers.
filesys_SRC += filesys/fsutil.c        # Utilities.
...
```

수정 파일 (Cont.)

- Makefile.tests 수정
- 변경하지 않으면 `make check`시 fail발생
  - 환경에 따라 테스트 수행시간을 지나칠 수 있음.

pintos/tests/Make.tests

```
...
ifdef PROGS
include ../../Makefile.userprog
endif

TIMEOUT = 60  /*Pintos의 테스트 수행시간을 60초에서 120초로 변경*/

clean::
        rm -f $(OUTPUTS) $(ERRORS) $(RESULTS)

grade:: results
       $(SRCDIR)/tests/make-grade $(SRCDIR) $< $(GRADING_FILE)
| tee $@
...
```

추가 함수

```
void vm_init(struct hash* vm)
   /* 해시테이블 초기화 */


void vm_destroy(struct hash *vm)
   /* 해시테이블 제거 */
```

추가 함수(Cont.)

```
struct vm_entry* find_vme(void *vaddr)
```

/* 현재 프로세스의 주소공간에서 vaddr에 해당하는 vm_entry를 검색 */

```
bool insert_vme(struct hash *vm, struct vm_entry *vme)
```

/* 해시테이블에 vm_entry 삽입 */

```
bool delete_vme(struct hash *vm, struct vm_entry *vme)
```

/* 해시 테이블에서 vm_entry삭제 */

추가 함수(Cont.)

```
static unsigned vm_hash_func(const struct hash_elem *e, void
*aux UNUSED)
        /* 해시테이블에 vm_entry 삽입 시 어느 위치에 넣을 지 계산 */

static bool vm_less_func(const struct hash_elem *a, const
struct hash_elem *b, void *aux UNUSED)
        /* 입력된 두 hash_elem의 주소 값 비교 */

static void vm_destroy_func(struct hash_elem *e, void *aux
UNUSED)
        /* vm_entry의 메모리 제거 */
```

추가 함수(Cont.)

```
bool handle_mm_fault(struct vm_entry *vme)
      /* 페이지 폴트 발생시 물리페이지를 할당 */

bool load_file(void* kaddr, struct vm_entry *vme)
      /* vme의 <파일,offset>로부터 한 페이지를 kaddr로 읽어들이는 함수 */

void check_valid_buffer(void* buffer, unsigned size,
                        void* esp, bool to_write)
      /* buffer내 vm_entry가 유효한지 확인하는 함수 */
void check_valid_string(const void* str, void* esp)
      /* 문자열의 주소값이 유효한지 확인하는 함수 */
```

수정 함수

```
static bool load_segment(struct file *file, off_t ofs,
                         uint8_t *upage, uint32_t read_bytes,
                         uint32_t zero_bytes, bool writable)
 /* ELF포맷 파일의 세그먼트를 프로세스 가상주소공간에 적절히 탑재하는 함수
    실행 파일을 로드 시 호출 */

static bool setup_stack(void **esp)
 /* 가상 메모리의 스택부분을 초기화 하는 함수 */
```

수정 함수(Cont.)

```
static void page_fault(struct intr_frame *f)
  /* 페이지 폴트 핸들러 */

static void syscall_handler(struct intr_frame *f UNUSED)
      /* 시스템 콜 넘버에 해당하는 시스템 콜을 호출 하는 함수 */

void check_address(void *addr, void* esp)
      /* 주소 값이 유저 영역에서 사용하는 주소 값인지 확인 하는 함수 */

void process_exit (void)
     /* 프로세스 종료 시 호출되어 프로세스의 자원을 해제 */
```

추가 및 수정 자료구조

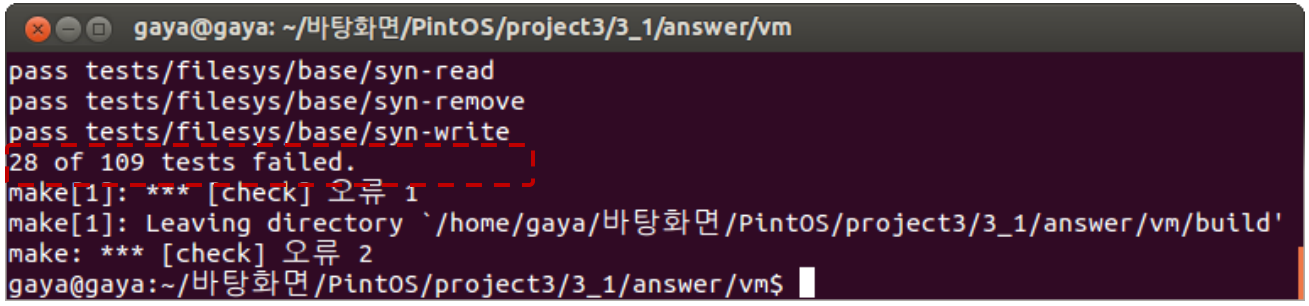
```
struct vm_entry
        /* 논리주소와 물리주소를 분리하여 "반드시" 필요한 페이지들만 탑재
           시키도록 하는 자료구조 */

struct thread
        /* 스레드의 정보를 가지고 있는 자료구조 */
```

가상메모리 과제 검증

- 가상메모리 과제를 완료 후 코드 동작 확인
  - 경로 : pintos/src/vm

```
$ make check
```

- 실행 결과 109개의 테스트 중 28개에서 fail 발생
  - pt-grow-stack
  - pt-grow-pusha
  - pt-big-stk-obj
  - pt-grow-stk-sc
  - page-linear
  - page-parallel
  - page-merge-seq
  - page-merge-par
  - page-merge-stk
  - page-merge-mm
  - mmap-read
  - mmap-close
  - mmap-unmap
  - mmap-overlap
  - mmap-twice
  - mmap-write
  - mmap-exit
  - mmap-shuffle
  - mmap-bad-fd
  - mmap-clean
  - mmap-inherit
  - mmap-misalign
  - mmap-null
  - mmap-over-code
  - mmap-over-data
  - mmap-over-stk
  - mmap-remove
  - mmap-zero

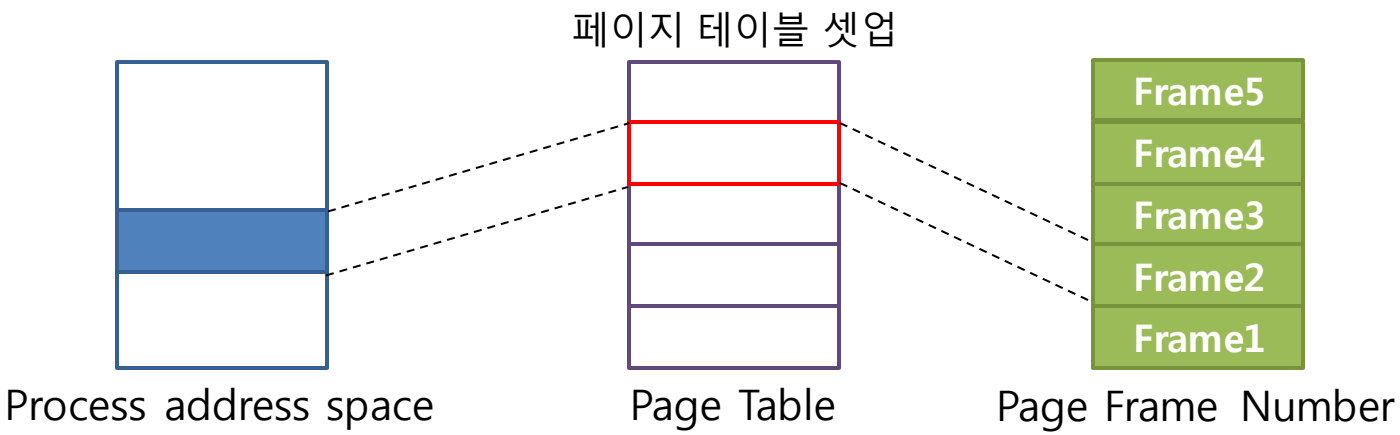


Appendix

페이지 주소 매핑함수

```
#include "usrprog/process.c"

  static bool install_page(void *upage, void *kpage,
                           bool writable)
```

- 페이지 테이블에 물리 주소와 가상주소를 맵핑 시켜주는 함수
- 물리 페이지 kpage와 가상 페이지 upage를 맵핑
- writable: 쓰기 가능(1), 읽기전용(0)



Pintos의 페이지 테이블 구조

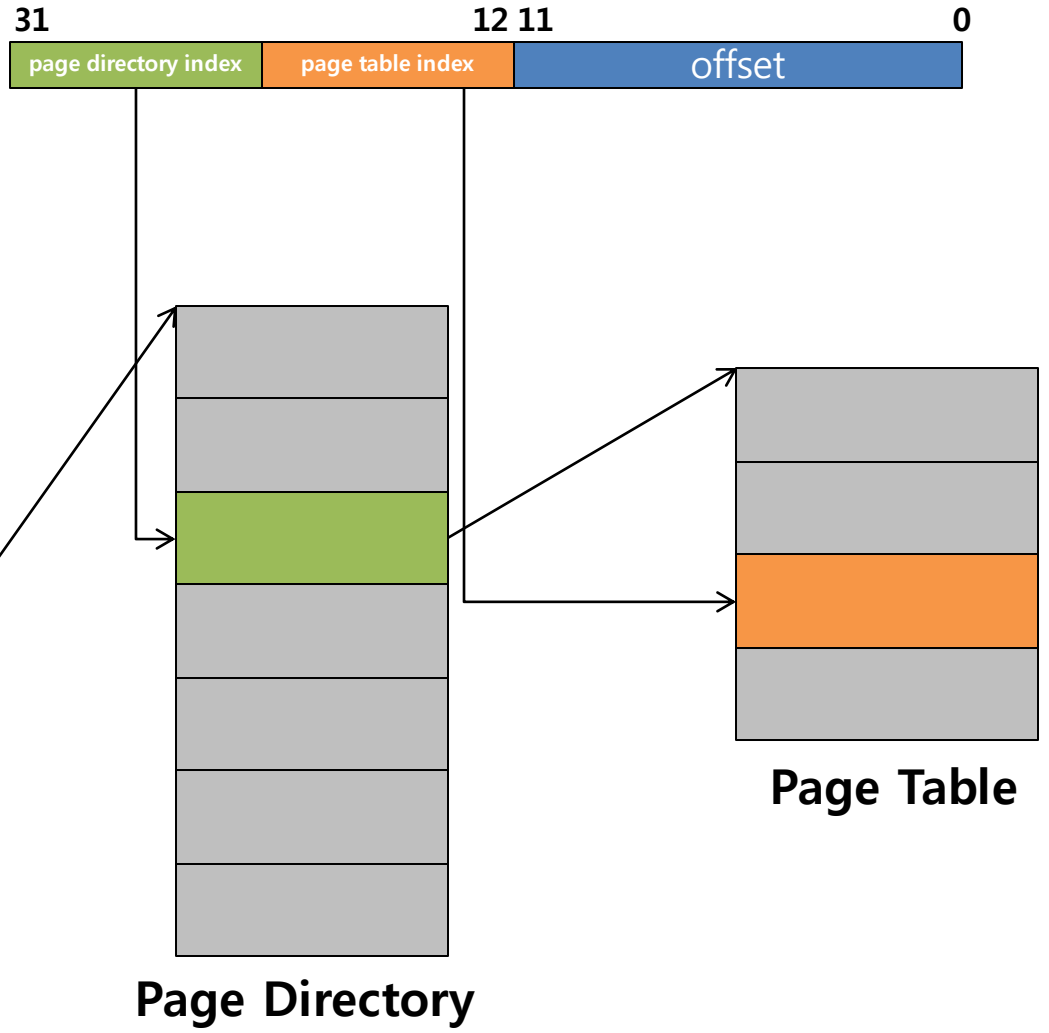
- 페이지 테이블과 페이지 디렉토리를 이용해서 물리 주소와 가상주소의 맵핑 관리
  - 페이지 디렉토리
    - 페이지 테이블의 주소를 갖고 있는 테이블
    - 모든 가상페이지에 대한 엔트리 사용은 비효율적이므로 한 단계 상위 개념인 페이지 디렉토리를 사용
  - 페이지 테이블
    - 가상 주소에 맵핑된 물리 주소를 갖고 있는 엔트리들의 집합

- 가상 메모리 주소에 포함된 index로 entry 접근

Pintos의 페이지 테이블 구조 (Cont.)

pintos/src/threads/thread.h

```
struct thread
{
        tid_t tid;
        ...
#ifdef USERPROG
        uint32_t *pagedir;
#endif
        ...
}
```



물리 페이지 할당 및 해제 인터페이스

```
#include <threads/palloc.h>
```

```
void *palloc_get_page(enum palloc_flags flags)
```

- 4KB의 페이지를 할당
- 페이지의 물리 주소를 리턴
- flags
  - PAL_USER: 유저 메모리풀에서 페이지 할당
  - PAL_KERNEL: 커널 메모리 풀에서 페이지 할당
  - PAL_ZERO: 페이지를 '0'으로 초기화

```
void palloc_free_page(void *page)
```

- 페이지의 물리주소를 인자로 사용
- 페이지를 다시 여유 메모리 풀에 넣음