CDA0017: Operating Systems

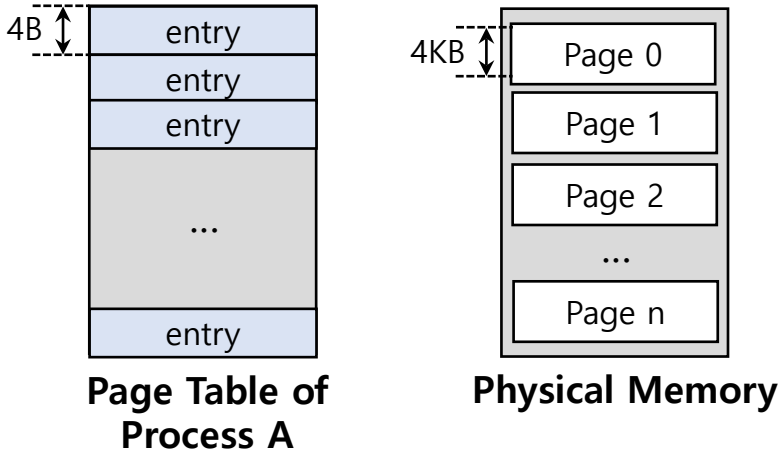
Donghyun Kang (donghyun@changwon.ac.kr)

NOSLab (https://noslab.github.io)

Changwon National University

Paging의 문제2

- 모든 프로세스는 하나의 페이지 테이블을 가짐
  - 32bit 주소 공간, 4KB 페이지, 4Byte 페이지 테이블 항목(PTE)

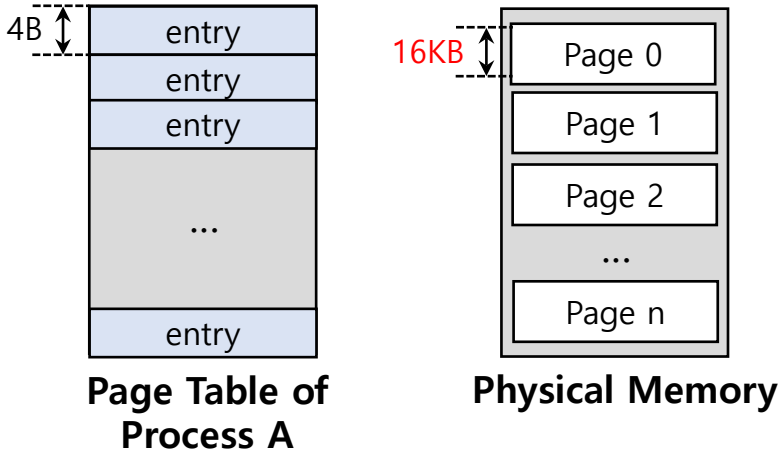


페이지 테이블 크기 = $\frac{2^{32}}{2^{12}} * 4Byte = 4MByte$

페이지 테이블을 위해 너무 많은 메모리가 낭비됨

해결 방법 1

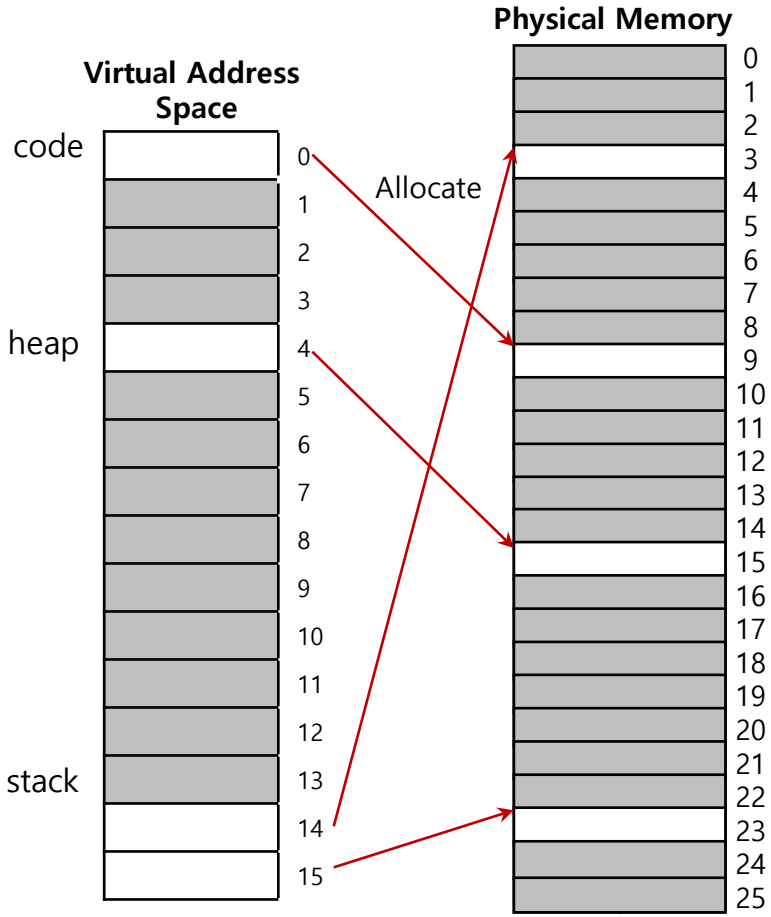
- 물리 메모리의 페이지 크기 확장
  - 32bit 주소 공간, 16KB 페이지, 4Byte 페이지 테이블 항목(PTE)



$\frac{2^{32}}{2^{16}} * 4 = 1MB$ **per page table**

내부 단편화 발생

해결 방법 2



A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

해결 방법 2

- 페이지 테이블 대부분 사용되지 않음

PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...	...	...	...	...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

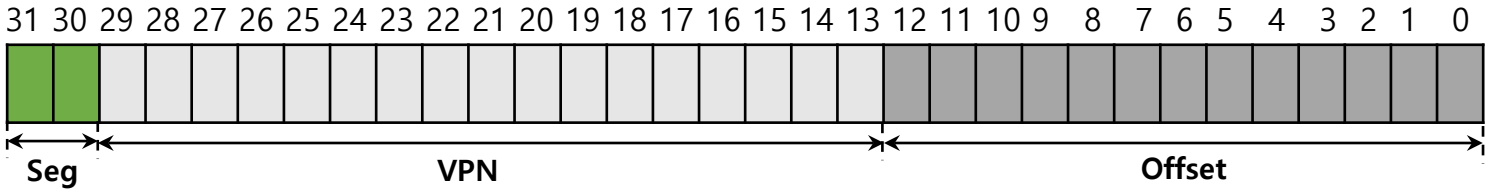
A Page Table For 16KB Address Space

해결 방법 2

- 하이브리드 접근 방식
  - 세그먼테이션
    - 가상 머신을 세그먼트 단위로 분할Divide virtual address space into segments
    - 세그먼트는 가변으로 사용 가능
  - 페이징
    - 각 세그먼트를 고정된 페이지로 분할함
    - 각 세그먼트는 하나의 페이지 테이블을 가짐
  - 각 세그먼트의 베이스는 세그먼트의 시작 물리 주소가 아닌 **페이지 테이블의 시작 물리 주소**를 가짐
  - 바운드(limit) 레지스터는 **페이지 테이블의 마지막 물리 주소**를 가짐

해결 방법 2

- 예
  - 각 프로세스는 3개의 페이지 테이블로 관리됨



32-bit Virtual address space with 4KB pages

Seg value	Content
00	unused segment
01	code
10	heap
11	stack

해결 방법 2

- TLB miss 의 경우, 하드웨어는 세그먼트 구분 bit을 기반으로 해당 물리 주소를 찾음

```
01:     SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT
02:     VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
03:     AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

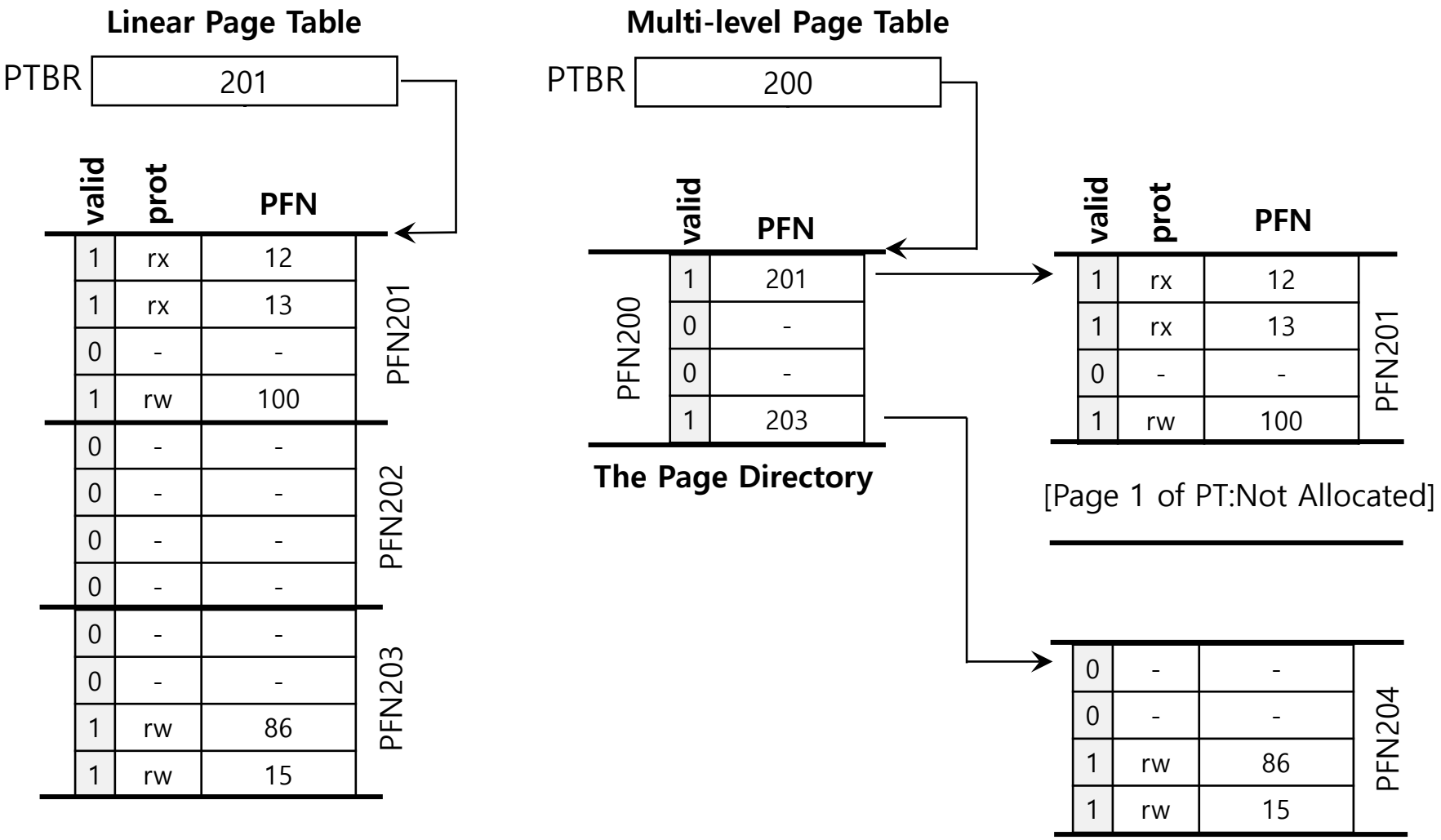
해결 방법 2

- 문제점
  - 외부 단편화 문제가 다시 발생함
  - 부분적으로 사용 / 해제되는 힙의 경우 메모리 테이블의 낭비가 여전히 발생함

멀티 레벨 페이지 테이블

- 선형의 페이지 테이블을 트리 구조의 테이브로 변환
  - 페이지 테이블을 페이지 크기로 분할함
  - 페이지 테이블의 페이지가 유효하지 않은 항목만 존재하면 해당 페이지를 할당하지 않음
  - **페이지 디렉터리** (page directory) 자료 구조를 사용하여 페이지 테이블의 각 페이지에 대한 할당 여부 및 위치 파악

멀티 레벨 페이지 테이블


Linear Page Table
PTBR
201
valid
prot
PFN
1 rx 12
1 rx 13
0 - -
1 rw 100
PFN201
0 - -
0 - -
0 - -
0 - -
PFN202
0 - -
0 - -
1 rw 86
1 rw 15
PFN203
Multi-level Page Table
PTBR
200
valid
PFN
PFN200
1 201
0 -
0 -
1 203
The Page Directory
valid
prot
PFN
1 rx 12
1 rx 13
0 - -
1 rw 100
PFN201
[Page 1 of PT:Not Allocated]
0 - -
0 - -
1 rw 86
1 rw 15
PFN204

멀티 레벨 페이지 테이블

- 페이지 디렉터리
  - 하나의 항목은 하나의 페이지 테이블을 가리킴
  - 다수의 페이지 디렉터리 항목(PDE: page directory entry)을 가짐
  - PDE는 페이지 테이블을 포함하는 물리 메모리 주소 (PFN)이 valid 한지 여부를 가지고 있음

멀티 레벨 페이지 테이블

- 장점
  - 실제 사용되는 메모리 주소 공간에 비례하여 페이지 테이블 공간이 할당됨
  - 페이지 테이블을 페이지 크기로 분할함으로써 메모리 관리가 용이함
    - Free 페이지 풀에 있는 빈 페이지를 사용함
  - 페이지 디렉터리를 이용하여 연속되지 않은 메모리의 위치 파악 가능
    - 즉, 외부 단편화 이슈가 발생하지 않음

- 단점
  - TLB miss가 발생하면, 주소 변환을 위해 두번의 메모리 접근이 필요함
  - 설계 복잡도 증가

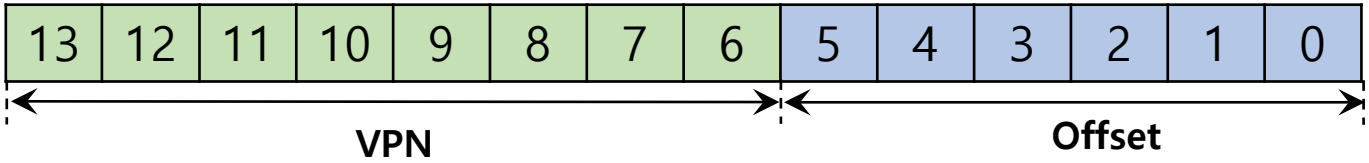
멀티 레벨 페이지 테이블

- 예제 (가정)

0000 0000	code
0000 0001	code
0000 0010	(free)
0000 0011	(free)
0000 0100	heap
0000 0101	heap
0000 0110	(free)
0000 0111	(free)
.....	... free ...
1111 1110	stack
1111 1111	stack

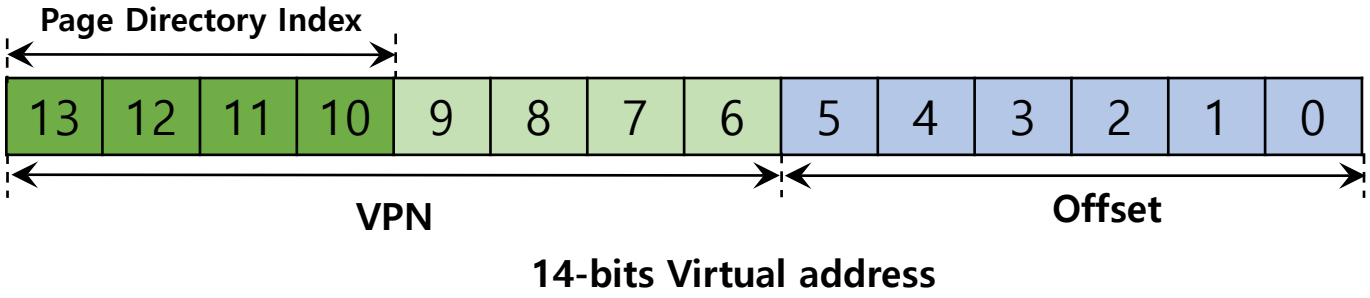
Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
Page table entry	$2^8(256)$

A 16-KB Address Space With 64-byte Pages



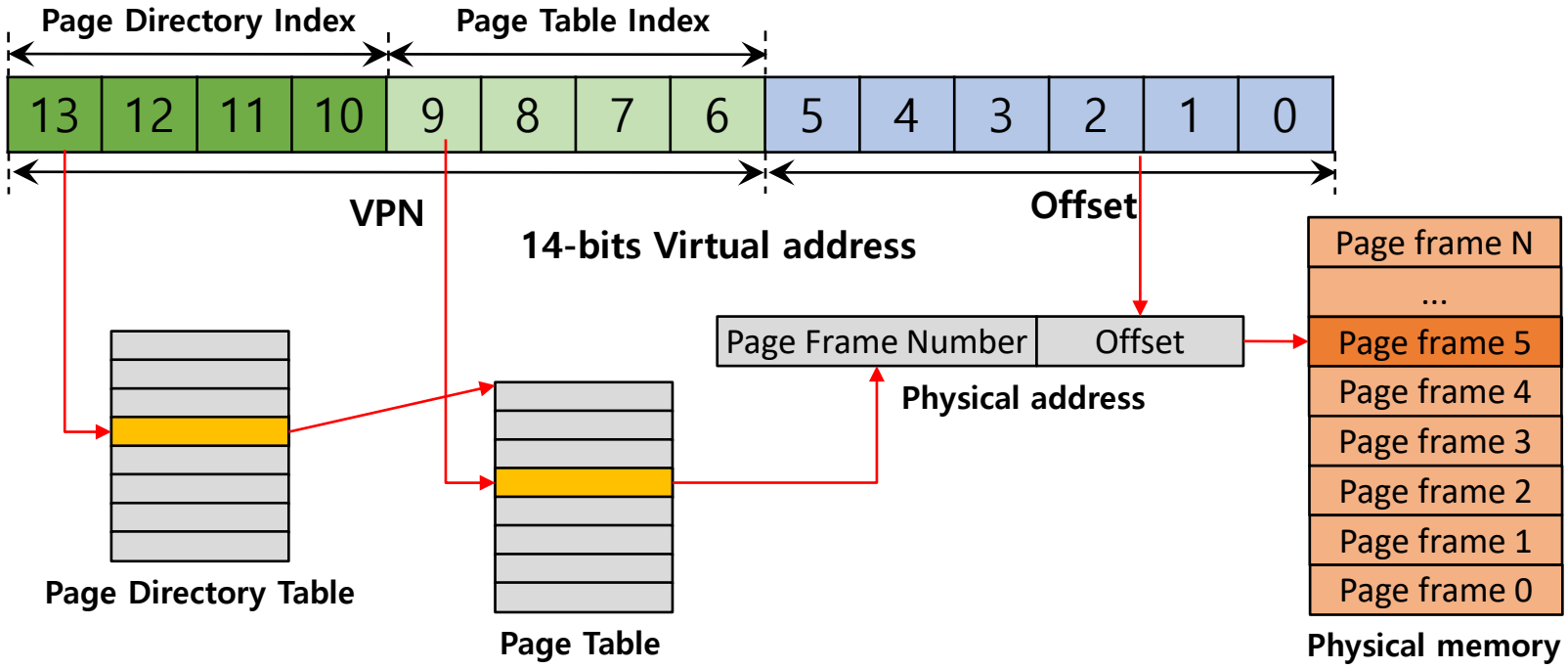
멀티 레벨 페이지 테이블

- 예제 (가정)
  - 하나의 PTE는 4byte의 크기를 가짐
  - 페이지 테이블의 크기: 1KB
    - 256 (2^8) x 4 (PTE 크기)
  - 하나의 페이지는 16개의 PTE을 가질 수 있음
    - 64Byte(페이지 크기) / 4 (PTE 크기)
  - 상위 4bit는 페이지 테이블을 가리킴



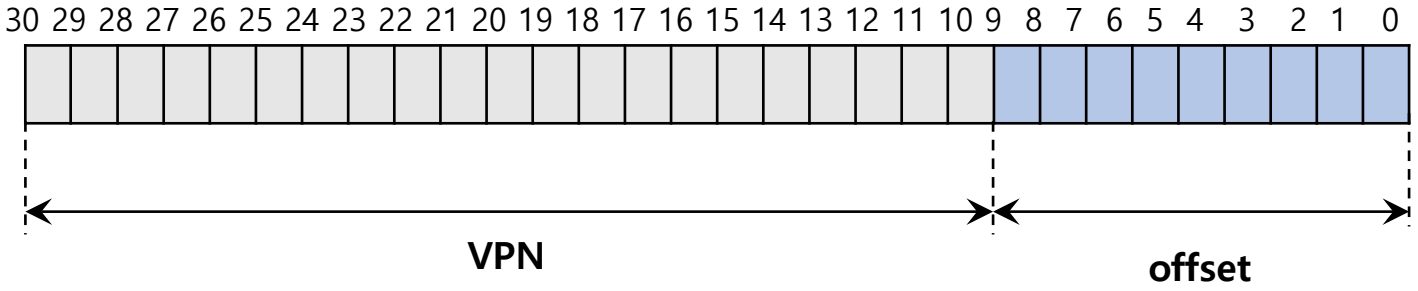
멀티 레벨 페이지 테이블

- 예제 (가정)
  - PDE가 invalid하면, 예외 발생
  - PDE가 valid하면, PTE 접근
    - PTE는 PFN를 가짐



멀티 레벨 페이지 테이블

- 2단계 페이지의 확장

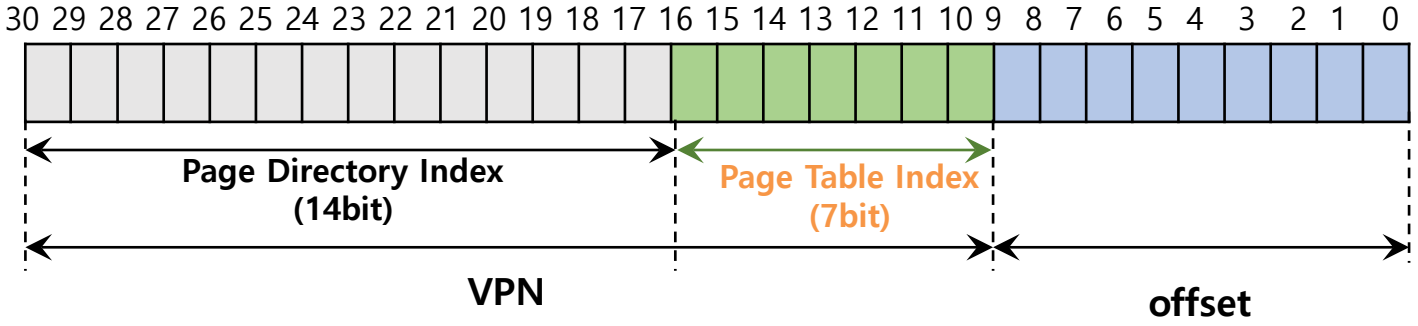


Flag	Detail
Virtual address	30 bit
Page size	512 byte
PTE size	4 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs = 512 (Page size) / 4 (PTE size)

→ $\log_2 128 = 7$

멀티 레벨 페이지 테이블

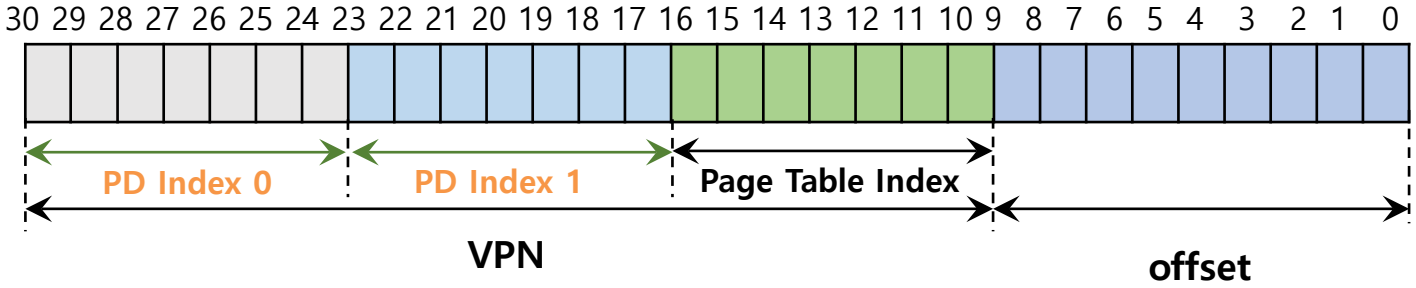
- 2단계 페이지의 확장



Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

멀티 레벨 페이지 테이블

- 2단계 페이지의 확장
  - 페이지 디렉터리 정보를 위해 128개의 페이지가 필요함
    - 16384 byte(2^14) * 4Byte(PTE size) / 512 (Page size)
  - 레벨 확장을 통해 문제 해결



역 페이지 테이블

- 각 물리 페이지 프레임을 위해 하나의 항목(entry)을 가지는 테이블
  - 프로세스와 가상 주소 페이지 → 물리 페이지 프레임 매핑
  - 최악의 경우, 전체 페이지 테이블을 검색해야 함
  - 해시(Hashing table)을 사용하여 검색함으로써 성능 제약을 해결함

- 장점
  - 메모리 공간을 낭비하지 않음

- 단점
  - TLB miss 상황에서의 성능 문제 발생

Q&A