3장 프로세스 기술(Description) 및 제어(Control)

프로세스의 개념

"Process is a program in execution"

프로세스의 문맥 (context)

→ CPU수행 상태를 나타내는 하드웨어 문맥

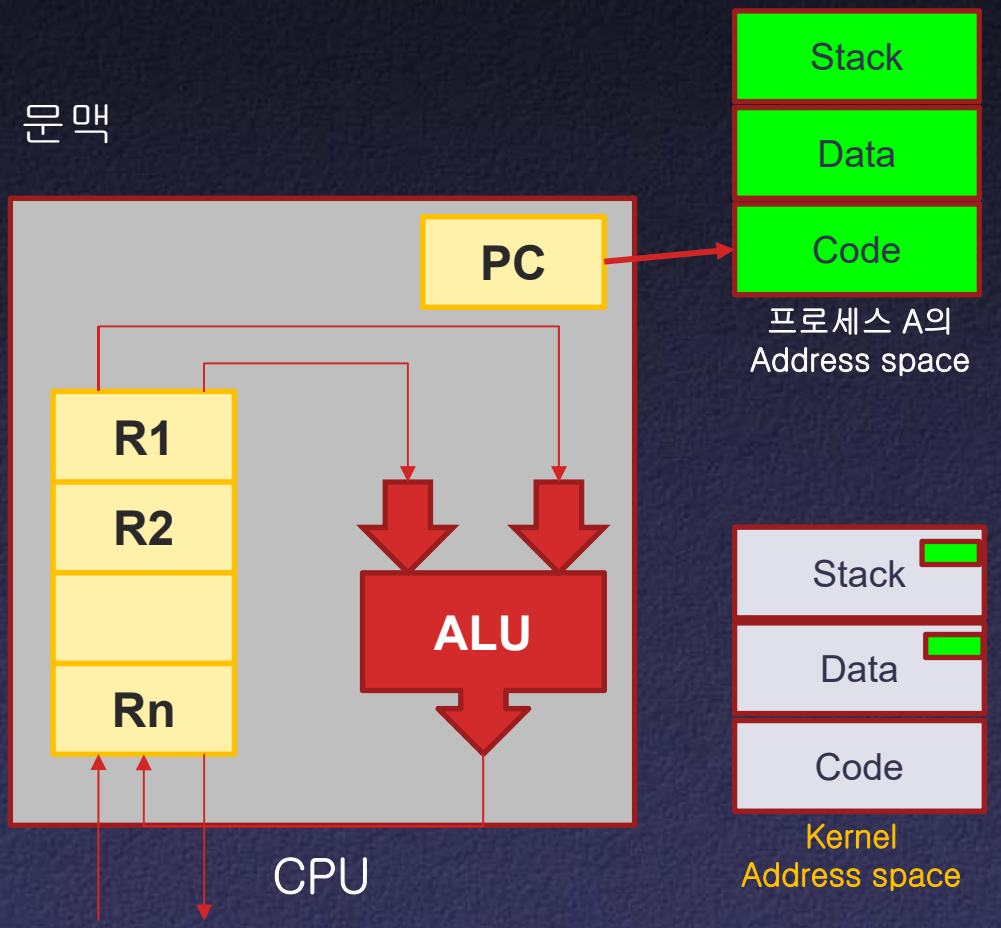
- Program Counter
- 각종 register

→ 프로세스의 주소공간

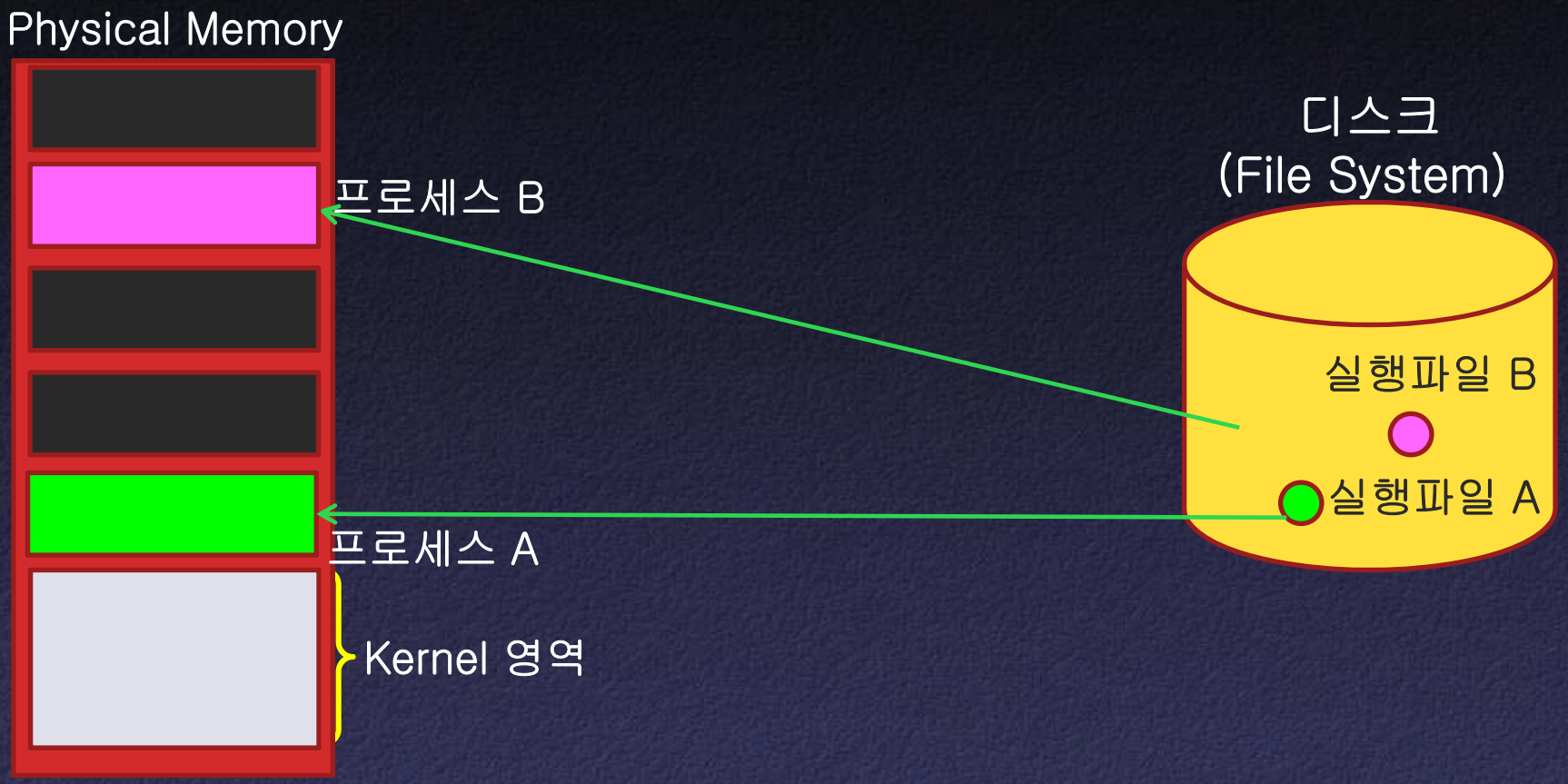
- Code, data, stack

→ 프로세스 관련 커널 자료 구조

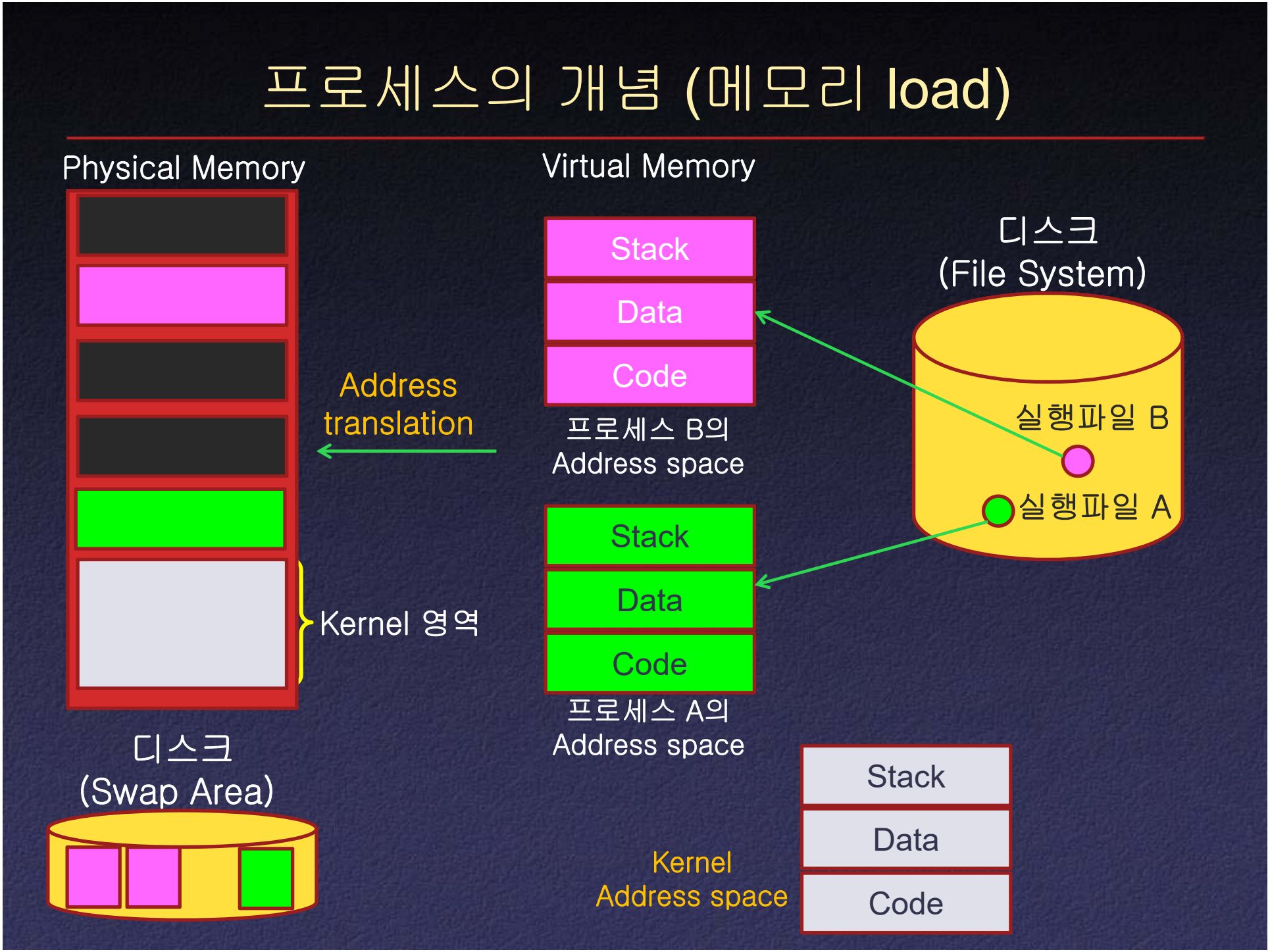
- PCB (Process Control Block)
- Kernel stack



프로세스의 개념 (메모리 load)



프로세스의 개념 (메모리 load)



프로세스의 상태 (Process State)

프로세스는 상태 (state)가 변경되며 수행된다.

→ Running

- CPU를 잡고 instruction을 수행중인 상태

→ Ready

- CPU를 기다리는 상태 (메모리 등 다른 조건을 모두 만족하고)

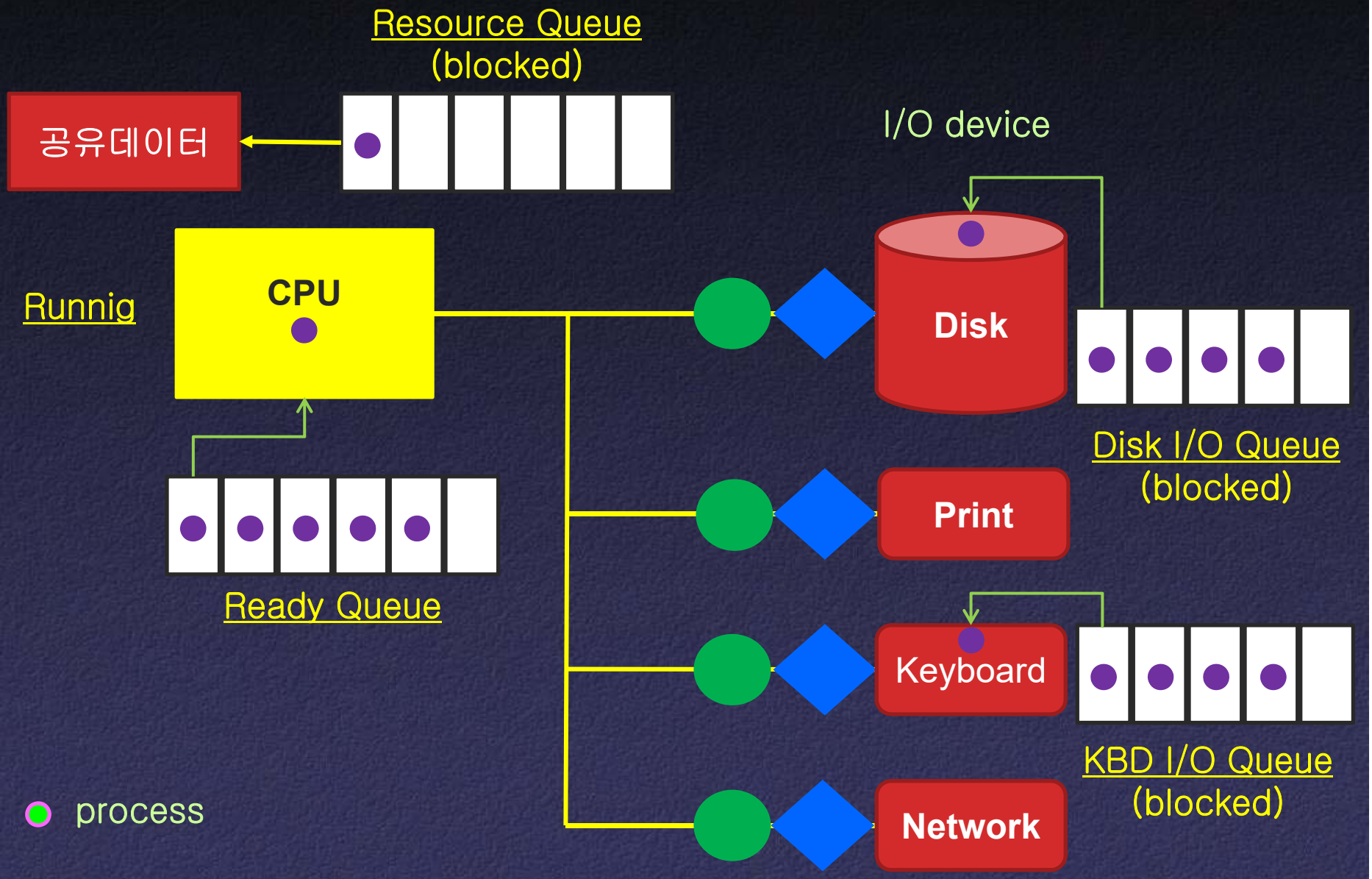
→ Blocked (wait, sleep)

- CPU를 주어도 당장 instruction을 수행할 수 없는 상태
- Process 자신이 요청한 event (예 : I/O)가 즉시 만족되지 않아 이를 기다리는 상태
- (예) 디스크에서 file을 읽어 와야 하는 경우

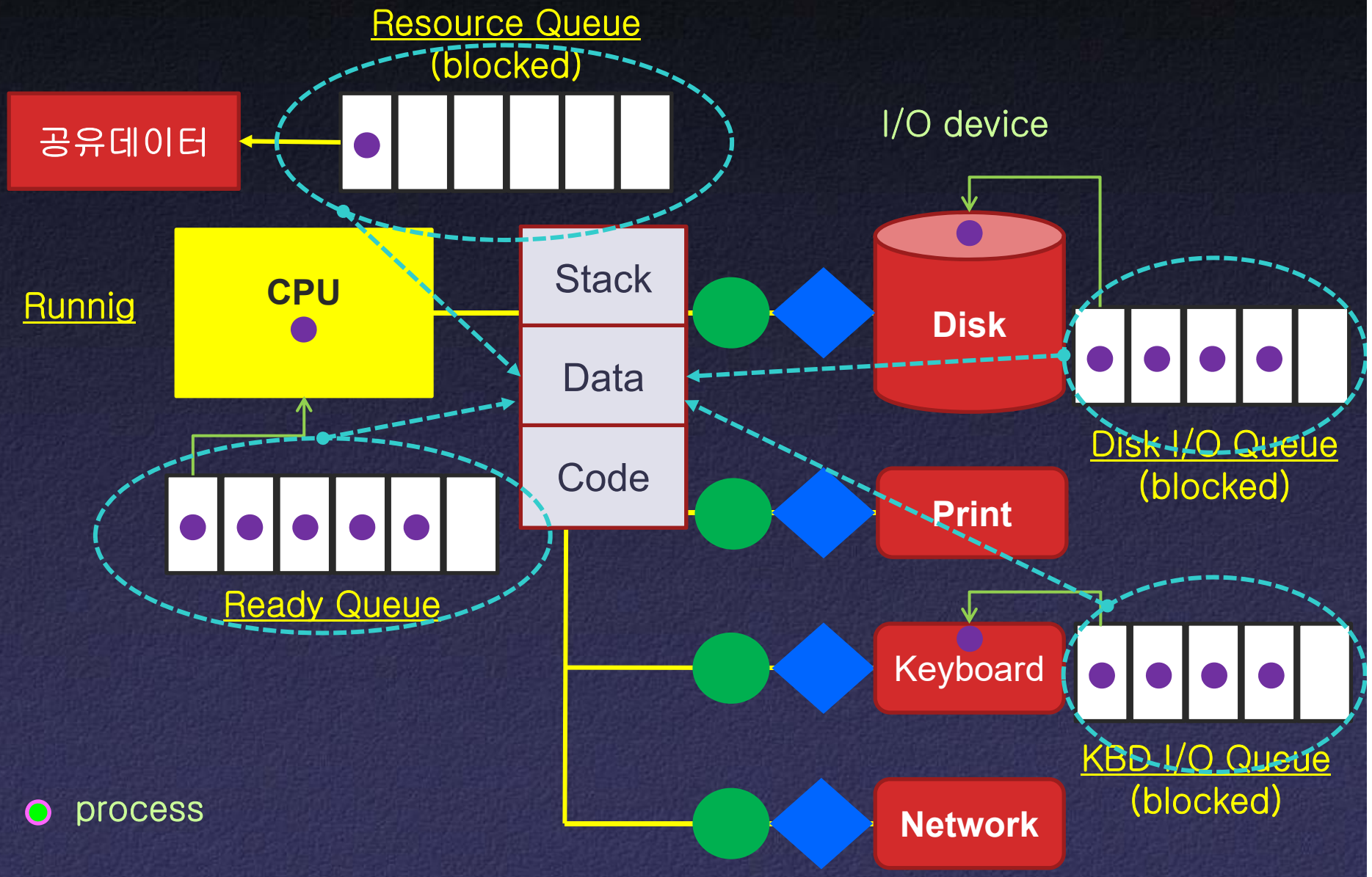
→ New : 프로세스가 생성중인 상태 (Create)

→ Terminated : 수행 (execution)이 끝난 상태

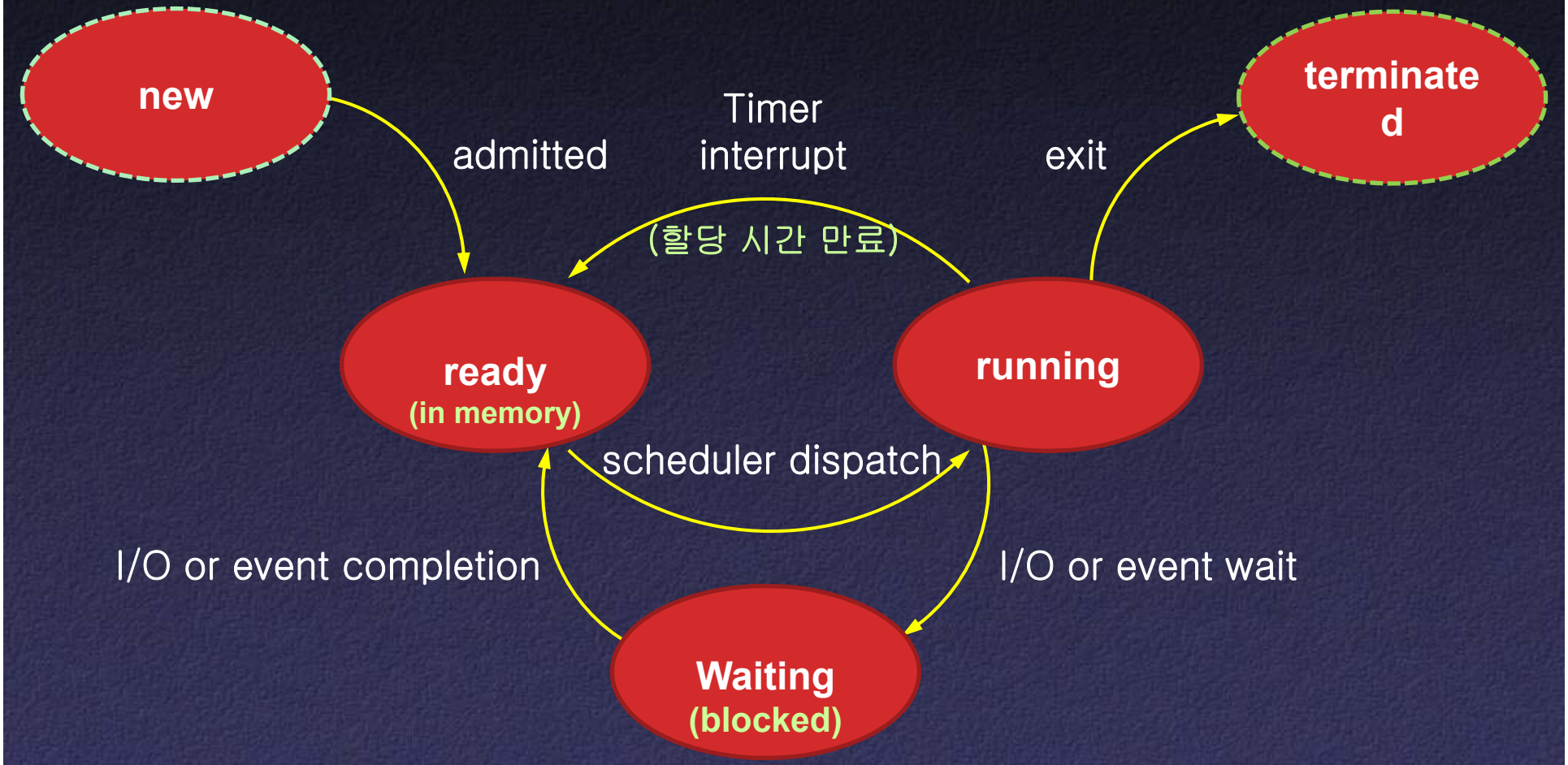
프로세스 상태


Resource Queue
(blocked)
공유데이터
I/O device
Runnig
CPU
Disk
Disk I/O Queue
(blocked)
Print
Ready Queue
Keyboard
KBD I/O Queue
(blocked)
Network
process

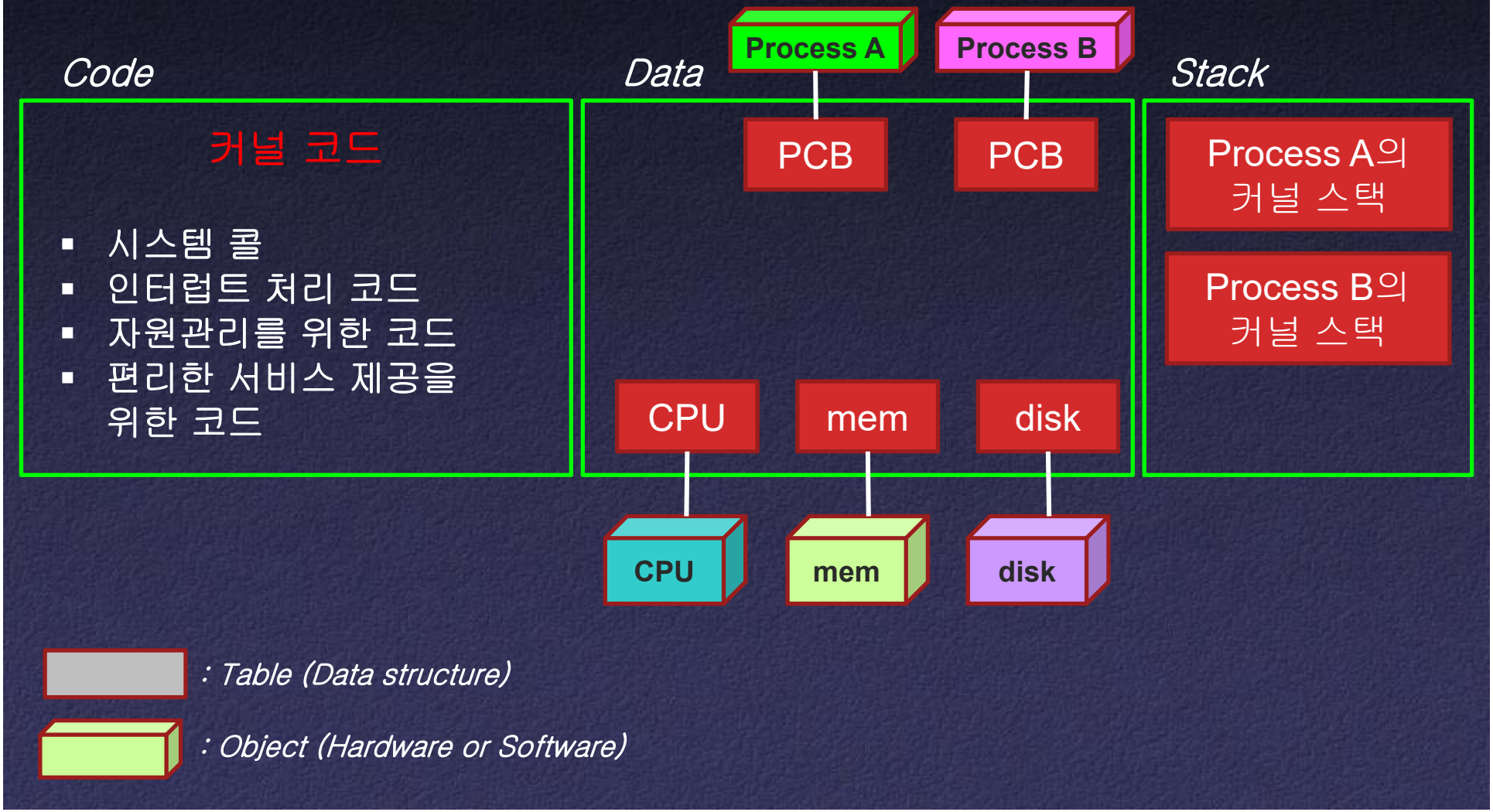
프로세스 상태


Resource Queue
(blocked)
공유데이터
I/O device
Runnig
CPU
Stack
Data
Code
Disk
Disk I/O Queue
(blocked)
Print
Ready Queue
Keyboard
KBD I/O Queue
(blocked)
Network
process

프로세스 상태도


new
admitted
Timer
interrupt
exit
terminate
d
(할당 시간 만료)
ready
(in memory)
running
scheduler dispatch
I/O or event completion
I/O or event wait
Waiting
(blocked)

커널 주소 공간의 내용

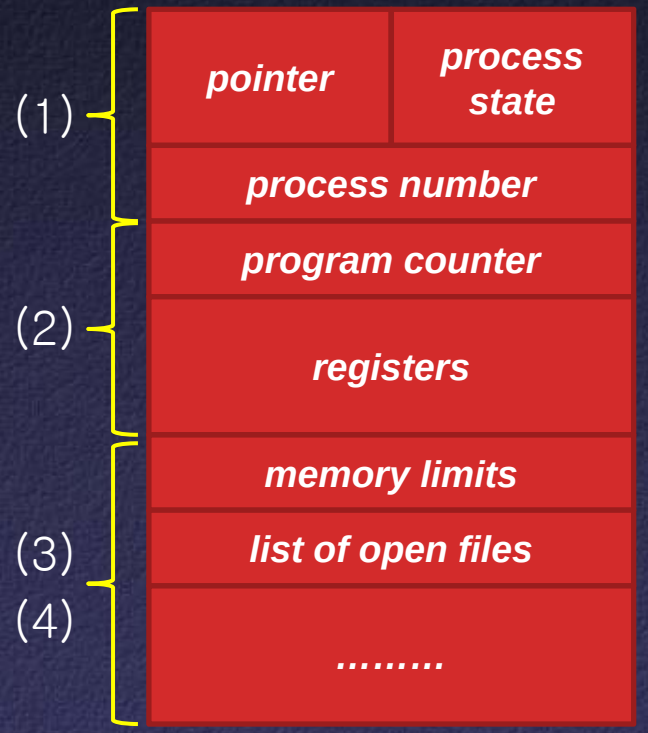

Code
Data
Stack
Process A
Process B
커널 코드
PCB
PCB
Process A의
커널 스택
Process B의
커널 스택
시스템 콜
인터럽트 처리 코드
자원관리를 위한 코드
편리한 서비스 제공을
위한 코드
CPU
mem
disk
CPU
mem
disk
: Table (Data structure)
: Object (Hardware or Software)

Process Control Block (PCB)

→ 운영체제가 각 프로세스를 관리하기 위해 프로세스 당 유지하는 정보

→ 다음의 구성요소를 가진다. (구조체로 유지)

- (1) OS가 관리상 사용하는 정보
  - ✓ Process State, Process ID
  - ✓ Scheduling information, Priority
- (2) CPU 수행 관련 하드웨어 값
  - ✓ Program counter, registers
- (3) 메모리 관련
  - ✓ Code, Data, Stack의 위치정보
- (4) 파일 관련
  - ✓ Open file descriptors



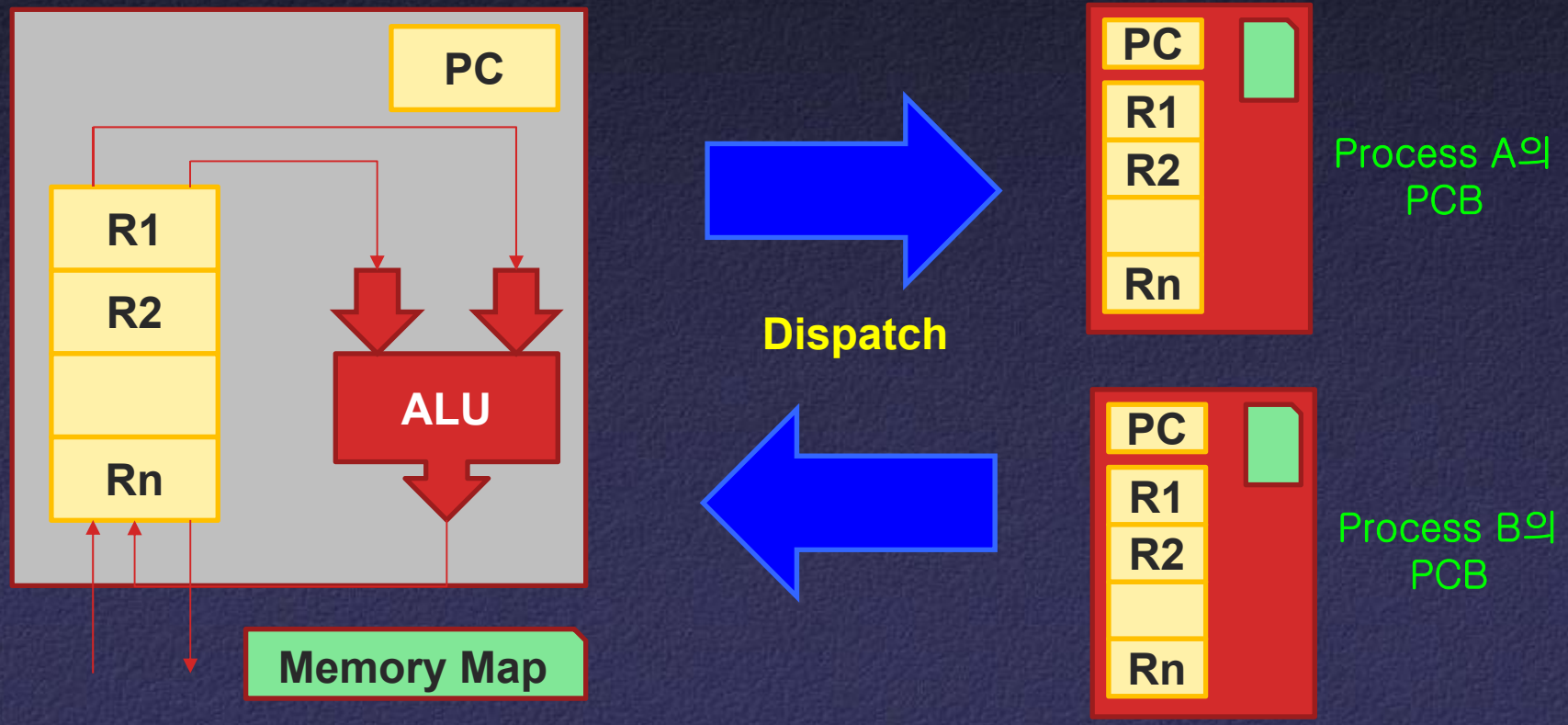
문맥 교환 (Context Switch)

CPU를 한 프로세스에서 다른 프로세스로 넘겨주는 과정

CPU가 다른 프로세스에게 넘어갈 때 운영체제는 다음을 수행

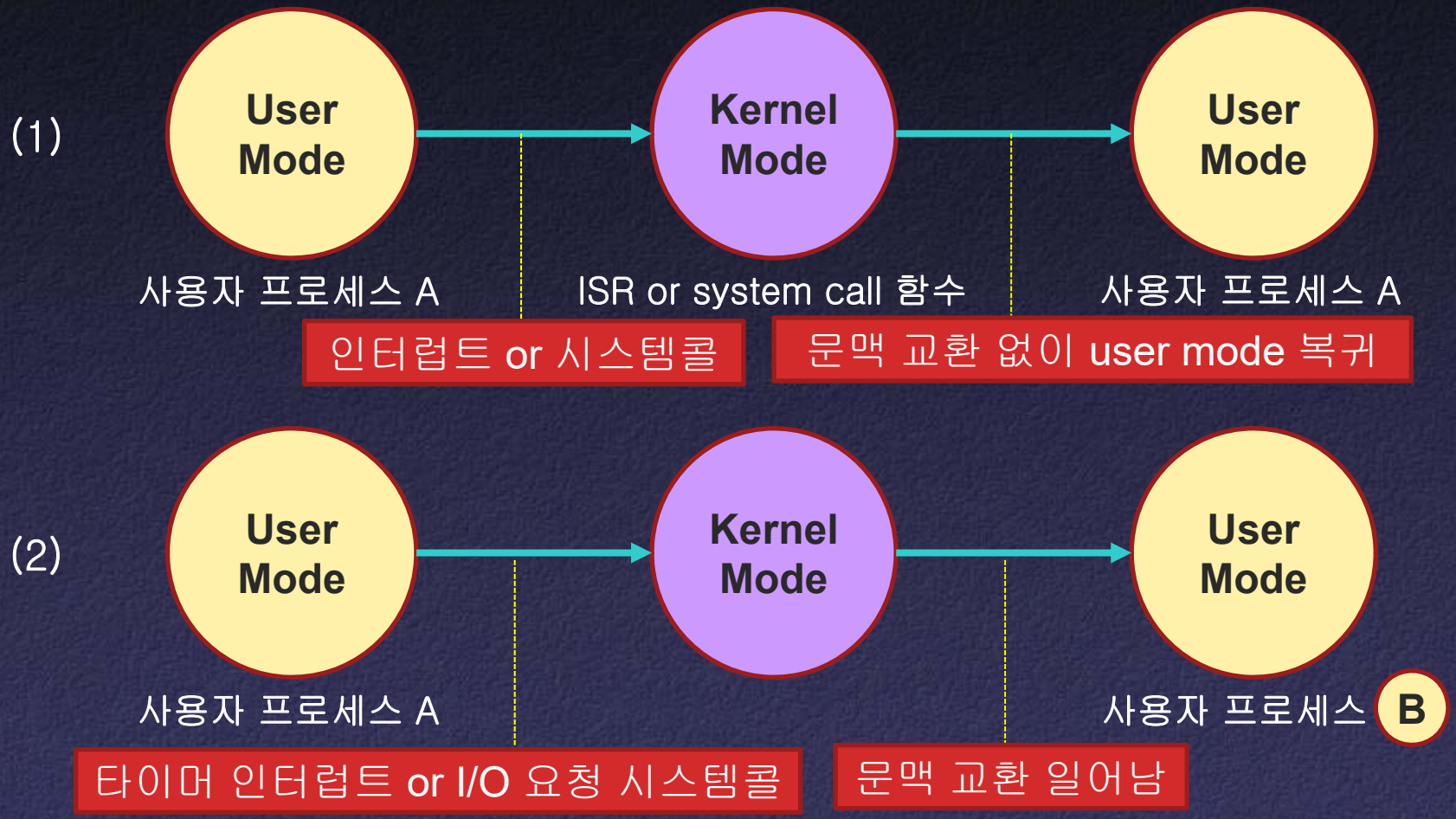
→ CPU를 내어주는 프로세스의 상태를 그 프로세스의 PCB에 저장

→ CPU를 새롭게 얻는 프로세스의 상태를 PCB에서 읽어옴



문맥 교환 (Context Switch)

System call이나 Interrupt 발생시 반드시 문맥 교환이 일어나는 것은 아님



*** (1)의 경우에도 CPU 수행 정보 등 context 의 일부를 PCB에 저장해야 하지만 문맥을 교환하는 (2)의 경우 그 부담이 훨씬 큼 (eg. cache memory flush)*

프로세스를 스케줄링하기 위한 큐

Job Queue

→ 현재 시스템 내에 있는 모든 프로세스의 집합

Ready Queue

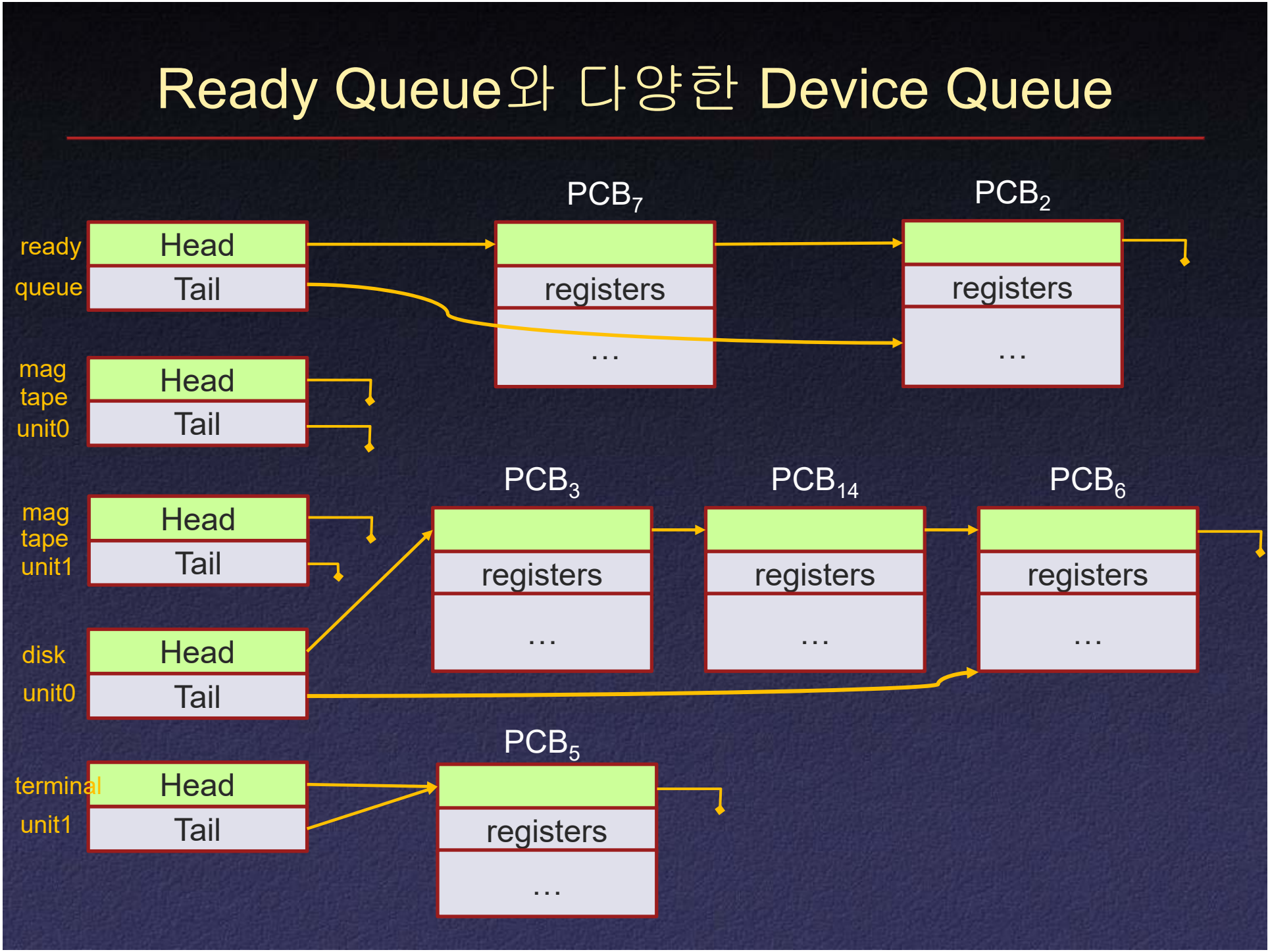
→ 현재 메모리 내에 있으면서 CPU를 잡아서 실행되기를 기다리는 프로세스의 집합

Device Queue

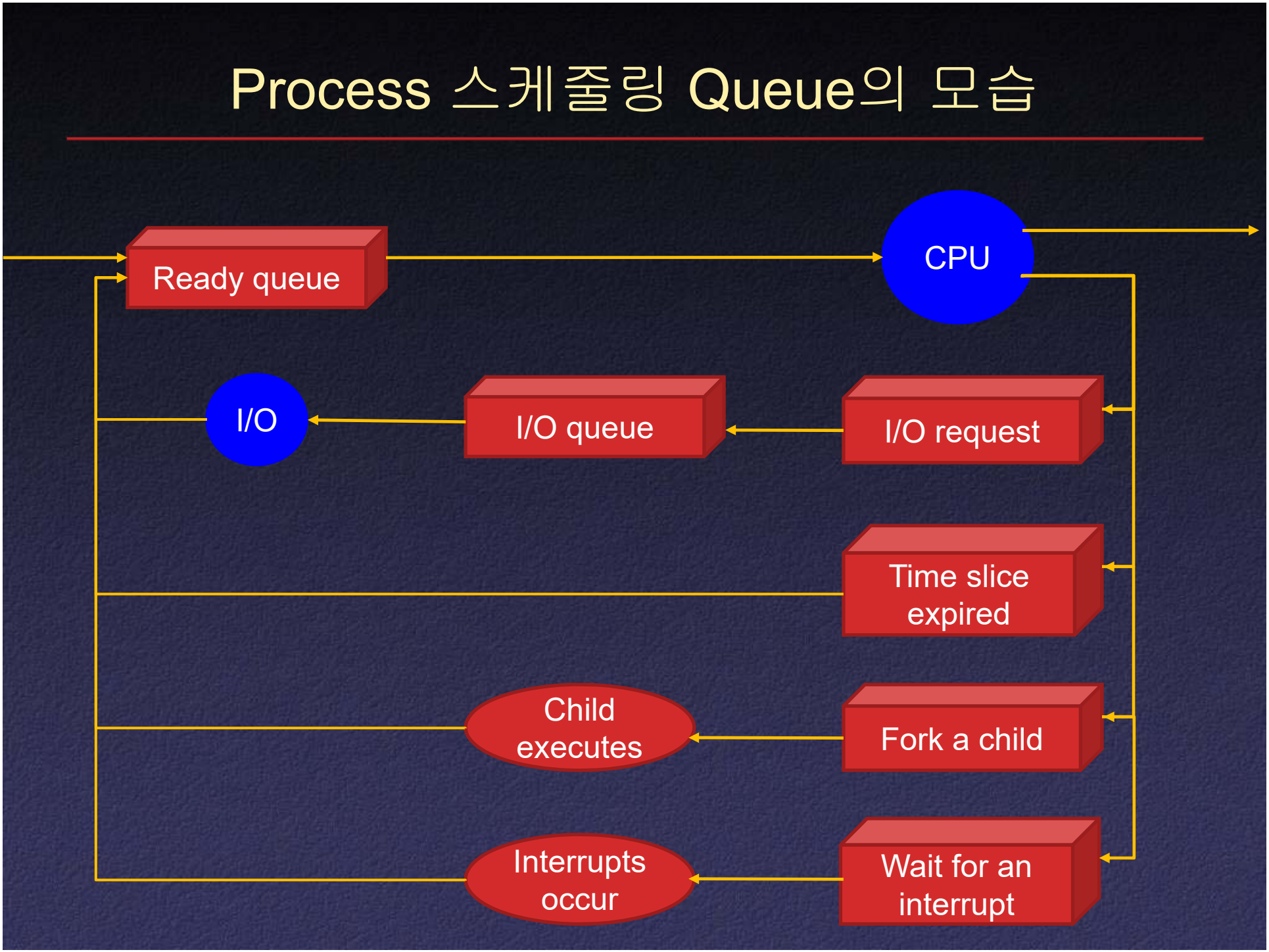
→ I/O 디바이스의 처리를 기다리는 프로세스의 집합

프로세스들은 각 큐들을 오가며 수행된다.

Ready Queue와 다양한 Device Queue



Process 스케줄링 Queue의 모습



스케줄러 (Scheduler)

Long-term scheduler (장기 스케줄러 or job scheduler)

→ 시작 프로세스 중 어떤 것들을 ready queue로 보낼지 결정

→ 프로세스에 memory (및 각종 지원)을 주는 문제

→ *degree of Multiprogramming*을 제어

→ *time sharing system*에는 보통 장기 스케줄러가 없음 (무조건 ready)

Short-term scheduler (단기 스케줄러 or CPU scheduler)

→ 어떤 프로세스를 다음 번에 running 시킬지 결정

→ 프로세스에 CPU를 주는 문제

→ 충분히 빨라야 함 (millisecond 단위)

Medium-term scheduler (중기 스케줄러 or Swapper)

→ 여유 공간 마련을 위해 프로세스를 통째로 메모리에서 디스크로 쫓아냄

→ 프로세스에게서 memory를 뺏는 문제

→ *degree of multiprogramming*을 제어

프로세스의 상태 (Process State)

→ Running

- CPU를 잡고 instruction을 수행중인 상태

→ Ready

- CPU를 기다리는 상태 (메모리 등 다른 조건을 모두 만족하고)

→ Blocked (wait, sleep)

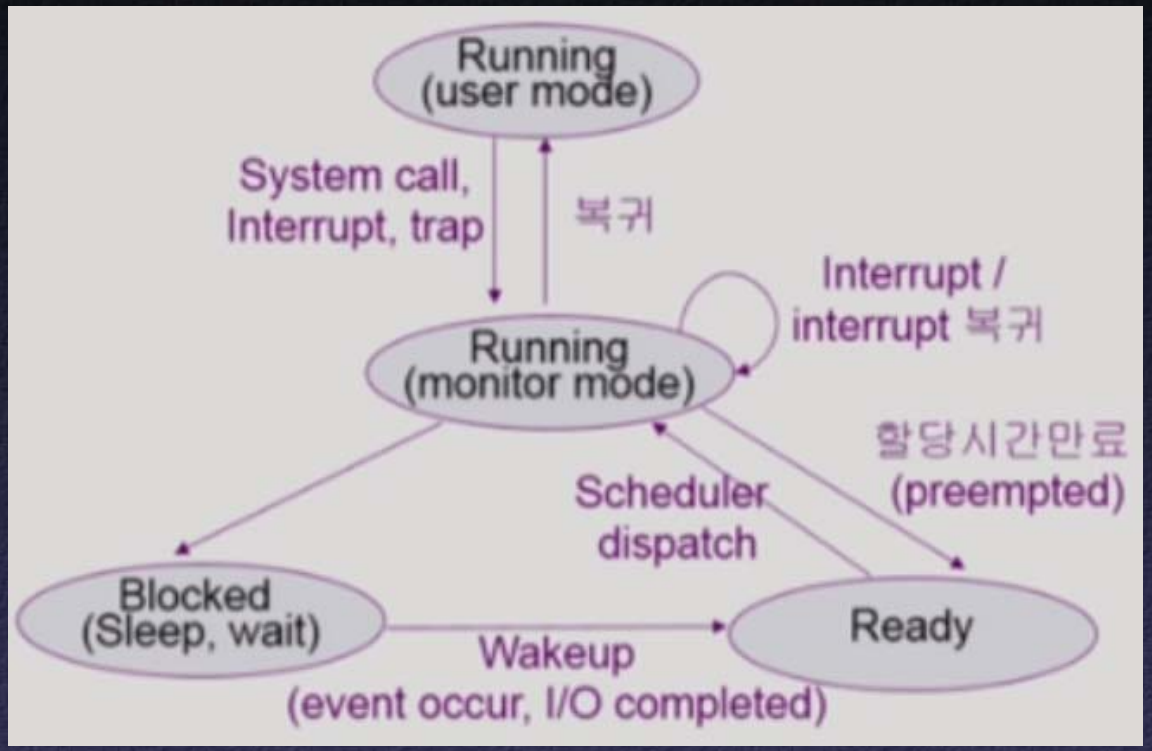
- I/O 등의 event를 (스스로) 기다리는 상태
- (예) 디스크에서 file을 읽어 와야 하는 경우

→ Suspended (stopped)

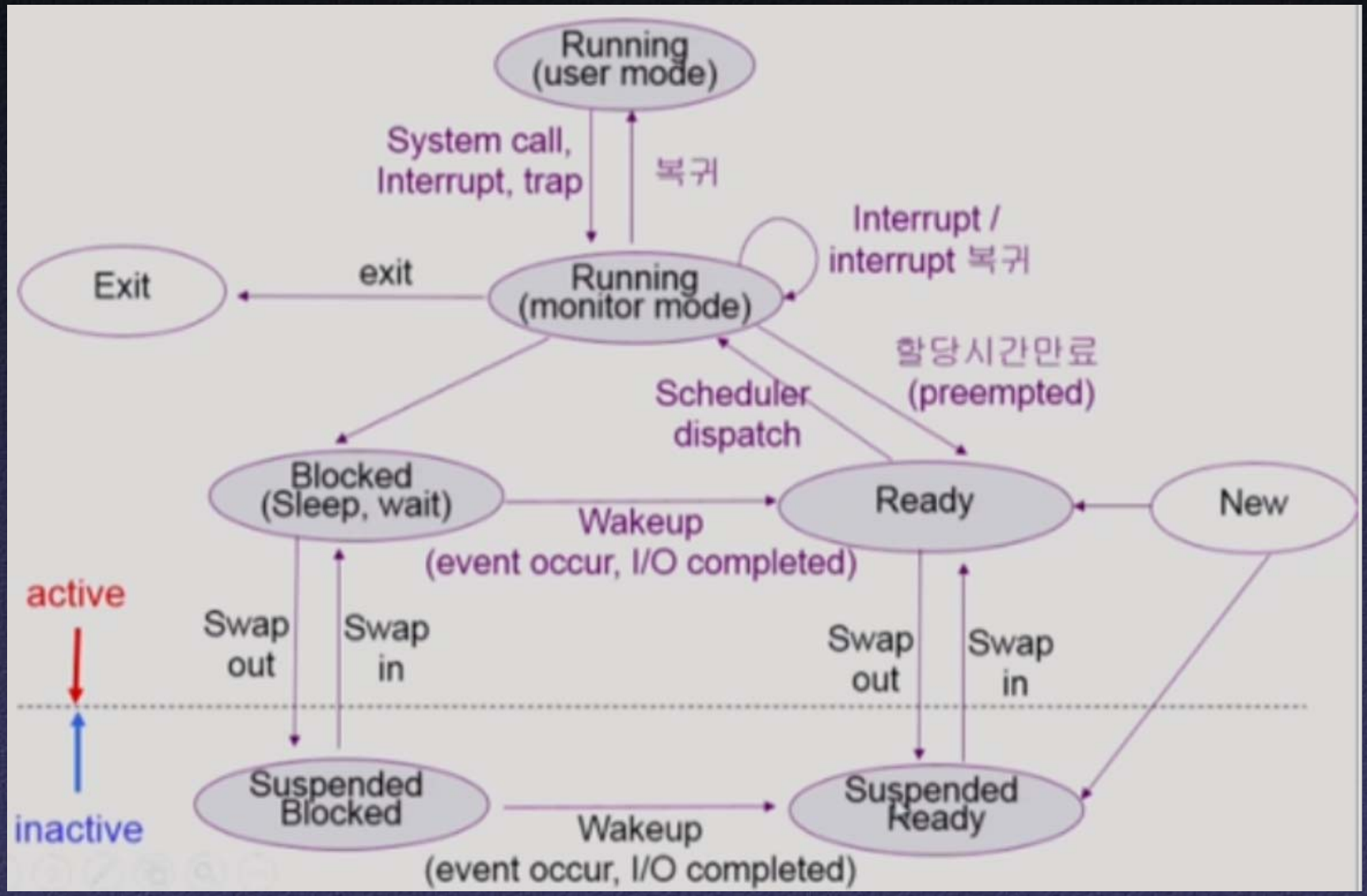
- 외부적인 이유로 프로세스의 수행이 정지된 상태
- 프로세스는 통째로 디스크에 swap out 된다
- (예) 사용자가 프로그램을 일시 정지시킨 경우 (break key)
  시스템이 여러 이유로 프로세스를 잠시 중단시킴
  (메모리에 너무 많은 프로세스가 올라와 있을 때)

*** Blocked : 자신이 요청한 event 가 만족되면 Ready / Suspended : 외부에서 resume go 주어야 Active*

프로세스의 상태도


Running
(user mode)
System call,
Interrupt, trap
복귀
Interrupt /
interrupt 복귀
Running
(monitor mode)
할당시간만료
(preempted)
Scheduler
dispatch
Blocked
(Sleep, wait)
Ready
Wakeup
(event occur, I/O completed)

프로세스의 상태도


Running
(user mode)
System call,
Interrupt, trap
복귀
Interrupt /
interrupt 복귀
Exit
exit
Running
(monitor mode)
할당시간만료
(preempted)
Scheduler
dispatch
Blocked
(Sleep, wait)
Wakeup
(event occur, I/O completed)
Ready
New
active
Swap
out
Swap
in
Swap
out
Swap
in
inactive
Suspended
Blocked
Wakeup
(event occur, I/O completed)
Suspended
Ready

프로세스의 생성 (Process Creation)

부모 프로세스(Parent process)가 자식 프로세스(Child process) 생성

프로세스의 트리(계층 구조) 형성

프로세스는 자원을 필요로 함

→ 운영체제로부터 받는다.

→ 부모와 공유한다.

Copy-on-write(COW)

자원의 공유

→ 부모와 자식이 모든 자원을 공유하는 모델

→ 일부를 공유하는 모델

→ 전혀 공유하지 않는 모델

수행 (Execution)

→ 부모와 자식은 공존하며 수행되는 모델

→ 자식이 종류(terminate) 될 때까지 부모가 기다리는(wait) 모델

프로세스의 생성 (Process Creation)

주소 공간 (Address space)

→ 자식은 부모의 공간을 복사함 (binary and OS data)

→ 자식은 그 공간에 새로운 프로그램을 올림

유닉스의 예

→ *fork()* 시스템 콜이 새로운 프로세스를 생성

- 부모를 그대로 복사 (OS data except PID + binary)

→ 주소 공간 할당

→ fork 다음에 이어지는 *exec()* 시스템 콜을 통해 새로운 프로그램을 메모리에 올림

fork() 시스템 콜

A process is created by the **fork()** system call

→ Creates a new address space that is a duplicate of the caller

```
int main()
{
   int pid;
   if (pid == 0)
     printf("\n Hello, I am child \n");
   else if (pid > 0)
     printf("\n Hello, I am parent \n");
}
```

Parent process
pid > 0

Child process
pid = 0

fork() 시스템 콜

A process is created by the **fork()** system call

→ Creates a new address space that is a duplicate of the caller

```
int main()
{
  int pid;
  if (pid == 0)
    printf("\n Hello, I am child \n");
  else if (pid > 0)
    printf("\n Hello, I am parent \n");
}
```

```
int main()
{
  int pid;
  if (pid == 0)
     printf("\n Hello, I am child \n");
  else if (pid > 0)
     printf("\n Hello, I am parent \n");
}
```

exec() 시스템 콜

A process can execute a different program by the **exec()** system call

→ Replaces the memory image of the caller with a new program

```
int main()
{
 int pid;
 pid = fork()
 if (pid == 0){
   printf("\n Hello, I am child
             Now I'll run date \n");
   execlp("/bin/date", "/bin/date",
                          (char *)0);
 }
 else if (pid > 0)
   printf("\n Hello, I am parent \n");
}
```

```
int main()
{
    ……
}
```

exec() 시스템 콜

A process can execute a different program by the **exec()** system call

→ Replaces the memory image of the caller with a new program

```
int main()
{
 printf("\n Hello, I am child Now I'll run date \n");
 execlp("/bin/date", "/bin/date", (char *)0);
 printf("\n Hello, I am parent \n");
}
```

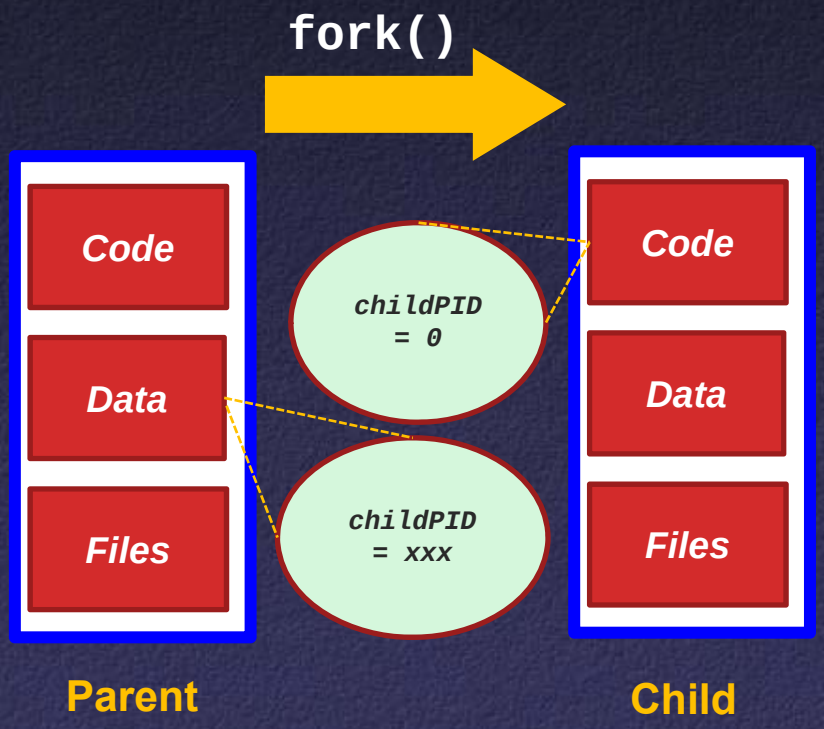
wait() 시스템 콜

프로세스 A가 **wait()** 시스템 콜을 호출하면

→ 커널은 child 가 종료될 때까지 프로세스 A를 sleep시킨다 (blocked 상태)

→ Child process가 종료되면 커널은 프로세스 A를 깨운다 (ready 상태)

```
int main()
{
 int childPID;
 S1;
 childPID = fork();
 if(childPID == 0)
     <code for child process>
 else {
     wait();
 }
 S2;
}
```



프로세스의 종료 (Process Termination)

프로세스가 마지막 명령을 수행한 후 운영체제에게 이를 알려줌(**exit**)

→ 자식이 부모에게 ouput data를 보냄 (via **wait**)

→ 프로세스의 각종 자원들이 운영체제에게 반납됨

부모 프로세스가 자식의 수행을 종료시킴 (**abort**)

→ 자식이 할당 자원의 한계치를 넘어섬

→ 자식에게 할당된 태스크가 더 이상 필요하지 않음

→ 부모가 종료(exit)하는 경우

- 운영체제는 부모 프로세스가 종료하는 경우 자식이 더 이상 수행되도록 두지 않는다.
- 단계적인 종료

`exit()` 시스템 콜

프로세스의 종료

→ 자발적 종료

- 마지막 statement 수행후 **`exit()`** 시스템 콜을 통해
- 프로그램에 명시적으로 적어주지 않아도 main 함수가 리턴되는 위치에 컴파일러와 넣어줌

→ 비자발적 종료

- 부모 프로세스가 자식 프로세스를 강제 종료 시킴
  - ✓ 자식 프로세스가 한계치를 넘어서는 자원 요청
  - ✓ 자식에게 할당된 태스크가 더 이상 필요하지 않음
- 키보드로 **`kill, break`** 등을 친 경우
- 부모가 종료하는 경우
  - ✓ 부모 프로세스가 종료하기 전에 자식들이 먼저 종료됨

프로세스와 관련한 시스템 콜

- fork() Create a child (copy)
- exec() Overlay new image
- wait() Sleep until child is done
- exit() Frees all the resources, notify parent

프로세스 간 협력

독립적 프로세스(independent process)

→ 프로세스는 각자의 주소 공간을 가지고 수행되므로 원칙적으로 하나의 프로세스는 다른 프로세스의 수행에 영향을 미치지 못함.

협력 프로세스(Cooperating process)

→ 프로세스는 협력 메커니즘을 통해 하나의 프로세스가 다른 프로세스의 수행에 영향을 미칠 수 있음

프로셋 간 협력 메커니즘(*IPC : Inter-process Communication*)

→ 메지시 전달하는 방법

- ***Message passing*** : 커널을 통해 메시지 전달

→ 주소 공간을 공유하는 방법

- ***Shared memory*** : 서로 다른 프로세스 간에도 일부 주소 공간을 공유하게 하는 공유 메모리 메커니즘이 있음.

*** Thread는 사실상 하나의 프로세스이므로 프로세스간 협력으로 보기 어렵지만 동일한 process를 구성하는 thread들 간에는 주소를 공유하므로 협력이 가능*

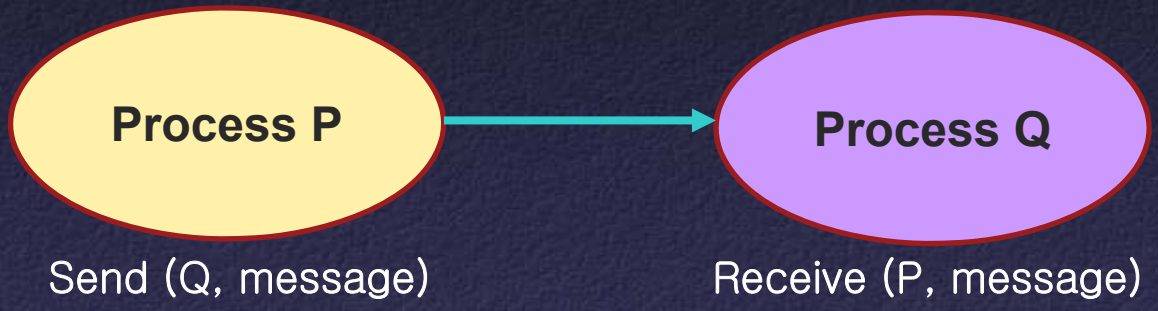
Message Passing

Message system

→ 프로세스 사이에 공유 변수(shared variable)를 일체 사용하지 않고 통신하는 시스템

Direct Communication

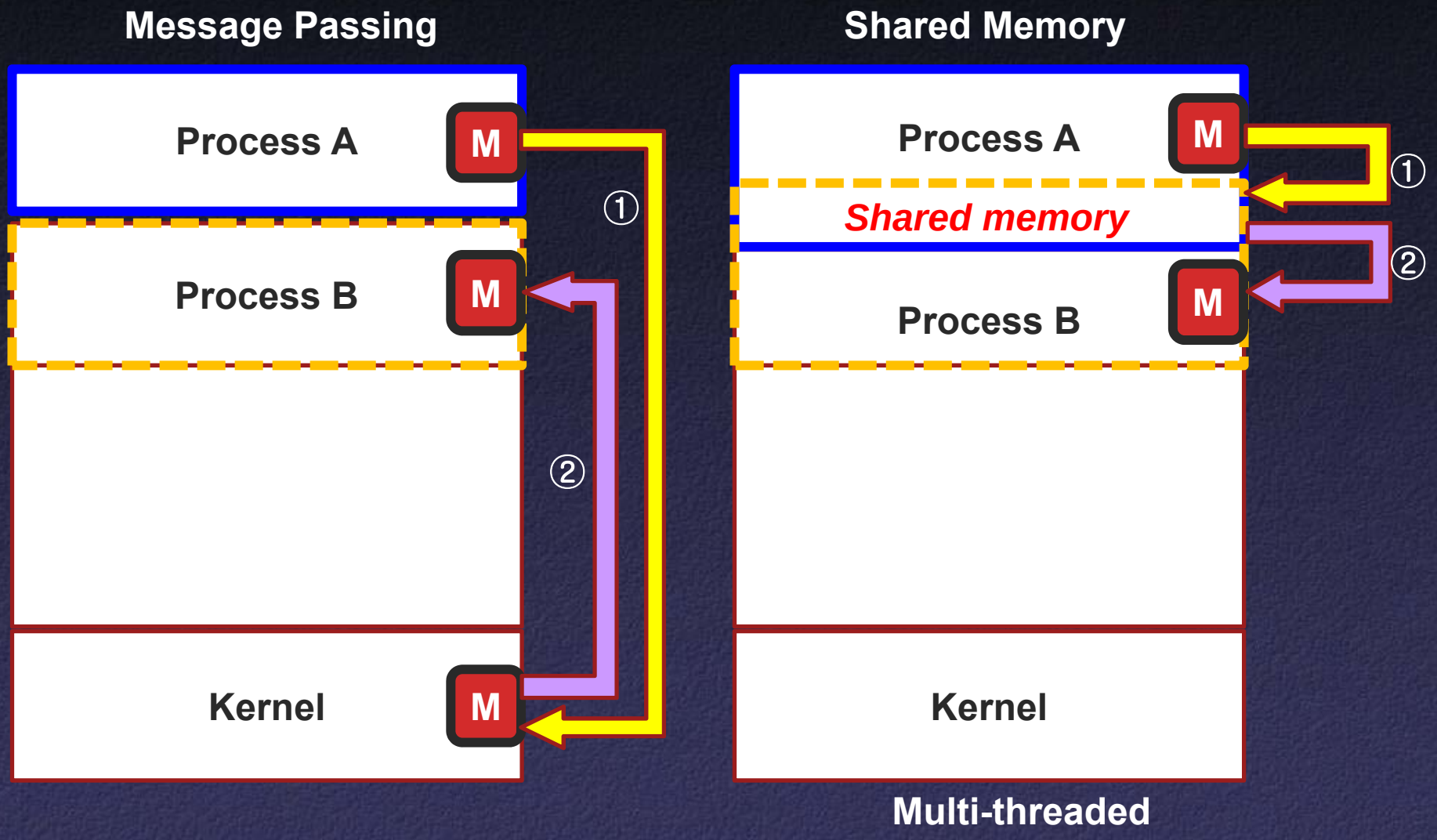
→ 통신하려는 프로세스의 이름을 명시적으로 표시



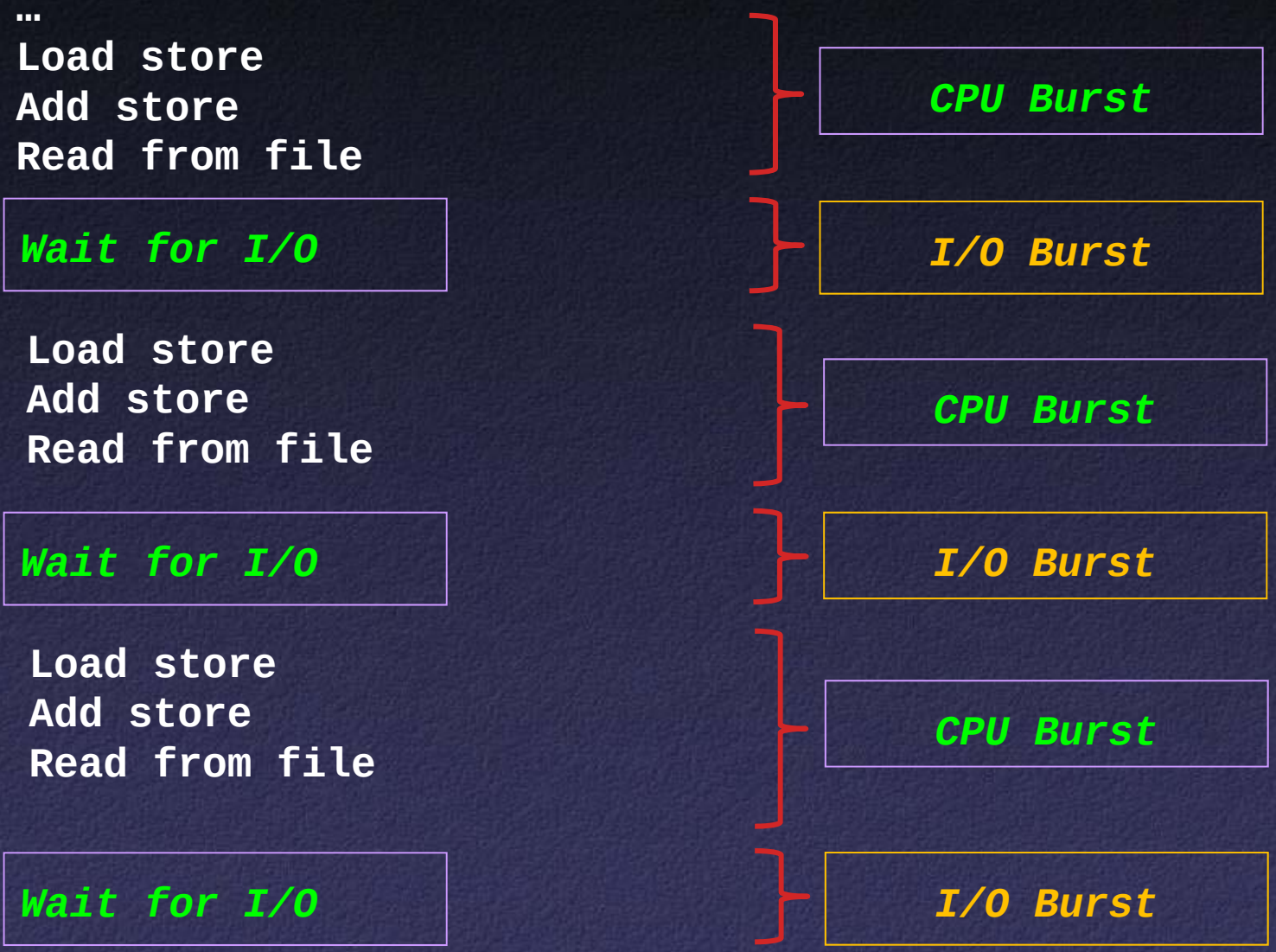
Indirect Communication

→ Mailbox (또는 port)를 통해 메시지를 간접 전달

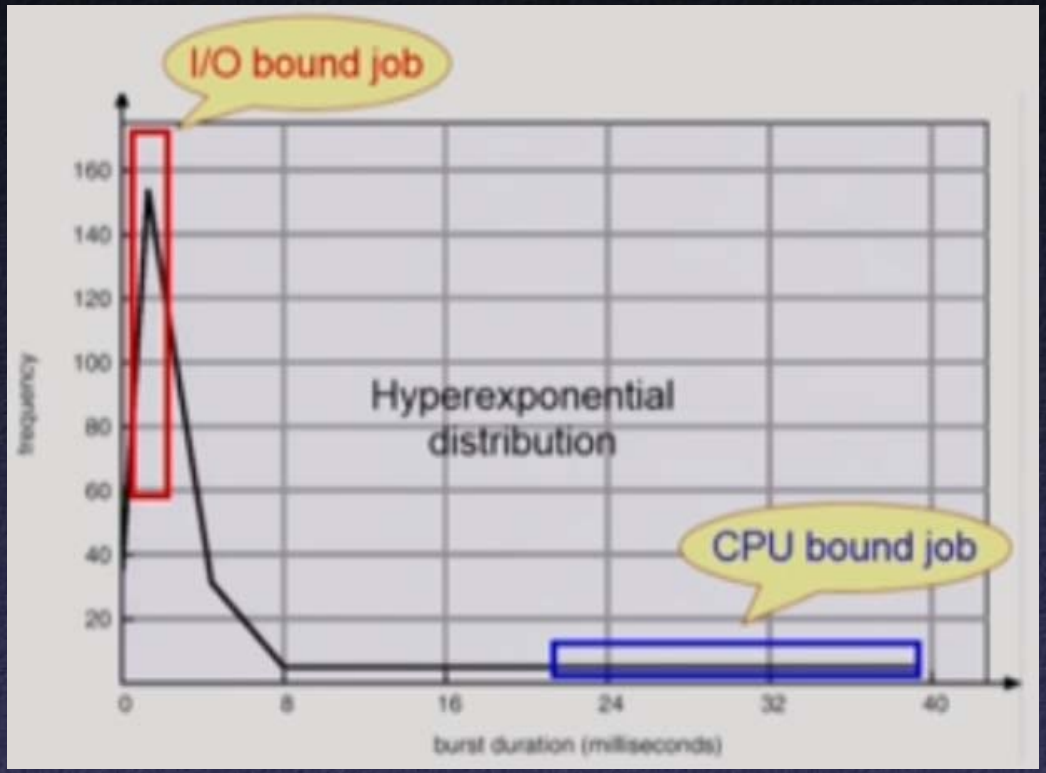
Inter-process Communication



CPU and I/O Burst in Program Execution



CPU-burst Time의 분포



여러 종류의 job (=process)이 섞여 있기 때문에 CPU 스케줄링이 필요

→ Interactive job에게 적절한 response 제공 요망

→ CPU와 I/O 장치 등 시스템 자원을 골고루 효율적으로 사용

프로세스의 특성 분류

프로세스는 그 특성에 따라 다음 두 가지로 나눔

→ I/O bound process

- CPU를 잡고 계산하는 시간보다 I/O에 많은 시간이 필요한 Job
- (many short CPU bursts)

→ CPU bound process

- 계산 위주의 job
- (few very long CPU bursts)

CPU Scheduler & Dispatcher

CPU Scheduler

→ Ready 상태의 프로세스 중에서 이번에 CPU를 줄 프로세스를 고른다.

Dispatcher

→ CPU의 제어권을 CPU scheduler에 의해 선택된 프로세스에게 넘긴다.

→ 이 과정을 context switch(문맥 교환)라고 한다.

CPU 스케줄링이 필요한 경우는 프로세스에게 다음과 같은 상태 변화가 있는 경우이다.

1. Running → Blocked (예 : I/O 요청하는 시스템 콜)
2. Running → Ready (예 : 할당시간 만료로 timer interrupt)
3. Blocked → Ready (예 : I/O 완료후 인터럽트)
4. Terminate

- ***1, 4에서의 스케줄링은 non-preemptive (=강제로 빼앗지 않고 자진 반납***
- ***All other scheduling is preemptive (= 강제로 빼앗음)***

Scheduling Criteria

Performance Index (= Performance Measure, 성능 척도)

CPU utilization

→ Keep the CPU as busy as possible

Throughput

→ # of processes that complete their execution per time unit

Turnaround time

→ Amount of time to execute a particular process

Waiting time

→ Amount of time a process has been waiting in the ready

Response time

→ Amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

3장 프로세스 기술(Description) 및 제어(Control)

3장의 학습 목표

- 프로세스(process)를 정의하고, 프로세스들과 프로세스 제어 블록들 사이의 관계를 설명할 수 있다.
- 프로세스 상태의 개념을 설명하고, 프로세스들의 상태 전이에 대해 설명할 수 있다.
- 프로세스들을 관리하기 위해, 운영체제가 사용하는 자료 구조 및 자료구조 구성요소들의 목적을 나 열하고 설명할 수 있다.
- 운영체제에서 프로세스 제어를 위한 요구사항들을 평가할 수 있다.
- OS 코드의 실행에 관련된 이슈들을 이해할 수 있다.
- UNIX SVR4의 프로세스 관리 기법을 설명할 수 있다.

목 차

운영체제 요구조건

- 프로세스 관리: 기본 요구 조건
  - 운영체제는 적절한 응답 시간을 제공하면서 처리기 이용률을 극대화할 수 있도록 여러 프로세스 수행을 인터리빙(interleaving) 해야 한다.
  - 운영체제는 교착상태[1]를 회피함과 동시에 특정 정책(예를 들어 어떤 함수 또는 응용에 보다 높은 우선순위를 부여)에 부합하도록 자원을 프로세스에게 할당해야 한다.
  - 운영체제는 프로세스 간 통신(Interprocess Communication)과 사용자의 프로세스 생성을 지원해야 하는데, 이들 모두 응용을 구조화하는데 도움이 될 수 있다.

프로세스란?

- 배경 지식
  - 컴퓨터 플랫폼은 하드웨어 자원들의 집합으로 구성
  - 컴퓨터 응용은 어떤 업무를 수행하기 위해 개발됨
  - 주어진 하드웨어 플랫폼 상에서 직접 응용을 작성하는 것은 비효율적
    - 공통 루틴 (common routines) 개발 유도
    - 자원 공유 (resource sharing)를 위한 소프트웨어 필요
    - 자원 보호 (resource protection) 기법 필요
  - OS는 응용이 사용할 수 있는 편리하고, 풍부하고, 안전하고, 일관성 있는 인터페이스를 제공
  - OS는 자원들에 대해 균일하고 추상화된 표현(representation)을 제공
    - 응용은 해당 자원을 요청하고 접근 가능

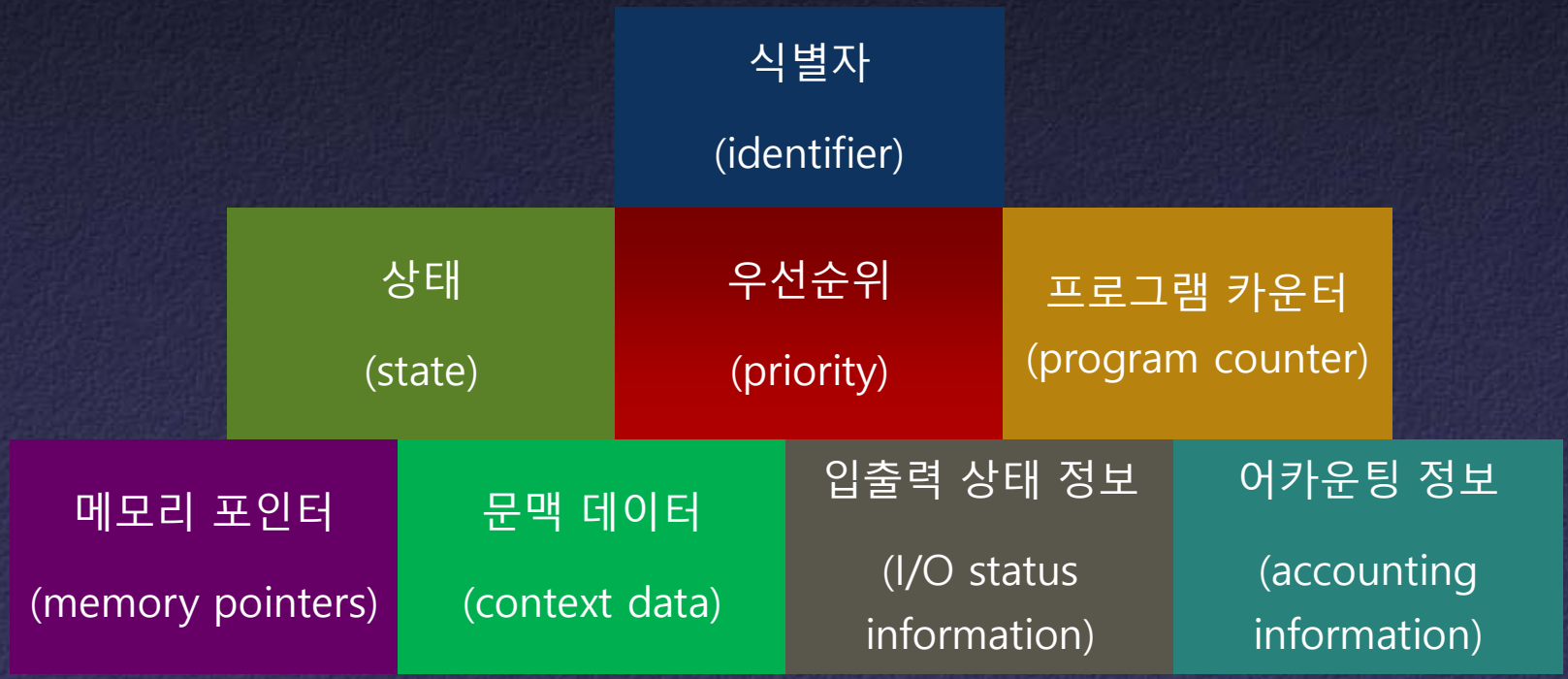
☞ Process: model the execution of applications
(main concept for multiprogramming, time sharing system)

프로세스란?

- 정의
  - 수행 중인 프로그램 (A program in execution): active entity
  - 컴퓨터에서 수행 중인 한 프로그램의 인스턴스(instance)
  - 한 처리기 상에 할당되어 실행될 수 있는 개체
  - 명령들의 순차 실행, 현재 상태, 연관되어 있는 시스템 명령들의 집합 등으로 특징지어지는 '활동의 단위'(a unit of activity)
- 구조 (Structures)
  - 프로그램 코드 (program code)
  - 일련의 데이터 (a set of data), stack (user stack vs. kernel stack)
  - 프로세스 정보 (process information)
- Program
  - Passive entity (code and data in a file system)

프로세스 구성요소(element)

- 프로그램이 실행될 때, 해당 프로세스는 여러 구성요소들에 의해 유일하게 특징지어질 수 있음



프로세스 상태: 수행 궤적 (1/3)

- A, B, C 프로세스들의 수행 궤적

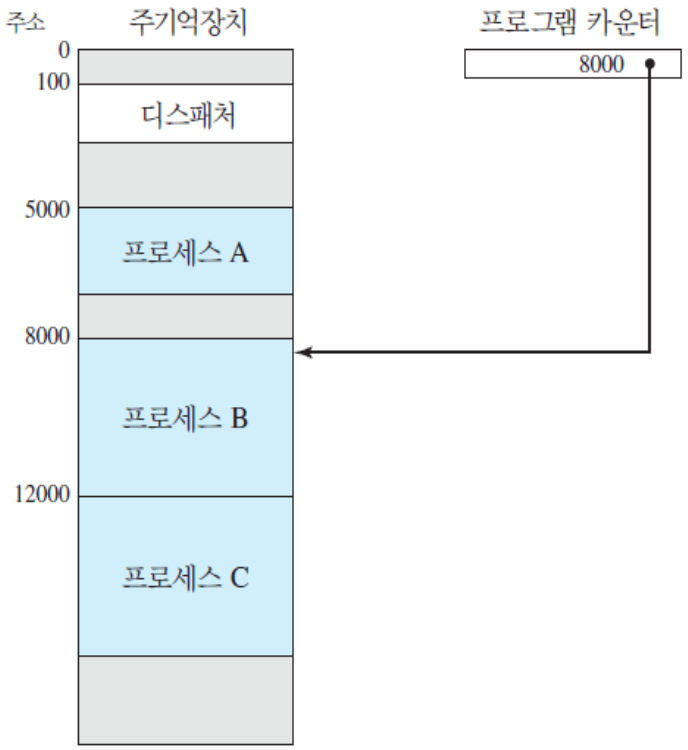


그림 3.2 명령어 사이클 13에서 단순 수행(그림 3.4) 예

(a) 프로세스 A의 궤적	(b) 프로세스 B의 궤적	(c) 프로세스 C의 궤적
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of process A
8000 = Starting address of program of process B
12000 = Starting address of program of process C

그림 3.3 그림 3.2에 있는 프로세스들의 궤적

프로세스 상태: 수행 궤적 (2/3)

- 처리기 관점에서 본 프로세스들의 수행 궤적

1	5000	27	12004
2	5001	28	12005
3	5002	----------시간 만료	
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------시간 만료		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	]103	36	5007
11	]104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------시간 만료	
16	8003	41	100
----------I/O 요청		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		----------시간 만료	

100 = 디스패처 프로그램의 시작 주소

그림자가 있는 영역은 디스패처 프로세스의 수행을 나타냄
첫 번째 및 세 번째 열은 명령어 사이클 번호를 나타냄
두 번째 및 네 번째 열은 수행 중인 명령어들의 주소를 나타냄

그림 3.4 그림 3.2 프로세스들의 혼합된 궤적

프로세스 상태: 수행 궤적 (3/3)

- Timing Diagram: 프로세스 상태 도입

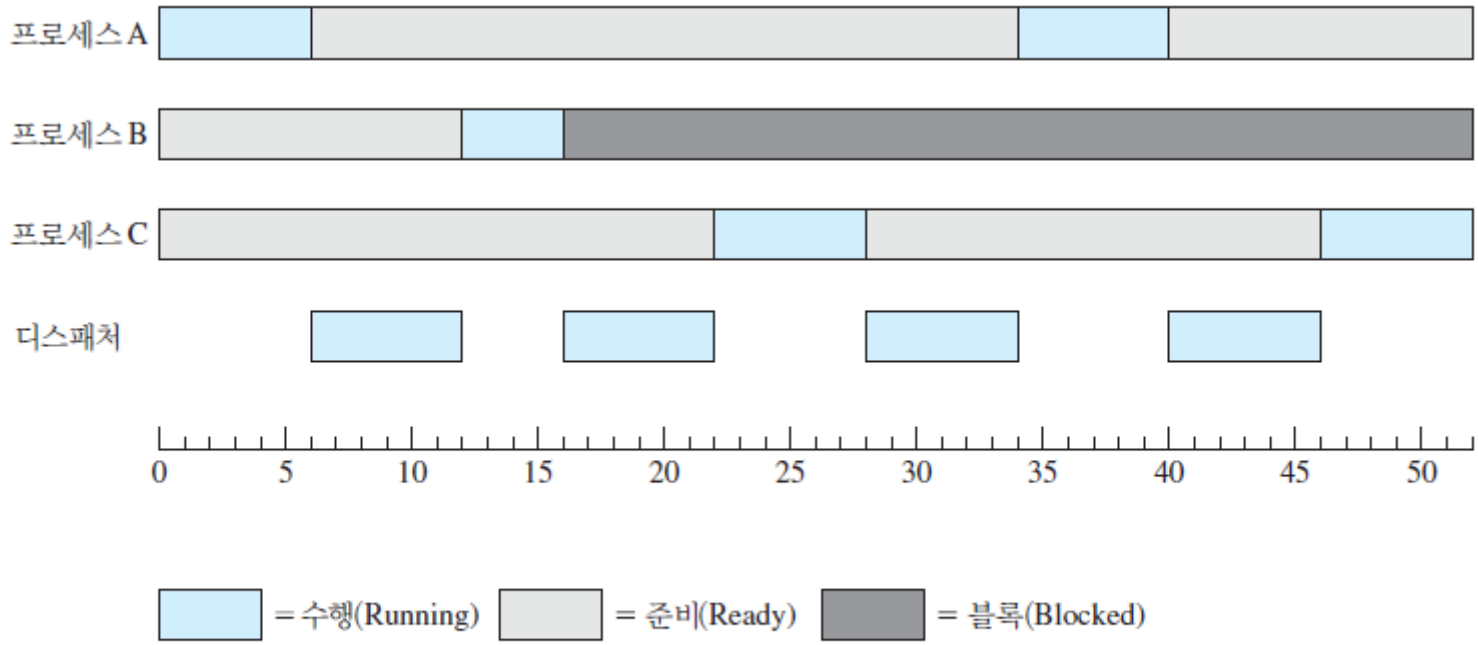
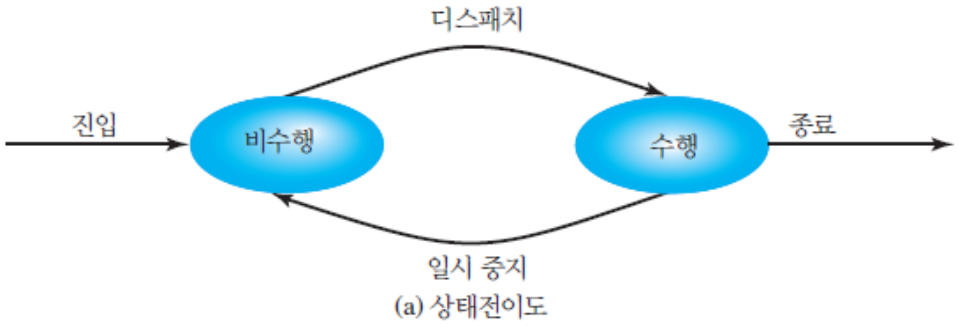


그림 3.7 그림 3.4의 궤적에 대한 프로세스 상태

프로세스 상태

- 2-상태 프로세스 모델
  - 수행(Running) 및 비수행(Not-running)



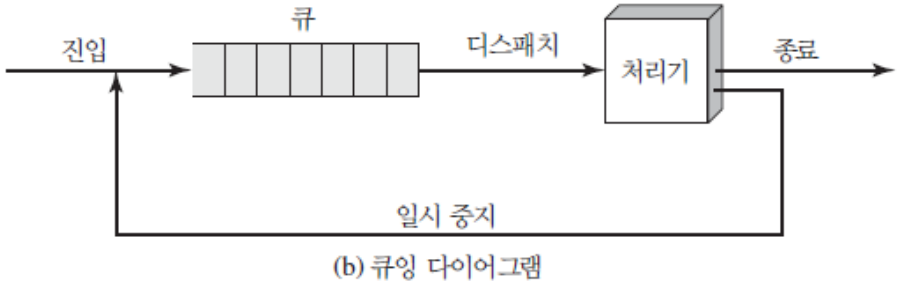


그림 3.5 2-상태 프로세스 모델

☞ each entry of the queue is a pointer to the PCB or linked list of PCBs

프로세스 상태

- 프로세스 생성: 주소공간 생성 및 자료구조 구축
  - *fork* (Linux) 또는 *CreateProcess* (Windows)를 이용함
  - 부모 프로세스 대 자식 프로세스

표3.1 프로세스 생성 이유

새로운 일괄처리 작업 (new batch job)	보통 테이프나 디스크를 통해 운영체제에게 일괄처리 작업 제어 스트림이 제공된다. 운영체제가 새로운 작업을 처리할 준비가 되면, 다음에 수행할 일련의 작업제어 명령을 읽어 들일 것이다.
대화형 로그온	사용자가 터미널에서 시스템에 로그온 한다.
서비스를 제공하기 위해 운영체제가 생성	사용자가 대기할 필요 없도록, 운영체제는 사용자 프로그램을 대신해 어떤 기능을 수행할 프로세스(예를 들어, 프린트 작업을 제어하는 프로세스)를 생성할 수 있다.
기존 프로세스에 의한 생성 (spawn)	모듈화를 위해서나 병렬성을 활용하기 위해, 사용자 프로그램은 많은 프로세스의 생성을 명령할 수 있다.

프로세스 상태

- 프로세스 종료

표3.2 프로세스 종료 이유

정상 완료	프로세스가 수행 완료를 알리는 운영체제 서비스 호출 수행
시간 한도 초과	프로세스가 명시된 전체 시간 한도보다 더 오래 수행되었다. 측정될 수 있는 시간의 종류에는 전체 경과 시간(실제 시간, wall clock time), 수행에 사용된 시간, 그리고 대화형 프로세스의 경우 마지막 사용자 입력이 제공된 이후 시간 등 여러 가지가 있을 수 있다.
메모리 부족	프로세스가 시스템이 제공할 수 있는 메모리보다 더 많은 용량 요청
경계범위 위반	프로세스가 접근이 허용되지 않는 메모리 위치에 접근하려 시도한다.
보호 오류	프로세스가 사용 금지된 자원이나 파일을 사용하려 하거나, 읽기전용 파일에 쓰기를 시도하는 등 파일을 잘못 사용하려 한다.
산술 오류	프로세스가 0으로 나누기와 같은 금지된 계산을 하거나, 하드웨어가 수용할 수 있는 것보다 큰 숫자의 저장을 시도한다.
시간 초과	프로세스가 어떤 이벤트 발생을 명시된 최대 시간을 초과하여 기다렸다.

프로세스 상태

- 프로세스 종료 (계속)

입출력 실패	원하는 파일을 찾을 수 없거나, 테이프에서 결함이 있는 부분을 만났을 때와 같이 명시된 최대 횟수 만큼의 시도 후에도 읽기 또는 쓰기가 실패한 경우, 또는 라인 프린터로부터의 읽기와 같이 유효하지 않은 연산을 시도한 경우 등, 입출력 도중 오류가 발생한다.
무효 명령어	프로세스가 (데이터 역역으로 분기한 뒤 그 데이터를 수행하려는 것과 같이) 존재하지 않는 명령어의 수행을 시도한다.
특권 명령어	프로세스가 운영체제만 사용하도록 예약된 명령어를 사용하려 시도한다.
데이터 오용	데이터 일부의 타입이 잘못되었거나 초기화되지 않았다.
오퍼레이터나 운영체제 간섭	어떤 이유(예를 들어 교착상태 발생)로 인하여 오퍼레이터 또는 운영체제가 프로세스를 종료시켰다.
부모 종료	부모 프로세스가 종료될 때, 운영체제는 그 프로세스가 생성한 모든 자식(offspring) 프로세스를 자동으로 종료시킬 수 있다.
부모 요청	부모 프로세스는 일반적으로 자신이 생성한 임의의 자식 프로세스를 종료시킬 수 있는 권한을 가지고 있다.

프로세스 상태

- 5-상태 프로세스 모델(5-state model)
  - 수행 (Running)
  - 준비 (Ready)
  - 블록 (Blocked)
  - 생성 (New)
  - 종료 (Exit)

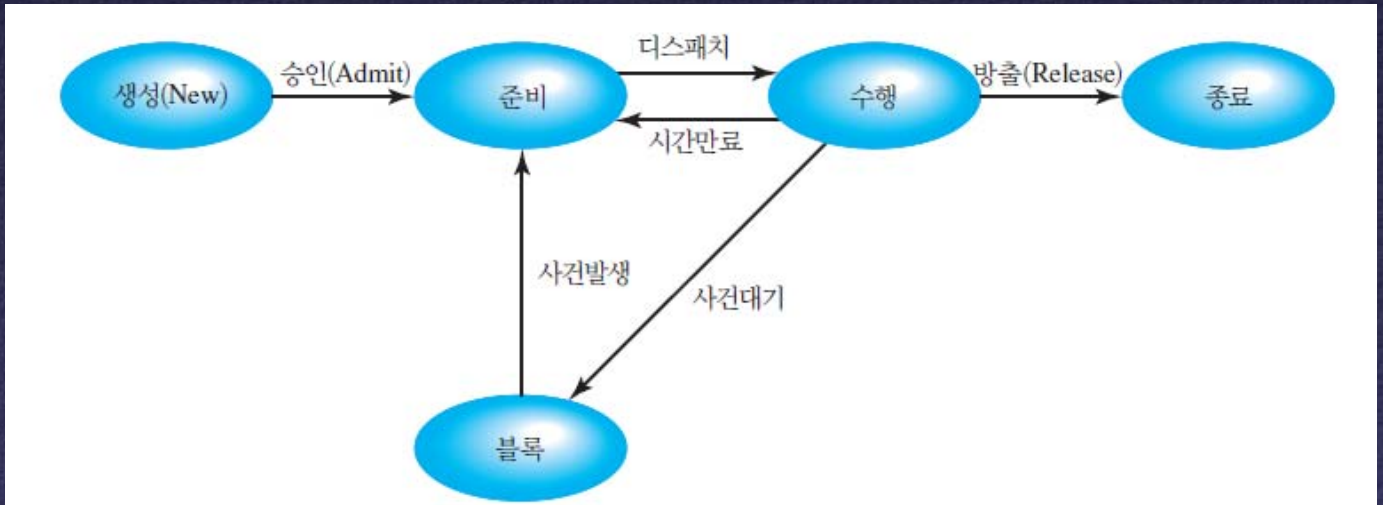


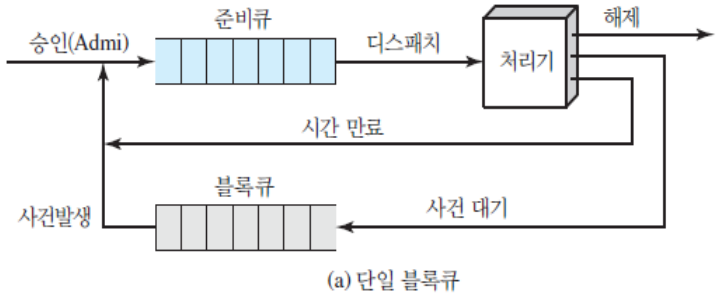
그림 3.6 5-상태 모델

프로세스 상태

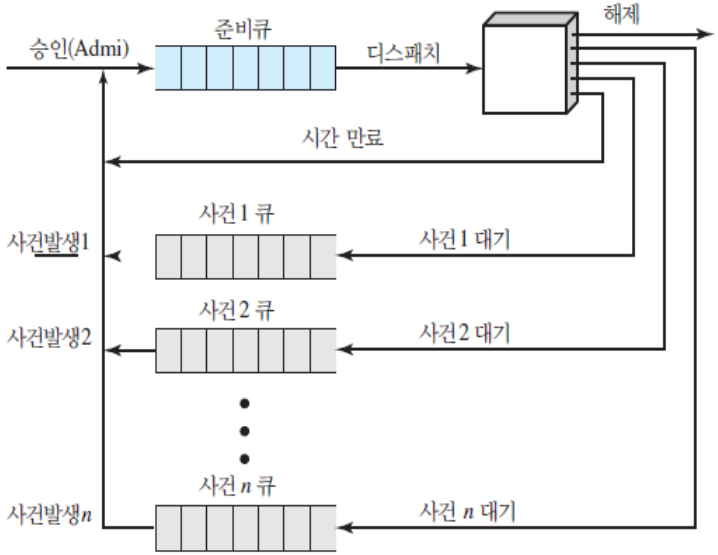
- 5-상태 프로세스 모델: 상태 전이
  - Null ➔ 생성(New): 생성(create)
  - 생성 ➔ 준비(Ready): 승인(admit)
  - 준비 ➔ 수행(Running): 디스패치(dispatch)
  - 수행 ➔ 준비: 선점 (시간만료 또는 높은 우선순위)
  - 수행 ➔ 블록(Blocked): 사건 대기 (수면)
  - 블록 ➔ 준비: 사건 발생 (깨움)
  - 수행 ➔ 종료(Exit): 종료 (또는 완료)
  - 준비 ➔ 종료
  - 블록 ➔ 종료

프로세스 상태

- 5-상태 프로세스 모델
  - 여러 큐를 사용
  - 특히, 여러 개의 블록 큐



(a) 단일 블록큐



(b) 멀티 블록큐

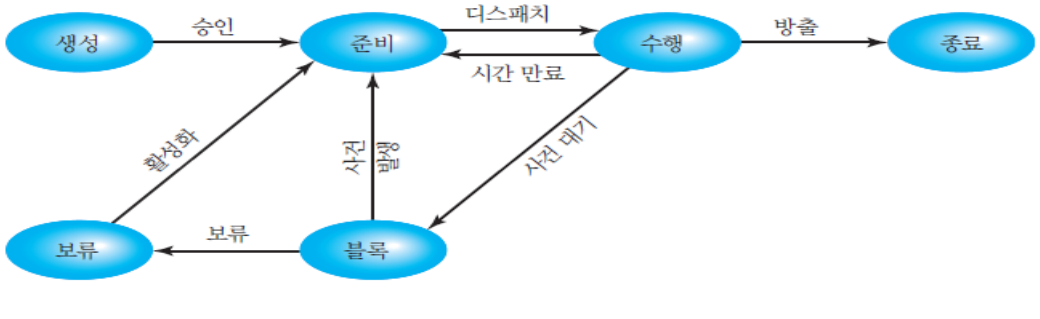
그림 3.8 그림 3.6에 대한 큐잉 모델

프로세스 상태

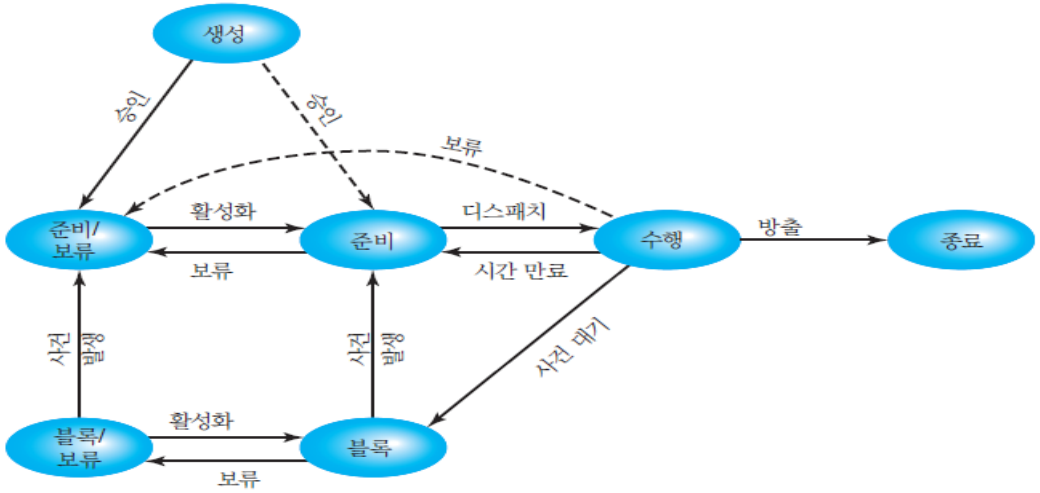
- 보류 상태의 프로세스(Suspended process) 도입
- 스와핑 (Swapping)
  - 스왑-인(swap-in) 및 스왑-아웃(swap-out)
  - 스왑-아웃
    - 종료되지 않은 프로세스의 전체 이미지 (또는 일부 이미지)를 주기억장치로부터 '스왑 공간'(디스크의 일부로 swap space 또는 swap area)으로 이동시키는 것
  - 스왑-인
    - 스왑-아웃된 프로세스의 전체 이미지 (또는 일부 이미지)를 스왑 공간으로부터 주기억장치로 적재하는 것
- 스와핑도 I/O 연산임

프로세스 상태

- 보류 상태를 가진 프로세스 상태 전이도



(a) 보류 상태가 하나인 경우



(b) 보류 상태가 둘인 경우

그림 3.9 보류 상태를 가진 프로세스 상태 전이도

프로세스 상태

- 스와핑의 필요성
  - 더 많은 가용 메모리(free memory)를 확보하기 위해
  - 주기억장치에 준비 상태의 프로세스가 없을 때

- 보류 (Suspension)

표3.3 프로세스 보류에 대한 이유

스와핑	수행할 준비가 된 프로세스를 불러 들여오기 위해 운영체제는 충분한 주기억장치를 해제할 필요가 있다.
운영체제의 다른 이유	운영체제는 후면 프로세스, 유틸리티 프로세스, 또는 문제를 야기한다고 의심되는 프로세스를 보류시킬 수 있다.
대화식 사용자의 요청	사용자는 디버깅을 위해서 또는 자원 사용과 관련해서 프로그램의 수행을 보류시킬 수 있다.
타이밍	주기적으로 수행되는 프로세스(예로, 과금 프로세스나 시스템 모니터링 프로세스)가 다음 주기까지 대기할 때 보류될 수 있다.
부모 프로세스 요청	부모 프로세스는 보류된 자식 프로세스를 검사하거나 수정하기 위해, 또는 다양한 자손들의 활동을 조정하기 위해 자식 프로세스의 수행을 보류시킬 수 있다.

프로세스 상태

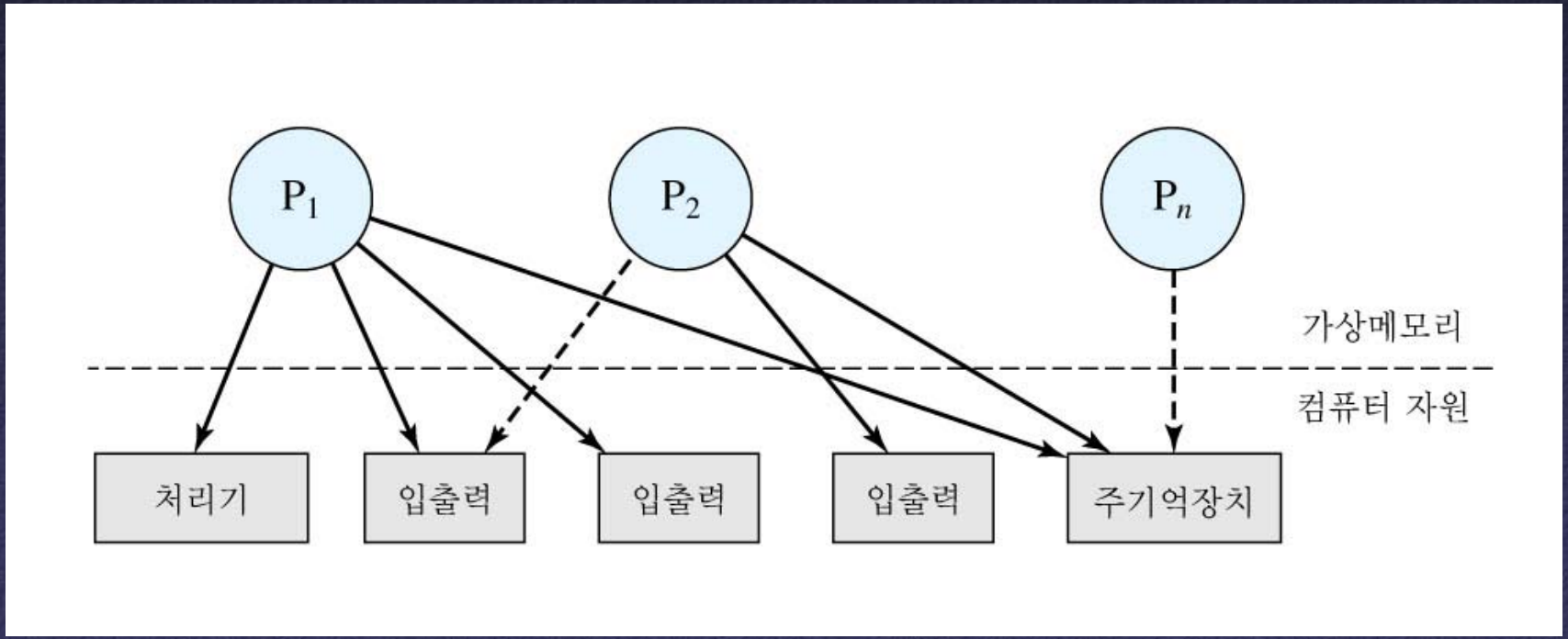
- 보류 프로세스를 포함한 상태 전이
  - Blocked ➔ Blocked/Suspend: 보류, 즉 스왑-아웃
  - Blocked/Suspend ➔ Ready/Suspend: 보류 상태에서 깨움
  - Ready/Suspend ➔ Ready: 활성화(activate), 즉 스왑-인
  - Ready ➔ Ready/Suspend: 보류, 즉 스왑-아웃

  - New ➔ Ready/Suspended
  - Blocked/Suspend ➔ Blocked
  - Running ➔ Ready/Suspend
  - 임의 상태 ➔ Exit

프로세스 기술(description): OS

- OS: 프로세스와 자원 관리

- 그림 3.10: 자원 할당의 예 (입출력은 I/O 장치를 의미)



프로세스 기술: OS

- OS 제어 구조(control structures)
  - 각 프로세스 및 자원의 현재 상태에 대한 정보를 유지
  - 보통, OS가 관리하는 각 개체(entity)를 위한 테이블이 구성됨
    - 메모리 테이블
    - I/O 테이블 = 입출력 테이블
    - 파일 테이블
    - 프로세스 테이블
  - 메모리 테이블
    - 프로세스들에 할당된 주기억장치 정보
    - 프로세스들에 할당된 2차 메모리(디스크) 정보
    - 공유 메모리 영역에 접근하기 위한 보호 속성(Protection attributes)
    - 가상 메모리(virtual memory)를 관리하는 필요한 정보

프로세스 기술: OS

- OS 제어 테이블의 일반적인 구조 (그림 3.11)

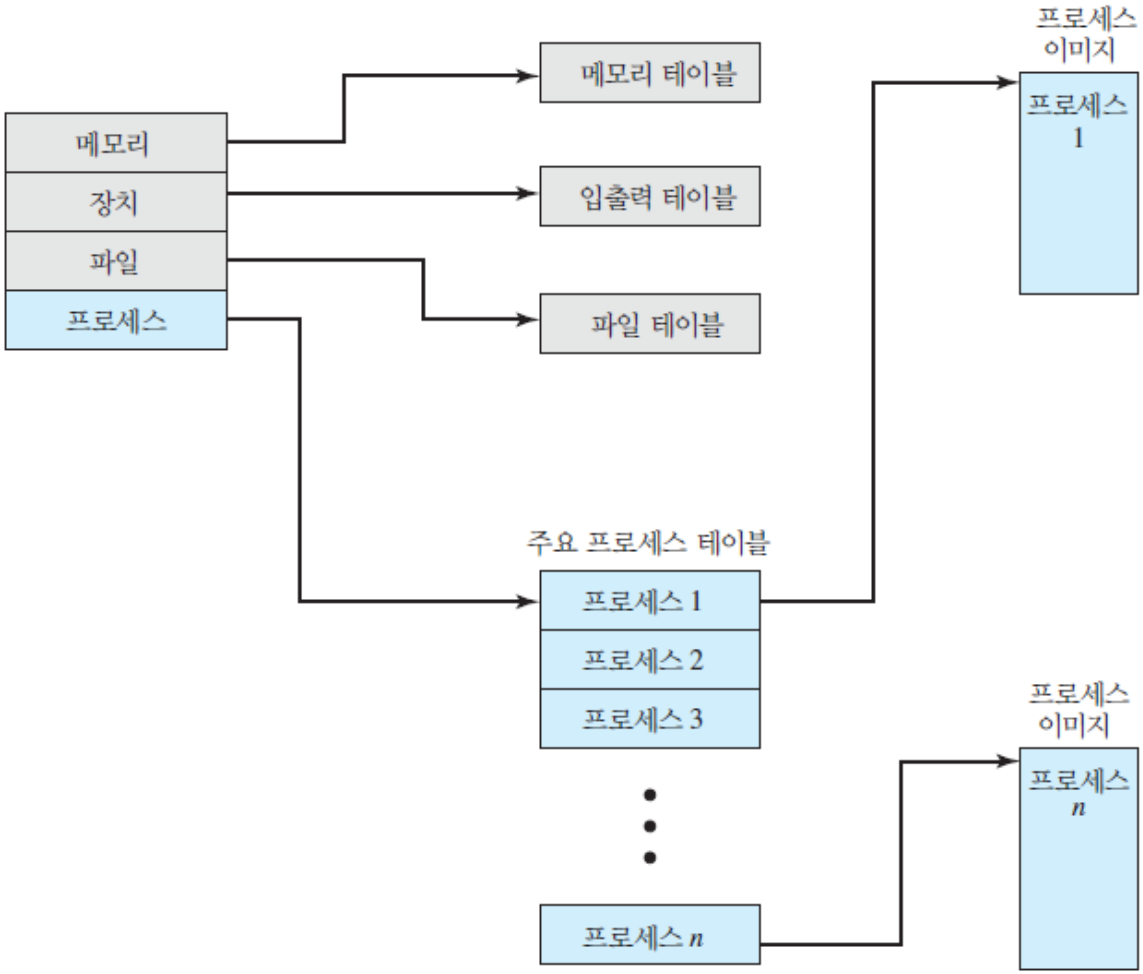


그림 3.11 운영체제 제어 테이블의 일반적인 구조

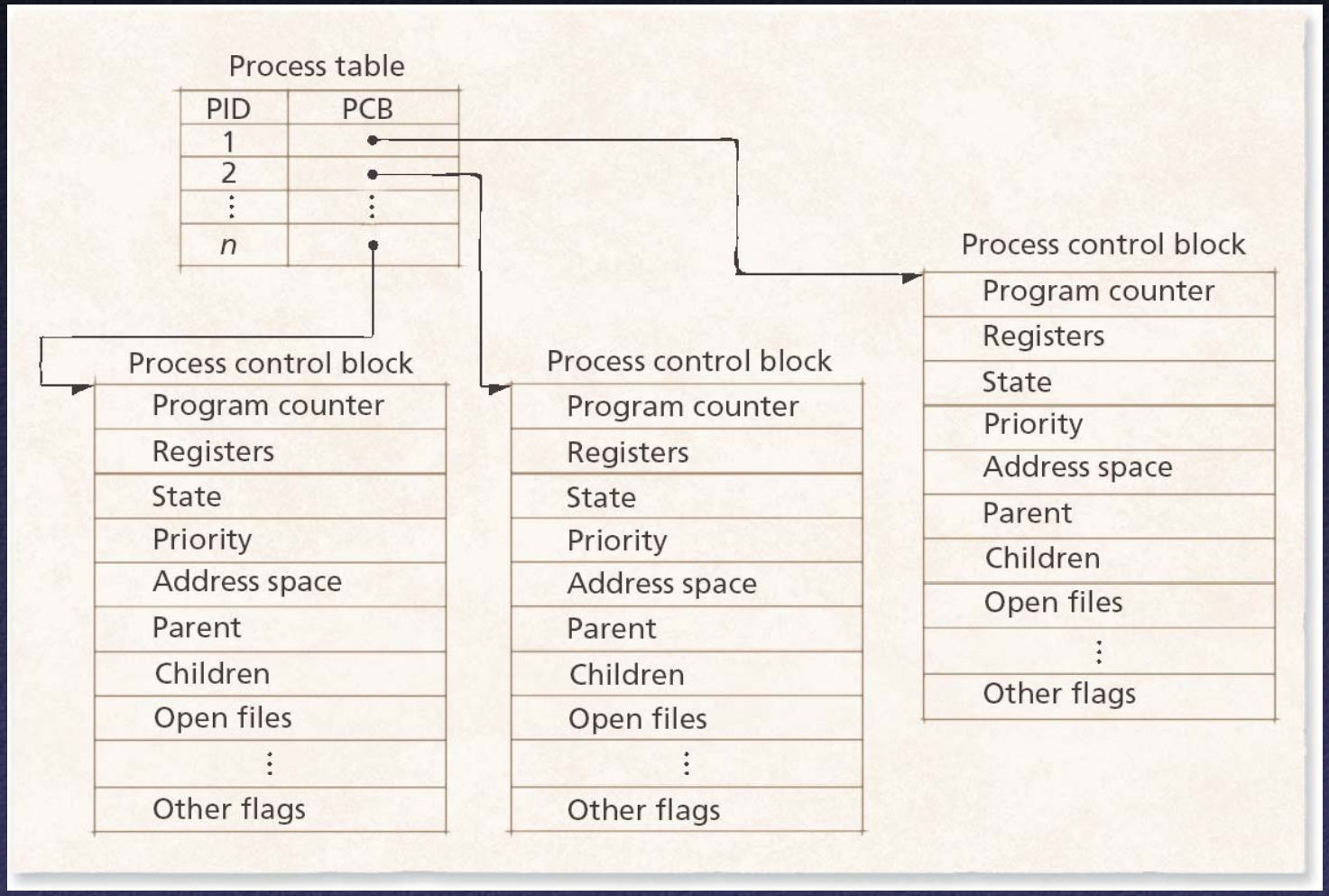
프로세스 기술: OS

- OS 제어 구조 (계속)
  - I/O 테이블
    - I/O 연산에 필요한 정보 유지
      - I/O 장치 이용 가능 또는 특정 프로세스에 할당되어 있음을 표시
    - I/O 전송에서 소스(source) 또는 목적지 주소로 사용될 주기억장치에서의 위치
  - 파일 테이블
    - 2차 메모리에서의 위치
    - 현재 상태, 속성(Attributes)
    - 파일 시스템을 관리하는데 필요한 정보
  - 프로세스 테이블
    - 프로세스의 위치 정보
    - 프로세스 제어블록(PCB)의 속성: 프로그램, 데이터, 스택

프로세스 제어 블록: process table

- Process table
  - The OS maintains pointers to each process's PCB in a system-wide or per-user process table
  - Allows for quick access to PCBs
  - When a process is terminated, the OS removes the process from the process table and frees all of the process's resources

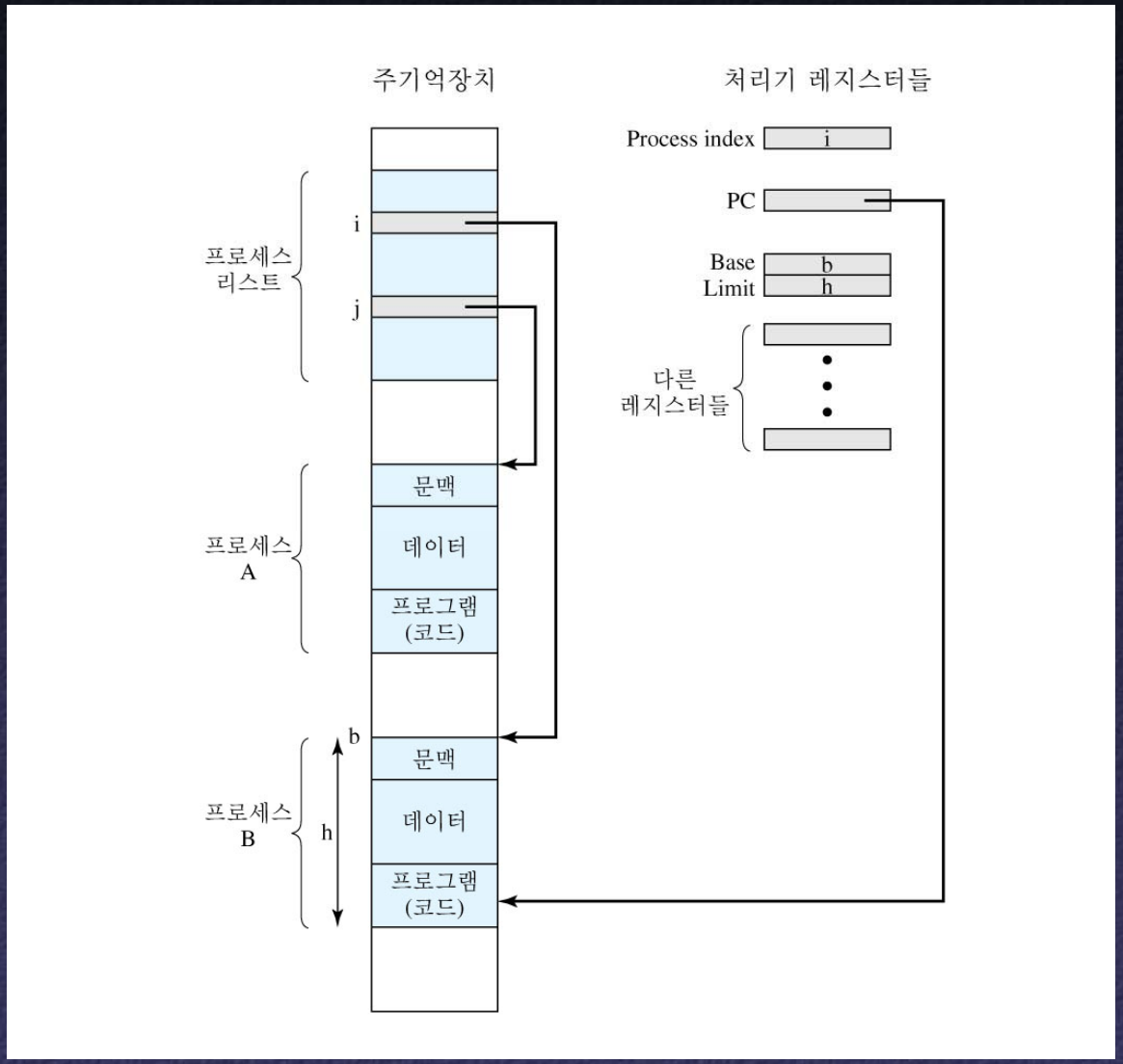
프로세스 제어 블록: process table



프로세스 기술: 프로세스 제어 구조

- 프로세스는 어디에 위치하나? → 그림 3.11 참조
- 프로세스 속성이란?
  - 프로세스를 제어하기 위해 OS가 사용하는 정보
  - 프로세스 제어 블록 ≡ 속성들의 집합
    - PCB ≡ 태스크제어블록 ≡프로세스 디스크립터(descriptor)
- 프로세스 이미지 (process image)
  - 프로그램 (code 또는 text)
  - 데이터
  - 스택
  - PCB (Process Control Block)
    - 프로세스 식별
    - 처리기 상태 정보
    - 프로세스 제어 정보

프로세스


주기억장치
처리기 레지스터들
Process index
i
PC
Base
b
Limit
h
다른
레지스터들
프로세스
리스트
i
j
문맥
데이터
프로그램
(코드)
프로세스
A
b
문맥
데이터
프로그램
(코드)
프로세스
B
h

프로세스 기술: 프로세스 제어 구조

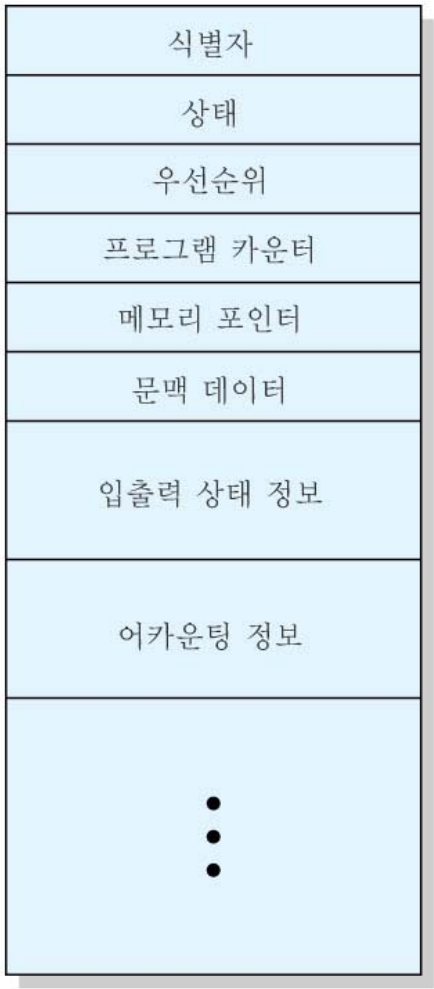
- 프로세스 이미지

표 3.4 프로세스 이미지의 일반적인 요소

사용자 데이터
사용자 공간에서 수정 가능한 부분, 프로그램 데이터와 사용자 스택 영역, 수정될 수 있는 프로그램이 포함될 수 있다.
사용자 프로그램
수행 될 프로그램
시스템 스택
각 프로세스는 하나 이상의 시스템 스택을 가진다. 스택은 프로시저와 시스템 호출에 필요한 매개변수와 호출 주소(복귀 주소)를 저장하는 데 사용된다.
프로세스 제어블록
프로세스를 제어하기 위해 운영체제가 필요로 하는 데이터(표 3.5참조).

프로세스 기술: 프로세스 제어 구조

- PCB
  - 프로세스 제어 블록
    (Process Control Block)
  - OS에 의해 관리
  - 다수의 프로세스를 지원하고 다중처리를 제공할 수 있게 지원하는 주요 자료구조



프로세스 기술: 프로세스 제어 구조

- PCB (계속)
  - 프로세스 식별: 숫자로 된 식별자
    - PID, PPID, UID, …
  - 처리기 상태 정보
    - User-Visible Registers :사용자가 이용 가능
    - 제어 및 상태 레지스터
      - *Program counter, Condition codes, Status information*
    - 스택 포인터
  - 프로세스 제어 정보
    - 스케줄링 및 상태 정보
      - *Process state, Priority, Scheduling-related information, Event*
    - 다른 프로세스들 간의 포인터 정보
    - 프로세스간 통신
    - 프로세스 권한
    - 메모리 관리
    - 자원 소유권 및 이용률

프로세스 기술: 프로세스 제어 구조

- PCB의 전형적인 구성요소

표 3.5 프로세스 제어블록의 일반적인 구성 요소

프로세스 식별
식별자 프로세스 제어블록에는 숫자로 된 다음과 같은 식별자가 있다. • 이 프로세스의 식별자 • 이 프로세스를 생성한 프로세스(부모 프로세스)의 식별자 • 사용자 식별자
처리기 상태 정보
사용자가 사용 가능한 레지스터 이것은 처리기가 사용자 모드에서 수행하는 기계 언어에 의해 참조될 수 있는 레지스터이다. 어떤 RISC 구현에서는 100개 이상 있을 수 있지만, 대개 8개에서 32개의 레지스터가 있다.
제어 레지스터 및 상태 레지스터 다음과 같이 처리기의 동작을 제어하기 위해 사용되는 다양한 처리기 레지스터들이 있다ㅁ• 프로그램 카운터 : 다음에 반입(fetch)할 명령어의 주소를 가짐 • **조건 코드(condition codes):** 가장 최근에 수행된 산술 또는 논리 연산의 결과(즉, 부호, 0, 자리올림, 같음, 오버플로) • **상태 정보:** 인터럽트 가능/불가 플래그들과 수행 모드를 가짐
스택 포인터 각 프로세스는 하나 이상의 후입선출(LIFO) 시스템 스택을 가진다. 스택은 프로시저와 시스템 호출의 매개변수와 호출 주소를 저장하는데 사용된다. 스택 포인터는 스택의 정상(top)을 가리킨다.

프로세스 기술: 프로세스 제어 구조

- PCB의 전형적인 구성요소 (계속)

프로세스 제어 정보

스케줄링과 상태 정보

운영체제가 스케줄링 기능을 수행하기 위해 필요한 정보들이다. 전형적인 항목들은 다음과 같다.

- **프로세스 상태:** 수행되기 위해 스케줄될 프로세스의 준비 상황을 정의한다(예로, 수행, 준비, 대기, 중단).
- **우선순위:** 프로세스의 스케줄링 우선순위를 나타내기 위해서 하나 이상의 필드가 사용될 수 있다. 어떤 시스템에서는 여러 개의 값이 요구된다(즉, 기본값, 현재값, 허용가능 최대치).
- **스케줄링과 관련된 정보:** 이 값은 사용되는 스케줄링 알고리즘에 따라 다르다. 프로세스가 대기하고 있었던 시간, 마지막에 수행되었던 시간 등이 그 예가 될 수 있다.
- **이벤트:** 프로세스가 재 시작될 수 있기 전에 기다리고 있는 사건을 알려준다.

자료구조화(Data Structuring)

프로세스는 큐나 링, 또는 다른 자료구조로 다른 프로세스와 연결될 수 있다. 예를 들어, 특정 우선순위를 가지고 대기 상태에 있는 모든 프로세스들은 하나의 큐에 연결될 수 있다. 프로세스는 다른 프로세스와 부모-자식(생성한 자와 생성된 자) 관계가 될 수도 있다. 그러한 구조를 지원하기 위해 프로세스 제어블록에는 다른 프로세스들을 가리키는 포인터를 포함할 수 있다.

프로세스간 통신(IPC)

여러 플래그와 시그널, 메시지 등이 두 개의 독립된 프로세스 사이의 통신과 연관될 수가 있다. 이 정보의 전부 또는 일부가 프로세스 제어블록 내에 유지된다.

프로세스 권한(Process Privileges)

프로세스는 접근할 수 있는 메모리 조건 및 수행할 수 있는 명령어 유형 등에 대해 어떤 권리를 가진다. 덧붙여, 이 권한은 시스템 유틸리티와 서비스를 사용하는 경우에도 적용될 수 있다.

메모리 관리

이 부분은 이 프로세스에 할당된 가상메모리를 나타내는 세그먼트나 페이지 테이블로의 포인터를 가질 수 있다.

자원의 소유권과 이용률

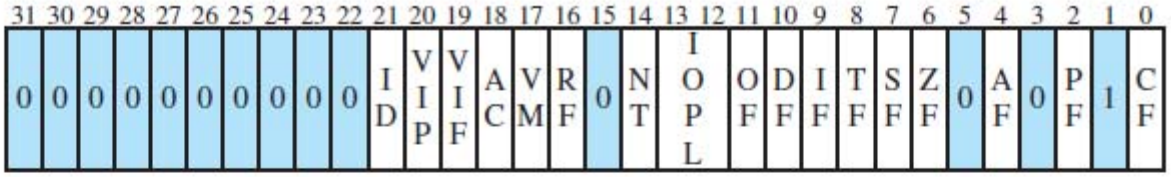
프로세스에 의해 제어되는 자원들은 개방된(opened) 파일처럼 표시될 수 있다. 처리기 및 자원들의 이용률에 대한 이력 정보도 포함될 수 있는데, 이 정보는 스케줄러에 의해 사용된다.

자원의 소유권과 이용률

프로세스에 의해 제어되는 자원들은 개방된opened) 파일처럼 표시될 수 있다. 처리기 및 자원들의 이용률에 대한 이력 정보도 포함될 수 있는데, 이 정보는 스케줄러에 의해 사용된다.

프로세스 기술: 프로세스 제어 구조

- 프로그램 상태 워드 (PSW)
  - 예: x86 상에서 EFLAGS 레지스터



X ID = 식별 플래그
X VIP = 가상 인터럽트 대기(pending)
X VIF = 가상 인터럽트 플래그
X AC = 정렬 검사(Alignment check)
X VM = 가상 8086 모드
X RF = 재개 플래그
X NT = 중첩 태스크 플래그
X IOPL = 입출력 특권 레벨
S OF = 오버플로우 플래그

C DF = 방향 플래그
X IF = 인터럽트 가능 플래그
X TF = 트랩 플래그
S SF = 부호 플래그
S ZF = 제로 플래그
S AF = 보조 자리올림 플래그
S PF = 패리티 플래그
S CF = 자리올림 플래그

S는 상태 (Status) 플래그를 나타냄
C는 제어 (Control) 플래그를 나타냄
X는 제어 (System) 플래그를 나타냄
음영으로 표시된 비트들은 나중을 위해 예약되어 있음

그림 3.12 x86 EFLAGS 레지스터

프로세스 기술: 프로세스 제어 구조

표3.6 x86 EFLAGS 레지스터 비트

상태 플래그들 (조건 코드들)
AF(보조 캐리 플래그) AL 레지스터를 사용하는 8비트 산술 또는 논리 연산의 반쪽 바이트(half-bytes) 간의 캐리와 빌림(borrow)을 나타냄.
CF(캐리 플래그) 산술 연산 다음에 맨 왼쪽 비트 위치로 캐리나 빌림이 있다는 것을 알려준다. 또한 몇몇 쉬프트나 회전(rotate) 연산으로 수정되기도 한다.
OF(오버플로우 플래그) 덧셈이나 뺄셈 후에 산술 오버플로가 발생했음을 알림
PF(패리티 플래그) 산술 또는 논리 연산 결과의 패리티로 1은 짝수 패리티, 0은 홀수 패리티를 표시
SF(부호 플래그) 산술 또는 논리 연산 결과의 부호를 나타낸다.
ZF(제로 플래그) 산술 또는 논리 연산 결과가 0인지 알려준다.
제어 플래그
DF(방향 플래그) 문자열 처리 명령어가 16비트 레지스터인 SI와 DI(16비트 연산)를, 또는 32비트 레지스터인 ESI와 EDI(32비트 연산)를 증가시킬지 또는 감소시킬지를 결정한다.

프로세스 기술: 프로세스 제어 구조

시스템 플래그 (응용 프로그램에 의해 수정되지 않아야 함)

AC(정렬 검사, Alignment check)
워드(word)나 이중워드(doubleword)의 주소가 비워드(nonword)나 비이중워드(non-doubleword)에 지정되면 비트가 설정된다.

ID(식별 플래그)
이 비트가 설정과 해제될 수 있다면, 이 처리기는 CPUID 명령어를 지원한다. 이 명령어는 벤더와 제품군, 모델에 대한 정보를 제공한다.

RF(재개 플래그, Resume flag)
프로그래머로 하여금 디버그 예외상황을 금지(disable)시킬 수 있게 하여, 바로 또 다른 디버그 예외상황이 발생되지 않게 하면서 디버그 예외상황 처리 후, 명령어가 다시 시작할 수 있도록 하여 준다.

IOPL(입출력 특권 레벨, I/O privilege level)
이 비트가 설정될 경우, 처리기가 보호 모드에서 동작되고 있는 동안 모든 입출력 장치 접근에 대해 예외상황이 발생되게 한다.

IF(인터럽트 가능 플래그)
이 비트가 설정되어 있으면, 처리기는 외부 인터럽트를 인식할 수 있다.

TF(트랩 플래그)
이 비트가 설정되어 있으면, 각각의 명령어를 수행한 후에 인터럽트를 발생시킨다. 이것은 디버깅을 하는데 사용된다.

NT(중첩 태스크 플래그, nested task flag)
보호 모드에서 동작되고 있는 다른 태스크에 현재 태스크가 포함되어 있음을 알림.

VM(가상 8086 모드)
프로그래머로 하여금 가상 8086 모드를 허용하거나 금지할 수 있게 하여, 처리기가 8086 컴퓨터로서 동작하는지를 결정한다.

VIP(가상 인터럽트 대기, Virtual interrupt pending)
가상 8086 모드에서 사용되며, 하나 이상의 인터럽트가 서비스를 기다리고 있다는 것을 나타낸다.

VIF(가상 인터럽트 플래그)
가상 8086 모드에서 IF 대신에 사용

프로세스 기술: 프로세스 제어 구조

- 가상 메모리에서 사용자 프로세스 (그림 3.13)

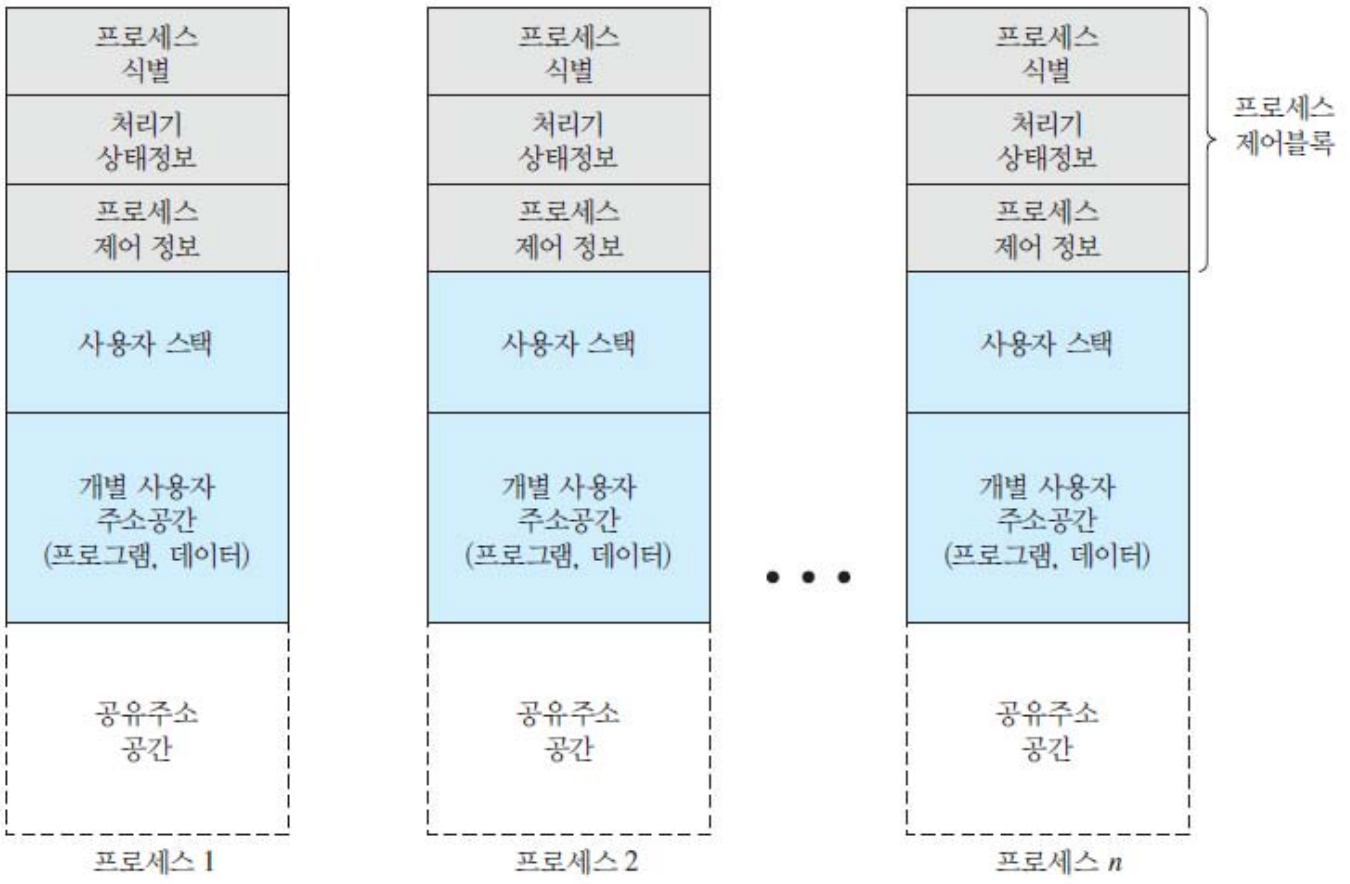
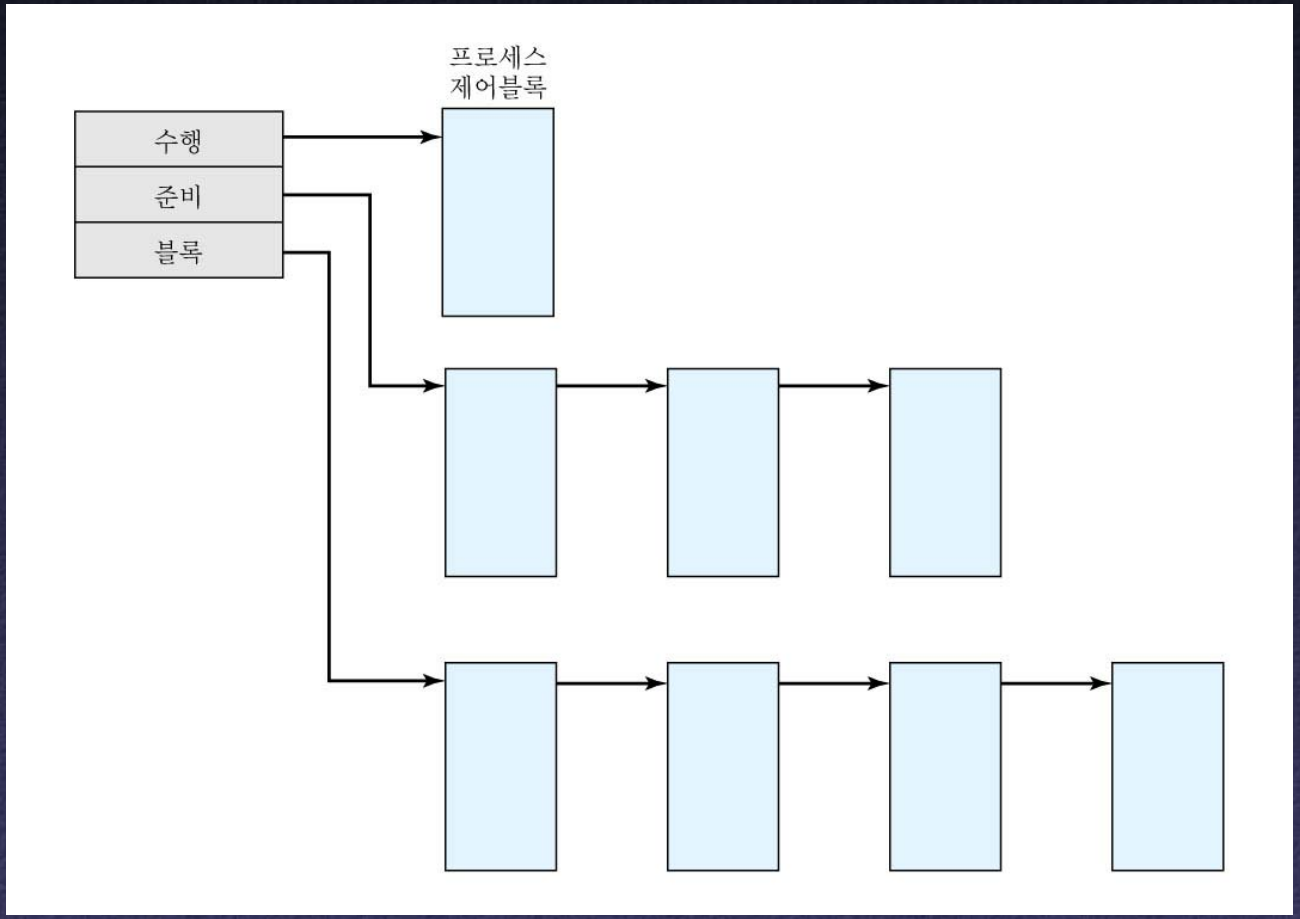


그림 3.13 가상메모리에서의 사용자 프로세스들

프로세스 기술: 프로세스 제어 구조

- 프로세스 리스트 구조



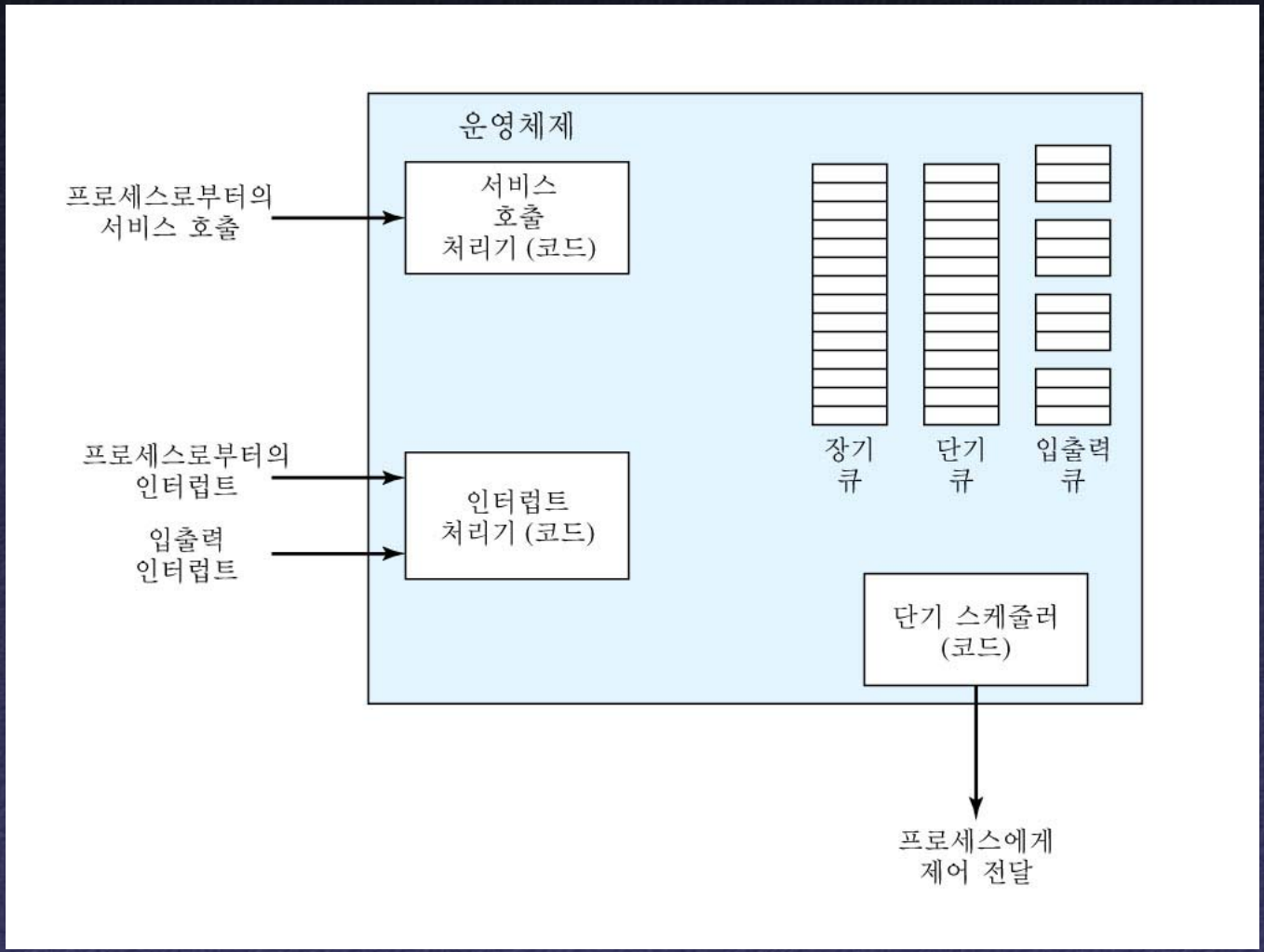
프로세스 제어

- 모드 전환(mode switching) ≡ 모드 전이
- 처리기 실행 모드
  - 대부분의 처리기는 최소한 두 가지의 수행 모드를 제공
  - 사용자 모드(user mode)
  - 시스템 모드(system mode, control mode, kernel mode)

- 프로세스 교환 (Process switching)
  - 선점(preemption) 및 디스패치(dispatch)
  - 문맥 교환(context switch)

프로세스 제어: 모드 전환(Mode Switch)

- 수행 모드가 전환(switch)될 때는?



프로세스 제어: 프로세스 교환

- 프로세스 교환을 유발하는 사건들
  - Clock interrupt: 최대 허용된 시간단위(time slice)가 지나면 발생
    - 수행 → 준비
  - I/O interrupt
    - 블록 → 준비
  - 메모리 폴트 (페이지 부재)
    - 수행 → 블록
  - 트랩 (Trap)
    - 수행 중에 발생한 오류 및 예외상황(exception)
    - 해당 프로세스가 종료될 수도 있음 ← 복구 불가능한 상황
  - 수퍼바이저 호출 (Supervisor call)
    - file open 등
    - 이 때 모드 전환 발생, 필요하면 프로세스 교환

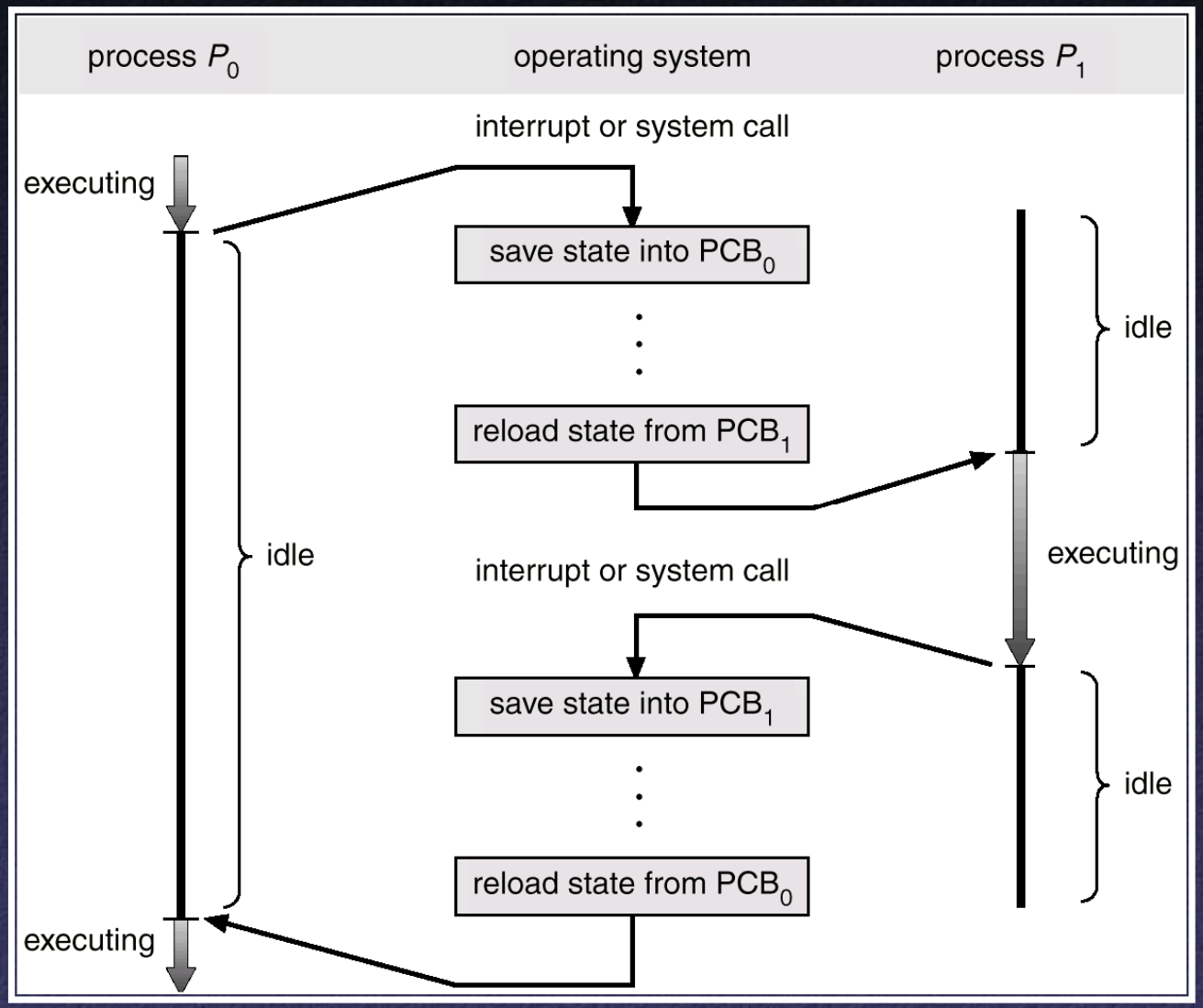
프로세스 제어: 프로세스 교환

- 프로세스 상태 변경(change)
  1. 프로그램 카운터 및 다른 레지스터들을 포함하여 처리기의 문맥을 저장
  2. 현재 수행 상태에 있는 프로세스(예로 P_A)의 PCB를 갱신
  3. 그 PCB를 준비큐, 블록큐, 또는 준비/보류큐 중의 하나에 삽입
  4. 실행할 다른 프로세스(예로 P_B)를 선택
  5. 새로 선택된 프로세스(P_B)의 PCB를 갱신
  6. 메모리 관리 자료구조를 갱신
  7. 선택된 프로세스(P_B)의 문맥을 복원(restore)

  ☞ context switch (문맥 교환)

  ☞ mode switch may occur without changing the state of the current running process

프로세스 제어: 프로세스 교환



프로세스 제어: 프로세스 생성

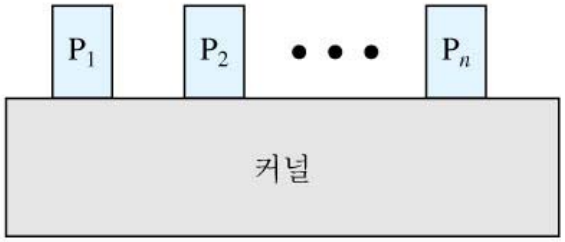
- 유일한 프로세스 식별자 할당
- 프로세스의 주소공간 할당
  - 사용자 주소공간 및 사용자 스택
  - 공유 영역 (shared area)
- PCB 초기화
  - PC, SP, 초기 상태, 우선순위, ...
- 적절한 연결 설정
  - 새로운 프로세스를 준비큐 또는 준비/보류큐에 삽입
- 다른 자료구조를 생성하거나 확장
  - OS may maintain an accounting file

운영체제의 실행(수행)

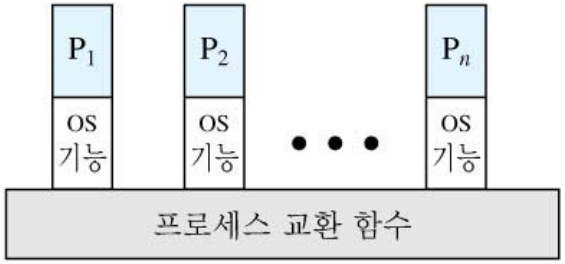
- OS 실행(execution)
  - 비 프로세스 커널 (Non-process Kernel, 분리된 커널)
    - OS 코드가 특권 모드(privileged mode)에서 분리된 개체(separate entity)로 실행
  - 사용자 프로세스 내에서 실행
    - 사용자 프로세스 문맥 내에 OS 소프트웨어가 유지
    - 프로세스가 OS 코드를 실행 중일 때는 특권 모드를 가짐
  - 프로세스 기반 OS (Process-Based OS)
    - 시스템 프로세스들의 집합으로 OS를 구현
    - 다중 처리기(multi-processor) 또는 다중 컴퓨터 환경에 유용

운영체제의 실행

- OS와 사용자 프로세스와의 관계 (그림 3.15)

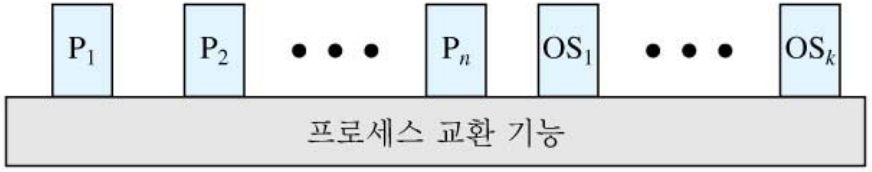


(a) 분리된 커널



(b) 운영체제 기능이 사용자 프로세스 내에서 수행

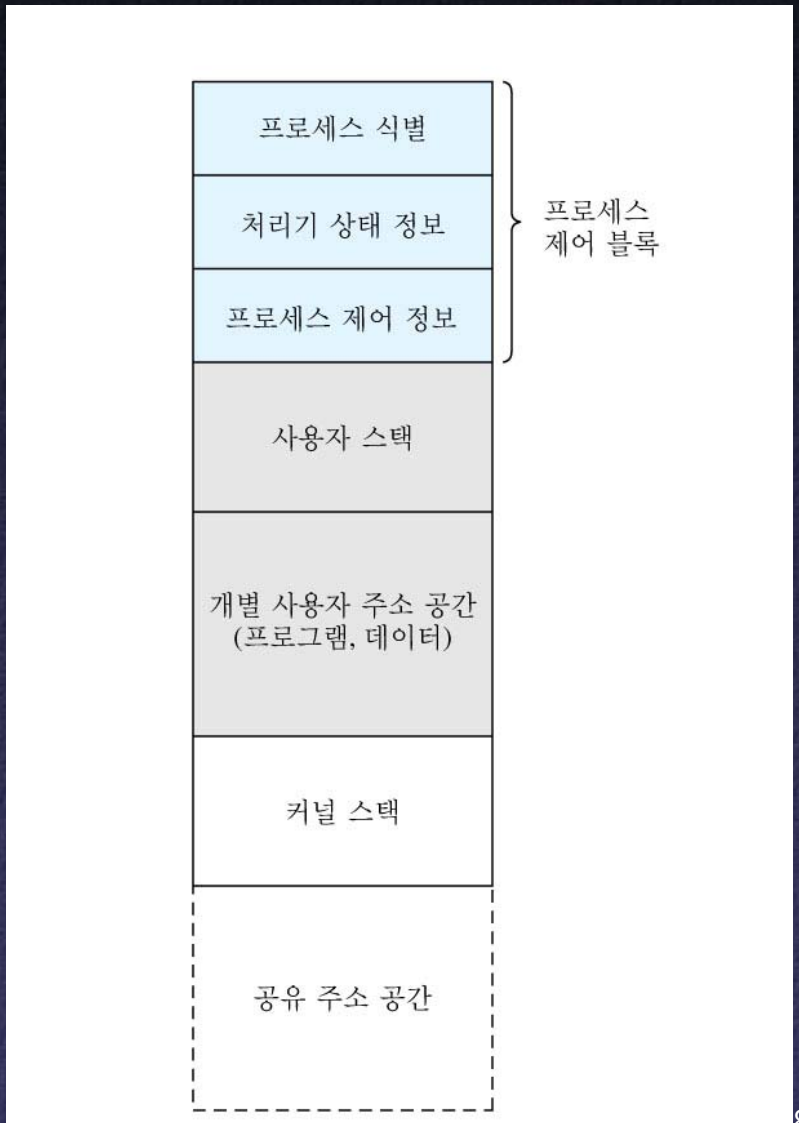
* 기능 = function = 함수



(c) 운영체제 기능이 분리된 프로세스로 수행

운영체제의 실행

- 사용자 프로세스 내에서 OS 실행
  - 프로세스 이미지 (그림 3.16)



UNIX SVR4의 프로세스 관리

- 모델: OS가 사용자 프로세스 환경 내에서 실행
  - 9개의 프로세스 상태를 가짐

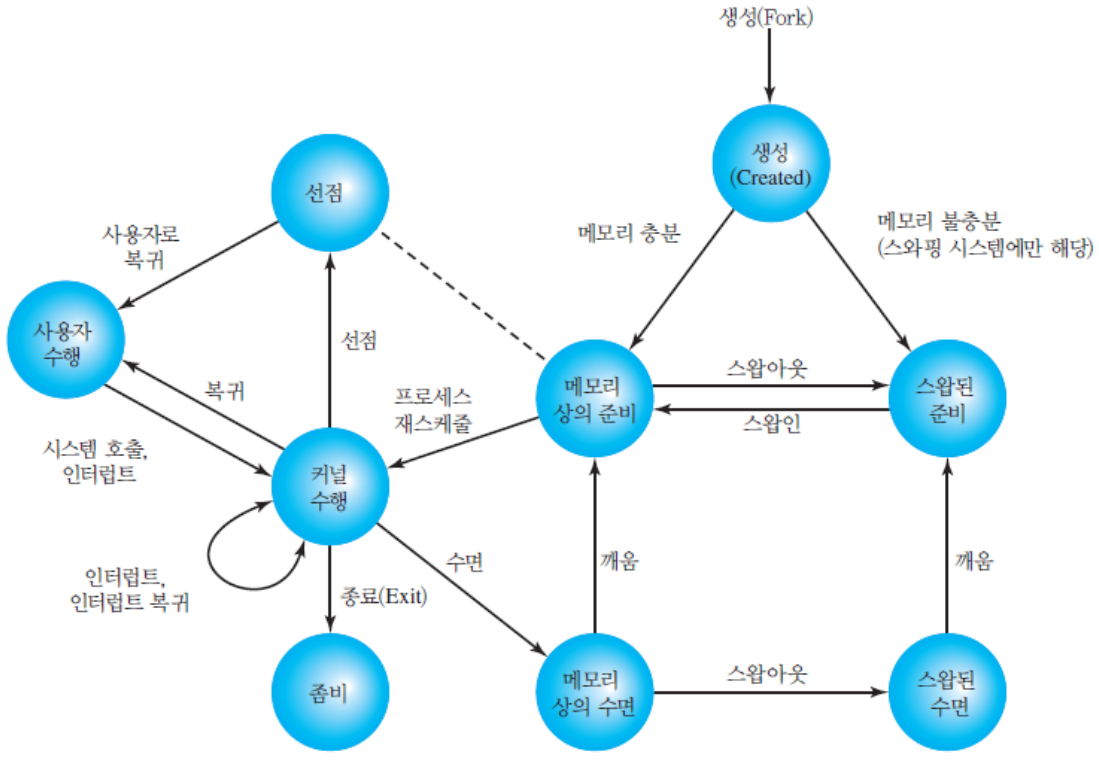


표 3.17 UNIX의 프로세스 상태전이도

- pid 0: swapper
- pid 1: init

UNIX SVR4의 프로세스 관리

- 프로세스 기술 (Process Description)

표 3.10 UNIX 프로세스 이미지

사용자 수준 문맥	
프로세스 텍스트	프로그램에서 수행 가능한 기계 명령어
프로세스 데이터	이 프로세스의 프로그램이 접근할 수 있는 데이터
사용자 스택	사용자 모드에서 수행되는 함수의 인자와 지역 변수, 포인터를 보유
공유 메모리	다른 프로세스와 공유하는 메모리로 프로세스 간 통신에 사용
레지스터 문맥	
프로그램 카운터	다음에 수행될 명령어의 주소로 해당 프로세스의 커널 또는 사용자 메모리 공간 내의 주소
처리기 상태 레지스터	선점될 때의 하드웨어 상태를 보유함. 내용과 형식은 하드웨어에 따라 다름
스택 포인터	특정 시점의 연산모드에 따라, 커널이나 사용자 스택의 정상을 가리킴
범용 레지스터	하드웨어에 따라 다름
시스템 수준 문맥	
프로세스 테이블 항목	프로세스 상태를 정의하며 운영체제는 항상 이 정보에 접근 가능
사용자 영역(U area)	프로세스 문맥에서만 접근할 필요가 있는 프로세스 제어 정보
프로세스 당 영역 테이블 (Per Process Region Table)	가상주소를 물리주소로 사상하는 방식을 정의한다. 또한 읽기 전용, 읽기와 쓰기, 읽기와 수행 등 프로세스에게 허용된 접근 종류를 나타내는 허가 필드도 포함하고 있음
커널 스택	프로세스가 커널 모드에서 수행될 때, 커널 프로시저의 스택 프레임(frame)을 포함

UNIX SVR4의 프로세스 관리

- 프로세스 테이블

표 3.11 UNIX 프로세스 테이블 항목

프로세스 상태	프로세스의 현재 상태
포인터	U 영역과 프로세스 메모리 영역(텍스트, 데이터, 스택)을 가리킴
프로세스 크기	프로세스에게 할당된 공간의 크기를 운영체제가 알 수 있도록 함
사용자 식별자	**실제 사용자** ID(real user ID)는 현재 수행중인 프로세스를 책임지고 있는 사용자를 식별한다. **유효 사용자** ID(effective user ID)는 프로세스가 특정 프로그램과 관련된 임시 권한을 얻기 위해 사용될 수도 있다. 그 프로그램이 프로세스의 일부로서 수행되는 동안 그 프로세스는 해당 프로그램의 유효 사용자 ID를 가지고 동작한다.
프로세스 식별자	이 프로세스의 ID와 부모 프로세스의 ID. 이 값들은 fork 시스템 호출 수행 중, 프로세스가 생성(Created) 상태로 진입할 때 설정된다.
사건 디스크립터	프로세스가 수면상태에 있을 때 유효하다. 사건(event)이 발생되면 프로세스는 준비 상태로 전이된다.
우선순위	프로세스 스케줄링을 위해 사용된다.
신호(signal, 시그널)	프로세스에게 보내졌으나 아직 처리되지 않은 신호들을 열거한다.
타이머	프로세스 수행 시간과 커널의 자원 이용률, 프로세스에게 알람 신호(alarm signal)를 보내기 위해 사용되는 사용자 설정 타이머가 포함된다.
P_link	준비 큐에서 다음에 연결된 프로세스를 가리키는 포인터(프로세스가 수행될 준비가 되어있을 경우에만 유효하다)
메모리 상태	프로세스 이미지가 주기억장치에 있는지 스왑아웃되어 보조 메모리에 있는지를 알려준다. 주기억장치에 있다면, 이 필드는 또한 프로세스 이미지가 스왑아웃될 수 있는지 또는 주기억장치에서 일시적으로 락(lock)이 걸려 있는지도 알려준다.

UNIX SVR4의 프로세스 관리

- U 영역(U area)

표 3.12 UNIX U 영역

프로세스 테이블 포인터	U 영역에 대응되는 항목을 가리킴.
사용자 식별자	실제 및 유효 사용자 ID. 사용자 권한을 결정하기 위해 사용됨.
타이머	프로세스(자손 프로세스들도 포함)가 사용자 모드와 커널 모드에서 수행된 시간을 기록함.
신호 핸들러 배열	시스템에서 정의된 각 시그널 유형에 대해, 프로세스가 해당 신호를 받았을 때 어떻게 동작할지를 알려줌(종료, 무시, 지정한 사용자 함수 수행).
제어 터미널	제어 터미널이 있다면, 이 프로세스에 대한 로그인 터미널을 지정함.
오류 필드	시스템 호출 수행 중에 생기는 오류를 기록함.
반환 값	시스템 호출의 결과 값을 저장함.
입출력 매개변수	전송될 데이터의 양, 사용자 공간에 있는 발생지(또는 목적지) 데이터 배열의 주소, 입출력에 대한 파일 오프셋 등을 기술함.
파일 매개변수	현재 디렉토리와 현재 루트는 그 프로세스의 파일 시스템 환경을 기술함.
사용자 파일 디스크립터 테이블	프로세스가 개방한 파일들을 기록한다.
한계 필드	프로세스의 크기와 프로세스가 쓸 수 있는 파일 크기를 제한함.
허가 모드 필드	프로세스가 생성한 파일에 대해 모드 설정을 마스크(mask)함.

UNIX SVR4의 프로세스 관리

- 프로세스 제어
  - fork()
    - 프로세스 테이블에 한 슬롯(slot, 엔트리)를 할당
    - 유일한 프로세스 ID (PID)를 할당
    - 부모의 프로세스 이미지를 복사
    - 부모가 소유하고 있는 모든 파일들의 참조계수(count)를 증가
    - 자식 프로세스를 준비 상태로 설정
    - 부모 프로세스에는 자식의 PID를, 자식 프로세스에는 0을 반환
  - 디스패처(Dispatcher)는 다음 중 하나를 수행
    - 제어를 부모 프로세스가 유지
    - 자식 프로세스에 제어를 넘김
    - 다른 프로세스에게 제어를 넘김