프로세스

프로세스

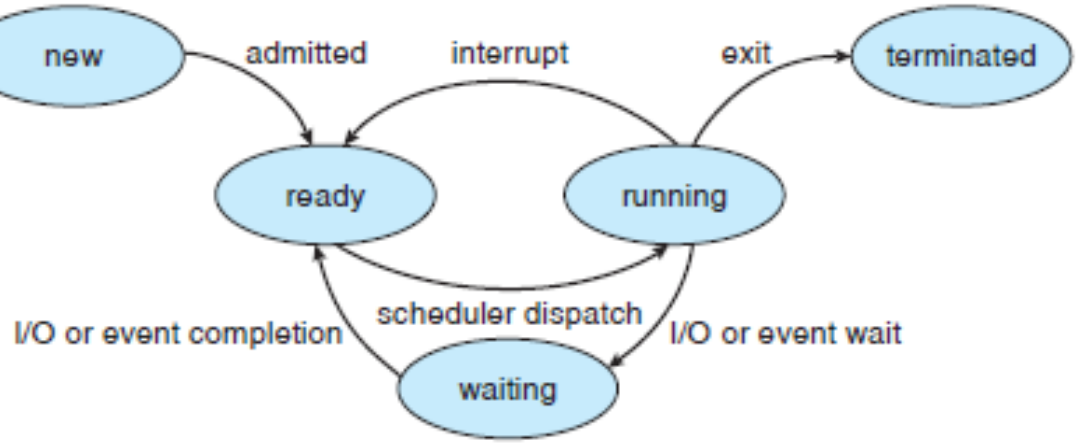
프로세스

- 프로세스는 실행중인 프로그램
- 프로그램은 명령어들의 집합이 저장된 것
- 프로세스는 다음으로 pc가 실행하기위한 명령어와 연관된 리소스를 가리키고 있는 active 한 개체
- 프로세스는 메모리에 실행가능한 프로그램이 로드되어 있는 것

- 프로세스에 포함되는 것
  1. Text section
  2. Pc(프로그램 카운터)
  3. 스택
  4. Data section(변수 저장)
  5. Heap

프로세스

- 프로세스의 상태
  1. New(생성)
  2. Running(동작)
  3. Waiting(I/O 같은 이벤트를 기다리는 중)
  4. Ready(작업대기)
  5. Terminated(종료)



- 오직 한 프로세스만 하나의 코어에서 실행가능, 나머지는 대부분 ready 또는 waiting 상태

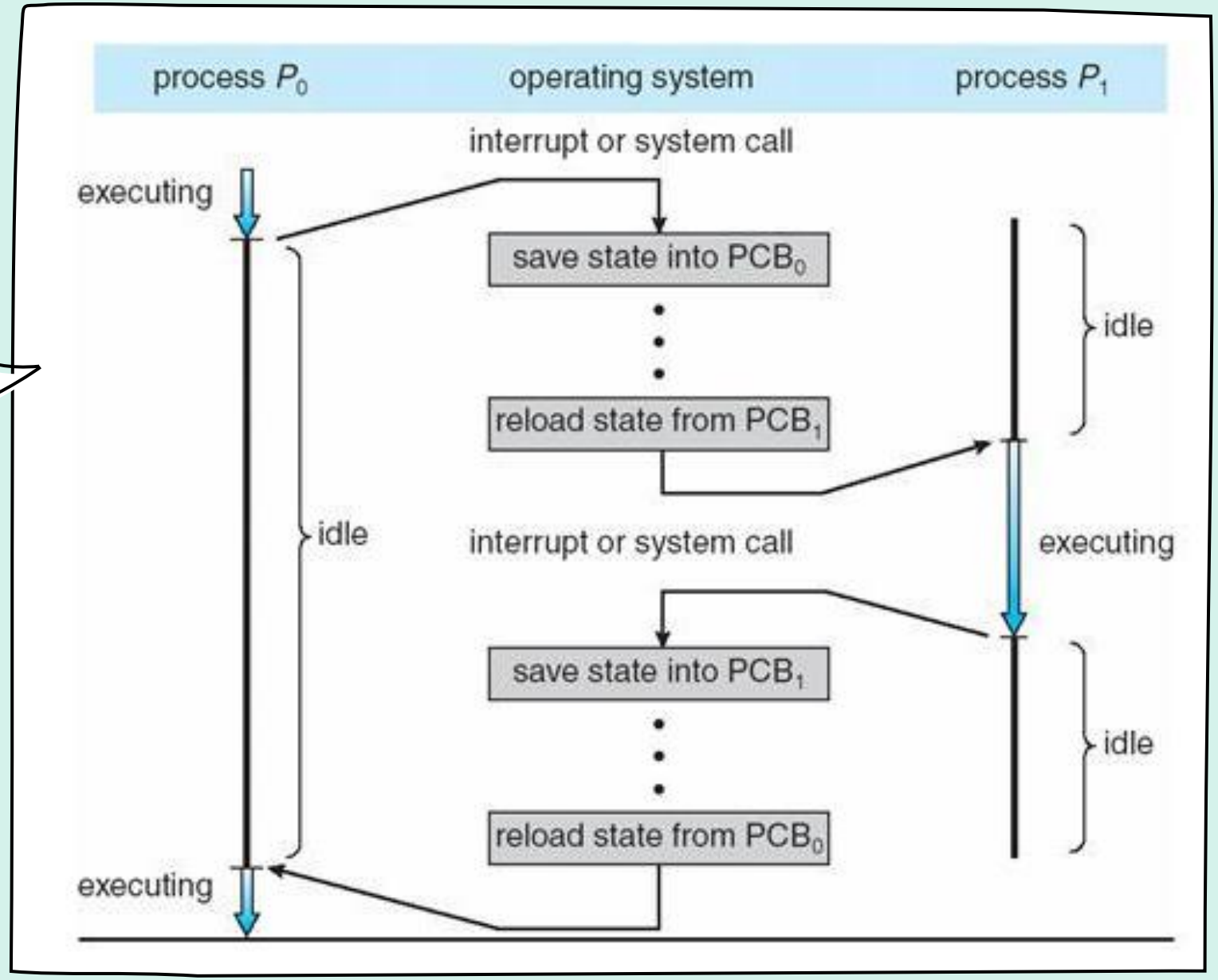
Process Control Block(PCB)

PCB

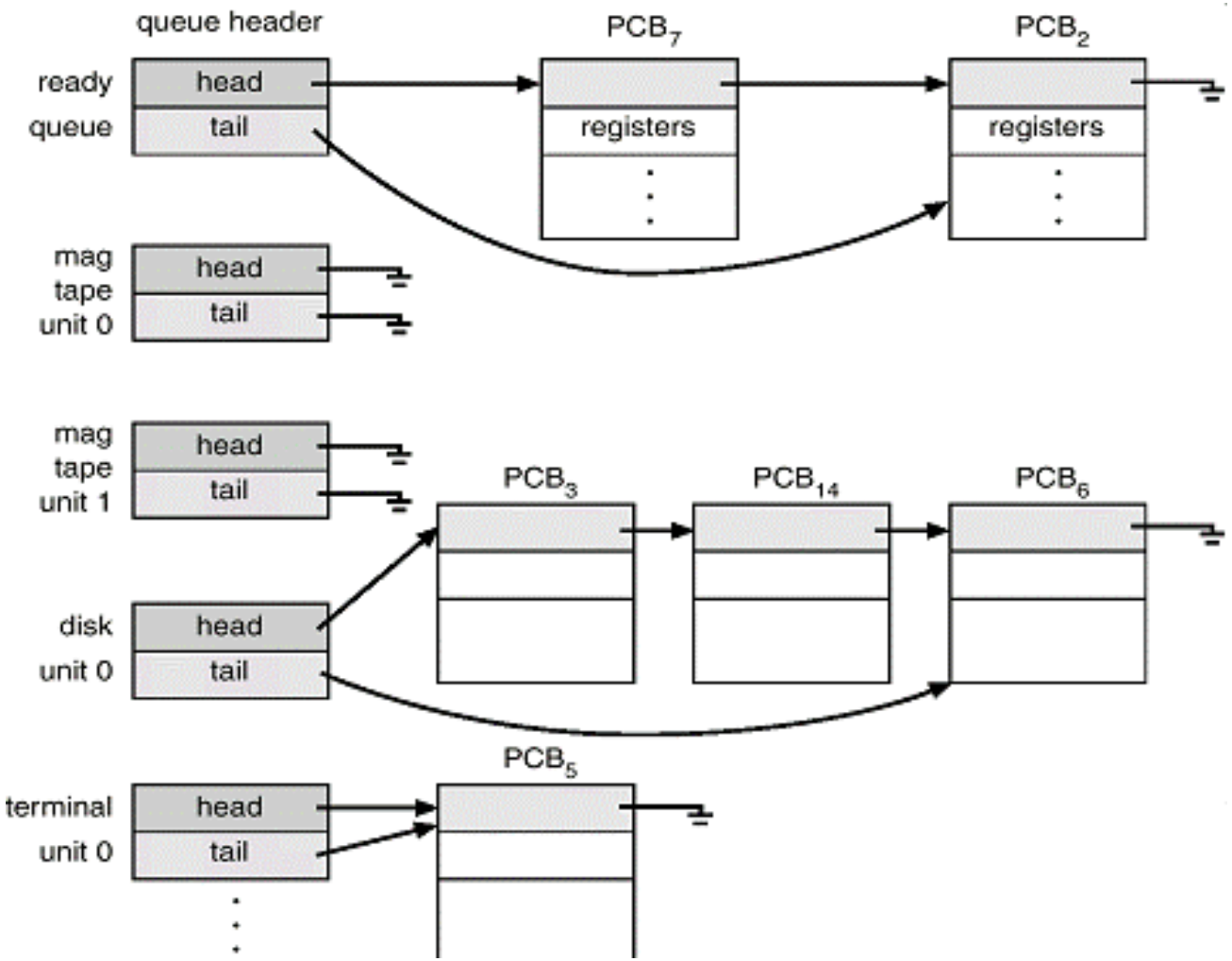
- 각 프로세스들은 운영체제에서 PCB로 표현됨
- PCB의 구성요소
  1. Program State(new, ready, running, waiting, halted 중 하나)
  2. Program Counter(다음 명령의 주소를 가리킴)
  3. CPU register(누산기, 인덱스 레지스터, 스택포인터 등등) **<<인터럽트가 일어나면 레지스터 정보들 저장>>**
  4. CPU-scheduling information(스케줄링에 필요한 파라미터 등을 저장)
  5. Memory-management information(메모리에 대한 정보 저장)
  6. Accounting information(cpu 사용량 등등)
  7. I/O status information

PCB

인터럽트가 일어나면
레지스터 정보들 저장



프로세스 스케줄링

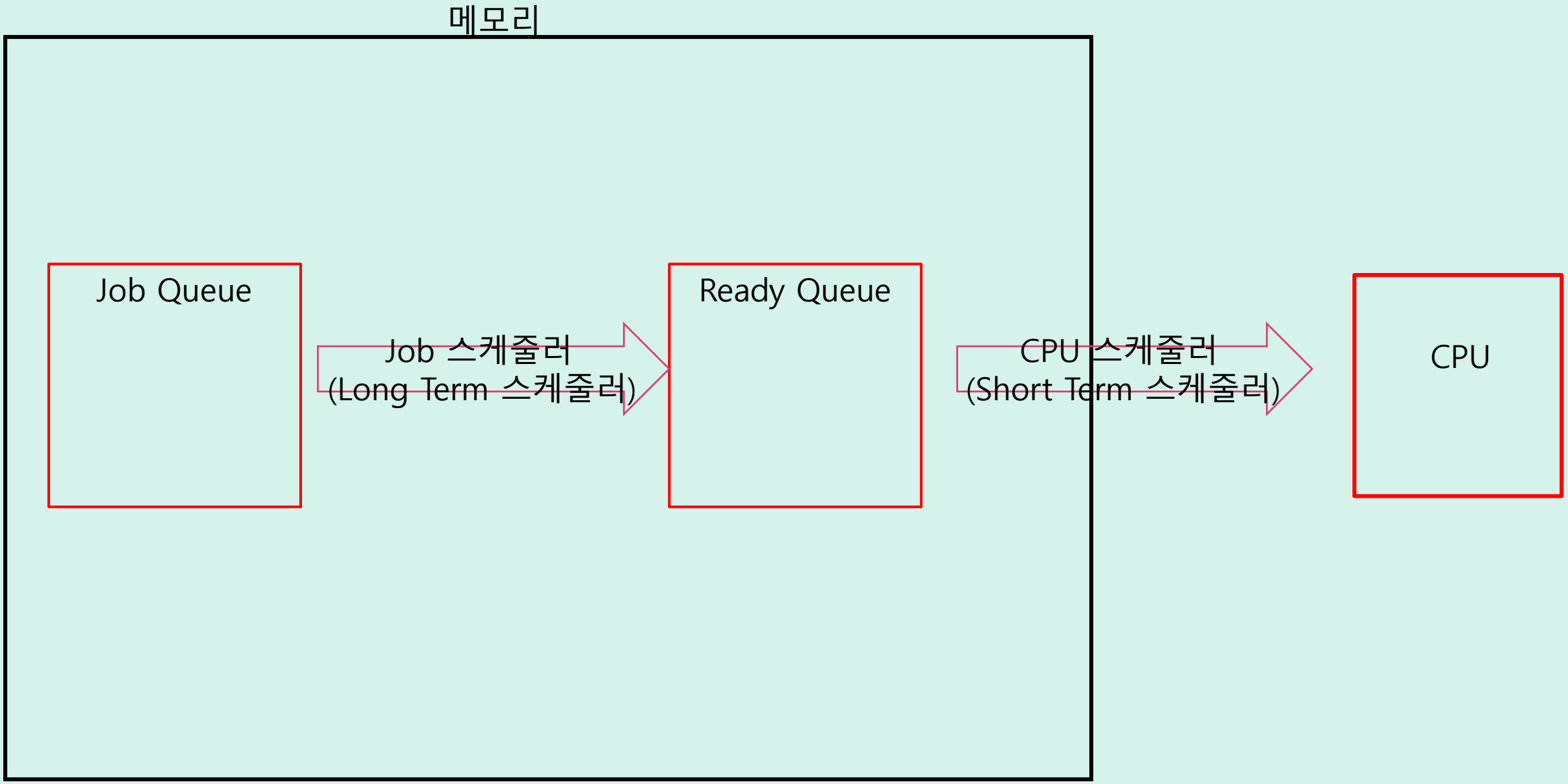


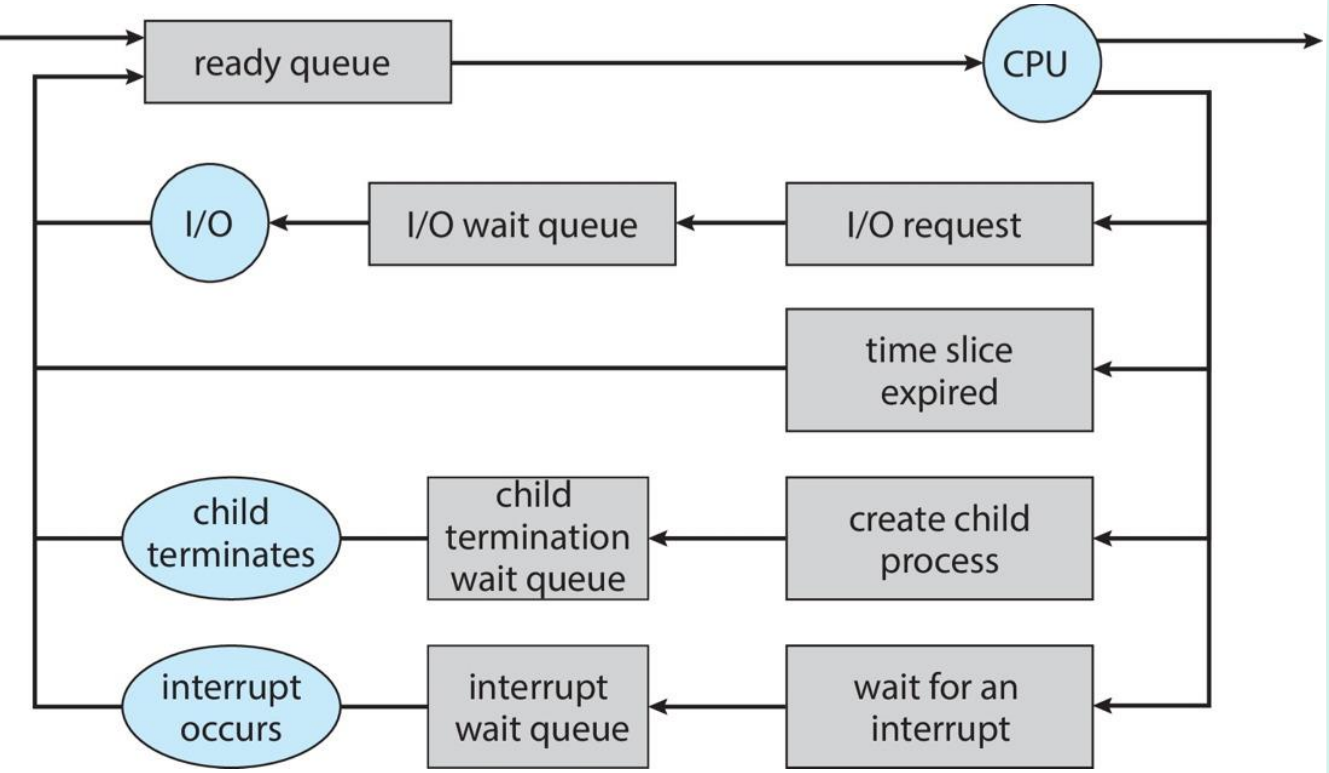
프로세스 스케줄링

- Cpu는 실행중인 프로세스를 자주 바꿔서 프로세스가 서로 상호 동작하도록 함
- CPU가 실행중인 프로세스를 자주 바꾸도록 도움을 줌

프로세스 스케줄링

- 시스템에서 예상보다 많은 프로세스가 실행되기를 희망하는 종종 경우가 있다
- 이 프로세스들은 디스크에 spool 된 후 ready queue에 올라가기 위해 대기한다.
- **Longterm scheduler** : 누구를 ready queue(메모리)에 올릴 것 인지 결정한다. I/O 바운드 프로세스와 CPU바운드 프로세스를 잘 믹스한다.
- **Shortterm scheduler** : ready queue에 있는 프로세스 중, 다음에 실행될 프로세스를 결정
- 둘의 차이는 실행 빈도, **shortterm scheduler가 더 자주 실행**된다.


메모리
Job Queue
Job 스케줄러
(Long Term 스케줄러)
Ready Queue
CPU 스케줄러
(Short Term 스케줄러)
CPU



스케줄링 큐

- 프로세스 상태에 따라 큐가 다름, 프로세스는 큐들을 왔다 갔다 할 수 있음
- 프로세스는 제일 처음 job queue에서 대기, job queue는 프로세스들이 실행되기 전에 기다리는 큐(메모리에 올라가기 전 대기)
- 메모리에 올라간 프로세스는 실행되기 위해 ready queue에서 대기, ready queue는 cpu 사용을 기다리는 큐
- 디바이스 큐는 디바이스 큐마다 존재, 디바이스 사용을 기다리는 큐

CPU 할당 받은 프로세스가 할 수 있는 일

1. I/O 사용 요청 후 I/O 큐에 들어감
2. 자식프로세스 생성 후 자식프로세스의 종료를 기다림
3. 인터럽트 때문에 cpu에 의해 중지됨, 다시 ready 큐로 들어감

Context Switching

Context Switching

- 인터럽트가 발생시, 현재 프로세스의 cpu레지스터 정보 등 context의 정보를 저장.
- 중단 될 프로세스의 정보를 저장하고 새로 시작될 프로세스의 정보를 복원하는 것
- 하드웨어 구현으로 context switching 시간을 줄 일 수 있음

Context Switching

- 프로세스는 exit함수를 사용해 OS한테 종료해도 가능한지 물어봄.
- 부모 프로세스는 wait 함수를 사용해 자식프로세스가 종료할 때 반환한 값을 받을 수 있음, wait 함수 호출 안 하면 자식 프로세스는 종료되지 못한 채 좀비 프로세스가 됨
- 프로세스는 종료 후 OS에 의해 자원 할당을 해제

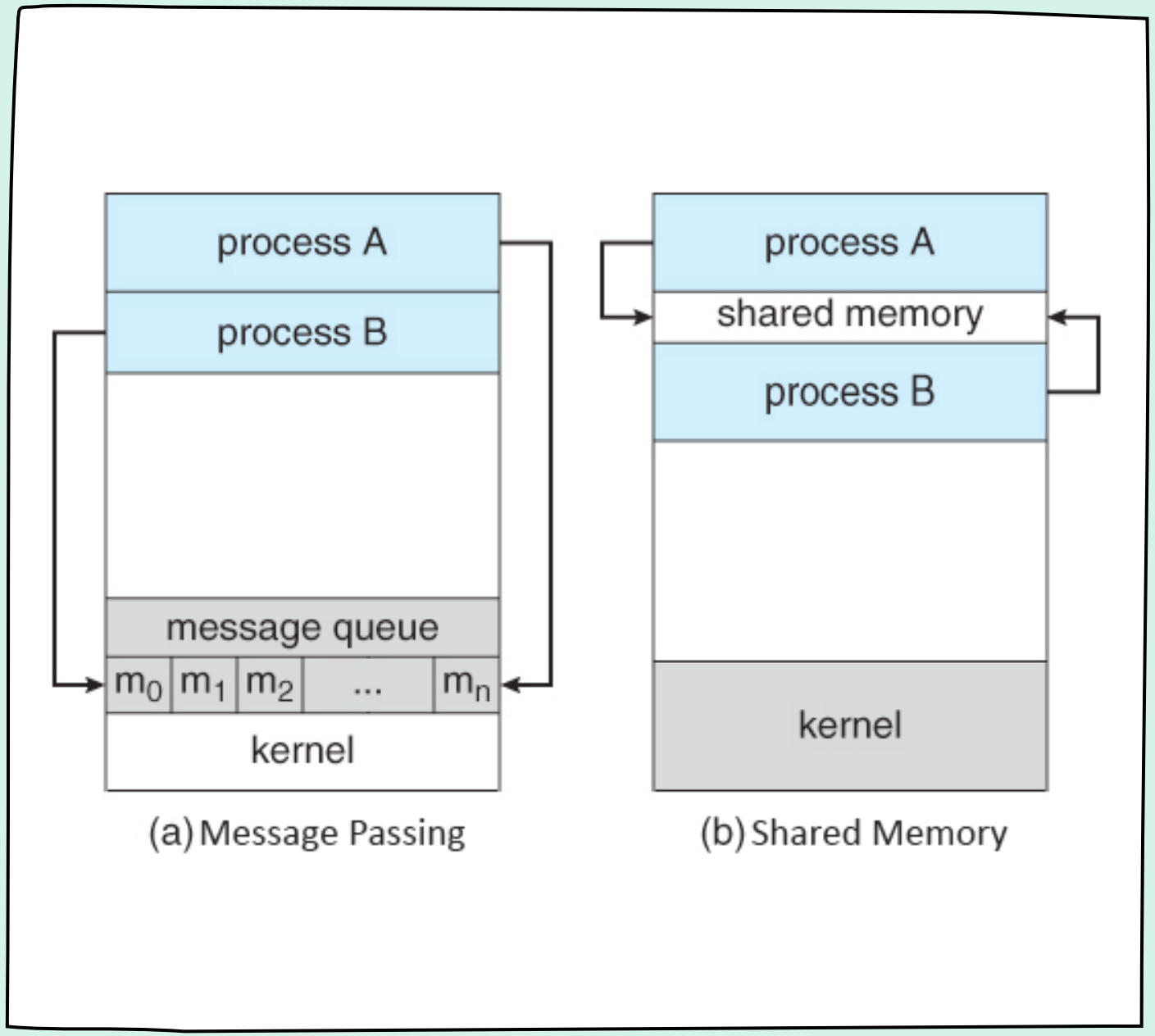
---- 부모에 의해 자식 프로세스가 종료되는 경우----

1. 자식이 자원을 초과해서 사용
2. 자식이 필요 없어짐
3. 부모가 종료, 자식도 종료

IPC(InterProcess Communication)

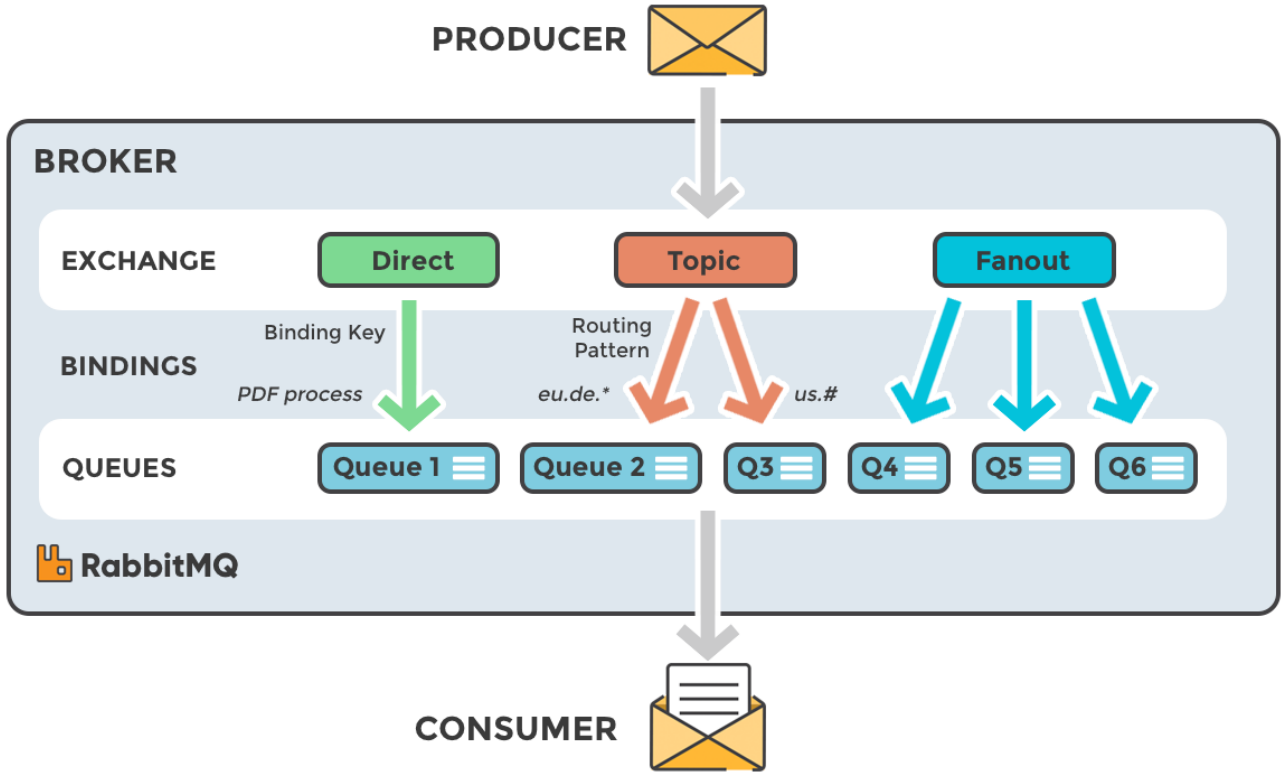
IPC

- 서로 다른 프로세스가 데이터와 정보를 공유 할 수 있게함
- IPC를 사용하는 이유
  1. 정보 공유
  2. 계산능력 향상
  3. 모듈성
  4. 편리함



공유 메모리

- 공유메모리를 통해 통신
- 임계영역 문제
- 소비자의 큐가 비어질 때 까지 생산지는 공유메모리에 데이터 못 씀



메시지 전달

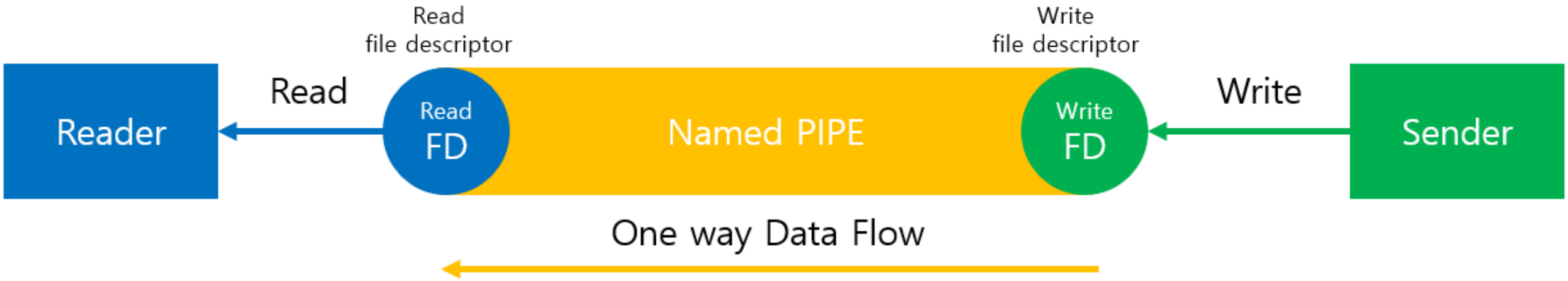
프로세스 -> 메세지 -> 운영체제 -> 메시지 -> 프로세스

소켓

- 엔드포인트로 정의된 통신
- Port 번호와 함께 ip주소로 구분됨

IPC

1. 파이프
   A. 일반적인 파이프
      - 한 프로세스가 쓰면 다른 프로세스는 읽음
      - 프로세스 종료시 파이프 소멸
      - 단방향
      - 만든 프로세스 이외에는 접근 불가능



1. 파이프
   B. 네임드 파이프
      - 부모 – 자식 관계 필요 없음
      - 여러 프로세스가 사용가능
      - 프로세스 종료 후에도 살아있음
      - 양방향