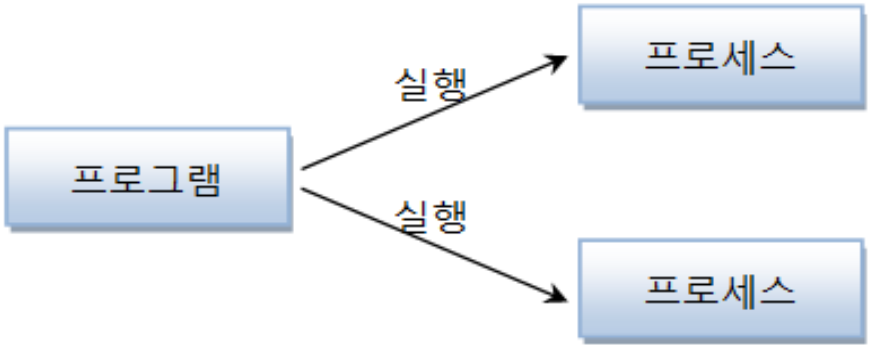
멀티 스레드

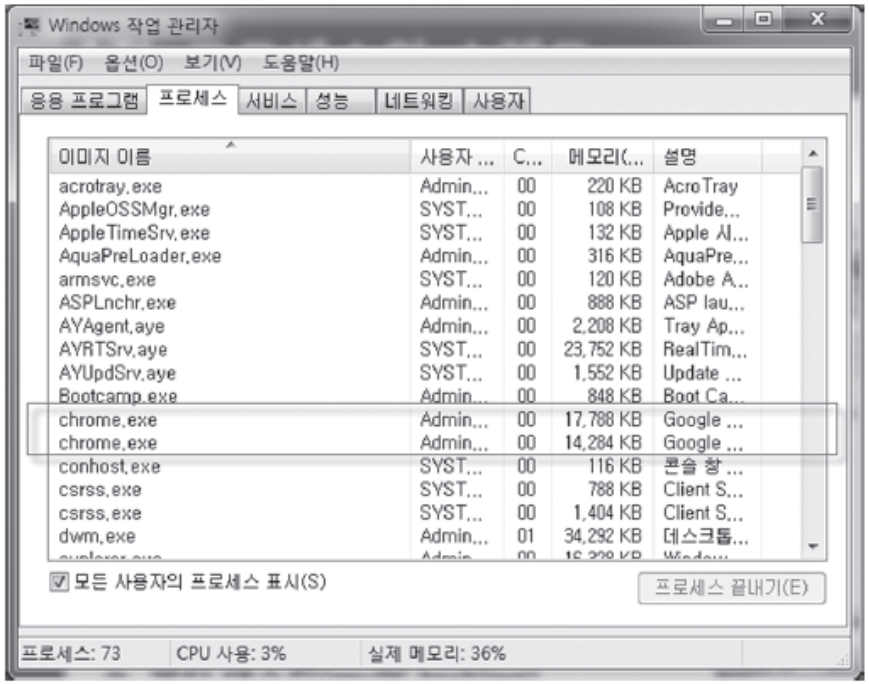
Contents

1절. 프로세스와 스레드

❖ 프로세스(process)

- 실행 중인 하나의 프로그램
- 하나의 프로그램이 다중 프로세스 만들기도

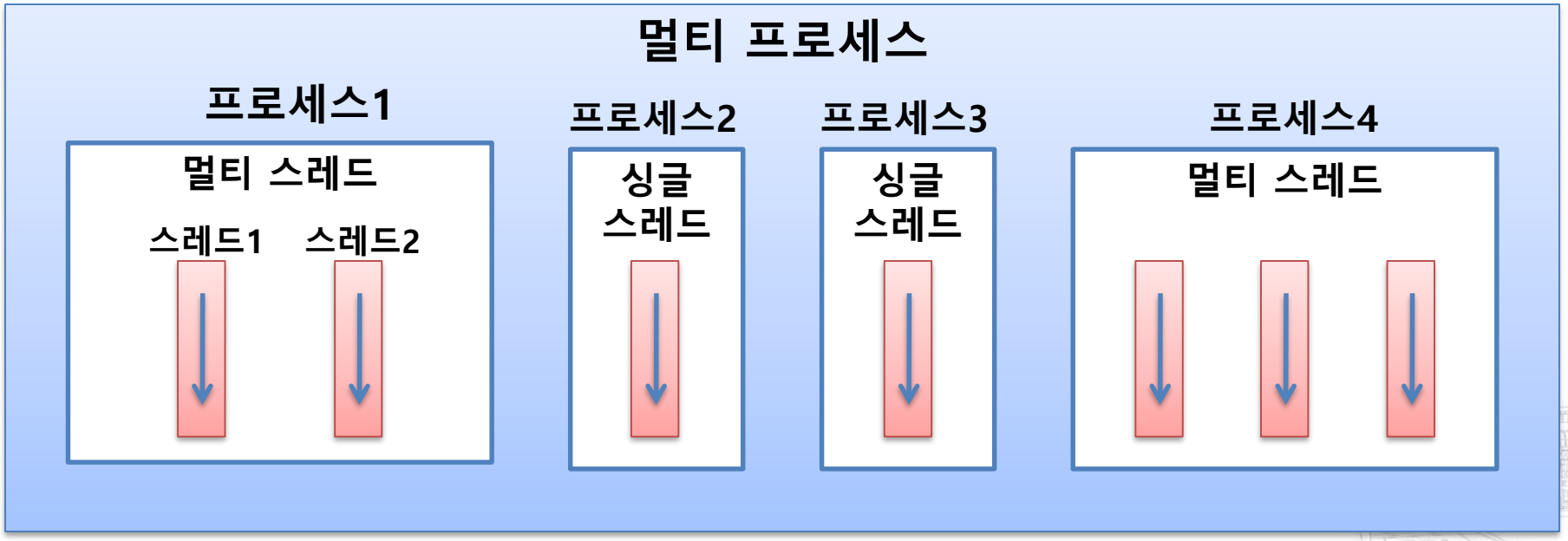




1절. 프로세스와 스레드

❖ 멀티 태스킹(multi tasking)

- 두 가지 이상의 작업을 동시에 처리하는 것
- 멀티 프로세스
  - 독립적으로 프로그램들을 실행하고 여러 가지 작업 처리
- 멀티 스레드
  - 한 개의 프로그램을 실행하고 내부적으로 여러 가지 작업 처리



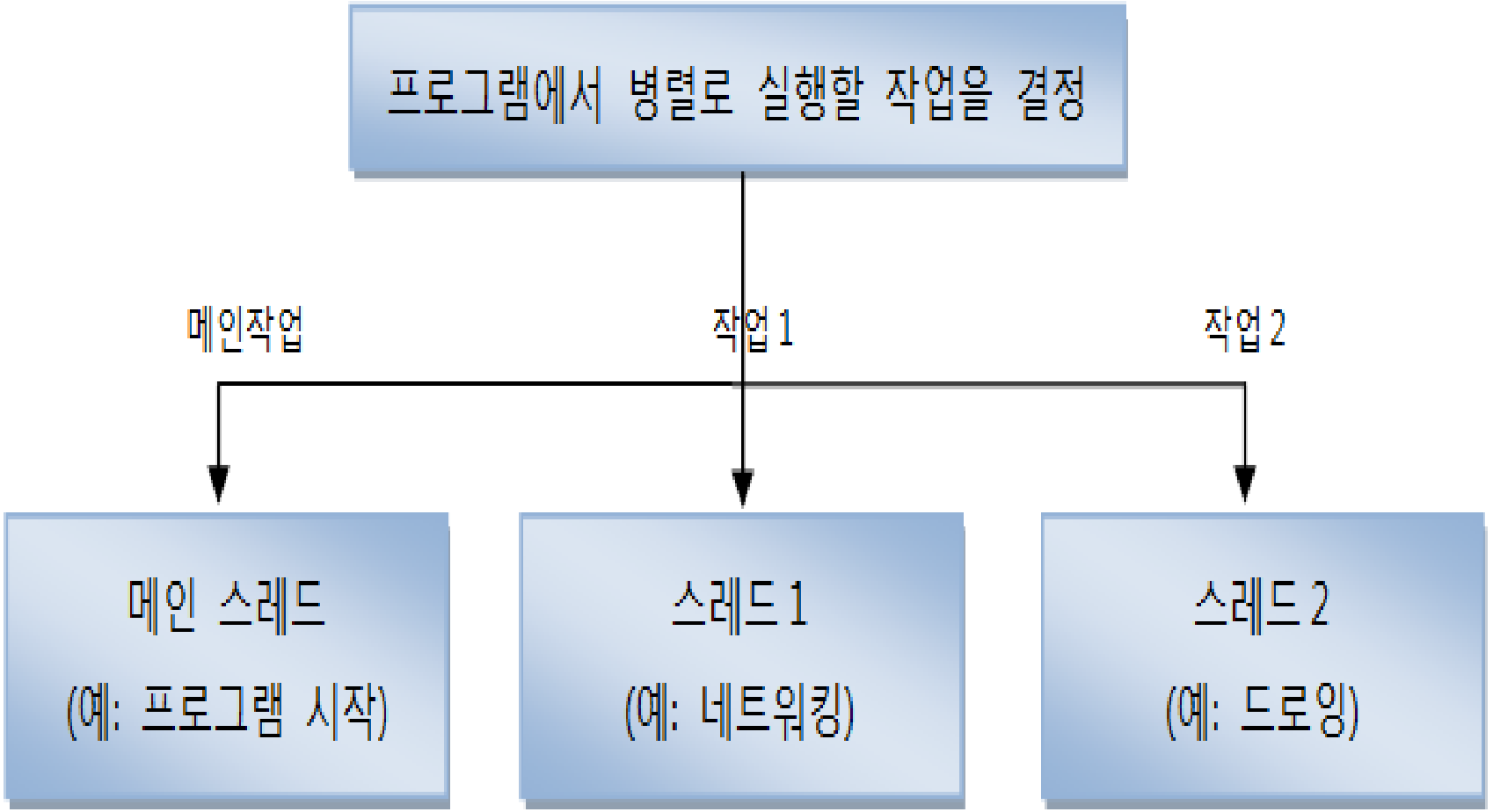
1절. 프로세스와 스레드

❖ 메인(main) 스레드

- 모든 자바 프로그램은 메인 스레드가 main() 메소드 실행하며 시작
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행
- 실행 종료 조건
  - 마지막 코드 실행
  - return 문을 만나면
- main 스레드는 작업 스레드들을 만들어 병렬로 코드들 실행
  - 멀티 스레드 생성해 멀티 태스킹 수행
- 프로세스의 종료
  - 싱글 스레드: 메인 스레드가 종료하면 프로세스도 종료
  - 멀티 스레드: 실행 중인 스레드가 하나라도 있다면, 프로세스 미종료

2절. 작업 스레드 생성과 실행

- 멀티 스레드로 실행하는 어플리케이션 개발
  - 몇 개의 작업을 병렬로 실행할지 결정하는 것이 선행되어야



2절. 작업 스레드 생성과 실행

- 작업 스레드 생성 방법
  - Thread 클래스로부터 직접 생성 (p.579~582)
    - Runnable을 매개값으로 갖는 생성자 호출
  - Thread 하위 클래스로부터 생성 (p.583~586)
    - Thread 클래스 상속 후 run 메소드를 재정의 (Overriding) 하여 스레드가 실행할 코드 작성

2절. 작업 스레드 생성과 실행

❖ 스레드의 이름 (p.586~588)

- 메인 스레드 이름: main
- 작업 스레드 이름 (자동 설정) : Thread-n

```
thread.getName();
```

- 작업 스레드 이름 변경

```
thread.setName("스레드 이름");
```

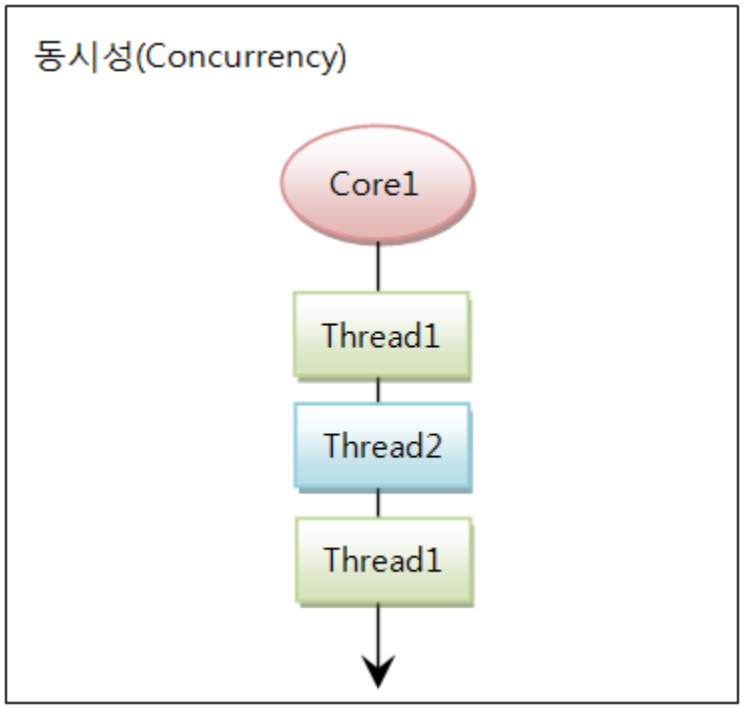
- 코드 실행하는 현재 스레드 객체의 참조 얻기

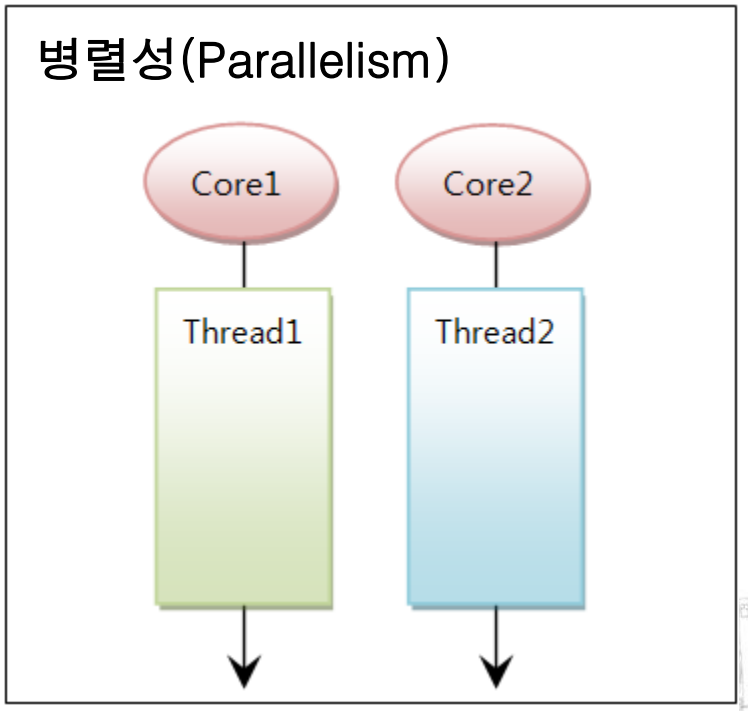
```
Thread thread = Thread.currentThread();
```

3절. 스레드 우선 순위

❖ 동시성과 병렬성

- 동시성
  - 멀티 작업을 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질
- 병렬성
  - 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질

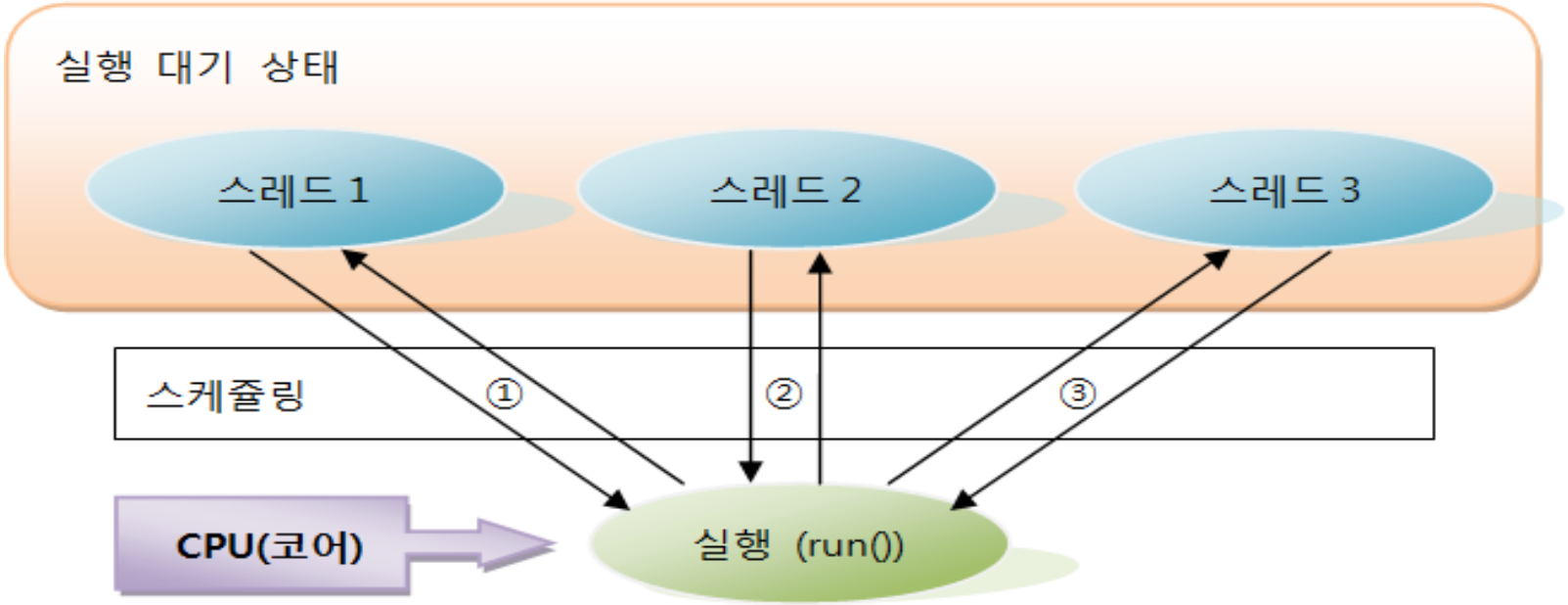




3절. 스레드 우선 순위

❖ 스레드 스케줄링

- 스레드의 개수가 코어의 수보다 많을 경우
  - 스레드를 어떤 순서로 동시성으로 실행할 것인가 결정→ 스레드 스케줄링
  - 스케줄링 의해 스레드들은 번갈아 가며 run() 메소드를 조금씩 실행



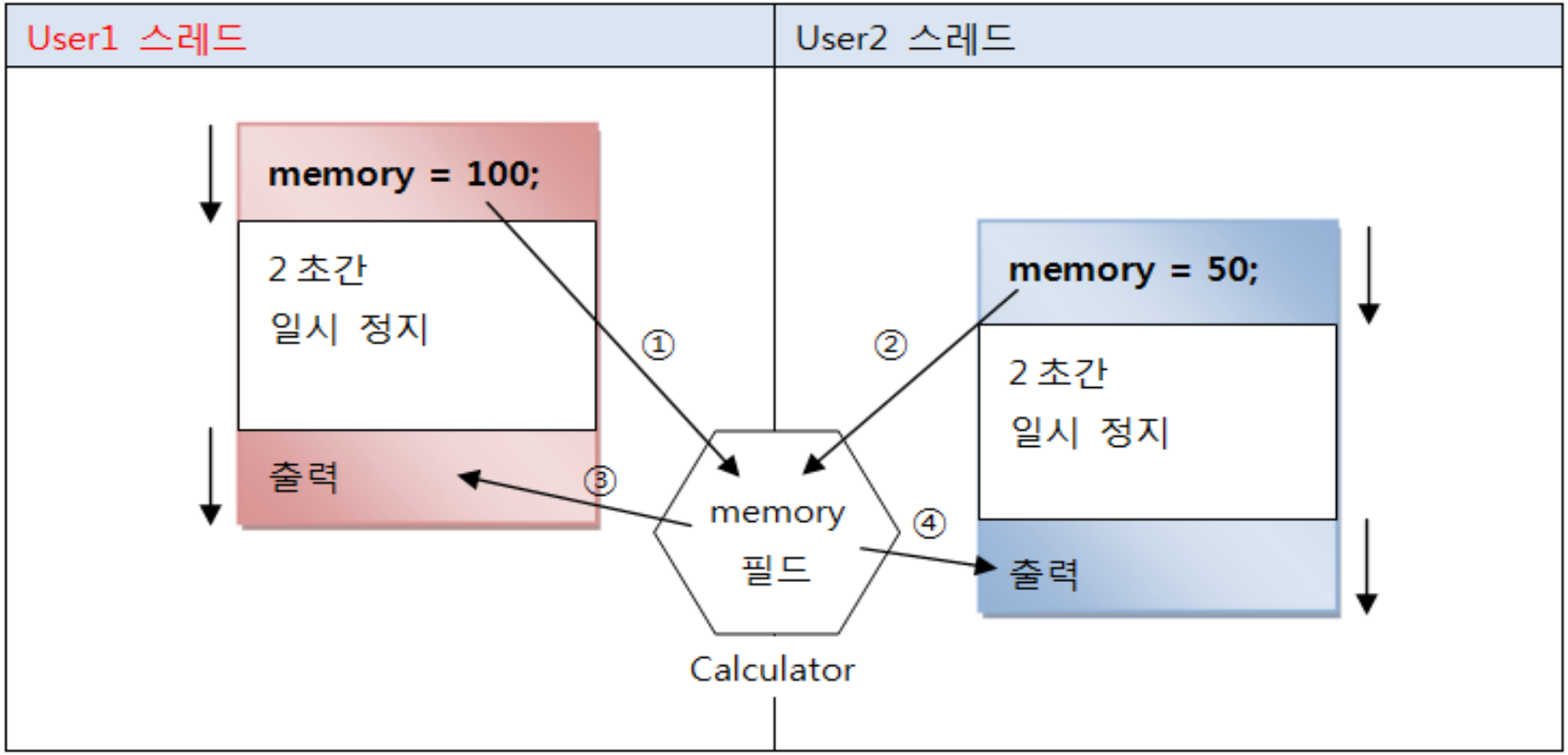
3절. 스레드 우선 순위

- 자바의 스레드 스케줄링 (p.588~590)
  - 우선 순위(Priority) 방식과 순환 할당(Round-Robin) 방식 사용
  - 우선 순위 방식 (코드로 제어 가능)
    - 우선 순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링
    - 1~10까지 값을 가질 수 있으며 기본은 5
  - 순환 할당 방식
    - JVM에 의해서 정해지기 때문에 코드로 제어할 수 없음 .
    - 시간 할당량(Time Slice) 정해서 하나의 스레드를 정해진 시간만큼 실행

4절. 동기화 메소드와 동기화 블록

❖ 공유 객체를 사용할 때의 주의할 점

- 멀티 스레드가 하나의 객체를 공유해서 생기는 오류 (p.591~593)



4절. 동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록 – synchronized(1)

- 단 하나의 스레드만 실행할 수 있는 메소드 또는 블록
- 다른 스레드는 동기화 메소드나 동기화 블록이 실행이 끝날 때까지 대기해야 함
- 동기화 메소드

```
public synchronized void method() {
    임계 영역;  //단 하나의 스레드만 실행
}
```

4절. 동기화 메소드와 동기화 블록

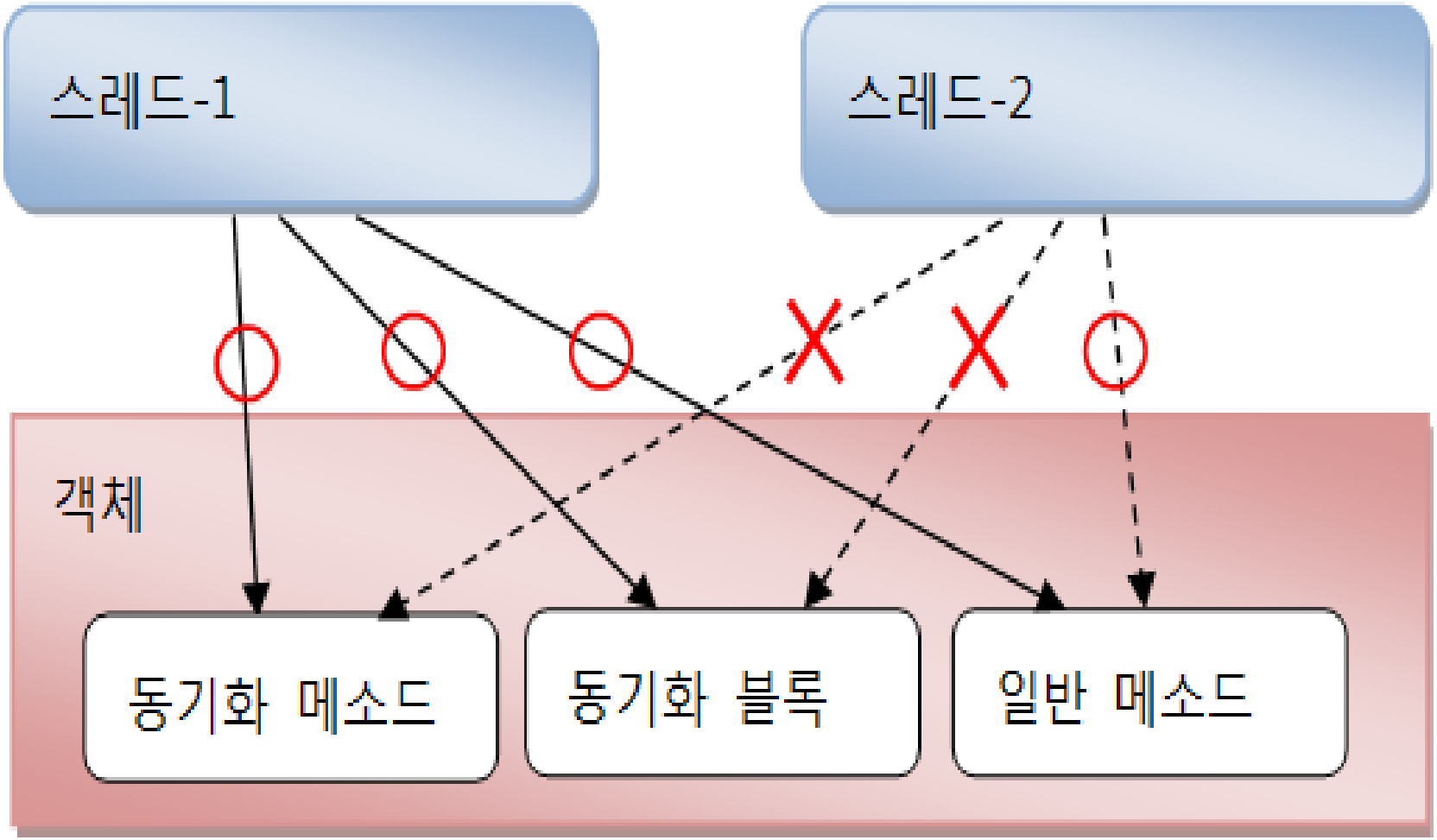
❖ 동기화 메소드 및 동기화 블록 – synchronized(2)

- 동기화 블록

```
public void method () {
  //여러 스레드가 실행 가능 영역
  ...
  synchronized(공유객체) {
    임계 영역 //단 하나의 스레드만 실행
  }
  //여러 스레드가 실행 가능 영역
  ...
}
```

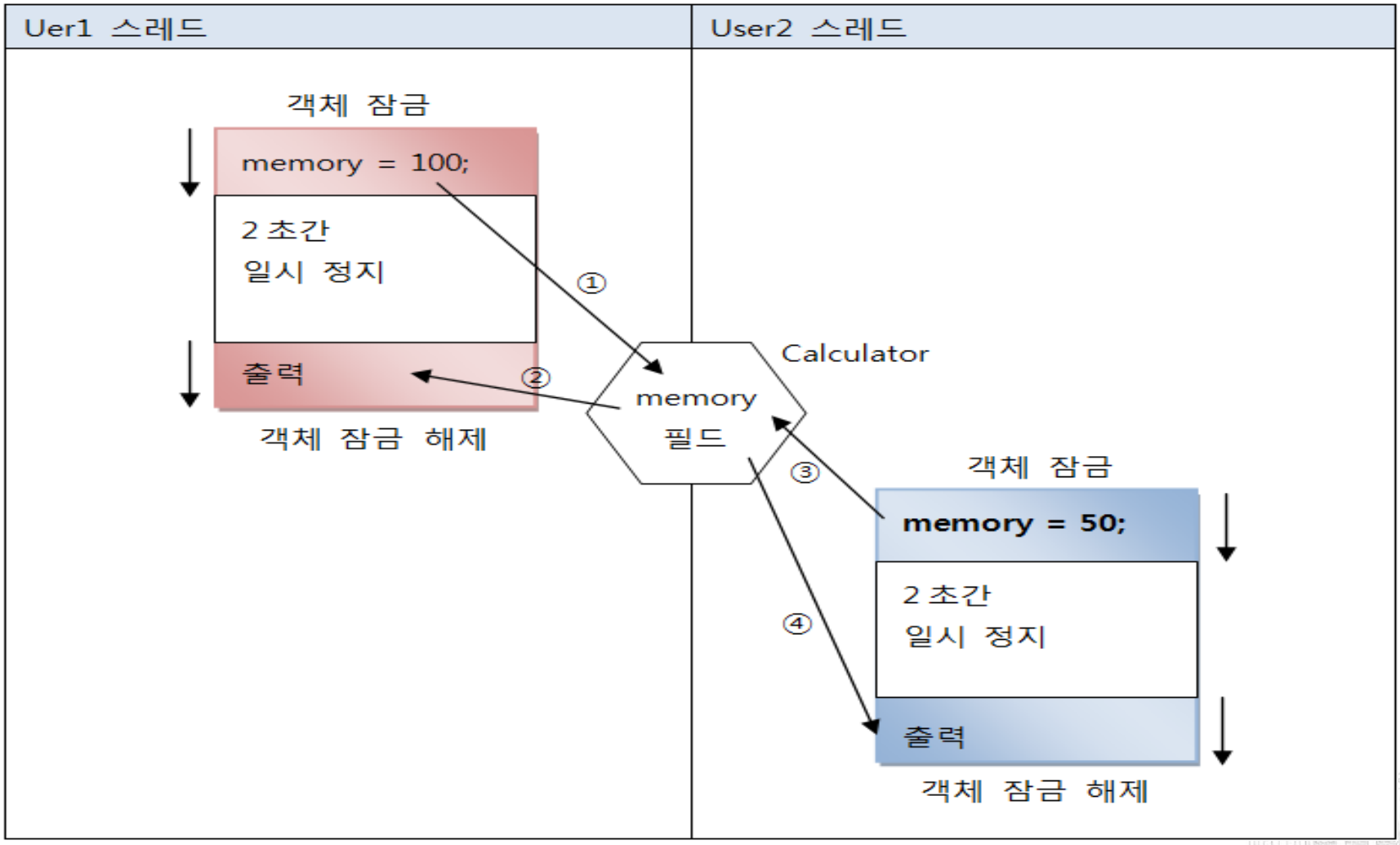
4절. 동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록 (p.593~597)(3)



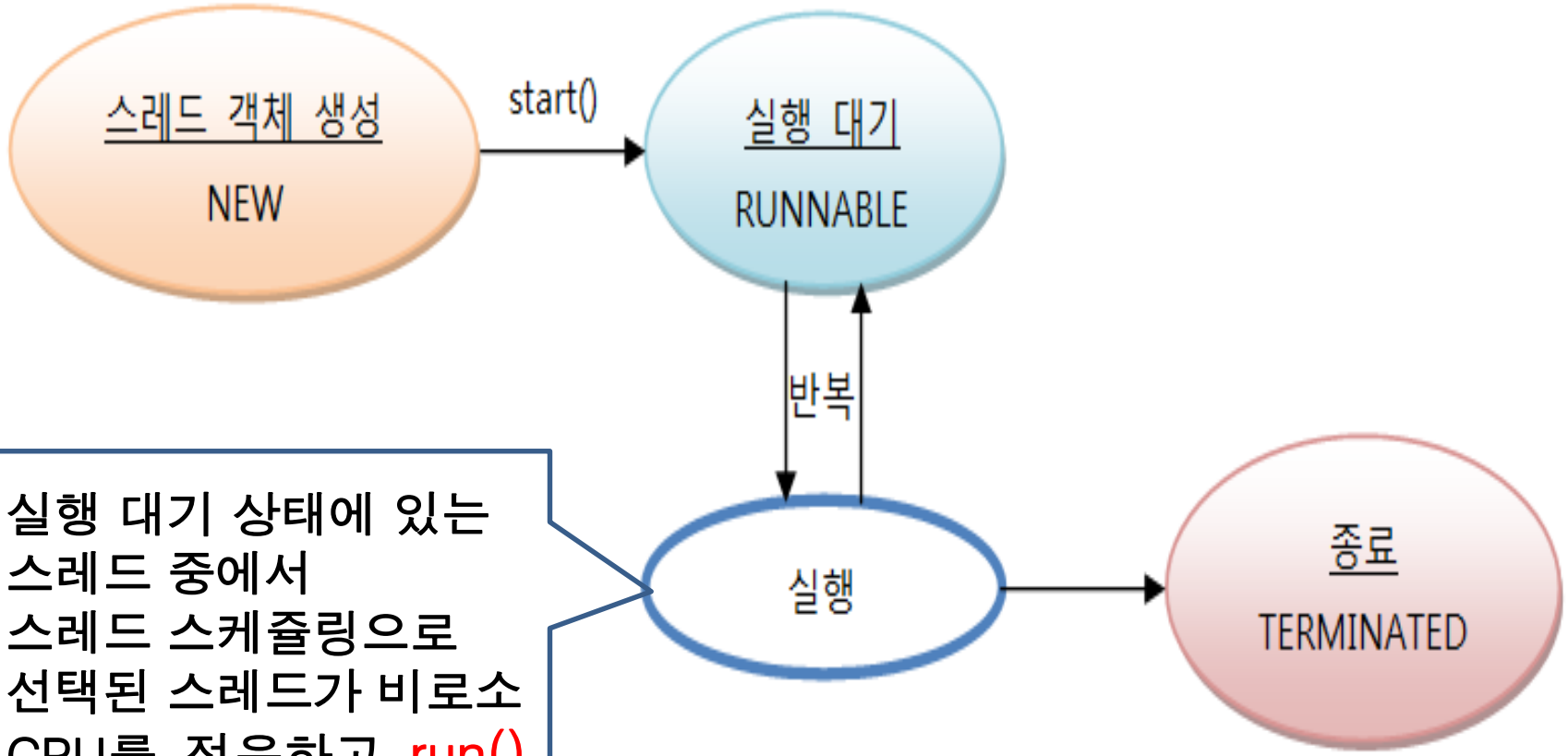
4절. 동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록 (p.593~597)(4)



5절. 스레드 상태

❖ 스레드의 일반적인 상태



실행 대기 상태에 있는 스레드 중에서 스레드 스케쥴링으로 선택된 스레드가 비로소 CPU를 점유하고 run() 메소드를 실행함.

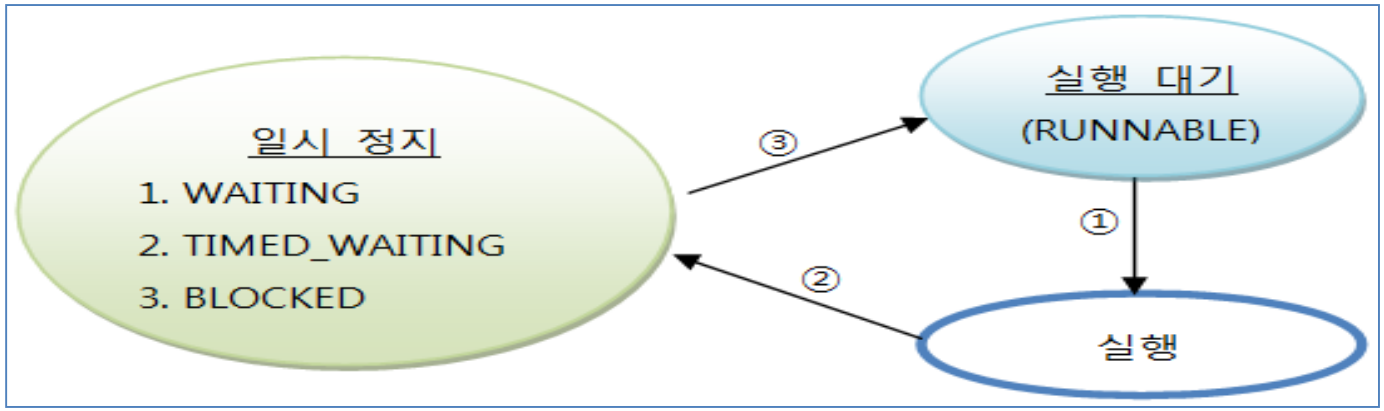
5절. 스레드 상태

- 경우에 따라서 스레드는 실행 상태에서 실행 대기 상태로 가지 않을 수 있음.
  - 실행 상태에서 일시 정지 상태로 가기도 함.
  - 일시 정지 상태
    - 스레드가 실행할 수 없는 상태
    - WAITING, TIMED_WAITING , BLOCKED
  - 일시 정지 상태의 스레드가 다시 실행 상태로 가기 위해서는 일시 정지 상태에서 실행 대기 상태로 가야 함.

5절. 스레드 상태

getState() 메서드를 이용하여 스레드 상태에 따라서 Thread.State 열거 상수를 반환함.

❖ 스레드에 일시 정지 상태 도입한 경우

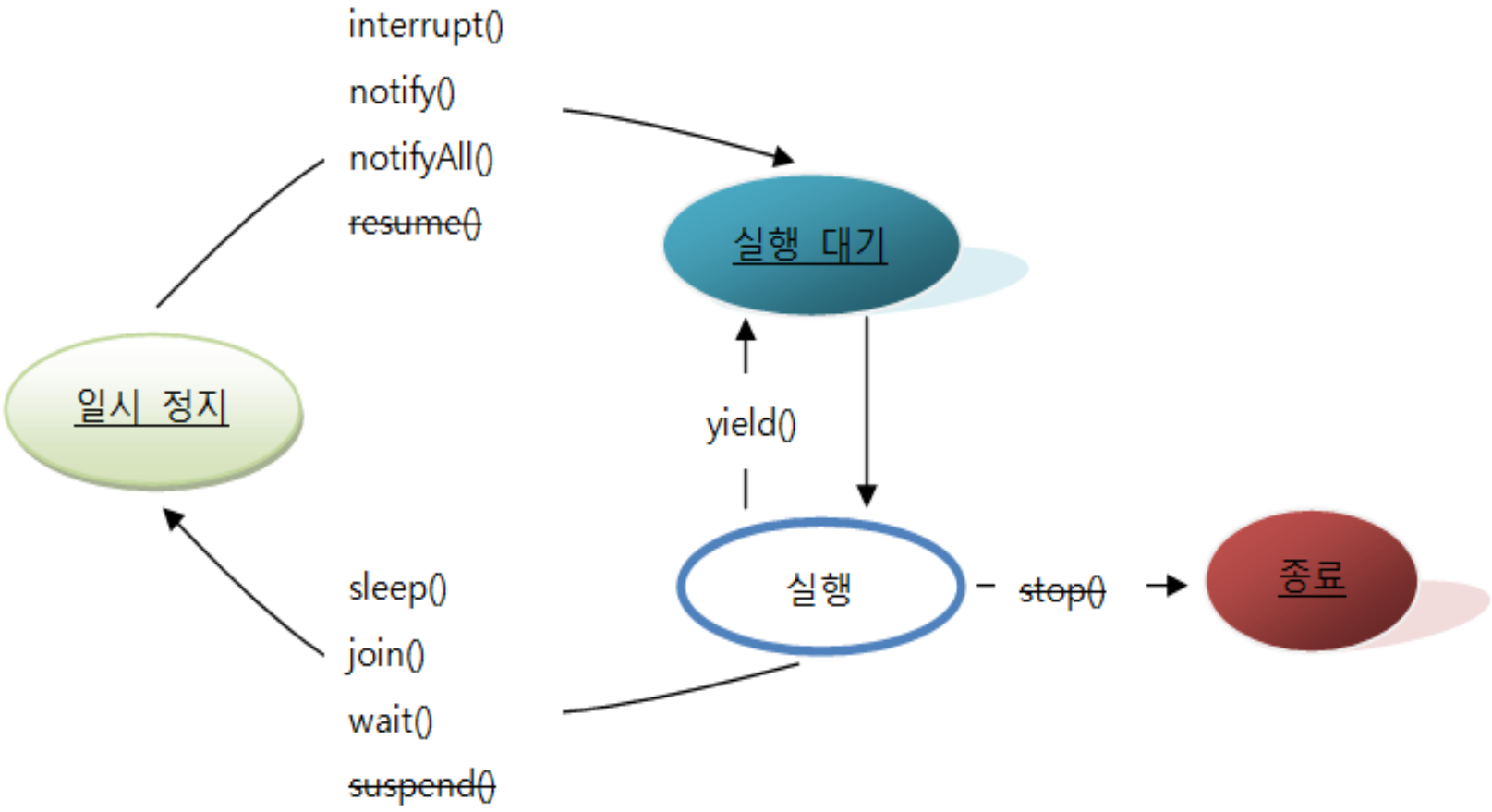


상태	열거 상수	설명
객체 생성	NEW	스레드 객체가 생성, 아직 start() 메소드가 호출되지 않은 상태
실행 대기	RUNNABLE	실행 상태로 언제든지 갈 수 있는 상태
일시 정지	BLOCKED	사용코저하는 객체의 락이 풀릴 때까지 기다리는 상태
	WAITING	다른 스레드가 통지할 때까지 기다리는 상태
	TIMED_WAITING	주어진 시간 동안 기다리는 상태
종료	TERMINATED	실행을 마친 상태

6절. 스레드 상태 제어

❖ 상태 제어

- 실행 중인 스레드의 상태를 변경하는 것
- 상태 변화를 가져오는 메소드의 종류 (취소선 가진 메소드는 사용 X)



6절. 스레드 상태 제어

❖ 주어진 시간 동안 일시 정지 – sleep()

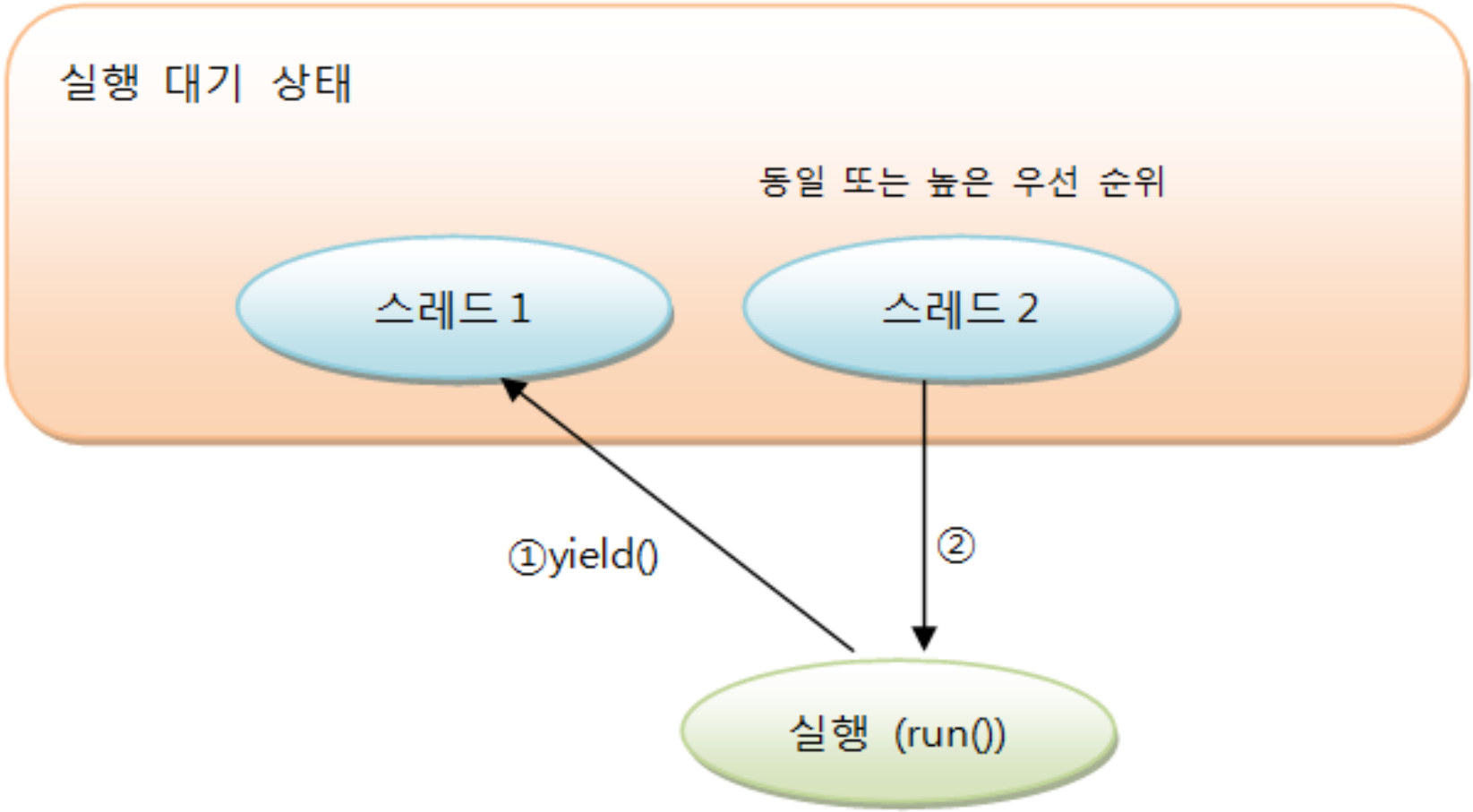
```
try {
   Thread.sleep(1000);
} catch(InterruptedException  e) {
   // interrupt() 메소드가 호출되면 실행
}
```

- 얼마 동안 일시 정지 상태로 있을 것인지 밀리 세컨드(1/1000) 단위로 지정
- 일시 정지 상태에서 interrupt() 메소드 호출
  - InterruptedException 발생

6절. 스레드 상태 제어

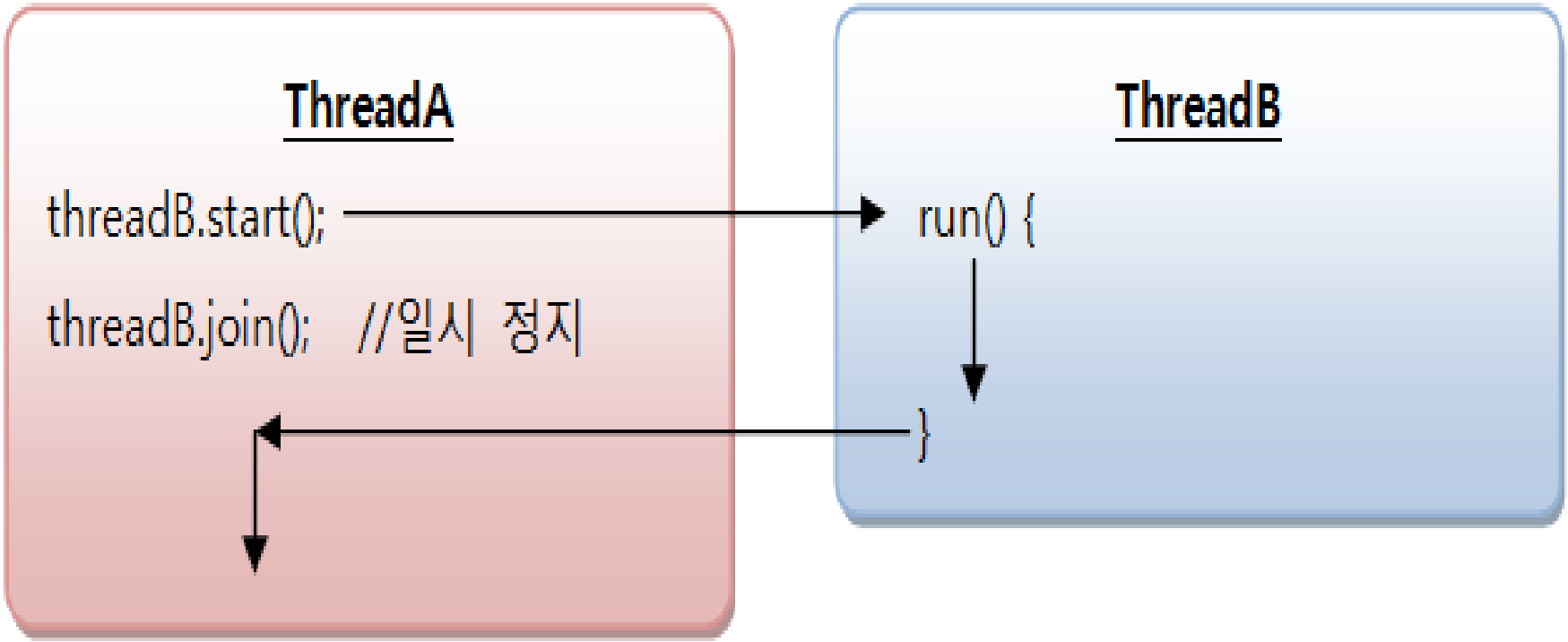
❖ 다른 스레드에게 실행 양보 – yield()

- Ex) 무의미하게 반복하는 스레드일 경우 (p.603~606)



6절. 스레드 상태 제어

❖ 다른 스레드의 종료를 기다림 – join()

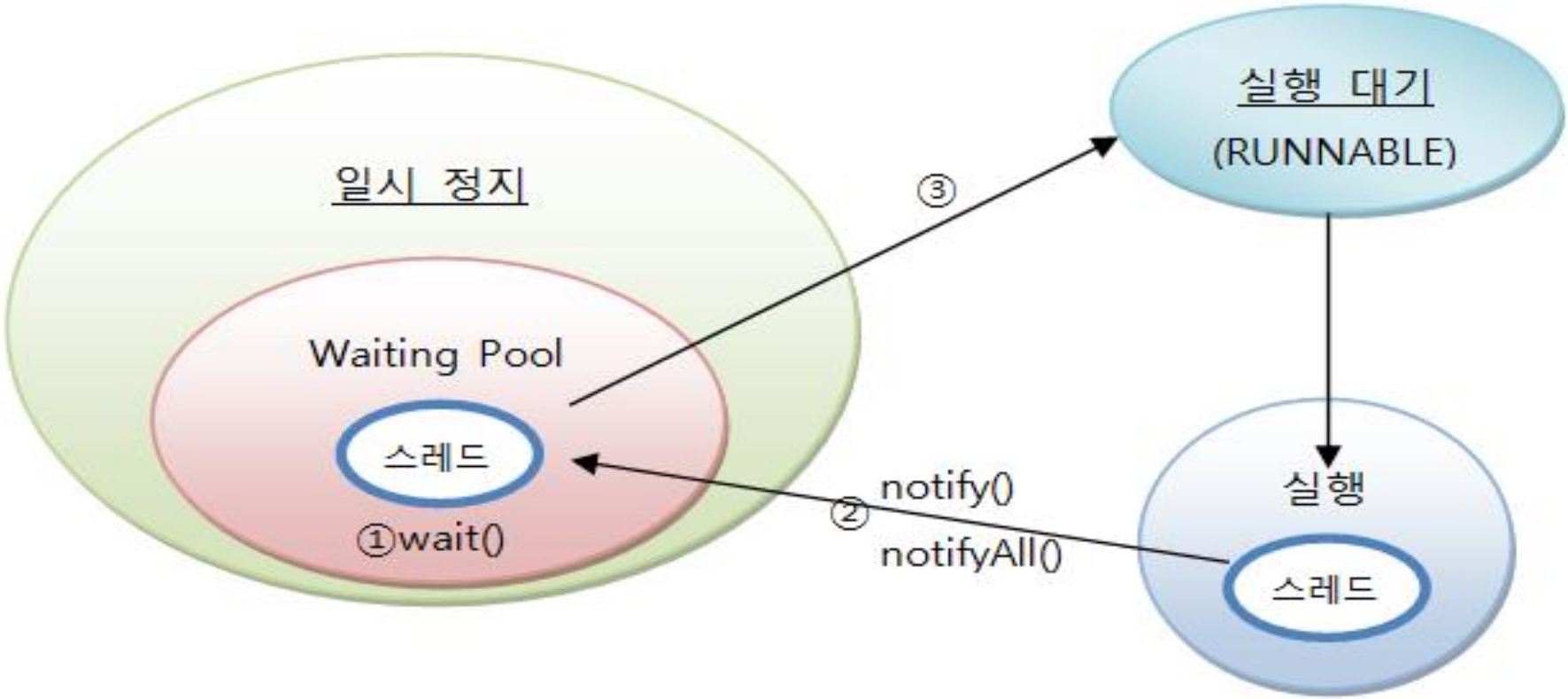


- 계산 작업을 하는 스레드가 모든 계산 작업 마쳤을 때, 결과값을 받아 이용하는 경우 주로 사용

6절. 스레드 상태 제어

❖ 스레드간 협업 – wait(), notify(), notifyAll()

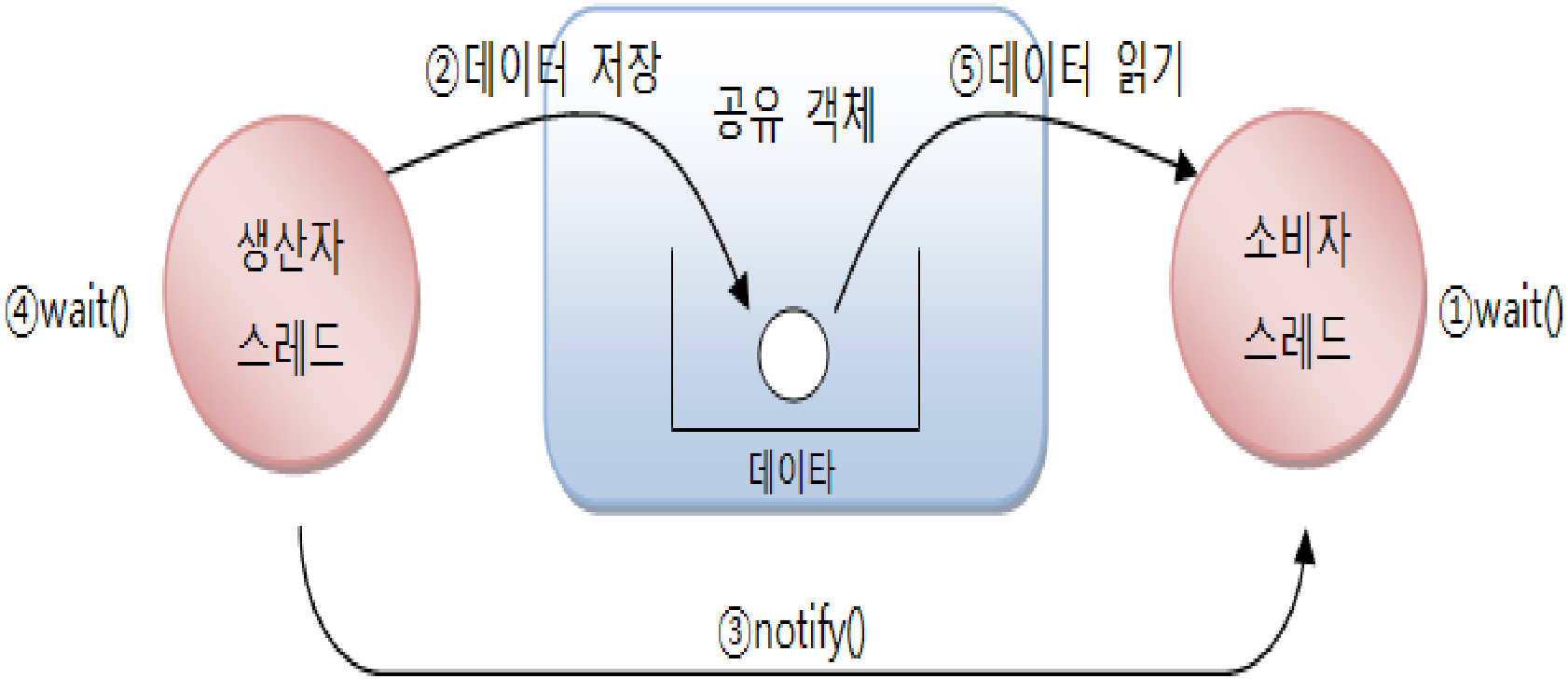
- p.608~613 에 사용 예제
- 동기화 메소드 또는 블록에서만 호출 가능한 Object의 메소드



6절. 스레드 상태 제어

❖ 스레드간 협업 – wait(), notify(), notifyAll()

- 두 개의 스레드가 교대로 번갈아 가며 실행해야 할 경우 주로 사용



6절. 스레드 상태 제어

- 스레드의 안전한 종료 – stop 플래그, interrupt()
  - 경우에 따라 실행 중인 스레드 즉시 종료해야 할 필요 있을 때 사용
  - stop() 메소드 사용시
    - 스레드 즉시 종료 되는 편리함
    - Deprecated – 사용 중이던 자원들이 불안전한 상태로 남겨짐

6절. 스레드 상태 제어

❖ 안전한 종료 위해 stop 플래그 이용하는 방법

- stop 플래그로 메소드의 정상 종료 유도

```
public class XXXThread extends Thread {
  private boolean stop;   //stop 플래그 필드

  public void run() {
    while( !stop ) {  ●---------- stop 이 true 가 되면 run() 이 종료된다.
      스레드가 반복 실행하는 코드;
    }
    //스레드가 사용한 자원 정리
  }
}
```

6절. 스레드 상태 제어

- 스레드의 안전한 종료
  - interrupt() 메소드를 이용하는 방법 (p.615~618)
    - 스레드가 일시 정지 상태일 경우 InterruptedException 발생 시킴
    - 실행대기 또는 실행상태에서는 InterruptedException 발생하지 않음
    - 일시 정지 상태로 만들지 않고 while문 빠져 나오는 방법으로도 쓰임