

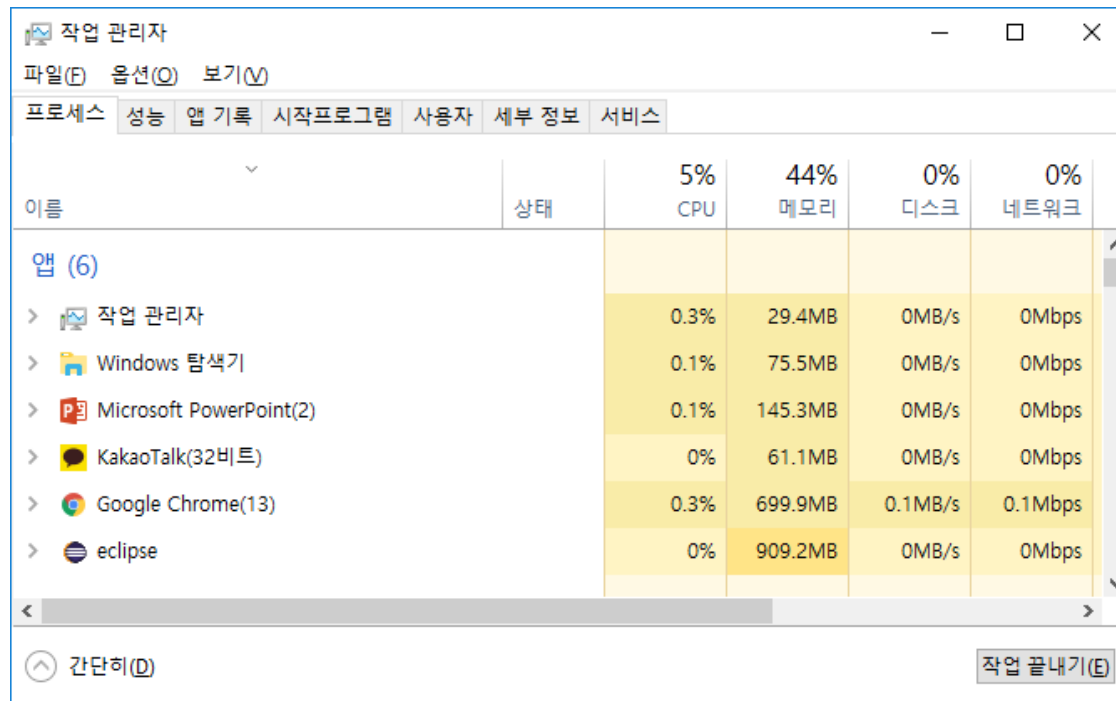
스레드 (Thread)

▶ 프로세스(Process)

간단한 의미로 실행 중인 프로그램

프로세스는 프로그램이 실행될 때마다 개별적으로 생성

하나의 프로세스는 프로그램을 수행함에 있어 필요한 데이터와 메모리 등의 할당 받은 자원, 그리고 하나 이상의 스레드로 구성됨



The screenshot shows the Windows Task Manager window titled '작업 관리자' (Task Manager). The '프로세스' (Processes) tab is selected. The table lists running applications with their names, status, and resource usage (CPU, Memory, Disk, Network). The processes listed are: 작업 관리자 (Task Manager), Windows 탐색기 (Windows Explorer), Microsoft PowerPoint(2), KakaoTalk(32비트), Google Chrome(13), and eclipse.

이름	상태	5% CPU	44% 메모리	0% 디스크	0% 네트워크
앱 (6)					
> 작업 관리자		0.3%	29.4MB	0MB/s	0Mbps
> Windows 탐색기		0.1%	75.5MB	0MB/s	0Mbps
> Microsoft PowerPoint(2)		0.1%	145.3MB	0MB/s	0Mbps
> KakaoTalk(32비트)		0%	61.1MB	0MB/s	0Mbps
> Google Chrome(13)		0.3%	699.9MB	0.1MB/s	0.1Mbps
> eclipse		0%	909.2MB	0MB/s	0Mbps

▶ 스레드(Thread)

프로세스 내에서 할당된 자원을 이용해서 실제 작업을 수행하는 작업단위
모든 프로세스는 하나 이상의 스레드를 가지며 각각 독립적인 작업단위를 가짐



▶ 스레드(Thread)

✓ 메인 스레드

모든 자바 프로그램은 메인 스레드가 `main()` 메소드를 실행하며 시작
`main()`의 첫 코드부터 아래로 순차적으로 실행되고, `return`을 만나면 종료
필요에 의해 작업 스레드들을 만들어서 병렬 코드 실행 가능
(멀티 스레드를 이용한 멀티 태스킹)

✓ 프로세스 종료

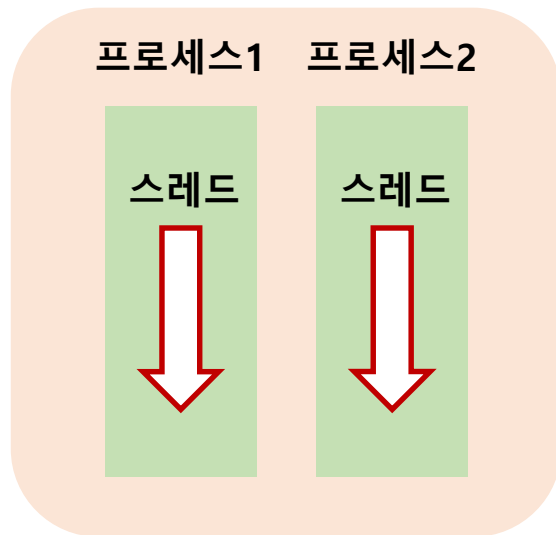
싱글 스레드의 경우 메인 스레드가 종료되면 프로세스도 종료되지만,
멀티 스레드의 경우 실행 중인 스레드가 하나라도 있다면 프로세스가
종료되지 않음

▶ 스레드(Thread)

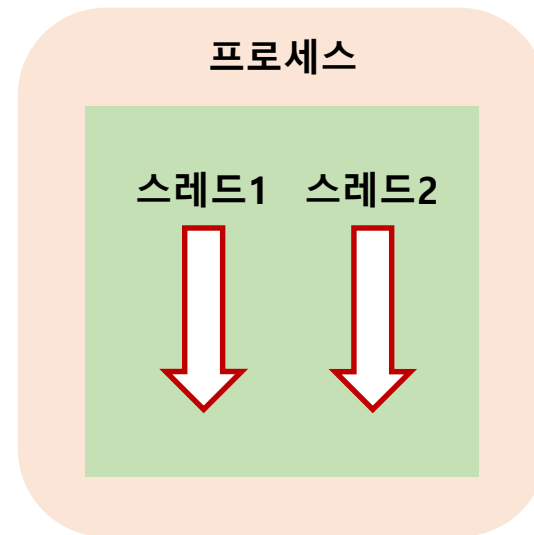
✓ 멀티 프로세스 VS 멀티 스레드

멀티 프로세스 : 각각의 프로세스를 독립적으로 실행하는 것

멀티 스레드 : 하나의 프로세스 내에서 여러 스레드가
동시에 작업을 수행하는 것



멀티 프로세스



멀티 스레드

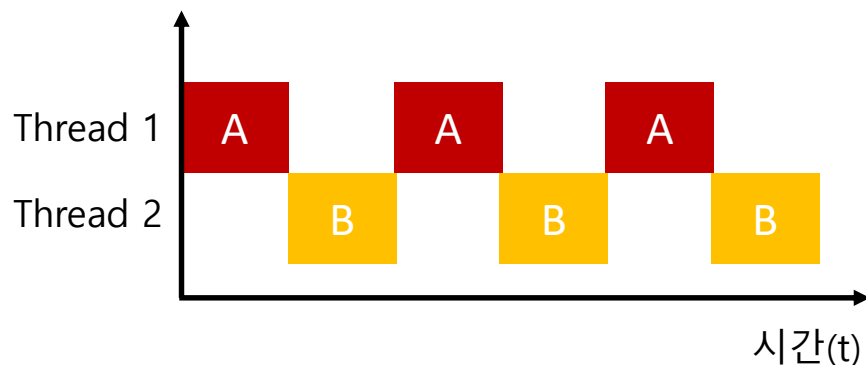
▶ 스레드(Thread)

✓ 싱글 스레드



메인 스레드 하나만 가지고 작업을 처리
→ 한 작업씩 차례대로 처리해 나감

✓ 멀티 스레드

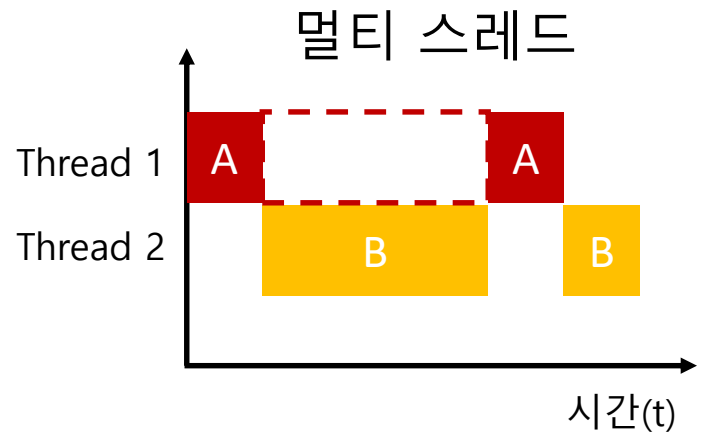
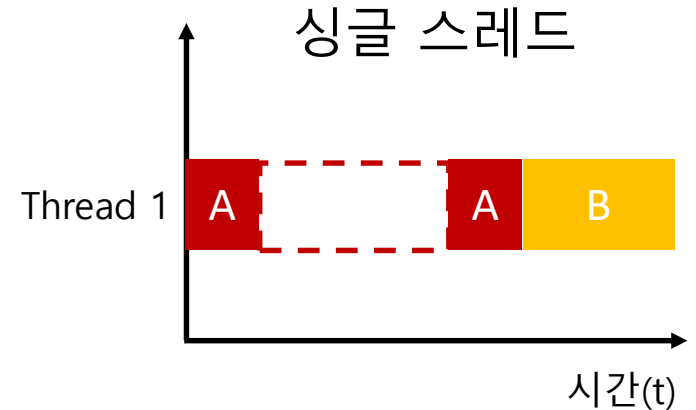



메인 스레드 외의 추가적인 스레드를
이용하여 병렬적으로 작업 처리

▶ 멀티 스레드의 장점과 단점

✓ 장점

- 자원을 보다 효율적으로 사용 가능
- 사용자의 대한 응답성 향상
- 애플리케이션의 응답성 향상
- 작업이 분리되어 코드가 간결해짐
- CPU 사용률 향상



 사용자의 입력을 기다리는 구간

▶ 멀티 스레드의 장점과 단점

✓ 단점

- 동기화(Synchronization)에 주의
- 교착상태(dead-lock)가 발생하지 않도록 주의

▶ 스레드 생성

✓ Thread클래스 상속

```
public class 클래스명 extends Thread{  
    // 상속 처리 후, run() 메소드 오버라이딩  
  
    @Override  
    public void run() {  
        // 작업하고자 하는 코드 작성  
    }  
}
```

```
public class Run {  
    public static void main(String[] args) {  
        클래스명 레퍼런스 = new 생성자(); //Thread를 상속한 객체 생성  
  
        레퍼런스.start();  
    }  
}
```

▶ 스레드 생성

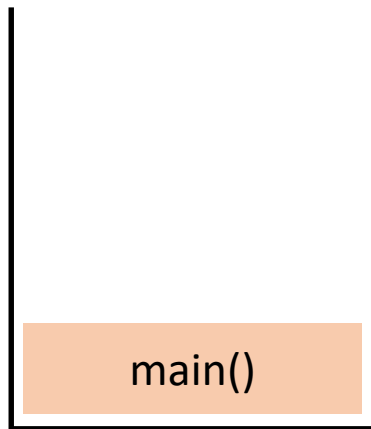
✓ Runnable 인터페이스 구현

```
public class 클래스명 implements Runnable {  
    // 상속 처리 후, run() 메소드 오버라이딩  
  
    @Override  
    public void run() {  
        // 작업하고자 하는 코드 작성  
    }  
}
```

```
public class Run {  
    public static void main(String[] args) {  
        클래스명 레퍼런스 = new 생성자(); //Runnable을 구현한 객체 생성  
  
        Thread thread = new Thread(레퍼런스);  
        thread.start();  
    }  
}
```

▶ run()과 start()

✓ run() 호출

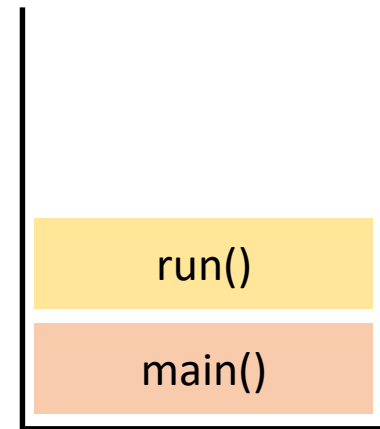


Main Thread

자바 프로그램 시작



main() 메소드에서
run() 호출

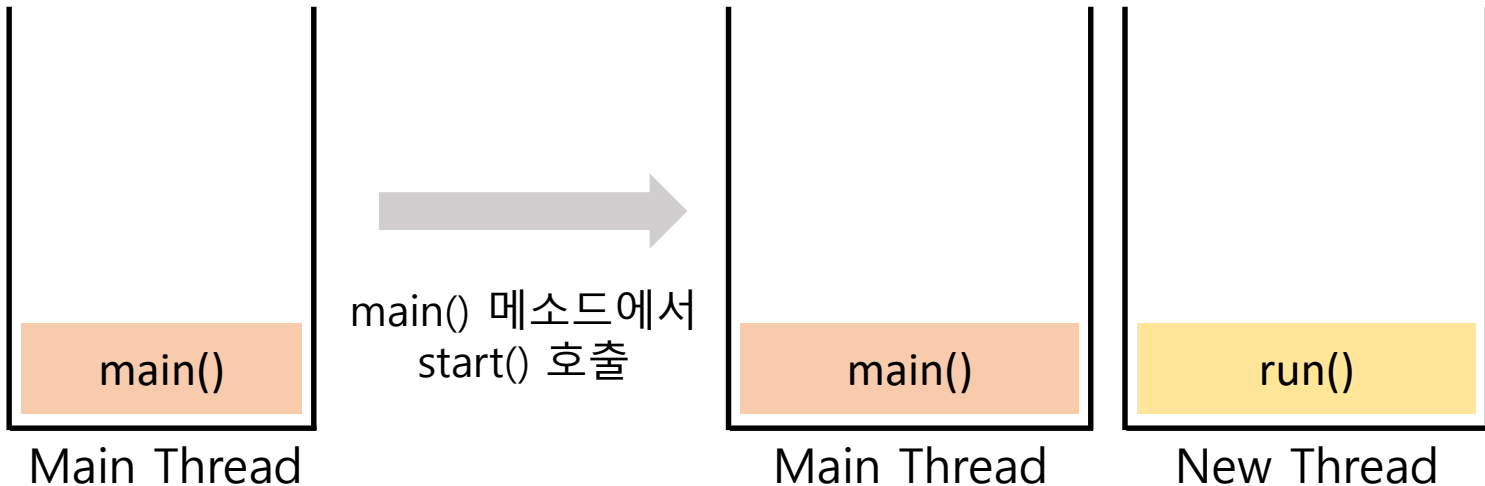


Main Thread

main()메소드 위로 쌓임

▶ run()과 start()

✓ start() 호출

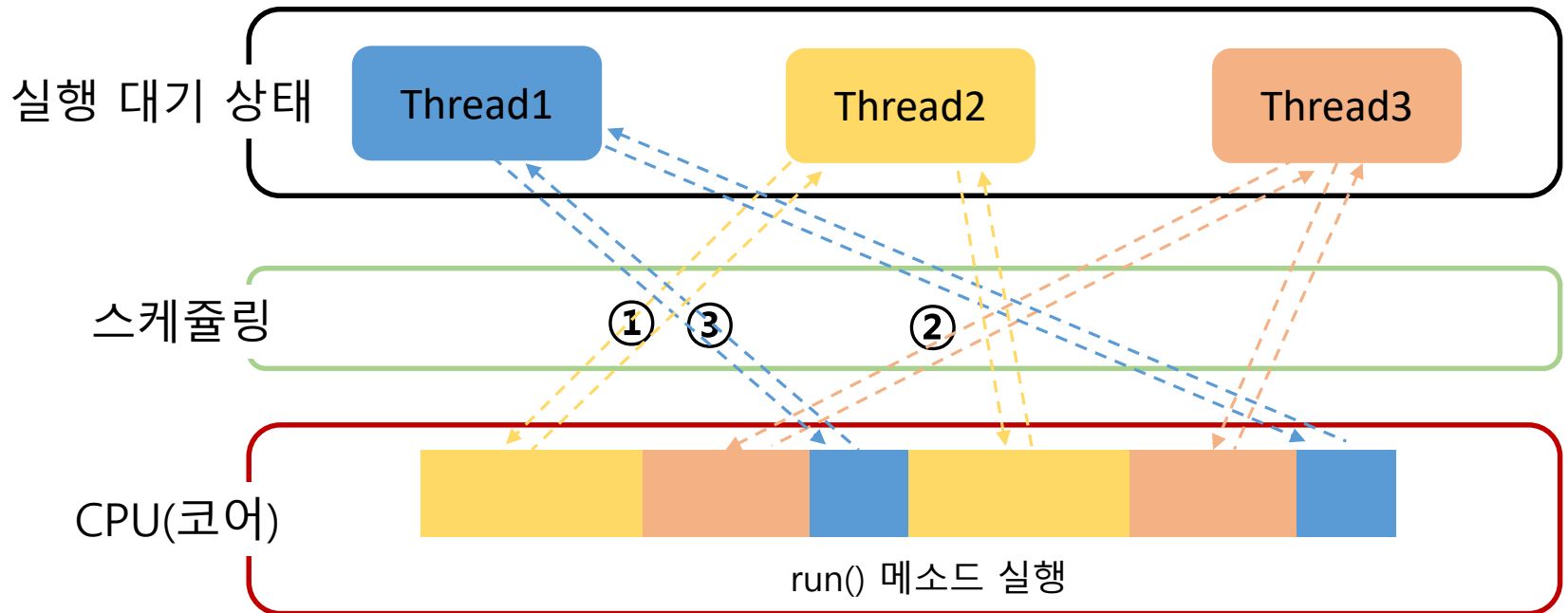


자바 프로그램 시작

start() 메소드가
새로운 스레드를 생성하고
그 위에 run() 메소드가 쌓임

▶ 스레드 스케줄링

스레드 개수가 코어의 수보다 많을 경우 스레드를 어떤 순서로 동시성을 실행할 것인가를 결정하는 것으로 스케줄링에 의해 스레드들은 번갈아가며 `run()` 메소드를 조금씩 실행



▶ 스레드 스케줄링 방식

✓ 우선 순위 방식(Priority)

- 우선 순위가 높은 스레드가 작업 시간을 더 많이 가지도록 하는 스케줄링 방식
- 스레드에 1 ~ 10 까지 우선 순위 번호 부여 가능
(번호가 높을수록 우선 순위가 높음)
- 스레드 생성 시 우선 순위 기본값은 5

```
public class Run {  
    public static void main(String[] args) {  
        클래스명 레퍼런스 = new 생성자(); //Thread를 상속한 객체 생성  
  
        레퍼런스.setPriority(1 ~ 10);  
    }  
}
```

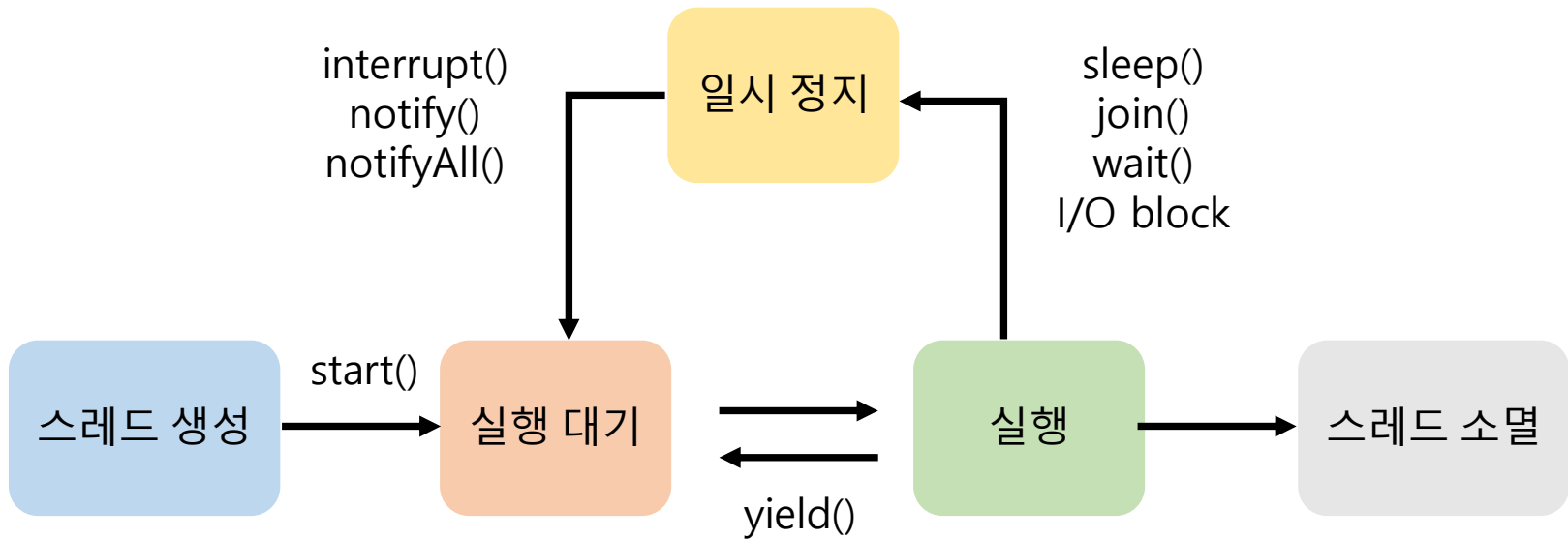
▶ 스레드 스케줄링 방식

✓ 순환 할당 방식(Round-Robbin)

- 시간 할당량(Time Slice)를 정하여 하나의 스레드를 정해진 시간만큼 실행시키는 방식
- JVM에 의해 정해짐(코드로 제어 불가능)

▶ 스레드 컨트롤

실행 중인 스레드의 상태를 제어하기 위한 것으로
효율적이고 정교한 스케줄링을 위한 스레드 상태를 제어하는 기능

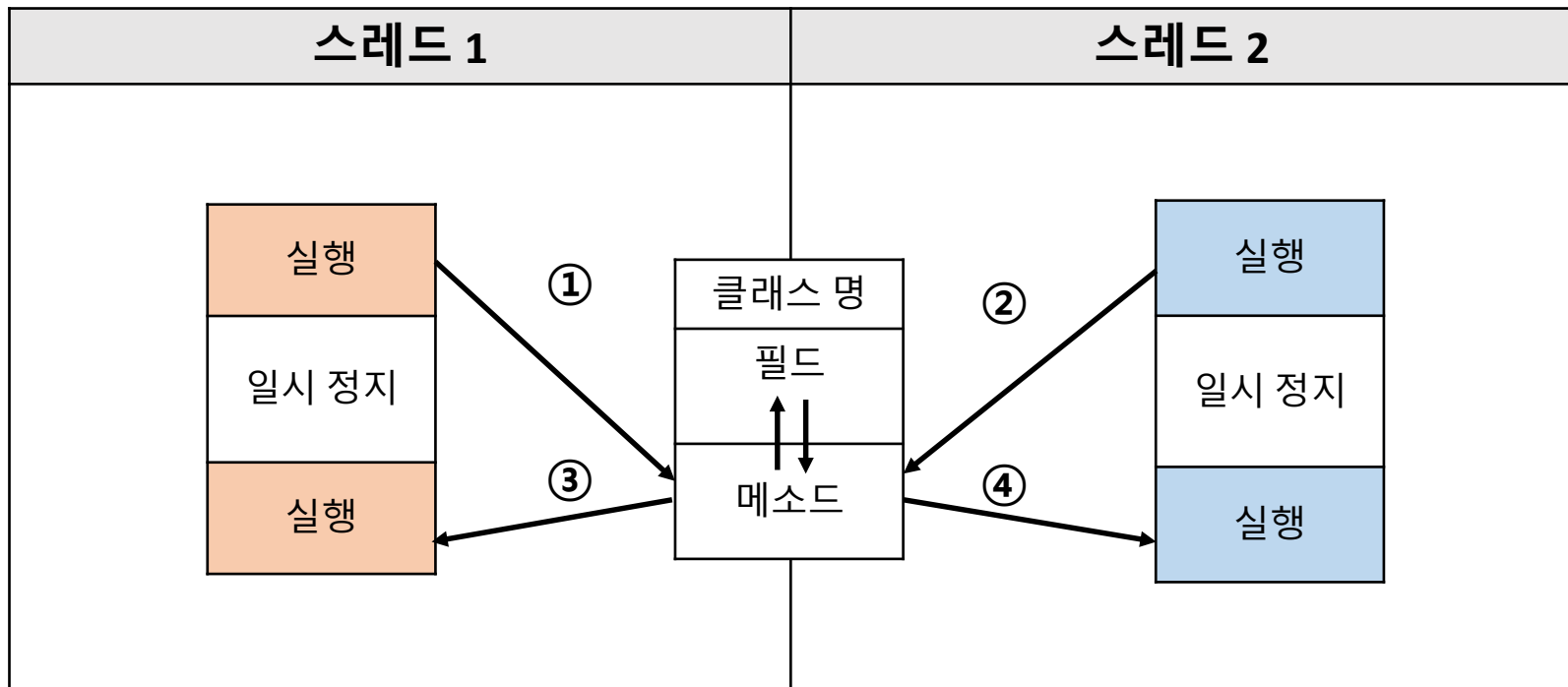


▶ 스레드 컨트롤

메소드	설 명
<code>void interrupt()</code>	<code>sleep()</code> 나 <code>join()</code> 에 의해 일시 정지 상태인 스레드를 실행 대기 상태로 만듦. 해당 스레드에서는 <code>InterruptedException</code> 이 발생해 일시 정지를 벗어남
<code>void join()</code> <code>void join(long millis)</code> <code>void join(long millis, int nanos)</code>	자신이 하던 작업을 잠시 멈추고 다른 스레드가 지정된 시간 동안 실행되도록 함. 지정된 시간이 지나거나 작업이 종료되면 <code>join()</code> 을 호출한 스레드로 다시 돌아와 실행을 계속 함.
<code>static void sleep(long millis)</code> <code>static void sleep(long millis, int nanos)</code>	지정된 시간 동안 스레드를 일시 정지 시킴. 지정한 시간이 지나고 나면, 자동적으로 다시 실행 대기 상태가 됨
<code>static void yield()</code>	실행 중에 다른 스레드에게 양보하고 실행 대기 상태가 됨
<code>void wait()</code> <code>void wait(long timeout)</code> <code>void wait(long timeout, int nanos)</code>	동기화된 블록 안에서 다른 스레드가 이 객체의 <code>notify()</code> , <code>notifyAll()</code> 을 호출하거나 지정된 시간이 지날 때 까지 현재의 스레드를 대기시킴
<code>void notify()</code>	동기화된 블록 안에서 호출한 객체 내부에 대기중인 스레드를 깨움. 여러 스레드가 있을 경우 임의의 스레드 하나에만 통보
<code>void notifyAll()</code>	동기화된 블록 안에서 호출한 객체 내부에 대기중인 모든 스레드를 깨움, 하지만 lock는 하나의 스레드만 얻을 수 있음

▶ 동기화(Synchronized)

한 번에 한 개의 스레드만 프로세스 공유 자원(객체)에 접근할 수 있도록 락(Lock)을 걸어 다른 스레드가 진행 중인 작업에 간섭하지 못하도록 하는 것



▶ 동기화(Synchronized)

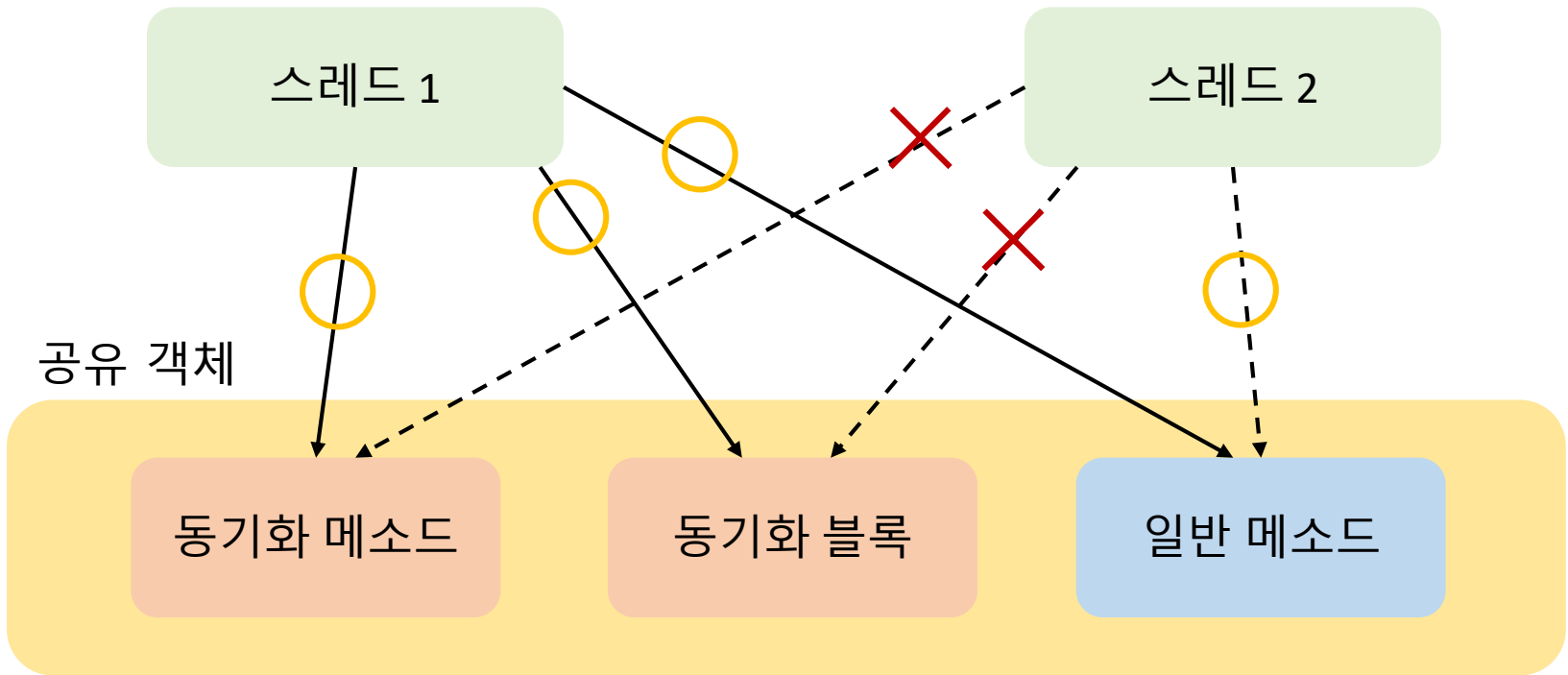
✓ 동기화 메소드

```
public synchronized void method() {  
    // 한 개의 스레드만 실행할 수 있음  
}
```

✓ 동기화 블록

```
public void method1() {  
    // 여러 스레드 실행할 수 있음  
  
    synchronized (공유객체) {  
        // 한 개의 스레드만 실행할 수 있음  
    }  
    // 여러 스레드 실행할 수 있음  
}
```

▶ 동기화 메소드와 동기화 블록



▶ 동기화 메소드와 동기화 블록

