프로세스, 스레드,리액티브

부종민

슬라이드 URL - https://goo.gl/eNgMnS

Hello!

부종민

boojongmin@gmail.com

목차

1.

Process, Thread

들어가며..
Thread 일반
Thread in JVM

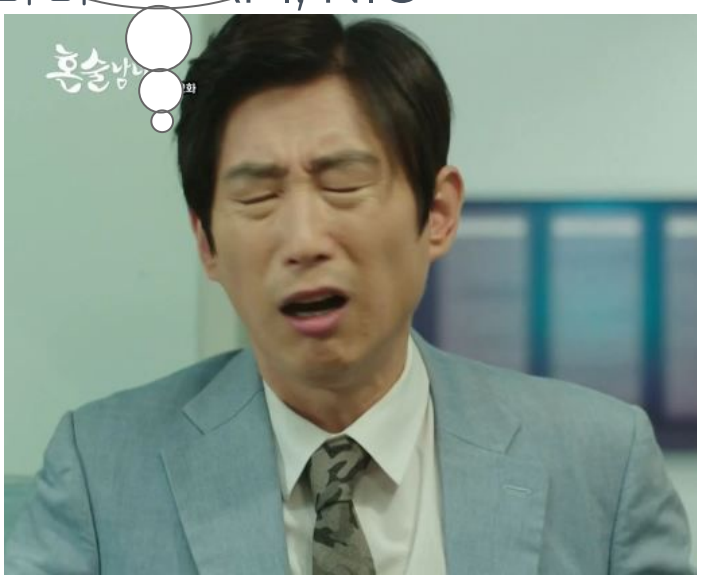
들어가며..

- 다룰 주제
  - Process, Thread, ThreadPool, 자바비동기API, NIO

들어가며..

범위가 넓어도 너무
넓어~~

- 다룰 주제
  - Process, Thread, ThreadPool, 자바IO API, NIO



들어가며..

- 프로세스, 스레드
  - 컴공 2~3학년때 배우는 개념
  - 맨날 듣는... 모른다고 하긴 뭐하고 안다고하긴 깨림직하고...
  - 현장에서 딱히 몰라도 개발 가능...
  - 내용은 너무 이론이라 딱딱....

들어가며..

- 프로세스, 스레드
  - 컴공 2~3학년때 배우
  - 맨날 듣는... 모른다고
  - 현장에서 딱히 몰라도
  - 내용은 너무 이론이라

직하고...

Thread 일반

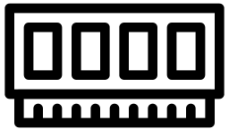
프로세스 - 독립된 실행의 단위

- 과거
  - cpu - 프로세스가 독점
  - memory - 프로세스가 메모리 공간을 자유롭게 사용
- 현대
  - cpu - OS의 스케줄러에 의해 time slice만큼 실행
  - memory - OS로 할당받은 메모리 공간 사용.
    - CODE, DATA, HEAP, STACK

Thread 일반

- (현대) 프로세스
  - 프로세스 := 스레드 + 메모리공간

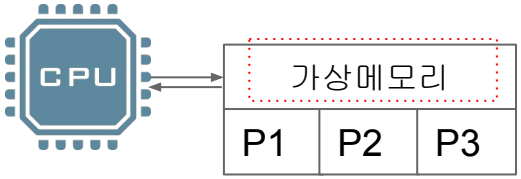
Thread 일반



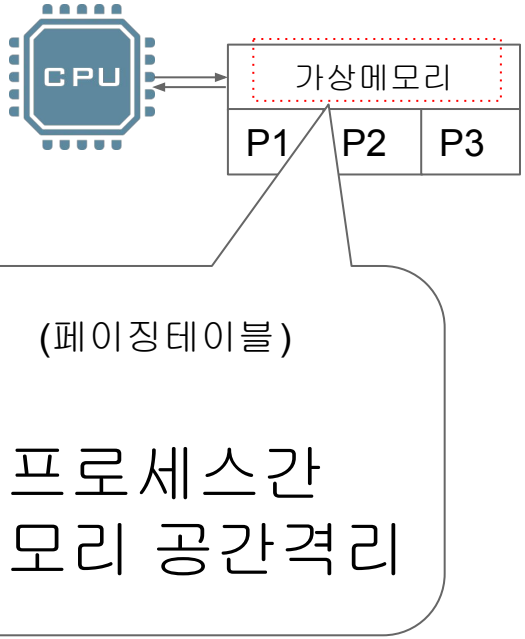
프로세스

프로그램

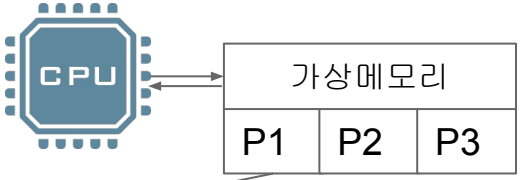
Thread 일반

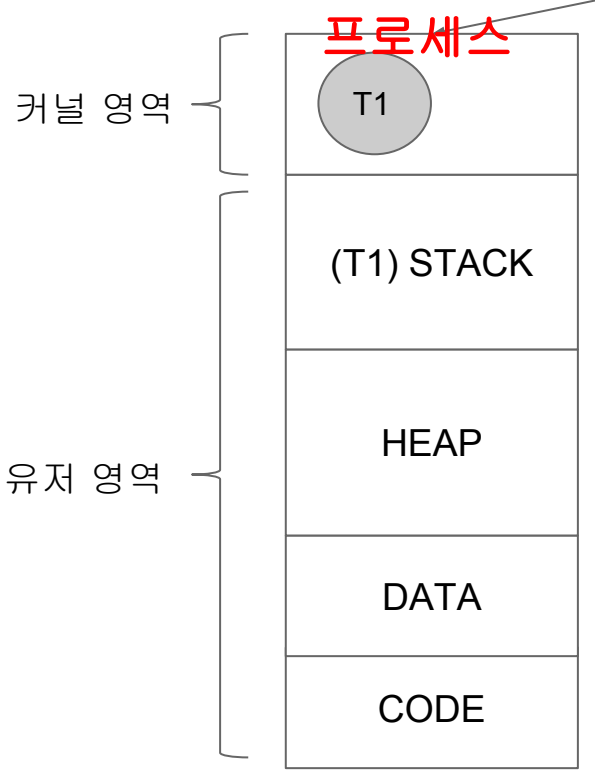


Thread 일반

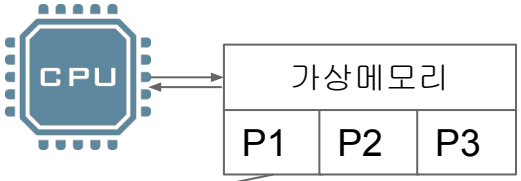


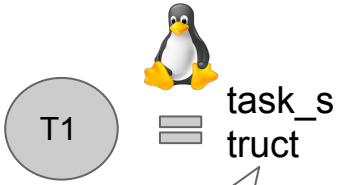
Thread 일반

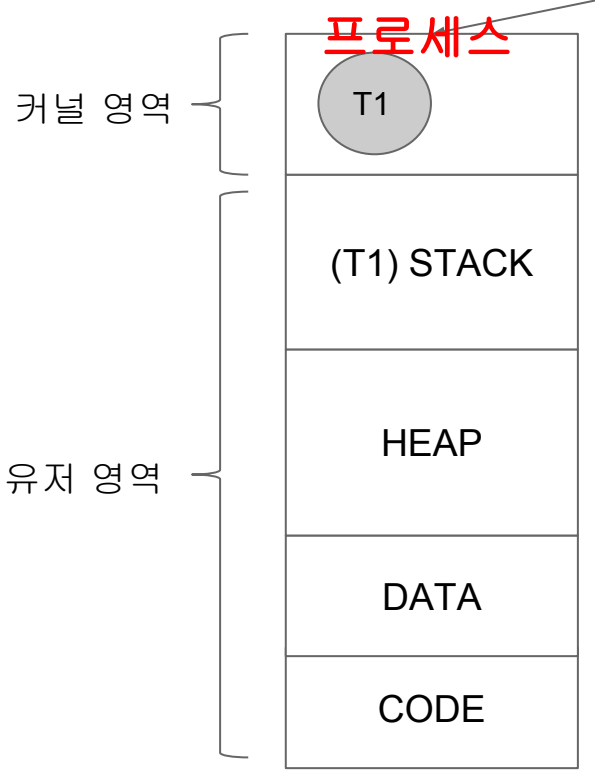




Thread 일반

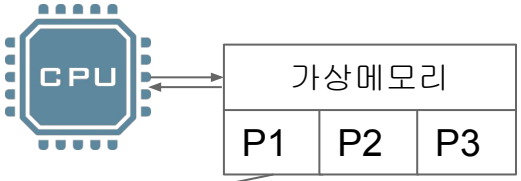


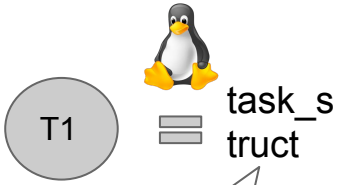


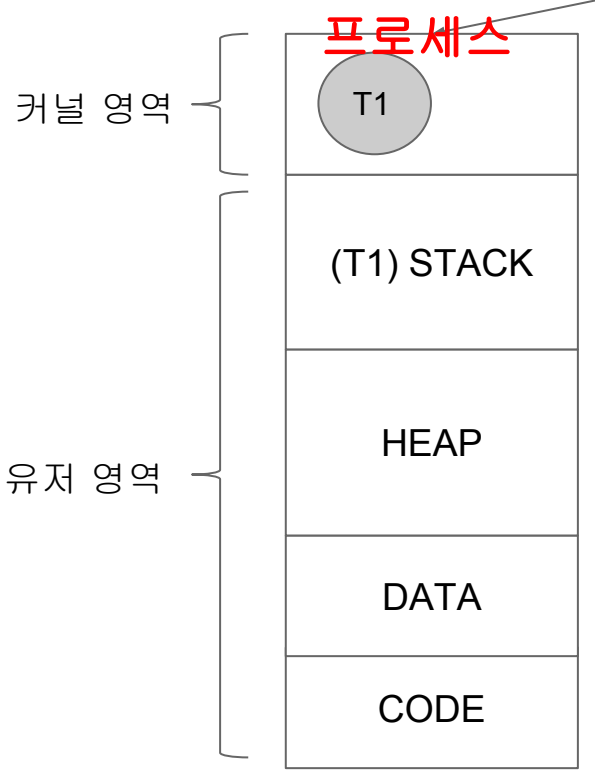


스케줄링 대상

Thread 일반

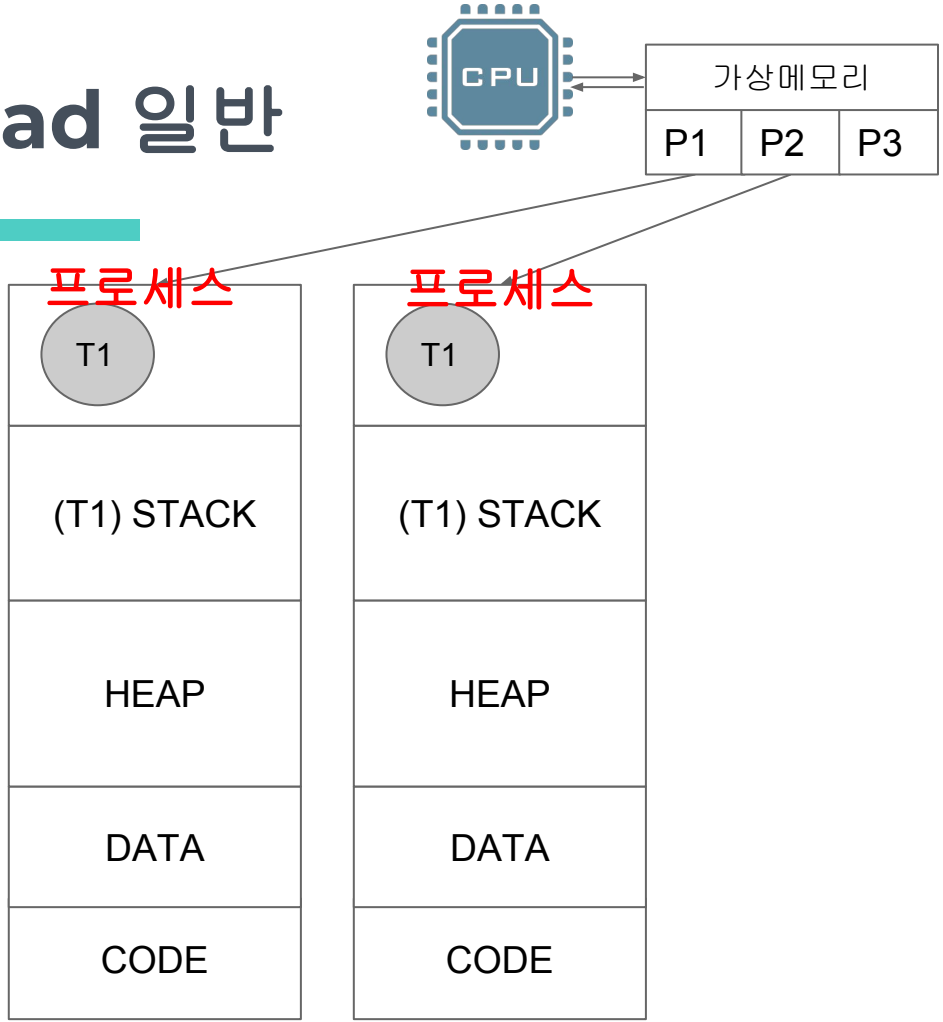




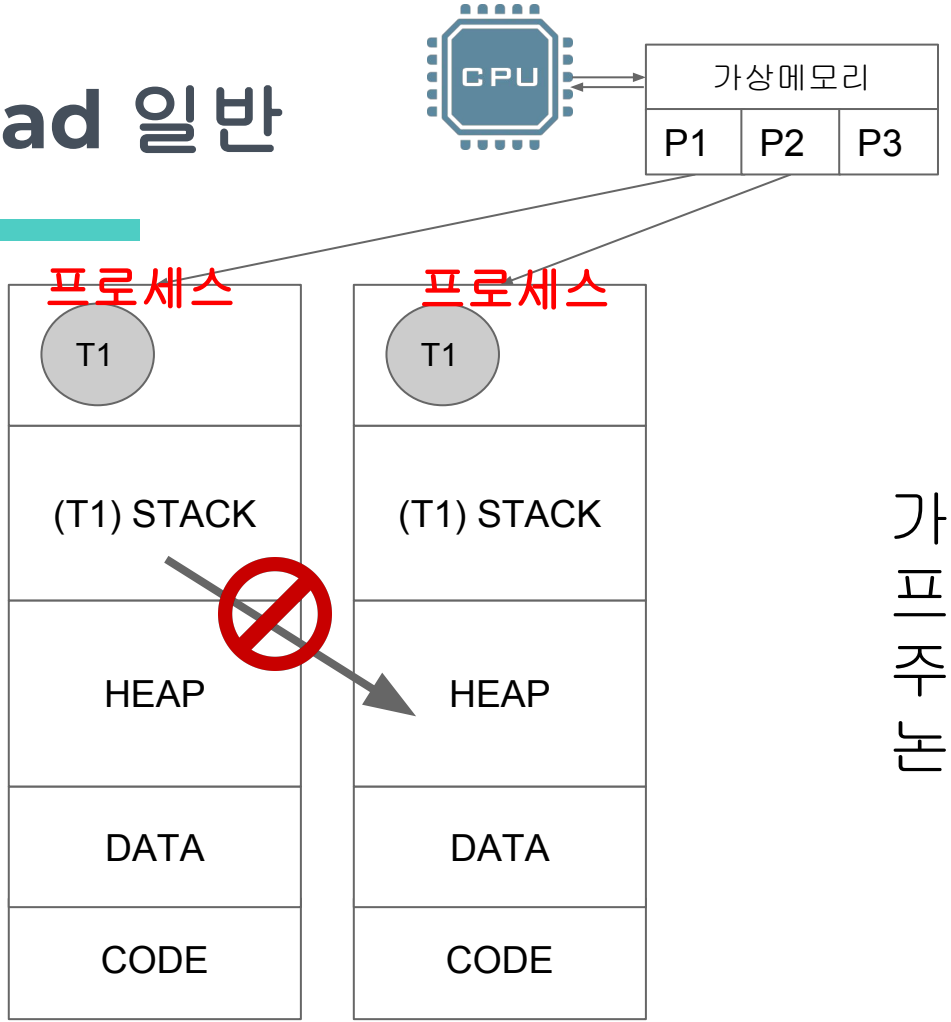


스케줄링 대상

Thread 일반

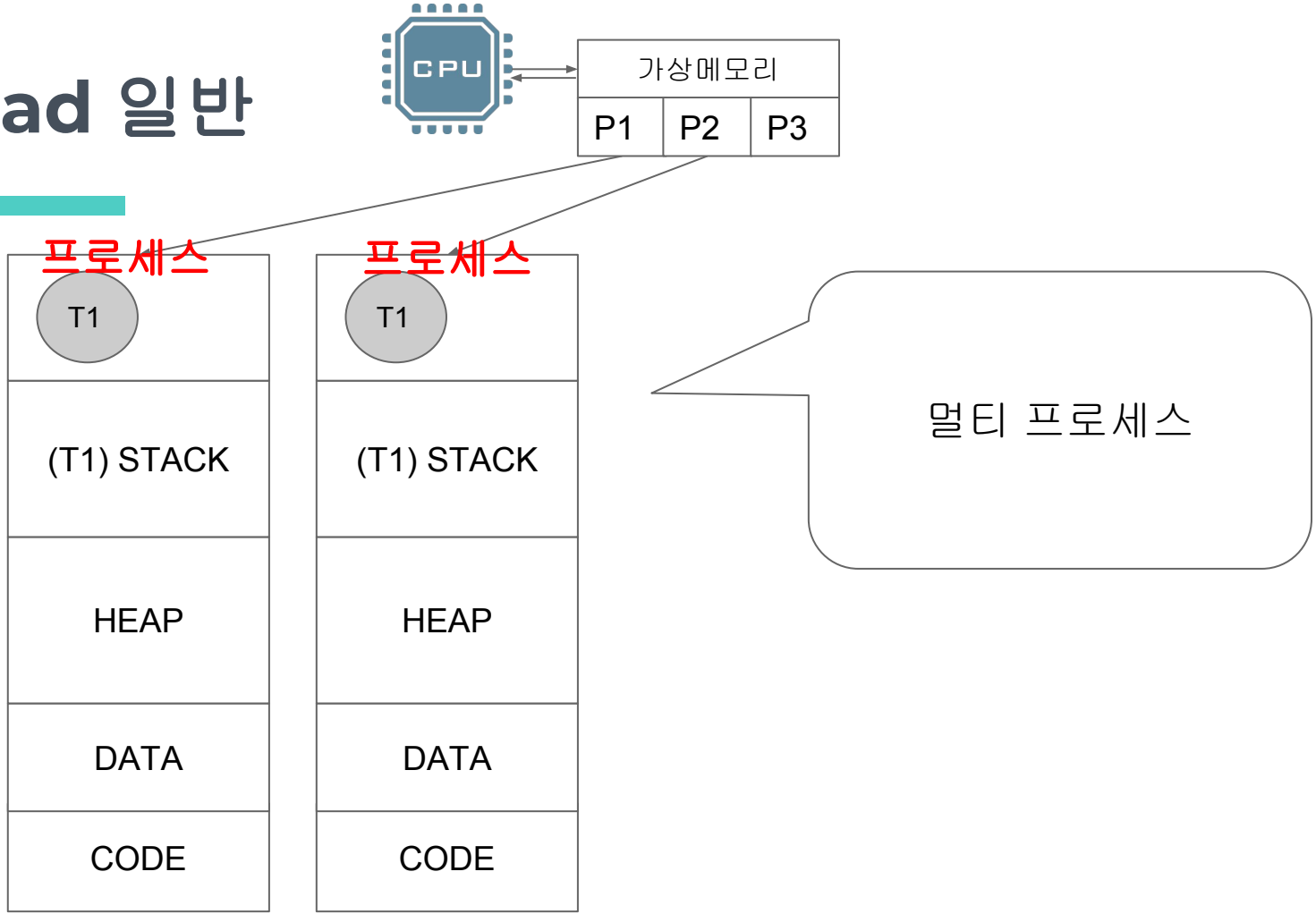

CPU
가상메모리
P1
P2
P3
프로세스
T1
(T1) STACK
HEAP
DATA
CODE
프로세스
T1
(T1) STACK
HEAP
DATA
CODE

Thread 일반

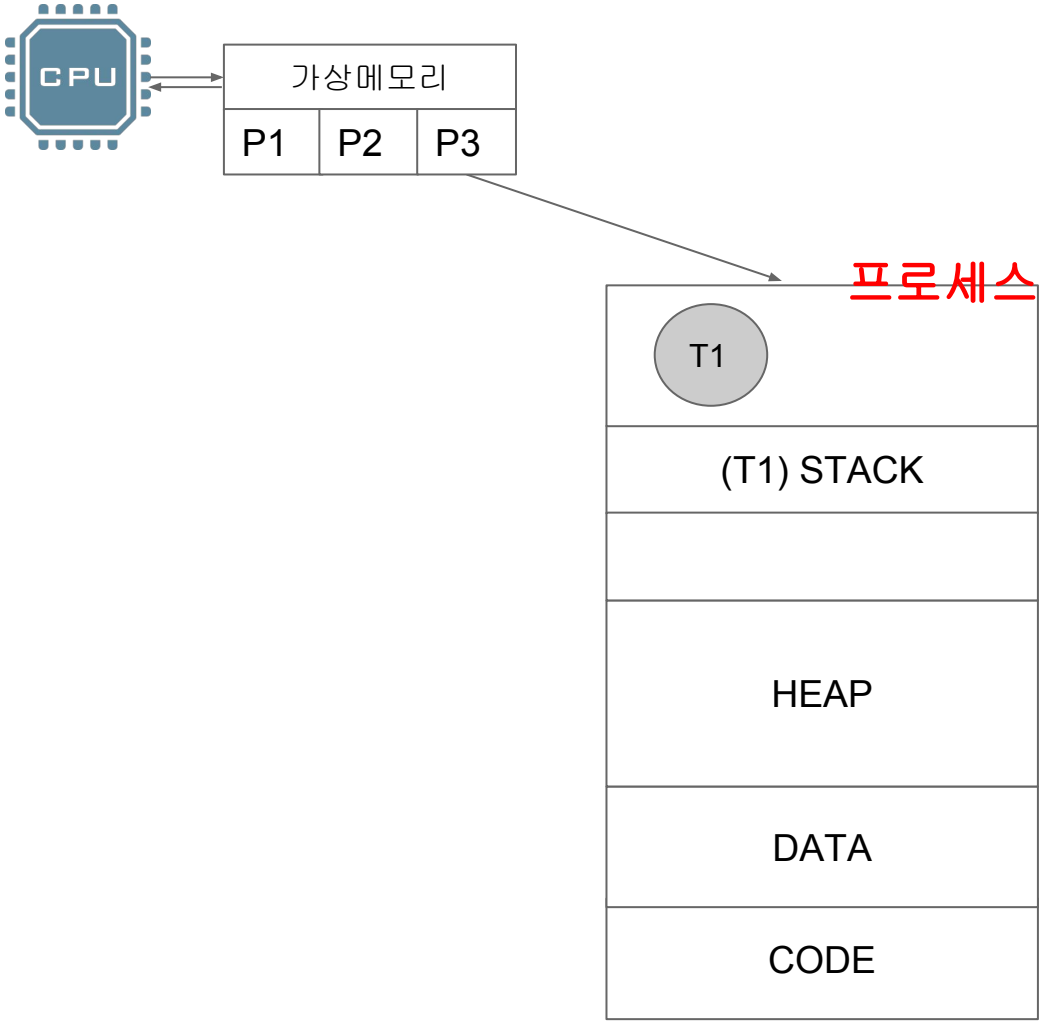


가상메모리는
프로세스간
주소공간을
논리적으로 차단

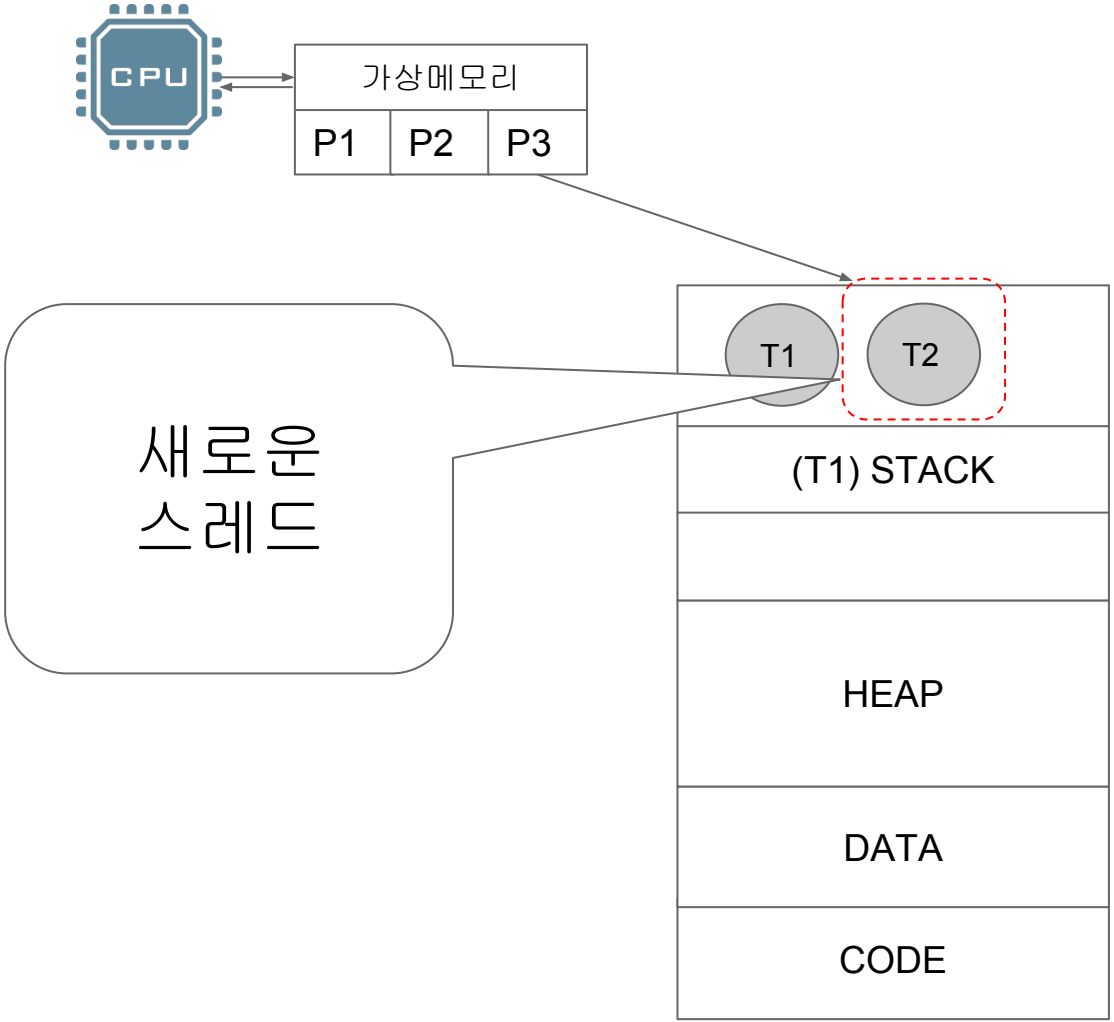
Thread 일반


CPU
가상메모리
P1
P2
P3
프로세스
T1
(T1) STACK
HEAP
DATA
CODE
프로세스
T1
(T1) STACK
HEAP
DATA
CODE
멀티 프로세스

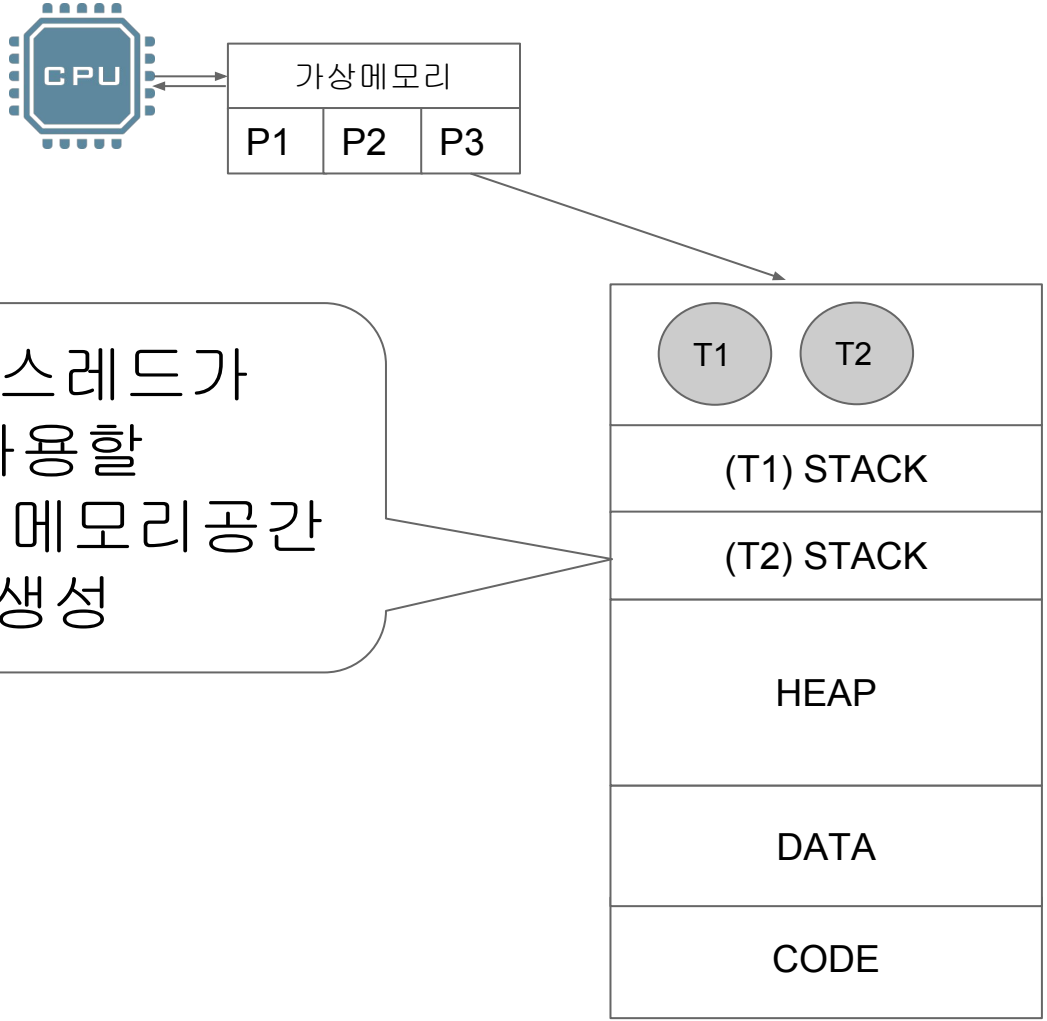
Thread 일반



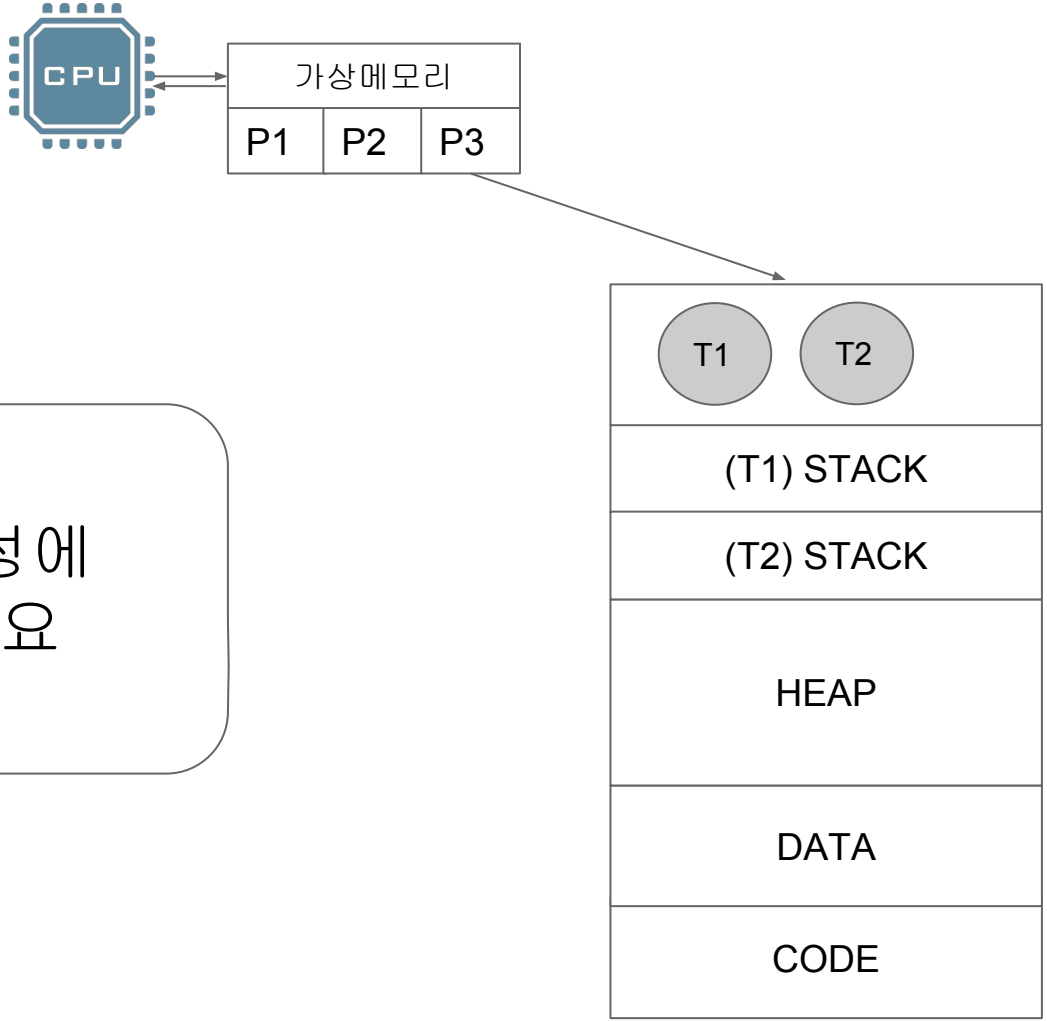
Thread 일반



Thread 일반


CPU
가상메모리
P1
P2
P3
신규 스레드가
사용할
STACK 메모리공간
생성
T1
T2
(T1) STACK
(T2) STACK
HEAP
DATA
CODE

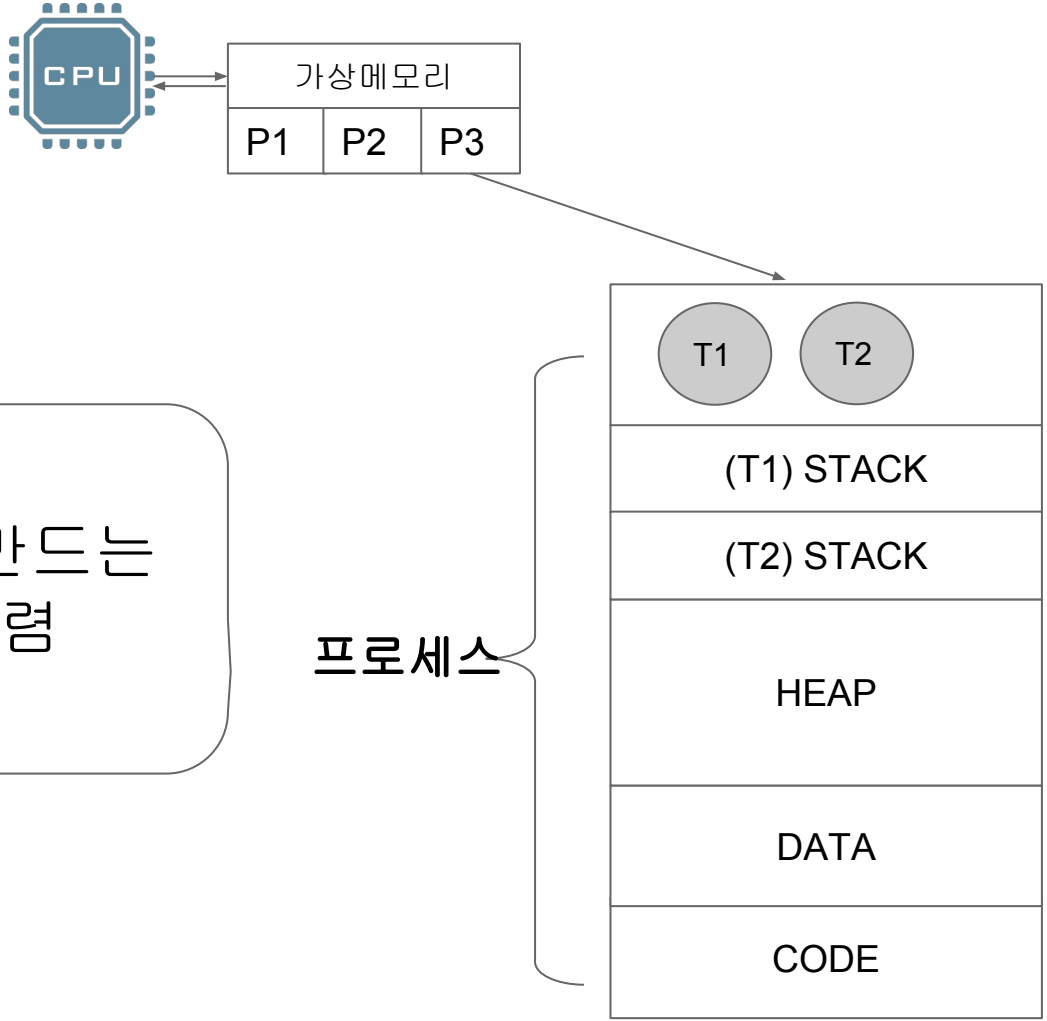
Thread 일반



스레드 장점

스레드 생성에
비용이 필요

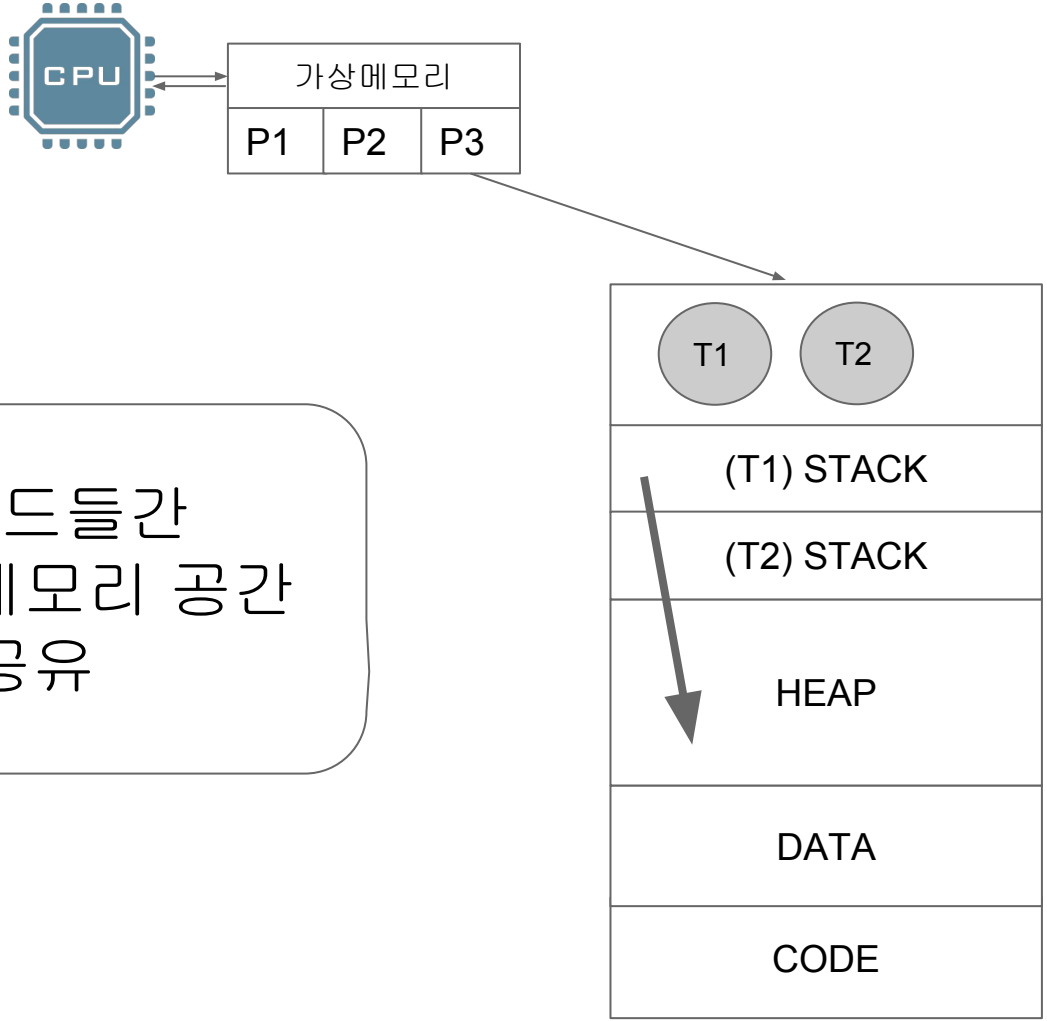
Thread 일반



스레드 장점

프로세스를 만드는 것보다 저렴

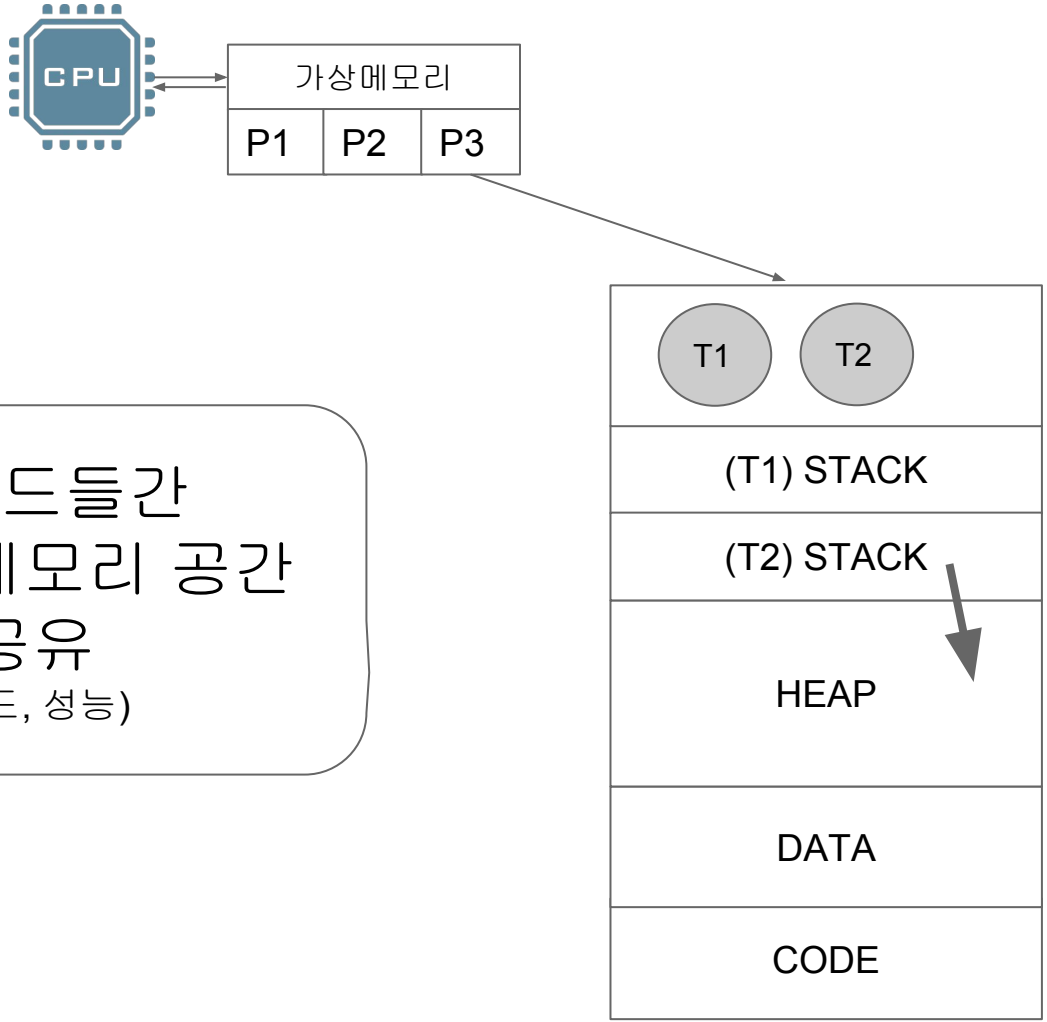
Thread 일반



스레드 장점

스레드들간
HEAP 메모리 공간
공유

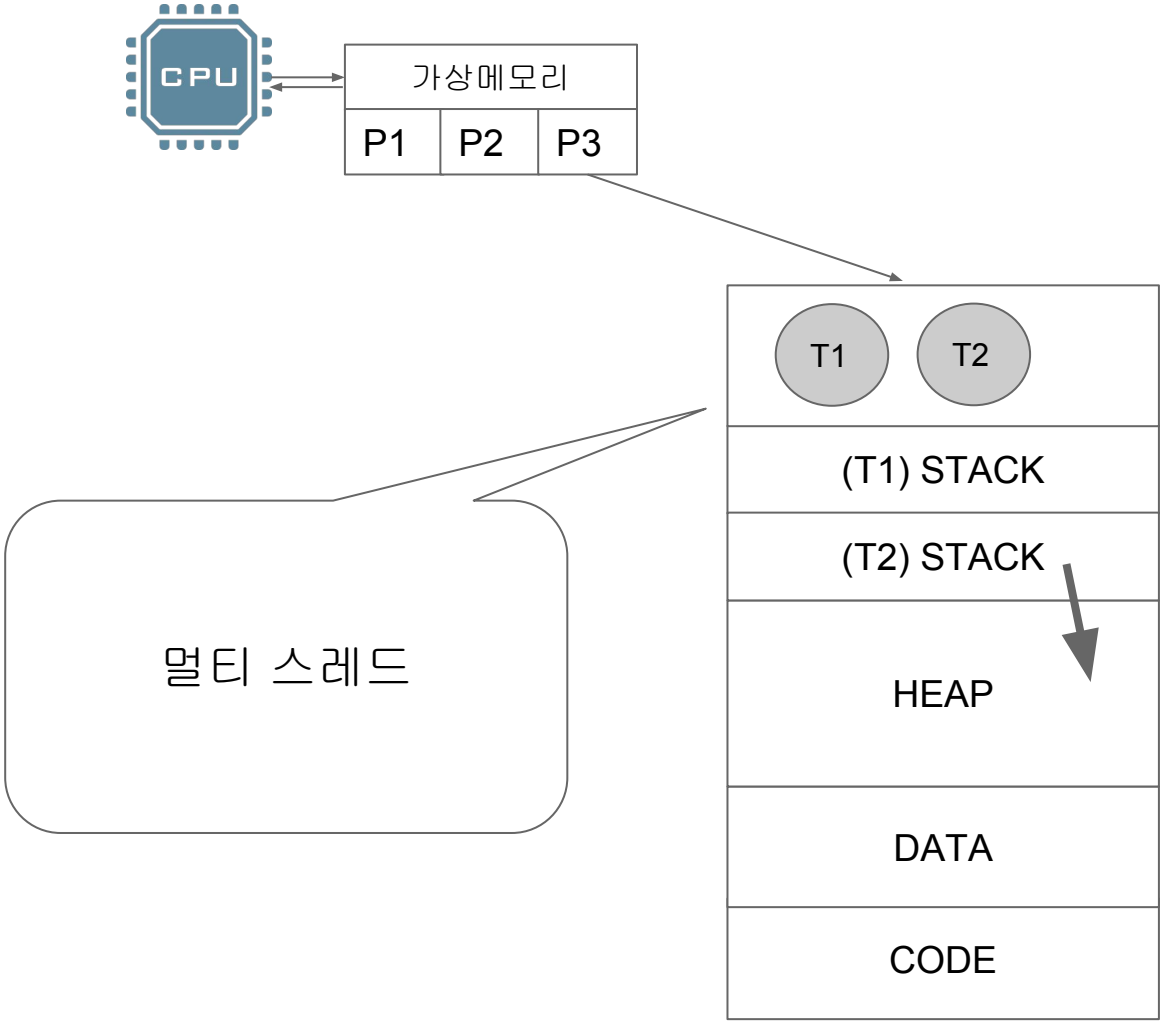
Thread 일반



스레드 장점

스레드들간
HEAP 메모리 공간
공유
(코드, 성능)

Thread 일반

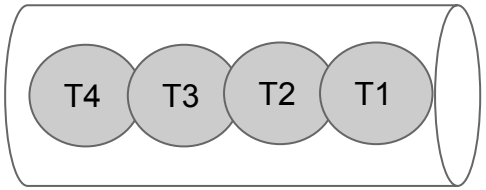


Thread 일반

Context Switch

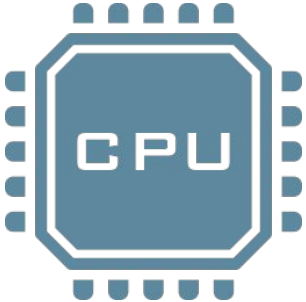
- 스케줄러에의해 스레드는 time slice만큼 CPU를 사용한 후 작업을 멈추고 CPU를 다음 스레드가 점유하는 것
  - 과정
    - 저장 - 반납되는 스레드는 마지막 상태를 메모리에 저장
    - 복원 - 실행되는 스레드는 메모리에서 저장된 상태를 복구

Thread 일반



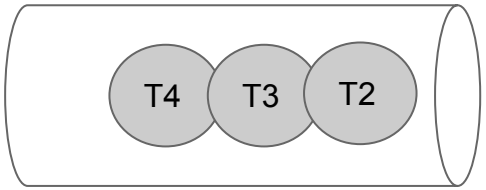
스레드큐

스케줄러



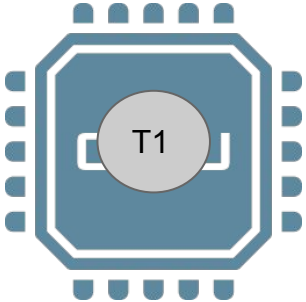
Thread 일반

Thread 일반

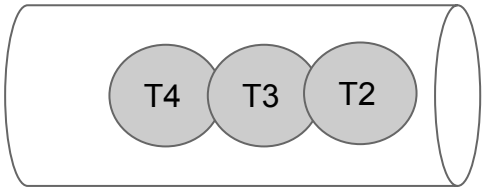


스레드큐

스케줄러

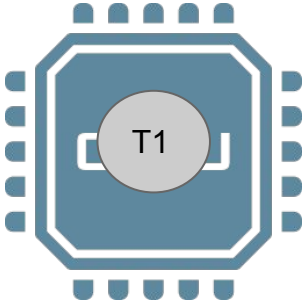


Thread 일반

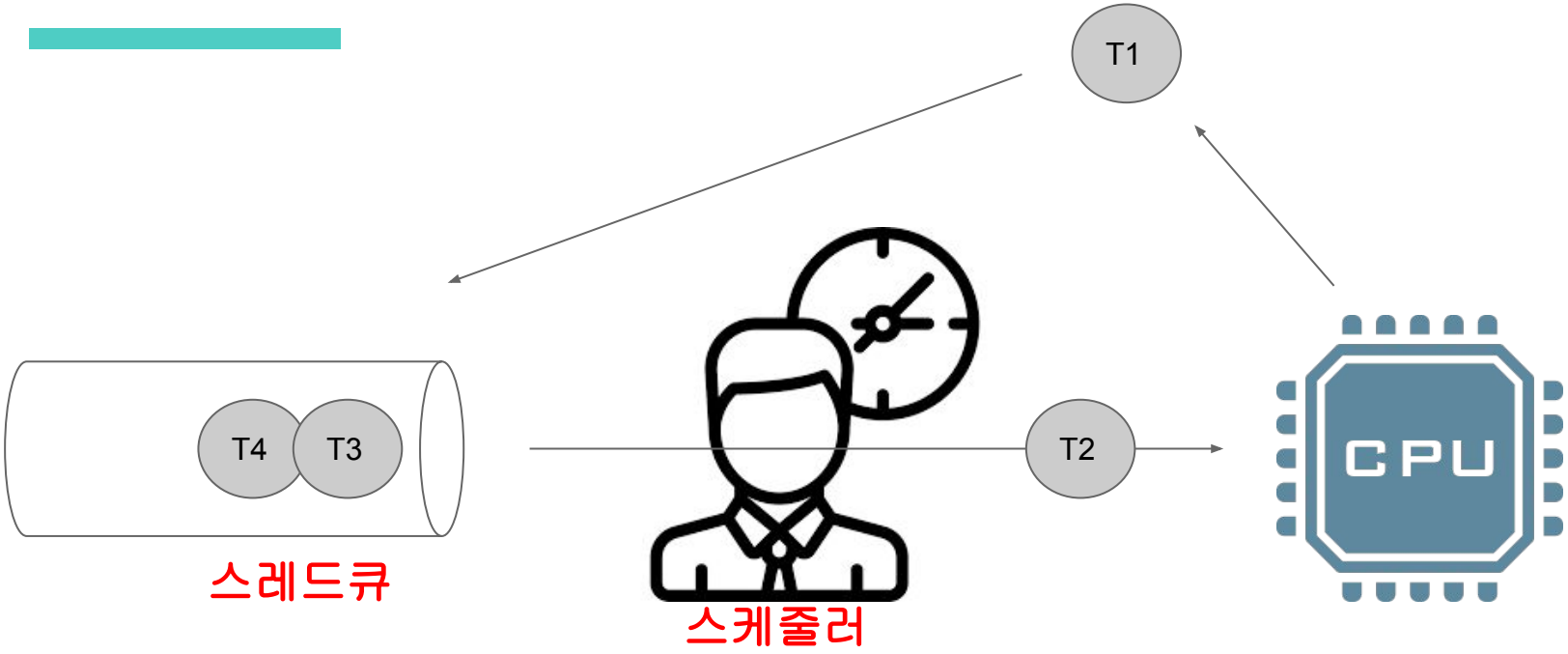


스레드큐

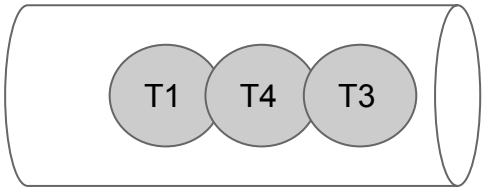
스케줄러



Thread 일반

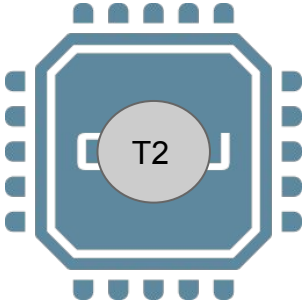


Thread 일반



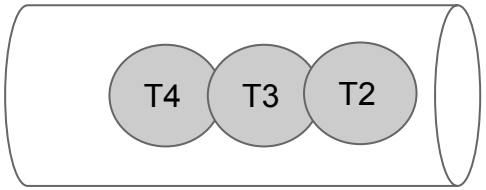
스레드큐

스케줄러



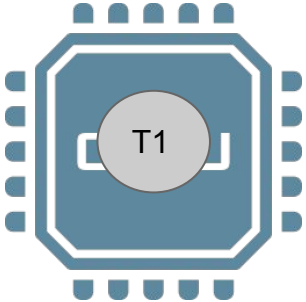
context switch
오버헤드?

Thread 일반

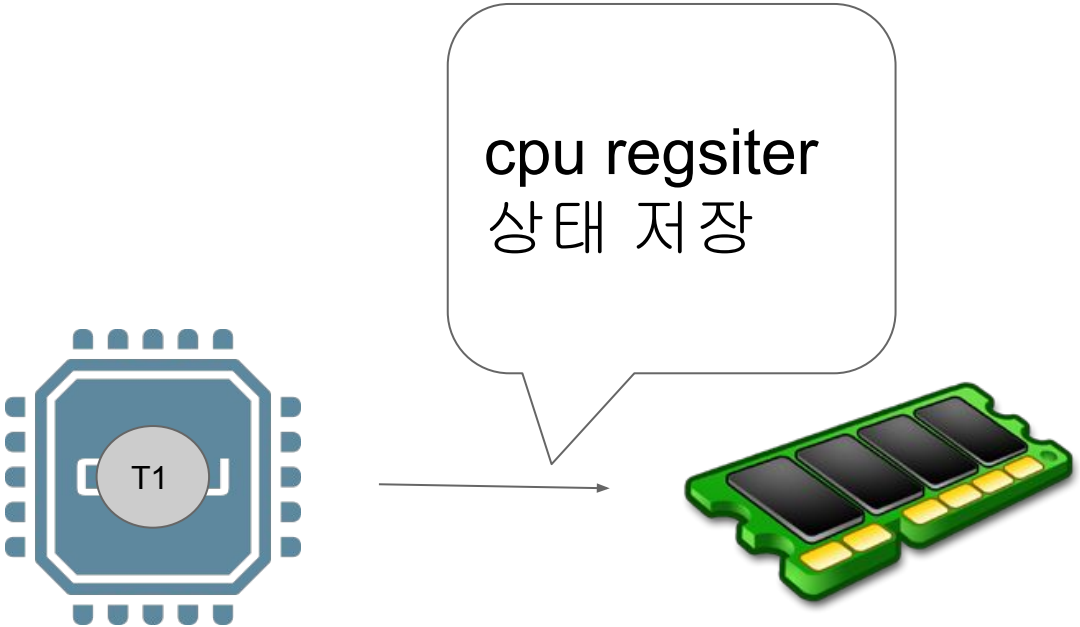


스레드큐

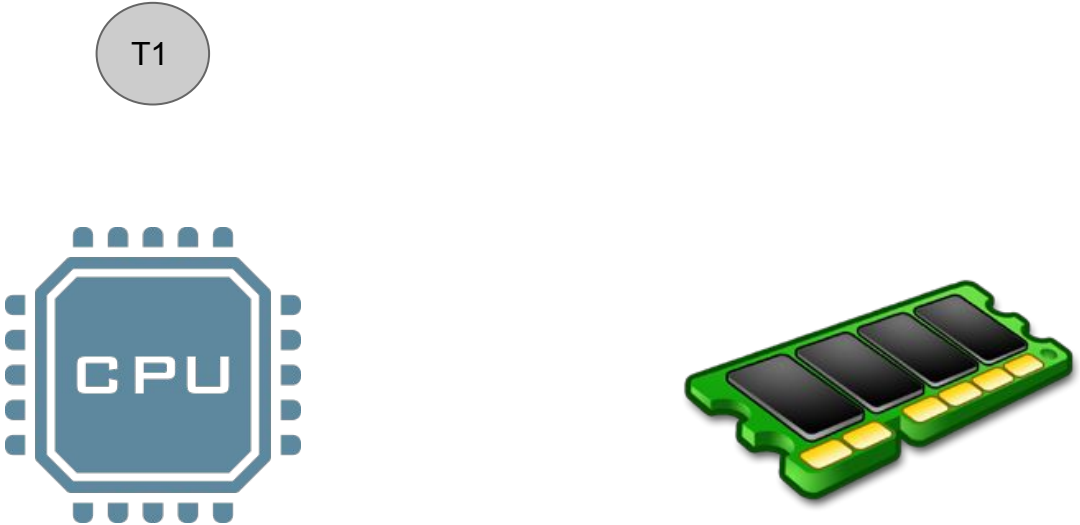
스케줄러



Thread 일반

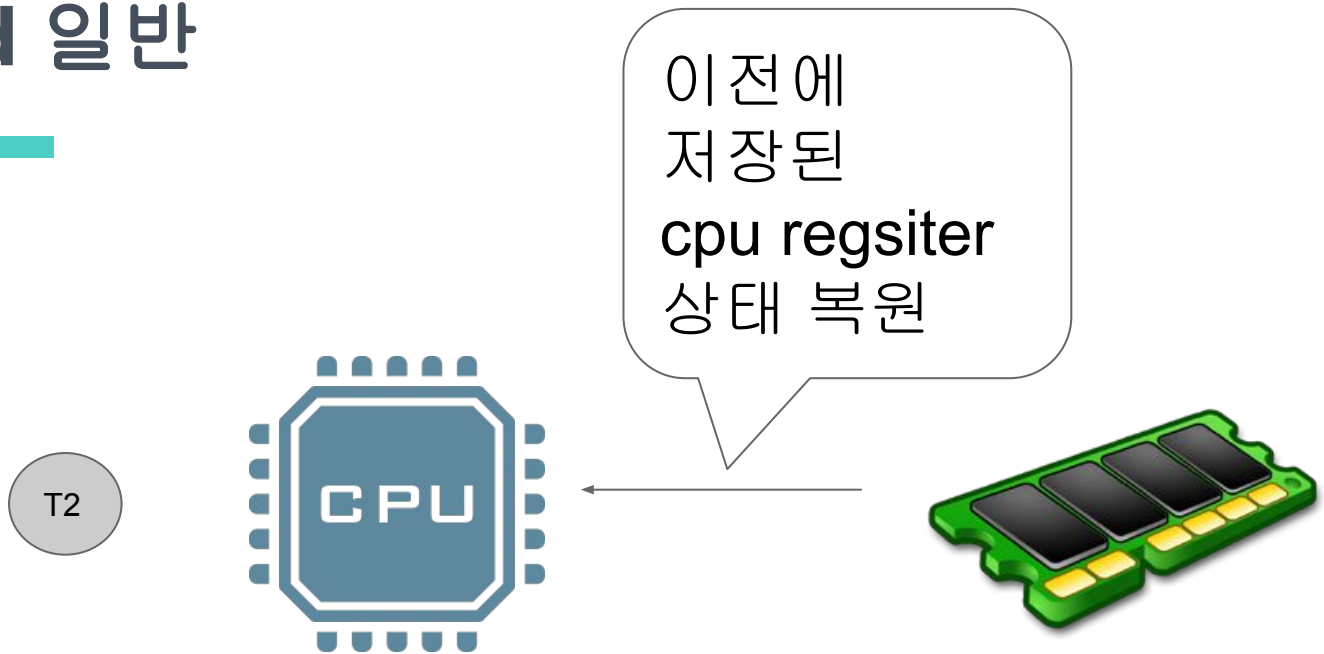


Thread 일반

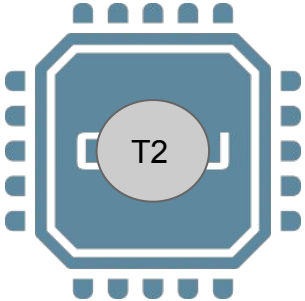


Thread 일반

Thread 일반



Thread 일반



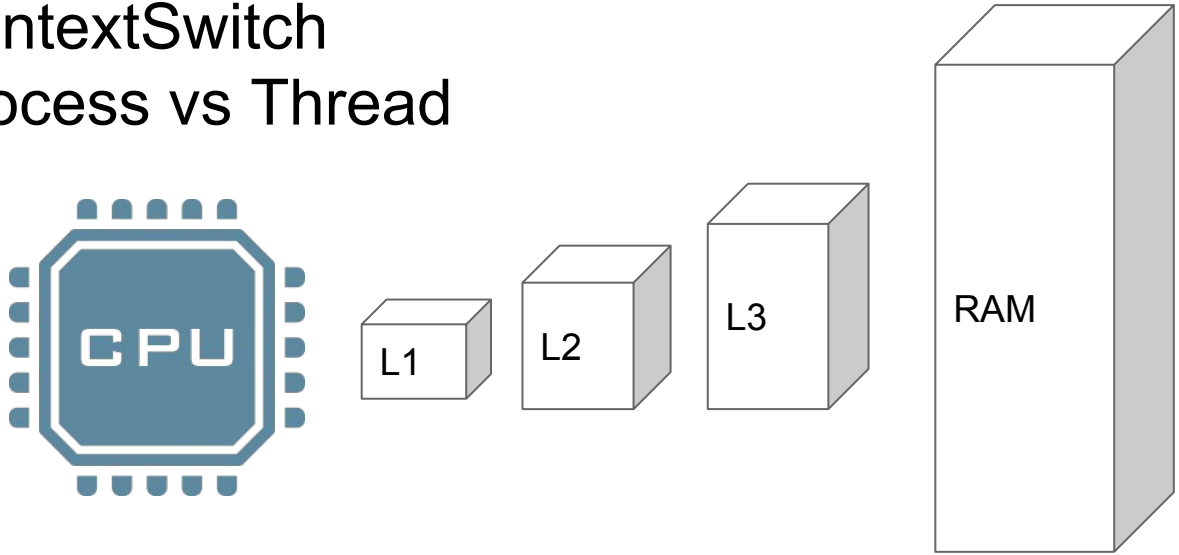
ContextSwitch 비용

멀티 프로세스 vs 멀티 스레드

(캐시...)

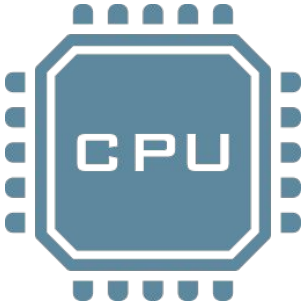
Thread 일반

ContextSwitch
Process vs Thread



Thread 일반

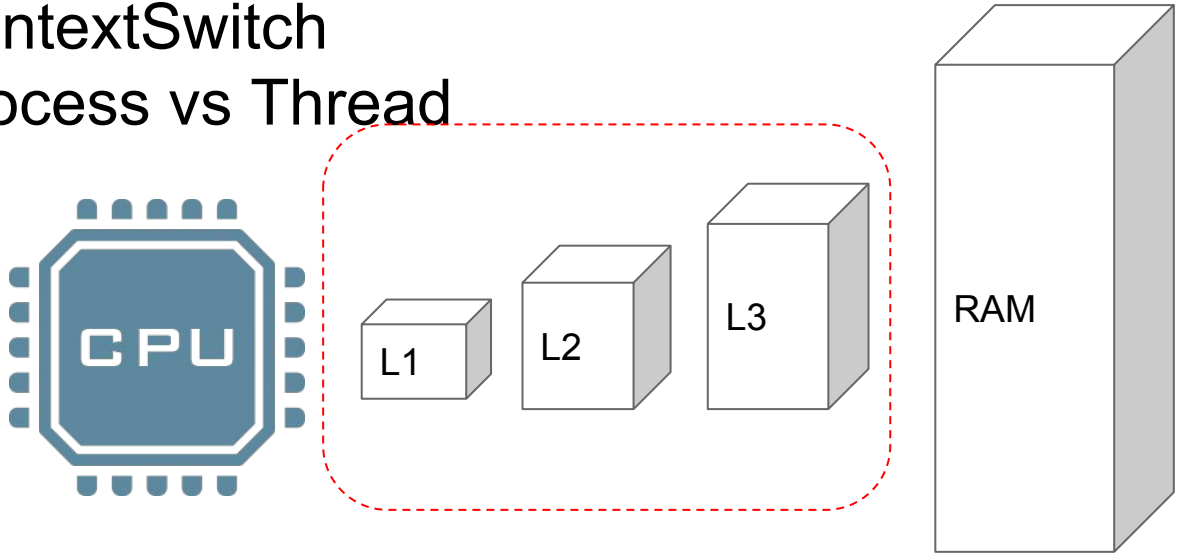
ContextSwitch
Process vs Threa



1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min

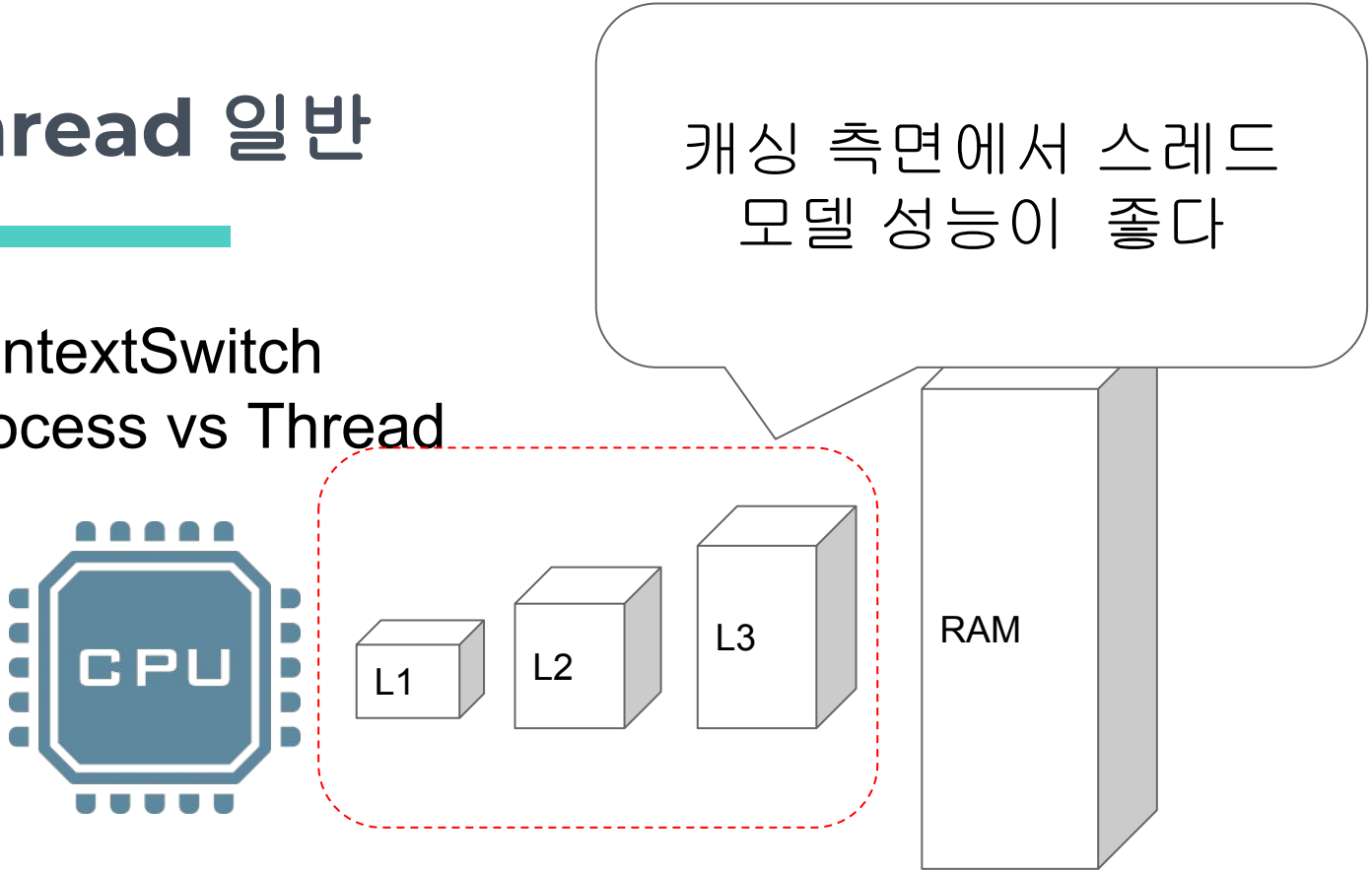
Thread 일반

ContextSwitch
Process vs Thread



Thread 일반

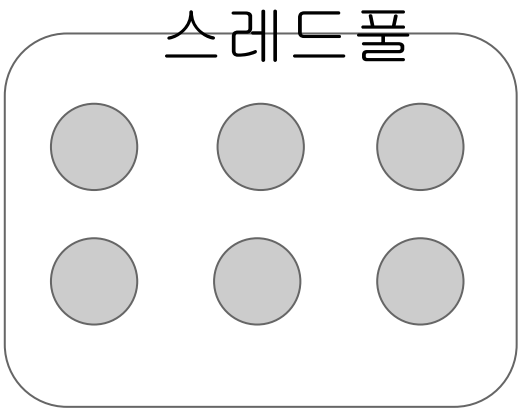
ContextSwitch
Process vs Thread



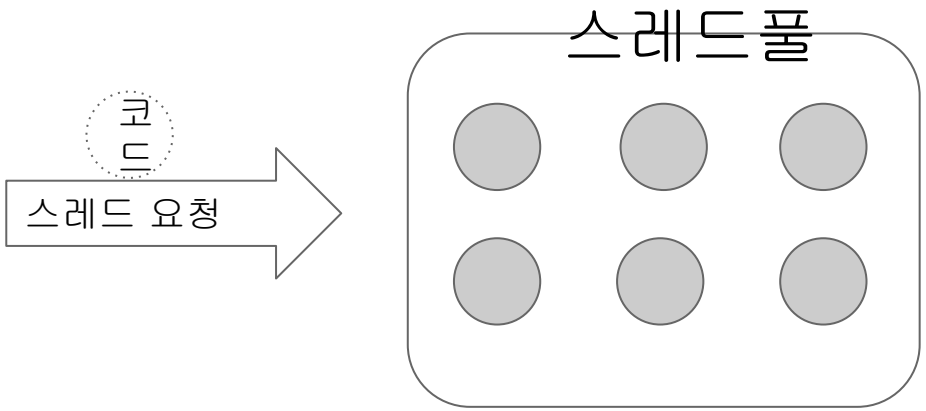
Thread 일반

- ThreadPool
  - 스레드를 미리 만들어 놓고 재사용하는 것
- 비용 감소
  - 스레드 생성 비용 - cpu, memroy

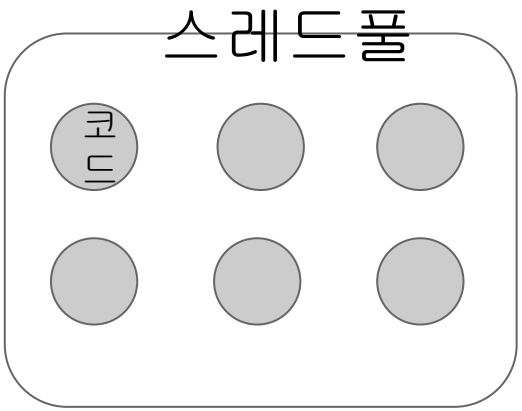
Thread 일반



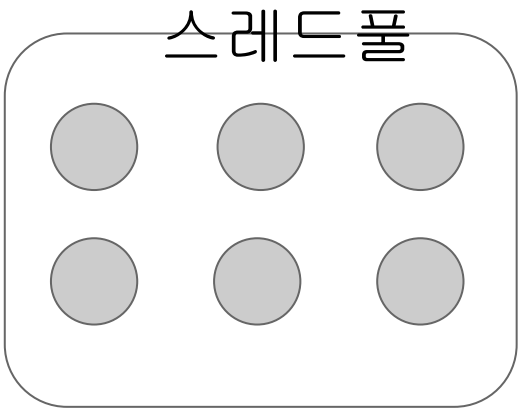
Thread 일반



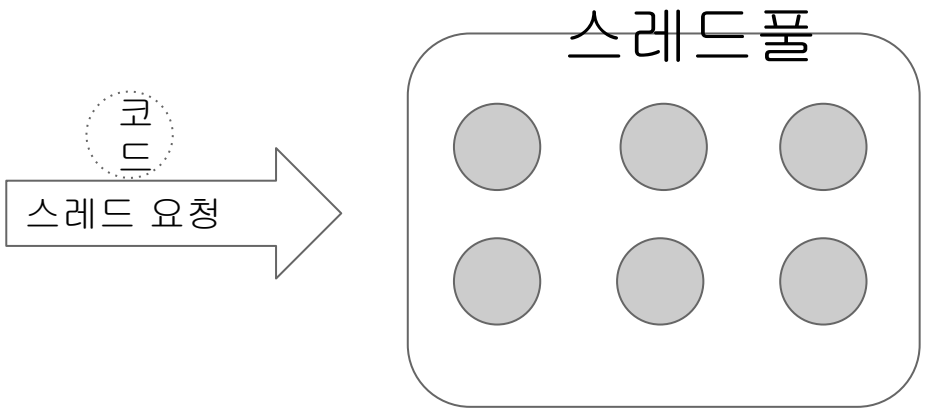
Thread 일반



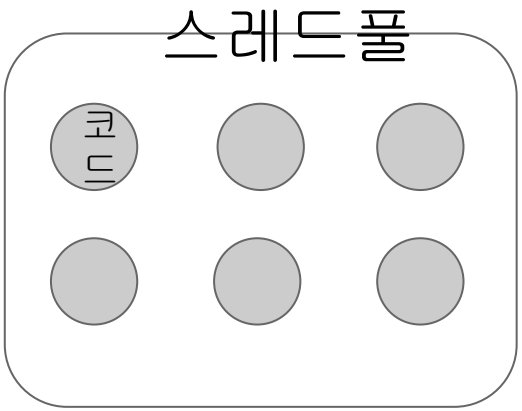
Thread 일반



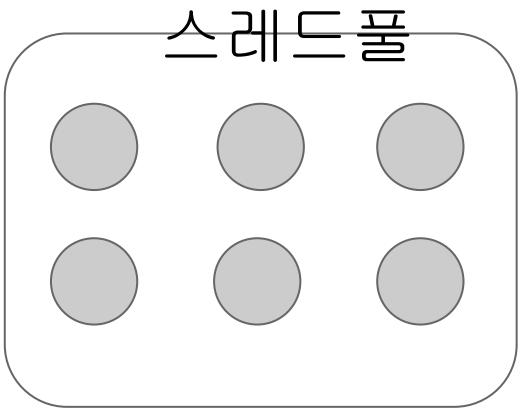
Thread 일반



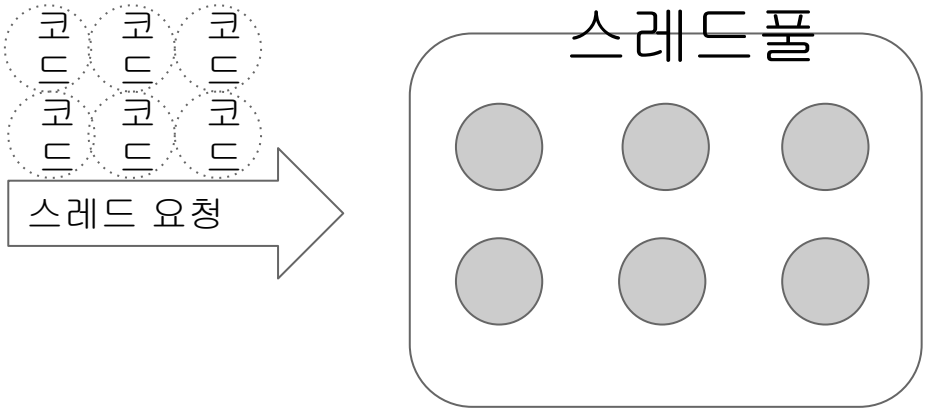
Thread 일반



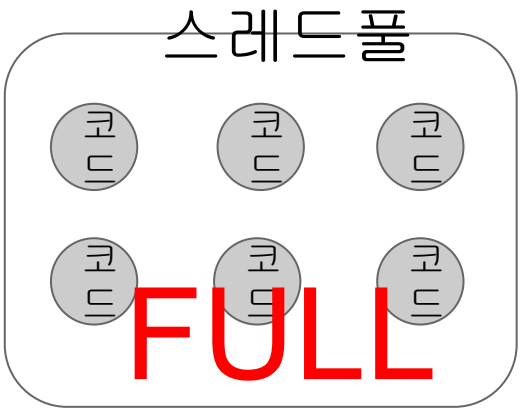
Thread 일반



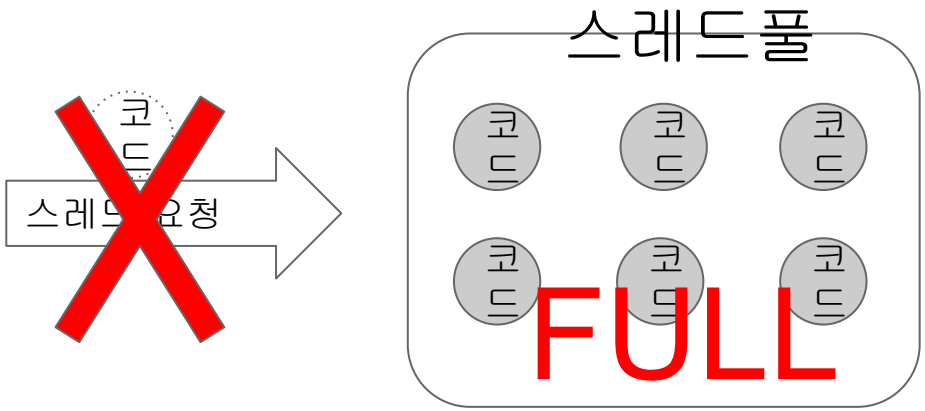
Thread 일반



Thread 일반

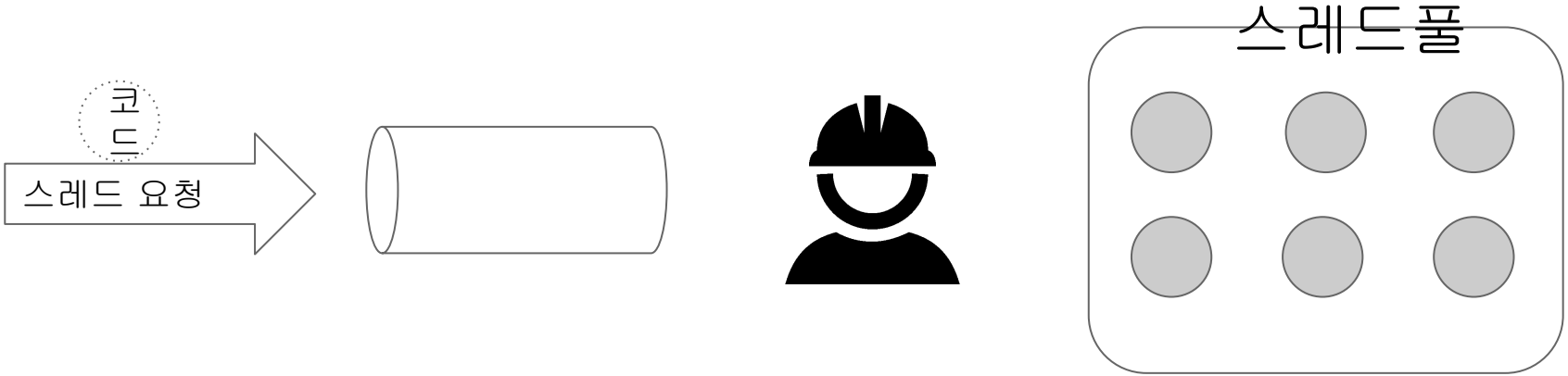


Thread 일반

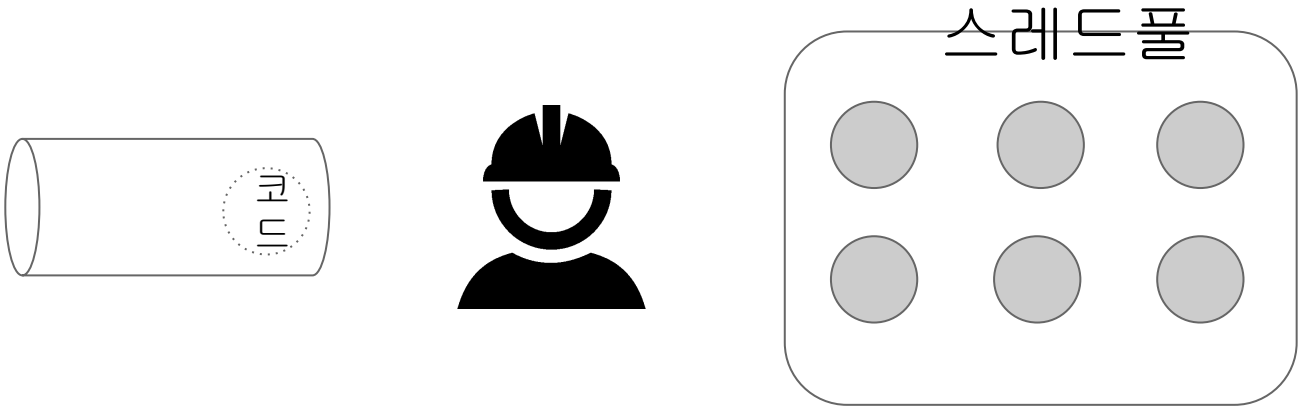


Thread 일반

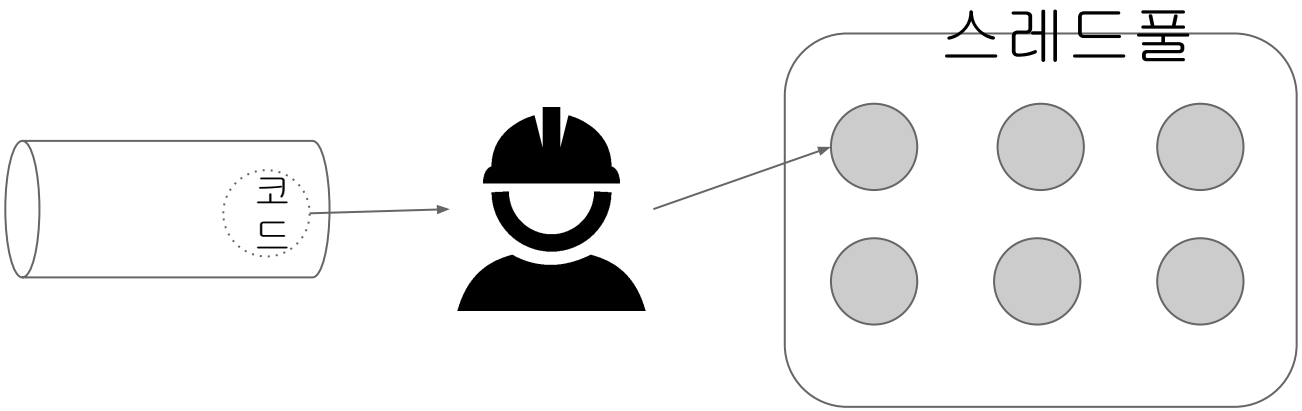
Thread 일반



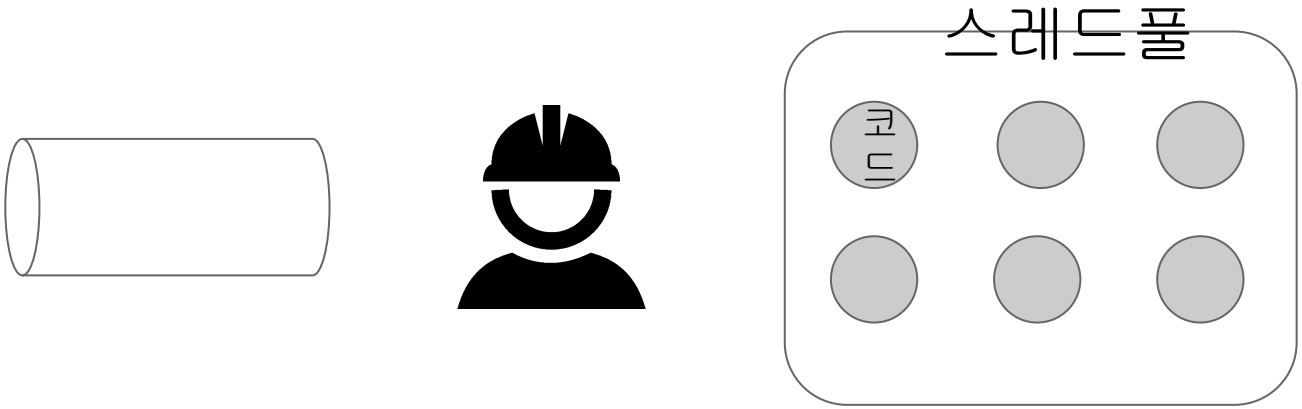
Thread 일반



Thread 일반

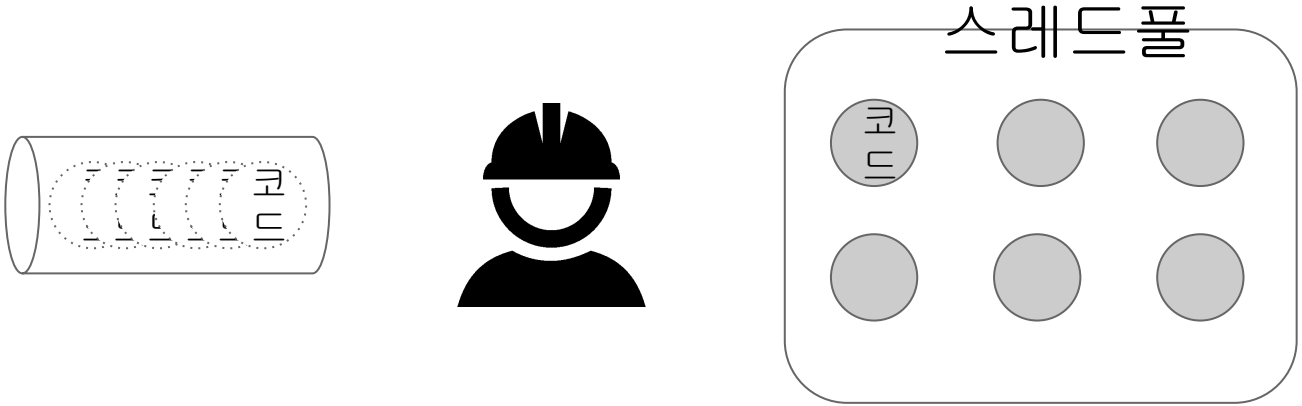


Thread 일반

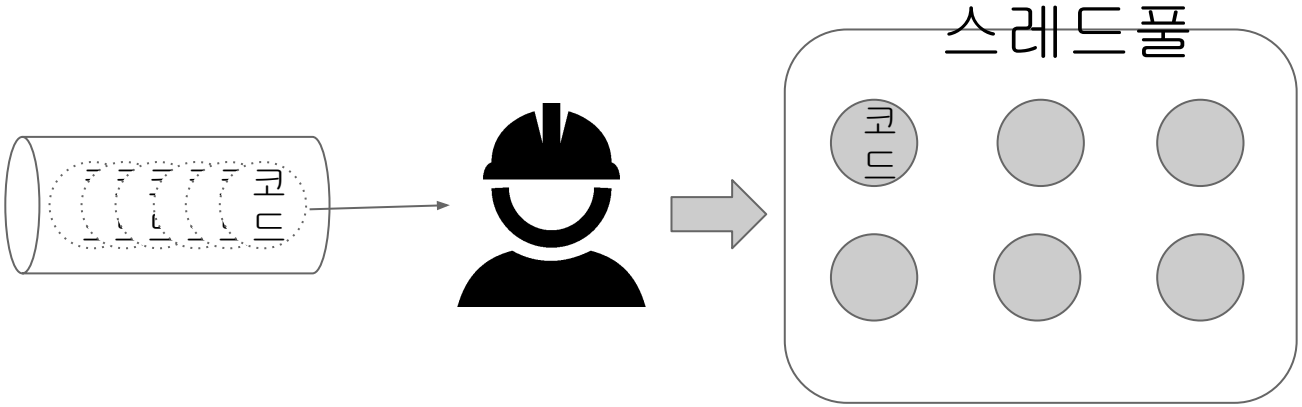


Thread 일반

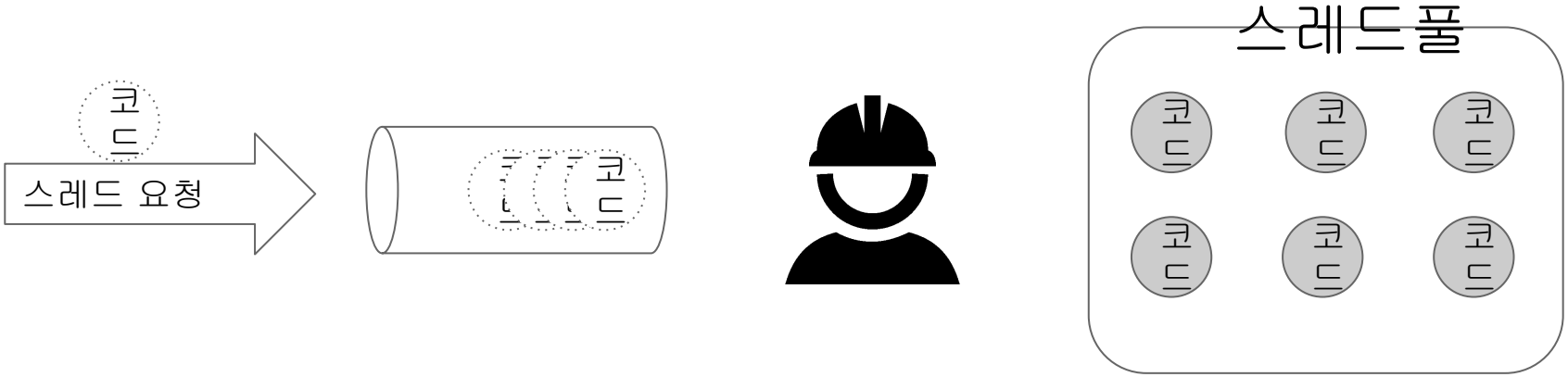
Thread 일반



Thread 일반



Thread 일반



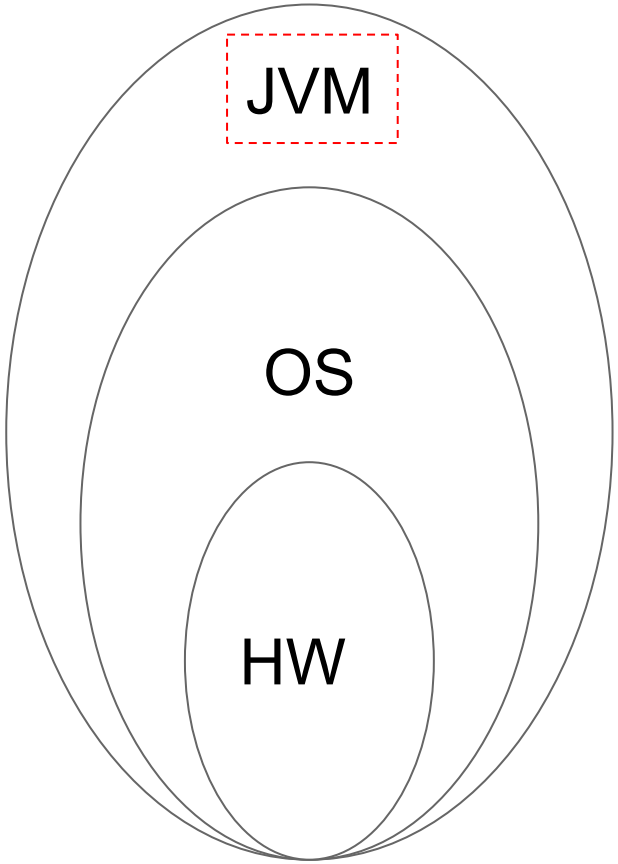
Thread in JVM

Thread in JVM

Java Virtual Machine

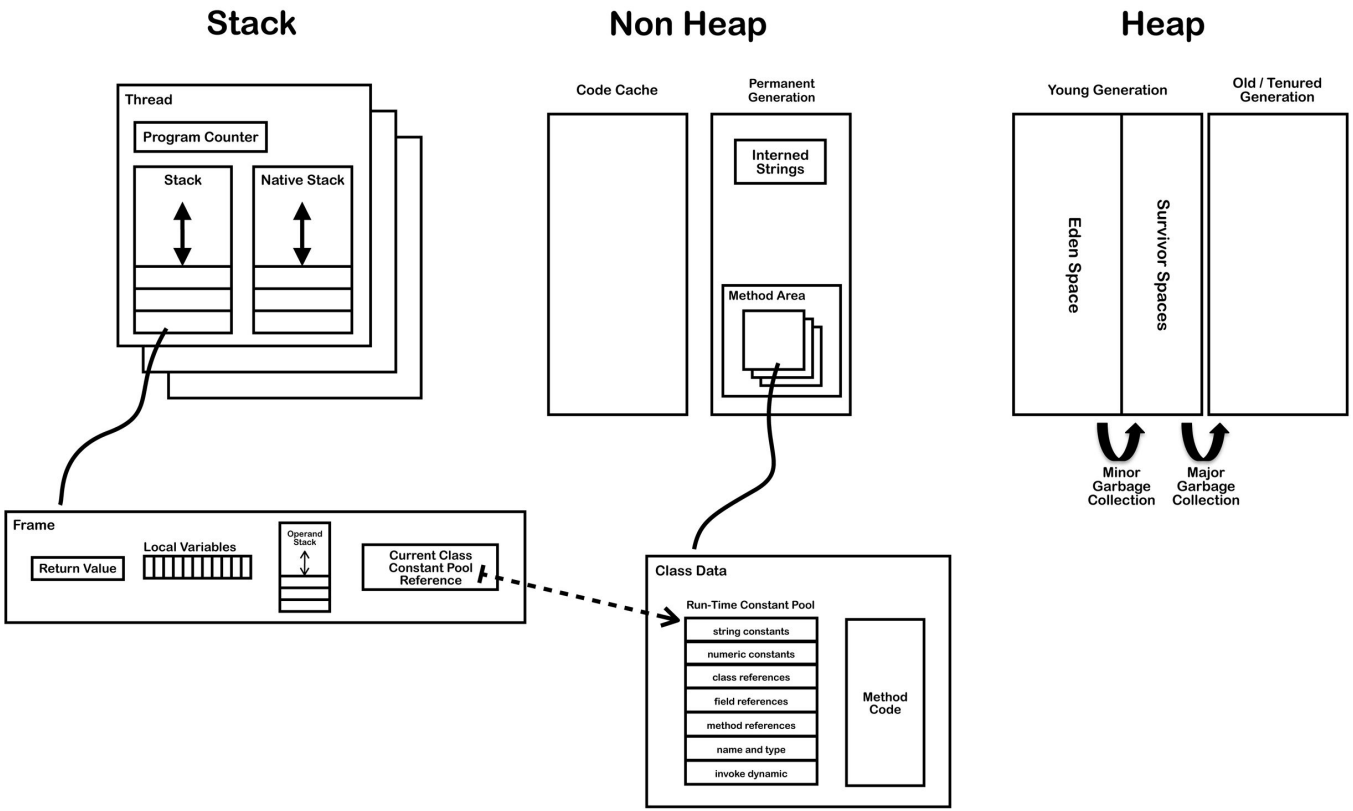
Thread in JVM

Java Virtual Machine



Thread in JVM

JVM internal



Thread in JVM

- JVM 스레드
  - JVM은 OS의 Thread를 사용
  - 가상머신이 만든 구조에서 동작해야하기 때문에 OS의 thread를 jvm이 사용할 수 있게 추상화됨
    - 레이어가 하나 더 있기 때문에 native thread보다 비용이 더 발생

Thread in JVM

- JVM 스레드 비용
  - Thread 생성비용(OS + JVM)
  - Context switching
  - Garbage collection

성능 최적화를 생각한다면 OS thread를 직접 사용할 때보다 더 많이 신경 써야한다.

2.

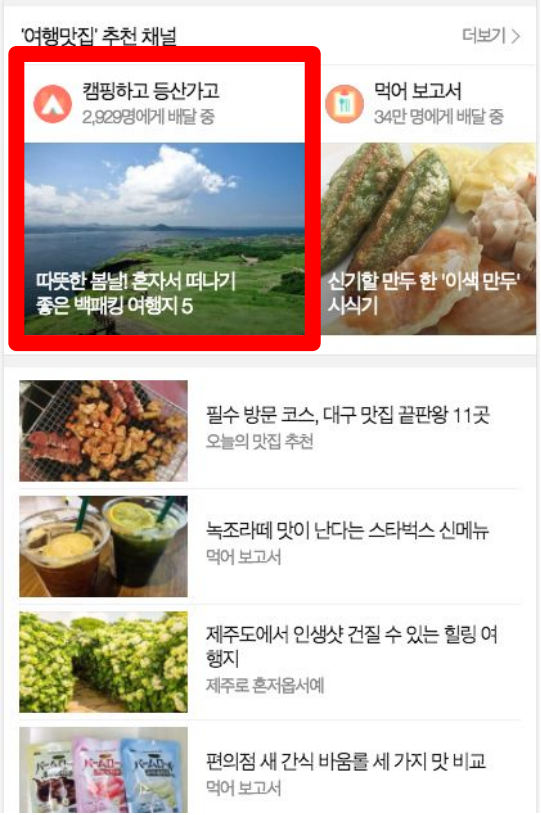
Java Async API

비동기가 필요한 상황
Java 비동기 API
느낀점

비동기가 필요한 상황

- CPU expensive
  - cpu 자원을 많이 사용하는 코드들중 분할가능한 코드는 병렬화해서 시간 단축
- I/O blocking
  - HDD, network card등 I/O 자원은 느리다.
  - I/O 작업 요청한 스레드는 I/O 작업의 완료를 대기한다.
    - 대기하면서 스레드 자원 낭비

비동기가 필요한 상황


'여행맛집' 추천 채널
더보기 >
캠핑하고 등산가고
2,929명에게 배달 중
따뜻한 봄날! 혼자서 떠나기 좋은 백패킹 여행지 5
먹어 보고서
34만 명에게 배달 중
신기할 만두 한 '이색 만두' 시식기
필수 방문 코스, 대구 맛집 끝판왕 11곳
오늘의 맛집 추천
녹조라떼 맛이 난다는 스타벅스 신메뉴
먹어 보고서
제주도에서 인생샷 건질 수 있는 힐링 여행지
제주로 흔저옵서예
편의점 새 간식 바움롤 세 가지 맛 비교
먹어 보고서

채널

ㅋㄷㅋㄷ 인기

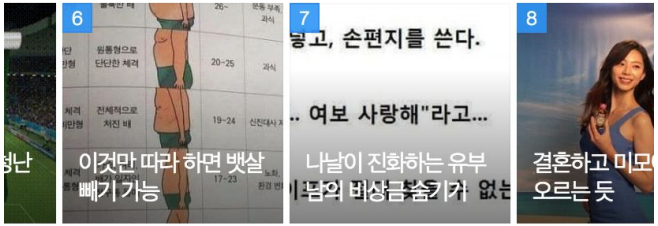
클릭하자마자 웃음 빵~

42,140명에게 배달 중 | 파트너 >

+ 채널 소식 배달 받기 | 공유

읽을거리 | 같이 볼만한 채널

인기 Best



전체 5,536

사회에서 여러모로 인정받는 인재

NEW

읽을거리 | 같이 볼만한 채널

오늘 나의 운세

오늘 하루 나의 운세는 어떨까? 로또 구매전 필독!

+ 9,467명과 배달 받기

이번 주 나의 운세

신통방통한 이번 주 운세 보고 가실게요!

+ 14,259명과 배달 받기

쿨내 진동 할리우드

너무 쿨한 할리우드 언니 오빠들의 세계

+ 3,213명과 배달 받기

인기채널 API

채널 구독 API

인기 컨텐츠 API

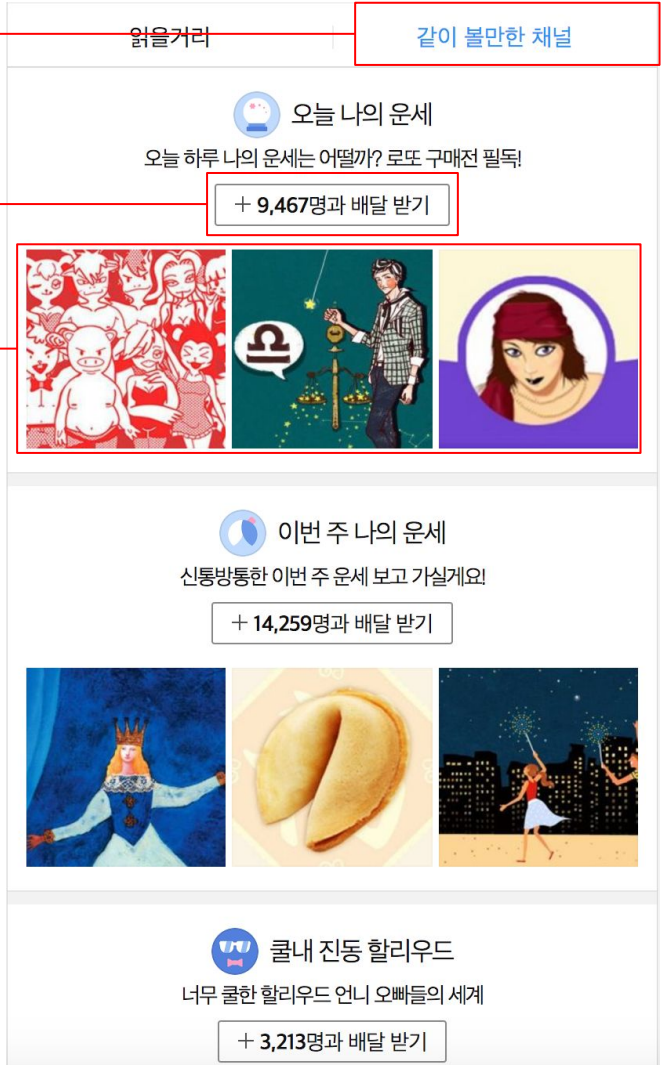
동영상 API

공감 API

연관 채널 API

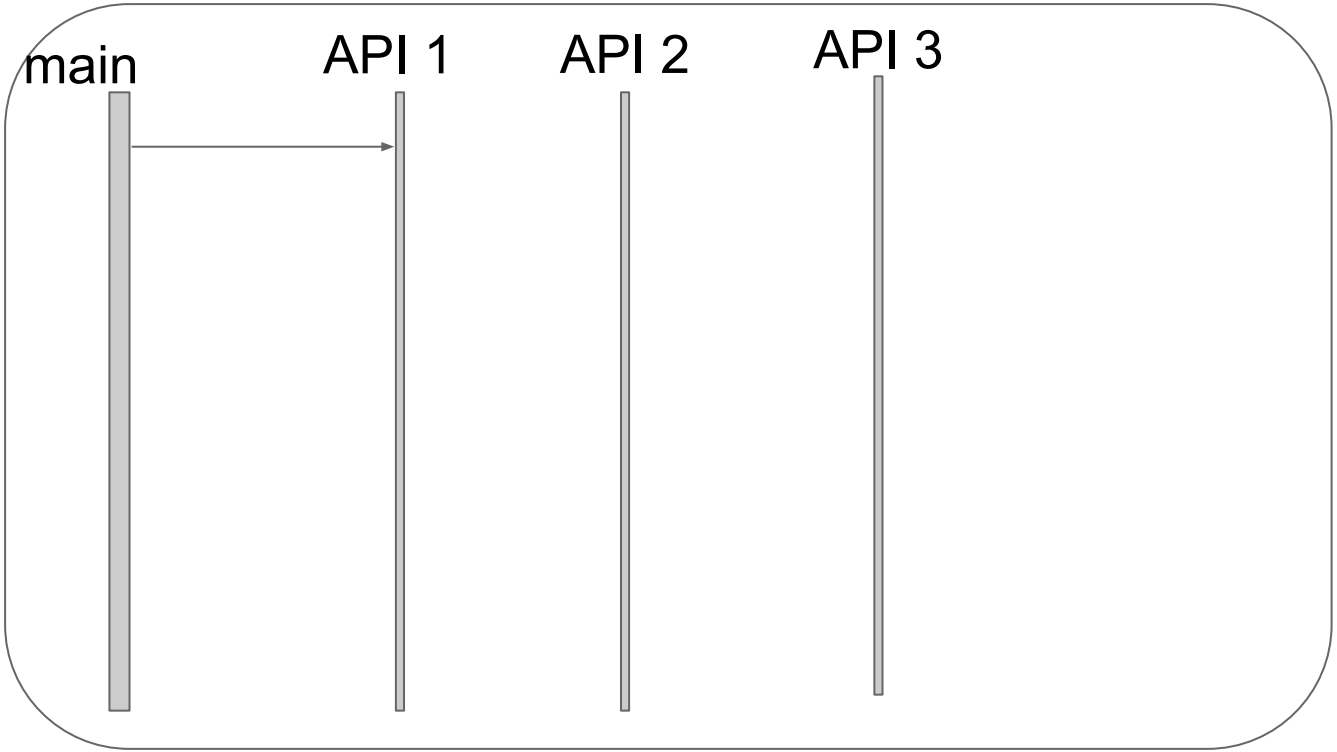
연관 채널 구독 API

인기 컨텐츠 API

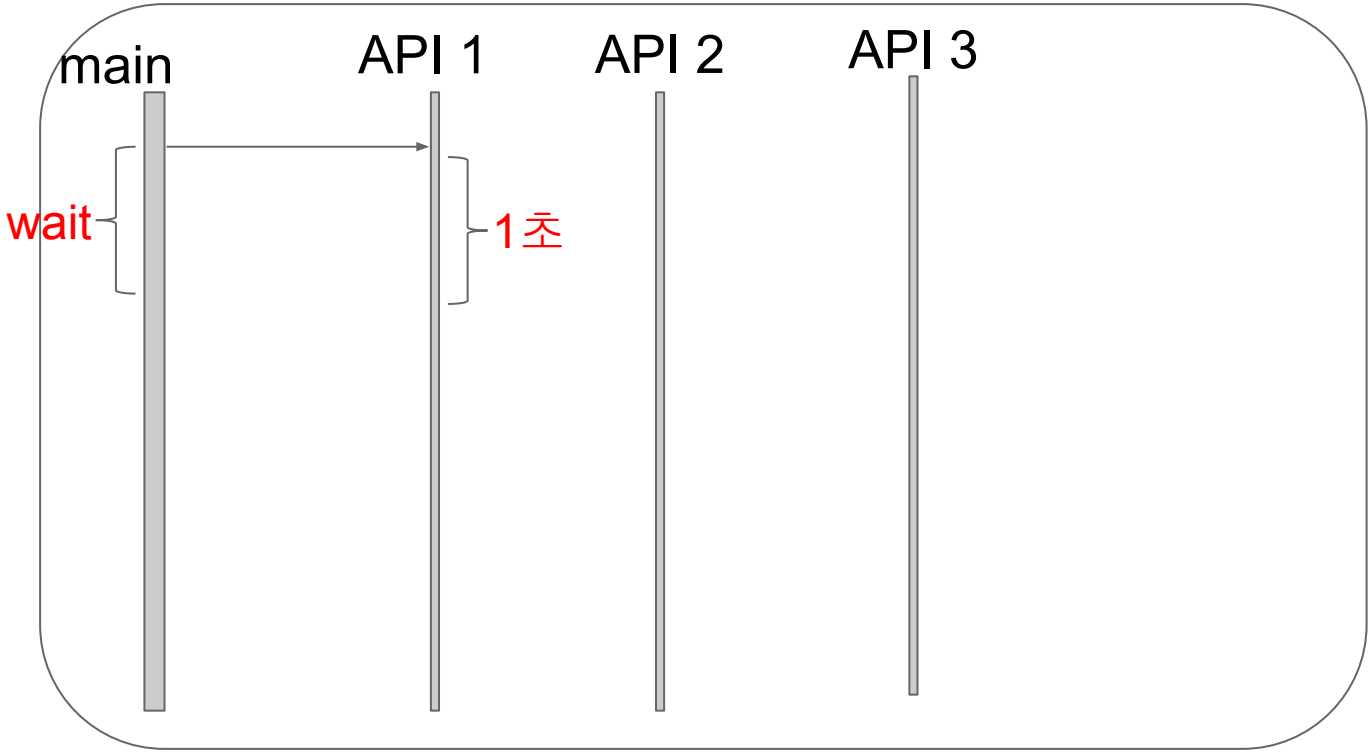


통신을 순차적으로 한다면?

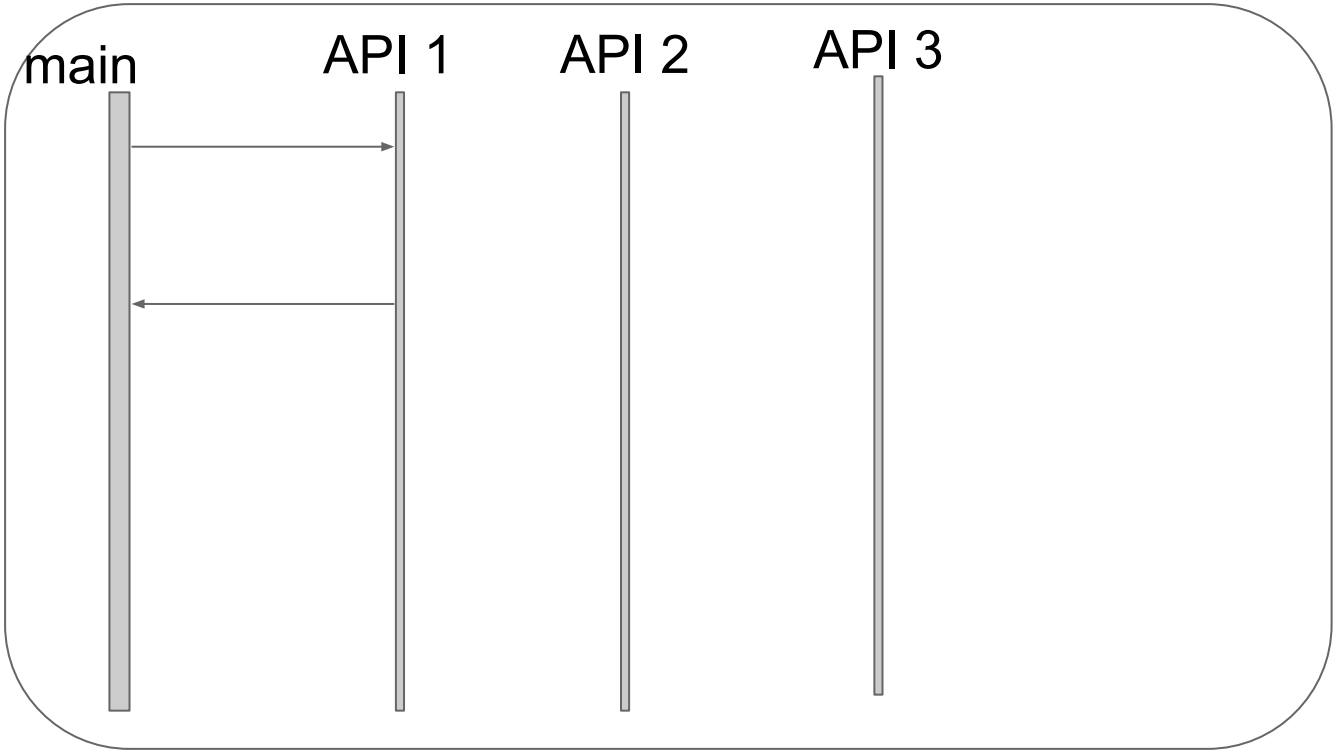
비동기가 필요한 상황



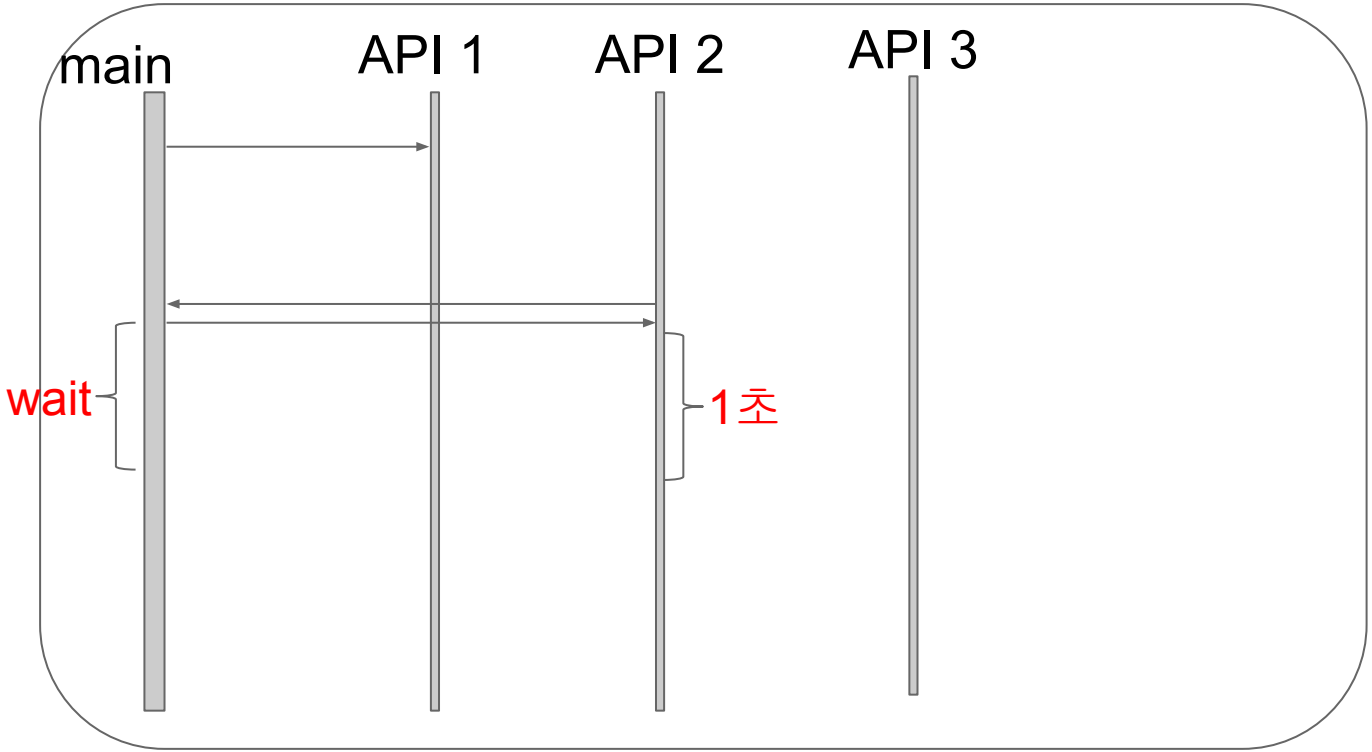
비동기가 필요한 상황



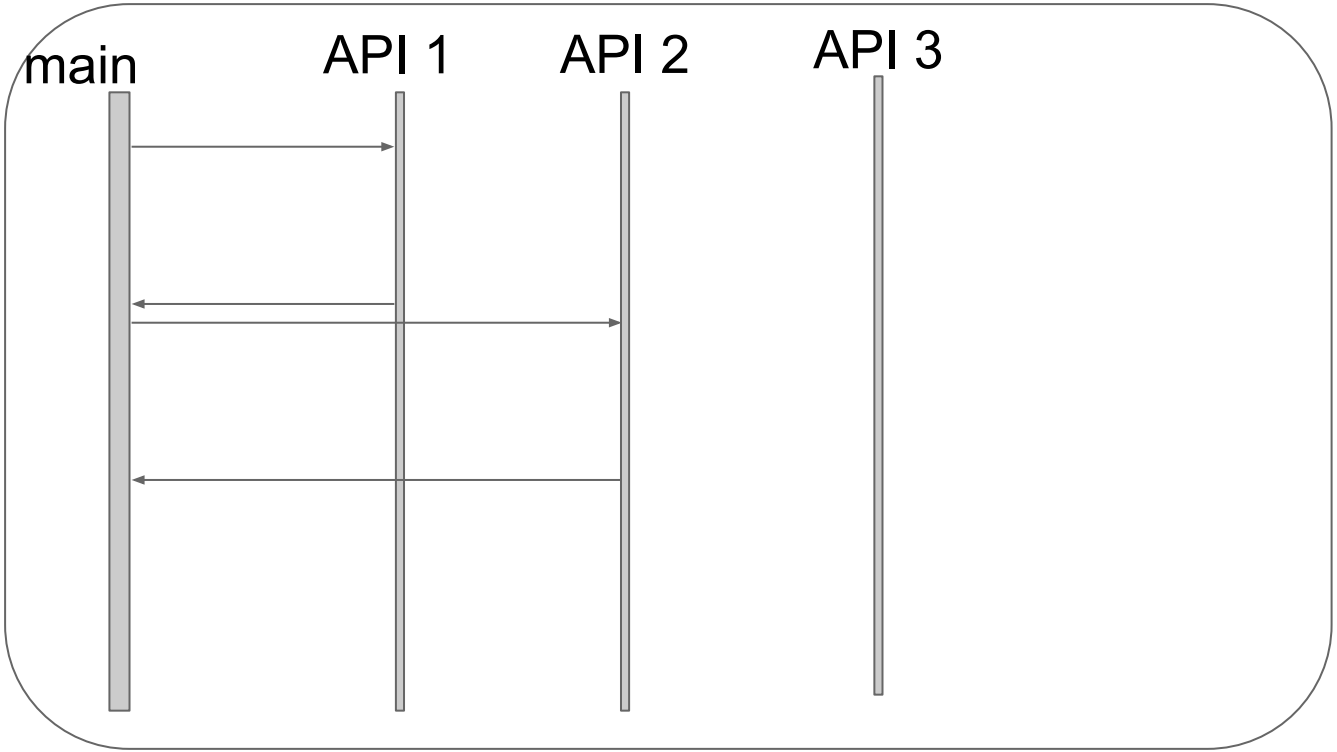
비동기가 필요한 상황



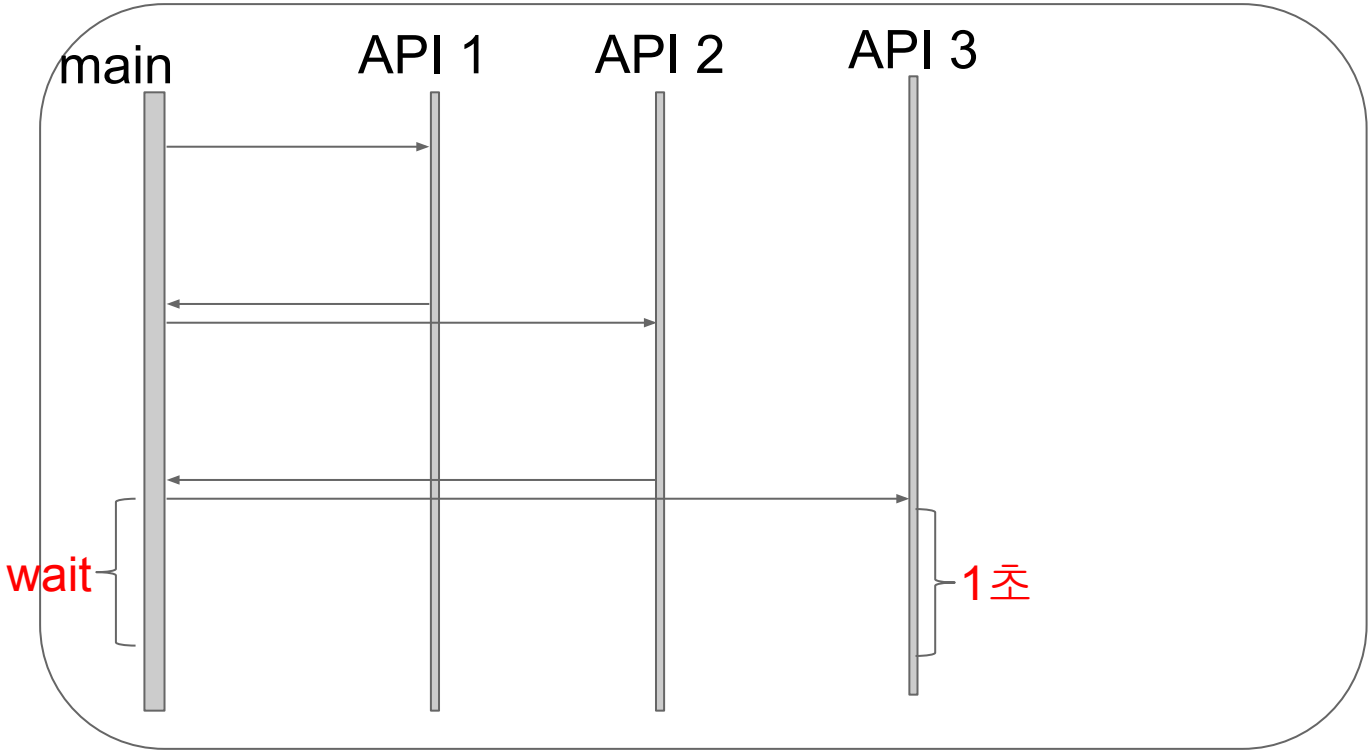
비동기가 필요한 상황



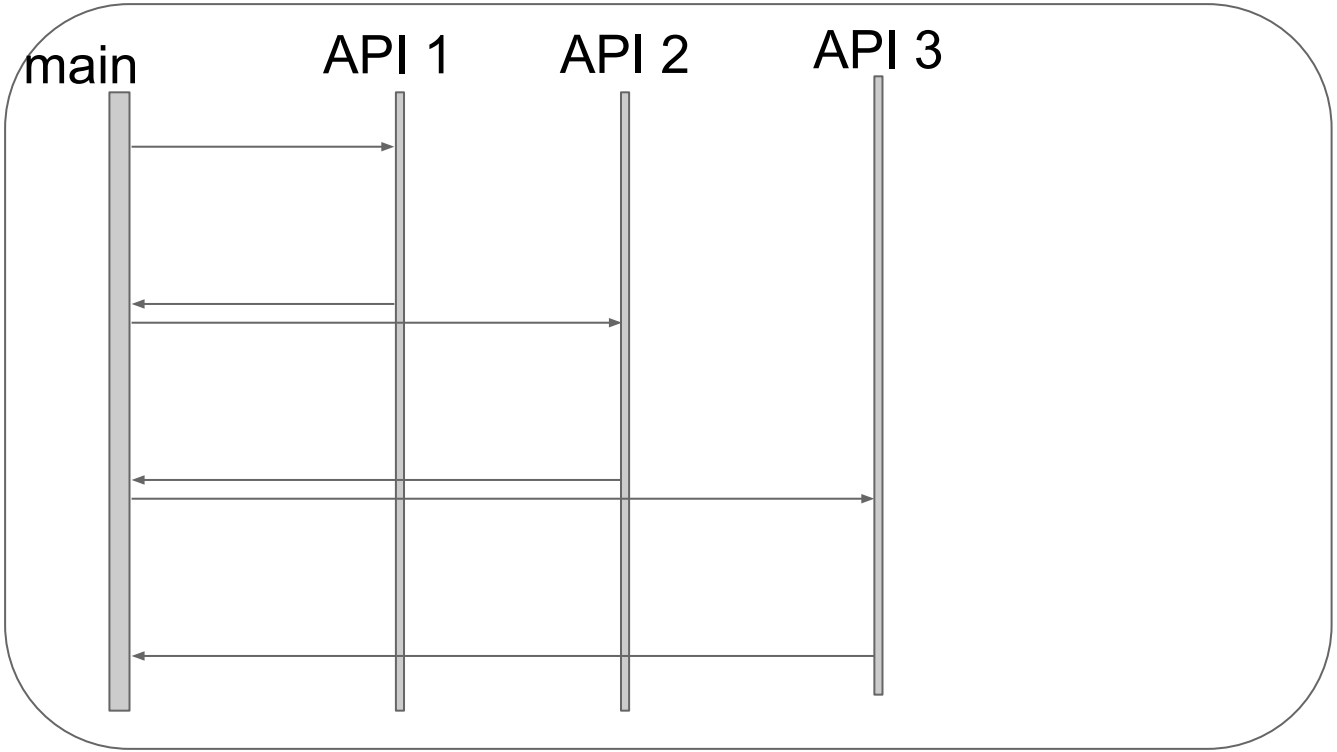
비동기가 필요한 상황



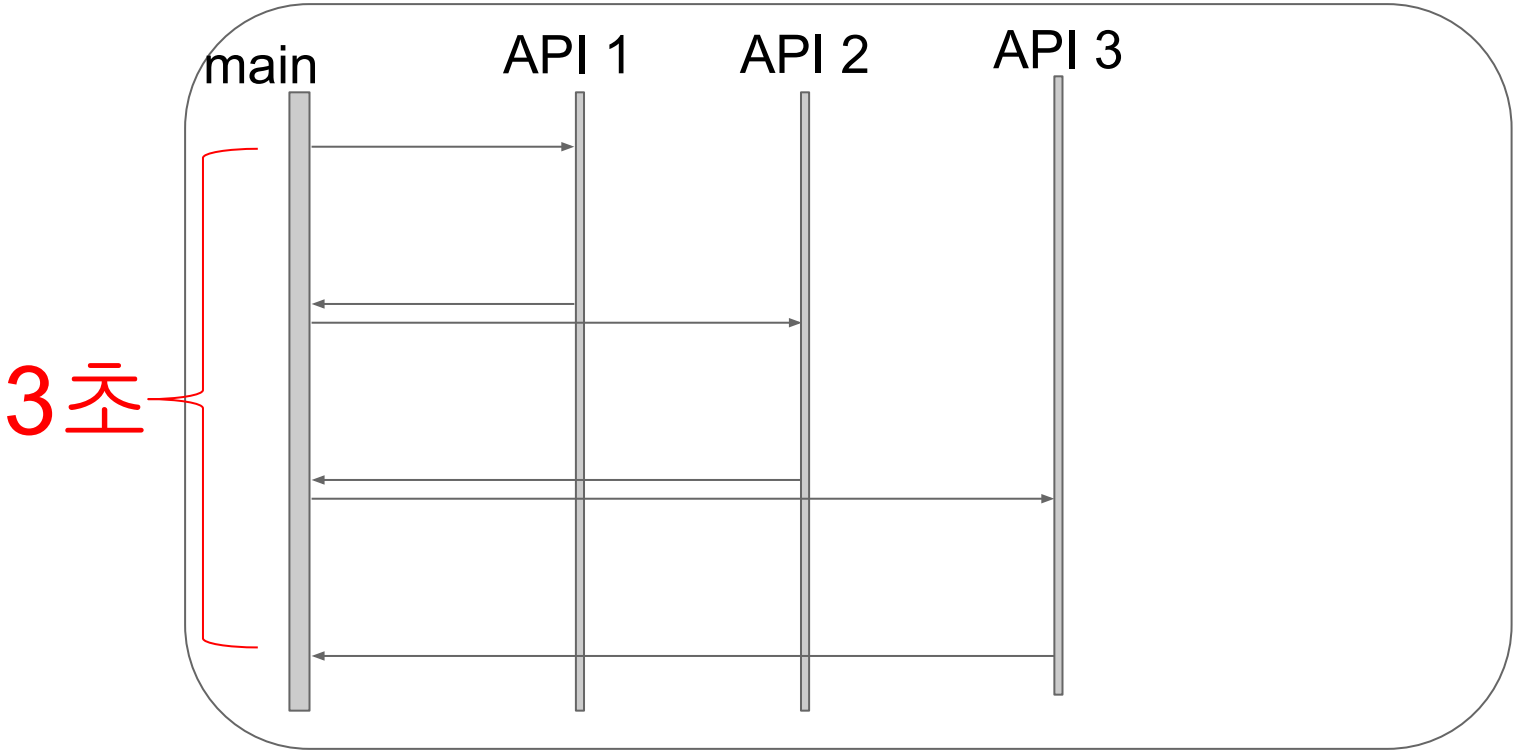
비동기가 필요한 상황



비동기가 필요한 상황

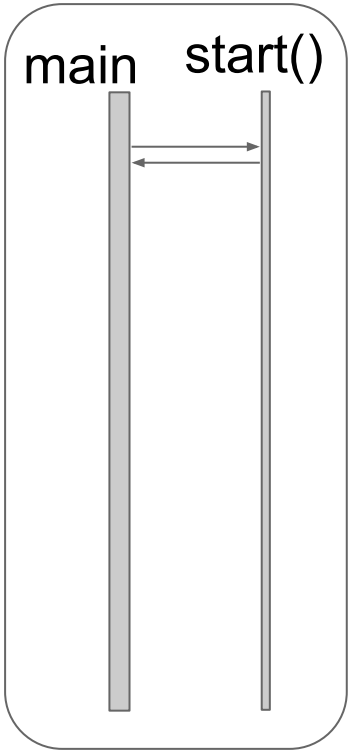


비동기가 필요한 상황

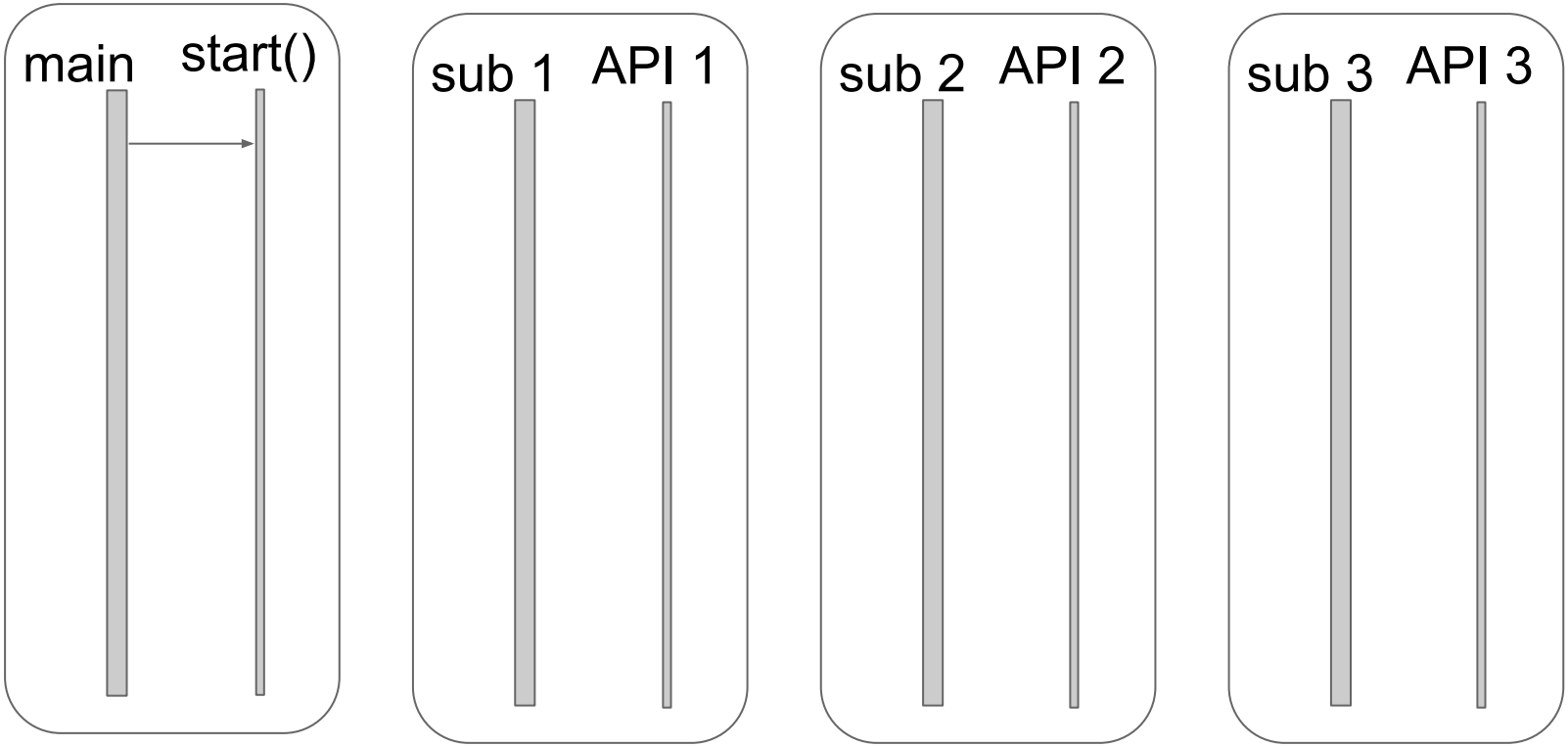


비동기로 바꾸면

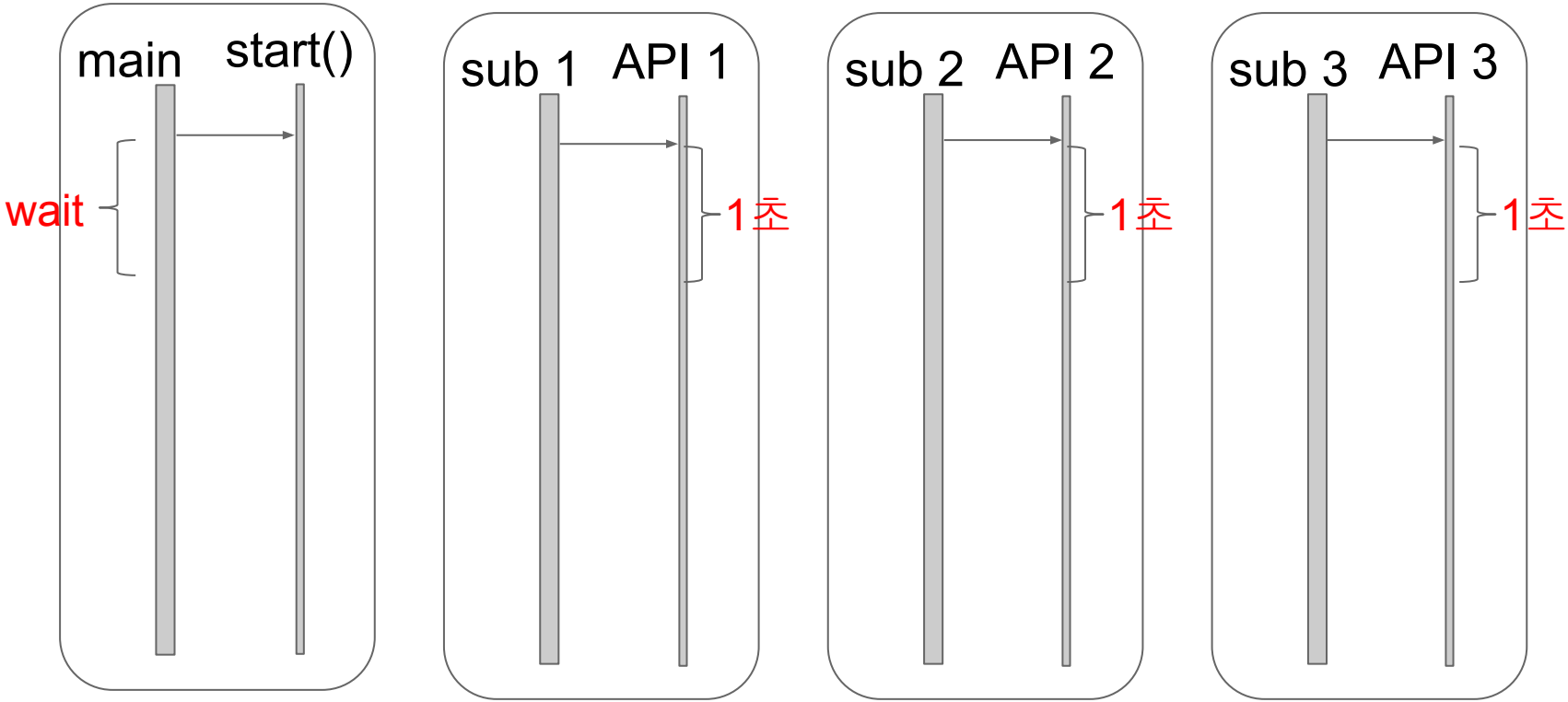
비동기가 필요한 상황



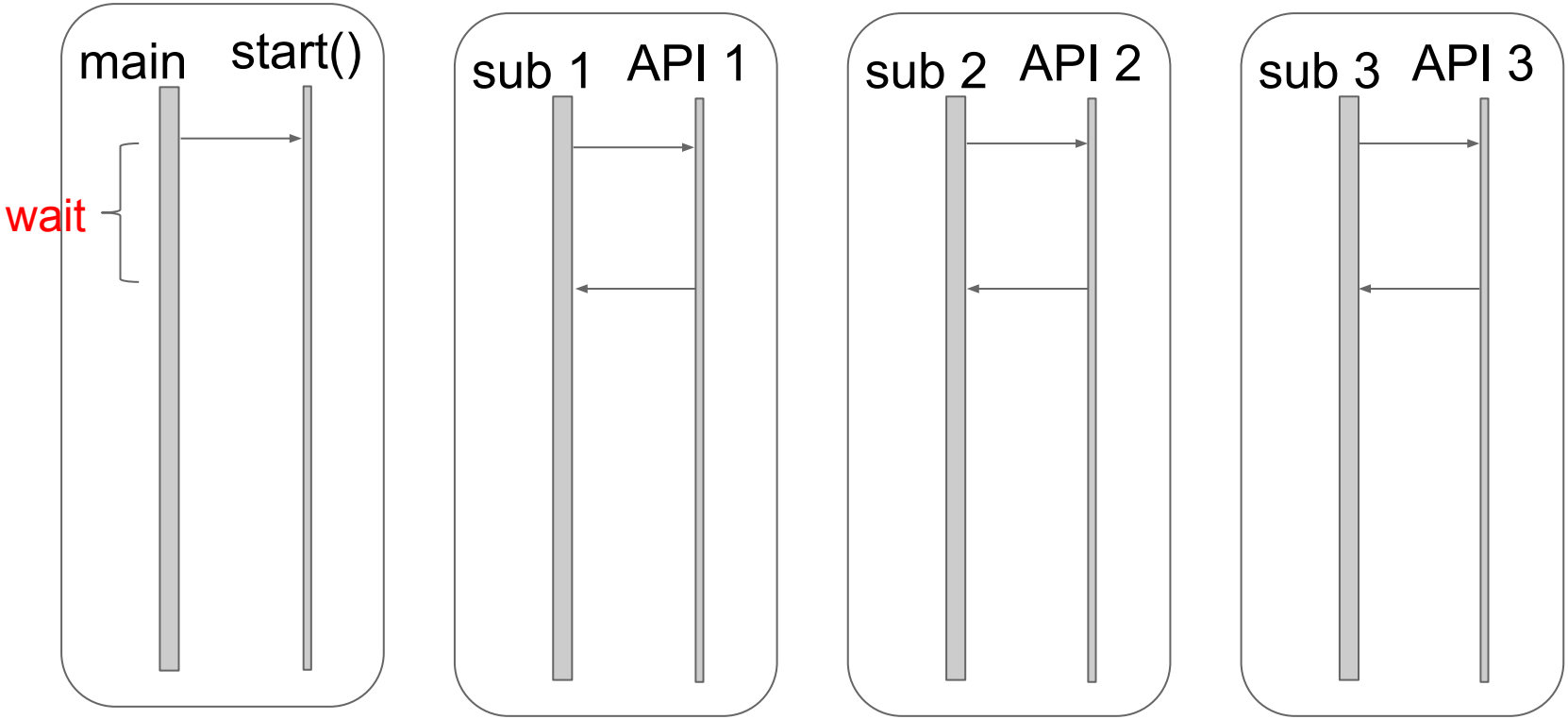
비동기가 필요한 상황



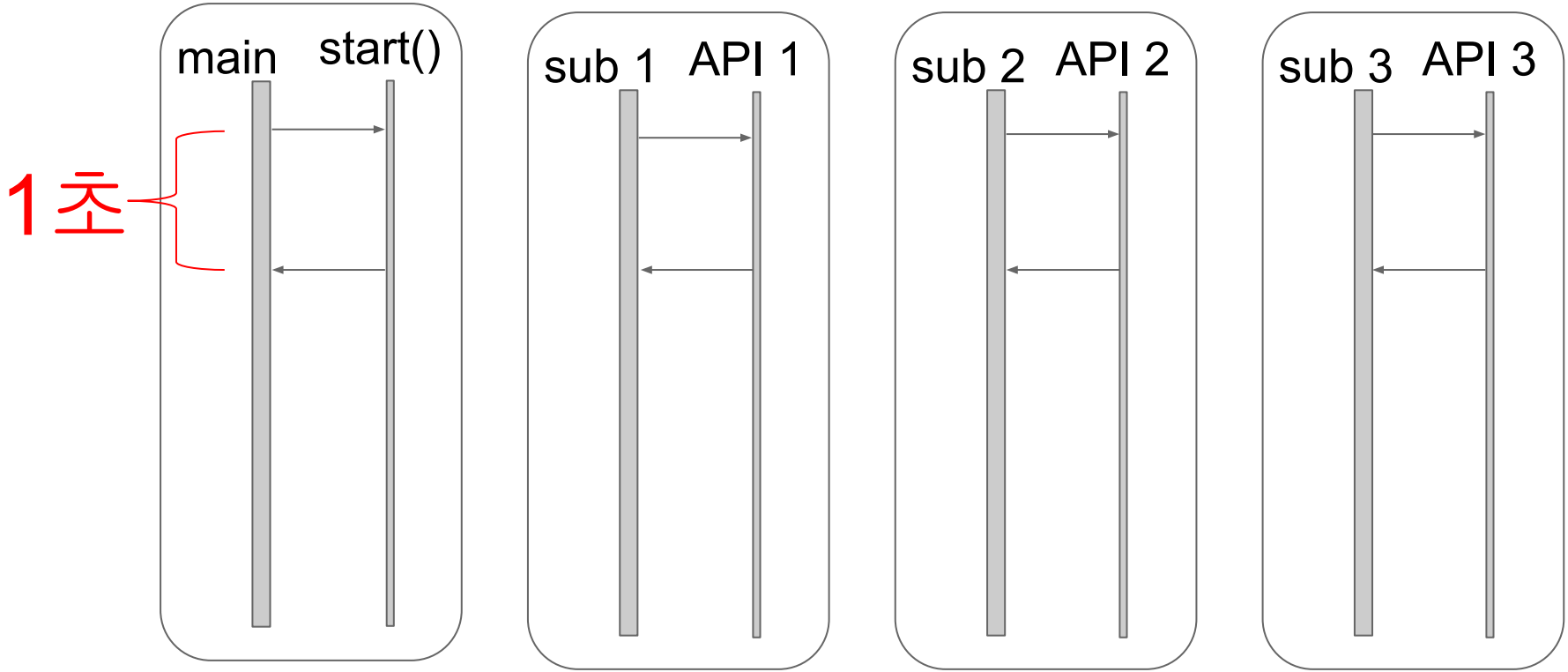
비동기가 필요한 상황



비동기가 필요한 상황



비동기가 필요한 상황



Java 비동기 API

- java.lang
  - class Thread
    - void start()
  - interface Runnable
    - void run()

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

Java 비동기 API

```
public static void main(String[]                                    tedException {

    Thread thread = new Thread
        @Override
        public void run() {
            System.out.println(
            Result result = get
        }
    });
    thread.start();
    System.out.println("main");

}
```

(jvm 실행시 실행)
main 스레드

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

Java 비동기 API

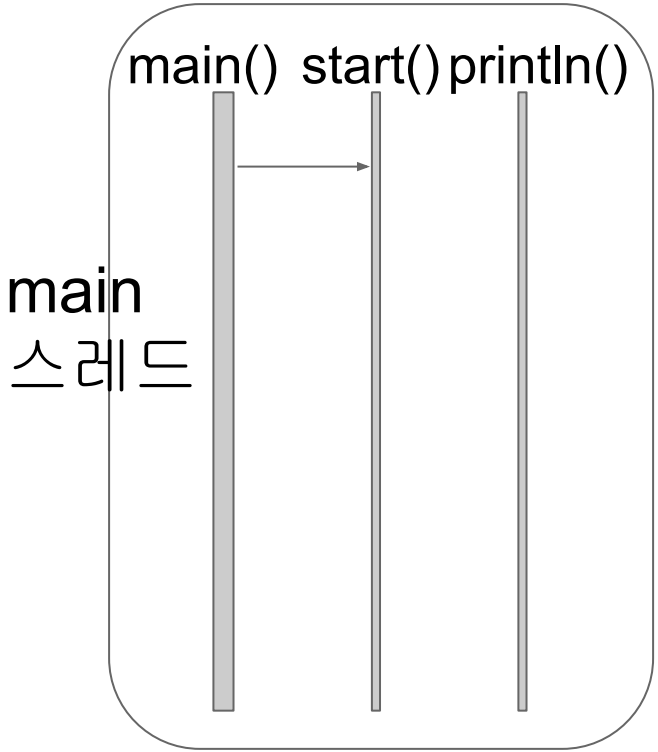
```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

sub
스레드

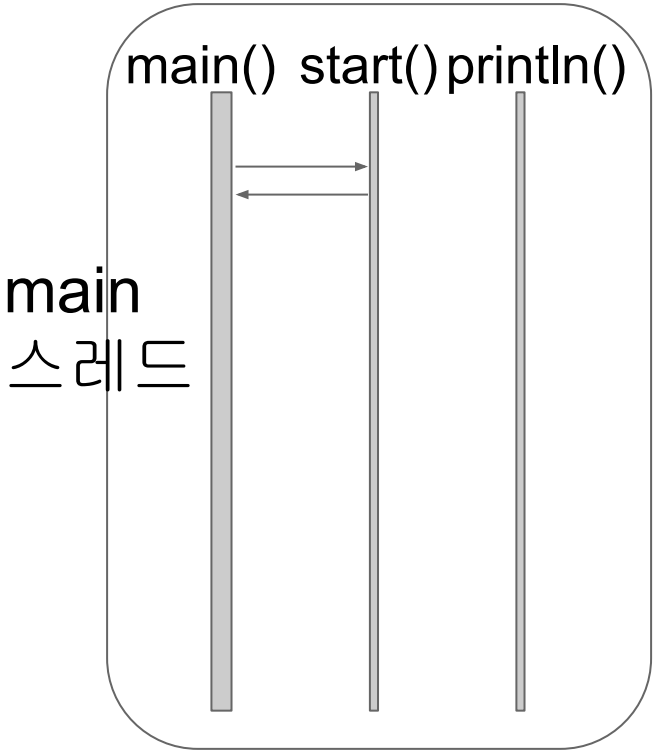
Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

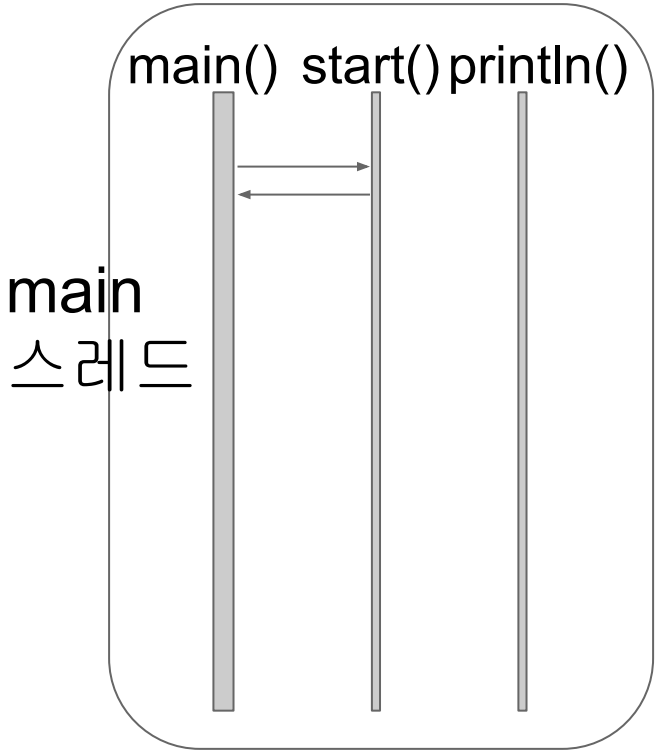
Java 비동기 API

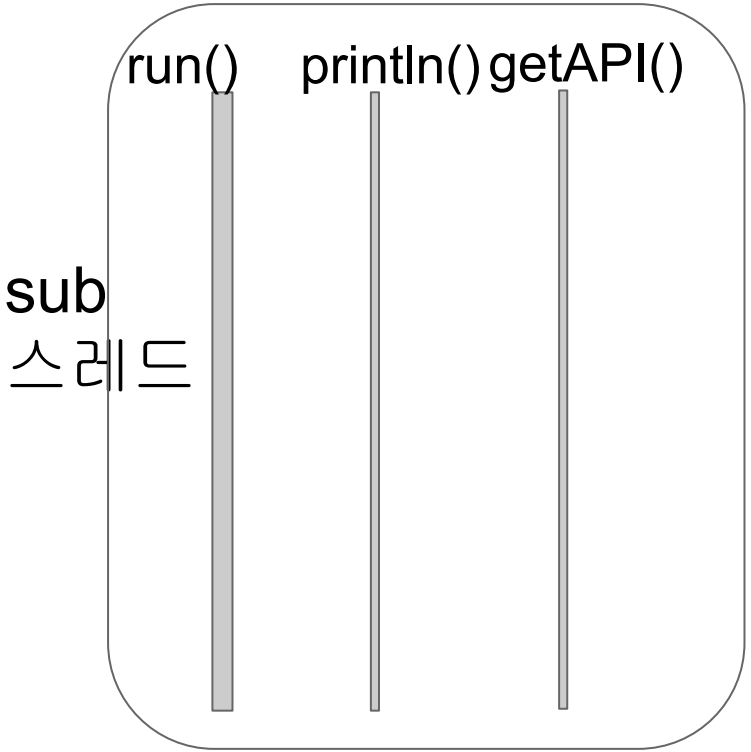


Java 비동기 API

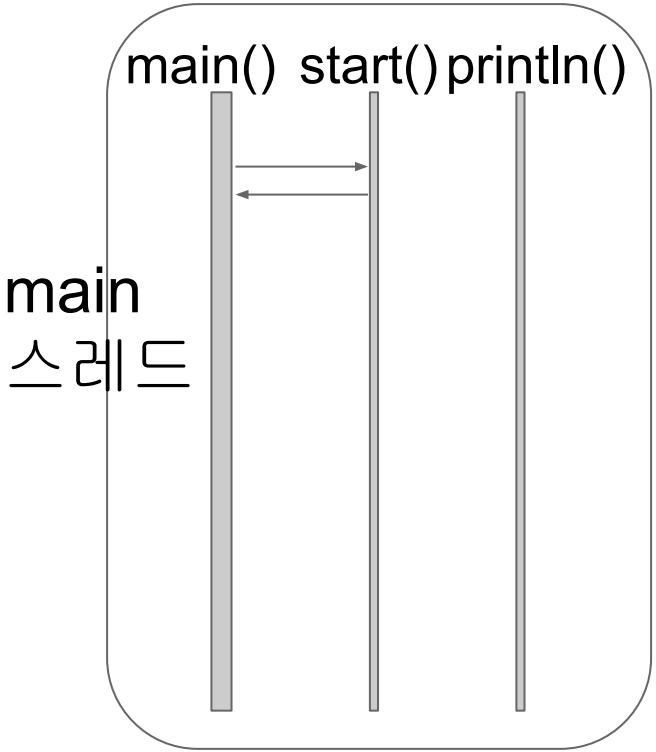


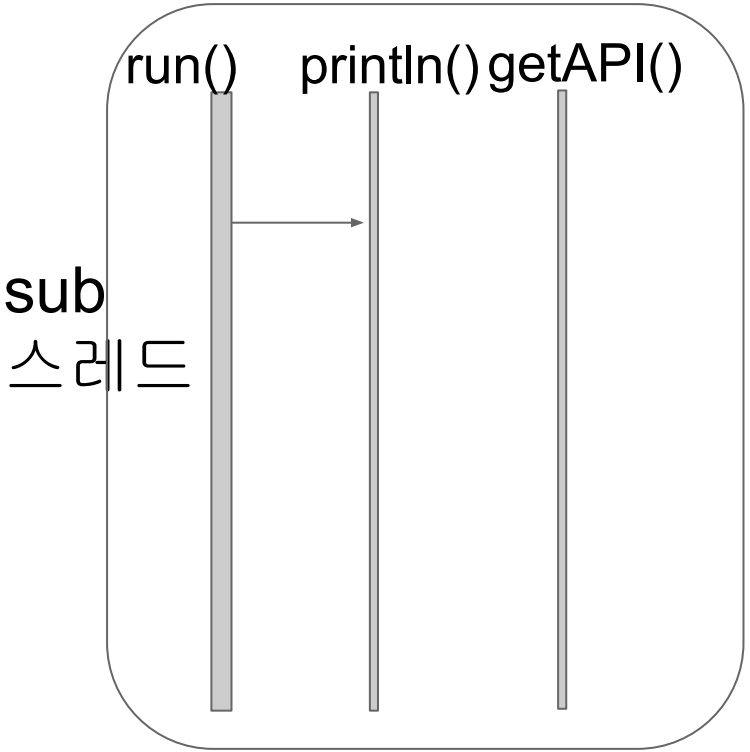
Java 비동기 API



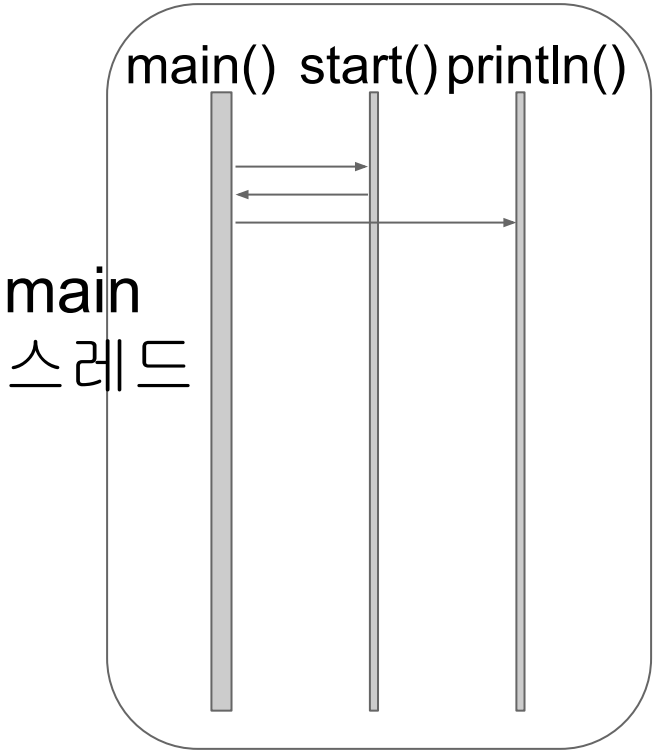


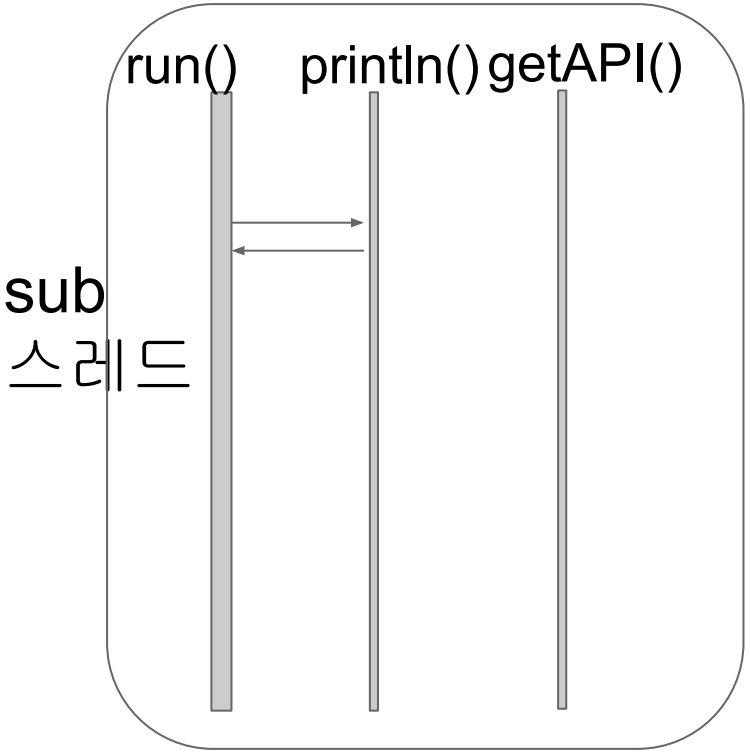
Java 비동기 API



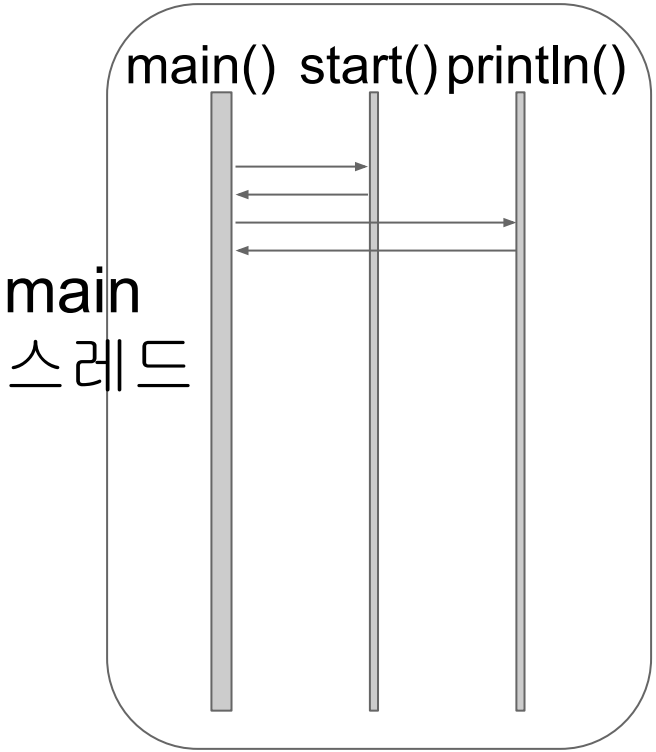


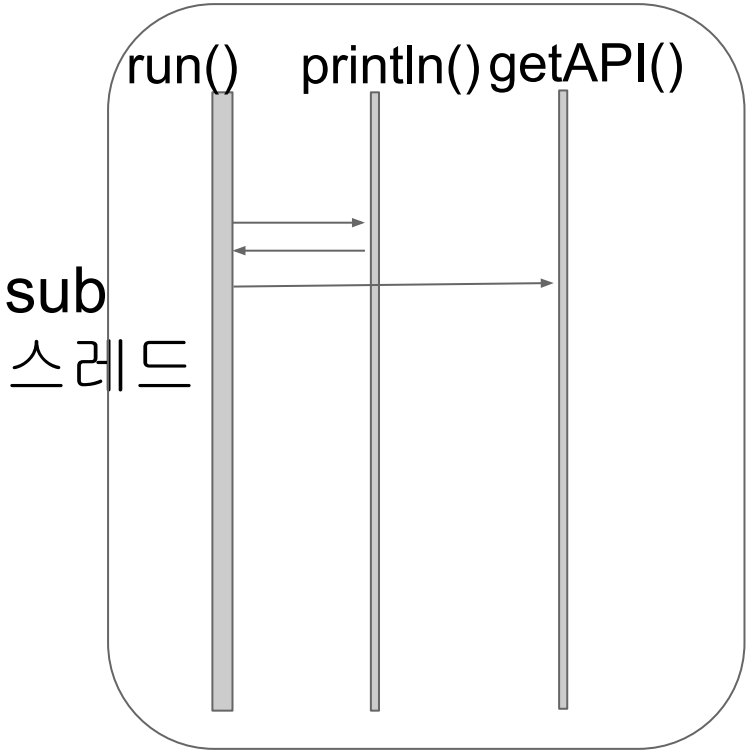
Java 비동기 API



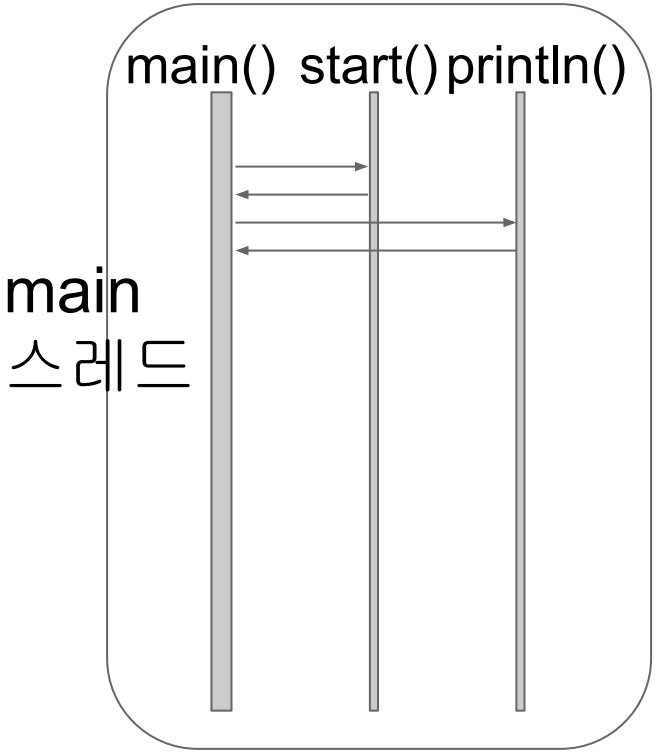


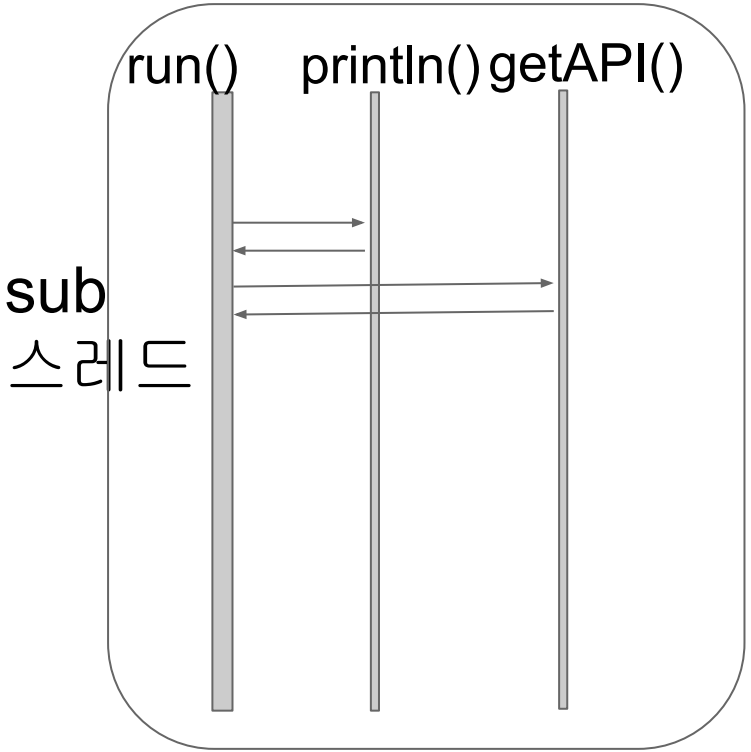
Java 비동기 API





Java 비동기 API





비동기 코드가 어려운점

실행 순서 흐름을 놓치기 쉽다.

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("run");
            Result result = getAPI();
        }
    });
    thread.start();
    System.out.println("main");

}
```

같은
파일내에
소스가
있지만

Java 비동기 API

main
스레드

```
public static void main(String[] args)
    thread.start();
    System.out.println("main");

}
```

sub
스레드

```
@Override
public void run() {
    System.out.println("run");
    Result result = getAPI();
}
```

interface Runnable

- void run()

비동기 결과값

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
        }
    });
    thread.start();
    System.out.println(map.get("API"));

}
```

Java 비동기 API

```
public static void main(String[] ar
    Map map = new HashMap();

    Thread thread = new Thread(new
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
        }
    });
    thread.start();
    System.out.println(map.get("API"));

}
```

객체는 힙에 저장
스레드는 힙을 공유

Java 비동기 API

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
        }
    });
    thread.start();
    System.out.println(map.get("API"));

}
```

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
        }
    });
    thread.start();
    System.out.println(map.get("API"));

}
```

Java 비동기 API

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
        }
    });
    thread.start();
    System.out.println(map.get("API"));

}
```

null

Java 비동기 API

```
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
        }
    });
    thread.start();
    System.out.println(map.get("API"));

}
```

getAPI()
호출시점과

map.get()
호출시점이 다름

- 스레드들간의 작업 순서 동기화 필요

- class Object
  - wait(), notify()

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
            synchronized (map) {
                map.notify();
            }
        }
    });
    thread.start();
    if(map.get("API") == null) {
        synchronized (map) {
            map.wait();
        }
    }
    System.out.println(map.get("API"));
}
```

Java 비동기 API

```java
public static void main(String[] args) throws InterruptedException {
    Map map = new HashMap();

    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Result result = getAPI();
            map.put("API", result);
            synchronized (map) {
                map.notify();
            }
        }
    });
    thread.start();
    if(map.get("API") == null) {
        synchronized (map) {
            map.wait();
        }
    }
    System.out.println(map.get("API"));
}
```

프로그램이 복잡해지고 notify를 빼먹는다면?

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        Result result = getAPI();
        map.put("API", result);
        /*synchronized (map) {
            map.notify();
        }*/
    }
});
```

좀더 추상화된 고급 기술

Future

비동기 개발에 사용한 API - Future

비동기작업의 결과를 받아오고 싶을 때.

- Interface Future<V>
  - V get(): 값을 가져오위해 wait, notify로했던 작업을 메서드 하나로 추상화
  - boolean isDone()
- Class FutureTask<V>
- interface Callable
  - V call()

비동기 개발에 사용한 API - Future

```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    Future<Result> future = es.submit(callable);
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

비동기 개발에 사용한 API - Future

```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    Future<Result> future = es.submit(callable);
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

비동기 개발에 사용한 API - Future

```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    Future<Result> future = es.submit(callable);
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

비동기 개발에 사용한 API - Future

```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    Future<Result> future = es.submit(c
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

block

비동기 개발에 사용한 API - Future

```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.[illegible]ThreadExecutor();
    Future<Result> future = es.submit(callable);
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

완료

비동기 개발에 사용한 API - Future

```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    Future<Result> future = es.submit(c
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

block 해제

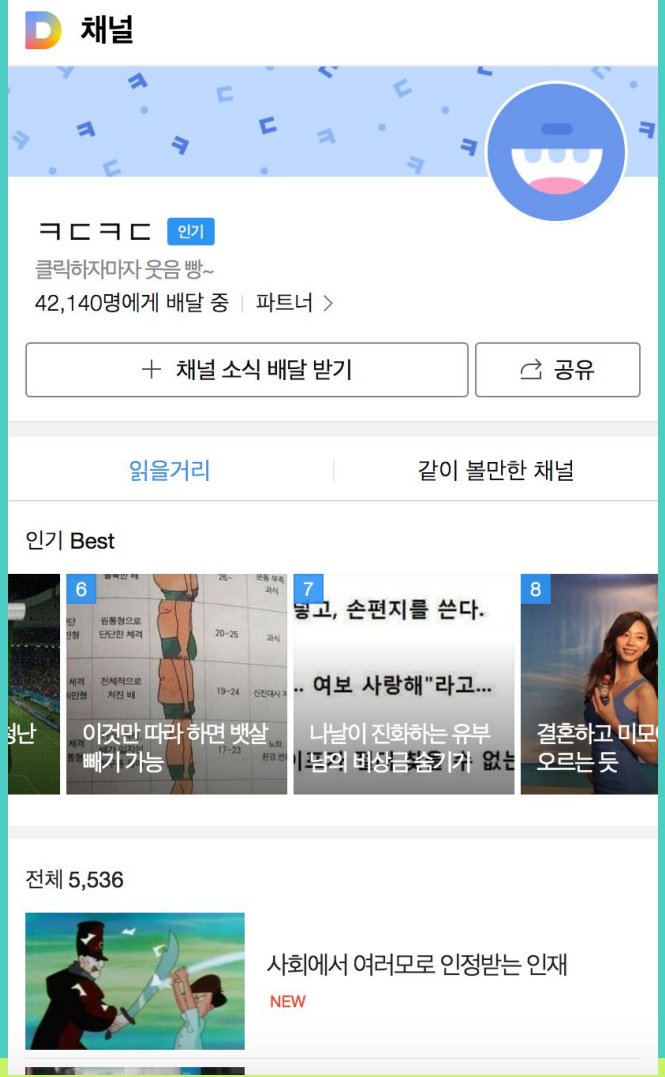
비동기 개발에 사용한 API - Future

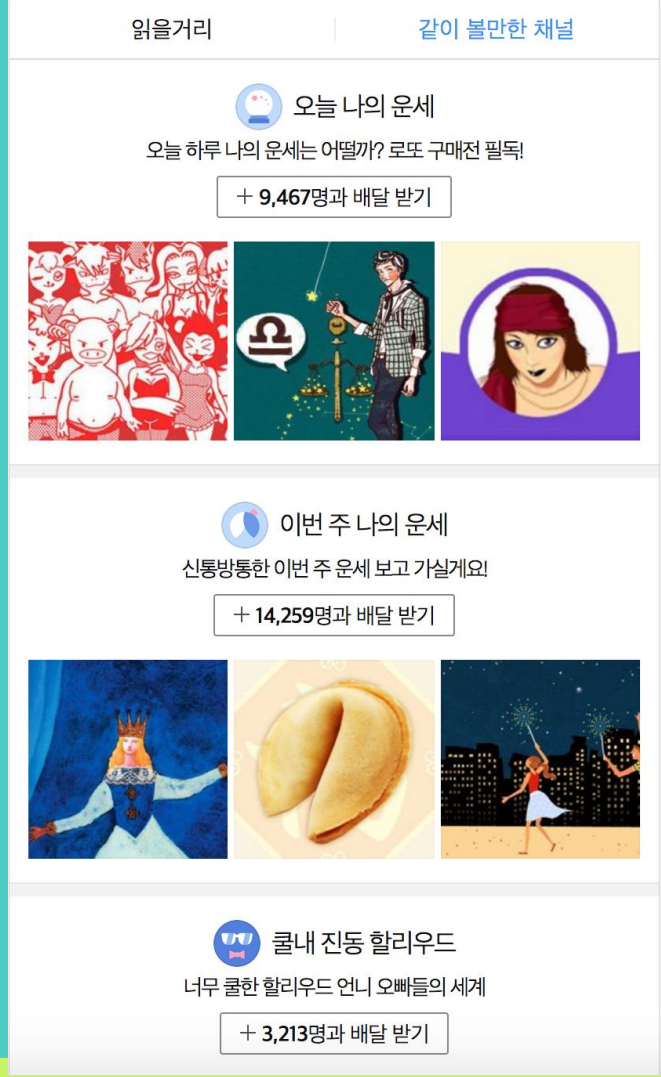
```
public static void main(String[] args) throws InterruptedExcept
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    Future<Result> future = es.submit(callable);
    Result result = future.get();
    System.out.println(result);
    es.shutdown();
}
```

비동기를
활용하여

개발완료!

BUT...





느낀점

- thread를 직접사용하는건 쉽지 않다 (지식 + 경험)

느낀점

- Future는 단일값만 처리하기 때문에 복합적으로 구현하기 어렵다.
  - A작업 -> B작업 -> C작업.....

느낀점

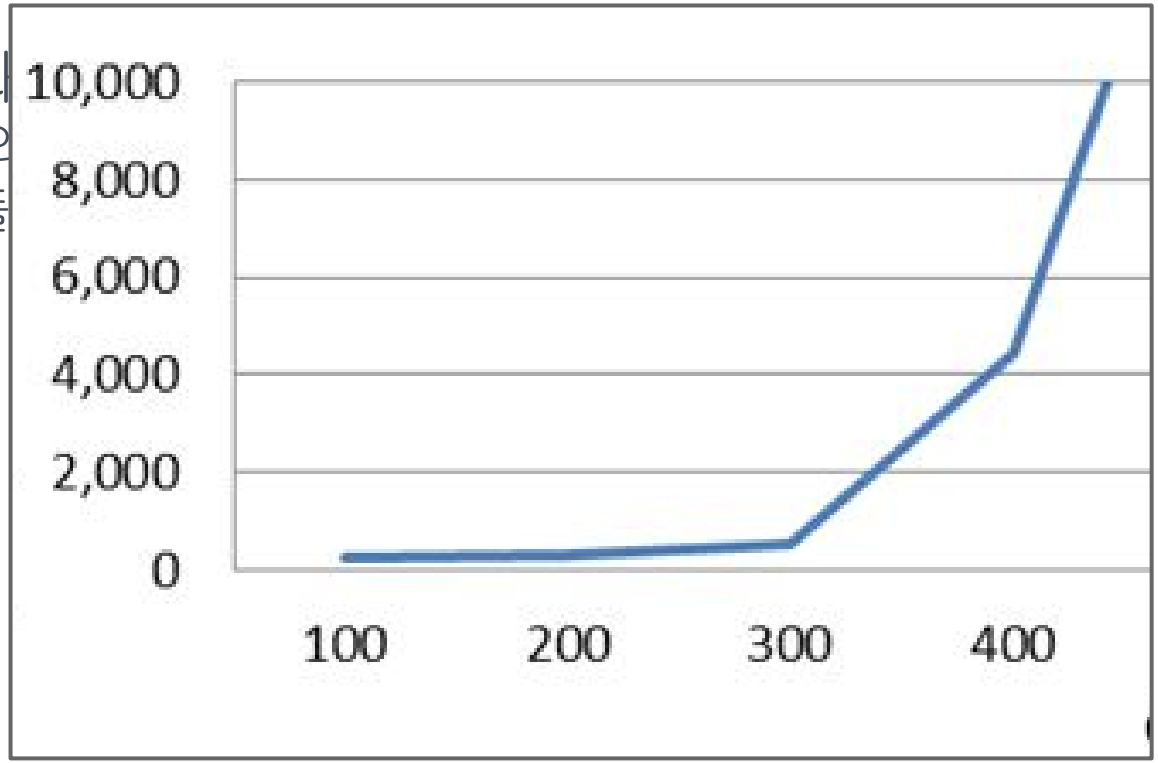
- threadpool에 대해서도 이해가 필요하다.

느낀점

- threadpool에 대해서도 이해가 필요하다.
  - threadpool내의 thread가 전체 사용중인경우 큐에 대기중인 요청들은 처리시간이 이전 태스크가 끝나야 처리되기때문에 응답 지연 증가된다.

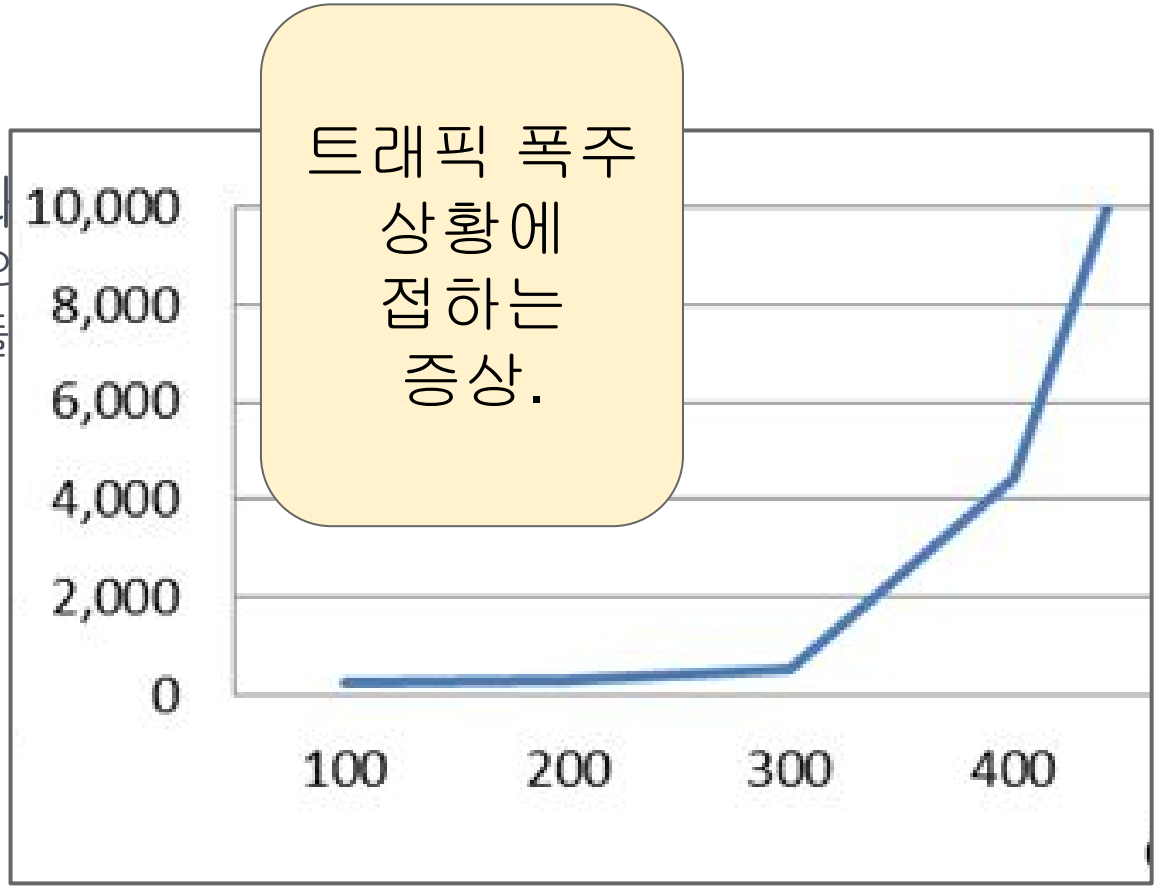
느낀점

- threadpool에 다
  - threadpool내의
    대기중인 요청들
    처리되기때문에



느낀점

- threadpool에 다
  - threadpool내의 대기중인 요청들 처리되기때문에



느낀점

- threadpool에 대해서도 이해가 필요하다.
  - threadpool내의 thread가 전체 사용중인경우 큐에 대기중인 요청들은 처리시간이 이전 태스크가 끝나야 처리되기때문에 응답 지연 증가된다.
  - 잘못 사용한 threadpool
    - newCachedThreadPool
      - 1client * 7 requet => 1 + 7 (threads)

느낀점

- threadpool에 대해서도 이해가 필요하다.
  - threadpool내 thread가 전체 사용중인경우 큐에 대기중인 요청들은 이전 태스크가 끝나야 처리되기 때문에 응답 지연 증가
  - 잘못 사용한 threadpool
    - newCachedThreadPool
      - 1client * 7 requet => 1 + 7 (threads)
      - 300client * 7 request => 300 + 2100 (threads)

3.

JAVA 비동기 프로그래밍 ver.2

도입
CompletableFuture

도입

- 기존 비동기 코드로 성능향상을 보았지만... 좀더 살펴보면.
  - Future는 하나의 단일 작업만 처리

도입

- 기존 Future의 단일 작업 처리문제
  - CompletableFuture (JDK 8)

A -> B -> C

의존성있는 태스크

기존 Future에서

해결하려면!

```java
Future<Result> firstFuture = es.submit(new Callable() {
    @Override
    public Object call() throws Exception {
        Result firstResult = getFirstAPI("Hello KSUG!!!");
        Future secondFuture = es.submit(new Callable() {
            @Override
            public Object call() throws Exception {
                Result secondResult = getSecondAPI(firstResult);
                Future thirdFuture = es.submit(new Callable() {
                    @Override
                    public Object call() throws Exception {
                        return getThirdAPI(secondResult);
                    }
                });
                return thirdFuture.get();
            }
        });
        return secondFuture.get();
    }
});
```

```
Future<Result> firstFuture = es.submit(new Callable() {
    @Override
```

Callback Hell

```
function doSomething(params){
  $.get(url, function(result){
    setTimeout(function(){
      startAsyncProcess(function(){
        $.post(url, function(response){
          if(response.good){
            setStateasGoodResponse(function(
              console.log('Hooray!')
            });
          }
        });
      });
    });
  });
}
```

```
});
```

CompletableFuture

- java.util.concurrent
  - Class CompletableFuture<T>
    - interface Future<T>
    - interface CompletionStage<T>
- Future 시리즈중 끝판왕

CompletableFuture

```
CompletableFuture<String> cf = new CompletableFuture<>();
new Thread(new Runnable() {
    @Override
    public void run() {
        cf.complete( value: "Hello KSUG!!!");
    }
}).start();
String result = cf.get();
System.out.println(result);
```

CompletableFuture

```
CompletableFuture<String> cf = new CompletableFuture<>();
new Thread(new Runnable() {
    @Override
    public void run() {
        cf.complete( value: "Hello KSUG!!!");
    }
}).start();
String result = cf.get();
System.out.println(result);
```

cf.get()
블럭

CompletableFuture

```
CompletableFuture<String> cf = new CompletableFuture<>();
new Thread(new Runnable() {
    @Override
    public void run() {
        cf.complete( value: "Hello KSUG!!!");
    }
}).start();
String result = cf.get();
System.out.println(result);
```

cf.get()
블럭 해제

CompletableFuture

```
CompletableFuture<String> cf = new CompletableFuture<>();
new Thread(new Runnable() {
    @Override
    public void run() {
        cf.complete( value: "Hello KSUG!!!");
    }
}).start();
String result = cf.get();
System.out.println(result);
```

CompletableFuture

```
CompletableFuture<String> cf = new CompletableFuture<>();
new Thread(new Runnable() {
    @Override
    public void run() {
        cf.complete( value: "Hello KSUG!!!");
    }
}).start();
String result = cf.get();
System.out.println(result);
```

익명메서드

verbose

CompletableFuture

Java8

```
CompletableFuture<String> cf = new CompletableFuture<>();
```


```
CompletableFuture<String> cf = new CompletableFuture<>();
```


```
new Thread(() -> cf.complete(value: "Hello KSUG!!!"))
    .start();
String result = cf.get();
System.out.println(result);
```

람다


```
String result = cf.get();
System.out.println(result);
```

CompletableFuture

태스크간 순서

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(msg -> getFirstAPI(msg))
        .thenApply(param -> getSecondAPI(param))
        .get();
```

CompletableFuture

태스크간 순서

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(msg -> getFirstAPI(msg))
        .thenApply(param -> getSecondAPI(param))
        .get();
```

CompletableFuture

태스크간 순서

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(msg -> getFirstAPI(msg))
        .thenApply(param -> getSecondAPI(param))
        .get();
```

CompletableFuture

태스크간 순서

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(msg -> getFirstAPI(msg))
        .thenApply(param -> getSecondAPI(param))
        .get();
```

CompletableFuture

Java8

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(msg -> getFirstAPI(msg))
        .thenApply(param -> getSecondAPI(param))
        .get();
```

람다

CompletableFuture

Java8

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(ThreadMain::getFirstAPI)
        .thenApply(ThreadMain::getSecondAPI)
        .get();
```

메서드
레퍼런스

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

CompletableFuture

조합

```
CompletableFuture<String> cf1 = CompletableFuture.completedFuture("Hello");
CompletableFuture<String> cf2 = CompletableFuture.completedFuture("KSGU");
CompletableFuture<String> cf3 = CompletableFuture.completedFuture("!!!");
String msg = cf1.thenCombine(cf2, (x, y) -> x + " " + y)
        .thenCombine(cf3, (x, y) -> x + y)
        .get();
System.out.println(msg);
```

```
Hello KSGU!!!
```

CompletableFuture

컨텍스트 변경

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(ThreadMain::getFirstAPI)
        .thenApplyAsync(ThreadMain::getSecondAPI)
        .get();
```

CompletableFuture

컨텍스트 변경

```
Result result = CompletableFuture
        .completedFuture("Hello KSUG!!!")
        .thenApply(ThreadMain::getFirstAPI)
        .thenApplyAsync(ThreadMain::getSecondAPI)
        .get();
```

```
[main] INFO examples.p00.threadmain5.ThreadMain
[ForkJoinPool.commonPool-worker-1] INFO example
```

CompletableFuture

CompletableFuture는 좋다!!

뭐가 부족한가 생각해보면...?

반복되는 작업을 Collection에 넣고 이를 CompletableFuture처럼 처리할 수 있다면?

4.

NIO

nio
nonblocking I/O

NIO

- 아직 풀지 못한 상황..
  - ThreadPool Full

NIO

- 아직 풀지 못한 상황..
  - ThreadPool Full
    - Nonblocking

NIO

- NIO
  - jdk 1.4 nio
  - jdk 1.7 nio2
- New IO
  - Native IO
  - Nonblocking IO

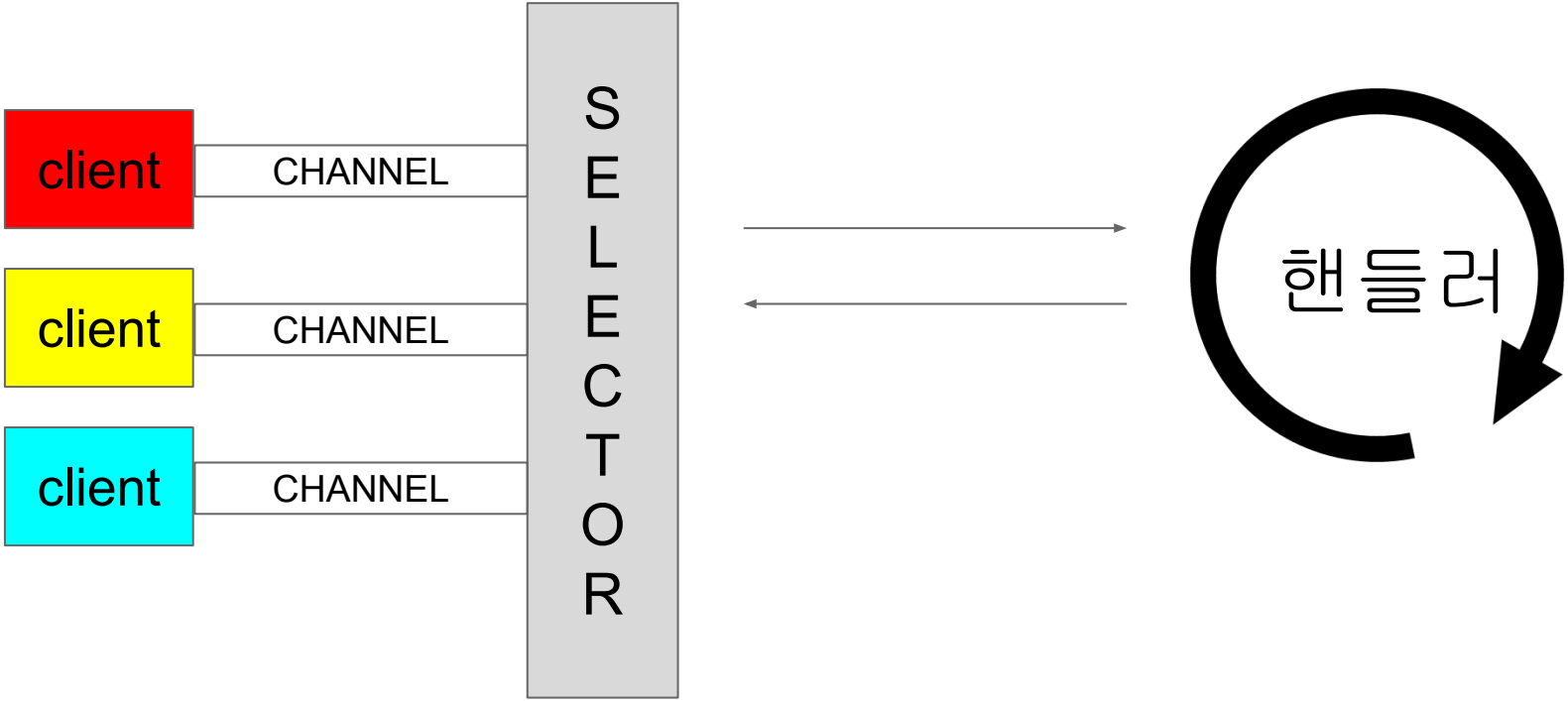
- packages
  - java.nio
  - java.nio.channels
  - java.nio.channels.spi
  - java.nio.charset
  - java.nio.charset.spi
  - java.nio.file
  - java.nio.file.attribute
  - java.nio.file.spi

NIO

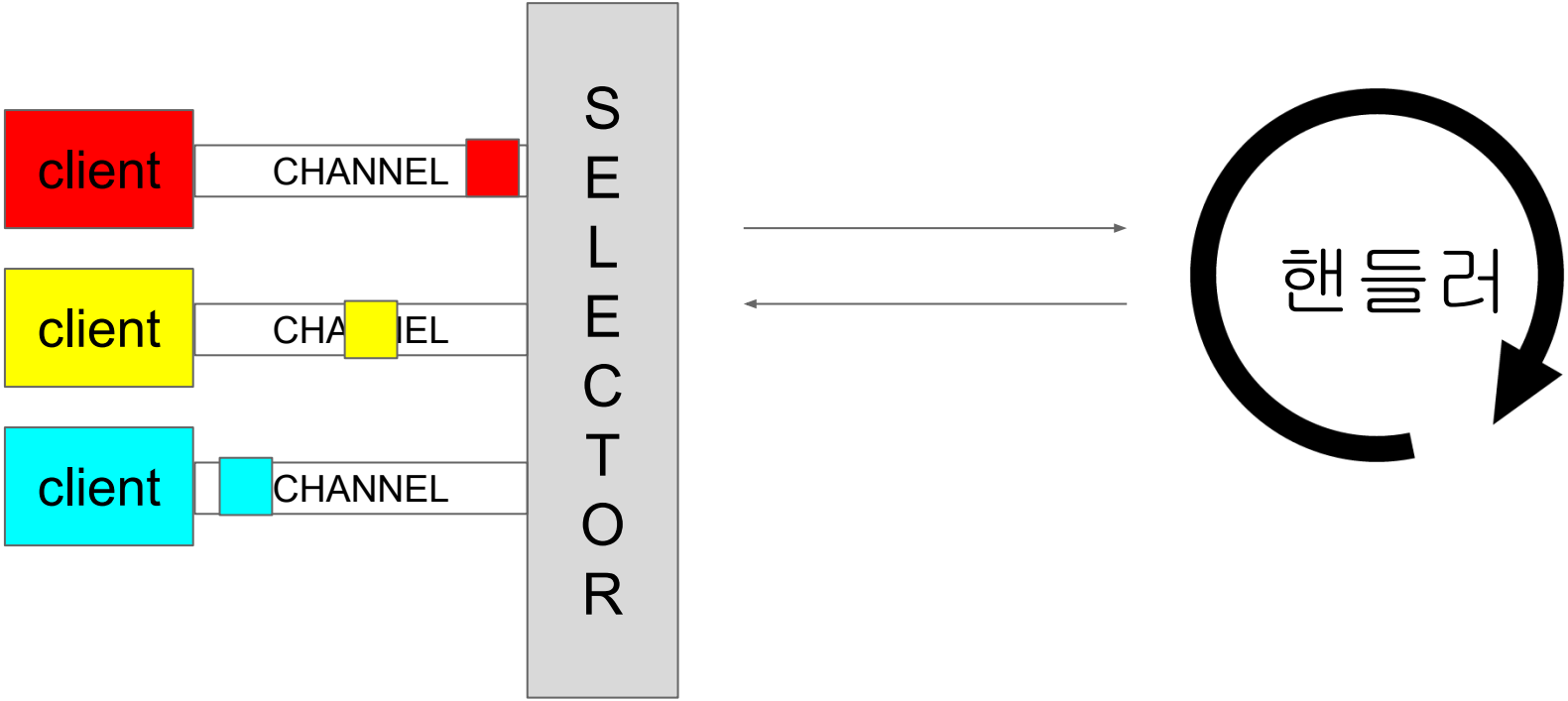
- NIO
  - jdk 1.4 nio
  - jdk 1.7 nio2
- New IO
  - Native IO
  - Nonblocking IO

- packages
  - java.nio
  - java.nio.channels
  - java.nio.channels.spi
  - java.nio.charset
  - java.nio.charset.spi
  - java.nio.file
  - java.nio.file.attribute
  - java.nio.file.spi

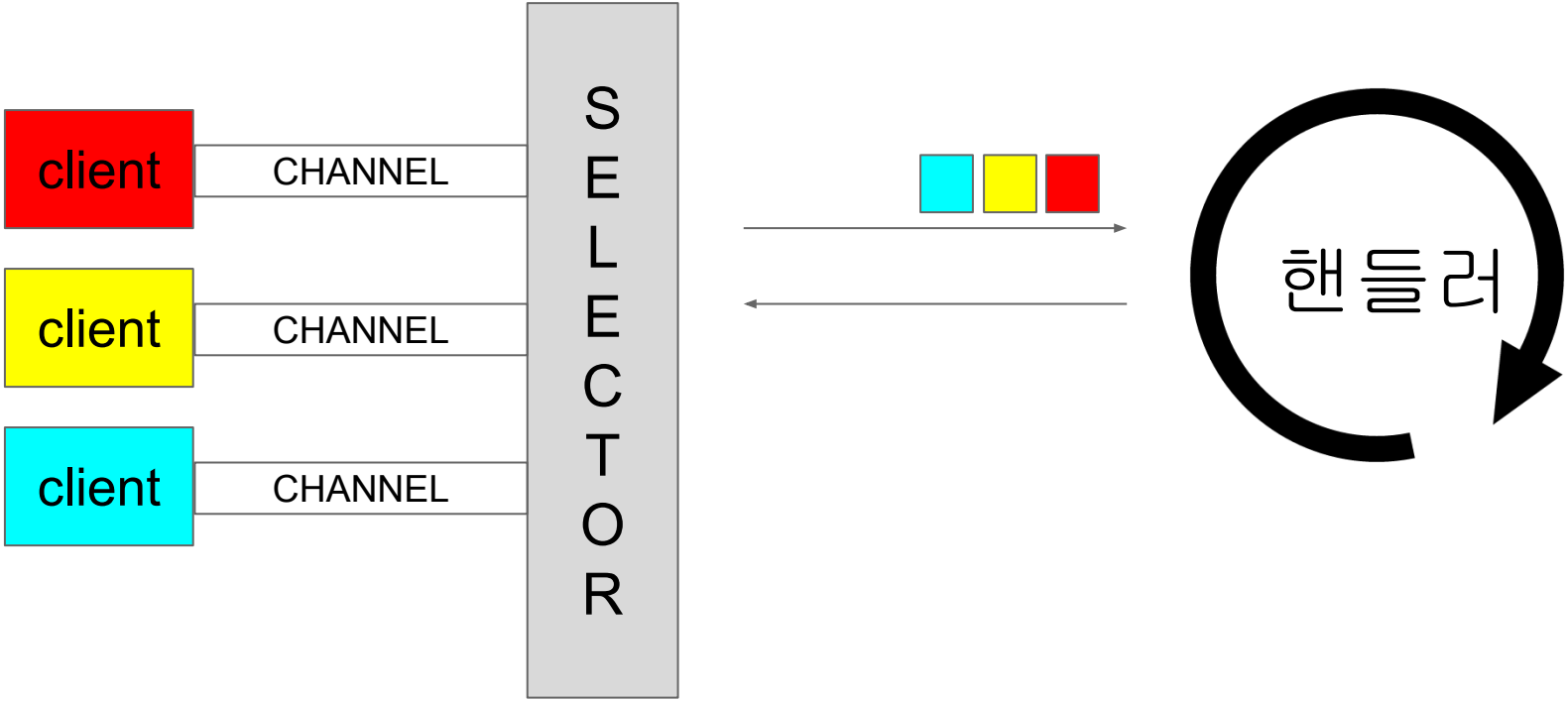
Nonblocking I/O



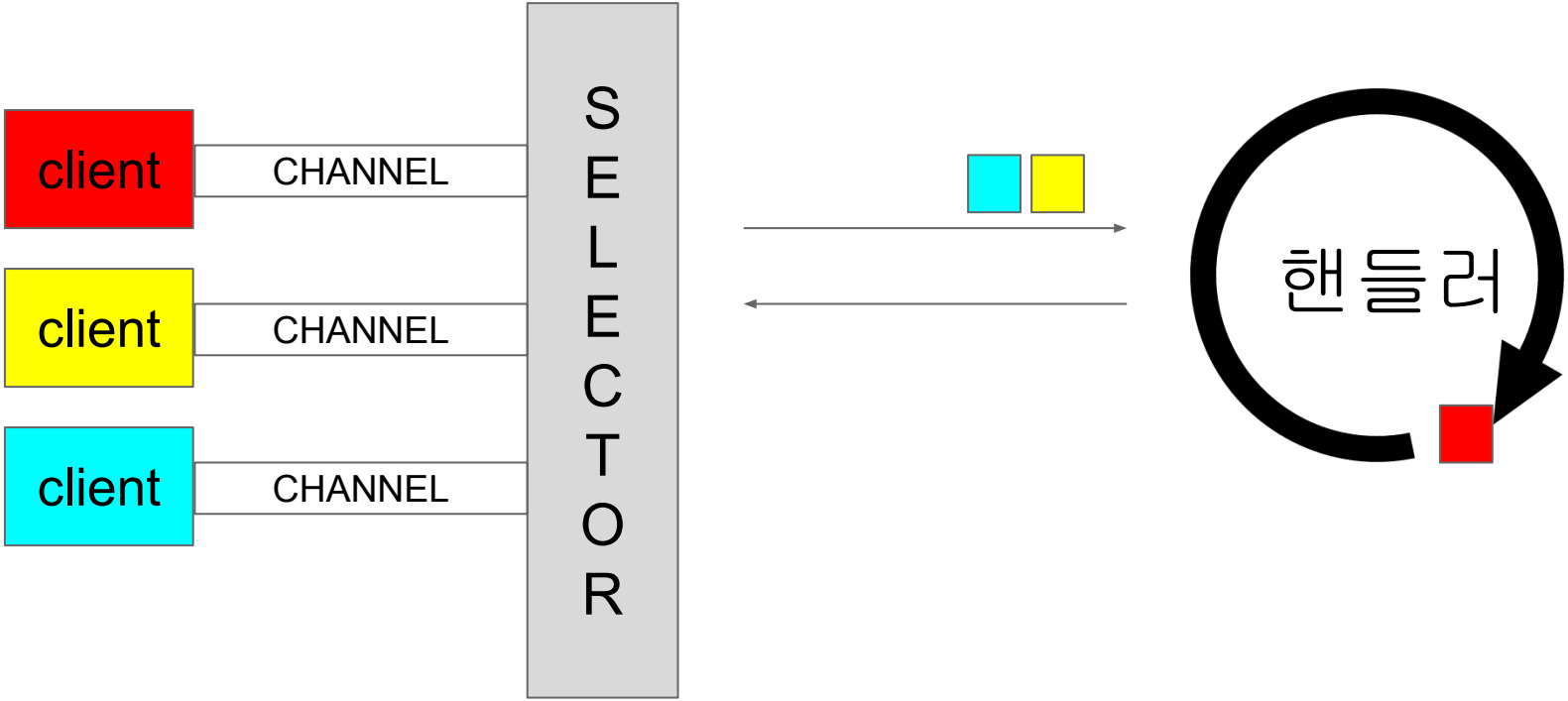
Nonblocking



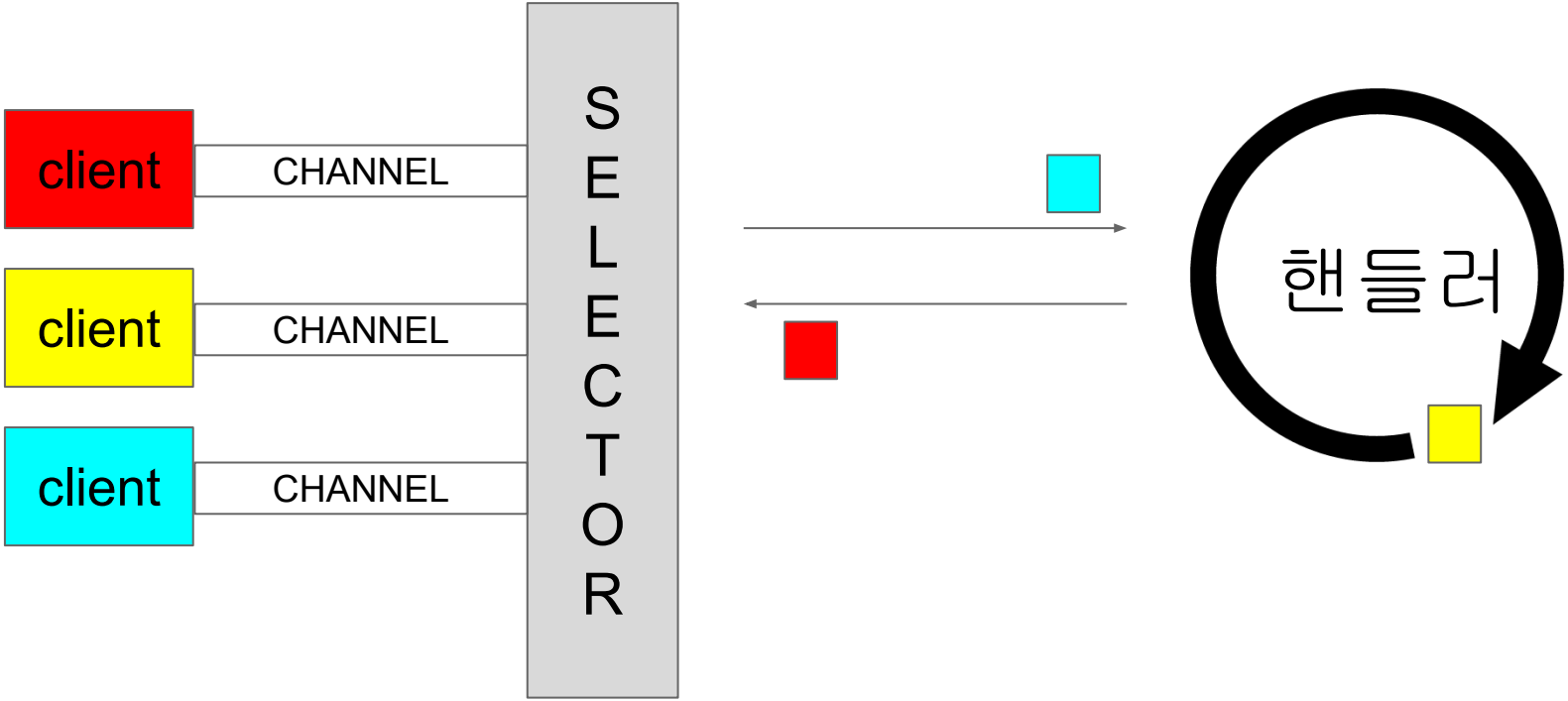
Nonblocking I/O



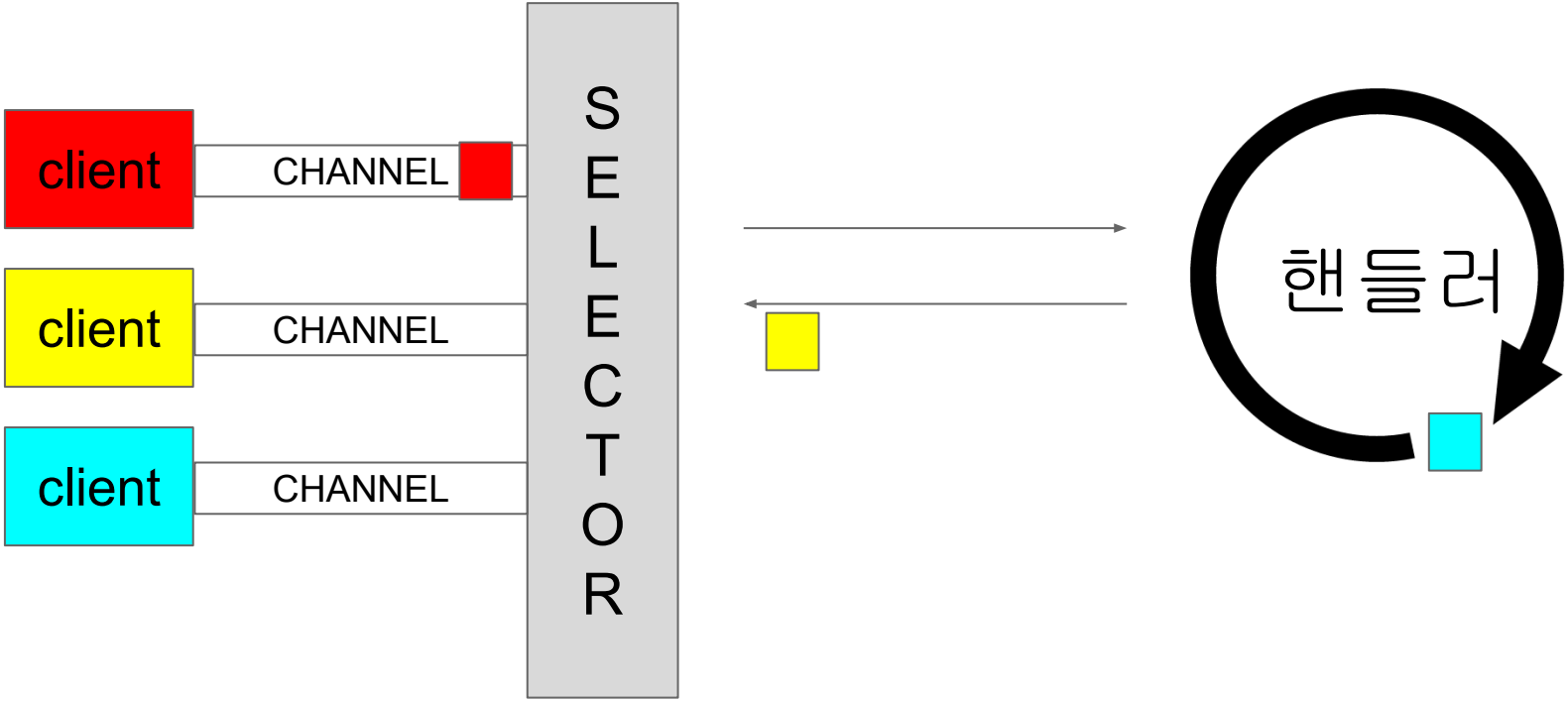
Nonblocking I/O



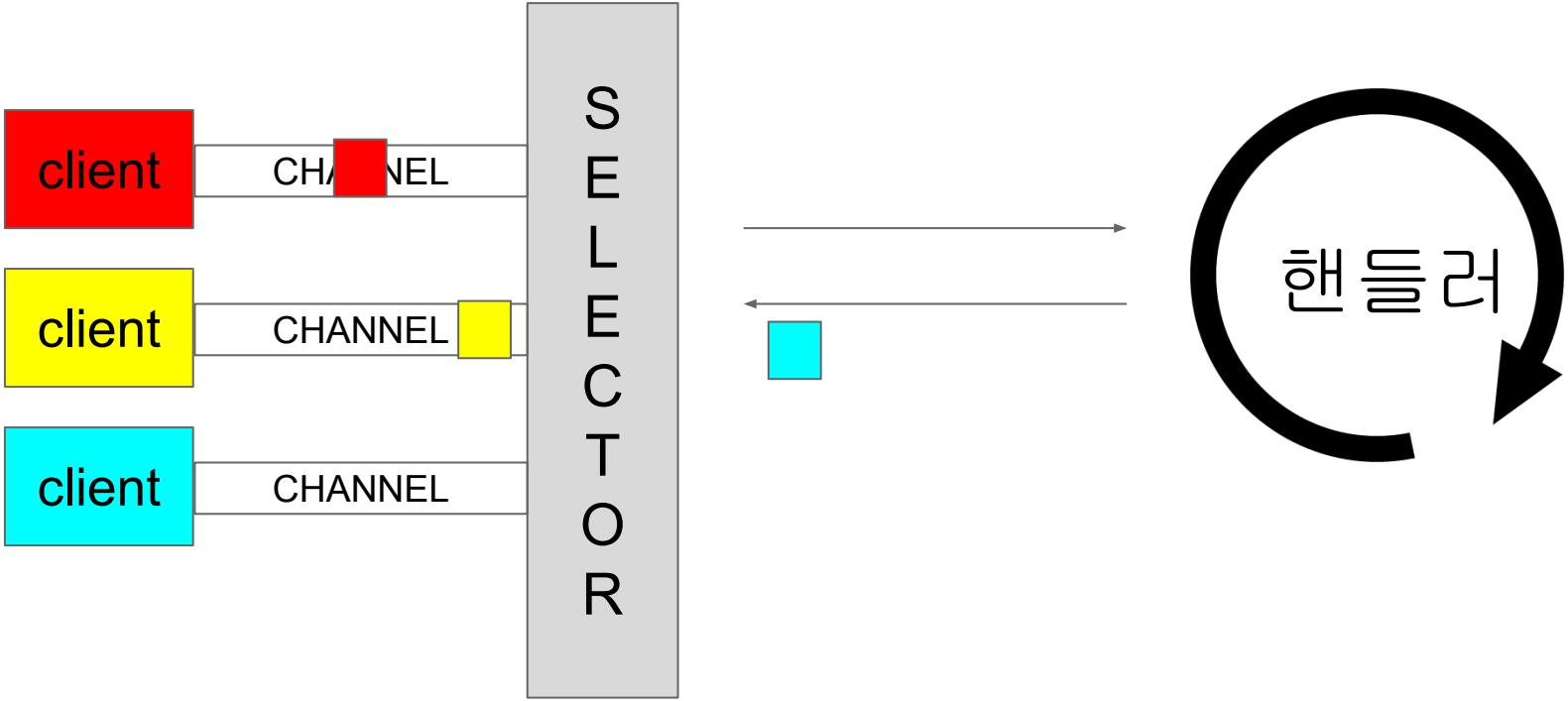
Nonblocking I/O



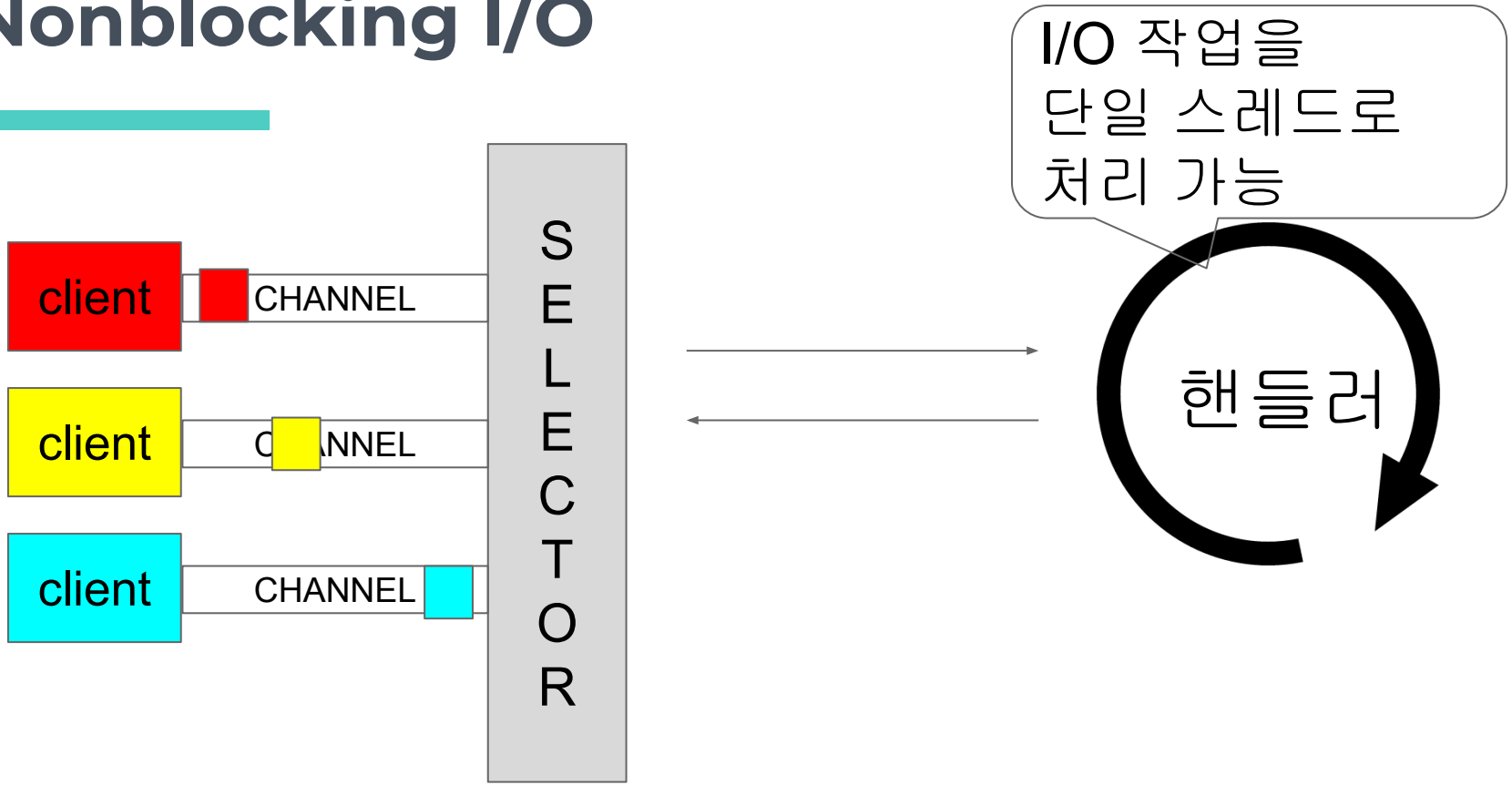
Nonblocking I/O



Nonblocking I/O



Nonblocking I/O



Nonblocking I/O

- Servlet 3.0
  - AsyncServlet - 스레드!!
- Servlet 3.1
  - Nonblocking I/O
- Spring MVC : DeferredResult

시연

4.

Reactive

Event Programming
마무리

Event programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.
  - 이벤트 발생 -> 처리

Event programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.
  - 이벤트 발생 -> 처리
    - CompletableFuture
      .complete(/* 값 */)

Event programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.
  - 이벤트 발생 -> 처리
    - CompletableFuture
      .complete(/* 값 */)

complete이벤트

Event programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.
  - 이벤트 발생 -> 처리
    - CompletableFuture
      .complete(/* 값 */)
    - CompletableFuture
      .thenApply(/* 콜백 */)

Event programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.
  - 이벤트 발생 -> 처리
    - CompletableFuture
      .complete(/* 값 */)
    - CompletableFuture
      .thenApply(/* 콜백 */)

이벤트에대한 처리

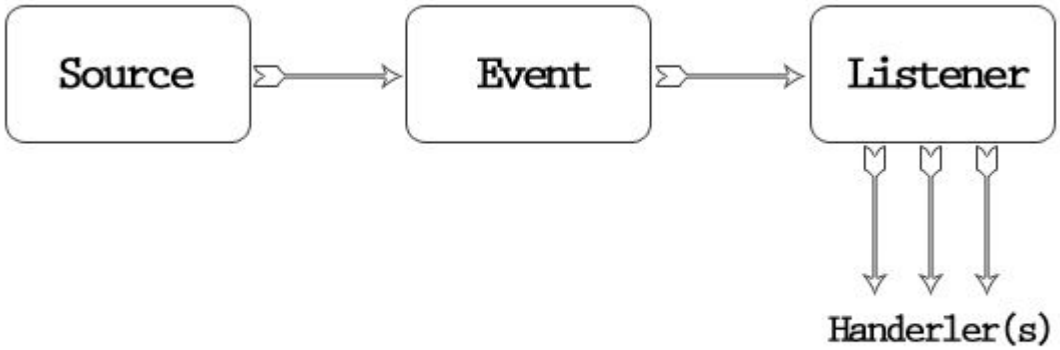
Event programming

- Event
  - 프로그램에 의해 감지되고 처리될 수 있는 동작이나 사건

Event programming

- Event programming
  - 절차지향 프로그래밍과 비교(?)되는 프로그래밍 패러다임
  - 이벤트가 발생한다는 전제를 깔고 이벤트를 처리하는 코드를 작성

Event programming



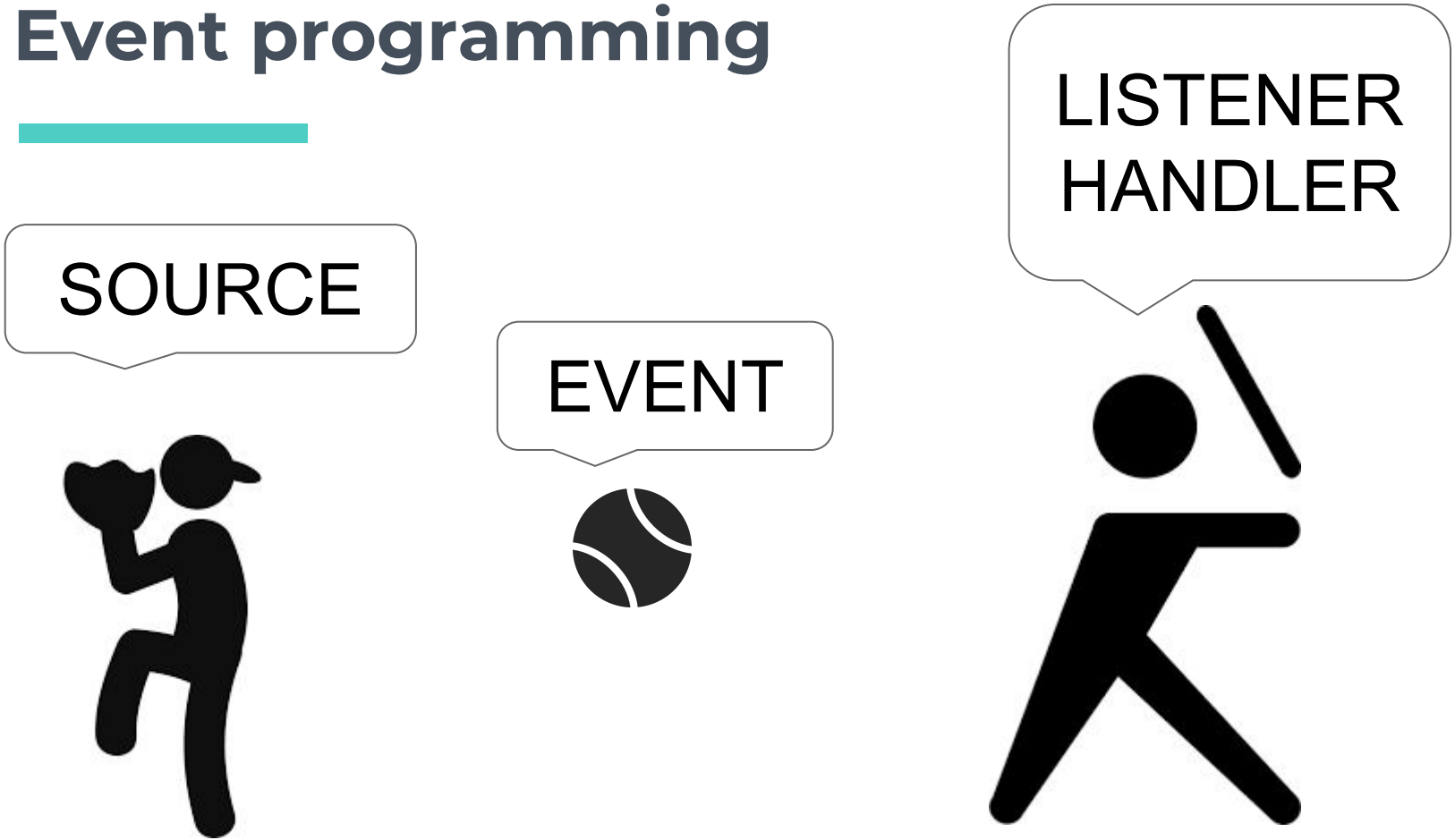
Event-Driving Programming Model

Event programming

Event programming

Event programming

Event programming



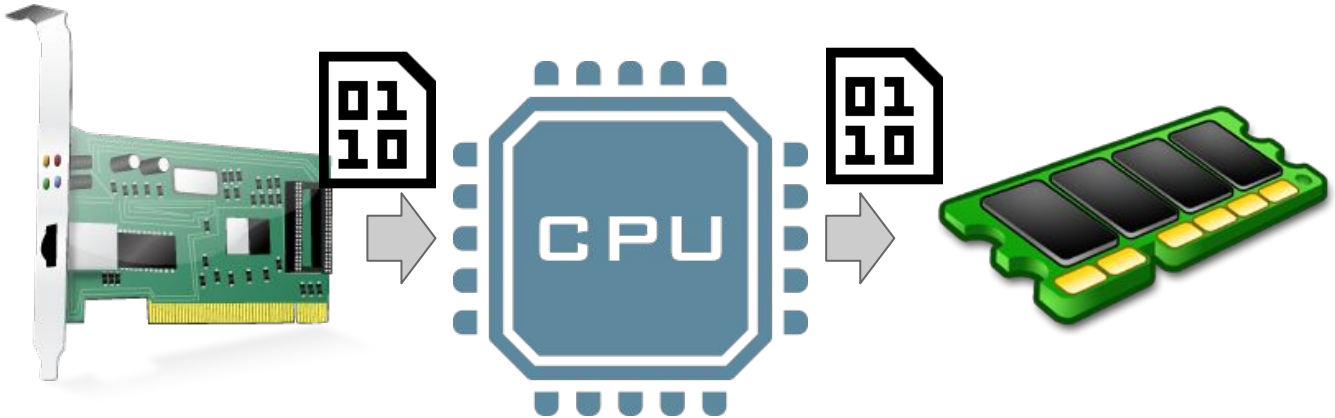
Event programming

우리는 이벤트 세상에 살고 있다.

Event programming

H/W

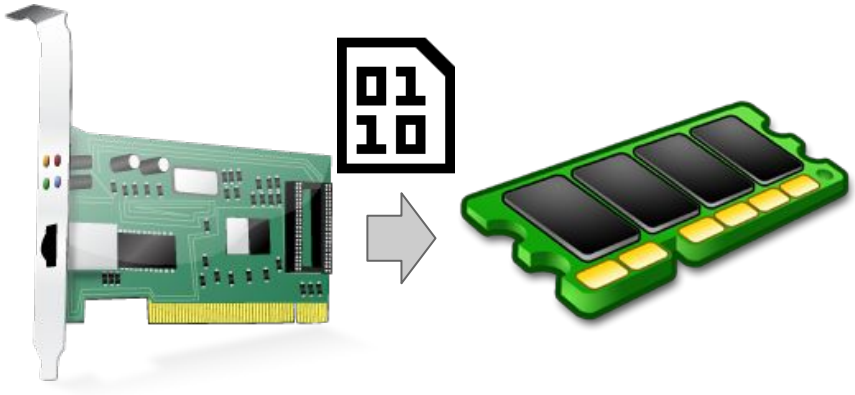
-> PIO(programmed I/O)



Event programming

H/W

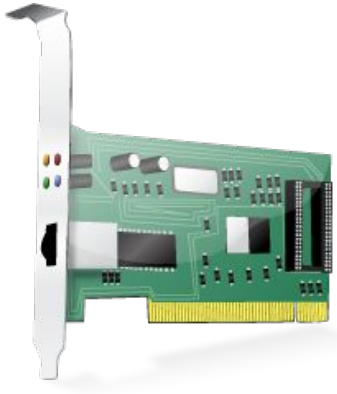
-> DMA(direct memory access)

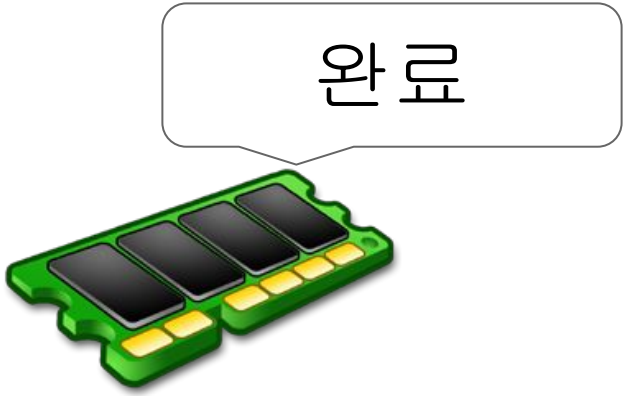


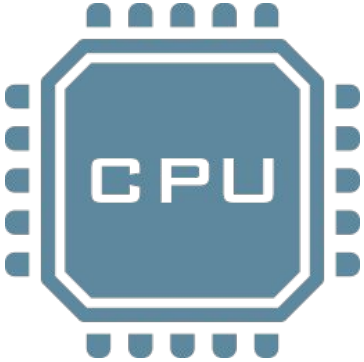
Event programming

H/W

-> DMA(direct memory access)



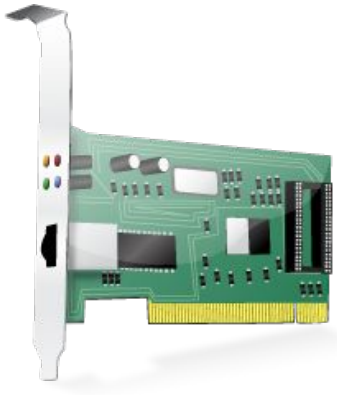


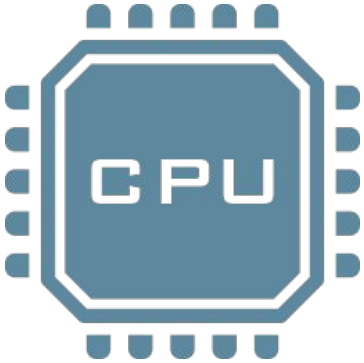


Event programming

H/W

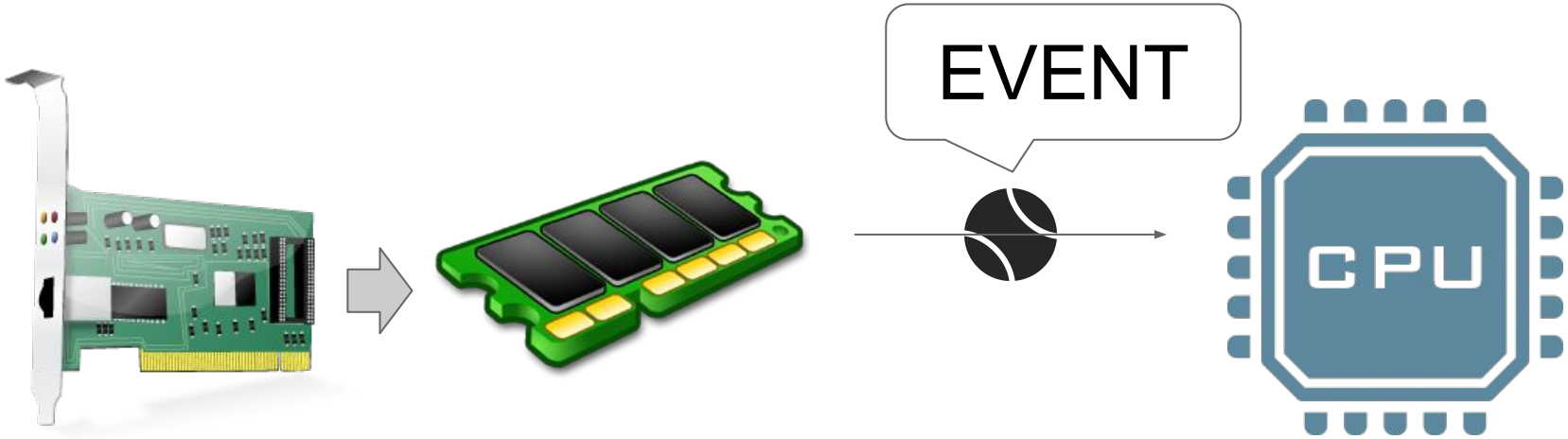
-> DMA(direct memory access)





Event programming

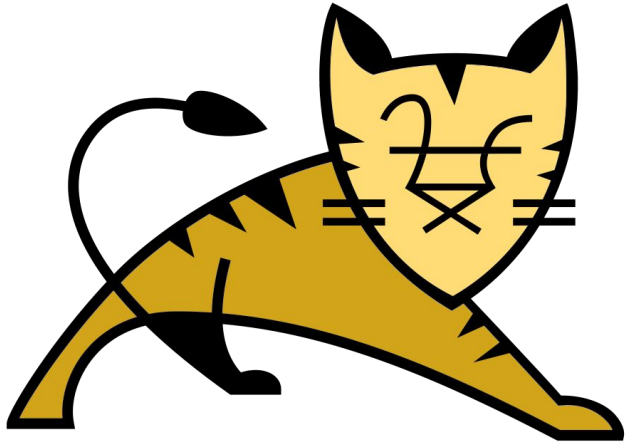
H/W는 이벤트 기반이다
-> DMA(direct memory access)



Event programming

Servlet

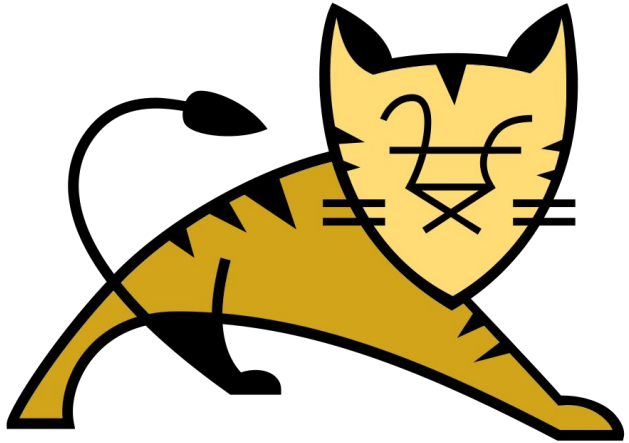
Request



Event programming

Servlet

Response



Event programming

Servlet request도 이벤트

정리

- 의문
  - 이벤트 세상에서 절차형으로 프로그래밍하려고하니 오히려 어려운게 아닐까?

정리

- 의문
  - 이벤트를 기반으로한 프로그래밍이 오히려 편해지지 않을까하는 생각이 든다.

정리

- ~~의문~~
  - 프로그래밍 패러다임중 핫한 리액티브 프로그래밍도 이벤트 기반 프로그래밍의 한축

정리

- 의문
  - 이벤트 프로그래밍을 위해 잘 추상화된 API가 있다면?

정리

- 의문
  - 이런 이벤트 프로그래밍관점에서 잘 추상화된 API가 있다면?
    - observable, Rxjava, Reactor, sodium, JDK9 Flow, AKKA...

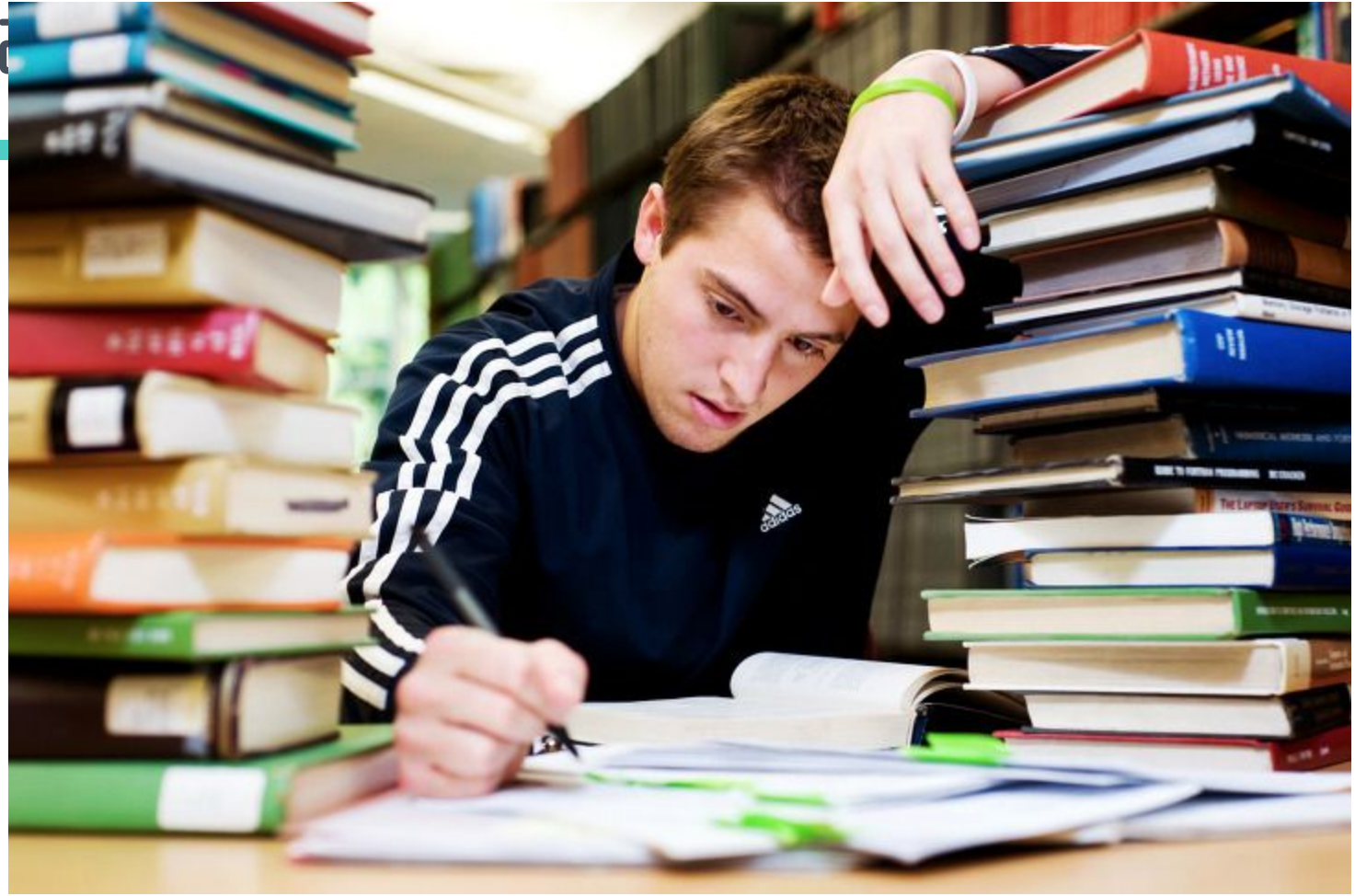
정리

- 의문
  - 이런 이벤트 프로그래밍관점에서 잘 추상화된 API가 있다면?
    - observable, Rxjava, Reactor, sodium, JDK9 Flow...
  - 그럼 웹개발에 활용할 프레임워크 쪽은?

정리

- 의문
  - 이런 이벤트 프로그래밍관점에서 잘 추상화된 API가 있다면?
    - observable, Rxjava, Reactor, sodium, JDK9 Flow, Vert.x core
  - 그럼 웹개발에 활용할 프레임워크 쪽은?
    - Spring 5 MVC, AKKA-http, Vert.x Web, Playframework...

정



정리

- 범위를 좁혀보자

정리

- 범위를 좁혀보자
  - 현장에서 가장 많이 쓰는 자바, 스프링

정리

- 범위를 좁혀보자
  - 현장에서 가장 많이 쓰는 자바, 스프링
  - 리액티브 프로그래밍

정리

- 범위를 좁혀보자
  - 현장에서 가장 많이 쓰는 자바, 스프링
  - 리액티브 프로그래밍
    - reactive-streams
    - Reactor
    - Spring 5 MVC

정리

- 범위를 좁혀보자
  - 현장에서 가장 많이 쓰는
  - 리액티브 프로그래밍
    - reactive-streams
    - Reactor
    - Spring 5

정리

마지막으로 성능!

정리

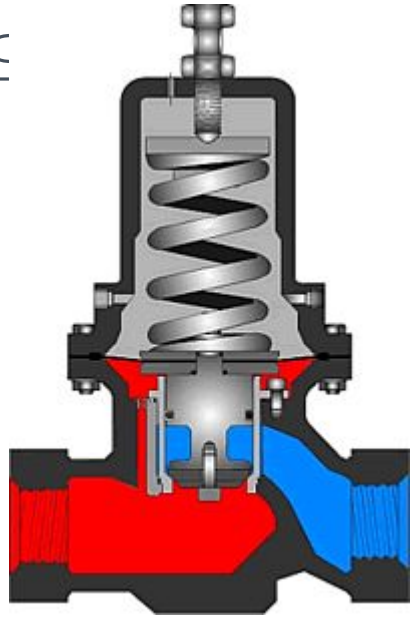
- 성능
  - Nonblocking은 이미 많이 이야기했으니...
  - backpressure

정리

- 성능
  - Nonblocking은 이미 많이 이야기했으니...
  - backpressure
    - back pressure valve(배압밸브)

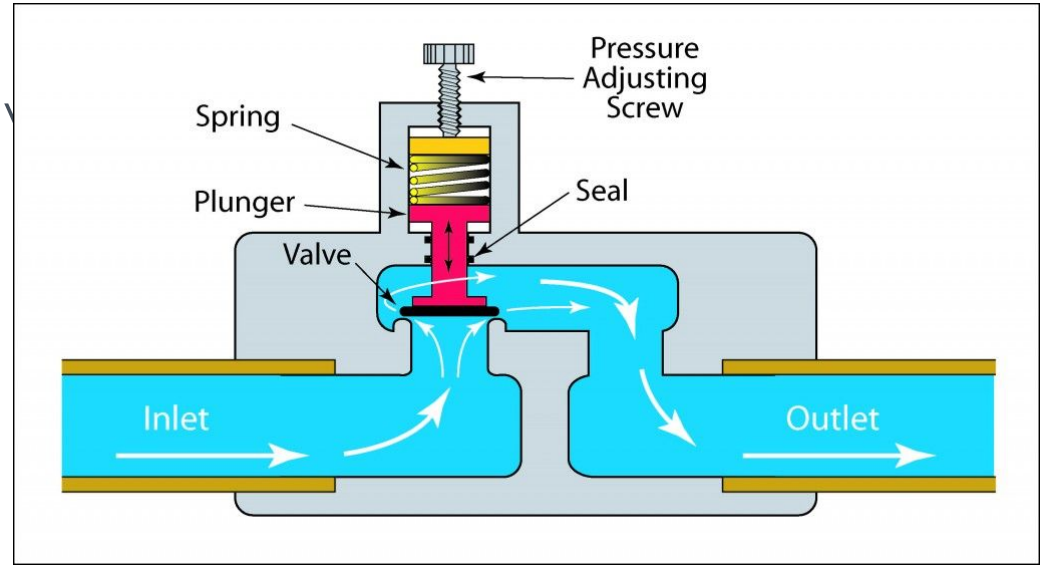
정리

- 성능
  - Nonblocking은 이미 많이 이야기했으
  - backpressure
    - back pressure valve(배압밸브)



정리

- 성능
  - Nonblocking은 이미 많이 이야기했으니...
  - backpressure
    - back pressure v



서킷브레이커랑...

Thanks!

Any questions?

Q&A

콩콩

예제 : https://github.com/boojongmin/presentations