

프로세스, 스레드, 리액티브

부종민

Hello!

부종민



boojongmin@gmail.com

event

dum

실시간 8 윤제문 ↑ 89

뉴스 랭킹 연예 스포츠 쇼핑 1boon 홈&쿠팡 스타일


조기 대선에 몰려드는 후보자들..역대 최다

文 스탠딩 수용 "서서 하나 앉아서 하나 무슨 상관"

22번째 촛불, 세월호 추모를 위해 타올랐다

국민연금, 산은에 "대우조선 망해도 상환보증해라"

포항 또 규모 2 지진..경주 등 3차례 발생


북한 태양절 열병식에 SLBM
첫 등장


세월호 사진전.. 훼손되고, 탈
취당하고

목차



Process, Thread
Java Async API
NIO
Reactive

1.

Process, Thread

들어가며..
Thread 일반
Thread in JVM

들어가며..

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

들어가며..

- 프로세스, 스레드
 - 대학교 2~3학년때 배우는 개념.
 - 현장...
 - 이론...

들어가며..

- 프로세스, 스레드
 - 대학교 2~3학년때 배
 - 현장...
 - 이론...



Thread 일반

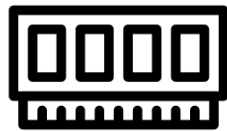
프로세스 - 독립된 실행의 단위

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

Thread 일반

- (현대) 프로세스
 - 프로세스 := 스레드 + 메모리공간

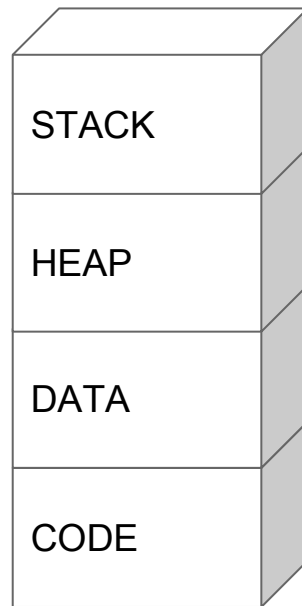
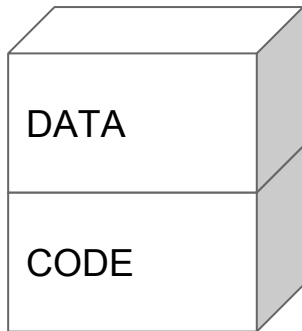
Thread 일반



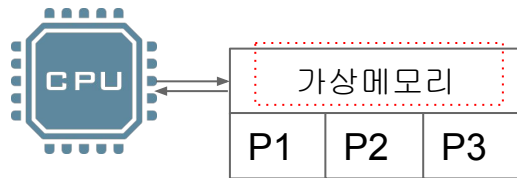
프로세스



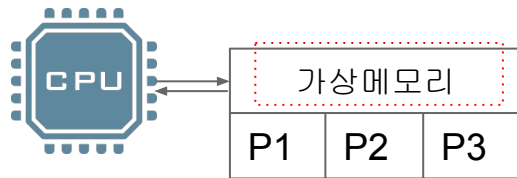
프로그램



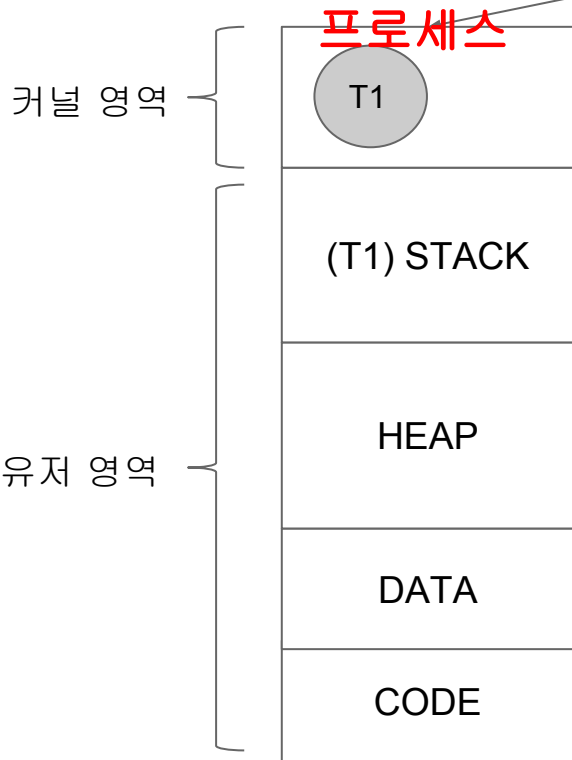
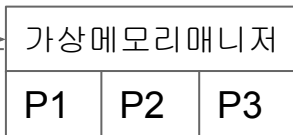
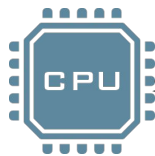
Thread 일반



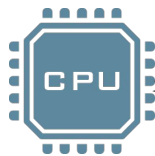
Thread 일반



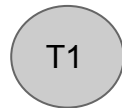
Thread 일반



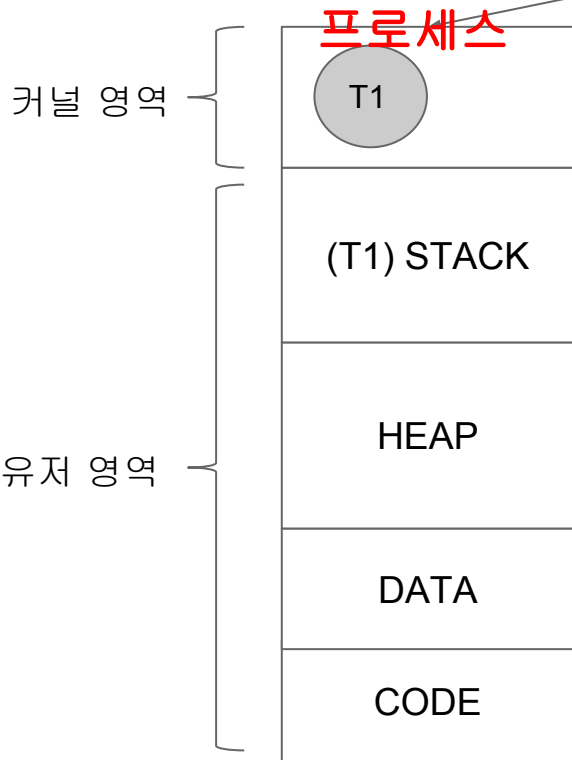
Thread 일반



가상메모리매니저		
P1	P2	P3

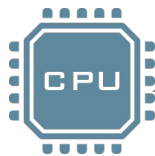


task_s
truct

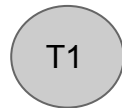


스케줄링 대상

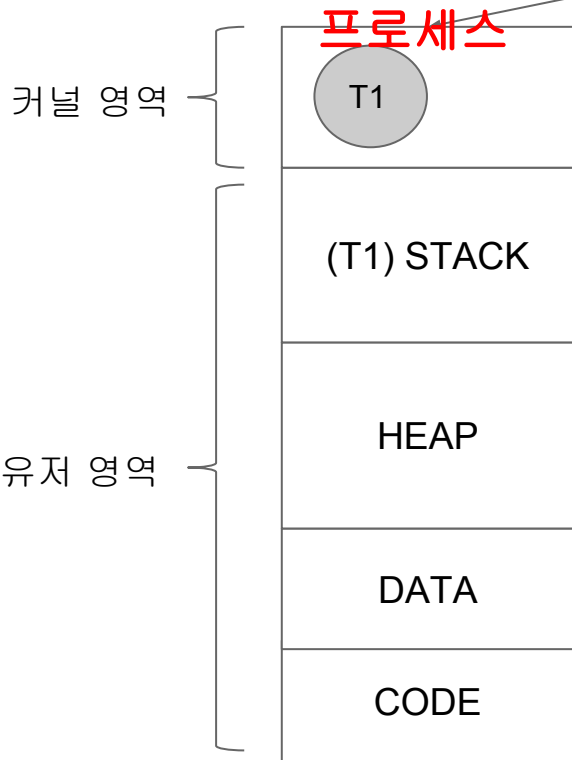
Thread 일반



가상메모리매니저		
P1	P2	P3

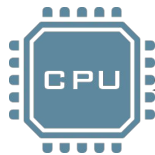


task_s
truct



스케줄링 대상

Thread 일반



가상메모리매니저

P1

P2

P3

프로세스

T1

(T1) STACK

HEAP

DATA

CODE

프로세스

T1

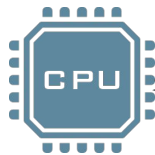
(T1) STACK

HEAP

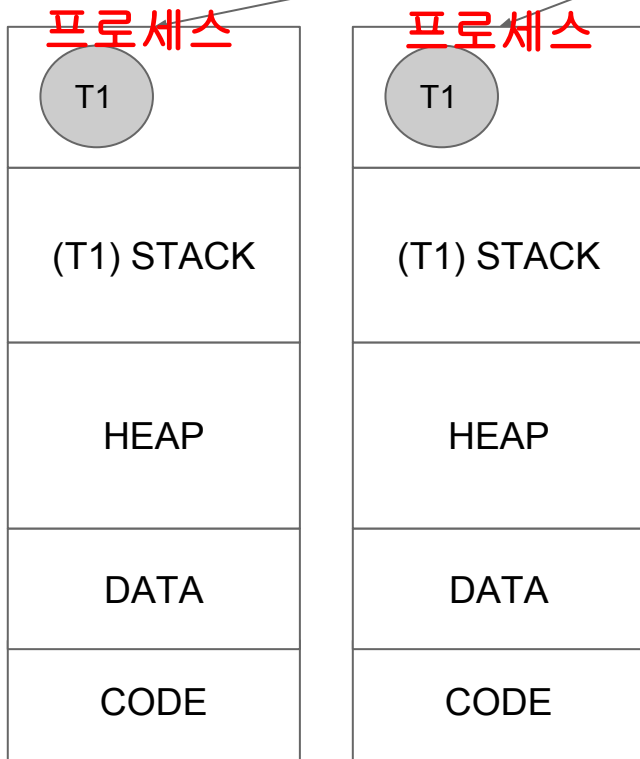
DATA

CODE

Thread 일반

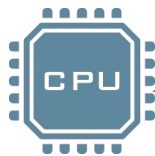


가상메모리매니저		
P1	P2	P3



멀티 프로세스

Thread 일반



가상메모리매니저

P1

P2

P3

프로세스

T1

(T1) STACK

HEAP

DATA

CODE

프로세스

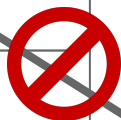
T1

(T1) STACK

HEAP

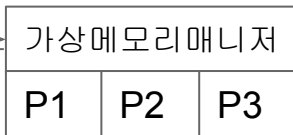
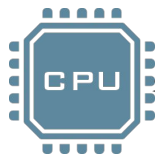
DATA

CODE

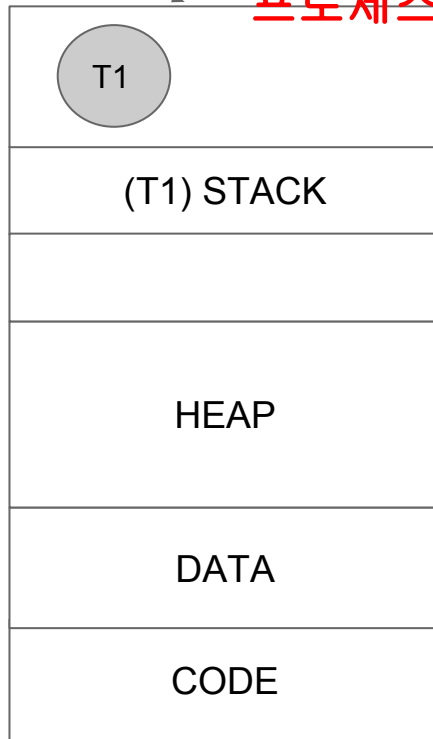


가상메모리는
프로세스간
주소공간을
논리적으로 차단

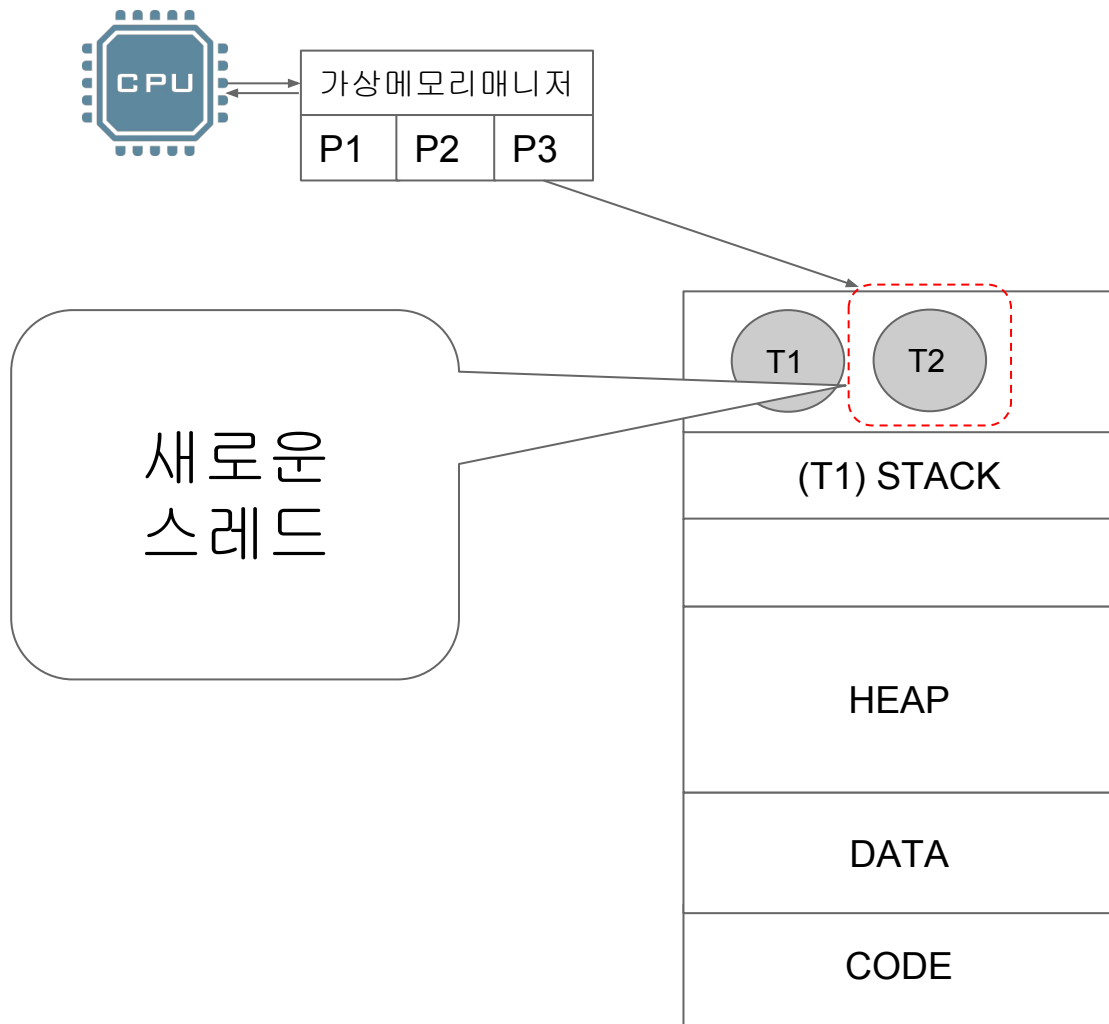
Thread 일반



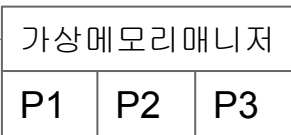
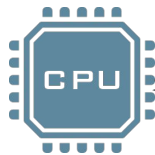
프로세스



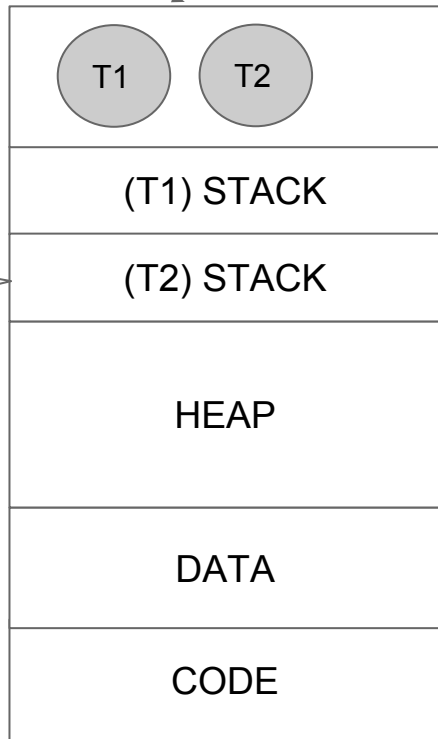
Thread 일반



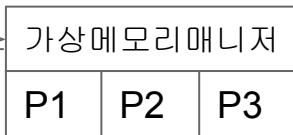
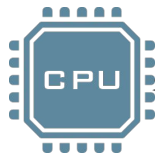
Thread 일반



신규 스레드가
사용할
STACK 메모리공간
생성

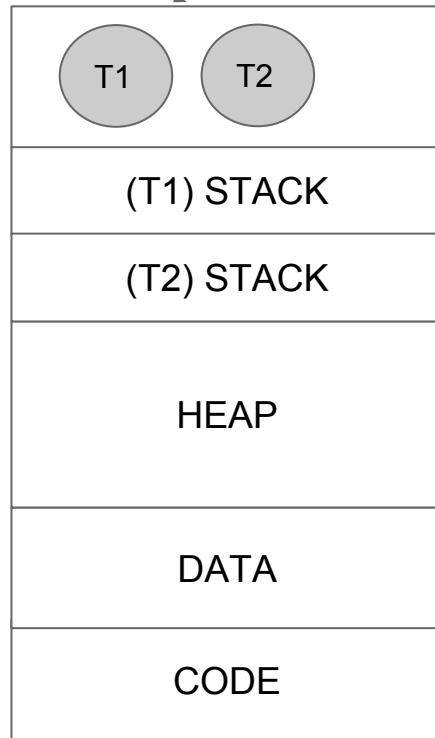


Thread 일반

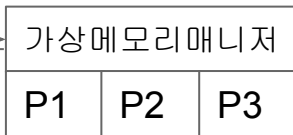
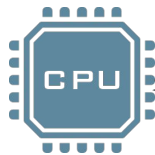


스레드 장점

스레드 생성에
비용이 필요



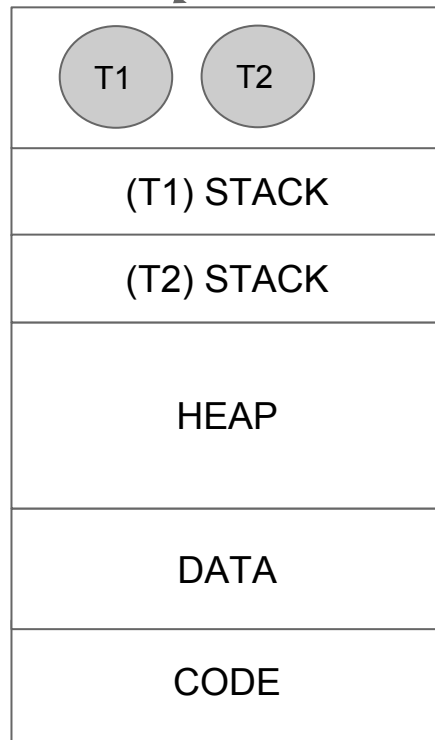
Thread 일반



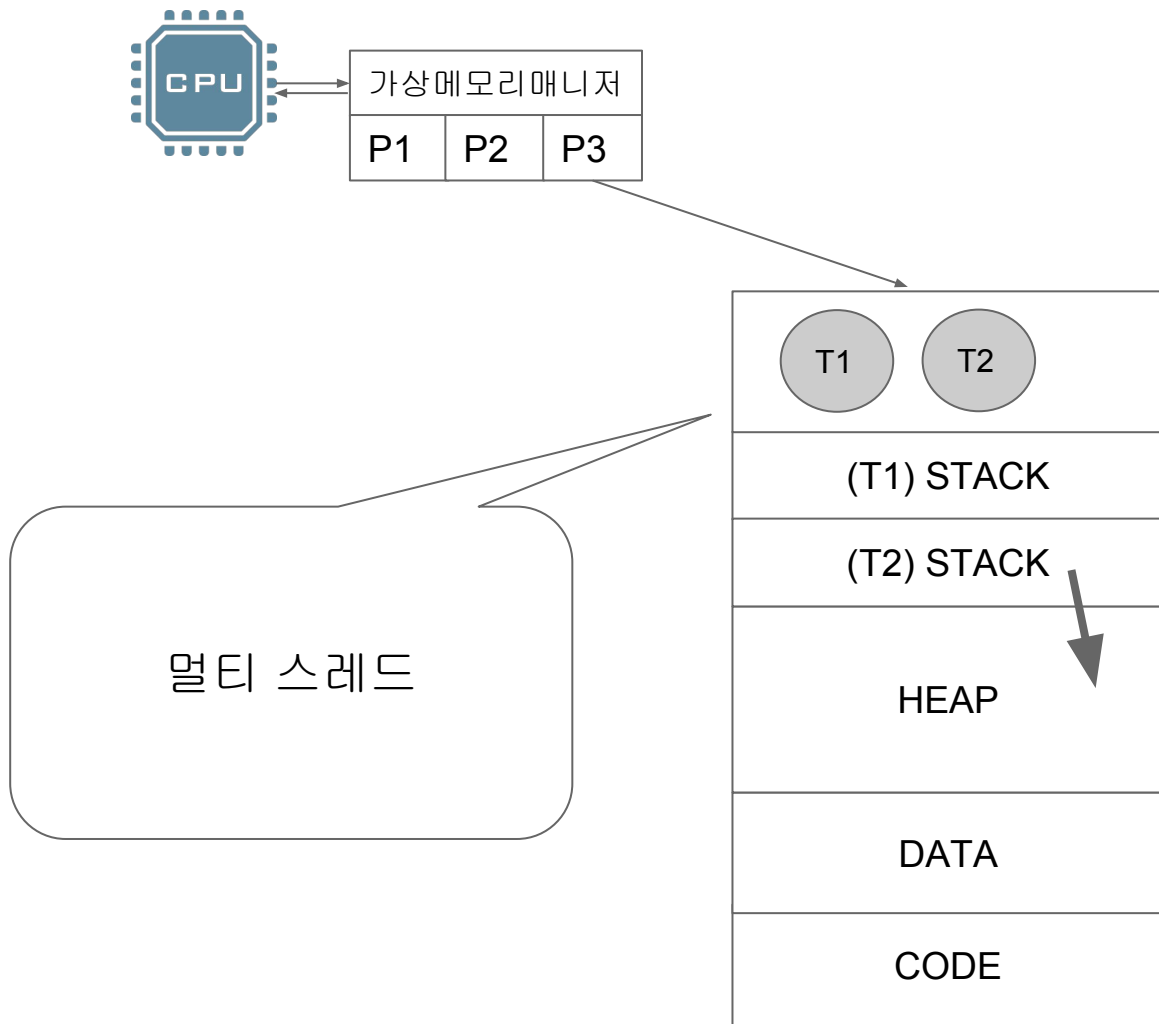
스레드 장점

프로세스를 만드는
것보다 저렴

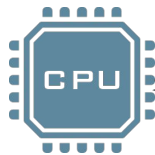
프로세스



Thread 일반



Thread 일반



가상메모리매니저

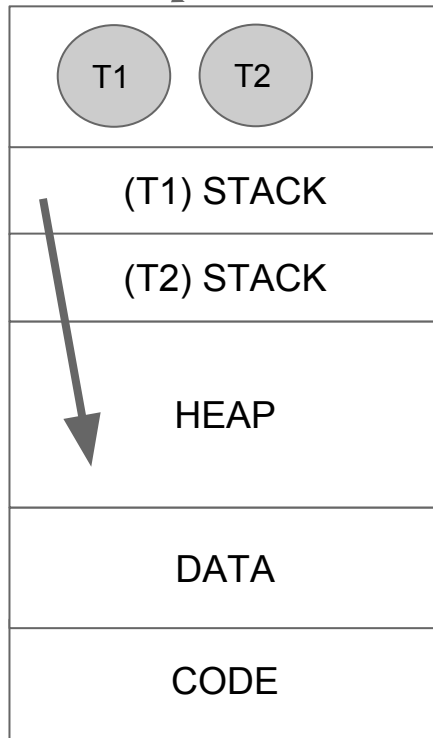
P1

P2

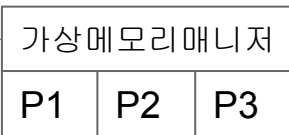
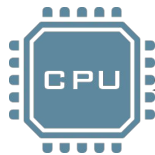
P3

스레드 장점

스레드들간
HEAP 메모리 공간
공유

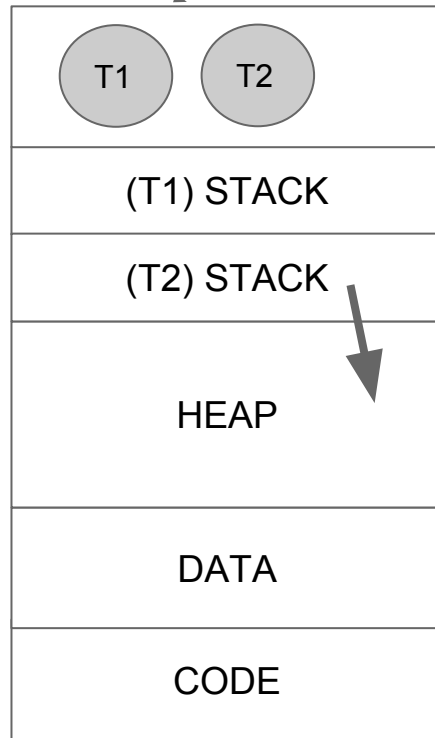


Thread 일반



스레드 장점

스레드들간
HEAP 메모리 공간
공유
(코드, 성능)

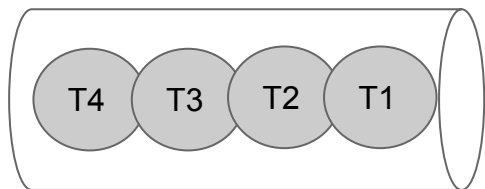


Thread 일반

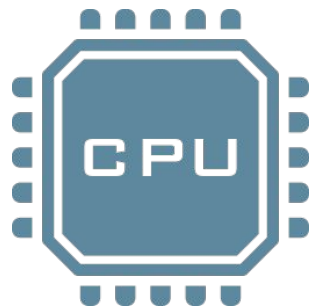
Context Switch

- 스케줄러에 의해 스레드는 **time slice**만큼 **CPU**를 사용한 후 작업을 멈추고 **CPU**를 다음 스레드가 점유하는 것
 - 과정
 - 저장 - 반납되는 스레드는 마지막 상태를 메모리에 저장
 - 복원 - 실행되는 스레드는 메모리에서 저장된 상태를 복구

Thread 일반



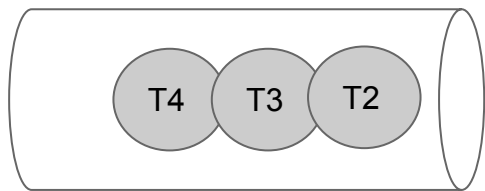
스케줄러



Thread 일반



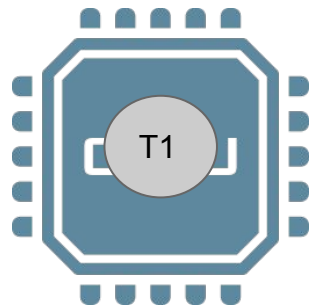
Thread 일반



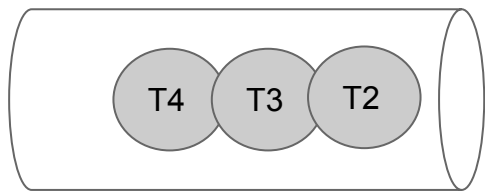
스레드큐



스케줄러



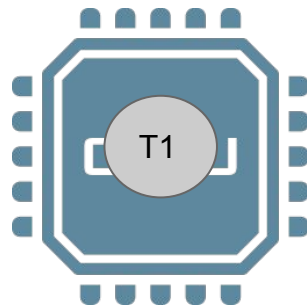
Thread 일반



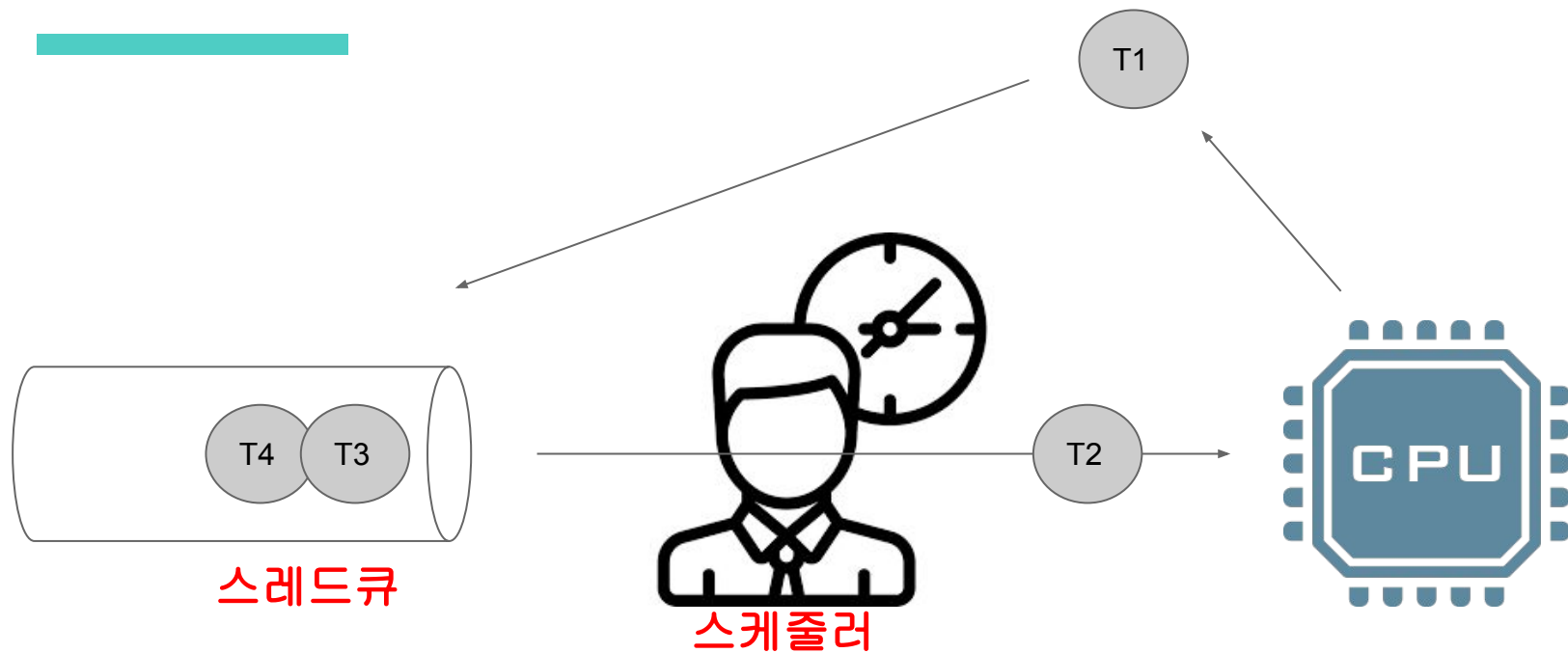
스레드큐



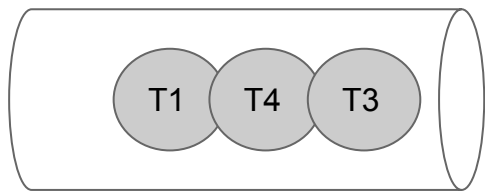
스케줄러



Thread 일반



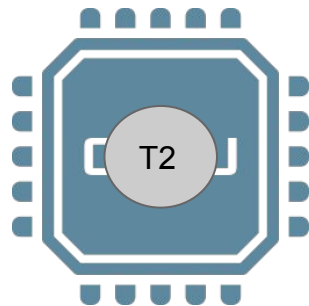
Thread 일반



스레드큐



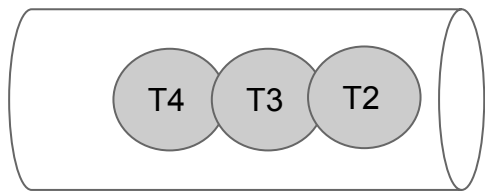
스케줄러





context switch 오버헤드?

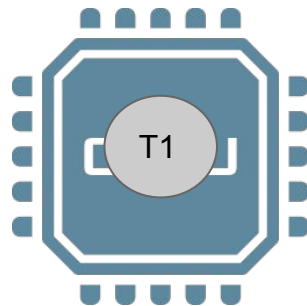
Thread 일반



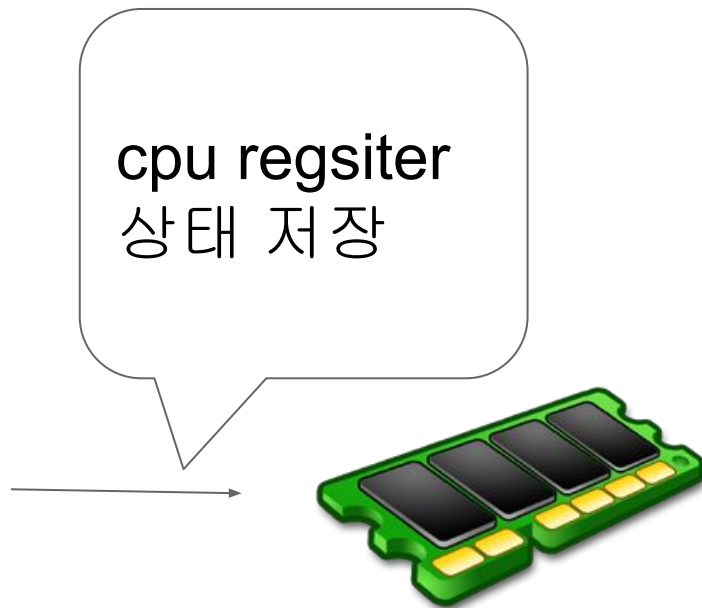
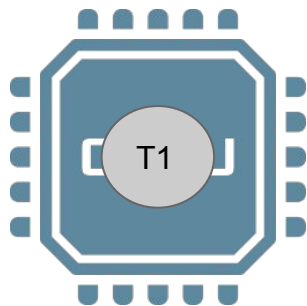
스레드큐



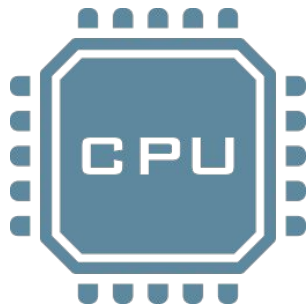
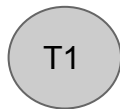
스케줄러



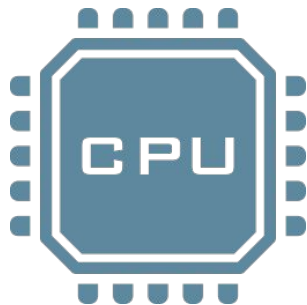
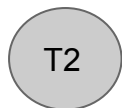
Thread 일반



Thread 일반

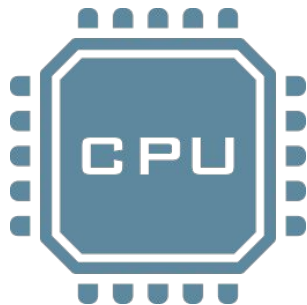


Thread 일반



Thread 일반

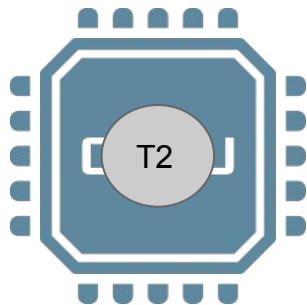
T2



이전에
저장된
cpu register
상태 복원



Thread 일반





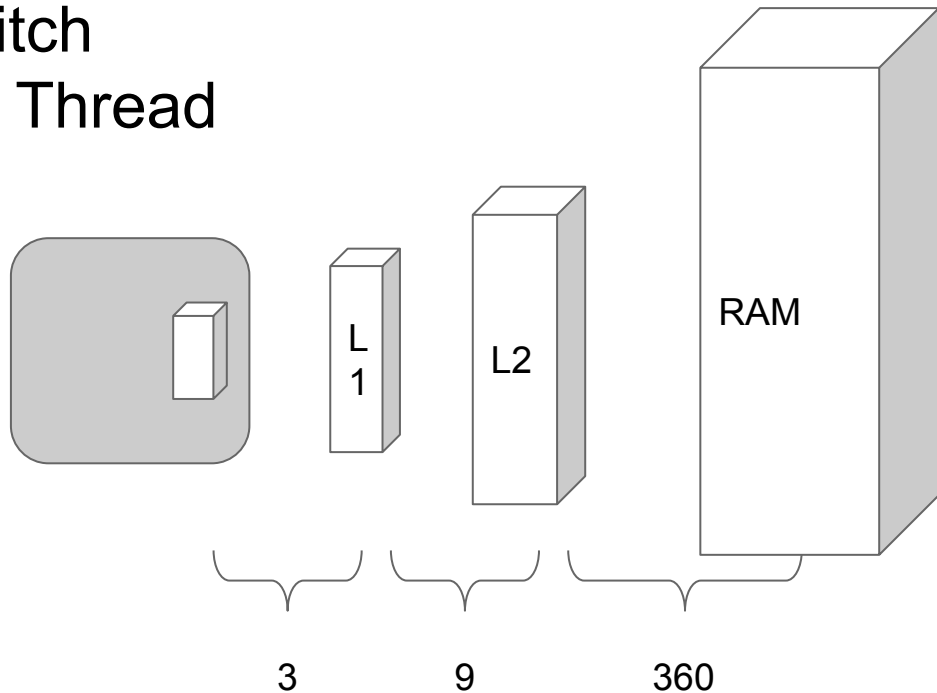
ContextSwitch 비용

멀티 프로세스 vs 멀티
스레드

(캐시...)

Thread 일반

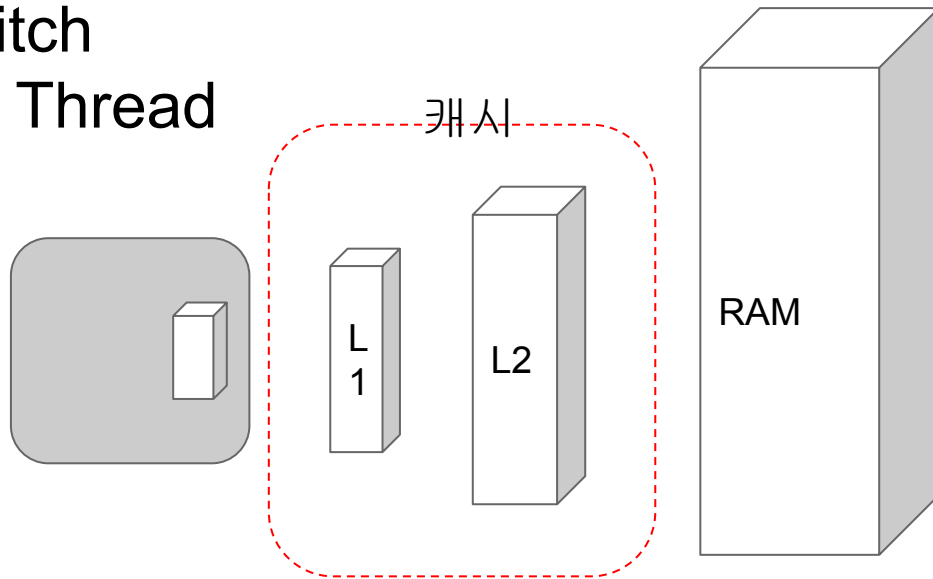
ContextSwitch Process vs Thread



램은
cpu clock
속도보다
많이~~~~
느리다...

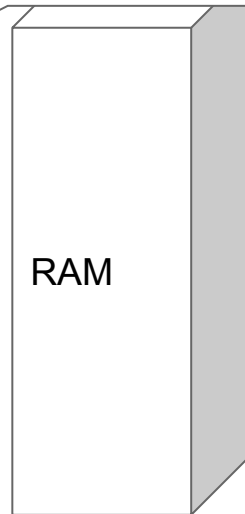
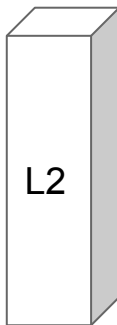
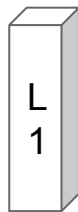
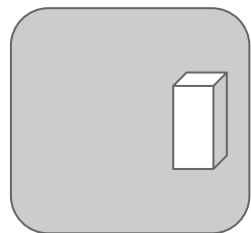
Thread 일반

ContextSwitch Process vs Thread



Thread 일반

ContextSwitch Process vs Thread



캐싱 측면에서 스레드
모델 성능이 좋다

Thread 일반

- ThreadPool

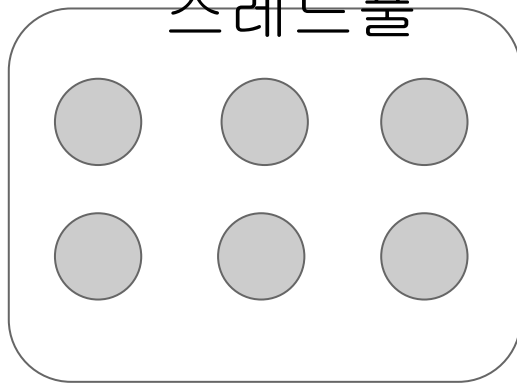
- 스레드를 미리 만들어 놓고 **재사용**하는 것

- 비용 감소

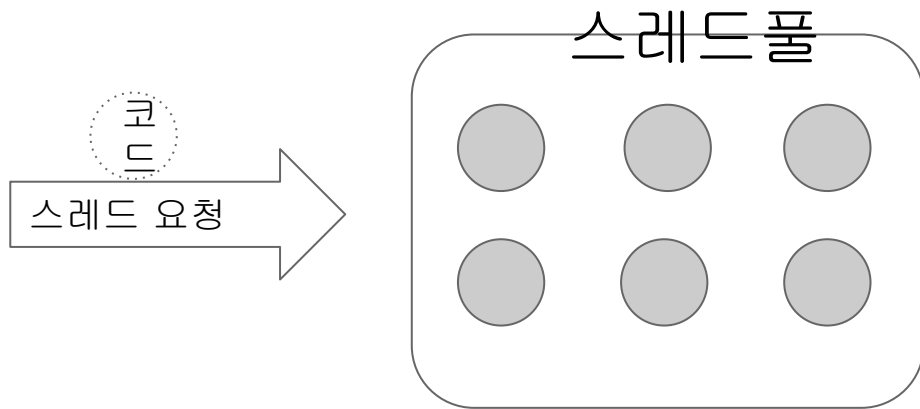
- 스레드 생성 비용 - cpu, memroy

Thread 일반

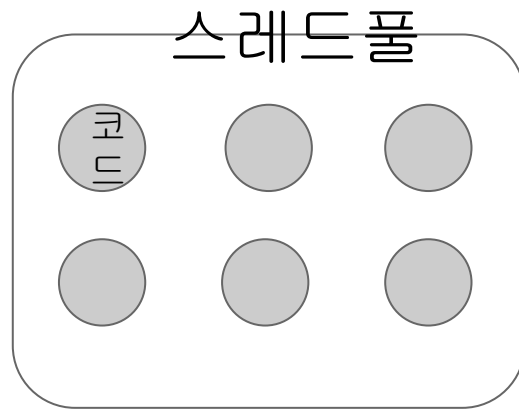
스레드풀



Thread 일반

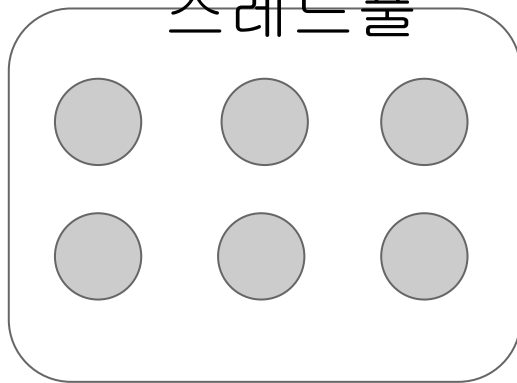


Thread 일반

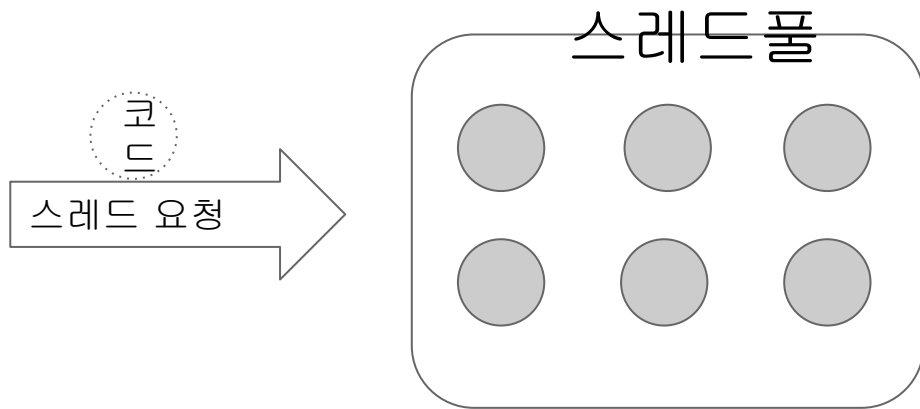


Thread 일반

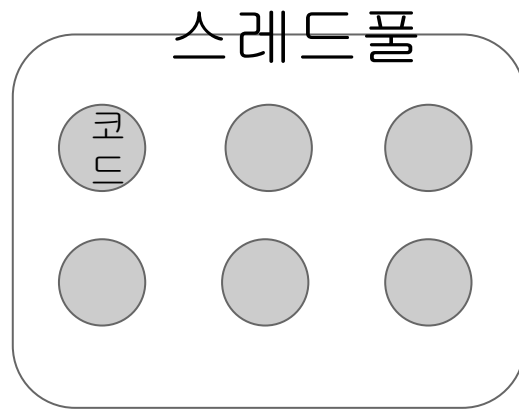
스레드풀



Thread 일반

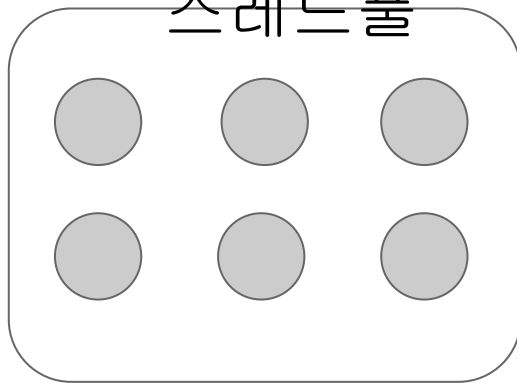


Thread 일반

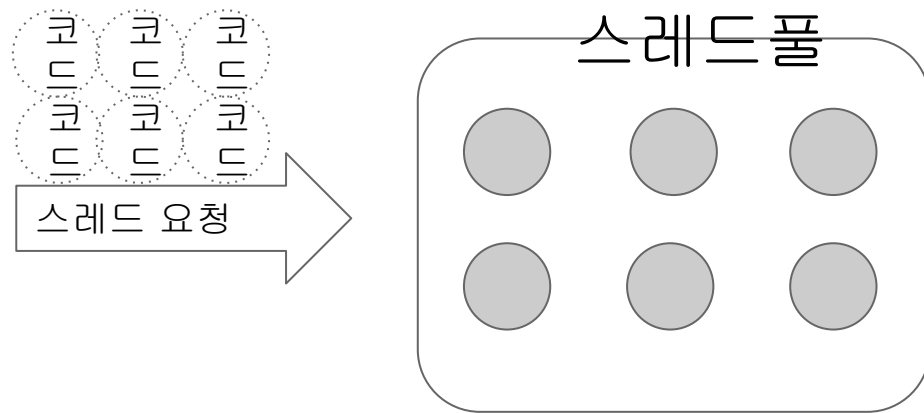


Thread 일반

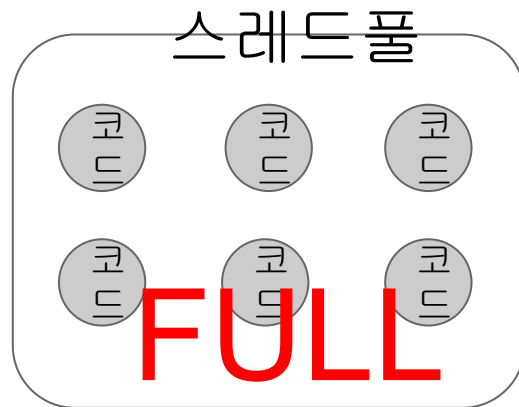
스레드풀



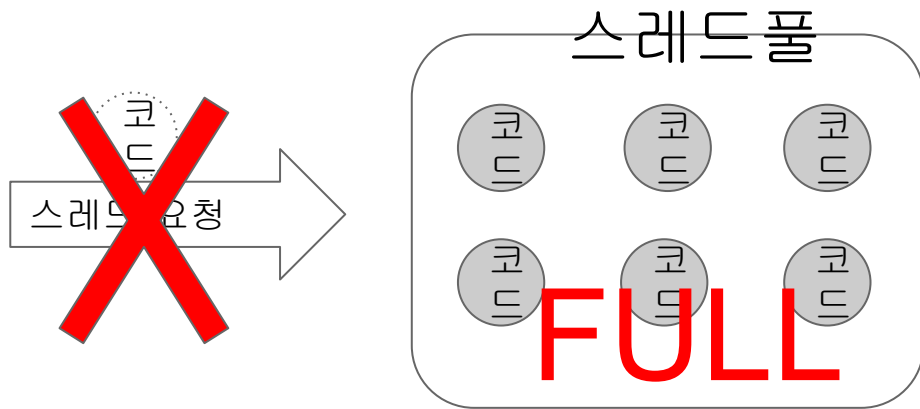
Thread 일반



Thread 일반



Thread 일반



Thread 일반

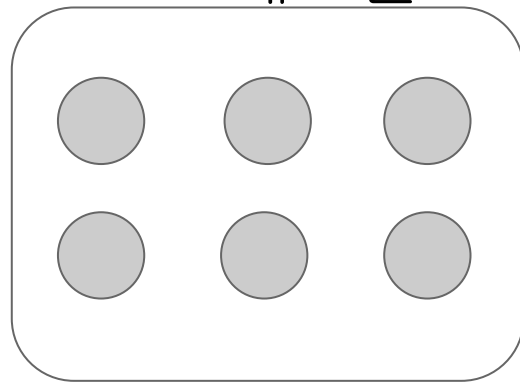
스레드큐



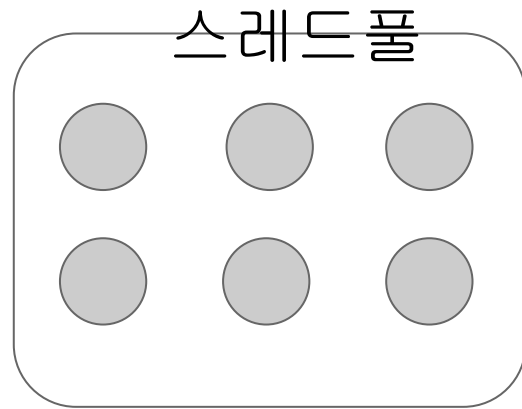
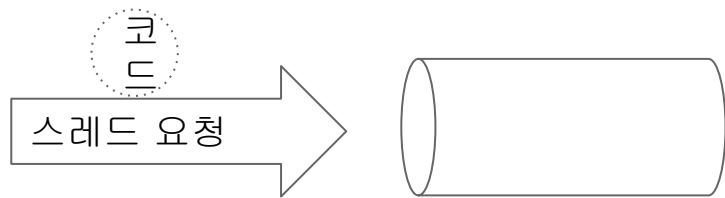
스레드풀
매니저



스레드풀



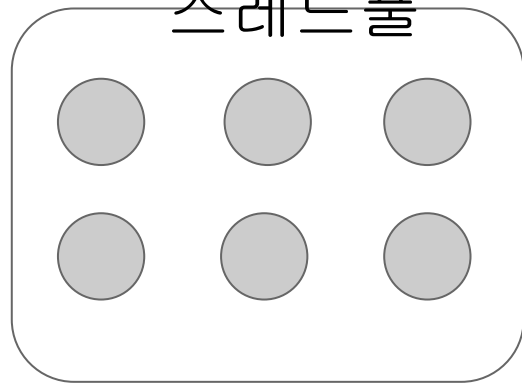
Thread 일반



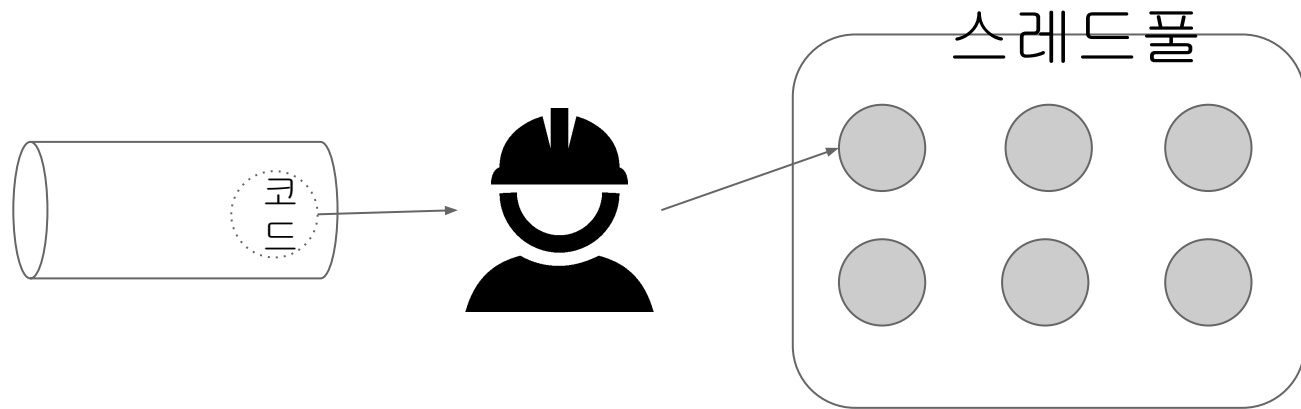
Thread 일반



스레드풀



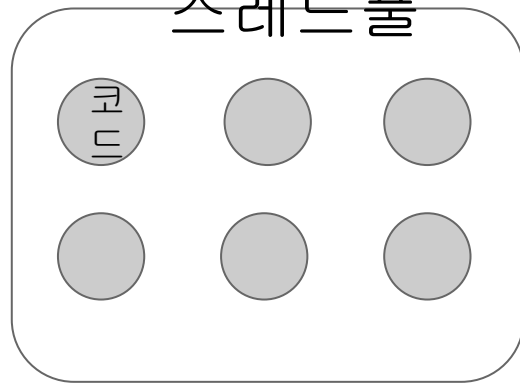
Thread 일반



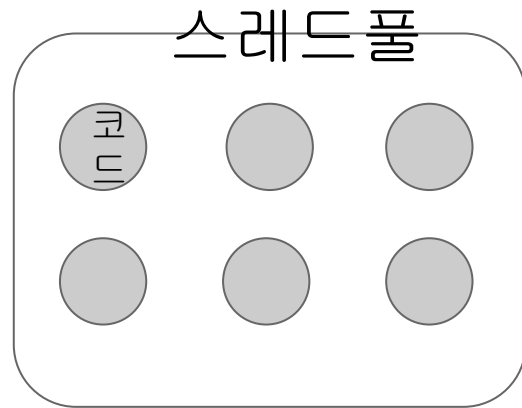
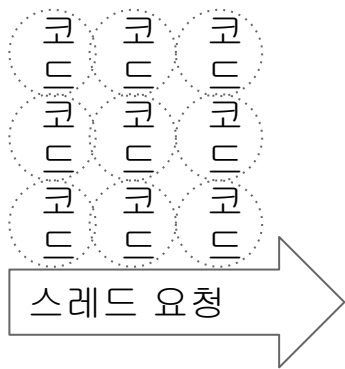
Thread 일반



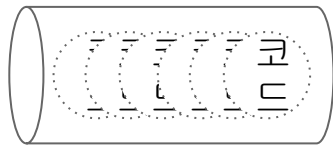
스레드풀



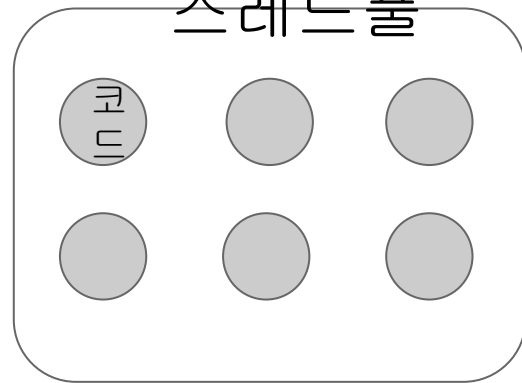
Thread 일반



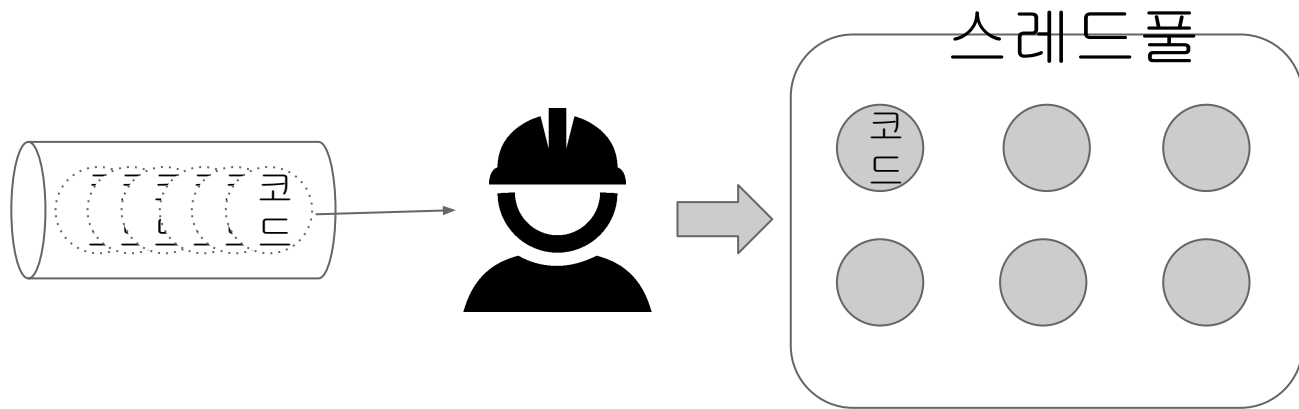
Thread 일반



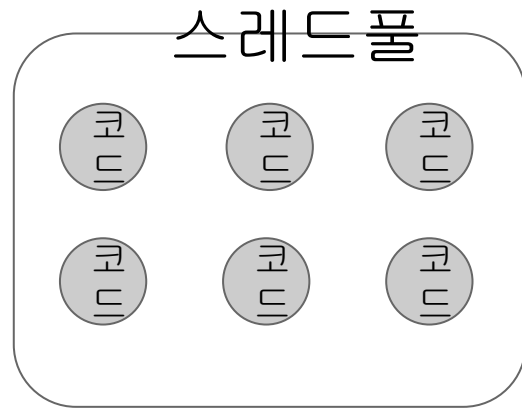
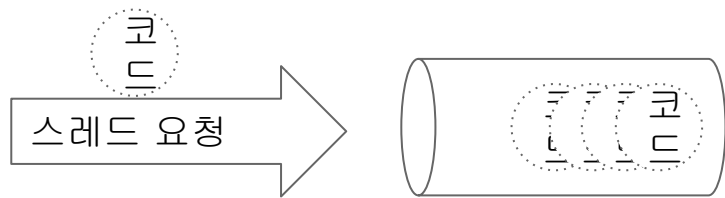
스레드풀



Thread 일반



Thread 일반





Thread in JVM

Thread in JVM

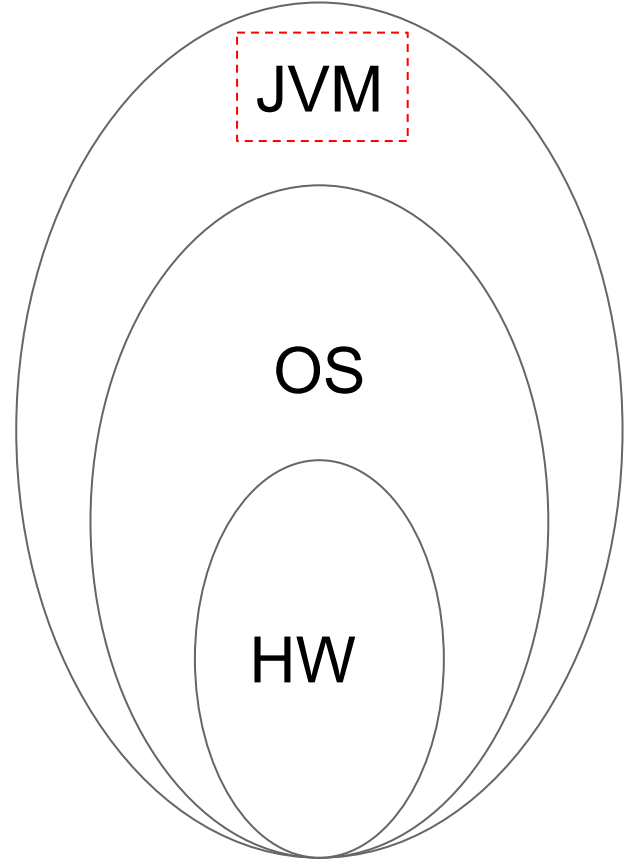


Java Virtual Machine

Thread in JVM

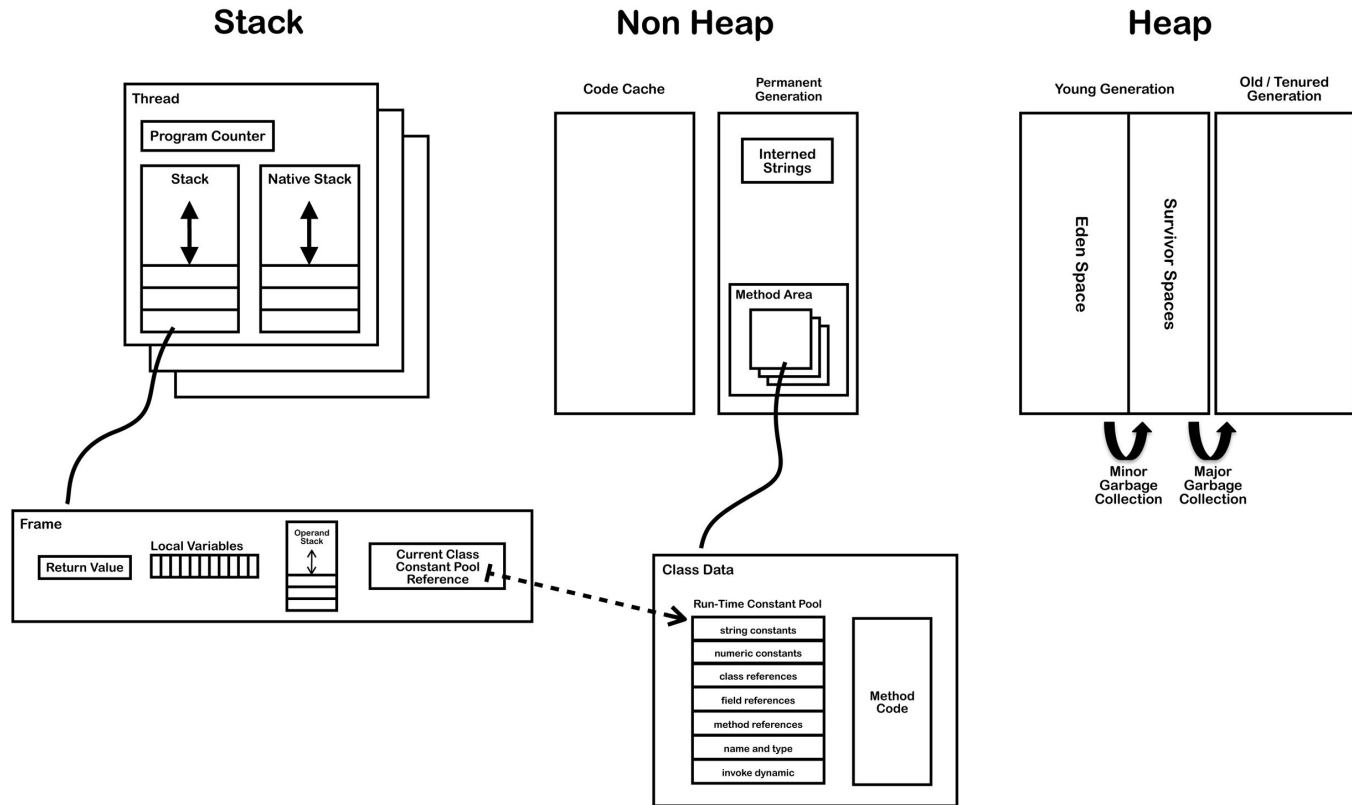


Java Virtual Machine



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.

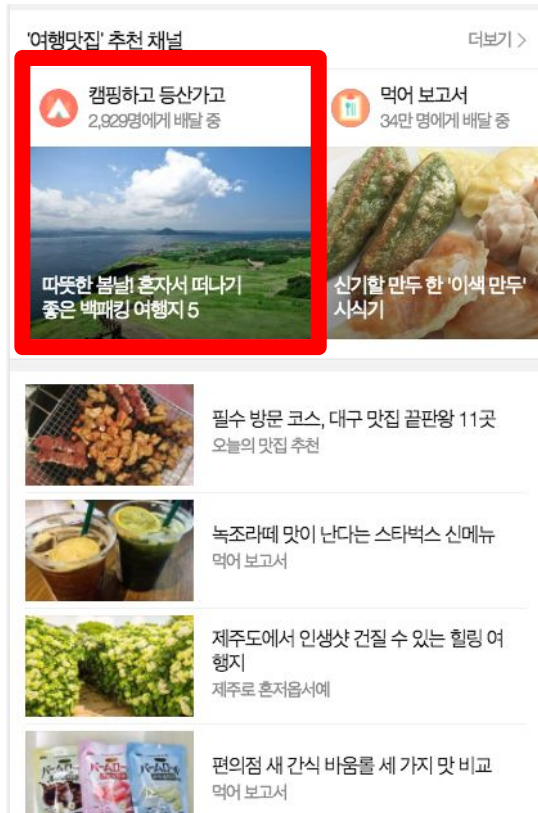
Java Async API

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, ethernet 등 I/O 자원은 느리다.
 - 쓰레드는 I/O 작업의 완료를 대기해야한다.

비동기가 필요한 상황





ㅋㅋㅋ 인기

클릭하자마자 웃음 빵~

42,140명에게 배달 중 | 파트너 >

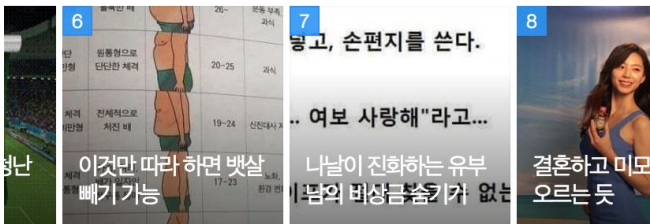
+ 채널 소식 배달 받기

공유

읽을거리

같이 볼만한 채널

인기 Best



전체 5,536



사회에서 여러모로 인정받는 인재

NEW

읽을거리

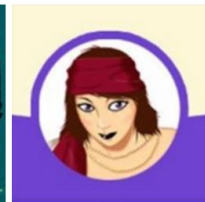
같이 볼만한 채널



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



인기 채널
API

채널 구독
API

인기 콘텐츠
API

동영상 API
공감 API

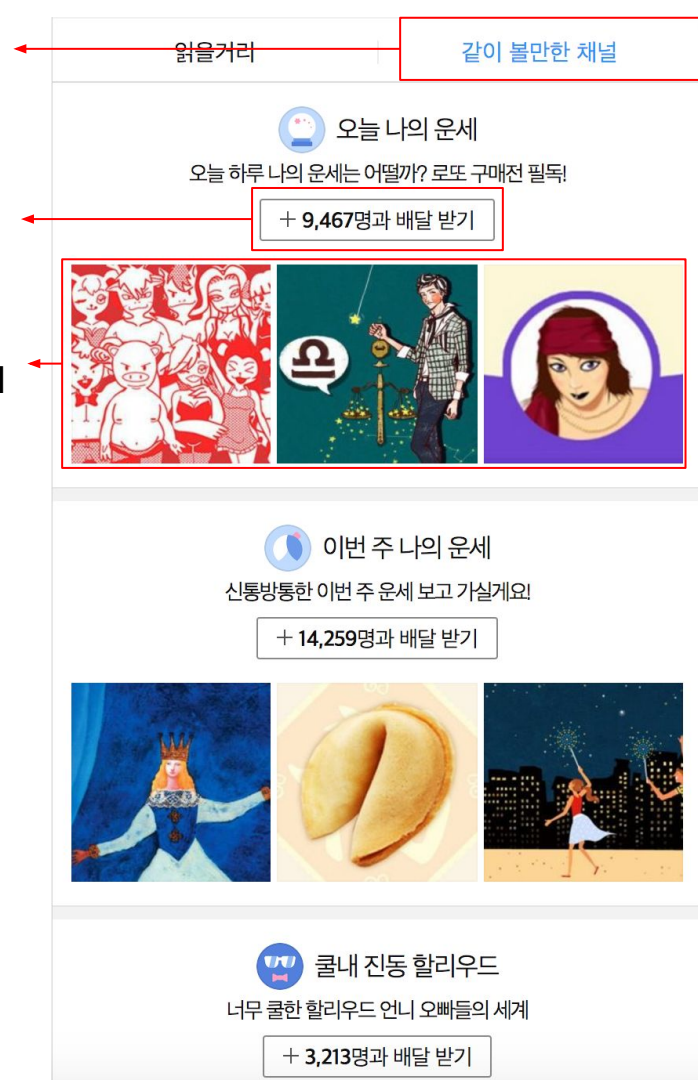


동영상 API
공감 API

연관 채널
API

연관 채널
구독 API

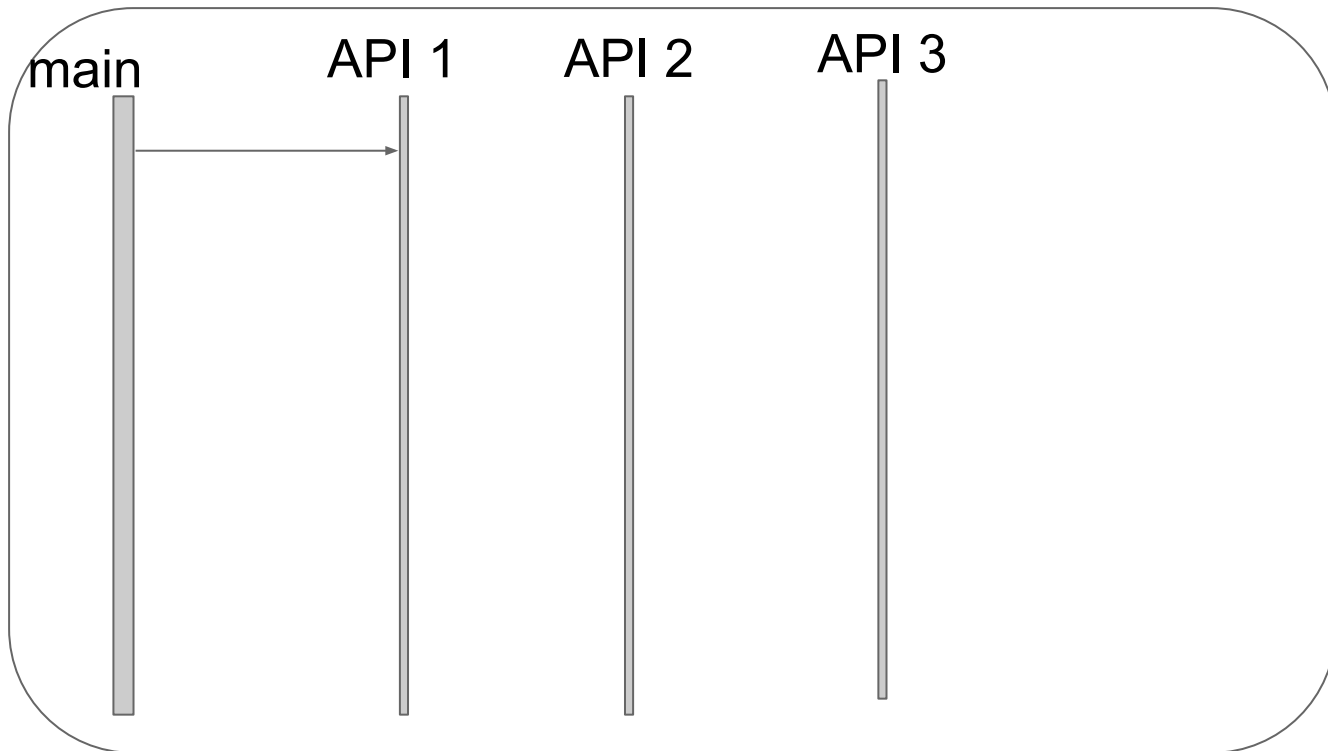
인기
컨텐츠 API



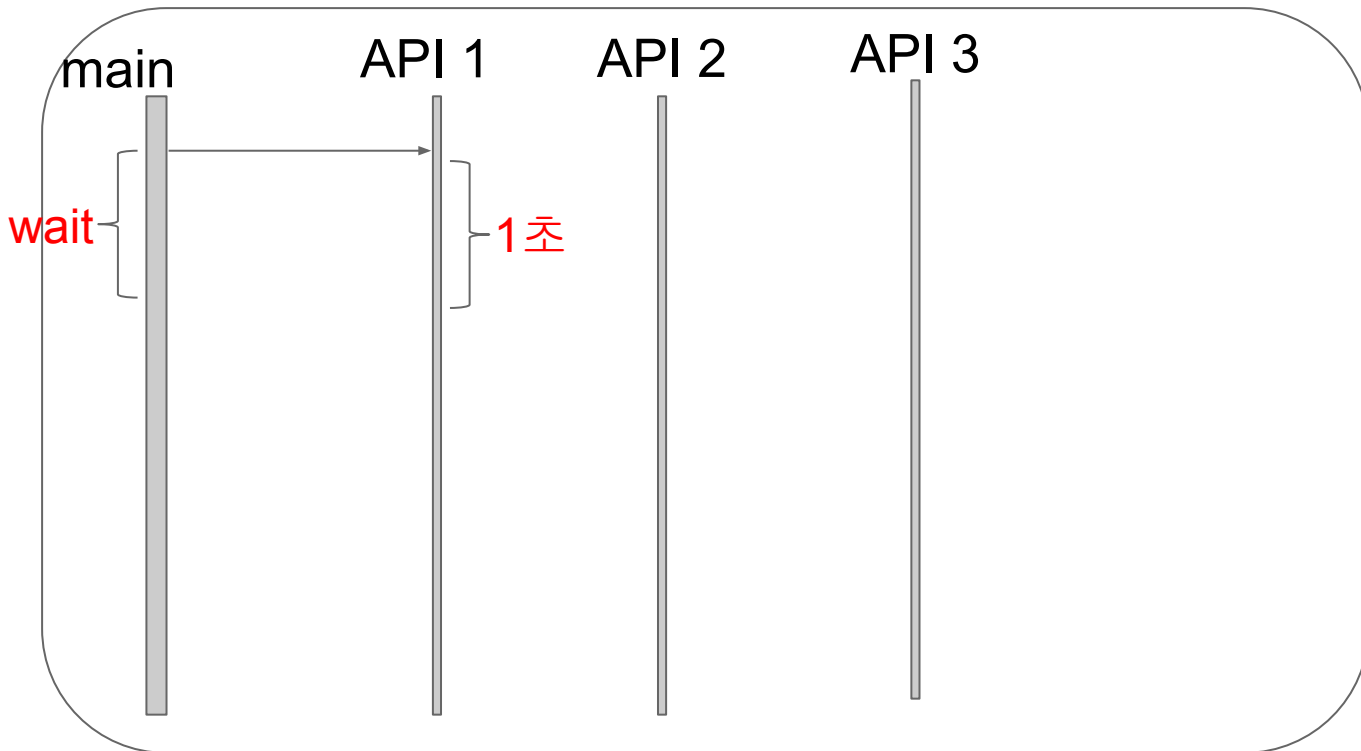


통신을 순차적으로 한다면?

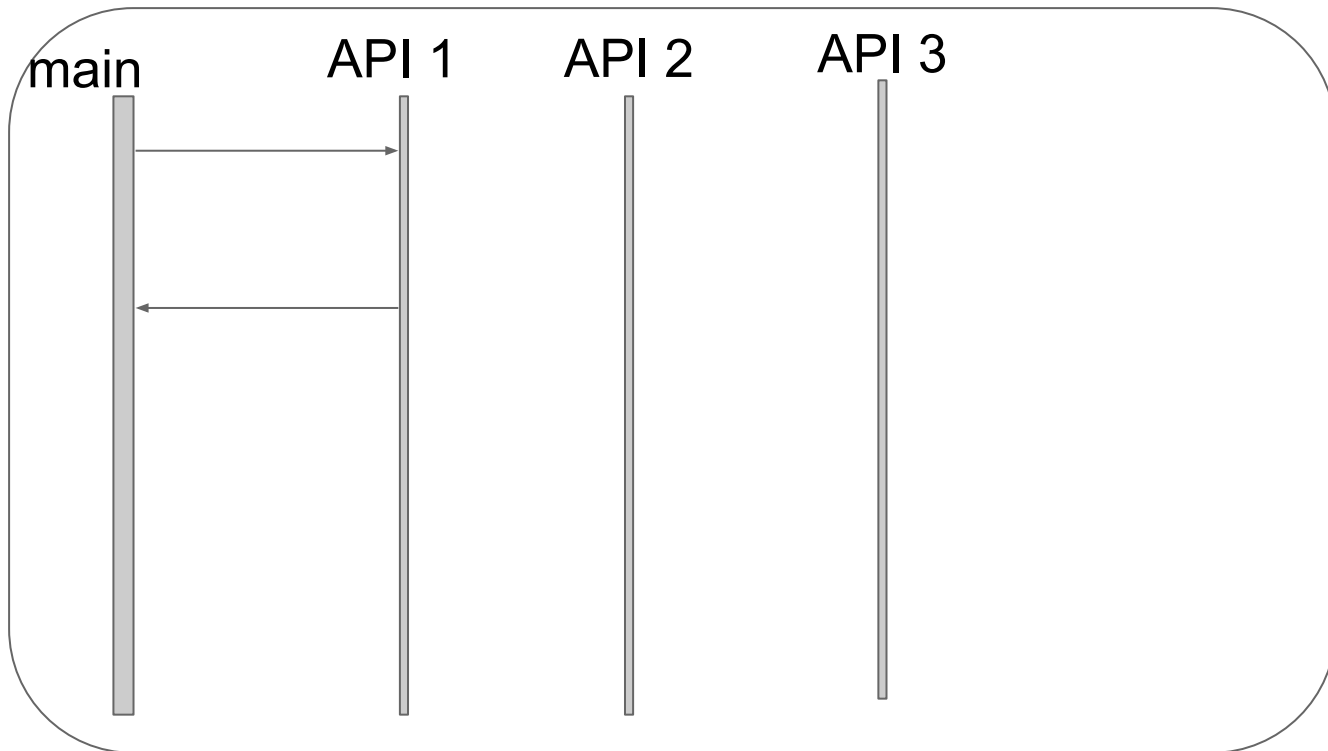
비동기가 필요한 상황



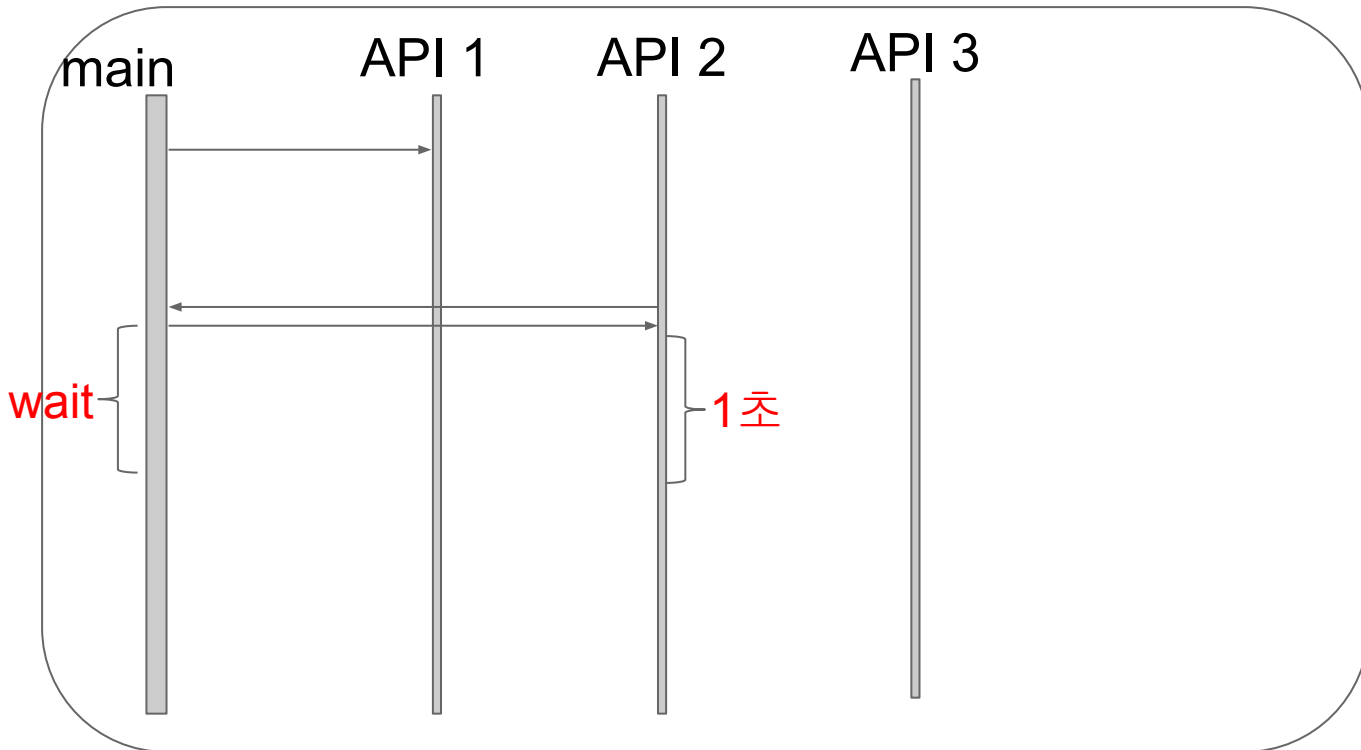
비동기가 필요한 상황



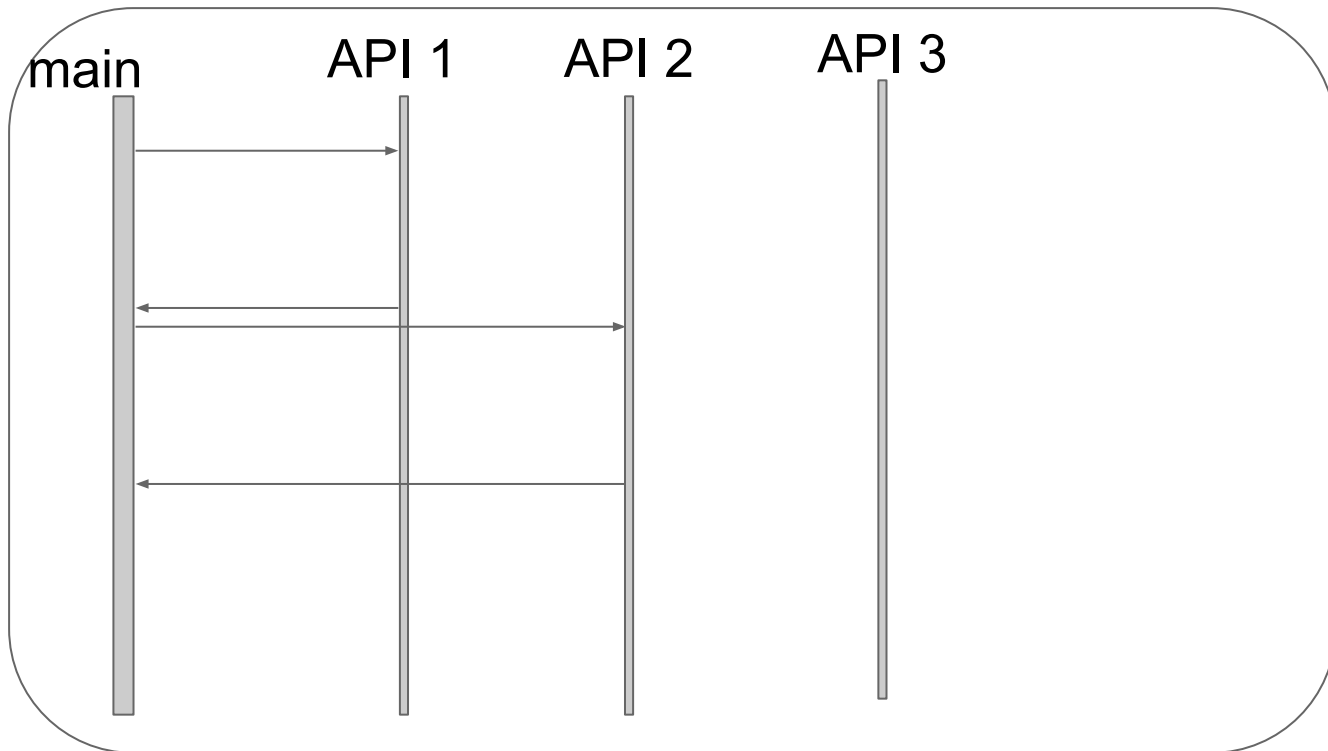
비동기가 필요한 상황



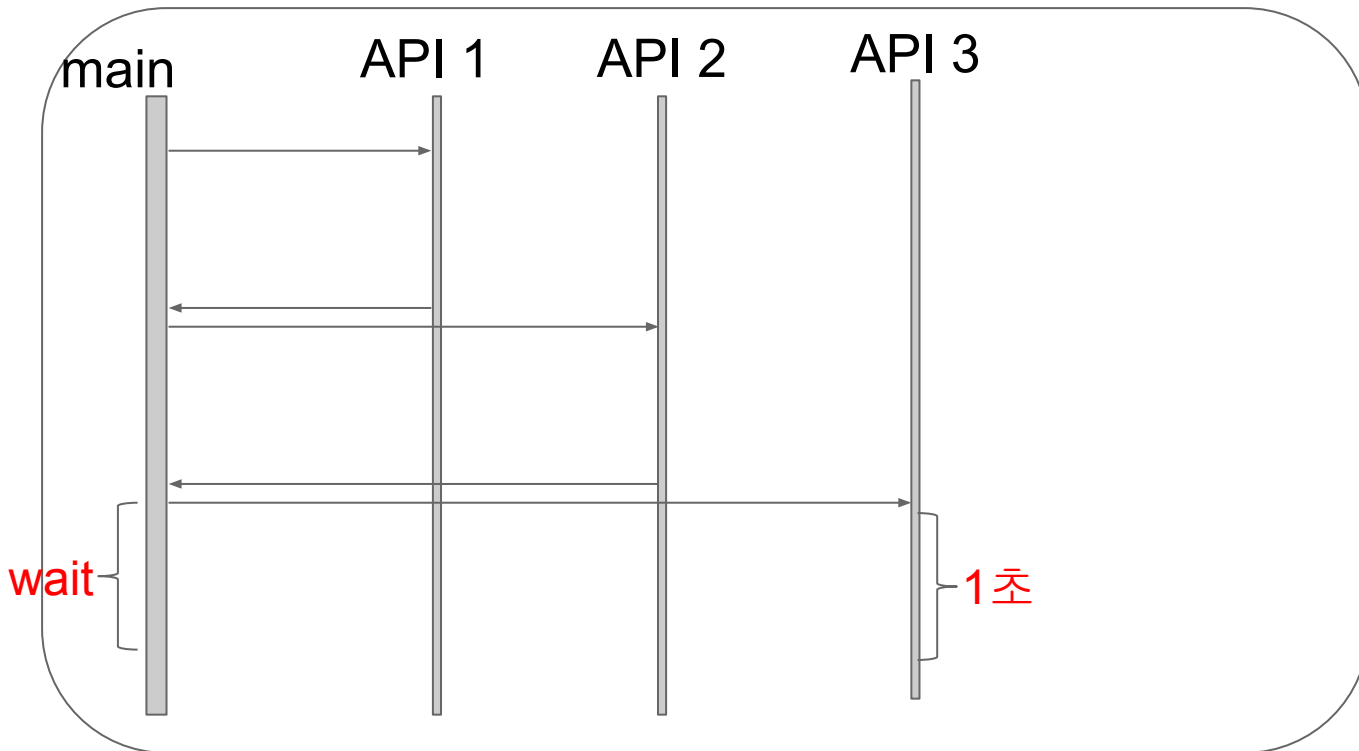
비동기가 필요한 상황



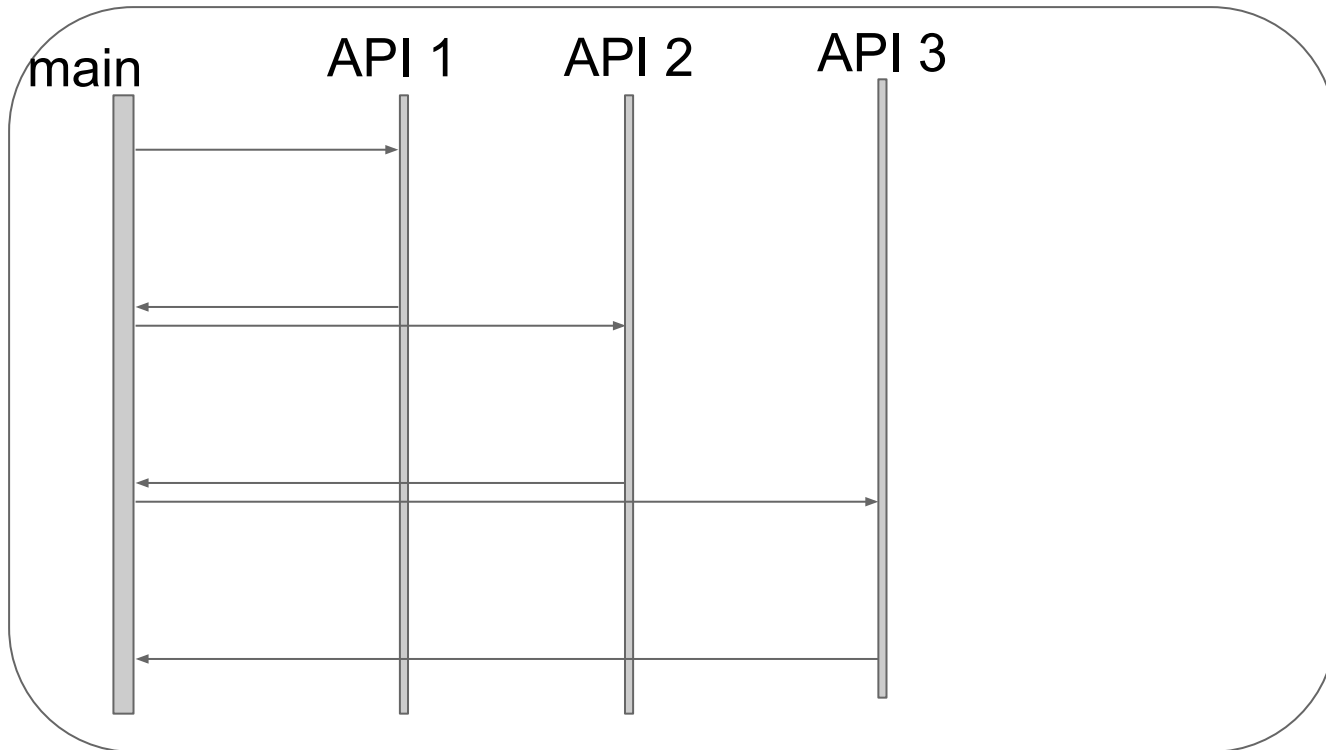
비동기가 필요한 상황



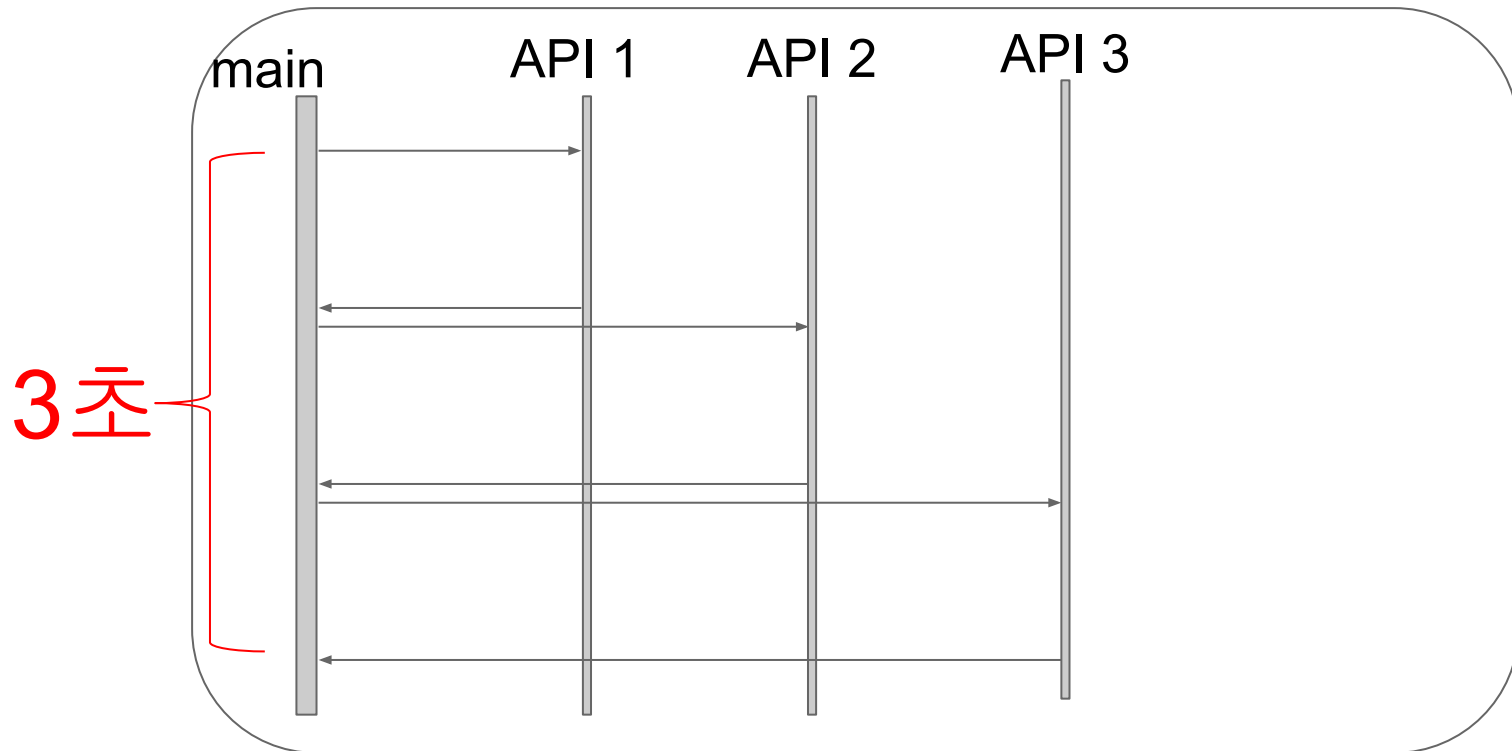
비동기가 필요한 상황



비동기가 필요한 상황



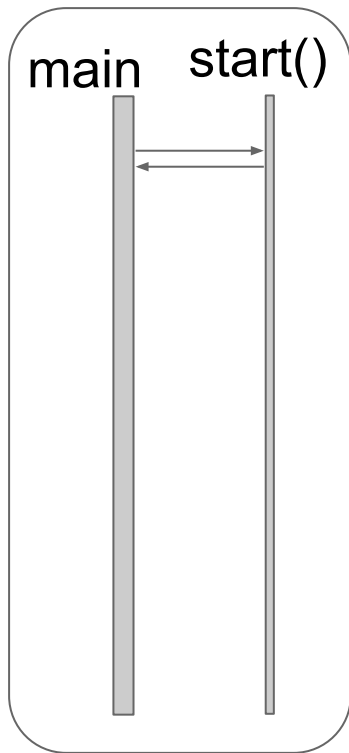
비동기가 필요한 상황



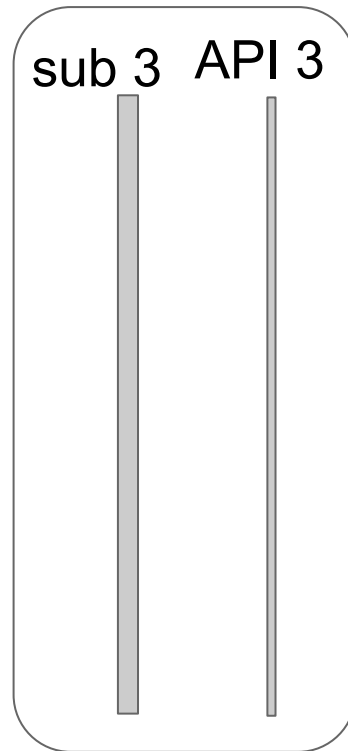
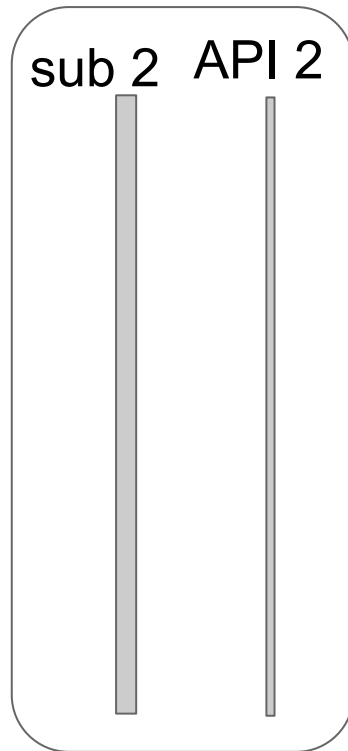
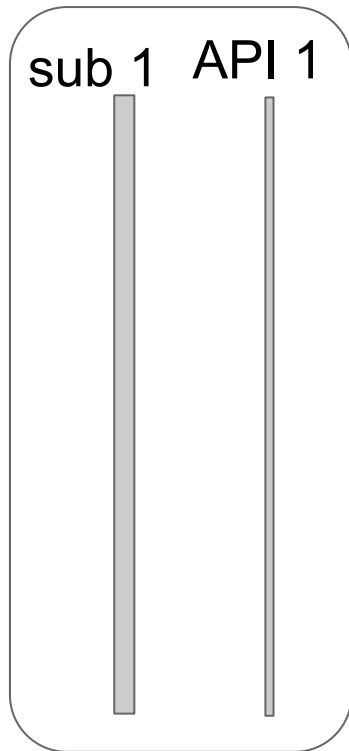
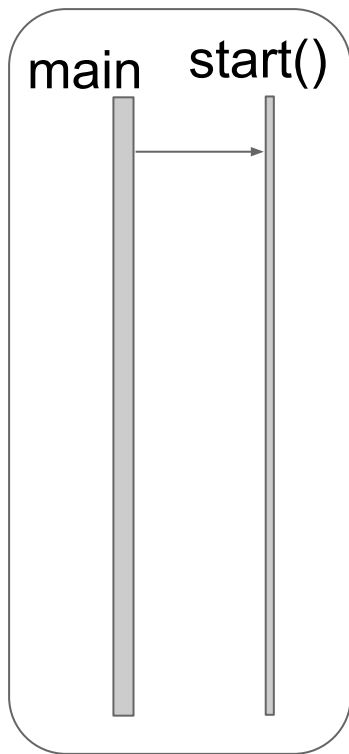


비동기로 바꾸면

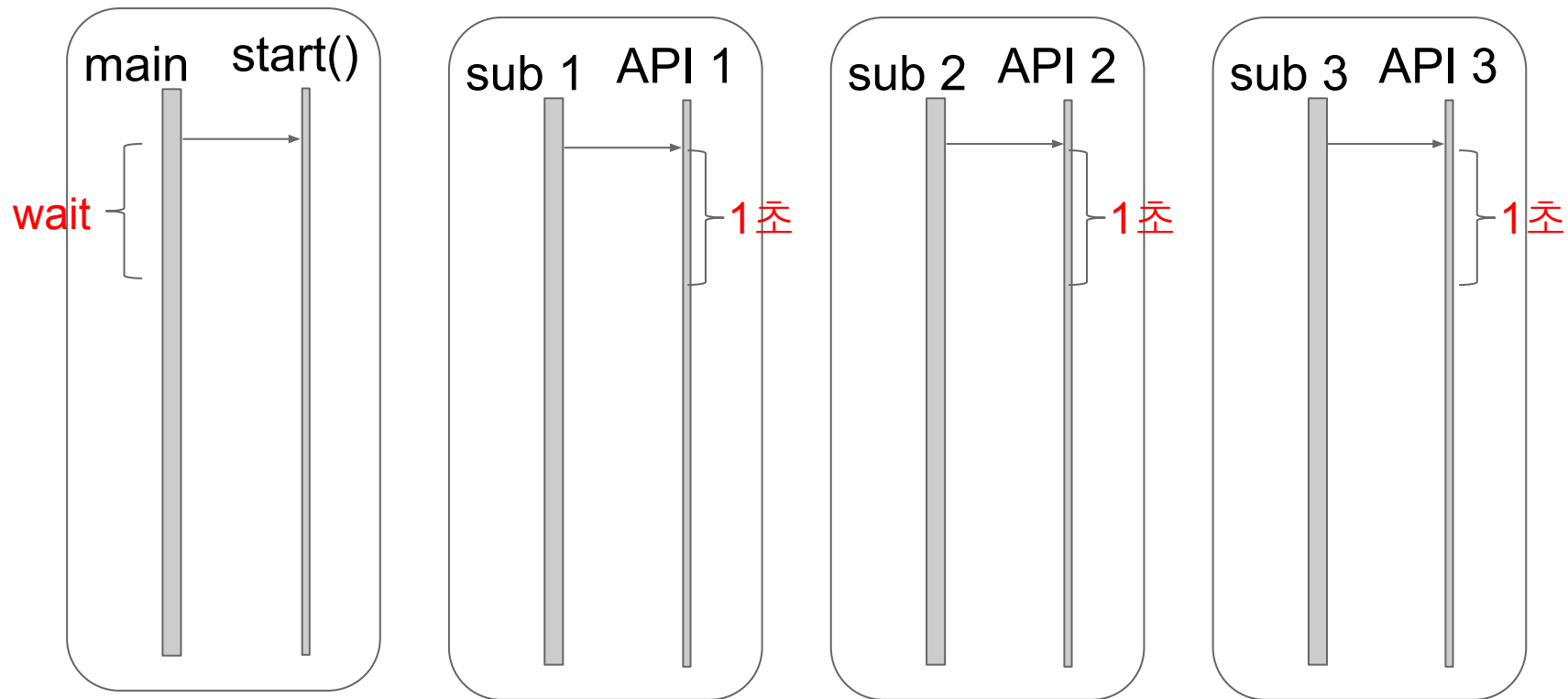
비동기가 필요한 상황



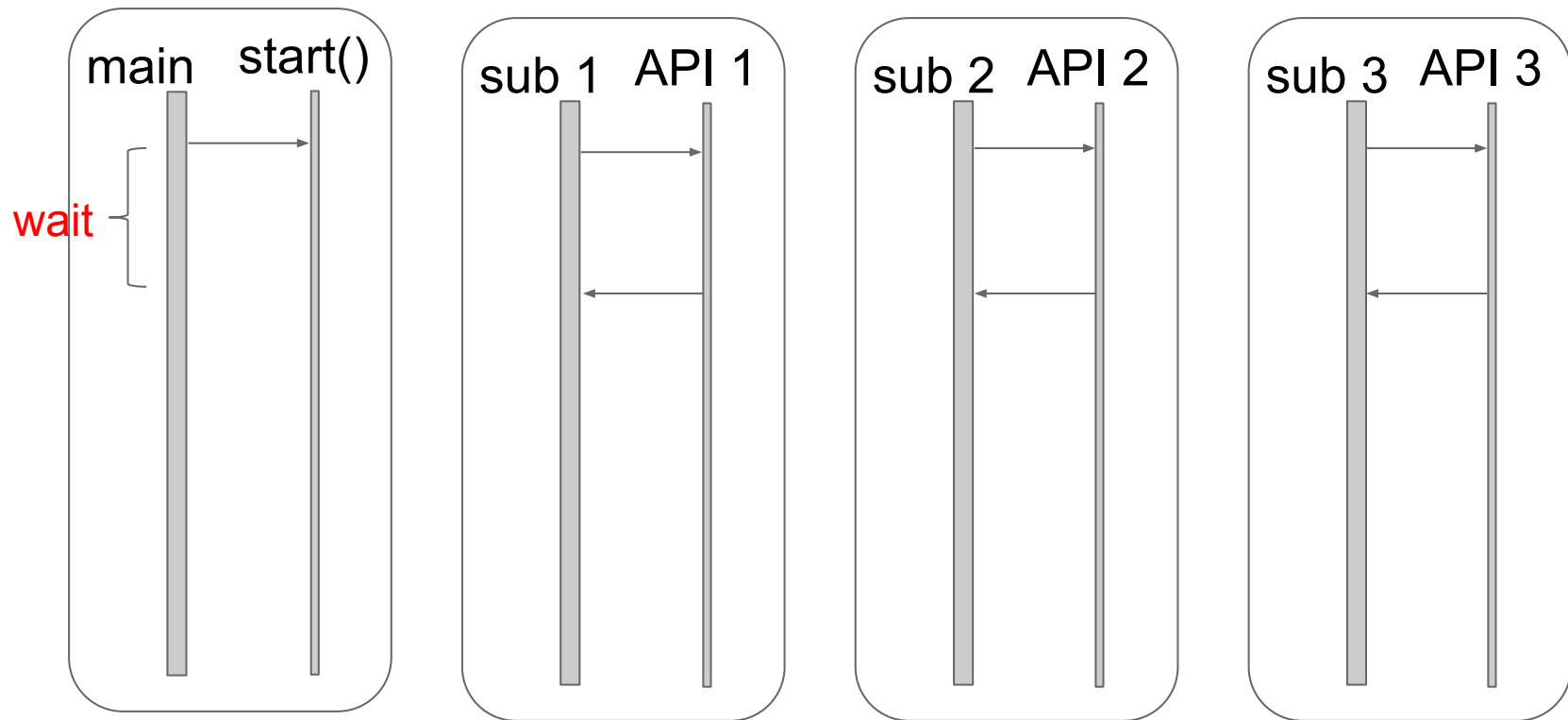
비동기가 필요한 상황



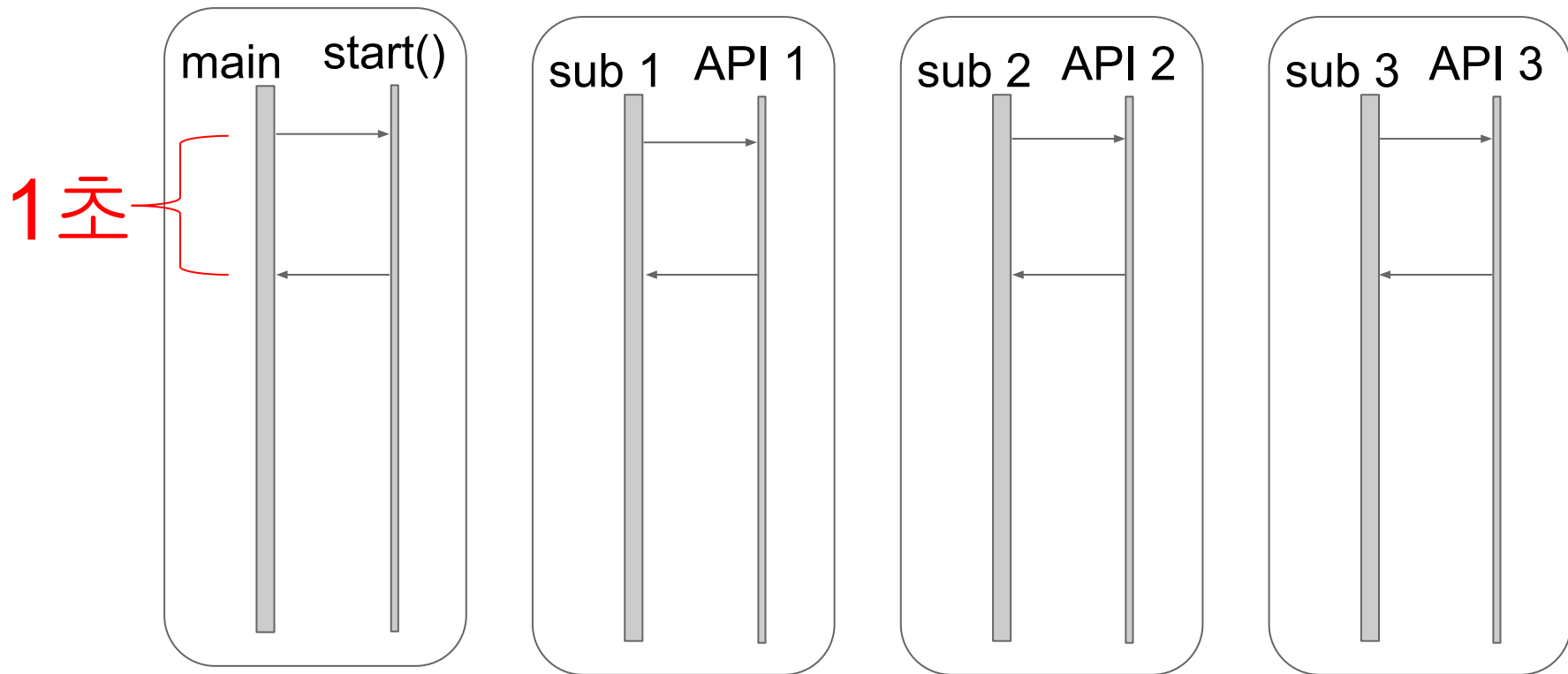
비동기가 필요한 상황



비동기가 필요한 상황



비동기가 필요한 상황



Java 비동기 API

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

Java 비동기 API

```
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[] 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[] 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[] 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[] args) throws InterruptedException {
```

(jvm 실행시 실행)

main 스레드

```
    Thread thread = new Thread();
```

```
    @Override
```

```
    public void run() {
```

```
        System.out.println("main thread");
```

```
        Result result = getResult();
```

```
    }
```

```
});
```

```
thread.start();
```

```
System.out.println("main");
```

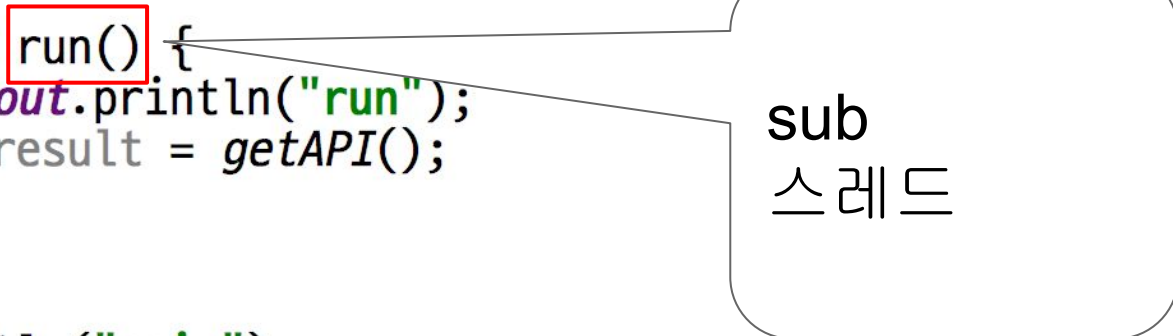
```
}
```


Java 비동기 API

```
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[] 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");  
}
```



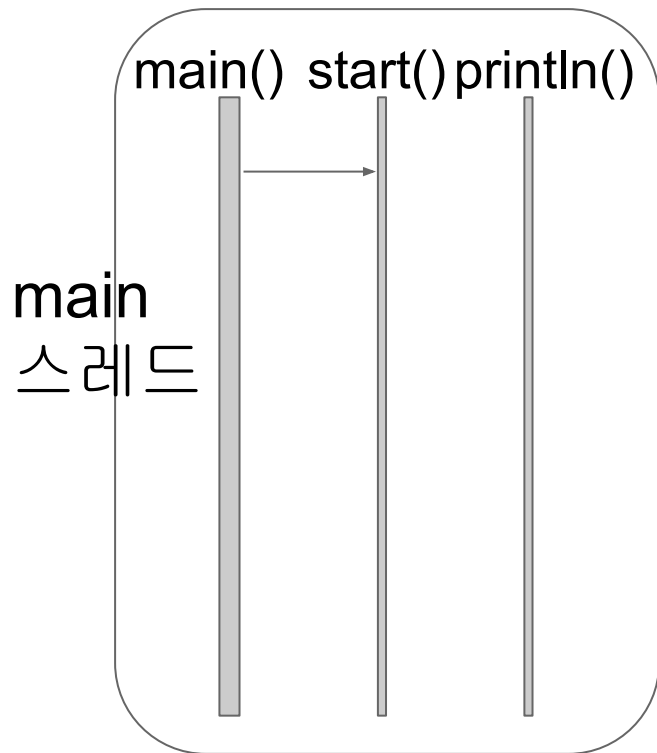
A diagram consisting of a rounded rectangle on the right side of the slide. Inside the rectangle, the text "sub" is on the top line and "스레드" (thread) is on the bottom line. Two lines originate from the left side of the rectangle: one points to the opening curly brace of the `run()` method, and the other points to the `getAPI()` call inside the `run()` method. The `run()` method name is highlighted with a red rectangular box.

sub
스레드

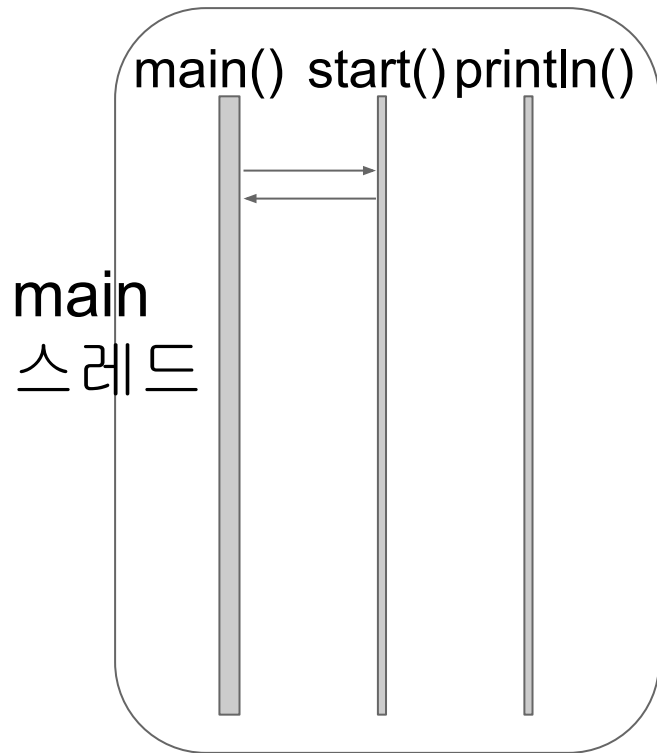
Java 비동기 API

```
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");  
}
```

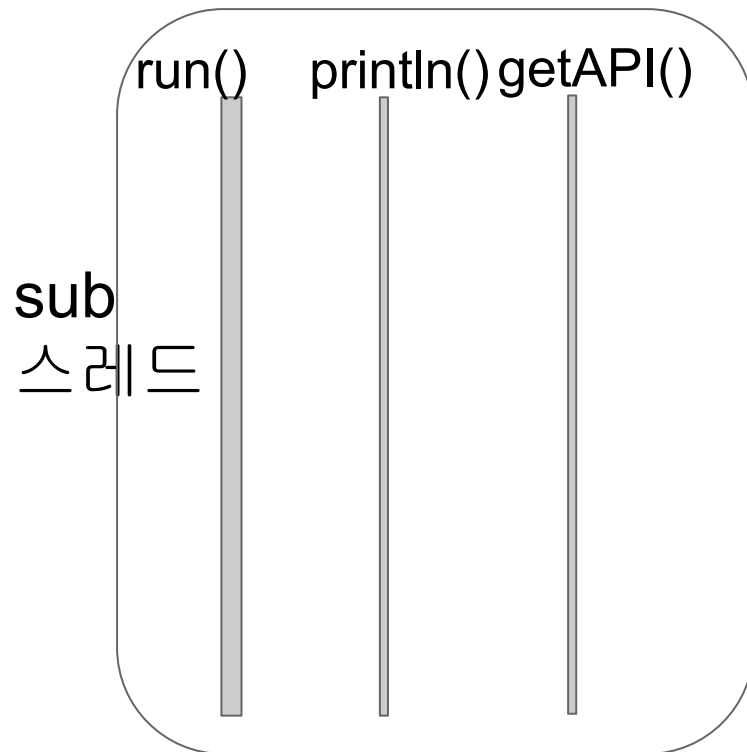
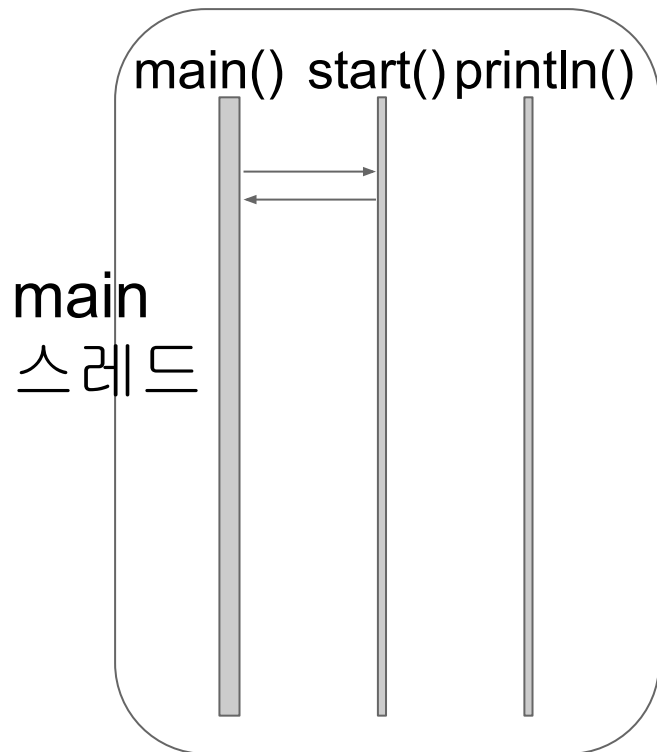
Java 비동기 API



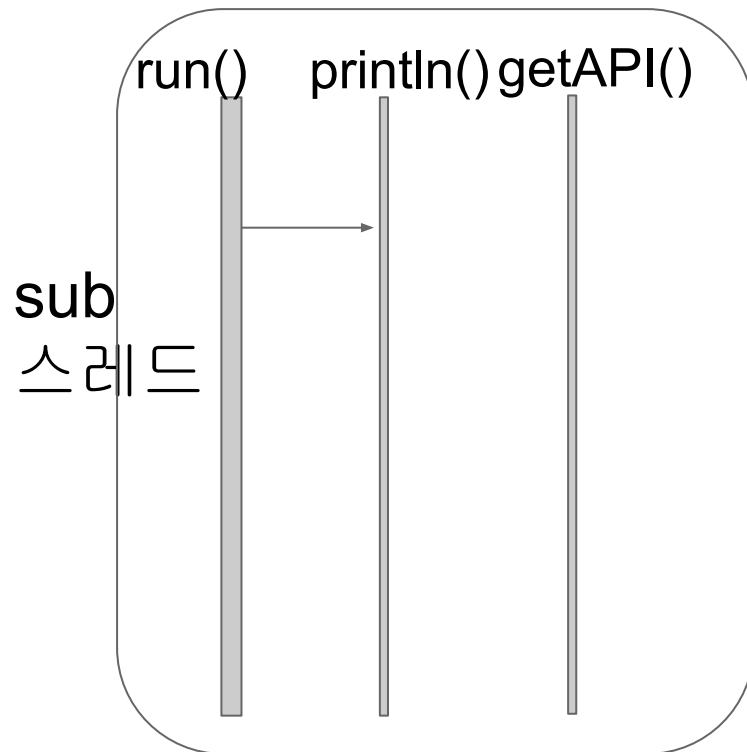
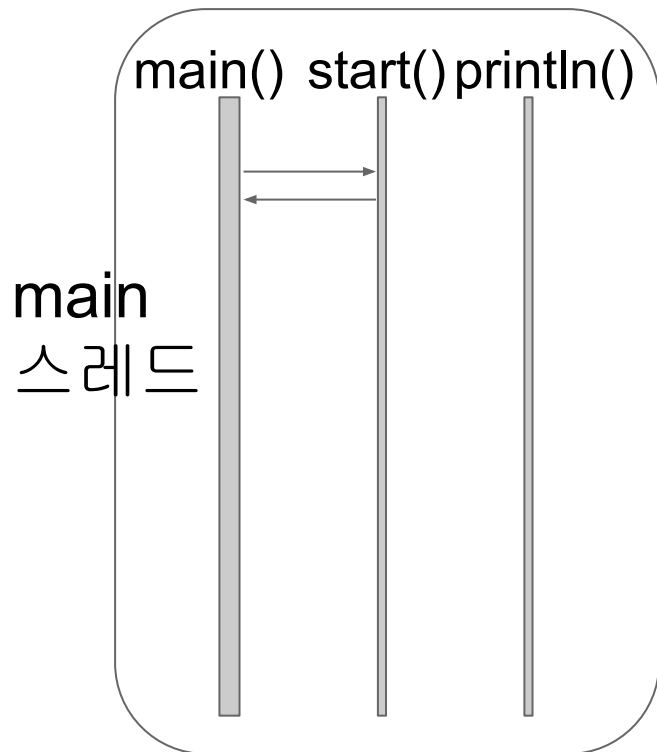
Java 비동기 API



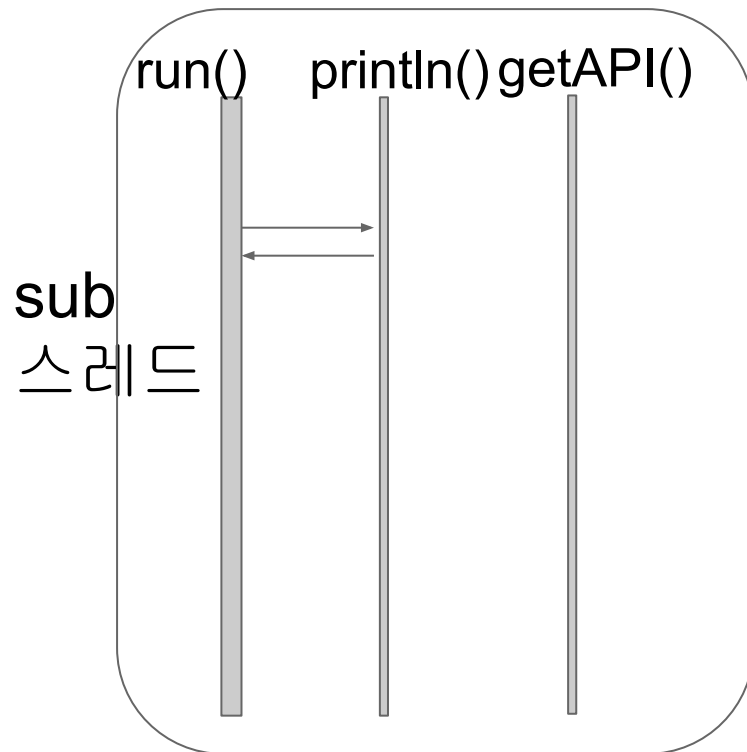
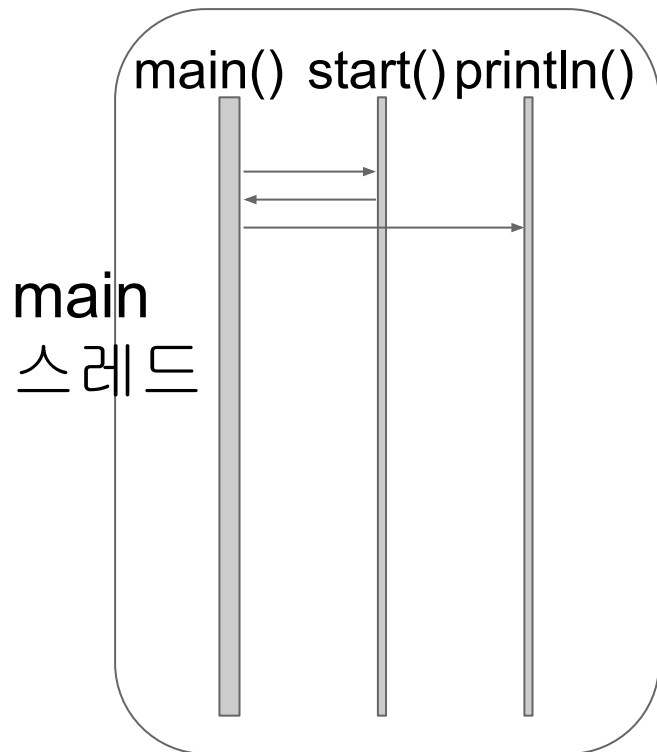
Java 비동기 API



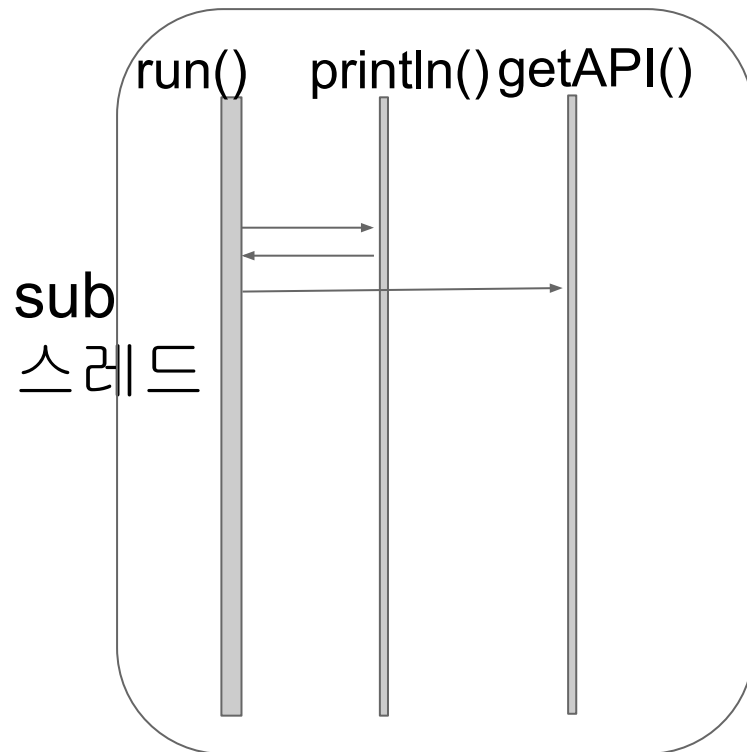
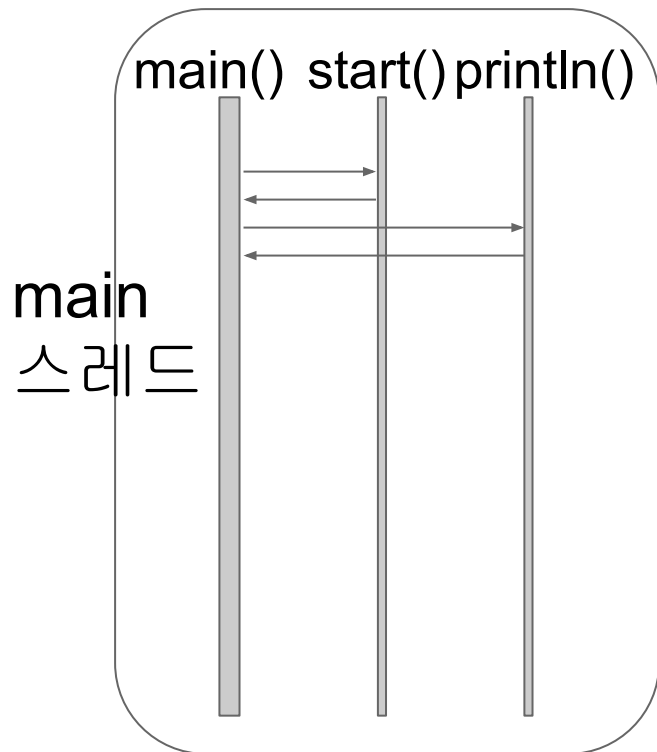
Java 비동기 API



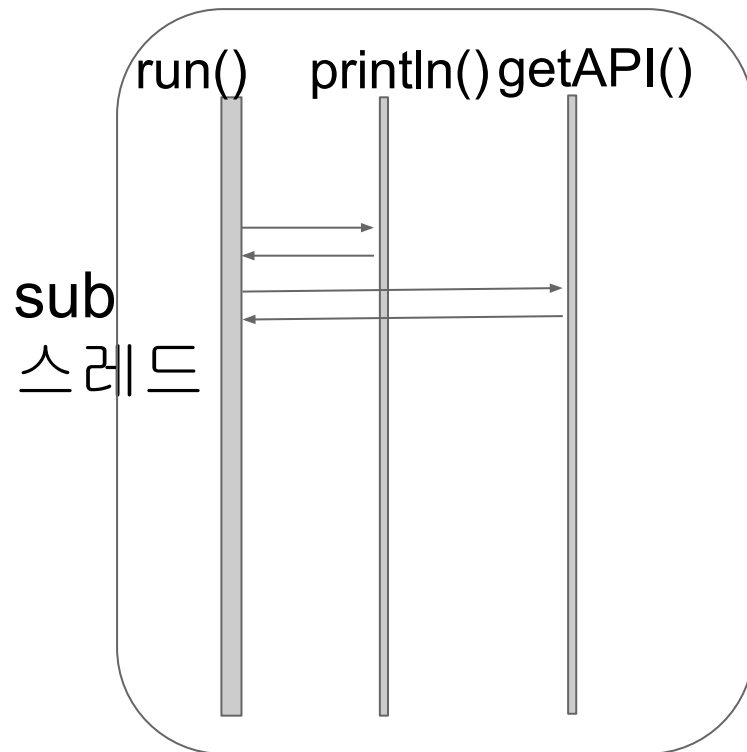
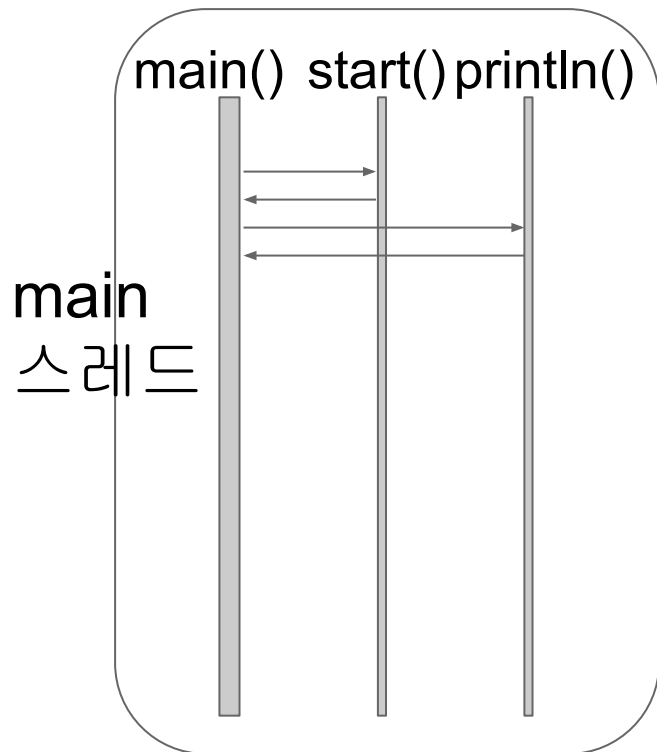
Java 비동기 API



Java 비동기 API



Java 비동기 API





비동기 코드가 어려운점

실행 순서 흐름을 놓치기
쉽다.

Java 비동기 API

```
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[] 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
파일 내에
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

```
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) {
    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

```
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"));
}
```

main

스레드가
종료되기 전
에 끝나지
않음.



- 스레드들간의
작업 순서
동기화 필요



- `class Object`
 - `wait(), notify()`

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);
            synchronized (map) {
                map.notify();
            }
        }
    });
    thread.start();
    if(map.get("API") == null) {
        synchronized (map) {
            map.wait();
        }
    }
    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);
            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, notify**로했던 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 InterruptedException {
    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

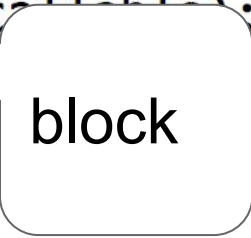
```
public static void main(String[] args) throws InterruptedException {
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();
    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 InterruptedException {
    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

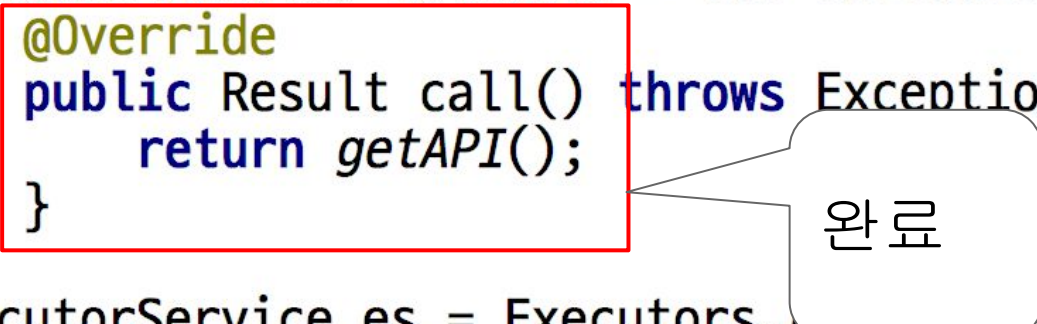
```
public static void main(String[] args) throws InterruptedException {
    Callable<Result> callable = new Callable<Result>() {
        @Override
        public Result call() throws Exception {
            return getAPI();
        }
    };
    ExecutorService es = Executors.newSingleThreadExecutor();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



block

비동기 개발에 사용한 API - Future

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



완료

비동기 개발에 사용한 API - Future

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

block 해제

비동기 개발에 사용한 API - Future

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

 채널



ㅋㅋㅋ 인기

클릭하자마자 웃음 빵~

42,140명에게 배달 중 | 파트너 >

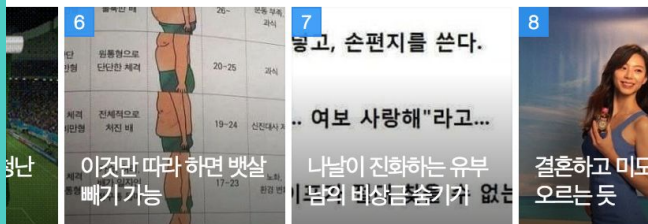
+ 채널 소식 배달 받기

공유

읽을거리

같이 볼만한 채널

인기 Best



전체 5,536



사회에서 여러모로 인정받는 인재

NEW

읽을거리

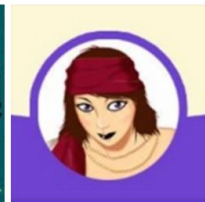
같이 볼만한 채널



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

느낀점

- thread를 직접 사용하는건 쉽지 않다 (지식 + 경험)

느낀점

- **Future**는 단일값만 처리하기 때문에 복합적으로 구현하기 어렵다.
 - A작업 -> B작업 -> C작업.....

느낀 점

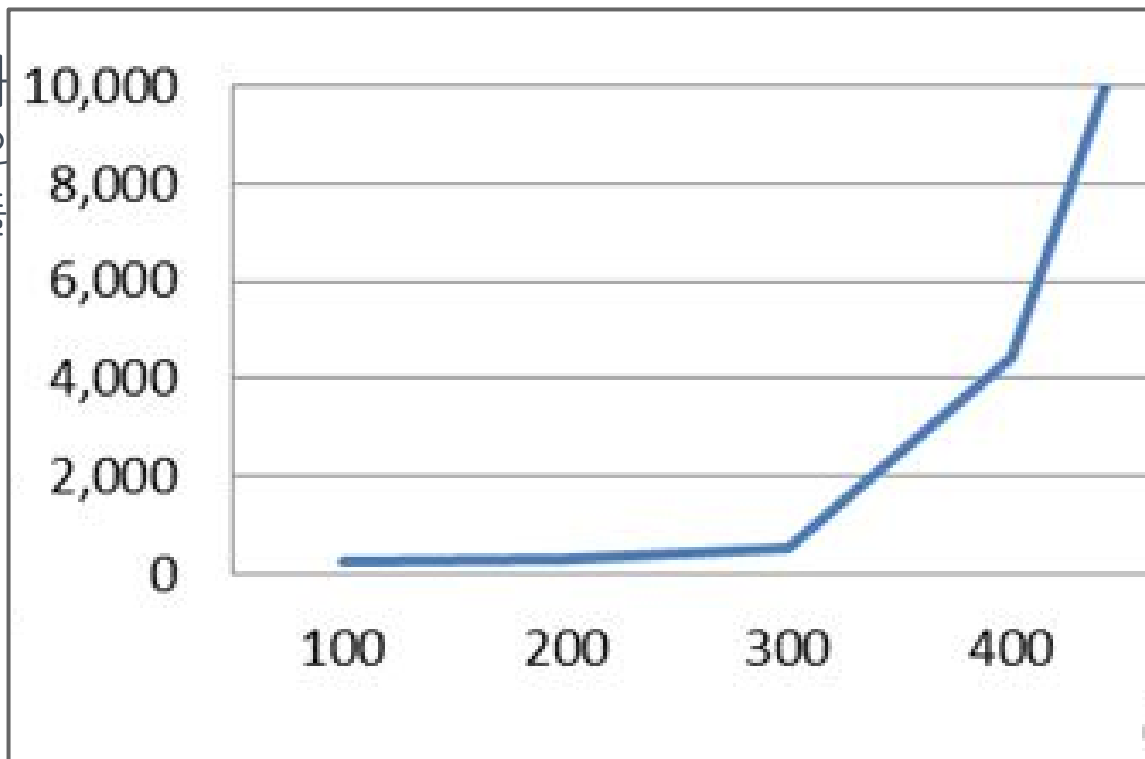
- threadpool에 대해서도 이해가 필요하다.

느낀 점

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

느낀 점

- threadpool에 다
- threadpool내의 대기중인 요청들 처리되기때문에



느낀 점

- threadpool에 다
- threadpool내의 대기중인 요청들 처리되기때문에



느낀 점

- 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 request => 1 + 7 (threads)

느낀 점

- 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 request => 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 **Future**의 단일 작업 처리문제 극복
 - **CompletableFuture** (JDK 8)



A -> B -> C

의존성있는 태스크

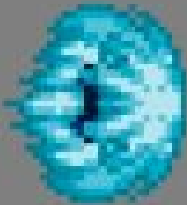
기존 **Future**에서

해결하려면!


```
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()
🕒

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()
블록 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

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

람다

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();
```

CompletableFuture

Java8

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

CompletableFuture

Java8

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

메서드
레퍼런스

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

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

CompletableFuture는 좋다!!

조금 불만이라면 반복되는 작업을
Collection에 넣고 비동기적으로 실행하고
정의된 콜백만 실행 싶을때가 있는데...

4.

NIO

nio
nonblocking

도입

- 아직 풀지 못한 상황..
 - ThreadPool Full

도입

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

Nonblocking

- 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

- NIO

- jdk 1.4 nio
- jdk 1.7 nio2

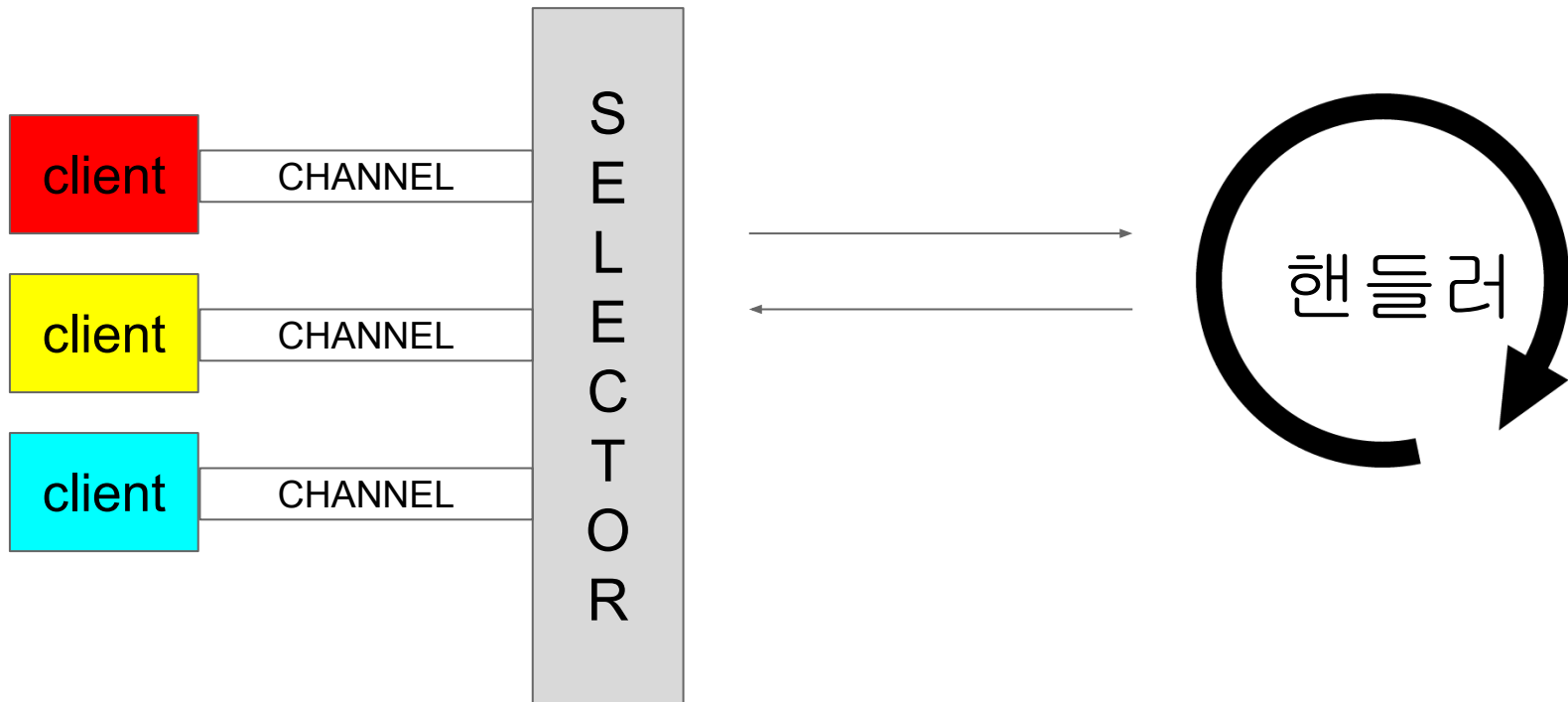
- New IO

- Native IO
- Nonblocking IO

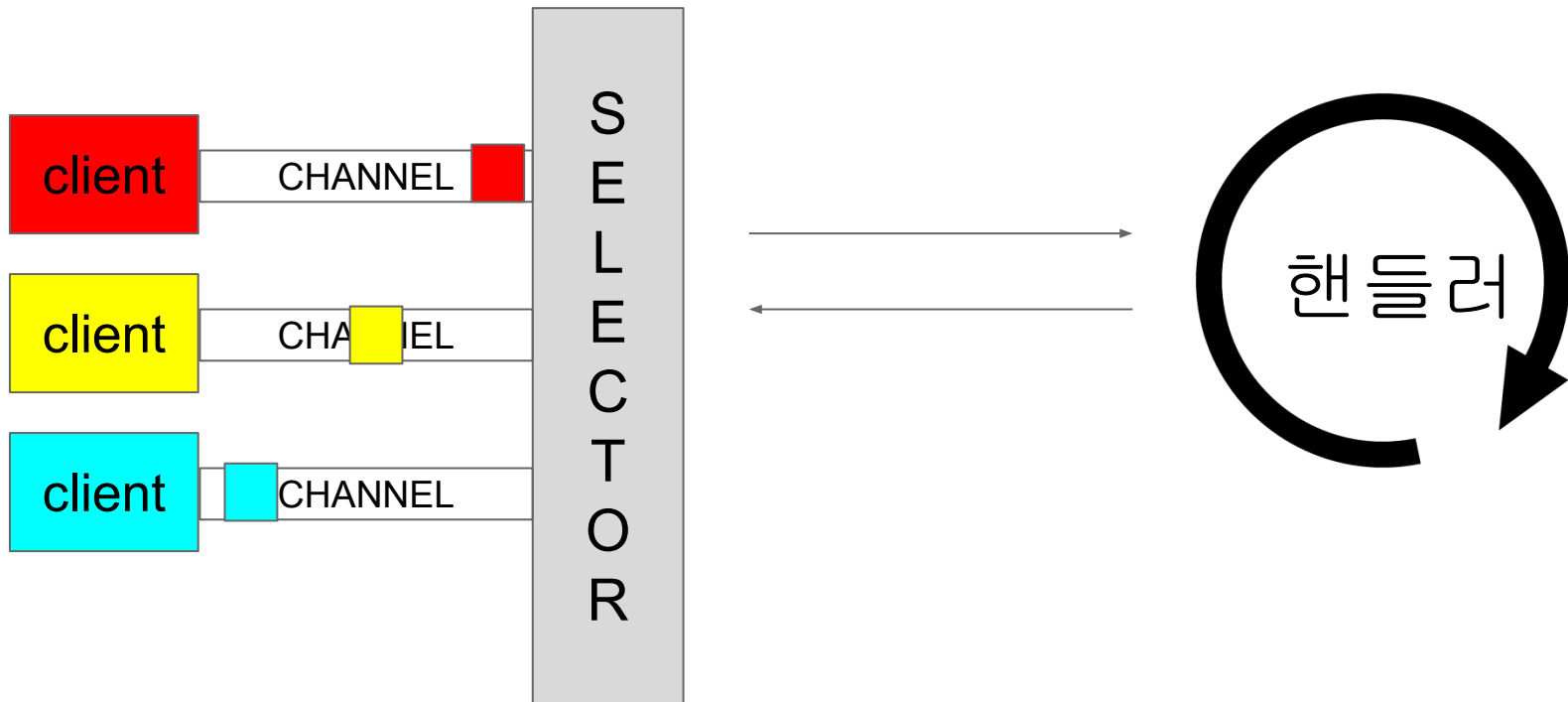
- packages

- 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

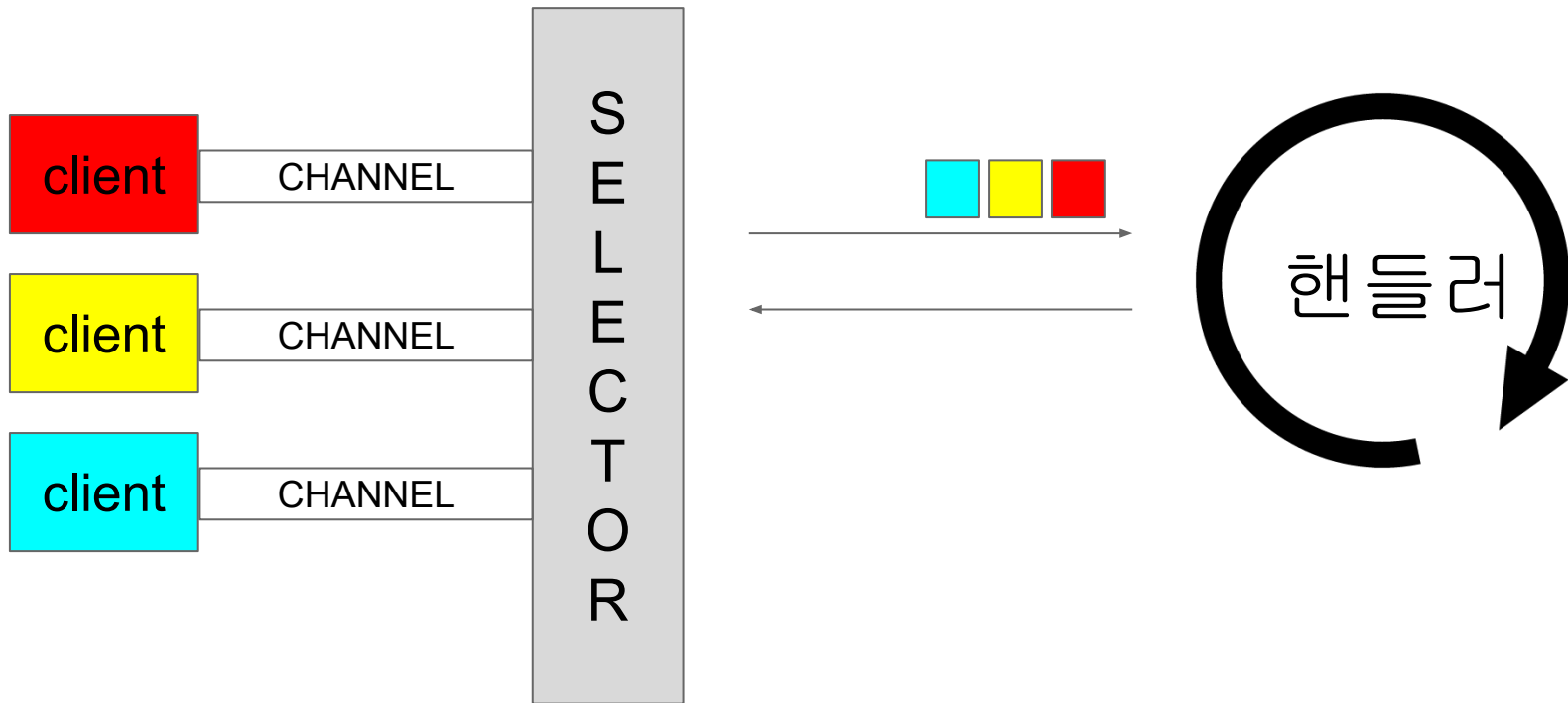
JAVA NIO Selector, Channel



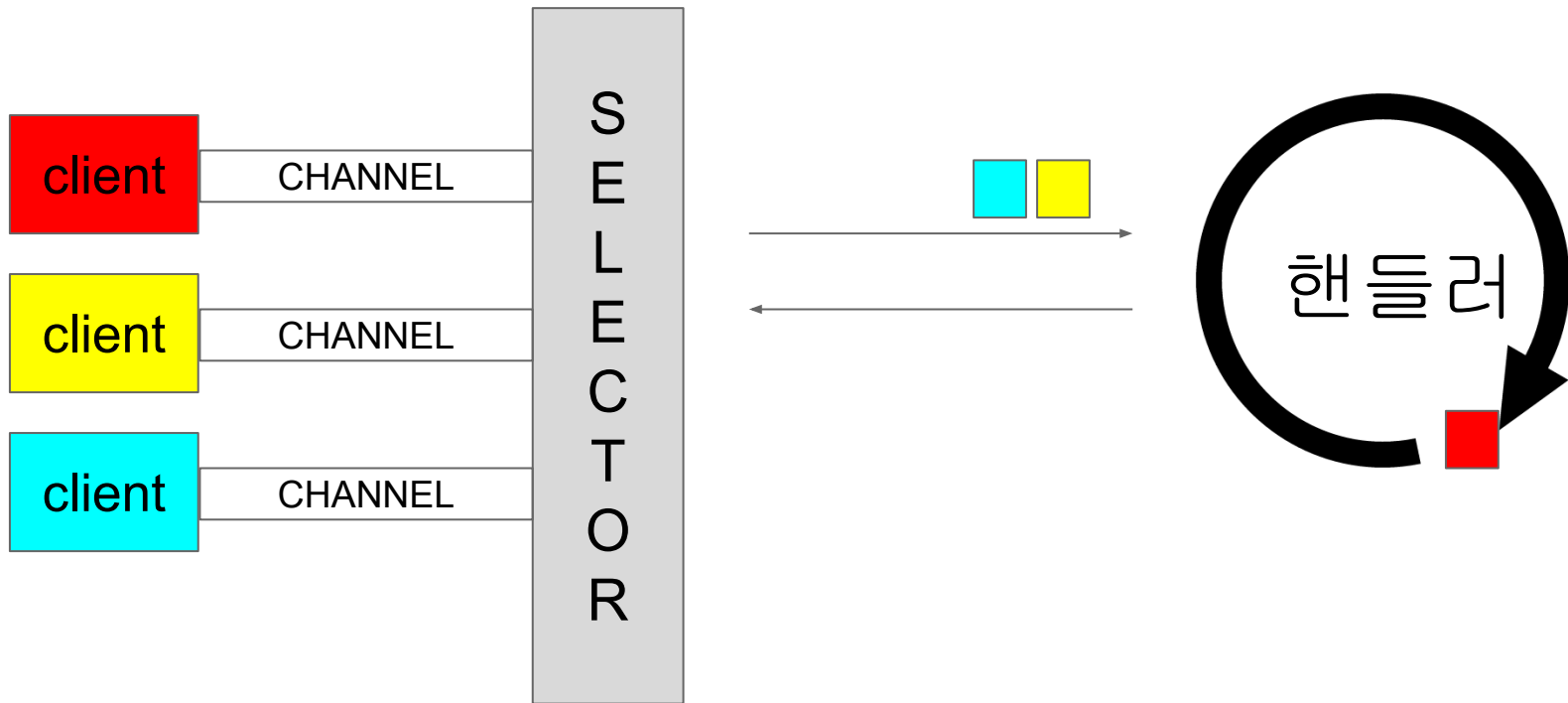
JAVA NIO Selector, Channel



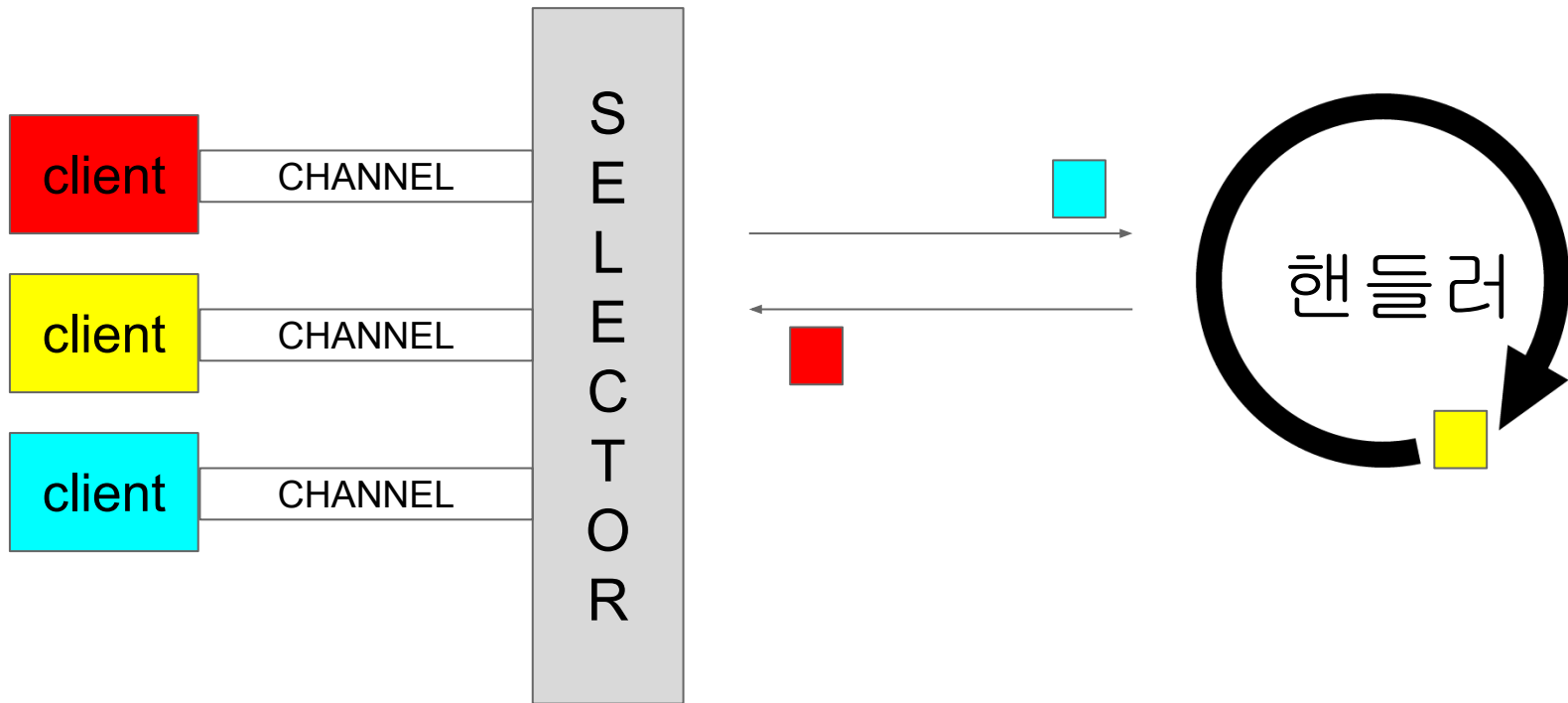
JAVA NIO Selector, Channel



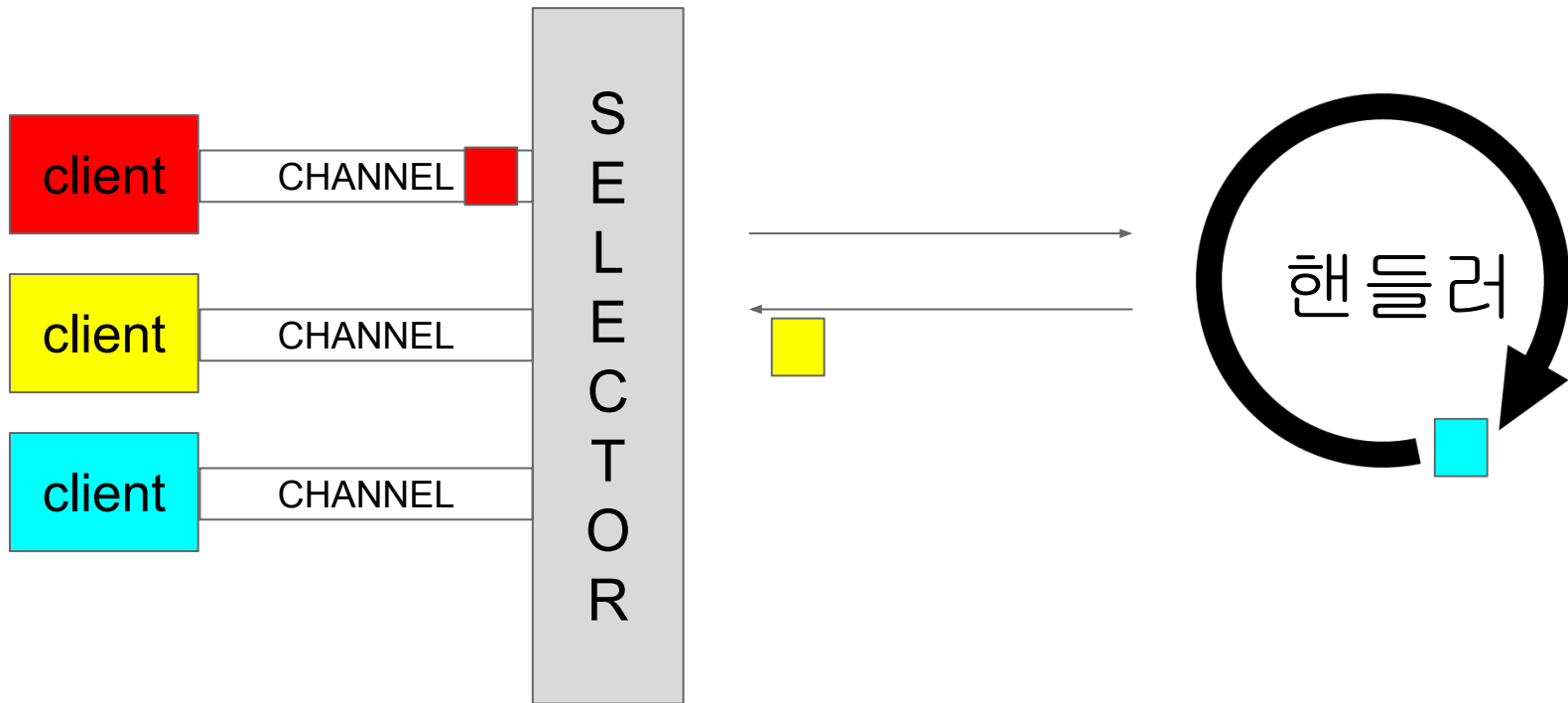
JAVA NIO Selector, Channel



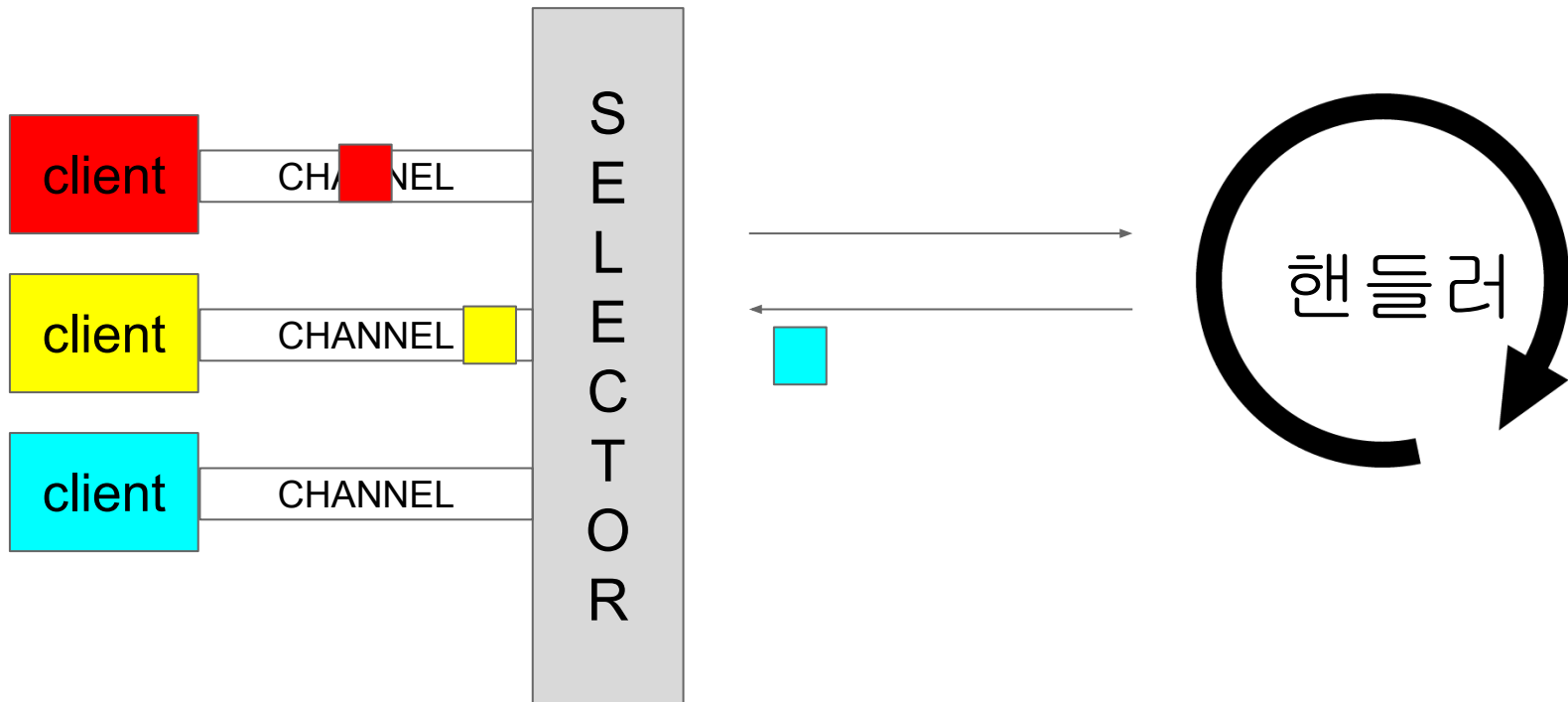
JAVA NIO Selector, Channel



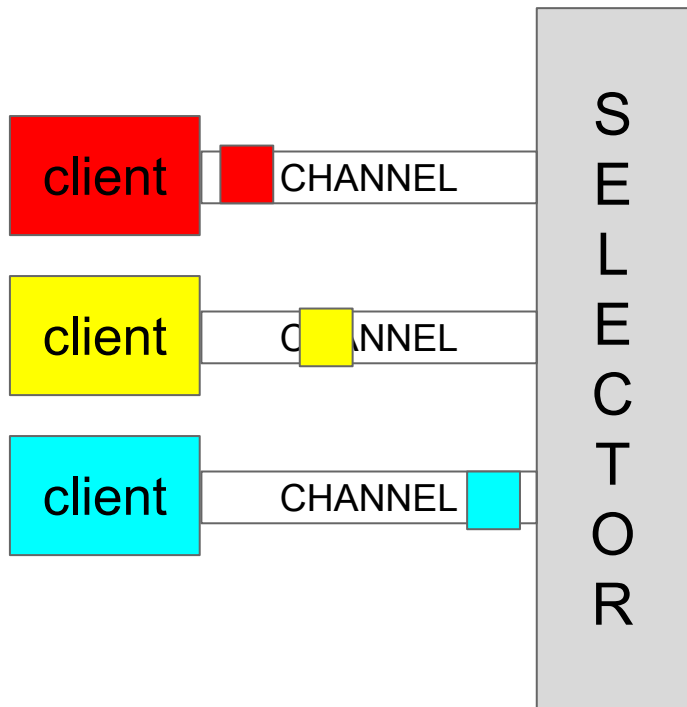
JAVA NIO Selector, Channel



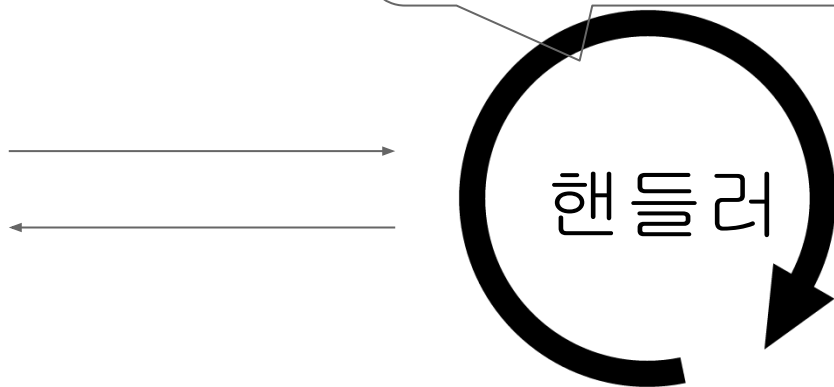
JAVA NIO Selector, Channel



JAVA NIO Selector, Channel



I/O 작업을
단일 스레드로
처리 가능



Nonblocking

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



시연

4.

Reactive

Event Driven Programming
마무리

Event driven programming

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

Event driven programming

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

Event driven programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.

- 이벤트 발생 -> 처리

- CompletableFuture
.complete(/* 값 */)

complete 이벤트

Event driven programming

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

Event driven programming

- CompletableFuture나 Nonblocking IO 방식의 공통점.

- 이벤트 발생 -> 처리

- CompletableFuture
 .complete(/* 값 */)

- CompletableFuture
 .thenApply(/* 콜백 */)

이벤트에 대한
처리

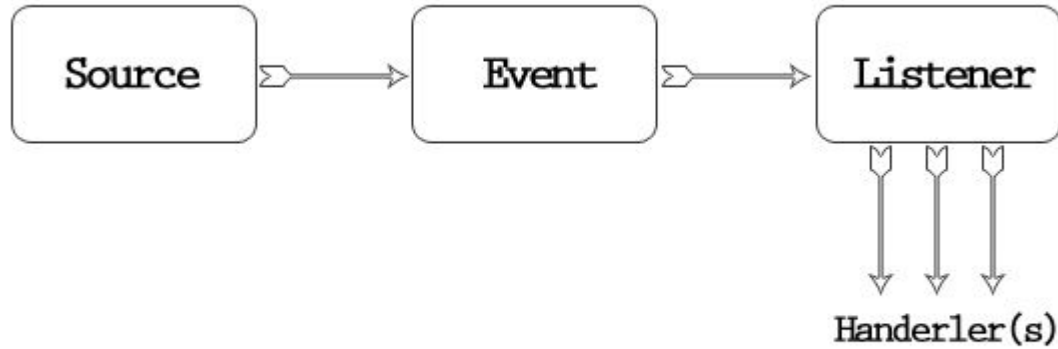
Event driven programming

- Event
 - 프로그램에 의해 감지되고 처리될 수 있는 동작이나 사건

Event driven programming

- Event driven programming
 - 절차지향 프로그래밍과 대조되는 프로그래밍 패러다임
 - 이벤트가 발생한다는 전제를 깔고 이벤트를 처리하는 코드를 작성

Event driven programming



Event-Driving Programming Model

Event driven programming



Event driven programming



Event driven programming



Event driven programming

SOURCE



EVENT



LISTENER
HANDLER



Event driven programming

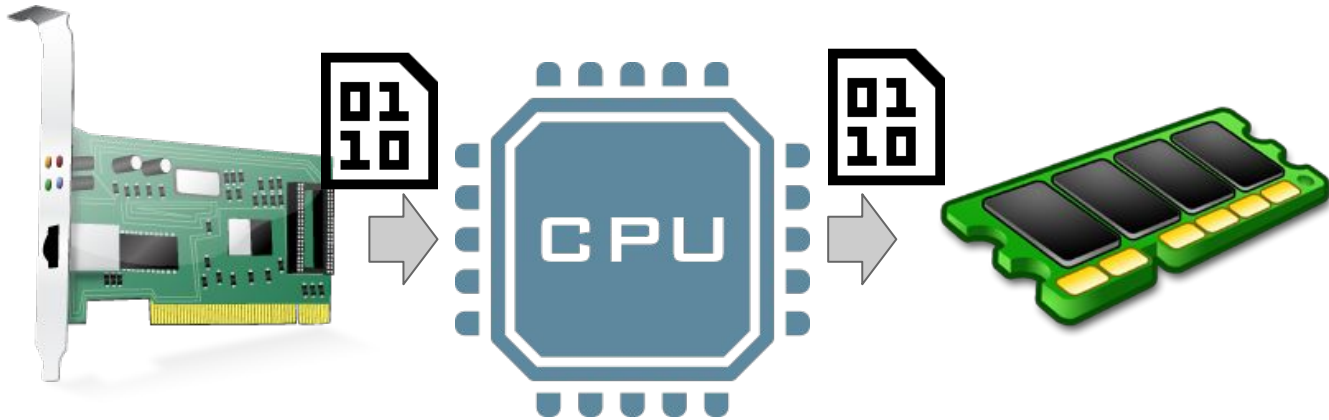


우리는 이벤트 세상에 살고 있다.

Event driven programming

H/W

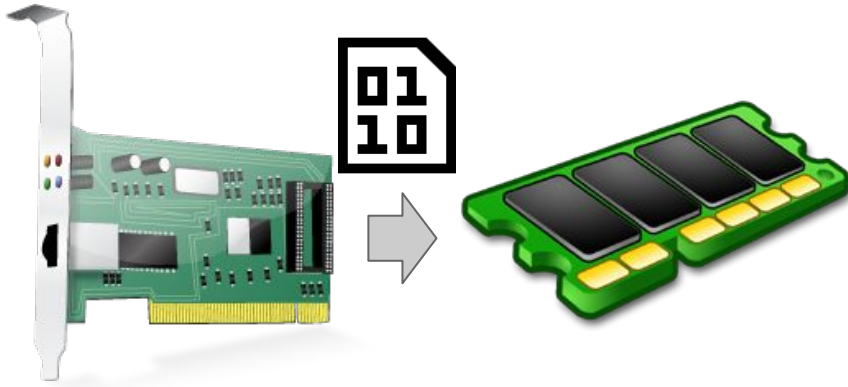
-> PIO(programmed I/O)



Event driven programming

H/W

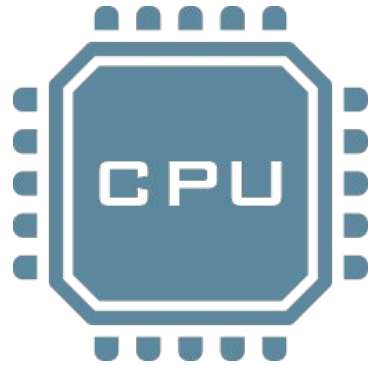
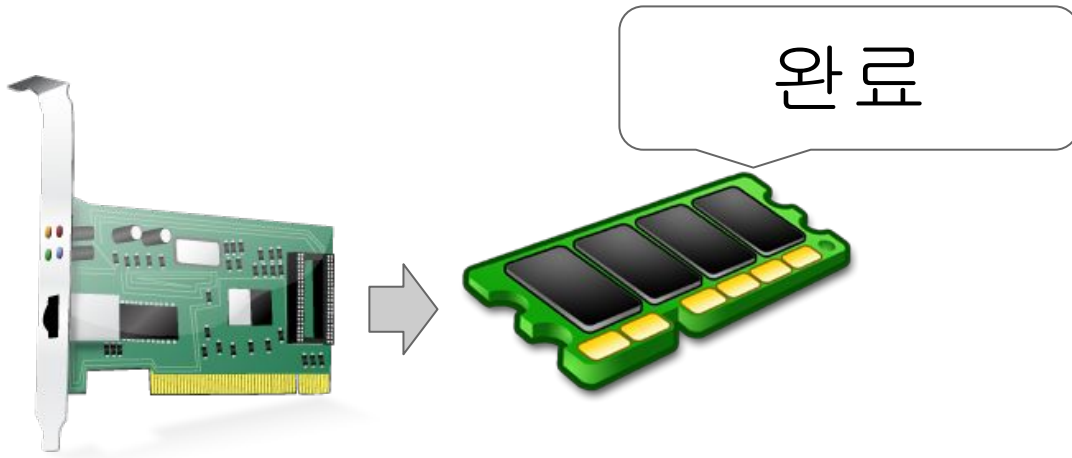
-> DMA(direct memory access)



Event driven programming

H/W

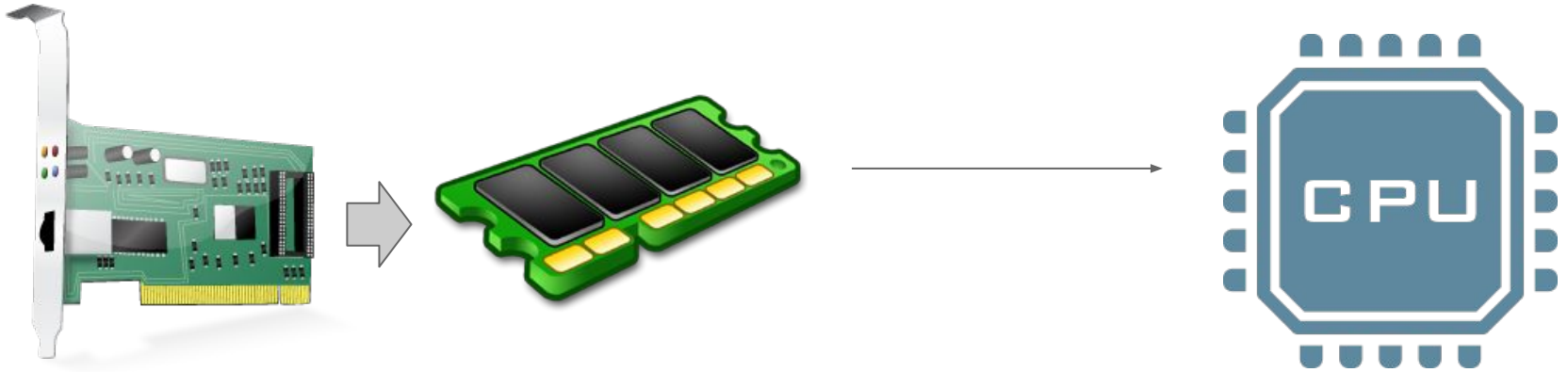
-> DMA(direct memory access)



Event driven programming

H/W

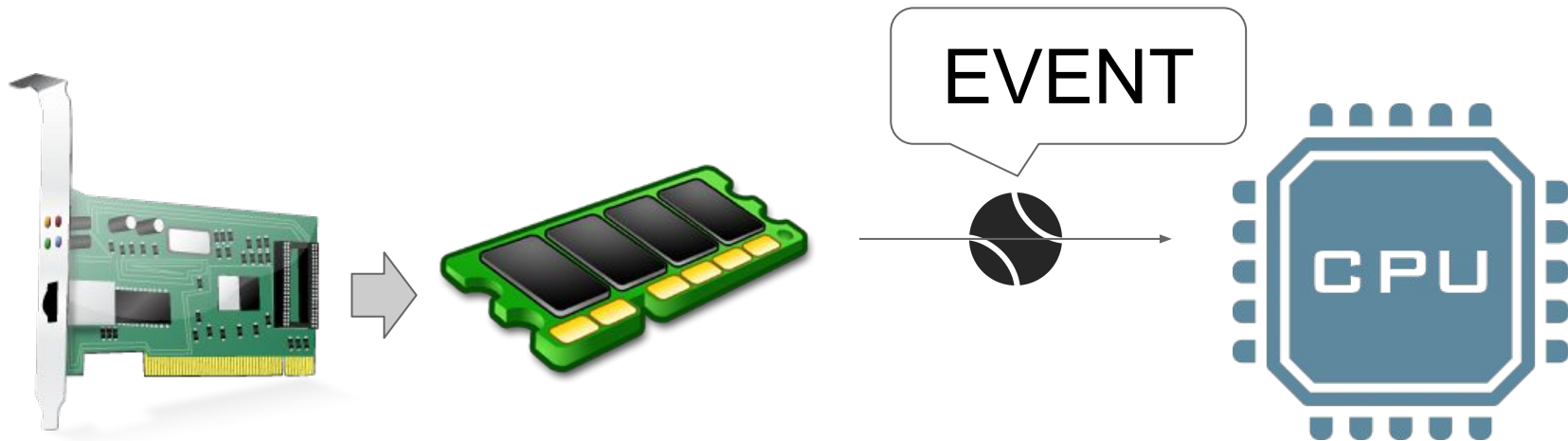
-> DMA(direct memory access)



Event driven programming

H/W는 이벤트 기반이다

-> DMA(direct memory access)

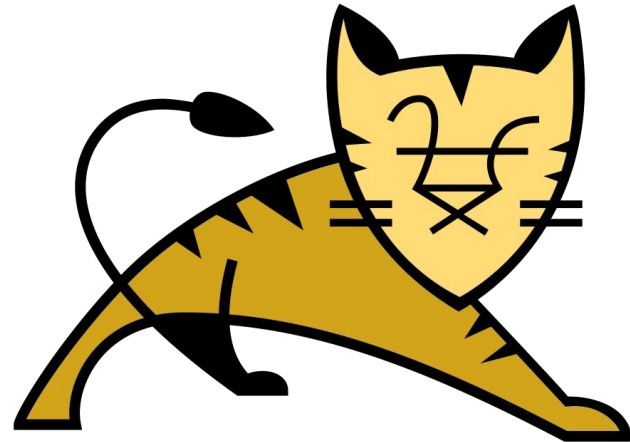


Event driven programming

Servlet



Request

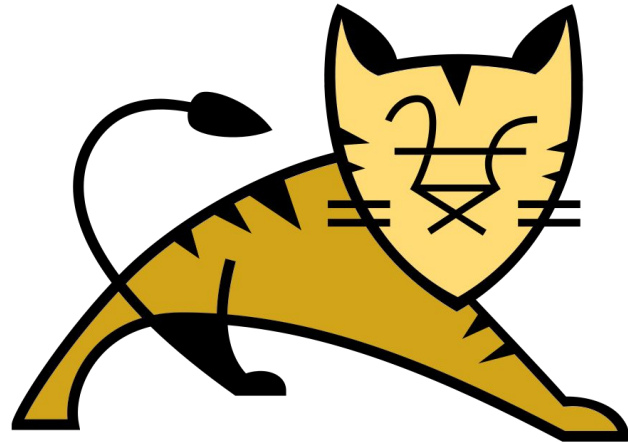


Event driven programming

Servlet



Response



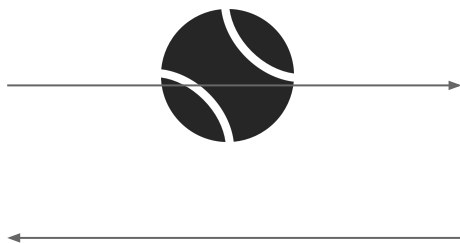
Event driven programming

Servlet request도 이벤트다

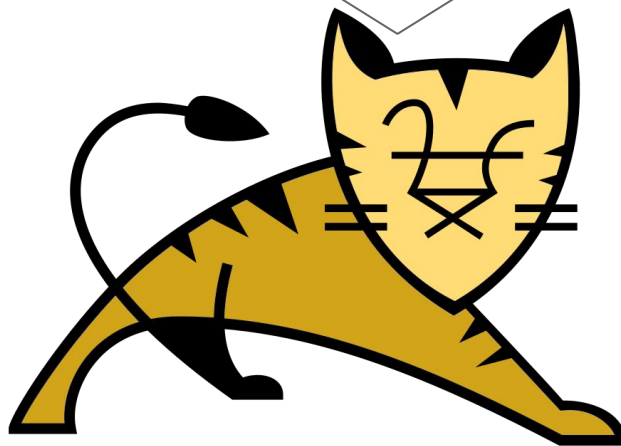
SOURCE



EVENT



LISTENER
HANDLER



정리

- 의문
 - 이벤트 세상에서 절차형으로 프로그래밍하려고 하니 오히려 어려운게 아닐까?

정리

- 의문

- 이벤트 기반 프로그래밍이 오히려 편해지지 않을까하는 생각이 든다.

정리

- 의문

- 프로그래밍 패러다임중 핫한 리액티브 프로그래밍도 이벤트 기반 프로그래밍의 한 축

정리

- 의문

- 이벤트 드리븐 프로그래밍을 위해 잘 추상화된 API가 있다면?

정리

- 의문

- 이벤트 드리븐 프로그래밍을 위해 잘 추상화된 API가 있다면?
 - observable, Rxjava, Reactor, sodium, JDK9 Flow...

정리

□ 의문

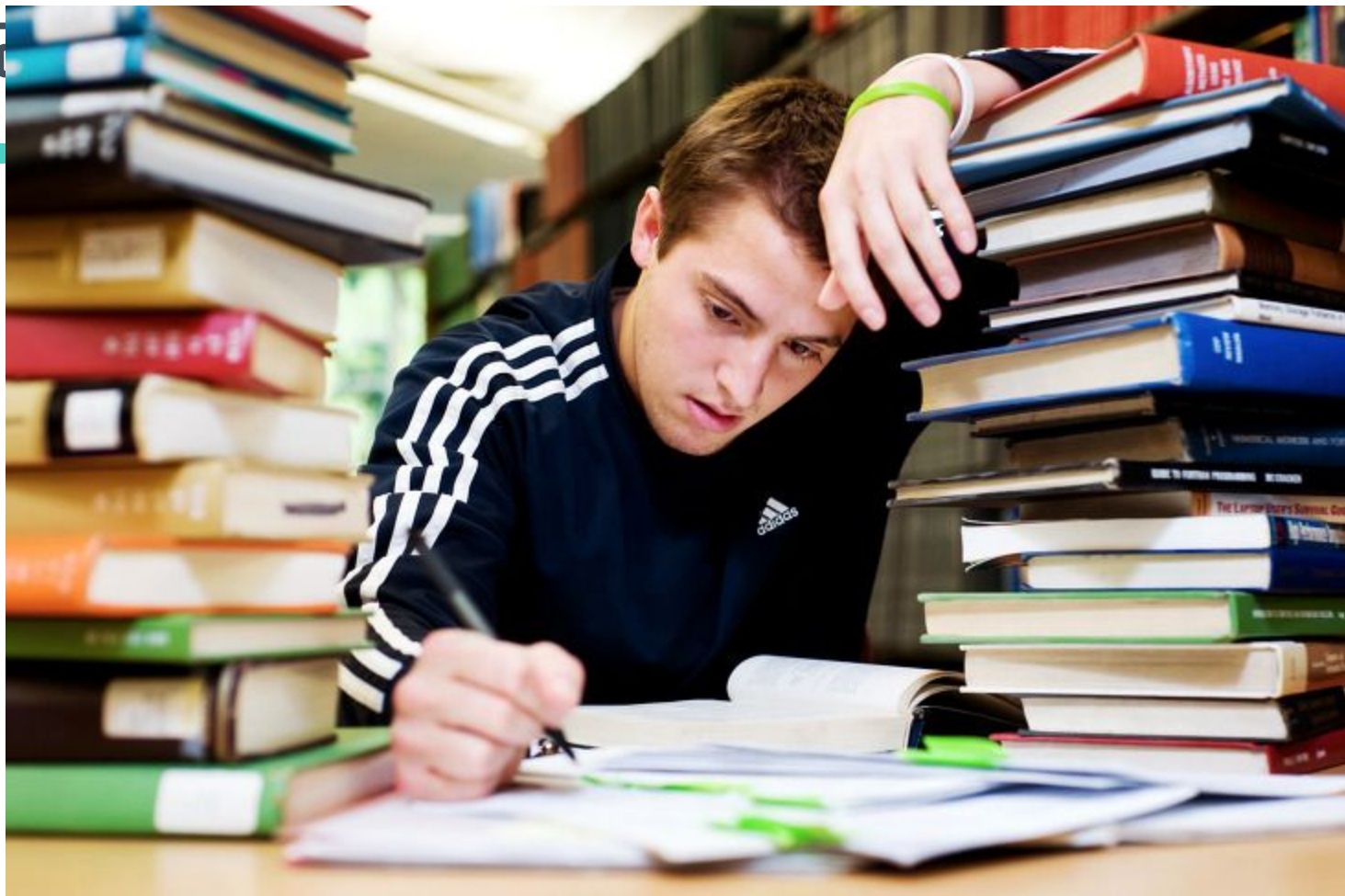
- 이벤트 드리븐 프로그래밍을 위해 잘 추상화된 API가 있다면?
 - observable, Rxjava, Reactor, sodium, JDK9 Flow...
- 이벤트 드리븐 기반 웹프레임워크가 있다면?

정리

□ 의문

- 이벤트 드리븐 프로그래밍을 위해 잘 추상화된 API가 있다면?
 - observable, Rxjava, Reactor, sodium, JDK9 Flow...
- 이벤트 드리븐 기반 웹프레임워크가 있다면?
 - Spring 5 MVC, AKKA, Vert.x, Playframework...

정리



정리

- ▣ 범위를 좁혀보자

정리

- ▣ 범위를 좁혀보자
 - ▣ 현장에서 가장 많이 쓰는 자바, 스프링

정리

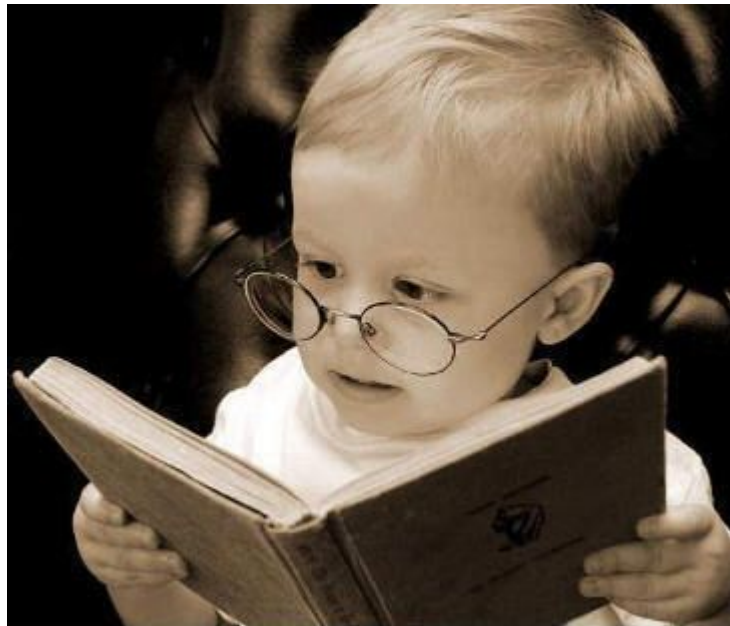
- ▣ 범위를 좁혀보자
 - ▣ 현장에서 가장 많이 쓰는 자바, 스프링
 - ▣ 리액티브 프로그래밍

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

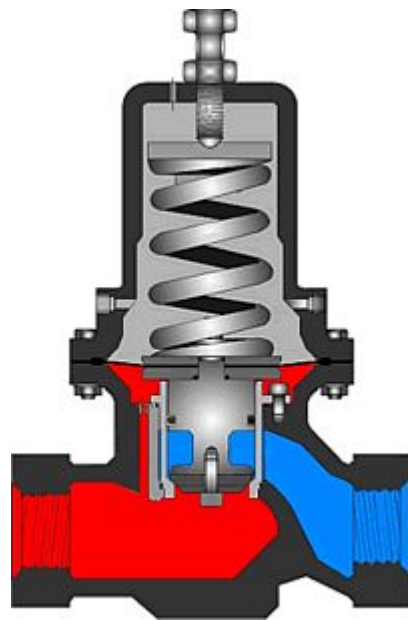


정리

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

정리

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

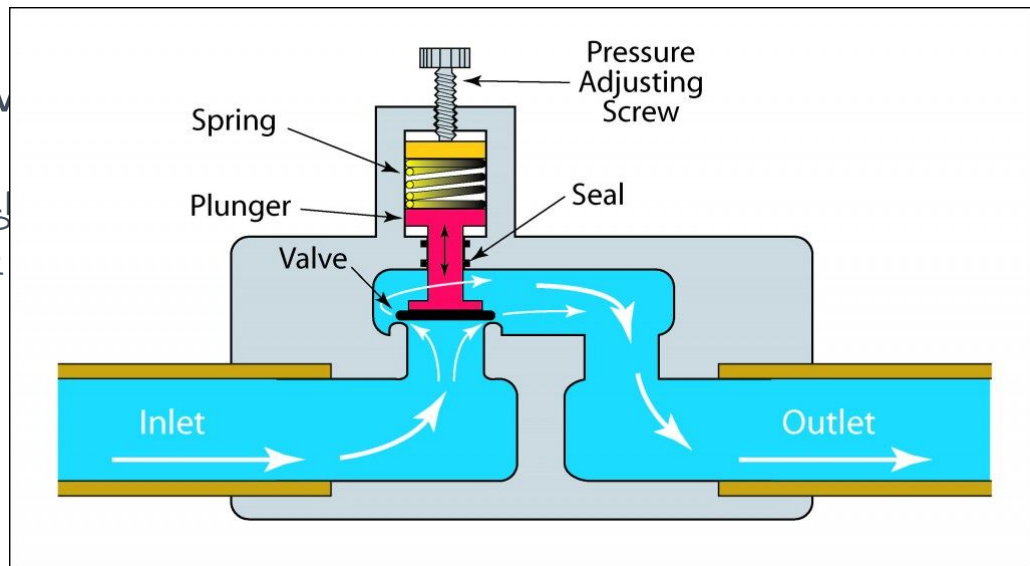


정리

- 마지막으로 성능!!!
 - Nonblocking은 이미 많이 이야기했으니...
 - backpressure
 - back pressure valve(배압밸브)
 - 일정압력에는 스프링에의해 닫힘
 - 일정 압력 이상이 되면 스프링이 압력을 이기지 못해 열리게 되어유체가 흐르게하는 밸브

정리

- 마지막으로 **성능!!!**
 - Nonblocking은 이미 많이 이야기했으니...
 - backpressure
 - back pressure valve
 - 일정압력에는
 - 일정 압력 이상 열리게 되어유



Thanks!

Any questions?