삼성 오픈소스 컨퍼런스
SAMSUNG OPEN SOURCE CONFERENCE

OPEN YOUR UNIVERSE WITH SOSCON

Going asynchronous with Netty

—

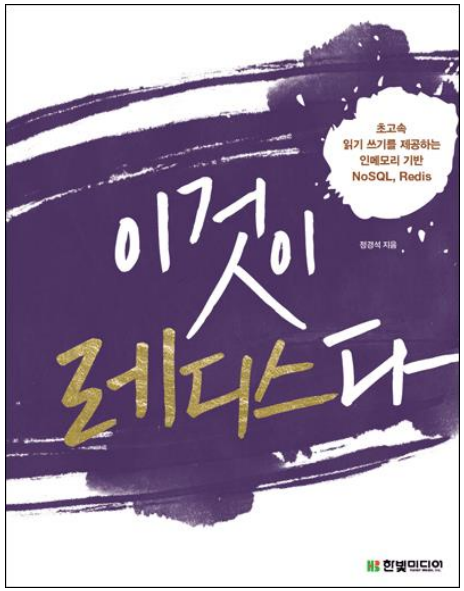
에어라이브코리아 - 인프라팀

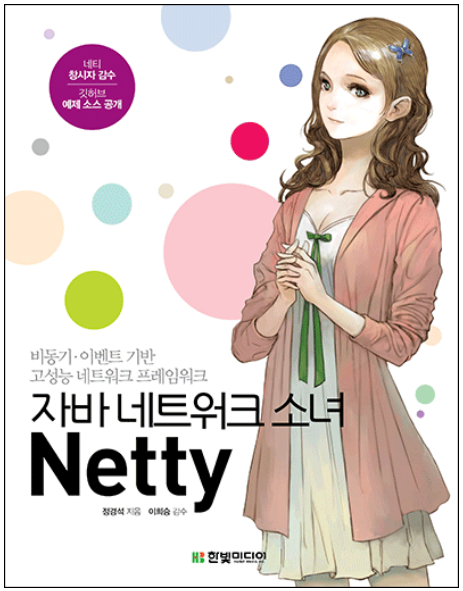
정경석

2015.10.27

발표자 소개

- 정경석
- 20세기와 21세기를 둘 다 겪은 오래된(?) 백엔드 개발자
- MS ASP를 시작으로 개발자의 길로
- 현재 AireLive 메시징 API와 영상 검색 서비스 개발





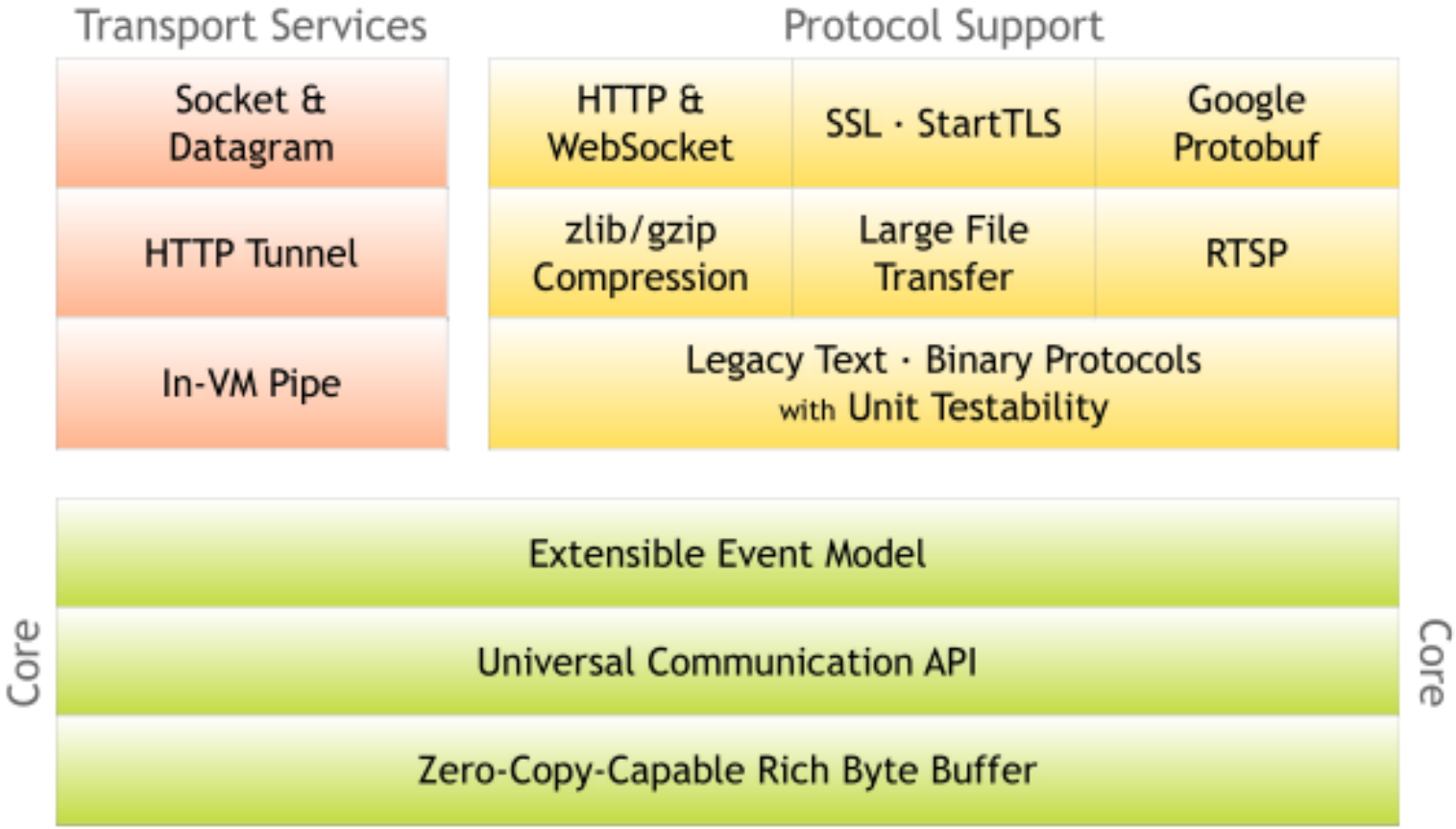
목차

- 네티란?
- 네티는 어떻게 구성되는가?
- 네티의 주요 컴포넌트
- 이벤트모델과 이벤트루프
- 네티의 비동기 처리방법
- 네티의 사용처

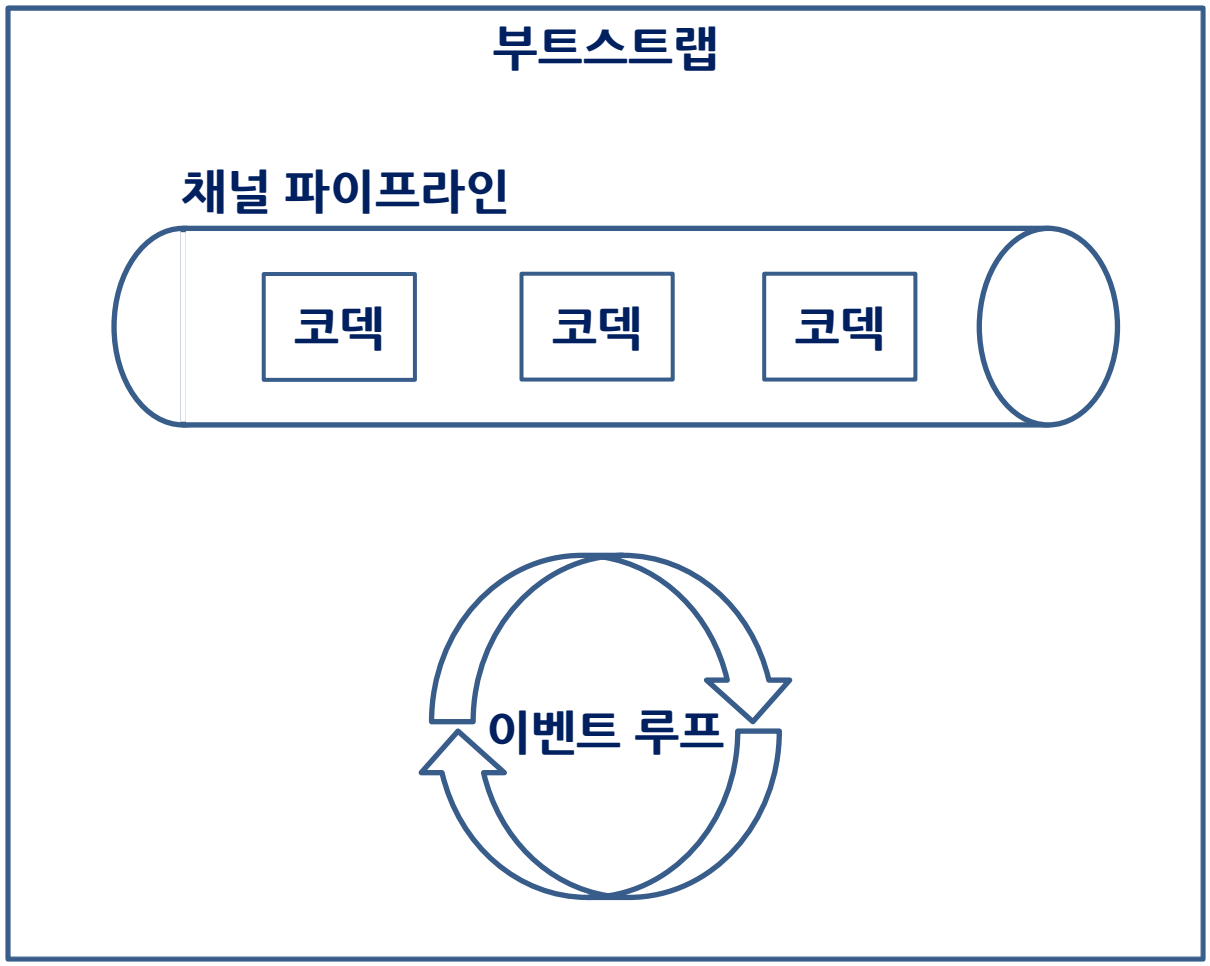
네티란?

- Java 네트워크 애플리케이션 프레임워크
- 이벤트 기반 비동기 처리
- 쉽고 빠른 네트워크 애플리케이션 작성
- IO API 추상화
- 애플, 트위터, 페이스북 등에서 사용중
- 현재 안정화 버전 3.10.5.Final, 4.0.32.Final, 5.0.0-Alpha2

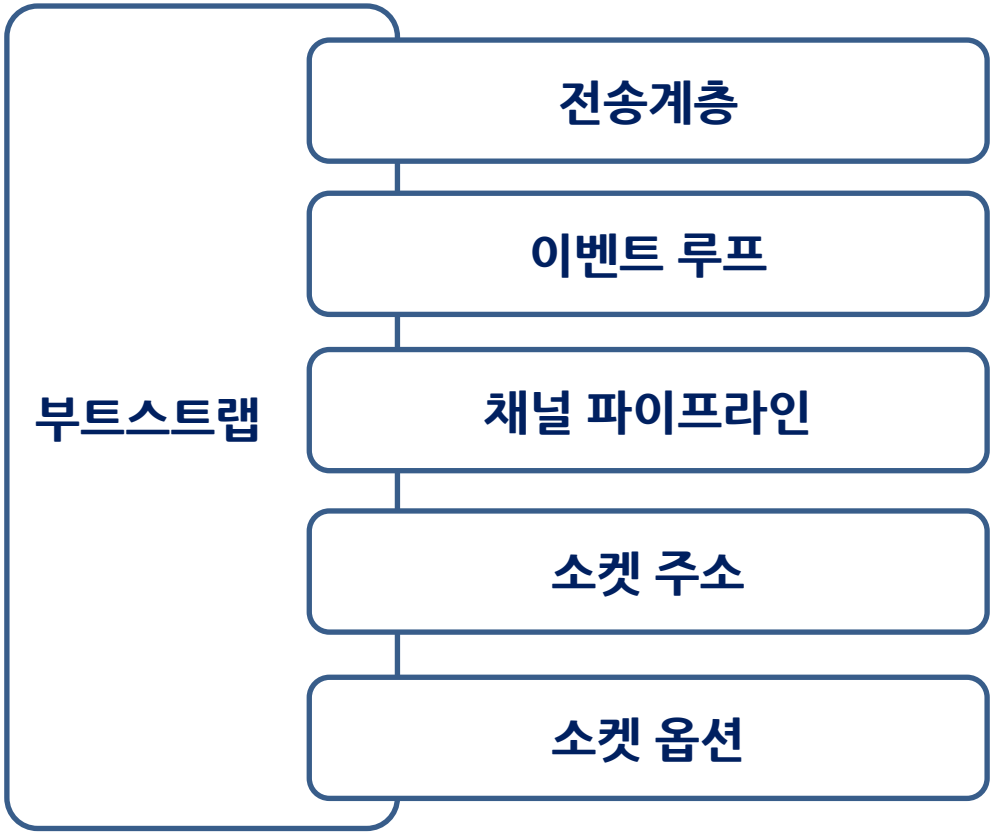
네티는 어떻게 구성되는가?



개발자가 알아야 할 구성



부트스트랩이란

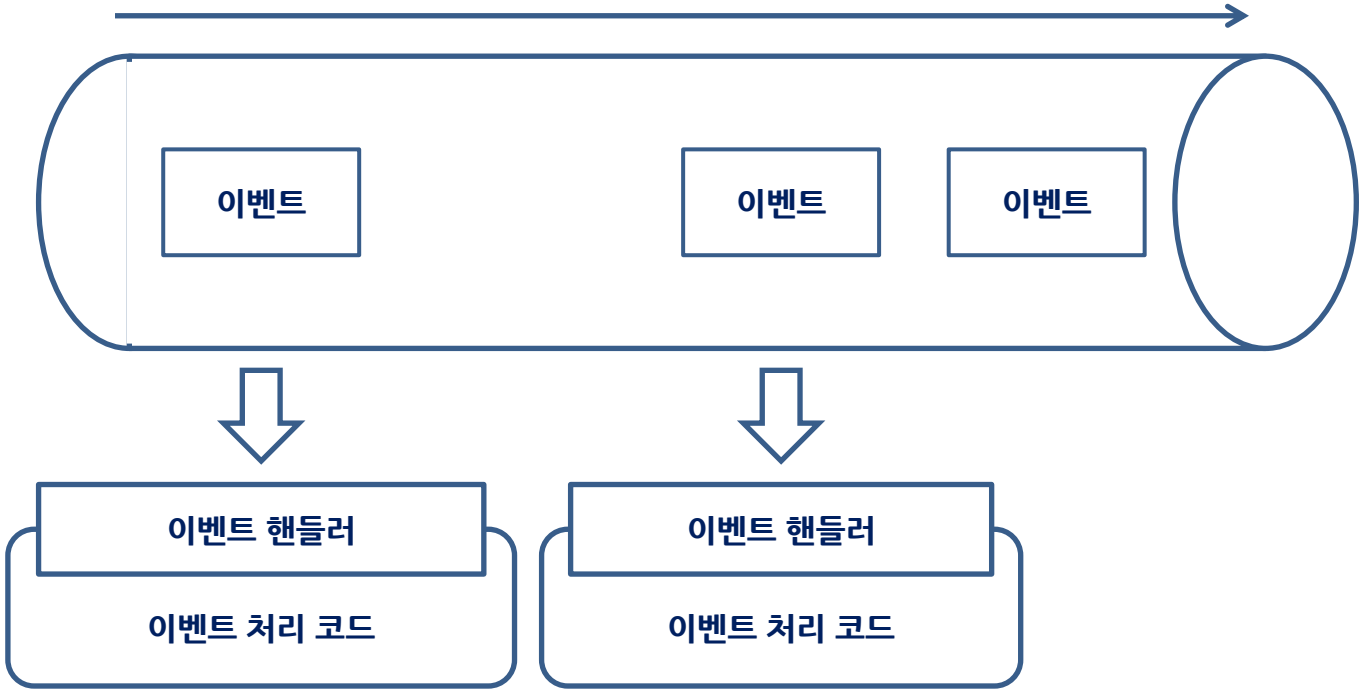


```
01.  public class EchoServer {
02.      public static void main(String[] args) throws Exception {
03.          EventLoopGroup bossGroup = new NioEventLoopGroup(1);
04.          EventLoopGroup workerGroup = new NioEventLoopGroup();
05.          try {
06.              ServerBootstrap b = new ServerBootstrap();
07.              b.group(bossGroup, workerGroup)
08.              .channel(NioServerSocketChannel.class)
09.              .childHandler(new ChannelInitializer<SocketChannel>() {
10.                  @Override
11.                  public void initChannel(SocketChannel ch) {
12.                      ChannelPipeline p = ch.pipeline();
13.                      p.addLast(new EchoServerHandler());
14.                  }
15.              });
16.
17.              ChannelFuture f = b.bind(8888).sync();
18.              f.channel().closeFuture().sync();
19.          }
20.          finally {
21.              workerGroup.shutdownGracefully();
22.              bossGroup.shutdownGracefully();
23.          }
```

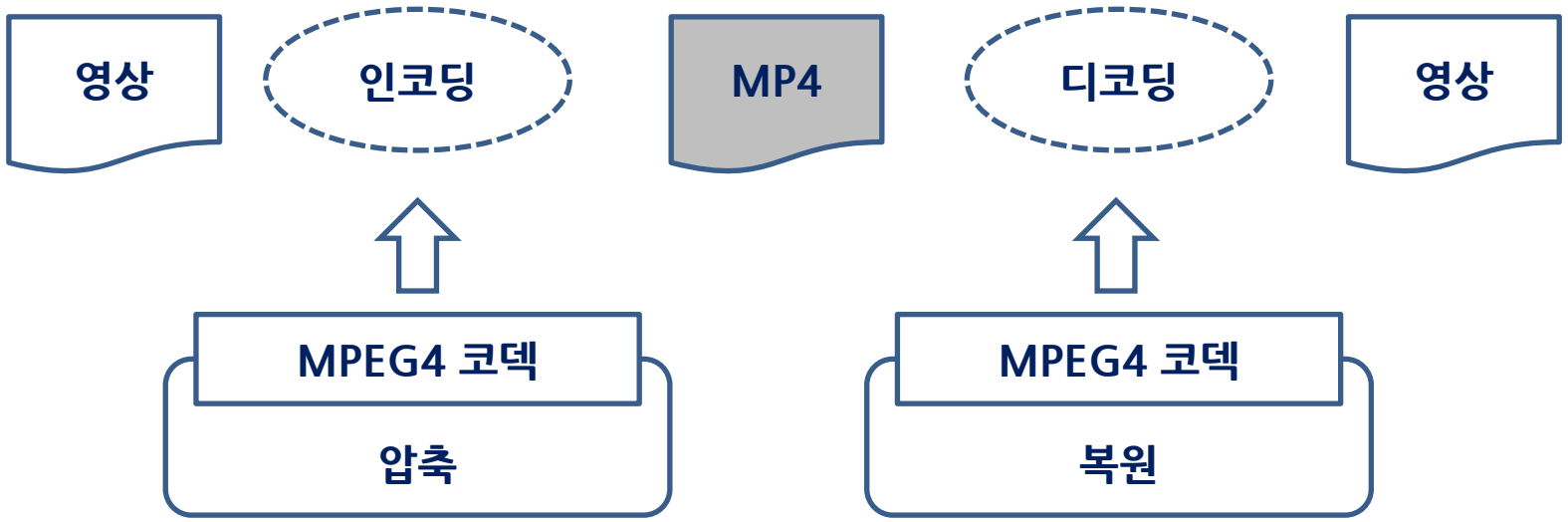
```
01.  public class EpollEchoServer {
02.      public static void main(String[] args) throws Exception {
03.          EventLoopGroup bossGroup = new EpollEventLoopGroup(1);
04.          EventLoopGroup workerGroup = new EpollEventLoopGroup();
05.           try {
06.               ServerBootstrap b = new ServerBootstrap();
07.               b.group(bossGroup, workerGroup)
08.                .channel(EpollServerSocketChannel.class)
09.                .childHandler(new ChannelInitializer<SocketChannel>() {
10.                   @Override
11.                   public void initChannel(SocketChannel ch) {
12.                       ChannelPipeline p = ch.pipeline();
13.                       p.addLast(new EchoServerHandler());
14.                   }
15.               });
16.
17.               ChannelFuture f = b.bind(8888).sync();
18.               f.channel().closeFuture().sync();
19.           }
20.           finally {
21.               workerGroup.shutdownGracefully();
22.               bossGroup.shutdownGracefully();
```

채널 파이프라인

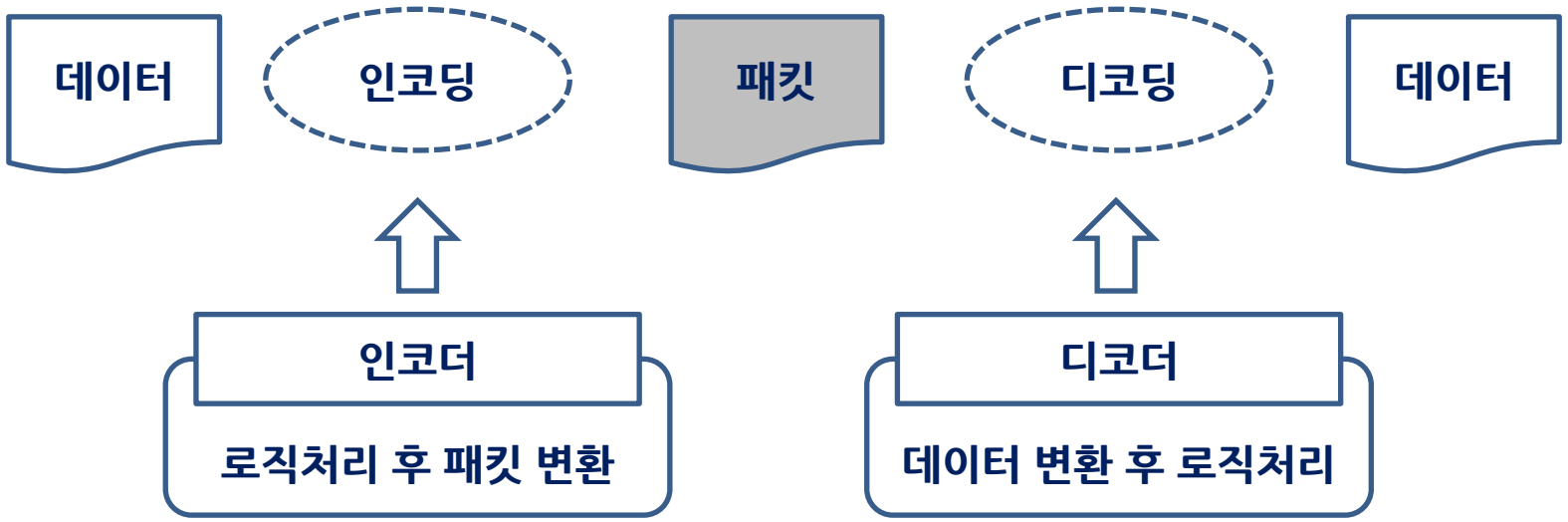
● 채널에서 발생한 이벤트가 이동하는 통로



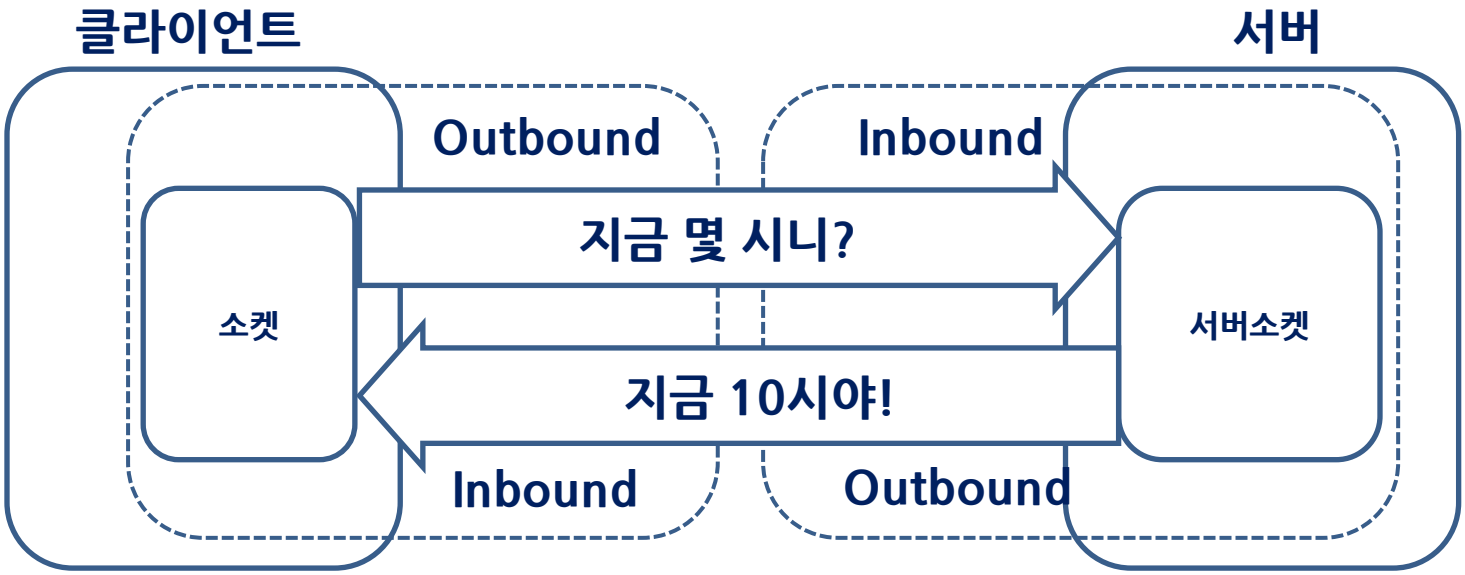
● 인코더와 디코더의 합성어



● 소켓 채널을 기준으로 인코딩과 디코딩이 이루어짐



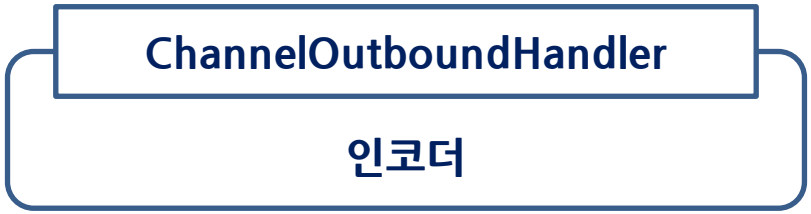
● 인바운드와 아웃바운드


클라이언트
서버
Outbound
Inbound
지금 몇 시니?
소켓
서버소켓
지금 10시야!
Inbound
Outbound

사용자 정의 코덱 작성

- ChannelInboundHandler, ChannelOutboundHandler

- channelActive
- channelInactive
- channelRead
- channelReadComplete
- channelRegistered
- ...



- bind
- close
- connect
- deregister
- disconnect
- ...

코덱

- 애플리케이션의 로직 = 코덱
- 네티의 기본 제공 코덱 목록

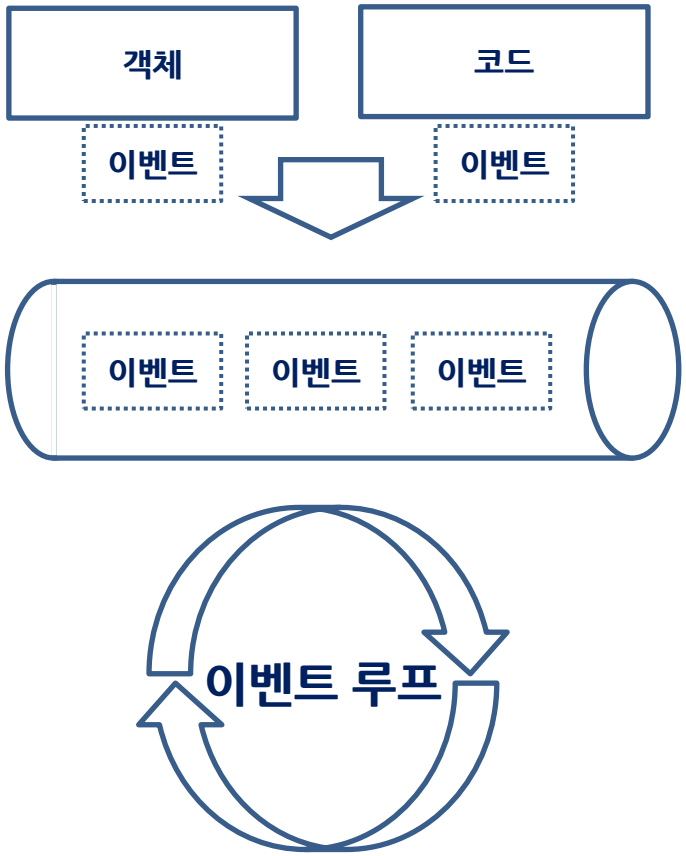
압축 코덱 zlib, gzip, Snappy	**BASE64 코덱** base64	**HTTP 코덱** websocket, CORS
marshalling 코덱 Marshaller, Unmarshaller	**spdy 코덱** TLS, SPDY	**rxtx 코덱** RxtxChannel
mqtt 코덱 MqttDecoder, MqttEncoder	**protobuf 코덱** ProtobufDecoder, ProtobufEncoder	등등 수 많은 코덱

네티의 이벤트

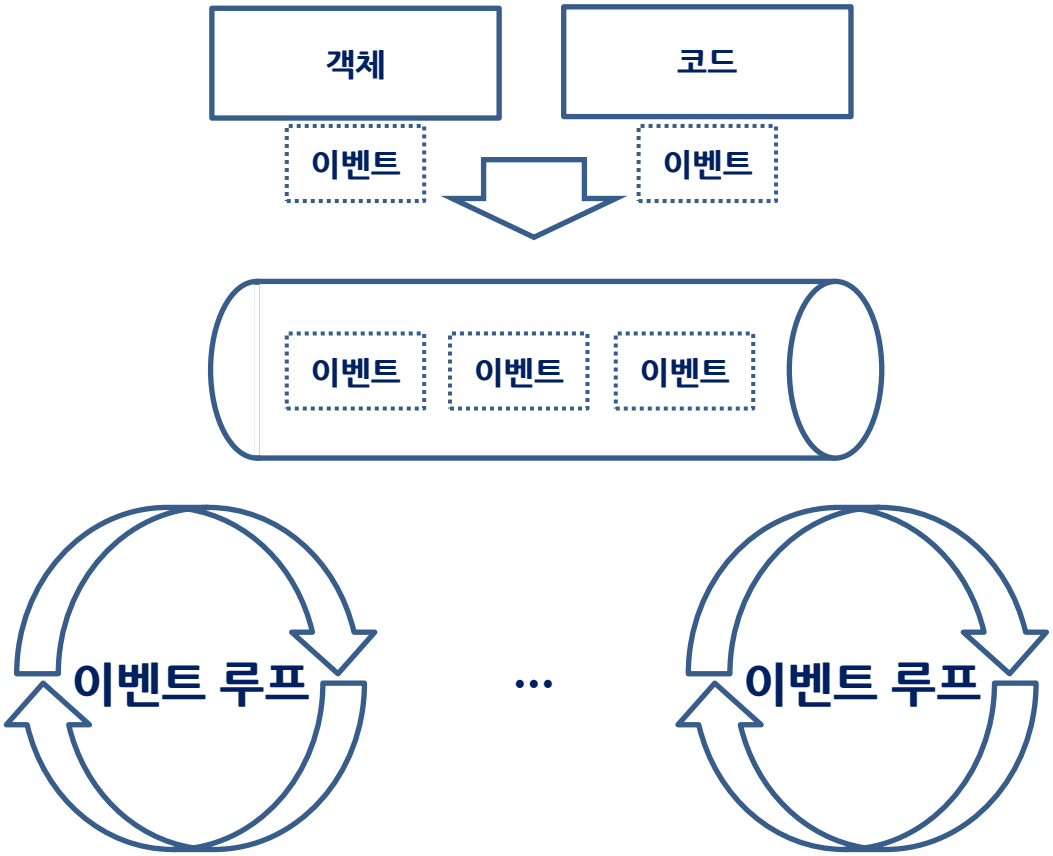
- 네티의 이벤트는 비동기로 처리됨
- 필요에 따라서 동기로 처리할 수 있음
- 이벤트는 모두 이벤트 루프에서 수행됨
- 네티는 단일 이벤트 루프, 다중 이벤트 루프를 제공
- 네티의 이벤트는 발생순서와 실행 순서가 항상 일치

이벤트루프

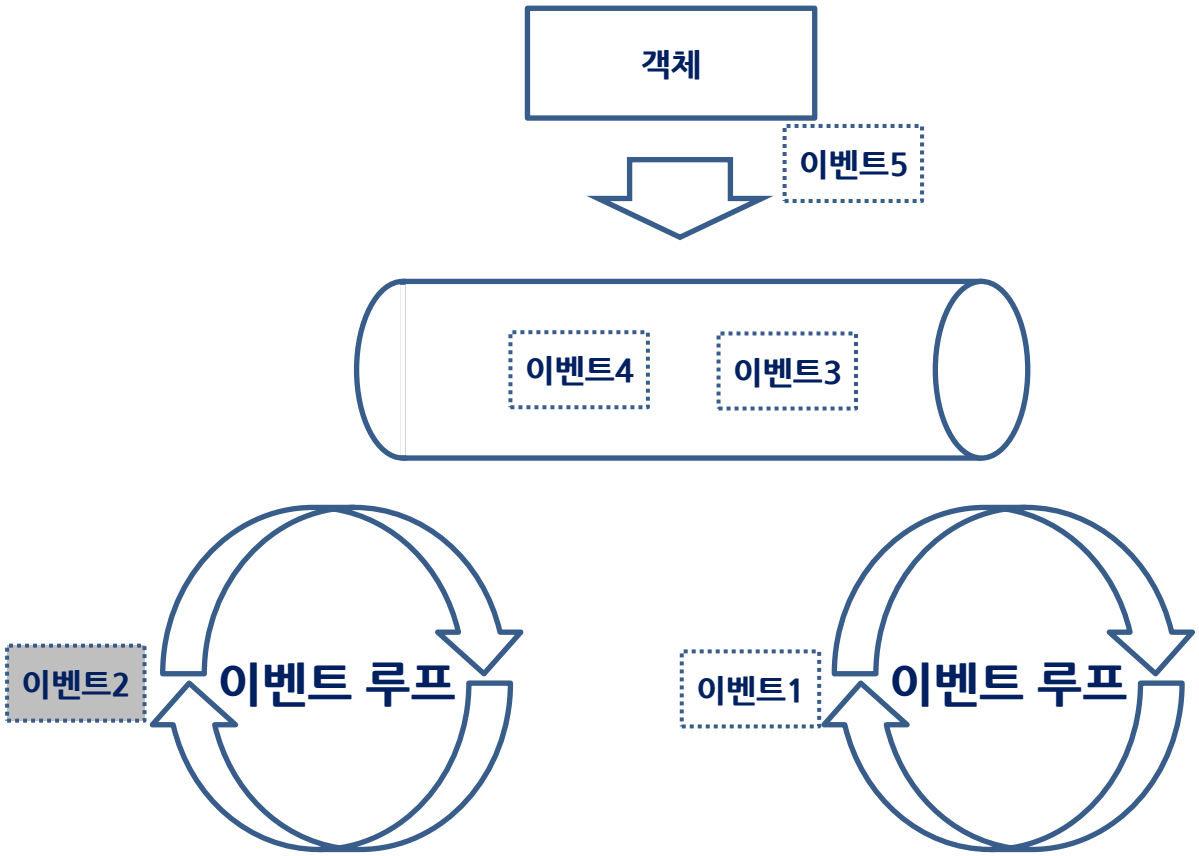
● 단일 이벤트 루프



● 다중 이벤트 루프

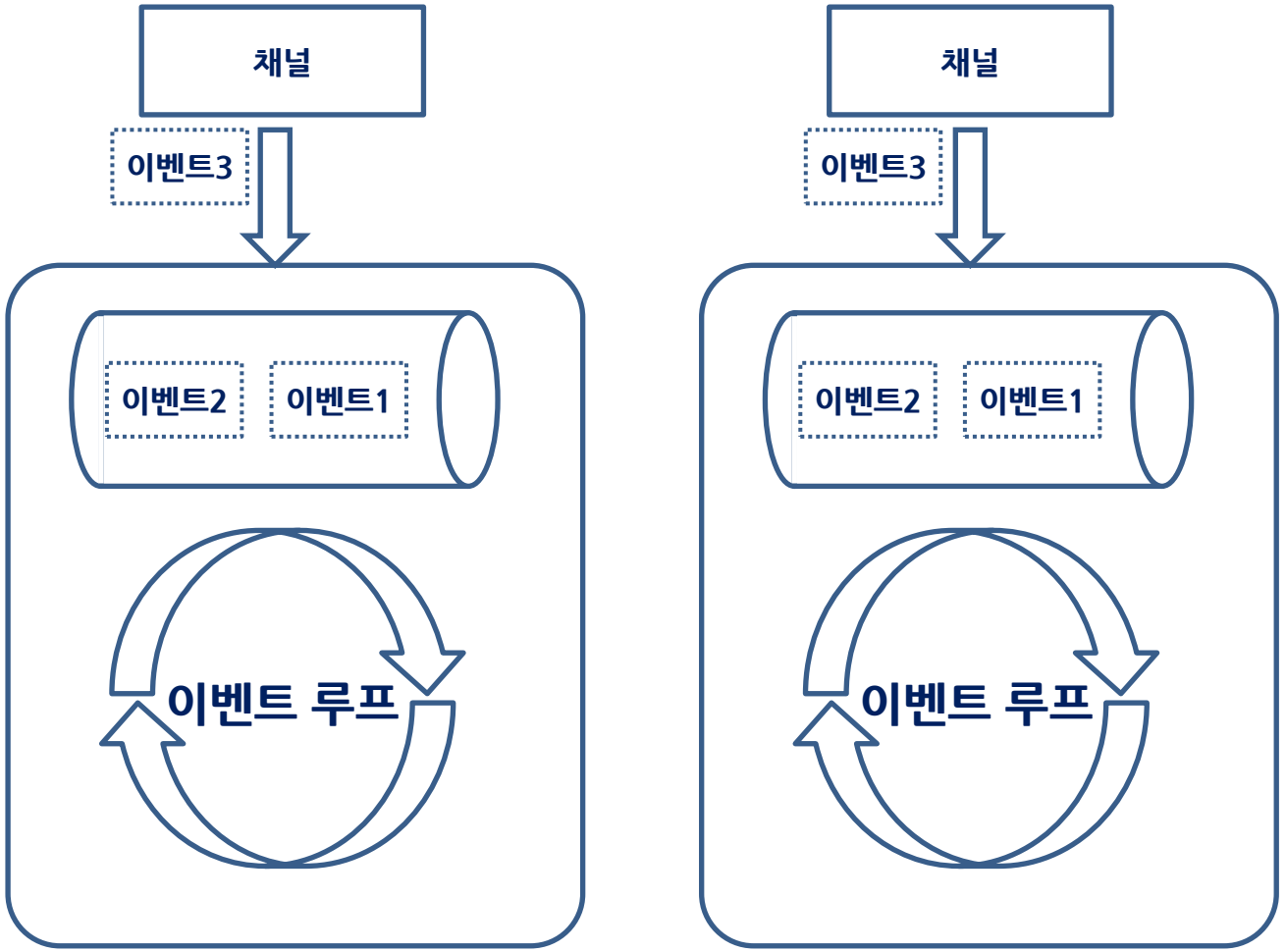


● 이벤트 처리순서



이벤트루프

● 네티의 이벤트



성능 향상 임계점

● 암달의 법칙
- 시스템의 특정 부분을 개선했을 때 얻을 수 있는 성능 향상치

$$\frac{1}{1-\text{범위} + \frac{\text{범위}}{\text{성능 향상률}}}$$

성능 향상 임계점

● 코드의 절반이 2배의 성능을 낼 수 있도록 튜닝 했을 때 얻을 수 있는 이득

$$\frac{1}{1-0.5 + \frac{0.5}{2}} = 133\%$$

네티의 비동기 처리

- ChannelHandlerContext, ChannelFuture

```
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) {
    // IO 처리 메서드 read, write, close, connect …
    ChannelFuture future = ctx.write(msg);

    // write 메서드의 작업이 완료되었을 때 이벤트를 받을 수 있는 리스너 설정
    future.addListener(listener);

    // IO 작업이 완료되었는지 확인할 수 있음.
    future.isDone();

    // IO 작업이 완료될 때까지 대기
    future.sync();
}
```

네티의 비동기 처리

- ChannelFutureListener

네티가 제공하는 기본 ChannelFutureListener

- ChannelFutureListener.CLOSE
- ChannelFutureListener.CLOSE_ON_FAILURE
- ChannelFutureListener.FIRE_EXCEPTION_ON_FAILURE

```
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) {
    ChannelFuture future = ctx.write(msg);

    future.addListener(ChannelFutureListener.CLOSE );
}
```

네티의 사용처

- Facebook
  - Netty-based Thrift transport implementation, Nifty
- Apple
  - Java service server
- AireLive
  - TCP based Messaging server, HTTP based API server
- ElasticSearch core
  - HTTP based API service, Client transport layer
- Apache Spark
  - Cluster data transfer
- Google - gRPC, Red Hat - Infinispan, Typesafe - Play Framework

마치며

- 네티 4.0이상부터 안드로이드를 공식 지원
- 하나의 네티 애플리케이션에서 두 개의 포트를 바인딩
  - 부트스트랩의 바인드 메서드를 두 번 호출
- 성능은 애플리케이션이 제공하는 기능에 따라서 달라진다.
  - 데이터베이스 조회 vs 캐시 조회
- 이벤트 루프의 개수는 테스트를 통해서 선택하라.
  - nGrinder, Jmeter
- 네티의 ChannelFuture.sync() 메서드는 신중히 사용할 것.
- 네티의 Channel.write() 메서드는 내부적으로 requestQueue를 관리함.
- 사용자 정의 코덱은 EmbeddedChannel 클래스를 통해서 테스트

THANK YOU!