Asynchronous, Event-driven Network Application Development with Netty

Rapid development of maintainable high performance protocol servers & clients

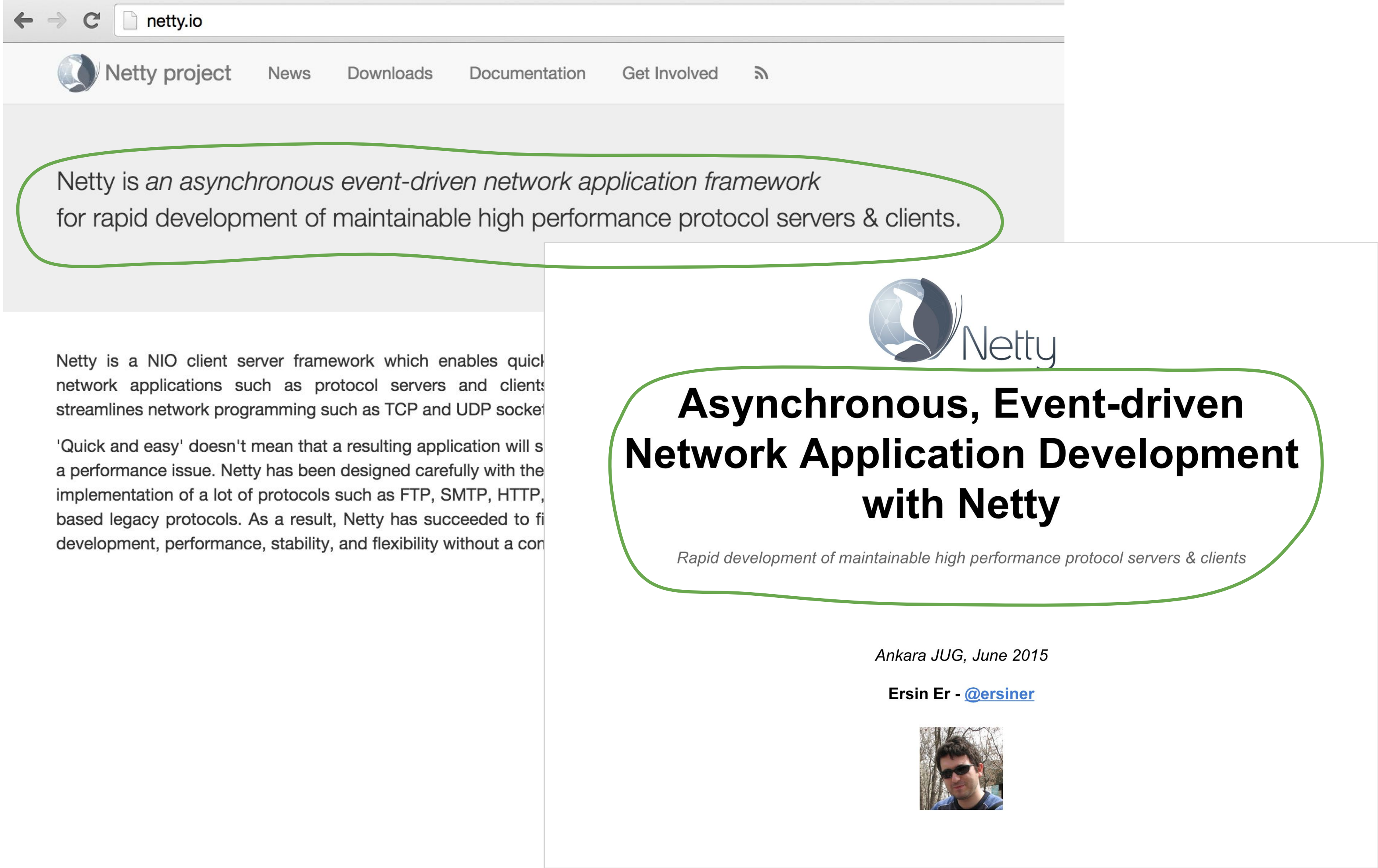
Ersin Er - @ersiner

About the speaker, Ersin Er

Summary of what's on LinkedIn

- Computer Sci. Student @ Hacettepe Univ. - BSc ('03), MSc ('06), ~~PhD~~ (~)
- Teaching Assistant [and System Admin.] @ Hacettepe Univ. ('03-'11)
- Committer and PMC Member @ Apache Software Foundation ('05-'10)
- Software Architect @ Peak Games ('11-'13)
- Co-founder and Solutions Architect @ Metasolid ('14-'15)
- Software/Solutions Architect @ Arçelik ('16-...)
- *Hands-on Solutions and Software Architect.*
- *Distributed Systems, Concurrency and Performance Programming enthusiast.*
- *Does tech biz, manages projects, deals with people.*
- *Used to be a perfectionist. Does not produce garbage.*

How to name your presentation



Netty is *an asynchronous event-driven network application framework* for rapid development of maintainable high performance protocol servers & clients.

Netty is a NIO client server framework which enables quick
network applications such as protocol servers and clients
streamlines network programming such as TCP and UDP socke

'Quick and easy' doesn't mean that a resulting application will s
a performance issue. Netty has been designed carefully with the
implementation of a lot of protocols such as FTP, SMTP, HTTP,
based legacy protocols. As a result, Netty has succeeded to fi
development, performance, stability, and flexibility without a cor

Asynchronous, Event-driven Network Application Development with Netty

Rapid development of maintainable high performance protocol servers & clients

Ankara JUG, June 2015

Ersin Er - @ersiner

Today’s Agenda

- Blocking vs Non-Blocking I/O
- NIO and Netty Abstractions
- What sets Netty apart
- Netty Reusables
- Netty & HTTP
- Servlet 3.0 and 3.1 (for our beloved JavaEE friends 🙊)
- Netty Ecosystem

Do not expect a 1-1 matched flow with these titles.

I prefer discussions to static information dumping.

Let’s review
Blocking Socket I/O in Java
by examples
in order to steer
our discussions on
Non-Blocking I/O,
Java NIO
and Netty.

Blocking I/O Socket Server Example

Single Client, Single Request

```java
public static void serve1() throws IOException {
   ServerSocket serverSocket = new ServerSocket(10000);

   Socket clientSocket = serverSocket.accept();

   InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
   BufferedReader in = new BufferedReader(isr);
   PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

   String request, response;
   if ((request = in.readLine()) != null) {
       response = request + " processed at " + new Date();
       out.println(response);
   }

   in.close();
   out.close();
}
```

Protocol Implementation

Blocking I/O Socket Server Example

Single Client, Multi Request

```
public static void serve2() throws IOException {
   ServerSocket serverSocket = new ServerSocket(10000);

   Socket clientSocket = serverSocket.accept();

   InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
   BufferedReader in = new BufferedReader(isr);
   PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

   String request, response;
   while ((request = in.readLine()) != null) {
       response = request + " processed at " + new Date();
       out.println(response);
   }

   in.close();
   out.close();
}
```

**Wow!
That was
easy!**

Blocking I/O Socket Server Example

Single Client, Multi Request, Client Exit Control

```java
public static void serve3() throws IOException {
    ServerSocket serverSocket = new ServerSocket(10000);
    Socket clientSocket = serverSocket.accept();
    InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
    BufferedReader in = new BufferedReader(isr);
    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
    String request, response;

    while ((request = in.readLine()) != null) {
        if ("Exit".equals(request)) {
            break;
        }
        response = request + " processed at " + new Date();
        out.println(response);
    }

    in.close();
    out.close();
}
```

More Protocol

Blocking I/O Socket Server Example

Multi Client, Multi Request, Client Exit Control

A new generation of Internet service supporting multiple users?..

```java
public static void serve4() throws IOException {
    ServerSocket serverSocket = new ServerSocket(10000);
    while (true) {
        Socket clientSocket = serverSocket.accept();
        InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
        BufferedReader in = new BufferedReader(isr);
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        String request, response;
        while ((request = in.readLine()) != null) {
            if ("Exit".equals(request)) {
                break;
            }
            response = request + " processed at " +
            out.println(response);
        }
        in.close();
        out.close();
    }
}
```

Blocking I/O Socket Server Example

Concurrent Clients, Multi Request, Client Exit Control

```java
public static void serve5() throws IOException {
    ServerSocket serverSocket = new ServerSocket(10000);
    while (true) {
        final Socket clientSocket = serverSocket.accept();
        new Thread() {
            public void run() {
                try {
                    InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
                    BufferedReader in = new BufferedReader(isr);
                    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                    String request, response;
                    while ((request = in.readLine()) != null) {
                        if ("Exit".equals(request)) { break; }
                        response = request + " processed at " + new Date(); out.println(response);
                    }
                    in.close(); out.close();
                } catch (IOException e) { e.printStackTrace(); }
            }
        }.start();
    }
}
```

Uncontrolled Power!

!!!

These are far from being production code. In fact, they are terrible ones.

Multi-Threaded Blocking I/O Exhausting Resources
What to do?

```
public static void serve5() throws IOException {
   ServerSocket serverSocket = new ServerSocket(10000);
   while (true) {
      final Socket clientSocket = serverSocket.accept();
      new Thread() {
         public void run() {
            try {
               InputStreamReader i                                       ());
               BufferedReader in =
               PrintWriter out = n
               String request, res
               while ((request = i
                   if ("Exit".equal
                   response = reque                                       se);
               }
               in.close(); out.close();
            } catch (IOException e) { e.printStackTrace(); }
         }
      }.start();
   }
}
```

Uncontrolled Power!

What to do?

- **Pooling?**
- **Executors?**
- **Queuing?**

!!!

These are far from being production code. In fact, they are terrible ones.

Multi-Threaded Blocking I/O Exhausting Resources
What to do? - Critical Discussion

```
public static void serve5() throws IOException {
   ServerSocket serverSocket = new ServerSocket(10000);
   while (true) {
       final Socket clientSocket = serverSocket.accept();
       new Thread() {
           public void run() {
               try {
                  InputStreamReader i
                  BufferedReader in =
                  PrintWriter out = n
                  String request, res
                  while ((request = i
                      if ("Exit".equal
                      response = reque
                  }
                  in.close(); out.close();
               } catch (IOException e) { e.printStackTrace(); }
           }
       }.start();
   }
}
```

Uncontrolled Power!

What to do

- **Pooling?**
- **Executor**
- **Queuing**

The discussion here is critical for switching (our minds) to Non-Blocking I/O, Java NIO and Netty

These are far from being production code. In fact, they are terrible ones.

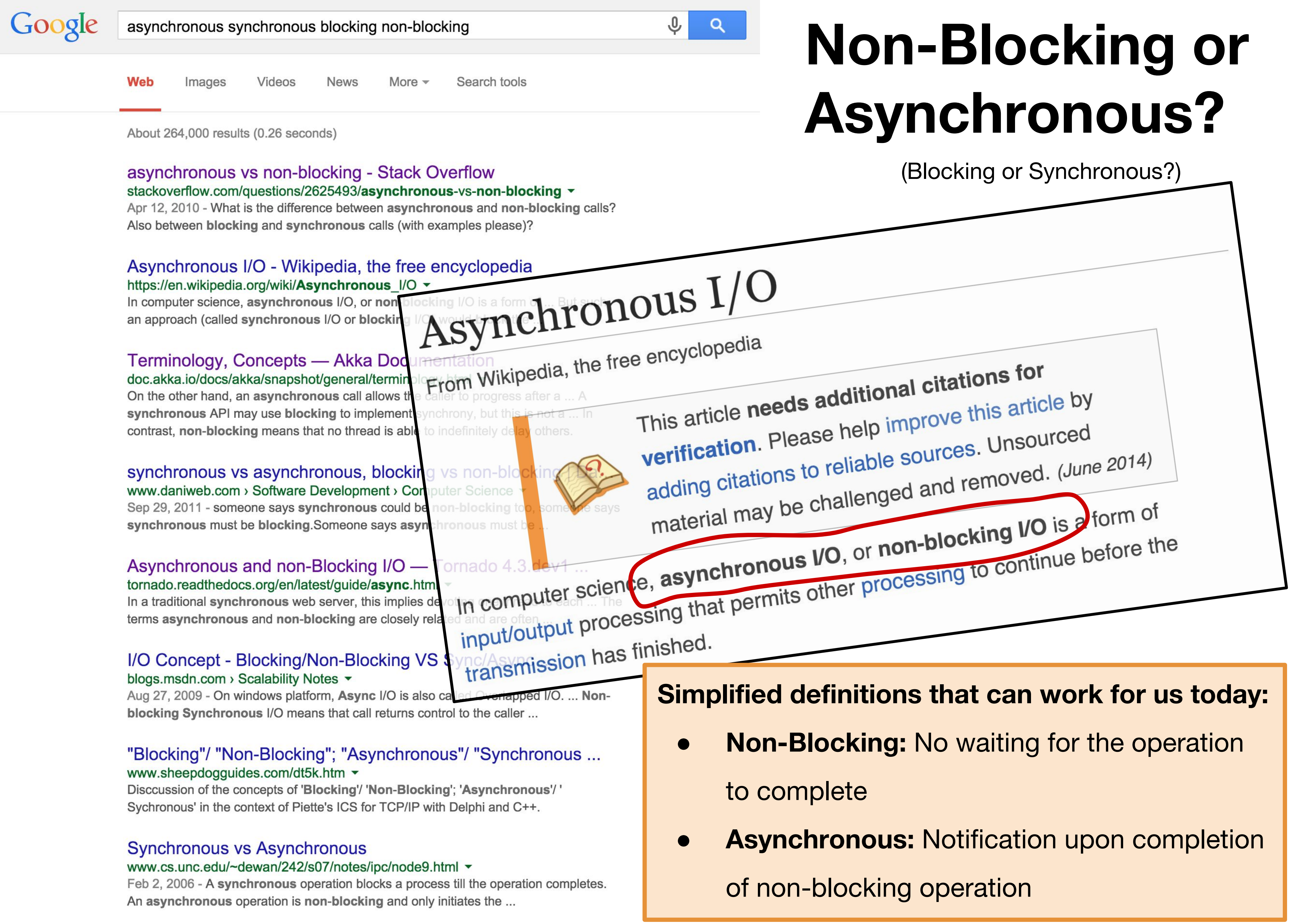
Non-Blocking I/O

- Sockets in Non-Blocking Mode
- I/O Multiplexing
  - *epoll* (Linux)
  - *kqueue* (FreeBSD)
  - *IOCP* (Windows, Solaris)

- Single Thread for many sockets (or file descriptors)
- Key to high performance servers

Non-Blocking or Asynchronous?

(Blocking or Synchronous?)



asynchronous synchronous blocking non-blocking

About 264,000 results (0.26 seconds)

asynchronous vs non-blocking - Stack Overflow
stackoverflow.com/questions/2625493/asynchronous-vs-non-blocking
Apr 12, 2010 - What is the difference between asynchronous and non-blocking calls? Also between blocking and synchronous calls (with examples please)?

Asynchronous I/O - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Asynchronous_I/O
In computer science, asynchronous I/O, or non-blocking I/O is a form of ... But such an approach (called synchronous I/O or blocking I/O) would block the ...

Terminology, Concepts — Akka Documentation
doc.akka.io/docs/akka/snapshot/general/terminology.html
On the other hand, an asynchronous call allows the caller to progress after a ... A synchronous API may use blocking to implement synchrony, but this is not a ... In contrast, non-blocking means that no thread is able to indefinitely delay others.

synchronous vs asynchronous, blocking vs non-blocking | DaniWeb
www.daniweb.com › Software Development › Computer Science
Sep 29, 2011 - someone says synchronous could be non-blocking too, someone says synchronous must be blocking.Someone says asynchronous must be ...

Asynchronous and non-Blocking I/O — Tornado 4.3.dev1 ...
tornado.readthedocs.org/en/latest/guide/async.html
In a traditional synchronous web server, this implies devoting one thread to each ... The terms asynchronous and non-blocking are closely related and are often ...

I/O Concept - Blocking/Non-Blocking VS Sync/Async
blogs.msdn.com › Scalability Notes
Aug 27, 2009 - On windows platform, Async I/O is also called Overlapped I/O. ... Non-blocking Synchronous I/O means that call returns control to the caller ...

"Blocking"/ "Non-Blocking"; "Asynchronous"/ "Synchronous ...
www.sheepdogguides.com/dt5k.htm
Disccussion of the concepts of 'Blocking'/ 'Non-Blocking'; 'Asynchronous'/ ' Sychronous' in the context of Piette's ICS for TCP/IP with Delphi and C++.

Synchronous vs Asynchronous
www.cs.unc.edu/~dewan/242/s07/notes/ipc/node9.html
Feb 2, 2006 - A synchronous operation blocks a process till the operation completes. An asynchronous operation is non-blocking and only initiates the ...

Asynchronous I/O

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(June 2014)*

In computer science, **asynchronous I/O**, or **non-blocking I/O** is a form of input/output processing that permits other processing to continue before the transmission has finished.

Simplified definitions that can work for us today:

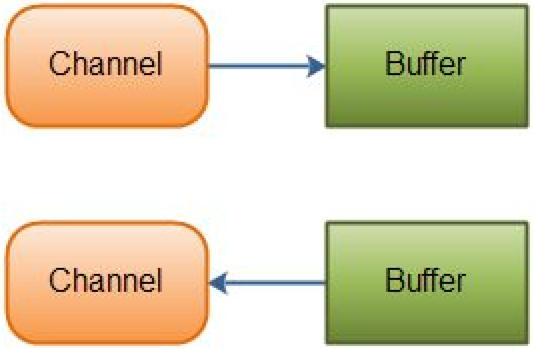
- **Non-Blocking:** No waiting for the operation to complete
- **Asynchronous:** Notification upon completion of non-blocking operation

Java NIO

- NIO = Non-Blocking I/O?
- It’s Java New I/O
- It’s no longer new (came with Java 1.4)
  - Java 7 comes with V2
- Not only for Socket I/O
- First Class Citizens
  - **Channels**
  - **Buffers**
  - **Selectors**

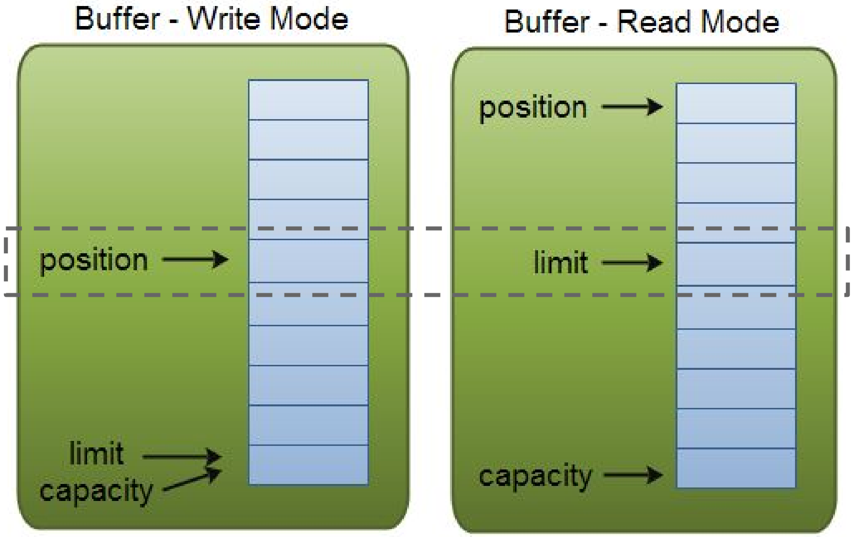
NIO - Channels and Buffers

- Channels
  - FileChannel
  - DatagramChannel
  - SocketChannel
  - ServerSocketChannel
- Buffers
  - ByteBuffer
  - CharBuffer
  - DoubleBuffer
  - FloatBuffer
  - IntBuffer
  - LongBuffer
  - ShortBuffer



- Data are *read* **from Channels into Buffers**
- Data are *written* **from Buffers into Channels**

NIO - Buffers are serious business



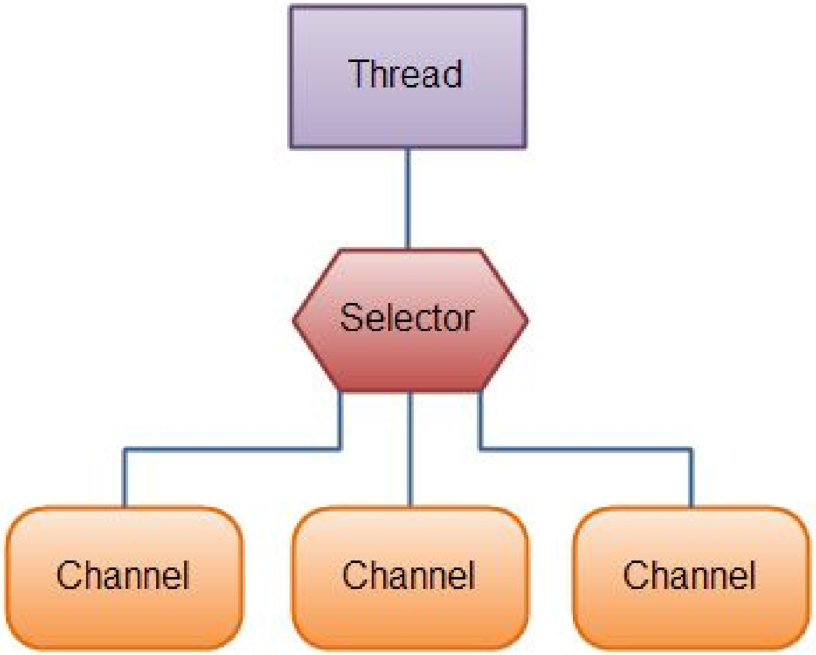
Buffer capacity, position and limit in write and read mode.

Buffer.flip() makes the mode change.

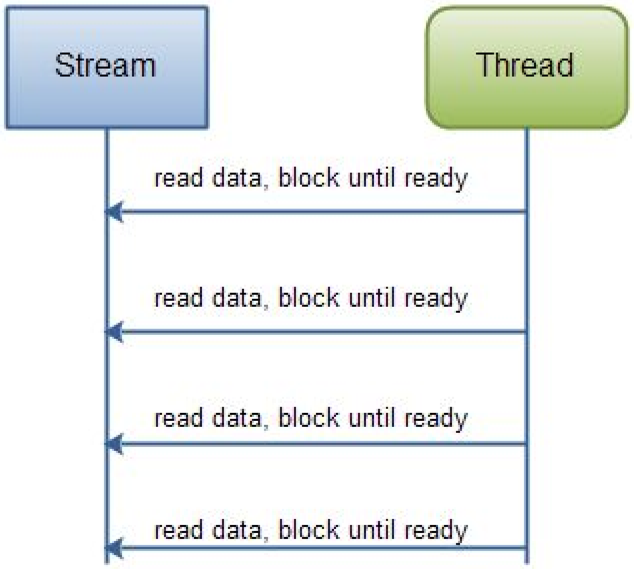
We also have:

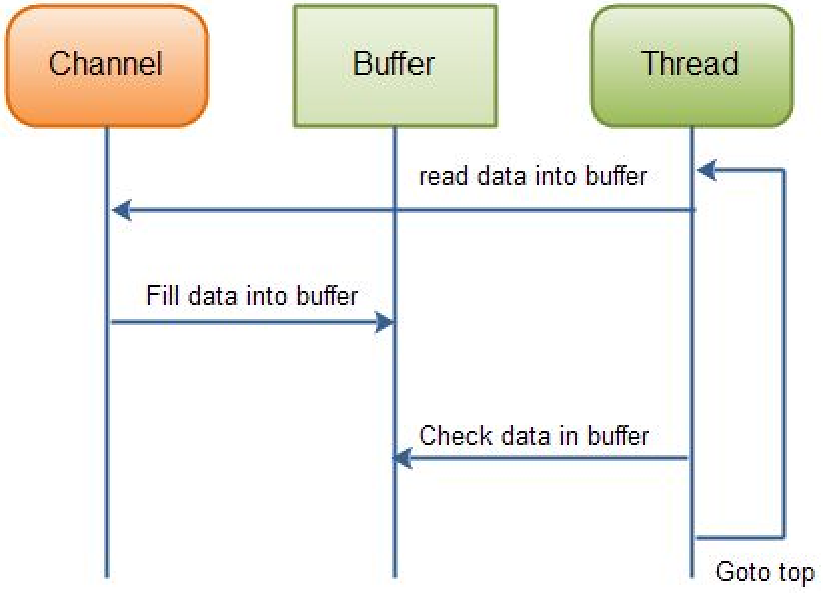
- Heap Buffers
  - Array Based
  - ByteBuffer.allocate()
- Direct Buffers
  - Off-Heap
  - ByteBuffer.allocateDirect()

NIO - Selectors & I/O Multiplexing



OIO vs NIO


Stream
Thread
read data, block until ready
read data, block until ready
read data, block until ready
read data, block until ready


Channel
Buffer
Thread
read data into buffer
Fill data into buffer
Check data in buffer
Goto top

From NIO to Netty

Using NIO directly is like using Naked Threads.

Netty replaces NIO APIs with superiors and provides incredible capabilities.

Netty Core Components and Utilities

- `Channel`s and Transports
- `ByteBuf` and Un/Pooled Allocation Management
- `ChannelHandler`s and `ChannelPipeline`
- The Codec Framework and Reusable Codecs
- `Bootstrap`s and `ChannelInitializer`s
- `Future`s and `EventLoop`s

Channels and Transports

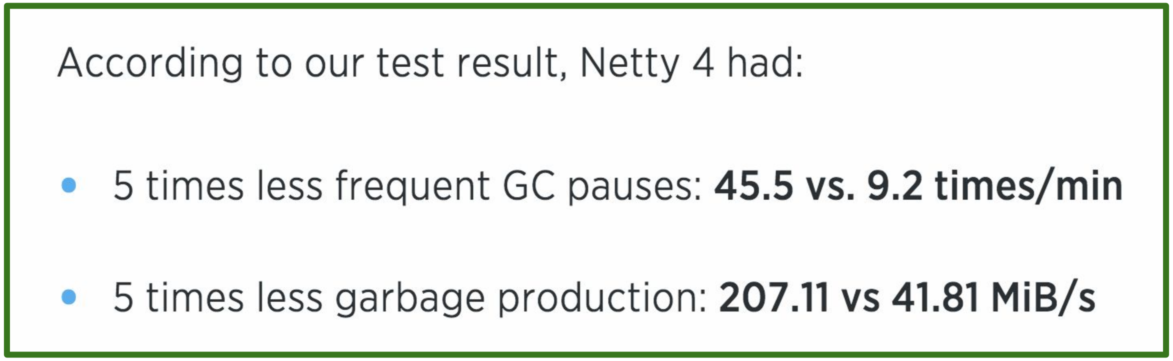
Package View

- io.netty.channel.**embedded**
- io.netty.channel.**epoll**
- io.netty.channel.**local**
- io.netty.channel.**nio**
- io.netty.channel.**oio**
- io.netty.channel.**rxtx**
- io.netty.channel.**sctp**
- io.netty.channel.**sctp.nio**
- io.netty.channel.**sctp.oio**
- io.netty.channel.**socket**
- io.netty.channel.**socket.nio**
- io.netty.channel.**socket.oio**
- io.netty.channel.**udt**
- io.netty.channel.**udt.nio**
- io.netty.channel.**unix**

- You can both read from write into Channels (they are *duplex* as opposed to streams)
- All I/O operations on channels are ***asynchronous*** and return ***listenable futures***
- Channels are implemented for various Transports types:
    - Unified API for **NIO** and **OID** (and others)
    - **Epoll** transport for extreme performance
    - **Local** transport for in VM communication
    - **Embedded** transport for Unit Testing

`ByteBuf` and Un/Pooled Allocation Management

- **`ByteBuf`** is improved version of `ByteBuffer`
- **`ByteBuf`** has *both write and read index*, does not need `flip()`
- **`CompositeByteBuf`** enables *Zero-Copy*
- **`ByteBuf`**s can be *pooled* for reducing garbage collector pressure

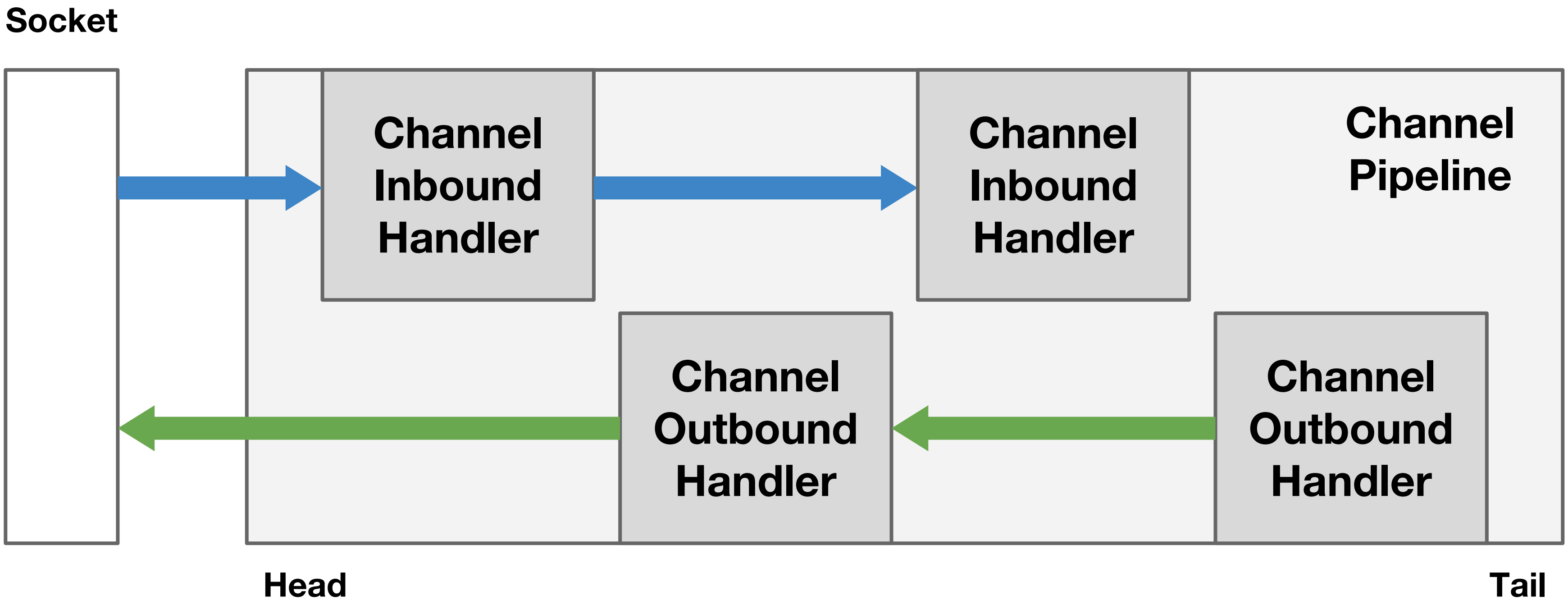


According to our test result, Netty 4 had:

- 5 times less frequent GC pauses: **45.5 vs. 9.2 times/min**
- 5 times less garbage production: **207.11 vs 41.81 MiB/s**

https://blog.twitter.com/2013/netty-4-at-twitter-reduced-gc-overhead

ChannelHandlers and ChannelPipeline



ChannelPipeline has been designed with Intercepting Filter pattern and resembles Servlet Filters

Codec Framework

- Simplified and focused API on top of ChannelHandlers
- Decoders are ChannelInboundHandlers
- Encoders are ChannelOutboundHandlers
- Codecs are both CIH and COH

Reusable Codecs

- io.netty.handler.codec.**base64**
- io.netty.handler.codec.**bytes**
- io.netty.handler.codec.**compression**
- io.netty.handler.codec.**haproxy**
- io.netty.handler.codec.**http**
- io.netty.handler.codec.**http.cookie**
- io.netty.handler.codec.**http.cors**
- io.netty.handler.codec.**http.multipart**
- io.netty.handler.codec.**http.websocketx**
- io.netty.handler.codec.**marshalling**
- io.netty.handler.codec.**protobuf**
- io.netty.handler.codec.**rtsp**
- io.netty.handler.codec.**sctp**
- io.netty.handler.codec.**serialization**
- io.netty.handler.codec.**socks**
- io.netty.handler.codec.**spdy**
- io.netty.handler.codec.**string**
- io.netty.handler.**logging**
- io.netty.handler.**ssl**
- io.netty.handler.**stream**
- io.netty.handler.**timeout**
- io.netty.handler.**traffic**

Some primitive ones

- DelimiterBasedFrameDecoder
- LengthFieldBasedFrameDecoder
- FixedLengthFrameDecoder
- LineBasedFrameDecoder

All of these represent patterns of protocol development

Bootstraps and ChannelInitializers

`Bootstraps` help bootstrap channels (server or client side)

- Set `EventLoopGroups`
- Set `ChannelHandlers`
- Bind to Network Interfaces

`ChannelInitializer` is a special ChannelHandler

- Handles **`channelRegistered`** event and applies its **pipeline** config to the channel
- (Suggested: See its source code)

Futures and EventLoops

- All I/O operations on Channels return listenable futures
- Each Channel is assigned to a single EventLoop and stays so during its lifetime
- EventLoops handle all I/O operations of their Channels
- EventLoopGroups are like Thread Pools and number of EventLoops they manage depends on number of CPU cores (and possible other factors)
- Listeners registered to Futures are handled by appropriate EventLoop selected by Netty

Now, Examples

Servlet

- Servlet 3.0 - Async Processing of Response
- Servlet 3.1 - Non-Blocking Processing of Request (Content)

🙊

Did you expect more?
Come on, this is Netty :-)

Netty Versions

- **3.x** Old, Stable
- **4.0.x** Active, Stable
  - Huge improvements over 3.x
  - https://github.com/netty/netty/wiki/New-and-noteworthy-in-4.0
- **4.1** Current, Stable
  - Mostly backward compatible with 4.0
  - Android support and lots of new codecs
  - https://github.com/netty/netty/wiki/New-and-noteworthy-in-4.1
- **5.0** Alpha - Backward Incompatible Improvements
  - https://github.com/netty/netty/wiki/New-and-noteworthy-in-5.0

Ecosystem - Related Projects

(The ones I've been interested in and mostly using Netty at its heart)

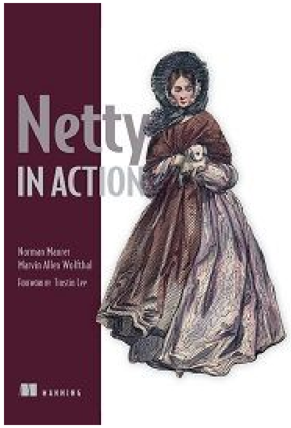
- **Vert.x** - A toolkit for building reactive applications on the JVM
- **Ratpack** - Simple, lean & powerful HTTP apps
- **async-http-client** - Asynchronous Http and WebSocket Client library for Java
- **RxNetty** - Reactive Extension (Rx) Adaptor for Netty
- **nettosphere** - A Java WebSocket/HTTP server based on the Atmosphere and Netty Framework
- **grpc-java** - The Java gRPC implementation (by Google)
- **Play Framework** - The High Velocity Web Framework For Java and Scala
- *More at http://netty.io/wiki/related-projects.html and http://netty.io/wiki/adopters.html*

Ecosystem - Resources

- **Netty Documentation & Examples**
    - http://netty.io/wiki/index.html

- **StackOverflow**
    - http://stackoverflow.com/questions/tagged/netty

- **Netty in Action**
    - http://www.manning.com/maurer/



Thanks

- You, attenders!
- The organizers, for having me here
- Jakob Jenkov, for allowing me use his diagrams for explaining NIO Concepts (http://tutorials.jenkov.com/java-nio/index.html)