Why **Netty** ?

@Netflix 2014 · Las Gatos · 2014/07/29

Norman Maurer, Principal Software Engineer / Leading Netty efforts
@ Red Hat Inc.

- Netty / All things NIO
- Author of Netty in Action
- @normanmaurer
- github.com/normanmaurer

> *Because writing fast, non-blocking network code is non-trivial.*
>
> — Why Netty ?

When I say fast, I mean fast

Best (bar chart) | Data table | Latency | Framework overhead

Best plaintext responses per second, i7-2600K hardware (67 tests)

Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	IA	Errors
grizzly	631,280	100.0%	Mcr	Jav	Svt	Grz	Lin	Rea	0
netty	624,786	99.0%	Plt	Jav	Nty	Non	Lin	Rea	0
undertow	612,146	97.0%	Plt	Jav	Utw	Non	Lin	Rea	197
undertow edge	610,948	96.8%	Plt	Jav	Und	Non	Lin	Rea	50

```
http://www.techempower.com/benchmarks/#section=data-r9&hw=i7&test=plaintext
```

Response

```
HTTP/1.1 200 OK
Content-Length: 15
Content-Type: text/plain; charset=UTF-8
Server: Example
Date: Wed, 17 Apr 2013 12:00:00 GMT

Hello, World!
```

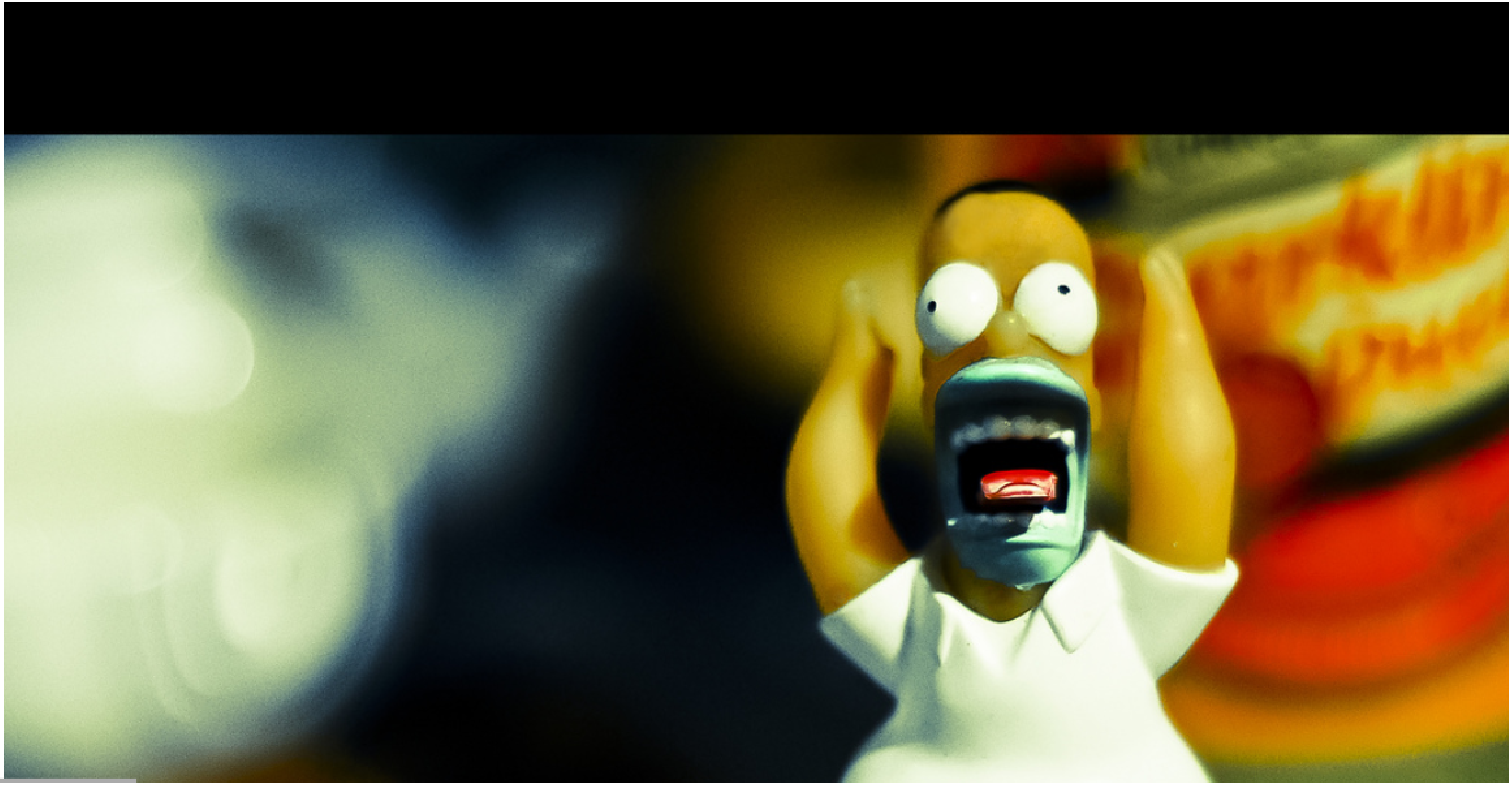
Even more speed in upcoming release

- 256 concurrent connections
- 256 requests pipelined

`24 cores`

```
[nmaurer@xxx]~% wrk/wrk -H 'Host: localhost' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' -H 'Connection: keep-
alive' -d 120 -c 256 -t 16 --pipeline 256  http://xxx:8080/plaintext
Running 2m test @ http://xxx:8080/plaintext
  16 threads and 256 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    18.77ms   16.68ms 452.00ms   92.67%
    Req/Sec   225.82k    41.95k  376.21k    67.34%
  429966998 requests in 2.00m, 57.66GB read
Requests/sec: 3583411.40
Transfer/sec:    492.11MB
```

But….. WHY?



Fully asynchronous

- Asychronous from the ground up
- Using java.nio or native method calls for non-blocking io
- Futures and callbacks provided for easy composing

> *Don’t call us, we’ll call you.*
>
> – Hollywood principle

Hide complexity but not flexibility

- Hides all the complexity involved when you use `java.nio` or `java.nio2`
- Still powers you with a lot of flexibility
- Unified API for every transport
- Allows easy testing of your custom code.

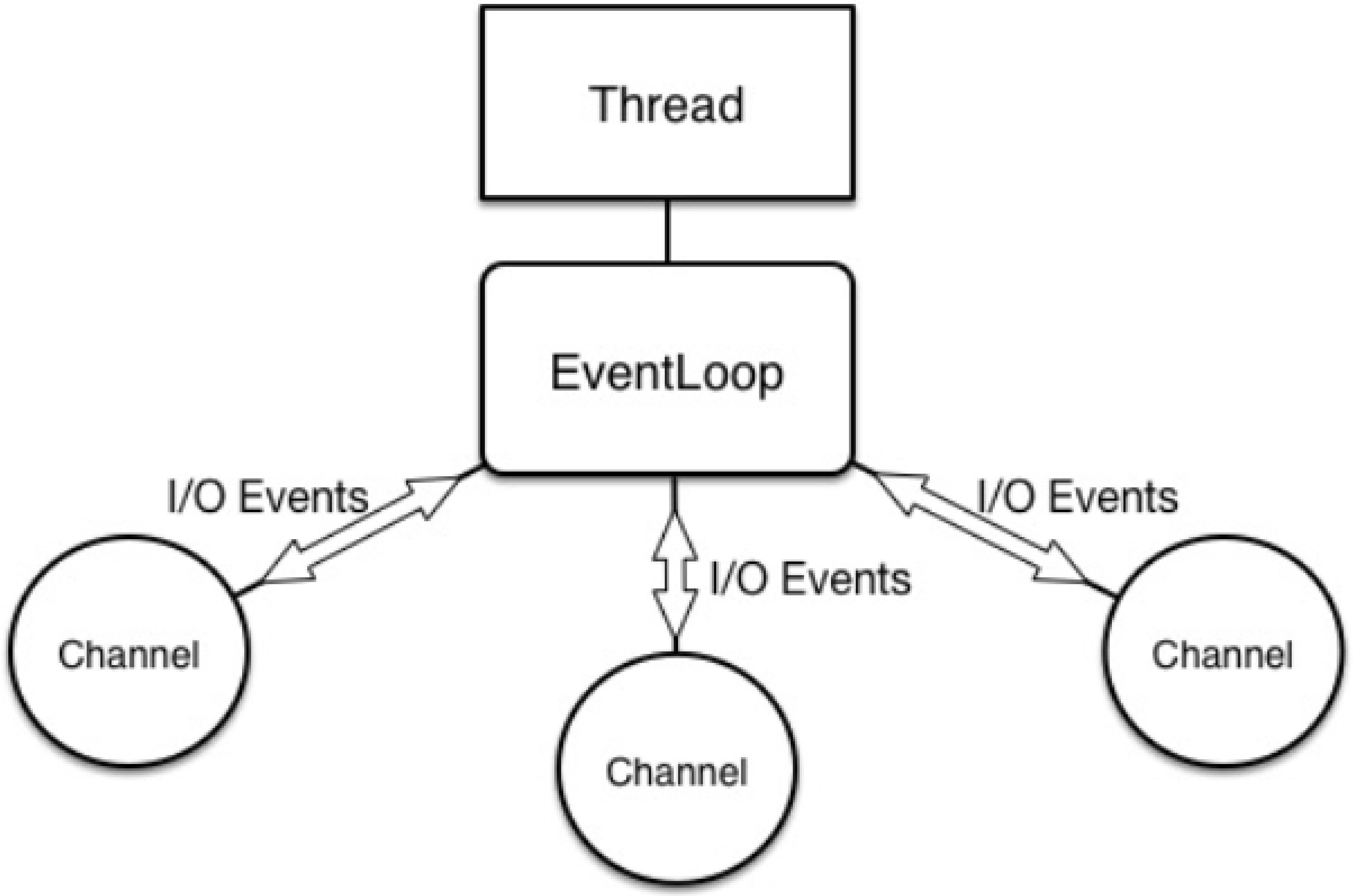
Protocol agnostic → Not just another HTTP server

- Supports `TCP`, `UDP`, `UDT`, `SCTP` out of the box
- Contains codecs for different protocols on top of these

Supported Codecs

- HTTP, WebSockets (+ compression) , SPDY, HTTP 2
- SSL/TLS, Zlib, Deflate
- Protobufs, JBoss Marshalling, Java Serialization
- DNS
- Memcached, Stomp, Proxy, MQTT
- …add your prefered protocol here…

Thread-Model - Easy but powerful

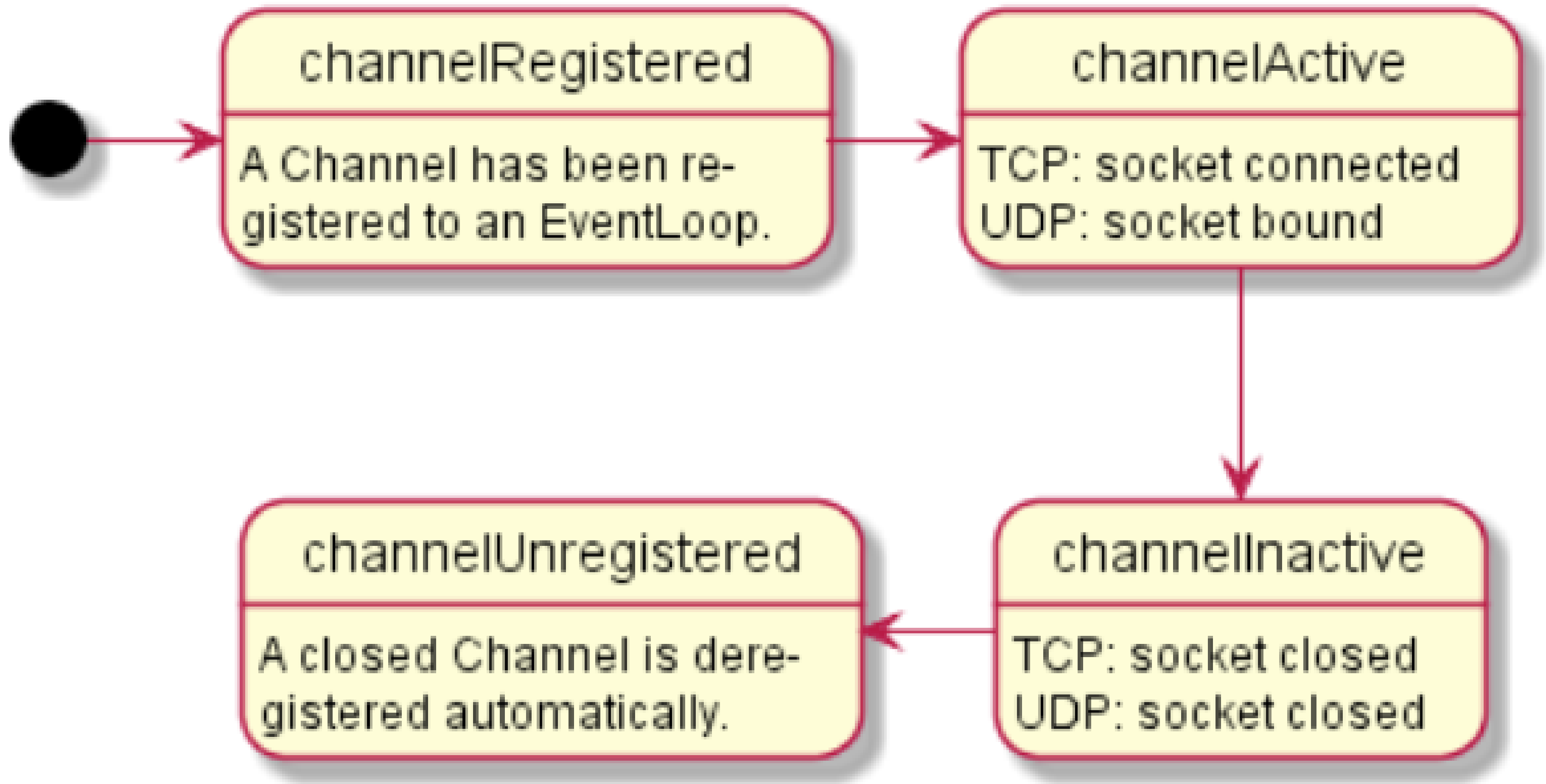


- Having inbound and outbound events handled by the same Thread simplifies concurrency handling a lot!

Simple state model

Allows to react on each state change by intercept the states via `ChannelHandler`.

Allows flexible handling depending on the needs.



ChannelPipeline

- Interceptor pattern

- Allows to add building-blocks (`ChannelHandler`) on-the-fly that transform data or react on events

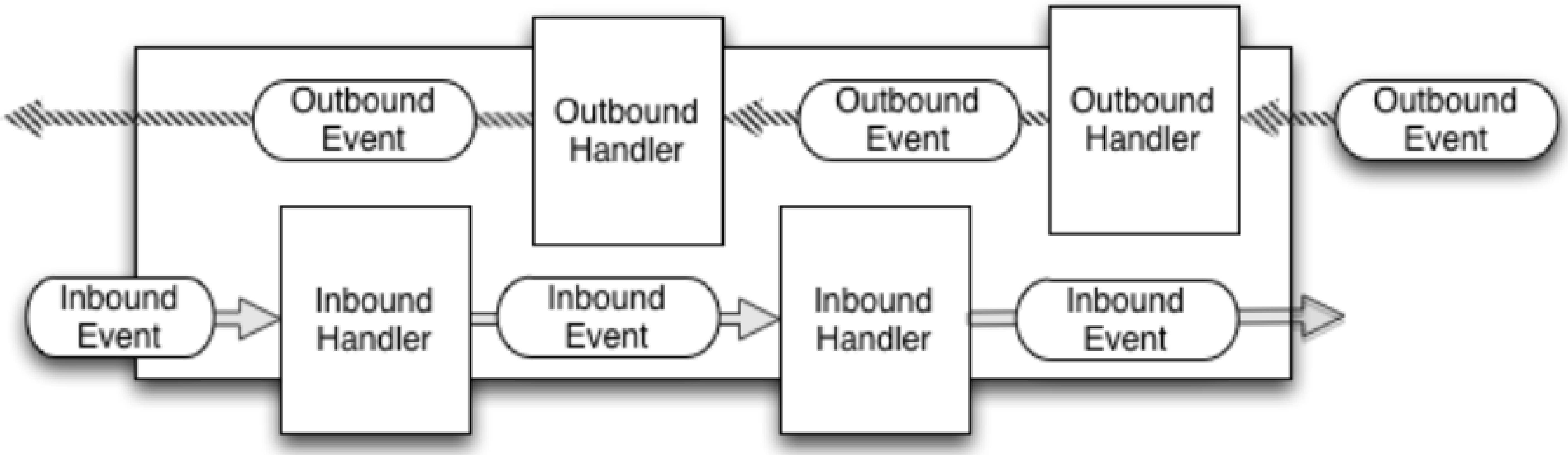
`Kind of a unix-pipe-like thing…`

```
$ echo "Netty is shit...." | sed -e 's/is /is the /' | cat (1)
                   Netty is the shit....
```

(1) Think of the whole line to be the `ChannelPipeline` and `echo`, `sed` and `cat` the `ChannelHandler` s that allow to transform data.

ChannelPipeline - How does it work

Inbound and outbound events flow through the `ChannelHandler` s in the `ChannelPipeline` and so allow to react one these events.



ChannelPipeline - Compose processing logic

Compose complex processing logic via multiple `ChannelHandler`.

```
public class MyChannelInitializer extends ChannelInitializer<Channel> {
  @Override
  public void initChannel(Channel ch) {
    ChannelPipeline p = ch.pipeline();
    p.addLast(new SslHandler(...)); ❶
    p.addLast(new HttpServerCodec(...)); ❷
    p.addLast(new YourRequestHandler()); ❸
  }
}
```

❶ Encrypt traffic

❷ Support HTTP

❸ Your handler that receive the HTTP requests.

ChannelHandler - React on received data

```
@Sharable
public class EchoHandler extends ChannelInboundHandlerAdapter {
  @Override
  public void channelRead(ChannelHandlerContext ctx, Object msg) { ❶
    ctx.writeAndFlush(msg);
  }

  @Override
  public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) { ❷
    cause.printStacktrace();
    ctx.close();
  }
}
```

❶ Intercept received message and write it back to the remote peer

❷ React on `Throwable` and close the connection

Decoder / Encoder - Transform data via ChannelHandler

- Different abstract base classes for Decoder and Encoder
- Handles buffering for you if needed (remember everything is non-blocking!)

Decoder / Encoder - Transform data via ChannelHandler

Transform received ByteBuf to String

```java
public class StringDecoder extends MessageToMessageDecoder<ByteBuf> {
  @Override
  protected void decode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out) {
      out.add(msg.toString(charset));
  }
}
```

Transform to be send String to ByteBuf

```java
public class StringEncoder extends MessageToMessageEncoder<String> {
    @Override
    protected void encode(ChannelHandlerContext ctx, String msg, List<Object> out) {
        if (msg.length() == 0) return;
        out.add(ByteBufUtil.encodeString(ctx.alloc(), CharBuffer.wrap(msg), charset));
    }
}
```

Adding other processing logic?

- Adding more processing logic is often a matter of adding just-another `ChannelHandler` to the `ChannelPipeline`.

Writing protocol multiplexers is a no-brainer!

```
public class PortUnificationServerHandler extends ByteToMessageDecoder {
  @Override
  protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) {
    if (in.readableBytes() < 5) return ❶
    if (isSsl(in)) { ❷
      ctx.pipeline().addLast("ssl", sslCtx.newHandler(ctx.alloc()));
      ctx.pipeline().remove(this);
    } else if (isGzip(in)) { ...
    } else { ❸
      ctx.close();
    }
  }
}
```

❶ Will use the first five bytes to detect a protocol.

❷ Check if SSL is used and if so add `SslHandler` to the `ChannelPipeline`.

❸ Unknown protocol, just close the connection

Flexible write behaviour

- `Channel.write(...)` ⇒ write through the `ChannelPipeline` but NOT trigger syscall like `write` or `writev`.

- `Channel.flush()` ⇒ writes all pending data to the socket.

- Gives more flexibility for when things are written and also allows efficient pipelining.

Easy HTTP Pipelining to safe syscalls by using write(…) and flush()!

```
public class HttpPipeliningHandler extends SimpleChannelInboundHandler<HttpRequest> {
  @Override
  public void channelRead(ChannelHandlerContext ctx, HttpRequest req) {
    ChannelFuture future = ctx.writeAnd(createResponse(req)); ❶
    if (!HttpHeaders.isKeepAlive(req)) {
      future.addListener(ChannelFutureListener.CLOSE); ❷
    }
  }
  @Override
  public void channelReadComplete(ChannelHandlerContext ctx) {
    ctx.flush(); ❸
  }
}
```

❶ Write to the `Channel` (No syscall!) but not flush yet

❷ After written close socket

❸ Flush out to the socket once everything was ready from the `Channel

Allow to detect slow remote peers

https://www.flickr.com/photos/kwl/4514986410

```
public class StateHandler extends ChannelInboundHandlerAdapter {
  @Override
  public void channelWritabilityChanged(ChannelHandlerContext ctx) { } ①
}
```

① Is triggered once `Channel.isWritable()` changes.

Execute ChannelHandler outside of EventLoop

Don’t block as these will effect all `Channel` s that are served by the same `Thread`.

ChannelPipeline allows to add `ChannelHandler` that are exected on different Thread to free up IO Thread (`EventLoop`).

```
Channel ch = ...;
ChannelPipeline p = ch.pipeline();
EventExecutor e1 = new DefaultEventExecutor(16);

p.addLast(new MyProtocolCodec()); (1)
p.addLast(e1, new MyDatabaseAccessingHandler()); (2)
```

1. Executed in `EventLoop` (and so the `Thread` bound to it)
2. Executed in one of the `EventExecutors` of e1

Built with GC pressure in mind

- Use pooling / `ThreadLocal` s to prevent GC pressure
- Prefer direct method invocation over fire event object

`http://25.media.tumblr.com/tumblr_me2eq0PnBx1rtu0cpo1_1280.jpg`

- Reduces GC-Pressure a lot!

ByteBuf - ByteBuffer on steroids!

- Separate index for reader/writer
- Direct, Heap and Composite
- Resizable with max capacity
- Reference counting / pooling
- uses `sun.misc.Unsafe` for maximal performance

```
ByteBuf buf = ...;
buf.writeInt(1).writeBytes(data).writeBoolean(true)...
```

Allow parsing without expensive range-checks

SlowSearch for ByteBuf :(

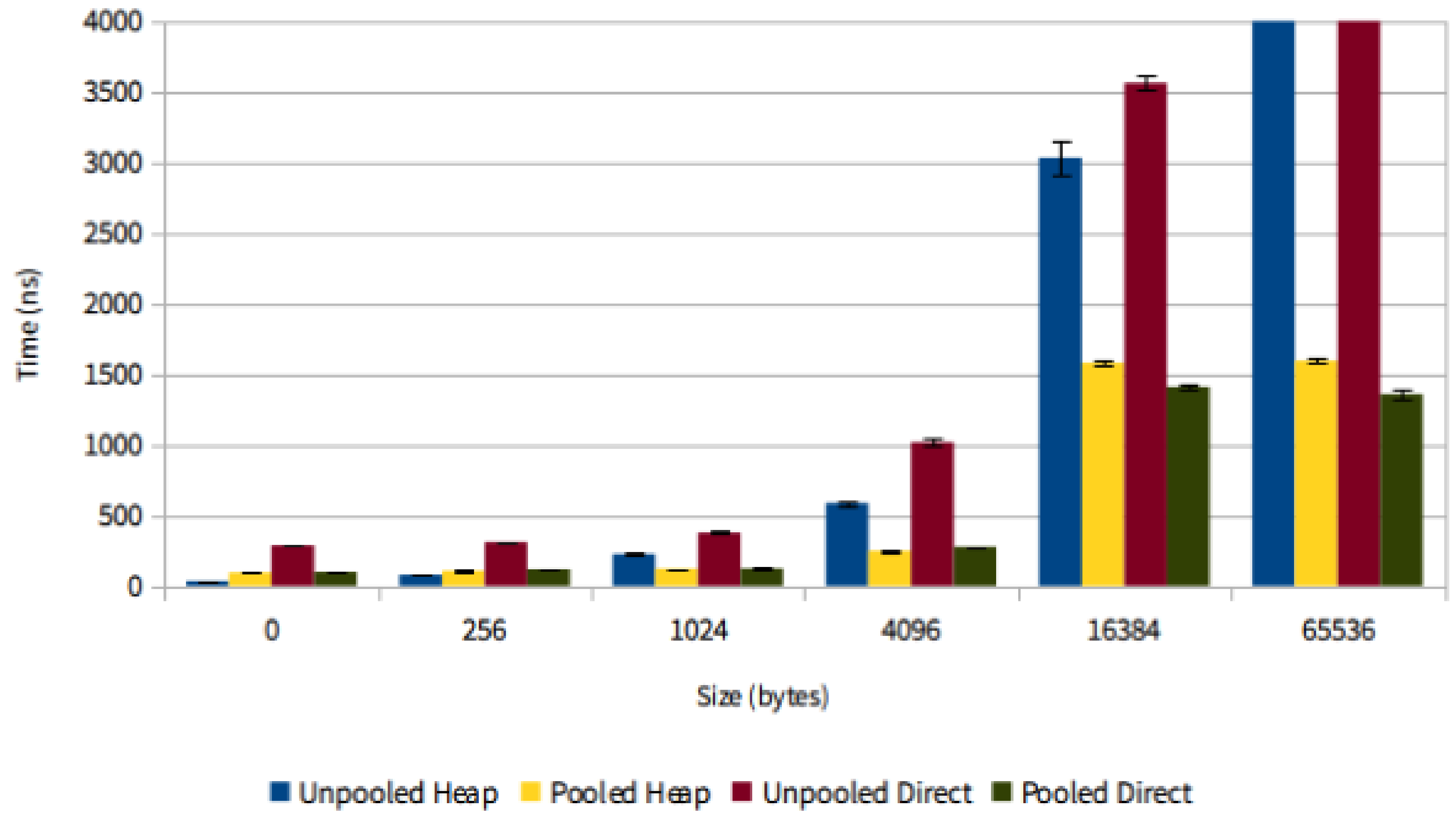
```
int index = -1;
for (int i = buf.readerIndex(); index == -1 && i <  buf.writerIndex(); i++) {
  if (buf.getByte(i) == '\n') {
    index = i;
  }
}
```

FastSearch for ByteBuf :)

```
int index = buf.forEachByte(new ByteBufProcessor() {
  @Override
  public boolean process(byte value) {
    return value != '\n';
  }
});
```

Use Pooling of buffers to reduce allocation / deallocation time!

- Pooling pays off for direct and heap buffers!



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

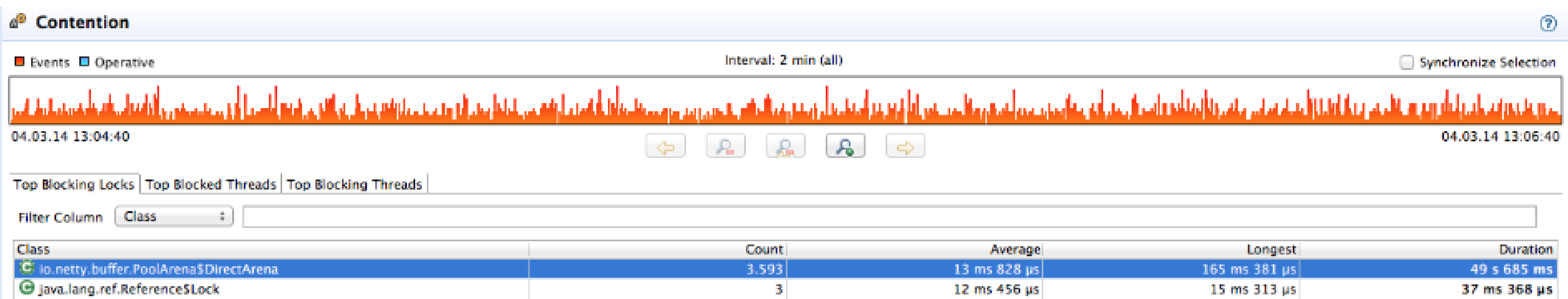
PooledByteBufAllocator - Algorithms / datastructures

- Algorithm is a hybrid of jemalloc and buddy-allocation
- `ThreadLocal` caches for lock-free allocation
- Synchronization per area that holds different chunks of memory, when not be able to serve via cache

PooledByteBufAllocator - without caches

Before caches were added

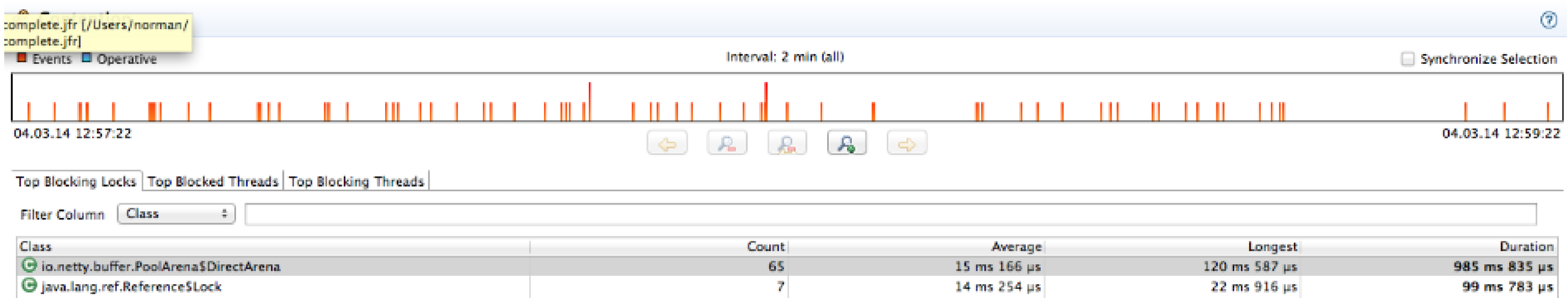
```
[nmaurer@xxx]~% wrk/wrk  -H 'Host: localhost' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
     -H 'Connection: keep-alive' -d 120 -c 256 -t 16 --pipeline 256
http://xxx:8080/plaintext
...
Requests/sec: 2812559.99
Transfer/sec:    388.93MB
```



PooledByteBufAllocator - caches to the rescue!

`With caches`

```
[nmaurer@xxx]~% wrk/wrk  -H 'Host: localhost' -H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
     -H 'Connection: keep-alive' -d 120 -c 256 -t 16 --pipeline 256
http://xxx:8080/plaintext
...
Requests/sec: 3022942.17
Transfer/sec:    418.02MB
```



Leak detection built-in

`simple leak reporting`

```
LEAK: ByteBuf.release() was not called before it's garbage-collected....
```

`advanced leak reporting`

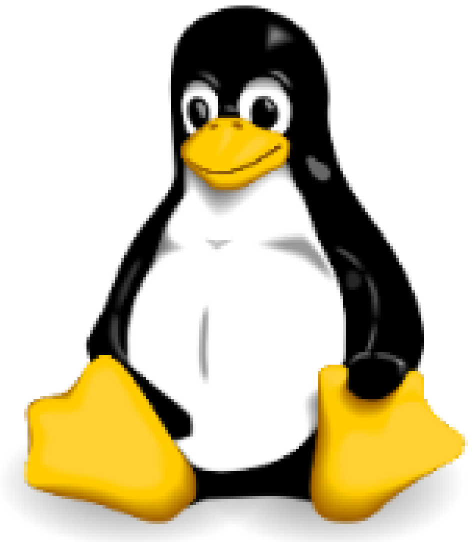
```
LEAK: ByteBuf.release() was not called before it's garbage-collected.
Recent access records: 1
#1:

io.netty.buffer.AdvancedLeakAwareByteBuf.toString(AdvancedLeakAwareByteBuf.java:697)
    ...
Created at:
    ...

io.netty.handler.codec.xml.XmlFrameDecoderTest.testDecodeWithXml(XmlFrameDecoderTest.ja
```

Native stuff in Netty 4

- OpenSSL based SslEngine to reduce memory usage and latency.
- Native transport for linux using epoll ET for more performance and less CPU usage.
- Native transport also supports SO_REUSEPORT and TCP_CORK :)



Switching to native transport is easy

`Using NIO transport`

```java
Bootstrap bootstrap = new Bootstrap().group(new NioEventLoopGroup());
bootstrap.channel(NioSocketChannel.class);
```

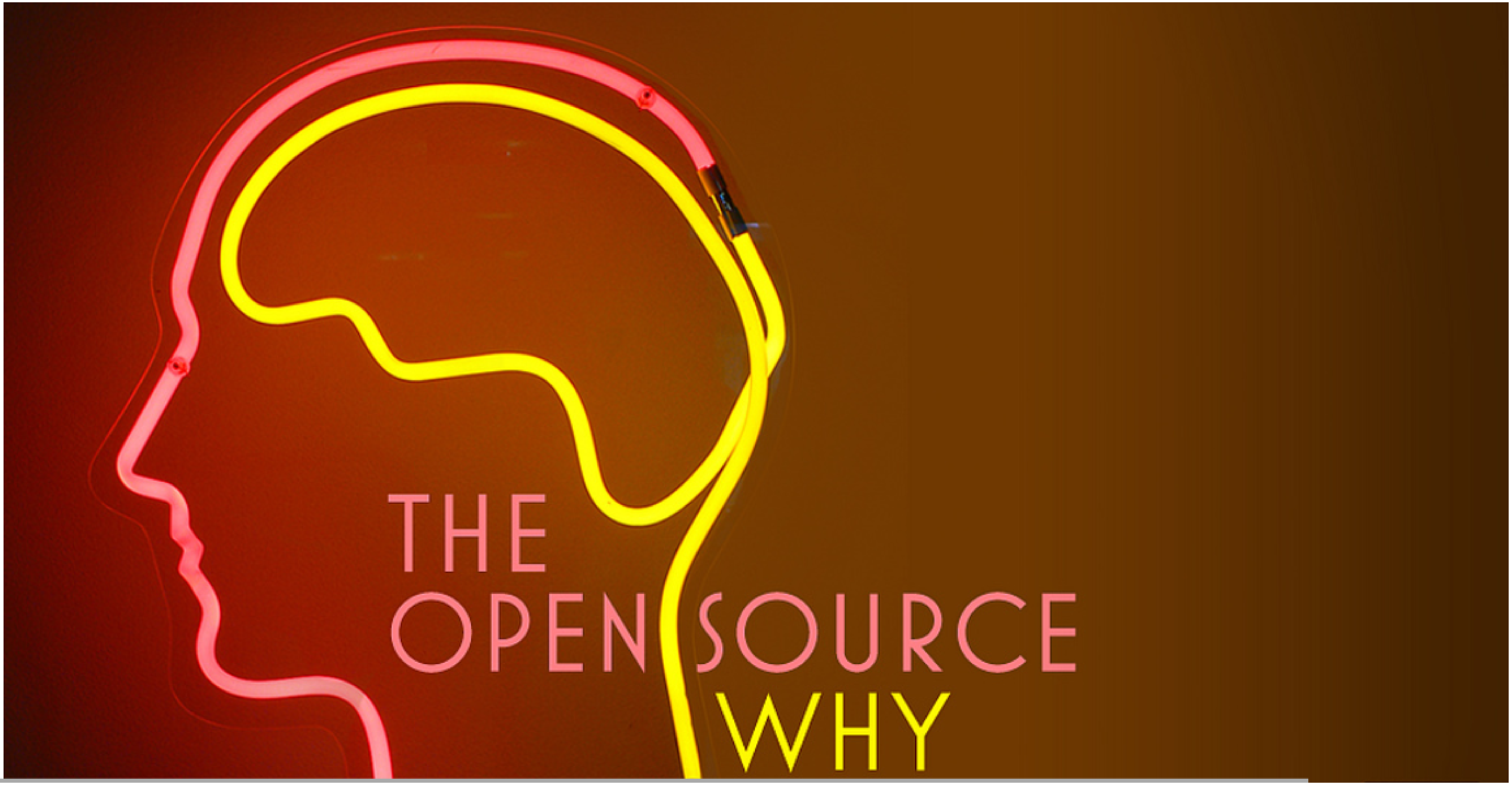
`Using native transport`

```java
Bootstrap bootstrap = new Bootstrap().group(new EpollEventLoopGroup());
bootstrap.channel(EpollSocketChannel.class);
```

Things to come

- Dynamic `Channel` re-register (based on metrics) - GSOC
- `ForkJoinPool` based `EventLoop` - GSOC
- Metrics

https://www.flickr.com/photos/mrpinkeyes/5140672916/


THE
OPEN SOURCE
WHY

Vibrant community!

June 08 2014 - **July 08 2014**

Period: **1 month**

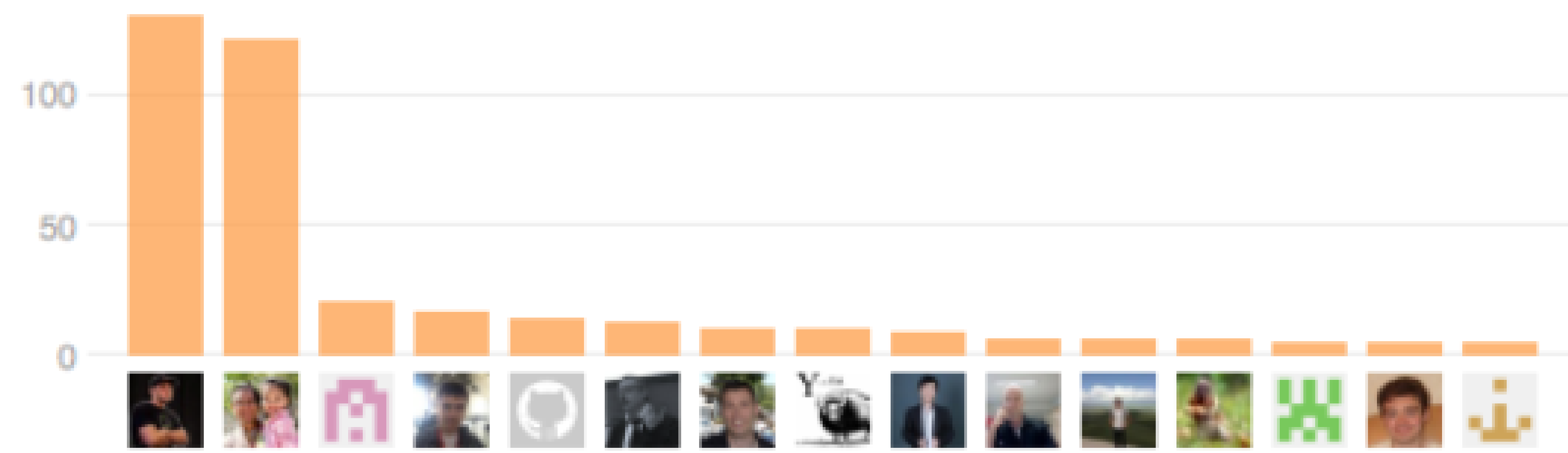
Overview

10 Active Pull Requests

56 Active Issues

0 Merged Pull Requests	**10** Proposed Pull Requests	**42** Closed Issues	**14** New Issues

Excluding merges, **25 authors** have pushed **102 commits** to master and **398 commits** to all branches. On master, **485 files** have changed and there have been **21,210 additions** and **6,740 deletions**.



Companies using Netty!

- … and many more …

(Opensource) Projects using Netty!

- … and many more …

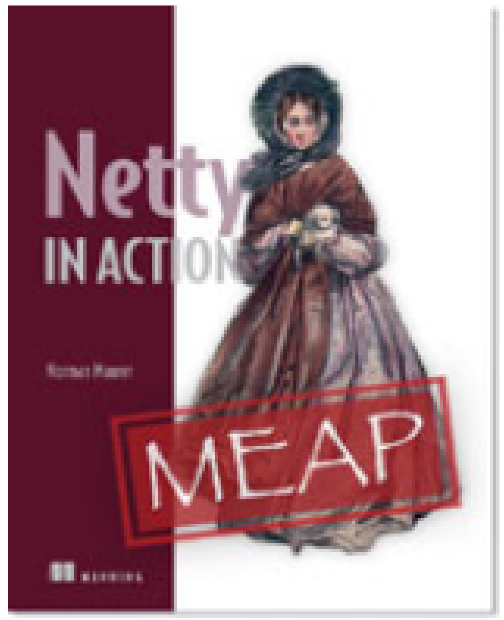
Get Involved - We love contributions

Netty

- Mailinglist - https://groups.google.com/forum/#!forum/netty
- IRC - #netty irc.freenode.org
- Website - http://netty.io
- Source / issue tracker - https://github.com/netty/netty/

Want to know more?

Buy my book Netty in Action and make me RICH.




```
http://www.manning.com/maurer
```

$ KA-CHING $

References

- Netty - http://netty.io
- Slides generated with Asciidoctor and DZSlides backend
- Original slide template - Dan Allen & Sarah White
- All pictures licensed with `Creative Commons Attribution` or `Creative Commons Attribution-Share Alike`

Norman Maurer

@normanmaurer