Overview of Grizzly and Grizzly 2.0

Oleksiy Stashok

Jeanfrancois Arcand

Sun Microsystems

Agenda

- What is project Grizzly
- Grizzly modules
- Grizzly 2.0
- RoadMap
- Summary

Footnote position, 12 pts.

1. What is Project Grizzly

- Open Source Project https://grizzly.dev.java.net
- Open Sourced under CDDL/LGPL license.
- Very open community policy.
  - All project communications are done on Grizzly mailing list. No internal, off mailing list conversations a-la-Sailfin.
  - Project meetings open to anyone
- Project decisions are made by project member votes. Code contributions frequents.
- Approx ~25 000 download for the last 3 months
- Mailing list quite busy some day!

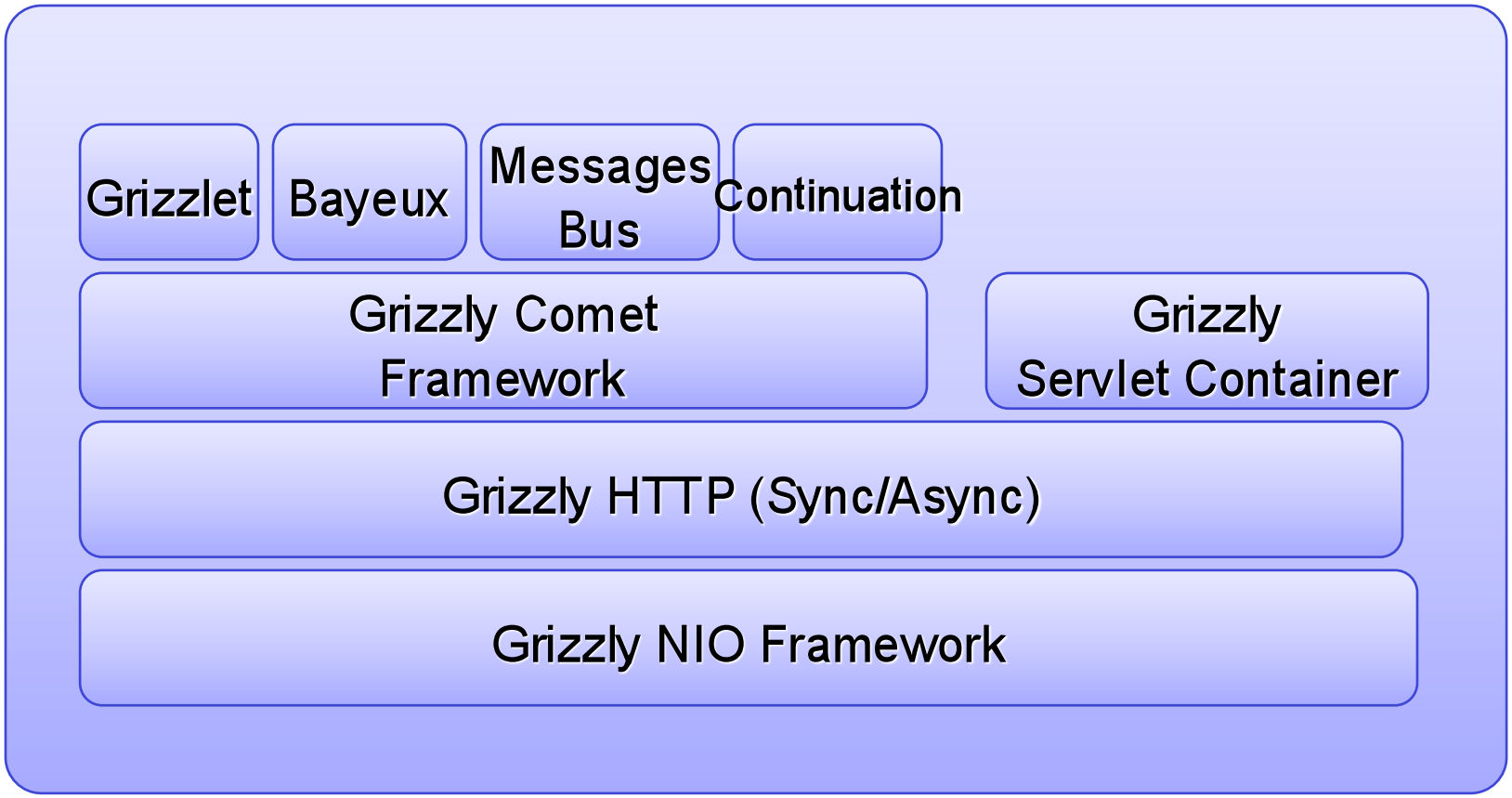
1. What is project Grizzly

- Commiters
  - Oleksiy Stashok (Sun) – Lead *(all modules)*
  - Jean-Francois Arcand (Sun) *(all modules)*
  - Shing Wai Chan (Sun) *(http, comet, bayeux)*
  - Ken Cavanaugh (Sun) *(framework)*
  - Charlie Hunt (Sun) *(framework)*
  - **John Vieten (Conclude GmbH)** *(tutorial, framework)*
  - **Sebastien Dionne (Consultant)** *(tutorial, framework)*
  - **Takai Naoto (ITOCHU Techno-Solution)** *(jruby, bayeux)*
- Active users list…the community answer for us!

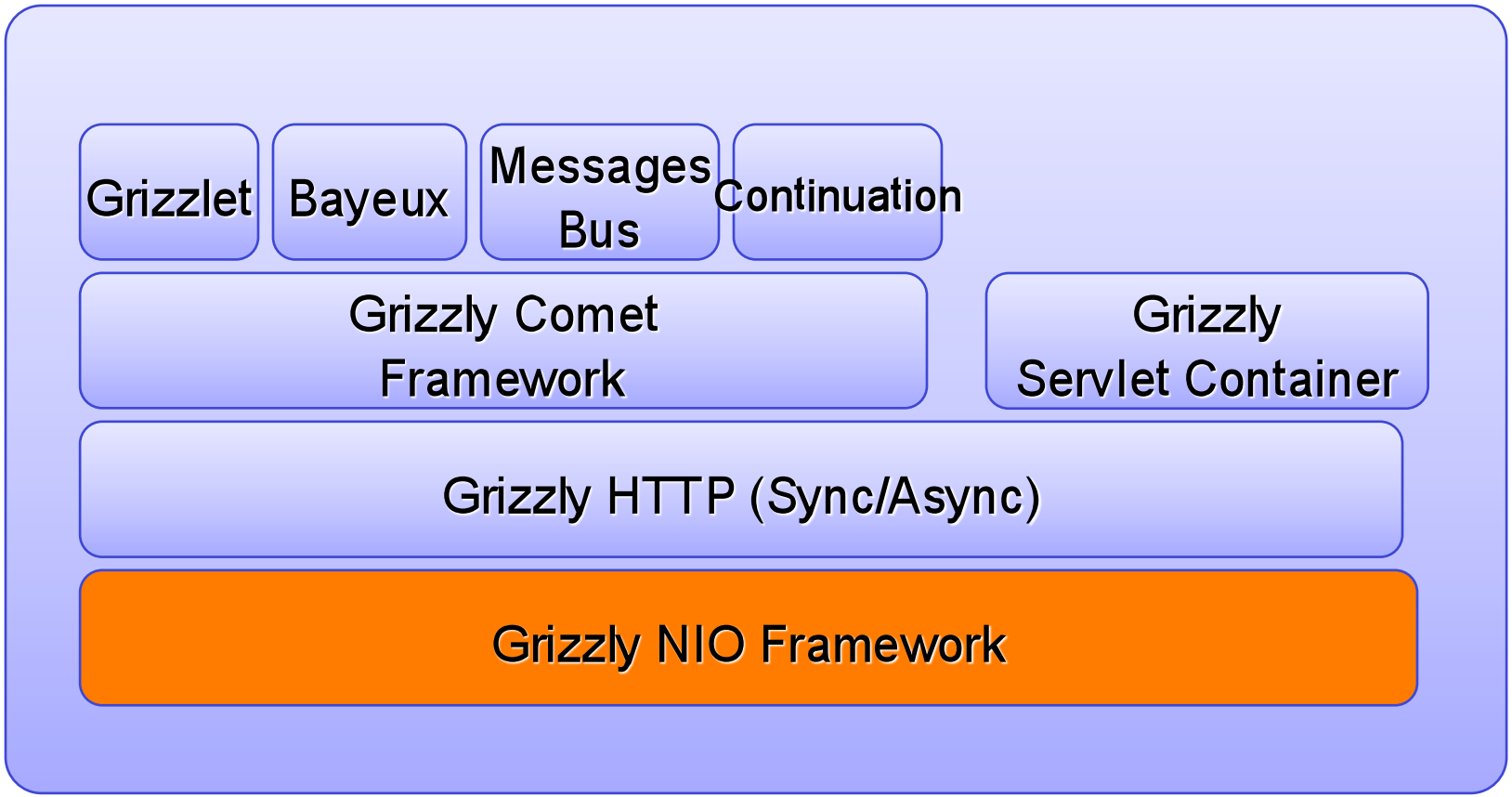
What is Project Grizzly

- Uses Java NIO primitives and hides the complexity programming with Java NIO.
- Started in 2004 (Grizzly 1.0). Own project February 2007 (Grizzly 1.5)
- 1.0 represented extracted network layer of Glassfish
- Easy-to-use high performance APIs for TCP, UDP and SSL communications.
- Utilizes high performance buffers and buffer management.
- Choice of several different high performance thread pools.
- Ship with an HTTP module which is really easy to embed.

2. Grizzly modules



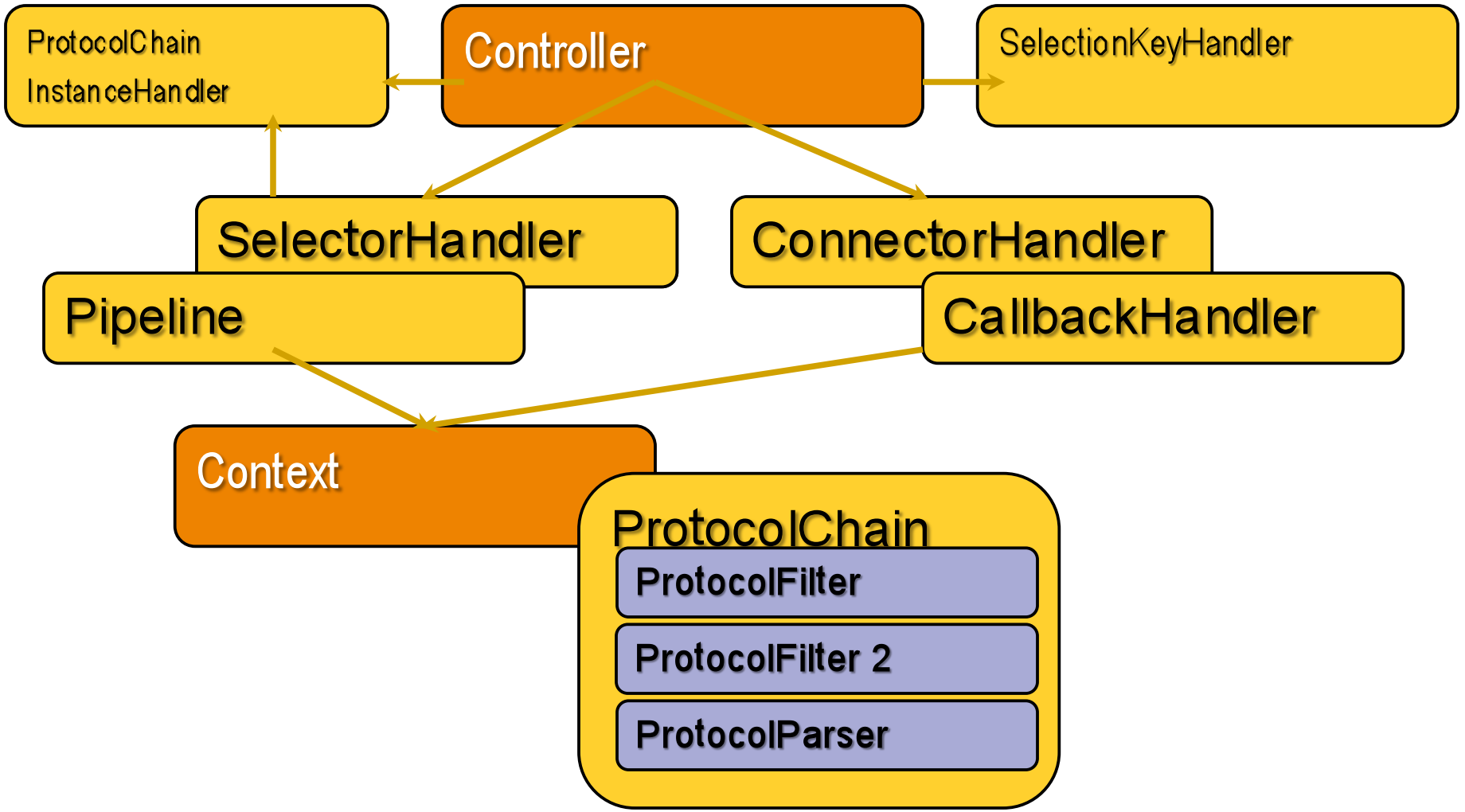
2. Framework


Grizzlet
Bayeux
Messages Bus
Continuation
Grizzly Comet Framework
Grizzly Servlet Container
Grizzly HTTP (Sync/Async)
Grizzly NIO Framework

2.1 Framework

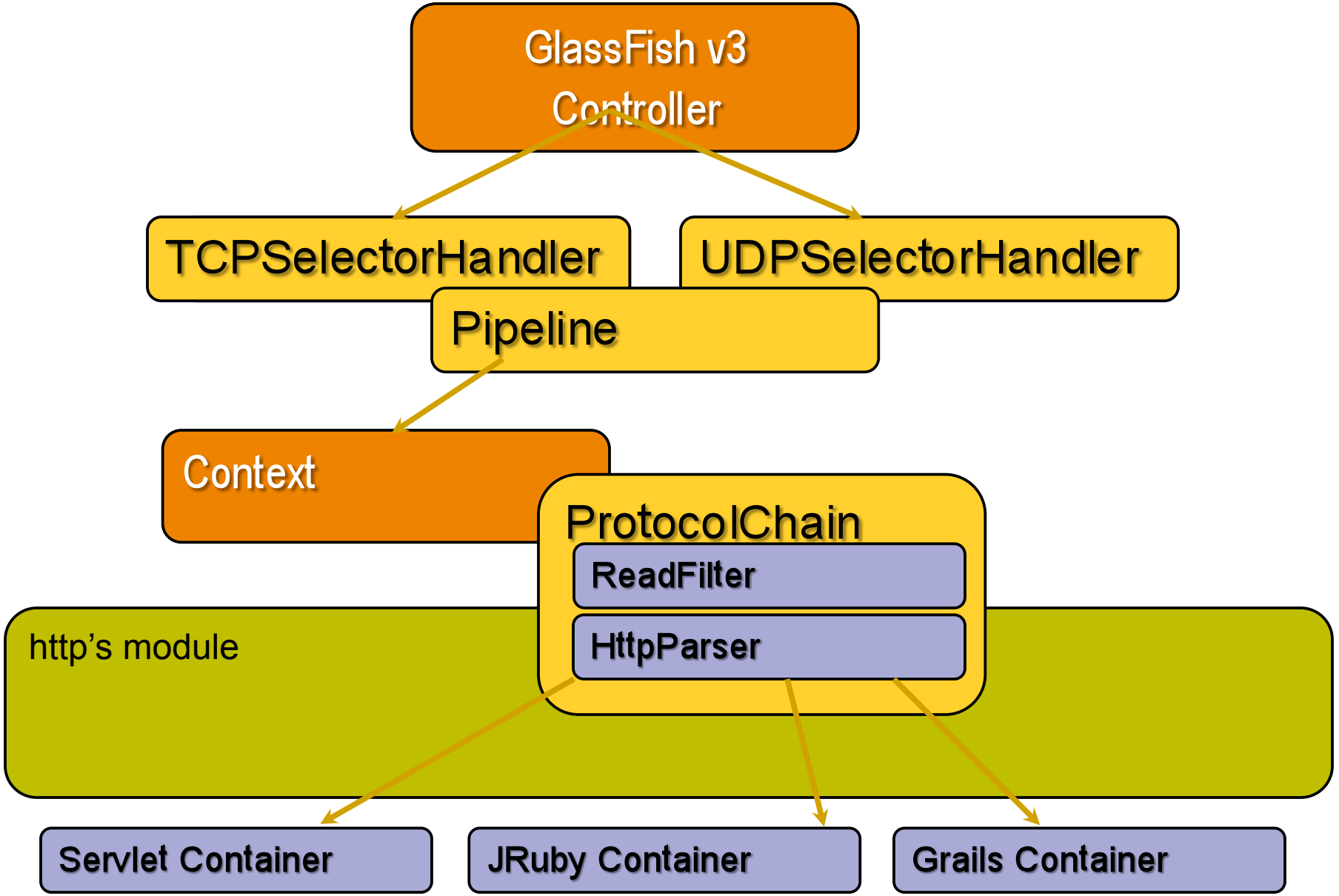
- The root of Grizzly. Everything build on top of this module
- The Grizzly Framework bundles default implementation for TCP, UDP and TLS/SSL transports.
- Used to implement protocol based applications like http, sip, custom protocol, etc.
- In this module, EVERYTHING is customizable. If the default isn't doing what you need , customize it!

Grizzly Framework's Monster face

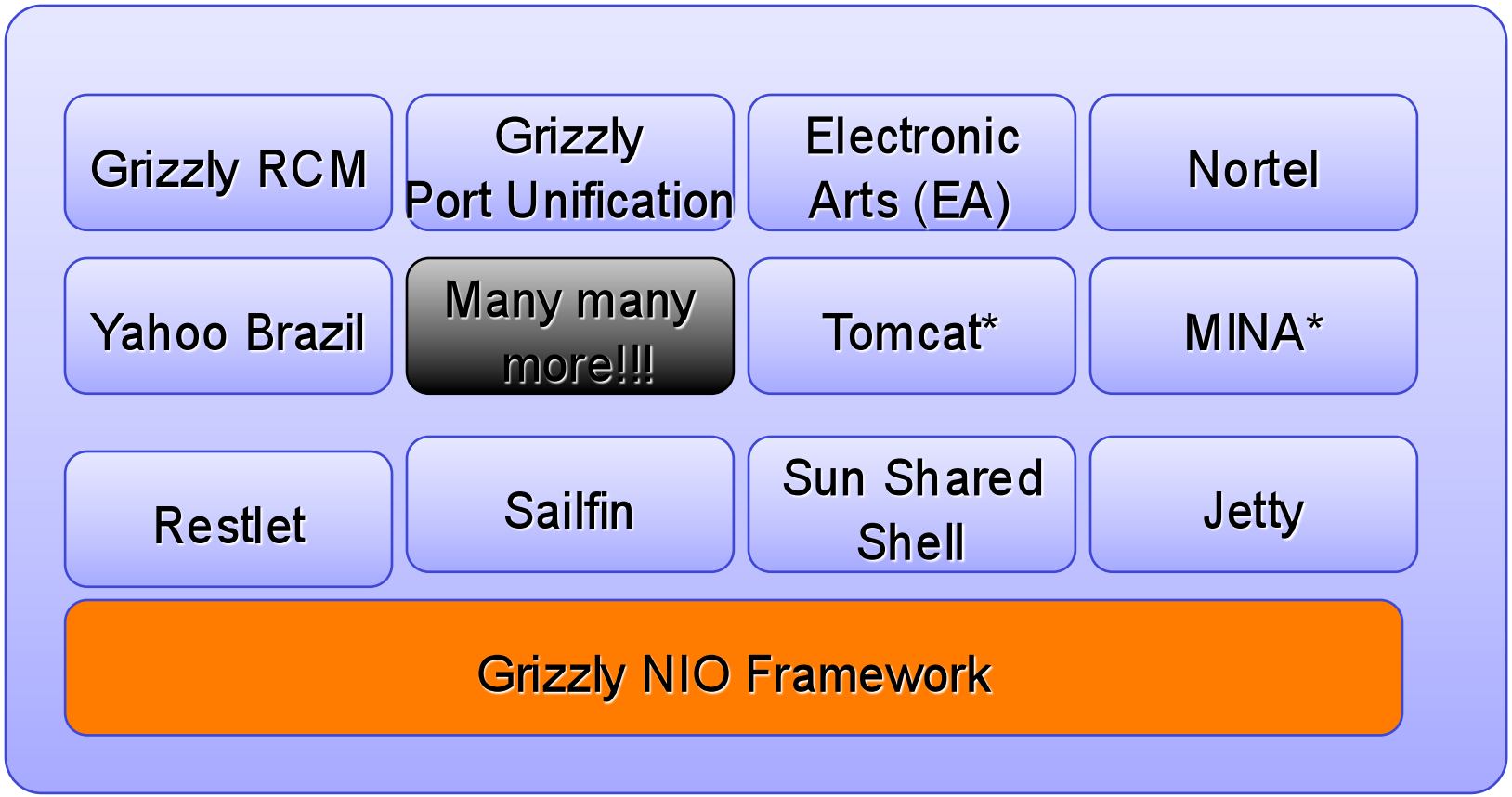


Example: Simple log server over TCP

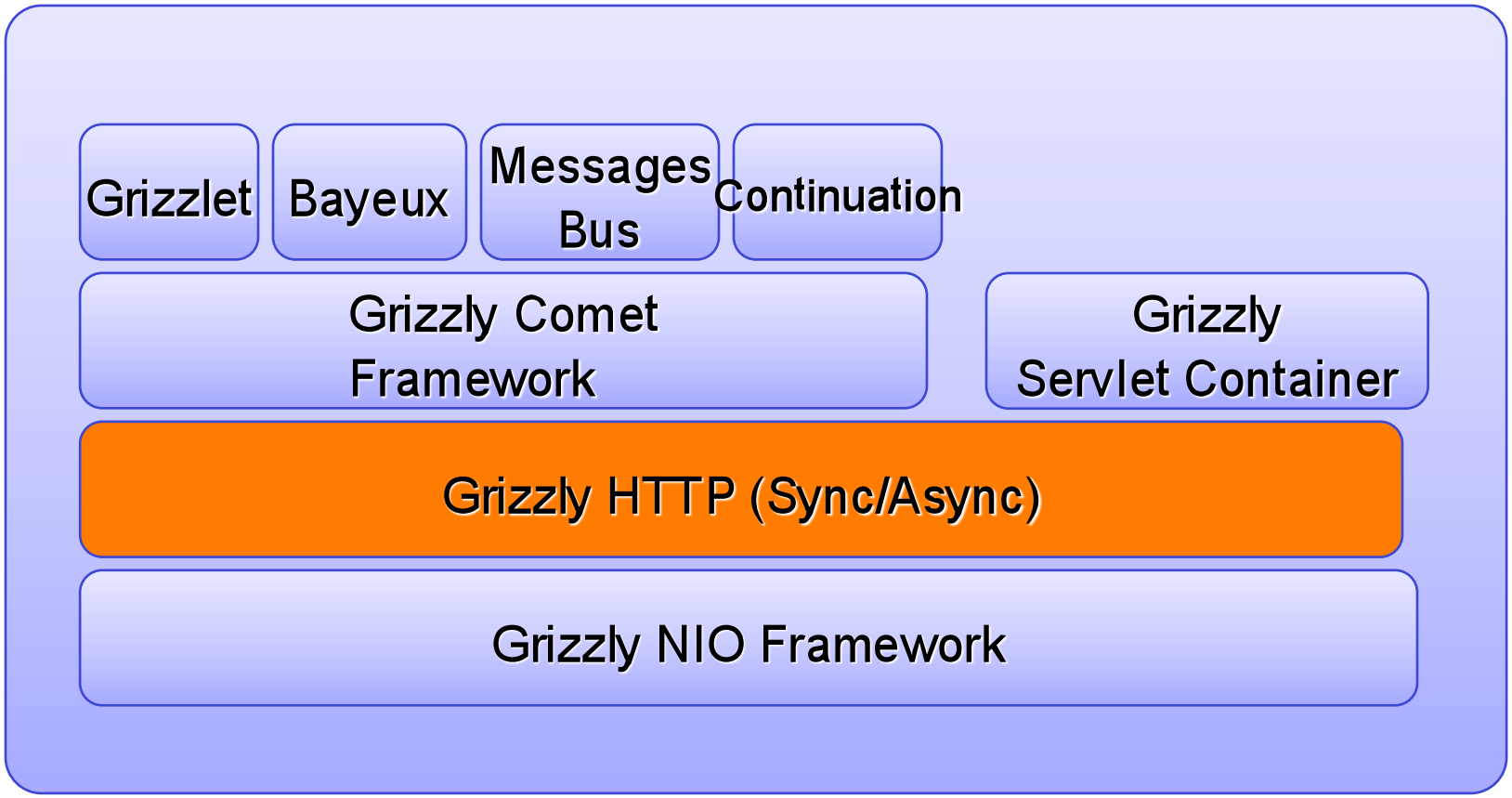
```
Controller controller = new Controller();
controller.setProtocolChainInstanceHandler(new
            DefaultProtocolChainInstanceHandler() {
              ProtocolChain protocolChain;
              // Stateless ProtocolChain
              public ProtocolChain poll() {
                if(protocolChain == null) {
                  protocolChain = new DefaultProtocolChain();
                  protocolChain.addFilter(new ReadFilter());
                  protocolChain.addFilter(new LogFilter());
                }
                return protocolChain;
});
controller.start();
```


GlassFish v3
Controller
TCPSelectorHandler
UDPSelectorHandler
Pipeline
Context
ProtocolChain
ReadFilter
HttpParser
http's module
Servlet Container
JRuby Container
Grails Container

1. Who?


Grizzly RCM
Grizzly Port Unification
Electronic Arts (EA)
Nortel
Yahoo Brazil
Many many more!!!
Tomcat*
MINA*
Restlet
Sailfin
Sun Shared Shell
Jetty
Grizzly NIO Framework

2.2 Http module



HTTP modules (http, http-utils)

- The Grizzly Framework also have an HTTP framework that can be used to build Web Server
- Extremely small, perfect for small devices
  - Like iPhone, Android, Smart Box, etc.
- Simple interface to allow customization of the HTTP Protocol
  - GrizzlyRequest: A utility class to manipulate the HTTP protocol request.
  - GrizzlyResponse: A utility class to manipulate the HTTP protocol response.
  - GrizzlyAdapter: A utility class to manipulate the HTTP request/ response object.

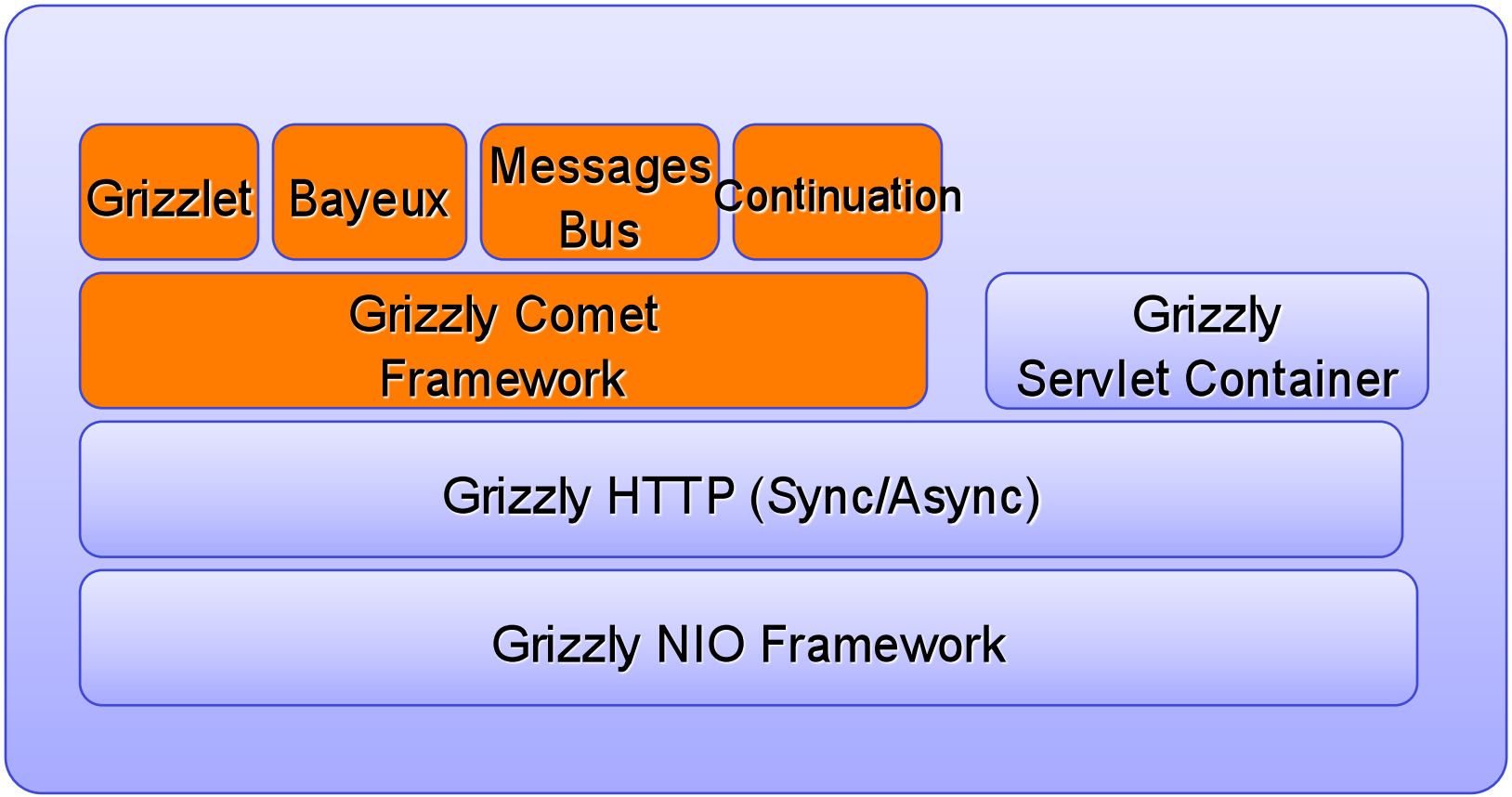
2.2 Http module

```
public class FileAdapter extends GrizzlyAdapter{
  public void service(GrizzlyRequest req,
                      GrizzlyResponse res){
    ByteChun bc = req.decodedURI();
    // Prevent XSS attack
    HttpRequestDecoded.decode(bc);
    // OK, we are safe
    res.write((loadFile(bc));
    res.flush();
    res.close();
```

2.2 Who?

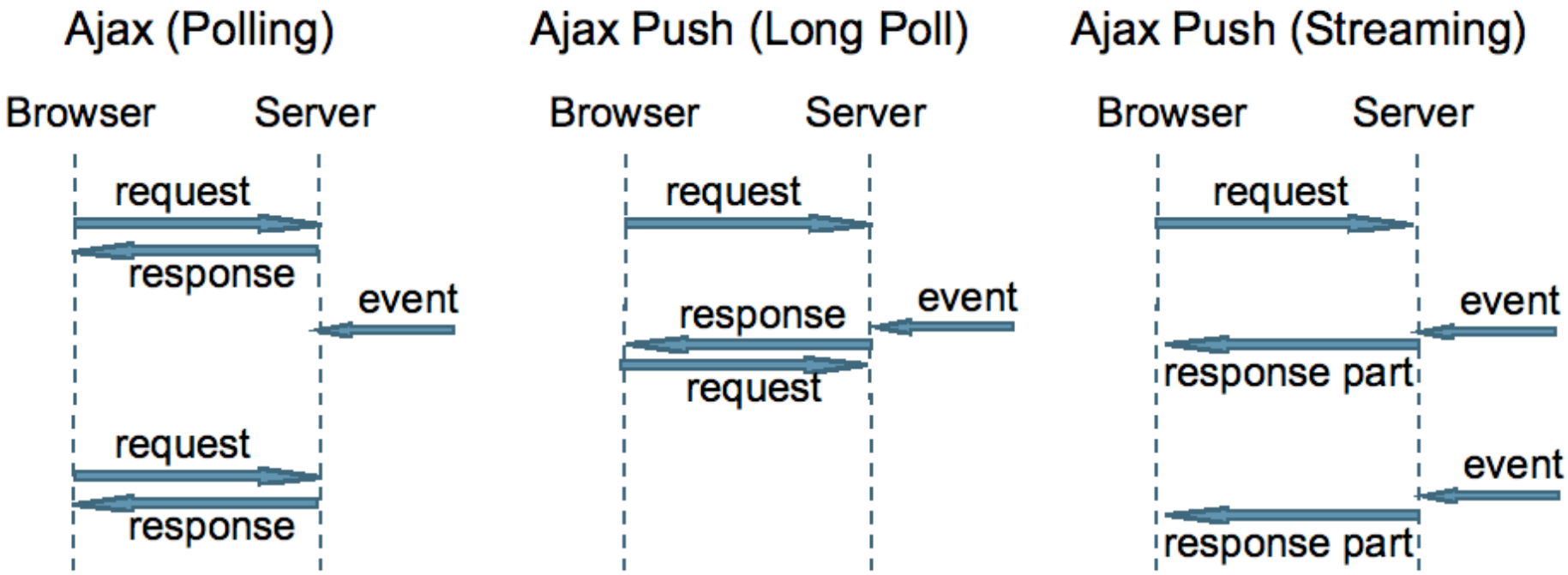

Many many more!!!
GlassFish Gem (JRuby)
Blog-city.com
EventGnosis
Sailfin CBL
GlassFish v3
GlassFish ESB
Futjisu
JXTA
Grizzly HTTP (Sync/Async)
Grizzly NIO Framework

2.3 Grizzly Comet Framework



What is Comet?

Bending the rules of HTTP.

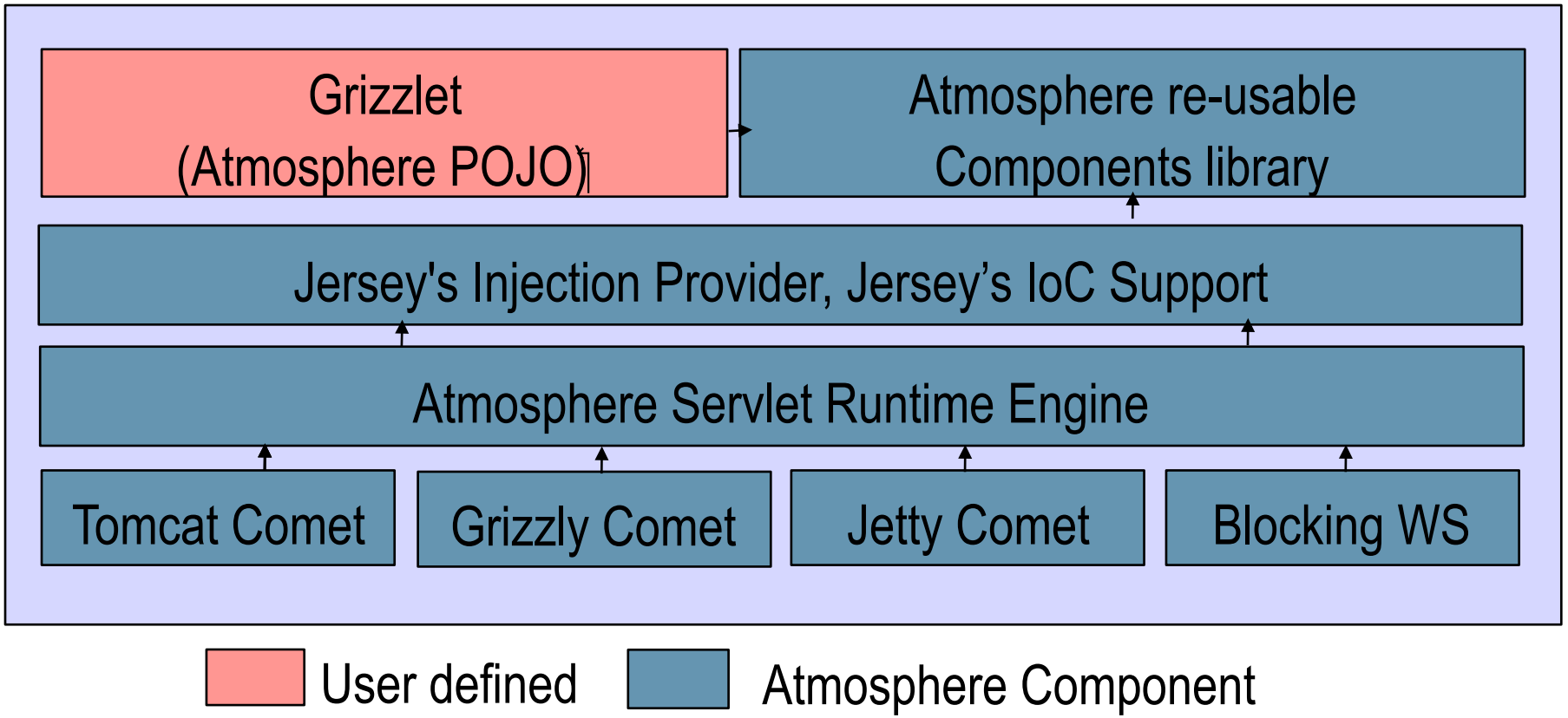


2.4 Grizzly Comet Framework

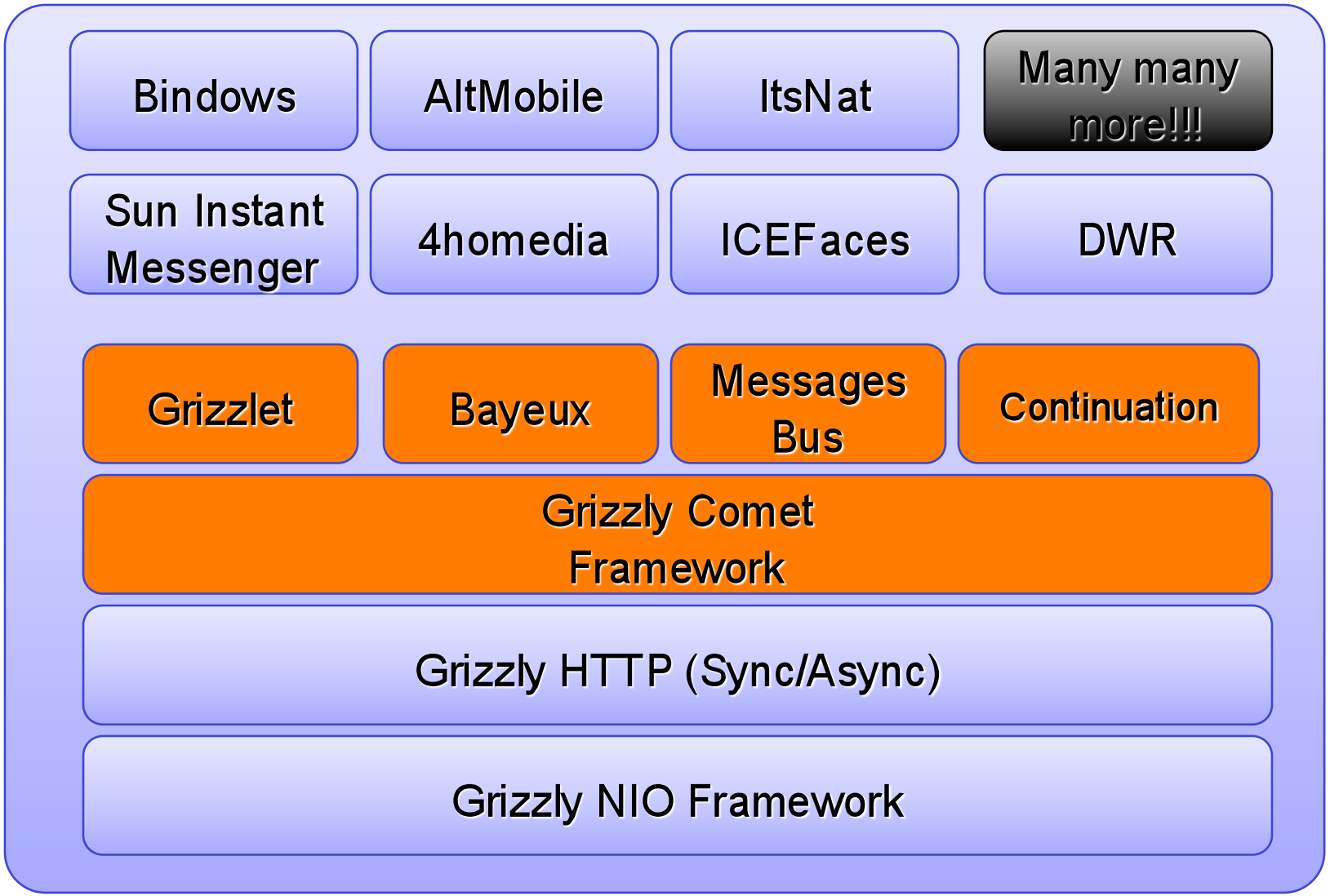
- Grizzly Comet is a framework that ship with GlassFish v1|2|3, and can also be embedded into any application.
  - One day ☺, the code will be moved to a new project called Atmosphere (Atmosphere.dev.java.net)
  - Grizzly Comet, for all web server!!
- The Grizzly Comet Framework includes a set of components that can be used for building Comet based application:
  - Grizzly Comet, Continuation, Grizzlet, Messages Bus, Bayeux support

Atmosphere….

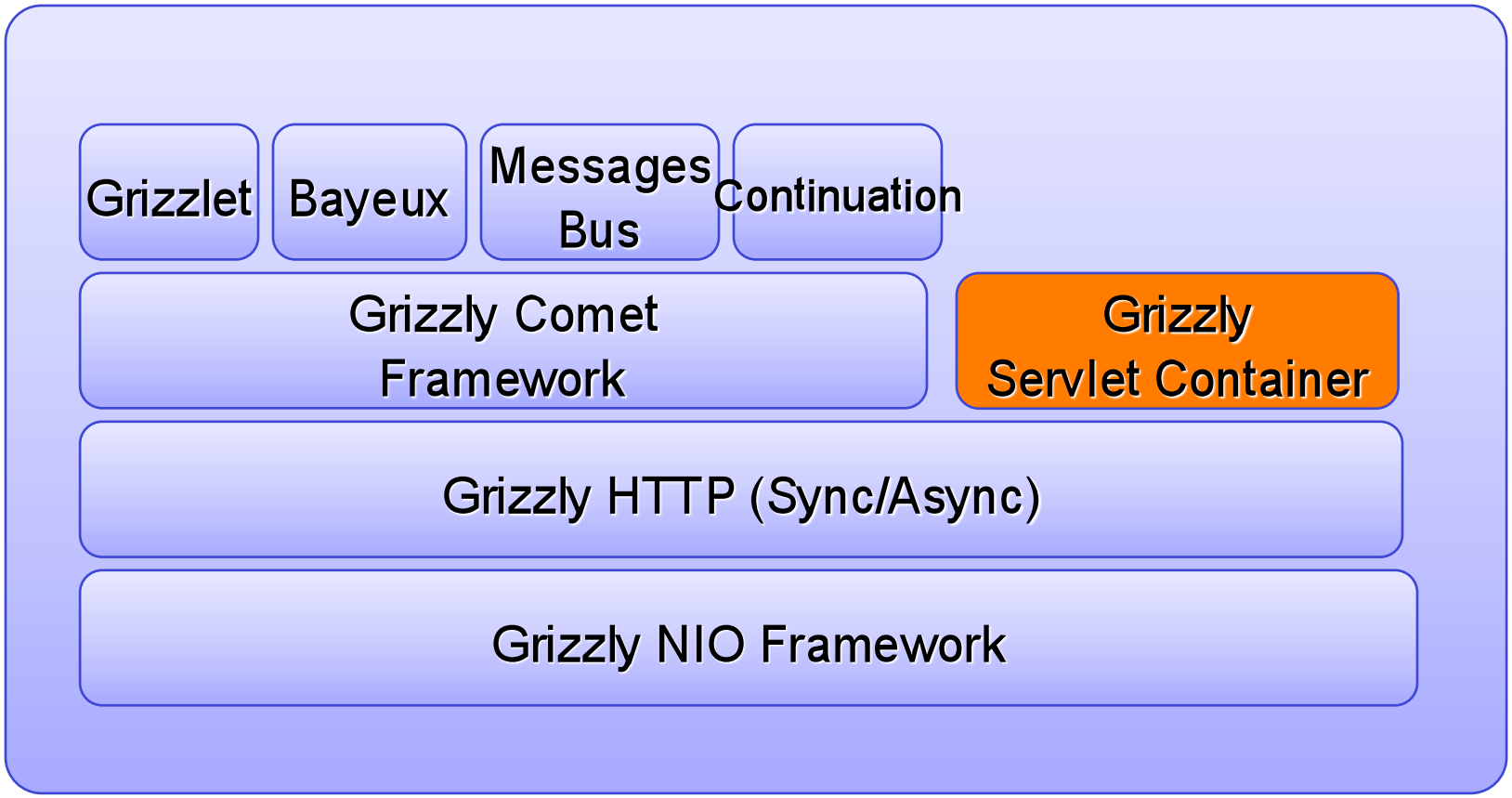
Reuse experience and code...from Grizzly Comet to Jersey!



2.4 Who?


Bindows
AltMobile
ItsNat
Many many more!!!
Sun Instant Messenger
4homedia
ICEFaces
DWR
Grizzlet
Bayeux
Messages Bus
Continuation
Grizzly Comet Framework
Grizzly HTTP (Sync/Async)
Grizzly NIO Framework

2.5 Grizzly Servlet Container

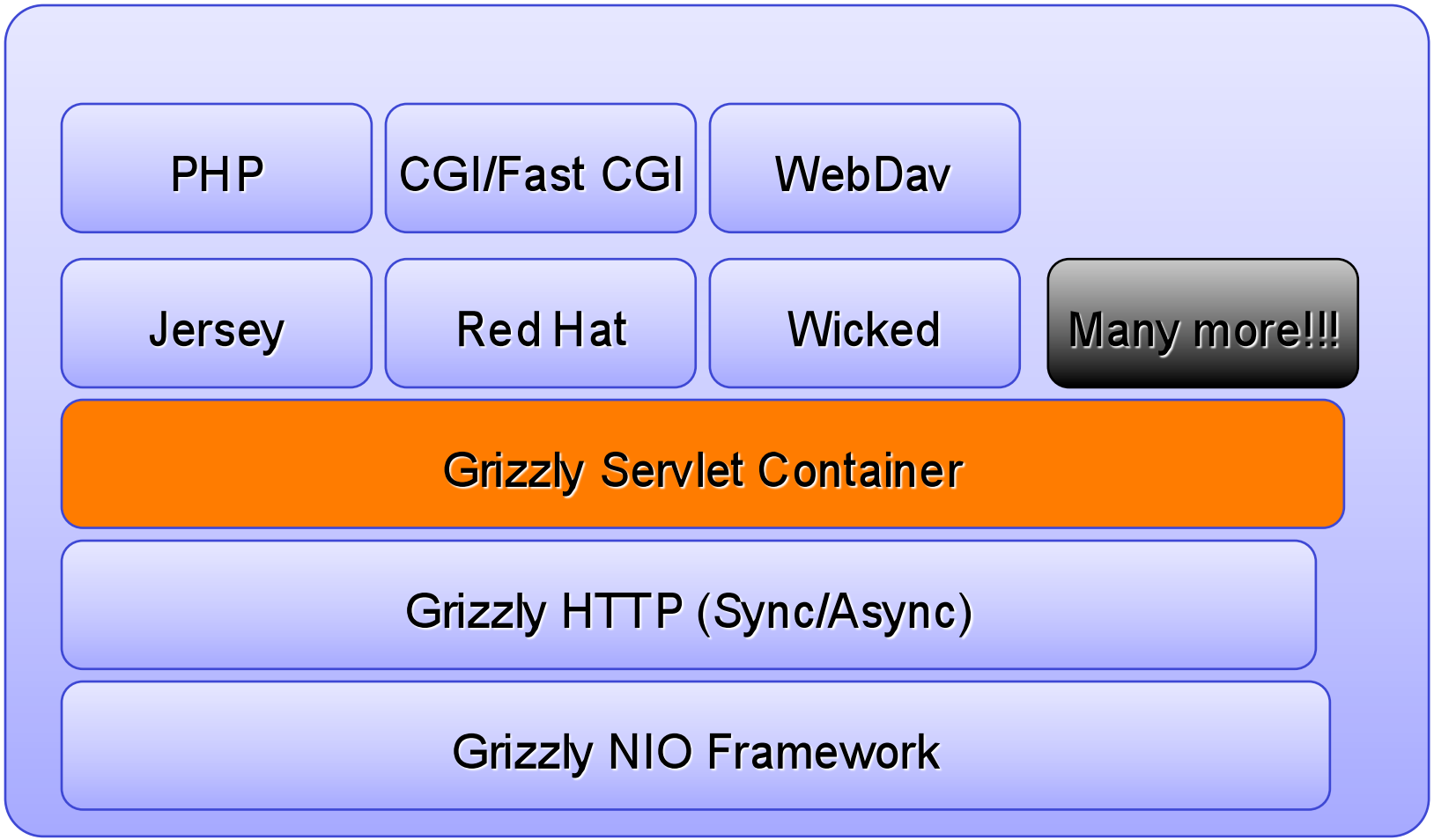


2.5 Grizzly Servlet Container

- The community started developing a Servlet Container
  - Incomplete, not (*yet*) Servlet compliant
  - Very small and fast
  - No extra…just the basic….with power
  - Easy to embed in any application.

- Really useful for writing unit test.

2.5 Who?


PHP
CGI/Fast CGI
WebDav
Jersey
Red Hat
Wicked
Many more!!!
Grizzly Servlet Container
Grizzly HTTP (Sync/Async)
Grizzly NIO Framework

4. Grizzly 2

- Introduction
- Goals
- Architecture
- New features
- Examples
- Roadmap

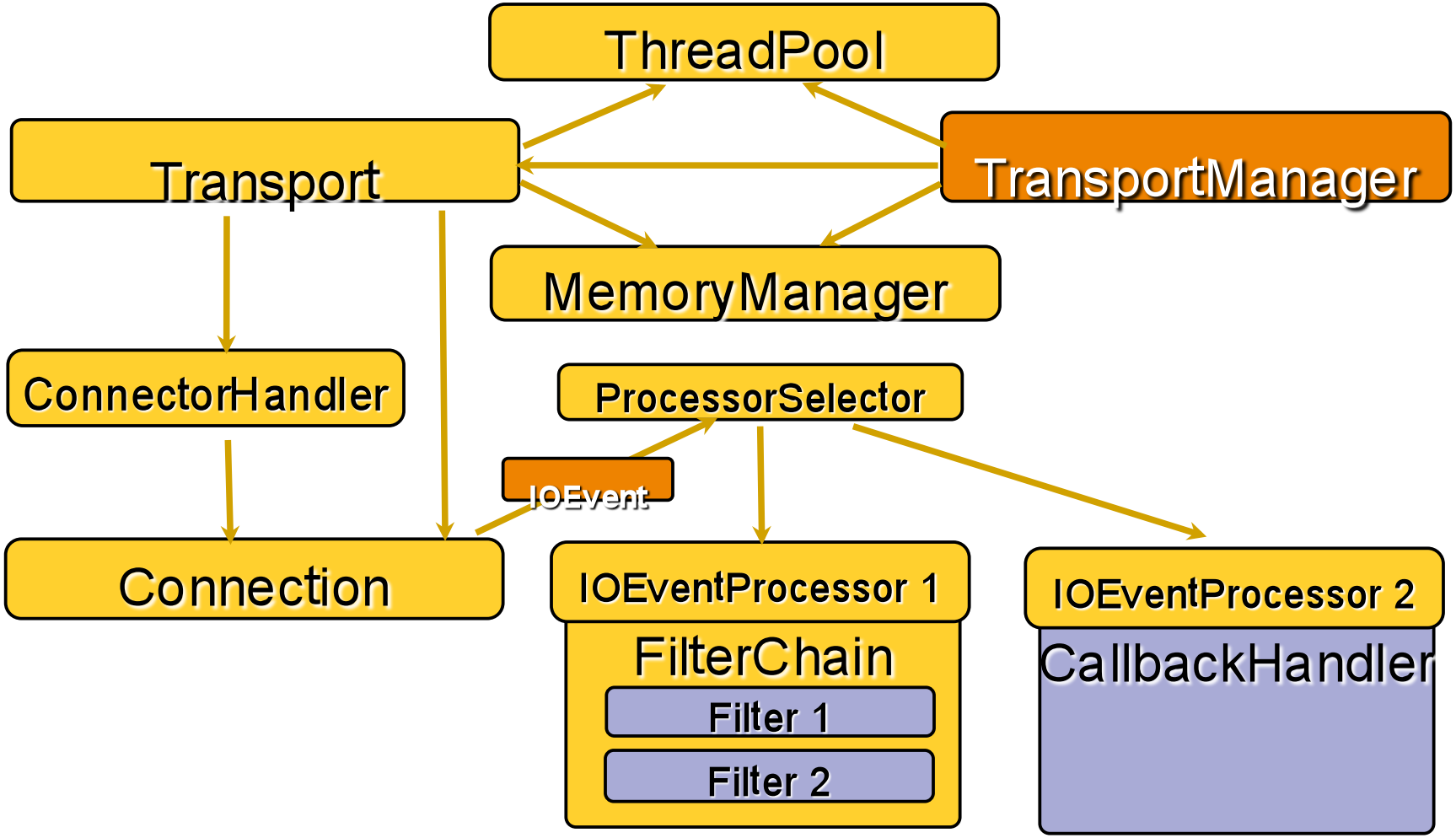
4.1 Introduction

- Started in June 2008.
- Completely new Grizzly core API.
- No backwards compatibility requirement.
  - 1.0 suffered design limitation like supporting SJSAS, Tomcat API **AND** Sun Web Server API. This was a major restriction and some API are quite ugly.
  - 1.5 was a tentative to keep our 1.0 Community on board, and grow.
  - 1.8.x is successful, but is now time to do a revolution, e.g. bye bye GlassFish API ☺

4.2 Goals

- Make framework core API more clear and useful, based on earlier experience. Especially client side.
- Performance. Make sure API changes will not affect the performance, which is still the main goal for Grizzly.
- Introduce unified Buffer/Memory management API.
- Unify ThreadPool implementation. Make it compatible with ExecutorService, provided by JDK.
- **Reborn a Grizzly based on Community Feedback and lessons learned since 2004.!**

4.3 Architecture



4.3 New features

- Filter chain standalone read/write.
- Message parsing using streams.
- Smart message parser

4.3.1 FilterChain standalone read/ write

- Leverage filters logic for a client side.
- Minimize the effort to build nice client code.
- Supports blocking and asynchronous modes

4.3.1 FilterChain standalone read/ write

```
TCPNIOTransport transport =
                        TransportManager.instance().createTCPTransport();

transport.getFilterChain().add(new TransportFilter());

transport.getFilterChain().add(new SSLFilter());

transport.getFilterChain().add(new StringFilter());

transport.getFilterChain().add(new EchoFilter());

--- (1) start the "transport" in the server mode

--- (2) connect client the "connection" to the "transport"

Future writeFuture = transport.getFilterChain().write(connection, "Hello
    world");

Future readFuture = transport.getFilterChain().read(connection);

String result = (String) readFuture.get();

// The result should contain "Hello world", sent by server
```

4.3.2 Message parsing using streams

- Simplify message parsing.
- No ByteBuffer positions, limits, capacities!
- Simple stream API, which supports Java primitive types and arrays.

4.3.2 Message parsing using streams

```
// initialize the filter chain
transport.getFilterChain().add(new TransportFilter());
transport.getFilterChain().add(new StreamFilter());
transport.getFilterChain().add(new CustomProtocolParser());
------------------------------ cut -------------------------------------
public class CustomProtocolParser extends FilterAdapter {
    public NextAction handleRead(FilterChainContext context) {
        StreamMessage message =
                        (StreamMessage) context.getMessage();
        Reader reader = message.getReader();
        if (reader.availabe(4)) {
            int a = reader.getInt();
            int b = reader.getInt();
            message.getWriter().putInt(a+b);
```

4.3.3 Smart message parser

- Knows how to parse the custom message, based on message class structure.
- POJO in enough, however could be driven by annotations.
- It's possible to provide custom encoder/decoder for complex types.

4.3.3 Smart message parser

```
transport.getFilterChain().add(new TransportFilter());
transport.getFilterChain().add(new SSLFilter());
// Parse the data according to MyMessage class
transport.getFilterChain().add(new SmartParserFilter(MyMessage.class));
// Echo MyMessage
transport.getFilterChain().add(new EchoFilter());
------------------------------ cut ----------------------------------
public class MyMessage {
    public MyMessage(int a, int b) { this.a = a; this.b = b;}
    public int a;
    public int b;
}
```

4.4 Example #1: Start the echo server

```
TCPNIOTransport transport =
                                TransportManager.instance().createTCPTransport();
transport.getFilterChain().add(new TransportFilter());
transport.getFilterChain().add(new EchoFilter());
try {
        transport.bind(PORT);
        transport.start();
---------- wait until server is running ---------
} finally {
        transport.stop();
        TransportManager.instance().close();
}
```

4.4 Example #2: Connect the client

```
TCPNIOTransport transport =
                        TransportManager.instance().createTCPTransport();
try {
    transport.start();
    Future future = transport.connect(HOST, PORT);
    Connectin connection =
                        future.get(connectionTimeout, TimeUnit.SECONDS);
    Future writeFuture = connection.write(message);
    Future readFuture = connection.read();
    System.out.println("Result: " + readFuture.get());
} finally {
    transport.stop();
    TransportManager.instance().close();
}
```

4.4 Example #3: EchoFilter

```
public class EchoFilter extends FilterAdapter {

    @Override
    public NextAction handleRead(FilterChainContext ctx,
            NextAction nextAction) throws IOException {
        Object message = ctx.getMessage();
        ctx.getFilterChain().write(ctx.getConnection(), message);
        return nextAction;
    }
}
```

5 Roadmap

- 1.8.6 was released 09/24
  - > Biggest tested community release.
- 1.9.0 still discussed
  - > Will include NIO.2 support
  - > May include an http-client new module
    - – Depends on the community
  - > Last release for the 1.x branch.
- Release Grizzly 2 framework core: December 2008
- Release Grizzly 2 http module: February 2009

6. Summary

- Project Grizzly is a simple NIO Framework.
  - With less than 50 lines, you can create power client or server side components.
  - Support Java NIO and NIO.2
  - Simple, but customizable.
- Grizzly 2.0 will be better than the original! Better API, same performance!
- *Healthy community: get a response in less than 2 hours :-)!*

For More Information

- Grizzly Active's Bloggers:
  - Alexey: http://blogs.sun.com/oleksiys/
  - Shing Wai: http://blogs.sun.com/swchan/
  - John: http://weblogs.java.net/blog/johnmann/
  - Sebastien: http://www2.sebastiendionne.ca
  - Jeanfrancois: http://weblogs.java.net/jfarcand
- News:
  - Every week or so, a list of interesting blogs talking about us
    http://www.nabble.com/forum/Search.jtp?forum=23249&local=y&query=News
- Project Grizzly mailing lists,
  - dev@grizzly.dev.java.net & users@dev.grizzly.java.net