

Introduction to the Grizzly Framework

Jeanfrancois Arcand

Senior Staff Engineer

Java WebTier

Agenda

- Introduction
 - > What is Grizzly
- The Grizzly Framework Architecture
- The Grizzly Extension
 - > Asynchronous Request Processing
 - > HTTP WebServer
 - > Comet and Cometd
 - > Application Resource Allocation
- Grizzly.Next
- Discussion

What is Grizzly Framework

- Grizzly is a multi protocols (HTTP, UDP, etc.) framework that uses lower level Java NIO primitives, and provides high-performance APIs for socket communications.
- Grizzly support blocking and non blocking socket operations, over plain or ssl connection.
- Source available:
 - > Started in glassfish/appserv-http-engine
 - > Moving to grizzly.dev.java.net

What is Grizzly Framework

- Grizzly started as a prototype for 8.0 PE and then:
 - > 8.1 PE – Throttled to avoid comparison with 8.1 EE
 - > 8.2 PE – Un throttled
 - > 8.2 EE – Available but not the default.
 - > 9.0 PE – Full version
 - > 9.1 PE/EE – With SSL non blocking support.

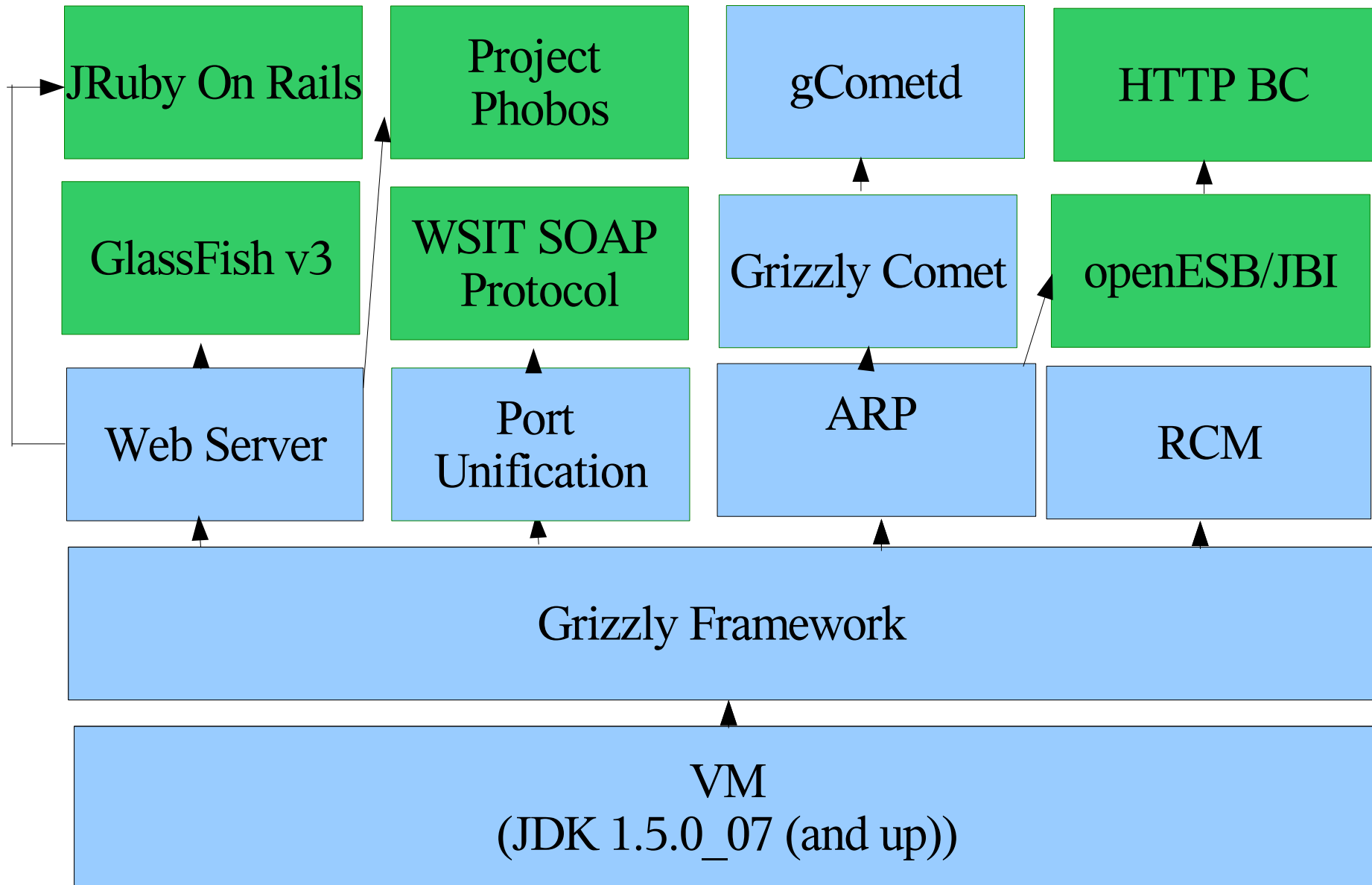
What is Grizzly Exactly (Cont.)

- Grizzly is currently used by several projects:
 - > Internally:
 - > Tango (WSIT) JRuby on Rails
 - > Alaska (Open ESJ) TCP Port Unification
 - > GlassFish
 - > Phobos (NetBeans Integration)
 - > Some concepts|code are duplicated in the ORB
 - > Externally
 - > AsyncWeb
 - > Jetty
 - > Brane Corporation/ Oracle
 - > 4Himedia
 - > Ning

Who collaborate on Grizzly

- Charlie Hunt
- Harold Carr
- Oleksiy Stashok
- Ken Cavanaugh
- Scott Oaks
- Naoto Takai
- Jan Luehe
- Jeanfrancois Arcand

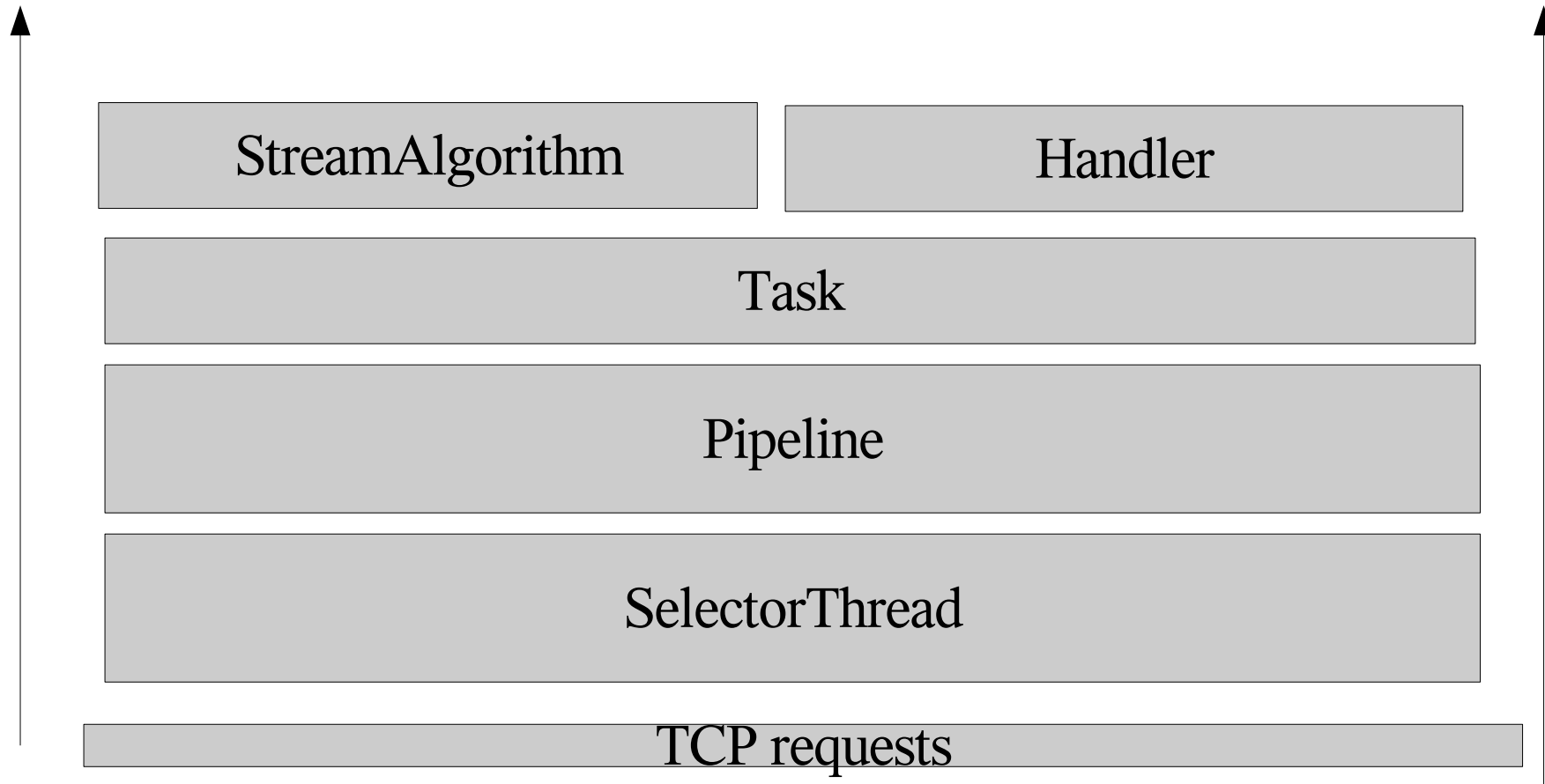
The Grizzly Framework Architecture



Grizzly Framework

- Grizzly can be extended in several places, from the management of low level byte buffer to thread management.
- Next couple of slides will describe where and how you can extend Grizzly.

Grizzly Extensible Components



Definitions

- SelectorThread: `java.nio.channels.Selector` implementation.
- Pipeline: An execution queue. Most of the time implemented as a wrapper around a thread pool.
- Task: An execution token which handles the life cycle of the `SocketChannel`
- StreamAlgorithm: The strategy used to pull out bytes from `SocketChannel` (when and how)
- Handler: a life cycle interceptor that can be used to manipulate the request/response bytes

Definition: SelectorThread

- Main entry point in Grizzly.
- Handles NIO low level events:
 - SelectionKey events: OP_ACCEPT, OP_READ
 - SelectionKey registration/de-registration with the `java.nio.channels.Selector`.
- Handles the allocation and life cycle of Pipelines, Tasks, StreamAlgorithms and Handlers.

Definition: Pipeline

- Responsible of the execution of a Task. The Pipeline can execute using the caller thread or create its own thread pool.
- Grizzly ships by default with two implementations:
 - `LinkedListPipeline`: thread pool based on a linked list
 - `ExecutorServicePipeline`: based on `java.util.concurrent.Executors`

Definition: Task

- Execution token that can be executed by a thread pool (implement `java.lang.Runnable`).
- Configurable using via the `StreamAlgorithm`.
- The `SelectorThread` will handle the life cycle of Tasks.
- Usually implement the request logic operations, e.g. How and when to read bytes, how to write bytes, etc.

Definition: Task (Cont')

- Several implementations:
 - ReadTask: handle the SocketChannel operations and decide what to do next using its associated StreamAlgorithm
 - ProcessorTask: HTTP processing implementation. Parse the HTTP request header and body.
 - AsyncReadTask: Same as ReadTask but can handle asynchronous requests.

Definition: StreamAlgorithm

- Implement the strategy of deciding:
 - The ByteBuffer type (direct, heap or view)
 - Deciding when we start/stop reading bytes from the SocketChannel.
 - The registration/de-registration on the SelectionKey with the main Selector (SelectorThread) or using a temporary Selector.
 - Ship with three implementation.

Definition: Handler

- Used to intercept the request/response.
- Can be added on the fly.
- GlassFish uses Handler for implementing a cache mechanism.

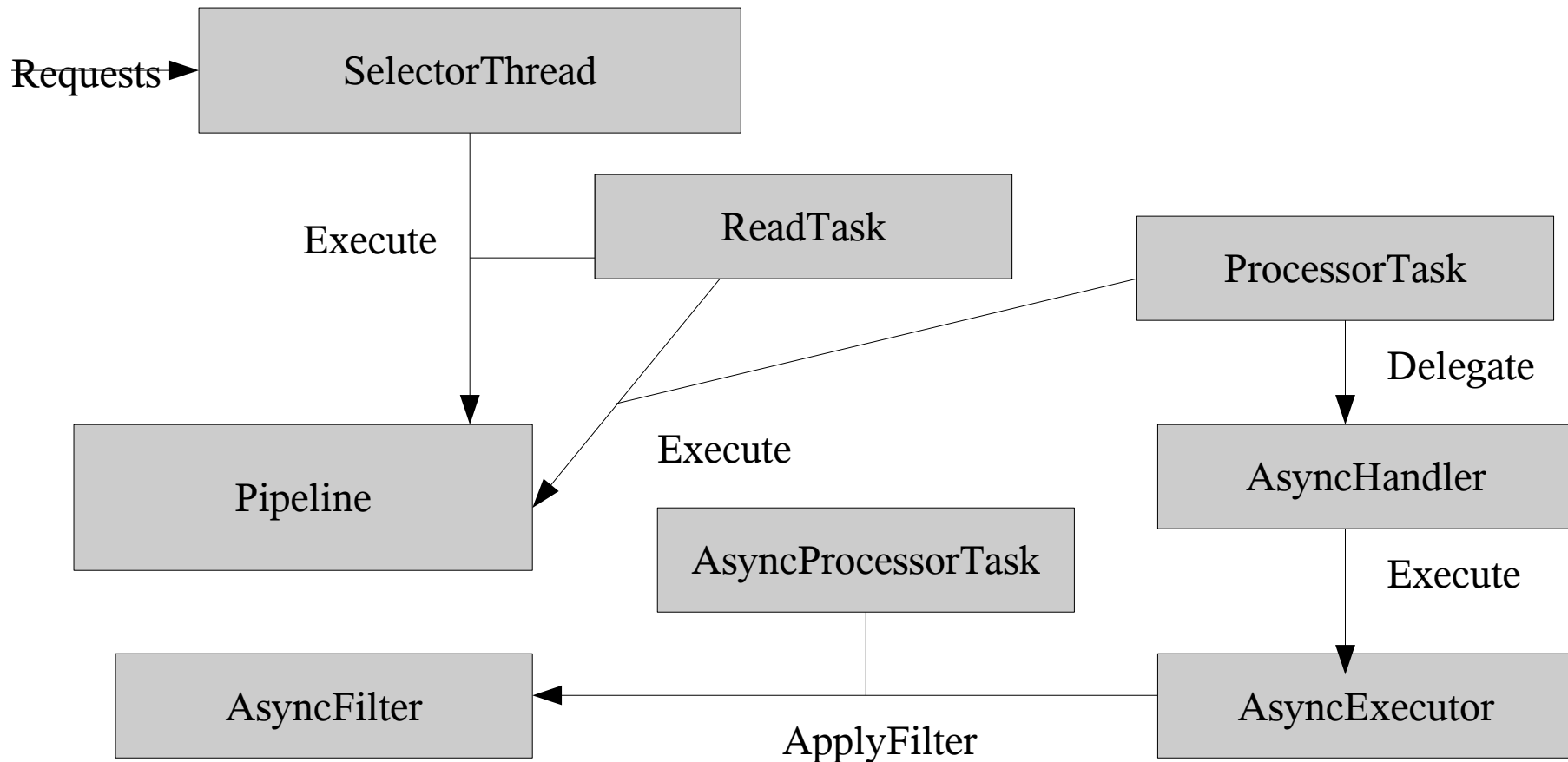
Grizzly extension: Asynchronous Request Processing

- Allow for “parking” a request; a type of “continuation” at the request processing level
- The goal is to be able to build, on top of Grizzly, a scalable ARP implementation that doesn't hold one thread per connection, and achieve as close as possible the performance of synchronous request processing (SRP).

Grizzly extension: Asynchronous Request Processing

- By default, every TCP connection are executed synchronously, e.g the request is parsed, the endpoint executed and the response flushed to the client.
- There is situation where this model doesn't work, e.g. when a business process is calling out to another one over a slow protocol, or if there is a work flow interruption, like the "manager approval email" case

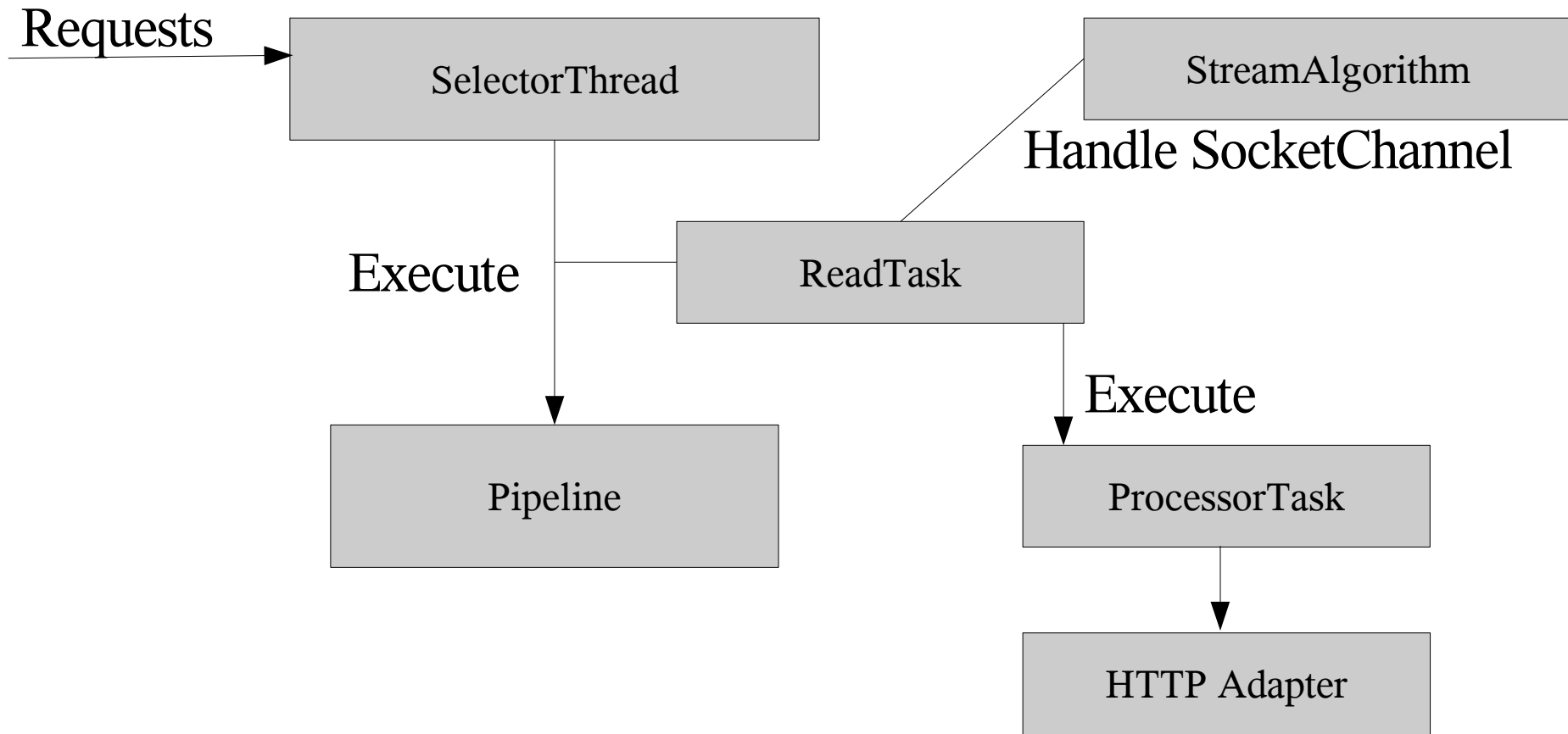
ARP Implementation Details



Grizzly WebServer

- Lightweight HTTP 1.0/1.1 WebServer
- Extremely easy to embed.
- Small footprint.
- Performance is extremely good.
- Discussion to add it to Ubuntu distribution.

Example: Grizzly Web Server



What is Comet Request Processing

- Definition from Wikipedia:

Comet is a programming technique that enables web servers to send data to the client without having any need for the client to request for it. It allows creation of event-driven web applications which are hosted in the browser.

What makes Comet Application different from traditional application (from <http://alex.dojotoolkit.org/?p=545>)?

- Fundamentally, they all use long-lived HTTP connections to reduce the latency with which messages are passed to the server.
- In essence, they do not poll the server occasionally. Instead the server has an open line of communication with which it can push data to the client.

What makes Comet Application different from traditional application? (from <http://alex.dojotoolkit.org/?p=545>)

Comet applications can deliver data to the client at any time, not only in response to user input. The data is delivered over a single, previously-opened connection. This approach reduces the latency for data delivery significantly.

Comet support in Grizzly: Details

- Implemented on top of the Grizzly Asynchronous Request Processing extension.
- Hide the complexity of NIO/Asynchronous Request Processing.
- Support clean and ssl connection.
- Make it available to JSF, JSP, Servlet, POJO, JavaScript (Phobos)
- **Main Goal: Make it simple!**

gCometd: Grizzly Cometd implementation

- Cometd is a scalable HTTP-based event routing bus that uses a push technology pattern known as Comet.
- This is well suited for Ajax web applications that allow multi channel messaging between client and server - and more importantly - between server and client. The paradigm is publish/subscribe to named channels.
- gCometd is built on top of Grizzly Comet.

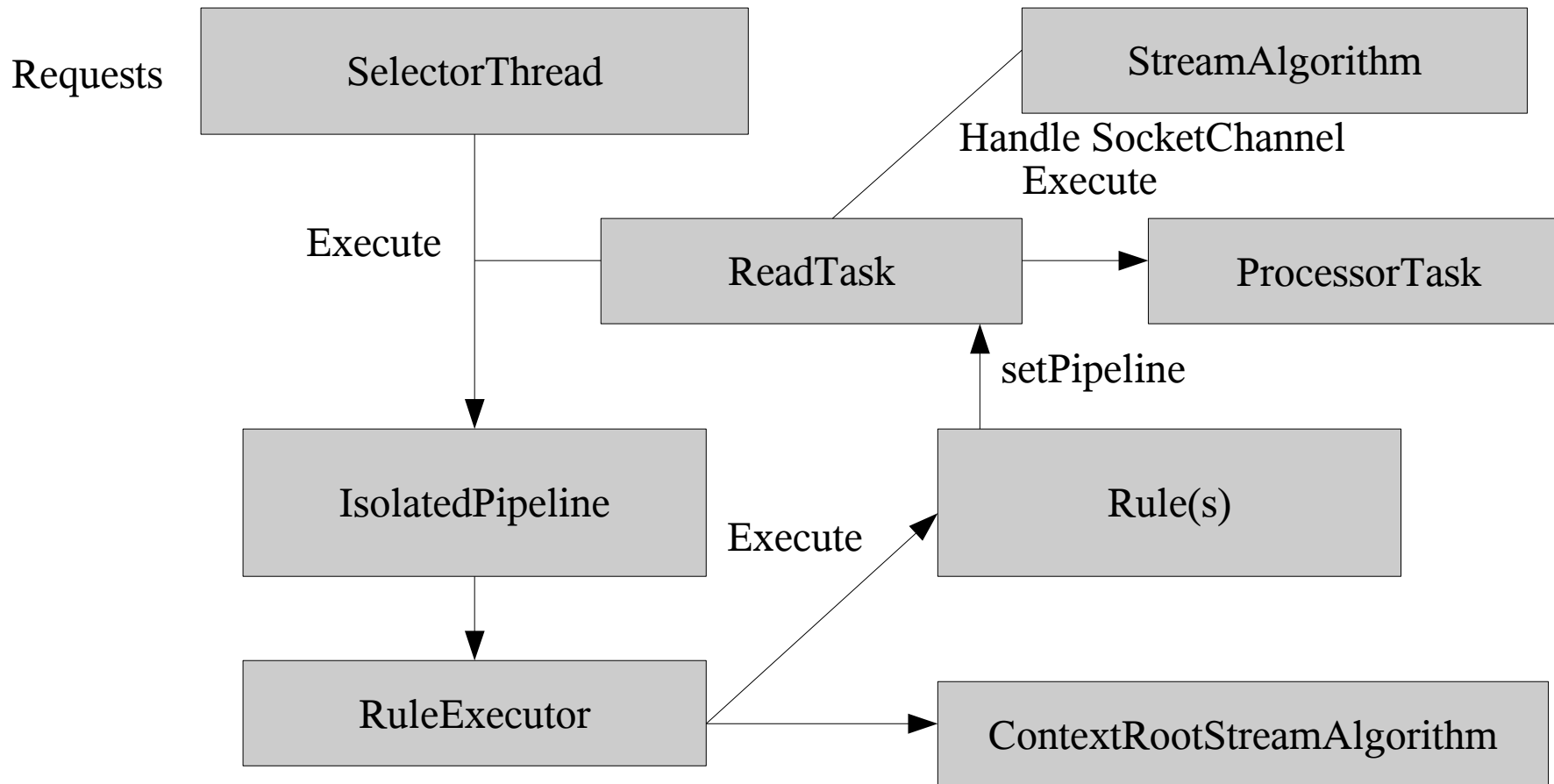
Grizzly extension: Application Resources Allocation

- The Grizzly's Application Resources Allocation (ARA) extension is an implementation of a Resource Consumption Management (RCM) system.
- RCM system are allowing you to enable virtualization of system resources per web application, similar to Solaris 10 zone or the outcome of the upcoming JSR 284.

Grizzly extension: Application Resource Allocation (ARA)

- Grizzly's ARA currently supports two rules that can be applied to a tcp request:
 - > Reserve a specific percentage of the available heap memory.
 - > Reserve a specific percentage of the available threads.

ARA Implementation Details



Example: ARA

- Lets say there are 100 threads in the Grizzly request processing thread pool (Pipeline).
- We reserve 30% of the Pipeline's threads to App A and 65% to App B.
- Now that leaves only 5% to other applications.
- So even if the other 95% threads are idle, we delay requests until the 5% unreserved threads can service requests.

Grizzly.Next

- An effort to unify all existing NIO implementation into a single unified framework. This include the current ORB, MQ, Jetty etc.
- V1.x is still difficult to extends as some NIO low level API are exposed. Grizzly.Next will address that issue.
- Grizzly.Next is a complete re-write, with no dependencies/limitations on GlassFish and SJS WebServer.

Discussion

- Browse documentations here:
 - > <https://grizzly.dev.java.net>

Introduction to the Grizzly Framework