



# Zero Latency HTTP

## The comet Technique

---

Filip Hanik

SpringSource Inc

Keystone, Colorado, 2008





# Who am I – bla bla

---

- [fhanik@apache.org](mailto:fhanik@apache.org)
- Tomcat Committer / ASF member
- Co-designed the Comet implementation
- Implemented NIO connector in 6
- Responsible for session replication and clustering
- Been involved with ASF since 2001



# What we will cover

---

- Brief History of HTTP
- How HTTP is used today
- Introduction to AJAX
- Beyond AJAX, The Comet Technique
- Comet Implementation – Under the hood
- Uni- vs. Bidirectional Comet
- Problems and their solutions
- Demo
- Q & A



# The History of HTTP

---

- 1.0 (7th release) RFC 1945 in 1996
- Co-authored by ASF's Roy T Fielding
- Current RFC 2616
- Lead by W3C
- Activity on specs are closed
- Standard has been achieved
- <http://www.w3.org/Protocols/History.html>



# What is HTTP

---

- HyperText Transfer Protocol
- Text based protocol
- Request/Response semantics
- Stateless
- Most commonly run over TCP/IP networks
- Protocol used for much more than just hypermedia information sharing



# What is HTTP

---

- Text based protocol – CRLN delimited

```
GET /Protocols/History.html HTTP/1.1CRLN
Host: www.w3.orgCRLN
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
  rv:1.8.1.2) Gecko/20070219 Firefox/2.0.0.2CRLN
Accept:
  text/xml,application/xml,application/xhtml+xml,text/html;q=
  0.9,text/plain;q=0.8,image/png,*/*;q=0.5CRLN
Accept-Language: en-us,en;q=0.5CRLN
Accept-Encoding: gzip,deflateCRLN
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7CRLN
Keep-Alive: 300CRLN
Connection: keep-aliveCRLN
CRLN
```



# What is HTTP

---

- Request/Response Semantics

```
GET /Protocols/History.html HTTP/1.1
Host: www.w3.org
Keep-Alive: 300
Connection: keep-alive
```

*Request*

```
HTTP/1.x 200 OK
Date: Mon, 19 Mar 2007 15:46:17 GMT
Server: Apache/1.3.37 (Unix) PHP/4.4.5
Keep-Alive: timeout=2, max=100
Content-Length: 19575
Connection: Keep-Alive
```

*Response*



# What is HTTP

---

- Stateless
  - TCP session can end after each request
  - State based on cookies (or other means)
- TCP/IP transport
  - No limitations for transporting with other protocols
- Delivers both text and binary data





# HTTP/Browser Limitations

---

- Request/Response
  - Change one field on a page requires a reload of the entire page
  - Client always has to initiate the request
- Stateless
  - Server required to keep state
  - State is timed out, if client is not cancelling it
- Finer grained communication is needed
  - AJAX
  - Rich Clients (Applets, FLEX, OpenLaszlo, ...)



# AJAX

---

- Asynchronous JavaScript + (and) XML
- Uses HTTP
- Asynchronous data processing
- Able to request data from server and update a page in the browser
- Common examples
  - [maps.google.com](http://maps.google.com)



# Benefits of AJAX

---

- Performance
  - More can be accomplished in less amount of transactions
- Less user interaction
  - Program can make intelligent decision about when a request needs to happen and what data it needs to fetch
- Sample Application for view of benefits
  - <http://www.ajaxinfo.com/ucst/index.html>



# Beyond AJAX

---

- What did AJAX not accomplish?
  - Still client poll based
  - Server push can be accomplished by a client poll followed by a delayed response
  - Traditional web/servlet containers are thread-per-request based
  - Server resource tied up for the duration of the request



# Introducing Comet

---

- The answer to our problems
  - Q: What problems?
  - A: The fact that port HTTP/80 has been replacing every decent TCP protocol over the years.
  - Q: What do you mean?
  - A: It all boils down to port 80/443 being the only open port to operate through

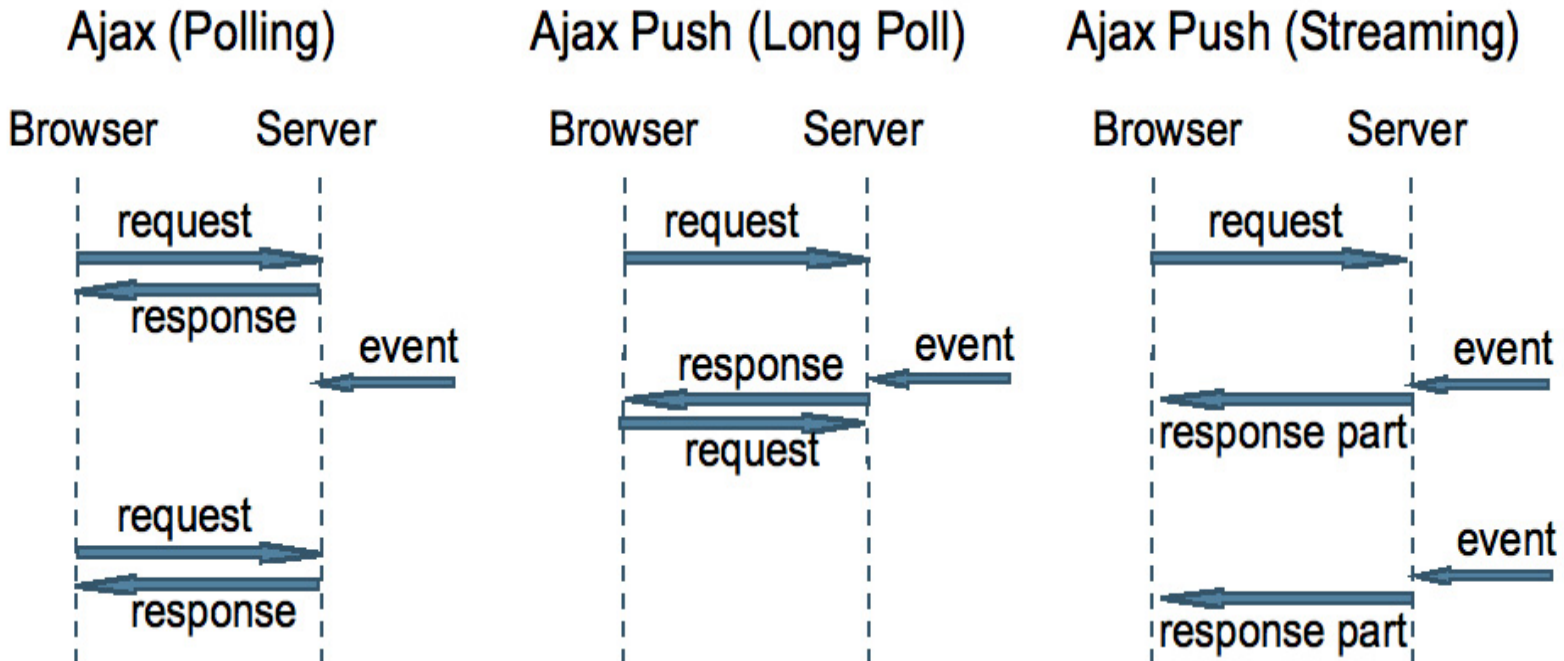


# Introducing Comet

---

- Send response when data is available
- Send response in chunks, as data becomes available
- Open up HTTP for full duplex operations
- In reality
  - LAN full duplex with thick client
  - WAN half duplex due to proxies, *etc.*

# Can we connect?





# Poll

---

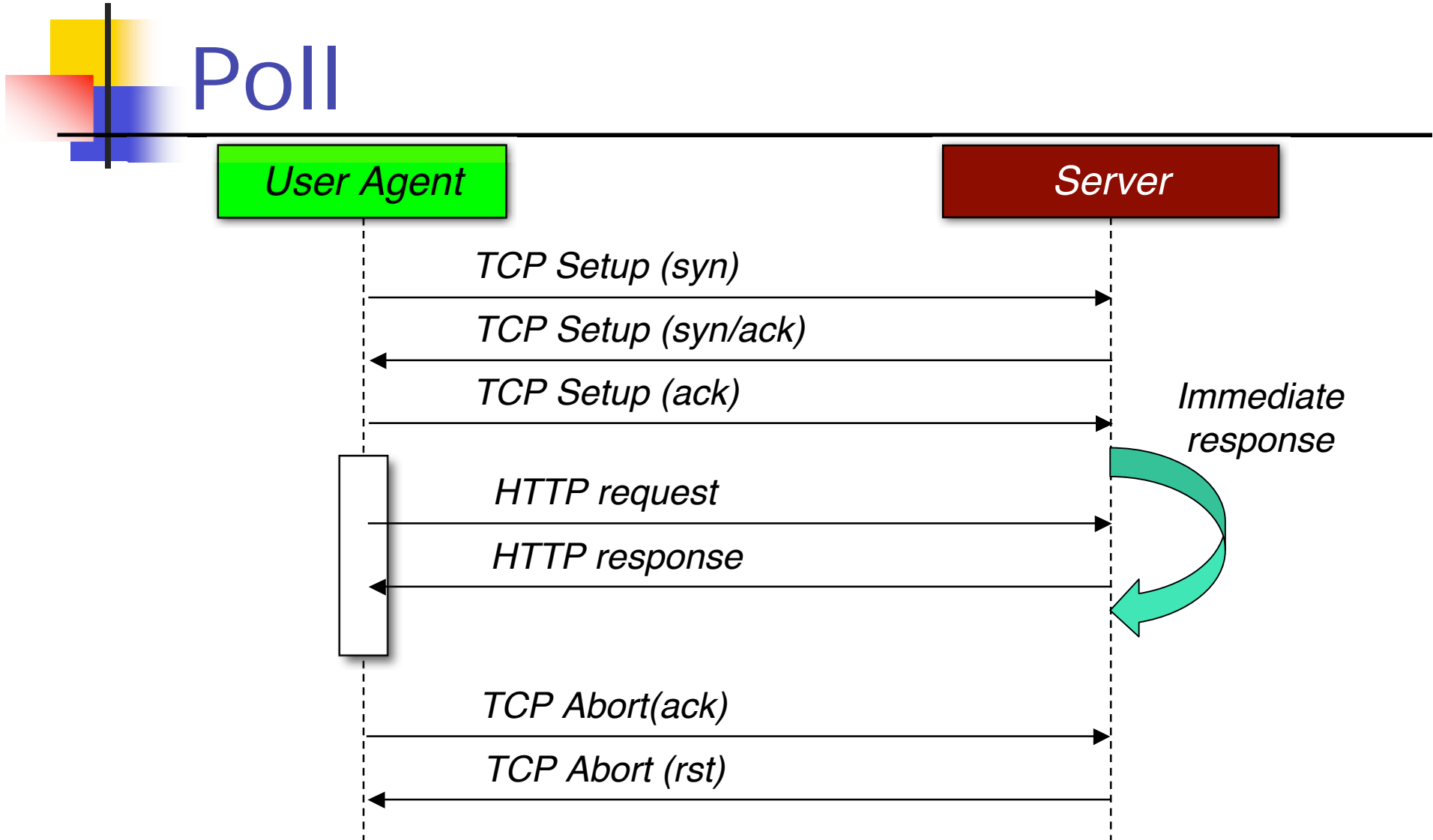
➤ Poll:

- Send a request to the server every X seconds.
- The response is “empty” if there is no update.

➤ Disadvantage:

- Long intervals between updates
- In order to get more frequent updates, more frequent HTTP requests (possibly connections)
- Risk of most requests being done in vain







# Long Poll

---

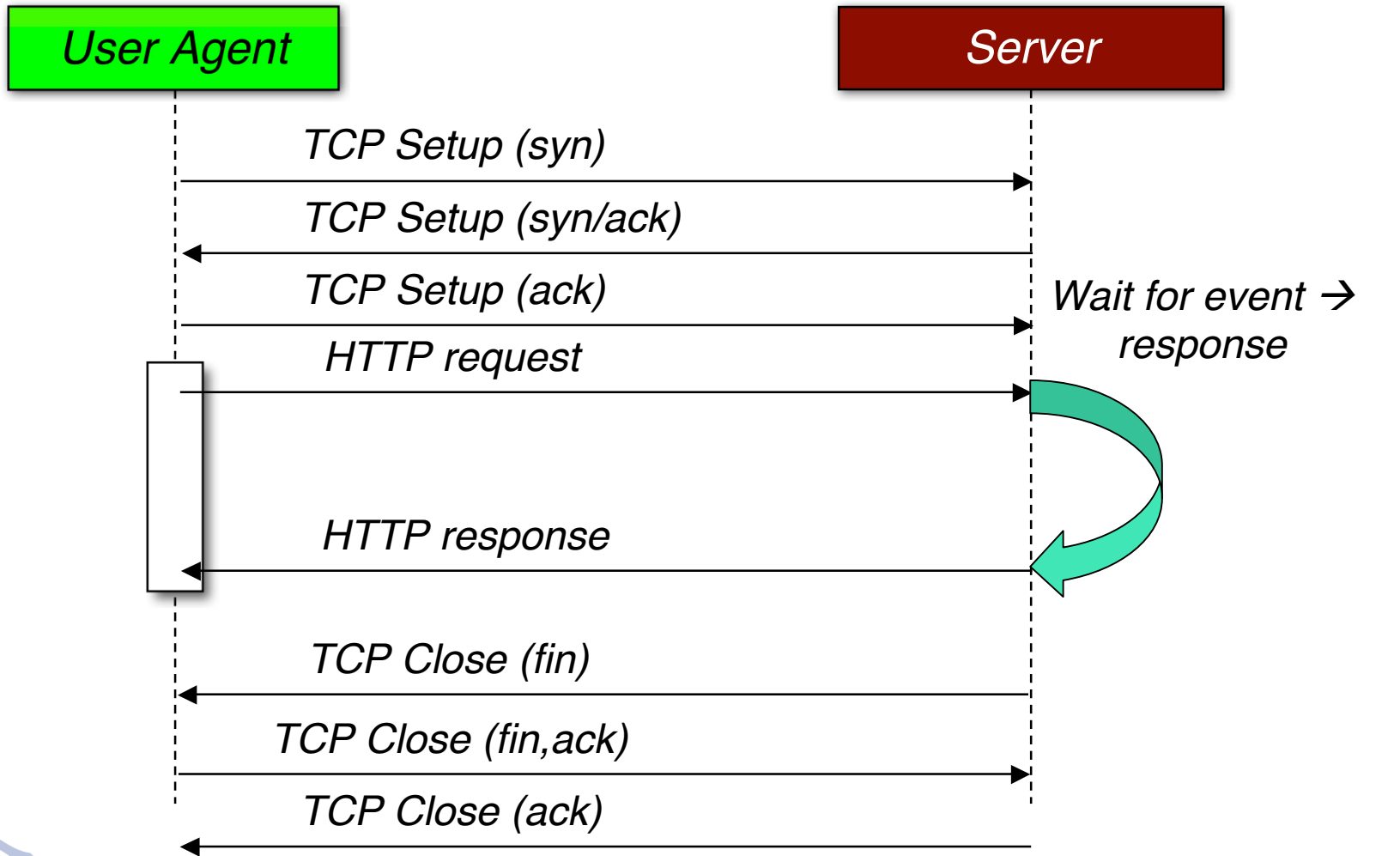
## ➤ Long Poll:

- Send a request to the server, wait for an event to happen, then send the response.
- The response only timeout causes empty responses
- HTTP specification satisfied: indistinguishable from “slow” server

## ➤ Disadvantage:

- Still very resource intensive
- Response delay in between requests

# Long Poll



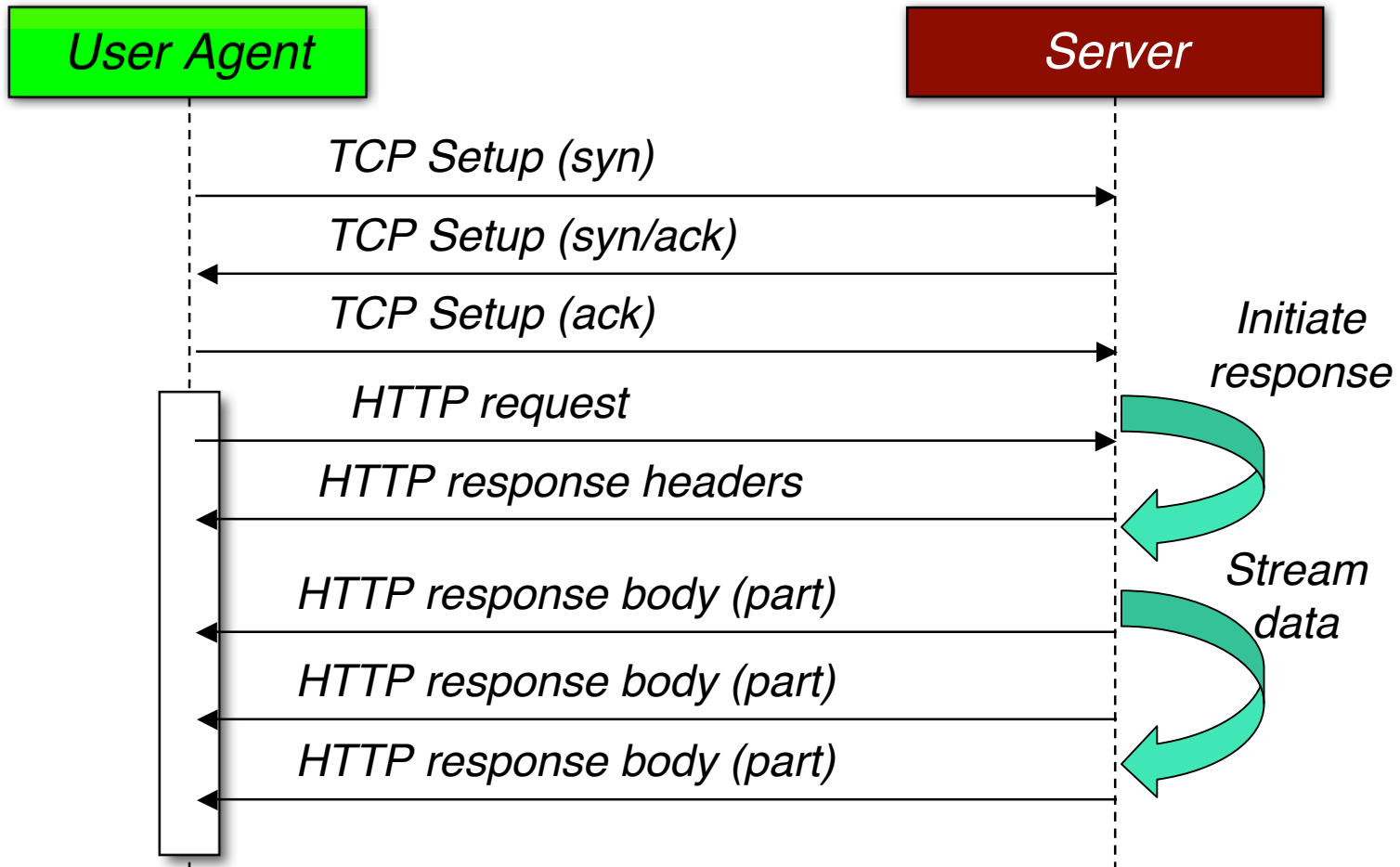


# Streaming

---

- Http Streaming:
  - Send a request, wait for events, stream multi-part/chunked response, and then wait for the events.
  - The response data is continually appended to a single HTTP response body
- Disadvantages:
  - Proxies, anything that buffers layer 7 data (HTTP)
  - More complex programming model

# Streaming





# Comet on the Server

---

- All this can be accomplished with a regular servlet
  - Servlet API is blocking
  - But you are using a Servlet thread
  - Limited Scalability
- Tomcat's CometProcessor
  - Decouples the thread from the request
  - Responses can happen async



# Under the hood

---

- Open HTTP request, leave it open ended
- Instant or delayed response(s), leave it open ended
- Tomcat processes just like servlet
  - CometProcessor extends HttpServlet
  - CometFilter extends Filter
- Connection remains open and is writable




# Under the Hood

---

- A normal HTTP request

```
POST /load/echo HTTP/1.1
User-Agent: Filips Comet Client/1.0
Host: 127.0.0.1:8080
Transfer-Encoding: chunked
<CRLF>
10
test data test 1      “End of request body”
0
```







# Under the Hood

---

- Open ended request

```
POST /load/echo HTTP/1.1
User-Agent: Filips Comet Client/1.0
Host: 127.0.0.1:8080
Transfer-Encoding: chunked
<CRLF>
10
test data test 1
```

*“No end”*



- Open ended allows for new data without the overhead for a new request

➤ Zero Latency HTTP




# Under the Hood

---

- Normal response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Transfer-Encoding: chunked
Content-Type: text/plain;charset=UTF-8
Transfer-Encoding: chunked
Date: Mon, 19 Mar 2007 22:25:36 GMT
<CRLF>
12
test data test 1
0
```

*“End of response body”*



# Under the Hood

---

- Open ended response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Transfer-Encoding: chunked
Content-Type: text/plain;charset=UTF-8
Transfer-Encoding: chunked
Date: Mon, 19 Mar 2007 22:25:36 GMT
<CRLF>
12
test data test 1
```

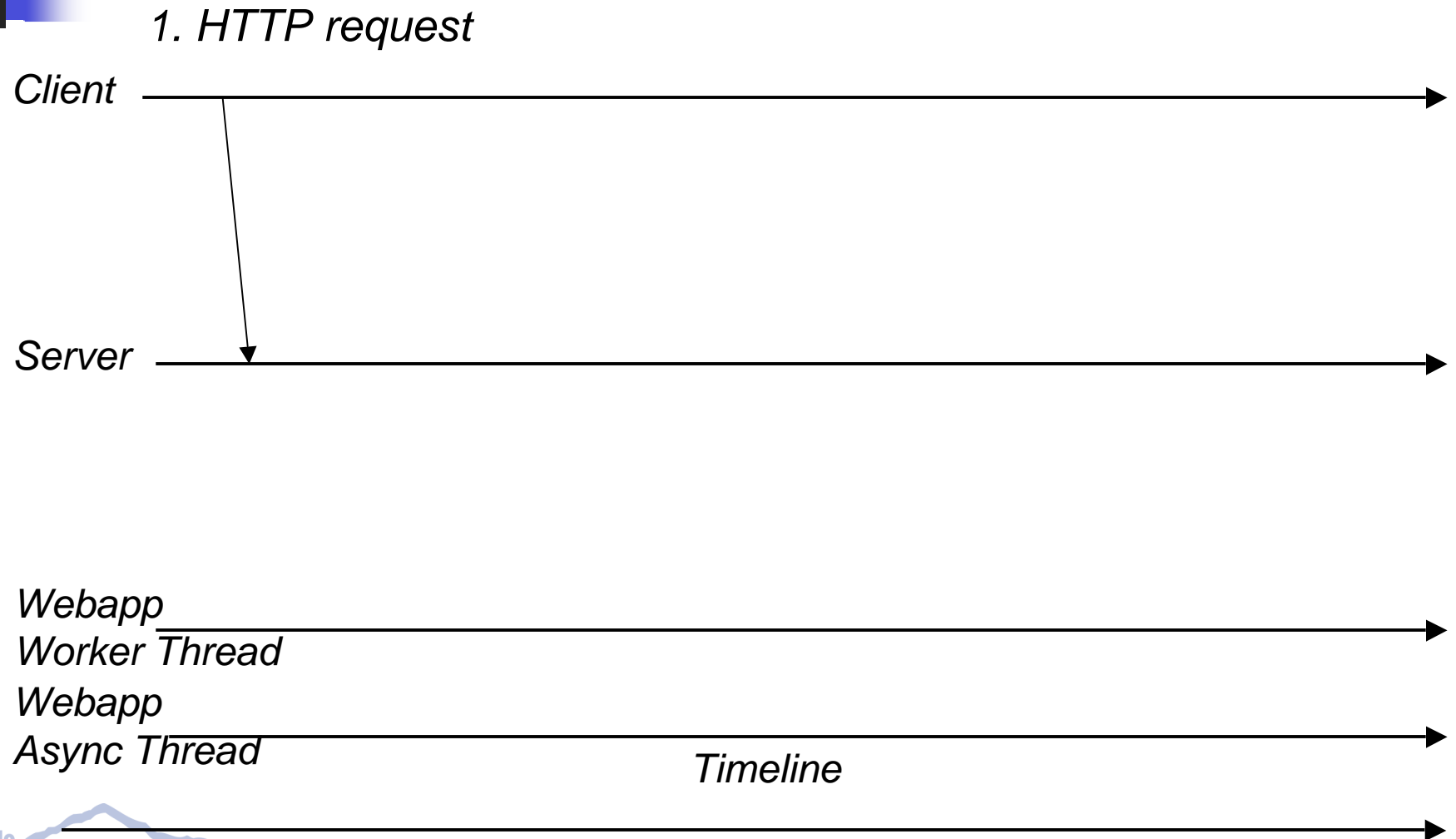
*“No end”*



- Open ended allows for new data without the overhead for a new request
  - Zero Latency HTTP



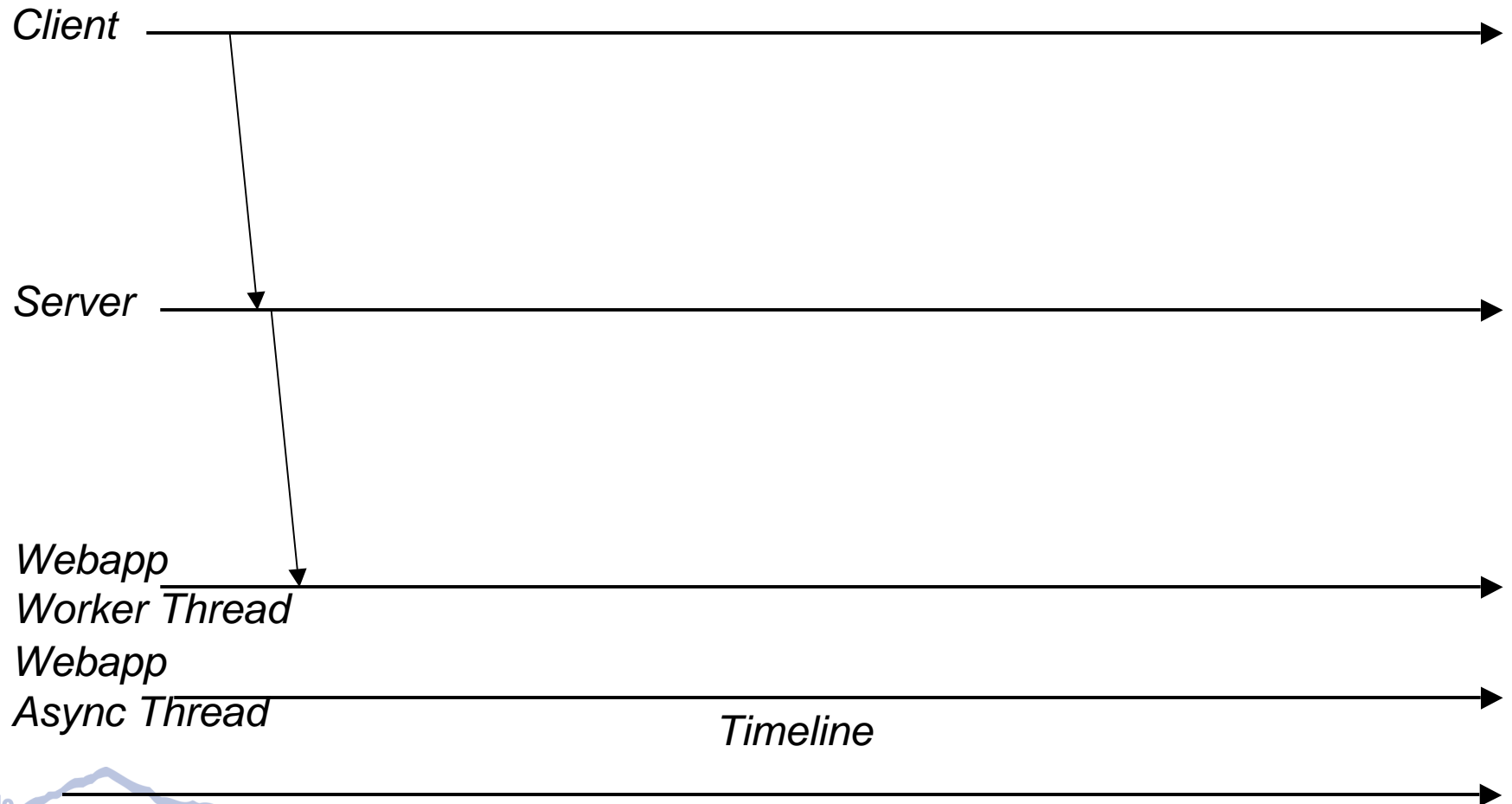
# How it Works





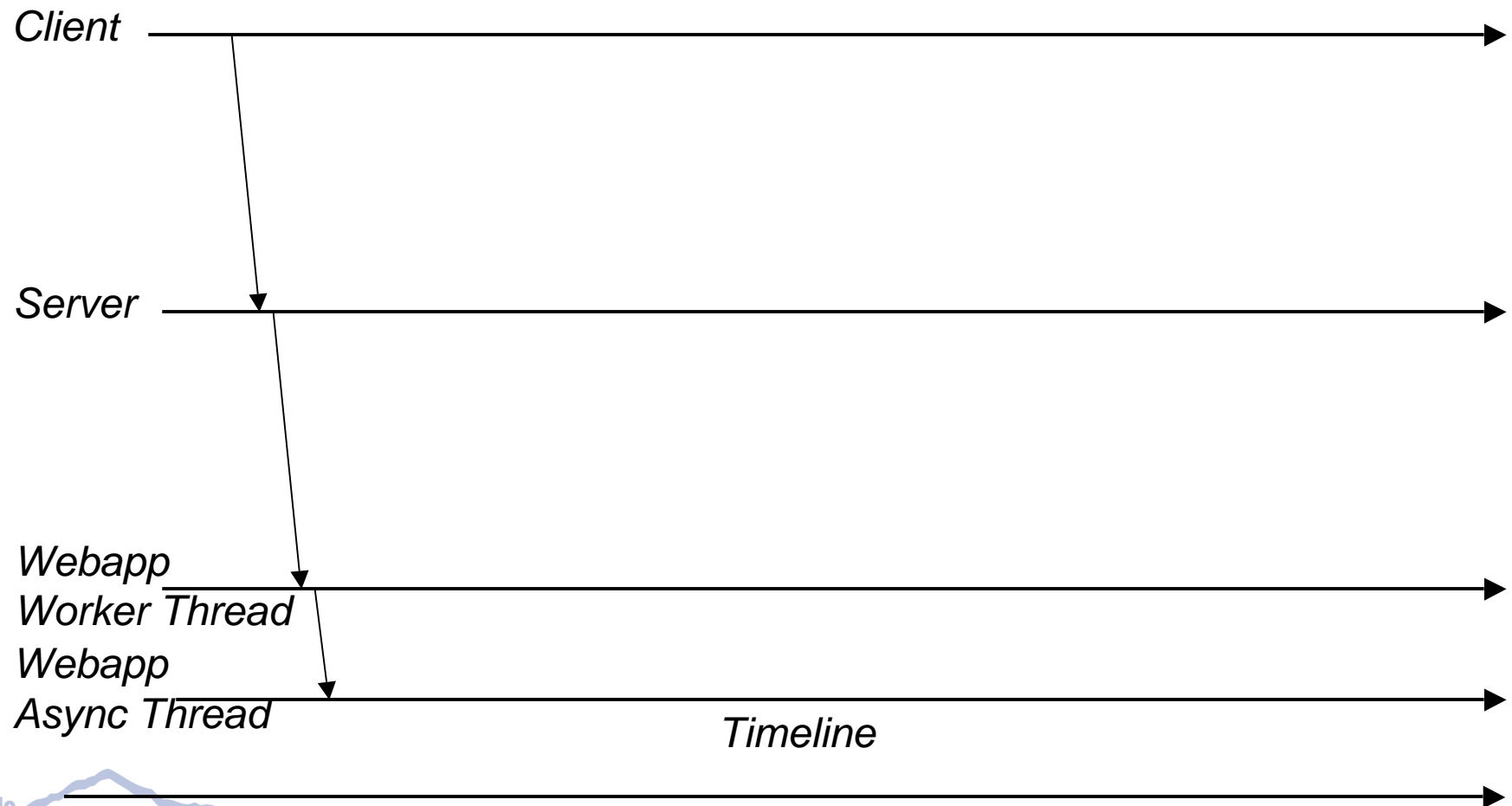
# How it Works

## 2. BEGIN Event



# How it Works

## 3. Register with background thread

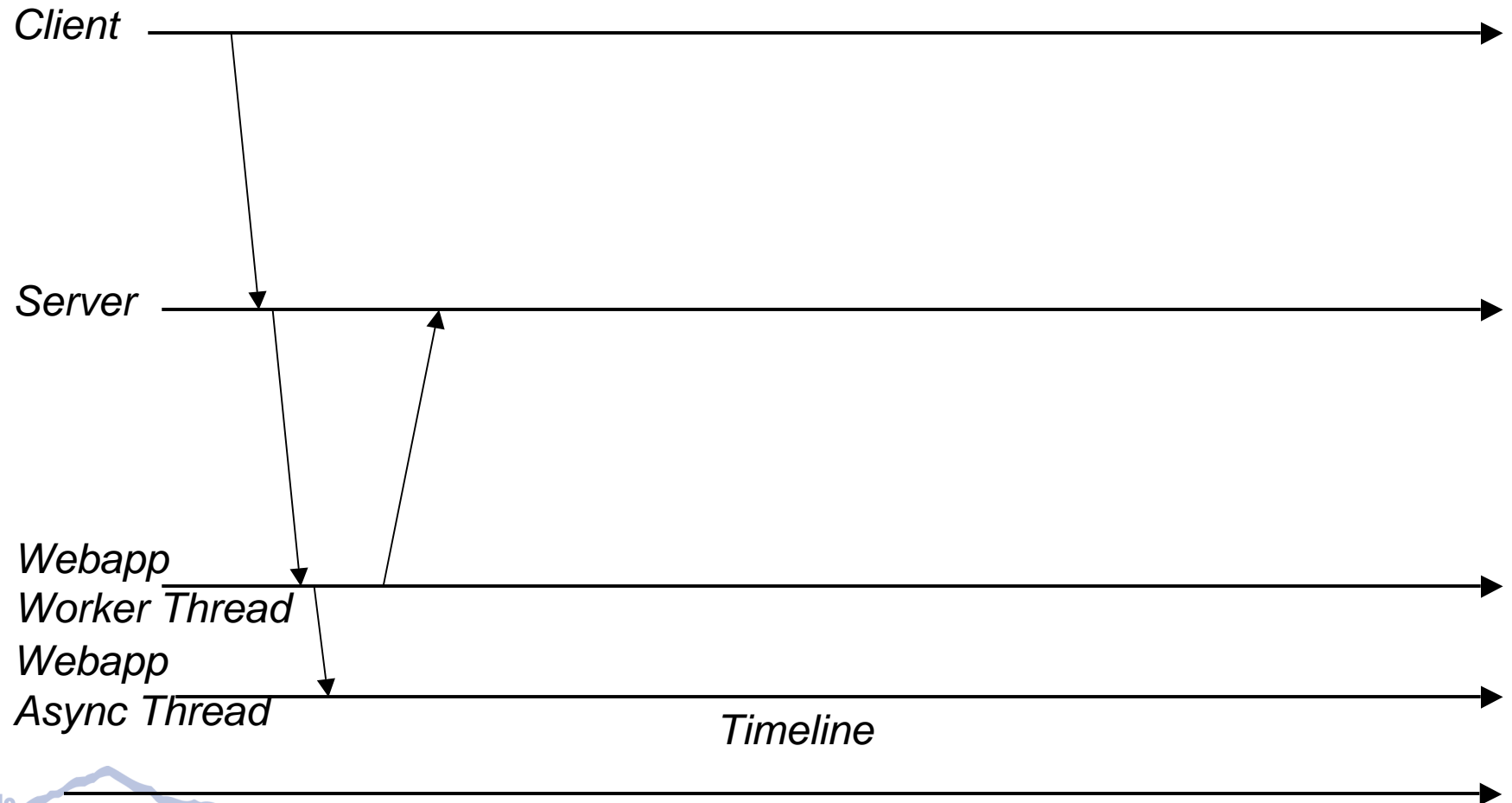




# How it Works

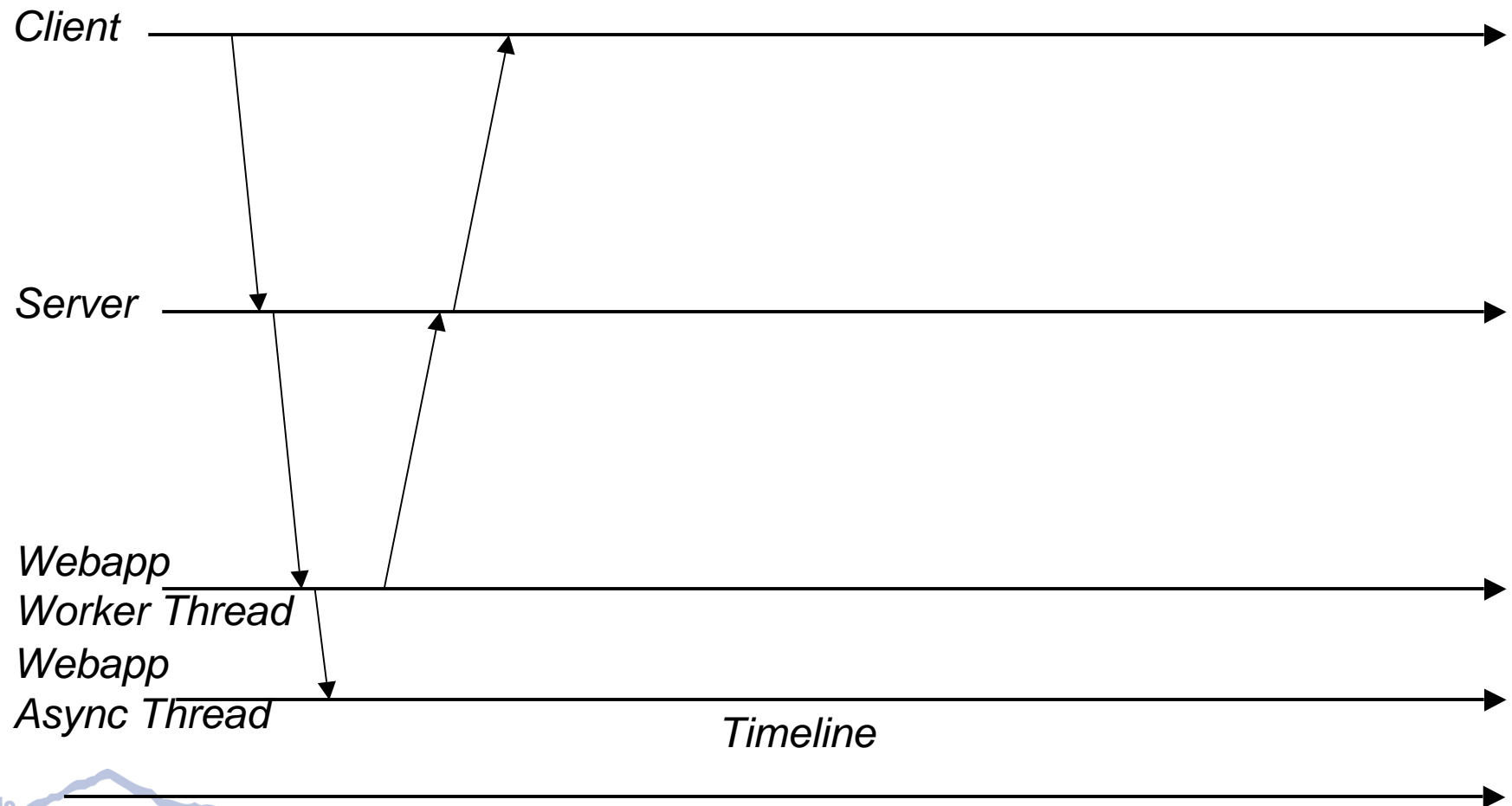
---

## 4. Event Complete



# How it Works

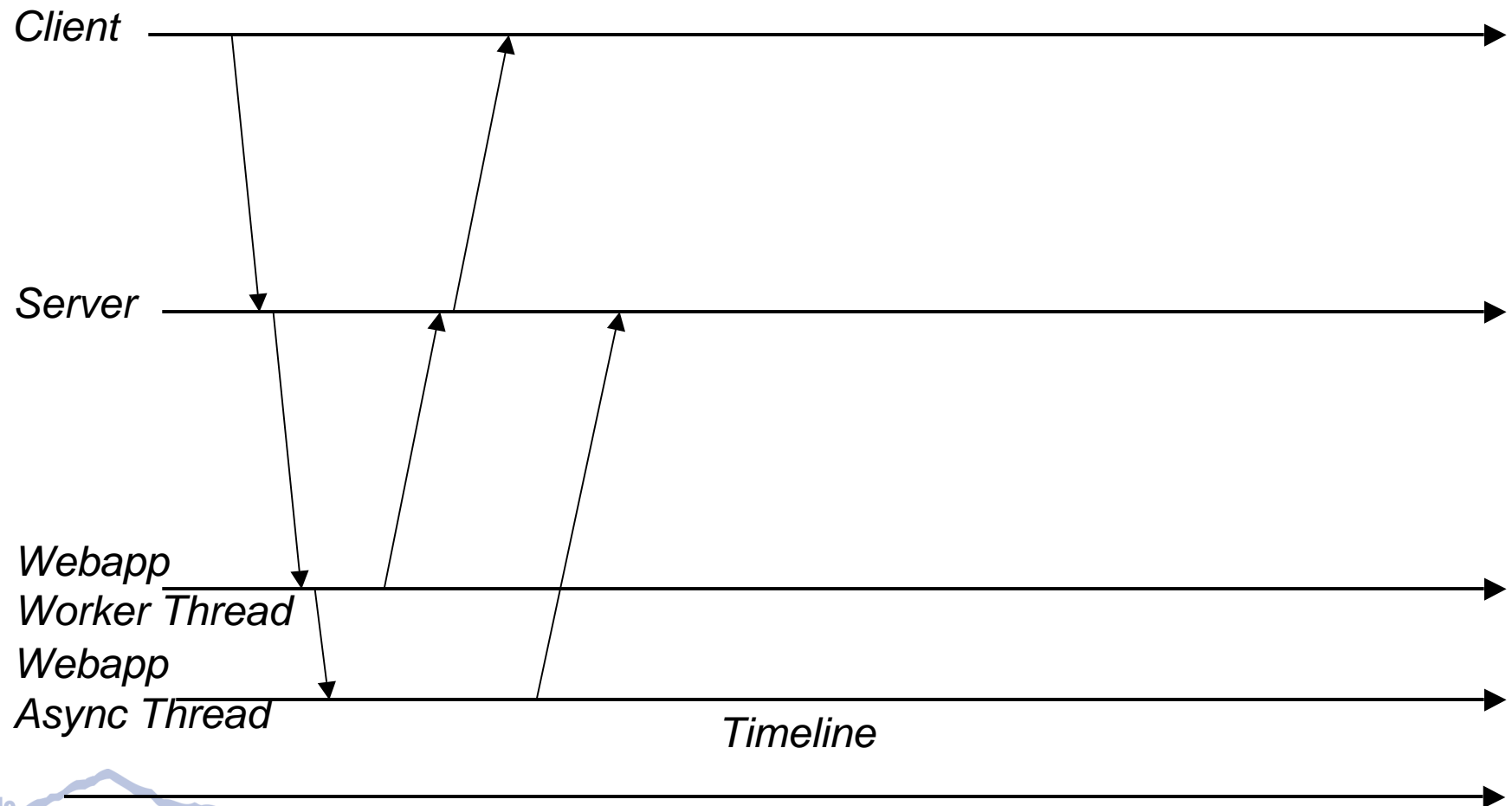
## 5. HTTP Response - 200 OK – Tx Enc: Chunked





# How it Works

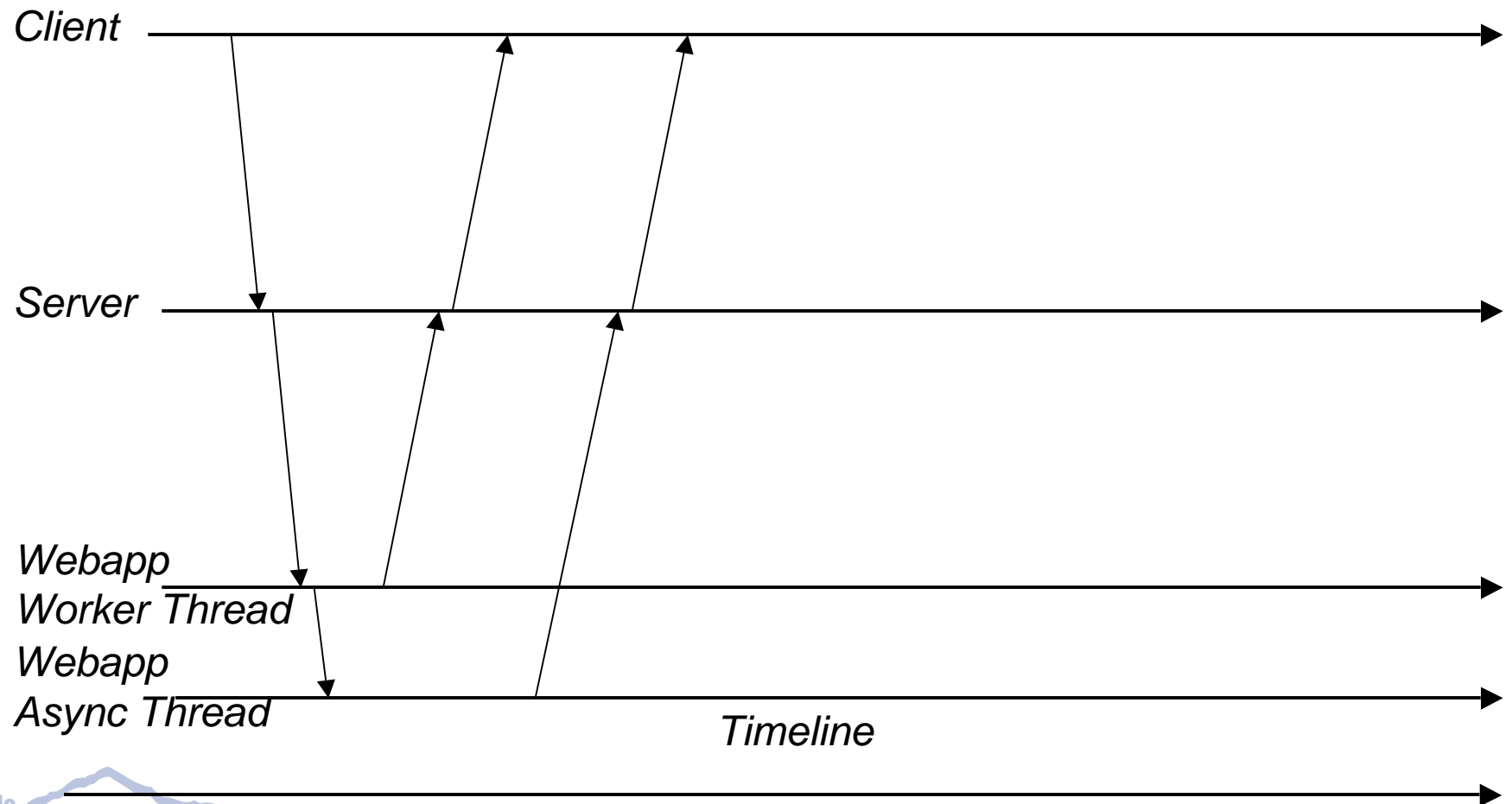
## 6. Server data push





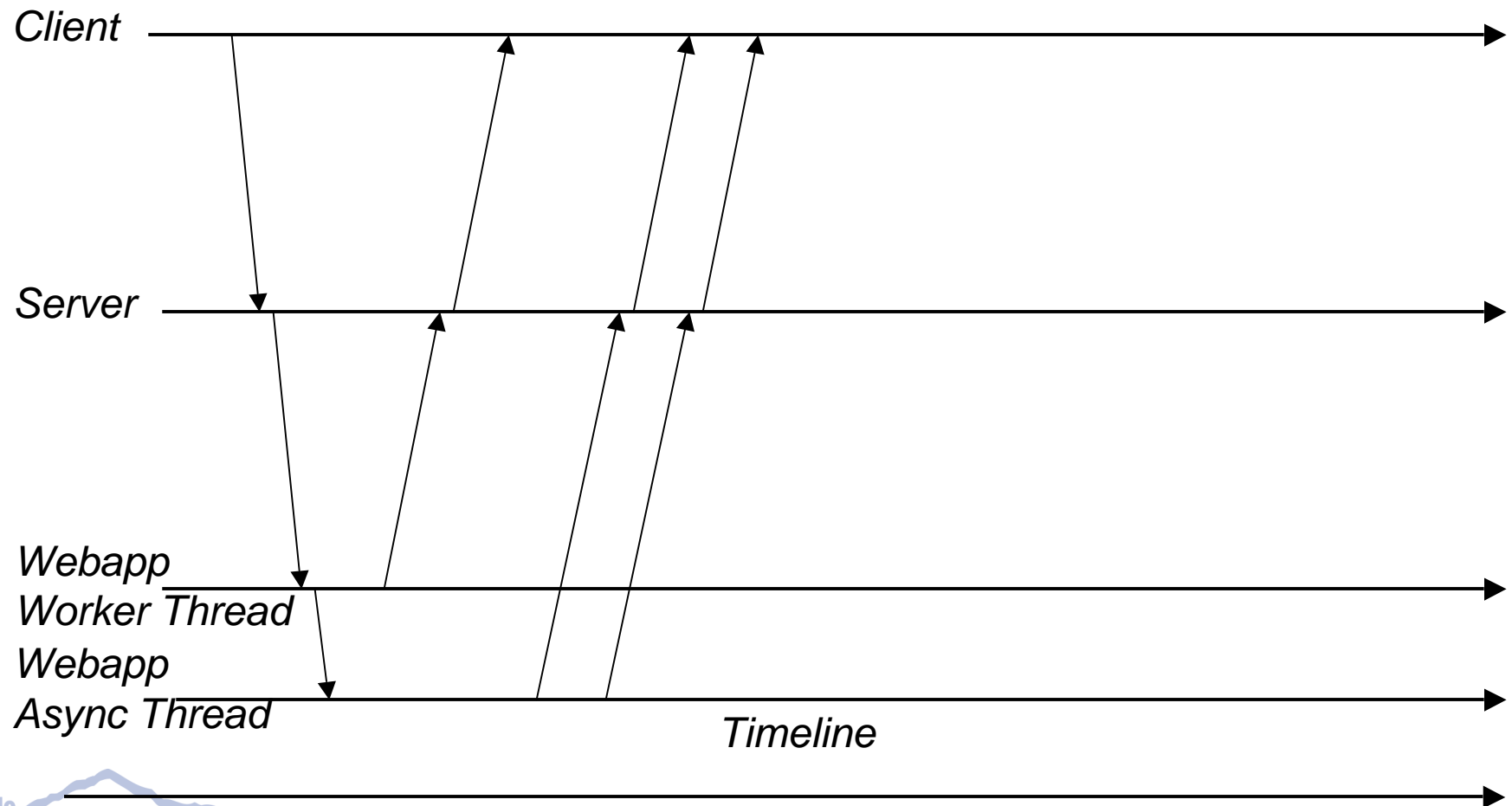
# How it Works

## 7. Encode and transfer



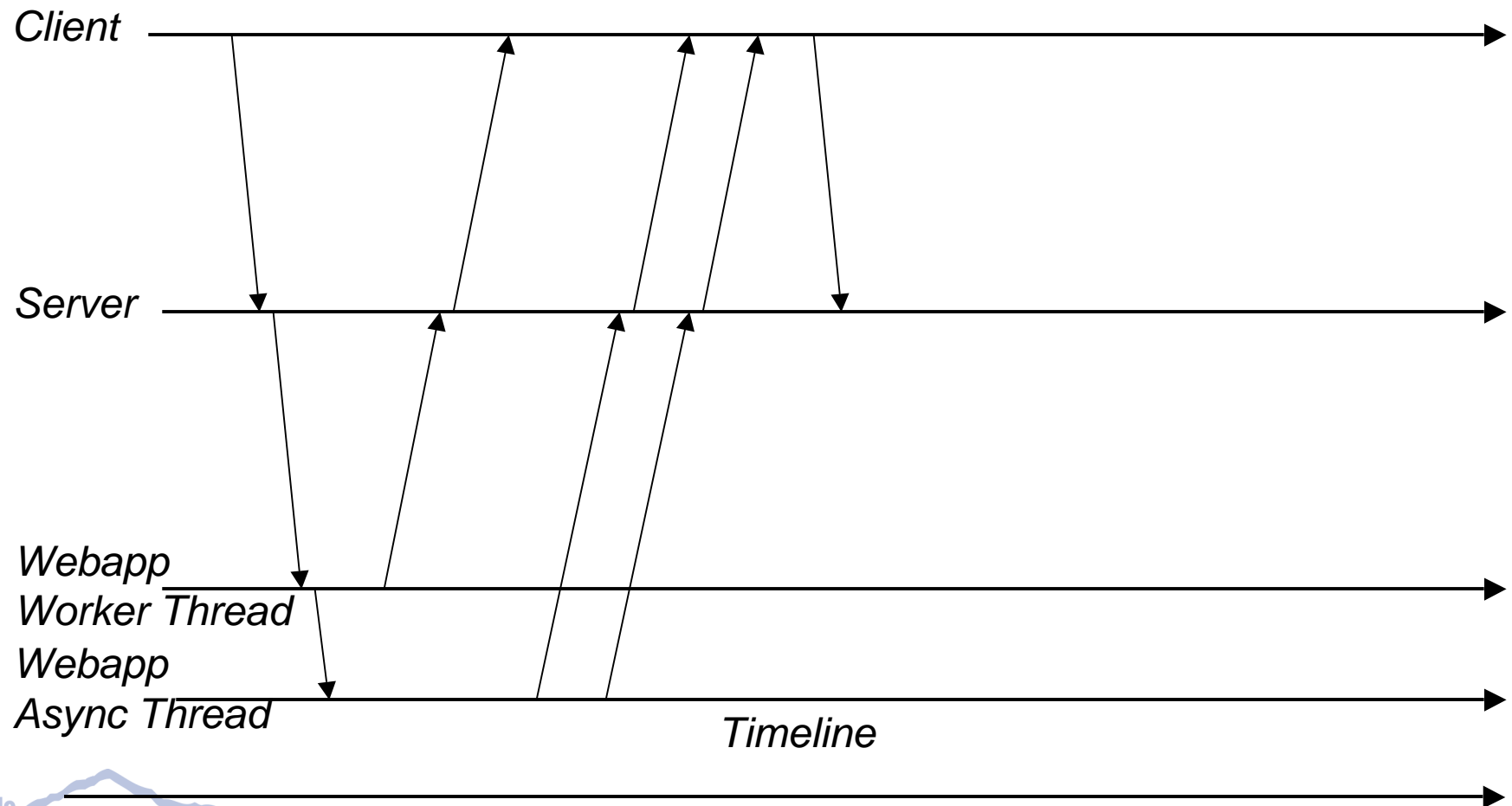
# How it Works

## 8. Server Push happens asynchronously



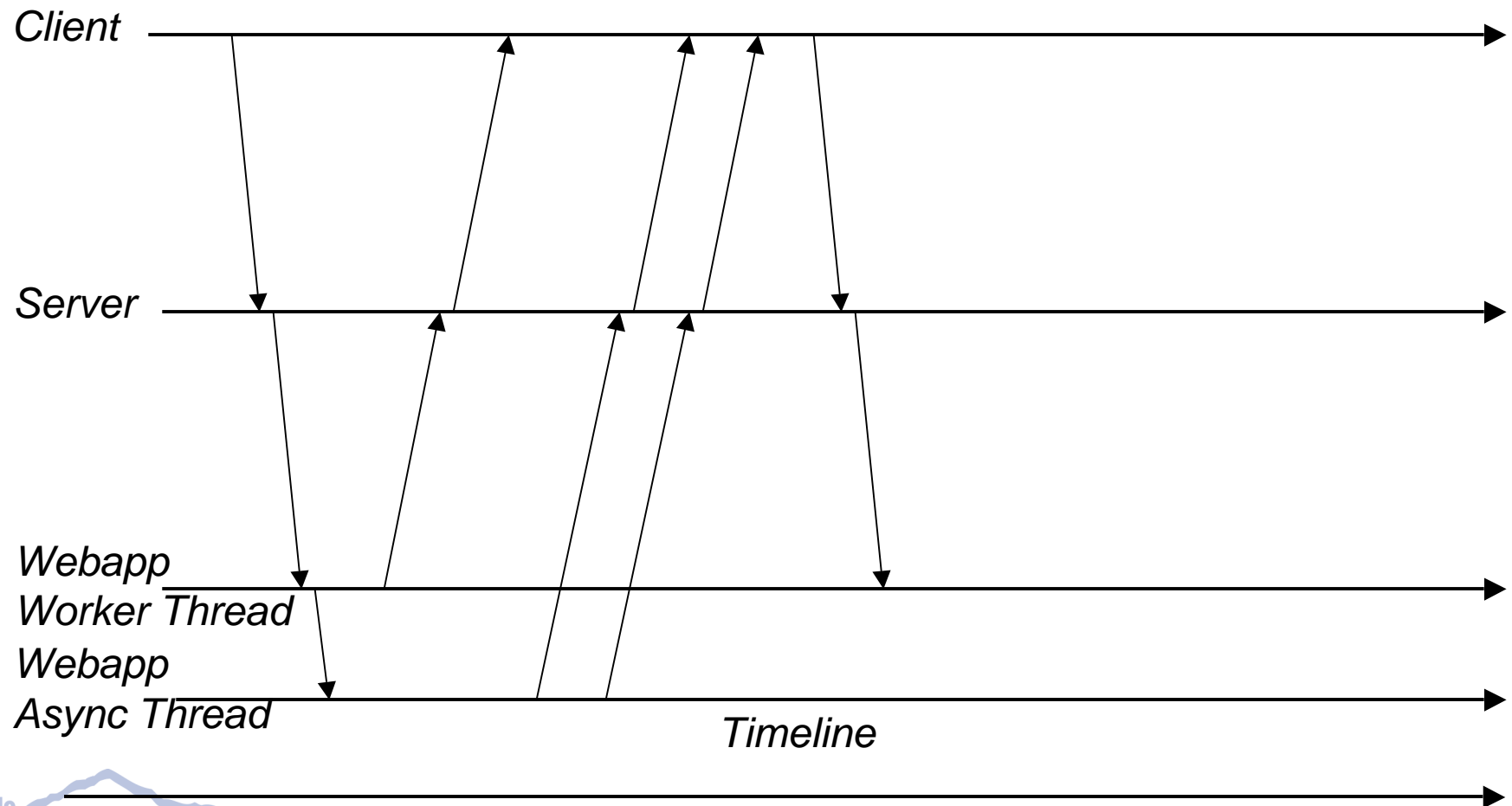
# How it Works

## 9. Client Push



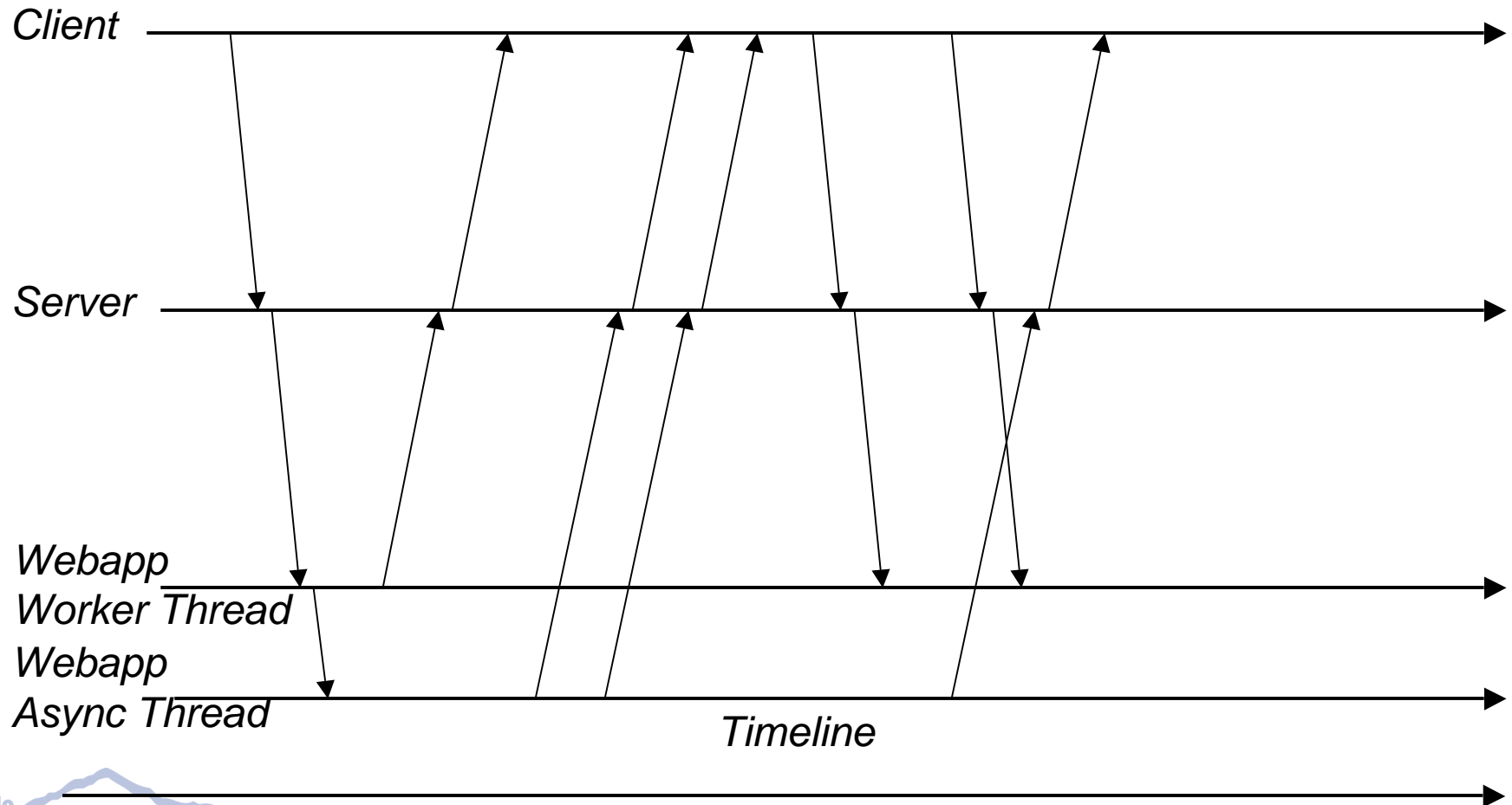
# How it Works

## 10. READ Event



# How it Works

## 11. Server Push and Client Push can happen at the same time





# Under the Hood

---

- CometProcessor
  - Extends HttpServlet to provide a fall back option should the connector not support Comet
- void event(CometEvent event)..
  - Types of Events:
    - BEGIN
    - READ
    - END
    - ERROR
- A request is a Comet request if the URL maps to a servlet that implements the CometProcessor interface



# Under the Hood

---

- CometFilter
  - Extends `javax.servlet.Filter` to provide fall back option should the Connector not support Comet
- `void doFilterEvent(CometEvent event, CometFilterChain chain) ...`
- Same mapping rules as CometProcessor





# Under the Hood

---

- BEGIN

- New connection created
- Request headers fully parsed and mapped to a servlet
- Partial body may or may not have been read
- Processed by a worker thread
- May proceed directly to a READ event



# Under the Hood

---

- READ

- New data available on the connection
- Processed by worker thread
- Read now to avoid repeated “read” events
- Read using servlet input stream  
`event.getHttpServletRequest().getInputStream()`



# Under the Hood

---

- END

- The request has naturally come to an end
- Server may shutdown, and needs to close connections
- Only if Connector shutdown prior to app, piggy back on `HttpServlet.destroy` for most apps instead

- ERROR

- When the connection times out or an error has occurred during invocation



# Under the Hood

---

- Events are “connection” centric
- Thrown based on IO events
- Reading should only be done when an event is thrown
- Writing can be done async
- Synchronize your write methods



# Gotcha's

---

- Configure Tomcat
  - APR or NIO connector

```
<Connector port="8080
  protocol="org.apache.coyote.http11.Http11NioProtocol"
  ...or...
  protocol="org.apache.coyote.http11.Http11AprProtocol«
```

- Otherwise Tomcat will invoke service(...)



# Gotcha's

---

- NIO – client disconnect is signaled by a READ with a -1 or EOFException by `InputStream.read()`
- The event lifecycle is not always what it appears to be
- While the API seems simple, it is pretty fragile
  - Easy to break if misused
  - Concurrency becomes a tricky issue



# Bidirectional Comet

---

- Tomcat supports bidirectional comet communication
  - Server can push data to the client
  - Client can push data to the server
- Timeout values can be set on a per-connection basis
- Unidirectional Comet means that only the server can push data to client



# Scalability?

---

- No more thread per connection or thread per request
- One asynch thread can handle writes for thousands of Comet connections





# Scalability?

---

- No overhead of request/response headers for each request
- No overhead for TCP setup/breakdown
- Memory overhead for open connections



# Examples of use

---

- Rich Clients
  - Mail
  - Maps
  - Online conferences
- Clients with need for server push
  - Stock tickers
  - Auction sites
  - Chat



# Limitations

---

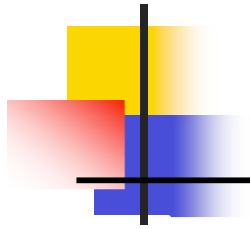
- Proxy Support
- Browser 2 connection limit
- JavaScript Support for Comet
  - Not really – no socket API on the browser
  - Currently mostly used for Async servlets
- Thick clients can benefit most
  - True socket API



# Future Improvements

---

- Non blocking reads
- Non blocking or buffered writes
- Bayeux protocol implementation
- Next Request – AJAX over Comet  
Keepalive Connections
- Comet through HTTP proxy(?)



# Demo Time

---

## Stock Ticker Demo & Java Code



# Build the demo

---

```
svn co --username open --password open
```

```
http://svn.hanik.com/svn/repos/filip/cometdemo
```

```
cd cometdemo/build
```

```
ant
```

→ cometdemo.war contains the demo code, including applet



# Looking at the code

---

- `service(...)` is not being used
  - Not invoked by Tomcat
- `event(...)` has replaced it for async invocation
- `HttpServletRequest` / `HttpServletResponse` can be used async
  - Use another thread(s) to manipulate input/output



# Building on top of Comet

---

- Please come and see my
- “What the Bayeux” talk
- Implementation of the Bayeux protocol
  - Built on top of Tomcat Comet





# Feedback

---

- Tomcat Dev List
- [fhanik@apache.org](mailto:fhanik@apache.org)
- <http://people.apache.org/~fhanik>
- Help and ideas are wanted