w:// webtide

geecon

Scaling Webapps With Servlet 3.1 Async I/O

geecon

Simone Bordet
sbordet@intalio.com

Who Am I

- **Simone Bordet**
  - sbordet@intalio.com
  - @simonebordet
- **Open Source Contributor**
  - Jetty, CometD, MX4J, Foxtrot, LiveTribe, JBoss, Larex
- **Lead Architect at Intalio/Webtide**
  - Jetty's HTTP/2, SPDY and HTTP client maintainer
- **CometD project leader**
  - Web messaging framework
- **JVM tuning expert**

Agenda

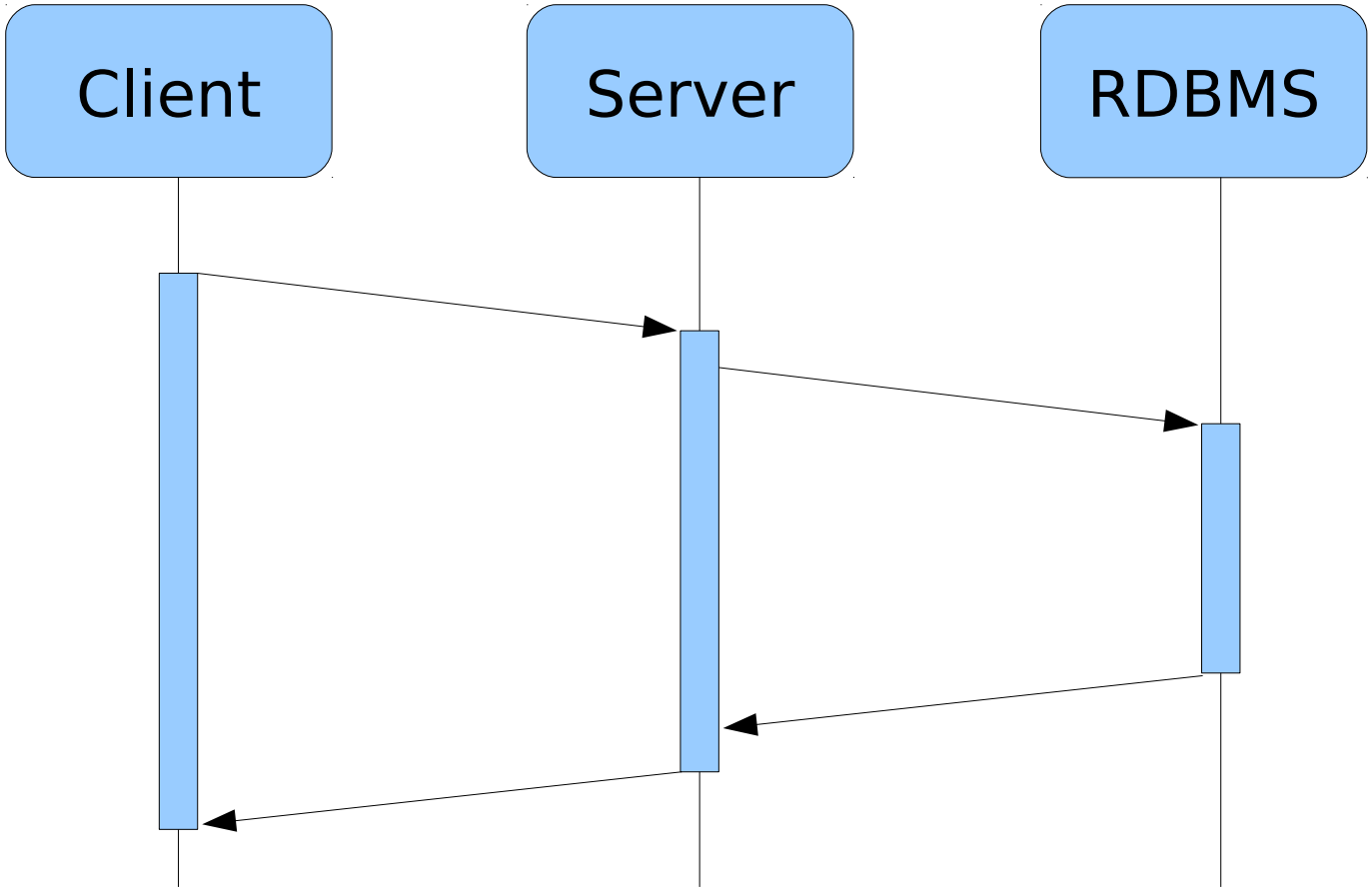
- Why Async ?
- Async Everywhere
- Async I/O API in the Servlet 3.1 Specification
- Correctly use the new Servlet Async I/O APIs

Simone Bordet
sbordet@intalio.com

Why Async ?

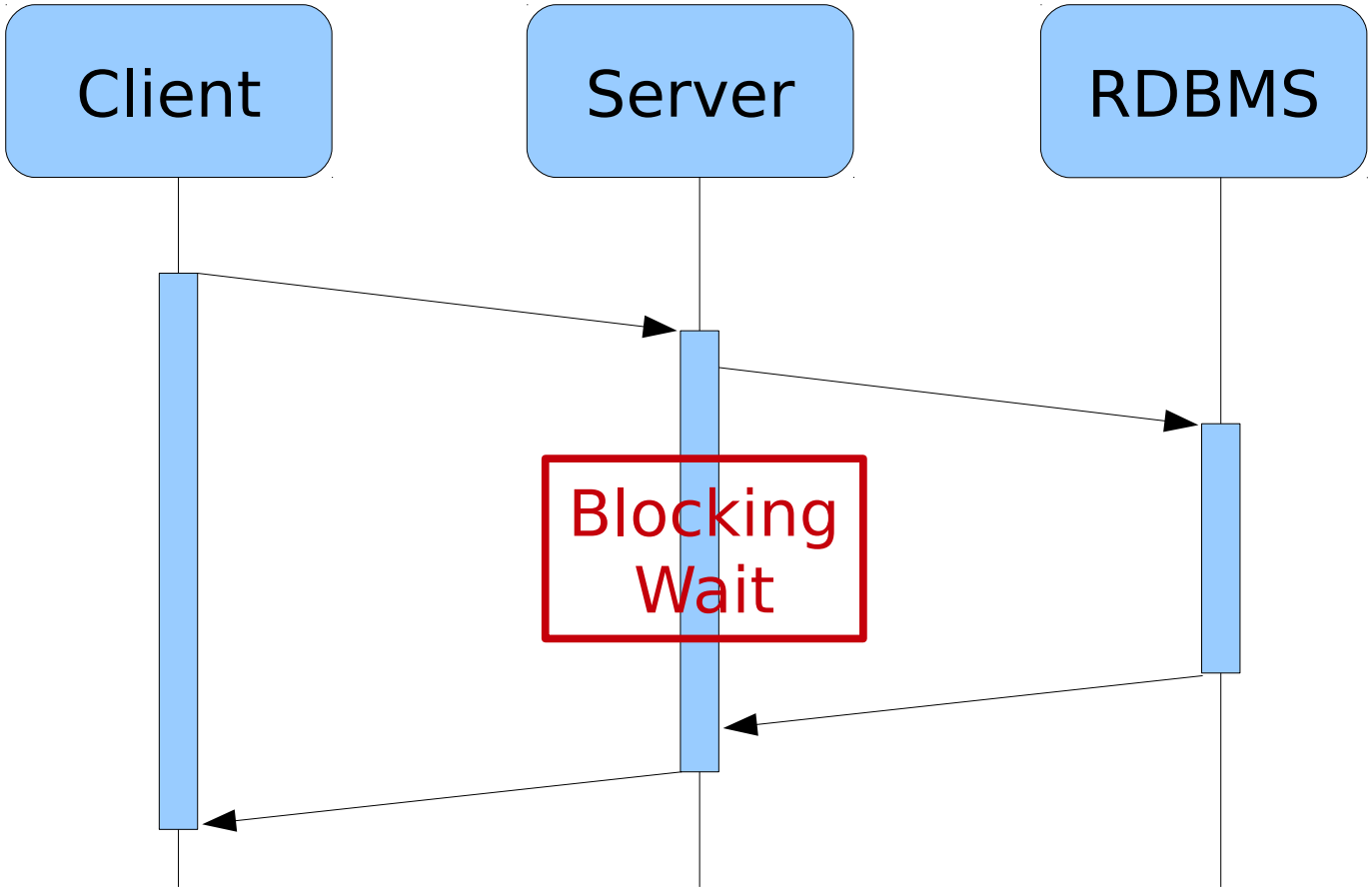
Simone Bordet
sbordet@intalio.com

Why Async ?



Simone Bordet
sbordet@intalio.com

Why Async ?



Simone Bordet
sbordet@intalio.com

Why Async ?

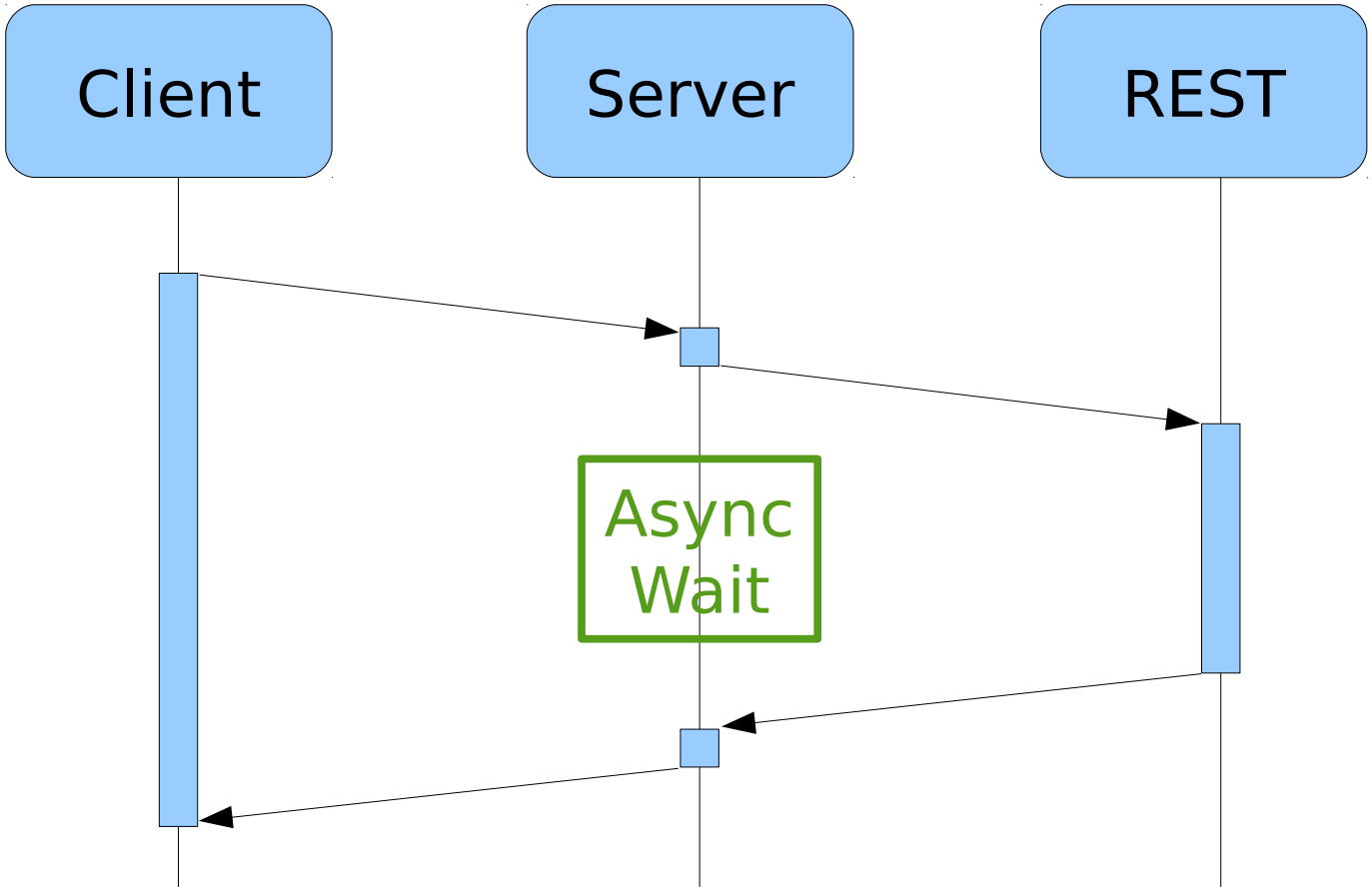
- **Why blocking waits are bad ?**

- **They consume resources**
  - Native Thread
  - Native Memory (thread stack: 1 MiB per thread)
  - OS scheduler data structures
  - ThreadLocals
  - Local variables in the stack frames
  - GC has to walk the stack frames

- **Blocking waits == $$$**

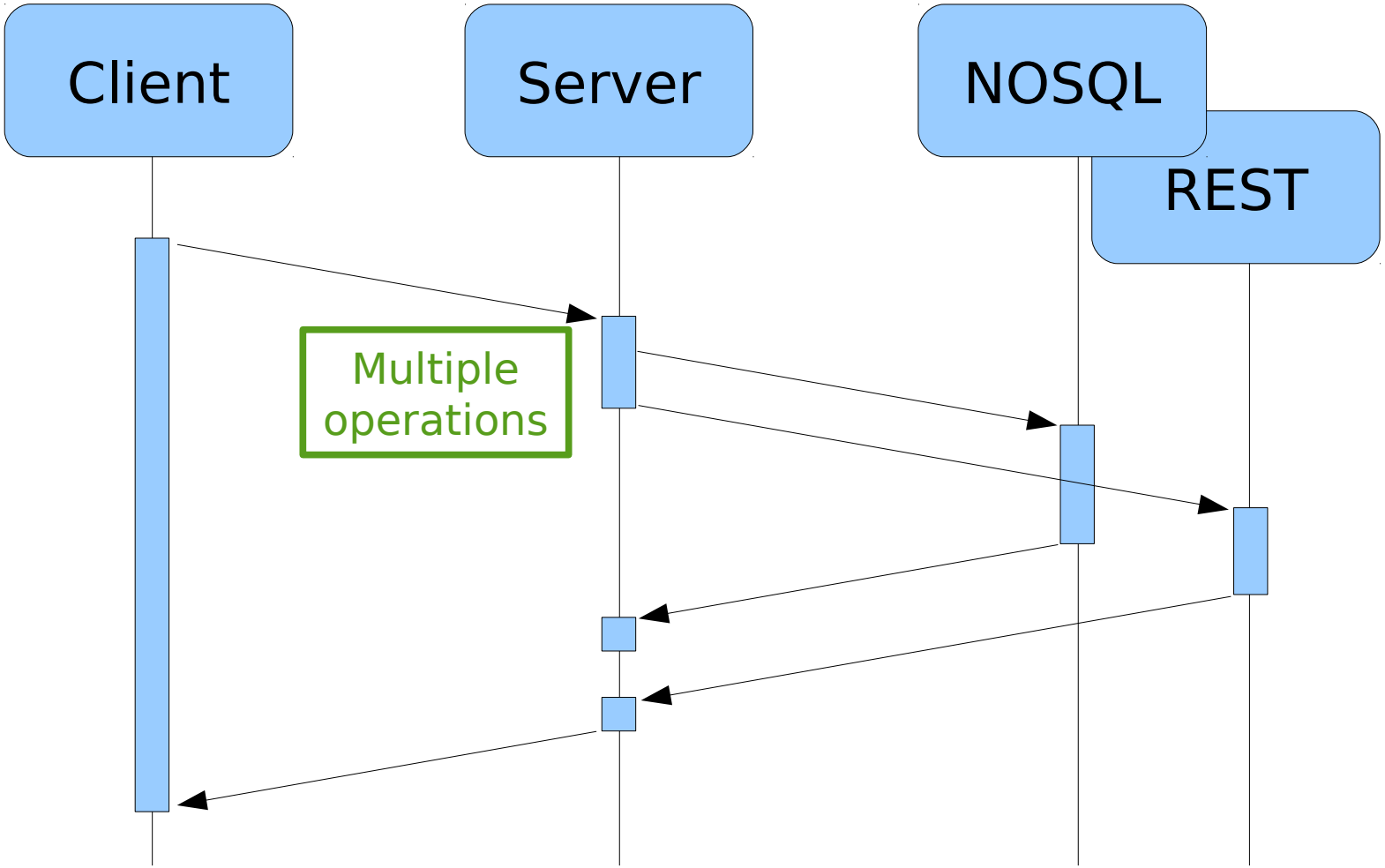
Simone Bordet
sbordet@intalio.com

Why Async ?



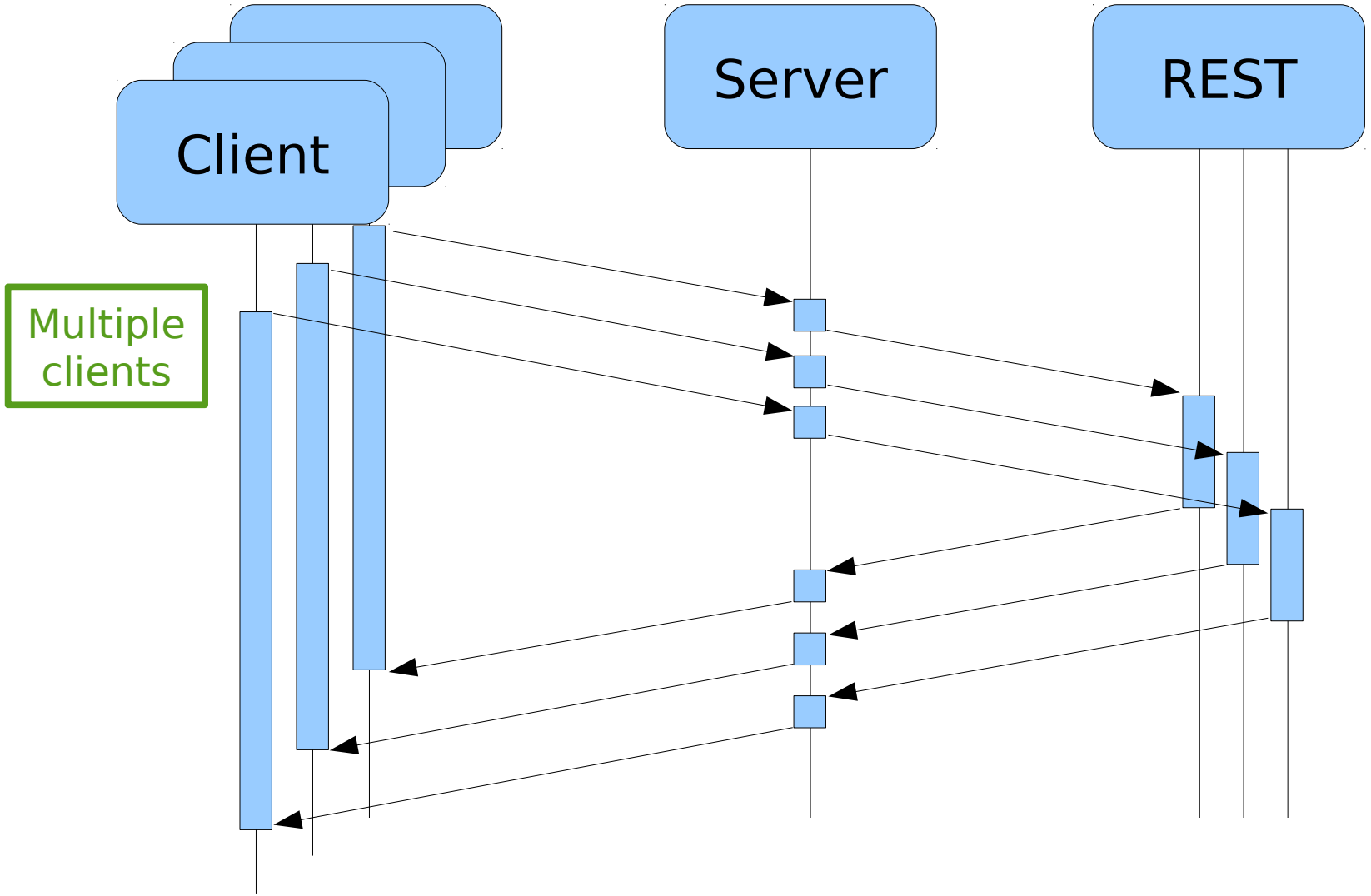
Simone Bordet
sbordet@intalio.com

Why Async ?



Simone Bordet
sbordet@intalio.com

Why Async ?



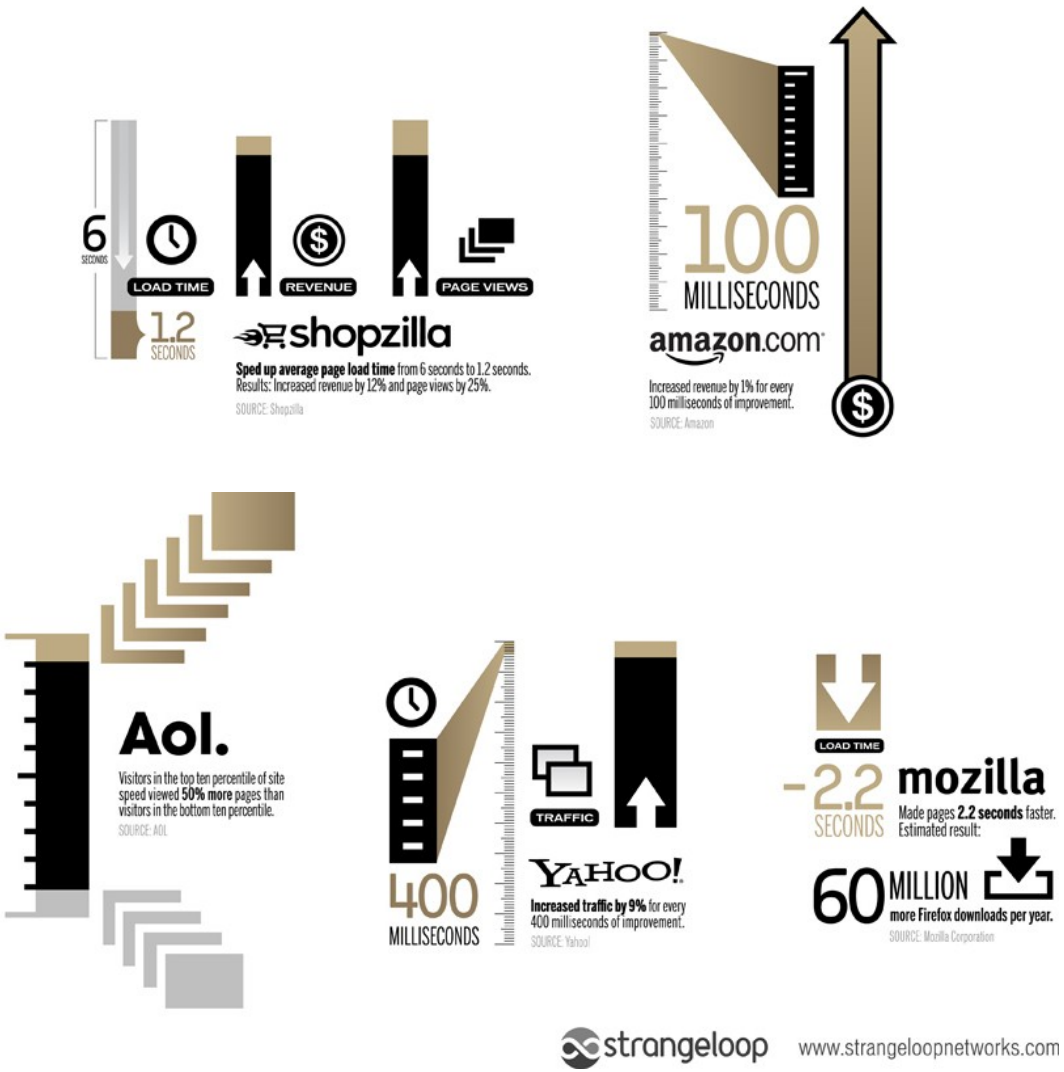
Simone Bordet
sbordet@intalio.com

Why Async ?

- **Improved latency**
  - By performing tasks concurrently with less resources
- **Better resource utilization**
  - The SAME thread can serve multiple clients
- **Async waits == $**
- **Async == Increased performance**
- **Increased performance == $$$**

Simone Bordet
sbordet@intalio.com

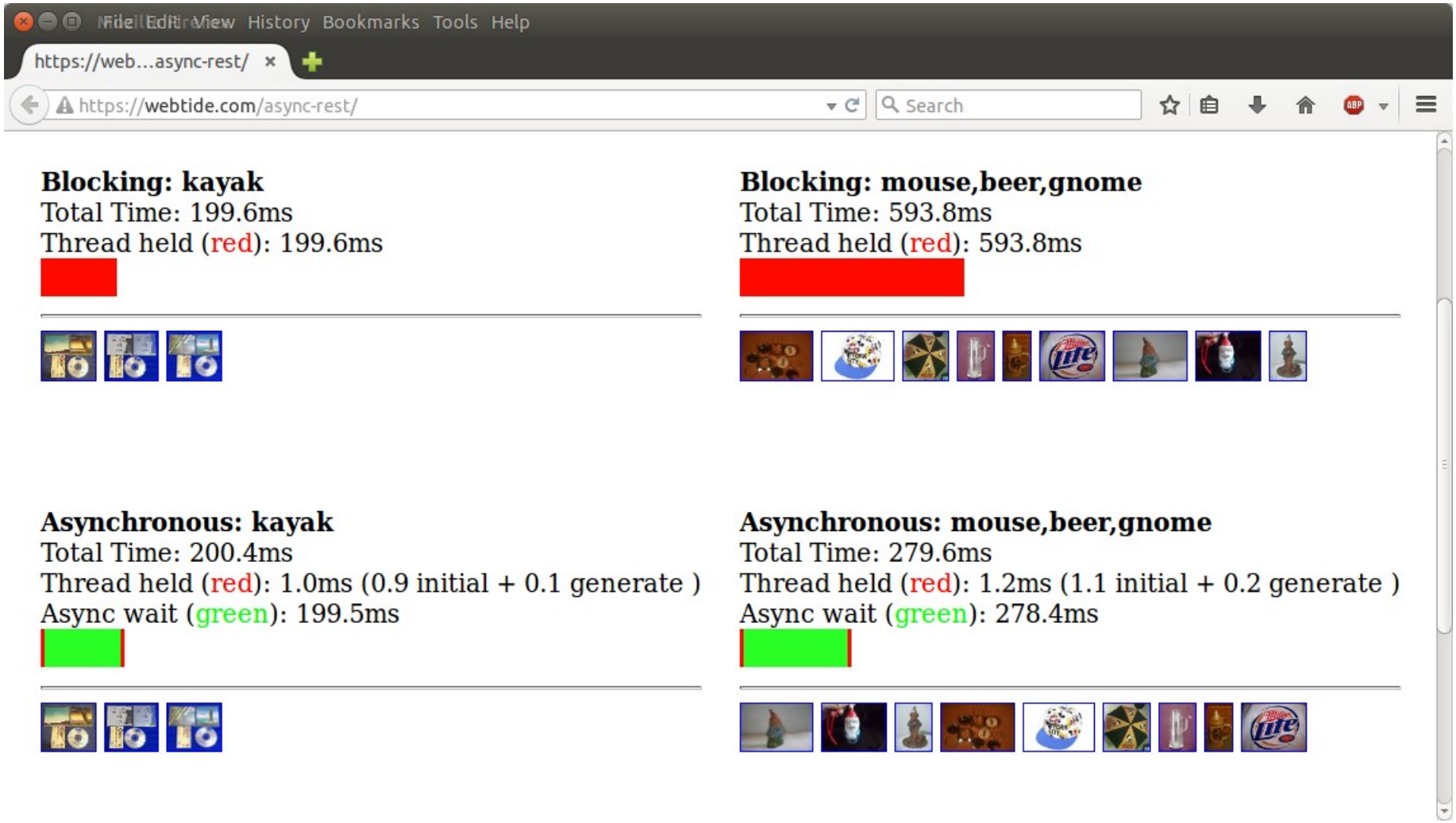
Why Async ?



Simone Bordet
sbordet@intalio.com

Demo

https://webtide.com/async-rest/



Simone Bordet
sbordet@intalio.com

Async Everywhere

Simone Bordet
sbordet@intalio.com

Async Everywhere

- **Until few years ago parallelism was not widespread**
  - Single core CPUs
- **Modern hardware has gone multi-core**
  - We want to be able to use those cores
- **Future hardware will have even more cores**
- **We need to leverage hw parallelism in software**

Simone Bordet
sbordet@intalio.com

Async Everywhere

- **Async programming is not new**
  - Spawning a thread is a primitive in OS since day one
- **Async programming is difficult...**
  - ... to write: callback hell, synchronization
  - ... to read: our brain reads from top to bottom
- **New APIs are born**
  - epoll, promises, actors, channels, etc.
- **New products are born**
  - Nginx vs Apache, etc.

Simone Bordet
sbordet@intalio.com

Async Everywhere

- **Example: Java 8 CompletableFuture**

```
CompletableFuture
   .supplyAsync(() -> fetchQueryFromREST(uri))
   .thenApplyAsync(sql -> fetchDataFromNOSQL(sql))
   .thenAccept(data -> updateUI(data));
```

- **We want a way to read code as if it is sequential**
  - But in fact it is asynchronous
- **You have to marry the API model**
  - Always use the model, no exceptions

Simone Bordet
sbordet@intalio.com

Async Everywhere

- Servlet 3.0: async processing of response

```
void doGet(request, response)
{
    OutputStream out = response.getOutputStream();
    AsyncContext ctx = request.startAsync();
    doAsyncREST(request).thenAccept(json -> {
            out.write(json);
            ctx.complete();
        });
}
```


Simone Bordet
sbordet@intalio.com

Async Everywhere

- **Two requirements for an async Servlet**
- **First requirement: doAsyncREST()**
  - Must use async libraries to do its work
  - Must return a CompletableFuture
    - Or take a callback parameter (or something similar)
- **Second requirement: thenAccept() lambda**
  - Must use async libraries

Simone Bordet
sbordet@intalio.com

Async Everywhere

- ✗ JDBC
- ✗ InputStream – OutputStream
- ✔ Asynchronous I/O (NIO2: files, sockets)
- ✔ REST
- ✔ Call other Servlets

Simone Bordet
sbordet@intalio.com

Async Everywhere

- Servlet 3.0: no async I/O

```
void doGet(request, response)
{
    OutputStream out = response.getOutputStream();
    AsyncContext ctx = request.startAsync();
    doAsyncREST(request).thenAccept(json -> {
            out.write(json);  Blocking!
            ctx.complete();
        });
}
```


Simone Bordet
sbordet@intalio.com

Async Everywhere

- Finally! That's why Servlet 3.1 introduced async I/O
- What API to use then ?
- Use `java.util.concurrent.Future` ?

```
Future<Integer> f = out.write(json);
int written = f.get().intValue();
```


Simone Bordet
sbordet@intalio.com

Async Everywhere

- Finally! That's why Servlet 3.1 introduced async I/O !
- What API to use then ?
- Use `java.util.concurrent.Future` ?

```
Future<Integer> f = channel.write(json);
int written = f.get().intValue();
```

This is just delayed blocking, not async

Simone Bordet
sbordet@intalio.com

Async Everywhere

- Use NIO2 ?

```
out.write(json, null, new CompletionHandler()
{
    void completed(Integer written, Object obj)
    {
        ctx.complete();
    }

    void failed(Throwable x, Object obj) {
    }
}
```


Simone Bordet
sbordet@intalio.com

Async Everywhere

- Use NIO2 ?

```
                        Allocation      Not a lambda
out.write(json, null, new CompletionHandler()
{    Not thread efficient
     void completed(Integer written, Object obj)
     {
          ctx.complete();
     }

     void failed(Throwable x, Object obj) {
     }
}
```

Prone to stack overflow (resolved by further thread dispatching)

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Servlet 3.1 async write I/O APIs**

```
void doGet(request, response) {
    ServletOutputStream out = response.getOutputStream();
    AsyncContext ctx = request.startAsync();
    out.setWriteListener(new WriteListener() {
        void onWritePossible() {
            while (out.isReady()) {
                byte[] buffer = readFromSomeWhere();
                if (buffer != null)
                    out.write(buffer);
                else
                    ctx.complete();
}}});}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- Servlet 3.1 async write I/O APIs

```
void doGet(request, response) {
    ServletOutputStream out = response.getOutputStream();
    AsyncContext ctx = request.startAsync();
    out.setWriteListener(new WriteListener() {
        void onWritePossible() {
Iterative   while (out.isReady()) {
                byte[] buffer = readFromSomeWhere();
                if (buffer != null)
                    out.write(buffer);   Async write
                else
                    ctx.complete(); break;
}}});}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Normal writes (fast connection)**
  - Iterative writes in the same thread
  - No thread dispatch
  - No stack overflow
  - No allocation
- **Incomplete writes (slow connection)**
  - Some iterative write, then **`isReady()`** returns false
  - When ready again, new thread calls **`onWritePossible()`**
- **Got it ! Is it really this simple ?**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #1**

```
void onWritePossible()
{
    out.write(allContent);
    logger.debug("written: {}", out.isReady());
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #1**

```
void onWritePossible()
{
    out.write(allContent);
    logger.debug("written: {}", out.isReady());
}
```

When isReady() == false onWritePossible() will be called again !

- **Content is written twice !**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #2**

```
void onWritePossible()
{
    if (dataAvailable() && out.isReady())
        out.write(getData());
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #2**

```
void onWritePossible()
{
    if (dataAvailable() && out.isReady())
        out.write(getData());
}
```

The order of the expressions in the if statement is significant !

- **If `dataAvailable()` returns false, `isReady()` is not called and `onWritePossible()` will never be called**

- **Switching the if expressions changes the semantic**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #3**

```
void onWritePossible()
{
    if (out.isReady()) {
        out.write(“<h1>Async</h1>”);
        out.write(“<p>Content</p>”);
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #3**

```
void onWritePossible()
{
    if (out.isReady()) {
        out.write("<h1>Async</h1>");
        out.write("<p>Content</p>");
    }
}
```

The first write may be pending !

- **You HAVE to use `isReady()` after each write**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #4**

```
void onWritePossible()
{
    if (out.isReady()) {
        out.write("<h1>Async</h1>");
        if (out.isReady())
            out.write("<p>Content</p>");
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- Bug #4

```
void onWritePossible()
{
    if (out.isReady()) {
        out.write("<h1>Async</h1>");
        if (out.isReady())
            out.write("<p>Content</p>");
    }
}
```

Content written multiple times !

- Don't misuse isReady()

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #5**

```
void onWritePossible()
{
    int read = file.read(buffer);
    while (out.isReady()) {
        out.write(buffer);
        if (file.read(buffer) < 0)
            ctx.complete(); return;
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #5**

```
void onWritePossible()
{
    int read = file.read(buffer);
    while (out.isReady()) {
        out.write(buffer);
        if (file.read(buffer) < 0)
            ctx.complete(); return;
    }
}
```

Write may be pending: you don't own the buffer

- **The buffer is overwritten by the new read**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #6**

```
void onWritePossible()
{
    try {
        ...
    } catch (Exception x) {
        response.sendError(500);
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #6**

```
void onWritePossible()
{
    try {
        ...
    } catch (Exception x) {
        response.sendError(500);
    }
}
```

sendError() is a blocking API !

- **Cannot mix async I/O and blocking I/O**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **When `ServletOutputStream` is put into async mode:**
  - Quacks like a stream
  - Walks like a stream
  - But it's NOT a stream !

- **Broken semantic:**
  - OutputStream.write() is expected to be blocking !
  - ServletOutputStream.isReady(): a query with side effects !

- **Breaking semantic consequences:**
  - Filters and wrappers are broken !
  - Cannot tell whether the stream is async or not

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Servlet 3.1 async read I/O APIs**

```
void doPost(request, response)
{
    AsyncContext ctx = request.startAsync();
    ServletInputStream in = request.getInputStream();
    in.setReadListener(new ReadListener() {
        void onDataAvailable() {
            while (in.isReady() && !in.isFinished()) {
                int len = in.read(buffer);
                if (len > 0)
                    store(buffer); // Blocking API
                if (in.isFinished())
                    ctx.complete(); return;
}}});}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- Servlet 3.1 async read I/O APIs

```
void doPost(request, response)
{
    AsyncContext ctx = request.startAsync();
    ServletInputStream in = request.getInputStream();
    in.setReadListener(new ReadListener() {
        void onDataAvailable() {
            while (in.isReady() && !in.isFinished()) {
                int len = input.read(buffer);
                if (len > 0)
                    store(buffer); // Blocking API
                if (in.isFinished())
                    ctx.complete(); return;
}}});}
```

Iterative

Async read

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Normal reads (fast connection)**
  - Iterative reads in the same thread
  - No thread dispatch
  - No stack overflow
  - No allocation
- **Incomplete reads (slow connection)**
  - Some iterative reads, then **`isReady()`** returns false
  - When ready again, new thread calls **`onDataAvailable()`**
- **Got it ! Is it really this simple ?**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #1**

```
void onDataAvailable() {
    while (in.isReady()) {
        in.read(buffer);
        store(buffer);
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #1**

```
void onDataAvailable() {
    while (in.isReady()) {
        in.read(buffer);
        store(buffer);
    }
}
```

isReady() returns true at EOF

- **Infinite loop**
  - The loop must call also **in.isFinished()**

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #2**

```
void onDataAvailable() {
    while (in.isReady() && !in.isFinished()) {
        int read = in.read(buffer);
        store(buffer, new Callback() {
            void done() {
                onDataAvailable();
            }
        });
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #2**

```
void onDataAvailable() {
    while (in.isReady() && !in.isFinished()) {
        int read = in.read(buffer);
        store(buffer, new Callback() {
            void done() {
                onDataAvailable();
            }
        });
    }
}
```

Stack overflow !

- **Stack overflows for fast connections**
- **Mixing different async models requires more careful coding**
  - Jetty's **`IteratingCallback`** is one solution

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #3**

```
void onDataAvailable() {
    String id = request.getParameter(“id”);
    ...
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Bug #3**

```
void onDataAvailable() {
    String id = request.getParameter("id");
    ...
}
```

Blocking API !

- **Same with `request.getParts()`**
  - Multipart APIs

Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

Echo Servlet Blocking I/O

```
public class SyncServlet extends HttpServlet
{
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        byte[] buffer = new byte[BUFFER_SIZE];
        while (true)
        {
            int read = request.getInputStream().read(buffer, 0, BUFFER_SIZE);
            if (read < 0)
                break;
            response.getOutputStream().write(buffer, 0, read);
        }
    }
}
```

Echo Servlet Async I/O

```
public static class AsyncServlet extends HttpServlet
{
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        AsyncContext asyncContext = request.startAsync(request, response);
        asyncContext.setTimeout(0);
        Echoer echoer = new Echoer(asyncContext);
        request.getInputStream().setReadListener(echoer);
        response.getOutputStream().setWriteListener(echoer);
    }

    private class Echoer implements ReadListener, WriteListener
    {
        private final byte[] buffer = new byte[BUFFER_SIZE];
        private final AsyncContext asyncContext;
        private final ServletInputStream input;
        private final ServletOutputStream output;
        private boolean complete;

        private Echoer(AsyncContext asyncContext) throws IOException
        {
            this.asyncContext = asyncContext;
            this.input = asyncContext.getRequest().getInputStream();
            this.output = asyncContext.getResponse().getOutputStream();
        }

        @Override
        public void onDataAvailable() throws IOException
        {
            while (input.isReady())
            {
                int read = input.read(buffer);

                output.write(buffer, 0, read);

                if (!output.isReady())
                    return;
            }

            if (input.isFinished())
            {
                complete = true;
                asyncContext.complete();
            }
        }

        @Override
        public void onAllDataRead() throws IOException
        {
        }

        @Override
        public void onWritePossible() throws IOException
        {
            if (input.isFinished())
            {
                if (!complete)
                    asyncContext.complete();
            }
            else
            {
                onDataAvailable();
            }
        }

        @Override
        public void onError(Throwable failure)
        {
            failure.printStackTrace();
        }
    }
}
```


Simone Bordet
sbordet@intalio.com

Servlet 3.1 Async I/O

- **Async I/O Echo Servlet**
- **Pros**
  - Async
- **Cons**
  - 4x-6x more code
  - Requires a state machine (a simple boolean, but still)
  - More difficult buffer handling
  - Complexity, maintainability, etc.

Simone Bordet
sbordet@intalio.com

Conclusions

Simone Bordet
sbordet@intalio.com

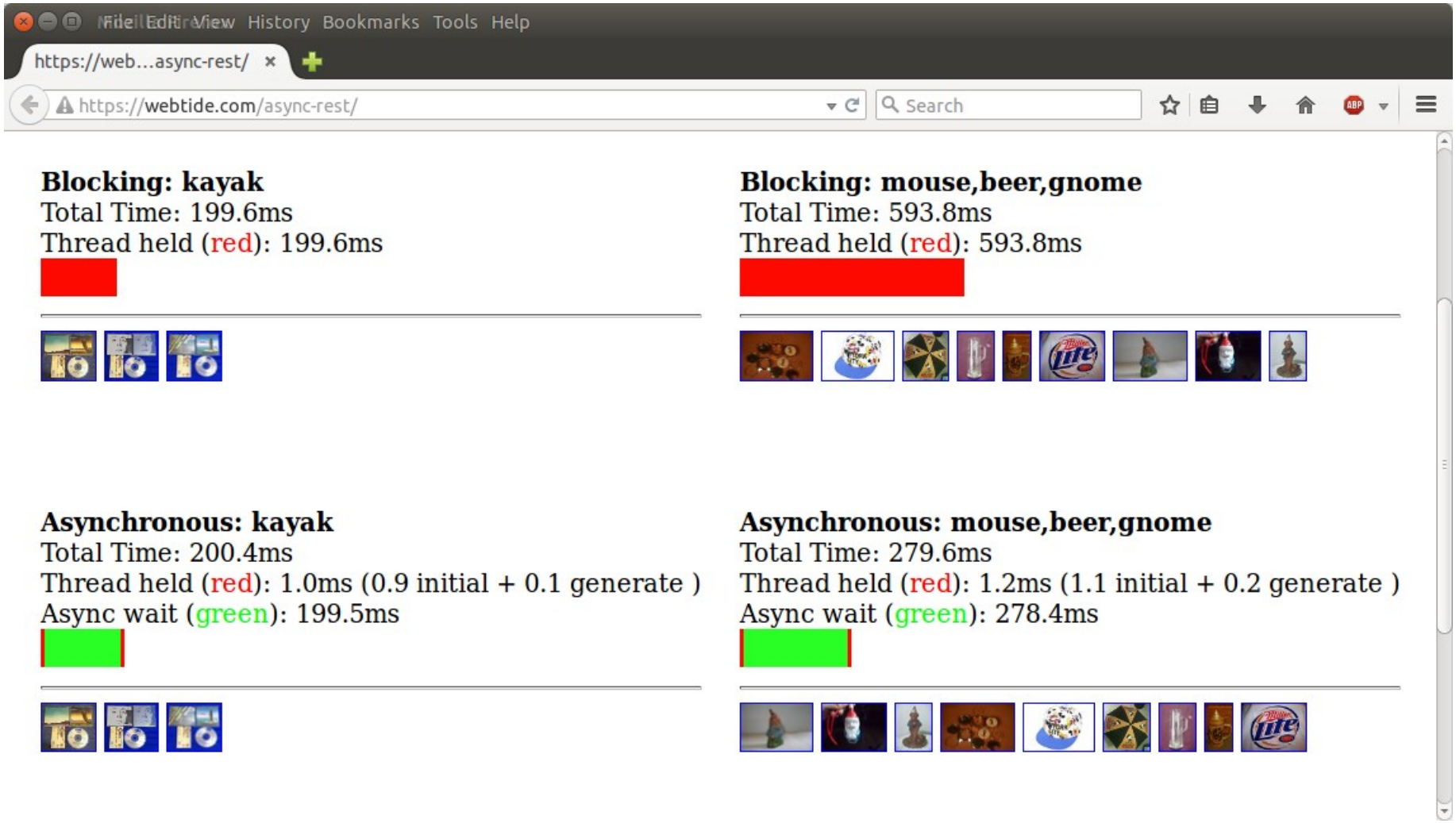
Servlet 3.1 Async I/O

- **Blocking code has a reason to exist**
  - And will continue to exist
- **Is the async I/O complexity worth it ?**
  - For a class of applications, YES !
  - Improved scalability
  - Decreased latency
  - Makes you more money

Simone Bordet
sbordet@intalio.com

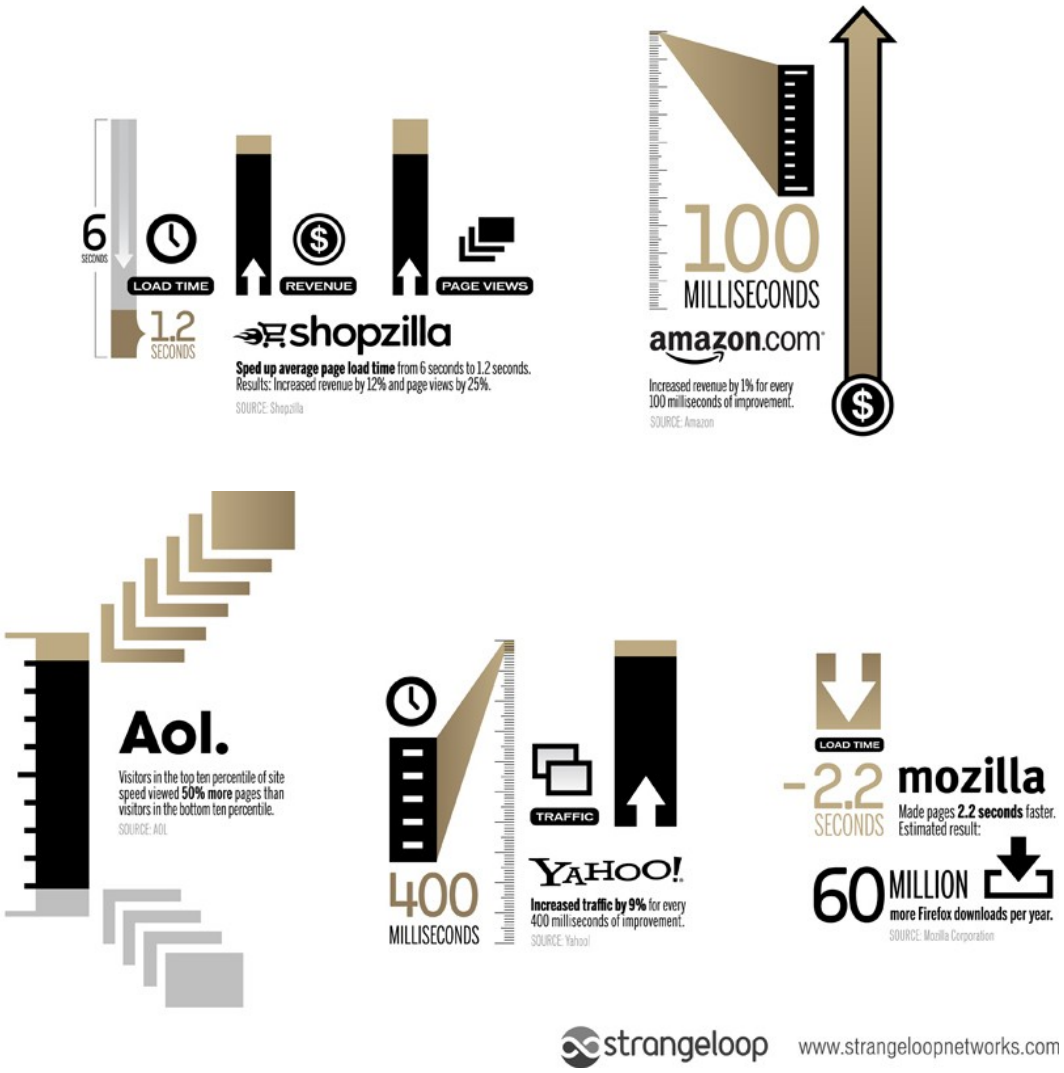
Takeaway #1

https://webtide.com/async-rest/



Simone Bordet
sbordet@intalio.com

Takeaway #2



Simone Bordet
sbordet@intalio.com

Questions
&
Answers

Simone Bordet
sbordet@intalio.com