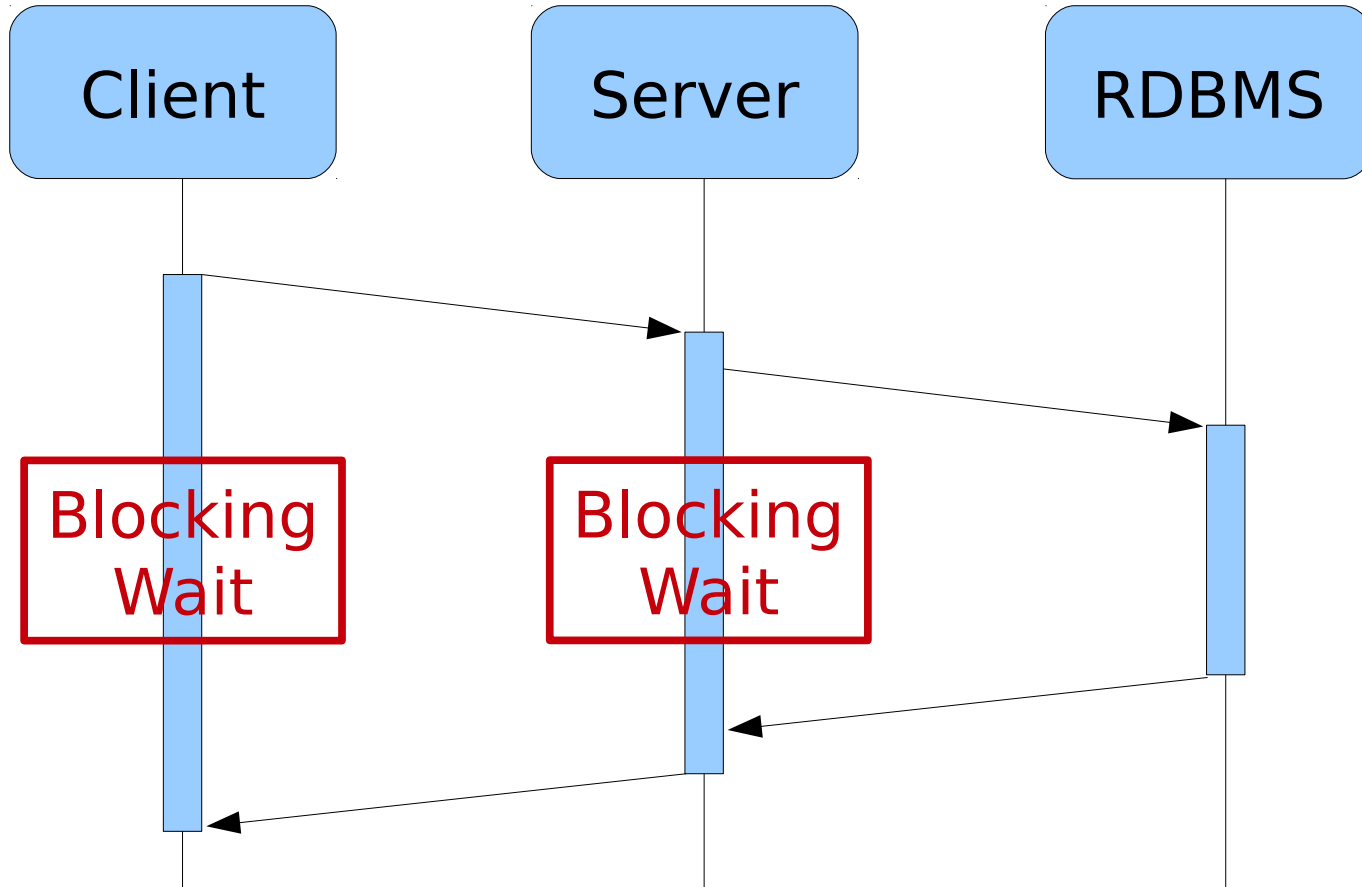
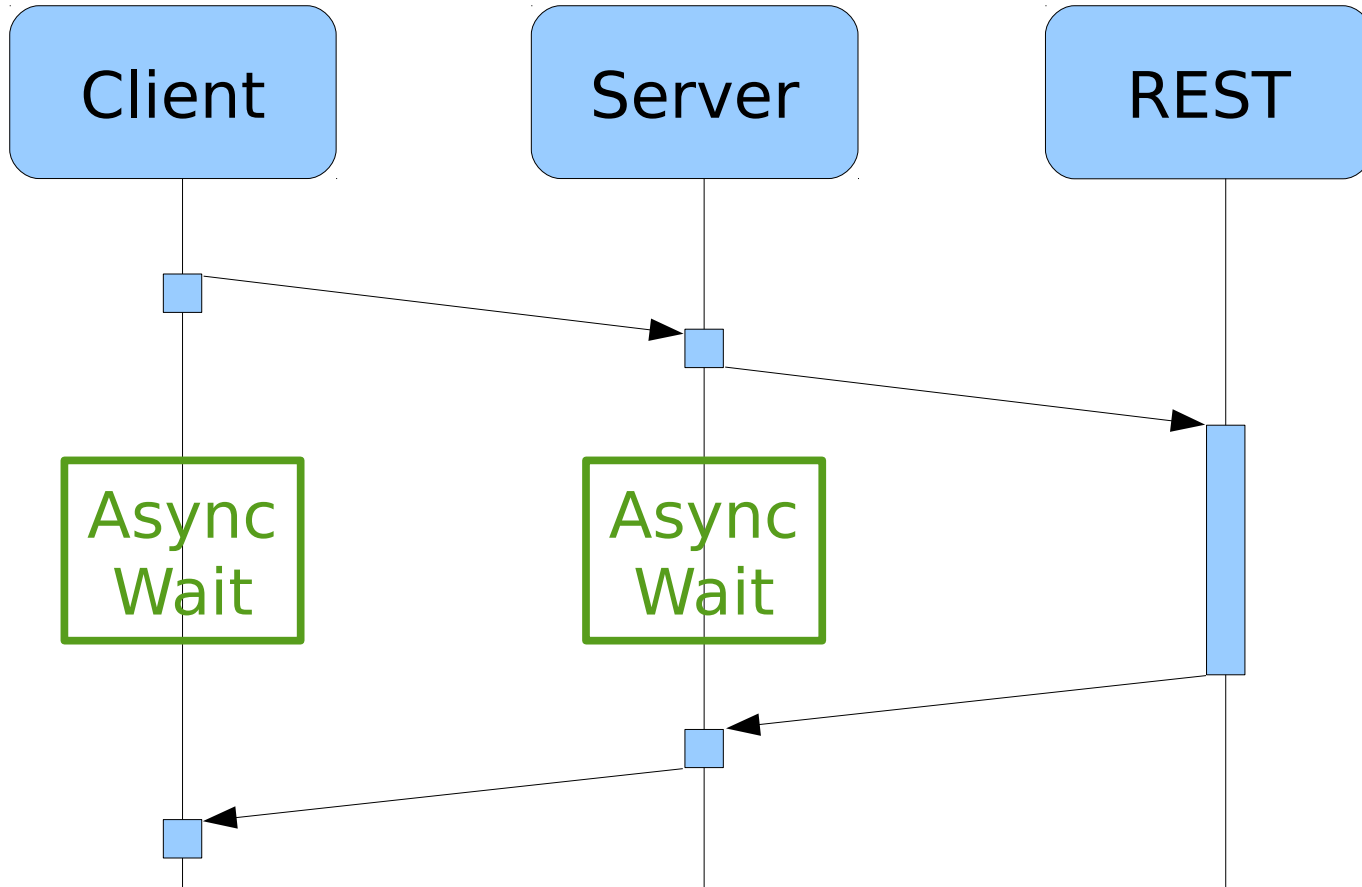


ApiConf

Reactive APIs





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

The screenshot shows a web browser window with the URL <https://webtide.com/async-rest/>. The page content is divided into four quadrants, each showing performance metrics for a specific operation. The top-left quadrant shows 'Blocking: kayak' with a total time of 199.6ms and a thread held for 199.6ms, accompanied by a red bar. The top-right quadrant shows 'Blocking: mouse,beer,gnome' with a total time of 593.8ms and a thread held for 593.8ms, accompanied by a red bar. The bottom-left quadrant shows 'Asynchronous: kayak' with a total time of 200.4ms, a thread held for 1.0ms (0.9 initial + 0.1 generate), and an async wait of 199.5ms, accompanied by a green bar. The bottom-right quadrant shows 'Asynchronous: mouse,beer,gnome' with a total time of 279.6ms, a thread held for 1.2ms (1.1 initial + 0.2 generate), and an async wait of 278.4ms, accompanied by a green bar. Each quadrant also features a row of small images representing the resources being used.

Operation	Blocking	Asynchronous
kayak	Total Time: 199.6ms Thread held (red): 199.6ms	Total Time: 200.4ms Thread held (red): 1.0ms (0.9 initial + 0.1 generate) Async wait (green): 199.5ms
mouse,beer,gnome	Total Time: 593.8ms Thread held (red): 593.8ms	Total Time: 279.6ms Thread held (red): 1.2ms (1.1 initial + 0.2 generate) Async wait (green): 278.4ms

```
public interface Publisher<T> {
    void subscribe(Subscriber<? super T> s);
}

public interface Subscriber<T> {
    void onSubscribe(Subscription s);
    void onNext(T t);
    void onError(Throwable t);
    void onComplete();
}

public interface Subscription {
    void request(long n);
    void cancel();
}

public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {}
```

■ JDK 9

- `java.util.concurrent.Flow`
- <http://download.java.net/java/jdk9/docs/api/java/util/concurrent/Flow.html>

```
public class Flow {  
    public interface Publisher<T> { ... }  
    public interface Subscriber<T> { ... }  
    public interface Subscription { ... }  
    public interface Processor<T, R> ...  
}
```

■ MongoDB Example

```
MongoCollection<Document> collection = ...
```

```
Publisher<Success> publisher = collection.insertOne(document);
```

```
publisher.subscribe(new Subscriber<Success>() {
```

```
    public void onSubscribe(Subscription s) {
```

```
        s.request(1); // <-- Only now insertion will occur
```

```
    }
```

```
    public void onNext(Success success) { /* Inserted */ }
```

```
    public void onError(Throwable t) { /* Failed */ }
```

```
    public void onComplete() {}
```

```
});
```

```
Publisher<Attendee> attendees = ...;
Flowable.fromPublisher(attendees)
    .filter(attendee -> attendee.birth.getMonth() == JULY)
    .map(attendee -> attendee.addNote("Awesome"))
    .flatMap(attendee -> saveViaREST(attendee))
    .subscribe(new Subscriber<Result>() {
        public void onSubscribe(Subscription s) {
            _subscription = s; s.request(1);
        }
        public void onNext(Document document) {
            System.out.println(document);
            _subscription.request(1);
        }
        ...
    });
```



```
Publisher<Attendee> attendees = ...;
Flux.from(attendees)
    .filter(attendee -> attendee.birth.getMonth() == JULY)
    .map(attendee -> attendee.addNote("Awesome"))
    .flatMap(attendee -> saveViaREST(attendee))
    .subscribe(new Subscriber<Result>() {
        public void onSubscribe(Subscription s) {
            _subscription = s; s.request(1);
        }
        public void onNext(Document document) {
            System.out.println(document);
            _subscription.request(1);
        }
        ...
    });
```

■ Embrace Reactive Programming

■ It's good for you

- Functional programming
- Asynchronous programming
- Standard APIs (learn one, learnt all)

■ It's good for your business

- More scalable systems
- Less costs, more profit