ZWEI14

Asynchronous Programming - Done right

ZWEI14.

ZWEI14 - A DIGITAL AGENCY WITH CREATIVE DNA.

Idea, concept, design, technology and engage in perfectly together.

We are young but experienced, creative but down to earth, quickly but meticulously, budget-conscious but sophisticated, focused but versatile.

Innovation first. In every project.

Overview

ZWEI14

- In this session: It is all about **asynchrony**
  - **Non-blocking** user interfaces (UIs)
  - **Maintainable** source code
  - **JavaScript** as used language for demonstration

- What **we will** cover:
  - Why should you think about asynchrony
  - Challenges of asynchronism
  - Proofed solution how to address these challenges

- What **we will not** cover:
  - New features of ES6 or ES7 (async, await, yield, function*)

Why asynchrony?

ZWEI14

- Asynchrony
  - „[…] occurrence of events independently of the main program flow and ways to deal with such events." [Alex, 2012]

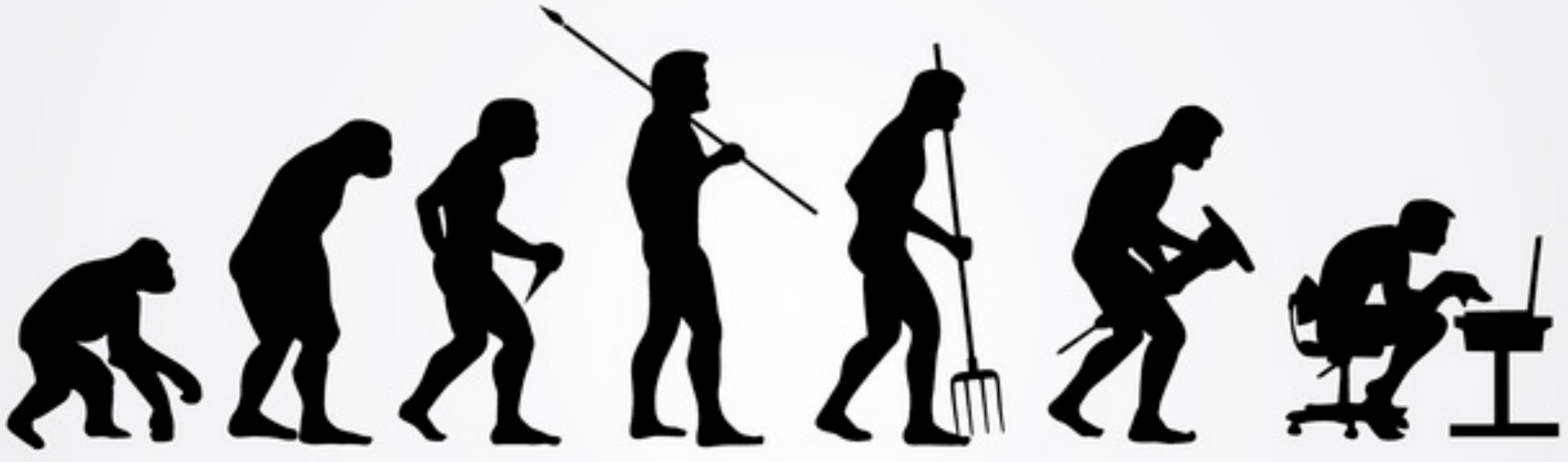
- Separation of processes from main thread (Non-blocking)
  - Communication between client and server
  - Complex business workflows that have to be triggered

- **Goal**: The main thread should not be blocked

Asynchrony is (difficult)

- Keep application state in sync
  - Variables keep state of the asynchronous process
- Error handling
  - Even more states have to be introduced
- Race conditions
  - Changing the state before another process was finished
- Memory leaks
  - Can be difficult to fix

But…



… the evolution goes further

Let’s use our toolbox

Composite

Visitor

Strategy

Observer

Abstract Factory

Prototype

Factory Method

Singleton

Decorator

Iterator

…

Facade

What have these in common?

Composite

Strategy

Visitor

Observer

Abstract Factory

Prototype

Factory Method

Singleton

Decorator

Iterator

...

Facade

Let us ask the GoF…

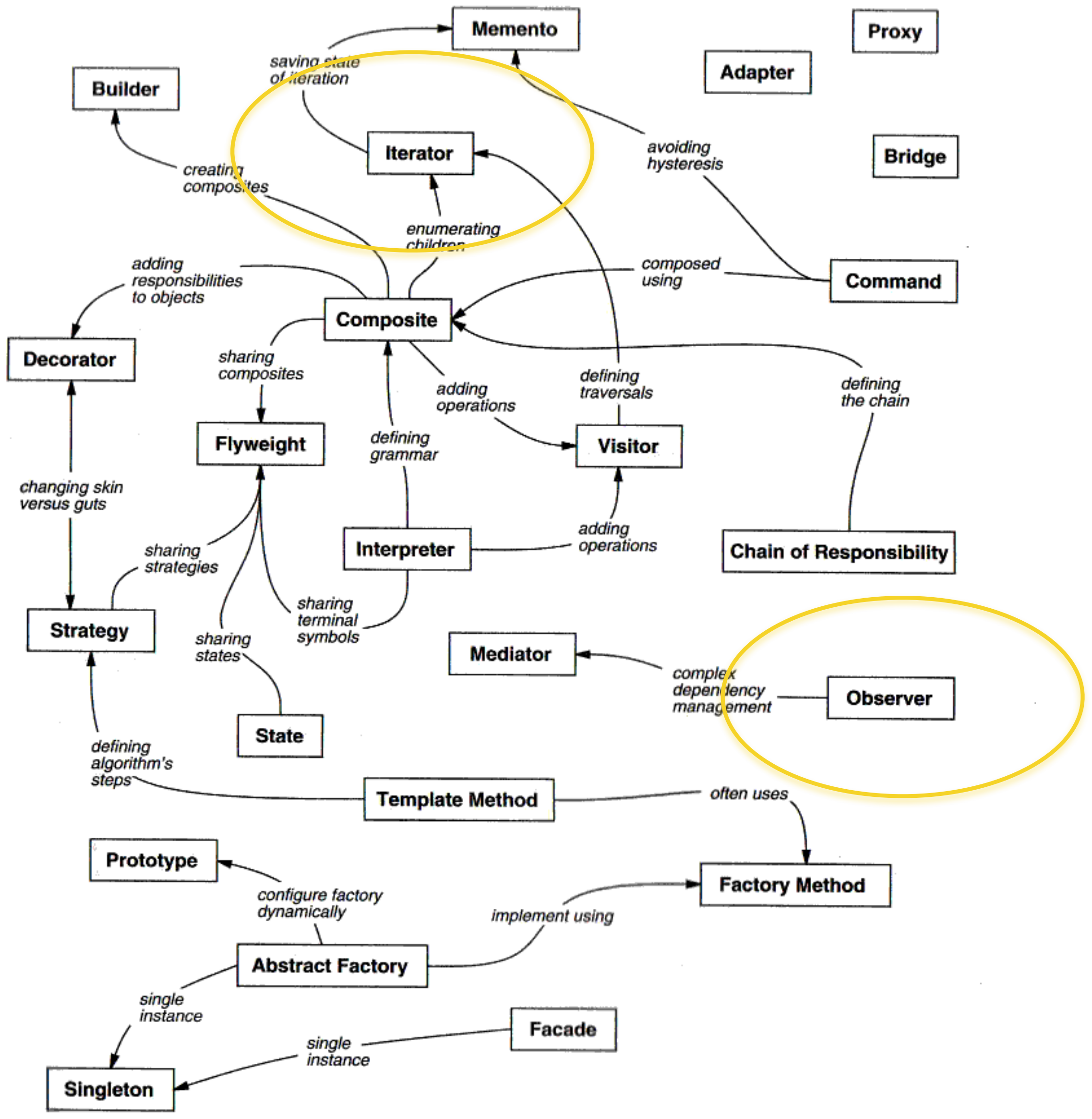


Figure 1.1: Design pattern relationships

Events

- An Event…
  - „[…] is an event is an action or occurrence recognised by software that may be handled by the software“
  - „[…] can be generated or triggered by the system, by the user or in other ways“

- But, events are (simply) collections that are filled over time

```
const events = [{val: 1}, {val: 3},
                {val: 5}]
```

Excerpt: Collections in JS

ZWEI14

ES6 Syntax

ForEach	`events.forEach(x => console.log(x.val))`
Map	`events.map(x => return x.val - 1)`
Filter	`events.filter(x => return x.val > 2)`
Reduce	`events.reduce((x,y) => return x + y), 0)`
ConcatAll	`events.concatAll()`
Concat	`events.concat(events)`

Amount of ordered items

ZWEI 14

ES6 Syntax

```
const amountOfOrderedItems = user => {
user.getOrderedItems.
 map(orderItems => {
  orderItems.
  filter(item => date > 02032016)}).
 concatAll().
 reduce((previous, next) => {
  return previous.amount + next.amount
 }, 0)
}

console.log(amountOrderedItems(user))
```

Observable

- New collection type as part of ReactiveX
  - Collection items over time

```
const eventsOverTime = [{val: 1}, {val: 3}, val: 5},…]
```

- Offers array functions to work with this new type
  - filter, map, reduce, concatAll
- Can be used for animations, events or requests
- Ported to several languages
  - Java, C, C#, JavaScript, Clojure, Swift, Scala,…
- https://github.com/ReactiveX

ReactiveX

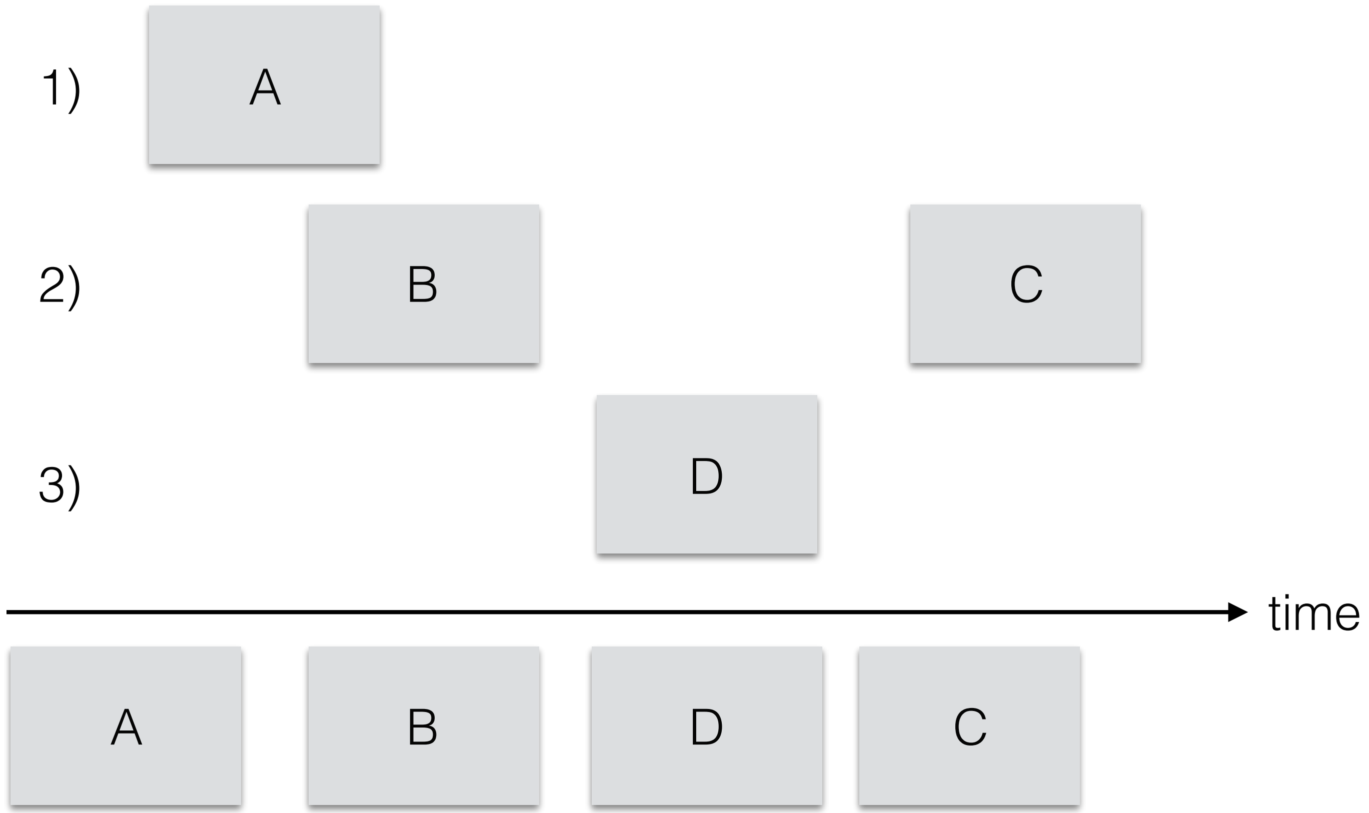
Observable

```
// Create an observable
const $input = $('#searchInput')
const searchInput = Rx.Observable.fromEvent($input,'keyUp')

// Subscribe and do something with the events
const subscription = searchInput.forEach(
   event => sendRequest(event)
   error => handleError(event)
   () => console.log('done')
)

// Unsubscribe
subscription.dispose()
```

ReactiveX

What about race conditions?

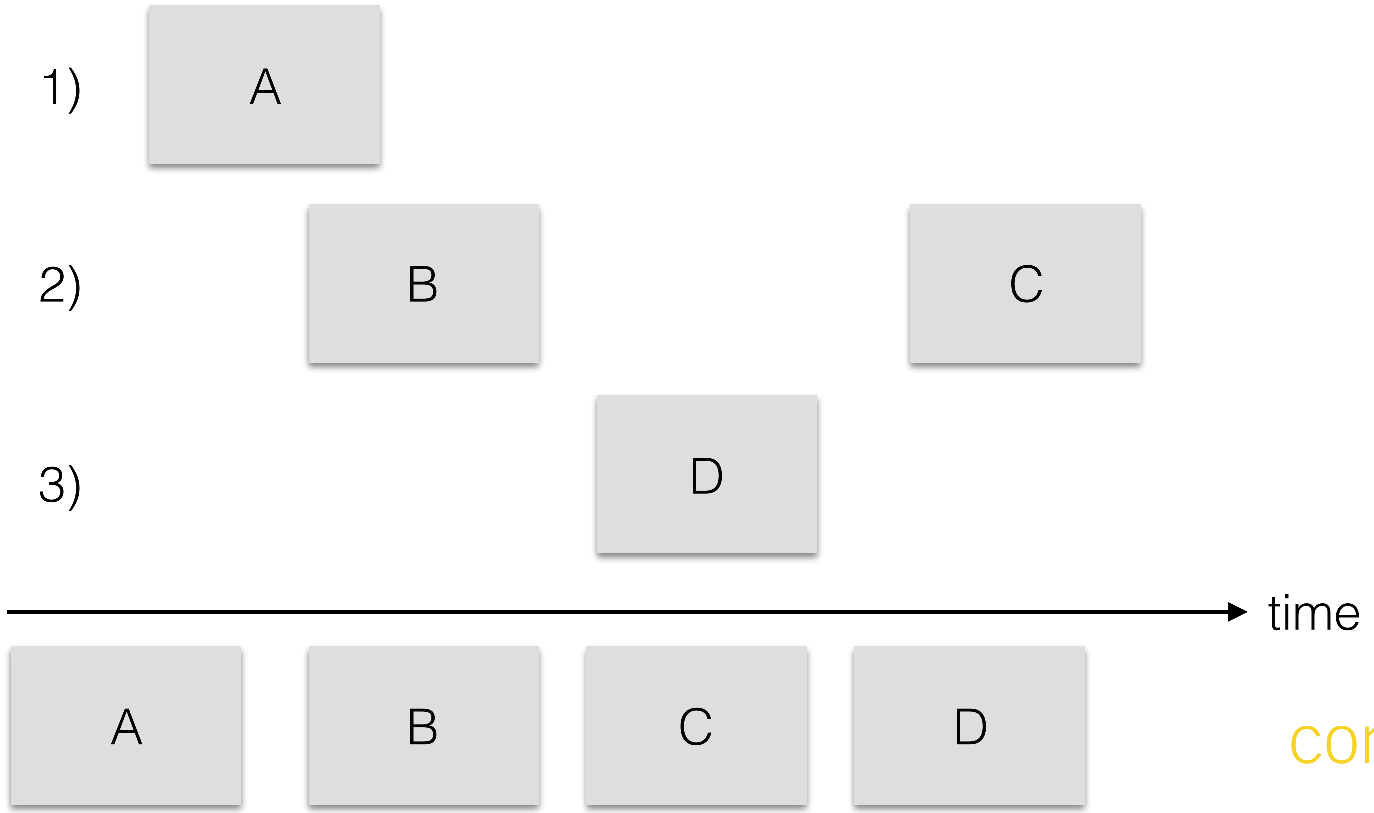
- Processed in order that we want to processed



ReactiveX

What about race conditions?

- Processed in order that we want to processed



concatAll()

Only possible by knowing when its done!!!

ReactiveX

Unsubscribe

- Manually unsubscribe when you do not need the event stream

  `subscription.dispose()`

- But, what happened if I forget to unsubscribe?
  - You are listening on the event stream
  - Consumes events that you do not need anymore
  - Can causes incorrect application state

!! Memory Leak !!

- How can I can automatize the unsubscription?
  - Use another event stream that triggers the unsubscription
  - **takeUntil(event)**

ReactiveX

Sum up

ZWEI14

- Keep application state in sync ✓
  - No additional variable necessary
- Error handling ✓
  - Observables offers direct error handling
- Race conditions ✓
  - Synchronized asynchronous processes
- Memory leaks ✓
  - Unsubscribe only when condition is fulfilled

But the most important point is: **It works in a great scale!**

Any questions?

Thanks for your attention