ELM WORKSHOP

Please `git clone`:

https://github.com/basti1302/elm-workshop.git

Please `git pull` again, if you already cloned it earlier.

```
npm install; npm start
```

HI THERE!

BASTIAN KROL

Developer/Consultant at codecentric
@bastiankrol
basti1302
bastian.krol@codecentric.de

WORKSHOP REPOSITORY

clone with HTTPS

https://github.com/basti1302/elm-workshop.git

or clone with SSH

git@github.com:basti1302/elm-workshop.git

or download: https://github.com/basti1302/elm-workshop/archive/master.zip

codecentric

- Software Development
- Agile Coaching
- Consulting
- Continuous Delivery
- Agile Software Factory
- Software Architecture
- DevOps
- Performance Tuning
- Big Data
- Operations

codecentric

WE'RE HIRING!

- Karlsruhe
- Stuttgart
- Frankfurt
- München
- Berlin
- Hamburg
- Solingen
- Düsseldorf
- Münster
- Dortmund

https://www.codecentric.de/karriere/offene-stellen/

WHAT IS ELM

- Purely Functional
- Static Type System
- Compiles to JavaScript
- Open Source
- Makes Web Development Delightful

MARKETING BLURBS

- Clean Syntax
- No Runtime Exceptions
- Friendly Compiler Error Messages
- Blazing Fast Rendering
- Libraries With SemVer Guarantees
- Smooth JavaScript Interop

SYNTAX BASICS

HELLO WORLD

Elm

```
module Example exposing (..)

import Html

main = Html.text "Hello React Days!"
```

Result

Hello React Days!

HELLO WORLD

Elm

```
module Example exposing (..)

import Html exposing (..)

main = ~~Html.~~text "Hello React Days!"
```

Result

Hello React Days!

HELLO WORLD

Elm

```elm
module Example exposing (..)

import Html exposing (..)

main = text "Hello React Days!"
```

Result

Hello React Days!

FUNCTIONS

```
import Html exposing (..)

-- define a function named greet
greet str =
  text str

-- use the greet function
main =
  greet "Hello React Days!"
```

TYPE ANNOTATIONS

```elm
import Html exposing (..)

greet : String -> Html a
greet str =
  text str

main : Html a
main =
  greet "Hello React Days!"
```

LISTS

```
list1 : List Int
list1 = [1, 2, 3]

list2 : List String
list2 = ["just", "some", "words"]

list3 = [ "some", "strings" ] ++ [ "and", "some", "more" ]
-- => [ "some", "strings", "and", "some", "more" ]

list4 = "before" :: [ "everything", "else" ]
-- => [ "before", "everything", "else" ]
```

LIST.MAP

```
double : Int -> Int
double a = a * 2

list = [ 1, 2, 3, 4 ]

mappedList = List.map double list
-- => mappedList = [ 2, 4, 6, 8 ]
```

HANDS ON!

EXERCISES:

2 - 4

HTML

HTML

Elm

```elm
-- <tag> [ attributes ] [ children ]
p [] [ text "foo" ]

div []
    [ span []
        [ text "bar" ]
    ]
```

HTML

```html
<!-- result -->
<p>foo</p>

<div>
  <span>bar</span>
</div>
```

HTML (2)

Elm

```
ul
    []
    [ li
        []
        [ text "First" ]
    , li
        []
        [ text "Second" ]
    , li
        []
        [ text "Third" ]
    ]
```

HTML

```
<ul>
  <li>First</li>
  <li>Second</li>
  <li>Third</li>
</ul>
```

LET

```
calculate : Int
calculate =
    let
        twentyfour = 3 * 8

        toThePowerOfTwo x =
            x ^ 2

        sixteen = toThePowerOfTwo 4

    in
        twentyfour + sixteen
```

HANDS ON!

EXERCISES:

5 & 6

TYPE SYSTEM

TYPE ALIAS

```
type alias Name = String
type alias Age = Int

type alias ManyStrings = List String
```

RECORDS

```
point = { x = 3, y = 4 }

-- access field
point.x
-- => 3

-- update
{ point | x = 99 }
-- => { x = 99, y = 4 }
```

RECORDS (2)

```
point = { x = 3, y = 4 }

-- multi update
{ point |
    x = point.x + 1,
    y = point.y + 1
}
-- => { x = 4, y = 5 }
```

RECORDS (3)

```
-- type alias for record
type alias Point =
  { x : Int
  , y : Int
  }

point : Point
point = { x = 3, y = 4 }
-- => { x = 3, y = 3 }
```

UNION TYPES

```
type LogLevel = Info | Warn | Error

logLevel : LogLevel
logLevel = Info
```

UNION TYPES (2)

```
type LogLevel = Info | Warn | Error

logLevelToMessage : LogLevel -> String
logLevelToMessage logLevel =
    case logLevel of
        Info ->
            "An information"

        Warn ->
            "A warning"

        Error ->
            "Red alert! Red Alert"
```

UNION TYPES (3)

```elm
type AuthenticationState = NotSignedIn | SignedIn String

authState1 : AuthenticationState
authState1 = NotSignedIn

authState2 : AuthenticationState
authState2 = SignedIn "example.user"
```

UNION TYPES (4)

```elm
type AuthenticationState = NotSignedIn | SignedIn String

authStateToMessage : AuthenticationState -> String
authStateToMessage authState =
    case authState of
        NotSignedIn ->
            "You are not signed in."

        SignedIn userName ->
            "Signed in as " ++ userName
```

HANDS ON!

EXERCISES:

8 & 9

You might want to take a break afterwards :)

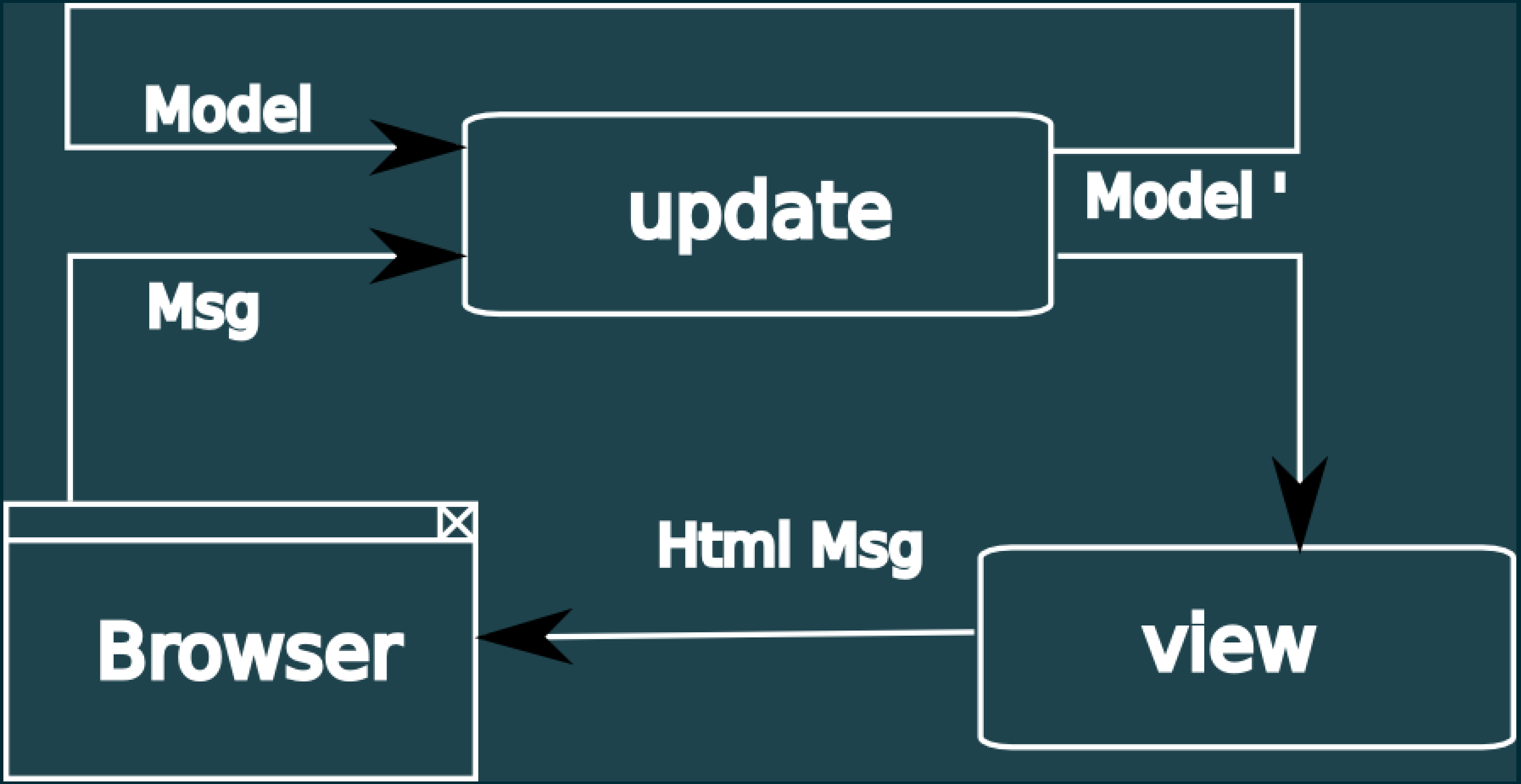
HAVE A BREAK :)

PAUSE :)

FEEDBACK SHEET

USER INTERACTION

TEA - THE ELM ARCHITECTURE



A COUNTER

(A VERY SIMPLE ELM APP)

COUNTER APP - MAIN

```
~~main : Html a~~
~~main = text "Hello React Days!"~~
```

```
main : Program Never Model Msg
main =
    Html.beginnerProgram
        { model = ...?
        , view = ...?
        , update = ...?
        }
```

COUNTER APP - MODEL

```
type alias Model = Int

model : Model
model = 0
```

COUNTER APP - MESSAGES

```
type Msg = Increment | Decrement
```

COUNTER APP - VIEW

```
import Html.Events exposing (..)

view : Model -> Html Msg
view model =
  div []
    [ p [] [ text (toString model) ]
    , button [ onClick Decrement ] [ text "-" ]
    , button [ onClick Increment ] [ text "+" ]
    ]
```

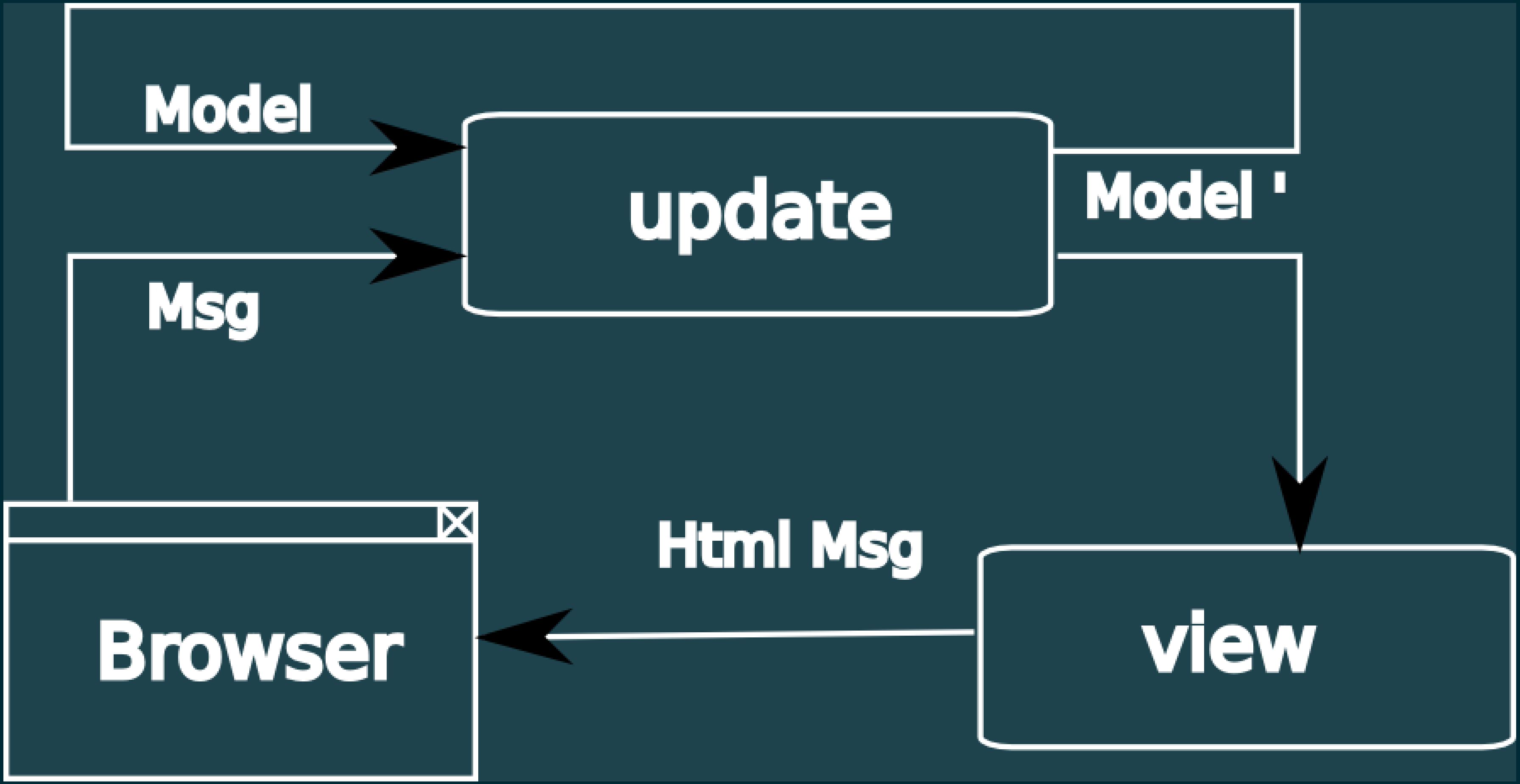
COUNTER APP - UPDATE

```
update : Msg -> Model -> Model
update msg model =
  case msg of
    Increment -> model + 1
    Decrement -> model - 1
```

COUNTER APP - MAIN

```
main : Program Never Model Msg
main =
    Html.beginnerProgram
        { model = model
        , view = view
        , update = update
        }
```

ELM ARCHITECTURE OVERVIEW



THIS IS THE ELM ARCHITECTURE

- **Model** (just a type)
- **`view`**: Model -> Html Msg
- **`update`**: Msg -> Model -> Model

That's all there is.

(nearly)

HANDS ON!

EXERCISE:

12

QUESTIONS?

THANK YOU!

@bastiankrol
basti1302
bastian.krol@codecentric.de
codecentric

BUT WHAT ABOUT SIDE EFFECTS?

- AJAX
- WebSockets
- Randomness
- ...

THE COMPLETE ELM ARCHITECTURE.

- **Commands** (`Cmd`): Elm apps execute side effects via Commands
- **Subscriptions** (`Sub`): Elm apps receive things via Subscriptions (think web sockets)
- Model
- `view`: Model -> Html Msg
- `update`: Msg -> Model -> **(Model, Cmd Msg)**