



elm Amsterdam
26 July 2016

Introduction to elm

Matthias Benkort @KtorZ



github.com/KtorZ/elm-amsterdam-chat

Functional (Reactive) Programming

In the Browser (JS compiled)

Reliable Packages

Expressive Compiler

The Elm Architecture

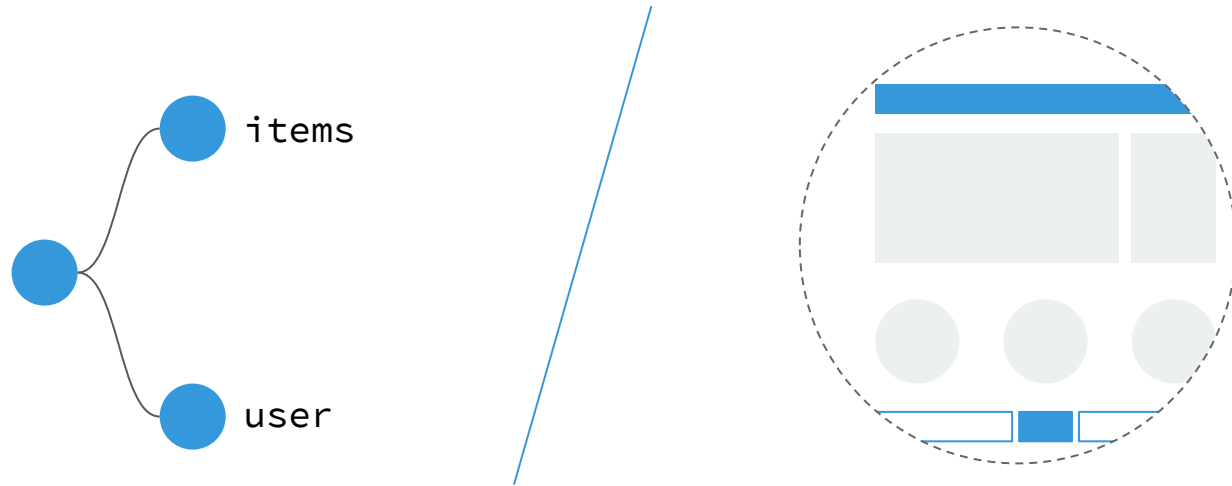
Growing Community



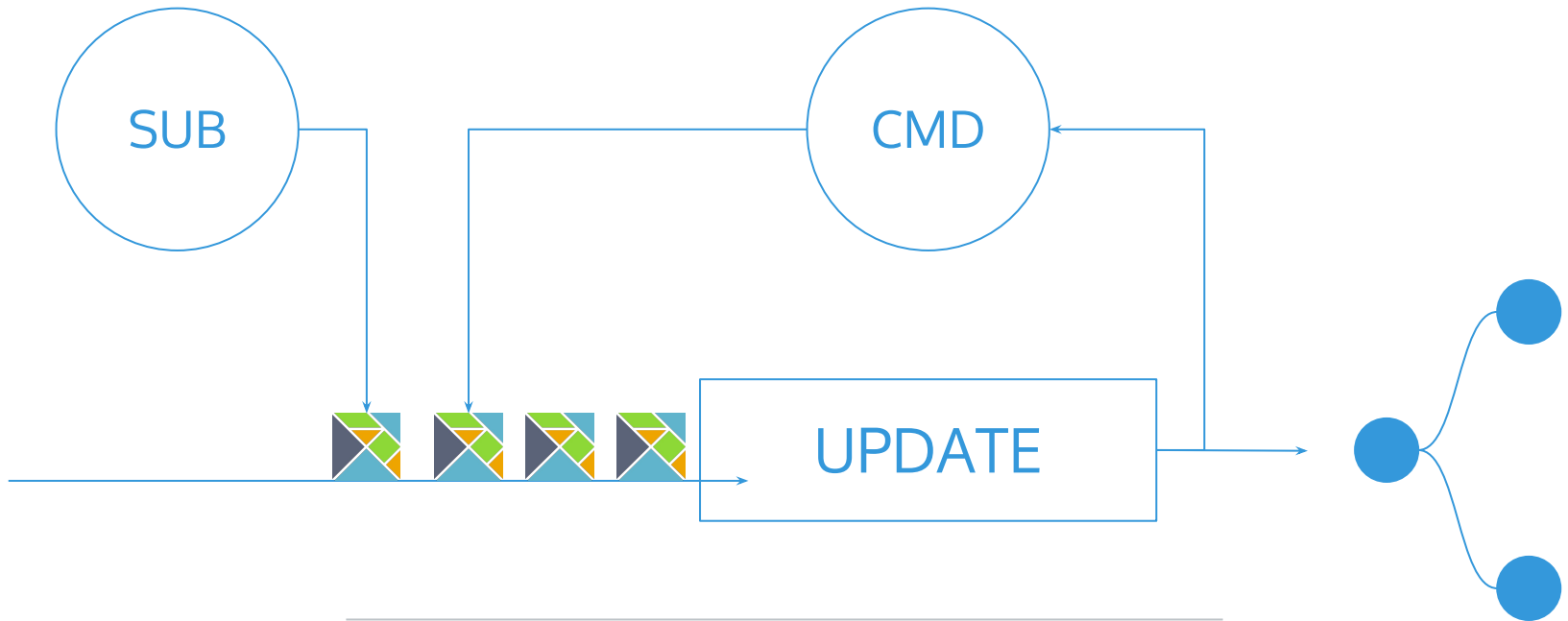
A diagram illustrating the factors contributing to the 'AWESOMENESS' of a technology. On the left, a vertical stack of six blue rectangular boxes contains the following text from top to bottom: 'Functional (Reactive) Programming', 'In the Browser (JS compiled)', 'Reliable Packages', 'Expressive Compiler', 'The Elm Architecture', and 'Growing Community'. To the right of this stack, the word 'AWESOMENESS' is written in large, bold, blue capital letters. A blue arrow points downwards from the top of the stack towards the word, and another blue arrow points upwards from below the word towards the bottom of the stack.

AWESOMENESS

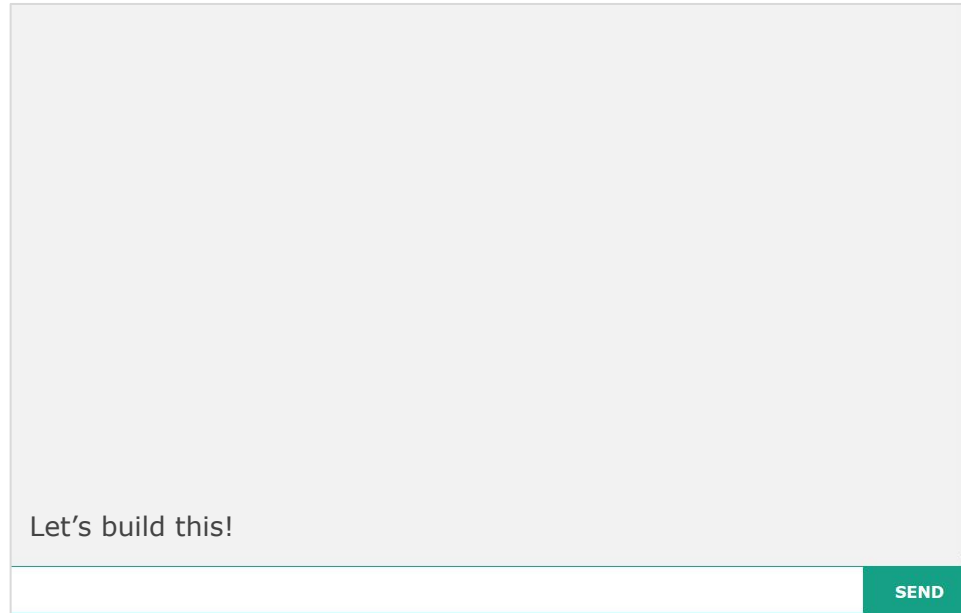
An application **model** represents possible **states**.
One **state** corresponds to a **view** representation.



Commands and subscriptions generate messages.
They **update** the application **model** accordingly.



Example: a Chat Client



Let's build this!

SEND

<http://package.elm-lang.org/packages/elm-lang/html/1.1.0/Html-App>



```
program :  
  { init : (model, Cmd msg)  
  , update : msg -> model -> (model, Cmd msg)  
  , subscriptions : model -> Sub msg  
  , view : model -> Html msg  
  } -> Program Never
```

init : (model, Cmd msg)

```
1 type alias Model =
2   { messages : List String
3     , input : String
4   }
5
6
7 socketAddress : String
8 socketAddress =
9   "ws://localhost:3000"
10
11
12 init : ( Model, Cmd Msg )
13 init =
14   ( { messages = []
15     , input = ""
16   }
17     , Cmd.none
18   )
```

There is no “object” in elm, but **aliases** can be defined for particular **records**

This is a declaration, won't change... ever

Notice that **Model** is used in the type signature

No command are initially intended

update : msg -> model -> (model, Cmd msg)

```
1 import WebSocket
2 import Platform.Cmd
3
4 type Msg
5     = SocketMsg String
6     | Send
7     | Input String
```

Package imports with **explicit** prefixes



```
10 update : Msg -> Model -> ( Model, Cmd Msg )
11 update msg model =
12     case msg of
13         SocketMsg str ->
14             ( { model | messages = model.messages ++ [ str ] }, Cmd.none )
15
16         Send ->
17             ( { model | input = "" }, WebSocket.send socketAddress model.input )
18
19         Input str ->
20             ( { model | input = str }, Cmd.none )
```

update : msg -> model -> (model, Cmd msg)

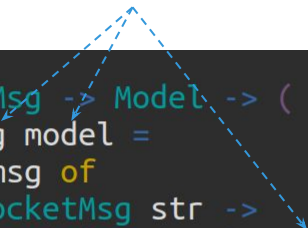
```
1 import WebSocket
2 import Platform.Cmd
3
4 type Msg
5   = SocketMsg String
6   | Send
7   | Input String
```

A **union type** defines **messages** which can trigger **transitions** in the application

```
10 update : Msg -> Model -> ( Model, Cmd Msg )
11 update msg model =
12   case msg of
13     SocketMsg str ->
14       ( { model | messages = model.messages ++ [ str ] }, Cmd.none )
15
16     Send ->
17       ( { model | input = "" }, WebSocket.send socketAddress model.input )
18
19     Input str ->
20       ( { model | input = str }, Cmd.none )
```

update : msg -> model -> (model, Cmd msg)

The **model** is **updated** based on the current one and the given **message**

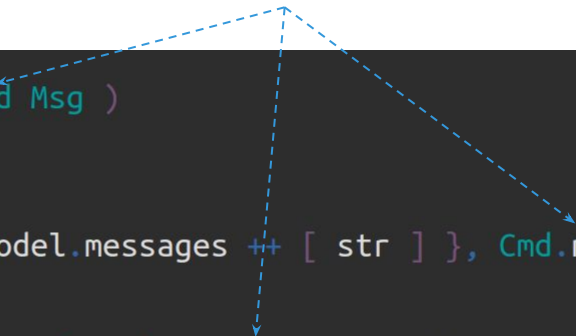


```
10 update : Msg -> Model -> ( Model, Cmd Msg )
11 update msg model =
12     case msg of
13     SocketMsg str ->
14         ( { model | messages = model.messages ++ [ str ] }, Cmd.none )
15
16     Send ->
17         ( { model | input = "" }, WebSocket.send socketAddress model.input )
18
19     Input str ->
20         ( { model | input = str }, Cmd.none )
```

`update : msg -> model -> (model, Cmd msg)`

Commands describe asynchronous actions that may have **side-effects**

```
10 update : Msg -> Model -> ( Model, Cmd Msg )
11 update msg model =
12     case msg of
13     SocketMsg str ->
14         ( { model | messages = model.messages ++ [ str ] }, Cmd.none )
15
16     Send ->
17         ( { model | input = "" }, WebSocket.send socketAddress model.input )
18
19     Input str ->
20         ( { model | input = str }, Cmd.none )
```



subscriptions : model -> Sub msg

```
1 import Platform.Sub
2
3
4 subscriptions : Model -> Sub Msg
5 subscriptions model =
6     WebSocket.listen socketAddress SocketMsg
```

A **Subscription** listens to incoming socket messages

subscriptions : model -> Sub msg

```
1 import Platform.Sub
2
3
4 subscriptions : Model -> Sub Msg
5 subscriptions model =
6     WebSocket.listen socketAddress SocketMsg
```

A **Subscription** listens to incoming socket messages

Commands

Created by us to perform asynchronous operations

Subscriptions

Used to listen to messages emitted sent by other modules

view : model -> Html msg

```
1 import Html exposing (..)
2 import Html.Attributes exposing (..)
3 import Html.Events exposing (..)
```

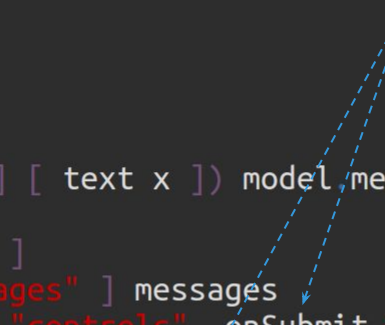
Packages can also be imported **implicitly** (optional prefix)

```
6 view : Model -> Html Msg
7 view model =
8   let
9     messages =
10       List.map (\x -> p [] [ text x ]) model.messages
11   in
12     div [ class "container" ]
13       [ div [ class "messages" ] messages
14         , Html.form [ class "controls", onSubmit Send ]
15           [ input [ type' "text", onInput Input, value model.input ] []
16             , button [ type' "submit" ] [ text "send" ]
17           ]
18       ]
```

view : model -> Html msg

View interactions generate messages

```
6 view : Model -> Html Msg
7 view model =
8   let
9     messages =
10      List.map (\x -> p [] [ text x ]) model.messages
11   in
12     div [ class "container" ]
13       [ div [ class "messages" ] messages
14         , Html.form [ class "controls", onSubmit Send ]
15           [ input [ type' "text", onInput Input, value model.input ] []
16             , button [ type' "submit" ] [ text "send" ]
17           ]
18       ]
```



A diagram consisting of two dashed blue arrows. The first arrow originates from the `onSubmit Send` expression in the `Html.form` block (line 14) and points to the `Send` constructor in the `model.messages` list (line 10). The second arrow originates from the `onInput Input` expression in the `input` block (line 15) and points to the `Input` constructor in the `model.messages` list (line 10). This illustrates how user interactions generate messages that are then processed by the view function.

program :

```
{ init : (model, Cmd msg)
, update : msg -> model -> (model, Cmd msg)
, subscriptions : model -> Sub msg
, view : model -> Html msg
} -> Program Never
```

```
1 import Html.App exposing (program)
2
3
4 main : Platform.Program Never
5 main =
6     program
7     { init = init
8     , update = update
9     , subscriptions = subscriptions
10    , view = view
11    }
```



76 sloc

67K
(minified)

No
Runtime
Error

Resources



elm-tutorial.org builtwithelm.co drboolean.gitbooks.io/mostly-adequate-guide/content



dailydrip.com/topics/elm elmseeds.thaterikperson.com elmweekly.nl frontendnewsletter.com



[isRuslan/awesome-elm](https://github.com/isRuslan/awesome-elm)



[@elmlang](https://twitter.com/elmlang), [@czaplic](https://twitter.com/czaplic), [@rtfeldman](https://twitter.com/rtfeldman)

Dankjewel.