FUNKTIONALE ENTWICKLUNG IM WEB MIT ELM

Carsten König

WAS IST ELM?

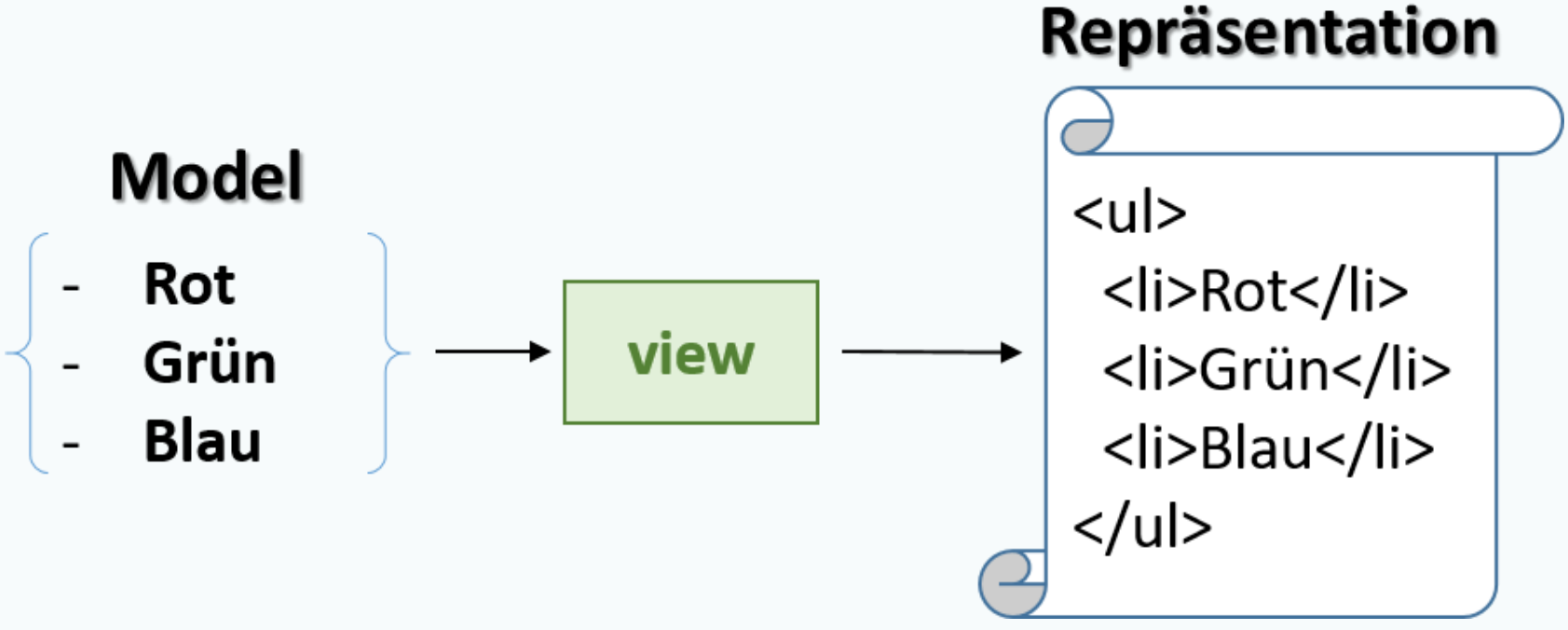
- Web-Frontendentwicklung
- *freundliche* funktionale Sprache
  - ML Syntaxfamilie (Ocaml, F#, Haskell, ...)
  - *pure/total*
- *keine Runtime-Exceptions* *

THE ELM ARCHITECTURE

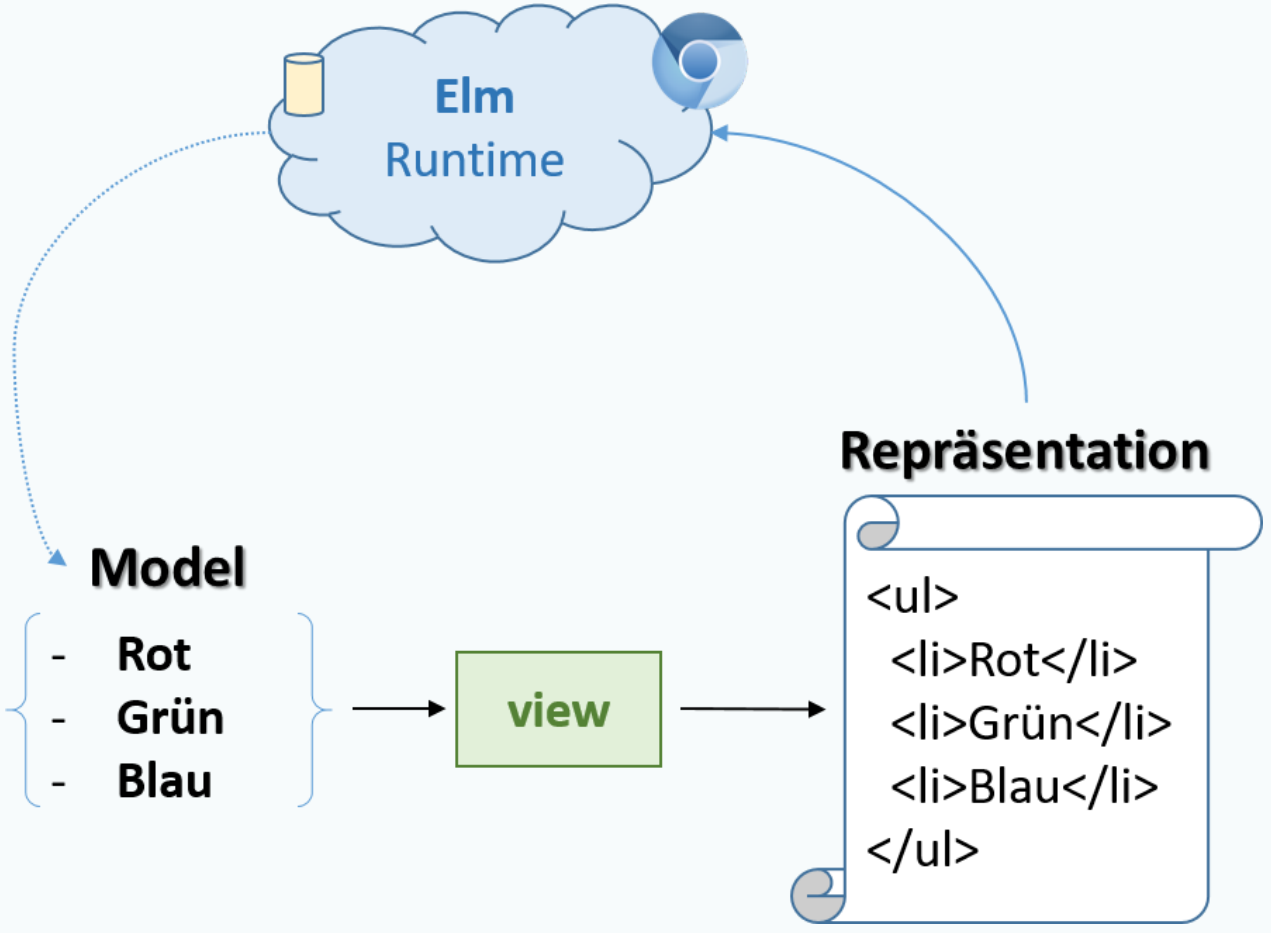
ZUSTAND

single source of truth

DARSTELLUNG




```
view : Model -> Html ..
```

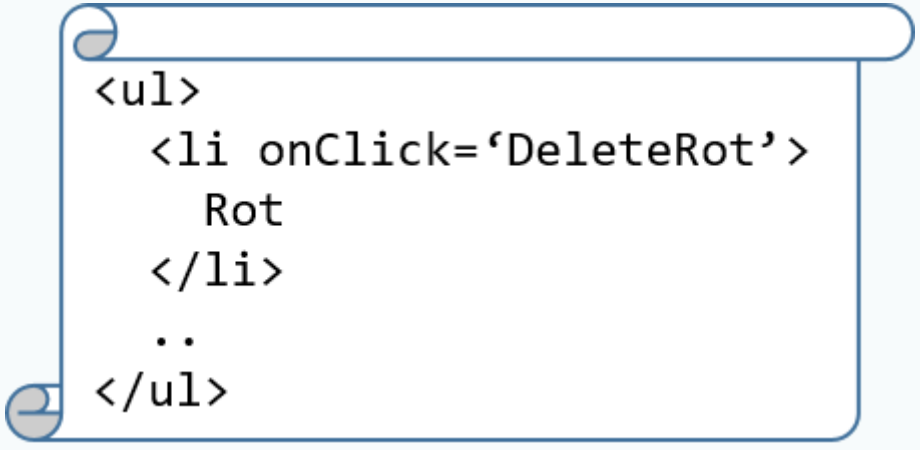

Elm
Runtime
Repräsentation
Model
- Rot
- Grün
- Blau
view

Rot
Grün
Blau

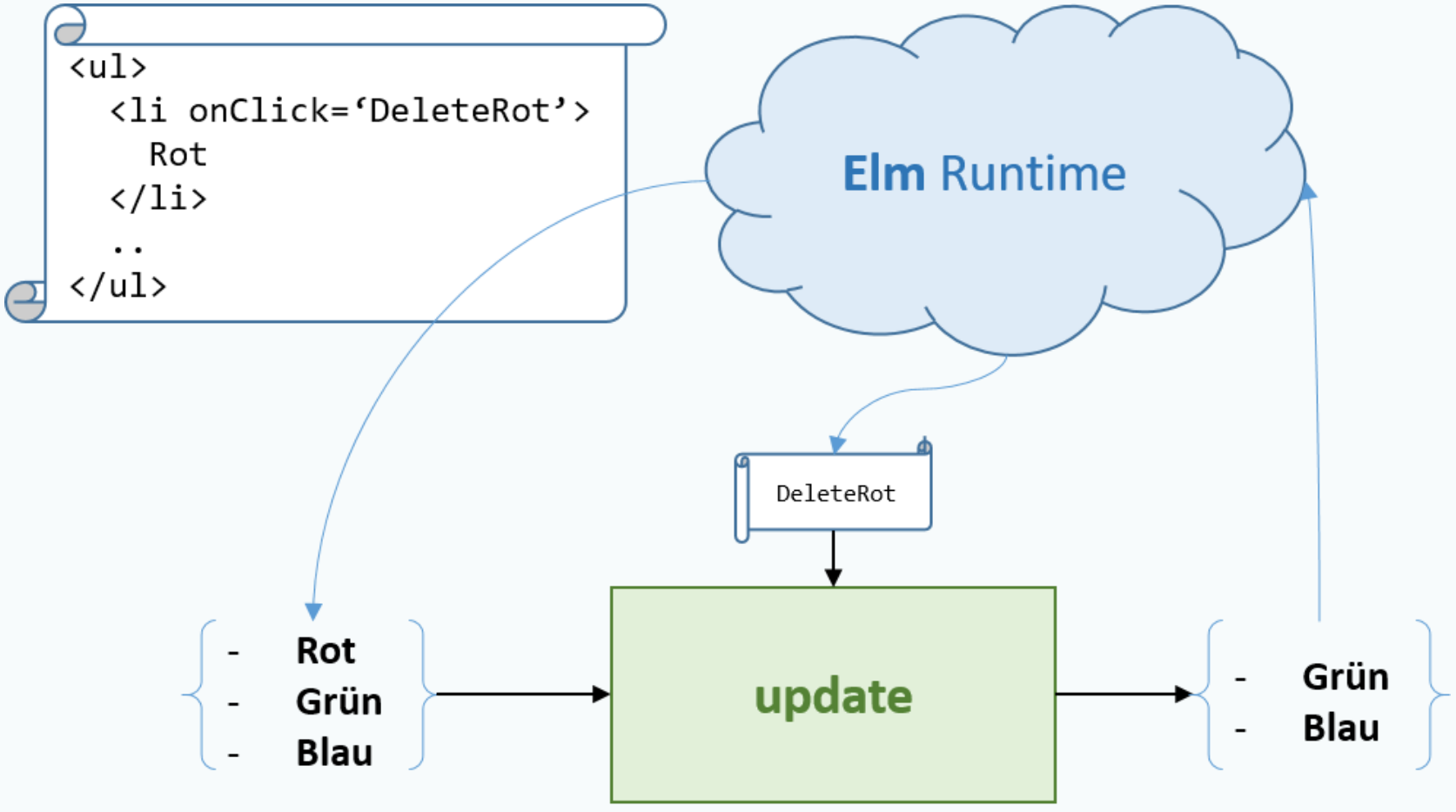
Runtime <-> DOM

ZUSTANDSÄNDERUNGEN

ausgelöst durch *Messages*

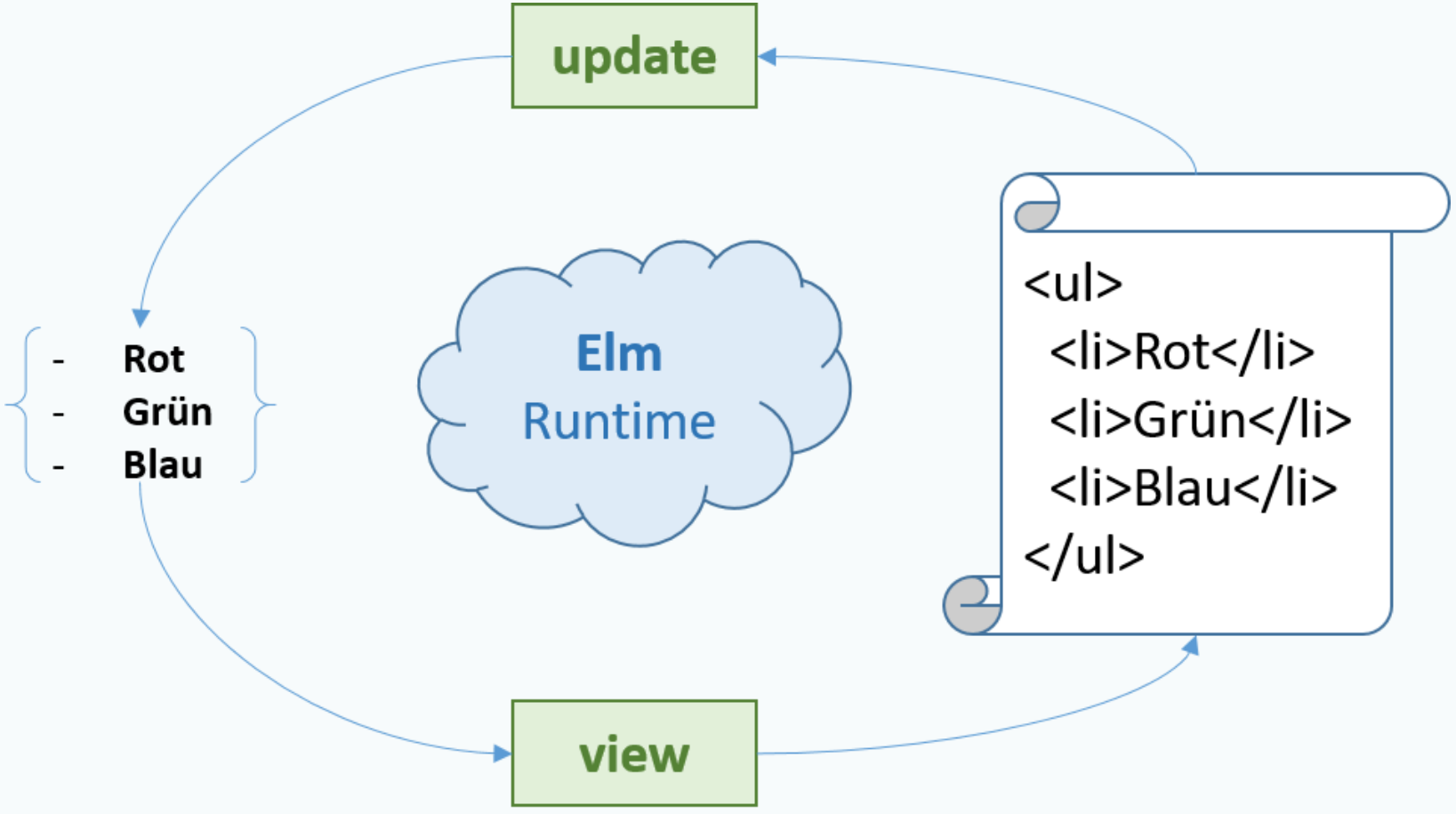



```
: Html Msg
```




```
update : Msg -> Model -> Model
```

ELM ARCHITEKTUR



ModelViewUpdate

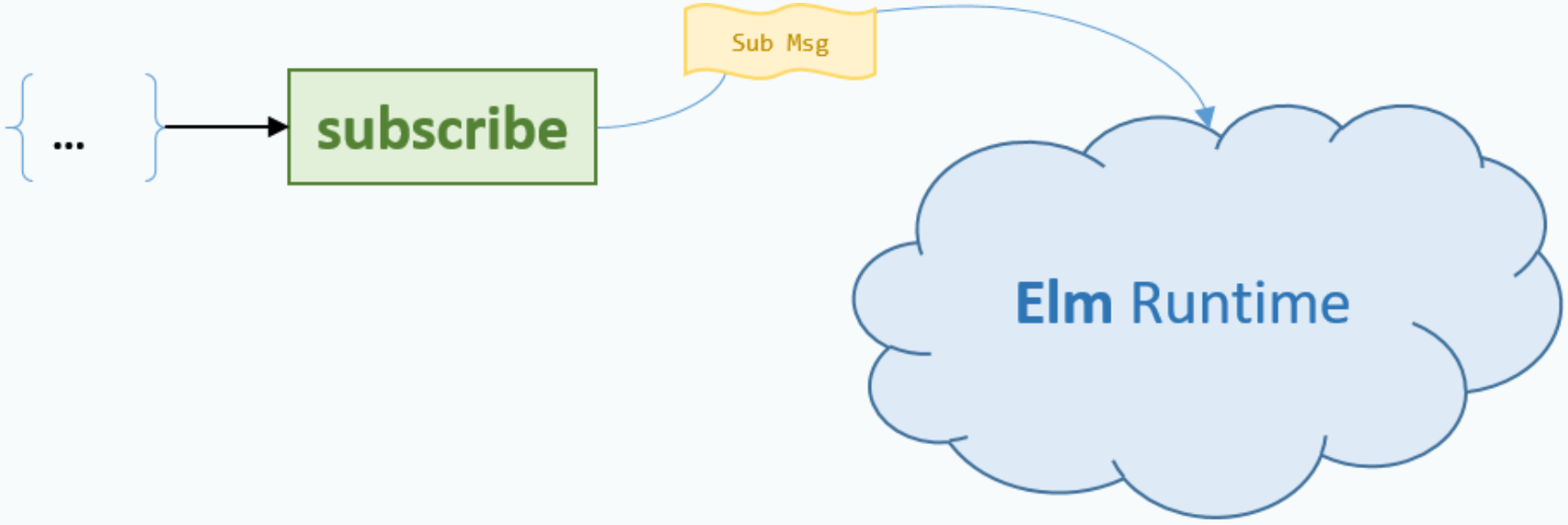
DEMO

TEA *NEXT LEVEL*

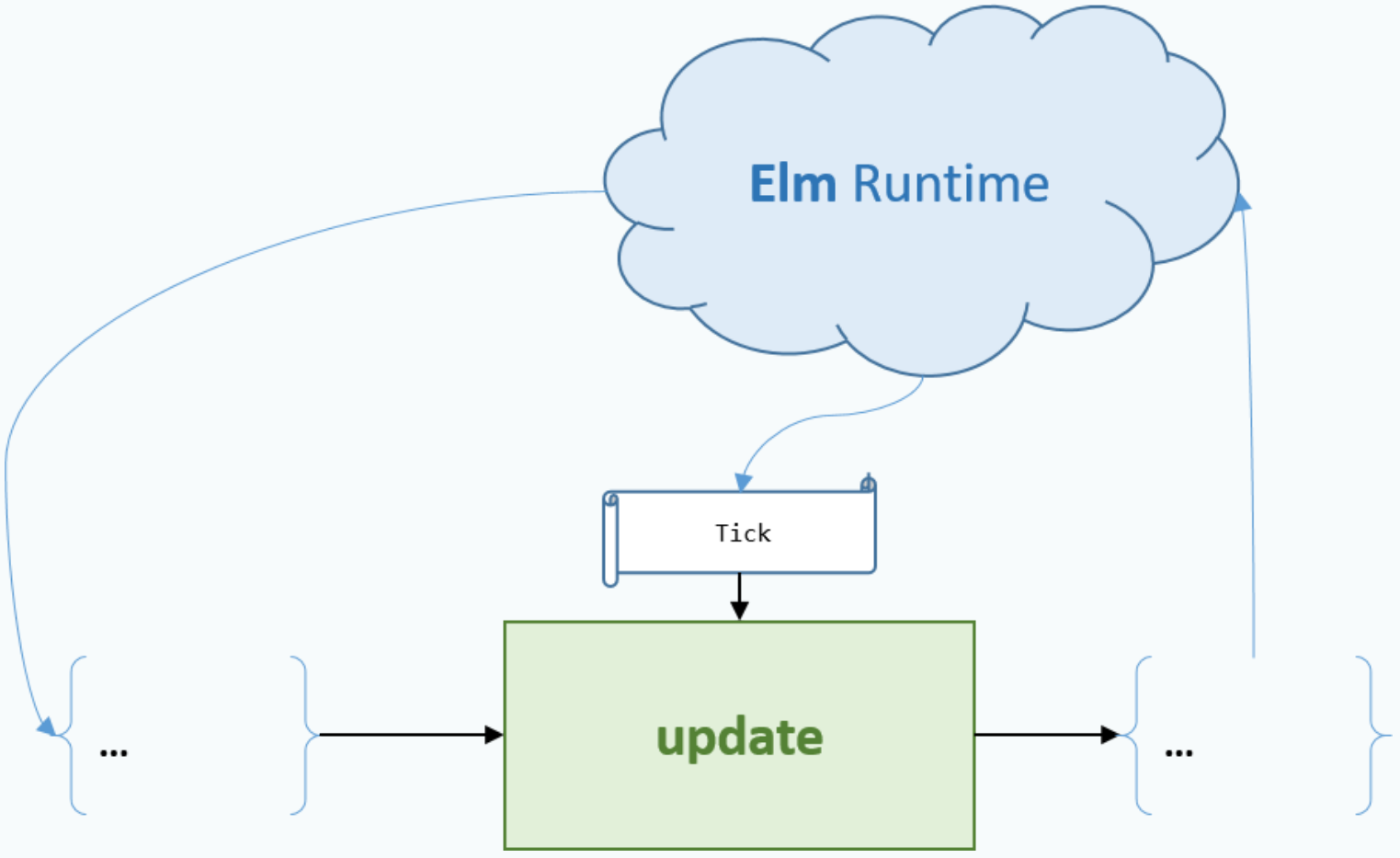
SEITENEFFEKTE

- wie definieren wir einen Timer?
  - **Subscriptions**
- wie kommunizieren wir mit dem Backend?
  - **Commands**

SUB

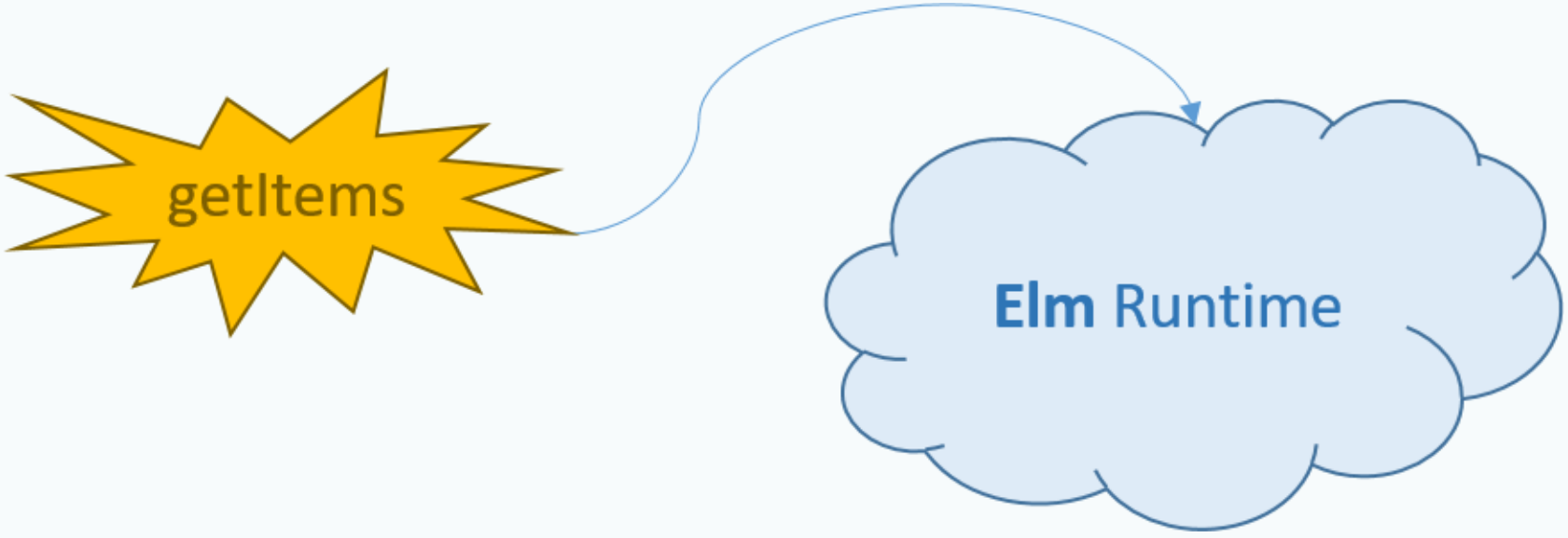


`subscriptions`

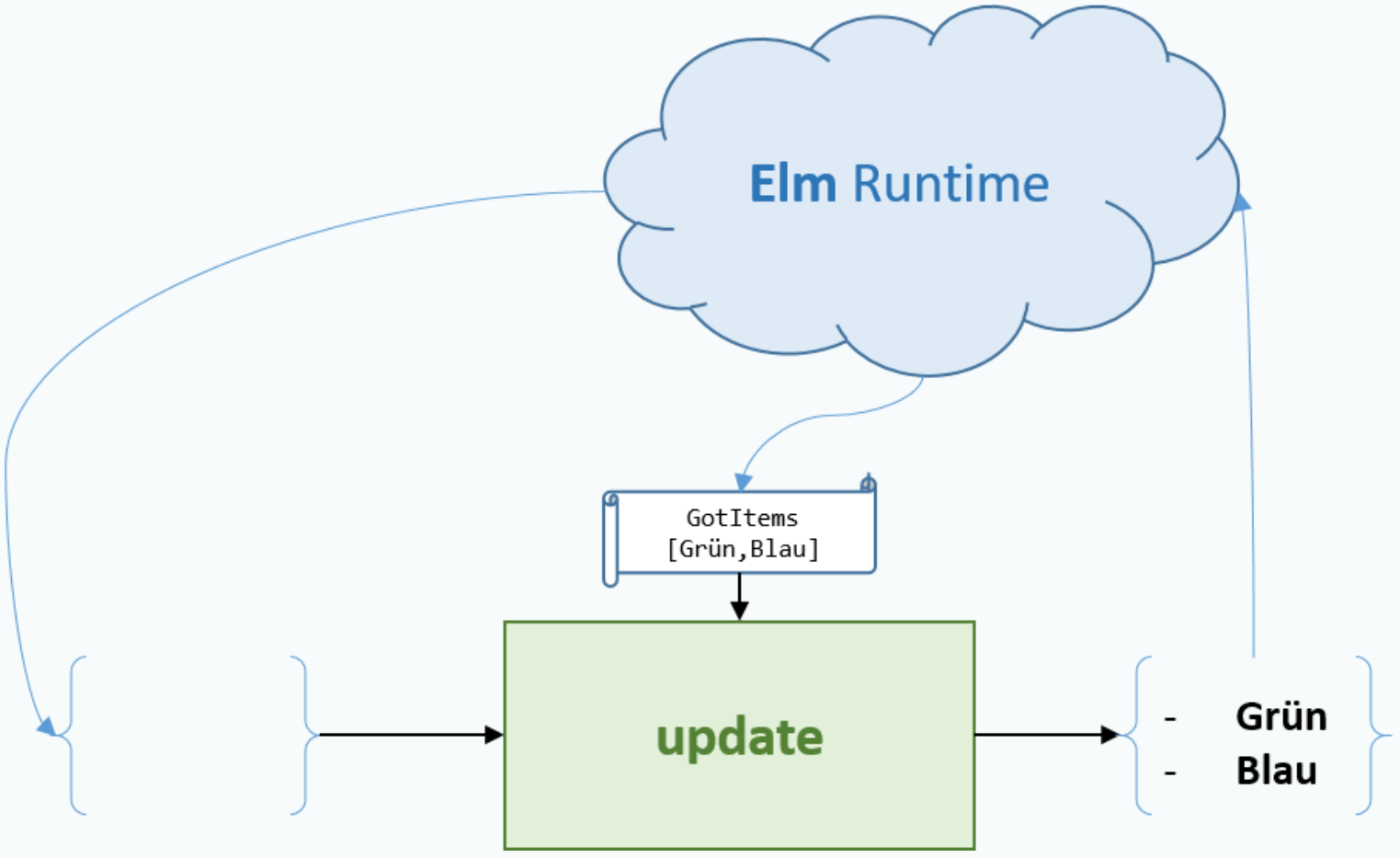


Verarbeitung über `update`

CMD



Runtime bekommt `Cmd`



Verarbeitung über `update`

ERWEITERTES PROGRAMM

```
element :
    { init          : (model, Cmd msg)
    , update        : msg -> model -> (model, Cmd msg)
    , subscriptions : model -> Sub msg
    , view          : model -> Html msg
    }
    -> Program () model msg
```

DECODER / REQUESTS

JSON → ELM-TYP

```
decodeString : Decoder a -> String -> Result String a
```

PRIMITIVE DECODER

```
Json.Decode.int    : Decoder Int
Json.Decode.string : Decoder String
Json.Decode.bool   : Decoder Bool
..
```

KOMBINATOREN

```
Json.Decode.list  :                Decoder a -> Decoder (List a)
Json.Decode.field : String ->      Decoder a -> Decoder a

Json.Decode.map2  : (a -> b -> val) ->
                    Decoder a -> Decoder b -> Decoder val
```

KOMMUNIKATION MIT BACKEND

```
Http.get  : { url : String, Expect msg } -> Cmd msg


-- Beispiel
get baseUrl toMsg todoId =
    Http.get
        { url = baseUrl ++ "todos/" ++ Todos.idToString todoId
        , expect = Http.expectJson toMsg Todos.itemDecoder
        }
```

DEMO

NAVIGATION

ROUTEN

```elm
type Route
    = RouteAll
    | RouteActive
    | RouteCompleted


routeP : Parser (Route -> a) a
routeP =
    Route.oneOf
        [ Route.map RouteAll Route.top
        , Route.map RouteActive (Route.s "active")
        , Route.map RouteCompleted (Route.s "completed")
        ]
```

APPLIKATION

```
application :
    { init : flags -> Url -> Key -> ( model, Cmd msg )
    , view : model -> Document msg
    , update : msg -> model -> ( model, Cmd msg )
    , subscriptions : model -> Sub msg
    , onUrlRequest : UrlRequest -> msg
    , onUrlChange : Url -> msg
    }
    -> Program flags model msg
```

```
UrlRequested urlRequest ->
    case urlRequest of
        Browser.Internal url ->
            ...
            (model
            , Nav.pushUrl
                model.navKey (Url.toString url)
                )

        Browser.External url ->
            ( model, Nav.load url )
```

```
UrlChanged newUrl ->
    ( { model | activeFilter = urlToFilter newUrl }, Cmd.none )
```

DEMO

JAVASCRIPT INTEROP

PORTS

ELM → JS

Elm:

```
port module PortModule exposing (..)

port toJS : String -> Cmd msg
```

JavaScript:

```
var app = Elm.Main.init(...);
app.ports.toJS.subscribe (function(text){
  alert(text);
});
```

ELM ← JS

Elm:

```
port module PortModule exposing (..)

port fromJs : (String -> msg) -> Sub msg
```

JavaScript:

```
app.ports.fromJS.send(input);
```

FRAGEN?

VIELEN DANK

LINKS UND CO.

- Code & Slides github.com/CarstenKoenig/DWX2019_Elm

- Elm Guide Online: https://guide.elm-lang.org/
- Installieren: https://guide.elm-lang.org/install.html
- Package Verzeichnis / Docs: http://package.elm-lang.org/

- *fancy Search* https://klaftertief.github.io/elm-search/