Elm

Functional reactive programming for the browser

Agenda

- Elm Architecture by example
- Why Elm?
- Is it production ready?

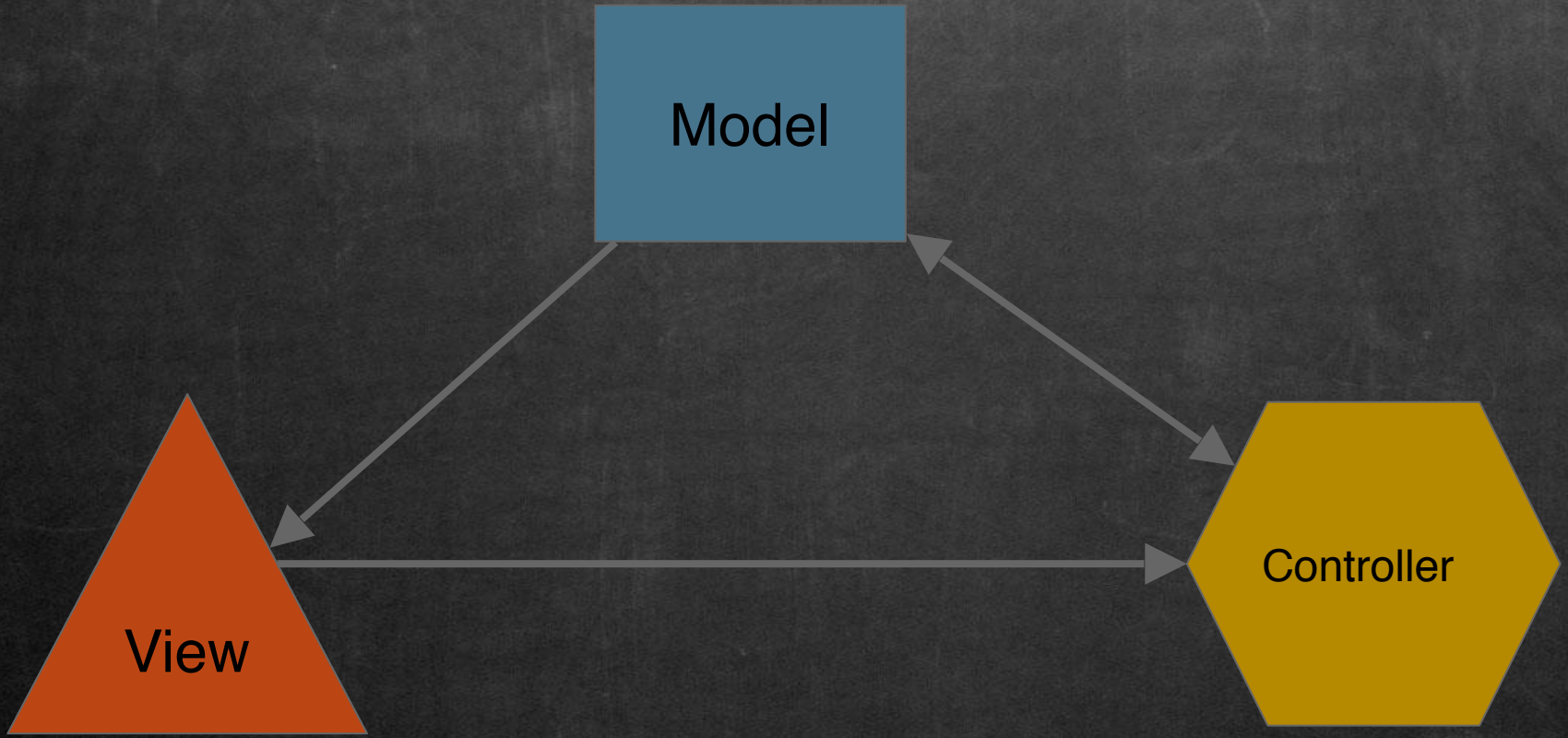
An Elm Program

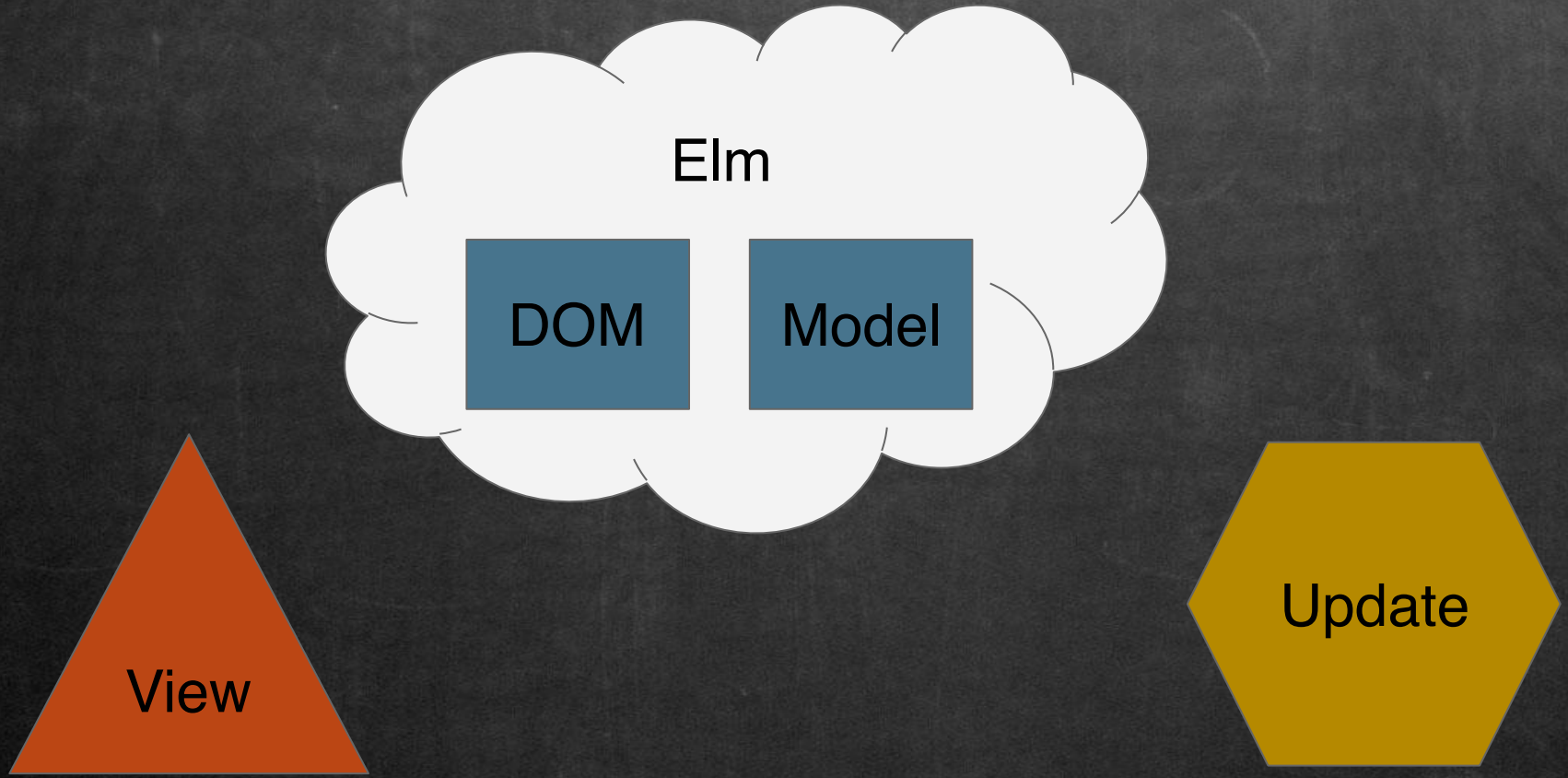
```elm
type Msg
  = Add
  | Sub


init = 0


update msg model =
  case msg of
    Add ->
      model + 1

    Sub ->
      model - 1
```

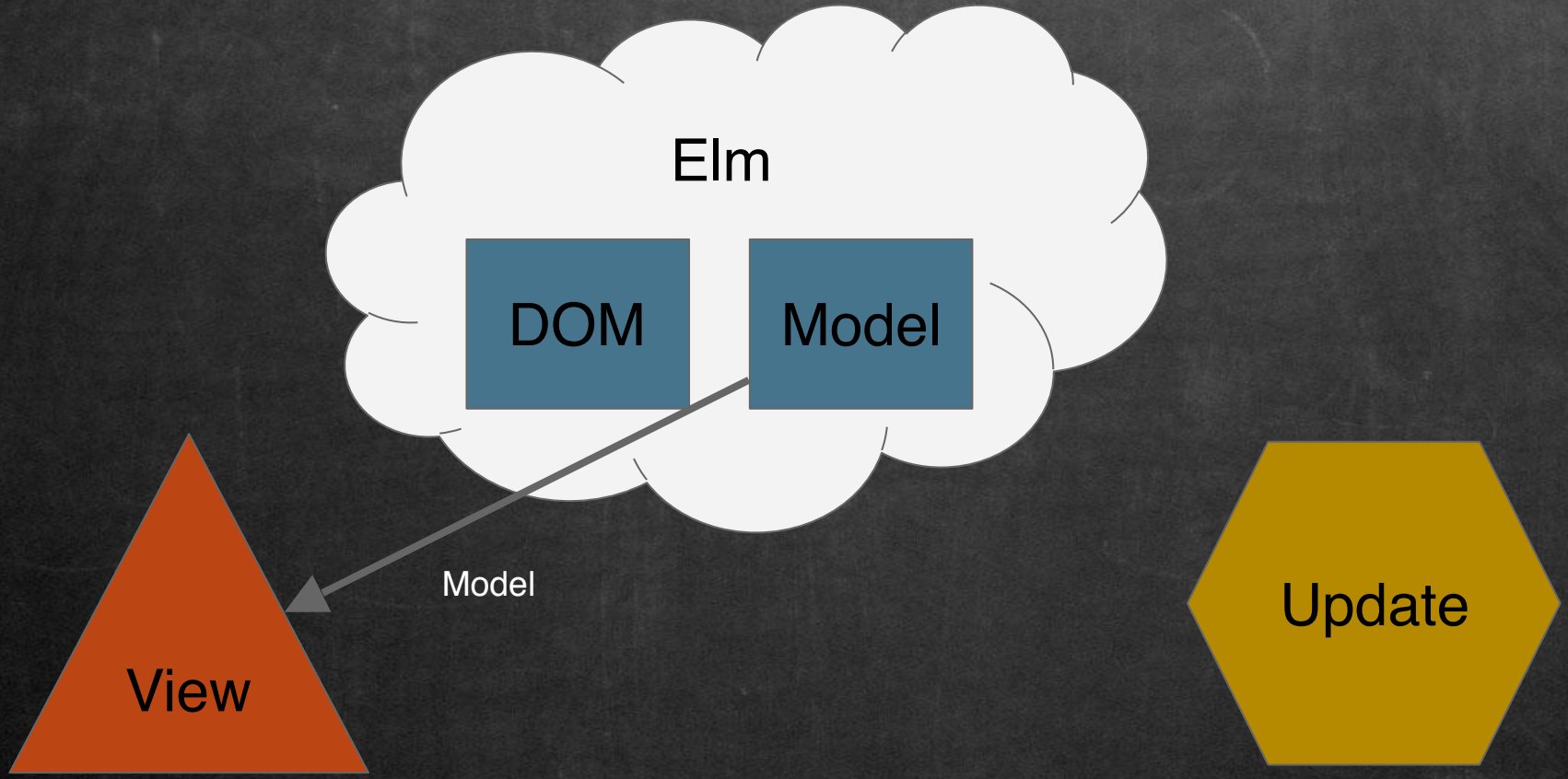
```elm
view model =
  div []
    [ span [] [text (toString model)]
    , button [ onClick Add ] [ text "+" ]
    , button [ onClick Sub ] [ text "-" ]
    ]
```

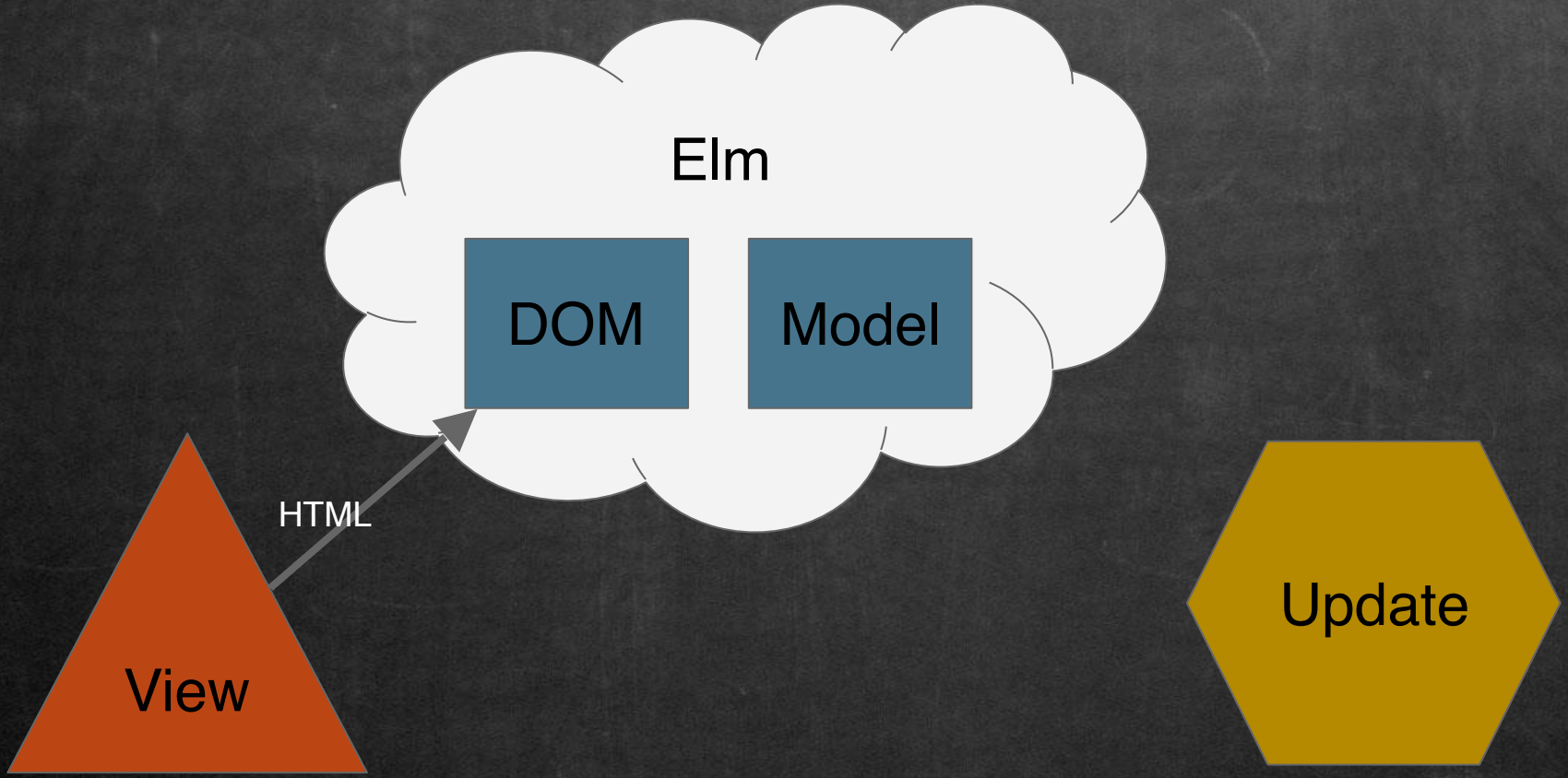
```elm
type alias Model = Int


type Msg
  = Add
  | Sub


init : Model
init = 0


update : Msg -> Model -> Model
update msg model =
  case msg of
    Add ->
      model + 1

    Sub ->
      model - 1
```

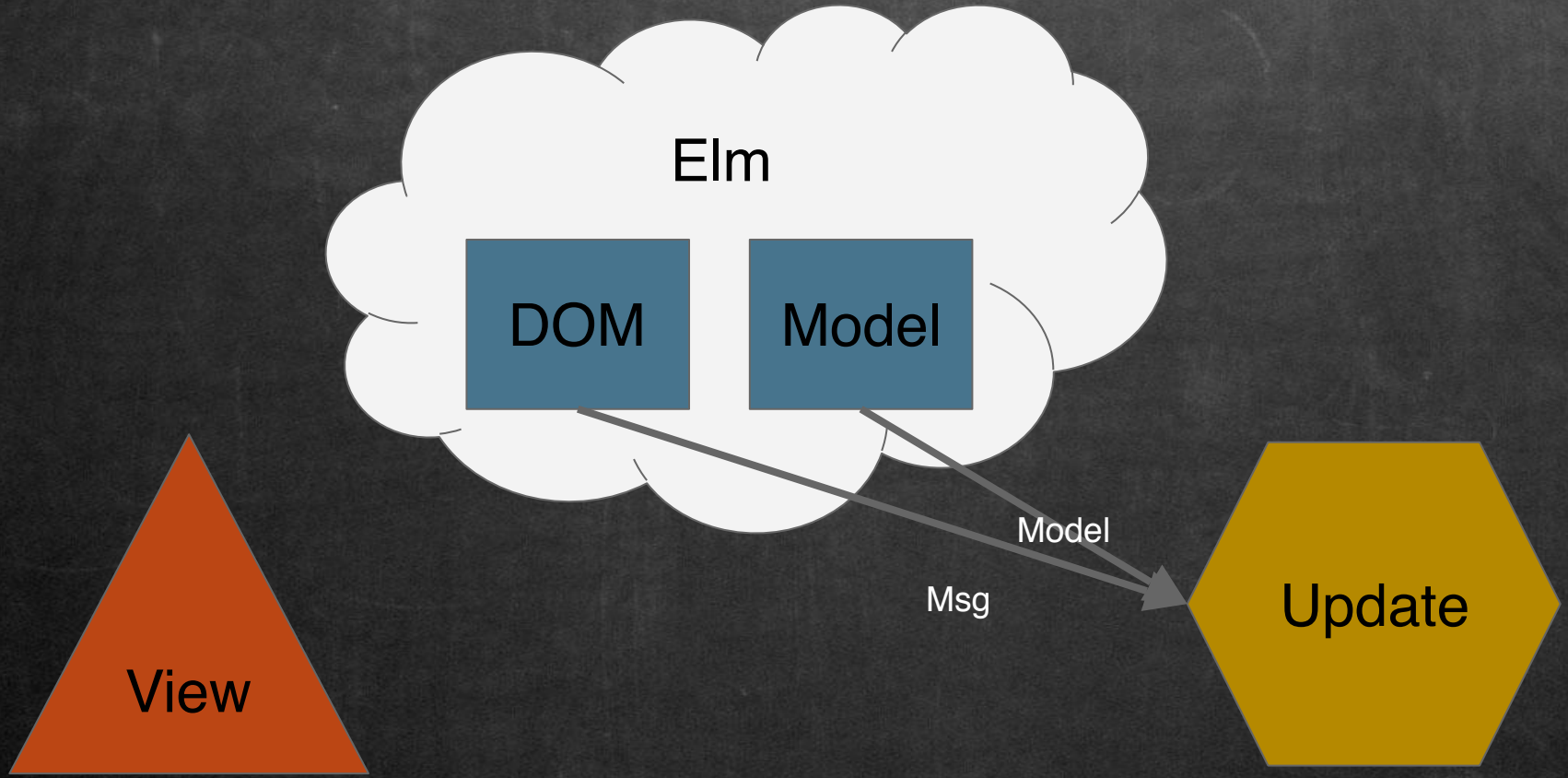
```elm
view : Model -> Html Msg
view model =
  div []
    [ span [] [text (toString model)]
    , button [ onClick Add ] [ text "+" ]
    , button [ onClick Sub ] [ text "-" ]
    ]
```

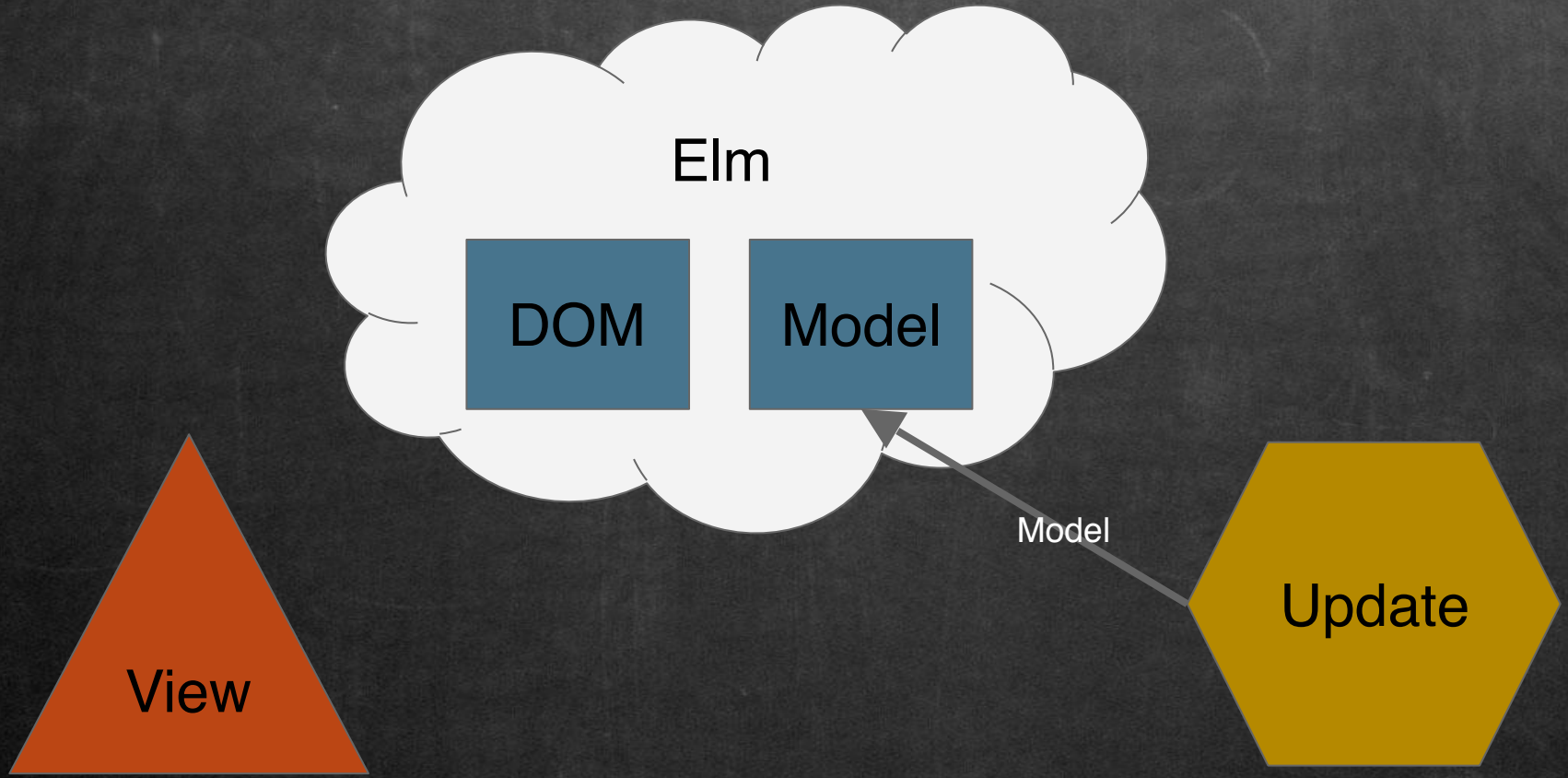

Model
View
Controller

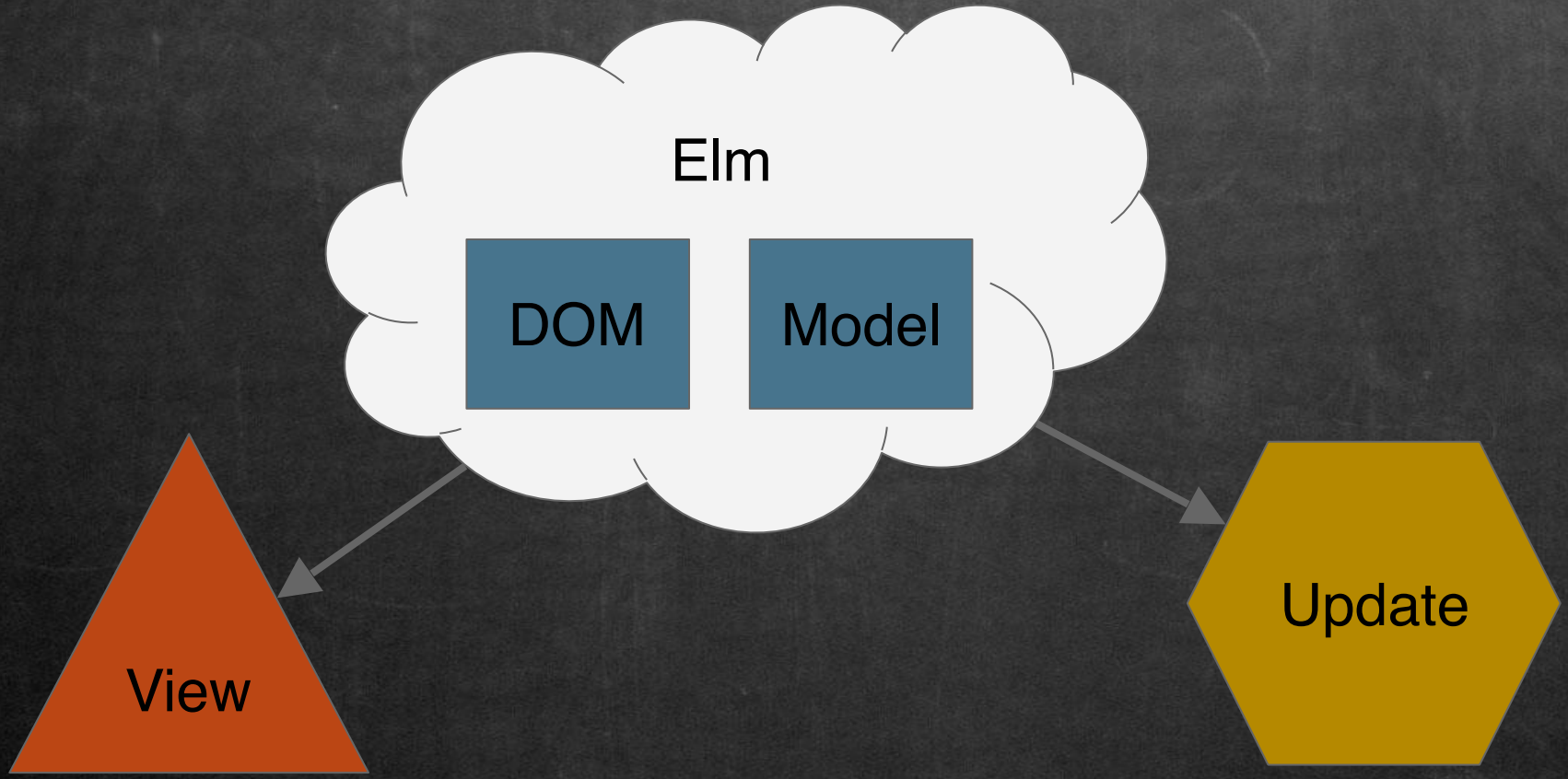

Elm
DOM
Model
View
Update


Elm
DOM
Model
Model
View
Update


Elm
DOM
Model
HTML
View
Update


Elm
DOM
Model
Model
Msg
View
Update


Elm
DOM
Model
Model
View
Update


Elm
DOM
Model
View
Update

```elm
type alias Model = Int


type Msg
  = Add
  | Sub


init : Model
init = 0


update : Msg -> Model -> Model
update msg model =
  case msg of
    Add ->
      model + 1

    Sub ->
      model - 1
```

```elm
view : Model -> Html Msg
view model =
  div []
    [ span [] [text (toString model)]
    , button [ onClick Add ] [ text "+" ]
    , button [ onClick Sub ] [ text "-" ]
    ]
```

Characteristics

An Elm program has / is

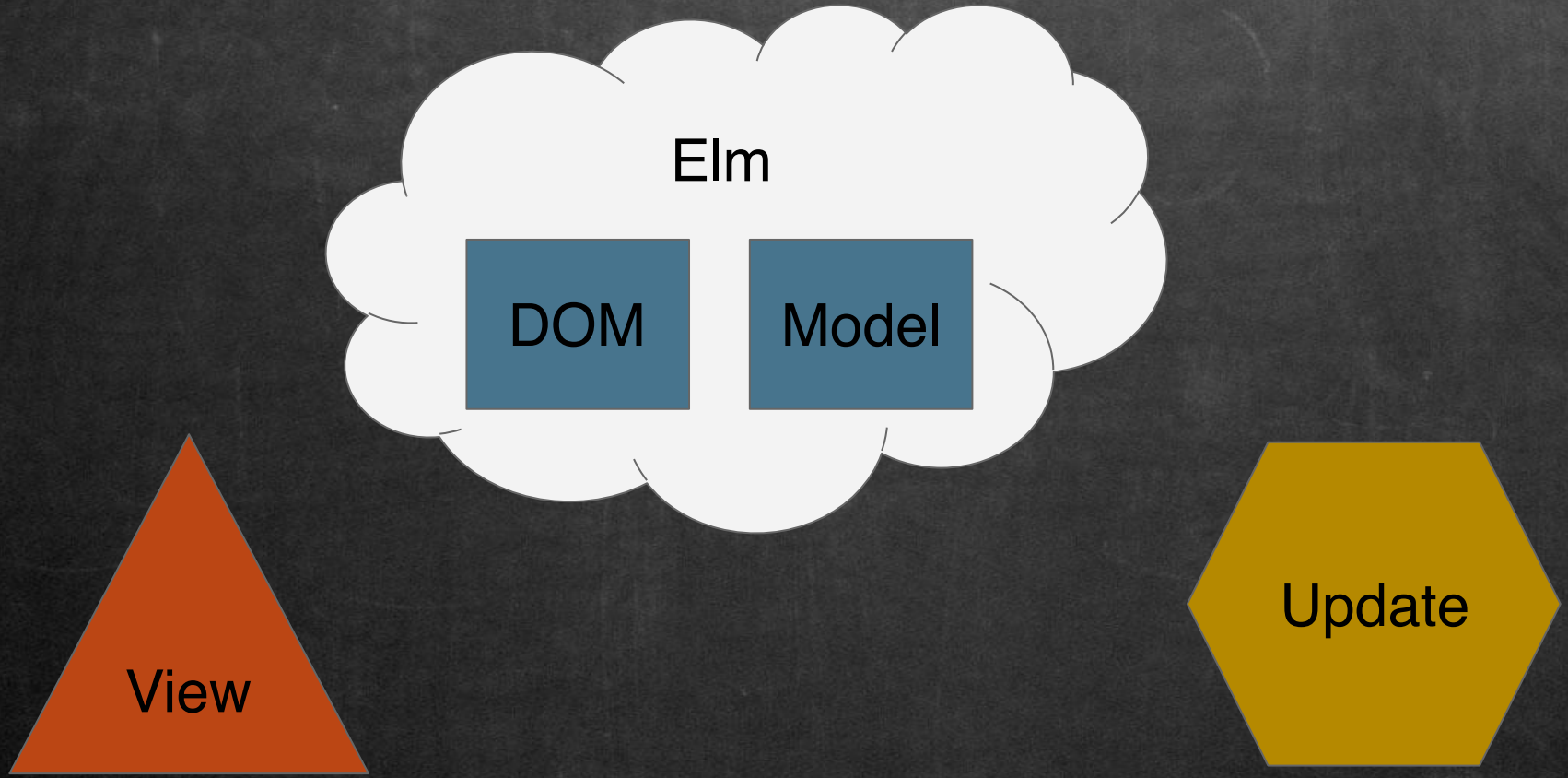
- only true (pure) functions (no side effects)
- immutable data
- no state (DOM / Model)
- reactive by design

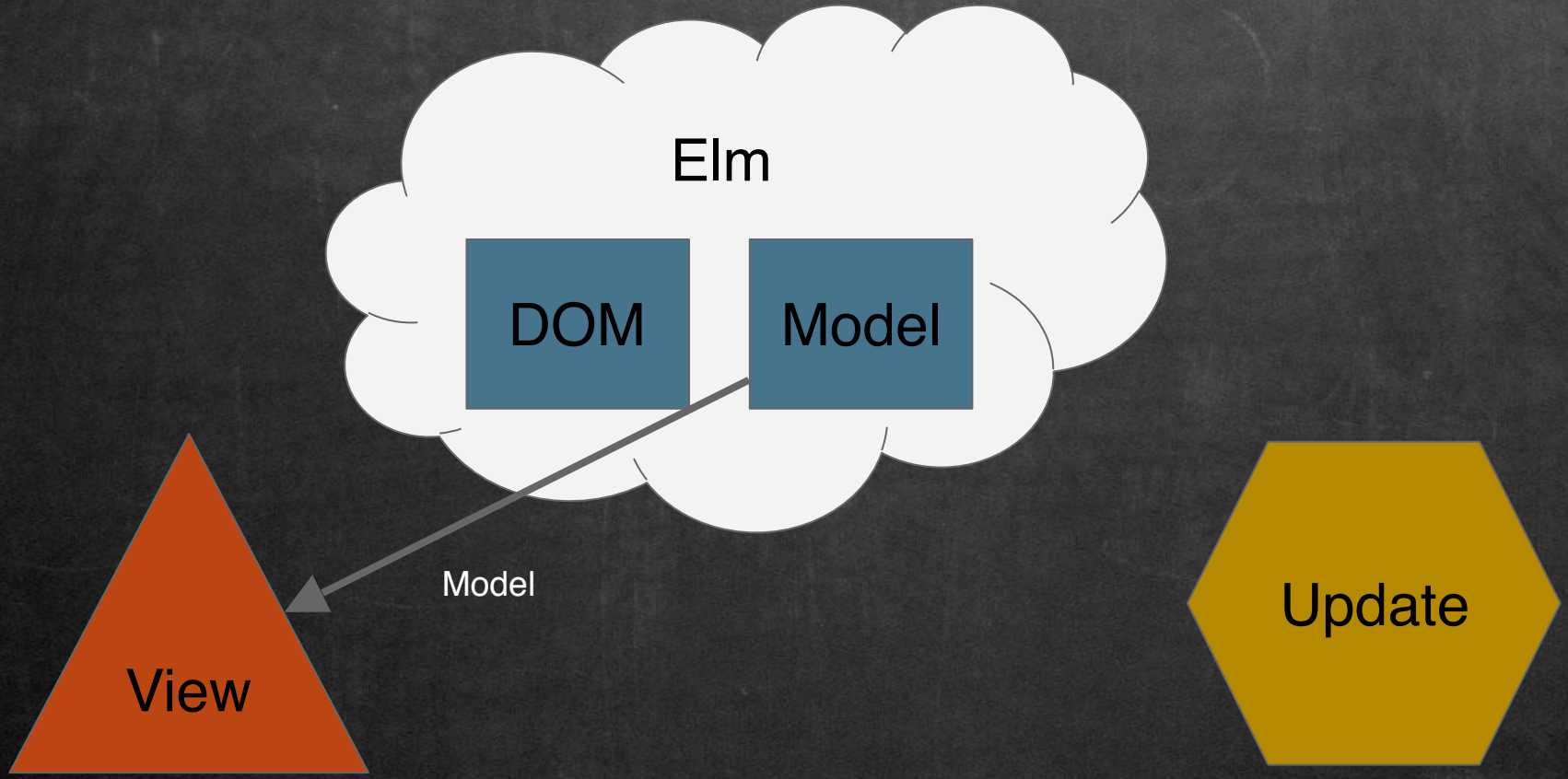
Benefits

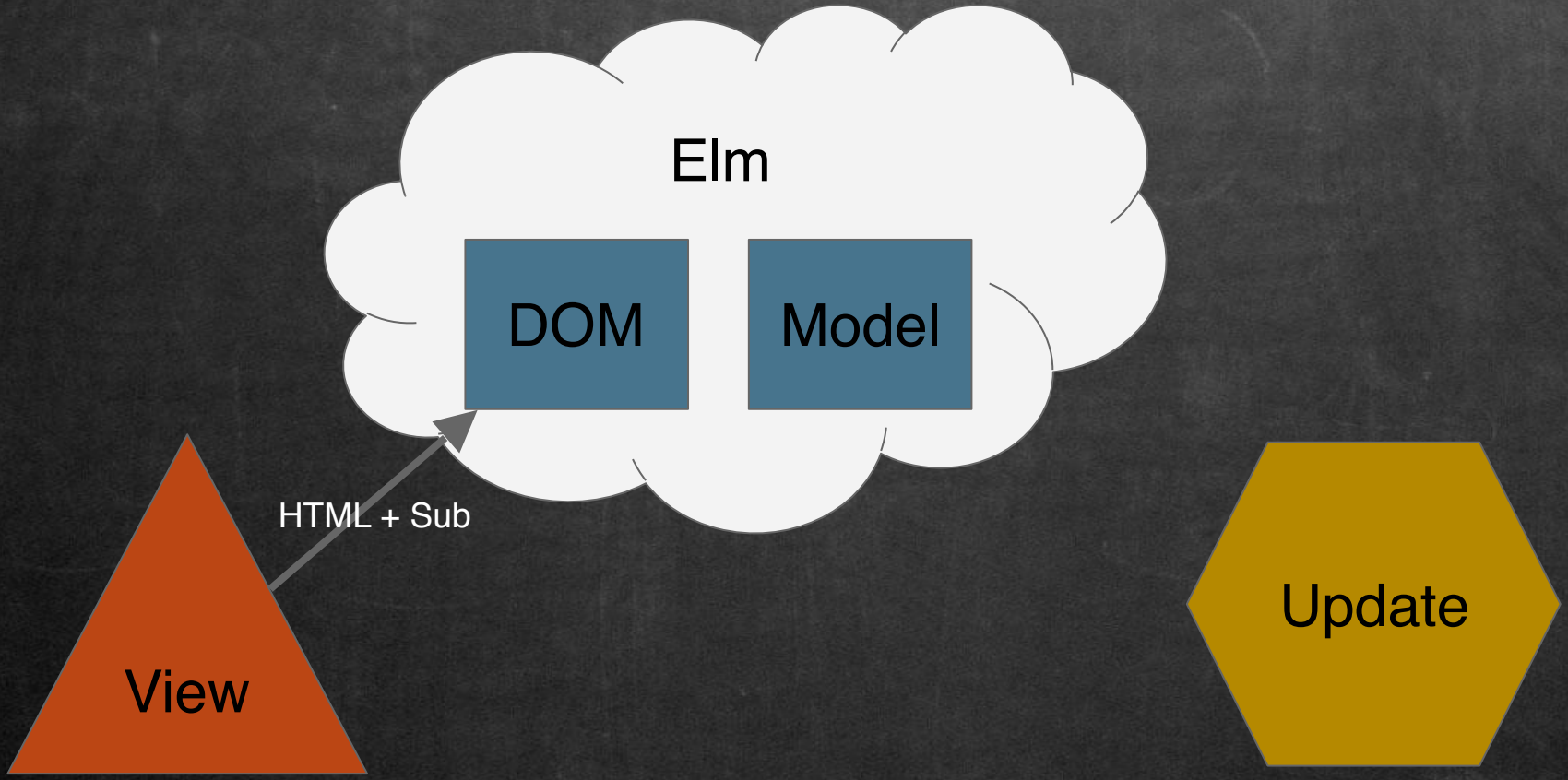
- testing of stateless / pure functions is easy
- testing without global state is easy
- better maintainability
- no concurrency problems

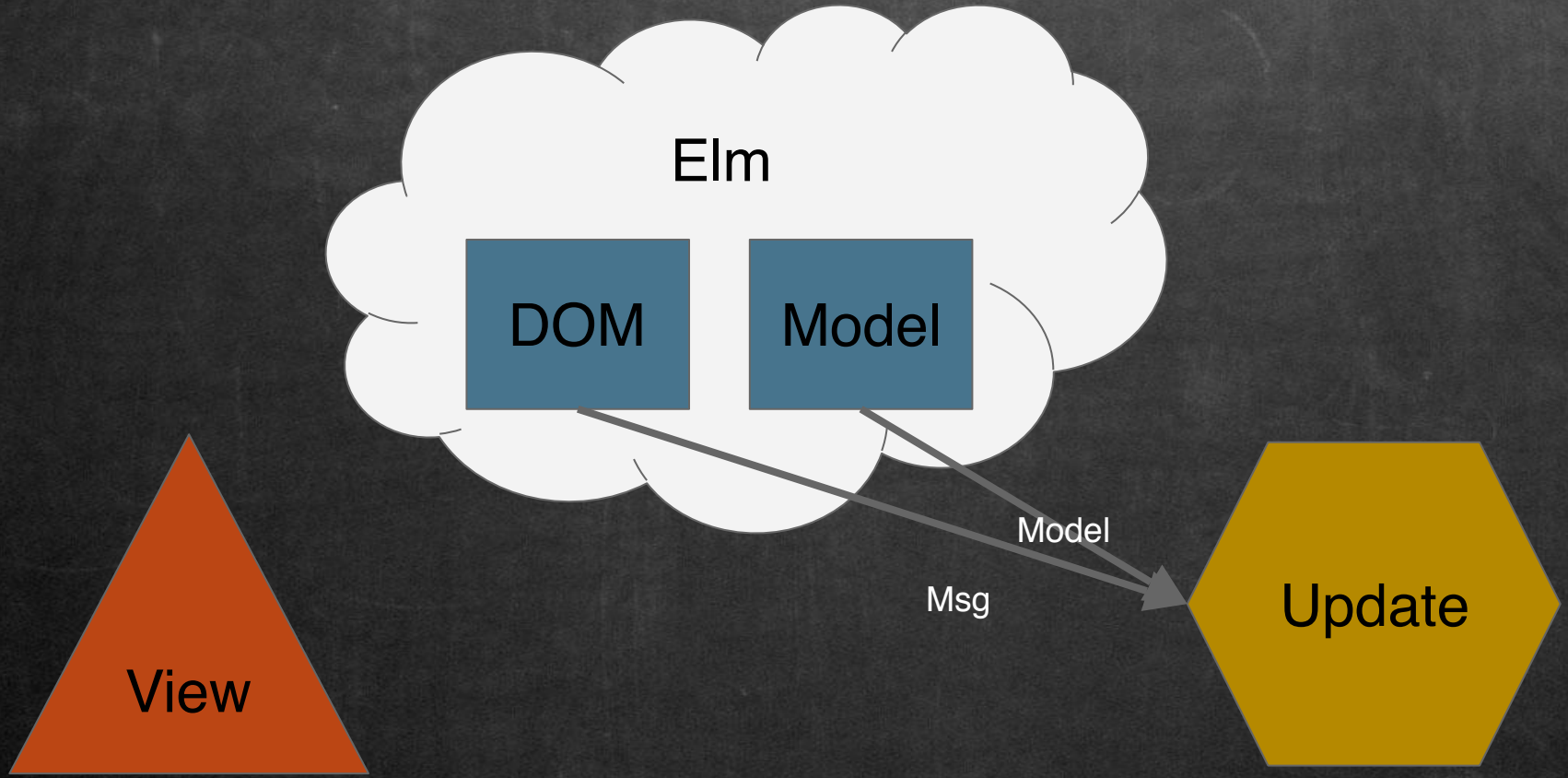
- Time Travel Debugger (build-in)(debug.elm-lang.org)

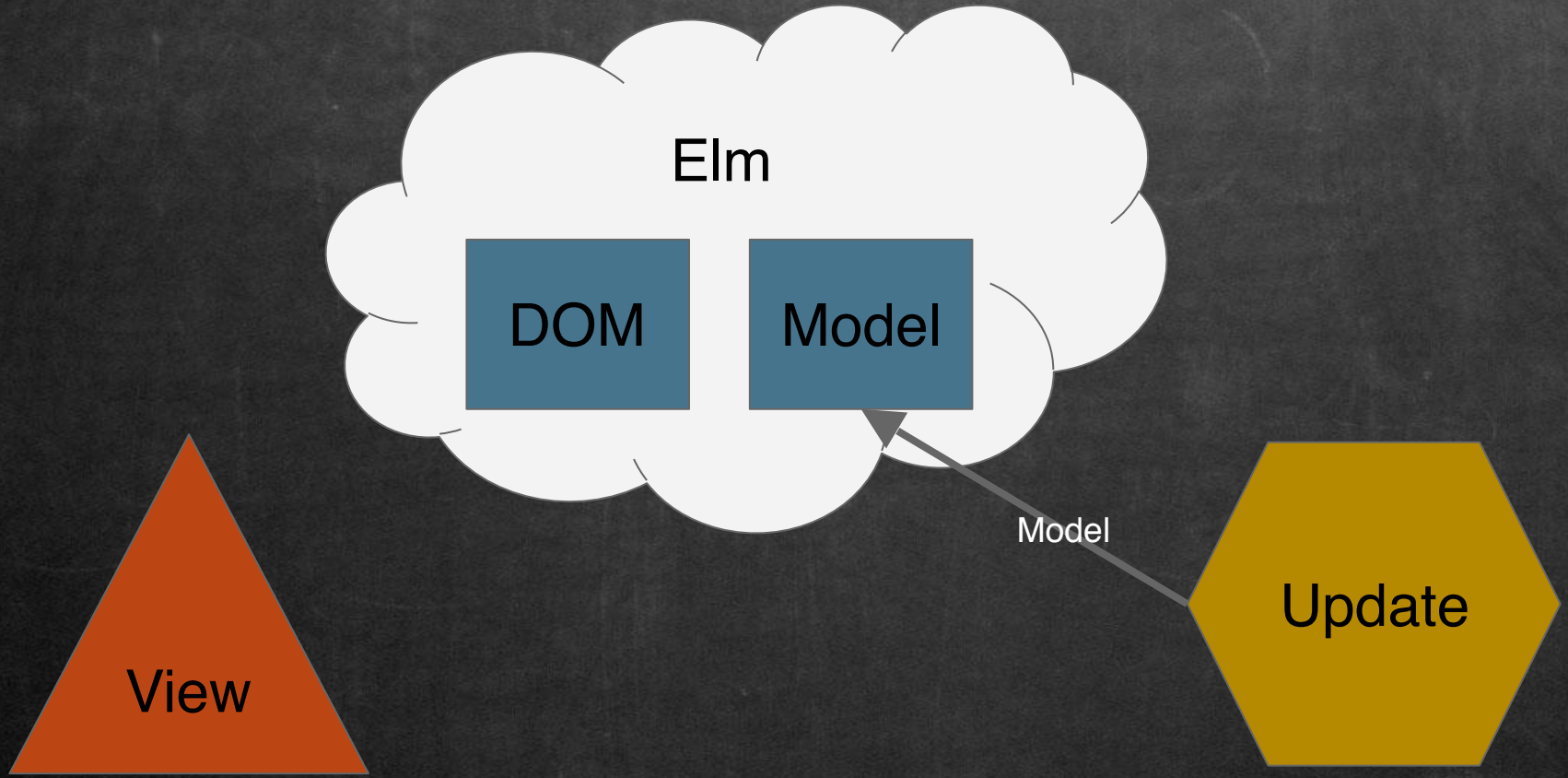
Reactive

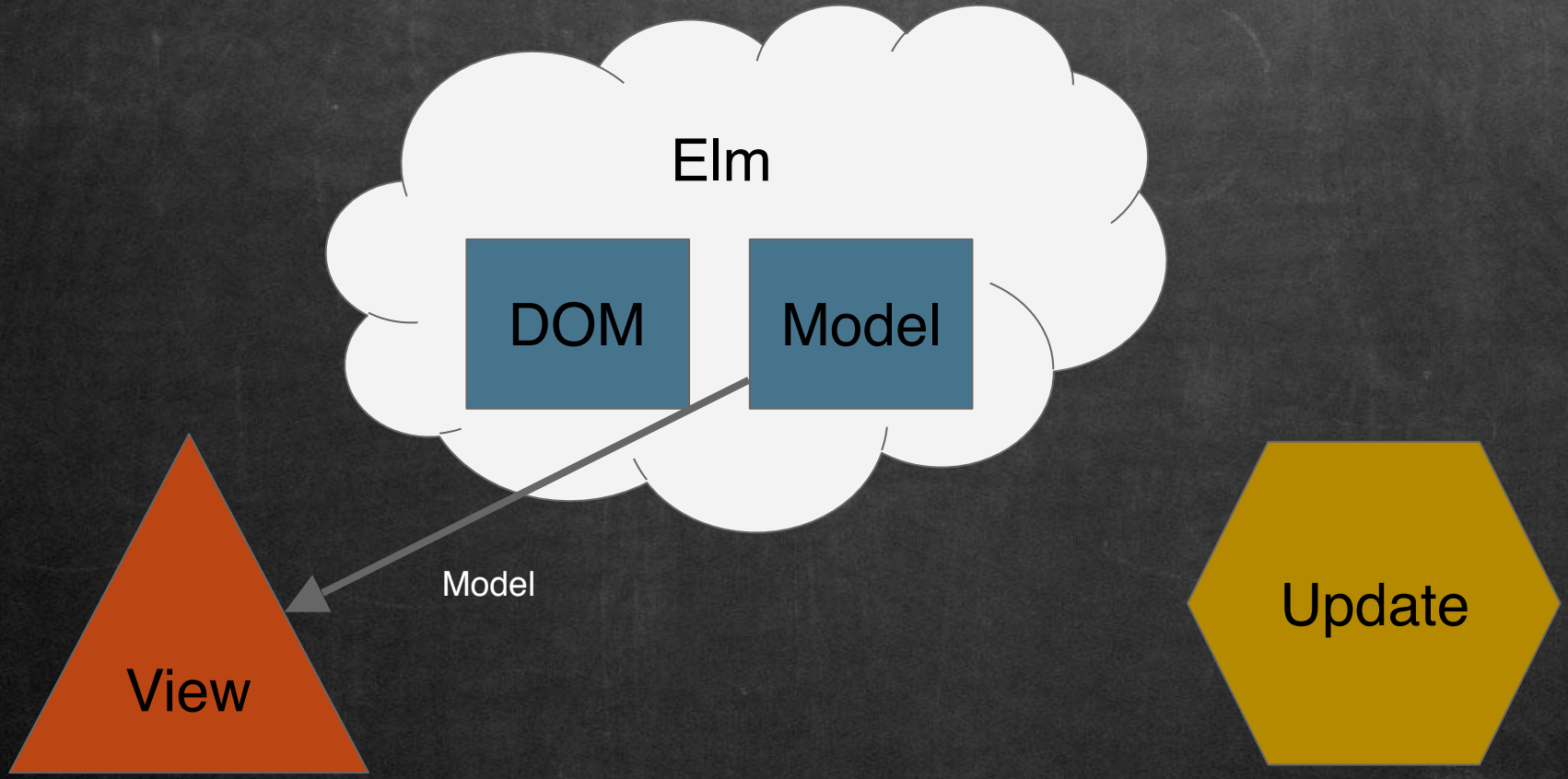

Elm
DOM
Model
View
Update

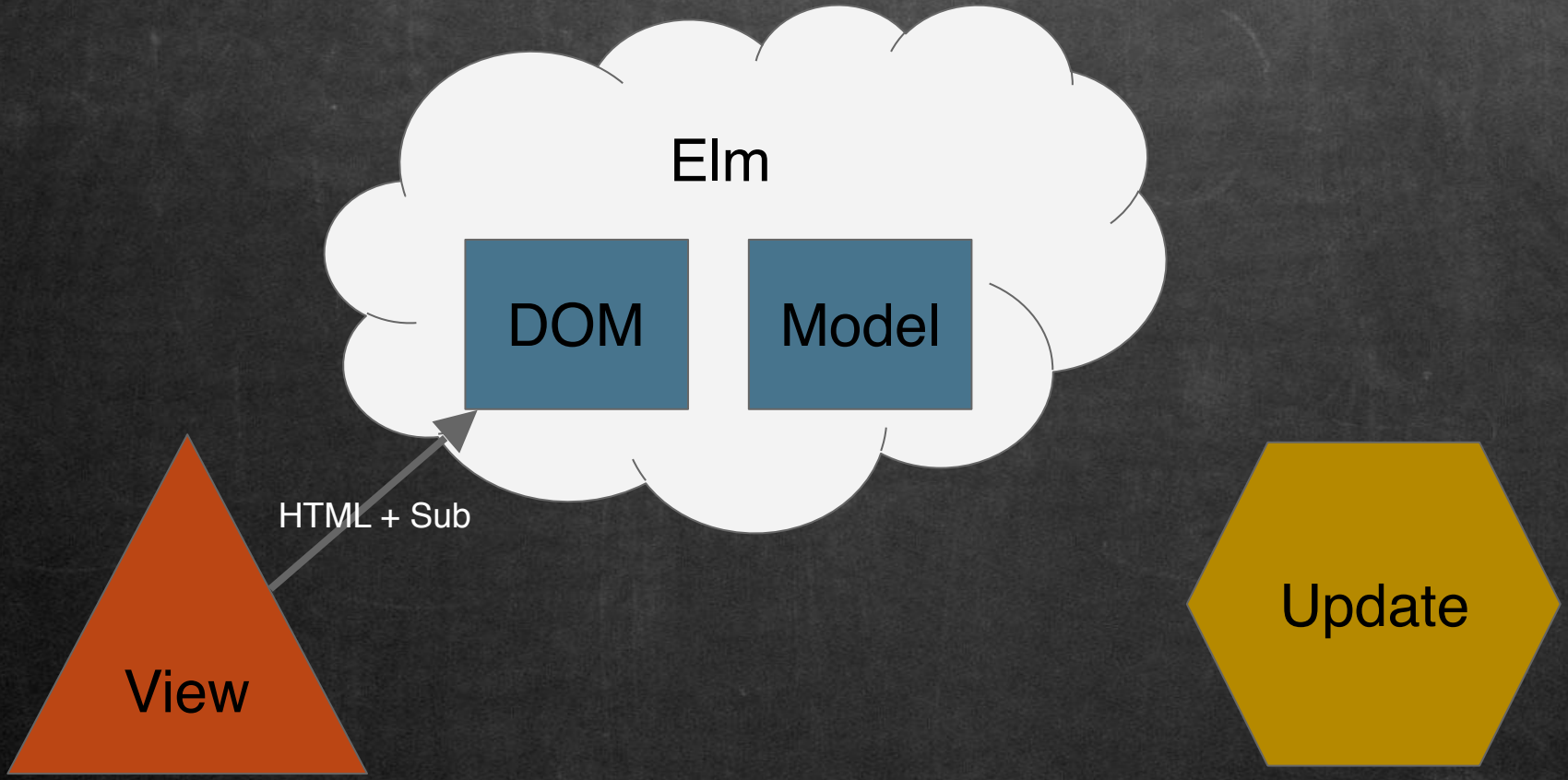

Elm
DOM
Model
Model
View
Update

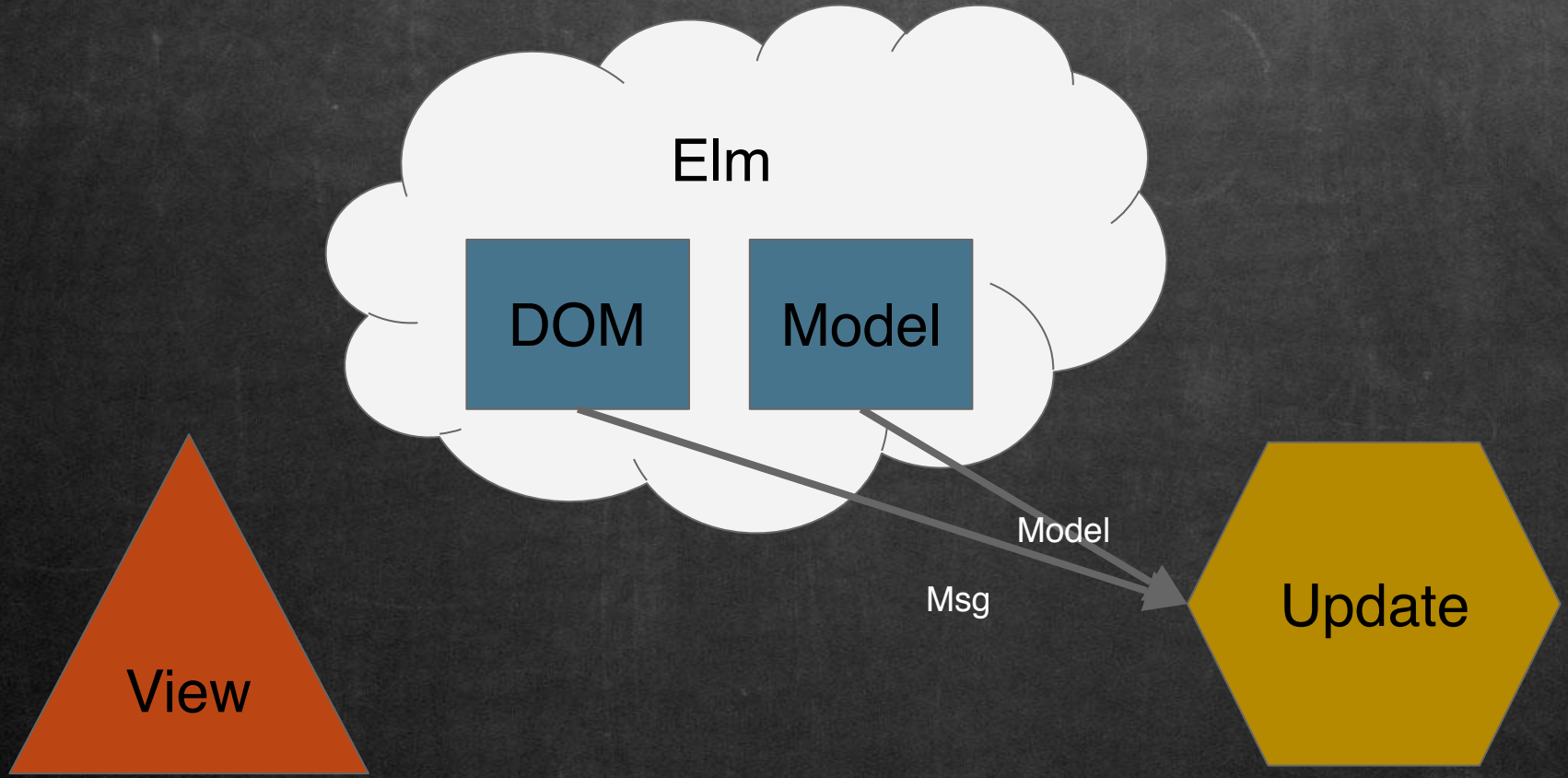

Elm
DOM
Model
HTML + Sub
View
Update

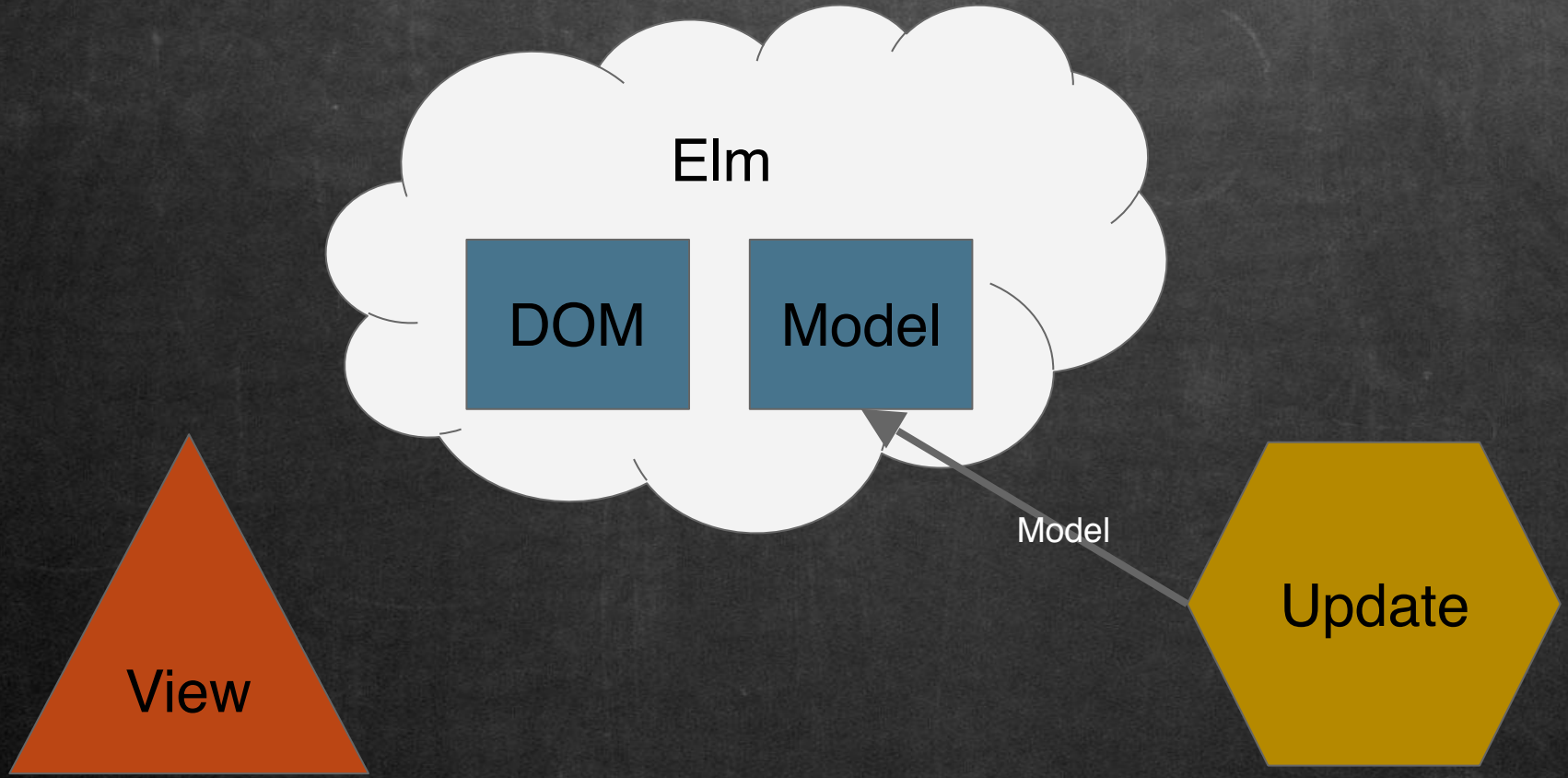

Elm
DOM
Model
Model
Msg
View
Update

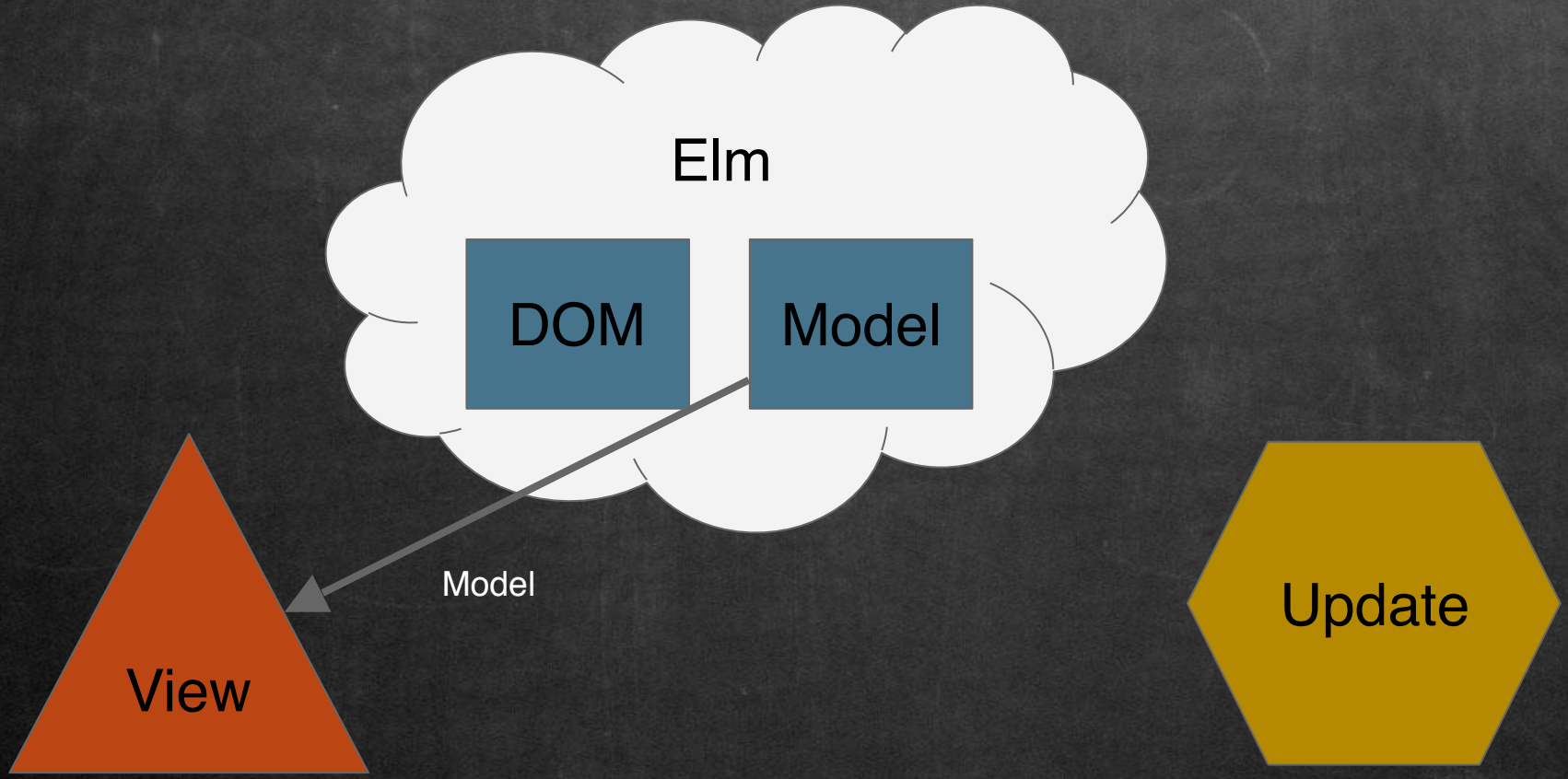

Elm
DOM
Model
Model
View
Update

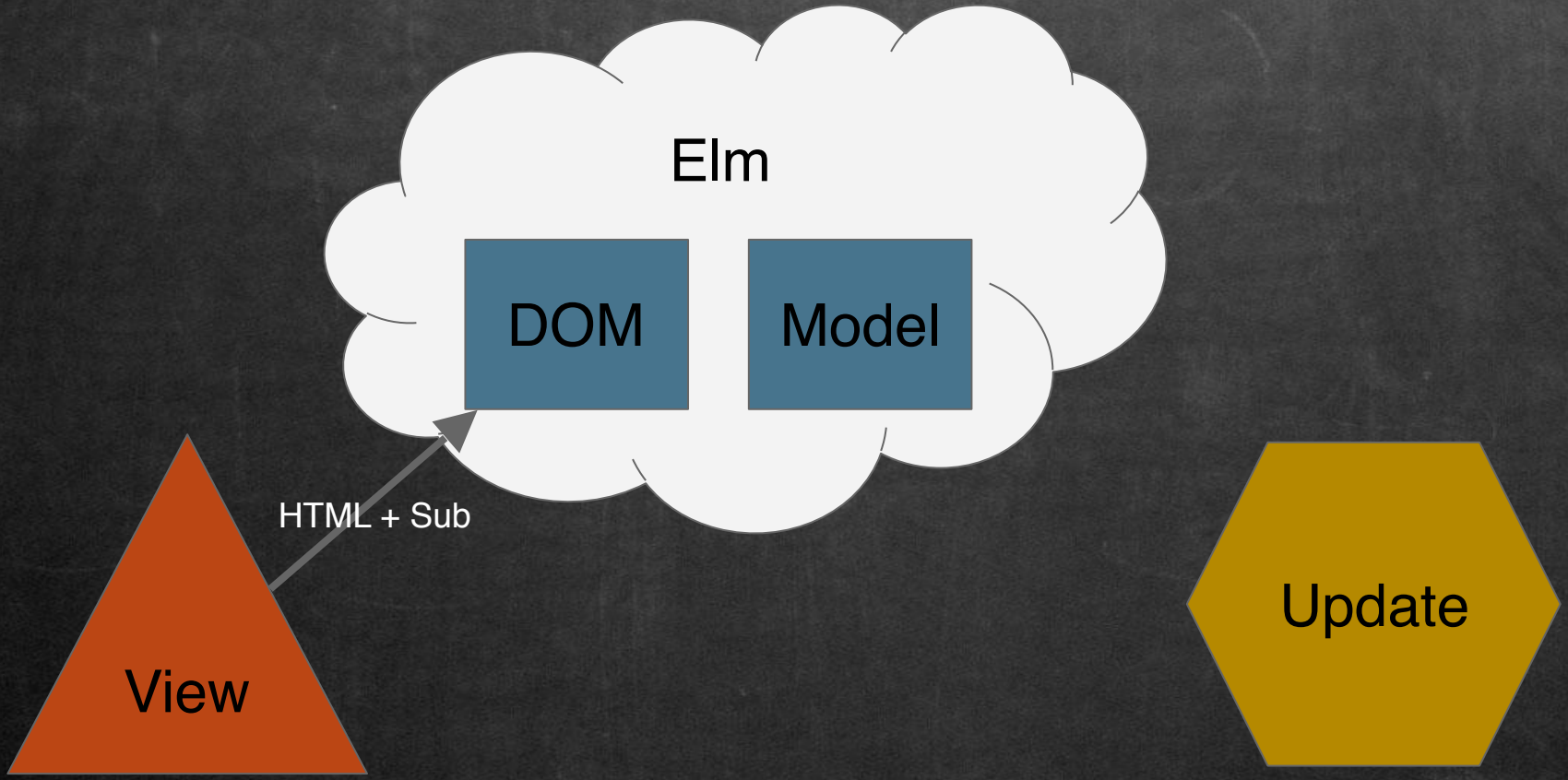

Elm
DOM
Model
Model
View
Update


Elm
DOM
Model
HTML + Sub
View
Update


Elm
DOM
Model
Model
Msg
View
Update


Elm
DOM
Model
Model
View
Update


Elm
DOM
Model
Model
View
Update


Elm
DOM
Model
HTML + Sub
View
Update

- driven by events (messages)
- view function delivers not only HTML but ‘subscriptions’ to DOM events

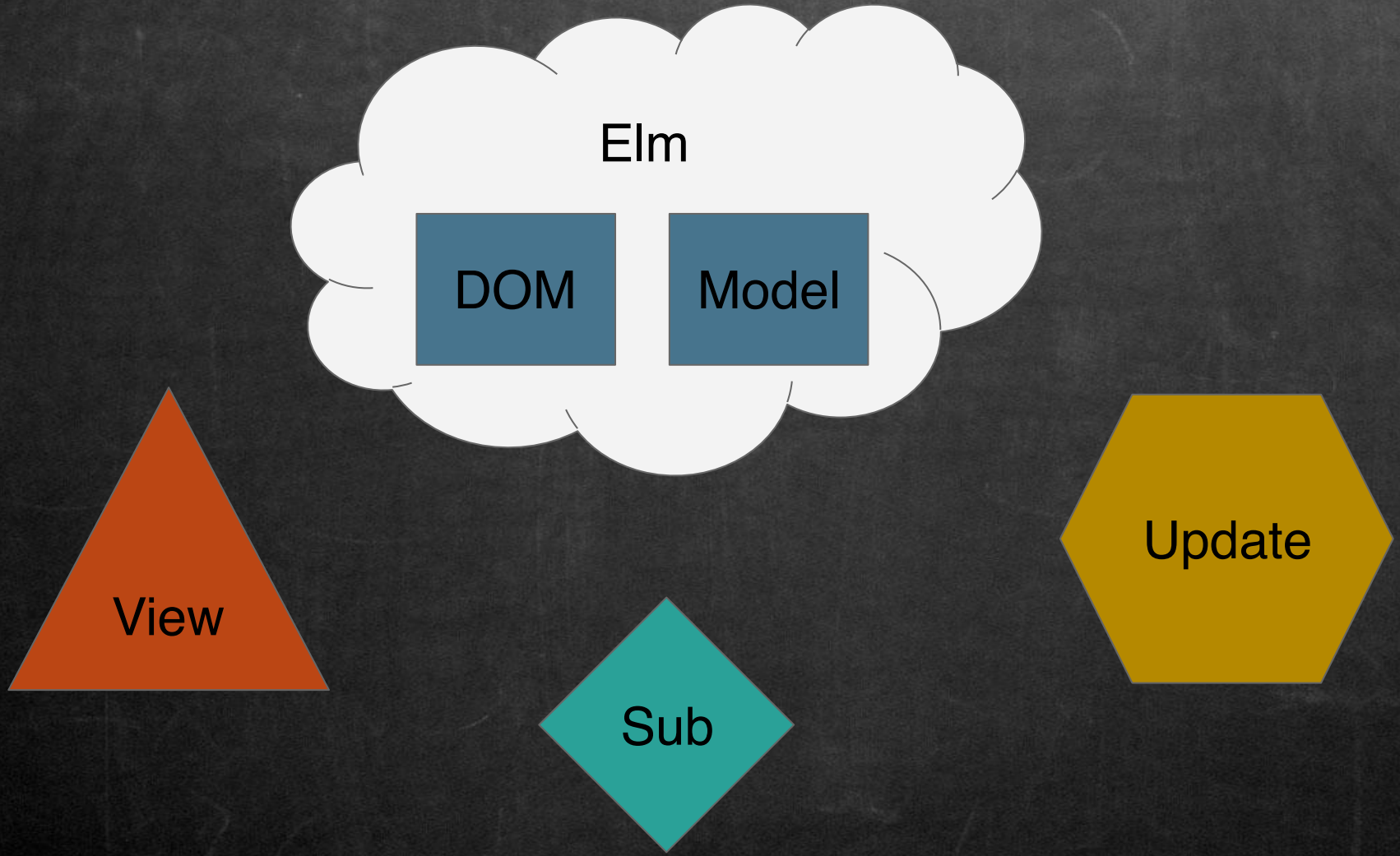
```
view : Model -> Html Msg
view model =
  div []
      [ span [] [text (toString model)]
      , button [ onClick Add ] [ text "+" ]
      , button [ onClick Sub ] [ text "-" ]
      ]
```

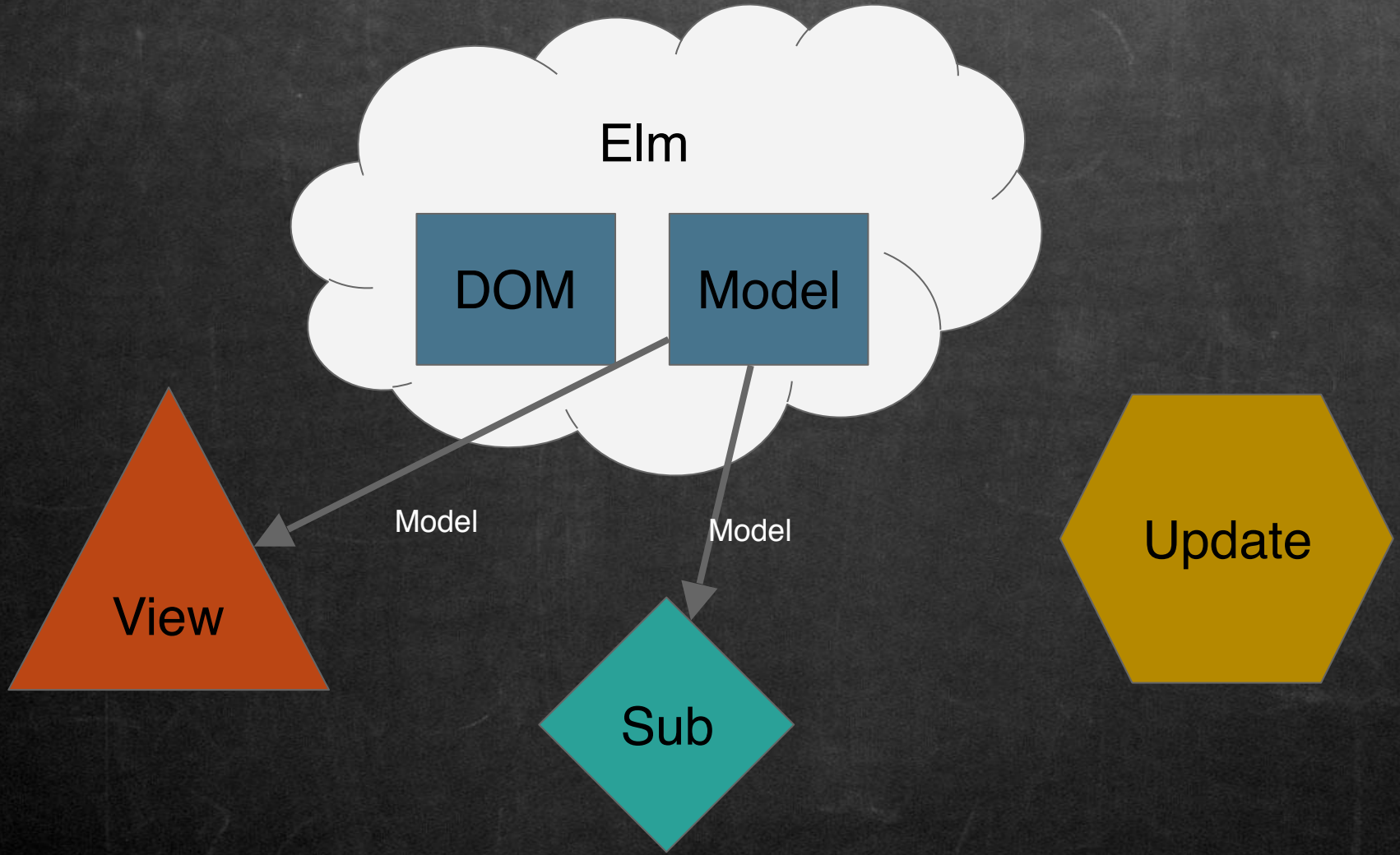
How to subscribe to other events?

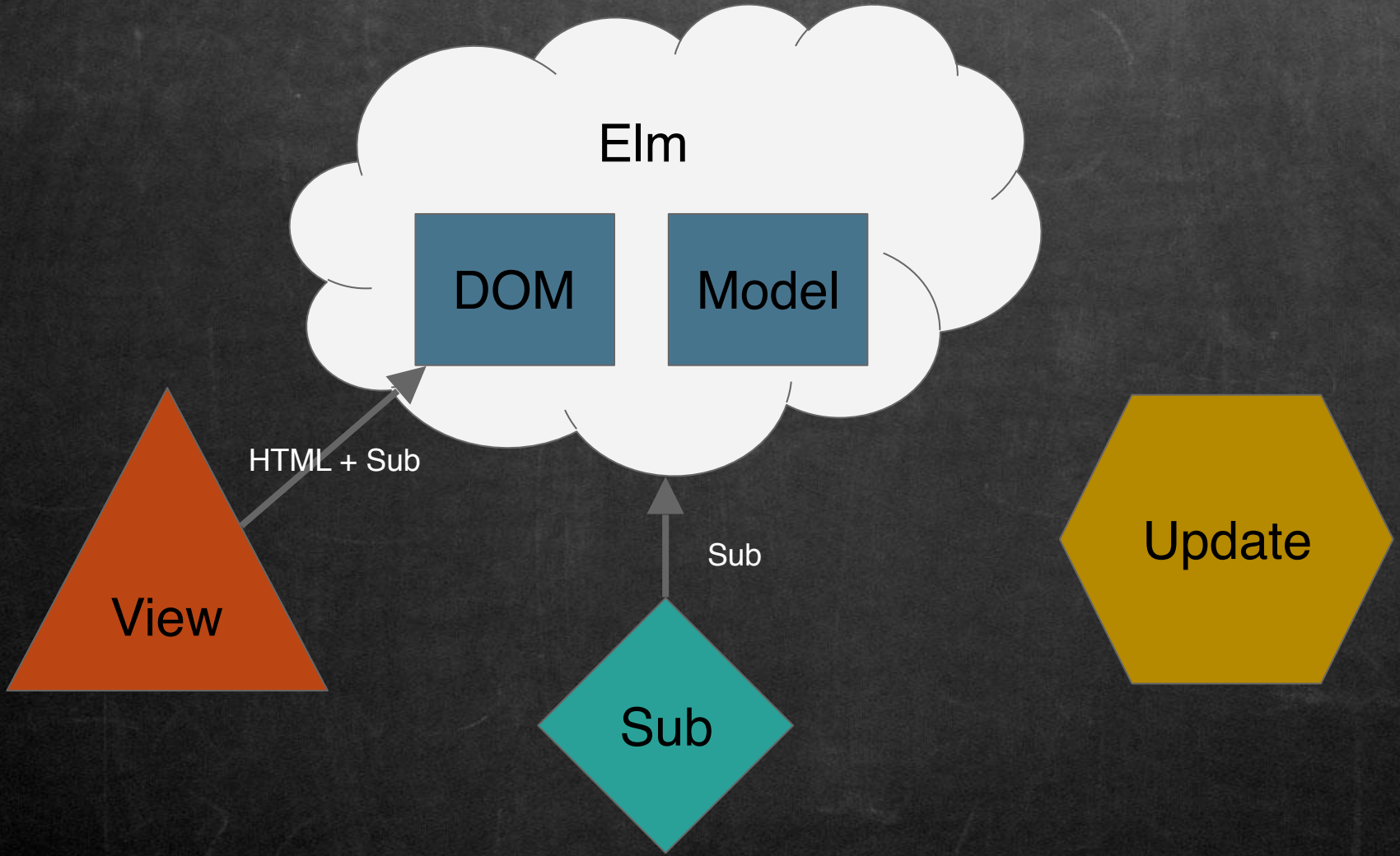
(WebSockets, URL changes, ...)

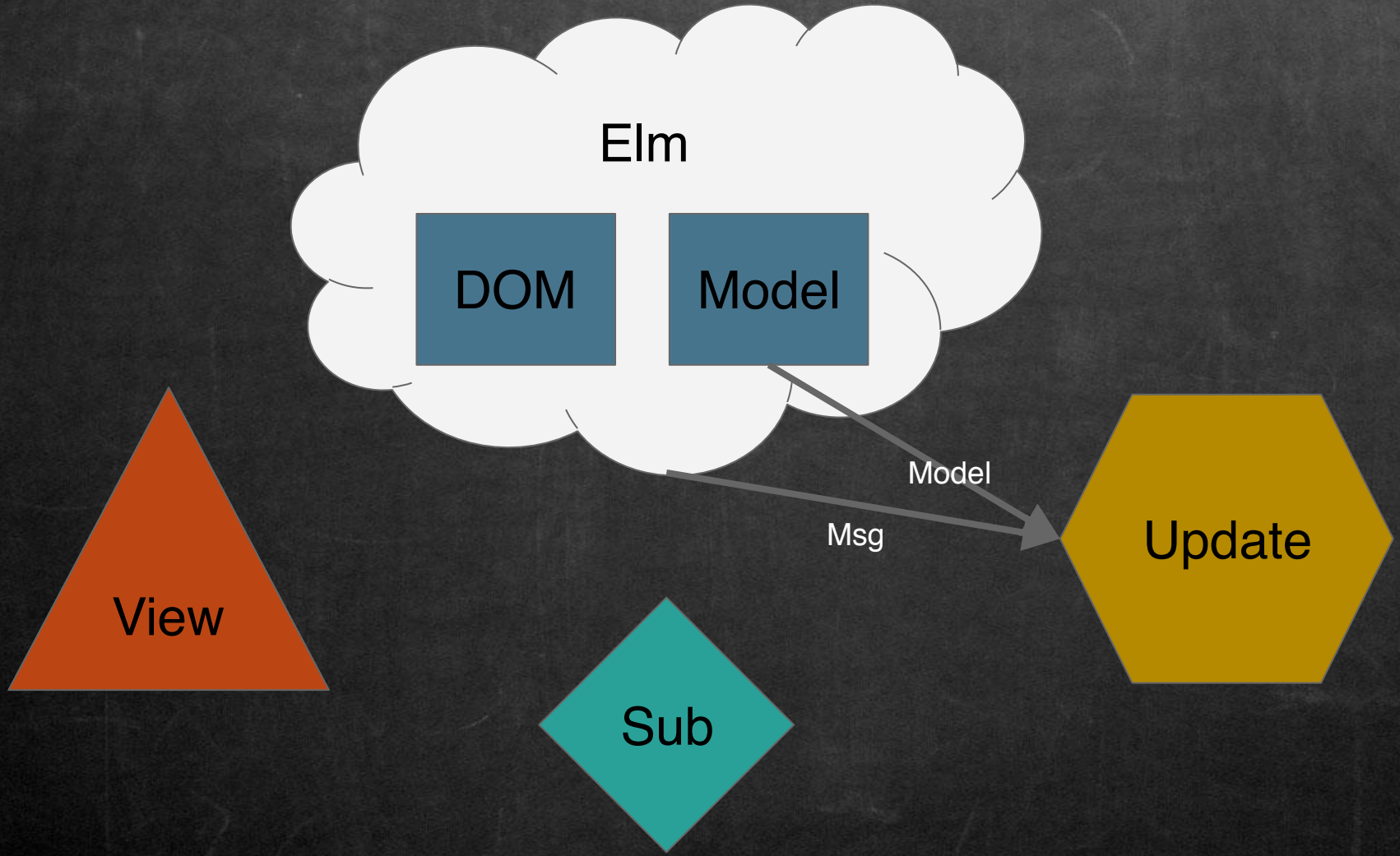
```elm
type alias Model = Time


type Msg
  = Tick Time


init : (Model, Cmd Msg)
init =
  (0, Cmd.none)


update : Msg -> Model -> (Model, Cmd Msg)
update msg model =
  case msg of
    Tick time ->
      (time, Cmd.none)
```

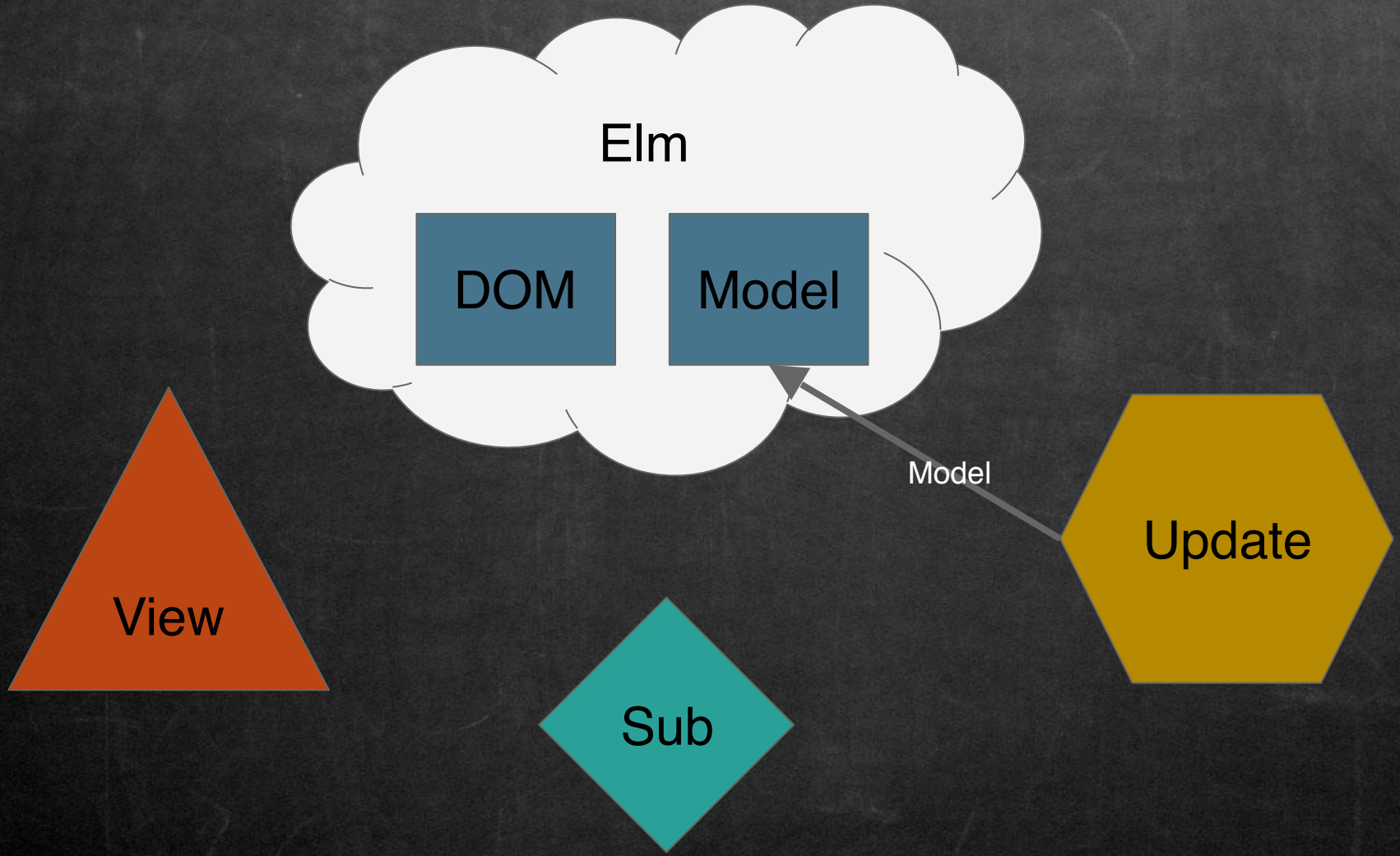
```elm
view : Model -> Html Msg
view model =
  div []
    [ span [] [text (model |> toString)]
    ]

subscriptions model =
    Time.every Time.second Tick
```

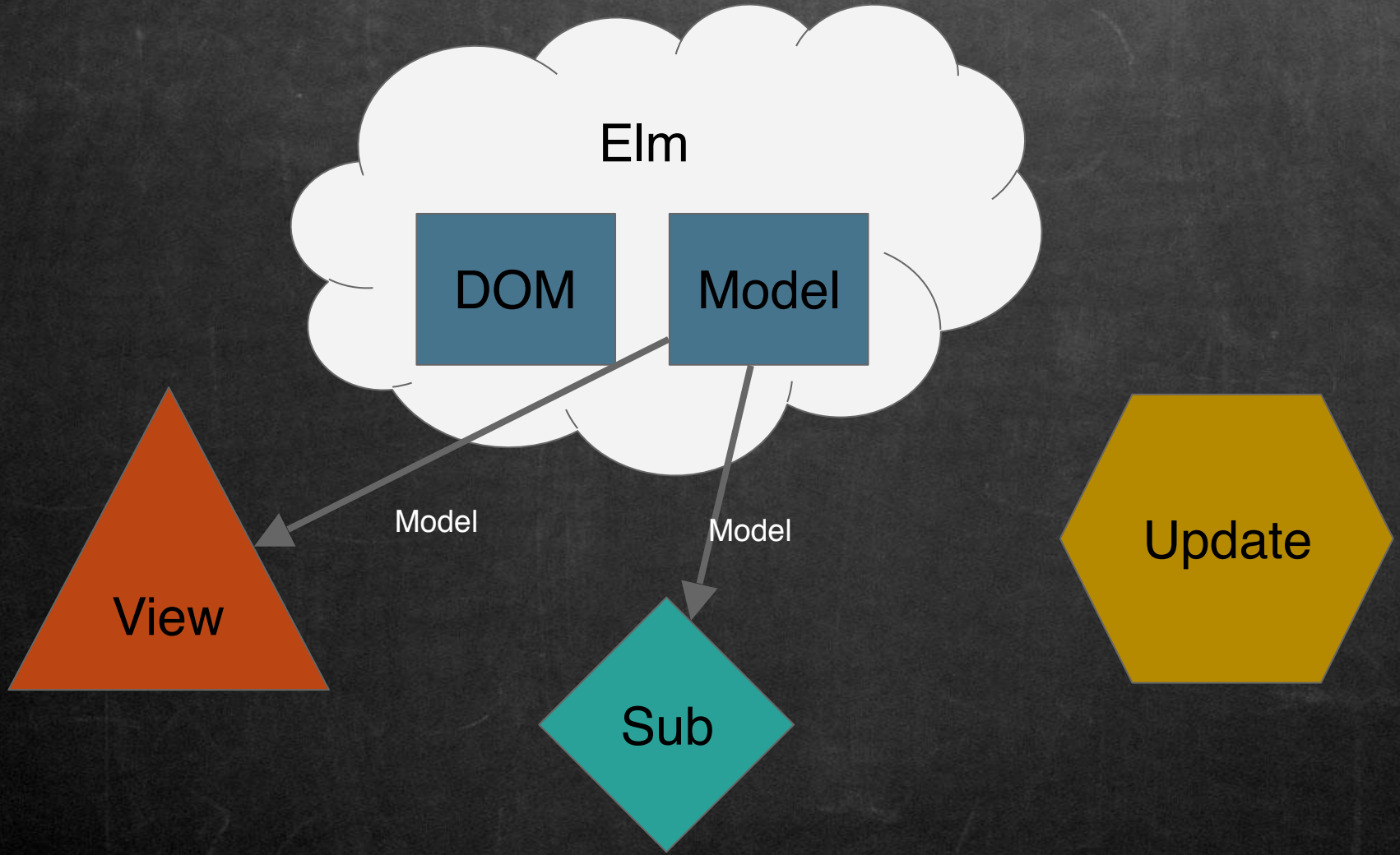

Elm
DOM
Model
View
Sub
Update

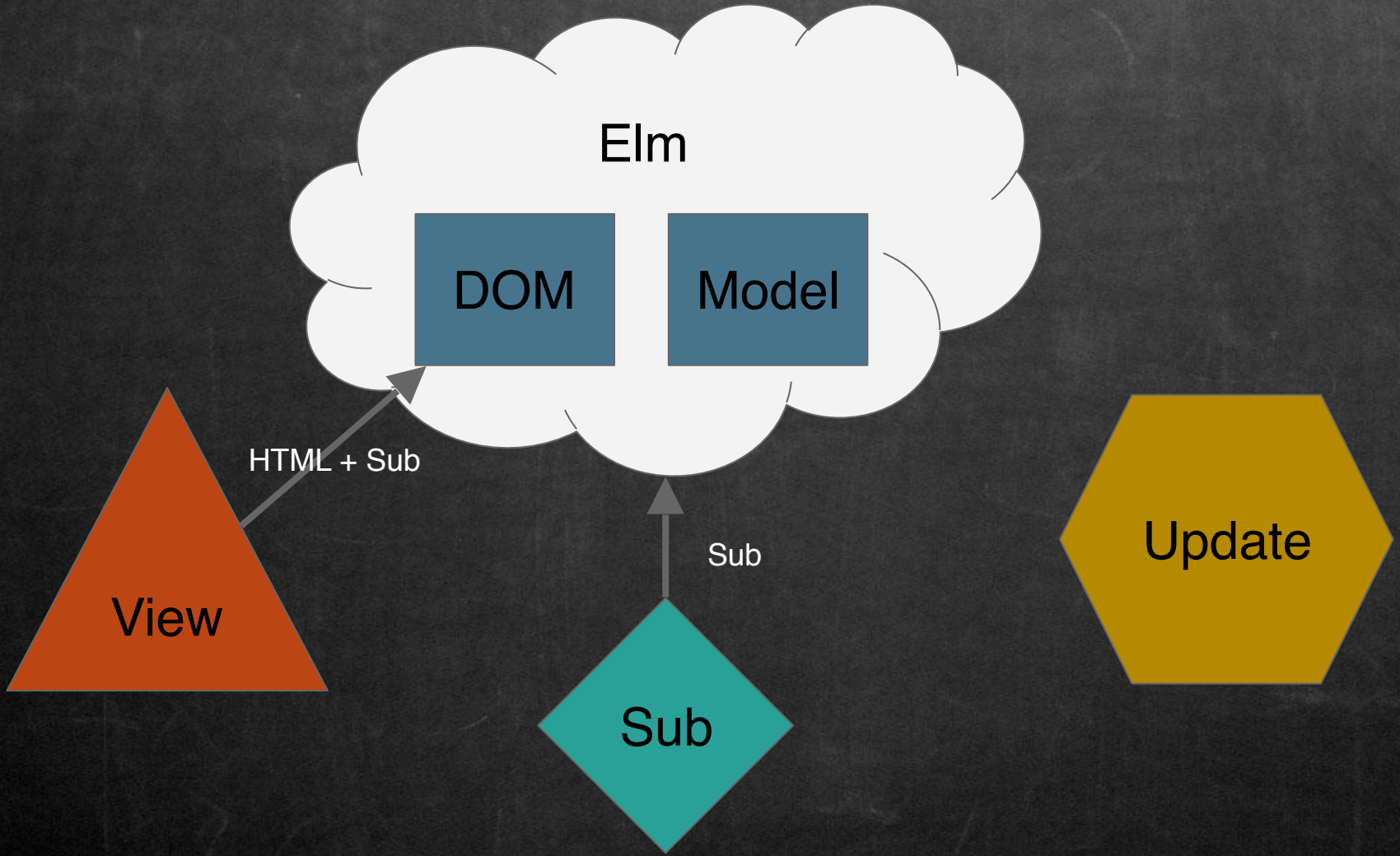

Elm
DOM
Model
Model
Model
View
Sub
Update


Elm
DOM
Model
HTML + Sub
Sub
View
Sub
Update


Elm
DOM
Model
Model
Msg
Update
View
Sub


Elm
DOM
Model
Model
Update
View
Sub


Elm
DOM
Model
Model
Model
View
Sub
Update

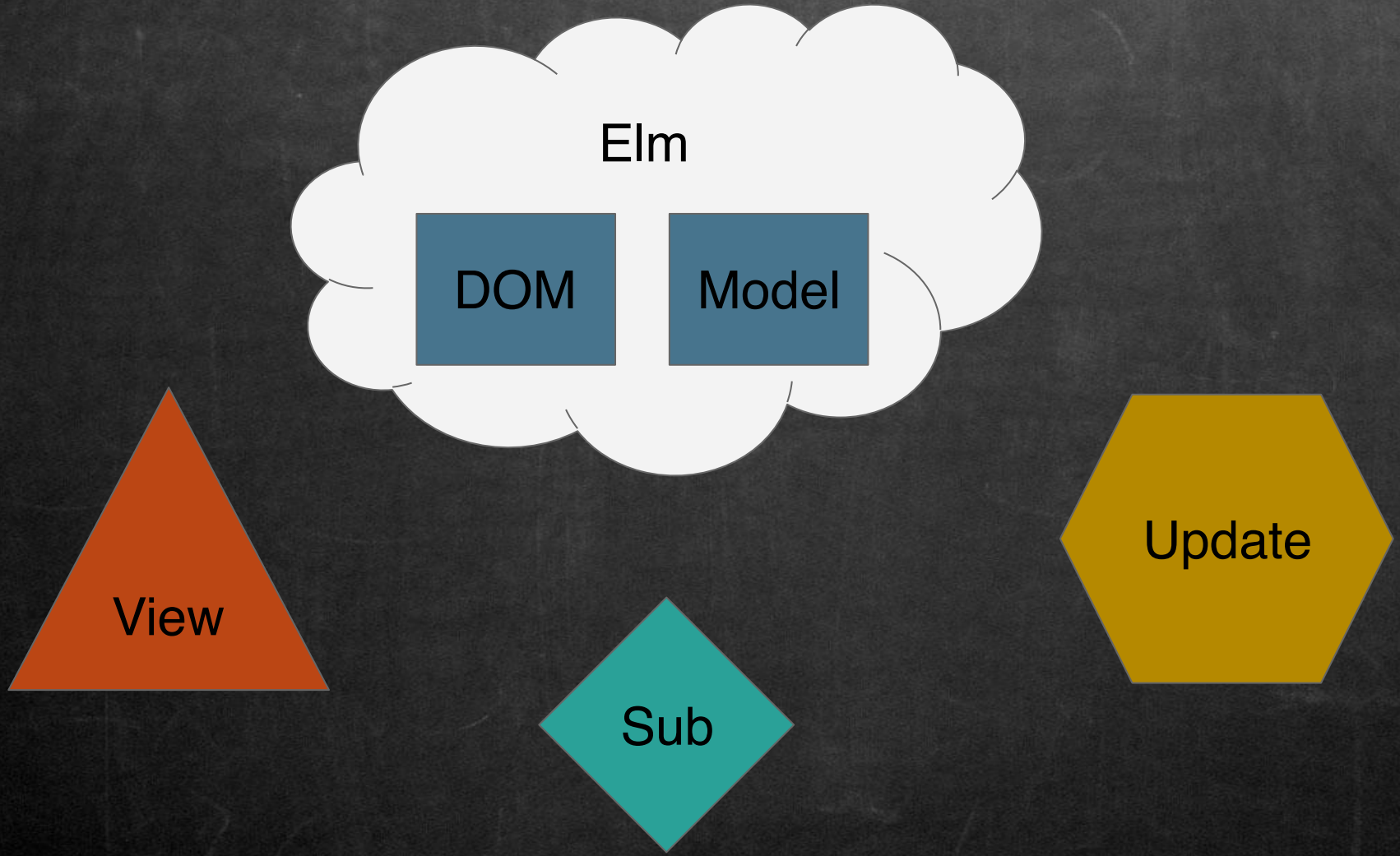

Elm
DOM
Model
HTML + Sub
Sub
View
Sub
Update

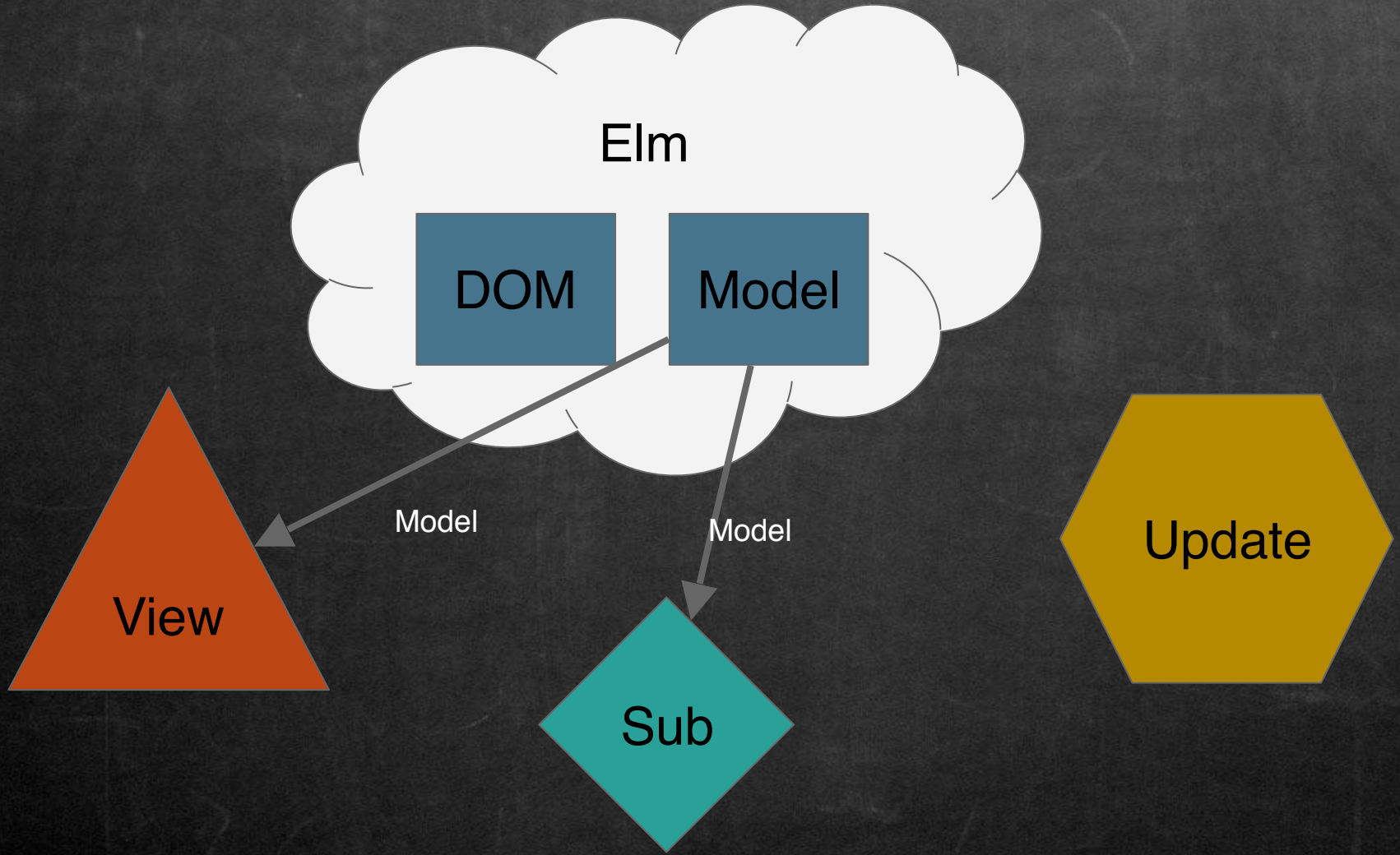
Side effects

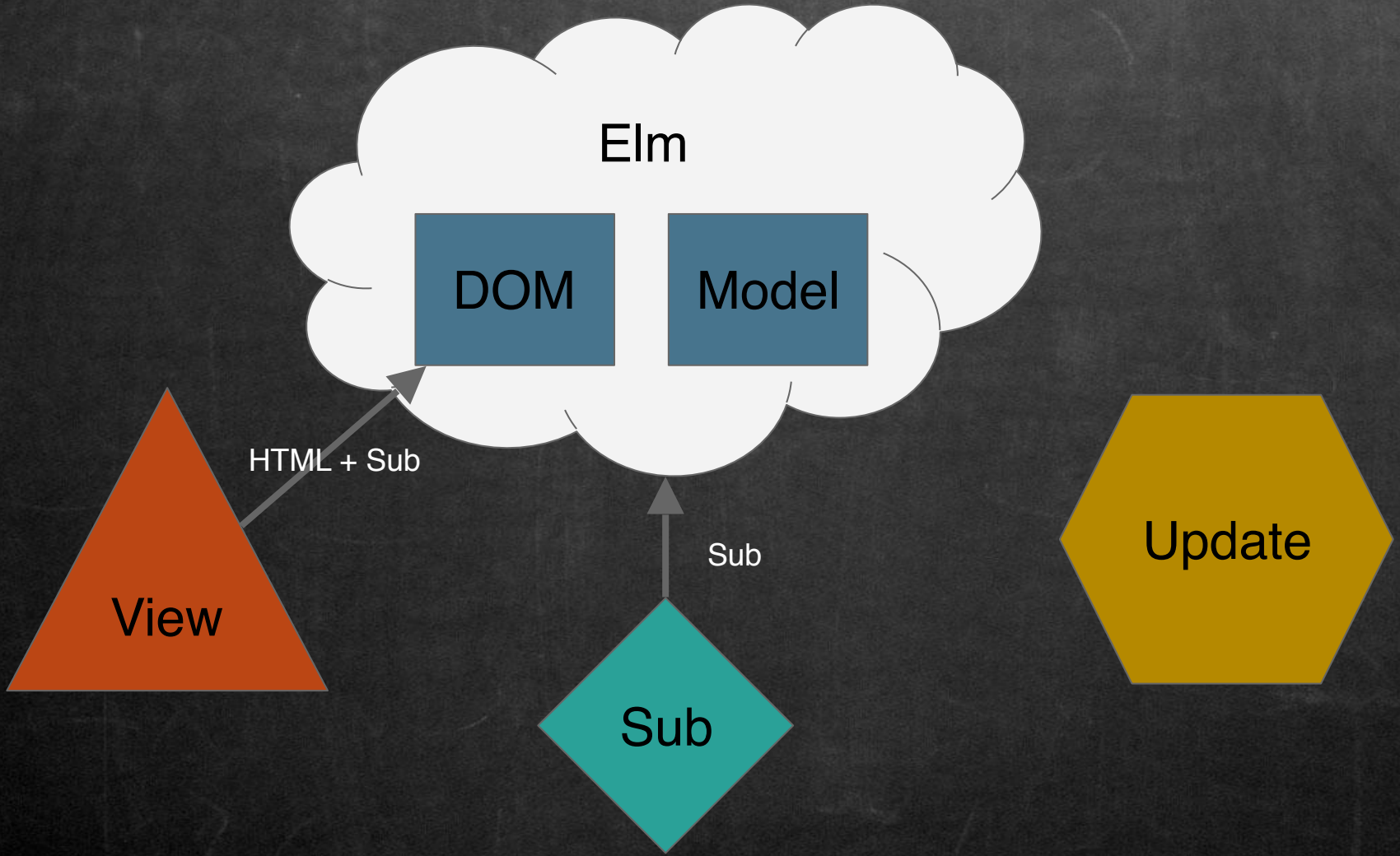
How to work with side effects?

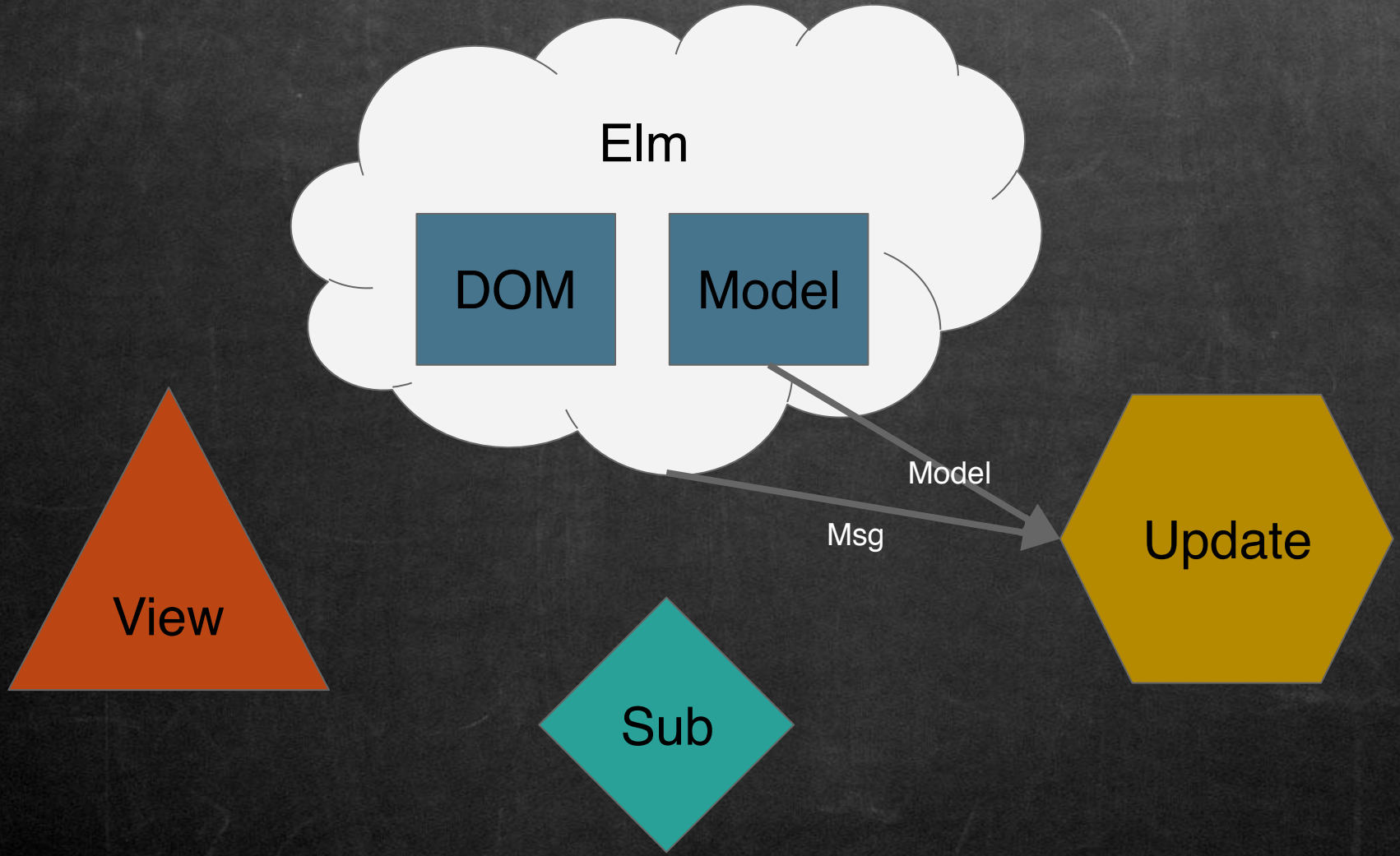
(REST call, Websocket send, ...)

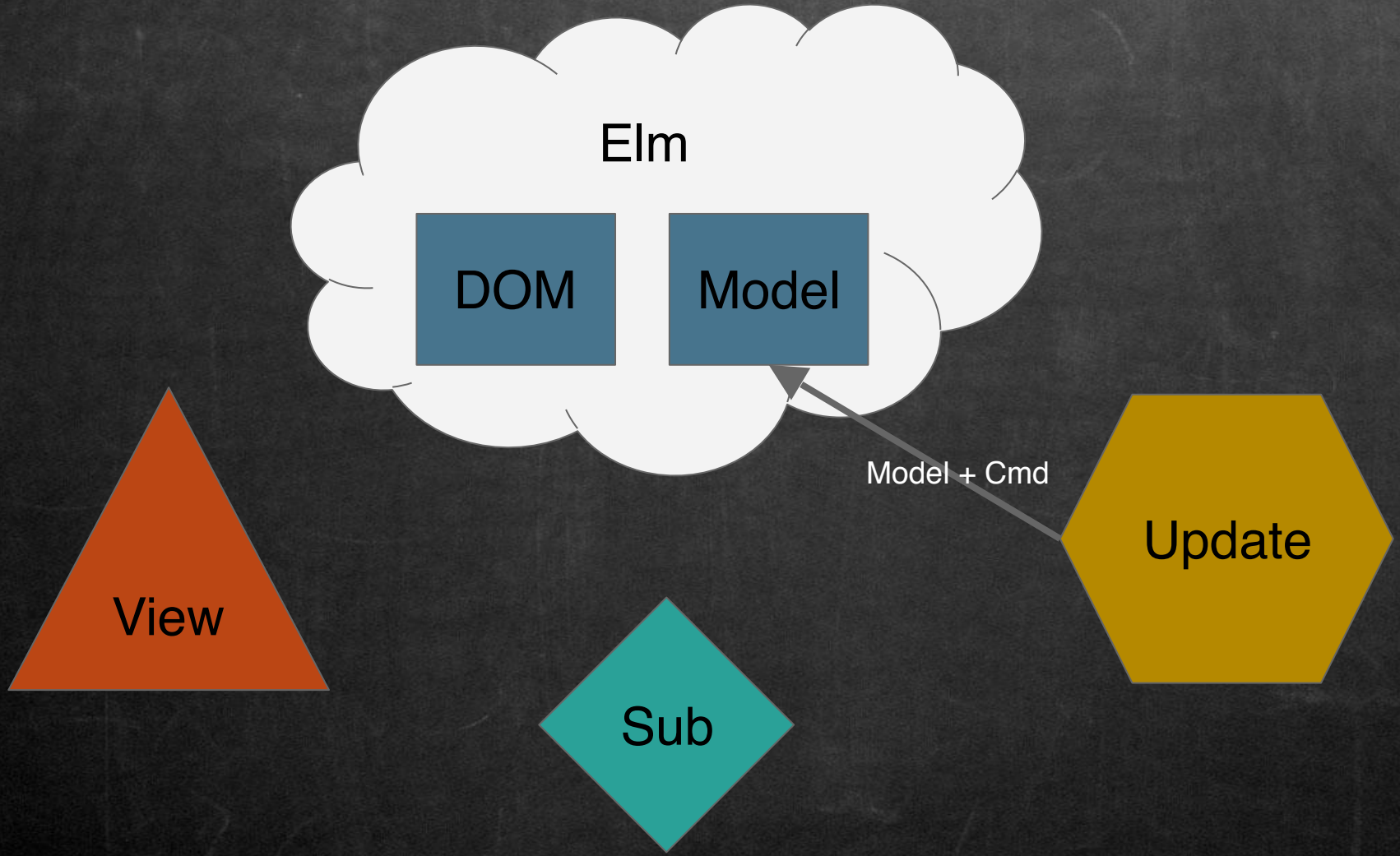
```elm
update : Msg -> Model -> ( Model, Cmd Msg )
update msg model =
  case msg of
    ReqRndVal ->
      ( model, Random.generate NewRndVal (Random.int 1 100) )

    NewRndVal value ->
      ( value, Cmd.none )


view : Model -> Html Msg
view model =
  div []
    [ span [] [ text (toString model) ]
    , button [ onClick ReqRndVal ]
             [ text "New random value" ]
    ]
```

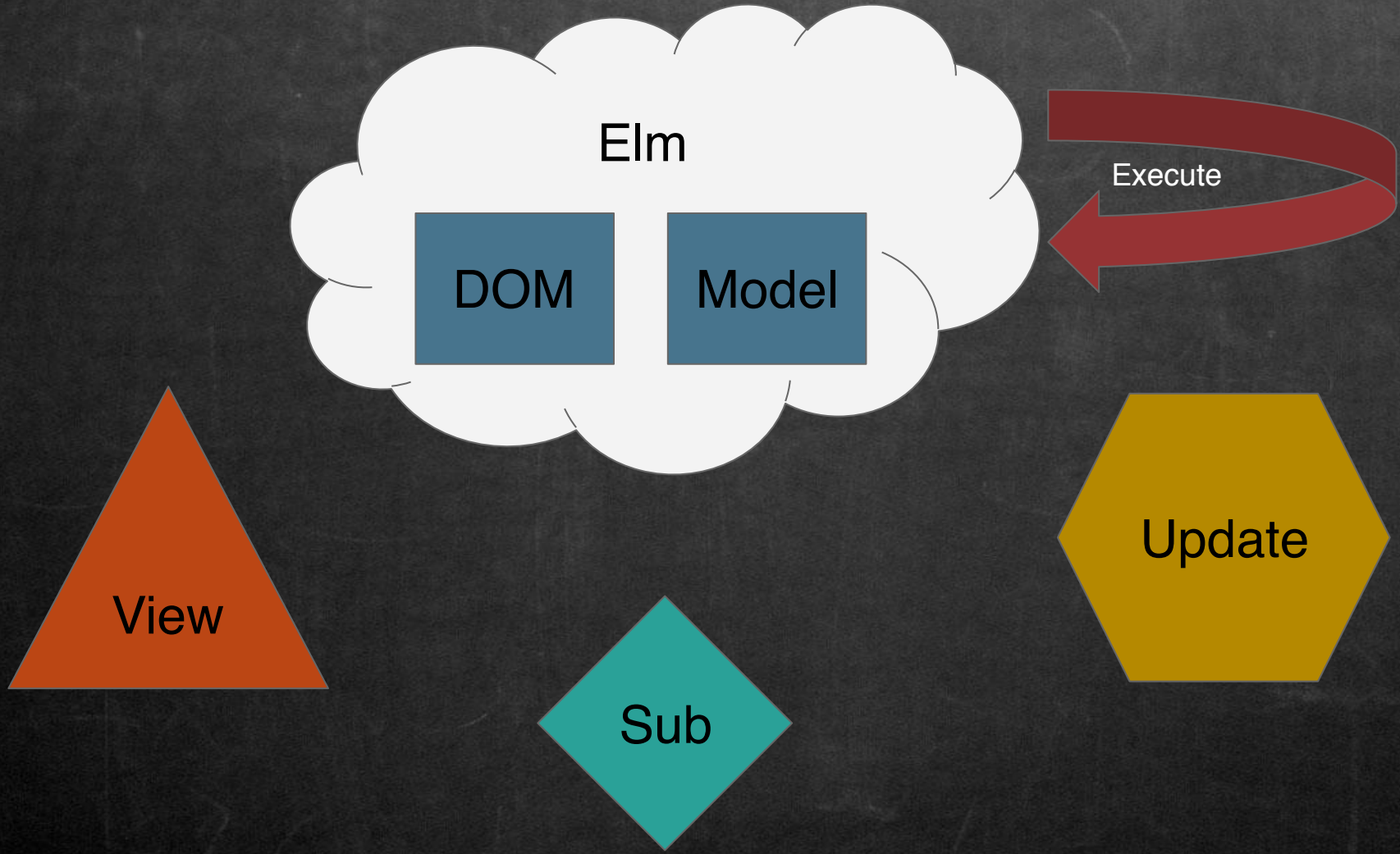

Elm
DOM
Model
View
Sub
Update

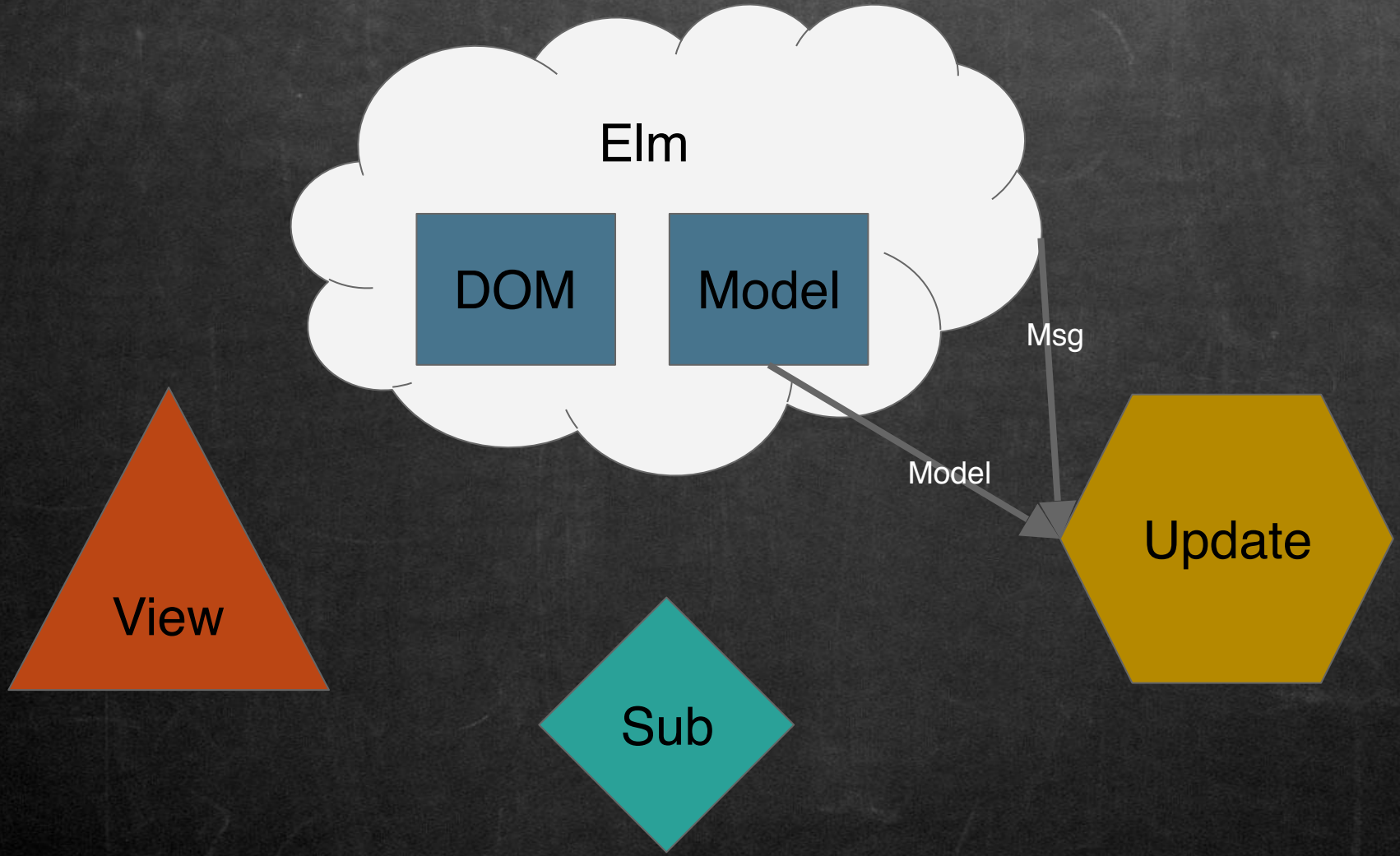

Elm
DOM
Model
Model
Model
View
Sub
Update

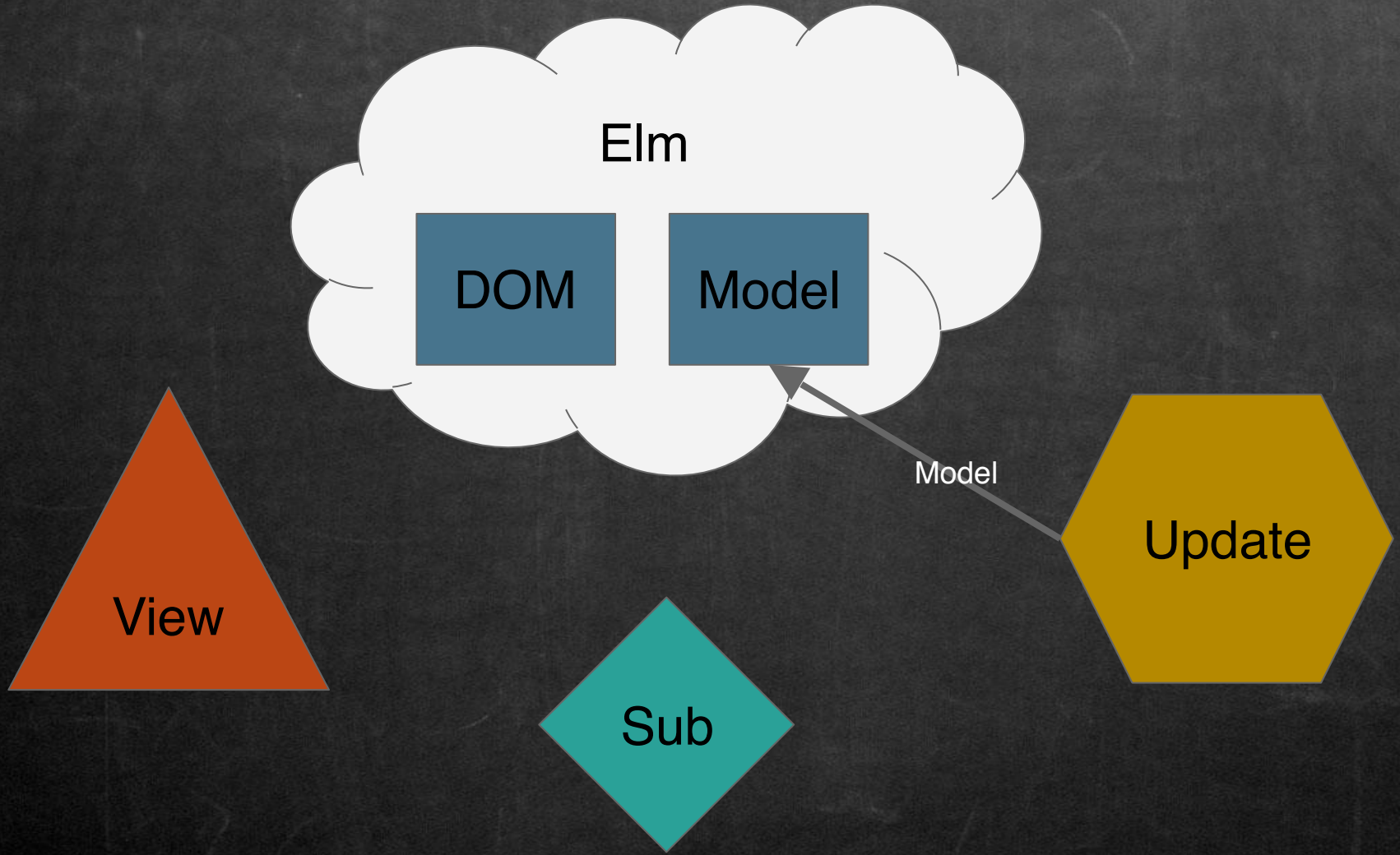

Elm
DOM
Model
HTML + Sub
Sub
View
Sub
Update

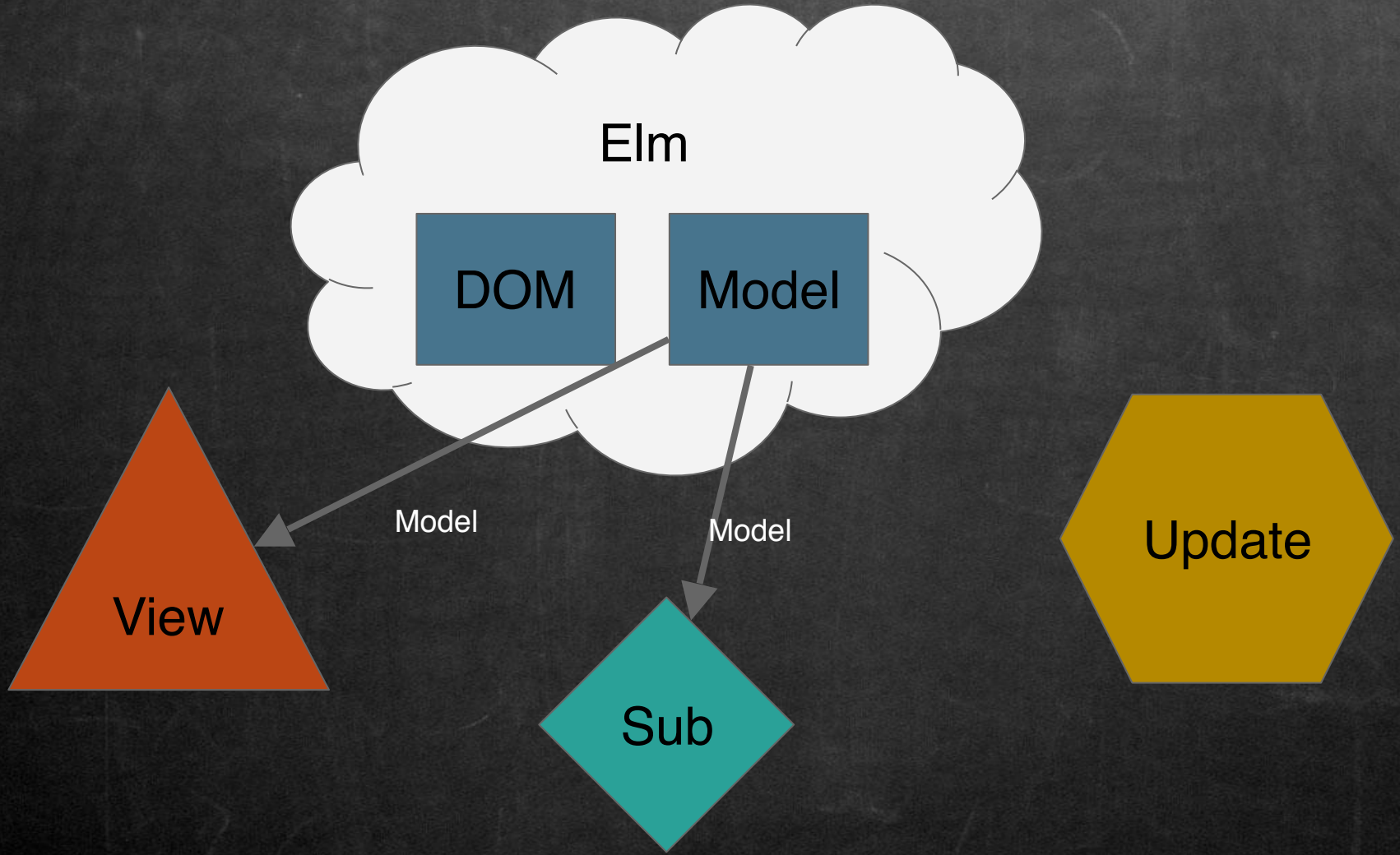

Elm
DOM
Model
Model
Msg
Update
View
Sub

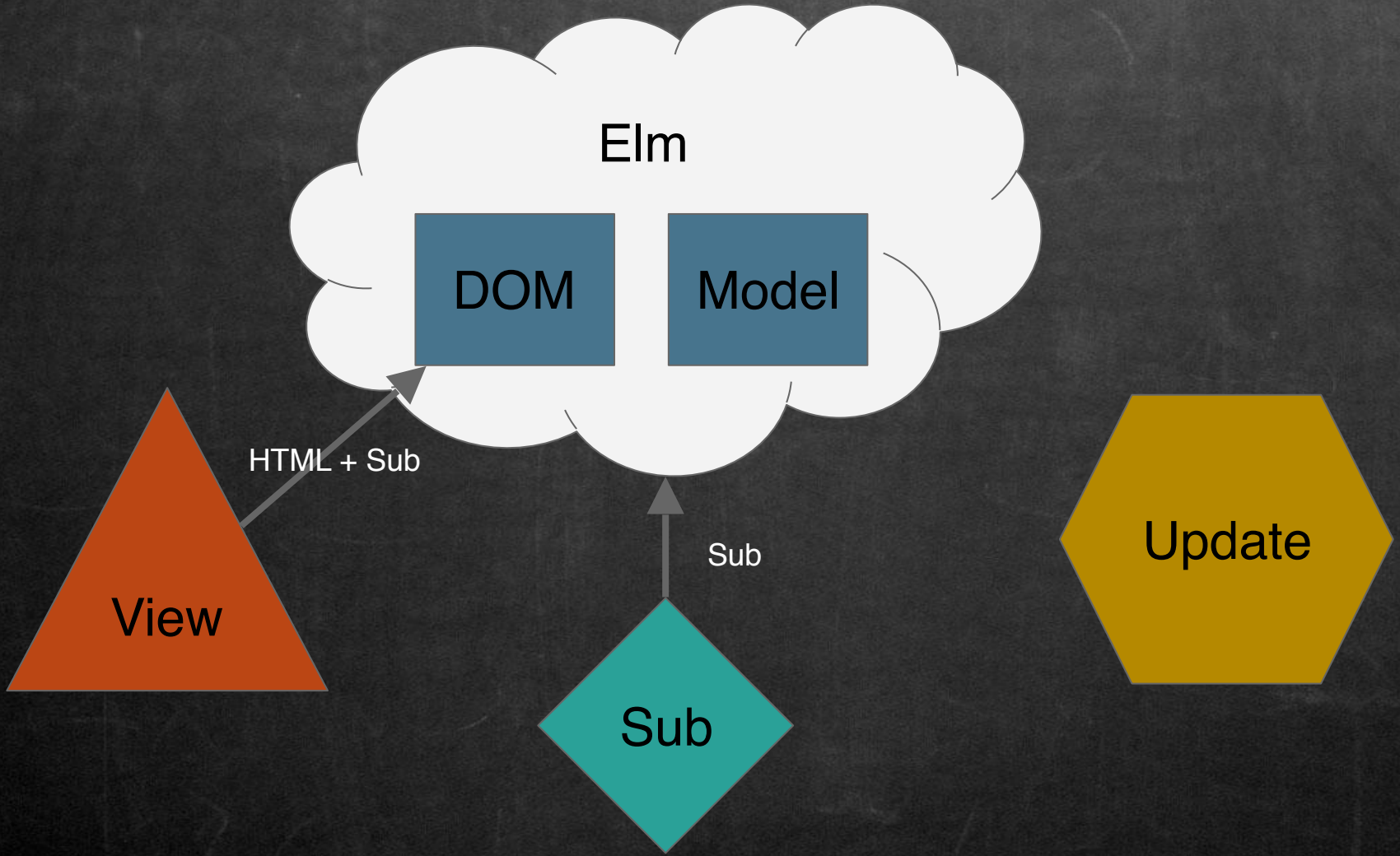

Elm
DOM
Model
Model + Cmd
Update
View
Sub


Elm
DOM
Model
Execute
View
Sub
Update


Elm
DOM
Model
Msg
Model
Update
View
Sub


Elm
DOM
Model
Model
Update
View
Sub


Elm
DOM
Model
Model
Model
View
Sub
Update


Elm
DOM
Model
HTML + Sub
Sub
View
Sub
Update

```elm
type Msg
  = RestPost User
  | RestPostResult (Result Http.Error UserId)


update : Msg -> Model -> ( Model, Cmd Msg )
update msg model =
  case msg of
    RestPost user ->
      ( model, RestClient.postUser RestPostResult user )

    RestPostResult (Ok userId) ->
      (model, Cmd.none)

    RestPostResult (Err error) ->
      (model, Cmd.none)
```

- side effects are delegated to Elm runtime
- Elm program is free of side effect → purely functional

Why Elm?

Why Elm ?

Why functional ?

- no mutable state
- no side effects

⇒ testable

⇒ predictable

Why Elm ?

Why not JavaScript ?

- ML class language
- concise

⇒ easy to understand

- strongly typed
- static
- side effect free

⇒ if it compiles it won’t crash

Why Elm ?

Why not PureScript ?

- only basic language features
- opinionated framework
- made for one purpose

⇒ easy to get into (Elm to PureScript is like C to C++)

⇒ similar architecture and design

Is Elm ready for production?

- Performance of JavaScript
- DOM Performance (Graph)
  http://elm-lang.org/blog/blazing-fast-html-round-two
- Stable ?
- Language not final
- Very helpful with migrations
- Mature tooling ? (versioning)
- Good documentation ?
- Active community