

Programmation Réactive

Principes fondamentaux et application au Web

Plan

- ▶ Quoi et pourquoi la réactivé
- ▶ Quelles limites de MVC
- ▶ Architecture Flux
- ▶ Réactivité, Vue et Vuex
- ▶ Traitement réactif de flux

Qu'est ce que la programmation réactive ?

Une approche visant à mieux gérer les flux

Deux types de flux

- ▶ Des événements discrets : frappe clavier, action utilisateur
- ▶ Des événements continus ou *comportements* : position souris

Dépasser les callbacks ou le patron Observer.

Alternative aux machines à état

Où avez vous vu ça ?

A screenshot of the Microsoft Excel ribbon interface. The 'Home' tab is selected. In the 'Font' section, the font is set to 'Calibri (Body)' at size '12'. The 'Font Color' dropdown shows yellow and red. Below the ribbon, the formula bar displays '=A1*2'. The worksheet shows a grid with columns A through F and rows 1 through 3. Cell A1 contains the value '1'. Cell B1 contains the formula '=A1*2'. The formula is currently selected, indicated by a blue border around the cell.

<http://www.hanselsolutions.com/blog/surf-talk/shiny-surf.html#/9>

Où avez vous vu ça ?

The screenshot shows a Microsoft Excel interface. The ribbon menu is visible at the top, with 'Home' selected. The formula bar below shows 'B2' is selected. A table is present with data in cells B1, B2, and C2. The cell B2 contains the value '10'. The font color of the text in cells B1, B2, and C2 is blue. The background of the slide features a light green gradient.

	A	B	C	D	E	F
1		5	10			
2						
3						

<http://www.hanselsolutions.com/blog/surf-talk/shiny-surf.html#/9>

Où avez vous vu ça ?

The screenshot shows a Microsoft Excel interface. The ribbon at the top has tabs for Home, Layout, Tables, Charts, SmartArt, and Formulas. The Home tab is selected. The formula bar below shows the cell reference **A2**. The main area contains a table with three rows and six columns labeled A through F. Row 1 contains values 6 and 12 in columns A and B respectively. Row 2 is selected, indicated by a blue border around column A. Row 3 contains empty cells. The ribbon's Font section shows Calibri (Body) in 12pt size, bold, italic, and underline options. The Edit section includes Paste, Fill, and Clear buttons.

	A	B	C	D	E	F
1	6	12				
2						
3						

<http://www.hanselsolutions.com/blog/surf-talk/shiny-surf.html#/9>

Pourquoi la programmation réactive ?

- ▶ Gestion d'évènements et de l'asynchrone
- ▶ Faible latence (contraintes sur les temps de réponse)
- ▶ Flux de données importants (et rapides).
- ▶ Tolérance aux fautes

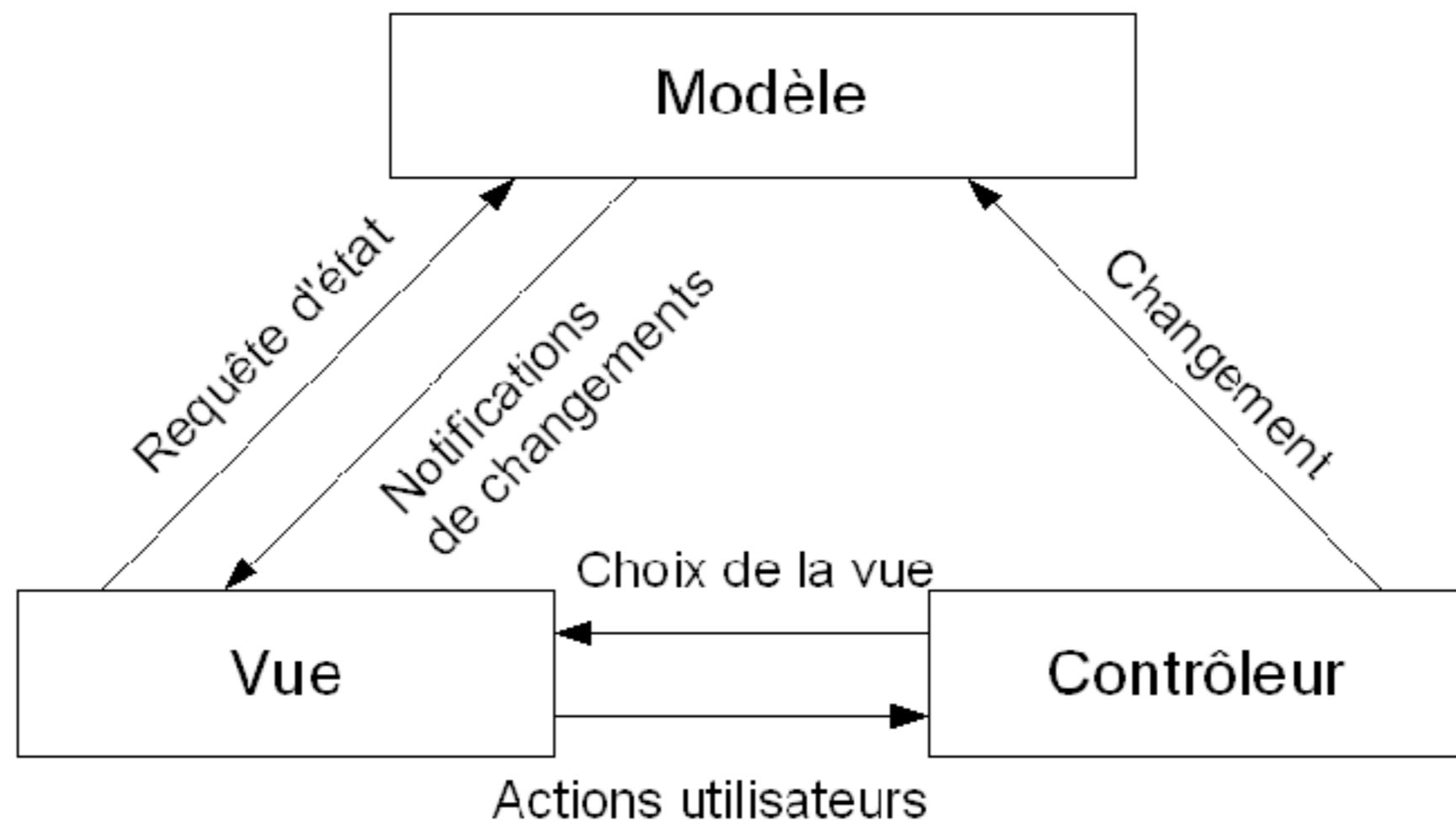
Exemples

À vous

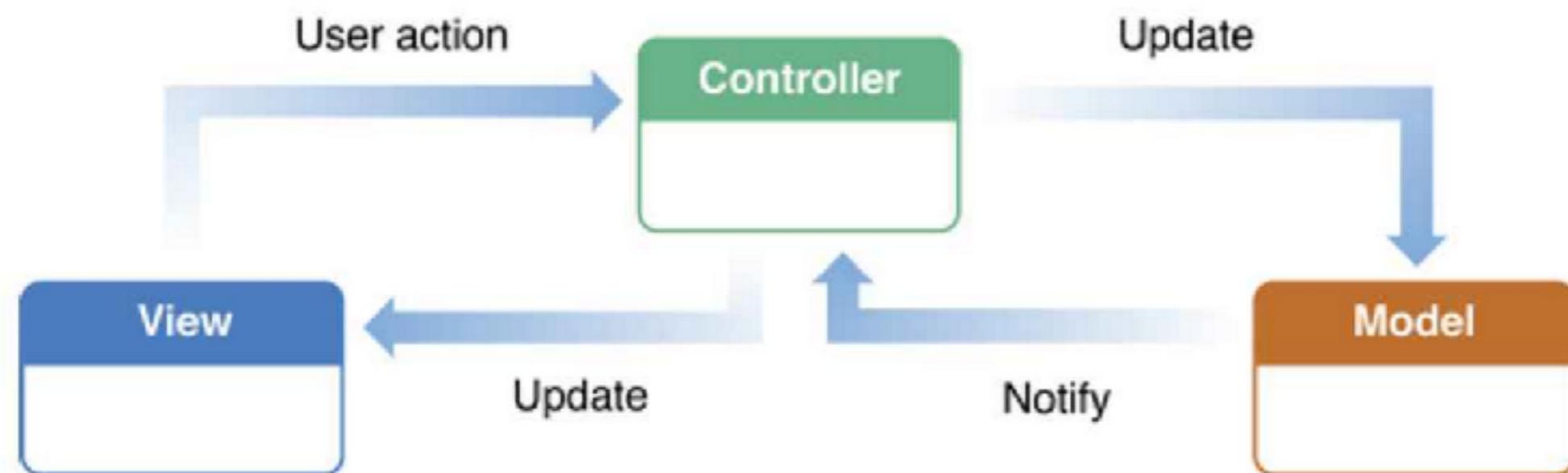
Plan

- ▶ Quoi et pourquoi la réactivé
- ▶ **Quelles limites de MVC**
- ▶ Architecture Flux
- ▶ Réactivité, Vue et Vuex
- ▶ Traitement réactif de flux

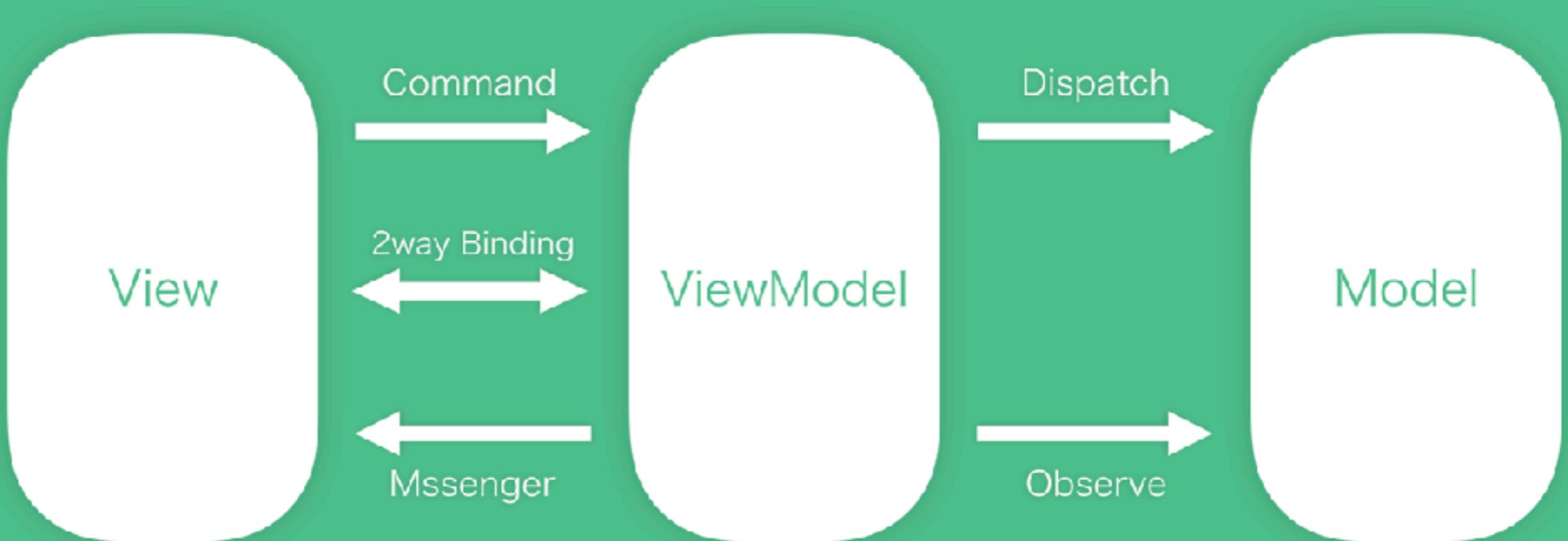
MVC - plutôt natif



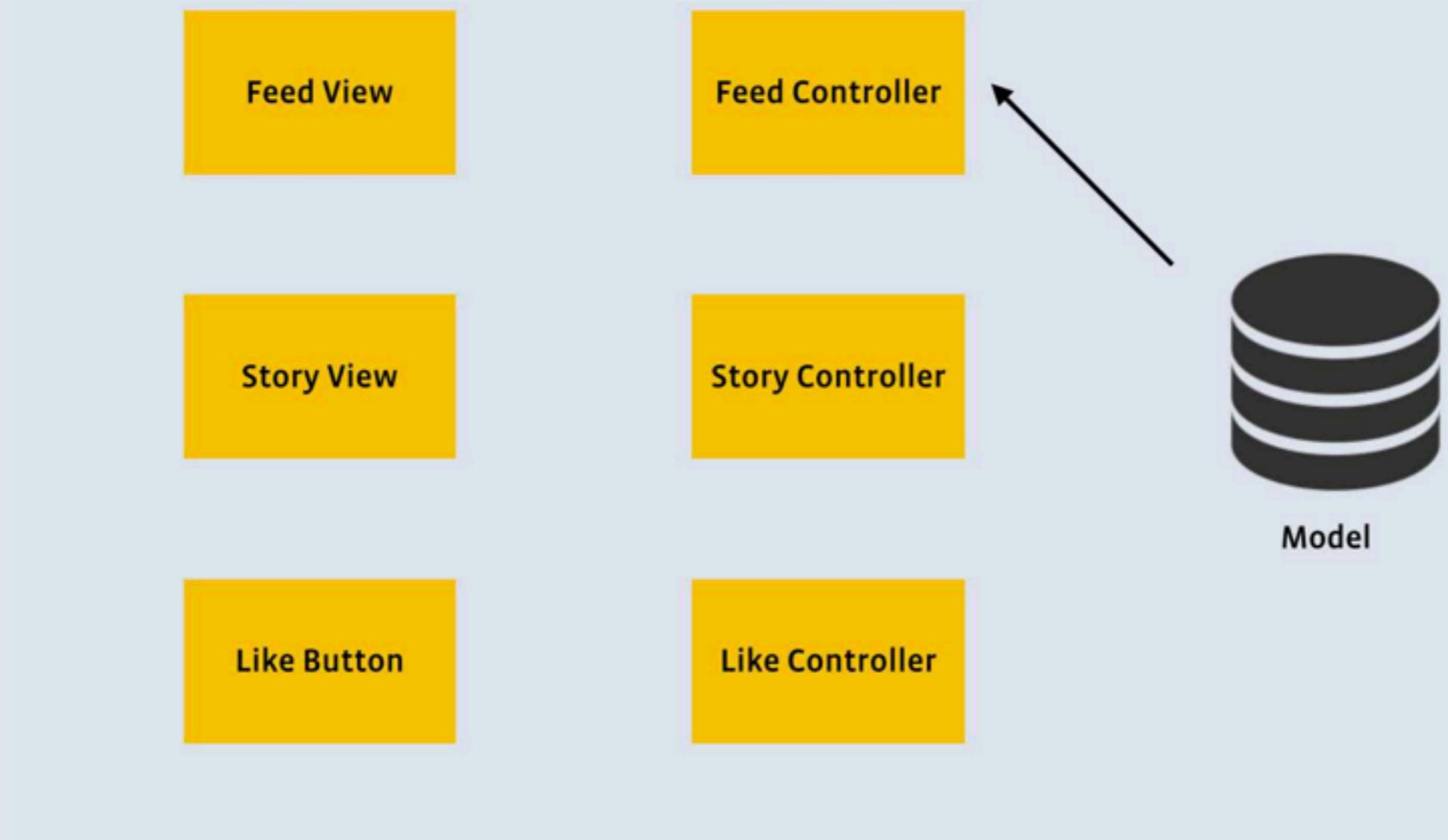
MVC - plutôt Web



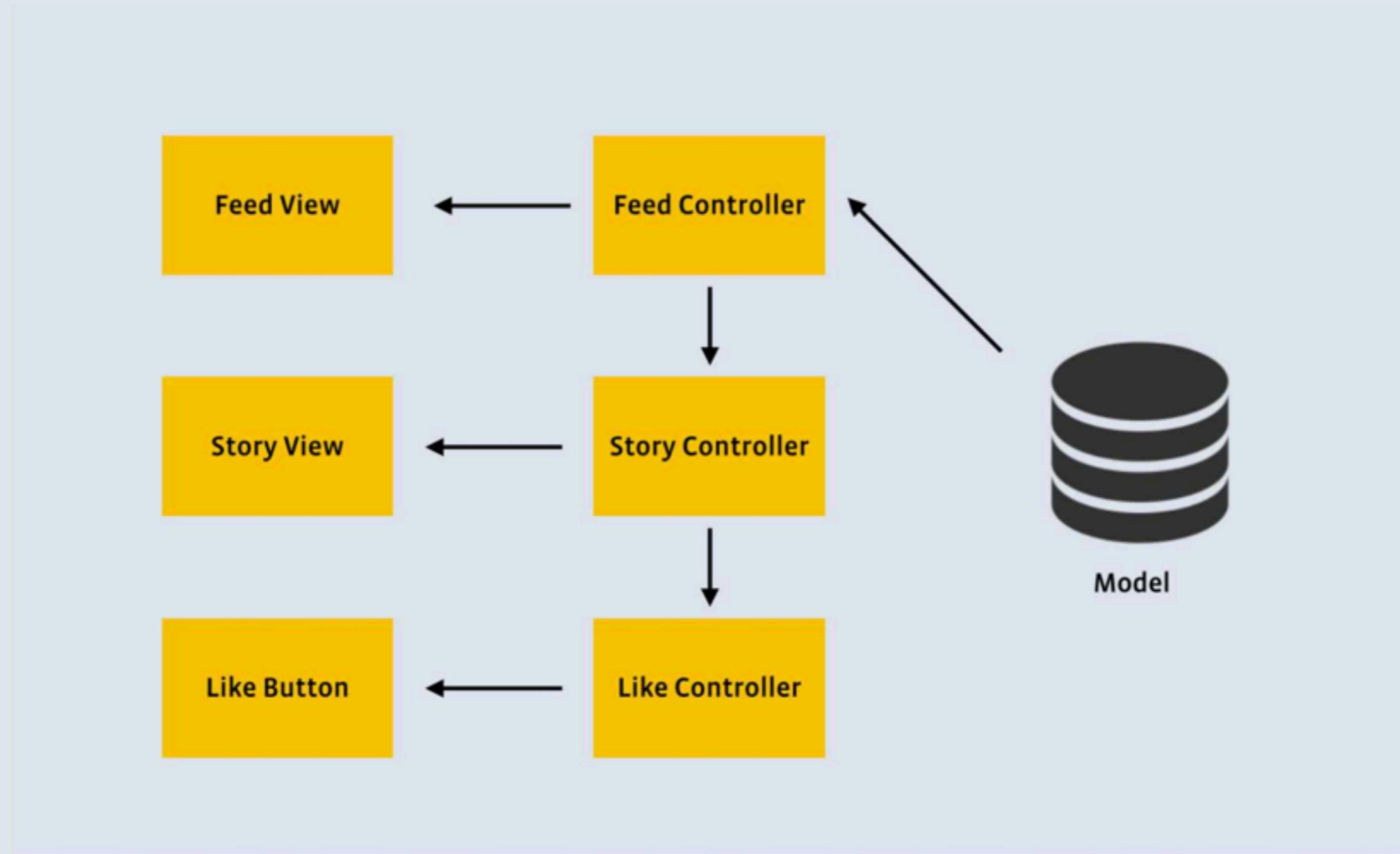
MVC ou MVVM (ou MVP)



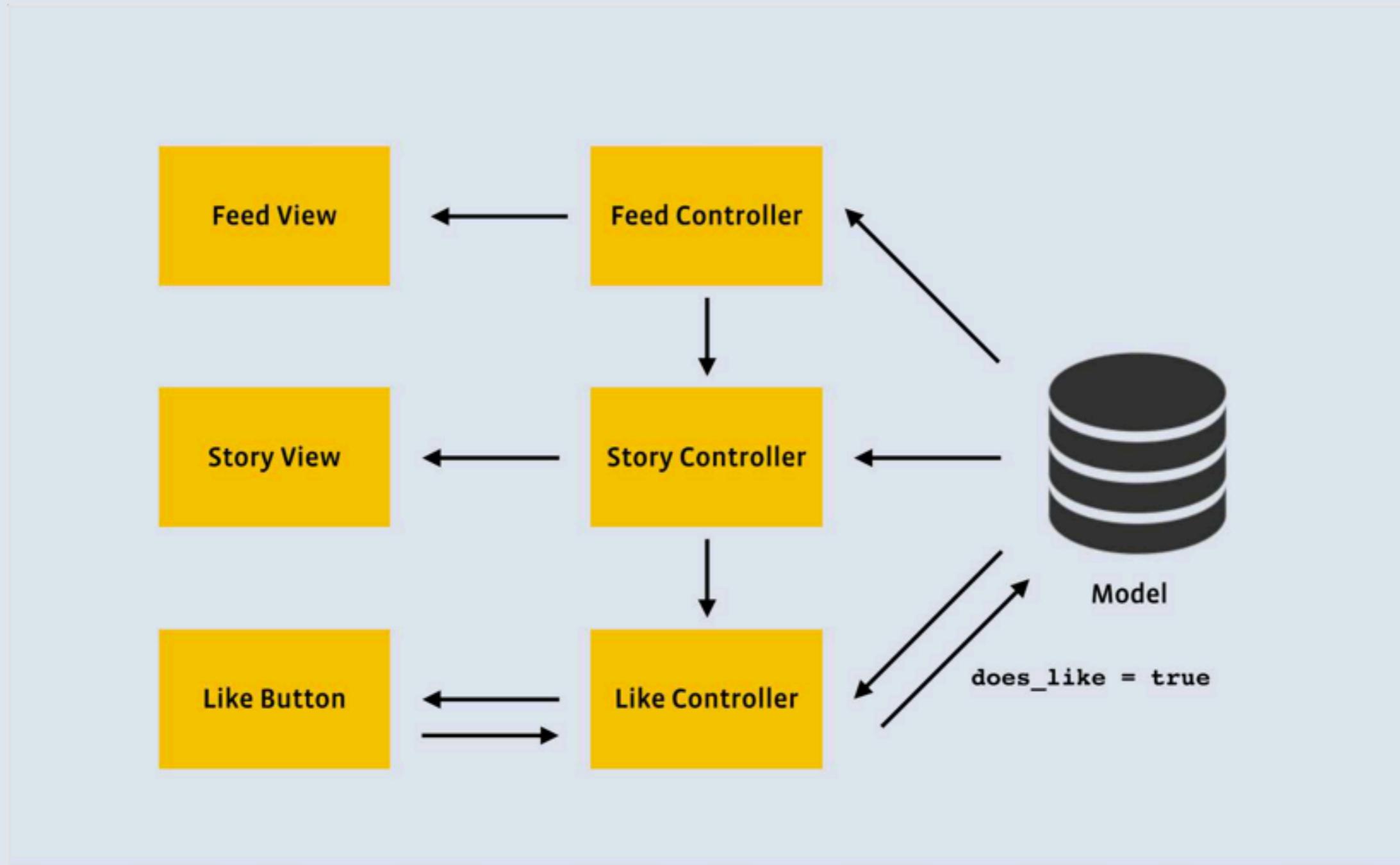
En pratique



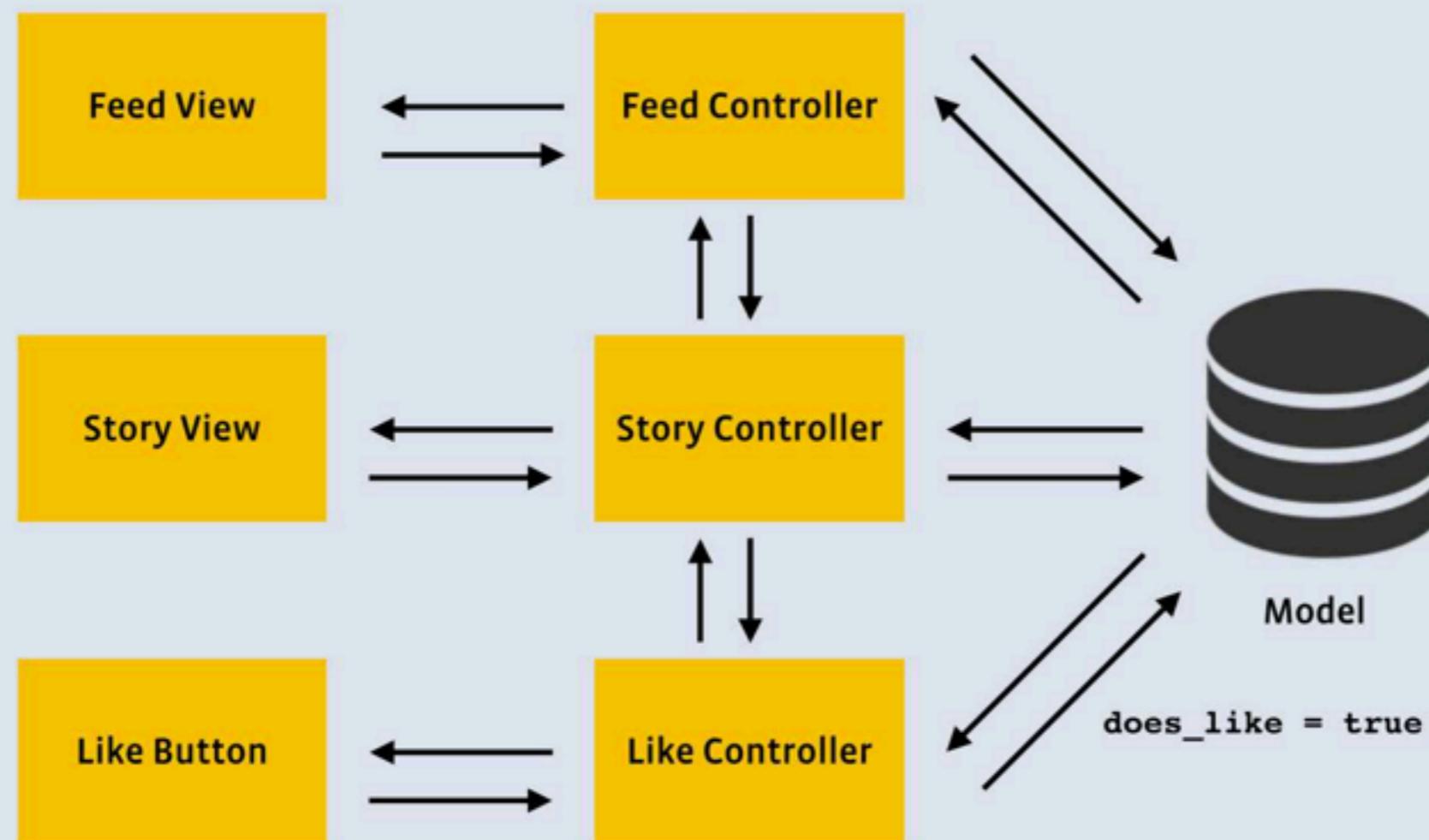
En pratique



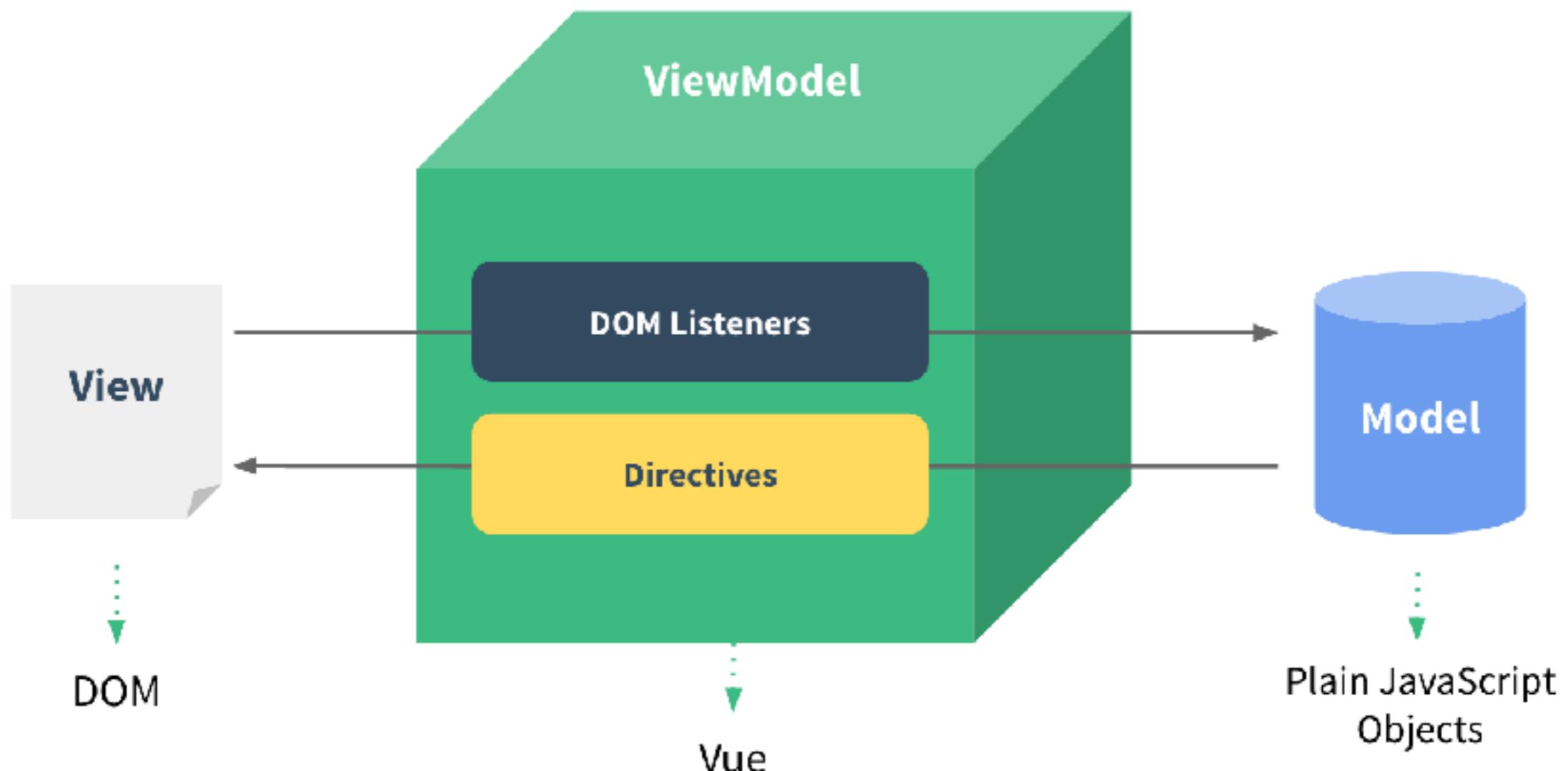
En pratique



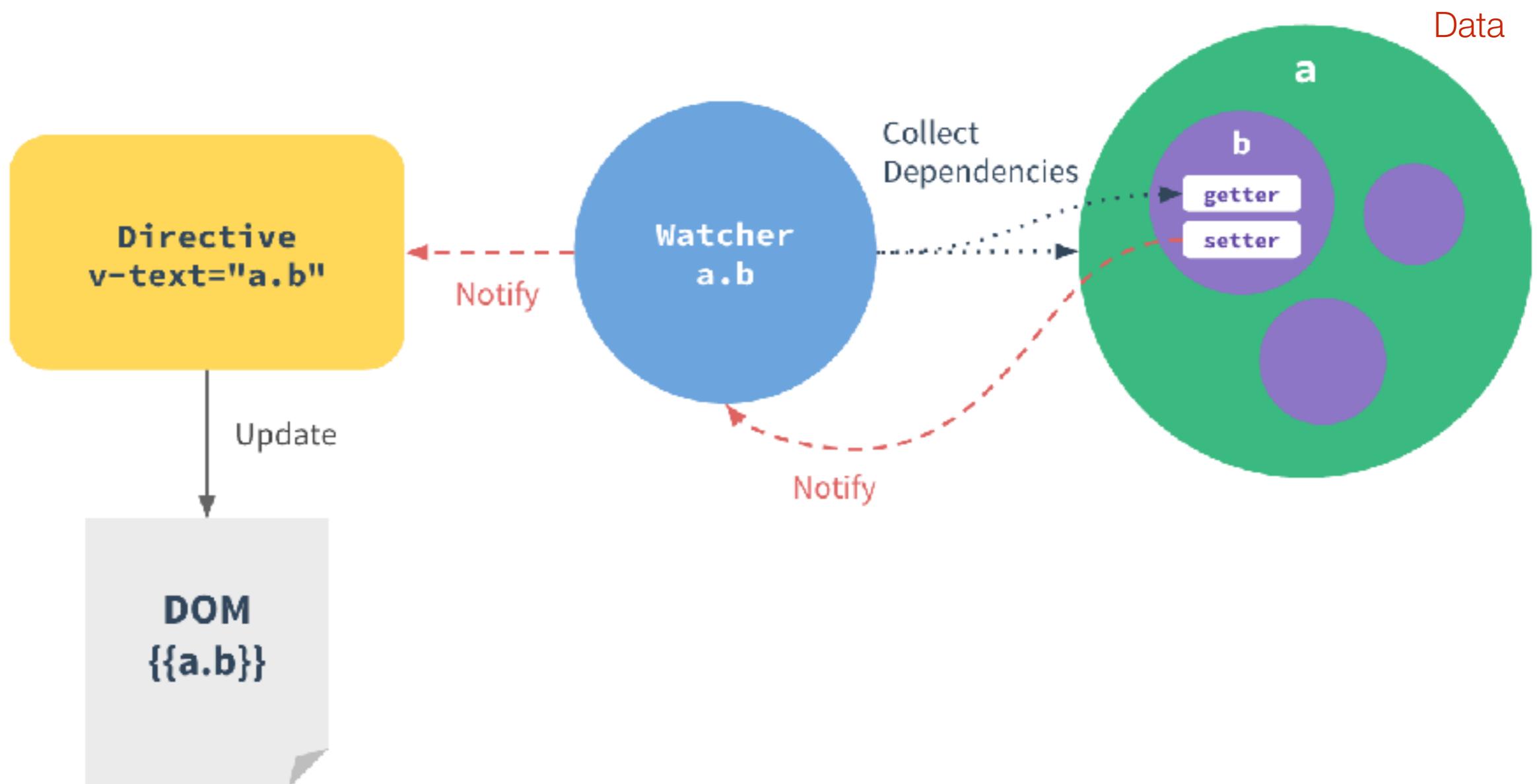
En pratique

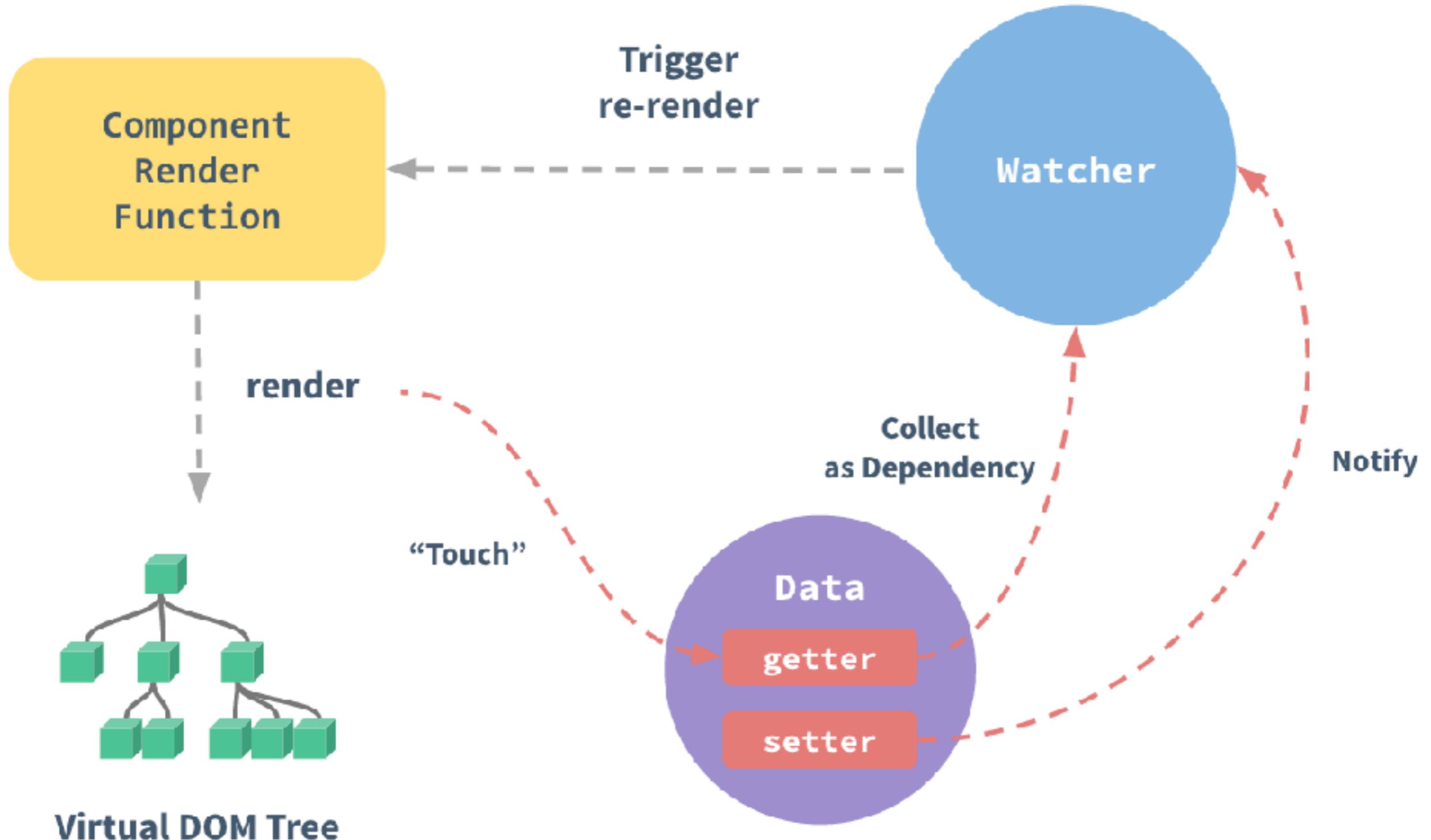


MVVM avec Vue



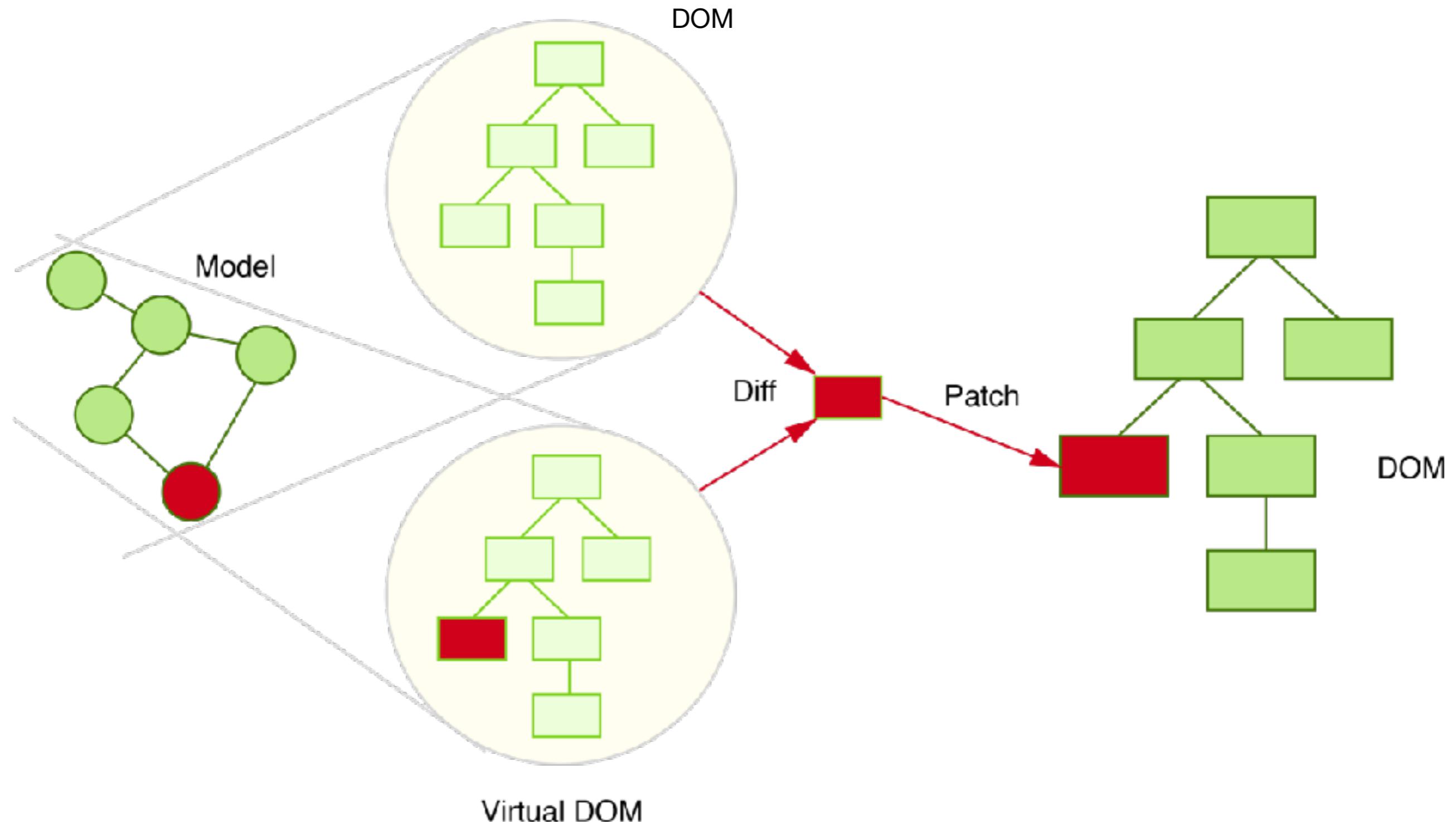
En pratique avec Vue





Un DOM Virtuel

<https://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html>



Les bibliothèques Javascript

Guide API Examples Blog Community ▾



Vue.js

Reactive Components for Modern Web Interfaces

Install v1.0.24

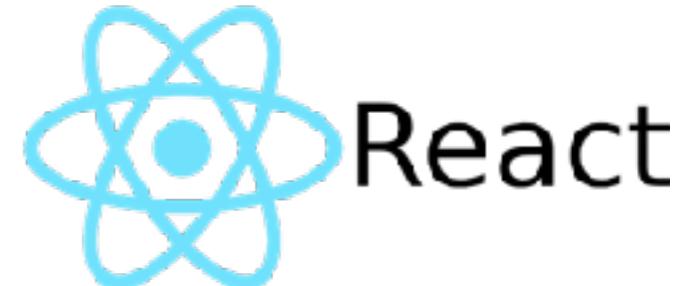
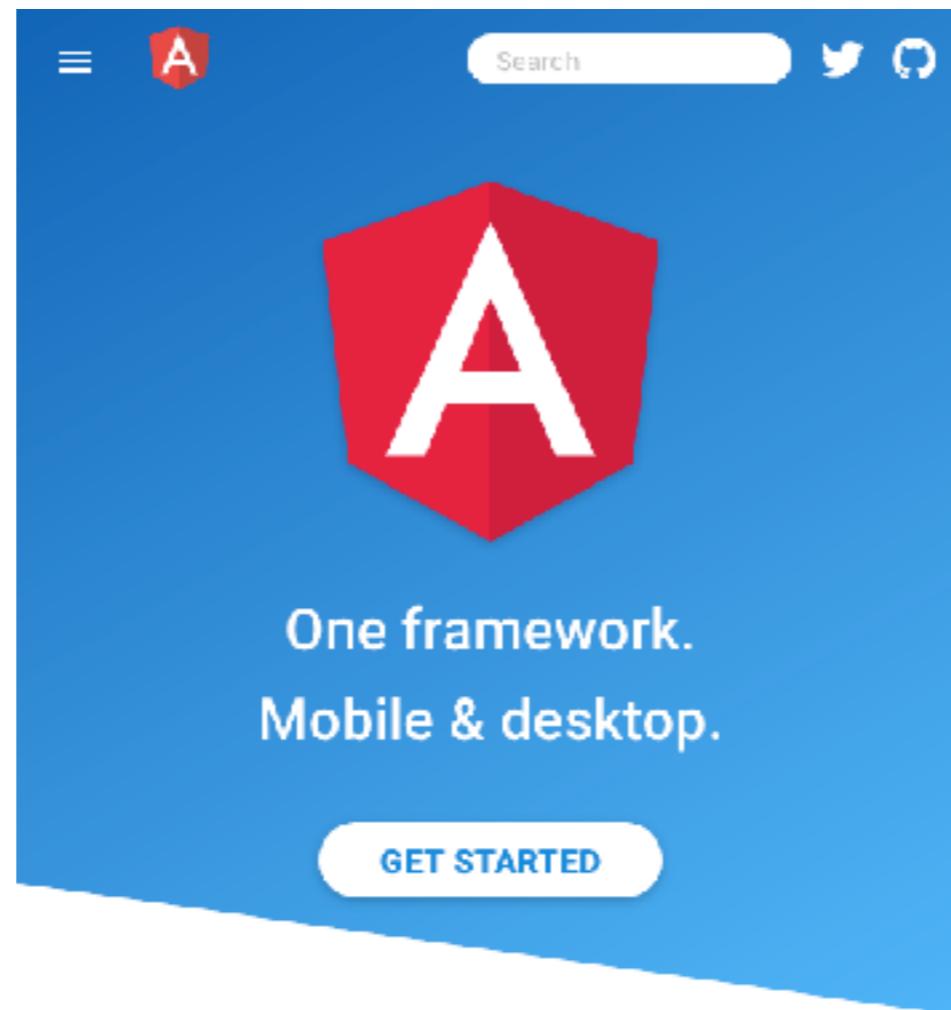
[Follow @vuejs](#)

[Star](#)

18,631

[Support Vue.js](#)

中文 | 日本語 | Italiano



En pratique

The screenshot shows a Facebook news feed page with several UI elements highlighted by hand-drawn red circles and arrows:

- Left sidebar:** Circles point to the "News Feed" and "Messages" links under the "FAVORITES" section.
- Top navigation:** Circles point to the search bar and the "Create Ad" button.
- Post 1 (House):** A large yellow circle highlights the main image of a house with people on the porch. Red circles highlight the "Like", "Comment", and "Share" buttons below it, and another points to a comment from Jim Anderson.
- Post 2 (Bracelet):** A red circle highlights the "Sponsored" label above the ad. Another red circle highlights the "60% OFF" offer and the website "www.ashleyandjewel.com".
- Post 3 (Physicist Tee):** A large red circle highlights the "Are you a physicist?" post. Another red circle highlights the "553 people like this" count.
- Comments:** Red circles highlight comments from Jim Anderson, Mario Aquino, and Dean Wetter.
- Right sidebar:** A red circle highlights the "Kristen Bruch likes Power of Positivity's photo" status update.
- Bottom right:** A red circle highlights the "Lance Cameron Kidwell" status update.

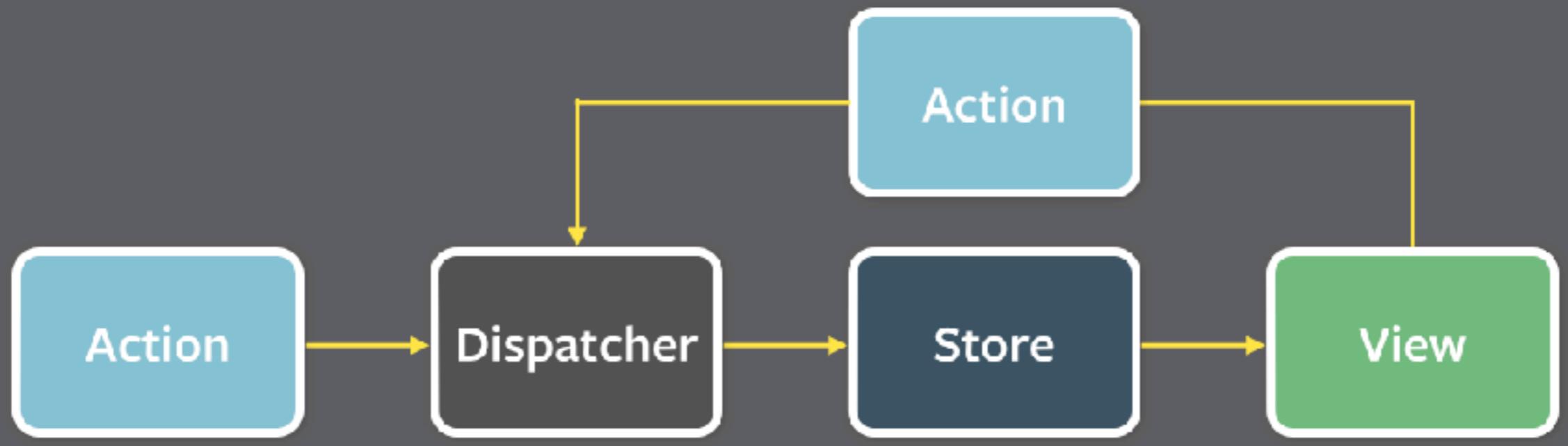
Problèmes

1. La vue gère son état “en interne” -> elle est mutable mais influe sur le modèle quand même.
2. Un changement implique une cascade d’inter-dépendances
 - ▶ Lenteurs (re-dessins multiples) sur le thread principal
 - ▶ Race conditions, à cause d’opérations atomique
 - ▶ Complexité et risque d’inter-blocage

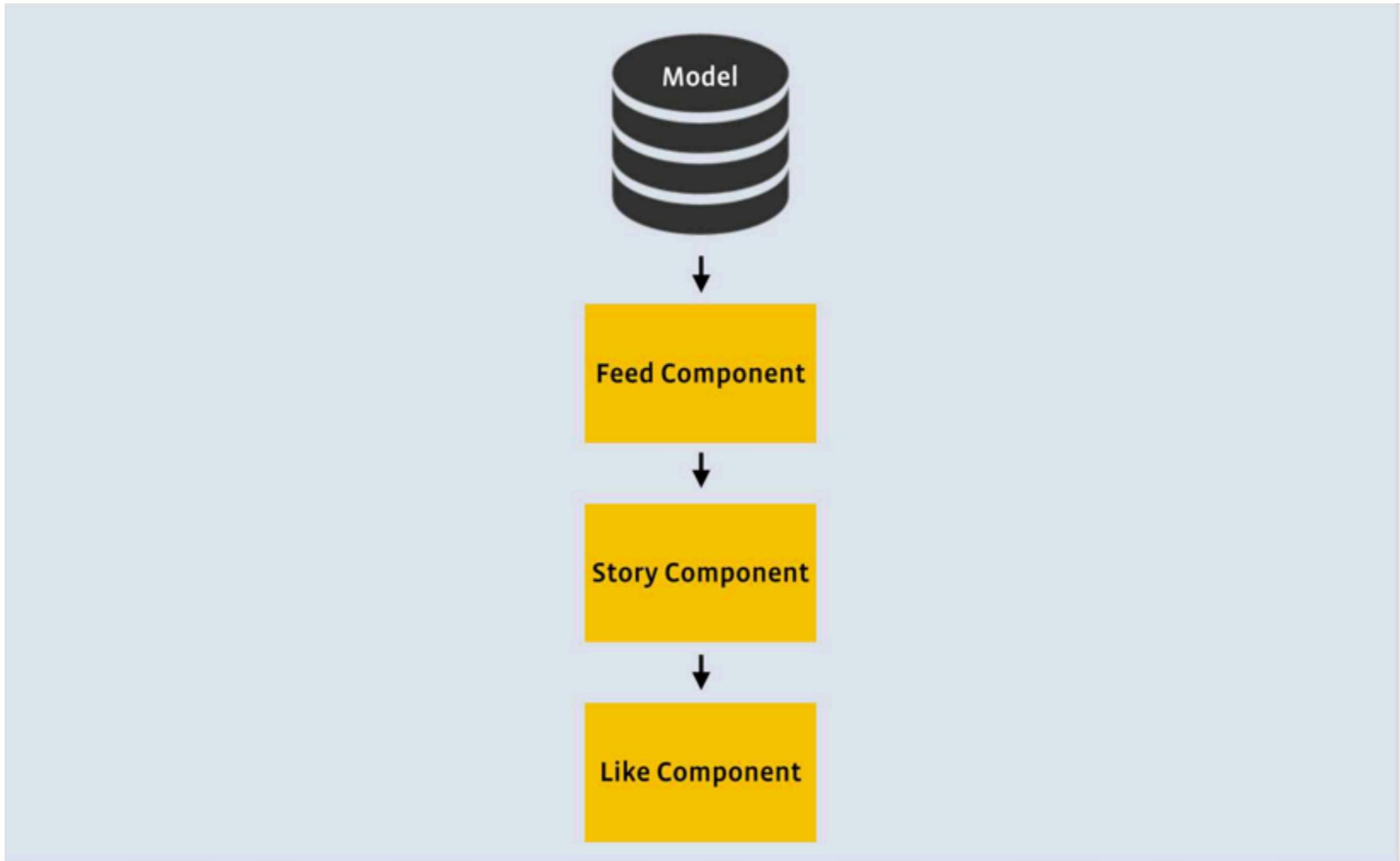
Plan

- ▶ Quoi et pourquoi la réactivé
- ▶ Quelles limites de MVC
- ▶ **Architecture Flux**
- ▶ Réactivité, Vue et Vuex
- ▶ Traitement réactif de flux

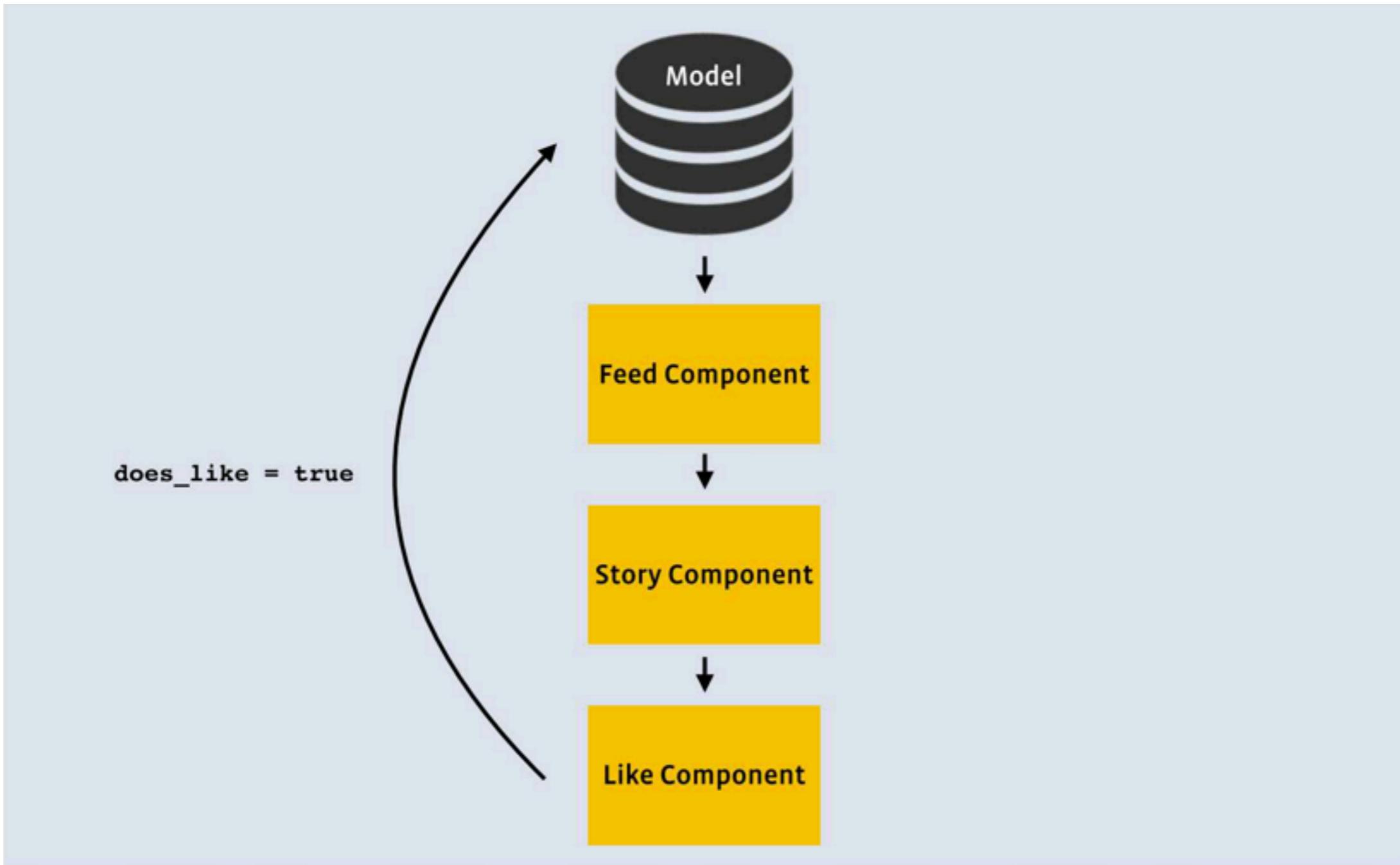
L'architecture Flux



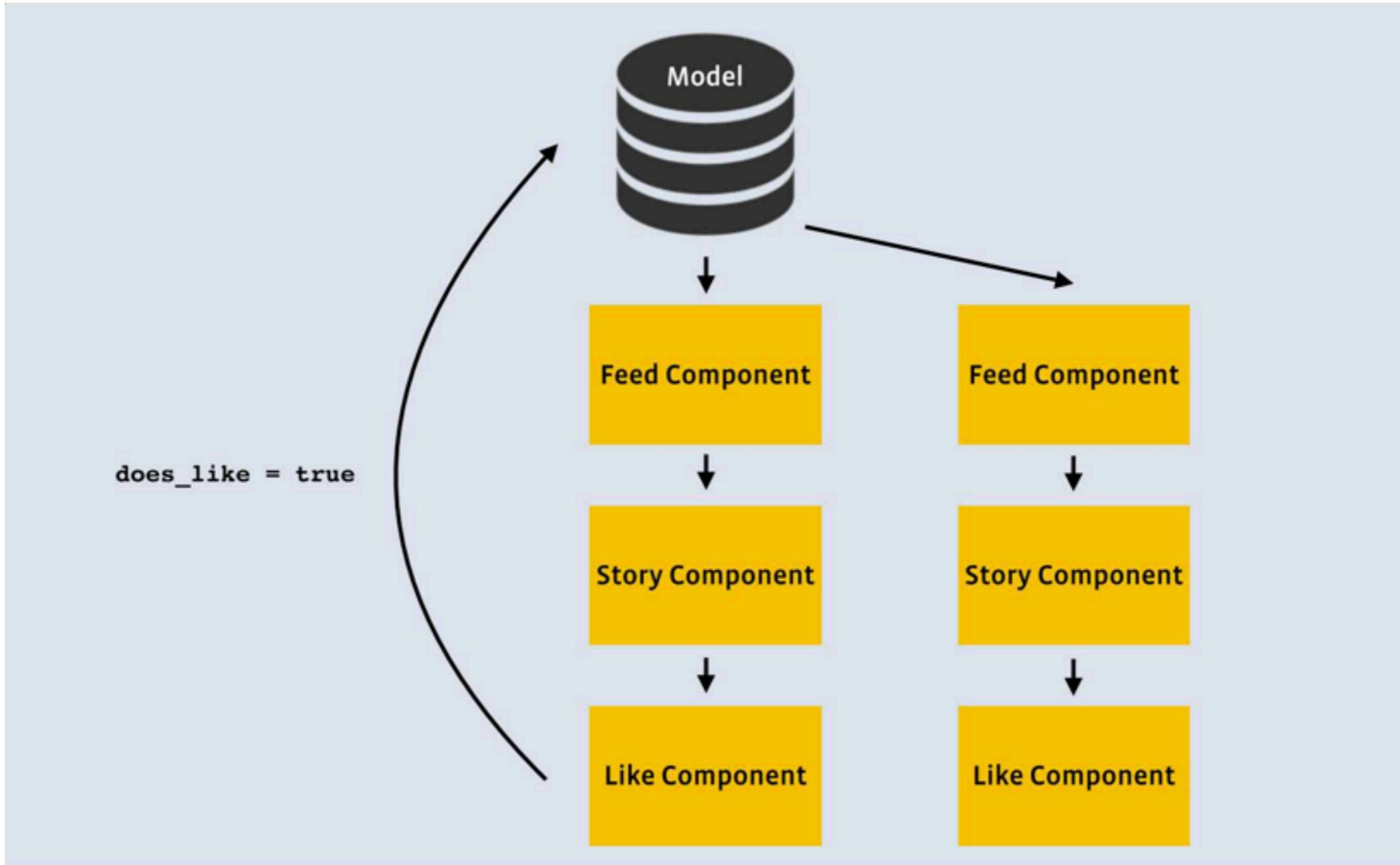
Principe généraux de Flux



On modifie le modèle directement



Nouvel arbre de rendu



Flux de données avec Vue

Deux approches au data-binding (lien entre données et vue):

- ▶ Two way data-binding

- ▶ One way data binding

Le flux de données est uni-directionnel, les enfants ne modifient pas les données qui sont contrôlées par leur parents.

Mais les enfants peuvent demander au parent de se mettre à jour (et tout rafraîchir depuis le modèle).

Un concept important : l'immuabilité

Objet immuable (Immutable object)

- ▶ Objet dont l'état ne peut pas être modifié après sa création
- ▶ Opposé d'objet variable

Facilite la prog. purement fonctionnelle (pratique pour plein de choses, évite les effets de bords, facilite le undo)

Une seule source de “vérité”

Facilite le caching

Mais ce n'est pas forcément assez : <https://codewords.recurse.com/issues/six/immutability-is-not-enough>

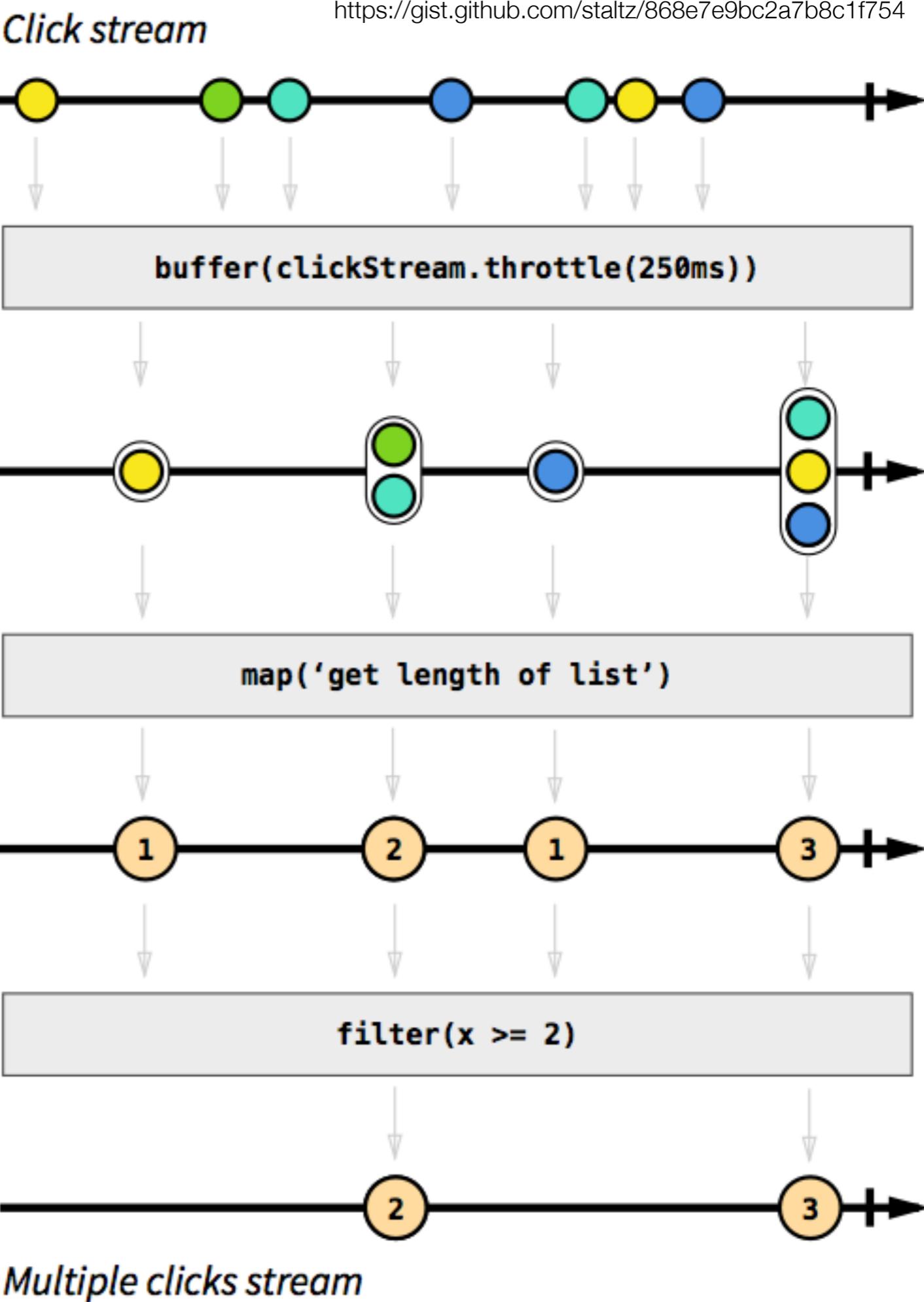
Modèle immuable

- ▶ Les objets restent immuables
- ▶ On ne modifie pas le modèle mais on effectue des opérations dessus
- ▶ Quand un changement arrive, un nouvel arbre est créé depuis le haut
- ▶ Les “stores” de haut niveau reçoivent des mise à jour (updates) de façon asynchrone

Deux façons de gérer les données

- ▶ Données qui changent (mutable) :
on utilise un état (data)
- ▶ Données qui ne changent pas (immutable) :
on utilise des propriétés (props)
- ▶ On essaie de minimiser les données qui changent
quitte à refaire des calculs

Un exemple de transformation



Plan

- ▶ Quoi et pourquoi la réactivité
- ▶ Quelles limites de MVC
- ▶ Architecture Flux
- ▶ **Réactivité, Vue et Vuex**
- ▶ Traitement réactif de flux

Vuex : gestion d'états

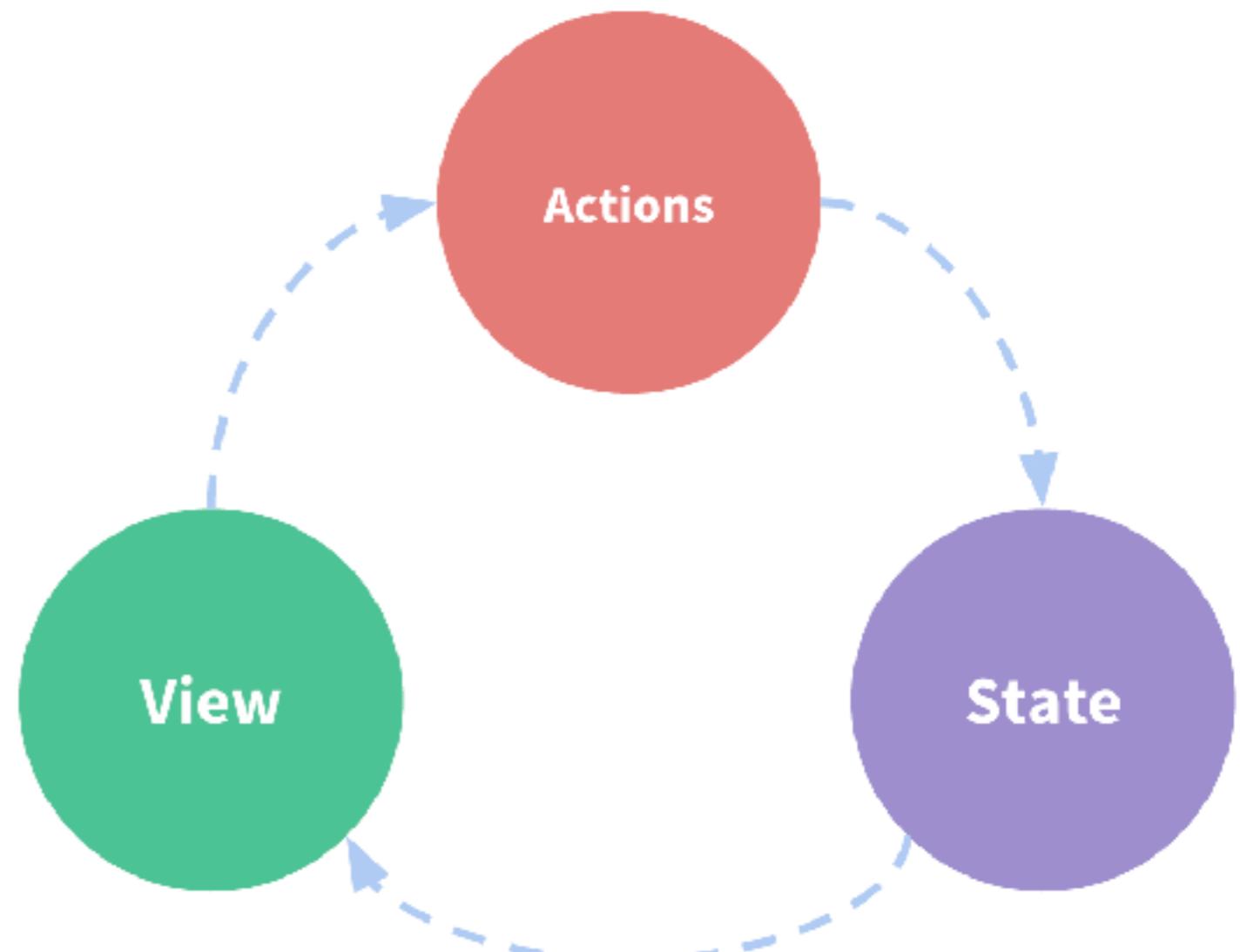
<https://vuex.vuejs.org/en/intro.html>

One-way data flow

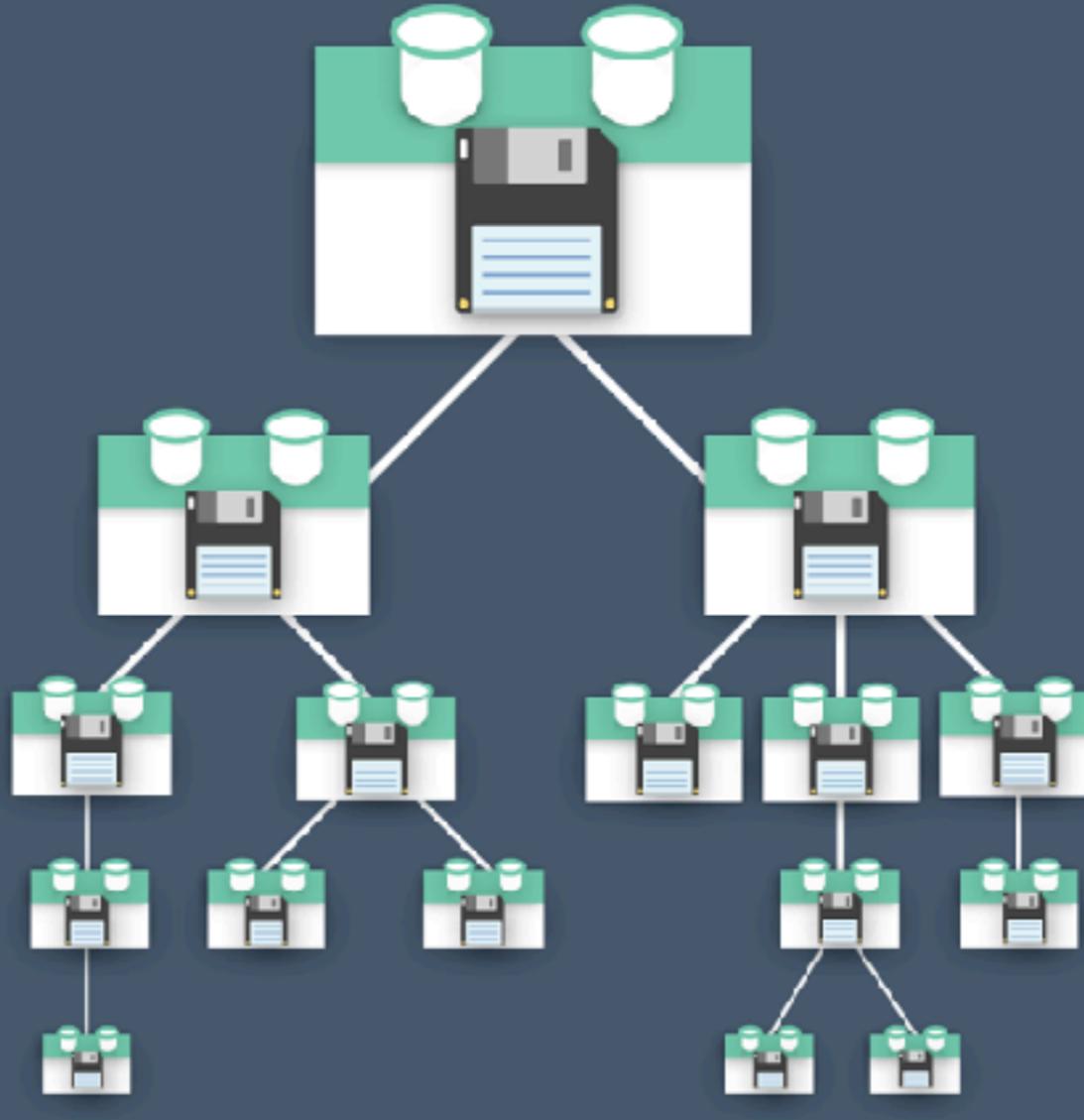
State (état), source de “vérité” pour l’application

View (vue), représentation de l’état (mapping déclaratif)

Actions, changements de l’état en réaction à des entrées de l’utilisateur au niveau de la vue (ou d’autres inputs)

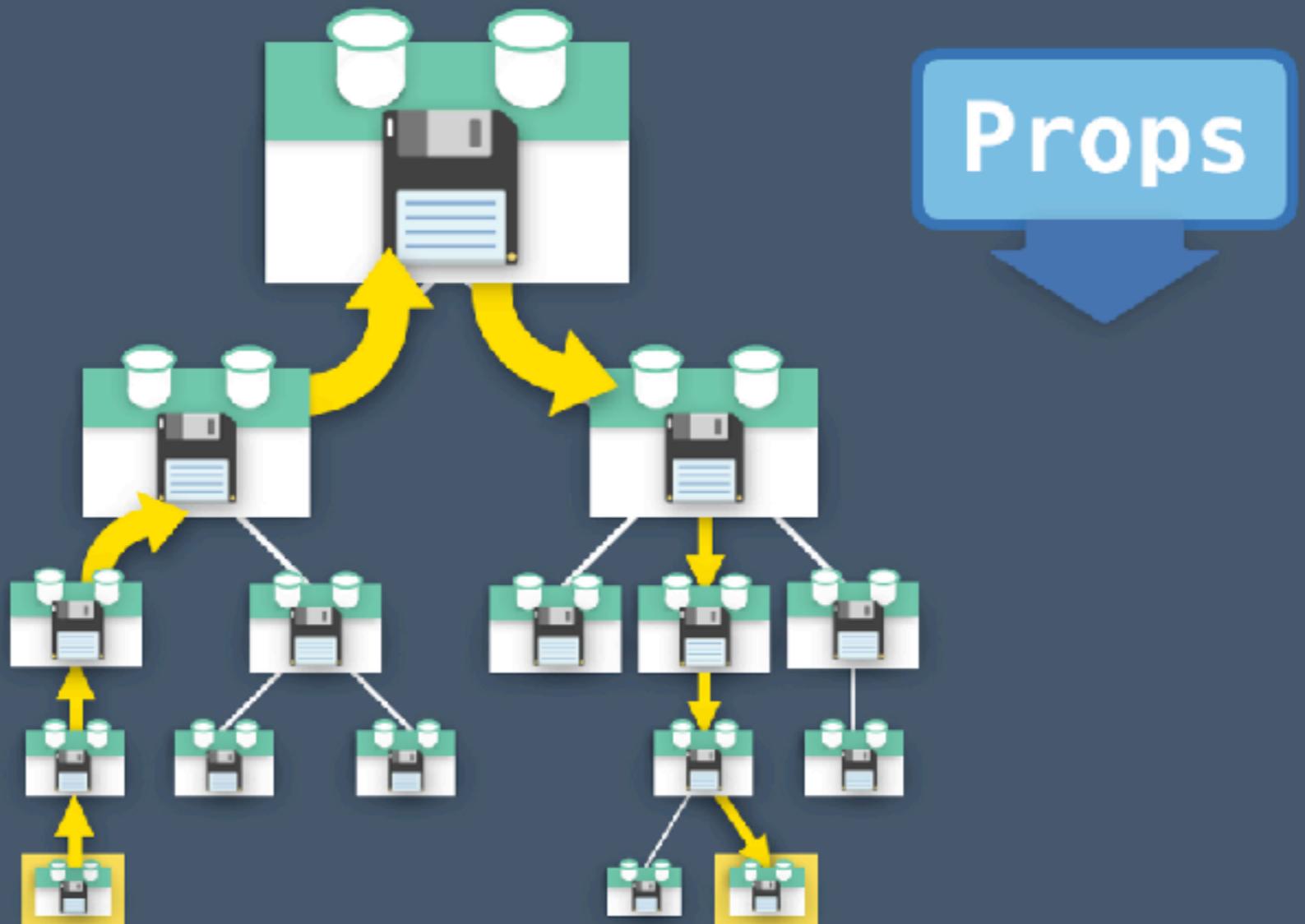


Vue et v-model



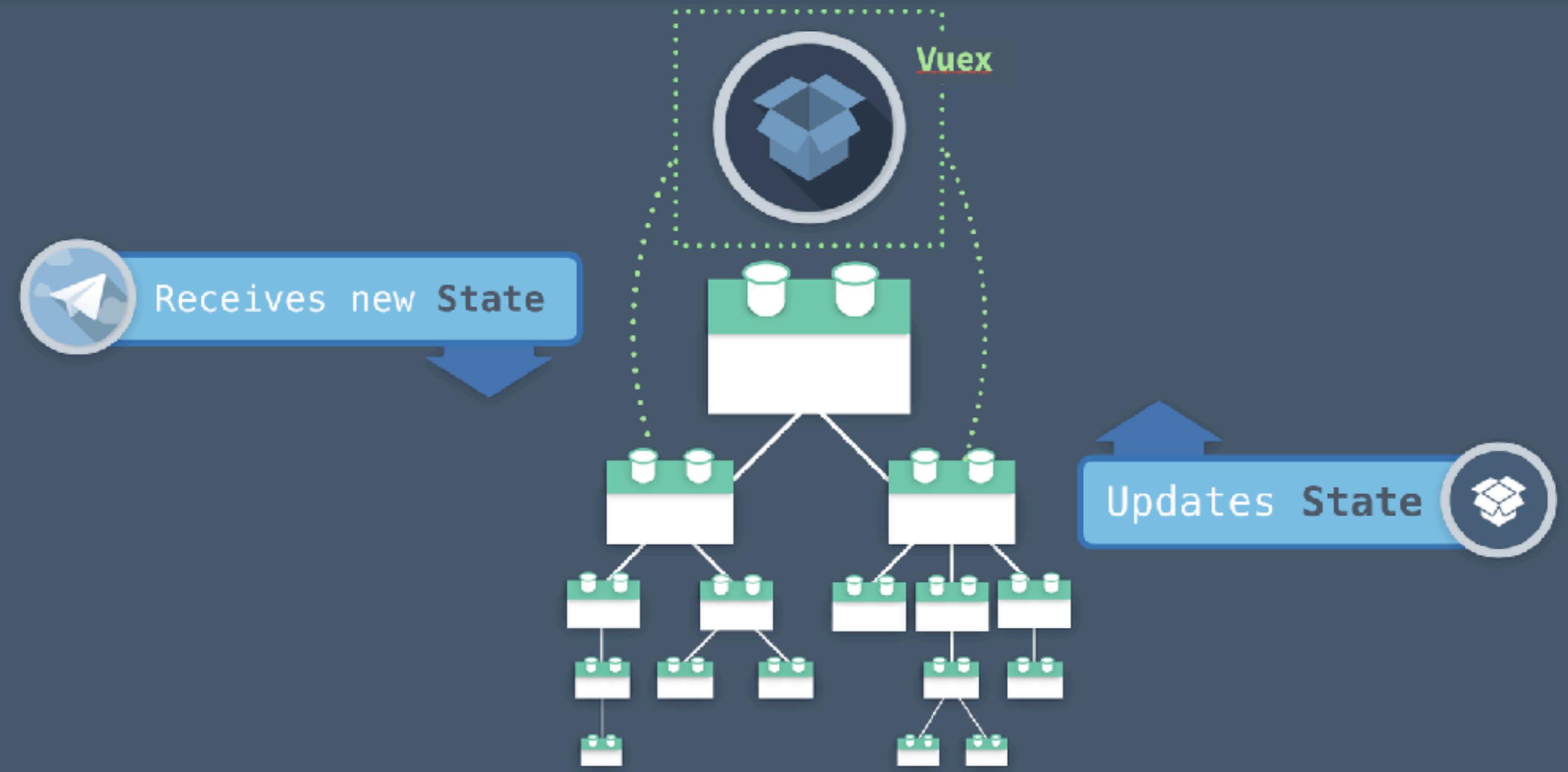
- ▶ Un état géré dans chaque composant

Problèmes



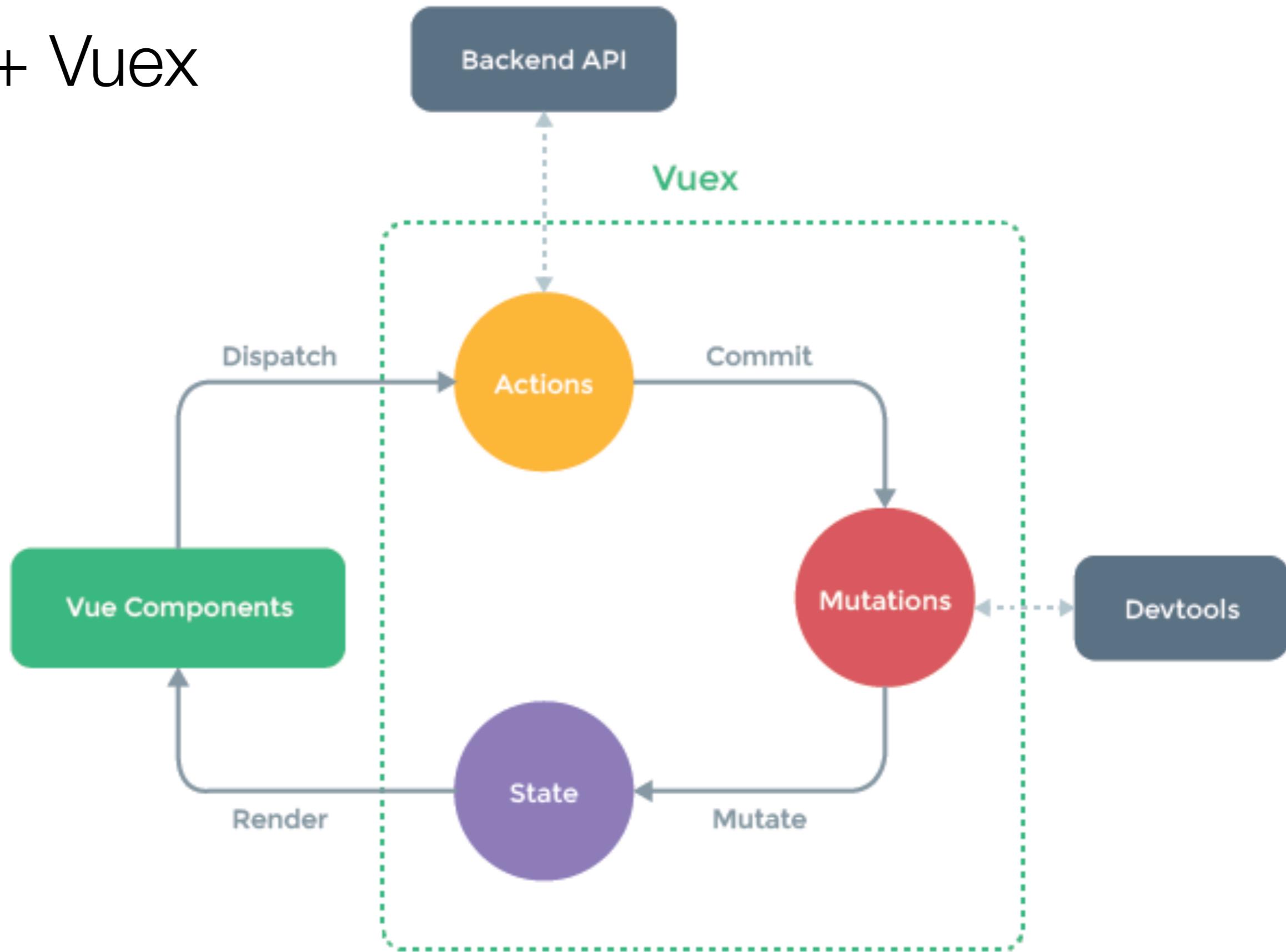
- ▶ Plusieurs vues dépendent du même bout d'état
- ▶ Plusieurs actions de différentes vues vont changer (mutate) un même bout d'état

Solution: Vuex



- ▶ l'état devient un gros singleton partagé par tous les composants

Vue + Vuex



Cas concret



```
▼ <Root>
  ▼ <App>
    <Todo key=0>
    <Todo key=1>

  ▼ state
    ▼ todos: Array[2]
      ▼ 0: Object
        done: true
        text: "créer une action"
      ▼ 1: Object
        done: false
        text: "créer une mutation"
```

Structure de projet

```
index.html
main.js
api
  ... # abstractions for making API requests
components
  App.vue
  ...
store
  index.js      # where we assemble modules and export the store
  actions.js     # root actions
  mutations.js    # root mutations
  modules
    cart.js       # cart module
    products.js   # products module
```

Créer son store (vue2+vuex3)

<https://vuex.vuejs.org/fr/guide/>

```
// app.js
import Vue from 'vue'
import store from './store'
import App from './components/
App.vue'

new Vue({
  store, // inject store to all
children
  el: '#app',
  render: h => h(App)
})
```

```
// store.js
import Vue from 'vue'
import Vuex from 'vuex'
import mutations from './mutations'
import actions from './actions'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    todos: [...] // état initial
  },
  actions,
  mutations
})
```

Créer son store (vue3+vuex4)

<https://vuex.vuejs.org/guide/>

```
import { createApp } from 'vue'
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state () {
    return {
      todos: [...]
    }
  },
  actions,
  mutations,
})

const app = createApp({ /* your root component */ })
app.use(store)
```

Les actions

```
export default {
  addTodo ({ commit }, text) {
    commit('addTodo', {
      text,
      done: false
    })
  },
  removeTodo ({ commit }, todo) {
    commit('removeTodo', todo)
  },
  toggleTodo ({ commit }, todo) {
    commit('editTodo',
      {todo, done: !todo.done})
  },
  editTodo ({ commit }, { todo, value }) {
    commit('editTodo',
      {todo, text: value})
  },
}
```

```
toggleAll ({ state, commit }, done) {
  state.todos.forEach((todo) => {
    commit('editTodo', { todo, done })
  })
},
clearCompleted ({ state, commit }) {
  state.todos.filter(todo => todo.done)
    .forEach(todo => {
      commit('removeTodo', todo)
    })
}
```

Les mutations

```
export const mutations = {
  addTodo(state, todo) {
    state.todos.push(todo);
  },
  removeTodo(state, todo) {
    state.todos.splice(state.todos.indexOf(todo), 1);
  },
  editTodo(state, { todo, text = todo.text, done = todo.done }) {
    const index = state.todos.indexOf(todo);

    state.todos.splice(index, 1, {
      ...todo,
      text,
      done,
    });
  },
};
```

Lien entre le composant TodoItem et le store

```
<template>
  <li class="todo"
      :class="{ completed: todo.done, editing:
editing }">
    <div class="view">
      <input class="toggle"
        type="checkbox"
        :checked="todo.done"
        @change="toggleTodo(todo)">
      <label v-text="todo.text"
            @dblclick="editing = true">
      </label>
      <button class="destroy"
            @click="removeTodo(todo)">
      </button>
    </div>
    <input class="edit"
          v-show="editing"
          v-focus="editing"
          :value="todo.text"
          @keyup.enter="doneEdit"
          @keyup.esc="cancelEdit"
          @blur="doneEdit">
  </li>
</template>
```

```
import { mapActions } from 'vuex'
export default {
  name: 'Todo',
  props: ['todo'],
  data () {...},
  directives: {...},
  methods: {
    ...mapActions([
      'editTodo',
      'toggleTodo',
      'removeTodo'
    ]),
    doneEdit (e) {
      const value = e.target.value.trim()
      const { todo } = this
      if (!value) {
        this.removeTodo(todo)
      } else if (this.editing) {
        this.editTodo({
          todo,
          value
        })
        this.editing = false
      }
    },
    cancelEdit (e) {...}
  }
}
```

Mutations

- ▶ Les mutations Vuex ont un “type” et un “handler”
- ▶ Pour déclencher un mutation handler, on appelle store.commit
- ▶ On utilise des constantes pour les types des mutations
- ▶ Les mutation handlers sont des fonctions **synchrones**

Critiques des mutations



posva commented on 22 Jan 2020

Member

...

I think disallowing direct state modification is a rule that should be enforced at a linter level instead because runtime-wise this would only be a dev-only warning, so it would be slower during dev and require more code in the library

Being able to directly modify the state (or using `patch`) is intentional to lower the entry barrier and *scale down*. After many years using Vuex, most mutations were completely unnecessary as they were merely doing a single operation via an assignment (`=`) or collection methods like `push`. They were always perceived as verbose, no matter the size of the project, adding to the final bundle size as well, and were useful only when grouping multiple modifications and some logic, usually in larger projects. In Pinia, you are still able to have these groups by using actions but you get to choose if you need them or not, being able to start simple, and scale up when necessary.

So I want to make people realise `mutations` is an unnecessary concept thanks to actions and that the cost of mutations and its verbosity is not enough to justify the possibility of organisation it brings. On top of that, one of the main aspects of mutations was devtools: being able to go back and forward. In Pinia, all modifications are still tracked so doing direct state modification still allows you to keep things organised despite it wasn't taught that way with Vuex



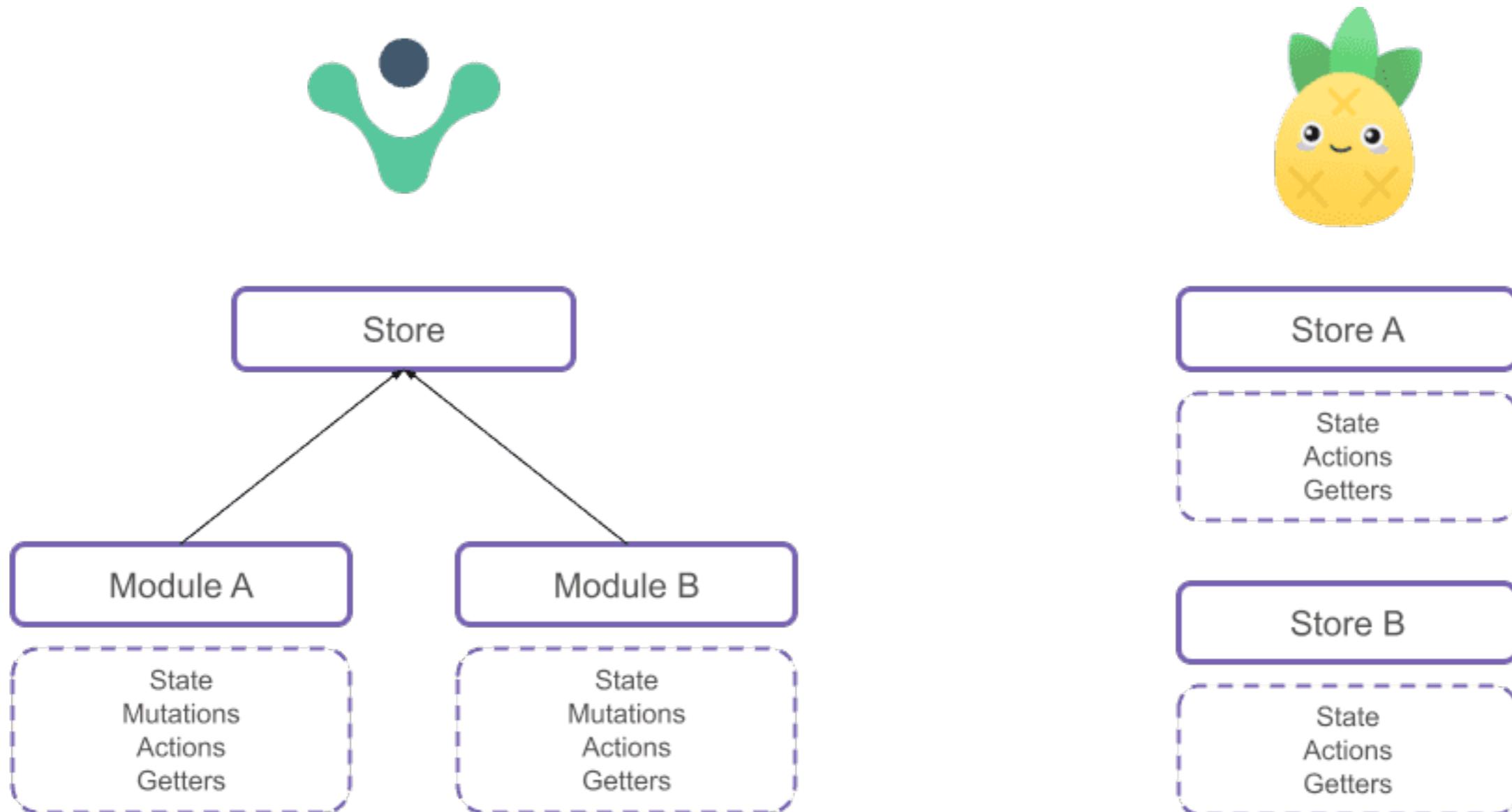
26



9

- ▶ Création de Pinia (Vuex5)
- ▶ Critique similaire côté Redux (React) -> création de redux-toolkit

Vuex4 vs. Pinia



Qui utilise Vue ?

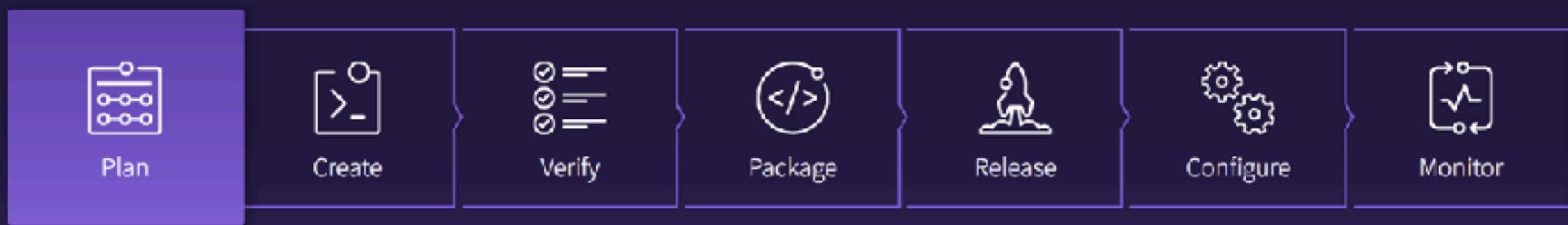
<https://about.gitlab.com/2016/10/20/why-we-chose-vue/>



From planning to monitoring

Create value faster with a single application for the whole software development and operations lifecycle.

[Try GitLab for Free](#)



Plan: Get your best ideas into development.

Whether you use Waterfall, Agile, or Conversational Development, GitLab streamlines your collaborative workflows. Visualize, prioritize, coordinate, and track your progress your way with GitLab's flexible

A screenshot of the GitLab web interface. The left sidebar shows 'Issues' (3,400) and 'Merge Requests' (498). The main area displays a board titled 'Development' with several cards. One card is titled 'Only first line of pre-processor and post-processor hooks-error is captured #10214' and another 'Error message shows up at the wrong place in repository settings. #20215'. Other cards include 'Do a better job of communicating when MR is blocked by a locked file. #10149', 'Unable to see user to add him to responsibilities #20211', 'Improving a failed build reports the status real fast after a previous failure. #20216', and 'No feedback when project limit is reached #20218'. The bottom right corner of the screenshot contains a note: 'The buttons to resolve a discussion are renamed on Firefox under my Debian Stretch 10.0.3.0'.

Plan

- ▶ Quoi et pourquoi la réactivité
- ▶ Quelles limites de MVC
- ▶ Architecture Flux
- ▶ Réactivité, Vue et Vuex
- ▶ **Traitement réactif de flux**

Les principes de base

- ▶ Responsive,
- ▶ Résilient,
- ▶ Élastique,
- ▶ Orienté message

Responsive

- ▶ Réponse en temps voulu, si possible
- ▶ Temps de réponses rapides et fiables (limites hautes)

Résilient

- ▶ Résiste à l'échec
- ▶ Principes :
RéPLICATION, conteneurs, isolement, délégation
- ▶ On fait en sorte qu'un échec n'impacte qu'un seul composant

Élastique

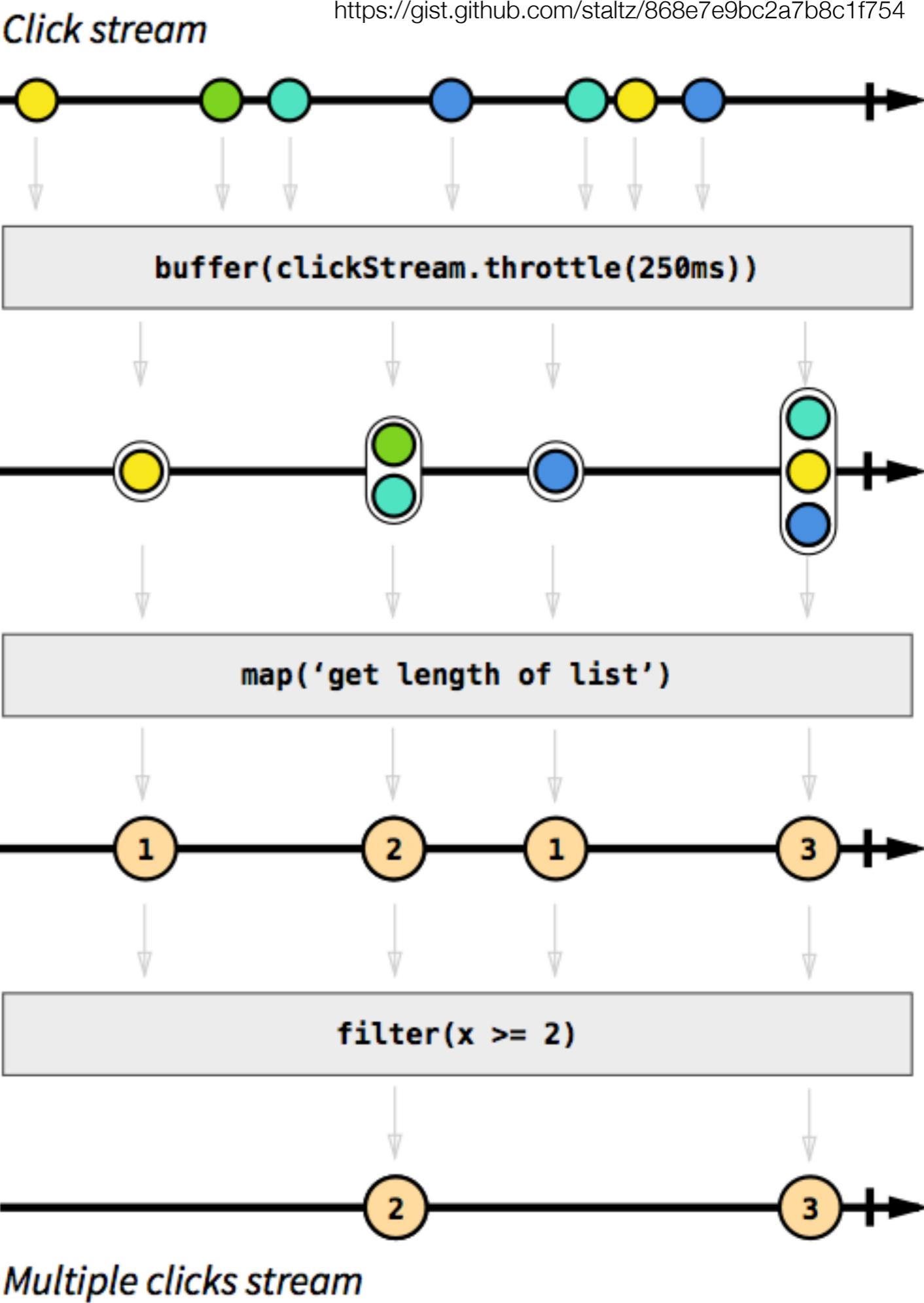
Le système reste réactif en cas de variation de la charge de travail.

- ▶ Pas de point central
- ▶ Pas de goulot
- ▶ Distribution des entrées entre composants

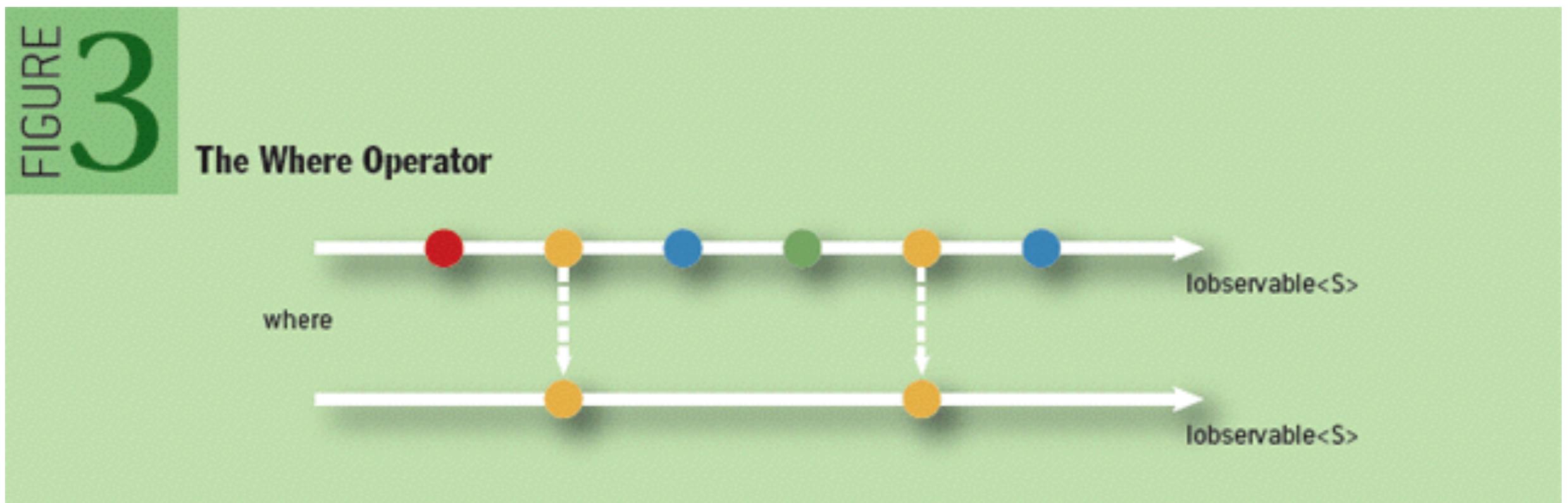
Orienté message

- ▶ Passage de messages asynchrones
 - > Couplage faible, isolation
- ▶ Pas de blocage, les composants consomment les ressources quand ils peuvent

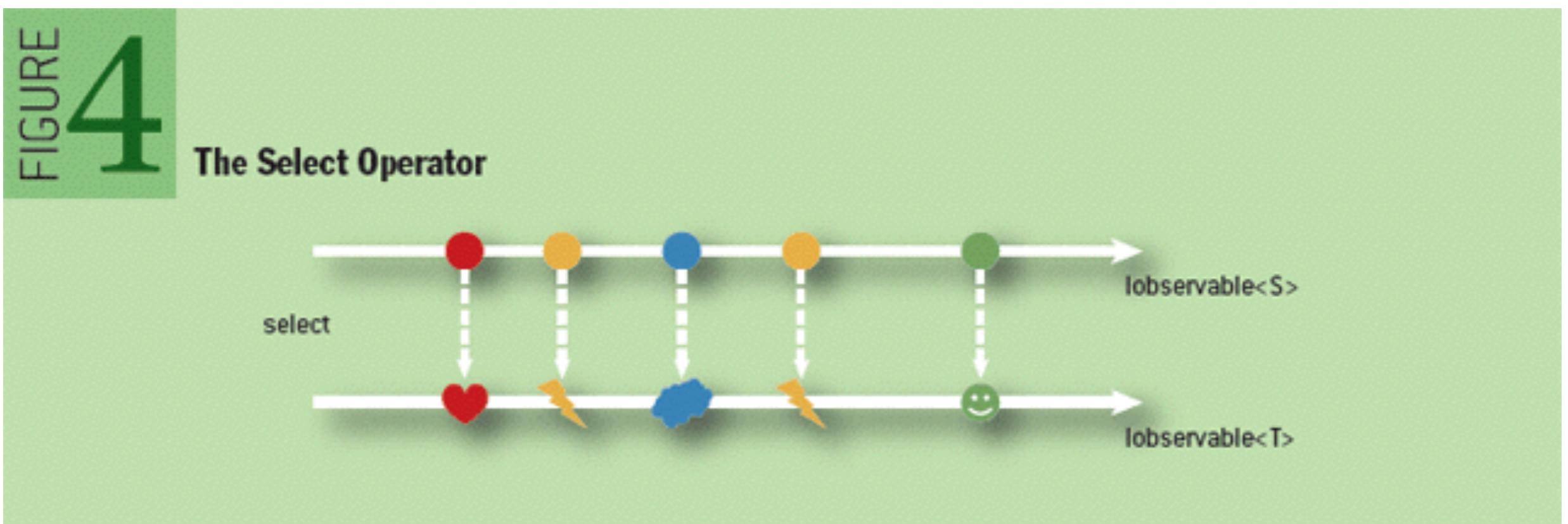
Un exemple de transformation



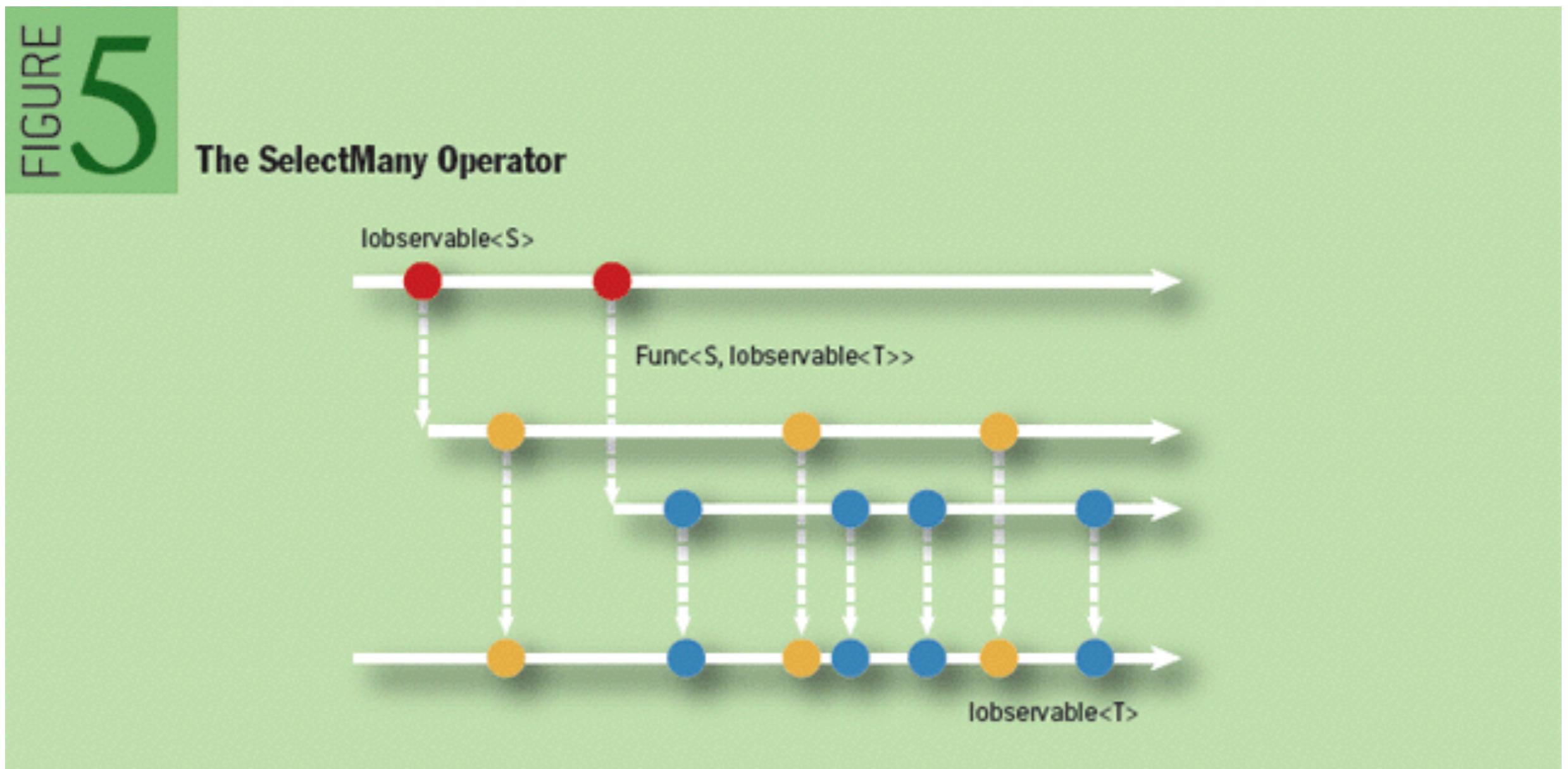
Where



Select



SelectMany : plusieurs flux



Throttle

