# Flux


- **Flux** is an **application architecture** for building **client-side web applications**.
- Introduced by **Facebook** to structure data flow in **React apps**.
- It is **not a library**, but a **pattern**.
- Key Idea: **Unidirectional Data Flow**

# Why Flux

Normally in React:

- Components can pass data **up and down (props & state)**.
- As the app grows, managing state across many components becomes messy → **"prop drilling"** and **inconsistent updates**.

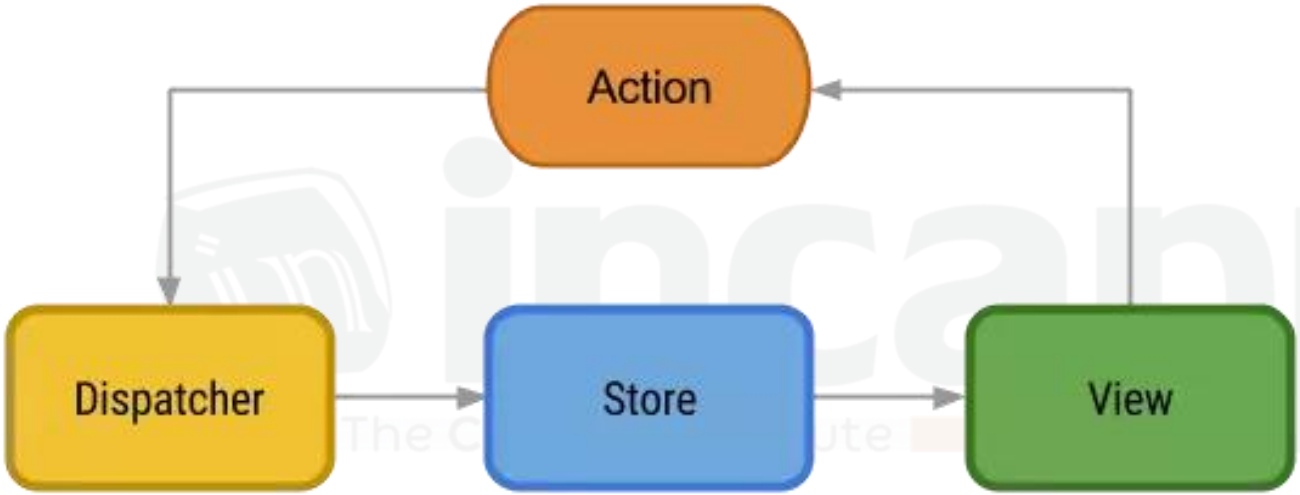
Flux solves this by:

- Enforcing **one-way data flow**.
- Centralizing updates via a **dispatcher** and **stores**.

**Real-World Use Cases**

- Facebook used Flux in **Messenger**.
- Useful when:
  - Large React app with complex state.
  - Many components need the same data.
  - Debugging unpredictable state updates.

# Flux Architecture



# Flux Architecture

Flux has **4 main parts**:

**1. Actions**

1. Represent **events** (e.g., user clicked button, data loaded).
2. They are simple objects with a **type** and **payload**.
   Example: { type: "ADD_TODO", text: "Learn Flux" }

**2. Dispatcher**

1. Central hub that receives **actions** and forwards them to the **stores**.
2. Ensures **all stores get the same action**.

**3. Stores**

1. Manage **application state & logic**.
2. Respond to actions, update data, and emit change events.
   Example: A **TodoStore** stores all todos.

**4. View (React Components)**

1. React components listen to stores.
2. When stores update → components re-render.

# Assignment

Build a **Shopping Cart App** using Flux:

- **Actions:** ADD_ITEM, REMOVE_ITEM, CLEAR_CART.
- **Store:** Manages cart items & total price.
- **Components:** Cart view + Add item form.