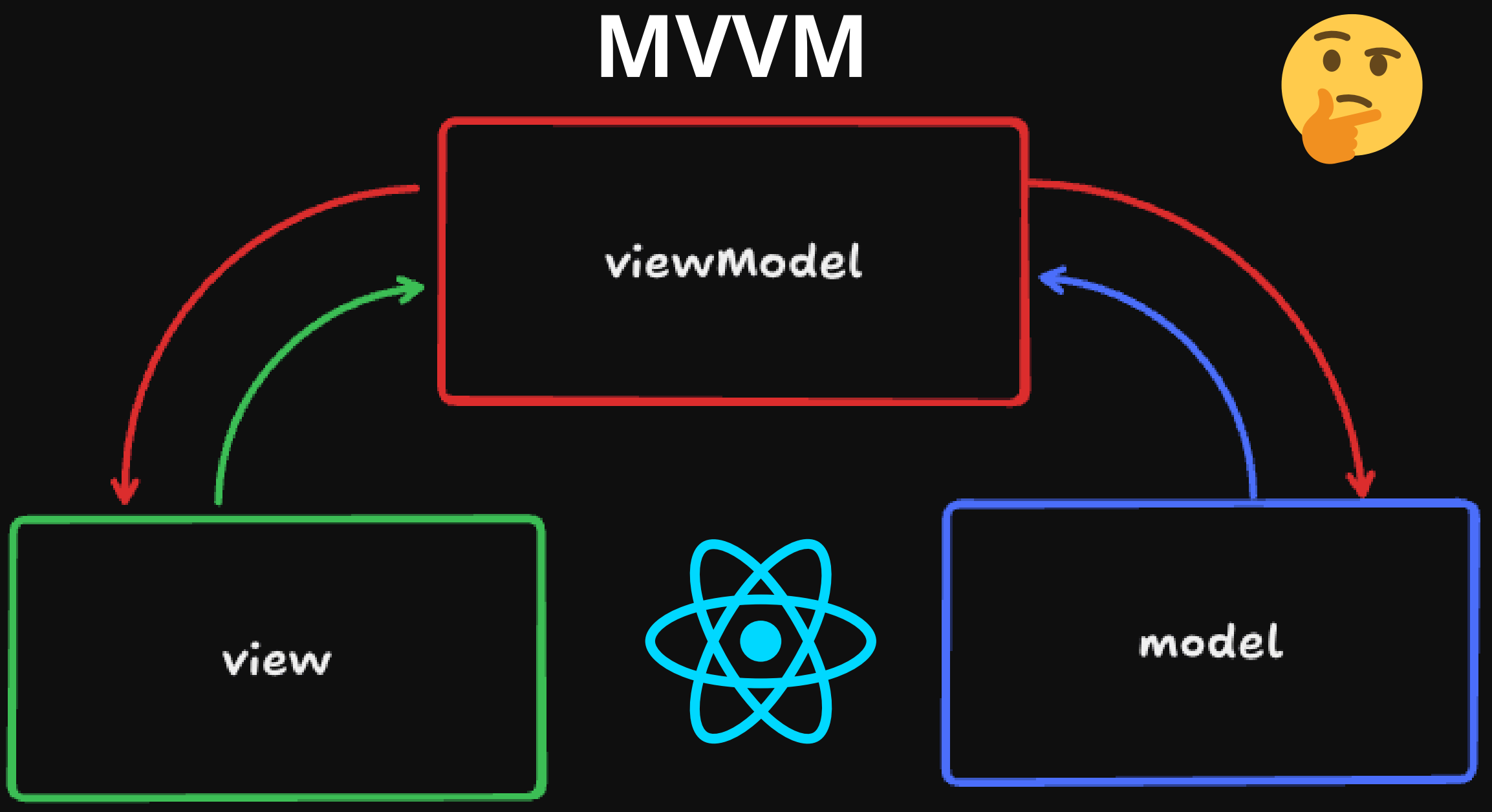
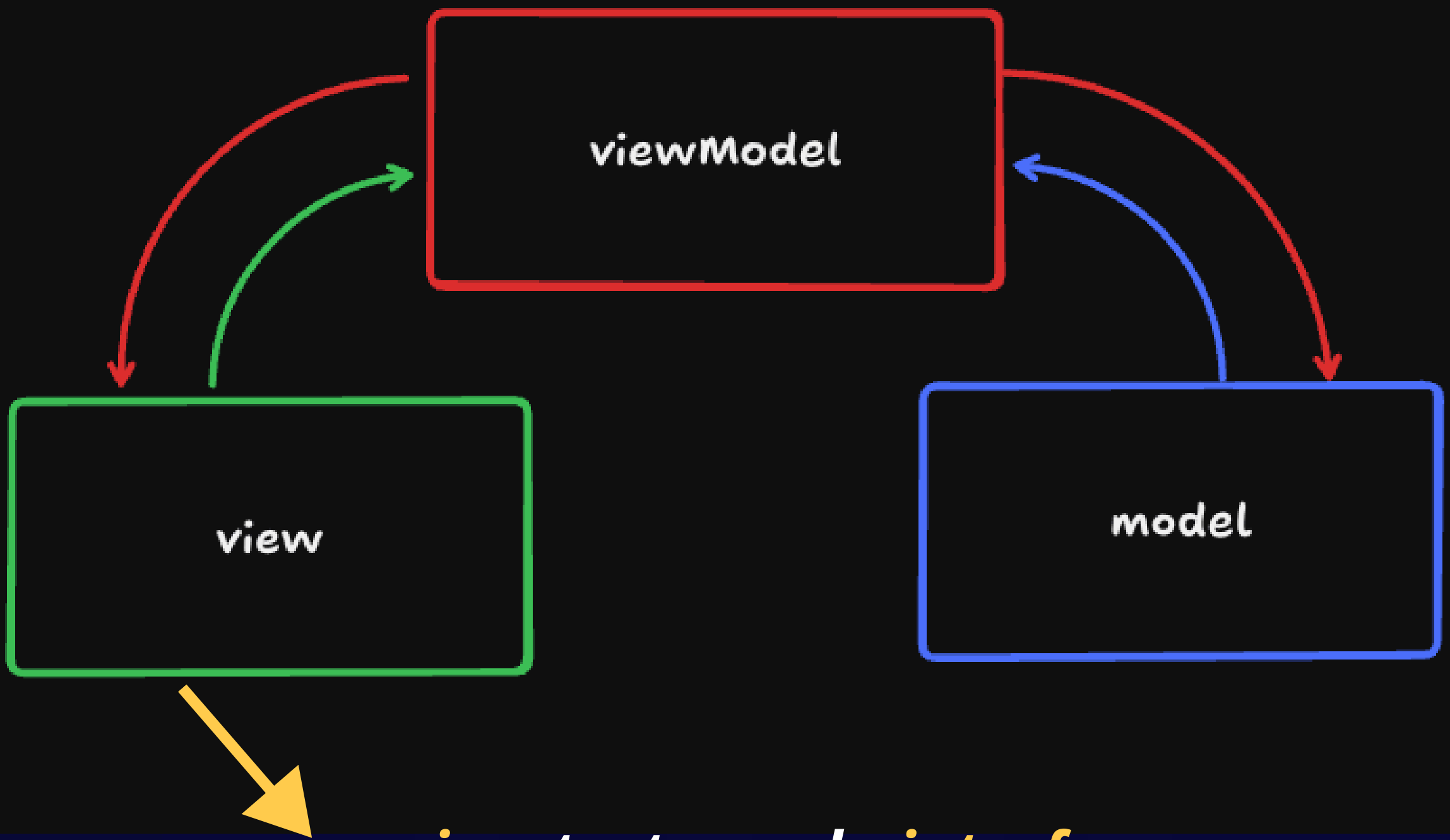
Desvendando arquitetura MVVM



E qual a diferença para custom hook Patterns ???

Isaac Gomes

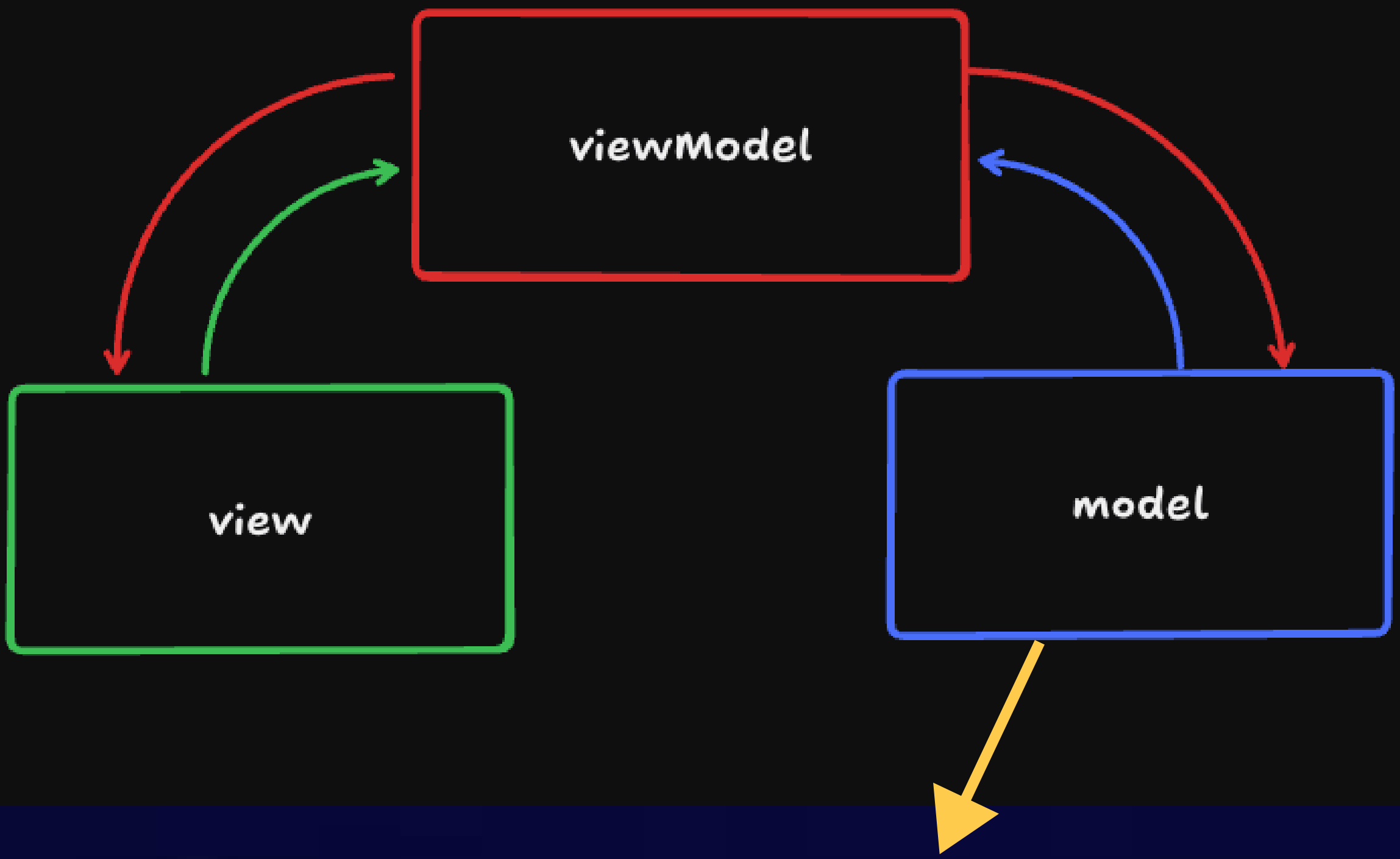
o que é MVVM ?



*a **view** trata-se da **interface** que o usuario sem logica. essa interface em termos gerais deve ser **burra** contendo a **menor** quantidade de **logica** possível*

Isaac Gomes

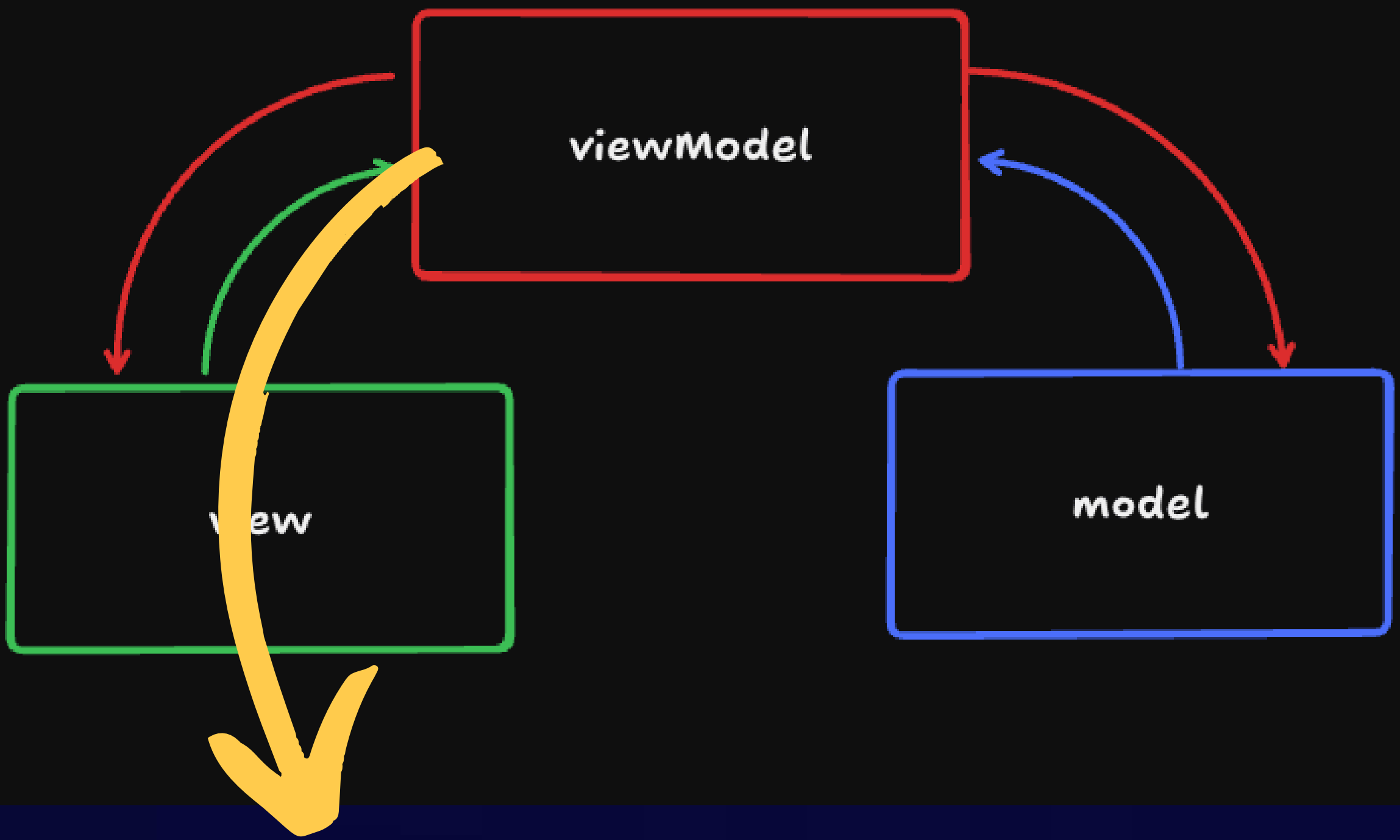
o que é MVVM ?



*o **Model** o model não contem nada visual somente **lógica de negócios** da aplicação*

Isaac Gomes

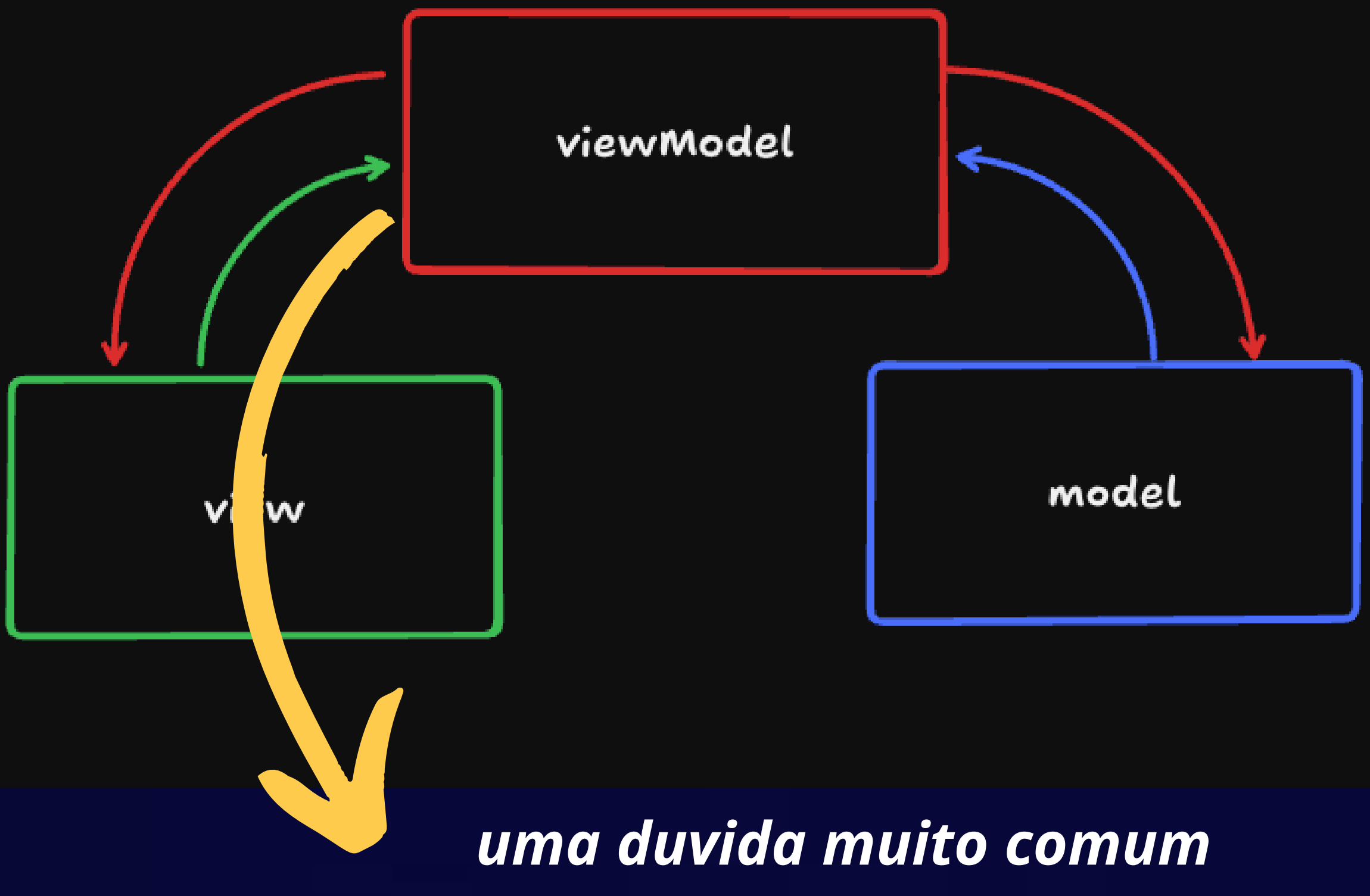
o que é MVVM ?



a viewModel Expõe os dados e os comandos que a View pode ligar (binding) ou seja ela Atua como um intermediário entre a View e o Model

Isaac Gomes

o que é MVVM ?

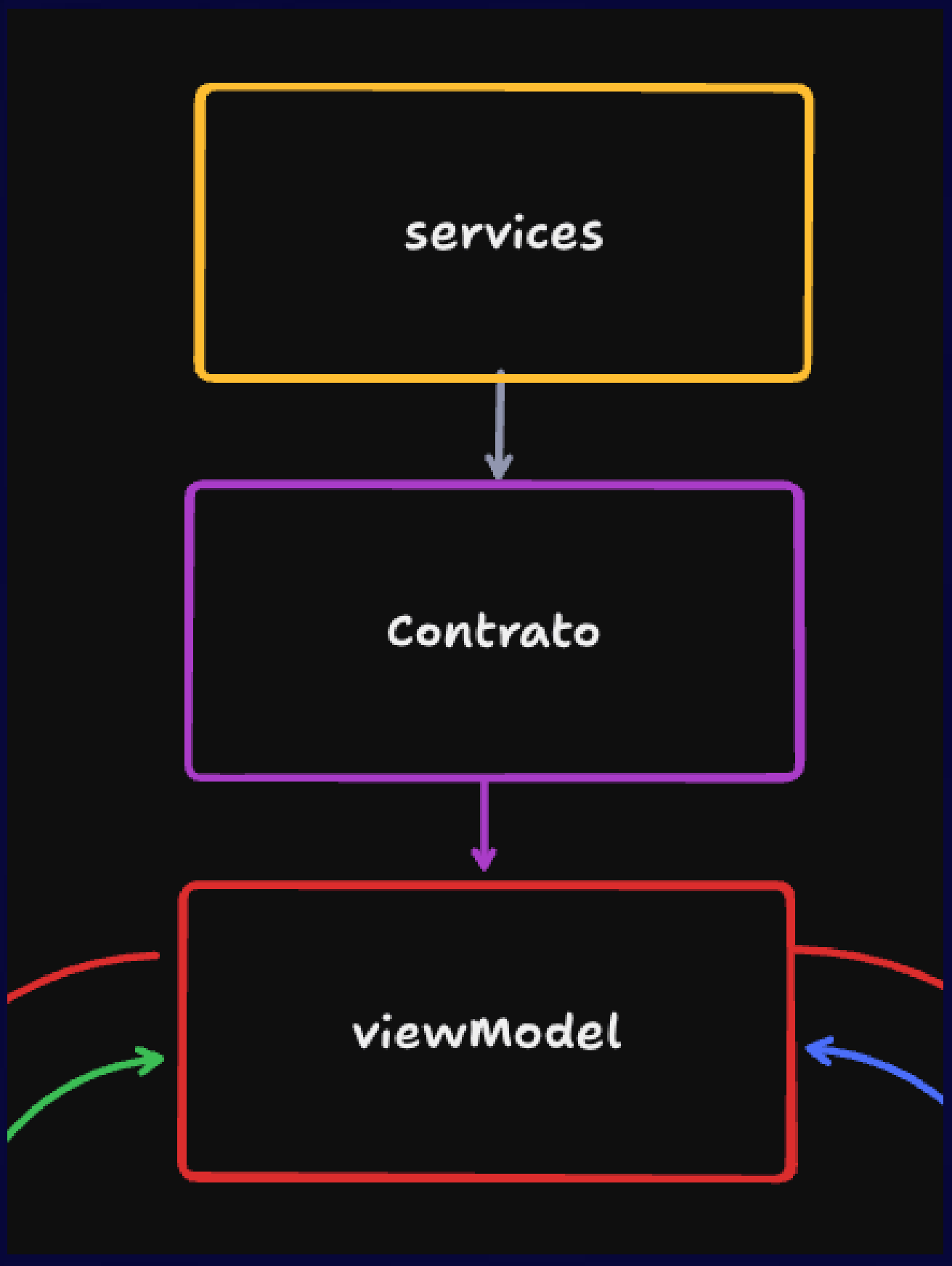


uma duvida muito comum

é qual a diferença entre
MVVM e Custom Hook Patterns??
afinal para que essa
viewModel no meio serve???

Isaac Gomes

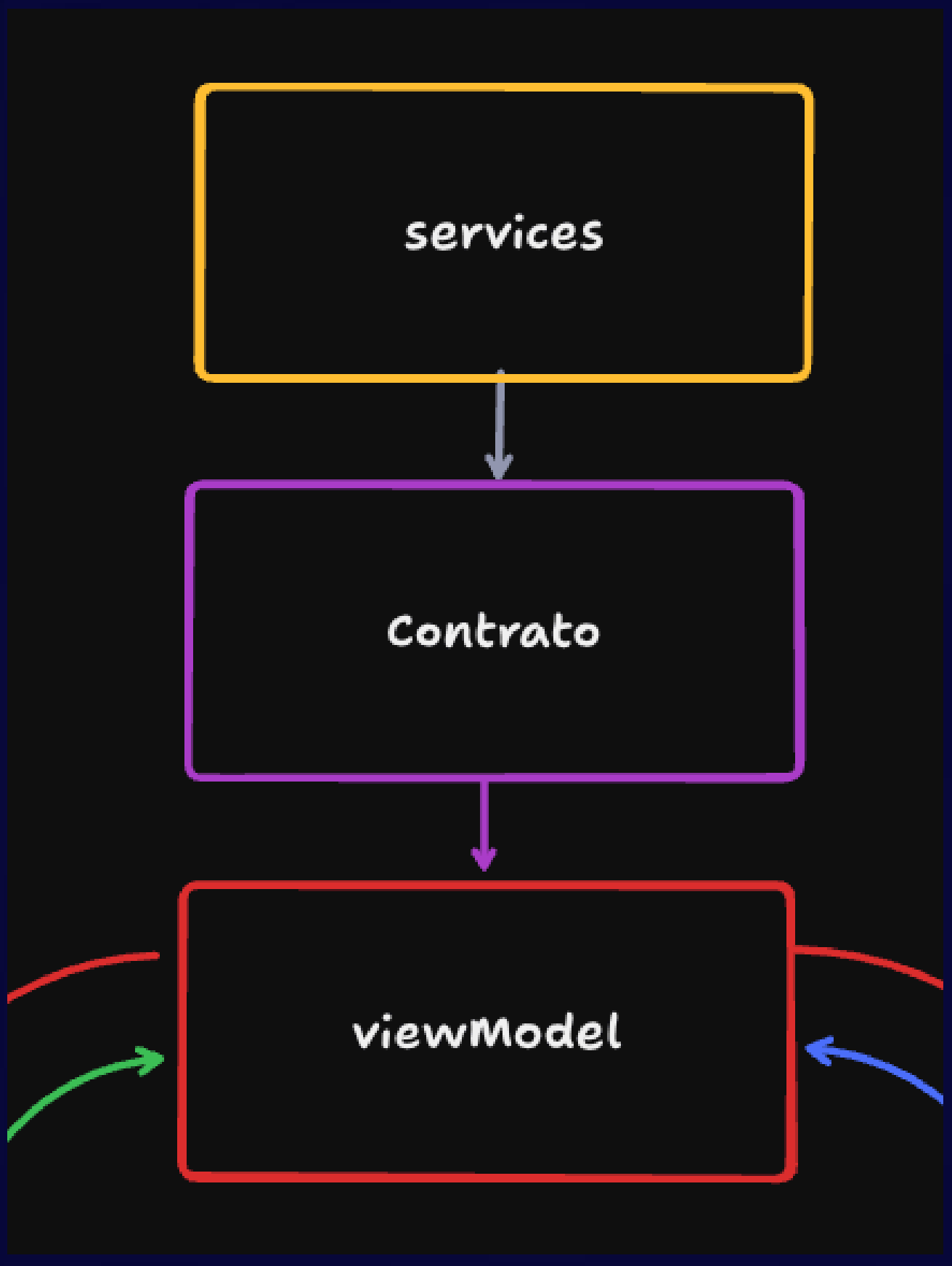
o que é MVVM ?



a **viewModel** alem de unir a **view** e o **model** pode ser um **orquestrado de recursos**, pense que vai acessar uma API podemos criar **componentes** muito claros do nosso **Software**

Isaac Gomes

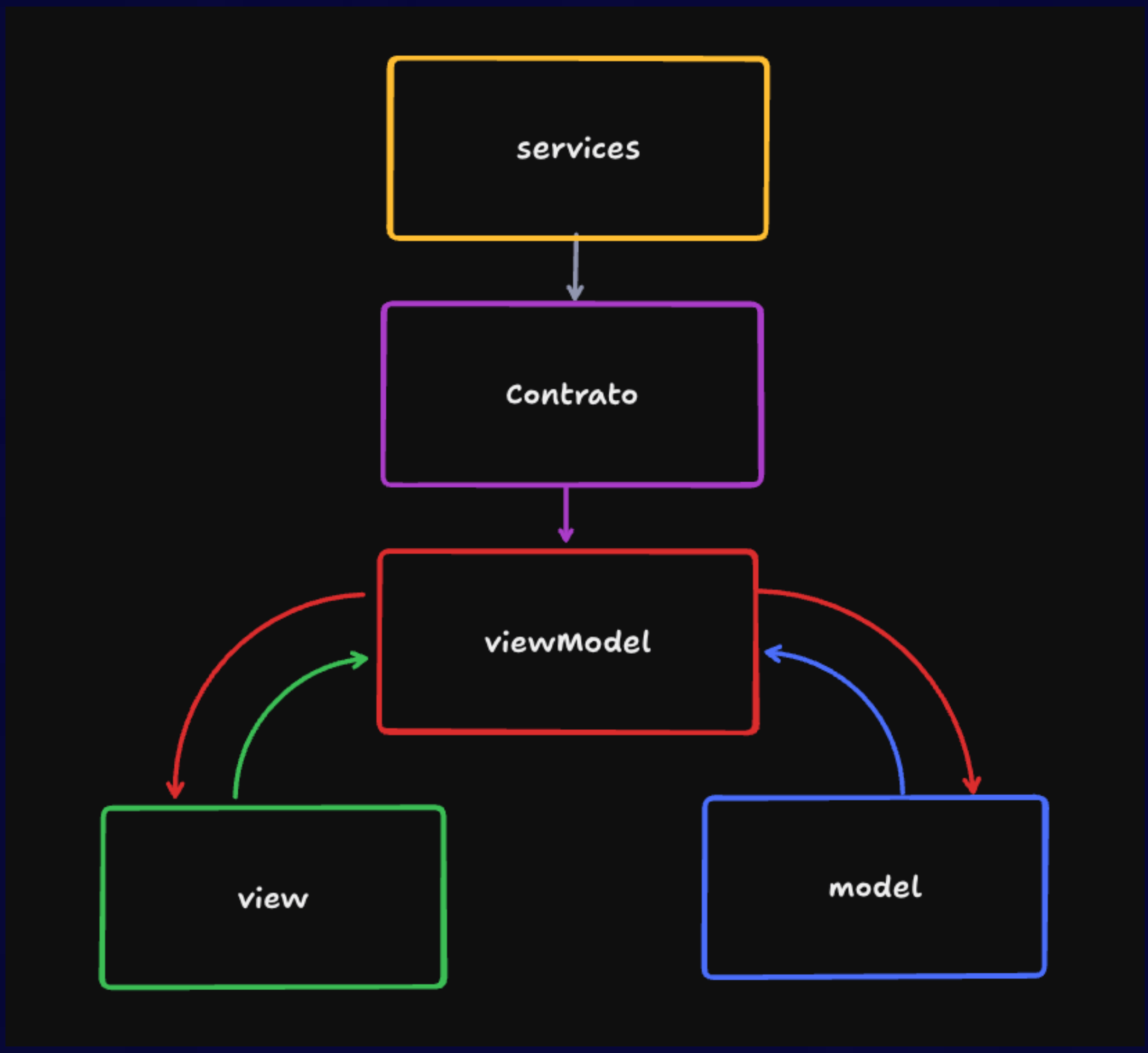
o que é MVVM ?



a **viewModel** alem de unir a **view** e o **model** pode ser um **orquestrado de recursos**, pense que vai acessar uma API podemos criar **componentes** muito claros do nosso **Software**

Isaac Gomes

o que é MVVM ?



Isaac Gomes

o que é MVVM ?

agora vamos montar um exemplo
para ficar mais claro

cenário

vamos criar um todo list
com as funções de listar,
adicionar e deletar

Isaac Gomes

1 - criar uma interface para as funções de acesso a API

```ts
export interface ITaskService {
  create: (text: string) => Promise<void>
  remove: (id: string) => Promise<void>
  getAll: () => Promise<Array<Task>>
}
```

Isaac Gomes

2 - Criar o Service de acesso a API

```
export class TaskService implements ITaskService {
  async create(text: string) {
    const { data } = await apiCore.post('/task/', { text })
    return data
  }

  async remove(id: string) {
    const { data } = await apiCore.delete(`/task/${id}`)
    return data
  }

  async getAll() {
    const { data } = await apiCore.get('/task')
    return data
  }
}
```

Isaac Gomes

TS

3 - criar a logica do nosso TODO usando a interface definida de acesso a API

```
export function useTodo(service: ITaskService) {
  const [taskText, setTaskText] = useState<string>('')
  const queryClient = useQueryClient()

  const invalidateTaskList = () => {
    queryClient.invalidateQueries({ queryKey: [QUERY_KEY.TODO_LIST] })
  }

  const { data: ListTasks, isLoading: isLoadinglistTask } = useQueryListTask({
    service: service.getAll,
  })

  const { mutate: mutateAddTask } = useMutationAddTask({
    service: service.create,
    onSuccess: invalidateTaskList,
  })

  const { mutate: mutateDeleteTask } = useMutationRemoveTask({
    service: service.remove,
    onSuccess: invalidateTaskList,
  })

  const handleAddTask = () => {
    if (!taskText) return
    mutateAddTask(taskText)
    setTaskText('')
  }

  return {
    taskText, setTaskText, ListTasks,isLoadinglistTask,
    handleAddTask,mutateDeleteTask,
  }
```

Isaac Gomes

4 - criar a nossa interface/view

```tsx
export function TodoView(props: ReturnType<typeof useTodo>) {
  const { taskText, setTaskText, ListTasks, handleAddTask } = props
  return (
    <div>
      <div>
        <input
          type="text"
          value={taskText}
          onChange={({ target: { value } }) => setTaskText(value)}
        />
        <button onClick={handleAddTask}> adiconar </button>
      </div>

      <div>{ListTasks?.length}</div>
      <div>
        {ListTasks?.map((task) => (
          <div key={task.id}>
            <p>{task.text}</p>
            <button>deletar</button>
          </div>
        ))}
      </div>
    </div>
  )
}
```

Isaac Gomes

TS

5 - criar nossa viewModel orquestrando nossos Componentes desenhados

```tsx
export default function Todo() {
  const taskService = new TaskService()
  const methods = useTodo(taskService)

  return <TodoView {...methods} />
}
```

Isaac Gomes

6 - agora vamos criar uma implementação falsa do acesso a API para fazer os testes

```
let tasks: Array<Task> = []
export function useTaskServiceMock(): ITaskService {
  const create = async (text: string): Promise<void> => {
    const task: Task = {
      id: String(Math.random()),
      text,
    }
    tasks = [...tasks, task]
  }

  const remove = async (id: string): Promise<void> => {
    tasks = tasks.filter((item) => item.id !== id)
  }

  const getAll = async (): Promise<Task[]> => {
    return tasks
  }

  return { create, remove, getAll }
}
```

aqui começa a ficar interessante vemos que estamos **mockando** um acesso a **API** e criando o **comportamento** esperado

Isaac Gomes

o **MVVM** é sempre mencionada pela **testabilidade**, agora vamos ver isso na pratica

quero testar meu Model

```
export const ModelTest = () => {
  const taskServiceMock = useTaskServiceMock()
  const methods = useTodo(taskServiceMock)
  return methods
}
```

Pronto podemos testar nossa logica

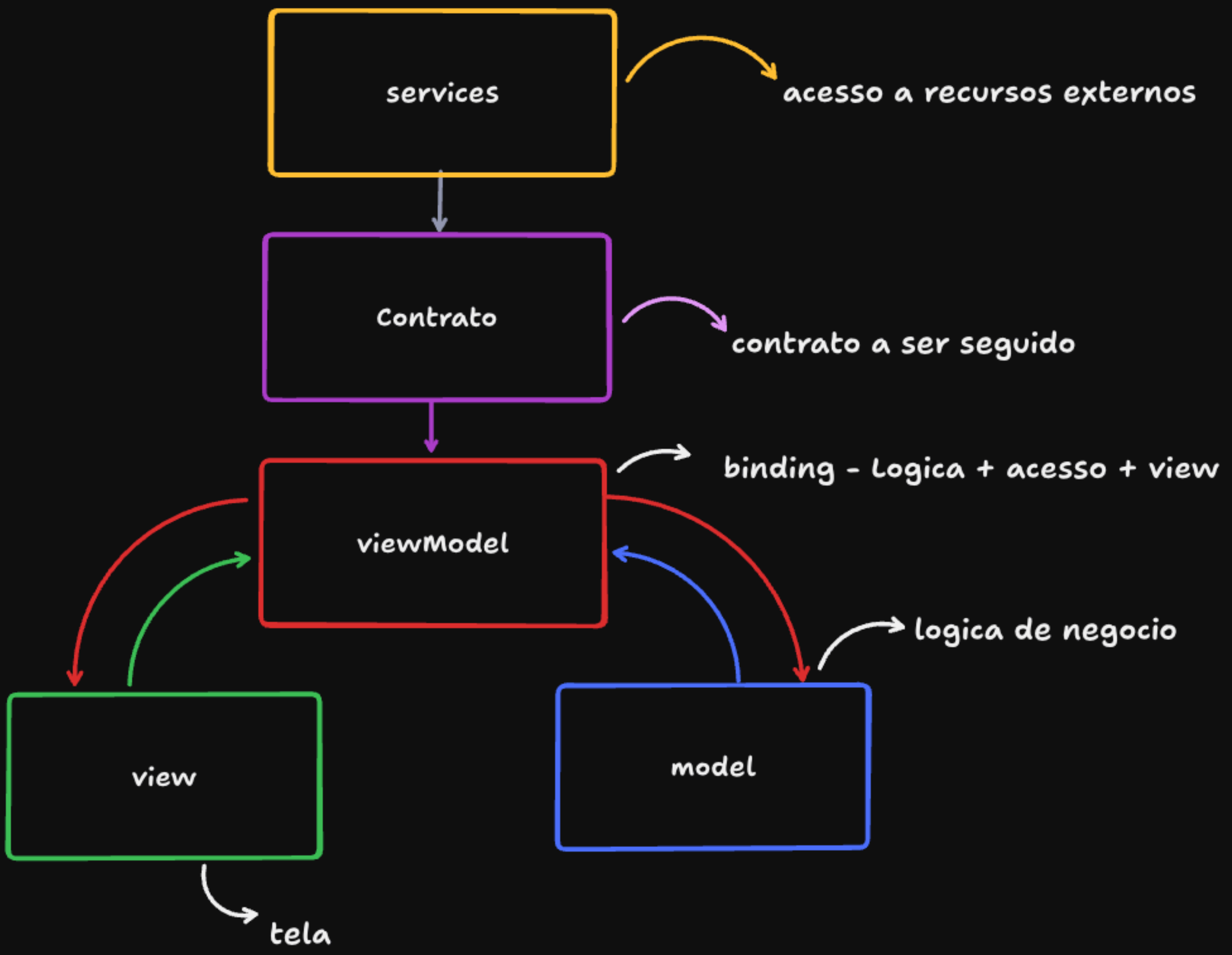
quero testar minha viewModel

```
export const ViewModelTest = ()=> {
  const taskServiceMock = useTaskServiceMock()
  const methods = useTodo(taskServiceMock)
  return <TodoView {...methods} />
}
```

pronto podemos testar nossa viewModel e conferir se a união esta como esperado

Isaac Gomes

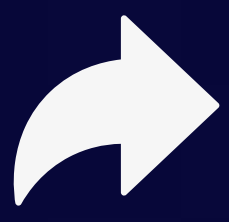
componentes da arquitetura



Isaac Gomes

Gostou?

Curta



Compartilhe

Salve

Isaac Gomes