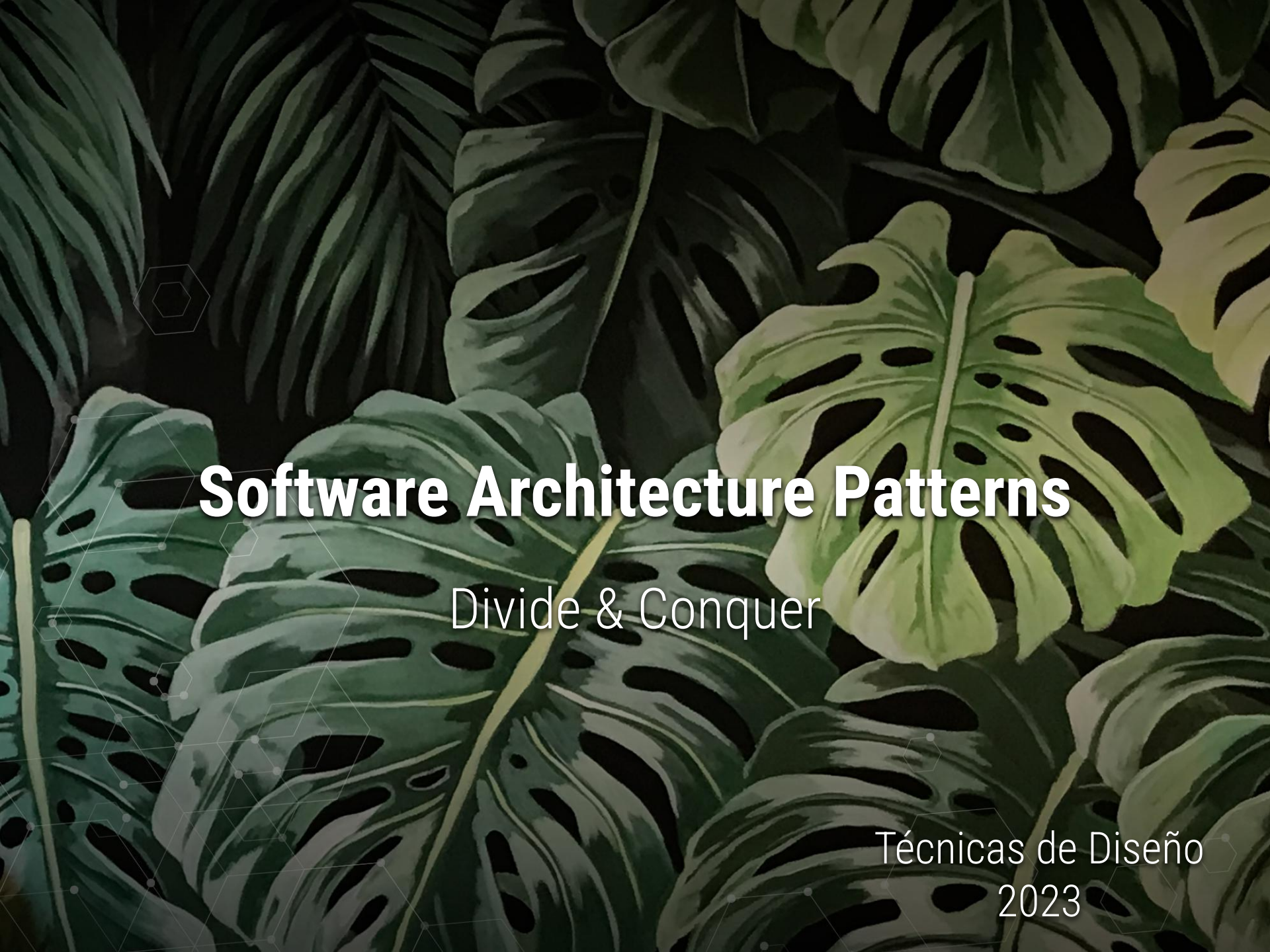


Software Architecture Patterns

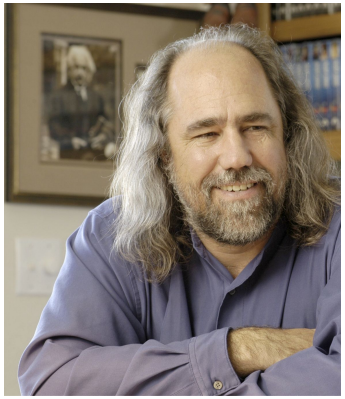
Divide & Conquer

Técnicas de Diseño
2023

Table of contents


Introduction

What is Software Architecture?



*"Architecture represents the set of significant **design decisions** that shape the form and function of a system, **where significant is measured by cost of change**"*
[Grady Booch]

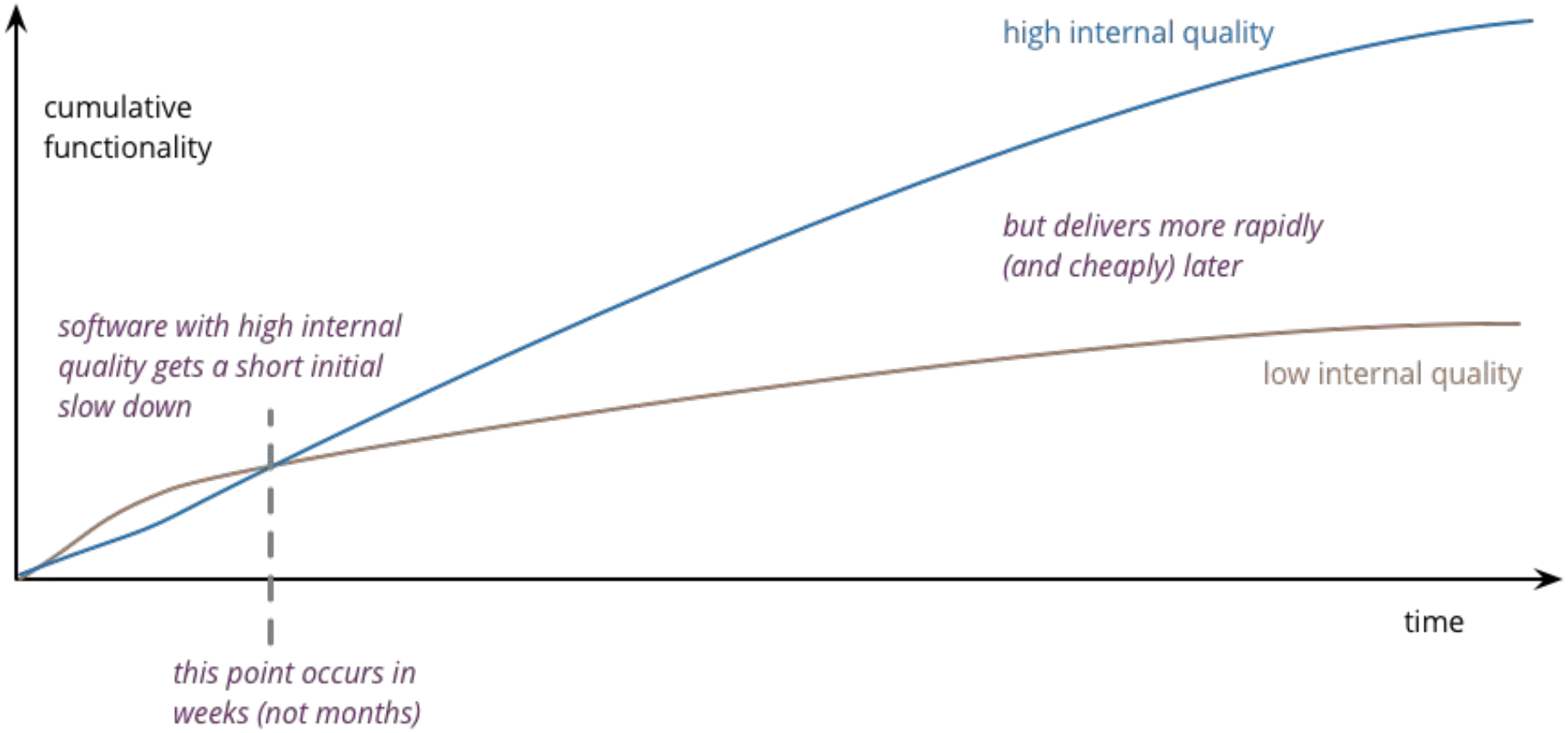
*"Architecture is **the important** stuff, whatever that happens to be."*
[Martin Fowler]

*"The architecture of a software system is **the shape** given to that system by those who build it. The form of that shape is in **the division of that system into components**, the arrangement of those components, and the ways in which **those components communicate with each other**."*
[Robert C. Martin / Uncle Bob]

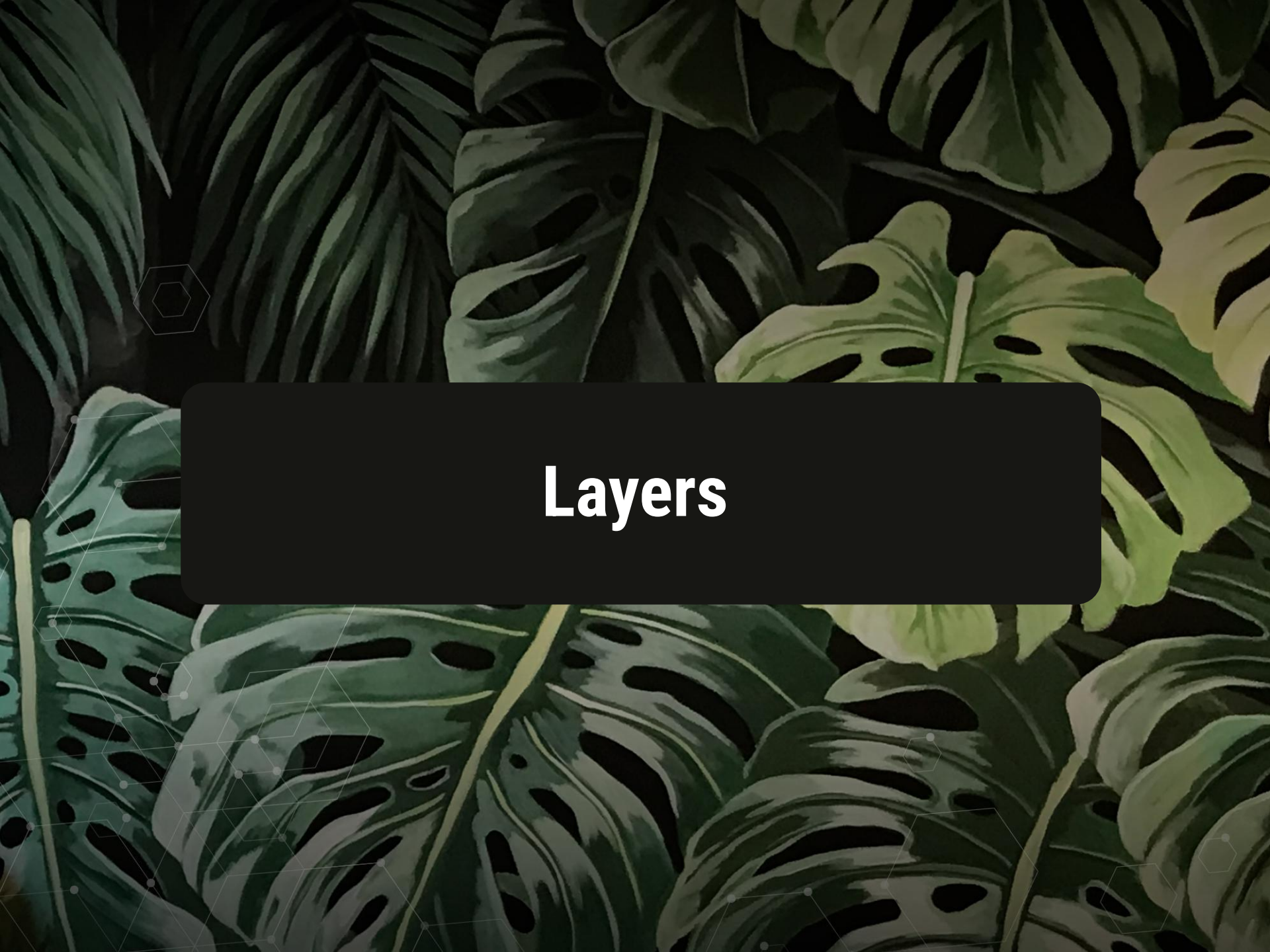
What is Software Architecture?

- Shape the system
- Significant design decisions
- Measured by cost of change
- The division of the system into components
- Communication between components

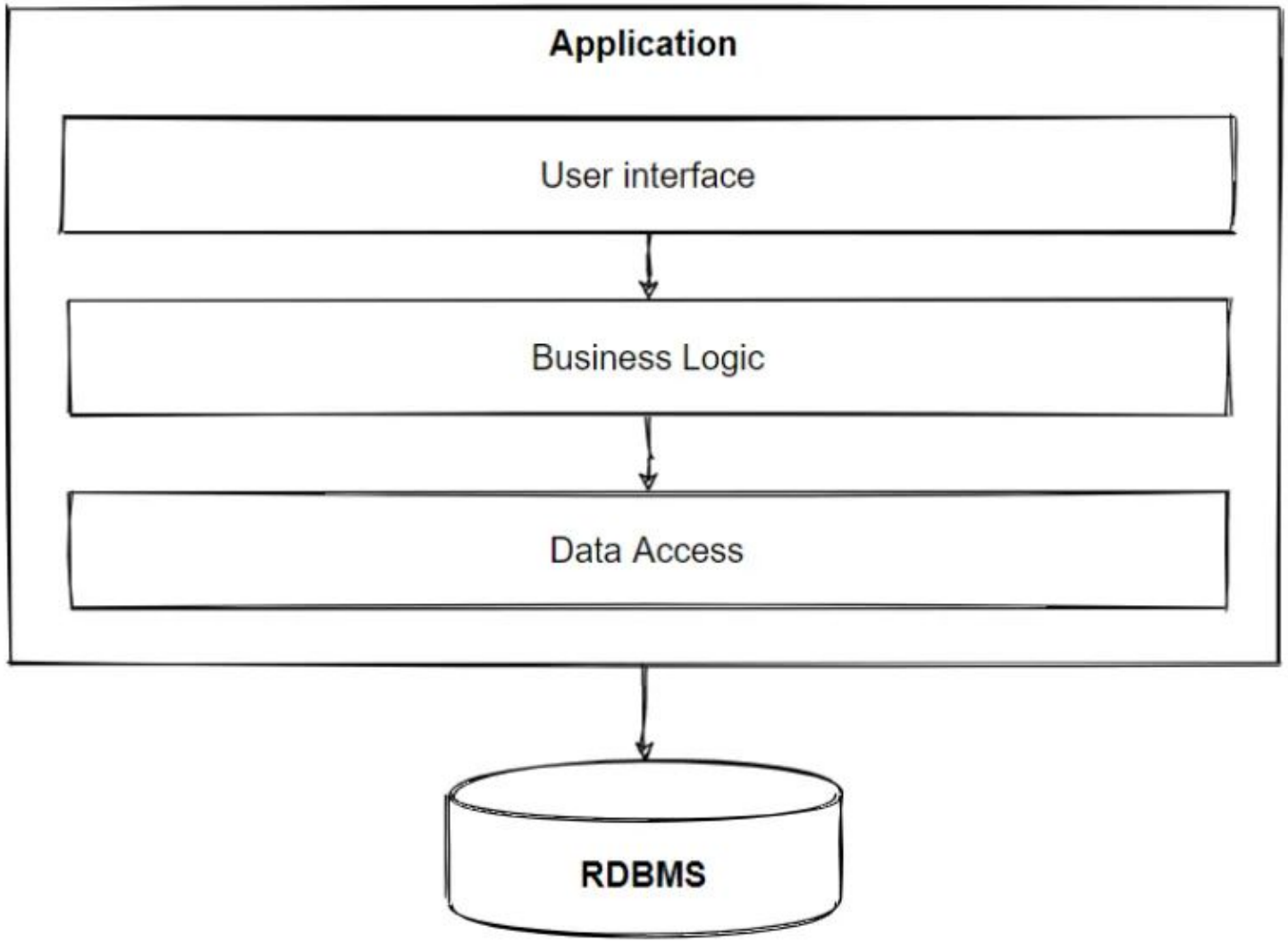
Why does architecture matter?



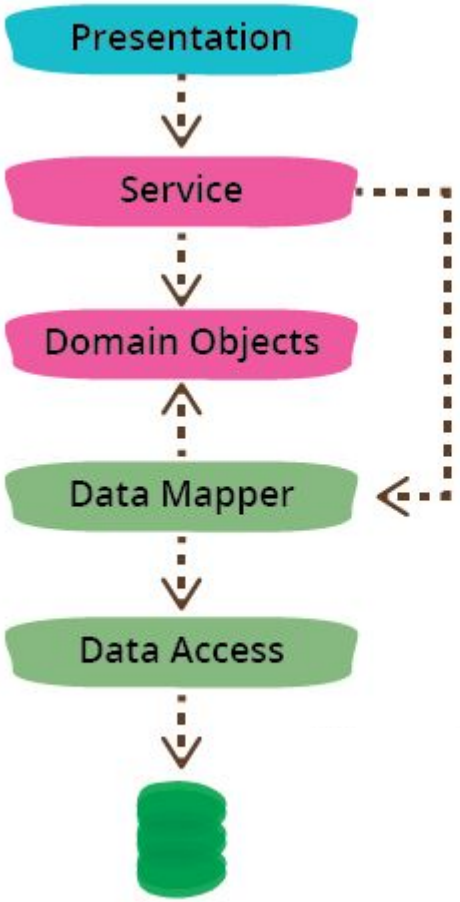

Architecture Patterns

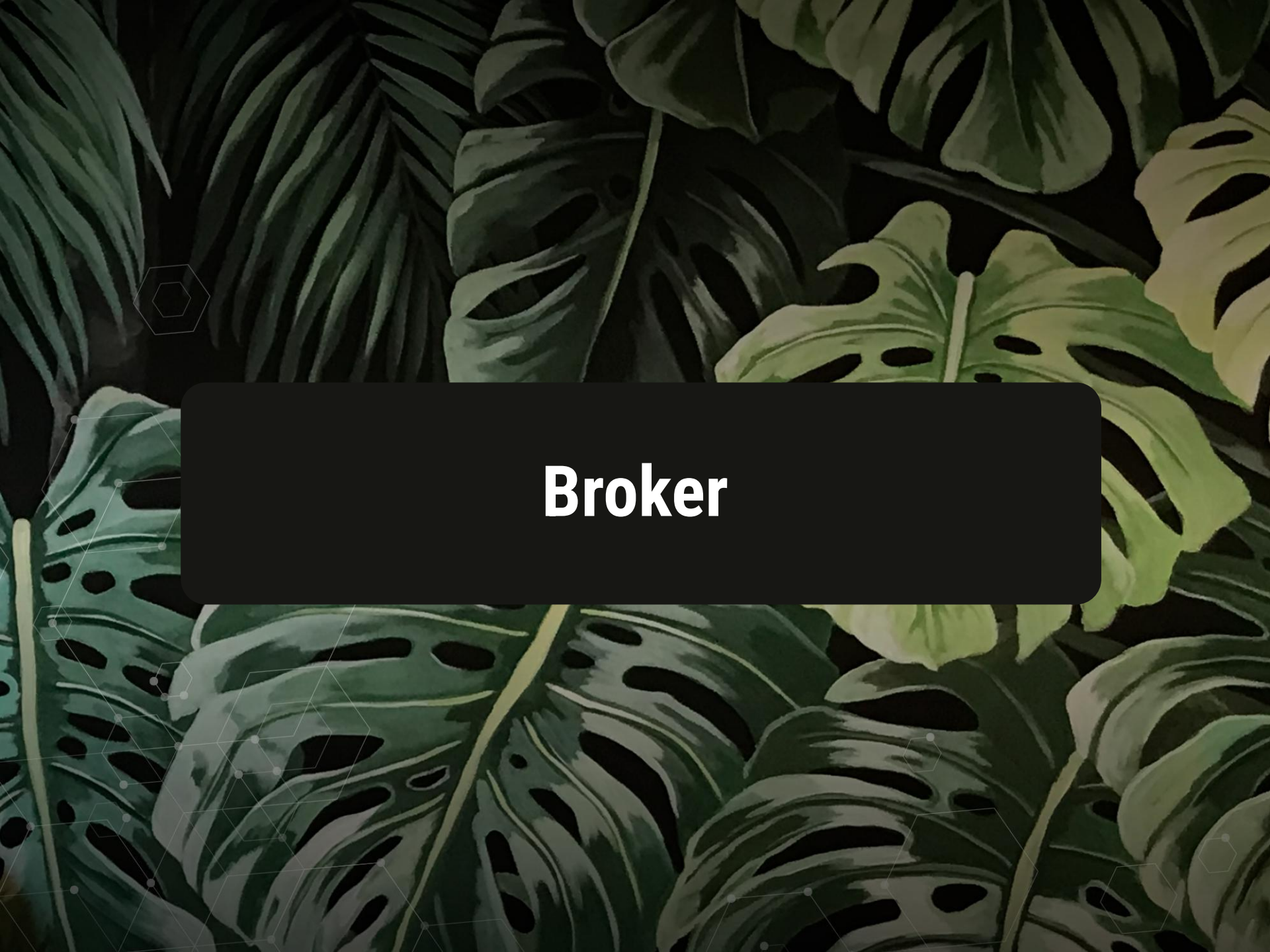

Layers

Layers

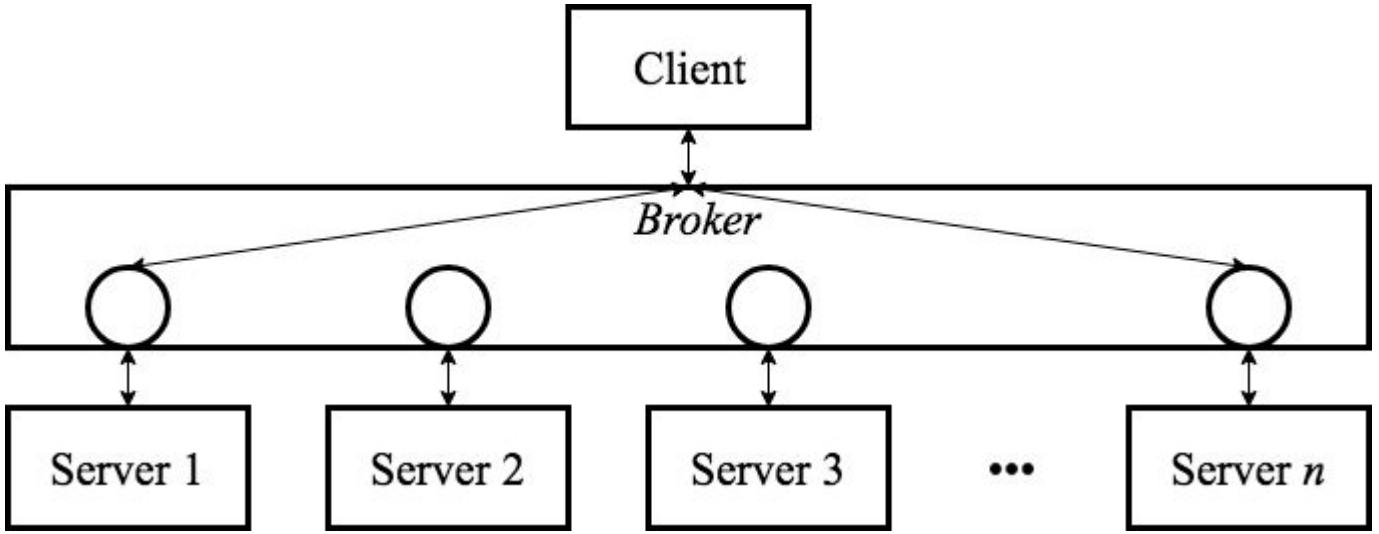

Application
User interface
Business Logic
Data Access
RDBMS

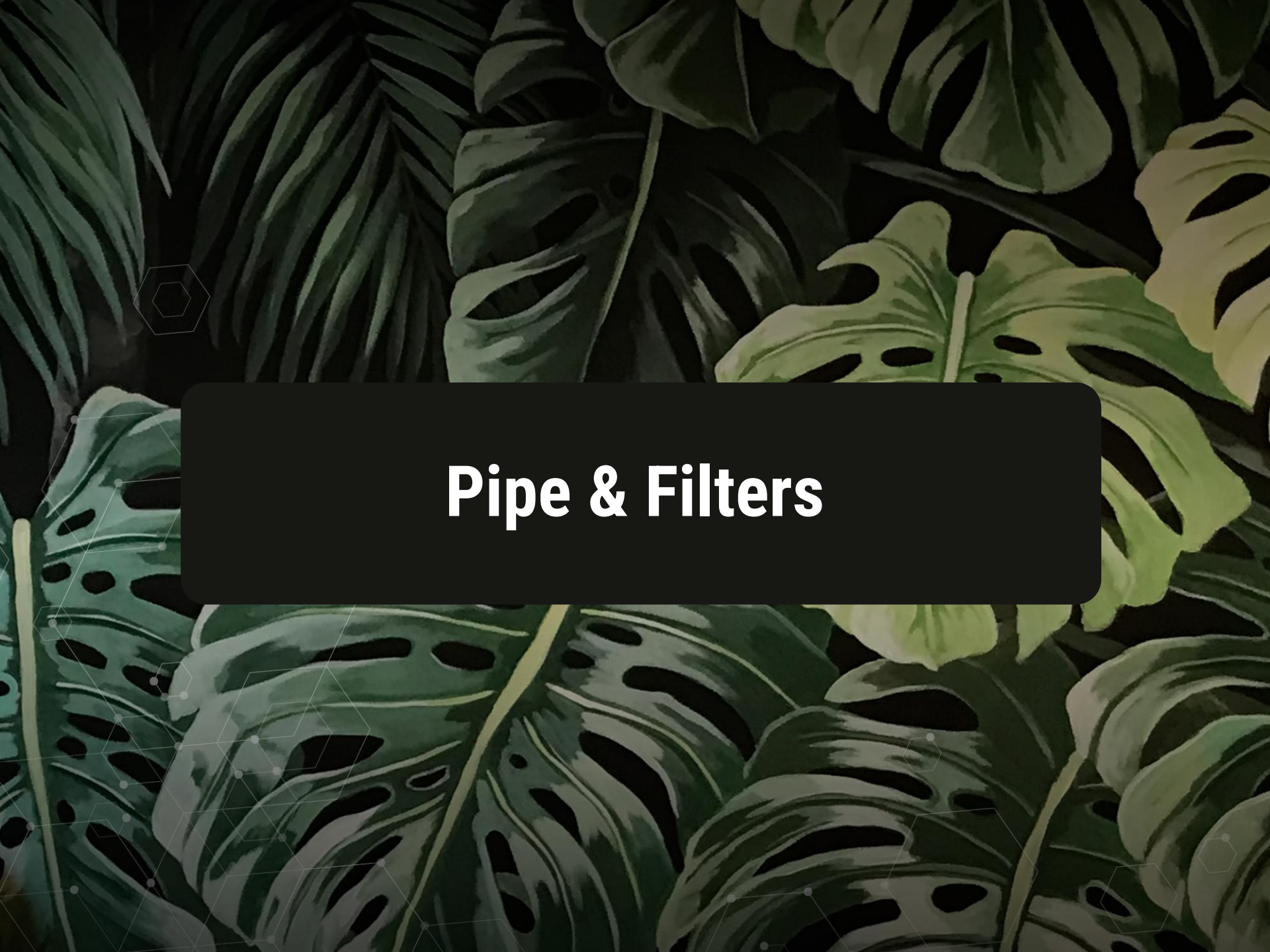
Layers


Presentation
Service
Domain Objects
Data Mapper
Data Access

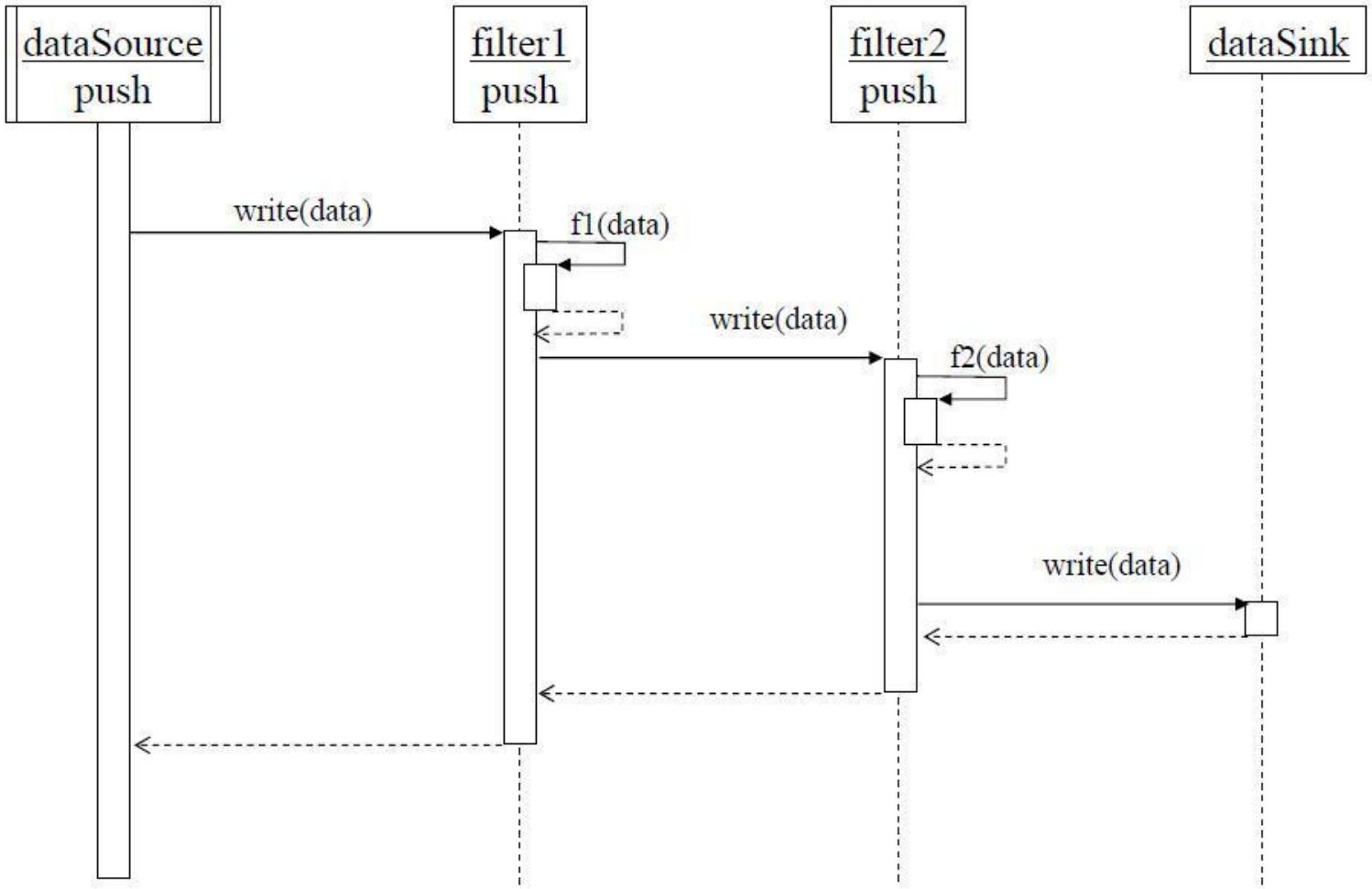

Broker

Broker




Pipe & Filters

Pipe & Filters



HTTP Filters

Unix Commands

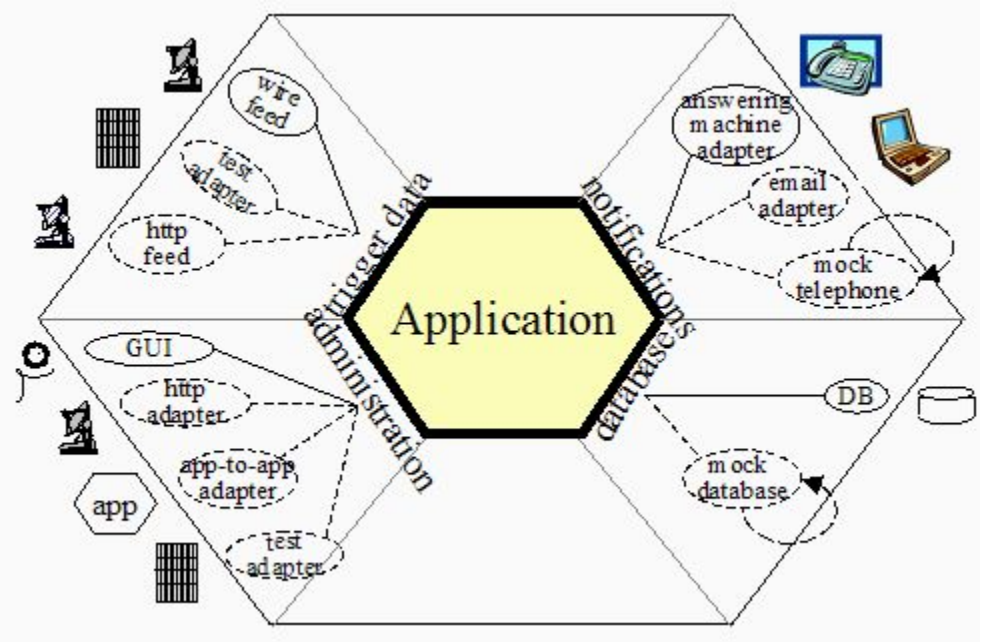
Compiler


Hexagonal Architecture

Hexagonal Architecture (Ports & Adapters)

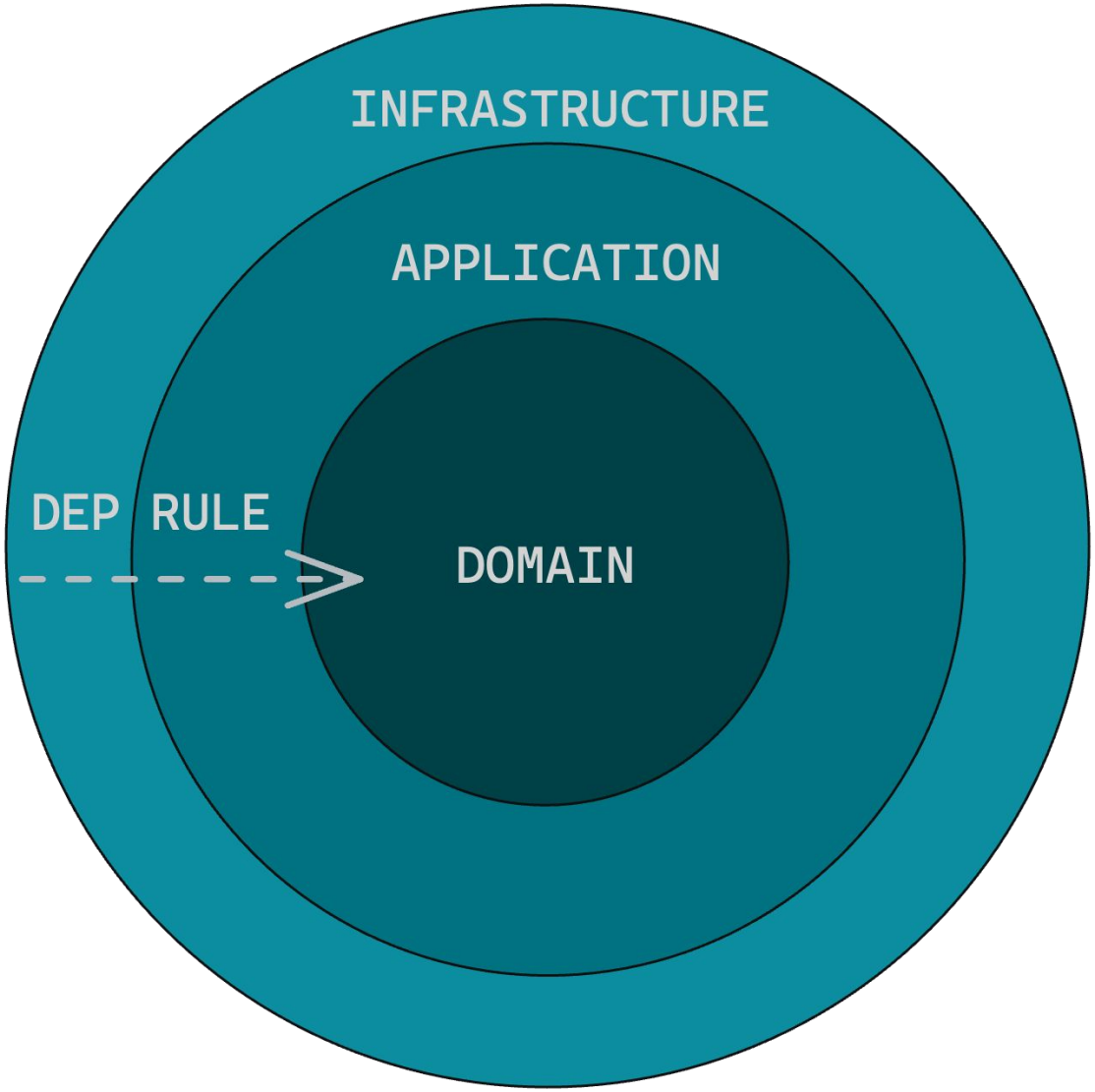
Why a Hexagon?

2005

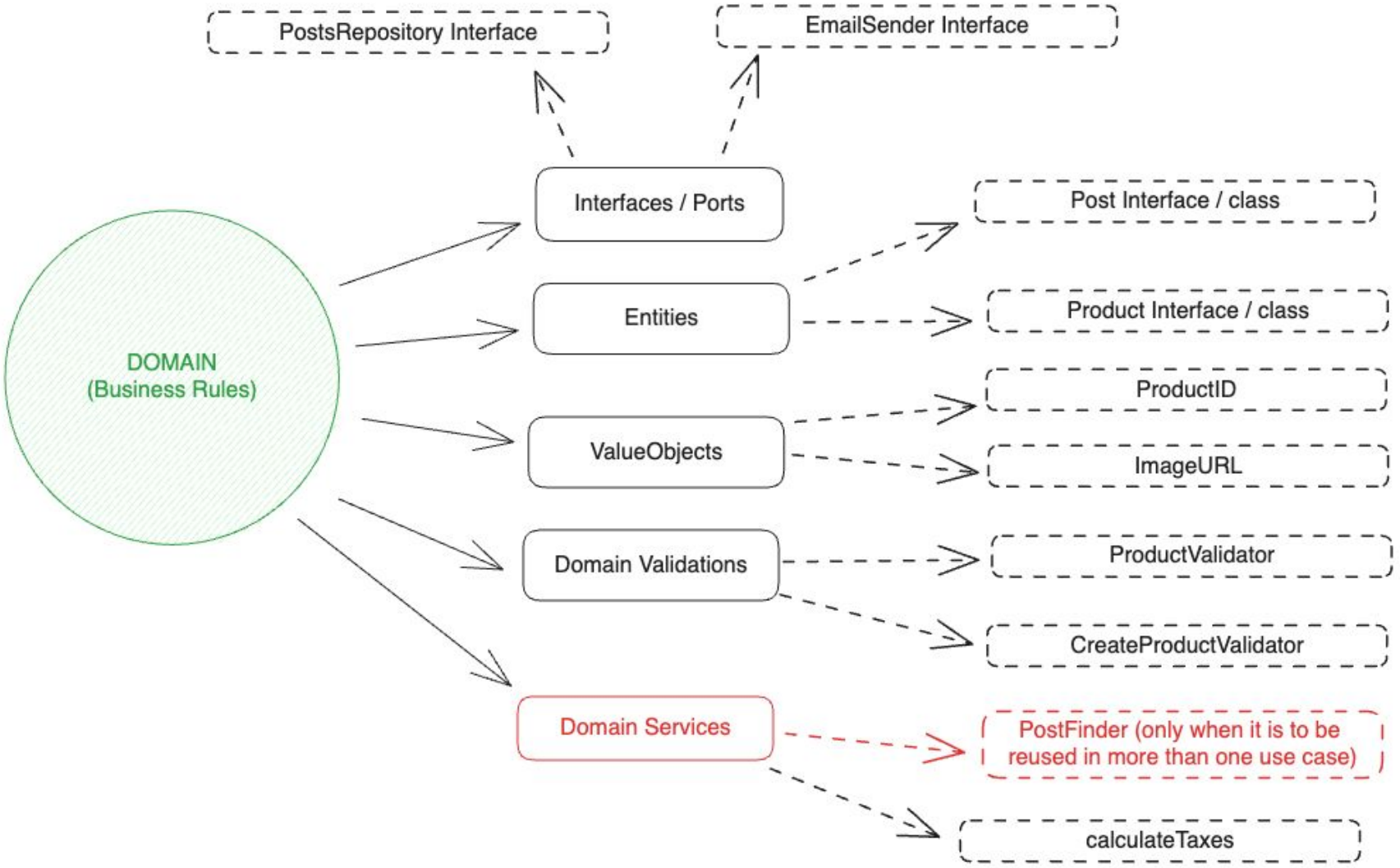


Alistair Cockburn

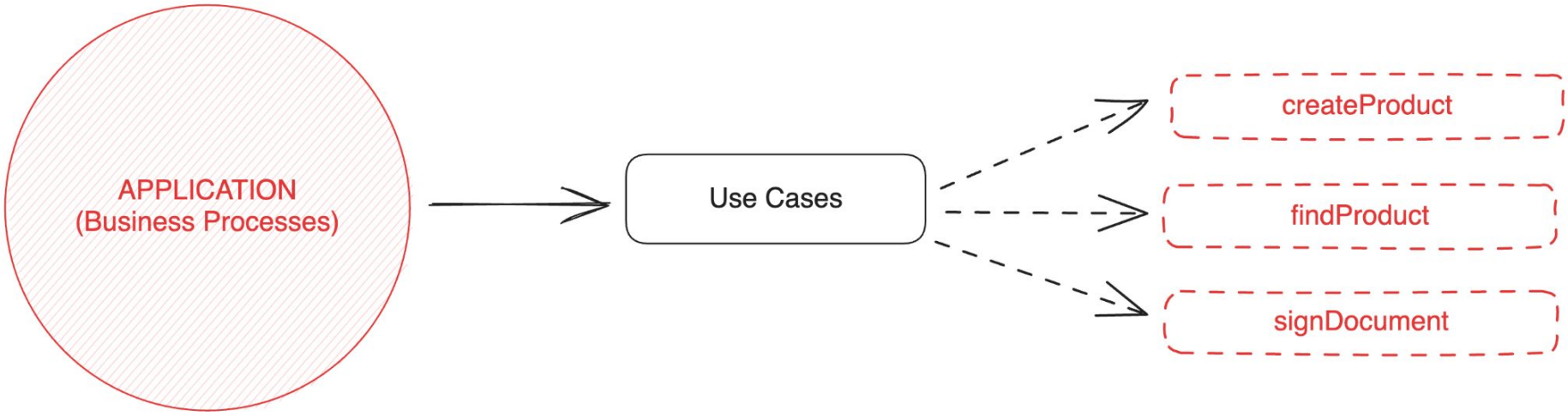
Hexagonal Architecture (Ports & Adapters)



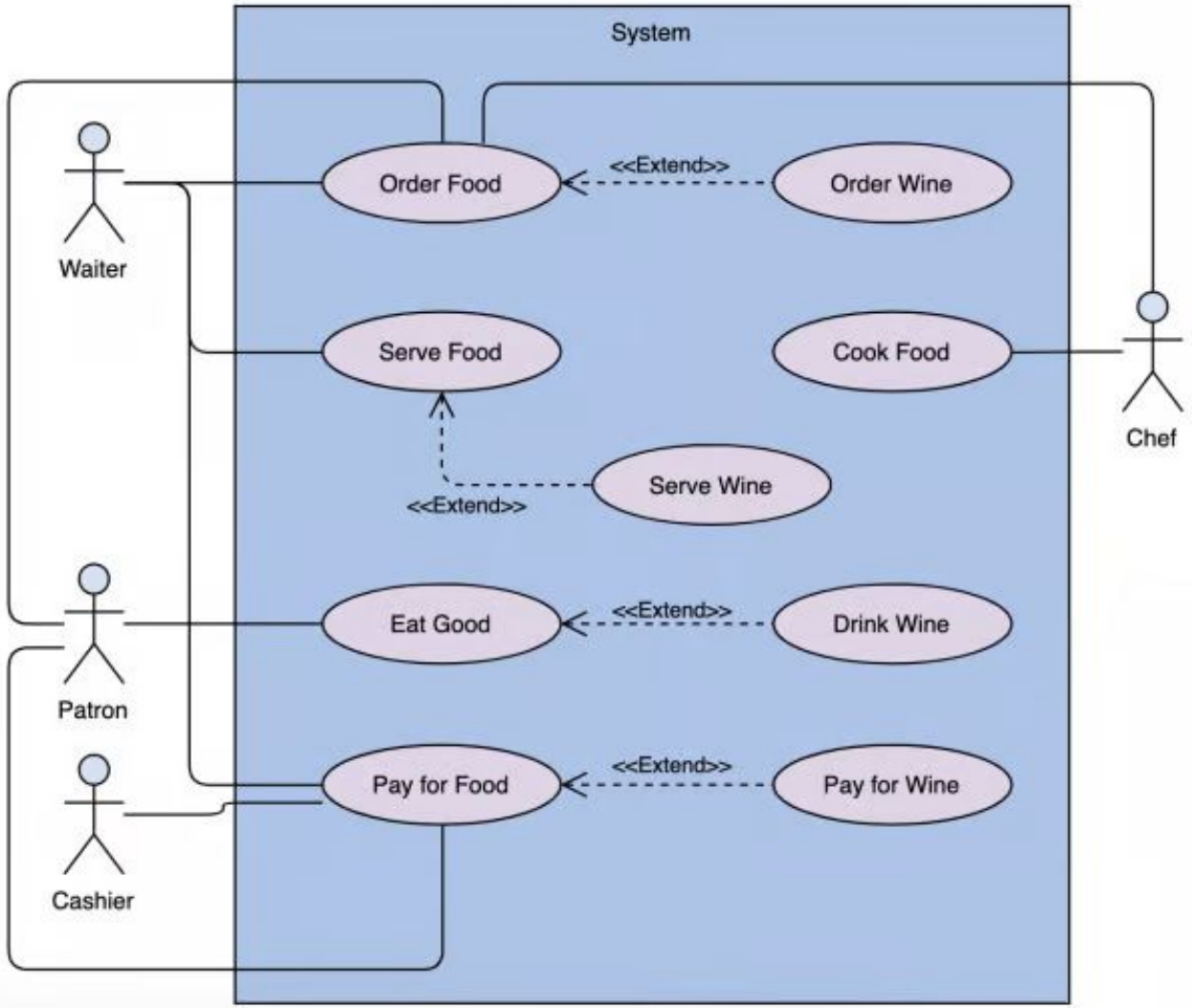
Domain Layer



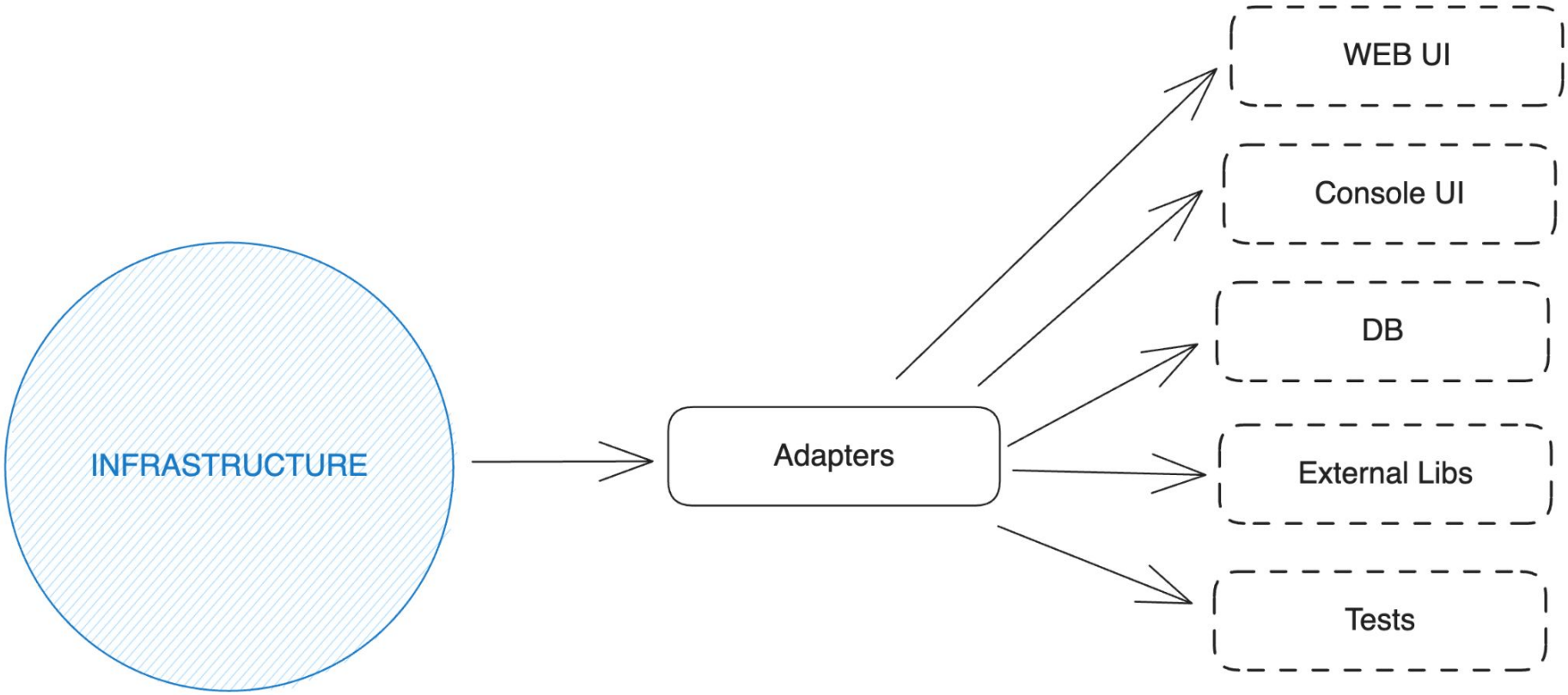
Application Layer



Use Cases


System
Waiter
Order Food
<<Extend>>
Order Wine
Serve Food
Cook Food
Chef
<<Extend>>
Serve Wine
Patron
Eat Good
<<Extend>>
Drink Wine
Cashier
Pay for Food
<<Extend>>
Pay for Wine

Infrastructure Layer



Ports & adapters

Use Case Example

```
export function createUser(user: User) {
  return db.collection('users').add(user);
}

export function findUser(userId: string) {
  return db.collection('users').doc(userId).get();
}

export function updateUser(userId: string, user: User) {
  return db.collection('users').doc(userId).update(user);
}

export function deleteUser(userId: string) {
  return db.collection('users').doc(userId).delete();
}
```

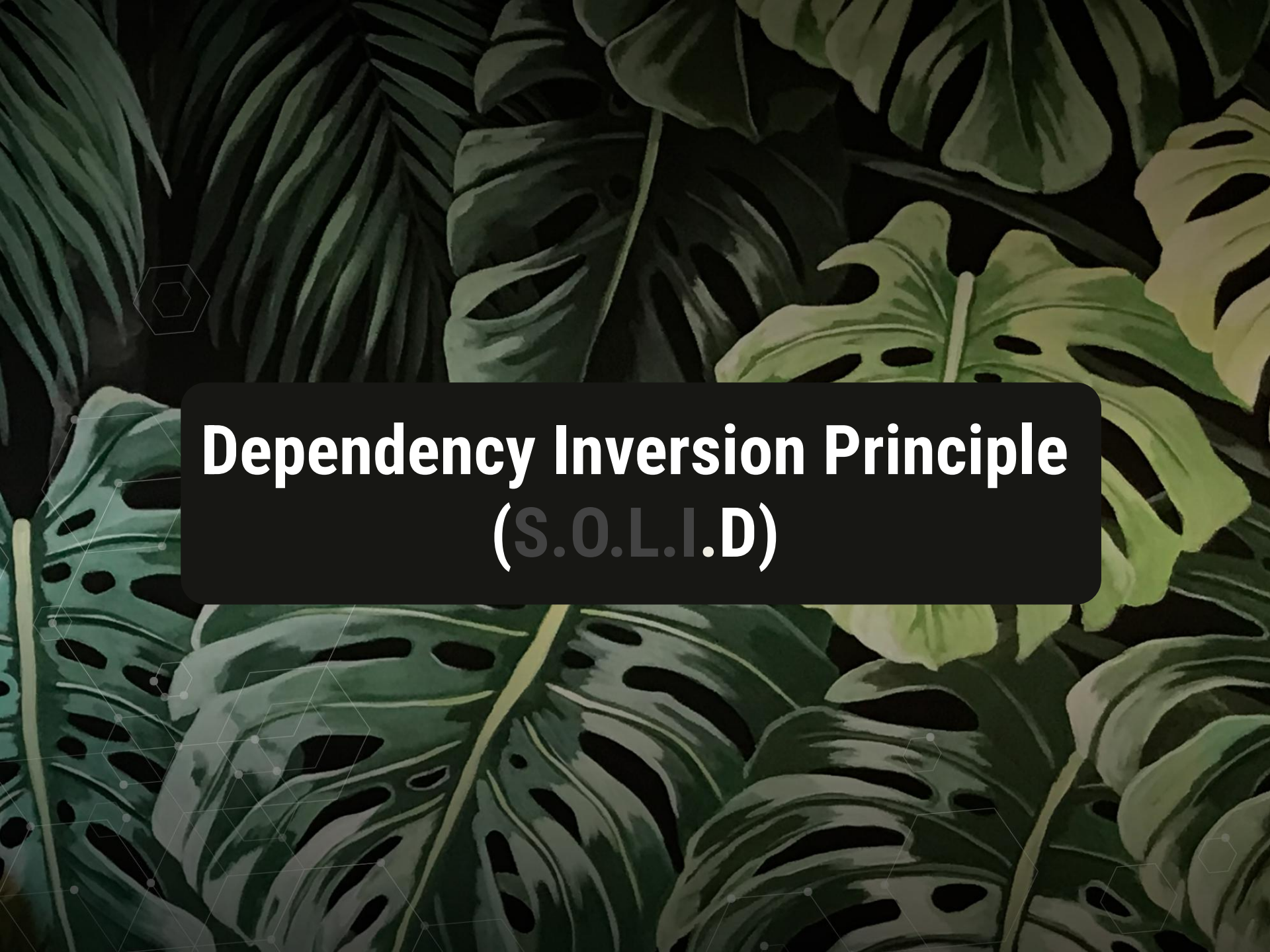
```ts
export async function createUser(user: User) {
  const response = await fetch('https://myapp.com/users', {
    method: 'POST',
    body: JSON.stringify(user),
  });
  return response.json();
}

export async function findUser(userId: string) {
  const response = await fetch(`https://myapp.com/users/${userId}`);
  return response.json();
}

export async function updateUser(userId: string, user: User) {
  const response = await fetch(`https://myapp.com/users/${userId}`, {
    method: 'PUT',
    body: JSON.stringify(user),
  });
  return response.json();
}

export async function deleteUser(userId: string) {
  const response = await fetch(`https://myapp.com/users/${userId}`, {
    method: 'DELETE',
  });
  return response.json();
}
```

Testing?

Repository Pattern

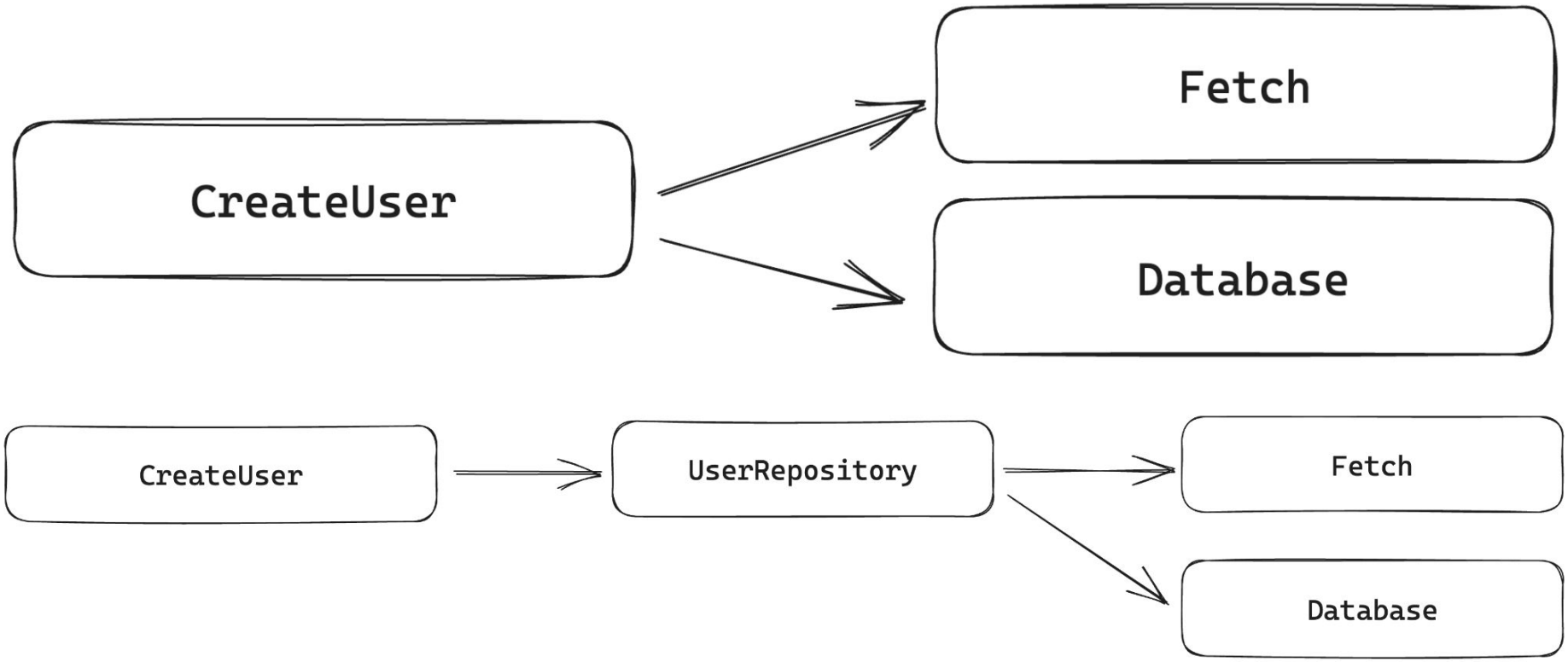
```
class UserRepository {
  async save(user: User): Promise<void> {
    const response = await fetch('https://myapp.com/users', {
      method: 'POST',
      body: JSON.stringify(user),
    });
    return response.json();
  }
....
}

function createUser(user: User) {
  const repository = new UserRepository();
  return repository.save(user);
}
```

Testing?


Dependency Inversion Principle
(S.O.L.I.D)

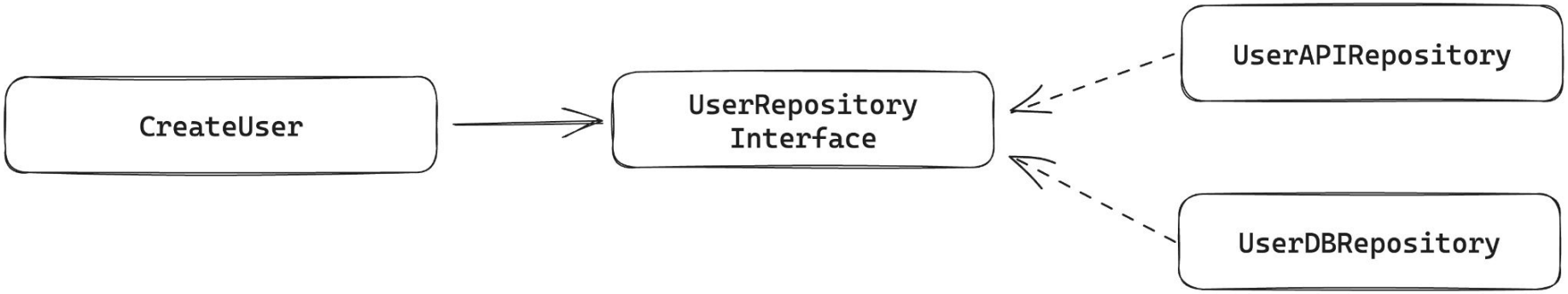
Dependency inversion principle

High-level modules should not depend on low-level modules. Both should depend on abstractions.



Dependency inversion principle

Abstractions should not depend on details. Details should depend on abstractions.



Dependency inversion principle

```ts
interface UserRepository {
  save(user: User): Promise<void>;
}

class UserAPIRepository implements UserRepository {
  async save(user: User): Promise<void> {
    const response = await fetch('https://myapp.com/users', {
      method: 'POST',
      body: JSON.stringify(user),
    });
    return response.json();
  }
  // ...
}

function createUser(user: User) {
  const repository = new UserAPIRepository();
  return repository.save(user);
}
```

Coupling?

Testing?

Dependency injection

```
function createUser(user: User, repository: UserRepository): Promise<void>
```

@param `user` — a user object

@param `repository` — a repository that implements the UserRepository interface

@returns — a promise that resolves to the created user

@example

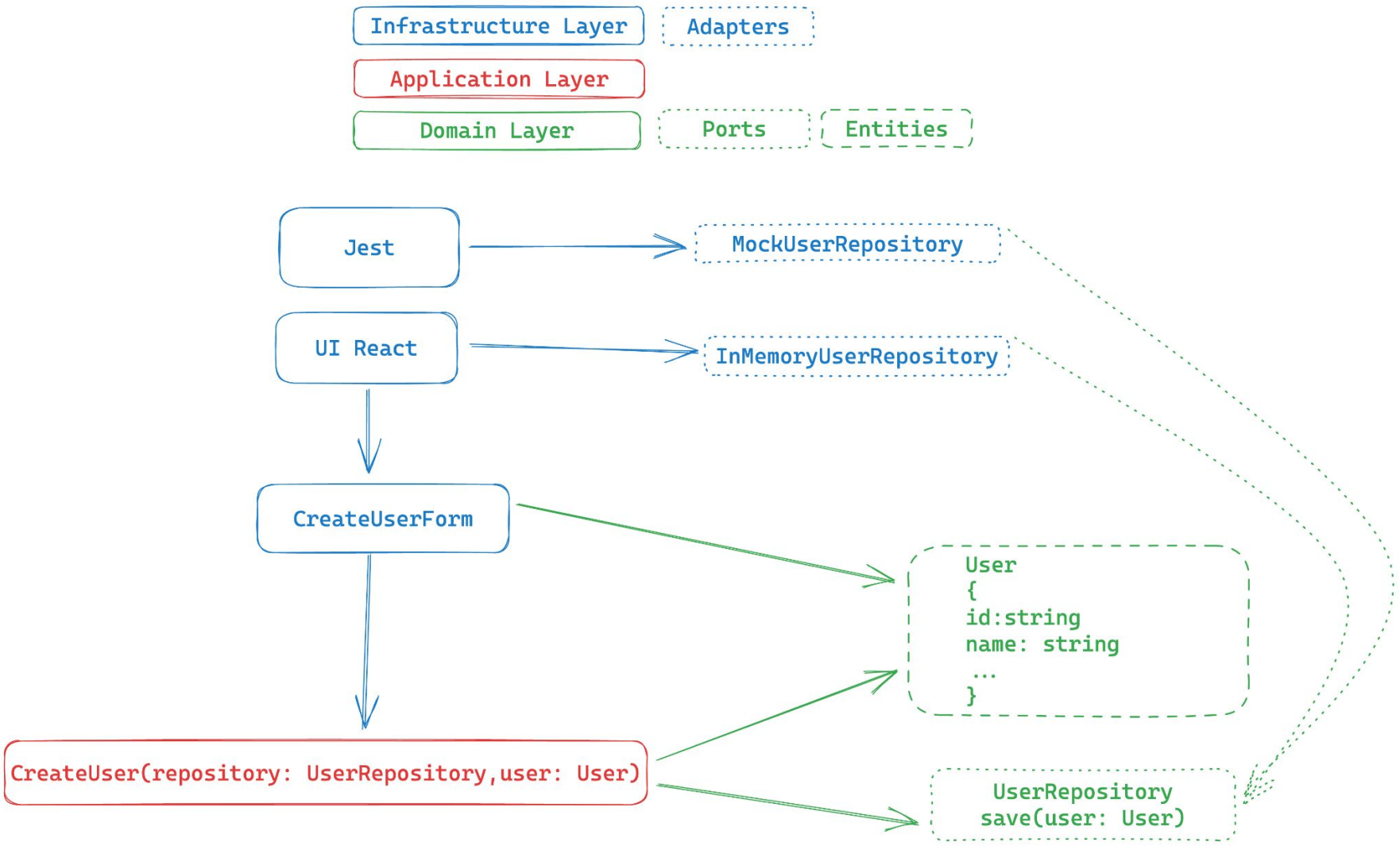
```
const user = await createUser({ name: 'John', email: 'xx@xx.com', age: 30 }, new
UserAPIRepository());
console.log(user);
// { name: 'John', email: 'xx@xx', age: 30 }
```

@example

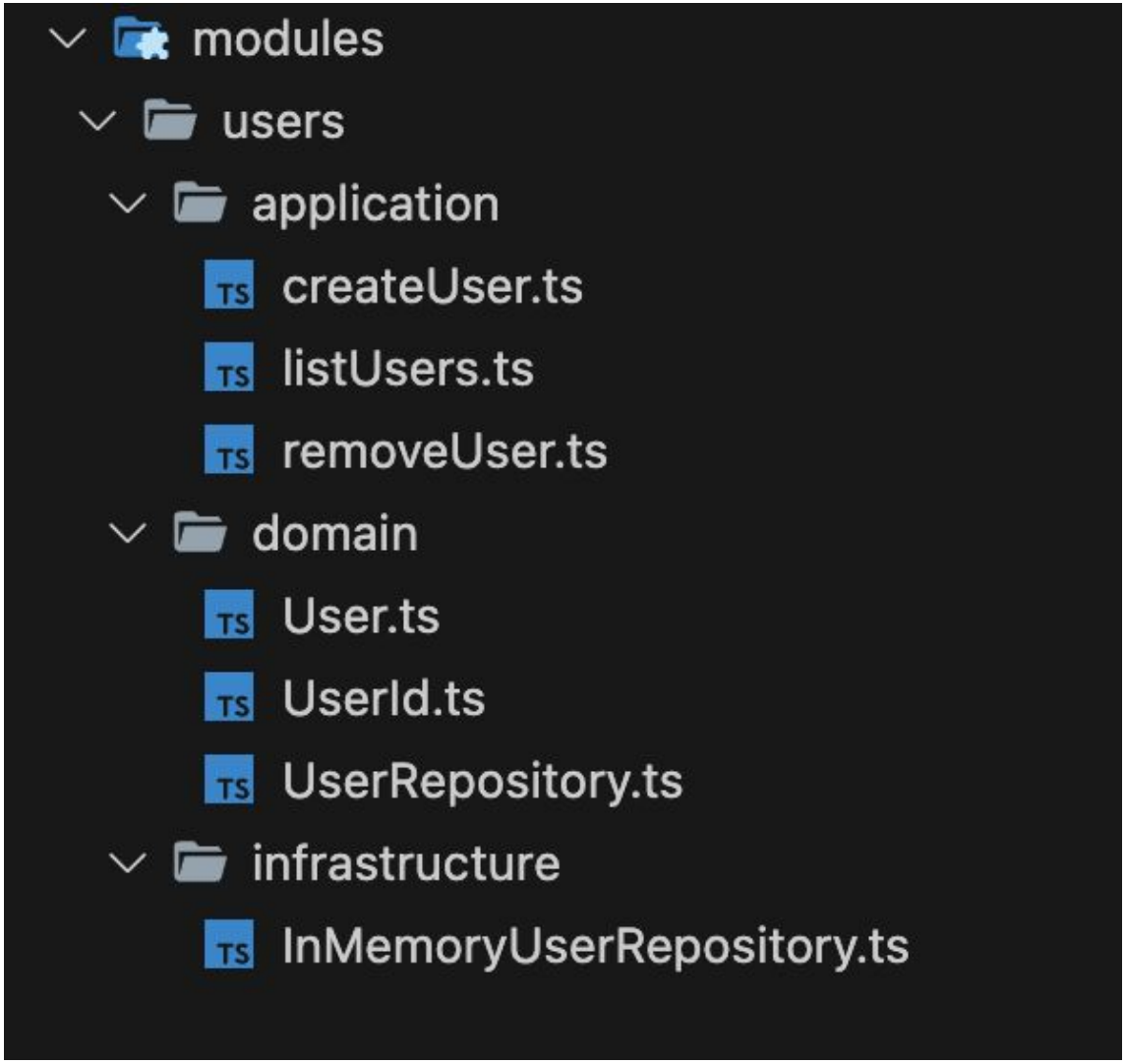
```
const user = await createUser({ name: 'John', email: 'xx@xx.com', age: 30 }, new
UserDBRepository());
```

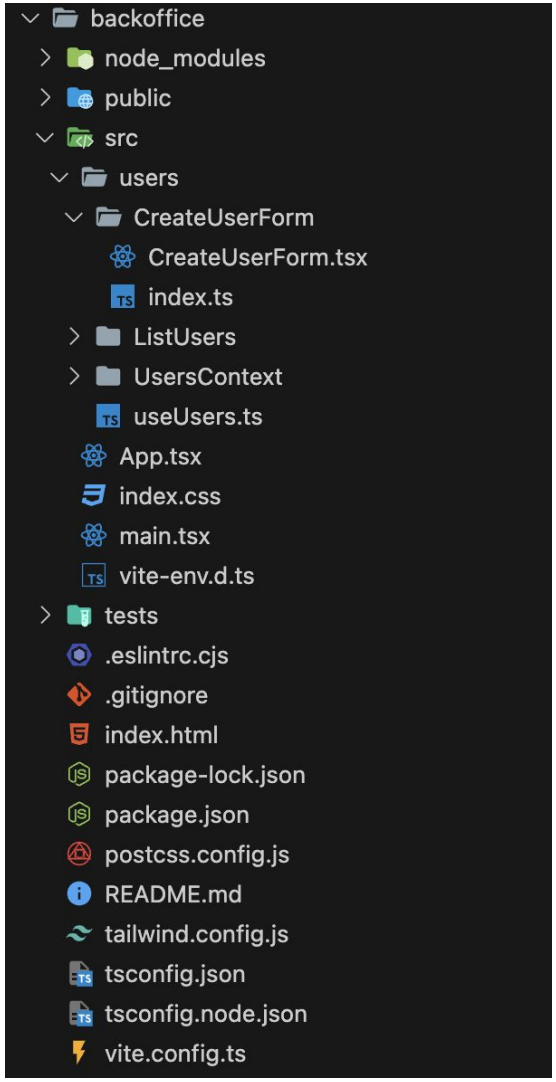
```
  return repository.save(user);
}
```

Workflow


Infrastructure Layer
Adapters
Application Layer
Domain Layer
Ports
Entities
Jest
MockUserRepository
UI React
InMemoryUserRepository
CreateUserForm
User
{
id:string
name: string
...
}
CreateUser(repository: UserRepository,user: User)
UserRepository
save(user: User)

Folder structure





Code

Workflow > modules > users > application > createUser.ts > ...

```ts
import { type User } from "../domain/User";
import { type UserRepository } from "../domain/UserRepository";

export function createUser(repository: UserRepository, user: User) {
  return repository.save(user);
}
```

```ts
export interface User {
  id: string;
  name: string;
  email: string;
  createdAt: Date;
  updatedAt: Date;
}
```

```ts
export interface UserRepository {
  save(user: User): Promise<void>;
  list(): Promise<User[]>;
  find(id: string): Promise<User | null>;
  delete(id: string): Promise<void>;
}
```

Code

```ts
export default class InMemoryUserRepository implements UserRepository {
  private users: User[] = [];

  async save(user: User): Promise<void> {
    await new Promise(resolve => setTimeout(resolve, 3000));
    this.users.push(user);
  }

  async find(id: string): Promise<User> {
    const user = this.users.find(user => user.id === id);
    if (!user) {
      throw new Error('User not found');
    }
    return user;
  }

  async list(): Promise<User[]> {
    return this.users;
  }

  async delete(id: string): Promise<void> {
    const user = this.users.find(user => user.id === id);
    if (!user) {
      throw new Error('User not found');
    }
    const index = this.users.indexOf(user);
    this.users.splice(index, 1);
  }
}
```

Code

```tsx
import CreateUserForm from "./users/CreateUserForm";
import { UserRepositoryContext } from "./users/UserRepositoryContext";
import InMemoryUserRepository from '../../../modules/users/infrastructure/InMemoryUserRepository';
import { UserRepository } from "../../../modules/users/domain/UserRepository";

const userRepository: UserRepository = new InMemoryUserRepository();

function App() {
  return (
    <UserRepositoryContext.Provider value={{
      repository: userRepository,
    }}>
      <CreateUserForm />
    </UserRepositoryContext.Provider>
  )
}

export default App;
```

Testing

```
describe('Create User', () => {
  it('should create a user', 
    const users: User[] = [];

    const repository: UserRep
      save: vi.fn(async (user
        users.push(user);
      }),
      delete: vi.fn(),
      find: vi.fn(),
      list: vi.fn(),
    };

    const mockUser: User = {
      id: '123',
      name: 'John Doe',
      email: 'johndoe@gmail.c
      createdAt: new Date(),
      updatedAt: new Date(),
    };

    await createUser(reposito

    expect(users).toEqual([mo
  });
});
```

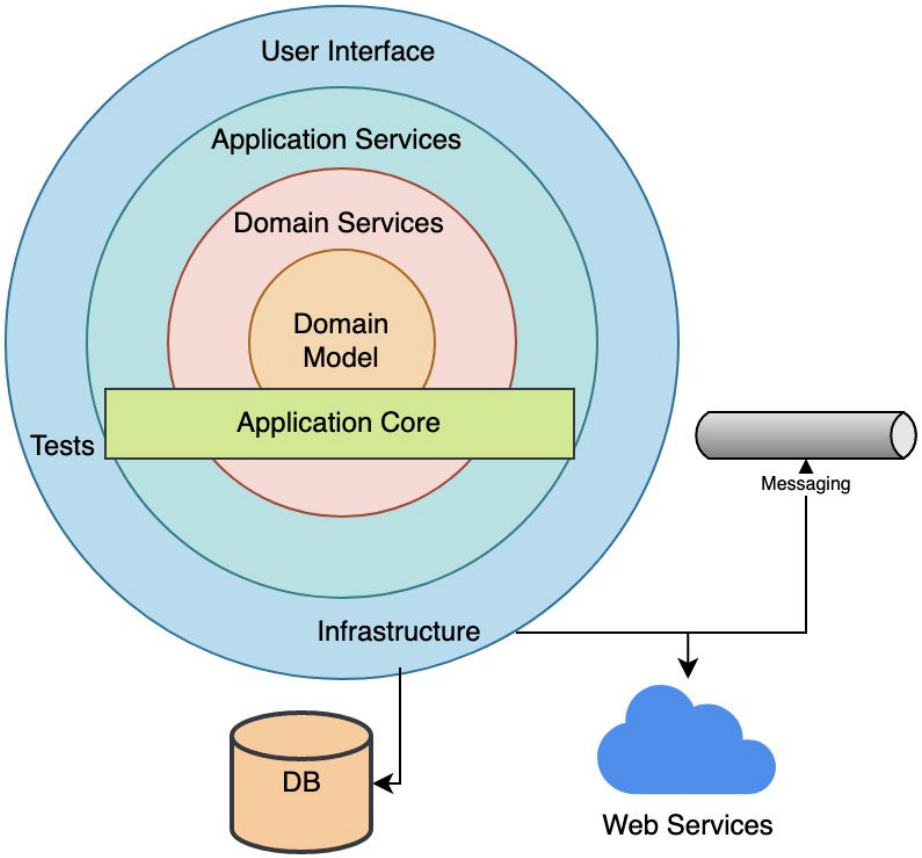
```
it('should create user when form is submitted', async () => {
  const users: User[] = [];
  const saveUser = vi.fn().mockImplementation((user) => {
    users.push(user);
  });
  renderWithProviders(<CreateUserForm />, { ...DEFAULT_REPOSITORY, save: saveUser });
  const name = 'John Doe';
  const email = 'johndoe@gmail.com';
  const confirmEmail = 'johndoe@gmail.com';
  const nameInput = screen.getByLabelText('Name');
  const emailInput = screen.getByLabelText('Email');
  const confirmEmailInput = screen.getByLabelText('Confirm Email');
  const submitButton = screen.getByRole('button');

  await userEvent.type(nameInput, name);
  await userEvent.type(emailInput, email);
  await userEvent.type(confirmEmailInput, confirmEmail);

  await userEvent.click(submitButton);

  await waitFor(() => expect(saveUser).toBeCalledTimes(1));
  await waitFor(() => expect(users).toHaveLength(1));
  await waitFor(() => expect(users[0]).toEqual({
    id: expect.any(String),
    name,
    email,
    createdAt: expect.any(Date),
    updatedAt: expect.any(Date)
  }));
});
```


Timeline
After Hexagonal Architecture

Timeline
After Hexagonal Architecture

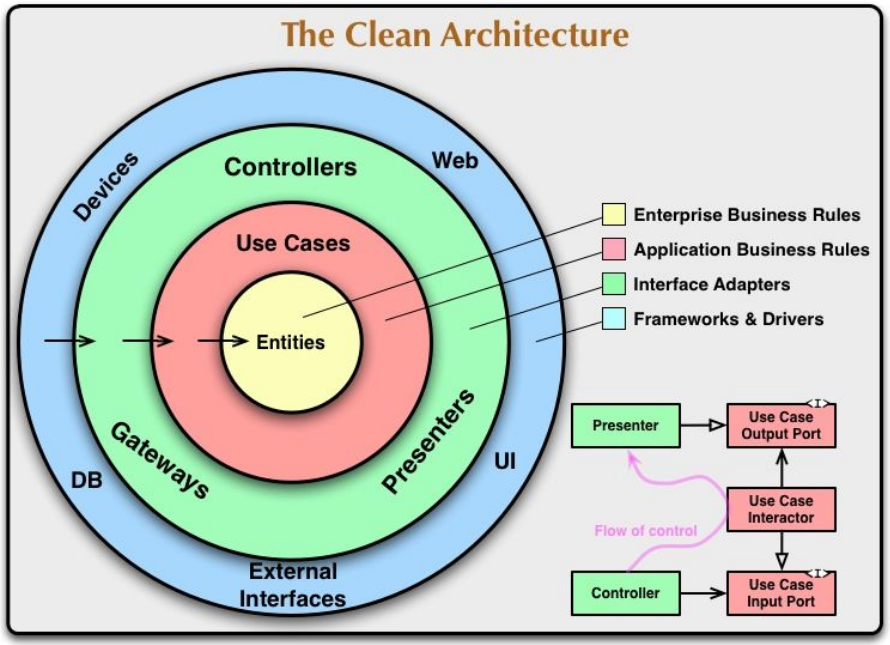
Onion Architecture (2008)

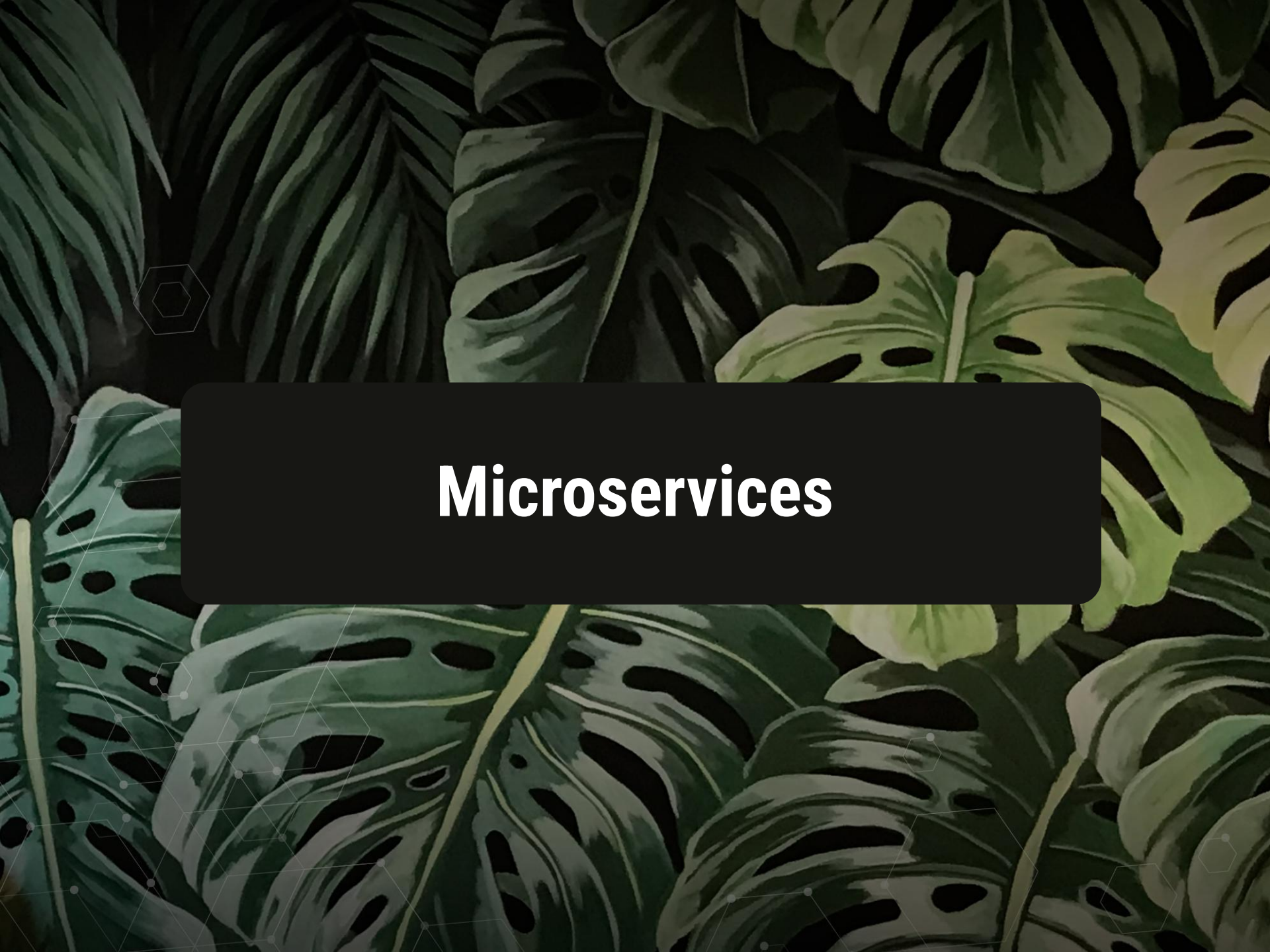
- Jeffrey Palermo



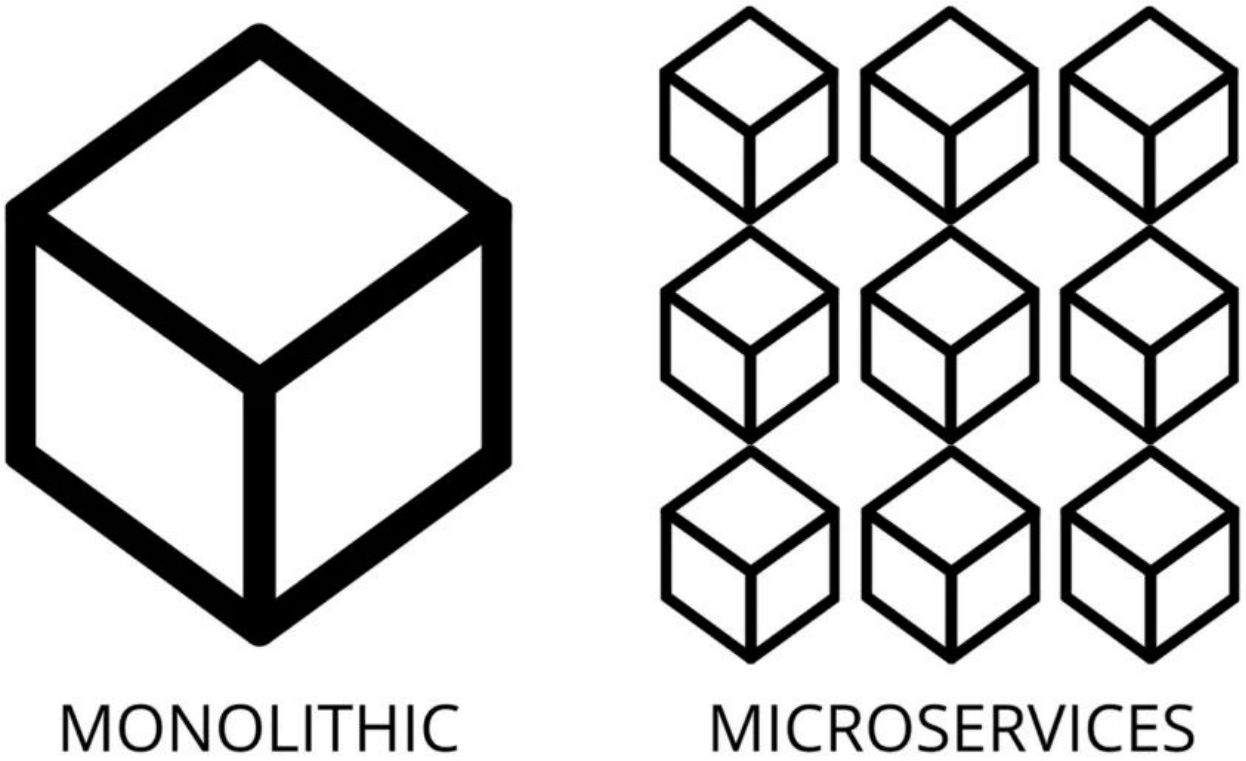
Clean Architecture (2012)

- Uncle Bob / Robert C. Martin
- Clean Architecture: A Craftsman's Guide to Software Structure and Design

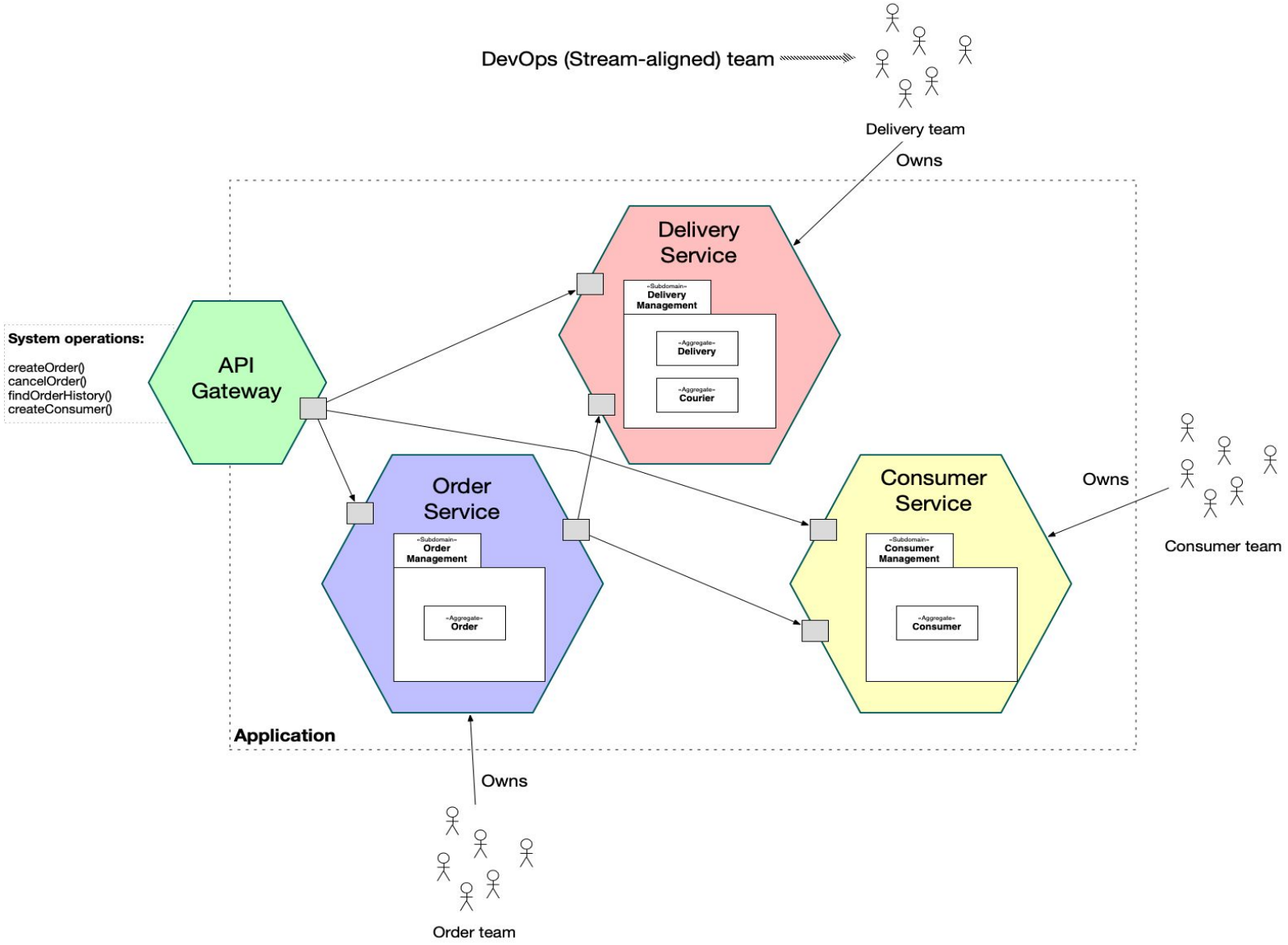



Microservices

Microservices

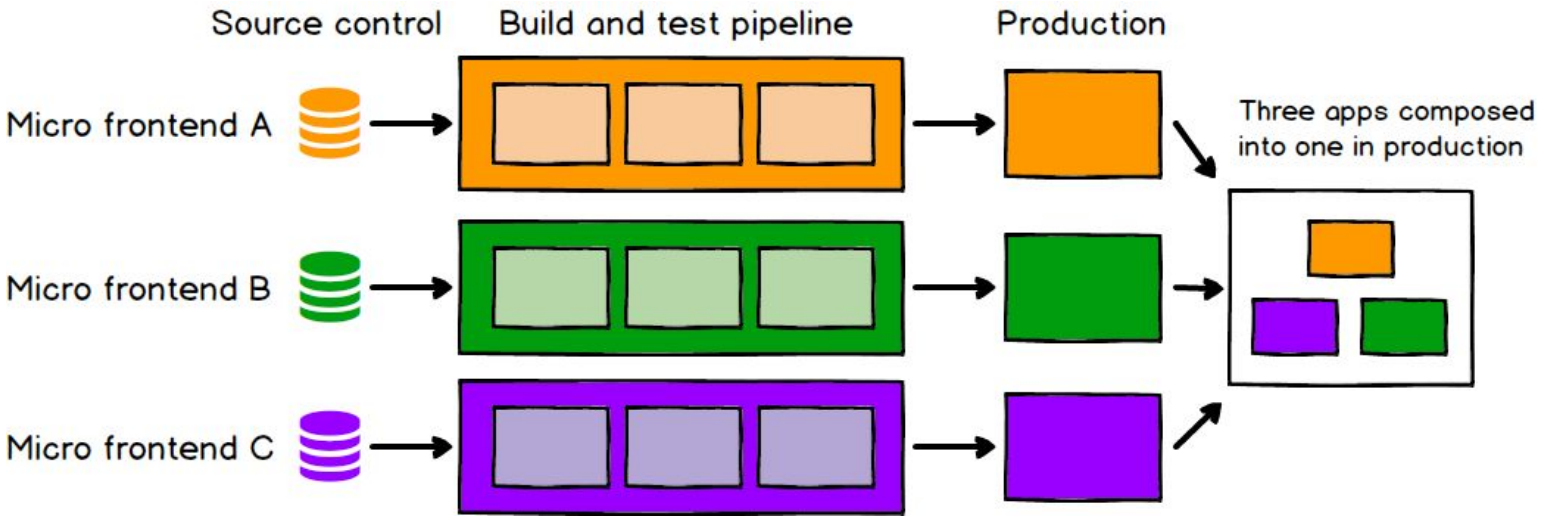


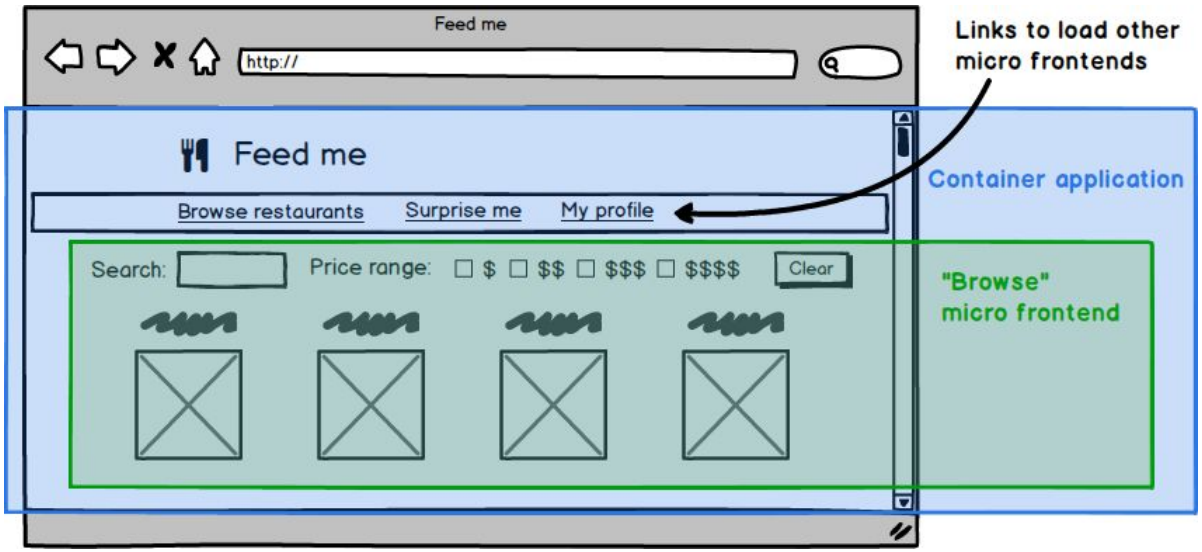
Microservices




Micro Frontends

Micro Frontends





Conclusions


Q&A


Thank You!

References

- https://www.amazon.com/-/es/Frank-Buschmann/dp/0471958697
- https://alistair.cockburn.us/hexagonal-architecture/
- https://martinfowler.com/architecture/
- Clean Architecture: A Craftsman's Guide to Software Structure and Design (Uncle Bob / Robert C. Martin)
- https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/
- https://medium.com/ssense-tech/dependency-injection-vs-dependency-inversion-vs-inversion-of-control-lets-set-the-record-straight-5dc818dc32d1
- https://martinfowler.com/articles/micro-frontends.html
- https://microservices.io/patterns/index.html