

Software Design

World, Rules & Relationship

Agenda

- Object Oriented rules and tools
- What I am doing at work?
- What can I use of this course?

Object Oriented rules and tools

Object Oriented rules 1

Calistenics and Code smells

- Only one level of indentation per method 🙌 Long Method
- Don't use the ELSE keyword 🙌 Long Method / Duplicated Code
- Wrap all primitives and strings 🙌 Primitive Obsession
- First class collections 🙌 Divergent Change / Large Class
- One dot per line 🙌 Message Chains
- Keep all entities small 🙌 Large Class / Long Method / Long Parameter List
- No classes with more than two instance variables 🙌 Large Class
- No getters / setters / properties 🙌 Feature Envy
- All classes must have state, no static methods, no utility classes 🙌 Lazy Class / Middle man / Feature envy

Object Oriented rules 2

SOLID principles

- Single Responsibility SRP
- Open Closed OCP
- Liskov Substitution LSP
- Interface Segregation ISP
- Dependency Inversion DIP

Object Oriented rules 3

- 1. EverythingIsAnObject.
- 2. Objects communicate by sending and receiving messages (in terms of objects).
- 3. Objects have their own memory (in terms of objects).
- 4. Every object is an instance of a class (which must be an object).
- 5. The class holds the shared behavior for its instances (in the form of objects in a program list)
- 6. To eval a program list, control is passed to the first object and the remainder is treated as its message.

I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea. The big idea is "messaging." (2)

(1) <https://wiki.c2.com/?AlanKaysDefinitionOfObjectOriented>

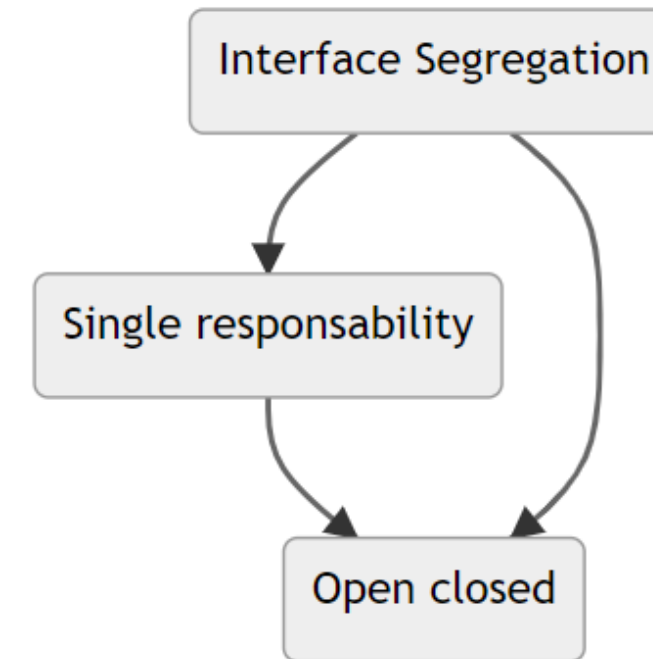
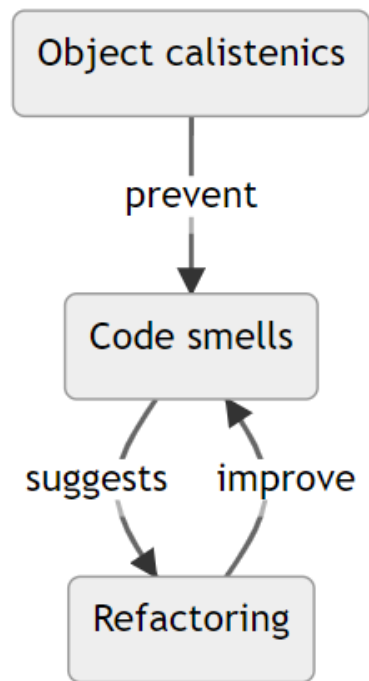
(2) <http://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html>

Object Oriented tools

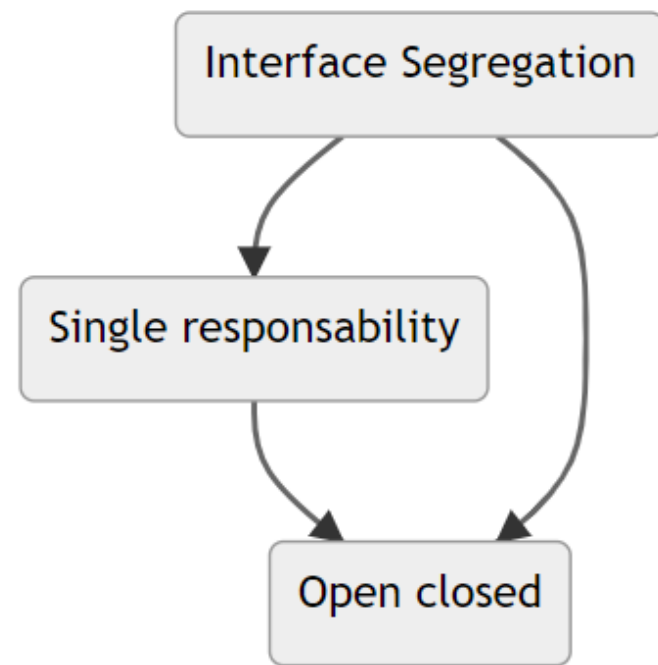
Refactoring

 Fix code smell by procedure (sometimes provided by IDE)

Rules & tools relationships






Rules



Code

```
public class Car {  
    public int CurrentMileage(){ ... }  
    public void TravelTo(Location location){ ... }  
    public void Save(){ ... }  
}
```

Rules

-  Single Responsibility
-  Open Close
-  Interface Segregation

Code

Behaviours attractor

```
public class Car {  
    public int CurrentMileage(){ ... }  
    public void TravelTo(Location location){ ... }  
    public void Save(){ ... }  
    public void SaveToFile(){ ... }  
    public void toJson(){ ... }  
}
```

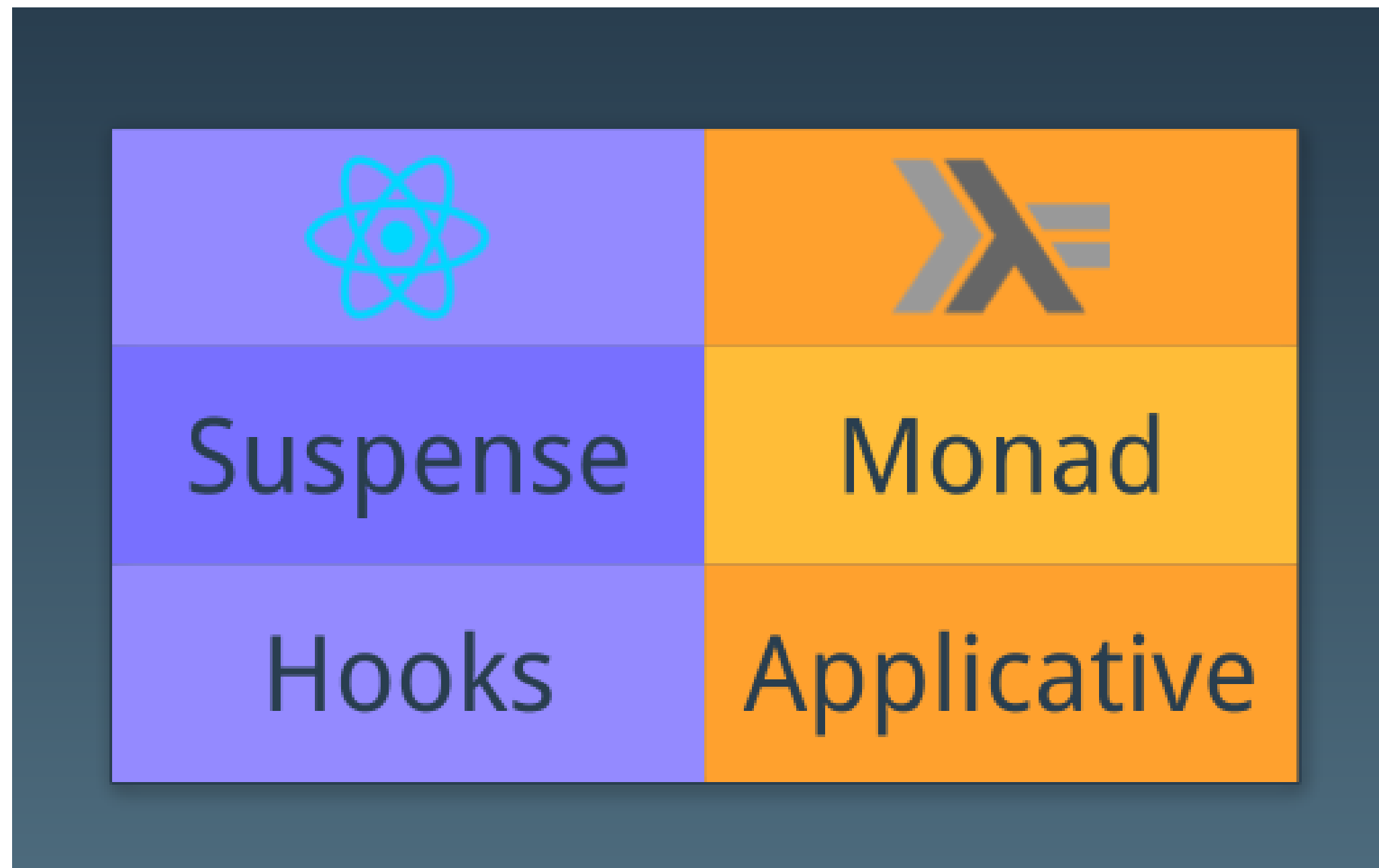
What am I doing at work?

In which world does my code belong to?

- Every world has own rules
- What am I doing?
 - Procedural code
 - Functional React Components
 - Hooks
 - Stores implements Funtor and Foldable
 - Event implements Profuntor and Functor
 - Combine stores and events
 - Lenses
 - Handling effects

Functional React Component

React Suspense is to a Monad as Hooks are to Applicative Notation



Lesson learned

- Language is not the paradigm
- Respect rules
- Compose
- Write declarative code

Object Oriented

```
function TicTacToe() {
  let player = Player.X;
  const board = Board();

  const getNextPlayer = () => {
    if (player === Player.O) {
      return Player.X;
    }

    return Player.O;
  };

  const play = (square) => {
    if (board.hasBeenPlayed(square)) {
      return;
    }

    board.play(square, player);
    player = getNextPlayer();
  };

  ...
}
```

Code

```
...

const getPlayer = () => {
  return player;
};

const getState = () => {
  const winner = board.getWinner();

  if (winner === Player.None) {
    return State.InProgress;
  }

  if (Player.X === winner) {
    return State.PlayerXWins;
  }

  return State.PlayerOWins;
};

return { getPlayer, play, getState };
}
```

Functional Programming Code

```
class IO {
  constructor(fn) {
    this.unsafePerformIO = fn;
  }

  [util.inspect.custom]() {
    return 'IO(?)';
  }

  // ——— Pointed IO
  static of(x) {
    return new IO(() => x);
  }

  // ——— Functor IO
  map(fn) {
    return new IO(compose(fn, this.unsafePerformIO));
  }

  ...
}
```

```
...

// ——— Applicative IO
ap(f) {
  return this.chain(fn => f.map(fn));
}

// ——— Monad IO
chain(fn) {
  return this.map(fn).join();
}

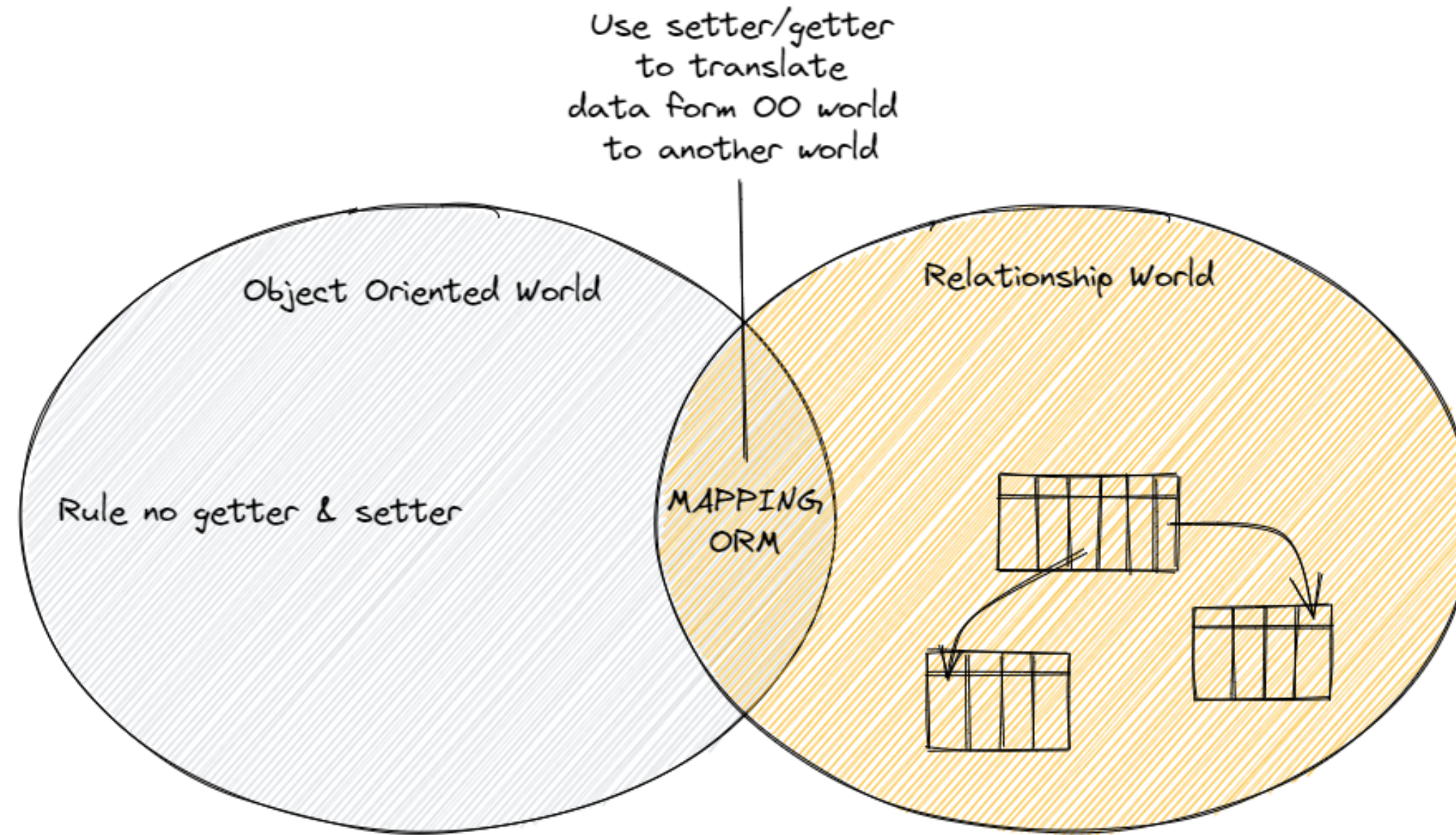
join() {
  return new IO(
    () => this.unsafePerformIO().unsafePerformIO()
  );
}
}
```


“Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”

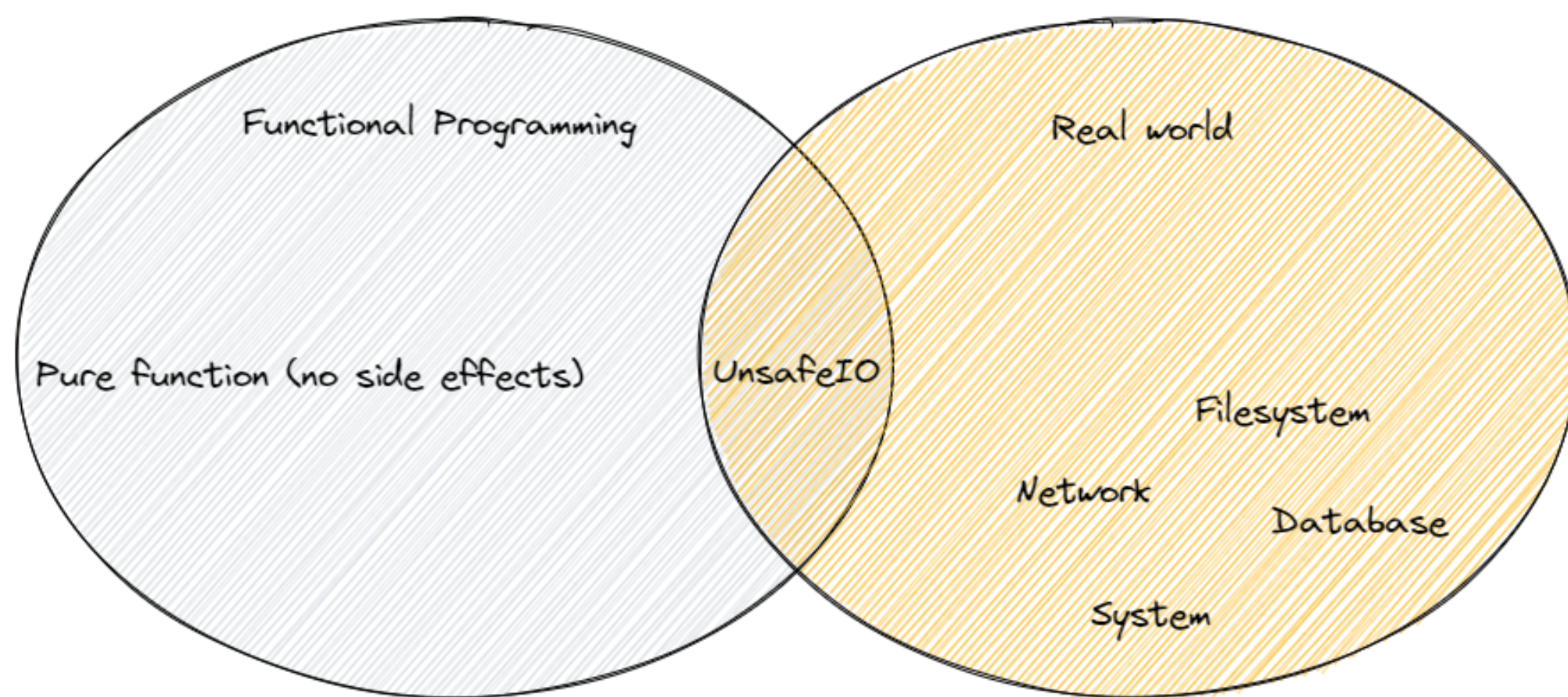
— Linus Torvalds

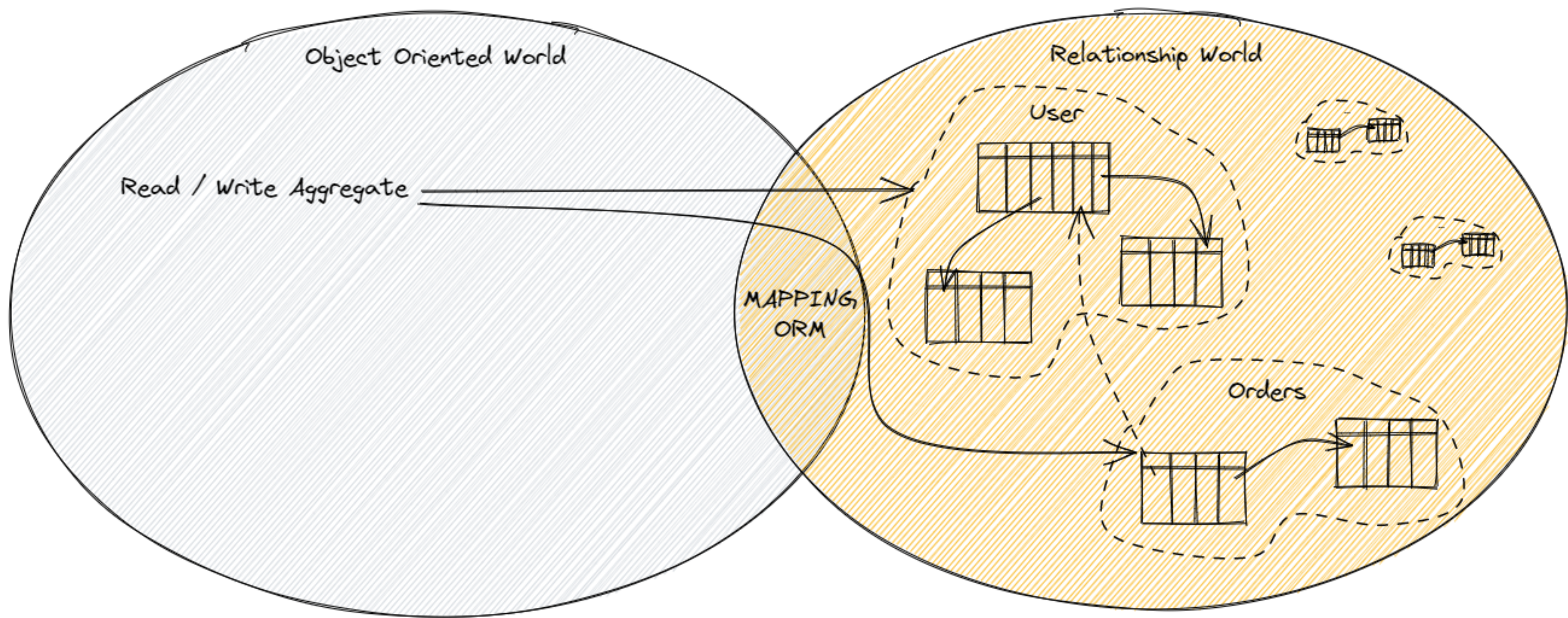
Across Boundaries

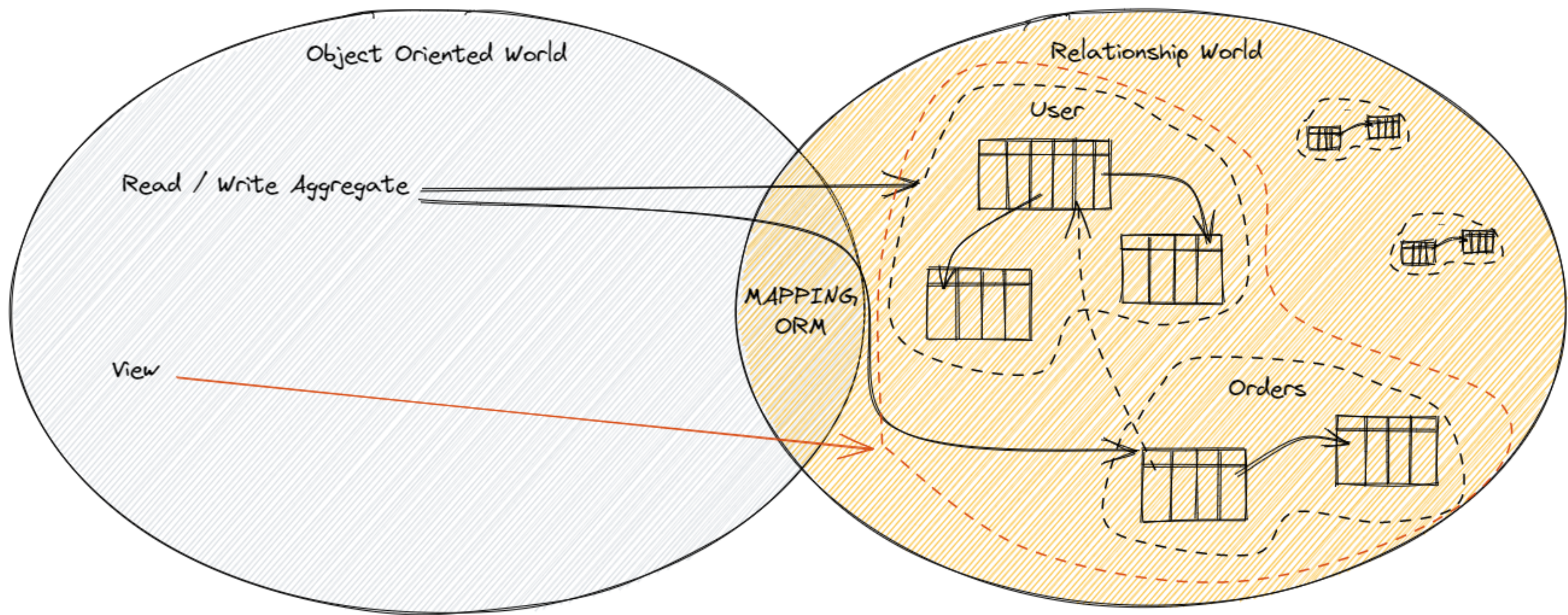
OOP Rules

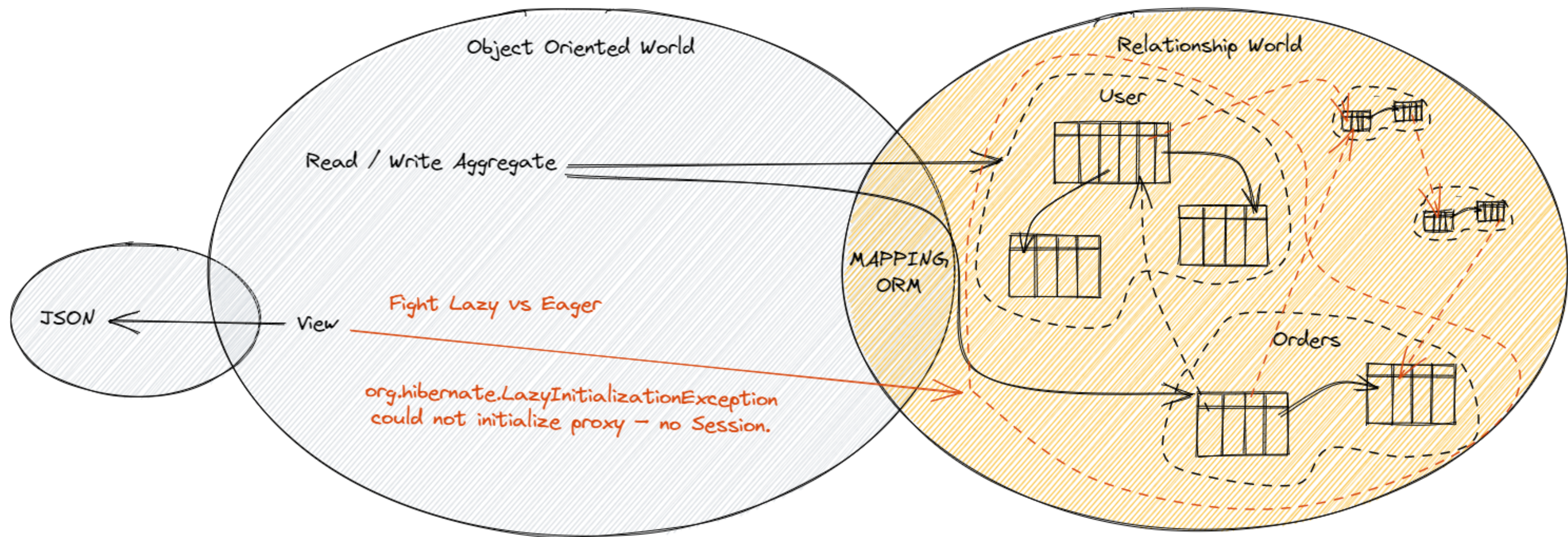


FP Rules









Questions?

Thank you

David Nussio, Software Engineer - EOC

 davidnussio

 @davidnussio