Functional Software Architecture

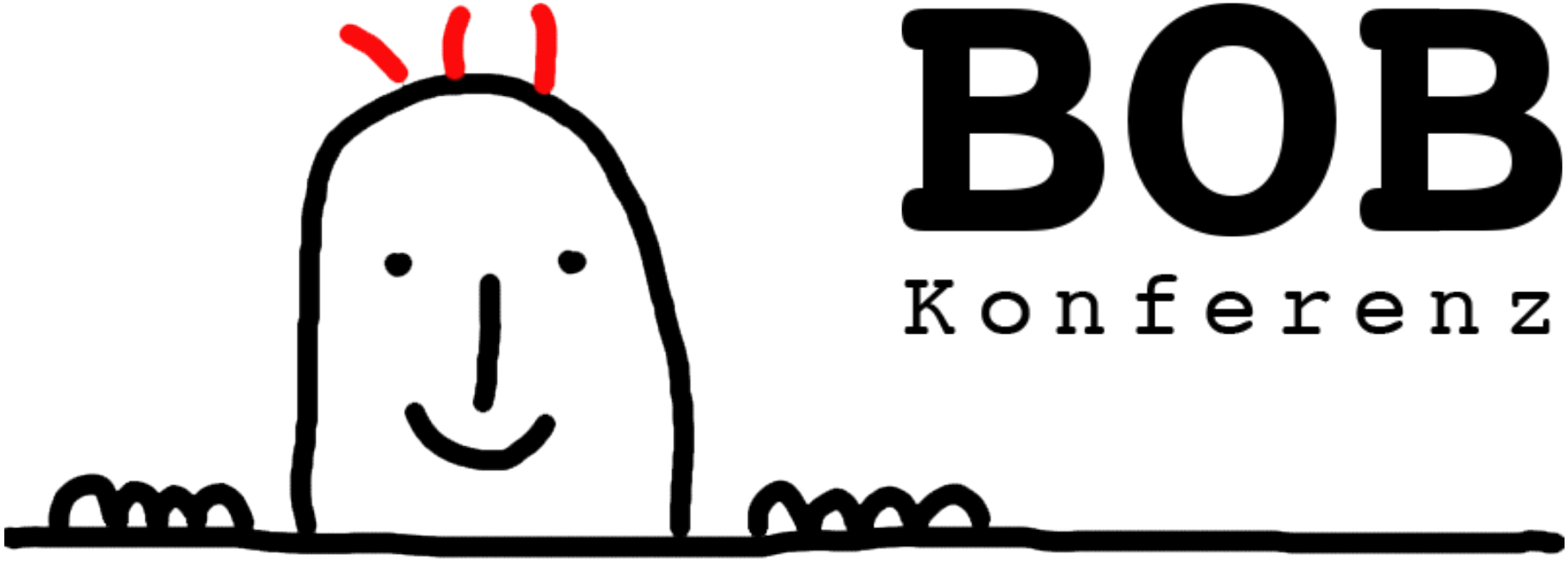
Michael Sperber

@sperbsen

- software project development
- in many fields
- Scala, Clojure, Erlang, Haskell, F#, OCaml
- training, coaching

www.active-group.de

funktionale-programmierung.de



Feb 26, 2021 – Berlin or online

Keynote: Jeremy Gibbons!

bobkonf.de

What is Software Architecture?

Organization of ...

- large systems
- long-lived systems

Background

Curriculum for

CPSA Certified Professional for Software Architecture®

– Advanced Level –

Module:
FUNAR

Functional Software Architecture

https://www.isaqb.org/certifications/advanced-level/?lang=en

Aspects of Software Architecture

- How do we split into parts?
- How do we implement the parts?

Organizing Parts of a Software System

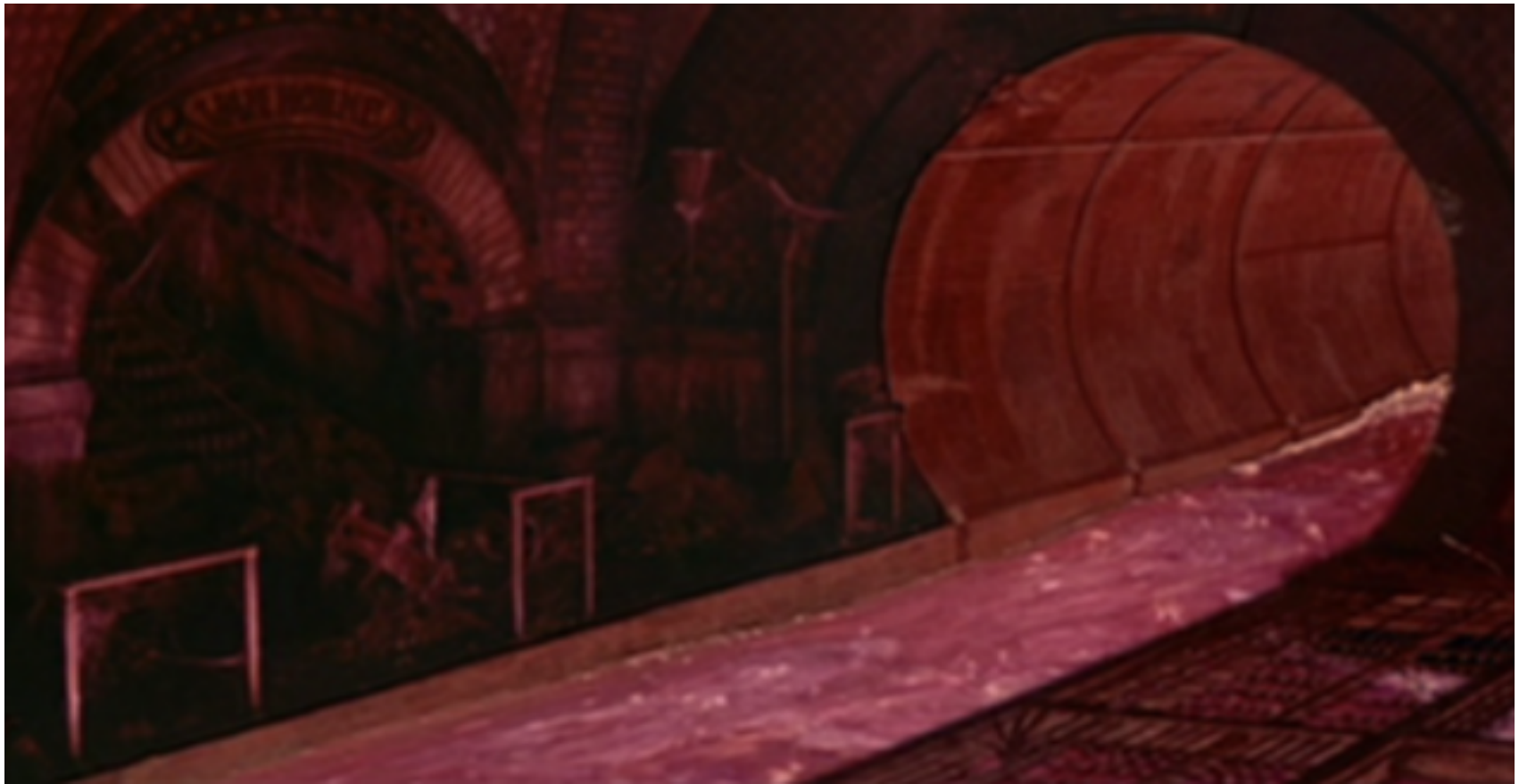
- create **domain models**
  - combinator models
  - embedded domain-specific languages
  - abstraction
  - algebra
  - category theory
- limit, control, and document **effects**

Requirements for Domain Models

- faithful
- expressive
- insightful

Why Control Effects?

- predictability
- avoid/document hidden dependencies



Functional Core, Imperative Shell

Hexagonal Architecture

How to Split a System into Parts

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content=
        "text/html; charset=us-ascii" />
        <script type="text/javascript">
            function reDo() {top.location.reload();}
            if (navigator.appName == 'Netscape') {top.onresize = reDo;}
            dom=document.getElementById;
        </script>
    </head>
    <body>
    </body>
</html>
```

```
function removeBackdrop() {
  this.$backdrop.remove()
  this.$backdrop = null
}

function escape() {
  var that = this
  if (this.isShown && this.options.keyboard) {
    $(document).on('keyup.dismiss.modal', function ( e ) {
      e.which == 27 && that.hide()
    })
  } else if (!this.isShown) {
    $(document).off('keyup.dismiss.modal')
  }
}
```

```
/// Font - Source Code Pro
/// Size - 11

class Program : Object
{
    static int _I = 1;

    /// <summary>
    /// The quick brown fox jumps over the lazy dog
    /// THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
    /// </summary>
    static void Main(string[] args)
    {
        Uri IllegalUri = new Uri("http://packmyboxwith/jugs.html?q=five-dozen&t=liquor");
        Regex OperatorRegex = new Regex(@"S#$", RegexOptions.IgnorePatternWhitespace);

        for (int O = 0; O < 123456789; O++)
        {
            _I += (O % 3) + ((O / 1) ^ 2) - 5;
            if (!OperatorRegex.IsMatch(IllegalUri.ToString()))
            {
                Console.WriteLine(IllegalUri);
            }
        }
    }
}
```

How to Split a System into Parts

WIKIPEDIA
The Free Encyclopedia

Article | Talk | Read | Edit | View history | Search Wikipedia

Conway's law

From Wikipedia, the free encyclopedia

Conway's law is an adage stating that organizations design systems that mirror their own communication structure. It is named after computer programmer Melvin Conway, who introduced the idea in 1967.[1] His original wording was:

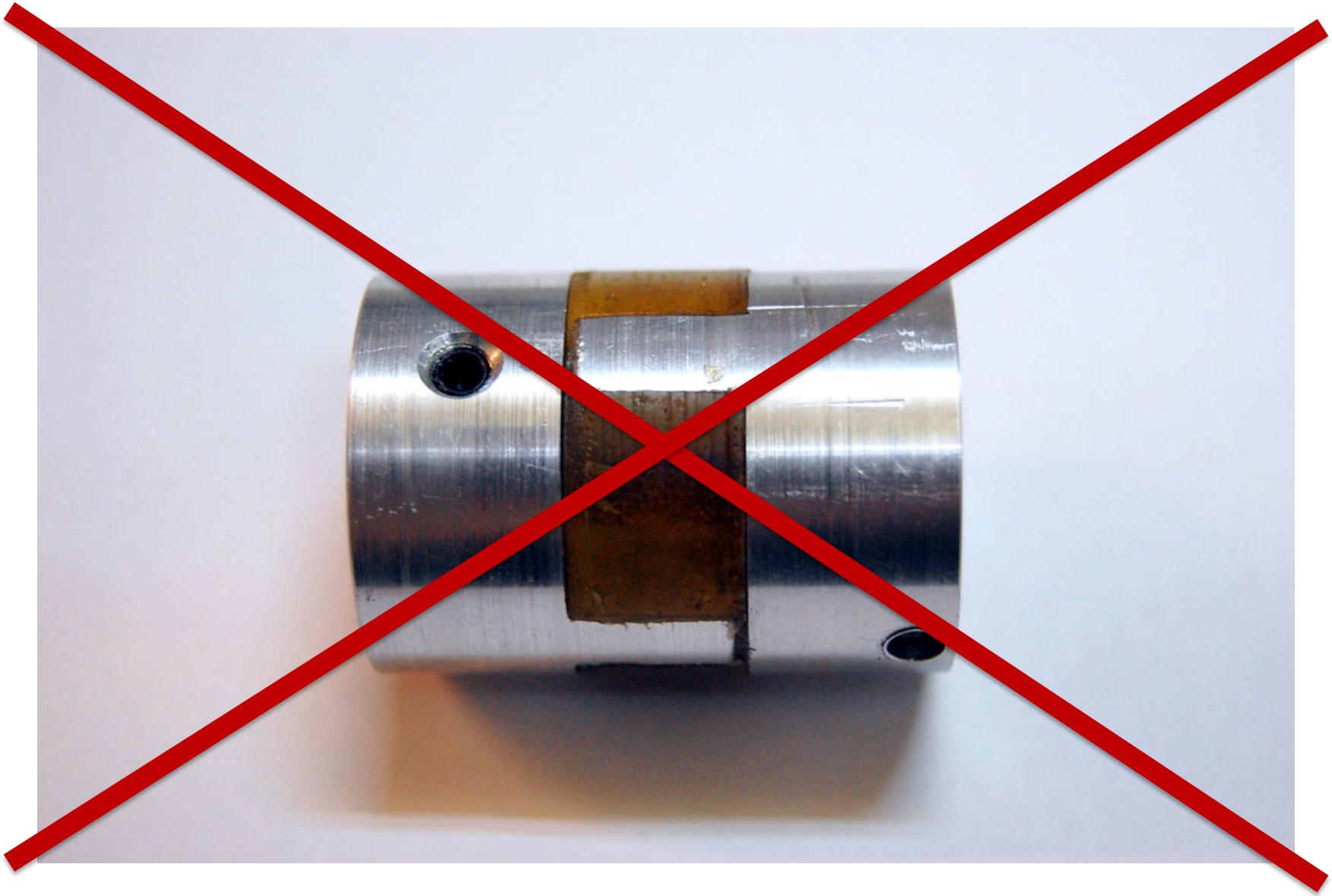
> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.[2][3]
>
> — Melvin E. Conway

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute

Help

Connect Parts of a Software System



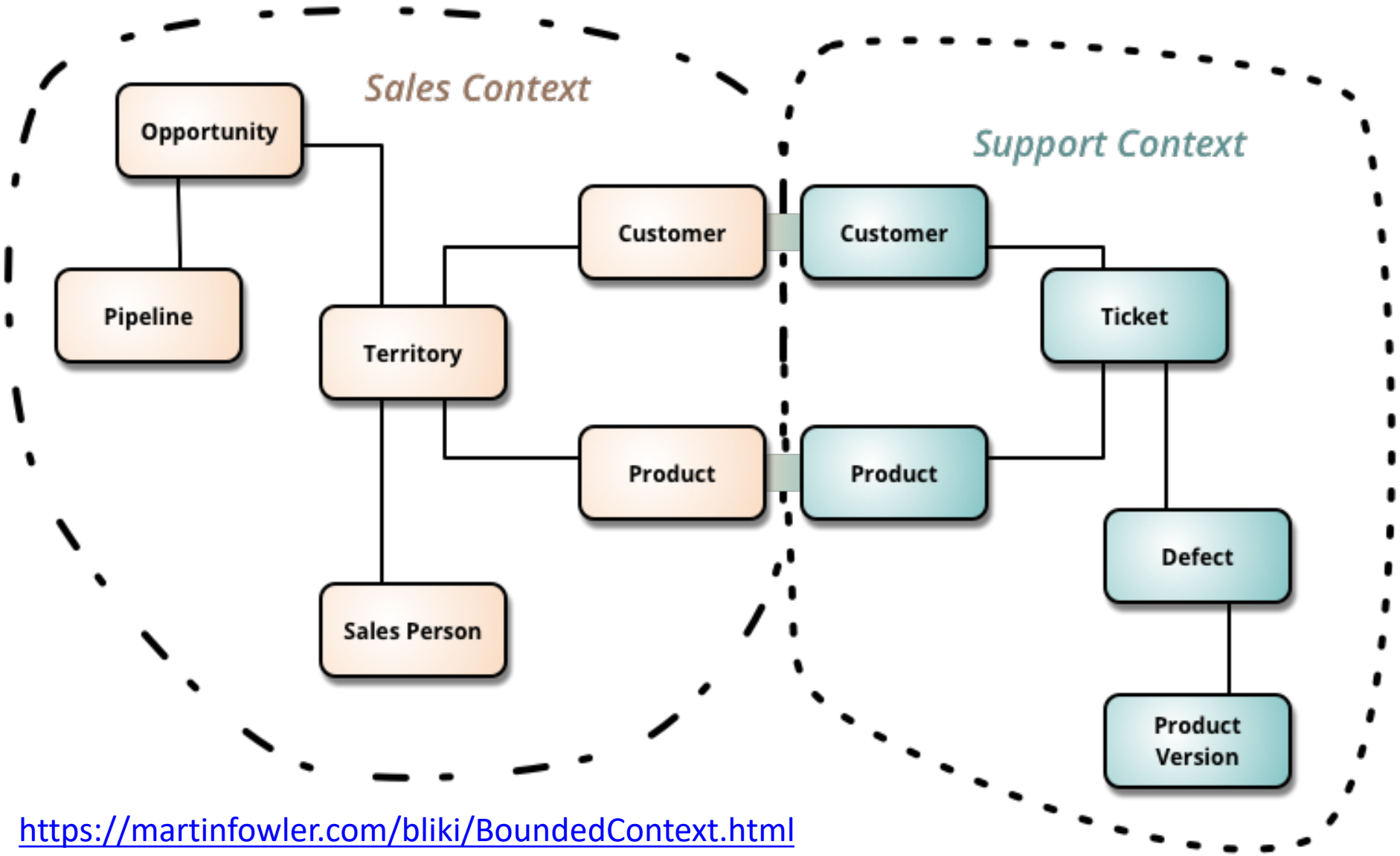
Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

Eric Evans

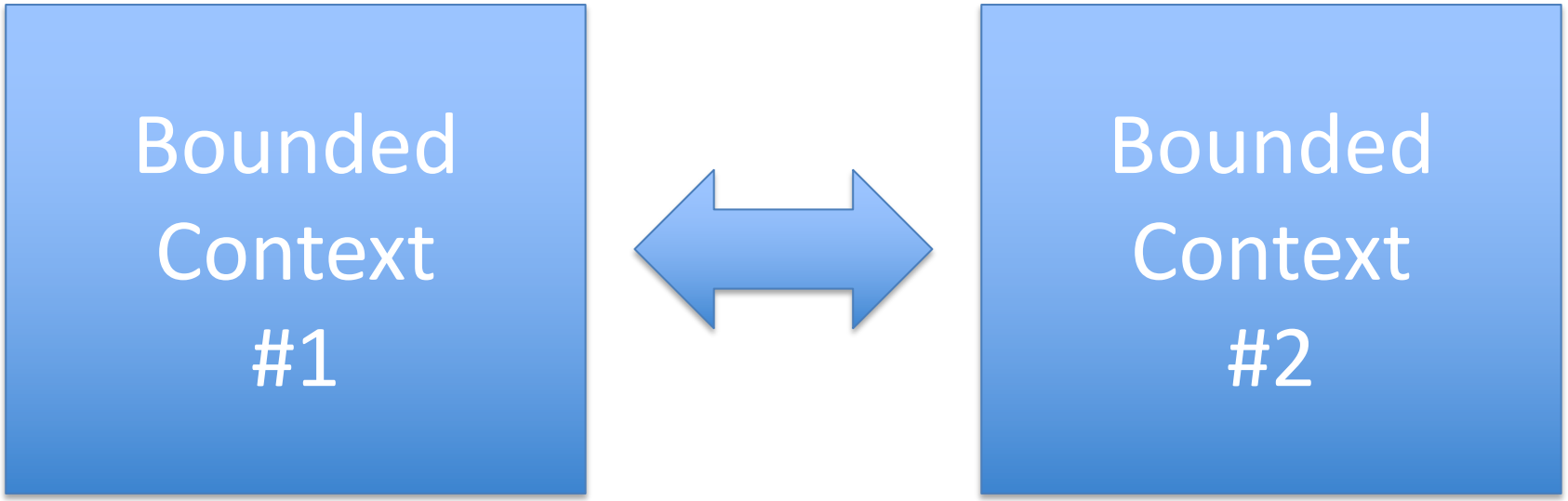
Foreword by Martin Fowler

Bounded Contexts

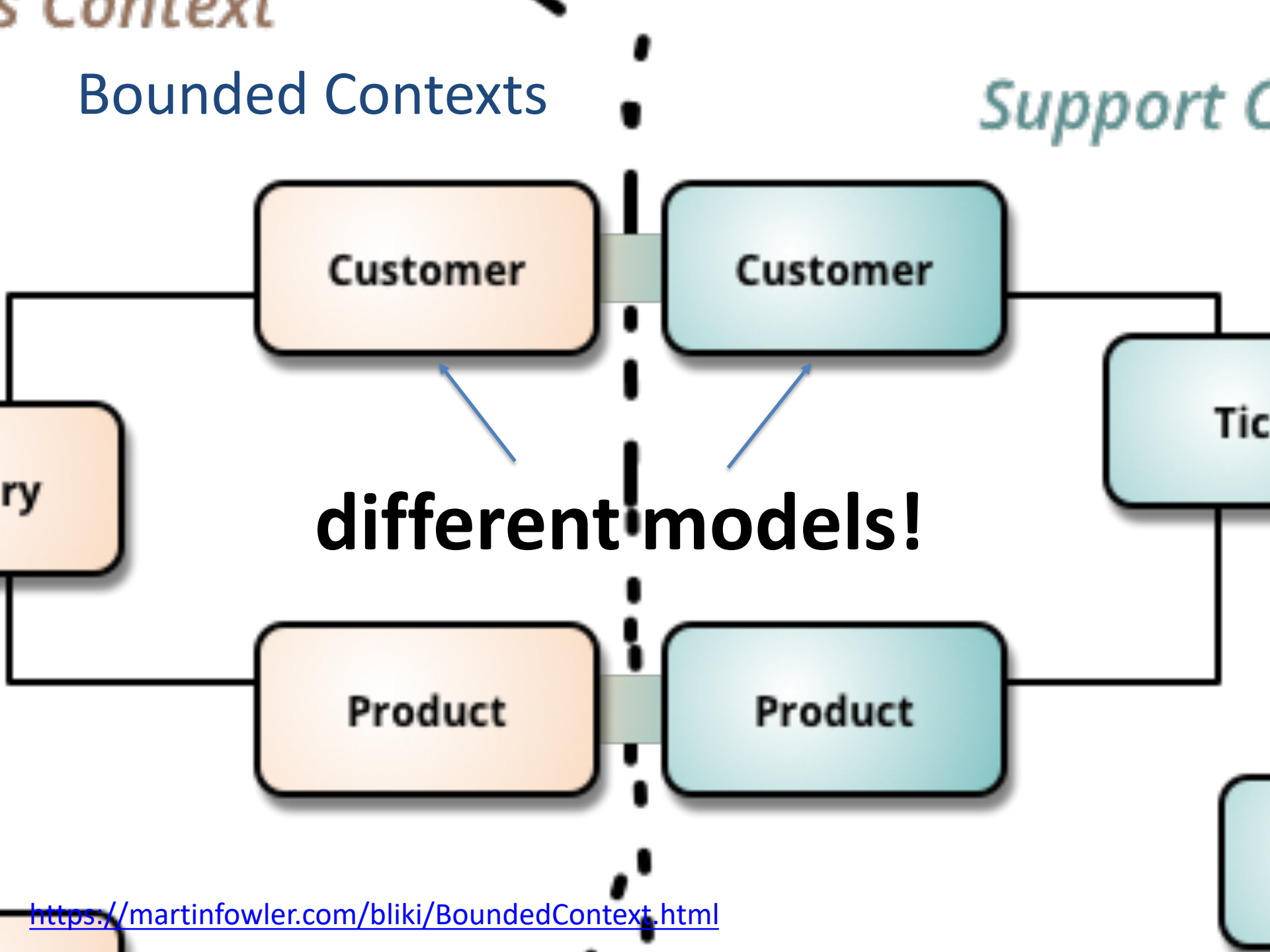


https://martinfowler.com/bliki/BoundedContext.html

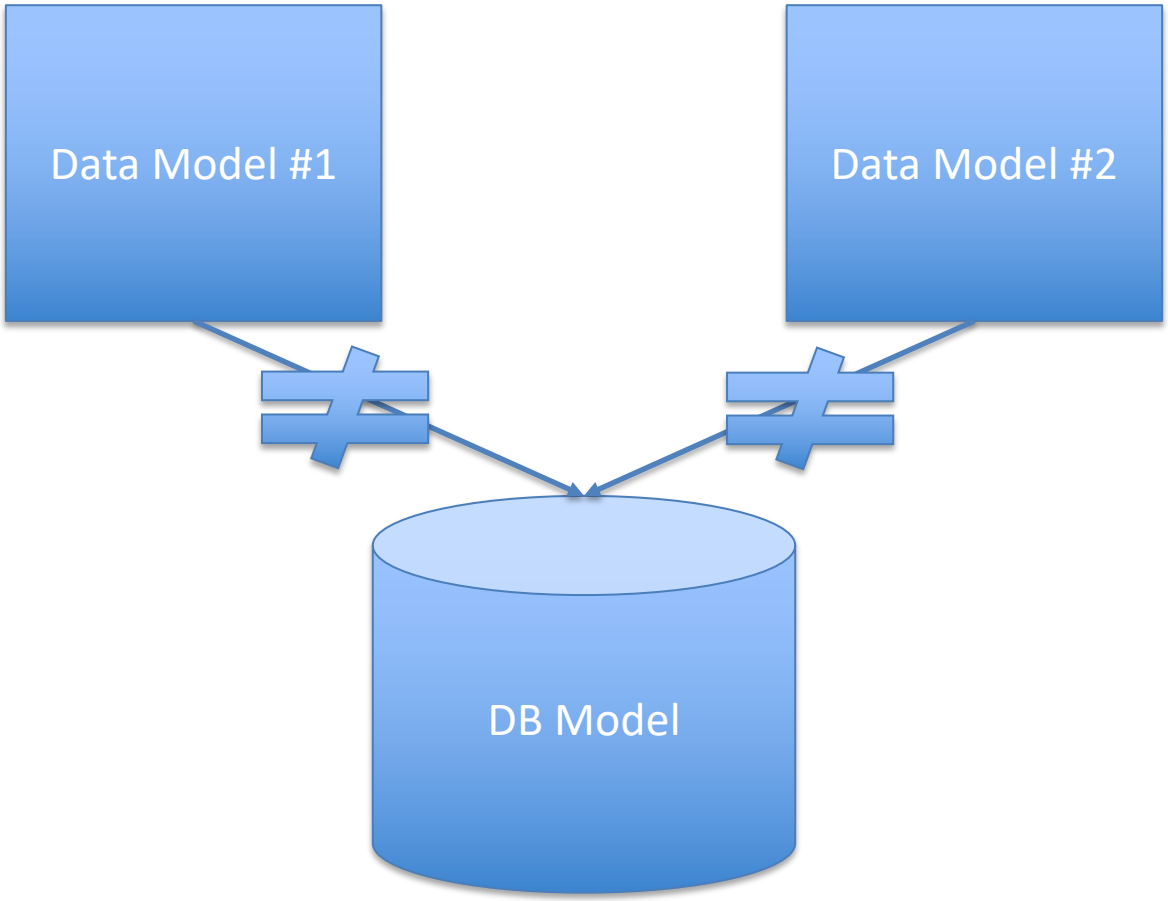
Goal



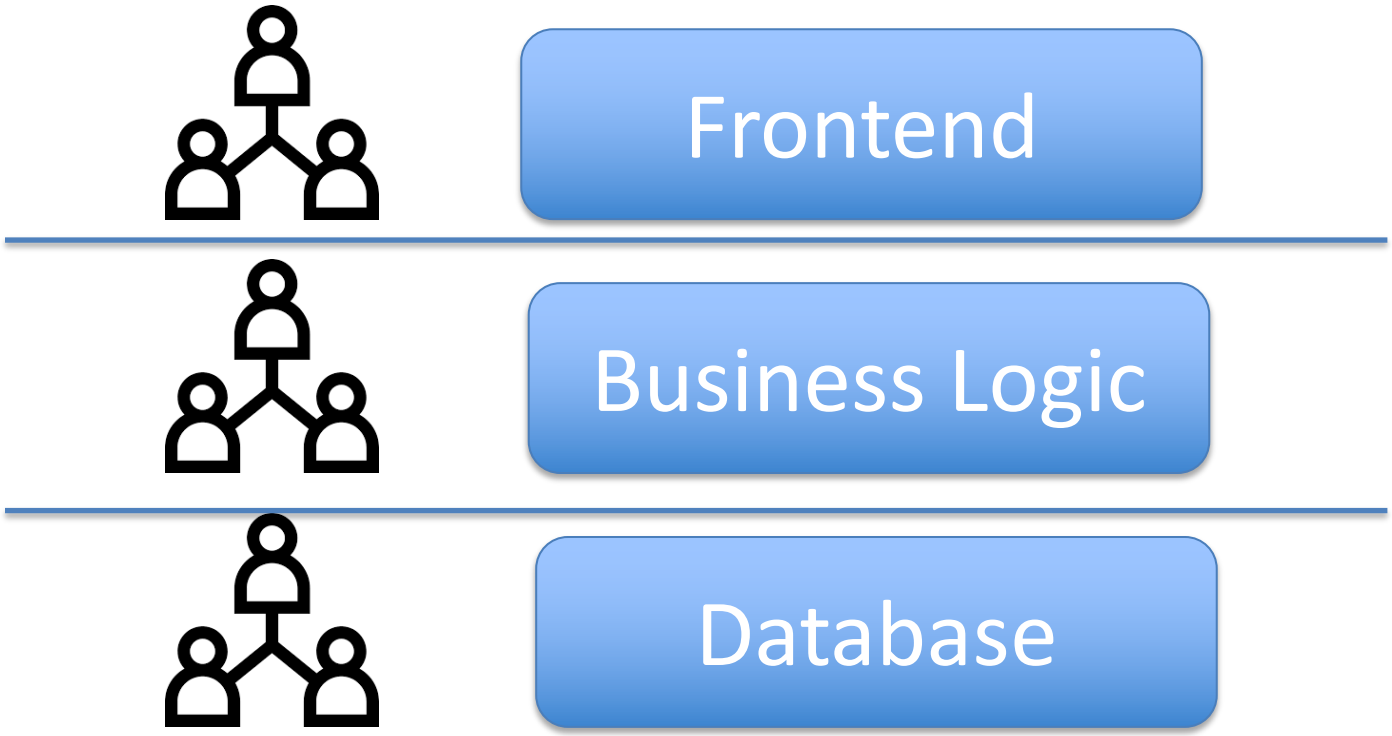
Bounded Contexts



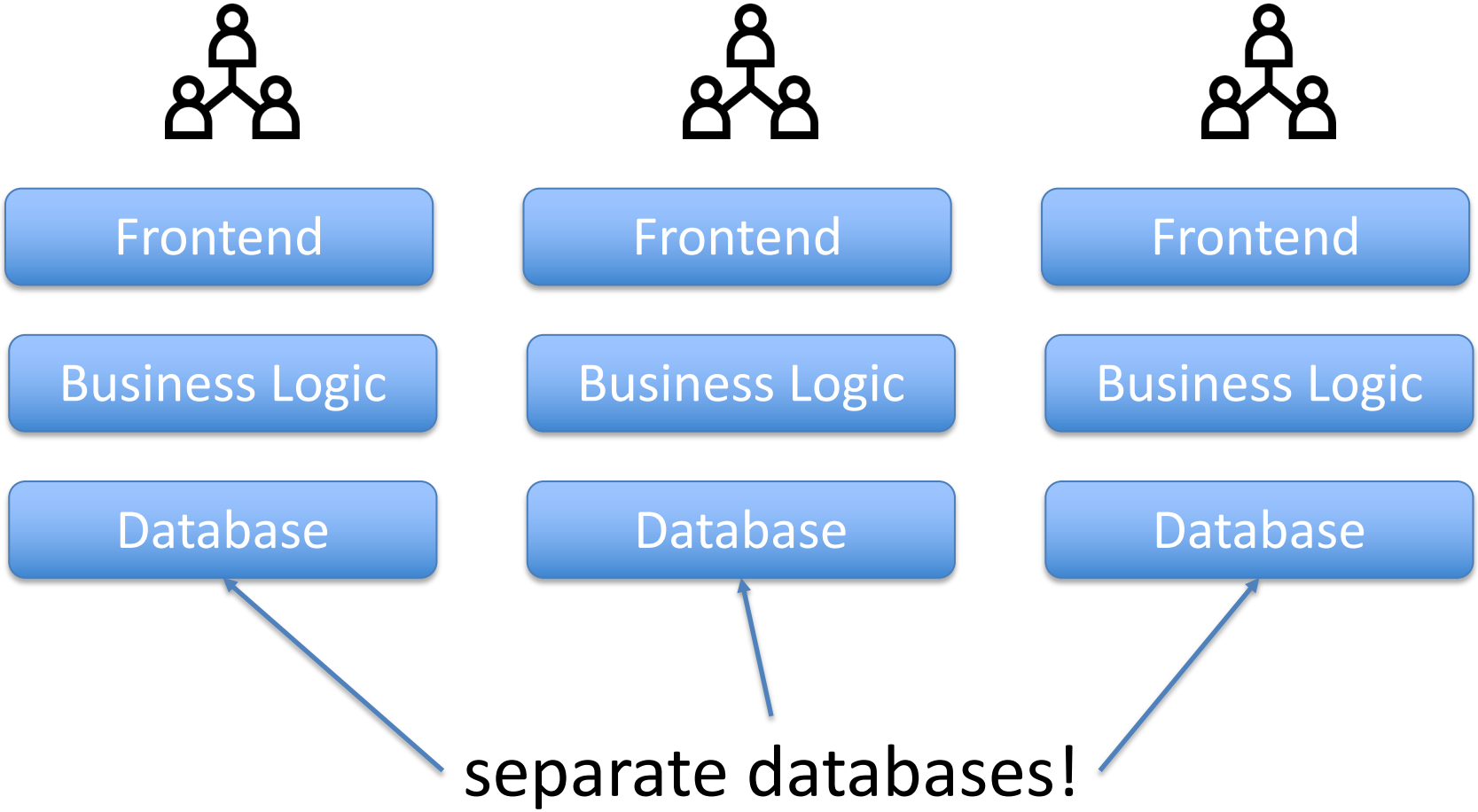
How to Sync Models?

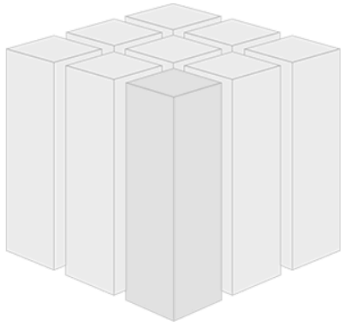


Layered Organization



Cross-Functional Teams



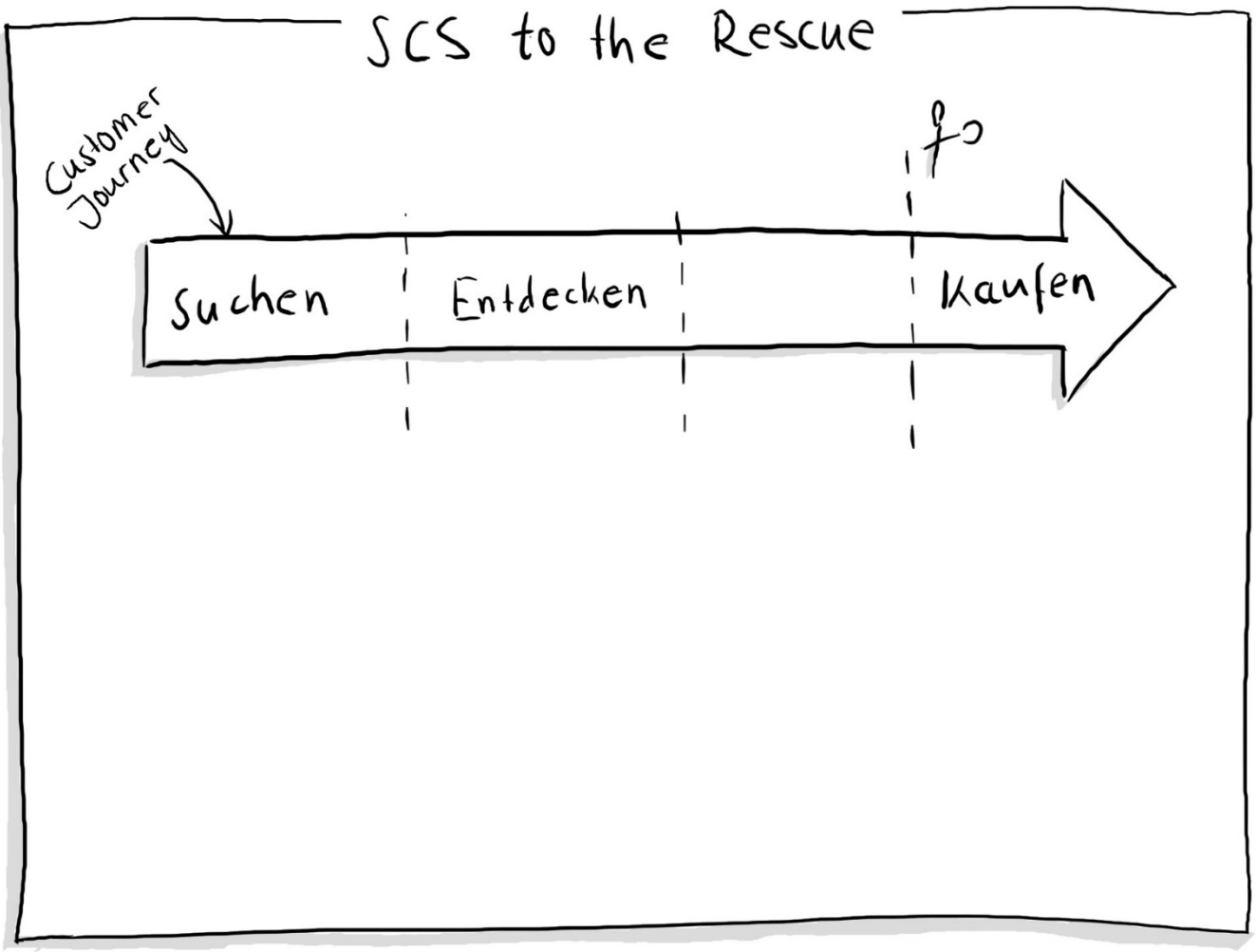


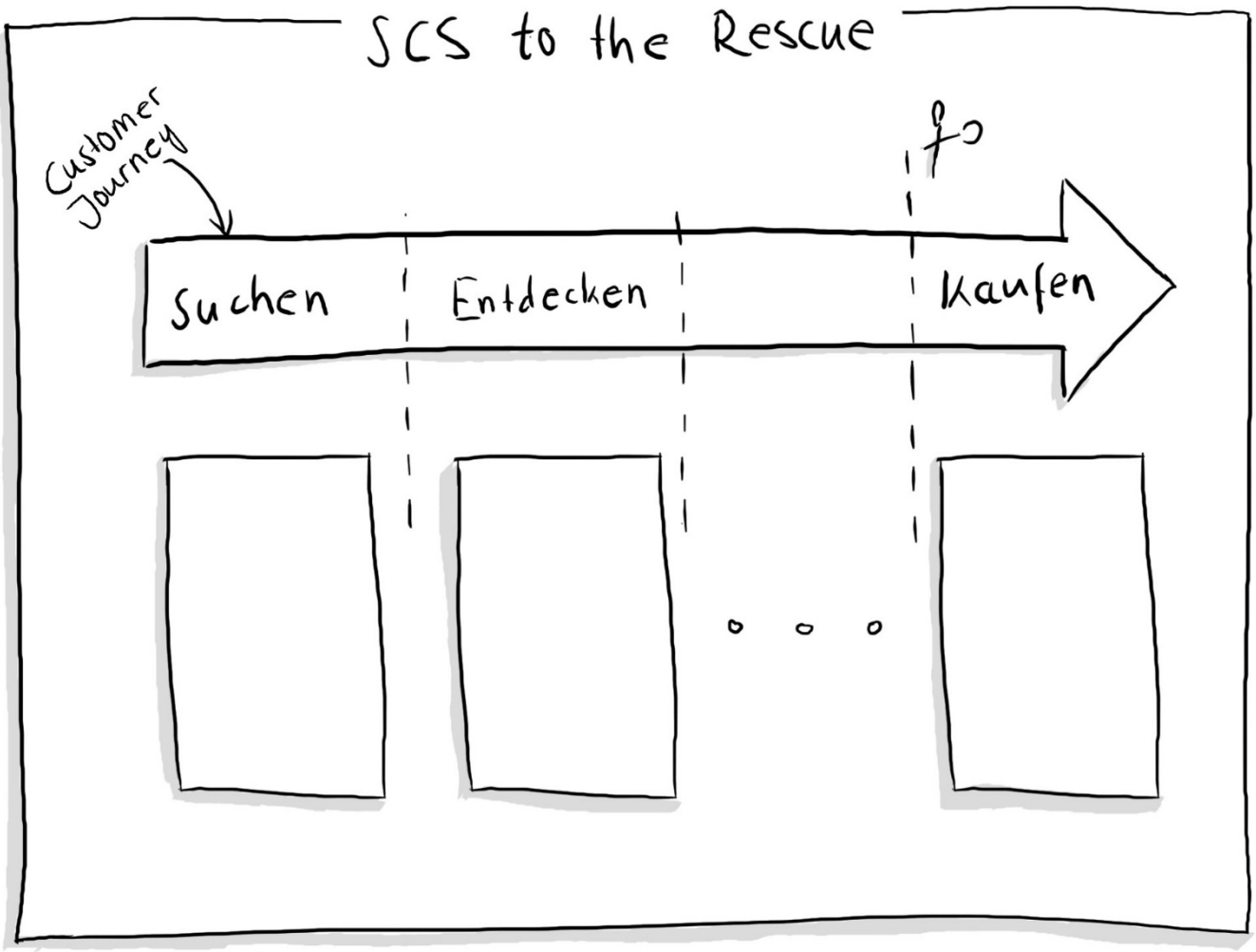
Self-Contained Systems

ASSEMBLING SOFTWARE FROM INDEPENDENT SYSTEMS

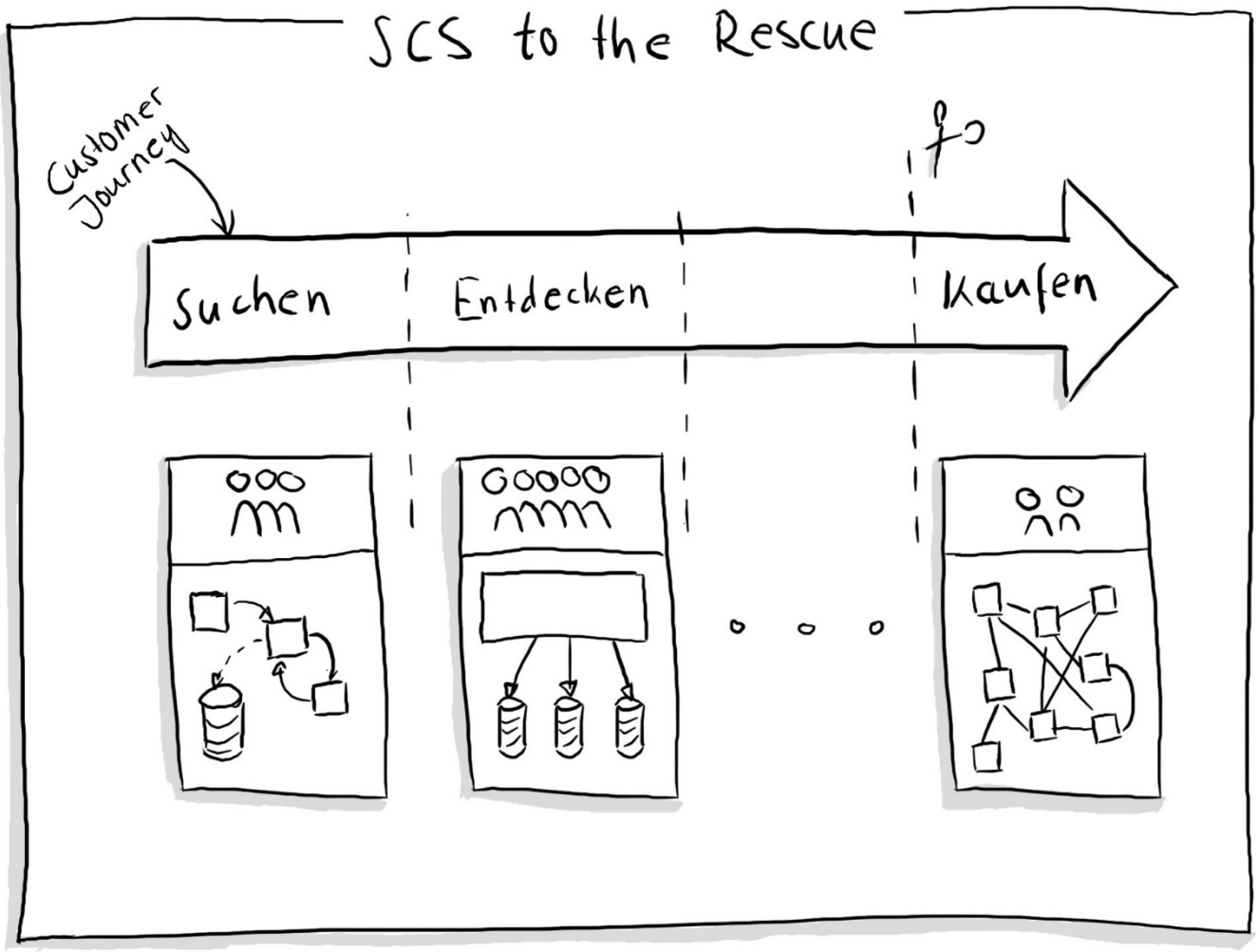
Introduction

The Self-contained System (SCS) approach is an architecture that focuses on a separation of the functionality into many independent systems, making the complete logical system a collaboration of many smaller software systems. This avoids the problem of large monoliths that grow constantly and eventually become unmaintainable. Over the past few years, we have seen its benefits in many mid-sized and large-scale projects.

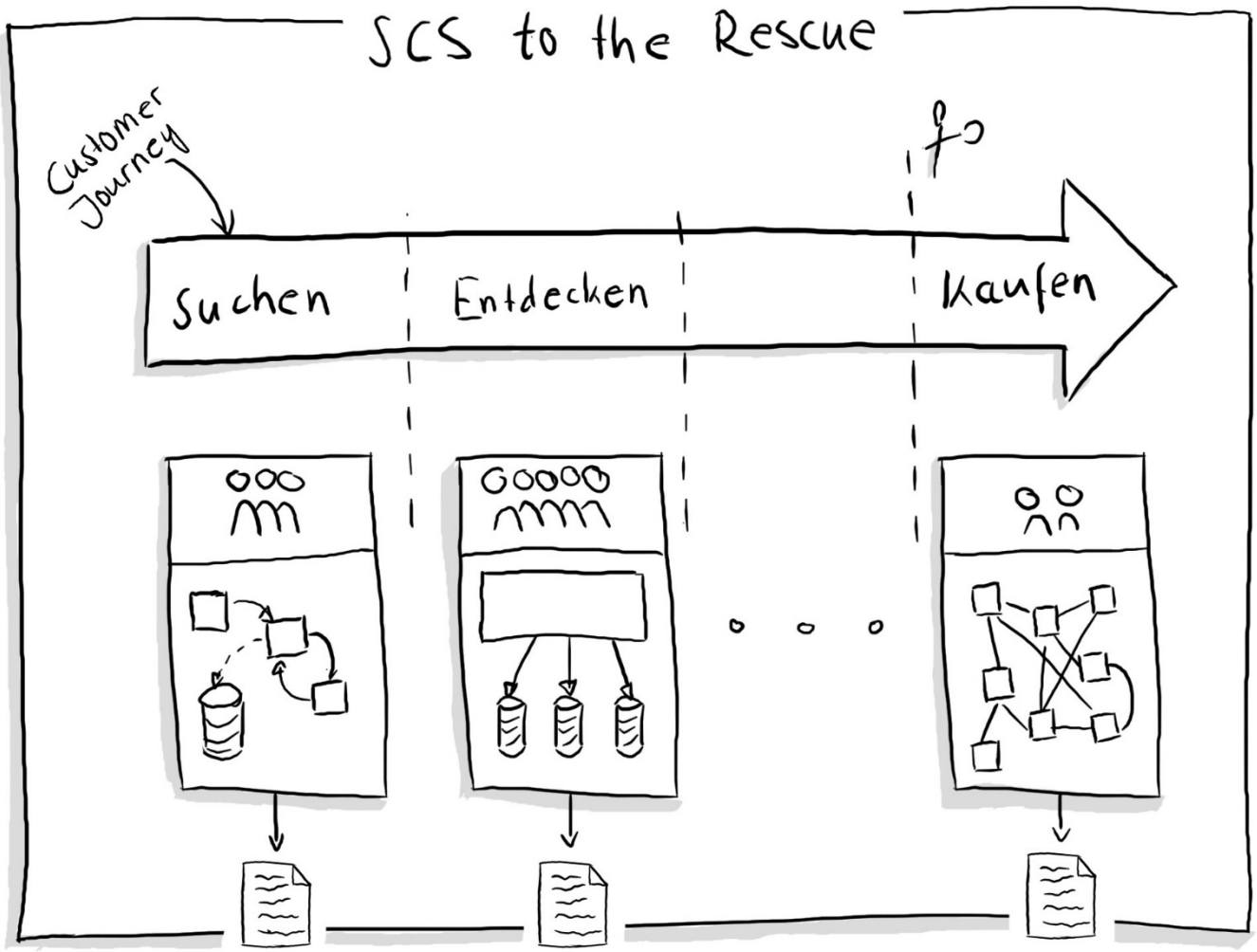

SCS to the Rescue
Customer Journey
Suchen
Entdecken
Kaufen



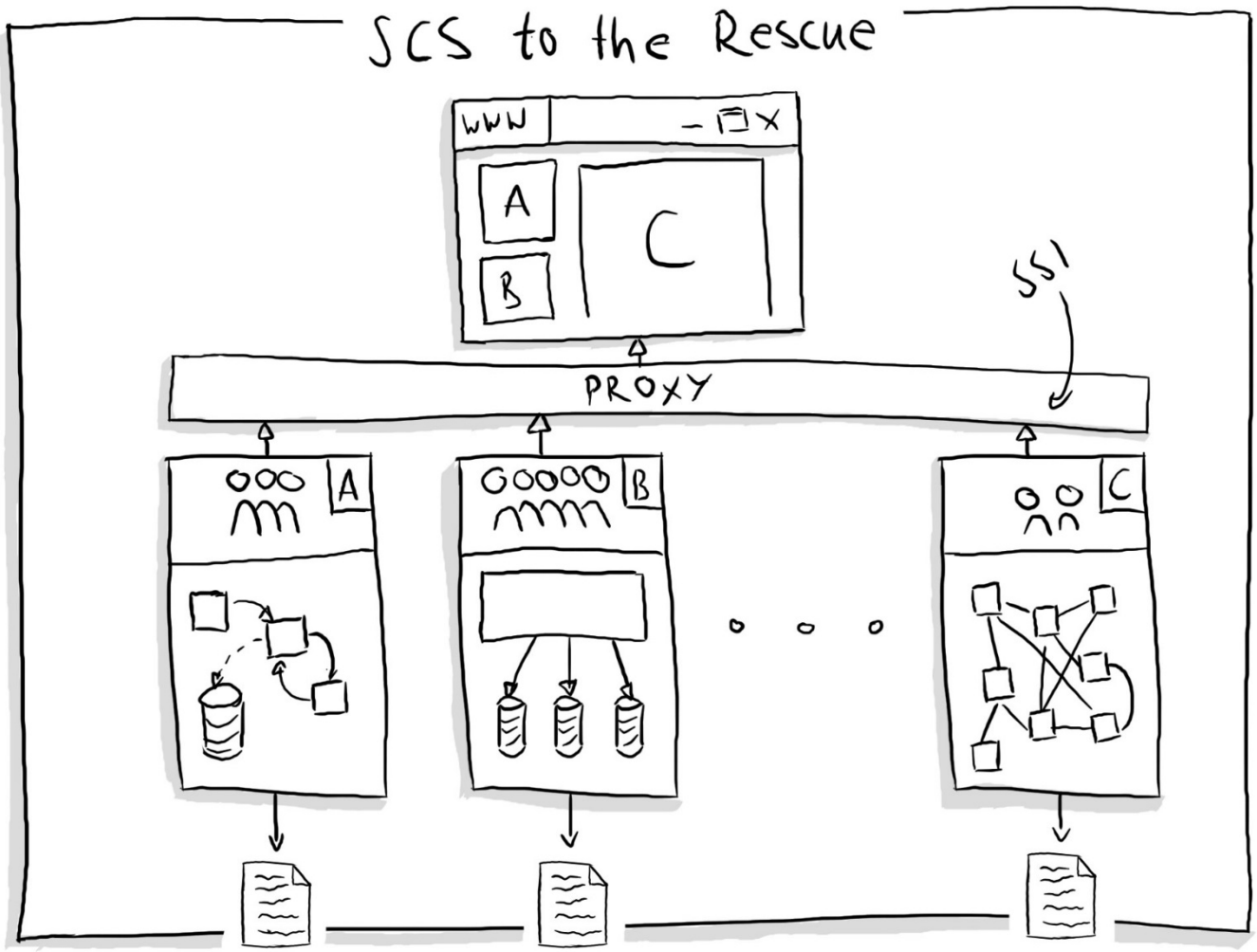
picture credit: Simon Härer



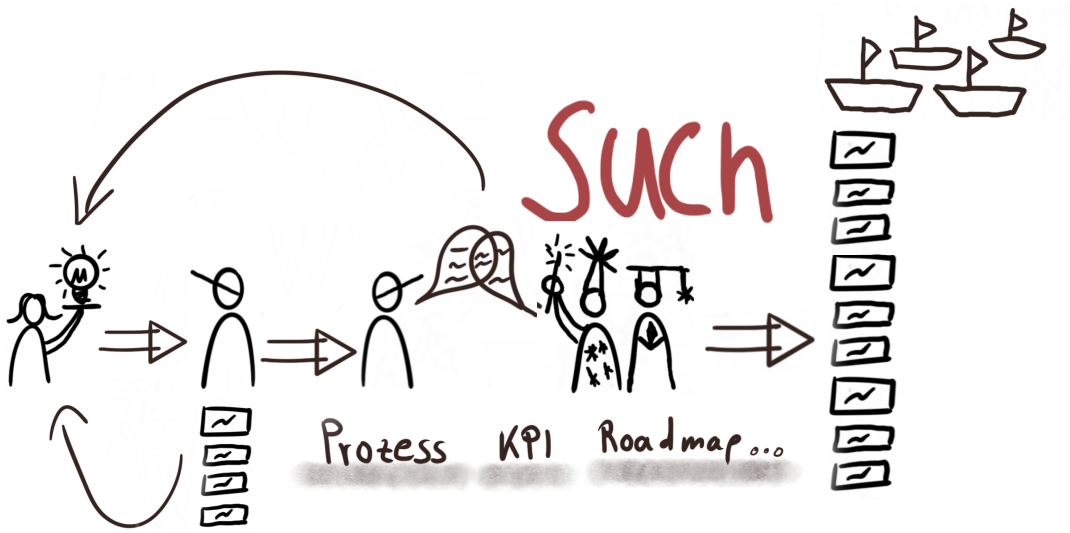
picture credit: Simon Härer



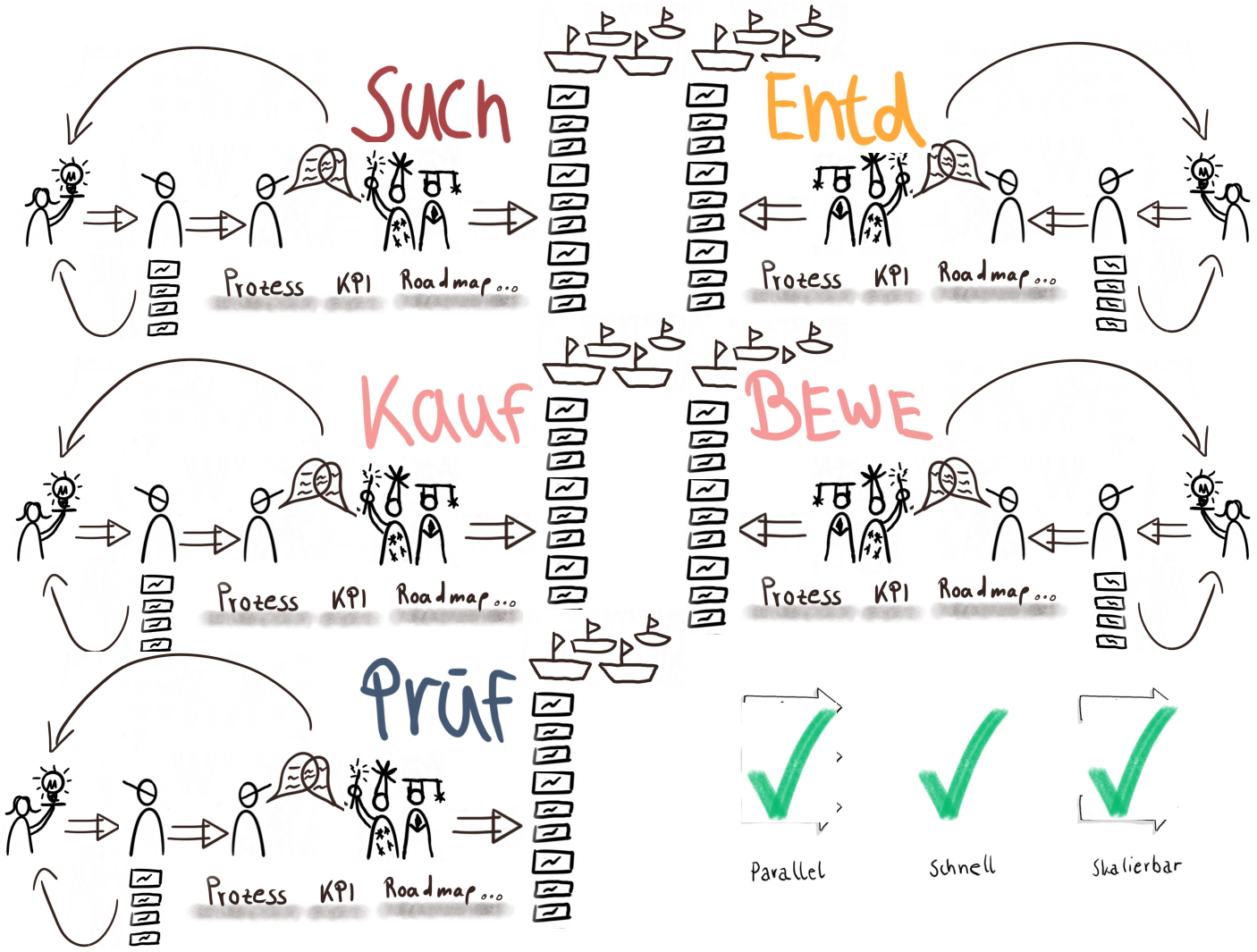
picture credit: Simon Härer



picture credit: Simon Härer

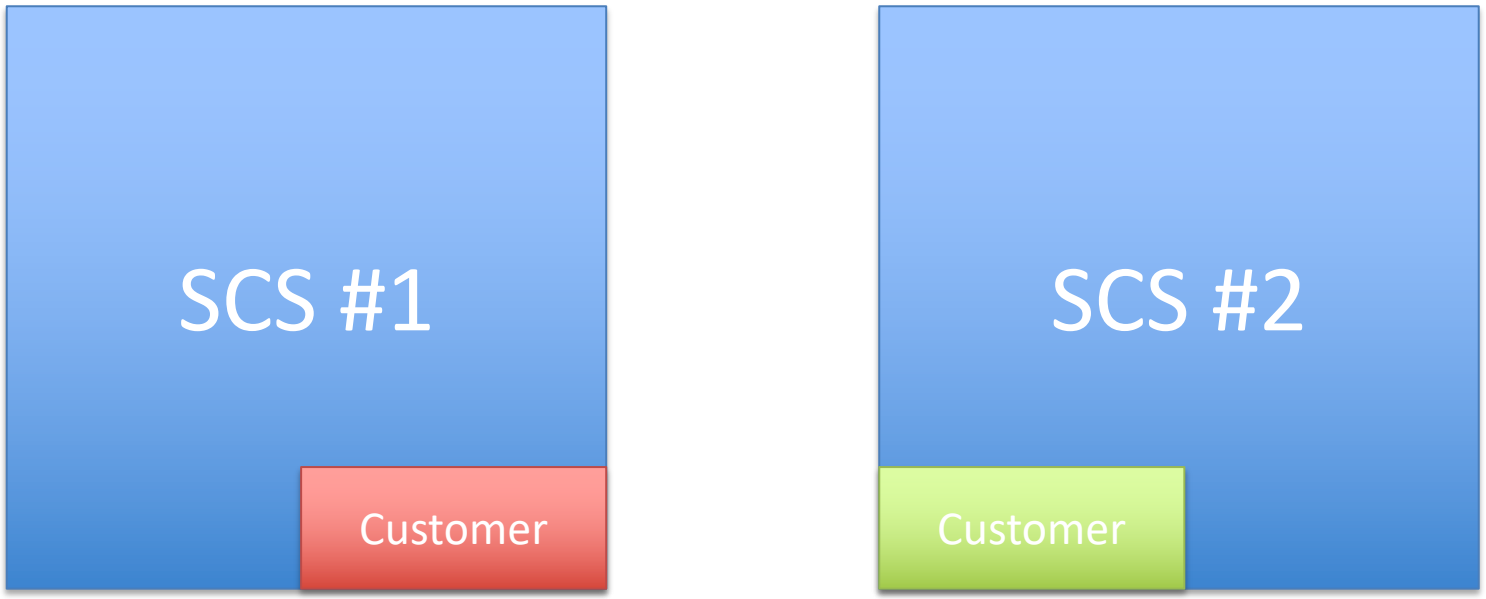


picture credit: Simon Härer



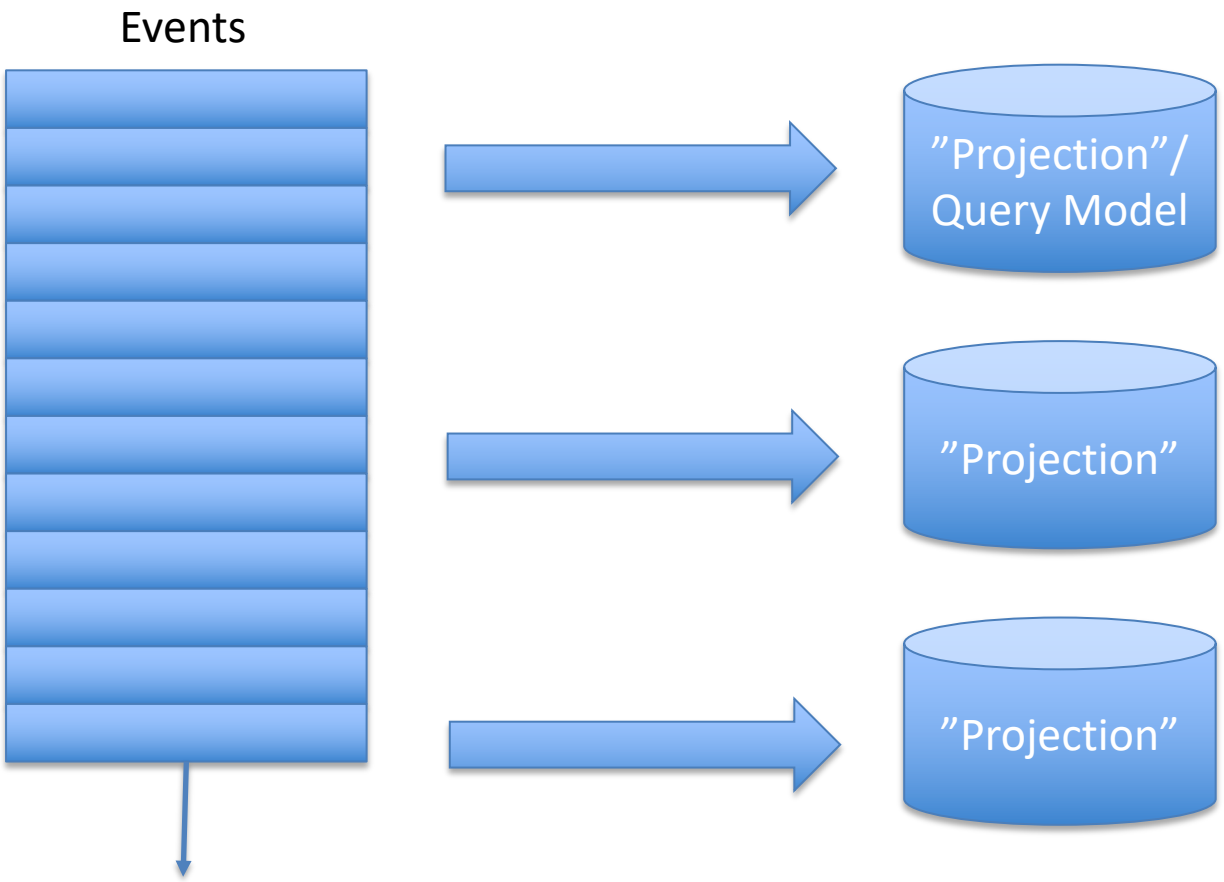
picture credit: Simon Härer

How to Communicate?

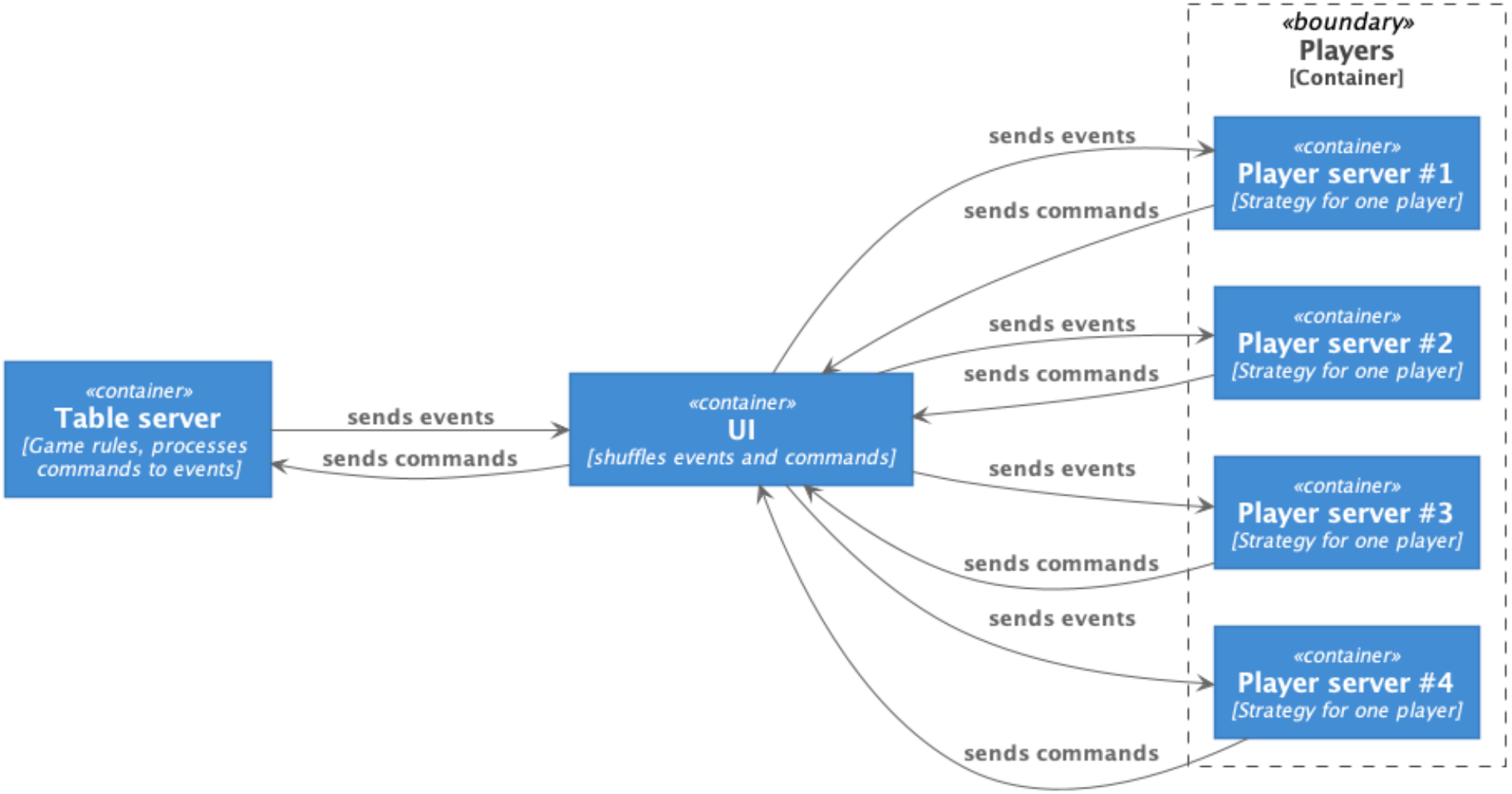


Store Facts, Not State!

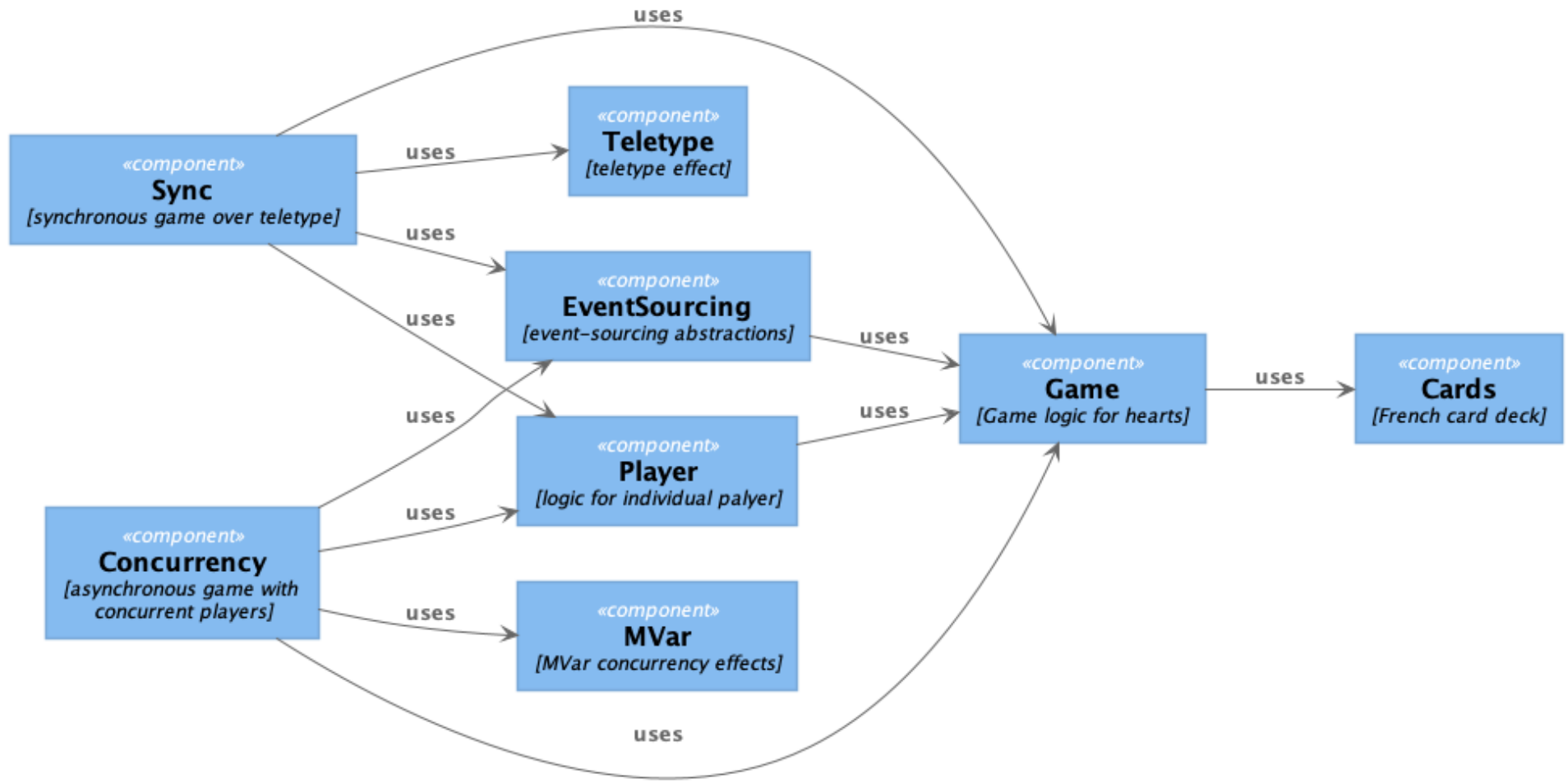
Events and Query Models



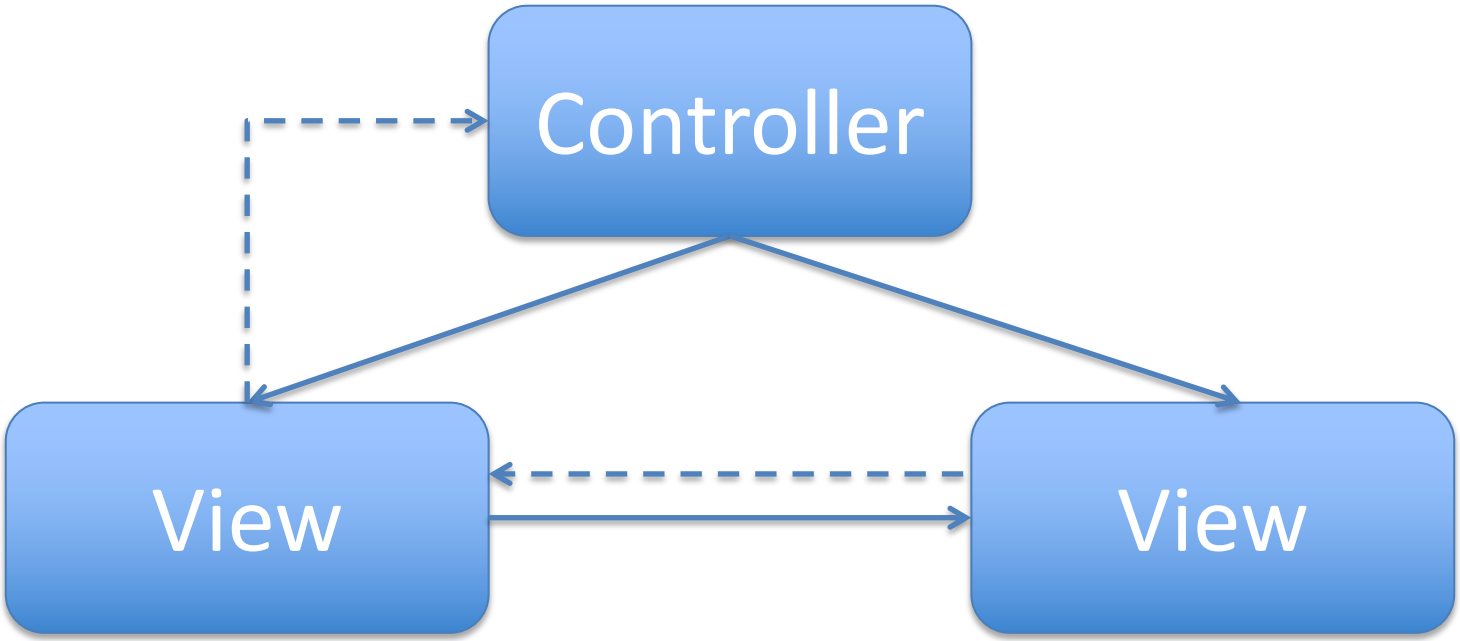
Hearts Containers



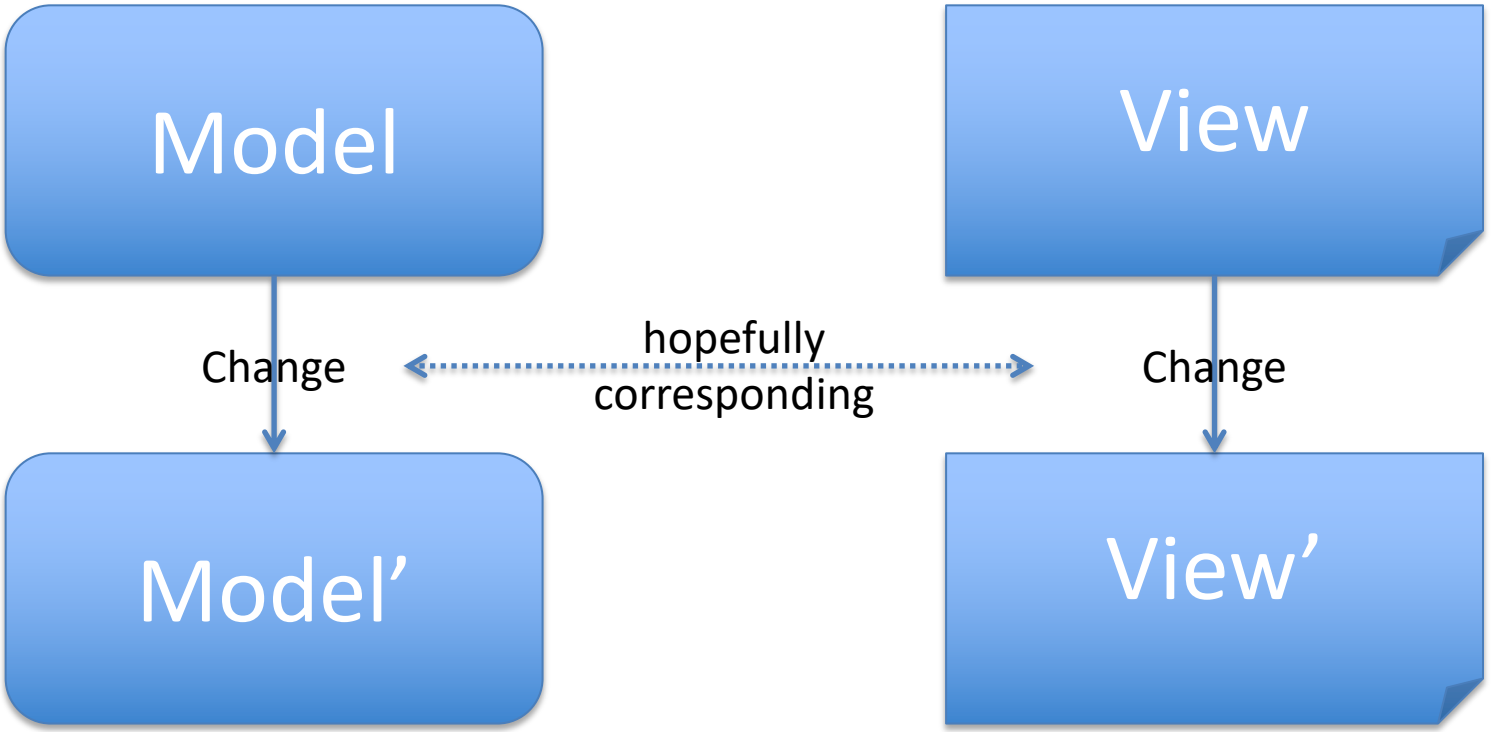
Hearts Components



MVC



Problem


Model
View
Change
hopefully
corresponding
Change
Model'
View'

Model - View - Update