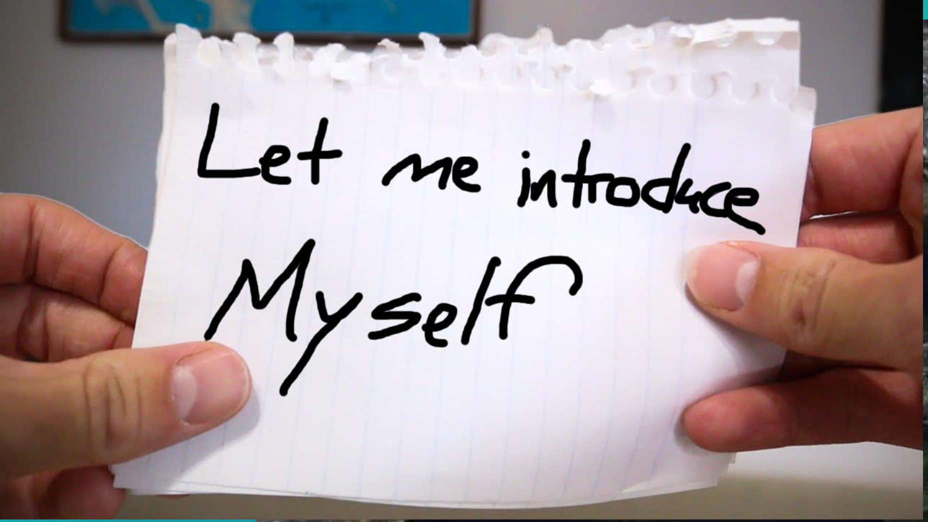
DISMANTLING THE MONOLITH

BY: BARRY TARLTON WITH MUCH SUPPORT FROM HIS FRIEND JEFF CAIN

AGENDA

- History of Building Monoliths
- Problems that come with Monoliths
- How Microservices help over come monolithic challenges
- How Cloud Technologies enable a Microservices approach
- What are API's and why are they so important
- How do we dissect the Monolithic problems into more manageable pieces
- How do Business Analyst play a key role in all of this


Let me introduce
Myself

EDUCATION

- Elementary & High School:
  - Basic Programming on Apple IIe

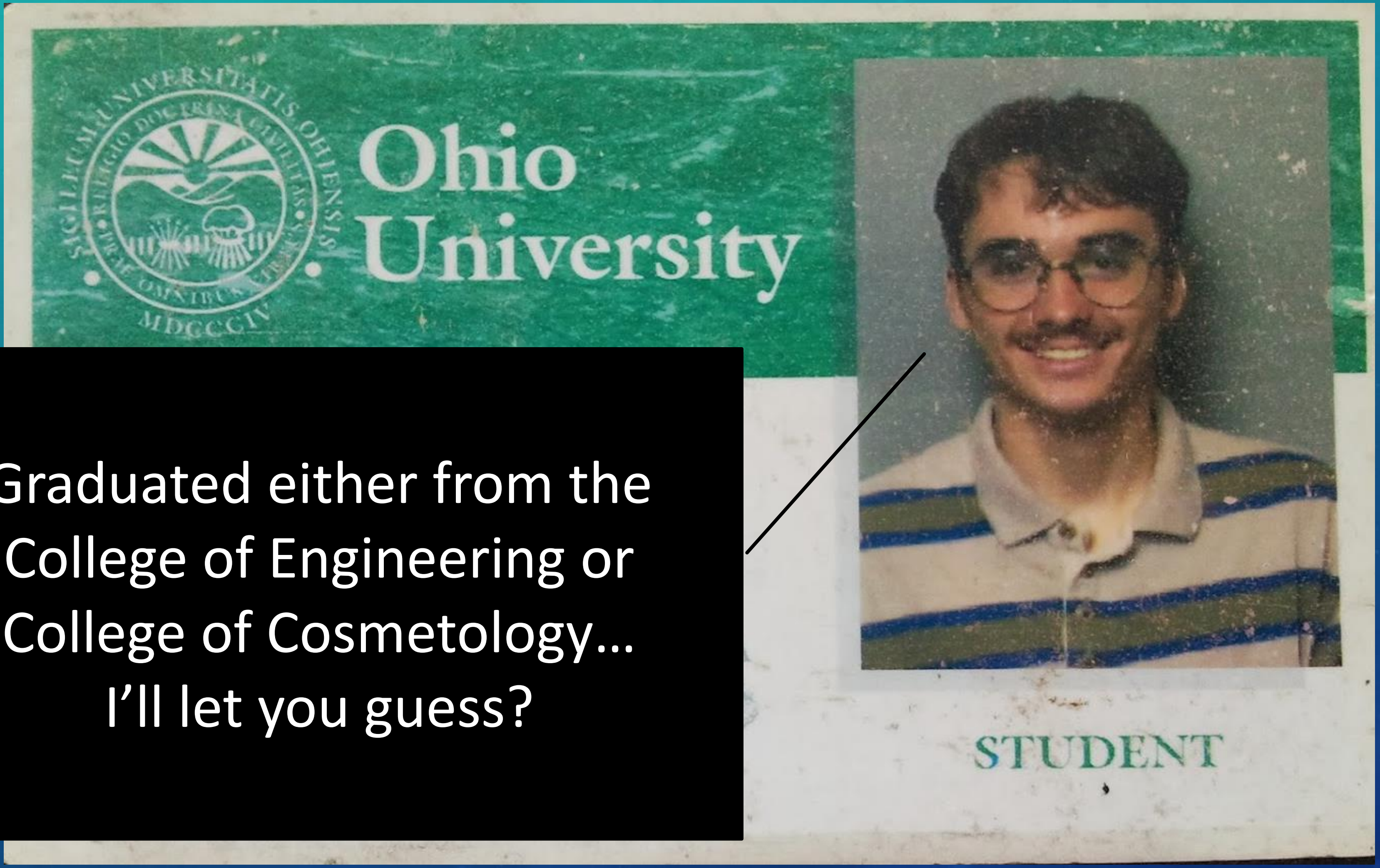
```
            APPLE II
  DOS VERSION 3.3  SYSTEM MASTER

          JANUARY 1, 1983

COPYRIGHT APPLE COMPUTER,INC. 1980,198

]
```

HIGHER EDUCATION



Graduated either from the College of Engineering or College of Cosmetology… I'll let you guess?

- 6 Years
- Retirement Applications for Public Sector

Covansys®

NATIONWIDE 12+ YEARS

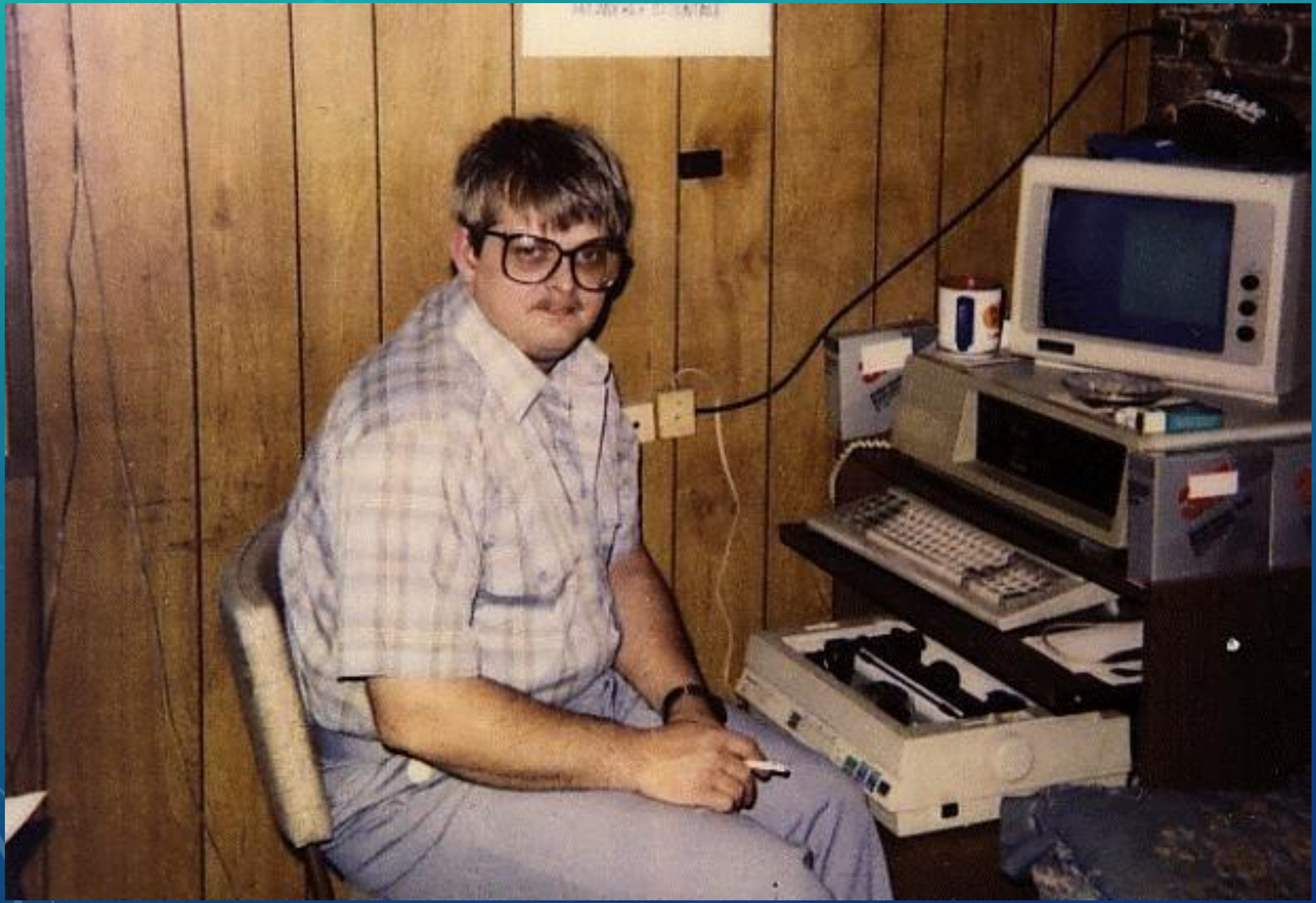
NATIONWIDE INSURANCE

Nationwide is on your side

In the Beginning… Was the computer…

In the Beginning…
Was the computer…

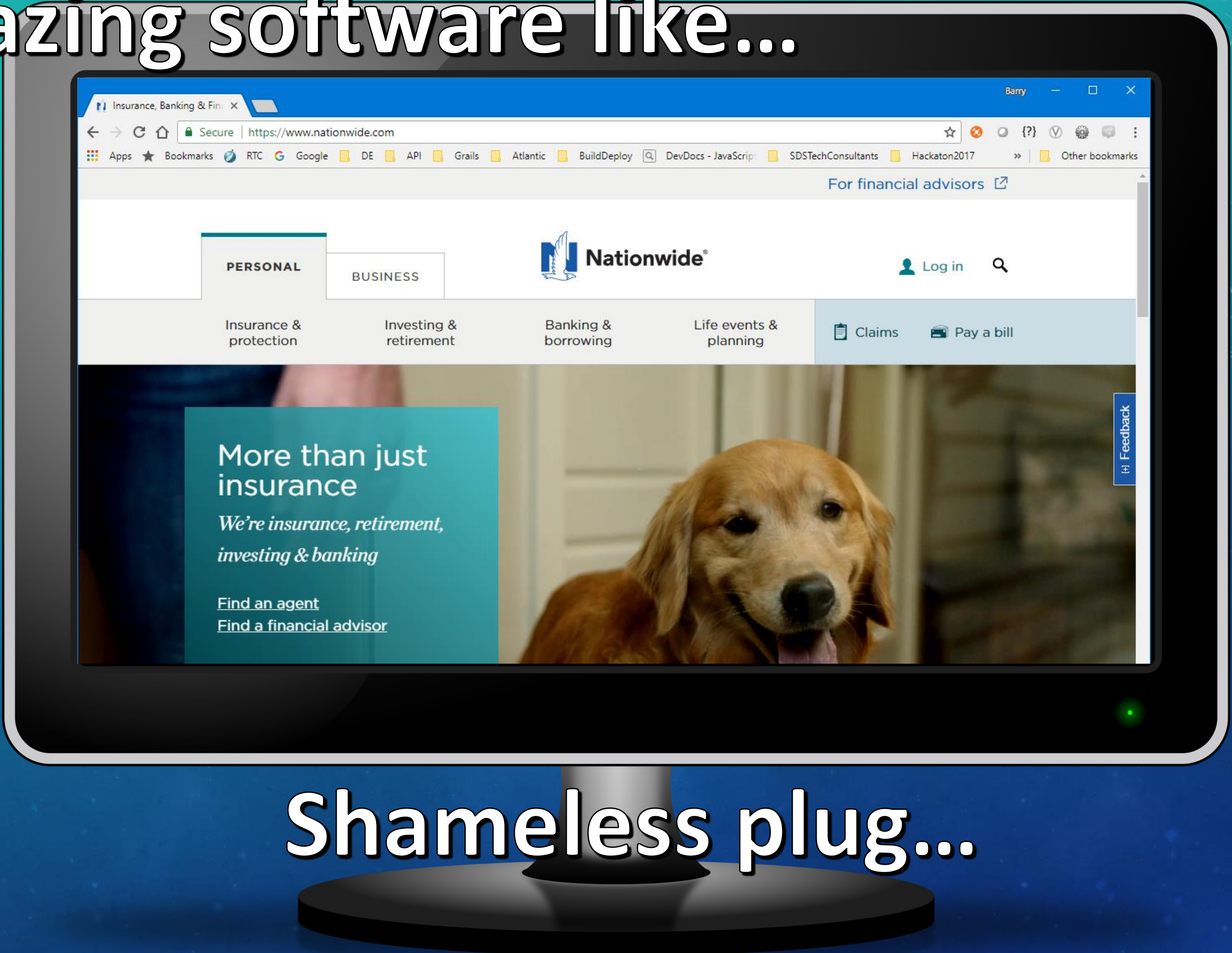
And ever since there were computers… there were Programmers…



That wrote code…
That no one cared about…

Then came the Business Analysts To ensure software did something useful!

Amazing software like…



Barry

Insurance, Banking & Fin

Secure | https://www.nationwide.com

Apps Bookmarks RTC Google DE API Grails Atlantic BuildDeploy DevDocs - JavaScrip SDSTechConsultants Hackaton2017 » Other bookmarks

For financial advisors

PERSONAL BUSINESS

Nationwide®

Log in

Insurance & protection | Investing & retirement | Banking & borrowing | Life events & planning | Claims | Pay a bill

More than just insurance

We're insurance, retirement, investing & banking

Find an agent

Find a financial advisor

Feedback

Shameless plug…

And software developers rejoiced!

MY FIRST MONOLITHIC APPLICATION – RELEASE 1

Retirement App

- Retiree Data
- Dependent’s data
- Employment History
- Calculate Years of Service
- Employee Contributions

SECOND RELEASE

Retirement App

- Retiree Data
- Dependent’s data
- Employment History
- Calculate Years of Service
- Employee Contributions
- Employer Contributions
- Calculate Interest
- Retirement Plans

THIRD RELEASE

Retirement App

- Retiree Data
- Dependent’s data
- Employment History
- Calculate Years of Service
- Employee Contributions
- Employer Contributions
- Calculate Interest
- Retirement Plans

- Address History
- Document History
- Non-contributory work history
- Calculate Years of Service

FOURTH RELEASE

Retirement App

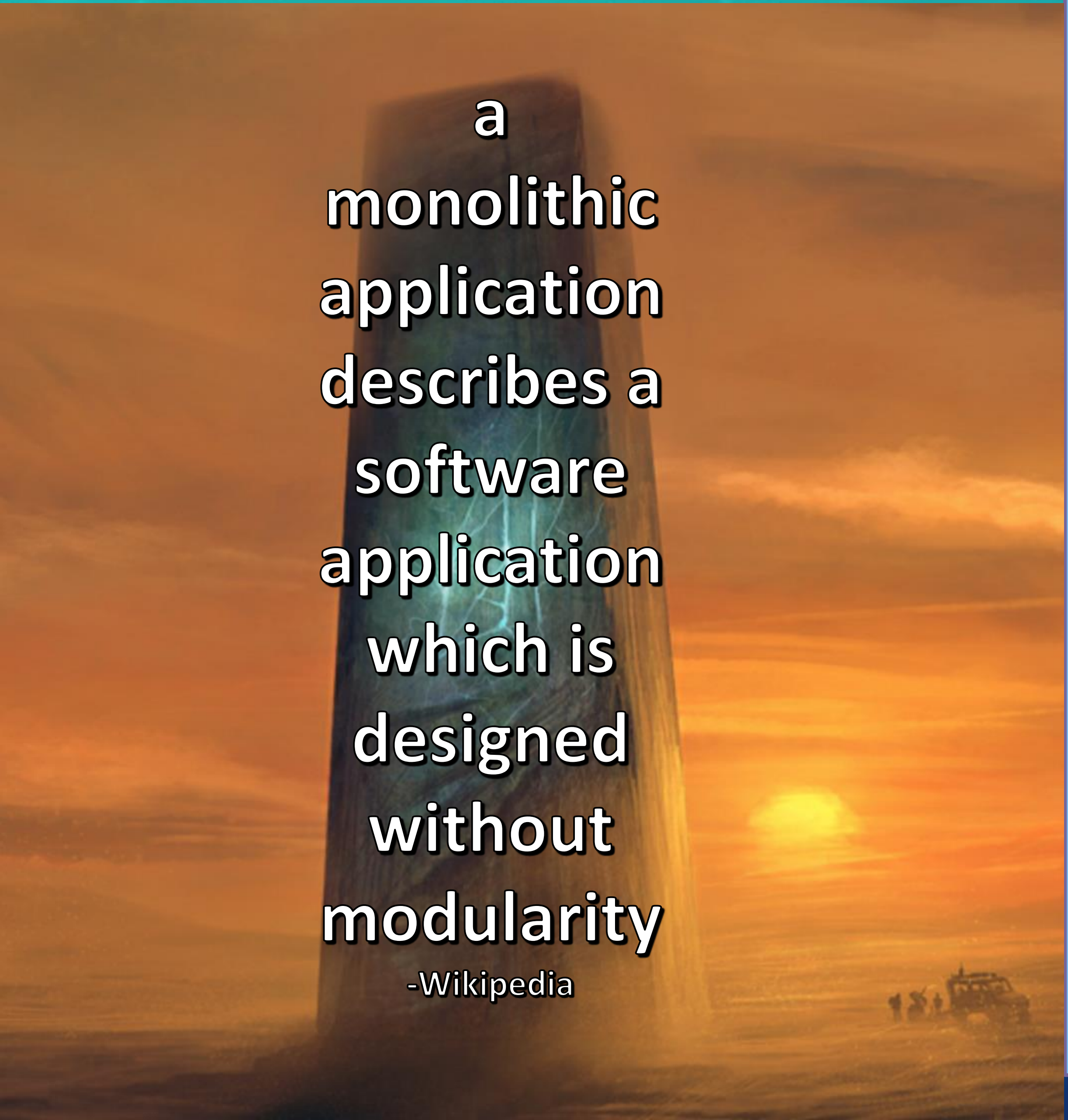
• Retiree Data	• Address History
• Dependent’s data	• Document History
• Employment History	• Non-contributory work history
• Calculate Years of Service	• Calculate Years of Service
• Employee Contributions	• Benefits Estimates
• Employer Contributions	• Retirement Options
• Calculate Interest	• Benefit Calculations
• Retirement Plans	• Beneficiary Management

FIFTH RELEASE

Retirement App

• Retiree Data	• Address History	• Self Service Member Data
• Dependent's data	• Document History	• Self Service Employment Hist
• Employment History	• Non-contributory work history	• Retirement Planning
• Calculate Years of Service	• Calculate Years of Service	• Pension Summary
• Employee Contributions	• Benefits Estimates	• Credit Summary
• Employer Contributions	• Retirement Options	• Banked Hours
• Calculate Interest	• Benefit Calculations	• Email Notifications
• Retirement Plans	• Beneficiary Management	• Eligibility Calculations

WHAT IS A MONOLITH?



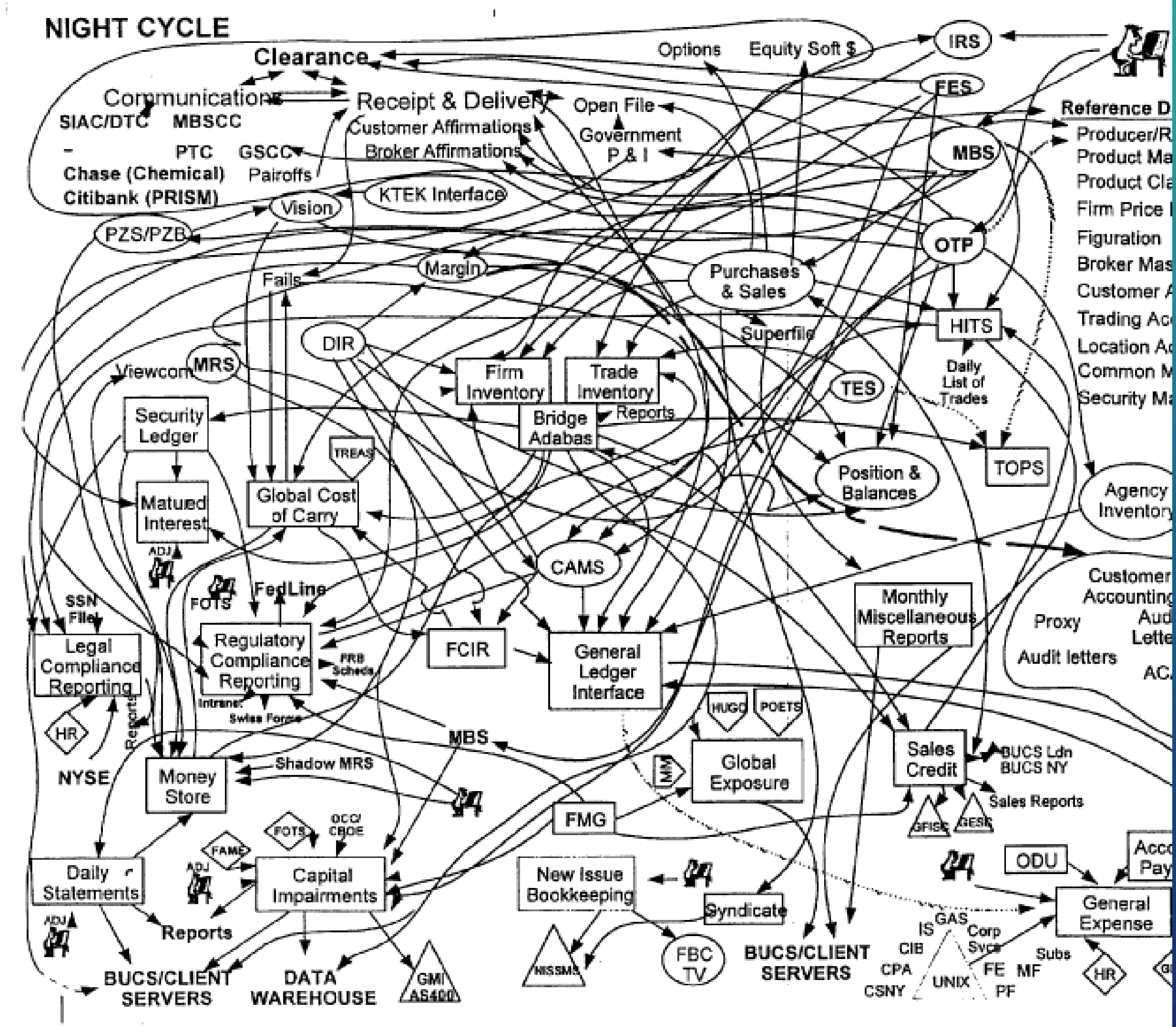
a monolithic application describes a software application which is designed without modularity

-Wikipedia

Monolith App

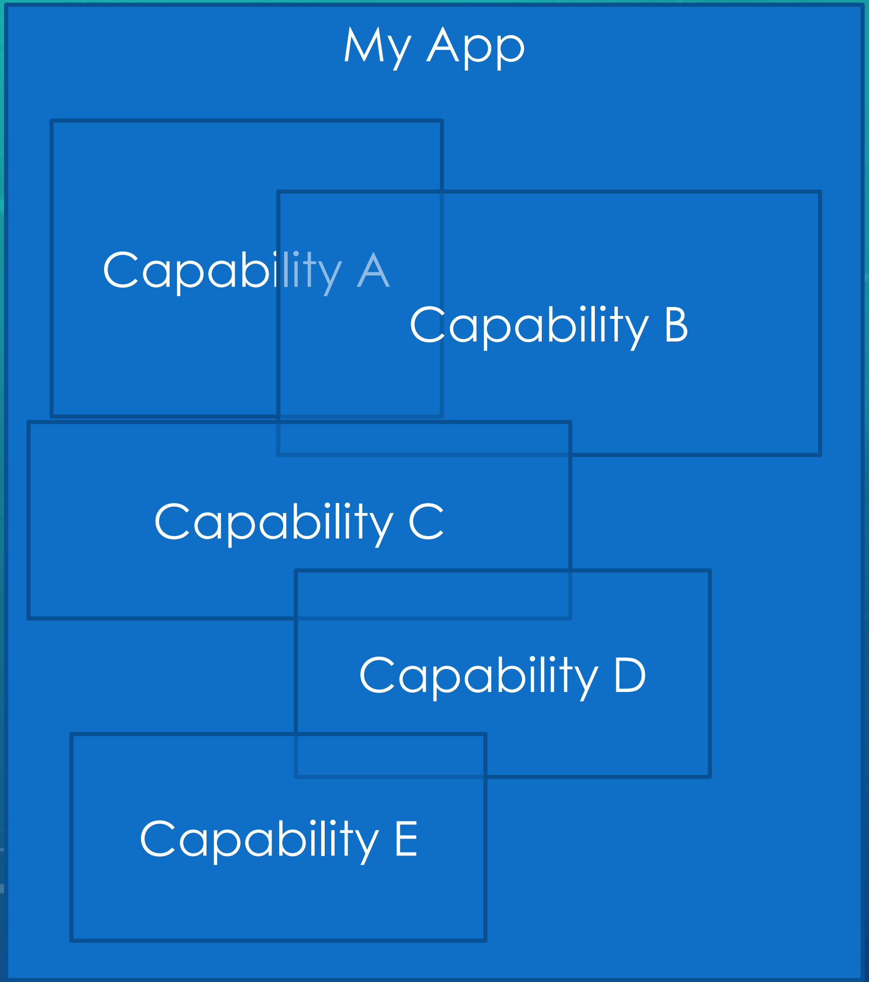
- Retiree Data
- nt's data
- Employment History
- Years of Service
- Employee Contributions
- Employer Contributions
- Calculate Interest
- Retirement Plans
- Address History
- Document History
- Non-contributory work history
- Calculate Years of Service
- Benefits Estimates
- Retirement Op
- Benefit Calculations
- Beneficiary Management
- Self Service Member Data
- Self Service Employment Hist
- Retirement Planning
- Pension Summary
- Credit Summary
- Banked Hours
- Email Notifications
- Eligibility Calculations

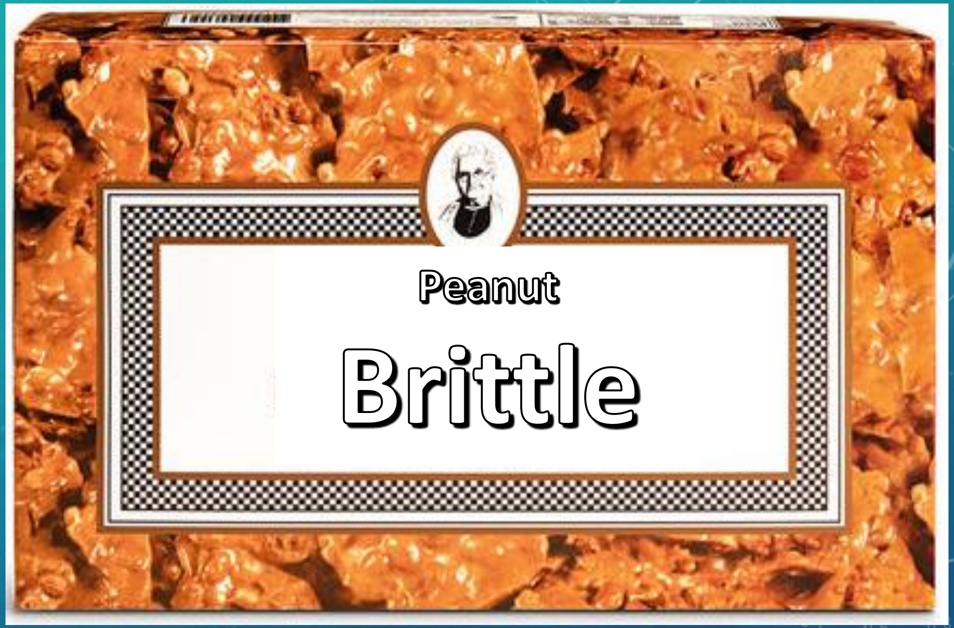
what's the Problem?



JUST TOO MUCH TO TAKE IN

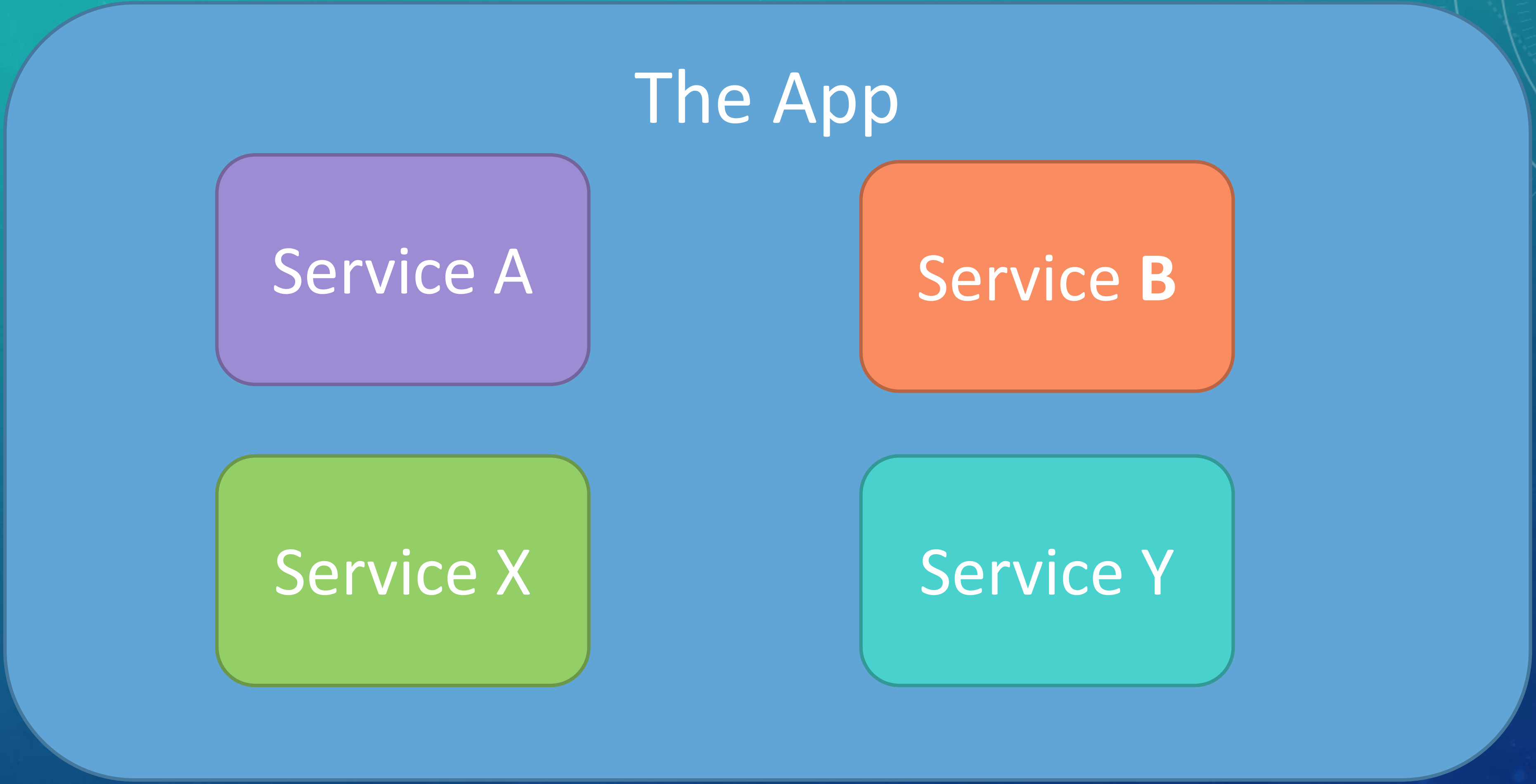
DIFFICULT TO MAINTAIN



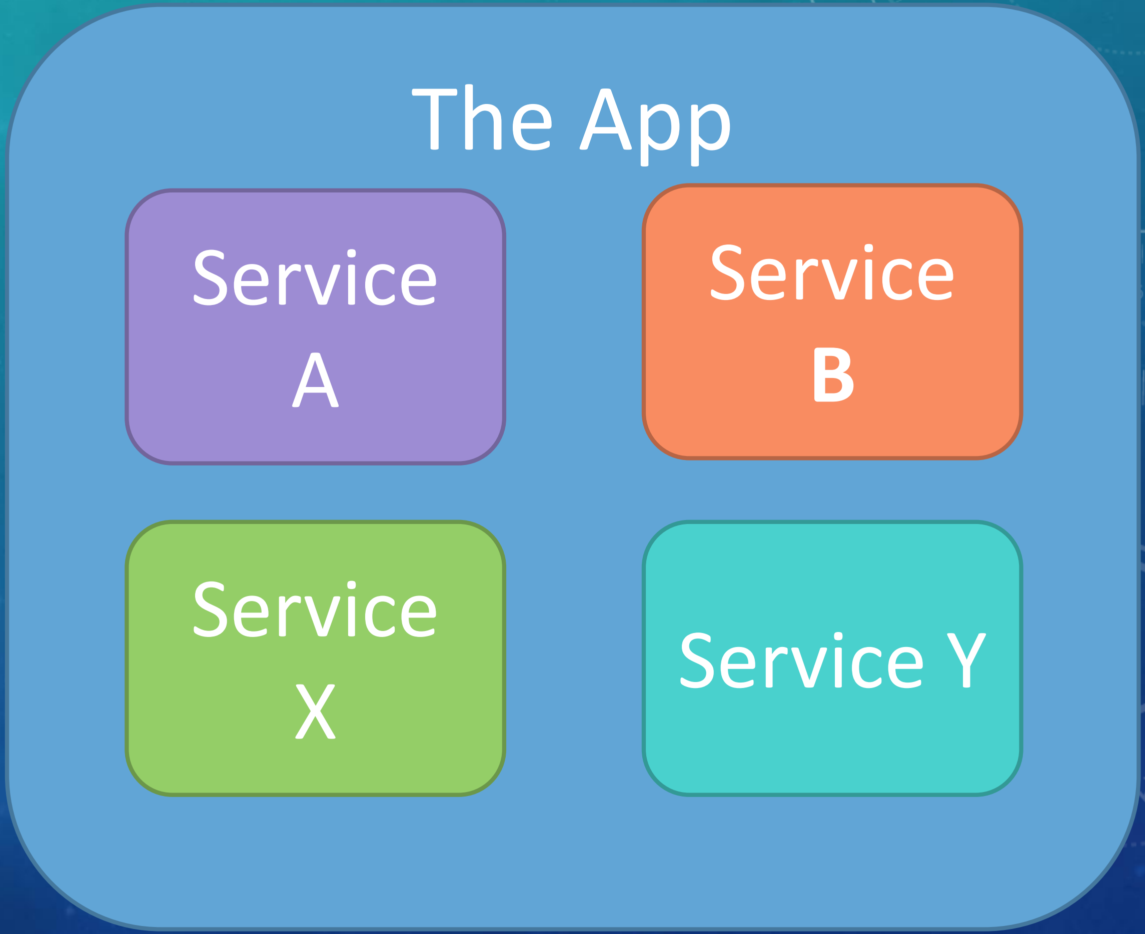


- Adding new features are difficult.
- Achieving Reliability Becomes Extremely Challenging

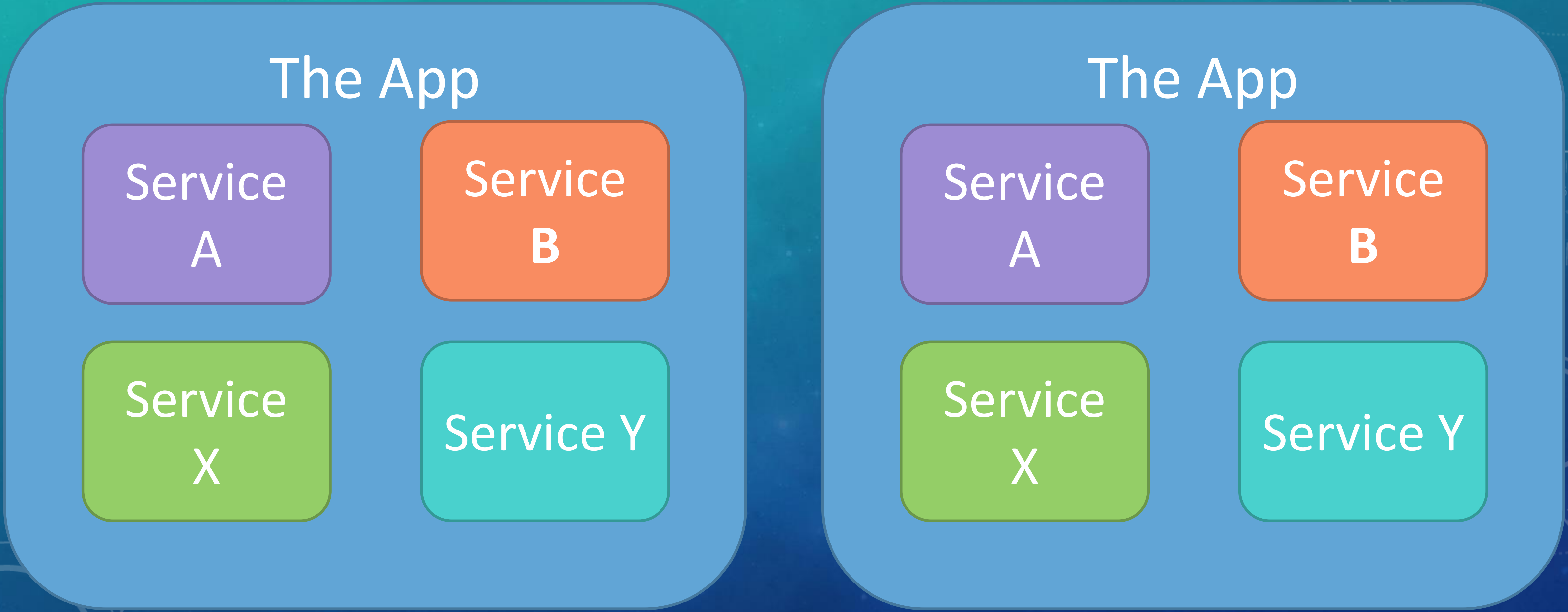
MONOLITHS ARE DIFFICULT TO SCALE



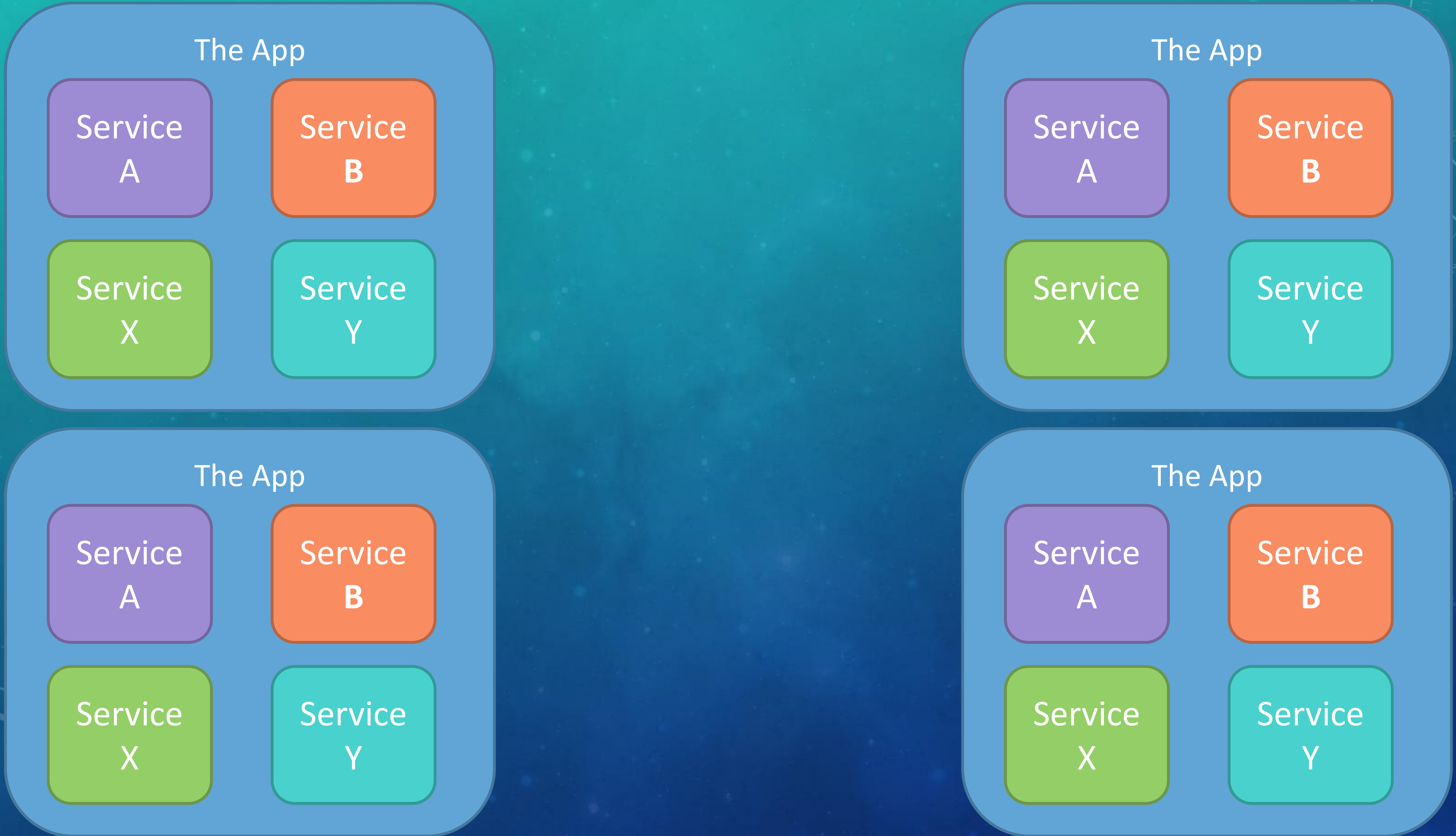
DIFFICULT TO SCALE



DIFFICULT TO SCALE



DIFFICULT TO SCALE



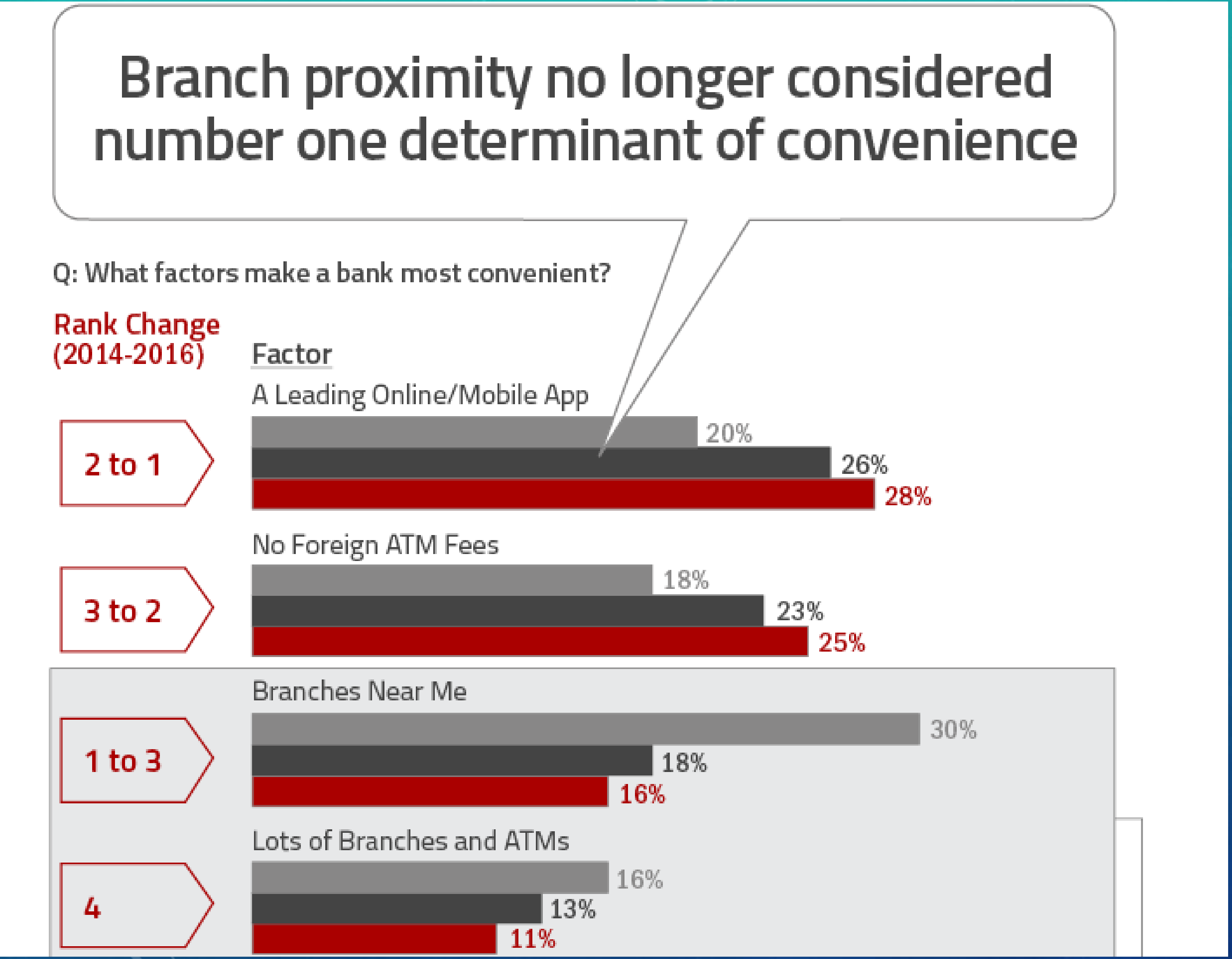
DEPLOYING A MONOLITH

- “Once your application has become a large, complex monolith, your development organization is probably in a world of pain. Any attempts at agile development and delivery will flounder. One major problem is that the application is overwhelmingly complex. It’s simply too large for any single developer to fully understand. As a result, fixing bugs and implementing new features correctly becomes difficult and time consuming. What’s more, this tends to be a downwards spiral. If the codebase is difficult to understand, then changes won’t be made correctly. You will end up with a monstrous, incomprehensible big ball of mud.”
  - - Chris Richardson –founder Cloud Foundry (https://www.nginx.com/blog/introduction-to-microservices/)

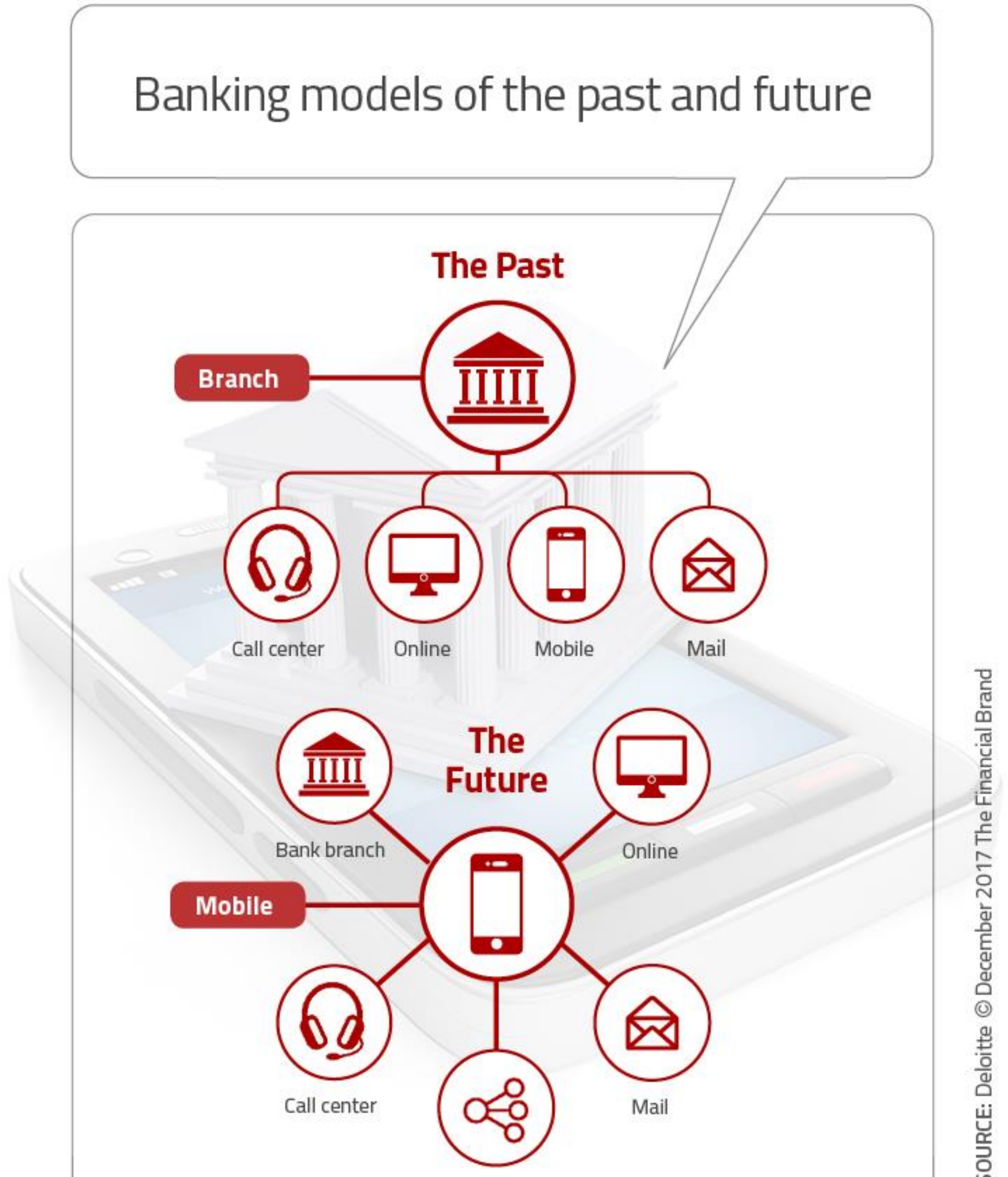
Slow Speed to Market

EXAMPLE OF CHANGING MARKET DEMAND

MOST CONVENIENT FEATURE?



Source: https://thefinancialbrand.com/69180/2018-top-banking-trends-predictions-outlook-digital-fintech-data-ai-cx-payments-tech/all/


Banking models of the past and future
The Past
Branch
Call center
Online
Mobile
Mail
The Future
Bank branch
Online
Mobile
Call center
Mail
SOURCE: Deloitte © December 2017 The Financial Brand

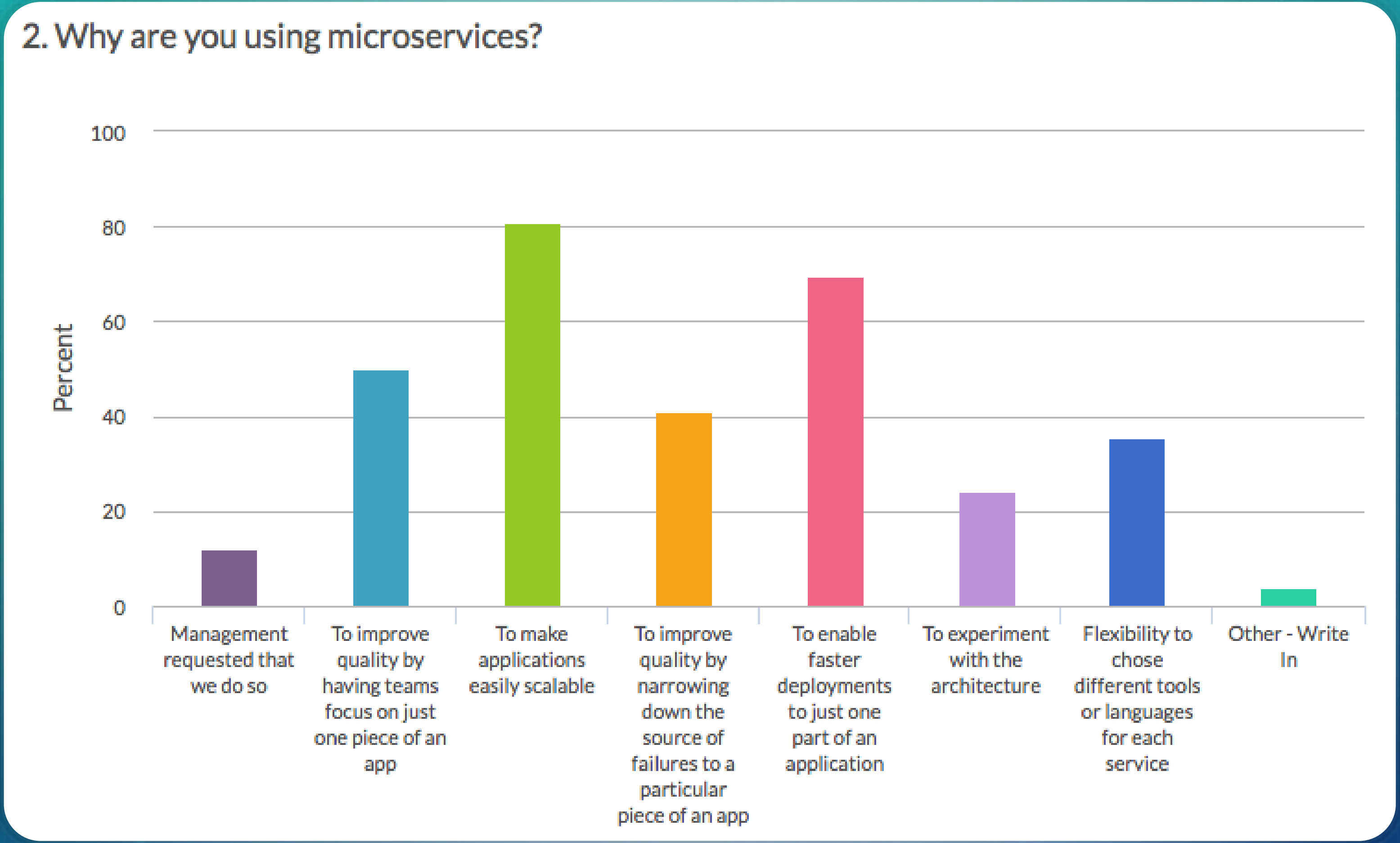
- "2018 will see banks getting much more serious about digitizing their current analog processes with a particular focus on their commercial customers and on mobile. Loan processing, account opening, service subscriptions, problem resolution and one-to-many payments are all examples of current processes that are ripe to be reimagined in order to gain speed, efficiency, and scale."

- – **Chris Nichols**, Chief Strategy Officer at CenterState Bank

- Source: https://thefinancialbrand.com/69180/2018-top-banking-trends-predictions-outlook-digital-fintech-data-ai-cx-payments-tech/all/

Tackling the Problem of the Giant Monolith

Why Microservices?

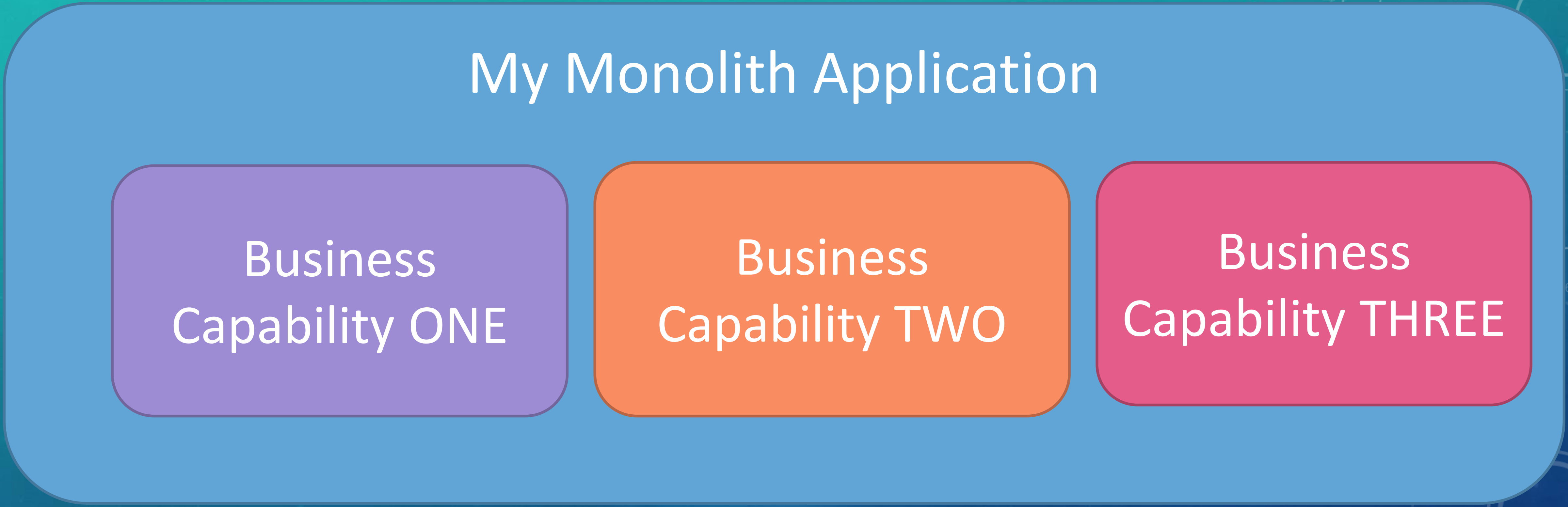


- "Those who were using microservices listed scalability and faster deployments as the leading factors, followed by the ability to improve quality by having teams focusing on smaller parts of the app. "
- - Source: DZone Research: Microservices Priorities and Trends

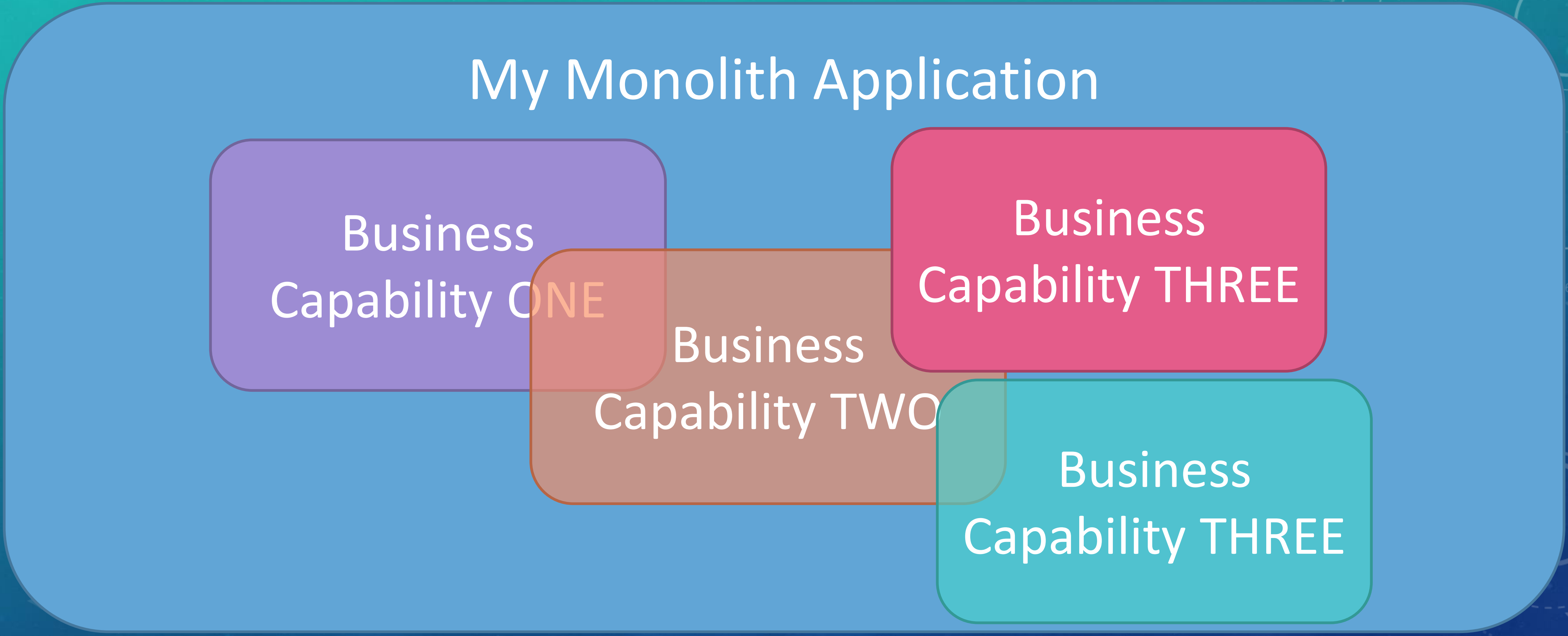
WHAT IS A MICROSERVICE?

- The microservice architecture uses services as the unit of modularity.

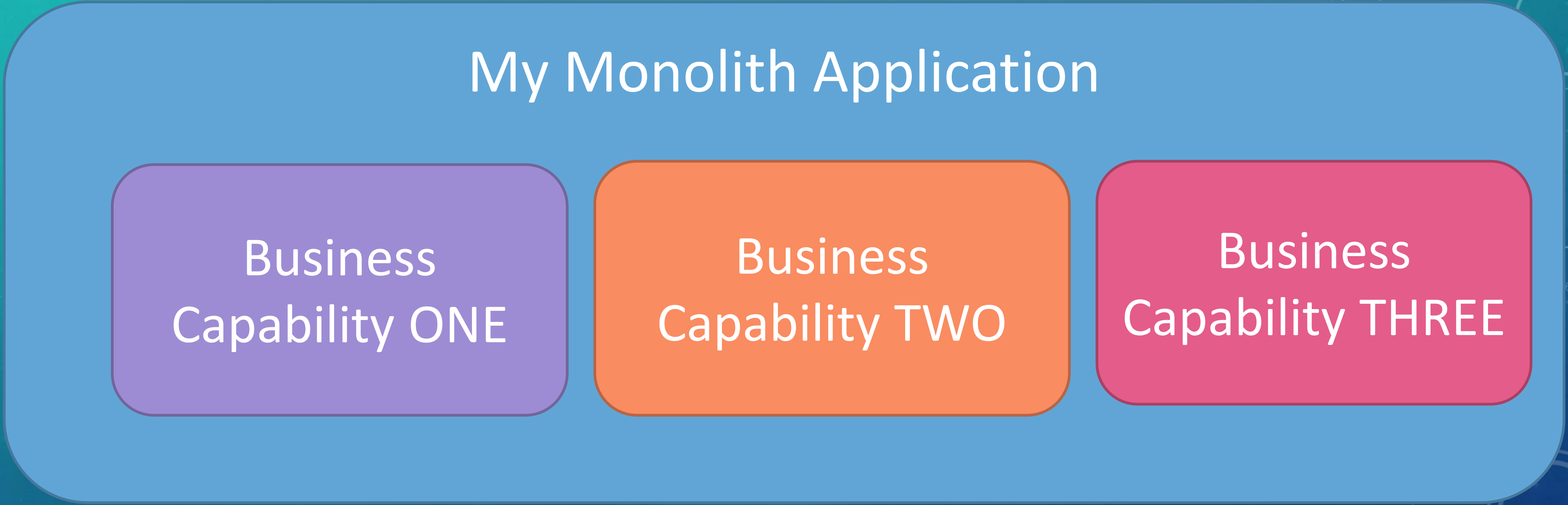
MONOLITH VS MICROSERVICE



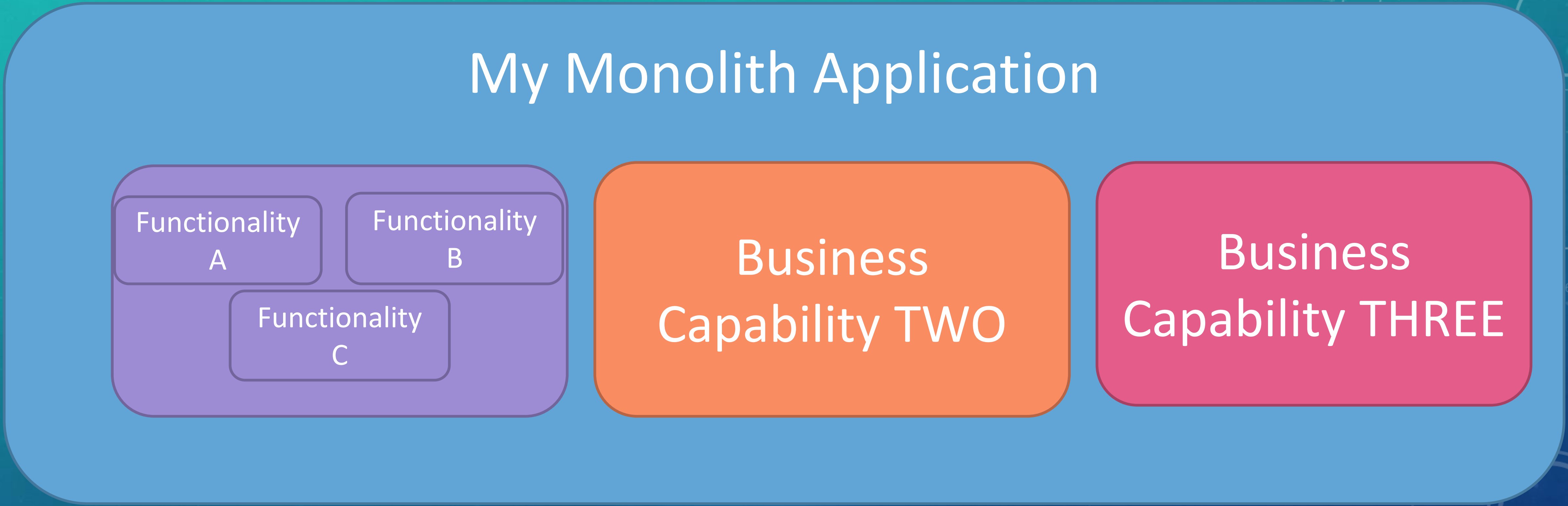
MONOLITH VS MICROSERVICE



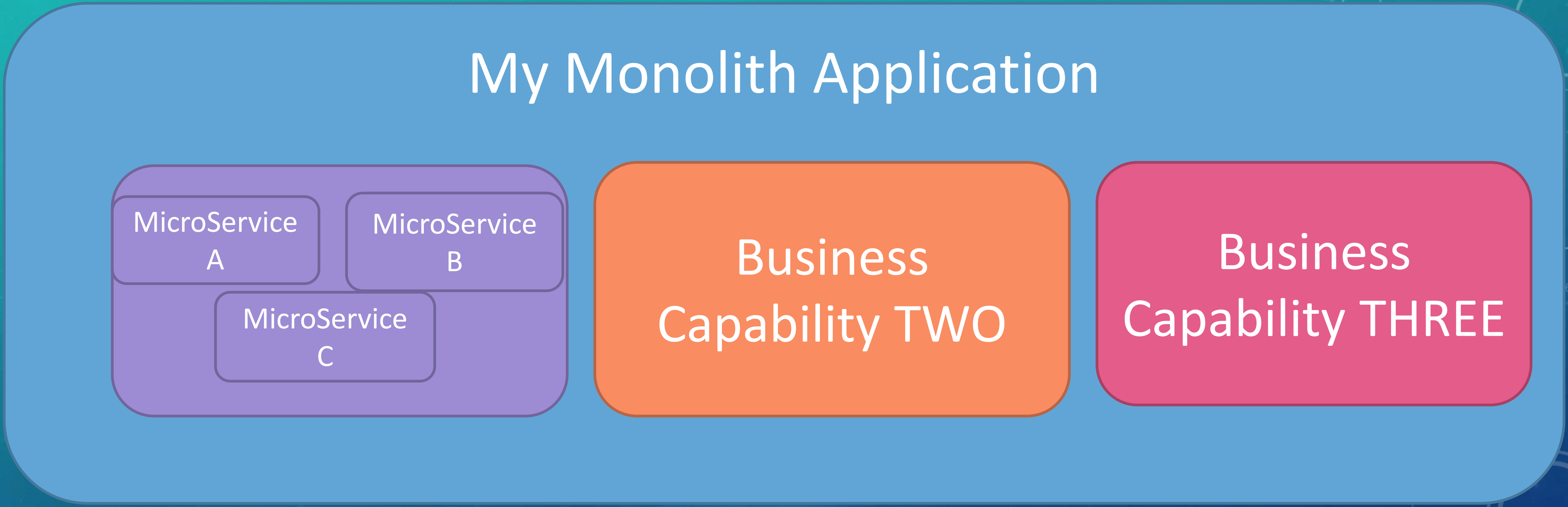
MONOLITH VS MICROSERVICE



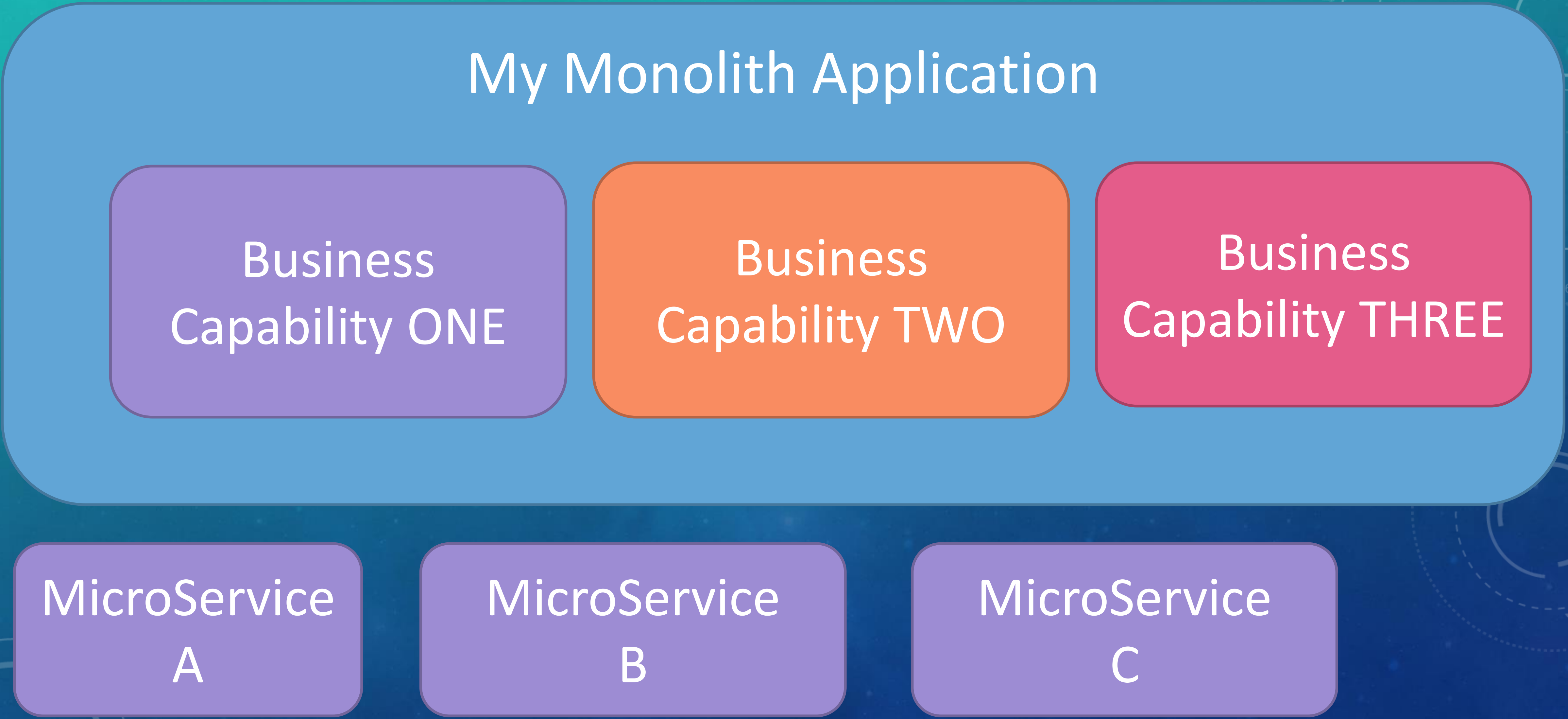
MONOLITH VS MICROSERVICE



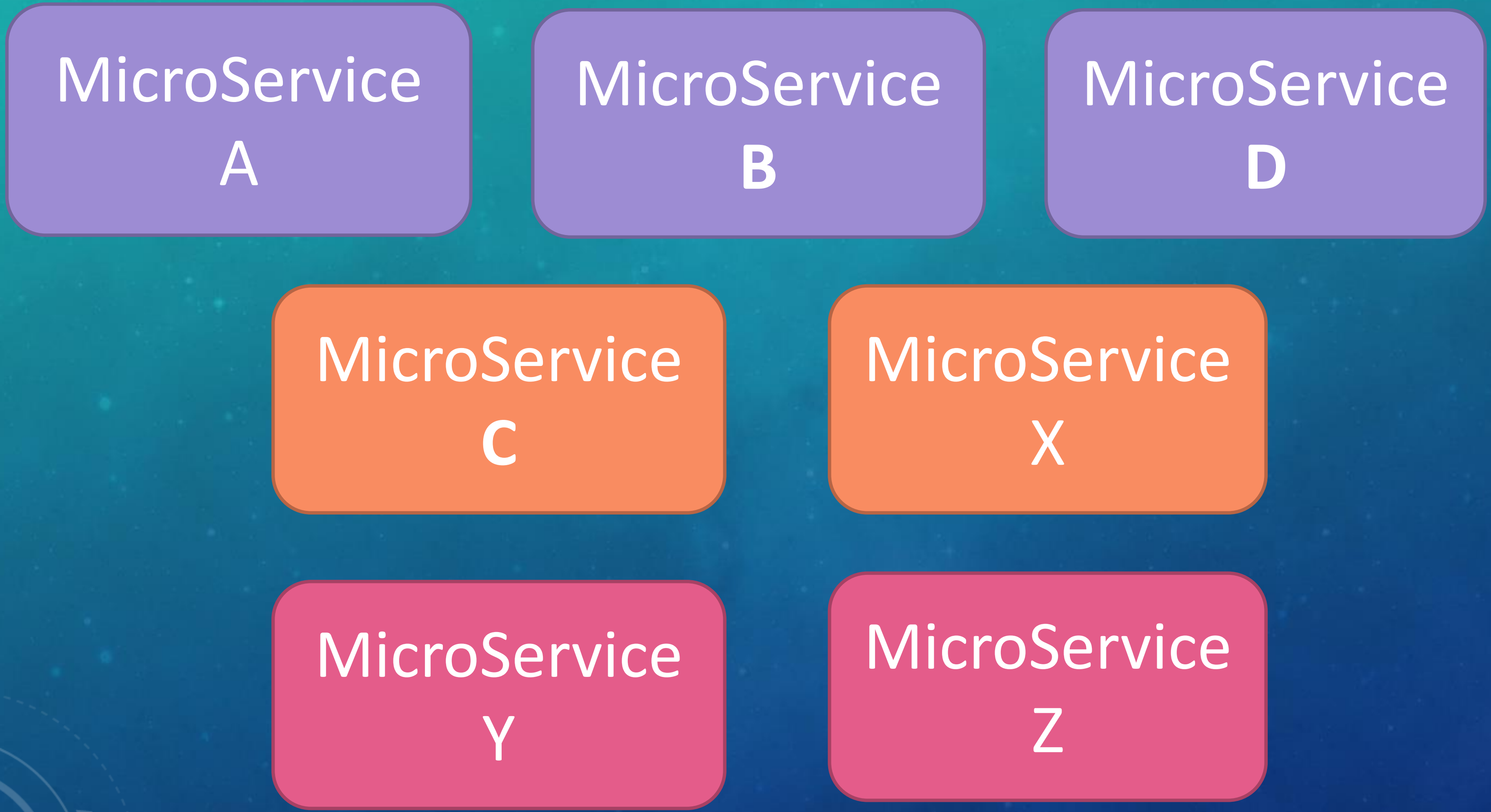
MONOLITH VS MICROSERVICE



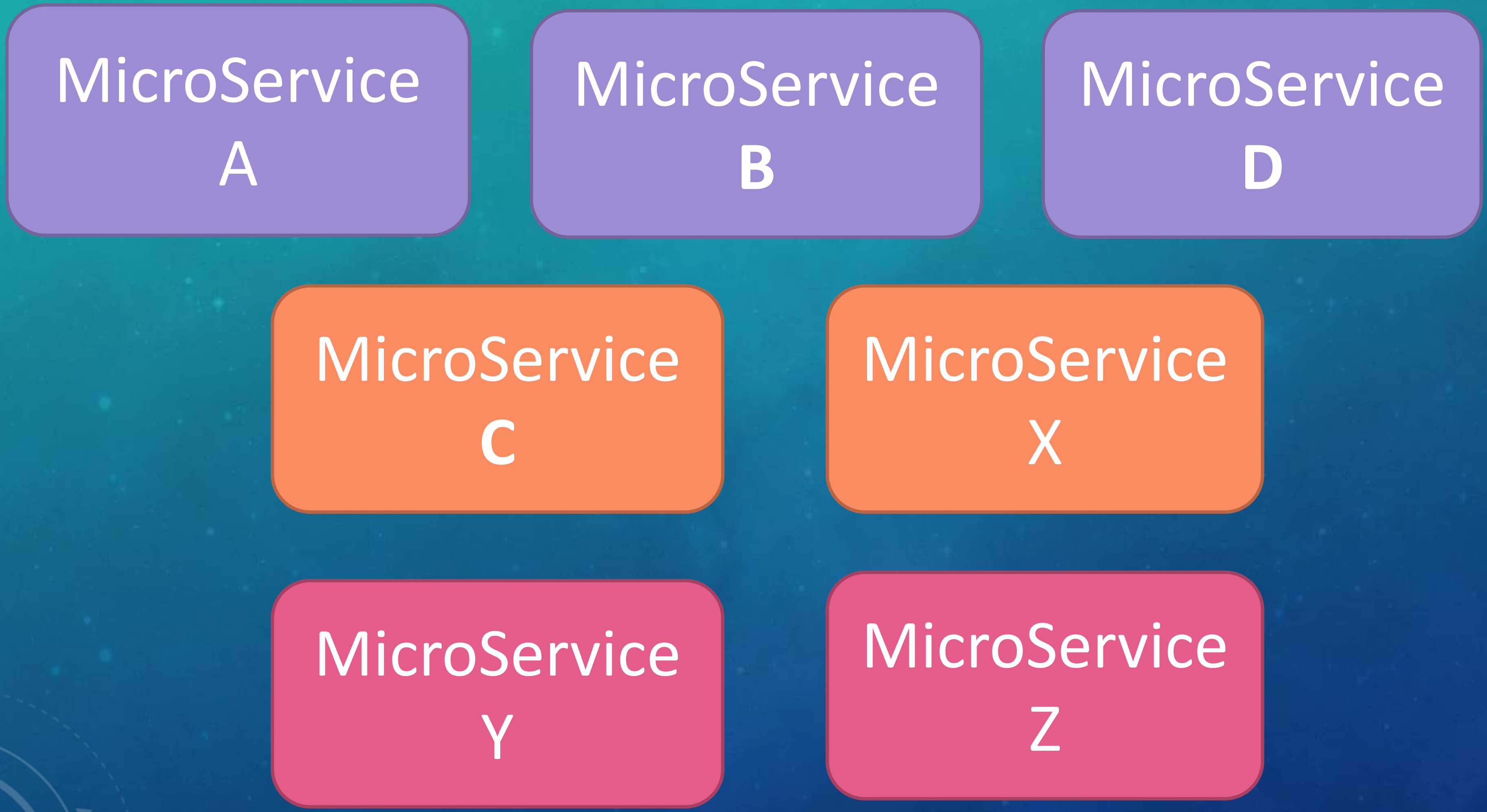
MONOLITH VS MICROSERVICE



MONOLITH VS MICROSERVICE



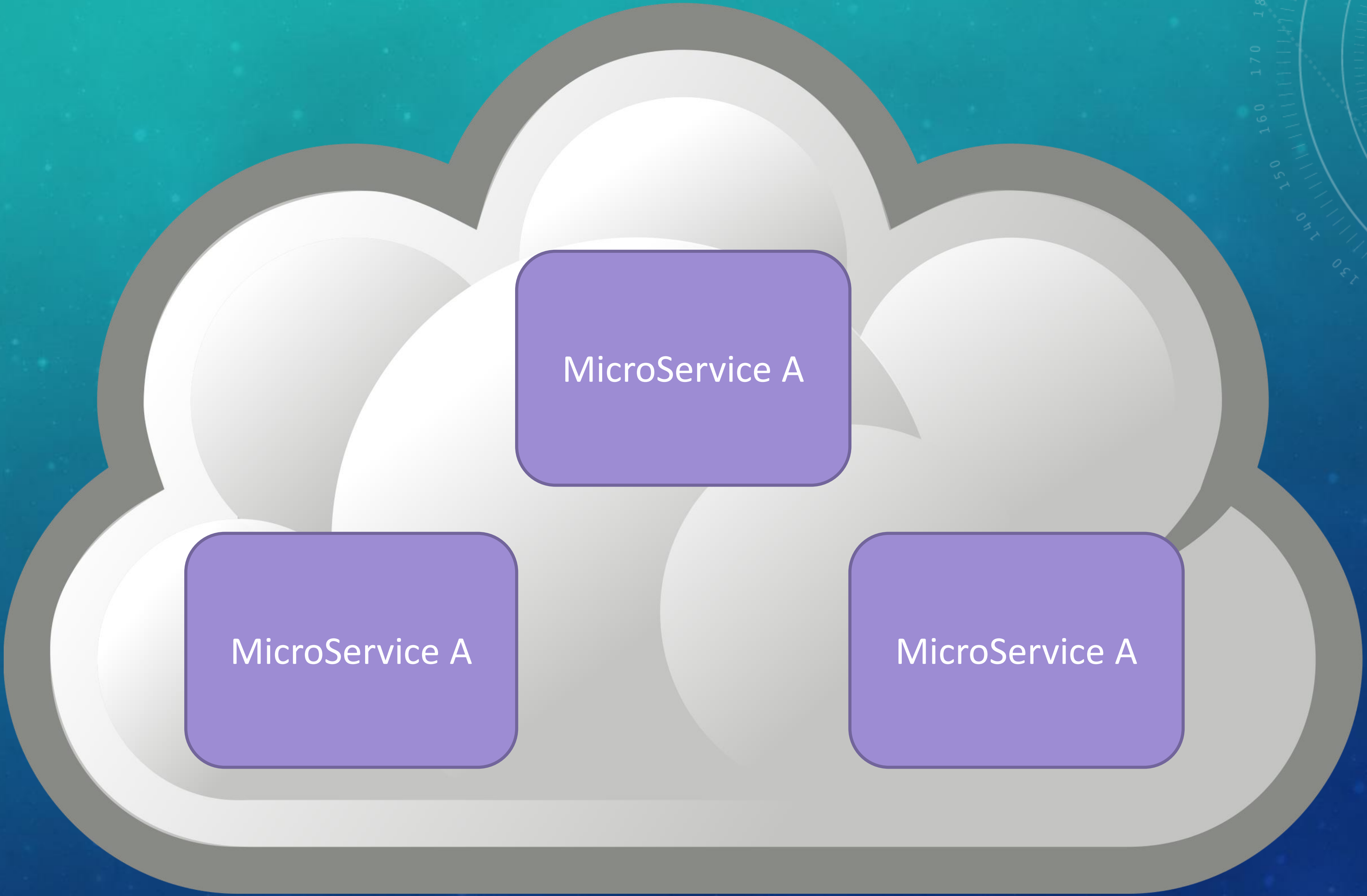
MICROSERVICES A PLENTY



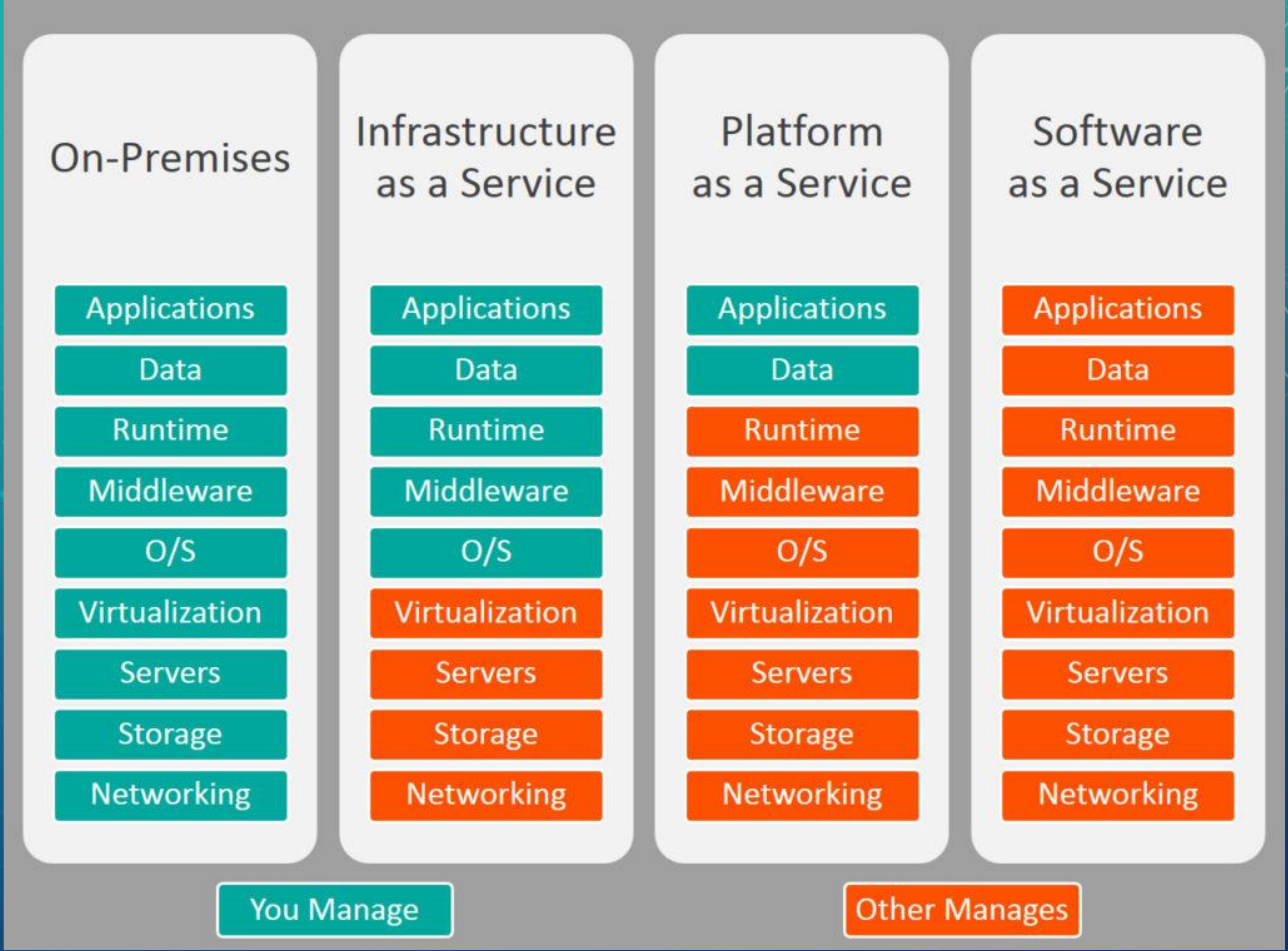
MICROSERVICES A PLENTY

MicroService A
MicroService B
MicroService D
MicroService
MicroService
SERVER

NEEDS INFRASCTRUCTURE



There is no cloud
it's just someone else's computer
stickermule



On-Premises	Infrastructure as a Service	Platform as a Service	Software as a Service
🟩 Applications	🟩 Applications	🟩 Applications	🟧 Applications
🟩 Data	🟩 Data	🟩 Data	🟧 Data
🟩 Runtime	🟩 Runtime	🟧 Runtime	🟧 Runtime
🟩 Middleware	🟩 Middleware	🟧 Middleware	🟧 Middleware
🟩 O/S	🟩 O/S	🟧 O/S	🟧 O/S
🟩 Virtualization	🟧 Virtualization	🟧 Virtualization	🟧 Virtualization
🟩 Servers	🟧 Servers	🟧 Servers	🟧 Servers
🟩 Storage	🟧 Storage	🟧 Storage	🟧 Storage
🟩 Networking	🟧 Networking	🟧 Networking	🟧 Networking

🟩 You Manage

🟧 Other Manages

Source: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/

CLOUD SERVICE ENABLE MICROSERVICES

You can have it all when you want it!

- Infrastructure
- Application servers
- Databases
- Scalability
- Fail/Over
- Etc…

MICROSERVICE STRONG BOUNDARY

- A service has an impermeable boundary that is difficult to violate. As a result, the modularity of the application is much easier to preserve over time.
- Microservices represent the business capabilities which is why the Analyst is key in helping to identify them.

MICROSERVICES NOT TIGHTLY COUPLED

- A key characteristic of the microservice architecture is that the services are loosely coupled. Not highly dependent on the internals of each other.

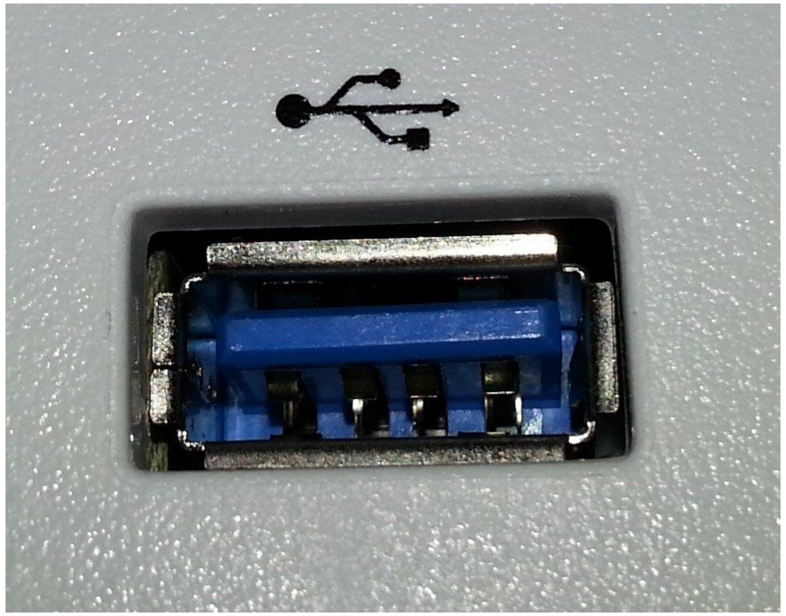
TALKING TO MICROSERVICES VIA APIS

- "Many companies are using microservices to encapsulate key capabilities within the organization in a way that is scalable and reliable. Microservices represent the important functional elements of your company's IT. But that is just part of the story. You also need the ability to expose these capabilities in a way that makes it easy to solve current business challenges. And that is where APIs come in."

- - Mike Amundsen from "Microservices, APIs and Innovation: The Power of APIs"
- Source: https://thenewstack.io/microservices-apis-and-innovation-the-power-of-apis/

WHAT IS AN API

APPLICATION PROGRAMMING INTERFACE



SO WHAT IS A AN API REALLY?

- Just a way to send information between applications
- Typically using HTTP and the message format is usually JSON

http://

{JSON}

JavaScript Object Notation

XML VS JSON

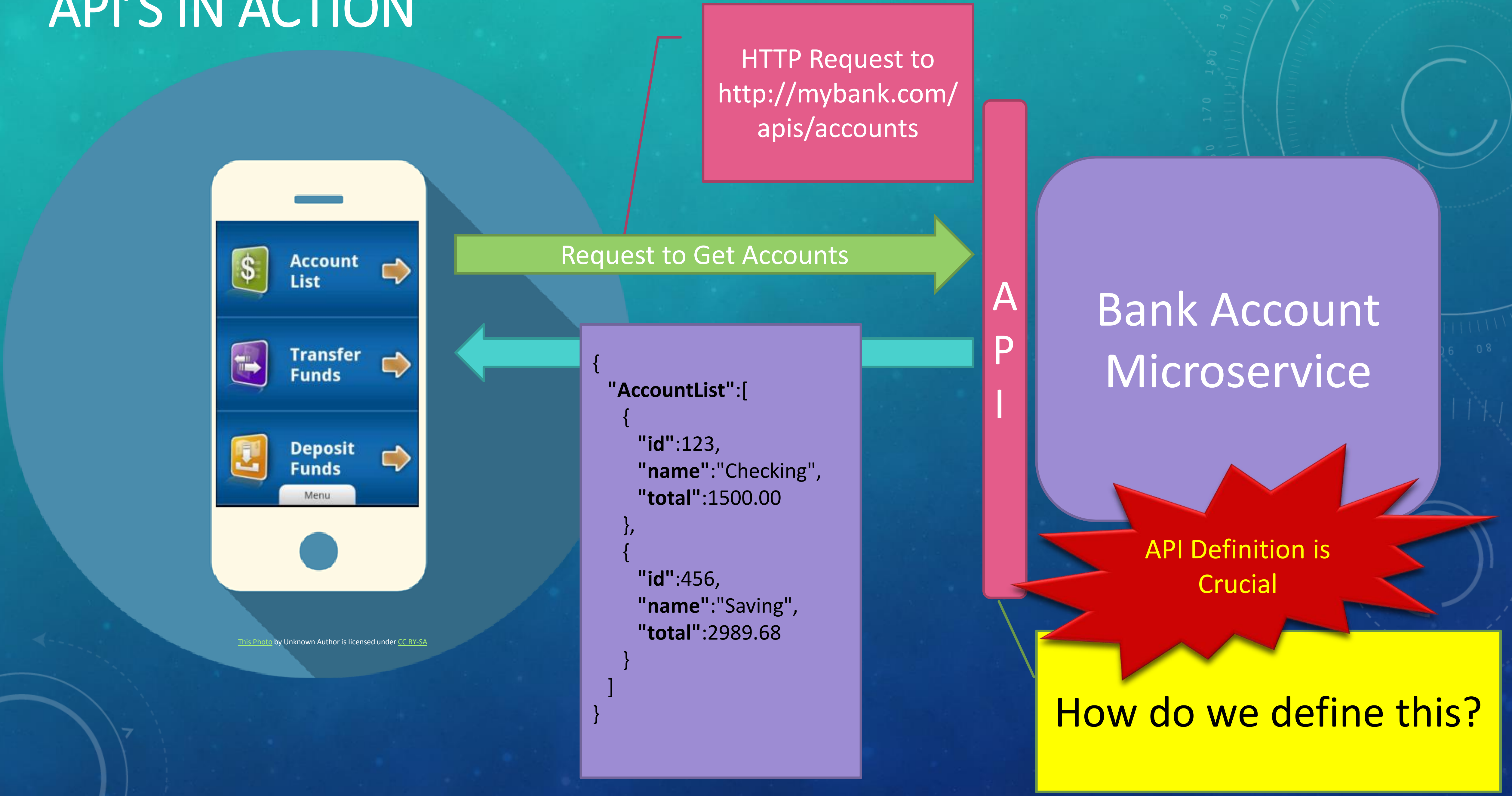
XML

```
<empinfo>
    <employees>
        <employee>
            <name>James Kirk</name>
            <age>40></age>
        </employee>
        <employee>
            <name>Jean-Luc Picard</name>
            <age>45</age>
        </employee>
        <employee>
            <name>Wesley Crusher</name>
            <age>27</age>
        </employee>
    </employees>
</empinfo>
```

JSON

```
{   "empinfo" :
        {
                "employees" :  [
                {
                    "name" : "James Kirk",
                    "age" : 40,
                } ,
                {
                    "name" : "Jean-Luc Picard",
                    "age" : 45,
                } ,
                {
                    "name" : "Wesley Crusher",
                    "age" : 27,
                }
                                ]
        }
    }
```

API’S IN ACTION


HTTP Request to http://mybank.com/apis/accounts
Account List
Transfer Funds
Deposit Funds
Menu
Request to Get Accounts
A P I
Bank Account Microservice
API Definition is Crucial
How do we define this?

```
{
  "AccountList":[
    {
      "id":123,
      "name":"Checking",
      "total":1500.00
    },
    {
      "id":456,
      "name":"Saving",
      "total":2989.68
    }
  ]
}
```

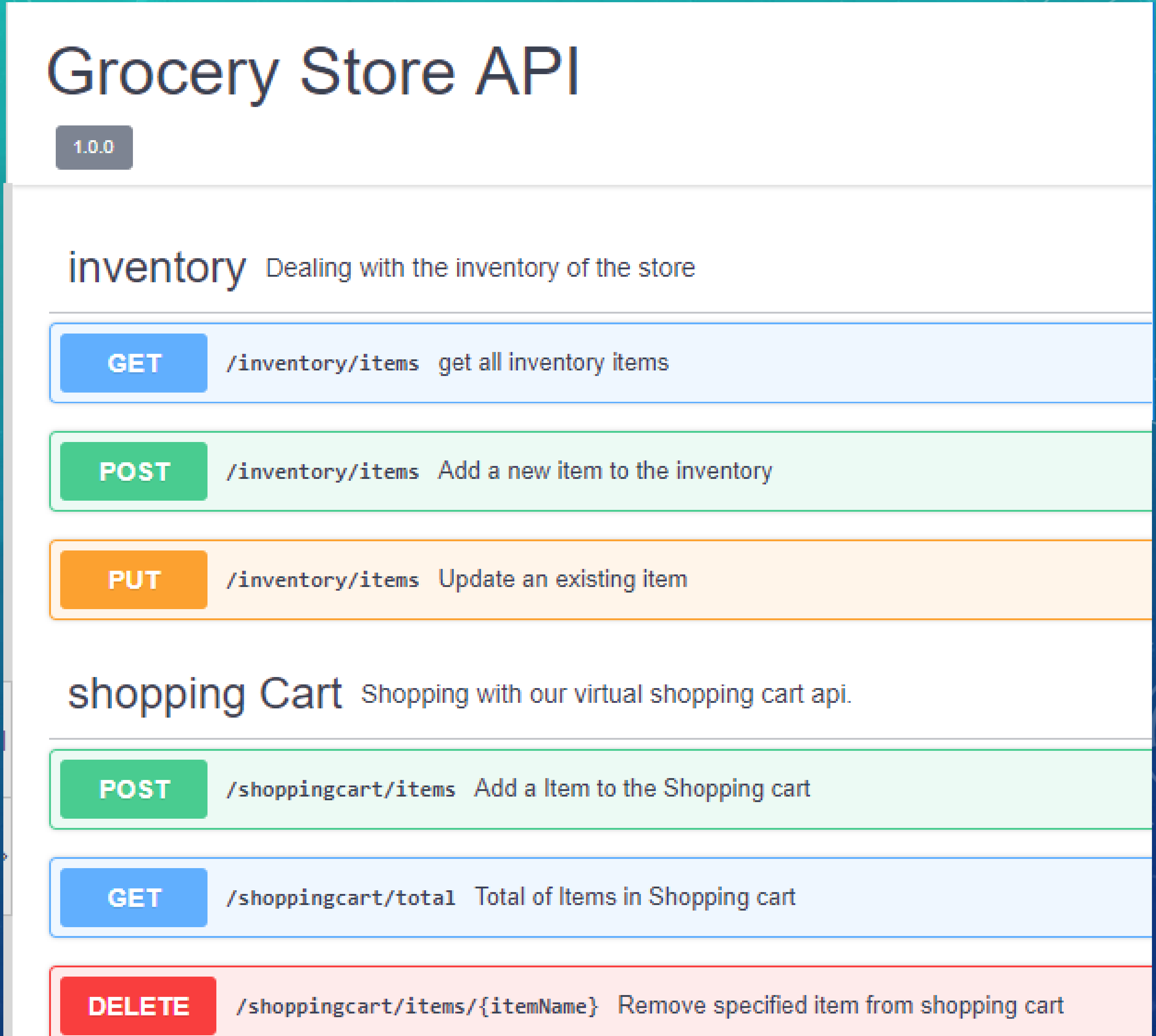
GET YOUR SWAGGER ON!

SWAGGER

OPEN {API} INITIATIVE

Specification backed Documentation For Your API

EXAMPLE SWAGGER / OAS DOCUMENT



Grocery Store API

1.0.0

inventory Dealing with the inventory of the store

GET /inventory/items get all inventory items

POST /inventory/items Add a new item to the inventory

PUT /inventory/items Update an existing item

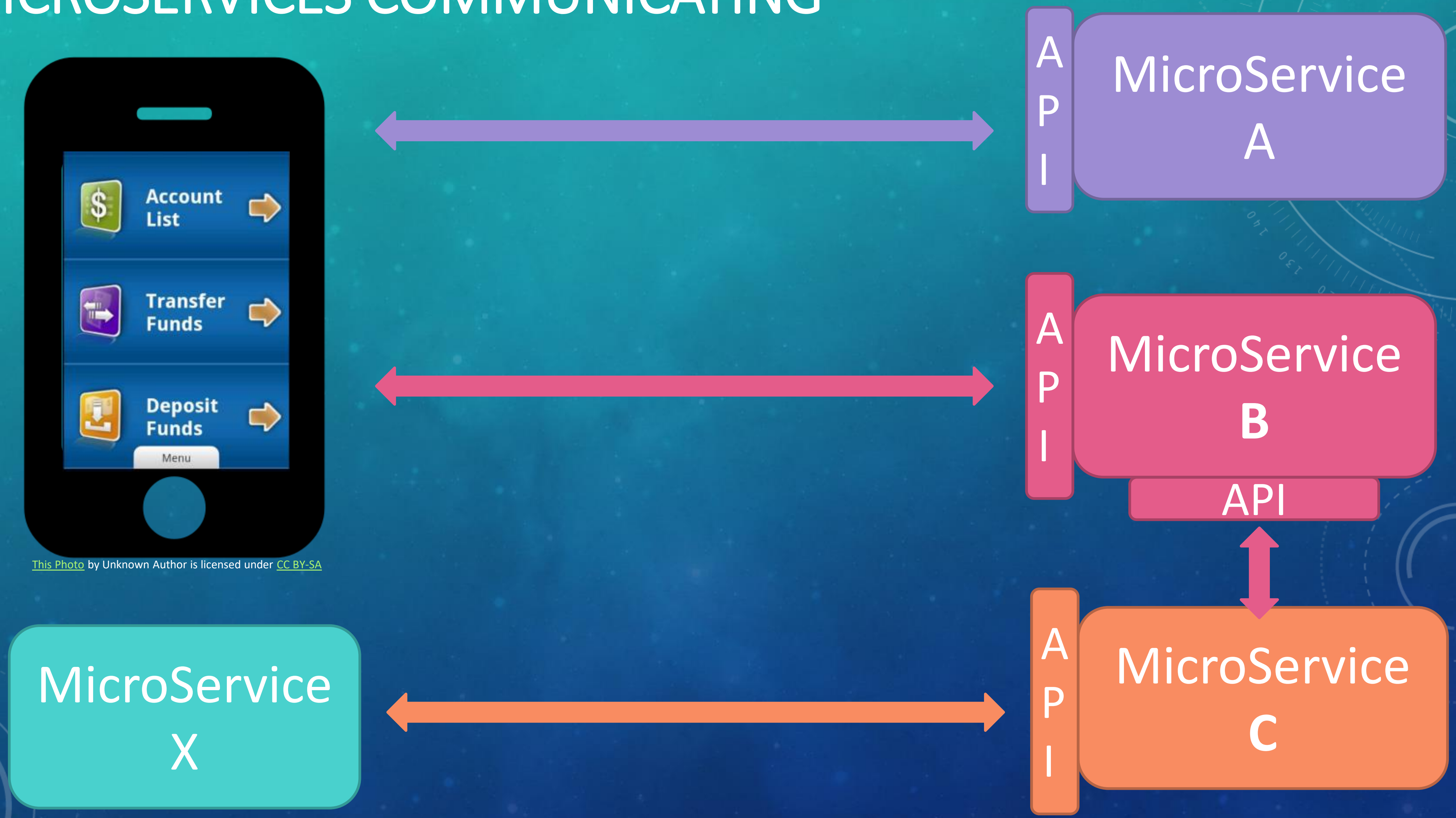
shopping Cart Shopping with our virtual shopping cart api.

POST /shoppingcart/items Add a Item to the Shopping cart

GET /shoppingcart/total Total of Items in Shopping cart

DELETE /shoppingcart/items/{itemName} Remove specified item from shopping cart

MICROSERVICES COMMUNICATING



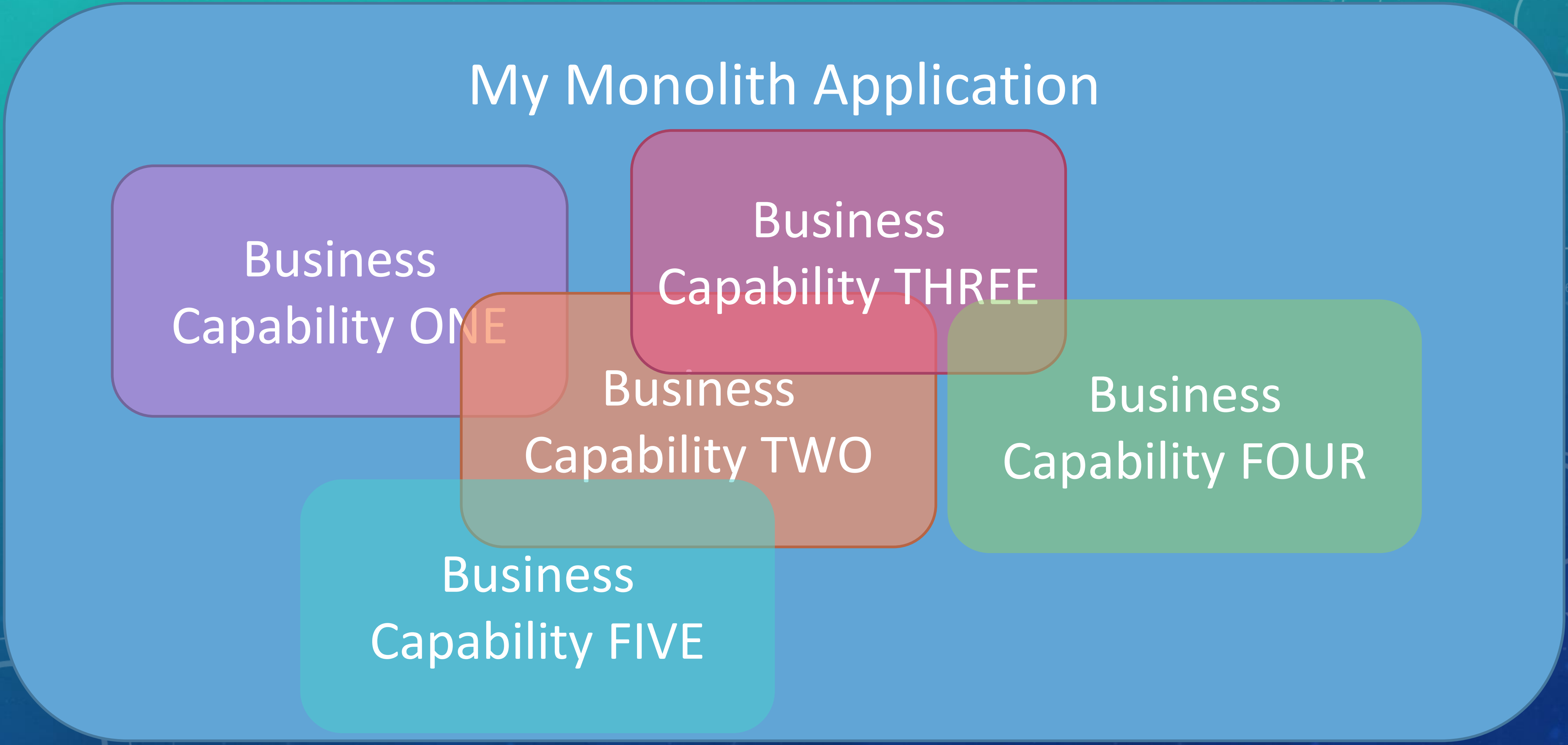
This Photo by Unknown Author is licensed under CC BY-SA

BREAKING IT DOWN?

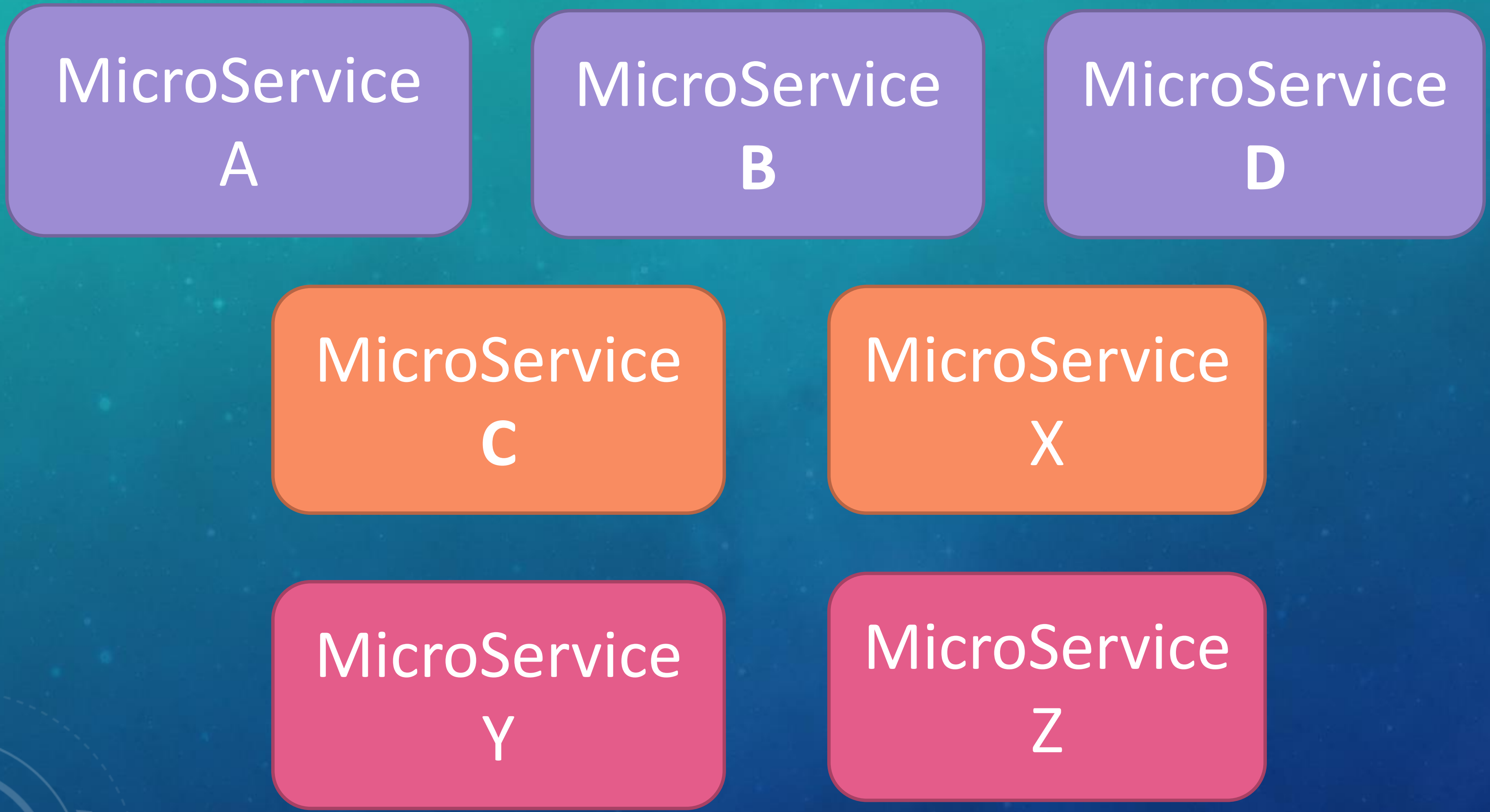
- An architectural style that functionally decomposes an application into a set of services.
- What matters is that each service has a focused, cohesive set of responsibilities

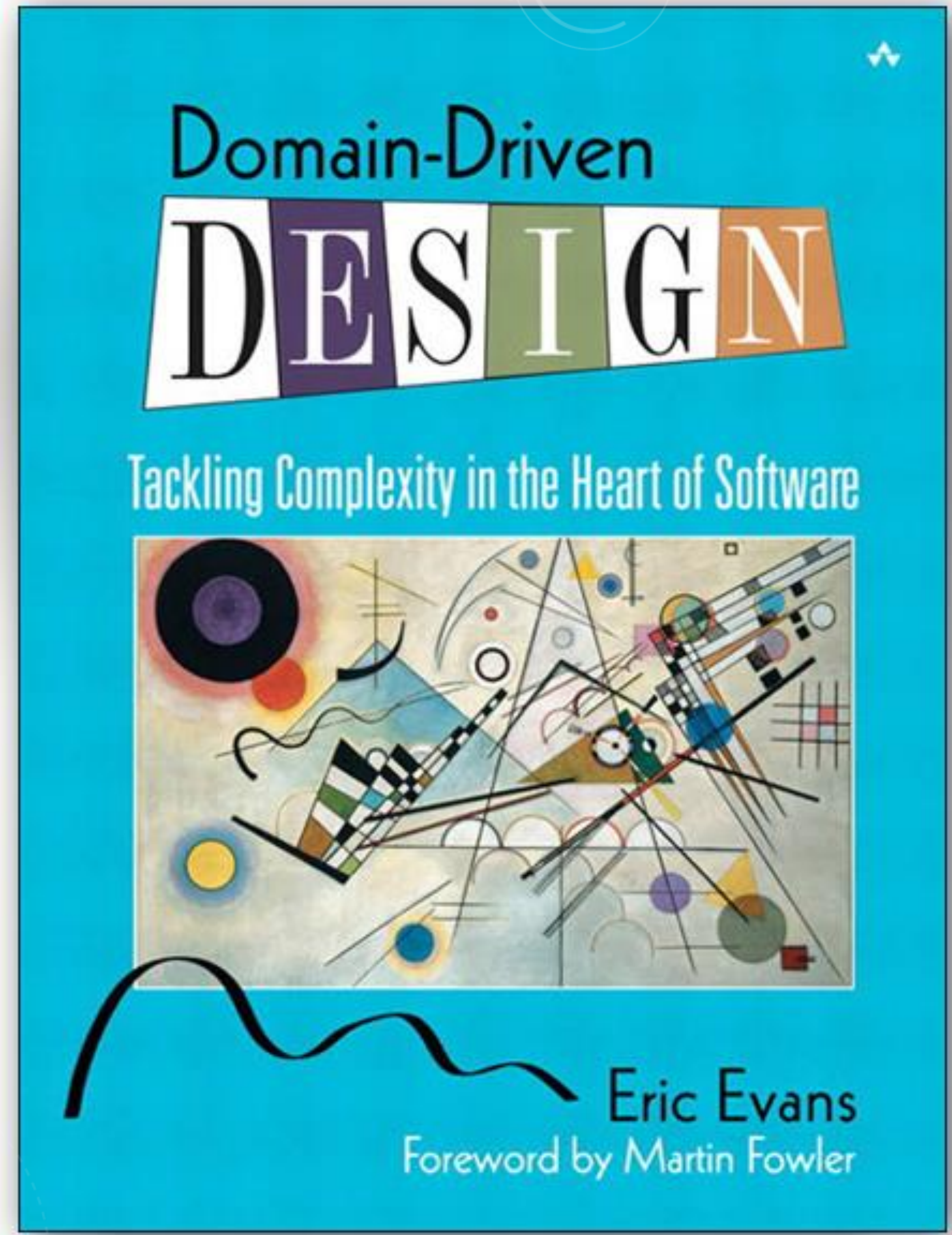
BREAK IT DOWN

BREAKING APART THE MONOLITH



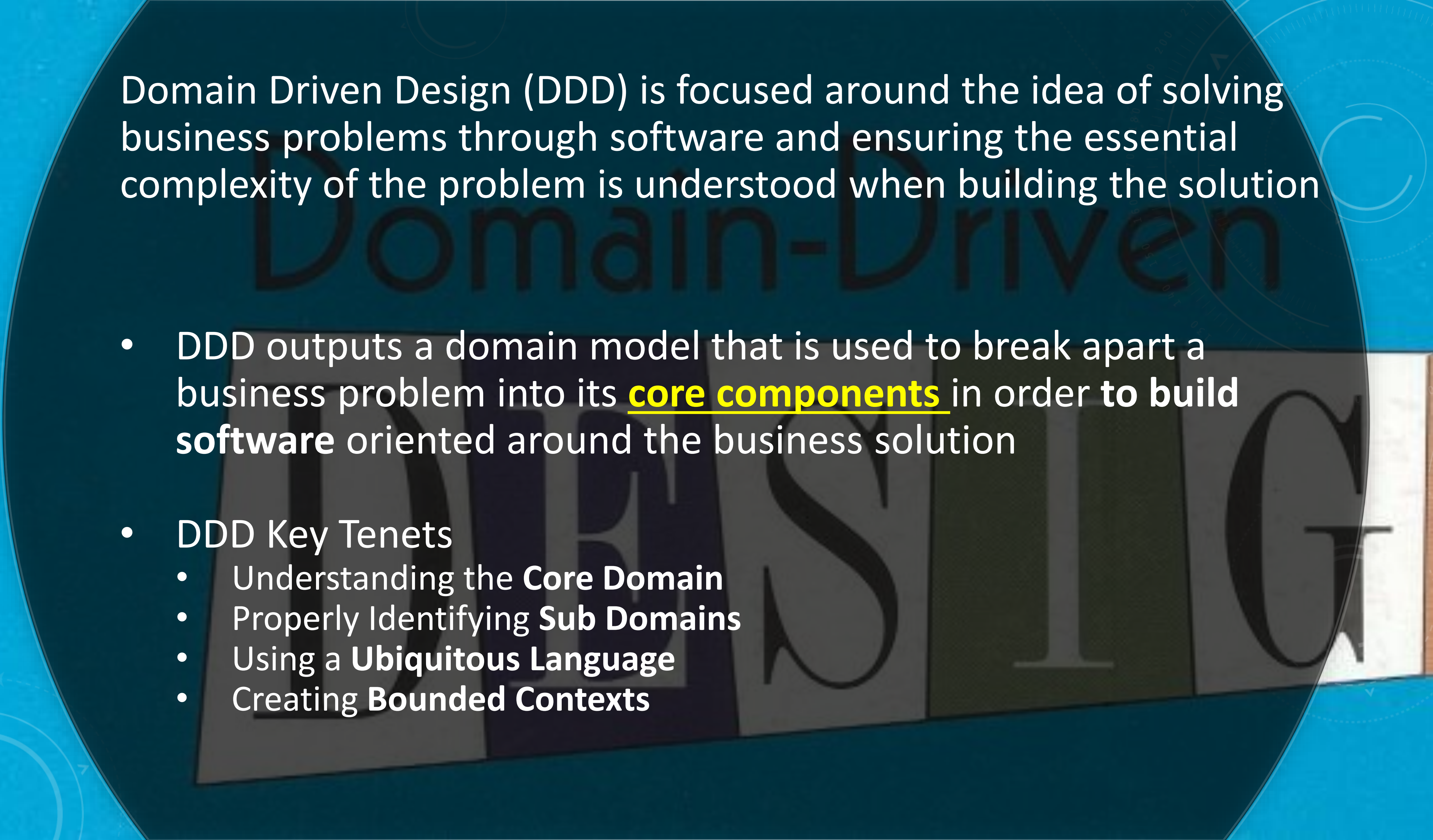
MONOLITH VS MICROSERVICE




Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

HOW DO WE BREAK IT OUT?

LEARN FROM DOMAIN DRIVEN DESIGN

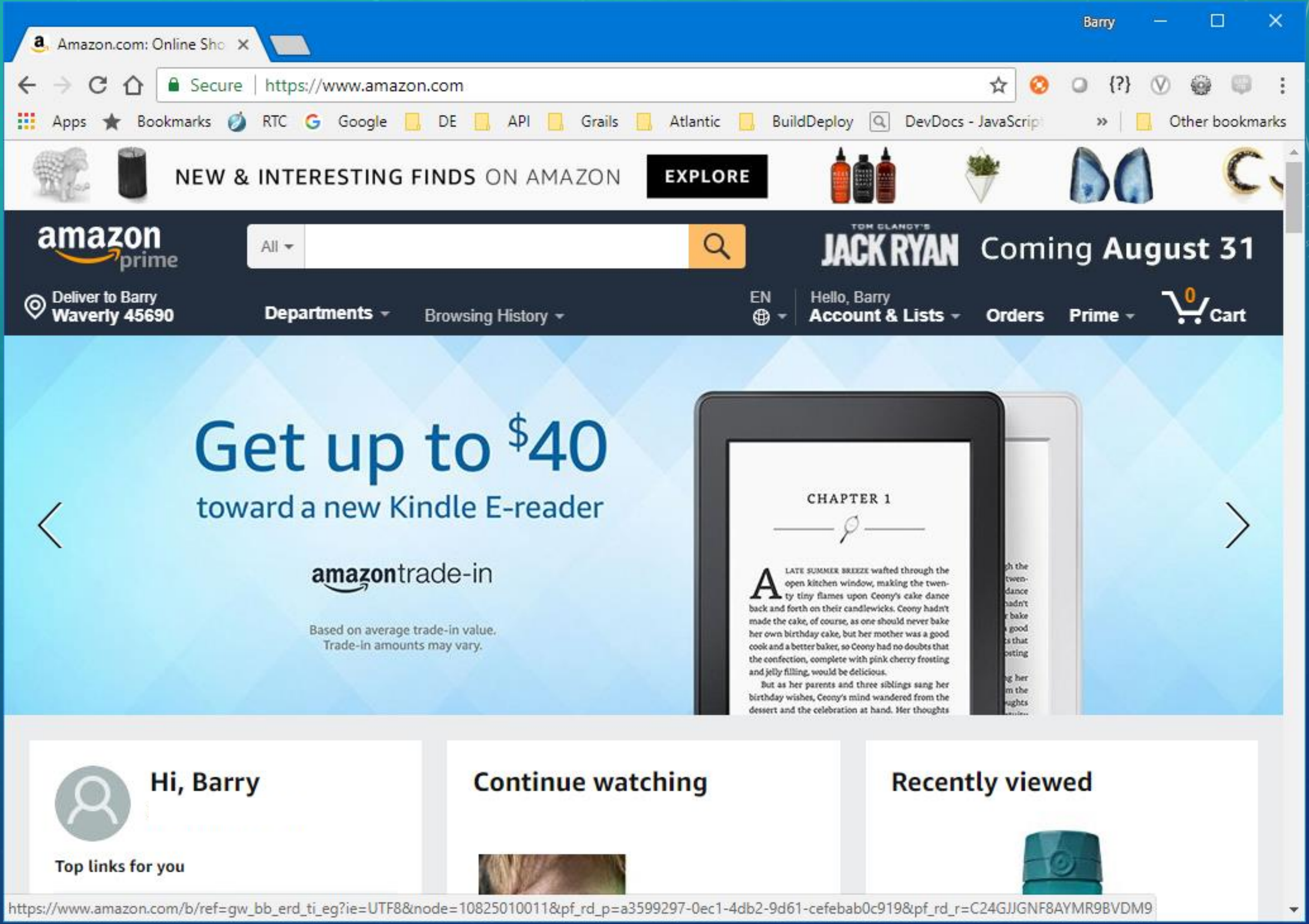


Domain Driven Design (DDD) is focused around the idea of solving business problems through software and ensuring the essential complexity of the problem is understood when building the solution

- DDD outputs a domain model that is used to break apart a business problem into its **core components** in order **to build software** oriented around the business solution

- DDD Key Tenets
  - Understanding the **Core Domain**
  - Properly Identifying **Sub Domains**
  - Using a **Ubiquitous Language**
  - Creating **Bounded Contexts**

BREAKING INTO DOMAINS



DOMAINS IN AMAZON

amazon

prime video

amazon now

2 Hour Delivery

560034

amazon music

amazon alexa

EXISTING ORGANIZATION STRUCTURE CAN DEFINE DOMAINS

- Customer Relationship Management
- Customer Assistance
- Policy Management
- Billing And Collections
- Claims Processing
- Loan Management

SUB DOMAINS IN SHOPPING SERVICE

Product Catalog Management

Order / Shopping Cart Management

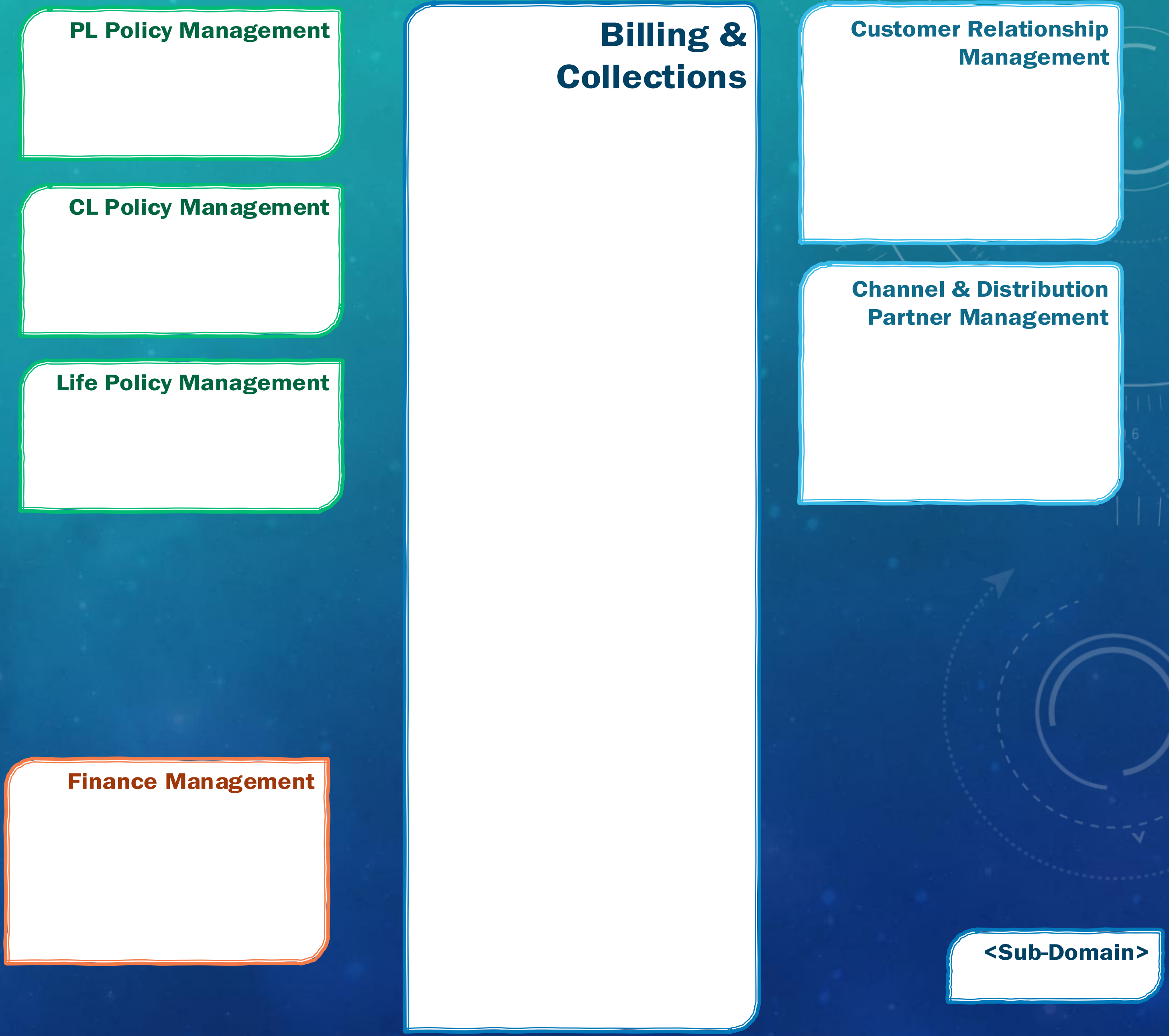
Payment Processing

Delivery / Shipping services

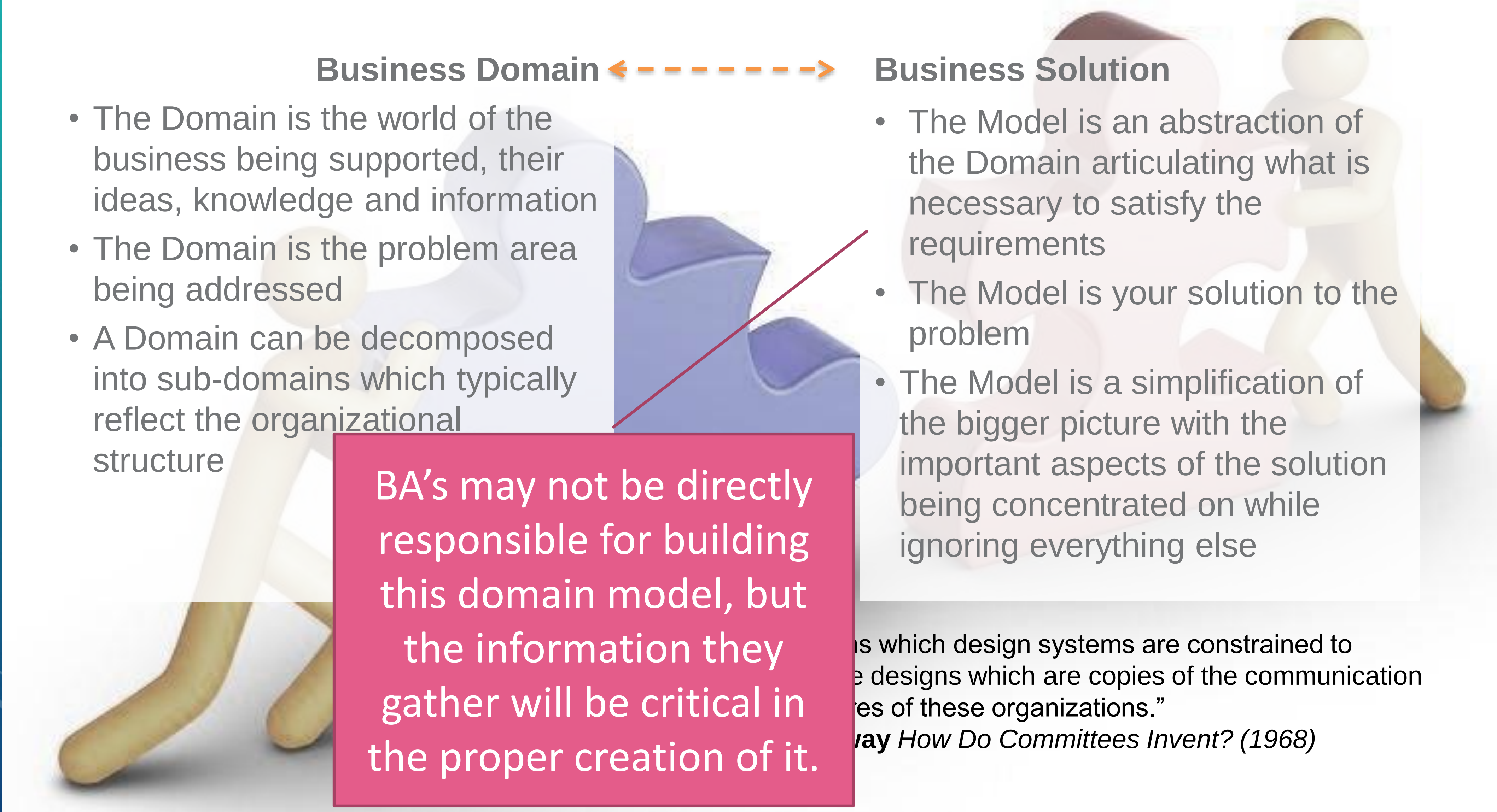
Inventory Management

IDENTIFYING SUB-DOMAINS INVOLVED IN PROBLEM

- The Business Domain Model starts by identifying the key Sub-Domain to be used as the anchor point defining the Problem space
- Through the coarse of exploring the problem, additional Sub-Domains will be identified as being involved in the Problem space
- These Sub-Domains are then added to the Domain Model to provide a comprehensive high-level view of the Problem space



SUMMARIZING DOMAINS



Business Domain ← - - - - - - - - → **Business Solution**

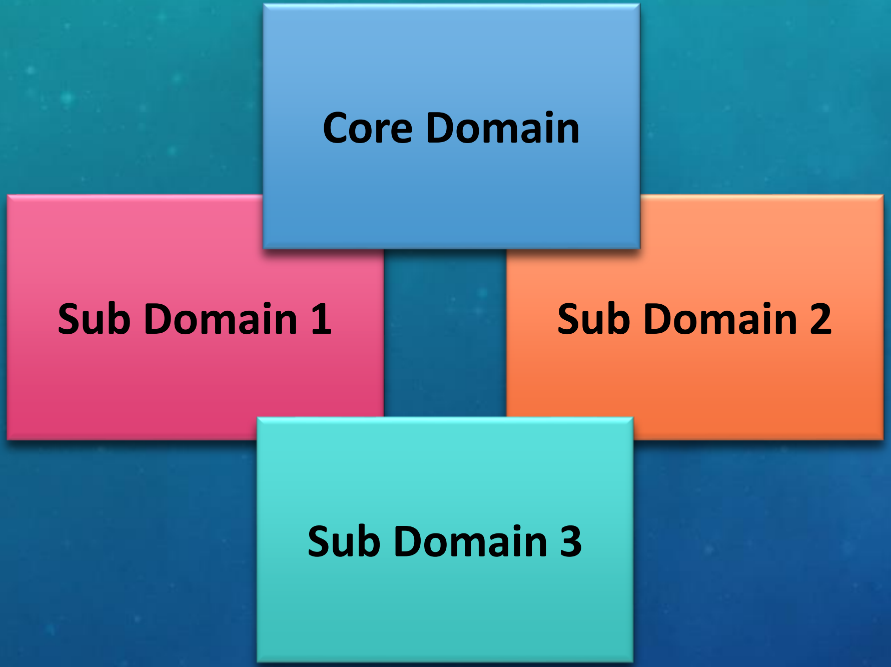
- The Domain is the world of the business being supported, their ideas, knowledge and information
- The Domain is the problem area being addressed
- A Domain can be decomposed into sub-domains which typically reflect the organizational structure

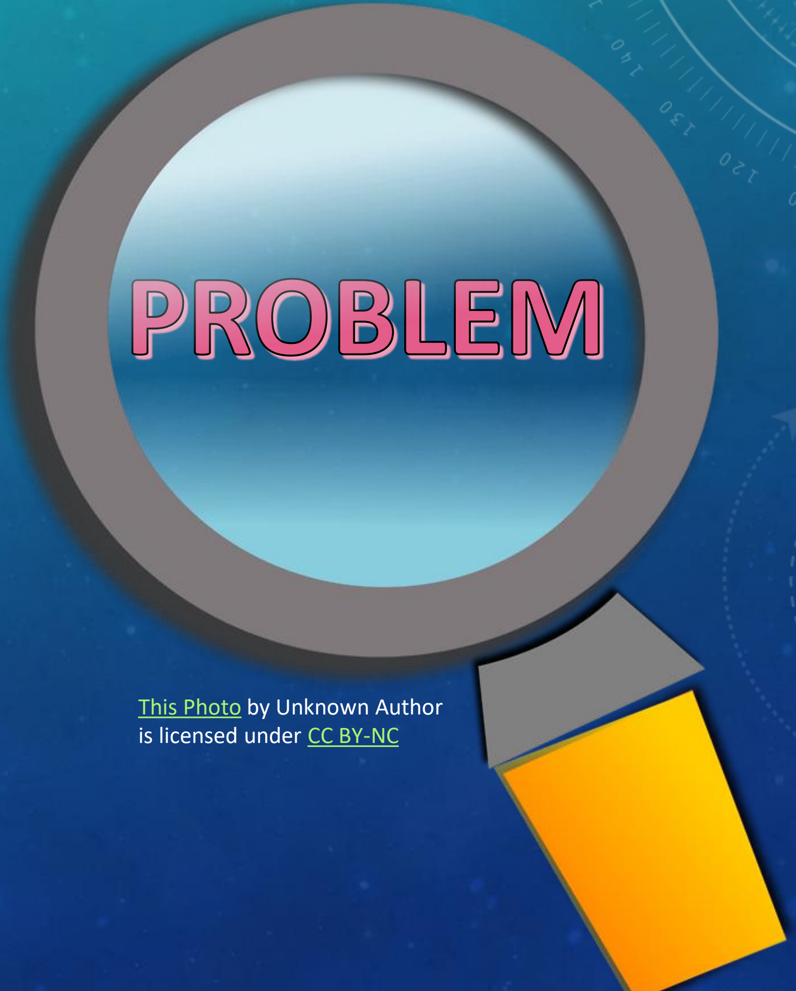
- The Model is an abstraction of the Domain articulating what is necessary to satisfy the requirements
- The Model is your solution to the problem
- The Model is a simplification of the bigger picture with the important aspects of the solution being concentrated on while ignoring everything else

BA’s may not be directly responsible for building this domain model, but the information they gather will be critical in the proper creation of it.

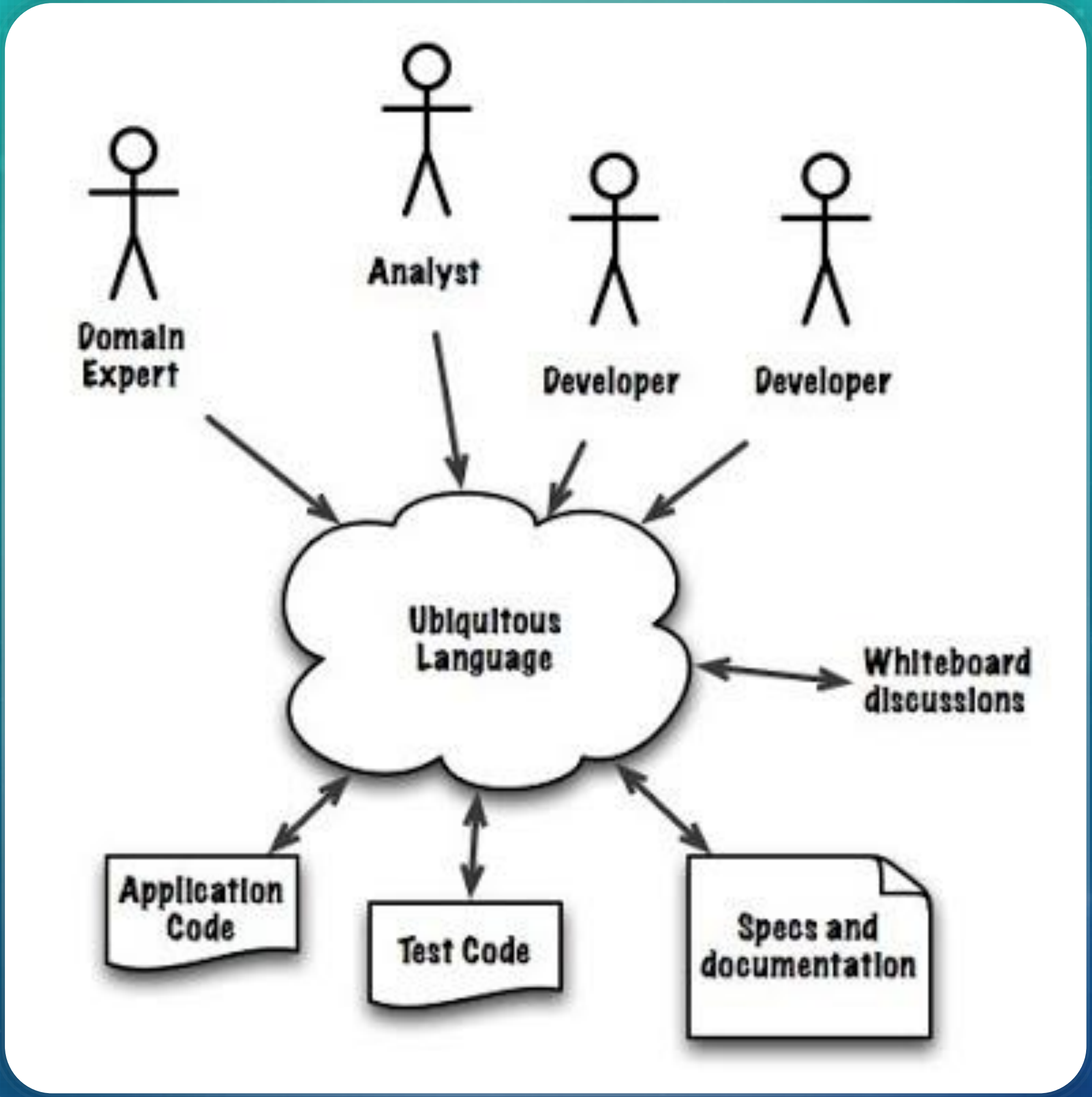
s which design systems are constrained to
e designs which are copies of the communication
es of these organizations.”
ay *How Do Committees Invent? (1968)*

Focus on what's important to solve the problem





Ubiquitous Language



UBIQUITOUS LANGUAGE: SPEAKING CLEARLY

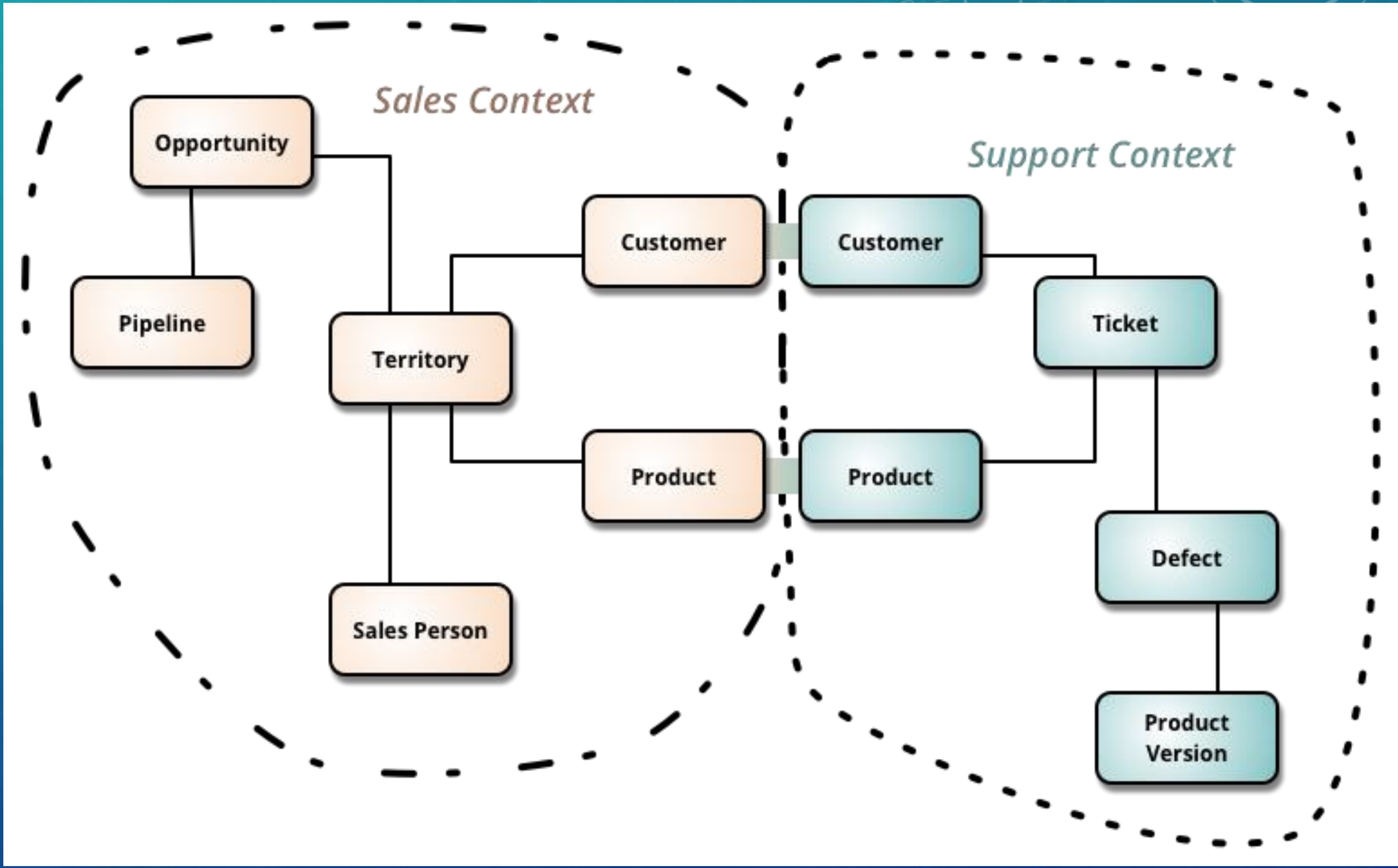
- The Business Domain experts, the Developers, and the Analyst all use the same language when discussing the domain.
- When someone says something about the domain, others should understand precisely what they mean
- Ex: A "Product" is referred to always as a "Product" within that Sub Domain. In discussions with the business as well as with the developers. Developers won't call a "Product" an "Item" in the code or anywhere else, so that the communication is always clear and focused around the business problem.

BUSINESS ANALYSTS ARE KEY

- Identifying the Ubiquitous language within a sub domain helps identify the context.
- If you are using the same words for different things, you may have jumped contexts.
- The identification of terms within the Sub Domain help you identify the boundaries of different context or what we call “Bounded Context”.
- The Bounded Context is what architects and Developers can use to effectively model the solution.

BOUNDED CONTEXTS HELP IDENTIFY BOUNDARIES AROUND CAPABILITIES

- As you try to model a larger domain, it gets progressively harder to build a single unified model. Different groups of people will use subtly different vocabularies in different parts of a large organization.
- A Bounded Context is the boundary around a model that uses a single context to define the language used for the design components
- Bounded Contexts have both unique concepts (such as a support ticket only existing in a customer support context) but also shared concepts (such as products and customers)
- Different contexts may have completely different models of shared concepts with mechanisms to map between these concepts for integration



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

- Monoliths hinder flexibility, speed to market, continuous delivery, and scalability.
- A well designed MicroServices approach can alleviate many of these problems.
- However, this requires a proper dissection of the Problem Domain into Sub Domains.
- This requires BA's to help define a Ubiquitous Language to help create a Bounded Context for the proper solution of the business problem.
- Whether you are dismantling an existing Monolith or trying to avoid building a new one, understanding Bounded Contexts are key.

Adventure is out there!
The END… or is it the Beginning?