Productivity *and* Cost-Effectiveness with DDD

“Defying the Microservices Deathstar”

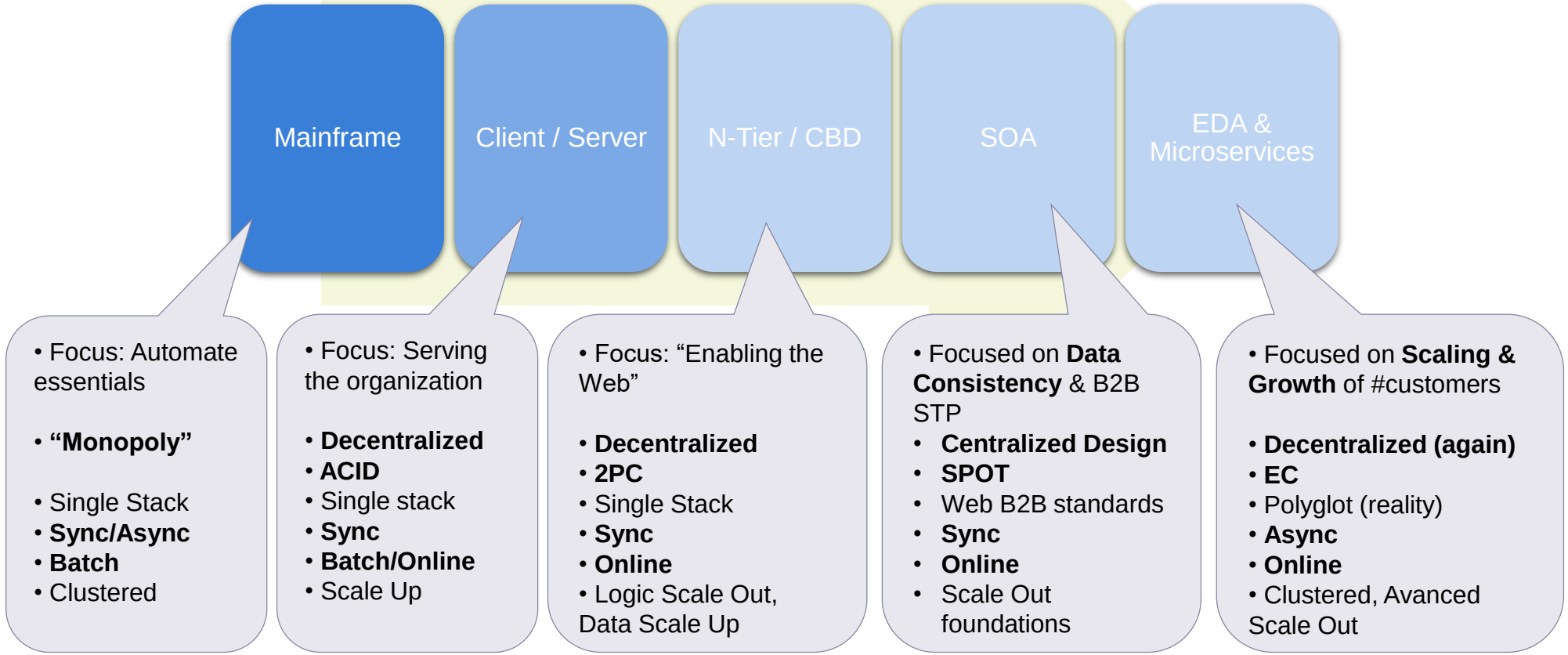
Raimond Brookman
Principal Architect

InfoSupport
Solid Innovator

@raimondb

https://www.flickr.com/photos/beraldoleal/8681750288

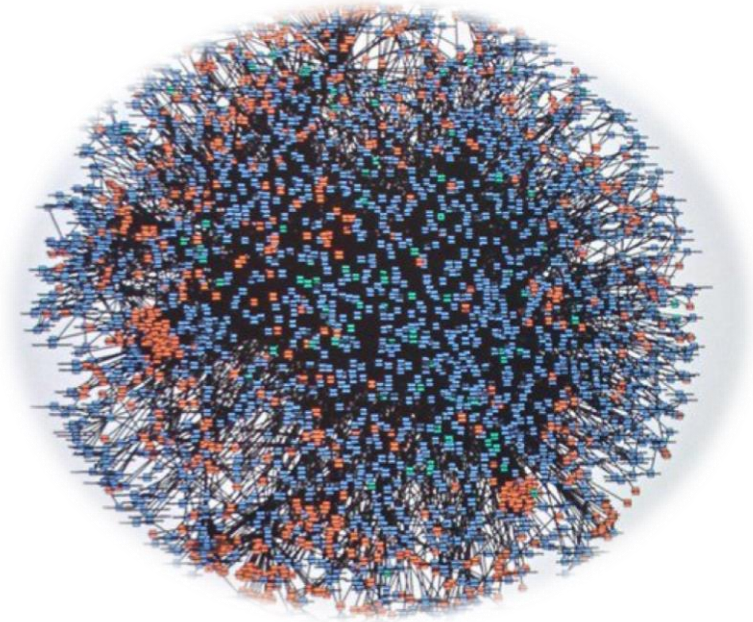
Past Problems, where did we come from?



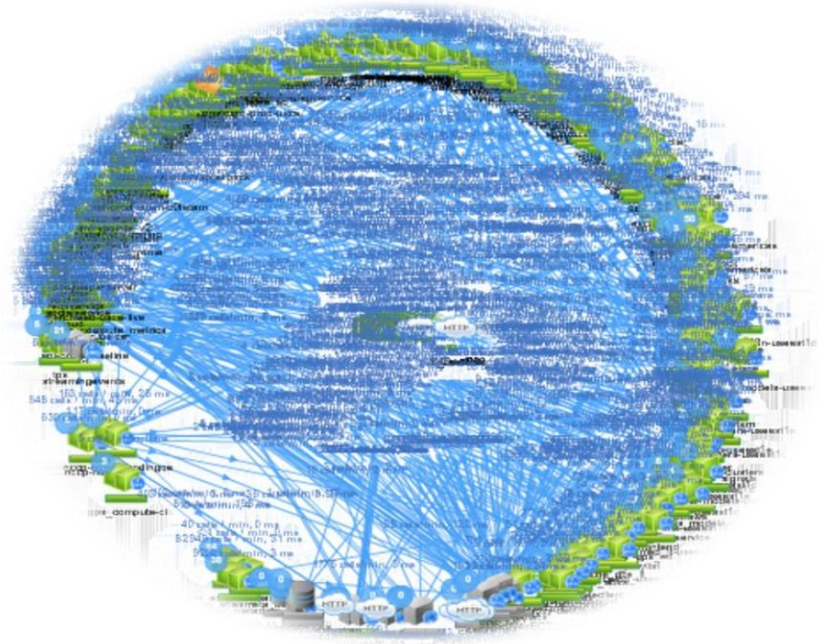
Solution by Microservices

- Decentralized
  - Small units
  - Clear goals, easier change
- Autonomy
  - Independent choices (polyglot)
  - Autonomous releases
- This is great, let’s scale it!
  - Less dependencies and coordination ☺
  - Oh wait…..

Dangers of large microservices ecosystems



Amazon

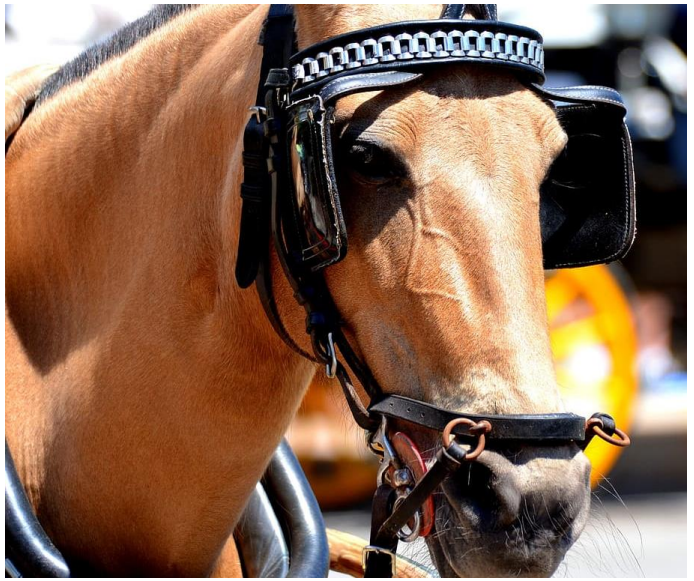


Netflix

Cause of Current Challenges

“Pile of Rubble Architecture” / “Smaller is better”

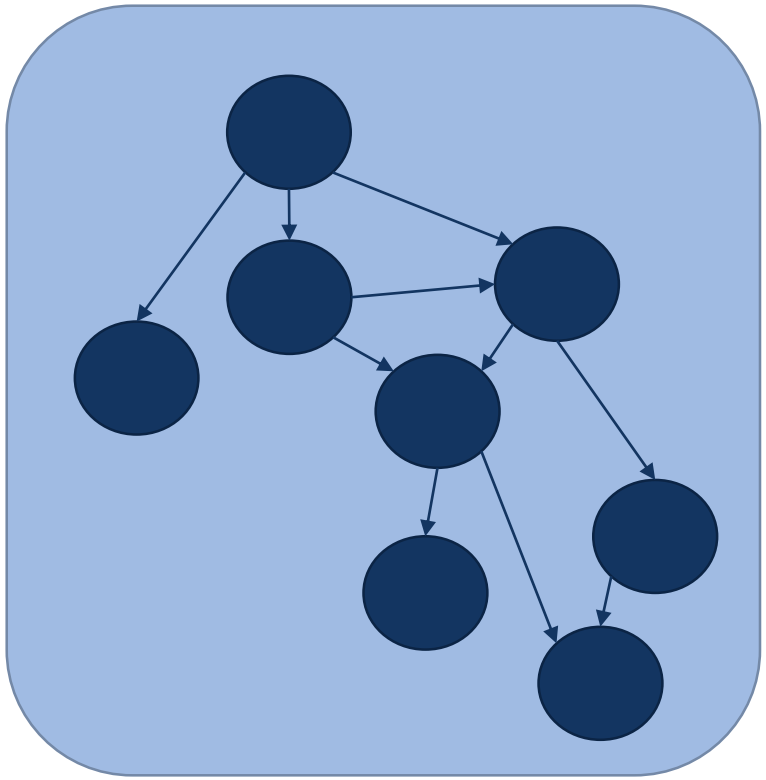
“Reuse is evil”

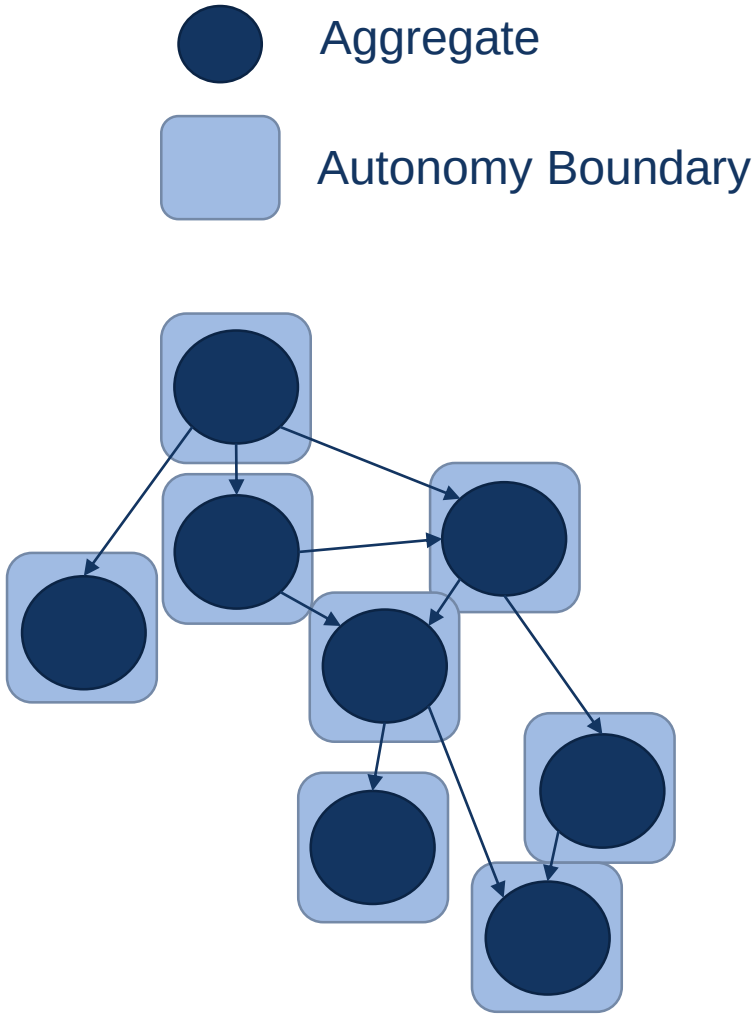


Unmanaged Emergence

Smaller is Better?

What is more complex?





Measuring Complexity

- Qualitative Measure:
  - Scott Woodfield's research (1979) An Experiment on Unit Increase in Problem Complexity
  - Summarized by Robert Glass (2003) in his book Facts and Fallacies of Software Engineering
  - Reformulated by Roger Sessions (2012) in the blog post The Equation every Enterprise Architect Should Memorize – Roger Sessions

Glass's Law:

> For every 25% increase in problem complexity (F), there is a 100% increase in complexity (C) of the software solution.

Calculating complexity

- Session's Summation:

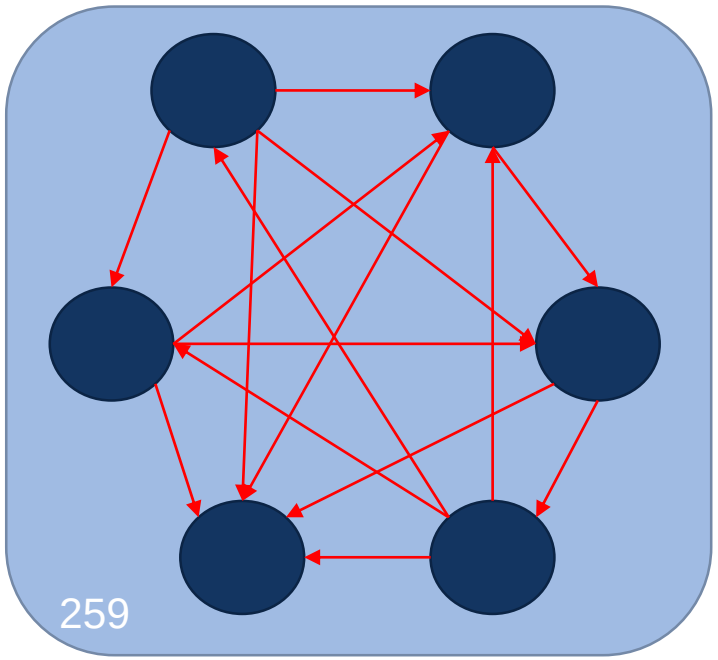
$$C = \sum_{i=1}^{m} \left(10^{3.1 \log(bf_i)} + 10^{3.1 \log(cn_i)} \right)$$

  - bf_i = number of functions **inside** a module
  - cn_i = number of connections to **other** modules

- Brookman's DDD Complexity:
  - bf_i = number of Aggregates **inside** an Autonomy Boundary
  - cn_i = number of distinct Aggregate-Bound Commands & Events dependencies to **other** Autonomy Boundaries

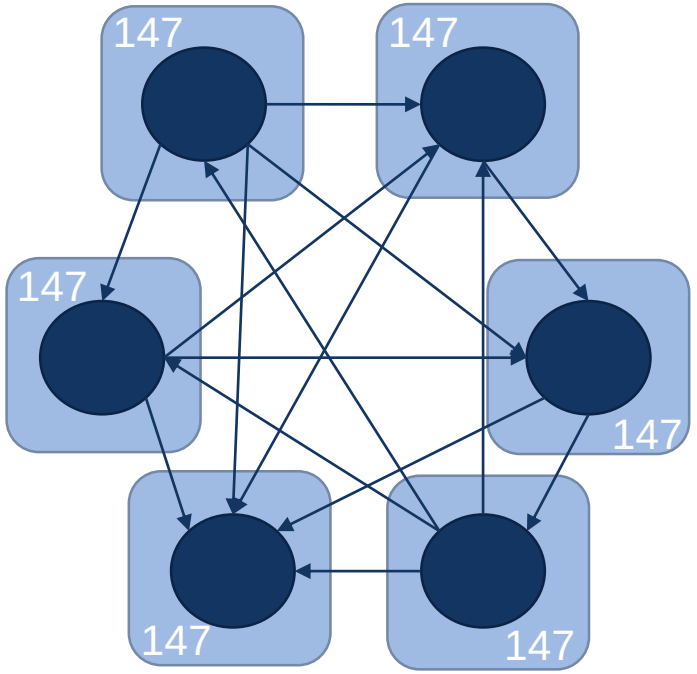
Example: Fully Connected Aggregates

Monolithic Context



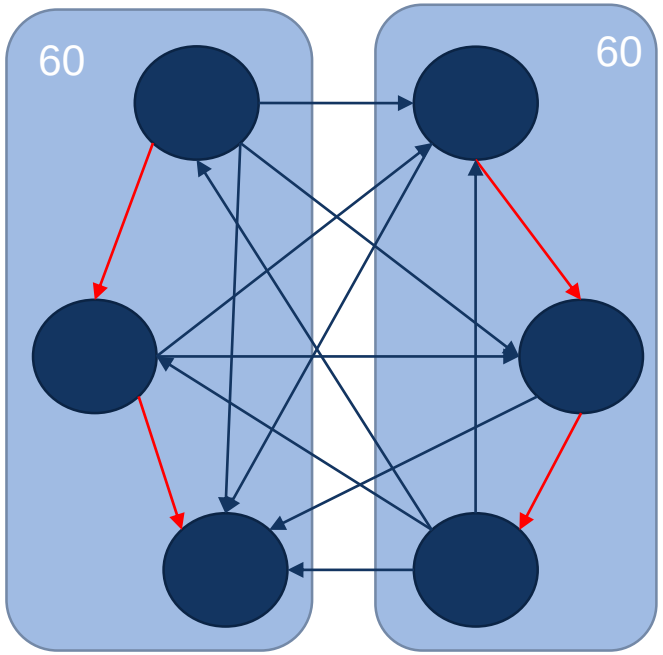
$C = 259$

Minimalized contexts



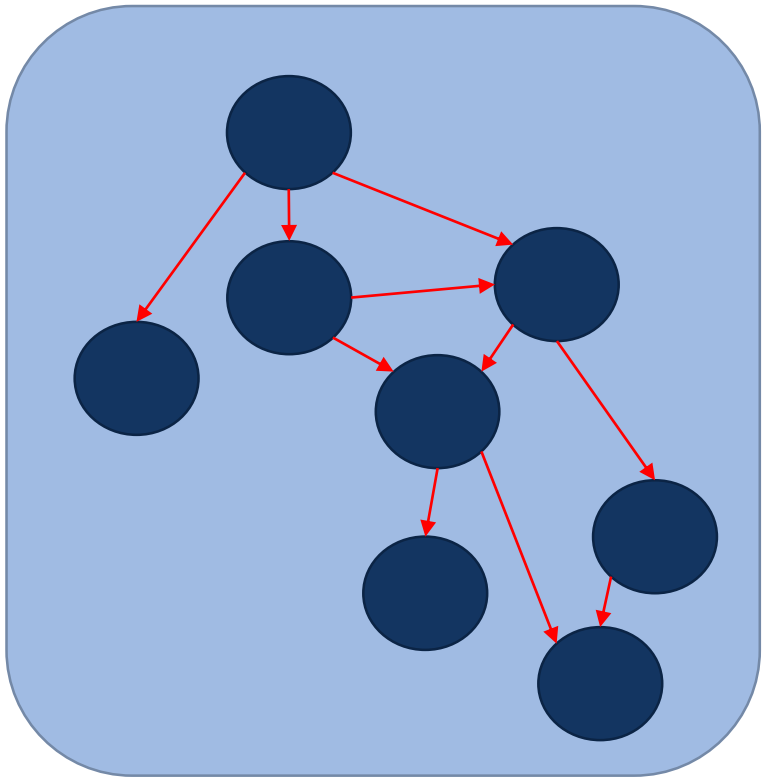
$C = 887$

Optimized contexts

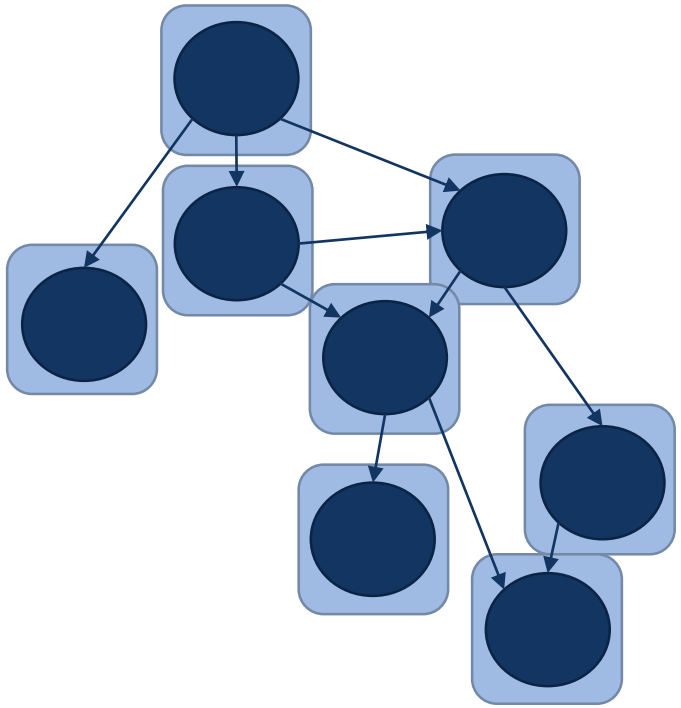


$C = 120$

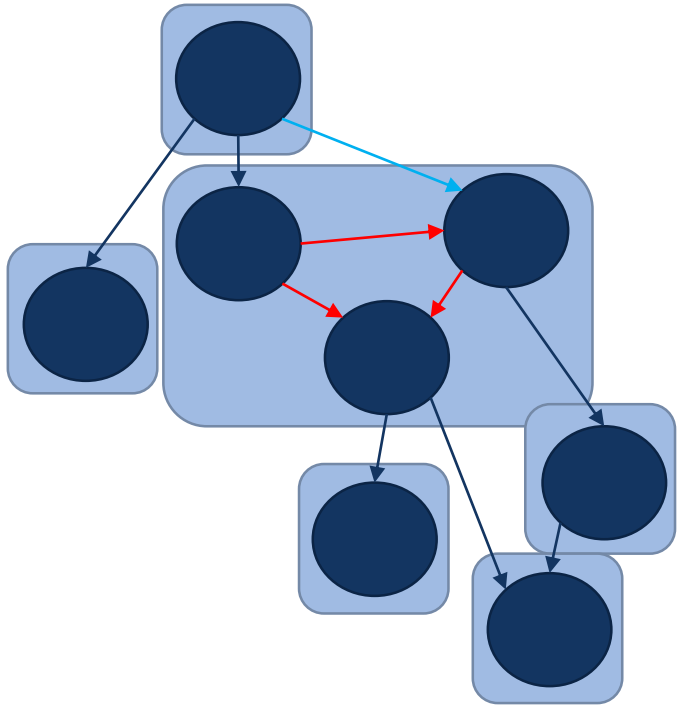
Back to our first Question…



$C = 631$



$C = 234$



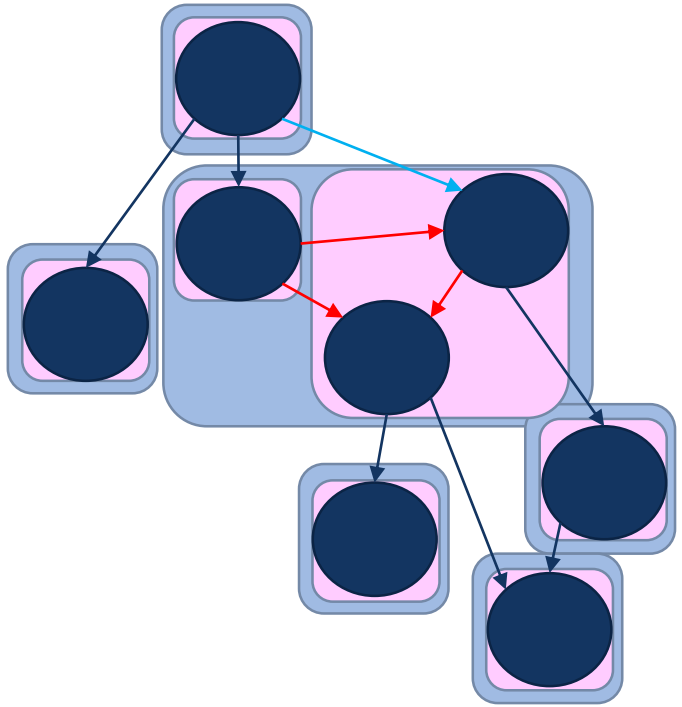
$C = 158$

What do we learn from this?

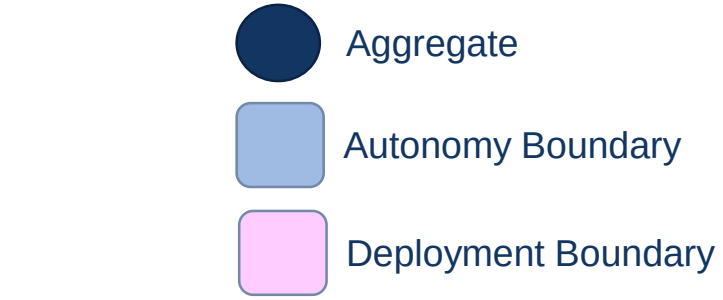
- YMMV on system complexity depending on connectedness
  - Disclaimer: Formula needs to be tuned per organization
- To keep the system level complexity low:
  - Minimize number of external connections per module
  - Minimize number of aggregates per module
- Balancing act of Cohesion & Loose Coupling
  - Components that must change in unison and / or evolve together should be co-located in the same Autonomy Boundary
  - This automatically leads to “clusters” of aggregates that have heavy functional interdependencies

But I want to scale my services independently!

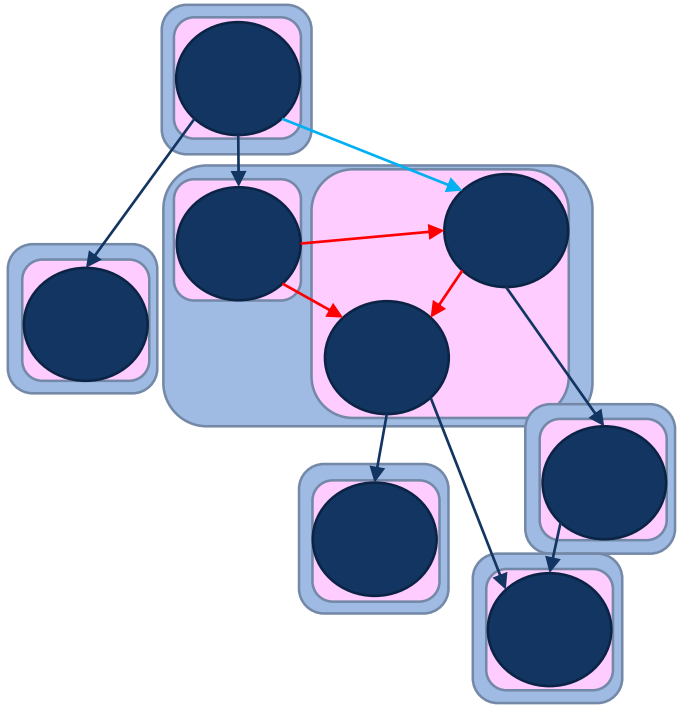
- There a more reasons for deploying separate services than functional autonomy
- The previous exercise was about Autonomy Boundaries.
- Microservices
  - are inside an Autonomy Boundary
  - have their own Deployment Boundary
  - A Deployment Boundary contains **1+ Aggregates**
- Complexity is mostly dependent on coordination, which is hardest across Autonomy Boundaries



So how does this fit to Bounded Contexts?



- Bounded Contexts allow for independent models and Ubiquitous Language
- So it is an Autonomy Boundary
- I often see this confused with Deployment Boundaries
- **Pro Tip:** follow Conway's Law and make sure a Bounded Context is the responsibility of a Single Team



Managing Complexity Conclusions

- Smaller is not always better
  - Don’t just smash your monolith into a “Pile of Rubble”
- Clear Autonomy Boundaries are most important!

Having a Wider View

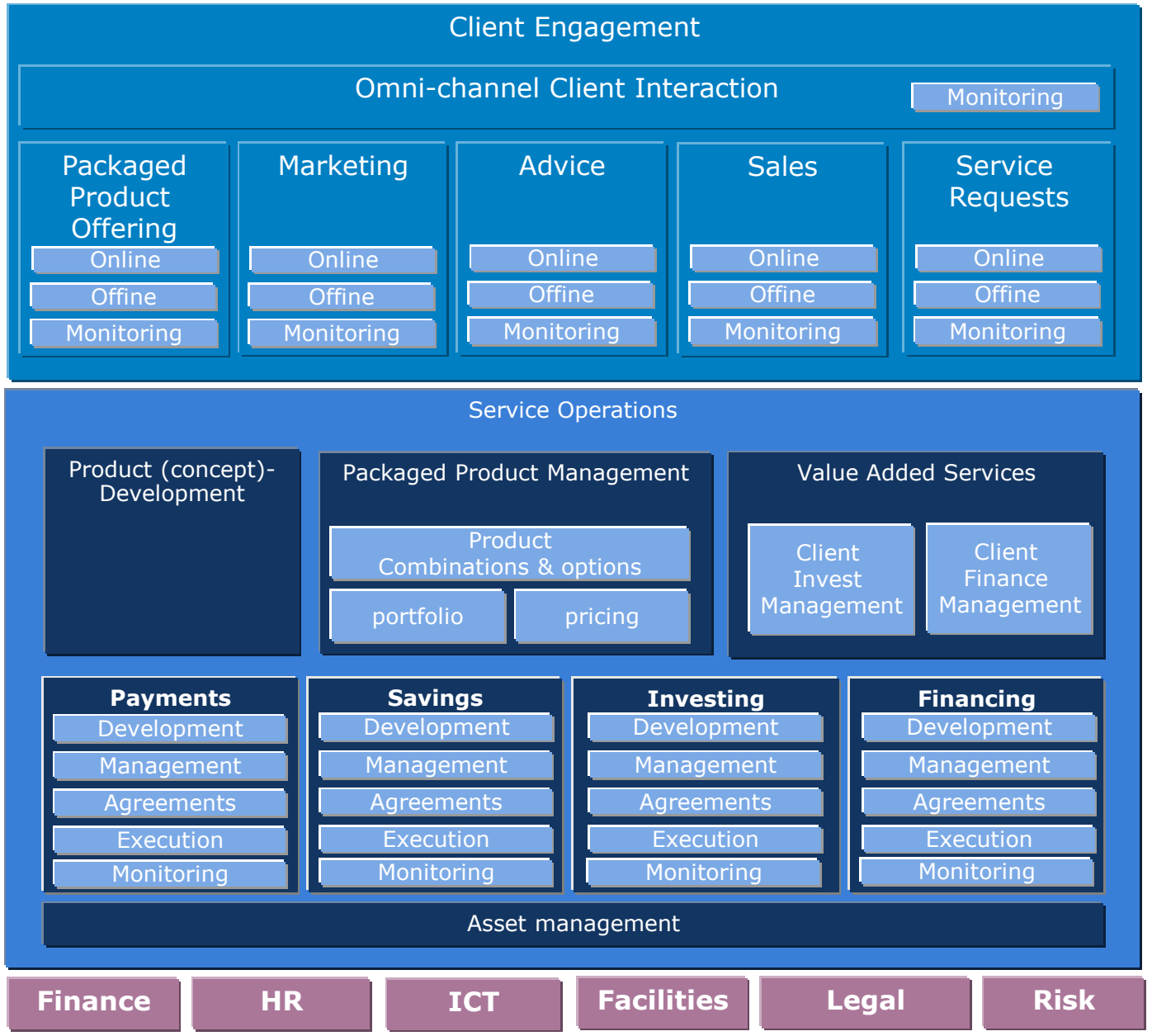
Finding the Business Domains

Event Storming

- Great for bottom-up analysis
- Helps to scope *your* Bounded Context
- Works well when in “Unknown territory”
  - E.g. Lean Start-up, new services
- Not so great for the Big Picture in larger organizations

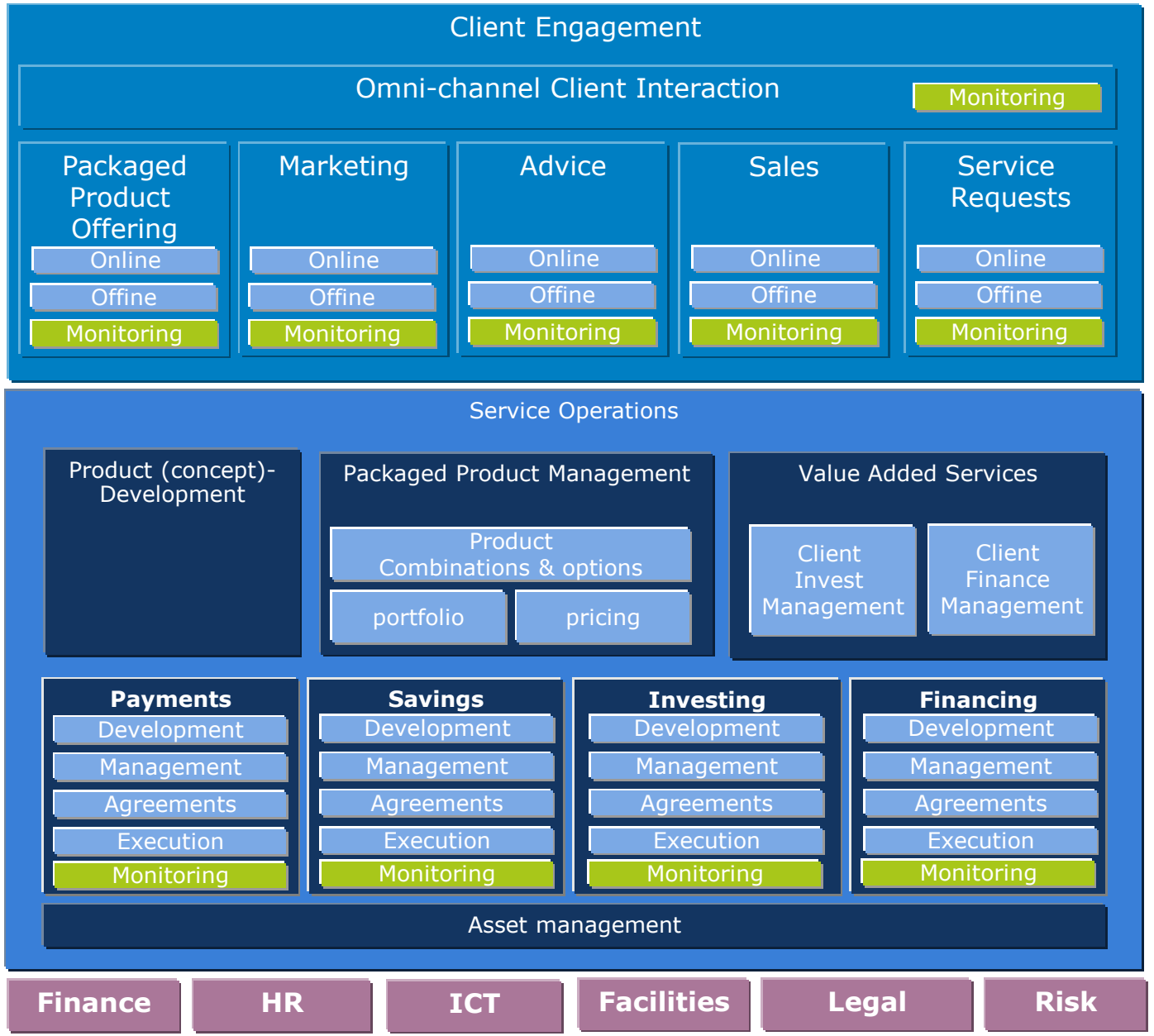
Business Capability Mapping : Domains

- Great for Big Picture
- Identify potential extractable Sub Domains



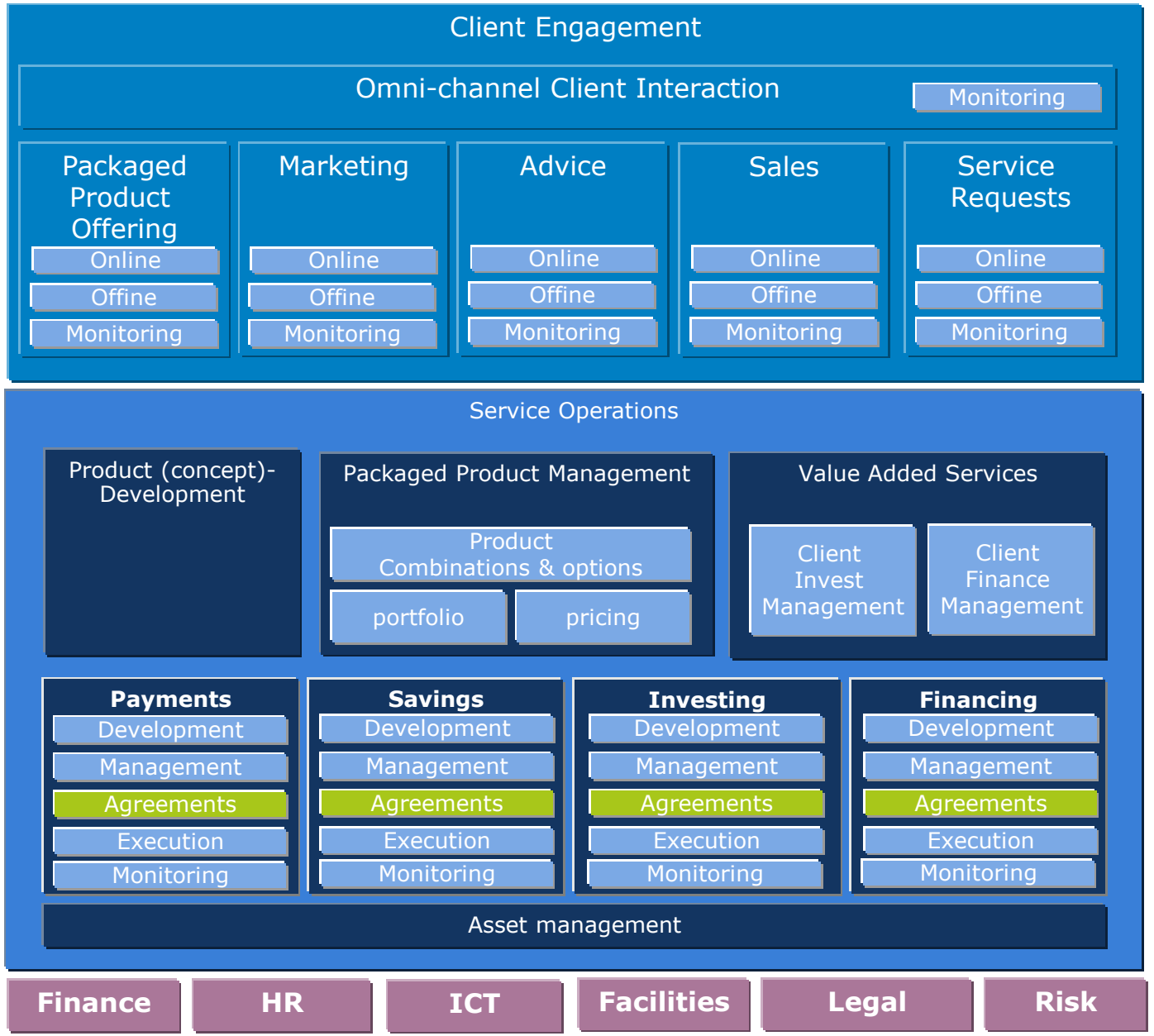
Business Capability Mapping : Domains

- Local Monitoring Choices for Agility?
- A Common Analytics Platform with Local solutions?
- Or a separate Capability offering services to other Domains?



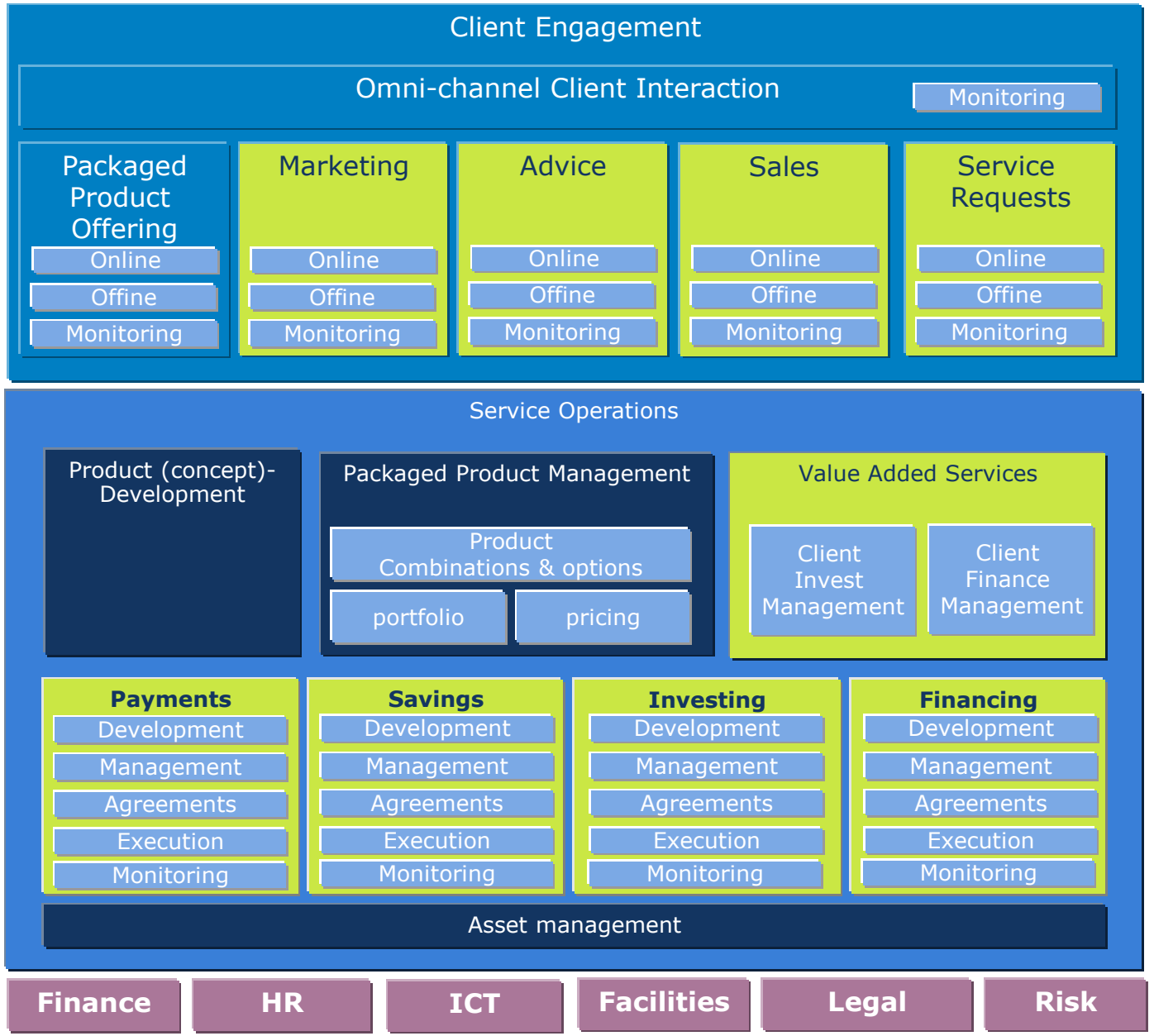
Business Capability Mapping : Domains

- Local Agreement Management Solutions?
- Or centralized?



Business Capability Mapping : Domains

- Local Client Management?
- Or Central CRM?
- Where to register
  - Touchpoints
  - Contact Information
  - Product/Services in Use
  - Customer Satisfaction



Identifying Bounded Contexts with a CU-matrix

1. Plot Business Functions / Capabilities

2. Plot Involved Aggregates

3. C : Creation/Commands
U: Used (Read or Events)

	Portfoliogegevens	Architectuur	Managementinformatie	Procesgegevens	Standaardaanvragengegeve	Opdrachtenadministratie	Projectgegevens	Projectmanagementgegevens	Changegegevens	Testmanagementgegevens	Designgegevens	Klantmeldingen	Incidentgegevens	Problemgegevens	Beschikbaarheids/capaciteits/	Klantgegevens	Service level gegevens	dienstencatalog
Portfoliomanagement	C																	
Architectuurmanagement		C					C				C							C
Procesmanagement																		
Standaardaanvragen																		C
Ordermanagement																		
Projectmanagement																		
Specificatiemanagement		U									C							
Change management				U		C		C	C			U	C	U			C	C
Design							C	C		U	C	U						
Realisation							C	C		U	C	U						
Test								C		C	U							
Incident management			C									U	C					
Problem management									C				C	C	U			C
Capacity-, availability & continuity management	U		C					C	C	U	C		U	U	C		C	
Call / Service desk management					C				C			C	C					C
Service level management														U		U	C	

4. Shift rows and columns collecting C's together

Aggregates / Business Objects

Business Functions

5. Identify Bounded Contexts Based on Cohesion and Loose Coupling

How to Decide?

Central

- Typical choice in “SOA era”
- Based on Data Integrity
- Too Little View on Process
- Too much generalization

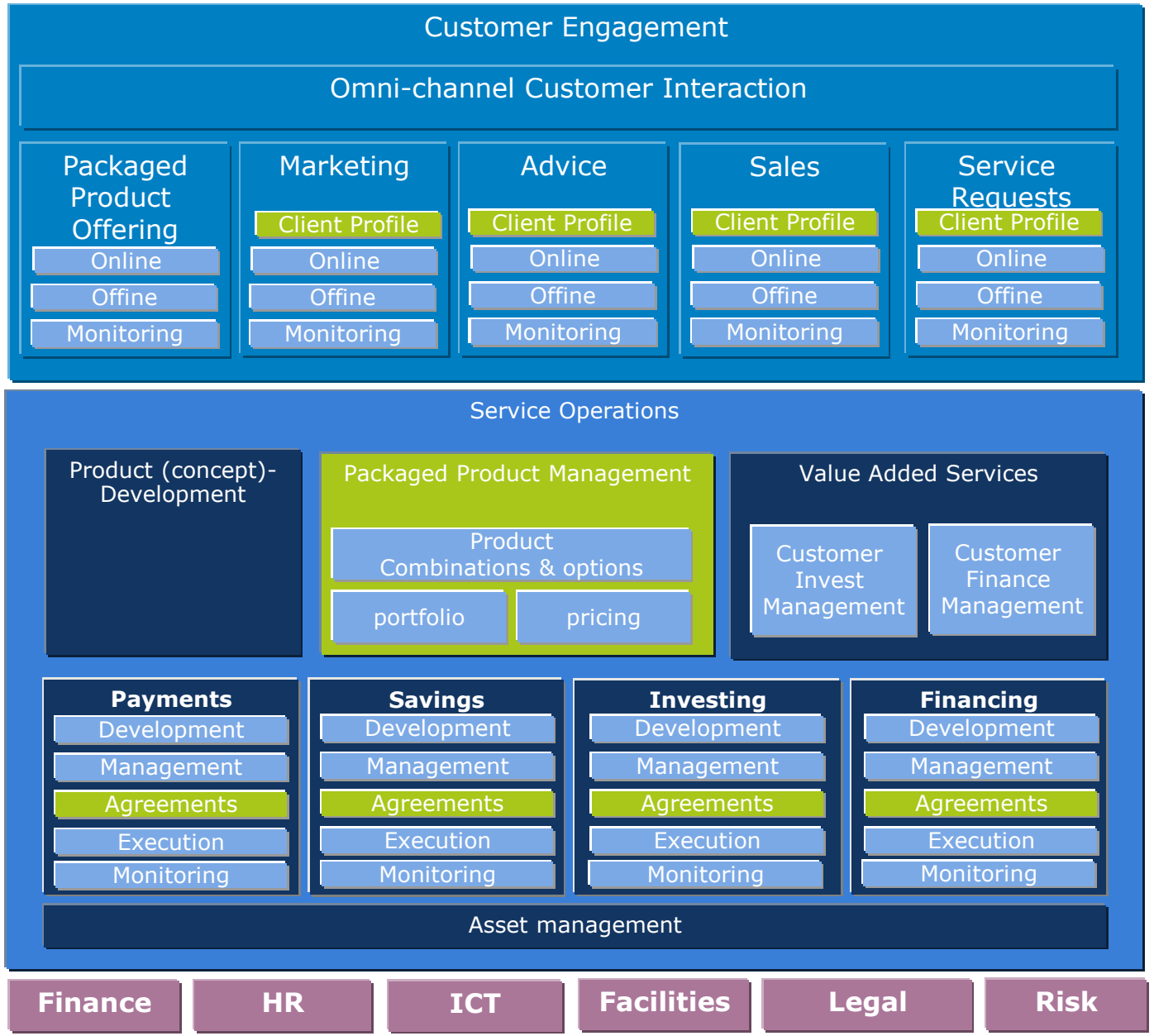
Both

- Strategic DDD focused
- Factor Aggregates based on **Both Data & Process**
- Optimized Choice, multi-level

Local

- Typical current choice
- Tactical DDD focused
- “Process Task” Driven
- Too Little view on Big Picture

Smart Reuse by splitting local and corporate responsibilities



Corporate Customer Management

- Customer
- Touchpoint
- Used Services
  - Contract Agreement

Payments Account Management

- Customer Payment Preferences
- Payment Contract Renewal

Other Opportunities

- **Output Management** (execution)
- **Data Analytics** (monitoring)

Packaged Product Management

- Model Contract Agreement
- Active Services Portfolio

Reuse Conclusions

- Take off your Blinkers!
- Find your balance by Middle Out

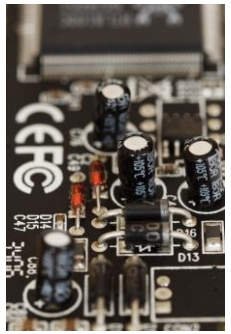
- Strategic DDD
  - Big Picture Bounded Contexts using Business Capability Maps
  - “Top Down”

- Tactical DDD
  - Refine Context splits & Boundaries using Event Storming
  - “Bottom up”

Managing Emergence

Modularity Evolution

1947



Core Components

Transistor, Resistor, Capacitor

3GL Language & Compilers

C: 1970s

-25

1960s

Generic ICs

Memory, I/O, Shift Register

Packaged Library & Management

Maven: 2004
NuGet: 2010
NPM: 2010

-45

1970s

Application Specific ICs

Harddisk Controller
Microprocessor

Packaged Subsystem Modules

Docker: 2013

-45

1980s

Open Systems Architecture

Technical Bus Architectures
Functional Interfacing Standards
(Graphics, Audio, Storage)

Standardized Infra & Functions (Cloud)

Kubernetes: 2015
Limited Functional standards

-35

The Bad News:

IT is wildly running behind in delivering high-level modularized solutions

The Good News:

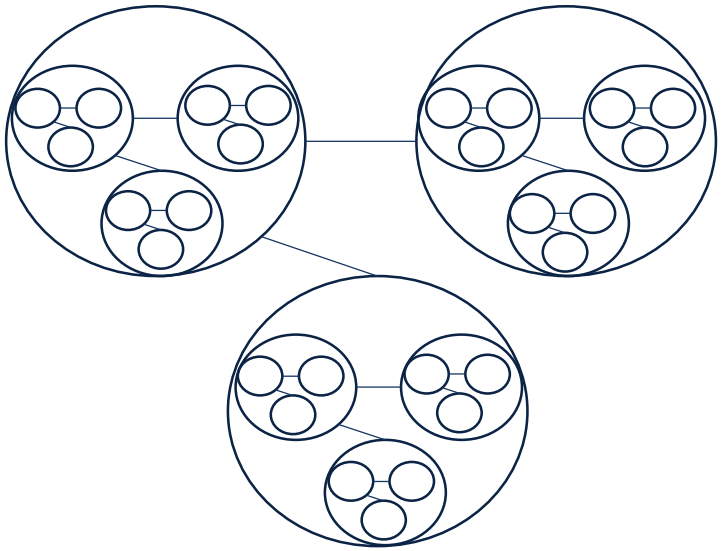
We are starting to catch up
The tools are in place ☺

Repeating Patterns

- Managing complexity in Hardware because of muti-level modularization
  - Hiding internals
  - Explicit Interface
  - Good old OOP Practices
- Nature manages complexity by repeating patterns (aka Fractals)

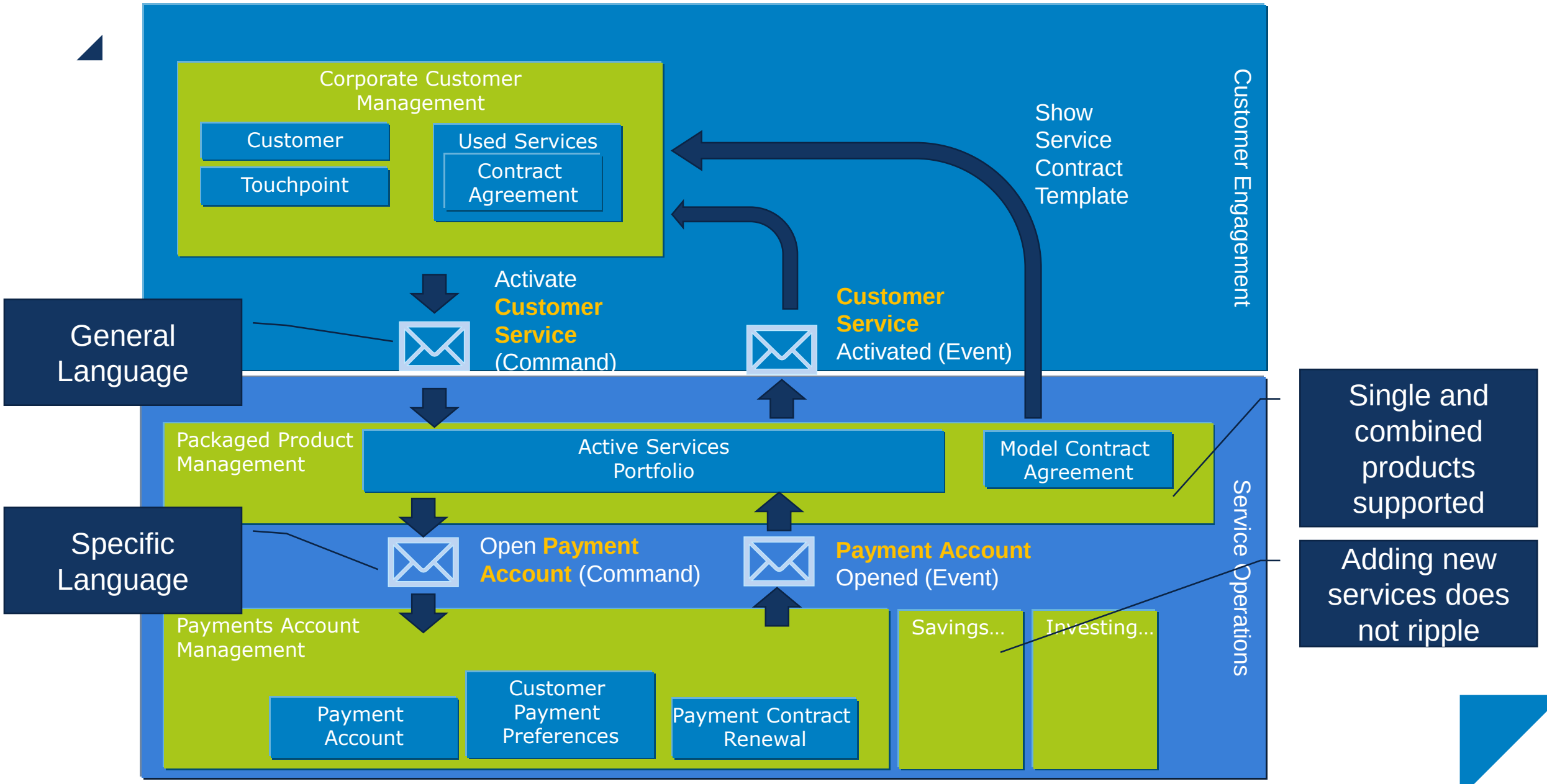
Back to the Deathstars..

- 2D “Single Level” Bounded Context
- Little OO Principles @ organization scale



- 3D “Fractal” Bounded Contexts
- A Context has its own Language at each Level
- Deeper levels can have more specialized communications

Multi-level Ubiquitous Language



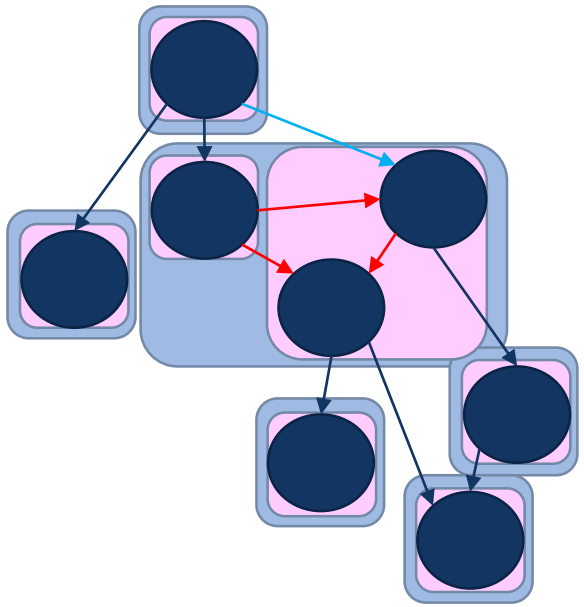
But isn’t this BDUF?

- Business Capabilities for “Rough Boundaries”
- Define high level Ubiquitous Language
  - Only Primary Business Concepts

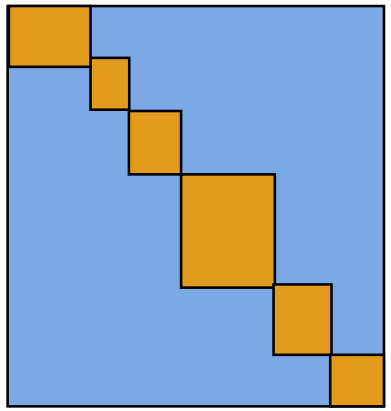
- Refine with Event Storming
- (Re-)establish “external interface” every time
- With each new Bounded Context also try to establish parent Context / Domain

Take Aways

- Smaller is not always better



- Big Picture Analysis to identify Balanced Reuse



- “Fractalize” your Bounded Contexts to manage complexity

Questions?

Raimond.Brookman@infosupport.com

@raimondb

Thanks!

Come see us at our Booth for further discussions!

Raimond.Brookman@infosupport.com

 @raimondb