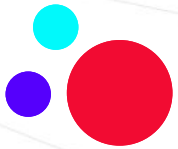
Architektura i narzędzia w systemach mikroserwisowych

infoShare Academy

Oleksii Tsyganov

infoShareAcademy.com

01. Agenda



Agenda

Welcome
History of Microservices
Problems of Monolith & SOA
Microservices Architecture
Problems Solved by Microservices
Designing Microservices Architecture
Deploying Microservices
Testing Microservices
Service Mesh
Logging And Monitoring
When NOT to use Microservices
Microservices and the Organization
Anti-Patterns and Common Mistakes
Breaking Monolith to Microservices
Case Study
Conclusion

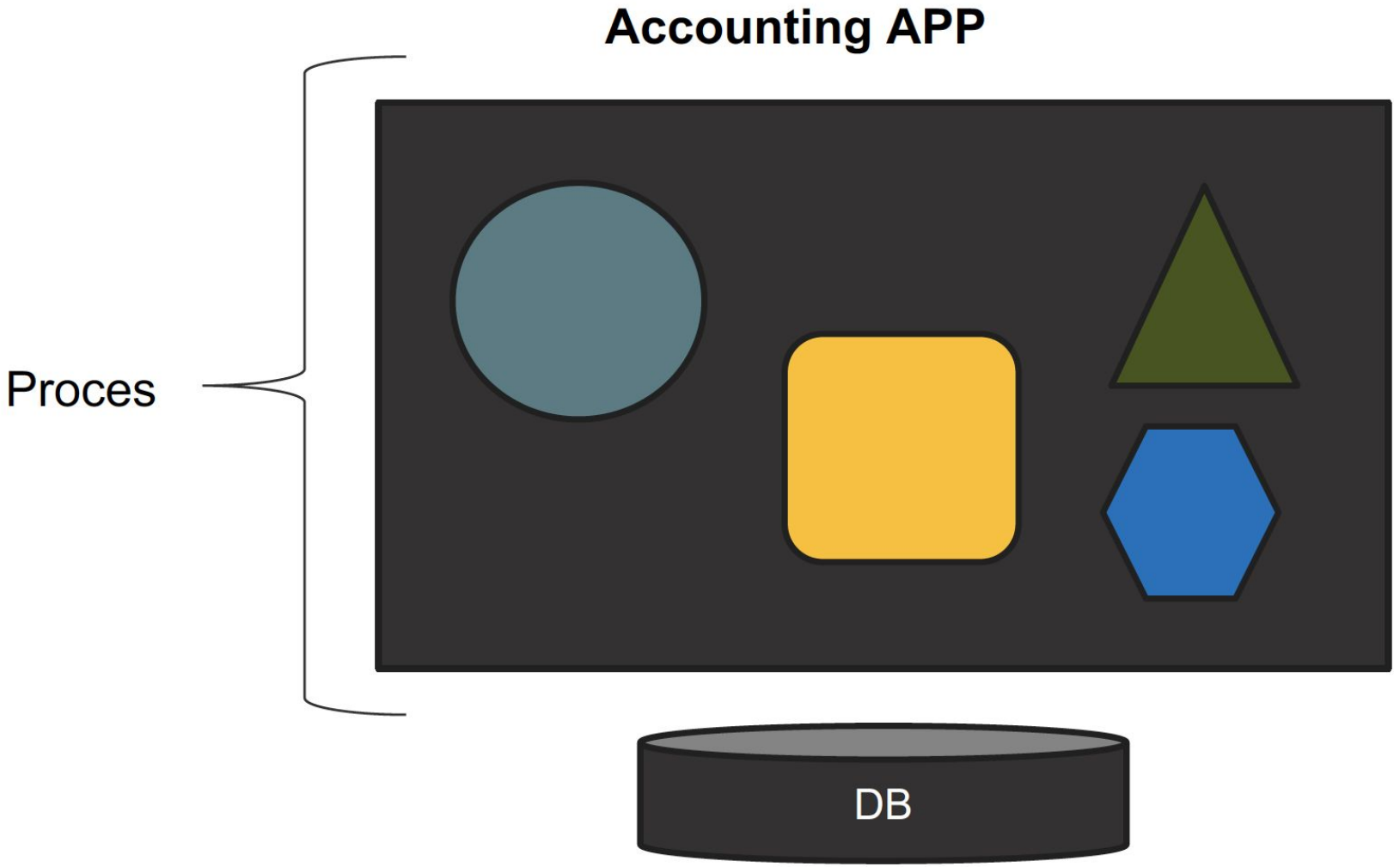
Before microservices. Monolith.

Jedyny proces

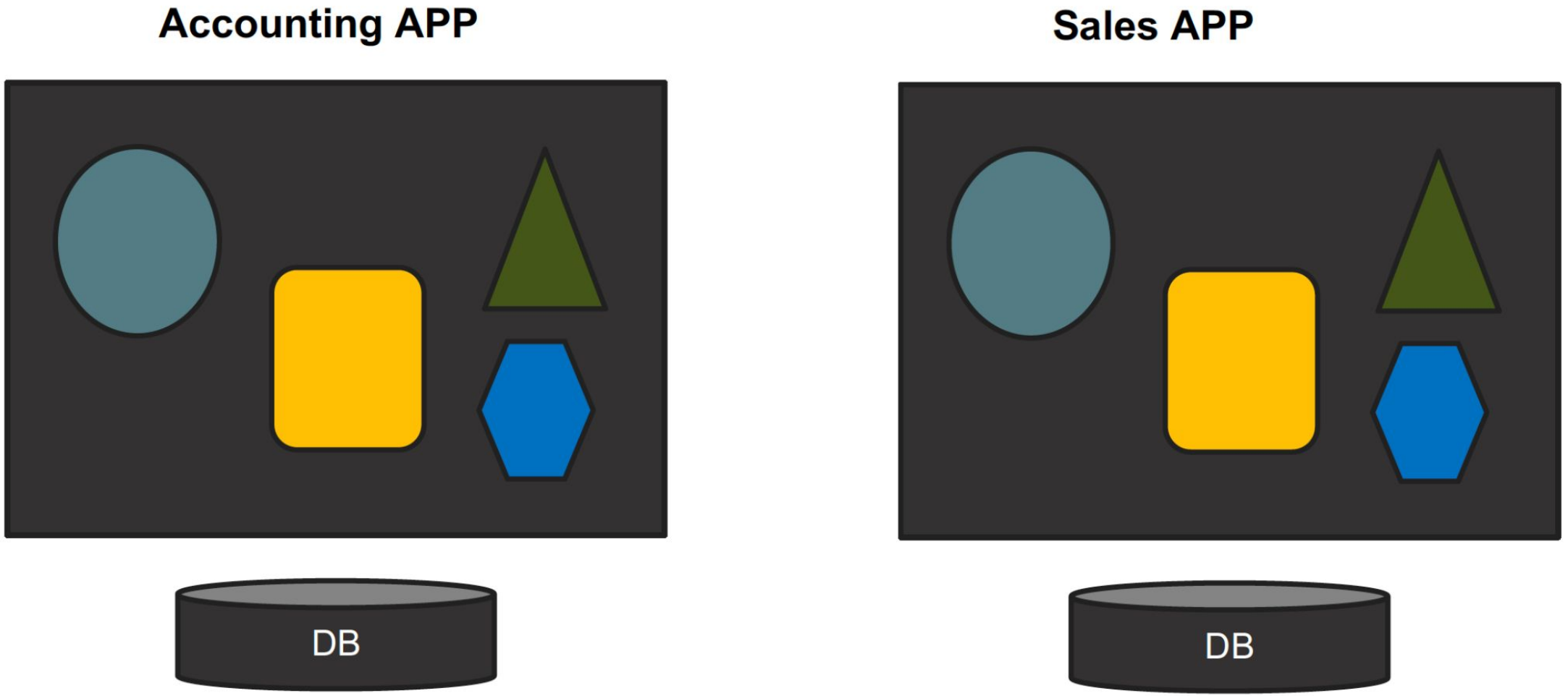
Związki pomiędzy wszystkimi klasami

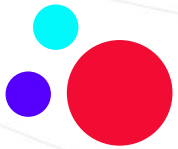
Implementowana jako “silos”

Before microservices. Monolith.



Before microservices. Monolith.



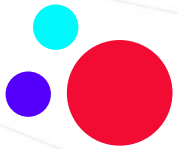


Monolith. Pros.

Łatwo projektować

Wydajność?

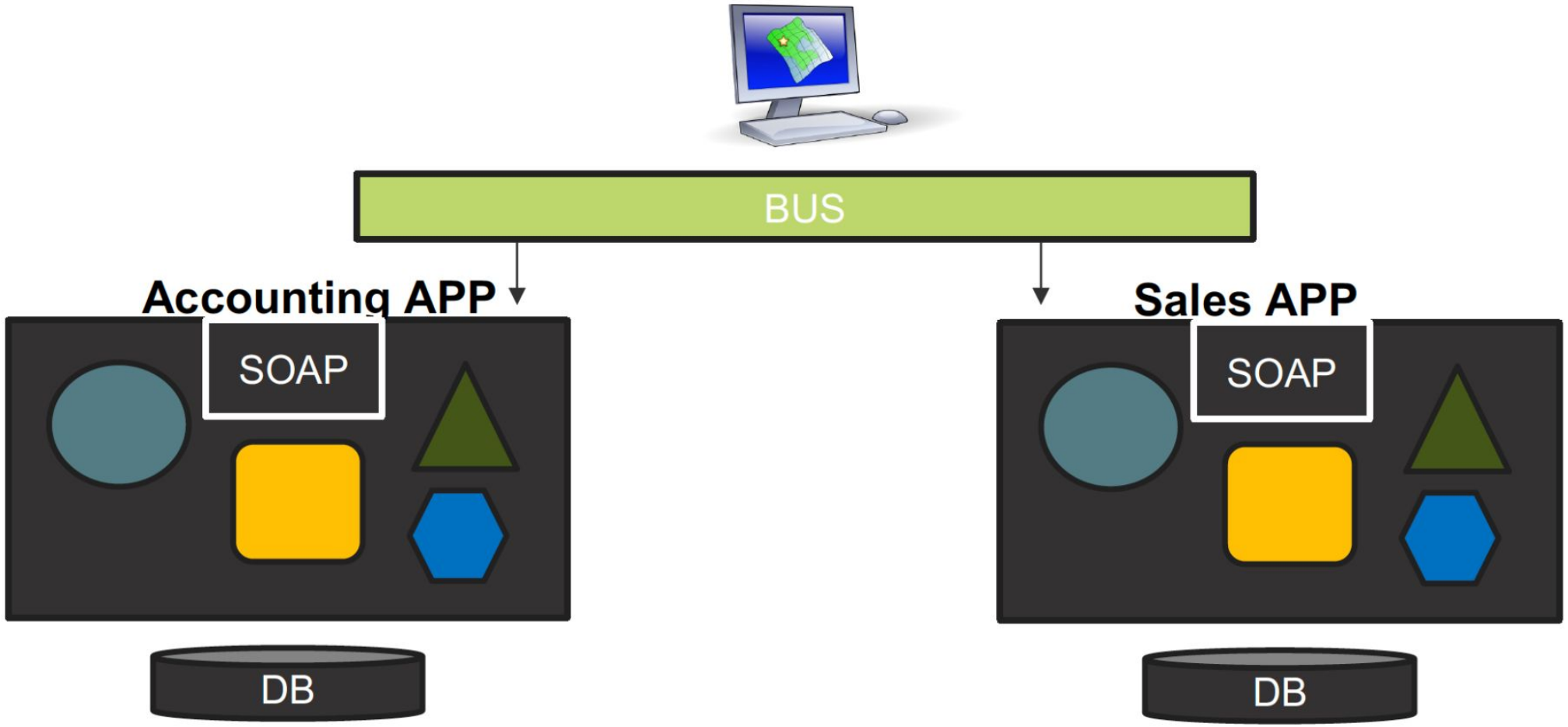
infoShare ACADEMY



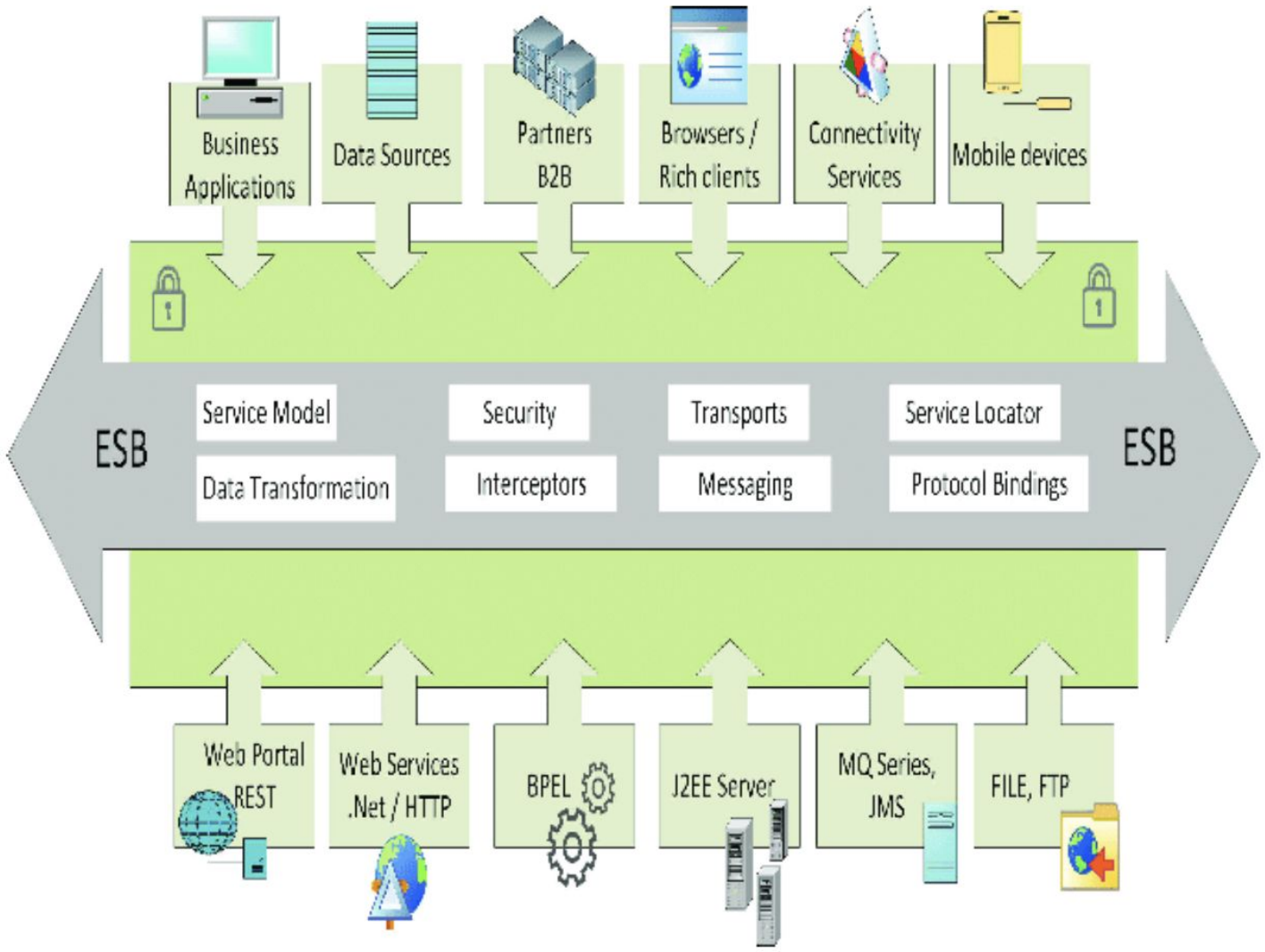
Before microservices. Service Oriented Architecture

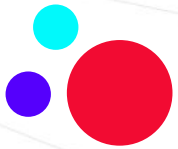
- Aplikację są serwisami, które wystawiają swoją funkcjonalność “w świat”
- Dzielenie się i dawanie
- Serwisy wystawiają metadane żeby zaznaczyć swoją funkcjonalność
- Zwykle implementowane za pomocą SOAP i WSDL
- Implementowane za pomocą Enterprise Service Bus (ESB)

Before microservices. Service Oriented Architecture



Before microservices. Service Oriented Architecture

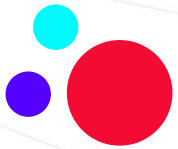




SOA. Pros.

Udostępnianie danych i funkcjonalności

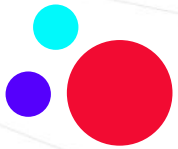
Polyglotowe podejście



Problems with Monolith and SOA

Jedyna platforma technologiczna:

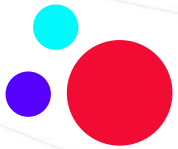
- wszystkie komponenty muszą być dewelopowane na tej samej platformie
- nie zawsze technologia jest lepsza dla zadania
- upgrade jest problemem



Problems with Monolith and SOA

Nieelastyczne wdrażanie:

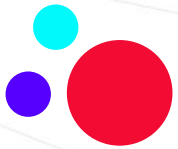
- Zawsze deployujemy całą aplikacje
- Nie ma możliwości deployu częściowego
- Update tylko jednego komponentu wymaga deploymentu całości “codebase”
- Wymuszanie uciążliwego procesu testów całości
- Długie cykle deploymentowe



Problems with Monolith and SOA

Nieelastyczne wdrażanie:

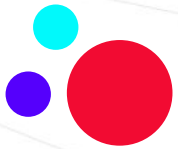
- W monolicie CPU i RAM dzielone pomiędzy wszystkimi komponentami
- Nie ma możliwości przydzielenia większej liczby zasobów dla komponenta systemu



Problems with Monolith and SOA

Skala i skomplikowaność:

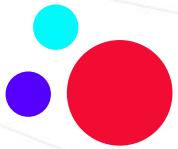
- “Codebase” jest skomplikowany i duży
- Mała zmiana może spowodować problemy z innymi komponentami
- Testowanie nie zawsze wykrywa bugi
- Wsparcie dla produktu jest bardzo trudne
- Bardzo trudne utrzymywanie
- Przestarzały system



Problems with Monolith and SOA

Skomplikowana i droga ESB:

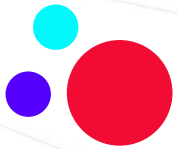
- ESB jest głównym komponentem
- Szybko staje przeciążony i drogi
- Próbuje robić „wszystko”
- Bardzo skomplikowany w utrzymaniu



Problems with Monolith and SOA

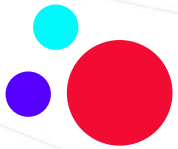
Brak narzędzi:

- dla SOA potrzebne krótkie cykle developmentu
- potrzeba w szybkim testowaniu
- nie ma narzędzi wspierających powyżej wskazane
- nie osiągnięto żadnego oszczędzania czasu w porównaniu z monolitem



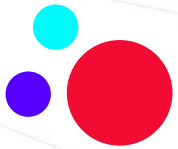
Architektura mikroserwisowa

- Problemy z monolitem i SOA przeprowadziły do nowego paradygmatu
- Musi być modularny, z prostym API
- Pojawił się w 2011, ale realnie otrzymał życie w 2014 roku
- Martin Fowler “Microservices” - standard “de-facto”



Cechy mikroserwisów

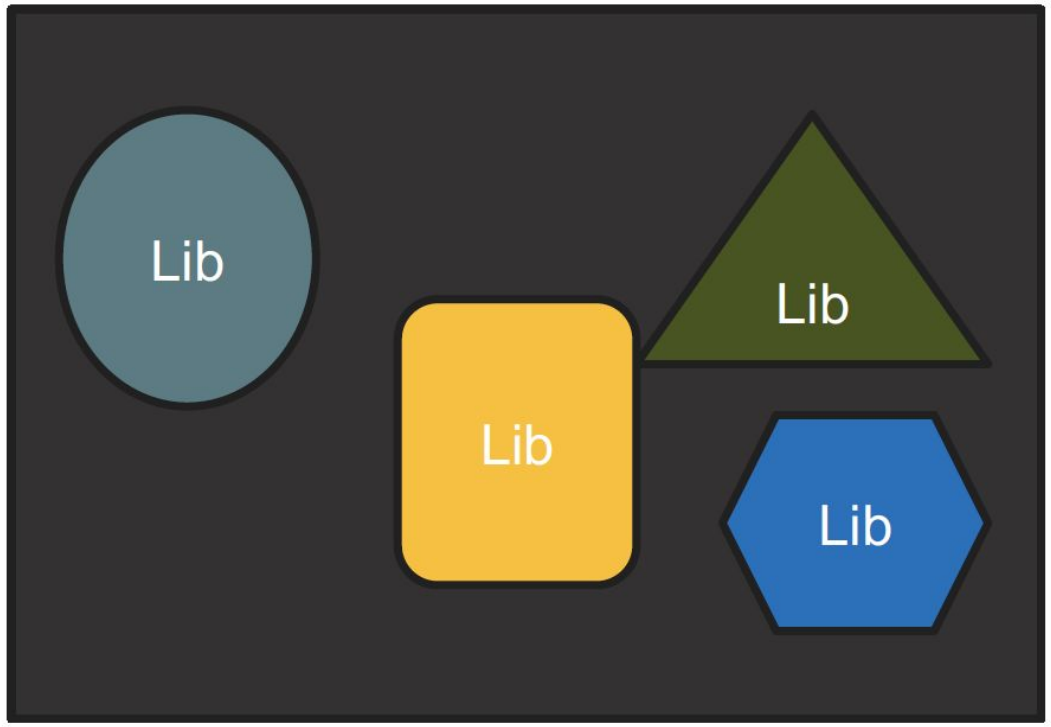
- Componentization via Services
- Organizes Around Business Capabilities
- Products not Projects
- Smart Endpoints and Dumb Pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for Failure
- Evolutionary Design

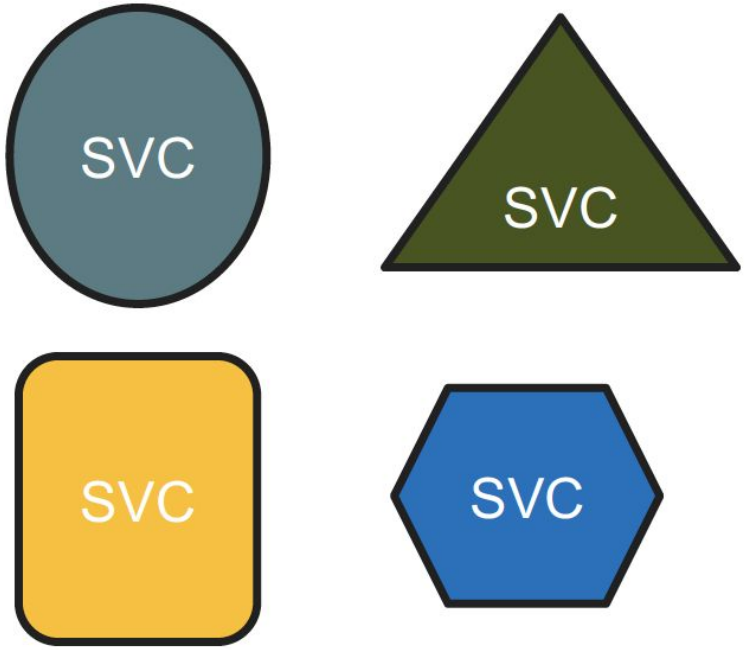


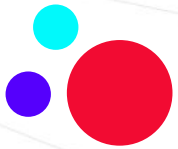
Componentization via Services

- Design modułowy zawsze jest dobrym pomysłem
- Komponenty są częściami które tworzą produkty
- Modularność może być osiągnięta poprzez:
  - biblioteki – wywołane bezpośrednio w procesie
  - serwisy – wywołane poza procesem (API, RPC)
- W mikroserwisach oczywiście preferujemy serwisy a nie biblioteki

Componentization via Services

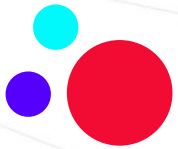






Organized Around Business Capabilities

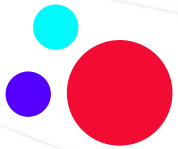
- **Tradycyjne projekty**:
- Mają zespoły z poziomową odpowiedzialnością: UI, DB, API
- Bardzo powolna komunikacja pomiędzy zespołami
- Nie używają tej wspólnej terminologii
- **Nie mają wspólnych celów**



Organized Around Business Capabilities

Zalety:

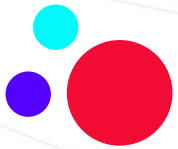
- Szybki development
- Dobrze oznaczone granice pomiędzy serwisami; również ludzie wiedzą, co trzeba robić



Products not Projects

Tradycyjne projekty:

- Celem jest dostarczenie pracującego kodu
- Nie ma trwałych relacji z klientem
- Często w ogóle nie znają klienta
- Po dostarczeniu kodu zespół przechodzi do kolejnego projektu

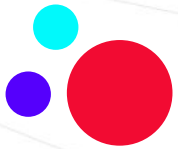


Products not Projects

Mikroserwisowe projekty:

- Celem jest dostarczenie pracującego produktu
- Produkt wymaga stałego wsparcia i bliskiej współpracy z klientem
- Zespół odpowiedzialny po dostarczeniu serwisu

“You build it, you run it”
W. Vogels, AWS CTO

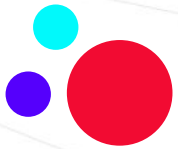


Smart Endpoints and Dumb Pipes

SOA projekty używają:

- ESB
- WS-* protokoły: WS-security, WS-messaging, WS-discovery

Pomiędzy serwisowa komunikacja stała zbyt skomplikowana i trudna do utrzymania

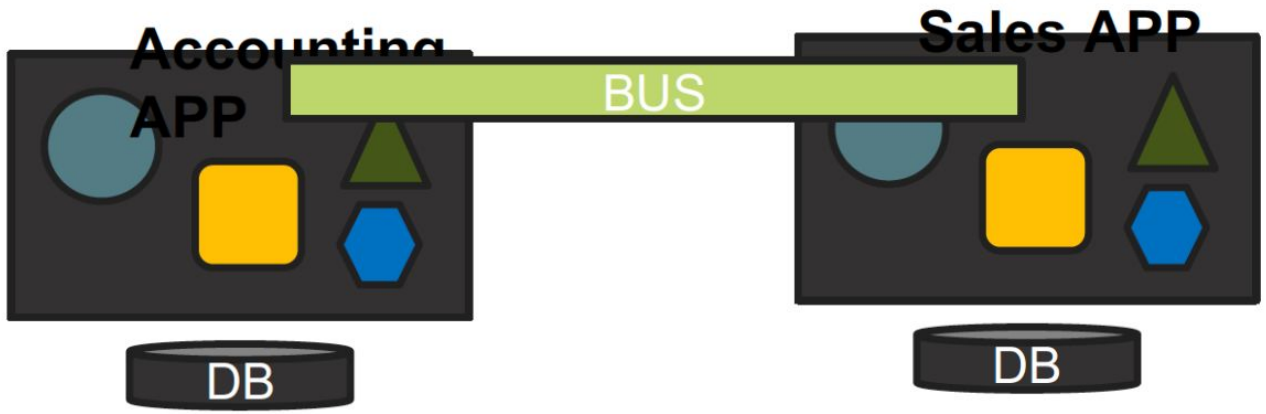


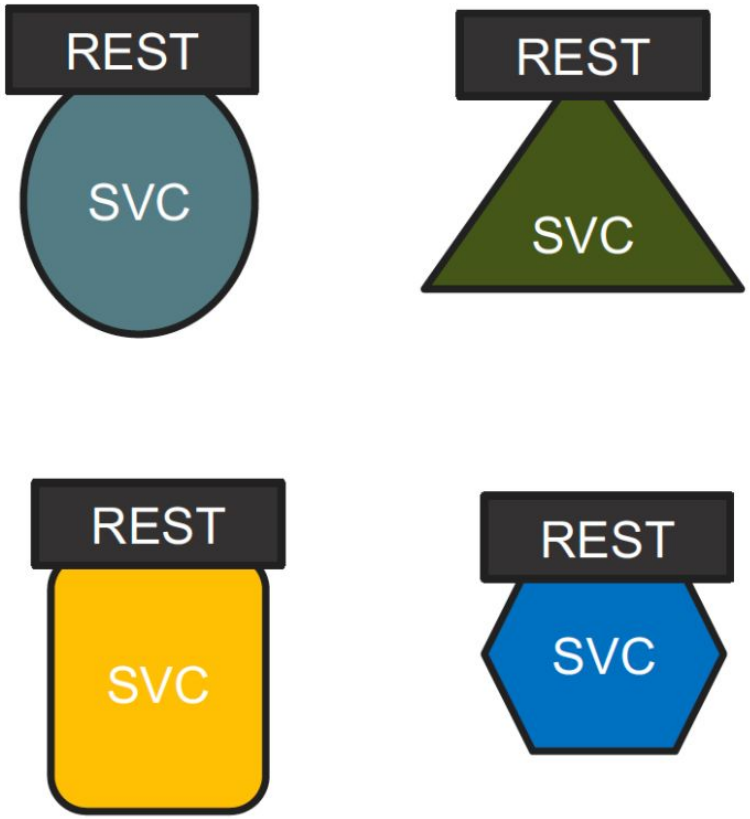
Smart Endpoints and Dumb Pipes

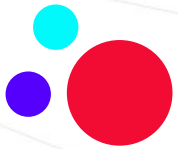
Systemy mikroserwisowe:

- Używają dumb pipes - prostych protokołów
- Nie wymyślają nowego, używają to co WEB proponuje
- Zwykłe REST API

Smart Endpoints and Dumb Pipes







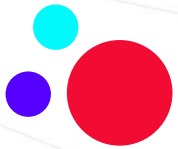
Smart Endpoints and Dumb Pipes

Ważne:

- Bezpośrednie połączenie pomiędzy serwisami nie jest dobrą opcją
- Lepsze rozwiązanie – używanie service discovery lub gateway
- Więcej protokołów w ostatnie lata (GraphQL, gRPC) – są skomplikowane

Co wygrywamy używając Smart Endpoint oraz Dumb Pipes:

- Przyspieszamy development aplikacji
- Robimy aplikacje prostą do obsługi



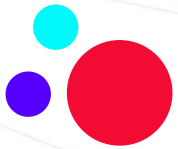
Decentralized Governance

Klasyczne projekty:

- Jest standard do prawie wszystkiego

Mikroserwisy:

- Każdy zespół jest odpowiedzialny za swoje decyzje
- każdy zespół jest odpowiedzialny za swój serwis
- System poliglota



Decentralized Data Management

Klasyczne projekty:

- Jedna baza danych - przechowuje wszystkie dane komponentów systemu

Mikroserwisy:

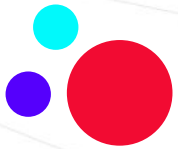
- Każdy serwis może mieć swoją bazę danych

Infrastructure Automation

Azure DevOps

GitLab

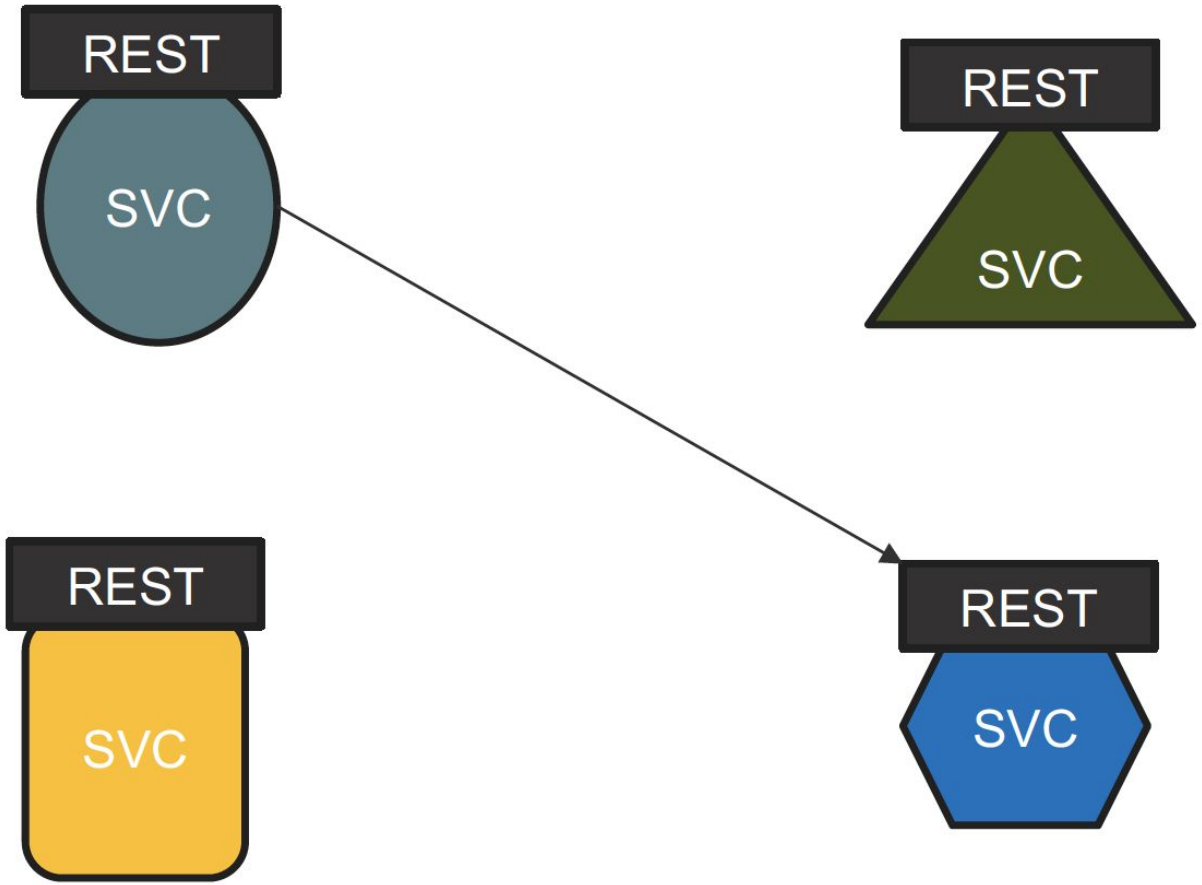
Jenkins



Design for Failure

- W architekturze mikroserwisowej dużo procesów I dużo ruchu sieciowego
- Dużo może pójść nie tak
- Kod musi ogarnąć takie błędy w prawidłowy sposób
- Trzeba logować i monitorować

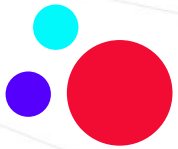
Design for Failure



Catch Exception

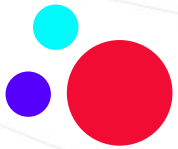
Retry

Log exception



Evolutionary Design

- Przejście na mikroserwisy powinno być postępowe
- Nie trzeba rozwalać wszystkiego i zaczynać od nowa
- Zmieniamy każdą część postępowo



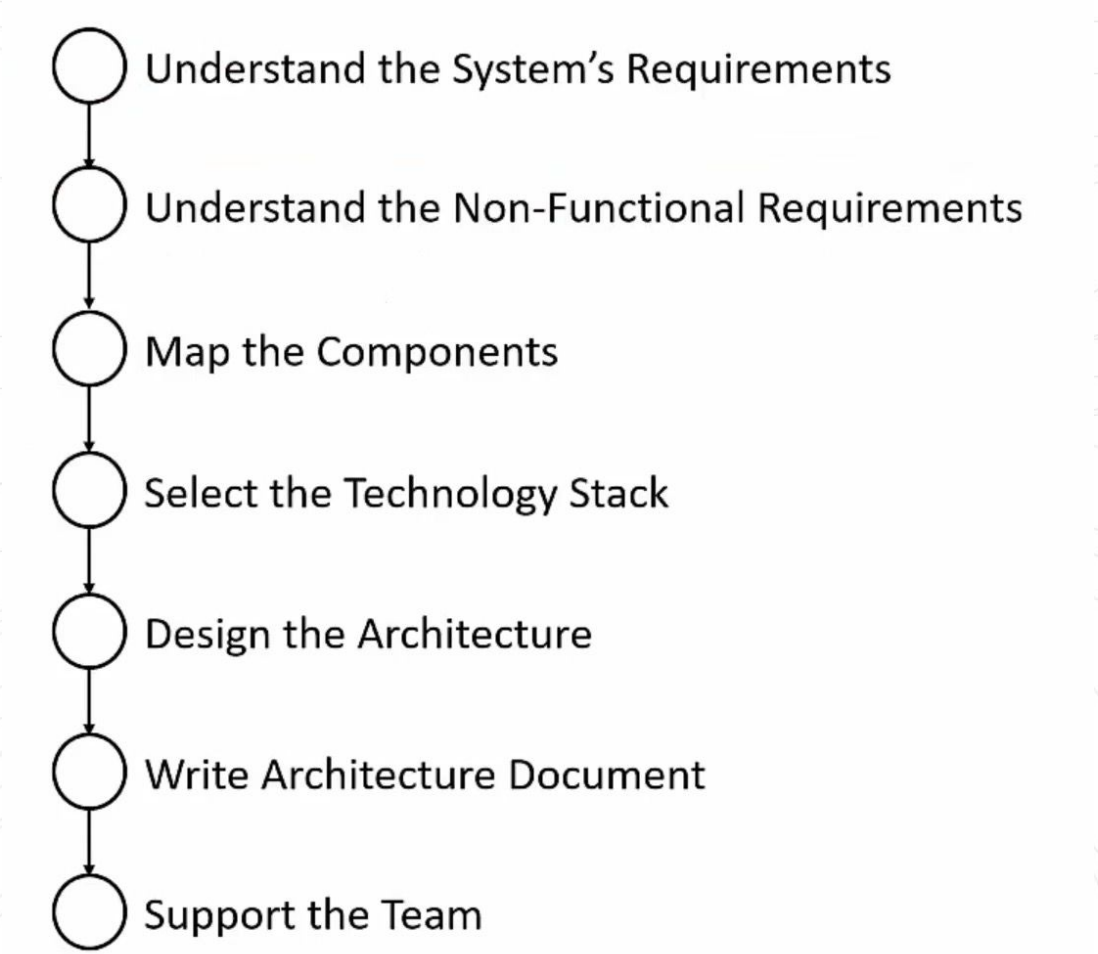
Problemy rozwiązywane za pomocą mikroserwisów

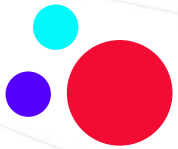
- Jedyna platforma technologiczna
- Nieelastyczny deployment
- Nieskuteczne zużycie zasobów
- Duża skala i skomplikowaność
- Drogie szyny serwisowe (ESB)
- Brak narzędzi

Projektowanie architektury mikroserwisowej

- Trzeba użyć metodologii
- Do not rush
- Więcej planuj, koduj mniej

Projektowanie architektury mikroserwisowej



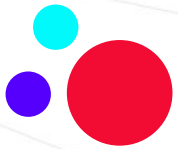


Projektowanie architektury mikroserwisowej. Mapowanie komponentów

- Najbardziej ważny krok całego procesu
- Definiuje jak system będzie wyglądał w przyszłości
- Ustalony - niełatwo zmienić

Mapowanie - definiowanie elementów systemu, z których tworzy się cały system.

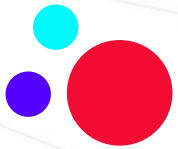
Komponenty = Serwisy



Projektowanie architektury mikroserwisowej. Mapowanie komponentów

Mapowanie musi bazować na:

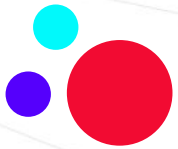
- **Wymaganiach biznesowych**
- Autonomia funkcjonalna
- Jednostki danych
- Autonomia danych



Projektowanie architektury mikroserwisowej. Mapowanie komponentów

Mapowanie musi bazować na:

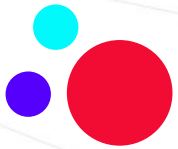
- Wymaganiach biznesowych
- **Autonomia funkcjonalna**
- Jednostki danych
- Autonomia danych



Projektowanie architektury mikroserwisowej. Mapowanie komponentów

Mapowanie musi bazować na:

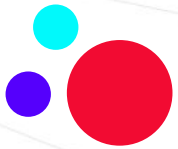
- Wymaganiach biznesowych
- Autonomia funkcjonalna
- **Jednostki danych**
- Autonomia danych



Projektowanie architektury mikroserwisowej. Mapowanie komponentów

Mapowanie musi bazować na:

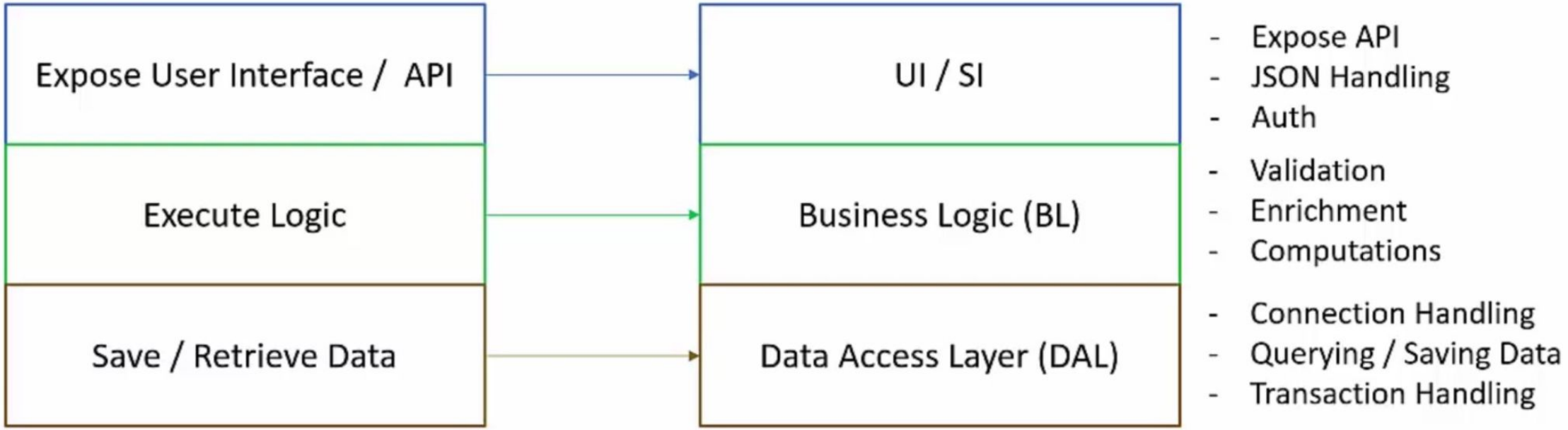
- Wymaganiach biznesowych
- Autonomia funkcjonalna
- Jednostki danych (zamówienia, towary)
- **Autonomia danych**



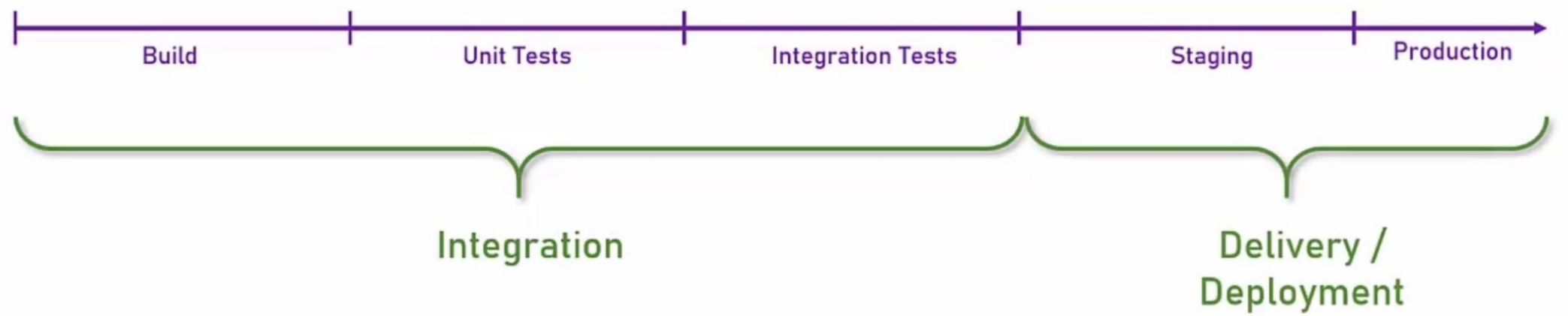
Projektowanie architektury mikroserwisowej. Wybór stosu technologicznego

	App Types	Type System	Cross Platform	Community	Performance	Learning Curve
.NET	All	Static	No	Large	OK	Long
.NET Core	Web Apps, Web API, Console, Service	Static	Yes	Medium and growing rapidly	Great	Long
Java	All	Static	Yes	Huge	OK	Long
node.js	Web Apps, Web API	Dynamic	Yes	Large	Great	Medium
PHP	Web Apps, Web API	Dynamic	Yes	Large	OK -	Medium
Python	All	Dynamic	Yes	Huge	OK -	Short

Projektowanie architektury mikroserwisowej.



Deploying Microservices. Integration and Delivery



Pozwala:

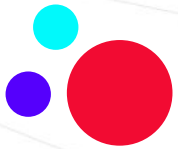
- Przyspieszyć release
- Zwiększyć niezawodność
- Na raportowanie

Deploying Microservices. Kontenery

Klasyczny deployment:

Kod skopiowany i zbudowany na prodzie

Są problemy/bugi znalezione na prodzie, zamiast test/stage środowiskach



Deploying Microservices. Kontenery

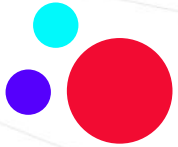
Kontenery:

Cienki model opakowania

Pakuje aplikacje, zależności oraz pliki konfiguracyjne

Może być kopiowany pomiędzy serwerami

Używa systemu operacyjnego host'a



Deploying Microservices. Kontenery

Dlaczego “tak” dla kontenerów:

Przewidywalność

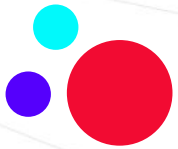
Wydajność

Gęstość

Dlaczego “nie” dla kontenerów:

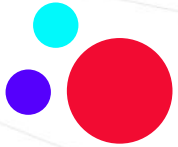
Izolacja

infoShare ACADEMY



Praktyki

- Infrastruktura mikroserwisowa na przykładzie CloudRun
- In-memory DBs
- Queues and brokers



Testing Microservices

- Testowanie ważne we wszystkich systemach i dla wszystkich typów architektur
- Z mikroserwisami jeszcze bardziej
- Testowanie mikroserwisowej architektury posiada własne wezwania

Typy testów:

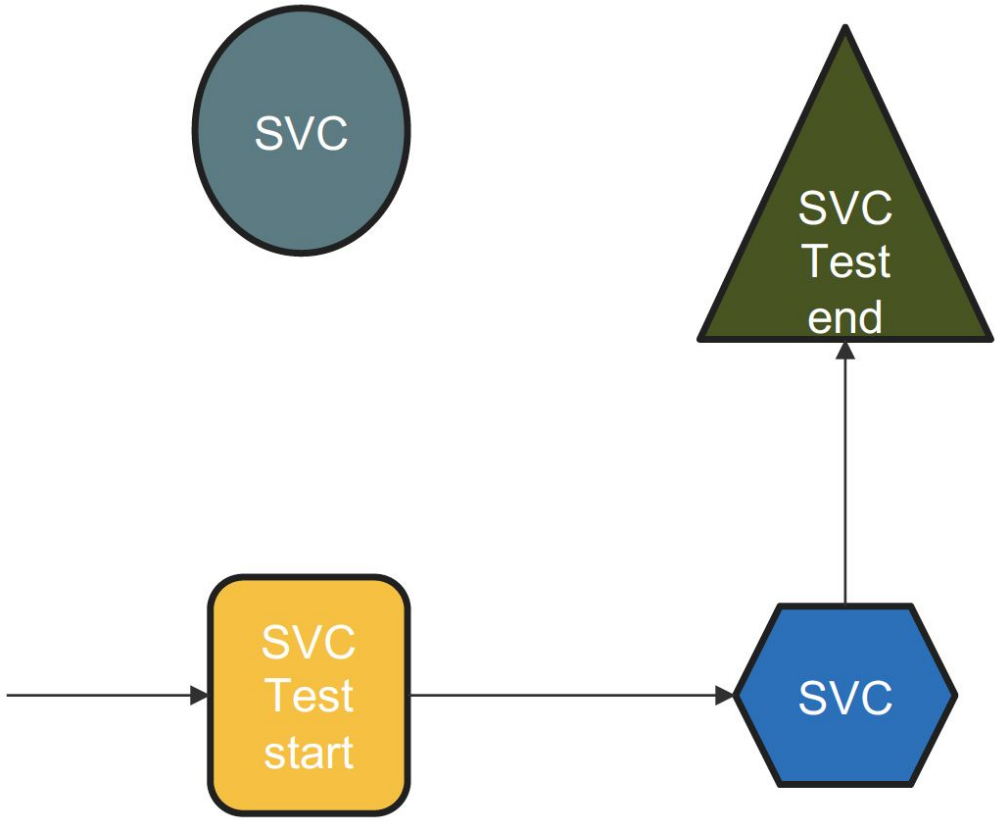
Unit

Integration

End-to-End

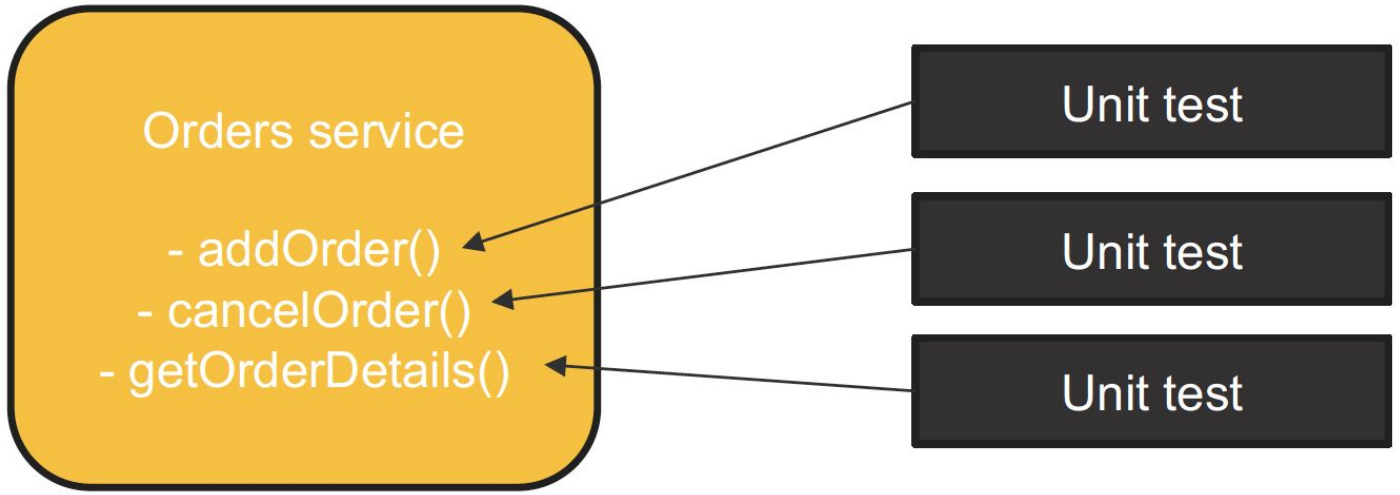
Challenges with Microservices Testing

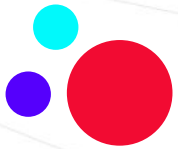
- Ruchome części
- Testowanie nie jest trywialne:
  - “cross” - testowanie serwisów
  - “nonfunctional” zależności



Unit tests

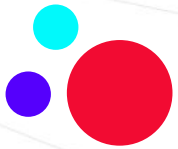
- Testowanie indywidualnych jednostek kodu (metody, interface'y)
- Zautomatyzowane
- Tworzone przez deweloperów





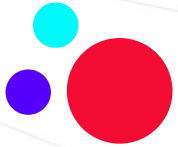
Integration tests

- Testowana funkcjonalność
- Nie interesuje nas jak serwis jest napisany
- Próbujemy pokryć całość kodu w serwisie
- Testujemy również zewnętrzne zależności (bazy danych, inne serwisy)



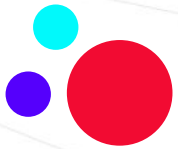
End-to-End tests

- Testujemy cały “flow”
- Sprawdzamy wszystkie serwisy
- Testy bardzo kruche
- Wymagają bezpośredniego dostępu do systemu (do db na przykład)
- Zwykle pokrywają tylko główne scenariusze



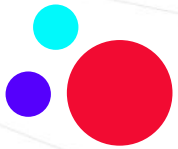
Service Mesh

- Zarządza komunikacją pomiędzy serwisami
- Dostarcza dodatkowe serwisy
- Platform-agnostic



Jaki problem rozwiązuje Service Mesh

- Mikroserwisy gadają pomiędzy sobą
- W komunikacji mogą być problemy:
  - Timeouty
  - Bezpieczeństwo
  - Retries
  - Monitoring



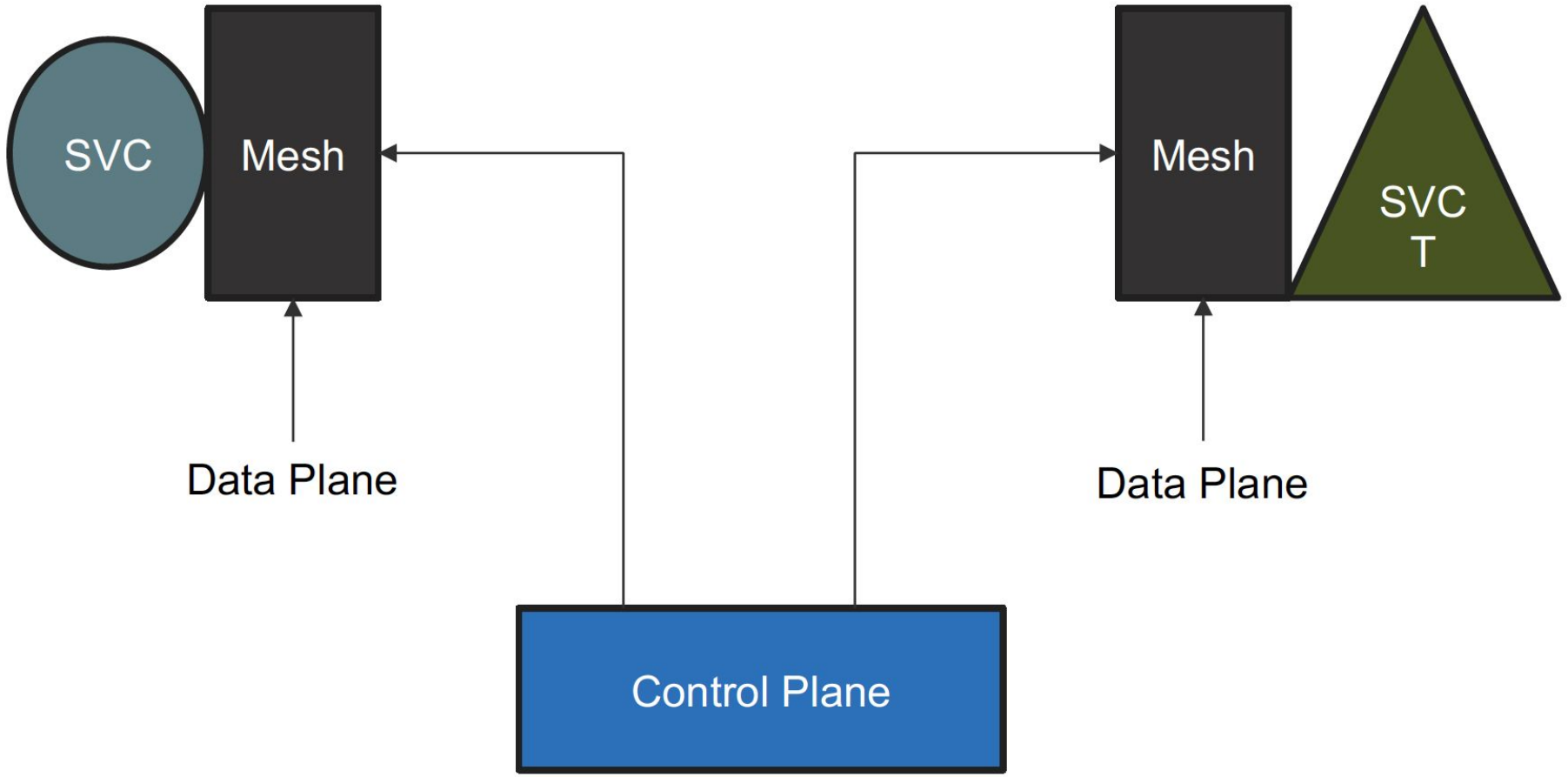
Service Mesh

- Zbiór komponentów, którzy znajdują się blisko usługi I zapewniają komunikację
- Zabezpiecza całość komunikacji
- Serwisy komunikują wyłącznie przez service mesh

Zabezpiecza:

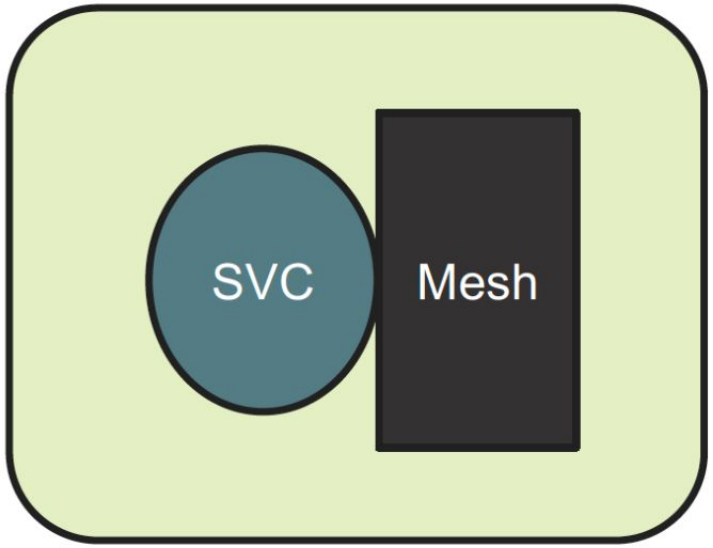
- Konwersję protokołów
- Bezpieczeństwo
- Uwierzytelnianie
- Niezawodność

Architektura Service Mesh



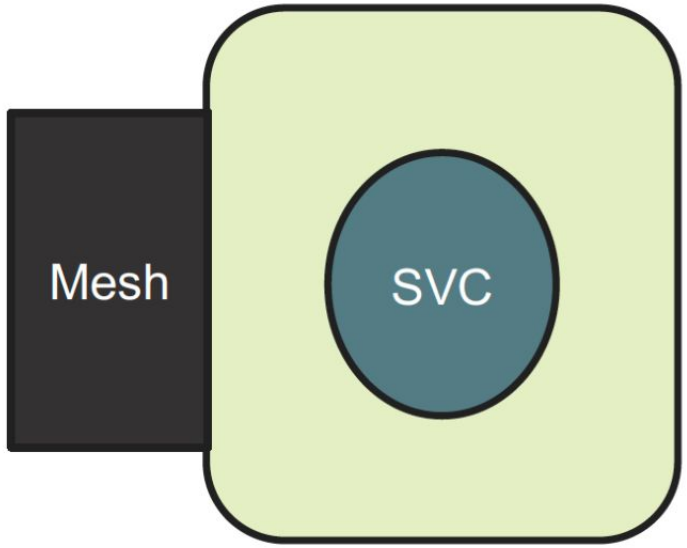
Rodzaje Service Mesh

In-process



Wydajny

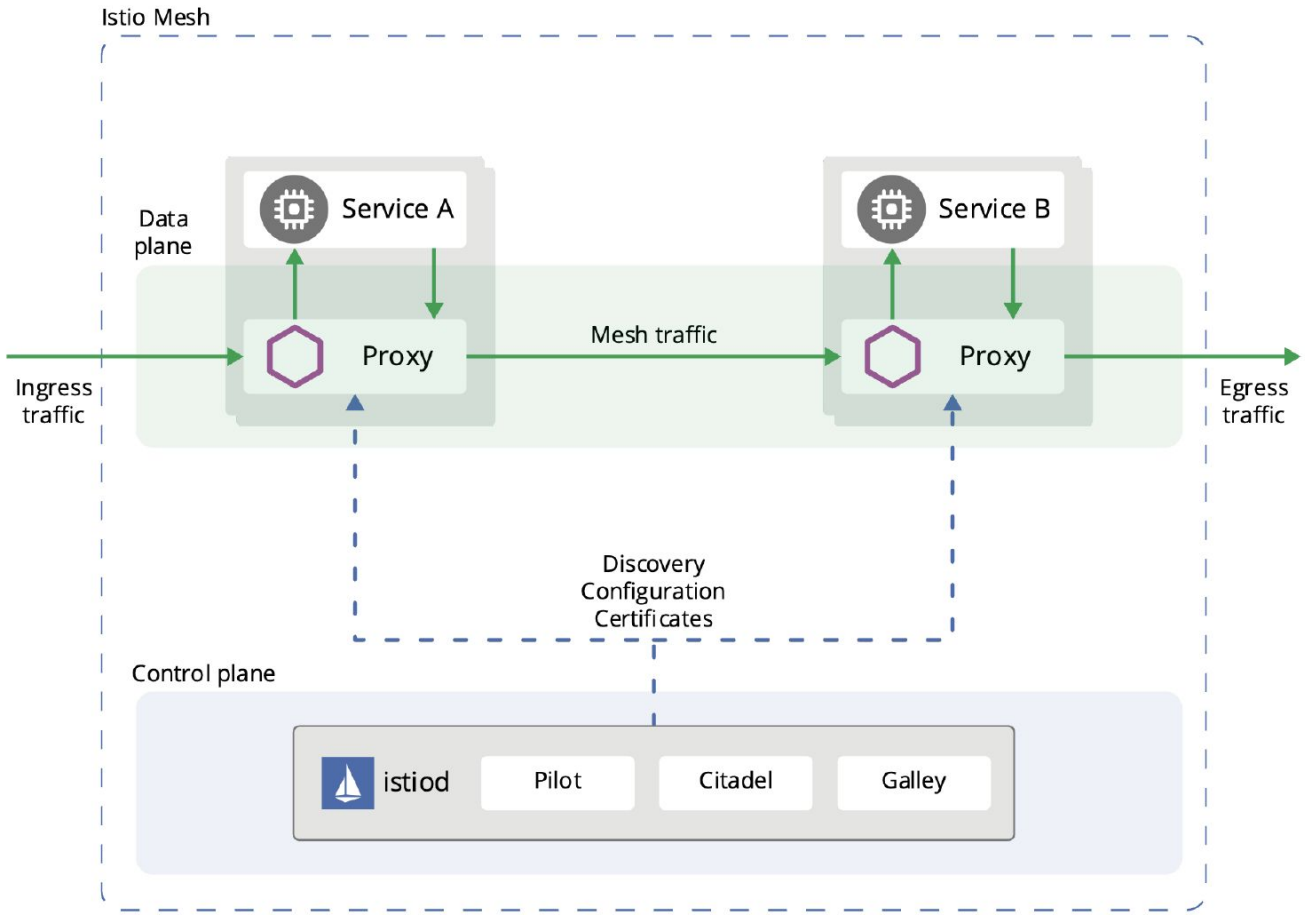
Sidecar

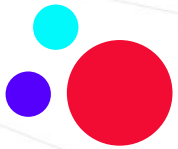


Platform Agnostic

Code Agnostic

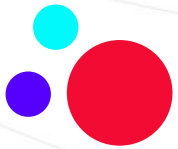
Istio Service Mesh





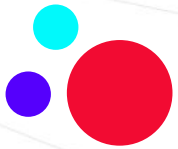
Demo: Anthos Service Mesh

- ASM - Istio



Logging and Monitoring

- Bardzo ważny
- Flow - przez wiele serwisów
- Ciężko otrzymać całościowy obraz
- Ciężko zrozumieć, co się dzieje z usługą



Logging vs Monitoring

Logging

Notuje aktywności systemowe

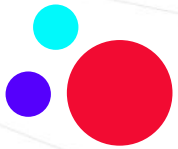
Zajmuje się audytem

Notuje błędy

Monitoring

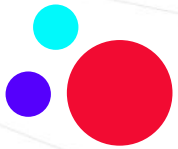
Kontroluje metryki systemowe

Alertuje w przypadku anomalii



Logging

- Logować jak najwięcej
- Logować cały end-to-end flow
- Pokazuje całościowy widok na system

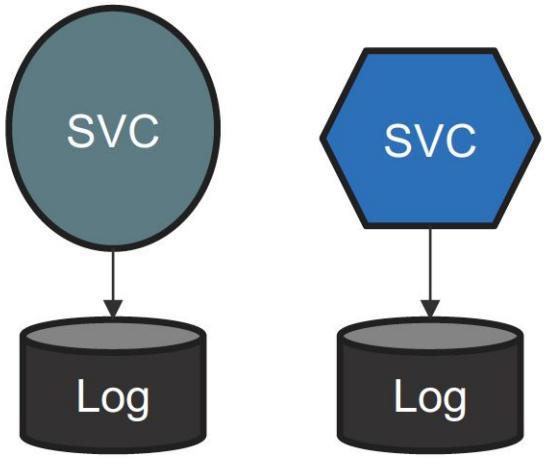


Logging and Monitoring

ActivityID	ServiceName	Logs
1234	ApiGateway	
1234	Payments	
1234	SMS	Exception
1234	GSM Operator	500

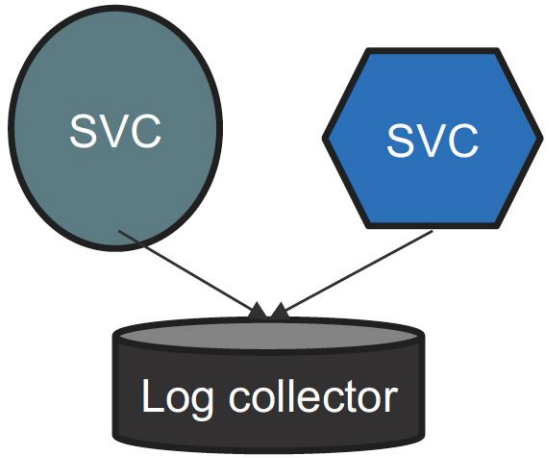
https://opentelemetry.io

Logging



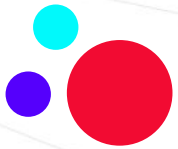
Klasyczny system

Osobne logi
Różne formaty
Nie zbierane
trudno analizować



Microservice'owy system

Uniwersalne
Agregowane
Łatwo analizowane



Monitoring

- Monitoring kontroluje metryki, wyświetla anomalie
- Udostępnia prosty widok stanu systemów
- Alertuje, kiedy jest problem

Monitoring

Status	Objective	Type	Alerts firing	Error budget
Healthy	98% - Distribution Cut - Calendar week	Request Based SLI		EDIT DELETE

Service-level indicator
99.3%

Error budget
99.8%

Alerts firing
0/0

Service Level Indicator

Graph represents the current fraction of successful requests to your service

Unhealthy	98% - Distribution Cut - Calendar week	Request Based SLI	1/1	99.8%	

Rodzaje monitoringu

Infrastruktury

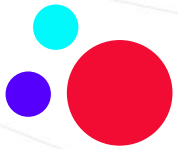
Monitoruje system.
Alertuje problemy z:
CPU
RAM
Dysk
Sieć

Źródło: agent

Aplikacja

Monitoruje aplikacje.
Kontroluje:
Zapytania
Zamówienia
Zamówienia za godzinę
Alertuje problemy z aplikacją

Źródło: logi aplikacji, wydarzeń



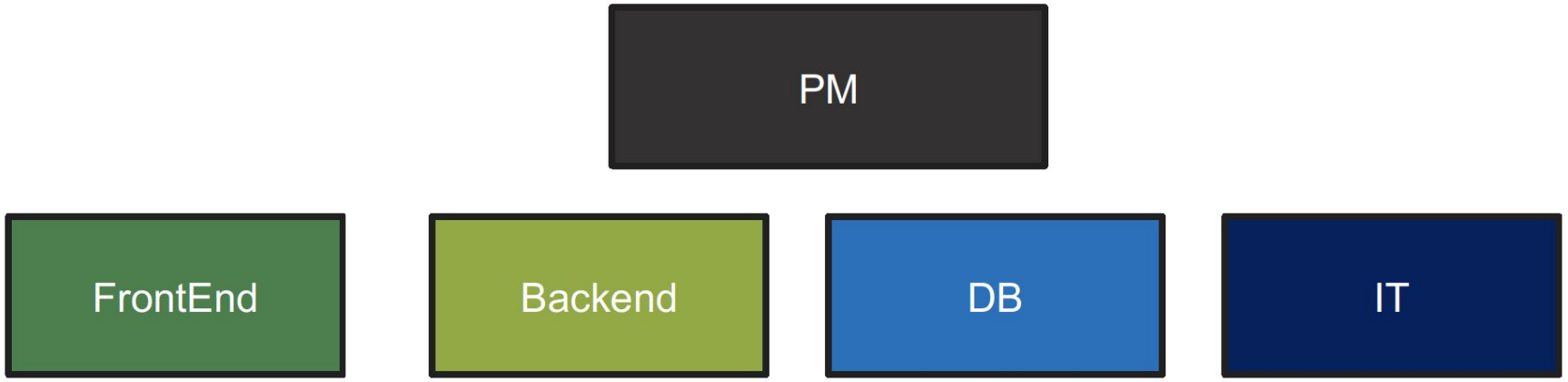
Kiedy nie używać microservice'ów

- Małe systemy
- Przeplatanie logiki aplikacji oraz danych
- Systemy wrażliwe na wydajność
- POC systems
- Embedded systems ("no-update" systems)

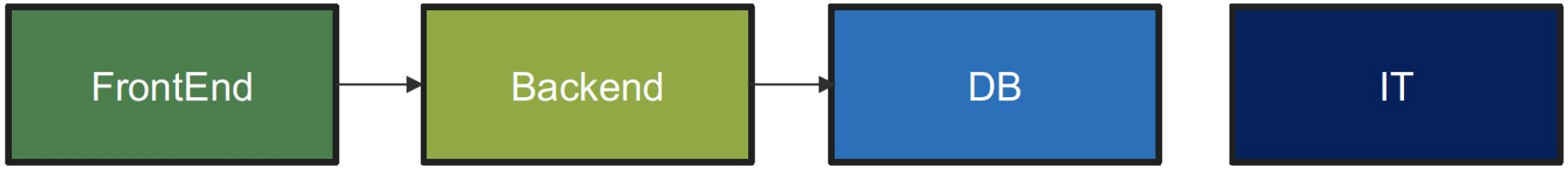
Microservices and Organization

Conway's Law:

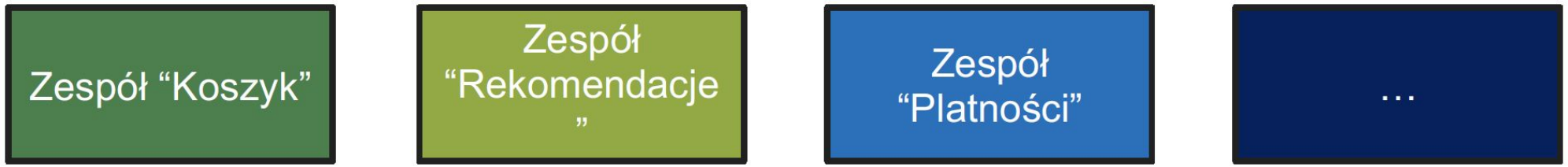
"Organizations design systems that mirror their own communication structure"



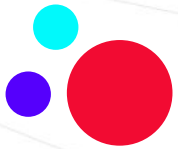
Microservices and Organization



Projektowe podejście



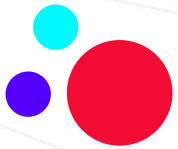
Serwisowe podejście



Anti-Patterns and Common Mistakes

Pamiętamy:

- Microservice'y wymagają dobrego projektowania
- Microservice'y nie "Fire and Forget"
- Łatwo zrobić błąd, który rozwali cały projekt

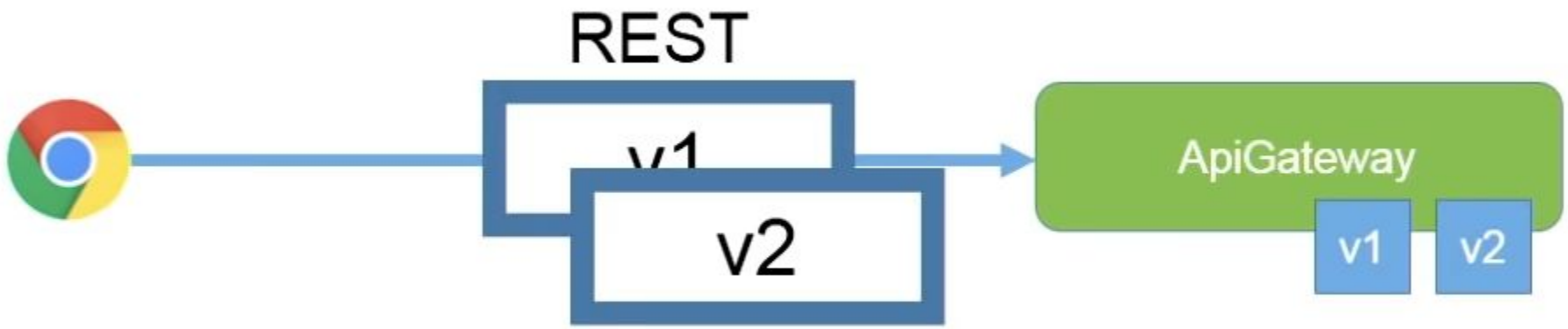


Anti-Patterns and Common Mistakes

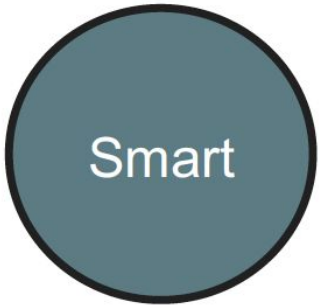
- Brak dobrze zdefiniowanych serwisów
- Brak dobrze zdefiniowanego API
- Istnieją pomiędzy serwisowe zależności

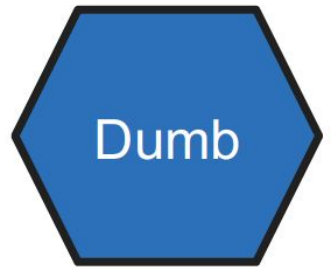
Functionality	Path	Return Codes
Get next list to be processed	`GET /api/v1/lists/next?location=…`	`200 OK` `400 Bad Request`
Mark item as collected / unavailable	`PUT /api/v1/list/{listId}/item/{itemId}`	`200 OK` `404 Not Found`
Export list's payment data	`POST /api/v1/list/{listId}/export`	`200 Ok` `404 Not Found`

Anti-Patterns and Common Mistakes



Types of Services



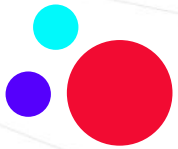


Payments:

Logika

Data Access Layers

SMS



Migracja Monolitu do Architektury microservice'owej

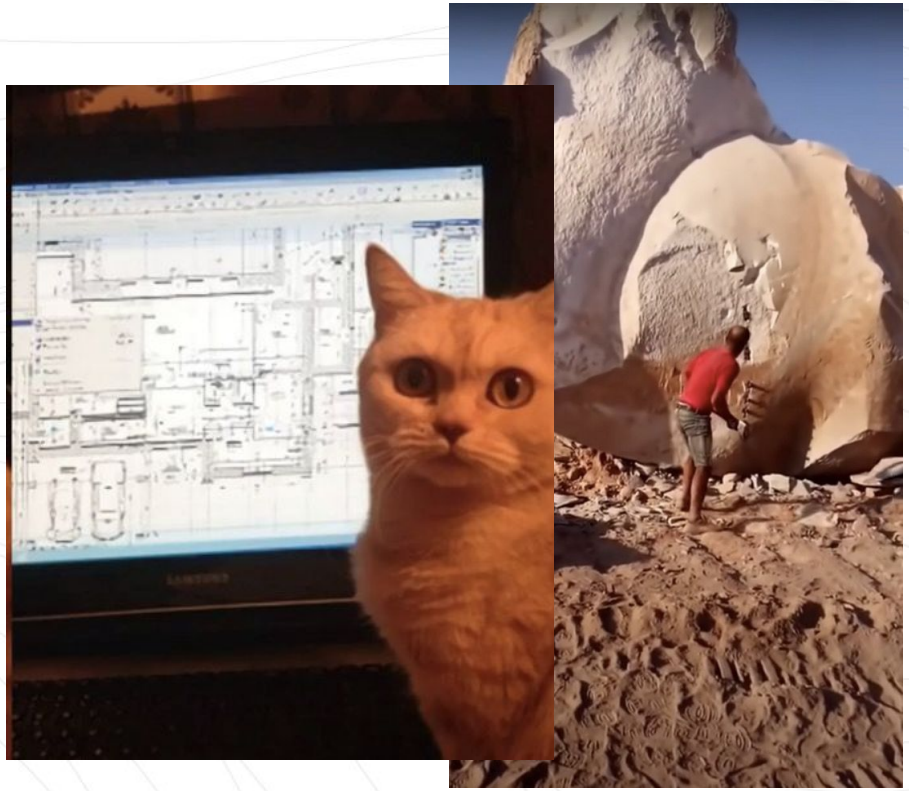
To pozwoli na:

- Skrócenie cyklu developmentu oraz update
- Modularyzację systemu
- Zmniejszenie kosztów
- Modernizację systemu
- Marketingowe zalety?

Strategie migracji

Rozdzielenie jest procesem delikatnym

Trzeba do dobrze planować



3 główne strategie:

- Nowe moduły jako serwisy
- Wydzielenie istniejących modułów
- Pełne przepisywanie

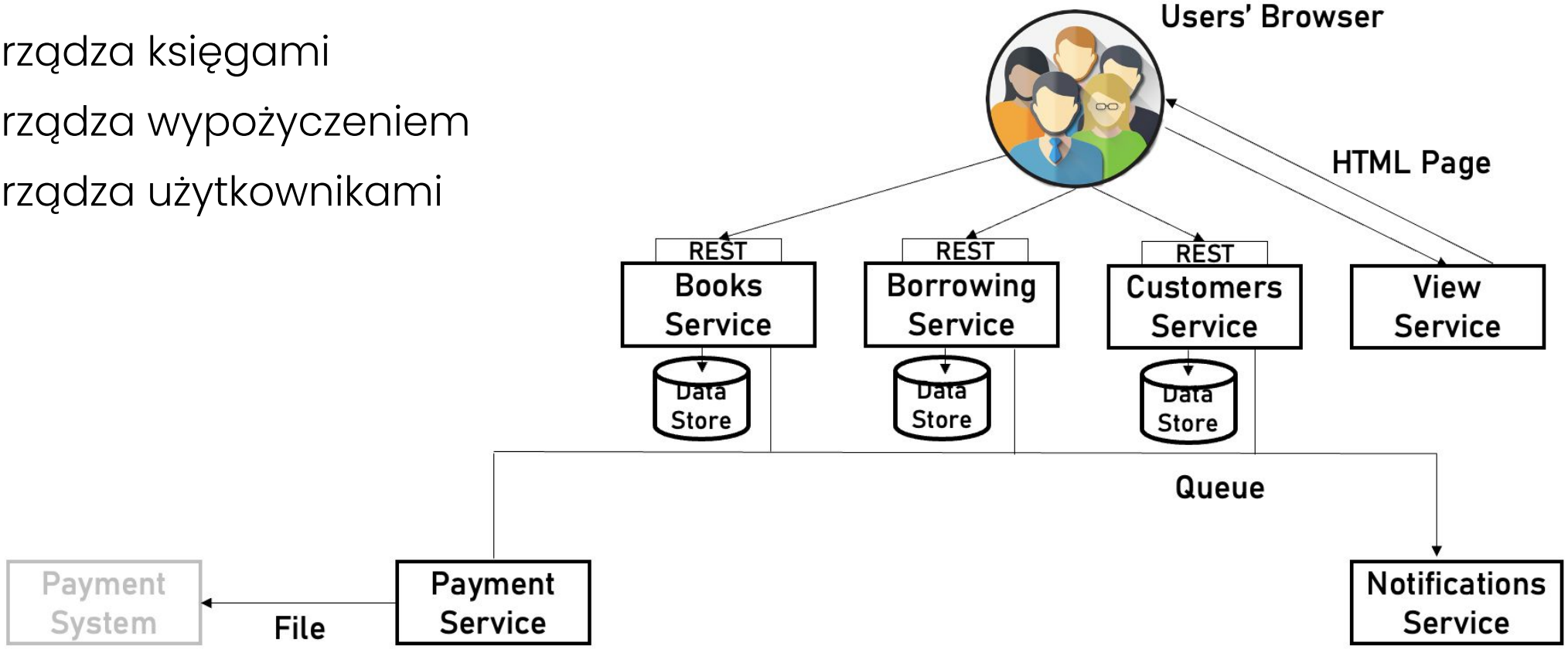
Case study

Library Management

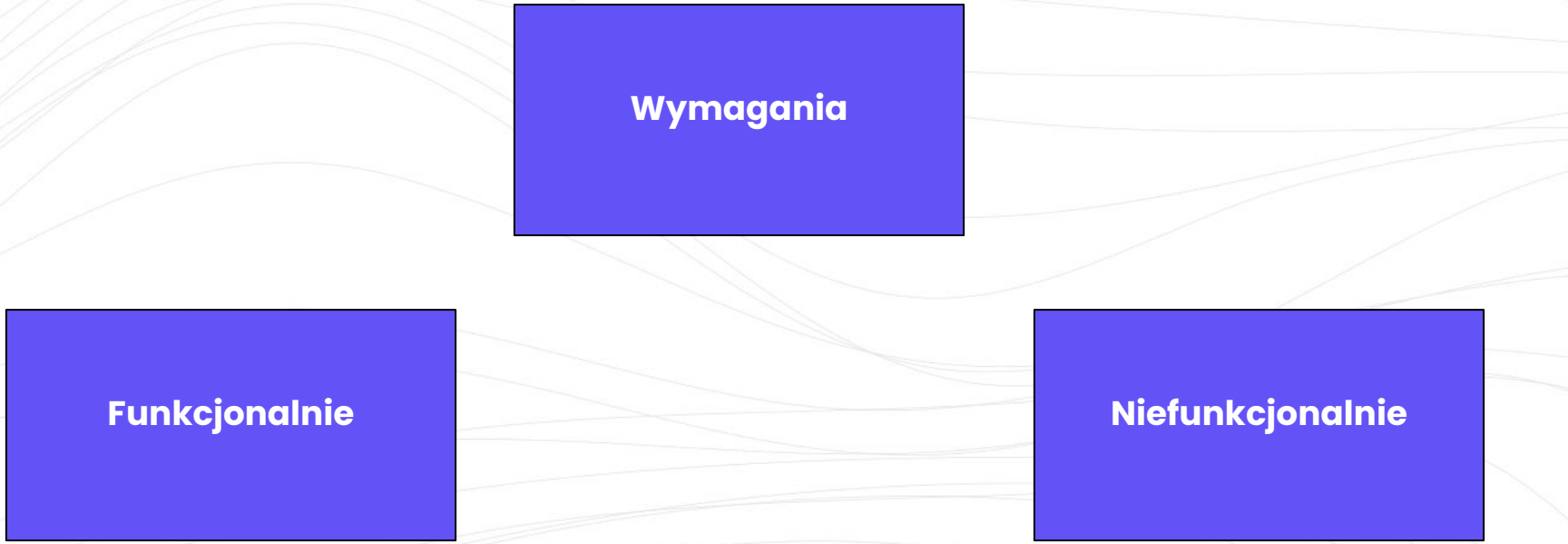
Zarządza księgami

Zarządza wypożyczeniem

Zarządza użytkownikami

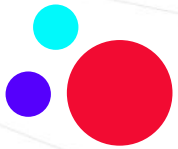


Case study



Co musi robić system:
Musi być “web-based”
Zarządzać księgami
Zarządzać wypożyczaniem
Zarządzać użytkownikami
Pokazywać powiadomienia
Kasować miesięczną opłatę

Z czym musi współpracować system



Case study

Książki

Powiadomienia

Wypożyczanie

Płatności

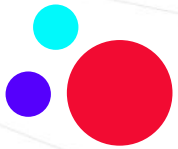
Użytkownicy

Web-page

infoShare ACADEMY

Migrate VM to Service

- Practice



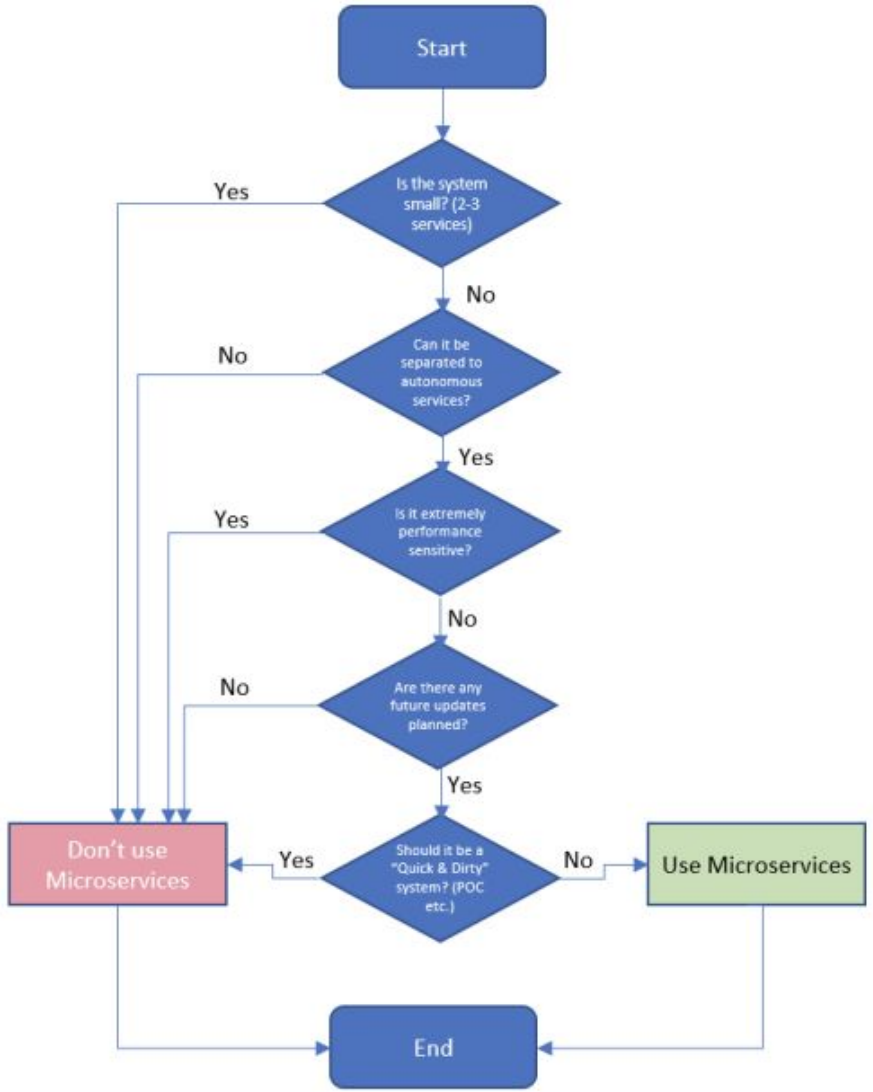
Microservices Security

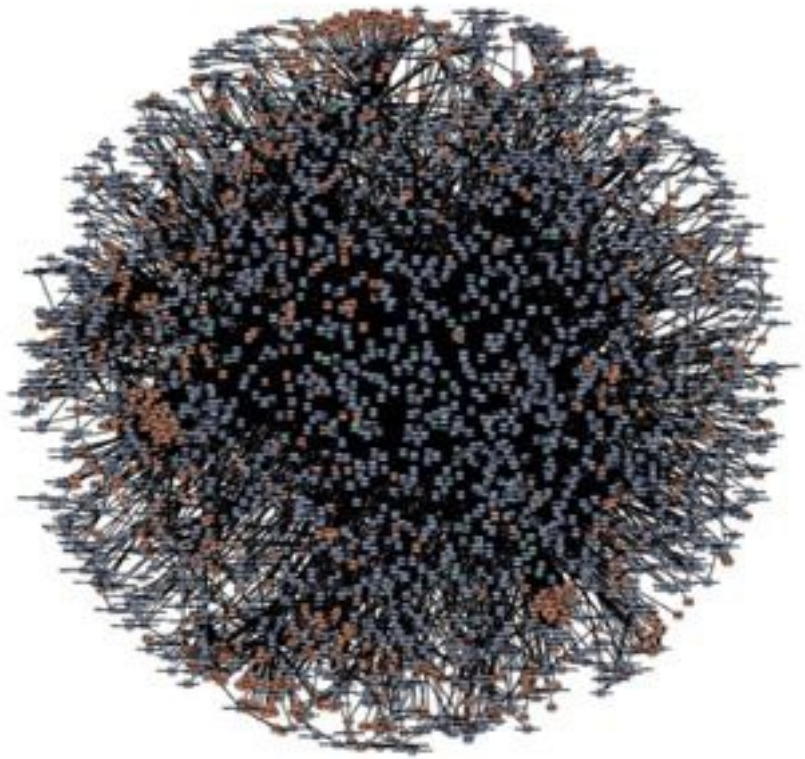
- Secure by design
- Scan for dependencies
- Https everywhere
- Access and Identity tokens
- Encrypt and protect secrets
- Slow down your attackers

Bonus

Microservices

Flow Chart

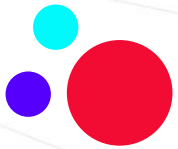




amazon.com

NETFLIX

infoShare ACADEMY

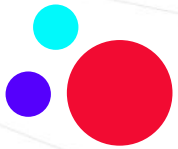


Przydatne linki

https://martinfowler.com/articles/microservices.html

https://encyclopedia2.thefreedictionary.com/WS*+protocols

https://techdozo.dev/grpc-for-microservices-communication/



Microservices Recap

Microservice'y są super:

1. Microservice'y ułatwiają development
2. Istnieje kilka należytych zespołów
3. Microservice'y ułatwiają deployment
4. Różnorodne skalowanie
5. Odporność na problemy
6. Mamy ZOO w stosie

Microservice'y są "evil":

1. Microservice'y utrudniają development
2. Microservice'y wymagają infrastruktury
3. Microservice'y utrudniają deployment

Dziękuję za uwagę!

infoShareAcademy.com