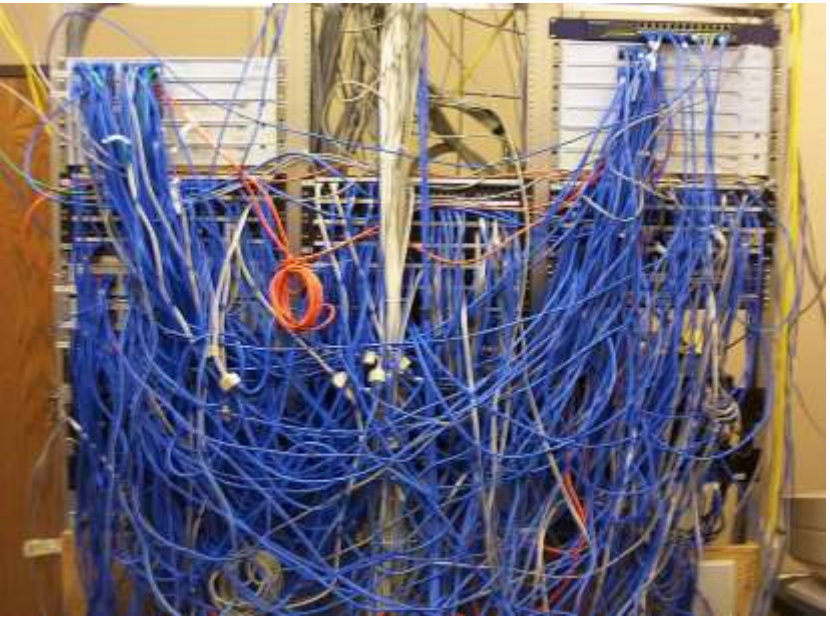
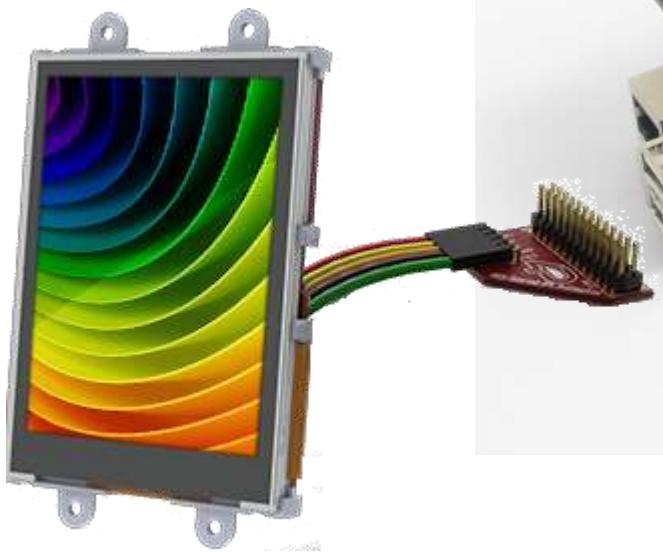
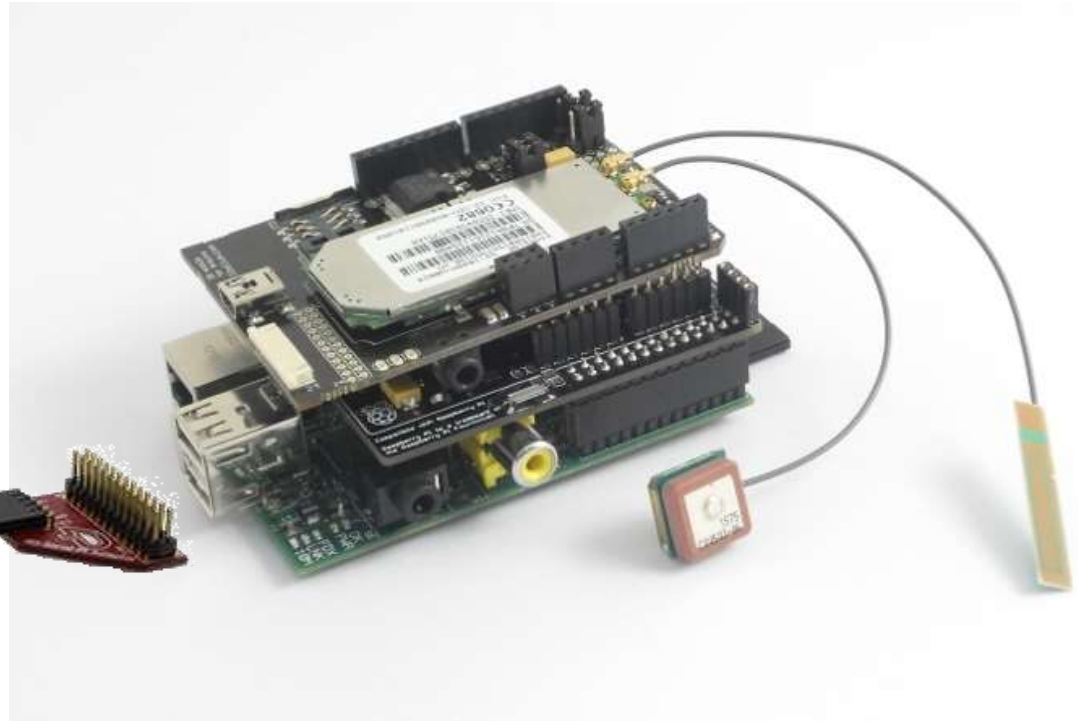
# Projektowanie architektoniczne


GLENNZ







# Podręcznik projektowania architektoniczno-budowlanego

- - Normy podstawowe
  - Podstawy wymiarowania. Proporcje
  - Projektowanie
  - Realizacja budowy
  - Części budynku
  - Ogrzewanie. Wentylacja
  - Fizyka budowli. Ochrona budowli
  - Oświetlenie. Światło dzienne. Szkło
  - Okna. Drzwi
  - Schody. Dźwigi
  - Ulice
  - Ogrody
  - Dom. Pomieszczenia gospodarcze
  - Pomieszczenia mieszkalne
  - Pływalnie
  - Pralnie
  - Balkony
  - Drogi
  - Domy letniskowe
  - Rodzaje zabudowy mieszkaniowej
  - Schrony
  - Modernizacja starych budynków
  - Szkoły
  - Szkoły wyższe. Laboratoria

- - Ośrodki dla dzieci
  - Biblioteki. Biura. Banki
  - Budynki użyteczności publicznej
  - Sklepy
  - Magazyny
  - Warsztaty. Zakłady przemysłowe
  - Zmiana przeznaczenia
  - Budownictwo wiejskie
  - Koleje
  - Parkingi. Garaże. Stacje paliw
  - Porty lotnicze
  - Gastronomia
  - Hotele. Motele
  - Zoo
  - Teatry.Kina
  - Urządzenia sportowe
  - Szpitale
  - Domy seniora
  - Kościoły.Muzea
  - Cmentarze
  - Ochrona przeciwpożarowa
  - Miary. Normy

# Pojęcia architektoniczne

- **System**
  - projekt nad którym pracujemy (kontekst) - aplikacji
  - np. dom
- **Podsystem**
  - Część systemu, która jest na tyle nie zależna, że może być systemem w swoim kontekście
  - np. kuchnia, pokój
- **Moduł**
  - Część systemu, która nie ma znaczenia poza jego kontekstem. Moduł jest hermetyczny i posiada API.
  - np. noga od stołu

# Pojęcia architektoniczne

- **Komponent**
  - jednostka oprogramowania zastępowana i aktualizowana niezależnie od systemu, hermetyzujący zbiór powiązanych funkcji (i/lub danych)
    - **Biblioteka** – komponent łączony z programem i wywoływany za pomocą wywołań bezpośrednich
    - **Usługa** – komponent pozaprocesowy z którym komunikacja wymaga protokołu takiego jak np. usługi Internetowe czy inną formę zdalnego wywołania.
  - W tej definicji komponent może być **implementacją tak podsystemu jak i modułu**

# Architektura oprogramowania

- Projektowanie architektoniczne
  - Proces projektowania, którego celem jest identyfikacja podsystemów, na które należy podzielić system, oraz szkieletu pozwalającego na kontrolę oraz wzajemną komunikację podsystemów.
- Architektura oprogramowania
  - Rezultat projektowania architektonicznego

# Abstrakcje architektoniczne

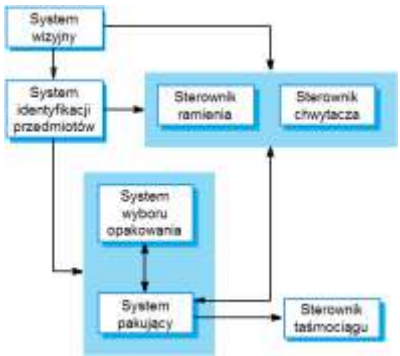
- **Architektura małej skali (ang. Architecture in the small)**
  - Architektura pojedynczych programów. Sposób w jaki pojedynczy program dekomponowany jest do poziomu mniejszych elementów (komponentów).
- **Architektura wielkoskalowa (ang. Architecture in the large)**
  - Dotyczy architektury systemów korporacyjnych, które składają się z systemów, aplikacji, komponentów. Systemy korporacyjne są rozproszone w sposób dopuszczający sytuację , w której poszczególne jego elementy mogą być zarządzane i stanowić własność różnych organizacji.

# Zalety jawnej architektury

- Porozumienie stron
  - Architektura może zostać wykorzystana jako element dyskusji pomiędzy zainteresowanymi stronami.
- Analiza systemu
  - W znaczeniu analizy mającej dać odpowiedź na pytanie czy system będzie w stanie zrealizować wymagania niefunkcjonalne.
- Wielokrotne użycie w wielkiej skali
  - Sama architektura może być podstawą wielokrotnego użycia
  - Koncepcja architektury linii produkcyjnej.

# Reprezentowanie (dokumentowanie) architektury

- Najczęściej
  - proste, nieformalne diagramy blokowe ukazujące jednostki systemu oraz relacje pomiędzy nimi.
  - Abstrakcyjne - nie pozwalają na reprezentację natury powiązań pomiędzy komponentami oraz widocznych (zewnętrznie) cech podsystemów.
  - Użyteczne dla celów porozumienia pomiędzy stronami oraz planowania projektu



# Architektura a oczekiwane cechy systemu

1. Efektywność
   - Gruboziarnistość, minimalizacja komunikacji między modułami.
2. Pewność (ang. safety)
   - Lokalizacja krytycznych elementów w małej liczbie komponentów.
3. Bezpieczeństwo (ang. security)
   - Warstwowość, lokalizacja krytycznych zasobów w warstwach wewnętrznych.
4. Dostępność
   - Nadmiarowe komponenty, mechanizmy tolerowania awarii.
5. Zdatność do pielęgnacji
   - Drobnoziarnistość, wymienne komponenty.

# Wzorce architektoniczne

# Wzorce architektoniczne

- Wzorce są sposobem reprezentowania, współdzielenia i wielokrotnego użycia **wiedzy**.
- Wzorzec (styl) architektoniczny to stylizowany opis dobrej praktyki projektowej, która została wypróbowana  i przetestowana w różnych środowiskach.
- Opis wzorca powinien określać jego stosowalność (kiedy warto a kiedy nie)
- Wzorzec może być reprezentowany w postaci tabelarycznej oraz graficznej.

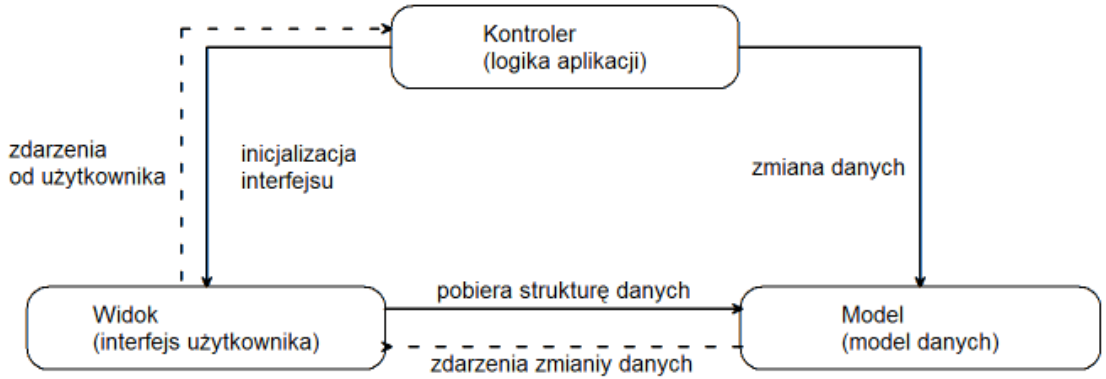
# Wzorzec Model-Widok-Kontroler

## Model-View-Controller (MVC)

| Nazwa | MVC (Model-View-Controller) |
|---|---|
| **Opis** | Oddziela prezentację oraz interakcję od danych systemu. System podzielony jest na trzy, oddziaływujące na siebie, logiczne komponenty:<br>Model – to komponent zarządzający danymi oraz operacjami na nich wykonywanymi.<br>Widok – to komponent definiujący oraz zarządzający sposobem prezentacji danych użytkownikom.<br>Kontroler – to komponent zarządzający interakcją użytkownika z aplikacją (np. kliknięcie myszą, naciśnięcie przycisku, itp.), który przekazuje interakcję tak do Widoku, jak i Modelu. |
| **Przykład** | Następny slajd pokazuje architekturę aplikacji webowej zorganizowanej z wykorzystaniem wzorca MVC. |
| **Stosowalność** | Wiele widoków oraz mechanizmów interakcji powiązanych z danymi.<br>Możliwe zmiany w wymaganiach dotyczących sposobu interakcji z danymi oraz metod ich prezentacji. |
| **Zalety** | Pozwala na niezależne zmiany w obrębie danych systemu oraz w obrębie ich reprezentacji.<br>Ułatwia prezentowanie tych samych danych na różne sposoby oraz pozwala na zmianę wszystkich widoków w odpowiedzi na modyfikację danych (np. wykonaną za pośrednictwem jednego z widoków). |
| **Wady** | W przypadku gdy dane oraz model interakcji użytkownika z aplikacją jest prosty może zwiększać złożoność kodu. |

# Wzorzec MVC

Mapuje akcje użytkownika
na aktualizacje modelu,
wybiera widoki

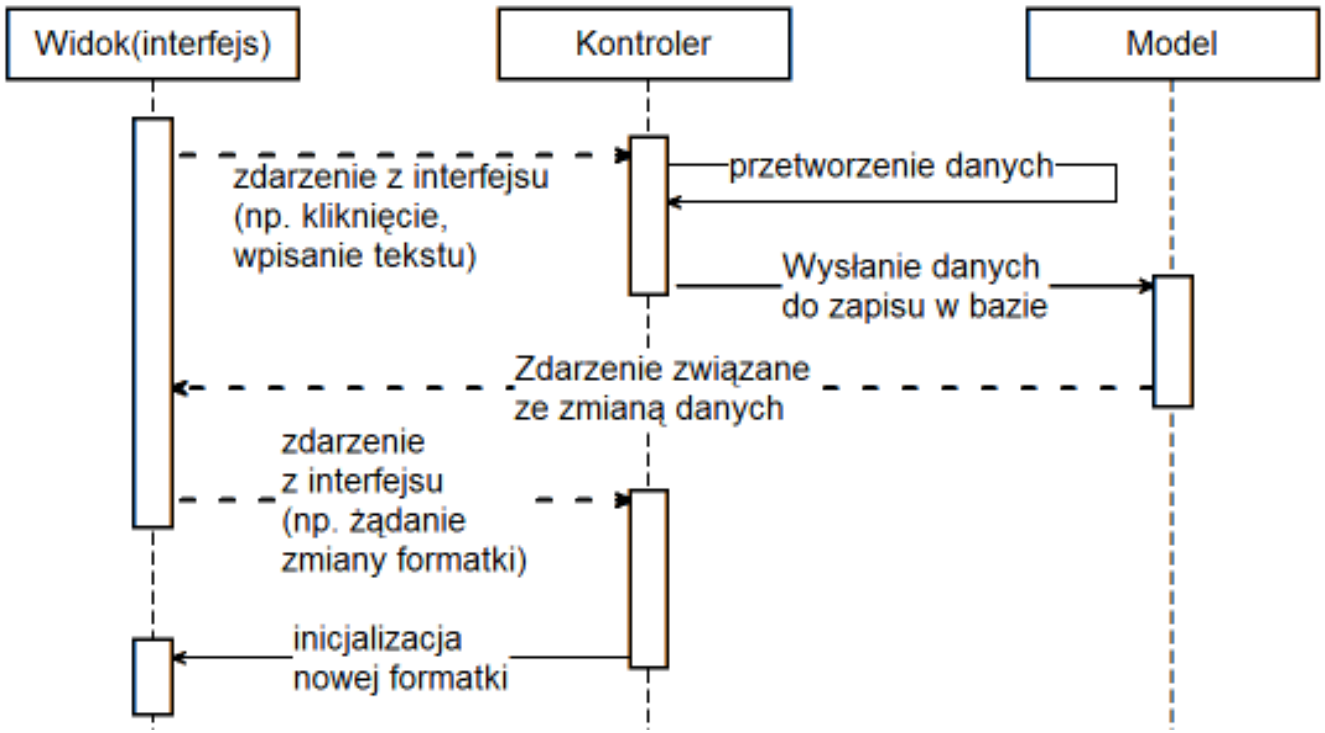



Renderuje dane,
wysyła żądania aktualizacji danych,
Wysyła informacje o zdarzeniach od użytkownika

Hermetyzuje stan aplikacji,
Informuje widoki o zmianie stanu


Architektura jest jak mechanizm przekładni z kół zębatych

# Wzorzec MVC...


Widok(interfejs)
Kontroler
Model
zdarzenie z interfejsu
(np. kliknięcie,
wpisanie tekstu)
przetworzenie danych
Wysłanie danych
do zapisu w bazie
Zdarzenie związane
ze zmianą danych
zdarzenie
z interfejsu
(np. żądanie
zmiany formatki)
inicjalizacja
nowej formatki

# Architektura warstwowa Layered architecture

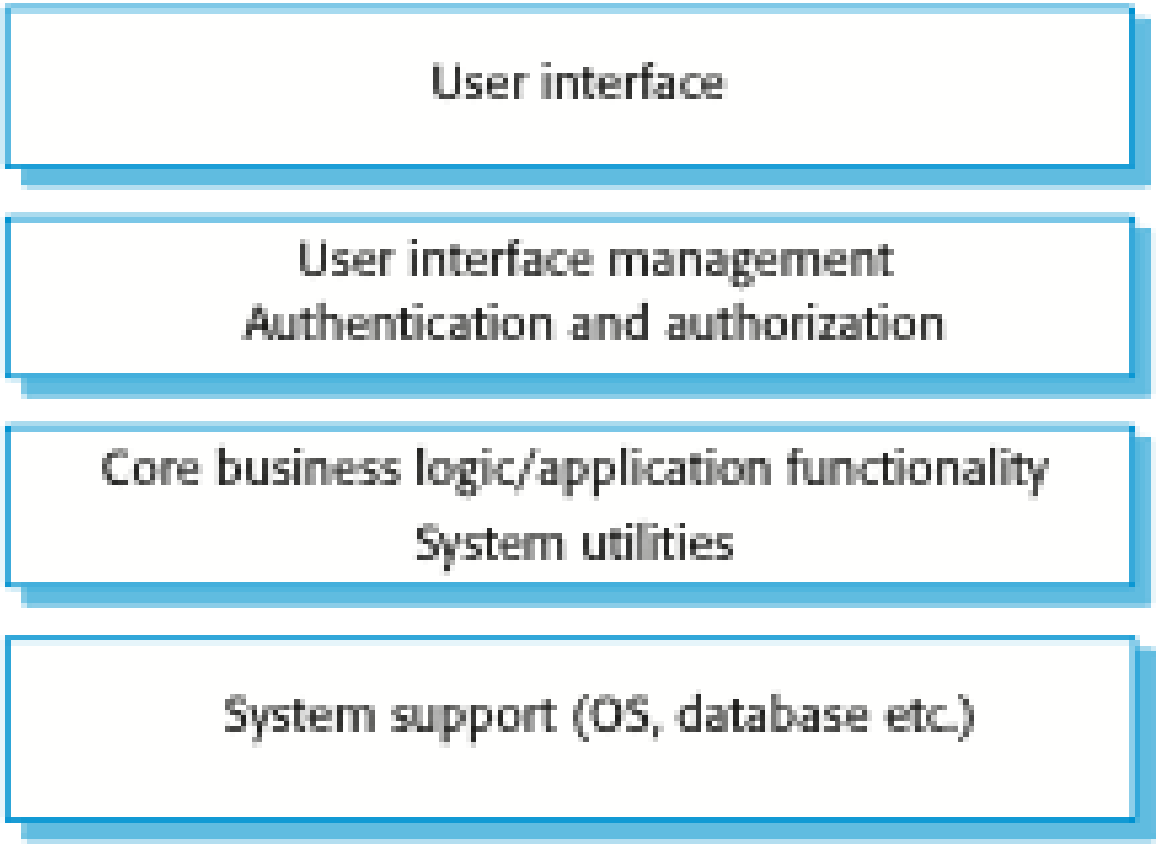
- Wykorzystywana do modelowania interfejsów podsystemów.
- Organizuje system w zbiór warstw (albo abstrakcyjnych maszyn). Każda z nich udostępnia zestaw usług.
- Ułatwia przyrostową realizację podsystemów w różnych warstwach. Zmiana interfejsu warstwy ma wpływ tylko na warstwy przylegające.
- Często powoduje sztuczną strukturalizację systemu.


Architektura jest jak Ogry (i cebule) – ma warstwy ☺

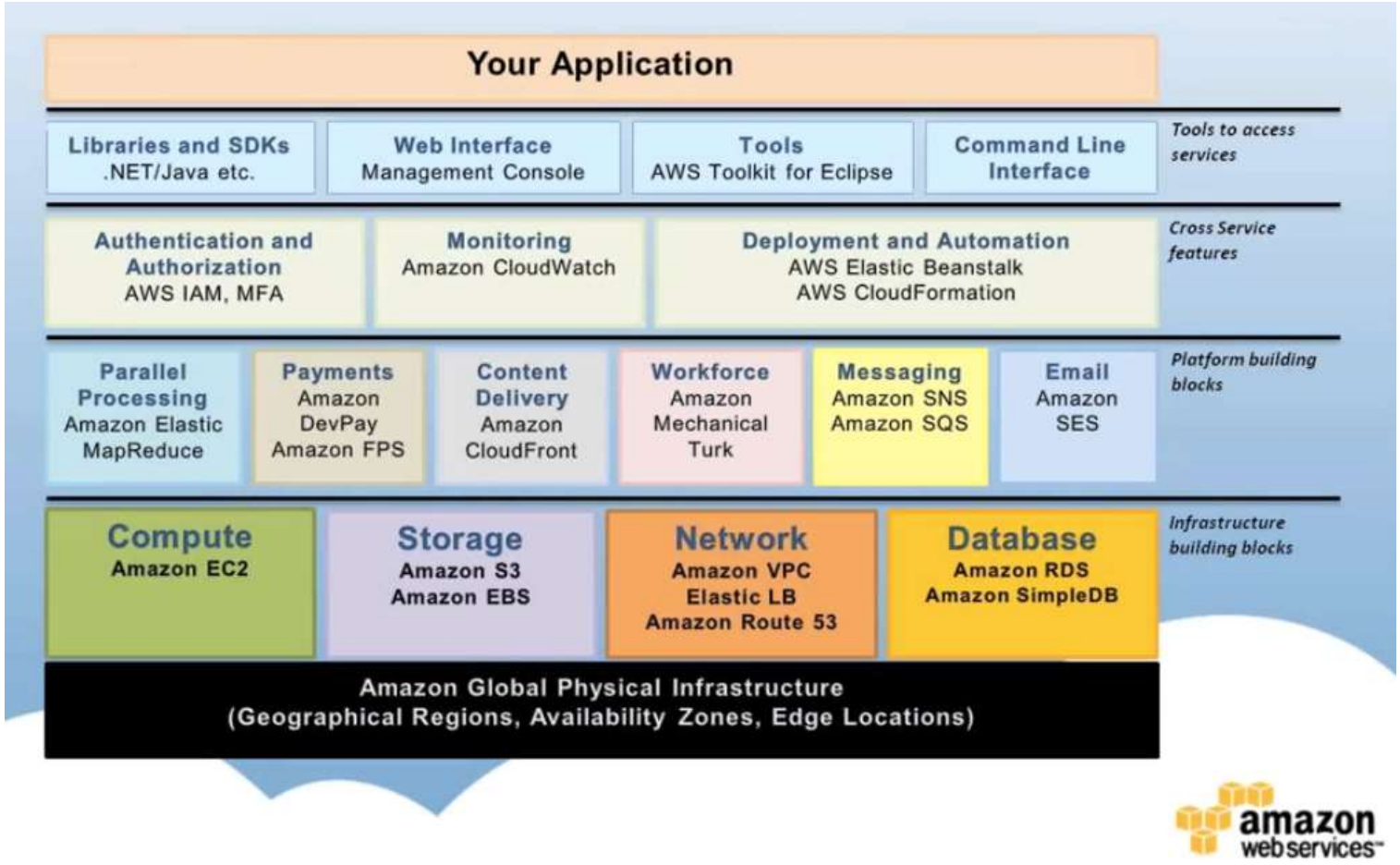
# Wzorzec architektury warstwowej

| Nazwa | Layered Architecture |
|---|---|
| **Opis** | Organizuje system w warstwy. Powiązana funkcjonalność znajduje się w określonej warstwie. Warstwa udostępnia usługi warstwie znajdującej się bezpośrednio nad nią. Warstwa położona najniżej reprezentuje podstawowe usługi, najczęściej wykorzystywane w systemie. |
| **Przykład** | Warstwowy model systemu umożliwiający współdzielenie dokumentów chronionych prawami autorskimi, przechowywanymi w różnych bibliotekach. |
| **Stosowalność** | Budowanie nowych elementów na „czubku” istniejących systemów. Rozwój jest podzielony pomiędzy odrębne zespoły i każdy z zespołów odpowiedzialny jest za określoną warstwę. Istnieje wymaganie wielopoziomowych zabezpieczeń. |
| **Zalety** | Pozwala na podmianę całej warstwy tak długo, jak interfejs pozostaje bez zmian. Nadmiarowe udogodnienia mogą być wprowadzane do każdej warstwy (np. uwierzytelnianie) w celu zwiększenia pewności systemu. |
| **Wady** | W praktyce pełna realizacja założeń wzorca jest trudna i może prowadzić do problemów z wydajnością (potrzeba przetwarzania żądania usługi przez każdą warstwę). |

# Generyczna architektura warstwowa



# „Warstwy chmury"



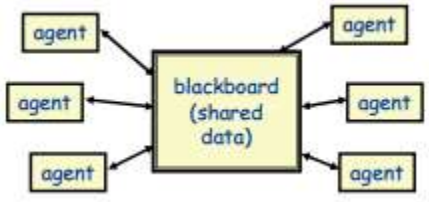

amazon web services™

# Hexagon (ports and adapters)

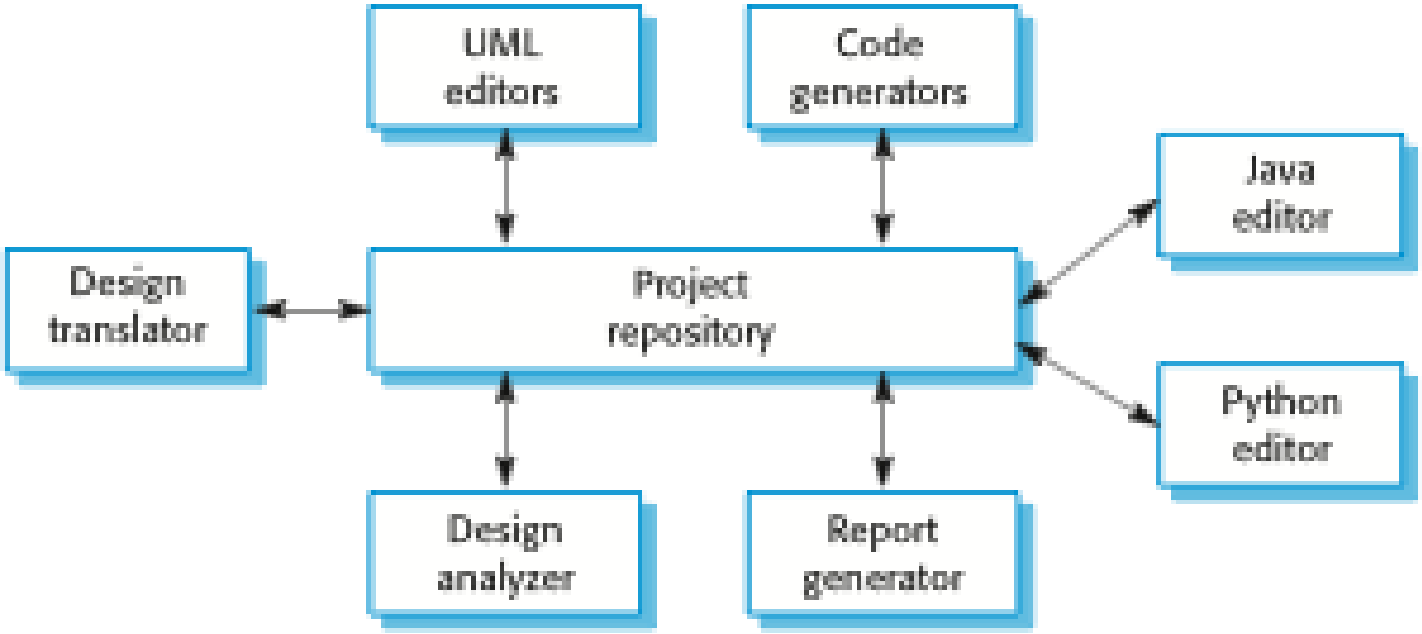
# Architektura warstwowa vs Architektura hexagonalna

# Wzorzec Repozytorium (Blackboard)

| Nazwa | Repository |
|---|---|
| **Opis** | Zarządzanie wszystkimi danymi systemu odbywa się za pośrednictwem centralnego repozytorium. Repozytorium dostępne jest dla wszystkich komponentów systemu. Komponenty współdziałają ze sobą tylko za pośrednictwem repozytorium. |
| **Przykład** | Następny slajd prezentuje przykład IDE, którego komponenty wykorzystują repozytorium informacji o projekcie. Każde narzędzie wchodzące w skład IDE generuje informacje, które mogą być wykorzystane przez pozostałe. |
| **Stosowalność** | System, w którym generowane są duże ilości danych, które muszą być przechowywane przez długi czas.<br>Systemy sterowane danymi, w których wprowadzenie danych wyzwala akcje (np. uruchamia narzędzie). |
| **Zalety** | Komponenty systemu mogą być niezależne – nie musza wiedzieć o swoim istnieniu. Zmiany wprowadzone przez jeden z komponentów mogą być propagowane do innych. Dane mogą być zarządzane w spójny sposób (np. kopie zapasowe wykonywane w tym samym czasie). |
| **Wady** | Repozytorium jest pojedynczym punktem awarii (single point of failure) – problem z repozytorium wpływa na cały system.<br>Organizowanie całej komunikacji za pośrednictwem repozytorium może być nieefektywne. Rozproszenie repozytorium na kilka komputerów może być trudne. |



# Architektura Repozytorium dla IDE

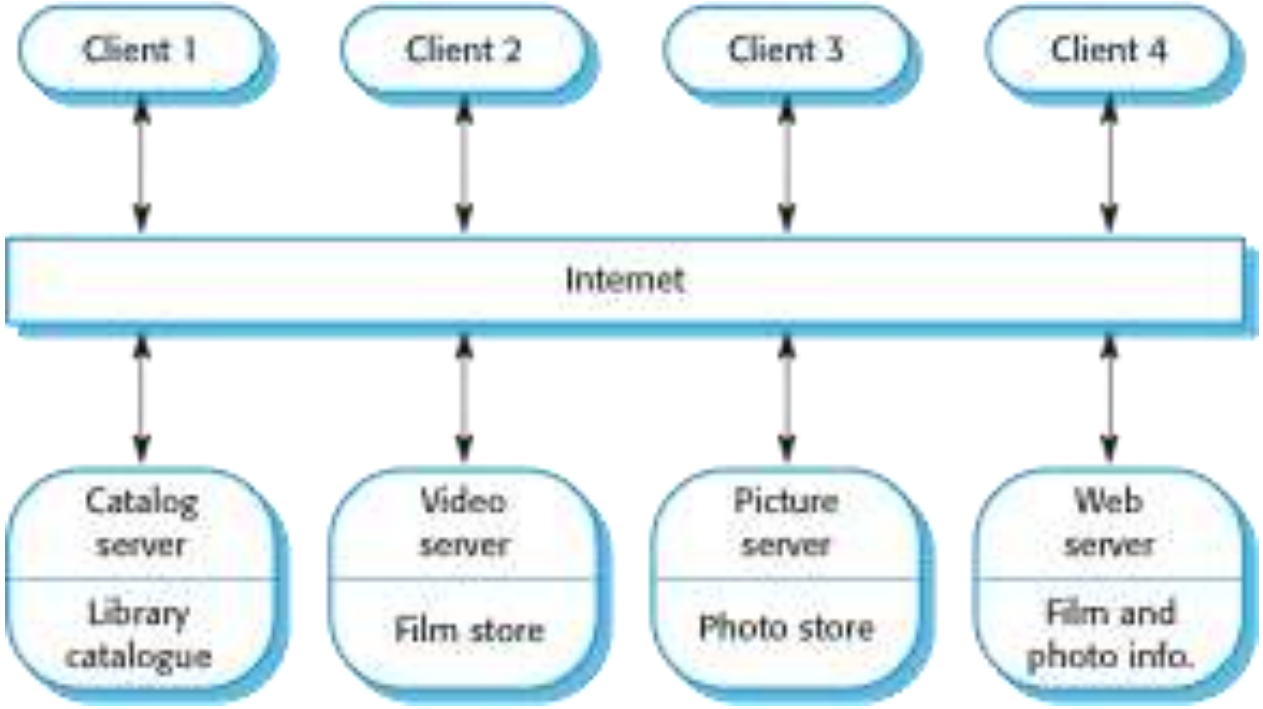



Architektura jest jak „lodówkowa” tablica ogłoszeń

# Wzorzec klient-serwer client–server

| Nazwa | Client-server |
|---|---|
| **Opis** | W architekturze klient-serwer funkcjonalność systemu reprezentowana jest przez usługi. Każda usługa udostępniana jest przez odrębny serwer. Klienci są użytkownikami usług i, aby je wykorzystać, uzyskują dostęp do serwerów. |
| **Przykład** | Następny slajd prezentuje przykład usług wykorzystującej architekturę klient-serwer. |
| **Stosowalność** | Dane przechowywane we współdzielonej bazie danych muszą być dostępne z dla różnych lokalizacji. Ponieważ serwery mogą podlegać replikacji, możliwe jest stosowanie w warunkach zmiennego obciążenia. |
| **Zalety** | Serwery mogą być rozmieszczone w sieci komputerowej. Ogólne funkcjonalności (np. usługa drukowania) mogą być dostępne dla wszystkich klientów i nie muszą być implementowane przez wszystkie usługi systemu. |
| **Wady** | Każda usługa jest pojedynczym punktem awarii. Wydajność może być nieprzewidywalna – zależna jest nie tylko od systemu ale również od sieci. Możliwe problemy z zarządzaniem jeżeli właścicielami serwery są różne organizacje. |

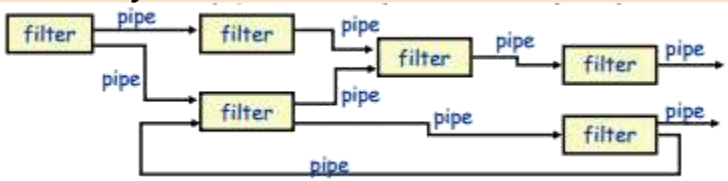
# Architektura klient-serwer wypożyczalni filmów



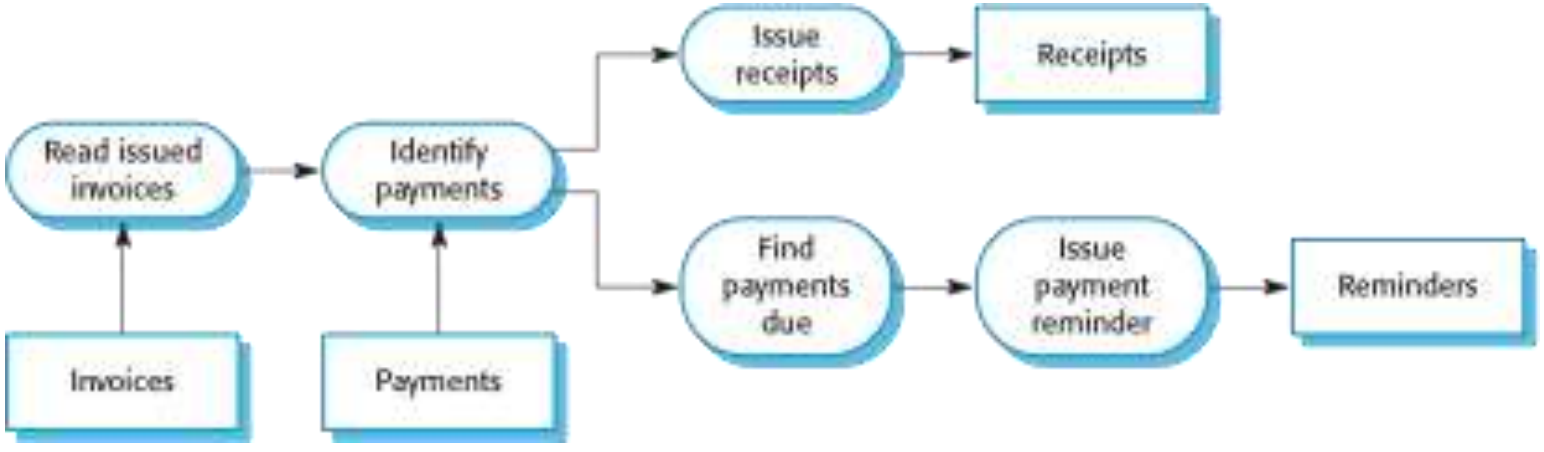

Architektura jest jak rynek


# Wzorzec Potoku pipe and filter

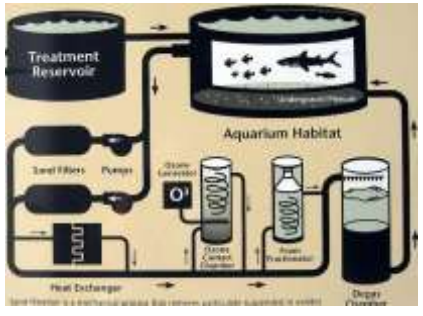
| Nazwa | Pipe and filter |
|---|---|
| **Opis** | Przetwarzanie danych w systemie zorganizowane jest w taki sposób, że każdy komponent przetwarzający dane (filtr) jest niezależny i wykonuje jeden typ transformacji. Dane podczas przetwarzania przepływają od jednego komponentu do kolejnego (jak w potoku, rurze). |
| **Przykład** | Następny slajd prezentuje przykład systemu potokowego do przetwarzania faktur. |
| **Stosowalność** | Szerokie zastosowanie w systemach przetwarzania danych (tak wsadowych jak i transakcyjnych), gdzie generację danych wyjściowych można oprzeć na serii niezależnych kroków przetwarzania danych wejściowych. |
| **Zalety** | Zrozumiałość, łatwość modyfikowania i wielokrotnego użycia.<br>Styl przepływu prac (workflow) odpowiadający strukturze wielu rzeczywistych procesów biznesowych.<br>Ewolucja systemu polega na dodaniu transformacji.<br>Może być zaimplementowana tak sekwencyjnie jak i współbieżnie. |
| **Wady** | Komunikujące się transformacje musza mieć uzgodniony format danych.<br>Każda transformacja musi parsować dane wejściowe z- oraz formatować dane wejściowe do- uzgodnionego formatu. Powoduje to zwiększenie obciążenia systemu i może oznaczać, że nie jest możliwe ponowne wykorzystanie transformacji, które używają niezgodnych struktur danych. |



# Przykład architektury potokowej



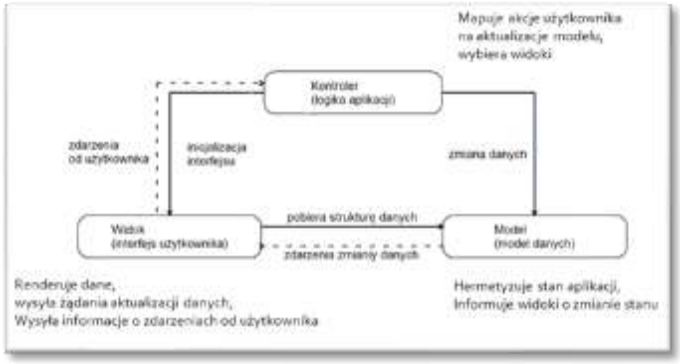

Architektura jest
jak filtr akwarystyczny

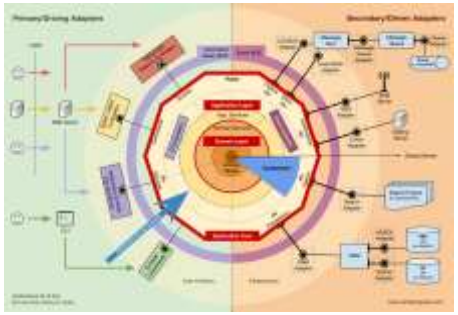


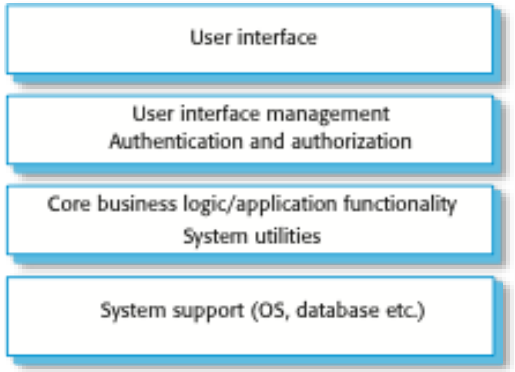
# Architektura zdarzeniowa

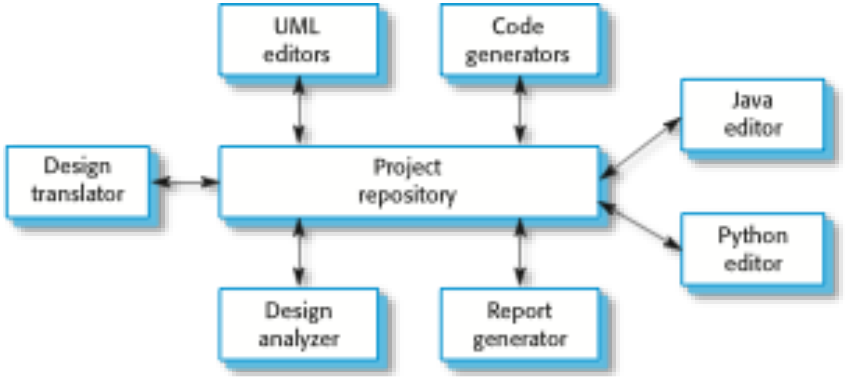
- Model systemu typu bodziec-reakcja, którego działanie opiera się na reakcjach na niezdeterminowane czasowo (asynchroniczne) zdarzenia.
- System składa się z luźno powiązanych komponentów (lub usług).
- Komponenty mogą emitować zdarzenia (agenci, emitenci, wydawcy) oraz je konsumować (konsument, subskrybent).

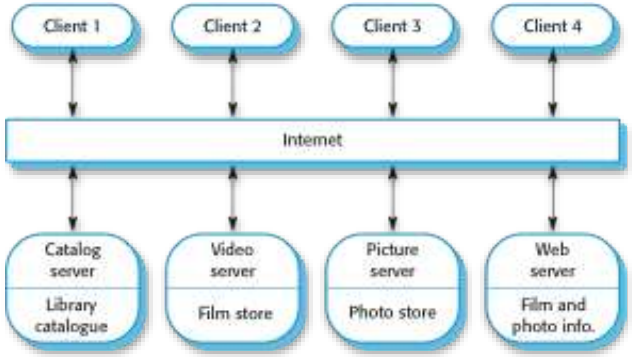
# Różne perspektywy

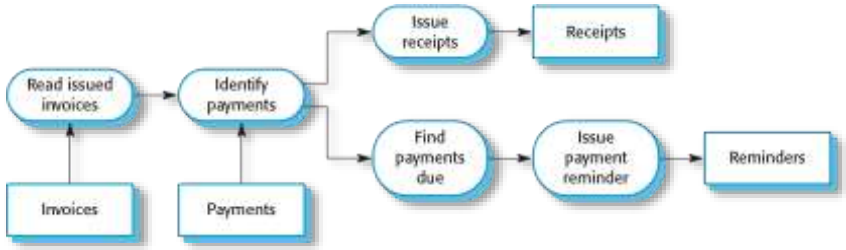





# Systemy rozproszone

- Zbiór niezależnych jednostek obliczeniowych, które z punktu widzenia użytkownika postrzegane są jako pojedynczy, spójny system.
- Przetwarzanie informacji jest rozproszone pomiędzy zespół komputerów – nie ogranicza się do pojedynczej jednostki.
- W praktyce prawie wszystkie duże systemy komputerowe są systemami rozproszonymi
  - Dlatego inżynieria systemów rozproszonych jest bardzo ważna dla systemów przemysłowych.

# Charakterystyka systemów rozproszonych

- Współdzielenie zasobów
  - Sprzętu i oprogramowania.
- Otwartość
  - Wykorzystanie sprzętu i oprogramowania pochodzącego od różnych dostawców.
- Współbieżność
  - W celu zwiększenia wydajności.
- Skalowalność
  - Zwiększenie przepustowości przez dodanie nowych zasobów.
- Tolerancja awarii
  - Zdolność do kontynuowania działań po wystąpieniu błędu.

# Problemy systemów rozproszonych

- Systemy rozproszone są bardziej skomplikowane niż te, które działają na pojedynczym procesorze.
- Złożoność związana jest z potrzebą niezależnego zarządzania częściami systemu
- Nie istnieje jeden organ odpowiedzialny za system - odgórne sterowanie jest niemożliwe.

# Problemy projektowe

- Przezroczystość (ang. Transparency)
- Otwartość (ang. Openness)
- Skalowalność (ang. Scalability)
- Zabezpieczenie (ang. Security)
- Jakość usługi (ang. Quality of service )
- Zarządzanie awariami (ang. Failure management )
  - Tolerowanie awarii (ang. Fault tolerance)

# Skalowalność + odporność na awarie = dostępność

Miara dostępności (ang. availability) – jedna z podstawowych miar określenia stopnia odporności systemu. Pojęcie dostępności oznacza czas bezawaryjnego działania usługi w stosunku do całości czasu, w którym usługa ta powinna być klientom świadczona.

$$A = \frac{T_m}{T_m + T_d} \begin{cases} A - dostępność \\ T_m - średni\ czas\ do\ wystąpienia\ usterki \\ T_d - średni\ czas\ naprawy \end{cases}$$

Źródło: http://pl.wikipedia.org/wiki/Dostępność_(niezawodność)

# Klasy dostępności

| Typ systemu | Czas niedostępności (w roku) | Czas niedostępności (w miesiącu) | Dostępność | Klasa dostępności |
|---|---|---|---|---|
| Unmanaged | 36,5 dni | 72 godzin | 90% | 1 |
| Managed | 3,65 dni | 7,2 godzin | 99% | 2 |
| Well Managed | 8,76 godzin | 43,8 minut | 99,9% | 3 |
| Fault Tolerant | 52,56 minut | 4,32 minut | 99,99% | 4 |
| High-Availability | 5,26 minut | 25,9 sekund | 99,999% | 5 |
| Very-High-Availability | 31,5 sekund | 2,59 sekund | 99,9999% | 6 |
| Ultra-Availability | 3,15 sekund | 0,259 sekund | 99,99999% | 7 |

SLA *(ang. Service Level Agreement)*

oznacza umowę utrzymania ustalonego poziomu jakości usług między klientem a usługodawcą. Potocznie przyjmuje się, że SLA oznacza klasę dostępności usług.

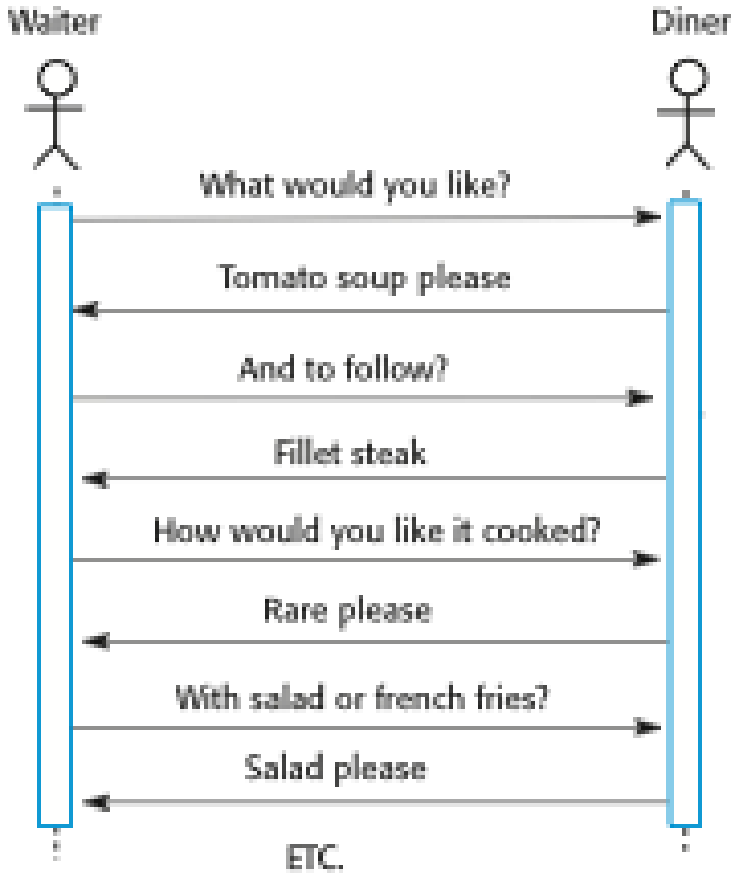
# Zasady budowania skalowanych systemów

- Bezstanowość
- Gruboziarnistość API
- Idempotentność
- Tolerowanie awarii
- Ogólna konfiguracja
- Proste, zautomatyzowane wdrażanie
- Monitoring
- Prostota - KISS (Keep It Simple Stupid)

# Modele interakcji

- Podstawowe typy interakcji pomiędzy komponentami w systemie rozproszonym:
  - **Komunikacja proceduralna**
    - Jeden komputer wywołuje znaną mu usługę i (zazwyczaj) oczekuje na odpowiedź.
  - **Interakcja oparta na komunikatach** – jednostka wysyła informacje o tym co jest potrzebne do innej jednostki. Oczekiwanie na odpowiedź nie jest wymagane.

# Komunikacja „proceduralna” pomiędzy klientem a kelnerem (serwisem)



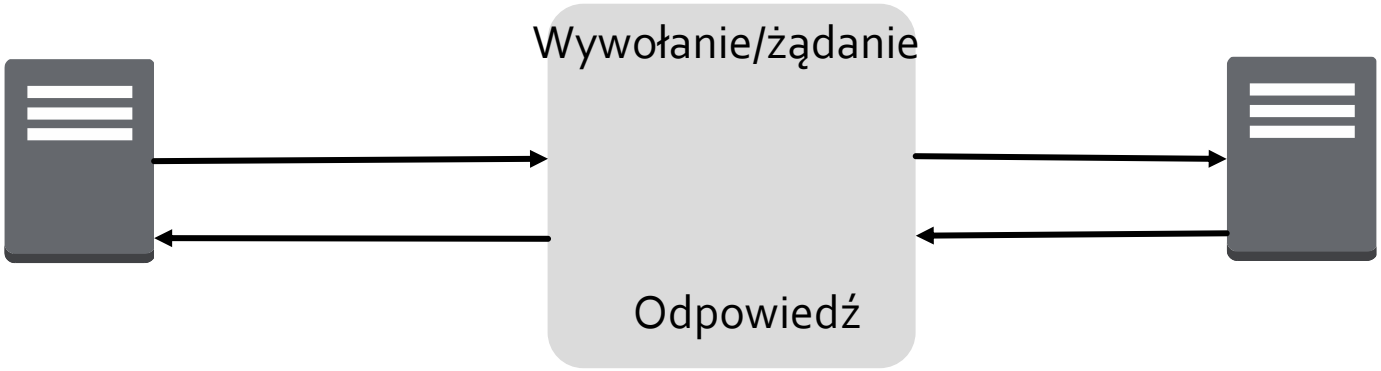
# Interakcja oparta na komunikatach pomiędzy kelnerem a kuchnią

```
<starter>
        <dish name = “soup” type = “tomato” />
        <dish name = “soup” type = “fish” />
        <dish name = “pigeon salad” />
</starter>
<main course>
        <dish name = “steak” type = “sirloin” cooking = “medium” />
        <dish name = “steak” type = “fillet” cooking = “rare” />
        <dish name = “sea bass”>
</main>
<accompaniment>
        <dish name = “french fries” portions = “2” />
        <dish name = “salad” portions = “1” />
</accompaniment>
```

# Zdalne wywołanie procedury (ang. Remote procedure call - RPC)

- Komunikacja proceduralna w systemie rozproszonym implementowana jest za pomocą zdalnych wywołań procedur.
- W RPC jeden z komponentów wywołuje inny komponent w taki sam sposób jakby wywoływał procedurę lokalną. Warstwa pośrednia przechwytuje wywołanie i przekazuje je do zdalnego komponentu.
- Zdalny komponent wykonuje obliczenia i, za pośrednictwem warstwy pośredniej, zwraca rezultat wywołującemu.
- Wymagania RPC:
  - wywołujący i wołany muszą być dostępni w czasie komunikacji.
  - wywołujący i wołany muszą wiedzieć jak się ze sobą komunikować.

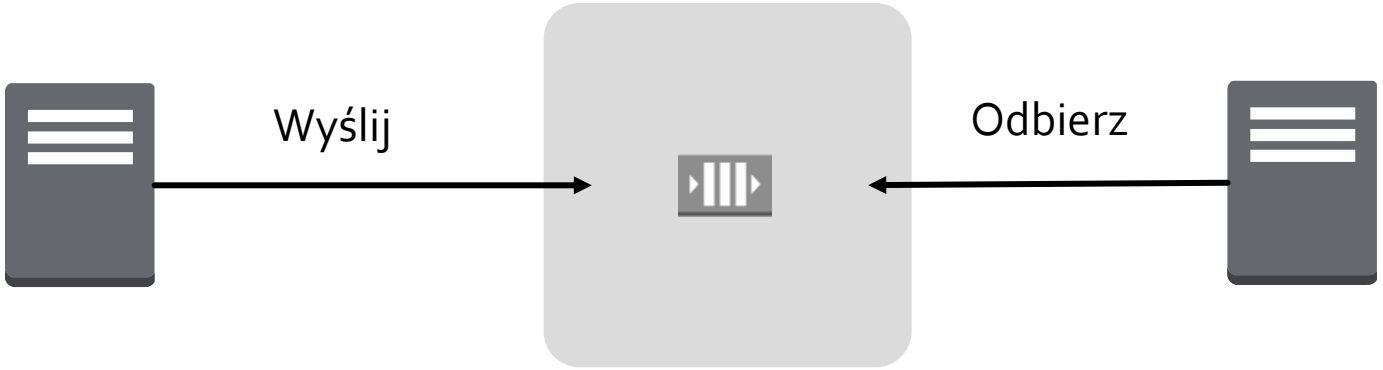
# RPC



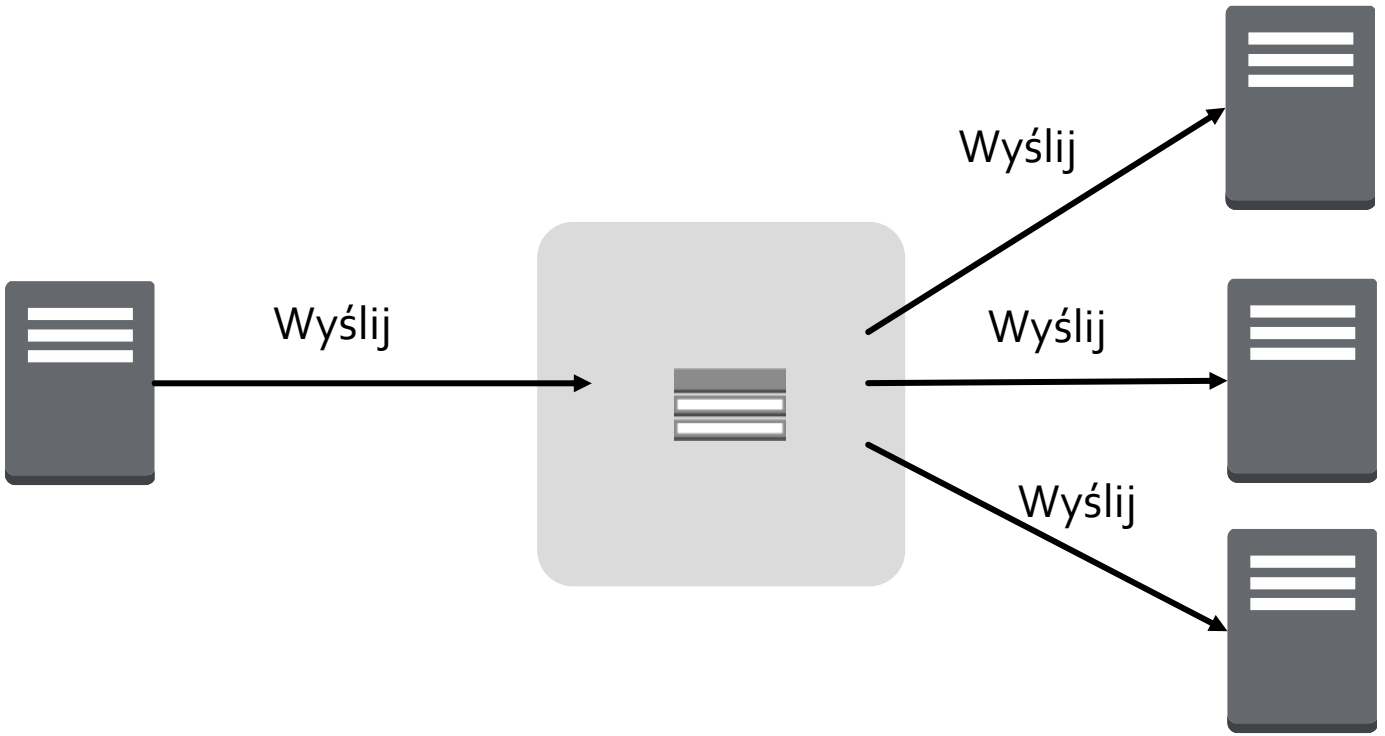
# Przekazywanie komunikatów

- Jeden komponent tworzy komunikat opisujący usługę do wykonania przez inny komponent.
- Komunikat wysyłany jest do odbiorcy za pośrednictwem tzw. „warstwy pośredniej”.
- Odbiorca parsuje komunikat, wykonuje żądane obliczenia. Wysłanie rezultatów obliczeń do odbiorcy wymaga utworzenia komunikatu.
- Nie jest wymagane aby wysyłający i odbiorca mieli świadomość swojego istnienia.

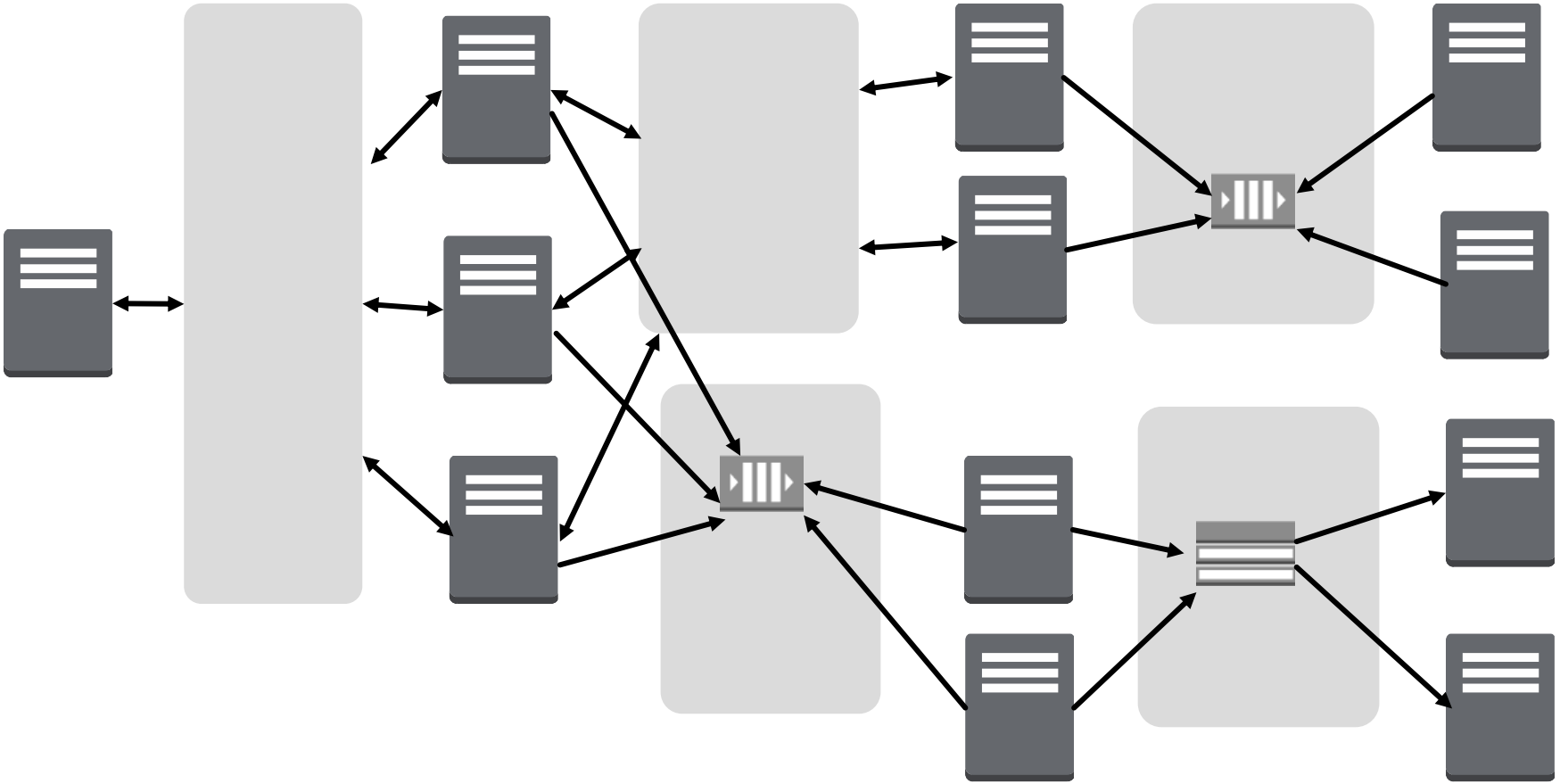
# Przekazywanie komunikatów



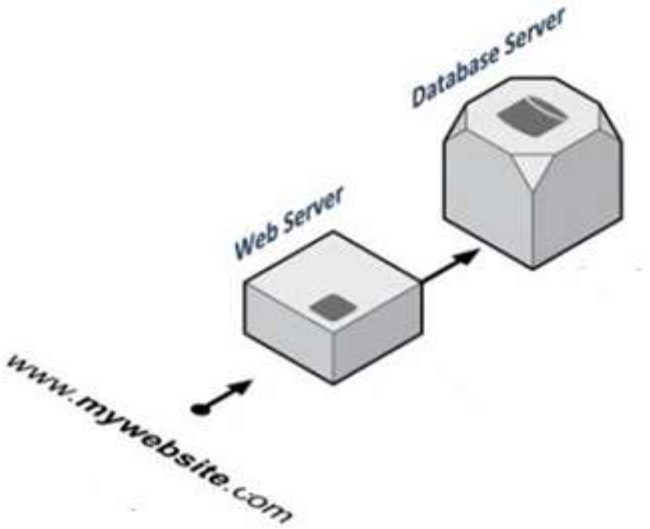
# Przekazywanie komunikatów z rozgłaszaniem - notyfikacja



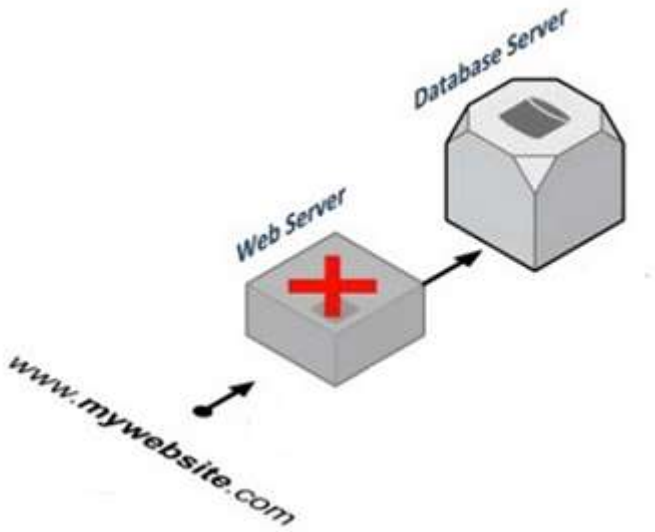
# Architektura rozproszona



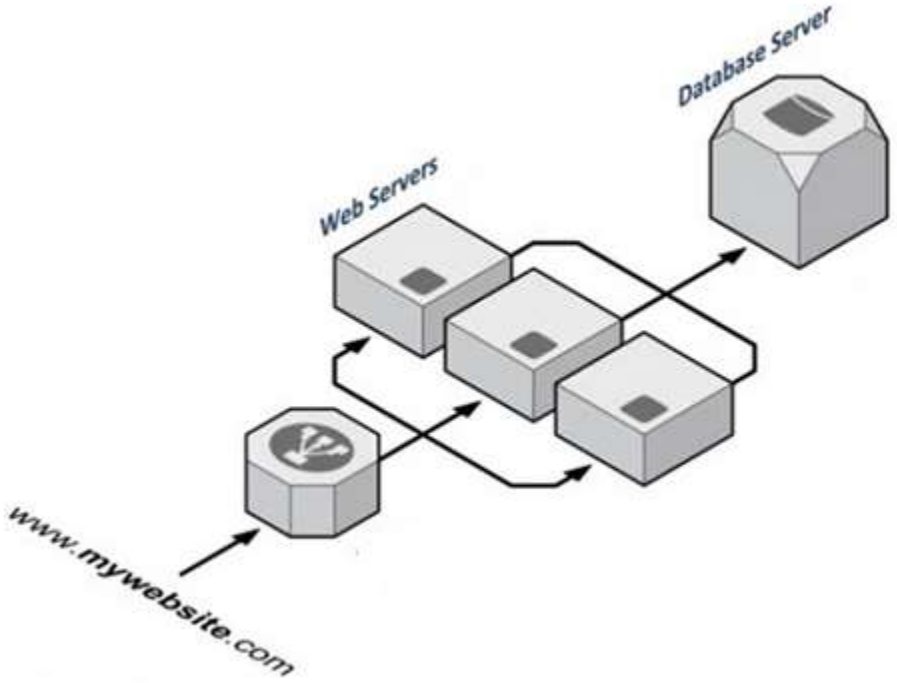
# Klasyczna konfiguracja aplikacji webowej



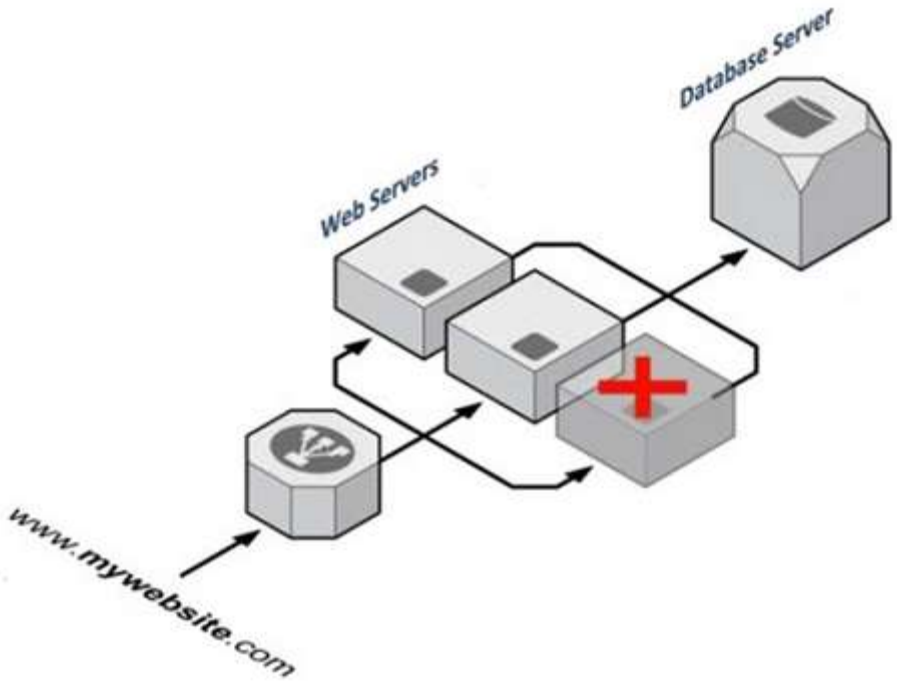
# Awaria serwera webowego



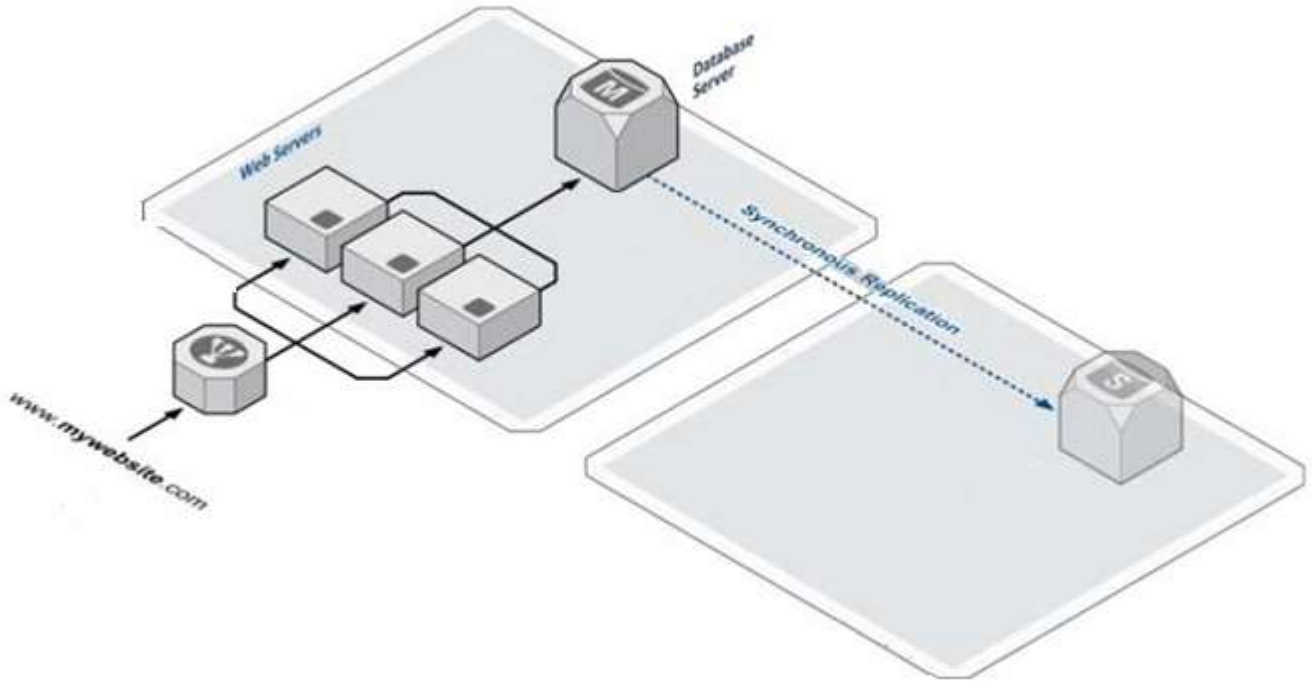
# Load Balancer



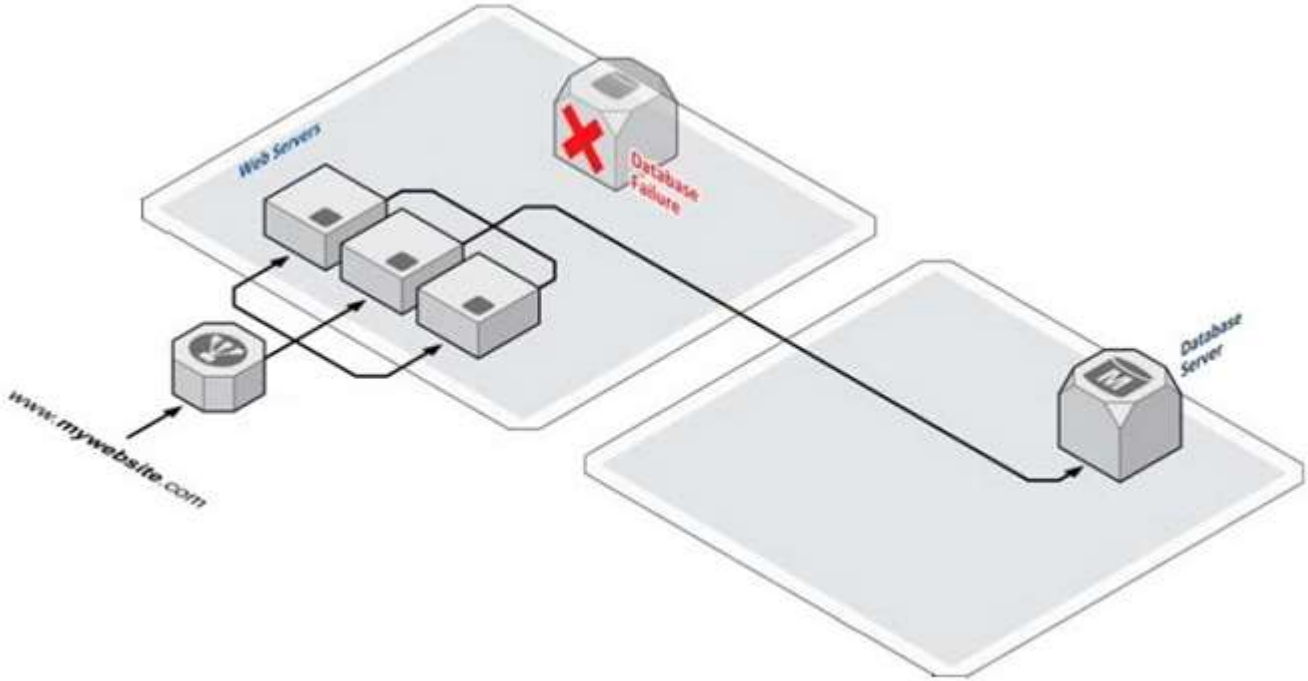
# Awaria serwera webowego



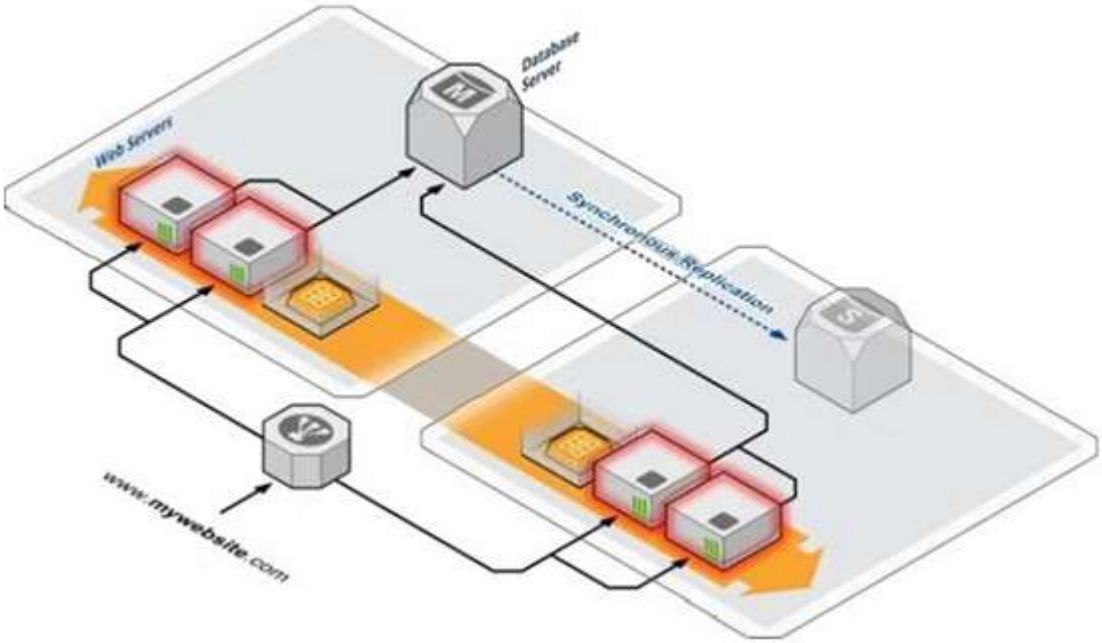
# Replikacja BD



# Awaria serwera BD



# Automatyczne skalowanie



# Oprogramowanie jako usługa (Software as a service - SaaS)

- Zdalny hosting oprogramowania i udostępnianie dostępu za pośrednictwem Internetu.
  - Oprogramowanie jest wdrażane na serwerze (zbiorze serwerów) i dostępne za pośrednictwem przeglądarki Internetowej.
  - Oprogramowanie jest najczęściej własnością udostępniającego a nie organizacji, która z niego korzysta.
  - Opłata może być pobierana za zakres wykorzystania, być powiązana z czasową subskrypcją lub innym modelem (np. wymaganie oglądania reklam).
- Jest to jeden z modeli tzw. chmur obliczeniowych (Cloud Computing)

# Chmura obliczeniowa

- Chmura obliczeniowa: model przetwarzania oparty na użytkowaniu usług dostarczonych przez usługodawcę.

- „Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. The cloud model of computing promotes availability.”

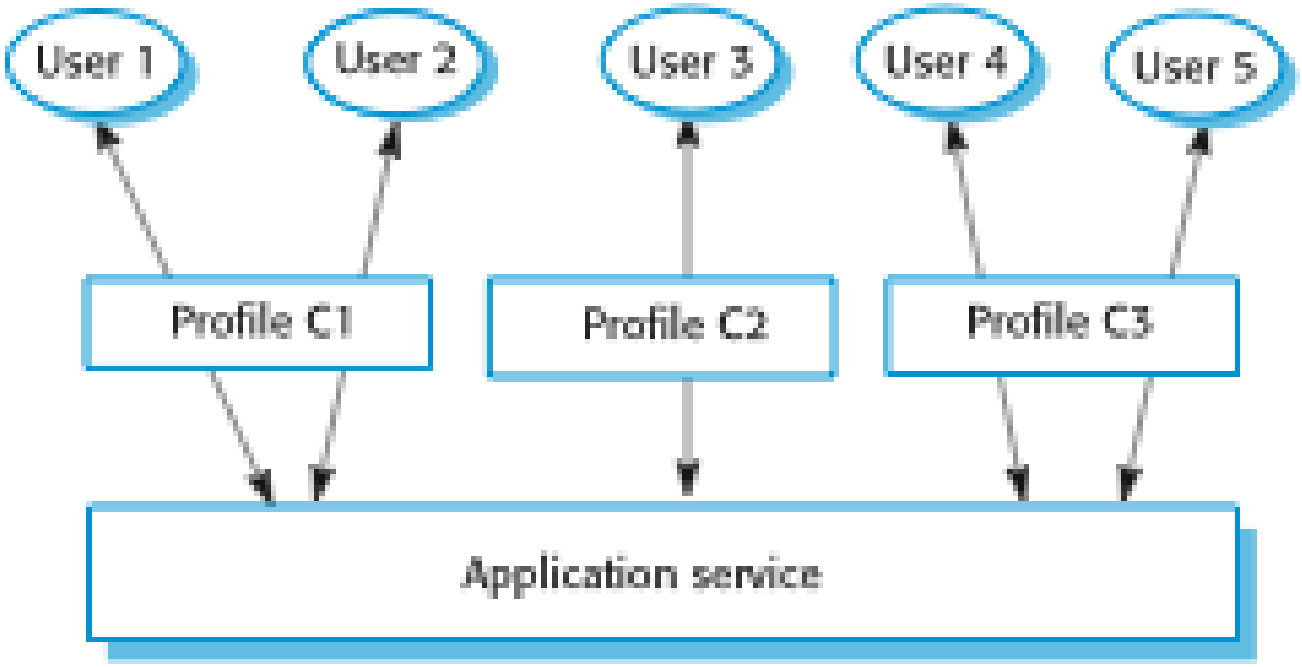
Źródło: US National Institute for Standards and Technology (NIST)

# Charakterystyka

- Główne cechy chmur obliczeniowych
  - Samoobsługa na żądanie *(ang. on-demand self-service)*
  - Szeroki dostęp sieciowy *(ang. broad network access)*
  - Pule zasobów *(ang. resource pooling and multi-tenancy)*
  - Elastyczność *(ang. elasticity)*
  - Mierzalność usług *(ang. measured service usage)*
  - Niezawodność *(ang. reliability, resiliency)*

# Czynniki implementacyjne dla SaaS (1)

- ***Konfigurowalność*** *- W jaki sposób konfigurować oprogramowanie dla specyficznych wymagań różnych organizacji?*



# Czynniki implementacyjne dla SaaS (2)

- Wynajmowanie wspólnej puli zasobów (ang. Multi-tenancy)
  - Udostępnianie oprogramowanie użytkownikowi, w taki sposób aby miał wrażenie, że pracuje na swojej własnej kopii systemu
  - Architektura systemu umożliwia efektywnie wykorzystanie zasobów.
  - Wymaga to istnienia bezwzględnej separacji funkcjonalności oraz danych systemu.

# Współ-wynajmowana baza danych

| Tenant | Key | Name | Address |
|---|---|---|---|
| 234 | C100 | XYZ Corp | 43, Anystreet, Sometown |
| 234 | C110 | BigCorp | 2, Main St, Motown |
| 435 | X234 | J. Bowie | 56, Mill St, Starville |
| 592 | PP37 | R. Burns | Alloway, Ayrshire |

# Czynniki implementacyjne dla SaaS (3)

- *Skalowalność – W jaki sposób zaprojektować system aby można go było skalować do nieprzewidywalnie dużej liczby użytkowników?*
  - Komponenty aplikacji implementowane w postaci prostych, bezstanowych usług, które mogą być wykonane na dowolnym serwerze.
  - Asynchroniczna interakcja – aplikacja nie musi czekać na rezultat interakcji (na przykład żądania odczytu).
  - Pula zasobów – zarządzanie zasobami (sieć, połączenia z bazą danych) w postaci puli – minimalizuje to prawdopodobieństwo braku zasobów dla realizacji usługi.
  - Drobnoziarnistość blokowania danych w bazie danych.

# Architektura zorientowana usługowo (Service Oriented Architecture SOA)

- Sposób rozwijania systemów rozproszonych, gdzie komponenty są niezależnymi usługami.
- Usługi mogą być wykonywane na różnych maszynach przez różnych dostawców.
- Standardowe protokoły pozwalają na komunikację z usługą oraz wymianę informacji.

# Usługi Internetowe w stylu REST

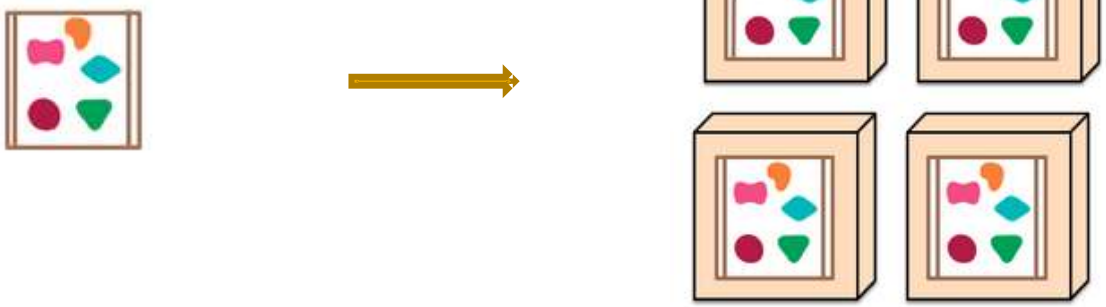
- REST (REpresentational State Transfer) – wzorzec architektoniczny bazujący na przesyłaniu przez serwer reprezentacji zasobu.
- Wzorzec opiera się na istniejącej infrastrukturze Internetowej (nie wprowadza nowych protokołów) co upraszcza implementację usług sieciowych.
- Usługi Internetowe typu REST (określane jako RESTful) wymagają mniejszych nakładów w stosunku do tzw. „dużych usług Internetowych”.

# Mikroserwisy

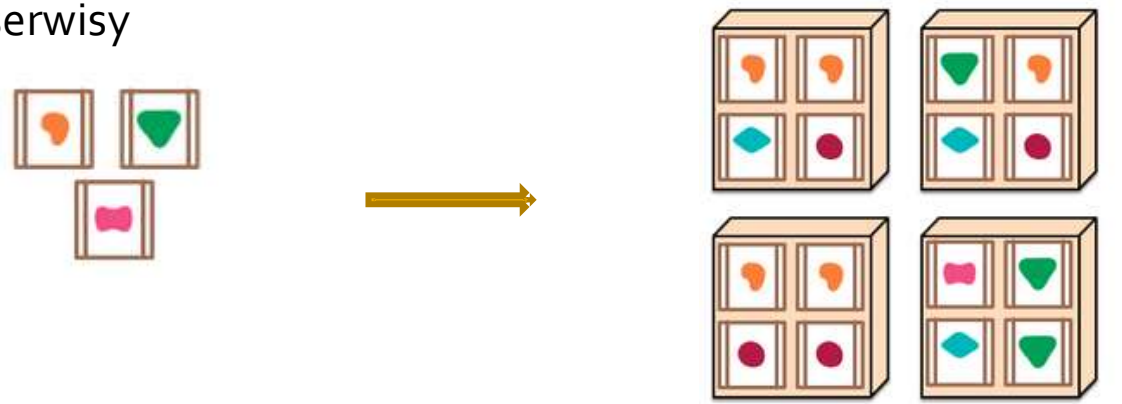
- Budowanie aplikacji z niezależnie wdrażanych usług.
  - Uruchomionych w odrębnych procesach.
  - Komunikujących się ze sobą za pomocą „lekkiego” protokołu.
  - Automatycznie wdrażanych.

# Skalowanie a mikroserwisy

Aplikacja monolityczna



Mikroserwisy

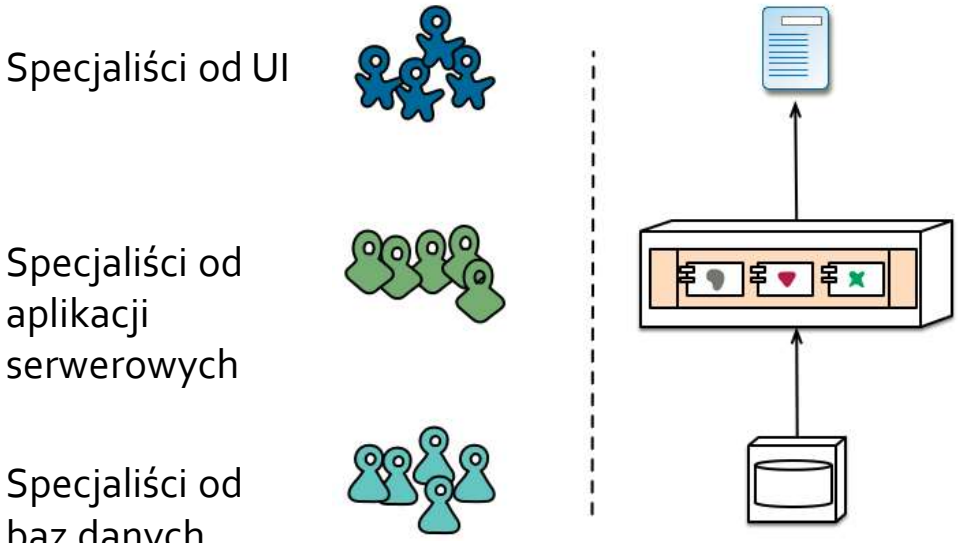


martinfowler.com/articles/microservices.html

# Charakterystyka architektury mikroserwisy

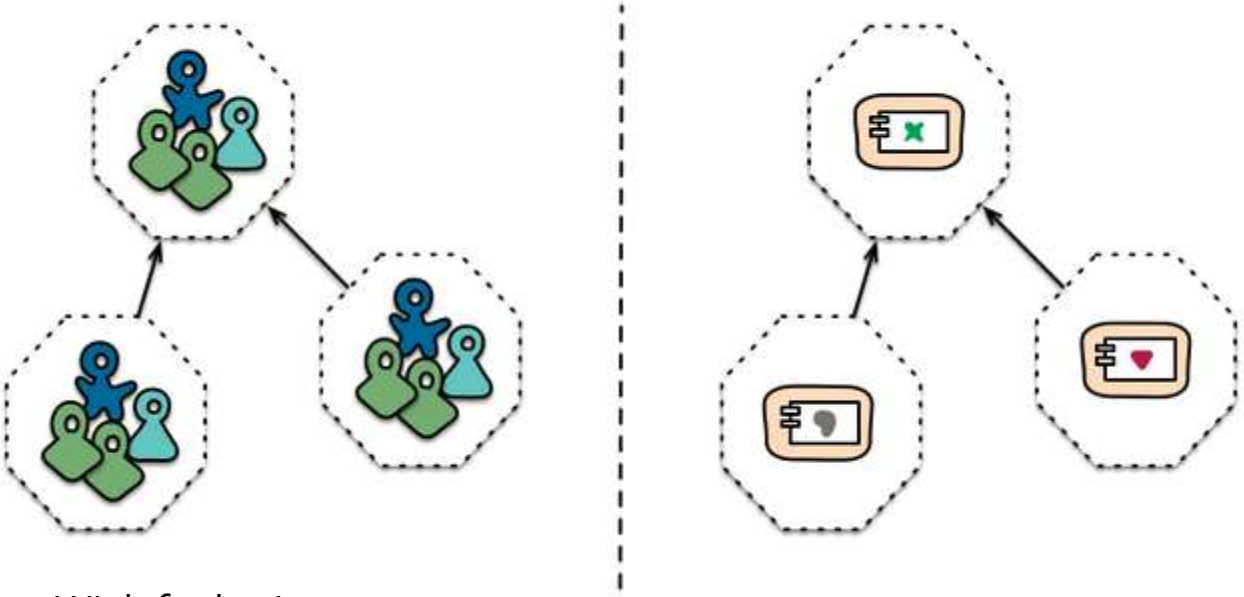
- Zorganizowana wokół charakteru biznesowego
  - Prawo Conway'a
    - Każda organizacja, która projektuje systemy (szeroko pojęte) dąży do takiej struktury, która jest kopią struktury komunikacyjnej organizacji.

http://www.melconway.com/Home/Committees_Paper.html




martinfowler.com/articles/microservices.html

# Mikroserwisy a podejście zwinne



Wielofunkcyjne zespoły samo-organizujące się wokół możliwości biznesowych

martinfowler.com/articles/microservices.html

# Charakterystyka architektury mikroserwisów

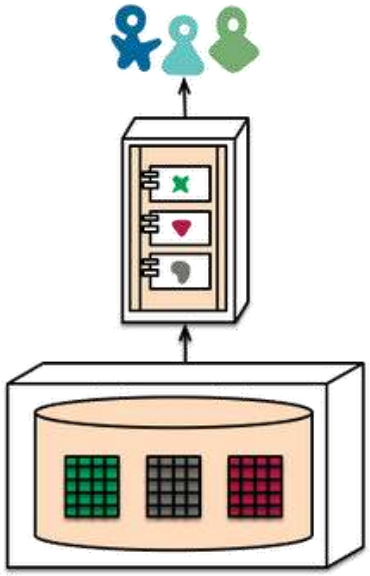
- Produkty a projekty
  - Podejście projektowe - Dostarczamy kompletne oprogramowanie i „rozwiązujemy zespół”
  - Podejście produktowe - Zespół powinien być „właścicielem produktu” przez cały czas jego życia

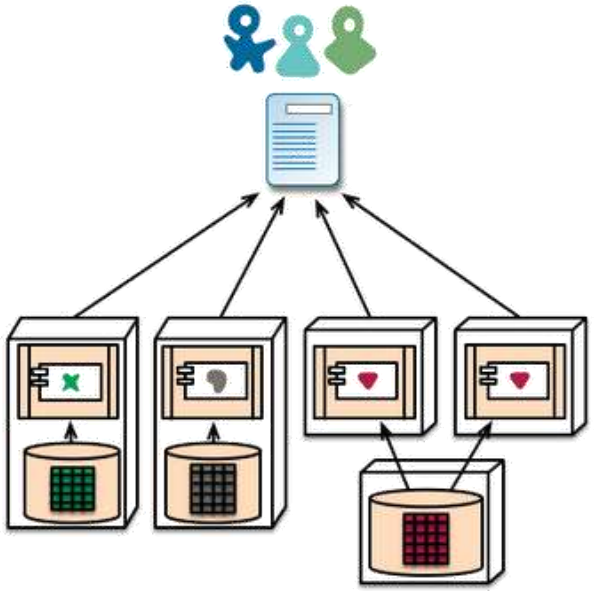
# Charakterystyka architektury serwisów

- Zdecentralizowane zarządzanie
  - Wiele platform technologicznych
    - Narzędzia odpowiednie dla problemu
    - Opracowywanie narzędzi dla programistów (opensource)
  - Build it / run it
    - Jesteś odpowiedzialny za działanie swojego systemu

# Charakterystyka architektury mikroserwisów

- Zdecentralizowane administrowanie danymi
  - Polyglot persistence
  - Koordynacja beztransakcyjna
    - Tolerowanie pewnego poziomu niespójności





martinfowler.com/articles/microservices.html

# Charakterystyka architektury mikroserwisów

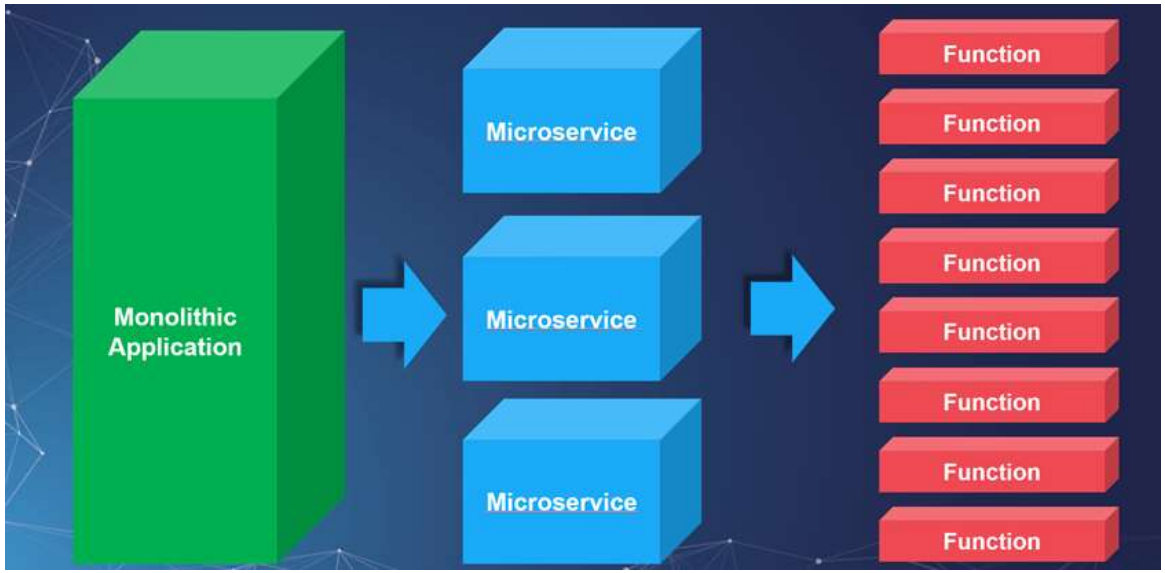
- Projektowanie z myślą o awariach
  - Każda usługa może ulec awarii, system musi być na to przygotowany
    - Szybkie wykrywanie awarii (monitoring, podgląd logów w czasie rzeczywistym)

# Charakterystyka architektury mikroserwisów

- Projektowanie ewolucyjne
  - Zmiana poprzez dodawanie nowych usług i usuwanie istniejących
    - Zastępowalność
    - Restrukturalizacja

# Architektura bezserwerowa

- Serverless architecture
  - BaaS, FaaS, Lambda



# Podsumowanie (1 z 2)

- Zaletą systemów rozproszonych jest możliwość ich skalowania w miarę wzrastających potrzeb. Może również kontynuować udostępnianie usług pomimo awarii oraz pozwala na współdzielenie zasobów.
- Do istotnych zagadnień, które należy wziąć po uwagę w trakcie projektowanie systemów rozproszonych należą: przezroczystość, otwartość, skalowalność, zabezpieczenie, jakość obsługi oraz zarządzanie awariami.
- Systemy typu klient-serwer mają strukturę warstwową. Warstwy mogą być rozproszone na różne komputery

# Podsumowanie (2 z 2)

- Do  wzorców architektury systemów rozproszonych należą: architektura master-slave, dwuwarstwowa i wielowarstwowa architektura typu klient-serwer, architektura rozproszonych komponentów oraz architektura peer-to-peer.
- Architektura rozproszonych komponentów wymaga istnienia warstwy pośredniej. Jej podstawowym zdaniem jest pośredniczenie w komunikacji pomiędzy komponentami oraz umożliwienie dodawania i usuwania komponentów do/z systemu.
- Architektura peer-to-peer reprezentuje zdecentralizowaną strukturę systemu bez podziału na usługobiorcę (klienta) i usługodawcę (serwer). Obliczenia mogą być rozproszone pomiędzy wiele systemów i organizacji.
- Oprogramowanie jako usługa (Software as a service) to metoda wdrażania systemów z cienkim klientem, w którym klientem jest przeglądarka Internetowa.