Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

- **Informacje organizacyjne**
- **Wprowadzenie**

Rozproszone systemy komputerowe

- **Konsultacje**
  - Institute of Informatics web site:
    https://wiz.pwr.edu.pl/en/students/teachers-consultations
    www.ii.pwr.edu.pl/~fras/index.php?str=konsultacje
  - Location
    - Build. C-3, room 301
- **Info**
  - http://eportal.pwr.edu.pl
    - Uwierzytelnianie: jak dla JSOS
    - Kurs: Rozproszone systemy informatyczne
      - Hasło zapisu: indywidualnie dla grup laboratoryjnych

Rozproszone systemy informatyczne

- **Zaliczenie**
  - Zaliczenie wykładu na podstawie testu (wiedzy)
    - Test zaliczeniowy elektroniczny
      - **22.06.2021** – ostatni wykład (opcjonalnie później, jeśli będzie wspólnie ustalone)
        - dostęp zdalny
        - z użyciem kamer – widok z boku
        - W przypadku nie użycia kamer może być uzgodnione zaliczenie w trybie hybrydowym
      - Termin poprawkowy – aktualnie **nie** jest przewidziany
        - uczestnictwo warunkowe (obecność na wykładach)
        - zaliczenie: **od 60%** maksymalnej liczby punktów
        - oceny – odpowiednio progi podwyższone
  - Ocena
    - Wykład: **od 50%** maksymalnej liczby punktów z testu
      - Kolejne, wyższe oceny – progi co 10%
      - **Nie** ma zaliczeń wykładu na podstawie oceny z laboratorium!
    - **Ocena końcowa z kursu** – średnia ocen z wykładu i laboratorium
      - Jeśli średnia dokładnie "po środku" wtedy decyduje ocena z wykła
        - Ocena "bliższa wynikowi testu" jest końcowa

Rozproszone systemy informatyczne

- **Cel i zakres przedmiotu**
  - Podstawowe architektury i zagadnienia aplikacji w rozproszonym środowisku przetwarzania
  - Podstawowe modele przetwarzania i mechanizmy stosowane w systemach rozproszonych
  - Teoretyczne i praktyczne aspekty konstrukcji i działania systemów rozproszonych
  - Wybrane techniki i technologie realizacji aplikacji rozproszonych
  - Wybrane algorytmy do realizacji przetwarzania w rozproszonych systemach komputerowych
- **Laboratorium rozproszonych systemów komputerowych**
  - Praktyczne umiejętności implementacji aplikacji rozproszonych w wybranych środowiskach
    - RPC
    - DO
    - WS
    - REST
    - …

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Wprowadzenie

- Ogólna charakterystyka systemów rozproszonych (SR)
- Przegląd architektur (architektury oprogramowania)

Definicje

- **Przetwarzanie zcentralizowane**
  - Pojedyncza jednostka (komputer/host - np. mainframe) realizuje wszystkie elementy przetwarzania włączając obsługę wejścia, wyjścia, przechowywania i uzyskiwania danych.
- **Przetwarzania rozproszone**
  - Wiele odrębnych jednostek realizuje różne elementy przetwarzania
- **System rozproszony**
  Zbiór samodzielnych jednostek, połączonych środkami komunikacyjnymi, wyposażonych w oprogramowanie tworzące zintegrowane środowisko przetwarzania.
  - Zbiór niezależnych komputerów, które z punktu widzenia użytkowników systemu sprawiają wrażenie pojedynczego logicznie zwartego systemu.
  - **Przetwarzanie kooperatywne**
    - Różnice między komputerami, ich odrębność i sposoby komunikacji są ukryte przed użytkownikiem
    - Przetwarzanie, dostęp i użycie współdzielonych zasobów jest przezroczyste dla użytkownika

⇓

  - To rodzi problemy i niełatwo zapewnić pożądane cechy systemów rozproszonych

System rozproszony

- **Podstawowe aspekty definicji**
  - Sprzętowy
    - Autonomiczność komputerów
      - nie mają wspólnej pamięci ani zegara
      - własne systemy operacyjne
    - Połączenie siecią komunikacyjną
      - możliwość przesyłania wiadomości
  - Logiczny
    - Wspólne przetwarzanie
    - Współbieżne wykonywanie przetwarzania
    - Współdzielenie zasobów
    - Koordynacja działań
  - Programowy
    - Rozproszone oprogramowanie systemowe
      - logiczne wiązanie zasobów
      - ukrywanie faktu rozproszenia
      - realizacja różnych funkcji na różnych poziomach

Ogólna charakterystyka systemów rozproszonych

- **Zalety**
  - Dzielenie zasobów
  - Zwiększenie niezawodności
  - Przyspieszenie przetwarzania
  - Komunikacja dostarczana na poziomie systemu
  - Zapewnienie wzrostu i skalowalności
  - Ekonomiczność
- **Wady**
  - Złożoność oprogramowania
  - Trudność zapewnienia współdziałania (standardy!)
  - Zawodność komunikacji
  - Bezpieczeństwo
- **Cele projektowe dla systemu rozproszonego**
  - Otwartość
  - Skalowalność
  - Współbieżność
  - Tolerowanie awarii
  - Przezroczystość

Ogólna charakterystyka systemów rozproszonych

- **Otwartość**
  - Zdolność do rozbudowy (sprzętowej, programowej) i rekonfiguracji – możliwość dodawania nowych usług bez głębokiej ingerencji w już istniejące
    - Oferowanie usług wg ściśle określonych reguł → standardy !
    - Określone za pomocą interfejsów
      - Standaryzowanie interfejsów
      - Języki opisu interfejsu (IDL)
      - **Kompletność**
      - **Neutralność**
        - Mogą być różnie implementowane
    - W sieciach komputerowych reguły określane protokołami
  - Zdolność do współdziałania
  - Przenośność
  - Elastyczność
    - Konfigurowalność z różnych składowych
    - Rozszerzalność
    - Elastyczność osiągana jest m.in. poprzez
      - Niezależność od sprzętu
      - **Nie**-monolityczność architektury
      - Definiowanie interfejsów wewnętrznych

Ogólna charakterystyka systemów rozproszonych

- **Skalowalność** – możliwość dostosowania systemu do rosnących rozmiarów problemów i wymagań użytkowników
  - Odnosi się do 3 obszarów
    - Wymiar rozmiaru
      - Problem obsługi bardzo wielu żądań/zadań/klientów
      - Problem przetwarzania zadań o bardzo dużych rozmiarach
    - Wymiar geograficzny
      - Problem synchronicznej komunikacji (opóźnienia)
      - Problem niezawodności komunikacji
      - Problem lokalizacji zasobu
    - Wymiar administracyjny
  - Podstawowe/ogólne metody zapewnienia skalowalności
    - Rozpraszanie komponentów (usług, danych)
      - Decentralizacja wykonania (usługa, serwer)
      - Decentralizacja danych
      - Decentralizacja algorytmów (możliwość działania bez znajomości stanu **całego** systemu)
    - Komunikacja asynchroniczna
    - Replikacja (zwielokrotnianie)
      - to generuje różne problemy (np. zapewnienie spójności)

Ogólna charakterystyka systemów rozproszonych

- **Współbieżność**
  - Możliwość współbieżnego (równoczesnego) ubiegania się o zasoby i ich użytkowanie
    - Wielu klientów działa równocześnie uruchamiając różne usługi
    - Wiele serwerów działa współbieżnie obsługując różnych klientów
  - Szereg problemów sterowania współbieżnością
    - Np. transakcje rozproszone, wzajemnie wykluczający się dostęp do tych samych zasobów, itp.
- **Tolerowanie awarii** – odporność na uszkodzenia sprzętu lub oprogramowania
  - Zapewnienie wysokiego stopnia dostępności
  - Redundancja (zwielokrotnianie) elementów systemu
    - sprzętu, procesów, danych
  - Możliwość odtworzenia spójnego stanu po awarii – odtwarzanie programowe

Formy przezroczystości

- **Przezroczystość** – ukrywanie przed użytkownikiem fizycznego odseparowania składowych systemu
  - Ukrycie faktu rozproszenia procesów i zasobów na różnych komputerach
- **Przezroczystość dostępu**
  - Zagwarantowanie jednolitego sposobu dostępu do zasobów zarówno lokalnych jak i zdalnych,
  - Ukrywanie różnic w reprezentacji danych i metodach udostępniania usług
- **Przezroczystość położenia**
  - Identyfikacja zasobów niezależna od ich fizycznej lokalizacji
  - Wymagania dla systemu nazewnictwa; nazwy logiczne

Formy przezroczystości

- **Przezroczystość migracji**
  - Zmiana fizycznej lokalizacji zasobu nie powoduje zmian w sposobie ich identyfikacji i dostępu
- **Przezroczystość relokacji (przemieszczania)**
  - Zmiana lokalizacji zasobu dokonywana w sposób niewidoczny dla aplikacji w czasie jego użytkowania
- **Przezroczystość awarii**
  - Zdolność ukrycia chwilowych nieprawidłowości funkcjonowania poszczególnych komponentów systemu
  - Maskowanie awarii
    - Jest to trudny problem w SR
    - Np. problem rozróżniania nieoperatywności od braków wydajnościowych

Formy przezroczystości

- **Przezroczystość replikacji (zwielokrotniania)**
  - Utrzymywanie i udostępnianie kilku egzemplarzy (kopii) zasobu w sposób ukrywający fakt wielokrotności wystąpień
  - Wymaga przezroczystości położenia
- **Przezroczystość współbieżności**
  - Dla współdziałania rywalizującego
  - Realizacja współbieżnego dostępu do zasobu w taki sposób, że konkurujące procesy nie przeszkadzają sobie wzajemnie
  - Problem: zachowanie spójności w systemie
- **Przezroczystość a efektywność $\rightarrow$ kompromis**

Architektury systemów

- **Model konceptualny**
  - Architektura określa strukturę systemu – sposób powiązania jego elementów
    - Elementami systemu są zasoby (sprzętowe, programowe i danych)
    - Sposobami łączenia są mechanizmy (sprzętowe i programowe) wymiany informacji pomiędzy elementami
  - Modelowe architektury dotyczą układu komponentów systemu, przy czym komponent jest często rozumiany jako proces
- **Podstawowe (typowe) architektury systemów rozproszonych**
  - Klient-serwer
  - Równy-z-równym (peer-to-peer, P2P)
  - Złożone (na bazie powyższych)
    - Wyróżniane są elementy realizujące istotne funkcje z punktu widzenia koncepcji danej architektury

Architektury systemów

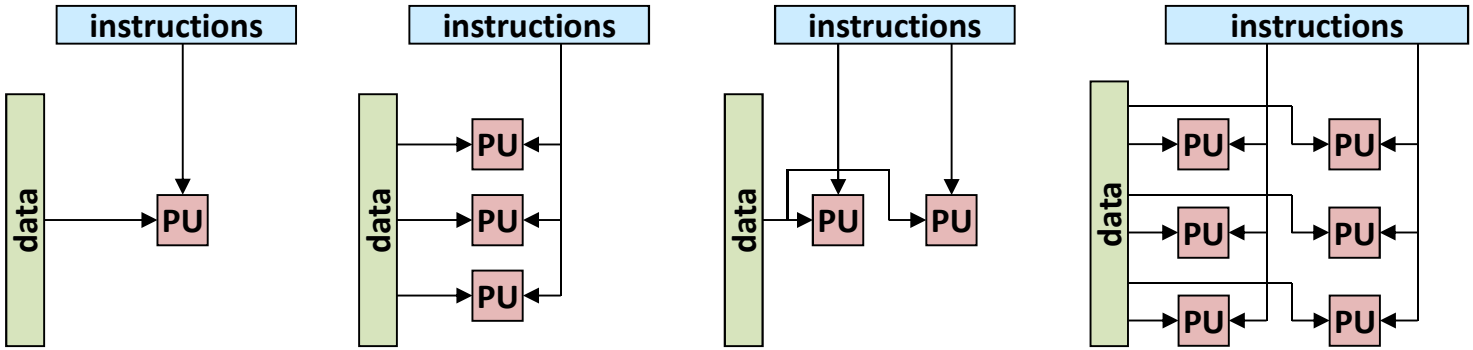
- **Zagadnienia konstrukcji systemów rozproszonych**
  - Architektura komunikacji
  - Synchronizacja działań
  - Alokacja zasobów rozproszonych
  - Zapewnienie wydajności algorytmów przetwarzania
  - Poprawność działania algorytmów rozproszonych
  - Ocena globalnego stanu przetwarzania
  - Realizacja zaawansowanych modeli przetwarzania
    - Transakcje rozproszone
    - Replikacja i spójność
    - …
  - Niezawodność
  - Bezpieczeństwo

Architektury systemów

- **Klasyfikacja architektur**
  - Mechanizm sterowania
    - SISD
    - MISD
    - SIMD
    - MIMD
  - Organizacja przestrzeni adresowej (pamięci)
    - Ścisłość powiązania
    - **Multiprocesory** – systemy ze wspólną pamięcią
    - **Multikomputery** – systemy z pamięcią rozproszoną
  - Struktura sieci połączeń
    - Sposób połączenia procesorów ze sobą definiuje topologię sieci połączeń
    - Różne topologie nadają się do różnych rodzajów przetwarzania (obliczeń)
  - Organizacja i funkcjonalność komponentów oprogramowania systemowego

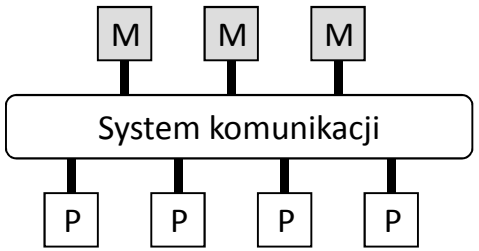
Klasyfikacja systemów komputerowych

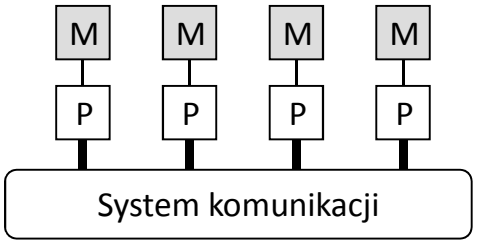
- **Klasyfikacja Flynna**
  - **SISD** (*Single Instruction Single Data*) - jeden strumień instrukcji, jeden strumień danych – pojedyncze komputery
  - **SIMD** (*Single Instruction Multiple Data*) - jeden strumień instrukcji, wiele strumieni danych – komputery wektorowe
  - **MISD** (*Multiple Instruction Single Data*) - wiele strumieni instrukcji, jeden strumień danych – systemy redundantne
  - **MIMD** (*Multiple Instruction Multiple Data*) - wiele strumieni instrukcji, wiele strumieni danych – komputery wieloprocesorowe, klastry komputerowe



Klasyfikacja systemów komputerowych

- **Organizacja przestrzeni adresowej (pamięci)**
  - Systemy ze wspólną pamięcią (Shared memory system)
    - **Multiprocesory** – wieloprocesory, systemy wieloprocesorowe – **ściśle powiązane** (ang. **tightly coupled**)
  - Systemy z pamięcią rozproszoną (Distributed memory system)
    - **Multikomputery** - systemy wieloprocesorowe – **luźno powiązane** (ang. **loosely coupled**)
      - Homogeniczne
      - Heterogeniczne





Klasyfikacja systemów komputerowych

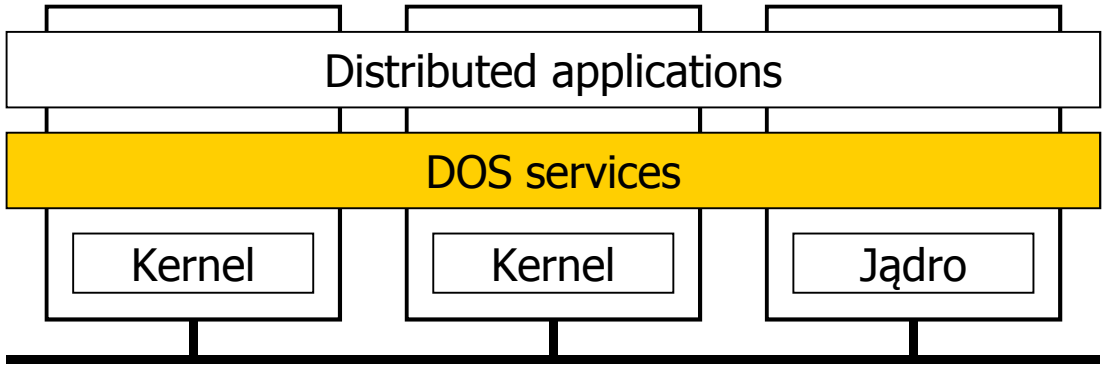
- **Skala powiązania** (nieco inna interpretacja)
  - Sieci rozległe, systemy internetowe
  - Sieci lokalne, Intranety
  - Systemy silnie/mocno powiązane
    - Klastry serwerów
    - Systemy końcowe, systemy wieloprocesorowe
- **Teoretyczne modele architektur równoległych**
  - PRAM (Parallel RAM)
    - Wiele procesorów, wspólna pamięć
    - Automatyczna synchronizacja działania
    - Pominięcie zagadnień synchronizacji i komunikacji (zapisu/odczytu z/do pamięci)
    - Modele przetwarzania jak u Flynna
  - Model sieciowy
    - Węzły połączone łączami – brak wspólnej pamięci
    - Wymiana danych wymaga przesyłania komunikatów

Oprogramowanie systemów rozproszonych

- **Systemy operacyjne systemów rozproszonych**
  - **DOS** (ang. *Distributed Operating System*) - **silnie powiązany** system operacyjny najczęściej dla multiprocesorów i jednorodnych multikomputerów
    - Globalne zarządzanie zasobami
    - Ukrywanie zasobów sprzętowych
  - **NOS** (ang. *Network Operating System*) - **słabo powiązany** system operacyjny dla heterogenicznych multikomputerów (dla sieci LAN i WAN)
    - Udostępnianie lokalne usług dla zdalnych klientów
  - **Oprogramowanie warstwy pośredniczącej** (ang. *Middleware*) – dodatkowa warstwa powyżej NOS implementująca usługi rozproszone ogólnego przeznaczenia
    - Zapewnienie rozproszonej przezroczystości

DOS

- **Silnie powiązany sprzęt i oprogramowanie**
  - Komputery w systemie nie są autonomiczne
    - Każdy ma własne jądro – moduły zarządzania lokalnymi zasobami
  - Multiprocesory lub multitikomputery homogeniczne
  - Jeden system operacyjny
- **Warstwa dostarczająca usługi rozproszone**
  - Zapewnianie przezroczystości
  - Użytkownicy nieświadomi rozproszenia – dostęp do zasobów zdalnych tak samo jak do zasobów lokalnych



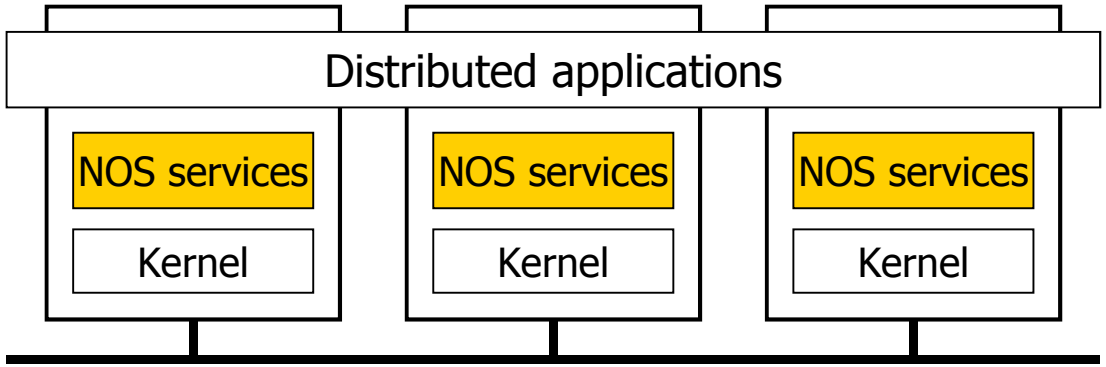
DOS

- **Cechy**
  - Duża tolerancja na błędy
  - Węzły przetwarzają z lokalną prędkością
    - Asynchronicznie (zegary niezależne) …lub…
    - Synchronicznie (zegary synchronizowane lub wspólny zegar globalny)
  - Przemieszczanie danych, obliczeń i procesów z jednego komputera na drugi odbywa się pod nadzorem SO.
    - migracja procesu może być jawna lub niejawna
    - równoważenie obciążenia, przyśpieszenie obliczeń
  - Komunikacja
    - **Rozproszona pamięć dzielona** (ang. *Distributed Shared Memory*)
      - Jednolita pamięć dzielona na strony i rozmieszczana we wszystkich węzłach systemu
      - Problemy z efektywnością przy odwoływaniu się do stron pamięci przechowywanych w innych węzłach

    …lub…
    - Komunikacja oparta na transmisji pakietów informacji
      - Trudniejsza implementacja oprogramowania
- **Przykłady systemów**
  - Amoeba, Mach, Chorus, DCE

NOS

- **Lokalne systemy operacyjne**
  - Autonomiczne
  - Lokalne zarządzanie zasobami
  - Usługi sieciowe
    - Model realizacji usług: klient-serwer
    - Protokół udostępniania usług
    - Zazwyczaj są zdefiniowane usługi typowe (powszechnie uznane)
- **Świadomość rozproszenia**
  - Konieczność jawnej lokalizacji usługodawcy



Architektura klient-serwer

- **Definicje**
  - **Serwer** (*server*)
    - Byt udostępniający **usługę**
      - Funkcje są wykonywane przez jednostkę (komputer), na którym serwer jest uruchomiony
    - W zależności od kontekstu: serwer = komputer który udostępnia usługę lub proces, a nawet obiekt, który realizuje/implementuje usługę
  - **Klient** (*client*)
    - Proces/aplikacja korzystający z usług serwera
    - Czasami klient może być serwerem w stosunku do innego klienta
      - Typowo w systemach P2P
  - **Usługa** (*service*)
    - Zbiór działań i rozwiązań, które są umieszczone w miejscu i są wykonywane, aby zapewnić powtarzalny i spójny zestaw wyników, rezultatów i wydajności dla osób, organizacji i systemów, które reprezentują konsumentów i beneficjentów takich wyników.
    - Zbiór programowych funkcjonalności, które mogą być używane wielokrotnie do różnych celów

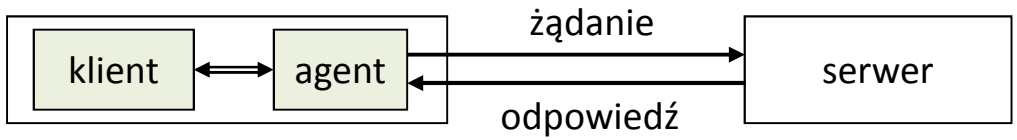
Więcej będzie omówione w kontekście RPC

Architektura klient-serwer

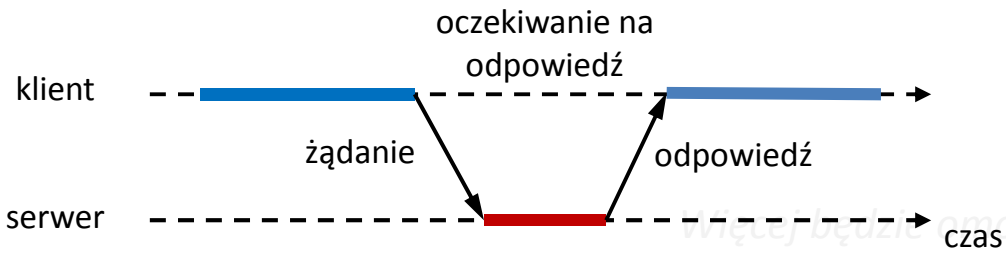
- **Definicje**
  - **Usługa sieciowa** (*network service*)
    - Komponent/aplikacja działająca w warstwie aplikacji sieci, mająca określoną funkcjonalność (np. przechowywanie danych, operowanie na danych, prezentacja, komunikacja lub inne), która jest implementowana wg określonej architektury (często klient-serwer lub peer-to-peer), z wykorzystaniem protokołów sieciowych warstwy aplikacji
  - **Protokół**
    - Zbiór zasad opisujących sposób współdziałania (łącznie z formatem danych) interaktywnych elementów
  - **Punkt dostępowy** (*endpoint*)
    - Element identyfikujący miejsce (adres) dostępu do usługi (adres wywołania usługi), a także często również sposób komunikacji, a nawet czasem rodzaj usługi

Architektura klient-serwer

- **Model realizacji usług klient-serwer**
  - Komponenty



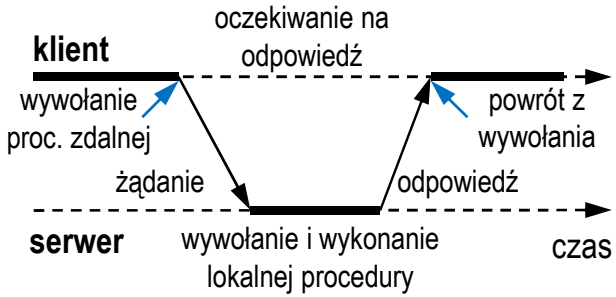
  - Agent
    - Funkcje pomocnicze dla obsługi żądań
    - Np. wsparcie przezroczystości (np. awarii, lokalizacji)
  - Wzorzec interakcji: zamówienie (żądanie) – odpowiedź
    - Określona zwłoka czasowa
    - Protokoły połączeniowe i bezpołączeniowe → niezawodność

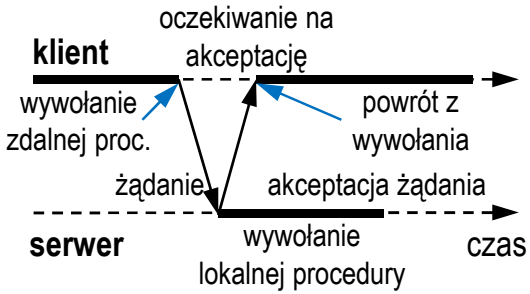


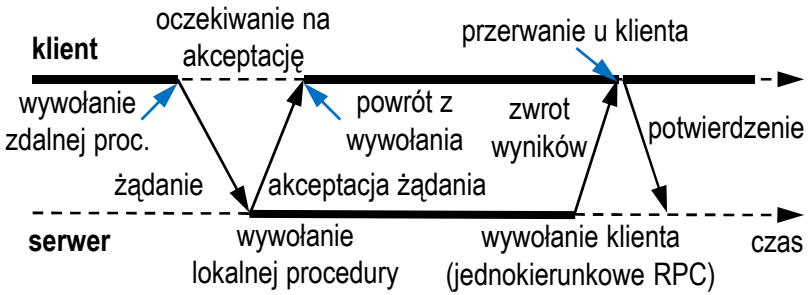
więcej będzie omówione w kontekście RPC

Architektura klient-serwer

- **Schematy współdziałania (komunikacji)**
  - Synchroniczne
    - Najczęstsza realizacja
    - Blokowanie klienta po *send*
  - Asynchroniczne
    - Klient nie musi lub nie może czekać
    - Istnieje niepewność realizacji
  - Odroczone asynchroniczne (z wywołaniem zwrotnym)
    - Zlecenia długich działań





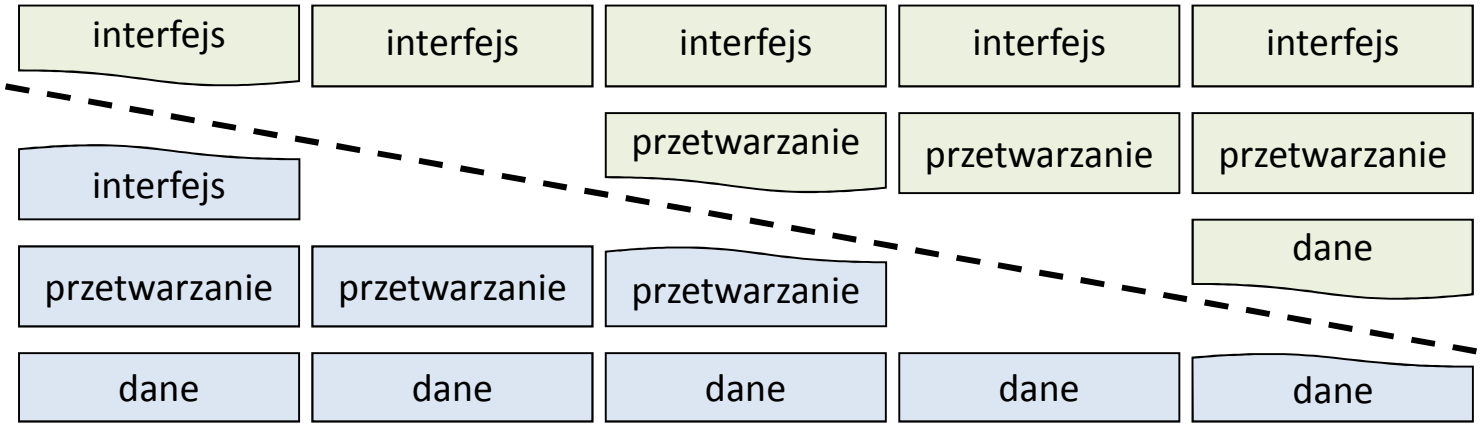


Architektura klient-serwer

- **Architektura warstwowa/poziomowa/piętrowa**
  - Architektura oprogramowania (aplikacji) – 3-warstwowa
    - **Warstwa interfejsu użytkownika** (UI – ang. *User Interface*)
    - **Warstwa logiki** / Przetwarzanie
    - **Warstwa danych**
      - Obejmuje różne zagadnienia związane z utrzymywaniem i dostępem do danych
      - M.in. zagadnienia trwałości i spójności danych
  - Architektura fizyczna (ze względu na odrębność lokalizacji elementów składowych)
    - **2-warstwowa**
      - Klient
      - Serwer
    - **Wielowarstwowa**
      - Komponenty oprogramowania umieszczane w wielu warstwach
      - Typowo: **architektura 3-warstwowa**

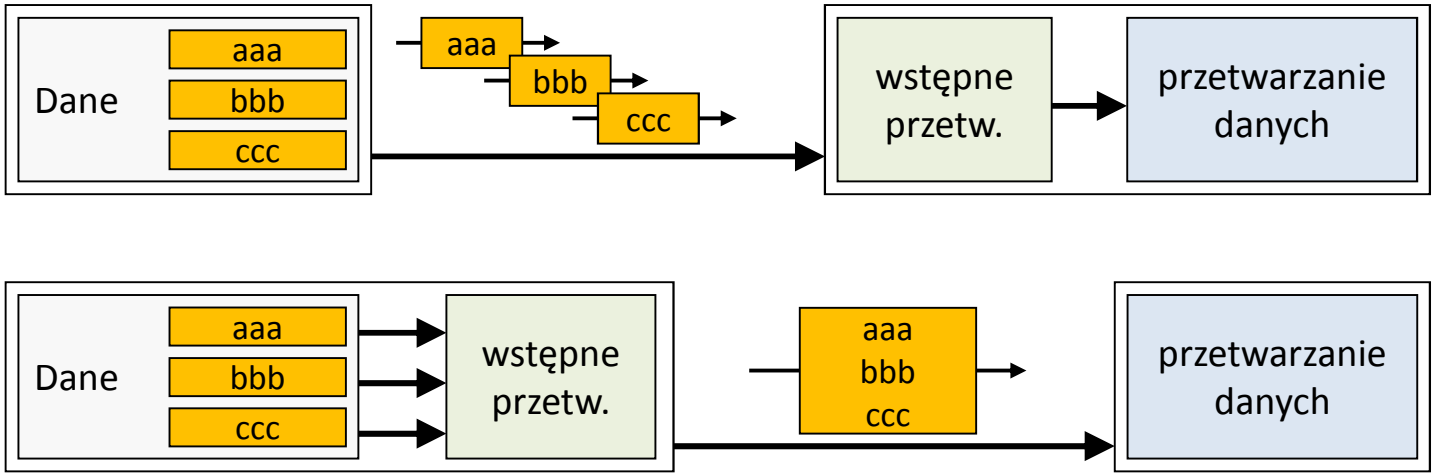
Architektura klient-serwer

- **Architektura 2-warstwowa**
  - Cienki klient (ang. *Thin client*)
  - Gruby klient (ang. *thick/rich client*)
  - Cechy
    - Może wymagać zmiany kodu klienta do portowania do innej bazy danych
    - Może generować duży transfer danych
    - Może wymagać mocy dużej obliczeniowej u klienta



Architektura klient-serwer

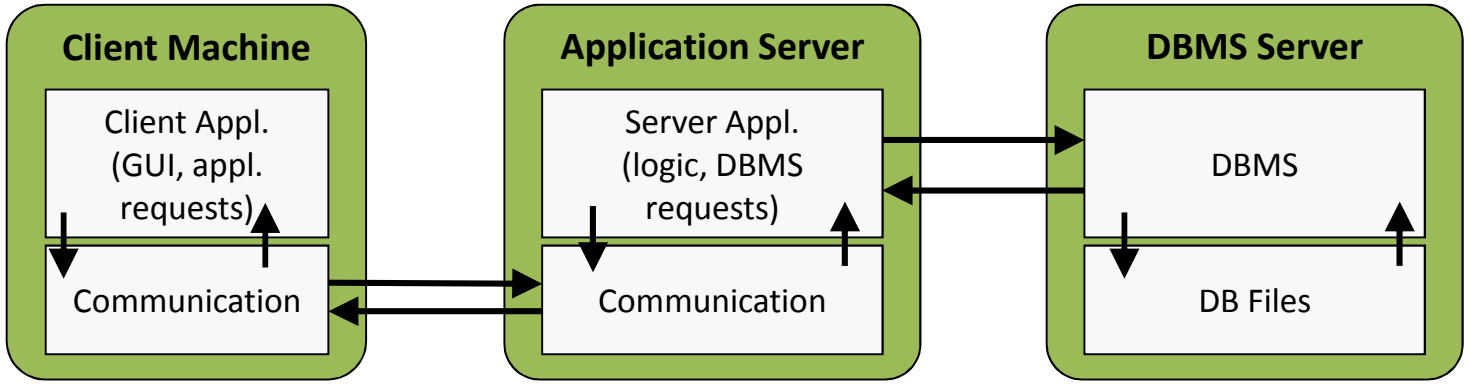
- **Otwartość i skalowalność**
  - Wybór modelu



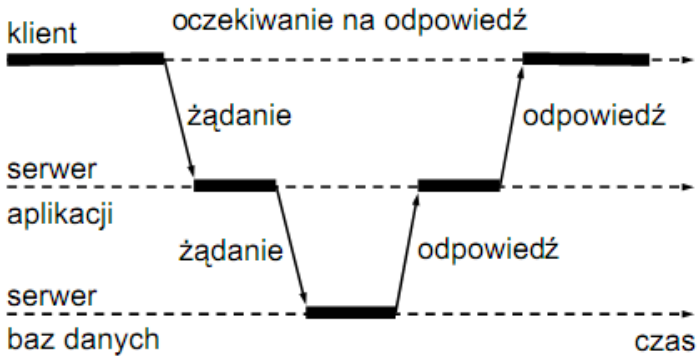
- Decyduje:
  - Elastyczność
  - Wydajność
  - Komunikacja

Architektura klient-serwer

- **Architektura 3-warstwowa**



- Cechy
  - Lepsza skalowalność
  - Większa elastyczność (wymiana/dystrybucja komponentów)
  - Mniejszy transfer w sieci
  - Dodatkowa zwłoka czasowa
  - Ogólnie lepsza wydajność przetwarzania



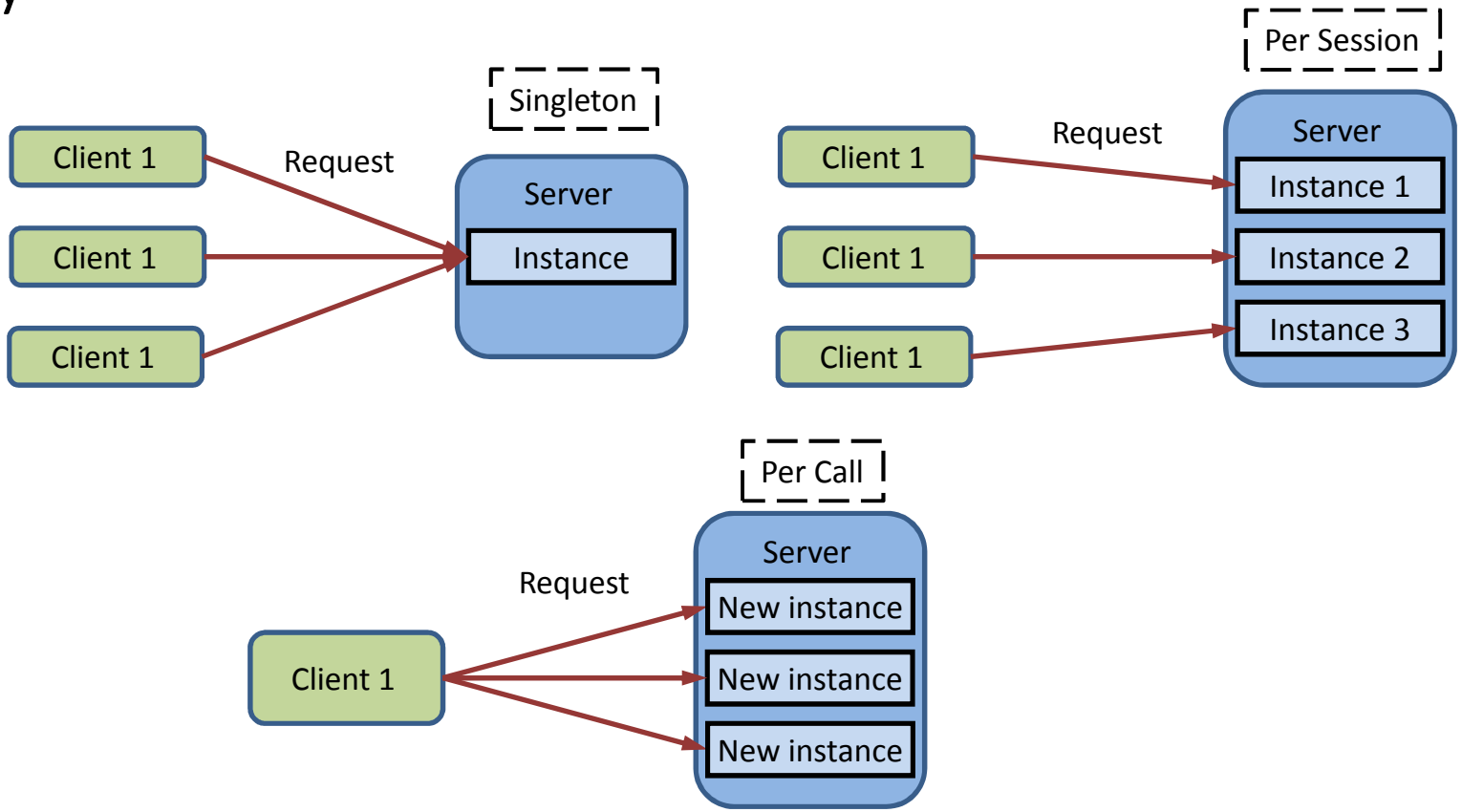
Architektura klient-serwer

- **Klienty**
  - Obsługa interfejsu
    - Grafika
    - Złożone operacje na dokumentach złożonych (od różnych aplikacji)
  - Zapewnianie przezroczystości
    - Położenia, migracji, replikacji, awarii – za pomocą pośrednika
- **Serwery**
  - Sposób obsługiwania żądań od klientów:
    - Serwery iteracyjne
    - Serwery współbieżne
  - Obsługa stanu serwera:
    - Serwery bezstanowe
    - Serwery pełnostanowe

Więcej będzie omówione w kontekście RPC, DO i awarii

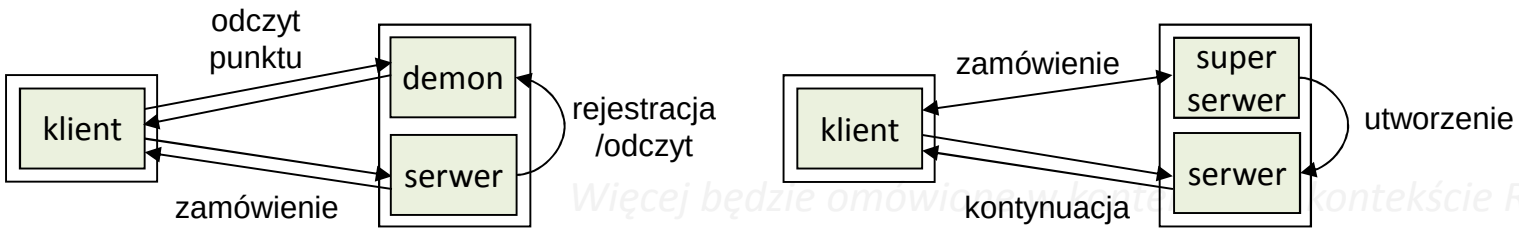
Architektura klient-serwer

- **Zarządzanie instancjami procesów usługowych**
  - Typy



Architektura klient-serwer

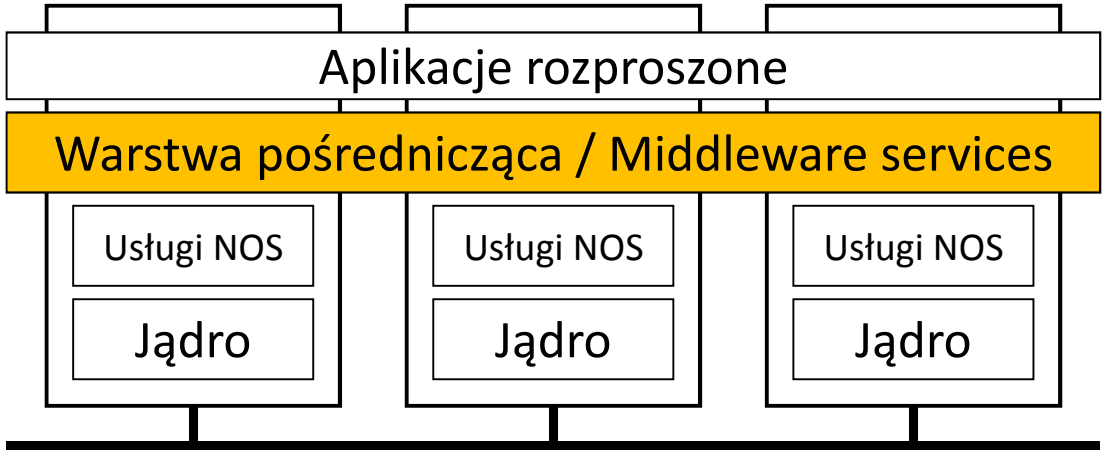
- **Wiązanie klienta z serwerem**
  - Zamówienia są kierowane do **punktu dostępowego**
    - **Punkt końcowy** (ang. *endpoint*) / lub inaczej **port** (uwaga: nie port TCP/UDP)
    - Specyfikuje odbiorcę komunikatów
  - Uzyskiwanie punktu końcowego – sposoby lokalizacji
    - Stałe porty dla dobrze znanych usług
    - Z demonem śledzącym
      - element systemu dostarczający informacji o punktach dostępowych
    - Z superserwerem nasłuchującym
      - j.w. ale na zamówienie tworzy instancje serwera



Więcej będzie omówione w kontekście RPC i DO

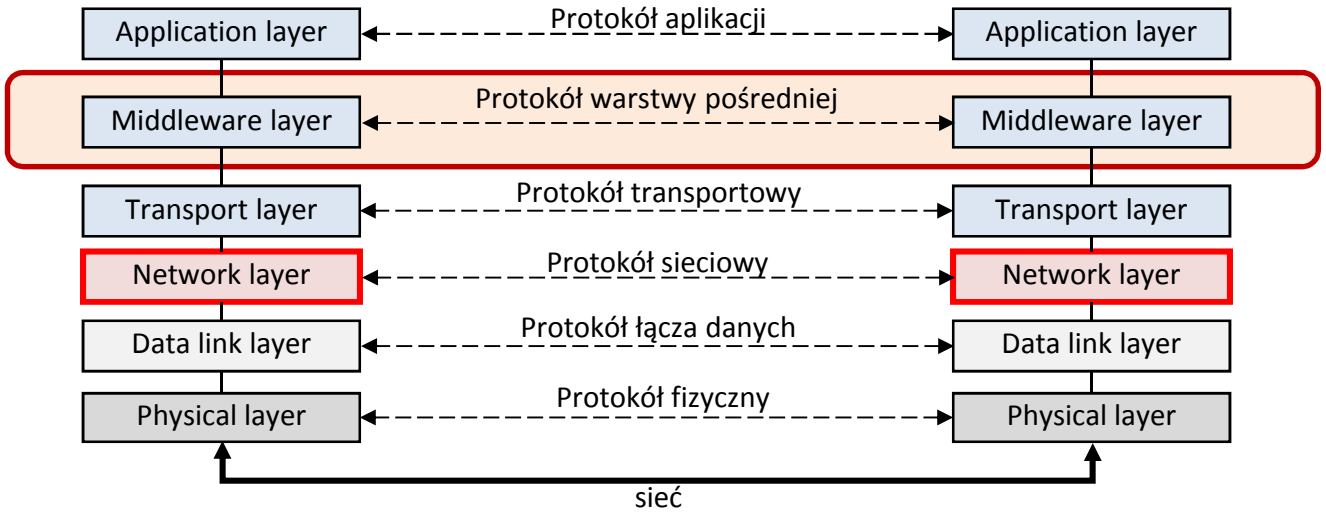
Warstwa pośrednicząca

- **Integracja i ukrycie heterogeniczności systemów**
  - Nadbudowa usług na bazie NOS
  - Zwiększenie poziomu abstrakcji dostępu do usług
    - Zapewnienie przezroczystości
    - Protokoły w każdej warstwie usług pośredniczących, oraz oferowany przez nie interfejs API powinny być takie same
- **Aplikacje zależne tylko od warstwy pośredniczącej**



Warstwa pośrednicząca

- **Warstwa pośrednicząca a sieć**
  - Warstwa sesji i prezentacji zastąpiona warstwą pośrednią
  - Aplikacja logicznie występująca w warstwie aplikacji, ale zawierająca wiele protokołów ogólnego przeznaczenia
  - Stworzona, aby zapewniać wspólne usługi i protokoły, z których może korzystać wiele różnych aplikacji



Warstwa pośrednicząca

- **Usługi warstwy pośredniczącej**
  - Komunikacja
    - Cechy przezroczystości
  - Dostęp do zasobów
    - Koordynacja
    - Zapewnienie jednolitości dostępu
  - Trwałość
    - Zapewnianie trwałości generowanych przez aplikację danych
      - Rozproszony system plików
      - Systemy bazodanowe
      - Rozproszona trwała pamięć dzielona
  - Transakcje rozproszone
    - Niepodzielność operacji na danych rozproszonych
  - Bezpieczeństwo
    - Kompleksowa realizacja w warstwie pośredniczącej

Warstwa pośrednicząca

- **Niektóre typy protokołów wysokiego poziomu:**
  - Wysokopoziomowe otwarte protokoły komunikacyjne
  - Protokoły nazewnictwa/nazewnicze – system nazewniczy
    - Dla lokalizacji elementów systemu
  - Rozproszona autoryzacja i potwierdzanie
  - Wsparcie dla replikacji i przechowywania podręcznego
- **Wysokopoziomowe usługi komunikacyjne warstwy pośredniczącej:**
  - Zdalne wywołanie procedur
    - Komunikacja ukryta w wywołaniu procedury umieszczonej na zdalnej maszynie (Sun RPC, DCE RPC, …)
  - Zdalne wywołanie obiektów
    - Interfejsy obiektu umieszczonego na danej maszynie (poprzez które mogą być wywoływane jego metody) dostępne są na innych maszynach (Java RMI, CORBA, SOAP, …)
  - Usługi kolejkowania komunikatów
  - Komunikacja zorientowana na strumienie

Porównanie rozwiązań

Cecha	DOS		NOS	System z warstwą pośrednicz.
	Multiproc.	Multikomp.		
Stopień przezroczystości	bardzo wysoki	wysoki	niski	wysoki
Jednorodność OS w węzłach	TAK	TAK	NIE	NIE
Liczba kopii OS	1	wiele	wiele	wiele
Komunikacja	pamięć współdzielona	komunikaty	różne	różne
Zarządzanie zasobami	globalne, centralne	globalne, rozproszone	lokalne w węźle	lokalne w węźle
Skalowalność	NIE	średnia	TAK	zmienna
Otwartość	NIE	NIE	TAK	TAK

Współczesne architektury

- **Rozproszenie pionowe**
  - Organizowanie przetwarzania modelem klient-serwer w architekturze wielopiętrowej
  - Uzyskiwane poprzez umieszczanie logicznie odmiennych komponentów na różnych maszynach
- **Rozproszenie poziome**
  - Klient lub serwer jest fizycznie podzielony na logicznie równoważne części, ale każda z nich działa na części całego zbioru danych, równoważąc obciążenie systemu
    - Przykład: sieci CDN (*Content Delivery Network*)
- **Rozproszenie równorzędne (systemy peer-to-peer (P2P))**
  - Usługi wzajemnej współpracy dostępne u klientów sieci rozproszonej dające dostęp do współdzielonych zasobów
    - Moc obliczeniowa, zasoby danych, zasoby komunikacyjne
- **Systemy zorientowane na usługi (SOA)**
- **Mikro-usługi**
- **Systemy agentowe**

Web systems

very short introduction

- **WWW (World Wide Web)**
  - The system of specially formatted, interlinked hypermedia documents, supported (supplied) by special servers (Web servers) and accessed with use of Web clients (especially Web browsers)
    - Built on top of the Internet
    - Allows clients to locate and view multimedia-based documents and to execute web-based applications
    - Documents/resources (provided by web servers/applications) are visible as web pages
- **HTTP (Hypertext Transfer Protocol)**
  - Application layer protocol of WWW service
  - Used for communication between Web entities and transfer of web documents
- **Hypermedia**
  - Multimedia with hyperlinks
  - Contain text, images, videos, animations, and more (including interactive components), interlinked with use of hyperlinks (or simply links)
- **Hyperlinks**
  - Can reference other web pages and/or its components
  - Permit navigation to other web content (resources of Web)

Web systems

very short introduction

- **URL (Uniform Resource Locator)**
  - **URL is a type of URI** that essentially identifies a resource via a representation of its primary access mechanism (e.g. network location)
  - The unified resource addressing format used on the Internet and LANs

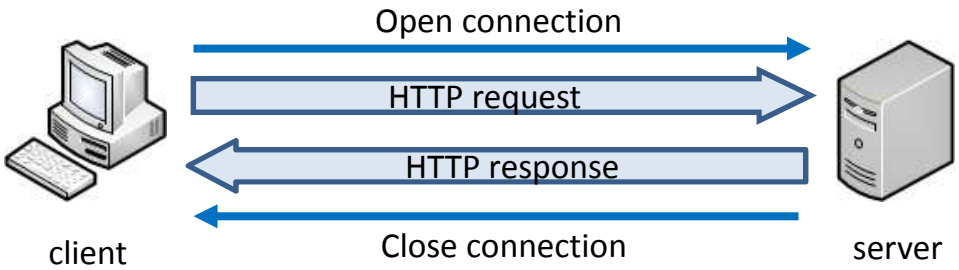
```
URL  =  protocol://host[:port]path[…]
```

    - **protocol** – access protocol and/or type of resource – e.g.: ftp, http, https, file, irc, mailto, telnet, ws, …
    - **host** – FQDN (or IP address) of server on which resource resides
    - **port** – optional transport protocol port
    - **path** – optional path leading to the recourse in the server
      - no path causes return of default resource (defined in the resource server)
      - for security reasons the path location is typically a virtual directory.
      - web server translates the virtual directory into a real location on the server
  - URL of documents available with use of HTTP starts with **http://**
    - E.g.: **`http://www.company.com/mydir/myfile.htm`**

HTTP

very short introduction

- **HTTP (Hypertext Transfer Protocol)**
  - Used for communication between Web entities and transfer of web documents
  - Application layer protocol of WWW service
    - Operates on top of TCP according to client-server architecture
    - Provides a standardized way of client-server communication: **request – responce**



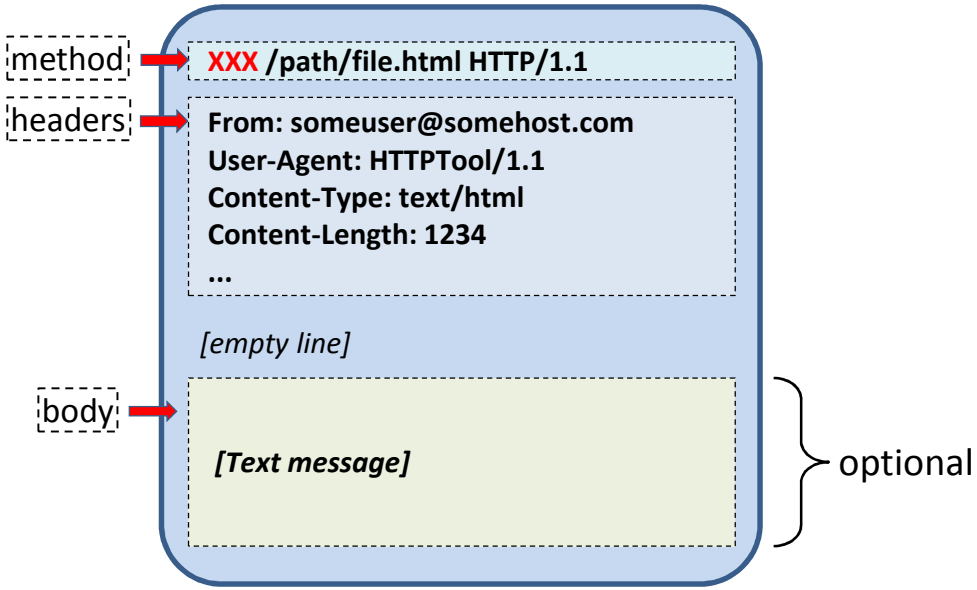
  - HTTP message
    - Single exchange of information between client and server
    - In request a **method** is specified

HTTP

very short introduction

- **HTTP message format**
  - Composed of **header** and **body**
    - **HTTP method**
      - Define request type and server reaction
      - Methods: GET, POST,…
    - Header fields (or headers)
      - Provide additional information about the data that will be sent
      - Examples:
        - **User-Agent** - identifies the client
          Content-Type – specify what format and code is used for the data
    - Responce status
  - POST request
    - A post request sends form data as part of the HTTP message

HTTP message – example:



MIME

very short introduction

- **MIME type (Multipurpose Internet Mail Extensions)**
  - The MIME standard specifies data formats, which programs can use to interpret data correctly.
  - Specified in Content-Type field in HTTP messages
  - Specifies the MIME type of the content that is transmitted in body.
  - Examples
    - **text/html** - the content is HTML document
    - **text/xml** - the content is document described by XML language
    - **image/jpeg** - the content is a JPEG image.
      - When the browser receives this MIME type, it attempts to display the image
    - **video/mpeg** – MPEG-1 standard video file
    - **application/…** - program specific content

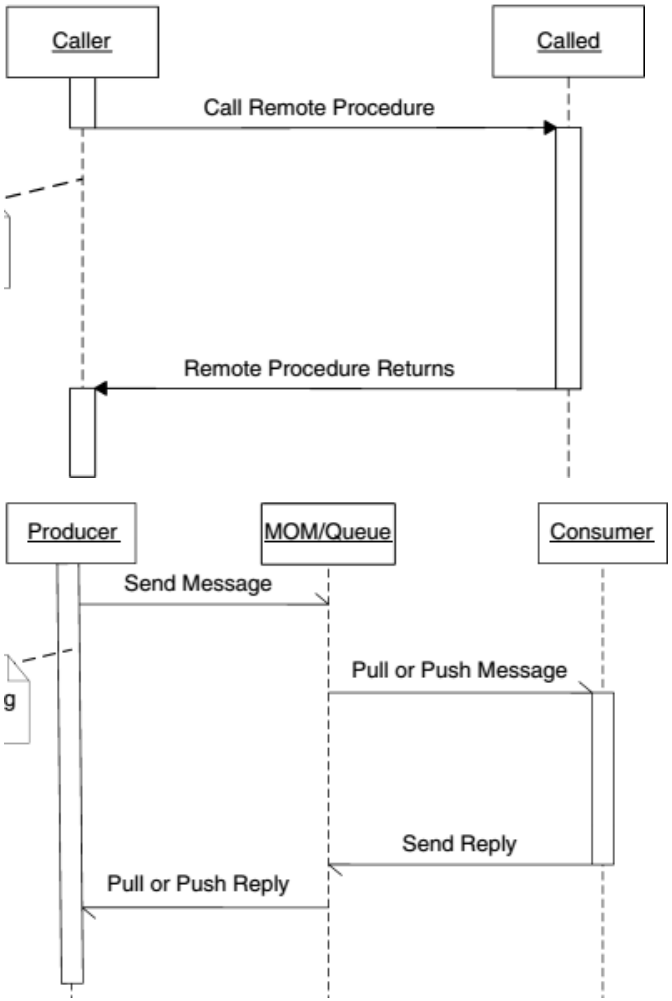
Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

- **Zdalne wywoływanie procedur (RPC)**
(wysokopoziomowe usługi komunikacyjne warstwy pośredniczącej)
  - XML-RPC, JSON-RPC
  - gRPC

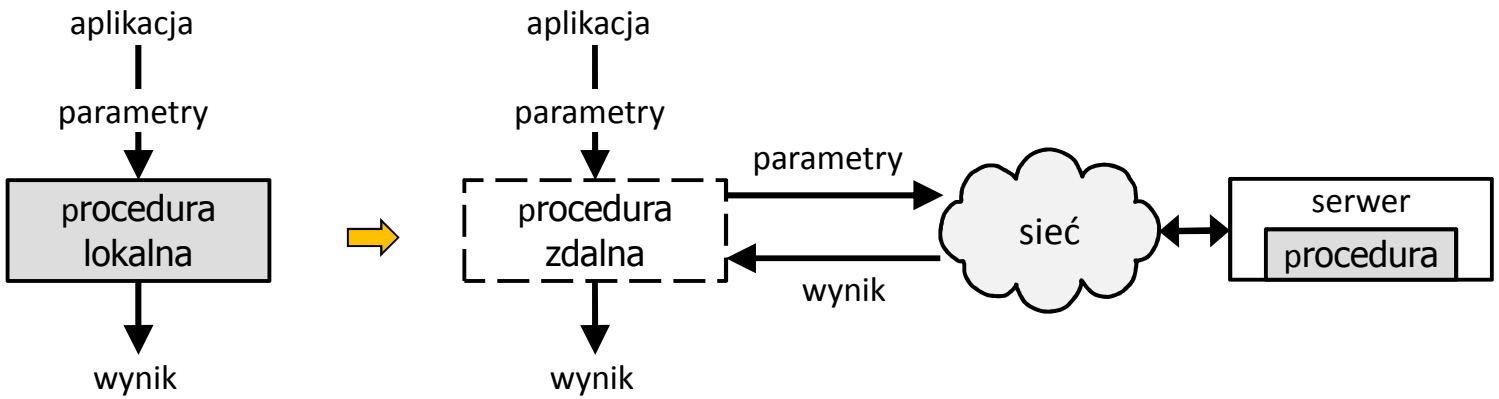
Modele interakcji w komunikacji

- **Dwa ogólne modele interakcji**
  - ***RPC-based***
    - Komponenty komunikujące się są wiązane
    - Zwykle komunikacja synchroniczna
      - Klient czeka na rezultat wywołania procedury
      - Ale są również inne schematy
  - ***MQ-based***
    - Specjalny element pośredniczący
    - Komunikacja asynchroniczna
    - Wszystkie elementy systemu zachowują niezależność przetwarzania
    - Zachowanie kontroli przetwarzania przez obiekt wywołujący
    - Zazwyczaj: **Systemy kolejkowania wiadomości (*Message Queuing Systems*)**

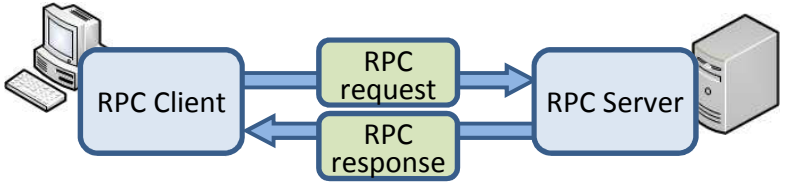


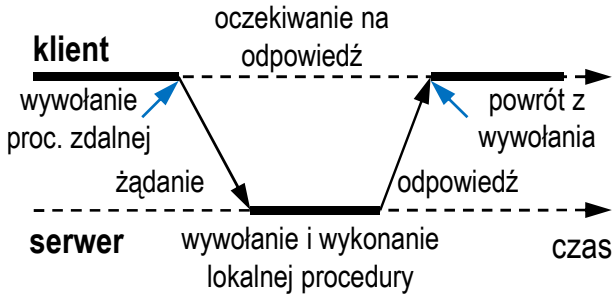
Zdalne wywoływanie procedur (RPC)

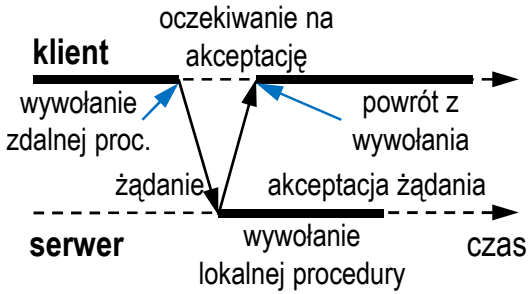
- **Geneza i założenia RPC (ang. *Remote Procedure Call*)**
  - **Problem**: jawna wymiana komunikatów → brak zapewnienia przezroczystości rozproszenia
  - **Cel**: ukrycie komunikacji sieciowej przed aplikacją
  - **Rozwiązanie**: wywoływanie **procedury zdalnej** tak jak lokalnej
  - **Pożądany efekt**: po powiązaniu klienta z serwerem, procedury wywołuje się (programuje się klienta) jakby nie było rozproszenia



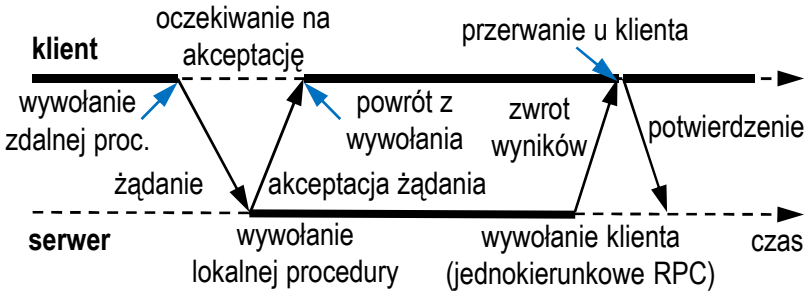
Zdalne wywoływanie procedur





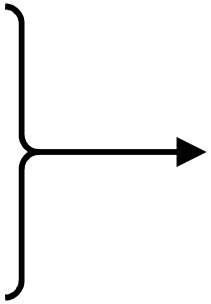


- **Synchroniczne RPC**
- **Asynchroniczne RPC**
- **Odroczone asynchroniczne RPC**



Zdalne wywoływanie procedur

- **Problemy:**
  - Ustalenie serwera realizującego usługę – **wiązanie** (ang. *binding*)
  - Brak możliwości korzystania ze zmiennych globalnych przez procedury zdalne
  - Zapewnienie odporności na awarie
    - Awaria w czasie wywołania procedury
      - Przed zrealizowaniem usługi czy po?
      - Powtarzać wywołanie/żądanie?
  - Różne przestrzenie adresowe procedur wywołujących i wywoływanych
  - Heterogeniczność maszyn

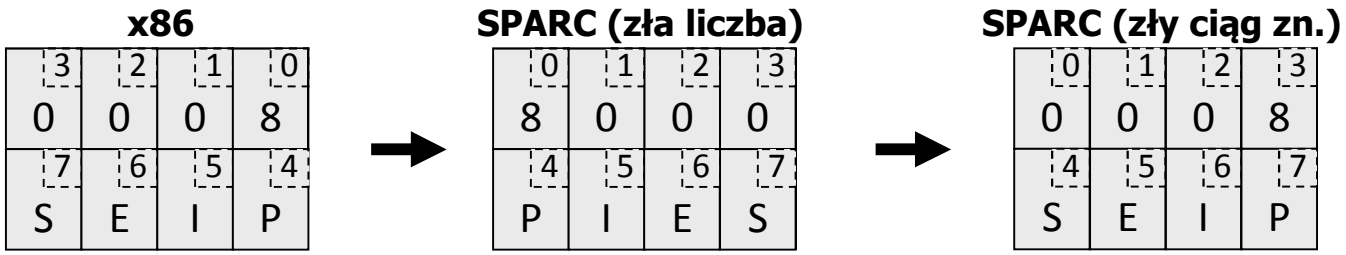


Problem przekazywania parametrów procedur

Zdalne wywoływanie procedur

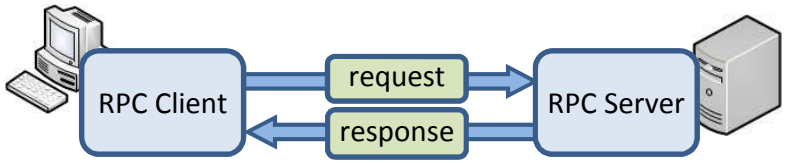
- **Przekazywanie parametrów - problemy**
  - Sposoby przekazywania parametrów
    - W różnych systemach różne typy parametrów są różnie przekazywane
      - Przez wartość (ang. *call by value*)
      - Przez referencję (ang. *call by reference*)
      - Przez kopiowanie i odtwarzanie (ang. *call by copy/restore*)
  - Przekazywanie referencji
    - Różne przestrzenie adresowe $\rightarrow$ wskaźnik **nie odnosi się** do tego samego
    - Kopiowanie i odtwarzanie całych struktur danych – kosztowne
    - Stosuje się usprawnienia:
      - rozróżnianie parametru: wejście lub wyjście
      - tylko wejściowy nie trzeba kopiować z powrotem
  - Specyficzne typy danych
    - Problemy ze złożonymi strukturami danych o nieokreślonym rozmiarze:
      - listy, struktury dla grafów, itp.
    - Stosuje się dodatkowe procedury uzgadniania postępowania

- **Przekazywanie parametrów - problemy**
  - Reprezentacja
    - Kodowanie znaków: Mainframe'y → EBCDIC, PC → ASCII, Unicode
    - Liczby: kody uzupełnieniowe U1, U2, …
    - Kolejność bajtów – np.: x86 → najpierw najmłodszy (ang. *little endian*), Sun SPARC → najpierw najstarszy (ang. *big endian*)
  - Interpretacja parametrów
    - Określenie które parametry są wejściowe, które wyjściowe, a które przekazują dane w obydwu kierunkach
      - czasami parametry (np. odniesienie w C) tego nie specyfikują
    - Określenie rodzaju parametru (liczba, napis,…)
  - Ilustracja:



Zdalne wywoływanie procedur

- **Przetaczanie parametrów (ang. *marshaling*)**
  - Rozwiązanie: niezależny, wspólny, ustalony format reprezentacji i konwersji

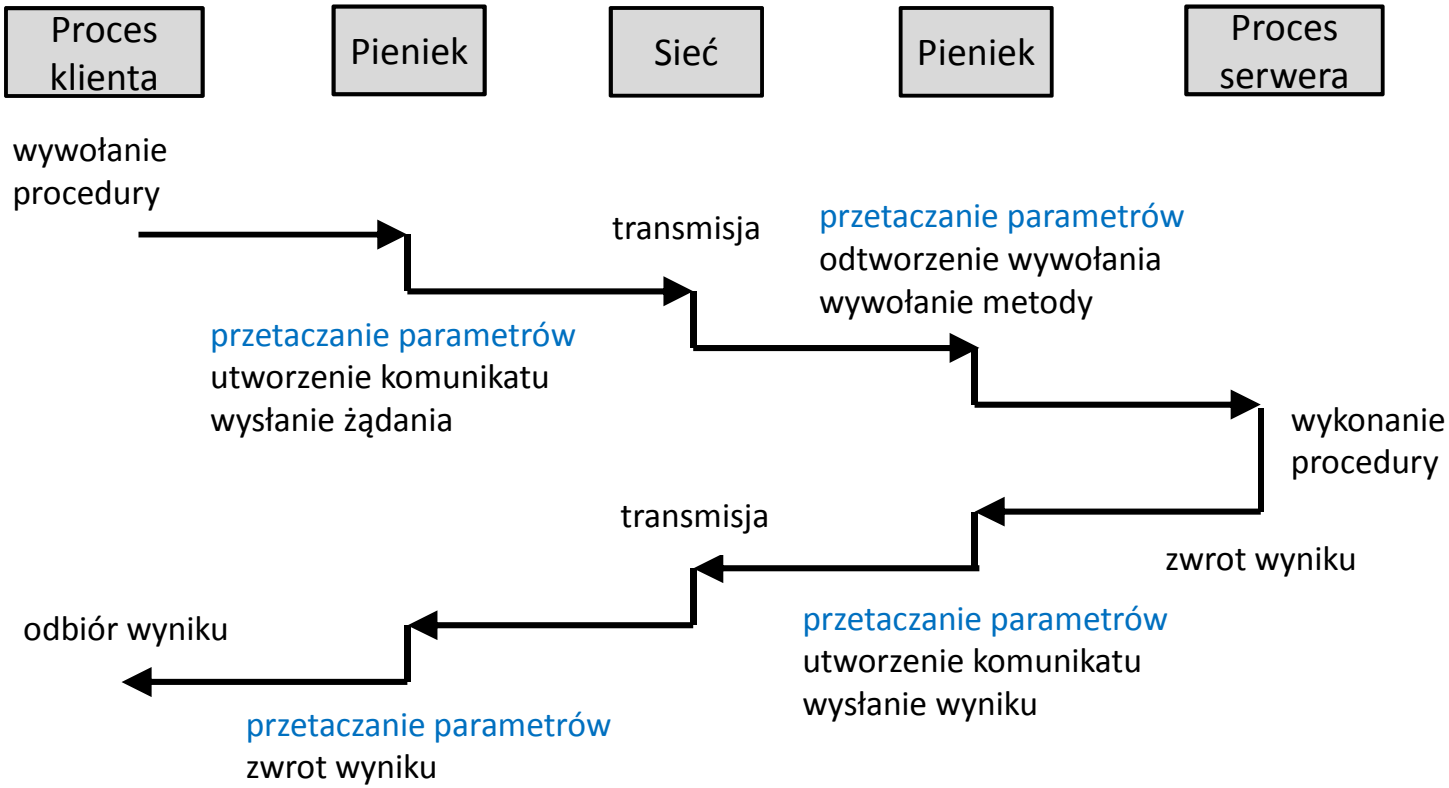


serialize/deserialize

  - Reprezentacja z użyciem uniwersalnego, otwartego formatu niezależnego od platformy
  - Reprezentacje binarne wiadomości
    - Serializacja strukturyzowanych danych do postaci binarnej
    - IDL do opisu intefejsu

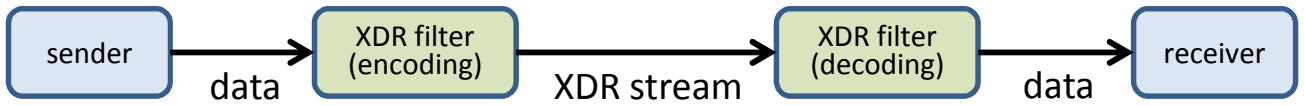
Zdalne wywoływanie procedur

- **Przetaczanie parametrów – ogólny schemat realizacji procedury**



Remote Procedure Call (RPC)

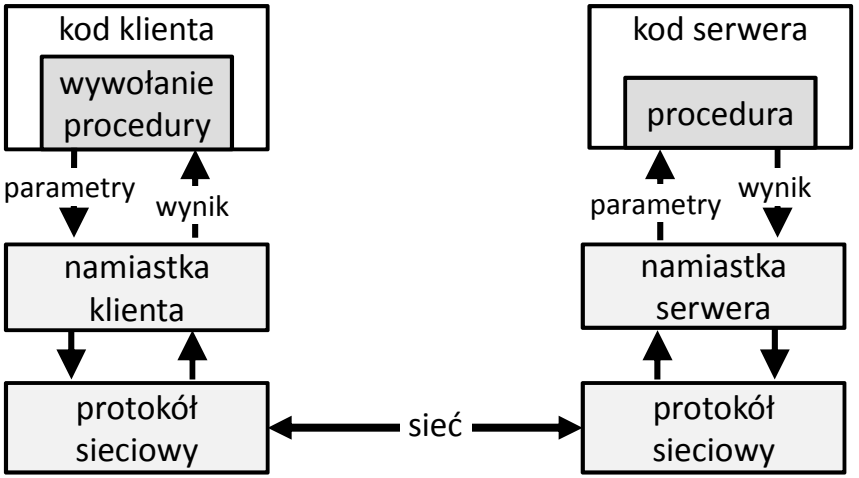
- **Marshaling**
  - Example solutions
    - Text-based data representations
      - E.g.: **XML, JSON**
    - Structured binary representations using specific rules known to sender and receiver
      - E.g.: **Protocol Buffers**
    - XDR filters (eXternal Data Representation)
      - The procedure for encoding/decoding data types from/to the pipe
      - Conventions:
        - Integer numbers: big endian
        - Floating point numbers: IEEE 754
        - Data types have a size of multiples of 4 bytes
        - The conversion is always carried out
        - The sender and recipient always know the type of data transferred



© maf

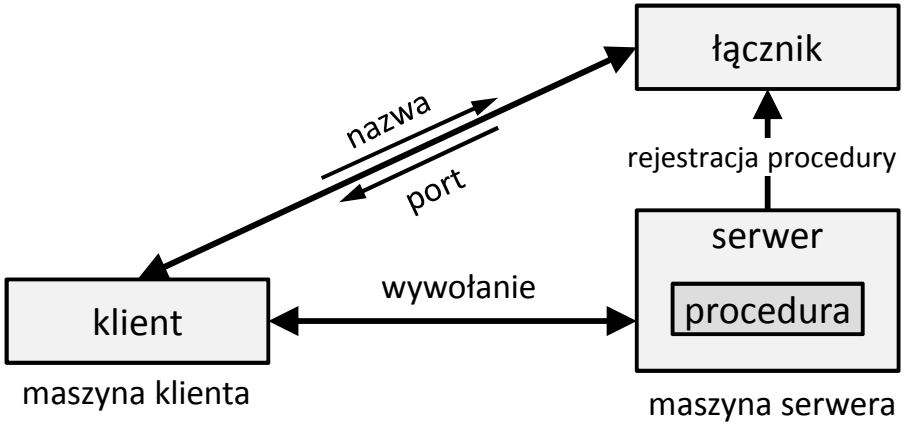
Zdalne wywoływanie procedur

- **Ogólna architektura**
  - **Pieniek/namiastka klienta** (ang. *stub*)
    - Reprezentacja serwera po stronie klienta
    - Komunikaty ***send*** + ***receive*** (z ewentualnym blokowaniem)
  - **Pieniek/namiastka serwera**
    - Reprezentacja klienta po stronie serwera
    - w pętli: ***receive***
    - po wykonaniu: ***send***



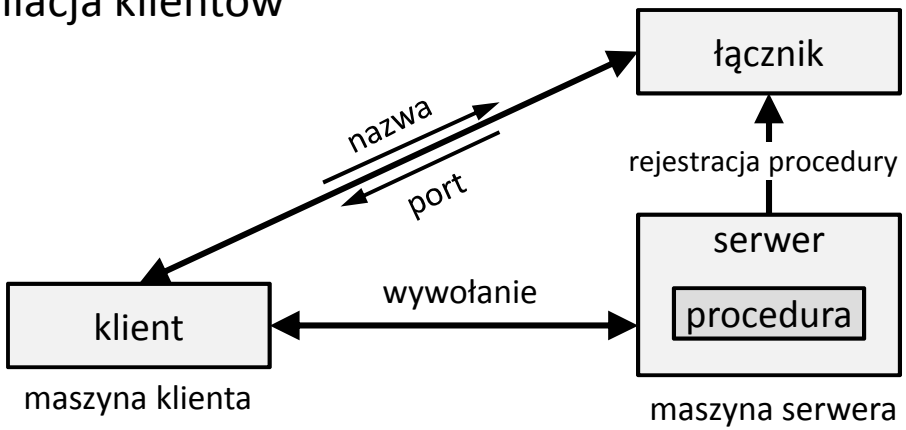
Remote Procedure Call (RPC)

- **Łączenie/wiązanie (ang. *binding*) klienta i serwera**
  - Cel: lokalizacja zdalnej procedury
  - **Łącznik** (ang. *binder*)
    - Specjalna usługa RPC utrzymująca tablicę odwzorowań nazw usług (procedur RPC) na adresy serwerów tych usług
    - Łącznik utrzymywany jest przez serwery które udostępniają adresy klientom



Zdalne wywoływanie procedur

- **Łączenie/wiązanie**
  - Funkcje łącznika:
    - **rejestruj**(nazwa_usługi, wersja, adres_sieciowy)
    - **wyrejestruj**(nazwa_usługi, wersja)
    - **szukaj**(nazwa_usługi, wersja) – zwraca: adres sieciowy
  - 3 strategie
    - Łącznik działa na komputerze którego adres jest dobrze znany
      - adres kodowany w programie
      - lokalizacja się zmienia → rekompilacja klientów
    - Za dostarczanie adresu łącznika odpowiada SO
    - Start klienta lub serwera lokalizuje łącznik za pomocą rozgłaszania



Zdalne wywoływanie procedur

- **Realizacja**
  - Na poziomie RPC usługa dostarczana przez serwer ma postać modułu z interfejsem, który eksportuje zbiór procedur
    - Te procedury można wywoływać z klienta
  - Programowa realizacja RPC (definiowanie interfejsów)
    1. Mechanizm RPC włączony do języka programowania
       - Specjalistyczny język do opisu interfejsu – IDL (ang. *Interface Definition Language*) – np. Sun RPC, Google Protocol Buffers
       - Generowanie tzw. **namiastki/pieńka** (ang. *stub*) – komponent obsługi wywołań
       - Definicja interfejsu w IDL jest kompilowana na namiastki klienta i serwera
       - Kod namiastki zależy tylko od interfejsu zdalnej procedury
    2. Bezpośrednie kodowanie i obsługa wywołań wg ustalonych zasad
       - Komunikujące się jednostki muszą znać wspólne reguły dialogu
       - Rozwiązanie bardziej podatne na błędy

Implementacje/standardy RPC

- **SUN RPC**
  - Tradycyjny standard RPC w systemach typu Unix (te Windows)
  - Najpowszechniejszym zastosowaniem tego standardu jest sieciowy system plików NFS
- **DCE RPC** (DCE = Distributed Computing Environment)
  - Standard wprowadzony przez wiele firm (HP, IBM, DEC)
- **MSRPC**
  - Implementacja/rozszerzenie DCE/RPC w produktach Microsoft
    - Na bazie MSRPC działają bardziej złożone rozwiązania, stosowane na platformie Windows (w .NET pewna zmiana)
    - Komponenty działające w technologii DCOM / ActiveX oraz sieciowy system plików SMB, działają na bazie MSRPC
- **Rozwiązania bazujące na rozwiązaniach Webowych**
  - XML-RPC, JSON-RPC, SOAP, …
- **gRPC**

- **Otwarte formaty dla współpracy otwartych systemów (krótkie przedstawienie)**
  - XML
  - JSON

XML

- **XML = Extensible Markup Language**
  - Tekstowy format reprezentacji strukturyzowanej informacji za pomocą znaczników (ang. *tag / markup*)
  - Meta-język pozwalający definiować zbiory znaczników
    - Dostarcza ustalone reguły pozwalające określać struktury znaczników, notacje i ich funkcje
    - W XML nie ma wbudowanych nazw znaczników (jak w HTML)
- **Znacznik / *Markup / Tag***
  - Wyróżniania fragmenty tekstu nadając im określone znaczenie
  - Specjalne znaki wskazują gdzie zaczyna się i kończy instrukcja znacznika
    - *start-tag* – np. <sometag>
    - *end-tag* – np. </sometag>
    - *empty-element tag* – np. <sometag/>
  - Np.

    <centre> Tu jest <bold> jakiś </bold> tekst. </centre>

    ⬇

    Tu jest **jakiś** tekst.
  - Znaczniki są łatwo rozumiane przez człowieka i maszyny

XML

- **Struktura dokumentu XML**
  - Dokument XML składa się z nazwanych kontenerów i zawartych w nich wartości danych
    - Kontenery są reprezentowane jako **deklaracje**, **elementy** i **atrybuty**
  - **Element**
    - Ciąg znaków pomiędzy znacznikiem otwierającym i zamykającym
    - Wewnątrz jest treść (ang. *content*)
    - Może zawierać elementy zagnieżdżone

```
<Title> My title  </Title>
```

  - **Atrybut**
    - Para **nazwa:wartość** powiązana z elementem i specyfikowana w znaczniku początkowym
    - Dodatkowe dane dokładniej specyfikujące treść elementu

```
<Chapter  chapter-number="7">
   The content of the chapter
 </Chapter>
```

  - ***Namespace* (przestrzeń nazw)**
    - Dostarcza funkcję kojarzenia elementów i atrybutów w całości lub części dokumentu wg określonego schematu oraz unikania konfliktów nazw
    - Zasięg w deklarowanym elemencie i elementach zagnieżdżonych
    - W postaci URI

```
<Address xmlns="http://www.firma.com/Addr">
  …
</Address>
```

XML

- **Struktura dokumentu XML**

Prologue

```
<?xml version="1.0" encoding="UTF-8"?>
<!- Filename: OperationRequest.xml -->

<BookList>
  <Book>
    <Title>Web Services: Principles and Technology</Title>
    <Author>Mike Papazoglou</Author>
    <Publisher>Prentice Hall</Publisher>
  </Book>
  <Book>
     ...
  </Book>
  ...
 <Book>
    <Title>A Canticle for Leibowitz</Title>
    <Author>Walter M. Miller</Author>
    <Publisher>J. B. Lippincott & Co.</Publisher>
  </Book>
</BookList>
```

XML declaration

Comment

Root element

Zagnieżdżanie elementów

XML

- **XML Schema**
  - **XSD – XML Schema Definition Language**
  - The XSD can be used to express a set of rules that an XML document must follow to be considered "valid" according to this scheme
  - The XML schema describes the elements and attributes that can be included in an XML document consistent with this scheme, and how elements can be arranged in the structure of the document
  - Schemas provide support for additional meta-data features, such as structural relations, numerosity, important values and data types
  - Each type of schema works as a method describing data properties and a method of applying rules and restrictions in an XML document

XML

- **XML Schema**
  - XML Schema refers to a document (XML) that specifies the content and structure for XML documents

Person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
 <Person
  xmlns:xsi="http://www.w3c.org/2001/
             XMLSchema-instance"
  xsi:NameSchemaLocation="Person.xsd">

  <FirstName>Nicolaus</FirstName>
  <LastName>Copernicus</LastName>
  <Age>35</Age>

 </Person>
```

Person.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema version="1.0" encoding="UTF-8"?>
 <Person xmlns:xs="http://www.w3c.org/
                   2001/XMLSchema">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FirstName"
         type="xs:string>
        <xs:element name="MiddleName"
         type="xs:string
         minOccurs="0">
        <xs:element name="LastName"
         type="xs:string>
        <xs:element name="Age"
         type="xs:integer>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:schema>
```

XML

- **Transformation of XML documents**
  - The **XSLT** process transforms the XML structure into any other required forms and structures.
  - XSLT uses **XML Path Language** or **XPath** to locate individual fragments of XML documents
    - XPath is a standard for creating expressions that can be used to find specific pieces of information in an XML document.
    - XPath is an abstract language that defines a tree model codifying the logical structure of an XML document
    -

```
<PurchaseOrder>
  <address>
    <street> Dubois st. </street>
    <number> 49 </ number >
    <postal-code> 51-123 </postal-code>
    <city> Wroclaw </city>
  </address>
  <product>
    <product-name> Plastic item </product-name>
    <quantity> 2 </quantity>
  </product>
</PurchaseOrder >
```

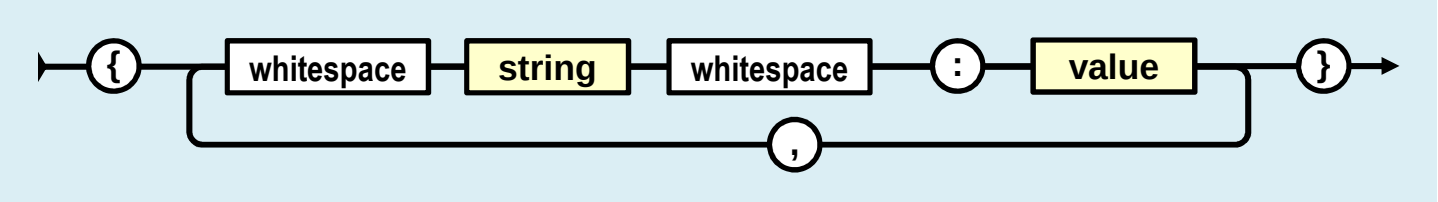
```
<PurchaseOrder>
  <address>
    <street> Dubois st. </street>
    <number> 49 </ number >
    <postal> 51-123 Wroclaw </postal>
  </address>
  <product>
    <product-name> Plastic item </product-name>
    <quantity> 2 </quantity>
  </product>
</PurchaseOrder >
```

JSON

- **JSON**
  - JSON = JavaScript Object Notation
    - Lightweight text-based data-interchange format for computer communication
      - Based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999
    - A completely language-independent data format (but uses conventions that are familiar to programmers of the C-family of languages)
    - It is easy for humans to read and machines to parse
    - Excelent data-interchange language
      - Used in **AJAJ** = Asynchronous JavaScript And JSON (AJAX with JSON)
  - Originally JSON was a strict subset of JavaScript,
    - But now not at 100%
    - Specification actually allows valid JSON documents that aren't valid JavaScript

JSON

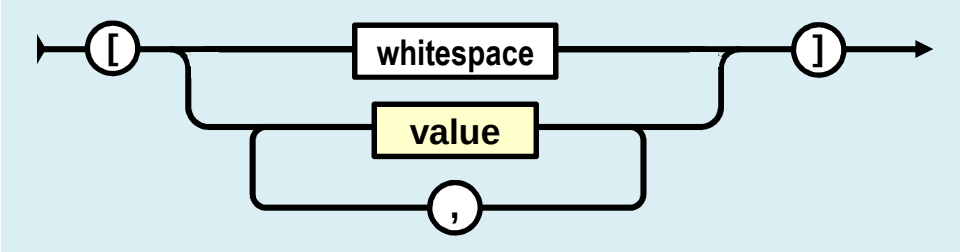
- **JSON format**
  - JSON data is built on two structures:
    - **Object** - an unordered collection of pairs: **name/value**



    - E.g.:

```
{ "firstName":"Jan", "lastName": "Nowak", "high" : 178.5 }
```

    - **Array** - an ordered list of **values**

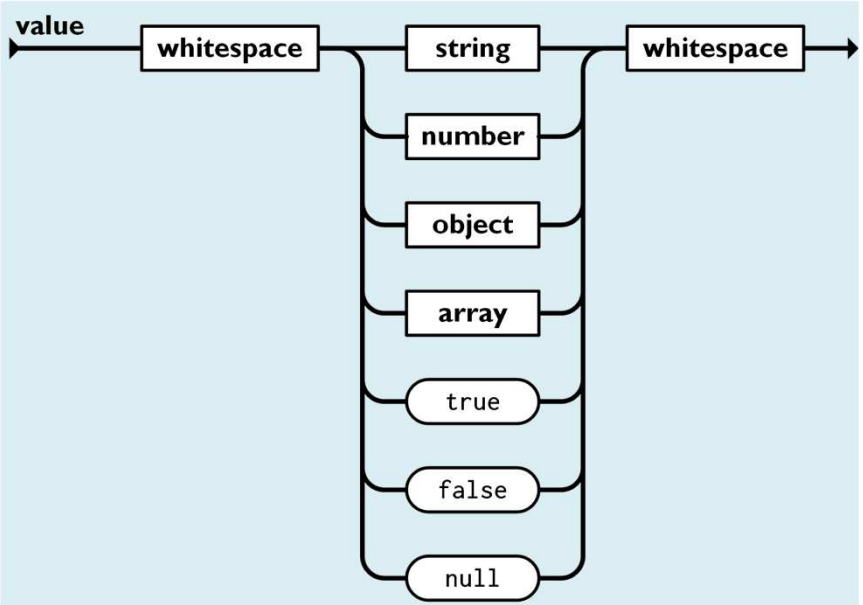


    - E.g.:

```
[ "Jan" , "Nowak" , 178.5 , 29 , "New York" ]
```

JSON

- **JSON format**
  - A value can be:
    - string
    - number
    - object
    - array
    - true or false
    - null
- **All components can be nested**

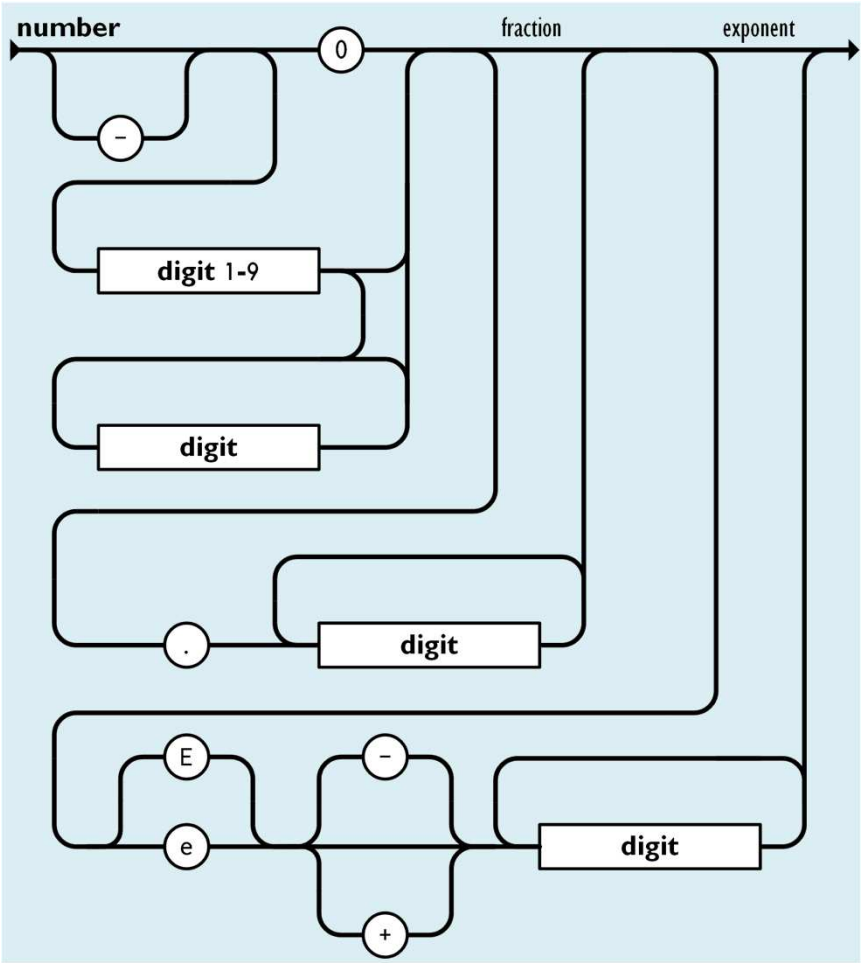


complex data structures

```
{ "id": 123,
  "first_name": "Jan",
  "lastName": "Nowak",
  "phone": [
              { "type":"private", "number":"71-123-45-67" },
              { "type":"office", "number":"71-987-65-43" }
            ]
}
```

JSON

- **JSON format**
  - A number
    - very much like a C or Java number, except that the octal and hexadecimal formats are not used.
    - Examples:
      123 , -178.5 , 3.7e-12
  - A string
    - a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes.
      - A character is represented as a single character string.



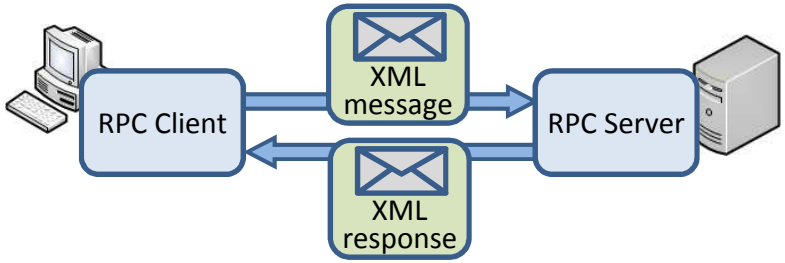
JSON

- **JSON format**
  - JSON data structures are universal .
    - Most modern programming languages support them in some form.
    - A collection of name/value pairs.
      - In various languages, this is represented as an object, record, struct, dictionary, hash table, keyed list, or associative array.
    - An ordered list of values.
      - In most languages, this is represented as an array, vector, list, or sequence.
- **MIME type**
  - The official MIME type (Content-Type for HTTP) is: **"application/json"**
    - most modern implementations have adopted this
    - The unofficial MIME type **"text/json"** and the content-type **"text/javascript"** get legacy support by many browsers, servers, web applications, libraries, frameworks, and APIs.
- **JSON Schema**
  - Specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control.
  - JSON Schema is based on the concepts from XML Schema (XSD)

- **XML-RPC**
- **JSON-RPC**
- **gRPC**

XML-RPC

- **XML-RPC (eXtensible Markup Language – RPC)**
  - Protokół umożliwiający zdalne wywoływanie procedur w heterogenicznym środowisku w Internecie
  - Używa kodowania wiadomości XML
    - Żądania i odpowiedzi
    - Żądanie: nazwa procedury i parametry
      - Wiele parametrów wywołania
      - Typy podstawowe, tablice i struktury
    - Jeden parametr zwrotny
      - Kod błędu lub wiadomość XML
  - Używa transportu HTTP
    - Wiadomość: żądanie HTTP-POST
  - Specyfikacja obejmuje
    - Model danych
    - Struktura żądań
    - Struktura odpowiedzi



serialize/deserialize

XML-RPC

- **Typy danych**
  - Podstawowe

Nazwa	Znacznik i przykład	Opis
boolean	<boolean>1</boolean>	Logiczna wartość (0 or 1)
integer	<i4>123</i4> lub <int>123</int>	4-bajt. wart. całkowitoliczbowa
double	<double>-12.34</double>	Wartość zmiennoprzecinkowa podwójnej precyzji
string	<string>Hello world!</string>	Ciąg znaków. Musi być zgodne z kodowaniem XML
date/ time	<dateTime.iso8601>14100715T12:01:55</dateTime.iso8601>	Format daty ISO 8601
base64	<base64>eW91IGNhbid0IHJlYWQgdGhpcyE=</base64>	Kodowanie binarne Base64

  - Tablice
  - Struktury

XML-RPC

- **Typy danych**
  - Tablice
    - Elementy różnych typów
    - Wielowymiarowe (zagnieżdżanie)

```
<array>
  <data>
    <value><i4>1410</i4></value>
    <value><string>Jakiś tekst</string></value>
    <value><i4>1</i4></value>
  </data>
</array>
```

  - Struktury
    - Składowe również tablice

```
<struct>
  <member>
    <name>promien</name>
    <value><i4>12</i4></value>
  </member>
  <member>
    <name>pole</name>
    <value><double>452.16</double></value>
  </member>
</struct>
```

XML-RPC

- **Żądanie**
  - Nagłówek musi specyfikować:
    - User-Agent
    - Host
    - Content-Type: **text/xml**
    - Content-length
  - Dane
    - Jedna struktura `methodCall`
    - Nazwa metody `methodName`
      - <a-z|A-Z|0-9|/|.|,>
    - Parametry (jeśli są) `params`
    - Dowolna liczba parametrów

```
POST /XMLRPC HTTP/1.0
User-Agent: myXMLRPCClient/1.0
Host: test.pwr.wroc.pl
Content-Type: text/xml
Content-length: 217
-----------------------------------------------------------------
<?xml version="1.0"?>
<methodCall>
  <methodName>MetodaRPC</methodName>
  <params>
    <param>
      <value><int>13</int></value>
    </param>
    <param>
      <value><double>3.14</double></value>
    </param>
  </params>
</methodCall>
```

XML-RPC

- **Odpowiedź**
  - Zawiera opis błędu lub wiadomość
  - Jeden parametr
    - Może być struktura → wiele parametrów
    - Jedna struktura `methodResponse`
  - Nagłówek musi specyfikować:
    - Content-Type: **text/xml**
    - Content-length

```
HTTP/1.1 200 OK
Date: Tue, 26 Mar 2013 17:05:01 GMT
Server: Apache.1.3.12 (Unix)
Connection: close
Content-Type: text/xml
Content-Length: 124
-----------------------------------------------------------------
<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value><double>18.24668429131</double></value>
  </param>
 </params>
</methodResponse>
```

XML-RPC

- **Odpowiedź**
  - Błąd wykonania w sekcji `fault`
  - Typowa odpowiedź z błędem:

```
<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>9</int></value>
               </member>
            <member>
               <name>faultDescription</name>
               <value><string>Too many parameters.</string></value>
            </member>
         </struct>
      </value>
   </fault>
</methodResponse>
```

XML-RPC

- **Programowanie**
  - Wiele bibliotek dla różnych języków programowania
    - Java: Apache XML-RPC, aXMLRPC, Redstone XML-RPC Library, …
    - C++: Libiqxmlrpc, XML-RPC for C and C++, AnyRPC, …
    - Python: xmlrpc.client
    - PHP: XML-RPC for PHP, Zend\XmlRpc (Zend Framework 2.2)
    - Java-Script: XML-RPC for Node.js, JS-XMLRPC, …
  - Apache XML-RPC
    - Pakiet **org.apache.xmlrpc**
    - **WebServer** – minimalny serwer HTTP
      - Metody: start, shutdown, run, …
      - addHandler – dodaje obiekt serwera z procedurą zdalną
    - **XmlRpcClient** – klient RPC
      - Metody: execute, executeAsync
      - Parametry: java.util.List (Vector, ArrayList,…) lub java.lang.Object[]
    - … inne klasy/interfejsy

XML-RPC

- **Przykład realizacji**
  - Apache XML-RPC

Klient

```
import java.util.*;
import org.apache.xmlrpc.*;

public class XMLClient {
 public static void main (String [] args) {
  try {
   int port = xxx;    // xxx - proper value
   XmlRpcClient agent = new XmlRpcClient("http://server.address.com:" + port);
   Vector params = new Vector();
   params.addElement(new Integer(13));
   params.addElement(new Integer(66));
   Object result = agent.execute("myServer.add", params);
   int suma = ((Integer) result).intValue();
    System.out.println("The sum is: " + suma);

  } catch (Exception ex) {
     System.err.println("XMLClient: " + ex);
    }
 }
}
```

XML-RPC

- **Przykład realizacji**
  - Apache XML-RPC

Serwer

```
import org.apache.xmlrpc.*;

public class XMLServer
{
  public Integer add(int x, int y)
  {
    return new Integer(x+y);
  }

  public static void main (String [] args) {
    try {
      System.out.println("Attempting to start XML-RPC Server...");
      WebServer serwer = new WebServer(xxx);        // xxx - proper value
      serwer.addHandler("myServer", new XMLServer());
      serwer.start();
      System.out.println("Started successfully.");
    } catch (Exception ex) {
     System.err.println(„XMLServer: " + ex);
    }
  }
}
```

JSON-RPC

- **JSON-RPC**
  - Protokół RPC podobny do XML-RPC
  - Jako dane transportowe **obiekty JSON**
    - Specyfikacja nie precyzuje protokołu transportowego
    - Stosowany jest transport HTTP lub gniazda TCP/IP
  - The official MIME type (Content-Type for HTTP) is: **"application/json"**
    - The unofficial MIME type **"text/json"** and the content-type **"text/javascript"** also get legacy support by many browsers, servers, web applications, libraries, frameworks, and APIs.

JSON-RPC

- **Działanie**
  - Dwa równorzędne elementy wymieniają dane poprzez łącze
  - Żądania
    - Procedury wywoływane przez żądania
  - Odpowiedzi
    - Na żądania muszą być odpowiedzi
  - Zawiadomienia
    - Na zawiadomienia nie ma odpowiedzi
- **Wersje**
  - RPC 1.0
    - peer-to-peer
  - RPC 2.0
    - client-server
    - Niezależność od warstwy transportowej
    - Dodatkowe pole wersji
    - Redukcja pól w komunikatach
    - Rozszerzenia dla komunikatów

JSON-RPC

- **Żądania** – jeden obiekt serializowany zgodnie z JSON
  - **id** - wartość dowolnego typu, używana do dopasowania reakcji do zapytania na które udzielana jest odpowiedź
  - **method** - ciąg znaków z nazwą metody do wywołania
  - **params** - tablica wartości przekazywanych jako parametry do zdalnej procedury

```
[13, 14]
```

```
["tak", "nie"]
```

```
[ { "param1":13}, {"param2":"tak" } ]
```

    - W wersji 2.0 są także ***named parameters***

```
{ "param1":13, "param2":14 }
```

  - **jsonrpc** – ciąg znaków specyfikujący wersję 2.0 JSON-RPC
- **Przykłady**

```
{"id": 1, "method": "subs", "params": [13, 16]}
```

```
{"id": 1, "jsonrpc":"2.0", "method": "subs", "params": [13, 16]}
```

JSON-RPC

- **Odpowiedzi**
  - Muszą zawierać:
    - **id** - id zapytania, na które odpowiada
    - **result** - dane zwrócone przez wywoływane metody
      - Jeśli wystąpił błąd podczas wywoływania metody, wartość ta musi wynosić **null**
    - **error** - kod błędu (jeśli nie było błędu musi być null)
    - W wersji 2.0 pomijane są ”puste” składowe: albo **result** albo **error**
  - Przykłady
    - v. 1.0

```
{ "result": -3, "error": null, "id": 1 }
```

    - v. 2.0

```
{ "id": 1, "jsonrpc":"2.0", "result": -3 }
```

```
{ "jsonrpc":"2.0", "result": -3, "id": "1" }
```

JSON-RPC

- **Błąd (v. 2.0)**
  - Obiekt który zawiera
    - **code** – kod błędu
    - **message** – wiadomość - opis błędu
    - **data** – struktura zawierająca dodatkowe dane

```
{ "id": 1, "jsonrpc": "2.0", "error": {"code": -101, "message": "Method not found"} }
```

- **Zawiadomienia**
  - Gdy nie oczekujemy odpowiedzi
    - Bez id:
      - null w wersji 1.0
      - pomijane w wersji 2.0
    - v.1.0

```
{ "method": "upd", "params": [11,222], "id": null }
```

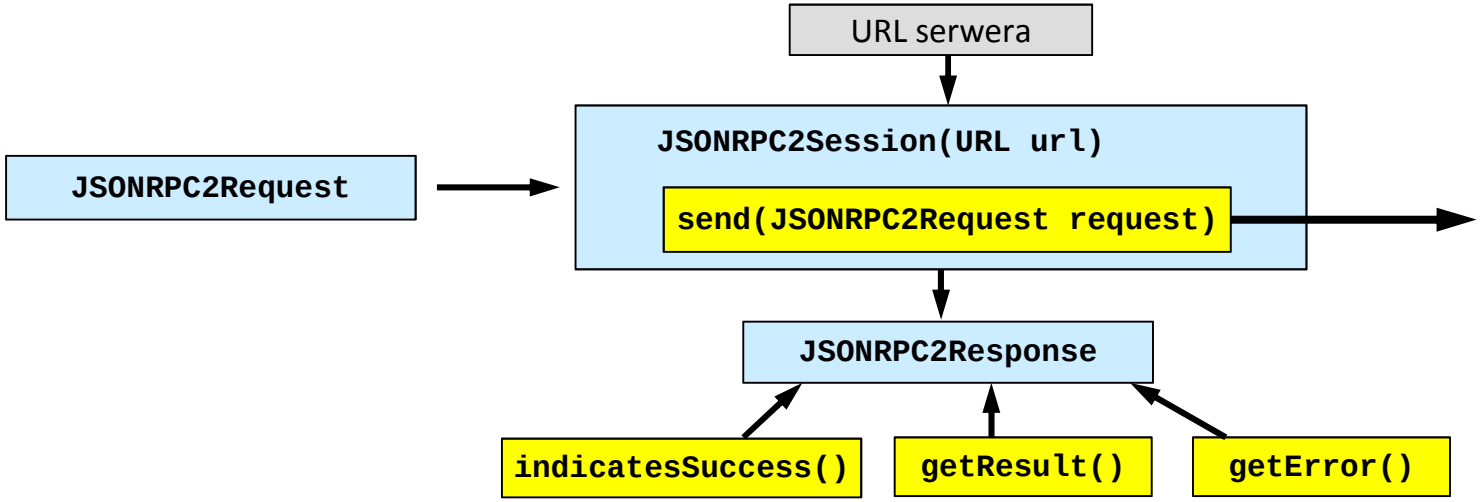
    - v.2.0

```
{ "jsonrpc": "2.0", "method": "upd", "params": [11,222] }
```

JSON-RPC

NIE OMAWIANY DODATEK

- **Przykład realizacji**
  - com.thetransactioncompany.jsonrpc2
  - Ogólny schemat programowania **klienta**




```
JSONRPC2Session(URL url)

JSONRPC2Request(String method, Object id)
JSONRPC2Request(String method, Map<String,Object> namedParams, Object id)
JSONRPC2Request(String method, List<Object> positionalParams, Object id)

JSONRPC2Response(Object result, Object id)
JSONRPC2Response(JSONRPC2Error error, Object id)
```

JSON-RPC

NIE OMAWIANY DODATEK

- **Przykład realizacji**
  - com.thetransactioncompany.jsonrpc2

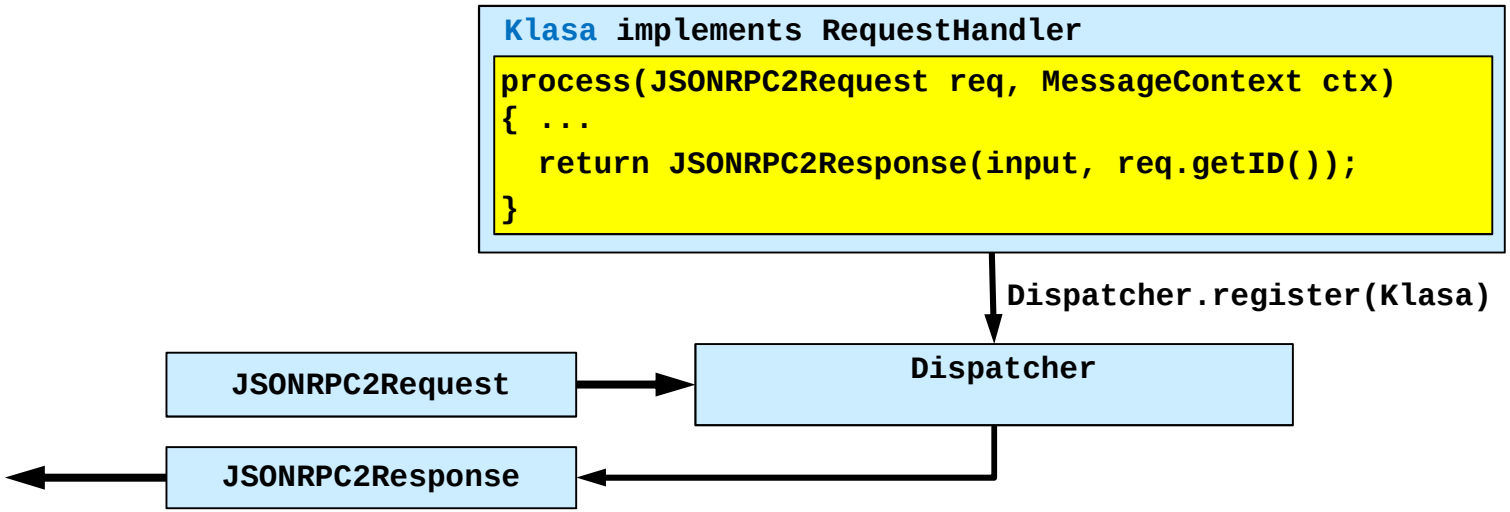
Klient

```
import com.thetransactioncompany.jsonrpc2.*;
...
URL serverURL = null;
try { serverURL = new URL("http://adres.serwera.com:8080");
    } catch (MalformedURLException e) { ... }
JSONRPC2Session mySession = new JSONRPC2Session(serverURL);

String method = "nazwaMetody";
int requestID = 1;
Map<String,Object> params = new HashMap<String,Object>();
params.put(„person", „Jan Nowak");
params.put(„high", 178.5);

JSONRPC2Request request = new JSONRPC2Request(method, params, requestID);
JSONRPC2Response response = null;
try {  response = mySession.send(request);
    } catch (JSONRPC2SessionException e) { ... }

if (response.indicatesSuccess())
      System.out.println(response.getResult());
else
      System.out.println(response.getError().getMessage());
```

JSON-RPC

NIE OMAWIANY DODATEK

- **Przykład realizacji**
  - com.thetransactioncompany.jsonrpc2
  - Ogólny schemat programowania **serwera**




```
public interface RequestHandler
  process(com.thetransactioncompany.jsonrpc2.JSONRPC2Request request,
          MessageContext requestCtx)

Dispatcher()
  register(RequestHandler handler)
  String[] handledRequests()
```

JSON-RPC

NIE OMAWIANY DODATEK

- **Przykład realizacji**
  - com.thetransactioncompany.jsonrpc2

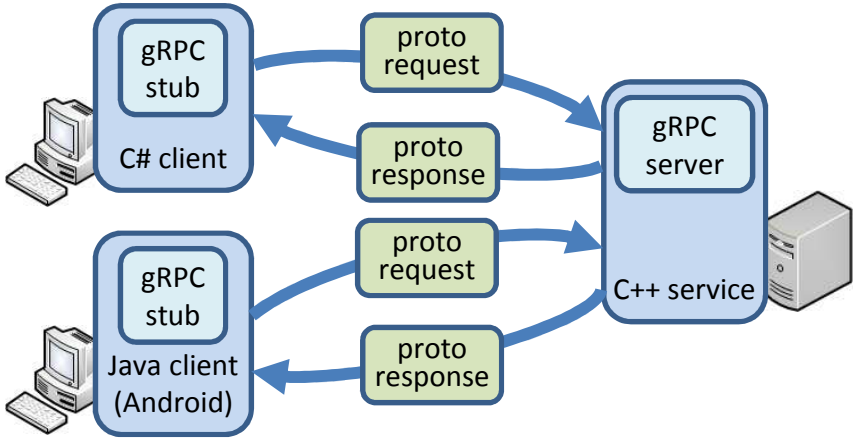
Server

```
import com.thetransactioncompany.jsonrpc2.*;
...

public static class mojaMetodaHandler implements RequestHandler {
  public String[] handledRequests() {
           return new String[]{"nazwaMetody"};
  }
  public JSONRPC2Response process(JSONRPC2Request req, MessageContext ctx) {
    if (req.getMethod().equals(„nazwaMetody"))
    { ...
      return new JSONRPC2Response(odpowiedz, req.getID());
    }
    else {
      return new JSONRPC2Response(JSONRPC2Error.xxx, req.getID());
    }
 }

public static void main(String[] args) {
  Dispatcher dispatcher =  new Dispatcher();
  dispatcher.register(new mojaMetodaHandler());
  ...
}
```

gRPC

- **gRPC (gRPC Remote Procedure Calls)**
  - An open source solution
  - It generates cross-platform client and server bindings for many languages.
  - RPC system that uses
    - HTTP/2 for transport and
    - **Protocol Buffers** as the IDL,

    with the folowing features:
    - authentication
    - bidirectional streaming and flow control,
    - blocking or nonblocking operation,
    - cancellation and timeouts.



gRPC

- **Protocol Buffers (Protobuf)**
  - A method of serializing structured data useful in programs to communicate with each other over a wire, or for storing data.
    - The method involves an **IDL** that describes the structure of some data … and …
    - a program (**code generator**) that generates source code from that description for generating or parsing a stream of bytes that represents the structured data.
  - Versions
    - proto2
    - proto3
  - Java, C++, C#, Python, PHP, Objective-C, a lite-runtime (Android Java/Kotlin), Ruby, Go, and JavaScript
    - Extra from separate vendors (e.g. Scala, …)

gRPC

- **Protobuf message**
  - Data structures called *messages* defined in the file **.proto**
  - Message – **name-value** pairs
  - Message fields (proto2):
    - **required**: must have exactly one of this field.
    - **optional**: can have zero or one of this field (but not more)
    - **repeated**: can be repeated any number of times (including zero)

example.proto

```
message Book {
  required int32 id = 1;
  required string title = 2;
  optional string subtitle = 3;

  message Author {
    required string name = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated Authors author = 4;
}
```

  - Message fields (proto3):
    - **singular**: can have zero or one of this field (but not more than one) – default
    - **repeated**: as above

gRPC

- **Service and messages definition**
  - The proto file defines the service, remote methods and parameters used (input and output)

example2.proto

```
service Hello {
  rpc sayHello (HelloRequest) returns (HelloResponse)
}

  message HelloRequest {
    string req = 1;
  }

  message HelloResponse {
    string reply = 1;
  }
}
```

  - 4 method types
    - Unary RPC – standard request-response (as above)
    - Server streaming RPC
    - Client streaming RPC
    - Bidirectional streaming RPC

```
rpc met (TheRequest) returns (stream TheResponse)
```

```
rpc met (stream TheRequest) returns (TheResponse)
```

gRPC

- **protoc compiler**
  - A defined proto file is compiled with **protoc** tool
  - It is generated appropriate source code for projects - e.g .:
    - C++
      - hello.pb.h - the header which declares your generated message classes
      - hello.pb.cc - contains the implementation of your message classes
      - hello.grpc.pb.h - the header which declares your generated service classes
      - hello.grpc.pb.cc - contains the implementation of your service classes
    - C#
      - Hello.cs - contains all the protocol buffer code to populate, serialize, and retrieve our request and response message types
      - HelloGrpc.cs - provides generated client and server classes
    - Java
      - Hello.java (and others) - contain all the protocol buffer code to populate, serialize, and retrieve our request and response message types.
      - HelloGrpc.java - contains (along with some other useful code):
        - a base class for Hello server to implement
        - stub classes that clients can use to talk to a server
      - special Builder classes for creating message class instances

gRPC

- **protoc compiler**
  - In generated classes simple accessors (seters and getters) – e.g.:
    - C++
      - fieldname() - e.g. title()
      - set_fieldname() - e.g. set_title()
    - Java
      - getFieldname() - e.g. getTitle()
      - setFieldname() - e.g. setTitle()
  - Various additional auxiliary methods
    - Testing data field (e.g. has_title ())
    - Cleaning the fields
    - Parsing and serialization of data
    - …
  - Java message classes are immutable
    - Constructing classes with use of builder classes

```
Book solaris =
  Book.newBuilder()
    .setId(1234)
    .setTitle("Solaris")
    .addAuthors(
      Book.Authors.newBuilder()
        .setAuthor("Lem"))
    .build();
```

gRPC

- **Operation support**
  - Synchronous and asynchronous operation
    - Language specific
  - Deadlines/Timeouts
    - How the deadline or timeout is specified varies from language to language
  - RPC termination and Cancelling
  - Channels
    - A gRPC channel provides a connection to a gRPC server on a specified host and port and is used when creating a client stub
    - Clients can specify channel arguments to modify gRPC's default behavior
  - Authentication
    - SSL/TLS
    - Token-based authentication with Google

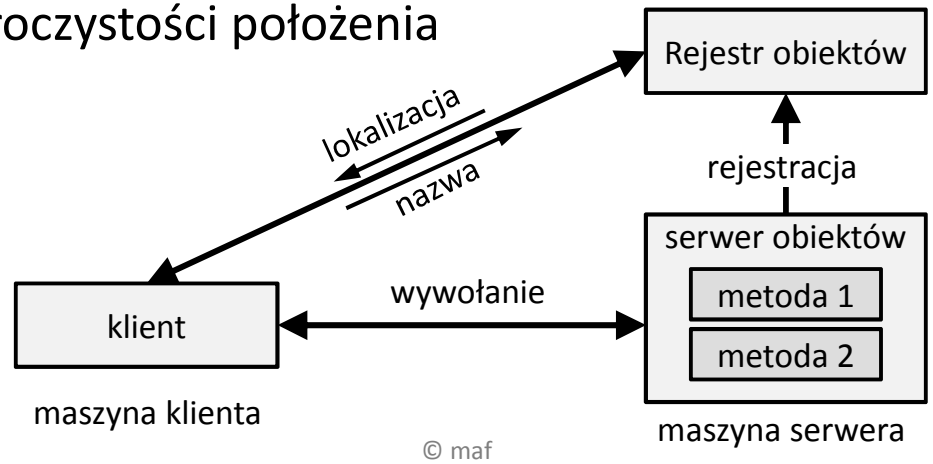
Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

- **Wysokopoziomowe usługi komunikacyjne (warstwy pośredniczącej)**
  - Obiekty rozproszone (DO)
    - Java RMI
  - Message Oriented Midleware (MOM) / Systemy kolejkowania wiadomości

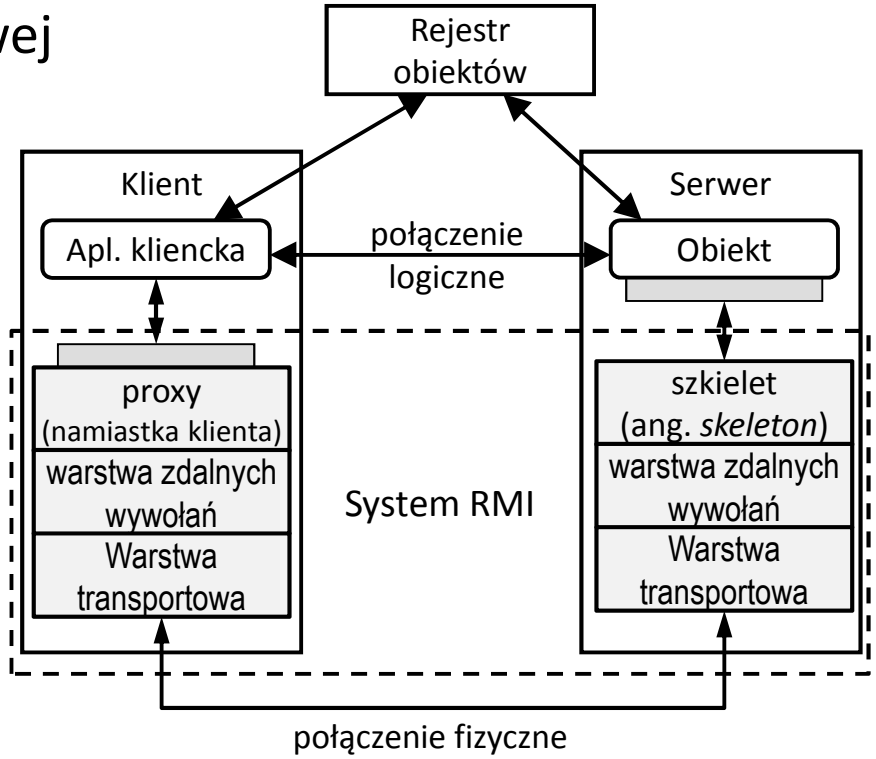
Obiekty rozproszone

- **Obiektowe RPC – zdalne wywoływanie metod**
  - Dane i operacje są zawarte w obiekcie
    - Stan obiektu
    - Operacje jako metody dostępne przez interfejsy
  - Model RPC wzbogacony o zarządzanie referencjami do obiektów
    - Pozwala przezroczyście odwoływać się do metod zdalnych obiektów
    - Serwer obiektów odpowiada za zarządzanie obiektami
    - Rejestr obiektów umożliwia lokalizację obiektów
      - Wsparcie przezroczystości położenia



Obiekty rozproszone

- **Pieniek klienta → pośrednik (ang. proxy)**
  - Implementuje interfejs taki jak u obiektu zdalnego
  - Przetwarzanie wywołań metod zdalnego obiektu na odpowiednią postać komunikatu sieciowego
  - Ładowany do przestrzeni adresowej podczas wiązania klienta z serwerem
- **Pieniek serwera → szkielet serwera**
  - Przetaczanie i wywoływanie obiektu
- **Funkcje rejestru:**
  - Umożliwienie serwerowi rejestracji obiektu
  - Umożliwienie klientowi lokalizacji obiektu zdalnego



Obiekty rozproszone

- **Rodzaje obiektów**
  - Implementacja obiektu
    - Obiektowa – obiekty poziomu języka (obiekty fazy kompilacji)
      - Z poziomu języka obiektowego (C++, Java,…)
      - Obiekt jest instancją klasy
      - Automatyczna generacja proxy i szkieletu
      - Wada: zależność od języka programowania
    - Nieobiektowa – obiekty budowane w fazie wykonania
      - Dowolny sposób implementacji
      - Wymagany adapter obiektu
      - Zadanie adaptera: nadanie implementacji charakteru obiektu
  - Trwałość obiektu
    - Obiekty ulotne – istnieją tak długo jak serwer
    - Obiekty trwałe – przechowywane w pamięci trwałej
      - Mogą być odczytywane/sprowadzane z przez różne serwery
  - Rozproszenie stanu obiektu
    - Obiekty są rozproszone (dostęp zdalny)
    - Obiekt jest rozproszony

Obiekty rozproszone

- **Ogólnosystemowe referencje (odniesienia) obiektowe**
  - Globalny identyfikator obiektu (też zdalnego)
    - Wykorzystywany dla wskazania obiektu
    - Musi zawierać informacje wystarczającą do realizacji dowiązania
      - Uzyskanie informacji niezbędnej do wywołania metody obiektu
    - Odniesienie określa serwer, obiekt i protokoły komunikacji
    - Możliwość swobodnego przesyłania referencji pomiędzy procesami
  - Problem reprezentacji adresu w odniesieniu
    - Adres sieciowy : punkt końcowy serwera : wskazanie obiektu
      - Punkt końcowy – zwykle port sieciowy
      - Wrażliwość na awarie, słabe wsparcie migracji
    - Rozwiązanie z demonem zarządzającym przypisania portów
      - Też ma wady: nie można przenosić demona i obiektów do innej lokalizacji
    - Rozszerzenie idei demona do rozwiązania z serwerem lokalizacji
      - Odniesienie = adres serwera lokalizacji + identyfikator obiektu
      - Problemy ze skalowalnością

Obiekty rozproszone

- **Schemat realizacji działań**
  - Uzyskanie referencji do obiektu zdalnego
    - w programie klienta reprezentowana przez standardową referencję
  - Utworzenie **wiązania** do zdalnego obiektu na podstawie referencji
  - Wywołanie metody związanej z obiektem zdalnym
  - Przetaczanie argumentów przez proxy
  - Lokalizacja serwera z obiektem zdalnym na podstawie referencji
    - Przekazanie danych do modułu komunikacji
  - Transmisja wywołania w warstwie sieciowej
  - Odszukanie na podstawie referencji, szkieletu i zdalnego obiektu po stronie serwera
  - Przetaczanie argumentów i wywołanie metody
  - Realizacja usługi
  - …

Obiekty rozproszone

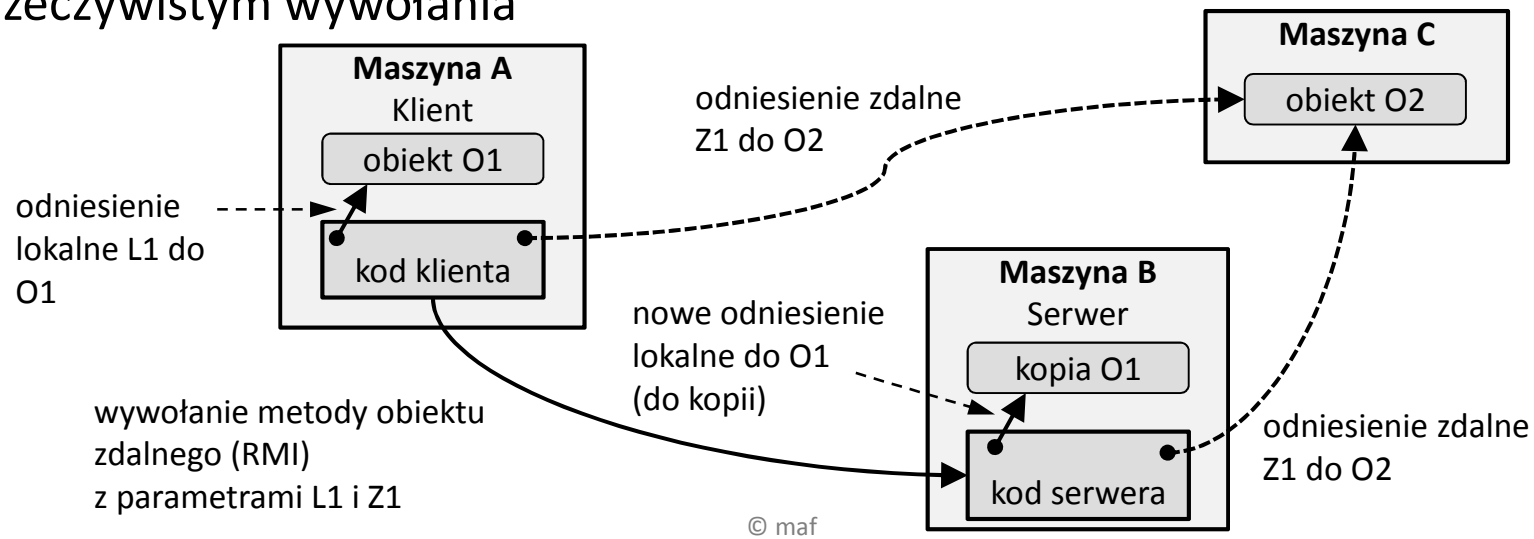
- **Wiązanie**
  - Dowiązanie tworzy pośrednika
    - Pośrednik tworzony dla określonego obiektu
    - Można tu dodatkowo ustanawiać funkcjonalność pośrednika
      - W referencji może być uchwyt do implementacji pośrednika
  - Typy wiązań
    - Niejawne
      - ustanowienie dowiązania w procesie klienta do obiektu na skutek wywołania metody za pośrednictwem referencji
    - Jawne
      - wymaga wywołania odpowiedniej funkcji (np. bind)
      - wywołanie metody dopiero następne

```
Rem_obj* ob_ref;      //ogólnosystemowa referencja
ob_ref = …;           //inicjalizacja referencji
ob_ref->metoda();     //wywołanie metody
                      //(niejawne związanie)
```

```
Rem_obj* ob_ref;          //ogólnosystemowa referencja
Loc_object* = ob_wsk;     //wsk. do lokalnego obiektu
ob_ref=…;                 //inicjalizacja referencji
ob_wsk = bind(ob_ref);    //jawne wiązanie
ob_wsk->metoda();         //wywołanie metody
```

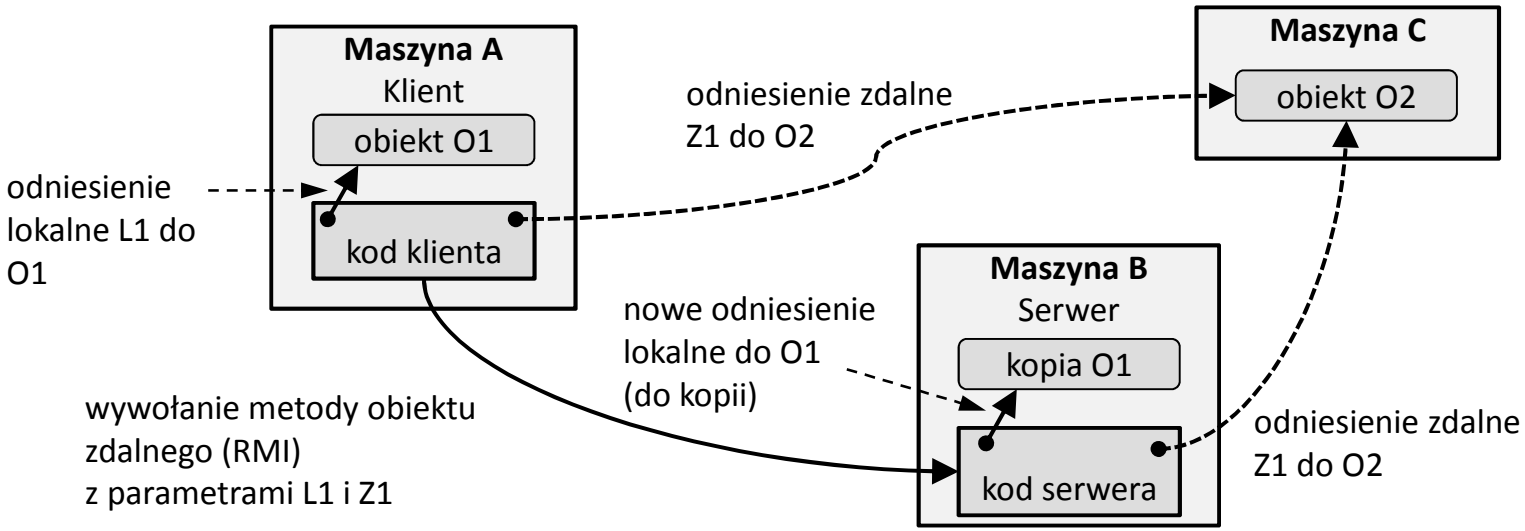
Obiekty rozproszone

- **Przekazywanie obiektów jako parametry**
  - 2 sposoby przekazywania parametrów będących obiektami
    - Przez odniesienie/referencję – dla obiektów zdalnych
      - Do zdalnej metody przekazywana jest referencja do obiektu, który jest parametrem rzeczywistym
    - Przez wartość – dla obiektów lokalnych
      - Do zdalnej metody przekazywana jest kopia obiektu, który jest parametrem rzeczywistym wywołania



Obiekty rozproszone

- **Obiekty zdalne a obiekty lokalne**
  - Rozróżnienie narusza przezroczystość i utrudnia pisanie aplikacji
    - Tylko obiekty zdalne → mniej efektywne
    - Przekazywanie przez wartość **wymaga serializacji obiektu**



Java RMI

- **Na środowisko składają się 3 procesy**
  - Proces klienta – wywołuje metody zdalnych obiektów
  - Proces serwera – dostarcza obiekty zdalne
  - Proces rejestru – wiąże obiekty z nazwami
    - Obiekty zgłaszane są przez serwer
    - Klient musi poznać nazwę
    - Używając nazwy klient może uzyskać referencję do obiektu
- **Etapy konstrukcji aplikacji**
  - Uruchomienie (aktywacja rejestru)
    - Może być uruchamiany w procesie serwera (przy starcie)
  - Utworzenie interfejsu zdalnego obiektu
  - Implementacja interfejsu (obiektu zdalnego) na serwerze
  - Rejestracja zdalnego obiektu
  - Uzyskanie referencji do zdalnego obiektu
  - Utworzenie proxy dla klienta i szkieletu dla serwera

Java RMI

- **Definicja interfejsu**
  - Dla realizacji proxy. Określa komunikację
  - Klasa zdalnego obiektu musi implementować interfejs **`java.rmi.Remote`**

```
public interface RemObject extends Remote {
  public Object remMethod(SomeParameterType p) throws RemoteException;
}
```

- **Implementacja interfejsu – obiekt na serwerze**
  - Może rozszerzać **`java.rmi.server.UnicastRemoteObject`** (**`RemoteObject + RemoteServer`**)
  - Klasa może implementować wiele zdalnych interfejsów

```
public class ImplRemObject extends UnicastRemoteObject implements RemObject {
  public ImplRemObject() throws RemoteException {
    super();
  }
  public Object remMethod(SomeParameterType p)
  { ... }
}
```

Java RMI

- **Rejestracja obiektu zdalnego – w serwerze**
  - Z użyciem **java.rmi.Naming**

```
import java.rmi.Naming;
public static void main()
{
  if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
  }
  try {
    String obj_name = "my_RemObiekt_name";
    RemObiekt ob = new ImplRemObiekt();
    Naming.rebind("host_URL" + "/" + obj_name, ob);
     or:  Naming.bind("host_URL" + "/" + obj_name, ob);
  }
  catch(...) {...}
}
```

    - URL w postaci: rmi://*adres_hosta/nazwa_obiektu*
  - Z użyciem **java.naming.Context**
    - Po utworzeniu obiektu typu `Context` – niemal tak samo

Java RMI

- **Rejestracja obiektu zdalnego – w serwerze (c.d.)**
  - Można w programie serwera tworzyć rejestr

```
 ...
 Registry reg = LocateRegistry.createRegistry(int port);
 ...
```

- **Uzyskanie referencji do obiektu zdalnego – w kliencie**
  - Z użyciem **java.rmi.Naming**

```
import java.rmi.Naming;
public static void main() {
 ...
 try {
   RemObject ob = (RemObject) Naming.lookup("host_URL" + "/object_name");
   ...
   ob.remMethod(parameter);    //wywołanie metody jak dla lokalnego obiektu
 }
 catch(...)
 {...}
}
```

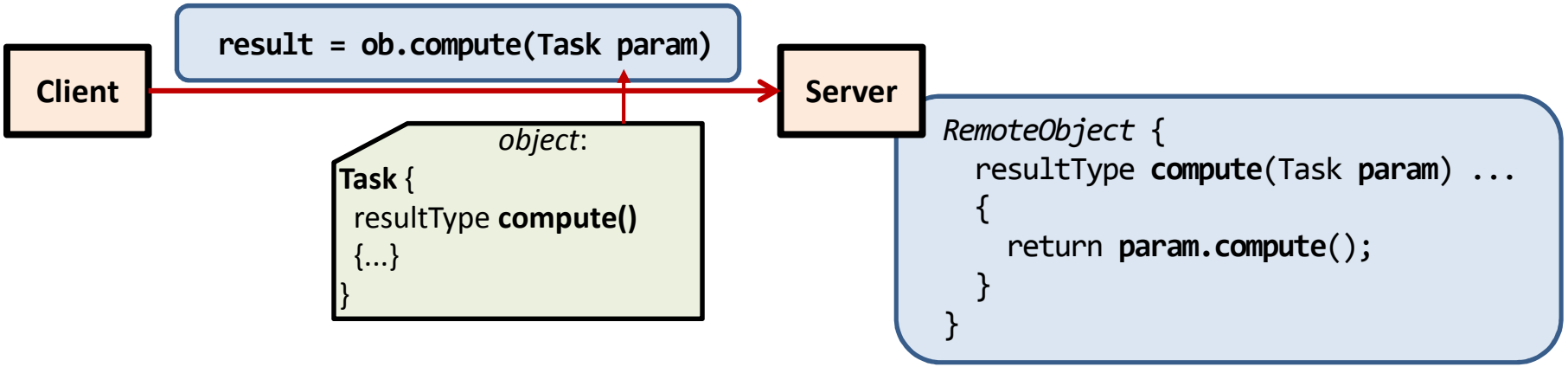
  - url w postaci: rmi://*adres_hosta/nazwa_obiektu*

Java RMI

- **Przekazywanie parametrów (obiektów)**
  - Obiekty lokalne
    - Klasa tego obiektu musi implementować: `java.io.Serializable`
  - Obiekty zdalne
    - Klasa tego obiektu musi rozszerzać interfejs: `java.rmi.Remote`
    - Zdalne metody obiektu zdalnego muszą wyrzucać wyjątek: `java.rmi.RemoteException`

Przetwarzanie rozproszone dowolnych zadań

- **Schemat ogólny**
  - Serwer–Worker ma zdalny obiekt z metodą, która wywołuje metodę parametru obiektu-zadania
  - Klient tworzy obiekt-zadanie i przekazuje go w wywołaniu



- **Istnieje problem "widzialności" klasy obiektu-zadania**
  - o tym później

Przetwarzanie rozproszone dowolnych zadań

- **Schemat ogólny**
  - Interfejsy **`Task`** i **`Worker`** znane klientowi i workerom
  - **`resultType`** odpowiedni do zadania

```
class MyClient
{
  Create / Bind to worker
  myComputations()
  {
    MyTask[] tasks ← create some tasks
    results = worker.wCompute(tasks[i]);
  }
}
```

```
class MyTask implements Task {
  resultType compute() {
    ... some computations
  }
}
```

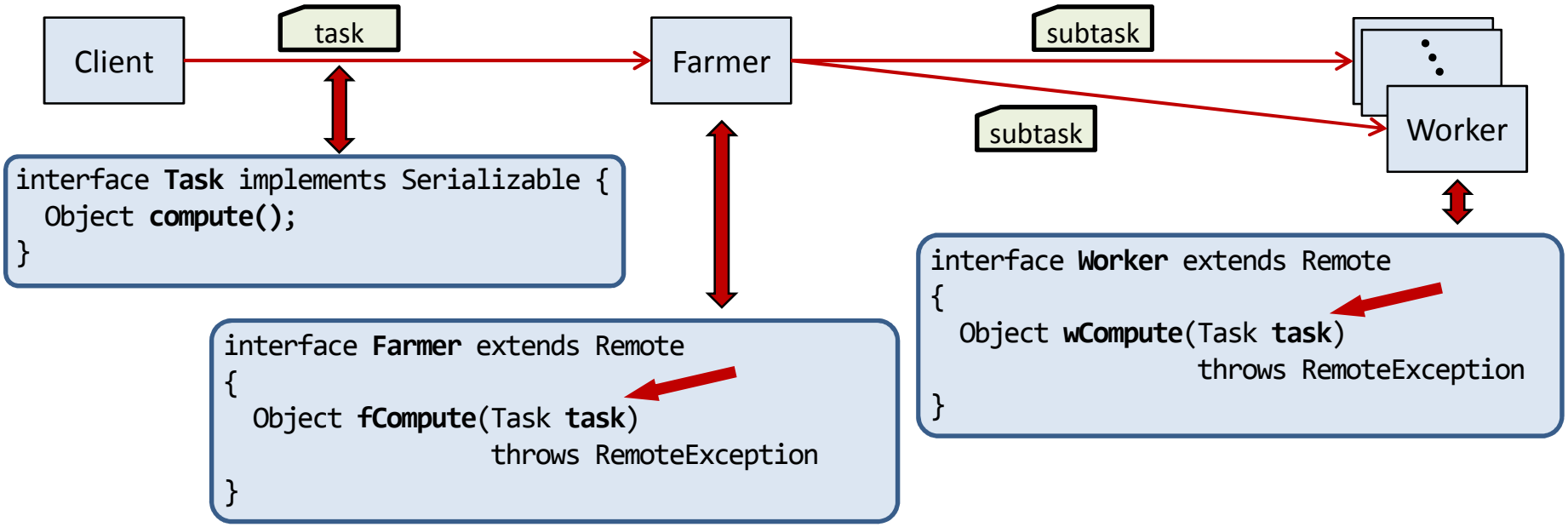
```
interface Task implements Serializable
{
  Object compute();
}
```

```
interface Worker extends Remote
{
  Object wCompute(Task t)
         throws RemoteException ;
}
```

```
class MyWorker extends UnicastRemoteObject
               implements Worker {
  Object wCompute(Task t) throws RemoteException {
    return t.compute();
  }
}
```

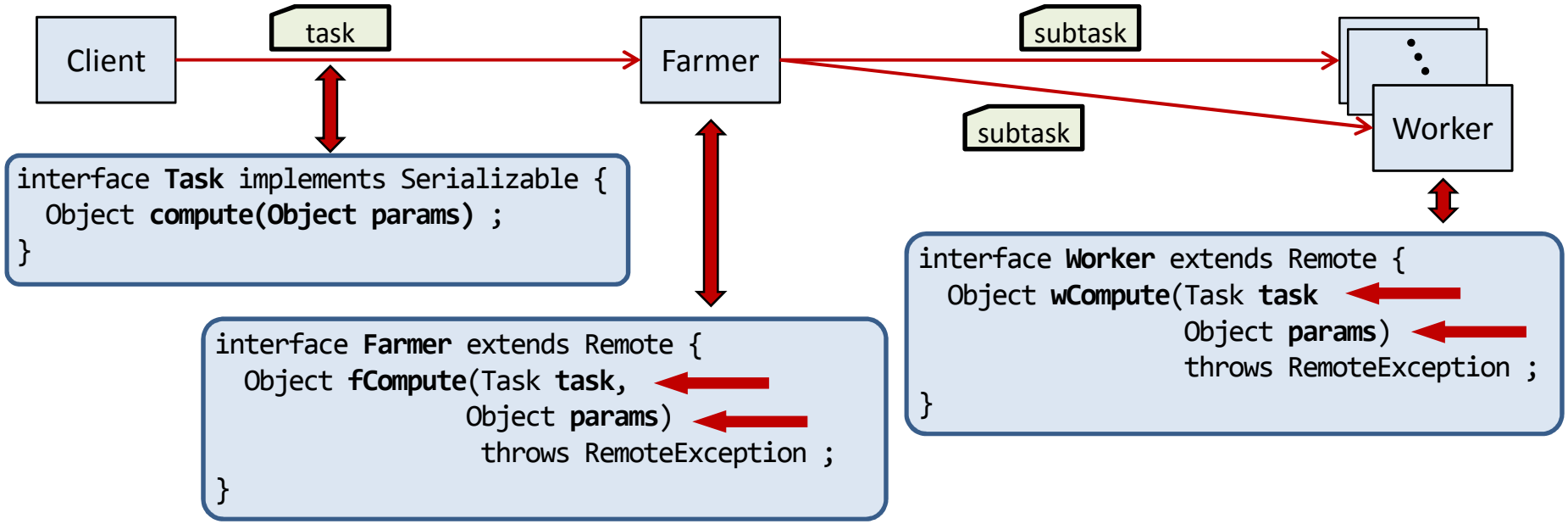
Model Farmer-Worker – schemat

- **Model Farmer-Worker**
  - Pośredni element (Farmer) dzieli zadanie i dystrybuuje je do wielu serwerów-workerów
- **Wersja I**
  - Parametry zadania (dane wej.) zawarte są w klasie/obiekcie Task



Model Farmer-Worker – schemat

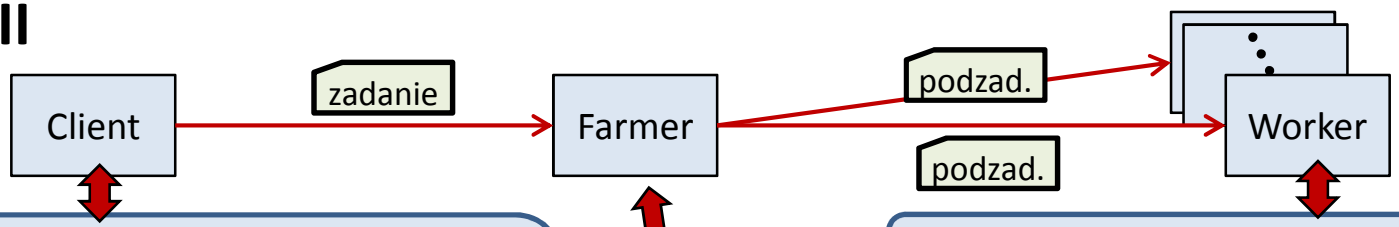
- **Wersja II**
  - Dodatkowy parametr (`params`) z parametrami zadania (danymi wej.)



- **Wersja III**
  - do Farmer'a w wywołaniu dodatkowo przekazywany jeszcze jeden obiekt potrafiący podzielić zadanie (może to być nietrywialne)

Model Farmer-Worker – schemat

- **Wersja II**




```
1 class MyTask implements Task {
    resultType compute(Object p) {
      ... some computations
      // muszą być odpowiednie rzutowania
    }
  }
  class MyClient {
    ...
    myComputations() {
2    Bind to Farmer  //lub w innym miejscu
3    MyTask myT = new MyTask();
4    MyParams myP = new MyParams();
5    result = Farmer.fCompute(myT, myP);
    }
  }
```

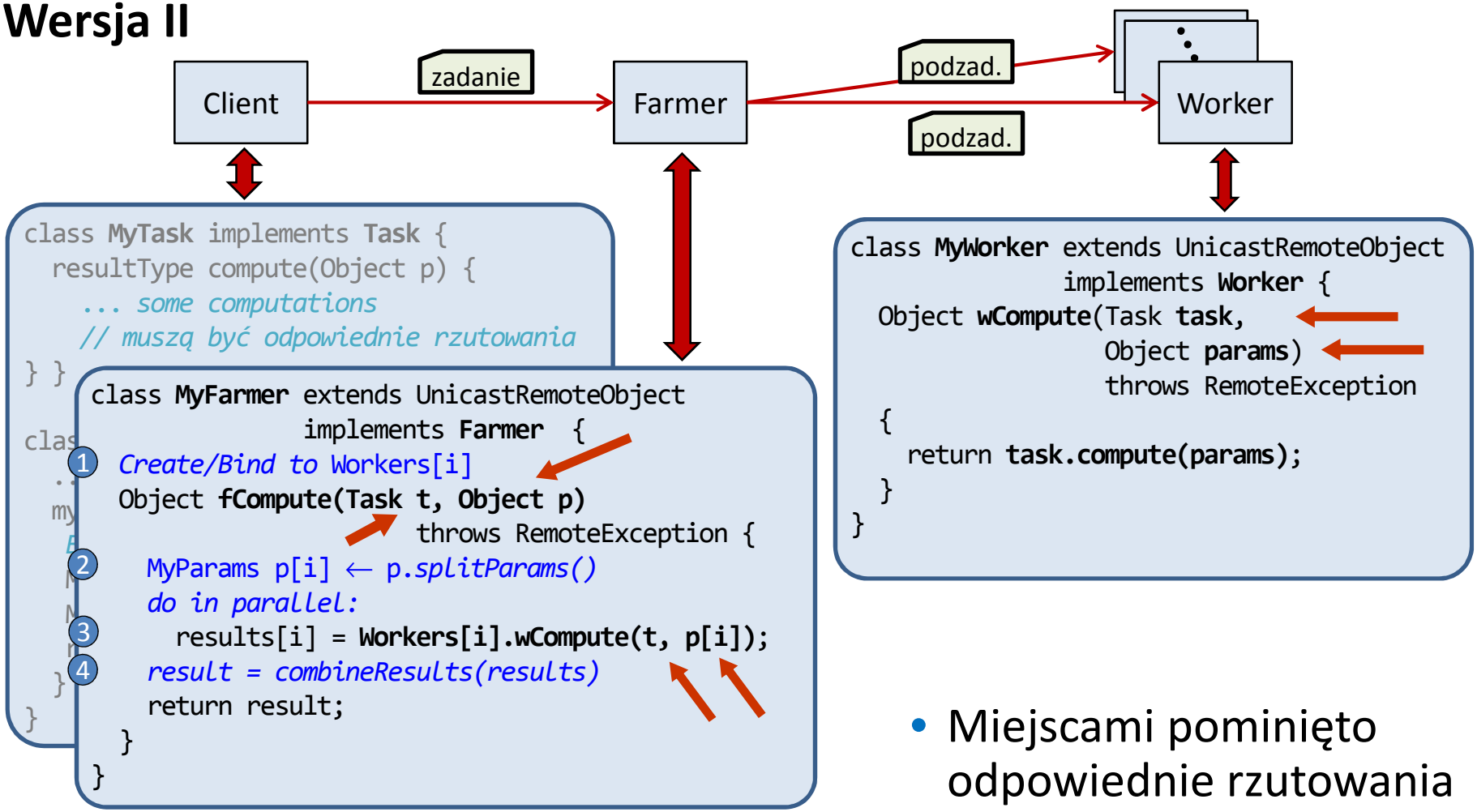
```
Interface Worker extends Remote {
 Object wCompute(Task task,
                 Object params) throws …
}
```

```
class MyFarmer extends UnicastRemoteObject
                implements Farmer  {
1  Create/Bind to Workers[i]
   Object fCompute(Task t, Object p)
                   throws RemoteException {
2    MyParams p[i] ← p.splitParams()
     do in parallel:
3      results[i] = Workers[i].wCompute(t, p[i]);
4    result = combineResults(results)
     return result;
} }
```

- **`resultType`** i **`MyParams`** odpowiednie do zadania (np. typu `Vector` lub własny typ)
- *na niebiesko* – działanie do wykonania, nie instrukcja języka

Model Farmer-Worker – schemat

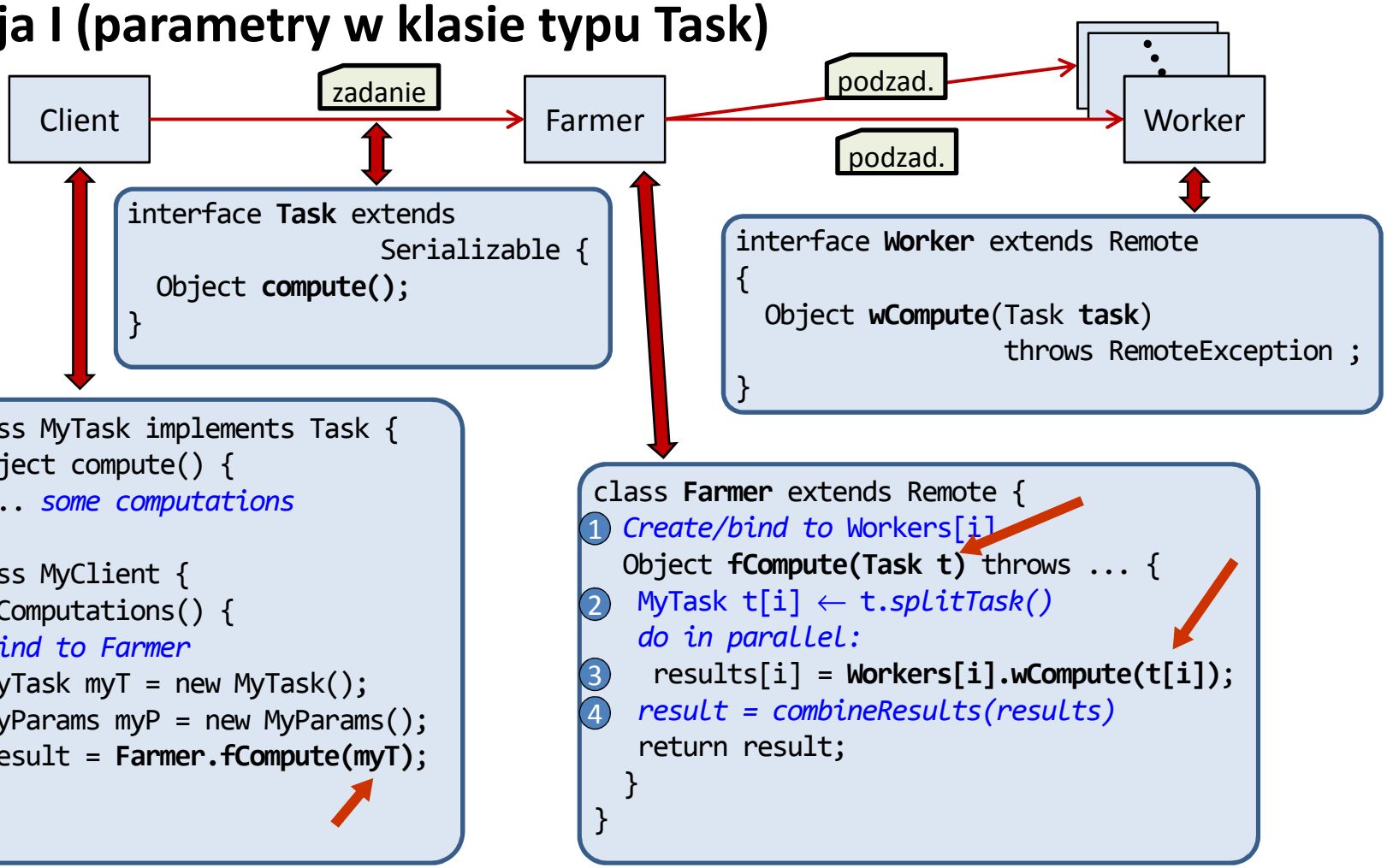
- **Wersja II**



- Miejscami pominięto odpowiednie rzutowania

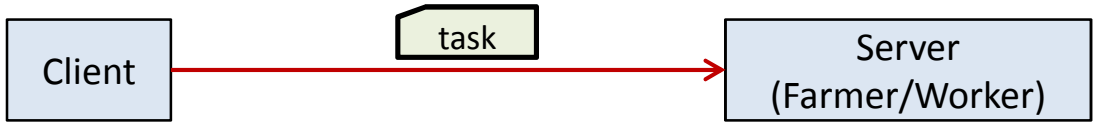
Model Farmer-Worker – schemat

- **Wersja I (parametry w klasie typu Task)**



Model Farmer-Worker – schemat

- **Warunki działania**



- Server (Worker i Farmer) **musi znać wymagane typy** t.j. klasę zadania przekazywanego przez klienta i klasę wyniku
  1. Dołączenie klas do serwera (na etapie kompilacji) ... lub ...
  2. Zdalne przesłanie bytecode'u klas z klienta do serwera
     - Serwer musi mieć dodatkowe metody do:
       - sprawdzenia czy ma wymaganą klasę
       - wgrania bytecode'u wymaganej klasy
     - Te metody mogą używać tylko standardowe typy. Pomocne są m.in:
       - `Object.getClass()`
       - `Class.getName()`
       - `Class.forName()`
     - Zdalne przesyłanie klas powinno być nadzorowane pod względem bezpieczeństwa

Model Farmer-Worker – schemat

- **Schemat dla problemu "widzialności" typów obiektów**
  - Serwer/Worker ma dwie dodatkowe metody
    - `hasClassCode(String name)` – sprawdza czy w procesie znana jest klasa o nazwie **name**
    - `storeClassCode(String name, byte[] t)` – wgrywa byte-code klasy podany w tablicy **t**
  - Klient najpierw używa tych metod dla sprawdzenia czy serwer ma daną klasę

```
interface Worker extends Remote {
  Object wCompute(Task task) throws ... ;
  boolean hasClassCode(String className) throws ... ;
  void storeClassCode(String className, byte[] t) throws ... ;
}
```

Client → **Server**

```
...
boolean isOk = myWorker.hasClassCode(className);
if (! isOK)
{ ...        // read bytecode from the file and store in byte table t
  myWorker.storeClassCode(className, t);
}
result = (resultType) myWorker.wCompute(myTask, myParams);
...
```

Model Farmer-Worker – schemat

- **Przykładowy kod rozszerzający Worker'a**

```
public class MyWorker extends UnicastRemoteObject implements Worker {
 ...
 public Object wCompute(Task task, Object params) throws RemoteException
 { ... }
 public boolean hasClassCode(String className) throws java.rmi.RemoteException {
  try {
   Class.forName(className);
   return true;
  } catch(Exception e) {
   return false;
  }
 }
```

To jak wcześniej

Te metody dodatkowo w interfejsie Worker'a

```
public void storeClassCode(String className, byte[] classByteCode) throws java.rmi.RemoteException {
  java.io.FileOutputStream out;
  try {
   out = new java.io.FileOutputStream(className + ".class");
   out.write(classByteCode);
   out.close();
  } catch (Exception e) {
   System.out.println("Blad zapisu kodu klasy: " + className);
   e.printStackTrace();
  }
 }
}
```

Model Farmer-Worker – schemat

- **Przykładowy kod rozszerzający klienta**

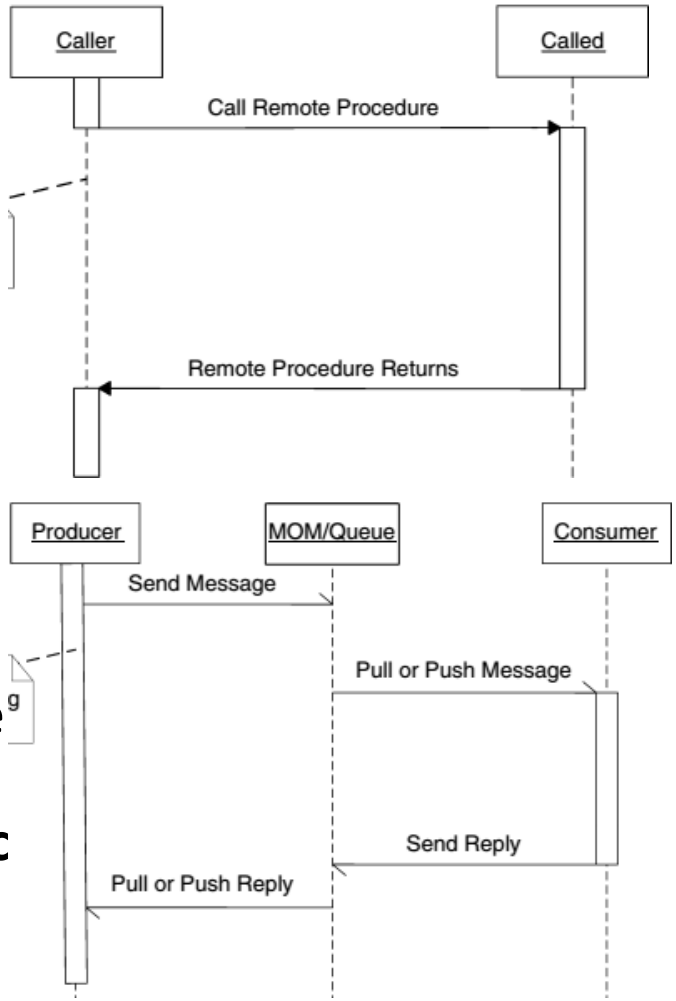
```
public class MyClient {
  ...
    myWorker = (Worker) java.rmi.Naming.lookup("//odpowiedni_adres");
  ...
  try {
    String className = ob.getClass().getName();          1
    boolean isOK = myWorker.hasClassCode(className);     2
    if (! isOK) {
      java.io.File f = new java.io.File(className + ".class");
      byte[] t = new byte[(int) f.length()];
      java.io.FileInputStream in = new java.io.FileInputStream(className + ".class");
      in.read(t);
      in.close();
      myWorker.storeClassCode(className, t);  3
    }
    wynik = (ResultType) myWorker.wCompute(myTask, myParams);
  }
  catch (Exception e) {
    e.printStackTrace();
    return;
  }
```

Tu MyTask powinien być już znany serwerowi

- **Message Oriented Middleware (MOM)**
  - Systemy kolejkowania wiadomości
    (*MQ Systems*)

Modele interakcji w komunikacji

- **Dwa ogólne modele interakcji**
  - ***RPC-based***
    - Komponenty komunikujące się są wiązane
    - Zwykle komunikacja synchroniczna
      - Klient czeka na rezultat wywołania procedury
      - Ale są również inne schematy
  - ***MQ-based***
    - Specjalny element pośredniczący
    - Komunikacja asynchroniczna
    - Wszystkie elementy systemu zachowują niezależność przetwarzania
    - Zachowanie kontroli przetwarzania przez obie wywołujący
    - Zazwyczaj: **Systemy kolejkowania wiadomośc (*Message Queuing Systems*)**



Systemy kolejkowania wiadomości

- **Systemy kolejkowania wiadomości (lub Systemy kolejkowe)**
  - Warstwa pośrednia dostarczająca usługi **komunikacji trwałej, zorientowanej na komunikaty**
- **MOM – ang. *Message Oriented Middleware***
  - Dostarcza usługi komunikacyjne wg asynchronicznego modelu interakcji
    - Bez blokowania
    - Umożliwia dostarczanie komunikatów gdy jedna strona jest nieaktywna
  - Zapewniania przechowywanie danych w trakcie przesyłania
    - Nadawca i odbiorca mogą działać całkowicie niezależnie
    - Wspieranie transferów komunikatów, które mogą trwać dłużej
  - Ogólnie nie ma gwarancji dostarczenia komunikatu ani gwarancji czasu dostarczenia
  - Podstawowy element systemu: **kolejka komunikatów**

Systemy kolejkowania wiadomości

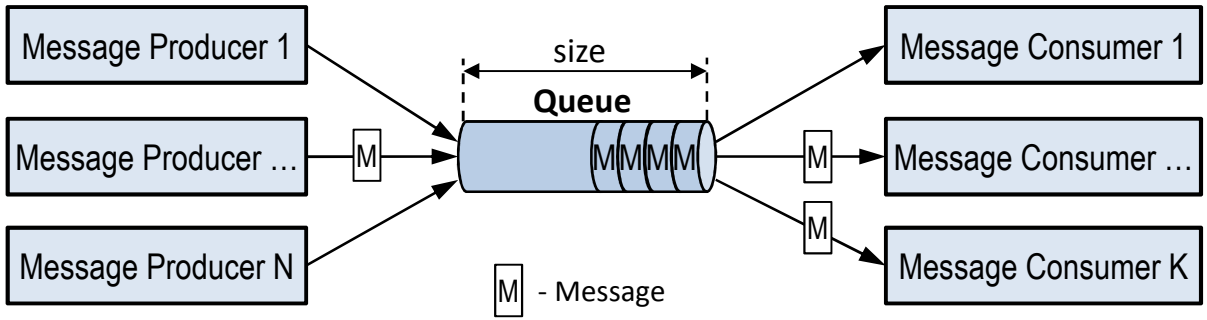
- **Zastosowanie**
  - Gdy proces wysyłający komunikaty nie może być wstrzymany
  - Gdy proces wysyłający komunikaty nie potrzebuje szybkiej informacji zwrotnej o tym czy komunikat dotarł do adresata
  - Gdy elementy składowe systemu nie są cały czas aktywne
  - Dla dużych systemów z masową wymianą wiadomości
    - Masywnie skalowalne systemy obsługują miliony żądań/godz.
  - Dla podsystemów mocno rozproszonych o zróżnicowanej architekturze

Systemy kolejkowania wiadomości

- **Wyróżniające cechy MOM**
  - Luźne powiązanie (*Loosely Coupling*)
    - Luźne powiązanie współdziałających elementów systemu
    - Możliwość łączenia aplikacji bez konieczności adaptacji
  - Niezawodność (*Reliability*)
    - Mechanizm *store and forward* zapobiegający gubieniu komunikatów mimo problemów działania elementów systemu
    - Często zapewniane dostarczanie wszystkim komunikatów ”dokładnie raz”
  - Dostępność (*Availability*)
    - Zapewniona wysoka responsywność i dostępność usług
    - Błąd lub chwilowa niedostępność podsystemów nie wpływa na dostępność całego systemu
  - Skalowalność (*Scalability*)
    - Na skutek luźnego powiązania niezależna wydajność podsystemów
    - Podsystemy skalowalne niezależnie (żaden lub mały wpływ na resztę)
    - Zawierają mechanizmy równoważenia obciążenia
      - Naturalne dzięki akceptacji komunikatów zamiast wymuszania odbioru

Systemy kolejkowania wiadomości

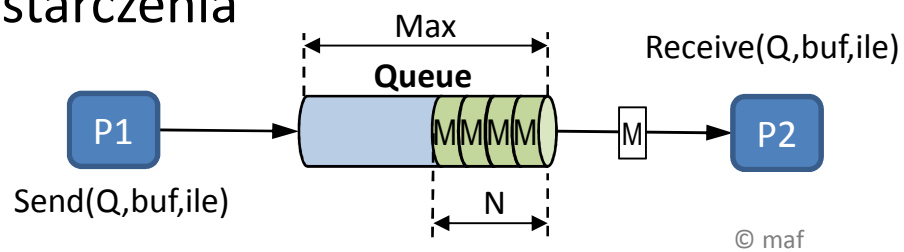
- **Kolejka komunikatów (ang. *Message Queue*)**
  - Podstawowy element koncepcji MOM
    - Pośredni obiekt komunikacyjny
    - Posiada nazwę którą procesy mogą zidentyfikować
  - Klienty MOM wysyłają/odbierają komunikaty do/z kolejek
  - Umożliwia zachowywanie komunikatów w systemie
  - Kolejka może być współdzielona
  - Zwykle systemy MOM mają wiele typów kolejek



© maf

Systemy kolejkowania wiadomości

- **Kolejka komunikatów (ang. *Message Queue*)**
  - Konfigurowalne parametry
    - Nazwa
    - Rozmiar
    - …wiele innych
    - Algorytm sortowania
      - › Standardowe rozwiązanie to FIFO
      - › Możliwość używania priorytetów
  - Operacje
    - Procesy mogą umieszczać komunikaty w kolejce (np. za pomocą operacji **`Send(Q,buf,ile)`**)
    - Procesy mogą pobierać komunikaty z kolejki (np. za pomocą operacji **`Receive(Q,buf,ile)`**)
    - Gwarancja dostarczenia do kolejki
    - Brak określenia momentu dostarczenia

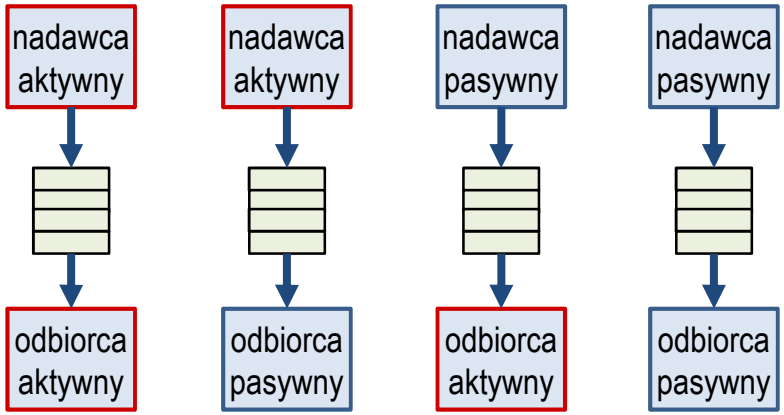


Liczba kom. w kolejce	Wysłanie komunikatu	Odbiór komunikatu
N = Max	blokada lub sygn. błędu	brak blokady
0 < N < Max	brak blokady	brak blokady
N = 0	brak blokady	blokada lub sygn. błędu

© maf

Systemy kolejkowania wiadomości

- **Tryby działania**



- **Operacje elementarne**

Operacja elementarna	Działanie
Put	Umieść komunikat w kolejce.
Get	Odbierz komunikat z kolejki. Gdy brak komunikatu następuje blokada.
Poll	Sprawdź czy w kolejce jest komunikat i pobierz go. Gdy nie ma komunikatu nie ma blokady.
Notify	Zawiadom gdy w kolejce pojawi się komunikat.

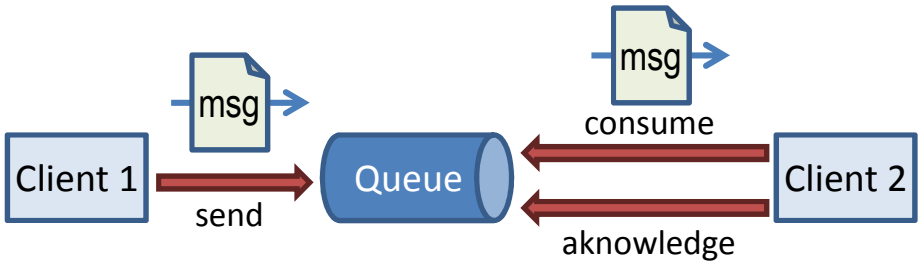
Systemy kolejkowania wiadomości

- **Typy kolejek**
  - Zwykle systemy MOM mają wiele typów kolejek
  - Przykłady:

Typ	Przeznaczenie
Public Queue	Publiczne z wolnym dostępem
Private Queue	Wymagające autoryzacji i autentykacji poprzez nazwę użytkownika i hasło
Temporary Queue	Tworzone na ograniczony czas. Istnieją przez jakiś odcinek czasu lub podczas trwania określonych warunków
Journal Queue	Do przechowywania rekordów wiadomości lub zdarzeń. Utrzymują kopie każdego komunikatu tworząc dziennik wiadomości (*journal of messages*)
Connector/Bridge Queue	Pozwala własnym implementacjom MOM współdziałać z zewnętrznymi MOM działając jak proxy. Realizuje translację formatów wiadomości dając dostęp do kolejek innych dostawców.
Dead-Letter / Dead-Message Queue	Dla wiadomości przeterminowanych lub niedostarczonych (np. z powodu użycia złego adresowania).

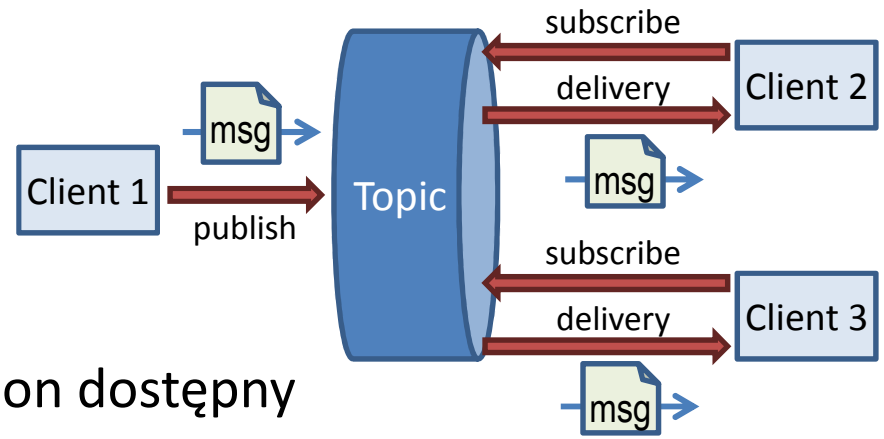
Systemy kolejkowania wiadomości

- **Dwa podstawowe modele kolejkowe**
  - Model **Point-to-Point**
  - Model **Publish/Subscribe**
- **Model Point-to-Point**
  - Bezpośrednia asynchroniczna wymiana komunikatów pomiędzy partnerami poprzez kolejkę (zwykle w trybie FIFO)
  - Bez ograniczeń na liczbę podłączonych jednostek do kolejki
  - Zwykle jeden odbiorca komunikatów
  - Wiadomość jest zawsze dostarczana i <u>tylko do jednego odbiorcy</u>
    - Wiadomość jest przechowywana aż odbiorca będzie gotowy ją odebrać
  - Schemat dla modelu przetwarzania typu żądanie-odpowiedź
    - Np. jak w usłudze WWW
  - Partnerzy komunikacji znają reguły dialogu



Systemy kolejkowania wiadomości

- **Model Publish/Subscribe**
  - Mechanizm dystrybucji komunikatów **jeden-do-wielu** lub **wiele-do-wielu**
  - Bez konieczności znajomości aplikacji odbierających
    - Wysyłka do tzw. **kanałów** (***channels*** lub ***topics***)
  - Subskrybcja do kanału, z którego można otrzymywać komunikaty
    - Usługa wysyła komunikaty do konsumentów wg subskrybcji
    - Każdy konsument subskrybowany do danego kanału otrzymuje przesłany do tego kanału komunikat
  - Dwie metody odebrania komunikatu
    - **Pull** – sondowanie kanału i "wyciąganie" komunikatu
    - **Push** – zgłaszanie jednostce wysłania komunikatu (wypchnięcia) jak tylko będzie on dostępny



Systemy kolejkowania wiadomości

- **Typowe usługi MOM**
  - Równoważenie obciążenia
    - Różne podejścia
  - Filtrowanie
    - ***Channel based*** – poprzez subskrybcje do grup kanałów
    - ***Subject based*** – wiadomości dodatkowo znakowane tematem
    - ***Content based*** – użycie języków zapytań (typu SQL) do filtracji
    - ***Content based with paterns*** (***Composite Events***) – reguły obejmujące wiele wiadomości
  - Transakcje
  - Transformacje komunikatów
  - Gwarancje operacji komunikacyjnych

Systemy kolejkowania wiadomości

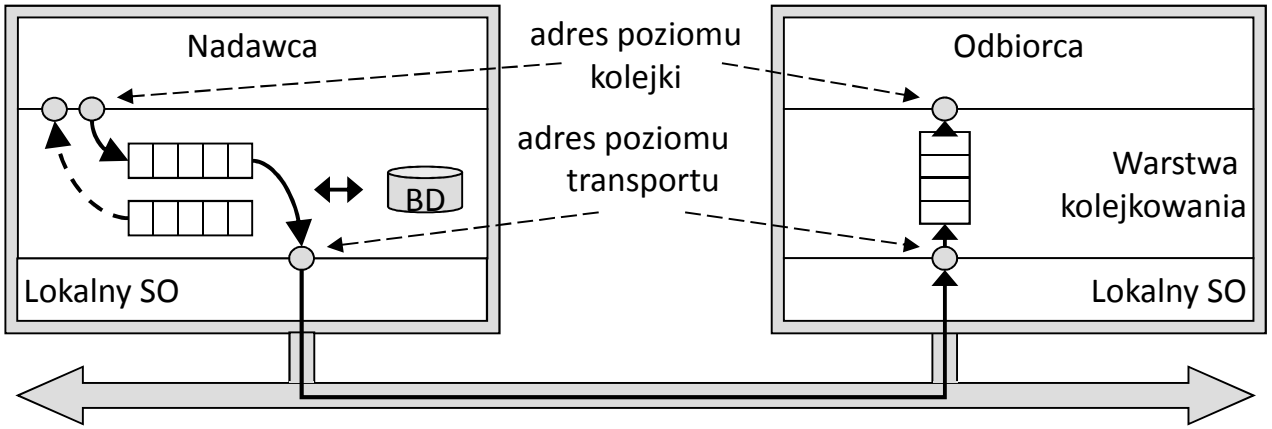
- **Typowe usługi MOM**
  - Transakcje
    - Grupowanie komunikatów (zadań) w transakcję do wykonania
    - Określone cechy: wykonanie wszystkiego lub nic, ACID, itp.
    - Zatwierdzanie:
      - Przez producenta: MOM utrwala wiadomości w pamięci trwałej
      - Przez konsumenta: MOM usuwa wiadomości z pamięci trwałej
    - Rollback:
      - Producenta: broker usuwa wszystkie komunikaty transakcji
      - Konsumenta: broker przesyła wszystko jeszcze raz

Systemy kolejkowania wiadomości

- **Typowe usługi MOM**
  - Gwarancja dostarczania
    - Potwierdzanie odbioru
      - Powtórne wysyłanie po przekroczeniu czasu przeterminowania
    - Rozszerzenie: certyfikowane dostarczanie
      - Wysyłanie potwierdzenia konsumpcji dla producenta wiadomości
  - Transformacja formatów wiadomości
    - Typowe formaty:
      - Text (włączając XML)
      - Object
      - HashMaps
      - Stream
      - Streaming Multimedia
    - Np. transformacje XSL dla dokumentów XML

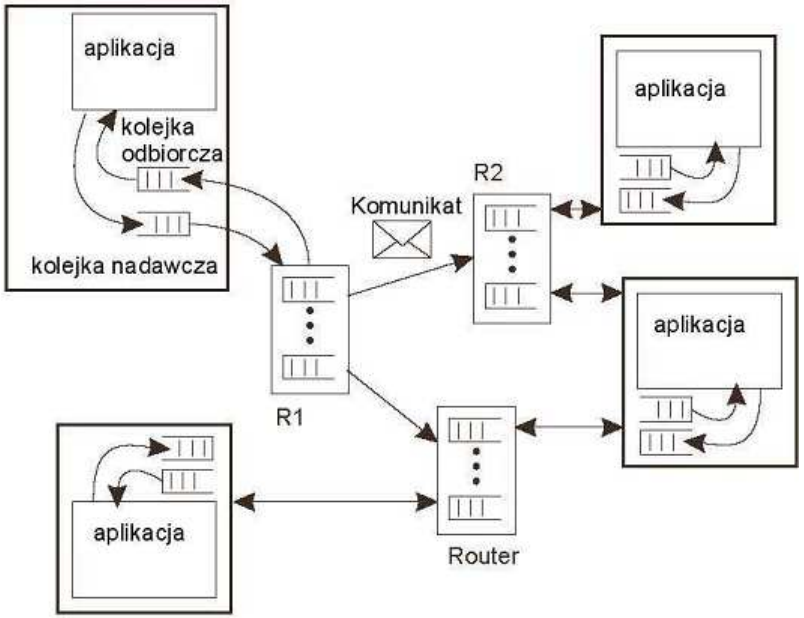
Systemy kolejkowania wiadomości

- **Ogólna architektura**
  - Kolejki **źródłowe** (lokalne) i **docelowe**
  - Kolejki **nadawcze** i **odbiorcze**
  - Adresowanie
    - Ogólnosystemowy mechanizm nazewnictwa kolejek
    - Mechanizm translacji nazw kolejek w adresy sieciowe
      - Konieczność utrzymywania BD nazw kolejek (system nazewniczy)
  - Zarządcy kolejek
    - Administrują kolejkami



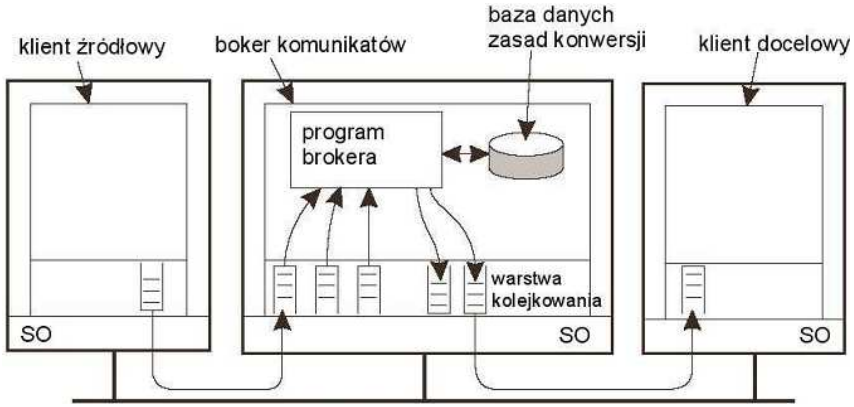
Systemy kolejkowania wiadomości

- **Ogólna architektura**
  - Zarządcy kolejek
    - Zazwyczaj bezpośrednia współpraca z aplikacją
    - Także rozbudowane funkcje jako przekaźniki lub routery
  - Gdy brak usług nazewniczych, statyczna topologia sieci
    - Każdy zarządca ma tablicę odwzorowań kolejek
  - Dynamiczne schematy routingu (jak w sieciach komputerowych)
    - Zarządca musi znać tylko najbliższy router
    - Dodatkowe funkcje jak np. *multicasting*
  - Sieć nakładkowa
    - Skalowalne systemy kolejkowania
    - Rozbudowane sterowanie komunikacją



Systemy kolejkowania wiadomości

- **Broker komunikatów (ang. Message Broker)**
  - Specjalny węzeł sieci kolejkującej
    - Usługi połączeniowe (fizyczne połączenia dla transportu wiad.)
    - Usługi dostarczania wiadomości
    - Usługi utrwalania
    - Usługi zabezpieczeń
    - Usługi monitorowania
    - Usługi konwersji wiadomości
    - …
  - Scalanie aplikacji w jeden spójny system rozproszony
    - Konwersja różnych formatów komunikatów
    - Prosty konwerter danych lub brama na poziomie aplikacji w systemie kolejkowania
      - Baza reguł konwersji komunikatów
      - Czasami możliwość utraty informacji



Systemy kolejkowania wiadomości

- **JMS - Java Message Service**
  - Standard dostarczający uniwersalny interfejs dla interakcji MOM
    - JMS jest specyfikacją API i nie określa implementacji usług kolejkowania
    - API pozwala pisać aplikacje do współdziałania z MOM
  - Pozwala tworzyć infrastrukturę kolejkowania wiadomości
    - Dialog dwóch implementacji MOM kompatybilnych z JMS wymaga pośrednika jak np. brokera
- **Websphere MQ (MQSeries)**
  - System MOM na wiele platform (Linux, Windows, …)
  - API dla różnych języków (C, Java, …)
    - Również JMS dla Javy
- **Apache ActiveMQ**
  - ”the most popular and powerful open source messaging and Integration Patterns server”
  - Broker wiadomości z klientem JMS
- **Oracle AQ**
  - MOM integrujący się z Oracle DB

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

- **Service Oriented Architecture (SOA)**
  - Geneza i idea
  - Definicje i elementy systemów SOA
  - Wybrane wzorce realizacji

SOA

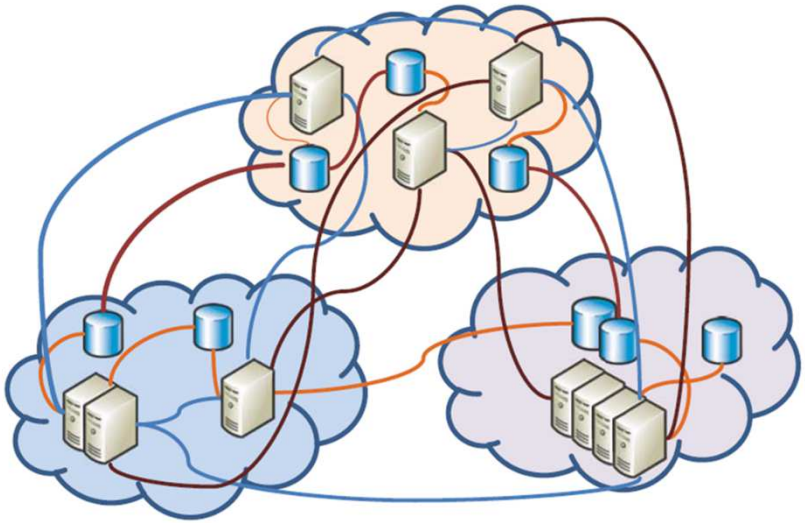
- **SOA – *Service Oriented Architecture***

> *"70% of the average IT department budget is devoted to data integration projects."*
> – IDC
>
> *"A typical enterprise will devote 35%–40% of its programming budget to programs whose purpose is solely to transfer information between different databases and legacy systems."*
> – Gartner Group

- Paradygmat otwartych, elastycznych systemów ułatwiających integrację aplikacji w środowisku rozproszonym
  - Pomimo realizacji systemów informatycznych za pomocą różnych technik i technologii
  - Brak odniesień do konkretnych technologii realizacji systemów

Geneza SOA

- **Zależności powstałe po latach rozwoju systemu z nieprzemyślaną architekturą**
  - Odrębne systemy
    - Potrzeba wymiany danych pomiędzy nimi
    - Potrzeba integracji z własnymi rozwiązaniami
  - Łączenie systemów często poprzez:
    - Nieustandaryzowane połączenia
    - lub bezpośredni dostęp jednego systemu do BD innego systemu
  - Po latach systemy są
    - Bardzo mocno powiązane ze sobą ze sobą
    - Muszą znać swoje szczegóły implementacyjne

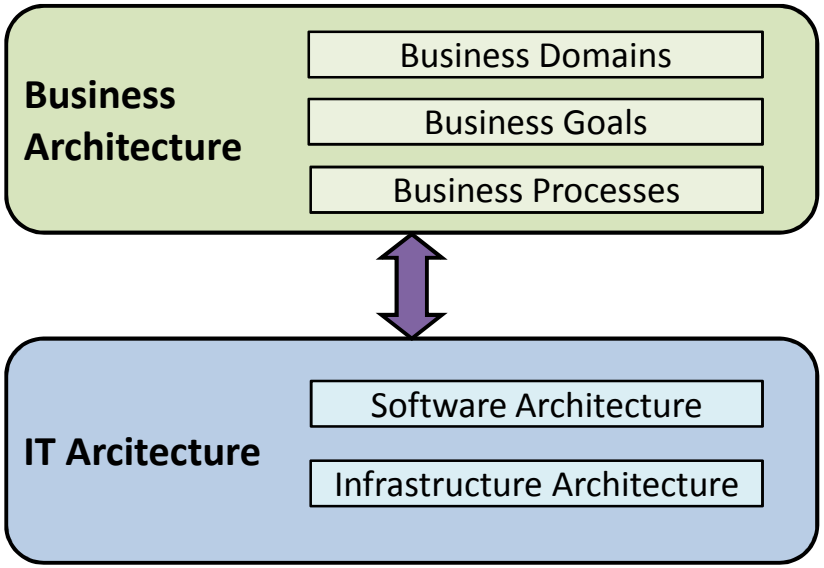


Geneza SOA

- **Konsekwencje systemu pełnego zależności**
  - Nie można:
    - Swobodnie zmieniać implementacje komponentów
      - Drobna zmiana w implementacji jednego komponentu, może sprawić, że przestanie działać inny komponent
    - Ponownie używać istniejących komponentów w innych systemach
  - Trudno jest:
    - Utrzymywać poszczególne komponenty
      - Członkowie obsługi muszą mieć dużą wiedzę o całym systemie
      - Komponenty mają wiele zależności więc nie można ich samodzielnie testować
    - Dołączać nowe komponenty
      - Gdy komponent programowy ma wiele zależności, trudno jest podłączyć go do innego systemu
    - Wymieniać komponenty na inne
- **SOA rozwiązuje przedstawiony problem, bo narzuca ograniczenia na wzajemną interakcję komponentów**

Geneza SOA

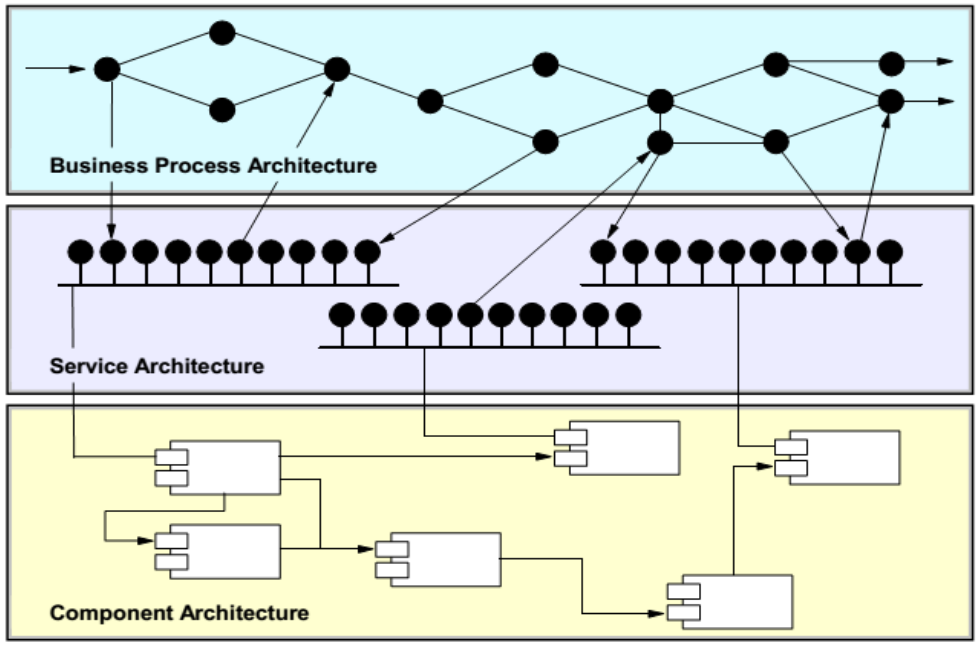
- **Procesy biznesowe w przedsiębiorstwie**
  - Organizacja posiada swoją architekturę biznesową, na którą składają się obszary, cele i procesy, a dla niej architekturę informatyczną
- **Opis procesów biznesowych**
  - Business Process Model and Notation (BPMN)
    - Graficzna notacja służąca do opisywania procesów biznesowych
  - Business Process Execution Language (BPEL)
    - Oparty na XML język do definiowania wykonywalnych procesów biznesowych



- **Celem jest dostosowanie architektury informatycznej dla wsparcia procesów biznesowych**

Definicje i charakterystyka

- **Proces biznesowy jako**
  - Zestaw procesów i przepływów
  - Realizowanych poprzez operacje dostarczane przez usługi
  - Implementowane przez komponenty programowe infrastruktury wykonawczej



Definicje i charakterystyka

- **SOA = Service Oriented Architecture = Architektura zorientowana na usługi**
- **Wikipedia**
  - *Koncepcja tworzenia systemów informatycznych, w której główny nacisk stawia się na **definiowanie usług**, które spełnią wymagania użytkownika. Pojęcie SOA obejmuje zestaw metod organizacyjnych i technicznych mający na celu lepsze powiązanie biznesowej strony organizacji z jej zasobami informatycznymi.*
- **Definicja wg IBM:**
  - *A service oriented architecture (SOA) is an architecture for building business applications as a set of **loosely coupled blackbox components** orchestrated to deliver a **well-defined level of service** by linking together business processes.*
- **Definicja wg Thomasa Erla**
  - *Service-oriented architecture is a term that represents a model in which **automation logic is decomposed into smaller, distinct units of logic**. Collectively, these units comprise a larger piece of business automation logic. Individually, these **units can be distributed**.*

Definicje i charakterystyka

- **SOA jest stylem architektonicznym tworzenia systemów rozproszonych opartych na interakcjach luźno powiązanych gruboziarnistych i autonomicznych komponentów nazywanych usługami.**
- **SOA to koncepcja budowania rozproszonych systemów, które dostarczają funkcjonalność poprzez usługi, zarówno dla użytkownika końcowego jak i do innych usług.**

- Obszar logiki reprezentowany przez usługi może być różny
  - Pojedyncza konkretna usługa może być mała, średnia lub duża
  - Może być definiowana na różnym poziomie, szczegółowości w zależności od charakterystyki procesu biznesowego

Definicje i charakterystyka

- **SOA sama w sobie nie jest żadną, konkretną architekturą. Jest tylko paradygmatem ("filozofią") prowadzącym do konkretnej architektury.**
  - Dalej: Architektura SOA = architektura zorientowana na usługi
- **SOA sama w sobie nic nie mówi o technologiach, protokołach itp., które mogą być jedynie środkiem osiągnięcia SOA**

Architektura SOA

- **Podstawowe zasady SOA dla usług**
  - **Autonomiczność**
    - Założenie istnienia indywidualnych jednostek logicznych, działających autonomicznie
      - ale nieodizolowanych od siebie
    - Usługa posiada pełną kontrolę nad obszarem logiki, który enkapsuluje
  - **Luźne powiązanie**
    - Powiązanie między usługami ograniczone do minimum
    - Współpraca poprzez przesyłanie wiadomości (komunikatów)
  - **Abstrakcja**
    - Wymusza na usłudze ukrycie enkapsulowanego obszaru logiki przed światem zewnętrznym
    - Interfejsy definiowane niezależnie od platformy programistycznej
    - Umożliwia: publikacja, odnajdowanie, wywoływanie

Architektura SOA

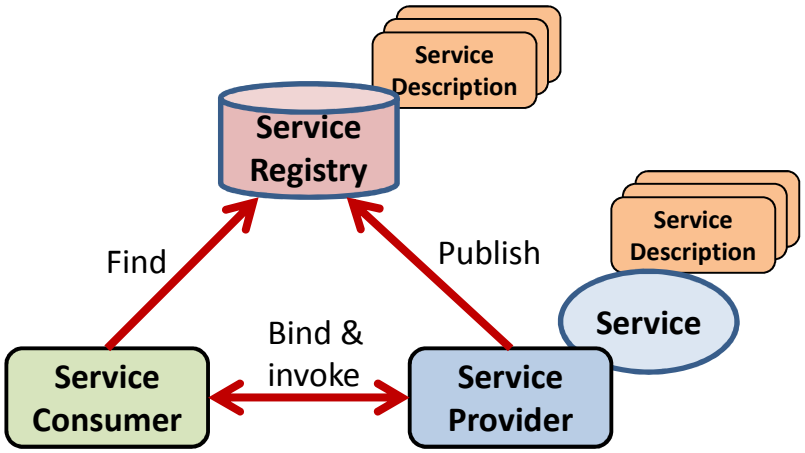
- **Podstawowe zasady SOA dla usług**
  - **Uniwersalna użyteczność / Reużywalność**
    - Rozbicie logiki na pojedyncze usługi traktowane jako komponenty, aby móc je wielokrotnie wykorzystać
  - **Komponentowość**
    - Umożliwienie łączenia różnych usług tak, aby na ich bazie powstawały inne
      - Usługi atomowe (*atomic* / *basic*)
      - Usługi złożone (*component*)
  - **Bezstanowość**
    - Usługi minimalizują przetrzymywanie danych zależnych od określonych akcji
      - np. usługa po wysłaniu wiadomości, traci nad nią całkowitą kontrolę
  - **Odkrywalność**
    - Dostępny opis usług
    - Metadane

Architektura SOA

- **Podmioty w SOA**
  - Konsument (Service Consumer)
    - Aplikacja, moduł programowy lub inna usługa, która wywołuje usługę
    - Inicjuje proces wyszukiwania usługi w rejestrze, tworzy powiązanie z usługą przez usługi transportowe a następnie ją wywołuje
    - Konsument korzysta z usługi na zasadach określonych przez udostępniony interfejs.
  - Dostawca usługi (Service Provider)
    - Jednostka która implementuje specyfikację usługi
    - Element dostępny poprzez sieć teleinformatyczną, który akceptuje i realizuje żądania konsumentów
    - Publikuje usługi w rejestrze oraz definiuje ich interfejs
  - Rejestr usług (Service Registry)
    - Repozytorium opisów usług i danych.
    - Dostawca usługi publikuję usługę poprzez zamieszczenie odpowiedniego opisu w repozytorium,

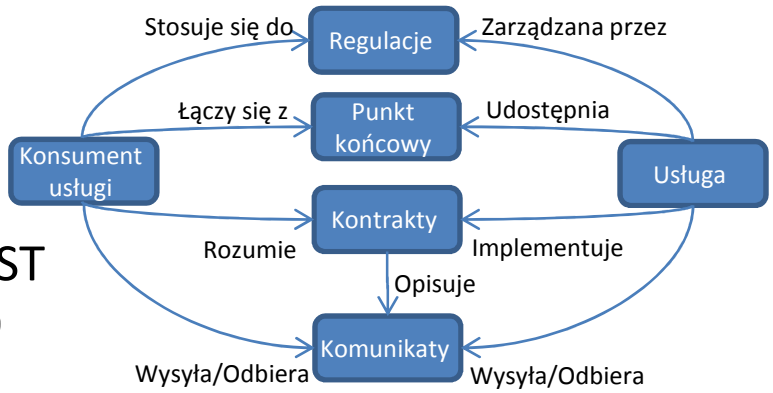
Architektura SOA

- **Współpraca w SOA**
  - Publikacja usług
    - Opis usług
      - Specyfikacja sposobu interakcji konsumenta z usługą
      - Specyfikacja formatu żądań i odpowiedzi
      - Opcjonalna specyfikacja warunków realizacji i QoS
  - Wyszukiwanie usług
    - Pozwala zlokalizować i uzyskać pewne informacje o wymaganej usłudze
  - Wiązanie i wywoływanie
    - Konsument wywołuje usługę zgodnie z informacjami zawartymi w opisie usługi



Definicje i elementy SOA

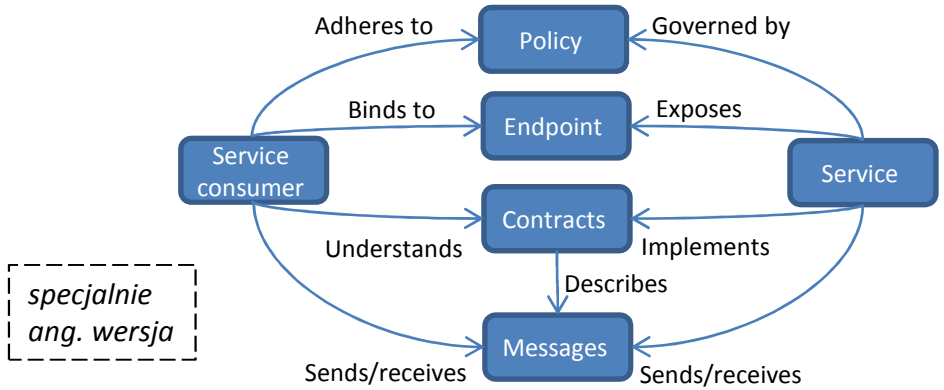
- **Komponenty definiowane przez SOA**
  - Usługa
    - Gruboziarnisty element logiczny odpowiadający funkcji biznesowej
    - Musi implementować wszystkie funkcje oferowane przez jej kontrakty
  - Kontrakt
    - Zbiór wszystkich komunikatów obsługiwanych przez usługę jest określany jako kontrakt
    - Jest analogiczny do interfejsu w programowaniu obiektowym
  - Komunikat
    - Jednostka komunikacji w SOA. Najczęściej:
      - SOAP (Simple Object Access Protocol)
      - Komunikaty protokołu HTTP (GET, POST, DELETE, PUT) w tzw. stylu REST
      - Komunikaty JMS (Java Message Services)



Definicje i elementy SOA

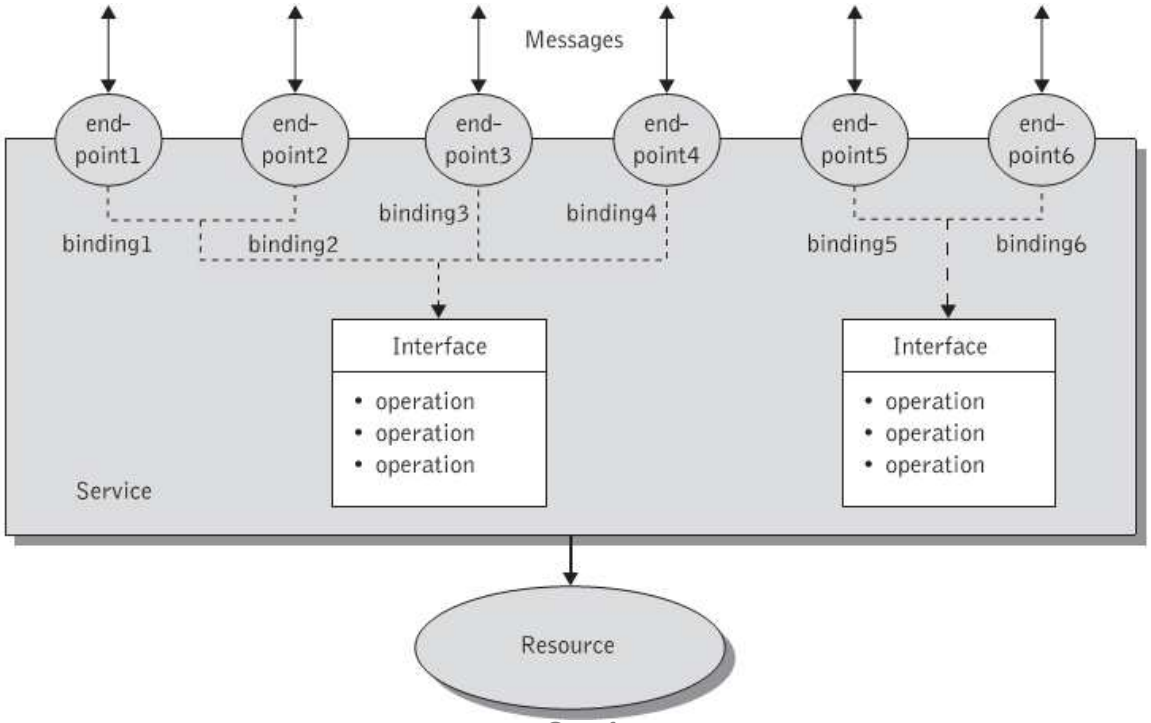
- **Komponenty definiowane przez SOA**
  - Punkt końcowy
    - Uniwersalny identyfikator określający miejsce dostępności usługi
    - Implementacja kontraktu jest dostępna w określonym punkcie końcowym, których usługa może posiadać większą liczbę
  - Regulacje
    - Definiują zasady i warunki tworzenia usług dla konsumentów
    - Określają właściwości dynamiczne takie jak szyfrowanie, uwierzytelnianie, autoryzacja, SLA

- **SOA definiuje także określone interakcje zachodzące pomiędzy tymi komponentami**



Architektura SOA

- **Dowiązanie – *binding***
  - Klient musi znać nie tylko interfejs, ale również jak komunikować się z usługą
    - Transport
    - Kodowanie
    - Protokół

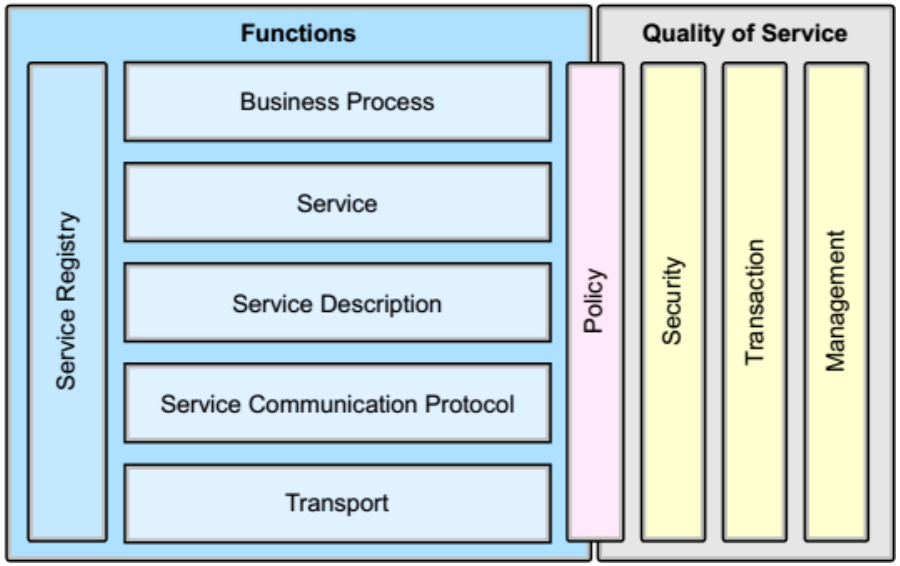


Architektura SOA

- **Stos architektoniczny**

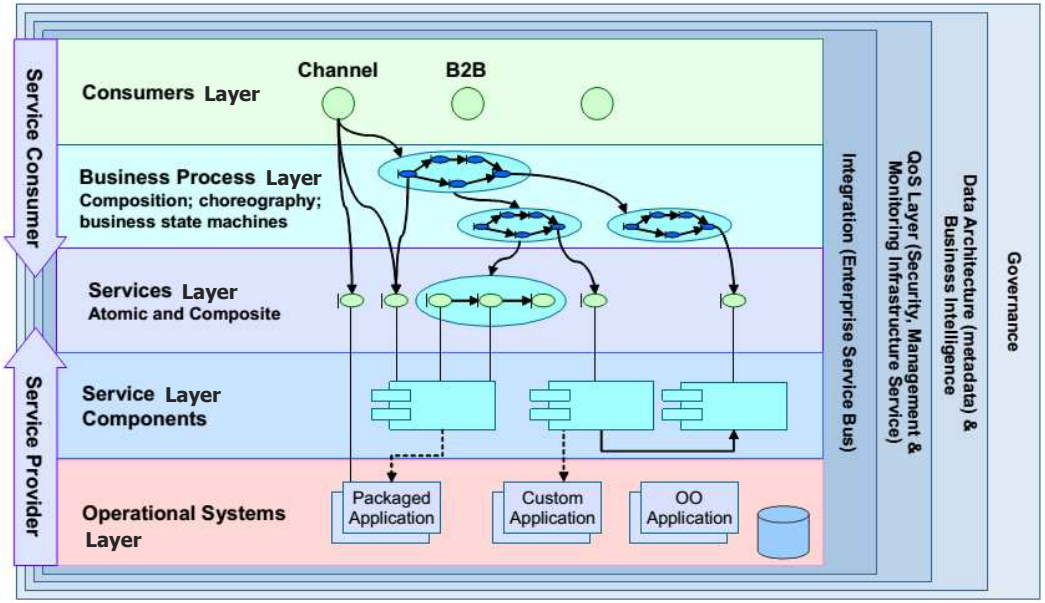
  Uwaga: niektóre podejścia przedstawiają inne obrazy j.w.

  - 2 aspekty składowych architektury:
    - Funkcjonalne
      - Mechanizm przekazywania wiadomości (danych)
      - Mechanizm komunikacji usługi i konsumenta usługi
      - Uzgodniony schemat opisu usługi
      - Opis dostępnych usług
      - Proces biznesowy jako kolekcja usług wywoływanych w określony sposób
      - Repozytorium informacji o usługach
    - Jakościowe
      - Różne obszary funkcjonalne i niefunkcjonalne



Architektura SOA

- **Architektura referencyjna środowiska wykonawczego**
  - Dedykowane i niestandardowe aplikacje
  - Komponenty programowe odpowiedzialne za realizację funkcjonalności i jakości oferowanych usług
  - Usługi definiowane w SOA (odkrywalne, dostępne, atomowe, złożone)
  - Procesy będące „złożeniem” (orkiestracja i choreografia) odpowiednich usług i innych procesów
  - Warstwa interfej-su konsumenta usługi

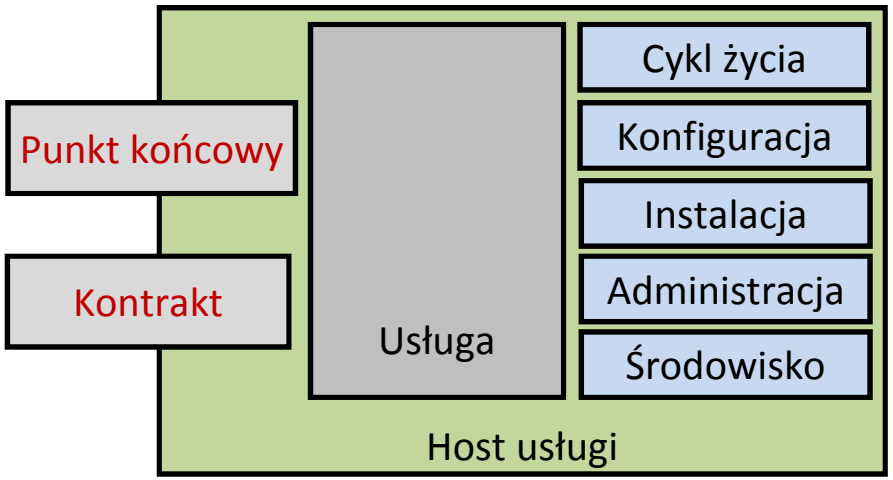


Wzorce realizacji SOA

- **Wzorce Projektowe (ang. Design Patterns)**
  - W architekturze oprogramowania sprawdzone rozwiązania często pojawiających się, powtarzalnych problemów projektowych
    - Wykorzystanie gotowych wzorców pozwala znacząco skrócić czas oraz zredukować koszty wdrożenia.
    - Jednocześnie minimalizuje się ryzyko popełnienia błędów w tworzeniu efektywnego rozwiązań i nie zrealizowania założonych celów
- **Wzorce wykorzystywane w SOA**
  - Jest duża liczba wzorców wykorzystywanych dla realizacji systemów SOA
  - Odnoszą się m.in. do następujących atrybutów jakościowych:
    - dostępność, elastyczność, integralność, wydajność, przenośność, niezawodność, skalowalność, bezpieczeństwo
  - **5 głównych grup wzorców SOA**
    - **strukturalne**
    - **wymiany komunikatów**
    - **wydajności, skalowalności i dostępności**
    - **bezpieczeństwa i zarządzalności**
    - **integracji usług**

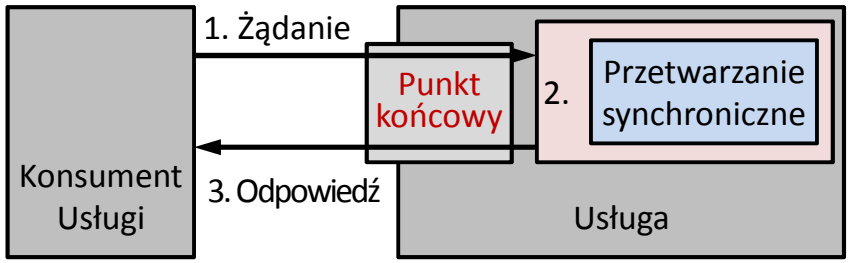
Wzorce realizacji SOA

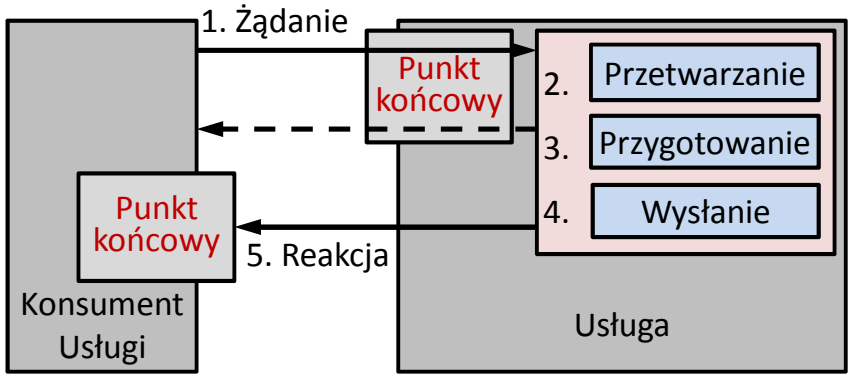
- **Service Host (Host usługi)**
  - Najbardziej podstawowy wzorzec strukturalny
    - Dotyczy środowiska, w którym uruchamiane są instancje danej usługi
    - Działa jak kontener dla usługi
      - powinien być konfigurowalny i przeprowadzać instalacje usługi oraz jej konfiguracje.
  - Wykonuje takie funkcje jak:
    - Zarządzanie cyklem życia
      - instancjonowanie usługi, czy też jej recykling w przypadku błędu.
    - Konfiguracja
      - powinna być odczytywana i stosowana do zabezpieczeń, polityki kontraktów itp.
    - Instalacja
      - konfiguracja wykonawcza w zakresie wiązania komponentów usługi
    - Administracja
      - kontrolowanie cyklu życia usługi i monitorowanie
    - Środowisko
      - Usługi takie jak pamięć podręczna, połączenie z bazą danych, biblioteki itp.
  - Charakterystyka
    - Program pośredniczący, który hostuje instancje usługi i wprowadza infrastrukturę możliwą do wykorzystania
      - Serwery aplikacji, lekkie kontenery, itp.



Wzorce realizacji SOA

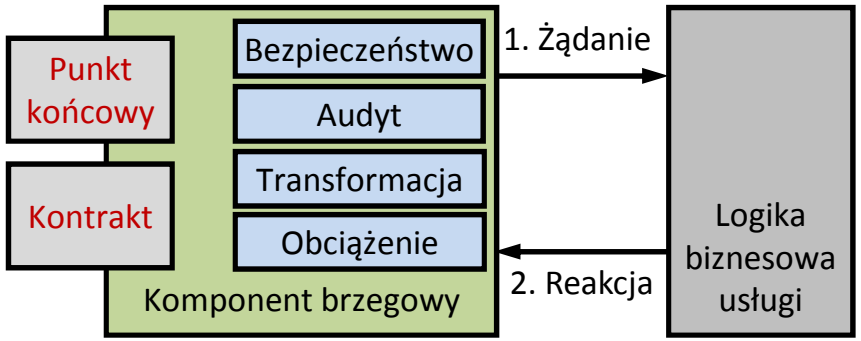
- **Request-response (Żądanie-odpowiedź)**
  - Komunikacja synchroniczna
- **Request-reaction (Żądanie-reakcja)**
  - Komunikacja asynchroniczna
  - Dwa oddzielne komunikaty
    - Inicjacja komunikacji od usługi do klienta → usługa musi posiadać informację gdzie zwrócić odpowiedź
    - Czasowe oddzielenie → potrzebny mechanizm identyfikacji powiązania dwóch komunikatów
      - najczęściej jest to identyfikator korelacji dołączany do żądania i odpowiedzi
    - Alternat. technologia **push**
      - Konsument otrzymuje w odp. adres reakcji (np. URI) i czas ETA
      - Po ETA wysyła żądanie o reakcję





Wzorce realizacji SOA

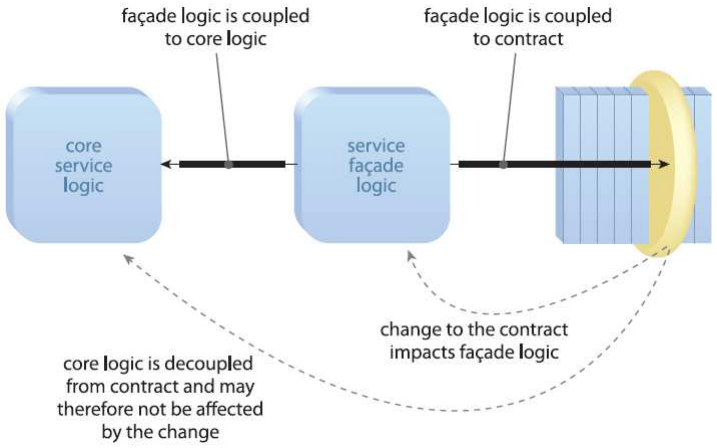
- **Service Façade (Fasada)**
  - Kolejny ważny wzorzec strukturalny

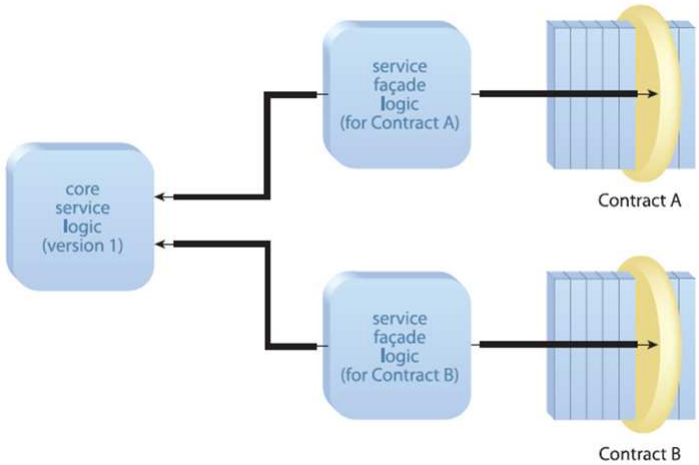


  - Dodaje nad poziom logiki biznesowej poziom separacji
    - rozdziela interfejs od implementacji aby zapewnić elastyczność i łatwość utrzymania
  - W komponencie brzegowym obsługiwane są funkcje nie będące częścią logiki biznesowej, takie jak audyt, różne punkty końcowe, filtry itp.
  - Wzorzec ten zapewnia fasadę dla implementacji usługi
  - Inna nazwa - **komponent brzegowy**

Wzorce realizacji SOA

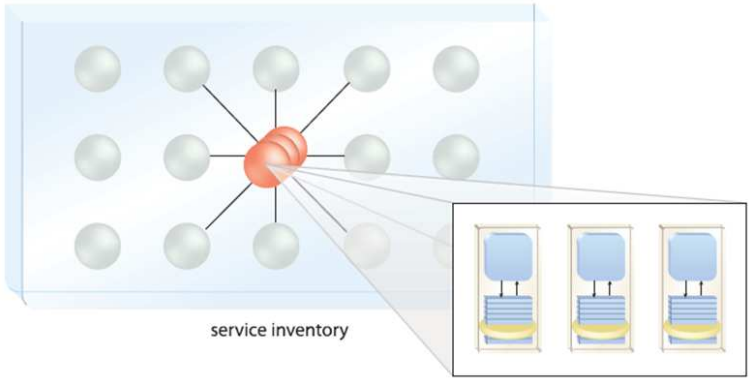
- **Service Façade (Fasada)**
  - Problem
    - Silne bezpośrednie powiązanie usługi (kontraktu) z logiką (implementacją)
    - Zmiana logiki działania (przetwarzania) ⟶ zmiana interakcji klienta i usługi, (np. zmiana formatu danych wymusza zmiany w obszarze logiki)
  - Cel
    - Zmniejszenie powiązania między kontraktem usługi, a jej logiką
  - Rozwiązanie
    - Usługa fasadowa – dodatkowa warstwa pośrednicząca realizująca część logiki biznesowej
      - System może być zastąpiony innym bez zmiany w logice usługi
    - Możliwe zmiany tylko w pośredniczącej warstwie logiki
      - Nienaruszalność samej logiki w monolicie





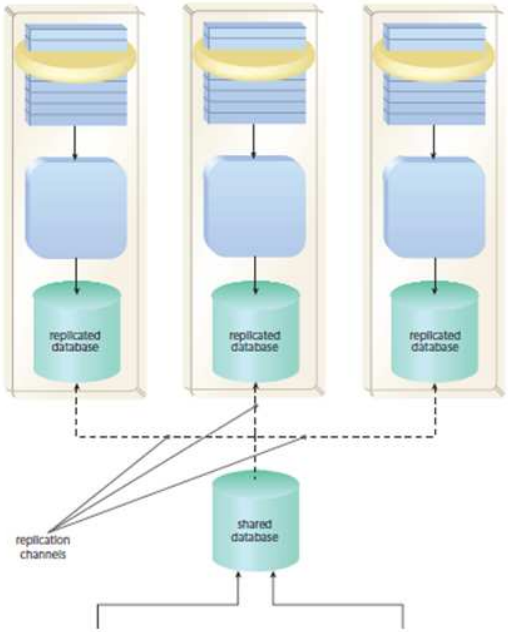
Wzorce realizacji SOA

- **Redundant implementation (Redundantna implementacja)**
  - Problem
    - Awaryjność pojedynczych usług, ważnych z globalnego punktu widzenia
  - Cel
    - Gwarancja większej stabilności i dostępności usług
  - Rozwiązanie
    - Zwielokrotniona liczba implementacji danej usługi
    - **Dwa schematy**
      - **Podział implementacji pomiędzy różne grupy użytkowników**
        - pomimo awarii zasobu usługi dla jednej grupy jest gwarancja ciągłości dla pozostałych grup
      - **Zachowanie innych implementacji jako opcji awaryjnych** w razie uszkodzenia usługi
        - w danym momencie tylko jedna wersja implementacji jest aktywna



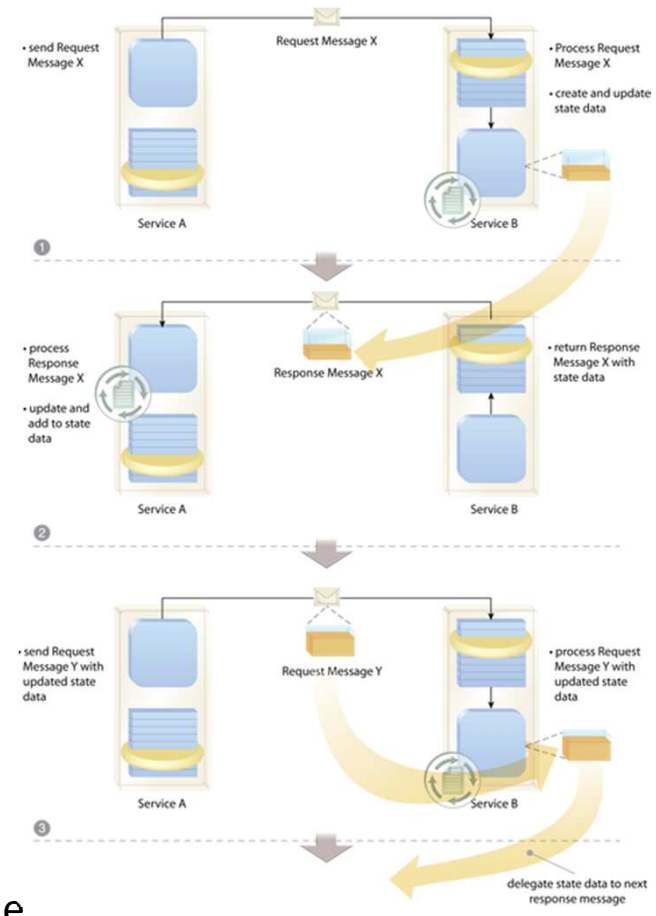
Wzorce realizacji SOA

- **Service data replication (replikacja danych usług)**
  - Problem
    - Utrata autonomiczności usług ze względu na korzystanie z jednej bazy danych
    - W przypadku awarii bazy danych awarii ulega cały system usług
  - Cel
    - Zwiększenie autonomiczności i dostępności usług
  - Rozwiązanie
    - Repliki bazy danych dla każdej z usług
    - Aktualizacja danych odbywa się kanałem replikacji, dzięki czemu każda usługa ma dostęp do zawsze aktualnej bazy danych
      - Należy dobrze zaprojektować mechanizmy replikacji, aby zachować spójność danych



Wzorce realizacji SOA

- **State Messaging (Wiadomości stanu)**
  - Problem
    - Przechowywanie stanu przez usługę
  - Cel
    - Rozpoznawanie i przekazywanie stanu
  - Rozwiązanie
    - Stan usługi jest dołączany do metadanych wymienianych komunikatów (np. w nagłówkach SOAP)
    - 2 warianty
      - Serwer wykorzystuje komunikat odpowiedzi do przesłania do klienta swojego stanu. W następnym odwołaniu do usługi klient dołącza dane poprzednio zwrócone przez nią jako stan
      - Także klient swój stan przesyła w komunikatach
    - Wady
      - Zagubienie jednego komunikatu powoduje utratę stanu całej dotychczasowej konwersacji
      - Niebezpieczeństwo ataków polegających na zmianie informacji o stanie przez złośliwego klienta
      - Zwiększenie kosztów komunikacji (przesyłanie dodatkowych danych)

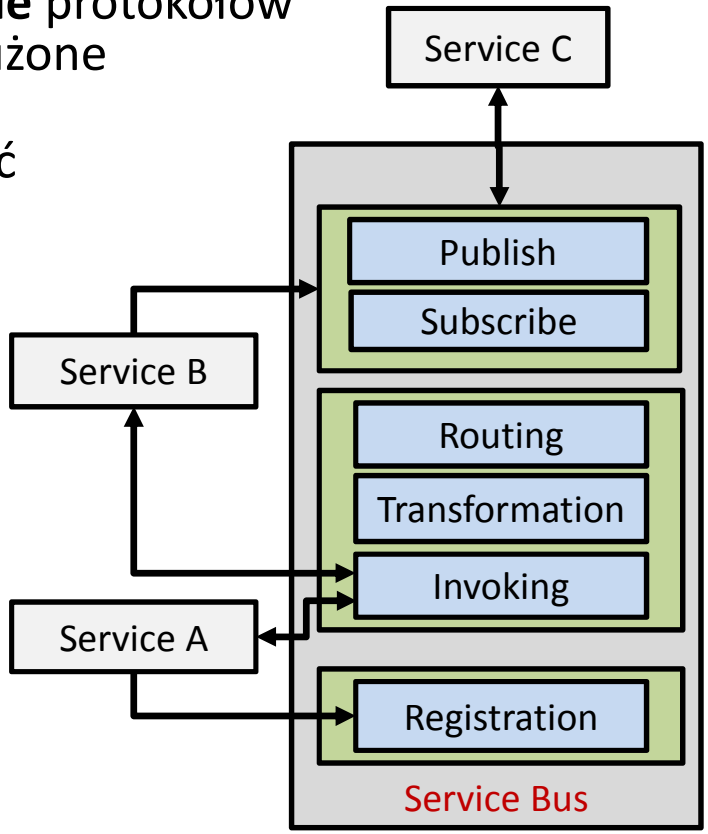


ESB

- **Motywacja**
  - Mnogość aplikacji, różnorodność komunikacji, protokołów, itp.
    $\rightarrow$ potrzeba integracji usług wykorzystujących różne protokoły
- **ESB – Enterpise Service Bus**
  - Centralny element oprogramowania dostarczający infrastrukturę dla komunikacji i integracji usług i ich konsumentów
    - Komunikacja
    - Orkiestracja
    - Zarządzanie
    - Środowisko uruchomieniowe dla usług
  - Magistrala usług umożliwia łączenie usług w sposób zapewniający luźnie powiązania
    - Uwalnia od sieci powiązań każdy-z-każdym, separuje dostawcę od odbiorcy usług oraz zapewnia przejrzystość usług
  - Zadaniem ESB jest zapewnienie solidnego, odpornego na awarię i wiarygodnego systemu wymiany komunikatów.
    - Zapewnia zunifikowaną infrastrukturę do przekształcania, pośredniczenia, routowania i wywoływania komunikatów
    - Różne protokoły transportowe: SOAP, HTTP(S), JMS, SMTP, …
  - Aplikacje łączą się z szyną
    - rejestrują swoje usługi, subskrybują i publikują komunikaty

Wzorce realizacji SOA

- **Wzorzec szyny usługowej -** podstawowe role:
  - Obsługa komunikatów
    - Magistrala umożliwia **wywoływanie** zarejestrowanych usług, **routowanie** komunikatów i **przekształcanie** protokołów i komunikatów tak aby mogły zostać obsłużone
  - **Rejestracja** usług
    - magistrala musi wiedzieć gdzie je odnaleźć
  - **Publikowanie / subskrypcja**
    - magistrala zapewnia usługi subskrypcji komunikatów dla usług,
    - gdy taki komunikat zostanie opublikowany magistrala wywołuje subskrybowane usługi
  - Wzorzec łączy ze sobą kilka innych wzorców projektowych
    - Magistrala komunikatów
    - Router komunikatów
    - Adapter kanałów



ESB

- **ESB**
  - Zalety ESB :
    - dobrze zdefiniowana architektura
    - łatwość podpięcia nowego systemu
    - niezawodność
    - łatwość skalowania
    - rozproszenie
    - luźne wiązania
    - uniezależnienie od lokacji
    - uniezależnienie od formatu wiadomości i protokołu
    - mediacja wiadomości
    - zcentralizowane zarządzanie
  - Wady ESB :
    - duży narzut na początkowy setup
    - duża złożoność - rozwój

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

- **Web Services**
  - Podstawy
    - SOAP
    - WSDL
    - WCF

Web Service

- ***A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.***

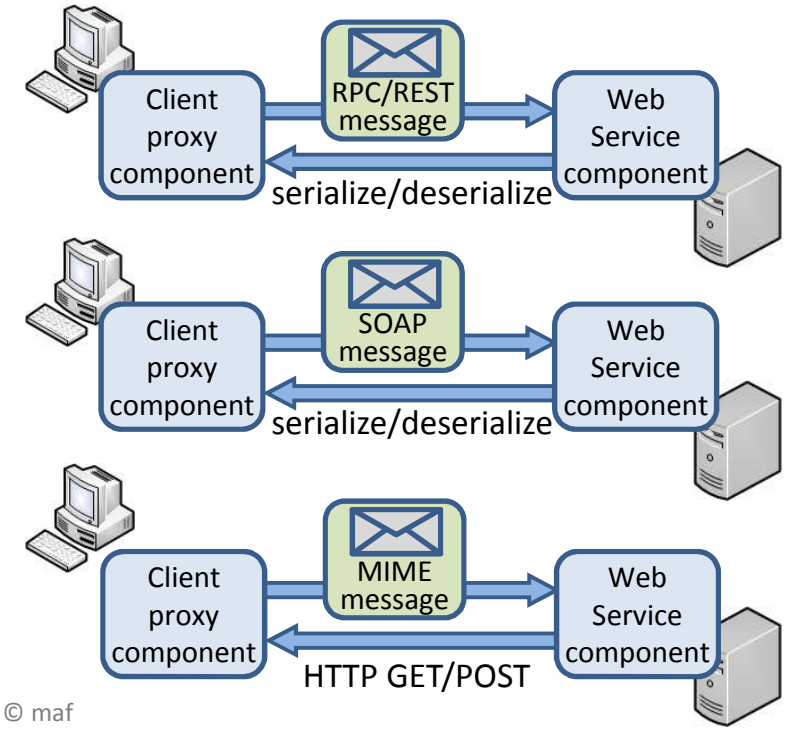
W3C

- ***A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols***

Web Services Architecture Working Group

Web Service

- **Usługa Webowa (Web Service)**
  - Komponent programowy niezależny od platformy wykonawczej i implementacji, dostarczający określonej funkcjonalności lub zasoby
    - W postaci usług (styl wywołania: ∗-RPC, SOAP, HTTP GET/POST)
    - W postaci reprezentacji zasobów (dla REST)
  - Wykorzystujący zestaw określonych technologii umożliwiających otwartą interoperacyjność
  - Mająca dostępny opis
    - Interfejs i wiązanie opisane formatem przetwarzanym maszynowo
    - Typowo XML i WSDL
  - Interakcyjność na podstawie opisu usługi, z użyciem komunikacji w oparciu o wiadomości
    - Niezależny od platformy język XML i protokół SOAP
    - Internetowe standardy komunikacji – np. HTTP



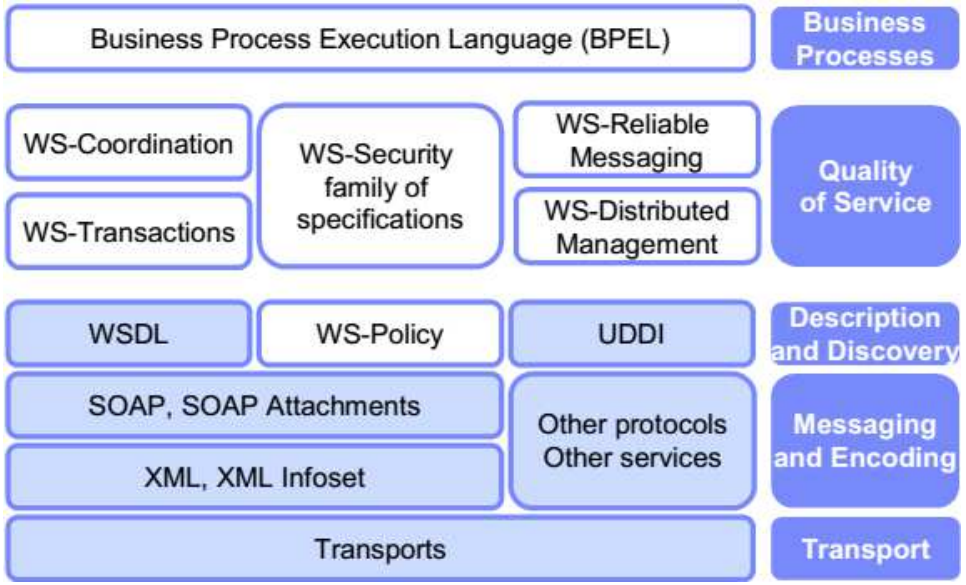
© maf

Web Service

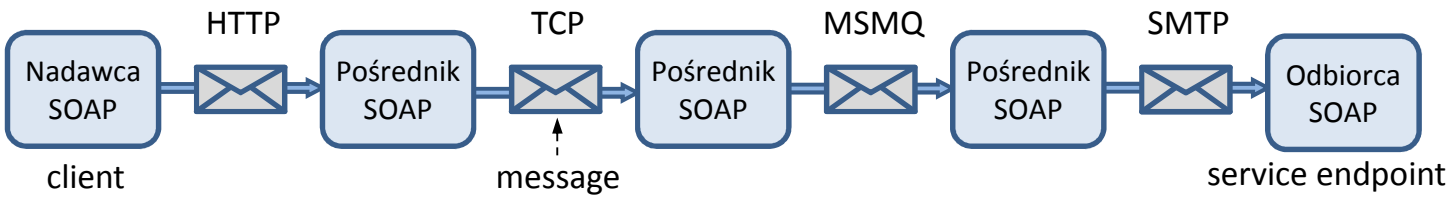
- **WS-I (Web Services Interoperability industry consortium)**
  - Organizacja której celem jest propagowanie dobrych praktyk i standardów dla serwisów działających niezależnie od platformy, systemu operacyjnego, czy języka programowania
- **Standardy WS-I**
  - SOAP
  - WSDL
  - UDDI
  - WS-*
    - WS-Adressing
    - …
  - BPEL
  - XML
  - XML Schema
  - …
- **Profile bazowe**
  - WS-I Basic Profile 1.0:
    - SOAP 1.1
    - WSDL 1.1
    - UDDI 2.0
    - XML 1.0 (Second Edition)
    - XML Schema Part 1: Structures
    - XML Schema Part 2: Datatypes
    - RFC2246: TLS Protocol Version 1.0
    - RFC2616: HyperText Transfer Protocol 1.1
    - The Secure Sockets Layer Protocol Version 3.0
    - …
- **Niektóre frameworki zgodne z WS-I BP**
  - ASP.NET 2.0 - Web Services, GlassFish Metro - JAX-WS, IBM WebSphere App. Server, Microsoft BizTalk Server, SpringWS, …

Web Service

- **Standardy realizacji systemów zgodnie z paradygmatem SOA**

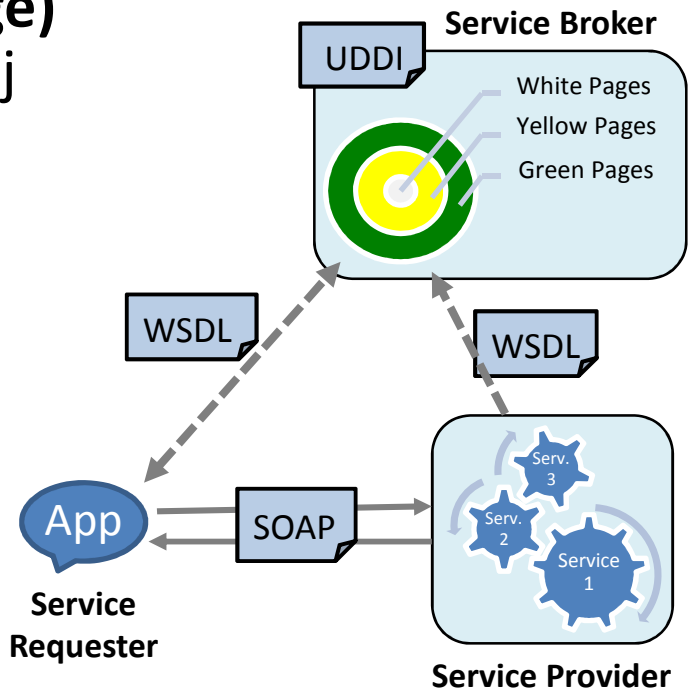


- **Transport**



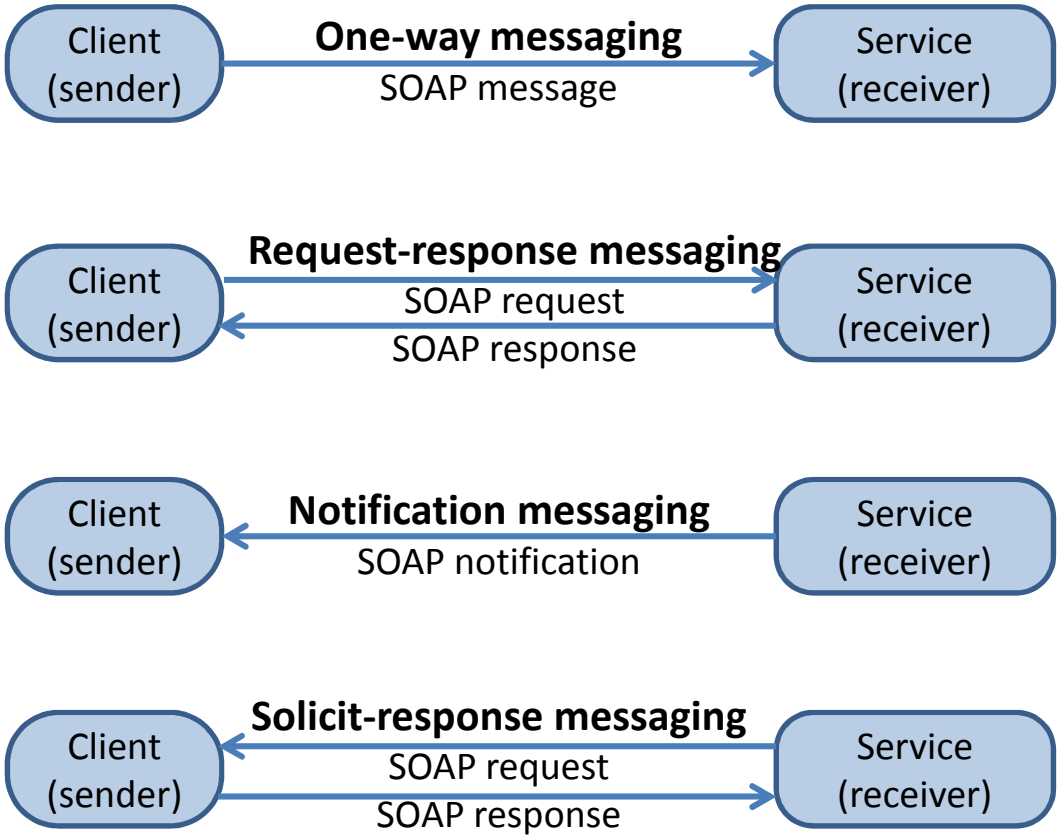
Web Service

- **SOAP (Simple Object Access Protocol)**
  - Protokół do wymiany informacji w środowisku rozproszonym (także w komunikacji z Web Servicem)
  - Transport za pomocą HTTP, SMTP,…
- **WSDL (Web Services Description Language)**
  - Oparty o XML język opisu usługi sieciowej
    - w jaki sposób wywoływać usługę
    - jakich dostarczyć parametrów
    - jaki będzie typ odpowiedzi
    - …
- **UDDI (Universal Description, Discovery and Integration)**
  - rejestr usług internetowych
    - repozytorium WSDL
  - W założeniach ma umożliwiać automatyczne wykrywanie usług



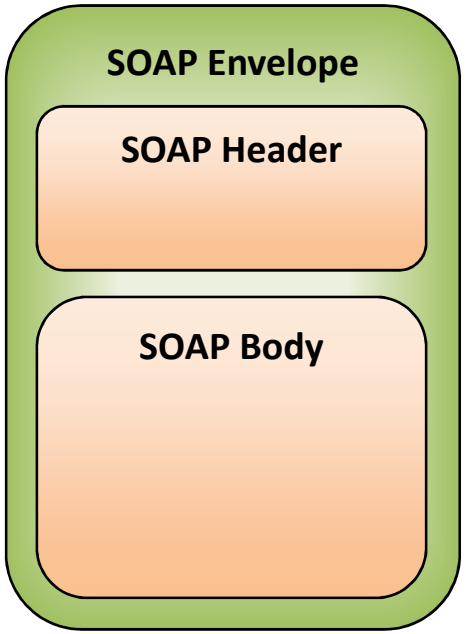
Web Service

- **Podstawowe wzorce komunikacji w Web Services**



SOAP

- **SOAP (Simple Object Access Protocol)**
  - Sam protokół definiuje sposób przetwarzania wiadomości SOAP, moduły i funkcjonalności, sposób adresowania komunikatu i konstrukcję wiadomości
  - Komunikat SOAP zbudowany jest z elementów:
    - Envelope
      - korzeń komunikatu,
      - pozwala na rozpoznanie komunikatu SOAP.
    - Header
      - nagłówek, służy do definiowania funkcjonalności
      - np. uwierzytelnianie
    - Body
      - właściwa treść komunikatu

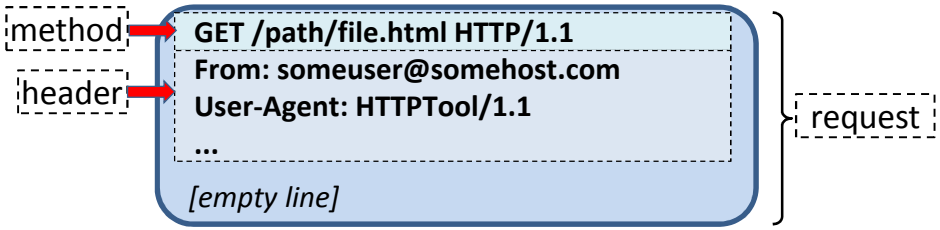


HTTP

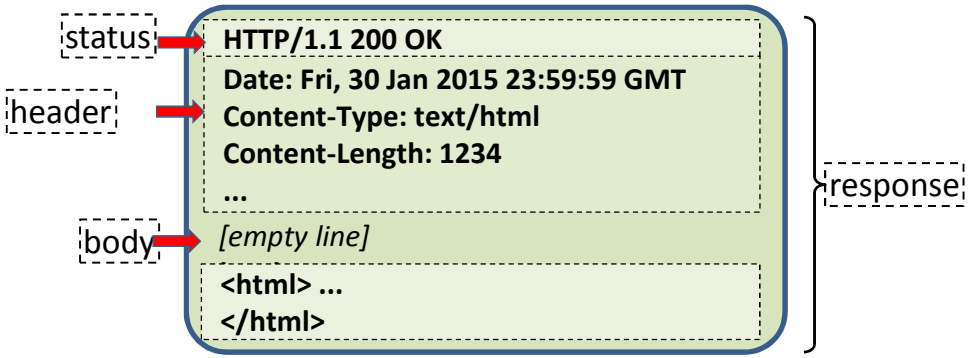
krótkie przypomnienie

- **HTTP message format**
  - Composed of **header** and **body**
    - Empty line after header
    - **HTTP method**
      - Define request type and server reaction
    - Responce status
    - Header fields
      - HTTP headers provide additional information about the data that will be sent
      - Examples:
        - **User-Agent** - identifies the client (browser)
        - **Content-Type** – specify what format and code page is sent in the document
        - **Transfer-Encoding** - Specifies the encoding of the document

HTTP message – example:
GET request:



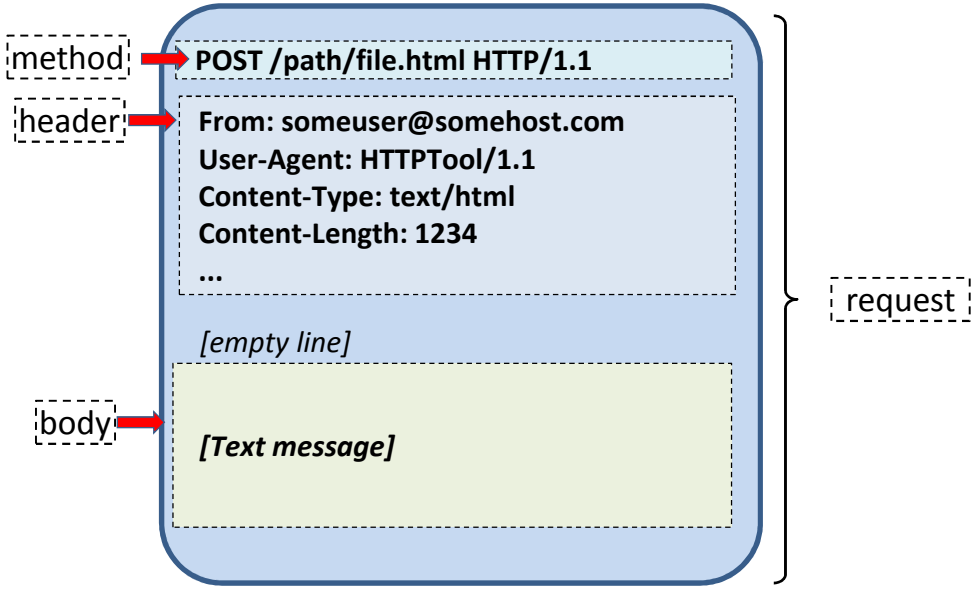
response :



HTTP

krótkie przypomnienie

- **Sending the data**
  - POST request
    - A post request sends form data as part of the HTTP message
    - it hides the submitted data from the user by embedding it in an HTTP message



XML

krótkie przypomnienie

- **XML = Extensible Markup Language**
  - Tekstowy format reprezentacji strukturyzowanej informacji za pomocą znaczników (ang. *tag / markup*)
  - Meta-język pozwalający definiować zbiory znaczników
    - Dostarcza ustalone reguły pozwalające określać struktury znaczników, notacje i ich funkcje
    - W XML nie ma wbudowanych nazw znaczników (jak w HTML)
- **Znacznik / *Markup / Tag***
  - Wyróżniania fragmenty tekstu nadając im określone znaczenie
  - Specjalne znaki wskazują gdzie zaczyna się i kończy instrukcja znacznika
    - *start-tag* – np. <sometag>
    - *end-tag* – np. </sometag>
    - *empty-element tag* – np. <sometag/>
  - Np.

    <centre> Tu jest <bold> jakiś </bold> tekst. </centre>

    ⬇

    Tu jest **jakiś** tekst.
  - Znaczniki są łatwo rozumiane przez człowieka i maszyny

XML

krótkie przypomnienie

- **Struktura dokumentu XML**
  - Dokument XML składa się z nazwanych kontenerów i zawartych w nich wartości danych
    - Kontenery są reprezentowane jako **deklaracje**, **elementy** i **atrybuty**
  - **Element**
    - Ciąg znaków pomiędzy znacznikiem otwierającym i zamykającym
    - Wewnątrz jest treść (ang. *content*)
    - Może zawierać elementy zagnieżdżone

```
<Title> My title  </Title>
```

  - **Atrybut**
    - Para **nazwa:wartość** powiązana z elementem i specyfikowana w znaczniku początkowym
    - Dodatkowe dane dokładniej specyfikujące treść elementu

```
<Chapter  chapter-number="7">
   The content of the chapter
 </Chapter>
```

  - ***Namespace* (przestrzeń nazw)**
    - Dostarcza funkcję kojarzenia elementów i atrybutów w całości lub części dokumentu wg określonego schematu oraz unikania konfliktów nazw
    - Zasięg w deklarowanym elemencie i elementach zagnieżdżonych
    - W postaci URI

```
<Address
  xmlns="http://some.URL/Name">
  …
</Address>
```

XML

krótkie przypomnienie

- **Struktura dokumentu XML**
  - Deklaracja **<?xml>** zawiera informacje o samym dokumencie

Prologue

```
<?xml version="1.0" encoding="UTF-8"?>
<!- Filename: OperationRequest.xml -->

<BookList
  xmlns="http://my.address.com/MyBooks">
  <Book>
    <Title>Web Services: Principles and Technology</Title>
    <Author>Mike Papazoglou</Author>
    <Publisher>Prentice Hall</Publisher>
  </Book>
  ...
  <Book>
    <Title>A Canticle for Leibowitz</Title>
    <Author>Walter M. Miller</Author>
    <Publisher>J. B. Lippincott & Co.</Publisher>
  </Book>
</BookList>
```

XML declaration

komentarz

Root element

zagnieżdżone elementy

XML

- **Namespaces and names**
  - XML *namespaces* provide a way to distinguish between elements that use the same local name but are in fact different
  - **QName** (Qualified Name) consists of a prefix and the local name separated by a colon
  - Namespace specified with prefix: **xmlns:<namespase prefix> = "http://some.URL/Name"**
  - Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<md:Mydata
 xmlns:md="http://my.address.com/MData"
 xmlns:bdata="http://some.address.com/BookData" >
  <md:Book>
    <bdata:Title>Web Services: Principles and Technology</bdata:Title>
    <bdata:Author>Mike Papazoglou</bdata:Author>
    <bdata:Publisher>Prentice Hall</bdata:Publisher>
  </md:Book>
  ...
</md:MyData>
```

XML

- **XML Schema**

  XSD – XML Schema Definition language
  - The XSD can be used to express a set of rules that an XML document must follow to be considered "valid" according to this scheme
  - The XML schema describes the elements and attributes that can be included in a document consistent with this scheme, and how elements can be arranged in the structure of the document
  - Schemas provide support for additional meta-data features, such as structural relations, numerosity, important values and data types
  - Each type of schema works as a method describing data properties and a method of applying rules and restrictions in an XML document

XML

- **XML Schema**
  - XMLSchema refers to a document that specifies the content and structure for XML documents

Person.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema version="1.0" encoding="UTF-8"?>
 <Person
  xmlns:xs="http://www.w3c.org/2001/XMLSchema"
  targetNamespace ="http://...">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FirstName"
         type="xs:string>
        <xs:element name="MiddleName"
         type="xs:string
         minOccurs="0">
        <xs:element name="LastName"
         type="xs:string>
        <xs:element name="Age"
         type="xs:integer>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:schema>
```

Person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
 <Person
  xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
  xsi:NameSchemaLocation="http://some.URI/Person.xsd">
   <FirstName>Nicolaus</FirstName>
   <LastName>Copernicus</LastName>
   <Age>35</Age>
 </Person>
```

XML

- **XML Schema**
  - The **`<schema>`** element is the root element of every XML Schema
    - May contain attributes
    - Especially references to namespaces
  - Elements are the ingredients of an XML schema
    - Are declared using the **`<xs:element>`** element from the XSD.
  - The element declaration defines the **element name**, content model, and allowable attributes and **data types** for each element type.

```
<xs:element name="xxx" type="yyy"/>
```

    - The **`<xs:element>`** element either denotes an element declaration, defining a named element and associating that element with a type, or is a reference to such a declaration
  - The XML document has a reference to an XML Schema
    - Reference to schema namespace – e.g.: `http://www.w3c.org/2001/XMLSchema-instance`
    - With use of **`schemaLocation`** attribute

XML

- **XML Schema**
  - XSD provides a type system for XML processing environments
    - **Simple types**, which define their content in terms of elements and attributes that can contain only data
      - There are a lot of built-in data types. Some common types are:
        - xs:string
        - xs:decimal
        - xs:integer
        - xs:boolean
        - xs:date
        - xs:time
    - **Complex types**, which define their content in terms of elements that may consist of further elements and attributes
  - XSD allows to define their own custom simple types, whose value spaces are subsets of the predefined built-in types
  - The complexType element is used to define structured types.
    - Complex type definitions appear as children of an `xs:schema` element and can be referenced from elsewhere in the schema and from other schemas
    - Complex types typically contain a set of element declarations, element references, and attribute declarations.
  - **Facets** - restrictions for to define acceptable values for XML elements or attributes
    - On values, on set of values (enumarations), on series of values (patterns), …

XML

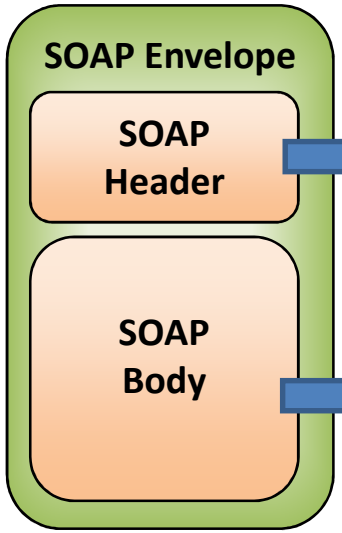
- **Transformation of XML documents**
  - The **XSLT** process transforms the XML structure into any other required forms and structures.
  - XSLT uses **XML Path Language** (or **XPath**) to locate individual fragments of XML documents
    - XPath is a standard for creating expressions that can be used to find specific pieces of information in an XML document.
    - XPath is an abstract language that defines a tree model codifying the logical structure of an XML document

```
<PurchaseOrder>
  <address>
    <street> Dubois st. </street>
    <number> 49 </ number >
    <postal-code> 51-123 </postal-code>
    <city> Wroclaw </city>
  </address>
  <product>
    <product-name> Plastic item </product-name>
    <quantity> 2 </quantity>
  </product>
</PurchaseOrder >
```

```
<PurchaseOrder>
  <address>
    <street> Dubois st. </street>
    <number> 49 </ number >
    <postal> 51-123 Wroclaw </postal>
  </address>
  <product>
    <product-name> Plastic item </product-name>
    <quantity> 2 </quantity>
  </product>
</PurchaseOrder >
```

SOAP

- **SOAP – XML message**
  - The XML structure of SOAP document defines the standard




```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap = "http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
   <!-- Header block -->
   <!-- Header block -->
   ...
 </soap:Header>

 <soap:Body>
   <!-- here message body -->
   <!-- this part is service specific -->
 </soap:Body>
</soap:Envelope>
```

  - A variety of standard header blocks that deal with such topics as security, transaction, QoS, … etc.

SOAP

- **Style komunikacji SOAP**
  - **RPC/Literal**
    (lub **RPC/Encoded**)
    - W Body specyfikacja metody z parametrami
    - Wystawiona/dostępna w postaci servletów, obiektów RMI, DCOM, … itp.
    - Wymagania
      - Lokalizowanie poprzez URI/URL
      - Żądanie:
        - Metoda z parametrami w pojedynczej strukturze
        - Parametry w określonych polach (in lub in-out)
        - Nazwy i wartości w określonym porządku w wiadomości
      - Odpowiedź – analogicznie
  - **Document/Literal**
    (lub **Document/Encoded**)
    - Treść bardzo specyficzna dla danej usługi

SOAP

- **Style komunikacji SOAP – przykład wiadomości**
  - RPC/Literal
  - RPC/Encoding

Request

```
<env:Envelope
 xmlns:env="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
  <env:Header>
   <tx:Transaction-id
    xmlns:tx="http://www.transaction.com/transactions"
    env:mustUnderstand='1'>
    32
   </tx:Transaction-id>
  </env:Header>
  <env:Body>
    <m:GetProductPrice>
      <prod-id> 450R6OP </prod-id >
    </m:GetProductPrice >
  </env:Body>
</env:Envelope>
```

standard SOAP

opis specyficzny dla usługi

w nagłówku elementy niefunkcjonalne

procedura, parametry,...

Response

```
<env:Envelope ... >
  <env:Header> ... </env:Header>
  <env:Body>
    <m:GetProductPriceResponse>
      <prod-price> 123.45 </prod-price >
    </m:GetProductPriceResponse>
  </env:Body>
</env:Envelope>
```

SOAP

- **SOAP communication styles**
  - Document/Literal
  - Document/ Encoding

W body specyficzny dla usługi dokument, który dana usługa "rozumie" (w tym przykładzie zamówienie)

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <tx:Transaction-id xmlns:tx="..." env:mustUnderstand='1'>
   512
  </tx:Transaction-id>
 </env:Header>
 <env:Body>
  <po:PurchaseOrder orderDate="2004-12-02"
   xmlns:po="http://www.plastics_supply.com/POs">
   <po:from>
    <po:accountName> RightPlastics </po:accountName>
    <po:accountNumber> PSC-0343-02 </po:accountNumber>
   </po:from>
   <po:to>
    <po:supplierName> Plastic Supplies Inc. </po:supplierName>
    <po:supplierAddress> Melbourne 12345 </po:supplierAddress>
   </po:to>
   <po:product>
    <po:product-name> injection molder </po:product-name>
    <po:product-model> G-100T </po:product-model>
    <po:quantity> 2 </po:quantity>
   </po:product>
  </po:PurchaseOrder>
 </env:Body>
</env:Envelope>
```

SOAP

- **Dowiązanie / *Binding***
  - SOAP może być przekazywane za pomocą różnych protokołów
    - Standard określa tzw. wiązanie protokołu SOAP
    - Jest to formalny opis w jaki sposób koperty SOAP przekazywane są za pomocą jakiegoś protokołu komunika-cyjnego

```
To: <soap@example.org>
From: <soap@client.com>
Reply-To: <soap@client.com>
....

<?xml version=3D"1.0" encoding=3D"UTF-
8"?>
<SOAP-ENV:Envelope ...
```

SMTP

Request

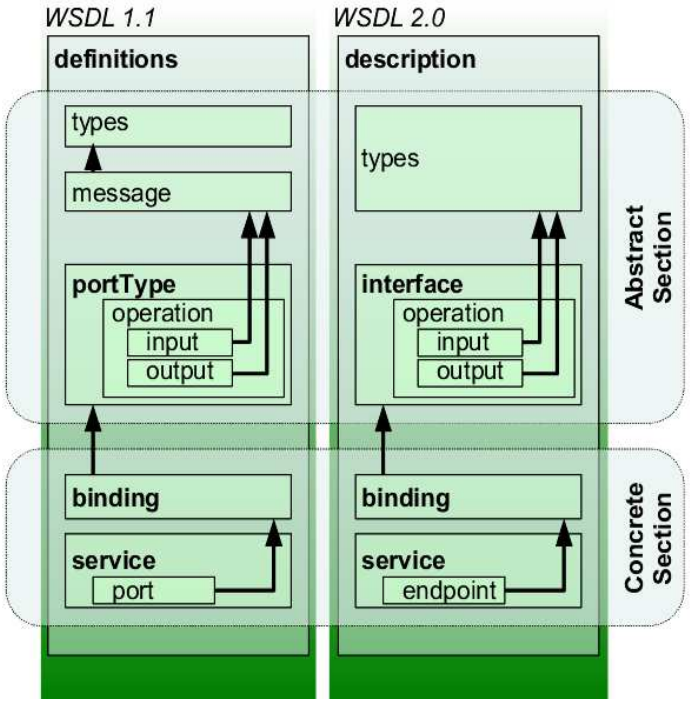
```
POST /path/bank/asmx HTTP/1.1
Content-Type: text/xml
SOAPAction: "urn:banking:transfer"
Content-Lenght: nnnn
<soap:Envelope ...
```

HTTP

Request

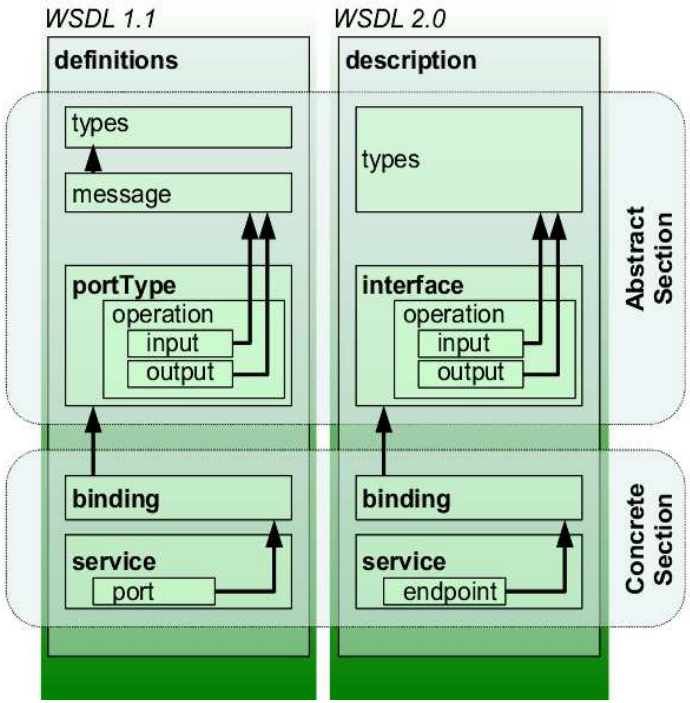
WSDL

- **WSDL (Web Services Description Language)**
  - Reużywalność Web Services polega na wielokrotnym ich użyciu przez różnych klientów w dowolnych zastosowaniach
  - W tym celu należy dostarczyć opis interfejsu usługi
  - Opis najczęściej realizowany za pomocą plików WSDL – dokumentów XML
    - Część abstrakcyjna
      - dotyczy jedynie interfejsu
    - Część druga (konkretna)
      - dotyczy implementacji
  - Uwaga: są różnice dla WSDL 1.1 i 2.0



WSDL

- **Sekcje dokumentu WSDL (1.1 / 2.0)**
  - **definitions / description**
    - **types / types** – definicja typów danych używanych przez usługę (za pomocą XML Schema (XSD))
    - **message / [brak]** – definiuje informacje potrzebne do wykonania operacji usługi – wiadomości jednokierunkowe
    - **portType / interface** – definicja interfejsu usługi – realizowanych operacji i wymienianych wiadomości
      - **operation** – sygnatura operacji (parametr i wynik)
    - **binding/ binding** – łączy abstrakcyjny interfejs z konkretnym protokołem (transport (SOAP) i styl SOAP (RPC lub REST)), także definiuje operacje
    - **service / service** – zbiór dostępnych funkcji
      - **port / endpoint** – adres usługi wiąże binding z konkretnym adresem sieciowym (zwykle URL)

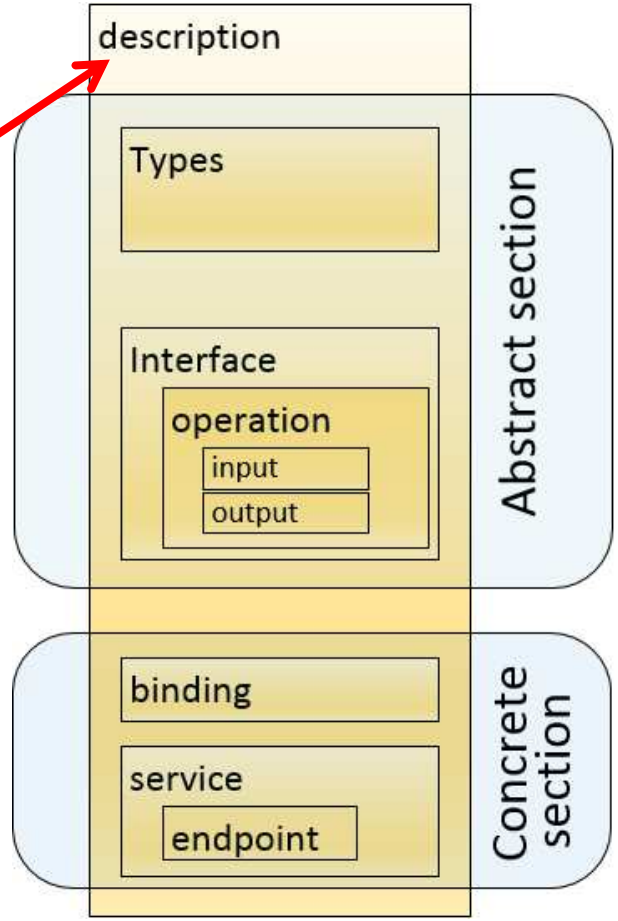


WSDL

Omawiane częściowo

- **Sekcje WSDL - description**
  - Opis dokumentu

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
  xmlns:tns="http://www.tmsws.com/wsdl20sample"
  xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  targetNamespace="http://www.tmsws.com/wsdl20sample">
  ...
```



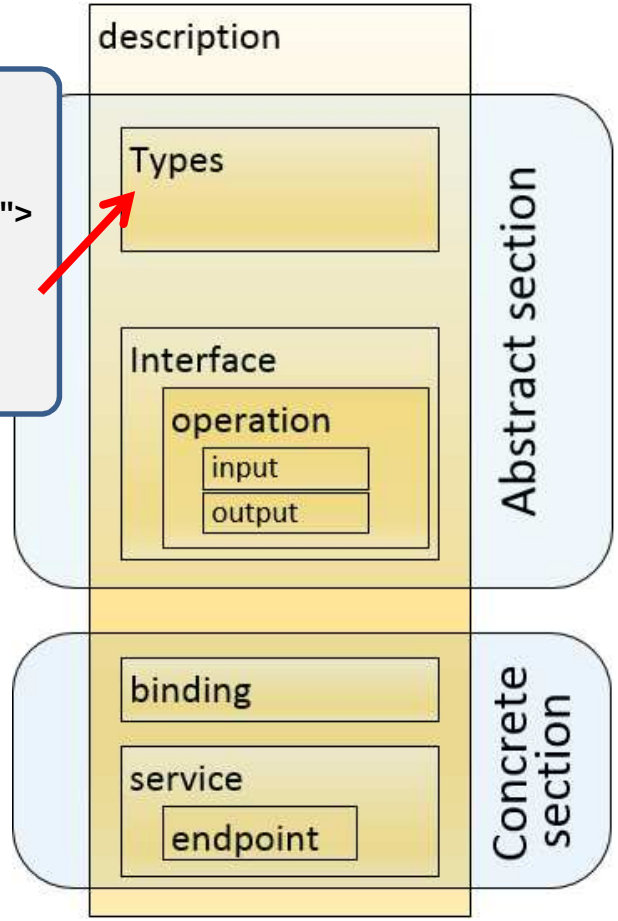
  - Informacja o wersji języka xml
  - użytych przestrzeniach nazw
  - standardowa część plików xml

WSDL

Omawiane częściowo

- **Sekcje WSDL - types**
  - Definicja typów

```
<types>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.tmsws.com/wsdl20sample"
    targetNamespace="http://www.example.com/wsdl20sample">
   <xs:element name="request"> ... </xs:element>
   <xs:element name="response"> ... </xs:element>
 </xs:schema>
</types>
```



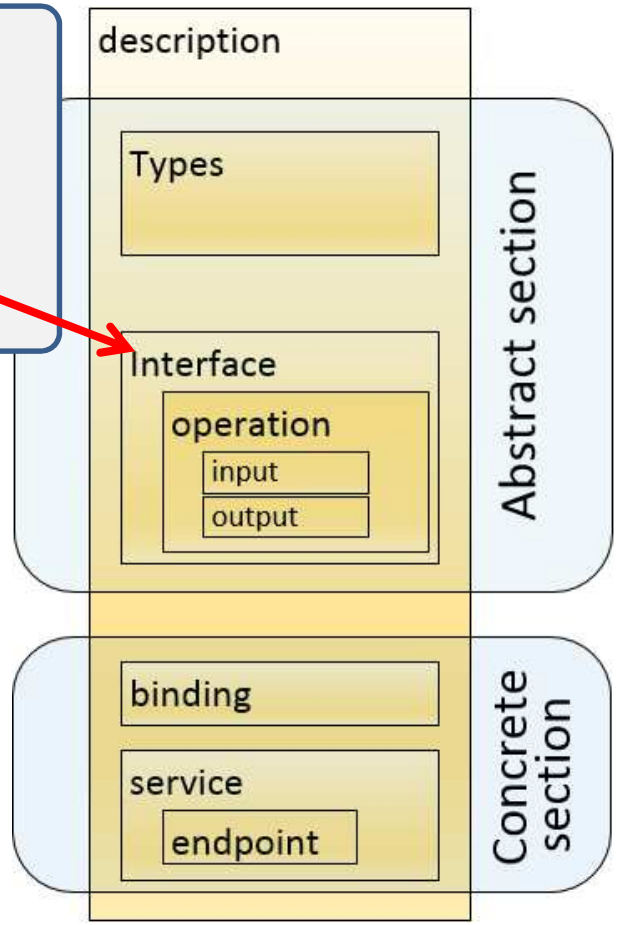
  - request i response – są to jedynie nazwy tych typów
  - Części składowe znajdują się wewnątrz tagów (np. informacja, że typ request będzie składał się z listy Stringów oraz jednego int'a)

WSDL

Omawiane częściowo

- **Sekcje WSDL - interface**

```
<interface name="Interface1">
  <fault name="Error1" element="tns:response"/>
  <operation name="Opp1"
             pattern="http://www.w3.org/ns/wsdl/in-out">
    <input messageLabel="In" element="tns:request"/>
    <output messageLabel="Out" element="tns:response"/>
  </operation>
</interface>
```



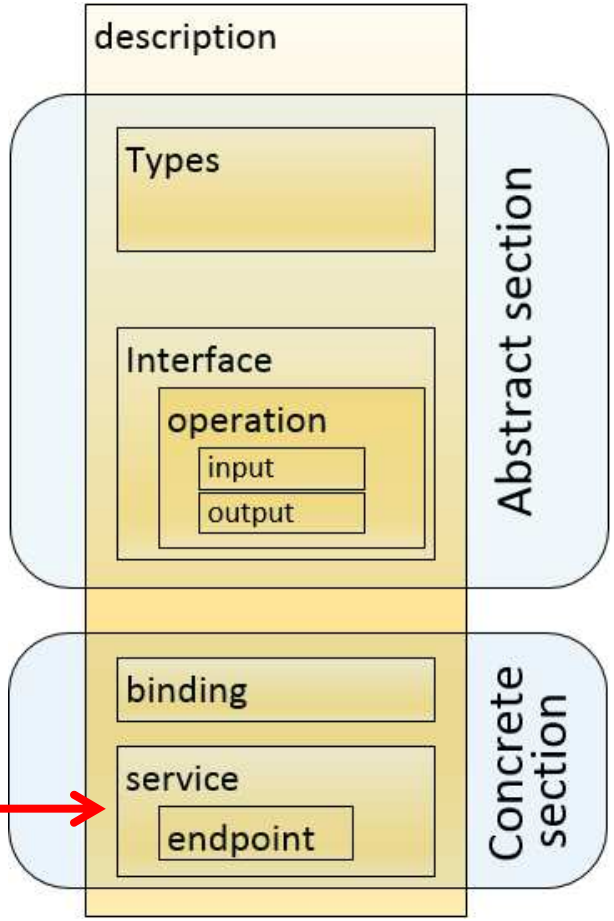
- Do opisu interfejsu należy część odpowiedzialna za opis błędów na tym interfejsie, oraz lista możliwych operacji (tu jedna operacja Opp1)
- Operacja ma nazwę oraz podane nazwy i typy parametrów oraz zwracanego wyniku

WSDL

Omawiane częściowo

- **Sekcje WSDL - service**
  - Opis konkretnej usługi, realizującej operacje z danego interfejsu
  - Ta sama usługa może mieć różne punkty dostępowe (endpoints)
  - Tu:
    - usługa Service1 realizuje interfejs Interface1
    - jest dostępna za pomocą REST pod podanym URL
    - i za pomocą SOAP pod podanym URL

```
<service name="Service1" interface="tns:Interface1">
  <endpoint name="HttpEndpoint"
            binding="tns:HttpBinding"
            address="http://www.example.com/rest/"/>
  <endpoint name="SoapEndpoint"
            binding="tns:SoapBinding"
            address="http://www.example.com/soap/"/>
</service>
```

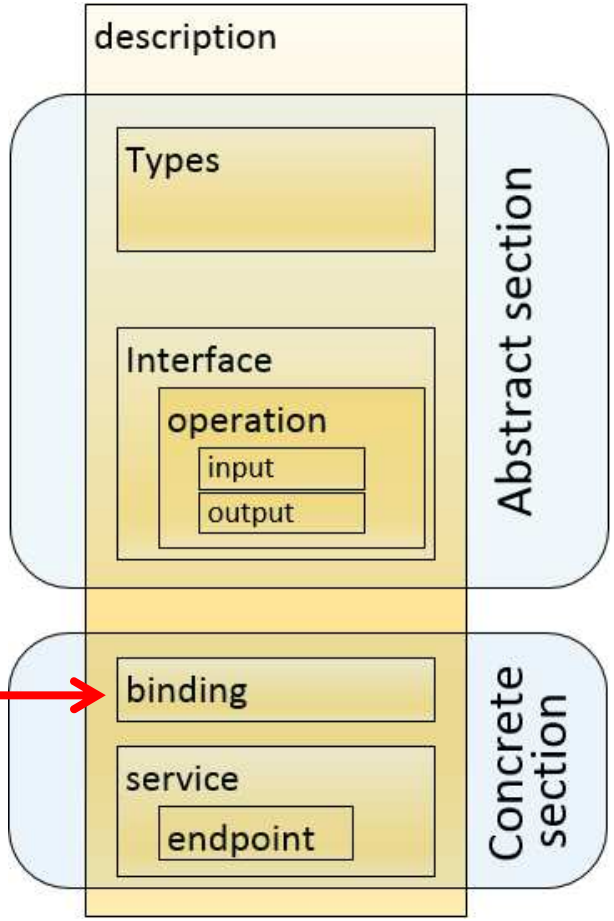


WSDL

Omawiane częściowo

- **Sekcje WSDL - binding**
  - Dowiązanie definiuje protokół oraz realizowane na nim operacje
    - Tu 2 dowiązania:
      - SOAP na HTTP i operację Get – styl żądanie-odpowiedź
      - Żądanie HTTP dla stylu REST

```
<binding name="SoapBinding" interface="tns:Interface1"
         type="http://www.w3.org/ns/wsdl/soap"
         wsoap:protocol="http://www.w3.org/
                         ..../bindings/HTTP/"
         wsoap:mepDefault="http://www.w3.org/
                           ... request-response">
  <operation ref="tns:Get" />
</binding>

<binding name="HttpBinding" interface="tns:Interface1"
         type="http://www.w3.org/ns/wsdl/http">
  <operation ref="tns:Get" whttp:method="GET"/>
</binding>
```



UDDI

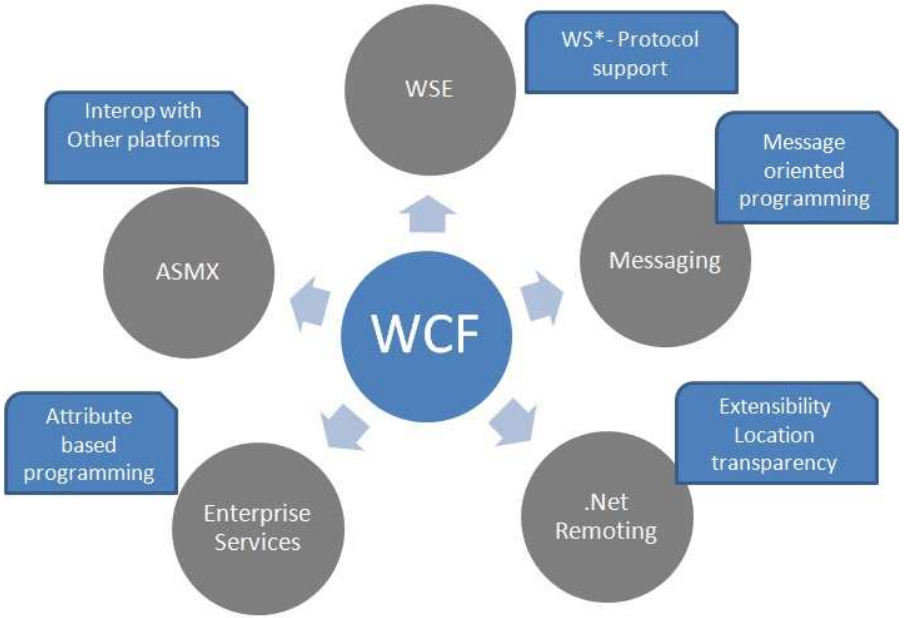
- **UDDI (ang. *Universal Description, Discovery, and Integration*)**
  - Specyfikacja specjalizowanej bazy danych
    - W celu uproszczenia dystrybucji dokumentów WSDL i semantyki komponentów usługowych Web Services
    - Dostępne komponenty są rejestrowane w specjalizowanej bazie danych, która może być następnie przeszukiwana przez twórców aplikacji-klientów lub przez same aplikacje-klientów z zamiarem odkrycia potrzebnych komponentów.
  - Bazy danych UDDI deklarują dostępne usługi (trzy poziomy opisu):
    - ***White Pages***: podstawowe dane kontaktowe o dostawcy usługi, obejmujące nazwę firmy, adres, identyfikator (np. numer identyfikacji podatkowej)
    - ***Yellow Pages***: lokalizacja usługi w taksonomiach kategoryzacyjnych
    - ***Green Pages***: techniczny opis usługi i jej semantyki, często też wskaźnik do pliku WSDL

UDDI

- **UDDI (ang. *Universal Description, Discovery, and Integration*)**
  - Dostęp do rejestru UDDI jest zwykle możliwy zarówno poprzez interfejs HTML, jak i poprzez interfejs programistyczny (np. biblioteka JAXR-API)
  - Dostępność
    - podczas trwania prac nad opracowaniem specyfikacji UDDI wiele uczestniczących w projekcie firm, m.in. Microsoft, IBM i SAP, uruchomiło **publiczne rejestry UDDI**
    - po zakończeniu prac projektowych, wstrzymano obsługę wielu publicznych rejestrów, w zamian oferując **komercyjne rejestry** umożliwiające uruchomienie własnych lokalnych rejestrów
    - **Prywatne rejestry**

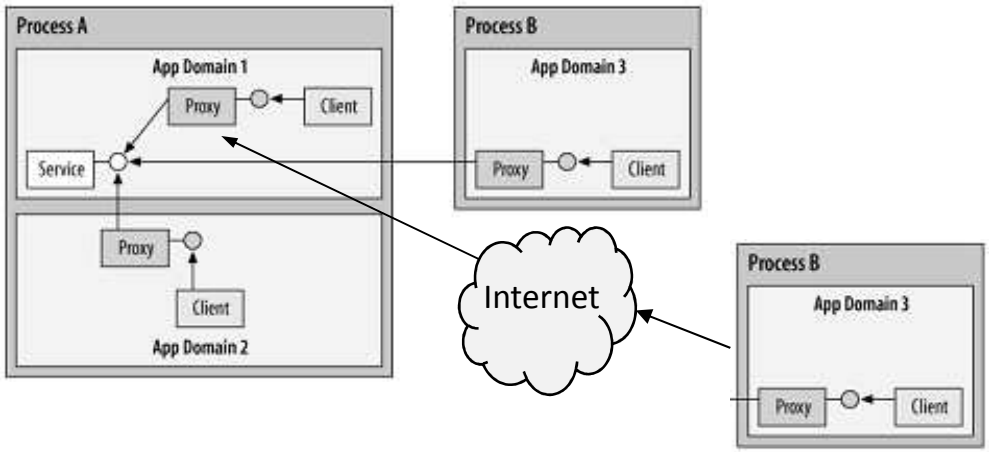
Windows Communication Foundation (WCF)

- **Technologia (platforma deweloperska i system uruchomieniowy) dla tworzenia aplikacji rozproszonych**
  - Fundamentalna cecha: **interoperacyjność**
  - Połączenie różnych technologii Microsoftu do tworzenia aplikacji rozproszonych i komunikacji
    - .NET Remoting,
    - MSMQ,
    - COM+,
    - WebServices
    - ...
  - Dostarcza ujednolicony model obiektowy dla wszystkich używanych technologii



Windows Communication Foundation (WCF)

- **Umożliwia komunikację procesów na jednym komputerze jak i przez sieć LAN oraz przez Internet**
  - Tworząc serwis w WCF nie przejmujemy się takimi kwestiami związanymi z komunikacją jak protokoły transportowe, szyfrowanie itp.
    - Te elementy są ustalane w pliku konfiguracyjnym i nie wpływają na implementację funkcji serwisu

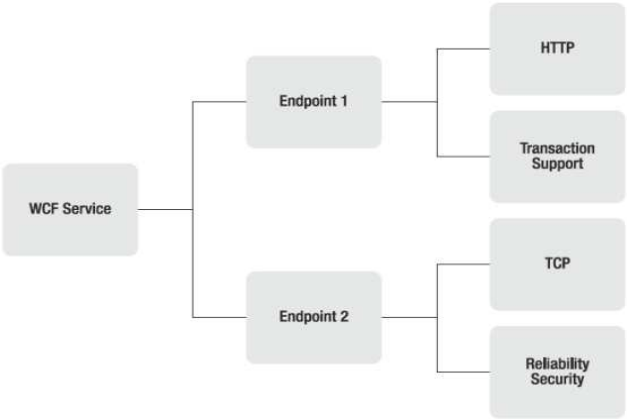


Fundamenty WCF

- ***Service***
  - Element funkcjonalny dostępny w danym środowisku
    - W technologii WCF tworzy się serwisy
  - Aplikacja napisana w WCF, która ma zdefiniowany sposób komunikacji z klientem (interfejsy), protokoły wykorzystywane do komunikacji, pewne dodatkowe właściwości jak bezpieczeństwo, szyfrowanie itp.
  - Hostowana w określony sposób
  - WCF typowo eksponuje metadane opisujące dostępną funkcjonalność
- ***Message***
  - Podstawowa jednostka przesyłana między klientem a *endpoint*'em
  - Format przesyłanych danych nie jest w żaden sposób określony
    - Zależy od serializacji jaką użyjemy (binarna, XML itd.)

Fundamenty WCF

- ***Endpoint***
  - Miejsce gdzie wiadomości są odbierane i/lub wysyłane
  - Określa wszystkie informacje wymagane do wymiany komunikatów
  - Serwis może mieć wiele endpoint'ów
    - Każdy jest pewnego rodzaju sposobem na komunikację z serwisem
    - Każdy endpoint może oferować inne funkcjonalności, różnić się protokołami, dzięki którym można się komunikować z serwisem, itp.



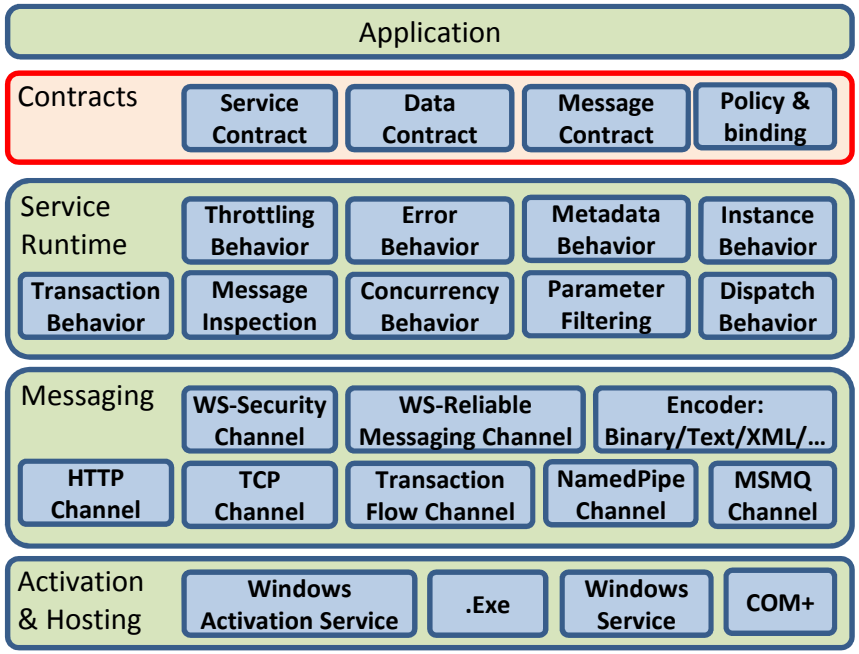
Fundamenty WCF

- ***Behavior***
  - Element, który kontroluje różnymi aspektami wykonania serwisu, endpoint'u czy operacji z nimi związanych
  - Określa pewne zachowania przy wywoływaniu serwisu
    - Np. dostępną liczbę równoległych połączeń
  - Bierze udział w tworzeniu kanałów (ang. *channel*) i może go modyfikować zgodnie z ustawieniami użytkownika.
- ***Channel***
  - Komponent który przetwarza wiadomość w określony sposób
    - Operuje na wiadomościach w celu ich wysłania lub odbioru
    - Realizując również dodatkowe działania jak autentykację lub szyfrowanie
  - 2 rodzaje
    - kanały transportowe – obsługują wysyłanie i odbiór wiadomości (np. TCP, HTTP, …)
    - kanały protokołów – realizują przetwarzanie wiadomości, często poprzez odczyt i zapis dodatkowych nagłówków wiadomości (np. SOAP)

Architektura WCF

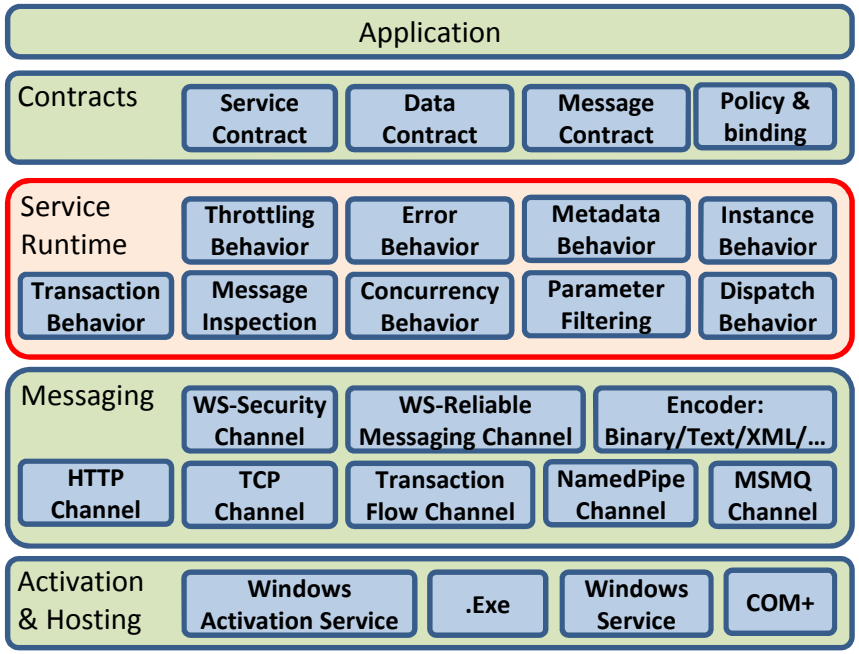
- **Główne komponenty WCF**
  - Contracts
    - Definiuje operacje serwisu, typy i format wiadomości oraz warunki wymagane do komunikacji z serwisem

      - *DataContract* – określa dane (typy) dla serwisu
      - *ServiceContract* – określa operacje/metody serwisu
      - *MessageContract* – definiuje format przesył. wiadomości
      - *Policy and Binding* – opisuje warunki wymagane do komunikacji z serwisem WCF (protokół transportowy, sposób szyfrowania, zasady bezpiecz., …)



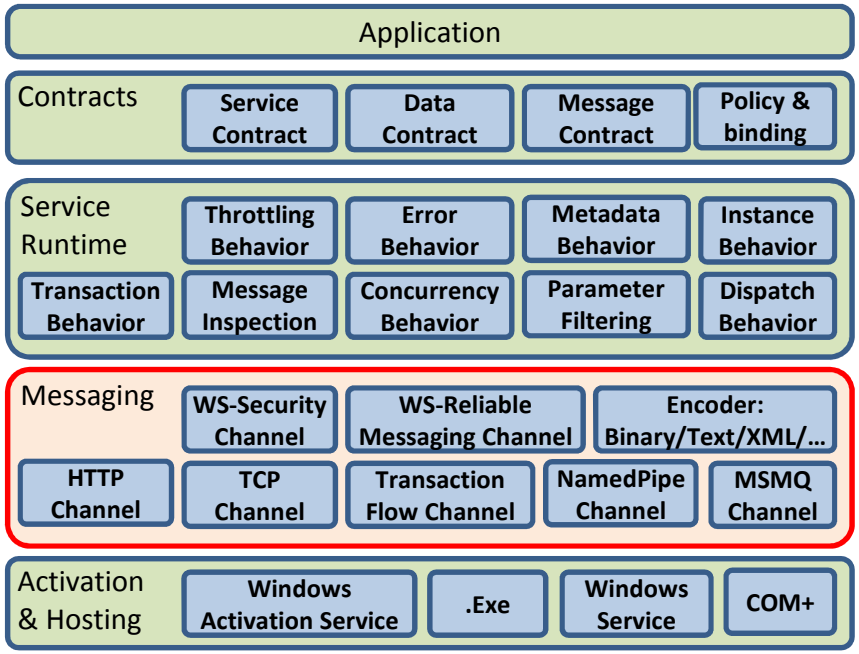
Architektura WCF

- **Główne komponenty WCF**
  - ServiceRuntime
    - Określa **zachowania** (ang. *Behaviors*) – różne własności serwisu, np. które występują podczas wywoływania serwisu
      - *Throttling Behavior* – określa ile wiadomości ma być przetwarzanych
      - *Instance Behavior* – określa ile ma być instancji serwisu
      - *Exception Behavior* – jakie i ile informacji ma być przesłanych do klienta w przypadku wystąpienia wyjątku
      - *Transaction Behavior* – w przypadku błędu umożliwia wycofanie transakcji
      - *Dispatch Behavior* – kontroluje jak jest przetwarzana wiadom. w infrastrukturze WCF



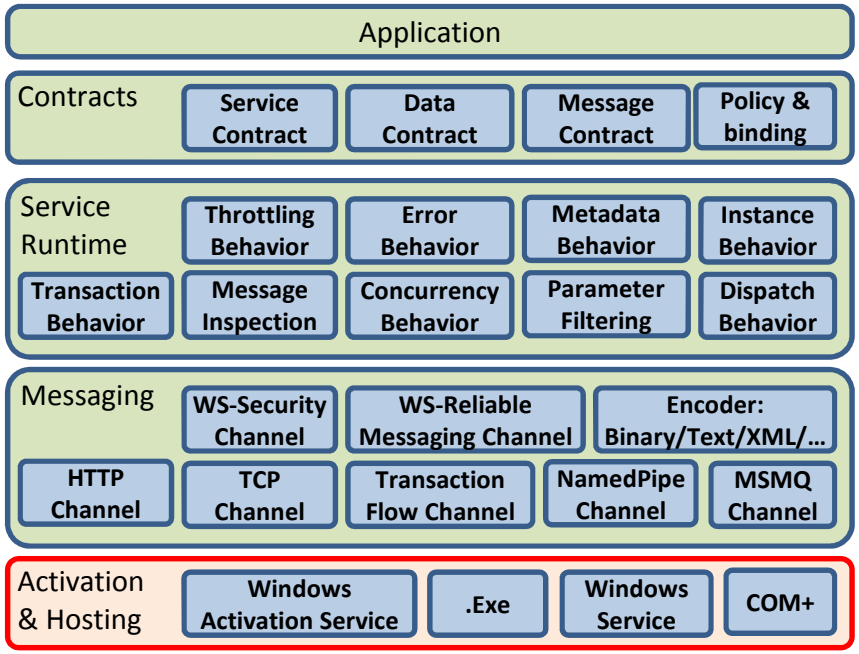
Architektura WCF

- **Główne komponenty WCF**
  - Messaging
    - Warstwa określająca szczegóły związane ze sposobem przesyłania wiadomości
    - Jej elementami są kanały (ang. *channel*), które przetwarzają wiadomości
    - Rozróżniamy dwa typy kanałów:
      - kanały transportowe (ang. *Transport channel*)
        - Obsługują wysyłanie i odbiór wiadomości z sieci (lub innego medium transmisji) np. TCP, MSMQ, HTTP, *named pipes*,
        - Niektóre używają kodowania
      - kanały protokołów (ang. *Protocol channel*)
        - realizują przetwarzanie wiadomości, często poprzez odczyt i zapis dodatkowych nagłówków wiadomości
        - Np. WS-* (SOAP)



Architektura WCF

- **Główne komponenty WCF**
  - Activation and Hosting
    - Serwisy mogą być różnie implementowane, uruchamiane i wykonywane
    - Self-Hosting
      - Pliki wykonywalne .exe (aplikacje konsolowe lub GUI (np. Windows Forms, WPF))
    - IIS
      - Serwer www wywołuje kod
    - WAS (Windows Activation Service)
      - nowy mechanizm aktywacji (w IIS 7)
      - oprócz http wspiera również TCP i *named pipes*
    - Windows Services
      - stand. usługi systemu Windows
    - COM+
      - można hostować komponenty COM+ jako aplikacje WCF



Fundamenty WCF

- **Service = ABC (Address + Binding + Contract)**
  - ***Address*** (gdzie?) - określa lokalizację serwisu (adres endpoint'a)
  - ***Binding*** (jak?) - określa sposób komunikowania się z endpointem
  - ***Contract*** (co?) - określa operacje jakie wykonuje endpoint
- ***Address* (gdzie?)**
  - Format URI:

    [scheme]://[machine address][:port][/path]
    - Formułowany w oparciu o adres bazowy:

      adres endpointa = **[base address]/[optional URI]**
    - Przykłady:
      - HTTP: http://localhost:8001
        https://localhost:8001/MyService
      - TCP: net.tcp://localhost:8002/MyService
      - IPC: net.pipe://localhost/MyPipe

Fundamenty WCF

- **Service = ABC (Address + Binding + Contract)**
  - ***Binding*** (jak?)
    - Jest wiele aspektów komunikacji, jak np. wzorzec komunikacji, kodowanie, protokół transportowy, …
    - Binding określa sposób, w jaki sposób odbywa się komunikacja
      - Transport (HTTP, TCP, IPC, MSMQ, SB)
      - Channel (one-way, duplex, request-reply)
      - Kodowanie (XML, binary, MTOM)
        MTOM - Message Transmission Optimization Mechanism – mechanizm efektywnej transmisji danych binarnych
      - Wspierane protokoły WS-* (bezpieczeństwo, niezawodność, …)
    - Podstawowe wiązania:

Web Service with SOAP messages

Nazwa	Transport	Kodowanie	Interoperacyjność
BasicHttpBinding	HTTP(S)	**Text**, MTOM	Tak
NetTcpBinding	TCP	Binary	Nie
NetNamedPipeBinding	IPC	Binary	Nie
WSHttpBinding	HTTP(S)	**Text**, MTOM	Tak
NetMsmqBinding	MSMQ	Binary	Nie

Fundamenty WCF

- **Service = ABC (Address + Binding + Contract)**
  - ***Contract*** (co?)
    - Definiuje różne aspekty związane z serwisem i komunikacją
    - Określa operacje jakie dostępne sa w endpoint'cie
      - *ServiceContract* – określa operacje/metody, jakie mogą być wywoływane przez klientów danego serwisu – w postaci interfejsu danego języka programowania
      - *DataContract* – określa dane (typy) przyjmowane oraz wysyłane przez serwis
      - *MessageContract* – definiuje kompozycję poszczególnych części przesyłanych wiadomości

```
[ServiceContract]
interface IMyContract
{
  [OperationContract]
  string myMethod(string text);
  //poniżej to już nie metoda kontraktu:
  string myOtherMethod(string text);
}

class MyService : IMyContract {
  public string myMethod(string text) {
    return "Hello " + text;
  }
  public string myOtherMethod(string text) {
    return "tego nie wywołam przez WCF";
  }
}
```

```
[DataContract]
public class MyClass
{
  [DataMember]
  public string myString;
  ...
}
```

Implementacja

- **Definiowanie serwisu – kontrakty i implementacja**

```
using ...

namespace Service1Contract
{
    [ServiceContract]
    public interface IKalkulatorLZ
    {
        [OperationContract]
        LiczbaZ DodajLZ(LiczbaZ n1, LiczbaZ n2);

    ... // inne operacje
}
...
```

*Typy podstawowe, lub **DataContract***

```
[DataContract]
   public class LiczbaZ
   {
       string opis = "Liczba zesp";

       [DataMember]
       public double czescR;

       [DataMember]
       public double czescU;

       [DataMember]
       public string Opis
       {
           get { return opis; }
           set { opis = value; }
       }

       public LiczbaZ(double czesc_rz,
                      double czesc_ur)
       {
           this.czescR = czesc_rz;
           this.czescU = czesc_ur;
       }
   }
}
```

```
using ...

namespace Service1Contract
{
    public class KalkulatorLZ : IKalkulatorLZ
    {
        public LiczbaZ DodajLZ(LiczbaZ n1, LiczbaZ n2)
        {
            Console.WriteLine("Wywolanie: DodajLZ");
            return new LiczbaZ(n1.czescR + n2.czescR,
            n1.czescU + n2.czescU);
        }
        ... // tu inne operacje
} }
```

Implementacja

- **Definiowanie serwisu – specyfikacja interfejsu i operacji**
  - Atrybuty dyrektyw

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace DuplexKontrakt
{
    [ServiceContract (SessionMode = SessionMode.Required,
                      CallbackContract=typeof(IDuplexKalkulatorCallback))]
    public interface IDuplexKalkulator
    {
        [OperationContract]
        void Dodaj(double n);
        [OperationContract(IsOneWay = true)]
        void SumujWszystko();
    }
    public interface IDuplexKalkulatorCallback
    {
        [OperationContract(IsOneWay = true)]
        void WynikDodaj(double result);
        [OperationContract(IsOneWay = true)]
        void WynikSumujWszystko(string eqn);
    }
}
```

Dodatkowa specyfikacja dla kontraktu

Tu: określenie typu instancji usługi

Tu: operacja ze zwrotnym wywołaniem

Dodatkowa specyfikacja dla operacji
Tu: operacja typu one-way (jednokierunkowa/asynchroniczna)

Tu: operacje wywoływane są zwrotnie
Te metody muszą być zaimplementowane w kliencie.

Implementacja

- **Definiowanie serwisu – specyfikacja zachowania**
  - Atrybuty dyrektyw
  - Przykład: typ serwisu *Single*, *Per Session* i *Per Call*

```
[ServiceBehavior (InstanceContextMode=InstanceContextMode.PerCall)]
public class KalulatorLZ : IKalkulatorLZ
{
    ... // kod klasy
}
```

```
[ServiceBehavior (InstanceContextMode=InstanceContextMode.PerSession)]
public class KalulatorLZ : IKalkulatorLZ
{
    ... // kod klasy
}
```

```
[ServiceBehavior (InstanceContextMode=InstanceContextMode.Single)]
public class KalulatorLZ : IKalkulatorLZ
{
    ... // kod klasy
}
```

Implementacja

- **Hostowanie serwisu i konfiguracja w kodzie C#**

```
class Program
{
  static void Main(string[] args)
  {
    // Krok 1 - Utworz URI dla bazowego adresu serwisu
    Uri baseAddress = new Uri("http://localhost:10001/WcfService1/");
    // Krok 2 - Utworz instancje serwisu
    ServiceHost mojHost = new ServiceHost(typeof(KalkulatorSerwis), baseAddress);
    try
    {
      // Krok 3 - Dodaj endpoint
      ServiceEndpoint endpoint1 = mojHost.AddServiceEndpoint(typeof(IKalkulator),new WSHttpBinding(),"KalkulatorSerwis");
      // Krok opcjonalny - konfigurowanie zachowania
      ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
      smb.HttpGetEnabled = true;
      mojHost.Description.Behaviors.Add(smb);
      // Krok 4 - Uruchom serwis
      mojHost.Open();
      ...  // Serwis działa
      // Na koniec – zamykamy serwis
      mojHost.Close();
    }
    catch (CommunicationException ce)
    {
      Console.WriteLine("Wystapil wyjatek: {0}", ce.Message);
      mojHost.Abort();
    }
  }
}
```

base address

są różne konstruktory

endpoint

są różne wersje – metody przeładowane

binding

Tu: opis serwisu dostępny protokołem HTTP (np. dla przeglądarki)

Implementacja

- **Konfiguracja serwisu i klienta – plik App.config**
  - Generowana manualnie lub przez narzędzia platformy

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.serviceModel>
    <services>
      <service name="WcfService1Contract.KalulatorLZ" behaviorConfiguration="Wydajnosc">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:10000/WcfService1" />
          </baseAddresses>
        </host>
        <endpoint address="/KalkulatorLZ" binding="wsHttpBinding" contract="WcfService1Contract.IKalkulatorLZ">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors> ...(na następnym slajdzie)... </behaviors>
  </system.serviceModel>
</configuration>
```

serwis

konfiguracja zachowania

adres bazowy

endpoint (adres względny)

binding

Implementacja

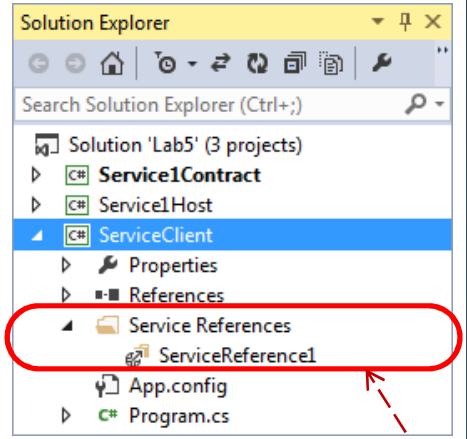
- **Konfiguracja serwisu i klienta – plik App.config**

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  ...(na poprzednim slajdzie)...

  <system.serviceModel>
    <services>
      <service behaviorConfiguration="Wydajnosc" name="WcfService1Contract.KalulatorLZ">
        ...(na poprzednim slajdzie)...
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="Wydajnosc">
          <serviceThrottling
            maxConcurrentCalls="5"
            maxConcurrentSessions="1"
            maxConcurrentInstances="1"
          />
          <serviceMetadata
            httpGetEnabled="true"
            httpGetUrl=""
          />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>

</configuration>
```

konfiguracja zachowania

Implementacja

- **Konfiguracja klienta**
  - Referencje do serwisu
  - App.config



Dodając referencję do serwisu App.config jest generowany przez platformę

Przykład wygenerowanego pliku

binding'i

endpoint'y

App.config*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.serviceModel>
      <bindings>
          <netTcpBinding>
              <binding name="NetTcpBinding_IKalkulatorLZ" />
          </netTcpBinding>
          <wsHttpBinding>
              <binding name="WSHttpBinding_IKalkulatorLZ" />
              <binding name="WSHttpBinding_IKalkulatorLZ2" />
          </wsHttpBinding>
      </bindings>
      <client>
          <endpoint address="http://localhost:10000/Service1/KalkulatorLZ"
              binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_IKalkulatorLZ"
              contract="ServiceReference1.IKalkulatorLZ" name="WSHttpBinding_IKalkulatorLZ">
              <identity>
                  <dns value="localhost" />
              </identity>
          </endpoint>
          <endpoint address="http://localhost:10000/Service1/KalkulatorLZ2"
              binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_IKalkulatorLZ2"
              contract="ServiceReference1.IKalkulatorLZ" name="WSHttpBinding_IKalkulatorLZ2">
              <identity>
                  <userPrincipalName value="DOMAF\Adsuro" />
              </identity>
          </endpoint>
          ...
      </client>
  </system.serviceModel>
</configuration>
```

Implementacja

- **Programowe (dynamiczne) tworzenie proxy w kliencie**

```
[ServiceContract]
public interface IMyClass
{
   [OperationContract]
   string function(string param1, int param2);
}
```

- *Na serwerze jest usługa realizująca pewien kontrakt*
- *Ten interfejs musi być znany także w kliencie*

```
WSHttpBinding myBinding = new WSHttpBinding();
myBinding.Security.Mode = SecurityMode.None;

myBaseAddress = new Uri("http://...");
myEndpoint = new EndpointAddress(myBaseAddress);

ChannelFactory<IMyClass> myCF = new ChannelFactory<IMyClass>(myBinding, myBaseAddress);
IMyClass myClient = myCF.CreateChannel();

string myresult;
myresult = myClient.function("Parameter 1", 123);
( (IClientChannel) myClient ).Close();

Console.WriteLine(myresult);  //any processing with result
```

binding (taki lub inny) 1

adres (Uri) … oraz endpoint 2

utworzenie obiektu obsługującego kanał komunikacji z serwisem (proxy) 3

wywołanie operacji 4

zamkniecie kanału 5

Implementacja

- **Nazwy i typy w hoście i kliencie**

```
[ServiceContract]
public interface IClassName
{
   ...
}
```

```
public class ClassName : IClassName
{
  ...
}
```

- Host:

```
...
// Tworzenie instancji serwisu
ServiceHost mojHost = new ServiceHost(typeof(ClassName), baseAddress);
...
// Endpoint
ServiceEndpoint endpoint1 = mojHost.AddServiceEndpoint(typeof(IClassName),new WSHttpBinding(), ...);
...
```

Nazwa typu klasy

Nazwa typu interfejsu

opcjonalnie adres

- Client:

```
...
// Krok 2 - Utworz instancje serwisu
NameClient klient1 = new NameClient("WSHttpBinding_IName");
```

binding

*Nazwa = nazwa typu interfejsu **bez** "I" + "Client"*

nazwa typu interfejsu
+ (opcjonalnie nr-y)

Działanie

prezentacja serwisu i WSDL dla niego prezentowane online

(niektóre ilustracja na kolejnych slajdach)

WSDL

- **http://localhost:10000/WcfService1?wsdl**

```
<wsdl:definitions name="KalulatorLZ" targetNamespace="http://tempuri.org/">
 <wsp:Policy wsu:Id="WSHttpBinding_IKalkulatorLZ_policy">...tu dużo informacji...</wsp:Policy>
 ...
 <wsdl:types>
  <xsd:schema targetNamespace="http://tempuri.org/Imports">
   <xsd:import schemaLocation="http://localhost:10000/WcfService1?xsd=xsd0"
               namespace="http://tempuri.org/"/>
   <xsd:import schemaLocation="http://localhost:10000/WcfService1?xsd=xsd1"
               namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
   <xsd:import schemaLocation="http://localhost:10000/WcfService1?xsd=xsd2"
               namespace="http://schemas.datacontract.org/2004/07/WcfService1Contract"/>
  </xsd:schema>
 </wsdl:types>

 <wsdl:message name="IKalkulatorLZ_DodajLZ_InputMessage">
  <wsdl:part name="parameters" element="tns:DodajLZ"/>
 </wsdl:message>
 <wsdl:message name="IKalkulatorLZ_DodajLZ_OutputMessage">
  <wsdl:part name="parameters" element="tns:DodajLZResponse"/>
 </wsdl:message>

 <wsdl:portType name="IKalkulatorLZ">
  <wsdl:operation name="DodajLZ">
   <wsdl:input wsaw:Action="http://tempuri.org/IKalkulatorLZ/DodajLZ"
               message="tns:IKalkulatorLZ_DodajLZ_InputMessage"/>
   <wsdl:output wsaw:Action="http://tempuri.org/IKalkulatorLZ/DodajLZResponse"
                message="tns:IKalkulatorLZ_DodajLZ_OutputMessage"/>
  </wsdl:operation>
 </wsdl:portType>
 ...
```

WSDL

- **http://localhost:10000/WcfService1?wsdl**

```
...
<wsdl:binding name="WSHttpBinding_IKalkulatorLZ" type="tns:IKalkulatorLZ">
 <wsp:PolicyReference URI="#WSHttpBinding_IKalkulatorLZ_policy"/>
 <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="DodajLZ">
   <soap12:operation soapAction="http://tempuri.org/IKalkulatorLZ/DodajLZ" style="document"/>
   <wsdl:input>
     <wsp:PolicyReference URI="#WSHttpBinding_IKalkulatorLZ_DodajLZ_Input_policy"/>
     <soap12:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
     <wsp:PolicyReference URI="#WSHttpBinding_IKalkulatorLZ_DodajLZ_output_policy"/>
     <soap12:body use="literal"/>
   </wsdl:output>
 </wsdl:operation>
</wsdl:binding>

<wsdl:service name="KalulatorLZ">
 <wsdl:port name="WSHttpBinding_IKalkulatorLZ" binding="tns:WSHttpBinding_IKalkulatorLZ">
   <soap12:address location="http://localhost:10000/WcfService1/KalkulatorLZ"/>
   <wsa10:EndpointReference>
     <wsa10:Address>http://localhost:10000/WcfService1/KalkulatorLZ</wsa10:Address>
     <Identity>
      <Dns>localhost</Dns>
     </Identity>
   </wsa10:EndpointReference>
 </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

WSDL

- **http://localhost:10000/WcfService1?xsd=xsd0**

```
<xs:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
  <xs:import schemaLocation="http://localhost:10000/?xsd=xsd2"
             namespace="http://schemas.datacontract.org/2004/07/ServiceContract"/>
  <xs:element name="DodajLZ">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="n1" nillable="true" type="q1:LiczbaZ"/>
        <xs:element minOccurs="0" name="n2" nillable="true" type="q2:LiczbaZ"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DodajLZResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="DodajLZResult" nillable="true" type="q3:LiczbaZ"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Funkcja1">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="s1" nillable="true" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Funkcja1Response">
  ...
  </xs:element>
</xs:schema>
```

WSDL

- **http://localhost:10000/WcfService1?xsd=xsd2**

```
<xs:schema elementFormDefault="qualified" namespace="http://schemas.datacontract.org/2004/07/ServiceContract"/>
  <xs:complexType name="LiczbaZ">
    <xs:sequence>
      <xs:element minOccurs="0" name="Opis" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="czescR" type="xs:double"/>
      <xs:element minOccurs="0" name="czescU" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="LiczbaZ" nillable="true" type="tns:LiczbaZ"/>
</xs:schema>
```

Działanie

d:\Users\Mariusz\PROGRAMY\VStudio\Wcf3\WcfService1Host\bin\Debug\WcfService1Host.exe

```
List all behaviors:

Service endpoint WSHttpBinding_IKalkulatorRU contains the following:
Binding: System.ServiceModel.WSHttpBinding
Contract: System.ServiceModel.Description.ContractDescription
ListenUri: http://localhost:10001/WcfService1/KalkulatorRU
ListenUriMode: Explicit
Serwis 1 jest uruchomiony.
Displaying information for contract: IKalkulatorRU
        Display all behaviors:
                System.ServiceModel.Dispatcher.OperationSelectorBehavior
        Configuration name: WcfService1Contract.IKalkulatorRU
        Contract type: WcfService1Contract.IKalkulatorRU
        Name: IKalkulatorRU
        Namespace: http://tempuri.org/
        Display Operations:
                DodajLRU
                Dzialanie1
                Dzialanie2
        SessionMode: Allowed
        Inherited Contracts:
Nacisnij <ENTER> aby zakonczyc.

Return: liczba urojona
Dzialanie 1: 111
Dzialanie 2: 222
Return: liczba urojona
Dzialanie 1: 111
Dzialanie 2: 222
Return: liczba urojona
Dzialanie 1: 111
Dzialanie 2: 222
```

w tym przykładzie inne są nazwy zmiennych
M.in.: RU → LZ
lub RU → Z

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

- **REST**
  (RESTful)

REST

- **REST = REpresentational State Transfer**
  - Jest to styl architektoniczny (wzorzec architektury oprogramowania) **zorientowany na zasoby**, który narzuca określone **zasady realizacji usług** systemu zarządzającego **hypermediami**
    - Definiuje zestaw ograniczeń na elementy architektury wymuszające pożądane własności tej architektury
    - Opisuje zbiór reguł i dobrych praktyk realizacji oprogramowania
    - Ułatwia obsługę żądań i odpowiedzi w nowy i łatwiejszy sposób
  - Idea:
    - Bez konieczności odwoływania się do złożonych dokumentacji
    - Dostęp i operowanie tekstową reprezentacją zasobów z użyciem zunifikowanych operacji bezstanowych
  - Hasło: " Change of state by representation „
    - REST jest zorientowany na zasoby, nie akcje
    - Każdy otrzymany zasób reprezentuje następny stan aplikacji
    - Zasoby = adresowalne operacje (np. poprzez URI)

REST

- **Zasady / ograniczenia architektoniczne (*archit. constraints*)**
  - *Client-server collaboration model*
    - *Separation of concerns*
      - M.in. separacja części klienckiej (interfejs, stan,…)
    - Pozwala na łatwą modyfikację systemu
  - *Layered system*
    - Hierarchia warstw systemowych realizujących różne aspekty przetwarzania
    - Klient nie ma wiedzy czy jest połączony bezpośrednio z serwerem końcowym
      - Może być podłączony do węzła pośredniczącego (o różnych funkcjonalnościach)
      - Sprzyja to skalowalności oraz bezpieczeństwu
  - *Cacheability*
    - Możliwość uzyskania odpowiedzi z pamięci podręcznej
      - Pamięć podręczna klienta lub serwer proxy
    - Odpowiedzi muszą być domyślnie lub jawnie definiowane jako buforowane lub nie, aby zapobiec ponownemu wykorzystaniu niewłaściwych danych
    - Dobrze zarządzane buforowanie może prowadzić do poprawy skalowalności oraz wydajności

REST

- **Zasady / ograniczenia architektoniczne** (kont.)
  - *Statelessness*
    - Bezstanowa interakcja pomiędzy klientem i serwerem
      - Serwer nie przechowuje kontekstu klienta pomiędzy wykonywaniem zapytań
    - Każde zapytanie zawiera tylko tyle informacji, ile jest do jego przetworzenia
    - Stan sesji może przechowywać klient
      - Może zostać poproszony do przesłania go do innej usługi (np. w celu przeprowadzenia autentykacji)
      - Klient przesyła żądania, gdy jest gotowy do przejścia do nowego stanu
  - *Code on demand*
    - Rozszerzenie funkcjonalności klienta przez dostarczenie kodu wykonawczego (z serwera)
      - Kod może zawierać takie elementy jak aplety Java lub JavaScript
      - Pozwala zmniejszyć ruch w sieci i odciążyć serwer
    - Te podejście pozwala na tworzenie uniwersalnego klienta wspierającego dostęp do wielu serwisów
      - Tzw. *rich client* (np. przeglądarka webowa)
    - Funkcja opcjonalna (nie każdy system musi ją dostarczać)

REST

- **Rules / architectural constraints** (cont.)
  - **A multiform representation**
    - The resources are separated from the various representation forms that can be returned to the client
      - The same resource can be delivered in multiple representations
      - E.g. XML / JSON / CSV / Atom / ...
    - It should be used platform/technology-independent data format to transmit information

- JSON:

```
{
    "ID": "13",
    "Name": "Kowalski",
    "Email": "kowalski@gmail.com",
    "Country": "Poland"
}
```

- XML:

```
<Person>
  <ID>13</ID>
  <Name>Kowalski</Name>
  <Email>kowalski@gmail.com</Email>
  <Country>Poland</Country>
</Person>
```

REST

- **Rules / architectural constraints** (cont.)
  - **Uniform interface**

    Fundamental constraints placed on interface between components in order to simplify all things
    - **Identification Of Resources**
      - A request must unambiguously identify the resource that it is looking for
    - **Resource Manipulation through Representation**
      - When a client receives resource representation (with metadata), it should have enough information to manipulate the resource
    - **Self Descriptive Messages**
      - …
    - **Hypermedia as the Engine of Application State (HATEOAS)**
      - …

REST

- **Rules / architectural constraints** (cont.)
  - **Uniform interface** (cont.)
    - **Identification Of Resources**
      - …
    - **Resource Manipulation through Representation**
      - …
    - **Self Descriptive Messages**
      - A message should have enough information to know how to process it (i.e. the type of request, MIME types (to know how to parse it), etc.)
      - With respect to multiform representation
    - **Hypermedia as the Engine of Application State (HATEOAS)**
      - Accessing resources should be analogous to accessing a web page
      - Client should be able to discover all available actions and resources it needs
      - A service can provide links to available actions in responses

REST

- **HATEOAS - Hypermedia as the Engine of Application State**
  - It is assumed that the REST client does not need to have any a`priori knowledge how to invoke actions that the application (service) provides beyond a generic understanding of hypermedia
  - The client cooperates with the application (service) using information discovered within resource representations returned from the server
    - Hyperlinks
    - Link relations
      - A descriptive **rel attribute** attached to a hyperlink in order to define the type of the link, or the relationship between the source and destination resources
      - Defined set of standardized link relations in RFC and procedure for extending it
  - In the resource representation are included **available actions based on current application state**
  - Client simply navigates through resources

REST

- **HATEOAS – simple request and response example**
  - Request for account resource

```
GET /accounts/12345 HTTP/1.1
 Host: bank.example.com
 Accept: application/xml
 ...
```

  - Response with XML representation

```
<?xml version="1.0"?>
 <account>
  <account_id>12345</account_id>
  <balance currency="usd">1000.00</balance>
  <link rel="deposit"  href="https://bank.example.com/accounts/12345/deposit" />
  <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
  <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
</account>
```

    - Response depends on state
    - E.g. if balance=0 the avail. actions could be different – e.g. with:

```
<link rel="close" href="https://bank.example.com/accounts/12345/close" />
```

REST

- **Exapmle of JSON based representation**
  - For get resource of user of id=3
    in *users* collection

this shows what action on resource is possible and how to do it
(the link is relative to base)

```
{ "users":
  { "id": 3,
    "name": "Adam",
    "country": "Poland",
    "links": [
      { "href": "/users/3",
        "rel": "edit the item",
        "method": "PUT"
      },
      { "href": "/users/3",
        "rel": "delete the item",
        "method": "DELETE"
      }
    ]
  },
  "links": {
    "href": "/users",
    "rel": "list all items",
    "method": "GET"
  },
  {
    "href": "/users",
    "rel": "add an items",
    "method": "POST"
  }
}
```

RESTful Web services

- **Web services that adhere to the REST constraints are called RESTful**
- **HTTP-based RESTful APIs are defined with the following aspects:**
  - URIs/URLs are used to locate the resource
  - **Base URL** (such as http://example.com/resources)
    - All other URLs are provided in resource representations
    - In **RESTless** services, URLs (APIs) are often provided in specification instead
  - An **Internet media type** that defines state transition data elements (e.g. application/json, application/xml)
    - The current representation tells the client how to compose requests for transitions to all the next available application states
    - This could be as simple as a URL or as complex as a Java applet
  - **Standard HTTP methods**

RESTful Web services

- **Basic HTTP methods used in RESTful**

Method	Operation performed on the resource	Operation property
GET	Reading an individual resource or collection	Safe
PUT	Replacement/update of the indicated resource or the whole collection (if the resource does not exist it will be created)	Idempotent
POST	Adding a new resource or collection	Safety depends on implementation
DELETE	Removal of the indicated resource or the entire collection	Idempotent
OPTIONS	Retrieving the list of available operations on the indicated resource	Safe
PATCH	Update a part of resource	Idempotent

- **The essence is combination of the method and the URL !**
  - URLs should be defined in compliance with the rule to be clean: simple, obvious, consistent, app. implementation independent, intuitive, self-explanatory

RESTful Web services

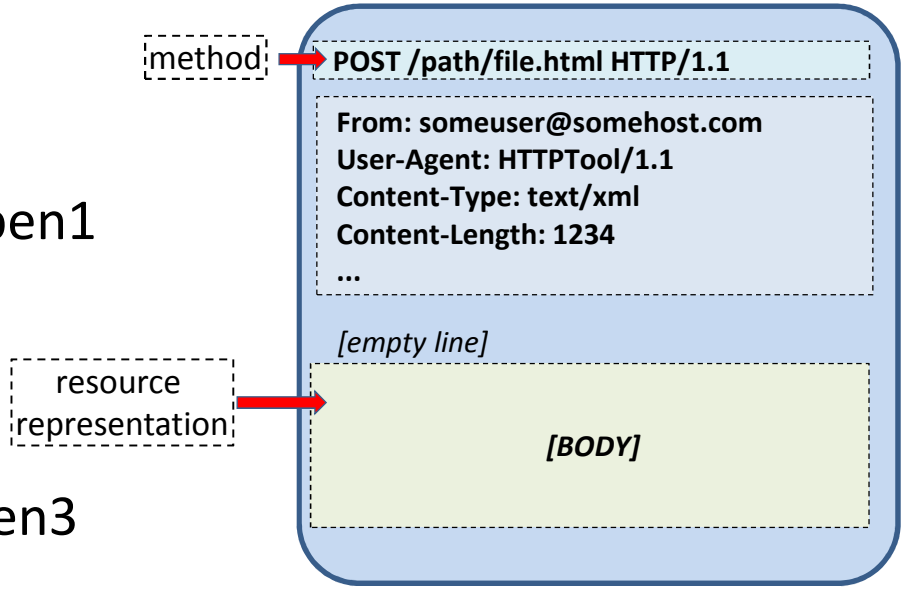
- **CRUD**
  - The resource operation define: **HTTP method and URL**
    - POST – **C**reate the resource
    - GET – **R**etrieve the resource
    - PUT – **U**pdate the resource (also create if {id} is known)
    - DELETE – **D**elete the resource

URL	Method	Operation
http://example.com/items	GET	Get list (collection) of items
http://example.com/items/{id}	GET	Get item data of item with id={id}
http://example.com/items	POST	Adding a new item in collection of items
http://example.com/items/{id}	PUT	Updating item data of item with id={id} (or adding an item with id={id} – this is not standard)
http://example.com/items	DELETE	Removal of the whole collection of items
http://example.com/items/{id}	DELETE	Removal of item with id={id}

- where {id} is specified unique identifier – any unique string
- e.g.: {id}=123 and URL is: http://example.com/items/123

RESTful Web services

- **CRUD – examples**
  - Retrieving the resource:
    - **GET** http://service.org/pens
    - **GET** http://service.org/pens/yellowpen1
  - Creation the resource:
    - **POST** http://service.org/pens
      - service creates new id – more robust
    - **POST** http://service.org/pens/bluepen3
      - when client delivers new id
      - not recommended
      - In such case some id control mechanism must be implemented
    - **PUT** http://service.org/pens/bluepen3
      - not standard !



RESTful Web services

- **CRUD – other examples**
  - Updating the resource
    - **PUT** http://service.org/pens/bluepen24
  - Deleting the resource:
    - **DELETE** http://service.org/pens/greenpen5
    - **DELETE** http://service.org/pens
      - Rarely allowed
  - Performing other operations on items

    – in general the **operation** should be specified **after specification of the resource**

    – e.g.:
    - http://service.org/pens/bluepen3/bid
    - http://service.org/pens/greenpen5/buy

RESTful Web services

- **Recommended result codes for REST**

HTTP method	CRUD	Entire Collection (e.g. /items)	Specific Item (e.g. /items/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of items. (use pagination, sorting and filtering to navigate big lists)	200 (OK), single item. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) … or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) … or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

- Some othe specific codes

401	Unauthorized	The request did not include an authentication token or the authentication token was expired.
403	Forbidden	The client did not have permission to access the requested resource.
500	Internal Server Error	The request was not completed due to an internal error on the server side.
503	Service Unavailable	The server was unavailable.

RESTful Web services

- **WCF RESTful – a short view on example implemention**
  - Note: here json format is used

```
[ServiceContract]
public interface IRestService1
{
  [OperationContract]
  [WebGet(UriTemplate = "/books", RequestFormat = WebMessageFormat.Json,
          ResponseFormat = WebMessageFormat.Json, BodyStyle = WebMessageBodyStyle.Bare)]
  List<Book> getAllBooks();

  [OperationContract]
  [WebGet(UriTemplate = "/books/{id}", RequestFormat = WebMessageFormat.Json,
          ResponseFormat = WebMessageFormat.Json)]
  Book getBookById(string Id);

  [OperationContract]
  [WebInvoke(UriTemplate = "/books/{id}", Method = "PUT", RequestFormat = WebMessageFormat.Json,
             ResponseFormat = WebMessageFormat.Json)]
  string updateBook(Book aBook, string Id);

  [OperationContract]
  [WebInvoke(UriTemplate = "/books", Method = "POST", RequestFormat = WebMessageFormat.Json,
             ResponseFormat = WebMessageFormat.Json)]
  string addBook(Book aBook);

  [OperationContract]
  [WebInvoke(UriTemplate = "/books/{id}", Method = "DELETE")]
  string delBookById(string Id);
}
```

relative URI of operation

```
List<Book> books;
...
public Book getBookById(string Id) {
 int bookId = int.Parse(Id);
 return books.Find(b => b.Id == bookId);
}
```

C# methods implementing operations

message format

HTTP method

WebGet dla metody HTTP GET

WebInvoke dla metod POST, PUT, ...

SOP and CORS

- **Same-Origin Policy (SOP)**
  - The same-origin policy is a critical security mechanism that restricts how a document or script loaded from one origin can interact with a resource from another origin. It helps isolate potentially malicious documents, reducing possible attack vectors.

    mozilla.org
  - In computing, the same-origin policy is an important concept in the web application security model. Under the policy, a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same origin

    wikipedia.org
  - **It is a security mechanism implemented in browsers, by means of which the possibility or lack of using resources (including executable scripts) from sources other than the original source of the obtained content is controlled.**

SOP and CORS

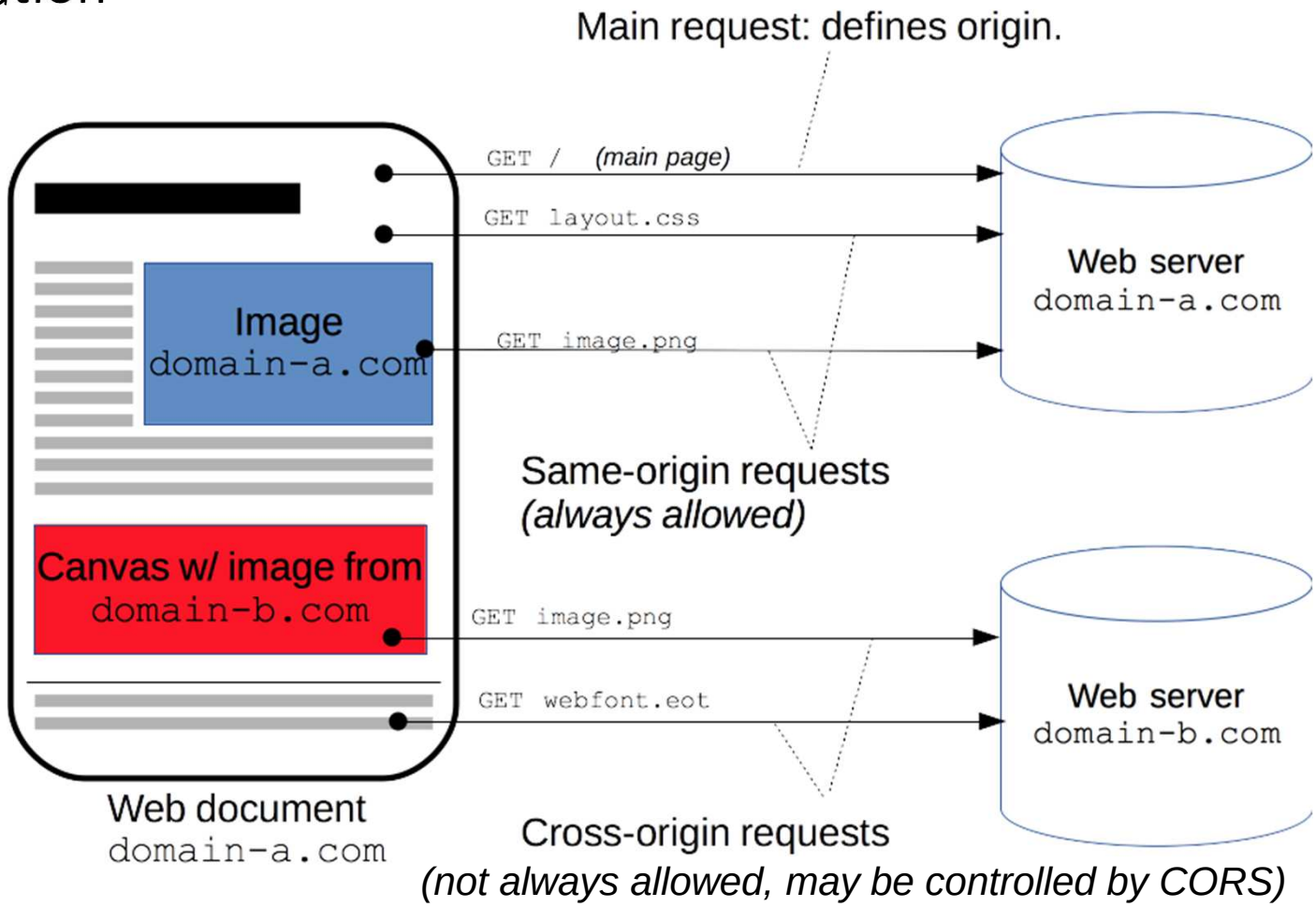
- **Cross-Origin Resource Sharing (CORS)**
  - A mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served

    wikipedia.org
    - CORS defines a way in which a browser and server can interact to determine whether or not it is safe to allow the cross-origin request
  - A mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin

    mozilla.org

SOP and CORS

- Ilustration



Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Systemy Webowe

- Architektura
- Podstawy HTTP
- SOP i CORS
- Podstawy HTML i CSS
- Klienckie aplikacje Webowe

Web systems

- **WWW (World Wide Web)**
  - The system of specially formatted, interlinked hypermedia documents, built on top of the Internet, supported (supplied) by special servers (Web servers) and accessed with use of Web clients (Web browsers)
    - Allows to locate and access/view multimedia-based documents and to execute web-based applications
    - Documents/resources (provided by web servers/applications) visible as web pages
  - Hypermedia
    - Multimedia with hyperlinks
      - Contain text, images, videos, animations, and more (including interactive components), interlinked with use of hyperlinks (or simply links)
  - HTML (HyperText Markup Language)
    - A markup language designed to specify the content and structure of web pages

Web documents

- **Web page**
  - In its simplest form, a web page is an HTML document
    - Web pages may contain text, images, videos, and other multimedia
- **Hyperlinks**
  - HTML documents contain hyperlinks
    - Hyperlinks can reference other web pages and/or its components (files, images, video, e-mail addresses, and more)
    - When a hyperlink is clicked a Web browser sends a query to Web server
    - Web server locates the requested web resource and sends it to the Web browser
  - Some form of hyperlink may be associated with applications
    - E.g. clicking the link in the form **mailto:*some_email_address***, loads default e-mail program
    - If a hyperlink references a resource (e.g. file) that the browser is incapable of displaying, the browser prepares to download it

Web systems

- **URI, URL, URN, and IRI**
  - **URI** (Uniform Resource Identifier)
    - A string that identifies unambiguously a resource in a namespace
      - separately defined naming scheme and set of syntax rules
    - Standard identifier of uniquely pointed resources (RFC2396, RFC3986)
      - Enables interaction with representations of the resource over a network using specific protocols
  - Definition/syntax

specific to scheme – hierarchical or nonhierarchical part

```
URI = scheme:[//[userinfo@][host][:port]]path[?query][#fragment]
```

- **scheme** – specifies a **concrete syntax** (identifier specification for that scheme) and **associated access protocol** (but not always)
- **userinfo** = username, = username[:password] - this element is deprecated
- **host** – DNS FQDN (or IP address) of server on which resource resides
- **port** – optional transport protocol port
- **path** – mandatory path leading to the resource in the server
- **querry** – not well defined (usually in the form: *attribute:value* pair(s))
- **fragment** - identifier providing direction to a "sub-resource" (e.g. doc. section)

Web systems

- **URI, URL, URN, and IRI**
  - **URL** (Uniform Resource Locator)
    - W3C: **URL is a type of URI** that essentially identifies a resource via a representation of its primary access mechanism (e.g. network location)
    - The unified resource addressing format used on the Internet and local area networks (RFC 1738)

```
URL = protocol://host[:port]path[?query][#fragment]
```

    - **protocol** – access protocol and/or type of resource – e.g.: ftp, http, https, file, irc, mailto, telnet, ws, …
    - **path** – optional path leading to the recourse in the server
      - no path causes return of default resource (defined in the resource server)
      - for security reasons the path location is typically a virtual directory.
      - web server translates the virtual directory into a real location on the server
    - URL of documents available with use of HTTP starts with **http://**
      - E.g.: **`http://www.company.com/mydir/myfile.htm`**

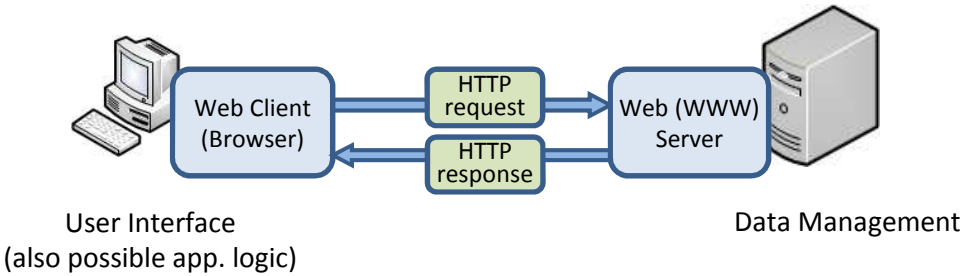
Web systems

- **URI, URL, URN, and IRI**
  - **IRI**
    - Extends URIs ASCII charset to the Universal Character Set UTF-8
      - RFC 3987
      - Percent-encoding
      - The weakness: vulnerability to phishing attacks
  - **URN** (Uniform Resource Name)
    - The unified format to identify resource in particular namespace
    - Originally defined to complement URLs as a form of URI
    - May not specify location or access mechanism
      - Using urn scheme
        - The form: **urn:<NID>:<NSS>**
        - NID – namespace identifier, NSS – namespace-specific string
    - An example is book identification
      - Its URN name is a unique ISBN number
    - Examples: **urn:ISBN:0-13-285620-4**
      urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66

Web systems

- **Web system architecture**
  - Web-based systems are often multitier systems (or n-tier systems) that divide functionality into separate tiers (i.e. logical groupings of functionality)

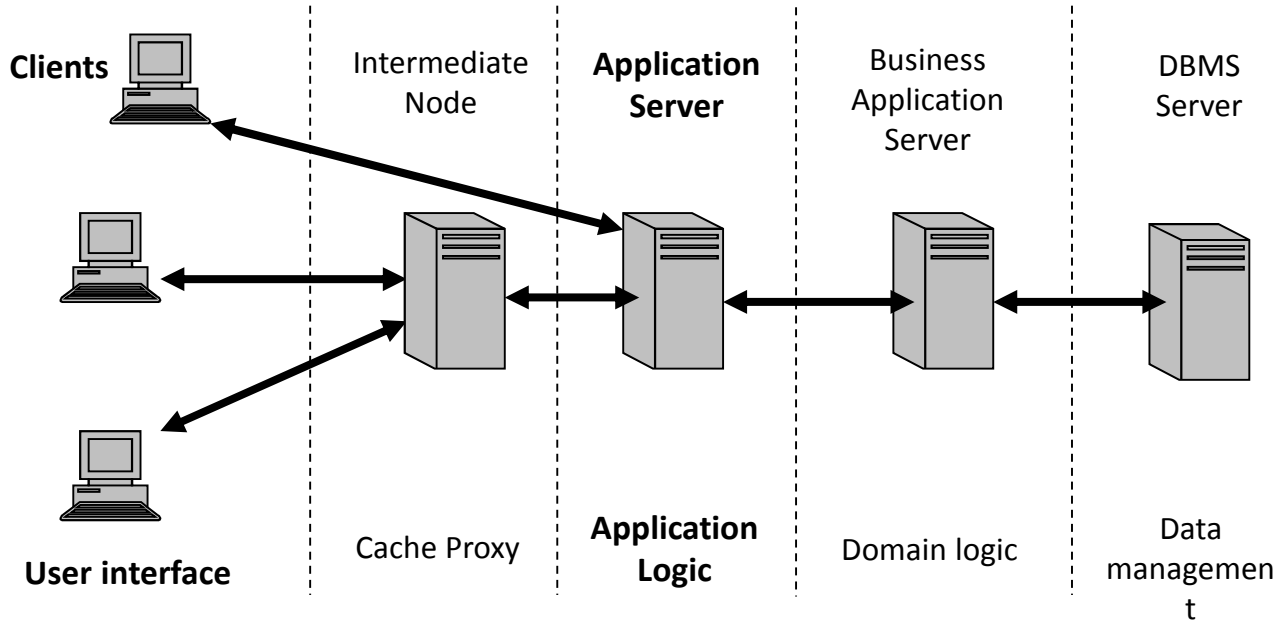
  - Client-server architecture



  - HTTP (Hypertext Transfer Protocol) – application layer communication protocol for Web systems

Web systems

- **Web system architecture**
  - N-tier (multi-tier) architecture

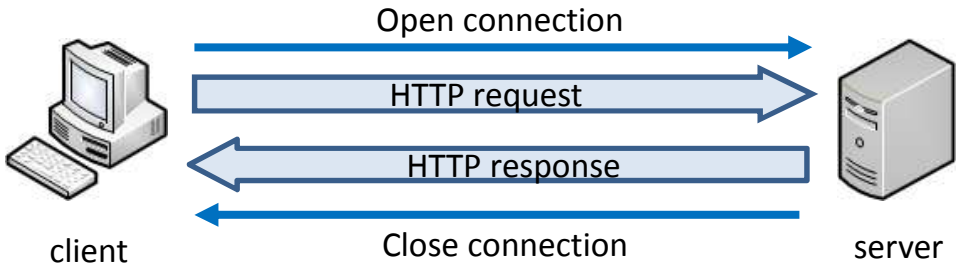


  - Vertically: network layer architecture comprised

WEB OPERATION: HTTP

Web operation

- **HTTP (Hypertext Transfer Protocol)**
  - Used for communication between Web entities and transfer of web documents/data
  - Application layer protocol of WWW services
    - Operates on top of TCP
    - Provides a standardized way of client-server communication according to **request – response** pattern



  - HTTP message
    - Single exchange of information between client and server
    - In request a **method** is specified

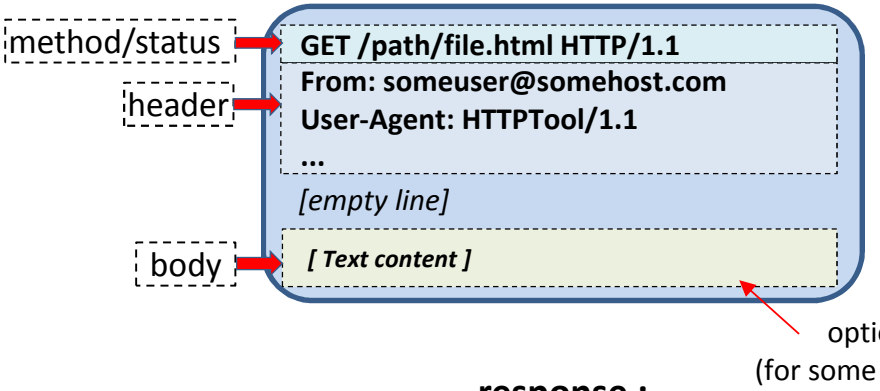
Web operation

- **HTTP (Hypertext Transfer Protocol)**
  - Lets to download resources (text, graphics), and helps when download other sources (eg. Streams) is necessary
  - HTTP versions:
    - HTTP 1.0 – almost not used,
    - HTTP 1.1 – widely used,
    - HTTP 2.0 – improved mechanisms – still not very often used
  - Classified as stateless protocols because it does not retain any information about previous transactions with the client
    - This can significantly reduce the load on the server, but is troublesome (e.g. when to remember the specific condition for the user who had already connect to the server is needed)
    - The most common solution to this problem is:
      - a mechanism of cookies on the client side
      - a mechanism of sessions on the server side

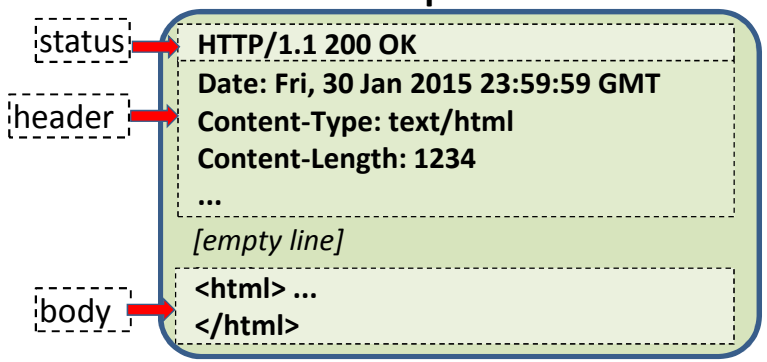
HTTP

- **HTTP message format**
  - Composed of **header** and **body**
    - Empty line after header
    - **HTTP method**
      - Define request type and server reaction
      - Resource path
    - Response status
    - Header fields
      - HTTP headers provide additional information about the data that are sent
      - Examples:
        - **User-Agent** - identifies the client (browser)
        - **Content-Type** – specify what format and code page is used in the document

HTTP request message (example: GET request):



response :



Web operation

- **HTTP Methods**
  - Mandatory
    - GET
      - request for specific resources - server returns data
    - HEAD
      - The same as GET but no body is sent
  - Optional
    - POST
      - Sending data to server to be used by it
        - e.g. sending data from HTML form
    - PUT
      - Sending a resource to server
    - DELETE
      - Removing a resource from server
    - OPTIONS, TRACE, CONNECT, PATCH
    - Optional methods may require authorization

Web operation

- **Response status**
  - **1xx** – informational (from HTTP 1.1)
    - Indicates a provisional response, consisting only of the Status-Line and optional headers, and terminated by an empty line
  - **2xx** – success
    - Indicates that the client's request was successfully received, understood, and accepted
    - The returned information is dependent on the method used in the request
  - **3xx** – redirection
    - Indicates that further action needs to be taken by the user agent in order to fulfill the request
    - The required action may be carried out by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD
  - **4xx** – client error
    - Intended for cases in which the client seems to have erred
    - The server should include an explanation of the error (except for HEAD request), and whether it is a temporary or permanent condition
    - User agents should display any included entity to the user
  - **5xx** – server error
    - Indicate cases in which the server is aware that it has erred or is incapable to perform the request
    - As for 4xx an explanation of the error should be included

Web operation

- **Response status code examples**

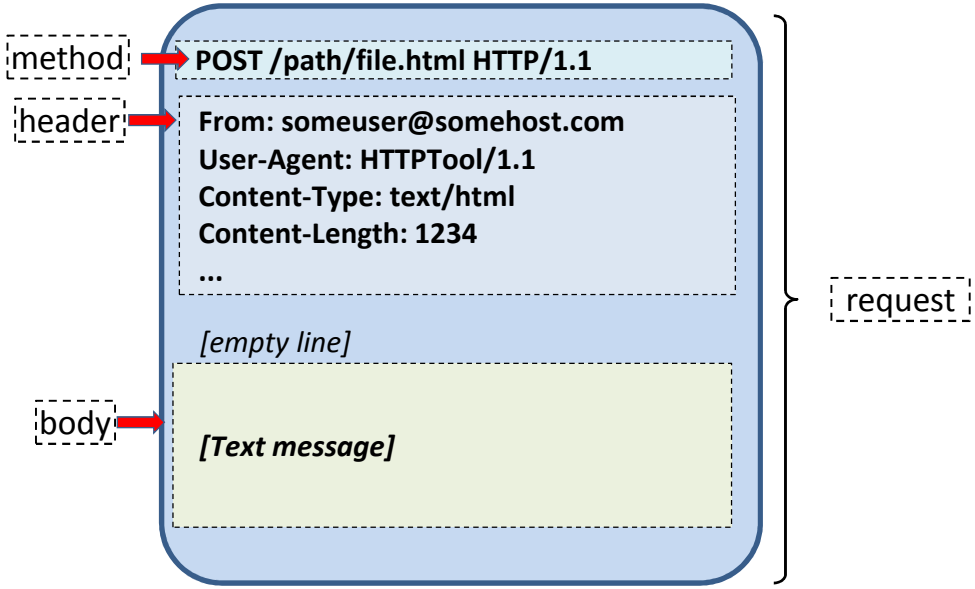
100 - Continue
101 - Switching Protocols
110 - Connection Timed Out
111 - Connection refused
200 - OK
201 - Created
202 - Accepted
203 - Non-Authoritative Information
204 - No content
205 - Reset Content
206 - Partial Content
300 - Multiple Choices
301 - Moved Permanently
302 - Found
303 - See Other
304 - Not Modified
305 - Use Proxy
307 - Temporary Redirect

400 - Bad Request
401 - Unauthorized
402 - Payment Required
403 - Forbidden
404 - Not Found
405 - Method Not Allowed
406 - Not Acceptable
407 - Proxy Authentication Required
408 - Request Timeout
409 - Conflict
410 - Gone
411 - Length required
412 - Precondition Failed
413 - Request Entity Too Large
414 - Request-URI Too Long
415 - Unsupported Media Type

416 - Requested Range Not Satisfiable
417 - Expectation Failed
500 - Internal Server Error
501 - Not Implemented
502 - Bad Gateway
503 - Service Unavailable
504 - Gateway Timeout
505 - HTTP Version Not Supported

Web operation

- **Sending the data**
  - POST request
    - A post request sends form data as part of the HTTP message
    - it hides the submitted data from the user by embedding it in an HTTP message



Web operation

- **Sending the data**
  - GET request
    - A get request appends data to the URL
    - A **name/value pair** is passed to the server with the name and the value separated by an equals sign (**=**).
    - The **?** separates the query strings from the rest of the URL in a request.
    - Each pair is separated by an ampersand (**&**)

```
http://www.mysite.pl/?lang=pl&page=venue
```

    - The server uses passed data to retrieve an appropriate resource
    - The server then sends a response to the client.
    - The request may be initiated by submitting an HTML form, or by typing the URL directly.

Web operation

- **Header fields**
  - HTTP headers provide additional information about the data that being sent or received
  - Example header fields (method GET)

GET	/http/ HTTP/1.1
Host:	tools.ietf.org
User-Agent:	Mozilla/5.0 (Windows NT 6.2; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept:	text/html ; q=0.8, text/plain ; q=0.2 , text/x-dvi
Accept-Language:	pl,en-us;q=0.7,de;q=0.3
Accept-Encoding:	gzip, deflate
Connection:	keep-alive
CRLF	

Web operation

- **Header fields**
  - Example header fields (received)

HTTP 1.1.	200OK
Date:	Mon, 15 Apr 2019 17:31:26 GMT
Server:	Apache/2.2.21 (Debian)
Content-Encoding:	gzip
Content-Length:	8461
Keep-Alive:	timeout=5, max=100
Connection:	Keep-Alive
Content-Type	text/html; charset=UTF-8
CRLF	

```
<html>
 <head> <title>Web Page Title</title> </head>
 <body bgcolor="blue"> <p>My text on the page</p><br><p>Next line here</p> </body>
</html>
```

Web operation

- **Header fields** (selected examples)
  - General
    - **Date** - the date on the server.
    - **Pragma** - used to include implementation-specific directives that might apply to any recipient along the request/response chain
      - Optional behavior e.g.:
        `pragma-directive  = "no-cache"`
    - **Cache-Control** - used to specify directives that must be obeyed by all caching mechanisms along the request/response chain

  - Request Header
    - **User-Agent** - identifies browser.
    - **Accept** - is used to determine the list of accepted by your browser MIME type of the document, and optionally the hierarchy of each type

Web operation

- **MIME**
  - The **MIME** (Multipurpose Internet Mail Extensions) **type** specified in Content-Type field
  - Specifies the MIME type of the content that the server is transmitting to the browser.
    - The MIME standard specifies data formats, which programs can use to interpret data correctly.
  - Examples
    - **text/html** - the content is HTML document
    - **text/xml** - the content is document described by XML language
    - **image/jpeg** - the content is a JPEG image.
      - When the browser receives this MIME type, it attempts to display the image
    - **video/mpeg** – MPEG-1 standard video file
    - **application/…** - program specific content

Web operation

- **Header fields** (selected examples)
  - Request Header
    - **Accept-charset** – specifies the preferred browser text encoding
    - **Accept-Language** – sets the language in which the browser user wishes to read pages.
    - **Accept-Encoding** – lists acceptable encodings of the content (compressed content)
    - **Authorization** – authentication credentials for HTTP authentication
    - **Host** – specifies the domain name of the server and optionally TCP port
      - Necessary for multiple virtual servers on a single IP address.
      - Mandatory in HTTP 1.1.
    - **Cookie** – cookies set on your browser (previously sent by server)

Web operation

- **Header fields** (selected examples)
  - Response Header
    - **Server** – header identifies the server and used software.
    - **Set-cookie** – header instructs the browser to set specific cookies.
    - **Allow** – give a list of acceptable HTTP methods for specified resource
    - **Expires** – the date beyond which the document will be invalid.
    - **Connection** – control options for the current connection – determines when to close connection
    - **Transfer-Encoding** – specifies the encoding of the document.
    - **Content-Type** – the server header tells the browser what format and code page is sent in the document
    - **Last-Modified –** the last modified date for the requested object
    - **Content-Length** – the length of body in octets (8-bit bytes)

SOP / CORS

SOP i CORS

- **Same-Origin Policy (SOP)**
  - The same-origin policy is a critical security mechanism that restricts how a document or script loaded from one origin can interact with a resource from another origin. It helps isolate potentially malicious documents, reducing possible attack vectors.

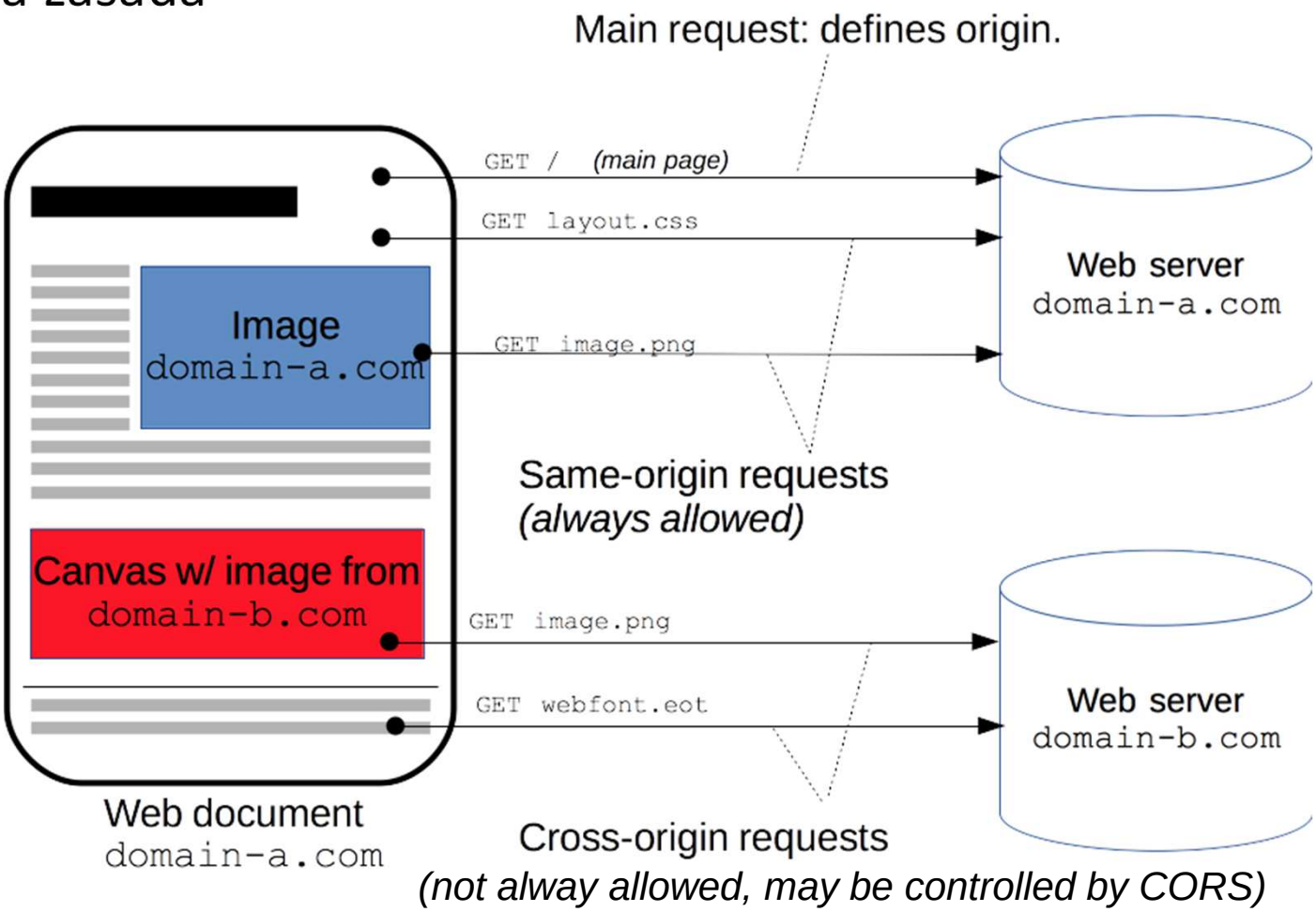
    mozilla.org
  - In computing, the same-origin policy is an important concept in the web application security model. Under the policy, a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same origin

    wikipedia.org (EN)

  - **Jest to mechanizm bezpieczeństwa implementowany w przeglądarkach, za pomocą którego kontrolowana jest możliwość lub jej brak wykorzystania zasobów (w tym także wykonywalnych skryptów) z innych źródeł niż pierwotne źródło uzyskiwanej treści**

SOP i CORS

- Ogólna zasada



SOP i CORS

- **Origin** (pochodzenie, "domena")
  - Jest zdefiniowany jako zestaw:
    URI scheme, host name, oraz ***port number***
  - Dwa URLe mają ten sam ***origin*** jeśli te elementy są takie same
  - Przykład porównania ***origin*** dla adresu (URL):
    http://www.example.com/dir/page.html

Compared URL	Same origin	Reason
http://www.example.com/dir/page2.html	YES	Same protocol, host and port
http://www.example.com/dir2/other.html	YES	Same protocol, host and port
http://username:password@www.example.com/dir2/other.html	YES	Same protocol, host and port
http://www.example.com:81/dir/other.html	NO	Same protocol and host but different port (80 is default)
https://www.example.com/dir/other.html	NO	Different protocol
http://en.example.com/dir/other.html	NO	Different host
http://example.com/dir/other.html	NO	Different host (exact match required)
http://v2.www.example.com/dir/other.html	NO	Different host (exact match required)
http://www.example.com:80/dir/other.html	Depends	Port 80 explicitly. Depends on implementation in browser.

SOP i CORS

- **Zasada działania mechanizmu**
  - SOP zabezpiecza przed niedozwolonymi operacjami i zasobami
  - SOP kontroluje interakcje między dwoma różnymi źródłami, na przykład gdy używany jest jakiś element HTML (np. <img…> - obrazek z innego źródła) lub XMLHttpRequest
    - Skrypty wykonywane ze stron z opcją about: blank lub javascript:URL dziedziczą pochodzenie dokumentu zawierającego ten adres URL, (gdyż nie zawierają info. o serwerze źródłowym)
  - 3 kategorie interakcji
    - **Cross-origin *writes*** – typowo są dozwolone
      - Przykłady: linki, przekierowania (redirects), i nadsyłanie danych formularzy (form submissions)
      - Niektóre żądania HTTP wymagają jednak tzw. ***preflight***
    - **Cross-origin *embedding*** – typowo są dozwolone
      - Czyli osadzanie zasobów na stronie (np. obrazków)
    - **Cross-origin *reads*** – typowo są niedozwolone
      - Np. ładowanie obrazka do canvas za pomocą skryptu

SOP i CORS

- **Zasada działania mechanizmu**
  - Przykładowe zasoby, które mogą być osadzone między źródł.
    - Obrazy wyświetlane poprzez `<img>`
      - Ale czytanie (np. ładownie obr. do *canvas* za pomocą JavaScript) zabronione
    - Media grane poprzez `<video>` oraz `<audio>`
    - JavaScript z użyciem `<script src="…"></script>`
    - CSS z użyciem `<link rel="stylesheet" href="…">`
      - Cross-origin CSS wymaga odpowieniego nagłówka (pola) Content-Type
    - Czcionki stosowane z użyciem `@font-face`
      - Niektóre przeglądarki zezwalają na cross-origin fonts, inne tylko same-origin
    - Wszystko umieszczone w `<iframe>`
  - Dla bezpieczeństwa, przeglądarki zabraniają żądań HTTP cross-origin wywoływanych z skryptów
  - **Żądania AJAX są domyślnie zabronione przez SOP**
  - Aby zezwolić na dostęp cross-origin trzeba użyć **CORS**

SOP i CORS

- **Cross-Origin Resource Sharing (CORS)**
  - A mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served

    wikipedia.org
  - A mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin

    mozilla.org
  - **Mechanizm umożliwiający nadanie uprawnień dostępu do zasobów z innego źródła (wykonanie żądań cross-origin) niż zasób z pierwotnego źródła**
    - Używa tzw. ***preflighted requests*** (żadań wstępnych/inicjujących)

SOP i CORS

- **Simple request**
  - Wybrane warunki konieczne definiujące Simple request:
    - Metody HTTP:
      - GET, HEAD lub POST
    - Nagłówki HTTP ustawiane automatycznie przez client (np. User-Agent)
    - Nagłówki, które można ustawić manualnie ograniczone do:
      - Accept,
      - Accept-Language
      - Content-Language
      - Content-Type
    - Ale dozwolone wartości Content-Type
      - text/plain
      - application/x-www-form-urlencoded
      - multipart/form-data
    - Proste żądanie:

SOP i CORS

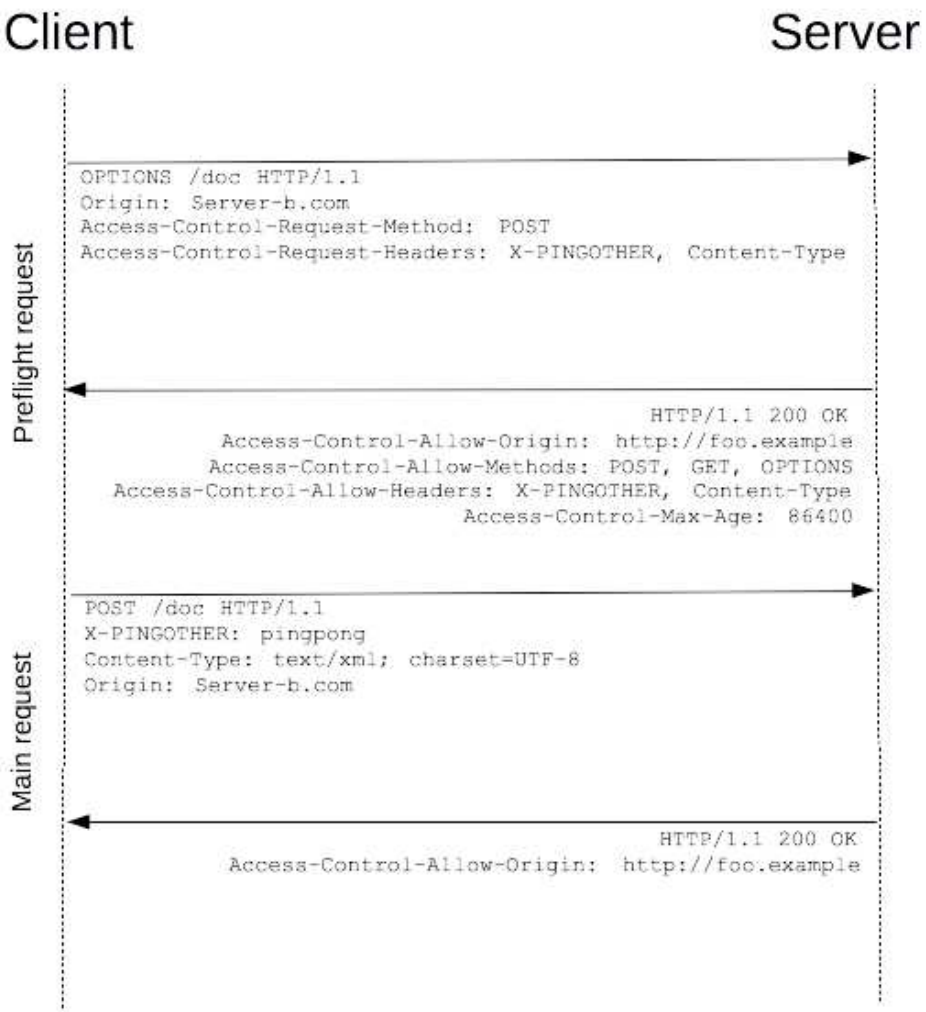
- **Non-simple requests**
  - Wybrane warunki definiujące *Preflighted request*:
    - Metody HTTP:
      - PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH
    - Inne wartości Content-Type niż dla *simple requests*
    - Zbiór nagłówków HTTP, które można ustawić manualnie – inne niż dla *simple requests*
  - Mechanizm
    - Nowoczesne przeglądarki obsługują CORS z dodatkowym żądaniem ***preflight*** ("kontroli wstępnej") w celu ustalenia, czy mają uprawnienia do wykonania akcji
      - Klient wysyła dodatkowe żądanie opcji OPTIONS specyfikując metodę i nagłówki
      - Serwer wysyła odpowiedź dla jakich warunków akceptuje akcję

```
Origin: http://www.example.com
Access-Control-Allow-Method:…
Access-Control-Allow-Headers:…
```

```
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Methods: PUT, DELETE, …
Access-Control-Allow-Headers: …
Access-Control-Max-Age: …
```

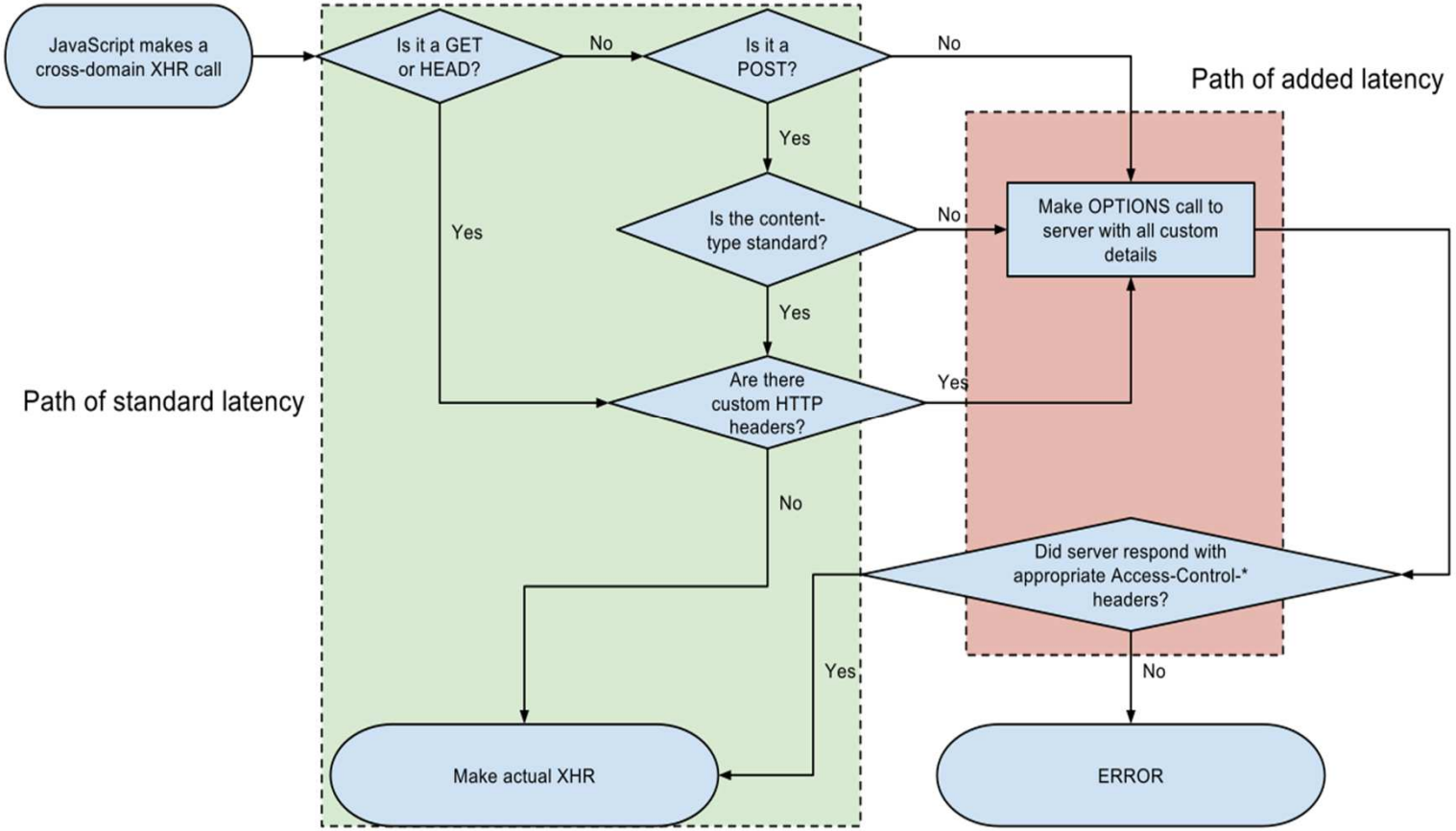
SOP and CORS

- **Preflighted request (kont.)**
  - Ilustracja żądania POST z zawartością XML
    - i dodatkowo z nagłówkiem użytkownika X-PINGOTHER



SOP i CORS

- **Procedura CORS dla żądań AJAX/AJAJ**



SOP and CORS

- **Wsparcie *preflighted requests* w WCF**
  - Dodatkowa klasa w projekcie Global.asax
    - Zawiera kod wykonywany przy obsłudze żądania usługi WCF
  - Opcja VStudio: *Add → New Item → Web\General\Global application class*

```
public class Global : System.Web.HttpApplication  {
 protected void Application_Start(object sender, EventArgs e)  { }
 protected void Session_Start(object sender, EventArgs e)  { }
 protected void Application_BeginRequest(object sender, EventArgs e)  {
  HttpContext.Current.Response.AddHeader("Access-Control-Allow-Origin", "*");
  if (HttpContext.Current.Request.HttpMethod == "OPTIONS")    {
   HttpContext.Current.Response.AddHeader("Access-Control-Allow-Methods","POST, PUT, DELETE");
   HttpContext.Current.Response.AddHeader("Access-Control-Allow-Headers","Content-Type, Accept");
   HttpContext.Current.Response.AddHeader("Access-Control-Max-Age", "3600");
   HttpContext.Current.Response.End();
  }
 }
 protected void Application_AuthenticateRequest(object sender, EventArgs e)  { }
 protected void Application_Error(object sender, EventArgs e)  { }
 protected void Session_End(object sender, EventArgs e)  { }
 protected void Application_End(object sender, EventArgs e)  { }
}
```

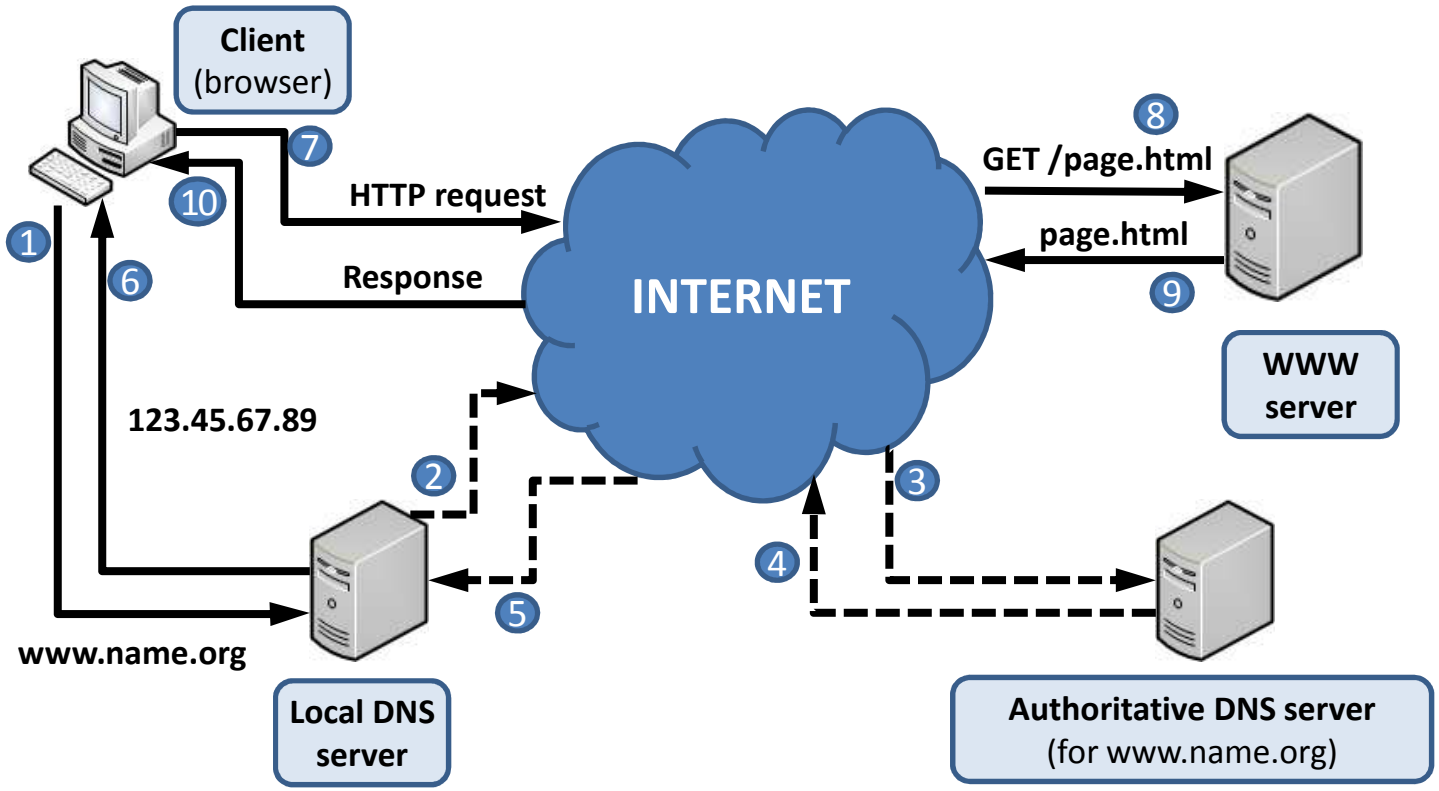
headers for CORS support

- methods
- headers
- timeout

WEB OPERATION: TRANSACTIONS

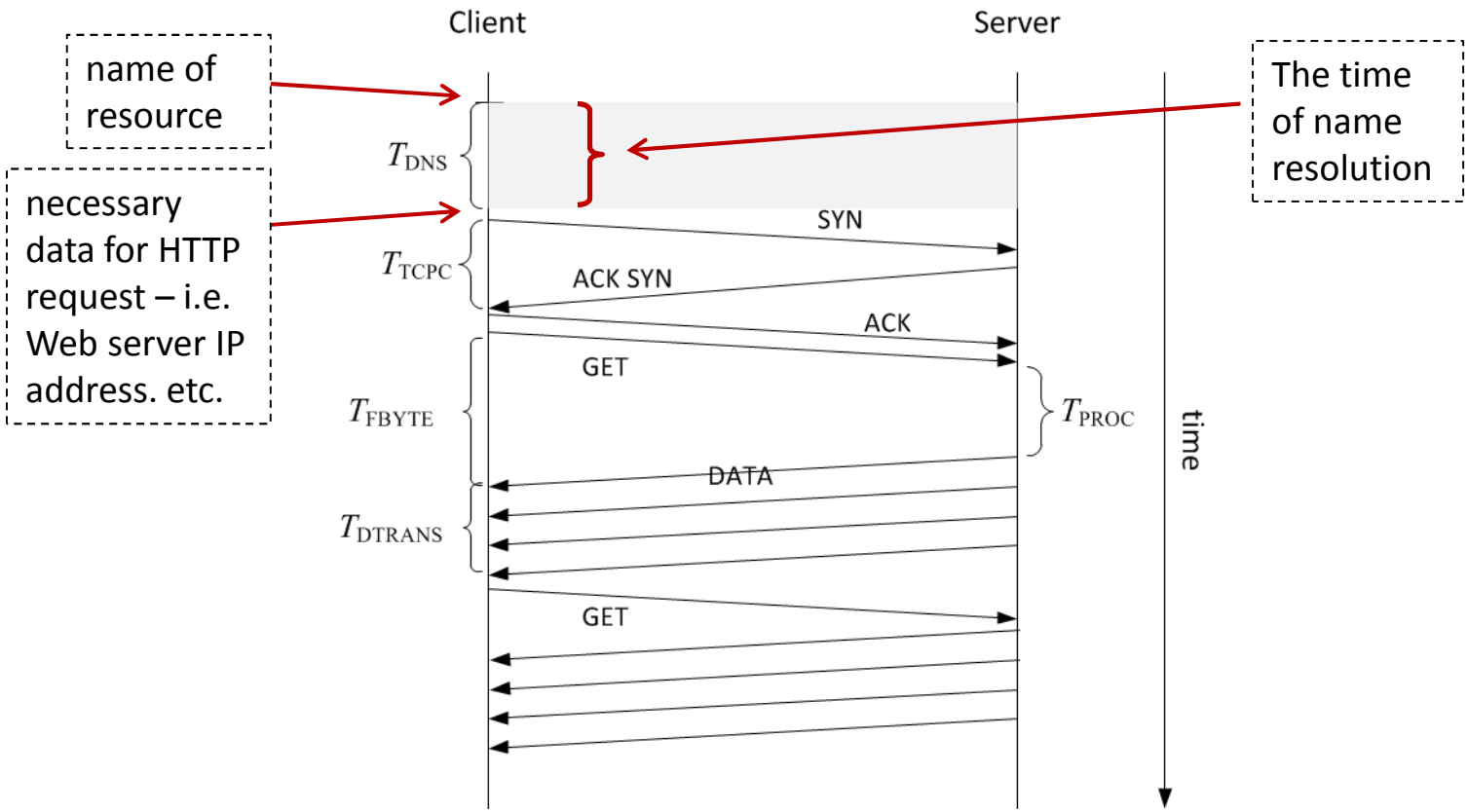
Web transaction

- **Simple/basic web content retrieval scenario**



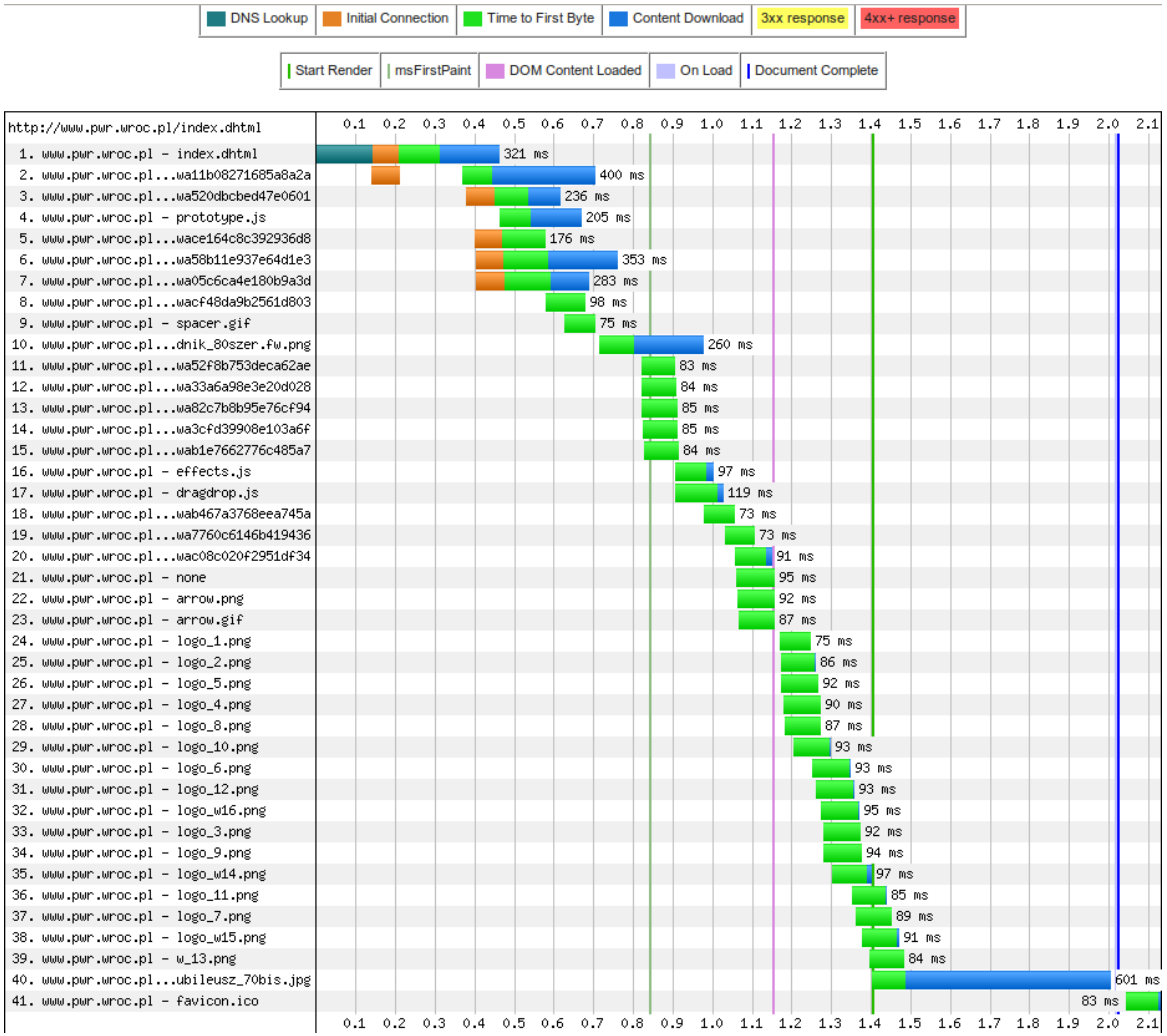
Web transaction

- **Basic HTTP transaction**



Web transaction

- **Resource Waterfall**

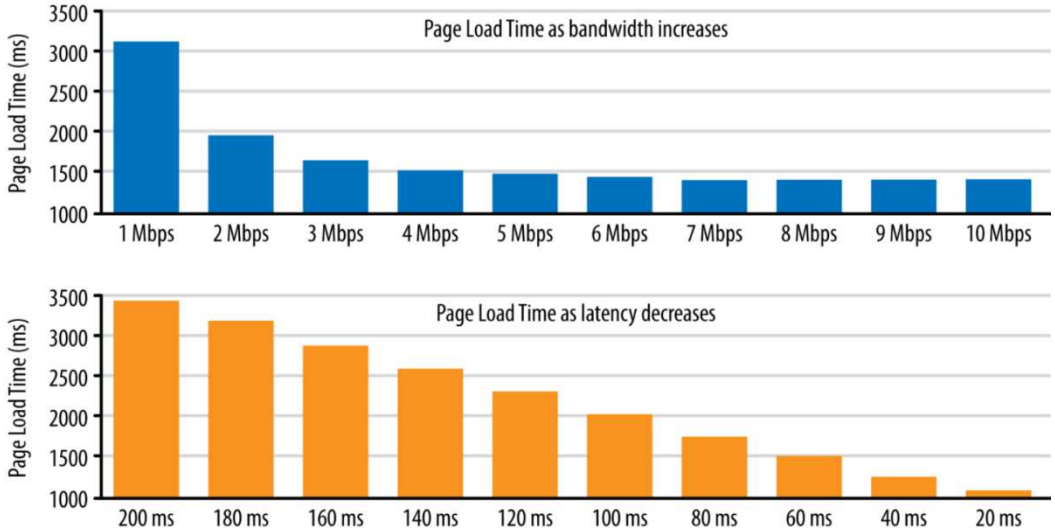


Web transaction

- **Page load time**
  - Considerable simplification:

$$T_{PRT} = \frac{S_{\mathrm{P}}}{B_{\min}} + (RTT \cdot Q) + T_{\mathrm{SPROC}} + T_{\mathrm{CPROC}}$$

    - S_{P} – page size, $B_{\min}$ – minimum bandwidth, Q – num. of queries, T_{xPROC} – server and client overall processing time
  - Latency and bandwidth impact

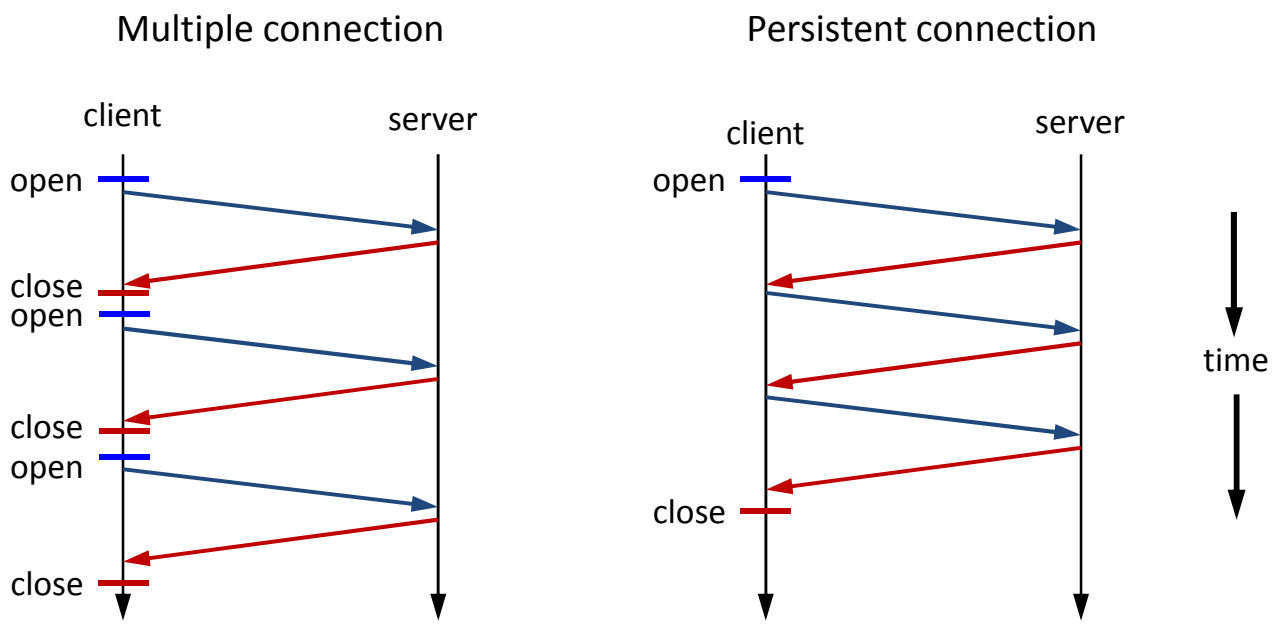


Web transaction

- **Time To First Byte (TTFB)**
  - TTFB measures the duration from the user or client making an HTTP request to the first byte of the page being received by the client's browser
  - Used as one of important indication of the responsiveness of a webserver or other network resource.
- **Page Load time**
  - The time it takes for a webpage to be loaded and usable
  - Many aspects affect the speed of page load
    - Server and client processing
    - Caching
    - Encryption
    - Transaction type
    - Interactivity
    - …

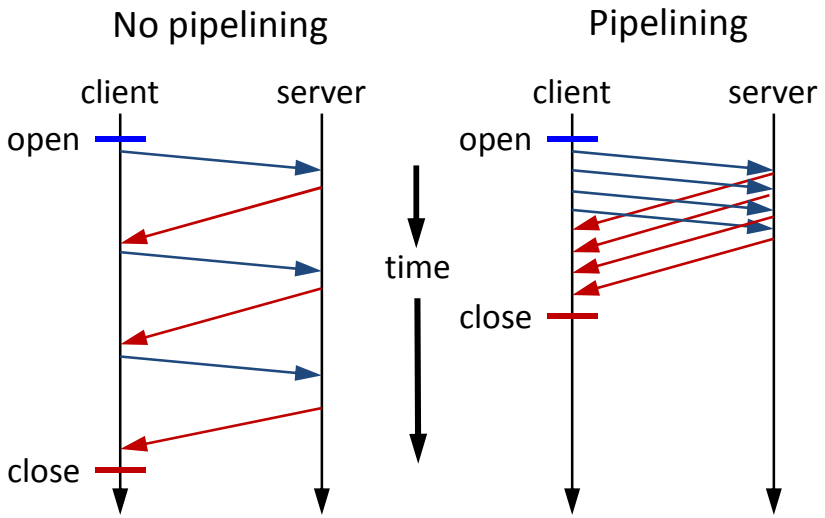
Web transaction

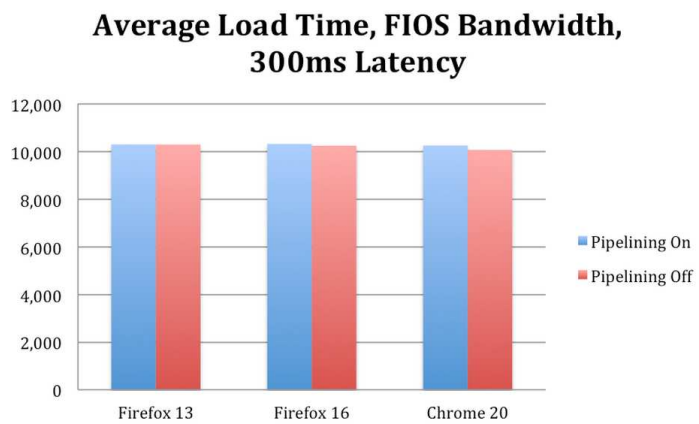
- **HTTP persistent connection / HTTP keep-alive / HTTP connection reuse**
  - Introduced in HTTP 1.1
  - Sending and receiving multiple HTTP requests/responses in single TCP connection



Web transaction

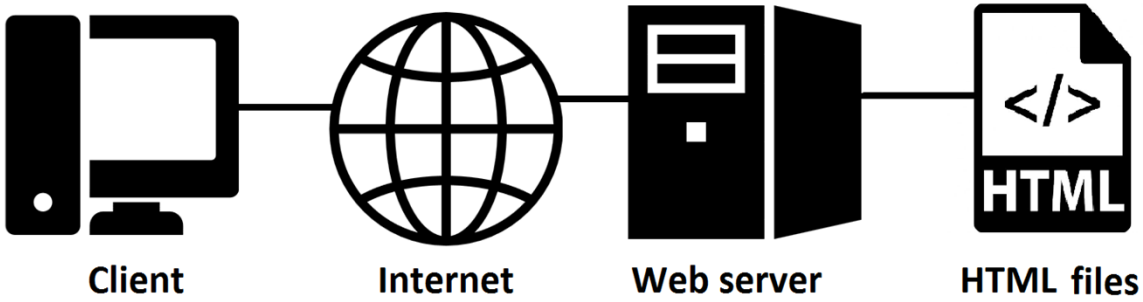
- **HTTP pipelining**
  - Introduced in HTTP 1.1
  - Multiple HTTP requests are sent on a single TCP connection without waiting for the corresponding responses
    - Limitation of HTTP 1.1: the server must send its responses in the same order that the requests were received
    - Not significant impact on performance

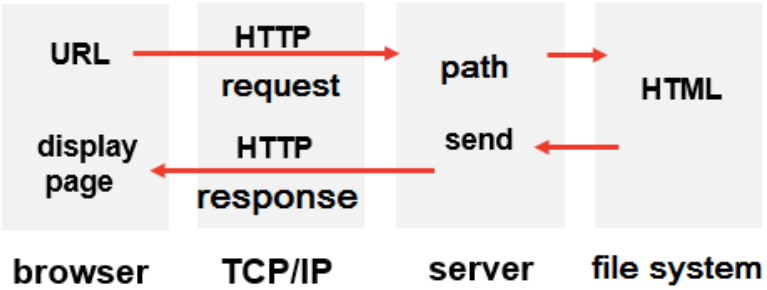




Web transaction

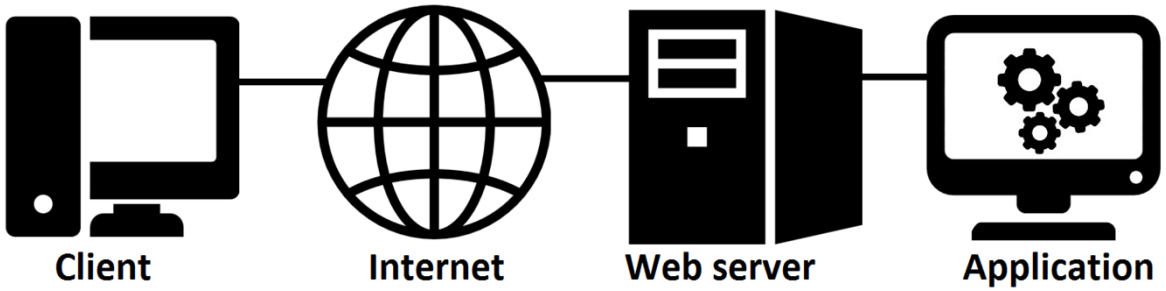
- **Static Transaction**

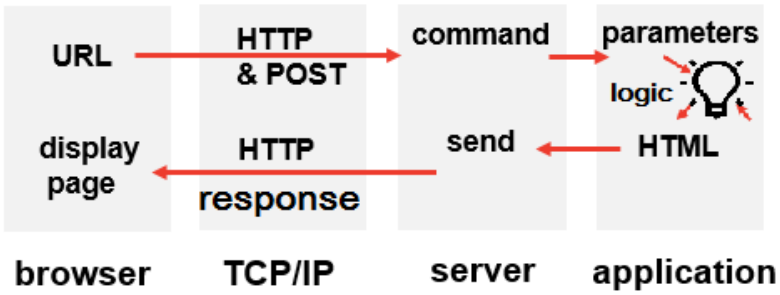




Web transaction

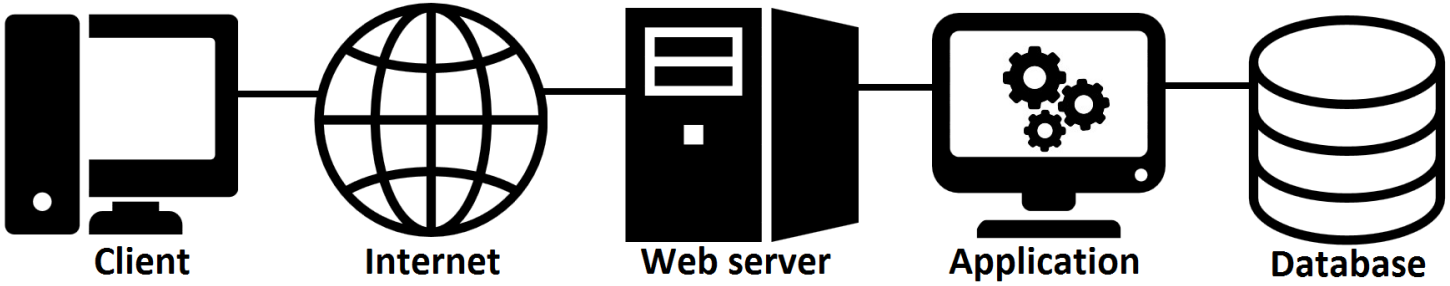
- **Dynamic Transaction**

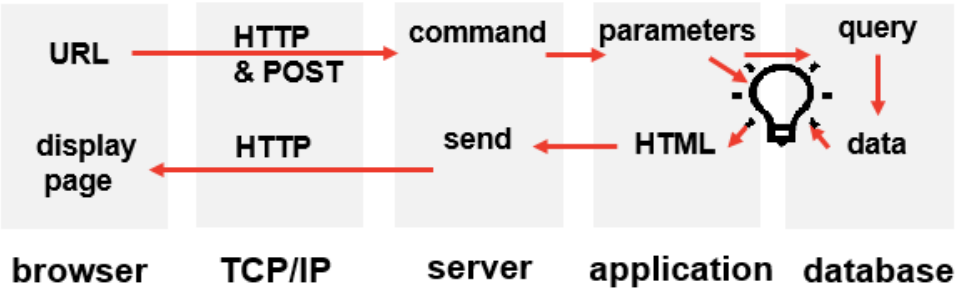




Web transaction

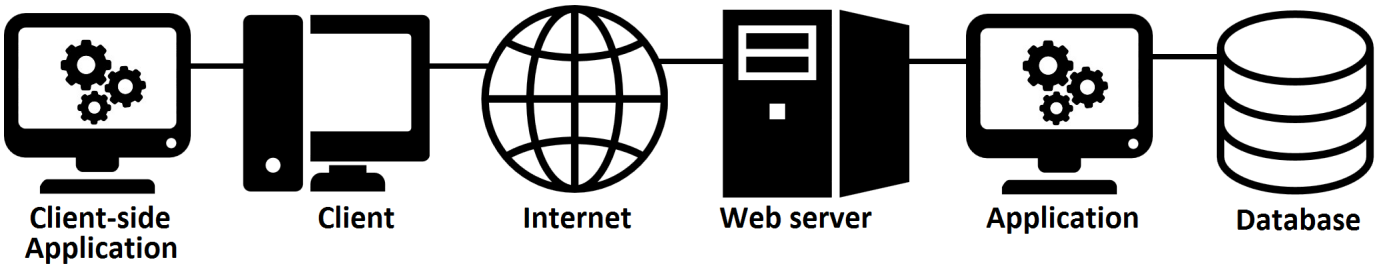
- **Database Transaction**

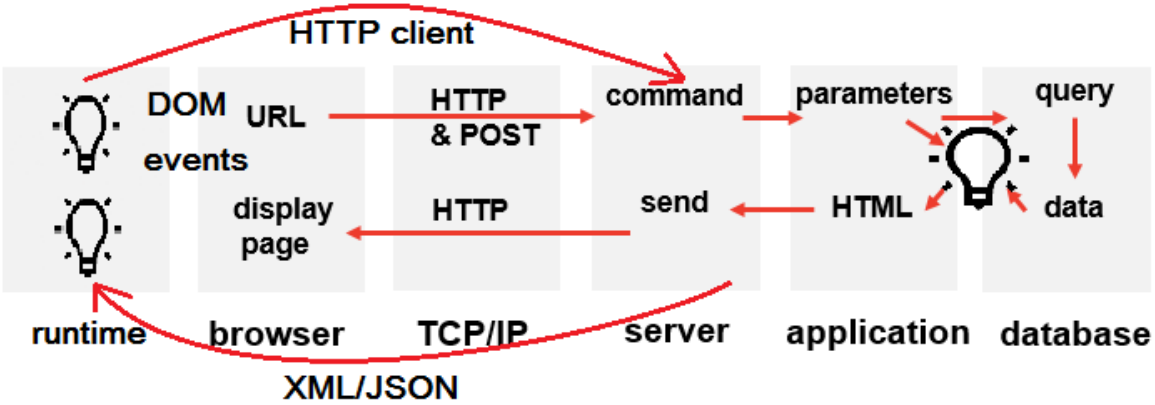




Web transaction

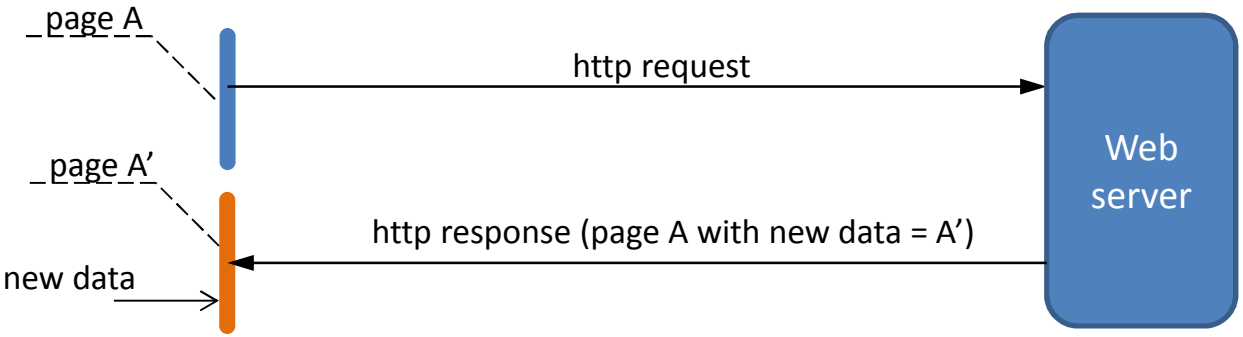
- **Rich Client Asynchronous Transaction**



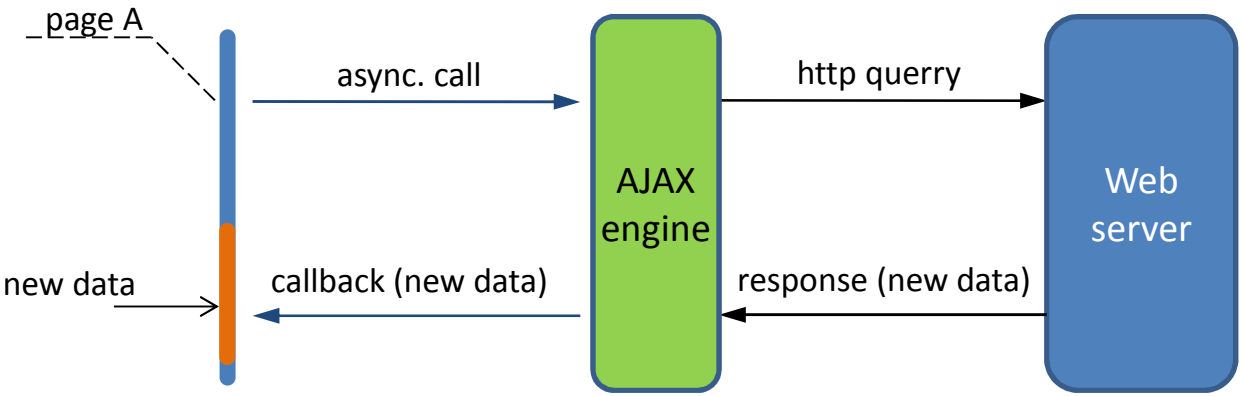


Web transaction

- **Tradycyjna transakcja webowa**

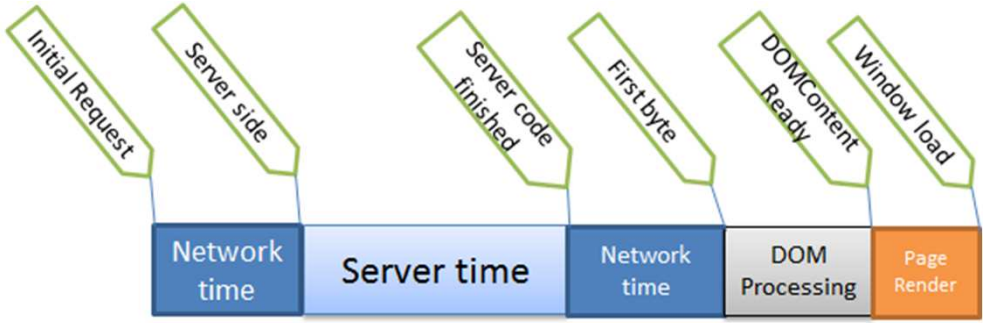


- **AJAX**

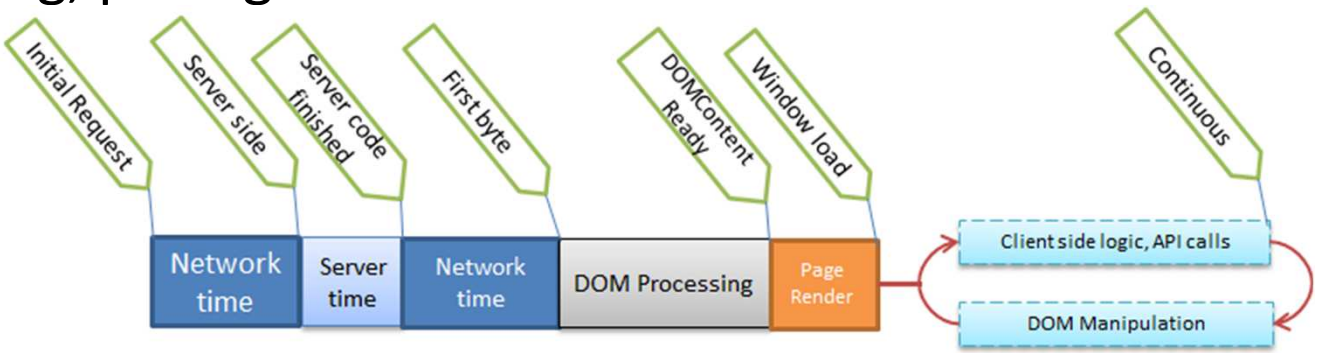


Web transaction

- **Traditional (single) web page load and rendering**



- **Rich application page**
  - DOM is continuously manipulated as part of user interaction or monitoring, polling and other events.



HTML & CSS

Web documents

- **HTML (HyperText Markup Language)**
  - A special type of markup language designed to specify the content and structure of web documents in a portable manner.
    - Markups / tags:
      - Featuring the parts of text giving them a specific meaning
      - Special character sequences point where begins and ends the tag instruction
      - Example:

        ** ** Here we have ****some**** text. ** **

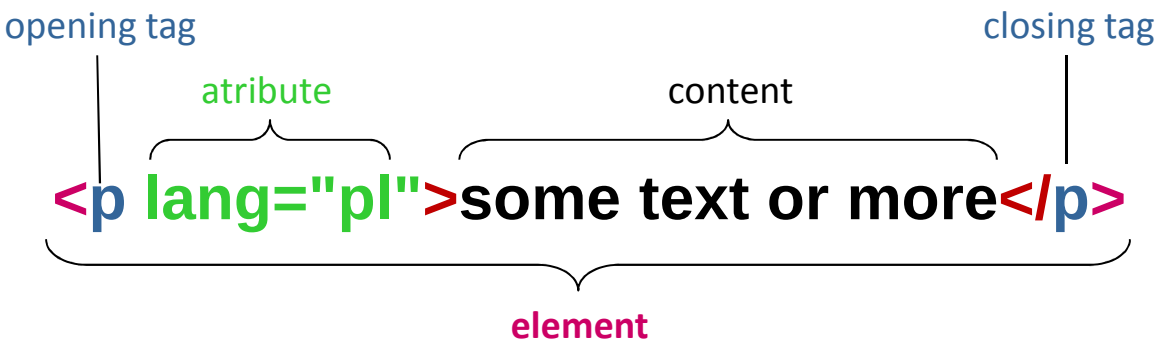
        Here we have **some** text.
  - May 2000: Last closed standard: **HTML 4.01**
  - **XHTML** (Extensible HyperText Markup Language) – a "stricter" version of HTML based on **XML** – still used today.
  - **HTML5** is the emerging version of HTML still under development

HTML basics

- **HTML5**
  - Enables to create content that will render appropriately wide range of devices connected to the Internet
    - desktop computers, notebooks, smartphones, tablets, …
  - Includes new techniques of accessing media and content presentation, and modern service implementation styles
    - Assisted by CSS and scripting languages (such as JavaScript) technologies
- **Support for HTML**
  - Oct 2014: HTML5 gains W3C status "recommended"
  - Dec 2017: HTML 5.2 gains W3C status "recommended„
  - HTML 5.3 – draft version
  - Support for HTML5 features (as well as CSS3 and JavaScript) varies by browser
    - The HTML5 Test website (http://html5test.com/) scores browsers based on its support for the latest features of standards

HTML basics

- **HTML structure**
  - Element
    - Building block od HTML docs
    - A part of content (e.g. a string of characters) between opening and closing tags



    - Define structure as well as format of the visible content
      - A weakness partially repaired by CSS
    - Can be nested

HTML basics

- **HTML structure**
  - Main elements
    - **<html>**
    - **<head>**
    - **<body>**
  - Variety of html tags and rules for using them
  - For partitioning of the content the grouping elements are used
    - ** **
    - HTML5 new elements
      - header, footer, main, nav, aside, article, section

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web Page Title</title>
  </head>
  <body bgcolor="blue">
    <div id="fline">
      <p>My text on the page</p>
    </div>
    <br>
    <p>Next line here</p>
    <!-- the comment -->
  </body>
</html>
```

HTML5 declaration

root element

nesting of elements within root element

comment

HTML basics

- **Very basic set of HTML tags**

Tag	Description
<html>	Defines an HTML document
<head>	Defines information about the document
<body>	Defines the document's body
<p>	Defines a paragraph
<h1> to <h6>	Defines HTML headings
<a>	Defines a hyperlink
<img>	Defines an image
** **	Inserts a single line break
<b>	Defines bold text
****	Defines emphasized text
****	Defines important text
<ul>	Defines an unordered list
<ol>	Defines an ordered list
<li>	Defines a list item
<table>	Defines a table
<tr>	Defines a row in a table
<td>	Defines a cell in a table
<form>	Defines an HTML form for user input
** **	Defines a section in a document
****	Defines a section in a document
<!-- ... -->	Defines a comment

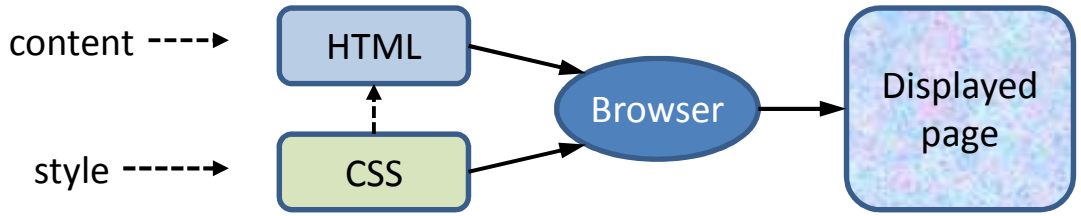
HTML basics

- **Attributes**
  - Additional information for elements
  - Pair: **name="value"**
    - In opening tag
    - Various meaning
  - Example

    <body **bgcolor="blue"**> … </body>
  - Hyperlinks defined with use of href attribute
    - **<a** href="http://some.url.com"**>** url name (text)**</a>**
  - **id** – element unique identifier
    - <p **id="my-id"**> … </p>
  - **class** – to specify a class of elements
    - <p id="my-id" **class="bold-class-name"**> … </p>
    - <h2 id="other-id" **class="bold-class-name italic-class"**> … </h2>

CSS basics

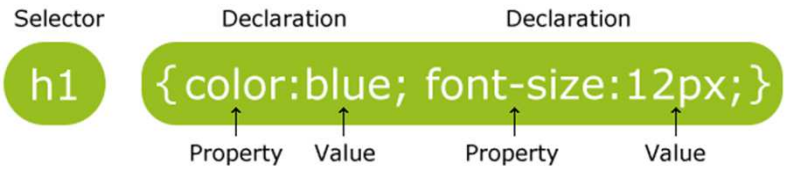
- **Cascading Style Sheets (CSS)**
  - HTML provides some capabilities for controlling a document's presentation, but it's better not to mix presentation with content
  - Cascading Style Sheets (CSS) **are preferred to specify the presentation, or styling of elements on a web page** (e.g. fonts, spacing, sizes, colors, positioning)



  - CSS was designed to style portable web pages independently of their content and structure
  - By separating page styling from page content and structure, you can easily change the look and feel of the pages on an entire website, or a portion of a website, simply by swapping out one style sheet for another
  - **CSS3** is the current version of CSS under development

CSS basics

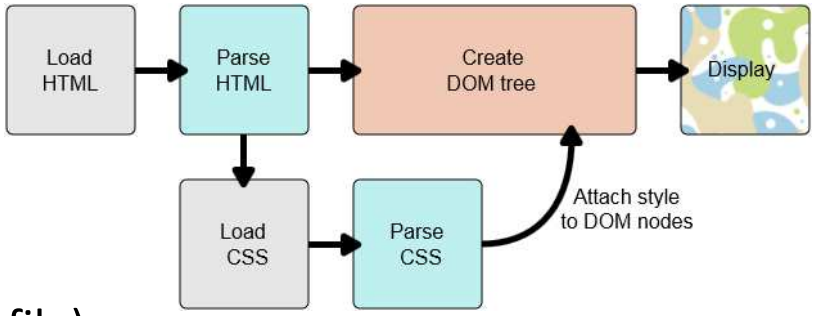
- **Syntax**



- **Selectors**
  - html element: **p {color:red;}**
  - element id: **#my-id {color:green;}**
  - element class: **.my-class {color:red;}**
    - Can be mixed: **p.my-class {color:blue;}**
  - depency on where declaration are placed
- **Pseudo-classes**
  - specifies a special state of the selected element(s)

    *selector***:pseudo-class** { *property*: *value*; }
  - Examples
    - **a:hover** – cursor over link – e.g.: a:hover {color:blue}
    - **a:visited** – visited link – e.g.: a:visited {color:green}

CSS basics



- **Applying CSS to HTML**
  - External style sheet (external .css file)
    - In section <head>

```
<link rel="stylesheet" type="text/css" href="file_name.css">
```

  - Internal style sheet
    - In section <head>

```
<style>
  selector {property: value; }
  ...
</style>
```

  - Inline style
    - In given element in html file

```
<p style="color:blue; margin-left:30px;">Any text</h1>
```

CSS basics

- **Example**

HTML file

```
...
<head>
  ...
  <link rel="stylesheet" type="text/css" media="screen" href="css/styles.css">
  <style>
   h2 {font-size:34px; color:#7f0000; font-weight:normal; font-family:'Verdana'}
  </style>
</head>
<body>
  <h2 id="id1" class="id2"> Title of my page </h2>
  <p class="id2"> This is my text content </p>
  <p style="font-weight:bold; color:red"> This is bold text in red</p>
  ...
</body>
...
```

CSS file

```
html { margin: auto; padding:0; }
body { font: 14px Arial, Verdana; color:#00cc00; background:#eaeaea; }
a { text-decoration: none; color: #204a88; font-weight: bold; }
a:hover { text-decoration: underline; }
.id2 {text-decoration:underline; color:#44ccbb}
p.id2 {text-decoration:none; color:#00cc44}
...
```

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Systemy Webowe (cz.2)

- ...
- Klienckie aplikacje Webowe
  - ✓ DOM
  - ✓ Javascript
  - ✓ jQuery
  - ✓ AJAX (+ REST)

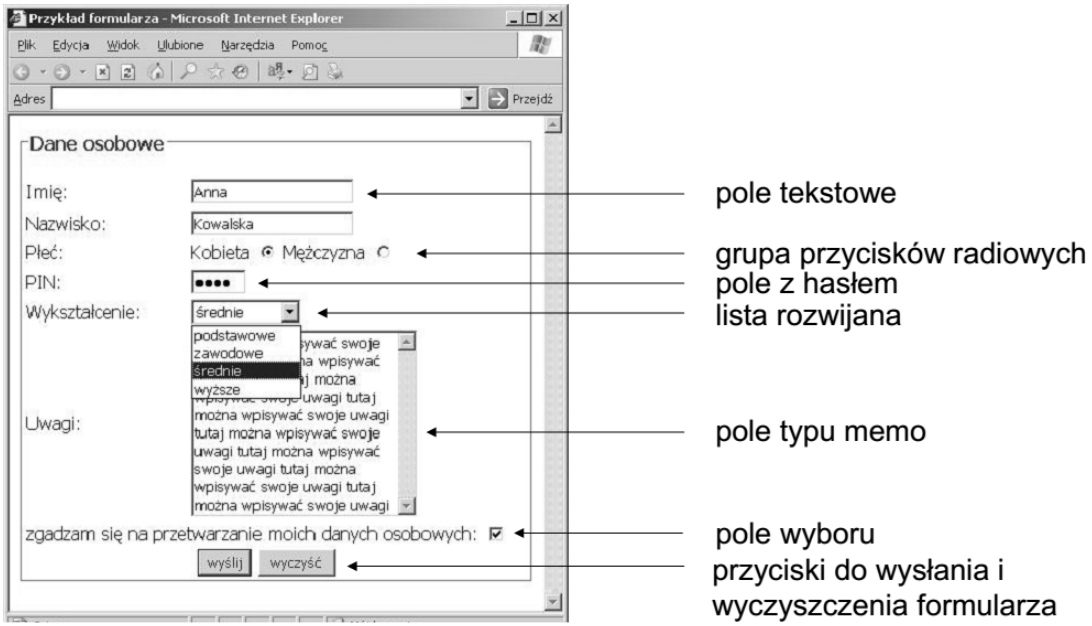
Web applications

- **Web page / Hypermedia**
  - Dokumenty hipertekstowe były genezą sieci World Wide Web
    - Najpierw tekst (plain text) z obsługą hiperłączy
    - Następnie dodatkowe zasoby multimedialne, takie jak obrazy i audio oraz wiele innych prymitywów dla tzw. *rich layout*
  - Ostatecznie postać hipermediów
- **Aplikacja webowa**
  - Zastosowanie **JavaScript** i późniejszych technik **Dynamic HTML** (DHTML) i **AJAX** przekształca prostą stronę webową w interaktywną aplikację webową, która pozwala realizować "akcje" użytkownikowi bezpośrednio w przeglądarce i zmieniać zawartość w locie.

Web applications

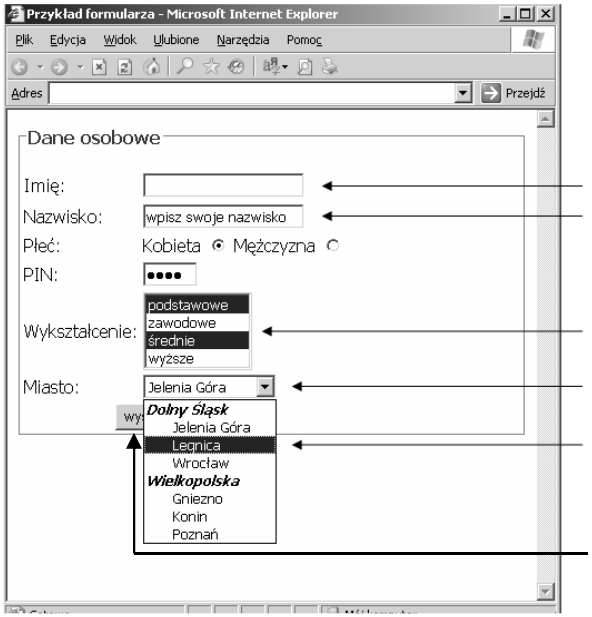
- **Podstawowa/tradycyjna metoda interakcji C-S**
  - Pozwala przesłać dane do serwera
  - Znacznik: **<form …> … </form>**
    - Atrybut **action** – adres url, mailto:
    - Atrybut **method** – GET, POST

```
<form action="http://sever.adress.com/form.php" method="post"> … </form>
```



Web applications

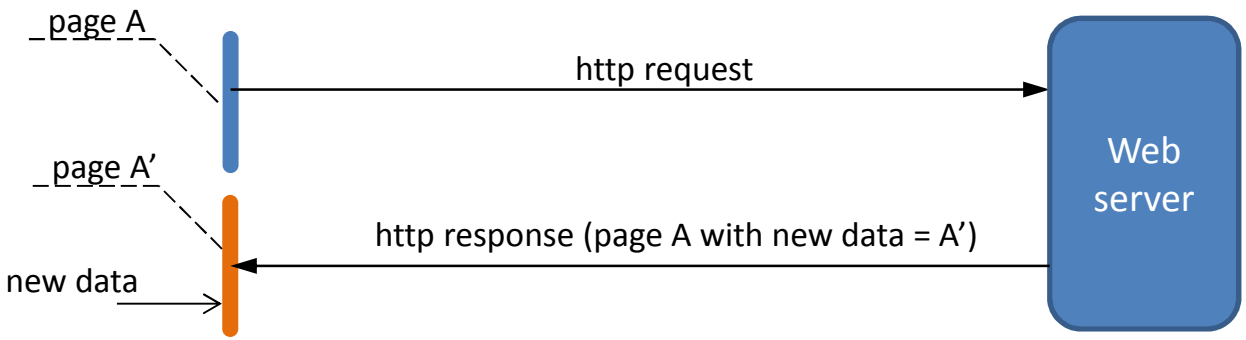
- **Zestaw znaczników do konstrukcji formularza**
  - <textarea> - pole wielowierszowe
  - <select> - lista rozwijalna
  - <input> - najczęściej wykorzystywany
    - Wiele atrybutów
      - › type
        - text
        - …
        - submit
      - › name
      - › value
      - › size
      - › readonly
      - › checked
      - › …
  - <hidden>
  - …



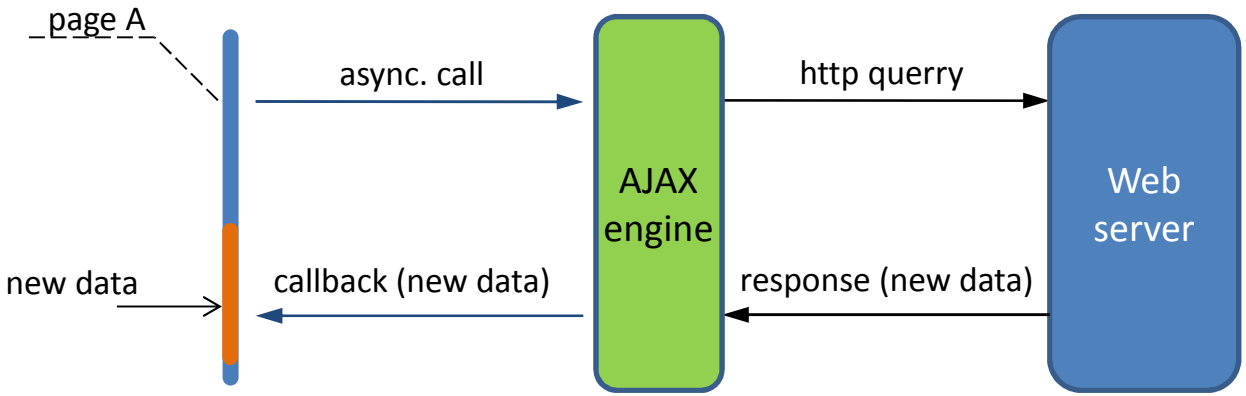
<input ... disabled="yes"/>
<input ... value="wpisz swoje nazwisko"/>
<select ... multiple="yes" size="4">
<select ...><optgroup label="Dolny Śląsk">
<option ... selected="yes"/>
<input type="submit" value="wyślij">

Web applications

- **Tradycyjna transakcja webowa**



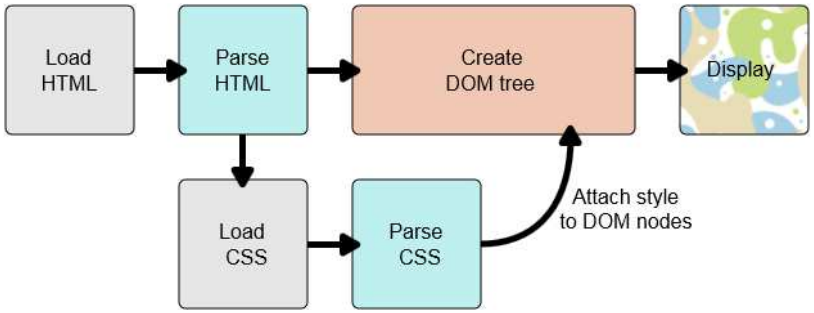
- **AJAX**



DOM
JavaScript

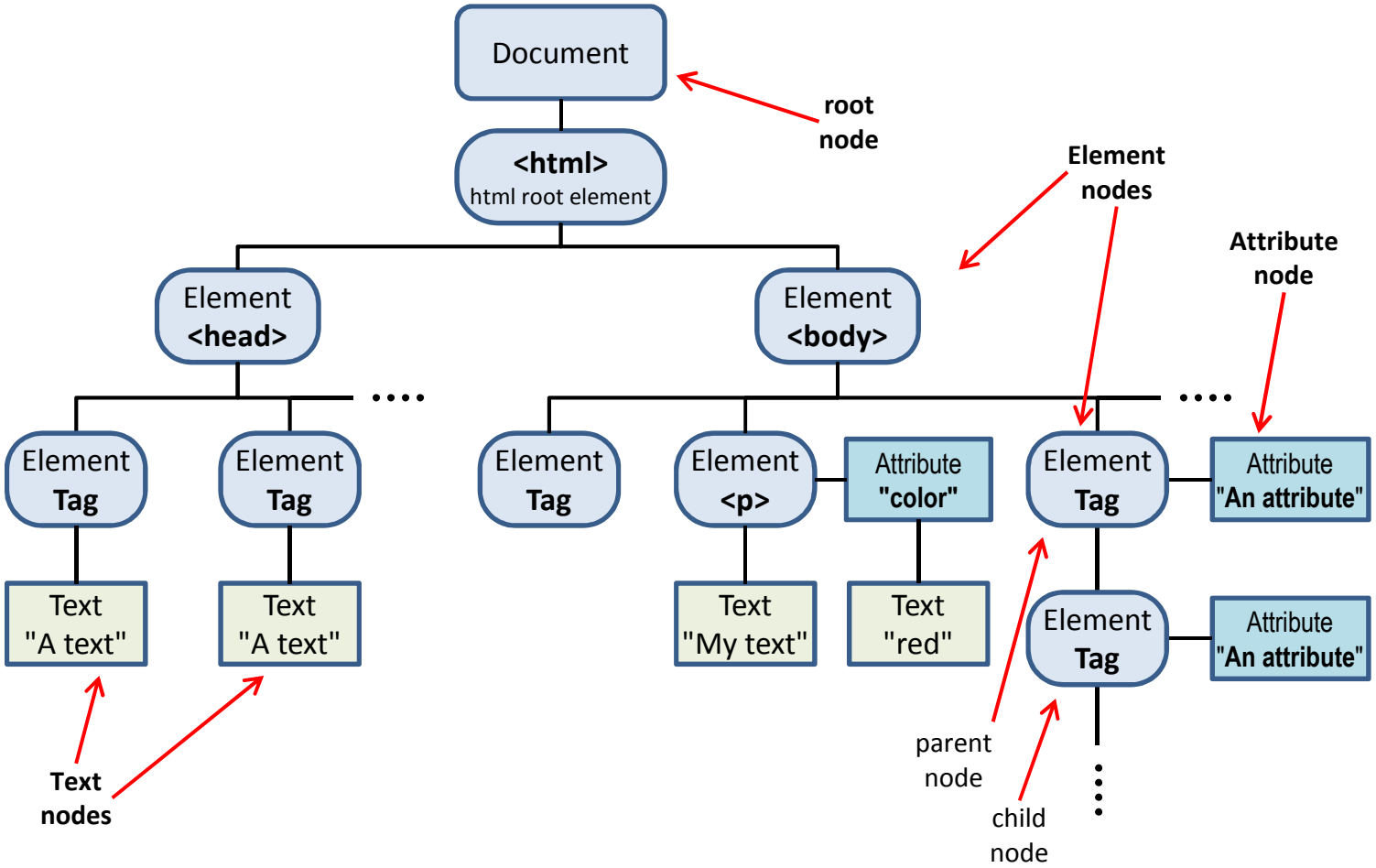
Document Object Model (DOM)

- **Document Object Model (DOM)**
  - A language-independent and cross-platform application programming interface for complex documents such as HTML or XML
  - Treats document as a tree structure wherein each node is an object representing a part of the document
  - HTML DOM
    - HTML document represented with document element objects
  - BOM – Browser Object Model
    - For browser elements (window, history,…)



Document Object Model (DOM)

- **HTML DOM**

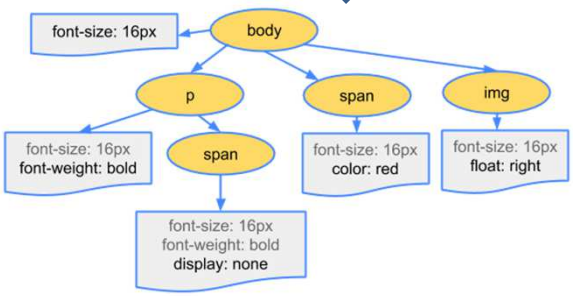


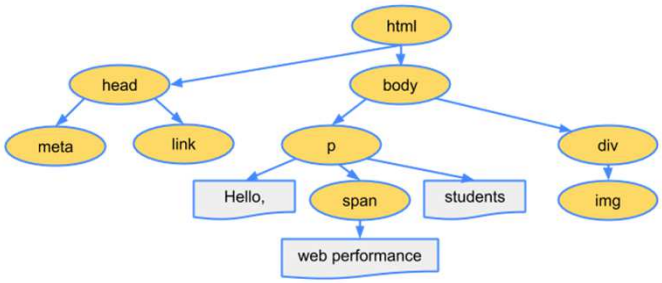
Document Object Model (DOM)

- **HTML DOM and CSSOM**
  - CSSOM – for CSS elements
  - DOM and CSSOM build includes:
    - Conversion
    - Tokenization
    - Lex analysis

```
<html>
 <head>
  <link href="style.css" rel="stylesheet">
  <title>Critical Path</title>
 </head>
 <body>
  <p>Hello <span> web performance </span> students! </p>
  <div> <img src="some-photo.jpg"> </div>
 </body>
</html>
```

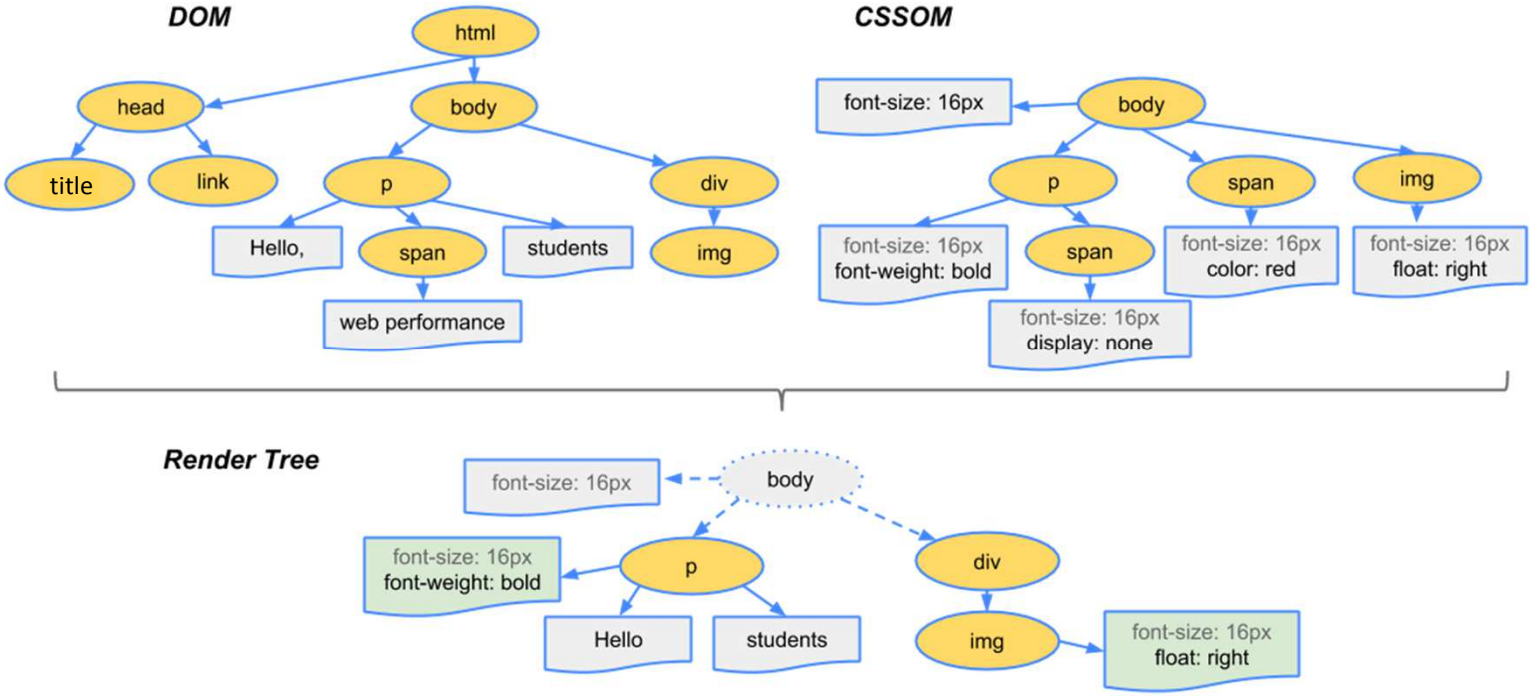
```
{ font-size: 16px }
p { font-weight: bold }
span { color: red }
p span { display: none }
img { float: right }
```





Document Object Model (DOM)

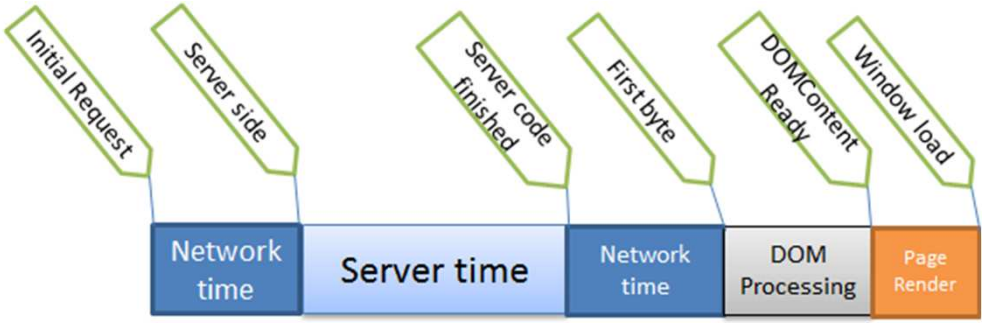
- **DOM + CSSSOM = Render-Tree**



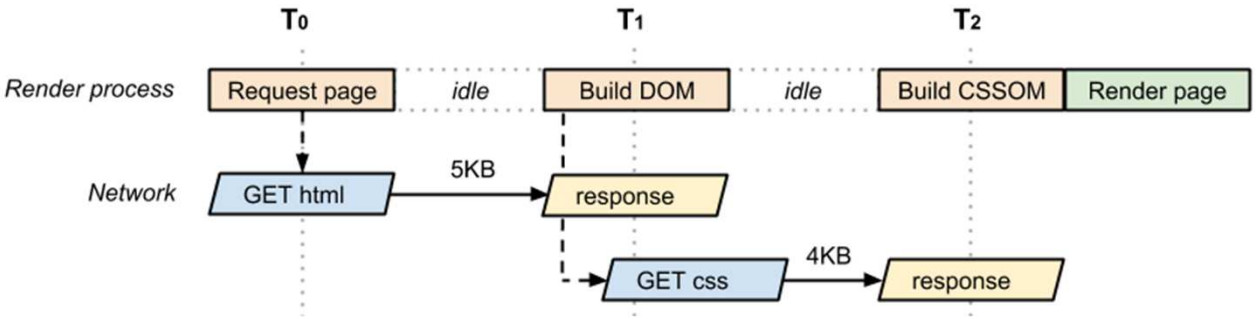
- The browser displays the content according the Render-Tree
- **While modifying Render-Tree displayed content is modified**

Web transaction

- **Web page load and rendering phases**
  - Traditional (single) web page load and rendering
  - Page without client-side processing (e.g. JS)



  - Client (browser) processing



Buit-in Web Browser APIs

- **Render-tree object manipulation $\rightarrow$ change of view**
  - The objects (document or window elements) can be manipulated programmatically and any visible changes occurring as a result may then be reflected in the display of the document
- **The DOM API**
  - Allows to manipulate HTML and CSS, creating, removing and changing HTML, dynamically applying new styles to your page, etc.
- **Other APIs**
  - **The Geolocation API**
    - Retrieves geographical information (this is how Google Maps is able to find and plot your location on a map)
  - **The Canvas and WebGL APIs**
    - Allow to create animated 2D and 3D graphics.
  - **Audio and Video APIs** (like HTMLMediaElement and WebRTC)
    - Allow to do various actions with multimedia (such as play audio and video right in a web page, or grab video from web camera and display it on someone else's computer)

JS Basics

- **JavaScript (JS)**
  - JavaScript is a lightweight and free for use scripting language
    - portable
    - object-oriented
    - interpreted or JIT-compiled
  - Programs written in JavaScript can run in web browsers
  - JavaScript helps to build dynamic web pages
    - i.e. pages that can be modified “on the fly” in response to events (such as user input, time changes and more) and applications
  - Application of JavaScript
    - It enables to do the client-side programming of web applications
      - Rich Internet Applications
      - Simple and complex processing on client side
        - E.g. data validation in HTML forms
      - Support for interactivity on web pages
        - Support for AJAX

JS basics

- **Using JS in Web pages**
  - Between <script> and </script> tags

```
<script type="text/javascript">
    document.getElementById("myID").innerHTML = "New Text";
  </script>
```

example function

  - In <head> or <body> section

```
<head>
  <script type="text/javascript">
   ...
  </script>
 </head>
```

```
<body>
  ...
  ...
  <script>  ...  </script>
</body>
```

at the end of document

  - External source of code
    - Local:
      `<script src="myScript.js"></script>`
    - Remote:
      `<script src="https://my.url.com/js/myScript.js"></script>`

JS basics

- **Instrukcje języka**
  - Wartości + operatory + wyrażenia + słowa kluczowe + komentarze
  - Oddzielane średnikami
- **Deklaracja zmiennych i funkcji**
  - Deklaracja z użyciem słowa kluczowego **var**
  - Opcjonalnie można do zmiennej przypisać domyślną wartość
  - Zmienna bez przypisanej wartości jest typu undefined
- **Operatory**
  - arytmetyczne, logiczne, przypisania, porównania, bitowe, specjalne
- **Instrukcje**
  - warunkowe: **if...else, switch**
  - iteracyjne: **for, do...while, while, break**
  - operujące na obiektach: **for...in, with**

```
var myVariable;    //undefined
var pi = 3.14;
```

Przykłady:

```
var a = b + c;
var tasty = fresh && sweet;
var _more *= 1.01;
var comp = (big > small);
var _xor = a ^ b;
var max = (a > b) ? A : b;
```

JS basics

- **Tablice**
  - Tablice są specjalnym rodzajem obiektów
  - Definiowanie: var *array_name* = [*item1*, *item2*, ...];
    var *array_name* = new Array (*item1*, ...);
  - Dodawanie elementów: *array_name*[...] = *item*
    lub tak: *array_name*.push(*item*)
  - Tablice asocjacyjne – indeksowane nazwami – jak obiekty
- **Funkcje**

```
function sin(arg) {
    return (Math.exp(arg) - Math.exp(-arg)) / 2;
}
```

  - Deklarowanie z użyciem słowa kluczowego **function**, po którym podajemy nazwę i w nawiasach **opcjonalną** listę argumentów,
  - Nawiasach klamrowych budujemy ciało funkcji
    - może zawierać niemal dowolne instrukcje języka JavaScript
  - Wywołanie poprzez nazwę funkcji
  - Funkcje są obiektami

JS basics

- **Argumenty funkcji**
  - Definiując funkcję, można zaznaczyć jej kluczowe argumenty nadając im nazwę
  - Nazwanie argumentów **nie ogranicza** możliwości wywołania funkcji z mniejszą lub większą ich liczbą

```
function test(a, b, c) {…}

test(1, 2, 3);      // komplet argumentów
test();
test(1, 2);
test(1, 2, 3, 4);  // więcej argumentów niż
                   // definiuje sygnatura funkcji
```

  - Każda funkcja ma powiązany ze sobą "magiczny" obiekt, do którego możemy odwołać się przez nazwę **arguments**
    - Obiekt ten indeksuje wszystkie parametry przekazane do funkcji

```
function test(a, b) {
    console.log(
        arguments[0],
        arguments[1],
        arguments[2]
    ); // 1, 2, 3
}
```

JS basics

- **Obiekty**
  - Obiekt to abstrakcja danych
    - Cechy (pola, składowe)
    - Funkcjonalność (funkcje, metody)
  - Metody tworzenia obiektów
    - deklaracja →
    - inicjalizacja →
    - konstruktor →

```
var punkt = new Object();
var punkt = {x:2, y:5, z:1};
function Point(x,y) {
  this.x = x;
  this.y = y;
}
var punkt = new Point(7,4)
```

  - Dostęp do składowych: *objectName.propertyName*
    lub: *objectName["propertyName"]*
  - Obiekty predefiniowane:
    - Array, Boolean, Date, String, Function, Math, Number, RegExp

JS basics

- **this**
  - The JavaScript **this** keyword refers to the object in which scope it is used
  - Contexts:
    - In a method **this** refers to the owner object
    - Alone **this** refers to the global object
    - In a function, **this** refers to the global object
    - In an event **this** refers to the element that received the event
    - In a function, in strict mode, **this** is undefined
    - Methods like call(), and apply() can refer **this** to any object.
  - Examples:
    - In method:

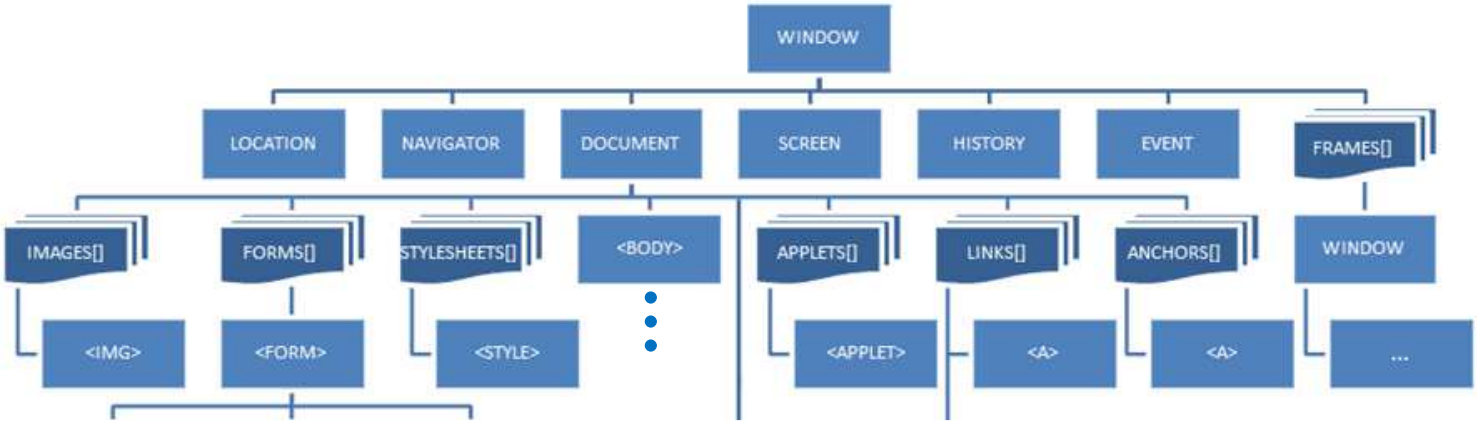
```
fullName : function() {
  return this.firstName + " " + this.lastName;
}
```

    - W event handler:

```
<button onclick="this.style.display='none'">
  Click Me!
</button>
```

JS basics

- **JS scripts and HTML document**
  - Selected standard objects available in JS:
    - Window
      - › Represents a browser window
      - › Created every time it encounters a <body> or a <frameset>
      - › Objects: document, location, frames, history, navigator, event, screen
    - Document
      - › Represents the document displayed in the browser
      - › Objects: body, links, anchors, images, forms, frames, scripts, …
    - …



JS basics

- **JS scripts and HTML document**
  - JS Allows to modify the appearance of a document or navigate through the history of the current window
    - Change HTML Content
    - Change HTML Attributes
    - Change HTML Styles (CSS)
    - Hide/show HTML Elements
  - JS Allows to choose an operations depending on detected platform (browser, operating system)

  - Kod JS jest wykonywany na skutek definiowanych zdarzeń

JS basics

- **JS HTML DOM**
  - Wyszukiwanie elementów HTML

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

```html
<html>
 <body bgcolor="blue">
  <p id="t1">First text</p>
  <p class="c2">Second text</p>
 </body>
</html>
```

```javascript
<script type="text/javascript">
  document.getElementById("t1").innerHTML = "New Text";
  document.getElementByClass("c2").innerHTML = "Text 2";
  document.getElementByTagName("p").innerHTML = "Text 3";
</script>
```

  - Dodawanie/usuwanie elementów

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

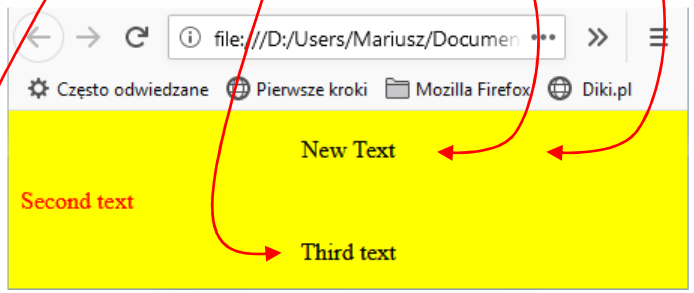
JS basics

- **JS HTML DOM**
  - Zmiana elementów HTML i CSS

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element
Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

```html
<html>
 <body bgcolor="blue">
  <p id="t1">First text</p>
  <p id="t2">Second text</p>
  <p id="t3">Third text</p>
 </body>
</html>
```

```javascript
<script type="text/javascript">
  document.getElementById("t1").innerHTML = "New Text";
  document.getElementById("t1").align = "center";
  document.getElementById("t2").style.color = "red";
  document.getElementById("t3").setAttribute("align","center");
</script>
```



JS basics

- **JS HTML DOM**
  - Wyszukiwanie obiektów HTML

Property	Description	DOM level
document.anchors	Returns all <a> elements that have a name attribute	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.documentElement	Returns the <html> element	3
document.domain	Returns the domain name of the document server	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

JS basics

- **JS events**
  - Kod JS jest wykonywany na **skutek (pre)definiowanych zdarzeń**
  - Zdarzenia są czymś co może dotyczyć (wydarzyć się dla) elementów HTML
    - Generowane przez użytkownika lub przeglądarkę
    - Dla każdego rodzaju elementu strony
  - Many mouse, keyboard, media,…, and window DOM events
    - Window:
      - › onLoad, onUnload, onError, onPageShow, …
    - Input:
      - › onMouseOver, onMouseOut, onClick, onMouseDown, onMouseUp, …,
      - › onKeyPress, …
    - Forms:
      - › onBlur, onFocus, onChange, onSelect, onSubmit, …
    - Media:
      - › onPlay, onPause, onEnded, …

JS Basics

- **Zdarzenia JS**
  - JS może zareagować na zdarzenie poprzez *event handler*
    - Definiowanie w HTML:

      *<element event=**'some JavaScript'**>*

      lub: *<element event=**"some JavaScript"**>*

zdarzenie jakaś funkcja

```
<!DOCTYPE html>
<html>
<body onload = "doSomething()">

<h1 onclick = "this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```

JS basics

- **JS HTML DOM**
  - Dodawanie obsługi zdarzenia

Method	Description
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

  - Dodawanie słuchacza

Method	Description
document.getElementById("myBtn"). addEventListener("click", listener);	Adding event listener function to an onclick event for element of id=myBtn

    - Other event handlers are NOT overridden
    - Many event handlers can be added
    - Can be remooved with `removeEventListener()` function

JS basics

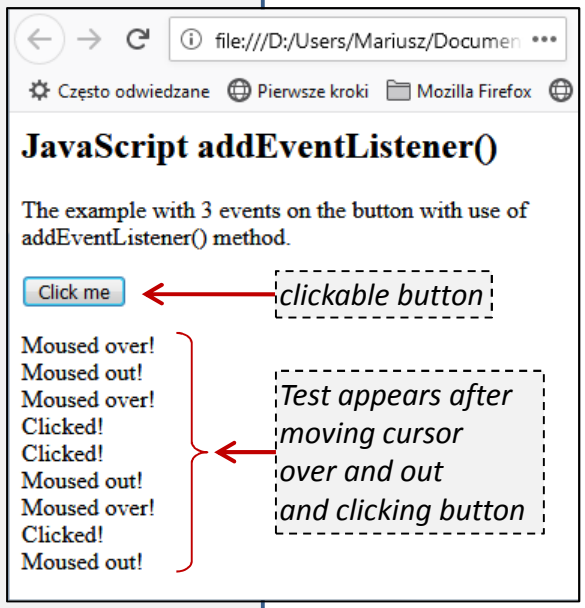
- **Przykład obsługi zdarzenia**

```
<html>
 <head>
  <script type="text/javascript">
    function mouseOver() {
      document.getElementById("parid1").innerHTML = "Text with pointer";
    }
    function mouseOut() {
      document.getElementById("parid1").innerHTML = "Point to me...";
    }
  </script>
 </head>
 <body>
  <p id="parid1" onMouseOver="mouseOver()" onMouseOut="mouseOut()" >
     Point to me...
  </p>
 </body>
</html>
```

JS basics

- **Przykład obsługi zdarzenia**

```
<html>
 <body>
  <h2>JavaScript addEventListener()</h2>
  <p>The example with 3 events on the button using addEventListener() method.</p>
  <button id="myBtn">Click me</button>
  <p id="text"></p>
  <script>
   var z = document.getElementById("myBtn");
   z.addEventListener("mouseover", myFunction1);
   z.addEventListener("click", myFunction2);
   z.addEventListener("mouseout", myFunction3);
   function myFunction1() {
    document.getElementById("text").innerHTML += "Moused over!<br>";
   }
   function myFunction2() {
    document.getElementById("text").innerHTML += "Clicked!<br>";
   }
   function myFunction3() {
    document.getElementById("text").innerHTML += "Moused out!<br>";
   }
  </script>
 </body>
</html>
```



JS basics

- **JS Timing**
  - The **window** object allows execution of code at specified time

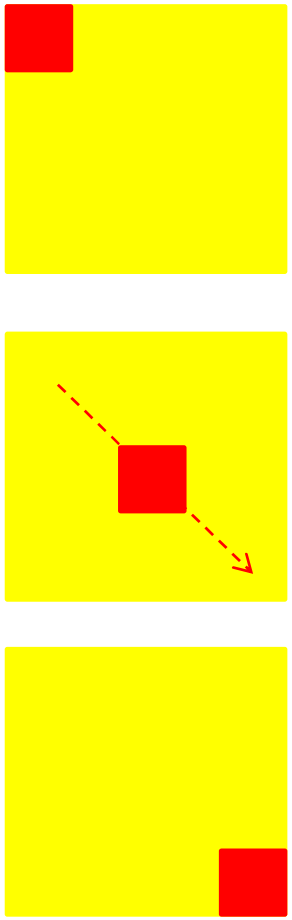
Method	Description
setTimeout(function, milliseconds)	Executes a function, after waiting a specified number of milliseconds.
setInterval(function, milliseconds)	Same as setTimeout(), but repeats the execution of the function continuously.

  - Example

```
<html>
<body>
 <p>Click "Do it" button. Wait 2 seconds, and the page will alert "Clicked!".</p>
 <button onclick="setTimeout(myFunction, 2000);">Do it</button>
 <script>
  function myFunction() {
   alert('Clicked!');
  }
 </script>
</body>
</html>
```

JS basics

- **Animations**



```
<html>
<style>
 #container {width:400px; height:400px; position:relative; background:yellow;}
 #animate {width:50px; height:50px; position:absolute; background-color:red;}
</style>
<body>
 <p><button onclick="myMove()">Click Me</button></p>
 <div id ="container">
  <div id ="animate"></div>
 </div>
 <script>
  function myMove() {
   var elem = document.getElementById("animate");
   var pos = 0;
   var id = setInterval(frame, 5);
   function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + "px";
      elem.style.left = pos + "px";
  }}}
 </script>
</body>
</html>
```

JS basics

- **JS JSON**
  - Wbudowane funkcje do obsługi JSON
  - JSON.parse(…) – konwersja tekstu do obiektu JSON
    - Obiekt JSON →

```
var obj = { "employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"} ] }
```

    - Tekst →

```
var text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

    - Obiekt JSON →

```
var obj = JSON.parse(text);
```

    - Przykład wykorzystania →

```
<p id="id1"></p>

<script>
  document.getElementById("id1").innerHTML =
  obj.employees[1].firstName + " " +
  obj.employees[1].lastName;
</script>
```

JS basics

- **JS JSON**
  - Wymiana danych z serwerem jest z użyciem string'ów
  - JSON.stringify (…) – konwersja obiektu JSON do tekstu
    - Obiek

```
var obj = { name: "John", age: 30, city: "New York" };
```

    - Konwersja do tekstu

```
var myTxtJSON = JSON.stringify(obj);
```

    - Przykład wykorzystania

```
<p id="id1"></p>
<script>
 var obj = { name: "John", age: 30, city: "New York" };
 var myTxtJSON = JSON.stringify(obj);
 document.getElementById("id1").innerHTML = myTxtJSON;
</script>
```

- **Obie funkcje przydatne dla konwersji danych przekazywanych w wiadomościach HTTP (np. w aplikacjach typu REST)**
  - Gdy przesyłamy obiekty JSON i z nich wyciągamy dane
  - Różne języki/platformy/frameworki maja wbudowane mech. konwersji

JS basics

- **Wersje JS**
  - ECMAScript – specyfikacja standaryzująca język skryptowy (JavaScript od 1997)
    - Różne implementacje i dialekty: JavaScript, JScript, JScript.Net, ActionScript, …
    - TypeScript – ścisły nadzbiór ES6 (+ statyczne typowanie i OOP oparte na klasach)
    - Oznaczenia: ECMAScript <Nr_wersji>, ES<NR_wersji>, ECMAScript <rok>, ES<rok>
  - ES5 lub ECMAScript 5
    - Przykładowe nowe cechy/funkcje (w stosunku do poprz.):
      - › strict mode (dyrektywa: use strict)
      - › funkcje JSON.parse(), JSON.stringify()
      - › …inne
  - ES6 lub ECMAScript 6 / ECMAScript 2015 / ES2015
    - Przykładowe nowe cechy/funkcje (w stosunku do poprz.):
      - › Domyślne wartości parametrów
      - › Parametr `const`
      - › …inne
  - Ostatni standard: ES2020 = ECMAScript 2020 (ES11)
  - Różne wsparcie różnych wersji w przeglądarkach
    - ECMAScript 5 ma pełne wsparcie przez niemal wszystkie przeglądarki
    - ECMAScript 7 wsparcie przez nowoczesne przeglądarki

jQuery

JS basics

- **JavaScript libraries**
  - Libraries of pre-written JavaScript which allows for easier development of JavaScript-based applications
- **jQuery**
  - jQuery is currently the most popular of many JavaScript libraries
  - jQuery simplifies JavaScript programming by making it easier to manipulate a web page's elements and interact with servers in a portable manner across various web browsers
  - Provides a library of custom GUI controls that can be used to enhance the look and feel of web pages
  - jQuery is very lightweight library - about 19KB in size (Minified and gzipped)
  - jQuery has cross-browser support, and works well in majority of most popular browsers

jQuery

- **Podstawowa charakterystyka**
  - Podstawowe cechy
    - HTML/CSS/DOM manipulation
    - HTML event handling
    - AJAX support
    - Visualize effects and animations
    - Different utilities and plug-in support
  - jQuery zbudowany na JS wspiera wszystkie elementy tego języka
    - Typy zmiennych, obiekty, tablice, funkcje, wywołania zwrotne (callbacks), argumenty, kontekst this, … itd.
  - Użycie biblioteki
    - Pobranie z jQuery.com
    - Użycie z CDNa (Google'a)

```
<head>
 <script src="jquery-3.5.1.min.js"></script>
</head>
```

```
<head>
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
```

jQuery

- **Podstawowe elementy programowania**
  - Składnia jQuery jest ukierunkowana na selekcję elementów HTML
  - Podstawowa składnia wywołania:

```
$=(selector).action()
```

    - selector – element specyfikujący element HTML
    - action – specyfikuje działanie/metodę do wykonania
    - znak **$** - obiekt dostępu do metod/własności jQuery (**$** jest aliasem **jQuery**)
  - Selektory
    - **Tag ID** (#id)
    - **Tag Name** (name)
    - **Class Name** (.class_name)
    - * – wszystkie elementy
    - this – element aktualny kontekstowo
    - … + kilka specyficznych
  - Różne połączenia i warianty selekcji

```
$("#id1").hide()
$("p").hide()
$(".test").hide()
$(*).hide()
$(this).hide()
$("p.test").hide()
$("[color]").hide()
```

- ukrywa element o id=id1
- ukrywa wszystkie elementy <p>
- ukrywa elementy z class="test"
- ukrywa wszystko
- ukrywa aktualny element
- ukrywa elem. <p> z klasą "test"
- ukrywa elementy z atryb. *color*

jQuery

- **Podstawowe elementy programowania**
  - Funkcje
    - Funkcje definiuje się jak dla JS (w tym, gdy parametrem metody jQuery jest funkcja)

```
$(selector).action(…, function() { … });
```

    - Często kod umieszczany jest w funkcji `ready(…)` →

```
$(document).ready(function() {
  // tu kod
});
```

      - Wywoływane po zbudowaniu drzewa DOM
  - Zdarzenia (*Events*)
    - Kilka sposobów wiązania obsługi zdarzeń z elementami treści (strony)
      - Metoda on()

```
$(selector).on("event_name", function( event ) { ... } );
```

np.:

```
$("#button_id").on("click", function() { alert("Hello"); } );
```

      - Skrócona forma dla popularnych zdarzeń →

```
$(selector).event_name( function( event ) { ... });
```

np.:

```
$("#button_id").click( function() { alert("Hello"); } );
```

      - Metoda bind() i unbind()

```
$(selector).bind("event_name", function( event ) { ... });
```

    - Powszechnie obsługiwane zdarzenia:
      - Są różnice nazw w stos. do JS

Mouse Events	click	dblclick	mouseenter	mouseleave
Keyboard Events	keypress	keydown	keyup	resize
Form Events	submit	change	focus	
Document/Window Events	load	blur	scroll	unload

jQuery

- **Podstawowe elementy programowania**
  - Większość metod jQuery jest wywoływanych na obiektach jQuery
    - Metody te są w przestrzeni nazw **$.fn** – można je traktować jako metody obiektowe jQuery
      - › Metody wywoływane na selektorze automatycznie zwracają element tak jak **this**
    - Wybrane, przydatne metody:
      - › Get/Set
        - `val()` – sets or returns the value of form fields
        - `html()` – sets or returns the content of elements (including HTML markup)
        - `text()` – sets or returns the text content of elem.
      - › Add/Remove
        - `append()` – inserts content at the end of elements
        - `prepend()` – inserts content at the beginning of elem.
        - `remove()` – removes the selected element (and its child elements)
      - › CSS
        - `css()` – sets or returns CSS property
      - › Traversing methods - wiele

html

```
<p id="id1">Enter <b>correct</b> name</p>

<p>Name: <input type="text" id="id2"
value="Miś Uszatek"></p>
```

zwraca: "Enter correct name"

zwraca: "Enter correct name"

```
alert("Text: " + $("#id1").text());
alert("HTML: " + $("#id1").html());
alert("Value: " + $("#id2").val());
```

skrypt

zwraca: "Miś Uszatek"

```
$("p").append("tekst1");
$("p").prepend("tekst2");
$("#div3").remove();
```

dołącza "tekst1" do elementów <p>

```
$("p").css("color", "red");
```

jQuery

- **Podstawowe elementy programowania**
  - Istnieje kilka metod, które działają bez selektora – są uważane za część przestrzeni nazw **$** - można je traktować jako metody podstawowe jQuery
    - Są to na ogół metody typu narzędziowego
      - › Nie działają z selekcją – kolekcjami jQuery (jak np. kolekcja paragrafów $("p"))
      - › Działają na zwykłych obiektach (*plain objects*)
      - › Nie są automatycznie przekazywane żadne argumenty (trzeba wprost przekazywać)
      - › Zwracane są różne wartości

```
$.action()
```

    - Przykład:

      $.each() – ogólna metoda iterująca obiekty, tablice i obiekty podobne do tablicy
      - › Iteruje i zwraca także złożone obiekty
      - › Tu: nie trzeba odwoływać się specyfikując indeks, czyli tak: `obj[key]`
      - › *Uwaga*: tu wyj. istnieje analogiczna met. Obiektowa **`$(selector).each(…)`**
        - tą metodę trzeba używać dla kolekcji jQuery !

```
var obj = {x:{i:"Ala",n:"Nowak"},
           y:{i:"Ola",n:"Kowal"}}

$.each( obj, function( key, val ) {
  alert("element "+key+".n"+" to "+ val.n);
});
```

jQuery

- **AJAX w jQuery**
  - Metoda $.ajax

tu: tradycyjny styl dla obsługi odpowiedzi

```
$.ajax({
   url         : "http://host.domain.org/service",  // adres wysyłki żądania
   method      : "POST",                            // typ połączenia/metoda, domyślnie GET
   dataType    : "json",                            // typ danych spodziewanych w odpowiedzi
   contentType : "application/json",                // typ dla wysyłanych danych
   data        : {                                  // dane do wysyłki (np. obiekt JSON)
                  name : "Mariusz",
                  country : "Polska"
                 }
   success     : function() {           // kod wywoływany jeśli żądanie zakończone powodzeniem
                   alert( "success" );
                 }
   error       : function() {           // kod wywoływany po błędzie
                   alert( "error" );
                 }
   complete    : function() {           // kod wywoływany zawsze
                   alert( "complete" );
                 }
});
```

To Be Continued…

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Systemy Webowe (cz.3)

- Klienckie aplikacje Webowe
  - ✓ ...
  - ✓ AJAX
  - ✓ Klient AJAX i REST

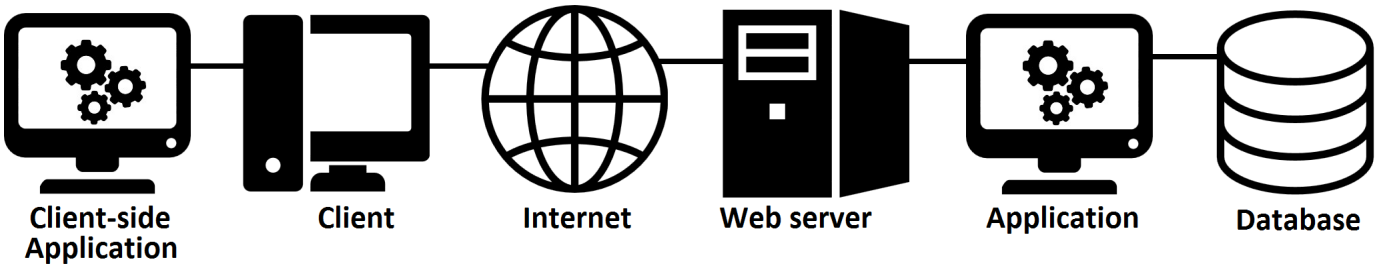
AJAX

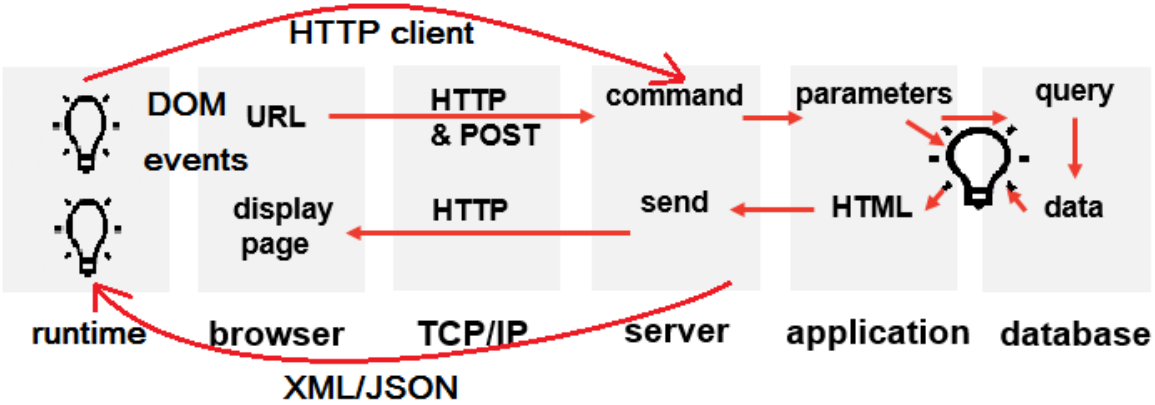
- **AJAX (Asynchronous JavaScript And XML)**
  - Nie jest nową odrębną technologią
  - Jest nową techniką tworzenia aplikacji webowych
    - sposobem wykorzystania kombinacji istniejących technologii HTML, CSS, DOM, JavaScript
  - Nowy sposób myślenia nt. realizacji aplikacji internetowych, który pozwala uzyskać poziom interaktywności zbliżony do aplikacji desktopowych
  - Idea:
    - Asynchroniczna komunikacja z serwerem z poziomu kodu JavaScript (wykonywanego w przeglądarce)
    - Serwer na żądanie wysyła dane, a nie całe strony
    - Dane wykorzystywane przez kod JavaScript w przeglądarce
    - Dokument w przeglądarce modyfikowany programowo przez interfejs DOM

Web transaction

PRZYPOMNIENIE

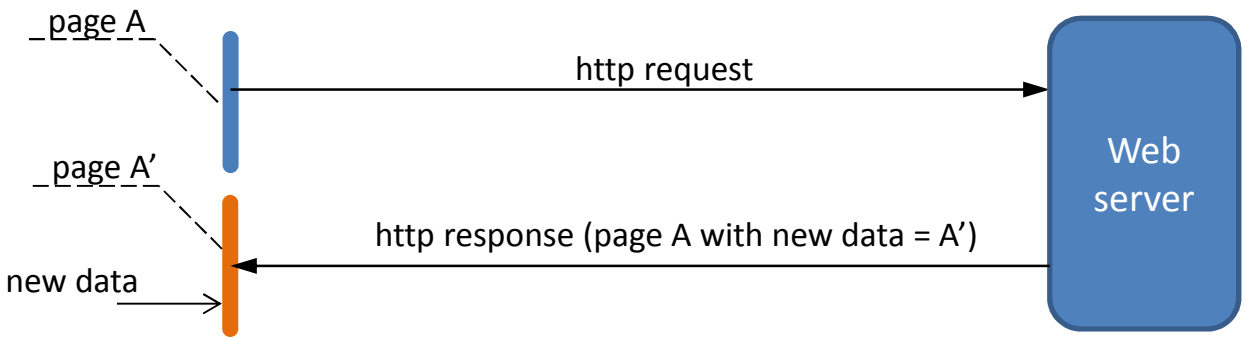
- **Rich Client Asynchronous Transaction**



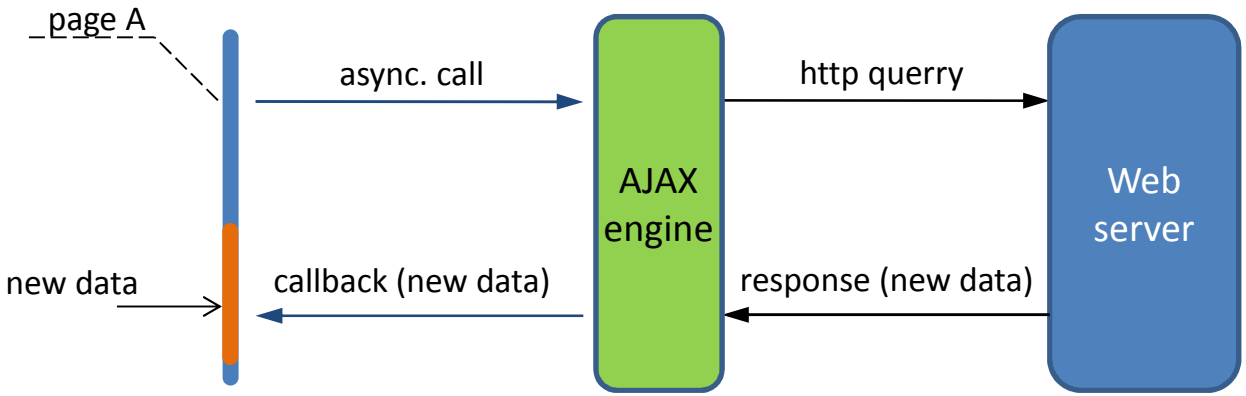


AJAX

- **Tradycyjna transakcja webowa**

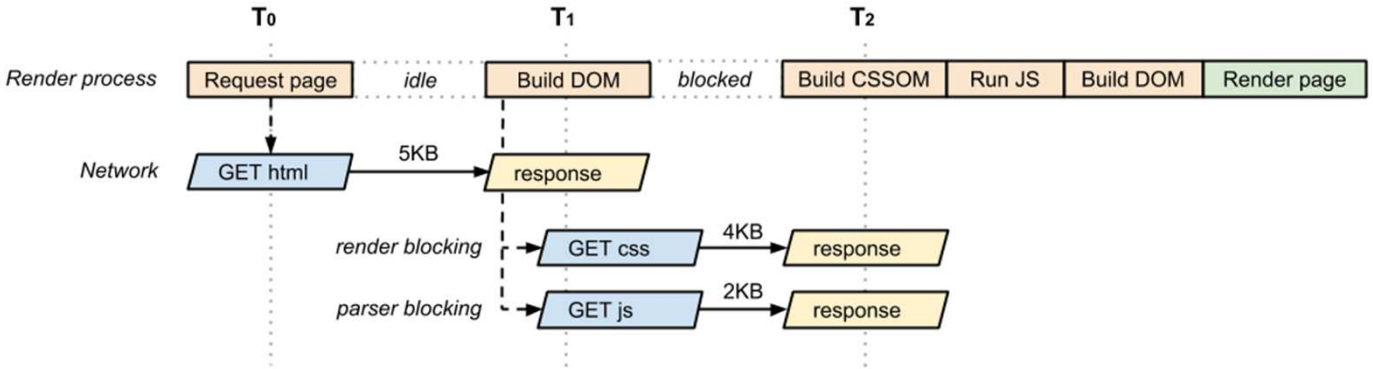


- **AJAX**

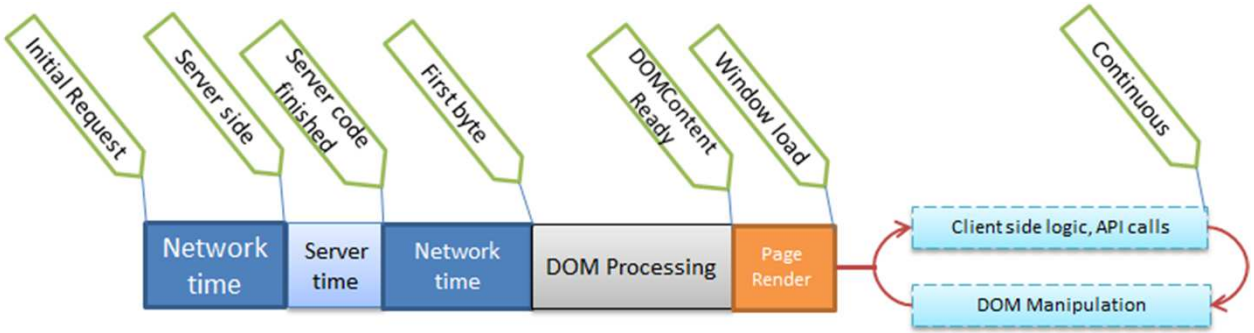


Web transaction

- **Web page load and rendering phases**
  - Page with client-side processing (e.g. JavaScript)



  - Rich application page
    - DOM is then continuously manipulated as a result of events



AJAX

- **XMLHttpRequest object** (**XHR** for short)
- **Methody inicjowania i wysłania żadania:**
  - **open(method, url),**
    open(method, url, async)
    open(method, url, async, username, password)
    - Metody HTTP
    - Tworzy żadanie (domyślnie asynchroniczne)
    - async=true – skrypt przetwarzany natychmiast po send(…)
  - **send(data)** – wysłanie żądania
  - **abort()** – przerwanie żadania
- **Metody dla nagłówka**
  - **void setRequestHeader(name, value)** – ustawia nagłówek
  - **string getResponseHeader(name)** – odczytuje nagłówek
  - **string getAllResponseHeaders()** – odczytuje wszyst. nagłówki

AJAX

- **XHR properties**
  - **readyState** – status wykonania (gdy nie ma błędów są maksymalnie 4 statusy)
    - 0 = uninitialized – zapytanie niezainicjowane przez open(…)
    - 1 = loading – połączenie zainicjowane ale zapytanie nie wysłane
    - 2 = loaded – nagłówek i status odpowiedzi są zwrócone
    - 3 = interactive – przetwarzanie zapytania – odbieranie danych
    - 4 = completed – żądanie zakończone – jest kompletna odpowiedź
  - **onreadystatechange**
    - referencja do funkcji wywoływanej zmianą stanu
  - **responseText**
    - zawiera odpowiedź (łańcuch znaków)
  - **responseXML**
    - odpowiedź jako XML; może być parsowany używając W3C DOM
  - **status**
    - kod odpowiedzi HTTP (np. 200, 404, itd.)
  - **statusText**
    - zawiera status jako ciąg znaków (np. "OK" lub "Not Found")

AJAX

- **Example of request implementing**

```
function doRequest()
{
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = doJob;     // callback
   xhr.open("GET", url, true);
   xhr.send();
}

function doJob()
{
  if (xhr.readyState==4) {
    if (xhr.status==200) {
      // ...process received data and/or do anything you want
    }
    else { alert("Data querry/response error"); }
  }
}
```

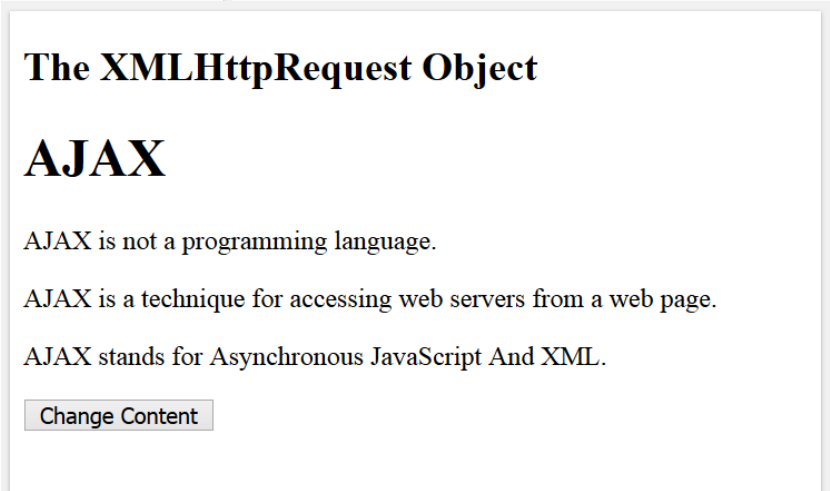
AJAX

- **Support for older IE versions**

```
function doRequest() {
  if (window.XMLHttpRequest) {
      // code for all modern browsers
      xhr = new XMLHttpRequest();
    } else {
      // code for old IE browsers
      xhr = new ActiveXObject("Microsoft.XMLHTTP");
   }
   ...
}
```

AJAX

- **Example 1 – screenshoots**



AJAX

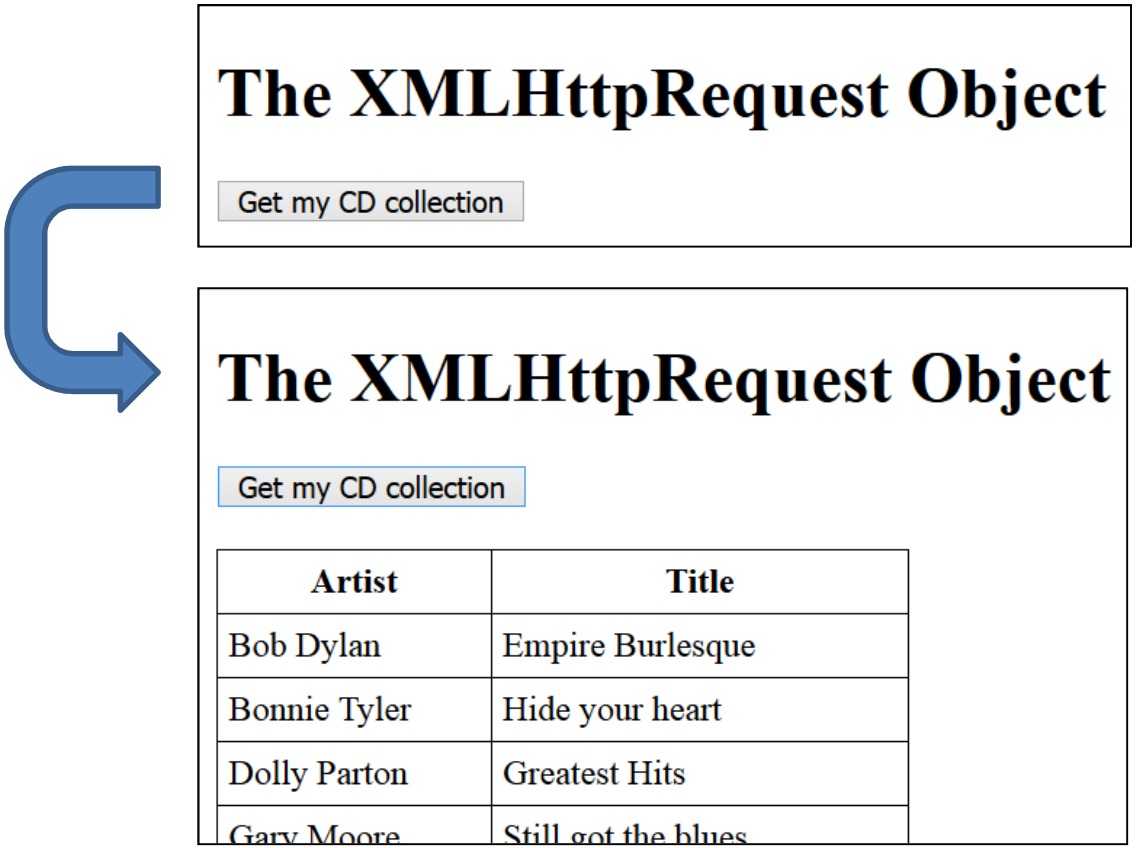
- **Example 1 - code**

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
  <h2>Let AJAX change this text</h2>
  <button type="button" onclick="loadData()">Change Content</button>
</div>
<script>
function loadData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = fun;
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
  function fun() {
    if (this.readyState == 4 && this.status == 200)
     document.getElementById("demo").innerHTML = this.responseText;
  };
};
</script>
</body>
</html>
```

in responseText the text to display after click

AJAX

- **XML example 2** – screenshoots



AJAX

- **XML example 2** – HTML

```
<head>
  ...
  <style>
   table,th,td {
    border : 1px solid black;
    border-collapse: collapse;
   }
   th,td {
     padding: 5px;
   }
  </style>
</head>
<body>
  <h1>The XMLHttpRequest Object</h1>
  <button type="button" onclick="loadData()">Get my CD collection</button>
  <br><br>
  <table id="demo"></table>
  <script>
   ...
  </script>
  ...
```

script here
(function loadData() and others)

AJAX

- **XML example 2 – cd_catalog.xml**
  - dane tego pliku zwracane przez serwis

```
<CATALOG class="...">
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  ...
</CATALOG>
```

AJAX

- **XML example 2** – HTML cont.

```
function loadData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
    myFunction(this);
    }
  };
  xhttp.open("GET", "cd_catalog.xml", true);
  xhttp.send();
}
function myFunction(xml) {
  var i;
  var xmlDoc = xml.responseXML;
  var table="<tr><th>Title</th><th>Artist</th></tr>";
  var x = xmlDoc.getElementsByTagName("CD");
  for (i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
```

XML response

AJAX

- **Zalety**
  - Interaktywne, szybkie i przyjazne aplikacje
  - Oparty o znane i dojrzałe technologie
  - Nie wymaga instalacji wtyczek w przeglądarce
  - Redukuje ruch w sieci i zmniejsza obciążenie serwera
- **Wady**
  - Język JavaScript nienajlepszy dla dużych aplikacji
  - Konieczność włączenia obsługi JavaScript
  - Wrażliwy na opóźnienia w komunikacji sieciowej
  - Trudna implementacja, testowanie i debugowanie aplikacji
  - Niedojrzałe biblioteki i narzędzia programistyczne

jQuery AJAX

- **AJAX w jQuery**
  - Metoda **`$.ajax`**

tu: tradycyjny styl dla obsługi odpowiedzi

```
$.ajax({
   url         : "http://host.domain.org/service",      // adres wysyłki żądania
   method      : "POST",                                // typ połączenia/metoda, domyślnie GET
   dataType    : "json",                                // typ danych spodziewanych w odpowiedzi
   contentType : "application/json",                    // typ dla wysyłanych danych
   data        : {                                      // dane do wysyłki (np. obiekt JSON)
                  name : "Mariusz",
                  country : "Polska"
                 }
   success     : function() {              // kod wywoływany jeśli żądanie
                   alert( "success" );     // zakończone powodzeniem
                 }
   error       : function() {              // kod wywoływany po błędzie
                   alert( "error" );
                 }
   complete    : function() {              // kod wywoływany zawsze
                   alert( "complete" );
                 }
});
```

jQuery AJAX

- **AJAX w jQuery**
  - Metoda $.ajax

tu: nowy styl dla obsługi odpowiedzi

```
$.ajax({
   url         : "http://host.domain.org/service",      // adres wysyłki żądania
   method      : "POST",                                // typ połączenia/metoda, domyślnie GET
   dataType    : "json",                                // typ danych spodziewanych w odpowiedzi
   contentType : "application/json",                    // typ dla wysyłanych danych
   data        : {                          // dane do wysyłki (np. obiekt JSON)
                  name : "Mariusz",
                  country : "Polska"
                 }
})

   .done( function( data) {                 // kod wywoływany jeśli żądanie
            alert( stringify(data) );       // zakończone powodzeniem
    })
   .fail( function(xhr, status, error) {    // kod wywoływany po błędzie
            alert(...);
    })
   .always( function() {                    // kod wywoływany zawsze
              alert( "complete" );
    });
```

jQuery AJAX

- **Inne metody (wybrane)**
  - $.load()
    - Loads data from a server and puts the returned data into the selected element
  - $.get()
    - Loads data from a server using an AJAX HTTP GET request
  - $.getJSON()
    - Loads JSON-encoded data from a server using a HTTP GET request
  - $.getScript()
    - Loads (and executes) a JavaScript from a server using an AJAX HTTP GET request
  - $.post()
    - Loads data from a server using an AJAX HTTP POST request
  - ajaxSuccess()
    - Specifies a function to run when an AJAX request completes successfully
  - ajaxError()
    - Specifies a function to run when the AJAX request completes with an error

REST and AJAX example application

REST Service and AJAX client example

Example application

- **WCF REST Service interface**

```
List<Book> books;
...
public Book getJsonBook(string Id) {
 int bookId = int.Parse(Id);
 return books.Find(b => b.Id == bookId);
}
```

```
[DataContract]
public class Book
{
  [DataMember]
  public int Id;
  [DataMember]
  public string Title;
  [DataMember]
  public double Price;
}
```

```
[ServiceContract]
public interface IRestService1
{
  [OperationContract]
  [WebGet(UriTemplate = "/books", RequestFormat = WebMessageFormat.Json,
          ResponseFormat = WebMessageFormat.Json, BodyStyle = WebMessageBodyStyle.Bare)]
  List<Book> getJsonBooks();

  [OperationContract]
  [WebGet(UriTemplate = "/books/{id}", RequestFormat = WebMessageFormat.Json,
          ResponseFormat = WebMessageFormat.Json)]
  Book getJsonBook(string Id);

  [OperationContract]
  [WebInvoke(UriTemplate = "/books/{id}", Method = "PUT", RequestFormat = WebMessageFormat.Json,
             ResponseFormat = WebMessageFormat.Json)]
  string updateJsonBook(Book aBook, string Id);

  [OperationContract]
  [WebInvoke(UriTemplate = "/books", Method = "POST", RequestFormat = WebMessageFormat.Json,
             ResponseFormat = WebMessageFormat.Json)]
  string addJsonBook(Book aBook);

  [OperationContract]
  [WebInvoke(UriTemplate = "/books/{id}", Method = "DELETE")]
  string deleteJsonBook(string Id);
}
```

relative URI of operation

C# methods implementing operations

message format

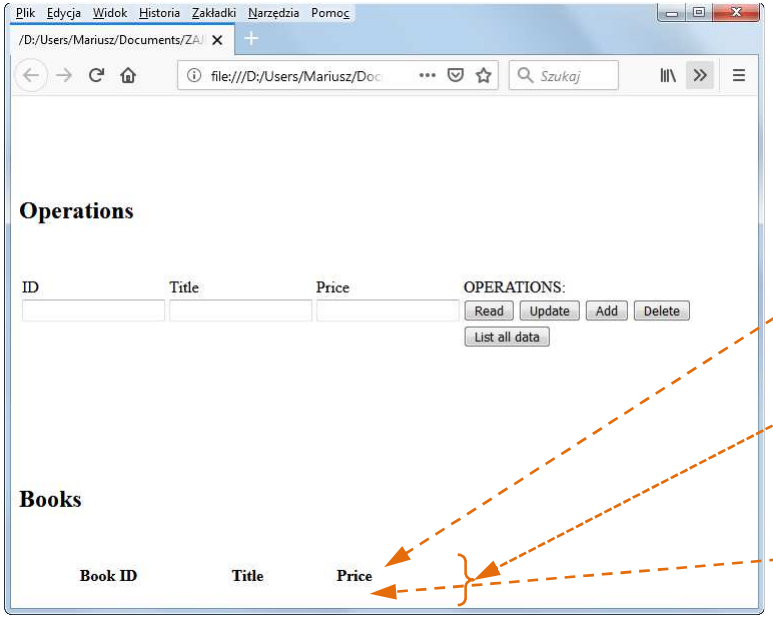
HTTP method

WebGet dla metody HTTP GET

WebInvoke dla metod POST, PUT, ...

Example application

- **AJAX based client**
  - HTML + JavaScript (jQuery)

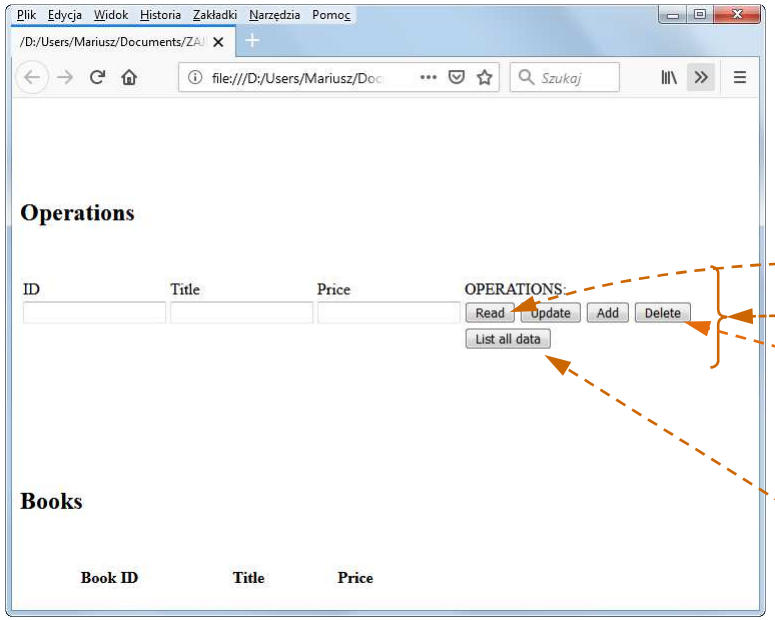



```
<html>
<head>
 ...
</head>
<body>
 ...
 <br />
 <h2 style="margin-top:50px;margin-
bottom:50px;">Books</h2>
 <table id="booktable" cellpadding="1" width="400">
  <thead>
   <tr>
    <th> Book ID </th>
    <th> Title </th>
    <th> Price </th>
    <th> </th>
    </tr>
  </thead>
  <!-- here goes tbody element -->
 </table>
</body>
</html>
```

script here

Example application

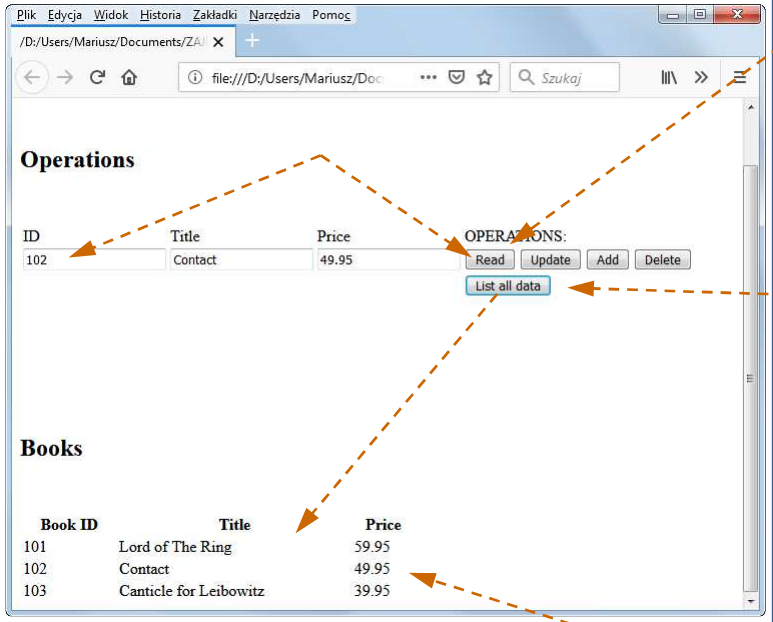
- **AJAX based client**
  - HTML + JavaScript




```
<body>
 <h2 style="margin-top:50px;margin-bottom:8px;">Operations</h2>
 <form>
  <table id="bookoperation" cellpadding="1">
   <tr> <td>ID</td>
        <td>Title</td>
        <td>Price</td>
        <td>OPERATIONS:</td>
   </tr>
   <tr>
    <td><input id="bid" type="text" /></td>
    <td><input id="title" type="text" /></td>
    <td><input id="price" type="text" /></td>
    <td>
     <button type="button" id="read" onclick="getJsonBook()">Read</button>
     <button type="button" id="upd" onclick="updateJsonBook()">Update</button>
     <button type="button" id="add" onclick="addJsonBook()">Add</button>
     <button type="button" id="del" onclick="deleteJsonBook()">Delete</button>
     </td>
   </tr>
   <tr>
    <td colspan="3"></td>
    <td><button type="button" id="showall"
                onclick="getJsonBooks()">List all data</button>
    </td>
   </tr>
  </table>
 </form>
 ...
```

Example application

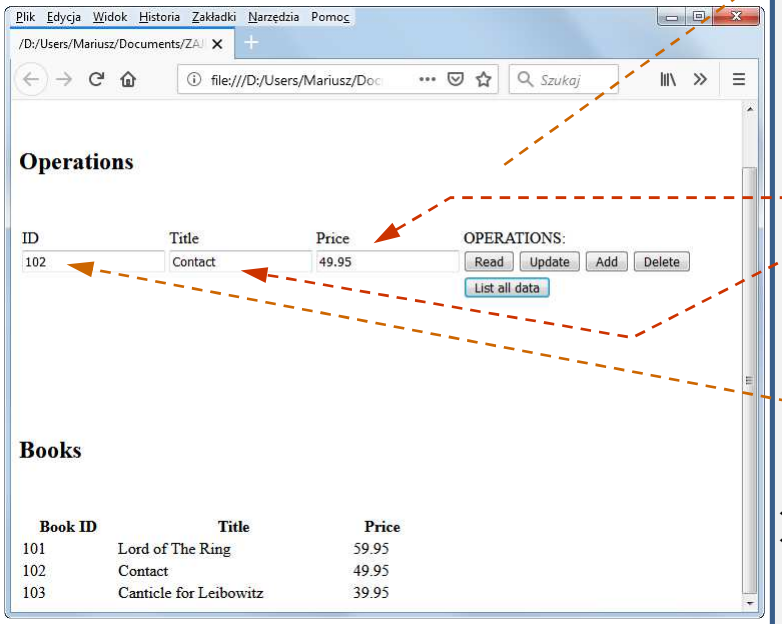
- **Read one item and list all items (books)**




```
...
 <tr>
  <td>
   <button type="button" id="read"
          onclick="getJsonBook()">Read</button>
   <button type="button" id="upd"
          onclick="updateJsonBook(this)">Update</button>
   <button type="button" id="add"
          onclick="addJsonBook(this)">Add</button>
    <button type="button" id="del"
           onclick="deleteJsonBook(this)">Delete</button>
   </td>
 </tr>
 <tr>
  <td colspan="3"></td>
  <td><button type="button" id="showall"
          onclick="getJsonBooks()">List all data</button>
  </td>
 </tr>
...
<table id="booktable" cellpadding="1" width="400">
 <thead>
  <tr>
   <th> Book ID </th>
   <th> Title </th>
   <th> Price </th>
   <th> </th>
   </tr>
 </thead>
 <!-- here goes tbody element -->
</table>
...
```

Example application

- **Read one item and display it**

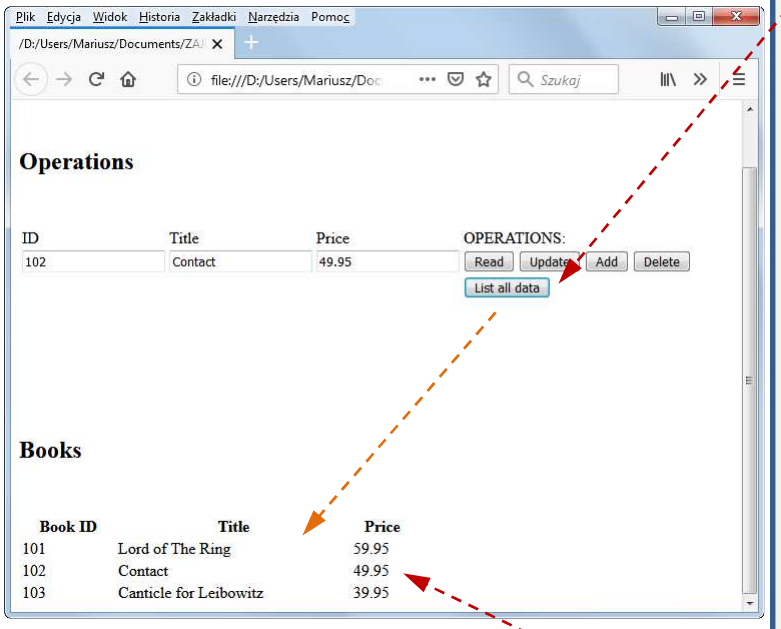



```
<html>
<head>
 <script src="https://code.jquery.com/jquery-3.2.1.min.js">
 </script>
 <script type="text/javascript">
  ...
 function getJsonBook() {
  var id = $("#bid").val();
  $.ajax({
    type: "GET",
    url: "http://localhost:3731/Service1.svc/json/books/"+id,
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: function(msg) {
                 document.getElementById("price").value=msg.Price;
                 document.getElementById("title").value=msg.Title;
    },
    error: function(errMsg) {
      alert(errMsg);
    }
  });
 }
 ...
 </script>
</head>
<body>
 ...
 <tr>
    <td><input id="bid" type="text" /></td>
    <td><input id="title" type="text" /></td>
    <td><input id="price" type="text" /></td>
 ...
</body>
</html>
```

```
[DataContract]
public class Book
{
  [DataMember]
  public int Id;
  [DataMember]
  public string Title;
  [DataMember]
  public double Price;
}
```

Example application

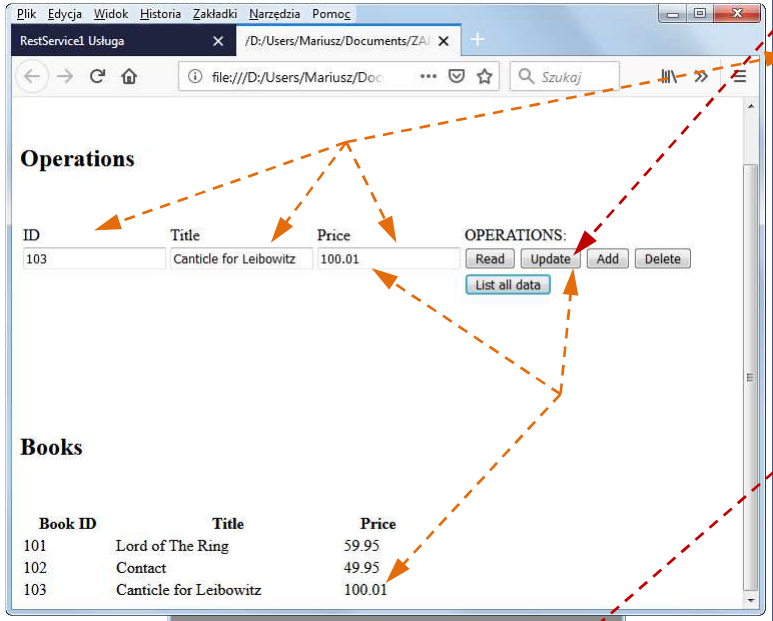
- **Read and list all items**

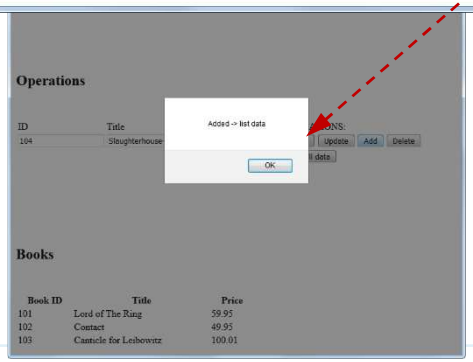



```
...
function getJsonBooks() {
 $.ajax({
  type: "GET",
  url: "http://localhost:3731/Service1.svc/json/books",
  contentType: "application/json; charset=utf-8",
  dataType: "json",
  success: function (msg) {
   $("#booktable tbody").remove();
   $.each(msg, function(id, book) {
    if ($("#booktable tbody").length == 0) {
      $("#booktable").append("<tbody></tbody>");
    }
    $("#booktable tbody").append("<tr>"+
                                 "<td>"+ book.Id +"</td>"+
                                 "<td>"+ book.Title +"</td>"+
                                 "<td>"+ book.Price +"</td>"+
                                 "</tr>");
   });
  },
  error: function (msg){
    alert("error");
  }
 });
}
...
<body>
 ...
 <table id="booktable" cellpadding="1" width="400">
  <thead> ... </thead>
  <!-- here goes tbody element -->
 </table>
 ...
</body>
</html>
```

Example application

- **Update item**

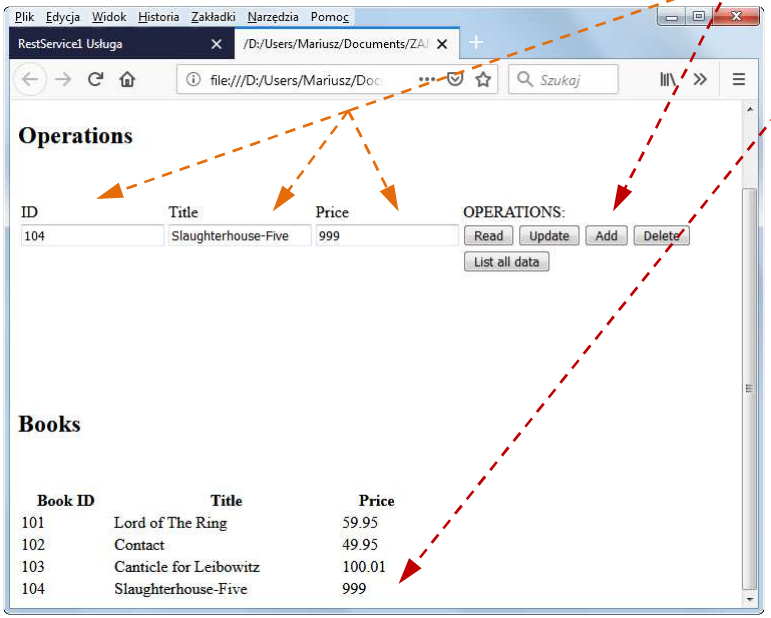





```
...
function updateJsonBook() {
 var book = Book;
 book.Id = $("#bid").val();
 book.Title = $("#title").val();
 book.Price = $("#price").val();
 var id = $("#bid").val();
 $.ajax({
    type: "PUT",
    url: "http://localhost:3731/Service1.svc/json/books/"+id,
    data: JSON.stringify(book),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: function(msg) {
      alert('Updated -> list data');
    },
    error: function(errMsg) {
      alert(errMsg);
    }
 });
}
...
</script>
</head>
<body>
...
<tr>
   <td><input id="bid" type="text" /></td>
   <td><input id="title" type="text" /></td>
   <td><input id="price" type="text" /></td>
...
</body>
</html>
```

Example application

- **Add item**

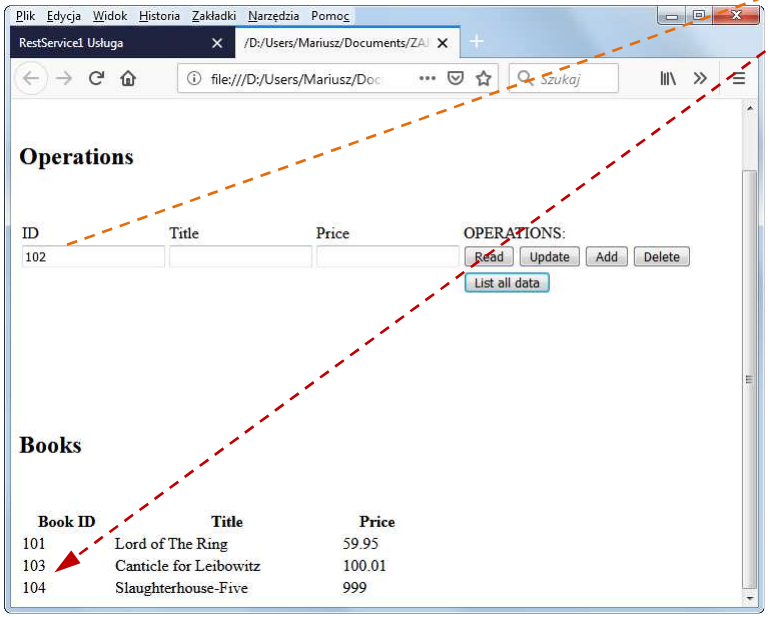



```
<html>
<head>
...
 function addJsonBook(item) {
  var book = Book;
  book.Id = $("#bid").val();
  book.Title = $("#title").val();
  book.Price = $("#price").val();
  var id = $("#bid").val();

  $.ajax({
    type: "POST",
    url: "http://localhost:3731/Service1.svc/json/books",
    data: JSON.stringify(book),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: function(msg) {
              alert('Added -> list data'); },
    error:   function(errMsg) {
                 alert(errMsg); }
  });
 }
 ...
</head>
<body>
 ...
    <td><input id="bid" type="text" /></td>
    <td><input id="title" type="text" /></td>
    <td><input id="price" type="text" /></td>
 ...
</body>
</html>
```

this may be ommited if sevice generates id

Example application

- **Delete item**




```
<html>
<head>
...
<script ...>
 ...
 function deleteJsonBook(item){
  var id = $("#bid").val();
  $.ajax({
   type: "DELETE",
   url: "http://localhost:3731/Service1.svc/json/books/"+id,
   dataType: "html",
   success: function(msg){
              alert('Usunieto - wyswietl liste');
            },
   error:   function(errMsg) {
              alert(errMsg);
            }
  });
 }
 ...
</script>
</head>
<body>
 ...
 <tr>
    <td><input id="bid" type="text" /></td>
    <td><input id="title" type="text" /></td>
    <td><input id="price" type="text" /></td>
 ...
</body>
</html>
```

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Komunikacja w systemach rozproszonych

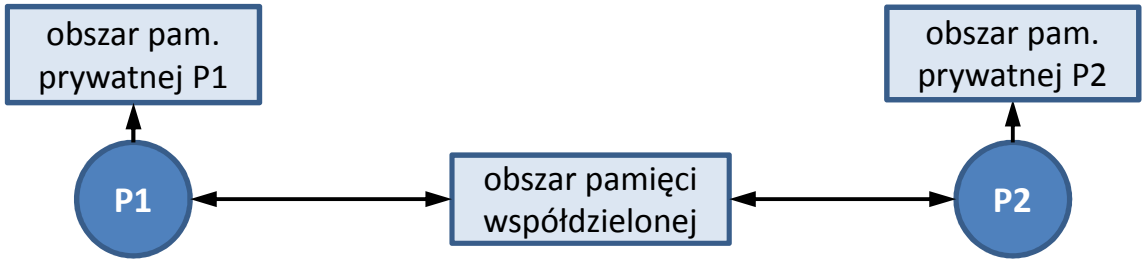
- Mechanizmy komunikacji
  - Przekazywanie komunikatów (ang. *message passing*)
- Schematy komunikacji
  - Koszty komunikacji

Usługi komunikacji

- **Komunikacja między rozproszonymi procesami**
  - Przenoszenie danych
  - Synchronizacja
  - Przenoszenie działania
- **Zagadnienia**
  - Modele komunikacji w SR
    - Model procesów komunikujących się poprzez pamięć dzieloną
    - Model procesów i komunikatów
  - Schematy komunikacji
    - Komunikacja punkt-punkt
    - Schematy komunikacji grupowej
  - Komunikacja w sieciach teleinformatycznych
    - Model warstwowy ISO/OSI
    - Warstwy i wydzielone funkcje (usługi)
    - Protokoły
    - Komunikacja połączeniowa i bezpołączeniowa
    - Enkapsulacja
    - Protokół TCP/IP

Usługi komunikacji

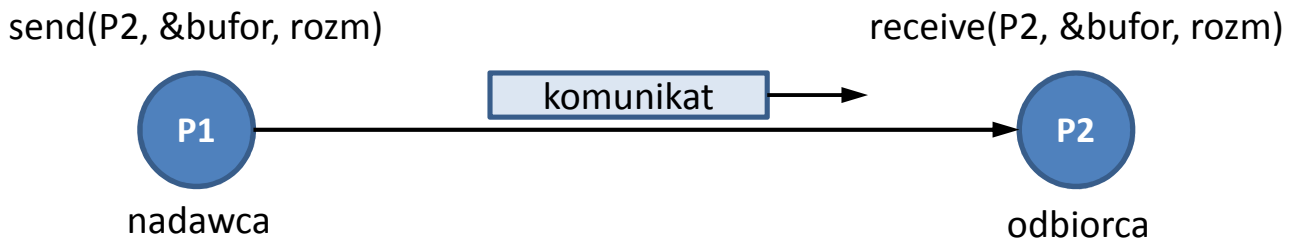
- **Komunikacja przez pamięć wspólną**
  - Aplikacja składa się ze zbioru procesów lub/i wątków
    - wykonywanych współbieżnie (ewentualnie też równolegle)
    - wykonujących się sekwencyjnie i używających swej pamięci lokalnej
  - Proces komunikuje się z innymi procesami za pomocą obszarów pamięci dzielonej



  - Dostęp do pamięci dzielonej jest synchronizowany za pomocą różnych narzędzi (muteksy, semafory, itd.)
  - Zazwyczaj w systemach ściśle powiązanych i nie rozproszonych

Usługi komunikacji

- **Przesyłanie komunikatów (ang. *Message passing*)**
  - Aplikacja składa się ze zbioru procesów sekwencyjnych używających swej pamięci lokalnej
  - Procesy mogą być wykonywane równolegle
  - Proces komunikuje się z otoczeniem za pomocą komunikatów
  - Są dwie podstawowe operacje komunikacyjne:



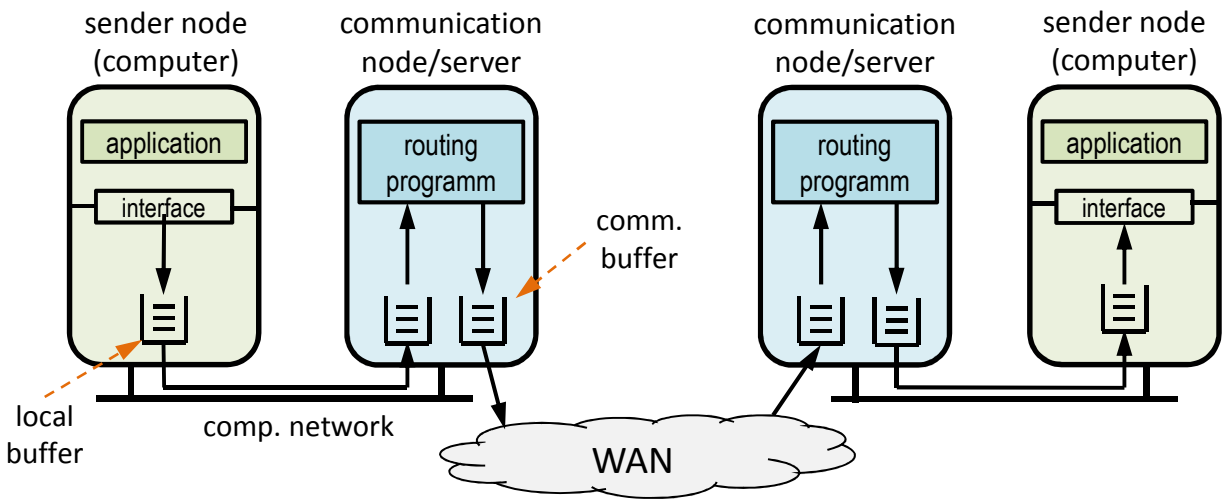
  - Przesłanie komunikatu = przesłanie pewnej liczby bajtów
  - Działania po stronie procesu nadawczego i odbiorczego:
    - Przenoszenie danych
    - Synchronizacja
  - Przesłanie komunikatu jest operacją atomową

Usługi komunikacji

- **Problemy związane przesyłaniem komunikatów**
  - Problem synchronizacji nadawcy i odbiorcy
    - Blokowanie – kto i kiedy czeka ?
    - Kwestie buforowania wiadomości
    - Czas trwania operacji
  - Problem trwałości komunikacji
    - Aktywność procesów podczas transferu wiadomości
    - Przechowywanie komunikatu w systemie
  - Problem adresowania – jaki system adresowania ?
  - Problem identyfikacji – czy procesy znają swoje identyfikatory ?
  - Problemy zapewnienia niezawodnego przesłania przez zawodny kanał komunikacyjny

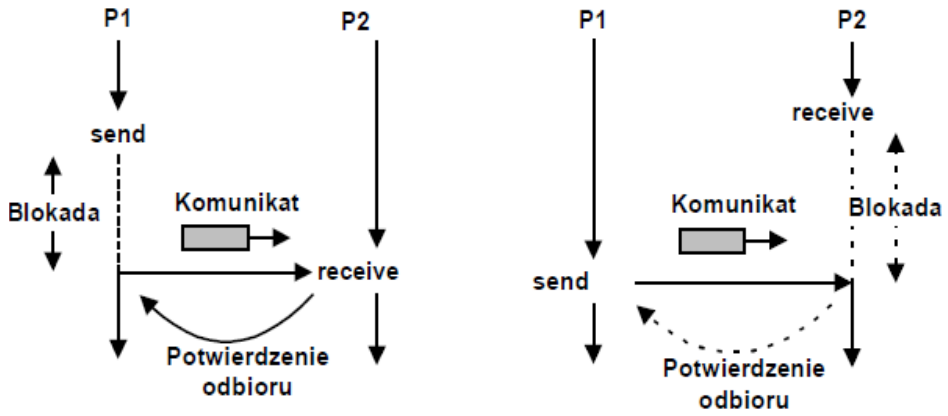
Schematy komunikacji

- **Komunikacja tymczasowa**
  - Komunikat jest usuwany jak tylko okazuje się, że nie może być dostarczony do funkcjonującego odbiorcy
- **Komunikacja trwała**
  - Komunikat jest przechowywany na serwerze komunikacyjnym tak długo jak zajmuje dostarczenie go do odbiorcy
    - Ewent. w buforach węzłów komunikacyjnych jeśli istnieje odp. oprogramowanie



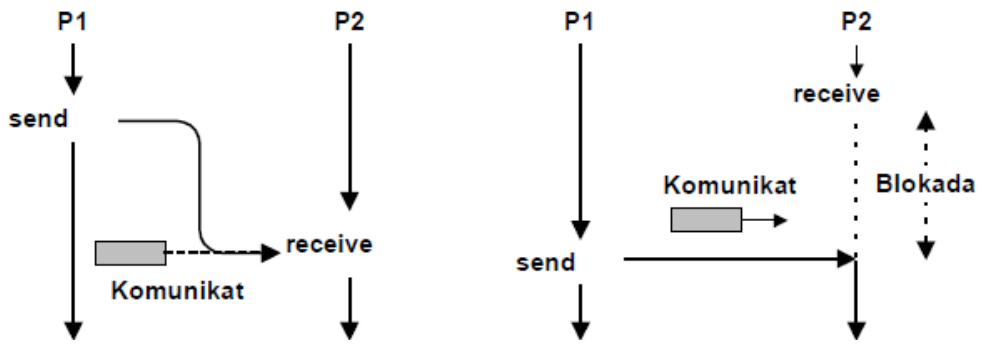
Schematy komunikacji

- **Komunikacja synchroniczna**
  - Nadawca jest blokowany dopóki jego komunikat nie trafi do bufora lokalnego odbierającego hosta lub nie zostanie rzeczywiście dostarczony do odbiorcy
  - Odbiorca blokowany jeśli pierwszy uruchomi operację *receive*
  - Obsługa błędów
    - **Przeterminowanie czasu oczekiwania** (ang. *timeout*)
  - Wady
    - Wstrzymywanie nadawcy podczas oczekiwania na odpowiedź
    - Z błędami trzeba radzić sobie natychmiastowo - bo nadawca czeka
    - Model taki czasami jest nieodpowiedni dla danego przetwarzania (np. poczta wiadomości)



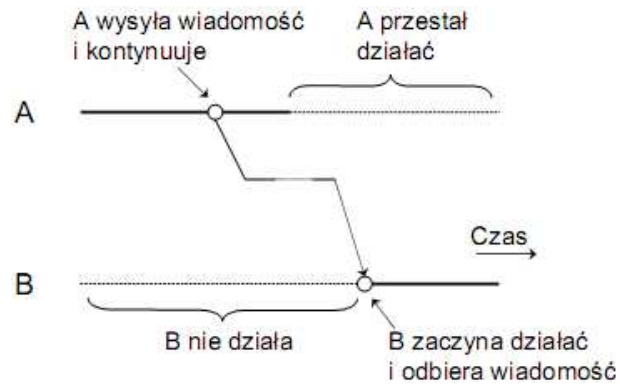
Schematy komunikacji

- **Komunikacja asynchroniczna**
  - Nadawca kontynuuje działanie natychmiast po tym, jak przekazał komunikat do transmisji
  - Zalety
    - Zwiększona efektywność działania nadawcy
    - Lepsza skalowalność systemu

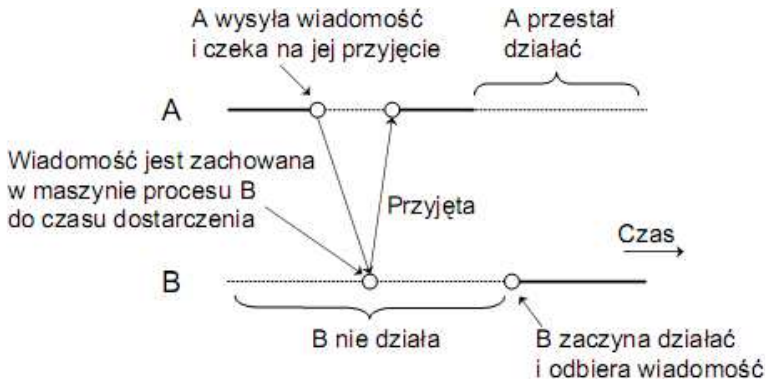


Schematy komunikacji

- **Trwała asynchroniczna**
  - Wiadomość jest zapamiętywana u nadawcy lub w serwerze komunikacyjnym
  - Podczas transferu wiadomości ani nadawca, ani odbiorca nie muszą być aktywni

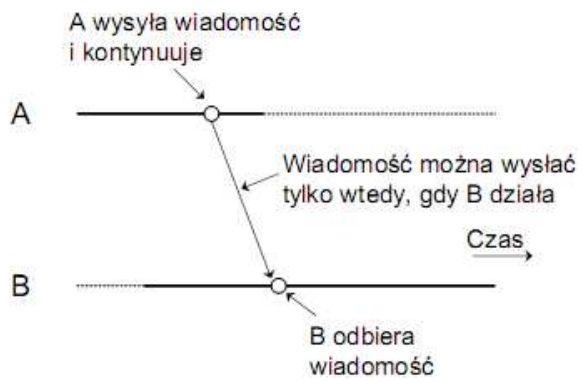


- **Trwała synchroniczna**
  - Nadawca jest blokowany do momentu otrzymania potwierdzenia zachowania wiadomości u odbiorcy
  - Odbiorca nie musi działać w momencie przybycia wiadomości

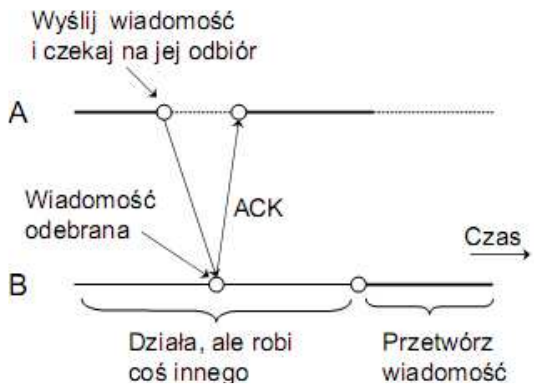


Schematy komunikacji

- **Nietrwała asynchroniczna**
  - Wiadomość tymczasowo zapamięty-wana w lokalnym buforze odbiorcy
  - Nadawca natychmiast kontynuuje pracę
  - Aplikacja odbiorcy powinna działać i oczekiwać na wiadomość
  - Jeśli nie: transfer nie udaje się

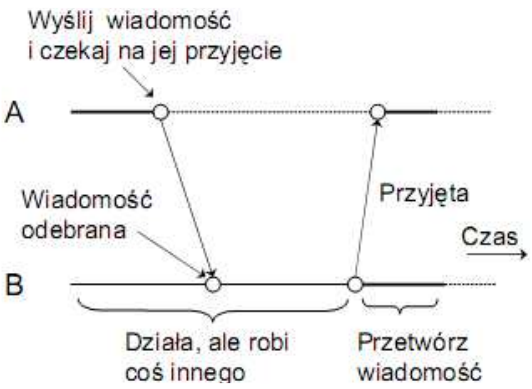


- **Nietrwała synchroniczna z potwierdzeniem**
  - Nadawca jest blokowany do momentu otrzymania potwierdzenia, że została zapamiętania wiadomość w lokalnym buforze komputera odbiorcy
  - Nadawca, po otrzymaniu potwierdzenia kontynuuje pracę

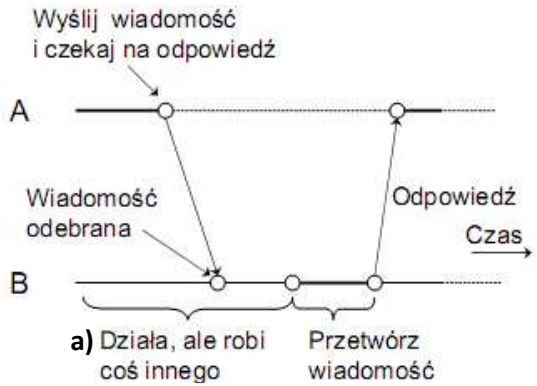


Schematy komunikacji

- **Nietrwała synchroniczna typu dostarczenie**
  - Nadawca jest blokowany do momentu otrzymania potwierdzenia, że wiadomość została dostarczona do aplikacji odbiorcy
  - Potwierdzenie może zostać wysłane przed przetworzeniem wiadomości



- **Nietrwała (a) / trwała (b) synchroniczna typu odpowiedź**
  - Nadawca jest blokowany do momentu, aż otrzyma potwierdzenie przetworzenia wiadomości przez aplikację odbiorcy
  - RPC, RMI



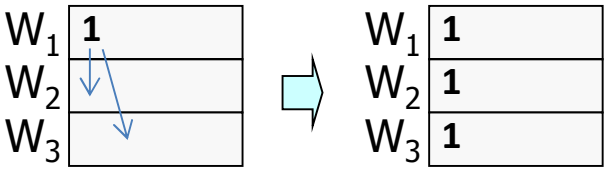
b) Nie musi działać

Schematy komunikacji

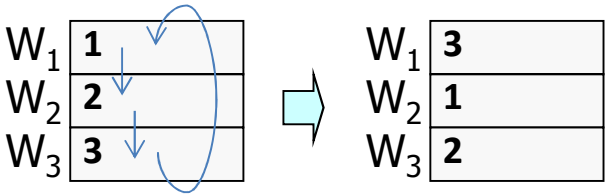
- **Adresowanie**
  - Do adresowania procesu docelowego stosuje się różne rozwiązania:
    - PID procesu
    - Porty
    - Obiekty
    - Gniazda (IP maszyny + numer portu TCP/UDP)
    - Grupy procesów
    - Nazwy symboliczne
  - Dąży się do uzyskania przeźroczystości położenia adresów
    - Adresowanie odbywa się w sposób symboliczny
    - Odwzorowanie adresu symbolicznego na fizyczny wykonywane jest przez odpowiednie warstwy oprogramowania.
  - Aby zastosować nazwy symboliczne konieczna jest **usługa nazewnicza** (ang. *Name service*)

Schematy komunikacji grupowej

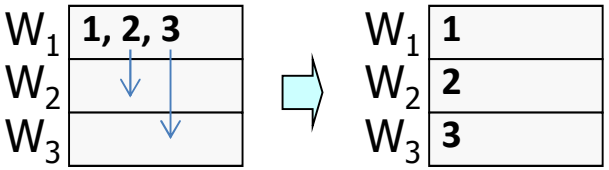
- **Schematy wymiany danych**
  - Broadcast



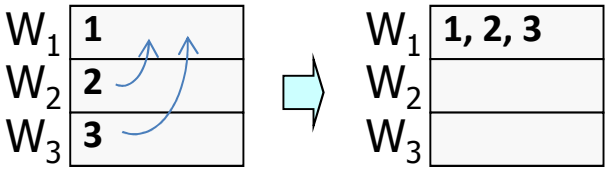
  - Shift



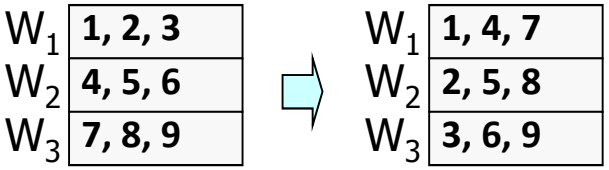
  - Scatter



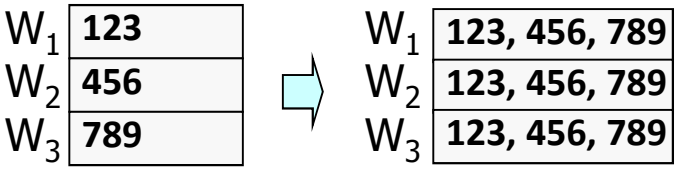
  - Gather



  - Scatter-gather



  - All-to-all / Total exchange



Koszty komunikacji

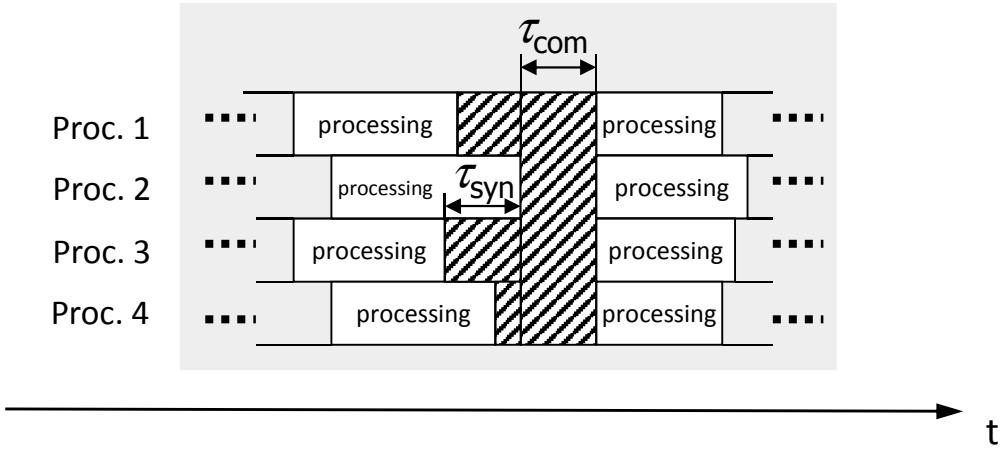
- **Koszty komunikacji w sieciach statycznych**
  - Czynniki
    - Przepustowość łączy (ang. bandwith)
    - Czas przygotowania (ang. startup time)
    - Liczba łączy (węzłów pośredniczących)
    - Rywalizacja o łącze (ang. link contention)
    - Rywalizacja w węźle (ang. node contention)
  - Podstawowe parametry
    - czas przygotowania / nawiązania połaczenia τ_s
    - czas przesłania nagłówka (ang. per-hop time) τ_h
    - czas przesłania jednostki danych (jednego słowa) τ_w
  - Dla komunikacji punkt-punkt

$$t = \underbrace{\tau_s + \tau_h}_{\text{bywa upraszczane jako jeden parametr}} + m \cdot \tau_w$$

m – liczba przesyłanych słów

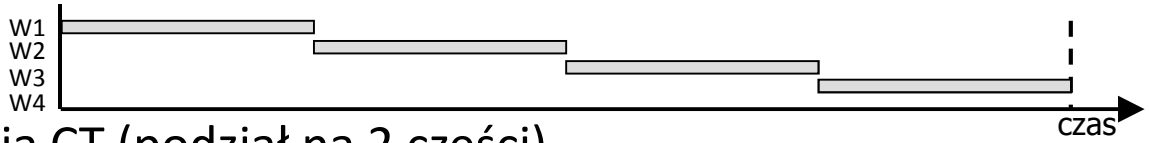
Koszty komunikacji

- **Koszty komunikacji w sieciach statycznych**
  - Modelowanie czasów komunikacji – czynniki:
    - Schemat komunikacji w aplikacji
      - Synchronizacja przetwarzania
      - Schemat wymiany danych
    - Struktura systemu przetwarzania
  - Synchronizacja
    - Oczekiwanie na osiągnięcie wspólnego momentu przetwarzania

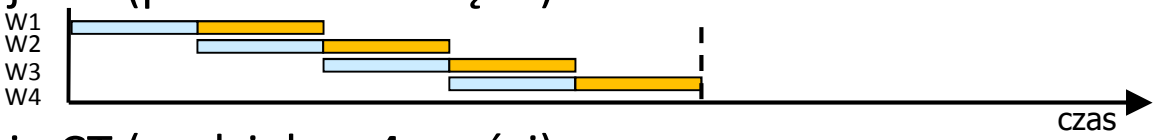


Koszty komunikacji

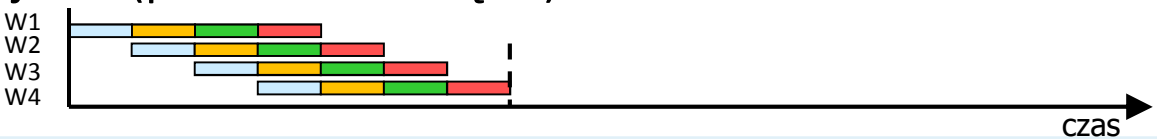
- Strategie przesyłania SF i CT
  - Store and Forward (SF – zapamiętaj i prześlij)
    - Każdy węzeł pośredniczący najpierw zapamiętuje cały komunikat w swojej lokalnej pamięci i dopiero potem wysyła komunikat dalej
  - Cut-Through (CT) routing
    - Komunikat jest dzielony na n części, które są przesyłane pomiędzy węzłami bez oczekiwania na odebranie całego komunikatu
- Ilustracja komunikacji dla strategii SF i CT
  - Komunikacja SF



  - Komunikacja CT (podział na 2 części)



  - Komunikacja CT (podział na 4 części)



Koszty komunikacji

- **Czas komunikacji dla strategii SF i CT**
  - Strategia SF

$$t_{SF} = \tau_s + n \cdot (\tau_h + m \cdot \tau_w)$$

  - Strategia CT

$$t_{CT} = \tau_s + n \cdot \tau_h + m \cdot \tau_w$$

- **Czas wymiany danych**
  - Zależny od schematu komunikacji
    - Rozpatrywane są różne typowe schematy wymiany danych
  - Musi uwzględniać strukturę sieci
    - Są ograniczenia dla zasobów (np. liczba niezależnych połączeń)

Architektury systemów

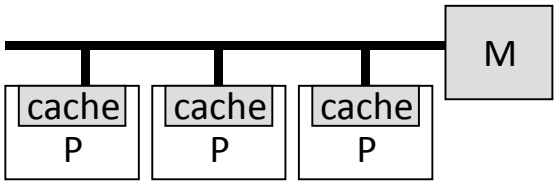
- **Struktura sieci połączeń**
  - Sposób połączenia procesorów ze sobą definiuje topologię sieci połączeń
  - Różne topologie nadają się do różnych rodzajów przetwarzania (obliczeń)

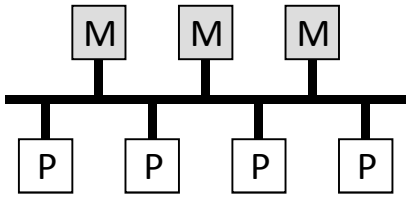
  - **Statyczne sieci połączeniowe**
    - Łatwiejsze do realizacji
    - Zwykle systemy rozproszone z pamięcią lokalną

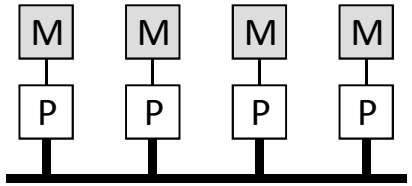
  - **Dynamiczne sieci połączeniowe**
    - Zwykle dla systemów ściślej powiązanych z pamięcią współdzieloną
    - Różne wymagania przepustowości sieci połączeniowej

Architektury systemów

- **Wspólna magistrala (szyna)**
  - Jedna sieć połączeń o architekturze szyny
  - Prosta i tania architektura
  - Czas oczekiwania na dostęp do pamięci – wąskie gardło
    - Zapis danych – rozwiązania:
      - Deterministyczny (priorytetowy)
      - Grupowy (łączność operacji → redukcja odwołań)
      - Niedeterministyczny (losowy proces wykonuje operację)
    - Stosowanie pamięć podręcznej (*cache*)
      - Lokalizacja odwołań moża usprawnić komunikację
      - Występuje problem spójności danych



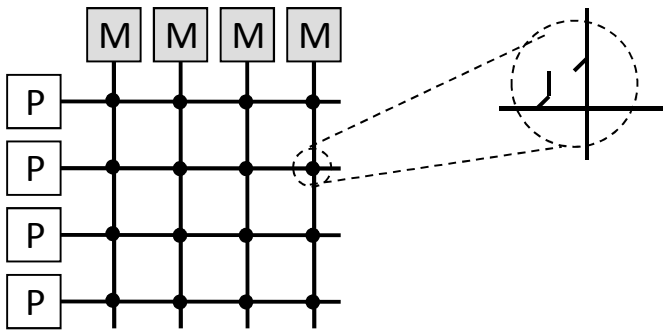




  - HP (HP9000),
    Silicon Graphics (Power Challenge)

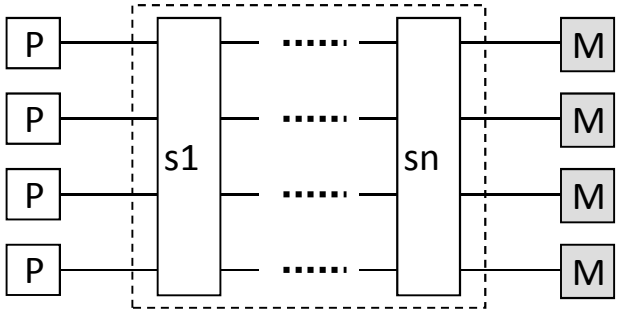
Dynamiczne sieci połączeniowe

- **Przełącznica krzyżowa (krzyżakowa)**
  - Sieć bez blokad
  - Drogie rozwiązanie
    - $O(p^2)$
  - Przyk.: Cray-YMP, Fujitsu VPP 500

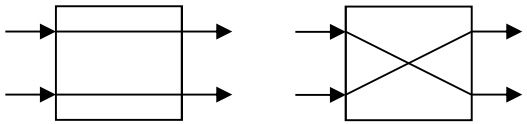


Dynamiczne sieci połączeniowe

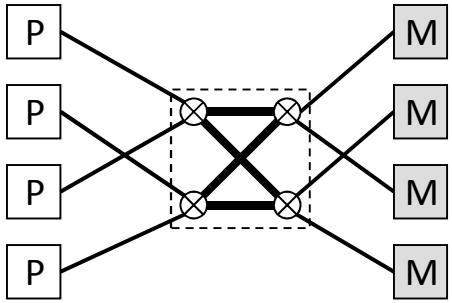
- **Wielostanowa sieć połączeń**
  - Pośrednie rozwiązanie
  - Przyk.: IBM RP-3



  - Sieć Omega / Ω / butterfly
    - Przełącznik
      - 2 stany:

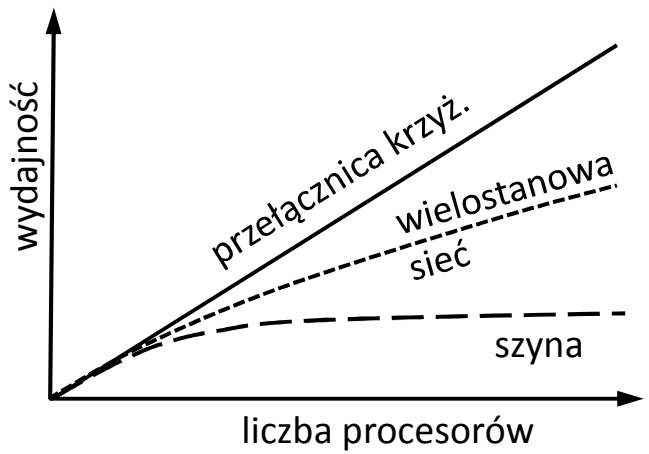


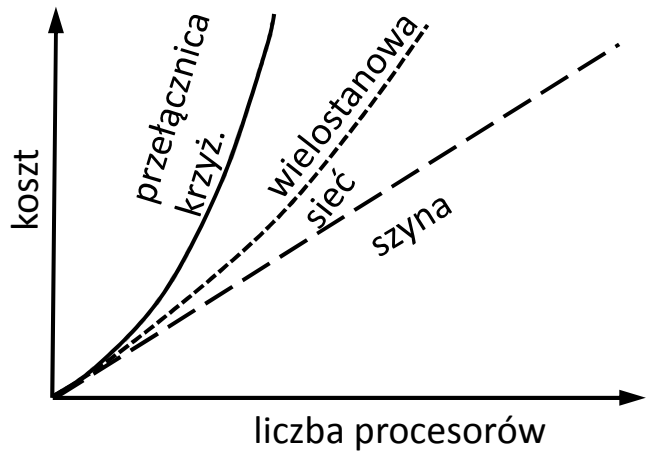
    - $p/2 \cdot \log_2 p$ elem. przełączających
    - $O(p \cdot \log_2 p)$



Dynamiczne sieci połączeniowe

- **Kryteria oceny**
  - Koszt podstawowy - połączenia stanowisk w jeden system
  - Wydajność
    - Koszt komunikacji
  - Niezawodność – wpływ awarii na działanie całego systemu
- **Koszt a wydajność dla sieci dynamicznych**



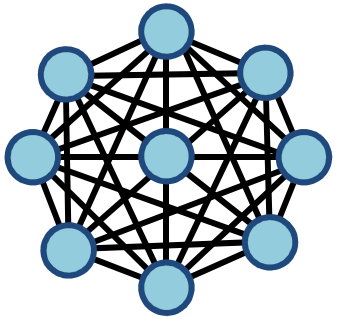


Topologie sieci statycznych

- **Graf struktury systemu przetwarzania**
  - $G\langle V, A\rangle$
  - $V_i = \{v_1, v_2, ..., v_i, ...v_I\}$ – wierzchołki (węzły - procesory)
  - $(v_i, v_j) \in A, \quad A \subseteq V \times V$ – krawędzie (powiązań) lub łuki
- **Kryteria oceny**
  - Maksymalny (minimalny) stopień wierzchołków
    - Wpływa na złożoność procedur komunikacyjnych $\rightarrow$ wydajność
  - Średnica / Maksymalna długość ścieżki
    - Im mniejsza tym lepiej
  - Szerokość pasma bisekcji
    - Minimalna liczba krawędzi koniecznych do usunięcia aby dokonać podziału sieci na dwie równe podsieci rozłączne
    - Miara m.in. charakteryzująca przepustowość $\rightarrow$ wydajność
  - Spójność krawędziowa
    - Min. Liczba krawędzie aby sieć stała się niespójna $\rightarrow$ odporność na uszk.
  - Liczba krawędzi
    - Najprostsza miara wyrażająca koszt sieci

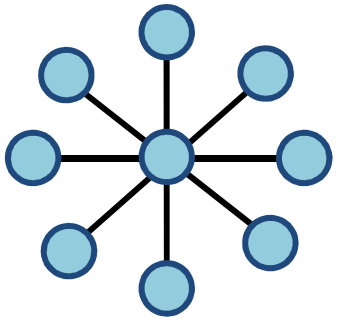
Topologie sieci statycznych

- **Połączenie pełne – sieć pełna**
  - koszt podstawowy – proporcjonalny do p^2
  - koszt komunikacji - niski
  - niezawodność - bardzo duża



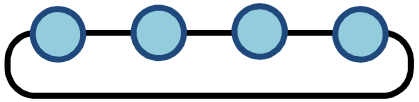
- **Połączenie częściowe**
- **Szyna wielodostępna – magistrala (ang. *Bus*)**
  - koszt podstawowy – liniowy (~p)
  - koszt komunikacji – zróżnicowany
  - niezawodność
    - awaria stanowiska → zakłócenie komunikacji
    - awaria szyny → rozbicie sieci

- **Gwiazda (ang. *Star*)**
  - koszt podstawowy – liniowy (~p)
  - koszt komunikacji - niski
    węzeł centralny może być wąskim gardłem
  - niezawodność: kluczowe stanowisko centralne

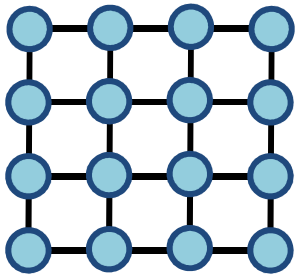


Topologie sieci statycznych

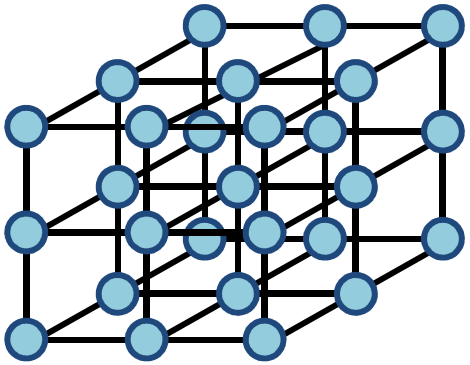
- **Pierścień (ang. Ring)**
  - koszt podstawowy – liniowy (~p)
  - koszt komunikacji - może być wysoki
  - niezawodność: awaria $\rightarrow$ rozbicie sieci



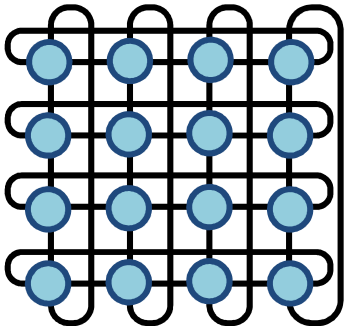
- **Krata (ang. Mesh)**
  - Mesh 2D



  - Mesh 3D

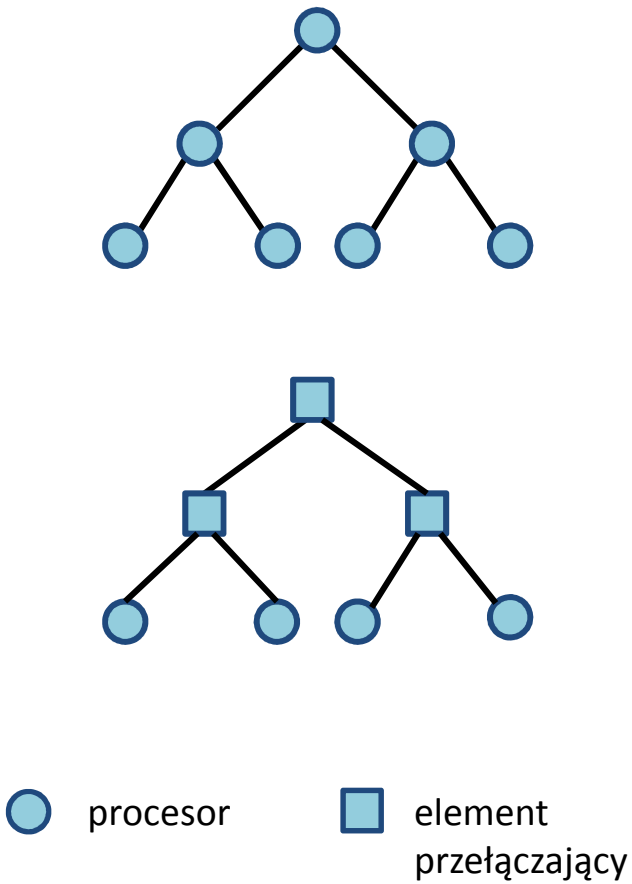


- **Torus**
  - 2D



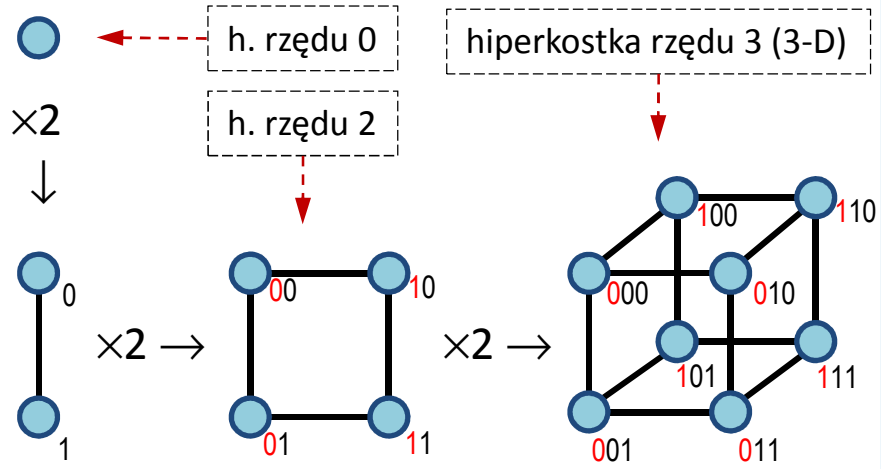
Topologie sieci statycznych

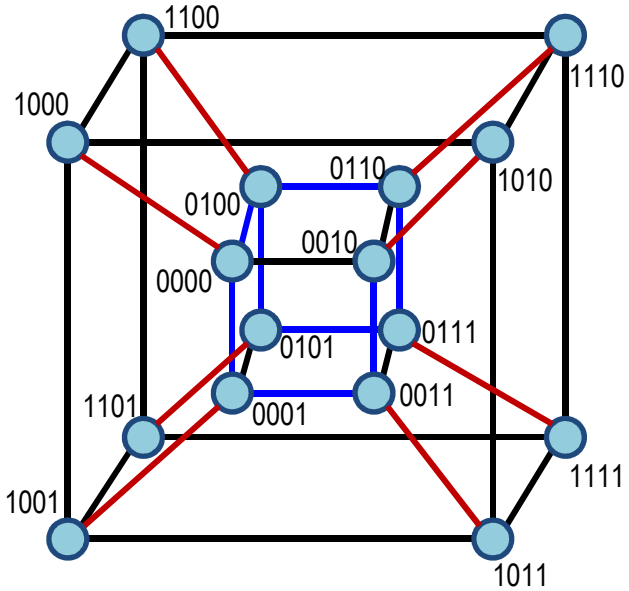
- **Hierarchia drzewo**
  - koszt podstawowy
    - mniejszy niż dla sieci pełnej
    - na ogół mniejszy niż dla sieci częściowej
  - koszt komunikacji
    - wyższy niż dla sieci pełnej
  - Niezawodność
    - awaria dowolnego węzła (z wyjątkiem liścia) powoduje częściowe rozbicie sieci



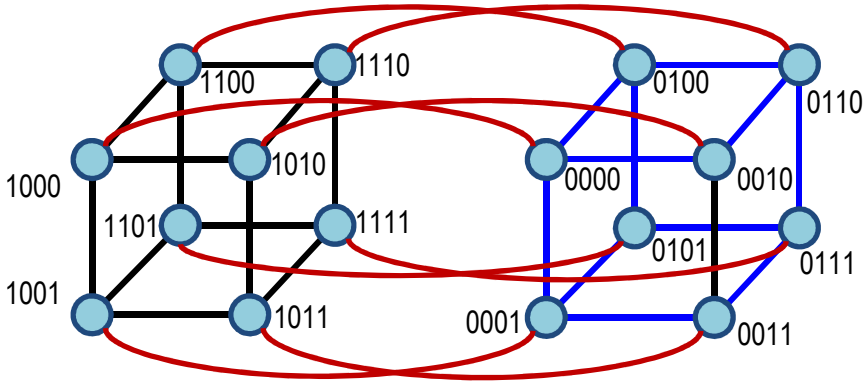
Topologie sieci statycznych

- **Hiperkostka (ang. Hypercube)**
  - Rząd (wymiar) sieci n-D
    - 2-D, 3-D, itd.
  - Istotne własności dla kolektywnej wymiany danych





≡



Topologie sieci statycznych

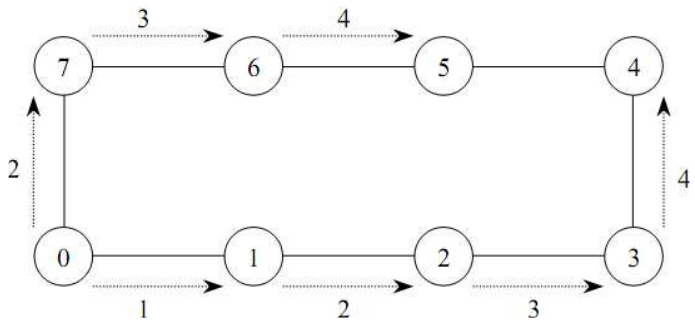
- **Porównanie rozwiązań**

Topologia	Maks. długość ścieżki	Szerokość pasma bisekcji	Maks. stopień wierzch.	Liczba krawędzi
Pełna sieć	1	$p^2/4$	p-1	$p\cdot(p-1)/2$
Gwiazda	2	1	1 / p-1	p-1
Drzewo binarne (pełne)	$2\cdot\log((p+1)/2)$	1	3	p-1
Magistrala	p-1	1	2	p-1
Pierścień	$p/2$	2	2	p
Mesh 2-D	$2\cdot(\sqrt{p} - 1)$	$\sqrt{p}$	4	$2\cdot(p - \sqrt{p})$
Torus 2-D	$2\cdot(\sqrt{p}/2)$	$2\cdot \sqrt{p}$	4	$2\cdot$p
Hiperkostka	$\log(p)$	$p/2$	$\log(p)$	$p\cdot\log(p)/2$

Globalna wymiana danych

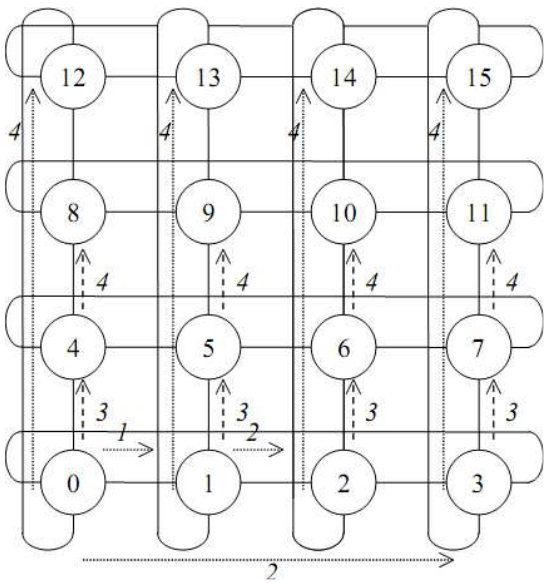
- **Pierścień**
  - Broadcast
  - SF
  - Czas:

$$T = (\tau_s + m \cdot \tau_w) \cdot \left\lceil \frac{p}{2} \right\rceil$$



- **Torus 2-D**
  - Broadcast
  - SF
  - Czas:

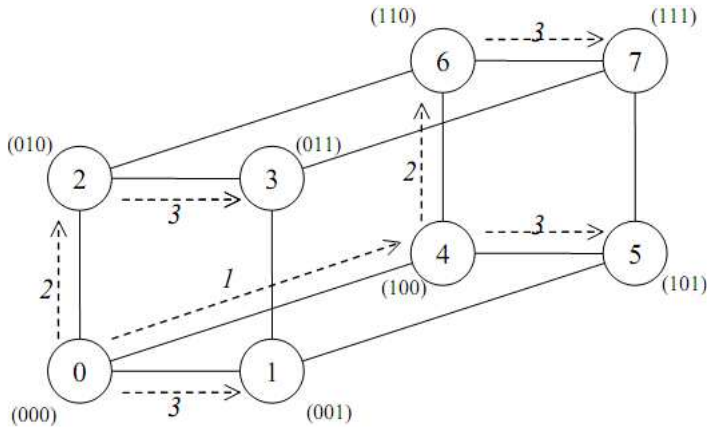
$$T = 2 \cdot (\tau_s + m \cdot \tau_w) \cdot \left\lceil \frac{\sqrt{p}}{2} \right\rceil$$



Globalna wymiana danych

- **Hiperkostka**
  - Broadcast
  - SF
  - Czas:

$$T = (\tau_s + m \cdot \tau_w) \cdot \log p$$



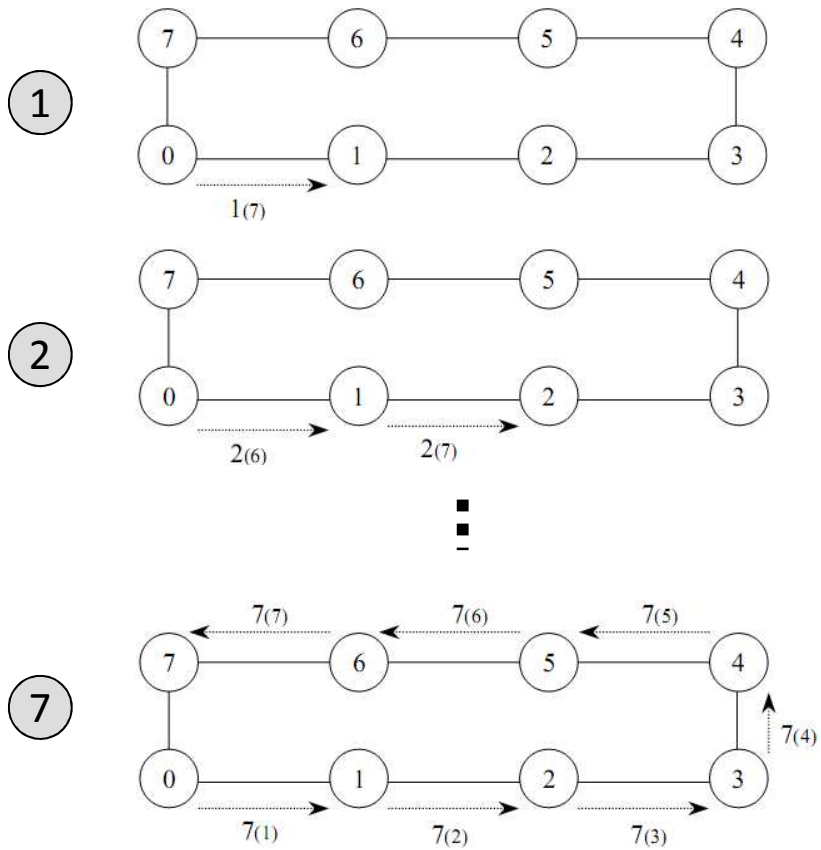
Globalna wymiana danych

- **Pierścień**
  - Scatter
  - SF
  - Czas:

$$T = (\tau_s + m \cdot \tau_w) \cdot (p-1)$$

dla: $M = m \cdot p$

$$T = \tau_s \cdot (p-1) + M \cdot \tau_w \cdot \frac{p-1}{p}$$



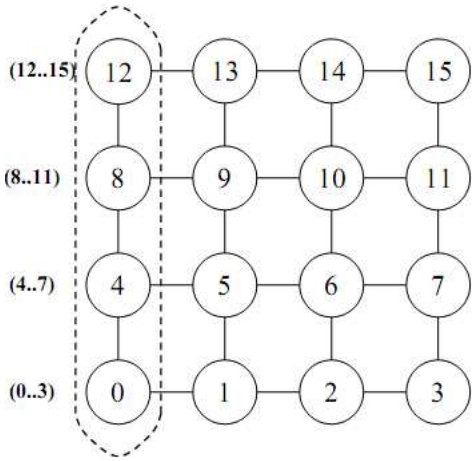
Globalna wymiana danych

- **Torus 2-D**
  - Scatter
  - SF
  - Czas:

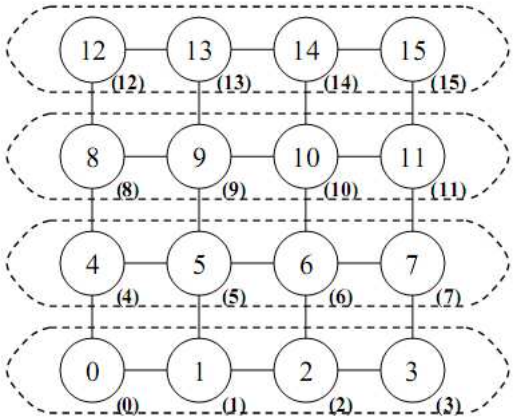
$$T = 2 \cdot \tau_s \cdot (\sqrt{p} - 1) + m \cdot \tau_w \cdot (p - 1)$$

$$T = 2 \cdot \tau_s \cdot (\sqrt{p} - 1) + M \cdot \tau_w \cdot \frac{p-1}{p}$$

1



2

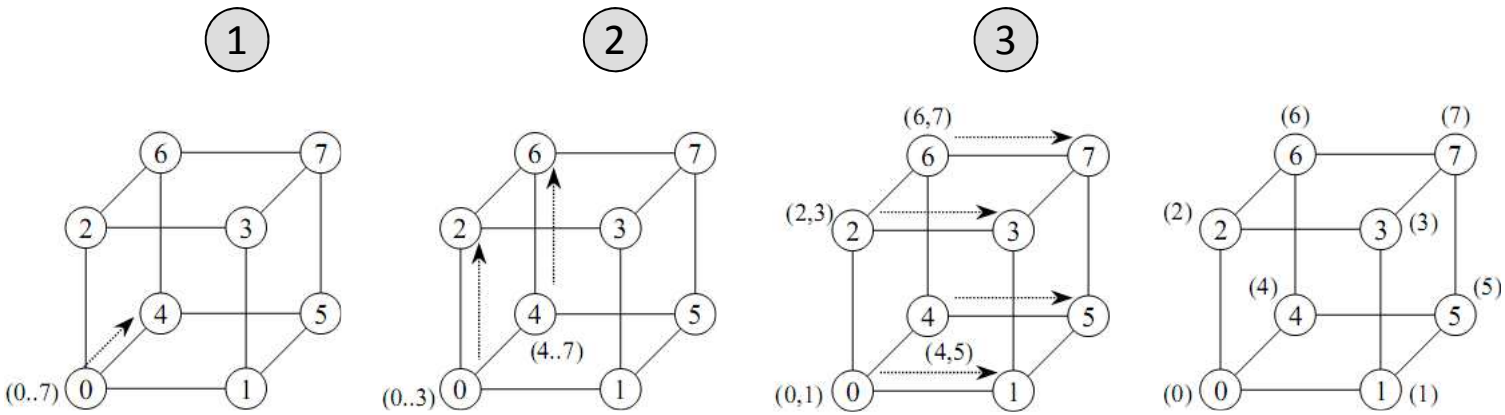


Globalna wymiana danych

- **Hiperkostka**
  - Scatter
  - SF
  - Czas:

$$T = \tau_s \cdot \log p + m \cdot \tau_w \cdot (p-1)$$

$$T = \tau_s \cdot \log p + M \cdot \tau_w \cdot \frac{(p-1)}{p}$$



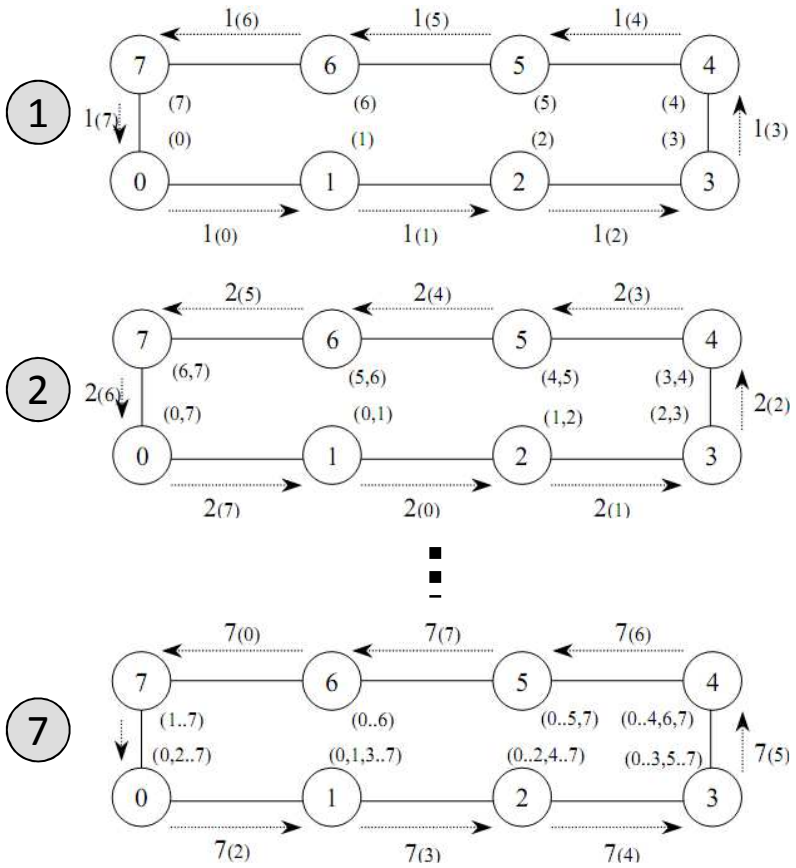
Globalna wymiana danych

- **Pierścień**
  - All-to-all
  - SF
  - Czas:

$$T = \tau_s \cdot (p-1) + m \cdot \tau_w \cdot (p-1)$$

$$T = \tau_s \cdot (p-1) + M \cdot \tau_w \cdot \frac{(p-1)}{p}$$

    - Przy założeniu że każdy proc. produkuje *m* danych



- Porównanie:
  - magistrala współdzielona

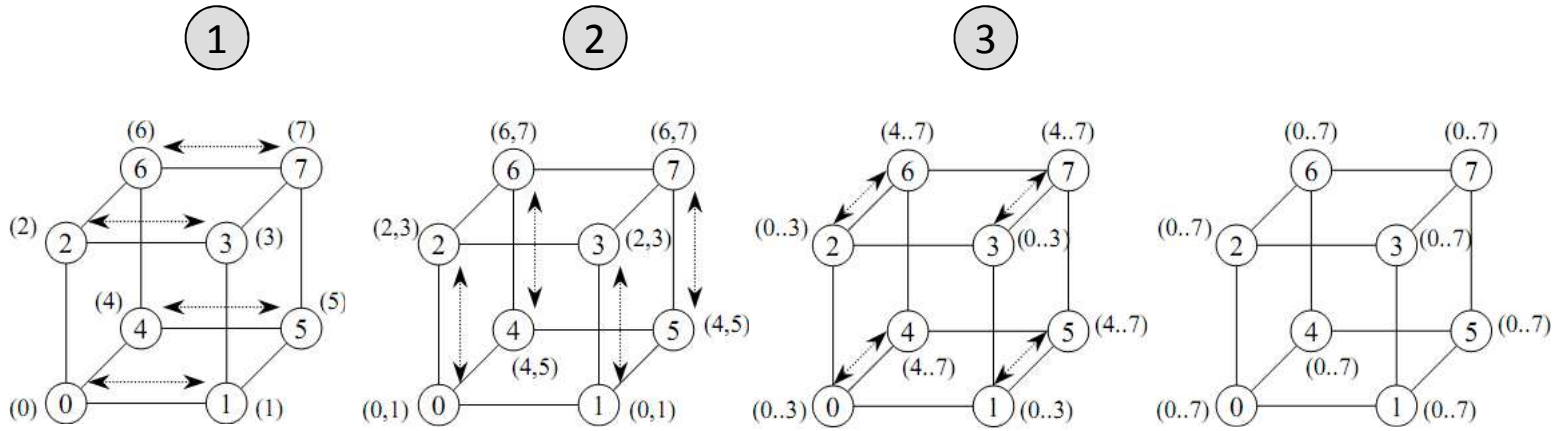
$$T = 2 \cdot \tau_s \cdot (p-1) + m \cdot \tau_w \cdot (p-1) \cdot p$$

Globalna wymiana danych

- **Hiperkostka**
  - All-to-all
  - SF
  - Czas:

$$T = \tau_s \cdot \log p + 2 \cdot m \cdot \tau_w \cdot (p-1)$$

$$T = \tau_s \cdot \log p + 2 \cdot M \cdot \tau_w \cdot \frac{(p-1)}{p}$$

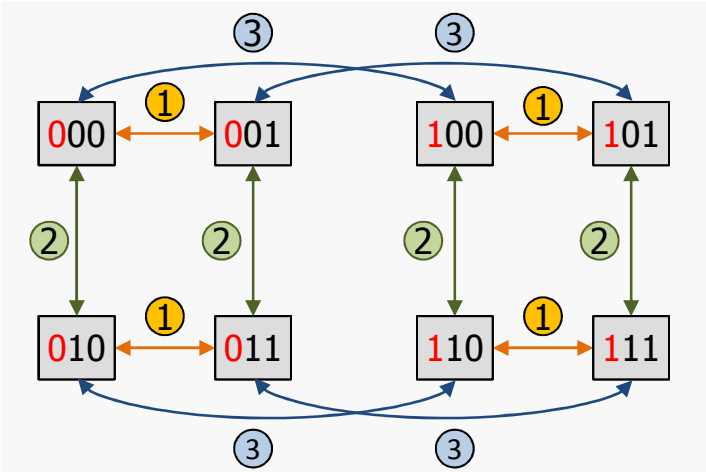


Globalna wymiana danych

- **Procedura wymiany all-to-all dla hiperkostki**
  - Oznaczenia
  - p – liczba węzłów $p=2^R$, $R=\log_2 p$
  - $\omega(Pu,Pv)$ – operacja przesłania wszystkich danych z procesora P_u do P_v i odwrotnie, $u, v \in [1,p]$
  - Procedura R-fazowej wymiany danych

```
procedure wymiana;
begin
  result = 0;
  for r=1 to log2p do
  begin
    msg = msg ∪ result;
    for u=1 to p do in parallel
    begin
      Pv = Pu xor 2^(r-1);
      result = ω(Pu,Pv , msg);
    end;
  end;
end;
```

$\mathbf{p_u}, \mathbf{p_v}$ – binarne identyfikatory węzłów



MPI

- **MPI – ang. Message Passing Interface**
  - Specyfikacja biblioteki funkcji opartych na modelu wymiany komunikatów – zestaw ustandaryzowanych procedur
    - MPI nie jest konkretnym produktem, biblioteką ani środowiskiem programowym
    - Fortran, C, C++
  - Wspiera model programowy dla:
    - Systemy z pamięcią rozproszoną (niezależne procesy we własnej przestrzeni adresowej)
    - Systemy z pamięcią współdzieloną
    - Mechanizmy komunikacji punkt-punkt oraz komunikacji grupowej
  - Wiele implementacji producentów systemów i public domain
    - MPICH, LAM, …
- **Implementacja dla ANSI C**
  - Nazwy funkcji z przedrostkiem MPI_

```
return_code = MPI_Xxx( param1, … )
```

MPI

- **Komunikacja**
  - Komunikacja procesów w ramach tzw. **komunikatora**
    - Określa grupę procesów, które poprzez niego mogą komunikować się
  - **Grupa** - uporządkowany zbiór identyfikatorów procesów
    - Proces w grupie posiada identyfikator – tzw. **rank** (liczba całkowita)
    - Używana w komunikatorze określającym komunikujące się procesy
  - Każdy proces może poznać liczbę procesów skojarzonych z komunikatorem
  - Komunikacja punkt-punkt
    - Komunikacja z blokowaniem
      - `MPI_Send(buffer,count,type,dest,tag,comm)`
      - `MPI_Recv(buffer,count,type,source,tag,comm,status)`
      - `...`
    - Komunikacja bez blokowania
      - `MPI_Isend(buffer,count,type,dest,tag,comm,request)`
      - `MPI_Irecv(buffer,count,type,source,tag,comm,request)`
      - `...`
        - Wymagają testowania zakończenia transmisji

MPI

- **Przetwarzanie w grupie procesów**
  - Mechanizmy wspólnego przetwarzania
    - W grupie procesów identyfikowanej przez komunikator
  - Dostępne mechanizmy:
    - Komunikacja grupowa
      - `MPI_Bcast(...)`
      - `MPI_Gather(...)`
      - `MPI_Scatter(...)`
      - `MPI_Allgather(...)`
      - `MPI_Alltoall(...)`
    - Operacje obliczeń globalnych
      - Jednoczesne obliczenia (różne operacje) na danych dostarczonych przez wszystkich członków grupy
      - `MPI_Reduce (..., op, ...)`
      - `MPI_Allreduce (..., op, ...)`
    - Mechanizm synchronizacji (barier)
      - `MPI_Barrier(comm)`
      - Funkcja wstrzymuje wykonanie procesu do momentu, gdy wszystkie pozostałe procesy w grupie wywołają tę funkcję

MPI

- **Ilustracja – prosty przykład wymiany komunikatów (przesłanie znaku)**

```
#include "mpi.h"
#include <stdio.h>

main(int argc, char *argv[]) {
  int numtasks, rank, dest, source, rc, tag=1;
  char inmsg, outmsg='x';
  MPI_Status Stat;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf ("Num. of processes = %d. My rank = %d\n", numtasks, rank);
  if (rank == 0) {
    dest = 1;
    source = 1;
    rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
    rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  }
  else if (rank == 1) {
    dest = 0;
    source = 0;
    rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
    rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }
  MPI_Finalize();
}
```

MPI

- **Ilustracja - przykład – komunikacja scatter**

```
#define SIZE 4

int main(int argc; char *argv[]) {
  int numtasks, rank, sendcount, recvcount, source;
  float sendbuf[SIZE][SIZE] = {
    {1.0, 2.0, 3.0, 4.0},
    {5.0, 6.0, 7.0, 8.0},
    {9.0, 10.0, 11.0, 12.0},
    {13.0, 14.0, 15.0, 16.0}  };
  float recvbuf[SIZE];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

  if (numtasks == SIZE) {
    source = 1;
    sendcount = recvcount = SIZE;
    MPI_Scatter(sendbuf, sendcount, MPI_FLOAT,
                recvbuf, recvcount, MPI_FLOAT, source, MPI_COMM_WORLD);
    printf(" My rank = %d. \n Results: %f %f %f %f\n", rank,recvbuf[0],
           recvbuf[1], recvbuf[2], recvbuf[3]);
  } else
    printf("Must specify %d processors. Now terminating...\n", SIZE);

  MPI_Finalize();
}
```

Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Koordynacja w systemach rozproszonych

- Synchronizacja czasu
  - Zegary logiczne
  - Stan globalny
- Wzajemne wykluczanie
  - Zakleszczenia

Koordynacja w systemach rozproszonych

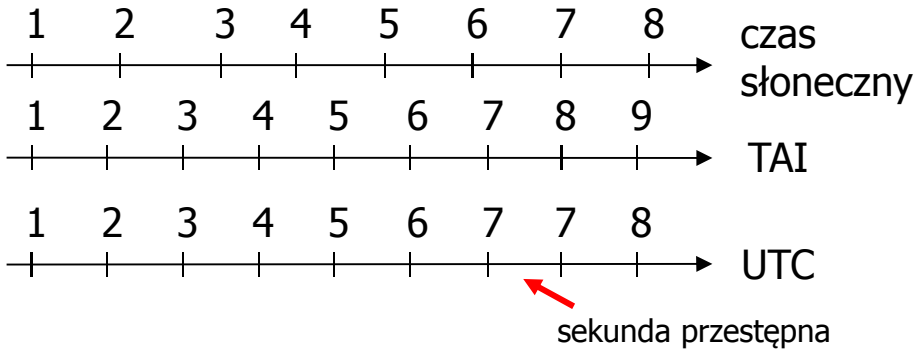
- **Wybrane zagadnienia**
  - Synchronizacja czasu
  - Mechanizmy wzajemnie wykluczającego się dostępu do zasobów
  - Mechanizmy postępowania z zakleszczeniami
  - Transakcje rozproszone
  - Elementy wiarygodności funkcjonalnej
    - Osiąganie porozumienia
    - Rozsyłanie uporządkowane
    - Zatwierdzanie transakcji
  - Ustalenie stanu globalnego systemu

Synchronizacja czasu

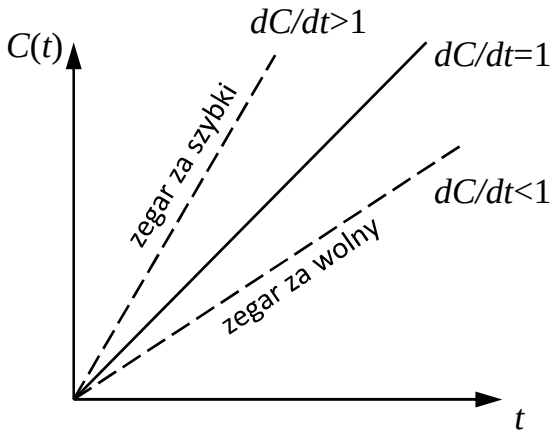
- **Odmierzanie czasu**
  - Wymagania temporalne w systemach informatycznych
    - Potrzeba określenia dokładnego momentu operacji
    - Wiele algorytmów zależy od synchronizacji
  - Problem jednoznaczności określenia momentów zdarzeń
    - System scentralizowany – czas określony jednoznacznie
    - System rozproszony
      - Nierówne odmierzanie czasu
      - Niezerowy, nieokreślony czas przesyłania komunikatów
      - Problem jednolitego określania momentów czasu przez zegary lokalne
- **Zegary fizyczne**
  - Układy zegarowe (*Timers*) w systemach informatycznych
- **Zegary logiczne**
  - Brak potrzeby znajomości czasu rzeczywistego
  - Upływ czasu nieautonomiczny – zależy od występujących zdarzeń

Synchronizacja czasu

- **Zegary fizyczne**
  - Czas rzeczywisty – czas atomowy (*International Atomic Time*) - TAI
  - Uniwersalny czas koordynowany (*Universal Coordinated Time*) - UTC
    - UTC = TAI + sekundy przestępne
    - Sygnały UTC rozgłaszane radiowo (stacje radiowe, satelity)

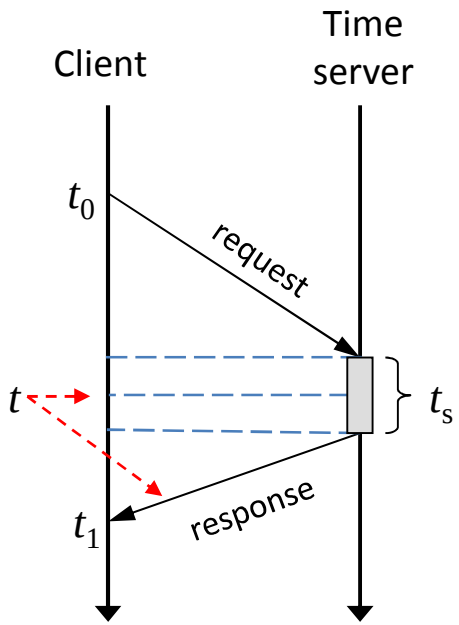


- Błędy wskazań zegarów fizycznych
  - Maks. tempo odchylenia: $1-r \le \frac{dC(t)}{dt} \le 1+r$
  - Maks. różnica po czasie Δt: $\delta = 2 \cdot r \cdot \Delta t$
  - Potrzeba synchronizacji wskazań zegarów



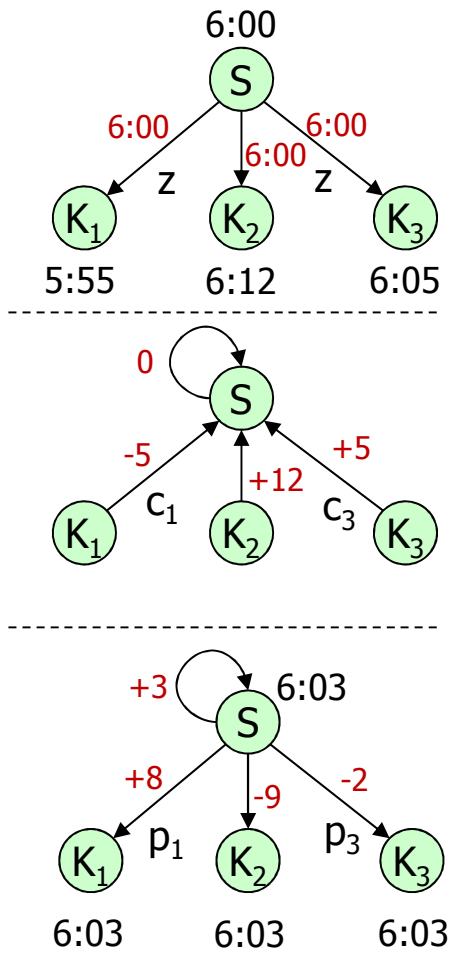
Synchronizacja czasu

- **Algorytm Cristiana**
  - Synchronizacja z serwerem czasu – np. z węzłem z odbiornikiem WWV
  - Zapewnienie błędu $\leq \delta$
    - Synchronizacja co $T_s = \delta/(2 \cdot r)$
  - Korekta
    - $C(t) = t + (t_1 - t_0) / 2$
    - $C(t) = t + (t_1 - t_0 - t_s) / 2$
  - Wymaganie monotoniczności zegara
    - Regulacja szybkości zegara
  - Fluktuacje czasu transmisji komunikatu
    - $t_1 - t_0 > t_{gr} \Rightarrow$ korekta odrzucana
    - Serie pomiarów
      - Średnia korekt
      - Najkrótszy czas komunikacji
  - Wada
    - Zależność od jednego węzła
      - Możliwość awarii lub błędnego działania



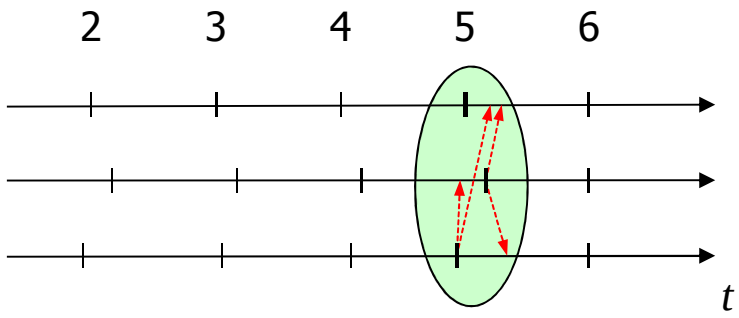
Synchronizacja czasu

- **Algorytm Berkeley**
  - Dla zespołu współpracujących węzłów K_i
  - Serwer S jest aktywny
  - Odpytuje okresowo pozostałe węzły o ich czas
  - Uśrednia i wysyła poprawkę korygującą
    - $C(t) = \mathrm{avg}(C_i(t))$
    - $\Delta_i = C(t) - C_i(t))$
  - Węzły wprowadzają poprawkę
    - $C_i(t) = C_i(t) + \Delta_i$
  - Problem czasów dostarczenia komunikatów
  - Warianty
    - Średnia tolerująca awarie
    - Uwzględnianie w puli czasu serwera
    - Rozsyłanie czasu serwera i przesyłanie różnic czasowych / korekt



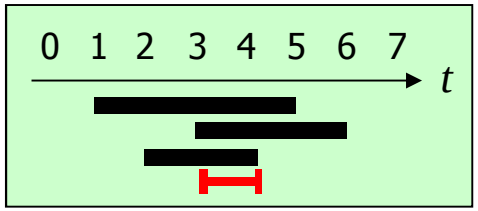
Synchronizacja czasu

- **Algorytm uśredniania**
  - Algorytm zdecentralizowany
  - Rozgłaszanie aktualnych czasów w momentach $T_0 + i \cdot T_P$ (i=1,2,…)
    - T_p – okres synchronizacji
  - Zbieranie innych rozgłoszeń
    - Nadchodzą w różnych momentach
    - Mogą być znakowane identyfikatorem
  - Korekta lokalnego zegara
    - Średnia: $\Delta_i = \mathrm{avg}(C_n(t)) - C(t_i)$
    - Korygowanie: $C(t_i) = C(t_i) + \Delta_i$
  - Ulepszenia:
    - Odrzucanie wartości skrajnych (potencjalnie całkowicie błędnych)
    - Korekcja o oszacowanie czasu transmisji komunikatu



Synchronizacja czasu

- **Algorytm Marzullo**
  - Oszacowanie czasu na podstawie kilku źródeł
    - Czas w postaci przedziału $[t - d , t + d]$
  - Znalezienie najmniejszego przedziału [***begin, end***] mającego część wspólną z jak największą liczbą źródeł



  - Algorytm:
    - Przedział czasu reprezentowany przez dwie pary w formacie <*time, type*>

$$[<t - d , +1> <t + d , -1>]$$

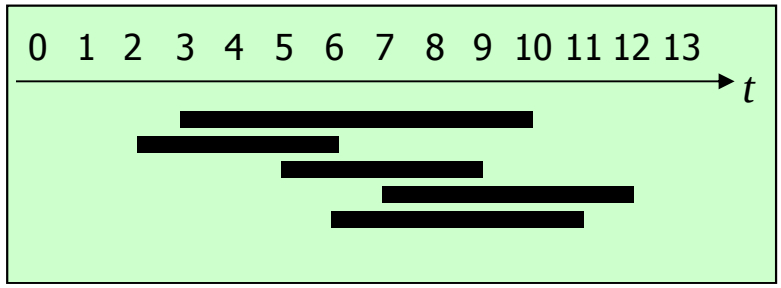
      › początek: *type* = +1, koniec: *type* = -1
    - Sortowanie par wg wartości *time*
    - Analiza kolejnych momentów czasowych początków lub końców przedziałów
      › Sumowanie wartości *type* – suma tych wartości wskazuje ile przedziałów zawiera obliczony przedział
      › Zapamiętanie nowego przedziału gdy suma zwiększa się
    - Czas jest funkcją przedziału [*begin*, *end*]
      › $C(t) = f(begin , end)$
      › np.: $C(t) = (end - begin) / 2$

```
zbuduj listę par
              <time,type>
sortuj listę par wg time
best = 0;
curr = 0;
for i=1 to i=liczba_par
  curr = curr+type[i];
  if curr > best do
    best = curr;
    begin = time[i];
    end = time[i+1];
  end;
end;
```

Synchronizacja czasu

- **Algorytm Marzullo - przykład**
  - Przykład dla przedziałów

```
[3,10],[2,6],[5,9],[7,12],[6,11]
```



  - Przedział czasu reprezentowany przez dwie pary formatu <*time*, *type*>

```
[3,10]  →  <3,+1>, <10,-1>
[2,6]   →  <2,+1>, <6,-1>
[5,9]   →  <5,+1>, <9,-1>
[7,12]  →  <7,+1>, <12,-1>
[6,11]  →  <6,+1>, <11,-1>
```

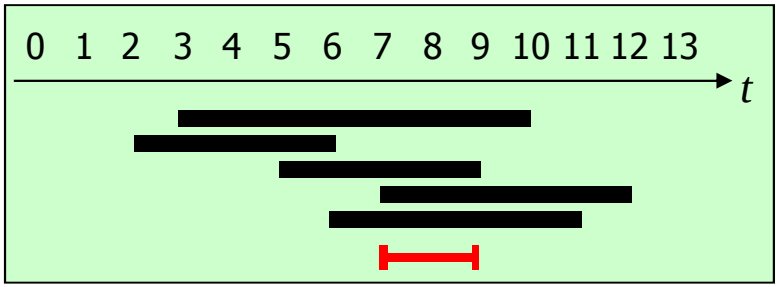
  - Sortowanie par wg wartości *time*

```
{< 2,+1>,
 < 3,+1>,
 < 5,+1>,
 < 6,-1>,
 < 6,+1>,
 < 7,+1>,
 < 9,-1>,
 <10,-1>,
 <11,-1>,
 <12,-1>}
```

Synchronizacja czasu

- **Algorytm Marzullo - przykład**
  - Obliczanie liczby *curr* obejmowanych w danym kroku przedziałów źródeł
    - dla nowego przedziału: *curr* = *curr* + *type*[*i*]
  - Jeśli *curr* > *best* zapamiętanie przedziału wynikowego
    - *begin* = *time* pary w danym kroku
    - *end* = następna wartość *time* kolejnej pary
  - C(t) = *f* (*begin* , *end*)
    - Np. C(*t*) = (9-7)/2 = 8

krok	<*time*,*type*>	*curr*	*best*	[*begin*,*end*]
		0	0	
1	<2,+1>	1	0→1	[2,3]
2	<3,+1>	2	1→2	[3,5]
3	<5,+1>	3	2→3	[5,6]
4	<6,-1>	2	3	[5,6]
5	<6,+1>	3	3	[5,6]
7	<7,+1>	4	3→4	[7,9]
8	<9,-1>	3	4	[7,9]
9	<10,-1>	2	4	[7,9]
10	<12,-1>	1	4	[7,9]
11	<13,-1>	0	4	**[7,9]**



Synchronizacja czasu

- **Protokół NTP** (ang. *Network Time Protocol*)
  - Standard synchronizowania zegarów w sieci Internet
    - Usługa umożliwiająca w klientom synchronizację z czasem UTC
    - Wiele serwerów i ścieżek połączeń
      - Zdolność do przetrwania długich braków połączeń i dużych, i zmiennych opóźnień
    - Duża skalowalność systemu
  - Struktura
    - **Serwery podstawowe (*primary*)** - bezpośrednio połącz. ze źródłami UTC
    - **Serwery pomocnicze (*secondary*)** - hierarchiczna sieć synchronizacji
  - Tryby synchronizacji
    - Tryb rozsyłania – nadawania wartości czasu do serwerów podległych
      - W szybkich sieciach lokalnych (nieduża dokładność)
    - Tryb wywołania procedur – podobny do algorytmu Cristiana
    - Tryb symetryczny – dla serwerów głównych i w wyższych warstwach systemu
      - Wymiana komunikatów i polepszanie wskazań m.in. z wykorzystaniem zmodyfikowanego algorytmu Marzullo
      - Największa dokładność ustawiania czasu
  - Całkowita dokładność: 1÷50 ms

Synchronizacja czasu

- **Czas wirtualny**
  - Inne podejście:
    uwzględnienie zdarzeń zachodzących w systemie i określenie tylko relacji kolejnościowej zdarzeń
  - Założenie: autonomiczny upływ czasu w procesach
  - Cechy
    - Zależny od występujących zdarzeń
    - Wyrażany za pomocą zegarów logicznych przypisujących określonym zdarzeniom określone wartości
    - Niektóre wartości czasu mogą nie występować
- **Wymagania dla zegarów logicznych**
  - Zapewnienie spójności wewnętrznej zegarów
  - Zapewnienie ścisłego porządkowania zdarzeń

Model procesu rozproszonego

- **Proces rozproszony**
  - Proces rozproszony Π to wykonanie zbioru $P=\{P_1, P_2, \ldots, P_N\}$ procesów sekwencyjnych opisany czwórką

$$\Pi=<\Sigma, \Sigma^0, \mathrm{E}, \Phi>$$

    - $\Sigma \subseteq S_1 \times S_2 \times \ldots \times S_N$ – zbiór stanów globalnych procesu rozproszonego
    - Stan procesu $S_i(t)$ to w ogólności **zbiór wartości zmiennych lokalnych** skojarzonych z procesem w chwili t oraz **ciągów wiadomości** wysłanych i odebranych do/z kanałów komunikacyjnych w chwili t
    - $\mathrm{E}= \mathrm{E}_1 \cup \mathrm{E}_2 \cup \ldots \cup \mathrm{E}_N$ – zbiór zdarzeń
    - Zdarzenie E_i^k interpretuje się jako wykonanie operacji atomowej zmieniającej stan S_i^{k-1} na stan S_i^k procesu i
    - Φ - funkcja tranzycji, taka że $\Phi \subseteq \Sigma \times \mathrm{E} \times \Sigma$
      - Funkcja tranzycji opisuje możliwe zmiany stanu S_i^{k-1} na S_i^k na skutek zajścia zdarzenia E_i^k
  - Klasy zdarzeń
    - *Send*
    - *Receive*
    - *Internal*

Model procesu rozproszonego

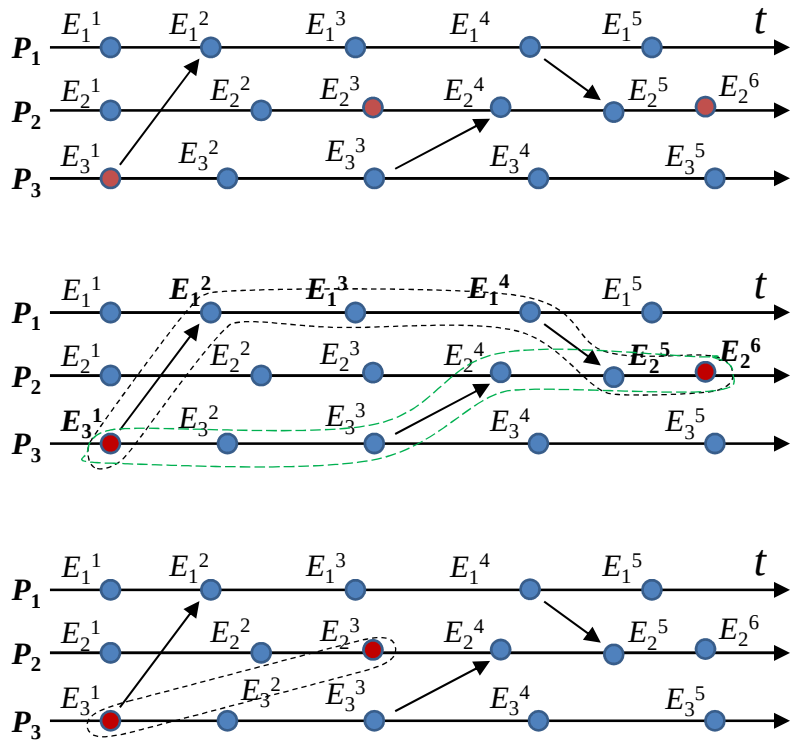
- **Relacja uprzedniości zdarzeń** (ang. *happened before relation*)
  - $E_A \mapsto E_B$ wtedy i tylko wtedy, gdy:
    1. Zdarzenia A i B występują na tym samym procesorze i zdarzenie A wystąpiło przed zdarzeniem B
    2. Zdarzenie A jest zdarzeniem wysłania wiadomości a zdarzenie B jest zdarzeniem odbioru wiadomości
    3. Istnieje sekwencja zdarzeń E_A , E_1 , ... , E_n , E_B taka, że dla każdej pary kolejnych zdarzeń zachodzi 1. lub 2.

$$E_i^k \mapsto E_j^l \Leftrightarrow \begin{cases} 1.\ i = j, k < l \\ 2.\ i \neq j,\ \text{zdarzenie } E_i^k \text{ jest typu } send(P_i, P_j), \\ \quad \text{a zdarzenie } E_j^l \text{ jest typu } receive(P_i, P_j) \\ 3.\ \text{istnieje sekwencja } \left\langle E^0, E^1, \ldots, E^S \right\rangle \text{ taka, że } E^0 = E_i^k \text{ i } E^0 = E_j^l \\ \quad \text{oraz dla wszystkich par } \left\langle E^u, E^{u+1} \right\rangle \text{ zachodzi 1 lub 2} \end{cases}$$

Model procesu rozproszonego

- **Zdarzenie przyczynowo zależne**
  - Takie dla których:
    $E_A \mapsto E_B$ albo $E_B \mapsto E_A$
- $\boldsymbol{E_3^1,\ E_2^3,\ E_2^6}$ **?**
  - Tu: $E_3^1 \mapsto E_2^6$
- **Zdarzenia współbieżne**
  - Jeśli między dwoma zdarzeniami nie istnieje relacja uprzedniości to te zdarzenia określa się jako równoczesne lub współbieżne
  - Tu: $E_3^1 \parallel E_2^3$

Diagramy przestrzenno-czasowe



Zegary logiczne

- **Zegar logiczny**

  jest funkcją T odwzorowującą zbiór zdarzeń E w uporządkowany zbiór Y, T: $\mathrm{E} \rightarrow Y$ taką, że:

$$\boxed{(E_A \mapsto E_B) \Rightarrow (T(E_A) < (T(E_B))}$$

gdzie: $<$ – to relacja porządku na zbiorze Y,

$\mapsto$ – to relacja uprzedniości zdarzeń.

$\boldsymbol{Y}$ – przeciwdziedzina – uporządkowany zbiór

- **Konstrukcje zegarów logicznych**
  - Polega na opracowaniu funkcji $T(E)$ spełniającej definicję, przy czym:
    - Czas zegara musi rosnąć
    - Poprawki można wprowadzać tylko przez dodawanie
  - Rodzaje
    - Skalarne
    - Wektorowe

Zegary logiczne

- **Zegar skalarny**
  - Etykietowanie każdego zdarzenia globalnym znacznikiem czasowym $C(E)$ – wartością skalarną taką że:
    - $C(E_A) < C(E_B)$ dla $E_A \mapsto E_B$ w ramach jednego procesu
  - W systemie rozproszonym: zestaw zgodnych zegarów logicznych utrzymywanych przez każdy proces
    - Lokalne liczniki C_i (zmienne, np. liczb naturalnych) reprezentujące wartość funkcji $T(E)$ dla ostatniego zdarzenia w procesie P_i
    - Każde kolejne zdarzenie w procesie P_i zwiększa stan licznika
    - Utrzymanie uprzedniości zdarzeń:

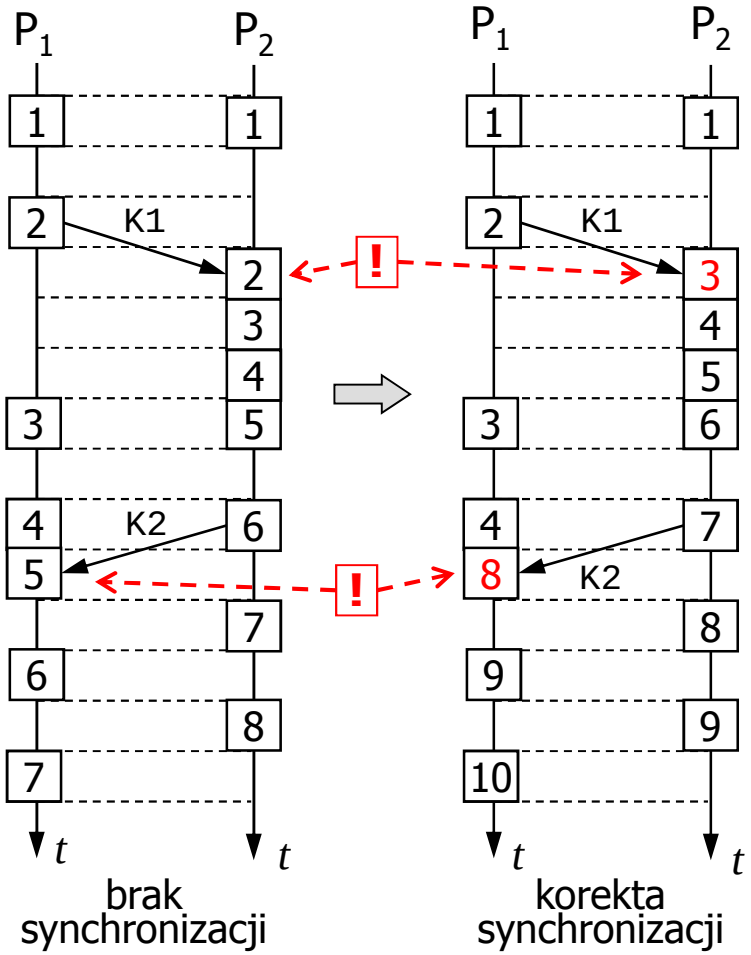
$$\boldsymbol{C(E_A) < C(E_B)}$$

      jeśli E_A jest operacją wysłania komunikatu, a E_B operacją odbioru tego komunikatu
  - Przykład rozwiązania: **zegar Lamporta**

Zegary logiczne

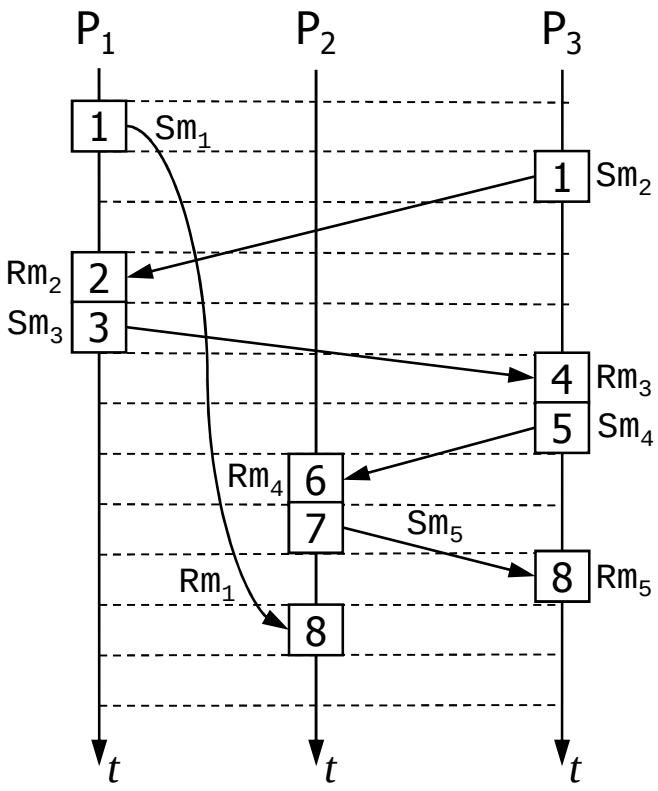
- **Zegar Lamporta**
  1. Każde zdarzenie w procesie P_i zwiększa licznik C_i o 1
  2. Procesy wymieniają komunikaty znakując je własnym czasem logicznym (znacznikiem czasowym)
  3. Gdy proces P_i z czasem C_i otrzyma komunikat z P_j z czasem C_j aktualizuje swój licznik wg zasady:

$$\boxed{C_i = \max[C_i, C_j] + 1}$$



Zegary logiczne

- **Problem**
  - Nie zachowuje relacji przyczynowości
- **Przykład**
  - Jest mechanizm szyfrowanego wyróżniania procesów
  - m_1 – wysłanie klucza (zaszyfrowanego wyróżnika procesu)
  - m_2, m_4 – zapytanie o wyróżnienie
  - m_3, m_5 – odpowiedź
  - Naruszenie przyczynowości np. gdy:
    - $Sm_1 < Sm_4$
      oraz
    - $Rm_1 > Rm_4$
  - Efekt: błąd – żaden proces nie jest wyróżniony



Zegary logiczne

- **Zegar Lamporta**
  - Zegar Lamporta gwarantuje:

$$(E_A \mapsto E_B) \Rightarrow (C(E_A) < C(E_B))$$

  - Zegar Lamporta nie gwarantuje:

$$(C(E_A) < C(E_B)) \Rightarrow (E_A \mapsto E_B)$$

    - Ponieważ zdarzenia równoległe są porządkowane arbitralnie

  - Często potrzebne jest zachowanie relacji przyczynowości

$\Downarrow$

  - Rozwiązanie: **zegar wektorowy**

Zegary logiczne

- **Zegary wektorowe**
  - Zegar wektorowy dla zbioru n procesów rozproszonych
    - Zegar logiczny którego przeciwdziedziną funkcji T jest n-elementowy wektor zmiennych liczb wskazujących porządek (np. naturalnych)
    - Zbiór n liczników wektorowych $C_i=V_i=[v_1,v_2,\ldots v_n]$, $i=1,\ldots,n$
      - $V_i=[i]$ – licznik odpowiadający ostatniemu zdarzeniu w procesie P_i
      - $V_i=[j]$, $j\neq i$ – wyobrażenie procesu P_i o czasie skalarnym procesu P_j
    - Do komunikatów doczepiane są wartości liczników wektorowych
      - Zawierają informacje ile zdarzeń poprzedzało operację wysłania komunikatu przez P_i
    - *Istnieje procedura aktualizacji własnego zegara dla zachowania uporządkowania przyczynowego*
  - Dla zegarów wektorowych:

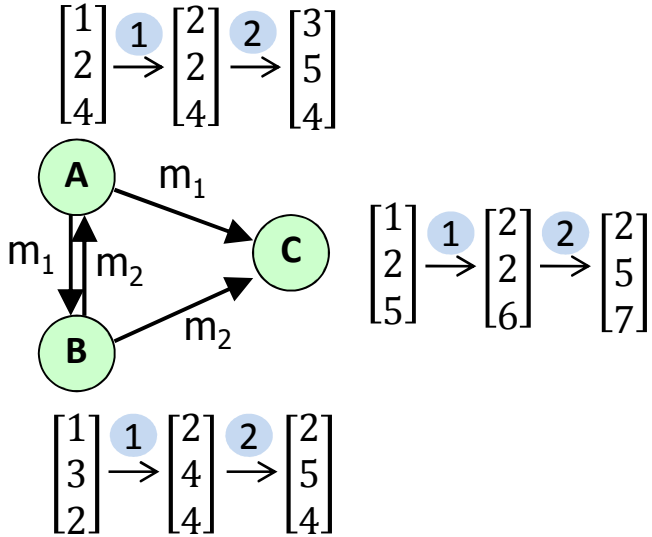
$$E_A \mapsto E_B \Leftrightarrow (T(E_A) < T(E_B))$$

Zegary logiczne

- **Procedura obsługi zegara wektorowego**
  - Wysłanie komunikatu m_l przez proces P_i
    - $V_i[i] = V_i[i] + 1$
    - Wysłanie komunikatu do P_j z doczepionym $V(m_l)$
  - Odbiór komunikatu przez P_j
    - $V_j[j] = V_j[j] + 1$
      - Aktualizacja czasu procesu odbierającego komunikat
    - Dla każdego $k \neq j$:
      $V_j[k] = \max(V_j[k], V(m_l)[k])$
      - Aktualizacja czasów innych procesów wg wiedzy nadawcy i odbiorcy
  - W każdej chwili dla każdego $i \neq j$:

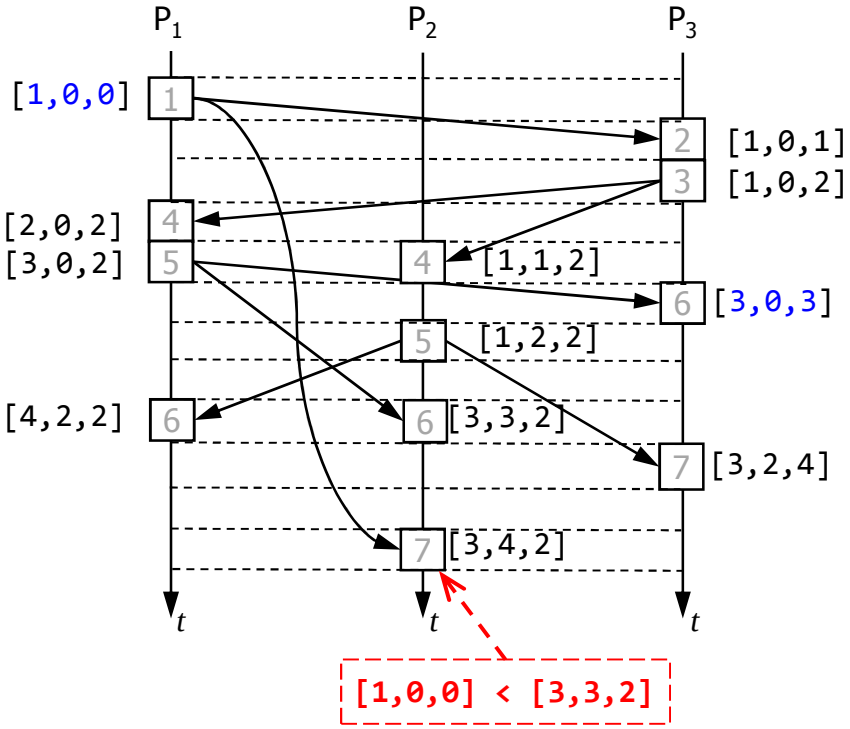
$$V_i=[i] \geq V_j=[i]$$

    - Żaden proces nie widzi większego czasu w innym procesie, niż jest w rzeczywistości



Zegary logiczne

- **Przykład**
  - Tu: nieco inaczej – każdy komunikat to tzw. **rozsyłanie** (nie zmienia to istoty zagadnienia)
  - Nie unikamy naruszenia przyczynowości
  - Ale możemy to wykryć
    - Porównując wskazania zegarów
  - Można podjąć procedury unikania błędów



- **Istnieje procedura wsparcia uporządkowania przyczynowego komunikatów z użyciem zegara wektorowego** – omówione będzie potem

Wzajemne wykluczanie

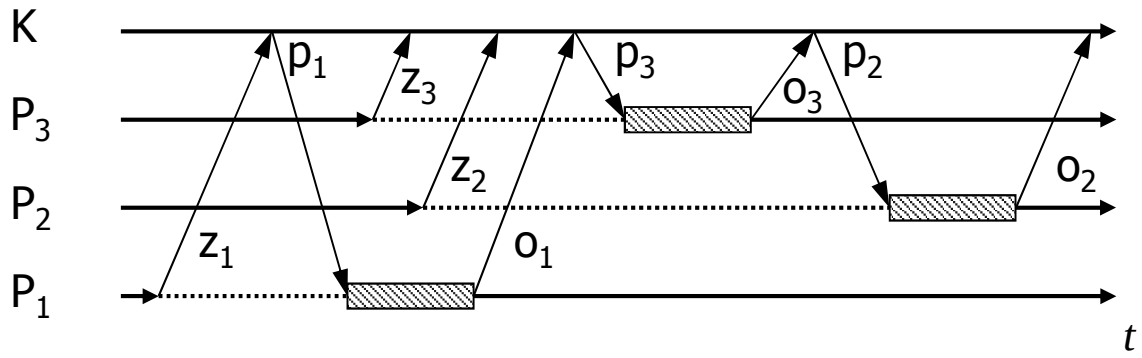
- **Problem**
  - Współbieżny dostęp do zasobów → problem zachowania spójności danych
  - Wzajemne wykluczenie zapewnia procesowi ochronę przy dostępie do zasobu → jako jedyny będzie mógł z niego korzystać
  - Systemy scentralizowane
    - Blokowanie zasobu: sekcja krytyczna (SK), semafory, monitory, …
  - Przetwarzanie rozproszone
    - Brak możliwości wykorzystania zmiennych dzielonych
    - Brak możliwości wykorzystania mechanizmów pojedynczego jądra systemu operacyjnego

Wzajemne wykluczanie

- **Wymagania**
  - Warunek bezpieczeństwa (WB)
    - W danej chwili w SK może działać tylko jeden proces
  - Warunek ruchu (WR)
    - Proces zamawiający SK musi otrzymać w końcu taki dostęp
  - Warunek uporządkowania (WU)
    - Prawo do wejścia w SK musi być udzielane z uwzględnieniem uprzedniości zdarzeń
- **Typy algorytmów:**
  - Podejście scentralizowane
    - Jeden wybrany proces (serwer) jest koordynatorem wejścia do SK
  - Algorytmy rozproszone
    - Procesy wspólnie koordynują dostęp
  - Algorytmy bazujące na żetonie

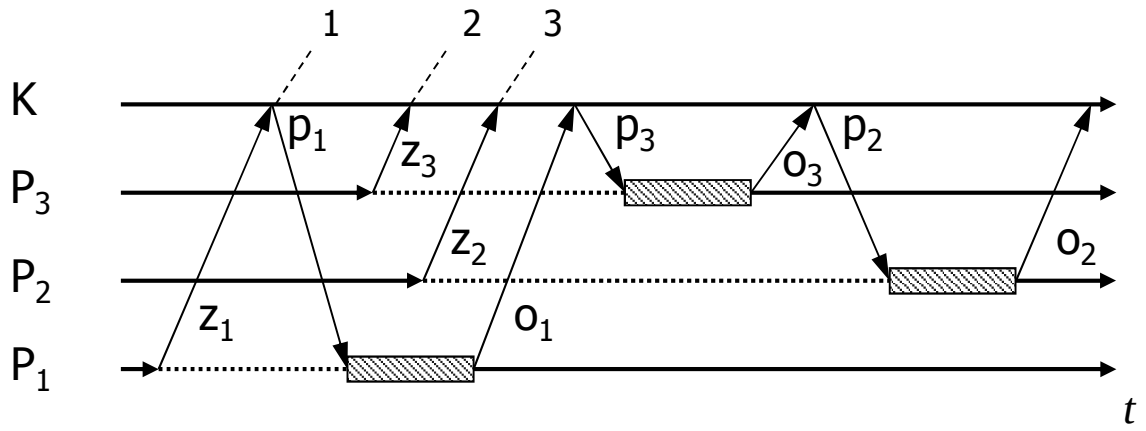
Wzajemne wykluczanie

- **Metoda zcentralizowana**
  - Jeden wybrany proces (serwer) jest koordynatorem wejścia do SK
  - Proces P_i , który chce wejść do SK wysyła wiadomość zamówienia z_i do koordynatora
  - Jeżeli SK jest wolna to koordynator odsyła pozwolenie p_i do P_i
  - Po otrzymaniu pozwolenia P_i wchodzi do SK
  - Po zwolnieniu SK proces P_i wysyła wiadomość zwolnienia o_i



Wzajemne wykluczanie

- **Metoda zcentralizowana**
  - Serwer składuje i rozpatruje zamówienia wg kolejności zgłoszeń
  - Procesy zamawiające wejście do zajętej SK muszą czekać na dostęp
    - Wykrywanie: zwłoka odpowiedzi lub odbiór wiadomości odmownej
  - Zalety i wady
    - Algorytm sprawiedliwy
    - Wrażliwość na awarie (serwera i klienta)
      - › Awaria koordynatora – nieodróżnialność braku odpowiedzi od awarii

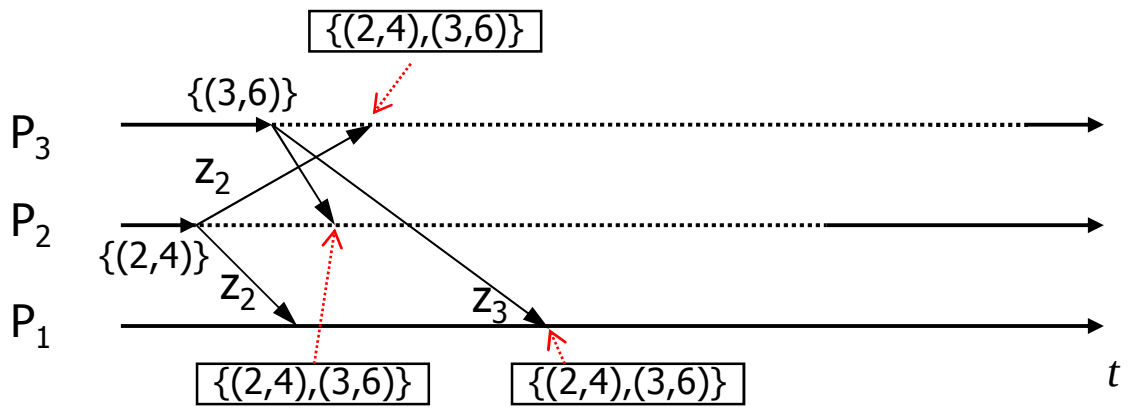


Wzajemne wykluczanie

- **Algorytmy rozproszone**
  - Jest zbiór R procesów od których wymagane jest pozwolenie wejścia do SK realizujących mechanizm synchronizacji
  - Wymaganie całkowitego uporządkowania zdarzeń
    - Procesy utrzymują zegary logiczne
    - Zegary procesów są synchronizowane algorytmem zegarowym
- **Algorytm Lamport'a**
  - Procesy utrzymują kolejki żądań wejścia do SK
    - Kolejki są porządkowane wg kolejności (czasu) zamówień
  - Zegar Lamporta
  - Komunikaty
    - Zamówienie zasobu **z**, odpowiedź **o** i uwolnienie zasobu **u**
    - Zawierają
      - ID zasobu
      - ID procesu
      - Wartość zegara – znacznik czasowy **C**

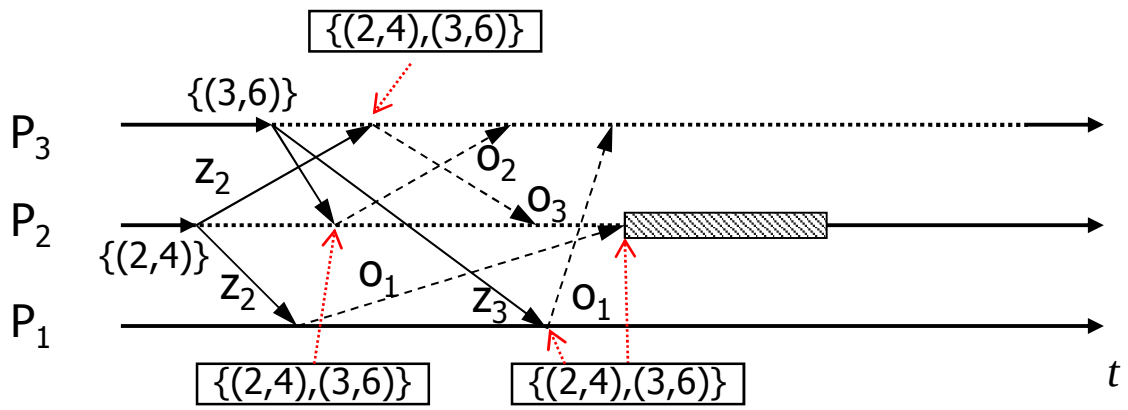
Wzajemne wykluczanie

- **Algorytm Lamport'a**
  - Procedura
    - Żądanie SK w procesie P_i
      - Wysłanie żądania wejścia do SK ze znacznikiem czasu $z_i(C_i)$ do wszystkich procesów zbioru R
      - Dodanie żądania do własnej kolejki żądań wejścia do SK
    - Reakcja procesów P_j
      - Umieszczenie żądania P_i w kolejce żądań wejścia do SK



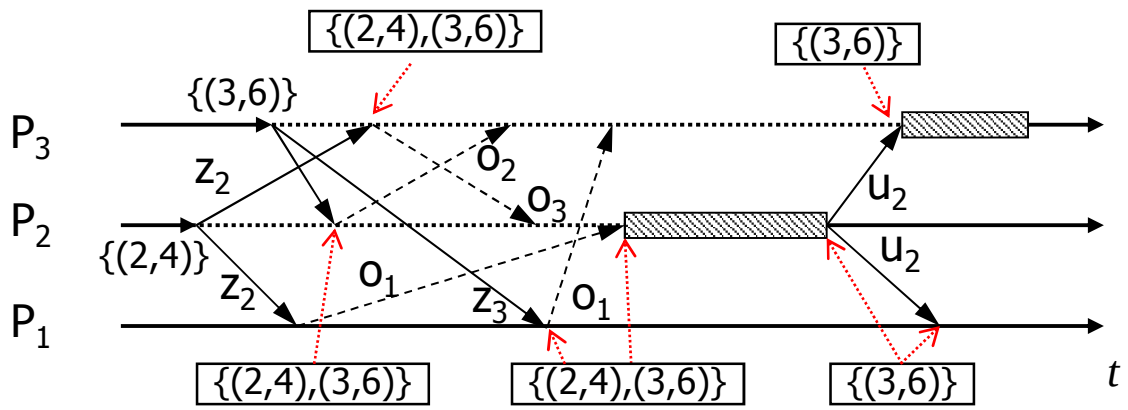
Wzajemne wykluczanie

- **Algorytm Lamport'a**
  - Reakcja procesów P_j (c.d.)
    - Wysłanie odpowiedzi $o_j(C_i)$ na żądanie wejścia do SK do procesu zamawiającego
- Warunek wejścia do SK
  - Otrzymanie odpowiedzi od wszystkich procesów ze znacznikami czasowymi C_j większymi od znacznika własnego żądania C_i
  - Żądanie procesu P_i jest na początku własnej kolejki żądań SK



Wzajemne wykluczanie

- **Algorytm Lamport'a**
  - Zwalnianie SK przez P_i
    - Wysłanie wiadomości u_i opuszczenia SK
    - Usunięcie własnego żądania z początku kolejki żądań SK
    - Po otrzymaniu wiadomości u_i usunięcie żądania P_i z kolejek przez inne procesy

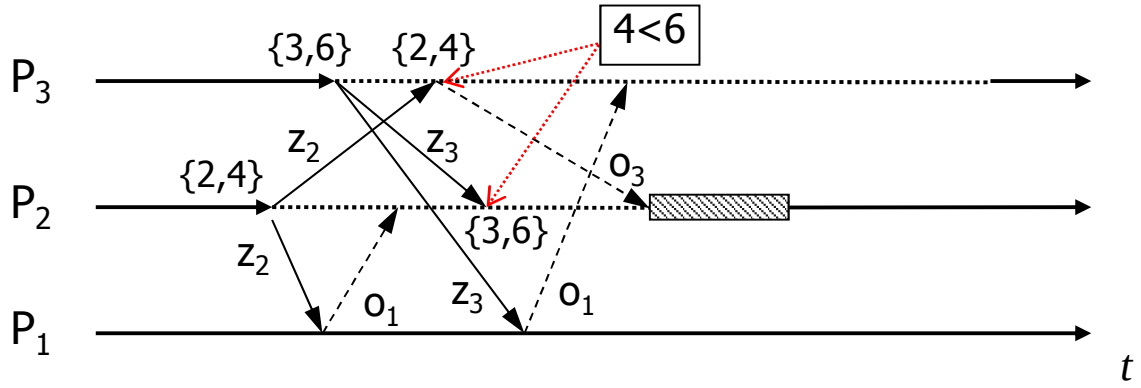


Wzajemne wykluczanie

- **Algorytm Lamport'a**
  - Charakterystyka
    - Wrażliwość na awarie członków grupy
      - Oczekiwanie na odpowiedzi o_i
      - Kontrola czasu przeterminowania otrzymania odpowiedzi o_i
    - Blokowanie do czasu otrzymania u_i
      - Potrzeba niezawodnej komunikacji grupowej
    - Dość wolny i kosztowny
    - Lepiej działa w małych grupach

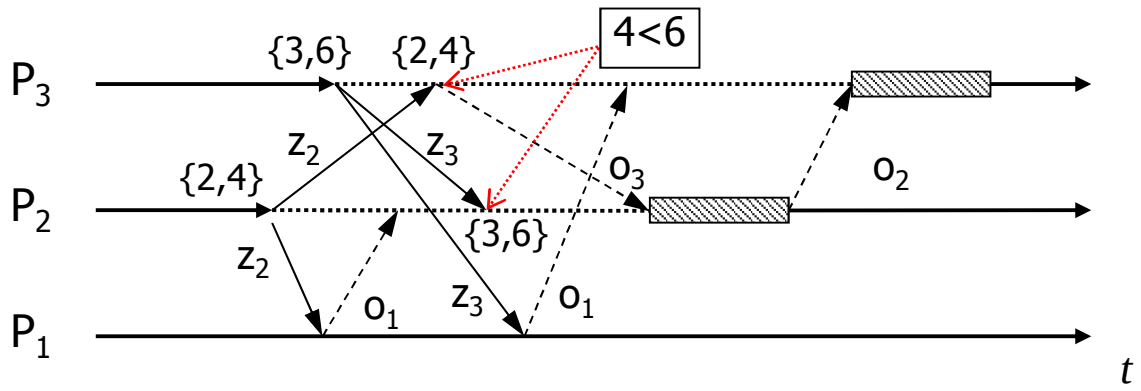
Wzajemne wykluczanie

- **Algorytm Ricarta-Agrawali**
  - Żądanie SK w procesie P_i
    - Wysłanie żądania SK ze znacznikiem czasu $z_i(C_i)$ do wszystkich procesów zbioru R_i
  - Reakcja procesów P_j
    - Jeśli P_j nie w SK i nie chce wchodzić do SK to wysłanie odpowiedzi $o_j(C_j)$ do procesu zamawiającego
    - Jeśli P_j chce wejść do SK ale jego C_j jest większy od C_i żądania z_i to wysłanie odpowiedzi $o_j(C_j)$ do procesu zamawiającego



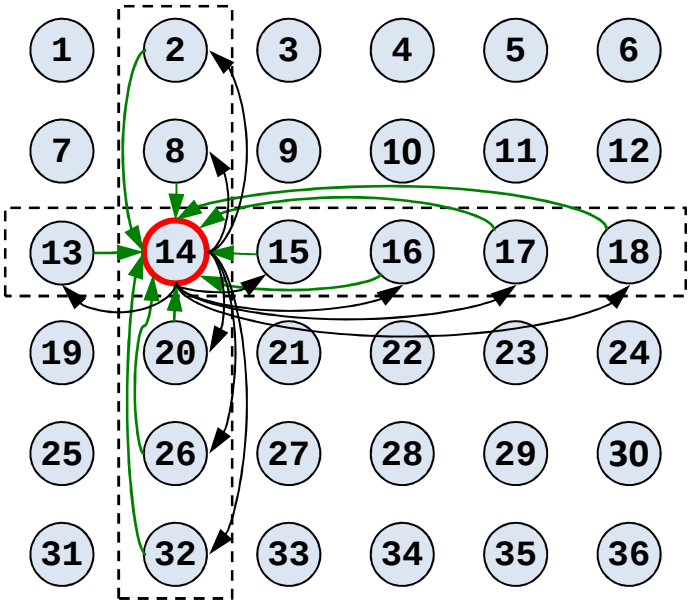
Wzajemne wykluczanie

- **Algorytm Ricarta-Agrawali**
  - Reakcja procesów P_j (c.d.)
    - Jeśli C_j procesu P_j jest mniejszy (lub P_j jest w SK) to żądanie z_i wstawiane do kolejki, a odpowiedź na nie wysyłana jest po wyjściu z SK
  - Wejście do SK po otrzymaniu odpowiedzi od wszystkich procesów
  - Wady i zalety
    - Brak zakleszczeń i zagłodzeń
    - Awaria procesu ⟶ awaria systemu



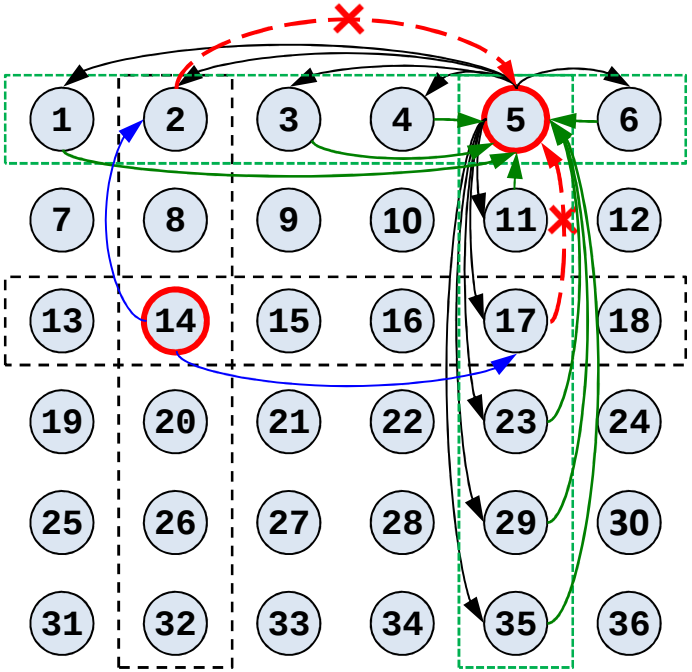
Wzajemne wykluczanie

- **Algorytm Maekawy**
  - Każdy procesor p ma ***voting district*** $S_p \in \{S_1, S_2, \ldots, S_N\}$
  - $\forall i, j\colon\ S_i \cap S_j \neq \emptyset$
  - $|S_i|$=K – równe rozmiary
  - Każdy proces jest w przybliżeniu w tej samej liczbie dystryktów
  - N=K(K-1)+1
  - $\mathrm{K} = O(2 \cdot \sqrt{N})$
- **Procedura**
  - Jeżeli proces chce wejść do sekcji krytycznej, potrzebuje pozwolenia tylko od pewnego podzbioru wszystkich procesów
    - ŻĄDANIE i ZEZWOLENIE (odpowiedź)
  - Proces może naraz wysłać tylko jedną odpowiedź (zezwolenie)
    - Jeśli wcześniej zezwolił innemu procesowi, to tylko po ZWOLNIJ od poprzedniego procesu



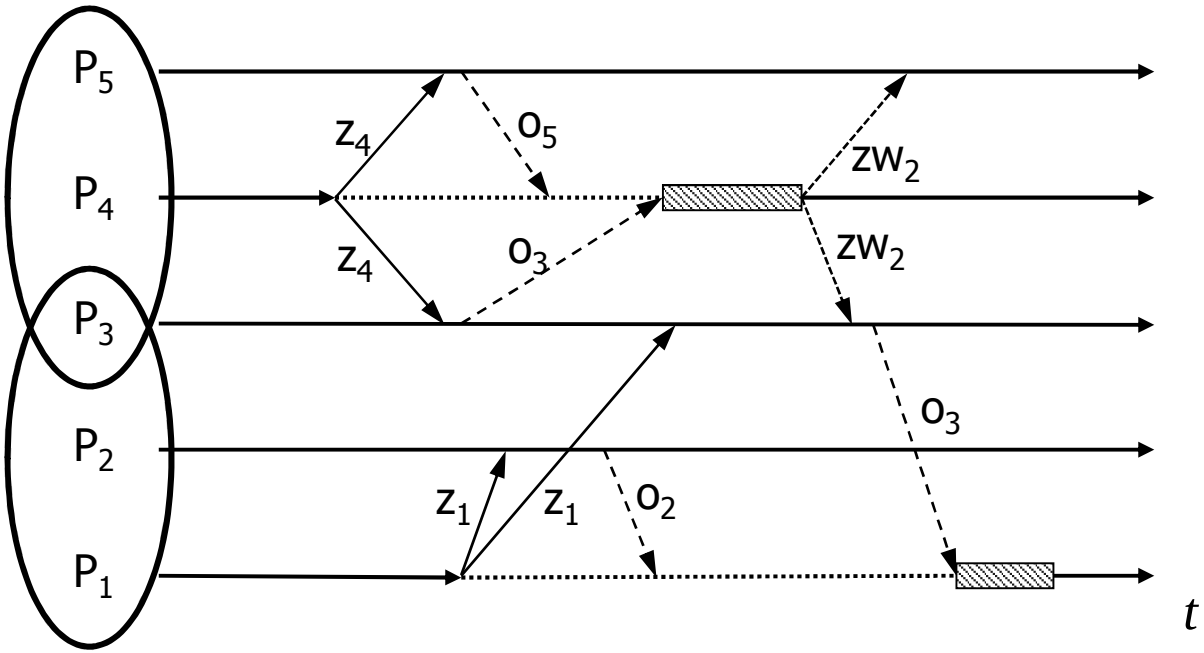
Wzajemne wykluczanie

- **Algorytm Maekawy**
  - Żądanie sekcji krytycznej
    - Wysłanie przez P_i żądania do wszystkich procesów w S_i
    - Odesłanie odpowiedzi lub odłożenie żądania do kolejki
  - Wejście do sekcji po otrzymaniu odpowiedzi od wszystkich procesów ze zbioru S_i
  - Zwalnianie sekcji krytycznej
    - Wysłanie wiadomości ZWOLNIJ do wszystkich procesów z S_i
    - Wysłanie odpowiedzi (ZEZWOLENIE) do kolejnych procesów, które są w kolejce



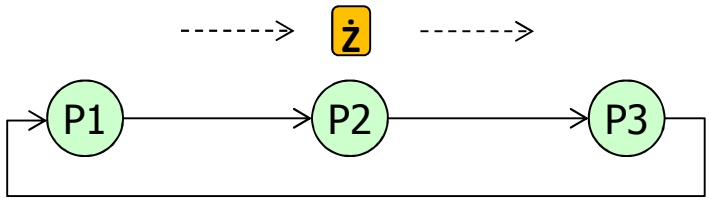
Wzajemne wykluczanie

- **Algorytm Maekawy**
  - Przykład



Wzajemne wykluczanie

- **Algorytm pierścieniowy/karuzelowy**
  - Metoda z żetonem
  - Procesy tworzą logicznie pierścień
  - Z zasobem związany jest żeton
  - Przekazywanie żetonu do sąsiada
  - Posiadanie żetonu pozwala wejść do SK
  - Wady zalety
    - Brak zagłodzeń dla pierścienia jednokierunkowego
    - Nie dotrzymuje własności WU
    - Awaria procesu ⟶ problem odbudowy pierścienia
    - Zgubienie żetonu ⟶ problem elekcji



Zakleszczenia

- **Zakleszczenie**
  - Zbiór procesów (dwa lub więcej) czeka w nieskończoność na zdarzenie, które może być wywołane tylko przez jeden z nich
- **Zakleszczenia w SR**
  - Rozwinięcie metod stosowanych w systemach nierozproszonych
- **Model wykorzystywany do analizy zakleszczeń**
  - W systemie występują różne zasoby **$Z=\{Z_1, Z_2, ..., Z_m\}$**
    - Np. bloki pamięci operacyjnej, obiekty danych, cykle CPU, …
  - Każdy zasób może mieć kilka instancji
  - W systemie istnieje zbiór procesów **$P=\{P_1, P_2, ..., P_n\}$** które mogą chcieć korzystać z zasobów
  - Schemat korzystania procesu z zasobu jest następujący
    - **Zamówienie (żądanie) zasobu** – jeśli nie może być natychmiast spełnione to proces musi czekać do momentu uzyskania go
    - **Użycie zasobu** – proces może korzystać z zasobu
    - **Zwolnienie zasobu**

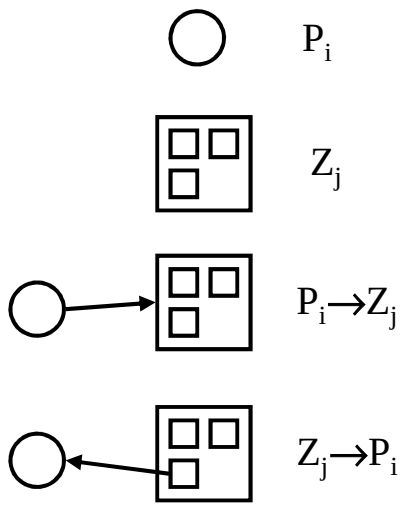
Zakleszczenia

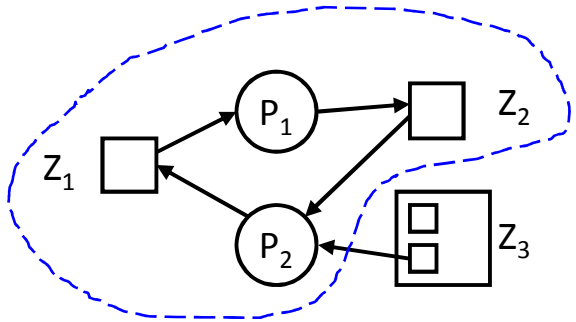
- **Warunki konieczne powstania zakleszczenia**
  - **Wzajemne wykluczanie** – istnieje taki zasób (niepodzielny), że w danej chwili może go używać tylko jeden proces
  - **Przetrzymywanie i oczekiwanie** – istnieje proces, który ma przydzielony co najmniej jeden zasób i oczekuje na przydzielenie kolejnego zasobu, który jest już przydzielony innemu procesowi
  - **Cykliczne oczekiwanie** – istnieje taki zbiór procesów $P=\{P_1, P_2, ..., P_n\}$, że proces P_i czeka na zasób przetrzymywany przez proces P_{i+1}, dla $i \in \{1,...,n-1\}$ i P_n, czeka na zasób przetrzymywany przez proces P_1
  - **Brak wywłaszczania** – zasoby nie podlegają lub nie mogą w określonym czasie podlegać wywłaszczaniu – są zwalniane przez proces przetrzymujący po zakończeniu działania

  - Warunki 2 i 3 są zależne

Zakleszczenia

- **Grafy przydziału zasobów (GPZ)**
  - Grafy skierowane
  - Wierzchołki
    - Elementy zbioru procesów P
    - Elementy zbioru zasobów Z
    - Dany typ zasobu może mieć wiele instancji
  - Krawędzie skierowane
    - Krawędzie żądań zasobów $P_i \rightarrow Z_j$
    - Krawędzie przydziału zasobów $Z_j \rightarrow P_i$
  - Przykład
    - $P=\{P_1, P_2\}$, n=2
    - $Z=\{Z_1, Z_2, Z_3\}$, m=3
    - $K=\{P_1 \rightarrow Z_2, P_2 \rightarrow Z_1, Z_1 \rightarrow P_1, Z_2 \rightarrow P_2, Z_3 \rightarrow P_2\}$
  - GPZ pozwalają wykrywać zakleszczenia
    - Graf bez cykli → brak zakleszczenia
    - Graf z cyklem → możliwość zakleszczenia
      - › Często model zakłada pojedynczość instancji
      - › Jeśli system nierozproszony i pojedyncze instancje to: **cykl = zakleszczenie**





Zakleszczenia

- **Postępowanie z zakleszczeniami**
  - **Zapobieganie zakleszczeniom** – system zabezpieczeń wykluczających wejście w stan zakleszczenia
    - Zapewnienie niespełnienia co najmniej jednego warunku koniecznego powstawania zakleszczenia
      - Na ogół ograniczenia na sposób zamawiania zasobów
      - Zwykle powoduje słabsze wykorzystanie zasobów
  - **Unikanie zakleszczeń** – dodatkowe reguły przydziału zasobów w celu unikania wejścia w stan zakleszczenia
    - Wymóg dysponowania dodatkowymi informacjami o wykorzystaniu zasobów
    - Rozstrzyganie dla każdego zamówienia czy trzeba czekać na zasób
  - **Wykrywanie i usuwanie zakleszczeń** – pozwala się wejść w stan zakleszczenia
    - Mechanizm (algorytm) sprawdzania stanu systemu – wykrywanie i usuwanie zakleszczeń
  - **Ignorowanie zjawiska powstawania zakleszczeń** – założenie że system nigdy (lub bardzo rzadko) nie wejdzie w stan zakleszczenia

Zakleszczenia

- **Unikanie zakleszczeń**
  - Algorytm bankiera
    - Taka kontrola zamówień zasobów, aby zapewnić stan bezpieczny systemu
    - W systemach rozproszonych wyznaczenie w systemie jednego procesu koordynującego dla algorytmu bankiera
- **Zapobieganie zakleszczeniom**
  - Globalne uporządkowanie zasobów w systemie (numeracja)
    - Zamawianie przez proces danego zasobu tylko, gdy zasoby o większym numerze nie są przetrzymywane przez ten proces

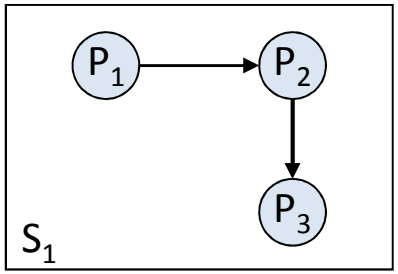
  lub
  - Przydział unikalnych priorytetów dla procesów przy dostępie do zasobów
    - Decydują czy proces P_j czeka na proces P_i
    - Czekać może tylko proces o wyższym priorytecie (niższy usuwany)
    - Brak cykli ale możliwość głodzenia procesów o niskich priorytetach

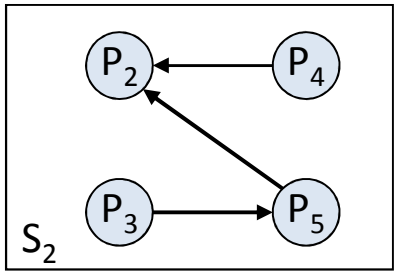
Zakleszczenia

- **Algorytm zapobiegania zakleszczeniom z priorytetami i znacznikami czasowymi (ZC)**
  - Przydział unikalnych priorytetów dla procesów
  - Przyporządkowanie znacznika czasowego uruchamianemu procesowi → pozbycie się problemu zagłodzenia
  - Metody:
    - Czekanie albo śmierć (*wait-die*) - bez wywłaszczeń
      - › Jeżeli proces P_j zamawia zasób przetrzymywany aktualnie przez P_i , to:
        - $ZC_j < ZC_i \Rightarrow P_j$ czeka
        - $ZC_j > ZC_i \Rightarrow P_j$ jest wycofywany
      - › Tendencja do czekania i możliwe częste umieranie
    - Zranienie albo czekanie (*wound-wait*)
      - › Jeżeli proces P_j zamawia zasób przetrzymywany aktualnie przez P_i , to:
        - $ZC_j < ZC_i \Rightarrow P_i$ jest wycofywany (zraniony)
        - $ZC_j > ZC_i \Rightarrow P_j$ czeka
  - ZC stale rosną więc procesy w końcu doczekają się zasobu
  - Wada tych algorytmów:
    - Występowanie niepotrzebnych wywłaszczeń

Zakleszczenia

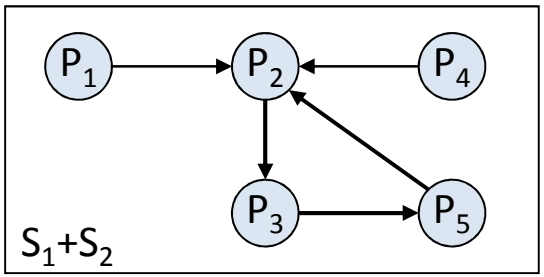
- **Konstrukcja globalnego grafu oczekiwań**
  - Stanowiska (węzły SR) utrzymują ***lokalne grafy oczekiwań***
    - Węzły grafu odpowiadają procesom lokalnym lub zdalnym, które użytkują lub zamawiają zasób
    - Konstruowane standardowo
      - Proces zamawiający zasób wysyła komunikat do stanowiska z zasobem
      - Do lokalnego grafu dodawana odpowiednia krawędź i węzeł procesu który użytkuje zamawiany zasób
  - W pojedynczym węźle:
    - Występowanie cyklu w grafie lokalnym → zakleszczenie
    - Brak cyklu w grafie lokalnym → nie wiadomo czy jest zakleszczenie





Zakleszczenia

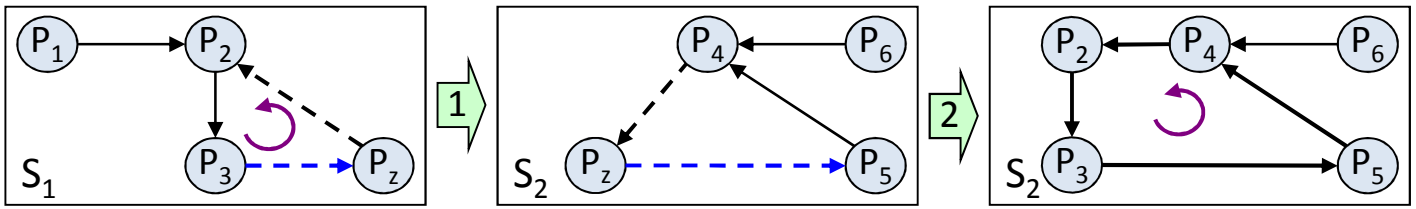
- **Konstrukcja globalnego grafu oczekiwań**
  - Centralny ***koordynator wykrywania zakleszczeń*** – pojedynczy proces
  - Konstruowany ***Graf oczekiwań*** przybliża ***graf rzeczywisty*** na podstawie sumowania grafów lokalnych
    1. Każdorazowo - przy każdej zmianie grafów lokalnych
    2. Okresowo - po określonej liczbie zmian w grafach lokalnych
    3. Każdorazowo - przy uruchamianiu algorytmu wykrywania zakleszczeń
  - Problemy:
    - wykrywanie fałszywych cykli
    - niepotrzebne wycofania przy usuwaniu procesów tworzących cykl bez wiedzy koordynatora



Zakleszczenia

NIEOMAWIANE

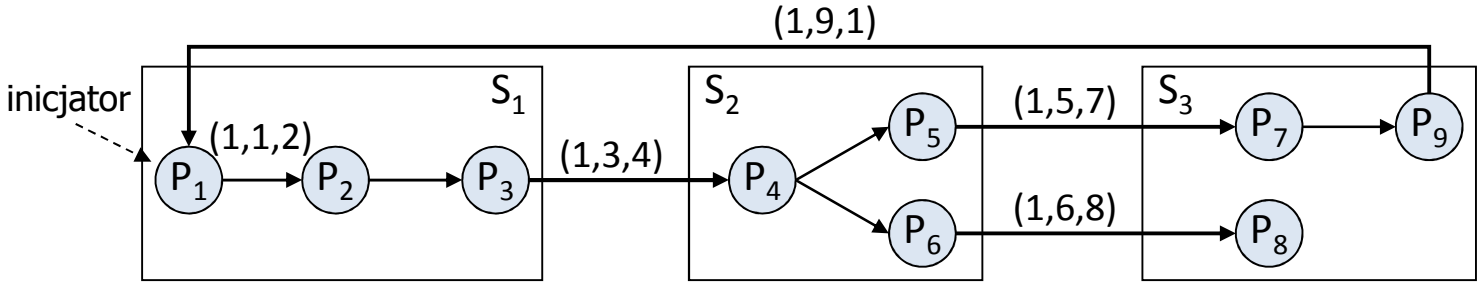
- **Rozproszone wykrywanie zakleszczeń**
  - Konstrukcja grafów lokalnych z dodatkowym węzłem zewnętrznym P_z
    - Łuk między P_i a P_z istnieje jeśli jest oczekiwanie na zasób, który przetrzymuje proces z innego stanowiska (i odwrotnie)
  - Wykrywanie zakleszczenia
    - Gdy jest cykl na S_i z węzłem P_z to wysłanie informacji o tym cyklu do innego stanowiska S_j
    - Proces nadzorczy w S_j uzupełnia graf lokalny i wyszukuje cykle
      - Brak cykli → brak zakleszczeń
      - Jest cykl bez P_z → jest zakleszczenie
      - Jest cykl z P_z → informacja dla kolejnego stanowiska



Zakleszczenia

- **Algorytm Chandy-Misra-Haas'a**
  - Gdy proces ma być zablokowany w oczekiwaniu na zasób używany przez inny proces, wysyła do tego procesu komunikat próbny:

    (NP_rozpoczynającego_czekanie, NP_wysyłającego_komunikat, NP_odbiorcy_komunikatu)
  - Gdy komunikat nadejdzie do innego procesu, ten wysyła zaktualizowany komunikat do procesów na które czeka
    1. Pierwsze pole jest zachowane.
    2. Drugie jest zastąpione własnym numerem procesu
    3. Trzecie numerem procesu na który czeka dany proces.
  - Jeśli komunikat powróci do procesu wysyłającego komunikat kontrolny to w systemie istnieje cykl, a więc i zakleszczenie



Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Koordynacja w systemach rozproszonych

- Transakcje, transakcje rozproszone
- Zatwierdzanie transakcji
- Sterowanie współbieżnością

Transakcje

- **Transakcja**
  - Mechanizm dostępu do wielu zasobów współdzielonych (danych), w postaci jednej niepodzielnej operacji
  - **Problem**: podstawowe operacje w rzeczywistości nie są atomowe
  - **Ochrona danych/obiektów współdzielonych**
    - Wykonywanie powiązanych operacji na różnych obiektach

      P_1: A-x , B+x $\rightarrow$ dla P_n: A+B == const

    - Wykonanie operacji przez różne procesy na tym samym obiekcie

      P_1: A=A+x || P_2: A=A+y $\rightarrow$ A=A+x+y

  - **Postępowanie**: grupowanie operacji i zapewnienie wykonania z odpowiednimi gwarancjami
    - Odpowiednie szeregowanie operacji

Transakcje

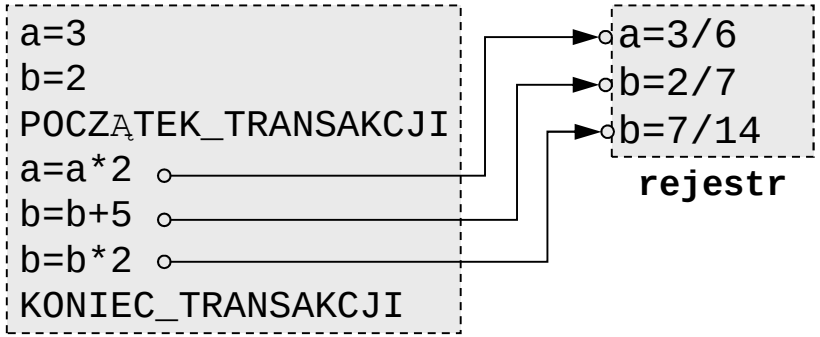
- **Własności ACID**
  - **Atomowość / Niepodzielność** (ang. *Atomicity*)

$$P_1: (O_1, O_2) \rightarrow O_1 \text{ i } O_2 \text{ albo } \emptyset$$

    - Wykonanie wszystkich operacji albo żadnej
    - W razie niepowodzenia możliwość odtworzenia stanu sprzed transakcji
  - **Spójność** (ang. *Consistency*)
    - Transakcja przeprowadza system z jednego stanu spójnego w drugi
    - Zapewnienie na zewnątrz nienaruszenia pewnych niezmienników systemowych (np. warunków, które muszą spełniać dane przed i po transakcji)
    - W czasie transakcji (wewnętrznie) mogą być przejściowo naruszone
  - **Izolacja / Szeregowalność** (ang. *Isolation/Serializability*)
    - Współbieżne transakcje nie oddziałują ze sobą;
    - Realizacja transakcji tak jakby były wykonane wg pewnego porządku sekwencyjnego
  - **Trwałość** (ang. *Durability*)
    - Po zatwierdzeniu efekty transakcji są trwałe i nieodwołalne $\rightarrow$
      $\rightarrow$ odporność na awarie

Implementacja transakcji

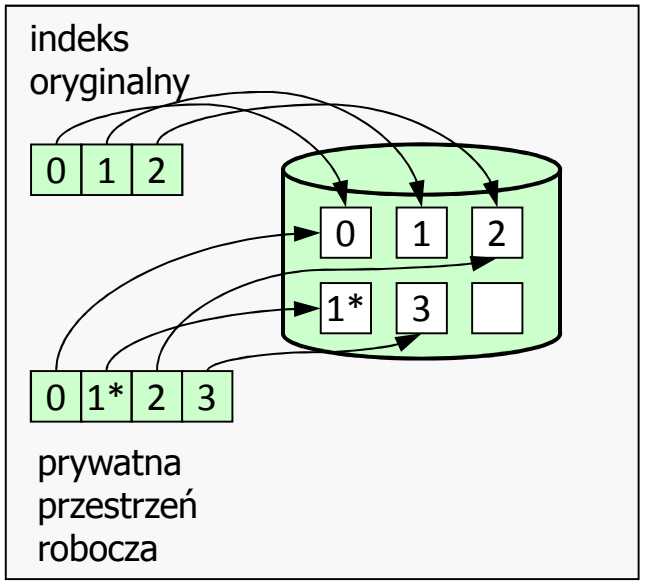
- **Rejestrowanie z wyprzedzeniem**
  - Rejestr transakcji – log operacji wykonywanych przez transakcję
  - Modyfikacja danych w miejscu występowania
  - Przed zmiana bloku zapis do rekordu rejestru starej i nowej wartości

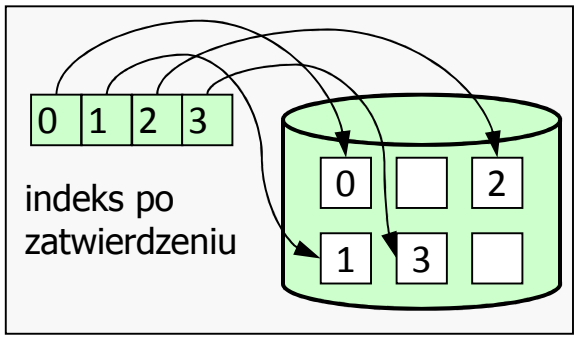


  - Zatwierdzenie transakcji
    - Zapis rekordu zatwierdzenia
  - Wycofanie transakcji
    - Odtwarzanie na podstawie danych z rejestru

Implementacja transakcji

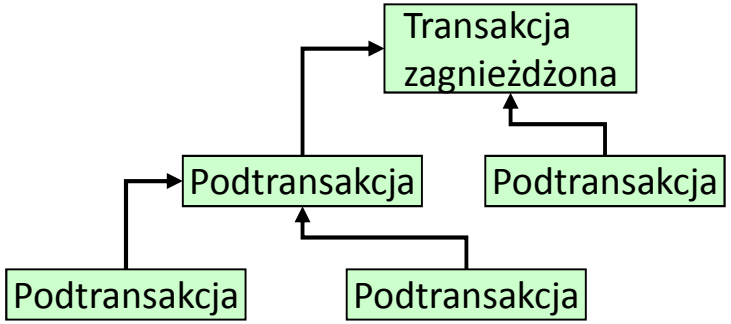
- **Prywatna przestrzeń robocza**
  - Proces transakcji operuje na prywatnej przestrzeni – kopii właściwych danych
  - Optymalizacje
    - Bez kopiowania danych tylko do odczytu
    - **Cienie bloków** (ang. *Shadow blocks*)
      - Kopiowanie odnośników (indeksu)
      - Indeks zawiera wskaźniki do oryginału
      - Przy zapisie tworzone kopie bloków (cienie bloków) i modyfikacja indeksu
  - Zatwierdzenie transakcji
    - Aktualizacja indeksów procesu macierzystego
  - Wycofanie transakcji
    - Usunięcie przestrzeni roboczej – zwrot bloków cieni do wolnych bloków





Typy transakcji

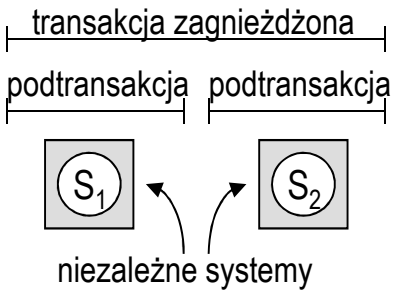
- **Typy transakcji**
  - Transakcje płaskie
    - Prosty ciąg operacji sekwencyjnych spełniające własności ACID
    - Wada: brak zatwierdzania i wycofywania wyników częściowych (pożądanych np. z chęci uzyskania części operacji)
  - Transakcje zagnieżdżone
    - Hierarchia podtransakcji
    - Oddzielne (współbieżne) wykonywanie i zatwierdzanie podtransakcji
    - Lepsza wydajność
    - Niezależne wycofywanie podtransakcji
      - › Niekoniecznie wycofywanie rodzica

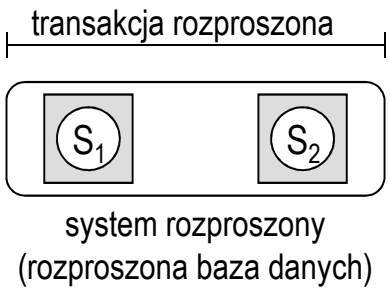


    - **Własność trwałości** w przypadku transakcji zagnieżdżonej rozumiemy jako **trwałość transakcji szczytowej**
      - › Problem trwałości:
        - Zatwierdzenie podtransakcji dostarcza jej wyniki transakcji nadrzędnej
        - Wycofanie transakcji nadrzędnej → wycofanie podtransakcji zakończonych

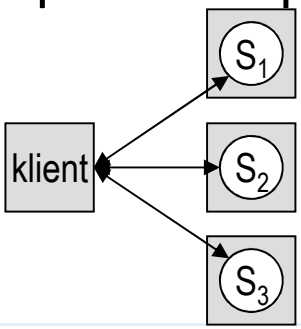
Typy transakcji

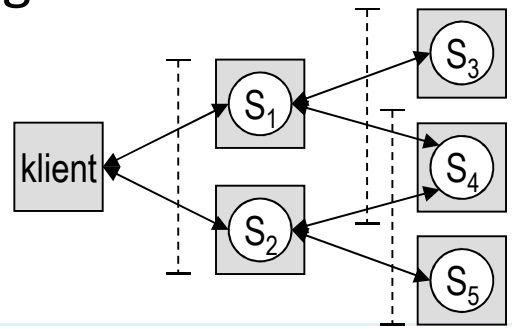
- **Typy transakcji (c.d.)**
  - Transakcje rozproszone
    - Transakcja która realizuje działania na kilku serwerach (w systemie rozproszonym)
    - Rozproszenie a zagnieżdżenie





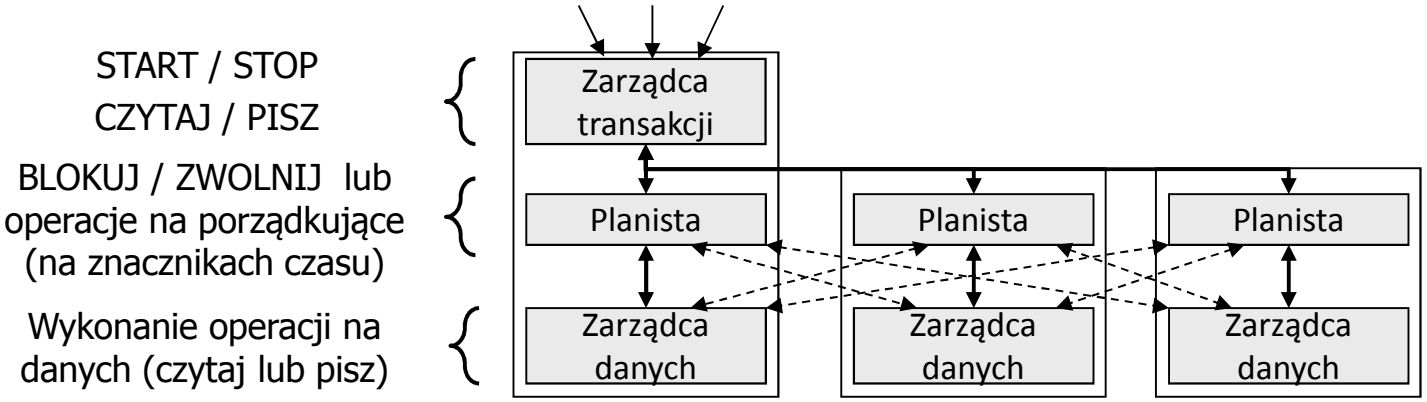
  - Transakcje rozproszone proste i zagnieżdżone





Transakcje

- **Model transakcji**
  - Sterowanie za pomocą elementarnych operacji
    - **POCZĄTEK** – rozpoczęcie transakcji
    - **KONIEC** – zakończenie transakcji i rozpoczęcie operacji zatwierdzenia
    - **CZYTAJ** – czytanie danych
    - **PISZ** – zapis danych
    - **ZANIECHAJ** – wycofanie transakcji (przywrócenie pierwotnego stanu)
  - Zarządzanie transakcjami
    - Koordynator transakcji rozproszonych
    - Mechanizm zatwierdzania transakcji w zaangażowanych serwerach
    - Mechanizmu rekonstrukcji (przywracania systemu po awarii)



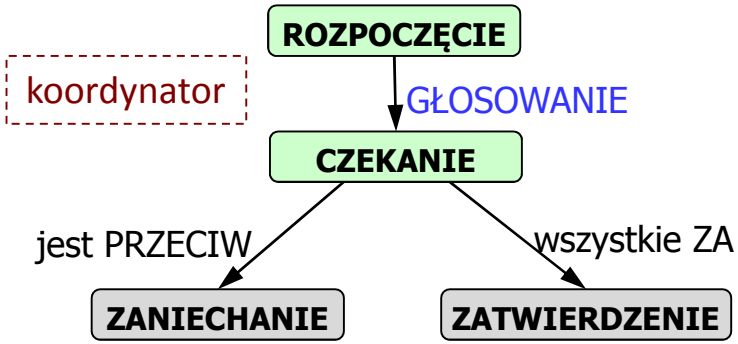
Zarządzanie transakcjami

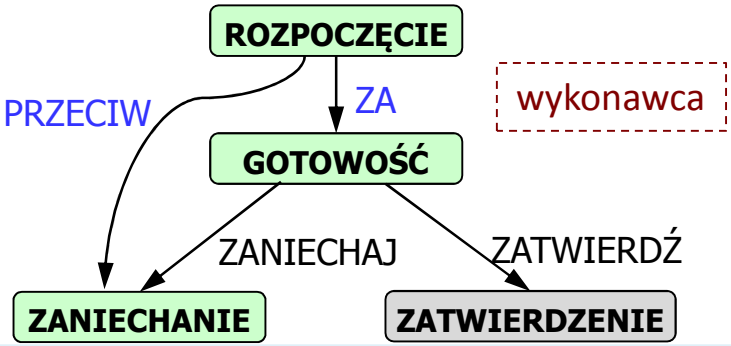
- **Zatwierdzanie transakcji**
  - Protokół jednofazowego zatwierdzania
  - Protokół dwufazowego zatwierdzania - 2PC
  - Protokół trójfazowego zatwierdzania - 3PC

- **Protokół dwufazowego zatwierdzania – 2PC**
  (ang. *Two Phase Commit*)
  - Umożliwia dowolnemu serwerowi zaniechanie transakcji
  - Koordynator prowadzi dialog z serwerami realizującymi transakcję lokalnie

Protokół dwufazowego zatwierdzania (2PC)

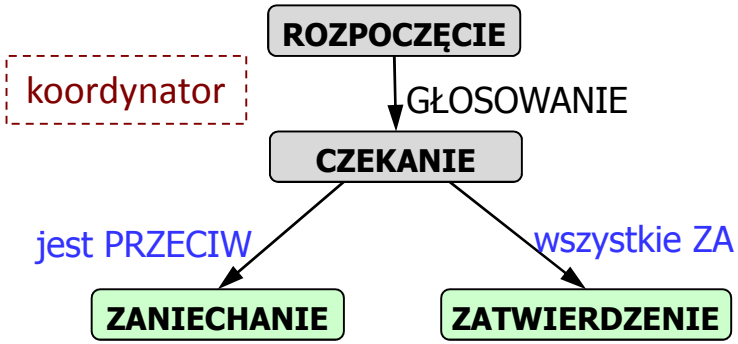
- **Faza I – głosowanie**
  - Wysłanie przez koordynatora komunikatu zlecającego głosowanie
    – komunikat GŁOSOWANIE
    – przechodzi w stan CZEKANIE
  - Wysłanie do koordynatora głosu ZA lub PRZECIW zatwierdzeniu
    - Głos PRZECIW ⇒ zaniechanie lokalne
      – przechodzi w stan ZANIECHANIE
    - Głos ZA ⇒ oczekiwanie na decyzję koordynatora
      – przechodzi w stan GOTOWOŚĆ

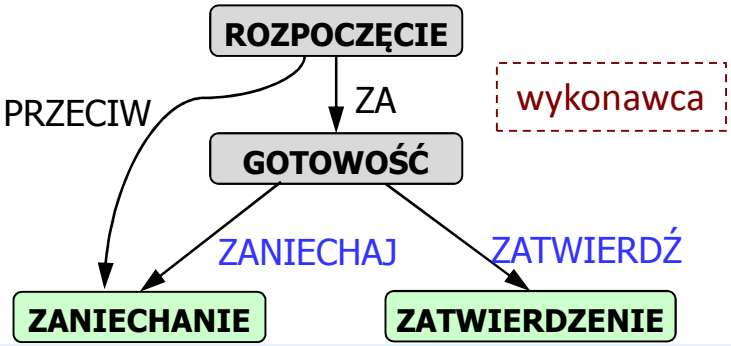




Protokół dwufazowego zatwierdzania (2PC)

- **Faza II – podjęcie decyzji**
  - Koordynator:
    - Wszystkie głosy ZA ⇒ zatwierdzenie globalne do wszystkich serwerów
      – komunikat ZATWIERDŹ
      – przechodzi w stan ZATWIERDZENIE
    - Jest głos PRZECIW ⇒ zaniechanie globalne do wszystkich serwerów
      – komunikat ZANIECHAJ
      – przechodzi w stan ZANIECHANIE
  - Serwery – zaniechanie lub zatwierdzenie lokalne
    - Komunikat ZATWIERDŹ ⇒ zatwierdzenie lokalne (stan ZATWIERDZENIE)
    - Komunikat ZANIECHAJ ⇒ zaniechanie lokalne (stan ZANIECHANIE)





Protokół dwufazowego zatwierdzania (2PC)

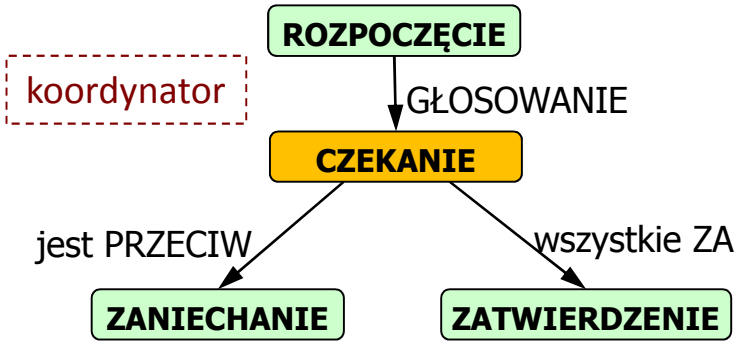
- **Problemy**
  - Procesy blokują się w oczekiwaniu na komunikaty
  - W przypadku awarii trzeba sobie z tym poradzić
  - W przypadku awarii koordynatora niemożność realizacji transakcji
- **Transakcje zagnieżdżone**
  - Tymczasowe zatwierdzanie podtransakcji
  - Transakcja nadrzędna na podstawie wykazu tymczasowo zatwierdzonych i zaniechanych podtransakcji decyduje o ostatecznym zatwierdzeniu lub zaniechaniu i dalszych działaniach

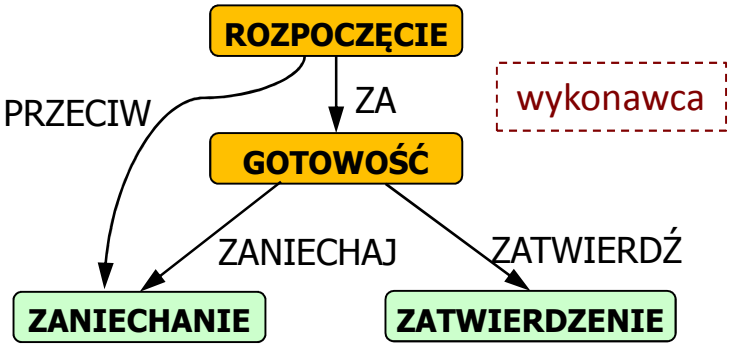
Protokół dwufazowego zatwierdzania (2PC)

- **Protokół dwufazowego zatwierdzania (2PC)**
  - Problemy
    - Procesy blokują się w oczekiwaniu na komunikaty
    - W przypadku awarii trzeba sobie z tym poradzić
    - W przypadku awarii koordynatora niemożność realizacji transakcji
  - Transakcje zagnieżdżone
    - Stan zatwierdzenia tymczasowego dla podtransakcji
    - Transakcja nadrzędna
      - Otrzymuje wykaz tymczasowo zatwierdzonych i zaniechanych podtransakcji
      - Na podstawie wykazu decyduje o ostatecznym zatwierdzeniu lub zaniechaniu i dalszych działaniach
      - Jeśli musi być zaniechana to wszystkie potomne też muszą być zaniechane
    - Protokół dwufazowego zatwierdzania w kolejnych serwerach nadrzędnych
      - W komunikatach informacja o zaniechanych transakcjach
      - Unikanie zatwierdzenia podtransakcji zaniechanej transakcji nadrzędnej

Protokół dwufazowego zatwierdzania (2PC)

- **Protokół dwufazowego zatwierdzania (2PC)**
  - Zatwierdzanie transakcji w sytuacjach awaryjnych
  - Komunikaty i stany:





  - Problem: stany blokad
    - Oczekiwanie na komunikaty
    - Awaria któregoś z uczestników grozi zawieszeniem transakcji

Protokół dwufazowego zatwierdzania (2PC)

- **Protokół dwufazowego zatwierdzania (2PC)**
  - Postępowanie
    - Pomiar czasu w stanie oczekiwania na komunikat
    - Przeterminowanie oczekiwania → uznanie awarii partnera komunikacji
    - Odpowiednie działanie w zależności od sytuacji
  - Awaria serwera
    - Oczekiwanie koordynatora na głos serwera (w stanie CZEKANIE)
    - Brak głosu serwera → zaniechanie globalne
  - Awaria koordynatora (I)
    - Oczekiwanie serwera na zlecenie głosowania (w stanie ROZPOCZĘCIE)
    - Brak komunikatu koordynatora:
      - Zaniechanie lokalne
      - Wysyłka głosu PRZECIW zatwierdzeniu

Protokół dwufazowego zatwierdzania (2PC)

- **Protokół dwufazowego zatwierdzania (2PC)**
  - Awaria koordynatora (II)
    - Oczekiwanie serwera na wynik głosowania (w stanie GOTOWOŚĆ)
    - Odpytanie innego serwera – komunikat PYTANIE_O_DECYZJĘ

Stan innego serwera	Działanie
ZATWIERDZENIE	Zatwierdzenie
ZANIECHANIE	Zaniechanie
ROZPOCZĘCIE	Zaniechanie
GOTOWOŚĆ	Odpytanie innego serwera

    - Jeśli wszystkie serwery w stanie GOTOWOŚĆ to problem stałej blokady

⇓

Protokół 3PC

Sterowanie współbieżnością

- **Problemy realizacji transakcji rozproszonych**
  - Wynikające ze współbieżnego wykonywania operacji czytaj i pisz
    - **Problem utraconej aktualizacji**
      - Problem zapewnienia aby: P_1: A=A+x || P_2: A=A+y $\rightarrow$ A=A+x+y
    - **Problem niespójnych odzyskań**
      - Problem zapewnienia aby: P_1: A-x , B+x $\rightarrow$ dla P_n: A+B == const
  - Problemy związane z wycofywaniem transakcji
    - **Problem zabrudzonego czytania**
      - Czytanie wartości zmienionej przez transakcję wycofaną
    - **Problem przedwczesnych zapisów**
      - Wycofywanie transakcji która zapisała daną zmienianą przez inne transakcje

Sterowanie współbieżnością

- **Problem utraconej aktualizacji**
  - Gdy dwie transakcje aktualizujące odczytają starą wartość zmiennej, a następnie używają jej do obliczenia nowej wartości.
    - Modyfikacja nieaktualnej wartości
  - Przykład
    - Początkowy stan konta A=10
    - 2 operacje dodania wartości
      - T1: A=A+3, T2: A=A+2
    - Niefortunny przeplot operacji czytaj i pisz

Transakcja 1	Transakcja 2	stan konta
x=A.czytaj()		10
	y=A.czytaj()	10
A.pisz(x+3)		13
	A.pisz(y+2)	**12**

Sterowanie współbieżnością

- **Problem niespójnych odzyskań**
  - Może pojawić się wtedy, gdy transakcja odzyskująca działa współbieżnie z aktualizującą
    - Odczyt niespójnych danych
  - Przykład
    - Stan kont: A=20 i B=30
      suma kont: Sum
    - Operacja przelewu
    - Sumowanie kont przed zakończeniem transakcji

Transakcja 1	stan konta	Transakcja 2	wartość
x=A.czytaj()	20		
A.pisz(x-5)	15		
		x=A.czytaj()	15
		y=B.czytaj()	30
		Sum.pisz(x+y)	**45**
x=B.czytaj()	30		
B.pisz(x+5)	35		

Sterowanie współbieżnością

- **Problem zabrudzonego czytania**
  - Wynika z interakcji między operacją czytania jednej transakcji, a wcześniejszą operacją pisania innej transakcji, odnoszących się do tego samego obiektu danych
  - **Transakcja wykonuje zabrudzone czytanie (*dirty read*), gdy dojdzie do wycofania wcześniejszej transakcji zapis.**
    - Wartość początkowa → X=100
    - Zapis na danej X przez transakcję T_1 → X.PISZ(20)
    - Odczyt danej X przez transakcję T_2 → X.CZYTAJ(20)
    - Wycofanie transakcji T_1 → X=100
    - T_2 wykonała „zabrudzone czytanie”
  - Postępowanie
    - Opóźnianie zatwierdzania T_2 do czasu zatwierdzania T_1
    - Wycofywanie transakcji jeśli wycofywana jest trans. ”wcześniejsza”

⇩

  - Problem zaniechań kaskadowych

Sterowanie współbieżnością

- **Problem przedwczesnych zapisów**
  - Wynika z interakcji między operacjami pisania, odnoszącymi się do tego samego obiektu danych
  - **Gdy dojdzie do wycofania wcześniejszej transakcji zapisującej (T_1) po zatwierdzeniu transakcji zapisującej realizowanej później (T_2)**, to przywrócony zostanie stan początkowy sprzed obydwu transakcji
    - Jest zaniechanie transakcji przez odtwarzanie obrazu poprzedniego
    - Niech: X=100 ;
      T1: X.PISZ(10) ;
      T2: X.PISZ(20)
      - Zaniechanie T2 $\rightarrow$ X=10 - **o.k.**
      - zaniechanie T1 $\rightarrow$ X=100 – **źle**
  - Aby zagwarantować poprawne wyniki, operacje pisania muszą zostać opóźnione aż do czasu zatwierdzenia lub zaniechania wcześniejszych transakcji aktualizujących te same obiekty danych

Sterowanie współbieżnością

- **Sterowanie wykonaniem operacji atomowych**
  - Tak aby uzyskać taki ich porządek, żeby uniknąć skutków problemów utraconej aktualizacji i niespójnych odzyskań
- **Wykonywanie transakcji**
  - Plan wykonania – ciąg wykonań operacji transakcji
  - Plan szeregowy – plan wykonania sekwencyjnego realizowanych transakcji
  - Operacje konfliktowe: **czytanie - pisanie**, **pisanie - pisanie**
- **Plan równoważny szeregowo**
  - Taki przeplot operacji konfliktowych, który daje taki sam wynik końcowy jak plan szeregowy
    - Zapobiega skutkom problemów utraconej aktualizacji i niespójnych odzyskań

Sterowanie współbieżnością

- **Równoważność szeregowa**
  - Przykład 1
    - T1:

```
POCZĄTEK;
A=7;
A=A-3;
KONIEC;
```

T2:

```
POCZĄTEK;
A=4;
A=A*2;
STOP;
```

Dozwolone:

A=7; A=A-3; A=4; A=A*2;
A=4; A=7; A=A-3; A=A*2;

Niedozwolone:

A=7; A=4; A=A-3; A=A*2;

  - Przykład 2
    - Plan szeregowy

Transakcja 1	stan konta	Transakcja 2	suma
x=czytaj(A)	20		
A=pisz(x-5)	15		
y=czytaj(B)	30		
B=pisz(y+5)	35		
		x=czytaj(A)	15
		y=czytaj(B)	35
		Sum=pisz(x+y)	50

    - Plan równoważny szeregowo

Transakcja 1	stan konta	Transakcja 2	suma
x=czytaj(A)	20		
		x=czytaj(A)	20
A=pisz(x-5)	15		
y=czytaj(B)	30		
		y=czytaj(B)	30
B=pisz(y+5)	35		
		Sum=pisz(x+y)	50

Sterowanie współbieżnością

- **Algorytmy sterowania współbieżnością**
  - Plan równoważny szeregowo stosowany jako kryterium przy opracowaniu protokołów sterowania współbieżności
  - **Odpowiednie planowanie transakcji ze względu na dostępy innych transakcji (operacje konfliktowe)**
    - Niedopuszczanie do równoczesnej realizacji operacji konfliktowych

    lub
    - Wszystkie kolidujące pary działań dwu transakcji (lub większej liczby) powinny być wykonywane w tym samym porządku
  - **Blokowanie lub jawne porządkowanie operacji dostępu do określonego obiektu danych**
- **Zapobieganie problemom związanym z wycofywaniem transakcji**
  - **Ścisłe wykonywanie transakcji**
    - Opóźnianie operacji zapisu i odczytu na obiekcie danych do czasu zatwierdzenia lub zaniechania transakcji, które zapisywały ten obiekt danych
  - **Wykonywanie operacji na kopiach tymczasowych**

Sterowanie współbieżnością

- **Algorytmy**
  - Blokowanie dwufazowe 2PL
    - Podstawowe
    - Ścisłe
  - Optymistyczne sterowanie współbieżnością
    - Z legalizacją wsteczną
    - Z legalizacją wyprzedzającą
  - Pesymistyczne porządkowanie według znaczników czasu
    - Z użyciem kopii tymczasowych
    - Z użyciem wielu wersji

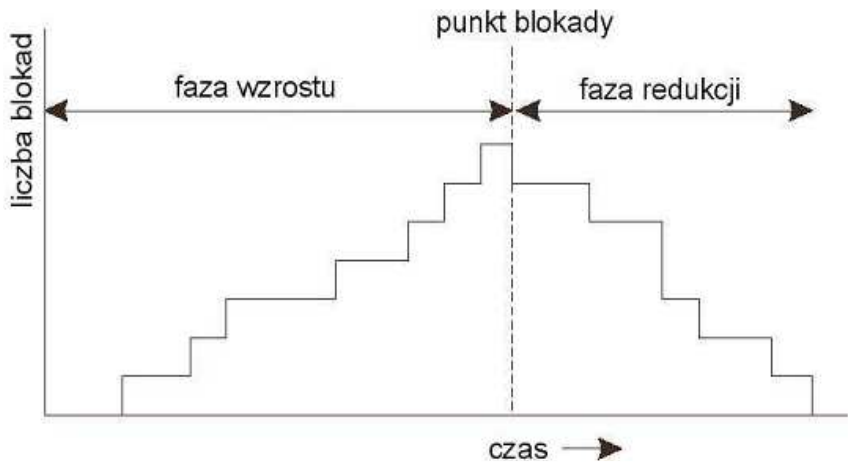
Blokowanie dwufazowe

- **Blokady**
  - Blokowanie obiektu danych w odpowiednim trybie i niedopuszczanie przez planistę do dostępów powodujących konflikty
    - Tryb wspólny / do czytania
    - Tryb wyłączny / do pisania
  - Konfliktowe operacje są wykonywane w kolejności zakładania blokad
  - Reguły konfliktów
    - Po wykonaniu przez transakcję operacji **czytania** inna transakcja **nie może wykonać operacji pisania** do czasu zatwierdzenia lub zaniechania pierwszej
    - Po wykonaniu przez transakcję operacji **zapisu** inna transakcja **nie może wykonać żadnej operacji** do czasu zatwierdzenia lub zaniechania pierwszej

Blokada transakcji 1	Blokada transakcji 2	
	do czytania:	do pisania:
Brak	możliwe	możliwe
Czytanie	możliwe	czekanie
Pisanie	czekanie	czekanie

Blokowanie dwufazowe

- **Faza wzrostu**
  - Planista sprawdza czy istnieje konflikt operacji
    - Nie ma blokady na obiekt danych → zakładana jest blokada
    - Jest blokada ale niekonfliktowa → blokada w trybie wspólnym
    - Jeśli jest konflikt to wstrzymanie do zwolnienia obiektu danych
    - Potrzeba promocji blokady → sprawdzanie konfliktu od nowa
  - Utrzymywanie blokady do czasu wykonania operacji z której powodu nałożono blokadę
- **Faza redukcji**
  - Zdejmowanie blokad następuje dopiero po nałożeniu ostatniej z potrzebnych blokad



- **Gwarancja równoważności szeregowej jeśli wszystkie transakcje stosują blokowanie dwufazowe**

Blokowanie dwufazowe

- **Przykład**
  - Realizacja dla problemu niespójnych odzyskań
    - C(x) - blokada w trybie wspólnym na obiekcie x
    - P(x) - blokada w trybie wyłacznym na obiekcie x

Transakcja 1	Zgł. blok.	Wyk. ?	stan konta	Transakcja 2	Blok.	suma
x=czytaj(A)			20			
				x=czytaj(A)		20
A=pisz(x-5)			15			
x=czytaj(B)			30			
B=pisz(x+5)			35			
				y=czytaj(B)		35
				Sum=pisz(x+y)		55

Transakcja 1	Zgł. blok.	Wyk. ?	stan konta	Transakcja 2	Blok.	suma
x=czytaj(A)	C(A)	o.k.	20			
				x=czytaj(A)	C(A)	20
A=pisz(x-5)	P(A)	**czeka**				
↓	↓	↓		y=czytaj(B)	C(B)	30
				Sum=pisz(x+y)		50
A=pisz(x-5)	P(A)	o.k.	15			
x=czytaj(B)	C(B)	o.k.	30			
B=pisz(x+5)	P(B)	o.k.	35			

Blokowanie dwufazowe

- **Uwagi dotyczące środowiska rozproszonego**
  - Możliwość różnego uporządkowania transakcji przez rozproszone serwery realizujące transakcję
    - Niebezpieczeństwo zakleszczeń (!)
    - Wykrywanie zakleszczeń i wycofywanie transakcji
  - Transakcje zagnieżdżone
    - Nie zezwala się na współbieżne działanie transakcji rodzicielskiej i jej potomków $\rightarrow$ zapobieganie konfliktom między poziomami
    - Dziedziczenie blokad po przodkach
      - Aby uzyskać blokadę do pisania/czytania wszyscy posiadacze blokad do czytania/pisania muszą być przodkami
    - Przy zatwierdzaniu transakcji wszystkie blokady przechodzą we władanie transakcji nadrzędnej

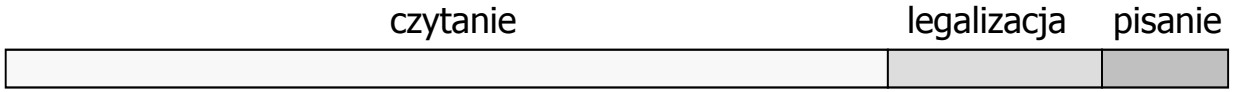
- **Stosuje się różne mechanizmy w schematach blokowania zwiększające współbieżność**

Optymistyczne sterowanie współbieżnością

- **Założenia:**
  - Konflikty są stosunkowo rzadkie
  - Wykonywanie wszystkich potrzebnych operacji na prywatnej kopii
  - Mechanizm cieni bloków
  - Rejestrowanie jednostek danych, które były odczytywane i zapisywane
  - Sprawdzanie konfliktów w fazie zatwierdzania
- **Własności**
  - Brak zakleszczeń
  - Maksymalne zrównoleglenie
  - Duże obciążenie $\longrightarrow$ prawdopodobieństwo przerwania transakcji rośnie
    - W takich systemach rozproszonych rzadko implementowane

Optymistyczne sterowanie współbieżnością

- **3 fazy**
  - Faza czytania
    - Kopia tymczasowa aktualizowanych obiektów danych
      - › Łatwość zaniechań
    - Operacje na dwóch różnych zbiorach danych
      - › Operacje czytania – na *zbiorze czytania* **ZbC**
        - Nie występuje zabrudzone czytanie
      - › Operacje pisania na oddzielnej kopii – *zbiorze pisania* **ZbP**
  - Faza legalizacji
    - Transakcje numerowane w porządku rosnącym przy wchodzeniu w fazę legalizacji
    - Ustalenie czy wykonane operacje nie były konfliktowe z operacjami innych transakcji
      - › Jeśli tak, to *Zaniechanie*
  - Faza pisania
    - Utrwalanie kopii tymczasowych
    - Zatwierdzenie transakcji



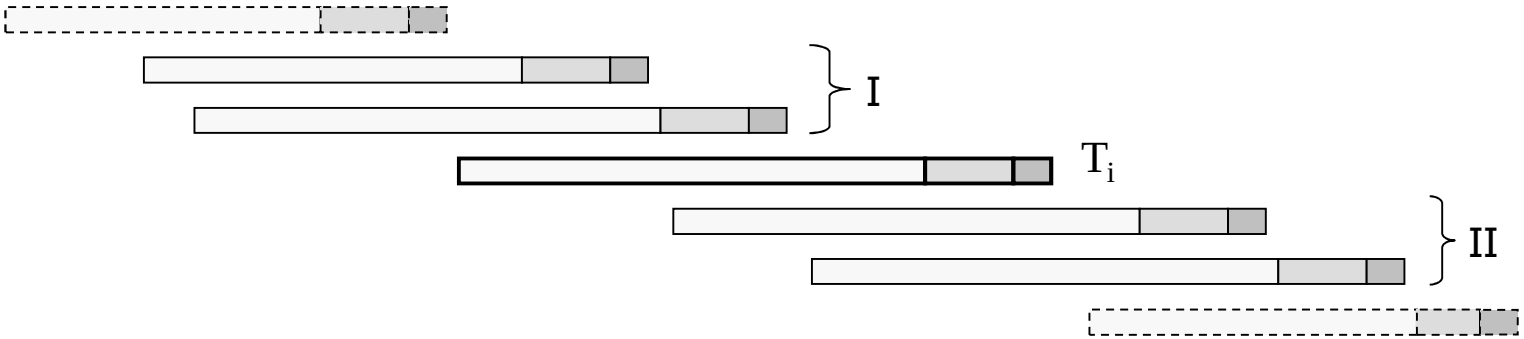
Optymistyczne sterowanie współbieżnością

- **Legalizowanie transakcji**
  - Test legalizacyjny (dla transakcji T_i)
    - Pomiędzy parami współbieżnych transakcji T_i i T_j ($i \neq j$)
    - T_i jest szeregowalna gdy spełnione są reguły

      1) T_j nie wolno czytać obiektów zapisywanych przez T_i
      2) T_i nie wolno czytać obiektów zapisywanych przez T_j
      3) T_i/T_j nie wolno zapisywać obiektów zapisywanych przez T_i/T_i
    - Zakłada się że legalizacja i pisanie są krótkie w porównaniu do fazy czytania i w tych fazach jest tylko jedna transakcja na raz, zatem **reguła 3 zawsze spełniona**
      - Np. realizacja zapisu w sekcji krytycznej
    - Dla sprawdzenia warunków **1** i **2**: testowanie par zbiorów czytania (ZbC) i pisania (ZbP) z transakcjami współbiegnącymi
- **2 rodzaje legalizacji**
  - Wsteczna
  - Wyprzedzająca

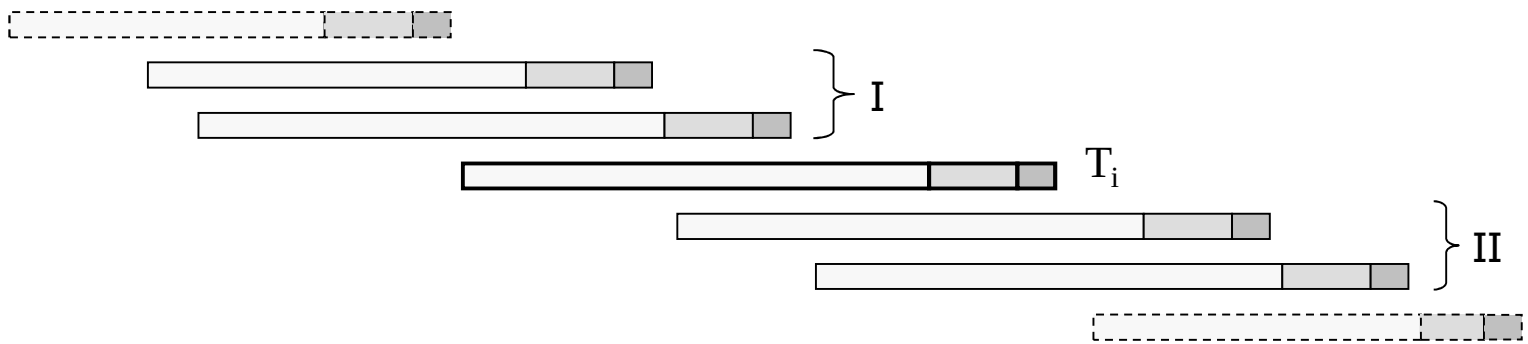
Optymistyczne sterowanie współbieżnością

- **Legalizacja wsteczna T_j**
  - Dla par (T_i,T_j) takich że $j<i$ i T_j nie zakończyła się przed rozpoczęciem T_i (rys. - transakcje I)
  - Reguła **1** jest spełniona ponieważ T_i pisze po T_j
  - Dla reguły **2**:
    - [Istnieje T_j takie że **ZbP**$(T_j) \cap$ **ZbC**$(T_i) \neq \emptyset$] $\Rightarrow$ ZANIECHAJ T_i
      w innym przypadku Legalizuj
    - Oznacza to, że:
      T_i nie wolno było czytać obiektów zapisanych przez T_j



Optymistyczne sterowanie współbieżnością

- **Legalizacja wyprzedzająca**
  - Dla par (T_i,T_j) takich że $j>i$ i T_i nie zakończyła się przed rozpoczęciem T_j (rys. – transakcje II)
  - Reguła **2** jest spełniona ponieważ T_i czyta przed zapisem T_j
  - Dla reguły **1**:
    - [Istnieje T_j takie że **ZbP**$(T_i) \cap$ **ZbC**$(T_j) \neq \emptyset$] $\Rightarrow$ ZANIECHAJ T_i
    - Bo T_i nie wolno zapisywać obiektów czytanych przez T_j
  - Jest kilka wariantów postępowania gdy reguła 1 nie jest spełniona



Porządkowanie z wykorzystaniem znaczników czasu

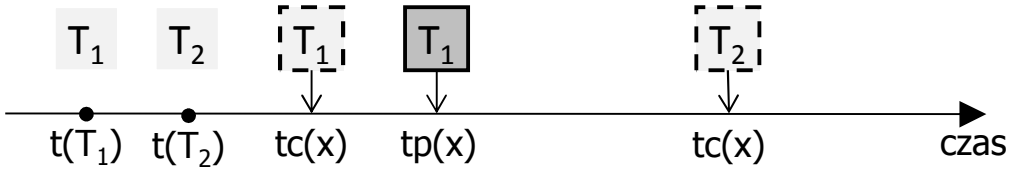
- **Pesymistyczne porządkowanie według znaczników czasu**
  - Każda transakcja T_i posiada swój znacznik czasowy $t(T_i)$
    - Równy czasowi rozpoczęcia transakcji
    - Znacznik jest unikalny (np. wg algorytmu Lamporta)
    - Wszystkie operacje tej transakcji są stemplowane tym czasem
  - Każdy obiekt danych x oznaczany jest dwoma znacznikami czasowymi:
    - **znacznik czasu zapisu tp(x)**
    - **znacznik czasu odczytu tc(x)**
  - Zapis realizowany jest na kopii tymczasowej
  - Obiekty tymczasowe mają własne znaczniki zapisu $tp_i(x)$

Porządkowanie z wykorzystaniem znaczników czasu

- **Pesymistyczne porządkowanie wg znaczników czasu**
  - Legalizowanie transakcji podczas wykonywania
  - Porządkowanie operacji – zasada:
    konfliktowe operacje wykonywane najpierw dla transakcji wcześniejszych
    - Tymczasowe wersje każdego obiektu danych są zatwierdzane w porządku określonym przez znaczniki czasu transakcji które je wytworzyły (istotna jest **kolejność transakcji !**)
      - Gdy jest potrzeba: oczekiwanie na zatwierdzenie zapisów poprzednich transakcji
  - W przypadku nierozwiązywalnego konfliktu operacji: wycofywanie transakcji z większym znacznikiem czasowym niż znacznik danej
- **Podstawowe cechy**
  - Nie występuje problem zakleszczania
  - Różne plany uzyskane w wyniku tego algorytmu mogą nie być możliwe dla algorytmu blokowania dwufazowego i odwrotnie

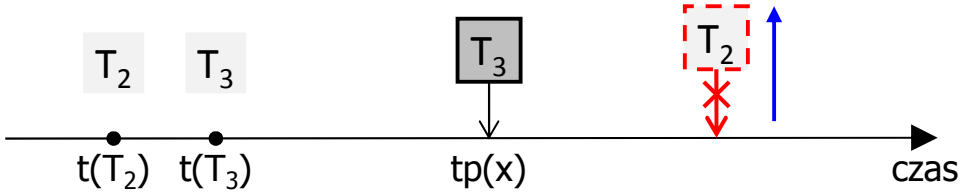
Porządkowanie z wykorzystaniem znaczników czasu

- **Zasada porządkowania operacji:**
  - Operacje **odczytu** są legalne tylko wtedy, gdy obiekt danych był **zapisywany tylko** przez **wcześniejsze** transakcje
    - Czytanie musi czekać na zatwierdzenie zapisu poprzednich transakcji
  - Operacja odczytu obiektu x : porównywanie $t(T_i)$ i $tp(x)$
    - $t(T_1) < t(T_2) < t(T_3)$
    - Legalny odczyt



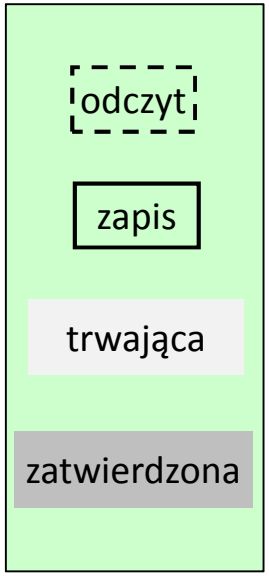
› $tp(x) < t(T_i) \Rightarrow tc(x) = \max[t(T_i), tc(x)]$

- Zaniechanie odczytu



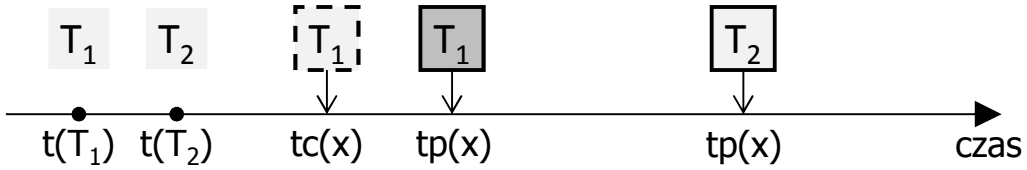
› $tp(x) > t(T_i) \Rightarrow$ zaniechaj T_i

Legenda:



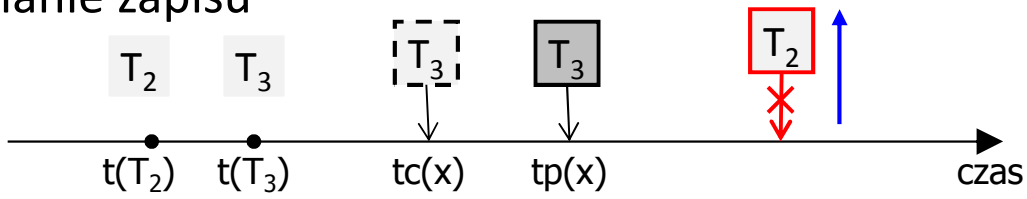
Porządkowanie z wykorzystaniem znaczników czasu

- **Zasada porządkowania operacji:**
  - Operacje **zapisu** są legalne tylko wtedy, gdy obiekt danych poprzednio był **czytany lub zapisywany tylko** przez **wcześniejsze** transakcje
  - Operacja zapisu obiektu x: porównywanie $t(T_i)$ z tc(x) i tp(x)
    - Legalny zapis



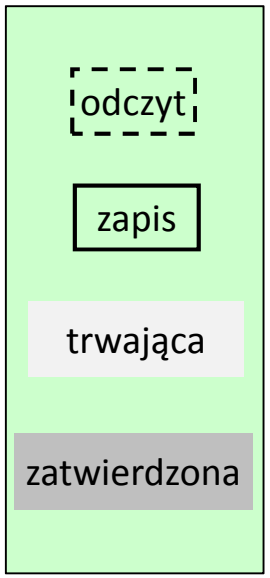
› $tp(x) < t(T_i) \Rightarrow tp(x) = \max[t(T_i), tp(x)]$
› $tc(x) < t(T_i) \Rightarrow tp(x) = \max[t(T_i), tp(x)]$

    - Zaniechanie zapisu



› $tp(x) > t(T_i) \Rightarrow$ zaniechaj T_i
› $tc(x) > t(T_i) \Rightarrow$ zaniechaj T_i

Legenda:

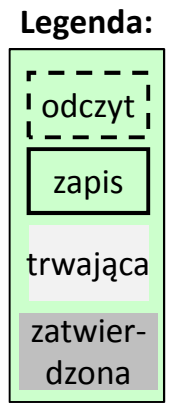


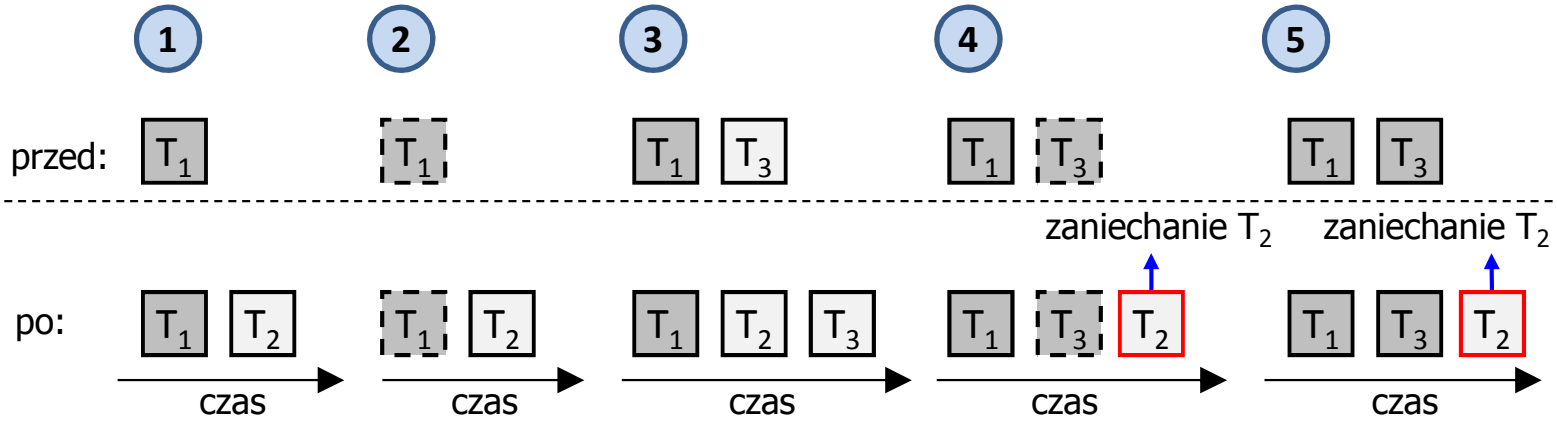
Porządkowanie z wykorzystaniem znaczników czasu

- **Reguła pisania**
  - Dla obiektu danych x zapisywanego przez transakcję T_i

```
IF ( ( t(Ti) ≥ tc(x) ) AND ( t(Ti) > tp(x) zatwierdzonej wersji ) )
{
   WYKONAJ pisz(x) na tymczasowej wersji AND tp(x) = t(Ti)
}
else
   ZANIECHAJ Ti
```

  - Operacje pisania dla T_2:pisz(x) – różne przypadki:
    - $t(T_1) < t(T_2) < t(T_3)$



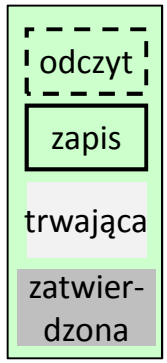


Porządkowanie z wykorzystaniem znaczników czasu

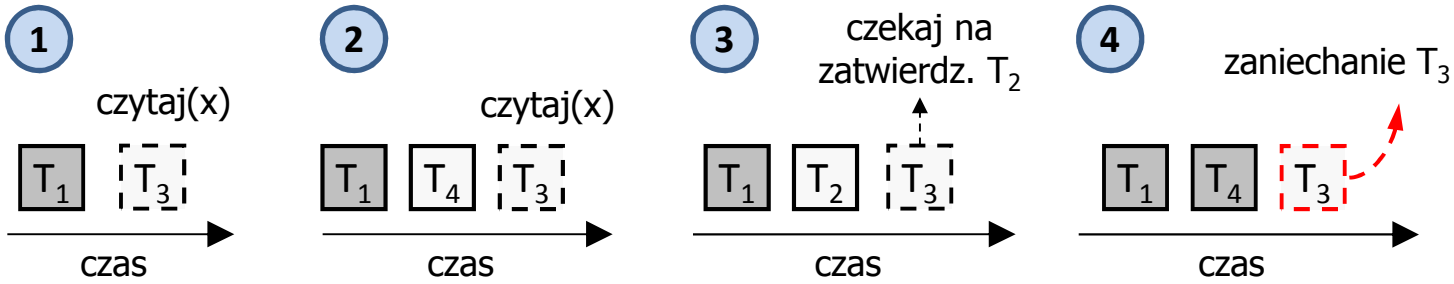
- **Reguła czytania**
  - Dla obiektu danych x czytanego przez transakcję T_i

```
IF  ( t(Ti) > tp(x) )  THEN                    // zatwierdzonej wersji x
{ x_max = max_{tp_j(x) < t(Ti)} (x)
   IF (x_max jest zatwierdzone) THEN {
      WYKONAJ czytaj(x)                        // na wersji zatwierdzonej x_max
   ELSE
      CZEKAJ na zatwierdzenie lub zaniechanie transakcji, która zapisała x_max
      POWTÓŻ regułę czytania
}}
ELSE
   ZANIECHAJ Ti                                // czytanie jest spóźnione
```

Legenda:



  - Operacje czytania dla T_3:czytaj(x) – różne przypadki:



Rozproszone systemy informatyczne

dr inż. Mariusz Fraś

Wiarygodność funkcjonalna systemów rozproszonych

- Tolerowanie awarii
- Osiąganie porozumienia
- Algorytmy elekcji
- Niezawodność komunikacji
  - Porządkowanie komunikatów

Wiarygodność funkcjonalna

- **Pojęcia**
  - **Wiarygodność funkcjonalna** systemu informatycznego (ang. ***dependability***)
    - własność, która pozwala jego użytkownikom mieć uzasadnione zaufanie do usług, które ten system dostarcza.
  - **Uszkodzenie / awaria** (ang. ***failure***)
    - stan systemu, w którym przestaje świadczyć usługę lub jest ona niezgodna ze specyfikacją
  - **Błąd** (ang. ***error***)
    - część stanu systemu, która jest odpowiedzialna za będące jego następstwem uszkodzenie
    - przyczyna uszkodzenia.
  - **Wada / defekt** (ang. ***fault***)
    - stwierdzona lub hipotetyczna przyczyna wystąpienia błędu.
- **Domena uszkodzenia**
  - Obszar gdzie uszkodzenie się objawia
    - **Uszkodzenie wartości** (ang. *value failure*) – system podaje nieprawidłową wartość (podejmuje nieprawidłową reakcję)
    - **Uszkodzenie synchronizacji** (ang. *timing failure*) – wartość jest prawidłowa ale pojawia się w niewłaściwym czasie

Wiarygodność funkcjonalna

- **Atrybuty wiarygodności funkcjonalnej**
  pozwalają określić właściwości oczekiwane od systemu i oszacować jakość systemu
  - **Dostępność** (ang. ***availability***)
    - Zdolność do natychmiastowego świadczenia usługi w każdej chwili
    - Miara: stosunek czasu w którym system jest sprawny do czasu w którym powinien być sprawny
  - **Niezawodność** (ang. ***reliability***)
    - Zdolność do nieprzerwanej bezbłędnej pracy
    - Miara: długookresowa praca bez uszkodzenia
  - **Bezpieczeństwo / pewność** (ang. ***safety***)
    - Zapewnienie że uszkodzenie nie powoduje katastrofalnych skutków
  - **Naprawialność / pielęgnowalność** (ang. ***maintainability***)
  - **Integralność** (ang. ***integrity***)
    - Niedopuszczalność niedozwolonych zmian stanu systemu
  - **Bezpieczeństwo** (ang. ***security***)
  - **Ochrona** (ang. ***confidentiality***)

Wiarygodność funkcjonalna

- **Środki osiągania wiarygodności funkcjonalnej**
  - Zapobieganie wadom / awariom
    - Działania na etapie projektowania i wytwarzania i testowania sprzętu i oprogramowania
  - Usuwanie wad – prowadzone na etapie tworzenia systemu
    - weryfikacja (czy zachowanie zgodne jest ze specyfikacją)
    - diagnoza i usuwanie defektów
  - Usuwanie awarii – podczas eksploatacji systemu
    - Rekonstrukcja uszkodzonych elementów systemu

  - **Tolerowanie błędów / awarii**
    - zachowanie ciągłości dostarczania usług pomimo występowania błędów lub awarii części systemu

Wiarygodność funkcjonalna

- **Zagadnienia tolerowania błędów/awarii**
  - Wykrywanie błędów (ang. *error detection*)
  - Maskowanie awarii
    - Maskowanie hierarchiczne
      - › Element/usługa wyższego poziomu maskuje awarie na niższym poziomie
    - Maskowanie grupowe
      - › Grupowanie elementów i maskowanie awarii poszczególnych członków grupy
  - Odzyskanie prawidłowego stanu systemu (ang. *system recovery*)

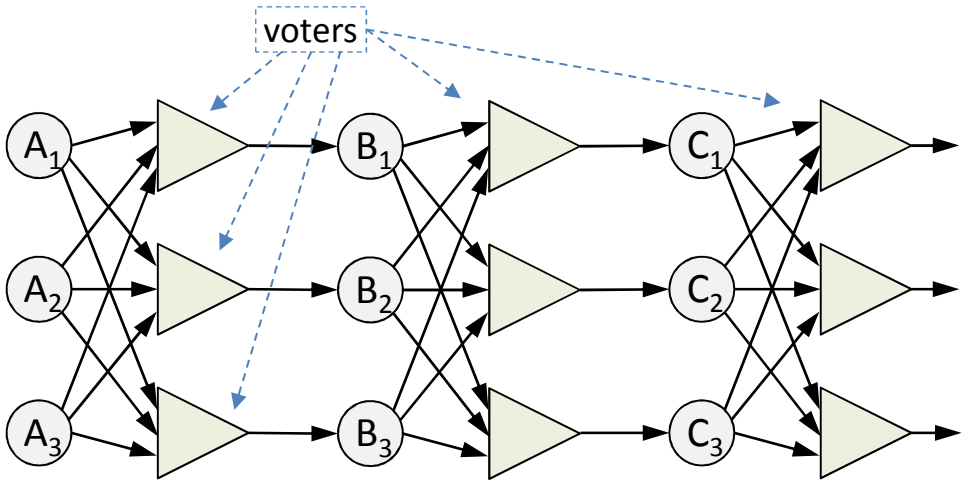
Maskowanie awarii

- **Maskowanie przez nadmiarowość (redundancję)**
  - Redundancja informacji
    - Dodatkowe dane (bity) umożliwiające odtwarzanie informacji
  - Redundancja czasu
    - Powielanie działań w przypadku wystąpienia awarii
  - Redundancja fizyczna
    - Zwielokrotnianie jednostek przetwarzania
      - Sprzętowe
      - Programowe

Maskowanie awarii

- **Zwielokrotnianie procesów**
  - Zwielokrotnianie procesów ⟶ grupy procesów współpracujących
  - Tolerowanie wadliwych procesów
    - Wynik określany na podstawie wyników poszczególnych członków grupy
      - › Od działających (lub najszybszych) … lub …
      - › Przez głosowanie większościowe
  - *Triple modular redundancy*
    - Reprodukcja wartości na podstawie większości poprawnych sygnałów
    - Zwiększona złożoność może zmniejszać niezawodność
      - › r – niezawodność pojed. elem.
      - › R – całkowita niezawodność:

$$R = r^3 + 3 \cdot r^2 \cdot (1-r) = 3 \cdot r^2 - 3 \cdot r^3$$

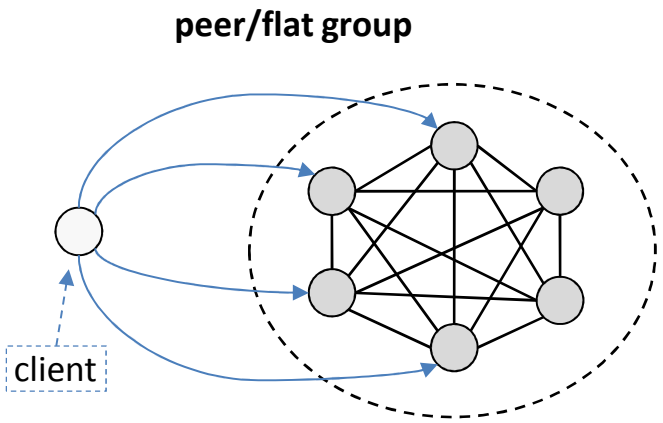


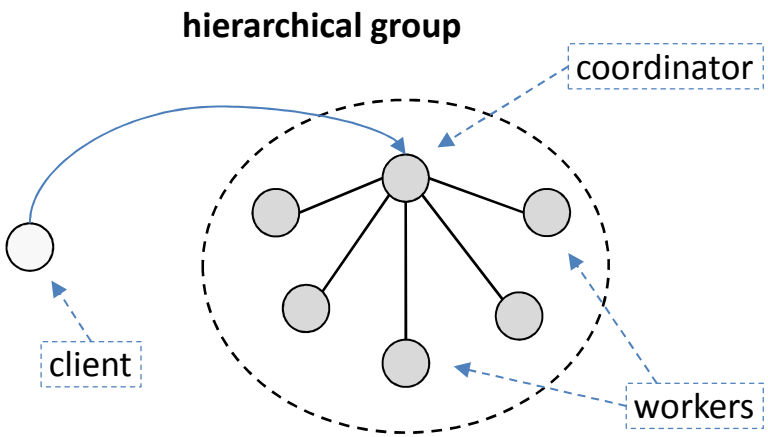
r=0,1 ⟶ R=0,028
r=0,5 ⟶ R=0,5
r=0,9 ⟶ R=0,97

    - Wskazuje na problem niezawodności syst. złożonych

Maskowanie awarii

- **Zwielokrotnianie procesów**
  - Rodzaje grup:
    - Równoprawne (ściśle synchronizowane)
      - › Żądania przetwarzane jednocześnie przez wszystkich członków
      - › Niewrażliwe na pojedynczy punkt awarii
      - › Trudne zarządzanie
    - Hierarchiczne (luźno synchronizowane)
      - › Wybrany węzeł podstawowy (koordynator) realizuje zamówienia
        - Propagacja / modyfikacja / rejestracja zamówień do węzłów roboczych
      - › Wrażliwość na awarię koordynatora
  - Wyróżniane problemy niezawodności:
    - **Niezawodność komunikacji grupowej**
    - **Awaria koordynatora**



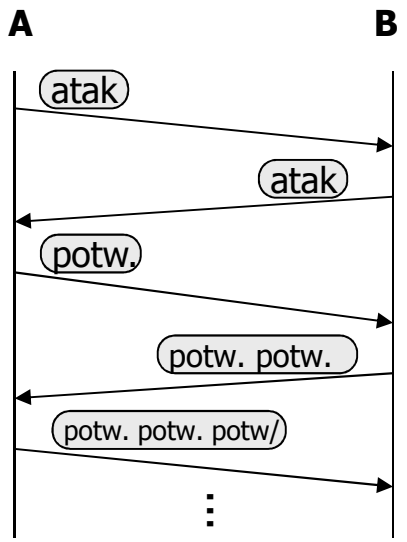


Maskowanie awarii

- **Stopień tolerancji k**
  - Wskaźnik dla tolerowania **k** uszkodzeń
    - **k** - liczba tolerowanych jednoczesnych awarii członków grupy
  - Zależność od charakteru awarii
    - Załamanie / awarie uciszające
      - › wymaganie **k+1** uczestników
    - Awarie wymagające głosowania
      - › wymaganie **2k+1** uczestników
    - Inne typy awarii
      - › mogą wymagać jeszcze większego stopnia tolerancji **(!)**

Osiąganie porozumienia

- **Porozumienie (ang. *agreement*)**
  - Osiąganie zgody co do wspólnej wartości uzyskane przez wszystkie sprawne procesy w skończonej liczbie kroków
  - Przykłady uzgodnień: synchronizacja działań (np. wzajemne wykluczanie), wybór koordynatora, zatwierdzenie transakcji
- **Problem dwóch armii**
  - Podjęcie wspólnej decyzji przez sprawne procesy
  - Problem: zawodne kanały komunikacji
    - Przykład: atak wymaga pewności wspólnej decyzji (osiągnięcia porozumienia / konsensusu)
    - Niepewność dostarczenia komunikatu wymaga potwierdzenia
    - Niepewność dotyczy również potwierdzenia, potwierdzenia potwierdzenia, … i tak w nieskończoność



  - **W systemie, w którym nie można zagwarantować dostarczania komunikatów w znanym, skończonym czasie, nie można zagwarantować osiągnięcia porozumienia, pomimo sprawnych procesów**

Osiąganie porozumienia

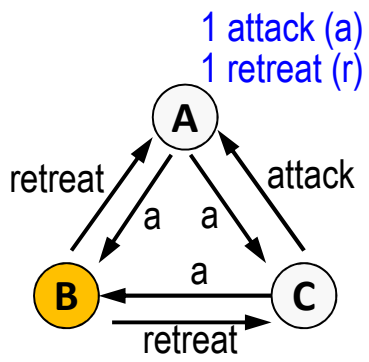
- **Model awarii uzgodnień rozproszonych (konsensusu)**
  - Większość procesów jest prawidłowych ale kilka jest wadliwych ("wrogich")
  - "Wróg" jest źródłem niepowodzenia uzgodnień
    - "Wróg" w ogólności może:
      - Testować/sprawdzać stan globalny systemu
      - Harmonogramować protokół wykonania
      - Niszczyć (nie produkować) lub modyfikować wiadomości
  - **Synchroniczność działania**
    - Poprawny proces odpowiada w określonym limicie czasowym T
    - Proces, który nie odpowiada w tym czasie jest uznawany za wadliwy
  - Dla systemów synchronicznych
    - Da się określić protokoły osiągania konsensusu
  - Dla systemów nie-synchronicznych konsensus jest w ogólności niemożliwy
    - Nie istnieje algorytm, który może zagwarantować, że wszystkie poprawne procesy osiągną porozumienie w skończonym czasie

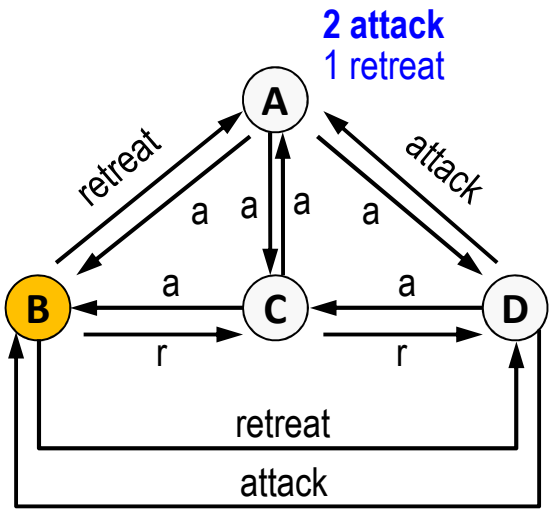
Osiąganie porozumienia

- **Problem bizantyjskich generałów**
  - Ilustruje problem osiągania porozumienia dla awarii bizantyjskich (w uproszczeniu: awarii dowolnych)
  - Procesy poprawne (generałowie lojalni) – muszą uzgodnić działania
    - Ustalenie poprawnej wartości
  - Procesy wadliwe (generałowie zdrajcy)
    - Mogą wysłać dowolną wiadomość (niepoprawną) lub żadną
  - Założenie niezawodnej komunikacja punkt-punkt
    - Wiadomości widziane w tym samym porządku
  - Założenie: synchroniczność systemu
  - Wymaganie:
    poprawne procesy powinni ustalić wspólnie poprawną wartość
    inaczej:
    lojalni generałowie podejmują tą samą decyzję

Osiąganie porozumienia

- **Model uwiarygodnionych bizantyjskich generałów**
  - Jest identyfikacja nadawców
  - Jest niezawodna komunikacja
  - Semantyka awarii obejmuje
    - Błędy odpowiedzi
    - Błędy pominięć
    - Błędy punktualności
- **Warunek osiągnięcia porozumienia dla awarii bizantyjskich**
  - Tolerowanie **k** uszkodzeń $\longrightarrow$ potrzeba **3k+1** członków grupy
- **Pełny model bizantyjski**
  - Brak sposobu ustalenia tożsamości nadawcy komunikatu (podszywanie się pod innych uczestników)
  - Jeszcze trudniejszy problem

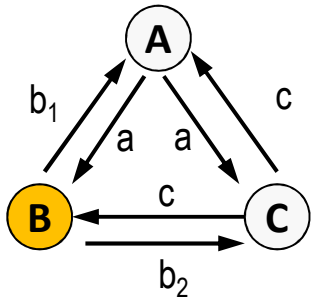


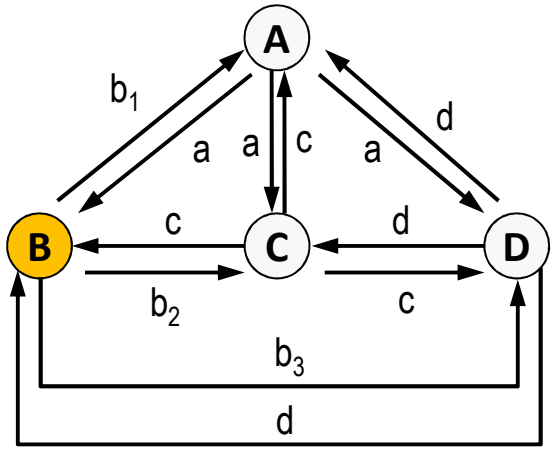


Osiąganie porozumienia

- **Ilustracja problemu biz. gen. :**
  - Atak armii dowodzonej przez **n** generałów
  - **k** generałów jest zdrajcami
    - usiłują nie dopuścić do porozumienia
    - procesy przesyłają nieprawdziwe i sprzeczne informacje
  - Zadanie: porozumienie się lojalnych generałów co do liczebności oddziałów (uzgodnienie wspólnej wartości)
  - Porozumienie to: **wektor V liczebności oddziałów**
    - **V[i]** liczebność oddziałów **i**-tego lojalnego generała …lub …
    - wartość nieokreślona w przypadku zdrajcy
  - Założenie: zdrajcy nie współpracują

a, c, d – przekazywane wartości
b_1, b_2, b_3 – wadliwe wartości





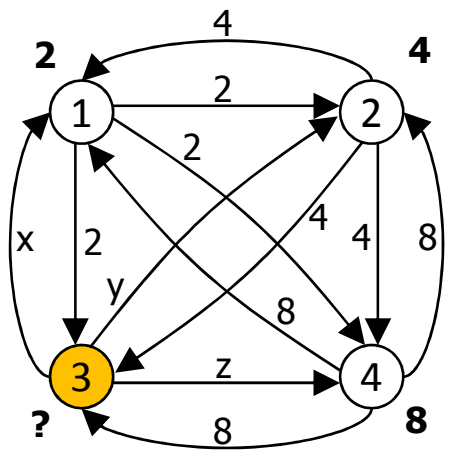
Osiąganie porozumienia

- **Algorytm Lamporta**

1. Wyślij do wszystkich generałów informację o liczebności swojego oddziału – **send(V[i])**

2. Zbierz wyniki od wszystkich generałów

- Przykład
  - Dane
    - n=4, k=1
    - Wartości: P1: 2, P2: 4, P4: 8, P3: ? (zdrajca)
  - Krok 1



  - Krok 2

1 otrzymał (2,4,x,8)
2 otrzymał (2,4,y,8)
3 otrzymał (2,4,..,8)
4 otrzymał (2,4,z,8)

Osiąganie porozumienia

- **Algorytm Lamporta**
  3. Przekaż wektor wyników (liczebność poszczególnych oddziałów) do wszystkich pozostałych generałów – **send(V)**
  4. Sprawdź wszystkie pozycje wszystkich wektorów
     - wartość będąca w większości staje się wynikiem
     - brak większości → pozycja nieokreślona
       - Dla procesów wadliwych (zdrajców)

- Przykład
  - Krok 2

1 otrzymał (2,4,x,8)
2 otrzymał (2,4,y,8)
3 otrzymał (2,4,..,8)
4 otrzymał (2,4,z,8)

  - Krok 3 i 4

1 otrzymał	2 otrzymał	4 otrzymał
(2,4,y,8)	(2,4,x,8)	(2,4,x,8)
(a,b,c,d)	(e,f,g,h)	(2,4,y,8)
(2,4,z,8)	(2,4,z,8)	(i,j,k,l)
(2,4,?,8)	**(2,4,?,8)**	**(2,4,?,8)**

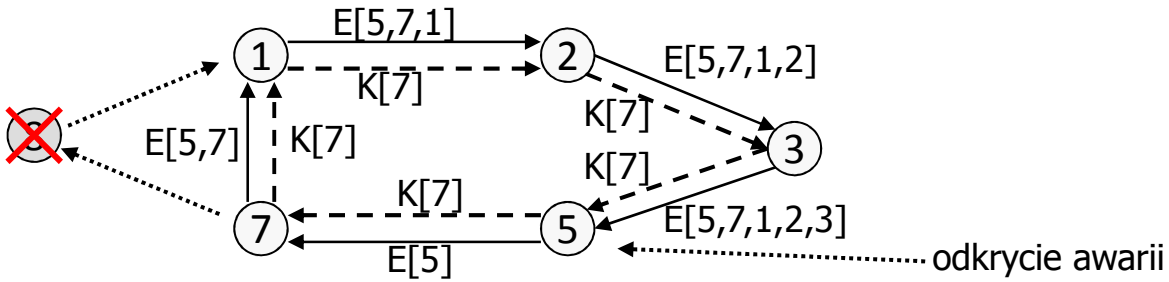
Algorytmy elekcji

- **Procedury rozstrzygające o odtworzeniu kopii procesu koordynatora w systemie (grupie), który uległ awarii**

- **Założenia i wymogi**
  - Komunikacja jest niezawodna
  - Każdy aktywny proces ma jednoznaczny numer procesu (o funkcji priorytetu) **NP**
  - Procesy znają numery pozostałych procesów
  - Nie wiadomo które z procesów działają
  - Koordynatorem jest proces o największym NP
  - Informacja o nowym koordynatorze musi być dostarczona do wszystkich procesów
  - Musi być mechanizm identyfikacji bieżącego koordynatora przez proces przywracany do działania po awarii
  - Jeśli elekcja rozpocznie się to musi się zakończyć powodzeniem
  - **T** – próg czasowy odpowiedzi na wysłanie komunikatu do procesu, którego przekroczenie oznacza, że proces uległ awarii

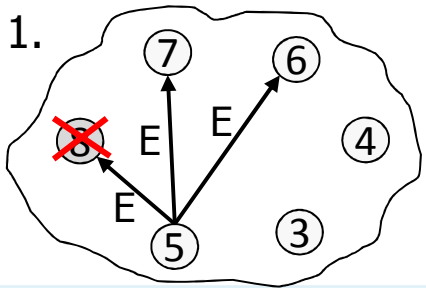
Algorytmy elekcji

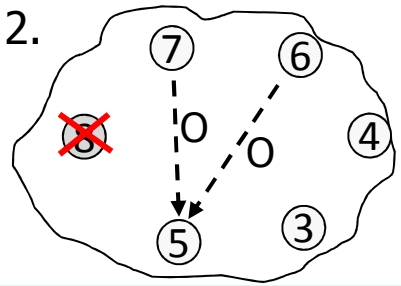
- **Algorytm pierścieniowy**
  - System z pierścieniową, jednokierunkową wymianą komunikatów
    - Przykład systemu z uporządkowanym schematem komunikacji
  - Procedura
    - Proces wykrywający brak koordynatora wysyła komunikat E, z listą zawierającą jego NP, do następnego czynnego procesu
    - Proces otrzymujący komunikat E dołącza do listy swój NP i wysyła komunikat do następnego czynnego procesu
    - Po okrążeniu pierścienia proces inicjujący elekcję wybiera proces o najwyższym NP jako koordynatora i wysyła komunikat K(NP_koordynatora)
    - Komunikat K jest usuwany po okrążeniu pierścienia



Algorytmy elekcji

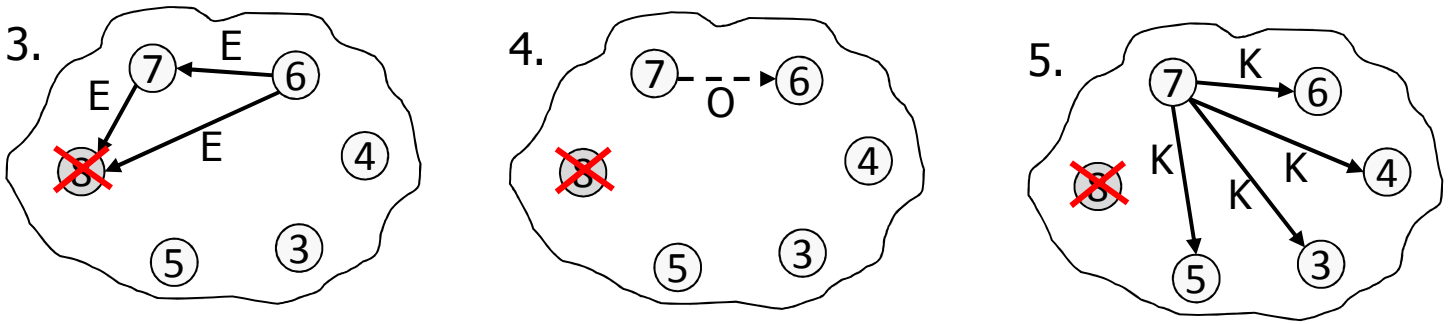
- **Algorytm tyrana**
  - Dla systemów o swobodnej wymianie komunikatów
  - Procedura elekcji po wykryciu przez P_i awarii koordynatora:
    - P_i rozpoczyna elekcję siebie na koordynatora
      - › Wysłanie komunikatu elekcji **E** do wszystkich procesów z wyższym NP i czekanie na odpowiedź **O** przez czas T_E
      - › Jeśli brak odpowiedzi (nie ma działających procesów o wyższym NP), to wybór siebie (procesu P_i) na koordynatora
        - - Wysłanie do wszystkich procesów o niższym NP komunikatu informującego **K** o elekcji siebie na koordynatora
      - › Jeśli jest odpowiedź, to czekanie przez czas T_K na komunikat **K**, że proces o wyższym NP został koordynatorem
        - - Odnotowanie nowego koordynatora
        - - Jeśli komunikat **K** nie nadszedł, to wznowienie elekcji od początku





Algorytmy elekcji

- **Algorytm tyrana (c.d.)**
  - Procedura elekcji – działania procesów P_j podczas elekcji P_i
    - $NP_j > NP_i$ – odebranie komunikatu **E** (P_i rozpoczął elekcję)
      - › P_j wysyła odpowiedź **O** do P_i
      - › P_j rozpoczyna procedurę elekcji siebie na koordynatora
    - $NP_j < NP_i$ – odebranie komunikatu **K**
      - › Odnotowanie, że P_i jest nowym koordynatorem
  - Proces po awarii natychmiast rozpoczyna algorytm elekcji
    - Jeśli ma najwyższy NP, to zostaje nowym koordynatorem



Niezawodna komunikacja

- **Zagadnienia**
  - **Komunikacja punkt-punkt**
    - Błędy: pominięcia, punktualności, samowolne i załamania
    - Niezawodność kanałów komunikacji
      - › Niezawodne protokoły komunikacji warstwy transportowej (np. TCP)
      - › Wykrywanie błędów
        - Podział informacji na ramki
        - Bitowa detekcja błędów
        - Numerowanie ramek
      - › Korekcja błędów
        - Redundancja informacji w pakiecie
        - Maskowanie błędów ominięcia – retransmisja pakietów
  - **Niezawodność komunikacji klient-serwer**
    - RPC w sytuacjach awaryjnych
  - **Niezawodna komunikacja grupowa**
    - Rozsyłanie <u>niezawodne</u> i rozsyłanie <u>niepodzielne</u>
    - <u>Porządkowanie komunikatów</u>

Niezawodność komunikacji klient-serwer

- **Obsługa błędów i awarii**
- **Możliwe problemy w komunikacji klient-serwer**
  - Klient nie może zlokalizować serwera
  - Zaginął komunikat zamawiający od klienta do serwera
  - Zaginęła odpowiedź od serwera do klienta
  - Serwer uległ awarii po odebraniu zamówienia
  - Klient uległ awarii po wysłaniu zamówienia

- **Typy awarii RPC (elementarne/ogólne – jako ilustracja dla modelu klient-serwer):**
  - Awaria komunikacji
  - Awaria serwera
  - Awaria klienta

Niezawodność RPC

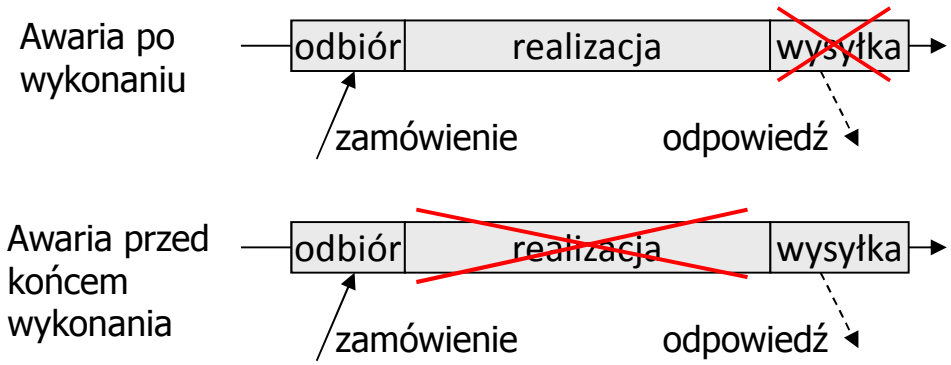
- **Awarie komunikacji**
  - Brak odpowiedzi na zamówienie
    - Zaginęło zamówienie od klienta dla serwera
    - Serwer działa powoli
    - Zaginęła odpowiedź od serwera do klienta
  - Postępowanie
    - Kontrola czasu w oczekiwaniu na odpowiedź
    - Retransmisja komunikatu
      - › Operacje idempotentne – wielokrotne wykonanie jest bezpieczne

Operacja idempotentna to taka operacja której wielokrotne wykonanie powoduje taki sam skutek jakby była wykonana jednokrotnie

      - › Operacje nie idempotentne → problem
        Potrzebne wtedy:
        - Dodatkowa informacja w komunikacji i retransmisji
        - Filtrowanie zamówień

Niezawodność RPC

- **Awarie serwera**
  - 2 przypadki awarii
  - Brak możliwości rozróżnienia dla klienta



- **3 semantyki RPC**

Semantyka wywołania RPC	Retransmisja komunikatu zamawiającego	Filtracja powtórzeń	Powtórne wykonanie lub retransmisja odpowiedzi
Być może	NIE	NIE DOTYCZY	NIE DOTYCZY
Co najmniej raz	TAK	NIE	POWTÓRNE WYKONANIE
Co najwyżej raz	TAK	TAK	RETRANSMISJA ODPOWIEDZI

Niezawodność RPC

- **Awarie klienta**
  - Awaria klienta: → proces sierota (ang. *orphan*)
    - Blokowanie zasobów
    - Niepotrzebne obciążanie serwera
  - Procedury usuwania sierot
    - Eksterminacja – jawna likwidacja
      - › Namiastka klienta rejestruje trwale zamówienia
      - › Po wznowieniu usuwa sierotę
      - › Wady:
        - Duży koszt rejestrowania
        - Nieskuteczność gdy: awaria sieci, klient nie wznowi się → osierocone potomstwo (sieroty wyprodukowały sieroty)
    - Reinkarnacja – globalne usuwanie sierot klienta
      - › Po wznowieniu klienta rozgłaszana nowa epoka
      - › Klient rozgłasza ją wszędzie
      - › Wszystkie procesy tego klienta ze starej epoki są usuwane
      - › Gdy klient nie wznowi się, wtedy jak w eksterminacji

Niezawodność RPC

- **Awarie klienta**
  - Procedury usuwania sierot
    - Łagodna reinkarnacja
      - Selektywne usuwanie sierot
      - W nowej epoce wyszukiwane są procesy bez właściciela
    - Przeterminowanie
      - Przydzielanie czasu na realizację procedury
      - Procedura zdalna otrzymuje kwant T na wykonanie zadania
      - Po wyczerpaniu czasu prośba do klienta o dodatkowy kwant
      - Brak prolongaty $\rightarrow$ likwidacja procesu
      - Odczekanie przez klienta czasu T likwiduje sieroty
        - Problem doboru T
  - Usuwanie sierot może być niebezpieczne
    - Np. problem blokad zasobów

Niezawodna komunikacja grupowa

- **Komunikacja grupowa (ang. *a group/collective communication*)**
  - Grupa (procesów)
    - Zbiór procesów uczestniczących we wspólnym przetwarzaniu i komunikujących się poprzez przekazywanie wiadomości
    - Wspiera cechy systemów rozproszonych:
      - Zwielokrotnianie usług
      - Tolerowanie awarii
      - Lepsza wydajność (przez zwielokrotnianie kopii)
- **Komunikacja grupowa obejmuje następujące aspekty:**
  - Zarządzanie grupami procesów
    - Usługa członkostwa (ang. *membership service*)
  - Mechanizm realizujący rozsyłanie (rozgłaszanie) wiadomości (ang. *multicast/broadcast*)
    - Typowa komunikacja w grupie procesów – wysyłanie komunikatu **do wszystkich**
    - Zagadnienia niezawodności i niepodzielności rozsyłania
    - Zagadnienia uporządkowania dostarczania komunikatów

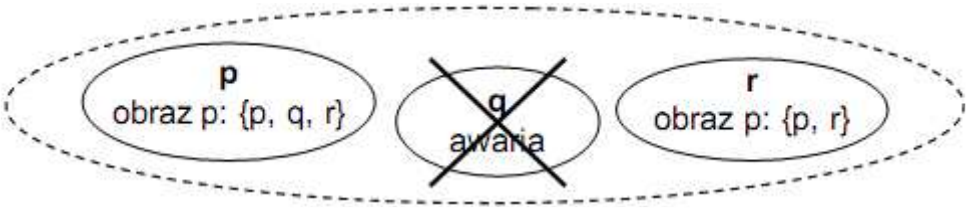
Niezawodna komunikacja grupowa

- **Rozsyłanie / rozgłaszanie**
  - **Niezawodne**
    - Założenie: procesy działają poprawnie w grupie (przynależność jest stała)
    - **<u>Zagwarantowanie</u> (lub maksymalizacja gwarancji) że jeśli proces P nie opuszcza grupy, komunikat *m* będzie dostarczony do P**
  - **Niepodzielne**
    - Założenie: procesy ulegają awariom (mogą opuszczać grupę lub dołączać się)
    - **<u>Zapewnienie</u>, że komunikat *m* zostanie dostarczony do wszystkich członków grupy albo nie zostanie dostarczony do żadnego procesu**
  - Własności niezawodnego rozgłaszania
    - Ważność (ang. *validity*) – jeśli jakiś poprawny proces rozgłosi wiadomość *m*, to w skończonym czasie wszystkie poprawne procesy dostarczą *m*
    - Zgodność (ang. *agreement*) – jeśli jakiś poprawny proces dostarczy wiadomość *m*, to w skończonym czasie wszystkie poprawne procesy dostarczą *m*
    - Integralność (ang. *integrity*) – dla każdej wiadomości *m*, każdy poprawny proces dostarcza *m* najwyżej raz i pod warunkiem, że *m* została wcześniej rozgłoszona przez jakiś proces

Rozsyłanie niepodzielne

NIEOMAWIANE

- **Rozsyłanie niepodzielne**
  - Rozsyłanie w warunkach awarii procesów oraz zmian przynależności grupowej
    - … i ewent. awarii kanałów komunikacyjnych
  - Dostarczanie komunikatu do **wszystkich** niewadliwych członków bieżącej grupy
    - Wszyscy członkowie powinni uzgodnić bieżącą przynależność grupową
  - **Obraz grupy** (ang. *view*) – grupa, widziana w danej chwili czasu rzeczywistego w pojedynczym procesie;
    - Obraz grupy jest generowany przez usługę członkostwa
    - Ogólnie, różne procesy mogą mieć różne obrazy tej samej grupy w tym samym momencie



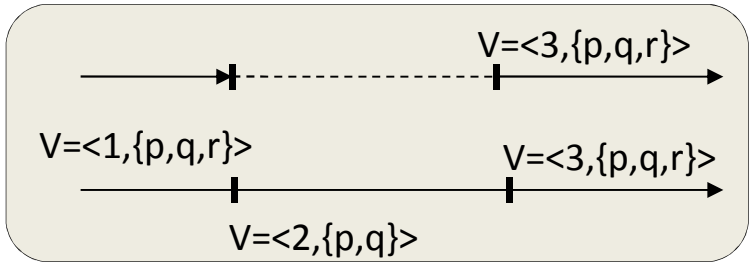
Rozsyłanie niepodzielne

NIEOMAWIANE

- **Model komunikacji w grupie**
  - Wysłanie – **`wyślij(p,m)`**
  - Odbiór – **`odbierz(p,m)`**
  - Dostarczenie – **`dostarcz(p,m)`**
  - Obraz grupy **`V=<id,{P_i}>`**
    - Samozawieranie (proces należy do własnego obrazu grupy
    - Zmiana obrazu
      `cv(p,V) = cv(p,<id,członkowie>)`
    - Monotoniczność lokalna
      › identyfikator danego obrazu jest większy od identyfikatorów wcześniejszych
      `cv(p,V1) ↦ cv(p,V2) ⇒ V1.id < V2.id`
  - Usługa członkostwa
  - Unikalność wiadomości
    - Każda wiadomość jest niepowtarzalna
    - Np. unikalny_id = id_nadawcy + numer_sekwencyjny (lub ZC)

← warstwa komunikacji

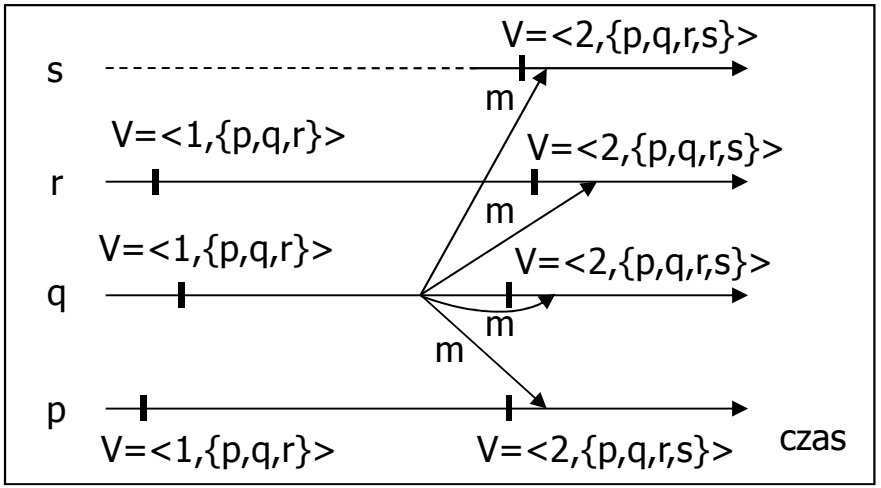
← warstwa aplikacji

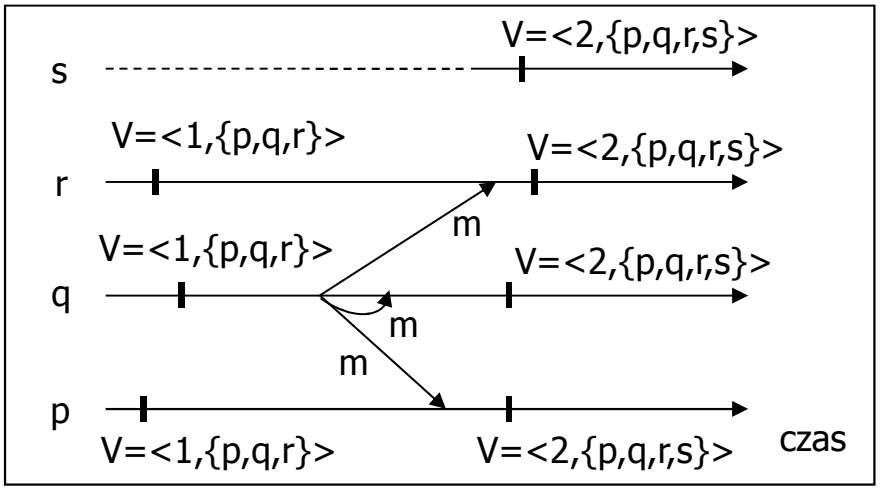


Rozsyłanie niepodzielne

NIEOMAWIANE

- **Rozsyłanie - własności**
  - Brak samogeneracji
    - Odebrana wiadomość jest wcześniej wysłana
  - Brak powielania
    - Jeden proces może odebrać wiadomość tylko jeden raz
  - Tryby dostarczenia:
    - W tym samym obrazie przez wszystkie procesy
      albo
    - W obrazie z momentu wysłania





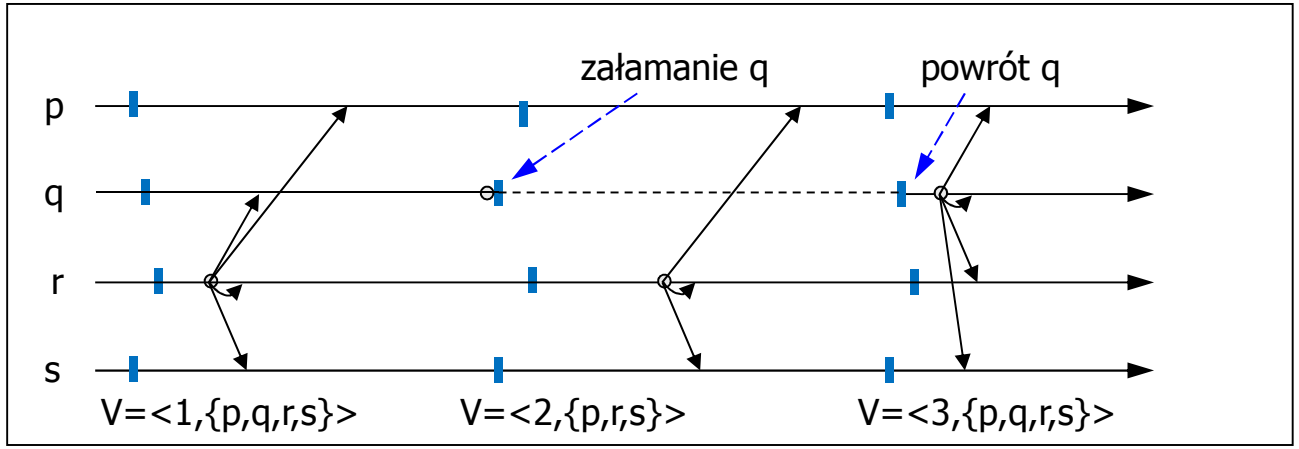
Rozsyłanie niepodzielne

NIEOMAWIANE

- **Wirtualna synchronizacja**
  - Procesy są synchronizowane kolejnymi zmianami obrazów
  - Dostarczanie do wszystkich procesów tych samych zbiorów wiadomości w tych samych obrazach
    - Dla każdego procesu p, q:

$$(cv(V,V') \wedge p,q \in V \wedge \textit{dostarcz}(p,m) \text{ w } V) \Rightarrow \textit{dostarcz}(q,m) \text{ w } V$$

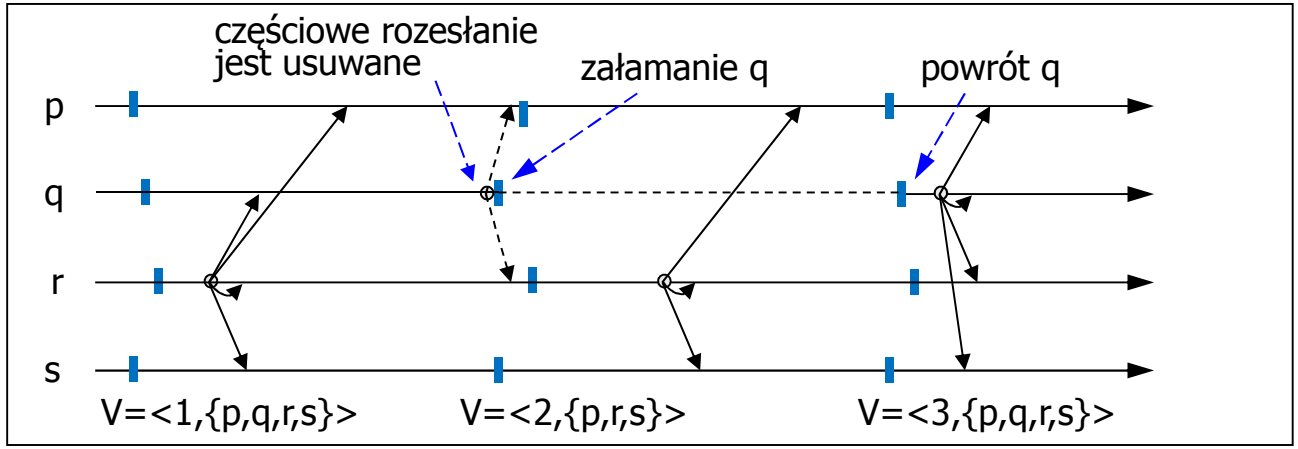
      › wszystkie procesy ze starego obrazu przed zainstalowaniem nowego obrazu muszą dostarczyć każdą wiadomość, którą dostarczył choć jeden proces w starym obrazie



Rozsyłanie niepodzielne

NIEOMAWIANE

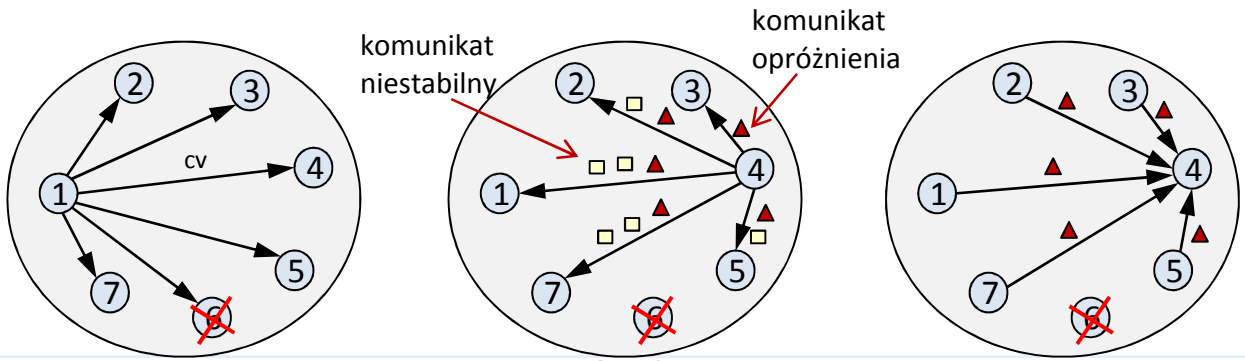
- **Wirtualna synchronizacja**
  - Jeśli w czasie rozsyłania *m* pojawia się *cv*(V,V')
    - Są dwa rozsyłania
    - Trzeba zagwarantować, że:
      - albo *m* jest dostarczone wszędzie przed *cv* w widoku V
      - albo *m* jest wszędzie ignorowane
    - Musimy zapewnić uporządkowanie całkowite
  - Jest tylko jeden przypadek gdy *m* może się nie udać
    - Proces wysyłający *m* ulega awarii



Rozsyłanie niepodzielne

NIEOMAWIANE

- **Rozgłaszanie niepodzielne w systemie ISIS**
  - Dostarczenie komunikatów do wszystkich niewadliwych procesów w widoku V_i zanim nastąpi kolejna zmiana składu grupy
  - Założenia
    - Niezawodna komunikacja punkt-punkt
    - Odbiór komunikatów w porządku wysyłania na poziomie sieci
  - Algorytm:
    - Przechowywanie komunikatu przez każdy proces w grupie do czasu upewnienia się że wszyscy go odebrali
    - Dostarczanie komunikatów ustabilizowanych
      - Odebranych przez wszystkich
    - Po sygnale zmiany widoku na V_{i+1} wysyłanie komunikatów nieustabilizowanych
    - Wysyłanie komunikatu opróżnienia
  - Komunikat opróżnienia jest zawsze odbierany po odbiorze komunikatów nieustabilizowanych



Rozsyłanie uporządkowane

- **Rodzaje uporządkowania**
  - Rozsyłanie nieuporządkowane
  - FIFO

$$[\text{wyślij}(P_i,m_1) \mapsto \text{wyślij}(P_i,m_2)] \Rightarrow [\text{dostarcz}(P_j,m_1) \mapsto \text{dostarcz}(P_j,m_2)]$$

    - Wiadomości wysyłane przez dany proces są dostarczane do odbiorców w takiej samej kolejności, w jakiej wysyłał je ten proces
  - Przyczynowe

$$[\text{wyślij}(P_i,m_1) \mapsto \text{wyślij}(P_j,m_2)] \Rightarrow [\text{dostarcz}(P_k,m_1) \mapsto \text{dostarcz}(P_k,m_2)]$$

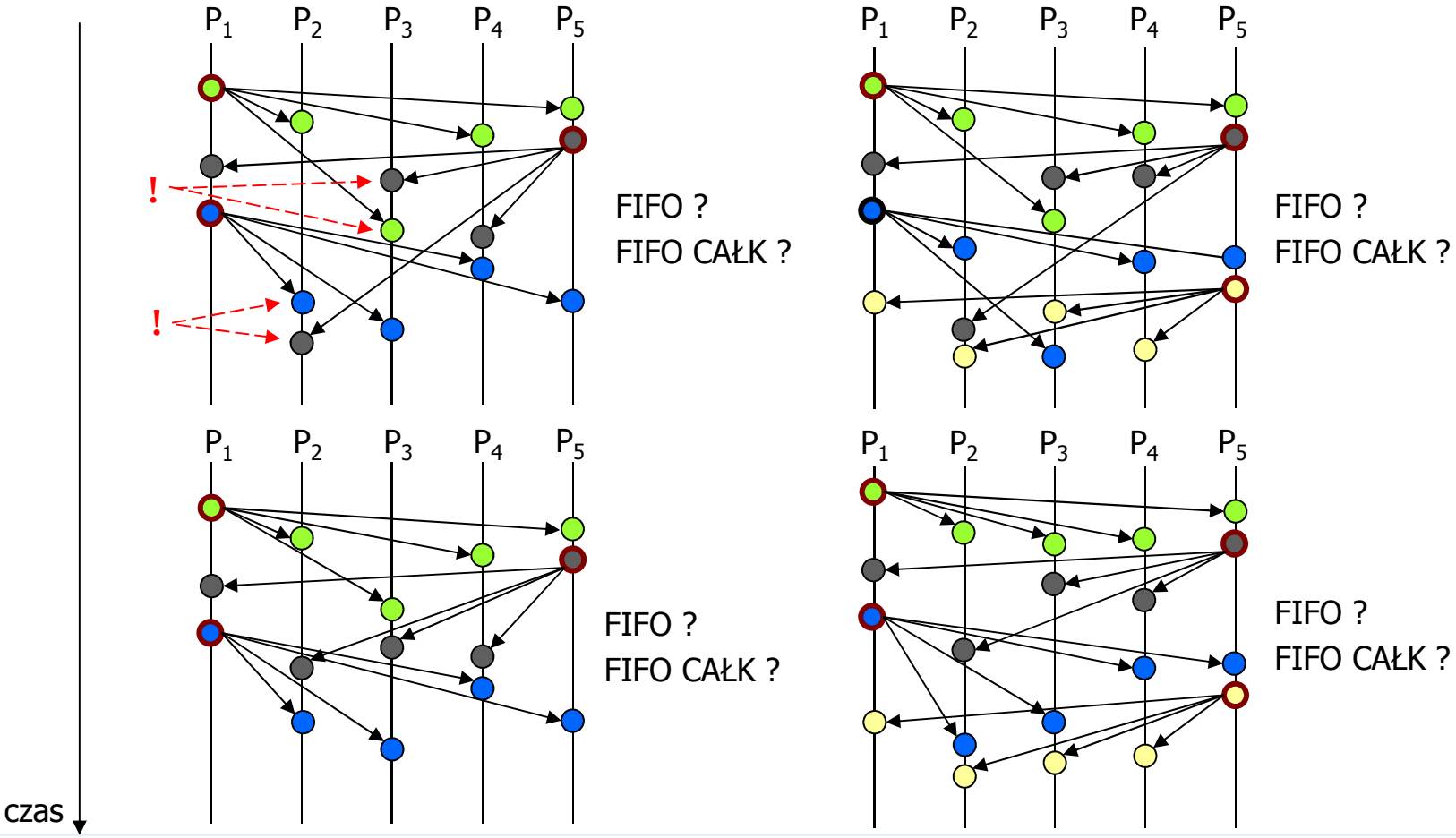
    - Dostarczanie wiadomości z zachowaniem przyczynowego uporządkowania
    - Uporządkowanie przyczynowe implikuje uporządkowanie FIFO
  - Całkowite
    - Dla każdego poprawnego procesu P_i i P_j dostarczających wiadomości m i m' :

$$[\text{dostarcz}(P_i,m) \mapsto \text{dostarcz}(P_i,m')] \Leftrightarrow [\text{dostarcz}(P_j,m) \mapsto \text{dostarcz}(P_j,m')]$$

    - Wszystkie wiadomości są dostarczane do wszystkich procesów w identycznej kolejności

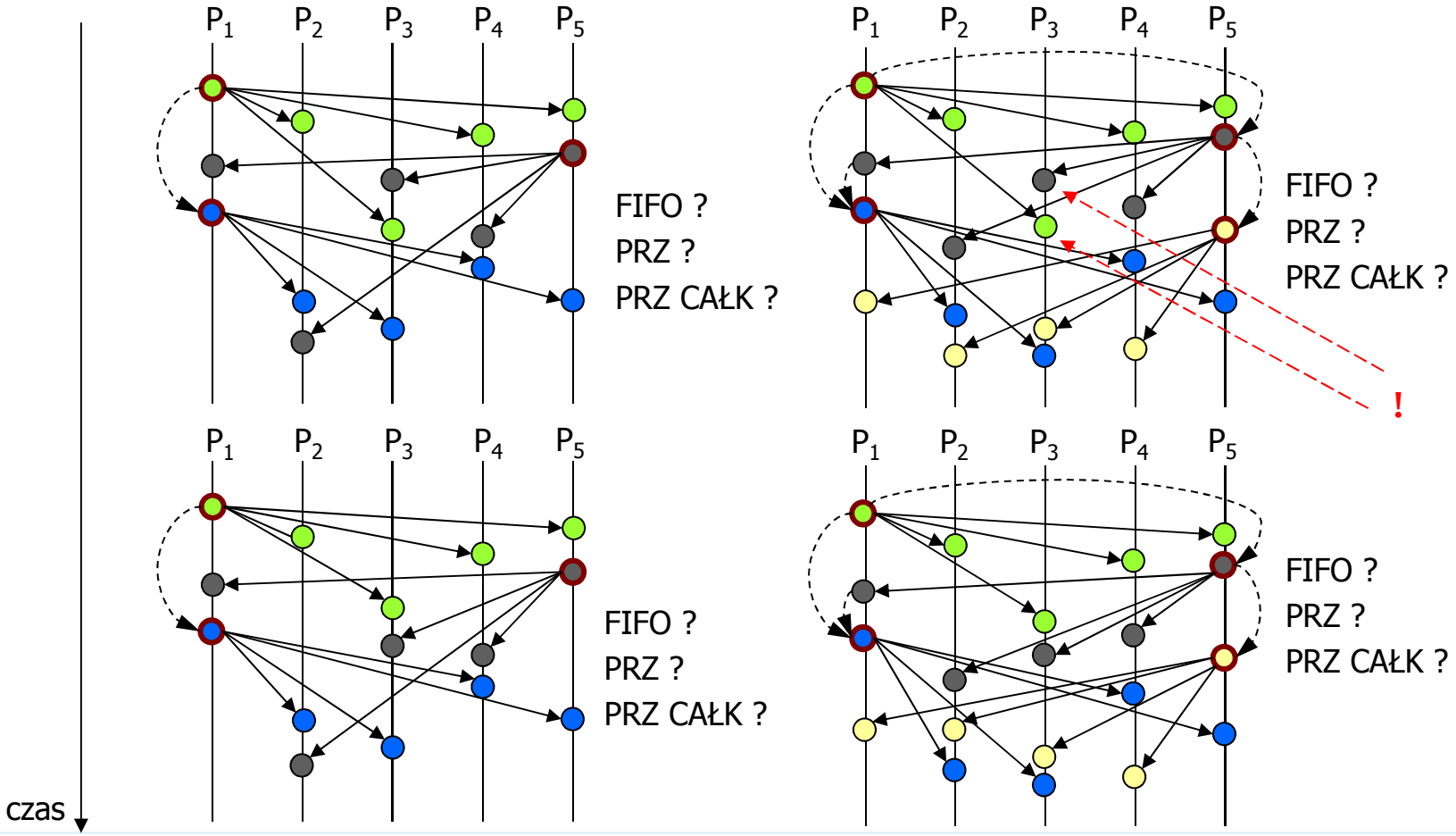
Rozsyłanie uporządkowane

- **Uporządkowania – przykłady**



Rozsyłanie uporządkowane

- **Uporządkowania – przykłady**



Niezawodne rozsyłanie

NIEOMAWIANE

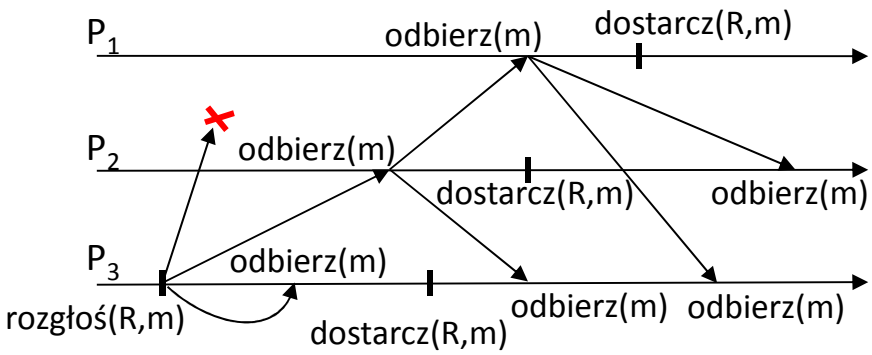
- **Algorytm RB (ang. *Reliable Broadcast*)**
  - Rozsyłanie nieuporządkowane
  - Po odebraniu najpierw rozgłoszenie potem dostarczenie

Algorytm dla procesu p

rozgłoś(R, m):
  m = m + wysyłający(m)
  wyślij(m) do wszystkich procesów, łącznie z p

odbierz(m):
  if wysyłający(m) ≠ p **then**
    wyślij(m) do wszystkich członków
    dostarcz(R, m)

  - Przy założeniu, że: *każde dwa poprawne procesy są połączone przez ścieżkę poprawnych łączy i procesów*, można udowodnić, że: *algorytm RB realizuje niezawodne rozgłoszenie*



Niezawodne rozsyłanie FIFO

NIEOMAWIANE

- **Algorytm FIFO-RB (ang. *FIFO-ordered RB*)**
  - Każda wiadomość ma numer sekwencyjny
  - *worek* – pamięta wiadomości odebrane ale nie dostarczone przez FIFO-RB
  - *następna*[q] – zawiera numer sekwencyjny wiadomości od procesu q
  - Wiadomości czekają w worku na dostarczenie w odpowiedniej kolejności
    - Przy dostarczaniu sprawdz. czy dana wiadomość jest następną w kolejności od danego procesu
    - Dostarczanie kolejno

```
Inicjalizacja:
  worek := ∅
  następna[q] := 1 dla każdego q
```

```
rozgłoś(F, m):
  rozgłoś(R, m)
```

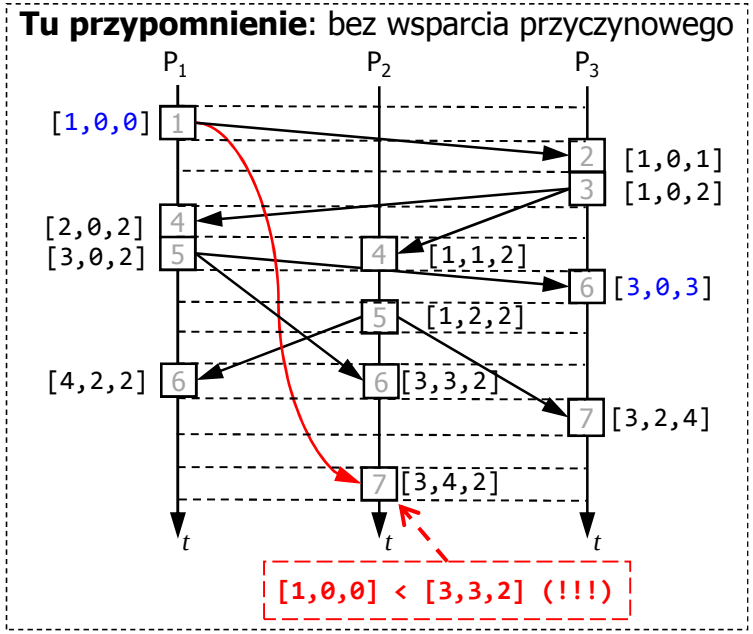
```
dostarcz(F,m):
  q := wysyłający(m)
  worek := worek ∪ {m}
  while ( istnieje m' ∈ worek takie że
          wysyłający(m') = q          and
          numer(m') = następna[q] )  do
        dostarcz(R, m')
        następna[q] := następna[q] + 1
        worek := worek – {m'}
```

Niezawodne rozsyłanie przyczynowe

- **Przyczynowe RB**
  - Rozgłaszanie z zachowaniem uporządkowania przyczynowego
    - Wszyscy członkowie grupy są adresatami wszystkich komunikatów
    - Stosowane są wektorowe zegary V_i w procesach P_i
    - Dostarczanie jest działaniem niepodzielnym
    - Procedura wsparcia przyczynowego dostarczania komunikatów z użyciem <u>zmodyfikowanego zegara wektorowego</u>
- ***Przypomnienie***:
  - Zegar wektorowy dla n procesów rozproszonych -
    - $V_i=[v_1,v_2,\ldots v_n]$, $i=1,\ldots,n$
      - › $V_i=[i]$ – licznik odpowiadający ostatniemu zdarzeniu w procesie P_i
      - › $V_i=[j]$, $j\neq i$ – wyobrażenie procesu P_i o zdarzeniach w procesach P_j
    - Do komunikatów m_l doczepiane wartości własnego zegara wektorowego $V(m_l)$
    - Przy odbiorze aktualizacja licznika własnego
      - › Scalanie regułą $\mathbf{max(V(m_l)[k]\ ,\ V_j[k])}$,gdzie $k=1,\ldots,n$

Rozsyłanie uporządkowane

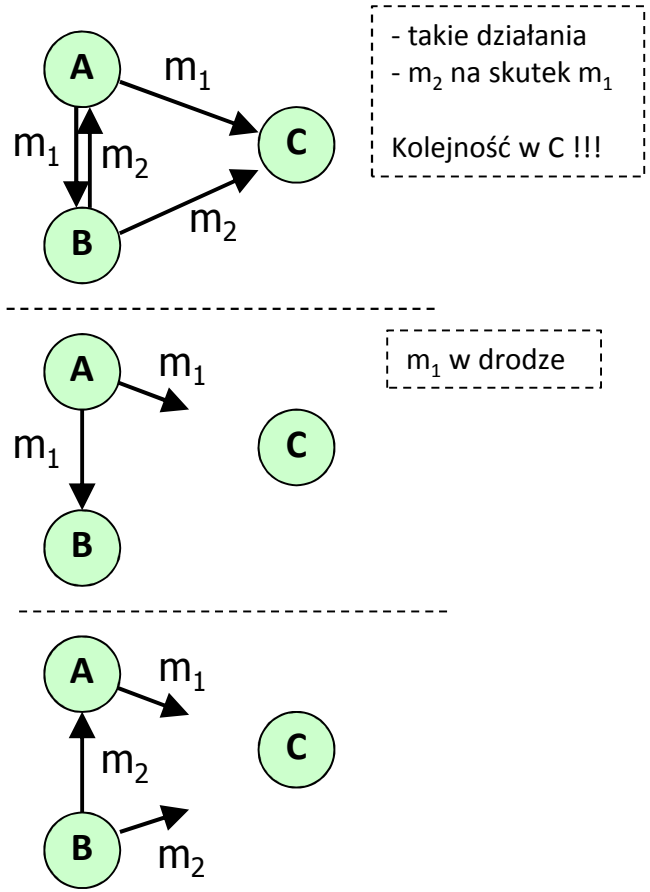
- **Wsparcie przyczynowego dostarczania komunikatów**
  - Zwiększanie własnego licznika $V_i[i]$ **tylko przy wysyłaniu komunikatu**
    - $V_i[i] = V_i[i] + 1$
  - Procedura opóźniania dostarczania wiadomości **od P_i do P_j** aż jest spełnione:
    - $V(m_l)[i] = V_j[i] + 1$
      - › Oznacza, że komunikat m_l od P_i jest tym, który miał przyjść następny
    - Dla każdego $k \neq i : V(m_l)[k] \leq V_j[k]$
      - › Oznacza, że: wszystkie komunikaty wcześniejsze od komunikatu m_l dostarczone do P_i powinny być wcześniej dostarczone również do P_j
      - › Lub inaczej: P_j dostał już wszystkie komunikaty (od innych procesów), które dostał P_i przed wysłaniem komunikatu m_l do P_j



  - Procedura sprawdzania kolejki komunikatów oczekujących na dostarczenia
    - Realizowana po każdym dostarczeniu nowego komunikatu

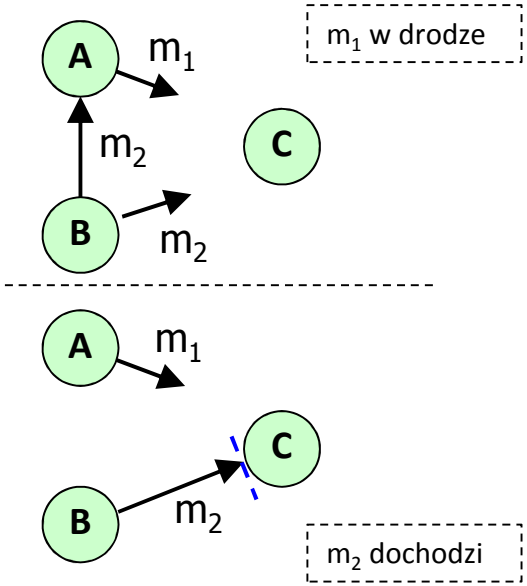
Rozsyłanie uporządkowane

- **Wsparcie przyczynowego dostarczania komunikatów – przykład**
  - Trzy procesy A, B, C
    - $V_A = V_B = V_C = [0,0,0]$
    - A wysyła m_1,
      B wysyła m_2 jako odpowiedź na m_1
  - A wysyła m_1
    - $V_A = [1,0,0]$ (*dlaczego?*)
    - $V(m_1)=[1,0,0]$ (*dlaczego?*)
  - B odbiera m1
    - $V_B = [0,0,0]$ oraz $V(m_1)=[1,0,0]$
    - *Jakie są warunki? Dostarczanie?*
    - $V_B = [1,0,0]$ (*dlaczego?*)
  - B wysyła m_2
    - $V_B = [1,1,0]$
    - $V(m_2)=[1,1,0]$



Rozsyłanie uporządkowane

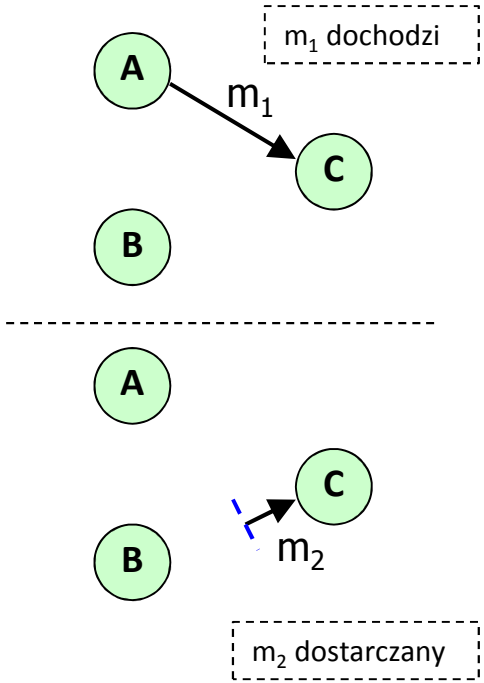
- **Wsparcie przyczynowego dostarczania komunikatów – przykład**
  - A odbiera m_2
    - $V_A = [1,0,0]$ oraz $v(m_2)=[1,1,0]$
    - *Jakie są warunki? Dostarczanie?*
    - $V_A = [1,1,0]$ (*dlaczego?*)
  - C odbiera m2
    - $V_C = [0,0,0]$ oraz $v(m_2)=[1,1,0]$
    - *Jakie są warunki? Dostarczanie?*
    - Odroczenie dostarczenia m_2 do C (*dlaczego?*)



    - Interpretacja:
      - › m_2 został wysłany po wystąpieniu m_1, w którym mogło być coś istotnego dla m_2
      - › m_2 powinien być dostarczony dopiero po m_1

Rozsyłanie uporządkowane

- **Wsparcie przyczynowego dostarczania komunikatów – przykład**
  - C odbiera m_1
    - $V_C = [0,0,0]$ oraz $v(m_1)=[1,0,0]$
    - Jakie są warunki? Dostarczanie?
    - $V_C = [1,0,0]$ (*dlaczego?*)
  - Sprawdzenie kolejki komunikatów w C
    - Jest m_2
    - $V_C = [1,0,0]$ oraz $v(m_2)=[1,1,0]$
    - Jakie są warunki? Dostarczanie?
    - $V_C = [1,1,0]$ (*dlaczego?*)
  - Po operacjach wartości wektorów:
    - $V_A = V_B = V_C = [1,1,0]$



Rozproszone systemy informatyczne

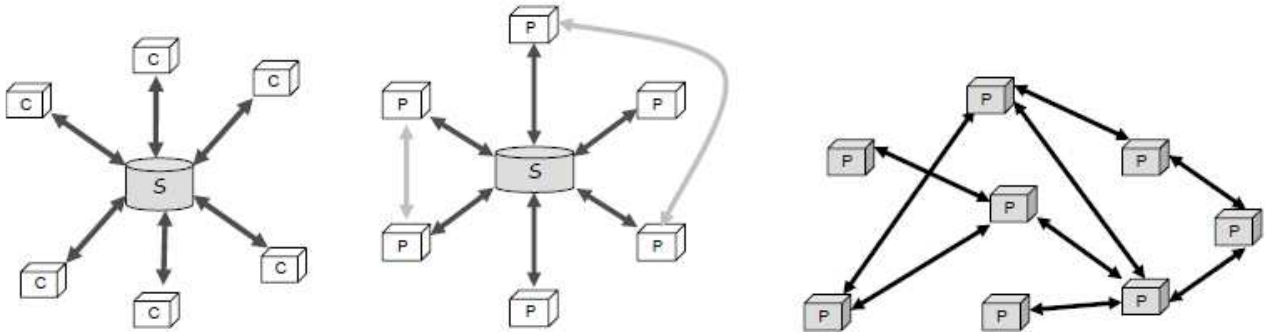
dr inż. Mariusz Fraś

Sieci Peer-to-peer (P2P)

- Niestrukturalne P2P
- Strukturalne P2P
  (Chord, CAN)

Architektura

- **Podstawowe typy systemów sieciowych**
  - Client-server
  - P2P
  - Hybrid



- **Wyróżniające cechy P2P**
  - Decentralizacja (użycia zasobów)
  - Skalowalność
  - Samoorganizacja
  - Odporność na awarie
  - Anonimowość

P2P

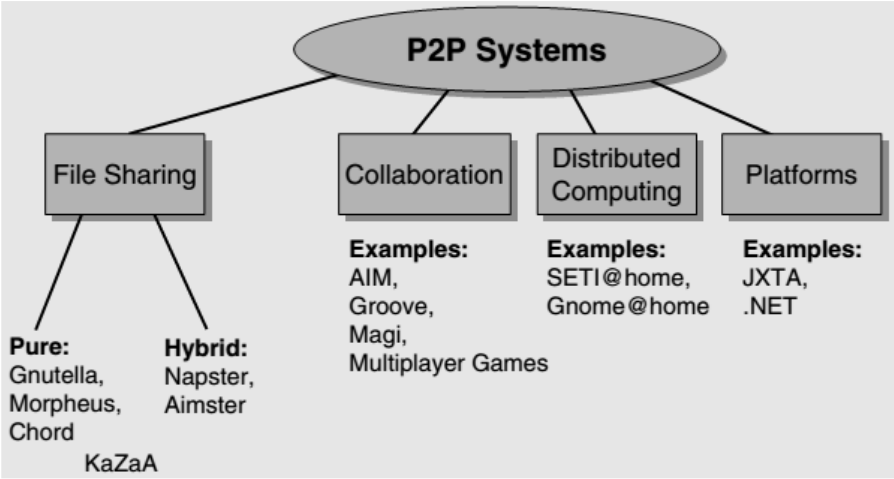
- **Sieci nakładkowe (ang. *Overlay networks*)**
  - Sieci nakładki odnoszą się do sieci, które są zbudowane "na wierzchu" innej sieci
- **System równy-z-równym (ang. *peer-to-peer*) – P2P**
  - Samoorganizujący się system równorzędnych i autonomicznych podmiotów (***peers***), którego celem jest wspólne wykorzystywanie rozproszonych zasobów w środowisku sieciowym unikając usług centralnych
    - System ze zdecentralizowaną samoorganizacją i współdzieleniem zasobów
  - P2P odnosi do aplikacji, które korzystają z zasobów (pamięć, treści, obecności człowieka, ...) dostępnych w systemach końcowych Internetu
- **Sieć nakładkowa P2P**
  - Sieć konstruowana przez równorzędne węzły (peers) w Internecie w warstwie aplikacji na wierzchu sieci IP

P2P

- **Podstawowe cechy modelu komunikacji P2P**
  - **Równoważność**
    - Każdy węzeł pełni taką samą rolę w sieci
      - Każdy komputer dysponuje podobnymi możliwościami oraz może inicjować połączenia
      - Nie ma ustalonej hierarchii ani centralnego serwera (w ogólności)
      - Ten sam komputer może równocześnie pełnić rolę serwera i klienta, czyli pobierać dane z innych komputerów i udostępniać swoje zasoby wszystkim pozostałym komputerom
  - **Autonomiczność**
    - Węzły nie są od siebie zależne, mogą w dowolnej chwili dołączyć się/odłączyć się od sieci
  - **Słabe powiązanie**
    - Komunikują się przez sieć ogólnego użytku za pomocą powszechnie stosowanych mechanizmów

P2P

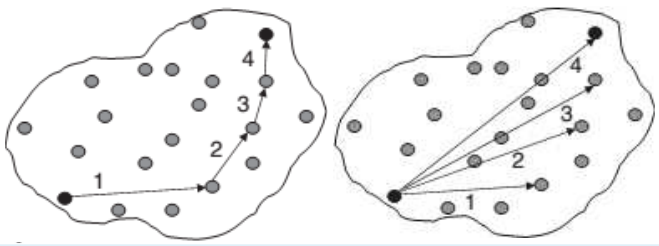
- **Kategorie aplikacji P2P**
  - Wymiana plików
  - Komunikacja (Chat/IRC, AOL, ICQ, Jabber, Skype)
  - Współdziałanie
    - Wspólne przetwarzanie
    - Dzielenie zasobów oblicz.
  - Platforma
    - P2P jako wsparcie (platforma wsparcia) dla współdziałania różnych komponentów systemu, na różnych zasadach, dzięki bezpośredniej wymianie informacji



- Celem sieci P2P jest zwykle współużytkowanie zasobów danych, w przeciwieństwie do przetwarzania siatkowego (systemów GRID), gdzie zwykle współdzielone są (w skoordynowany sposób) zasoby obliczeniowe

Zagadnienia konstrukcji systemów P2P

- **Perspektywa teorii grafów** (*Graph-theoretic perspective*)
  - Sieć nakładkowa jako skierowany graf G(V,E)
    - Węzły
      - Unikalny identyfikator PID i adres sieciowy NID
    - **Przyległość** (ang. *adjacency*) peer'ów p i q
      - Krawędź (p,q) oznacza możliwość bezpośredniego wysłania wiadomości przez sieć używając NID
    - **Sąsiedztwo** p i q
      - Peery p i q są **sąsiadami** jeśli PID_p poprzedza PID_q lub na odwrót
    - G musi być spójny
    - G jest dynamiczny
      - Trzeba wymieniać informację o G w przypadku od/do-łączania się peerów
  - Ścieżka trasowania (*Routing path*)
    - Sekwencja krawędzi lub przeskoków od peera wysyłającego wiadomość do celu
    - Rodzaje
      - Iteracyjna (*Iterative*)
      - Rekursywna (*Recursive*)

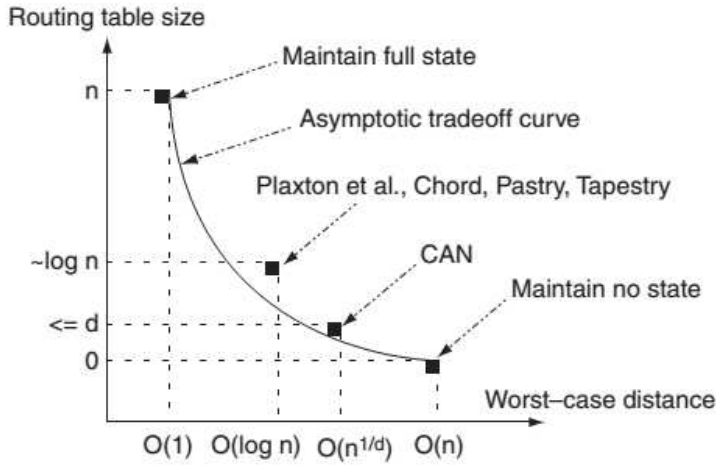


Zagadnienia konstrukcji systemów P2P

- **Perspektywa teorii grafów** (*Graph-theoretic perspective*)
  - Podstawowe parametry
    - Średnica (Diameter)
    - Stopień węzła (Degree)

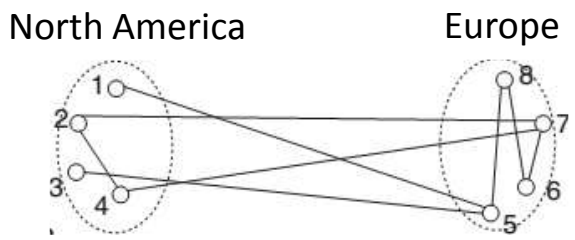
Graph	Degree	Diameter
de Bruijin	k	$\log_k N$
Trie	$k+1$	$2 \log_k N$
Chord	$\log_2 N$	$\log_2 N$
CAN	$2d$	$\frac{1}{2} dN\ 1/d$
Pastry	$(b-1) \log_b N$	$\log_b N$
Classic butterfly	k	$2 \log_k N(1-o(1))$

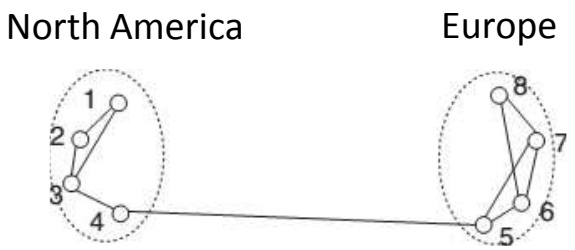
- **Perspektywa wydajności trasowania / wydajności wyszukiwania** (*Routing performance perspective*)
  - Zależność pomiędzy rozmiarem informacji w węźle i długością ścieżki wyszukiwania
    - Skrajności
      - Pełna informacja w tablicy
      - Tylko informacja o następcy



Zagadnienia konstrukcji systemów P2P

- **Perspektywa wydajności trasowania** (*Routing performance perspective*)
  - Wydajność zależy od relacji pomiędzy topologią warstwy nakładkowej (***overlay***) i warstwy sieci (***underlay***)
    - Randomly connected overlay
      - › Przyległość peerów konstruowana bez uwzględniania underlay
      - › Każdy przeskok w sieci najkładkowej to wiele przeskoków w underlay
    - Locality-aware overlay
      - › Większa efektywność
      - › Wymaga zbierania informacji o dystansach w sieci
      - › Może zmniejszyć niezawodność w przypadku mechanizmów wykorzystujących sąsiedztwo



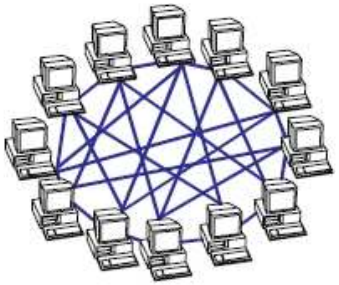


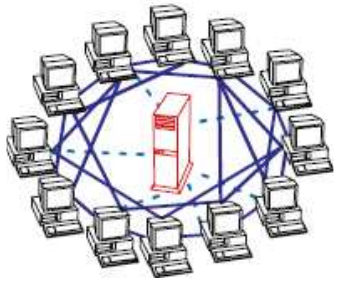
Architektura

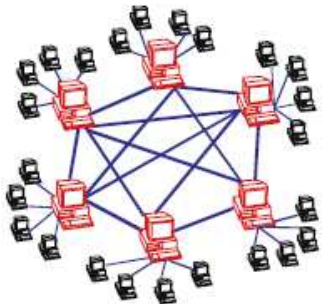
- **Architektury sieci nakładkowych P2P**
  - ***Unstructured***
    - Sieć zorganizowana w postaci przypadkowego grafu (*random graph*) płaskiego lub hierarchicznego
  - ***Structured***
    - Oparta na DHT (DHT – ang. *Distributed Hash Table*)
      - W sieci przypisane są klucze do elementów danych (do zasobów) i organizuje się peery w graf który mapuje każdy klucz zasobu do peera
      - Typowo dane mogą być uzyskane z sieci ze złożonością O(log N)
  - ***Overlay Mutlticast networks***
    - Węzły (*Peers*) zorganizowane w strukturze drzewa multicastowego

Architektura

- **Unstructured**
  - **Pure**
    - Bez centralnej koordynacji
    - Połączenia nawiązywane losowo
    - Gnutella 0.4, FreeNet
  - **Centralized directory based**
    - Centralna usługa indeksowania
    - Połączenia nawiązywane na żądanie zasobu
    - Napster, Bittorrent
  - **Hybrid**
    - Dodatkowa warstwa węzłów specjalnych
    - KaZaA (FastTrack), Gnutella 0.6

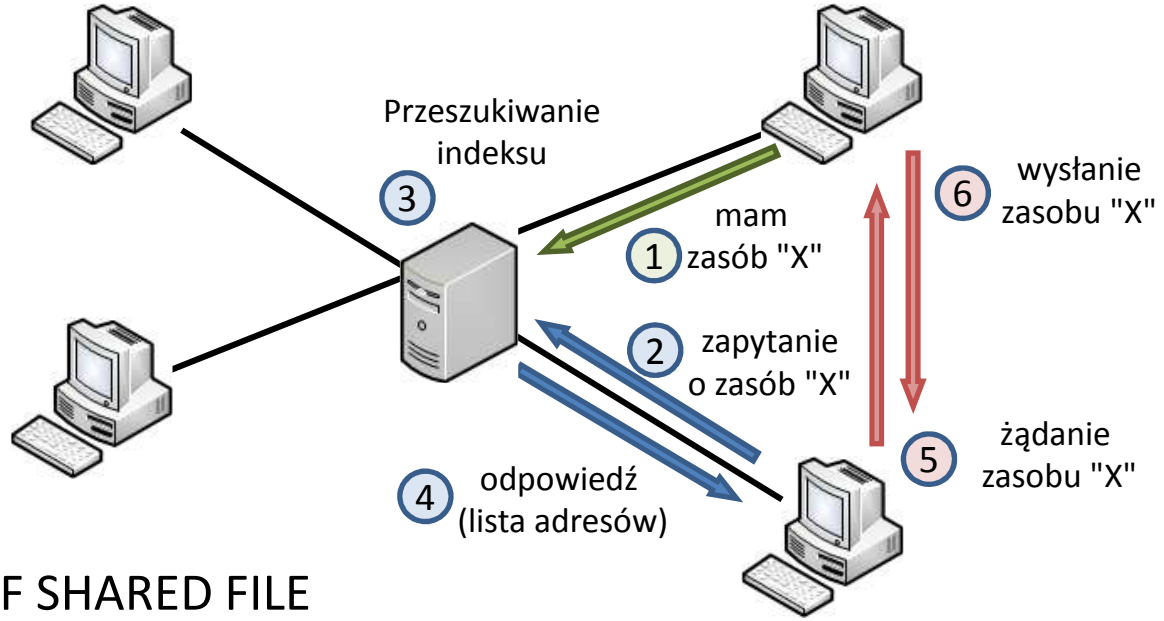






- **Structured**
  - DHT-based
    - Chord, Tapestry, CAN, Kademlia, …

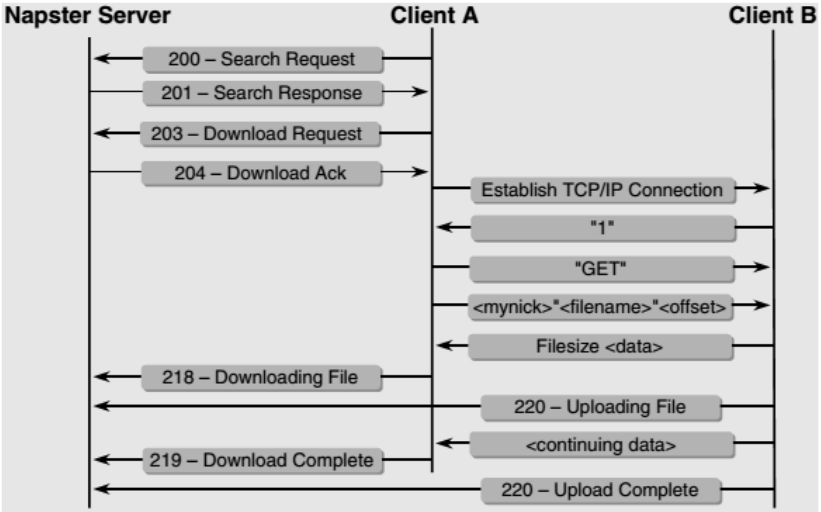
- **Zcentralizowane**
  - Przykład:
    Napster



  - Komunikaty
    - CLIENT NOTIFICATION OF SHARED FILE
      - › Nazwa pliku i MD5-hash value
        100-Client notification of shared file-"<filename>"<md5><size><bitrate><frequency><time>
    - SEARCH
      - › Zawiera nazwę zasobu i różne parametry filtrujące

Sieci niestrukturalne

- **Zcentralizowane**
  - Sesja protokołu Napster (przykład)
    - Komunikaty:
      - **200—Search request—[FILENAME CONTAINS "artist name"] MAX_RESULTS <max> [FILENAME CONTAINS "song"] [LINESPEED <compare> <link-type>] [BITRATE <compare> "<br>"] [FREQ <compare> "<freq>"] [WMA-FILE] [LOCAL_ONLY]**
      - **201—Search response—"<filename>" <md5> <size> <bitrate> <frequency> <length> <nick> <ip><link-type> [weight]**
      - **203—Download request—<nick>"<filename>"**
      - **204—Download ack—<nick><ip> <port> "<filename>"<md5> <linespeed>**
      - **...**

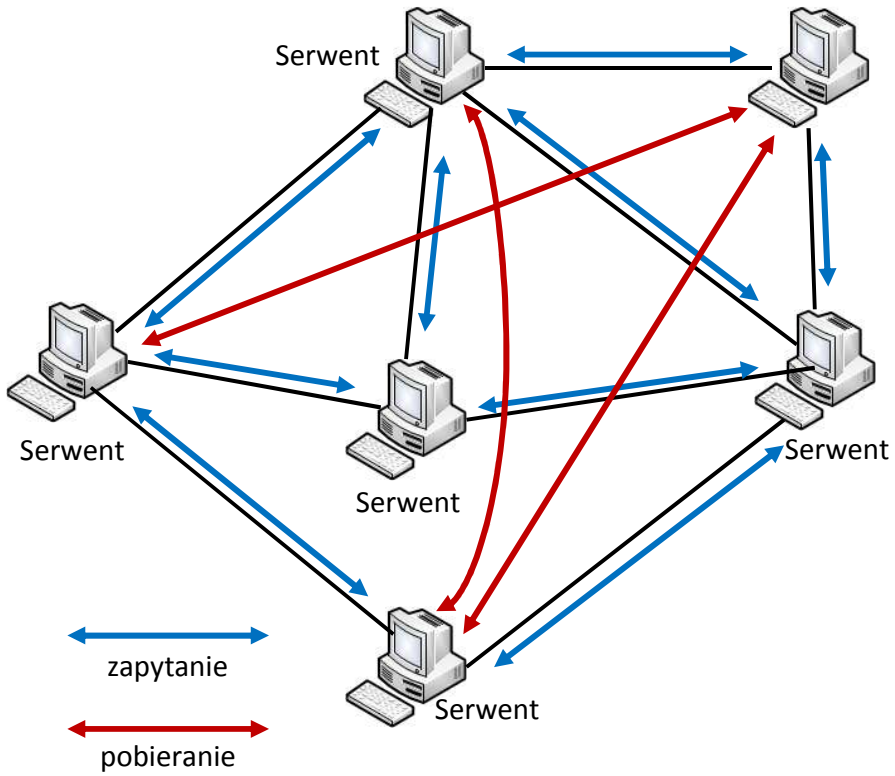


Sieci niestrukturalne

- **Czyste (*Pure*)**
  - Przykład: Gnutella 0.4
    - 4 wiadomość

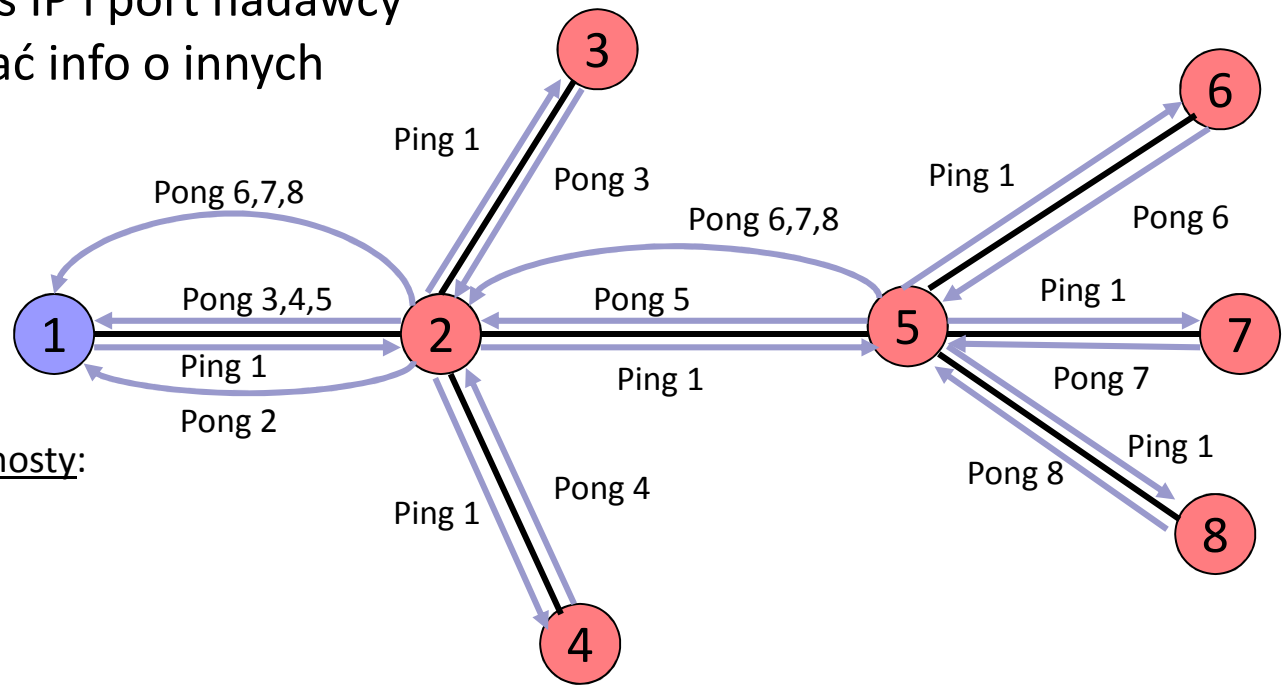
      Do odkrywania sieci:
      - › PING
      - › PONG

      Do uzyskiwania zasobów:
      - › QUERY
      - › QUERYHIT



Sieci niestrukturalne

- **Odkrywanie sieci**
  - Przykład procedury w Gnutella – podłączanie do sieci
  - Ping – zgłoszenie nowego węzła w sieci – odkrywanie sieci
  - Pong – odp. na Ping,
    - Zawiera adres IP i port nadawcy
    - Może zawierać info o innych peerach



<u>Znane hosty</u>:

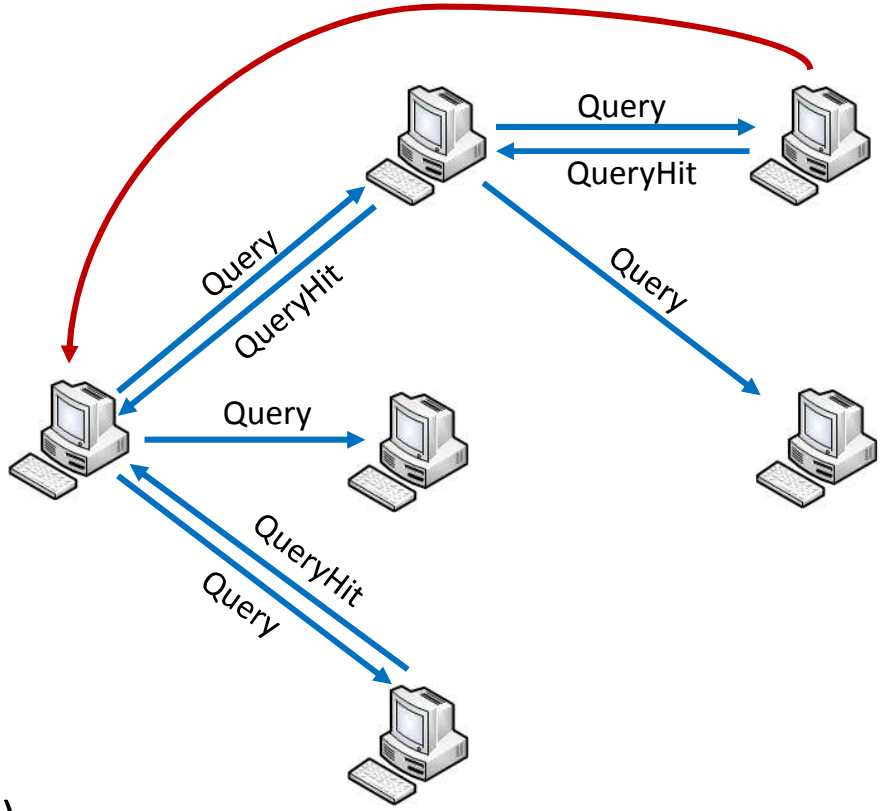
1) 2

2) 3,4,5

3) 6,7,8

Sieci niestrukturalne

- **Wyszukiwanie zasobów**
  - Query – zapytanie o zasób
    - Podstawowy typ wiadomości
  - Rodzaje
    - Flooding (Zalewanie)
    - Random Walk
      (losowy spacer)
    - Random BSF
      (losowe przeszukiwanie wszerz)
    - Hill climbing
      (algortm wspinaczki na szczyt)
    - Preference directed queries
      (oparty o podobieństwo zainteresowań)



Sieci niestrukturalne

- **Ruch w sieci Gnutella**
  - Liczba wiadomości:

$$L = 2 \cdot \sum_{i=0}^{TTL} C \cdot (C-1)^i$$

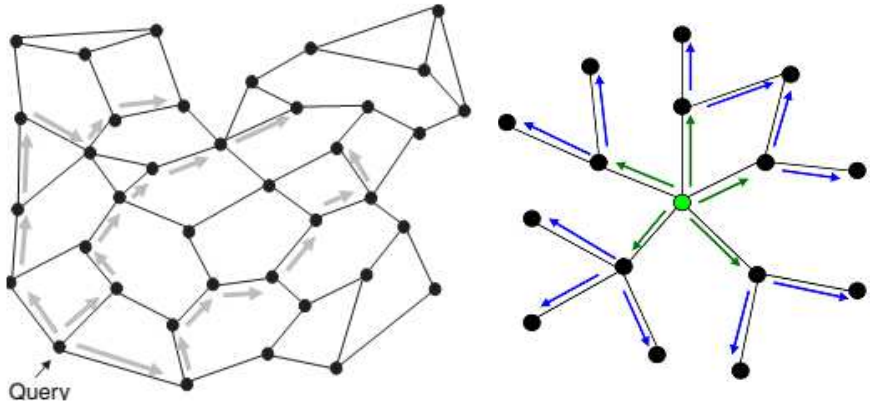
    - C- śr. liczba poł. węzłów
  - Dla: TTL=7 i C=4:

    L= 26470
  - 2002r – 5 godz. eksp. Badawczy (Univ. Of California)
    - 56 mln pozycji
    - 63% Ping + Pong
    - 33% Qery,
    - 3% QueryHit
    - 66% ruchu przez tzw. "Freeriders"
    - 37% odpowiedzi tylko przez 1% hostów

Sieci niestrukturalne

- **Wyszukiwanie zasobów**
  - *Query flooding*
    - Ograniczany do maksymalnej liczby przeskoków (n hops)
      - Jako czas TTL
      - Określany przez nadawcę
  - *Expanding ring / Iterative deepening*
    - Wyszukiwanie zalewowe ze stopniowo powiększanym TTL

```
FloodForward(Query q, Source p)
{
  // czy było już to zapytanie ?
  if(q.id ∈ oldIdsQ) return   // tak → odrzuć
  oldIdsQ = oldIdsQ ∪ q.id    // zapamiętaj zapyt.
  // czy czas przeterminowania minął?
  q.TTL = q.TTL – 1
  if q.TTL ≤ 0 then return    // tak → odrzuć
  // nie → przekaż dalej do pozostałych sąsiadów
  foreach(s ∈ Neighbors) if (s ≠ p) send(s,q)
}
```

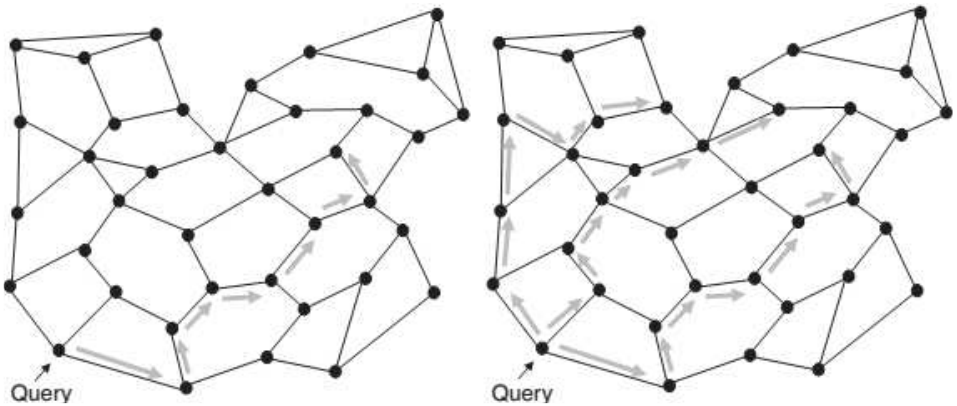


Sieci niestrukturalne

- **Wyszukiwanie zasobów**
  - Random walk

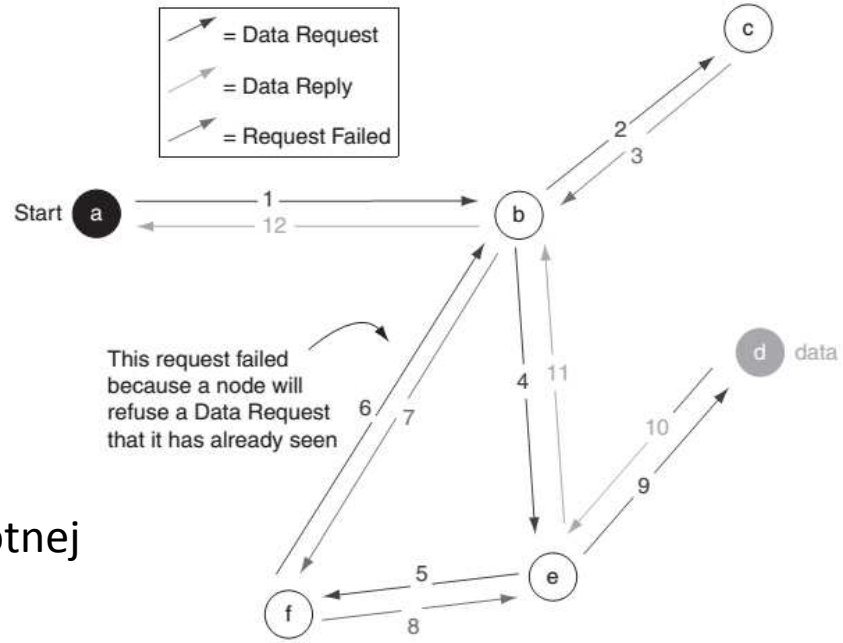
```
RandomWalk(source, query, TTL) {
  if (TTL > 0) {
    TTL = TTL – 1
    // wybierz następnego peera, nie wysyłaj ścieżką powrotną
    while((next_hop = neighbors[random()]) == source)
      send(source, query,TTL)
} }
```

  - K-way parallel random walk



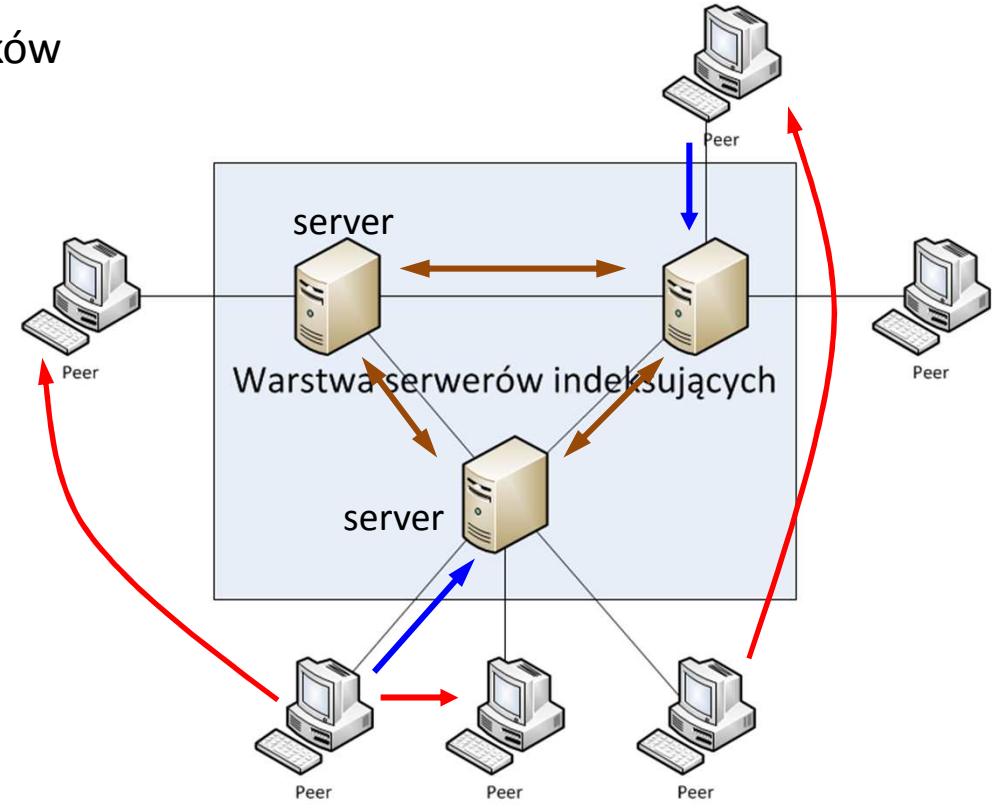
Sieci niestrukturalne

- **Czyste (*Pure*)**
  - Przykład **FreeNet**
    - Zasoby i peer'y mają identyfikatory
      - › Tworzone funkcją haszującą (skrótu) SHA
    - Każdy peer ma tablicę routingu o stałym rozmiarze
      - › Id peera (tzw. routing key) w tablicy
    - Rutowanie oparte na kluczach
      - › Żądanie jest kierowane do peera o najbardziej pasującym kluczu
      - › Jeśli nie powiedzie się to do następnego
      - › Algorytm wspinaczkowy z powrotami
      - › Do momentu wyczerpania TTL
    - Dla zwiększenia prawdopodobieństwa dostępu:
      - › keszowanie obiektów na ścieżce powrotnej



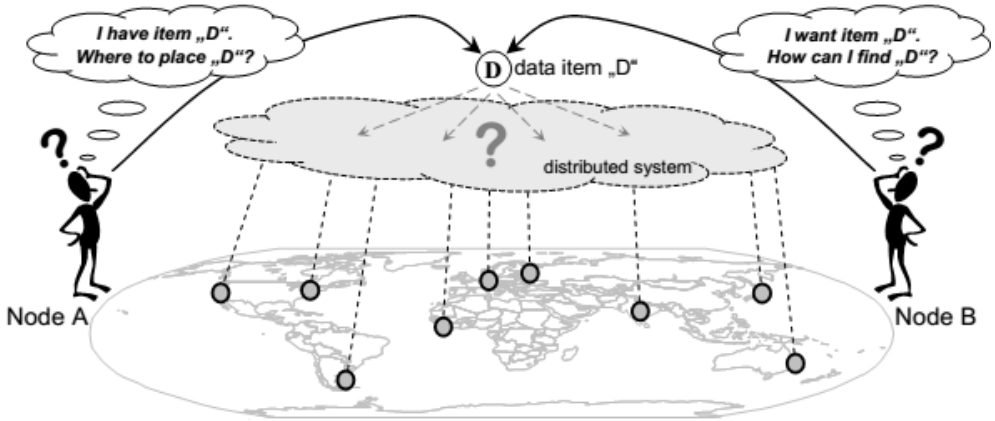
Sieci niestrukturalne

- **Hybrydowe**
  - Przykład **eDonkey**
    - Serwer
      - Indeksowanie lokalizacji plików
      - Logowanie do systemu
      - Informowanie o lokalizacji
    - Klienty
      - Wymieniają pliki bezpośrednio między sobą

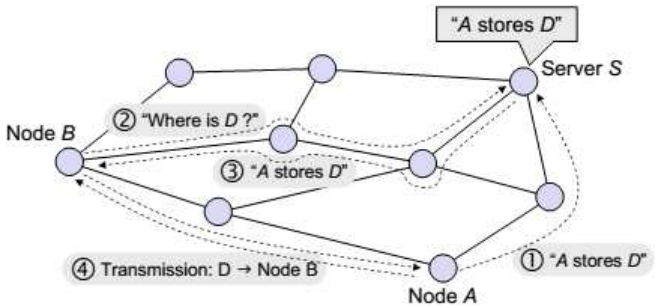


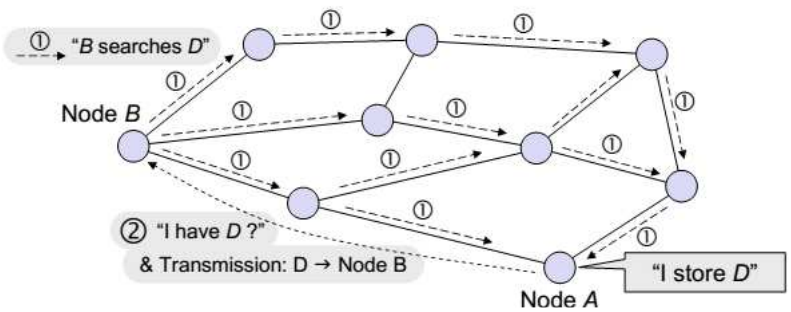
Problem lokalizacji zasobu

- **Problem lokalizacji zasobu**



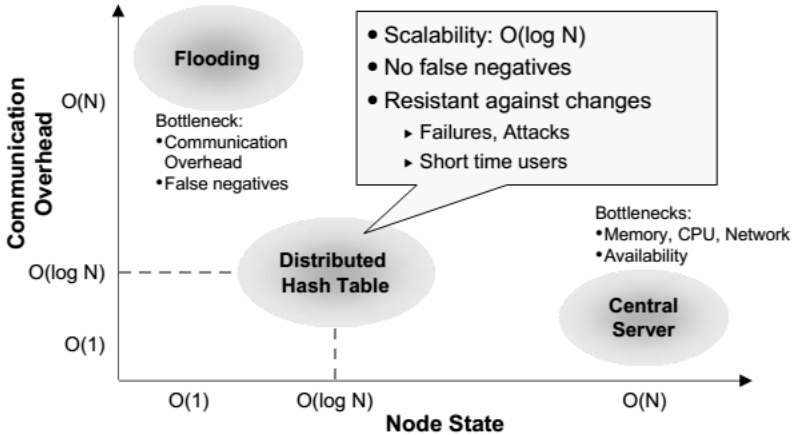
- Wyszukiwanie w sieciach niestrukturalnych – np.:
  - Serwer centralny
  - Przeszukiwanie przez zalewanie

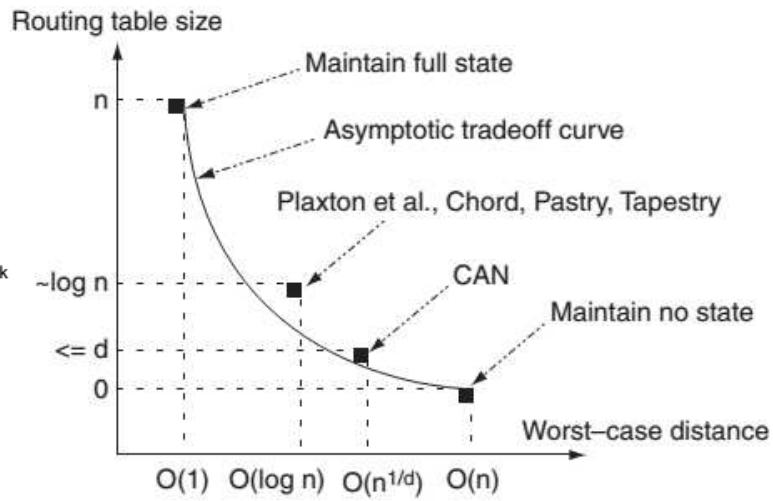




Problem lokalizacji zasobu

- **Koszt komunikacji**



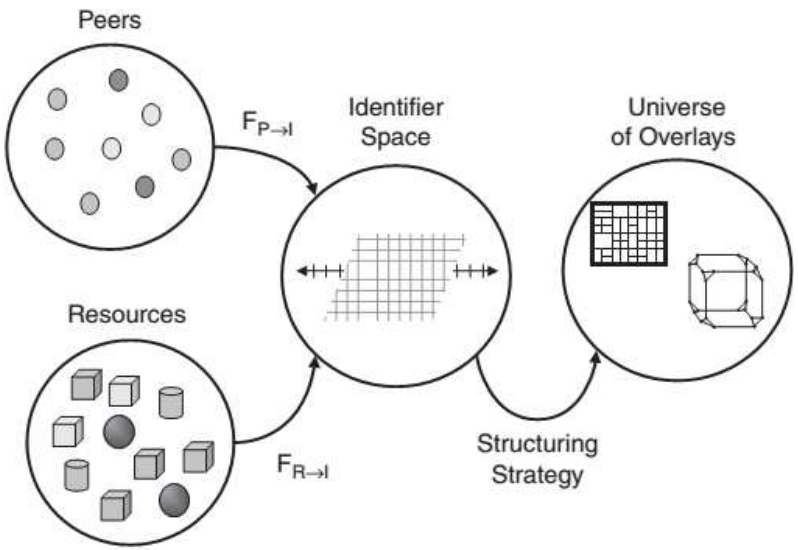


- Porównanie

System	Koszt w węźle	Koszt komunikacji
Serwer centralny	O(N)	O(1)
Zalewanie	O(1)	≥ O(N^2)
DHT	O(log N)	O(log N)

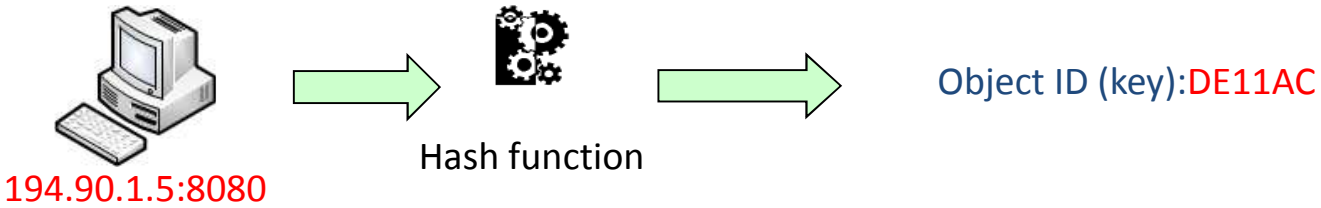
Zagadnienia konstrukcji systemów P2P

- **Perspektywa projektowa/modelu** (*a design space perspective*)
  - Przestrzeń identyfikacji
    - Są w niej odwzorowywane ID-y wszystkich węzłów i zasobów
      - › Węzły: muszą mieć unikalne id.
      - › Ważne dla ich lokalizowania
      - › Zasoby: dla ich odkrywania
      - › Typowo: losowe odwz. funkcją haszującą
    - Własności:
      - › rozmiar, relacja porządkowania, relacja odległości, przezroczystość lokalizacji
      - › wsparcie dla metryki odległości
    - Powinna być skalowalna
  - Sieć nakładkowa
    - Określona jest **geometria** – statyczna struktura grafu
    - Wiele rozwiązań biorących pod uwagę:
      - › Średnica grafu
      - › Dystrybuanta stopni węzłów
      - › Spójność (połączenia) grafu
  - Strategia ustrukturyzowania
    - Integruje geometrię modelu, zasady trasowania i utrzymywania sieci

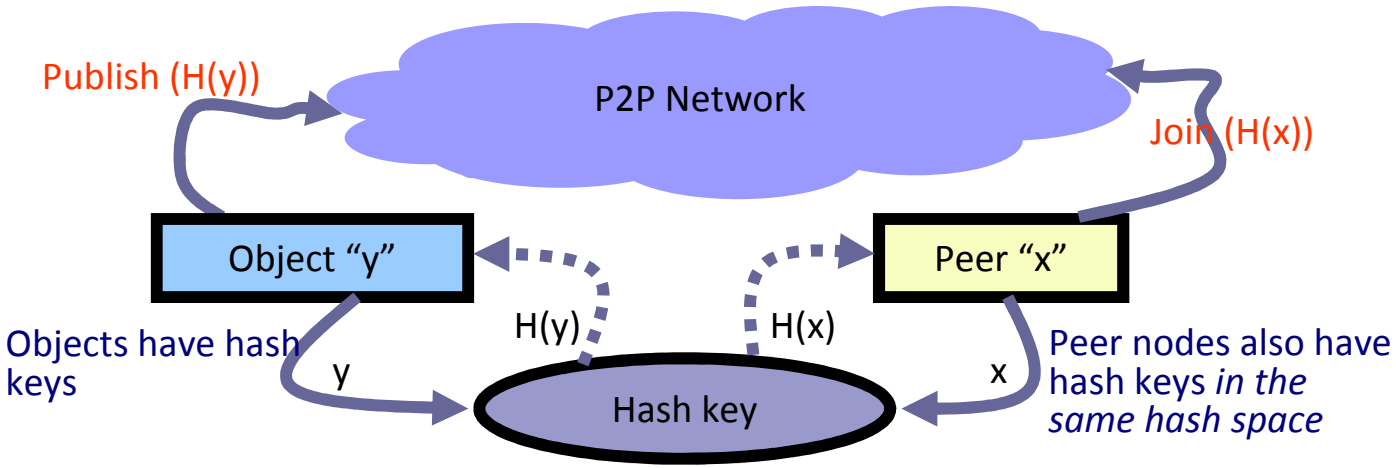


Sieci strukturalne

- **Idea**
  - Nadawanie identyfikatorów (**kluczy**) funkcją haszującą

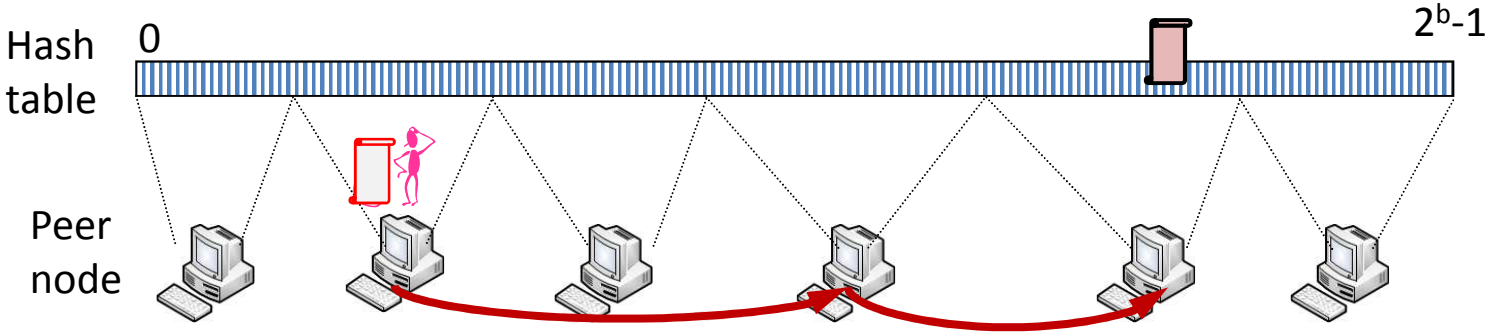


  - Klucze mają peery i zasoby (obiekty)



Sieci strukturalne

- **Idea**
  - Dystrybucja obiektów (ciągłego przedziału wartości kluczy) do wszystkich peerów odpowiedzialnych za dany zakres



  - Każdy peer musi "coś wiedzieć" przynajmniej o jednej kopii każdego obiektu z jego zakresu (o kluczu z jego zakresu)
    - Peer przechowuje obiekt
    - lub: Peer przechowuje wskaźnik do lokalizacji obiektu
  - Zapewnienie odpowiedniego schematu trasowania
    - Dla każdego obiektu, peer który za niego odpowiada musi być osiągalny "**krótką ścieżką**"

Sieci strukturalne

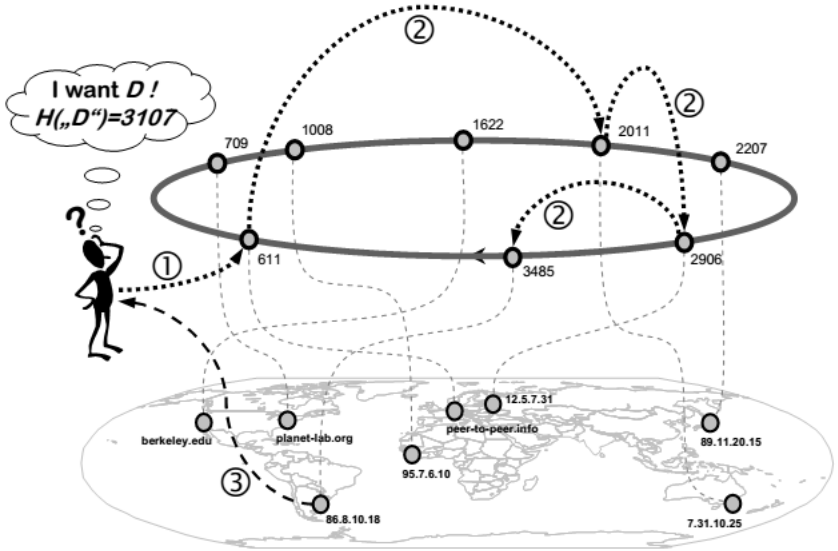
- **Idea**
  - Liczba sąsiadów powinna się dobrze skalować ze wzrostem liczby peerów
  - Złożoność wyszukiwania i składowania powinna być lepsza niż O(N) – np. rzędu O(logN)
  - Mechanizm powinien być w pełni rozproszony
    - Brak centralnych elementów systemu
  - Wsparcie dla dołączania/opuszczania sieci nakładkowej
    - Rozpoznawanie sąsiadów
    - Powtórny rozdział przestrzeni kluczy
- **API**
  - Każdy węzeł (peer) zapewnia funkcje:
    - Dostarcza klucz zasobu
    - Zwraca IP węzła odpowiedzialnego za dany klucz

Sieci strukturalne

- **DHT = Distributed Hash Table**
  - Rozproszone indeksowanie w formie DHT
    - Identyfikacja węzłów (peer'ów) za pomocą GUID
    - Odwzorowanie węzłów i danych w jednolitą przestrzeń adresową
    - Lokalizacja peer'a (dystrybucja zapytań) za pomocą tablicy routingu
  - GUID
    - Jest najczęściej ciągiem bitów najczęściej 64, 128 lub 160 zn.
    - Funkcja haszująca – np. SHA
    - Metody przyznawania GUID:
      - Peer sam tworzy GUID
      - GUID zostaje przydzielany peer'owi w momencie dołączenia do istniejącej sieci
  - Wymaganie gwarancji znalezienia zasobu umieszczonego w systemie
  - Równomierna dystrybucja zasobów

Sieci strukturalne

- **Trasowanie**
  - Trasowanie w oparciu o klucze
    - Każdy z węzłów ma tablicę rutingu zawierającą pewien podzbiór peerów podłącznych do sieci
    - Wpis do tablicy składa się z:
      - GUID
      - Adres (IP, port)



  - Organizacja tablic routingu jest różna dla różnych algorytmów sieci P2P
    - Rózne rozwiązania (np. Chord, Pastry, CAN,…) różnią się tylko podejściem do trasowania
    - Funkcja NextHop jest różna dla różnych rozwiązań

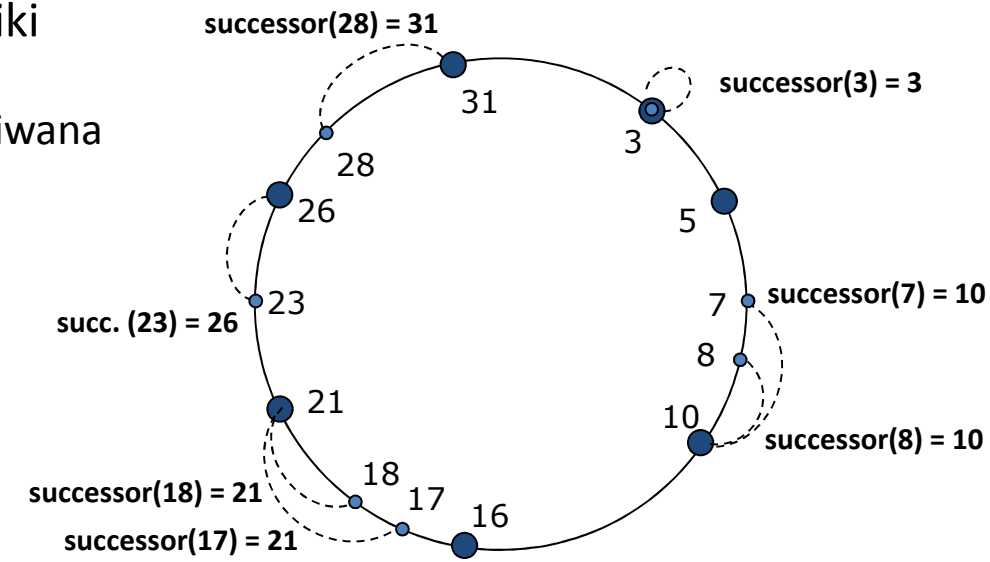
Sieci strukturalne

- **Typy sieci**
  - Przykładowe charakterystyki:
    - Maksymalna liczba przeskoków zapytania w sieci
    - Organizacja przestrzeni adresowej
    - Kryterium wybo-ru następnego przeskoku
    - Geometria sieci nakładkowej
    - Utrzymywanie sieci nakładkowej
    - *Locality and topology awareness*

Type	Systems	Applications	Implementations
Logarithmic mesh with prefix routing	PRR, S-PRR		
	Tapestry	Oceanstore, Bayeaux	Chimera, OverlayWeaver
	Pastry	PAST, Scribe	FreePastry,
	P-Grid	UniStore, PIX-Grid	OverlayWeaver
	Bamboo		P-Grid
	Z-Ring		OpenDHT
Ring with embedded logarithmic degree mesh	Chord	DHash, CFS	Chord, OverlayWeaver
	DKS		DKS
	Chord#		
Constant degree	Ulysses		
	Koorde		OverlayWeaver
	Cycloid		
Logarithmic degree with specialized distance metrics	CAN		
	Kademlia		KAD, OverlayWeaver
O(1)-hop	Kelips		
	OneHop		
	EpiChord		
	D1HT		

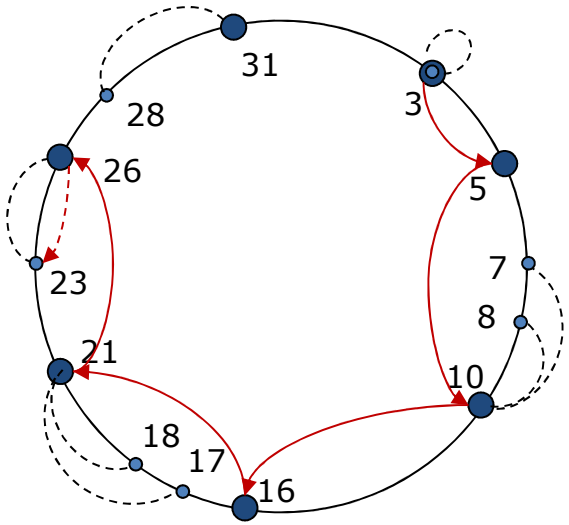
Chord

- **Pierścień z wbudowaną siatką logarytmiczną (ang. *Ring with embedded logarithmic degree mesh*)**
  - Każdy element danych i węzeł posiada deterministycznie przypisany identyfikator
    - **Identyfikator węzła ID** – unikalny m-bitowa wartość (np. hash adresu IP)
    - **Klucz *k*** (ang. key) – m-bitowy identyfikator elementu danych (np. hash sekwencji bajtów)
  - Węzły są uporządkowane w formie okręgu o liczbie $N=2^m$
  - Węzły zawierają wskaźniki do lokalizacji zasobu
    - Para (*k, value*) jest obsługiwana przez węzeł, którego ID jest większy lub równy *k*
    - Taki węzeł jest nazywany **następcą / sukcesorem** (ang. ***successor***) klucza *k*



Chord

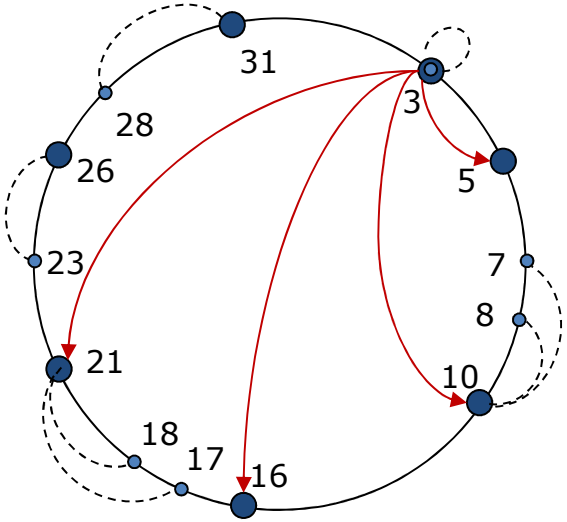
- **Trasowanie**
  - Węzeł w okręgu Chord (okręgu ze zwiększającymi się identyfikatorami), jest odpowiedzialny za wszystkie klucze, które poprzedzają go
  - Wyszukiwanie polega na obejściu okręgu po wskaźnikach, aż do zlokalizowania zasobu
    - Dla dużej przestrzeni zasobów może to być proces o bardzo wielu etapach
    - Proste wyszukiwanie może być nieefektywne
  - Aby przyspieszyć lokalizację zasobu dodatkowe informacje o trasie są przechowywane w "***finger tables***"



Chord

- **Trasowanie**
  - Tablice trasowania w chord - "***Finger tables***"
    - Każdy węzeł ma tablicę m rekordów – O(log N)
    - **i-ty "odcisk"** w węźle n wskazuje na **seccessor(n+ 2^i)** – węzeł który jest następnikiem odległym od n o minimum 2^i
      - › Na węzeł który odpowiada za zasoby o kluczach większych od $n+2^i$ (do $n+2^{i+1}$)
      - › **Przedział** (interwał) **int** wartości dla rekordu wynosi: **$[n+2^i, n+2^{i+1})$**
    - Wartości oblicza się **modulo 2^m** (!)
  - Tablica dla węzła n=3

i	ID	int	succ
0	$3+2^0=$**4**	[4,5)	5
1	$3+2^1=$**5**	[5,7)	5
2	$3+2^2=$**7**	[7,11)	10
3	$3+2^3=$**11**	[11,19)	16
4	$3+2^4=$ **19**	[19,3)	21

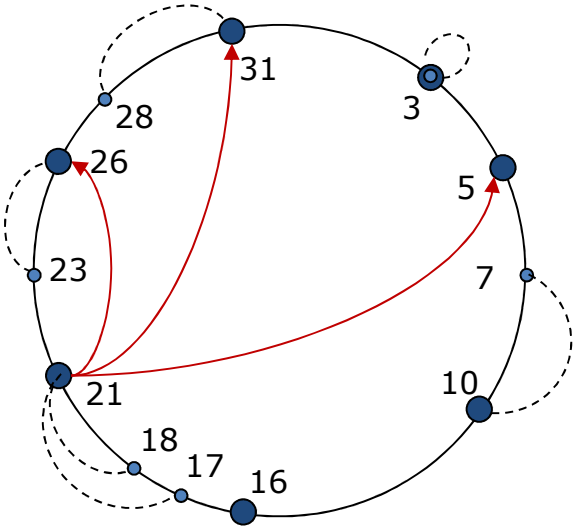


Chord

- **Trasowanie**
  - Tablica dla węzła n=21

i	ID	int	succ
0	$21+2^0=$**22**	[22,23)	26
1	$21+2^1=$**23**	[23,25)	26
2	$21+2^2=$**25**	[25,29)	26
3	$21+2^3=$**29**	[29,5)	31
4	$21+2^4=$ **5**	[5,21)	5

bo: $21+2^4 = 37$
37 modulo 32 = 5



- Trasowanie polega na wyszukaniu węzła (successor danego klucza) w tablicy do którego przekazywane jest zapytanie o zasób
  - Maksymalnie O(log N) hopów (komunikatów)

Chord

- **Lookup(…)**

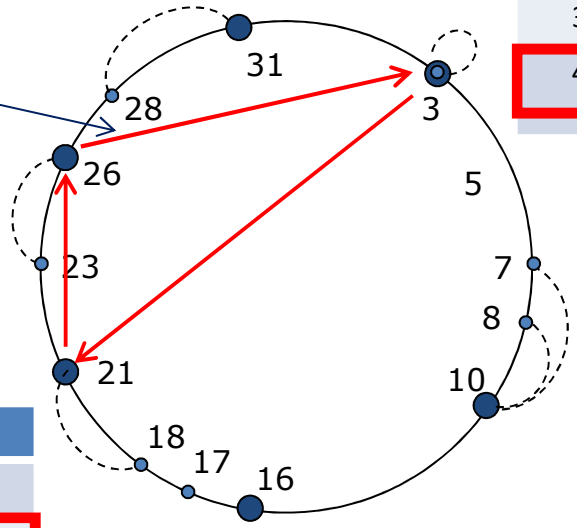
Jestem węzeł 3.
Znajdź klucz 23 - Lookup(23)

1

i	ID	int	succ
0	$3+2^0=4$	[4,5)	5
1	$3+2^1=5$	[5,7)	5
2	$3+2^2=7$	[7,11)	10
3	$3+2^3=11$	[11,19)	16
4	$3+2^4=19$	[19,3)	21

3 odpowiedź na zapytanie

adres lub zasób dla key=23



23 > 19
23 ∈ [19,3)

Next Hop = 21

2

i	ID	int	succ
0	$21+2^0=22$	[22,23)	26
1	$21+2^1=23$	[23,25)	26
2	$21+2^2=25$	[25,29)	26
3	$21+2^3=29$	[29,5)	31
4	$21+2^4= 5$	[5,21)	5

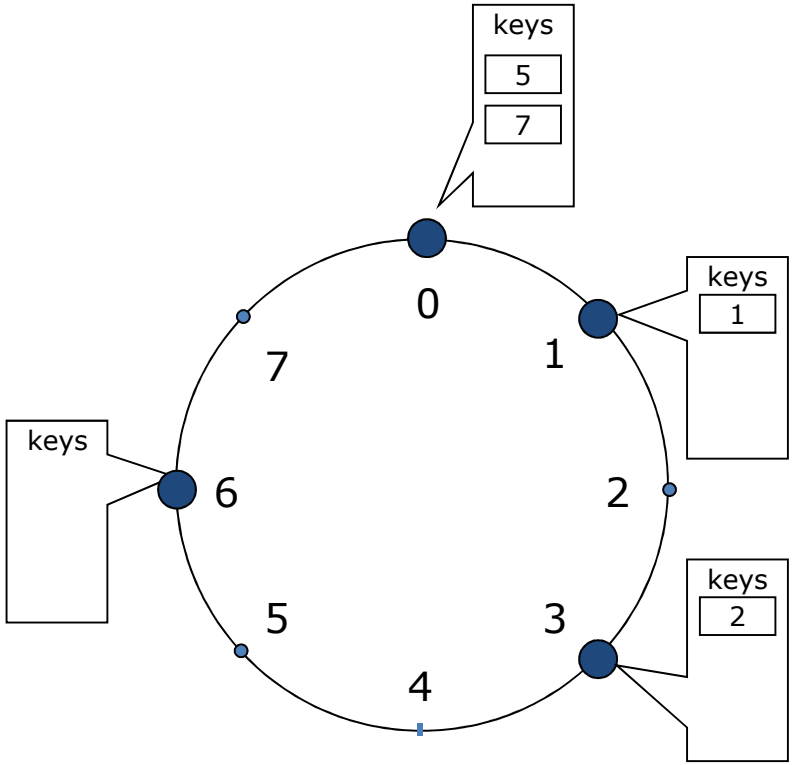
23 ≥ 23 i 23 < 25
23 ∈ [23,25)

Next Hop = 26

Chord

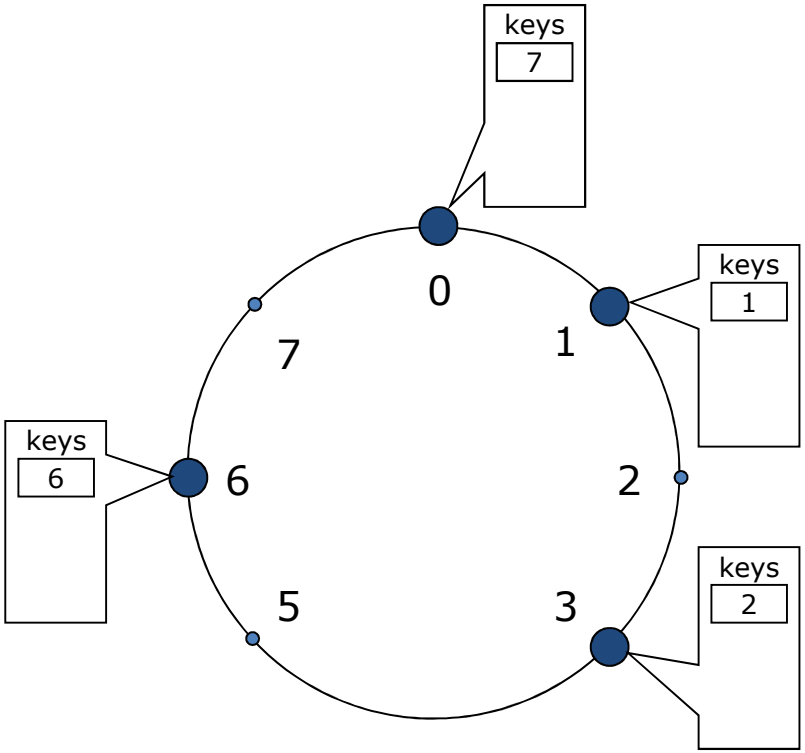
- **API**
  - lookup(key)
  - insert(key, value) - wstawienie key/value do węzła
  - update(key, newval)
  - join(n)
  - leave(n)

- **Wstawianie węzła**
  - join(n)



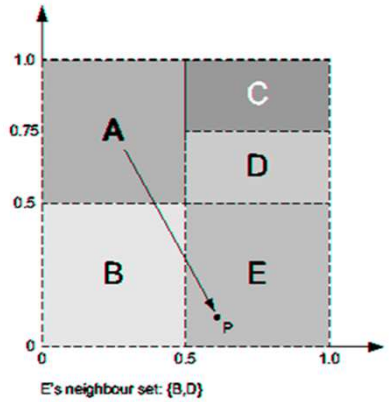
Chord

- **Usuwanie węzła**
  - leave(n)



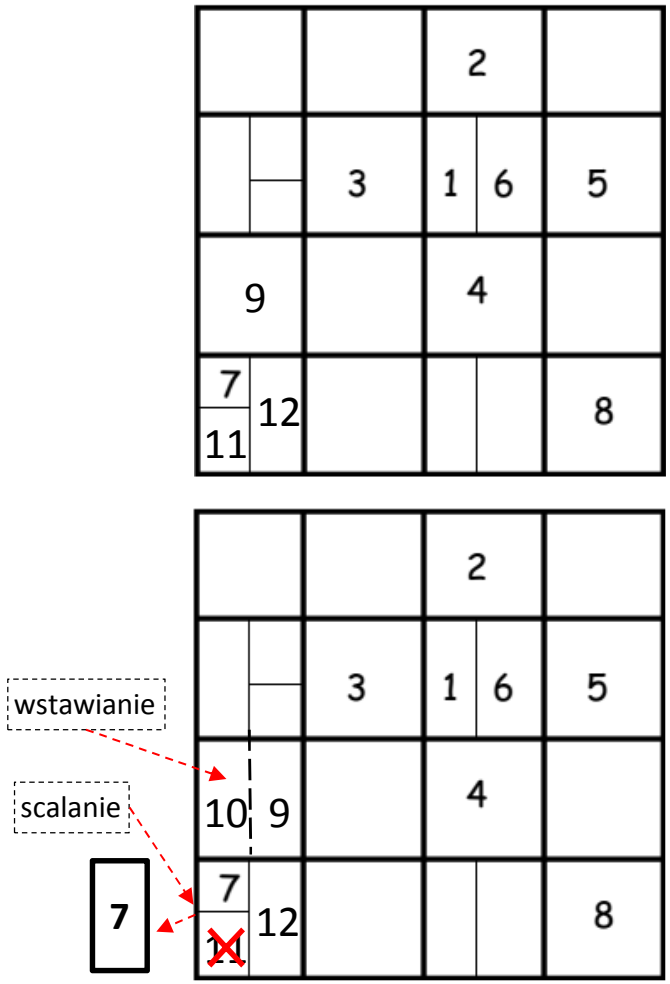
CAN

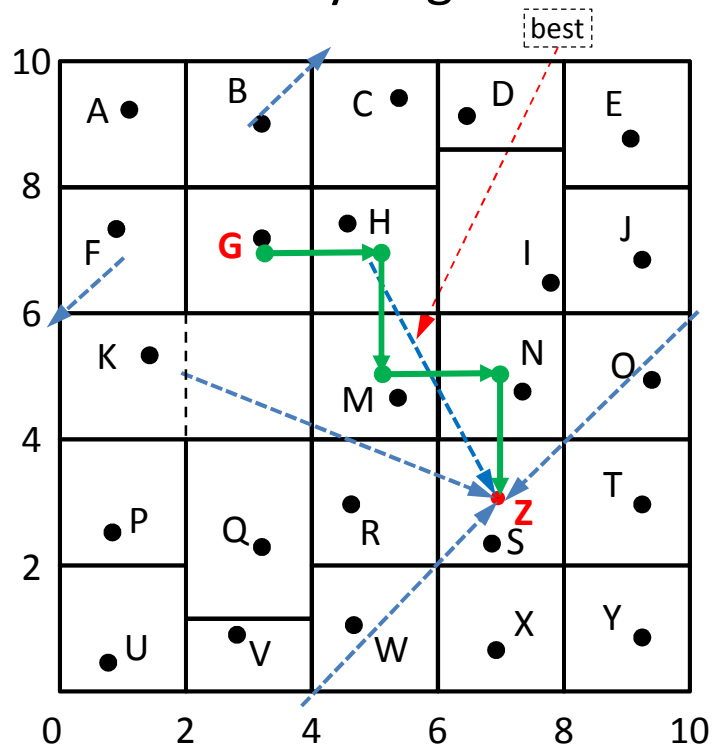
- **CAN (ang. *Content Addressable Network*)**
  - Przestrzeń identyfikatorów zasobów
    - Węzły i zasoby mają identyfikatory z tej samej przestrzeni – wartości ***hash***,
    - Przedstawione w d-wymiarowej przestrzeni $<d_1,d_2,\ldots,d_d>$
      - Najprościej uzyskać poprzez przyporządkowanie określonej liczby bitów wartości hash dla każdego wymiaru
    - Operacje na identyfikatorach: modulo największa koordynata w każdym wymiarze $\rightarrow$ przestrzeń tworzy **torus d-wymiarowy**
    - Bywa przedstawiana w układzie znormalizowanym (jak na rys.)
  - Przestrzeń podzielona jest na strefy
  - Strefy przydzielone są poszczególnym węzłom
  - Obiekt jest umieszczany w strefie która zawiera jego klucz
    - Np. (0.6 , 0.1) umieszczany jest w E



CAN

- **Sąsiedztwo**
  - Węzły znają swoich sąsiadów w każdym wymiarze
  - Węzły są sąsiadami jeśli pokrywają częściowo d-1 wymiarów a w jednym wymiarze "stykają" się
    - Np. 6 ma sąsiadów: 1, 2, 5, 4
- **Wstawianie**
  - Wybór/losowanie punktu i podział danej strefy na pół
    - Znalezienie węzła
    - Wybór strefy
    - Uaktualnienie stanów sąsiadów
- **Usuwanie węzła**
  - Scalanie stref w d-wymiarową bryłę wypukłą (tu prostokąt)



CAN

- **Trasowanie / wyszukanie zasobu**
  - Aby zlokalizować dany zasób należy znaleźć obszar odpowiadający parze (key, value) przypisanej zasobowi
  - Węzły posiadają informacje o sąsiednich obszarach wymagane do routingu
  - Węzeł sam może wyliczyć kierunek dla szukanego zasobu
  - Algorytm: wybór sąsiada z obszaru o koordynatach bliższych obszaru docelowego
    - mierzone od sąsiada … lub
    - od siebie w kolejnych wymiarach
  - Przy założeniu równego podziału dla n peerów (stref) i d wymiarów średnia dł. ścieżki: **sds = (d/4)·($n^{1/d}$)**