Systemy rozproszone danych strukturalnych

Seminarium Systemy Rozproszone 2010/2011

Marcin Walas

21 kwietnia 2011

NoSQL

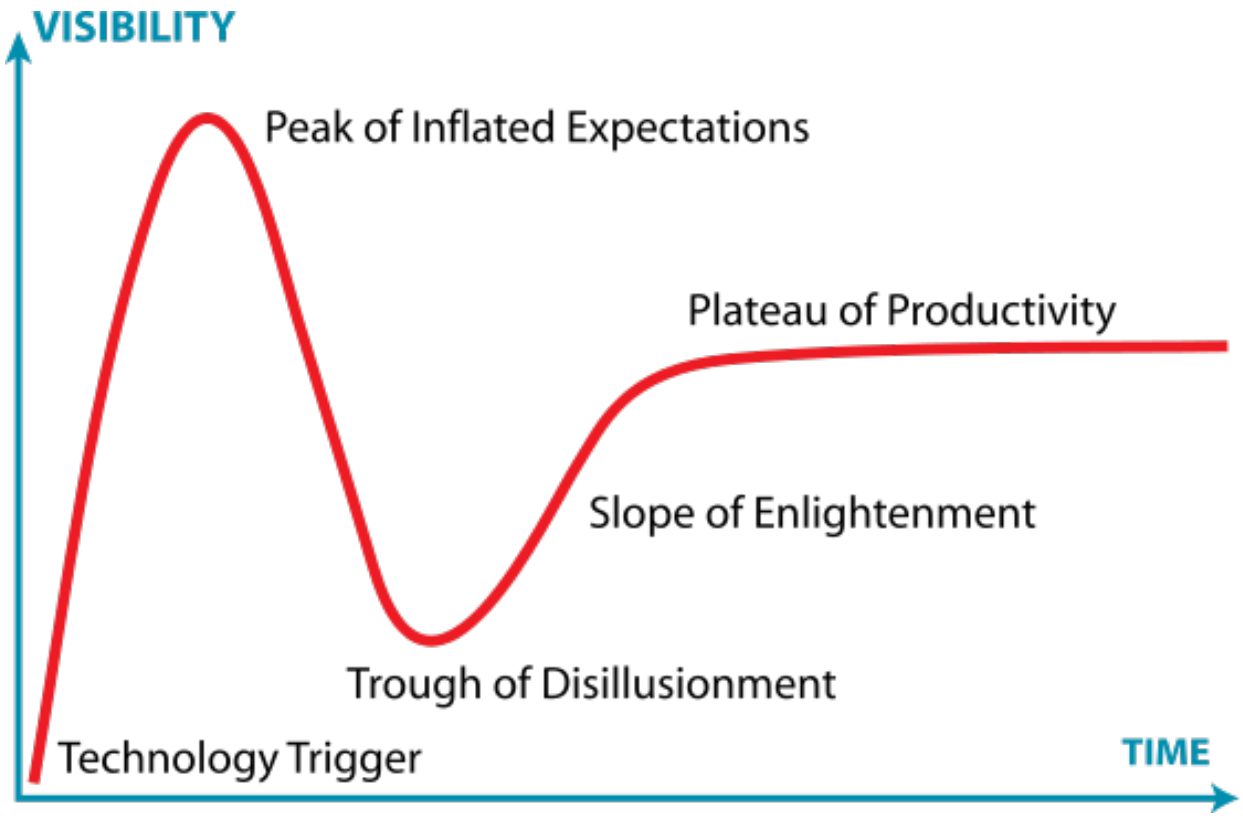
NoSQL to określenie na systemy zarządzania bazami danych, które różnią się od klasycznych RDBMSes w pewien sposób.

- z reguły nie mają ustalonych schematów tabelarycznych
- unikają operacji typu join
- dobrze się skalują

Dlaczego o tym na seminarium z Systemów Rozproszonych?

- dużo danych
- wysokie wymogi dostępności
- skalowanie się
- czyli, idziemy w chmurę

Odrobinka historii



Rodzaje systemów danych strukturalnych

Będziemy po trochu zajmować się wieloma systemami:

- document stores (CouchDB, MongoDB, OrientDB)
- graph stores (Triplestores: Virtuoso)
- key-value stores (Dynamo, Cassandra)
- object databases (Loxim)
- tabular stores (BigTable, Mnesia)
- tuple stores (Jini, Apache River - JavaSpaces)

Przypomnienie

ACID - wymagania transakcyjne

- atomowość
- spójność
- izolacja
- trwałość

Twierdzenie **CAP** (Brewer'a)

System rozproszony nie może jednocześnie zapewniać:

- spójności
- dostępności
- odporności na podziały

Document stores

Dokumenty, czyli:

- JSON
- xml

Potencjalne zastosowania

- CMS'y
- szybkie systemy raportujące
- prosty model danych i łatwa do wymuszenia spójność między dokumentami

CouchDB

http://couchdb.apache.org/

Napisany w Erlangu, zorientowany na dokumenty system bazodanowy

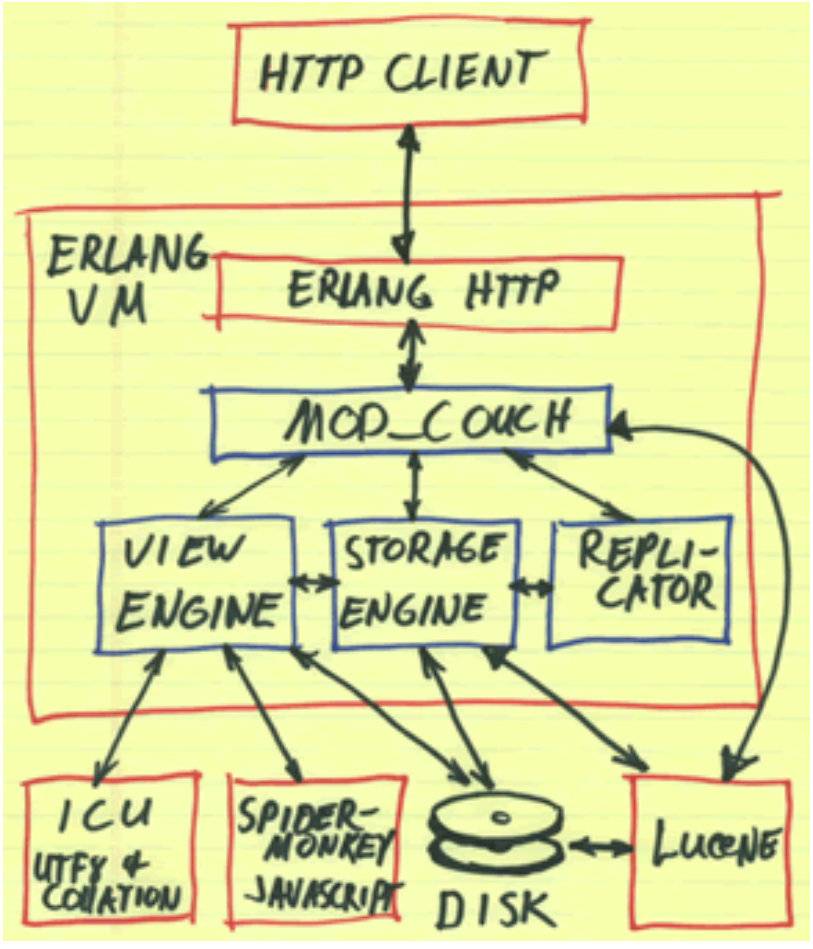
- JSON
- brak schematu - NoSQL!
- klucz-wartość + dowolne załączniki
- replikacja, widoki, map-reduce
- SpiderMonkey - zapytania w JavaScript
- RESTful

CouchDB

Przykład

- widoki - funkcje mapujące stają się indeksami
- ACID

CouchDB - Architektura



http://horicky.blogspot.com/2008/10/couchdb-implementation.html

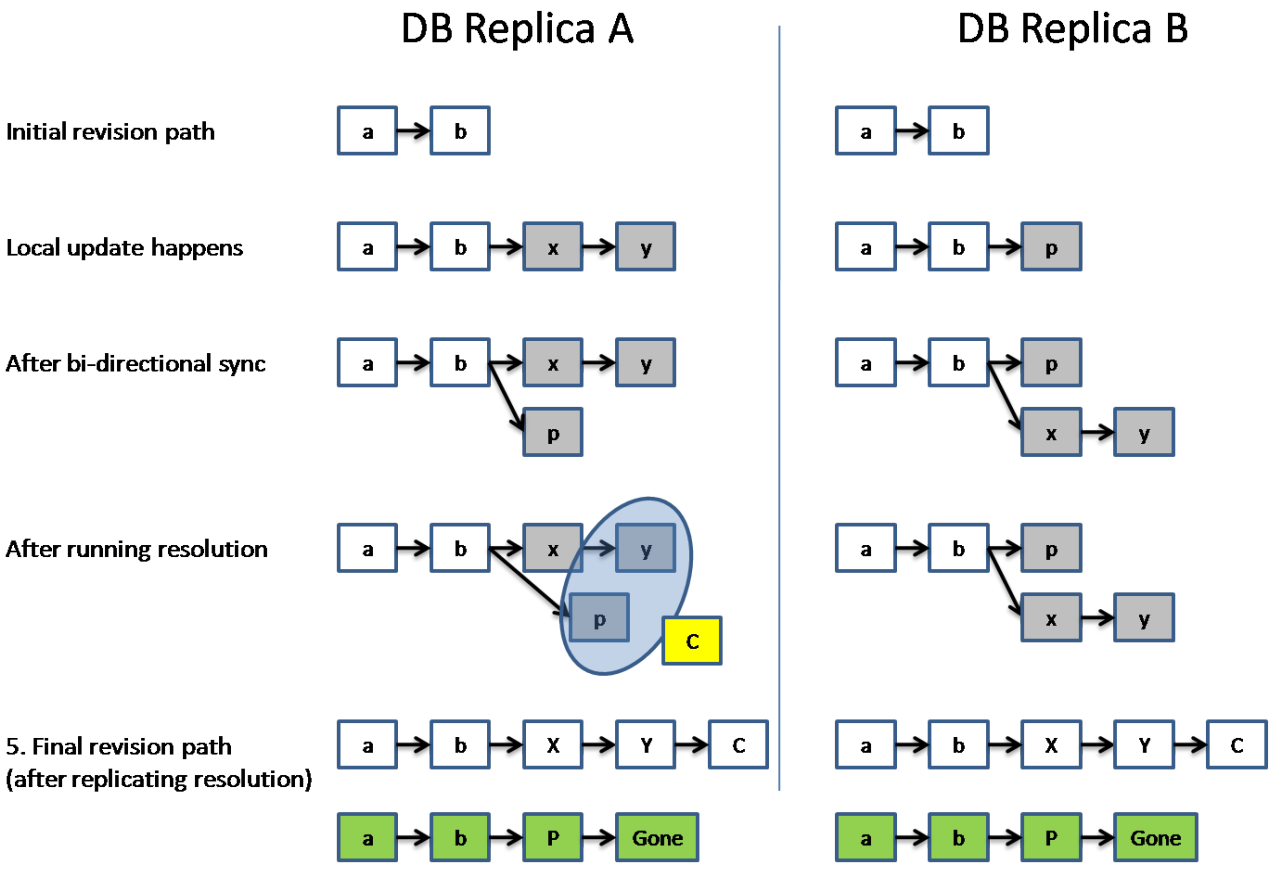
CouchDB - ACID

Własności:

- baza się nie wyłącza - dokonuje zapisów inkrementacyjnych - można ją "ubijać",
- czytelnicy i pisarze nigdy nie czekają,
- spójność danych tylko w obrębie jednego dokumentu,
- zserializowane "update'y" są zapisywane na dysku,
- Multi-Version Concurrency Control
- dokumenty indeksownane w b-drzewach,
- kompaktowanie danych na dysku - co jakiś czas.

http://couchdb.apache.org/docs/overview.html

CouchDB - replikacja



CouchDB - rozproszenie

- load balancing
- replikacja
- plotkowanie zmian

Wbrew temu, co zapewniają twórcy,

problem jednak w tym, że trzeba tutaj tak naprawdę zewnętrznego systemu użyć.

MongoDB

Różnice w porównaniu z CouchDB:

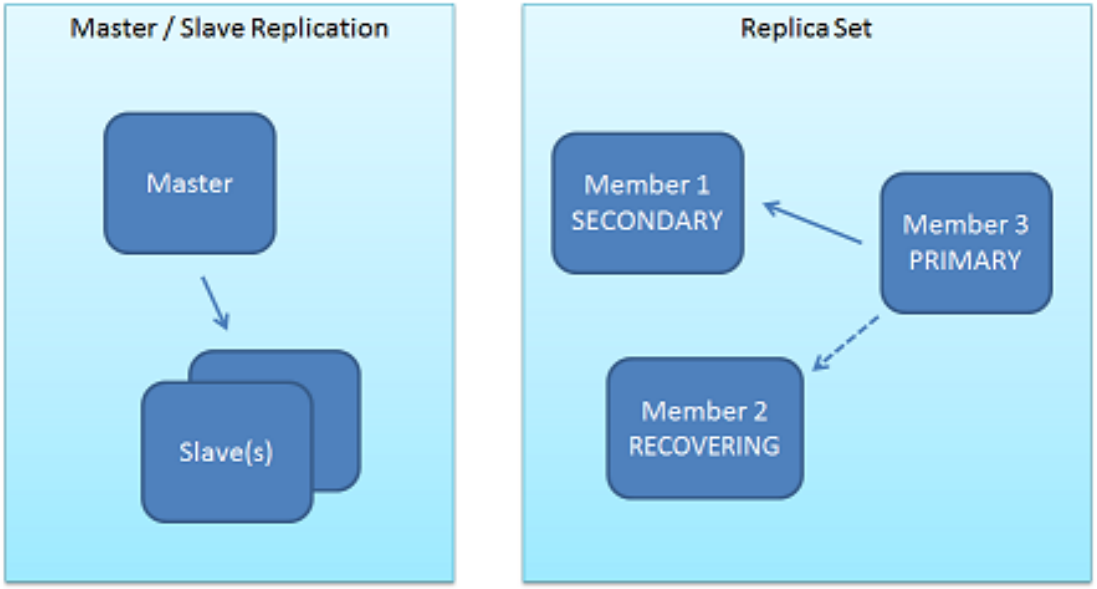
- w testach - niewielki bardzo nadrzut nad RAM,
- ... bo możliwa utrata danych - głównie “persystencja” w RAM,
- napisana w C++,

Produkcja:

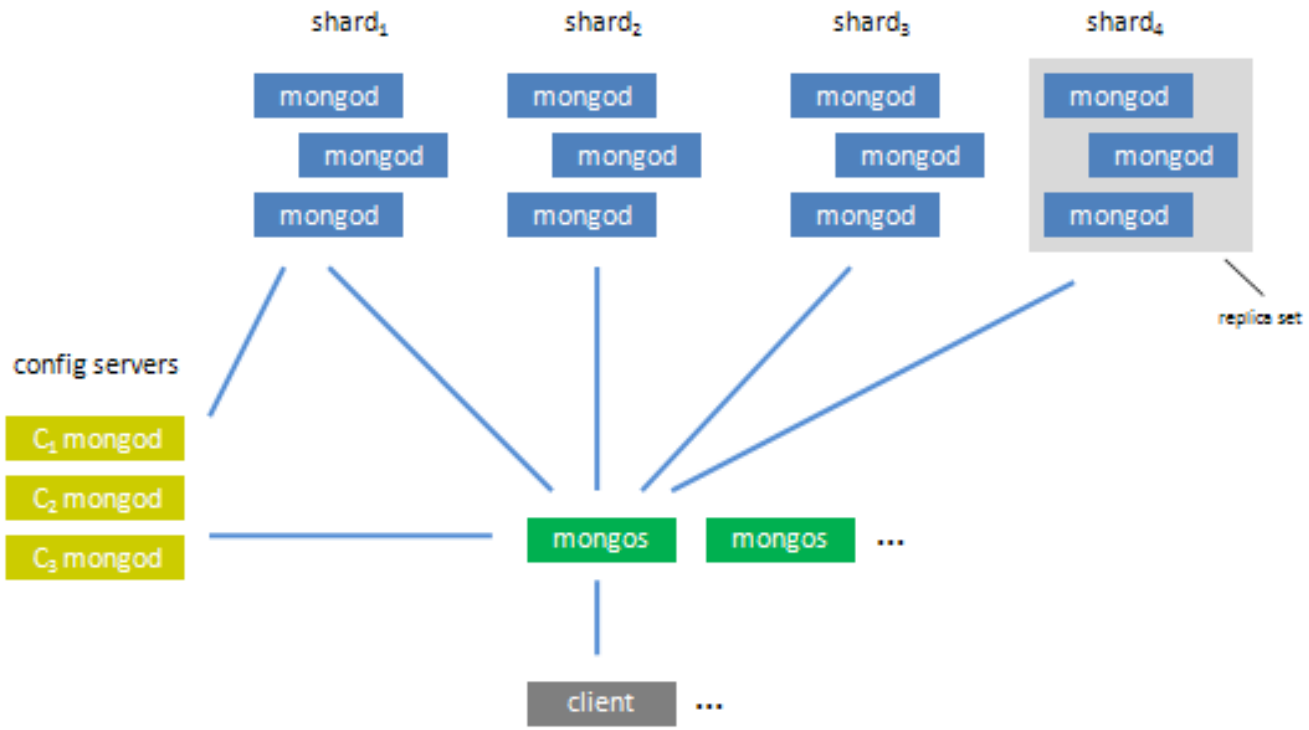
- foursquare™
- sourceforge

Przykład

MongoDB - replikacja



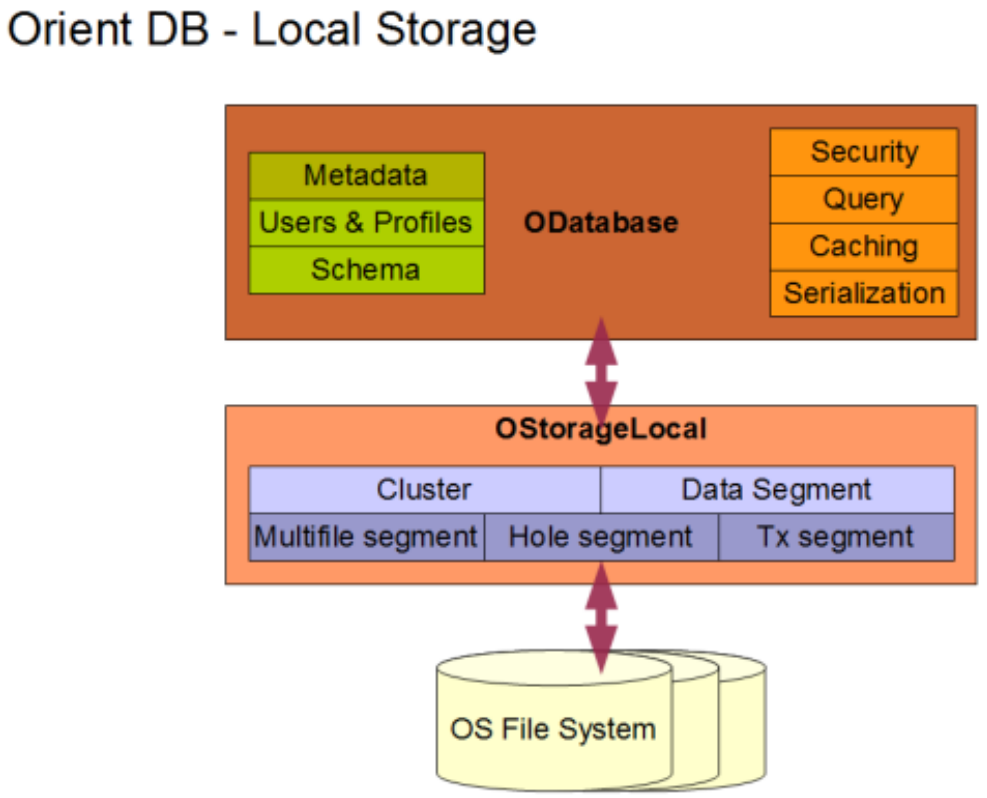
MongoDB - sharding



OrientDB

- baza dokumentów,
- napisana w Javie,
- posiada także cechy graph database - związki między dokumentami można definiować,
- częściowo ustalony schemat dokumentów - klasy,
- szybkie trawersowanie dokumentów,
- ograniczony SQL,
- transakcyjność nie tylko w obrębie dokumentu

OrientDB - architektura

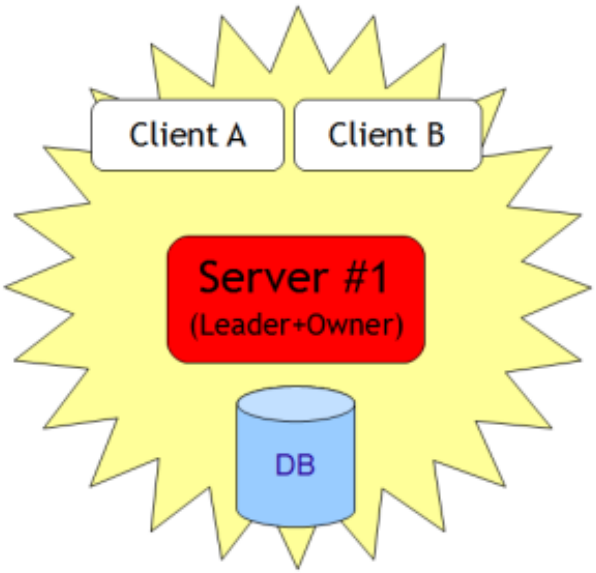


For more information visit: http://www.orientechnologies.com

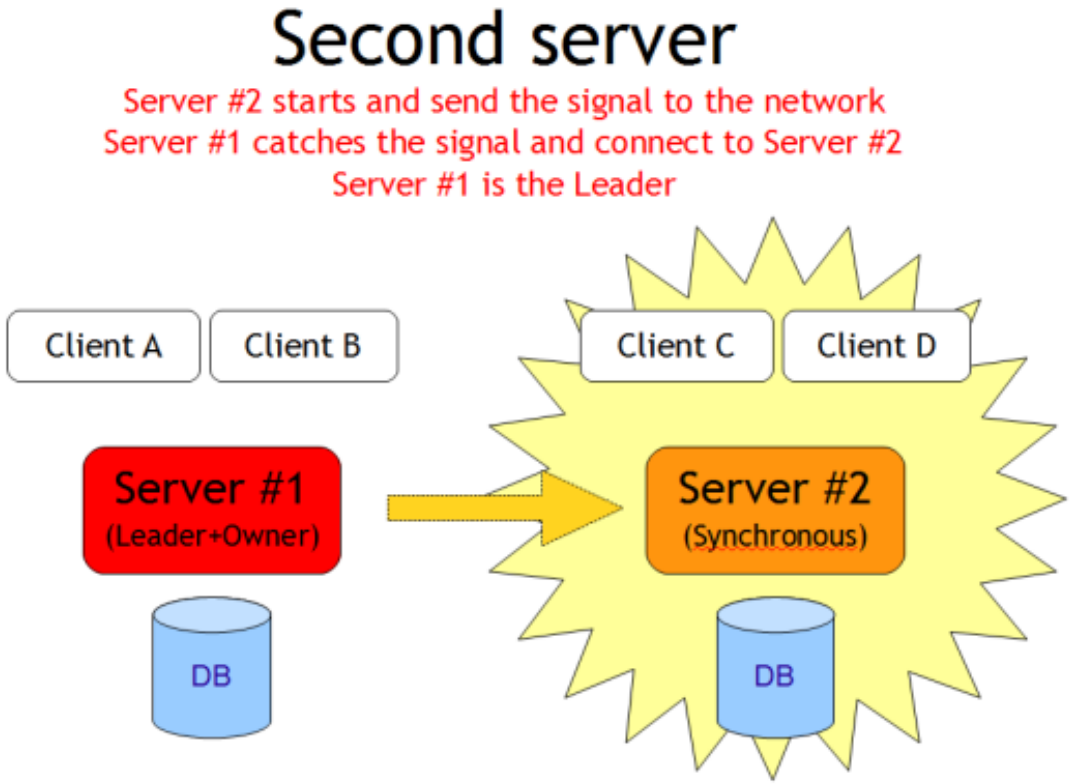
OrientDB - clustering

Single server

Server #1 starts and send the signal to the network
After a timeout no server answer, so no cluster is available
Server #1 listens for other server nodes



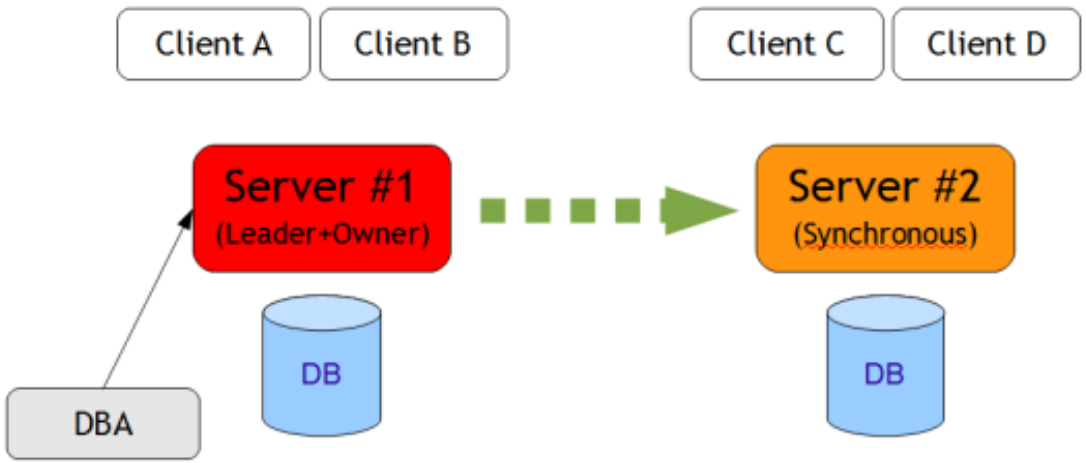
OrientDB - clustering



OrientDB - clustering

Share db

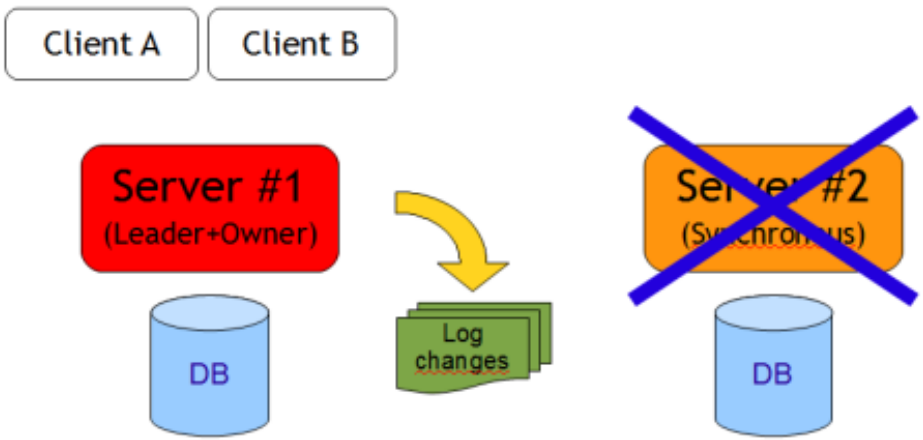
DBA requests a sharing of database to Server #1 in synchronous mode
Server #1 exports the db and send it via streaming to Server #2
When aligned, Server #2 is a synchronous copy of Server #1
Server #1 and #2 are **Always Consistent**



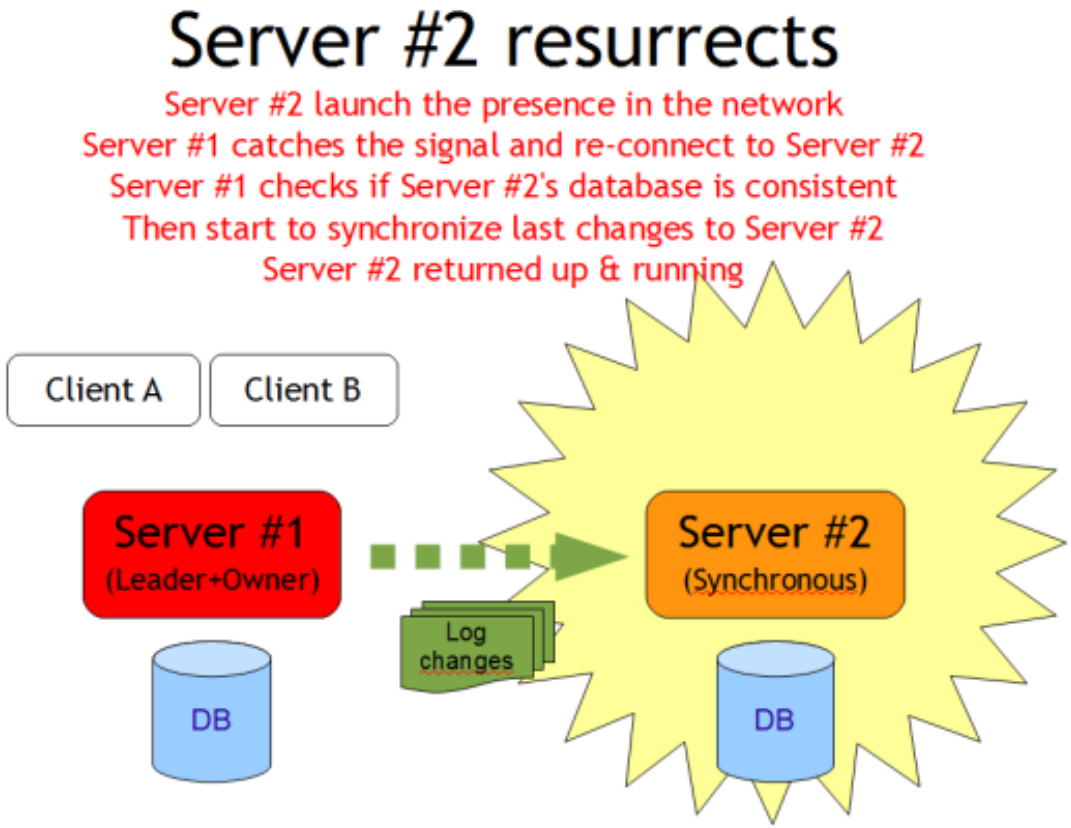
OrientDB - clustering

Server #2 crashes

Server #1 catches it thanks to the heart-beat
Server #1 logs changes while Server #2 is unavailable



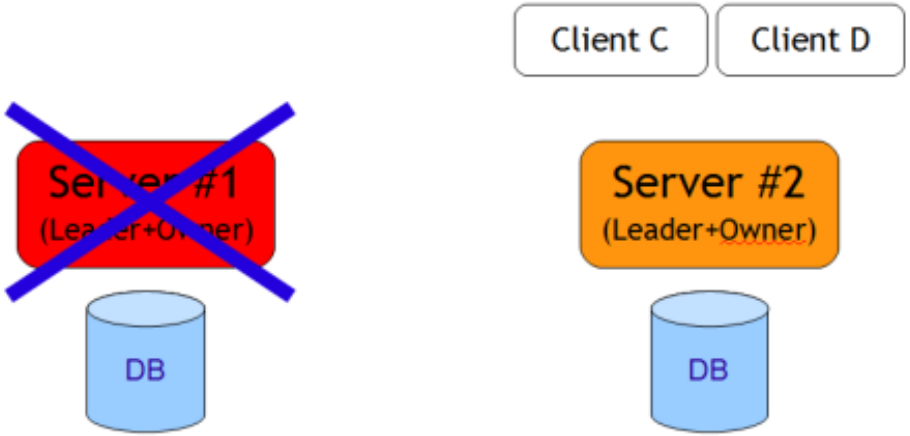
OrientDB - clustering



OrientDB - clustering

Server #1 crashes

Server #2 catches by checking last heart-beat received
Server #2 is the new Leader of the cluster



Erlang

- wpływ na jego rozwój miał Prolog
- stworzony dla obliczeń współbieżnych/rozproszonych
- podobno coś z niego zaczerpnęła Scala i Clojure (wielowątkowy Lisp)
- luźno implementuje CSP
- bazuje na “zielonych wątkach”

Erlang - kawałek kodu

Dlaczego warto na to poświęcić chwilę?

Podobne rzeczy w **Go** od Google'a oraz **Scala**.

Scala - kawałek kodu

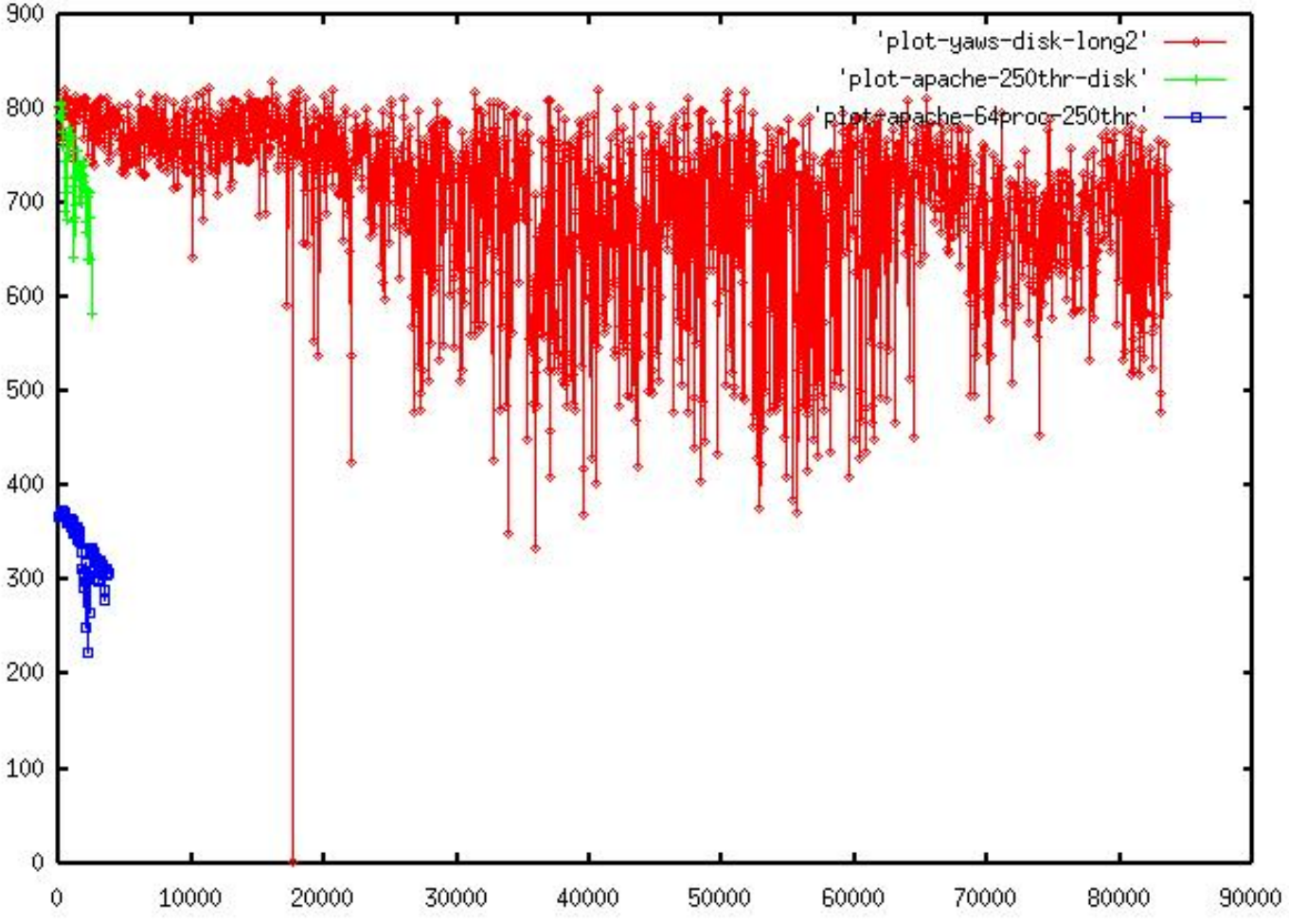
Scala - **Aktorzy**

Możliwi zieloni lub ciężkie, systemowe wątki.

No i bardziej Java-way-of-life, jak ktoś lubi.

Więcej tutaj na przykład:
http://www.infoq.com/news/2008/06/scala-vs-erlang

Yaws - Yet another web server



przepustowość (KBytes/second) vs. obciążenie
zaczerpnięte z http://www.sics.se/ joe/apachevsyaws.html

Postęp...

Będziemy po trochu zajmować się wieloma systemami:

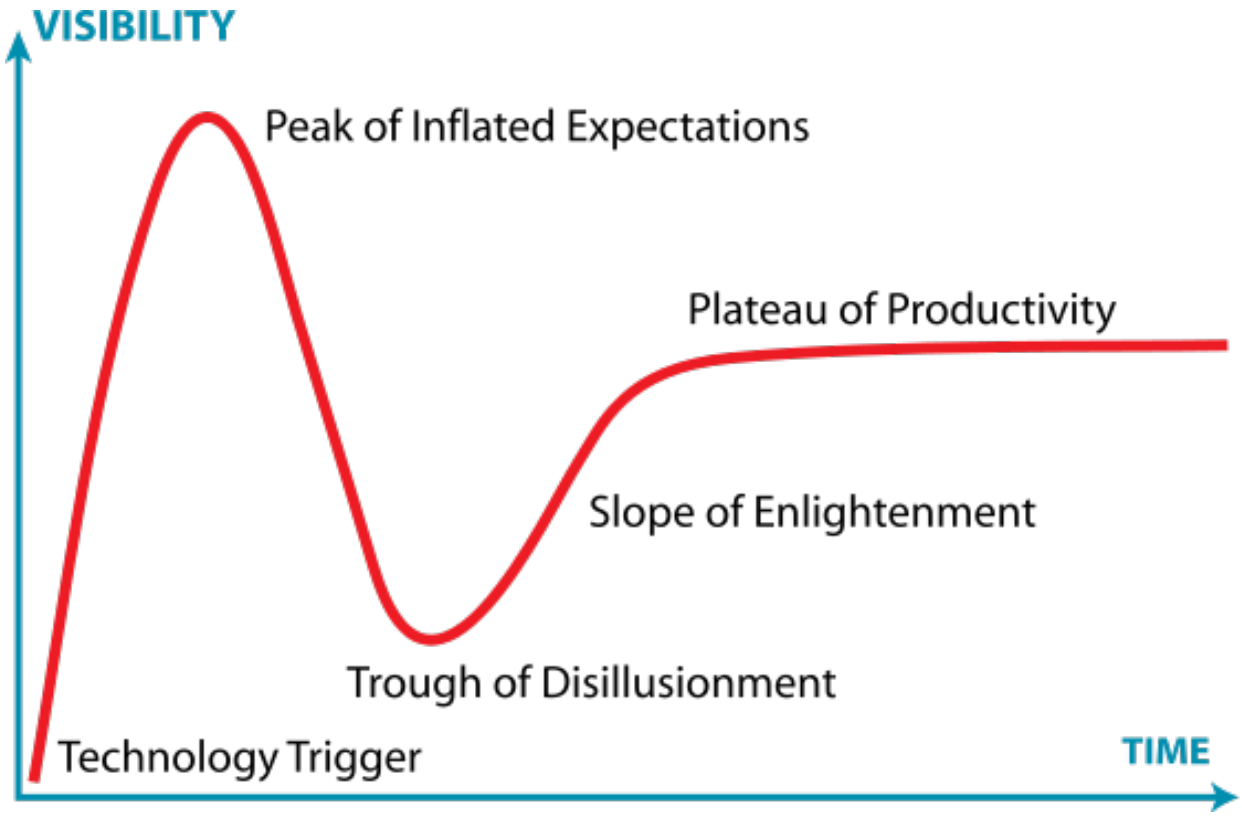
- **document stores (CouchDB, MongoDB, OrientDB)**
- graph stores (Triplestores: Virtuoso)
- key-value stores (Dynamo, Cassandra)
- object databases (Loxim)
- tabular stores (BigTable, Mnesia)
- tuple stores (Jini, Apache River - JavaSpaces)

Triplestores

Bazy danych zorientowane na przechowywanie informacji w postaci RDF - etykietowanych grafów.

- wiele projektów upiorów
- trochę prac teoretyków

Status sieci semantycznej



A może ten graf się po prostu nie stosuje do niczego związanego ze sztuczną inteligencją?

Resource Description Framework

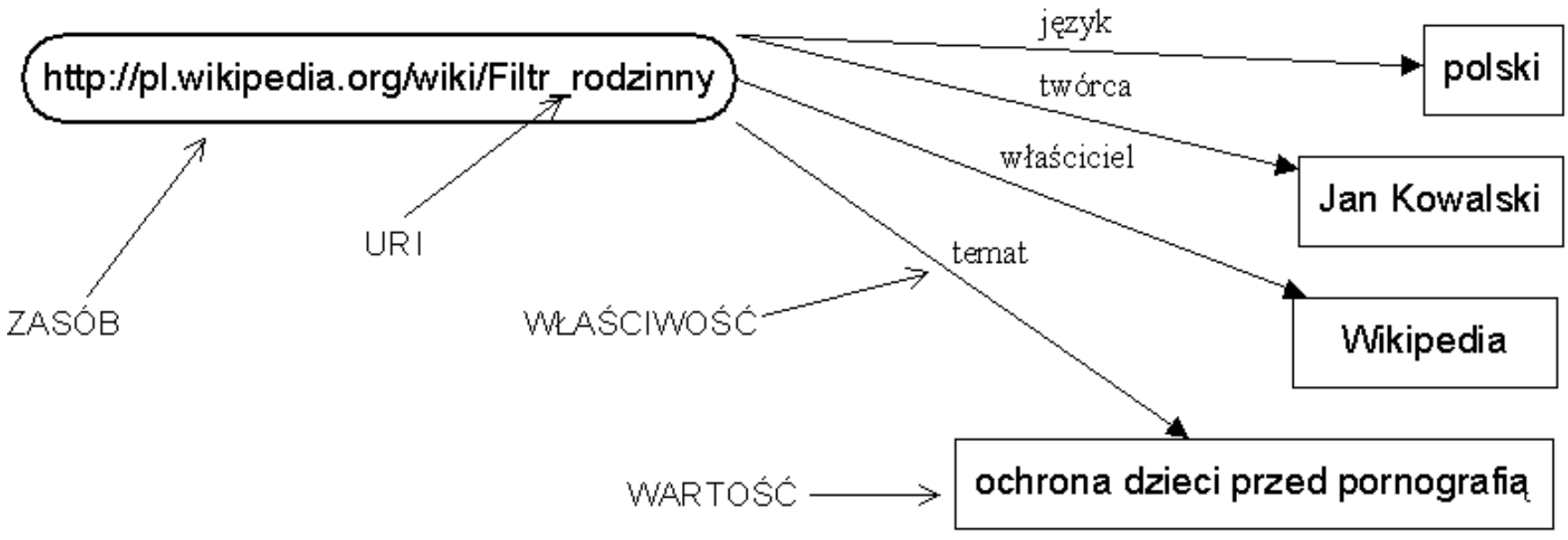
```
<?xml version="1.0"?>
<RDF>
<Description about="http://pl.wikipedia.org/wiki/Wikipedysta:KoziK">
    <autor>Krzysztof Kozłowski</autor>
    <utworzono>1 Maja 2009</utworzono>
    <zmodyfikowano>1 Stycznia 2010</zmodyfikowano>
</Description>
</RDF>
```

Opis świata w postaci:

- podmiotu,
- orzeczenia/predykatu (własność)
- dopełnienia/obiektu (wartość)

Dużo trójek - stąd triplestores.

Resource Description Framework



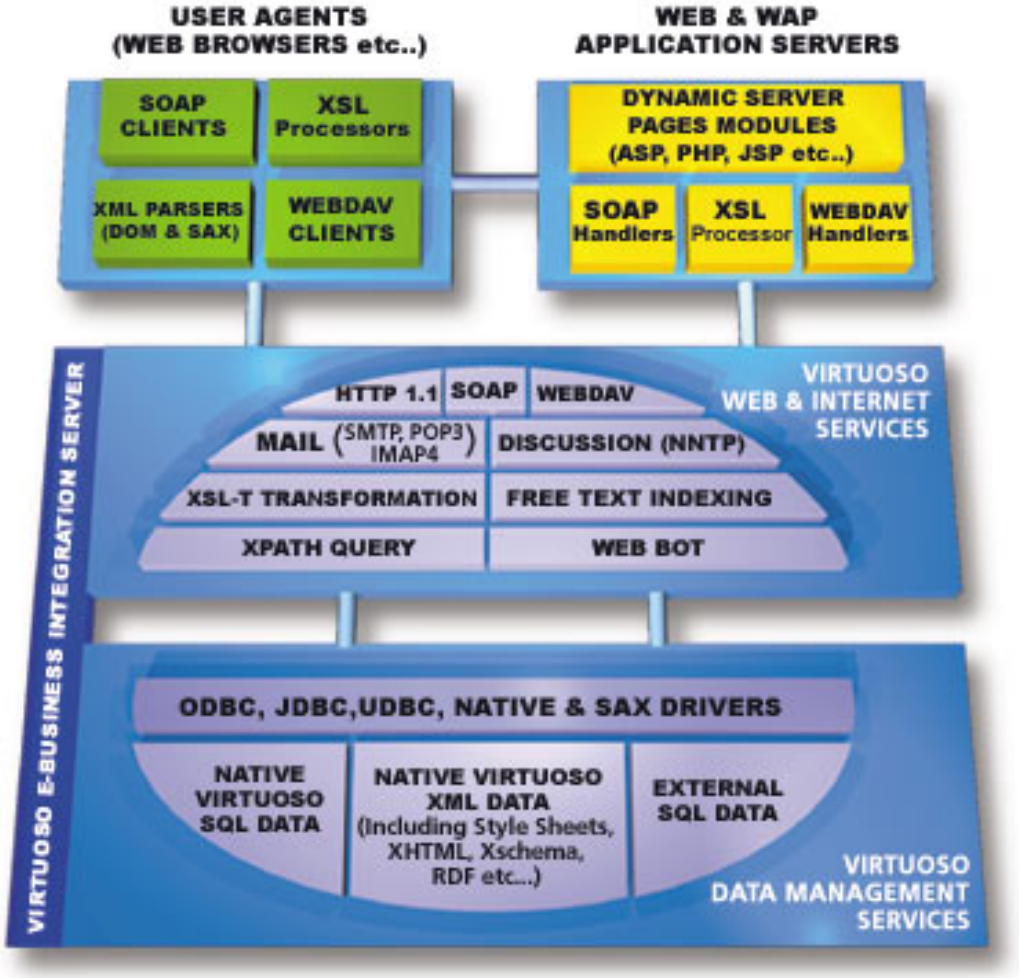
Virtuoso

- uniwersalny serwer - czyli nas interesuje tylko kawałeczek
- komercjalny, jest wersja Open Source
- duży produkt: SQL, XML,WebDAV, federacje, SOAP

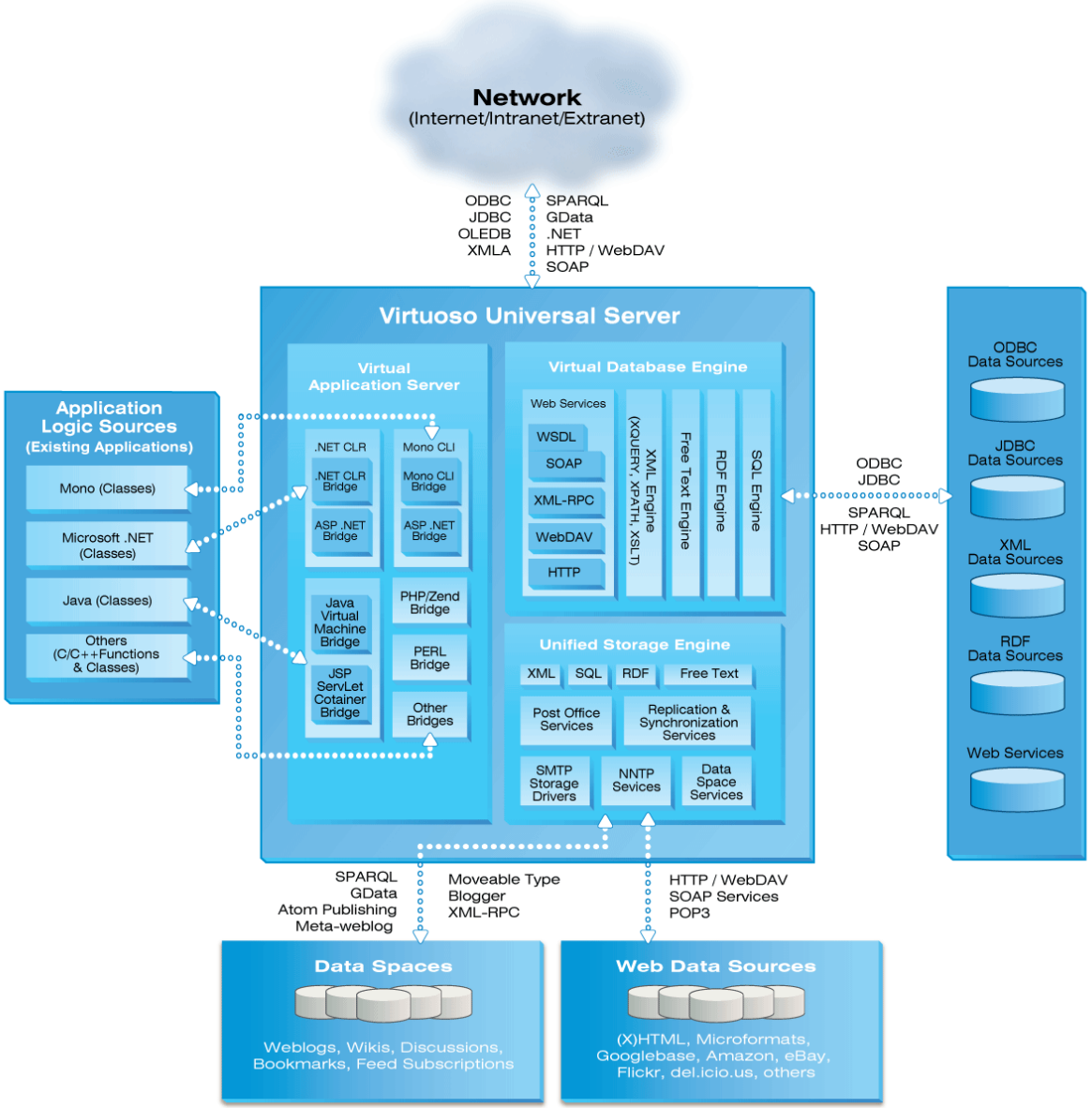
Używa tego:

- 

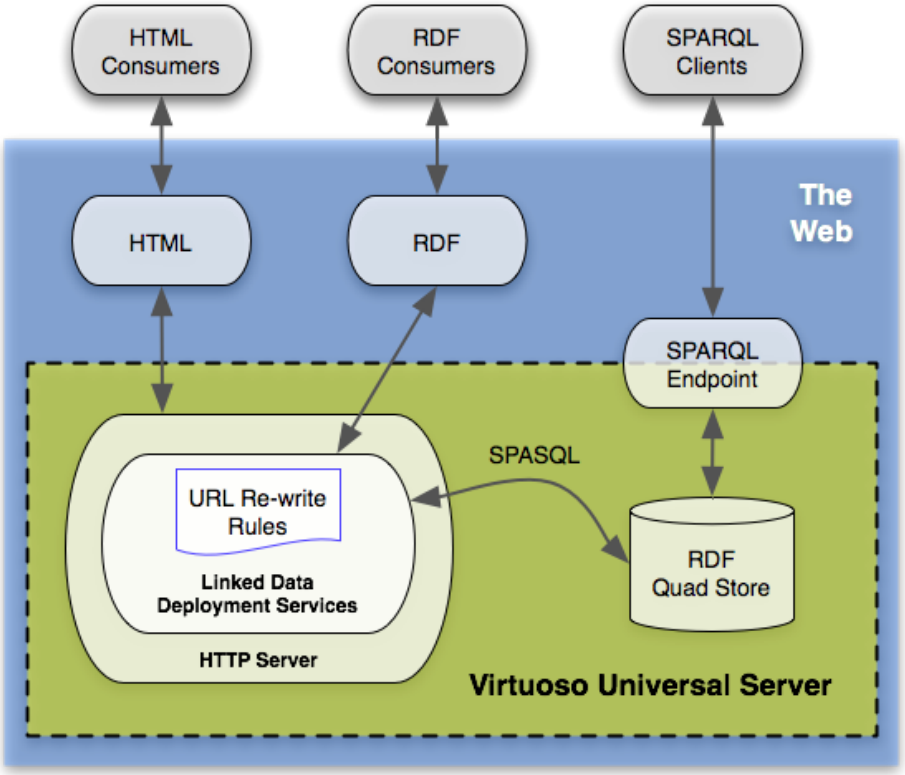
Architektura



Architektura



Przykład - DBPedia



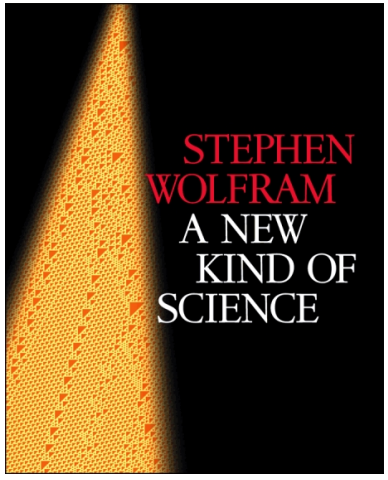
Przy okazji

- Freebase - ze storage grafowym od Metaweb,
- google-refine

WolframAlpha

Czy ktoś nie wie, co to jest?

Lub to:



A New Kind of Science?

Automaty komórkowe:

- proste algorytmy,
- ewolucja,
- samoaplikacja

Wolfram Alpha:

- oparty na Mathematica - flagowy produkt Wolfram Research,
- **nie jest** wyszukiwarką sematyczną,
- wizualizuje zgromadzoną wiedzę, przez składanie prostych algorytmów,
- podobny do Cyc [sajk]

OpenCyc

- rozpoczęty w 1984 roku,
- podobno zbudowanie “zdrowego rozsądku” zajęłoby 350 man-years,
- podobny do Lisp’a

Niestety, najciekawsze dla nas rzeczy są “zrób to sam”

- jest komunikacja sieciowa,
- ale ostatecznie i tak trzeba to pisać ręcznie.

gridMathematica

Wolfram gridMathematica 7 SERVER

PARALLEL COMPUTATIONS MADE EASY ACROSS A NETWORK

- load-balancing,
- parametryzowany scheduling zadań,
- durability: po awarii powinien system wrócić sam do pracy,
- obsługa klastrów, gridów, możliwość użycia zwykłych, tanich maszyn,

Produkt komercyjny, więc tak naprawdę nic nie wiadomo...

Postęp...

Będziemy po trochu zajmować się wieloma systemami:

- **document stores (CouchDB, MongoDB, OrientDB)**
- **graph stores (Triplestores: Virtuoso)**
- key-value stores (Dynamo, Cassandra)
- object databases (Loxim)
- tabular stores (BigTable, Mnesia)
- tuple stores (Jini, Apache River - JavaSpaces)

Przestrzeń krotek

Co to?

- wielki shared memory,
- proces wkłada krotkę,
- inny proces wkłada krotkę pasującą do żądania

Implementacje

- JavaSpaces
- PyLinda - prosta impl
- GigaSpaces - komercyjne, też .NET

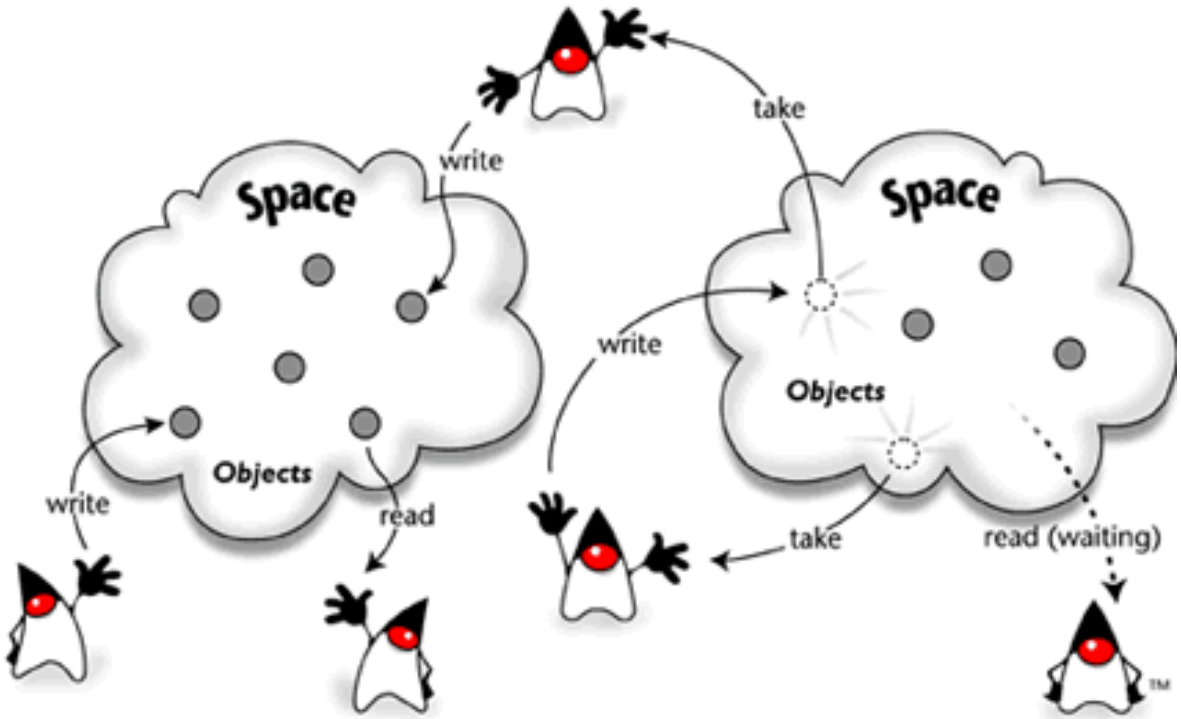
Javaspaces

- zamiast krotek - obiekty dziedziczące po *Entry*,
- mamy wielki worek na obiekty,
- najczęściej używane w trybie Master-Slave,
- raczej bez persystencji obiektów

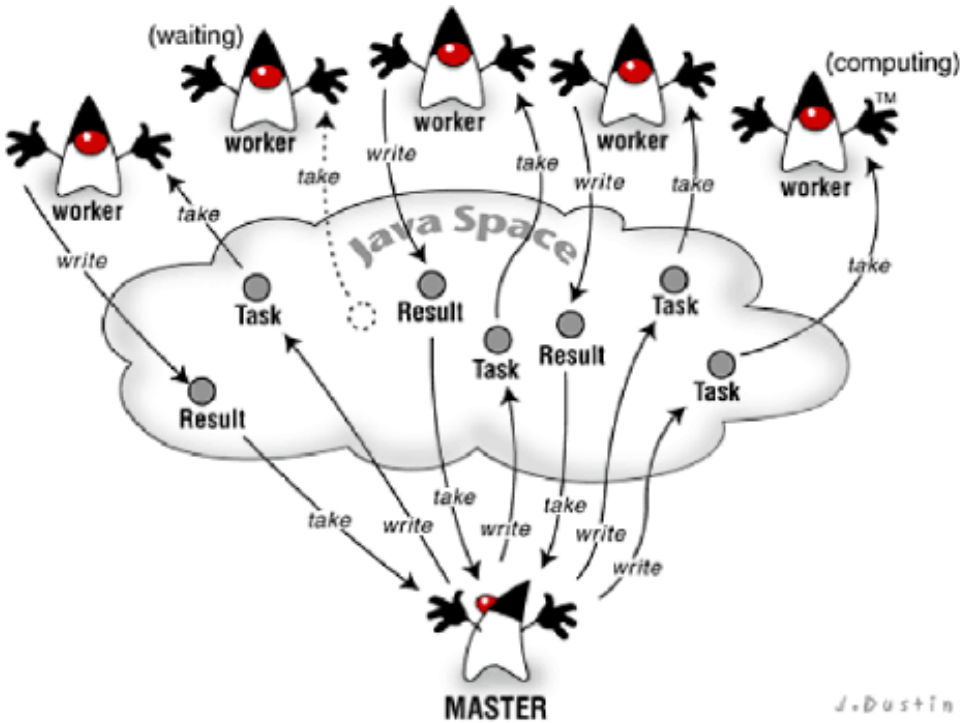
Operacje:

- write,
- read,
- take,
- notify

Javaspaces


Space
Objects
write
read
take
write
Space
Objects
write
take
read (waiting)

Javaspaces


(waiting)
worker
worker
worker
worker
(computing)
worker
write
take
Java Space
Task
Result
Task
Result
Task
Task
Result
MASTER
J.Dustin

Javaspaces - przykład

```
import net.jini.core.entry.*;

public class MessageEntry implements Entry {
   public String content;

   public MessageEntry() {
   }

   public MessageEntry(String content) {
     this.content = content;
   }

   public String toString() {
     return "MessageContent: " + content;
   }
}
```

Javaspaces - przykład

```
JavaSpace space = getSpace();
MessageEntry msg = new MessageEntry();
msg.content = "Hello there";
space.write(msg, null, Lease.FOREVER);
```

Javaspaces - przykład

```
MessageEntry template = new MessageEntry();
// template content field = NULL, match everything
MessageEntry output = (MessageEntry) space.read(template, null,
                                                Long.MAX_VALUE);
```

Apache River

- kiedyś Jini,
- framework do budowy systemów rozproszonych,
- zawiera implementację Javaspaces,
- bardziej rozwinięte JRMI,
- automatyczne wykrywanie usług,
- SOA - Service Oriented Architecture,

Komunikacja

Przez protokół JERI. Implementacje JERI:

- TCP,
- SSL,
- HTTP,
- HTTPS,
- Kerberos-TCP

Od małego do dużego...

- **Najmniejszy**: usługa i klient
- **Średni**: usługa, klient, rejestr
- **Największy**: usługa, klient, rejestr, serwer klas

Apache River - co zapewnia

Peter Deutsch z Sun Labs *Eight Fallacies of Distributed Computing*:

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Apache River - co zapewnia

Jini tak naprawdę odnosi się częściowo do wszystkich tych założeń,

- niezależność od protokołu - wiele implementacji, można dopisać własną (:)!),
- niezależność od lokalizacji, można przenosić usługi bez wiedzy klientów,
- eliminacja powiązań - może wiele implementacji tej samego interface'u współistnieć w sieci,
- automatyczny tuning wydajności

Pytania?

Bardzo dziękuję za uwagę.