





Jeden rozmiar nie dla każdego
NOSQL



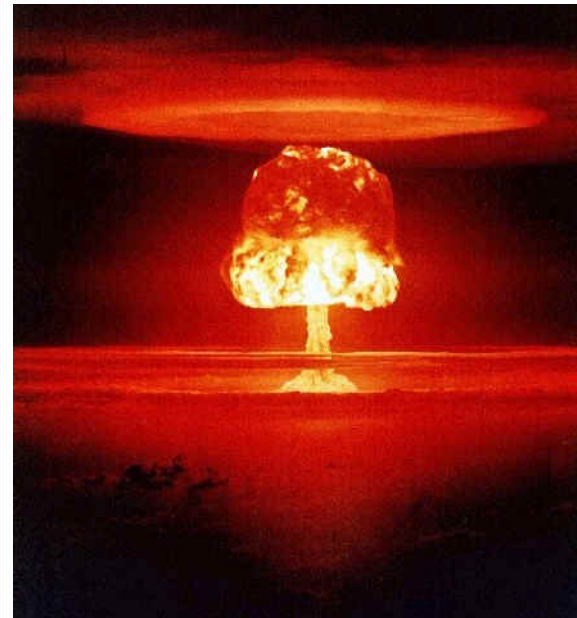
988,
000,
000,
000,
000,
000,
000



Potrzeba matką wynalazków

- **Eksplozja danych**

- 988,000,000,000,000,000,000 bajtów danych dostępnych w Sieci w 2010
- 45 GB na mieszkańca
- 60% procentowy wzrost z roku na rok
- 1,800 exabajtów do 2011 (według danych IDC)



Drogi użytkowniku!

•User Generated Content

- Do końca 2013, 155 milionów użytkowników (tylko w USA), będzie korzystało z informacji stworzonych przez innych,
- 115 milionów użytkowników w USA będzie aktywnie tworzyć zawartość Sieci
- Udostępniamy dane 15x częściej niż pobieramy w porównaniu z rokiem 2008 (dane 2008)



Globalna eksplozja danych

- Zorganizowane
 - «funkcjonujące w ramach formalnej struktury»
 - ... struktury zdefiniowanej przez użytkownika.



Globalna eksplozja danych

- Dostępne
 - «nietrudne do zdobycia»
 - «łatwe do przyswojenia»
 - «takie, z którymi łatwo można nawiązać kontakt, do których jest łatwy dostęp»



Globalna eksplozja danych

- Przydatne
 - «przydające się do czegoś lub mogące się do czegoś przydać»
 - «pomocne w czymś»



Eric Brewer i CAP

- **C**onsistency
Usługa, która jest spójna działa w pełni lub w ogóle
- **A**vailability
Usługa jest dostępna gdy odpowiada na żądania w założonym czasie
- **P**artition Tolerance
Jedynie zupełna awaria sieci może spowodować, że system odpowie niepoprawnie



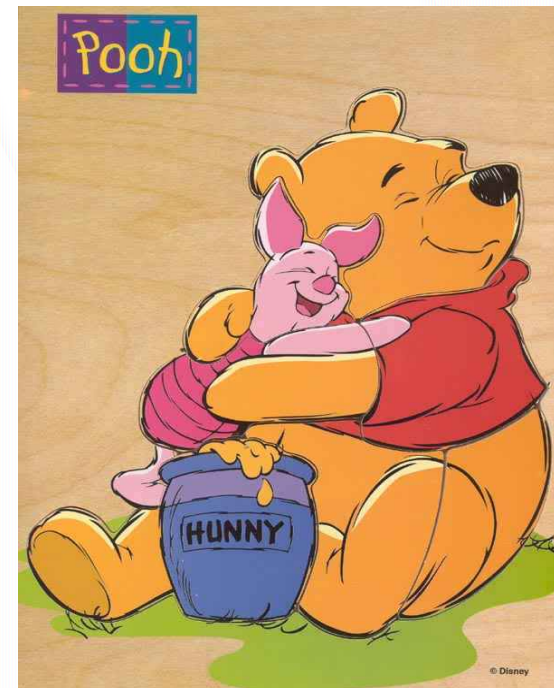
Gorzka prawda

rozproszony system może zaspokoić dowolne
dwie z powyższych gwarancji w tym samym
czasie, ale nie wszystkie trzy jednocześnie



Paradoks continuum czasoprzestrzennego czyli

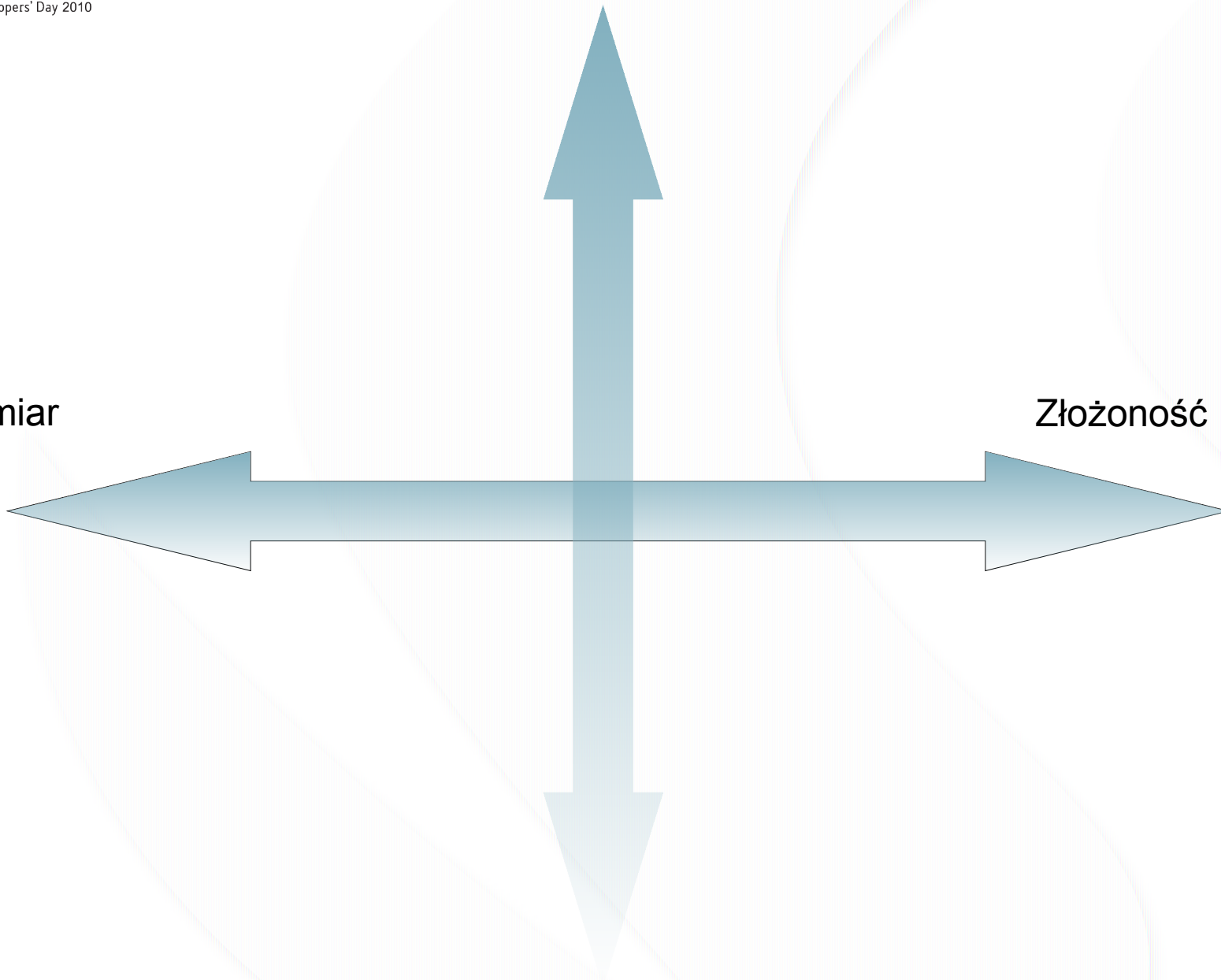
„Im bardziej Puchatek zaglądał do środka, tym bardziej Prosiaczka tam nie było”



Dokładność

Rozmiar

Złożoność



Konsystencja



Not necessary QL



Wolność wyboru

performance

Eventual Consistency

dropping ACID

hate SQL

scaling

open standards

speed

throughput

Open Source

ease of use

sharding

CAP



Think for yourself and question authority



To co czujesz

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**
- **Basically Available**
- **Soft state**
- **Eventually consistent**



Bazar próżności

- Column oriented
- Document store
- Key value store
- Graph
- Object database
- XML database



Demokratyczne wybory



ORACLE[®] 11^g
BERKELEY DB



Embrace this moment. Remember.
We are eternal.
All this pain is an illusion.





Powrót do podstaw

- Baza danych.equals(kolekcja dokumentów)
 - Futon
 - RESTful API
- Dokument
 - JSON
 - `_id`
 - `_rev`



RESTful API

- CRUD
 - PUT `/{database}/{id}`
 - GET `/{database}/{id}`
 - DELETE `/{database}/{id}`



Nowe perspektywy

- Widoki i perspektywy (design documents)
 - Widoki i tymczasowe widoki
 - map
 - reduce
 - show
 - list
 - walidacja



```
1 {
2   "_id": "design/events",
3   "_rev": "10-f810768421c065866cd6c6bcb3bd2188" ,
4   "language": "javascript",
5
6   "views": {
7     "bycity": {
8       "map": ""
9     },
10    "artists": {
11      "map": "",
12      "reduce": ""
13    }
14  },
15
16  "shows" : {
17    "post" : ""
18  },
19
20  "lists" : {
21    "recent" : ""
22  },
23
24  "lib" : {
25    "templates" : {
26      "post" : ""
27    }
28  }
```



Mapuj i redukuj

- **Map**

```
function (doc) {  
    emit (null, doc);  
}
```

- **Reduce**

```
function (key, values, rereduce) {  
    return sum (values);  
}
```



I jakby tego było mało

- GET `/_{database}/_design/{design}/_view/{view}`
- Parametry zapytania
 - key
 - startkey oraz endkey
 - descending
 - group
 - group_level
- Klucze złożone



Nie bój się redukować

- Jeśli `group=true`
 - Funkcja `reduce` wywoływana jest raz dla każdego unikalnego klucza
- Jeśli `group=false`
 - Funkcja `reduce` wywoływana jest raz dla całego widoku, a parametr `key` = `[[key1,docid1], [key2,docid2]]`



Wieża Babel

- View/Query serwer
 - JavaScript
 - Ruby
 - Python
- jcouchdb



Kilka dobrych rad

- Blogi, portale informacyjne, mashups
- Niepowiązane ze sobą encje
- „Free style” model danych
- MongoDB



Life is not longevity and beauty is the only goal





Neo4J

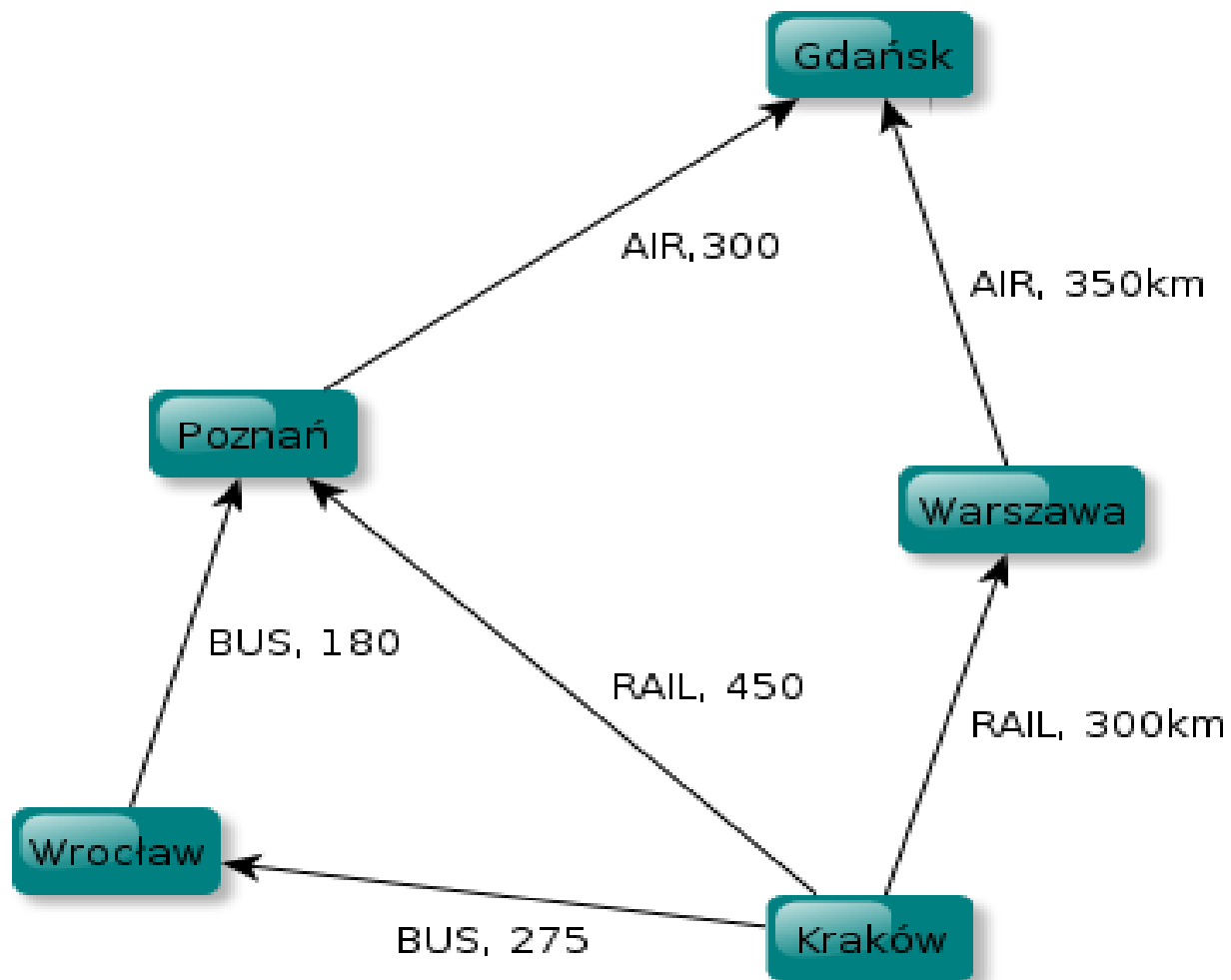


Grafy, wierzchołki, krawędzie

- Graf
 - Wierzchołki (`org.neo4j.graphdb.Node`)
 - Krawędzie (`org.neo4j.graphdb.Relationship`)
 - `org.neo4j.graphdb.RelationshipType`
 - `org.neo4j.graphdb.Direction`
 - Własności
(`org.neo4j.graphdb.PropertyContainer`)
- ACID (JTA)
- That's all my baby



A teraz coś dla przykładu



Podróże w czasie i przestrzeni

```
node.traverse(Traverser.Order,  
             StopEvaluator,  
             ReturnableEvaluator,  
             RelationshipType,  
             Direction)
```



Tu podróż się nie kończy

- TraverseOrder
 - DEPTH_FIRST
 - BREADTH_FIRST
- StopEvaluator
 - DEPTH_ONE
 - END_OF_GRAPH
- ReturnableEvaluator
 - ALL
 - ALL_BUT_START_NODE
- (RelationshipType, Direction)*



Algorytmy + struktury danych

- Neo-graph-algo
 - Najtańsza ścieżka, Dijkstra lub A*
 - Wszystkie ścieżki
 - Proste ścieżki
 - Eccentricity
 - Network diameter
 - Network radius
 - Stress centrality
 - Closeness centrality
 - Betweenness centrality
 - Eigenvector centrality



Droga do domu

- `Dijkstra<T>` (`T`, `initialCost`, `startNode`, `endNode`, `evaluator`, `adder`, `comparator`, `direction`, `relationshipTypes`)
- **CostEvaluator**
- **CostAccumulator**
- **CostComparator**



A gdzie podziały się moje dane

- neo4j-index
- Umożliwia wyszukiwanie wierzchołków i krawędzi,
 - Index
 - IndexService
 - LuceneIndexService
 - LuceneFulltextIndexService



Kilka dobrych rad

- Wszystko co można za modelować jako graf



The world is round, My square don't fit at all



ORACLE
BERKELEY DB **11^g**



Zaspany kot

- Key value store
 - Implementacja napisana w C
 - JNI „wrapper” dla implementacji napisanej w C
 - Natywna implementacja napisana w Javie
- API
 - Base API
 - Direct Persistence Layer (DPL) API
 - Java Collections API
- Wsparcie dla JTA



Coś dla koneserów

- Java Collections API
 - `StoredMap` (`java.util.Map`)
 - `StoredList` (`java.util.List`)
 - `StoredValueSet` (`java.util.Set`)
 - `StoredSortedMap` (`java.util.SortedMap`)
 - `StoredSortedValueSet` (`java.util.SortedSet`)
- ... czyli transakcyjne, zreplikowane oraz rozproszone kolekcje



SQL dla bardzo ubogich

- Direct Persistence Layer
 - Encje
 - @Entity
 - @PrimaryKey
 - @SecondaryKey
 - ONE_TO_ONE
 - MANY_TO_ONE
 - ONE_TO_MANY
 - MANY_TO_MANY
 - Zapytania
 - PrimaryIndex i SecondaryIndexes
 - Key ranges
 - EntityJoin



Wszystkie obiekty na dysk

```
Environment env = new Environment(envHome, envConfig);  
EntityStore store = new EntityStore(env, "Books", storeCfg);  
inventory = store.getPrimaryIndex(String.class, Book.class);  
byAuthor = store.getSecondaryIndex(inventory, String.class,  
"author");  
byPublishingDate = store.getSecondaryIndex(inventory,  
Date.class,  
"publishingDate");
```



Zapytania i relacje

```
Date startDate = DATE_FORMAT.parse("1900/01/01");  
Date endDate = DATE_FORMAT.parse("2012/01/01");  
EntityCursor<Book> entities = books.byPublishingDate.entities(  
startDate, true, endDate, true);  
  
EntityJoin<String, Book> join = new EntityJoin<String, Book>(  
books.inventory);  
join.addCondition(books.byAuthor, "Macho Man");  
join.addCondition(books.byPublishingDate,  
DATE_FORMAT.parse("2010/01/30"));
```



Kilka dobrych rad

- Jeżeli twoje SQL sprowadzają się do
 - `SELECT * FROM tab ORDER BY col ASC;`
 - `SELECT * FROM tab WHERE col LIKE 'prefix%';`
 - `SELECT * FROM tab WHERE col >= A AND col <= B;`
 - `SELECT * FROM tab WHERE col1 = A AND col2 = B;`



Dobre linki

- <http://nosql-database.org/>
- <http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>
- <http://books.couchdb.org/relax/>
- <http://neo4j.org/>
- <http://www.oracle.com/technology/products/berkeley-db/index.html>
- <https://nosqleast.com/2009/>
- <http://bitbucket.org/kcrimson/nyac-code-snippets>

