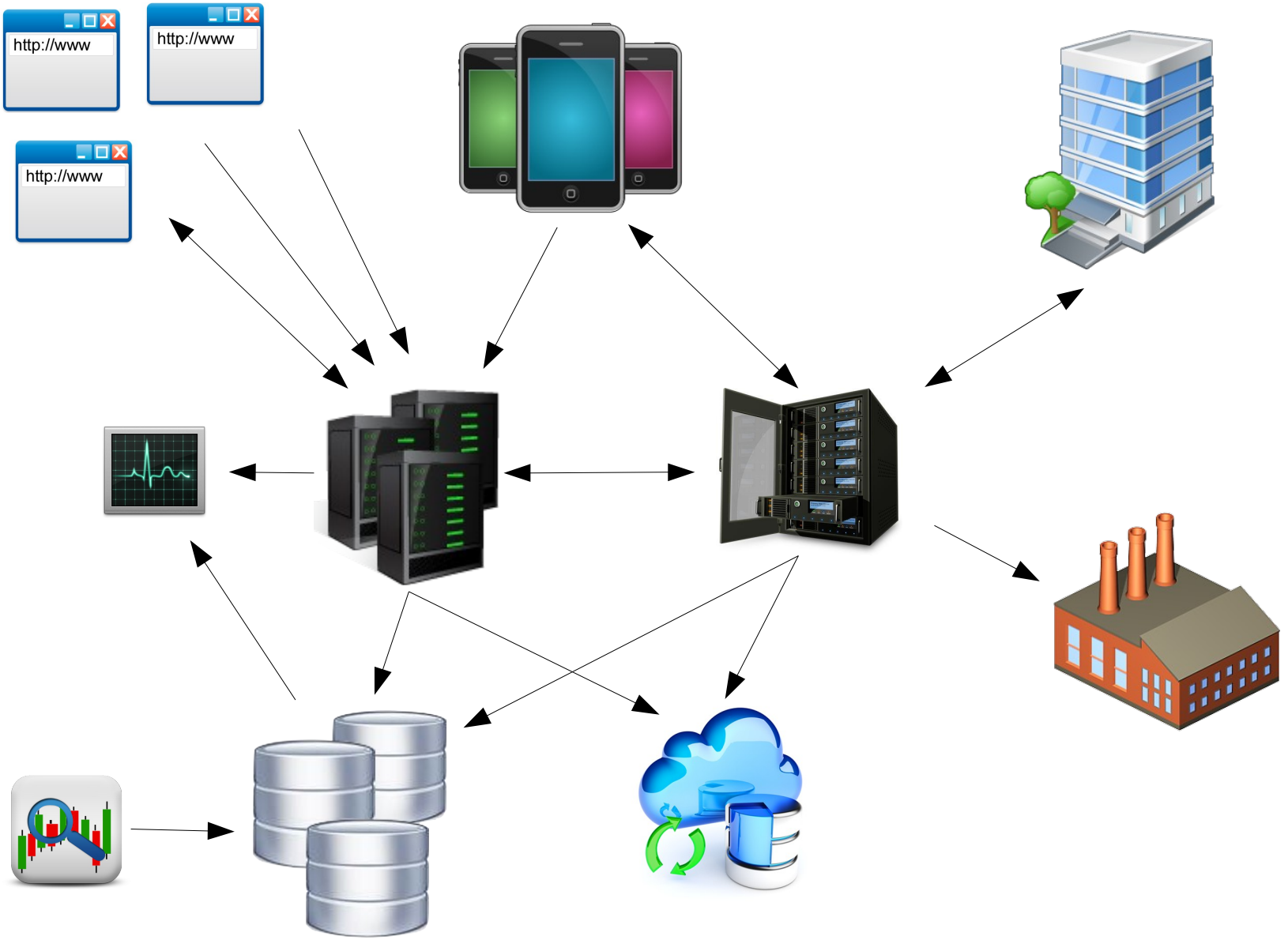
Aplikacje Reaktywne

i okolice

Aplikacje webowe kiedyś

Aplikacje webowe dzisiaj



Problemy

- Wydajność
  - Tradycyjne blokujące wołanie oraz obsługa to cios dla klienta i serwera
- Niezawodność
  - Części systemu mogą ulec awarii
  - Komunikacja jest niepewna
  - Zależymy od usług dostarczanych z zewnątrz

Rozwiązania wydajnościowe

- Nieblokujące wywołania
  - Biblioteki działające na poziomie TCP, HTTP, a nawet całego API
- Nieblokująca obsługa
  - Operacje dyskowe, długotrwałe przetwarzanie w tle
- Należy być konsekwentnym i nie symulować
  - Pomaga nam system operacyjny (Linux), języki programowania oraz ich biblioteki

Nie-do-końca-Rozwiązania komunikacyjne

- Abstrakcje takie jak RPC

  Z założenia są „przezroczyste” – zdalny komponent jest traktowany jak lokalny obiekt

  - Są blokujące (ale nie tylko)
  - Dają złudne wrażenie niezawodności
  - Nie pozwalają zarządzać awaryjnymi komponentami

Głośne hasła

loosely coupled

non-blocking

highly-available

real-time

asynchronous

concurrent

fault-tolerant

push instead of pull

low latency

scalable

distributed

event-driven

high throughput

reactive

Podejście „reaktywne”

- Sterowane zdarzeniami (*event-driven*) systemy mogą być:
  - Skalowalne (*scalable*)
  - Zdolne do samoleczenia (*resilient*)
- Dzięki czemu użytkownik obserwuje krótki czas reakcji (*responsive*)

Te zasady zostały niedawno sformułowane w „manifeście epoki”

- Stanowią swojego rodzaju „koncepcyjny” wzorzec projektowy
- Istotą jest uwzględnianie powyższych cech od początku tworzenia aplikacji, na jej wszystkich poziomach

Na przykładzie...

- Język: Scala
  - Obiektowy
  - Funkcyjna składnia i filozofia (ale można zrezygnować)
    - Type-safety oraz immutability
  - Implementacja na JVM (są plusy i minusy)

The Future of Scala

- Future to asynchroniczna operacja, która zwraca wynik lub wyjątek
- Wykonuje się w kontekście wątku, najczęściej z puli
- Obecność w Javie, ale tylko blokujące czekanie na wynik
- Scala dodaje operacje (prawie) monadyczne
  - Na Future można rejestrować przekształcenia i callbacki do (asynchronicznego) wykonania
  - Funkcyjna notacja Scali pomaga to zwięźle zapisać

Na przykładzie, dalej...

- Framework: Akka
  - „Bo (niskopoziomowa) współbieżność jest trudna”
  - Podstawowy byt: „jednowątkowy” Aktor
  - Komunikacja za pomocą przesyłania wiadomości pomiędzy aktorami
  - Wysyłanie (z reguły) nie jest blokujące: wiadomości są buforowane w „skrzynce odbiorczej”

Akka dokładniej

- Możliwe jest przezroczyste rozproszenie na wiele maszyn
  - Automatyczna serializacja i deployment za pomocą pliku konfiguracyjnego
- Podstawowa wersja nie kontroluje typów komunikatów
  - Prostota i elastyczność, ale Scala i pattern-matching pozwala łatwo decydować, co zrobić

Akka reaktywnie

- Event-driven:

  Komunikaty opisują zdarzenia, które są współbieżnie przetwarzane

- Scalable:

  Dobrze napisana aplikacja z łatwością skaluje się na większą liczbę rdzeni, a także większą liczbę maszyn

Akka reaktywnie

- Resilient:

  Zakładamy, że aktor-odbiorca może stać się niedostępny w dowolnej chwili

  - Z powodu wewnętrznego błędu logicznego, awarii maszyny lub problemu z siecią
  - Wysłanie wiadomości nie gwarantuje jej dostarczenia, a tym bardziej pomyślnego przetworzenia
  - Programista musi tak projektować przepływ komunikatów, aby uwzględnić te ograniczenia

Akka reaktywnie

- Responsive:

  Wiemy, że nieblokująca architektura pozwala efektywnie wykorzystać CPU

  - Programista nie jest ograniczony jednym wątkiem – może dzielić dostępną moc pomiędzy różne części aplikacji mające różne wymagania
  - Celem jest minimalizacja opóźnień tam, gdzie to najbardziej pożądane

Resilient + responsive

- Co zrobić gdy programista jednak się pomyli...
  - Czasem blokująca operacja?
    - „Bulkheading” (jak w tankowcach)
    - Tworzymy ograniczone pule wątków z których korzystają aktorzy-workerzy
  - Czasem błąd logiczny?
    - „Let it crash” (filozofia Akki, wzięta z Erlangu)
    - Rodzic aktora zostanie poinformowany i odpowiednio zareaguje

Resilient + responsive

- Podejście reaktywne zakłada dzielenie aplikacji na części mogące działać niezależnie
  - Czasowa niedostępność pewnych funkcji nie musi oznaczać zatrzymania systemu
  - Akka wymusza „hierarchię nadzoru”

    Jest to wysokopoziomowa, deklaratywna obsługa błędów, w której instruujemy framework jak postępować z zawodnymi aktorami

Zastosowania

- Middleware – koordynacja aktorów i komunikatów
  - dynamiczna topologia
  - przetwarzanie w tle
- Życiowe przykłady
  - Gry internetowe, zakłady bukmacherskie
  - Finanse, telekomunikacja
  - Media społecznościowe (Twitter, LinkedIn)
    - Indeksowanie, wyszukiwanie z wielu źródeł
    - Aktualizacje dla użytkownika

Single page web applications

Plan prezentacji

- Single page web application (w skrócie SPA)
- AngularJS

SPA - charakterystyka

- aplikacja w przeglądarce
- strona pobierana raz
- mnóstwo logiki po stronie klienta
- strona nie przeładowywuje się
- dane (JSON, asynch, PUSH)

SPA - cecha nr 1

SPA potrafi renderować się jak aplikacja desktopowa

SPA - cecha nr 2

SPA potrafi reagować jak aplikacja desktopowa

SPA - cecha nr 3

SPA potrafi powiadomić użytkownika o stanie aplikacji jak aplikacja desktopowa

SPA - cecha nr 4

Wysoka dostępność

SPA - cecha nr 5

Cross-platform’owość

SPA w Javascript'cie vs Flash i Java

- nie wymaga wtyczek
- lżejszy
- jeden język po stronie klienta
- jednolity wygląd strony

SPA w Javascript'cie

- przeglądarka jest najczęściej używaną aplikacją
- Javascript stał się szybki
- ewoluowanie Javascriptu (zaawansowane feature'y)
- wdrożenie strony w Javascripcie jest trywialne
- Javascript w całym stosie technologicznym aplikacji

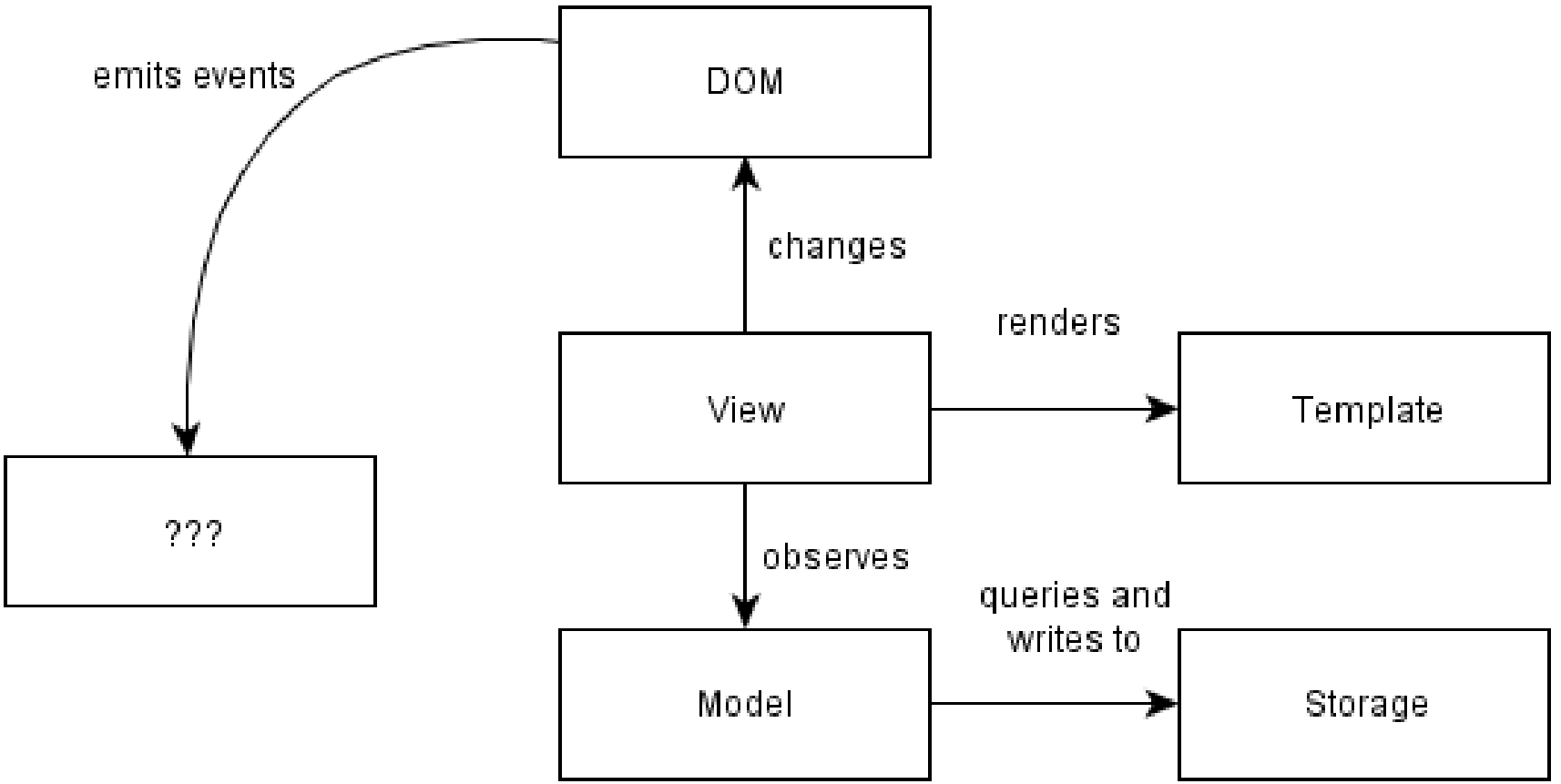
Podsumowanie: SPA - zalety

- lepszy “User Experience”
- wyższa wydajność
- dane muszą być dostępne przez API

Podsumowanie: SPA - wady

- duży “load” na początku
- duplikacja kodu
- potrzebny wystarczająco mocny klient
- Javascript
- wycieki pamięci

Architektura



- framework do tworzenia SPA
- prace od 2009 roku
- wspierany przez Google
- projekt open-source (lic. MIT)

Filozofia AngularJS'a

1. **programowanie deklaratywne** (UI, łączenie komponentów oprogramowania), **programowanie imperatywne** (logika biznesowa)
2. oddzielenie manipulowania DOM'em od logiki aplikacji
3. pisanie kodu aplikacji tak samo ważne, jak pisanie testów
4. oddzielenie warstwy klienckiej od warstwy serwerowej

AngularJS - feature 1

Dwukierunkowe wiązanie danych

(ang. two way data-binding)

AngularJS - feature 2

Szablony

(ang. templates)

AngularJS - feature 3

MVC

AngularJS - feature 4

Wstrzykiwanie zależności

(ang. dependency injection)

AngularJS - feature 5

Dyrektywy

(ang. directives)

Technologie, narzędzia

1. yeoman
2. bower
3. grunt
4. bootstrap
5. jasmine
6. karma
7. underscore.js

Standardy komunikacji serwer-klient

1. WebSocket
2. Server-Sent Events w HTML5
3. Aplety Java’owe, wtyczki Flash’owe
4. AJAX
5. **SocketIO**

Warstwa przechowywania

Cassandra

Wymagania

- Efektywność (opóźnienie, przepustowość)
- Transakcyjność
  - atomowość, współbieżność, izolacja, trwałość
- Funkcjonalność
- Skalowalność
- Niezawodność
- Bezpieczeństwo
- Replikacja

Być może nie potrzebujemy wszystkiego

- Efektywność (opóźnienie, przepustowość)
- Transakcyjność
  - atomowość, współbieżność, izolacja, trwałość
- Funkcjonalność
- Skalowalność
- Niezawodność
- Bezpieczeństwo
- Replikacja

MySQL

- Transakcyjność
- Funkcjonalność
- Niezawodność
- Bezpieczeństwo
- Skalowalność

Redis

- Baza typu klucz – wartość
- Udoskonalony memcached a raczej memcachedb
- Efektywny
- Jednowątkowy
- Mimo wszystko może zgubić dane

Redis

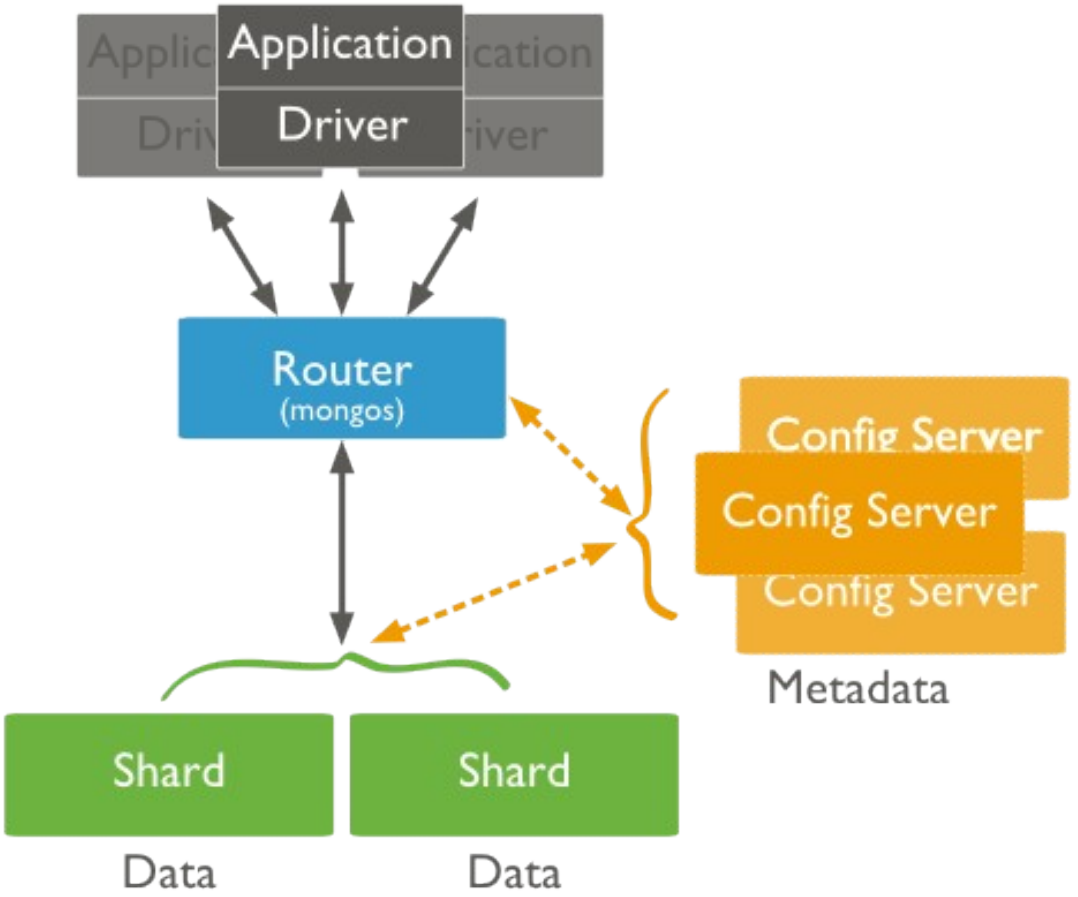
- Efektywność
- Transakcyjność
- Funkcjonalność
- Niezawodność
- Skalowalność
- Bezpieczeństwo

Szybki i funkcjonalny, ale nie gwarantuje bezpieczeństwa danych i słabo się skaluje.

MongoDB

- Baza przechowuje dokumenty (Document-oriented storage)
- Dopuszcza modyfikacje dokumentów
- Szerokie zastosowania
- Ograniczona transakcyjność
- Najpopularniejsza baza NOSQL

MongoDB


Application
Driver
Router
(mongos)
Config Server
Config Server
Config Server
Metadata
Shard
Shard
Data
Data

MongoDB

- Efektywność
- Funkcjonalność
- Niezawodność
- Skalowalność
- Bezpieczeństwo

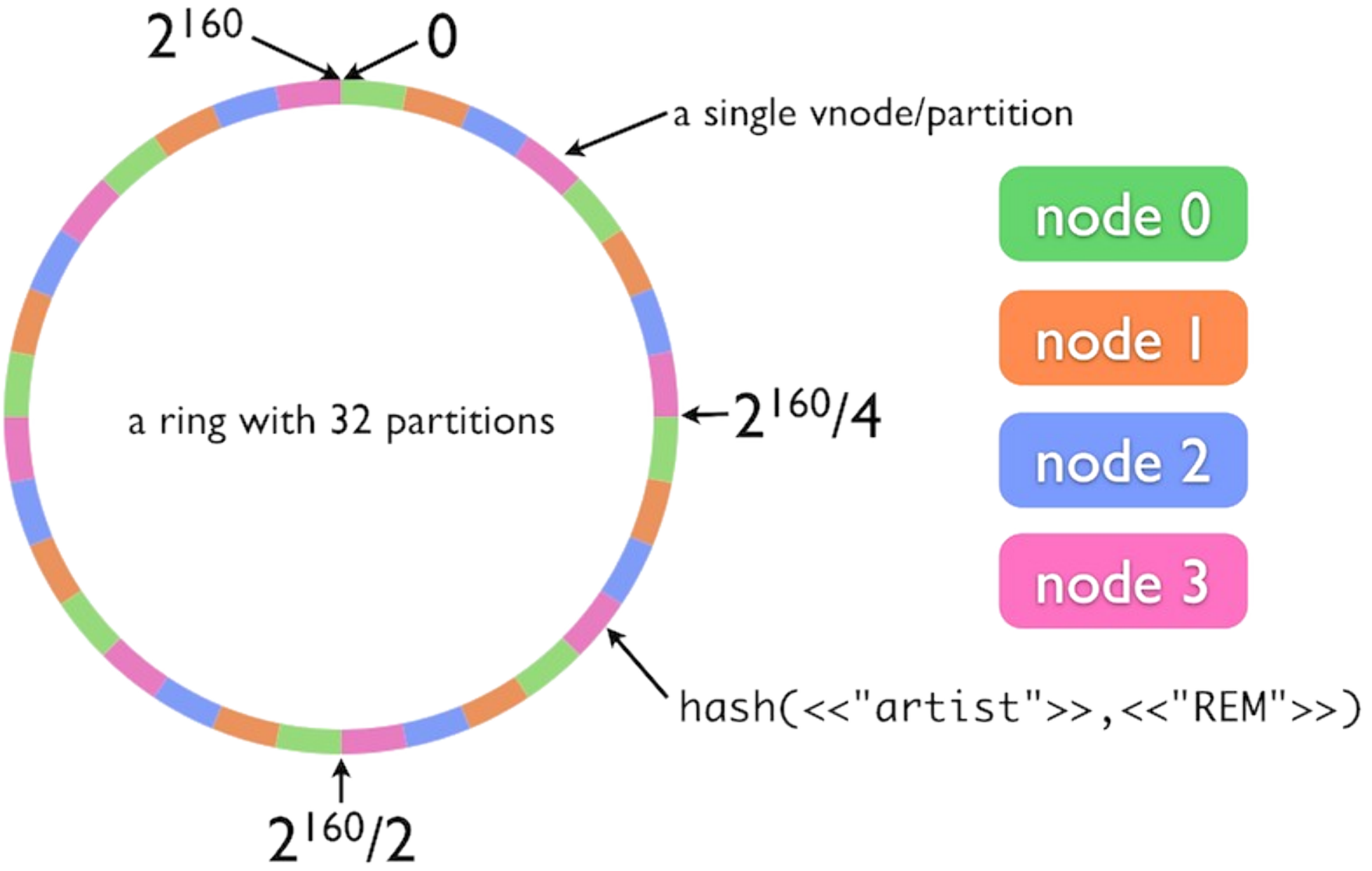
- Transakcyjność

Oferuje skalowalność i bezpieczeństwo, jednak kosztem transakcyjności

Riak

- Zaprojektowany pod kątem skalowalności i replikacji
- Wysoki poziom gwarancji (Unexpected is expected)
- Ograniczony zestaw operacji
- Brak atomowych operacji – trzeba rozwiązywać konflikty

Riak


2^{160}
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
$2^{160}/4$
hash(<<"artist">>,<<"REM">>)
$2^{160}/2$

Riak

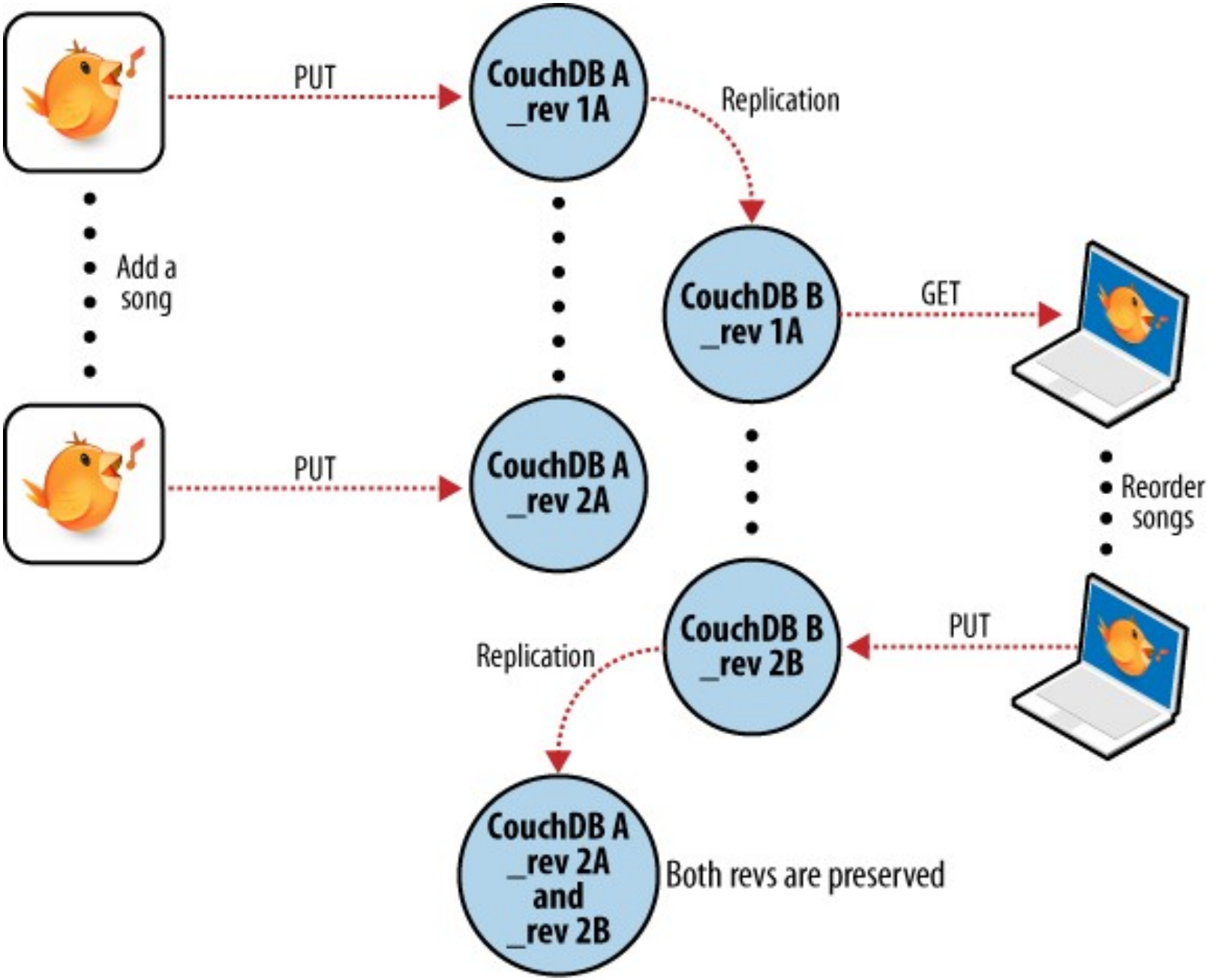
- Efektywność
- Niezawodność
- Skalowalność
- Bezpieczeństwo
- Transakcyjność
- Funkcjonalność

Vector clocks może się okazać niewygodnym rozwiązaniem. Każda modyfikacja wymaga odczytania całej wartości.

CouchDB

- Nigdy nie modyfikuje danych
- Odczyt nigdy nie jest wstrzymywany
- Możemy wysłać skrypt do wykonywania odczytów i „modyfikacji”
- Konflikty rozwiązujemy podobnie jak w GIT

CouchDB


PUT
CouchDB A
_rev 1A
Replication
Add a
song
CouchDB B
_rev 1A
GET
PUT
CouchDB A
_rev 2A
Reorder
songs
CouchDB B
_rev 2B
PUT
Replication
CouchDB A
_rev 2A
and
_rev 2B
Both revs are preserved

CouchDB

- Niezawodność
- Skalowalność
- Bezpieczeństwo
- Funkcjonalność
- Efektywność

Często zmieniające się dane mogą powodować problemy wydajnościowe

Źródła

- http://www.reactivemanifesto.org/
- http://highscalability.com/blog/2013/5/8/typesafe-interview-scala-akka-is-an-iaas-for-your-process-ar.html
- http://net.tutsplus.com/tutorials/javascript-ajax/5-awesome-angularjs-features
- http://angularjs.org/
- http://singlepageappbook.com/goal.html
- Michael S. Mikowski, Josh C. Powell. “Single Page Web Applications, JavaScript end-to-end”
- http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
- http://redis.io/documentation
- http://docs.mongodb.org/manual/
- http://docs.basho.com/riak/latest/theory/concepts/
- http://docs.couchdb.org/en/latest/intro/overview.html

Nowe Rejestracje?