

NoSQL: Riak

dr inż. Sebastian Ernst
Katedra Informatyki Stosowanej

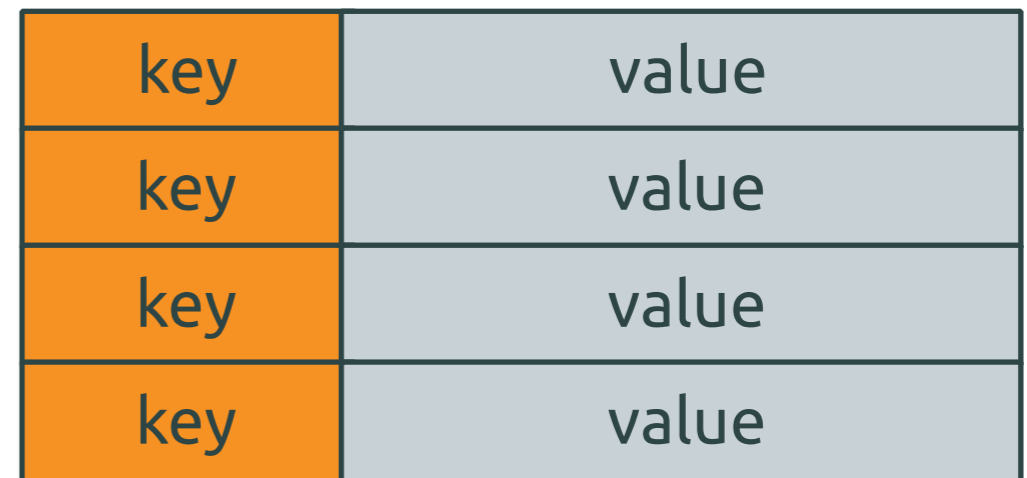
Twierdzenie CAP

- W przypadku rozdziału węzłów (partition), możliwe jest zachowanie jednej z dwóch cech:
 - spójności (consistency) – wszystkie węzły „widzą” dokładnie te same dane,
 - dostępności (availability) – cała funkcjonalność systemu bazodanowego jest osiągalna.

Riak

- Baza NoSQL typu klucz-wartość.
- Rozproszona, wszystkie węzły (ang. *nodes*) równoważne (obsługują zapis oraz odczyt).
- Hierarchia organizacji danych:
bucket – key – value

bucket



The diagram illustrates a bucket structure. It consists of a dashed rectangular border containing a table with four rows. Each row has two columns: the first column is labeled 'key' and has an orange background, and the second column is labeled 'value' and has a light blue background.

key	value
key	value
key	value
key	value

Dostępność w Riak

- Każdy węzeł może obsłużyć dowolne żądanie.
- W przypadku awarii (odłączenia) węzła, sąsiedni węzeł przejmuje jego operacje zapisu/modyfikacji danych, i przekazuje mu je po tym gdy znów stanie się dostępny (tzw. *hinted handoff*).

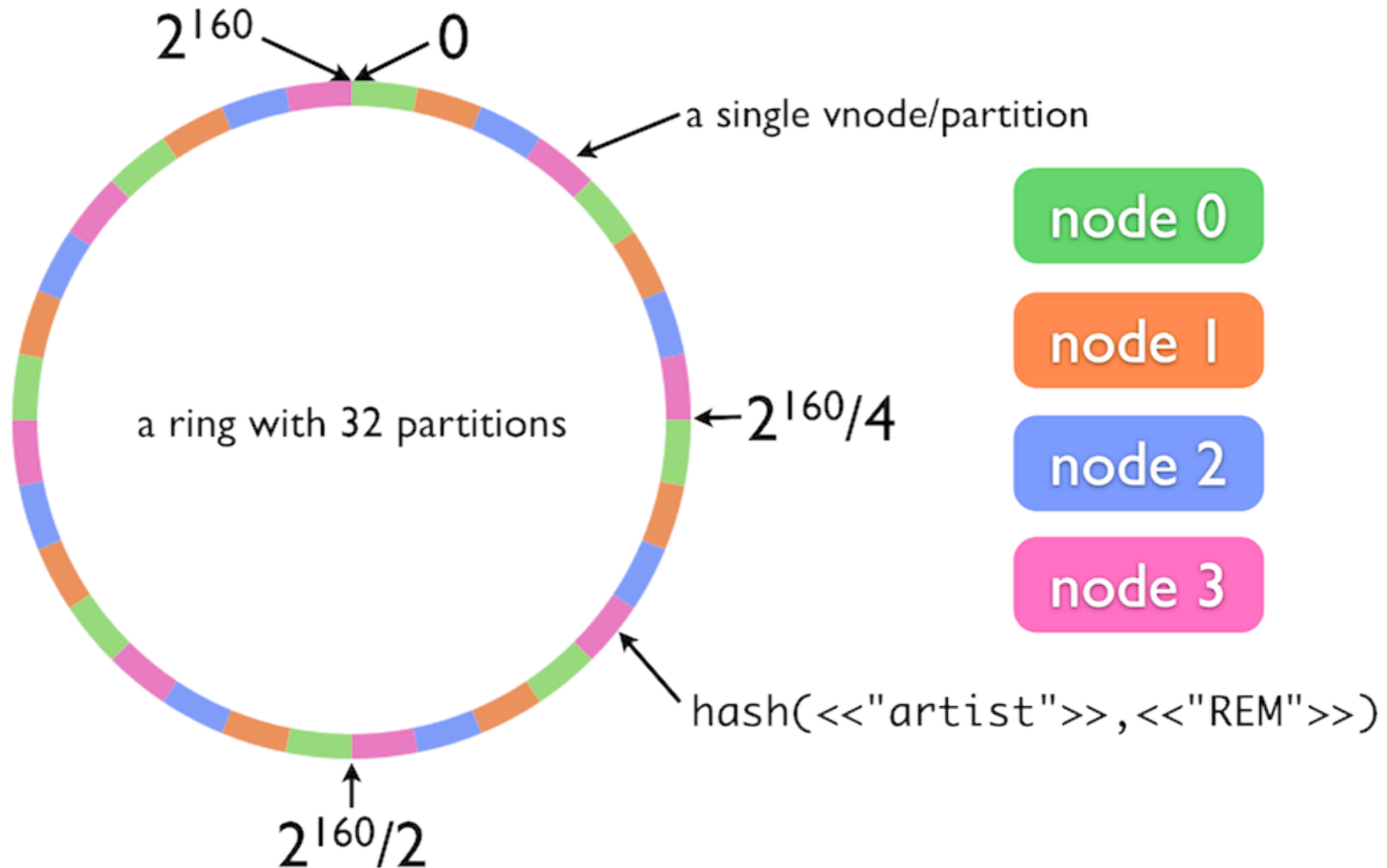
Rozproszenie danych

- Bazy relacyjne źle skalują się horyzontalnie – stosuje się m.in. metodę zwaną *sharding* – podział danych wg. atrybutów na węzły.
- Kosztowne we wdrożeniu i utrzymaniu.
- Może prowadzić do powstania tzw. *hot spots*.

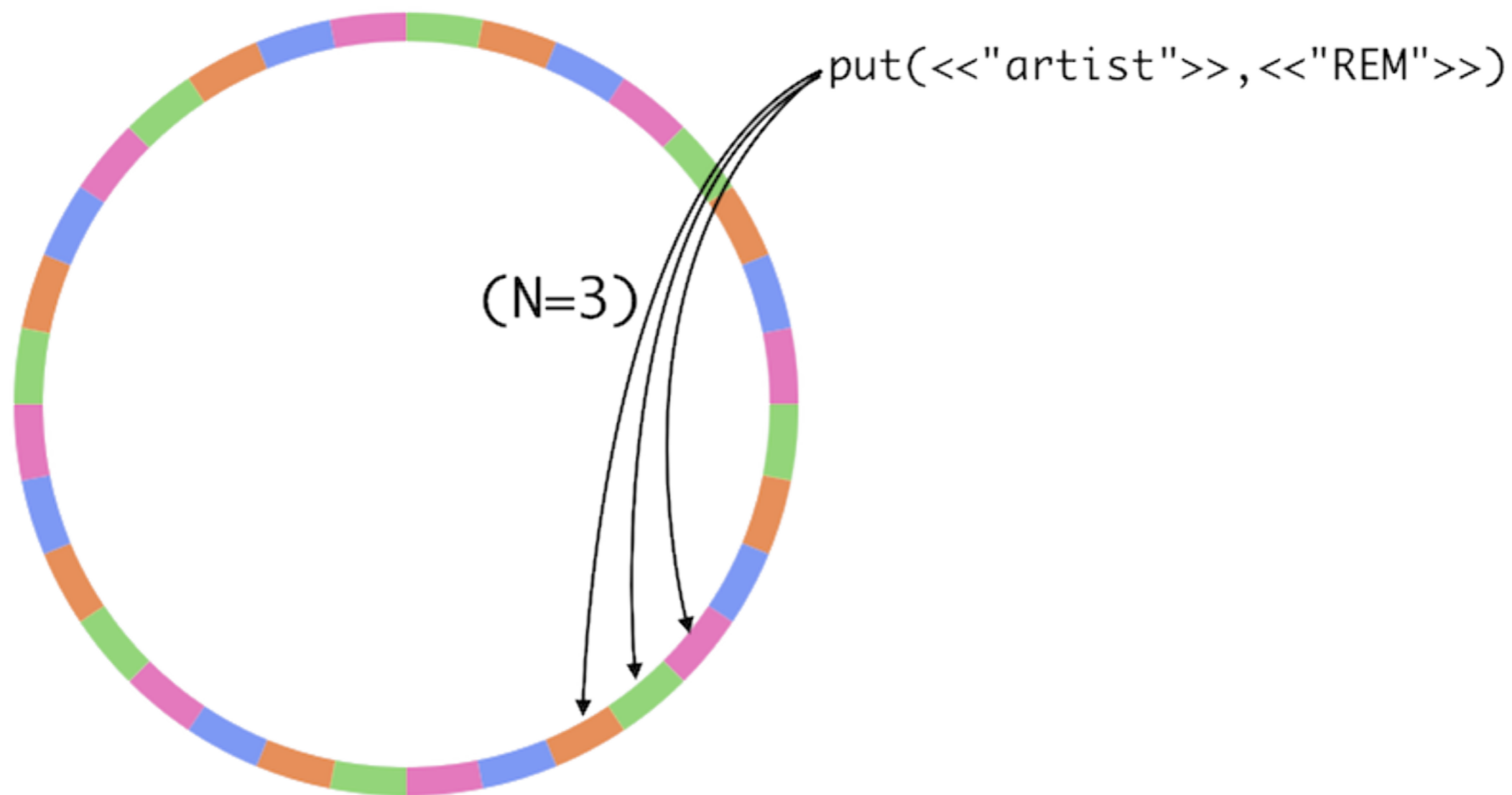
Model danych Riak

- Dane (klucz+wartość) – nazywane obiektami – pogrupowane w tzw. wiadrach (*buckets*).
- Pary wiadro+klucz są hashowane SHA1 – powstają skróty o długości 160 bitów.
- Przestrzeń kluczy jest dzielona na partycje (domyślnie 64) – każda partycja odpowiada za zakres zawierający $2^{160}/n$ wartości.

Model danych Riak



Replikacja w Riak



Spójność

- Eventual Consistency – dane *kiedyś* osiągną stan spójny.
- W przypadku wystąpienia konfliktów, wybierana jest najnowsza wersja przy pomocy mechanizmu *vector clocks*.
- Zapytania mogą wymagać określonego kworum węzłów przez określenie parametru r (dla odczytu) oraz w (dla zapisu).
- *Read repair*: jeżeli węzeł zwraca nieaktualną wersję (lub komunikat *not found*), dane w nim są automatycznie naprawiane.

Wyszukiwanie w Riak

- Prosty model danych oznacza brak zbiorów, transakcji czy złączeń (bo nie ma kolumn ani wierszy).
- Dane można przeszukiwać poprzez:
 - **riak search**: rozproszony silnik wyszukiwania pełnotekstowego,
 - **secondary indexing (2i)**: możliwość przypisywania do obiektów tagów, które potem mogą stanowić element zapytania,
 - **MapReduce**: definiowanie procedur przy pomocy JavaScript lub Erlanga.

Riak Search

- Rozproszona, pełnotekstowa wyszukiwarka.
- Wsparcie dla różnych formatów (tekst, JSON, XML, Erlang) oraz analizatorów (whitespace, integer, noop).
- Wildcards, zakresy, AND/OR/NOT, grupowanie, prefiksy, wyszukiwanie w okolicy, *term boosting*.

Riak Search

- Stosujemy wtedy, gdy:
 - zbieramy i parsujemy treści takie jak artykuły, blogi, itd.,
 - chcemy indeksować dane JSON,
 - potrzebujemy szybkiego dostępu i zarazem bogatych możliwości odpytywania.
- Nie stosujemy, gdy wystarczyłoby tagowanie danych, dla danych binarnych lub gdy wymagamy pewnej spójności (wyszukiwarka nie wspiera *read repair*).

Riak Search

- Stosujemy wtedy, gdy:
 - zbieramy i parsujemy treści takie jak artykuły, blogi, itd.,
 - chcemy indeksować dane JSON,
 - potrzebujemy szybkiego dostępu i zarazem bogatych możliwości odpytywania.
- Nie stosujemy, gdy wystarczyłoby tagowanie danych, dla danych binarnych lub gdy wymagamy pewnej spójności (wyszukiwarka nie wspiera *read repair*).

Riak Search – użycie

- Można skorzystać z polecenia shellowego *search-cmd*, np.:
bin/search-cmd search books "title:\\"See spot run\\""
- Dostępny jest też interfejs Solr:
curl "http://localhost:8098/solr/books/select?start=0&rows=10000&q=prog"*
- Wykorzystanie jako wstępnego filtra dla MR.

```
{ "inputs" : {  
    "bucket" : "mybucket",  
    "query" : "foo OR bar"  
  },  
  "query" : ... }
```

Riak Search – użycie

- Wyszukiwanie terminów w określonym sąsiedztwie:
"See spot run"~20
- Wyszukiwanie zakresów:
"field:[red TO rum]"
- Zwiększanie priorytetu terminów:
*red^5 **OR** blue*
- Operacje logiczne:
*red **AND** blue **AND NOT** yellow*
+red +blue -yellow
- Grupowanie: *(red **OR** blue) **AND NOT** yellow*

Secondary Indexes (2i)

- Udogatępniają możliwość tagowania obiektów.
- Dostępne typy: liczby całkowite i ciągi znaków (również binaria).
- Dokładne dopasowanie lub zakres wartości.
- Możliwa paginacja, streaming.
- Również można wykorzystać jako wejście do MR.

Secondary Indexes (2i)

- Używamy, gdy:
 - chcemy szukać w oparciu o coś innego niż para wiadro/klucz,
 - przechowujemy bloby i chcemy mieć możliwość sprawnego wyszukiwania.
- Nie używamy, gdy:
 - mamy >512 partycji,
 - potrzebujemy czegoś innego niż dokładne dopasowanie lub zakresy,
 - wymagamy złożonych zapytań.

Secondary Indexes (2i)

- Dodawanie obiektu z 2i:

```
curl -X POST \  
-H 'x-riak-index-twitter_bin: jsmith123' \  
-H 'x-riak-index-email_bin: jsmith@basho.com' \  
-d '...user data...' \  
http://localhost:8098/buckets/users/keys/john_smith
```

- Odpytywanie:

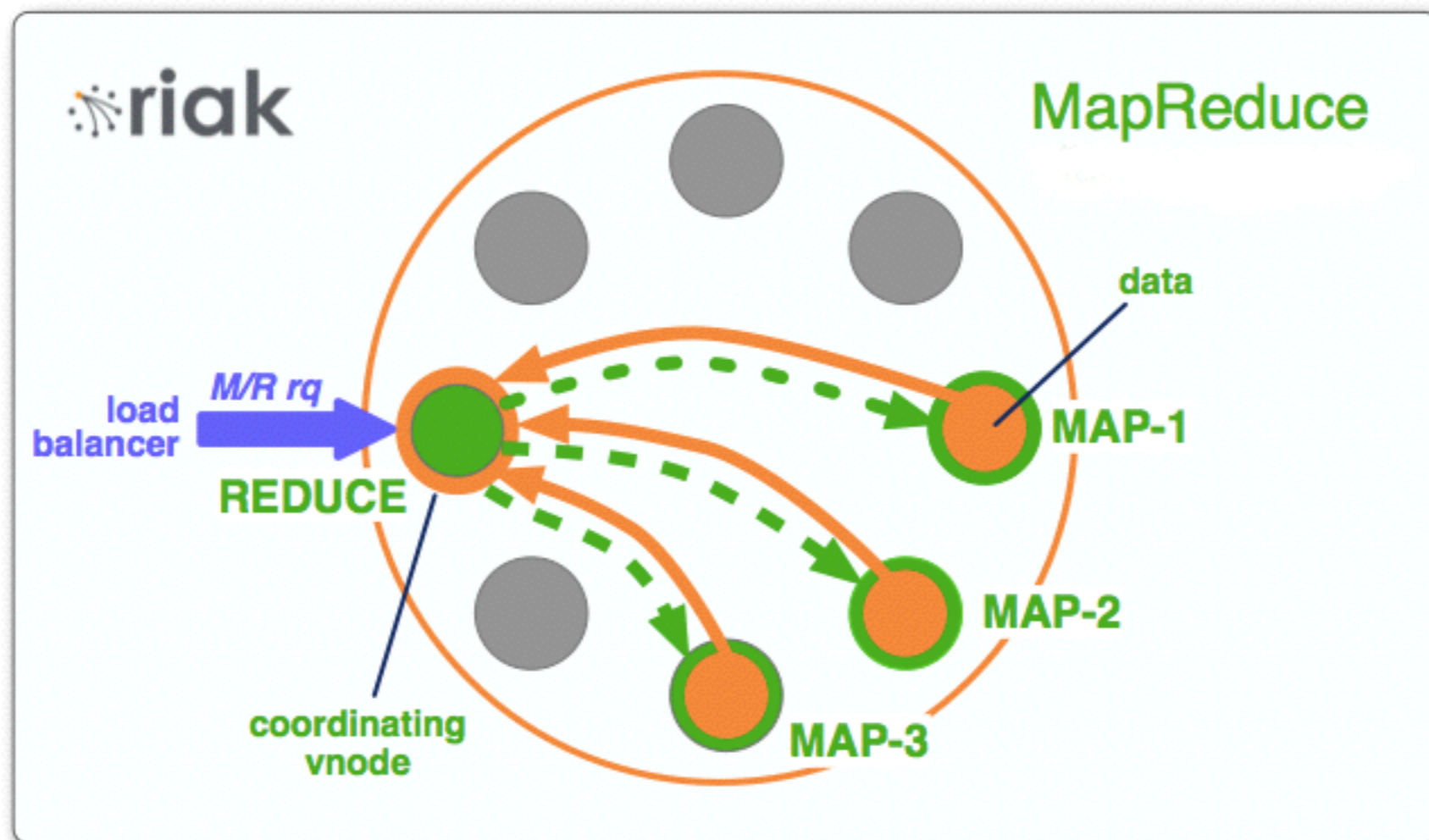
```
curl localhost:8098/buckets/users/index/twitter_bin/  
jsmith123
```

MapReduce w Riak

- Faza *map* wykonywana na węzłach posiadających określone dane, *reduce* – na węźle któremu zlecono zadanie.
- Używamy MapReduce:
 - gdy znamy zbiór par wiadro/klucz do przetworzenia,
 - gdy wyniki mają zawierać wartości bądź ich fragmenty (a nie same klucze),
 - gdy wymagana jest większa elastyczność niż ta w wyszukiwarce czy 2i.
- Nie używamy do przeglądu całego wiadra, lub gdy chcemy deterministycznego czasu wyszukiwania.

MapReduce w Riak

- Zapytanie MR zawiera listę wejść (pary wiadro/klucz) oraz listę faz (definicje funkcji *map*, *reduce* i *link*).



MapReduce: przykład

```
$ curl -X POST -H "content-type: application/json" \  
    http://localhost:8098/mapred --data @-<<\EOF  
{ "inputs": [ [ "alice", "p1" ], [ "alice", "p2" ], [ "alice", "p5" ] ]  
, "query": [ { "map": { "language": "javascript", "source": "  
  
" } }, { "reduce": { "language": "javascript", "source": "  
  
" } } ] ]  
EOF
```

MapReduce: przykład

```
$ curl -X POST -H "content-type: application/json" \  
    http://localhost:8098/mapred --data @-<<\EOF  
{"inputs":[[ "alice", "p1"], ["alice", "p2"], ["alice", "p5" ]]  
,"query": [{"map": {"language": "javascript", "source": "  
  
function(v) {  
    var words = v.values[0].data.toLowerCase().match( '\\w*', 'g' );  
    var counts = [];  
    for(var word in words)  
        if (words[word] != '') {  
            var count = {};  
            count[words[word]] = 1;  
            counts.push(count);  
        }  
    return counts;  
}  
"}}, {"reduce": {"language": "javascript", "source": "  
  
"} } ] }  
EOF
```

MapReduce: przykład

```
$ curl -X POST -H "content-type: application/json" \  
    http://localhost:8098/mapred --data @-<<\EOF  
{"inputs":[["alice","p1"],["alice","p2"],["alice","p5"]]  
,"query":[{"map":{"language":"javascript","source": "  
"  
"}}, {"reduce":{"language":"javascript","source": "  
"  
function(values) {  
    var result = {};  
    for (var value in values) {  
        for(var word in values[value]) {  
            if (word in result)  
                result[word] += values[value][word];  
            else  
                result[word] = values[value][word];  
        }  
    }  
    return [result];  
}  
"  
"}]]]  
EOF
```

Przykład modeli danych

Zastosowanie	Klucz	Wartość
Sesja	ID sesji/użytkownika	Dane sesji
Reklamy	ID kampanii	Treść reklamy
Logi	Data	Zawartość logów
Dane z czujników	Data/czas	Parametry odczytów
Dane użytkowników	Login, e-mail, UUID	Atrybuty użytkownika
Treść	Tytuł/ID	Zawartość dokumentu