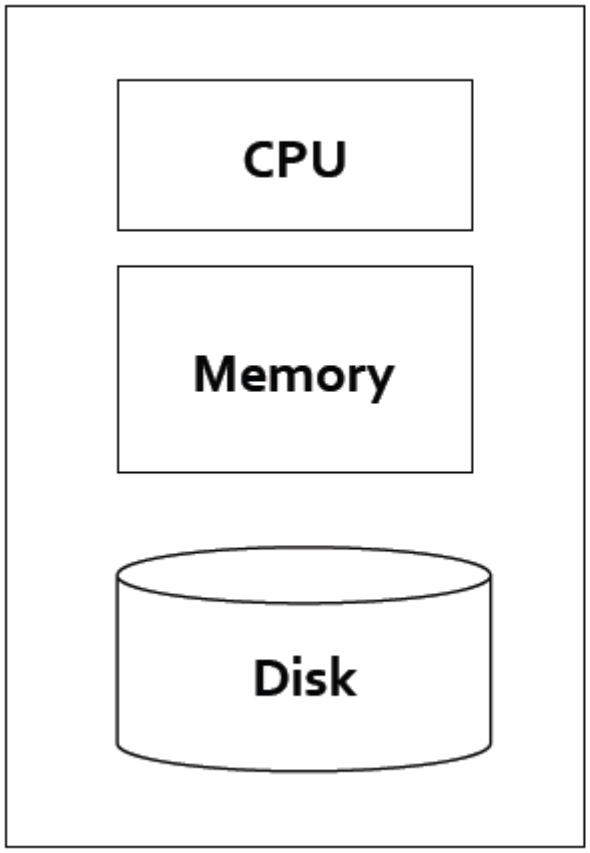
ALGORYTMICZNA I STATYSTYCZNA ANALIZA DANYCH

22/01/2015

WFAiS UJ, Informatyka Stosowana
II stopień studiów

Map-Reduce system

„Single-node” architektura



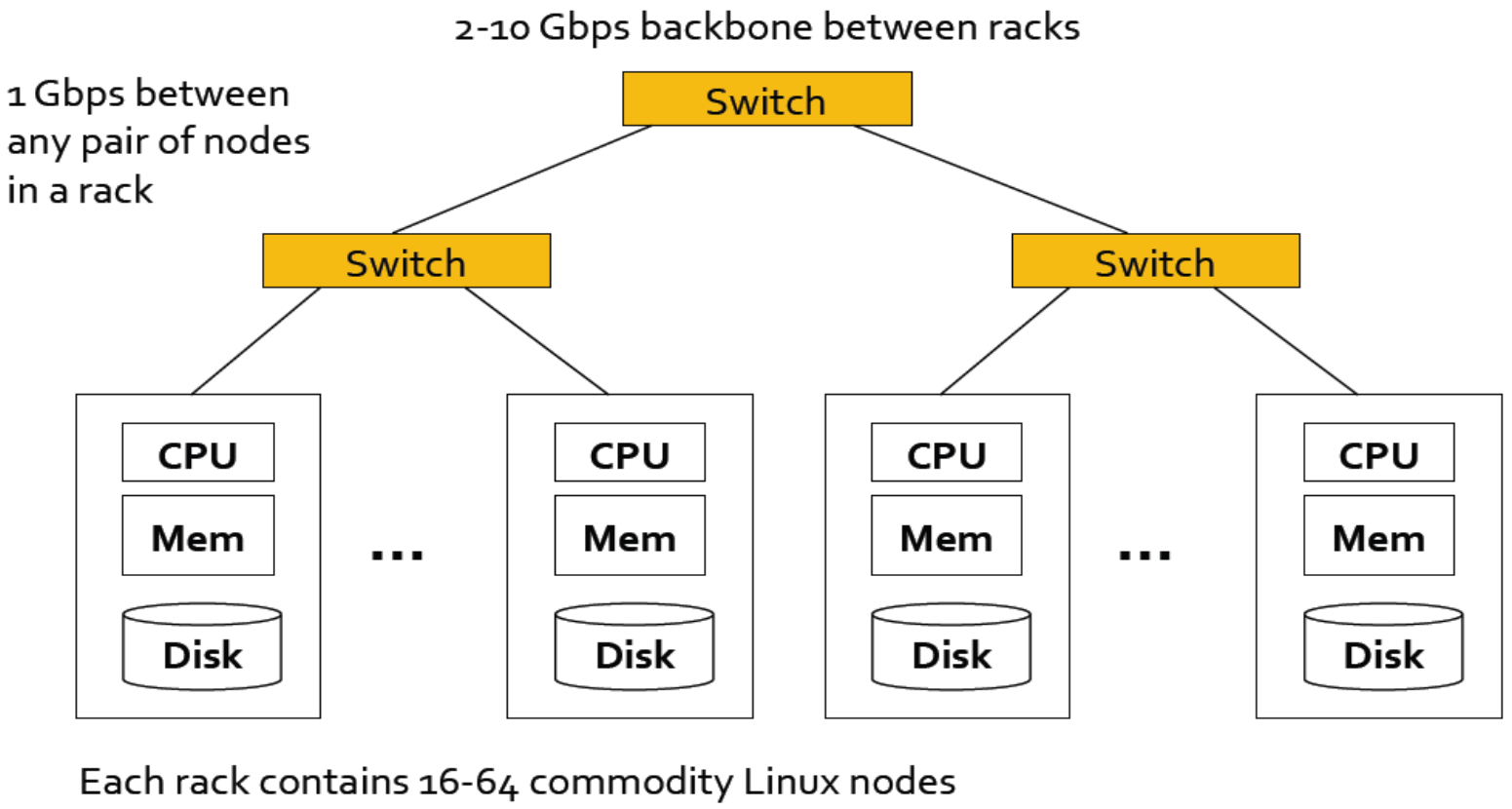
Machine Learning, Statistics

“Classical” Data Mining

Przykład Googla

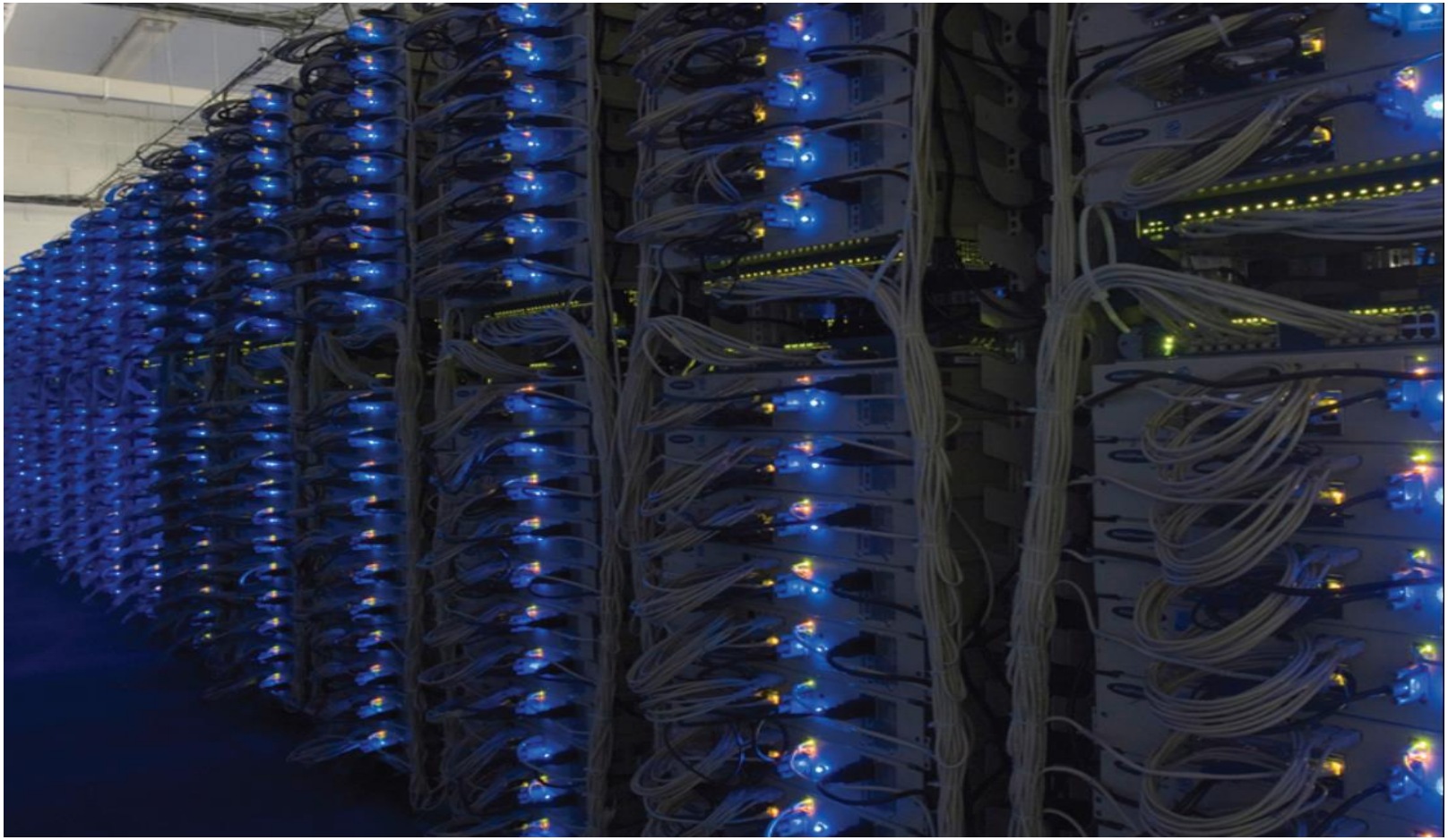
- 10 miliardów stron internetowych
- Średnia wielkość strony = 20kB
- 10 miliardów * 20 KB = 200 TB
- Disk read bandwidth = 50MB/sec
- Czas potrzeby na wczytanie = 4 miliony sec = 46+ dni
- Jeszcze dłużej aby procesować dane

Architektura klastra



W roku 2011 oszacowano że Google ma 1M procesorów

Klaster CPU



Wyzwania dla dużych klastrów

- „Node failures”
  - Pojedynczy procesor średnio działa 3 lata bez potrzeby resetu (1000 dni)
  - 1000 serverów w klastrze -> raz dziennie reset
  - 1M serwerów w klastrze -> 1000 reset/dzień
- W jaki sposób zapisywać dane na dysku aby zapewnić stabilność obliczeń mimo konieczności reset poszczególnych procesów?
- Jak sobie radzić ze stabilnością bardzo długich obliczeń?

Wyzwania dla dużych klastrów

- Problem transferu danych
  - Network bandwidth = 1Gps
  - Transfer 10 TB zajmuje średnio 1 dzień

- Pisanie kodu który jest zrównolegniony jest bardzo trudne. Raczej należy to rozwiązać algorytmicznie. Potrzebny prosty model który pozwala ukryć trudności takiego programowania

Map-Reduce

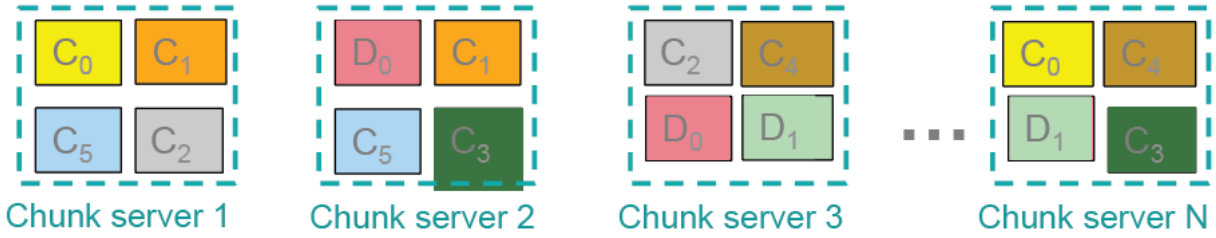
- Technika która rozwiązuje te problemy:
  - Dane są zapamiętane w kilku kopiach aby ograniczyć transfer oraz zapewnić stabilność dostępności
  - Przenieść obliczenia do danych aby zminimalizować transfer danych
  - Proste model programowania aby ukryć (wykorzystać dostępne narzędzia) trudności

Redundantny system przechowywania danych

- Rozproszony system plików
  - Goolgle GFS; Hadoop HDFS
- Typowe zastosowania
  - Duża liczba plików (100s of GB to TB)
  - Dane są rzadko zmieniane
  - Czytanie i updatowanie poprzez dołączenie

Rozproszony system plików

- Dane przechowywane w małych plikach rozproszonych po wielu maszynach
- Każdy plik przechowywany w kilku kopiach
  - Zapewnia niezawodność dostępu do danych



Chunk servers also serve as compute servers

Bring computation to data!

Map-Reduce

- Masz do przeanalizowania ogromny plik tekstowy
- Cel: policzyć ile razy dane słowo wystepuje w pliku
- Zastosowanie:
  - Np..przeanalizować logfile web serwera aby znaleść najbardziej popularne URL

Zadanie: ilość wystąpień słowa

- Przypadek 1:
  - Plik za duży aby go wczytać do pamięci, ale lista <słowo, ilość> się mieści w pamięci.
- Przypadek 2:
  - Nawet lista <słowo, ilość> nie mieści się w pamięci.
  - Words(doc.txt) | sort | uniq –c
    - Metoda „words” bierze plik danych i wypisuje słowa jedno na linie
    - W sposób naturalny można zrównoleglić

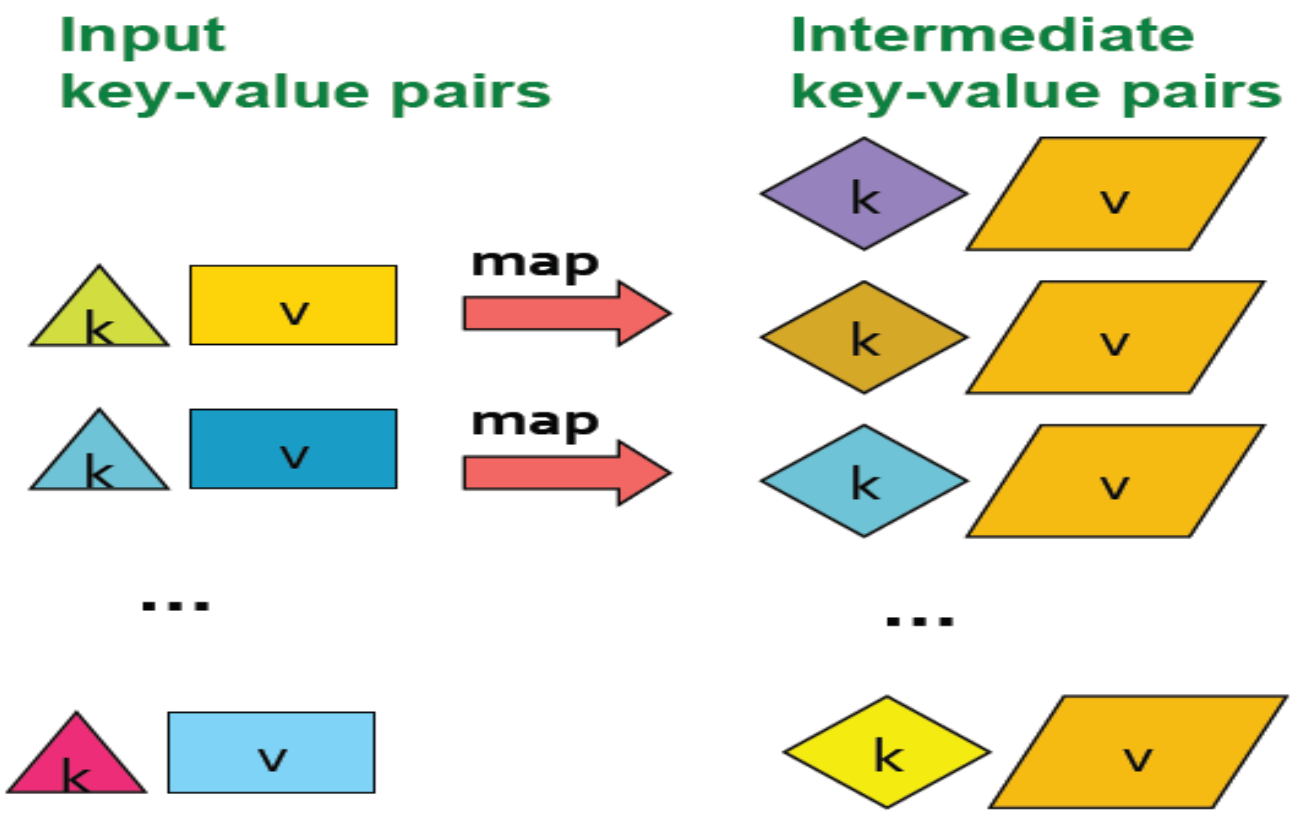
Map-Reduce

words(doc.txt) | sort | uniq –c

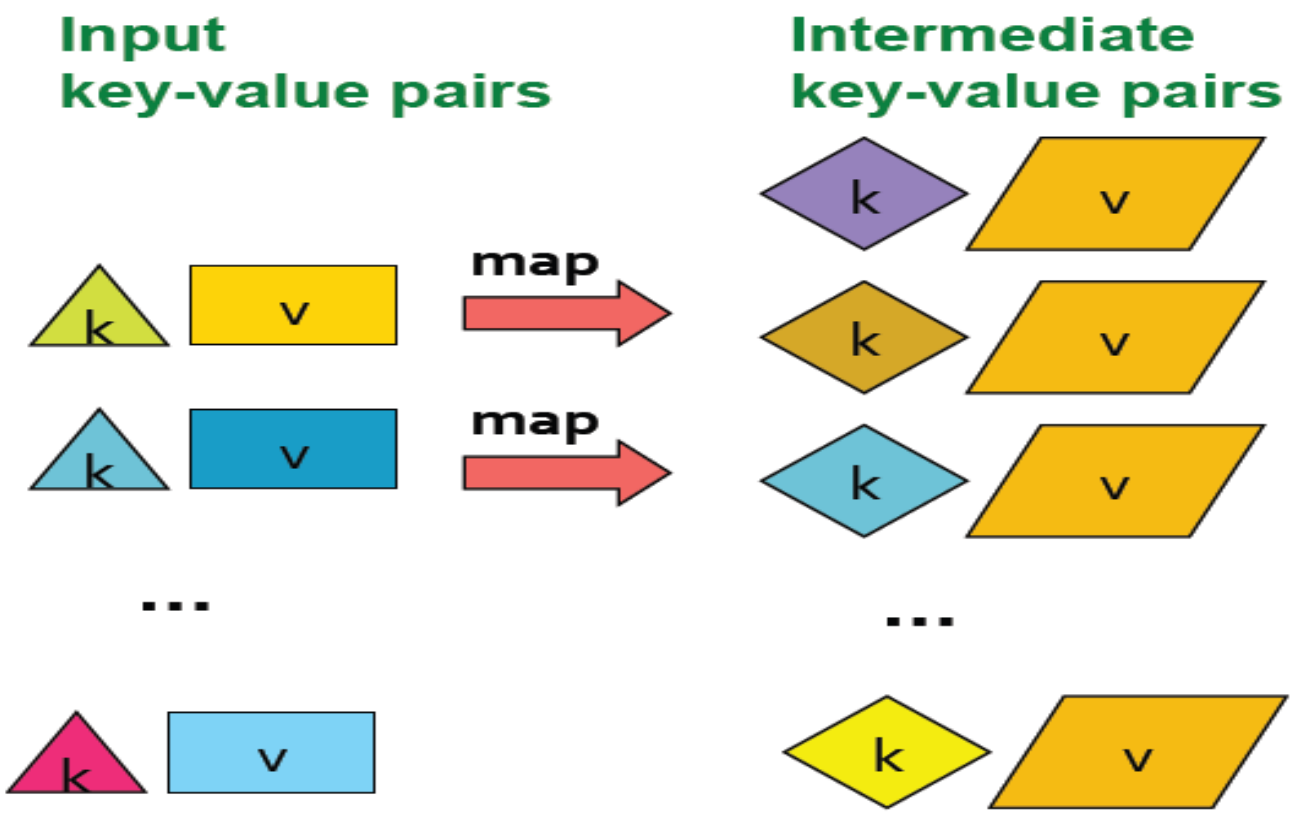
- **Map**
  - Przeglądaj plik imputowy
  - Wybierz informację która Cię interesuje
- **Pogrupuj wg. Klucza**
  - Posortuj i przekaż dalej
- **Reduce**
  - Poklastruj, przefiltruj, etc.
  - Wypisz wynik

Outline stays the same, **Map** and **Reduce** change to fit the problem

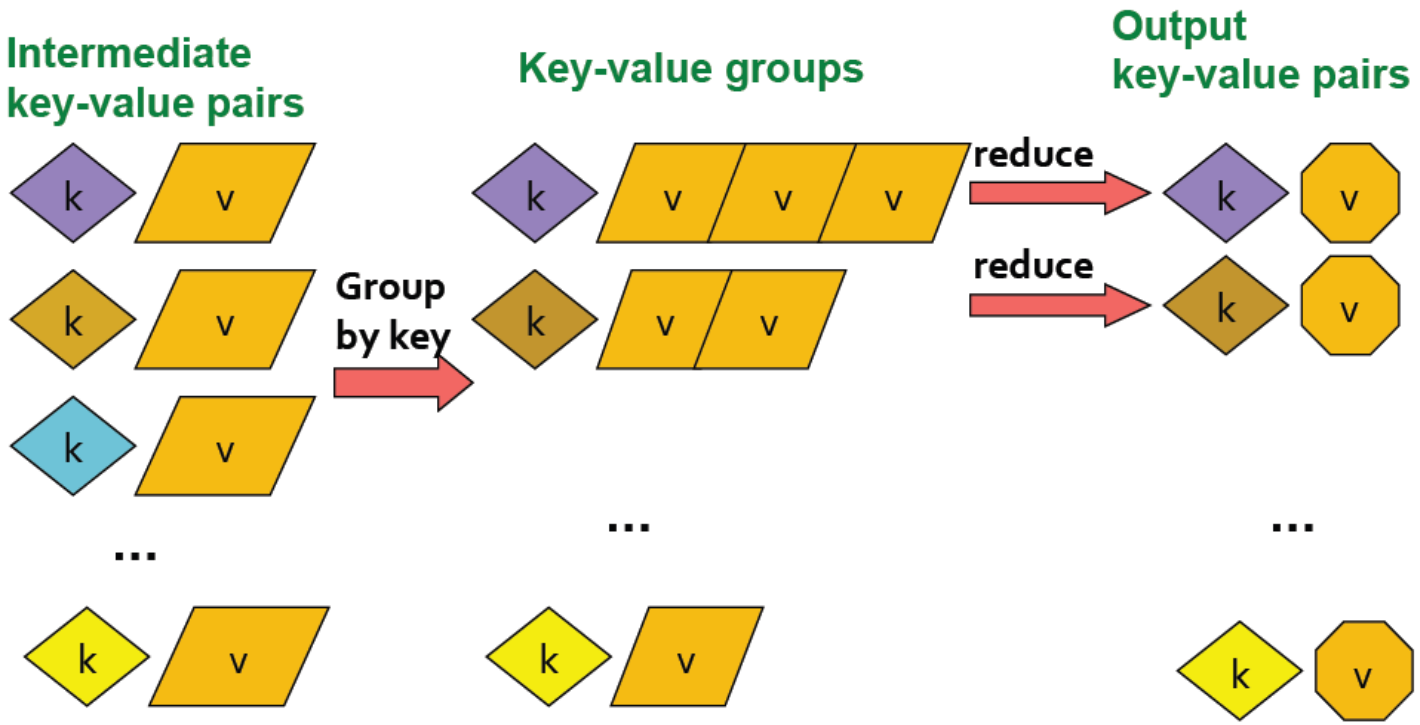
Map-Reduce: Map



Map-Reduce: Map



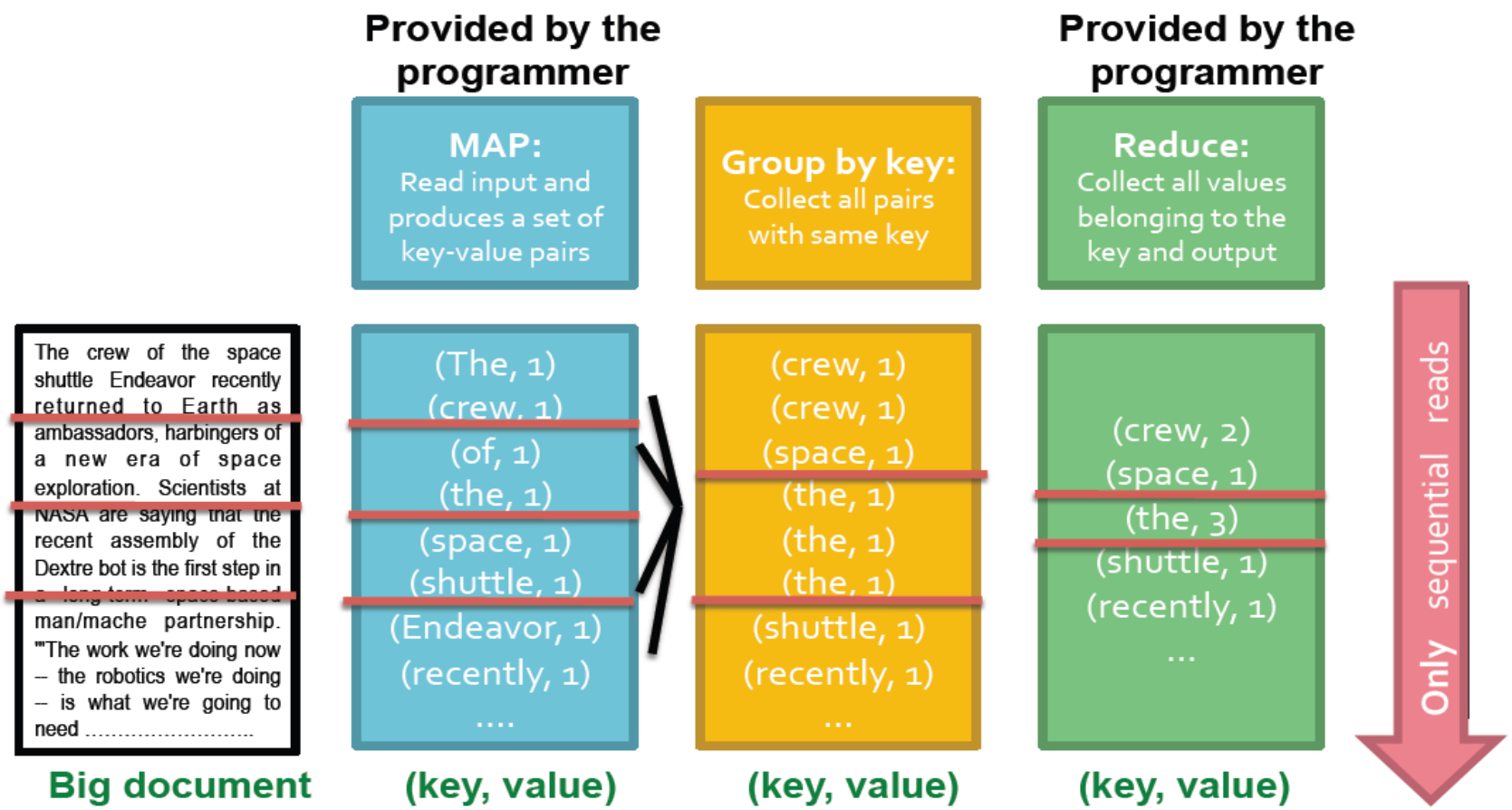
Map-Reduce: Reduce



Bardziej formalnie ….

- Input: zbiór par <klucz, wartość>
- Programista przygotowywuje dwie metody
  - Map(k,v) -> <k', v'>
    - Pobiera parę <klucz, wartość> i oddaje zbiór par <klucz', wartość'>
    - Map wywołana tylko raz dla każdej pary <klucz, wartość>
  - Reduce(k', <v'>*)-> <k', v">*
    - Wszysteki wartość v' z ta samą wartością klucza k' są „zredukowane" razem
    - Jest tylko jedna funkcja „Reduce" dla każdej wartości klucza

Map-Reduce: liczenie słów



Map-Reduce: liczenie słów


```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
      emit(w, 1)
```

```
reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

Map-Reduce: host size

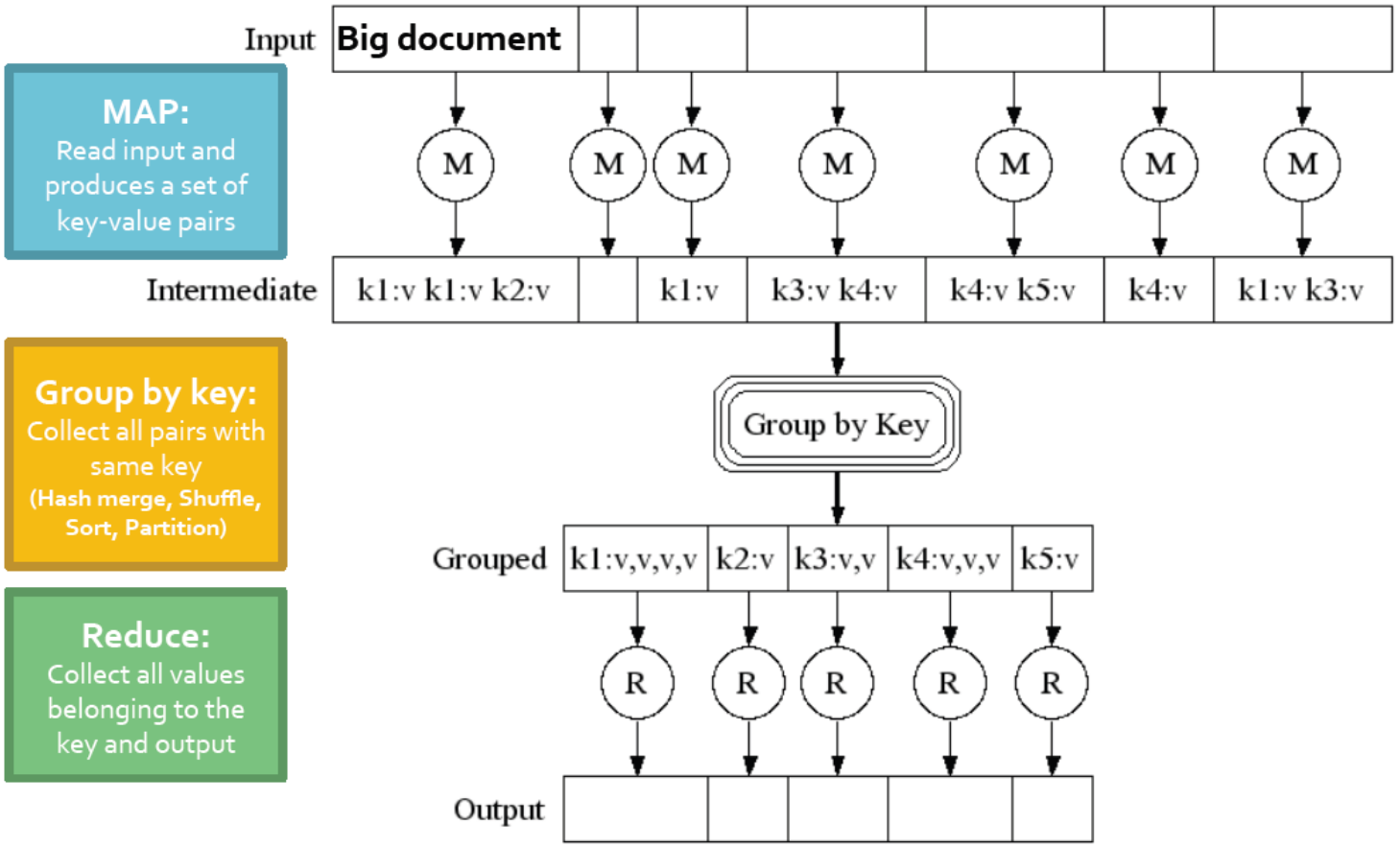
- Przypuśćmy że mamy ogromny zbiór danych sformatowanych w następujący sposób:
  - Każdy record w postaci: **(URL, rozmiar, data, …)**

- Zadanie: dla każdego host oceń jego rozmiar.
  - **Map:** dla każdego recordu wypisz (hostname(URL), rozmiar)
  - **Reduce:** sumuj rozmiar dla każdego host

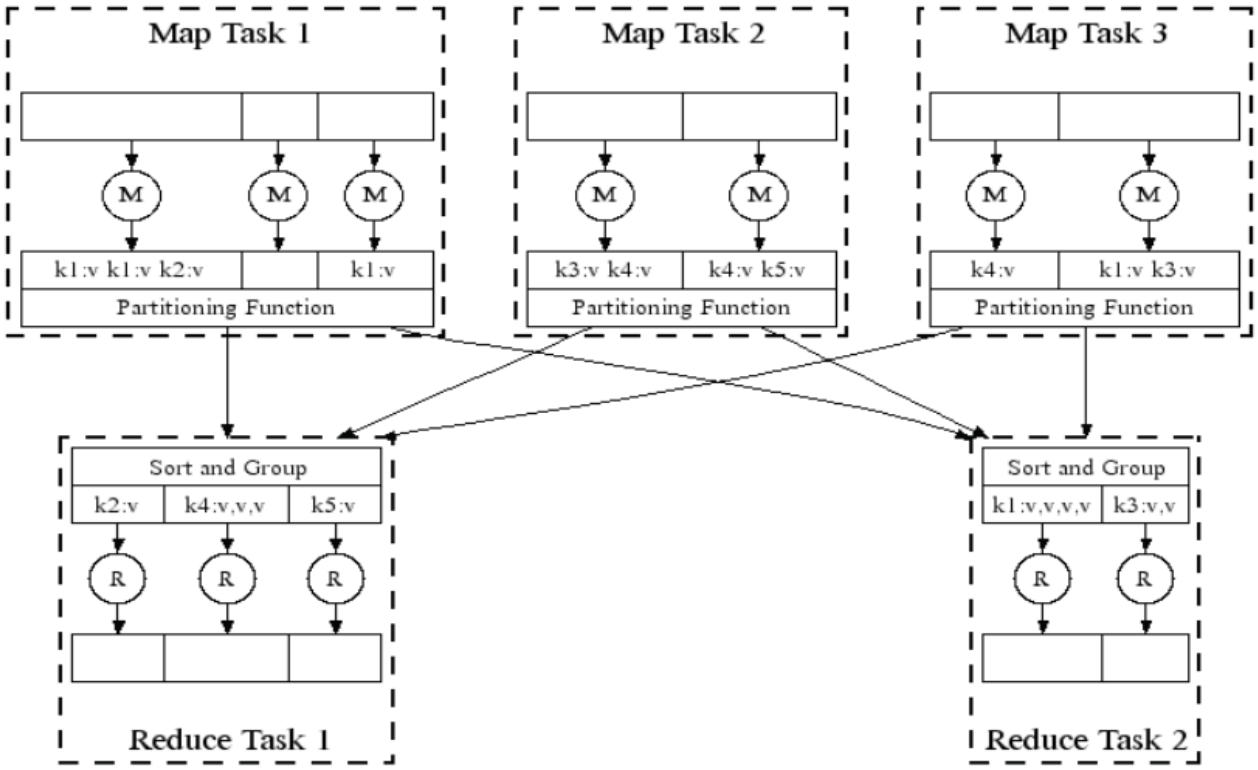
Map-Reduce: Model języka

- Policz ile razy każda z sekwencji 5-ciu słów występuje w danym bardzo dużym zbiorze dokumentów
- **Map:** Extract (5-słów sekwencja, ilość) dla danego dokumentu
- **Reduce:** Sumuj ilości dla każdej sekwencji

Map-Reduce: diagram



Map-Reduce: zrównoleglanie



All phases are distributed with many tasks doing the work

Map-Reduce: środowisko

Środowisko zapewnia:

- Podział na części danych wejściowych
- Zarządza procesowaniem programu na klastrze CPU
- Przeprowadza grupowanie na podstawie klucza
- Zarządza komunikacja pomiędzy maszynami

Map-Reduce: flow danych

- Input i output są przechowywane w rozproszonym systemie plików (DFS)
  - System stara się przydzielać procesory jak najbliżej danych
  - Wyniki pośrednie są przechowywane na lokalnych FS nodów wykonujących operacje Map i Reduce.
  - Output jest często inputem do kolejnego kroku Map-Reduce

Implementacja

- Google MapReduce
  - Używa Google File System
  - Niedostepny poza systemem Googla
- Hadoop
  - Open-source implementacja
  - Używa HDFS jako stabilnego systemu dyskowego
- Hive, Pig
  - Dodatkowa funkcjonalność SQL-podobna