

# DATOS MASIVOS I

## UNIDAD II MODELO DE MAPEO Y REDUCCIÓN

---

### ALGORITMOS DEL MODELO MAP – REDUCE

---

16 de Febrero de 2023

# Sincronización de Tareas

---



Imagen tomada de <https://www.abc.es/>

# Sincronización de Tareas

---



Sincronización de tareas.

Las operaciones de mapeo y reducción se ejecutan de manera sincronizada.

Los nodos de reducción reciben las llaves de manera ordenadas.

# Sincronización de Tareas

---



Sincronización de tareas.

Las operaciones de mapeo y reducción se ejecutan de manera sincronizada.

Los nodos de reducción reciben las llaves de manera ordenadas.



Creación de estructuras algebraicas que contribuyan a optimizar los procesos.

# Algoritmos del Modelo Map – Reduce

---

- El modelo de programación implementa diversos algoritmos matemáticos para dividir una tarea en pequeñas partes y para asignarlas a múltiples nodos.

# Algoritmos del Modelo Map – Reduce

---

Ordenamiento.

Búsqueda.

Índice invertido.

## Combiner Function (Herramienta adicional)

---



# Combiner Function (Herramienta adicional)

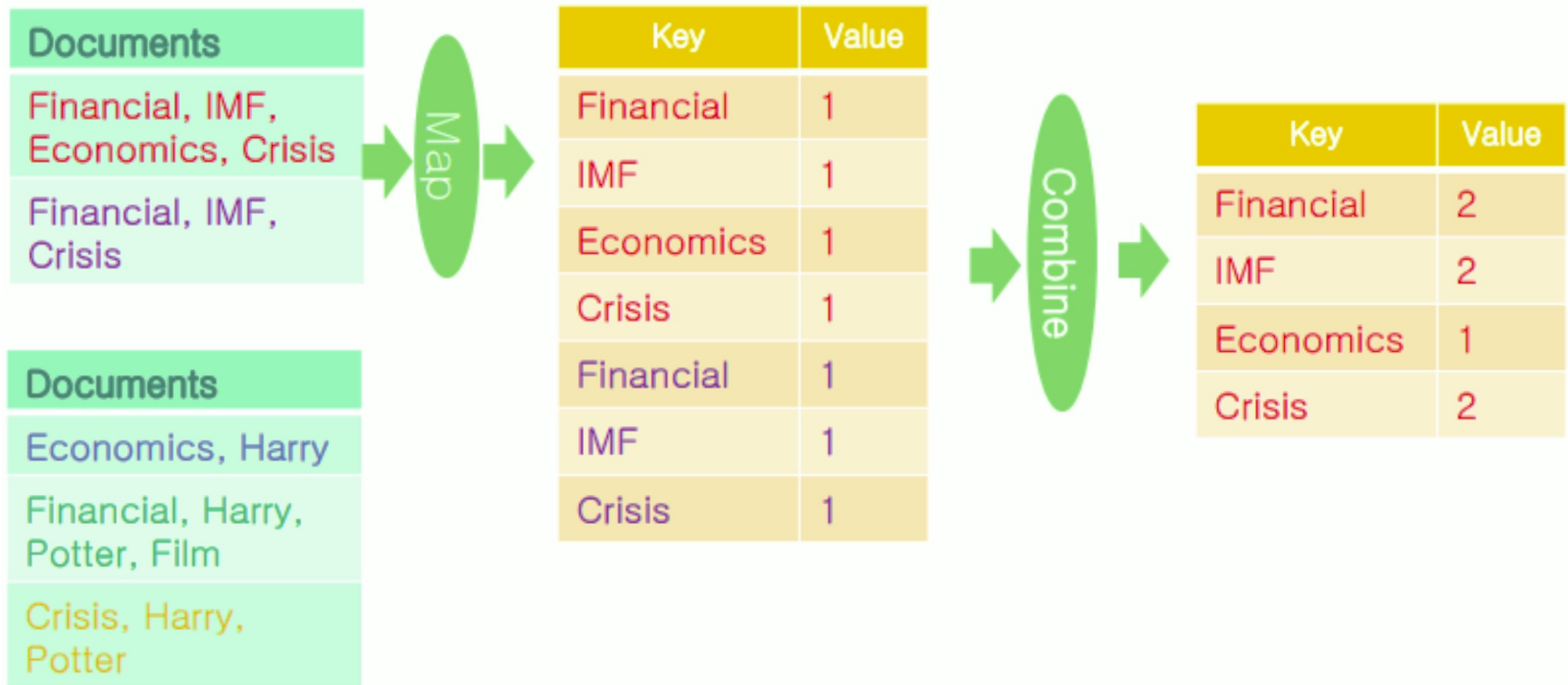


Imagen tomada de Kyuseok Shim, 2013.



# Combiner Function

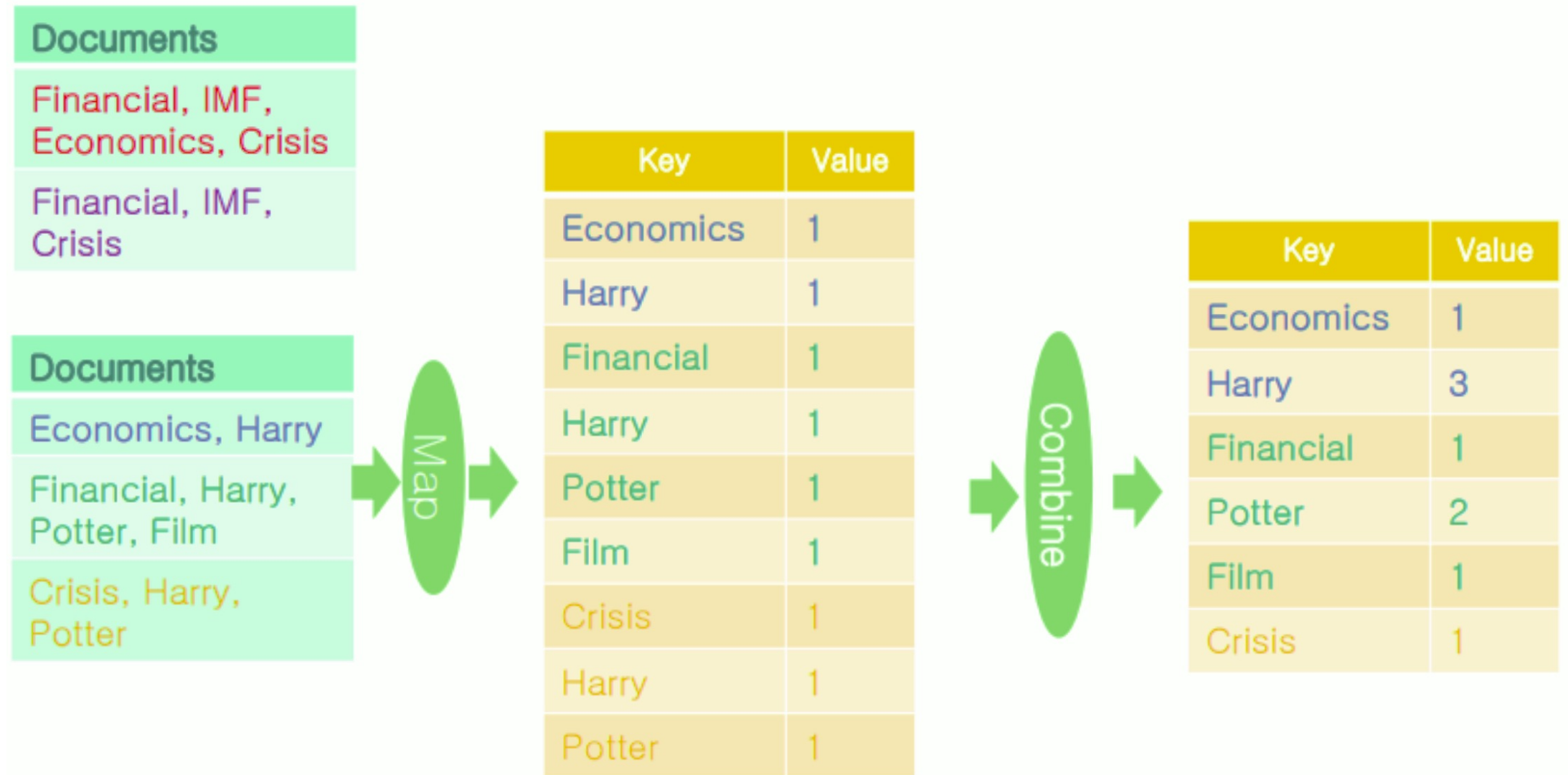


Imagen tomada de Kyuseok Shim, 2013.

# Combiner Function

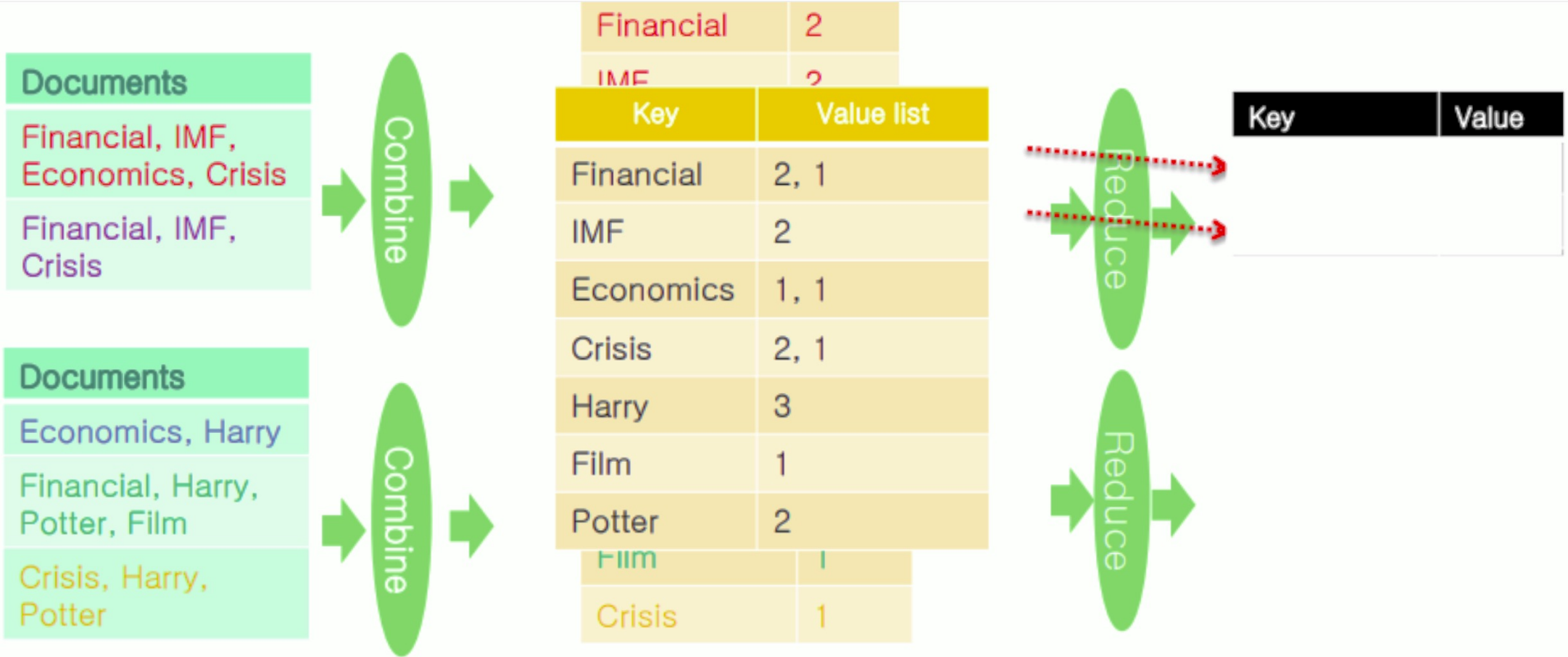


Imagen tomada de Kyuseok Shim, 2013.

Tarea. Ordenar un conjunto de archivos, un valor por línea.

Archivo1  
900 líneas

Archivo2  
5000  
líneas

Archivo3  
350 líneas

Archivo1  
50 líneas

Tarea.  
Ordenar un  
conjunto de  
archivos, un  
valor por  
línea.

- Función de mapeo.
  - La llave será el nombre de archivo y número de línea, el valor será el contenido de línea.
  - Regresa el valor como llave  $(k, v) \Rightarrow (v,)$
- Función de reducción.
  - Función de identidad (One reducer).
- Algoritmo.
  - Toma ventaja de las propiedades del reductor por pares (*llave, valor*) las cuales son procesados en orden por *llave*. Los reductores se ordenan ellos mismos.

# Ordenamiento

---

Aprovecha el ordenamiento de llaves por sistema de manejo de tareas, se define una función de partición tal que:

$$k_1 < k_2 \implies \text{hash}(k_1) < \text{hash}(k_2)$$

- Es usado como prueba de velocidad de Hadoop.
- Es una carrera de resistencia *entradas – salidas*.

Tarea. Encontrar documentos que contienen un patrón dado.

# Búsqueda

---

## Función de mapeo.

- La llave será el nombre de archivo y número de línea, el valor será el contenido de línea.
- Regresa nombre de archivo como llave si se encuentre el patrón en el contenido.

## Función de reducción.

- Identidad.

# Búsqueda

---



Una vez que se identificó al documento con el patrón, es necesario marcar ese documento (una sola vez).



Se usa la función *Combiner* para convertir pares redundantes en un solo archivo (*filename,* ).



Reduce las entradas / salidas de la red.



Este algoritmo es el más utilizado en los sistemas de recuperación de información (por ejemplo, motores de búsqueda).

¿Cómo trabaja este algoritmo?

# Índice Invertido

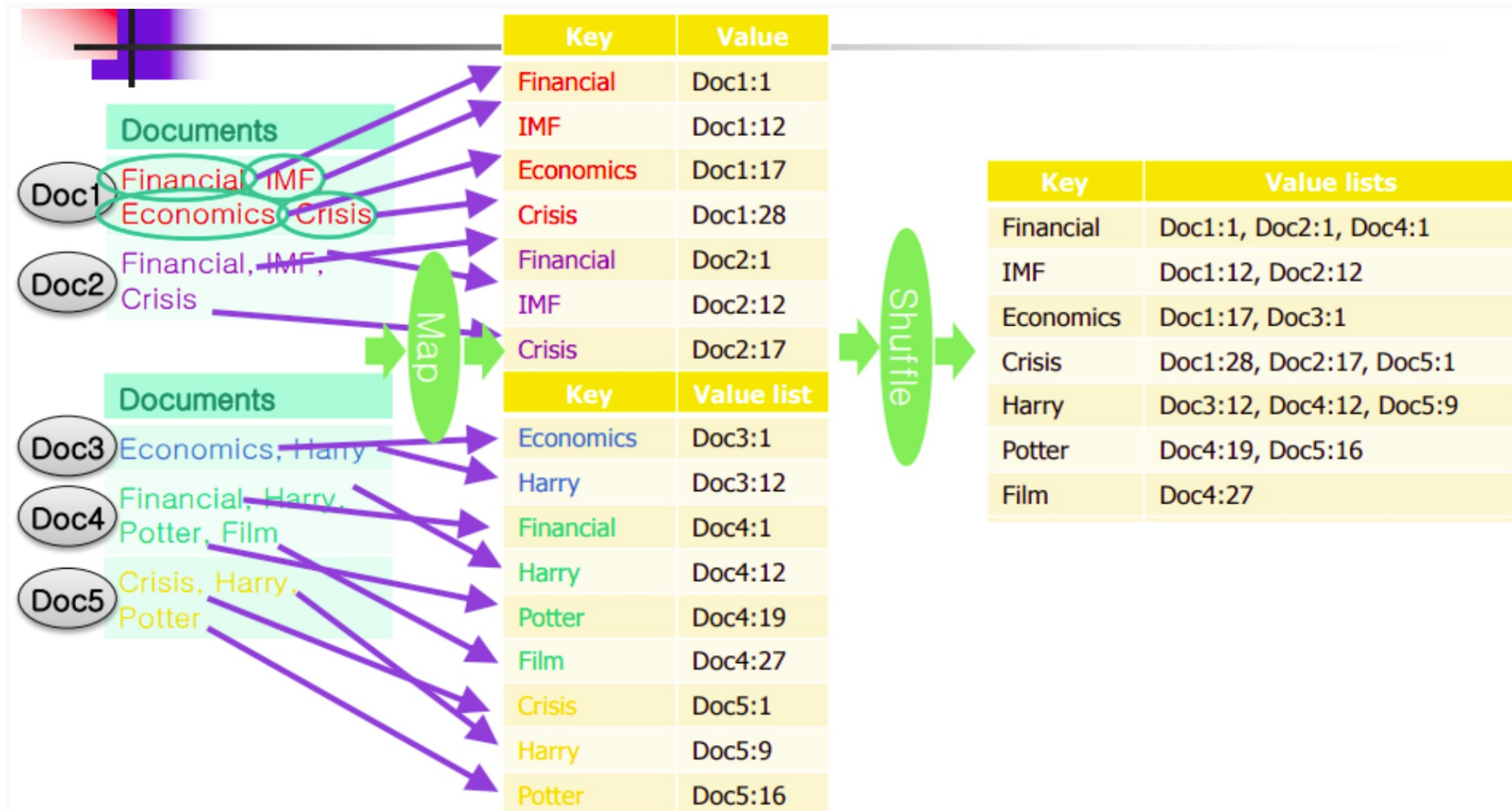
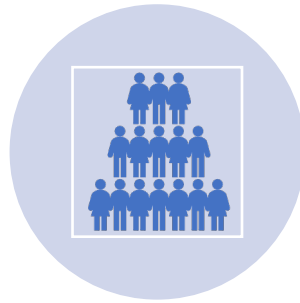


Imagen tomada de Kyuseok Shim, 2013.

# Índice Invertido

---



PARA CADA TÉRMINO ***T***, GUARDAMOS  
TODOS LOS DOCUMENTOS QUE  
CONTIENEN ***T***.



IDENTIFICAMOS CADA  
DOCUMENTO POR UN ***DOCID***, EL  
CUÁL ES UN NÚMERO  
INCREMENTAL.

# Grafos, ¿Qué son?

---

$$G = (V, E)$$

- $V$  representa un conjunto de vértices (nodos).
- $E$  representa un conjunto de aristas (enlaces).
- Tanto vértices como aristas pueden contener información adicional.

## Tipos de grafos.

- Dirigidos o no dirigidos.
- Con y sin ciclos.

# Grafos, ¿En dónde se observan?

---



Redes sociales.



Hipervínculos  
en la web.



Estructuras  
físicas de las  
computadoras.

# Algunos Problemas con los Grafos

---

Encontrar la ruta más corta.

- Tráfico en internet.

Búsqueda de árboles mínimos.

- Tendido de fibra óptica.

Encontrar el flujo máximo.

- Programación de aerolíneas.

Identificar comunicaciones 'especiales'.

- Terrorismo / conspiraciones.
- Enfermedades.

El  
procesamiento  
con grafos:

Ejecutar cálculos en cada uno de los nodos: basados en características de los nodos y de las aristas.

---

Propagar los cálculos 'atravesando' el grafo.

---

# Grafos y Map – Reduce

---

El  
procesamiento  
con grafos:

Ejecutar cálculos en cada uno de los nodos: basados en características de los nodos y de las aristas.

---

Propagar los cálculos 'atravesando' el grafo.

---

Preguntas:

¿Cómo representar los datos de un grafo en Map – Reduce?

---

¿Cómo recorrer el grafo usando Map – Reduce?

---



## Grafos y Map – Reduce

Los grafos comúnmente son representados como una matriz de adyacencias o una lista de adyacencias.

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>1</b>	0	1	0	1
<b>2</b>	1	0	1	1
<b>3</b>	1	0	0	0
<b>4</b>	1	0	1	0

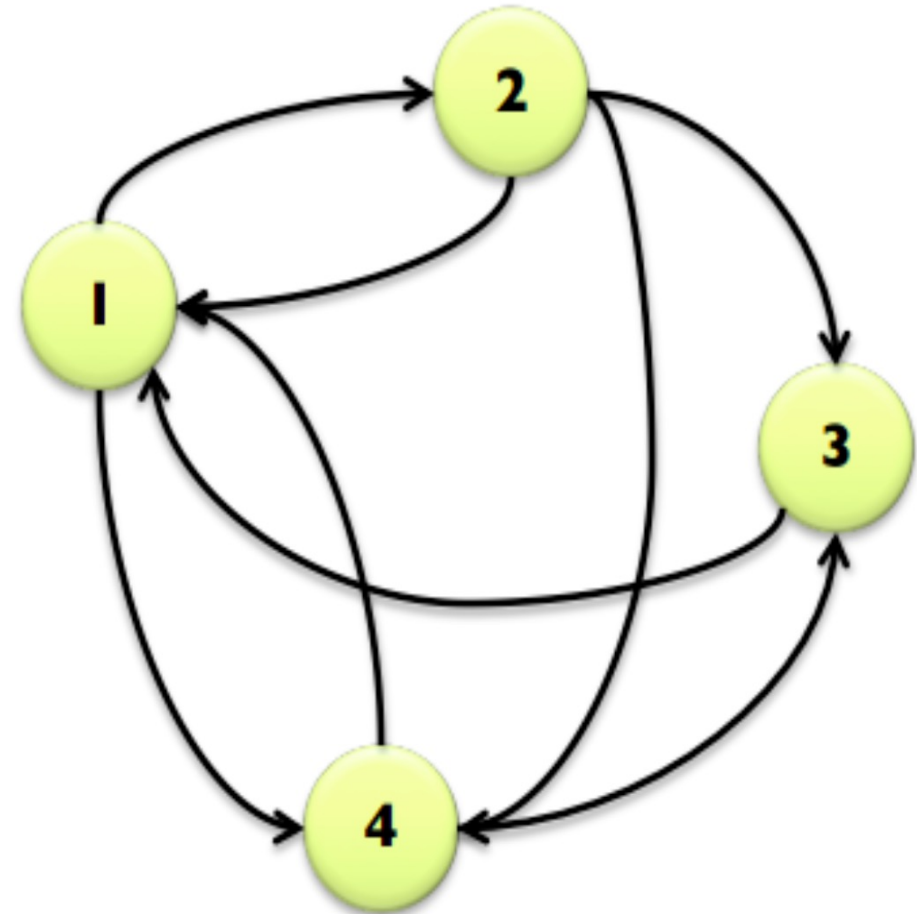


Imagen tomada de Jimmy Lin, 2013.

# Matrices de Adyacencia: Pros and Cons

---

## Ventajas.

- Manipulación accesible de los datos.
- Iteración sobre filas y columnas, correspondientes con los enlaces salientes y entrantes.

## Desventajas.

- Muchos ceros.
- Ocupan una gran cantidad de espacio.

## Listas de Adyacencia

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>1</b>	0	1	0	1
<b>2</b>	1	0	1	1
<b>3</b>	1	0	0	0
<b>4</b>	1	0	1	0

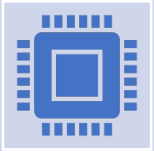


**1: 2, 4**  
**2: 1, 3, 4**  
**3: 1**  
**4: 1, 3**

Imagen tomada de Jimmy Lin, 2013.

# Encontrar el Camino Más Corto (Shortest Path: BFS)

---



Realizar cálculos en una estructura de datos de grafos requiere procesamiento en cada nodo.



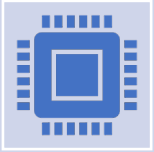
Cada nodo contiene datos específicos del nodo, así como enlaces (aristas) a otros nodos.



El cálculo debe atravesar el grafo.

# Encontrar el Camino Más Corto (Shortest Path: BFS)

---



Realizar cálculos en una estructura de datos de grafos requiere procesamiento en cada nodo.



Cada nodo contiene datos específicos del nodo, así como enlaces (aristas) a otros nodos.

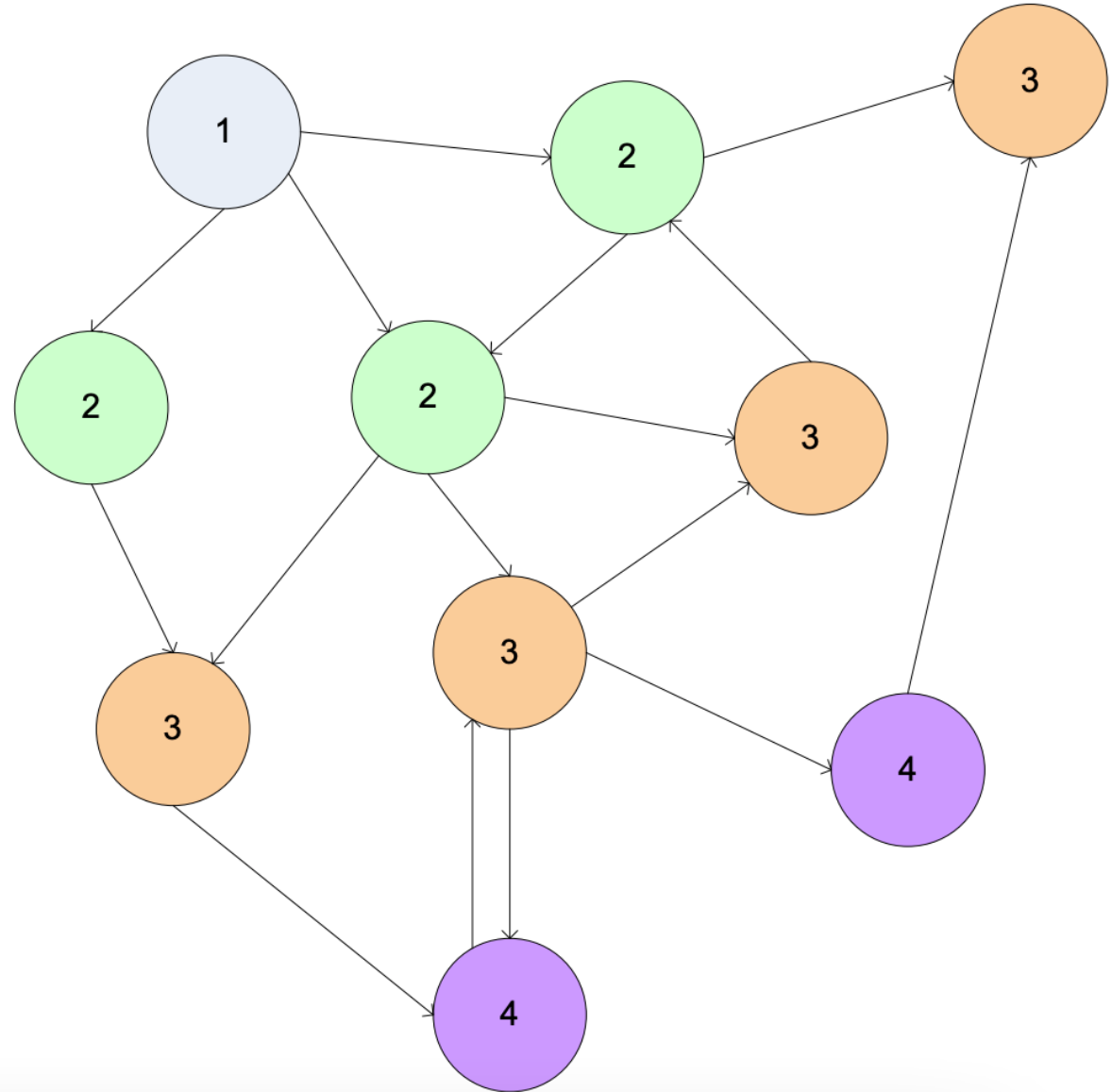


El cálculo debe atravesar el grafo.

¿Cómo recorreremos un grafo en MapReduce?  
¿Cómo representamos el grafo para esto?

## Encontrar el Camino Más Corto (Shortest Path: BFS)

- Breadth-First Search es un algoritmo iterativo sobre grafos.
- La frontera avanza desde el origen un nivel con cada paso.



## Encontrar el Camino Más Corto (Shortest Path: BFS)

---

- Problema: Esto no “encaja” en Map – Reduce.

## Encontrar el Camino Más Corto (Shortest Path: BFS)

---

Solución: Pasos iterados a través del grafo.

Map – Reduce: Mapea algunos nodos, el resultado incluye nodos adicionales que se introducen en pasos sucesivos de Map – Reduce.



# Intuición

---

Podemos definir la solución a este problema de forma inductiva:

$$\text{Distancia}(\text{nodoDelInicio}) = 0$$

- Para todos los nodos  $n$  accesibles directamente desde  $\text{nodoDelInicio}$ ,

$$\text{DistanceA}(n) = 1$$

- Para todos los nodos  $n$  accesibles desde algún otro conjunto de nodos  $S$ ,

$$\text{DistanciaA}(n) = 1 + \min(\text{DistanciaA}(m), m \in S)$$

- Una tarea de Mapeo recibe un nodo  $n$  como clave (llave) y  $(D, \text{points-to})$  como su valor.
  - $D$  es la distancia al nodo desde el inicio.
  - $\text{points-to}$  es una lista de nodos accesibles desde  $n$ .
  - $\forall p \in \text{points-to}, \text{emite } (p, D+1)$

# De la Intuición al Algoritmo

---



La tarea Reduce recolecta las distancias posibles a una  $p$  dada y selecciona la mínima.



Esta tarea de Map – Reduce puede avanzar (y actualizar) a la frontera conocida en un salto.

## Criterio de Terminación (BFS)

---



El algoritmo parte de un nodo inicial.



Las iteraciones subsecuentes incluyen muchos más nodos del grafo a medida que la frontera avanza.

## Criterio de Terminación (BFS)

---



El algoritmo parte de un nodo inicial.



Las iteraciones subsecuentes incluyen muchos más nodos del grafo a medida que la frontera avanza.

**¿Esto termina alguna vez?**

Si, eventualmente se dejarán de descubrir rutas entre nodos y no se encontrarán mejores distancias (más cortas).

## Criterio de Terminación (BFS)

---

Cuando la distancia es la misma, nos detenemos.

---

El Mapper debe emitir  $(n, D)$  para garantizar que la "distancia actual" es llevada a Reduce.

# Encontrar el Camino Más Corto (Shortest Path: BFS)

```
1: class MAPPER
2:   method MAP(nid  $n$ , node  $N$ )
3:      $d \leftarrow N.DISTANCE$ 
4:     EMIT(nid  $n$ ,  $N$ )                                ▷ Pass along graph structure
5:     for all nodeid  $m \in N.ADJACENCYLIST$  do
6:       EMIT(nid  $m$ ,  $d + 1$ )                            ▷ Emit distances to reachable nodes

1: class REDUCER
2:   method REDUCE(nid  $m$ , [ $d_1, d_2, \dots$ ])
3:      $d_{min} \leftarrow \infty$ 
4:      $M \leftarrow \emptyset$ 
5:     for all  $d \in \text{counts } [d_1, d_2, \dots]$  do
6:       if ISNODE( $d$ ) then
7:          $M \leftarrow d$                                 ▷ Recover graph structure
8:         else if  $d < d_{min}$  then                        ▷ Look for shorter distance
9:            $d_{min} \leftarrow d$ 
10:     $M.DISTANCE \leftarrow d_{min}$                         ▷ Update shortest distance
11:    EMIT(nid  $m$ , node  $M$ )
```



## Criterio de Terminación (BFS)

---

- El algoritmo de Dijkstra es más eficiente porque en cualquier paso del algoritmo, solo persigue bordes (aristas) desde el camino de costo mínimo dentro de la frontera.

## Criterio de Terminación (BFS)

---

No es tan eficiente en general, pero la arquitectura es más escalable.



La versión de Map – Reduce explora todas las rutas en paralelo.

## Criterio de Terminación (BFS)

---

No es tan eficiente en general, pero la arquitectura es más escalable.



La versión de Map – Reduce explora todas las rutas en paralelo.



Equivalente a Dijkstra para peso = 1

Las tareas de mapeo y reducción:

- Son hilos independientes en cada nodo.

## Combiner functions

- Reduce el tamaño de las funciones de mapeo.
- Ejecuta mini-funciones de reducción en cada nodo.
- Disminuye el costo para el ordenamiento.

Cada tarea de mapeo y reducción puede opcionalmente usar dos funciones:

- *init()* llamado al inicio de cada tarea.
- *close()* llamado al final de cada tarea.