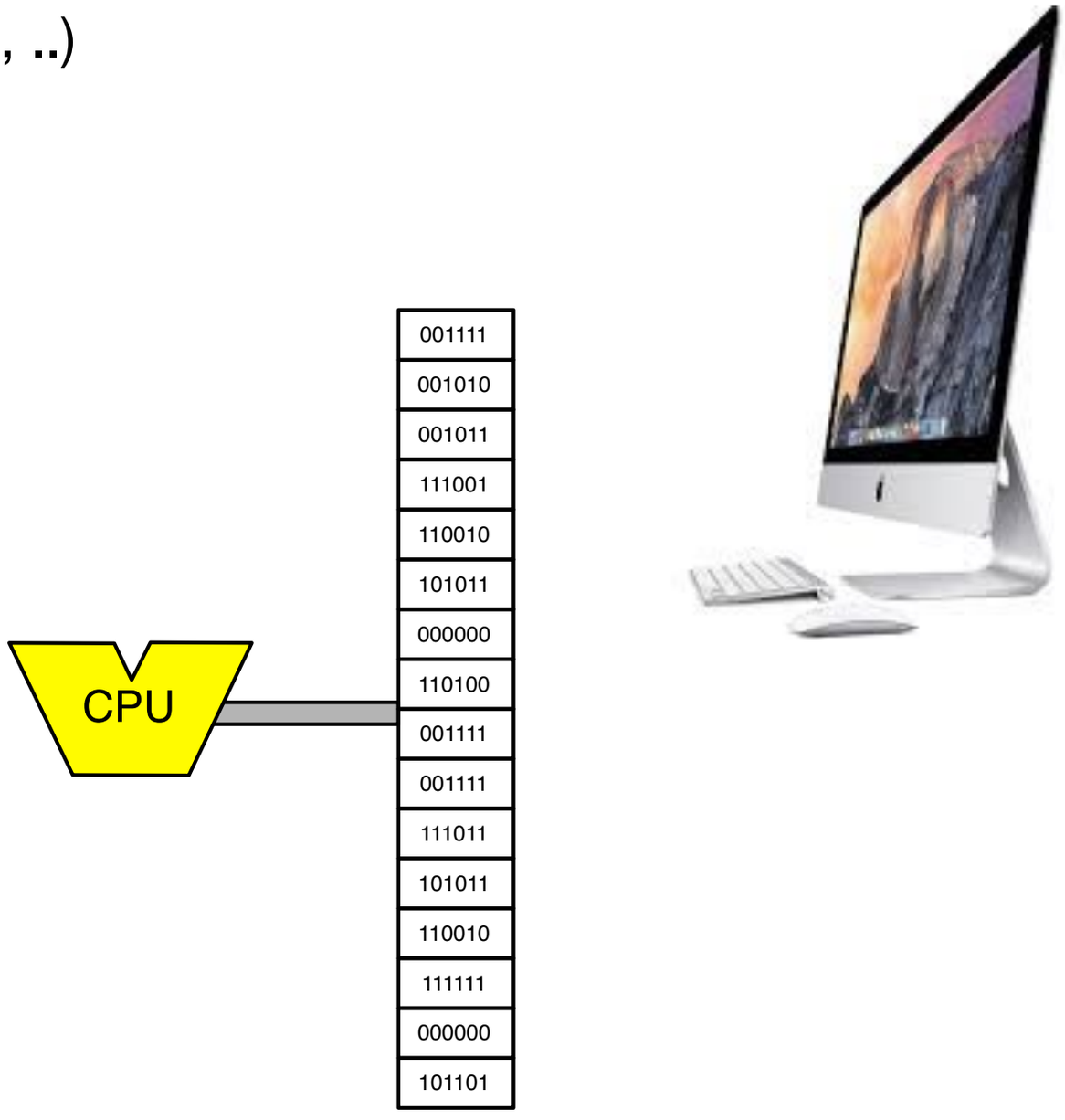
Massively Parallel Computation

Philip Bille

Sequential Computation

- Computation.
  - Read and write in storage
  - Arithmetic and boolean operations
  - Control-flow (if-then-else, while-do, ..)

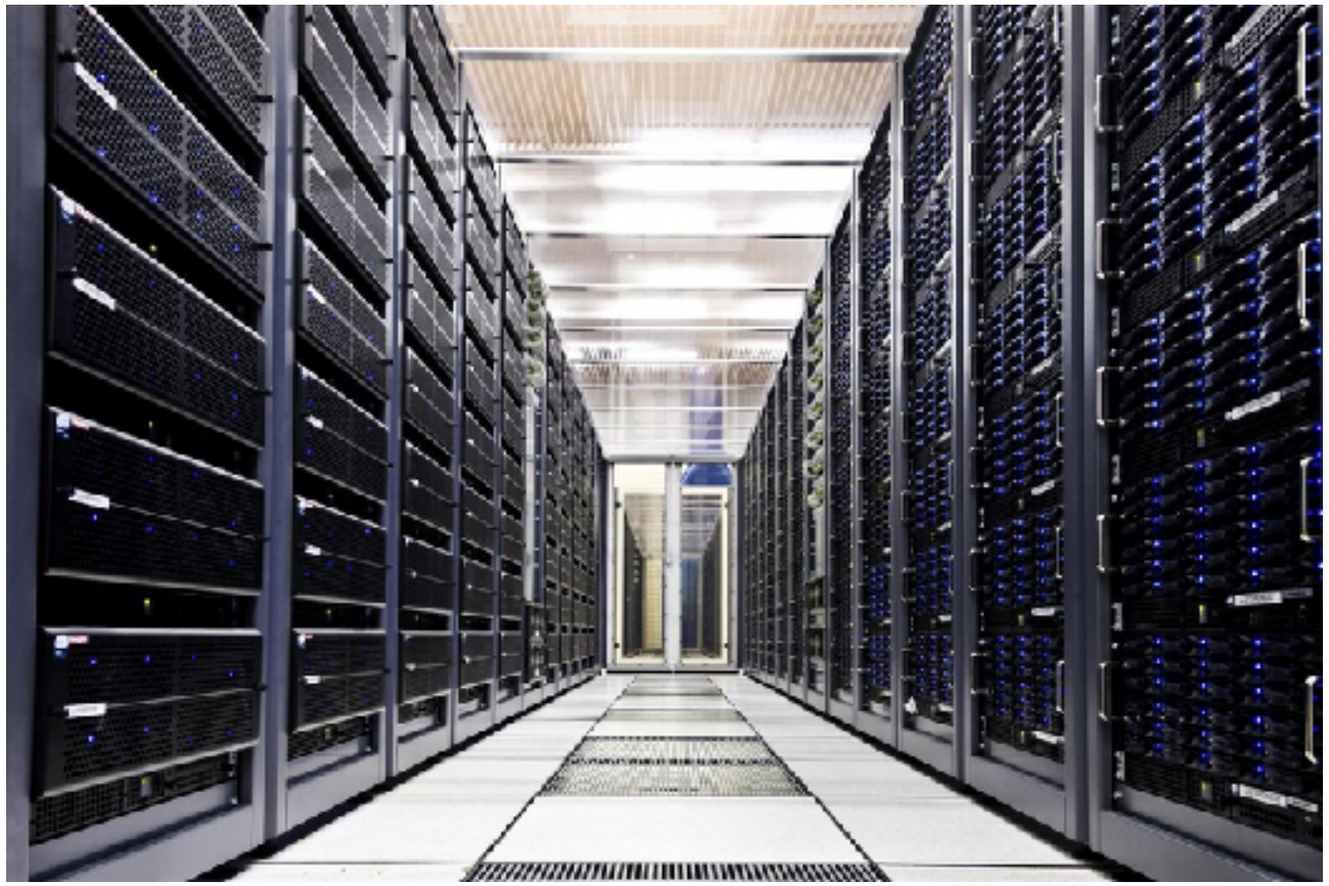
- Scalability.
  - Massive data.
  - Efficiency constraints.
  - Limited resources.



Massively Parallel Computation

- Massively parallel computation.
  - Lots of sequential processors.

- Parallelism.
  - Communication.
  - Failures and error recovery.
  - Deadlock and race conditions
  - Predictability
  - Implementation



MapReduce

MapReduce

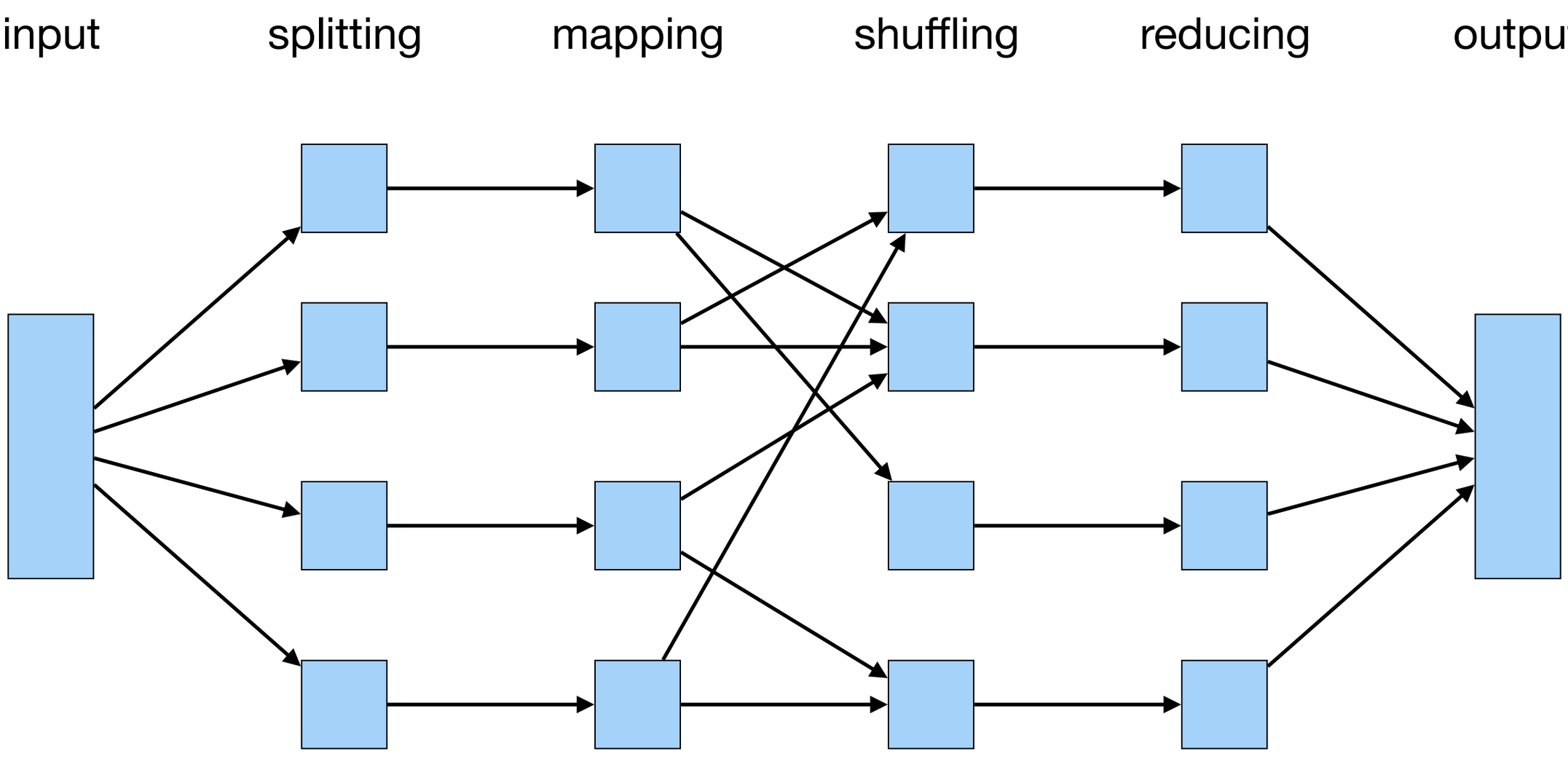
- “MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.” — Wikipedia.

MapReduce

- Dataflow.
  - Split. Partition data into segments and distribute to different machines.
  - Map. Map data items to list of <key, value> pairs.
  - Shuffle. Group data with the same key and send to same machine.
  - Reduce. Takes list of values with the same key <key, [$value_1$, ..., $value_k$]> and outputs list of new data items.
- You only write map and reduce function.

- Goals.
  - Few rounds, maximum parallelism.
  - Work distribution.
  - Small total work.

MapReduce



map(data item) → list of <key, value> pairs

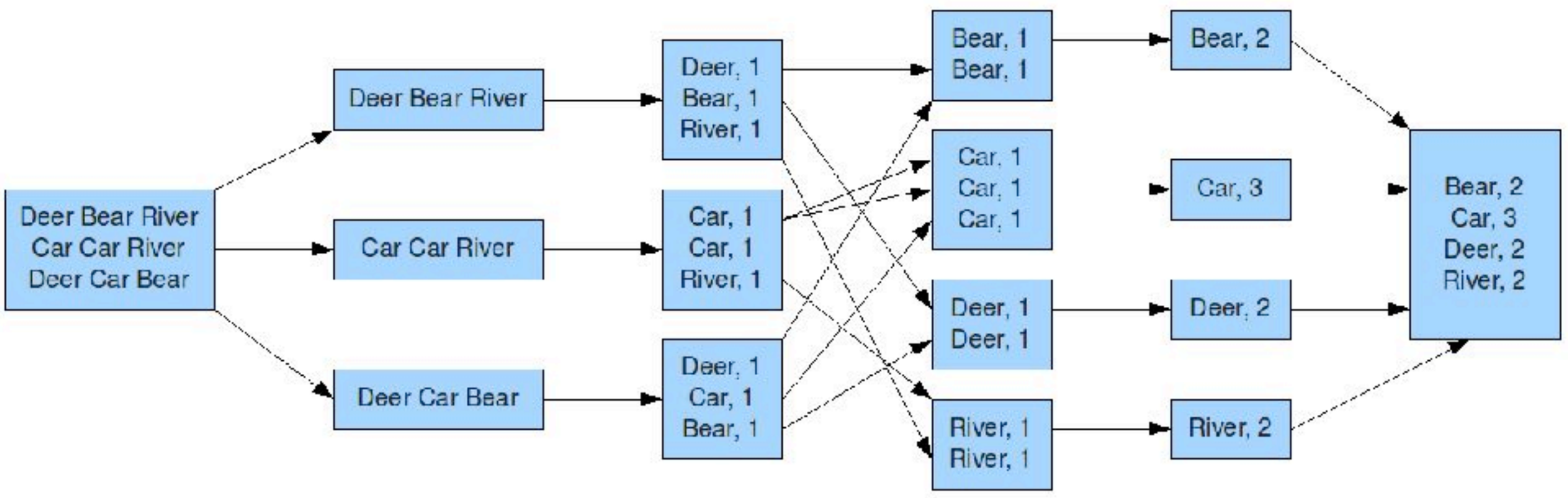
reduce(key, [$value_1$, $value_2$, ..., $value_k$]) → list of new items

Word Counting

- Input.
  - Document of words
- Output.
  - Frequency of each word

- Document: “Deer Bear River Car Car River Deer Car Bear.”

- (Bear, 2), (Car, 3), (Deer, 2), (River, 2)



map(word) → <word, 1>

reduce(word, [1, 1, .., 1]) → <word, number of 1's>

Inverted Index

- Input.
  - Set of documents
- Output.
  - List of documents that contain each word.

- Document 1: “Deer Bear River Car Car River Deer Car Bear.”
- Document 2: "Deer Antilope Stream River Stream"

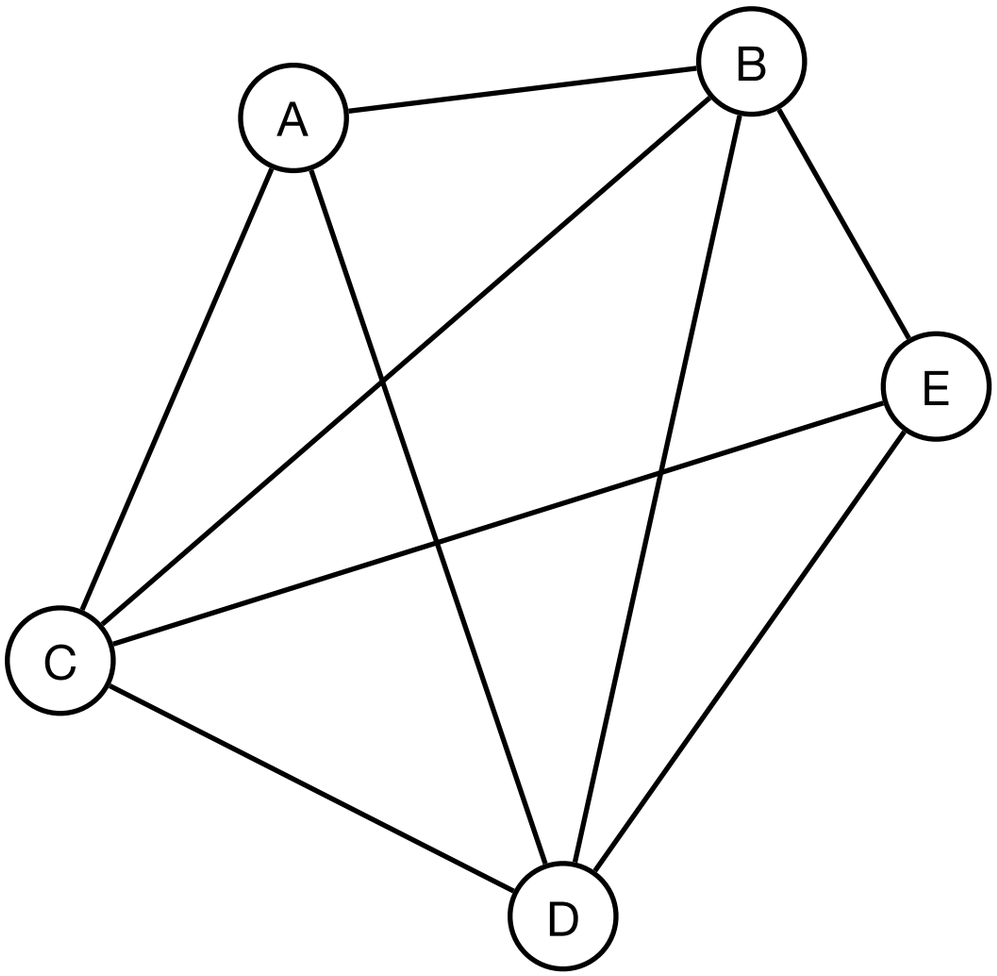
- (Bear, [1]), (Car, [1]), (Deer, [1,2]), (River, [1,2]), (Antilope, [2]), (Stream, [2])

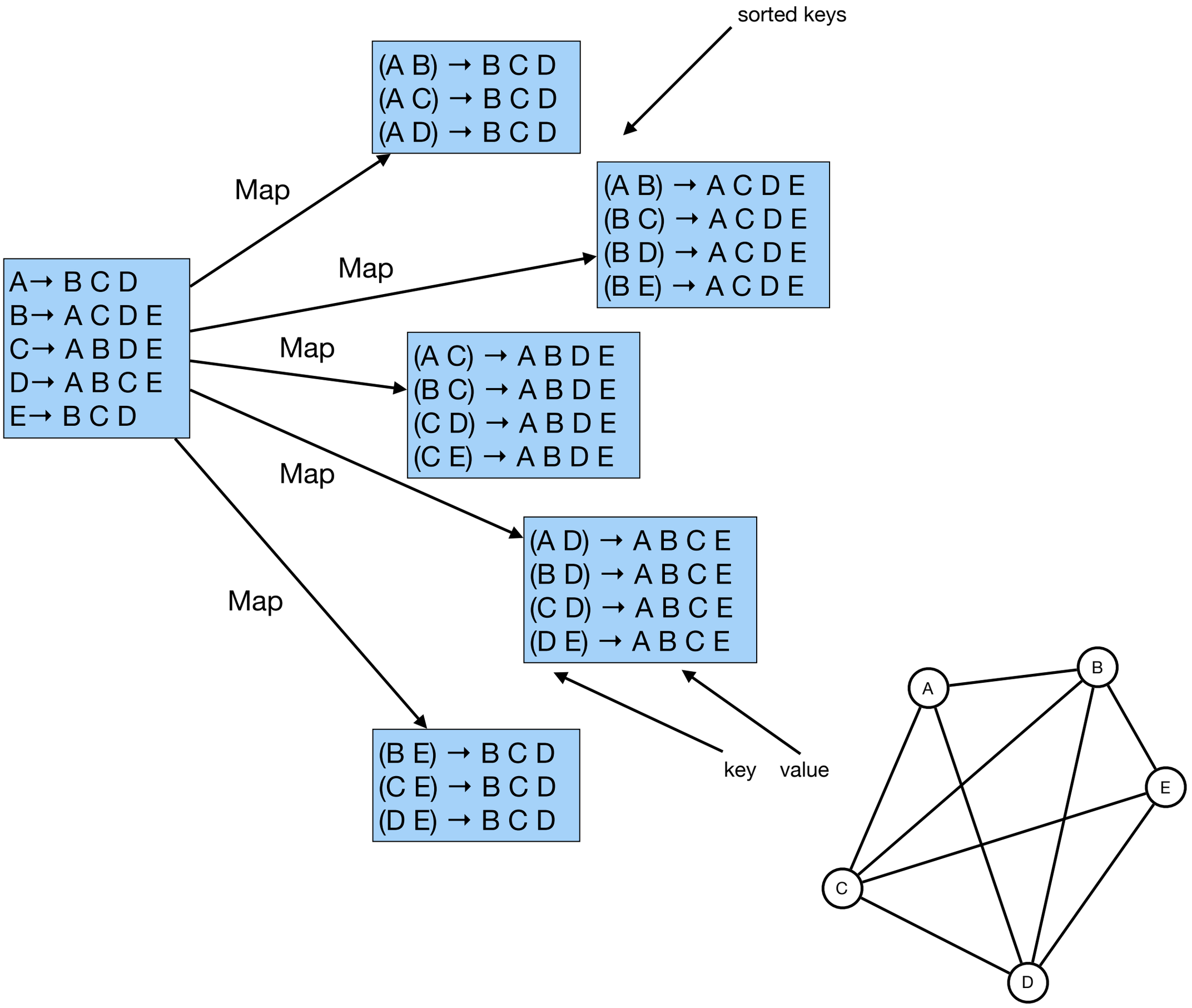
Common Friends

- Input.
  - Friends lists
- Output.
  - For pairs of friends, a list of common friends

A→ B C D
B→ A C D E
C→ A B D E
D→ A B C E
E→ B C D

(A B) → (C D)
(A C) → (B D)
(A D) → (B C
(B C) → (A D E)
(B D) → (A C E)
(B E) → (C D)
(C D) → (A B E)
(C E) → (B D
(D E) → (B C)




sorted keys
(A B) → B C D
(A C) → B C D
(A D) → B C D
Map
(A B) → A C D E
(B C) → A C D E
(B D) → A C D E
(B E) → A C D E
A→ B C D
B→ A C D E
C→ A B D E
D→ A B C E
E→ B C D
Map
Map
(A C) → A B D E
(B C) → A B D E
(C D) → A B D E
(C E) → A B D E
Map
(A D) → A B C E
(B D) → A B C E
(C D) → A B C E
(D E) → A B C E
Map
(B E) → B C D
(C E) → B C D
(D E) → B C D
key
value
A
B
C
D
E

(A B) → B C D
(A C) → B C D
(A D) → B C D

(A B) → A C D E
(B C) → A C D E
(B D) → A C D E
(B E) → A C D E

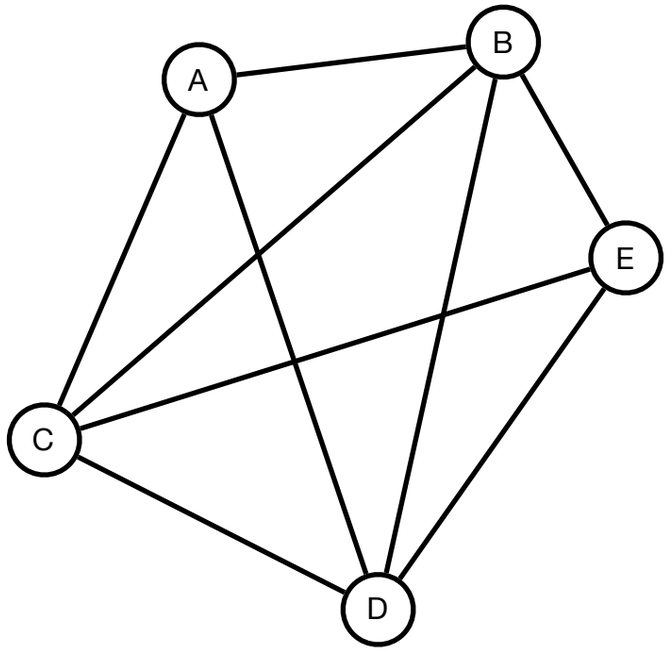
(A C) → A B D E
(B C) → A B D E
(C D) → A B D E
(C E) → A B D E

(A D) → A B C E
(B D) → A B C E
(C D) → A B C E
(D E) → A B C E

(B E) → B C D
(C E) → B C D
(D E) → B C D

Group by key

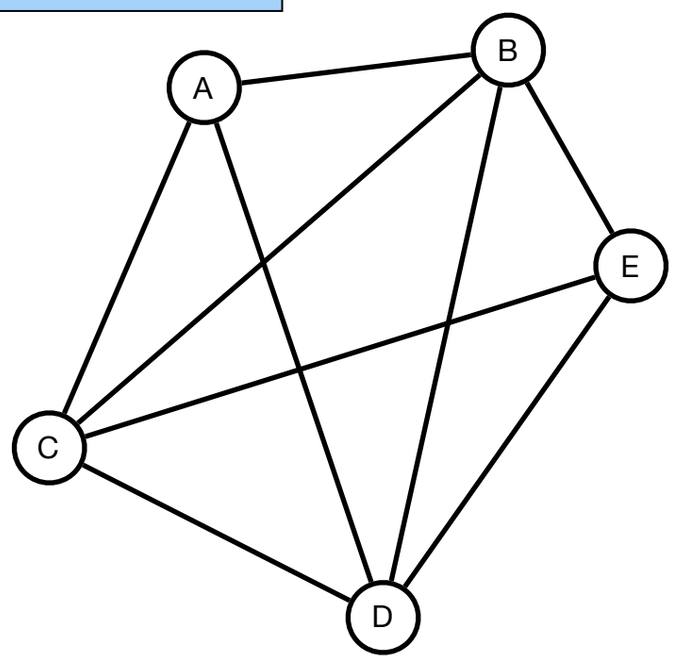
(A B) → (A C D E) (B C D)
(A C) → (A B D E) (B C D)
(A D) → (A B C E) (B C D)
(B C) → (A B D E) (A C D E)
(B D) → (A B C E) (A C D E)
(B E) → (A C D E) (B C D)
(C D) → (A B C E) (A B D E)
(C E) → (A B D E) (B C D)
(D E) → (A B C E) (B C D)

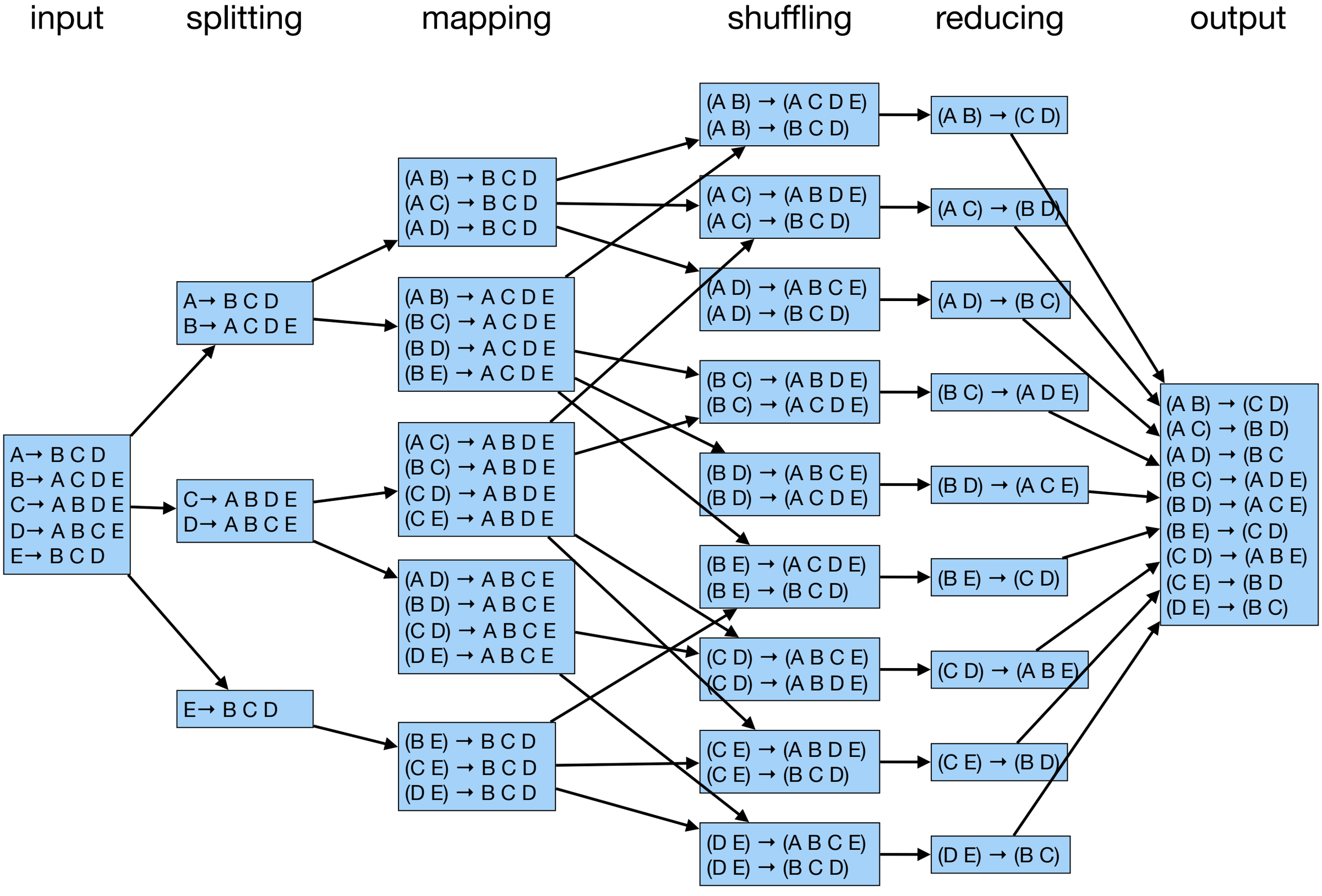


(A B) → (A C D E) (B C D)
(A C) → (A B D E) (B C D)
(A D) → (A B C E) (B C D)
(B C) → (A B D E) (A C D E)
(B D) → (A B C E) (A C D E)
(B E) → (A C D E) (B C D)
(C D) → (A B C E) (A B D E)
(C E) → (A B D E) (B C D)
(D E) → (A B C E) (B C D)

Reduce

(A B) → (C D)
(A C) → (B D)
(A D) → (B C
(B C) → (A D E)
(B D) → (A C E)
(B E) → (C D)
(C D) → (A B E)
(C E) → (B D
(D E) → (B C)



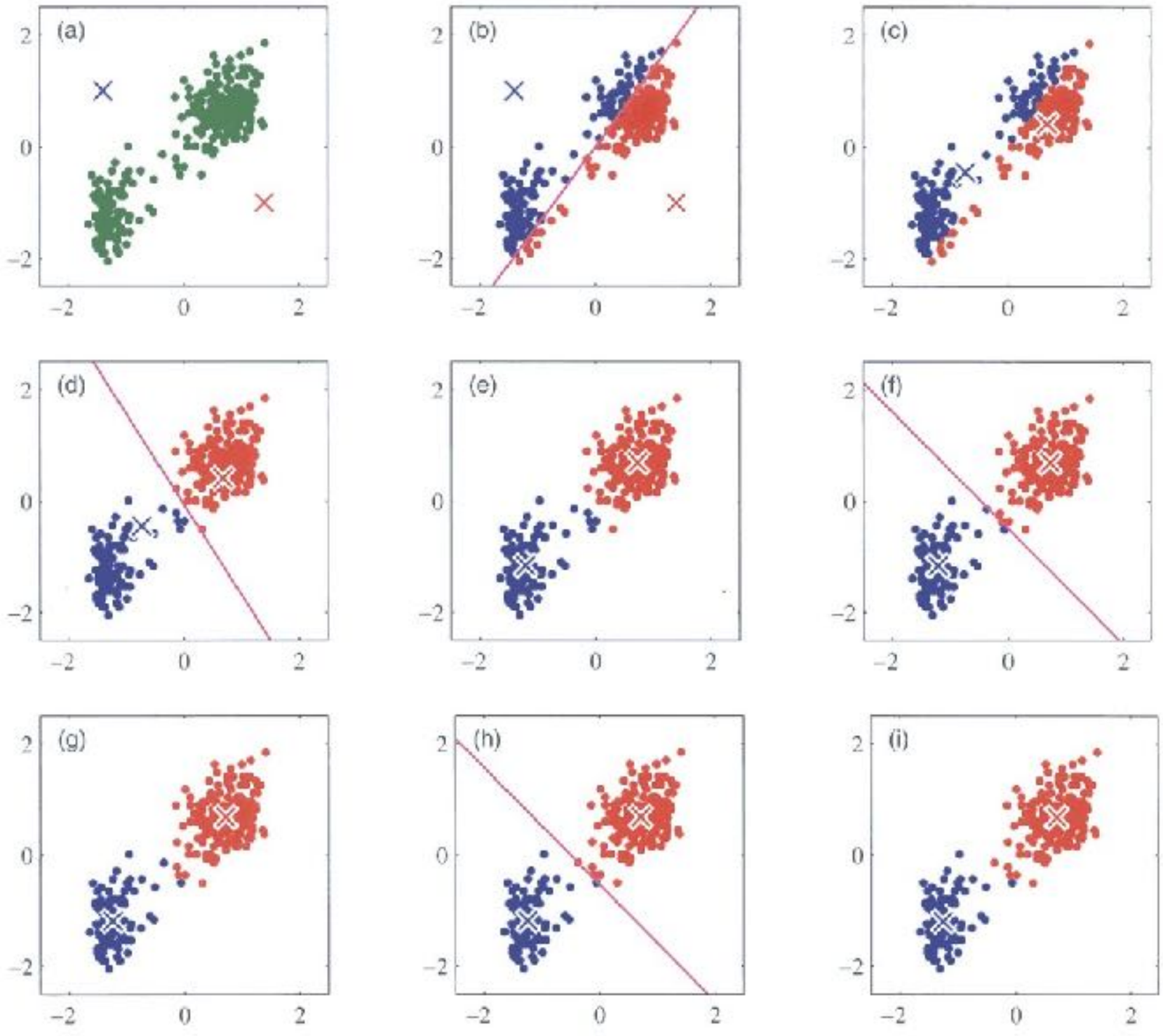

input
splitting
mapping
shuffling
reducing
output
A→ B C D
B→ A C D E
C→ A B D E
D→ A B C E
E→ B C D
A→ B C D
B→ A C D E
C→ A B D E
D→ A B C E
E→ B C D
(A B) → B C D
(A C) → B C D
(A D) → B C D
(A B) → A C D E
(B C) → A C D E
(B D) → A C D E
(B E) → A C D E
(A C) → A B D E
(B C) → A B D E
(C D) → A B D E
(C E) → A B D E
(A D) → A B C E
(B D) → A B C E
(C D) → A B C E
(D E) → A B C E
(B E) → B C D
(C E) → B C D
(D E) → B C D
(A B) → (A C D E)
(A B) → (B C D)
(A C) → (A B D E)
(A C) → (B C D)
(A D) → (A B C E)
(A D) → (B C D)
(B C) → (A B D E)
(B C) → (A C D E)
(B D) → (A B C E)
(B D) → (A C D E)
(B E) → (A C D E)
(B E) → (B C D)
(C D) → (A B C E)
(C D) → (A B D E)
(C E) → (A B D E)
(C E) → (B C D)
(D E) → (A B C E)
(D E) → (B C D)
(A B) → (C D)
(A C) → (B D)
(A D) → (B C)
(B C) → (A D E)
(B D) → (A C E)
(B E) → (C D)
(C D) → (A B E)
(C E) → (B D)
(D E) → (B C)
(A B) → (C D)
(A C) → (B D)
(A D) → (B C
(B C) → (A D E)
(B D) → (A C E)
(B E) → (C D)
(C D) → (A B E)
(C E) → (B D)
(D E) → (B C)

K-means

- Input
  - List of points, integer k
- Output
  - k clusters

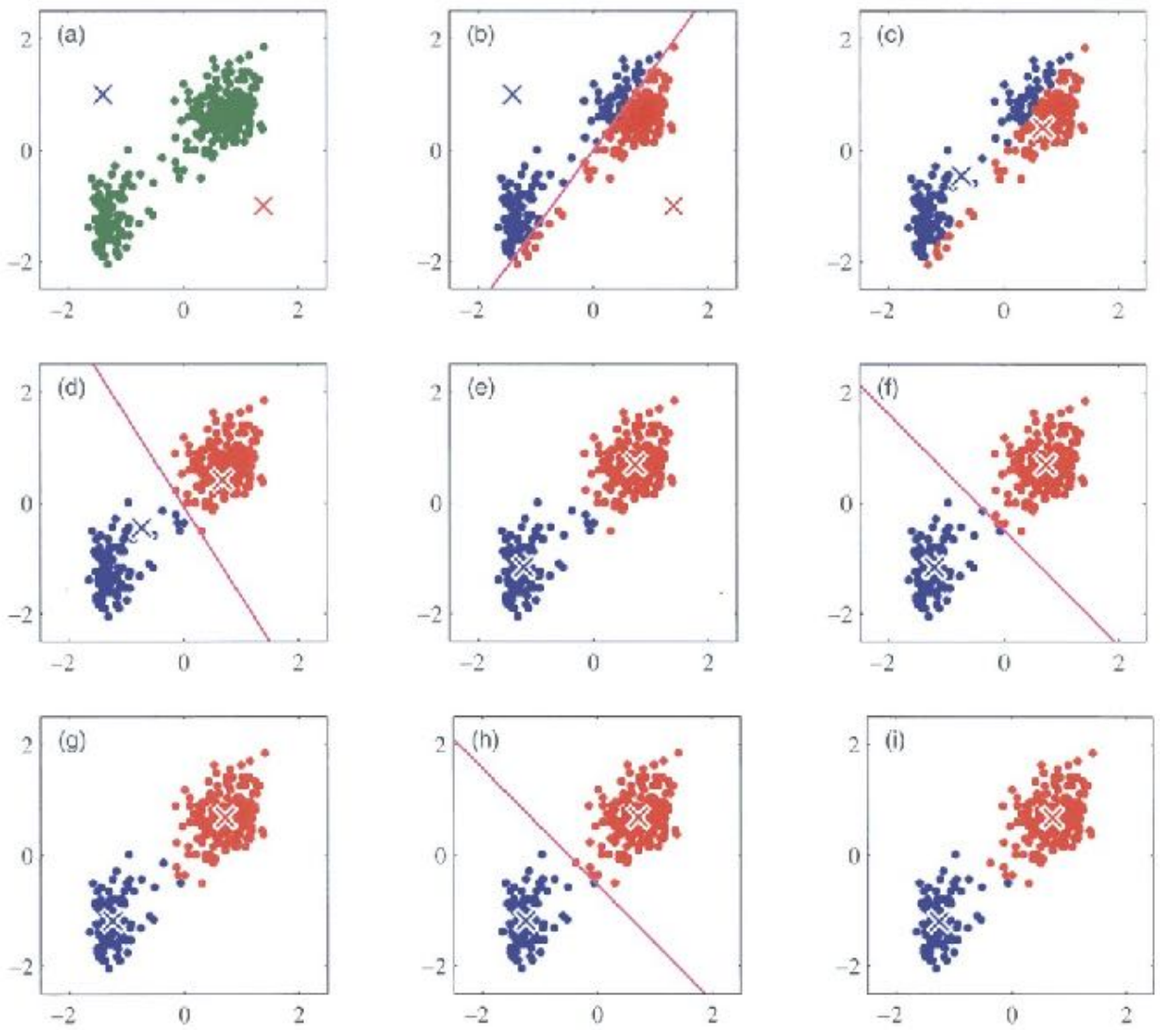
- Algorithm (sequential).

1. Pick k *random* centers
2. Assign each point to the nearest center
3. Move each center to centroid of cluster.
4. Repeat 2-4 until all centers are *stable*.


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)

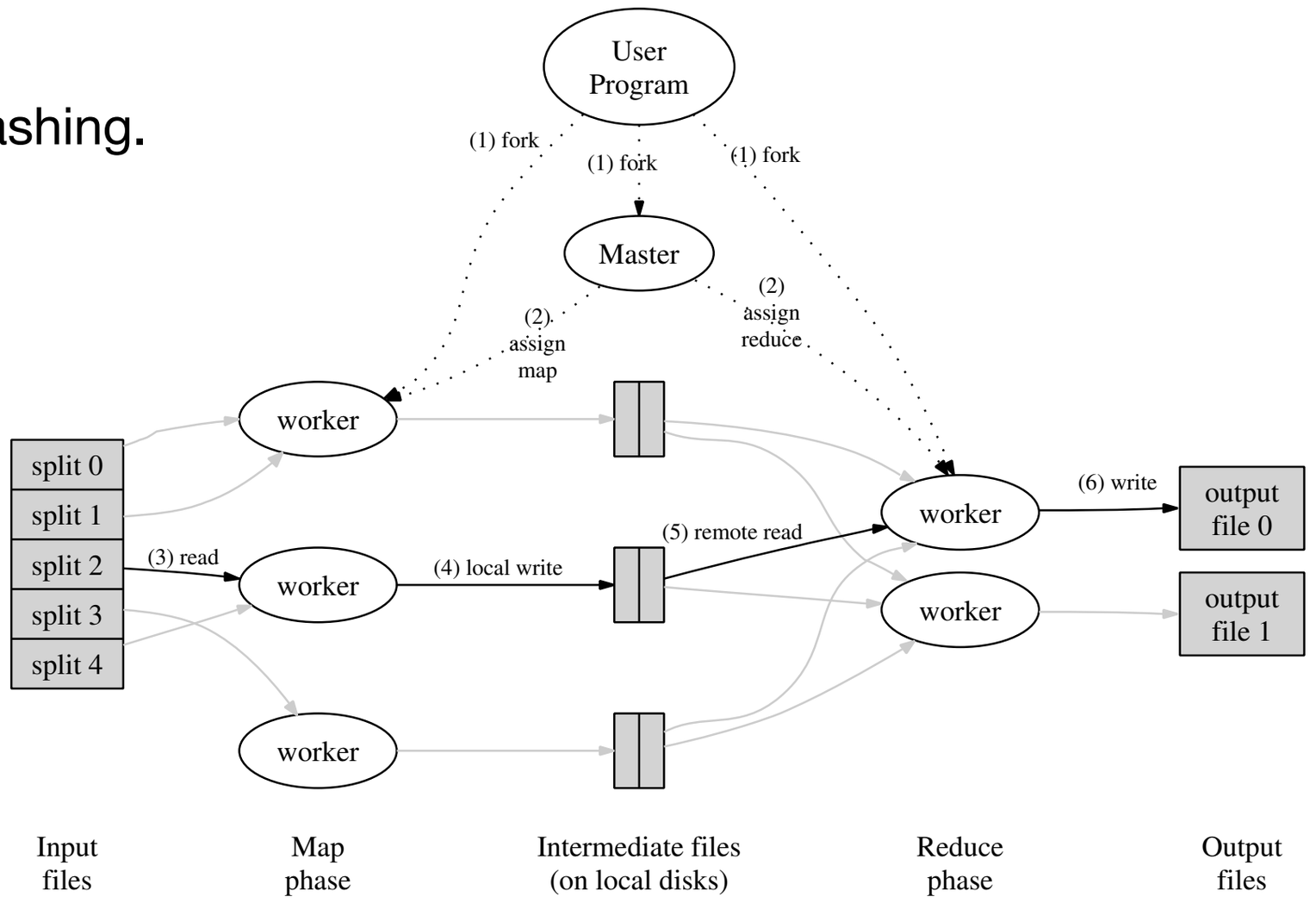
K-means in MapReduce

- K-means iteration.
  - map(point, list of centers) → <closest center, point>
  - reduce(center, [$point_1$, ..., $point_k$]) → centroid of $point_1$, ..., $point_k$



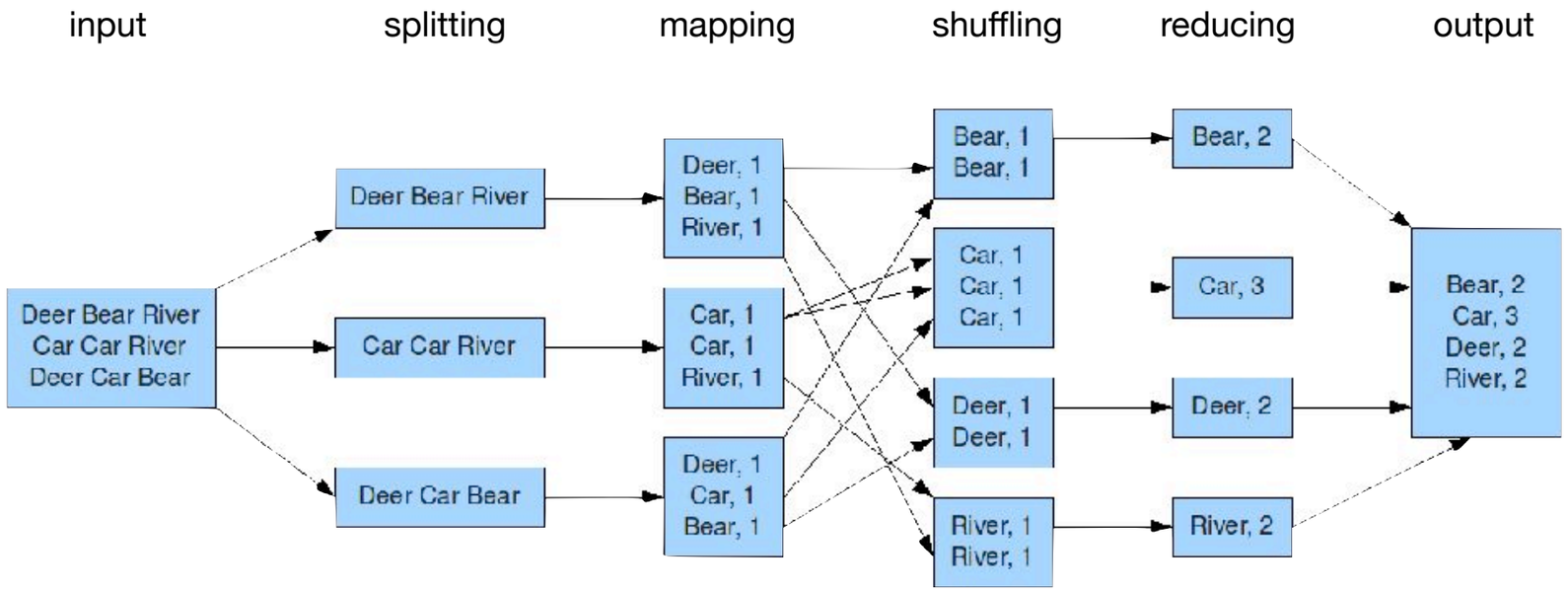
MapReduce Architecture

- Master.
  - Dispatches map and reduce task to workers
- Worker.
  - Performs map and reduce task.
  - Buffered input/output.
  - Splitting and shuffling via hashing.
  - Combiners.
- Fault tolerance.
  - Worker checkpointing.
  - Master restart.



MapReduce and Massively Parallel Computation

- Parallelism.
  - Communication.
  - Failures and error recovery.
  - Deadlock and race conditions
  - Predictability
  - Implementation



map(word) → <word, 1>
reduce(word, [1, 1, .., 1]) → <word, number of 1's>

MapReduce Applications

- Design patterns.
  - Counting, summing, filtering, sorting
  - Cross-correlation (data mining)
  - Iterative message processing (graph processing, clustering)
- More examples.
  - Text search
  - URL access frequency
  - Reverse web-link graph

MapReduce Implementation and Users

- Implementations.
  - Google MapReduce (2004)
  - Apache Hadoop (2006)
  - CouchDB (2005)
  - Disco Project (2008)
  - Infinispan (2009)
  - Riak (2009)
- Example uses.
  - Yahoo (2008): 10.000 linux cores, The Yahoo! Search Webmap
  - FaceBook (2012): Analytics on 100 PB storage, +.5 PB per day.
  - TimesMachine (2008): Digitized full page scan of 150 years of NYT on AWS.