DATA MINING
&
DATA WAREHOUSING

Module III

- **Association Rule Mining**
  - ● What is AR
  - ● Methods to discover AR
  - ● Apriori algo
  - ● Partition algo
  - ● Pincer seaarch algo
  - ● FPtree growth algo
  - ● Incremental algo
  - ● Border algo
  - ● Generalized ARs

- **Algorithm Apriori – Advantages and Disadvantages**
  - **Advantages**
    - **Easy to parallelize and implement**
    - **Use frequent itemset property**
    - **finds all the rules with the specified support and confidence**
  - **Disadvantages**
    - **Requires many database scans**
    - **Assumes transaction DB is memory resident**
    - **Very slow**

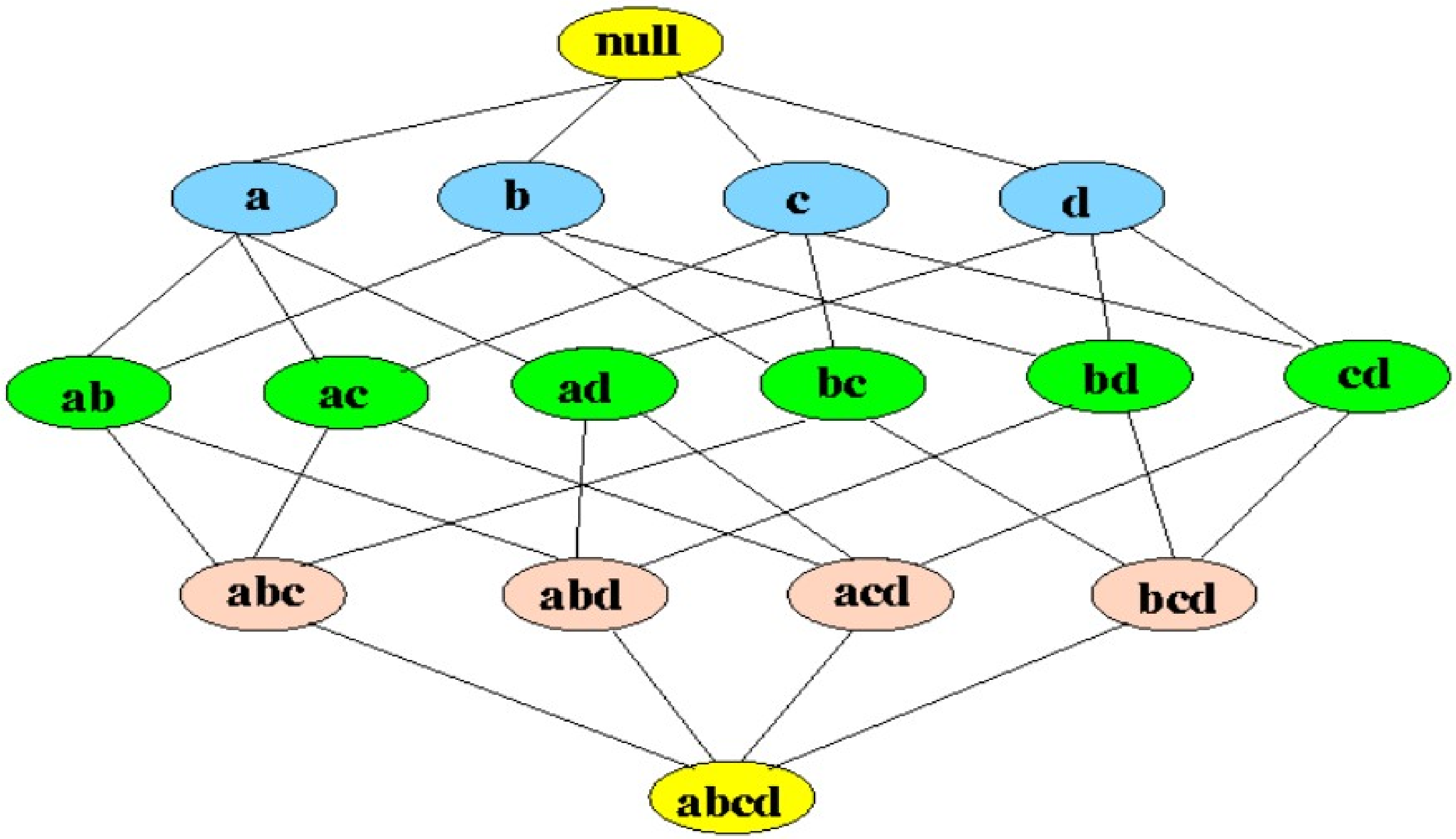
Improving the Efficiency of Apriori

- Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm.
  - Some of the variations are
    - Hash-based technique (hashing itemsets into corresponding buckets):
    - Transaction reduction (reducing the number of transactions scanned in future iterations):
    - Sampling (mining on a subset of the given data):
    - Dynamic itemset counting (adding candidate itemsets at different points during a scan):

- Interesting Properties of frequent itemsets for a given D wrt given min_sup value
  - Downward closure – any subset of a frequent set is a frequent set
  - Upward closure – any superset of an infrequent set is an infrequent set
    - Discovering all FIs and their support is significant
      - if |A| and T are large
      - where A is set of literals or items and |A| is the cardinality of A
      - & T is the transaction DB
    - If |A|=m then the number of possible distinct itemsets is 2^m

- **Interesting Properties of frequent itemsets for a given D wrt given min_sup value**
  - **Maximal Frequent Set(MFS)**
    - **An itemset is MFS**
      - **if it is a frequent set**
      - **and no superset of this is a frequent set**
    - **If we can find set of all MFS wrt min_sup then we can find all frequent sets witout extra scan of the DB**
  - **Border set**
    - **An itemset is a border set**
      - **If it is not a frequent set**
      - **but all its proper subsets are frequent sets**

- **Lattice of subsets**
  - If A={a,b,c,d} the lattice is given as
  - There are 2^{k-1} non-empty subsets of a k-item set



Lattice of Subsets :

In this lattice the set of MFSs acts as a boundary between the set of all frequent sets and set of all infrequent sets

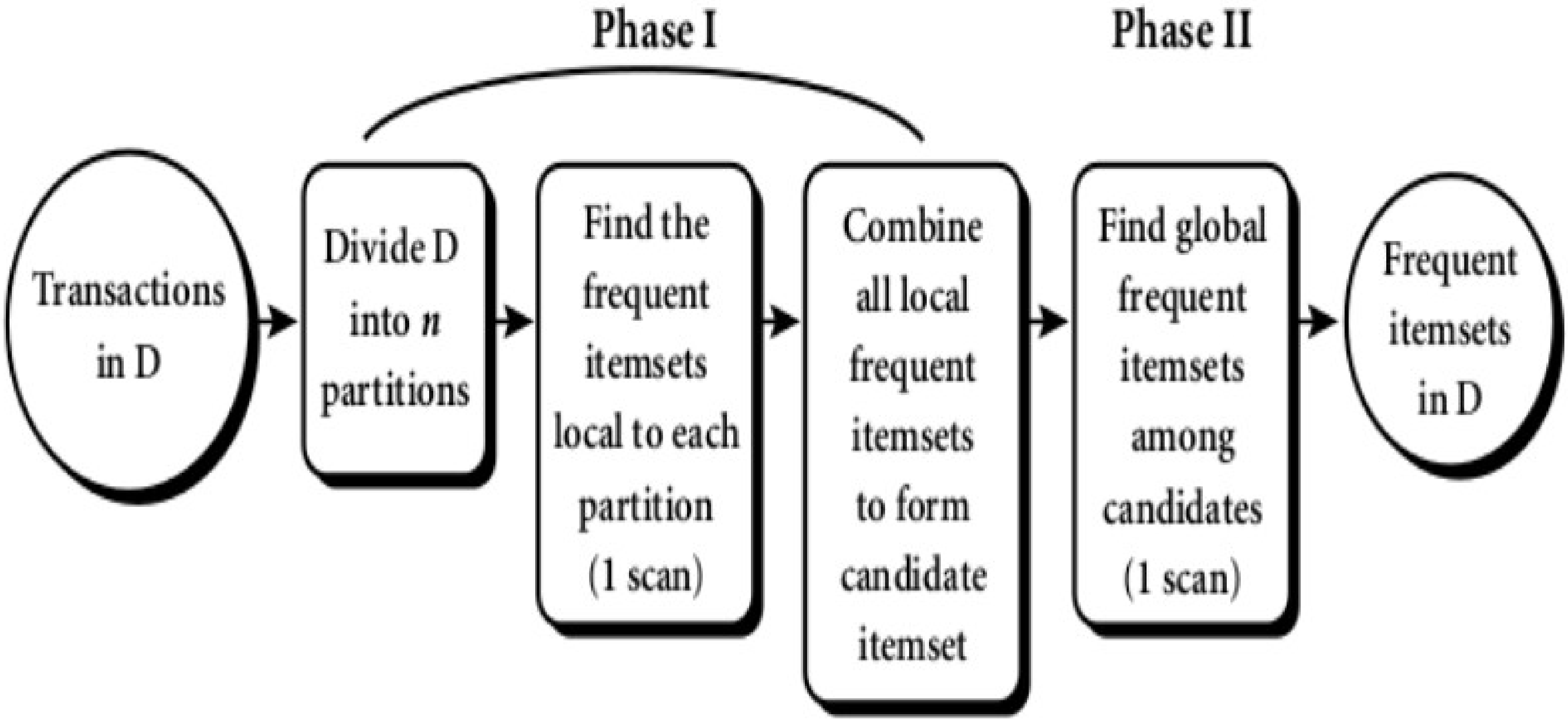
Partition Algorithm

Partitioning

- Here we discuss Partitioning Algorithm that is introduced
  - to overcome the following disadvantage of Apriori algo
    - '<u>Assumes transaction DB is memory resident</u>'
- The partition algorithm was proposed when the large transaction DB can not be accommodated in the memory

- A partitioning technique
  - doesn't require the entire DB to be in the memory
  - It requires two database scans to mine the frequent itemsets .
- The algorithm subdivides the transactions of D into '*n' nonoverlapping partitions*.

- For each partition, all frequent itemsets within the partition are found,
  - referred to as **local frequent itemsets**.
  - A local frequent itemset may or may not be frequent wrt the entire database, D.
- Any itemset that is potentially frequent wrt D must occur as a frequent itemset in at least one of the partitions.
- Therefore, all **local frequent itemsets are candidate itemsets** with respect to D.


Phase I
Phase II
Transactions in D
Divide D into n partitions
Find the frequent itemsets local to each partition (1 scan)
Combine all local frequent itemsets to form candidate itemset
Find global frequent itemsets among candidates (1 scan)
Frequent itemsets in D

- **The collection of frequent itemsets from all partitions**
  - **forms the global candidate itemsets with respect to D.**
- **Partition size and the number of partitions are set**
  - **so that each partition can fit into main memory**
  - **& therefore be read only once in each phase.**

Algorithm

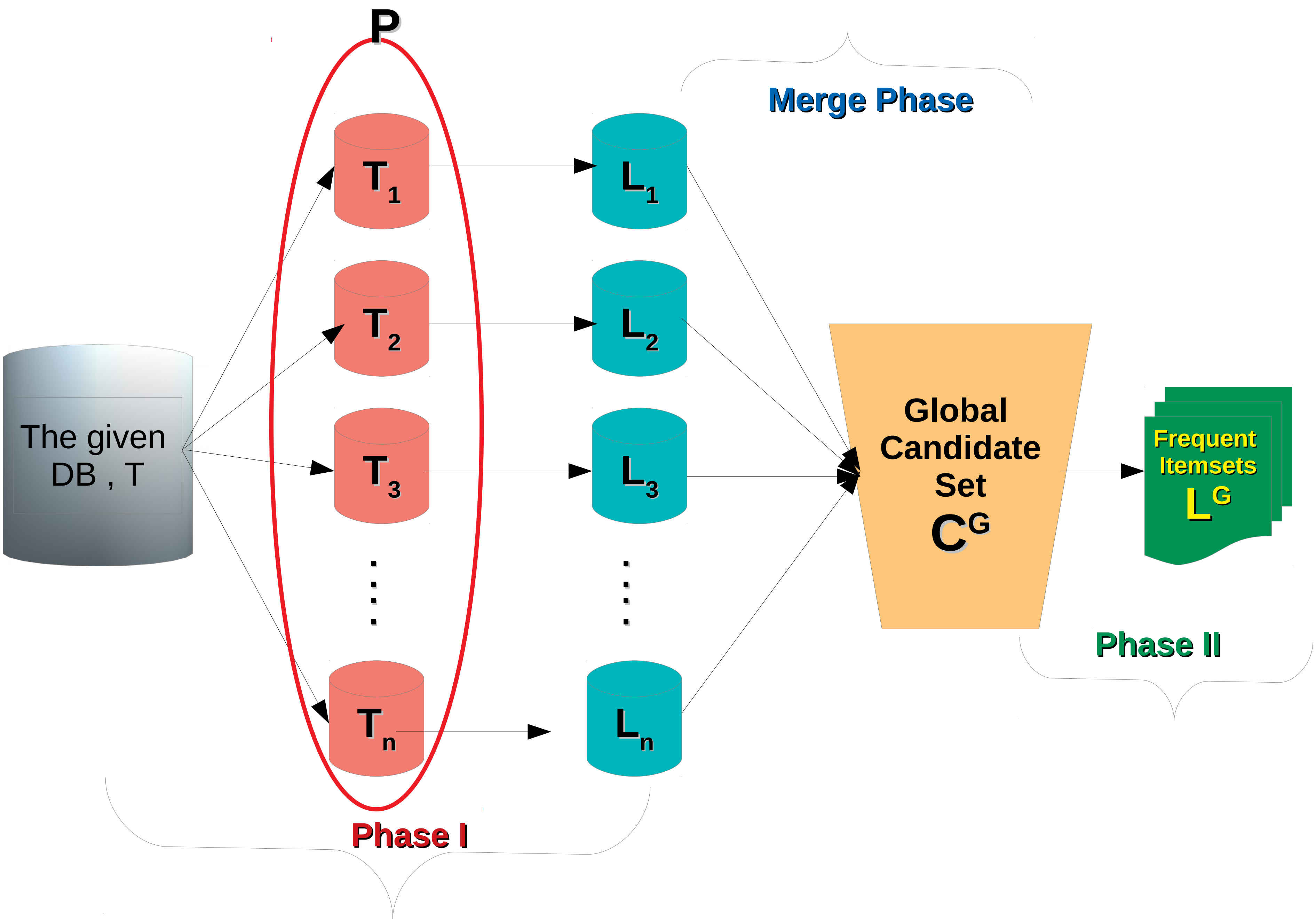
- In Partitioning the set of transactions are divided into smaller segments
- Whole segment can be read at once for calculating support values
- Two scans are used
  - One scan – to collect the frequent itemsets –
  - Next scan to count support value

Algorithm

- Two phases

- I phase (includes merge phase) :
  - divide the database into non-overlapping partitions
  - For each partiiton find the frquent itemset
  - If ‘n’ partitions $(T_1,T_2 \ldots T_n)$ – ‘n’ iterations – n local frquent itemsets $(L_1,L_2 \ldots L_n)$
  - At the end these n local frequent itemsets are merged to generate global candidates C^G

- II phase :
  - Actual support for these candidate itemsets in C^G are counted wrt entire D
  - Then identify frequent itemsets



Success is the sum of small efforts repeated day in and day out

Partition Algorithm

P = partition_database(T); n = Number of partitions

// Phase I

for $i = 1$ to n ***do begin***

read_in_partition(T_i in P)

L^i = generate all frequent itemsets of T_i using *a priori* method in main memory.

end

// Merge Phase

for $(k = 2; L_k^i \neq \varnothing, i = 1, 2, \ldots, n; k{+}{+})$ ***do begin***

$$C_k^G = \bigcup_{i=1}^{n} L_i^k$$

end

// Phase II

for $i = 1$ to n ***do begin***

read_in_partition(T_i in P)

for all candidates $c \in C^G$ compute $s(c)_{T_i}$

end

$L^G = \{c \in C^G \mid s(c)_{T_i} \geq \sigma\}$

Answer = L^G