Big Data Association Rules

- **Overview**
    - Supermarket basket analysis: What products were often purchased together?
    - Frequent items: Identify items that bought together by large number of customers
    - Process the sale transaction log to find frequent items
    - Classical example:
        - Customers, who bought diapers, have also bought beers
        - Place these items next to each other's.
    - What are the subsequent purchases after buying a PC?
    - What kinds of DNA are sensitive to this new drug?
    - Broad applications:
        - Basket data analysis, cross-marketing, catalog design, sale campaign analysis
        - Web log (click stream) analysis, DNA sequence analysis, etc.
    - Why associations:
        - Placement
        - Advertising
        - Sales
        - Coupons

- **Frequent Patterns**
    - Frequent pattern: pattern (set of items, sequence, etc.) that occurs frequently in a dataset.
    - Basic Concepts:

        A set of items: $I=\{x1, \ldots, xk\}$

        Transactions: $D=\{t1,t2, \ldots, tn\}$, $tj \subseteq I$

        A k-Itemset: $\{Ii1,Ii2, \ldots, Iik\} \subseteq I$

    - Support of an itemset:
        - Percentage of transactions that contain that itemset.
    - Large (Frequent) itemset:
        - Itemset whose number of occurrences is above a threshold.

- Basic definitions:
  - A set of items: $I=\{x_1, \ldots, x_k\}$

  - Transactions: $D=\{t_1,t_2, \ldots, t_n\}, t_j \subseteq I$

  - A k-Itemset: $\{I_{i1},I_{i2}, \ldots, I_{ik}\} \subseteq I$

  - Association Rule (AR):
    - implication $X \Rightarrow Y$

      where $X,Y \subseteq I$ and $X \cap Y = \emptyset$;
  - **Support** of itemset:
    - Support of an itemset: Percentage of transactions that contain that itemset.
    - AR (s) $X \Rightarrow Y$: Percentage of transactions that contain $X \cup Y$
  - **Confidence** of itemset:
    - AR (a) $X \Rightarrow Y$: Ratio of number of transactions that contain $X \cup Y$ to the number that contain X

  - **Association Rule Problem:**
    - Identify all association rules $X \Rightarrow Y$ with a **minimum support and confidence.**
    - Large (Frequent) itemset: Itemset whose number of occurrences is above a threshold.
  - Example:

Transactions	Items
T1	Break, Jelly, PeanutButter
T2	Bead, PeanutButter
T3	Bead, Mild, PeanutButter
T4	Beer, Bread
T5	Beer, Mild

X⇒ Y	Support	Confidence
Bread ⇒ Peanutbutter	= 3/5 %= 60%	= (3/5)/(4/5)%=75%
Peanutbutter ⇒ Bread	60%	= (3/5)/(3/5)%=100%
Jelly ⇒ Milk	0%	0%
Jelly ⇒ Peanutbutter	=1/5 % = 20%	= (1/5)/(1/5) % = 100%

- Association Rules techniques:
  - Find all frequent itemsets.
  - Generate strong association rules from the frequent itemsets:
    - those rules must satisfy minimum support and minimum confidence.
  - Regular algorithms such as A-priori, take **k passes** to find frequent itemsets of size k.
  - Can we use fewer passes?
    - Use 2 or fewer passes for all sizes
    - Random sampling
    - SON (Savasere, Omiecinski, and Navathe)
    - Toivonen

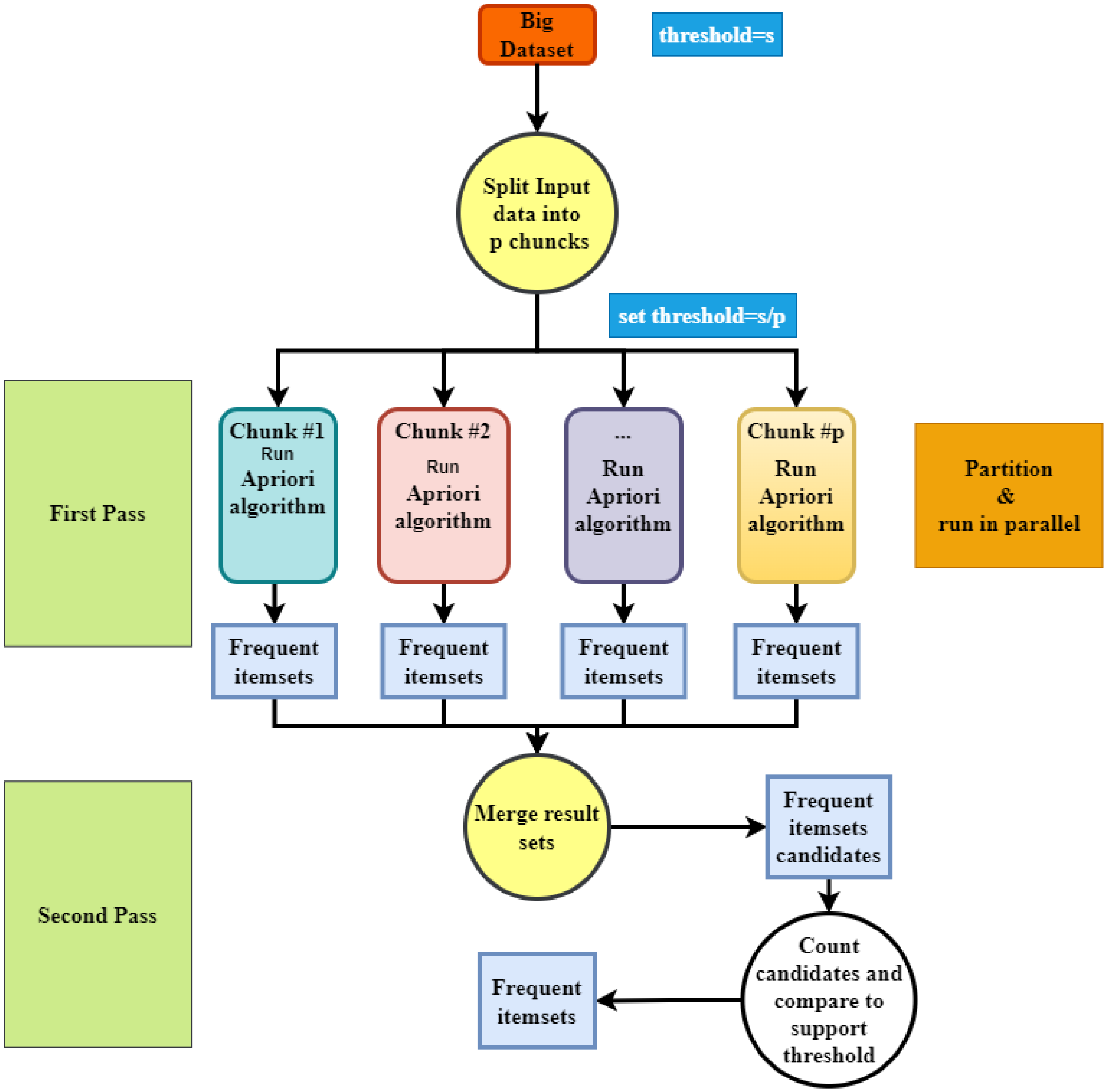
- **Random Sampling**
  - Algorithm:
    - Take a random sample of the market basket
    - Run a-priori algorithm or any other association rule algorithm in main memory
    - No disk I/O
    - Reduce support threshold proportionally to match the sample size
  - Example: if the sample size x%,
    - The support threshold=size of the basket/x

Main Memory*
Copy of sample backet
Space for counts

* Since the data is in the main memory, we can process the data as many times as we need.

- Optionally, verity that the candidate pairs are truly frequent in the data set by a second pass ➔ avoid false positive
- We may not find sets that are frequent in the whole, but not in the sample ➔ false negative
- Smaller threshold, e.g., s/20, helps find more truly frequent itemsets, but require more memory

- **SON Algorithm**
  - It is also known as ”Partition algorithm”
  - It uses parallel processing and mapreduce to find frequent itemsets in a big dataset:
    - Partition data and test each one of them.
    - Combine extracted results
    - It does parallel computing which saves time and memory
  - It uses two passes:
      - **Pass 1: Find the candidate itemsets**
        - Split the data into chunks that can be processed in main memory.
        - Read one chunk at the time
        - In parallel, find all frequent itemsets for each chunk.
          - Threshold = s/number of chunks
          - An itemset becomes a candidate if it is found to be frequent in any one or more chunks of the baskets.
      - **Pass 2: Find true frequent itemsets**
        - Verify that the itemsets are truly frequent in the entire data set to eliminate **false positives**
        - Count all the candidate itemsets and determine which itemsets are frequent in the entire set.



- Mapreduce Implementation:
  - Pass 1:
    - First Map Function:
      - Find the itemsets frequent in the subset using an association rule algorithm such as apriori using a lower threshold from s to s/p.

        - The output is a set of key-value pairs (F, 1), where F is a frequent itemset from the sample. The value is always 1 and is irrelevant.
      - First Reduce Function:
        - First Reduce Function:
          - Each Reduce task is assigned a set of keys, which are itemsets. The value 1 is ignored, and the Reduce task simply produces those keys (itemsets) that appear one or more times. Thus, the output of the first Reduce function is the candidate itemsets.
  - Pass 2:
      - Second Map Function:
        - The Map tasks take **<u>all the output from the first Reduce Function</u>** (the candidate itemsets) and a **<u>portion of the input data file</u>**.
        - Each Map task counts the number of occurrences of each of the candidate itemsets in the portion of the dataset that it was assigned.
        - The output is a set of key-value pairs (C, v), where
          C is one of the candidate itemset and
          v is the support for that itemset among the baskets that were input to this Map task.
      - Second Reduce Function:
        - The Reduce tasks take the itemsets they are given as keys and **sum the associated values**.
        - The result is the total support for each of the itemsets that the Reduce task was assigned to handle.
        - Those itemsets whose sum of values is **at least s** are frequent in the whole dataset, so the Reduce task outputs these itemsets with their counts.
        - Ignore itemsets with lower support (< s).

- Drawback:
  - We may not catch frequent itemsets in the whole dataset
    - ➔ **False Negative**
  - Smaller threshold, e.g., s /kp, where p is the number of chucks, helps catch more truly frequent itemsets.
    - ➔ But requires more space.