CSCI 4502/5502

Data Mining - Fall 2023 - Lecture 8

Ravi Starzl, PhD

Mining Frequent Patterns

1. Interestingness Measure
2. Frequent Pattern Mining
3. Frequent Itemset Mining

Ravi Starzl, PhD - Data Mining Fall 2023

Interestingness Measure

- Association rule
  - A ⇒ B [support, confidence]
- A strong association rule
  - play basketball ⇒ eat cereal [40%, 66.7%]
- The rule is misleading
  - overall, 75% of students eat cereal
  - play basketball ⇒ not eat cereal [20%, 33.3%]

Correlation Rules

- Correlation rule
  - A ⇒ B [support, confidence, correlation]
- Measure of dependent/correlated events

$$lift(A,B) = \frac{P(A \cup B)}{P(A)P(B)}$$

  - lift = 1? independent
  - lift < 1? negatively dependent
  - lift > 1? positively dependent

Example

$$lift(A,B) = \frac{P(A \cup B)}{P(A)P(B)}$$

	basketball	not basketball	sum (row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum (col)	3000	2000	5000

$$lift(B,C) = \frac{2000/5000}{(3000/5000) \times (3750/5000)} = 0.89$$

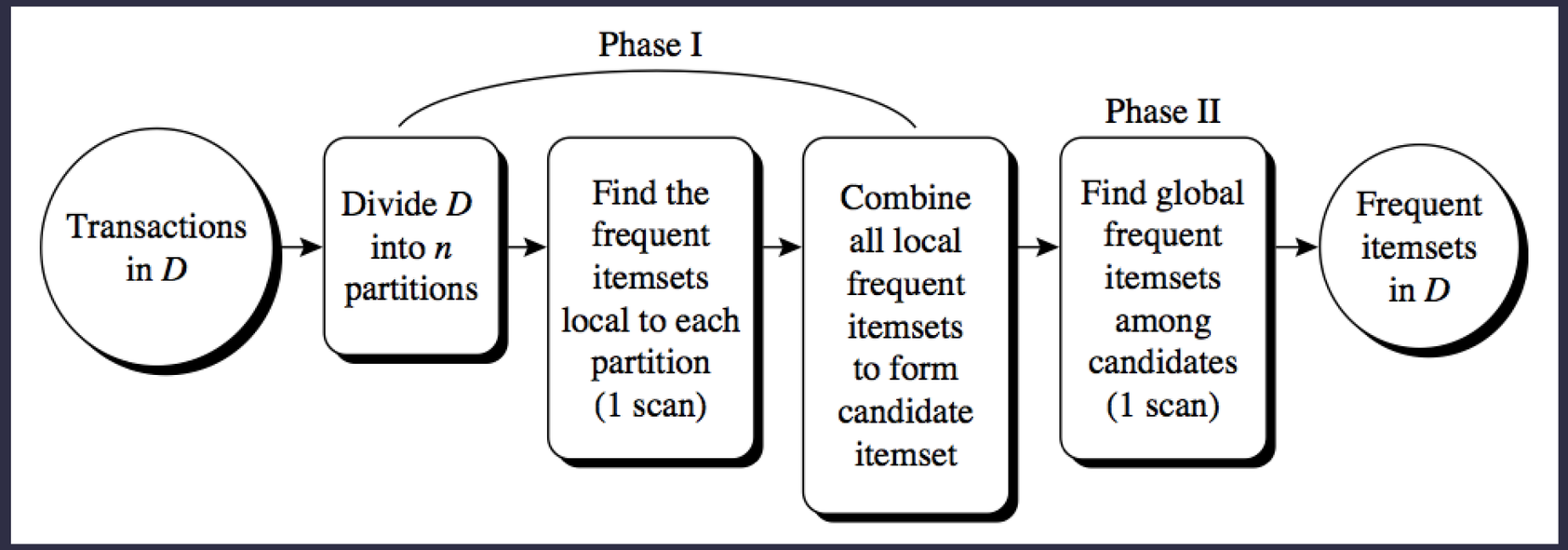
$$lift(B,\overline{C}) = \frac{1000/5000}{(3000/5000) \times (1250/5000)} = 1.33$$

Frequent Pattern Mining

- Challenges
  - multiple scans of the whole data set
  - a huge number of candidates
  - tedious support counting for candidates
- Improving Apriori: general ideas
  - reduce data scans
  - reduce number of candidates
  - facilitate support counting of candidates

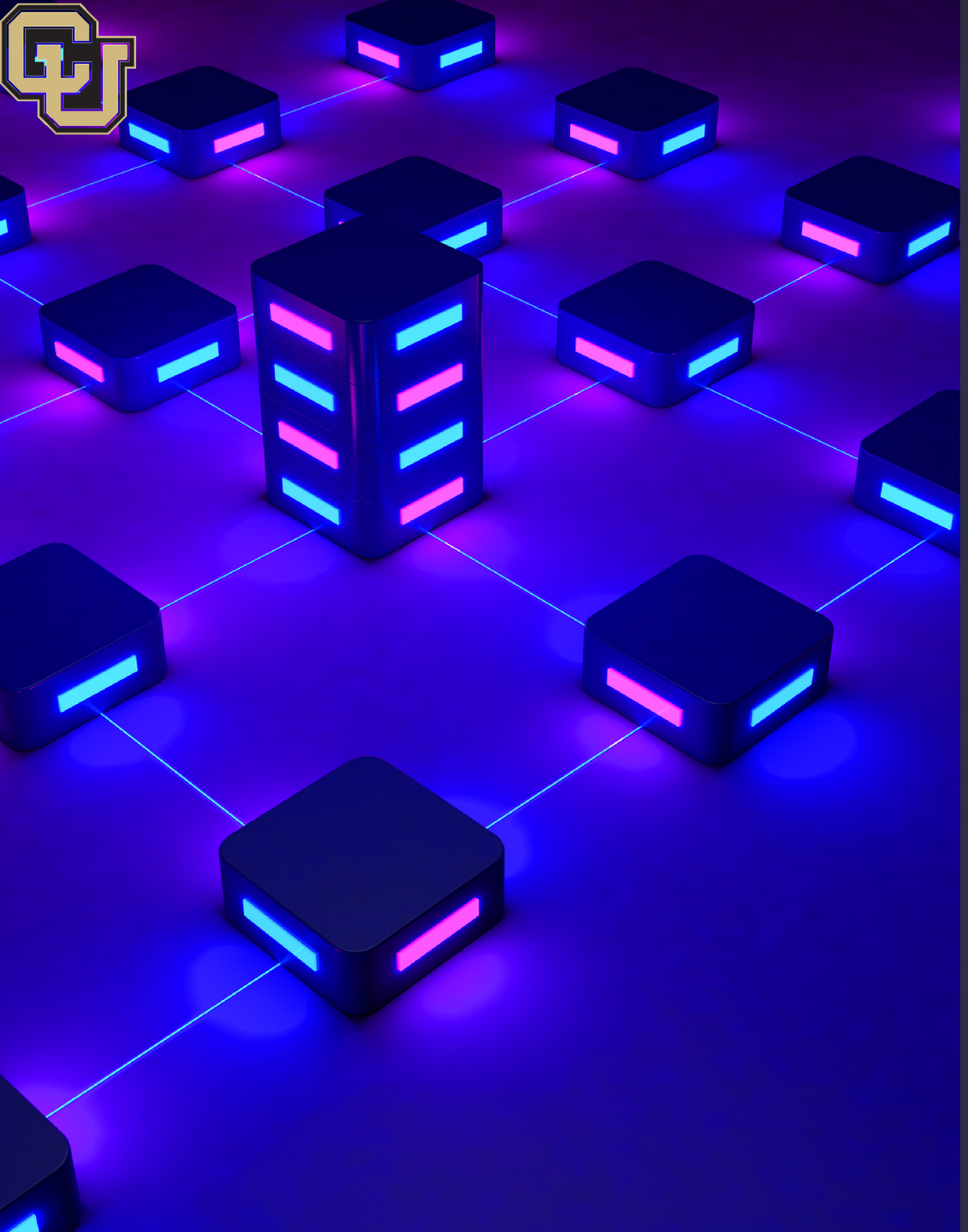
Partition: Two Data Scans

- A frequent itemset must be frequent in at least one partition
- Partition size? # of partitions?
  - each partition fits into main memory



Sampling for Frequent Patterns

- Select a sample data set
- Mine frequent patterns within sample
- may use a lower min_sup
- Scan whole data set for actual support
  - only check closed patterns
  - e.g., check abcd instead of ab, acd, ..., etc.
- Scan again to find missed frequent patterns
- Sample size?



Transaction Reduction

- If a transaction T does not contain any frequent k-itemset
  - then for any h > k, no need to check T when searching for frequent h-itemset
- Implementation
  - sequential scan
  - vs. random access

Reduce # Candidates

- Hash itemsets to buckets
- If a hash bucket count is below support threshold
  - then itemsets in that hash bucket are not frequent itemsets

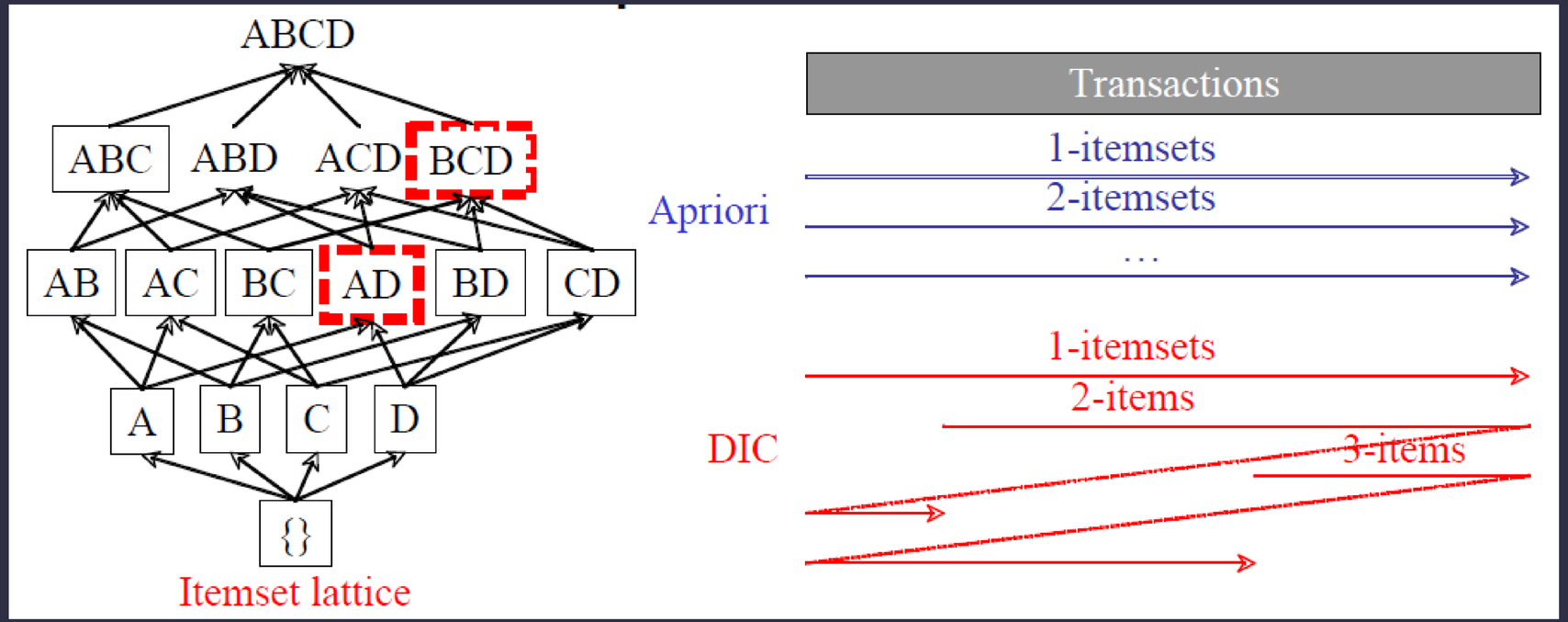
Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

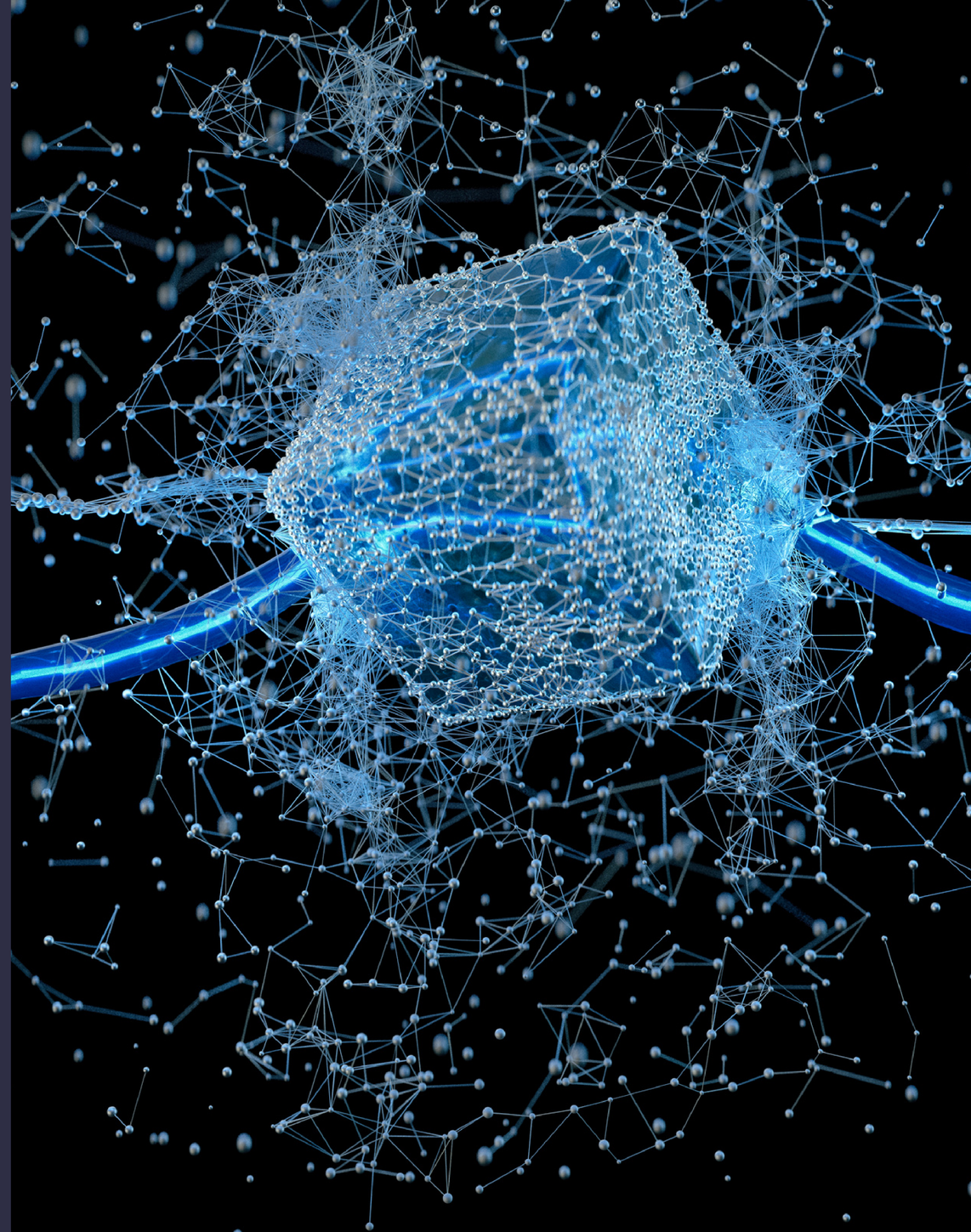
Dynamic Itemset Counting

- If A & D are freq., start count for AD
- If BC, BD, CD are freq., start count for BCD

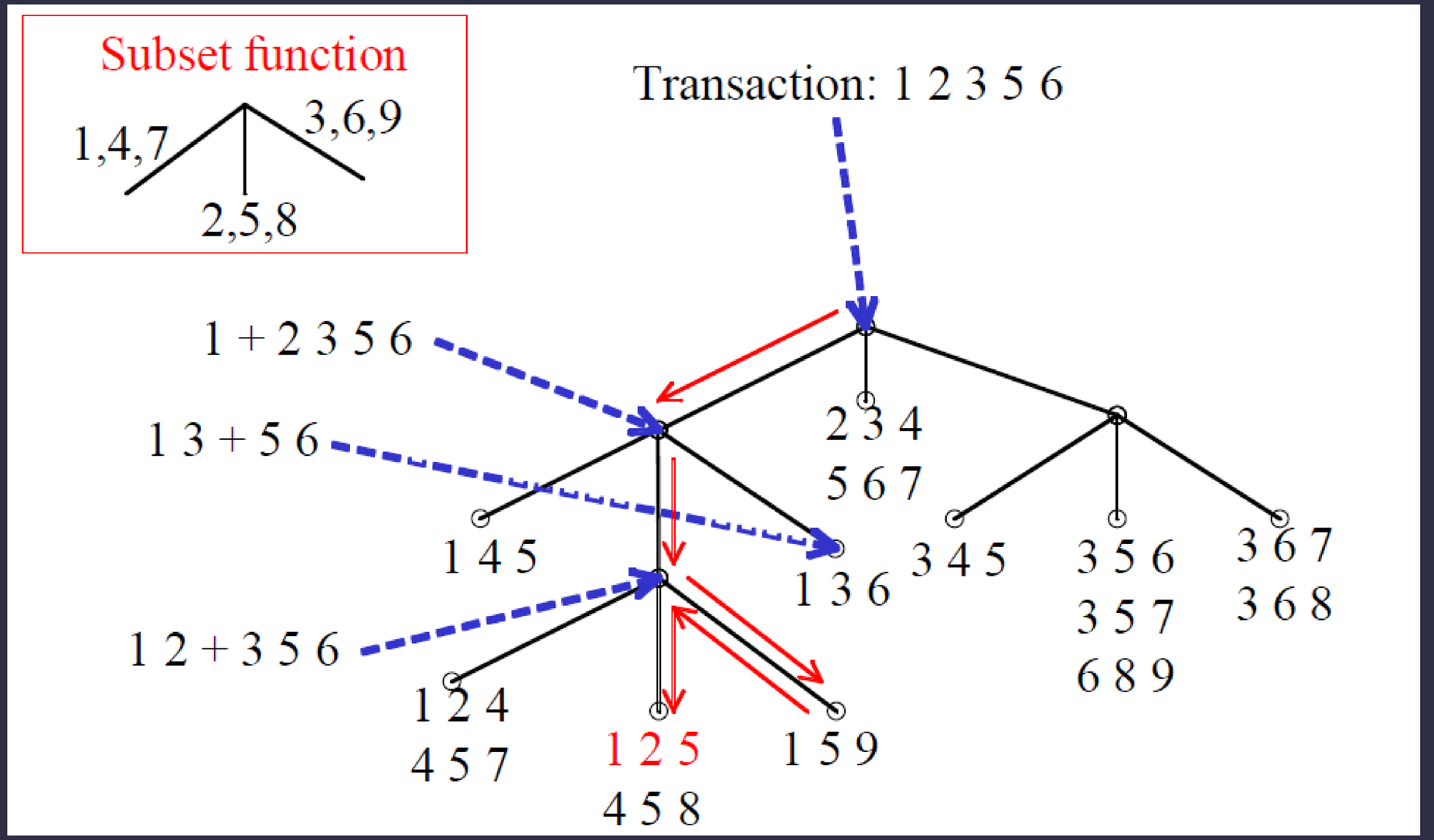


Count Support of Candidates

- Why counting candidate support a problem?
  - #candidates: total, per transaction
- Method
  - store candidate itemsets in a hash-tree
  - leaf-node contains a list of itemsets and counts
  - interior node contains a hash table
  - subset function: finds all candidates contained in a transaction



Example


Subset function
1,4,7
3,6,9
2,5,8
Transaction: 1 2 3 5 6
1 + 2 3 5 6
1 3 + 5 6
1 2 + 3 5 6
1 4 5
1 3 6
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
2 3 4
5 6 7
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8

Vertical Data Format

- Horizontal data format
  - T1: {A, D, E, F}
- Vertical data format
  - t(AD) = {T1, T6, ...}
- Derive closed pattern via vertical intersection
  - t(X) = {T1, T2, T3} and t(Y) = {T1, T3, T4}
  - t(XY) = {T1, T3}

Frequent Itemset Mining

- Multiple data scans are costly
- Mining long patterns needs many scans and generates lots of candidates
  - e.g., 100 items: #scans, #candidates
- Bottleneck: candidate generation & test
- Can we avoid candidate generation?

FP-growth (1)

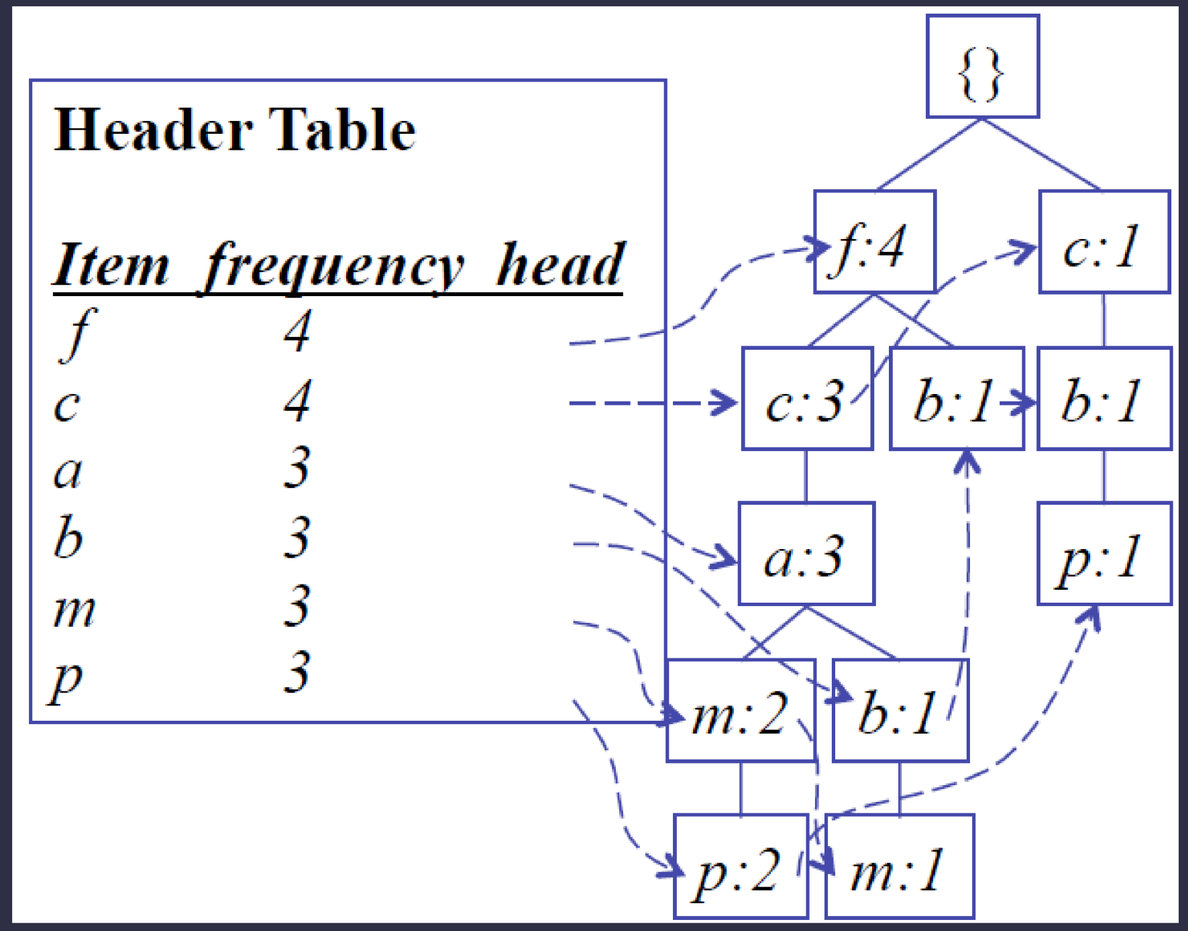
- Find frequent itemsets without candidate generation
- Grow long patterns from short ones using local frequent items
- Example
  - abc is a frequent itemset; get all transactions with abc: DB | abc
  - d is a local frequent item in DB | abc
  - then abcd is a frequent itemset

FP-tree Construction

- min_sup = 0.6

TID | Items bought | (ordered) frequent items
100 | {f, a, c, d, g, i, m, p} | {f, c, a, m, p}
200 | {a, b, c, f, l, m, o} | {f, c, a, b, m}
300 | {b, f, h, j, o, w} | {f, b}
400 | {b, c, k, s, p} | {c, b, p}
500 | {a, f, c, e, l, p, m, n} | {f, c, a, m, p}

- Scan, find freq. 1-itemset
- Sort freq. items in descending frequency
- Scan, construct FP-tree

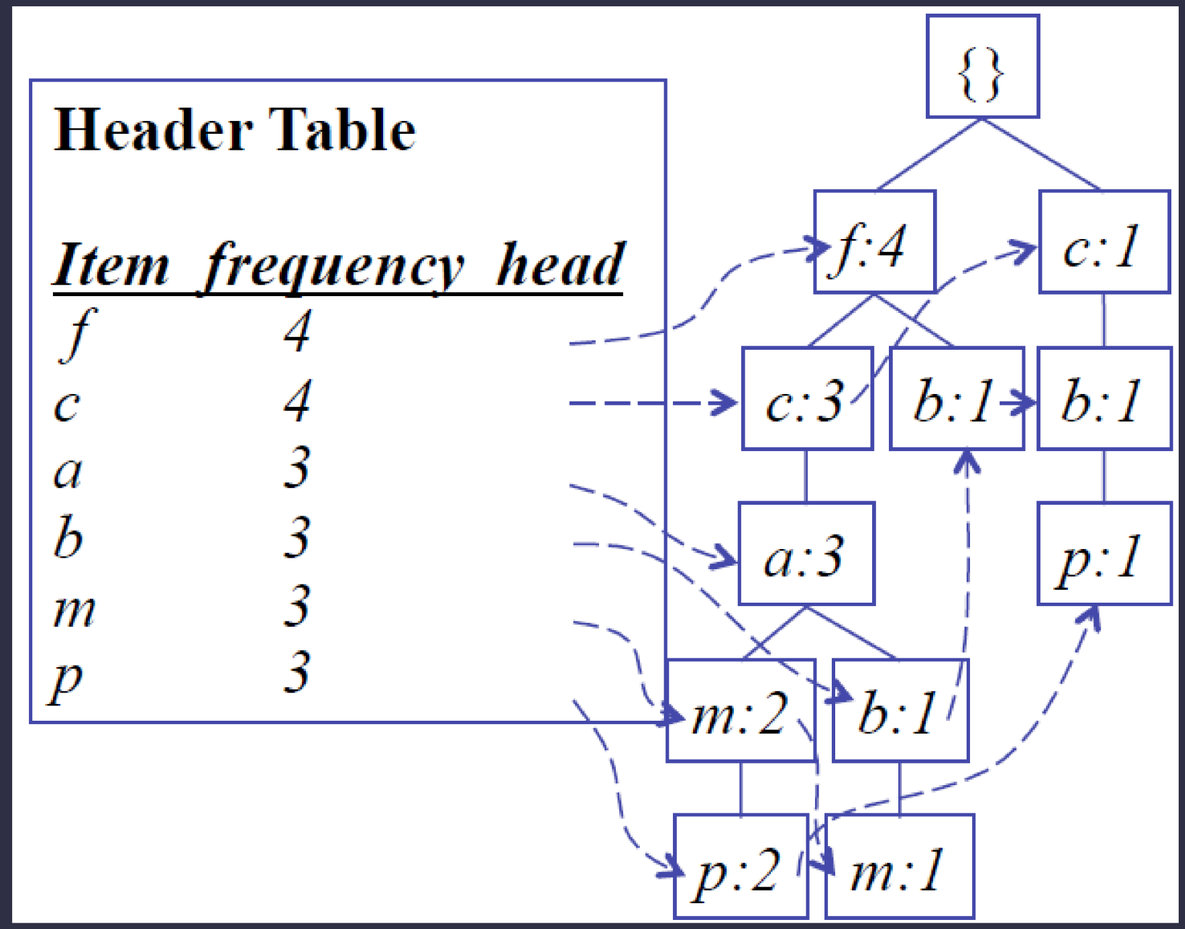


Conditional Pattern Base

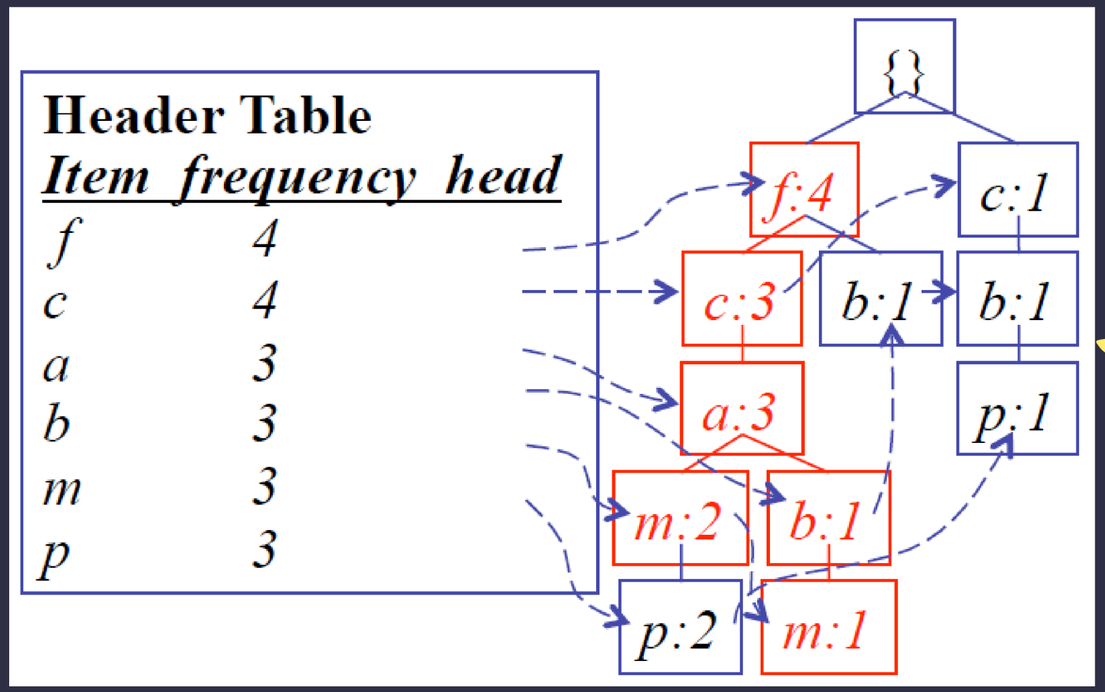
- Traverse links of each frequent item, prefix paths

Conditional pattern bases

item | cond. pattern base
c | f:3
a | fc:3
b | fca:1, f:1, c:1
m | fca:2, fcab:1
p | fcam:2, cb:1



Conditional FP-trees



m-conditional pattern base:

fca:2, fcab:1

{}

f:3

c:3

a:3

m-conditional FP-tree

All frequent patterns relate to m

m,

fm, cm, am,

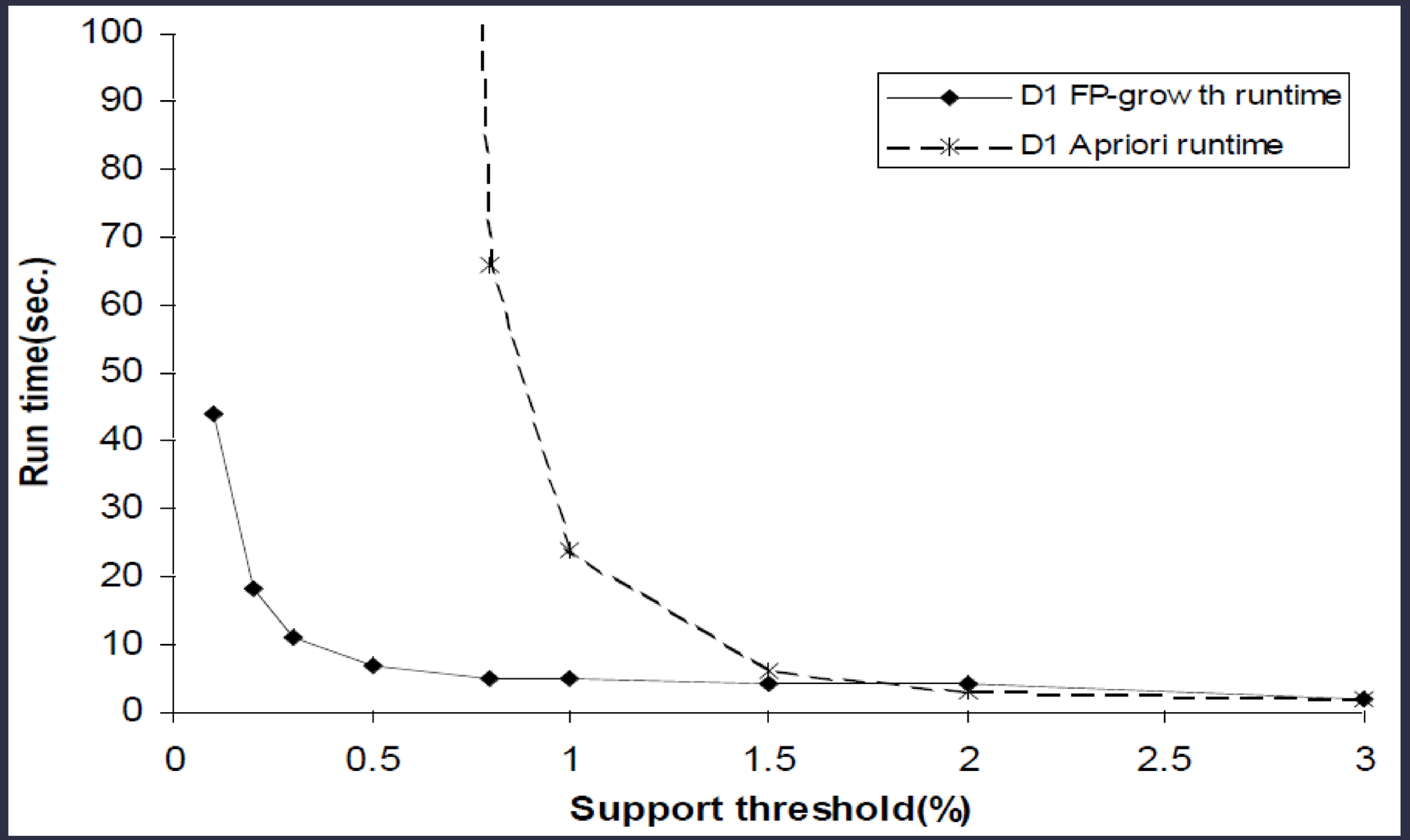
fcm, fam, cam,

fcam

FP-growth (2)

- Idea: Frequent pattern growth
  - recursively grow freq. patterns by pattern and data partition
- Method
  - freq. item => conditional pattern base => conditional FP-tree
  - repeat on each newly created FP-tree
  - until FP-tree is empty or single path

FP-growth vs. Apriori



Summary

- Partition: Two Data Scans
- Transaction Reduction
- Reduce Hash Candidates
- Dynamic Itemset Counting
- Count Support of Candidates
- Vertical Data Format
- Frequent Pattern Mining
- FP-Growth
- FP-Tree
- Conditional Pattern Base
- Conditional FP-trees

Thank you

A special thank you to Qin Lv for her slides,
on which this lecture is based