Frequent Itemsets & Association Rules

Lecture 2

Agenda

1. Sets, frequent itemsets, association rules
   – Formalism
   – Why is it useful?

2. What makes a rule interesting?
   – Support, Confidence
   – Lift

3. Generating association rules
   – Confidence pruning
   – Frequent itemsets?

4. Apriori
   – Algorithm
   – Issues

5. FP-Growth
   – Algorithm
   – Issues, comparison with Apriori

6. Representative itemsets
   – Maximal, closed

Association Analysis

Input

Trans. ID	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, OJ
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, OJ

Output

- {diapers} $\Rightarrow$ {wine}

Plagiarism: documents = {sentences}

Drug Trials: patients = {side effects, +/-medication}

Politics: politician = {voting, party}

Some Definitions

- Items
- Transactions
  (N is the size)

$$\mathcal{I} = \{i_1, i_2, \ldots, i_M\}$$

$$\mathcal{D} = \{t | t \in (\mathrm{TID}, X), X \subseteq \mathcal{I}\}$$

$$N := |D|$$

- Association rule

$$X \Rightarrow Y$$

$$X \subset \mathcal{I},$$

$$Y \subset \mathcal{I},$$

$$X \cap Y = \emptyset$$

What Makes a Rule “Interesting”?

- Let’s define items that frequently co-occur as **frequent itemsets**
- So then an “interesting” rule would be one that identifies a strong relationship between frequently co-occurring items
  - Not always true, but a good starting point :)

Itemset Support

- Let's define the following function as the number of times a particular itemset occurs within a dataset

$$\sigma(X) = |\{t | t \in \mathcal{D}, X \subseteq t\}|$$

- Then the **support** of an itemset is the percentage of the dataset that contains the itemset

$$s(X) = \frac{\sigma(X)}{N}$$

Quick Check

Input

Trans. ID	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, OJ
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, OJ

Itemsets with $\sigma \geq 3$?

Output

{soy milk}: 4
{lettuce}: 4
{diapers}: 4
{wine}: 3
{soy milk, lettuce}: 3
{soy milk, diapers}: 3
{lettuce, diapers}: 3
{diapers, wine}: 3

Quick Check

Input

Trans. ID	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, OJ
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, OJ

s({diapers, wine}) = ?

Output

3 / 5 = 0.6

Association Rule Support

- The support of an association rule is the support of the comprising elements

$$s(X \Rightarrow Y) = s(X \cup Y) = \frac{\sigma(X \cup Y)}{N}$$

- What does this say about association rules derived from frequent itemsets (i.e. those for which $\sigma(X) \geq \sigma_{min}$)?

Quick Check

Input

Trans. ID	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, OJ
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, OJ

Output

3 / 5 = 0.6

i.e. support is a ***symmetric*** *measure*

s({diapers, wine}) = ?

s({diapers} ⇒ {wine}) = ?

s({wine} ⇒ {diapers}) = ?

Association Rule Confidence

- The **confidence** of an association rule is *one* measure of the strength of co-occurrence between the *antecedent* (X) and *consequent* (Y)
- It measures the probability of Y occurring given that X has

$$c(X \Rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Quick Check

Input

Trans. ID	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, OJ
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, OJ

c({diapers} ⇒ {wine}) = ?

c({wine} ⇒ {diapers}) = ?

Output

3 / 4 = 0.75

3 / 3 = 1.0

i.e. confidence is an ***asymmetric*** *measure*

Quick Check

Describe a situation in which an association rule has a high confidence, but is likely uninteresting (i.e. does not reflect co-occurrence)

Trans. ID	Items
0	a, b
1	a
2	a
3	a
4	a

$$c(a \Rightarrow b) = 0.2$$

$$c(b \Rightarrow a) = 1.0$$

Measures of Interest

- There are many other measures
  - See TSK:6.7
- Each measure has *properties* that, in a particular task, may legitimize its use
  - Content-oriented: relates to classes of rules (not) identified by the measure
  - Algorithmic: relates to its efficient automation (e.g. [anti-]monotonicity)

Example: Lift

- The **lift** of an association rule is similar to confidence, but also includes the support of the consequent (note: 1=independent)

$$
\begin{aligned}
lift(X \Rightarrow Y) &= \frac{c(X \Rightarrow Y)}{s(Y)} \\
&= \frac{s(X \cup Y)}{s(X) \cdot s(Y)} \\
&= \frac{\sigma(X \cup Y) \cdot N}{\sigma(X) \cdot \sigma(Y)}
\end{aligned}
$$

Quick Check

Input

Trans. ID	Items
0	a, b
1	a
2	a
3	a
4	a

c({a} ⇒ {b}) = ?
c({b} ⇒ {a}) = ?

lift({a} ⇒ {b}) = ?
lift({b} ⇒ {a}) = ?

Output

1 / 5 = 0.2

1 / 1 = 1.0

(1 * 5) / (1 * 5) = 1

(1 * 5) / (5 * 1) = 1

Mining Association Rules (1)

- **Problem**. Given a dataset, find all association rules that have support ≥ s and confidence ≥ c
- Algorithm sketch…
  - For all possible rules, keep those with support ≥ s and confidence ≥ c

Quick Check

Input

Trans. ID	Items
0	a, b
1	a, b, c, d
2	a, b, c
3	c, d
4	a, b, d

Output

2 / 5 = 0.4

All association rules derived from a common itemset have the same support!

$s(\{a\} \Rightarrow \{b, c\}) = ?$

$s(\{b\} \Rightarrow \{a, c\}) = ?$

$s(\{c\} \Rightarrow \{a, b\}) = ?$

$s(\{a, b\} \Rightarrow \{c\}) = ?$

$s(\{a, c\} \Rightarrow \{b\}) = ?$

$s(\{b, c\} \Rightarrow \{a\}) = ?$

Algorithmic Optimization #1

In order to avoid redundant computation, it is common to decompose the problem into two subparts: first collect all itemsets with sufficient support, and *then* find associated rules with sufficient confidence (since they are already guaranteed to have sufficient support)

Mining Association Rules (2)

- **Problem**. Given a dataset, find all association rules that have support ≥ *s* and confidence ≥ *c*

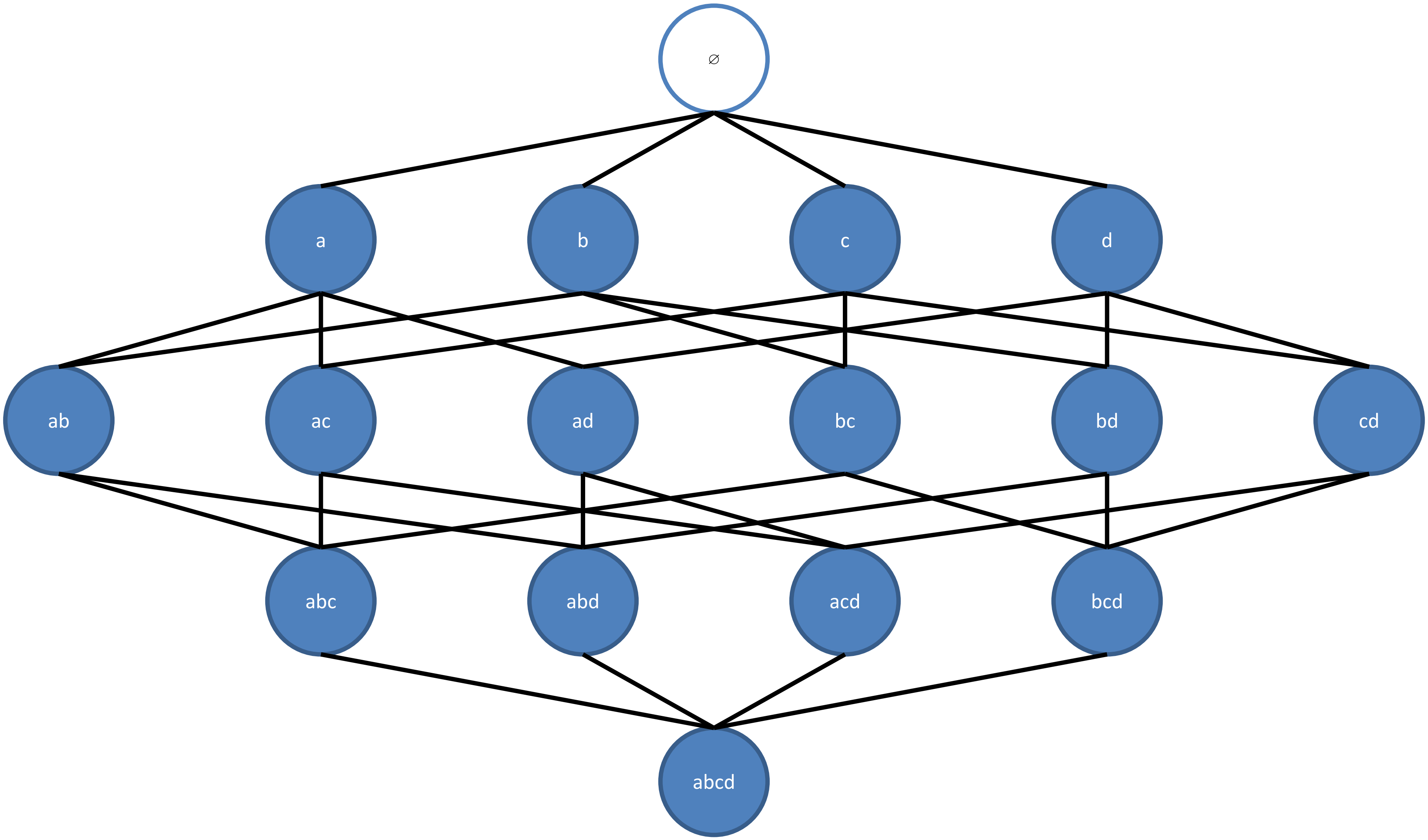
- Algorithm sketch…
  1. Given dataset D, find frequent itemsets F
     a) For all distinct itemsets, count occurrences
     b) F = itemsets with support ≥ s
  2. Given F, find interesting rules R
     a) For each frequent itemset
        – For all possible rules, keep those with confidence ≥ c

Focus 1a: “For all distinct itemsets”

Given 4 items (a, b, c, d) – how many distinct itemsets are there?

- Size 1 (4): a, b, c, d
- Size 2 (6): ab, ac, ad, bc, bd, cd
- Size 3 (4): abc, abd, acd, bcd
- Size 4 (1): abcd
- Total: 15

Itemset Lattice for $\mathcal{I}$ = {a, b, c, d}

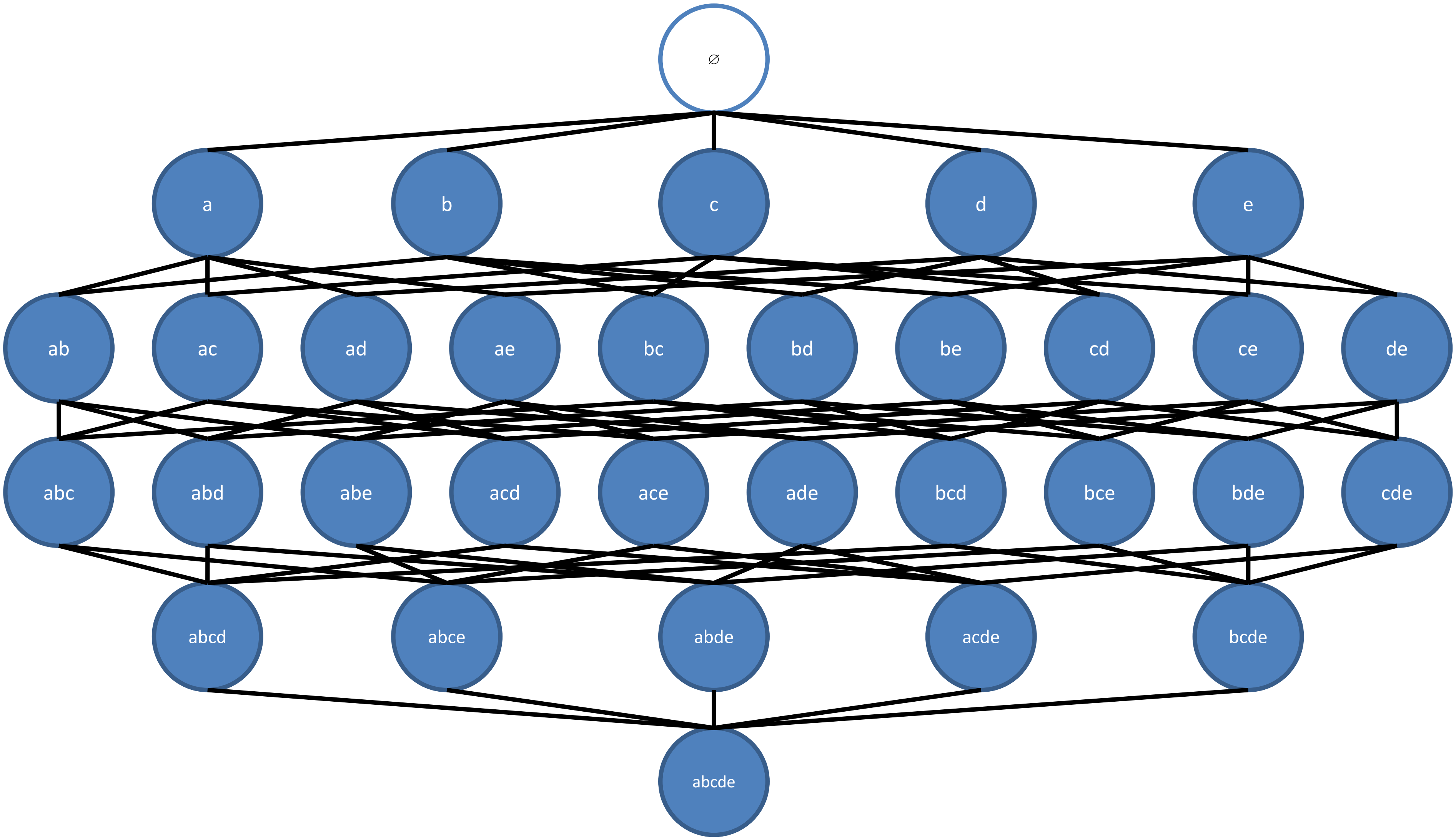


Focus 1a: "For all distinct itemsets"

Given 5 items (a, b, c, d, e) – how many distinct itemsets are there?

- Size 1 (5): a, b, c, d, e
- Size 2 (10): ab, ac, ad, ae, bc, bd, be, cd, ce, de
- Size 3 (10): abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde
- Size 4 (5): abcd, abce, abde, acde, bcde
- Size 5 (1): abcde
- Total: 31

Itemset Lattice for $\mathscr{I}$ = {a, b, c, d, e}



Focus 1a: "For all distinct itemsets"

In general, given k items – how many distinct itemsets are there?

2^k-1, "Proof": encode presence/absence of each item as a binary variable – count the number of distinct binary strings (excluding all zeros = null)

So generating frequent itemsets is combinatorial in the worst case (i.e. hard). ***We will revisit this in a bit*** for algorithms that prune the search space (Apriori) and compress information from repeated item subsets (FP-Growth).

Candidate Association Rules

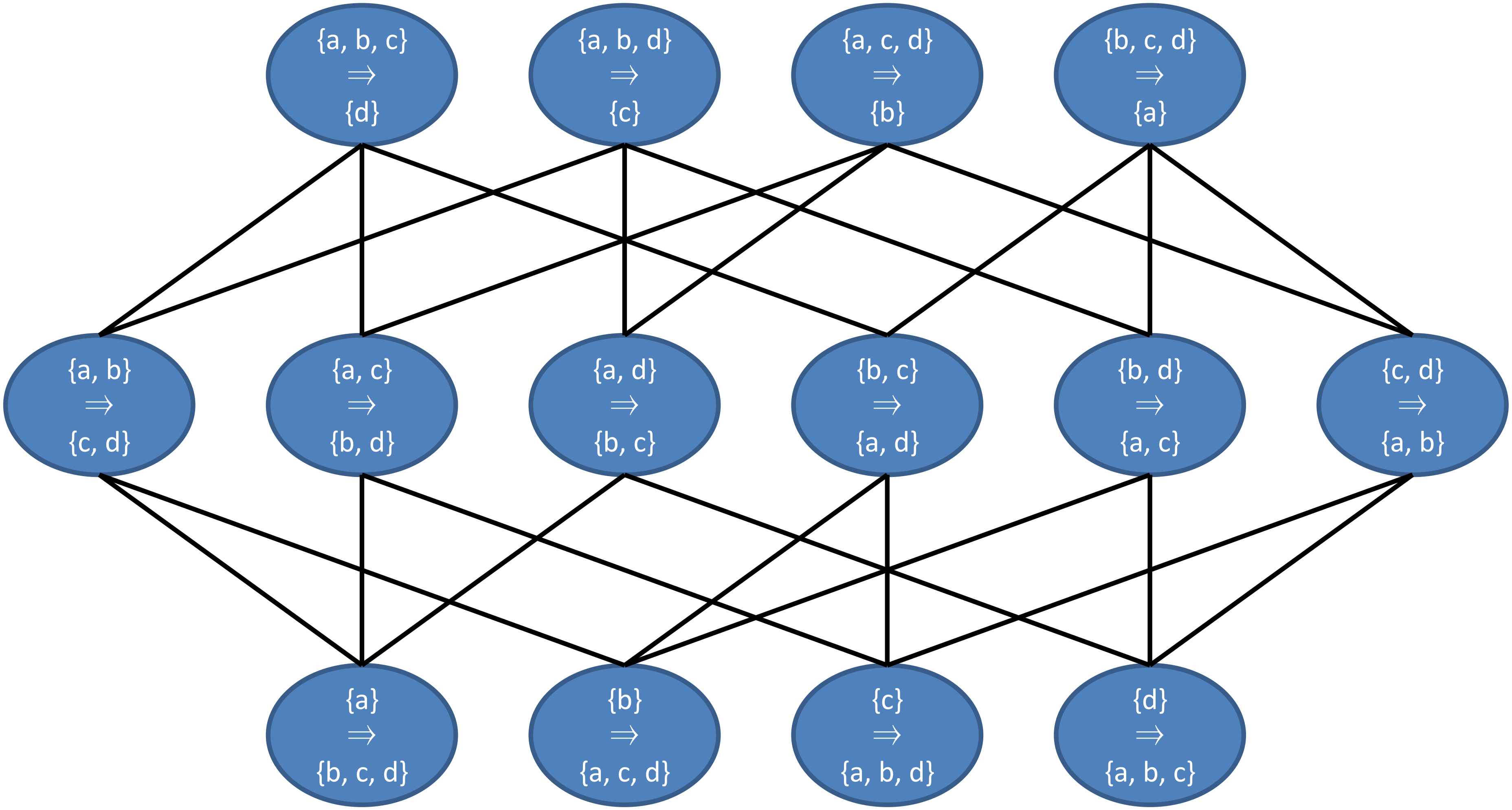
- Given frequent itemset F={a, b, c}, what are all the candidate rules (X ⇒ F-X) that could be generated
  - Exclude… F ⇒ ∅, ∅ ⇒ F

{a}⇒{b,c}, {b}⇒{a,c}, {c}⇒{a,b},
{a,b}⇒{c}, {a,c}⇒{b}, {b,c}⇒{a}

- In general, given a frequent itemset of size *k*, how many candidate rules could be generated?

2^k-2

Rule Lattice for F={a, b, c, d}



Now Check Confidence

- Quick check: $c(X \Rightarrow Y)=?$

$$c(X \Rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

- Algorithm sketch…
  - For each frequent itemset, F
    - Generate all candidate association rules
      - For each candidate association rule R
        - » If $c(R) \geq c_{min}$, keep R

Can We Do Better?

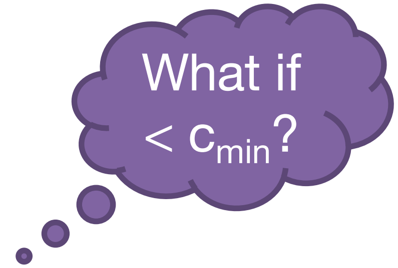
- **Claim**
  - IF $c(\{a, b, c\} \Rightarrow \{d\}) < c_{min}$
  - THEN $c(\{a,b\} \Rightarrow \{c,d\}) < c_{min}$
- Why?

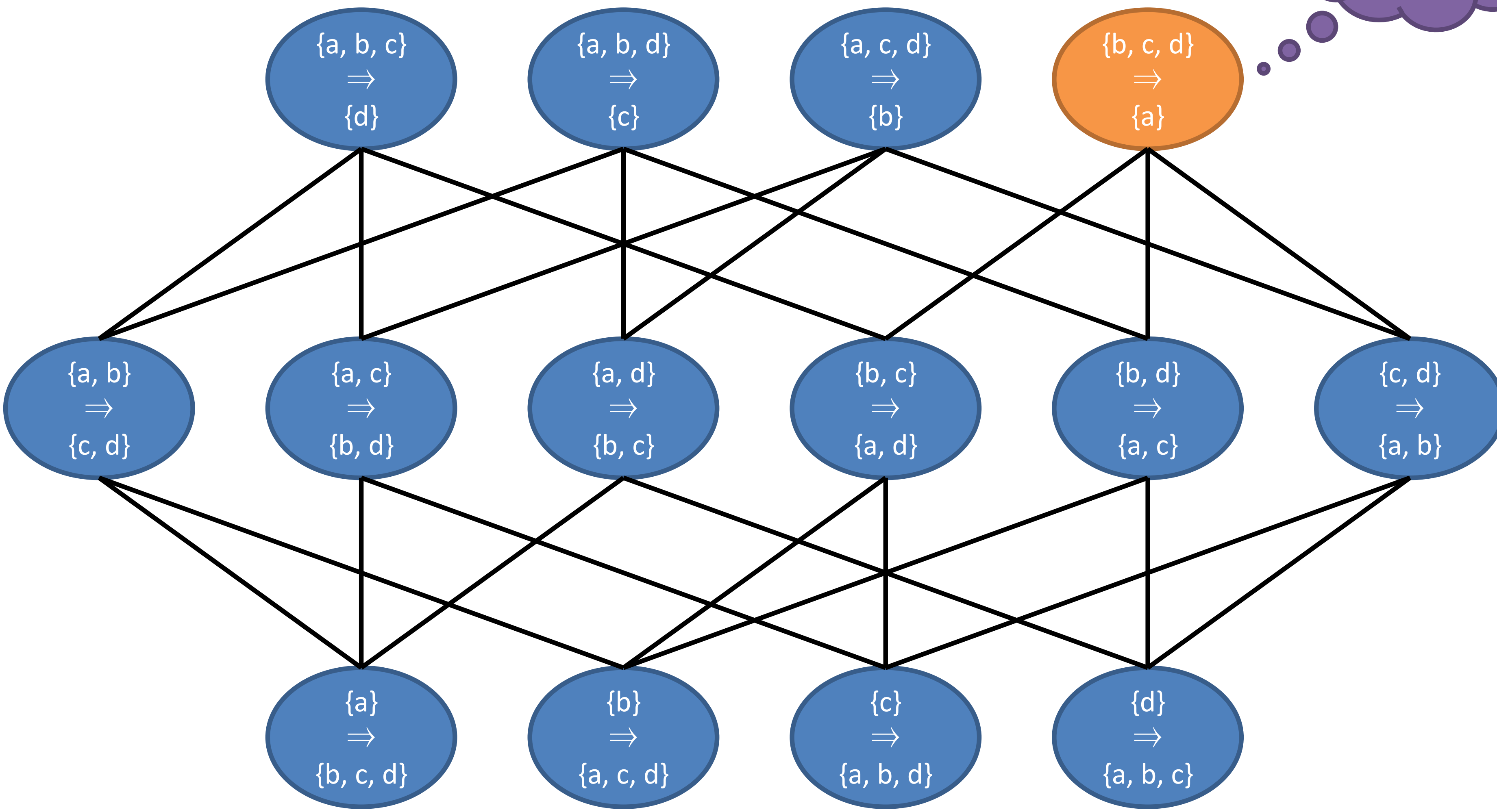
$$c(X \Rightarrow F - X) = \frac{\sigma(F)}{\sigma(X)}$$

Does this change?

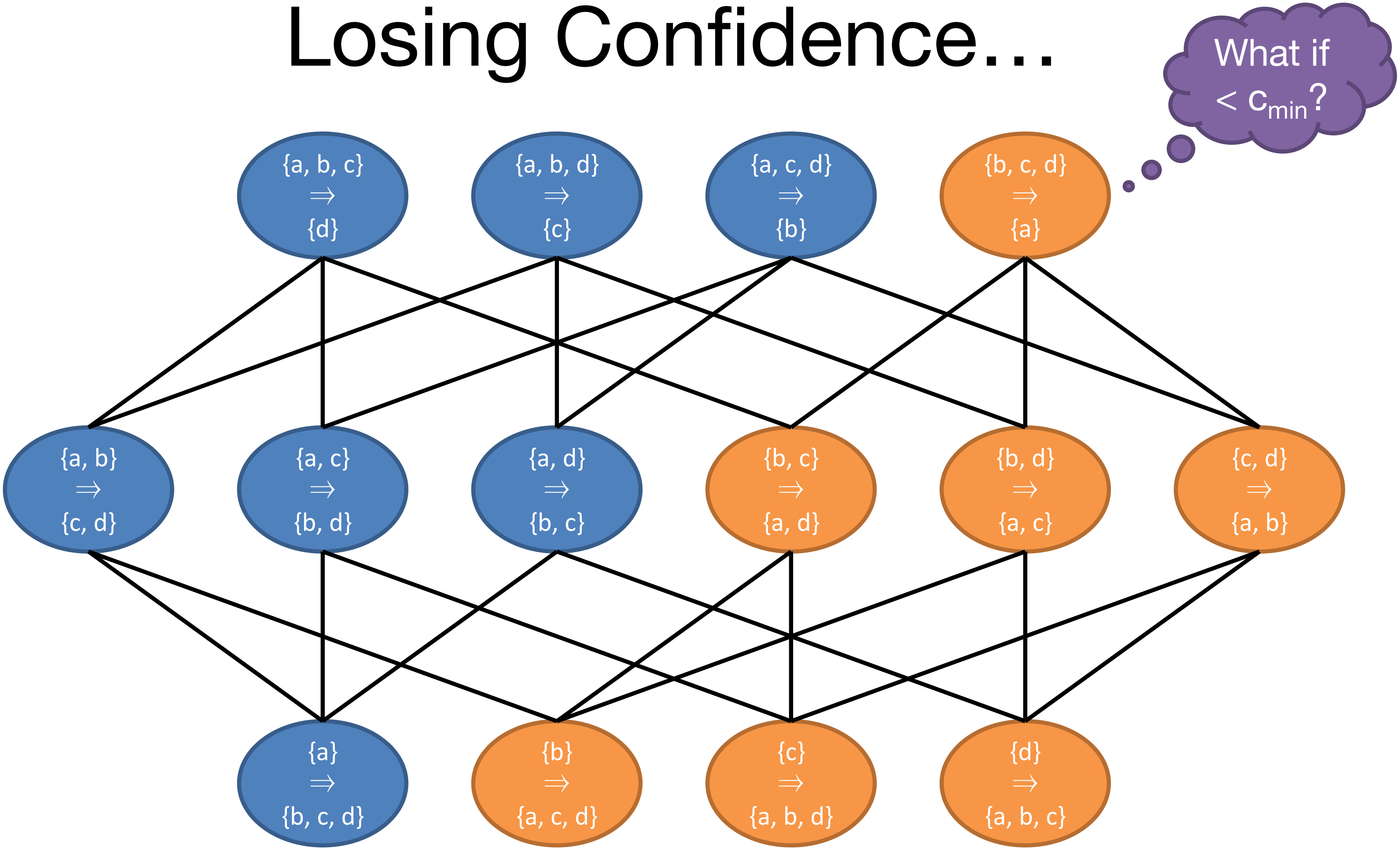
How can this change if $X' \subset X$

Losing Confidence…





Losing Confidence…



Confidence-Based Pruning

- *If a rule $X \Rightarrow Y - X$ does not satisfy the confidence threshold, then any rule $X' \Rightarrow Y - X'$, where X' is a subset of X, must not satisfy the confidence threshold as well.*

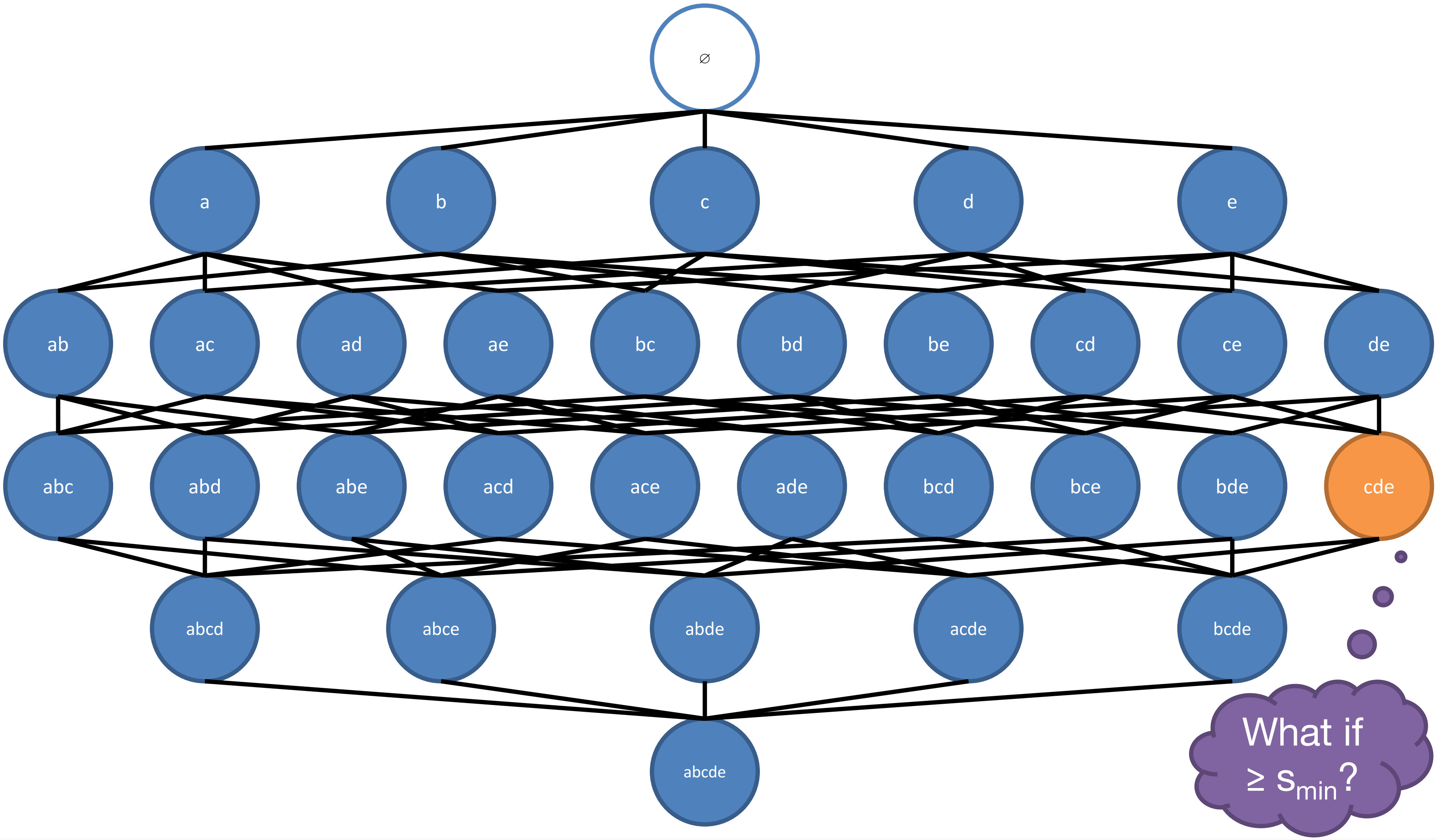
- Start with “big” antecedents ($F \Rightarrow \{\}$)
  - Candidates = antecedent minus each of the single elements
  - Check confidence, if above threshold recurse

Mining Association Rules (3)

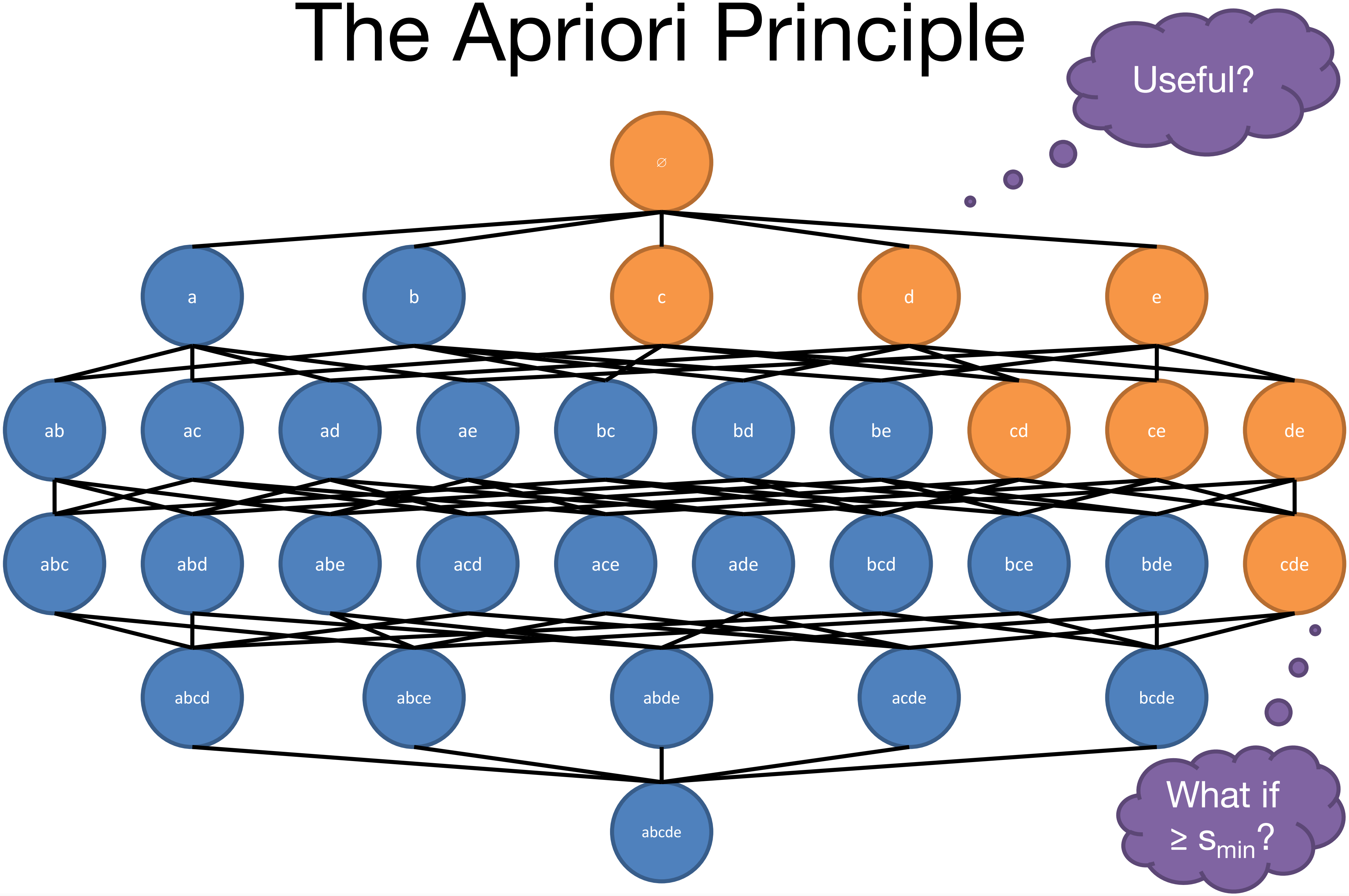
- **Problem**. Given a dataset, find all association rules that have support ≥ *s* and confidence ≥ *c*

- Algorithm sketch…
  1. Given dataset D, find frequent itemsets F
     a) For all distinct itemsets, count occurrences
     b) F = itemsets with support ≥ s
  2. Given F, find interesting rules R
     a) For each frequent itemset
        – Keep those with confidence ≥ c (incrementally pruning antecedent subsets on confidence)

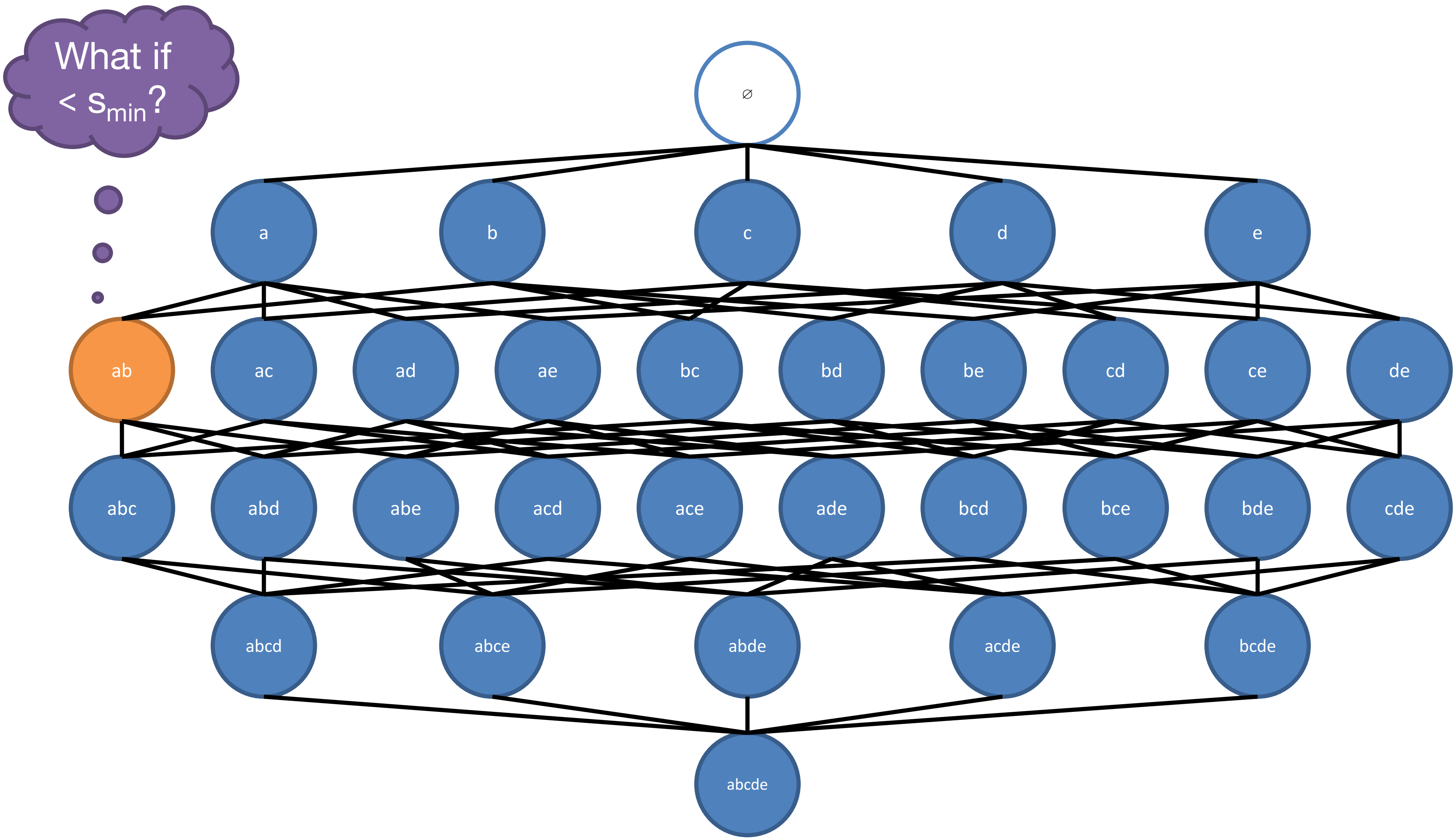
Revisiting the Itemset Lattice



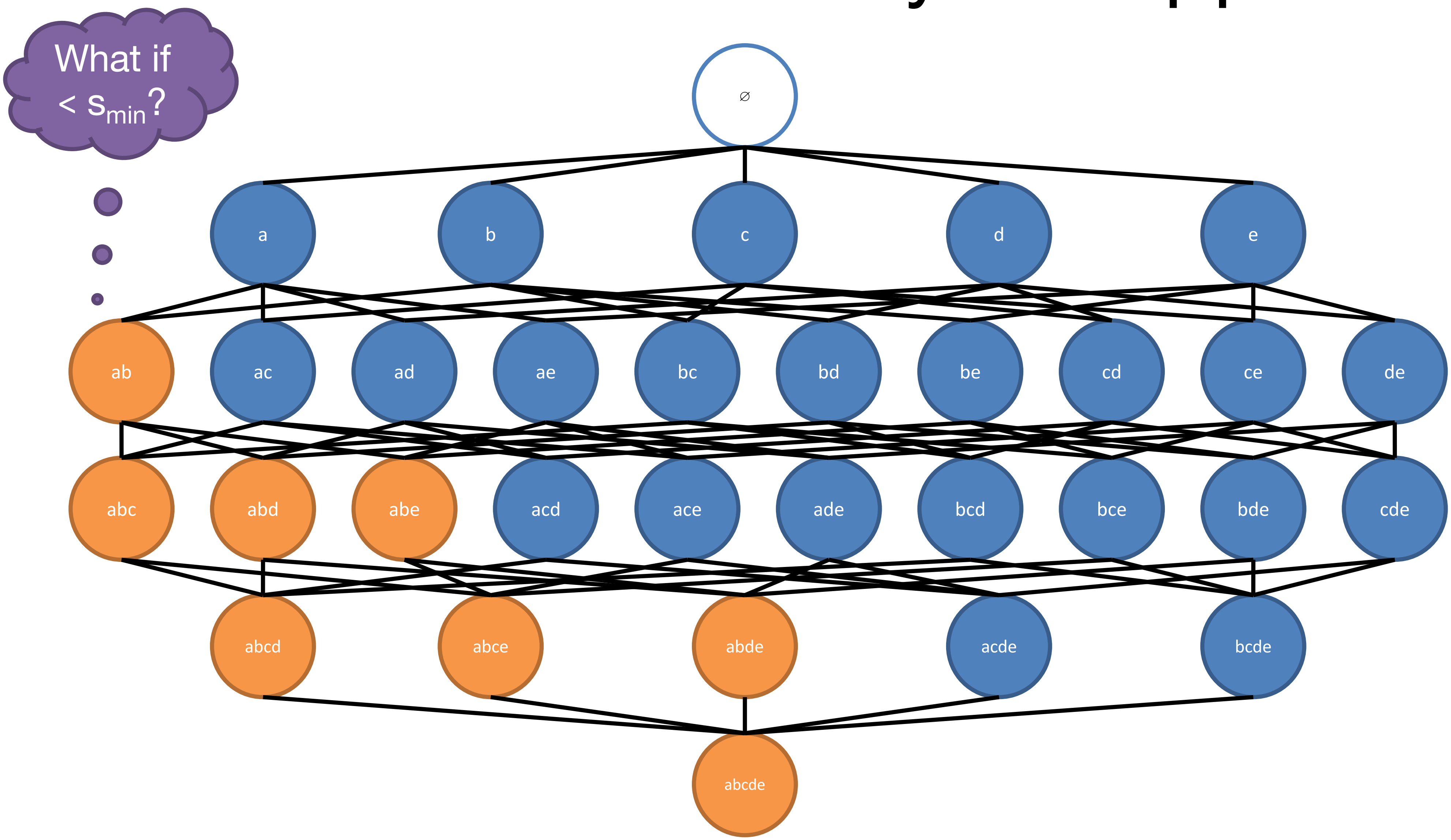
The Apriori Principle



And the Converse?



The Anti-Monotonicity of Support



The Apriori Algorithm Illustrated

Candidate 1-Itemsets

Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Minimum support count = 3

Candidate 2-Itemsets

Itemset	Count
{Beer, Bread}	2
{Beer, Diapers}	3
{Beer, Milk}	2
{Bread, Diapers}	3
{Bread, Milk}	3
{Diapers, Milk}	3

Itemsets removed because of low support

Candidate 3-Itemsets

Itemset	Count
{Bread, Diapers, Milk}	3

Figure 6.5. Illustration of frequent itemset generation using the *Apriori* algorithm.

The Apriori Algorithm

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

$k = 1.$
$F_k = \{\, i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}.$ {Find all frequent 1-itemsets}
repeat
  $k = k + 1.$
  $C_k = \text{apriori-gen}(F_{k-1}).$ {Generate candidate itemsets}
  for each transaction $t \in T$ **do**
    $C_t = \text{subset}(C_k, t).$ {Identify all candidates that belong to t}
    for each candidate itemset $c \in C_t$ **do**
      $\sigma(c) = \sigma(c) + 1.$ {Increment support count}
    end for
  end for
  $F_k = \{\, c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}.$ {Extract the frequent k-itemsets}
until $F_k = \emptyset$
Result $= \bigcup F_k.$

Apriori: Basic Flow

1. Count transaction occurrence for single items – keep frequent
2. Loop
   - *a)* ***Generate candidates***
   - b) For each candidate
     - *i.* ***Count transaction occurrence*** – keep frequent
   - c) Return if no new itemsets generated

Apriori: Generating Candidates

Goals

- Avoid unnecessary candidates
- Avoid duplicate candidates
- Do not miss any frequent itemsets (*complete*)

Approaches

- Brute Force
- $F_{k-1} \times F_1$
- $F_{k-1} \times F_{k-1}$

Brute Force

Items

Item
Beer
Bread
Cola
Diapers
Milk
Eggs

→

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Milk}
{Beer, Bread, Eggs}
{Beer, Cola, Diapers}
{Beer, Cola, Milk}
{Beer, Cola, Eggs}
{Beer, Diapers, Milk}
{Beer, Diapers, Eggs}
{Beer, Milk, Eggs}
{Bread, Cola, Diapers}
{Bread, Cola, Milk}
{Bread, Cola, Eggs}
{Bread, Diapers, Milk}
{Bread, Diapers, Eggs}
{Bread, Milk, Eggs}
{Cola, Diapers, Milk}
{Cola, Diapers, Eggs}
{Cola, Milk, Eggs}
{Diapers, Milk, Eggs}

→

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

Figure 6.6. A brute-force method for generating candidate 3-itemsets.

Brute Force Analysis

- Generates C(N, k) candidates
- Some cost in generation, but more so validating of frequency of occurrence

F_{k-1} x F_1

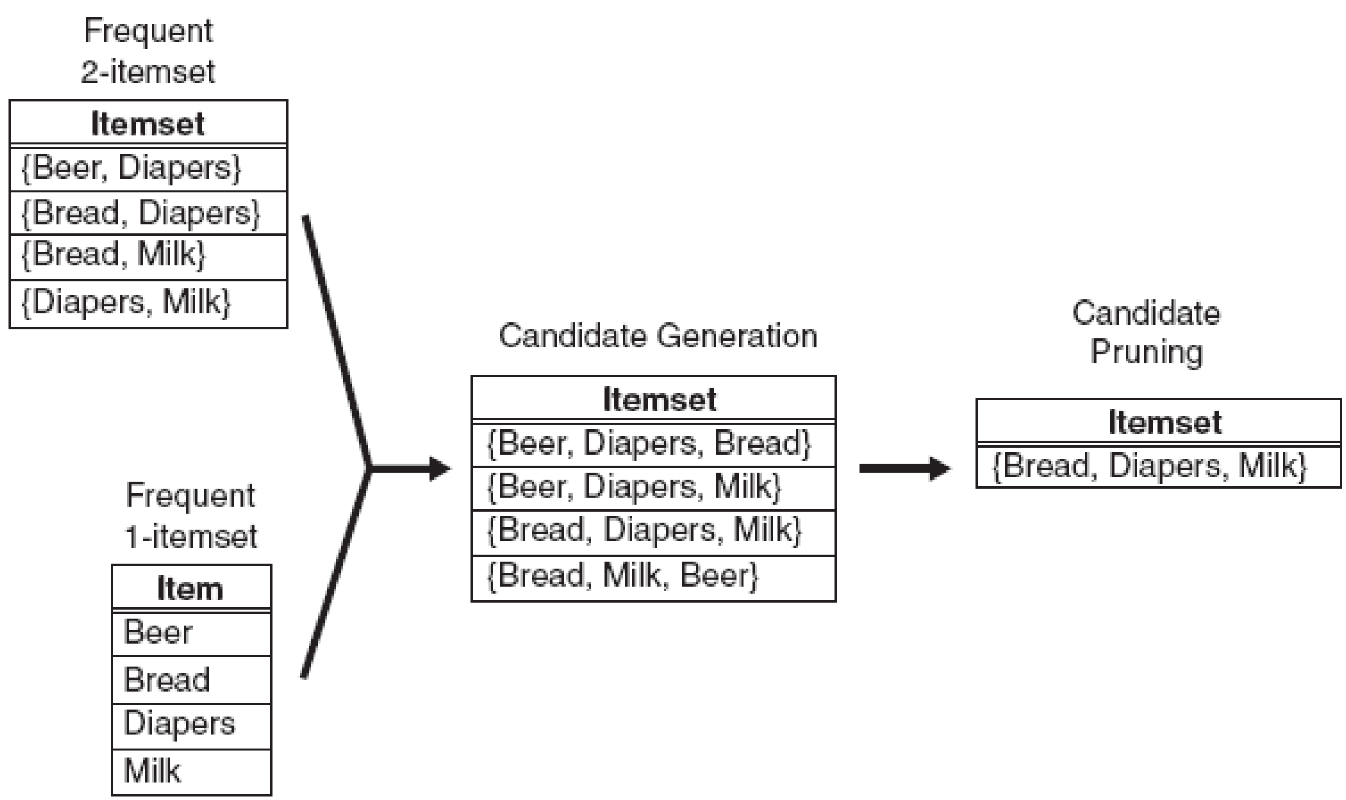


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

F_{k-1} x F_1 Analysis

- Improvement: still potential for large number of unnecessary candidates
- Can produce duplicates
  - Fixed via sorted order of candidate items

$F_{k-1} \times F_{k-1}$

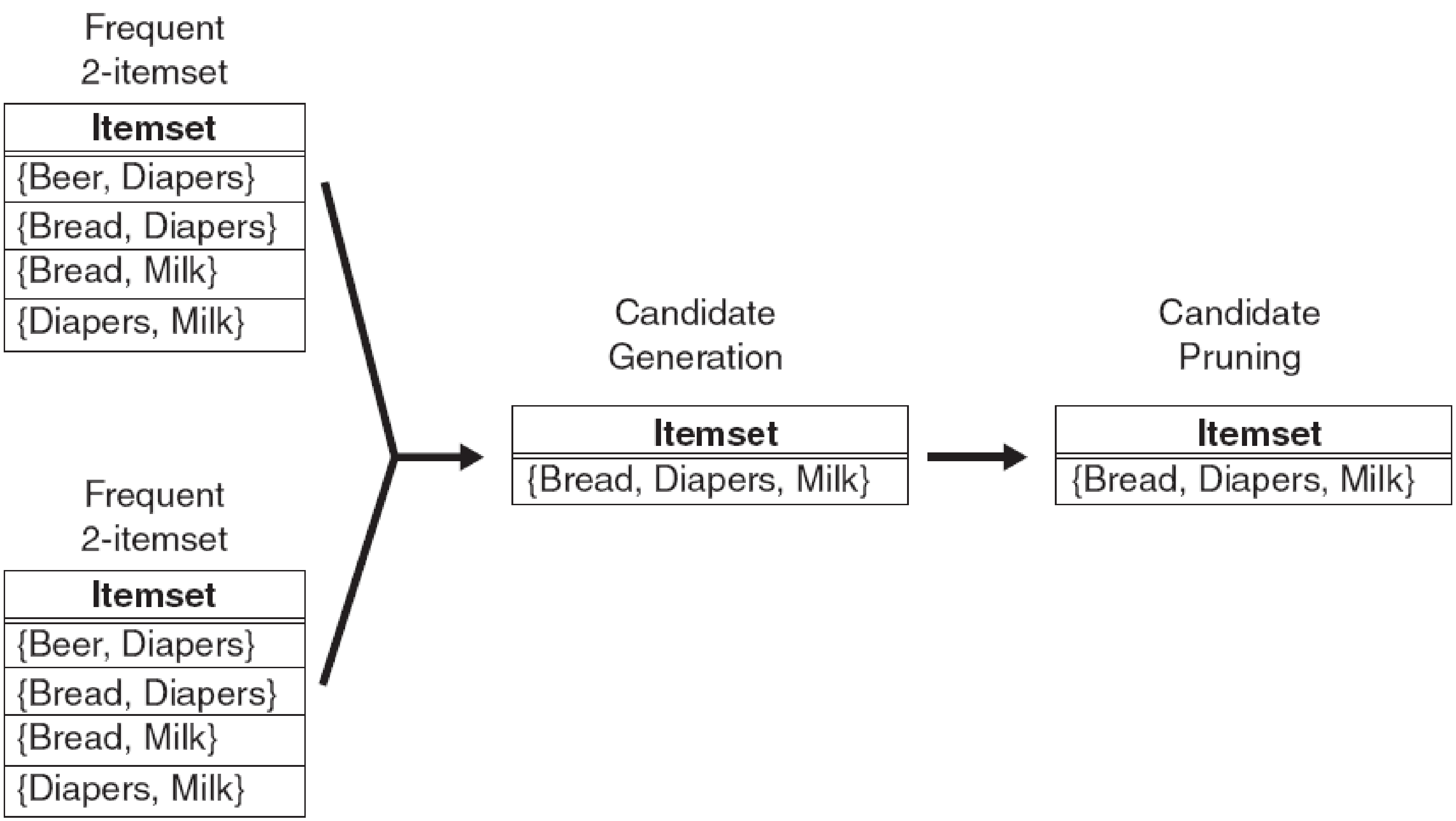


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

F_{k-1} x F_{k-1} Analysis

- Basic: combine if not same & differ by 1
- Improvement: sort items, combine if first (k-2) elements same (but not k-1!)
  - Still need some validation

Apriori: Basic Flow

1. Count transaction occurrence for single items – keep frequent
2. Loop
   a) ***Generate candidates***
   b) For each candidate
      i. ***Count transaction occurrence*** – keep frequent
   c) Return if no new itemsets generated

Support Counting

- Brute Force: transactions x candidates
- Transactions -> k-sets
  - With sorting, can update efficiently
- Hash table (see TSK)
- **Trie/Prefix Tree**

Apriori Analysis – ALL the DB Scans

1. Count transaction occurrence for single items – keep frequent
2. Loop
   a) ***Generate candidates***
   b) For each candidate
      i. ***Count transaction occurrence*** – keep frequent
   c) Return if no new itemsets generated

Frequent Pattern Growth Algorithm

- Allows discovery of frequent itemsets
  - Without candidate generation
  - Only 2 passes over transaction dataset

- FP-Growth algorithm sketch…
  1. 2-pass over transactions -> builds FP-Tree
  2. Multiple passes over FP-Tree -> frequent itemsets

FP-Growth.1: Building an FP-Tree

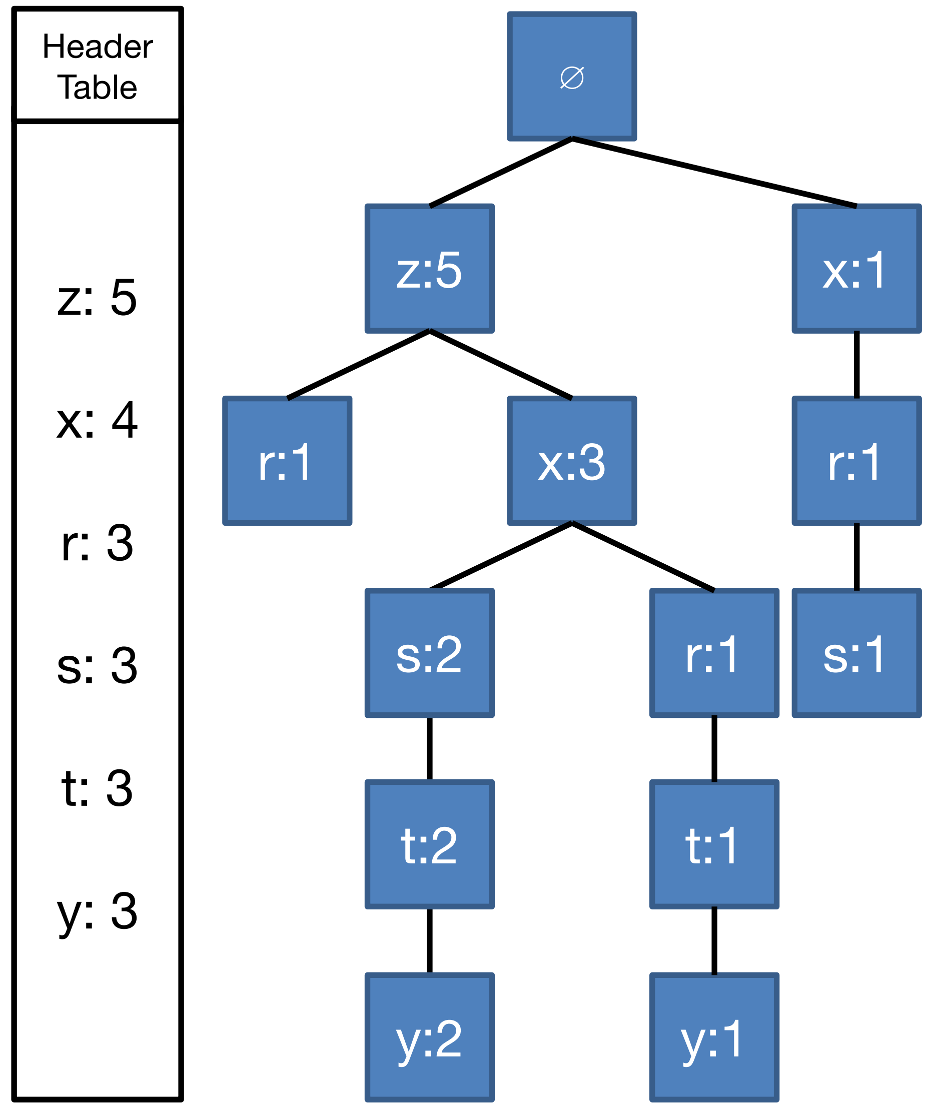
- Stores frequency of occurrence for sets of items in the transaction dataset
  - Set: path in tree

- Common prefixes will share paths
  - Hence: compression!

Example FP-Tree

Transactions*

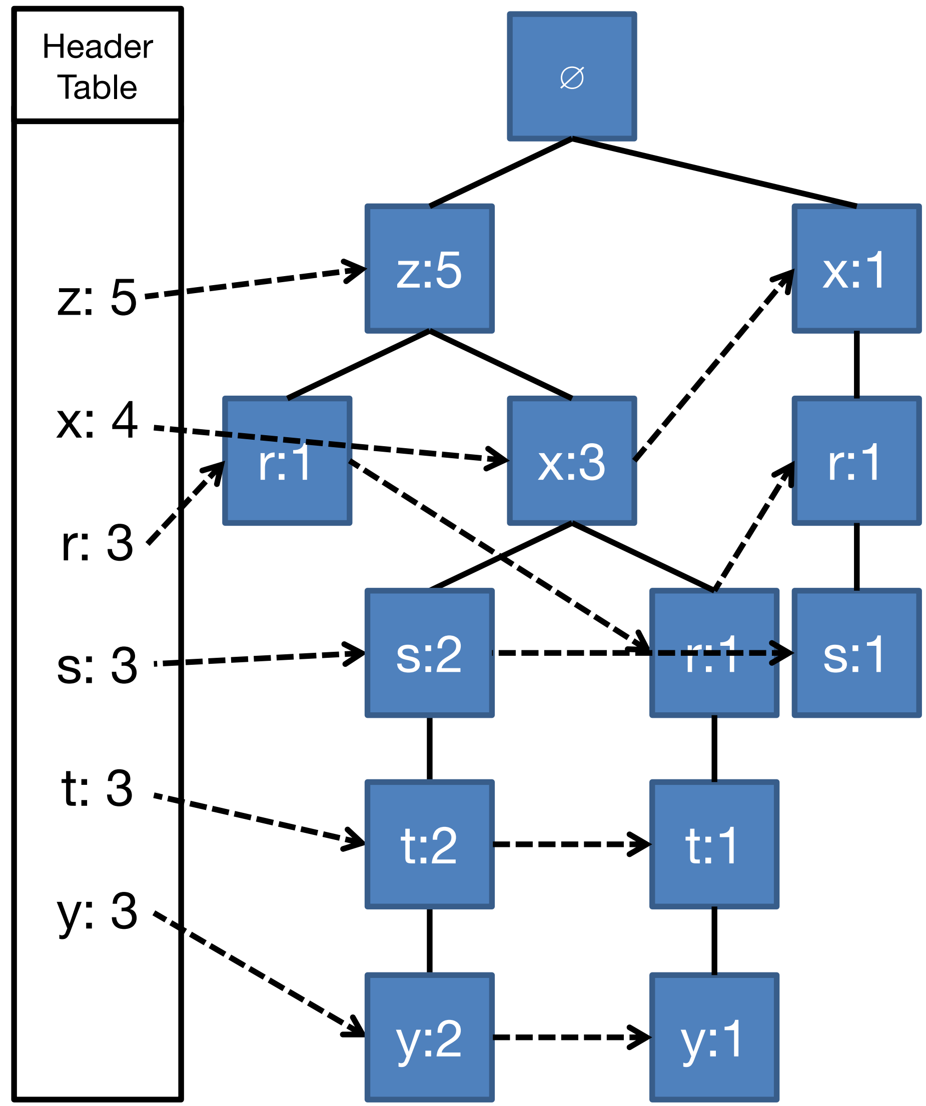
- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y
- z, x, s, t, y



Example FP-Tree (with Item Pointers)

Transactions*

- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y
- z, x, s, t, y



Building an FP-Tree: Pass 1

1. for each transaction t
   a) for each item i in t
      i. support[i] += 1

Count item support

2. for each item i in keys(support)
   a. if support[i] < s_{min}
      i. delete(support[i])

Remove items below threshold

3. sort(support, *by values decreasing*)

4. for each transaction t
   a. delete items not in support
   b. sort(t, *by keys(support)*)

Sort transactions via increasing support (removing items below threshold)

Example FP-Tree: Pass 1

Original Transactions

- r, z, h, j, p
- z, y, x, w, v, u, t, s
- z
- r, x, n, o, s
- y, r, x, z, q, t, p
- y, z, x, e, q, s, t, m

Filtered & Sorted

- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y
- z, x, s, t, y

Item Counts (s_{min}=3)

r:3, z:5, ~~h: 1~~, ~~j:1~~, ~~p:2~~, y:3, x:4, ~~w:1~~, ~~v:1~~, ~~u:1~~, t:3, s:3, ~~n:1~~, ~~o:1~~, ~~q:2~~, ~~e:1~~, ~~m:1~~

- z:5, x:4, r:3, s:3, t:3, y:3

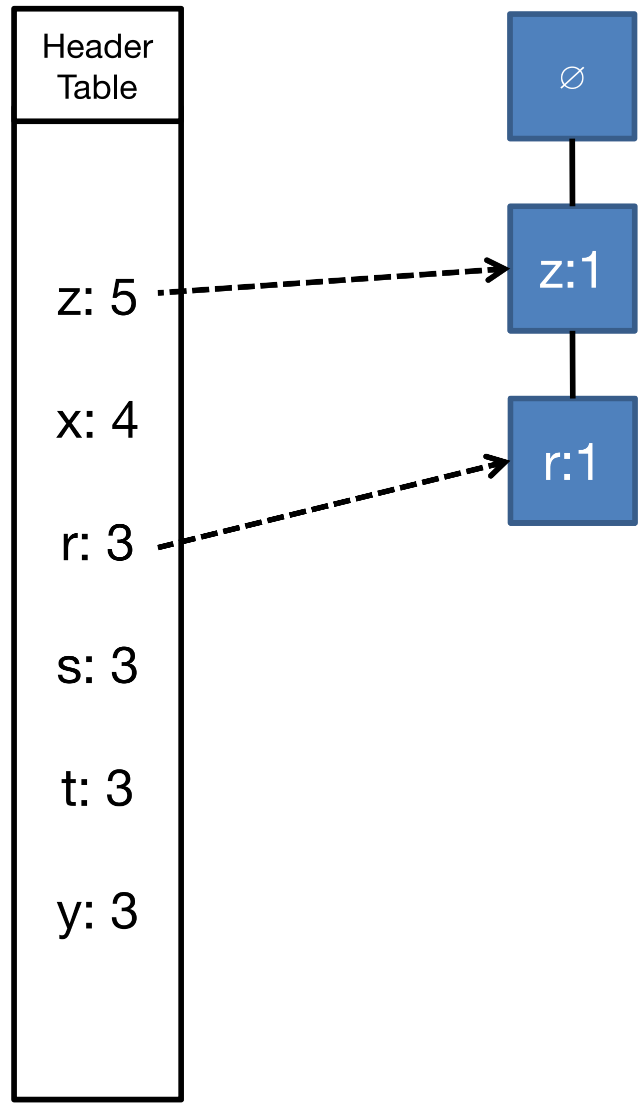
Building an FP-Tree: Pass 2

1. for each filtered+sorted transaction t
   a. add path to tree
      i. update counts at each existing node
      ii. update pointers for each node

Example FP-Tree, Pass 2: T1

Transactions*

- z, r

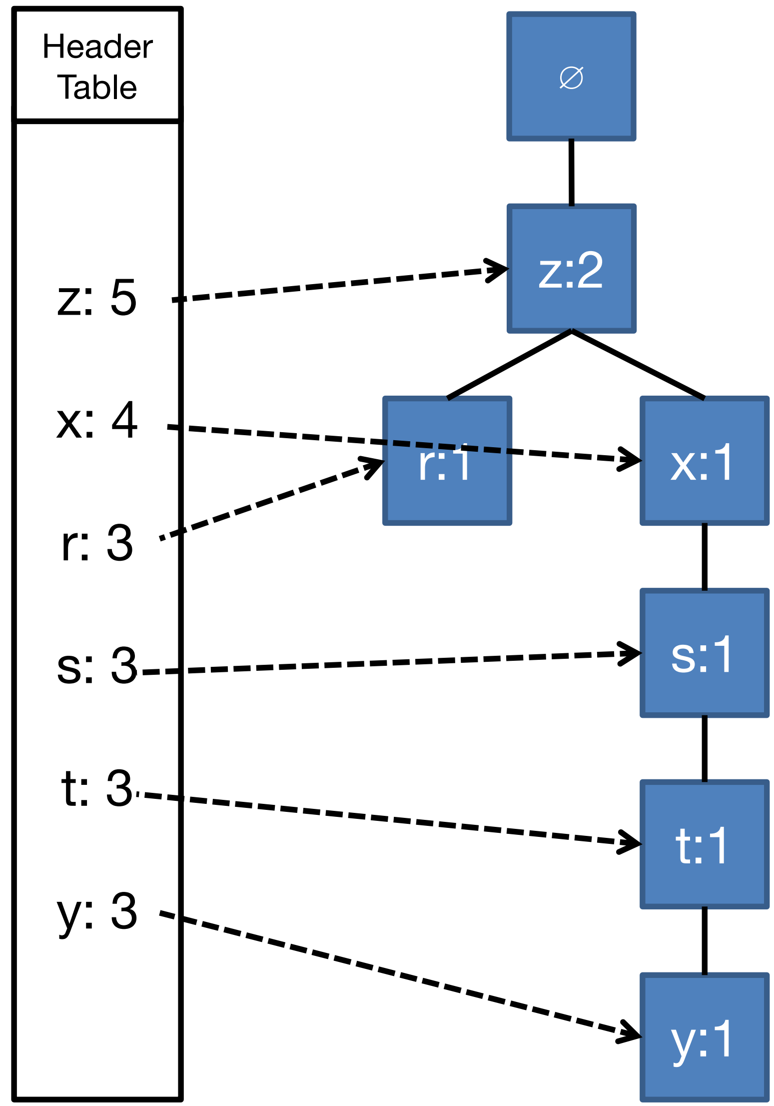


Next: z, x, s, t, y

Example FP-Tree, Pass 2: T2

Transactions*

- z, r
- z, x, s, t, y

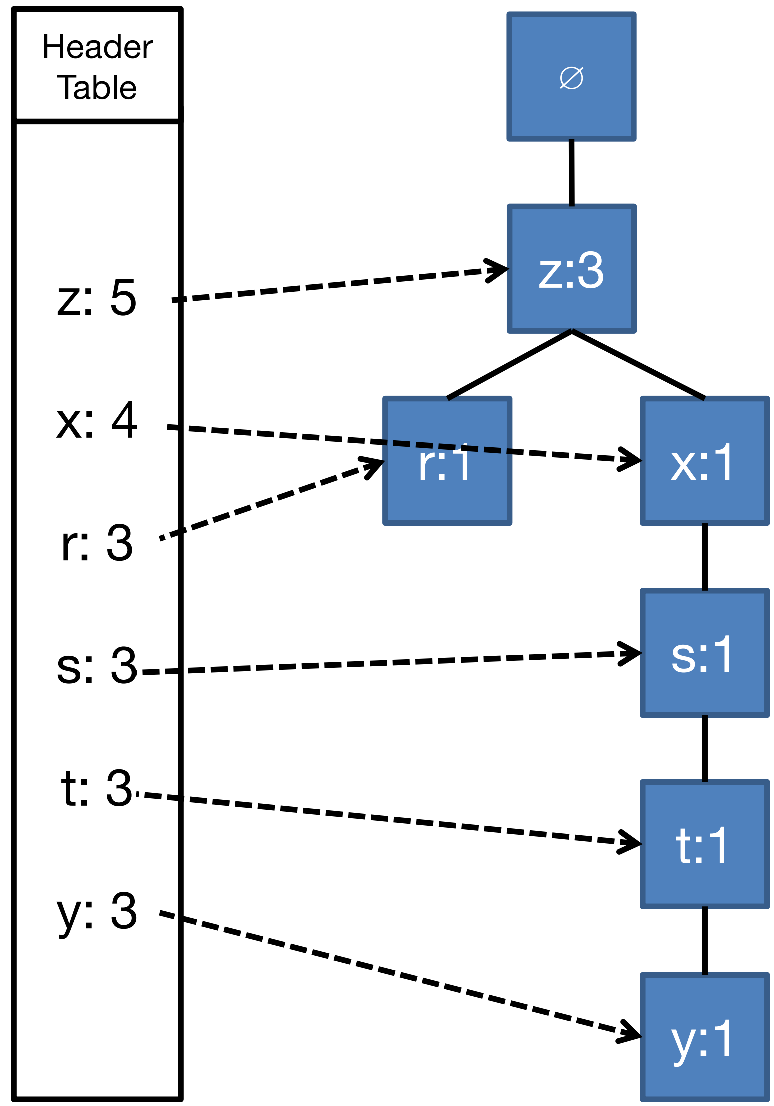


Next: z

Example FP-Tree, Pass 2: T3

Transactions*

- z, r
- z, x, s, t, y
- z

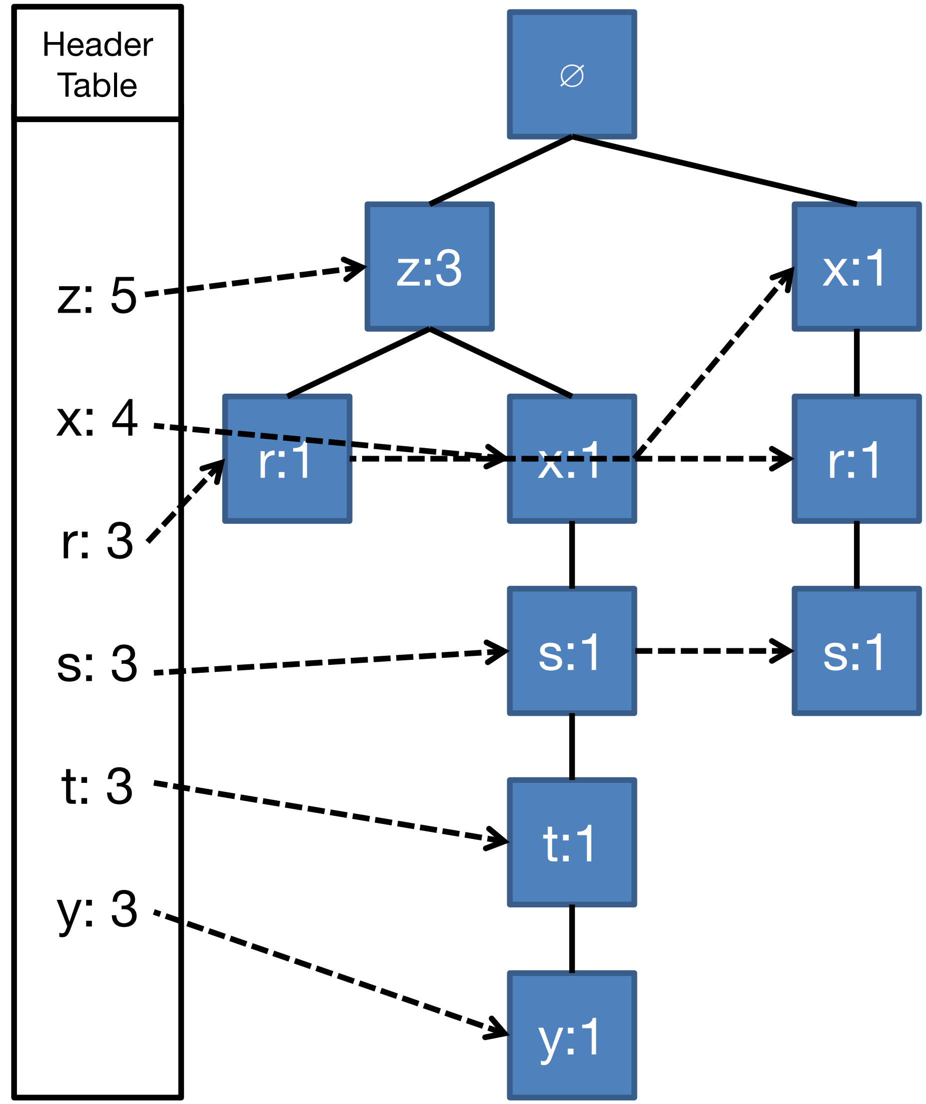


Next: x, r, s

Example FP-Tree, Pass 2: T4

Transactions*

- z, r
- z, x, s, t, y
- z
- x, r, s

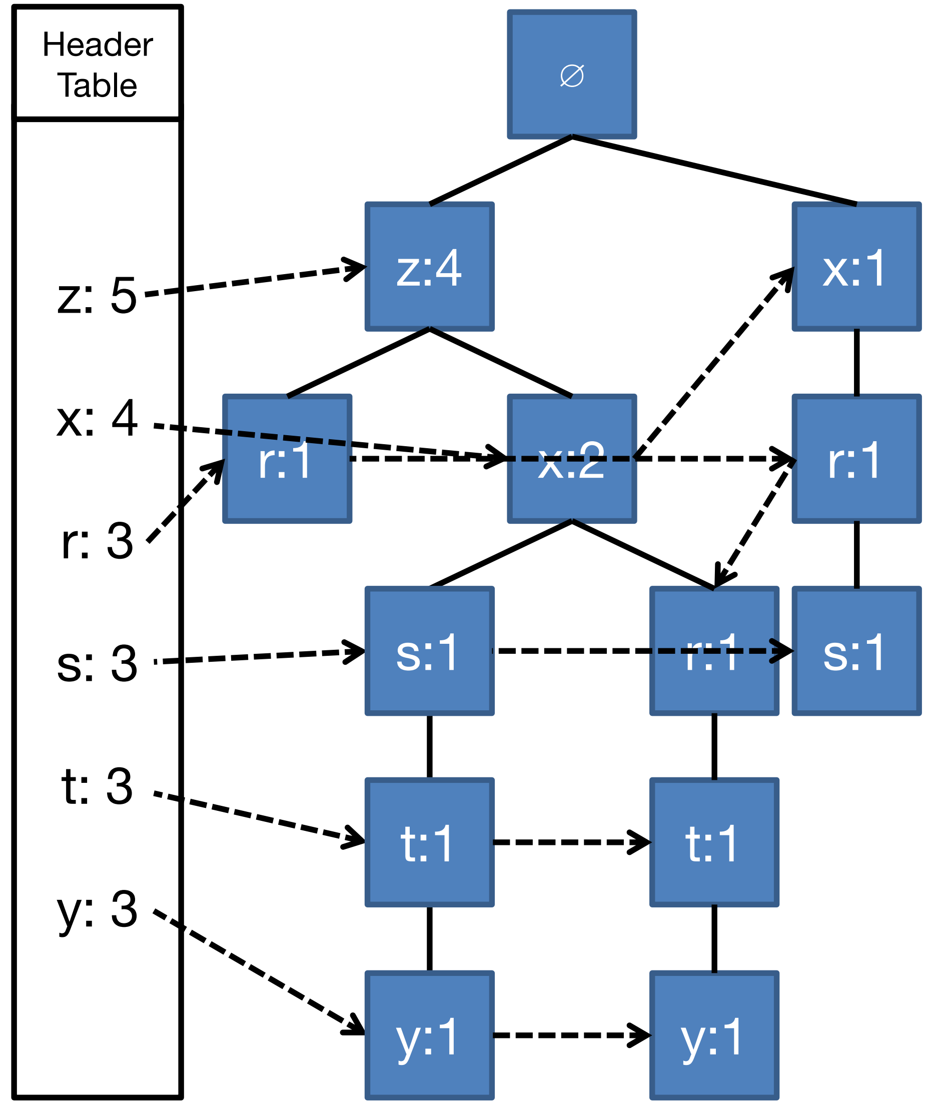


Next: z, x, r, t, y

Example FP-Tree, Pass 2: T5

Transactions*

- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y

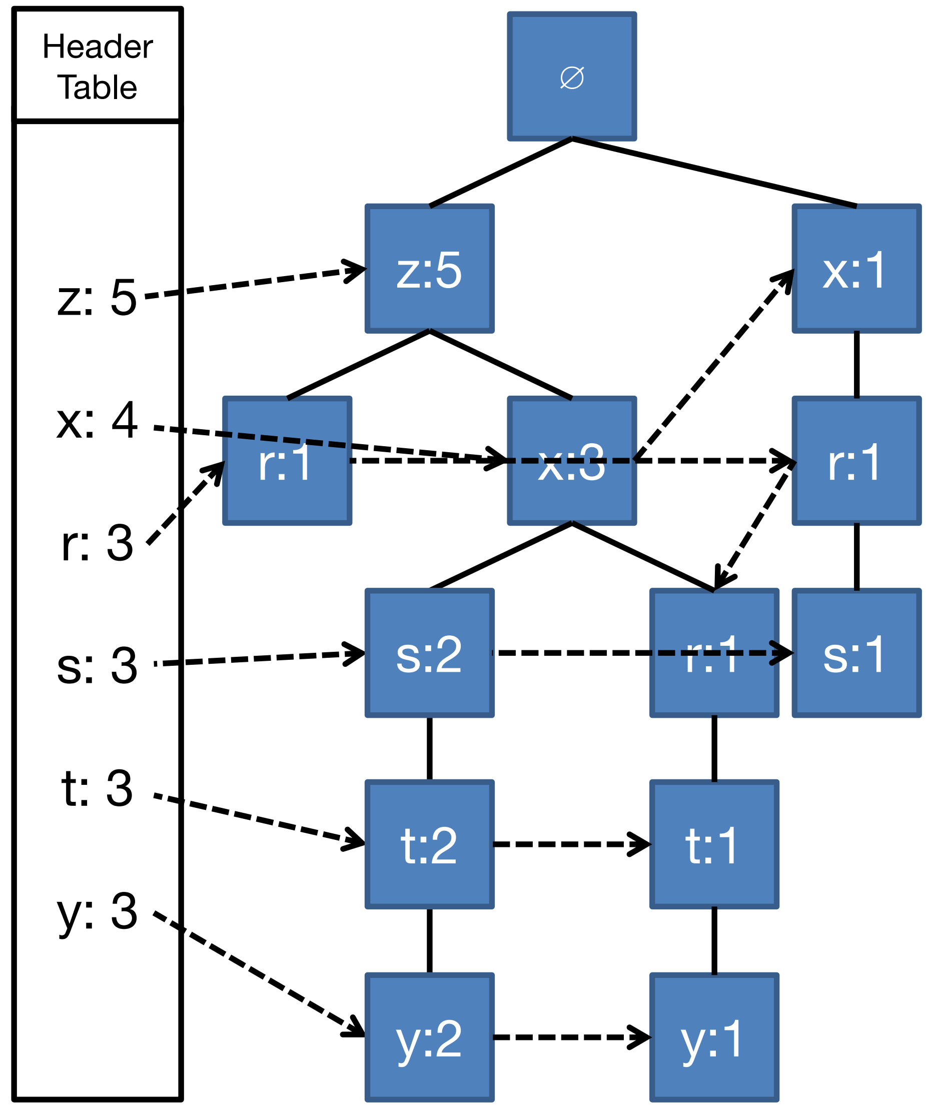


Next: z, x, s, t, y

Example FP-Tree, Pass 2: T6

Transactions*

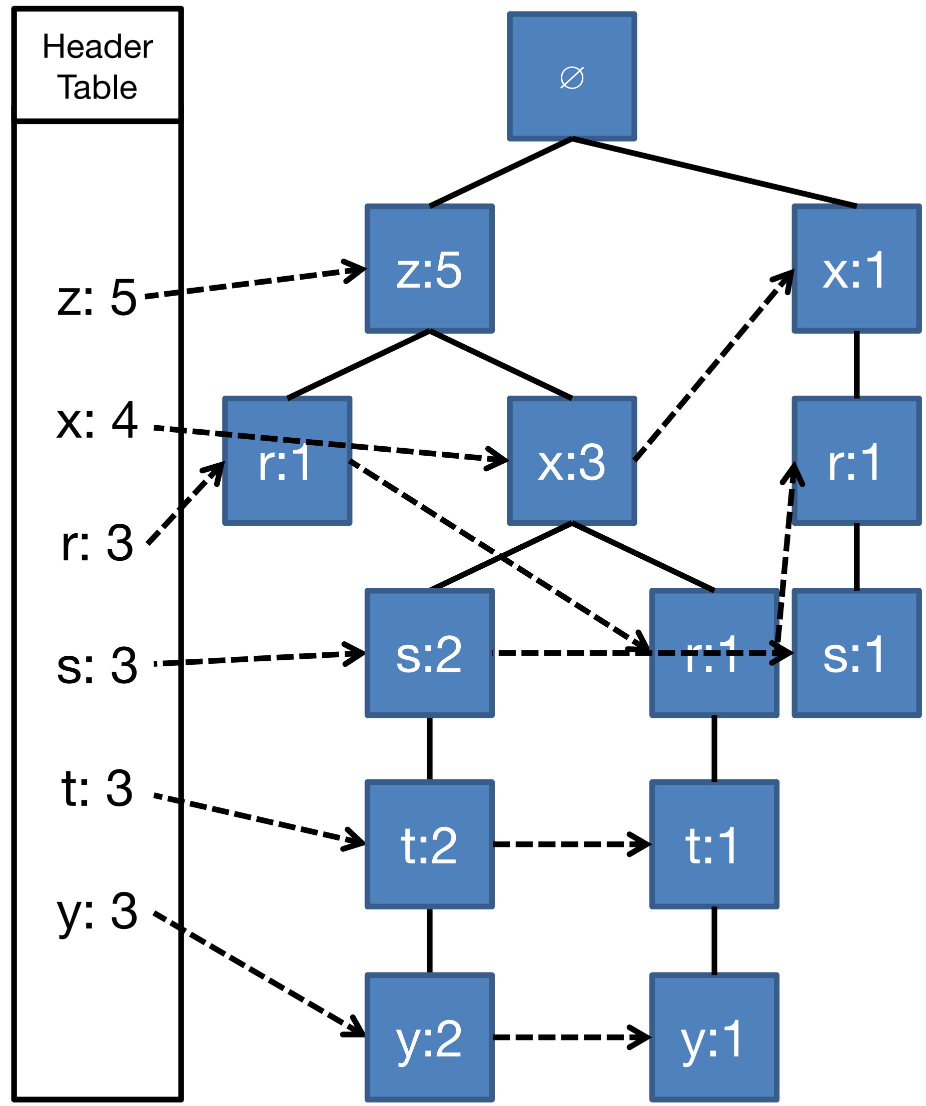
- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y
- z, x, s, t, y



Prettier (for next steps)

Transactions*

- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y
- z, x, s, t, y



Now You Try! (s_{min}=3)

TID	Items Bought
1	a, b, f
2	b, g, c, d
3	h, a, c, d, e
4	a, d, p, e
5	a, b, c
6	a, b, q, c, d
7	a
8	a, m, b, c
9	a, b, n, d
10	b, c, e, m

a: 8
b: 7
~~f: 1~~
~~g: 1~~
c: 6
d: 5
~~h: 1~~
e: 3
~~p: 1~~
~~q: 1~~
~~m: 2~~
~~n: 1~~

Now You Try! (s_{min}=3)

TID	Items Bought
1	a, b, f
2	b, g, c, d
3	h, a, c, d, e
4	a, d, p, e
5	a, b, c
6	a, b, q, c, d
7	a
8	a, m, b, c
9	a, b, n, d
10	b, c, e, m

a:8, b:7, c:6, d:5, e:3

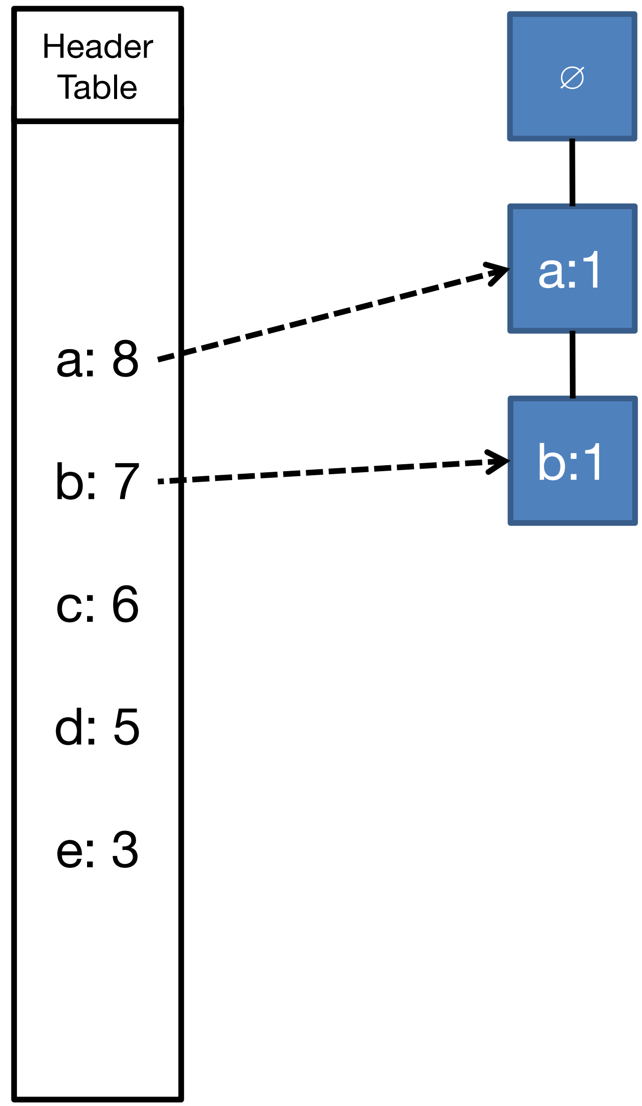
Filtered + Sorted

TID	Items Bought
1	a, b
2	b, c, d
3	a, c, d, e
4	a, d, e
5	a, b, c
6	a, b, c, d
7	a
8	a, b, c
9	a, b, d
10	b, c, e

Now You Try: T1

Transactions*

- a, b

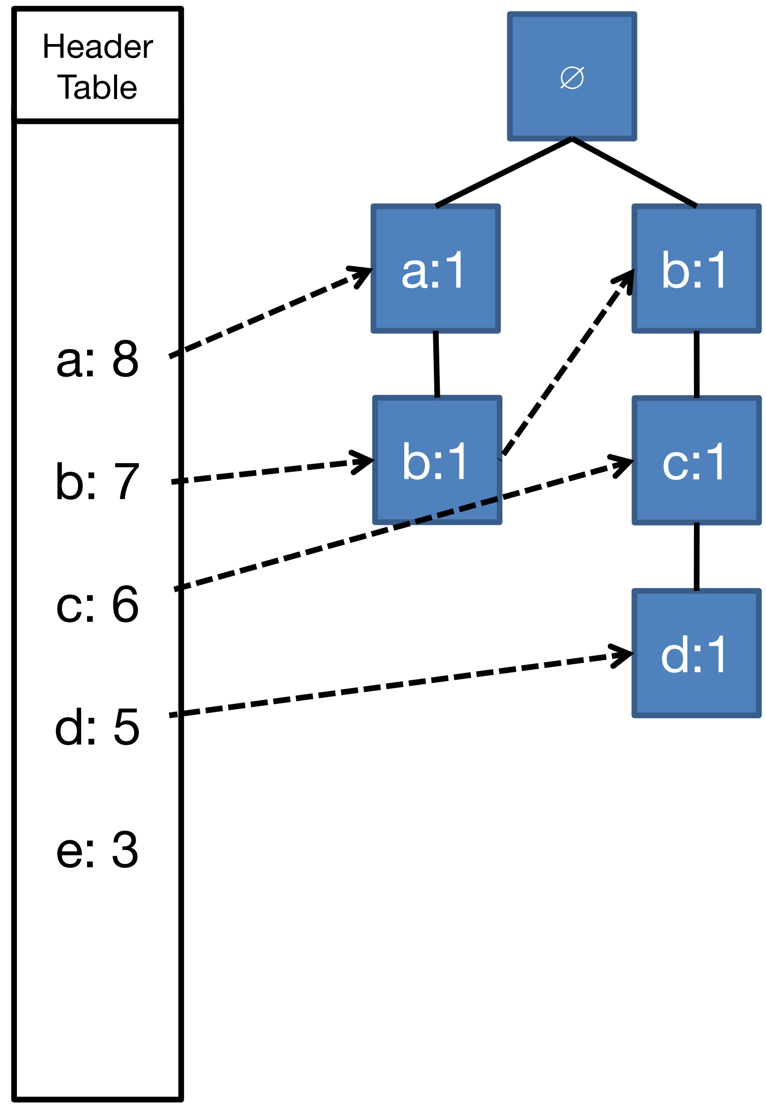


Next: b, c, d

Now You Try: T2

Transactions*

- a, b
- b, c, d

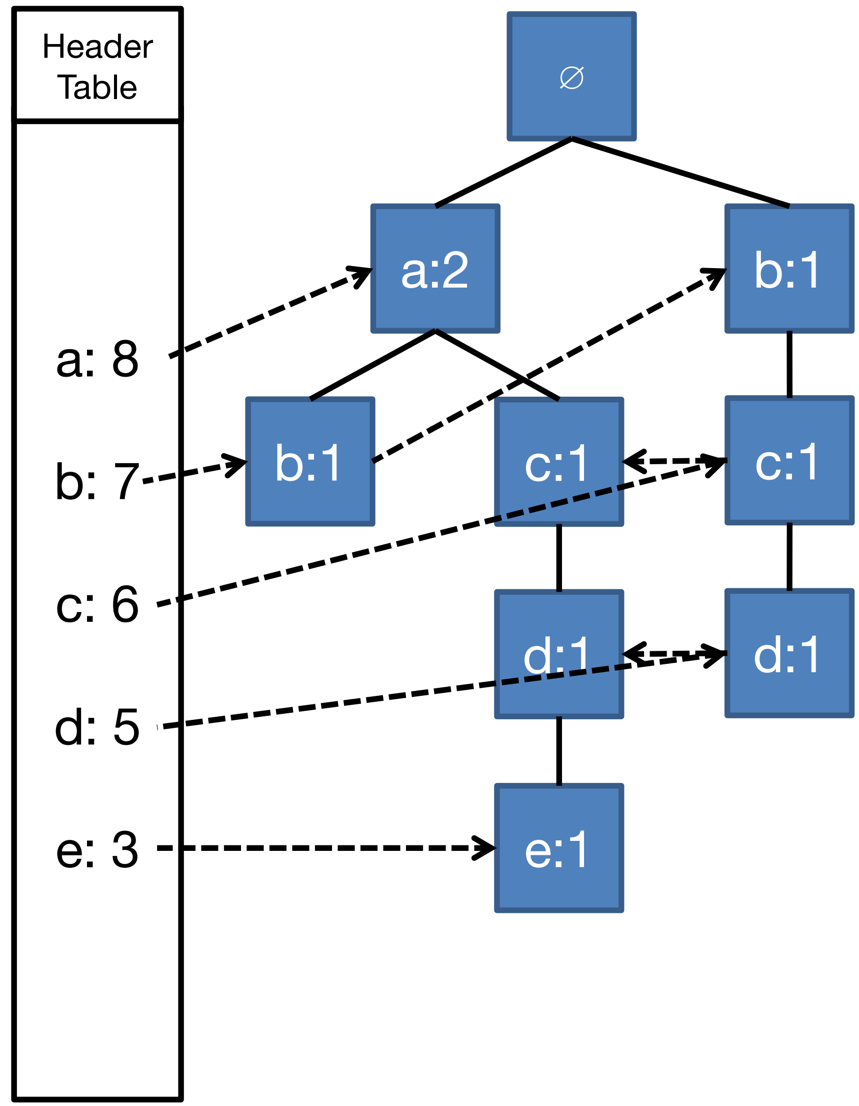


Next: a, c, d, e

Now You Try: T3

Transactions*

- a, b
- b, c, d
- a, c, d, e

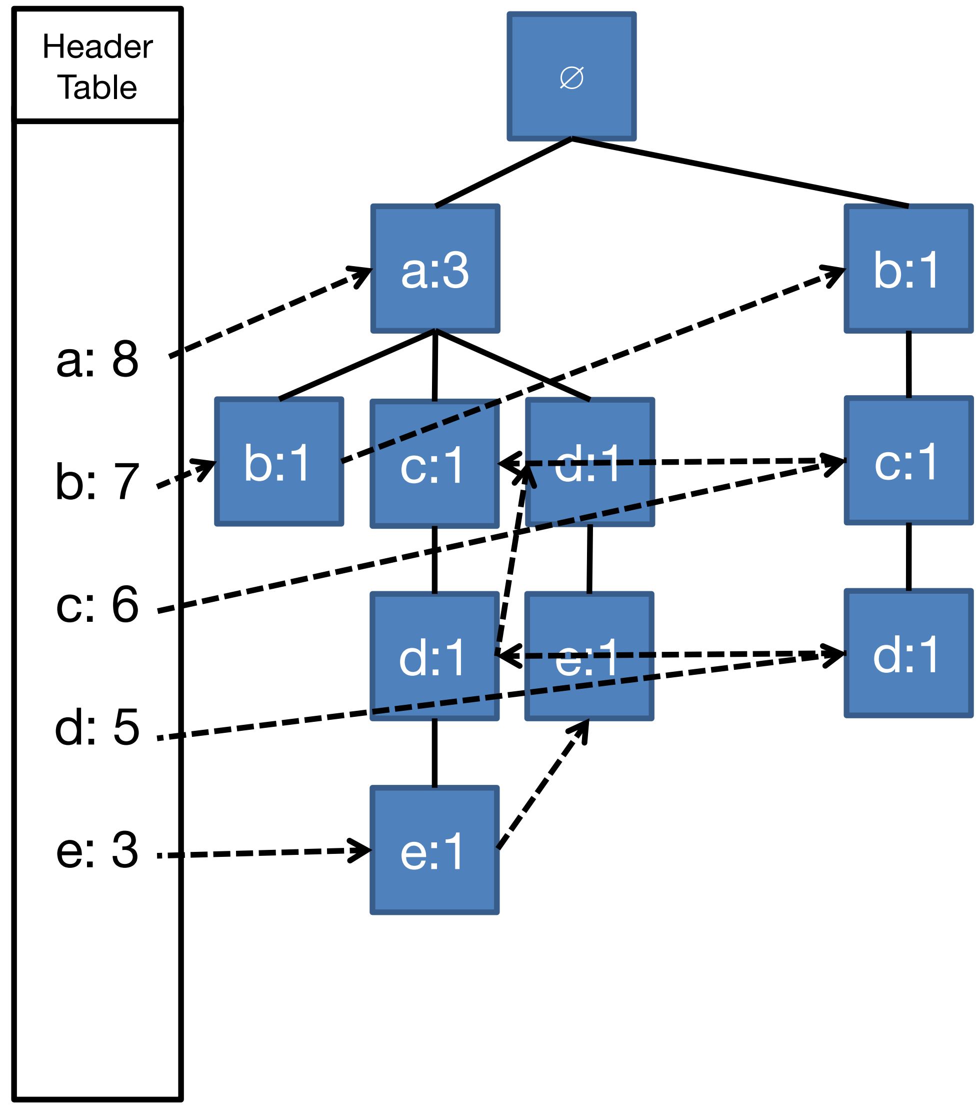


Next: a, d, e

Now You Try: T4

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e

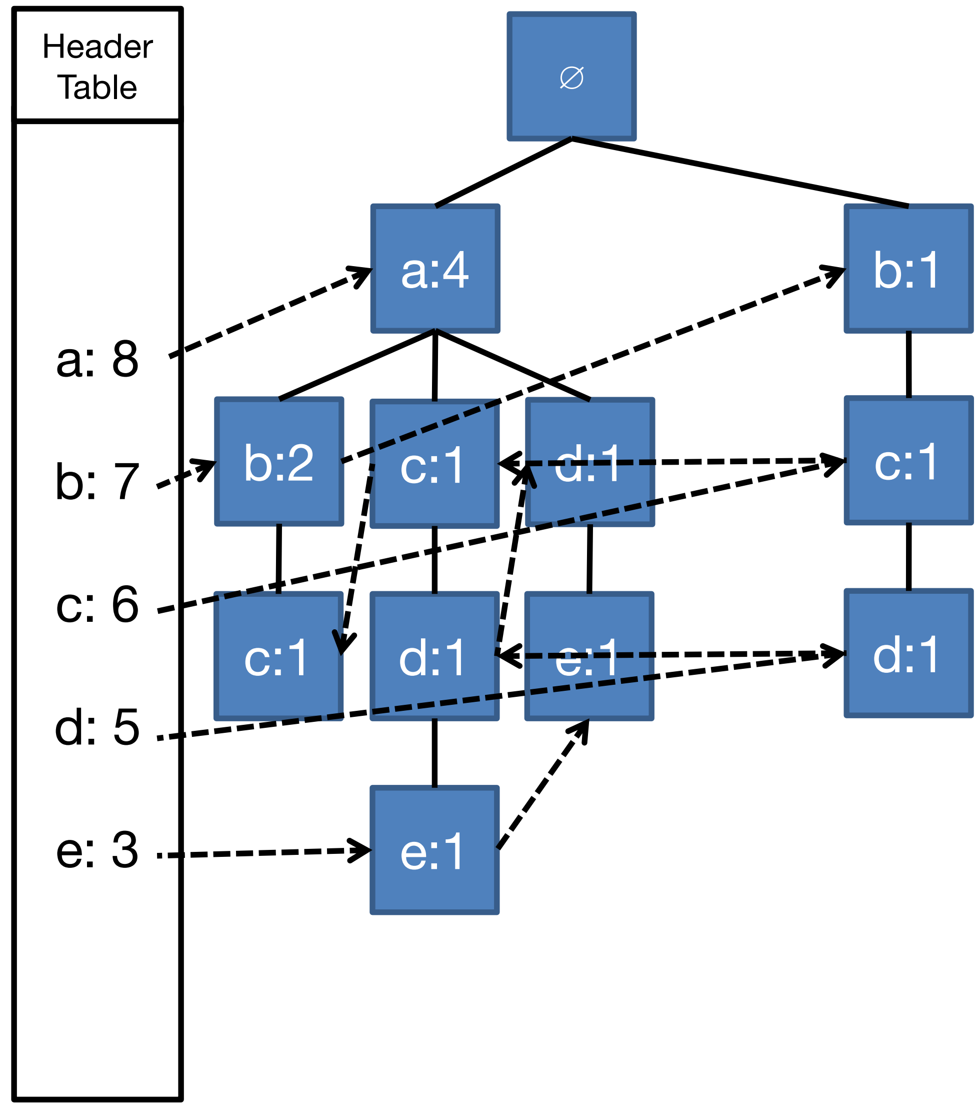


Next: a, b, c

Now You Try: T5

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c

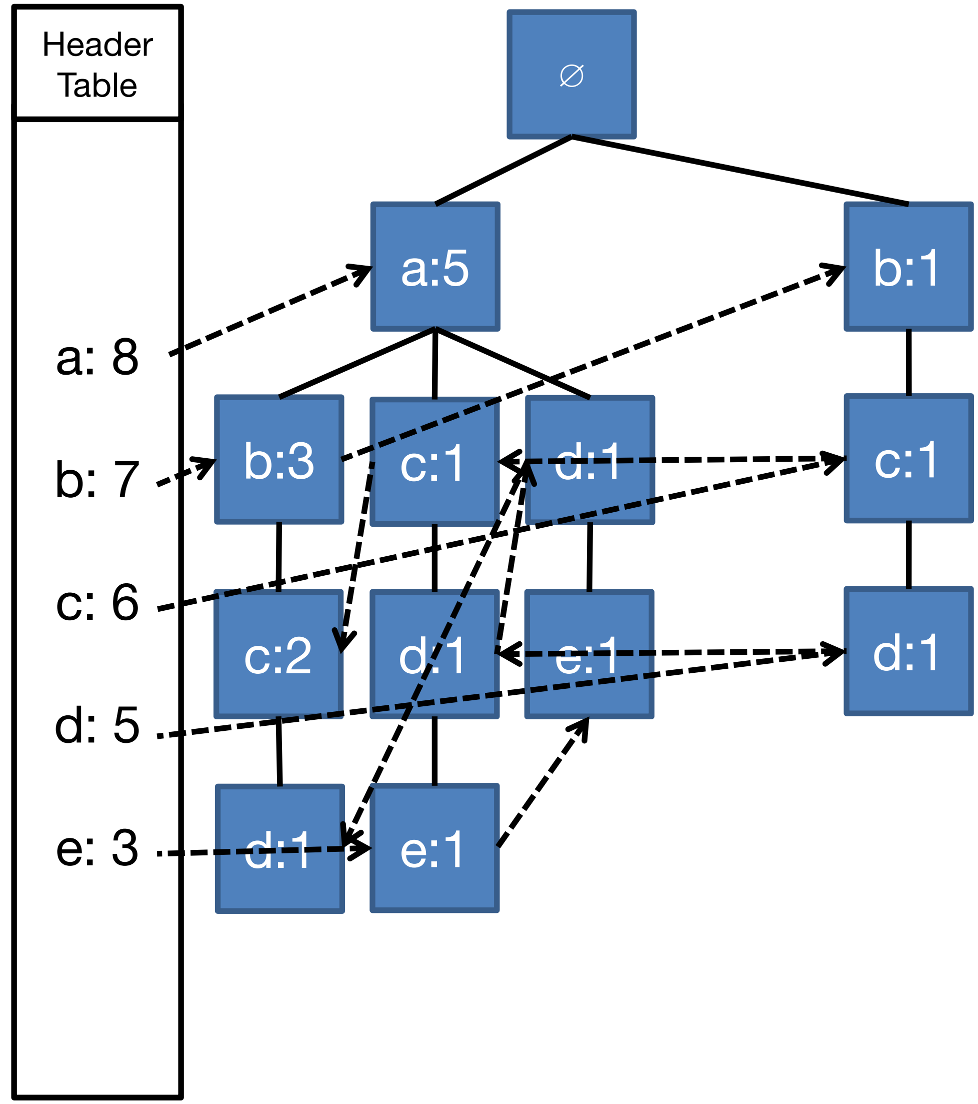


Next: a, b, c, d

Now You Try: T6

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c
- a, b, c, d

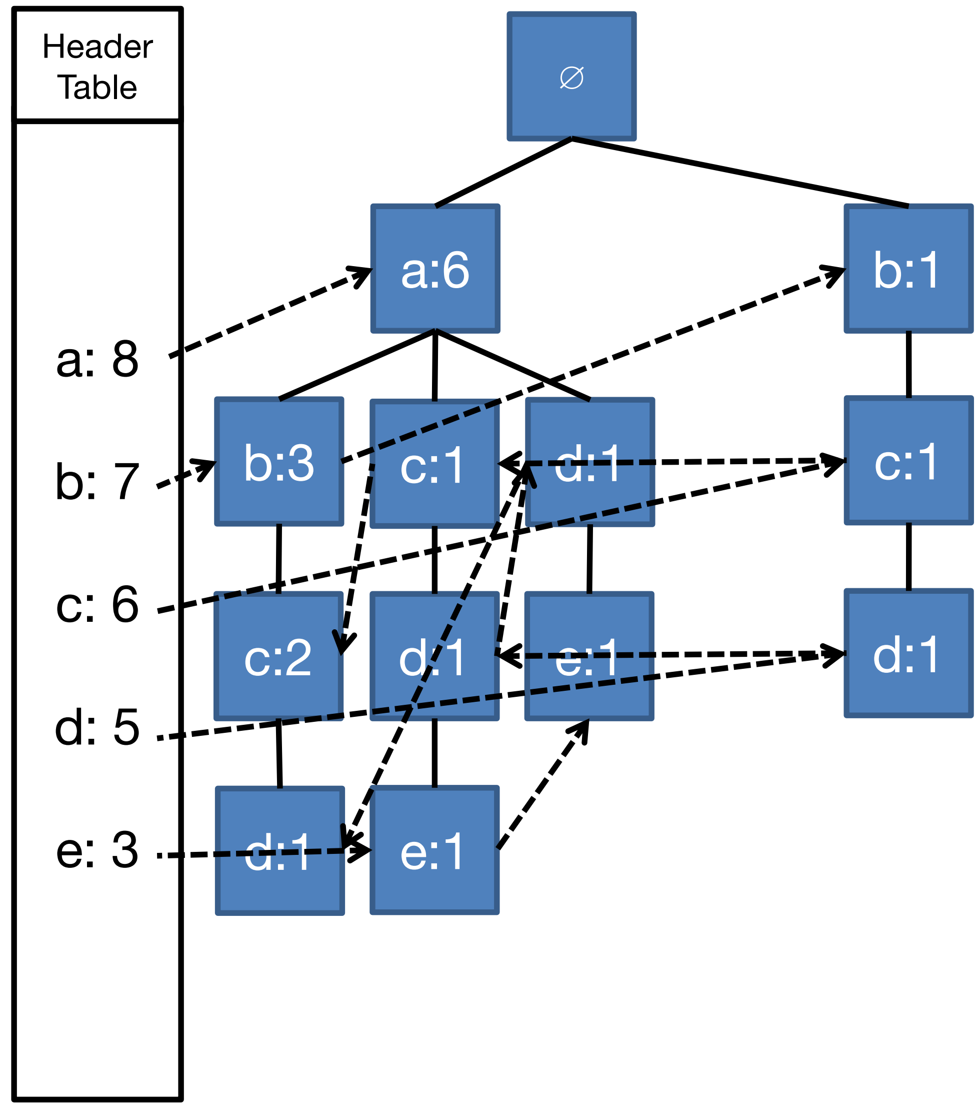


Next: a

Now You Try: T7

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c
- a, b, c, d
- a



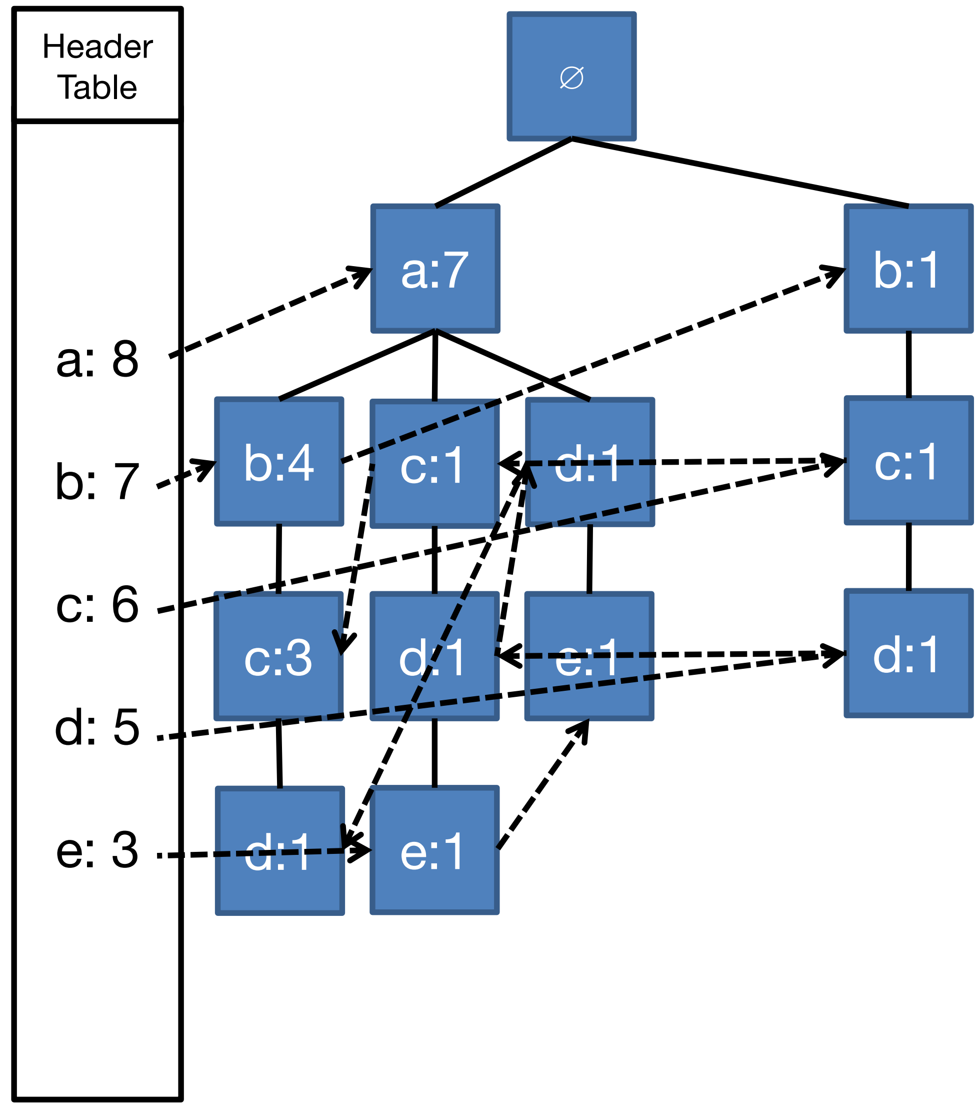
Next: a, b, c

Now You Try: T8

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c
- a, b, c, d
- a
- a, b, c

Next: a, b, d

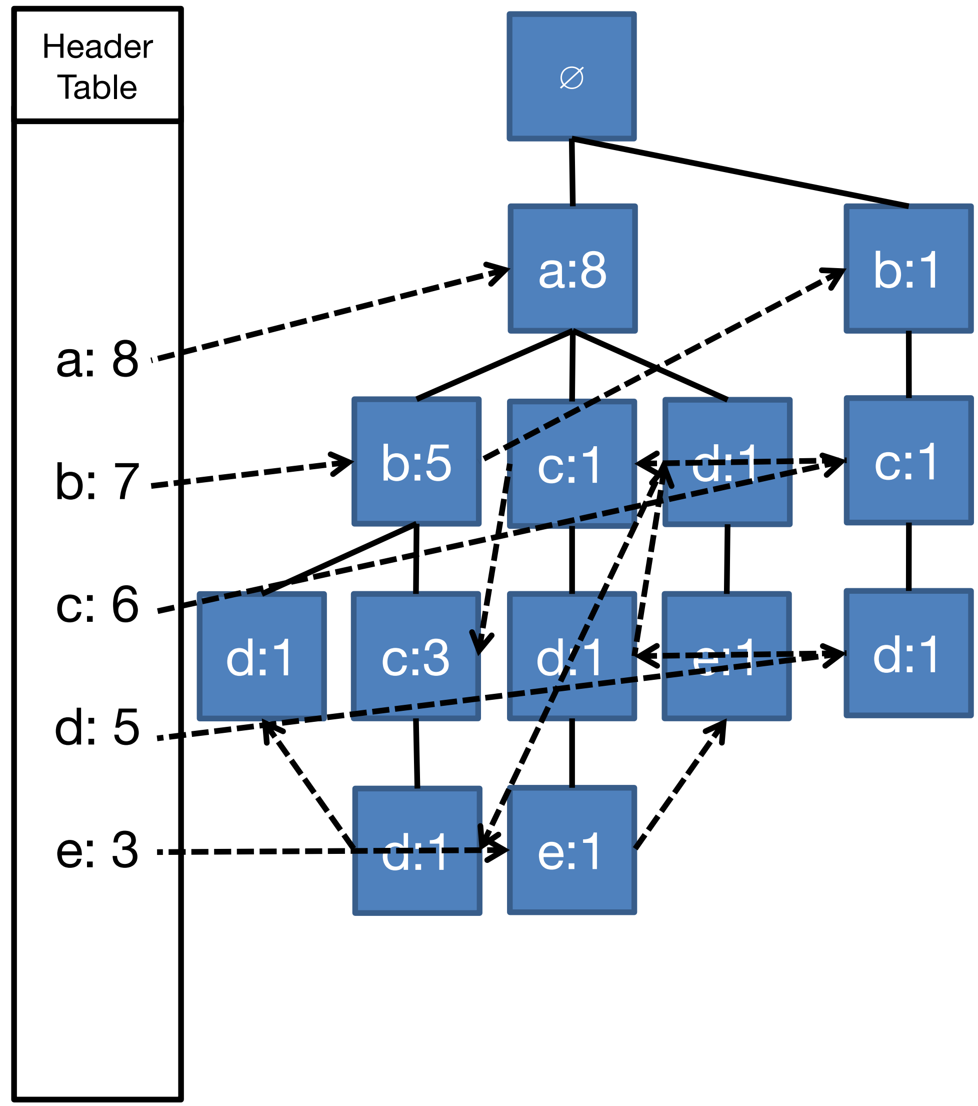


Now You Try: T9

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c
- a, b, c, d
- a
- a, b, c
- a, b, d

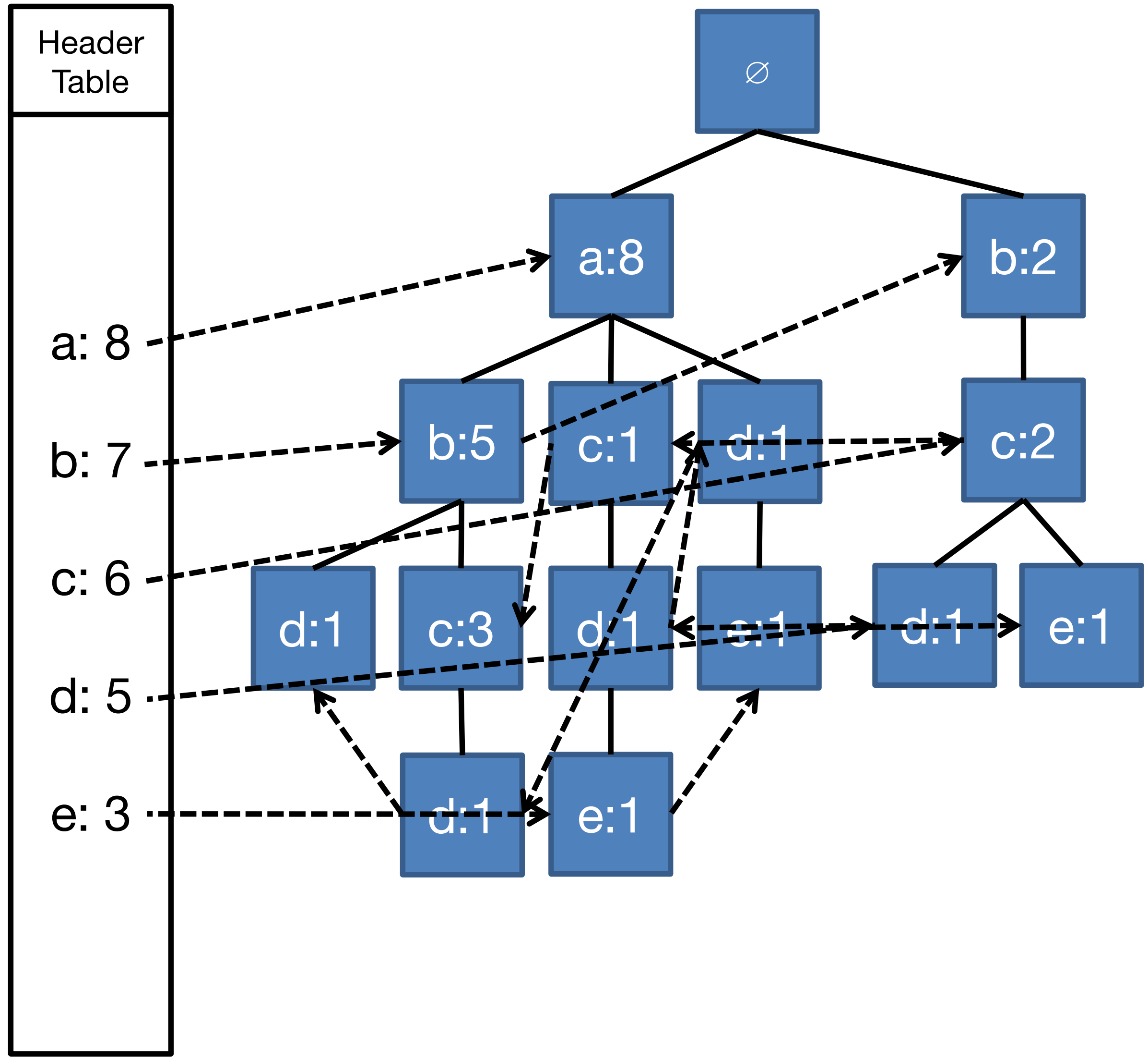
Next: b, c, e



Now You Try: T10

Transactions*

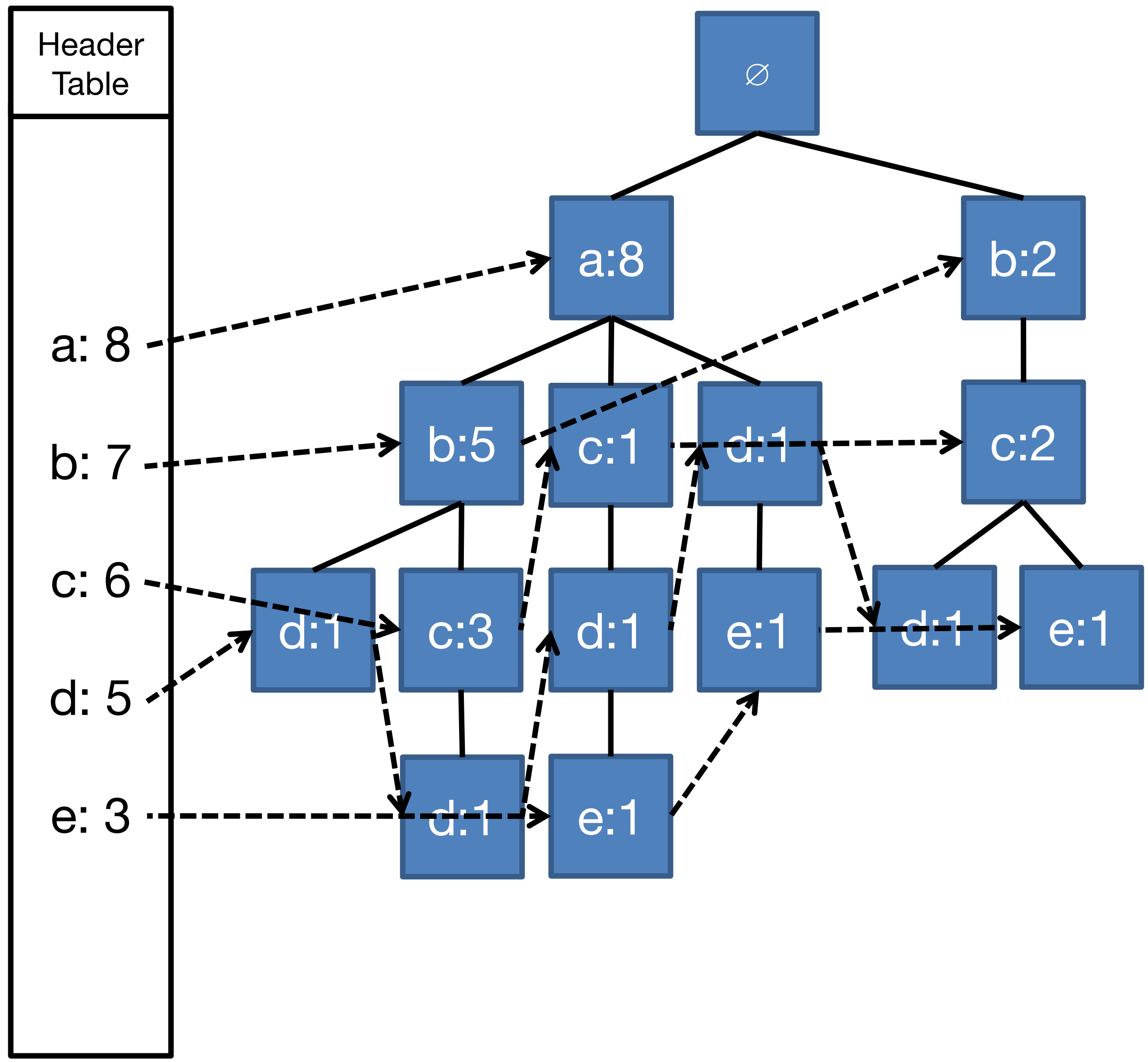
- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c
- a, b, c, d
- a
- a, b, c
- a, b, d
- b, c, e



Prettier (for next steps)

Transactions*

- a, b
- b, c, d
- a, c, d, e
- a, d, e
- a, b, c
- a, b, c, d
- a
- a, b, c
- a, b, d
- b, c, e

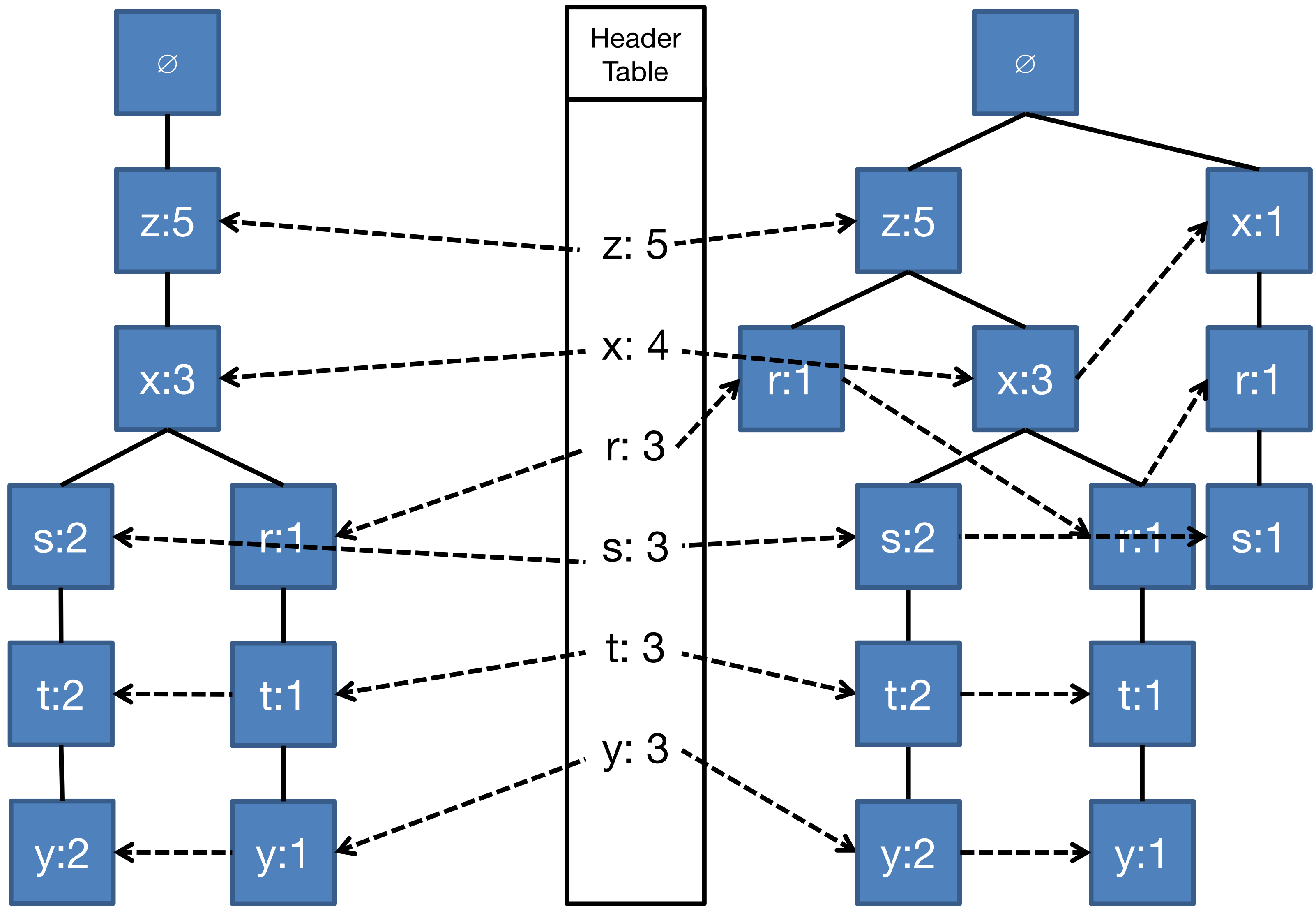


Mining Patterns from an FP-Tree

- You now have a more compact data structure that has all the information necessary to extract frequent itemsets
  - Divide and conquer! Look for item, then subsets (i.e. subtrees), recurse!
  - Similar to Apriori, but on the FP-Tree

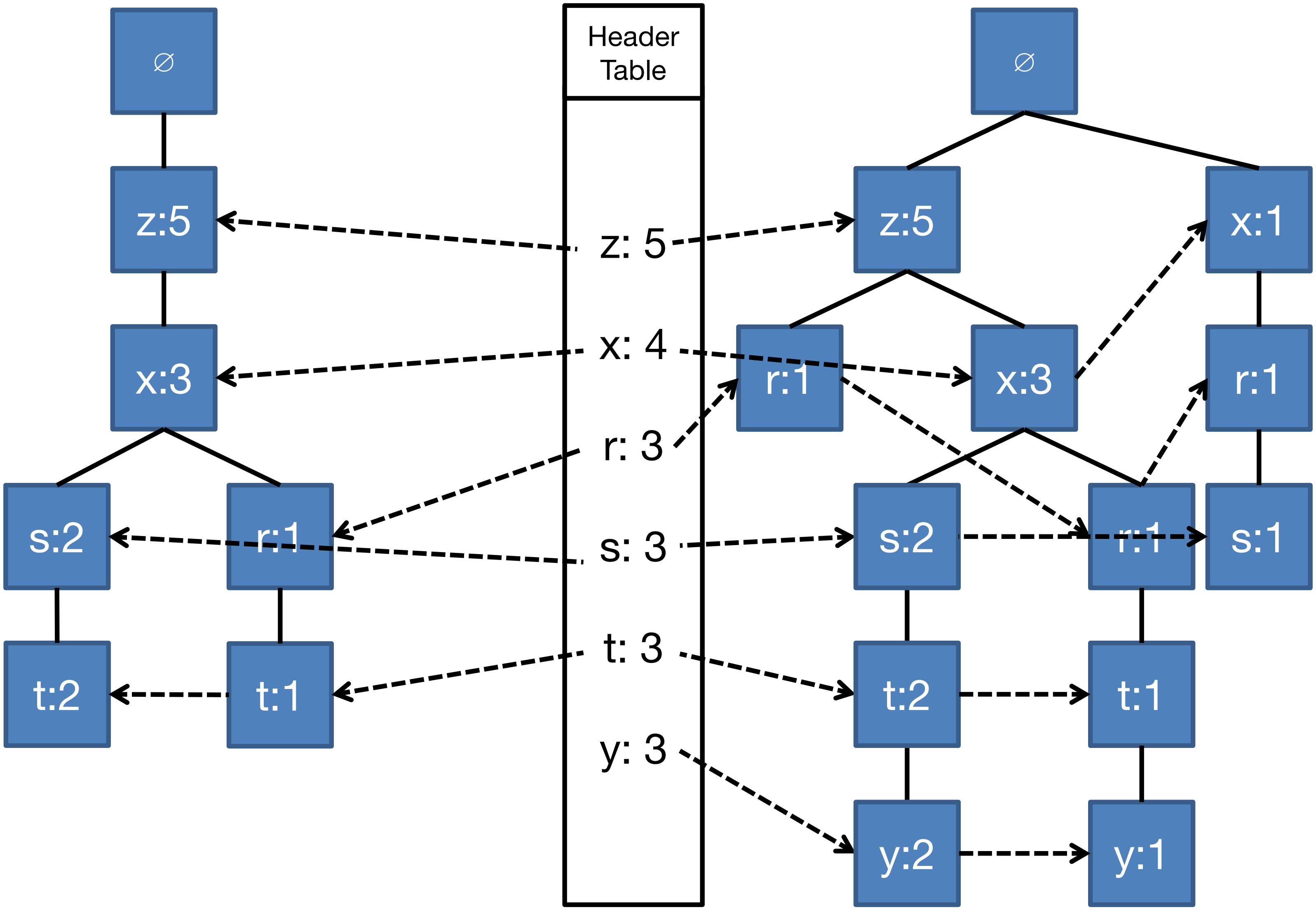
- Algorithm sketch…
  1. For each candidate item i
     a) Extract subtrees ending in i
        – "Conditional pattern base"
     b) Construct Conditional FP-Tree for i
     c) Recurse!

Example FP-Tree: Ending in y



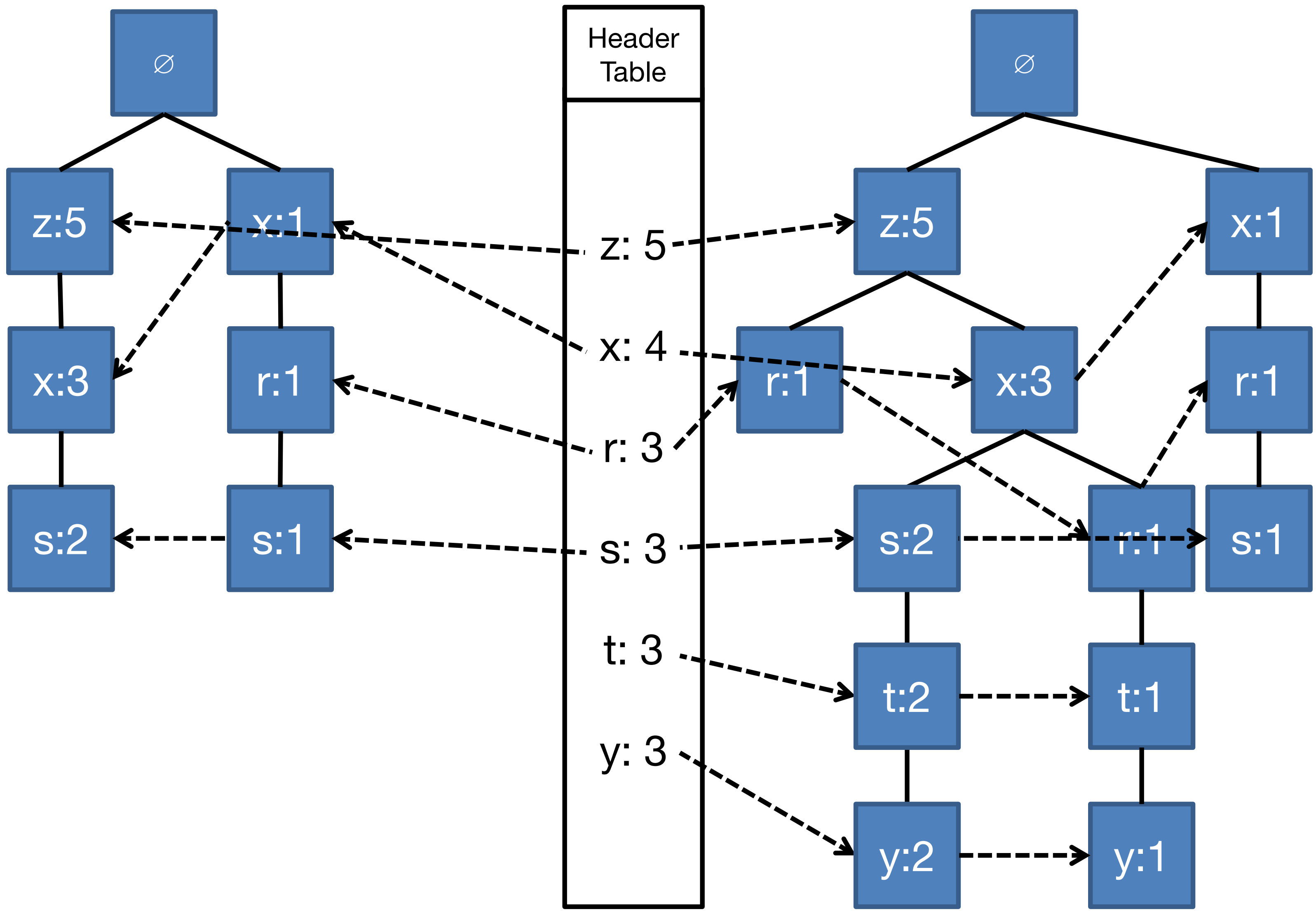
s_{min}=3

Example FP-Tree: Ending in t



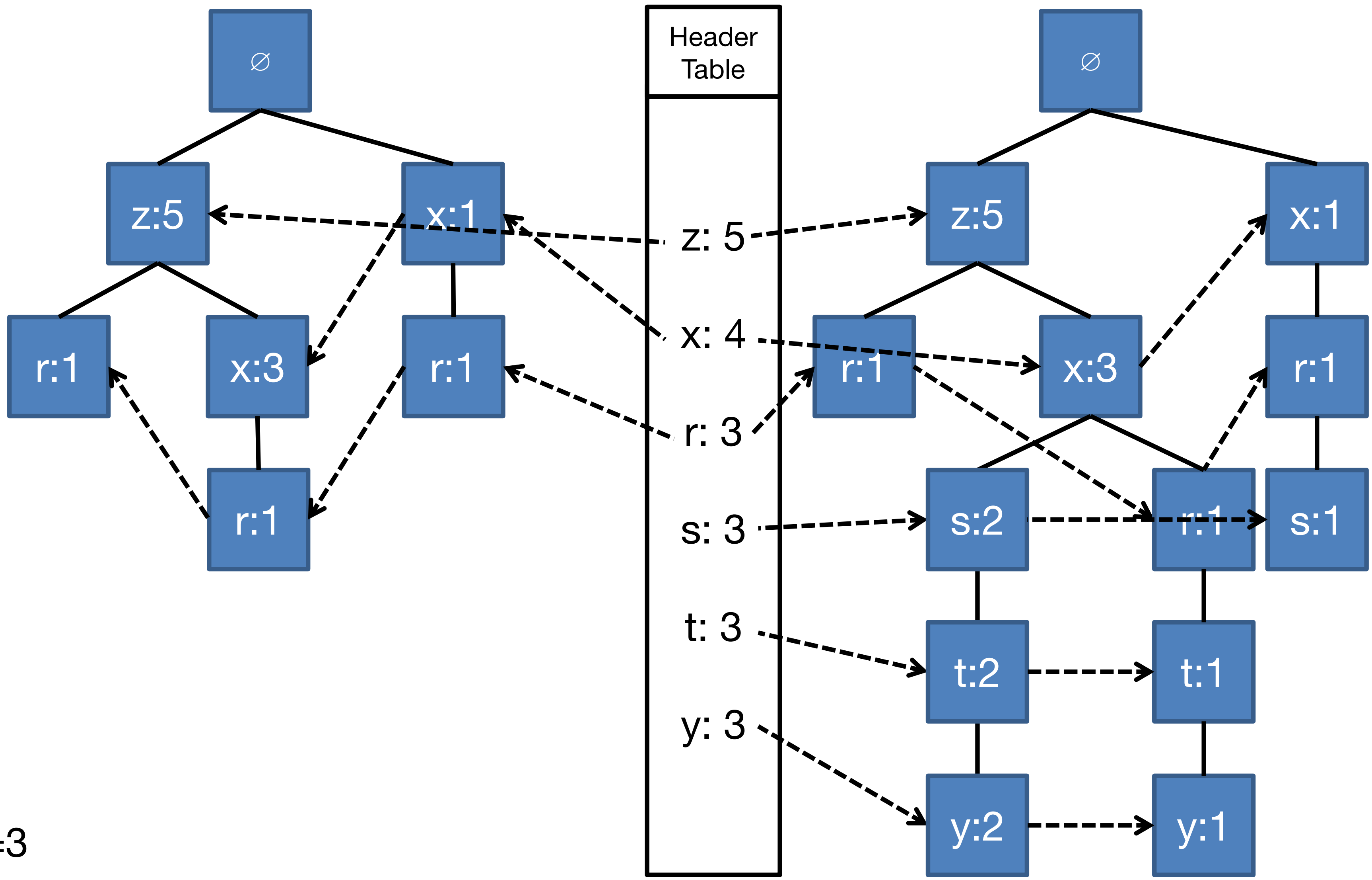
s_{min}=3

Example FP-Tree: Ending in s



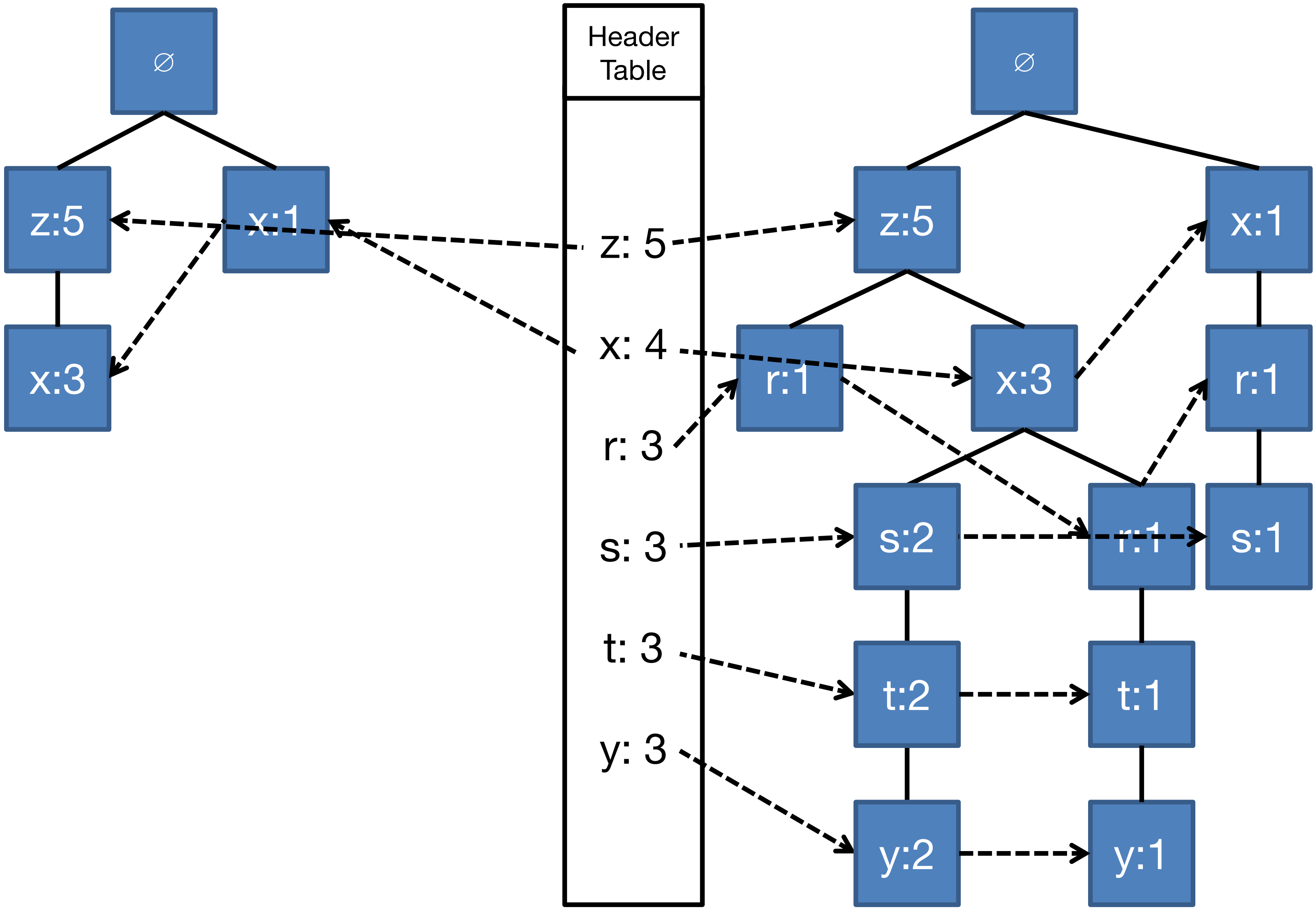
s_{min}=3

Example FP-Tree: Ending in r



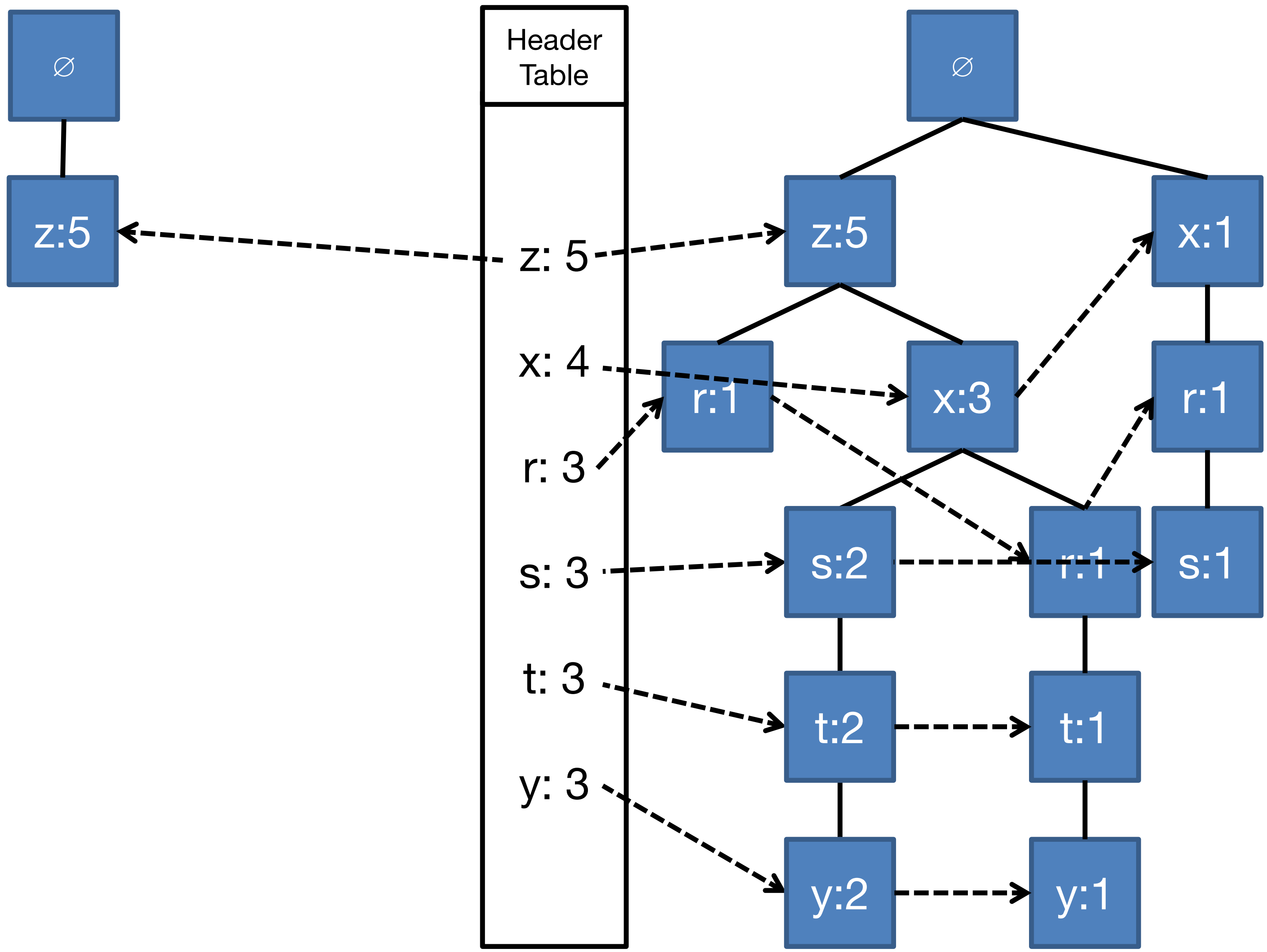
s_{min}=3

Example FP-Tree: Ending in x



s_{min}=3

Example FP-Tree: Ending in z



$s_{min}=3$

Before Conditional

- Given a subtree ending in i
  - Follow the pointers for i, add up support
  - If greater than threshold
    - Extract as frequent itemset
    - Continue

- Note: we know this is true for the top level (items would be there otherwise), but may not be the case for recursive sub-problems

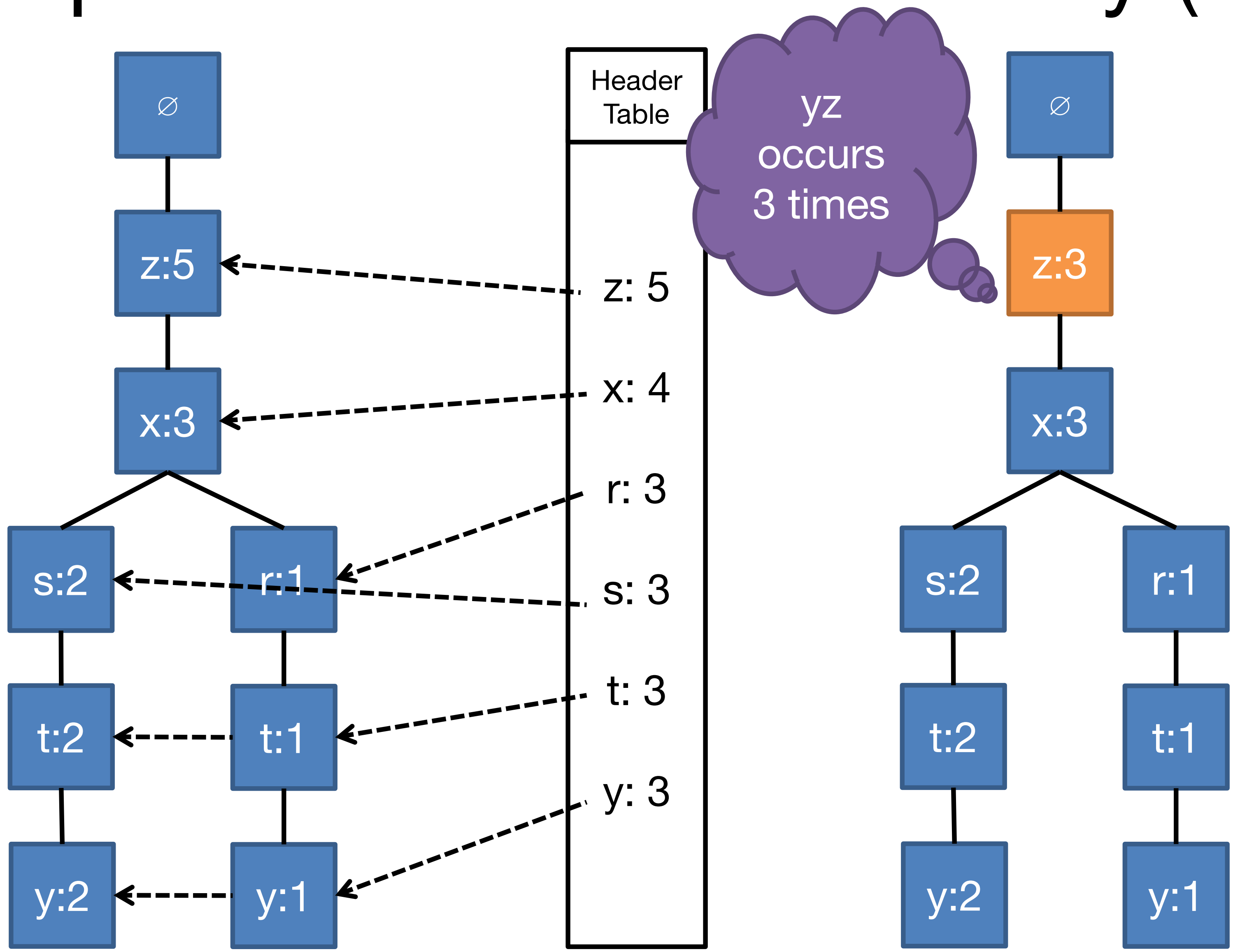
Scorecard :)

Frequent Item	Frequent Itemsets
z	{z}
x	{x}
r	{r}
s	{s}
t	{t}
y	{y}

Conditional FP-Tree

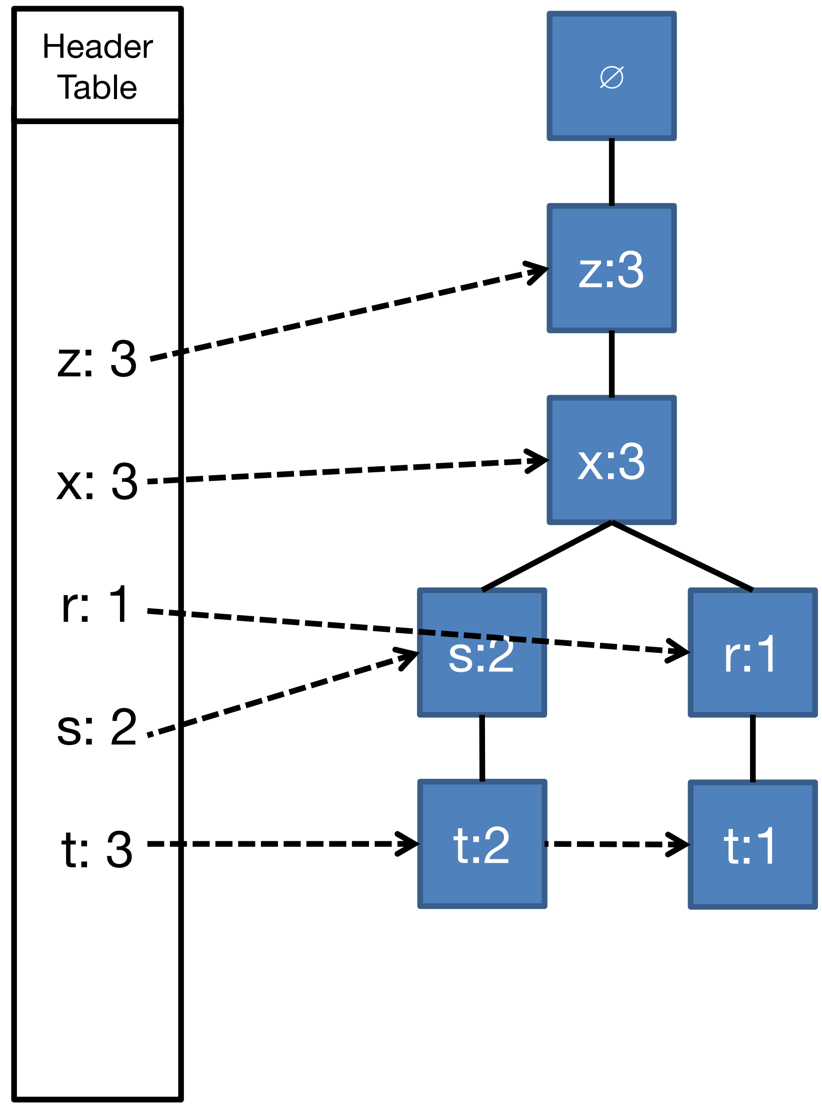
- A new FP-Tree representing prefixes of a removed itemset
- Basic flow
  - Update counts in subtree
  - Remove leaves
  - Remove infrequent nodes

Example FP-Tree: Conditional y (1)



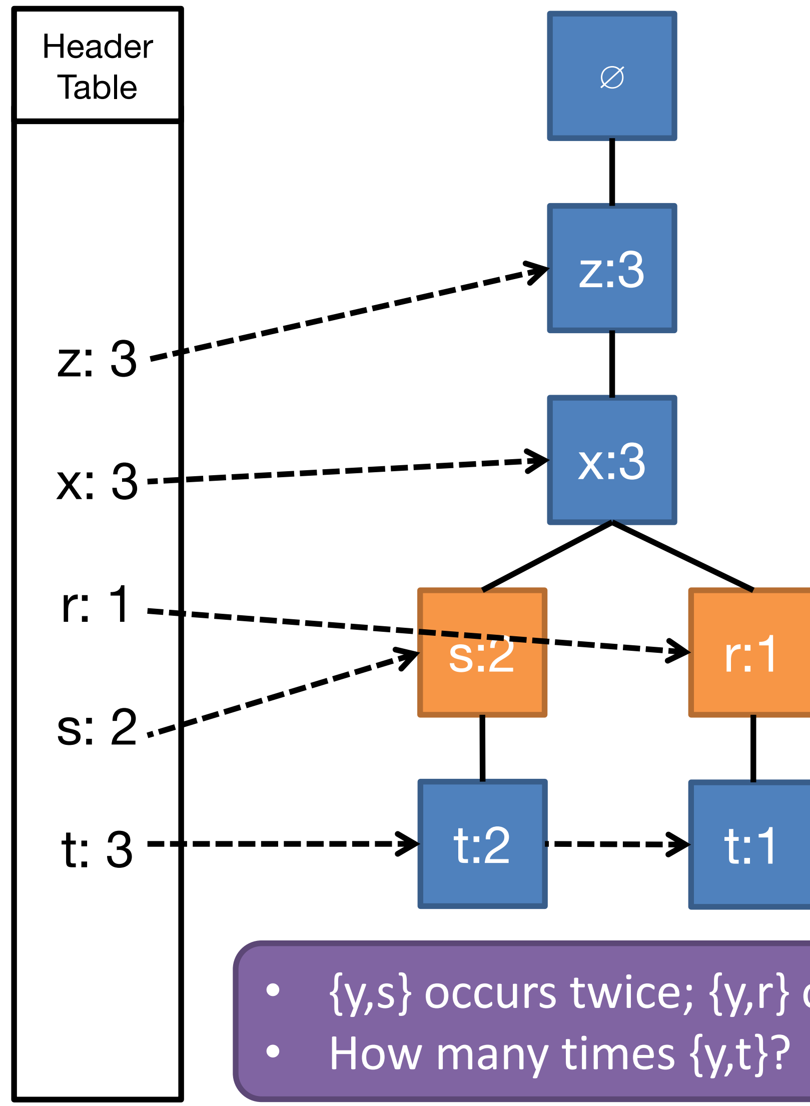
$s_{min}=3$

Example FP-Tree: Conditional y (2)



$s_{min}=3$

Example FP-Tree: Conditional y (2)

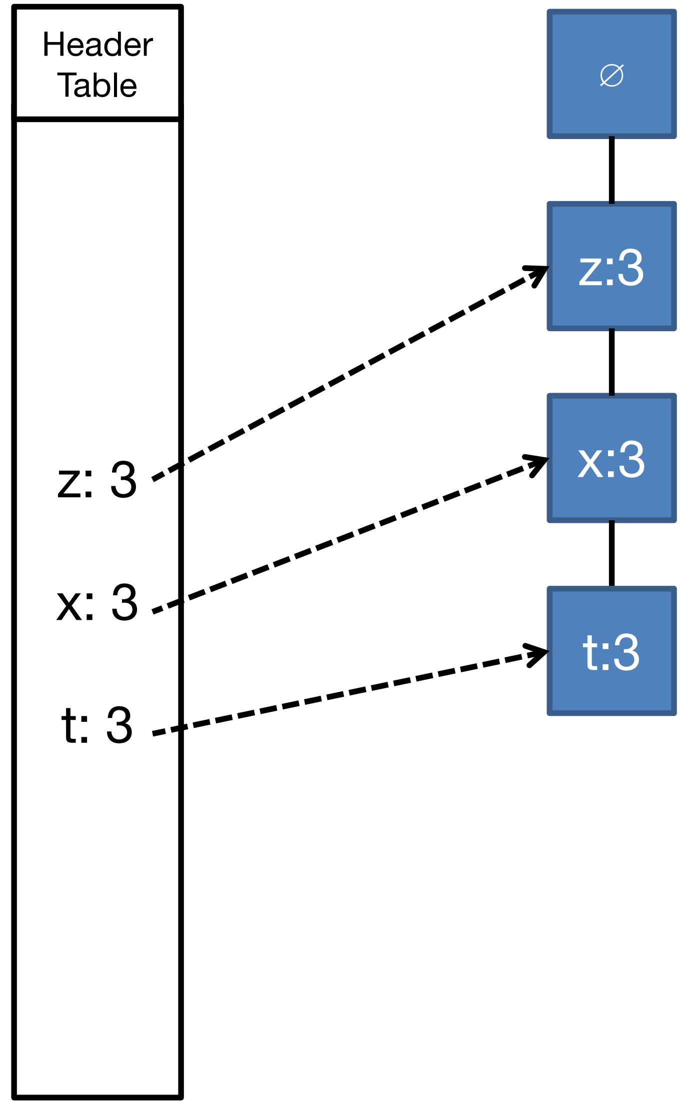


- {y,s} occurs twice; {y,r} once
- How many times {y,t}?

$s_{min}=3$

Example FP-Tree: Conditional y

- Recurse…
  - Prefix: yt, yx, yz

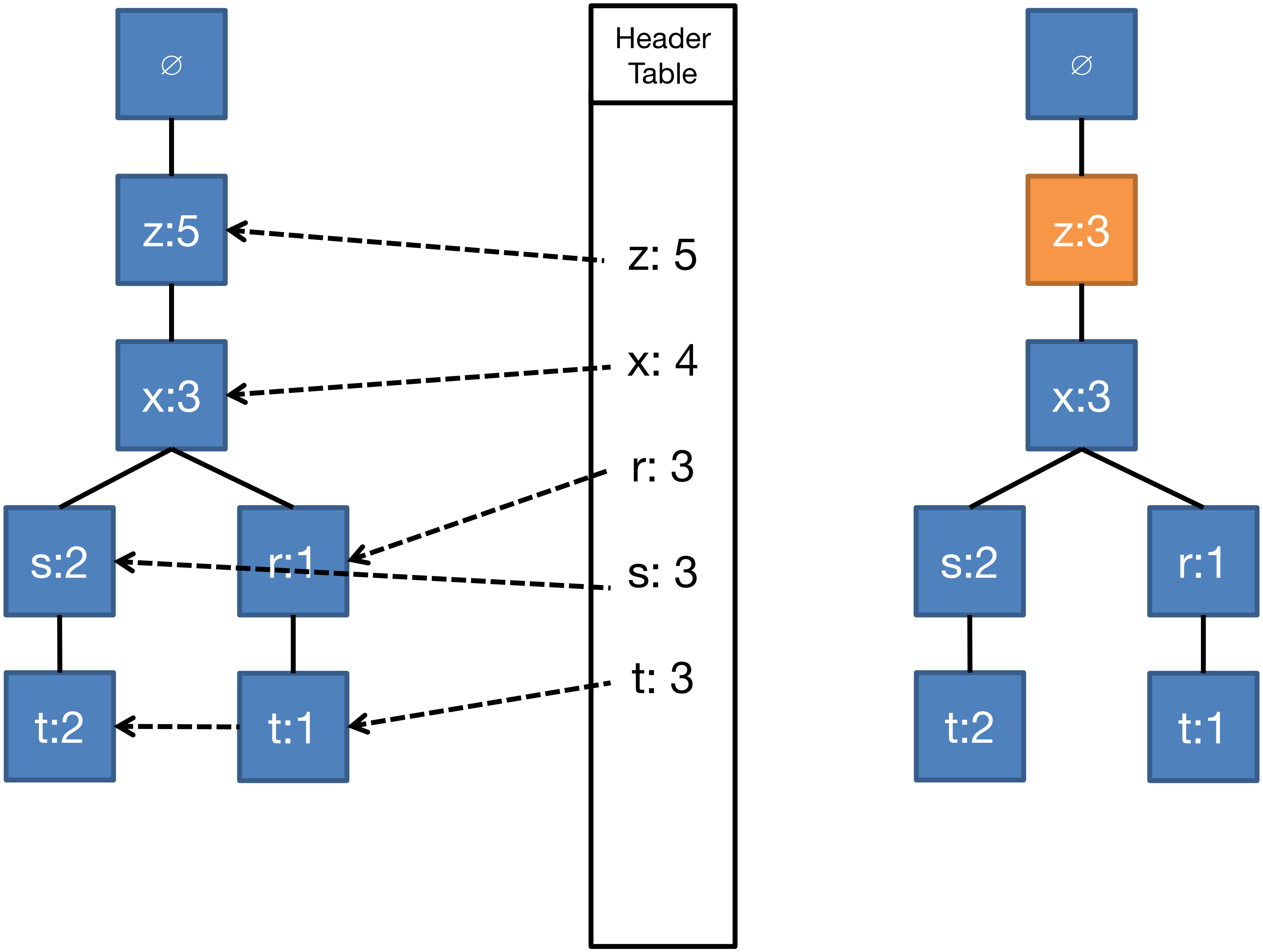


s_{min}=3

Scorecard :)

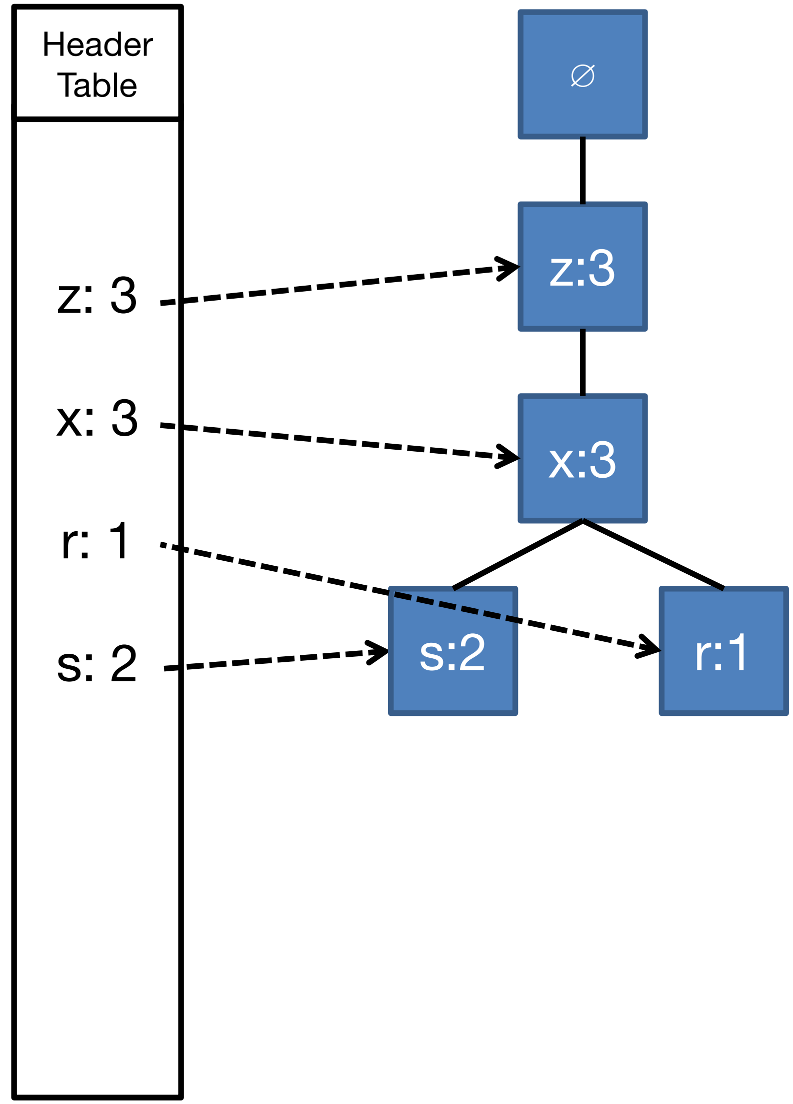
Frequent Item	Frequent Itemsets
z	{z}
x	{x}
r	{r}
s	{s}
t	{t}
y	{y}, {y,t}, {y,x}, {y,z}, {y,t,x}, {y,t,z}, {y,x,z}, {y,t,x,z}

Example FP-Tree: Conditional t (1)



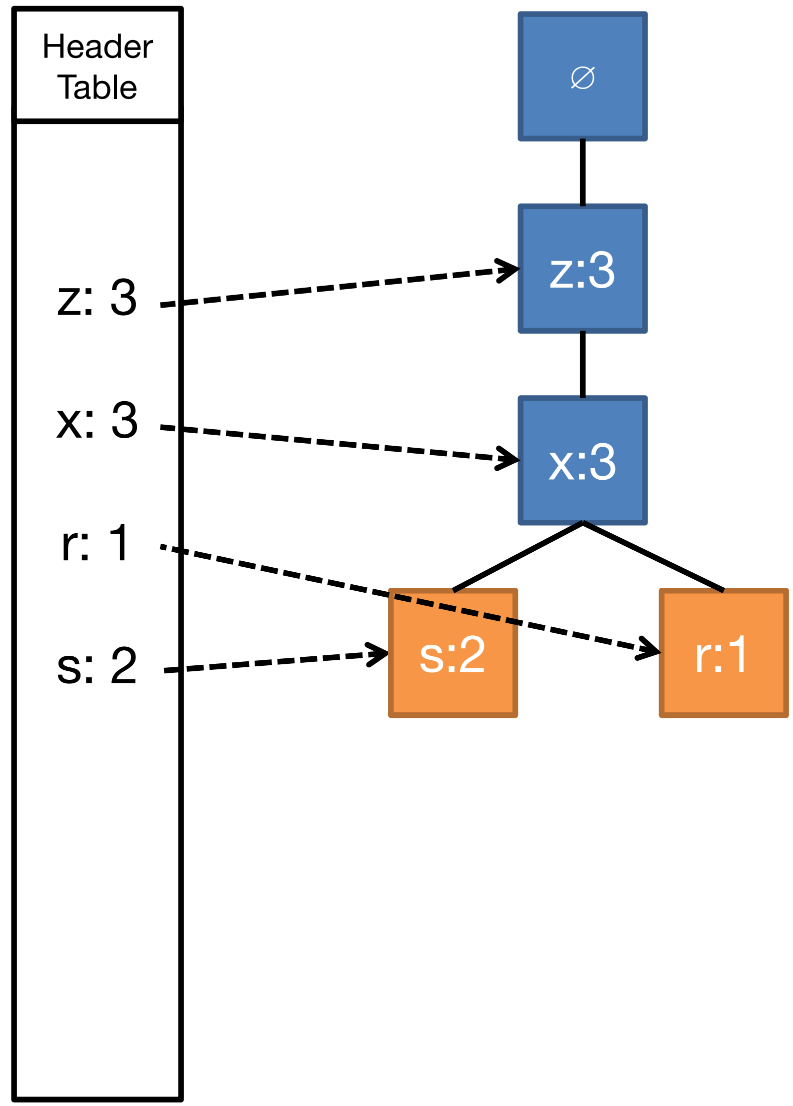
s_{min}=3

Example FP-Tree: Conditional t (2)



$s_{min}=3$

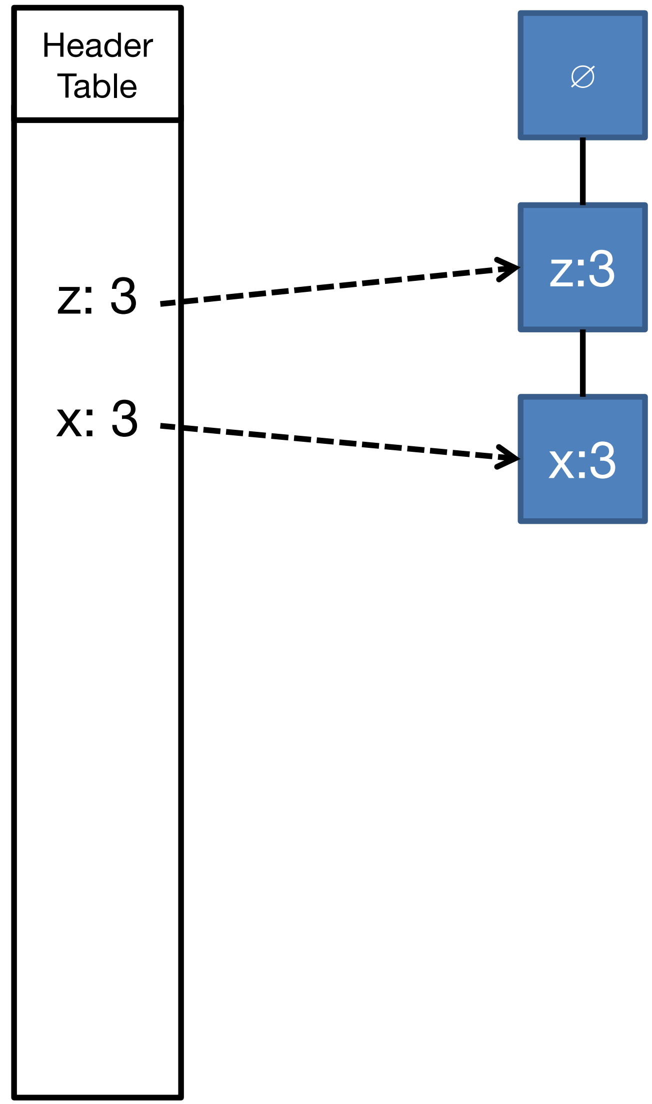
Example FP-Tree: Conditional t (2)



$s_{min}=3$

Example FP-Tree: Conditional t

- Recurse…
  - Prefix: tx, tz

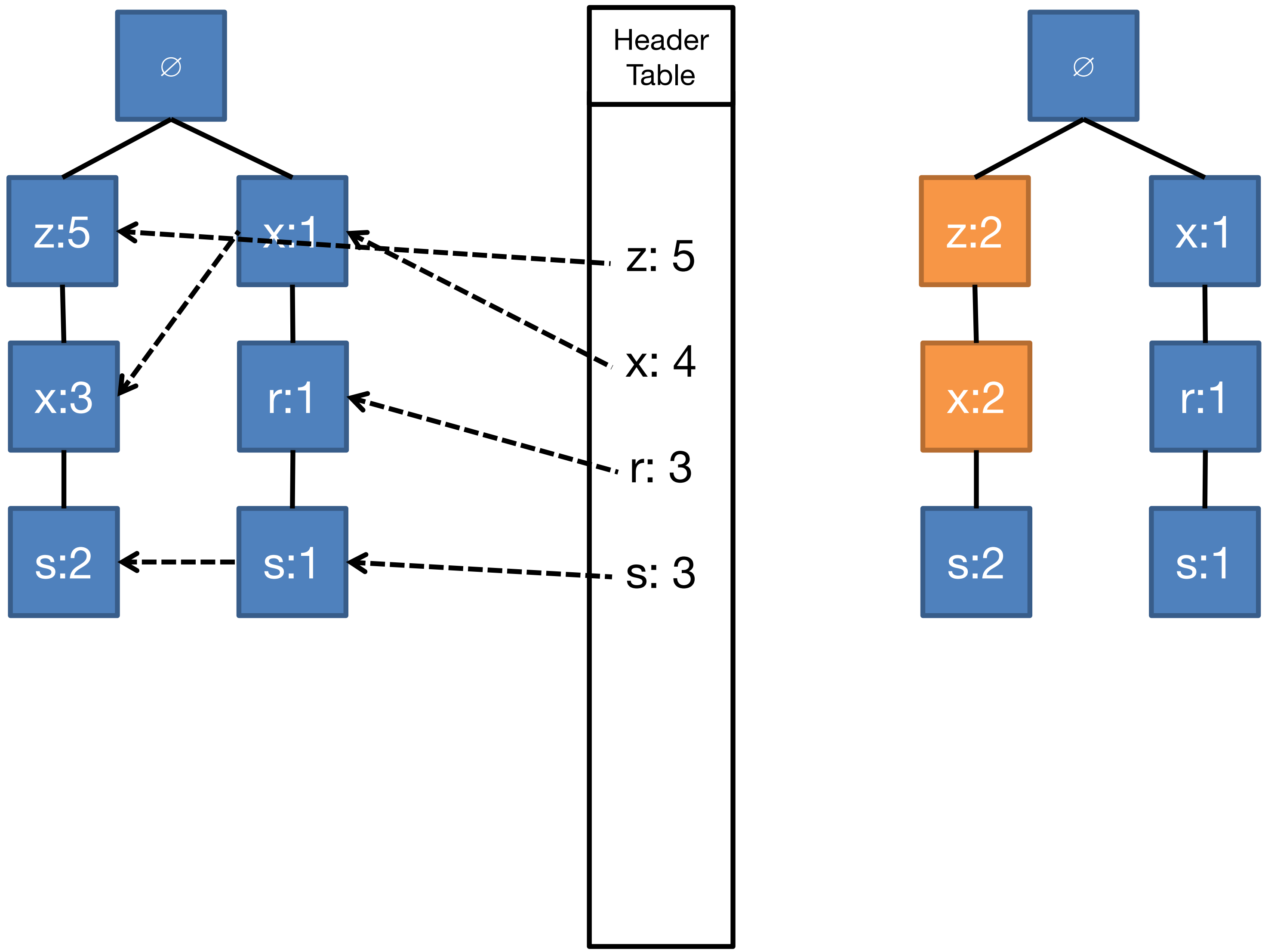


s_{min}=3

Scorecard :)

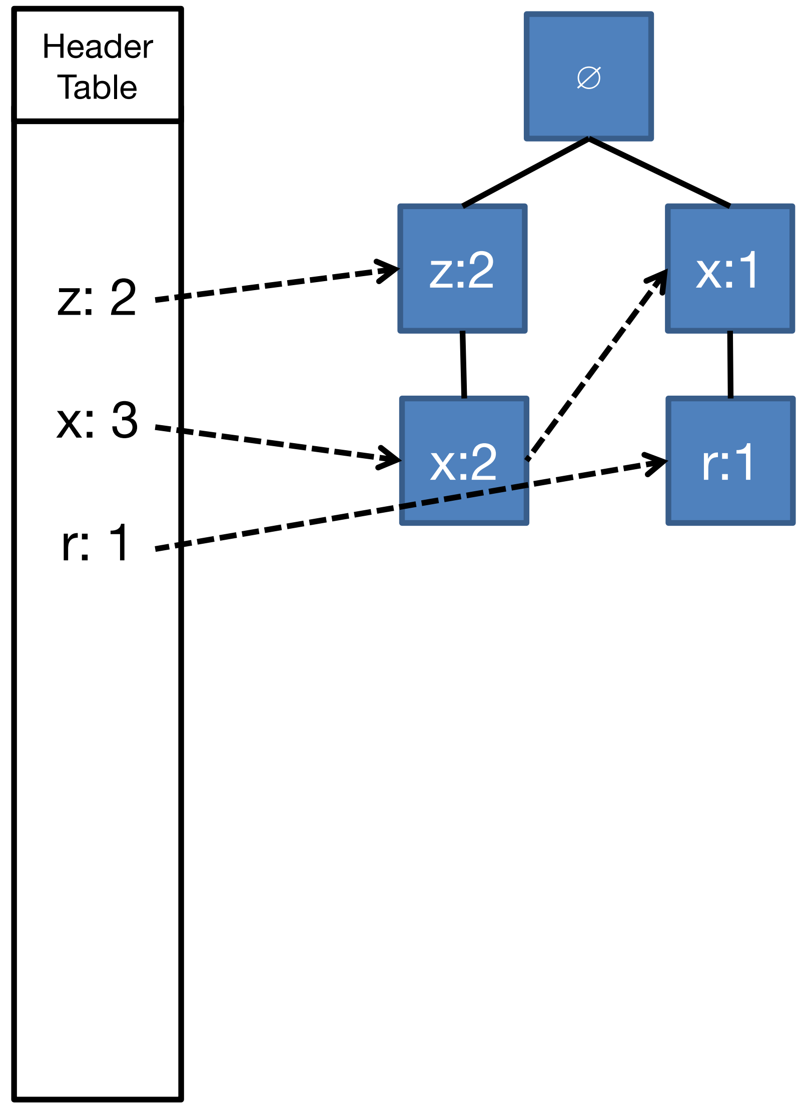
Frequent Item	Frequent Itemsets
z	{z}
x	{x}
r	{r}
s	{s}
t	{t}, {t,x}, {t,z}, {t,x,z}
y	{y}, {y,t}, {y,x}, {y,z}, {y,t,x}, {y,t,z}, {y,x,z}, {y,t,x,z}

Example FP-Tree: Conditional s (1)



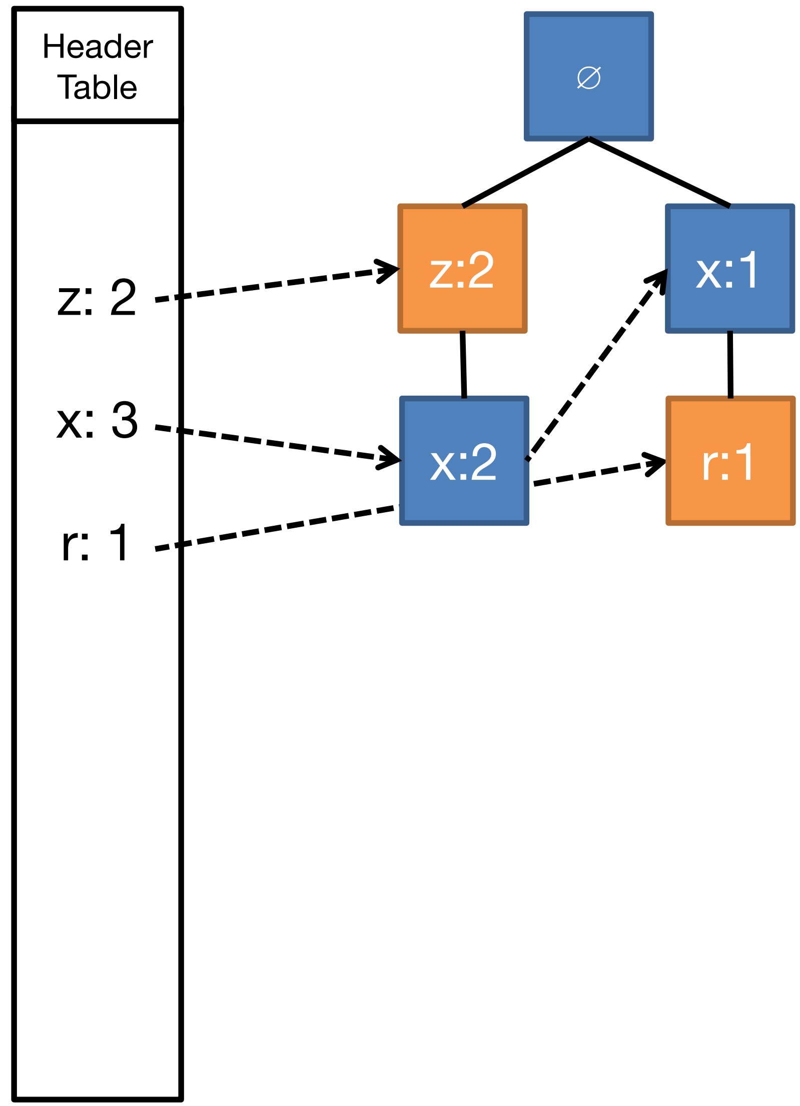
s_{min}=3

Example FP-Tree: Conditional s (2)



s_{min}=3

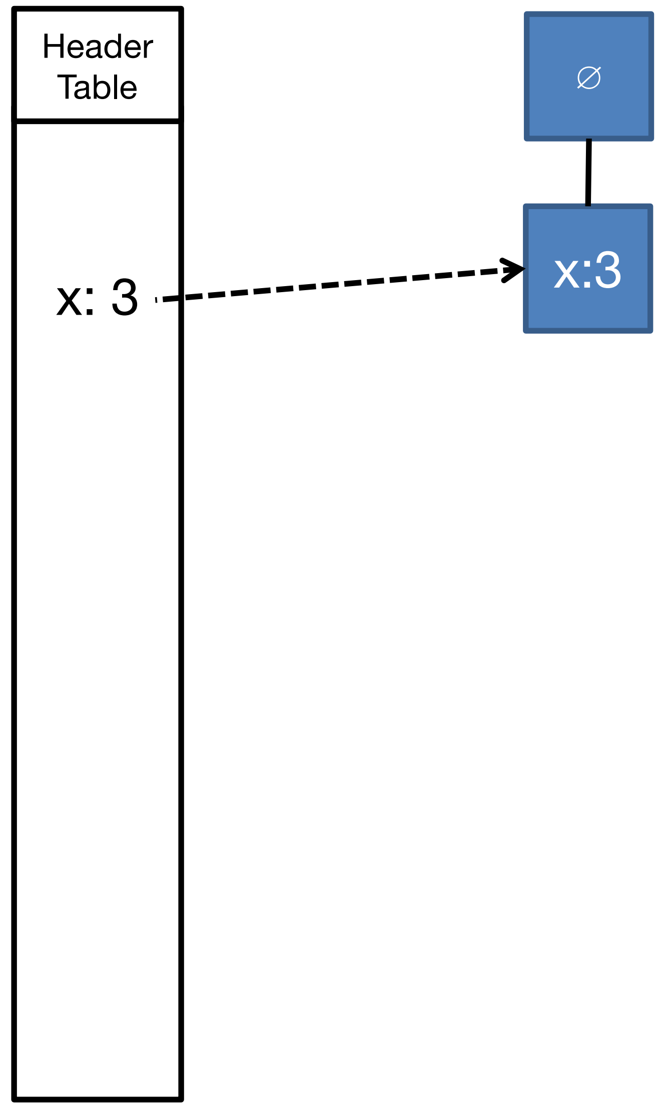
Example FP-Tree: Conditional s (2)



s_{min}=3

Example FP-Tree: Conditional s

- Recurse…
  - Prefix: sx

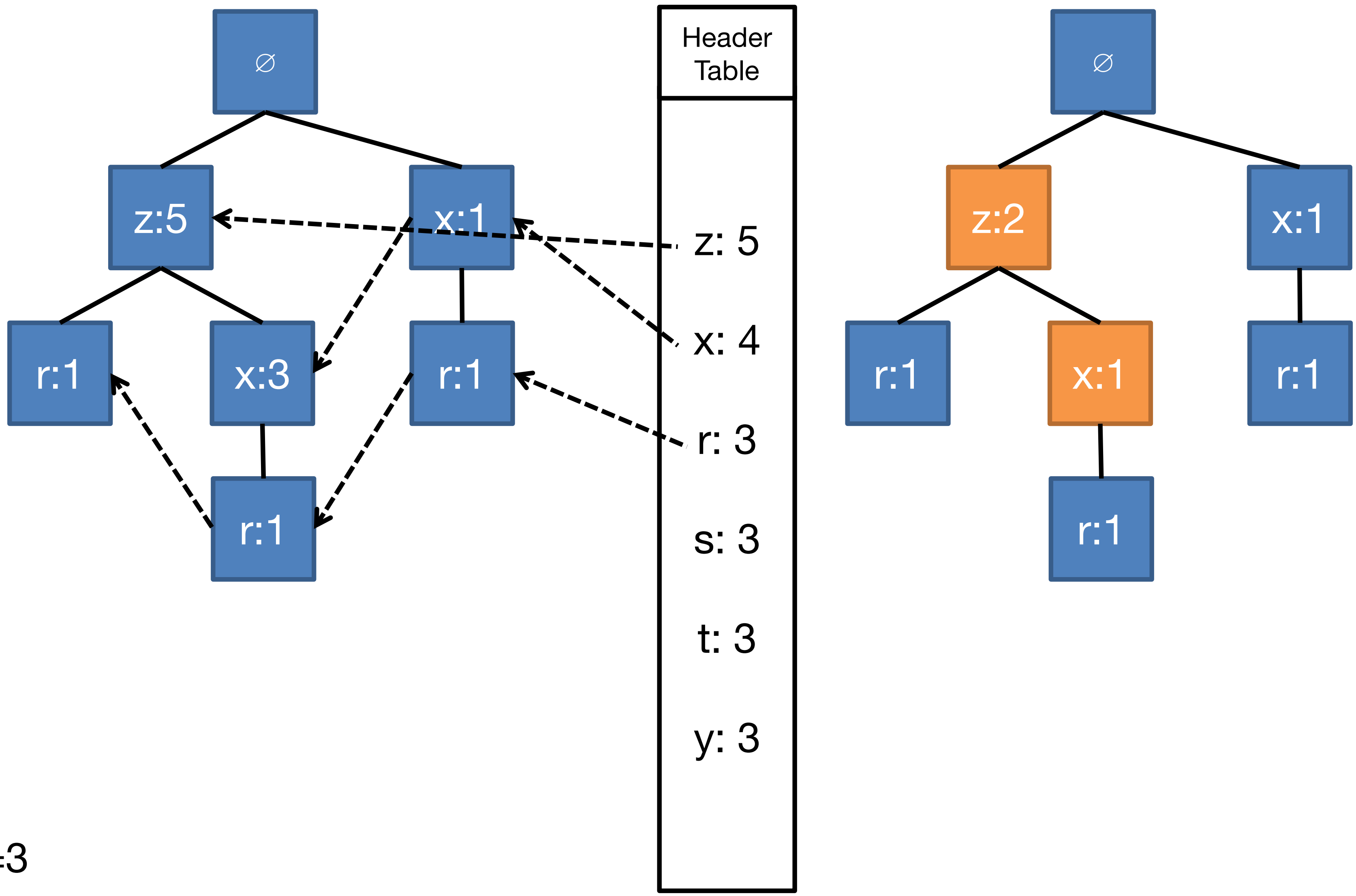


$s_{min}=3$

Scorecard :)

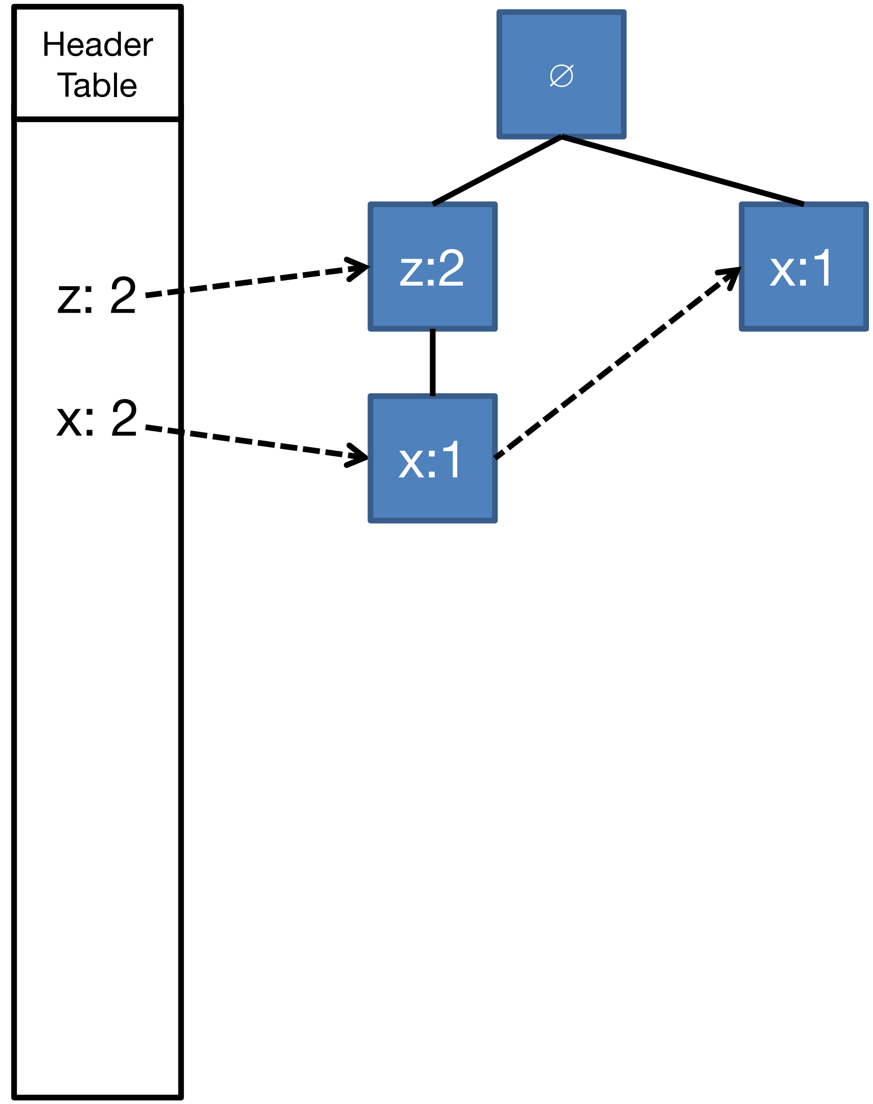
Frequent Item	Frequent Itemsets
z	{z}
x	{x}
r	{r}
s	{s}, {s,x}
t	{t}, {t,x}, {t,z}, {t,x,z}
y	{y}, {y,t}, {y,x}, {y,z}, {y,t,x}, {y,t,z}, {y,x,z}, {y,t,x,z}

Example FP-Tree: Conditional r (1)



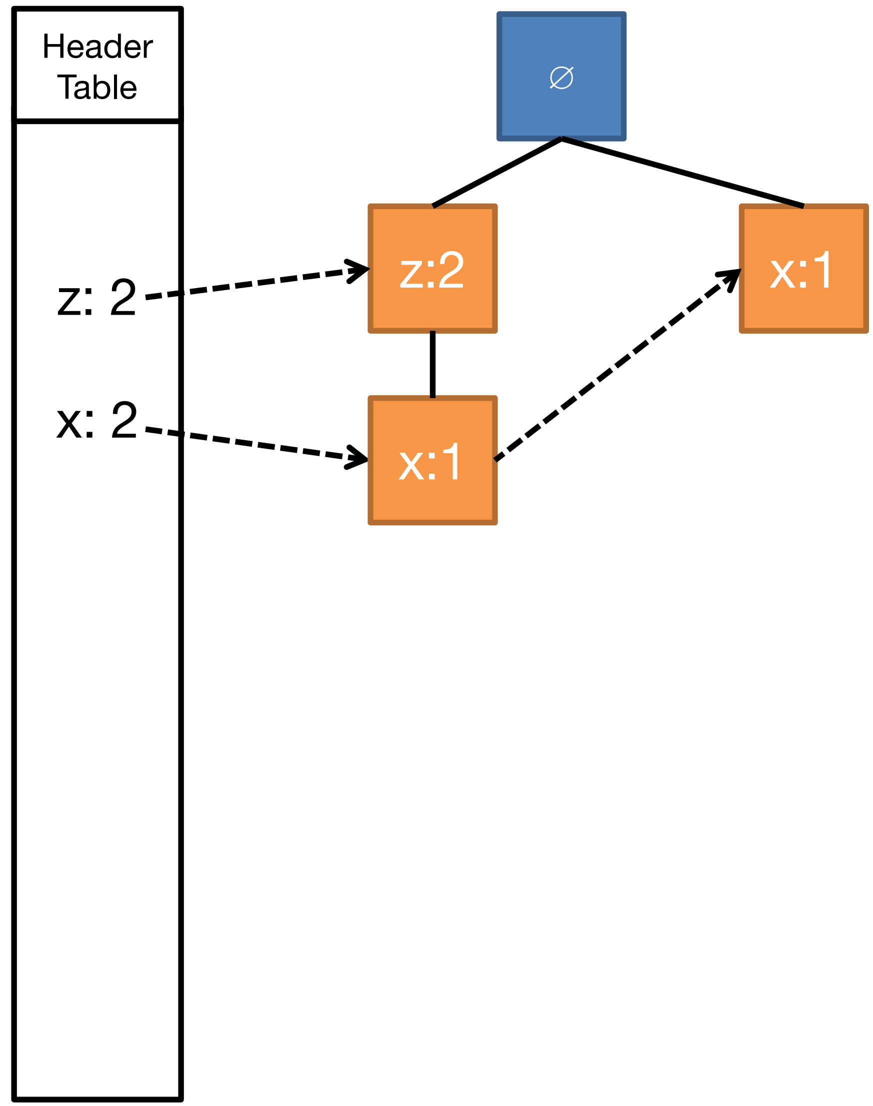
$s_{min}=3$

Example FP-Tree: Conditional r (2)



$s_{min}=3$

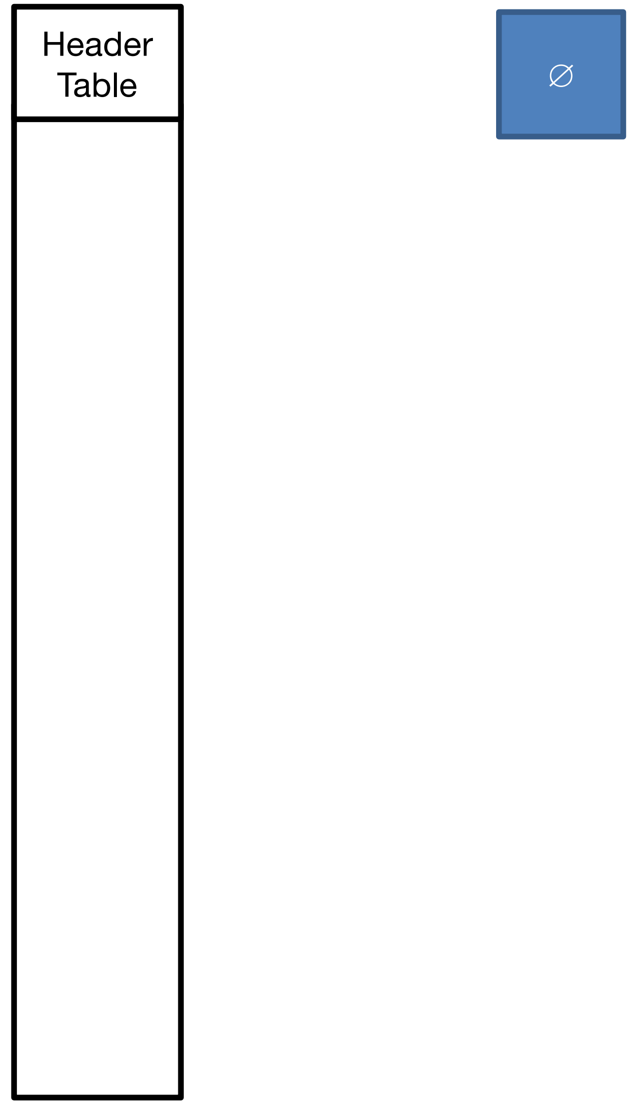
Example FP-Tree: Conditional r (2)



s_{min}=3

Example FP-Tree: Conditional r

- No recursion

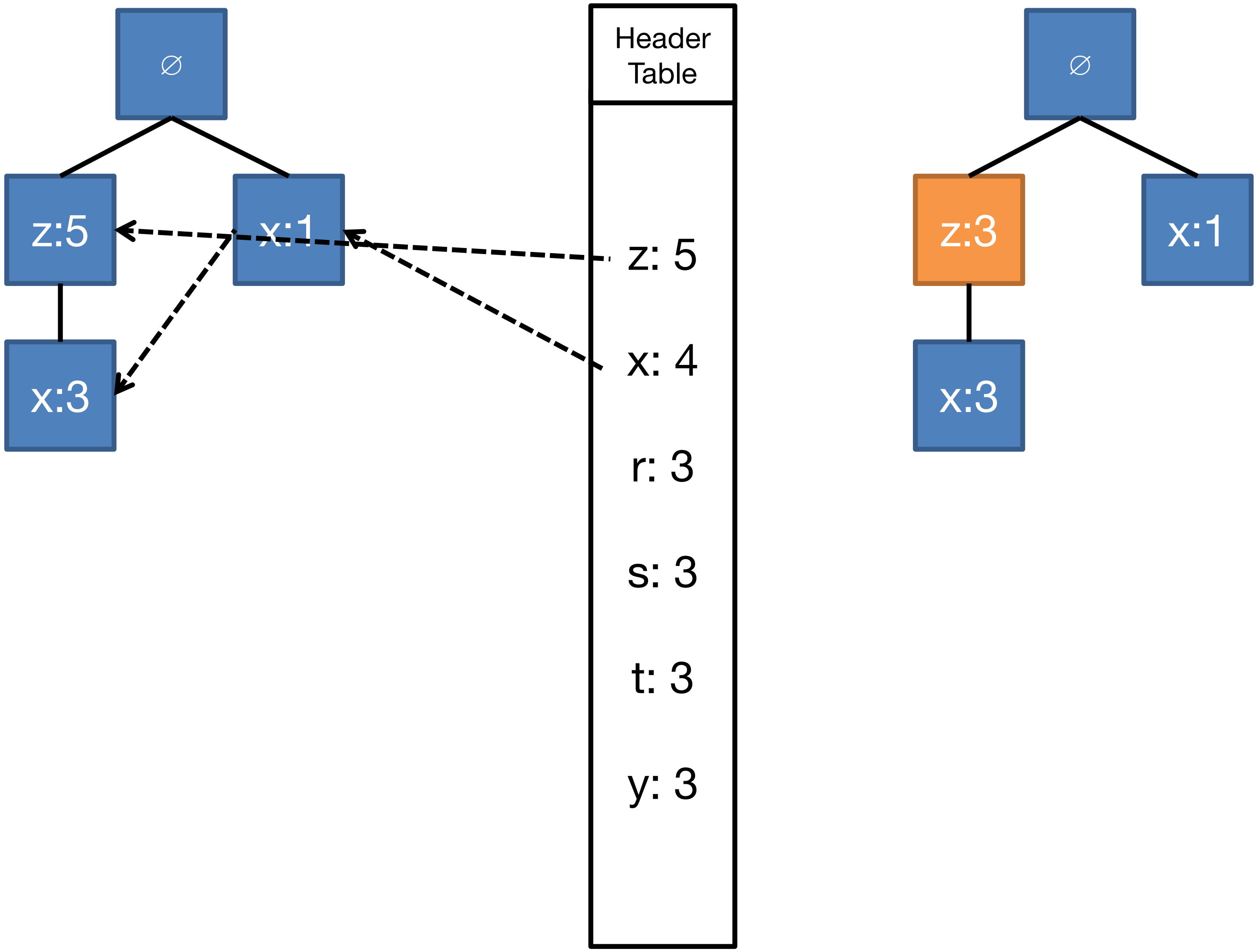


s_{min}=3

Scorecard :)

Frequent Item	Frequent Itemsets
z	{z}
x	{x}
r	{r}
s	{s}, {s,x}
t	{t}, {t,x}, {t,z}, {t,x,z}
y	{y}, {y,t}, {y,x}, {y,z}, {y,t,x}, {y,t,z}, {y,x,z}, {y,t,x,z}

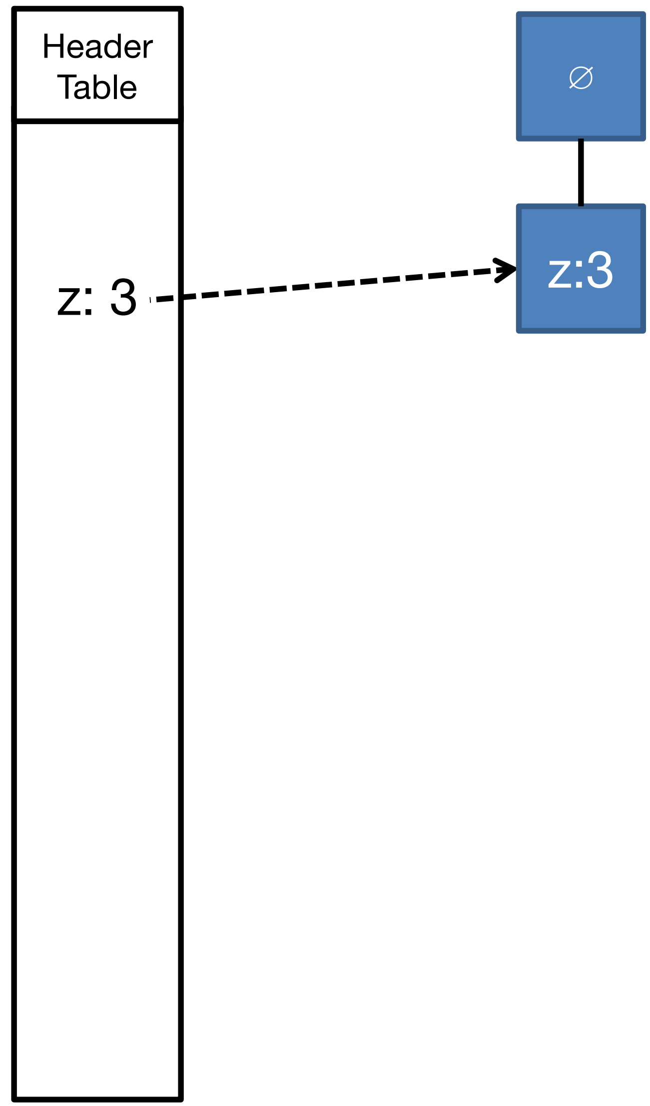
Example FP-Tree: Conditional x (1)



$s_{min}=3$

Example FP-Tree: Conditional x

- Recurse…
  - Prefix: xz

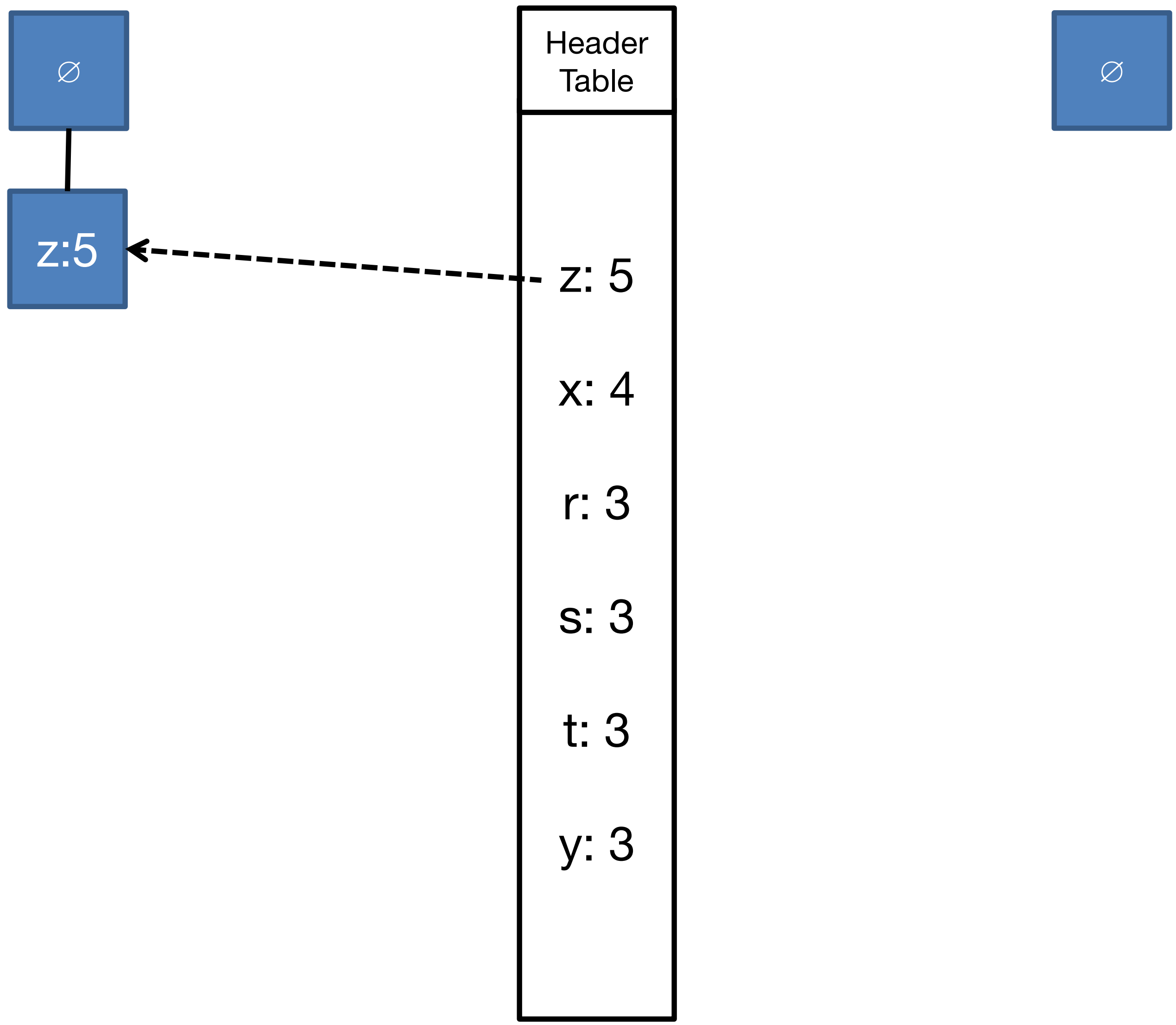


s_{min}=3

Scorecard :)

Frequent Item	Frequent Itemsets
z	{z}
x	{x}, {x,z}
r	{r}
s	{s}, {s,x}
t	{t}, {t,x}, {t,z}, {t,x,z}
y	{y}, {y,t}, {y,x}, {y,z}, {y,t,x}, {y,t,z}, {y,x,z}, {y,t,x,z}

Example FP-Tree: Conditional z



$s_{min}=3$

Final Scorecard!

Frequent Item	Frequent Itemsets
z	{z}
x	{x}, {x,z}
r	{r}
s	{s}, {s,x}
t	{t}, {t,x}, {t,z}, {t,x,z}
y	{y}, {y,t}, {y,x}, {y,z}, {y,t,x}, {y,t,z}, {y,x,z}, {y,t,x,z}

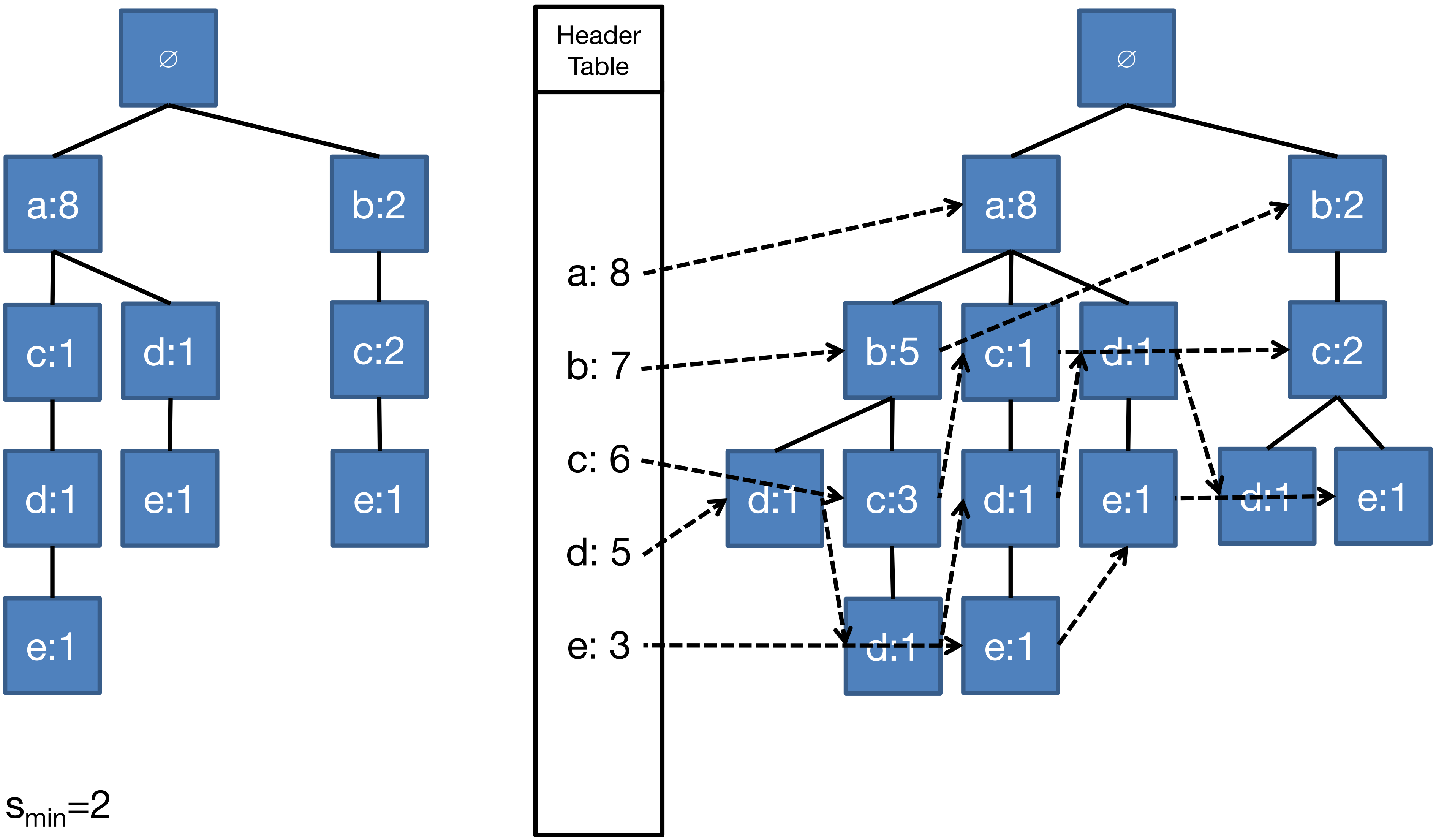
- r, s, t, x, y, z
- sx, tx, tz, xz, yt, yx, yz
- txz, ytx, ytz, yxz
- ytxz

Frequent Itemsets

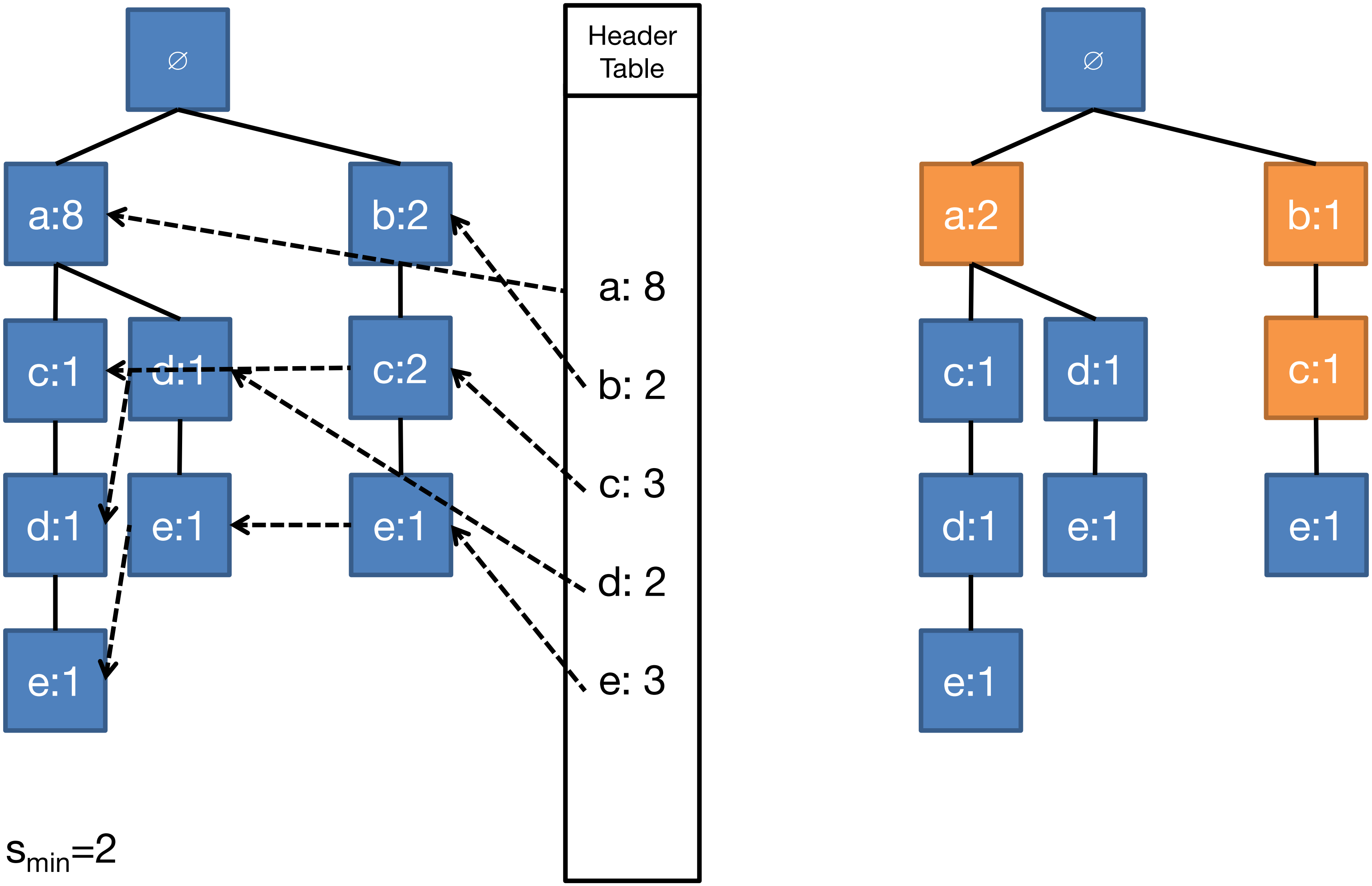
- z, r
- z, x, s, t, y
- z
- x, r, s
- z, x, r, t, y
- z, x, s, t, y

- r, s, t, x, y, z
- sx, tx, tz, xz, yt, yx, yz
- txz, ytx, ytz, yxz
- ytxz

Try: Ending in e

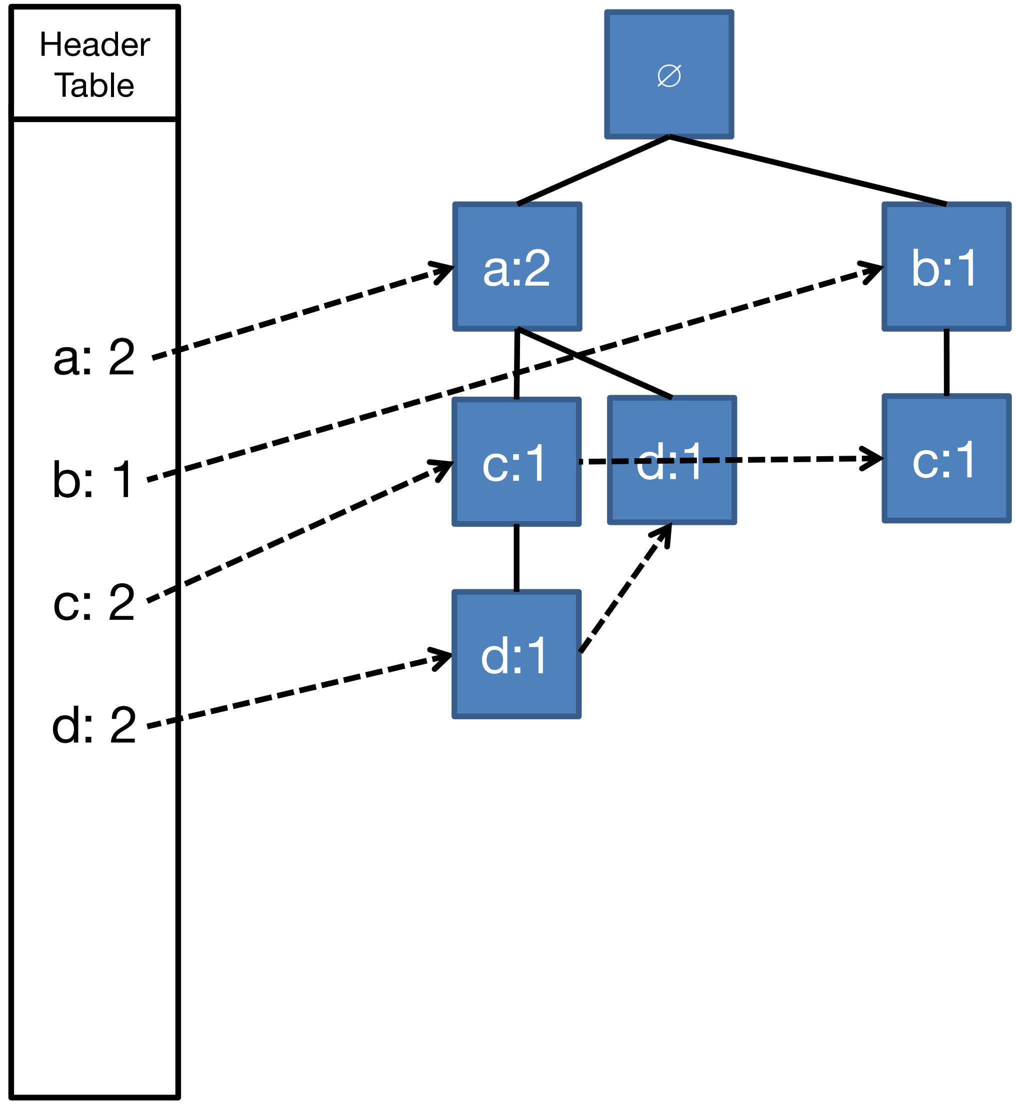


Try: Conditional e (1)



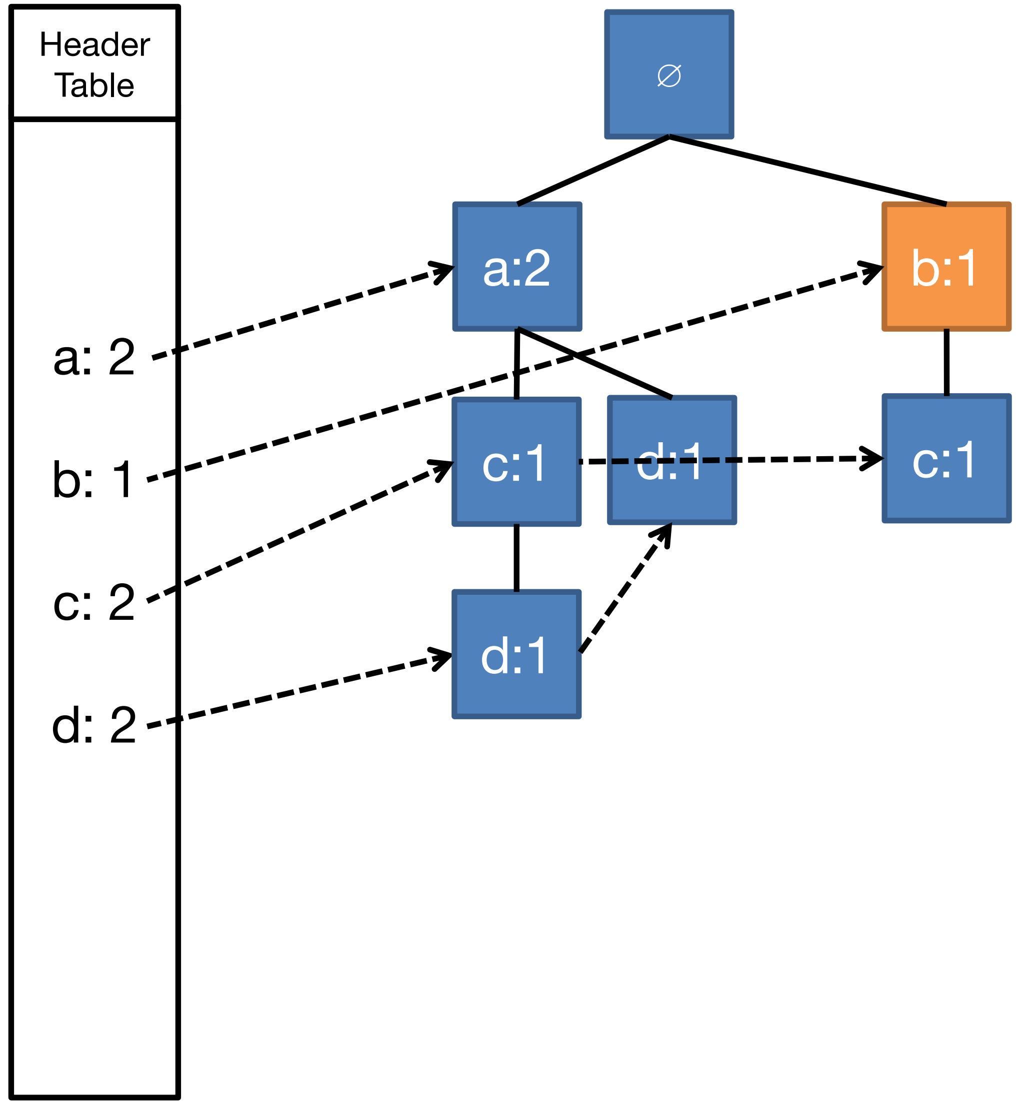
$s_{min}=2$

Try: Conditional e (2)



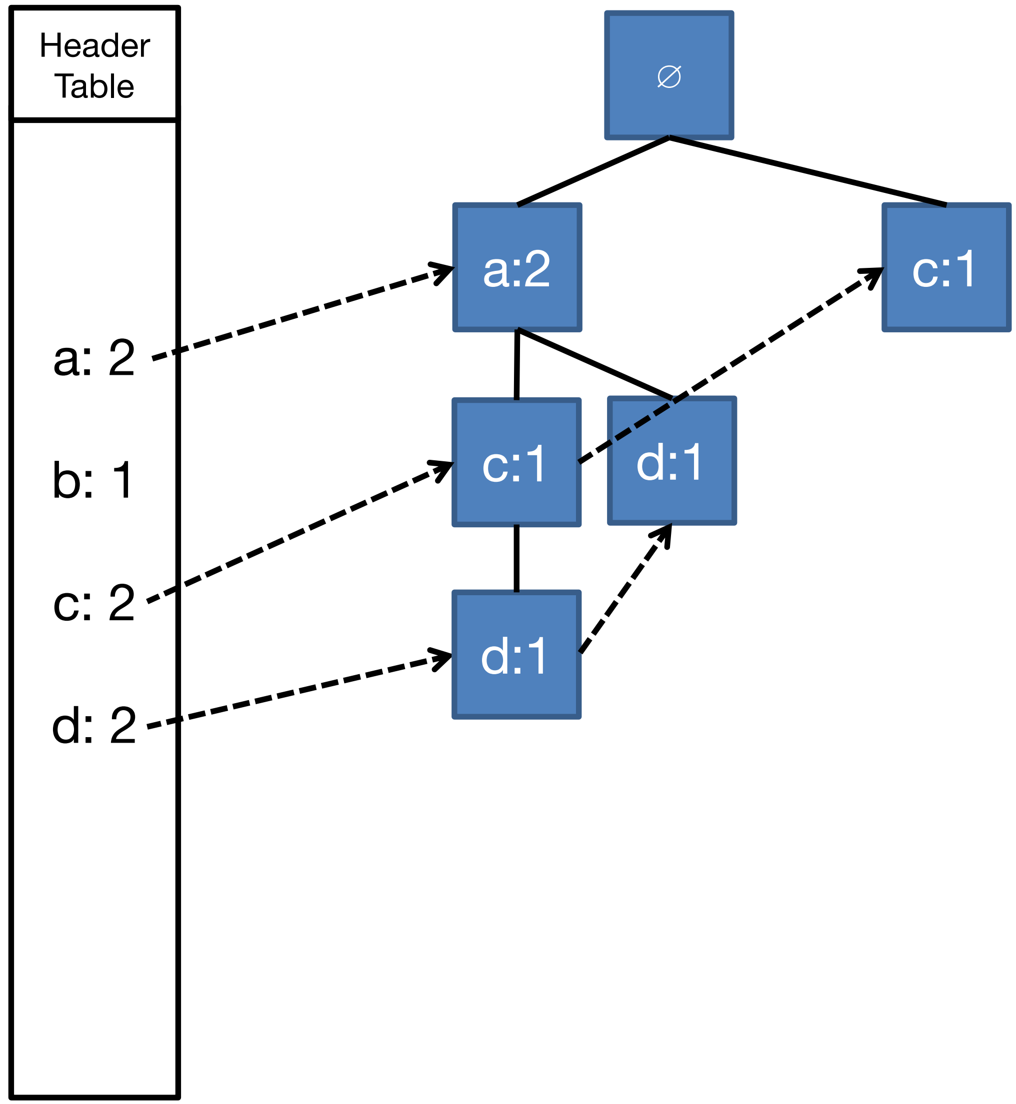
$s_{min}=2$

Try: Conditional e (2)



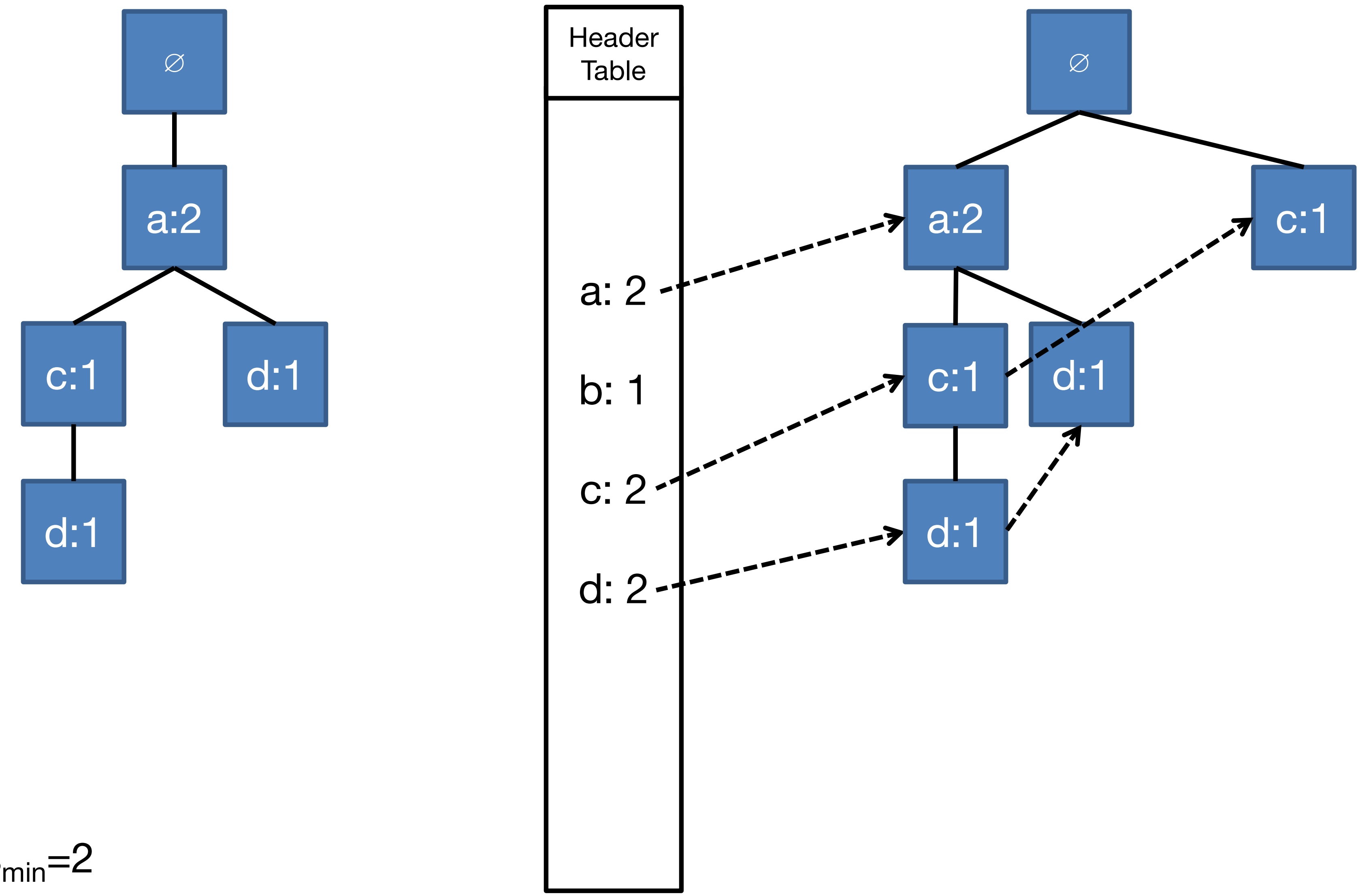
$s_{min}=2$

Try: Conditional e



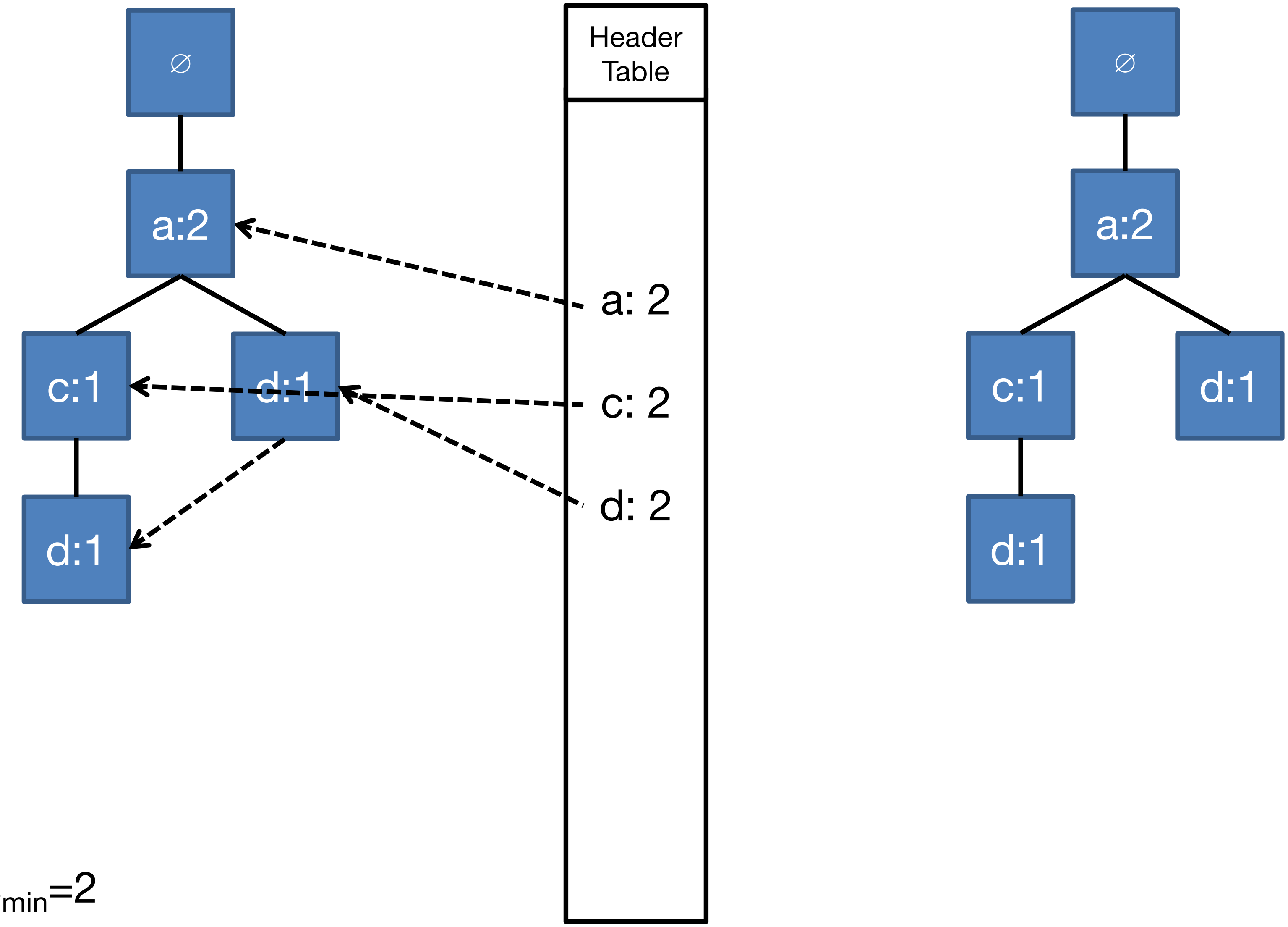
s_{min}=2

Try: Ending in de



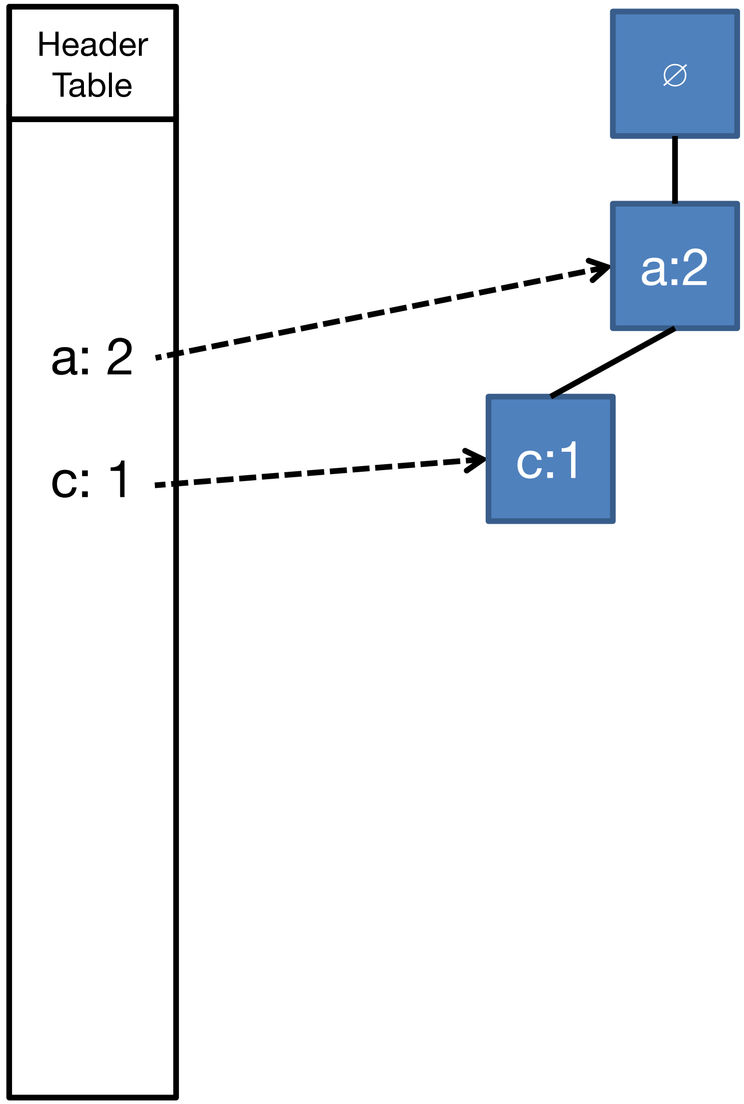
s_{min}=2

Try: Conditional de (1)



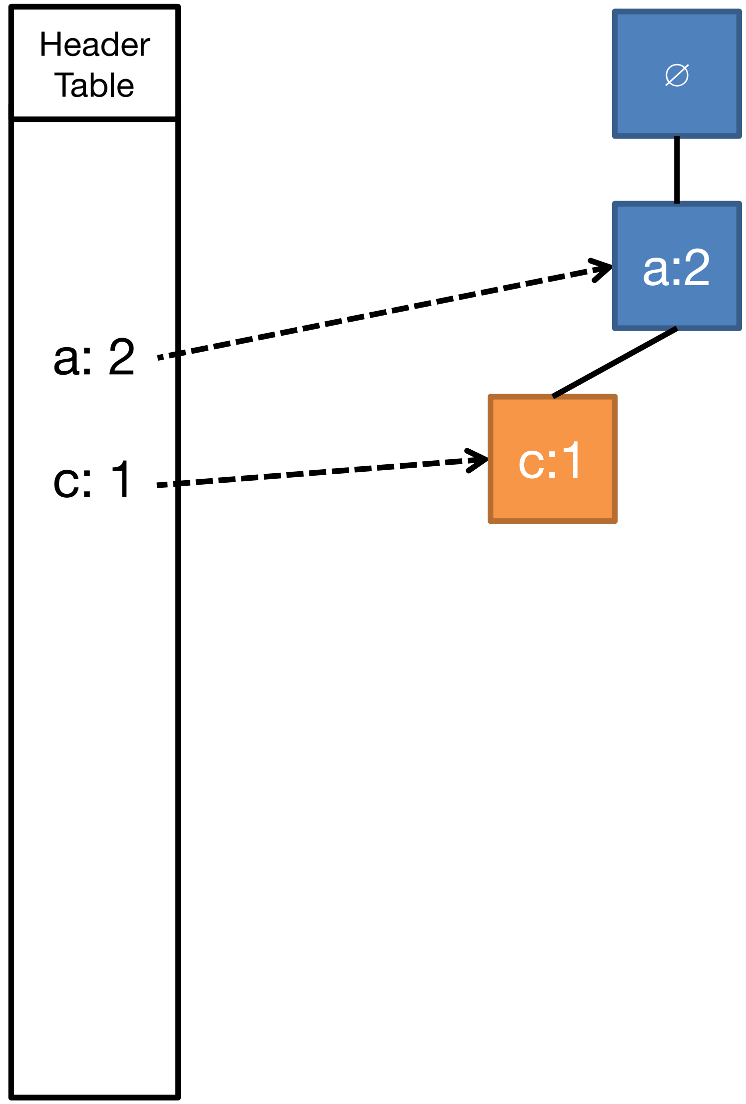
s_{min}=2

Try: Conditional de (2)



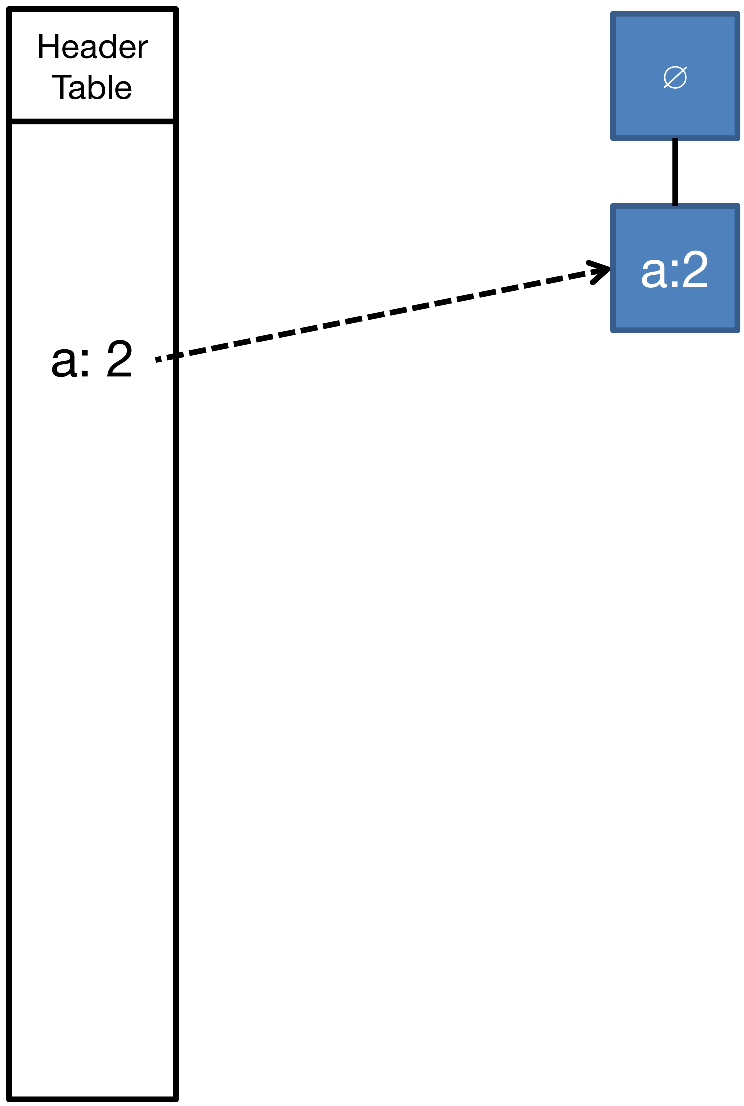
s_{min}=2

Try: Conditional de (2)



s_{min}=2

Try: Conditional de



$s_{min}=2$

FP-Growth Analysis

Advantages

- 2 passes over dataset (yay I/O!)
- "Compresses" dataset
- No candidate generation
- Typically much faster than Apriori

Disadvantages

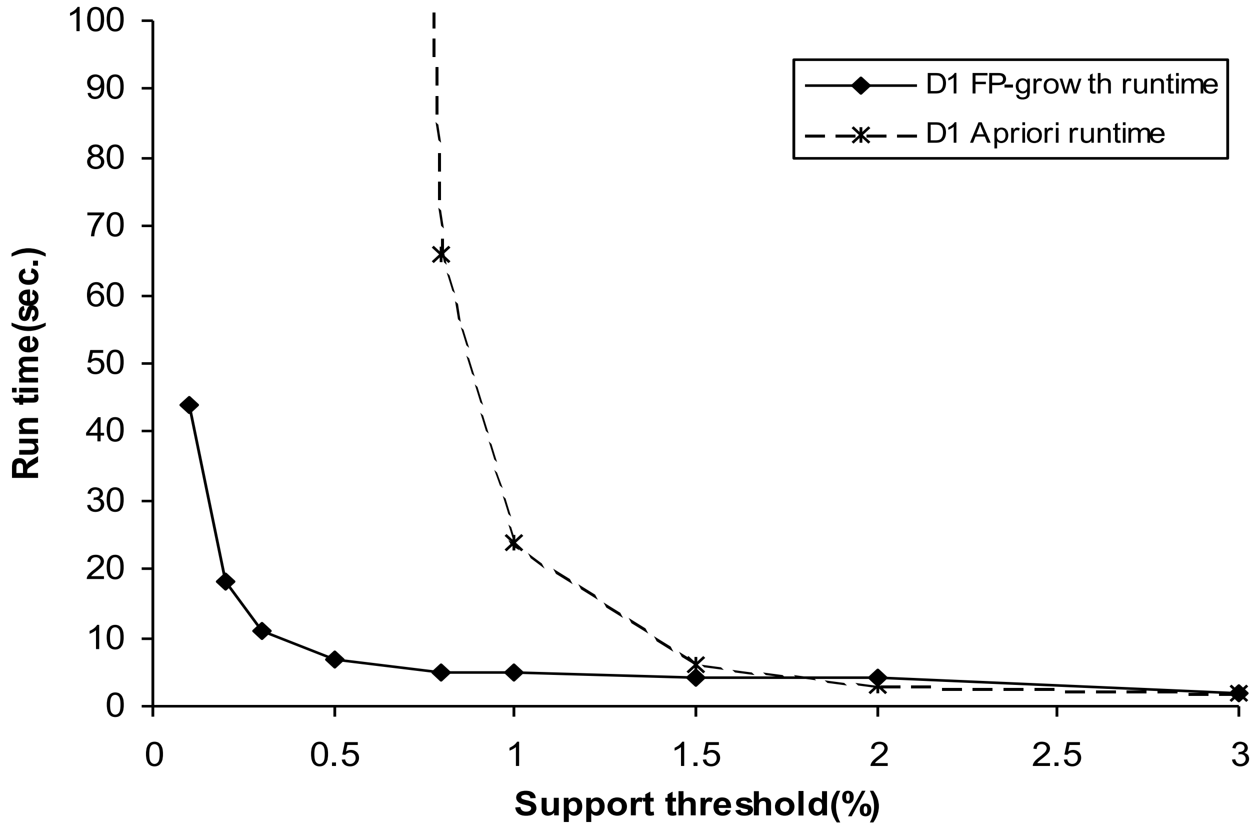
- May not fit in memory
  - Distributed version!
- Time wasted to build the FP-Tree if need to change the threshold

Apriori vs FP-Growth (1)

File	Apriori	FP-Growth
Simple Market Basket test file	3.66 s	3.03 s
"Real" test file (1 Mb)	8.87 s	3.25 s
"Real" test file (20 Mb)	34 m	5.07 s
Whole "real" test file (86 Mb)	4+ hours (Never finished, crashed)	8.82 s

https://www.singularities.com/blog/2015/08/apriori-vs-fpgrowth-for-frequent-item-set-mining

Apriori vs FP-Growth (2)



Jiawei Han, Micheline Kamber and Jian Pei (Chapter 6)

Compacting Output

- As you have seen, many, often redundant, itemsets can be generated
- As an optional post-processing step, we can output only "representative" itemsets
  - Maximal: no immediate superset is frequent
    - Smallest set from which all other frequent itemsets can be derived
  - Closed: no immediate superset has same count (assuming c>0)
    - Can re-compute not only non-closed frequent itemsets, but also support (see TSK:6.4.2)

Maximal Frequent Itemsets

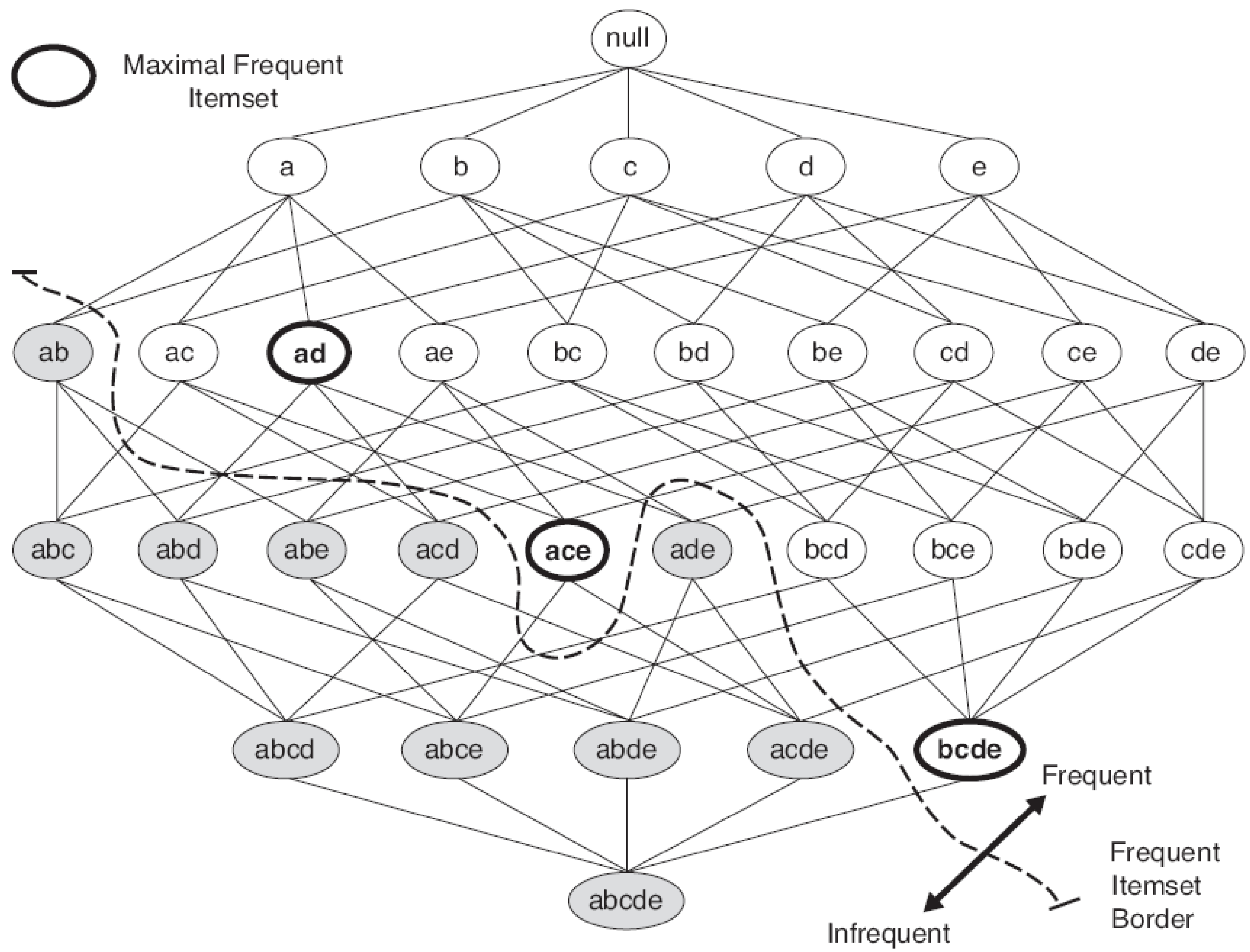


Figure 6.16. Maximal frequent itemset.

Closed Frequent Itemsets

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

minsup = 40%

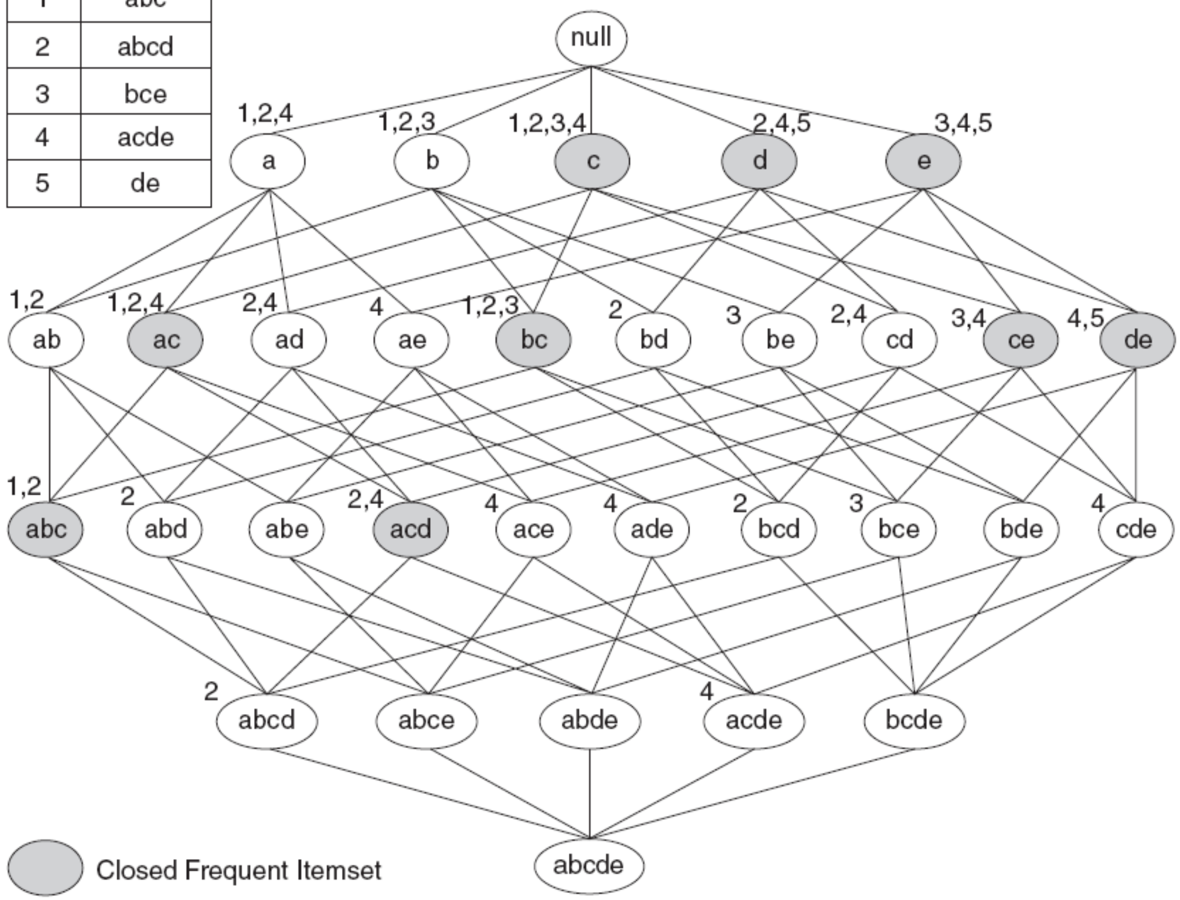


Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

Relationship

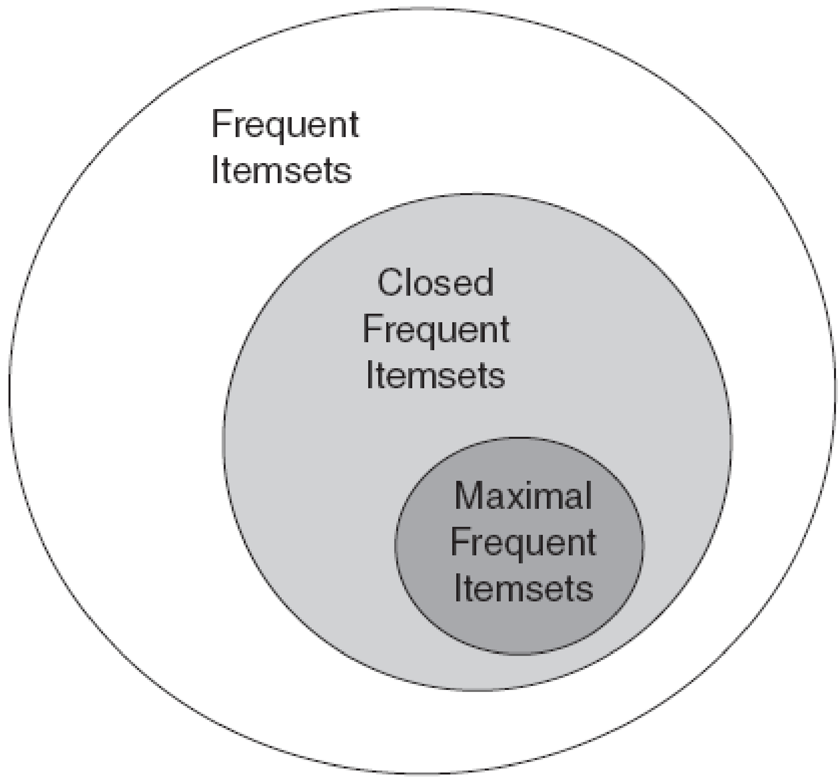


Figure 6.18. Relationships among frequent, maximal frequent, and closed frequent itemsets.

Example

TID	Itemsets
1	{b, c, d}
2	{a, d, e}
3	{b, d}
4	{a, c}
5	{b, c, d}
6	{a, b, c, d}
7	{a, b, c}
8	{b, c}

$s_{min}=3$

Frequent Itemsets

- a(4), b(6), c(6), d(5)
- ac(3), bc(5), bd(4), cd (3)
- bcd(3)

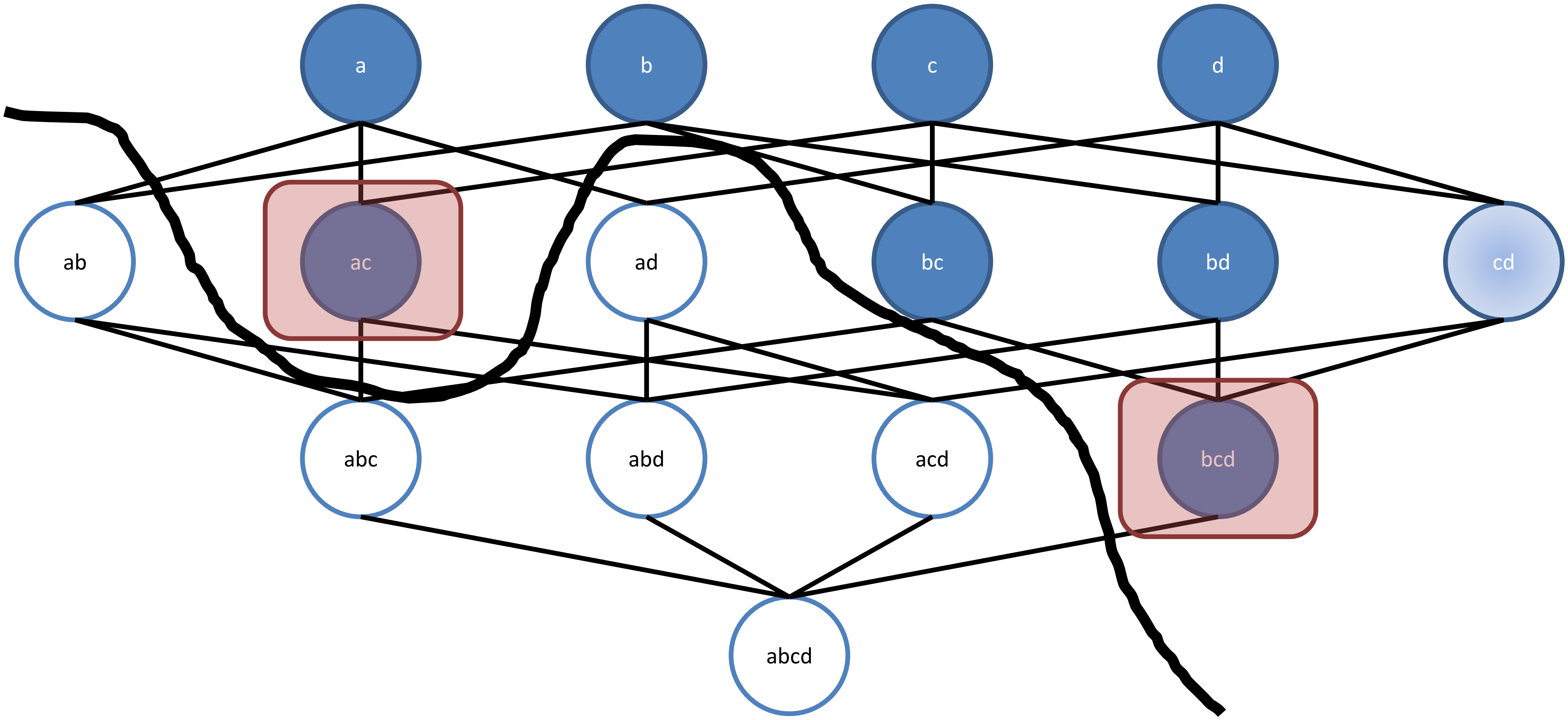
Closed

- a, b, c, d
- ac, bc, bd
- bcd

Maximal

- ac
- bcd

Itemset Lattice for $\mathcal{I}$ = {a, b, c, d}



Summary (1)

- Association analysis attempts to find "interesting" rules associating co-occurring sets of items
- The typical approach is to find frequent itemsets, then interesting rules
- There are many metrics, each with tradeoffs in content and efficiency
  - Confidence leads to rule-generation pruning
  - Support leads to Apriori principle

Summary (2)

- Apriori exploits non-monotonicity in support to prune itemset search, but must scan the dataset for each iteration
- FP-Growth improves upon Apriori by building a data structure (FP-Tree) with just 2 searches across the dataset
- Maximal and Closed itemsets more compactly represent frequent itemsets