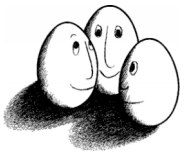
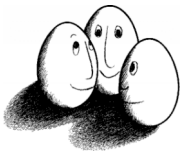


Häufige Mengen ohne Kandidatengenerierung

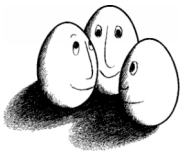
Jiawei Han, Micheline Kamber 2006 (2nd ed.)

- Ziel 1: Kompression der Datenbank in eine Frequent-Pattern Tree Struktur (FP-Tree)
  - Stark komprimiert, vollständig bzgl. des Findens häufiger Mengen
  - Vermeidung von aufwändigen Datenbankdurchläufen
- Ziel 2: Entwicklung einer effizienten Methode zur Suche nach häufigen Mengen auf Basis von FP-Trees
  - Divide-and-conquer: Zerlegung der Aufgaben in kleinere Teilaufgaben
  - Ohne Kandidatengenerierung



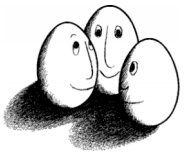
FP-Tree: Transaktionen

- Ein FP-Tree fasst Transaktionen als Wörter auf und stellt gemeinsame Präfixe verschiedener Wörter dar.
- Für jede Transaktion wird ein Pfad im FP-Tree angelegt:
  - Pfade mit gemeinsamem Präfix
    → Häufigkeit aller Knoten auf dem Weg + 1 und Suffix darunter hängen
  - Kein gemeinsamer Präfix vorhanden
    → neuen Zweig anlegen



FP-Tree: Items

- Einfügen der Items in den Transaktionen nach Häufigkeiten sortiert,
  nicht-häufige Items werden nicht eingetragen
- Parallel anlegen: Header Table verweist auf das Vorkommen der items im Baum. Auch die Tabelle ist nach Häufigkeit geordnet.
  - Zu jedem Eintrag wird eine Liste von zugehörigen Knoten im Baum gespeichert

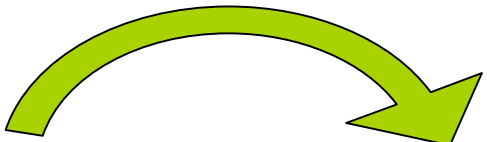


Konstruktion eines FP-Trees

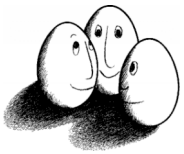
Schritte:

1. Finde häufige 1-Mengen
2. Ordnung nach absteigender Häufigkeit
3. Konstruiere FP-Tree und Header Table während eines 2. Datenbankdurchlaufs

Schritt 1 und 2:
Bestimmung der häufigen 1-Mengen für $s_{min} = 0.5$ und Ordnung nach Häufigkeit



TID	*Items*	*(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}	{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}

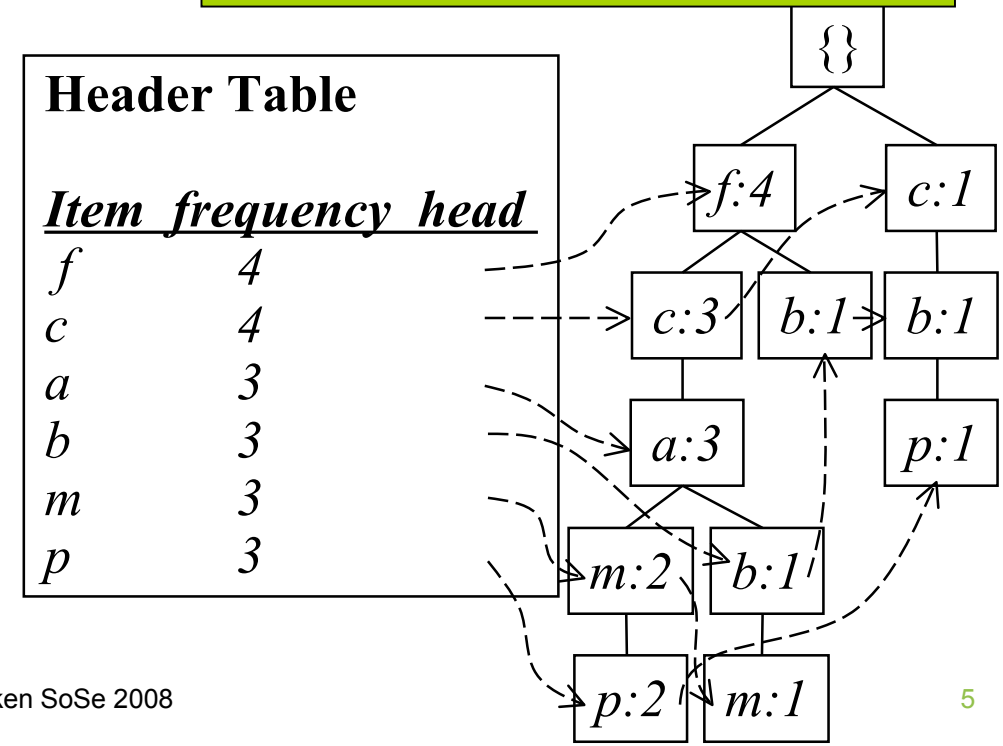


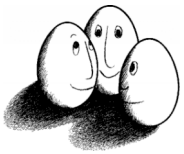
Konstruktion eines FP-Trees

Schritte:

1. Finde häufige 1-Mengen
2. Ordnung nach absteigender Häufigkeit
3. Konstruiere FP-Tree und Header Table während eines 2. Datenbankdurchlaufs

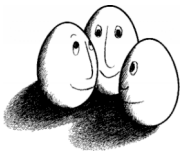
Schritt 3: Konstruktion des FP-Trees und der Header Table





Vorteile von FP-Trees

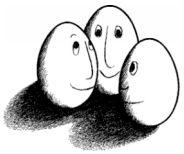
- Vollständigkeit:
  - Es wird niemals ein langes Pattern einer Transaktion geteilt
  - Die notwendigen Informationen zum Finden der Frequent Itemsets bleiben vollständig erhalten
- Kompaktheit:
  - Reduktion irrelevanter Informationen – nicht-häufige Items werden nicht gespeichert.
  - Durch die Ordnung nach absteigender Häufigkeit werden häufigere Items wahrscheinlicher mehrfach verwendet.
  - Speicheraufwand ist niemals größer als für die Originaldatenbank. Beispiel: Für Connect-4 DB ist die Kompressionsrate größer als 100!



Finden häufiger Mengen mit FP-Trees

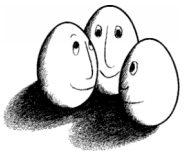
- Grundidee (divide-and-conquer)
  - Rekursives Anwachsen häufiger Mengen unter Ausnutzung des FP-Trees

- Methode
  - Konstruiere die Conditional Pattern Base (Schritt 1) für jedes Item und berechne auf dieser dann den Conditional FP-Tree (Schritt 2)
  - Wiederhole diesen Prozess für jeden neuen Conditional FP-Tree…
  - …bis der resultierende FP-Tree leer ist.



Schritt 1: FP-Tree zu Conditional Pattern Base

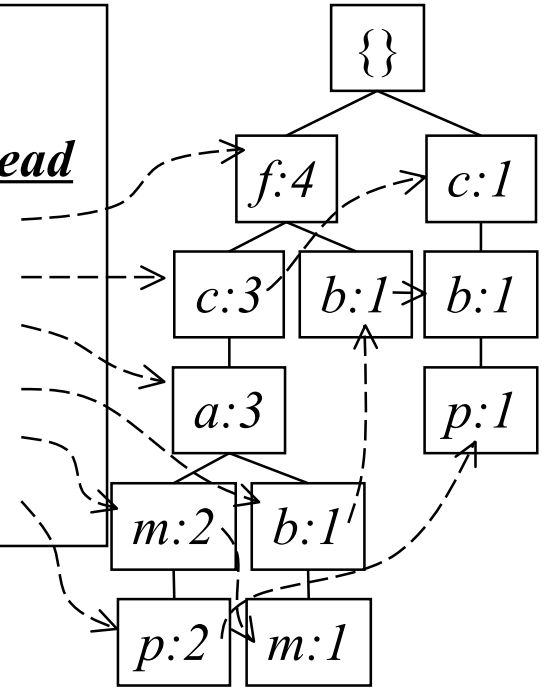
- Die Header Tabelle durchgehen. Die Verweise führen zu den Pfaden, in denen das Item vorkommt.
  - Das Item wird als Suffix betrachtet und alle Präfixe davon als Bedingungen für dieses Suffix. Die Präfixe werden in die Conditional Pattern Base eingetragen.
  - Die Häufigkeiten der Präfixe werden im Knoten selbst abgelesen und ebenfalls eingetragen.



Schritt 1: Conditional Pattern Base

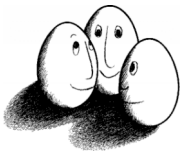
Header Table

Item	*frequency*	*head*
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

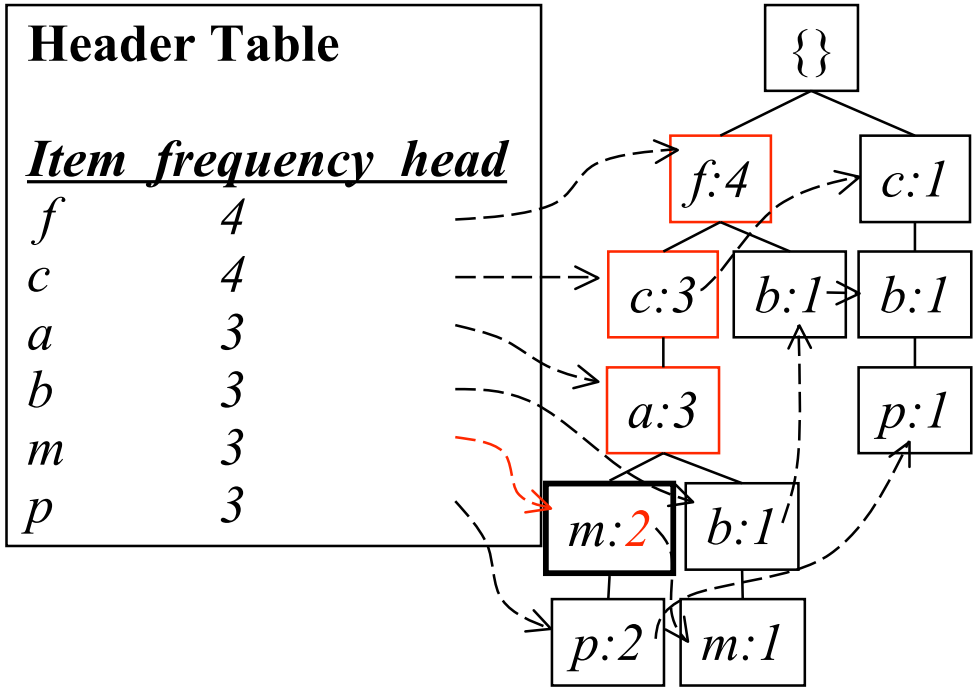


***Conditional* pattern bases**

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*

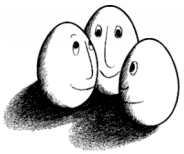


Schritt 1: Conditional Pattern Base

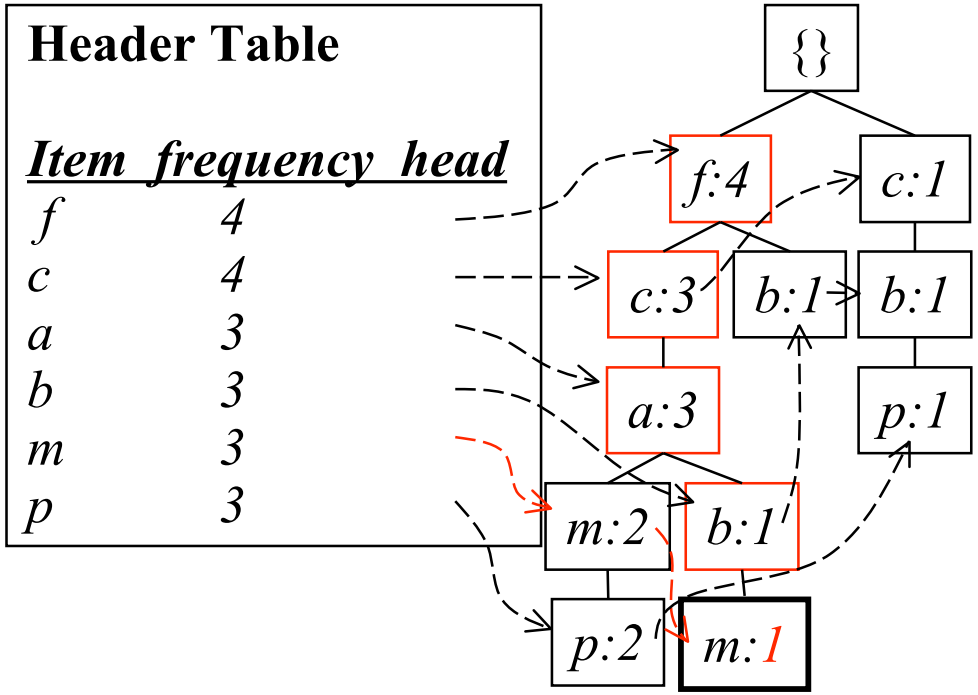


***Conditional* pattern bases**

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*

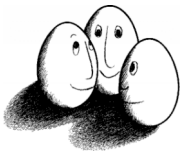


Schritt 1: Conditional Pattern Base



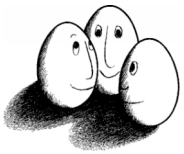
***Conditional* pattern bases**

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*



Schritt 2: Conditional Pattern Base zu Conditional FP Tree

- Präfixpfade eines Suffixes bilden die bedingte Basis, d.h. sie werden im Prinzip wie Transaktionen einer ursprünglichen Datenbank behandelt.
- Diejenigen Präfixpfade, die häufiger als smin sind, bilden den bedingten FP-Tree (wie zuvor).
- Es kann mehrere Pfade im bedingten Baum geben!

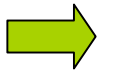


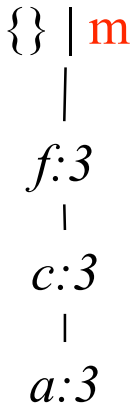
Schritt 2: Conditional FP-Tree

- Benutze jeden Eintrag in der Conditional Pattern Base als “Transaktion” einer Datenbank (hier: 2 Transaktionen mit fca und eine Transaktion mit fcab)

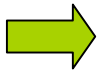
- Generiere hieraus den zugehörigen Conditional FP-Tree

m-conditional pattern base:
fca:2, fcab:1



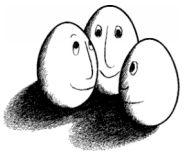


m-conditional FP-tree



Rekursive Aufrufe liefern alle häufigen Mengen mit m

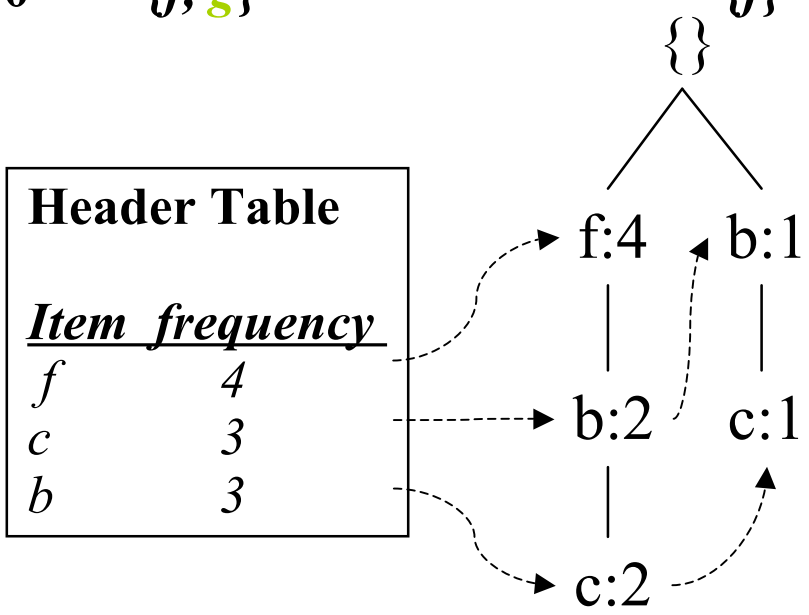
m,
fm, cm, am,
fcm, fam, cam,
fcam



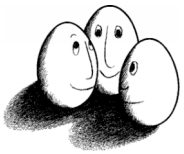
Vollständiges Beispiel

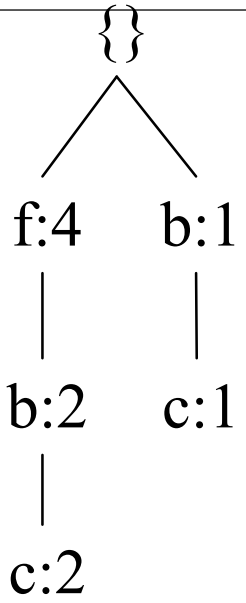
TID	*Items*	*(ordered) frequent items*
100	{*b, c, f*}	{*f, b, c*}
200	{*a, b, c*}	{*b, c*}
300	{*d, f*}	{*f*}
400	{*b, c, e, f*}	{*f, b, c*}
500	{*f, g*}	{*f*}

Entferne alle nicht-häufigen Mengen (s_{min} = 0,4) und ordne Transaktionen nach Häufigkeit.



Initialisiere Header Table und baue den ersten FP-Tree auf.



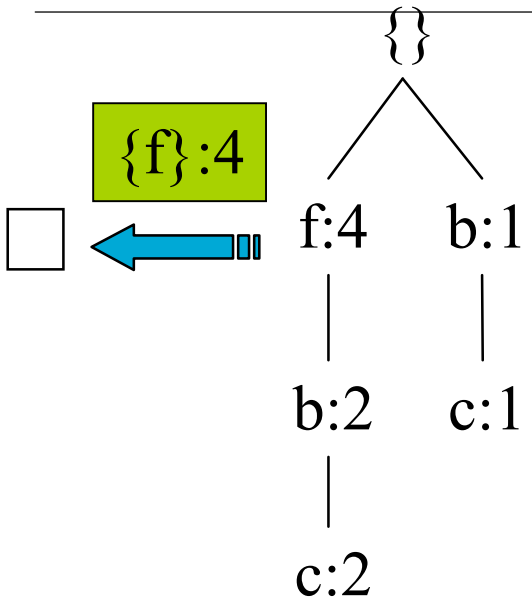


Header Table 1

Item	*frequency*
f	*4*
c	*3*
b	*3*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

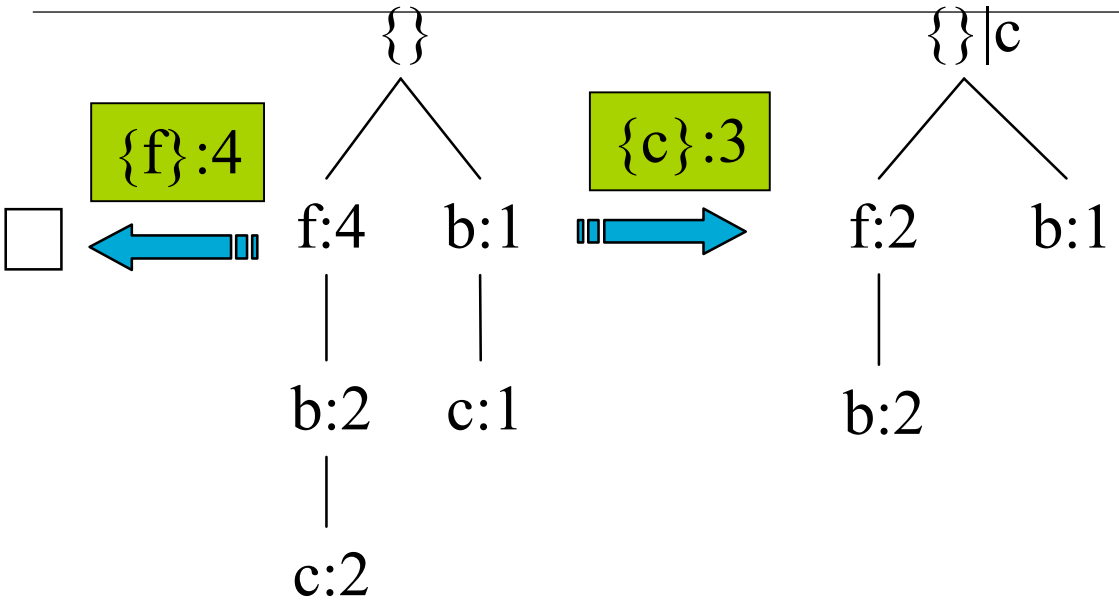


Header Table 1

Item	*frequency*
f	*4*
c	*3*
b	*3*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

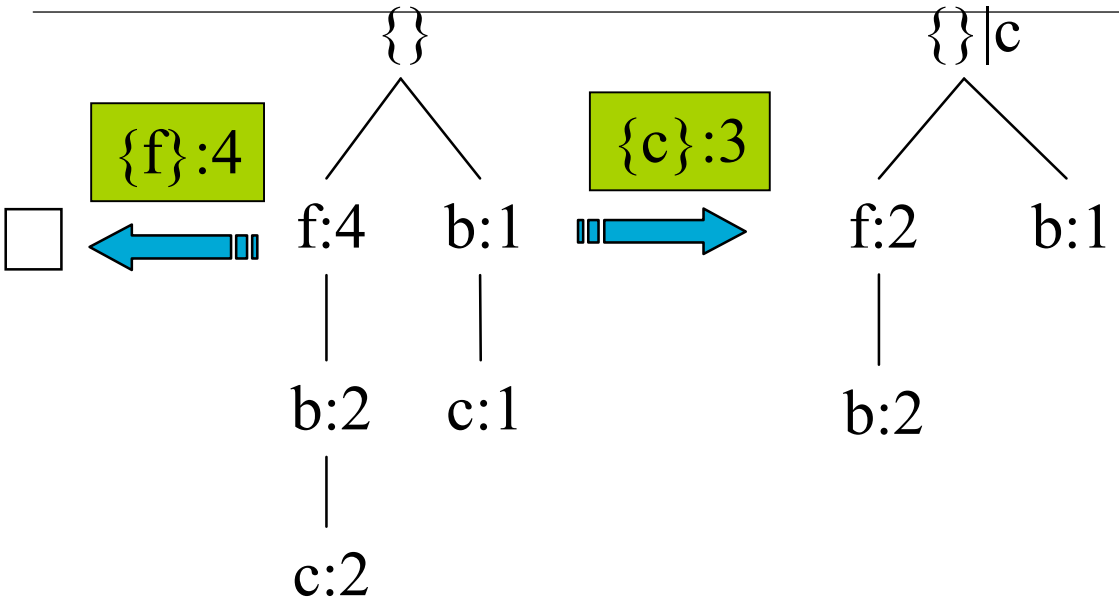


Header Table 1

Item	*frequency*
f	*4*
c	*3*
b	*3*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

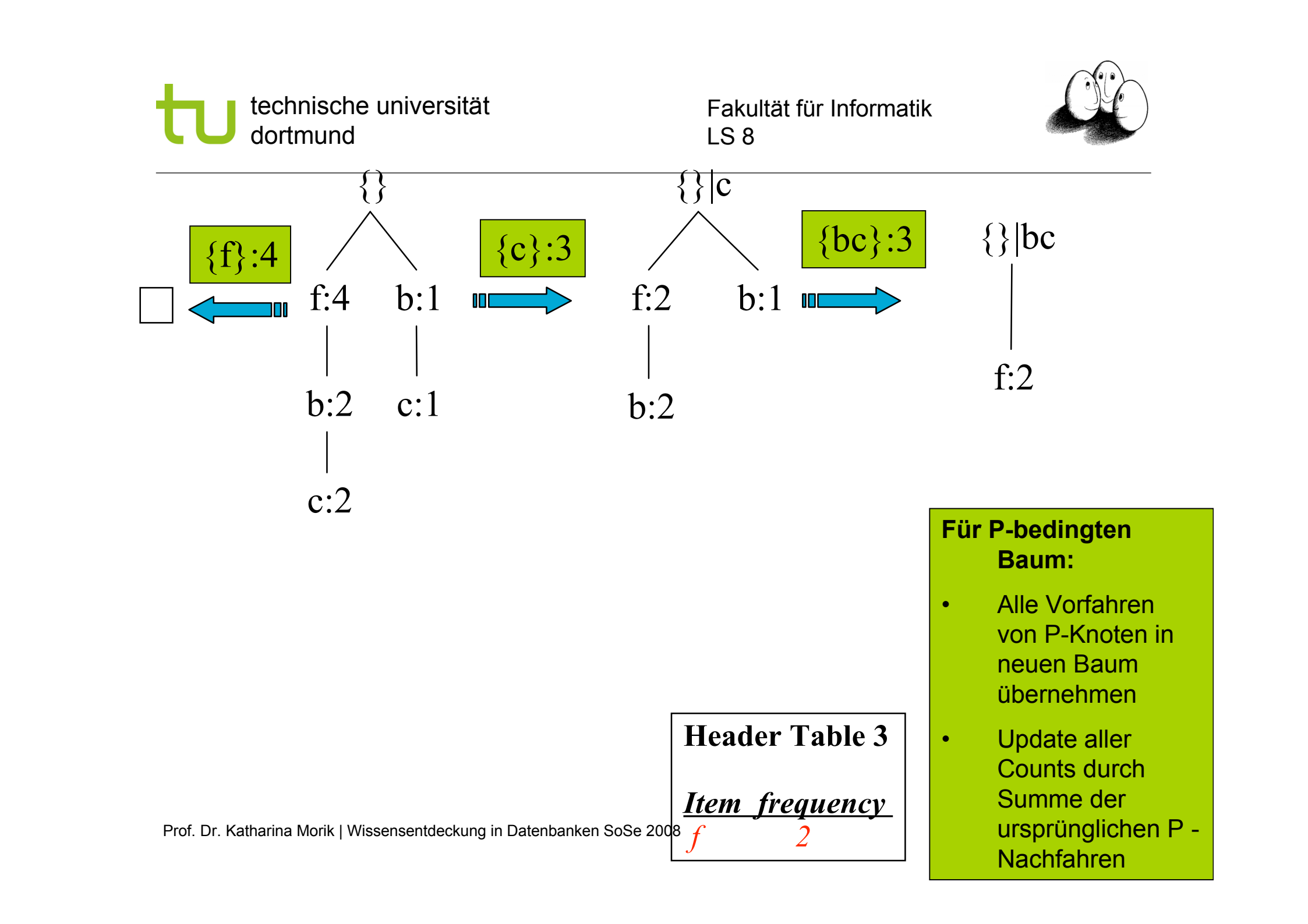


Header Table 2

Item	*frequency*
b	*3*
f	*2*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

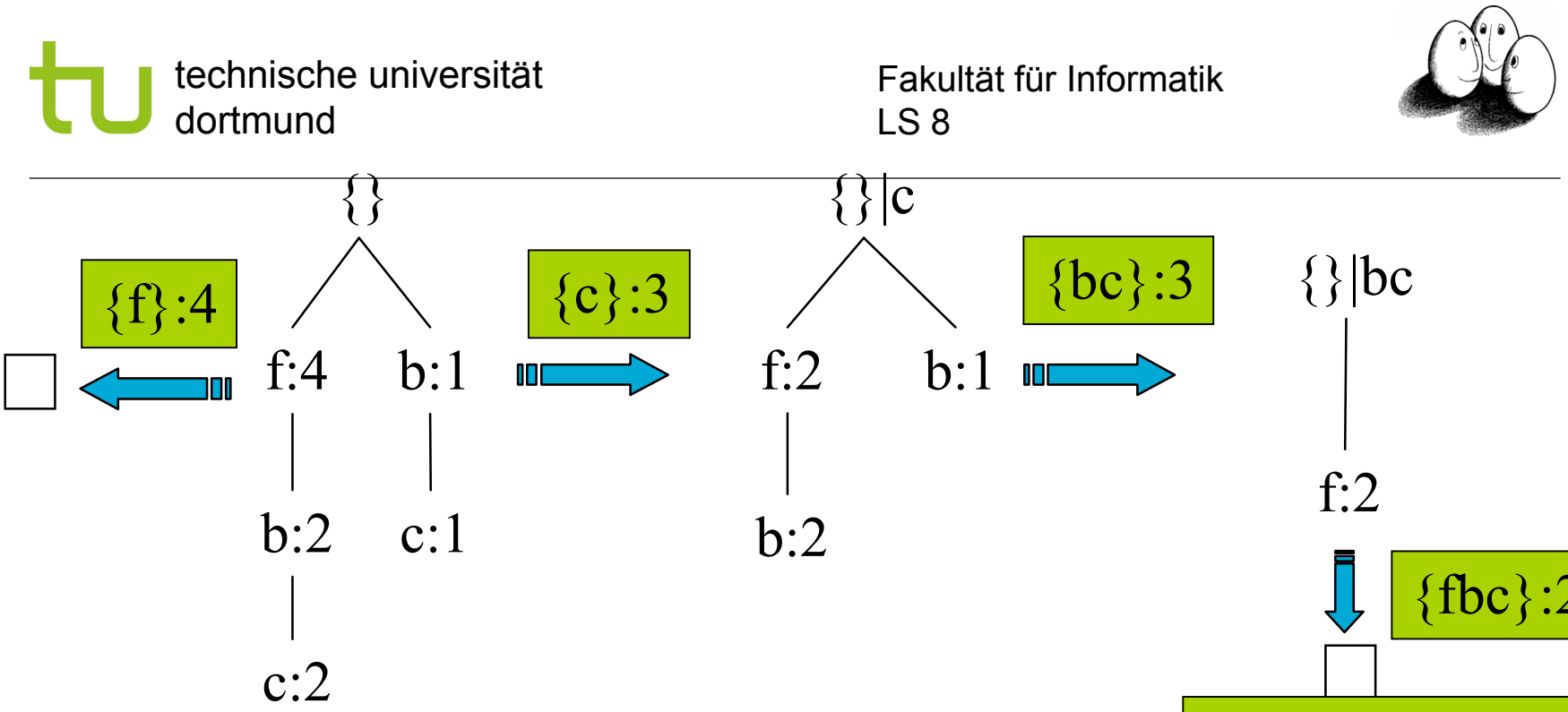


Header Table 3

Item	*frequency*
f	*2*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

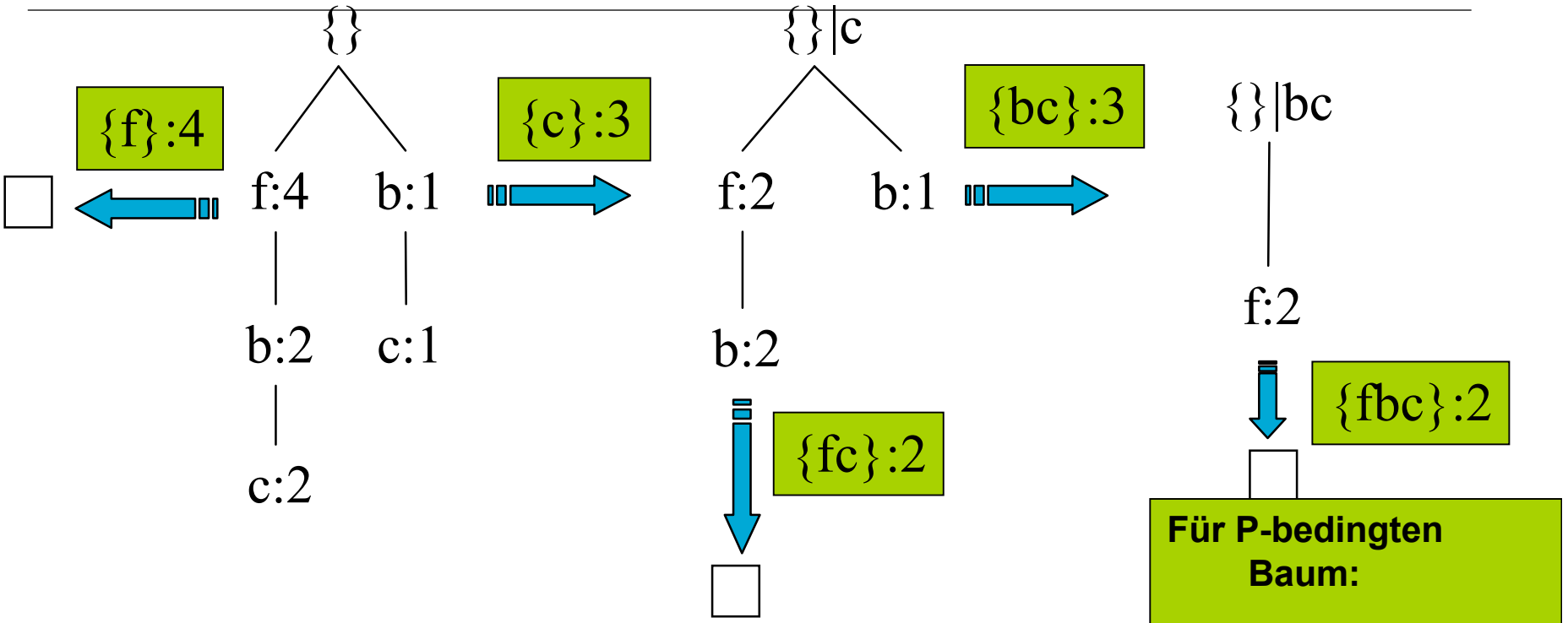


Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

Header Table 3

Item	*frequency*
f	2

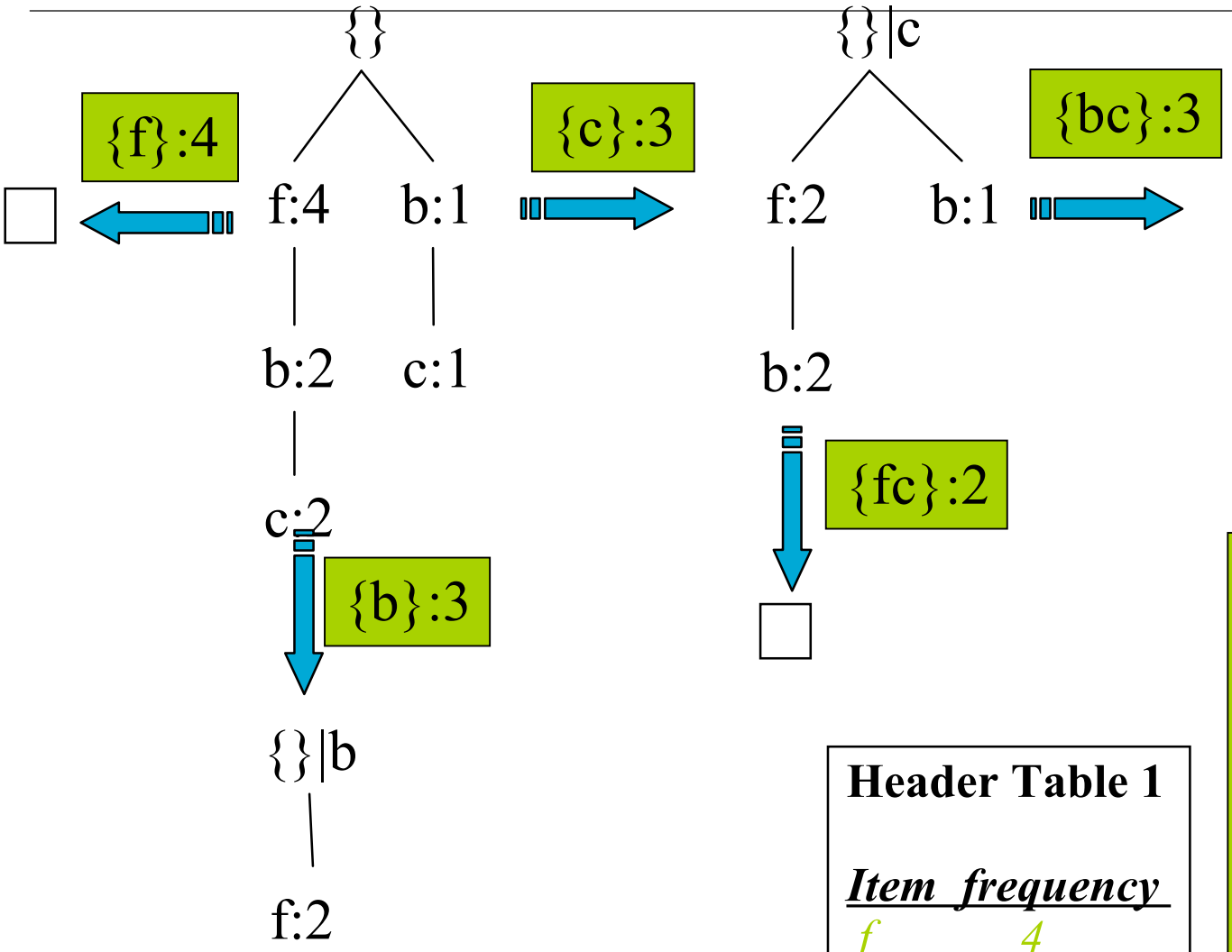


Header Table 2

Item	*frequency*
b	*3*
f	*2*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren

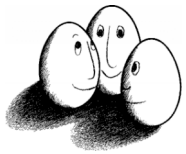


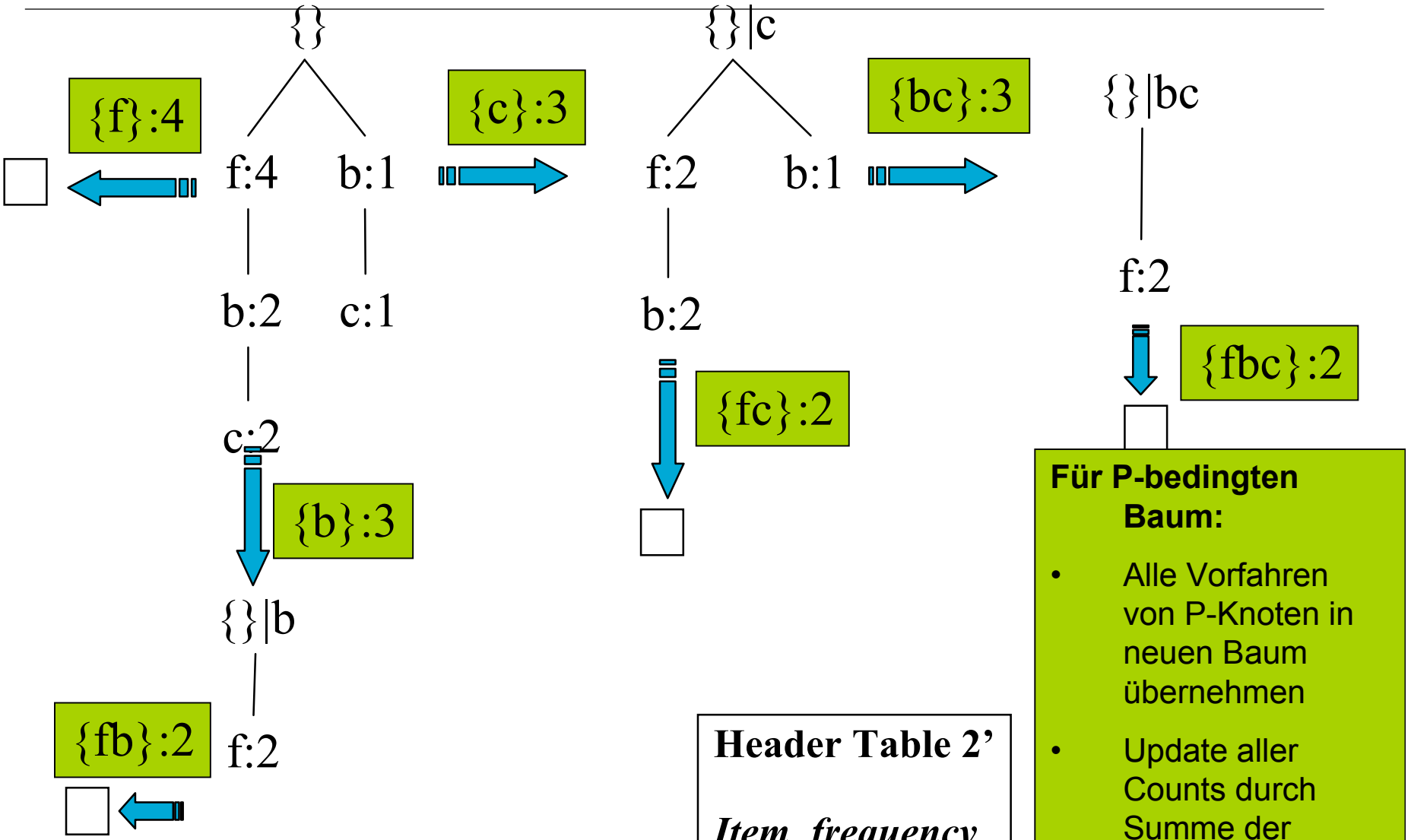
Header Table 1

Item	*frequency*
f	*4*
c	*3*
b	*3*

Für P-bedingten Baum:

- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren



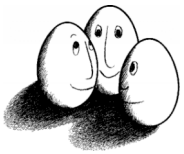


Header Table 2'

Item	*frequency*
f	2

Für P-bedingten Baum:

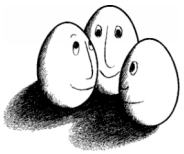
- Alle Vorfahren von P-Knoten in neuen Baum übernehmen
- Update aller Counts durch Summe der ursprünglichen P - Nachfahren



Grundlagen Frequent Pattern Growth

- Pattern Growth Property
  - Sei α ein häufiges Item Set und B sei die Conditional Pattern Base von α. Sei weiter β ein Item Set in B. $\alpha \cup \beta$ ist ein häufiges Item Set gdw. β häufig ist in B.

- “abcdef ” is ein häufiges Item Set gdw.
  - “abcde ” häufig ist und
  - “f ” häufig ist in der Menge von Transactionen, die “abcde ” enthalten

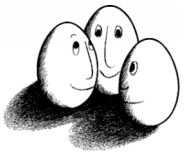


Algorithmus FP-growth

- Input: D eine Transaktionsdatenbank, smin ein Schwellwert der Häufigkeit

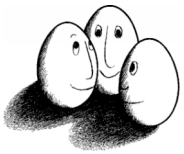
- Scan von D, Erstellen der Menge F häufiger items und ihrer Häufigkeiten, Ordnen von F in absteigender Häufigkeit.
- Wurzel des FP Trees ist Null. Für jede Transaktion Trans in D:
  - nach Häufigkeit gemäß F geordnete items in Trans werden zur Liste [p|P], wobei p das erste item und P die restlichen sind. insert_tree([p|P],T)

  - FP-growth(FP_tree, null)



Routine: insert_tree([p|P],T)

- Wenn T ein Kind N hat mit N.item_name = p.item_name dann erhöhe Häufigkeit von N +1.
- Sonst bilde neuen Knoten N mit Häufigkeit = 1 direkt unter T und füge Knotenverweise zu den Knoten i mit dem selben i.item_name ein.
- Solange P nicht {} ist, insert_tree(P,N).

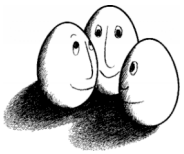


FP-growth(Tree, a)

- Wenn Tree ein einziger Pfad P ist,
  - dann generiere für jede Kombination β von Knoten in P Muster $\beta \cup \alpha$ mit support = min { support eines items in β }.

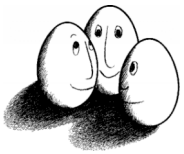
- Sonst für jedes a_i in header von Tree
  - generiere Muster $\beta = a_i \cup \alpha$ mit support = a_i.support;
  - konstruiere Conditional Pattern Base von β und daraus den Conditional FP-Tree von β : Treeβ
  - Wenn Treeβ noch nicht {}, dann FP-growth(Treeβ, β)

  - EINFACH rekursiv bis leer

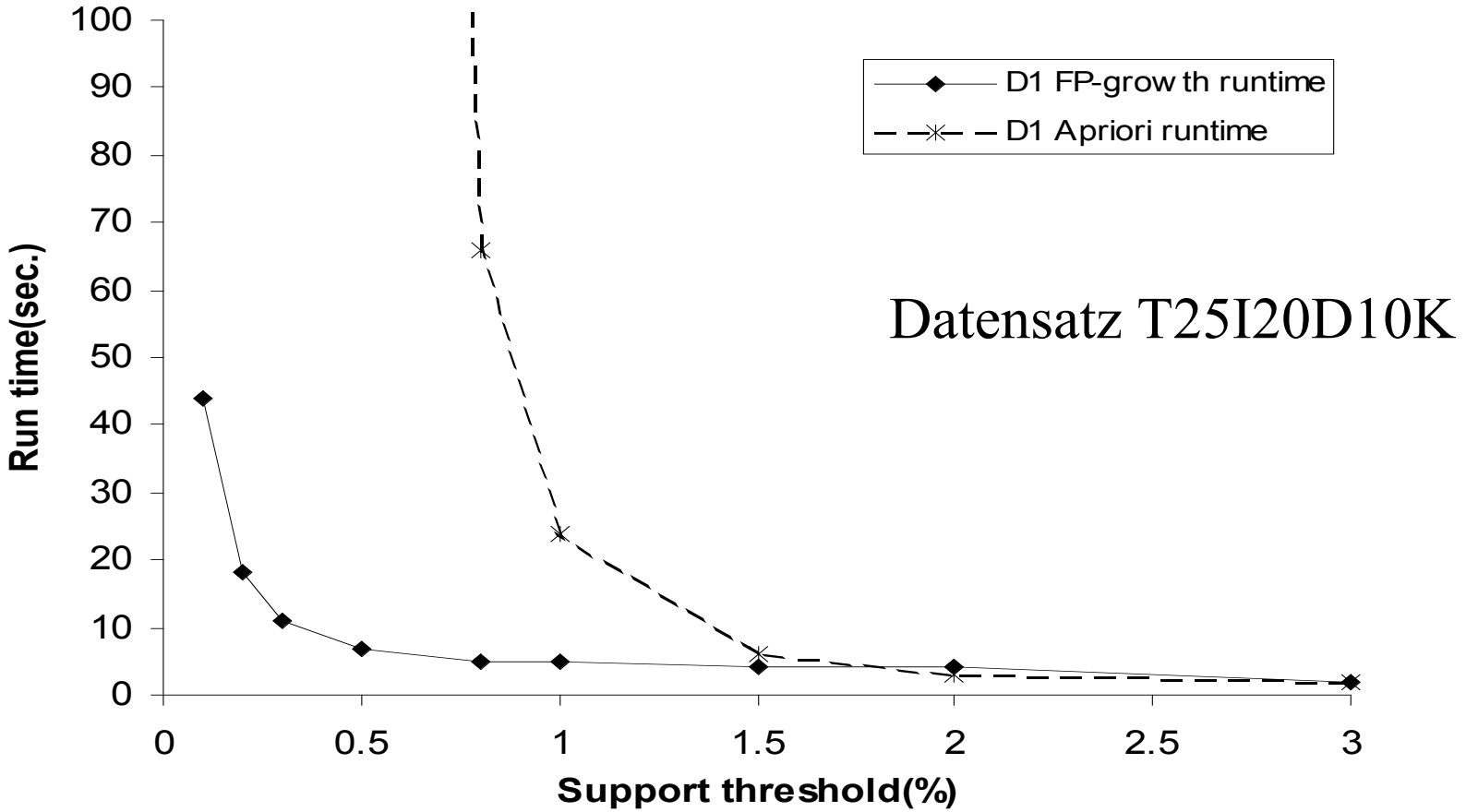


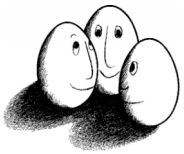
Wie schnell ist FP-growth?

- Empirische Performanzanalysen zeigten, dass FP-growth etwa eine Größenordnung schneller ist als Apriori
- Begründung
  - Keine Kandidatengenerierung, keine Kandidatentests
  - Wesentlich kompaktere Datenstruktur
  - Keine wiederholten Datenbankscans
  - Basisoperationen sind Zählen und Aufbau der FP -Trees



FP-growth vs. Apriori: Einfluss von s_{min}





Was wissen wir jetzt?

- FP-growth als Alternative zu Apriori
  - Schneller, weil keine Kandidaten generiert werden
  - Kompaktes Speichern
  - Basisoperation ist einfach Zählen
- Der FP-Tree gibt Präfixbäume für ein Suffix an
- Die Ordnungsrelation ist die Häufigkeit der Items
  - Der Baum wird vom häufigsten zum seltensten gebaut