


Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z3003 · DATA MINING: SCALING TO LARGE DATA

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

*Lecture 14 · Thursday 15 October 2026*

**Dr. Innocent Nyalala** | BS Semester 5 (DSAI) | IIT Madras Zanzibar

# Session Outline

IIT Madras Zanzibar

1. (0–5 min) Recap: Apriori's join → prune → count
2. (5–12 min) What makes Apriori slow: passes and candidates
3. (12–35 min) FP-Growth: build the FP-tree by hand, then mine it
4. (35–50 min) PCY: hashing pairs to shrink the hardest level
5. (50–60 min) Random sampling and the SON algorithm
6. (60–75 min) Exercise, summary, questions

# Learning Outcomes

IIT Madras Zanzibar

*By the end of this session, you will be able to:*

| | |
|---|---|
| **Explain** | why passes over the data, not arithmetic, dominate cost on large data |
| **Build** | an FP-tree by hand from a small basket list |
| **Mine** | frequent itemsets from an FP-tree using conditional pattern bases |
| **Apply** | PCY's hash buckets to cut candidate pairs |
| **Describe** | random sampling and SON, and what each guarantees |

# What Do the Names Mean?

IIT Madras Zanzibar

| Name | Stands for | One-line idea | Problem it fixes |
|---|---|---|---|
| FP-Growth | Frequent Pattern Growth | squeeze baskets into a tree, read answers off the tree | too many passes, too many candidates |
| PCY | Park, Chen, Yu (1995), the inventors | drop pairs into boxes; skip pairs in small boxes | too many pair counters |
| SON | Savasere, Omiecinski, Navathe (1995) | split data into piles; find winners per pile; check once | data bigger than memory |

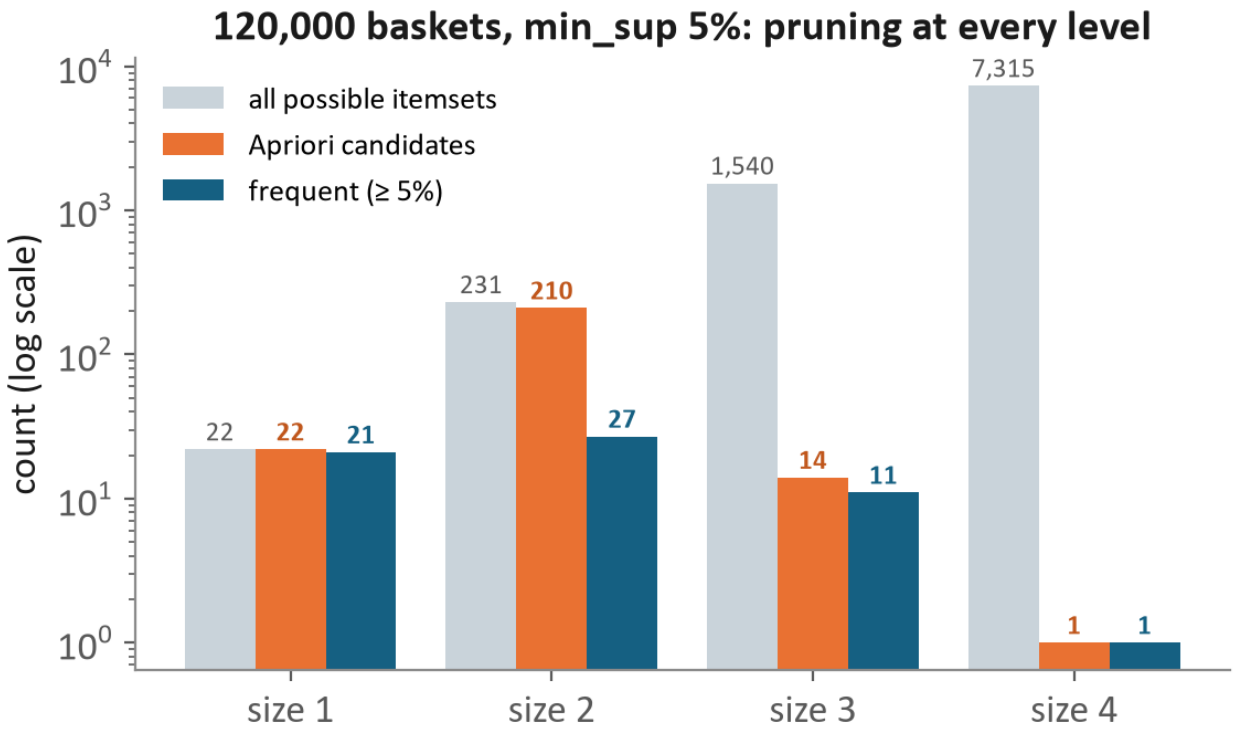
**frequent pattern = a group of items that is often bought together**

# Key Words

IIT Madras Zanzibar

| Word | What it means | Tiny example |
|---|---|---|
| basket | one shopping trip: the items bought together | T1 = {H, R, U} |
| support | how many baskets contain the itemset | {U, R} is in 4 baskets |
| min_sup | the cut-off we choose | min_sup = 2 |
| frequent | support is at least min_sup | {U, R}: 4 ≥ 2, so frequent |
| pass | reading the whole data once, start to end | Apriori: one pass per level |
| candidate | an itemset we must count to check | Apriori level 2: 210 pairs |
| FP-tree | the baskets squeezed into one tree | shared starts stored once |
| hash bucket | a numbered box that many pairs fall into | pair → box 0, 1 or 2 |
| chunk | a slice of the data small enough for memory | 4 chunks of 30,000 |

# Recap: Apriori




**1** **Strength**
exact; level 3: 14 candidates instead of 1,540

**2** **Weak spot 1**
level 2: 210 candidate pairs counted

**3** **Weak spot 2**
one full pass over the data per level (4 here)

**Today:** attack both weak spots — fewer passes (FP-Growth, SON) and fewer pair candidates (PCY).

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

IIT Madras Zanzibar

→ **Foundations: What Makes Apriori Slow?**

FP-Growth: Build the Tree

FP-Growth: Mine the Tree

Limited-Pass Algorithms: PCY

Random Sampling and SON

In-Class Exercise

# What Really Costs Time: Passes Over the Data

IIT Madras Zanzibar

| Where the data is | Typical time to read 1 GB |
|---|---|
| memory (RAM) | ≈ 0.1 second |
| solid-state disk | ≈ 1–2 seconds |
| spinning disk / network | ≈ 10 seconds or more |

$$\text{cost} \approx (\text{passes over the data}) \times (\text{data size})$$

| Algorithm | passes |
|---|---|
| Apriori | k (longest itemset + 1) |
| FP-Growth | 2 |
| PCY | k, but level 2 much cheaper |
| SON | 2 |

**Second cost — memory:** 10,000 items → 10,000 × 9,999 / 2 ≈ 50 million candidate pairs to keep counters for. Level 2 is where memory runs out.

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

IIT Madras Zanzibar

Foundations: What Makes Apriori Slow?

→ **FP-Growth: Build the Tree**

FP-Growth: Mine the Tree

Limited-Pass Algorithms: PCY

Random Sampling and SON

In-Class Exercise

# FP-Growth: The Idea

IIT Madras Zanzibar

1. **Pass 1**
   count each item; drop infrequent items; sort items by frequency
2. **Pass 2**
   insert every basket (sorted the same way) into a prefix tree
3. **Mine the tree**
   in memory, no more data passes, no candidates

**Analogy: a dictionary's prefixes**

'cat', 'car', 'cart' all share 'ca'.

Baskets that start with the same frequent items share a path in the tree, so the data shrinks.

**Why sort by frequency?** the most common items come first, so many baskets share the top of the tree — maximum sharing, smallest tree.

# Easy Example First: 4 Baskets

IIT Madras Zanzibar

| basket | items (in order Milk, Bread, Eggs) |
|---|---|
| 1 | Milk, Bread |
| 2 | Milk, Bread, Eggs |
| 3 | Milk, Eggs |
| 4 | Bread |

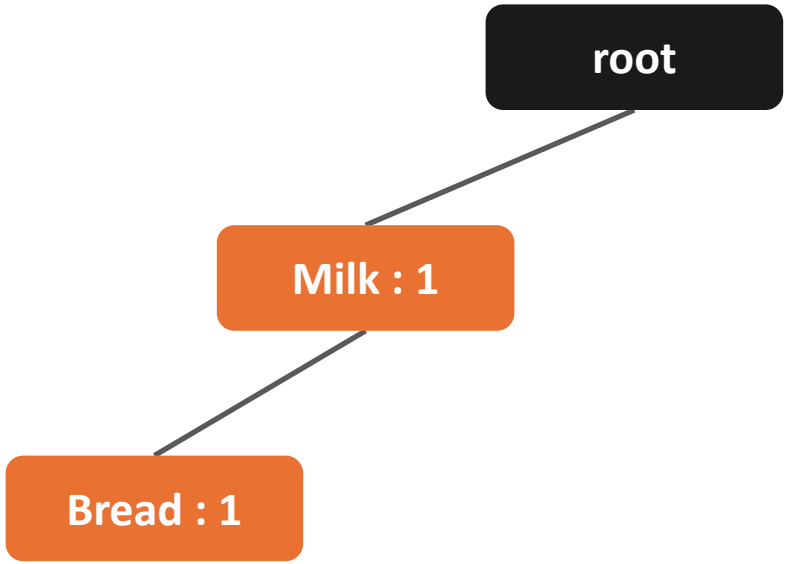
| item | how many baskets | keep? |
|---|---|---|
| Milk | 3 (baskets 1, 2, 3) | yes |
| Bread | 3 (baskets 1, 2, 4) | yes |
| Eggs | 2 (baskets 2, 3) | yes |

**Rule: a group is frequent if it is in at least 2 baskets (min_sup = 2).**
**Pass 1 = count items, then write every basket most-common-first.**

# Build the Tree: Basket 1 of 4

IIT Madras Zanzibar

1. Start with an empty tree: just 'root'.
2. Basket 1 = Milk, Bread.
3. Nothing is there yet, so make Milk : 1, then Bread : 1 under it.

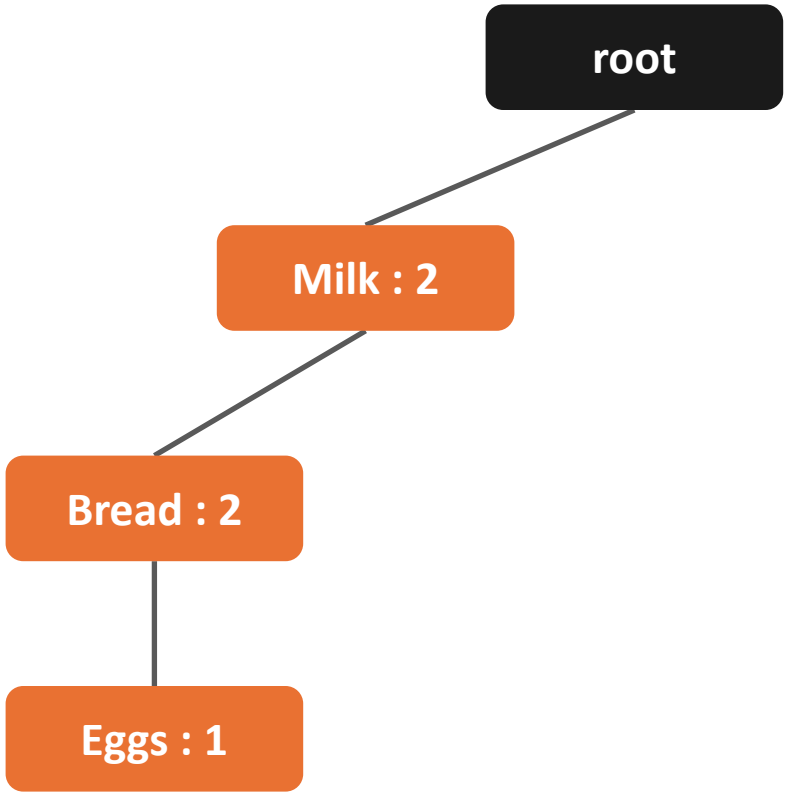



**Rule: walk down from root. Item already there? add 1. Not there? make a new box with 1.**

# Build the Tree: Basket 2 of 4

IIT Madras Zanzibar

1. Basket 2 = Milk, Bread, Eggs.
2. Milk is already there: 1 becomes 2.
3. Bread under Milk is there: 1 becomes 2.
4. Eggs is not there: make Eggs : 1.

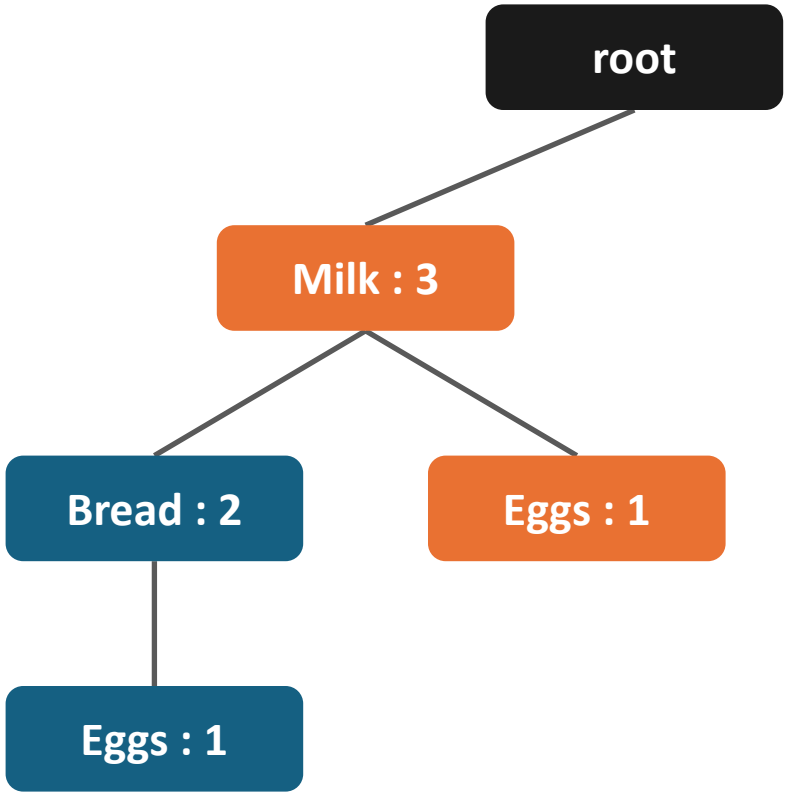



**Rule: walk down from root. Item already there? add 1. Not there? make a new box with 1.**

# Build the Tree: Basket 3 of 4

IIT Madras Zanzibar

1. Basket 3 = Milk, Eggs.
2. Milk is there: 2 becomes 3.
3. Under Milk there is Bread, but no Eggs.
4. So make a NEW Eggs : 1 directly under Milk.

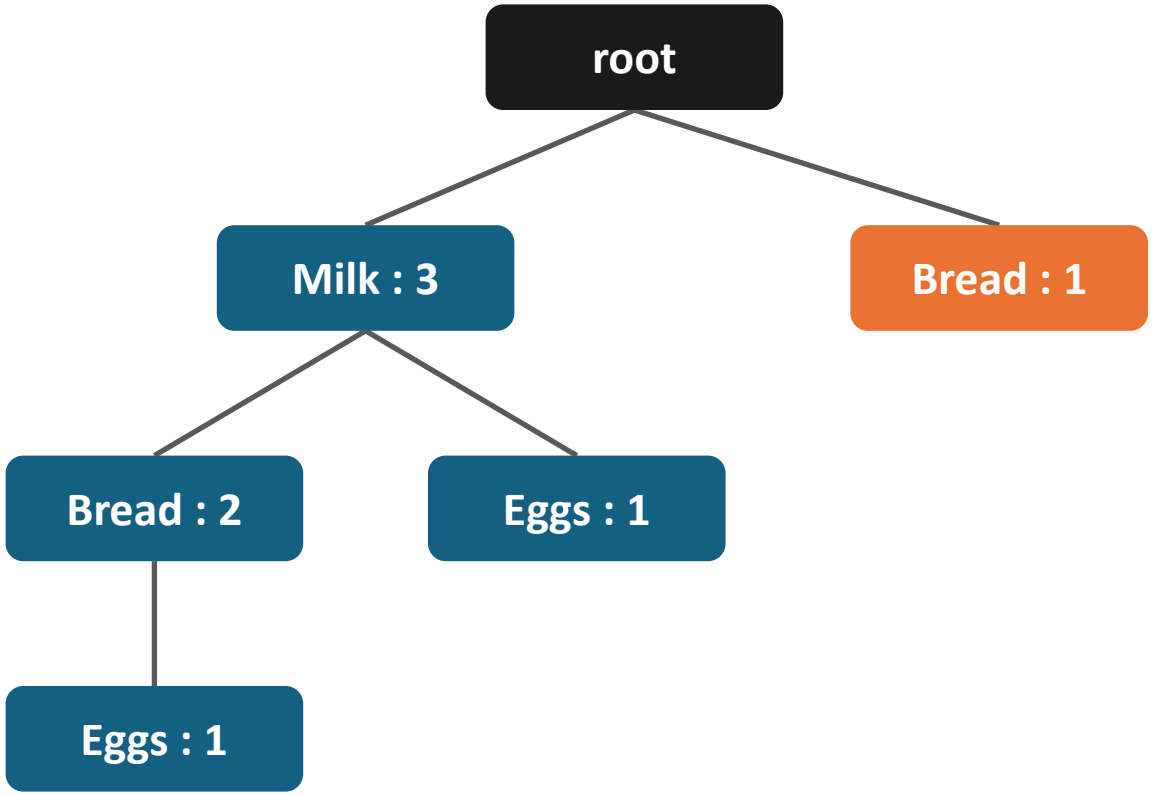



**Rule: walk down from root. Item already there? add 1. Not there? make a new box with 1.**

# Build the Tree: Basket 4 of 4

IIT Madras Zanzibar

1. Basket 4 = Bread.
2. Under root there is only Milk, no Bread.
3. So start a NEW branch: Bread : 1.
4. Check: 3 + 1 = 4 baskets. Done.

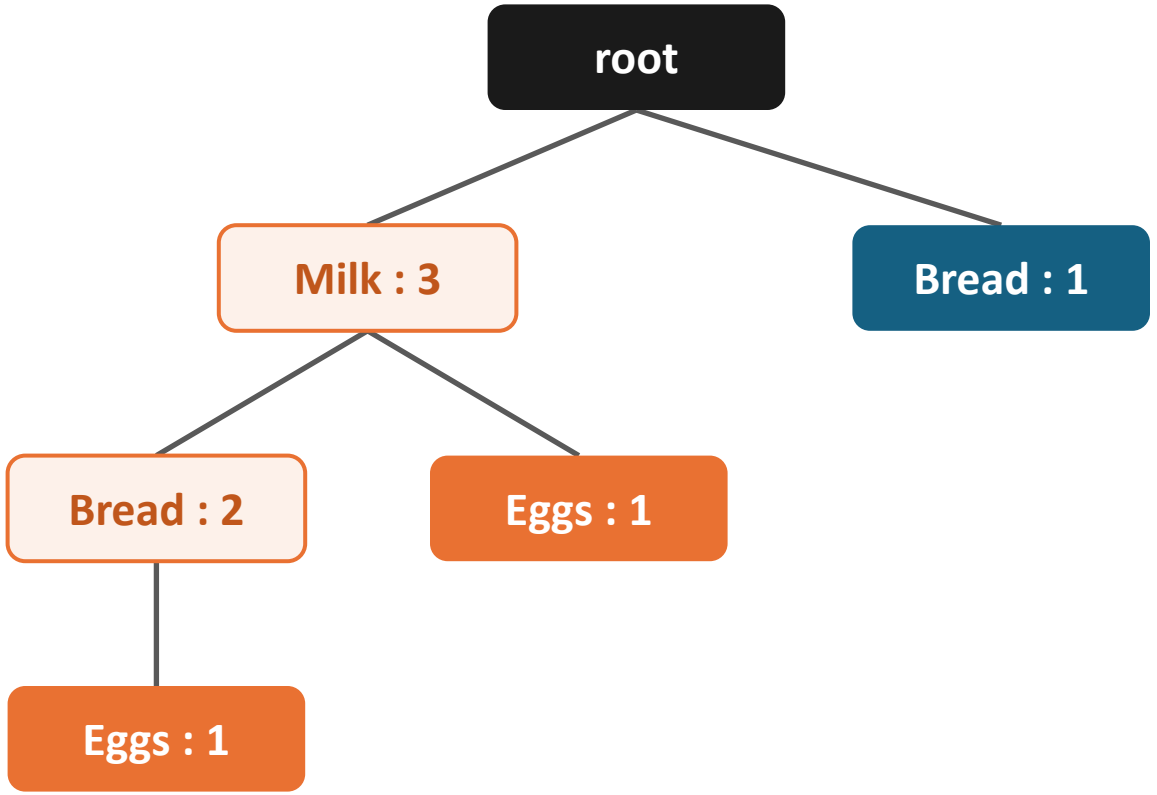



**Rule: walk down from root. Item already there? add 1. Not there? make a new box with 1.**

# Read the Tree: What Was Bought WITH Eggs?

IIT Madras Zanzibar

1. Find every Eggs box (orange).
2. Read the boxes ABOVE each one: Milk, Bread (×1) and Milk (×1).
3. Add up: Milk 1+1 = 2.  Bread = 1.
4. Milk reaches 2, so {Milk, Eggs} is frequent. Bread does not.

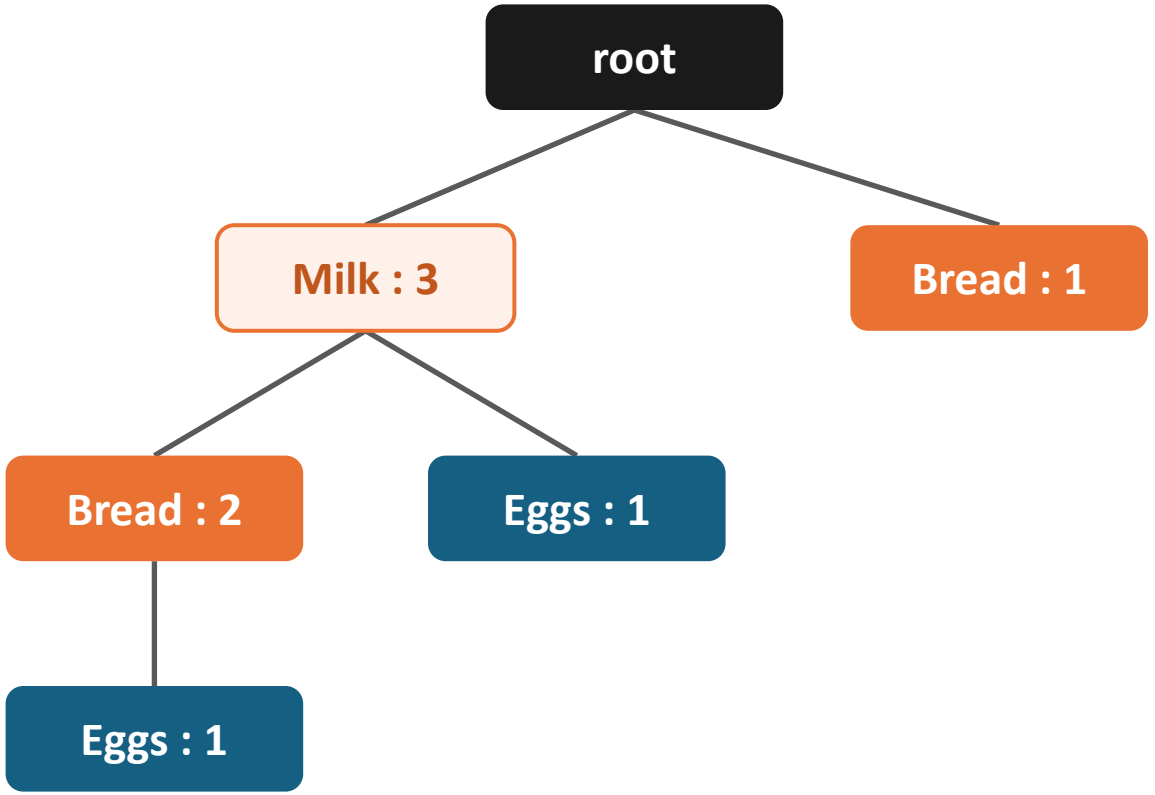



**'Conditional pattern base' = the list of boxes above each Eggs box. That is all.**

# Read the Tree: What Was Bought WITH Bread?

IIT Madras Zanzibar

1. Find every Bread box (orange).
2. Above Bread : 2 is Milk. Above Bread : 1 is nothing.
3. Add up: Milk = 2.
4. Milk reaches 2, so {Milk, Bread} is frequent.
5. Milk is at the top: nothing above, nothing to read.




**Same question for every item, starting from the bottom: what is above it?**

# Easy Example: Final Answer and Check

IIT Madras Zanzibar

| group | from the tree | check in the baskets | frequent? |
|---|---|---|---|
| {Milk, Bread} | 2 | baskets 1, 2 | yes |
| {Milk, Eggs} | 2 | baskets 2, 3 | yes |
| {Bread, Eggs} | 1 | basket 2 only | no |

**Data read only twice (pass 1: count; pass 2: build tree).**
**No list of guesses (candidates) was ever made.**

# Step 1: Count and Sort the Items

IIT Madras Zanzibar

| item | count | order |
|---|---|---|
| U | 7 | 1 |
| R | 6 | 2 |
| S | 6 | 3 |
| H | 2 | 4 |
| P | 2 | 5 |

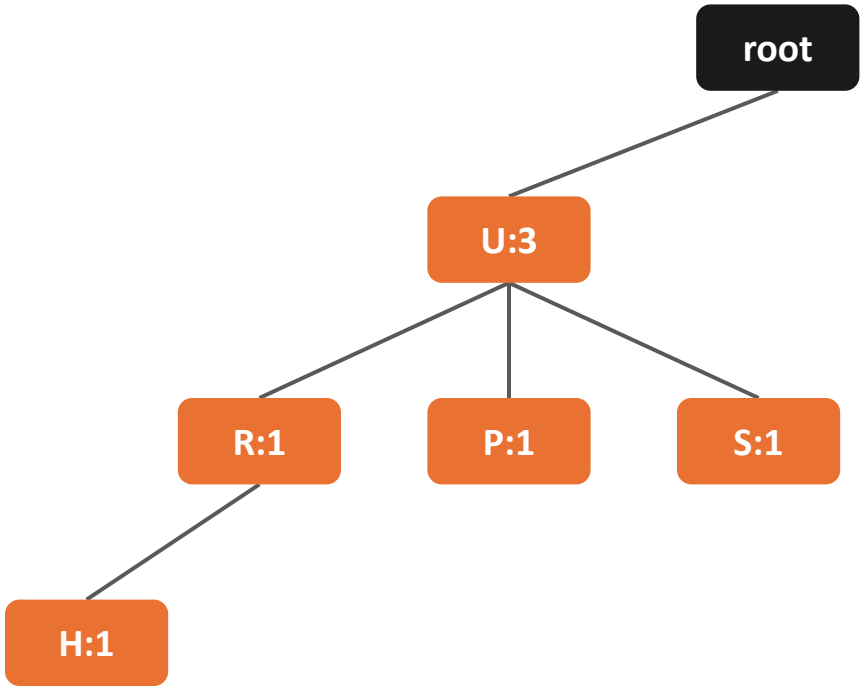
**Rule:** sort every basket by the global item counts, most frequent first.

| basket | original | sorted by frequency |
|---|---|---|
| T1 | H, R, U | U, R, H |
| T2 | P, U | U, P |
| T3 | S, U | U, S |
| T4 | P, R, U | U, R, P |
| T5 | R, S | R, S |
| T6 | S, U | U, S |
| T7 | R, S | R, S |
| T8 | H, R, S, U | U, R, S, H |
| T9 | R, S, U | U, R, S |

# Build the Tree, Step by Step (1 of 3)

| basket | sorted | what happens |
|---|---|---|
| T1 | U, R, H | empty tree: make path U:1 → R:1 → H:1 |
| T2 | U, P | U exists: U becomes 2. New child P:1 |
| T3 | U, S | U becomes 3. New child S:1 |

After T3: orange = touched now

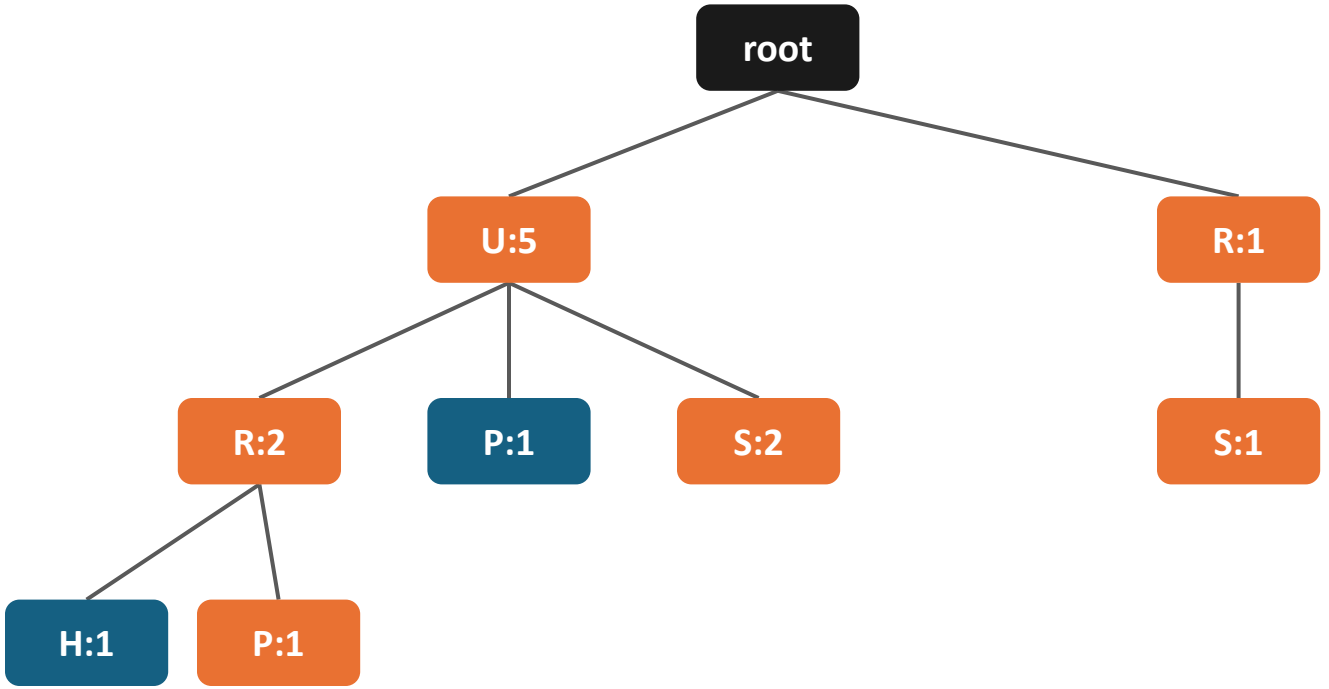



**Rule: walk down the tree. If the next item is already there, add 1. If not, make a new node with count 1.**

# Build the Tree, Step by Step (2 of 3)

| basket | sorted | what happens |
|---|---|---|
| T4 | U, R, P | U → 4, R → 2. New P:1 under R |
| T5 | R, S | root has no R child: start a NEW branch R:1 → S:1 |
| T6 | U, S | U → 5, S under U → 2. Nothing new |

After T6: orange = touched now

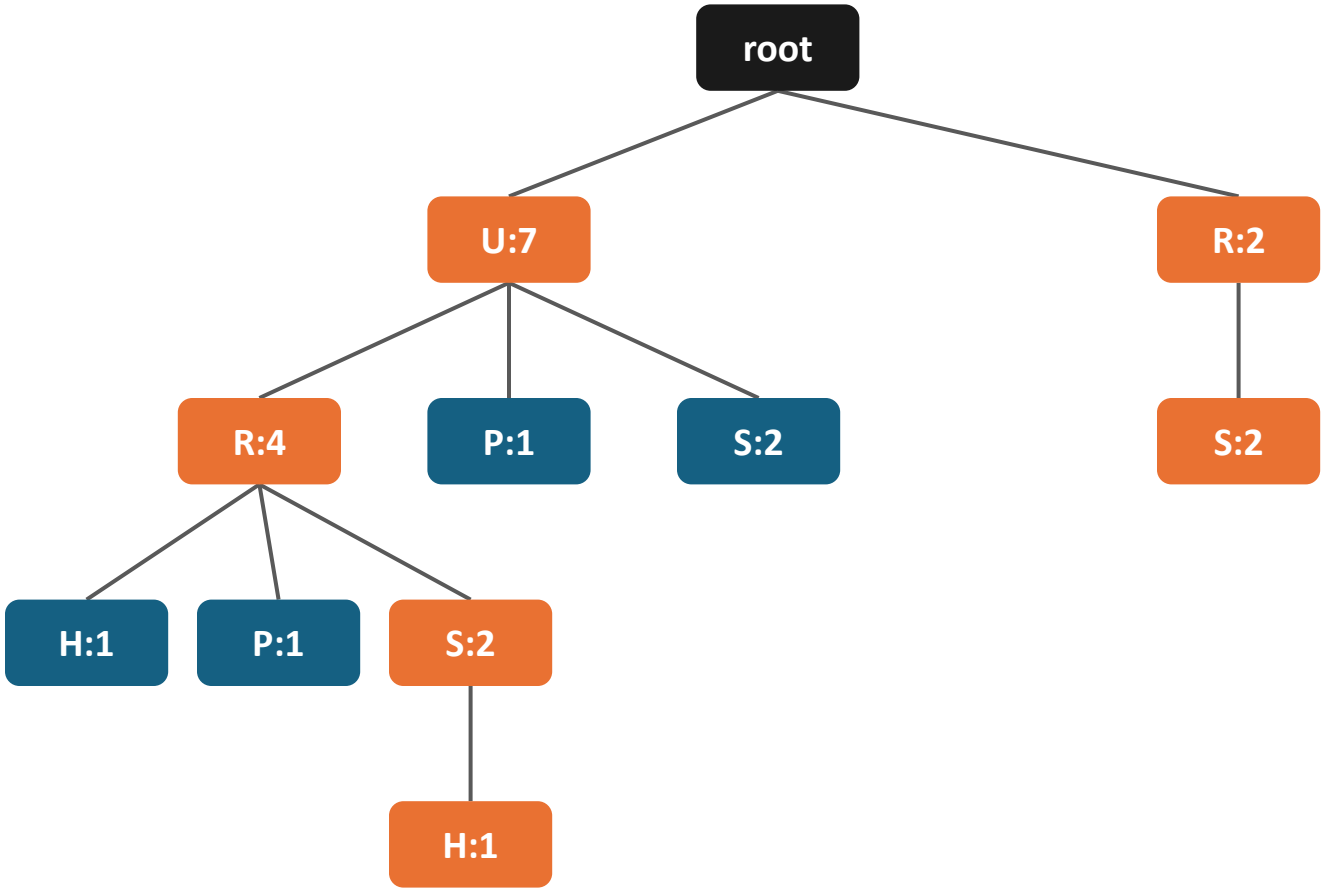



**T5 is the important one: a basket only shares a path if it STARTS with the same item.**

# Build the Tree, Step by Step (3 of 3)

| basket | sorted | what happens |
|---|---|---|
| T7 | R, S | right branch: R → 2, S → 2 |
| T8 | U, R, S, H | U → 6, R → 3. New S:1, then new H:1 |
| T9 | U, R, S | U → 7, R → 4, S → 2. Done |

After T9: orange = touched now




**Check: top counts 7 + 2 = 9 baskets. 25 items written in the data became just 10 nodes.**

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

IIT Madras Zanzibar

Foundations: What Makes Apriori Slow?

FP-Growth: Build the Tree

→ **FP-Growth: Mine the Tree**

Limited-Pass Algorithms: PCY

Random Sampling and SON

In-Class Exercise

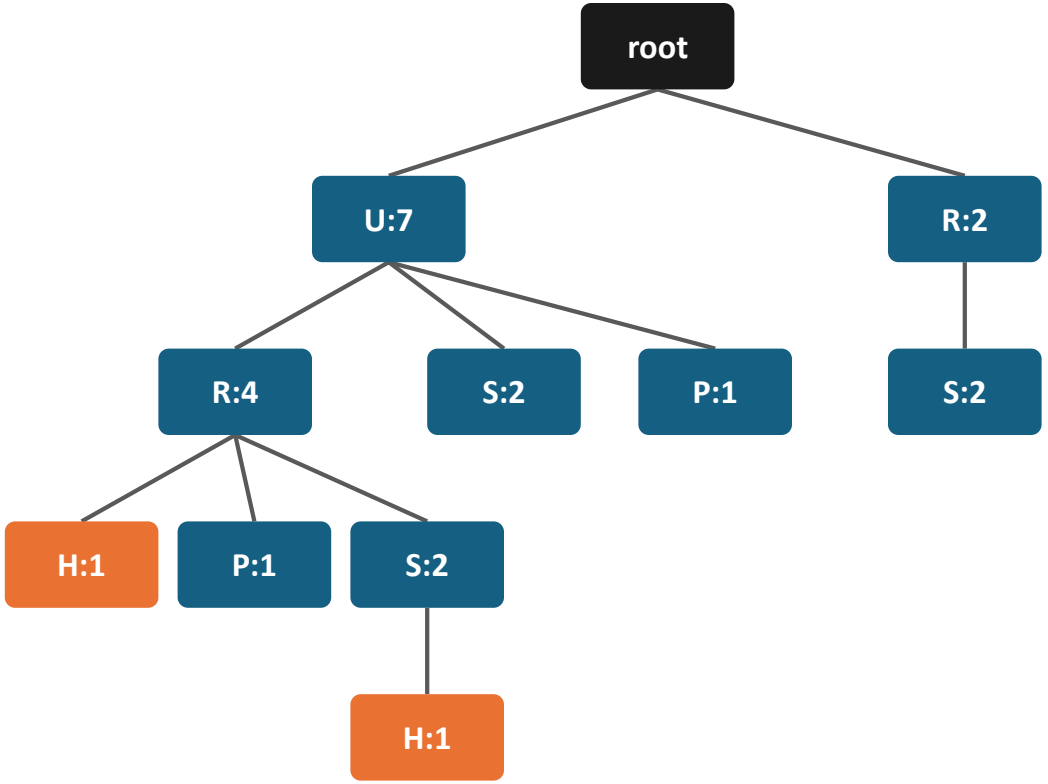
# How to Mine the Tree: a 4-Step Recipe

IIT Madras Zanzibar

| | | |
|---|---|---|
| 1 | **Pick an item** | start from the bottom (rarest): H, then P, S, R |
| 2 | **Collect its paths** | for every node of that item, write the path from the root down to it (item left out), with the node's count |
| 3 | **Count inside the paths** | items reaching min_sup are frequent WITH the chosen item |
| 4 | **Go one level deeper** | check if survivors appear together often enough → longer itemsets |

**The long name 'conditional pattern base' = the list of paths from step 2. Nothing more.**

# Mining the Tree: Conditional Pattern Bases




| path to an H node | count |
|---|---|
| U → R | 1 |
| U → R → S | 1 |

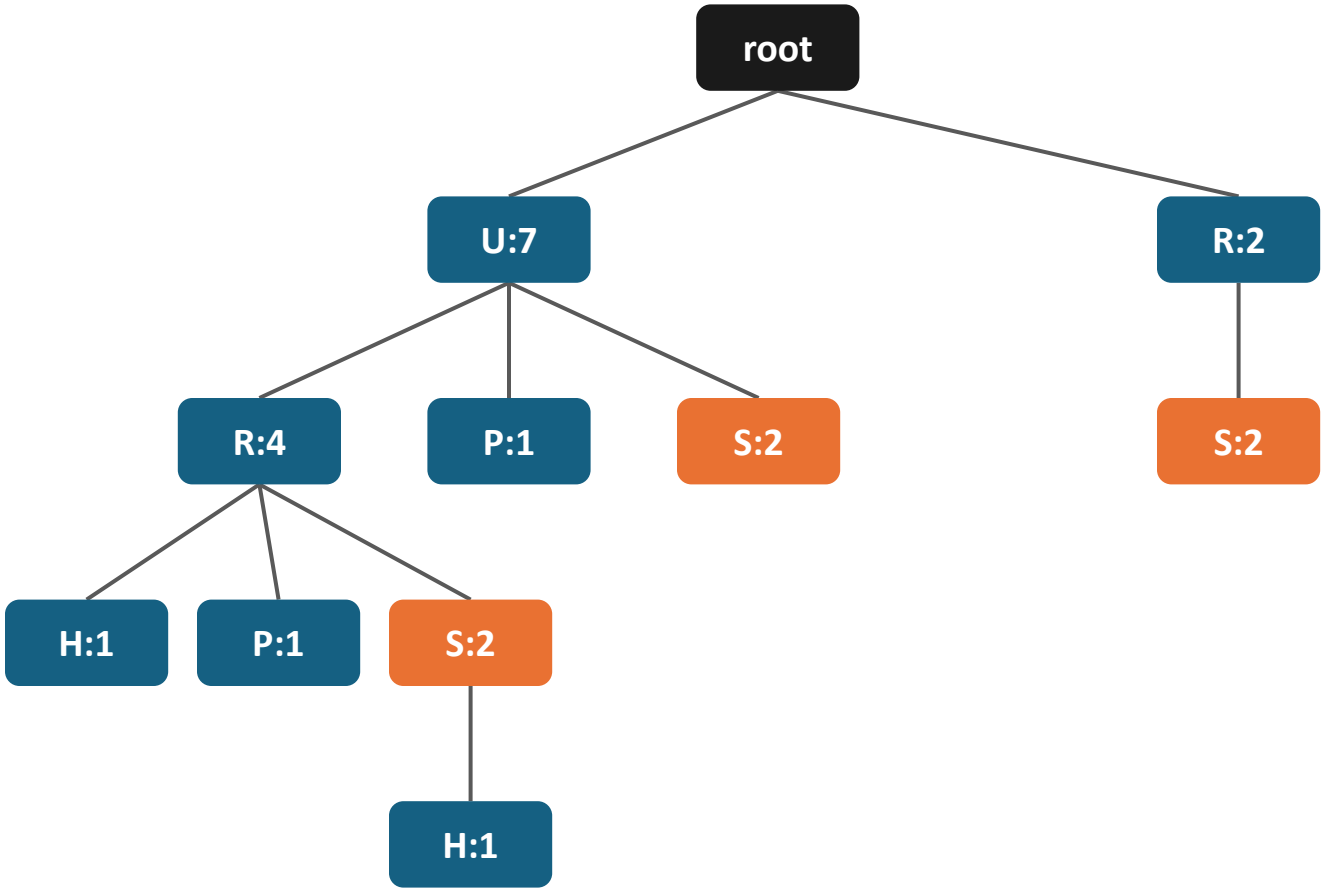
**1** **Count in the base**
U: 2  R: 2  S: 1

**2** **Frequent with H**
{U,H}:2  {R,H}:2  {U,R,H}:2

**Same answer as Apriori,** {H,R,U} with count 2 — found without generating a single candidate.

# Worked Example: Mining Item S

| step | S |
|---|---|
| 2 paths | U : 2 · U,R : 2 · R : 2 |
| 3 count | U = 2+2 = 4 R = 2+2 = 4 |
| so | {U,S}:4 {R,S}:4 |
| 4 deeper | U and R together: only U,R : 2 |
| so | {U,R,S}:2 |




**Same result as Apriori, read straight from the tree: no candidates, no extra data pass.**

# Mining the Rest of the Tree

IIT Madras Zanzibar

| suffix | conditional pattern base | frequent itemsets found |
|---|---|---|
| H | U,R : 1 · U,R,S : 1 | {U,H}:2 {R,H}:2 {U,R,H}:2 |
| P | U : 1 · U,R : 1 | {U,P}:2 |
| S | U,R : 2 · U : 2 · R : 2 | {U,S}:4 {R,S}:4 {U,R,S}:2 |
| R | U : 4 | {U,R}:4 |
| U | (nothing above U) | — |
| + single items | | U:7 R:6 S:6 H:2 P:2 |

**Total:** 5 + 6 + 2 = 13 frequent itemsets — identical to Apriori, with 2 data passes and zero candidates.

# Apriori vs FP-Growth: An Honest Comparison

IIT Madras Zanzibar

| | Apriori | FP-Growth |
|---|---|---|
| passes over the data | k (one per level) | 2 |
| candidates generated | yes, level by level | none |
| memory | candidate counters | the FP-tree (must fit in memory) |
| best when | few items, high min_sup, data in memory | many items, low min_sup, data on disk |
| on our 120,000 baskets | ≈ 0.05 s | ≈ 0.2 s — similar |

**Honest lesson:** with only 22 items, both are fast in memory. FP-Growth's real advantage is fewer passes and no candidates — which matters when data is too big for memory. You will time both in the tutorial.

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

IIT Madras Zanzibar

Foundations: What Makes Apriori Slow?

FP-Growth: Build the Tree

FP-Growth: Mine the Tree

→ **Limited-Pass Algorithms: PCY**

Random Sampling and SON

In-Class Exercise

# PCY: Use the Idle Memory in Pass 1

IIT Madras Zanzibar

1. **Pass 1**
   count items AND hash every pair to a bucket: bucket[h(pair)] += 1

2. **Between passes**
   mark buckets with count ≥ min_sup as frequent (1 bit each)

3. **Pass 2**
   count a pair only if both items are frequent AND its bucket is frequent

**Hashing in plain words**

a rule that turns any pair into a bucket number — like assigning students to rooms by the last digit of their ID.

**Why it is safe:** a bucket's count ≥ the count of every pair inside it. If the bucket is below min_sup, every pair in it is too — so skipping them never loses a frequent pair.

# PCY by Hand: a Tiny Example (Pass 1)

IIT Madras Zanzibar

| basket | items | pairs in it |
|---|---|---|
| 1 | A, B, C | AB, AC, BC |
| 2 | A, B | AB |
| 3 | A, C | AC |
| 4 | B, D | BD |

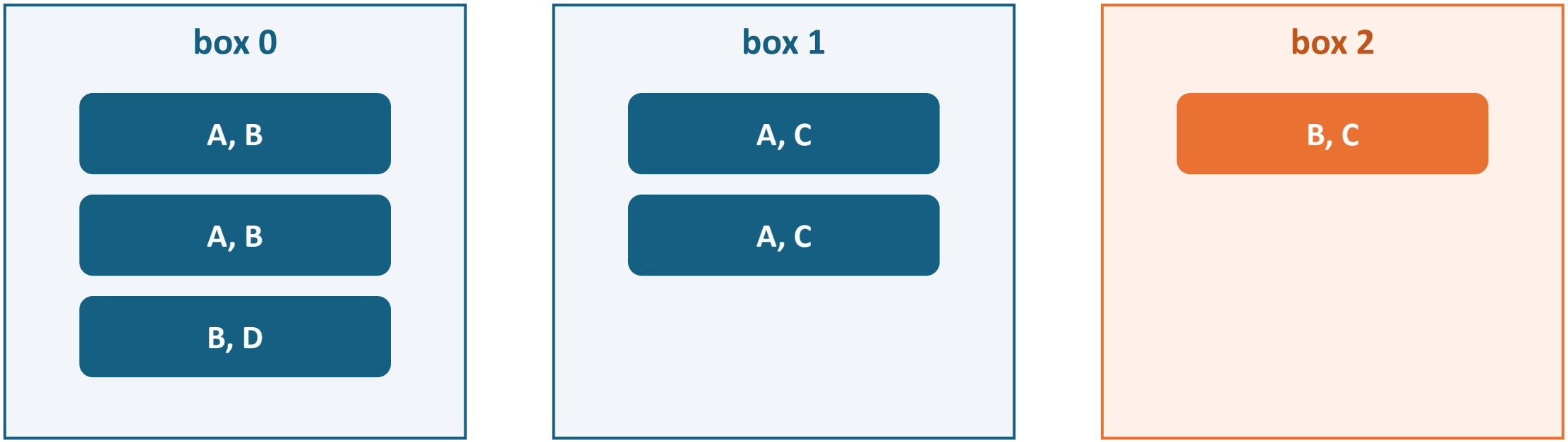
| pair | hash: (a+b) mod 3 | box |
|---|---|---|
| AB | (1+2) mod 3 = 0 | 0 |
| AC | (1+3) mod 3 = 1 | 1 |
| BC | (2+3) mod 3 = 2 | 2 |
| BD | (2+4) mod 3 = 0 | 0 |

**Single items: A 3 B 3 C 2 D 1 (D not frequent)**

**Box totals: box 0 = 2+1 = 3 box 1 = 2 box 2 = 1**

**Code values: A=1, B=2, C=3, D=4. min_sup = 2.**
**In pass 1 we keep only 3 box counters, not one counter per pair.**

# PCY in One Picture: Pairs Fall Into Boxes




**total = 3**

**≥ 2: keep, count later**

**total = 2**

**≥ 2: keep, count later**

**total = 1**

**< 2: SKIP every pair**

**A box total is at least as big as any pair inside it. Box total 1 < 2, so B,C cannot be frequent.**

# PCY by Hand: Which Pairs Get Counted (Pass 2)

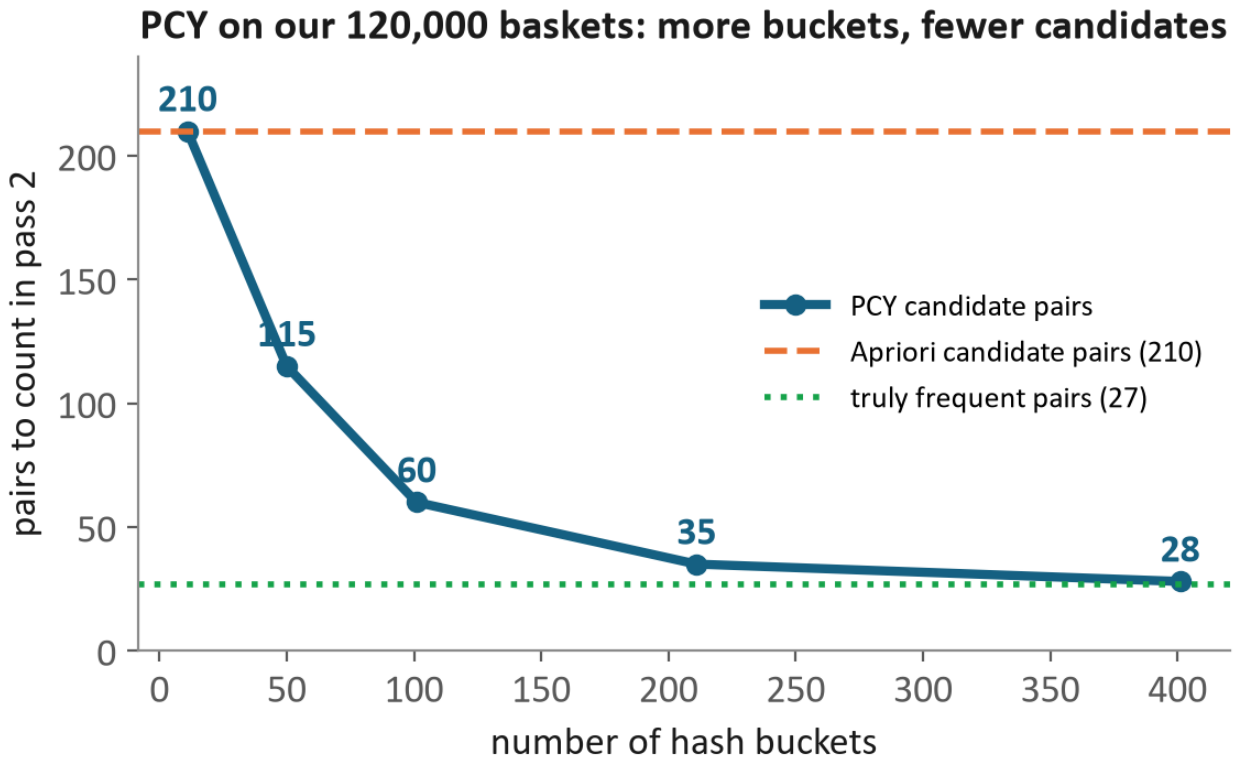
IIT Madras Zanzibar

| pair | both items frequent? | box frequent? | count in pass 2? |
|---|---|---|---|
| AB | yes | box 0 = 3  yes | YES |
| AC | yes | box 1 = 2  yes | YES |
| BC | yes | box 2 = 1  no | skip |
| BD | no (D = 1) | box 0 = 3  yes | skip |

**Apriori would count: AB, AC, BC  (3 pairs)**
**PCY counts: AB, AC  (2 pairs)**

**Why skipping BC is safe: box 2 holds only 1 in total, so no pair inside it can reach 2.**

# PCY on Our 120,000 Baskets




| buckets | pass-2 candidates |
|---|---|
| Apriori | 210 |
| 11 | 210 |
| 101 | **60** |
| 401 | **28** |
| truly frequent | 27 |

**Result:** same frequent pairs as Apriori, but pass 2 counts 60 pairs instead of 210 with 101 buckets — and 28 with 401.

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

IIT Madras Zanzibar

Foundations: What Makes Apriori Slow?

FP-Growth: Build the Tree

FP-Growth: Mine the Tree

Limited-Pass Algorithms: PCY

→ **Random Sampling and SON**

In-Class Exercise

# Random Sampling: Mine a Sample in Memory

IIT Madras Zanzibar

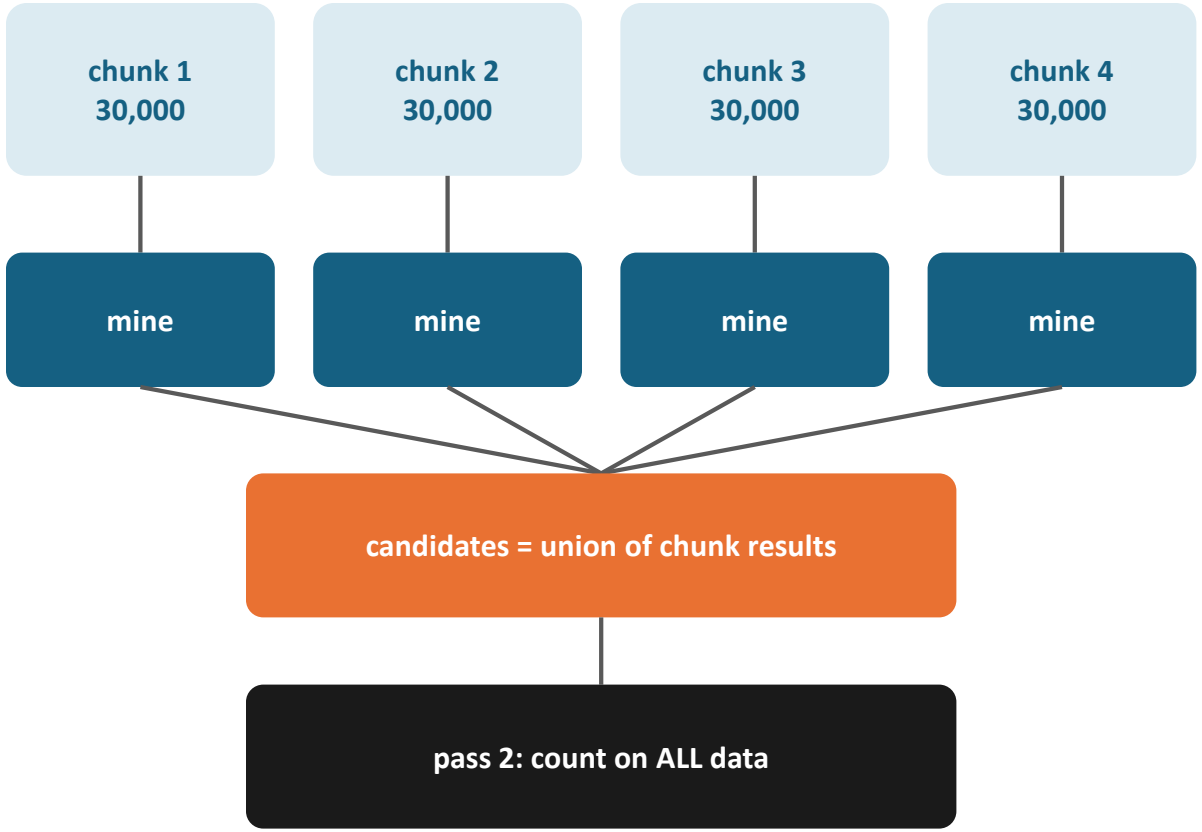
1. **Draw a sample**
   e.g. 1% of baskets — fits in memory (Lecture 8)
2. **Mine it in memory**
   threshold scaled: 1% × min_sup count
3. **Verify**
   one pass over the full data to check the candidates

| Risk | Meaning |
|---|---|
| false negative | frequent overall, missed in the sample |
| false positive | frequent in the sample by luck |

**Fix:** lower the sample threshold slightly, then verify on the full data.

**Trade-off:** very fast, but approximate unless you add the verification pass.

# SON: Split, Mine Each Chunk, Then Verify



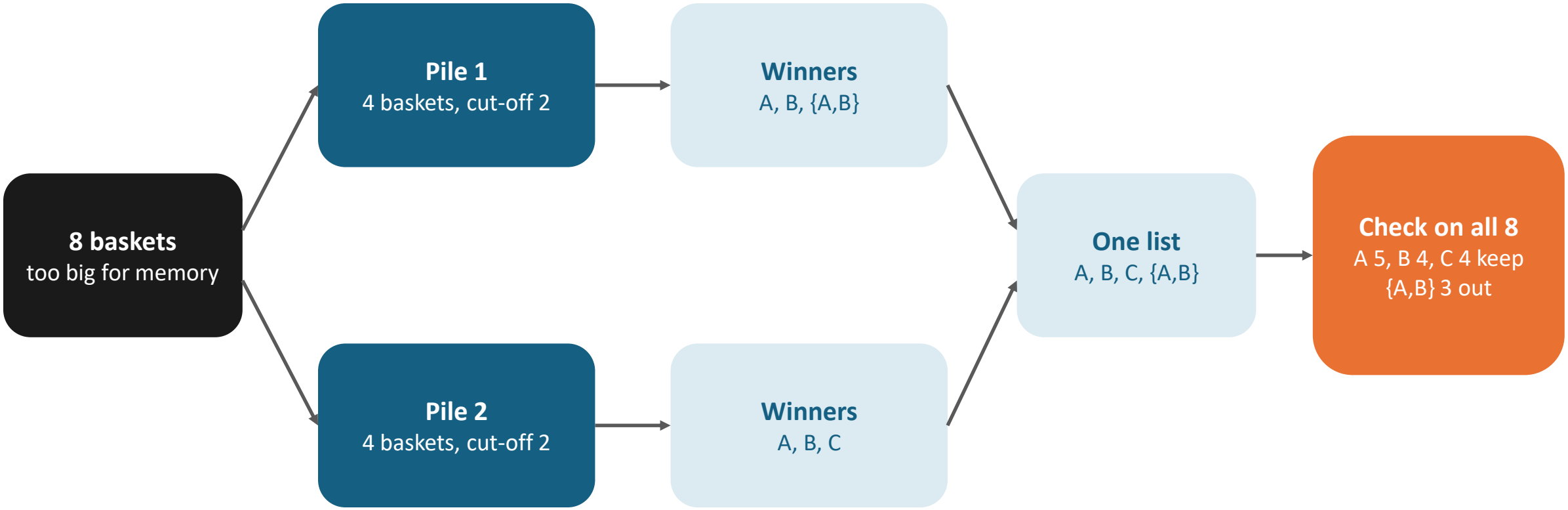

1. **Pass 1**
   mine each chunk in memory with threshold × (chunk size / total)

2. **Pass 2**
   count every candidate on the full data; keep the frequent

3. **No misses**
   frequent overall ⇒ frequent in at least one chunk

4. **Parallel**
   one chunk per computer — the MapReduce / Spark way

# SON in One Picture: Split, Find Winners, Check Once




**Final answer: A, B, C. Common overall ⇒ common in at least one pile, so nothing is missed.**

# SON by Hand: 8 Baskets, 2 Chunks

IIT Madras Zanzibar

| chunk 1 | chunk 2 |
|---|---|
| {A, B} | {A, B} |
| {A, C} | {B, C} |
| {A, B} | {C, D} |
| {D} | {A, C} |

**min_sup = 4 overall**
**→ 2 inside each chunk**

| | chunk 1 (≥2) | chunk 2 (≥2) | all 8 (≥4) |
|---|---|---|---|
| A | 3 ✓ | 2 ✓ | 5 ✓ |
| B | 2 ✓ | 2 ✓ | 4 ✓ |
| C | 1 | 3 ✓ | 4 ✓ |
| {A,B} | 2 ✓ | 1 | 3 ✗ |
| D | 1 | 1 | not a candidate |

**Pass 1: candidates = frequent in ANY chunk → A, B, C, {A,B}**
**Pass 2: count them on all data → keep A, B, C. {A,B} was only lucky in chunk 1.**

# Choosing a Method

IIT Madras Zanzibar

| Method | Passes | Exact? | Use when |
|---|---|---|---|
| **Apriori** | k | yes | data in memory, few items |
| **FP-Growth** | 2 | yes | many items, low min_sup; the tree fits in memory |
| **PCY** | k (cheaper level 2) | yes | pair counters don't fit in memory |
| **Random sample** | 1 (+1 to verify) | approximately | quick exploration |
| **SON** | 2 | yes | data far bigger than memory; parallel machines |

# Scaled Pattern Mining in the Real World

IIT Madras Zanzibar

**1 Supermarket chains**

millions of receipts a day → SON / Spark FP-Growth

**2 Spark MLlib**

ships a parallel FP-Growth used in industry

**3 E-commerce**

billions of sessions → sampling then verification

**4 Telecom**

call and data records mined on clusters

**5 Mobile money**

transaction patterns mined nightly in parallel

**6 Web search**

queries often searched together

**Common thread:** when data outgrows memory: fewer passes, fewer candidates, more machines.

# Scaling Pattern Mining: FP-Growth and Limited-Pass Algorithms

IIT Madras Zanzibar

Foundations: What Makes Apriori Slow?

FP-Growth: Build the Tree

FP-Growth: Mine the Tree

Limited-Pass Algorithms: PCY

Random Sampling and SON

→ **In-Class Exercise**

# In-Class Exercise (12 minutes, pairs)

IIT Madras Zanzibar

**IN-CLASS TASK**

Baskets: {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}.  min_sup count = 3.

1. Count each item and give the frequency order (drop infrequent items).
2. Rewrite each basket in that order and build the FP-tree.
3. Give the conditional pattern base for D, and the frequent itemsets ending in D.
4. PCY: pairs hash to 3 buckets with counts 2, 7, 9. Which bucket's pairs are never counted in pass 2?

# Solution

IIT Madras Zanzibar

1. **Counts**
   A:8, B:7, C:6, D:5, E:3 → order A, B, C, D, E (all frequent)

2. **Tree**
   two branches under the root: A:8 (8 baskets) and B:2 (holding {B,C,D} and {B,C,E})

3. **D's base**
   paths A,B,C:1 · A,B:1 · A,C:1 · A:1 · B,C:1 → A:4, B:3, C:3 → {A,D}:4, {B,D}:3, {C,D}:3 (no triple reaches 3)

4. **PCY**
   the bucket with count 2 (< 3): none of its pairs can be frequent

# Summary & Exam Checkpoints

IIT Madras Zanzibar

**Passes dominate**

On large data, cost ≈ passes × data size. Apriori: k passes; FP-Growth and SON: 2.

**FP-Growth**

Sort items by frequency, build a prefix tree, mine conditional pattern bases from the bottom up. No candidates.

**PCY**

Hash pairs into buckets in pass 1; skip pairs in buckets below min_sup. Ours: 210 → 60 candidates.

**Sampling and SON**

Sample: fast, approximate. SON: chunks in memory + one verification pass; exact and parallel.

# Common Questions

**Q** **Is FP-Growth always faster than Apriori?**

**A** No. With few items and data in memory they are similar (as on our data). It wins on passes and candidates at scale.

**Q** **What if the FP-tree does not fit in memory?**

**A** Split the data (SON-style) or project the database item by item and build smaller trees.

**Q** **Can PCY ever miss a frequent pair?**

**A** No. A frequent pair always makes its bucket frequent, so it always survives to pass 2.

**Q** **Where would I use SON in practice?**

**A** Spark or Hadoop: each machine mines its chunk; one more pass verifies the candidates.

# What's Next

IIT Madras Zanzibar

**Tut** **Tutorial 5 (today, 4:45)**
Apriori from scratch on 9 baskets; mlxtend Apriori vs FP-Growth on 120,000; rules with lift

**Fri** **Pattern evaluation**
$\chi^2$, all-confidence, Kulczynski and the null-transaction problem

**Mini-project:** dataset approval is due tomorrow, Friday 16 October.

# References & Further Reading


- Han, J., Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation. Proc. ACM SIGMOD, 1–12.
- Park, J. S., Chen, M.-S. & Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. Proc. ACM SIGMOD, 175–186.
- Savasere, A., Omiecinski, E. & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases. Proc. VLDB, 432–444.
- Leskovec, J., Rajaraman, A. & Ullman, J. D. Mining of Massive Datasets — Chapter 6.3–6.4 (PCY, limited-pass algorithms).