MONASH
INFORMATION
TECHNOLOGY

FIT5202 – Data Processing for Big Data

Revision
Presented by
Peter Liu

Developed by
Prajwol Sangat

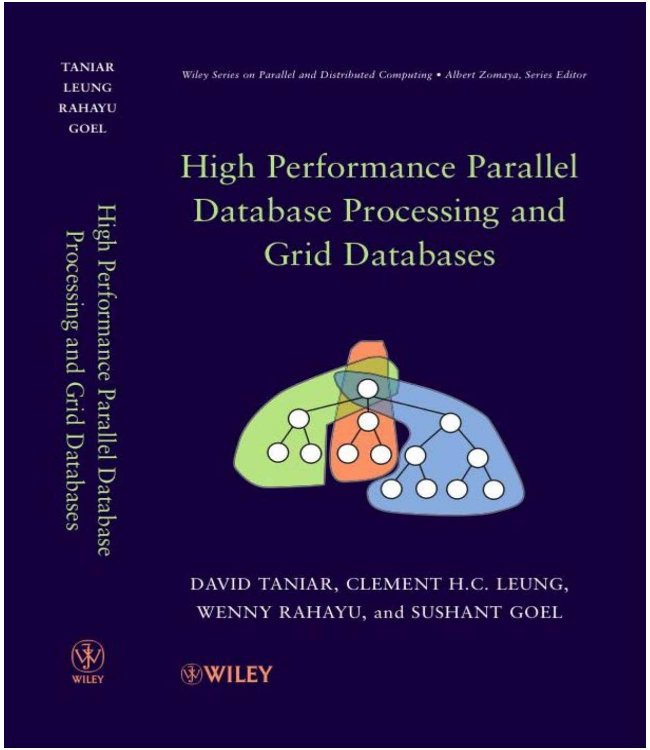
Unit Overview

1. **Volume** ➔ Sessions 1, 2, 3, 4
   – How to process Big Data Volume?

2. **Complexity** ➔ Sessions 5, 6, 7, 8
   – How to apply machine learning algorithms to every aspect of Big Data?

3. **Velocity** ➔ Sessions 9, 10, 11
   – How to handle and process Fast Streaming Data?

Volume ➔ Session 1, 2, 3, 4



Chapter 1 Introduction

Flux Quiz 1

Using the current processing resources, we can finish processing 1TB (one terabyte) of data in 1 hour. Recently the volume of data has increased to 2TB and the management has decided to double up the processing resources. Using the new processing resources, we can finish processing the 2 TB in 60 minutes.

Is this speed up or scale up?

Solution:

Using x resources (current resources), 1TB = 60 minutes
When the resources are doubled (e.g. x becomes 2x now), a linear scale up is being able to complete 2TB in 60 minutes.

In the question, using 2x resources, it finishes 2 TB in 60 minutes. Therefore, it is linear scale up.

Flux Quiz 2

There is a job that will take 1 hour to complete, if this is done by 1 processor. The serial part of this job is 10%. There are 4 processors to use in this job, but each processor will have an overhead of 20% due to waiting time, communication time, etc. What type of speed up do we get?

Solution:

1 processor = 60min; Serial part = 10% = 6min; Parallel part = 54min

4 processors = 54min/4 = 13.5min
Overhead = 20%
Hence, parallel processing part = 13.5min + 20%overhead = 13.5min+2.7min = 16.2min
Total time = 6min + 16.2min = 22.2min

Speed up = 60min / 22.2min = 2.7
Linear speedup should be 4. Speed up of 2.7 is Sub-Linear Speedup

Flux Quiz 3

There 100,000 records in the table to be distributed to 32 processors. Assuming that the skewness degree is high (theta = 1), what is the estimated number of records in the heaviest processor?

$$|R_i| = \frac{|R|}{i^{\theta} \times \sum_{j=1}^{N} \frac{1}{j^{\theta}}} \qquad \text{where } 0 \leq \theta \leq 1 \qquad (2.1)$$

Solution:
i = 1(heaviest processor)
$\theta = 1$
1+ ½ + 1/3 + ¼ + …+ 1/32 = 4.05
Number of records = 100,000 / 4.05 = 24691

Skew

- Data Skew

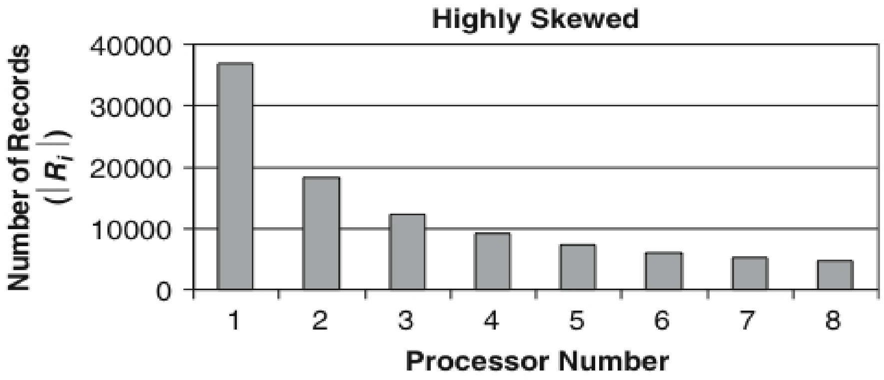
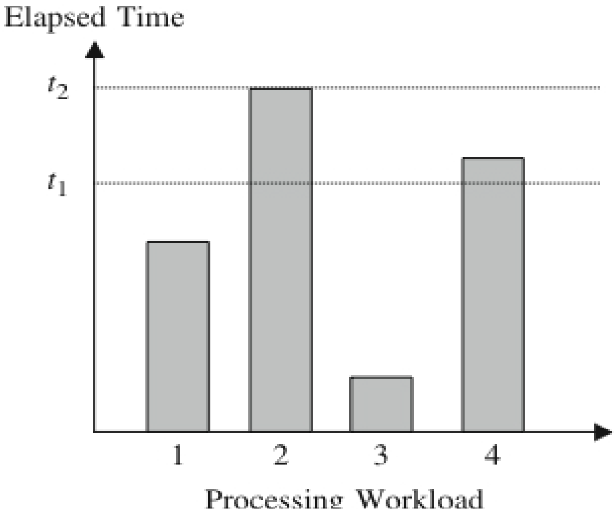


Figure 2.2 Highly skewed distribution

- Processing Skew



Shared-Something Architecture

A mixture of shared-memory and shared-nothing architectures

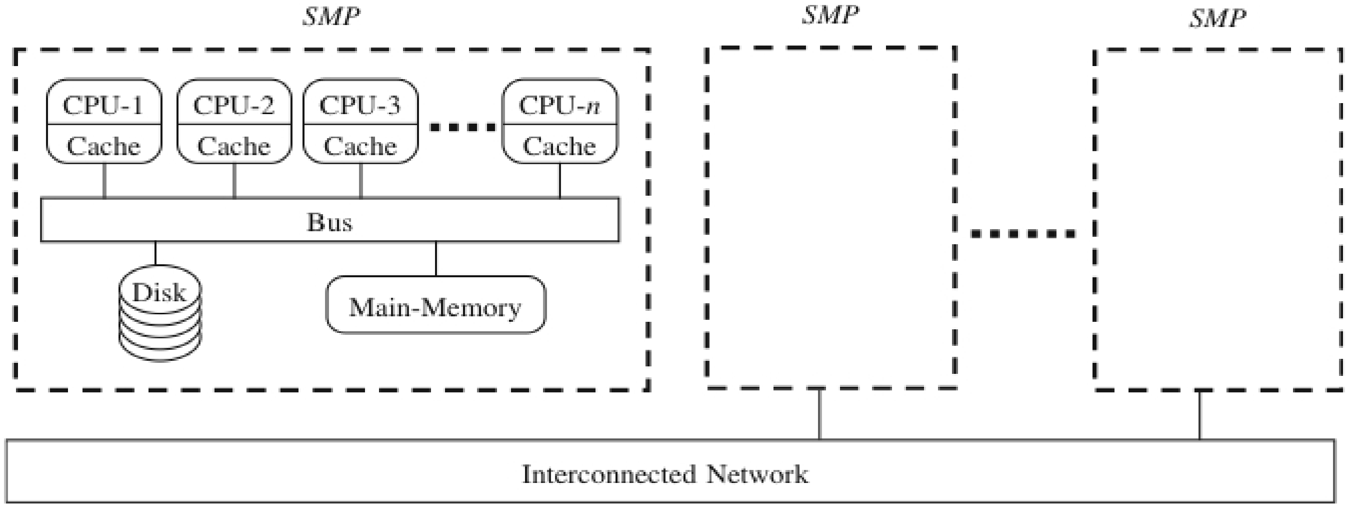
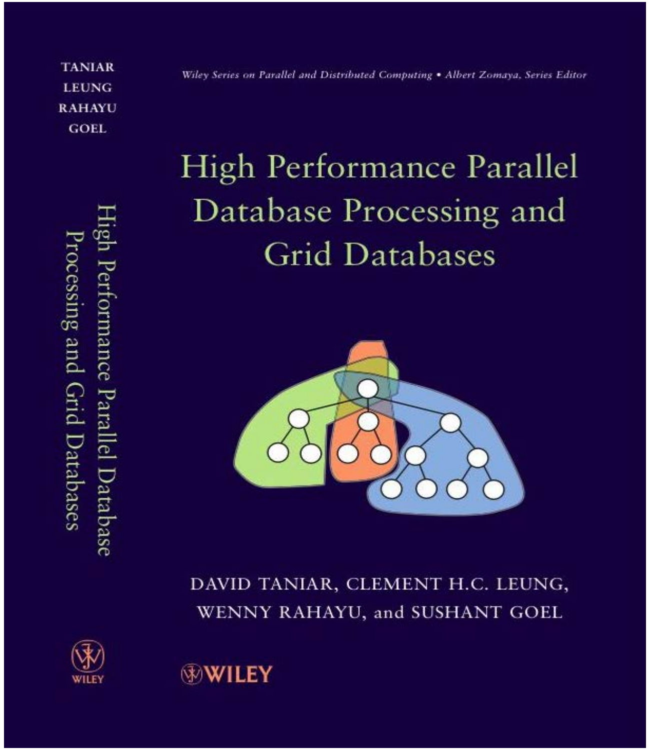


Figure 1.14 Cluster of SMP architectures



Chapter 3
Parallel Search

Data Partitioning

- **Basic Data Partitioning**
  - Round-robin or random equal data partitioning
  - Hash data partitioning
  - Range data partitioning
  - Random-unequal data partitioning

- **Complex Data Partitioning**
  - Complex data partitioning is based on multiple attributes or is based on a single attribute but with multiple partitioning methods
  - Hybrid-Range Partitioning Strategy (HRPS)
  - Multiattribute Grid Declustering (MAGIC)
  - Bubba’s Extended Range Declustering (BERD)

Data Partitioning (cont'd)

Round-robin data partitioning

- Each record in turn is allocated to a processing element in a clockwise manner
- "Equal partitioning" or "Random-equal partitioning"
- Data evenly distributed, hence supports load balance
- But data is not grouped semantically

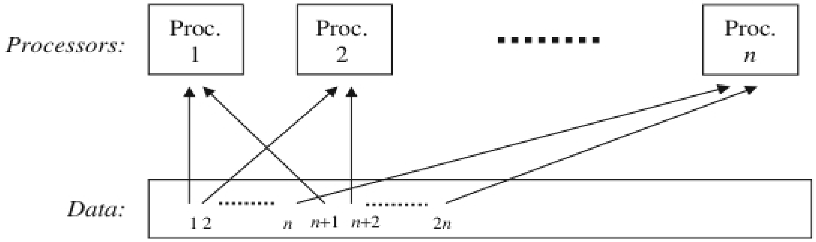


Figure 3.3 *Round-robin* data partitioning

Data Partitioning (cont'd)

Hash data partitioning

- A hash function is used to partition the data
- Hence, data is grouped semantically, that is data on the same group shared the same hash value
- Selected processors may be identified when processing a search operation (exact-match search), but for range search (especially continuous range), all processors must be used
- Initial data allocation is not balanced either

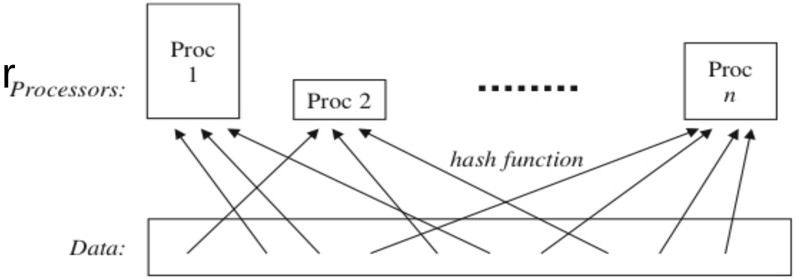


Figure 3.4 *Hash* data partitioning

Data Partitioning (cont'd)

Range data partitioning

- Spreads the records based on a given range of the partitioning attribute
- Processing records on a specific range can be directed to certain processor only
- Initial data allocation is skewed too

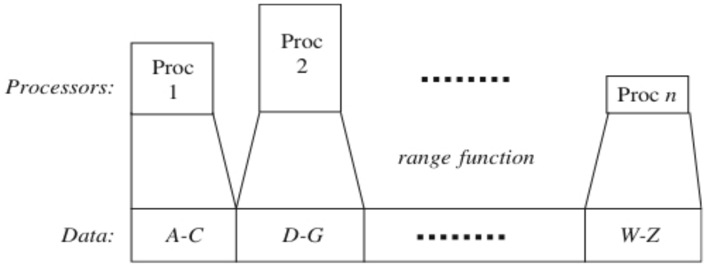


Figure 3.5 *Range* data partitioning

Search Algorithms

Serial search algorithms:

- Linear search
- Binary search

Parallel search algorithms:

- Processor activation or involvement
- Local searching method
- Key comparison

Search Algorithms (cont'd)

Processor activation or involvement

- The number of processors to be used by the algorithm
- If we know where the data to be sought are stored, then there is no point in activating all other processors in the searching process
- Depends on the data partitioning method used
- Also depends on what type of selection query is performed

Table 3.6 Processor activation or involvement of parallel search algorithms

		Data Partitioning Methods			
		Random-Equal	Hash	Range	Random-Unequal
Exact Match		All	1	1	All
Range Selection	Continuous	All	All	Selected	All
	Discrete	All	Selected	Selected	All

Search Algorithms (cont'd)

Local searching method

- The searching method applied to the processor(s) involved in the searching process
- Depends on the data ordering, regarding the type of the search (exact match of range)

Table 3.7 Local searching method of parallel search algorithms

		Records Ordering	
		Ordered	**Unordered**
Exact Match		Binary Search	Linear Search
Range Selection	**Continuous**	Binary Search	Linear Search
	Discrete	Binary Search	Linear Search

Search Algorithms (cont’d)

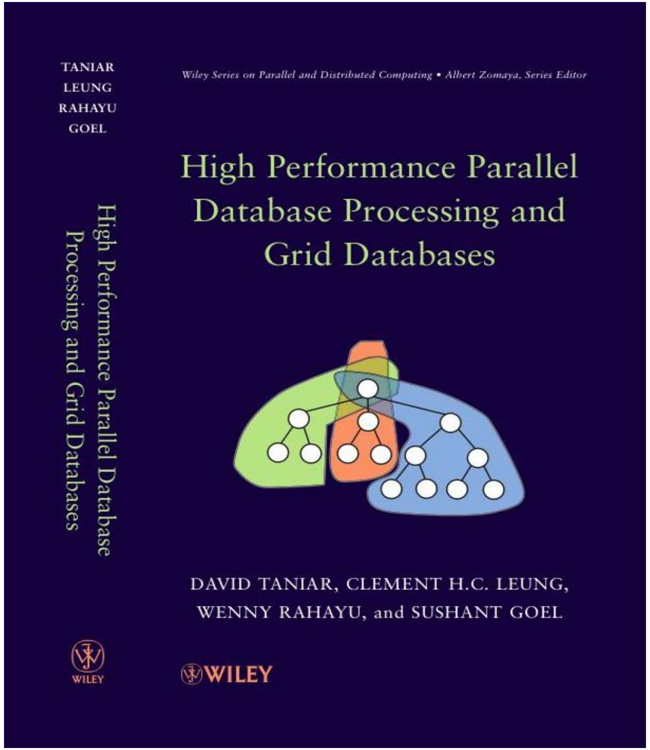
Key comparison

- Compares the data from the table with the condition specified by the query
- When a match is found: continue to find other matches, or terminate
- Depends on whether the data in the table is unique or not

Table 3.8 Key comparison of parallel search algorithms

		Search Attribute Values	
		Unique	**Duplicate**
Exact Match		Stop	Continue
Range Selection	**Continuous**	Continue	Continue
	Discrete	Continue	Continue

Volume → Session 1, 2, 3, 4



Chapter 5
Parallel Join

Join Algorithms

- Parallel Inner Join components
  - **Data Partitioning**
    - Divide and Broadcast
    - Disjoint Partitioning
  - **Local Join**
    - Nested-Loop Join
    - Sort-Merge Join
    - Hash Join
- Example of a Parallel Inner Join Algorithm
  - **Divide and Broadcast, plus Hash Join**

Serial Join Algorithms (cont'd)

Hash-based Join Algorithm

- The records of files *R* and *S* are both hashed to the *same hash file*, using the *same hashing function* on the join attributes *A* of *R* and *B* of *S* as hash keys
- A single pass through the file with fewer records (say, *R*) hashes its records to the hash file buckets
- A single pass through the other file (*S*) then hashes each of its records to the appropriate bucket, where the record is combined with all matching records from *R*

Serial Join Algorithms (cont'd)

Table S

Arts	8
Business	15
CompSc	2
Dance	12
Engineering	7
Finance	21
Geology	10
Health	11
IT	18

hashed into →

Hash Table

Index	*Entries*	
1	Geology/10	
2	CompSc/2	Health/11
3	Dance/12	Finance/21
4		
5		
6	Business/15	
7	Engineering/7	
8	Arts/8	
9	IT/18	
10		
11		
12		

Figure 5.6. Hashing Table *S*

Serial Join Algorithms (cont'd)

Table R

Adele	8
Bob	22
Clement	16
Dave	23
Ed	11
Fung	25
Goel	3
Harry	17
Irene	14
Joanna	2
Kelly	6
Lim	20
Meng	1
Noor	5
Omar	19

probed into

Hash Table

Index	***Entries***	
1	Geology/10	
2	CompSc/2	Health/11
3	Dance/12	Finance/21
4		
5		
6	Business/15	
7	Engineering/7	
8	Arts/8	
9	IT/18	
10		
11		
12		

Join Results

Adele	8	Arts
Ed	11	Health
Joanna	2	CompSc

Figure 5.7. Probing Table *R*

Parallel Join Algorithms (cont’d)

Divide and Broadcast-based Parallel Join Algorithms

- Two stages: data partitioning using the divide and broadcast method, and a local join
- Divide and Broadcast method: Divide one table into multiple disjoint partitions, where each partition is allocated a processor, and broadcast the other table to all available processors
- Dividing one table can simply use equal division
- Broadcast means replicate the table to all processors
- Hence, choose the smaller table to broadcast and the larger table to divide

Parallel Join Algorithms (cont'd)

Processor 1

R1

Adele	8
Bob	22
Clement	16
Dave	23
Ed	11

S1

Arts	8
Dance	15
Geology	10

Processor 2

R2

Fung	25
Goel	3
Harry	17
Irene	14
Joanna	2

S2

Business	12
Engineering	7
Health	11

Processor 3

R3

Kelly	6
Lim	20
Meng	1
Noor	5
Omar	19

S3

CompSc	2
Finance	21
IT	18

Figure 5.10 Initial data placement

Parallel Join Algorithms (cont'd)

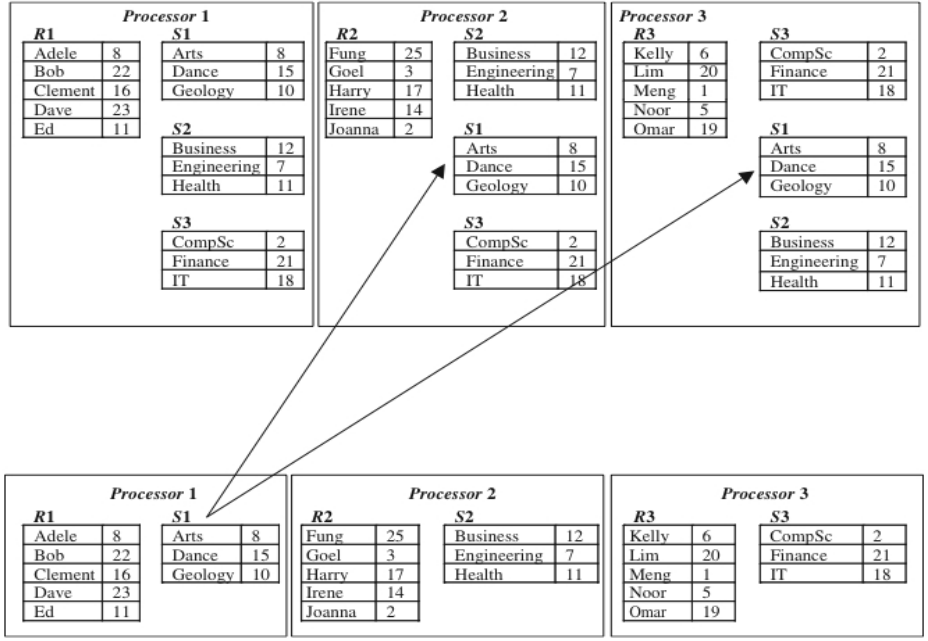


Figure 5.11 Divide and broadcast result

Parallel Join Algorithms (cont'd)

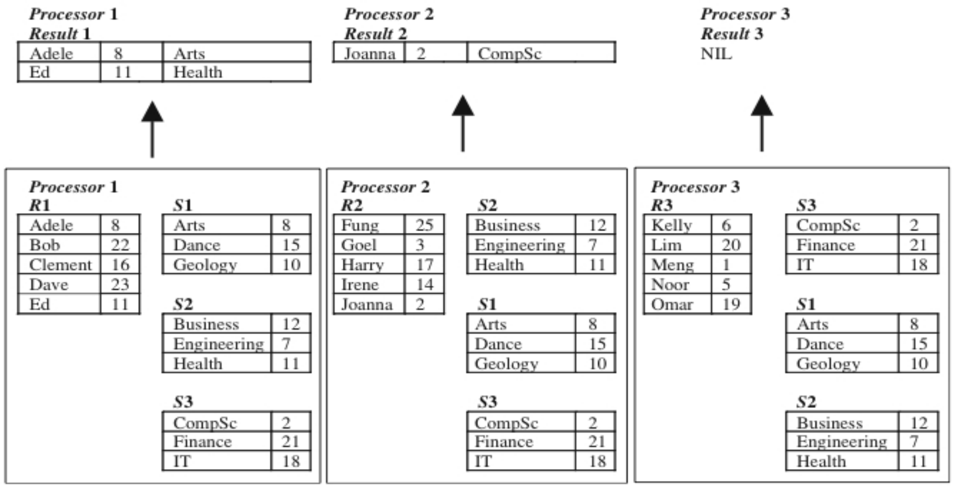


Figure 5.12 Join results based on divide and broadcast

Cost Models?

- **Divide and Broadcast**
  - Join two tables (**table R** and **table S**)
  - The two tables have been partitioned and stored in 3 processors
  - The tables have been partitioned using the random-equal data partitioning method
  - The table fragments are called R1, R2, R3, and S1, S2, S3 (in general, each fragment is called **Ri** or **Si**, where i is the processor number)

Flux Quiz 4

|S| = 600 records, each record has the length of 100 bytes, and N=3.

Solution:

S = 60000 bytes
Si = S/N = 60000/3 = 20000 bytes

|S| = 600 records
|Si| = 600/3 = 200 records

Divide and Broadcast based parallel join

Transfer cost = (Si/P) x (N-1) x (mp+ml)

Receiving cost = (S/P – Si/P) x (mp)

Table 2.1 Cost notations

Symbol	Description
Data parameters	
R	Size of table in bytes
R_i	Size of table fragment in bytes on processor i
$\lvert R \rvert$	Number of records in table R
$\lvert R_i \rvert$	Number of records in table R on processor i
Systems parameters	
N	Number of processors
P	Page size
H	Hash table size
Query parameters	
π	Projectivity ratio
σ	Selectivity ratio
Time unit cost	
IO	Effective time to read a page from disk
t_r	Time to read a record in the main memory
t_w	Time to write a record to the main memory
t_d	Time to compute destination
Communication cost	
m_p	Message protocol cost per page
m_l	Message latency for one page

Parallel Join Query Processing

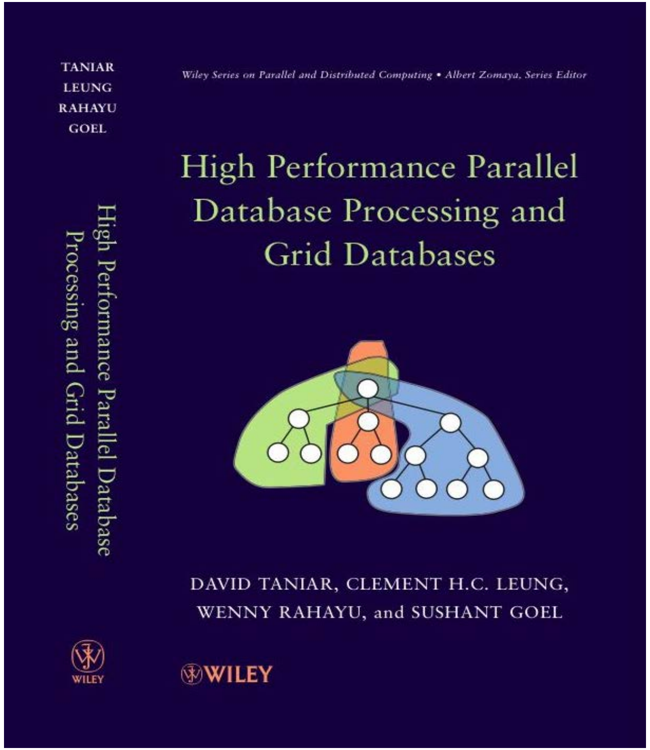
Parallel Outer Join processing methods

- **ROJA** (Redistribution Outer Join Algorithm)
- **DOJA** (Duplication Outer Join Algorithm)
- **DER** (Duplication & Efficient Redistribution)

Load Balancing

- **OJSO** (Outer Join Skew Optimization)

	ROJA	DOJA	DER
Steps	**Step 1:** Distribute or reshuffle the data based on the join attribute. **Step 2:** Each processor performs the Local outer Join.	**Step 1:** Replication. We duplicate the small table. **Step 2:** Local Inner Join **Step 3:** Hash redistribute the inner join result based on attribute X. **Step 4:** Local outer join	**Step 1:** Replication. We broadcast the left table. **Step 2:** Local Inner Join **Step 3:** Select the ROW ID of left table with no matches. **Step 4:** Redistribute the ROW ID. **Step 5:** Store the ROW ID that appears as many times as the number of processors. **Step 6:** Inner join
Pros	fast performance, only two steps	None. ROJA is faster than DOJA.	Redistributes dangling row IDs instead of actual records.
Cons	redistribution of data -> data skew, communication cost	In the replication step, if the table is large, the replication cost is expensive. In the distribution step, data skew and communication cost similar to ROJA	In the replication step, if the table is large, the replication cost is expensive.



Chapter 4
Parallel Sort and GroupBy

Sorting, and Serial Sorting

- Serial Sorting – **INTERNAL**
  - The data to be sorted fits entirely into the main memory
  - Bubble Sort
  - Insertion Sort
  - Quick Sort

- Serial Sorting - **EXTERNAL**
  - The data to be sorted DOES NOT fit entirely into the main memory
  - Sort-Merge

Flux Quiz 5

There are 150 data pages to be sorted. The machine that we have has a limited memory, and can only take 8 pages at a time.

How many passes will it take to sort the 150 data pages?

Solution:

File size to be sorted = 150 pages, number of buffer (or memory size) = 8 pages

Number of subfiles = 150/8 = 19 subfiles (Last subfile has only 6 pages)

Pass 0 (sorting phase): For each subfile, read from disk, sort in main-memory, and write to disk

Merging phase: We use **7 buffers for input** and **1 buffer for output**

Pass 1: Read 7 sorted subfiles and perform 7-way merging. Repeat the 7-way merging until all subfiles are processed. Result = 3 subfiles

Pass 2: Merge the 3 subfiles

Summary: 150 pages and 8 buffer pages require 3 passes

Parallel External Sort

- Parallel Merge-All Sort
- Parallel Binary-Merge Sort
- Parallel Redistribution Binary-Merge Sort
- Parallel Redistribution Merge-All Sort
- Parallel Partitioned Sort

Parallel External Sort (cont'd)

Parallel Merge-All Sort

- A traditional approach
- Two phases: local sort and final merge
- Load balanced in local sort
- Problems with merging:
  - Heavy load on one processor
  - Network contention

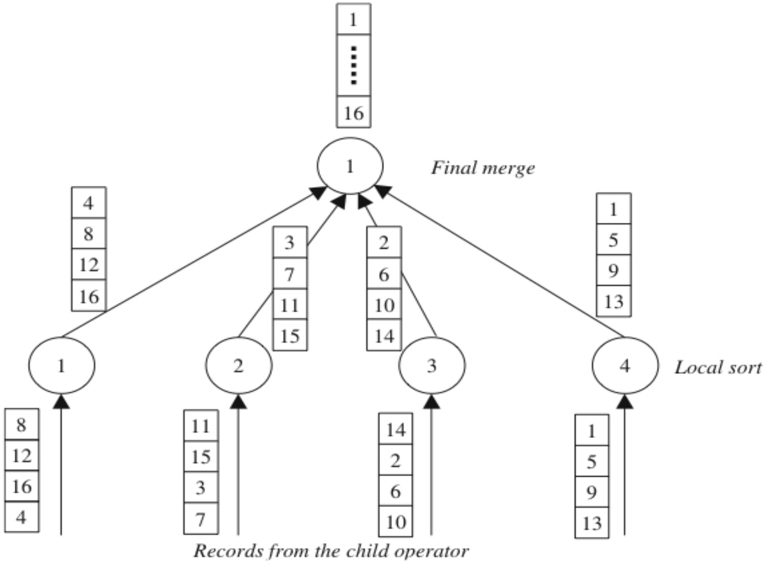


Figure 4.3 Parallel merge-all sort

Parallel External Sort (cont’d)

- **Parallel Binary-Merge Sort**
  - Local sort similar to traditional method
  - Merging in pairs only
  - Merging work is now spread to pipeline of processors, but merging is still heavy

5

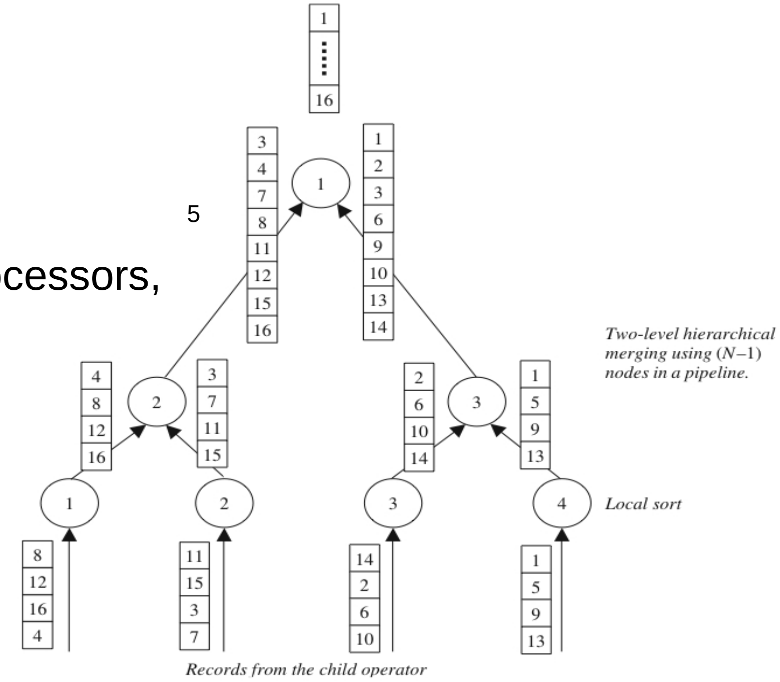


Figure 4.4 Parallel binary-merge sort

Parallel Redistribution Binary-Merge Sort

Parallelism at all levels in the pipeline hierarchy

Step 1: local sort

Step 2: redistribute the results of local sort

Step 3: merge using the same pool of processors

Benefit: merging becomes lighter than without redistribution

Problem: height of the tree

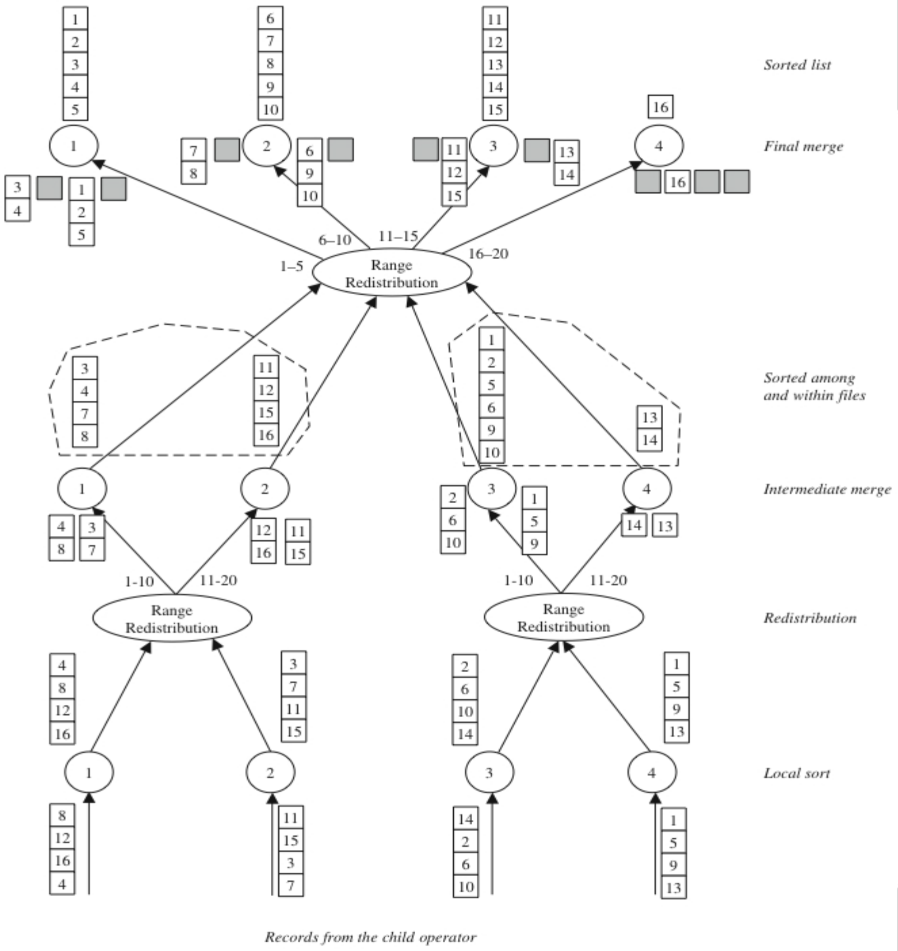


Figure 4.6 Parallel redistribution binary-merge sort

Parallel Redistribution Merge-All Sort

- Reduce the height of the tree, and still maintain parallelism
- Like parallel merge-all sort, but with redistribution
- The advantage is true parallelism in merging
- Skew problem in the merging

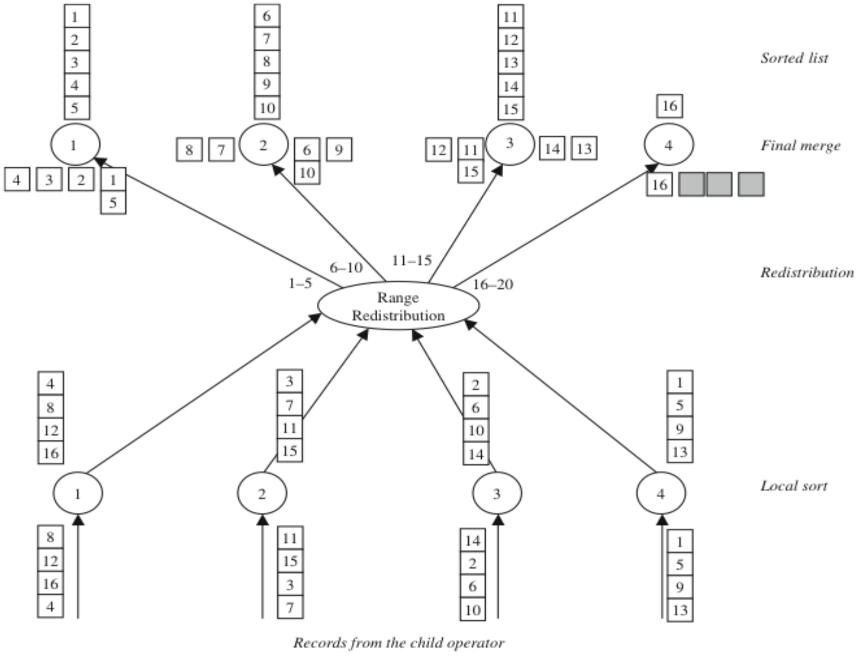


Figure 4.7 Parallel redistribution merge-all sort

Parallel Partitioned Sort

- Two stages: Partitioning stage and Independent local work
- Partitioning (or range redistribution) may raise load skew
- Local sort is done after the partitioning, not before
- No merging is necessary
- Main problem: **Skew** produced by the partitioning

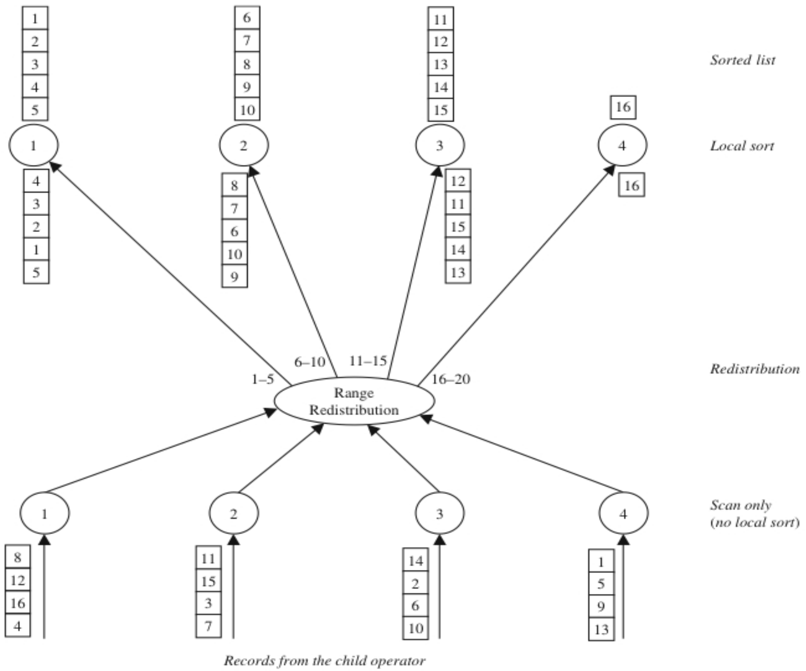


Figure 4.8 Parallel partitioned sort

- **Exercise**
- Given a data set *D* = {8,11,14,1,12,15,2,5,16,3,6,9,4,7,10,13} and four processors, show step by step how the **Parallel Partitioned Sort** works.

- **Initial Data Partitioning** (Round Robin):
  - P1 = {8,12,16,4}, P2 = {11,15,3,7}, P3 = {14,2,6,10} and P4 = {1,5,9,13}
- **Range Redistribution** (Range Logic: P1 = 1-5, P2 = 6-10, P3 = 11-15, P4 = 16-20)
  - P1 = {4,3,2,1,5}, P2 = {8,7,6,10,9}, P3 = {12,11,15,14,13}, P4 = {16}
- **Local Sort**
  - P1 = {1,2,3,4,5}, P2 = {6,7,8,9,10}, P3 = {11,12,13,14,15}, P4 = {16}

Parallel Group By

- Traditional methods (Merge-All and Hierarchical Merging)
- Two-phase method
- Redistribution method

Parallel Group By (cont'd)

Traditional Methods

Step 1: local aggregate in each processor

Step 2: global aggregation

May use a Merge-All or Hierarchical method

Need to pay a special attention to some aggregate functions (AVG) when performing a local aggregate process

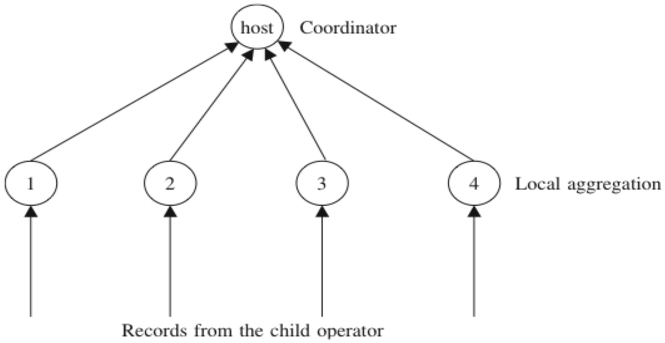


Figure 4.10 Traditional method

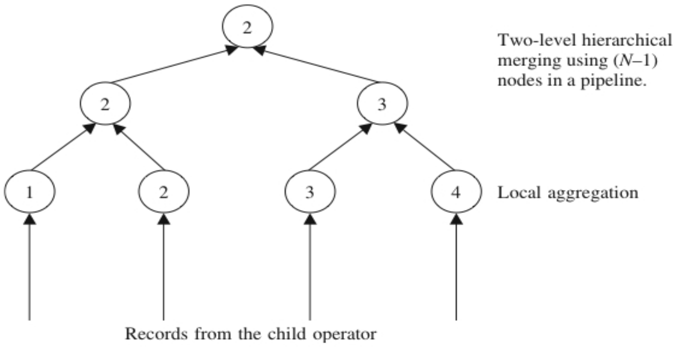


Figure 4.11 Hierarchical merging method

Parallel Group By (cont'd)

Two-Phase Method

Step 1: local aggregate in each processor. Each processor groups local records according to the groupby attribute

Step 2: global aggregation where all temp results from each processor are redistributed and then final aggregate is performed in each processor

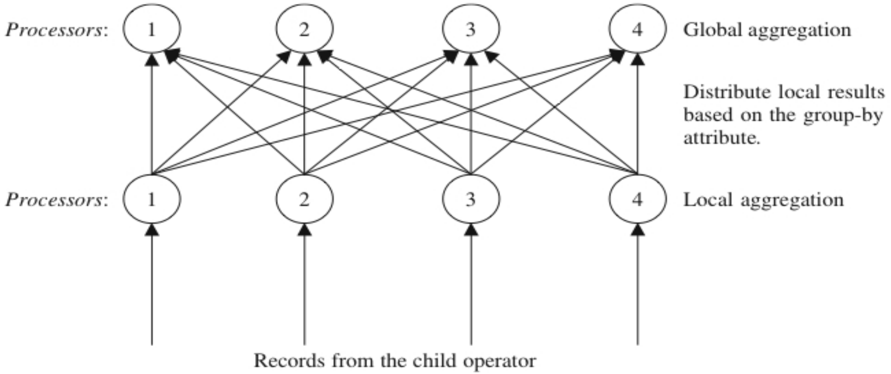


Figure 4.12 Two-phase method

Parallel Group By (cont'd)

Redistribution Method

Step 1 (Partitioning phase): redistribute raw records to all processors

Step 2 (Aggregation phase): each processor performs a local aggregation

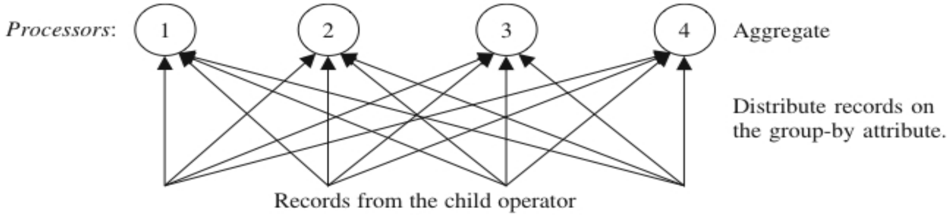


Figure 4.13 Redistribution method

Flux Quiz 7

The Redistribution Method has a load balancing option, through the Task Stealing method.

The Two-Phase Method does not have a load balancing problem.

Solution: False

Unit Overview

1. **Volume** ➔ Sessions 1, 2, 3, 4
   – How to process Big Data Volume?

2. **Complexity** ➔ Sessions 5, 6, 7, 8
   – How to apply machine learning algorithms to every aspect of Big Data?

3. **Velocity** ➔ Sessions 9, 10, 11
   – How to handle and process Fast Streaming Data?

Machine Learning

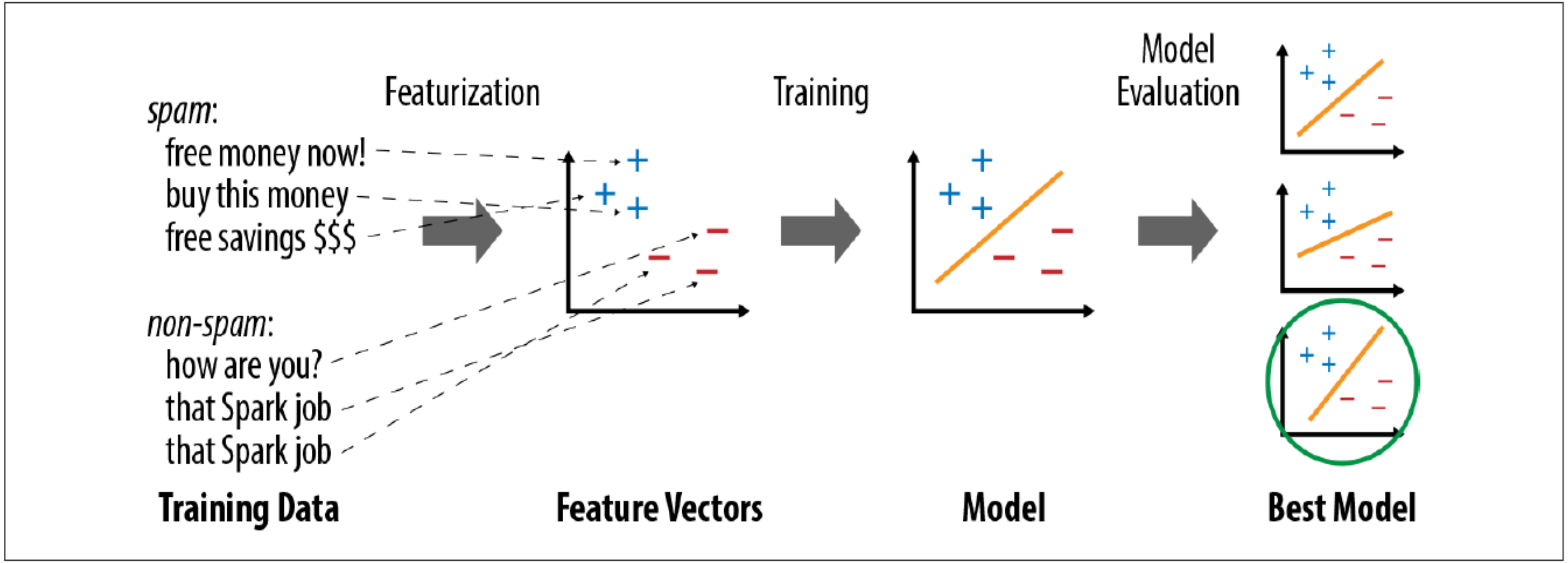


Figure 11-1. Typical steps in a machine learning pipeline

Machine Learning: Featurization

- **Extraction:** Extracting features from “raw” data
  - Count Vectorizer
  - TF-IDF
  - Word2Vec
- **Transformation:** Scaling, converting, or modifying features
  - Tokenization
  - Stop Words Remover
  - String Indexing
  - One Hot Encoding
  - Vector Assembler
- **Selection:** Selecting a subset from a larger set of features
  - Vector Slicer

Flux Quiz 8

In a product recommendation task, simply adding another feature (e.g., realizing that which book you should recommend to a user might also depend on which movies she’s watched) could give a large improvement in results.

Solution: True

Types of Machine Learning

- Supervised,
- Unsupervised

Types of Machine Learning: Supervised

- In supervised machine learning, the data consists of a set of input records.
- Each of these records have associated labels.
- The goal is to predict the output label(s) given a new unlabeled input.
- Two types of supervised machine learning:

1. *Classification* and
2. *Regression*.

Supervised Machine Learning: Classification

Binary classification example: dog or not dog

Supervised Machine Learning: Classification

Multinomial classification example: Australian shepherd, golden retriever, or poodle

Decision Trees: To Jog or Not To Jog

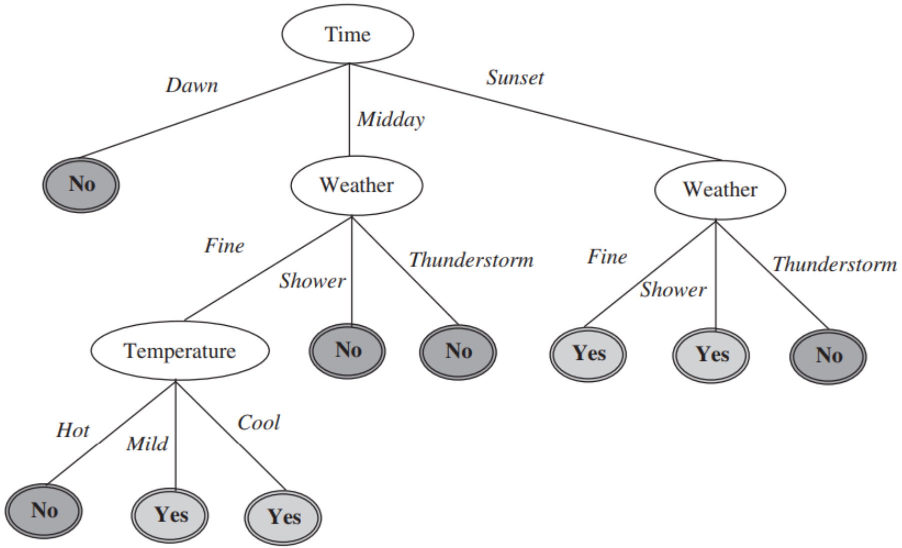


Figure 17.15 Final decision tree

A decision tree is constructed based only on the given training dataset. It is not based on a universal belief.

Flux Quiz 9

What is the entropy for the training data set in the table below?

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

$$H(S) = \sum_{x \in X} p(x) \log \frac{1}{p(x)}$$

$$IG(S,A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

ID3 (Iterative Dichotomiser 3)

- ## Steps

1. Compute the entropy for data set
2. For every attribute/feature:

   2.1. Calculate entropy for all categorical values

   2.2 Take average information entropy for the current attribute

   2.3 Calculate gain for the current attribute
3. Pick the highest gain attribute
4. Repeat until the tree is complete

ID3

Entropy for the given probability of the target classes, $p_1, p_2, \ldots, p_n$ where $\sum_{i=1}^{n} p_i = 1$, can be calculated as follows:

$$entropy(p_1, p_2, \ldots, p_n) = \sum_{i=1}^{n} (p_i \log(1/p_i)) \quad (17.2)$$

$$entropy(Yes, No) = 5/15 \times \log(15/5) + 10/15 \times \log(15/10) = 0.2764 \quad (17.3)$$

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

- Step 1: Calculate entropy for the training dataset in Figure 17.11. The result is previously calculated as 0.2764 (see equation 17.3).

Jog	
Yes	No
5	10

ID3

$$entropy(\text{Weather}=Fine) = 4/7\times\log(7/4) + 3/7\times\log(7/3) = 0.2966 \quad (17.4)$$

$$entropy(\text{Weather}=Shower) = 1/4\times\log(4/1) + 3/4\times\log(4/3) = 0.2442 \quad (17.5)$$

- Step 2: Process attribute *Weather*
  - Calculate weighted sum entropy of attribute *Weather*:
    $entropy(Fine) = 0.2966$
    $entropy(Shower) = 0.2442$
    $entropy(Thunderstorm) = 0 + 4/4\times\log(4/4) = 0$
    $weighted\ sum\ entropy(Weather) = 0.2035$
  - Calculate information gain for attribute *Weather*:
    $gain\ (Weather) = 0.0729$

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

		Jog		
		Yes	**No**	
Weather	**Fine**	4	3	7
	Shower	1	3	4
	Thunderstorm	0	4	4
				15

ID3

Weighted sum entropy (*Weather*) = Weighted entropy (*Fine*)
+ Weighted entropy (*Shower*)
+ Weighted entropy (*Thunderstorm*)

$$= 7/15 \times 0.2966 + 4/15 \times 0.2442 + 4/15 \times 0$$
$$= 0.2035 \quad (17.6)$$

- Step 2: Process attribute *Weather*
  - Calculate weighted sum entropy of attribute *Weather*:
    $entropy(Fine) = 0.2966$
    $entropy(Shower) = 0.2442$
    $entropy(Thunderstorm)=0 + 4/4\times\log(4/4) = 0$
    $weighted\ sum\ entropy(Weather) = 0.2035$
  - Calculate information gain for attribute *Weather*:
    $gain\ (Weather) = 0.0729$

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

		Jog		
		Yes	No	
Weather	Fine	4	3	7
	Shower	1	3	4
	Thunderstorm	0	4	4
				15

ID3

$$gain(Weather) = entropy(\text{training dataset } D) - entropy(\text{attribute } Weather)$$
$$= 0.2764 - 0.2035$$
$$= 0.0729 \quad (17.7)$$

- Step 2: Process attribute *Weather*
  - Calculate weighted sum entropy of attribute *Weather*:
    $entropy(Fine) = 0.2966$ (equation 17.4)
    $entropy(Shower) = 0.2442$ (equation 17.5)
    $entropy(Thunderstorm)=0 + 4/4\times\log(4/4) = 0$
    $weighted\ sum\ entropy(Weather) = 0.2035$ (equation 17.6)
  - Calculate information gain for attribute *Weather*:
    $gain\ (Weather) = 0.0729$ (equation 17.7)

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

ID3

- Step 3: Process attribute *Temperature*
  - Calculate weighted sum entropy of attribute *Temperature*:
    $entropy(Hot) = 2/5\times\log(5/2) + 3/5\times\log(5/3) = 0.2923$
    $entropy(Mild) = entropy(Hot)$
    $entropy(Cool) = 1/5\times\log(5/1) + 4/5\times\log(5/4) = 0.2173$
    $weighted\ sum\ entropy(Temperature) = 5/15\times0.2923 + 5/15\times0.2173$
    $= 0.2674$
  - Calculate information gain for attribute *Temperature*:
    $gain\ (Temperature) = 0.2764 - 0.2674 = 0.009$

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

		Jog		
		Yes	No	
Temperature	Hot	2	3	5
	Mild	3	2	5
	Cool	1	4	5
				15

ID3

- Step 4: Process attribute *Time*
  - Calculate weighted sum entropy of attribute *Time*:

    $entropy(Dawn) = 0 + 5/5 \times \log(5/5) = 0$

    $entropy(Midday) = 2/6 \times \log(6/2) + 4/6 \times \log(6/4) = 0.2764$

    $entropy(Sunset) = 3/4 \times \log(4/3) + 1/4 \times \log(4/1) = 0.2443$

    $weighted\ sum\ entropy\ (Time) = 0 + 6/15 \times 0.2764 + 4/15 \times 0.2443 = 0.1757$

  - Calculate information gain for attribute *Time*:

    $gain\ (Temperature) = 0.2764 – 0. 1757 = 0.1007$

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

		Jog		
		Yes	No	
Time	Dawn	0	5	5
	Midday	2	4	6
	Sunset	3	1	4
				15

ID3

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

Step 5: Process attribute *Day*

- Calculate weighted sum entropy of attribute *Day*:

$$entropy(Weekday) = 4/10 \times \log(10/4) + 6/10 \times \log(10/6) = 0.2923$$

$$entropy(Weekend) = 1/5 \times \log(5/1) + 4/5 \times \log(5/4) = 0.2173$$

$$weighted\ sum\ entropy\ (Day) = 10/15 \times 0.2923 + 5/15 \times 0.2173 = 0.2674$$

- Calculate information gain for attribute *Day*:

$$gain(Temperature) = 0.2764 — 0.2674 = 0.009$$

		Jog		
		Yes	No	
Day	Weekend	4	6	10
	Weekday	1	4	5
				15

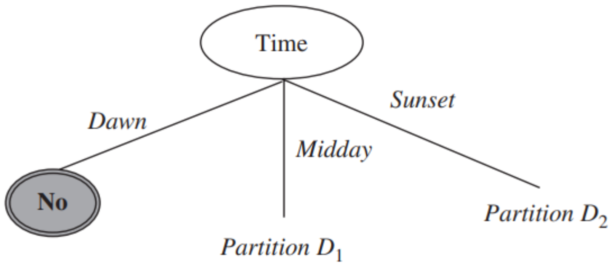


Figure 17.13 Attribute *Time* as the root node

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

Comparing equations 17.7, 17.8, 17.9, and 17.10 ,and 17.10 for the gain of each other attributes (Weather, Temperature, Time, and Day), the biggest gain is *Time*, with gain value = 0.1007 (see equation 17.9), and as a result, attribute *Time* is chosen as the first splitting attribute. A partial decision tree with the root node *Time* is shown in Figure 17.13.

ID3

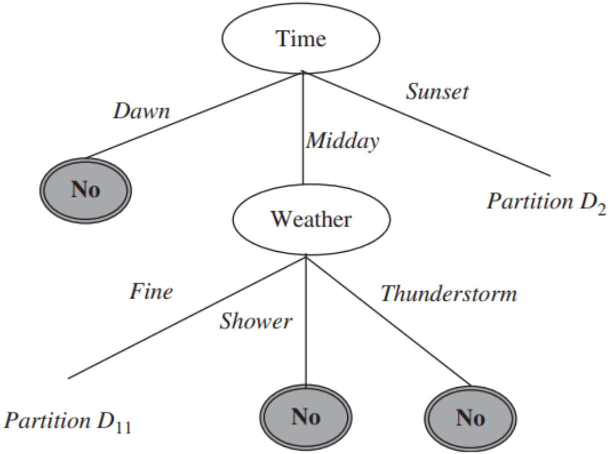


Figure 17.14 Attribute *Weather* as next splitting attribute

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

- The next stage is to process partition D_1 consisting of records with Time=*Midday*. Training dataset partition D_1 consists of 6 records with record#: 3, 6, 8, 9, 10, and 15. The next task is to determine the splitting attribute for partition D_1, whether it is *Weather*, *Temperature*, or *Day*.

Decision Trees: To Jog or Not To Jog

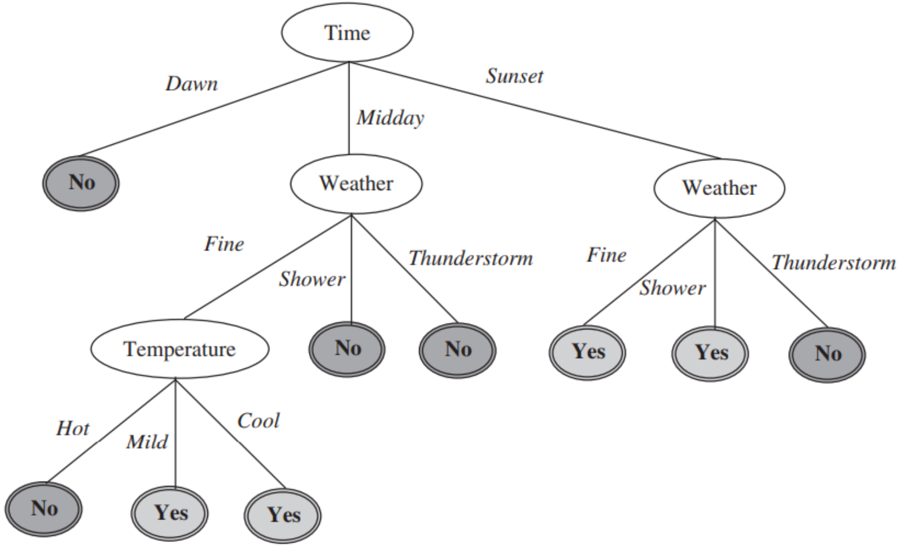
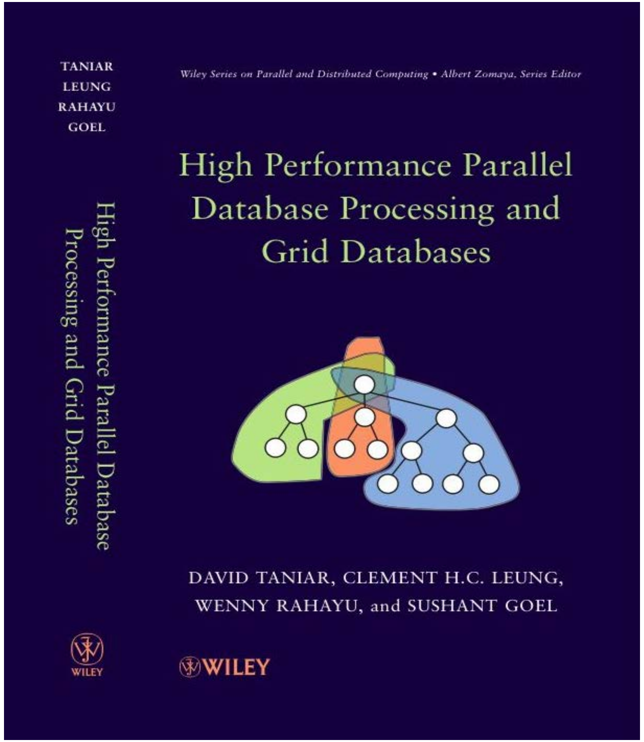


Figure 17.15 Final decision tree

A decision tree is constructed based only on the given training dataset. It is not based on a universal belief.



Chapter 17
Parallel Clustering and Classification

Parallel Classification: Decision Tree

Data parallelism

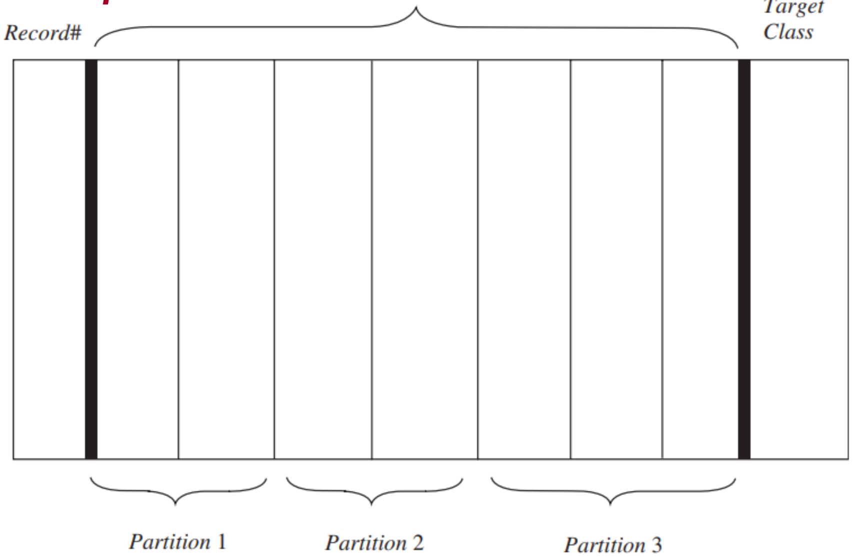


Figure 17.16 Vertical data partitioning of training data set

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

Parallel Classification: Decision Tree

Data parallelism:

Rec#	Weather	Temperature	Jog (*Target Class*)
1	Fine	Mild	Yes
2	Fine	Hot	Yes
3	Shower	Mild	No
4	Thunderstorm	Cool	No
5	Shower	Hot	Yes
6	Fine	Hot	No
7	Fine	Cool	No
8	Thunderstorm	Cool	No
9	Fine	Cool	Yes
10	Fine	Mild	Yes
11	Shower	Hot	No
12	Shower	Mild	No
13	Fine	Cool	No
14	Thunderstorm	Mild	No
15	Thunderstorm	Hot	No

Partition 1

Rec#	Time	Day	Jog (*Target Class*)
1	Sunset	Weekend	Yes
2	Sunset	Weekday	Yes
3	Midday	Weekday	No
4	Dawn	Weekend	No
5	Sunset	Weekday	Yes
6	Midday	Weekday	No
7	Dawn	Weekend	No
8	Midday	Weekday	No
9	Midday	Weekday	Yes
10	Midday	Weekday	Yes
11	Dawn	Weekend	No
12	Dawn	Weekday	No
13	Dawn	Weekday	No
14	Sunset	Weekend	No
15	Midday	Weekday	No

Partition 2

Parallel Classification: Decision Tree

Data parallelism

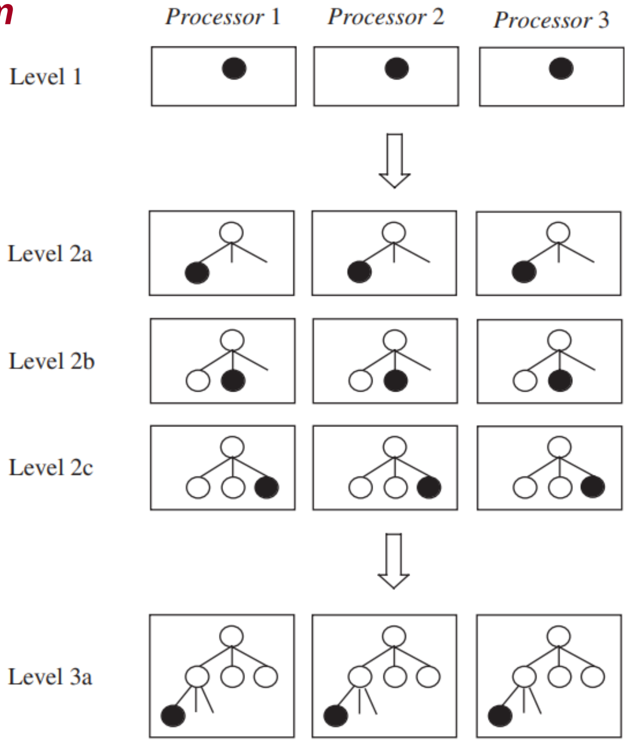


Figure 17.17 Data parallelism of parallel decision tree construction

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

Figure 17.11. Training dataset

Parallel Classification: Decision Tree

Result parallelism

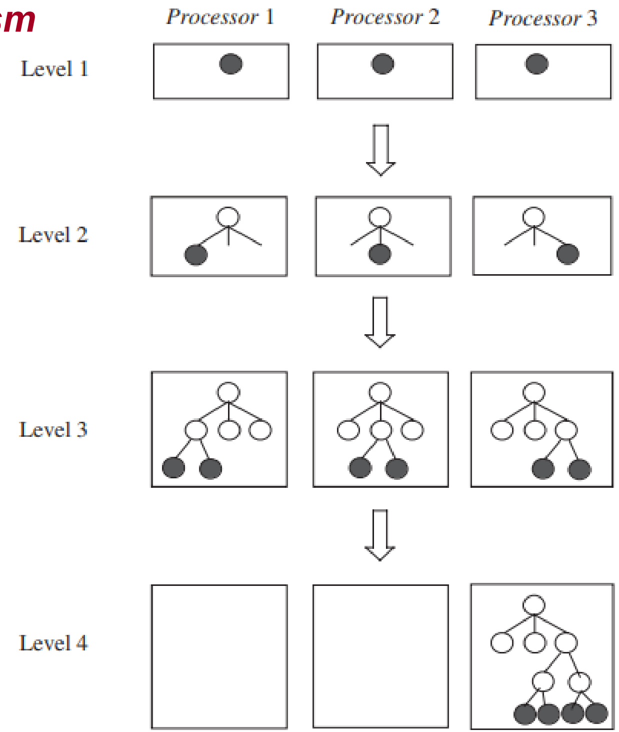


Figure 17.20 Result parallelism of parallel decision tree construction

Rec#	Weather	Temperature	Time	Day	Jog (*Target Class*)
1	Fine	Mild	Sunset	Weekend	Yes
2	Fine	Hot	Sunset	Weekday	Yes
3	Shower	Mild	Midday	Weekday	No
4	Thunderstorm	Cool	Dawn	Weekend	No
5	Shower	Hot	Sunset	Weekday	Yes
6	Fine	Hot	Midday	Weekday	No
7	Fine	Cool	Dawn	Weekend	No
8	Thunderstorm	Cool	Midday	Weekday	No
9	Fine	Cool	Midday	Weekday	Yes
10	Fine	Mild	Midday	Weekday	Yes
11	Shower	Hot	Dawn	Weekend	No
12	Shower	Mild	Dawn	Weekday	No
13	Fine	Cool	Dawn	Weekday	No
14	Thunderstorm	Mild	Sunset	Weekend	No
15	Thunderstorm	Hot	Midday	Weekday	No

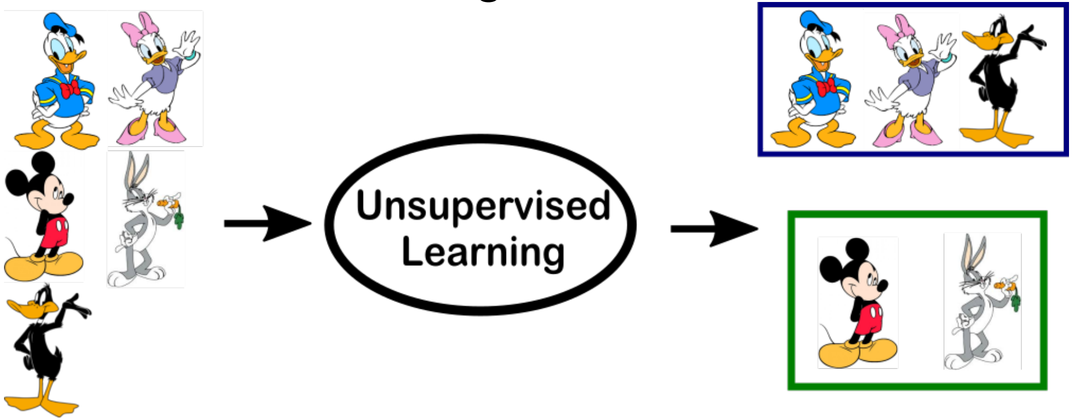
Figure 17.11. Training dataset

Types of Machine Learning: Unsupervised

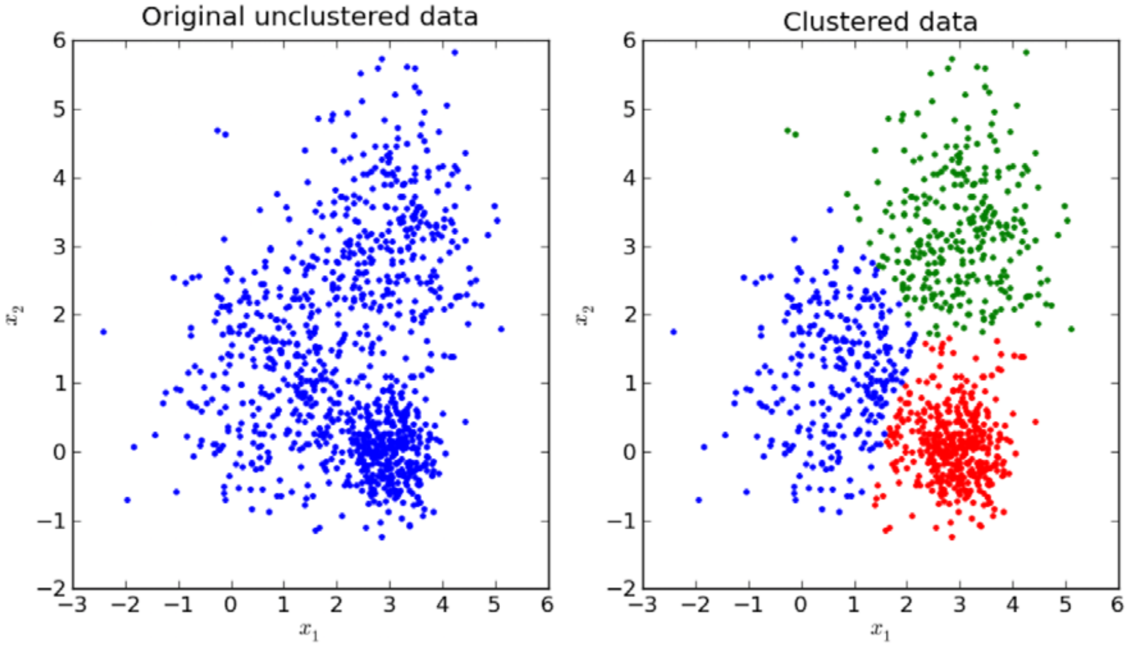
- Instead of predicting a label, unsupervised ML helps you to better understand the structure of your data.

- Two types of unsupervised machine learning:

1. *Clustering* and
2. *Association*.



Unsupervised Machine Learning: Clustering



Clustering example

K-Means clustering

- **Algorithm** k-Means:
  - Specifies *k* number of clusters, and guesses the *k* seed cluster centroid
  - Iteratively looks at each data point and assigns it to the closest centroid
  - Current clusters may receive or loose their members
  - Each cluster must re-calculate the mean (centroid)
  - The process is repeated until the clusters are stable (no change of members)

Algorithm: k-means

```
Input:
    D={x1, x2, …, xn}          //Data objects
    k                          //Number of desired clusters
Output:
    K                          //Set of clusters
1. Assign initial values for means m1, m2, …, mk
2. Repeat
3.    Assign each data object xi to the cluster
      which has the closest mean
4.    Calculate new mean for each cluster
5. Until convergence criteria is met
```

Flux Quiz 10

Data D = {5, 19, 25, 21, 4, 1, 17, 23, 8, 7, 6, 10, 2, 20, 14, 11, 27, 9, 3, 16}
Number of clusters: k = 3
Initial centroids: m1=6, m2=7, and m3=8.

Which of the following grouping is correct after applying K-Means algorithm?

Solution:
C_1={1, 2, 3, 4, 5, 6}
C_2={7, 8, 9, 10, 11, 14}
C_3={16, 17, 19, 20, 21, 23, 25, 27}

k-Means: Step-By-Step Example

- Data D = {5, 19, 25, 21, 4, 1, 17, 23, 8, 7, 6, 10, 2, 20, 14, 11, 27, 9, 3, 16}
- Number of clusters: $k = 3$
- Initial centroids: m_1=6, m_2=7, and m_3=8

- **First Iteration**
  - Clusters:
    - C_1={1, 2, 3, 4, 5, 6}
    - C_2={7}
    - C_3={8, 9, 10, 11, 14, 16, 17, 19, 20, 21, 23, 25, 27}
  - Re-calculated centroids: m_1=3.5, m_2=7, and m_3=16.9

k-Means: Step-By-Step Example

- Clusters:
  - C_1={1, 2, 3, 4, 5, 6}
  - C_2={7}
  - C_3={8, 9, 10, 11, 14, 16, 17, 19, 20, 21, 23, 25, 27}
- New centroids: m_1=3.5, m_2=7, and m_3=16.9
- **Second Iteration**
  - Clusters:
    - C_1={1, 2, 3, 4, 5}
    - C_2={6, 7, 8, 9, 10, 11}
    - C_3={14, 16, 17, 19, 20, 21, 23, 25, 27}
  - Re-calculated centroids: m_1=3, m_2=8.5, and m_3=20.2

k-Means: Step-By-Step Example

- Clusters:
  - C_1={1, 2, 3, 4, 5}
  - C_2={6, 7, 8, 9, 10, 11}
  - C_3={14, 16, 17, 19, 20, 21, 23, 25, 27}
- New centroids: m_1=3, m_2=8.5, and m_3=20.2
- **Third Iteration**
  - Clusters:
    - C_1={1, 2, 3, 4, 5}
    - C_2={6, 7, 8, 9, 10, 11, 14}
    - C_3={16, 17, 19, 20, 21, 23, 25, 27}
  - Re-calculated centroids: m_1=3, m_2=9.29, and m_3=21

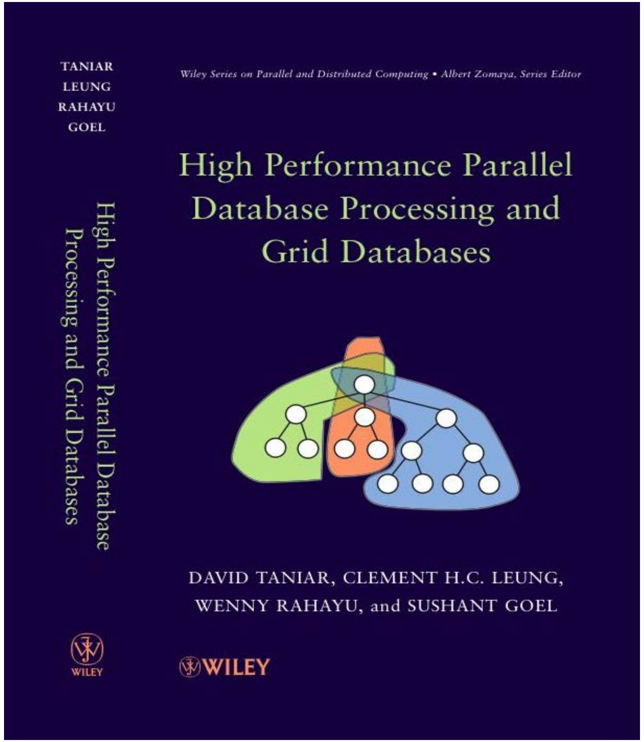
k-Means: Step-By-Step Example

- Clusters:
  - C_1={1, 2, 3, 4, 5}
  - C_2={6, 7, 8, 9, 10, 11, 14}
  - C_3={16, 17, 19, 20, 21, 23, 25, 27}
- New centroids: m_1=3, m_2=9.29, and m_3=21
- **Fourth Iteration**
  - Clusters:
    - C_1={1, 2, 3, 4, 5, 6}
    - C_2={7, 8, 9, 10, 11, 14}
    - C_3={16, 17, 19, 20, 21, 23, 25, 27}
  - Re-calculated centroids: m_1=3.5, m_2=9.83, and m_3=21

k-Means: Step-By-Step Example

- Clusters:
  - C_1={1, 2, 3, 4, 5, 6}
  - C_2={7, 8, 9, 10, 11, 14}
  - C_3={16, 17, 19, 20, 21, 23, 25, 27}
- New centroids: m_1=3.5, m_2=9.83, and m_3=21
- **Fifth Iteration**
  - No data movement from clusters (Process Terminated)

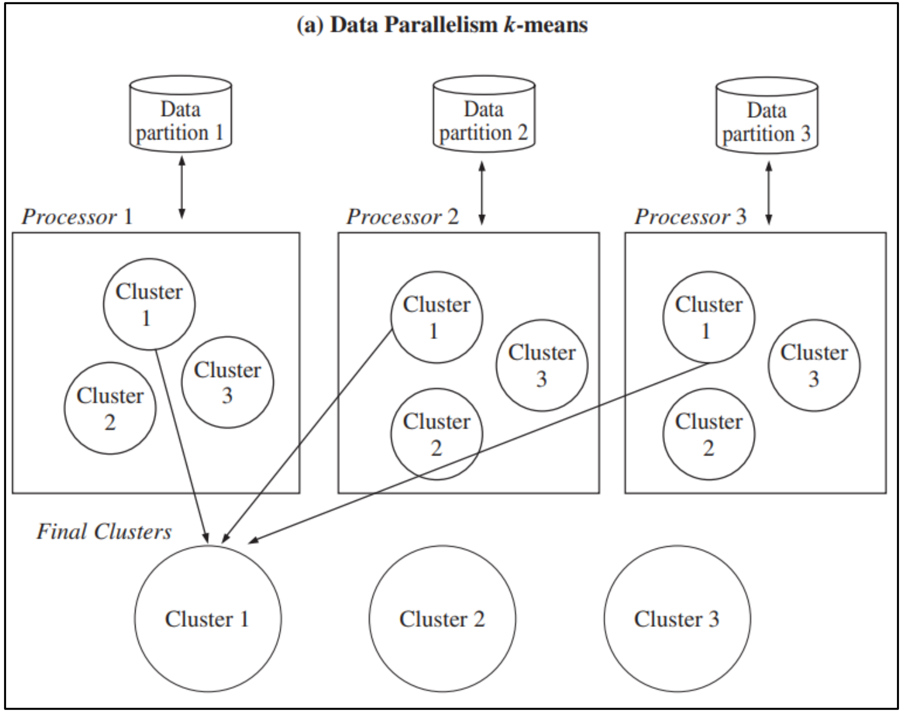
m_1	m_2	m_3	C_1	C_2	C_3
6	7	8	1, 2, 3, 4, 5, 6	7	8, 9, 10, 11, 14, 16, 17, 19, 20, 23, 25, 27
3.5	7	16.9	1, 2, 3, 4, 5	6, 7, 8, 9, 10, 11	14, 16, 17, 19, 20, 21, 23, 25, 27
3	8.5	20.2	1, 2, 3, 4, 5	6, 7, 8, 9, 10, 11, 14	16, 17, 19, 20, 21, 23, 25, 27
3	9.29	21	1, 2, 3, 4, 5, 6	7, 8, 9, 10, 11, 14	16, 17, 19, 20, 21, 23, 25, 27
3.5	9.83	21	1, 2, 3, 4, 5, 6	7, 8, 9, 10, 11, 14	16, 17, 19, 20, 21, 23, 25, 27



Chapter 17
Parallel Clustering and Classification

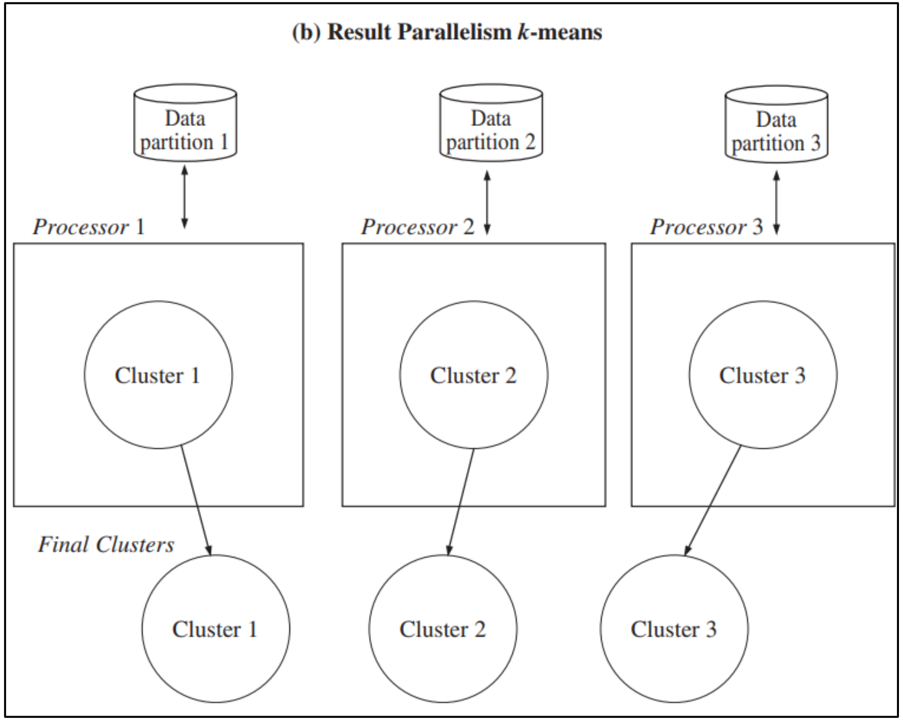
Parallel K-means clustering

- *Data parallelism* of k-means



Parallel K-means clustering

- ***Result Parallelism*** of k-means



Product Recommendation

According to McKinsey study, 35% of what consumers purchase on Amazon and 75% of what they watch on Netflix is driven by machine learning–based product recommendations.

Flux Quiz 11

Collaboration Filtering: Walkthrough Example

Name	Star Trek	Star wars	Superman	Batman	Hulk
Harry	4	2	?	5	4
John	5	3	4	?	3
Rob	3	?	4	4	3

Aim: Recommend top-2 movies to Harry

Collaborative Filtering Process

Data Collection -> Data Processing -> Calculate Referrals -> Derive Results

- Data collection: Collecting user behaviour and associated data items
- Data processing: Processing the collected data
- Recommendation Calculation: The recommended calculation method used to calculate referrals
- Derive the result: Extract the similarity, sort it, and extract the top N to complete

Step 1: Calculate the similarity between Harry and all other users

Name	Star Trek	Star wars	Superman	Batman	Hulk
Harry	4	2	?	5	4
John	5	3	4	?	3
Rob	3	?	4	4	3

Cosine similarity

$$sim(u,u') = cos(\theta) = \frac{\mathbf{r}_u \cdot \mathbf{r}_{u'}}{\|\mathbf{r}_u\| \|\mathbf{r}_{u'}\|} = \sum_i \frac{r_{ui} r_{u'i}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{u'i}^2}}$$

Step 1: Calculate the similarity between Harry and all other users

Name	Star Trek	Star wars	Superman	Batman	Hulk
Harry	4	2	?	5	4
John	5	3	4	?	3
Rob	3	?	4	4	3

Cosine similarity

Sim(Harry, John) = $\frac{(4*5)+(2*3)+(4*3)}{sqrt(4^2+2^2+4^2)*sqrt(5^2+3^2+3^2)}$

$= 0.97$

Sim(Harry, Rob) = $\frac{(4*3)+(5*4)+(4*3)}{sqrt(4^2+5^2+4^2)*sqrt(3^2+4^2+3^2)}$

$= 1.00$

Step 2: Predict the ratings of movies for Harry

Name	Star Trek	Star wars	Superman	Batman	Hulk
Harry	4	2	?	5	4
John	5	3	4	?	3
Rob	3	?	4	4	3

Calculate k as a normalising factor

k = $\frac{1}{(0.97+1)}$ = 0.51

R(Harry, Superman) = k*$((sim(Harry,\ John) * R(John, Superman)) + (sim(Harry, Rob) * R(Rob, Superman)))$

$= 0.51((0.97 * 4) + (1 * 4))$

$= 4.02$

Step 3: Select top-2 rated movies for Harry

Name	Star Trek	Star wars	Superman	Batman	Hulk
Harry	4	2	***4.02***	5	4
John	5	3	4	?	3
Rob	3	?	4	4	3

Top-2(Harry, movies)= Batman, Superman

Unit Overview

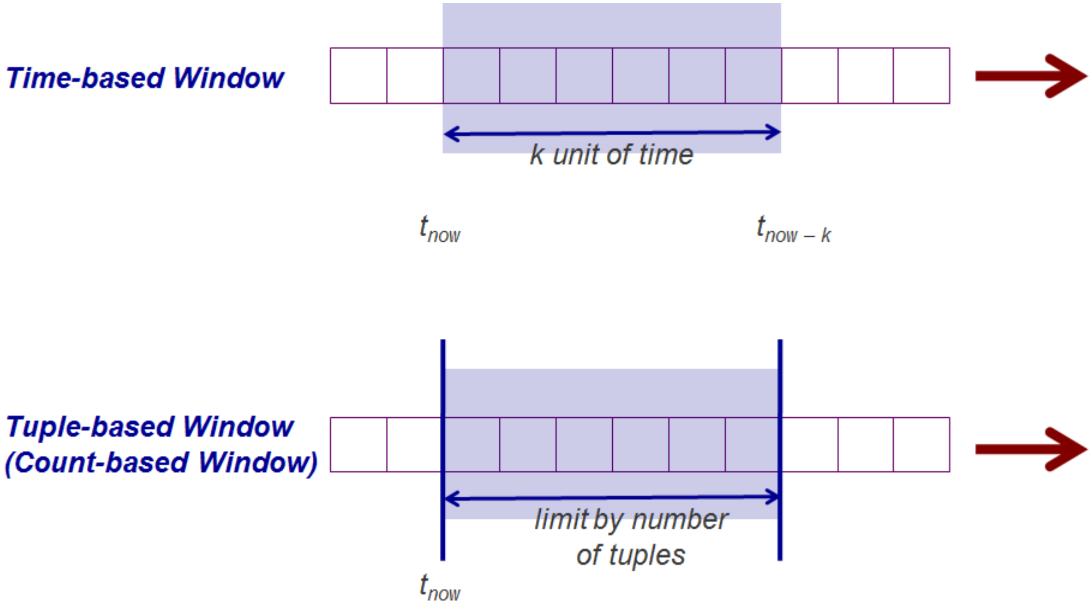
1. **Volume** ➔ Sessions 1, 2, 3, 4
   – How to process Big Data Volume?

2. **Complexity** ➔ Sessions 5, 6, 7, 8
   – How to apply machine learning algorithms to every aspect of Big Data?

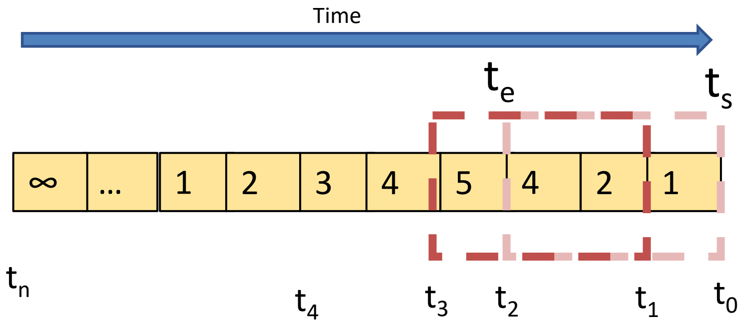
3. **Velocity** ➔ Sessions 9, 10, 11
   – How to handle and process Fast Streaming Data?

Windowing System in Unbounded Streams

A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items.



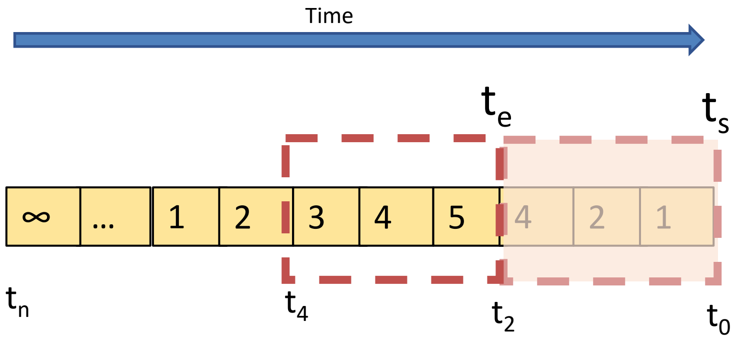
Stream Window – Time Based Window



Overlapping Sliding Window

Window size 2 seconds
Slides 1 second.

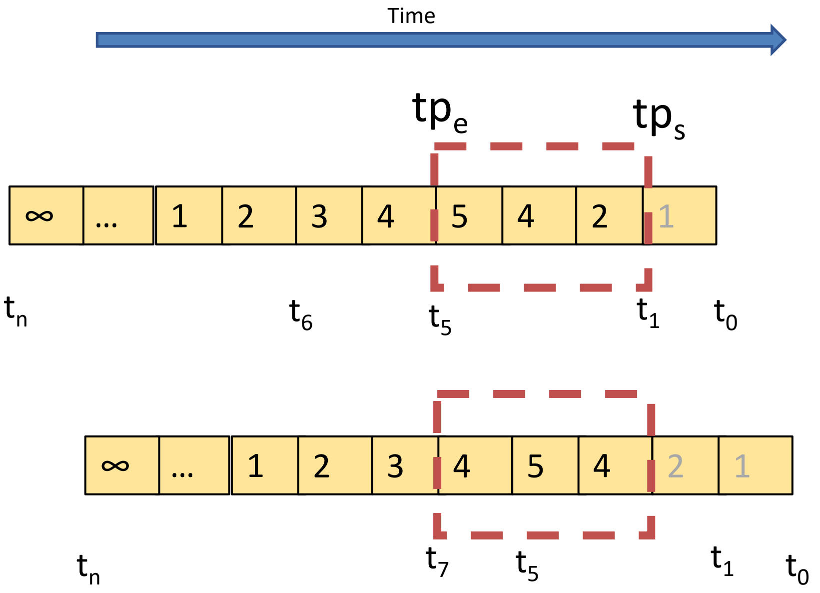
Stream Window – Time Based Window



Non-Overlapping Sliding Window

- Window size is based on time, eg 2 seconds
- Window can be advanced by:
    - t_e-t_s (window size)
    - Duration less than the window size (sliding window).
- In uniform data rate, the number of tuples will be the same for each window.

Stream Window – Tuple Based Window



Window size is 3 tuples
Slide window by 1 tuple

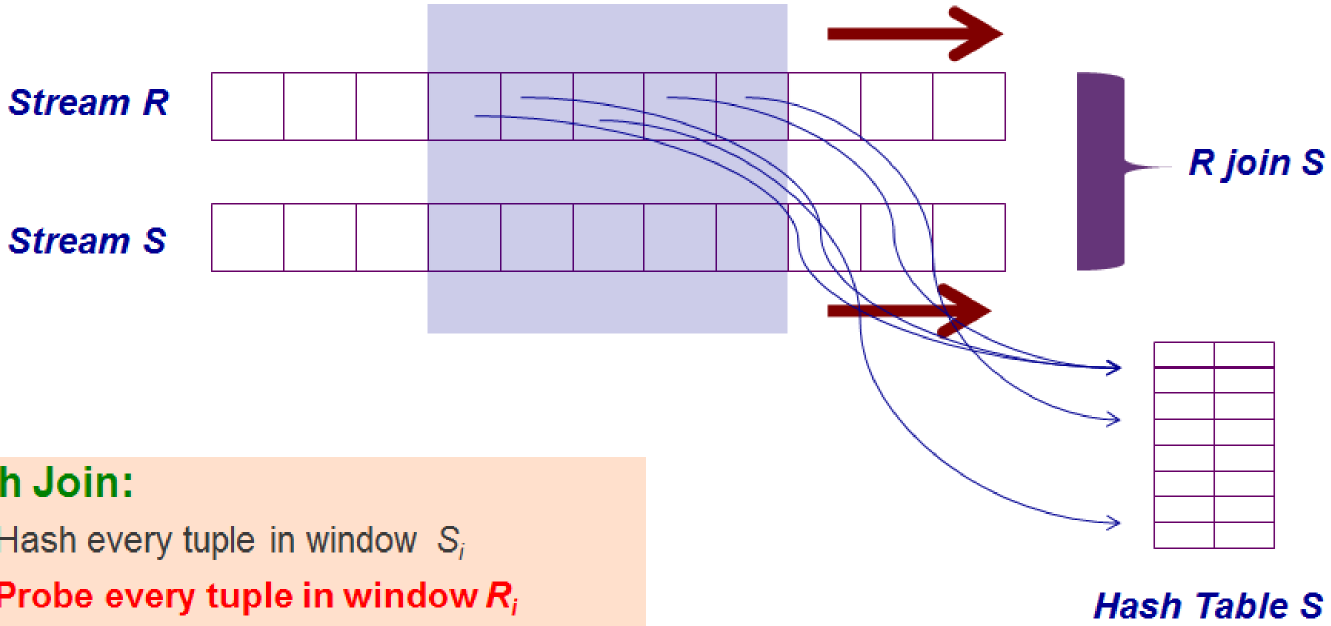
Event vs Processing time

- Event time: the time when the data is produced by the source.
- Processing time: the time when data is arrived at the processing server.
- In ideal situation, event time = processing time.
- In real world event time is earlier than the processing time due to network delay.
- The delay can be uniformed (ideal situation) or non-uniform (most of real network situation).
- Data may arrive in “burst” (bursty network).

Joins In Data Streams

- Symmetric Hash Join
- M Join
- AM Join
- Handshake Join

Hash Join

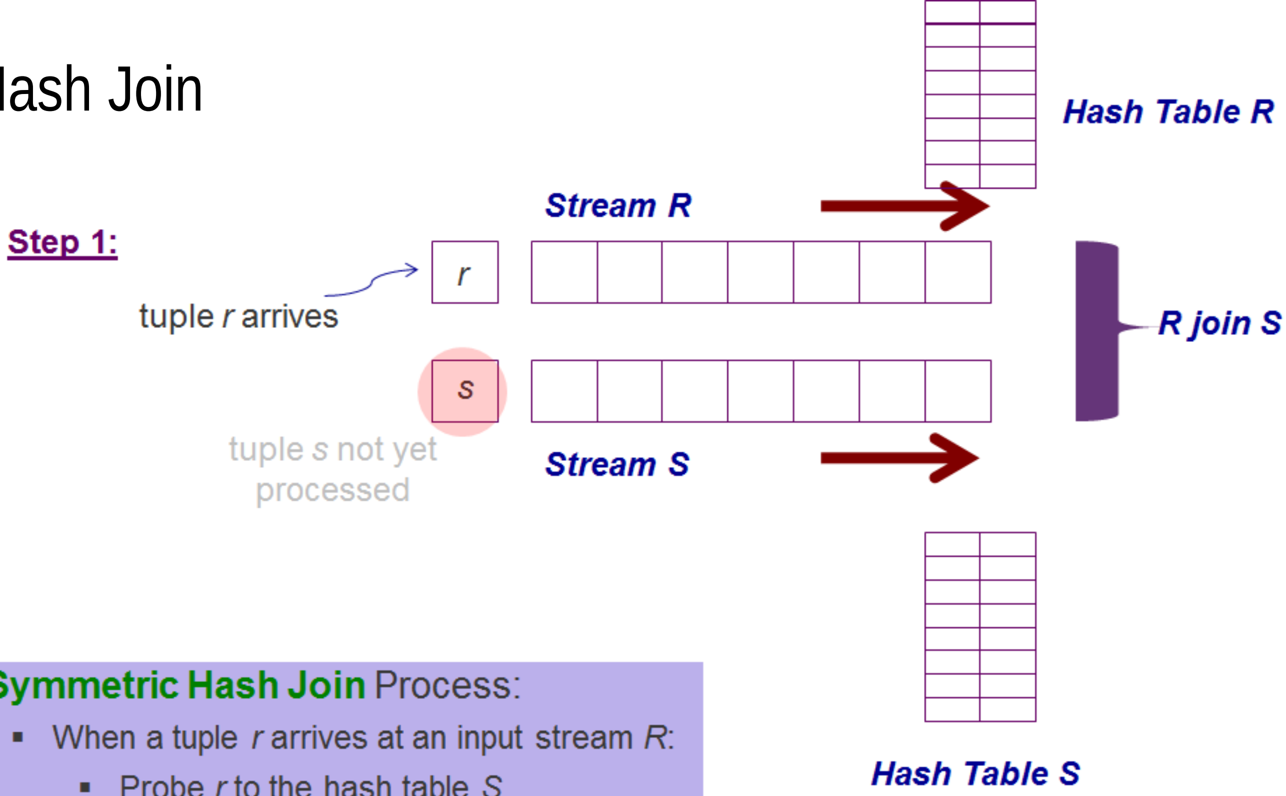


Hash Join:

- Hash every tuple in window S_i
- **Probe every tuple in window R_i**

What's the problem?

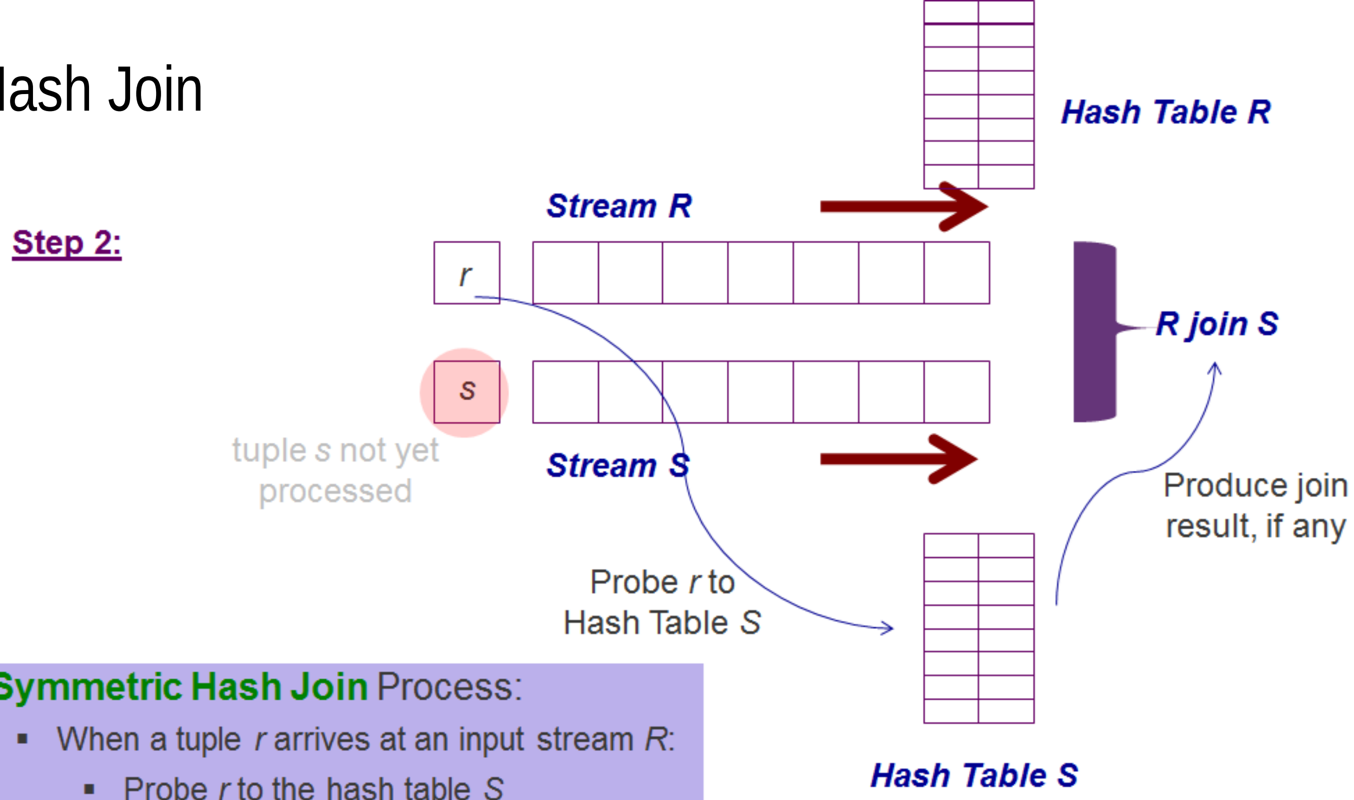
Symmetric Hash Join



Symmetric Hash Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - Probe *r* to the hash table *S*
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

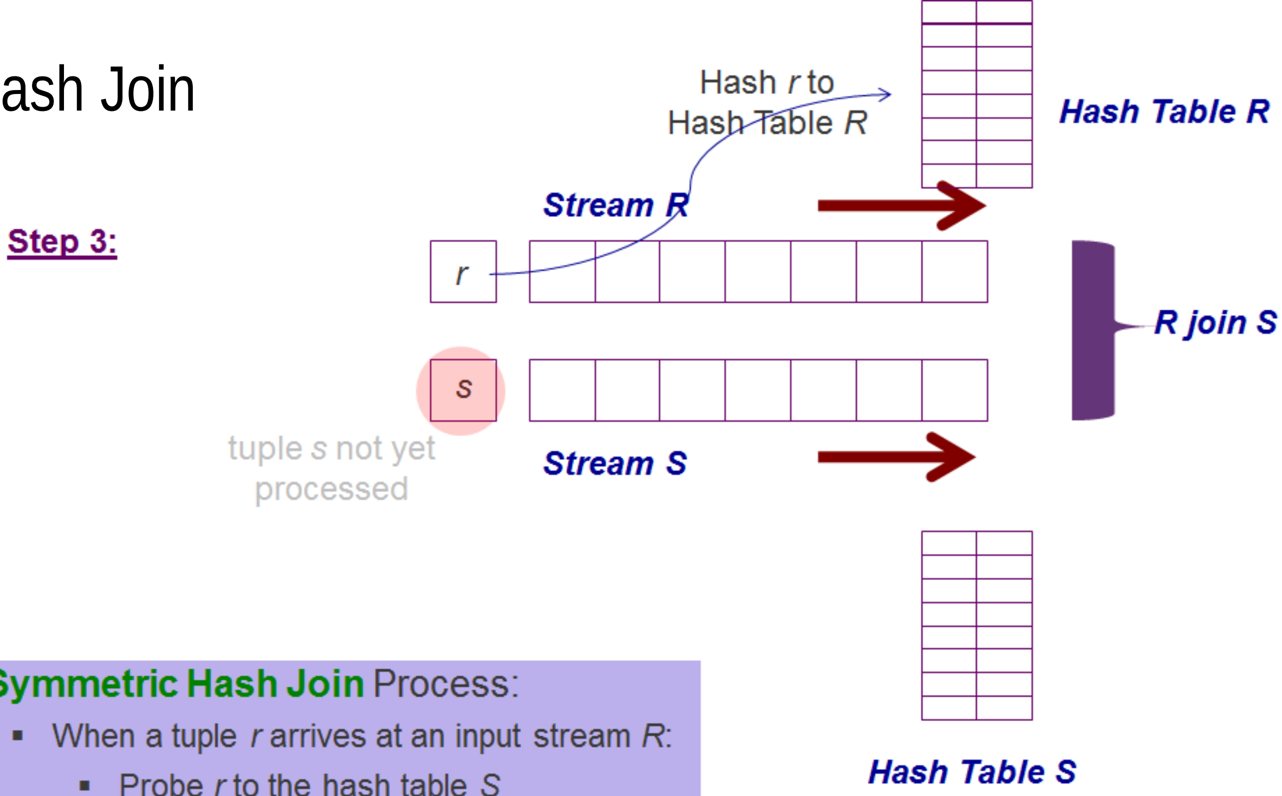
Symmetric Hash Join



Symmetric Hash Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - Probe *r* to the hash table *S*
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

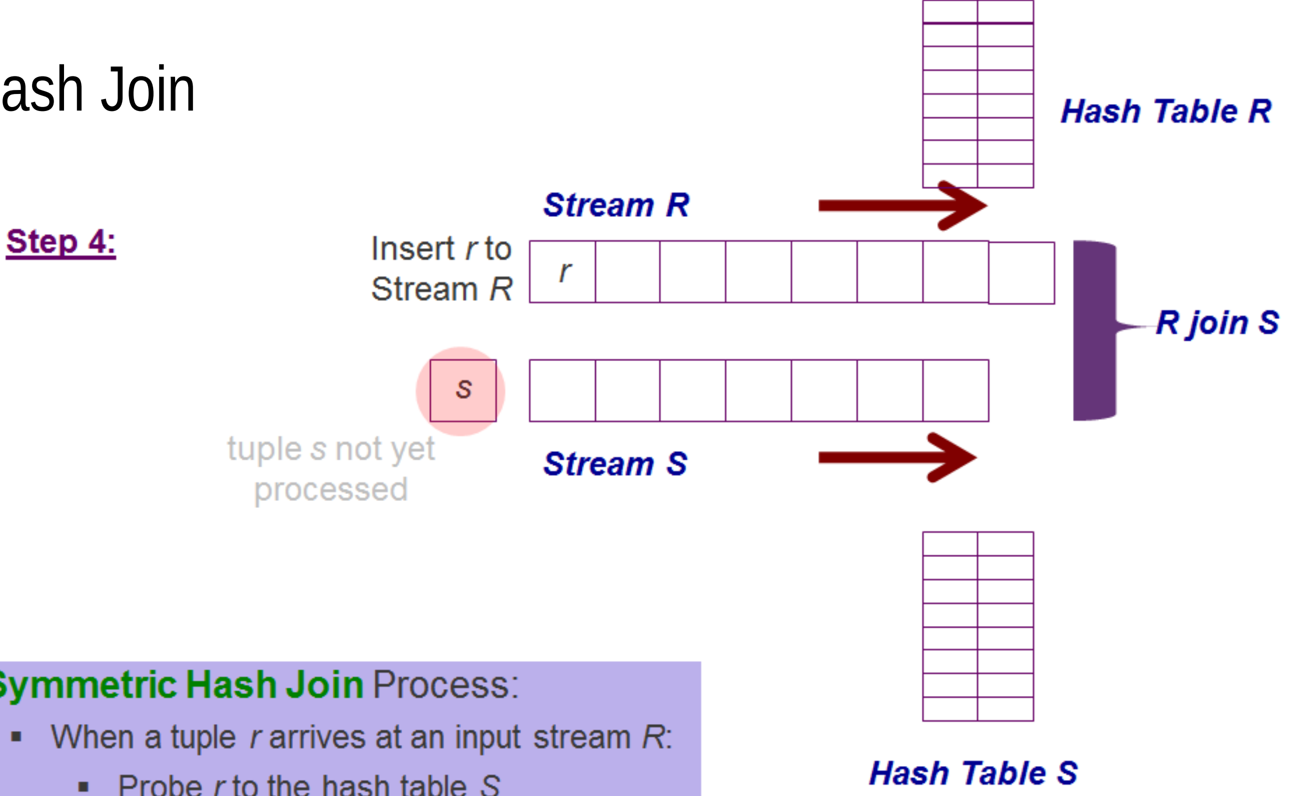
Symmetric Hash Join



Symmetric Hash Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - Probe *r* to the hash table *S*
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

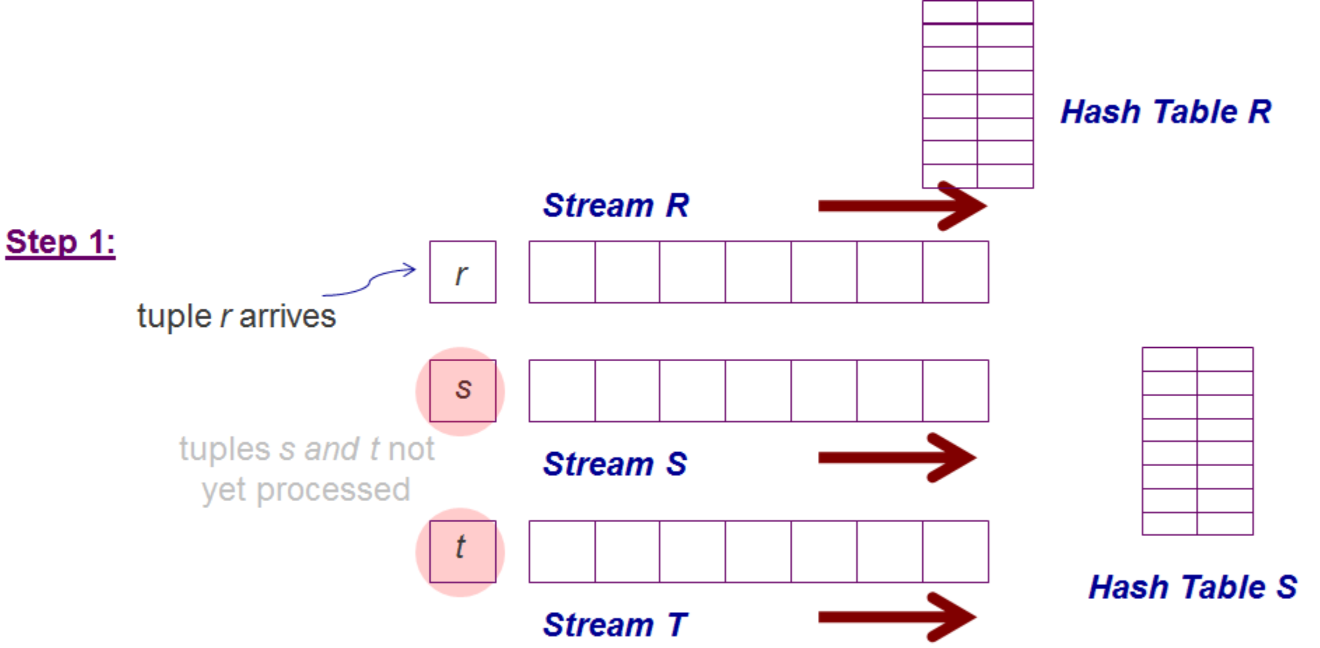
Symmetric Hash Join



Symmetric Hash Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - Probe *r* to the hash table *S*
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

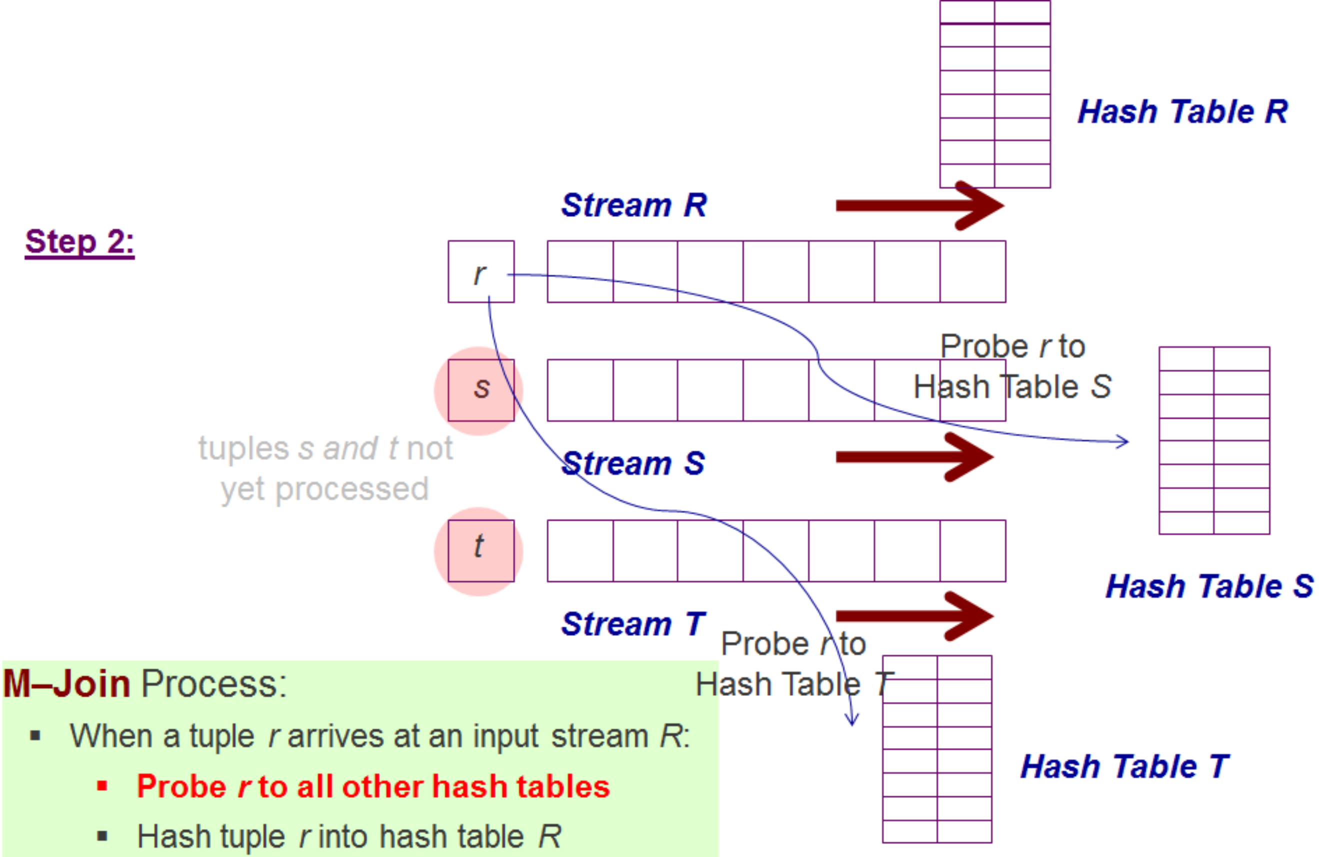
M-Join



M–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to all other hash tables**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

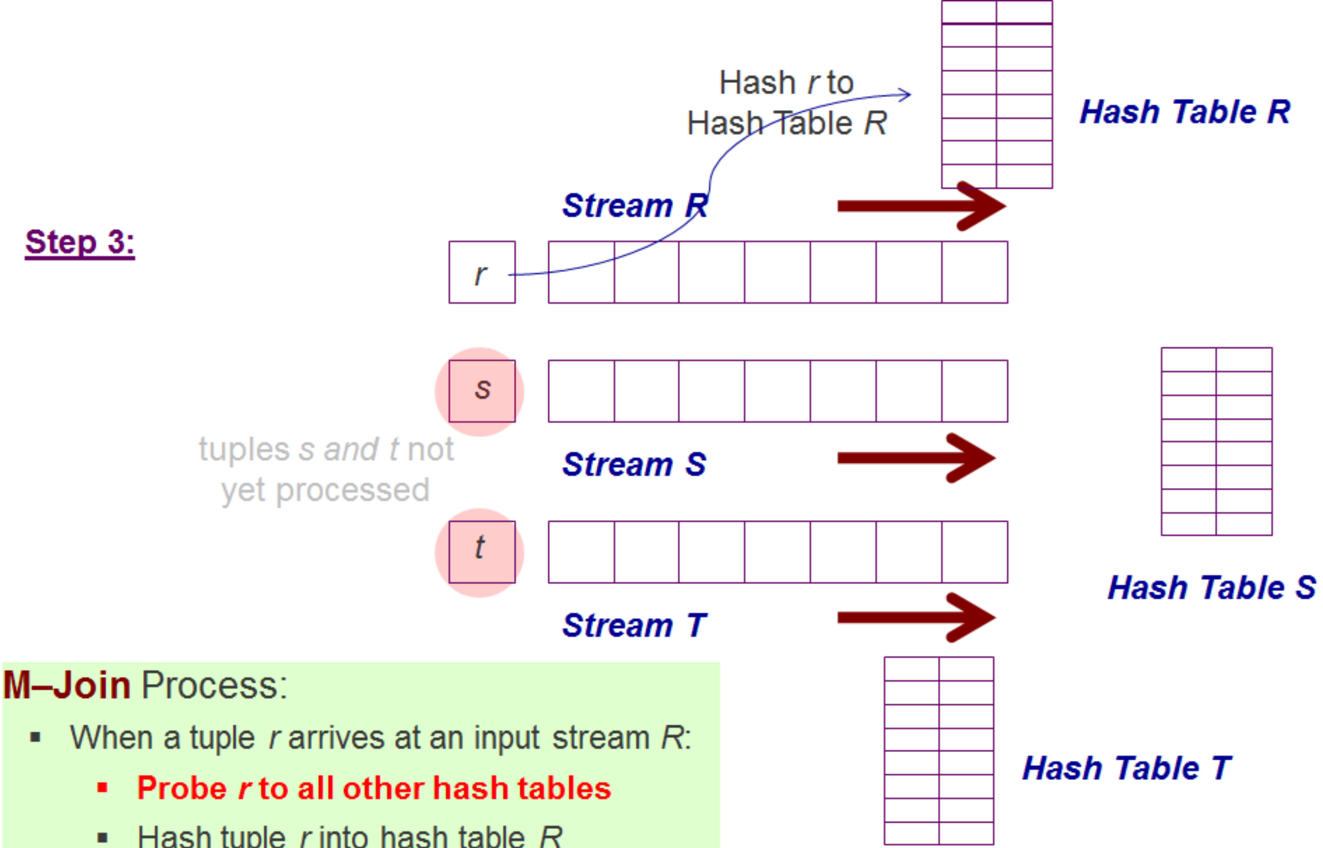
M-Join



M–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to all other hash tables**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

M-Join

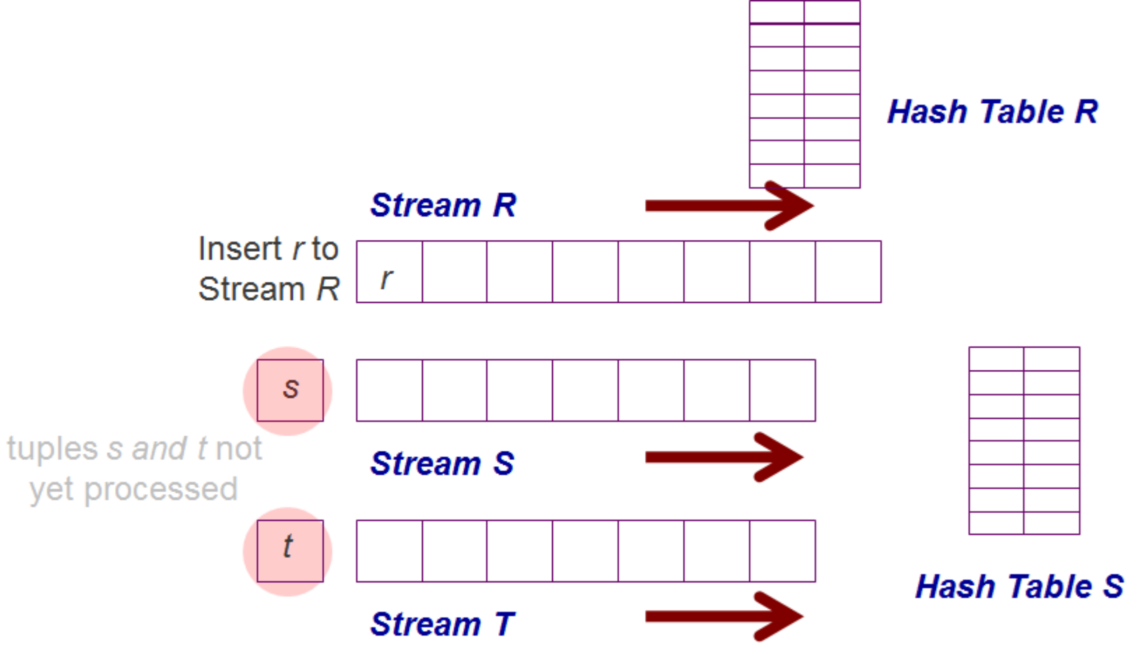


M–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to all other hash tables**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

M-Join

Step 3:

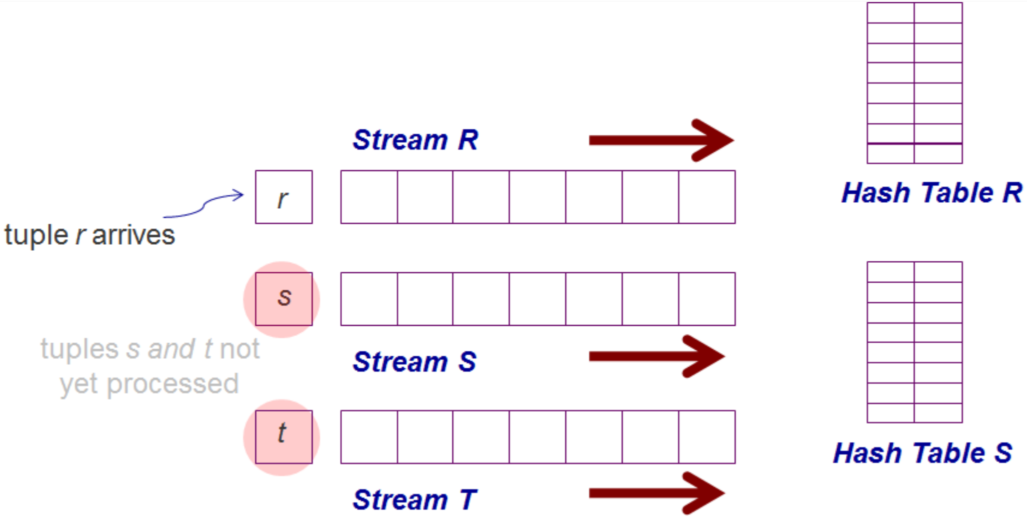


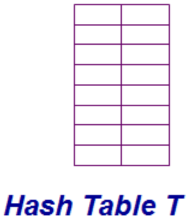
M–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to all other hash tables**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

AM-Join

Step 1:

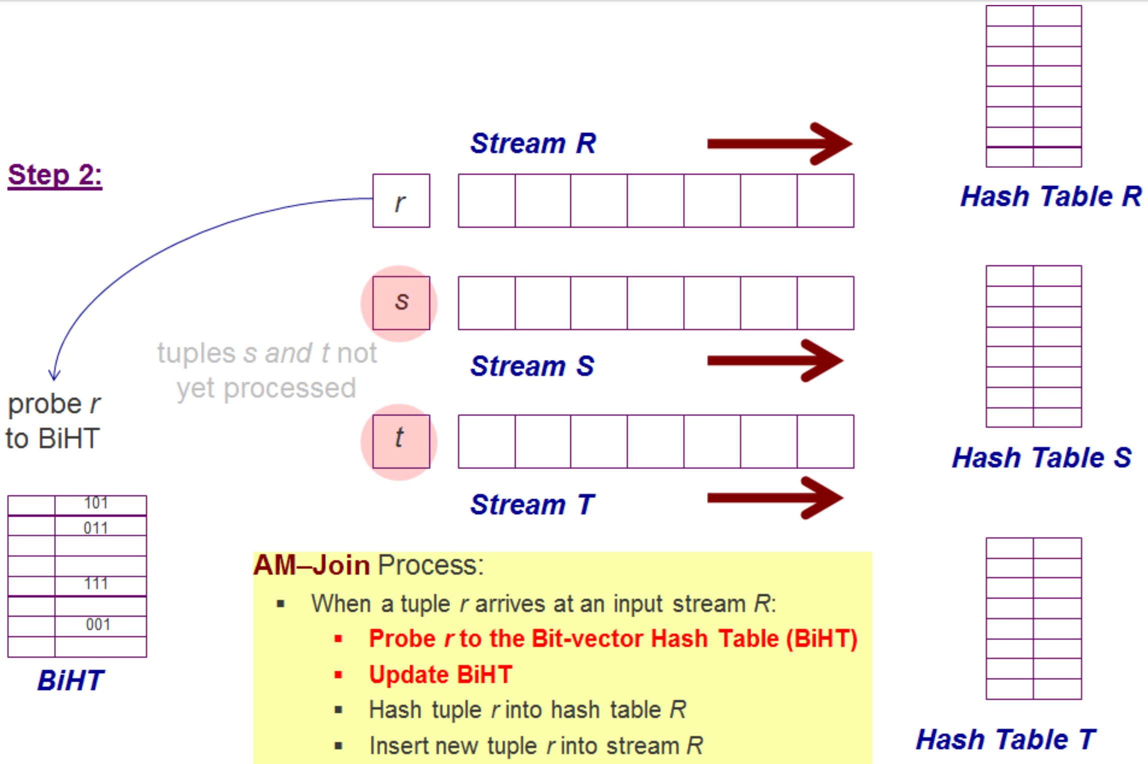




AM–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to the Bit-vector Hash Table (BiHT)**
  - **Update BiHT**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

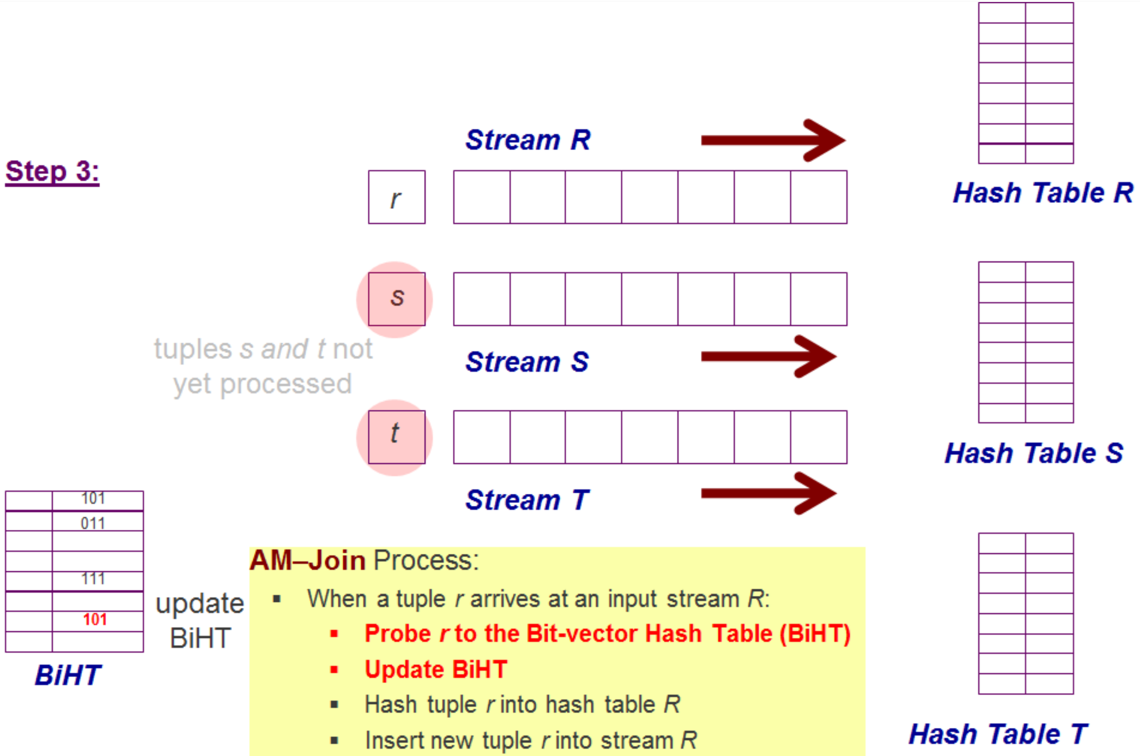
AM-Join



AM–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to the Bit-vector Hash Table (BiHT)**
  - **Update BiHT**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

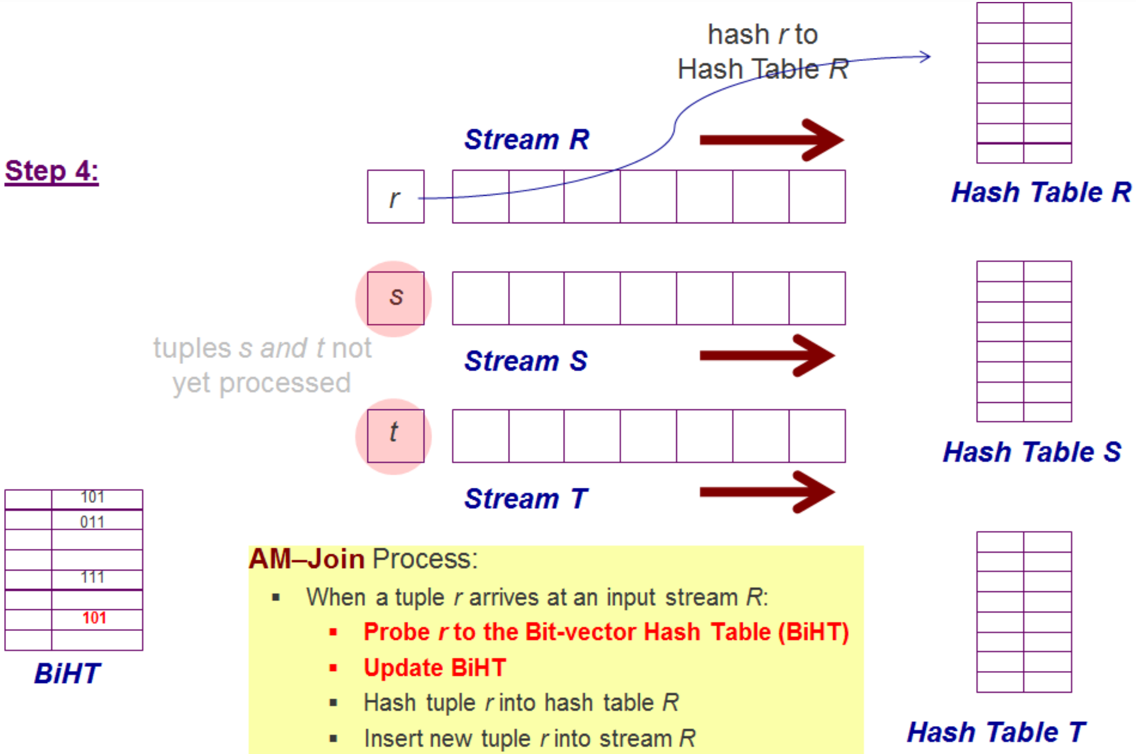
AM-Join



AM–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to the Bit-vector Hash Table (BiHT)**
  - **Update BiHT**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

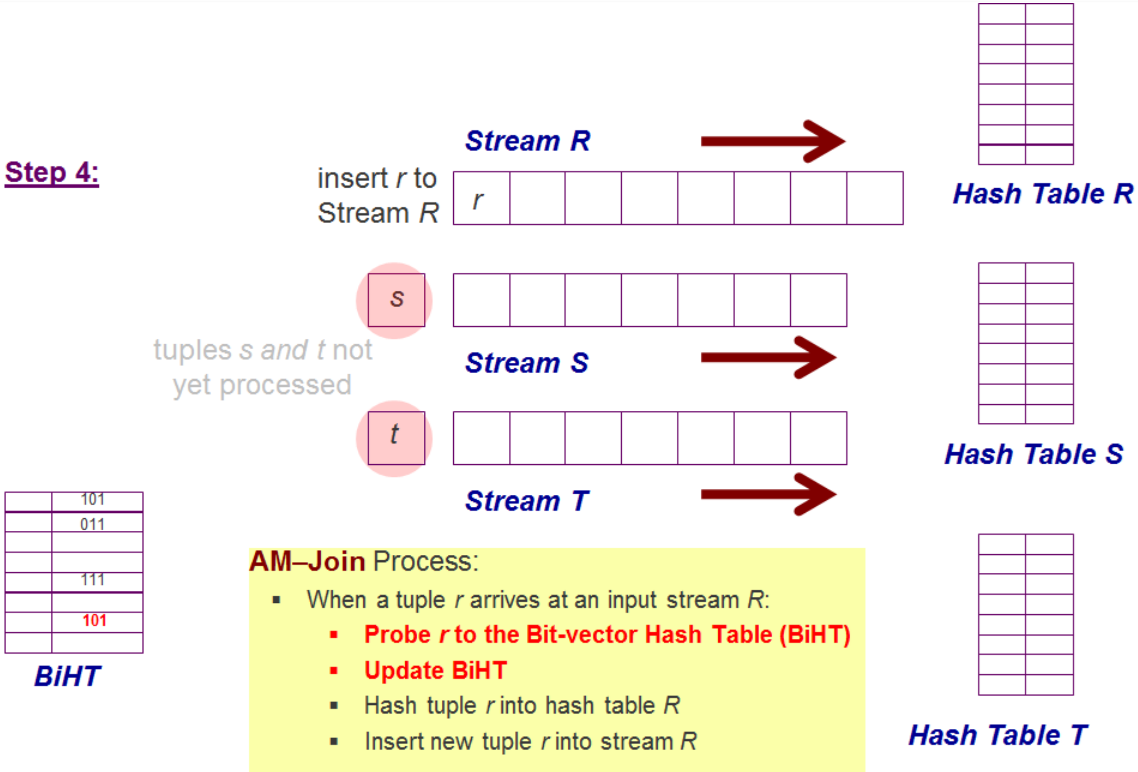
AM-Join



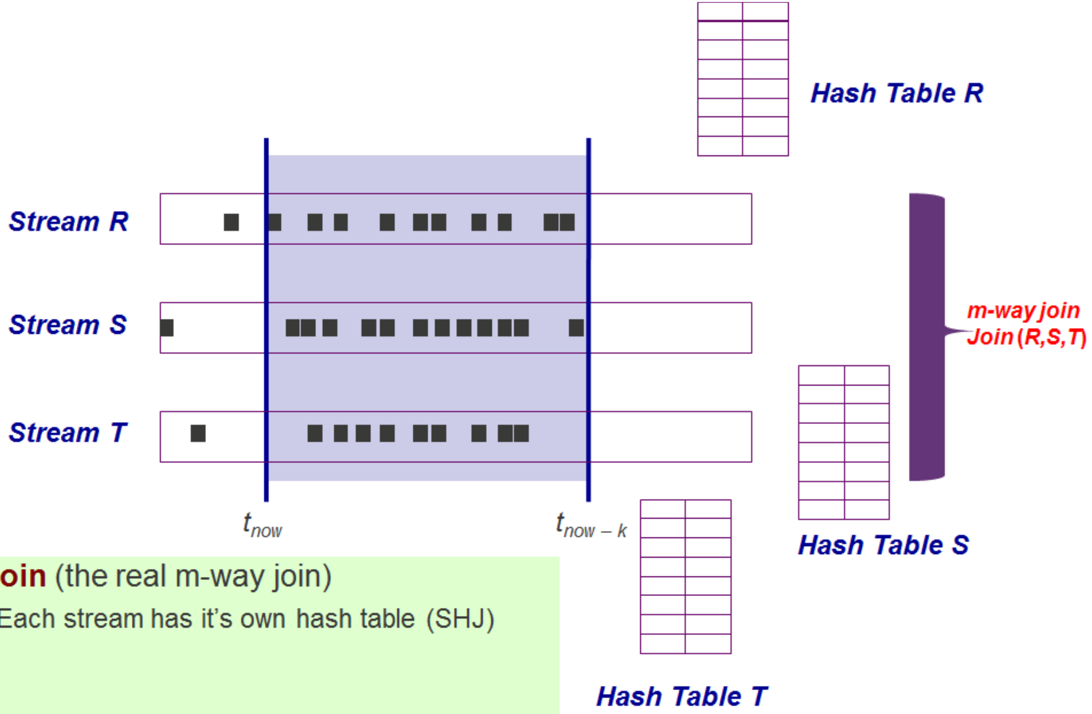
AM–Join Process:

- When a tuple *r* arrives at an input stream *R*:
  - **Probe *r* to the Bit-vector Hash Table (BiHT)**
  - **Update BiHT**
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

AM-Join


Step 4:
Stream R
insert r to Stream R
r
Hash Table R
s
tuples s and t not yet processed
Stream S
Hash Table S
t
Stream T
101
011
111
101
BiHT
AM–Join Process:
When a tuple r arrives at an input stream R:
Probe r to the Bit-vector Hash Table (BiHT)
Update BiHT
Hash tuple r into hash table R
Insert new tuple r into stream R
Hash Table T

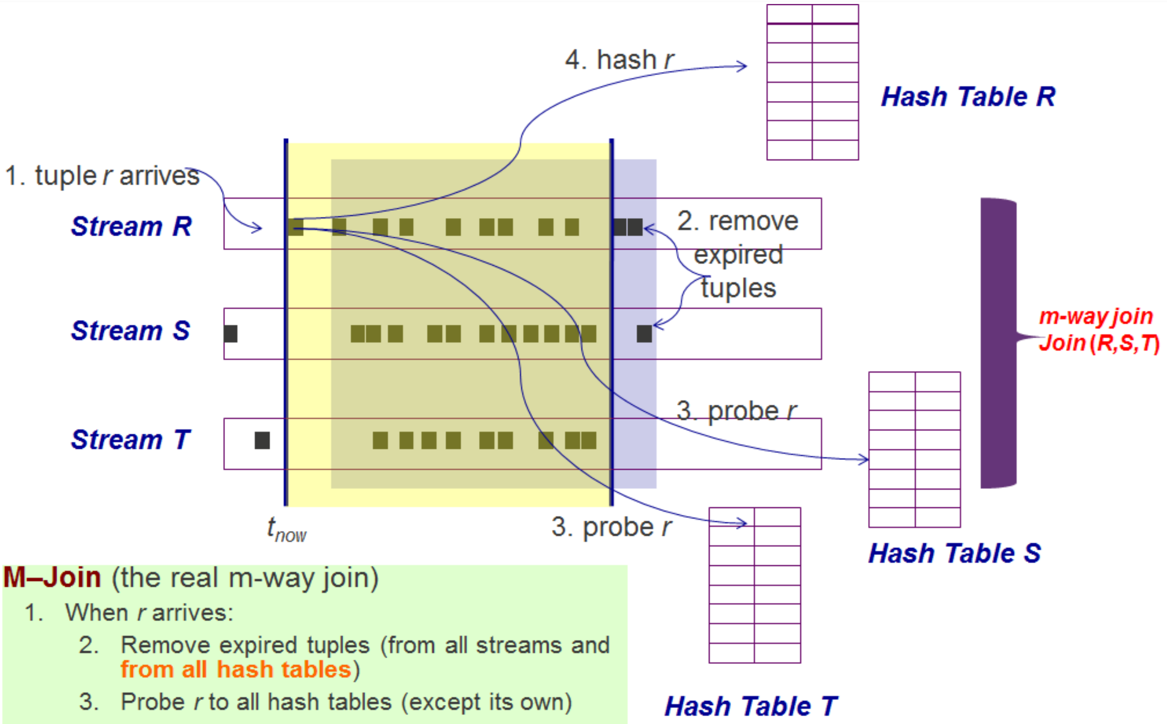
Tuple Slide (Using M–Join)



M–Join (the real m-way join)

- Each stream has it's own hash table (SHJ)

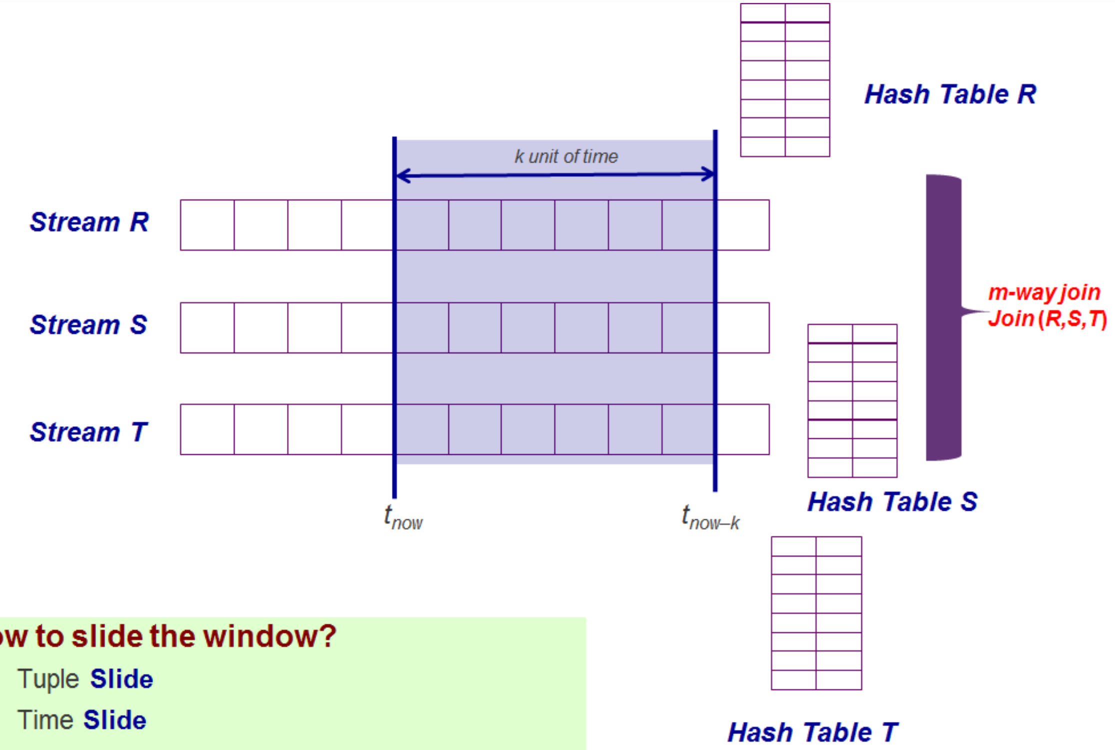
Tuple Slide (Using M–Join)



M–Join (the real m-way join)

1. When *r* arrives:
    2. Remove expired tuples (from all streams and **from all hash tables**)
    3. Probe *r* to all hash tables (except its own)
    4. Hash *r* to hash table *R*

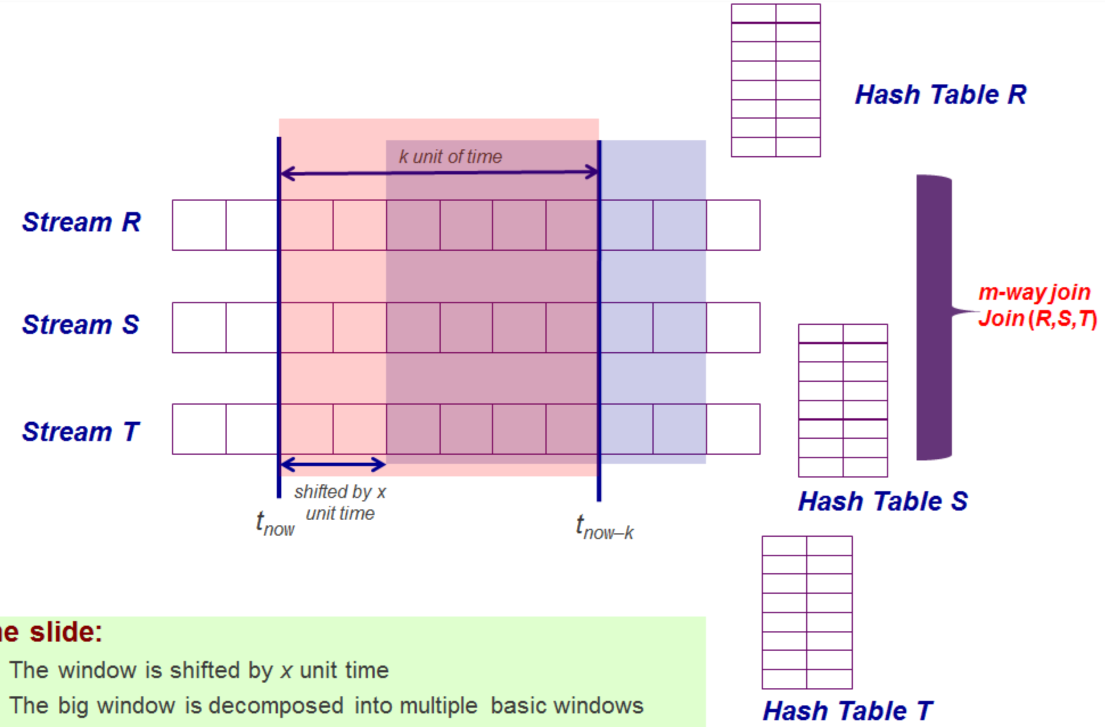
Time Slide (Using M-Join)



How to slide the window?

- Tuple **Slide**
- Time **Slide**

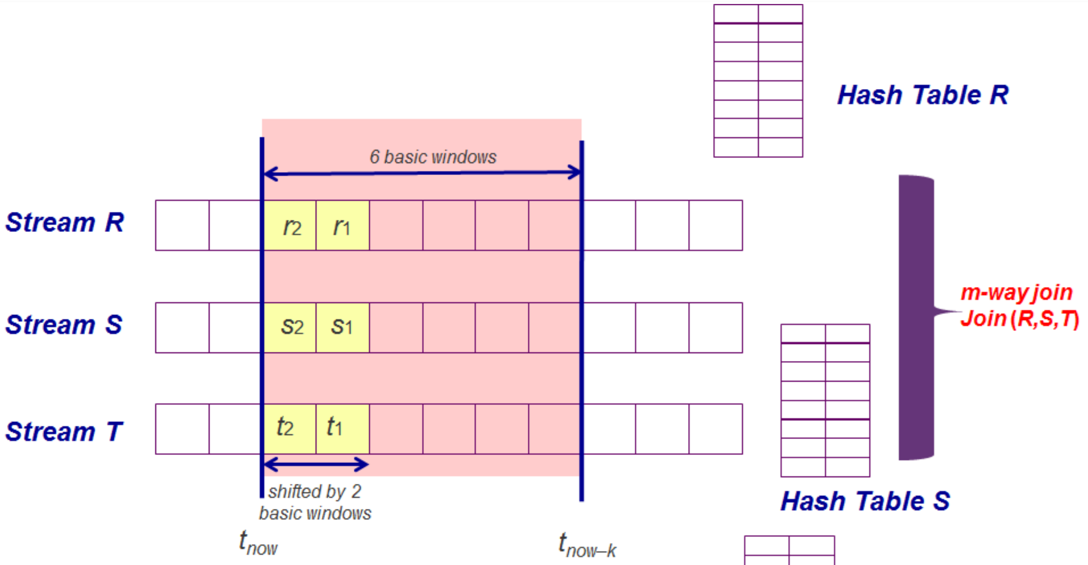
Time Slide (Using M-Join)



Time slide:

- The window is shifted by *x* unit time
- The big window is decomposed into multiple basic windows
- The big window is then **shifted by 1 or more basic windows**

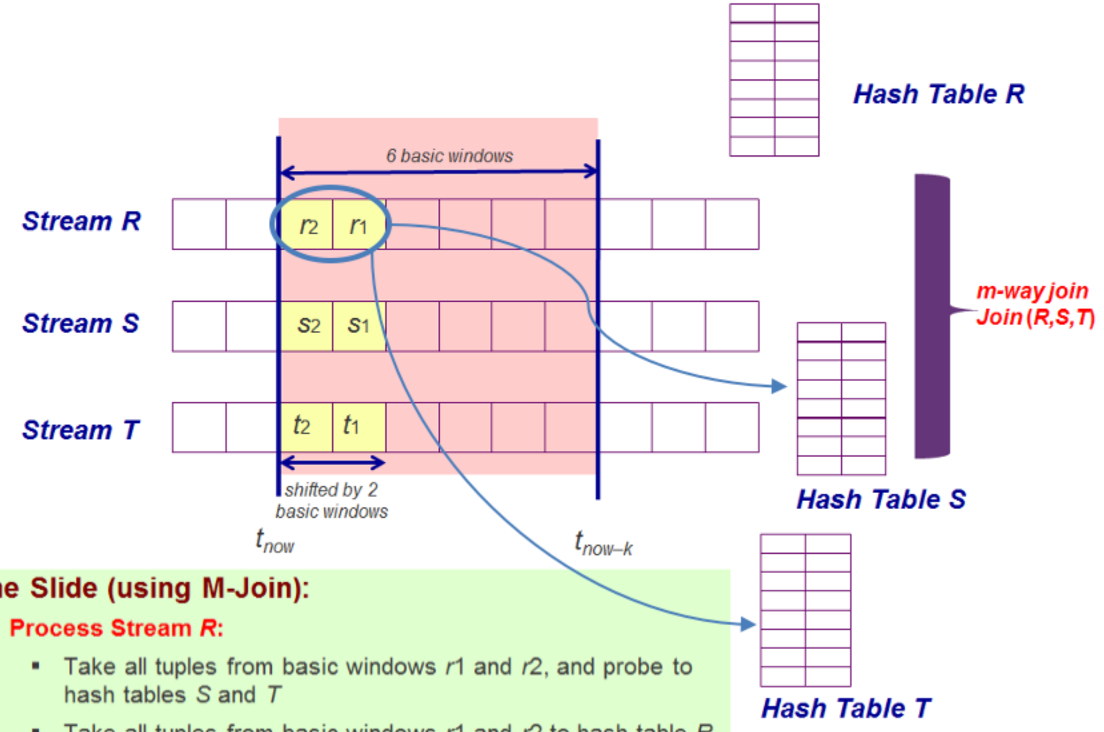
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **We use the current window** (assume the previous expired basic windows have been removed as part of the previous join process)
- In the above example, the latest 2 basic windows are new basic windows
- Note that the **hash tables do not yet have tuples from these 2 basic windows**

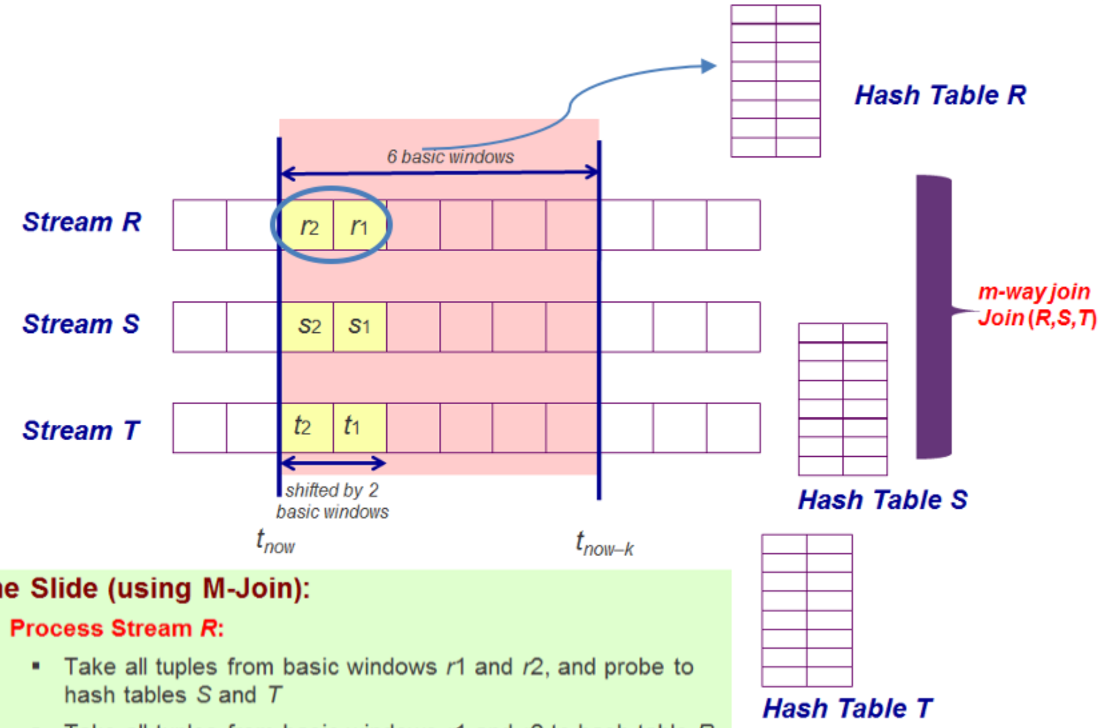
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *R*:**
  - Take all tuples from basic windows *r*1 and *r*2, and probe to hash tables *S* and *T*
  - Take all tuples from basic windows *r*1 and *r*2 to hash table *R*

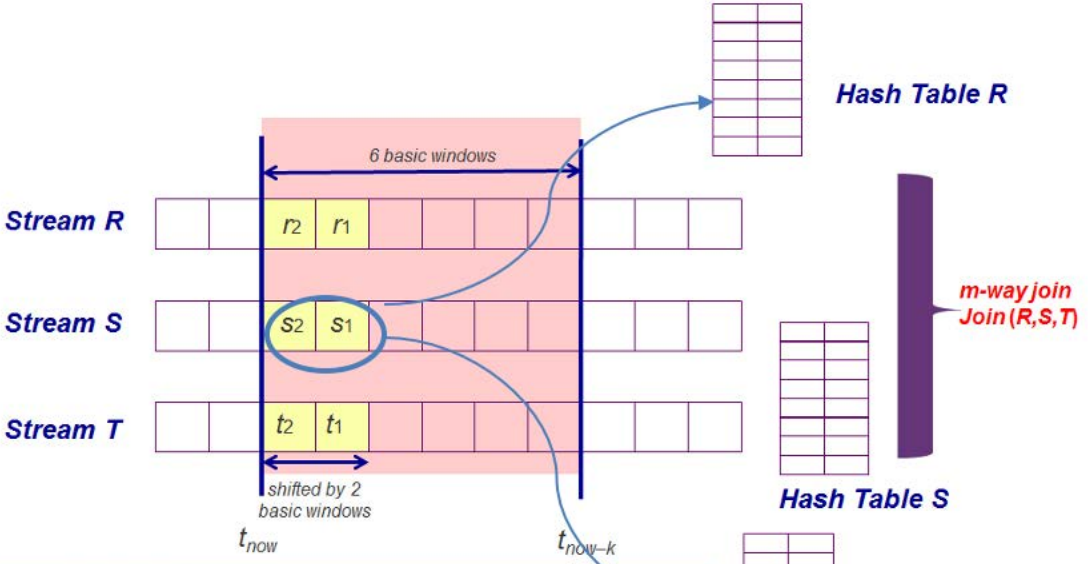
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *R*:**
  - Take all tuples from basic windows *r*1 and *r*2, and probe to hash tables *S* and *T*
  - Take all tuples from basic windows *r*1 and *r*2 to hash table *R*

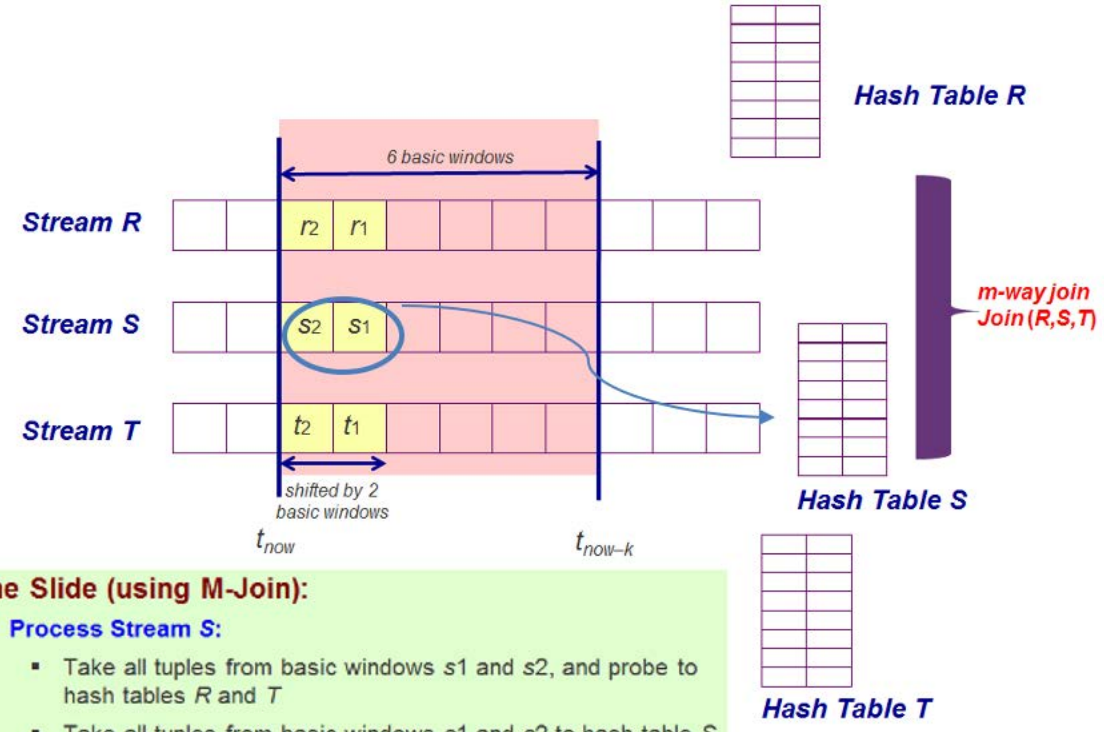
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *S*:**
  - Take all tuples from basic windows $s1$ and $s2$, and probe to hash tables R and T
  - Take all tuples from basic windows $s1$ and $s2$ to hash table S

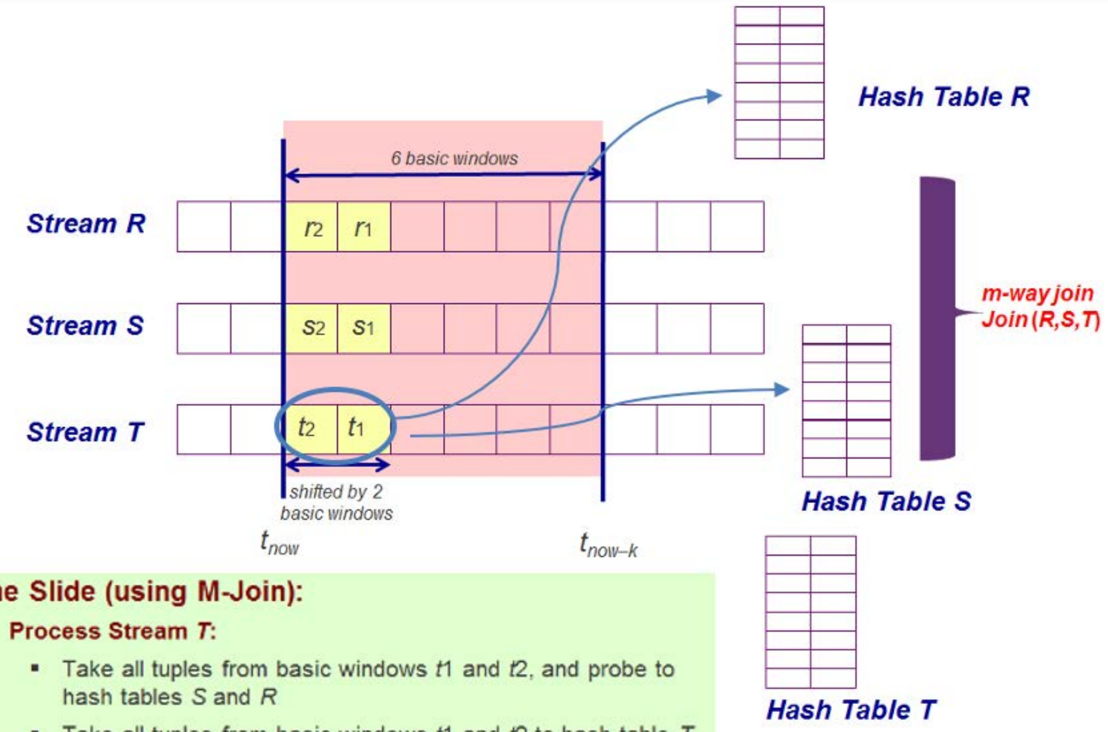
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *S*:**
  - Take all tuples from basic windows *s*1 and *s*2, and probe to hash tables *R* and *T*
  - Take all tuples from basic windows *s*1 and *s*2 to hash table *S*

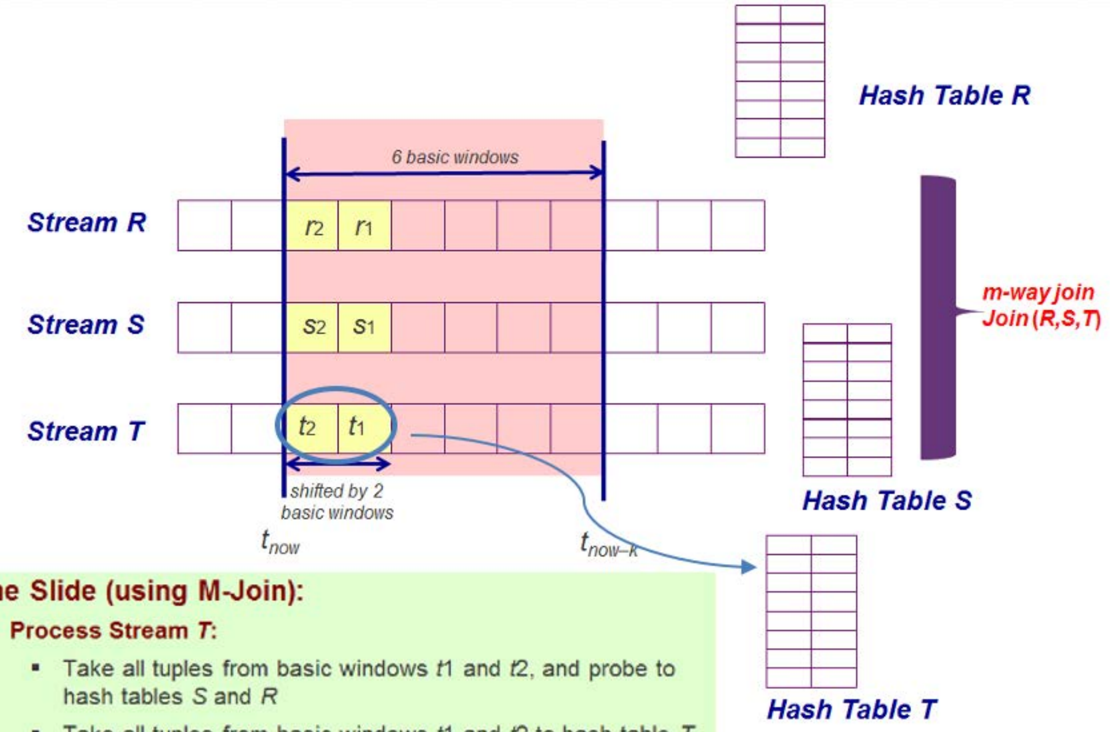
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *T*:**
  - Take all tuples from basic windows *t1* and *t2*, and probe to hash tables *S* and *R*
  - Take all tuples from basic windows *t1* and *t2* to hash table *T*

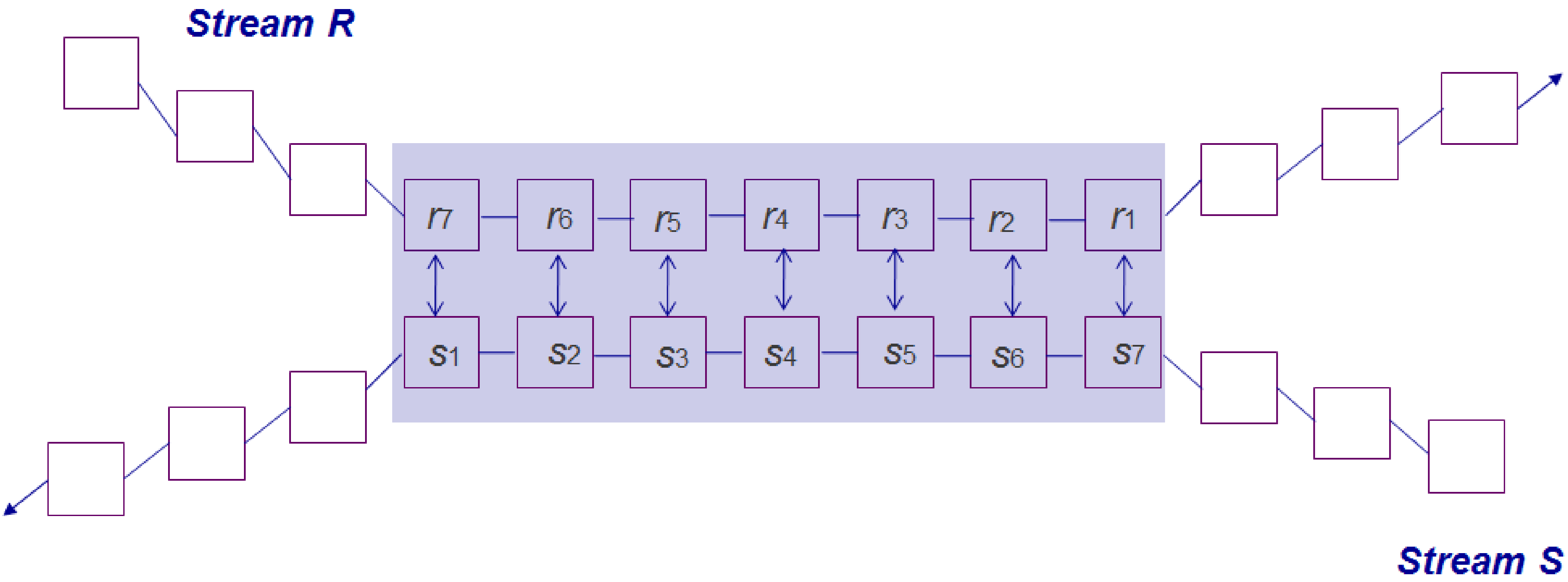
Time Slide (Using M-Join)



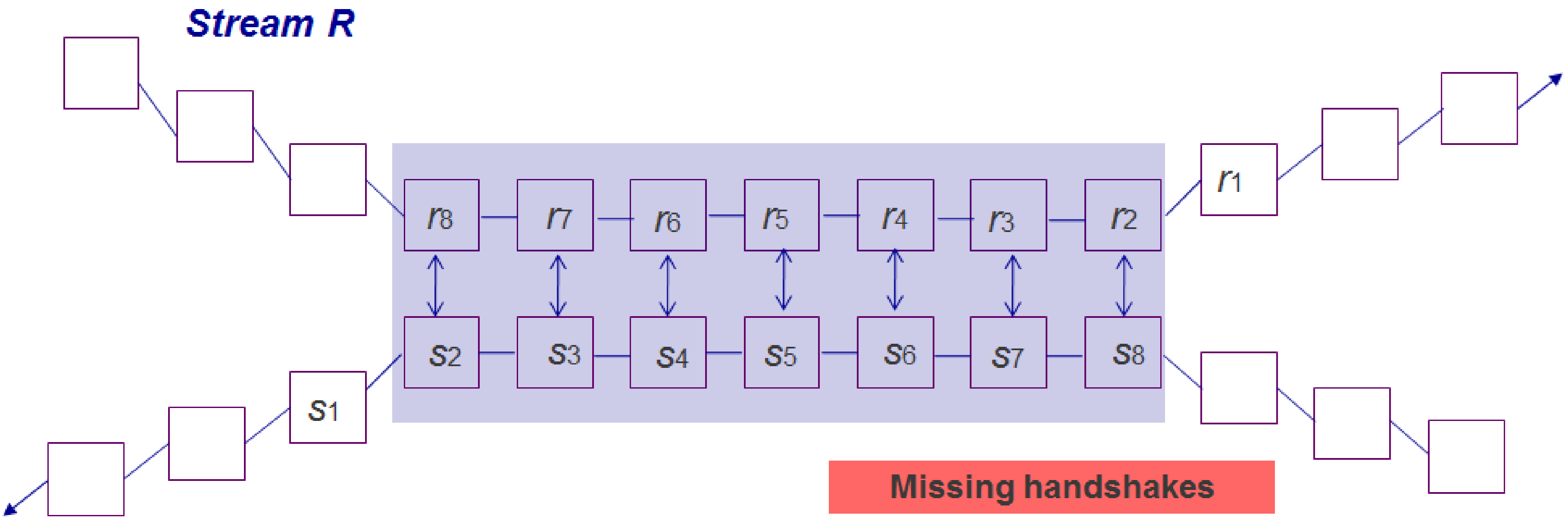
Time Slide (using M-Join):

- **Process Stream *T*:**
  - Take all tuples from basic windows *t1* and *t2*, and probe to hash tables *S* and *R*
  - Take all tuples from basic windows *t1* and *t2* to hash table *T*

Handshake Join



Handshake Join

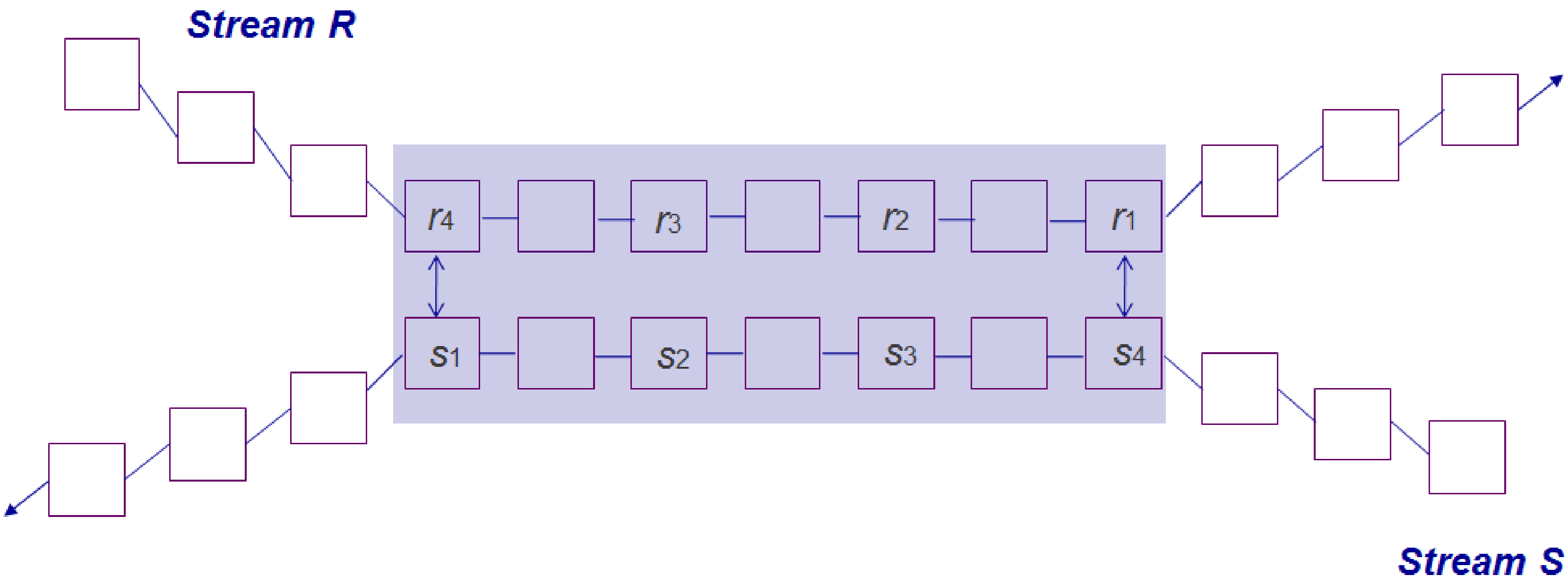


Missing handshakes

Think of at least 2 solutions…

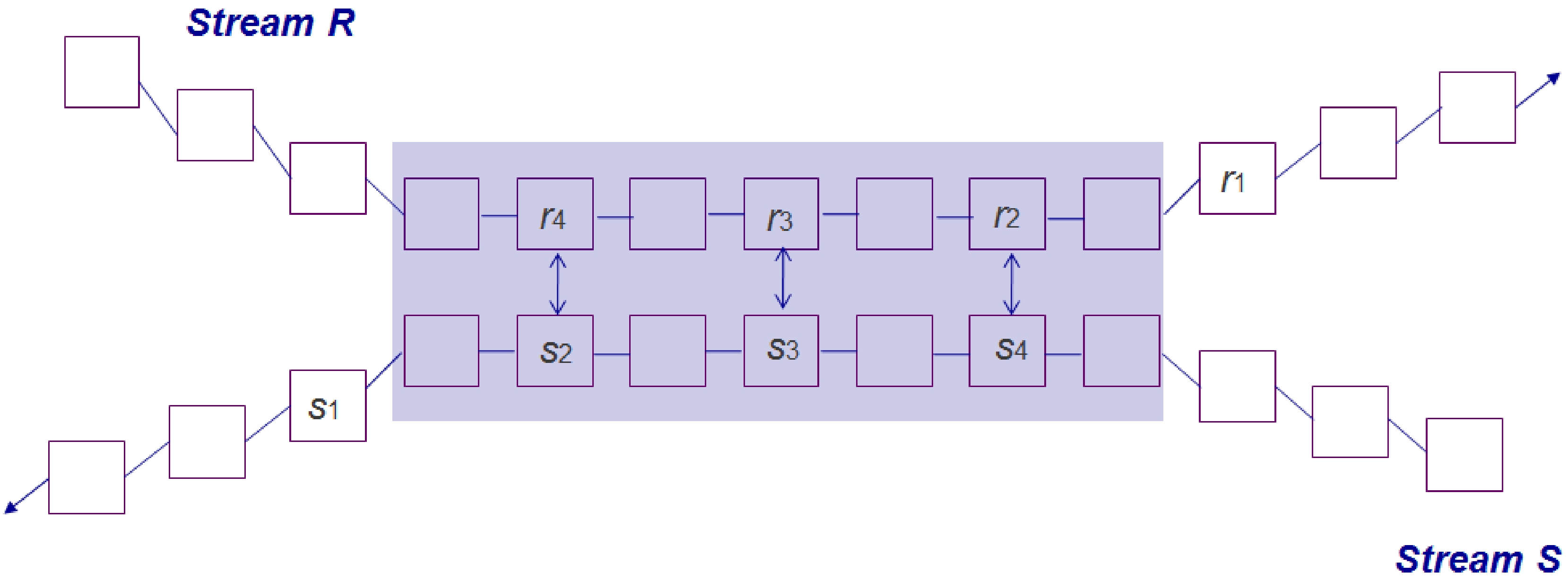
Handshake Join (Solution 1)

Solution 1:



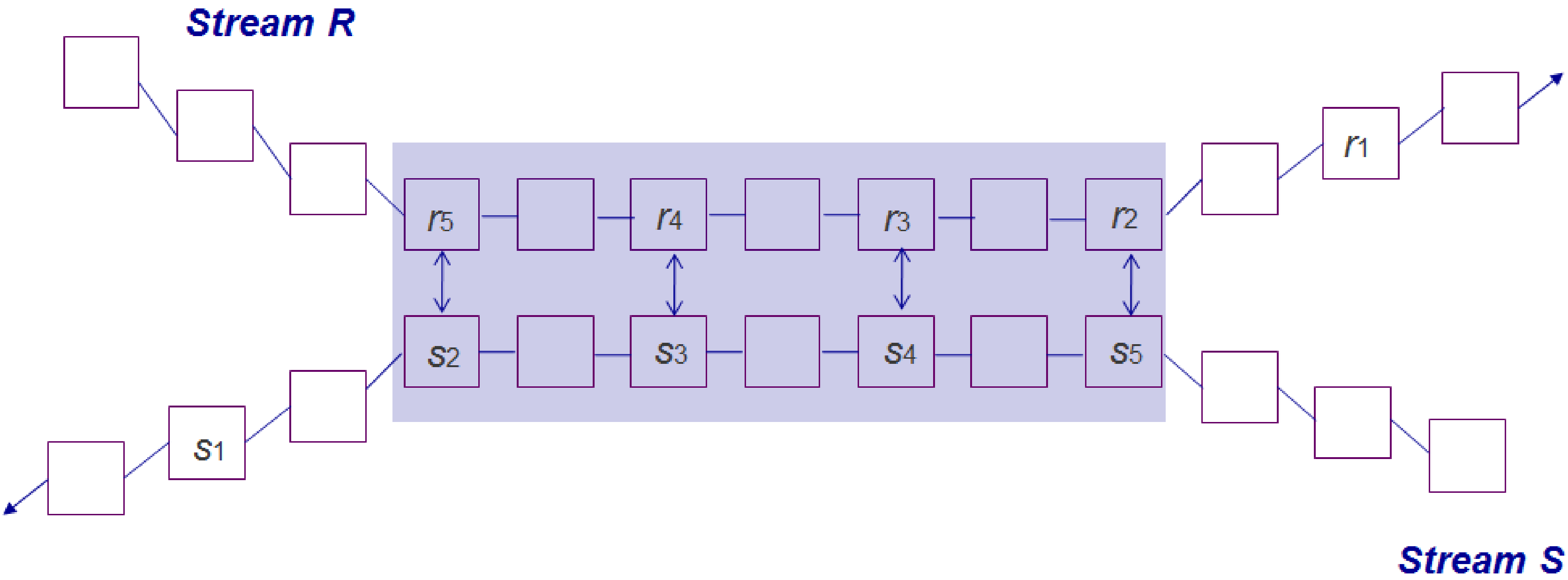
Handshake Join (Solution 1)

Solution 1:



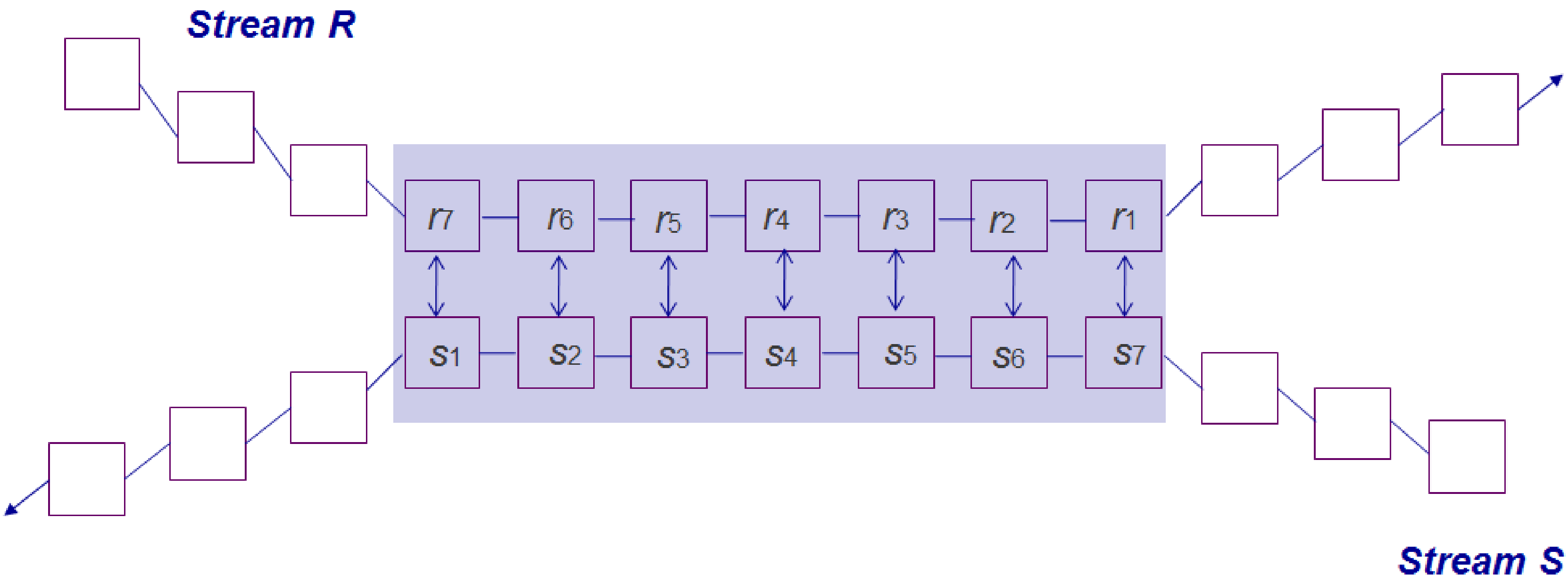
Handshake Join (Solution 1)

Solution 1:



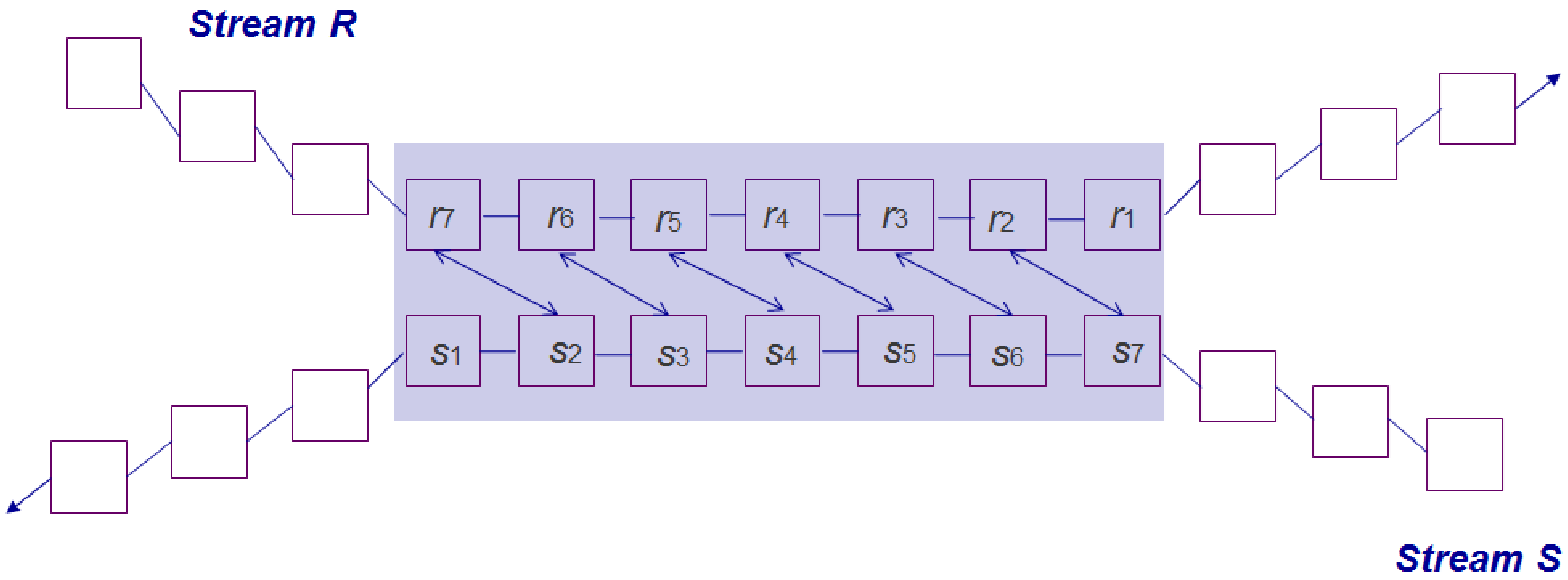
Handshake Join (Solution 2)

Solution 2:



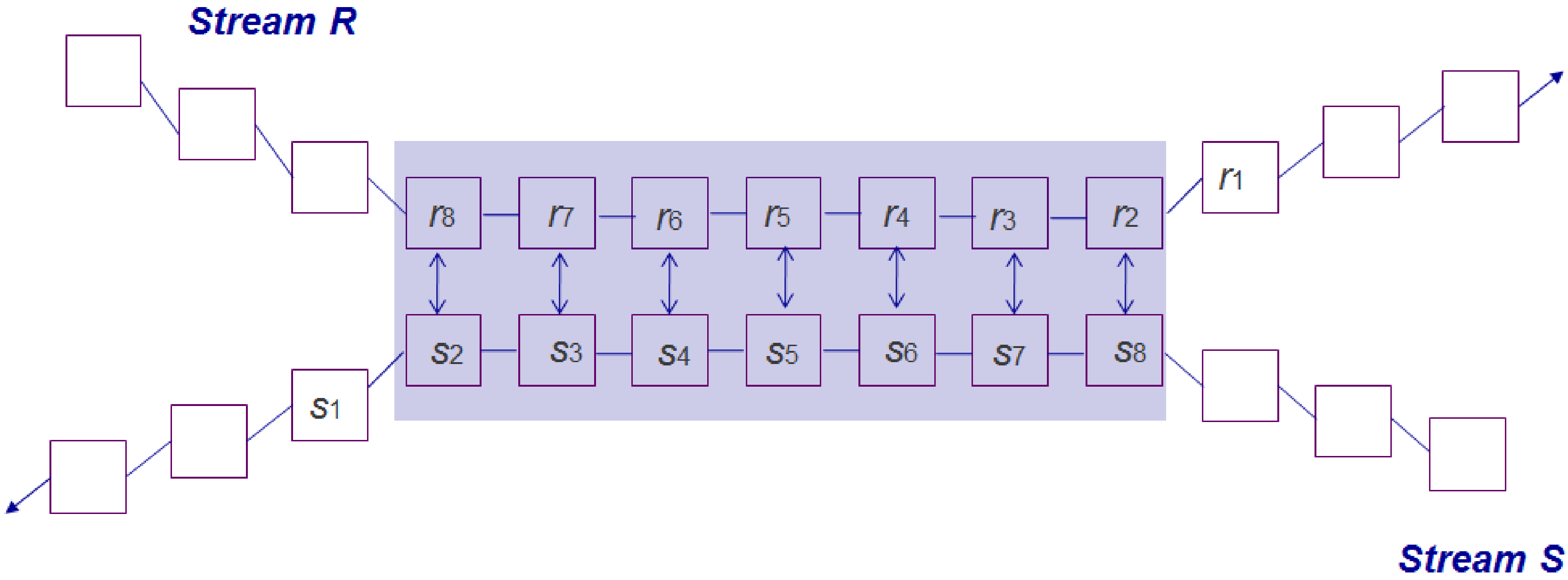
Handshake Join (Solution 2)

Solution 2:



Handshake Join (Solution 2)

Solution 2:



Granularity Reduction In Data Streams

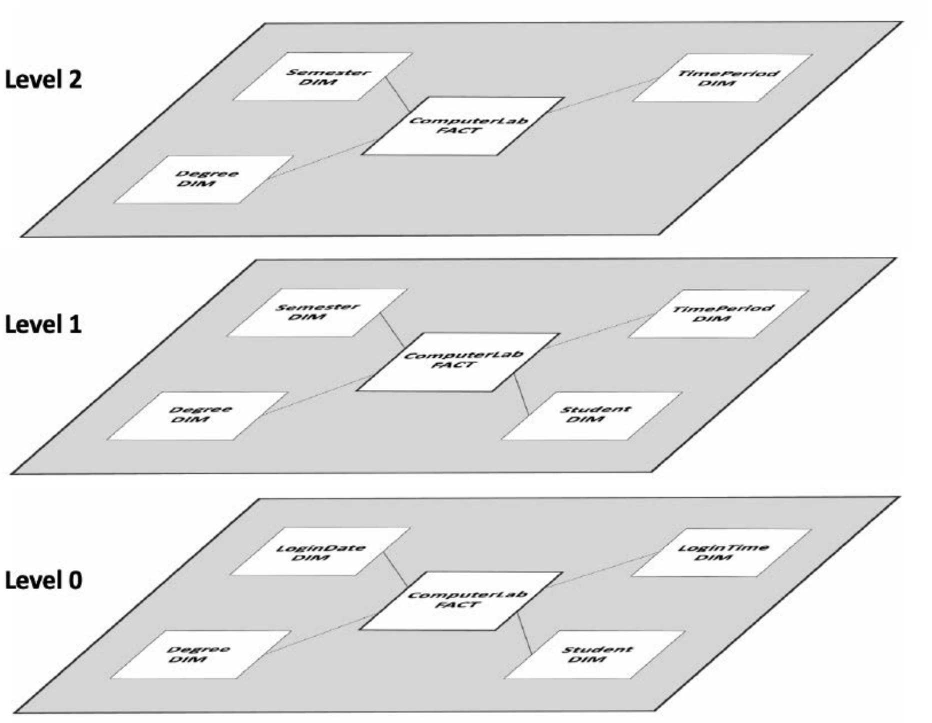
- Group By and Aggregation

Granularity

- **Granularity** is the level of detail at which **data** are stored in a database.

- level-0, the bottom level indicating no aggregation.

- level-1 and level-2 with more aggregation.



Flux Quiz 13

In a sales scenario, if one record is a one-month sales amount, a window of 6 months is used to calculate the running 6-month average sales amount. In this case, the window size is 6 records, and the slide is every record. The number of records in the moving average will be the same as the original number of records. If one year has 12 records of sales, the moving average will also contain 12 records. Hence, no reduction in terms of the number of records. This is a pure moving average (also known as rolling mean).
The above-mentioned case is an example of:

A. Overlapped Windows - No granularity reduction

B. Overlapped Windows - With granularity reduction

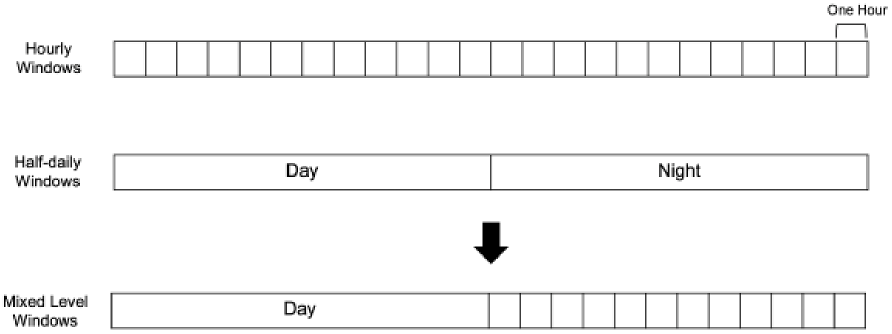
C. Non-Overlapped Windows - Granularity reduction

Mixed Levels of Granularity

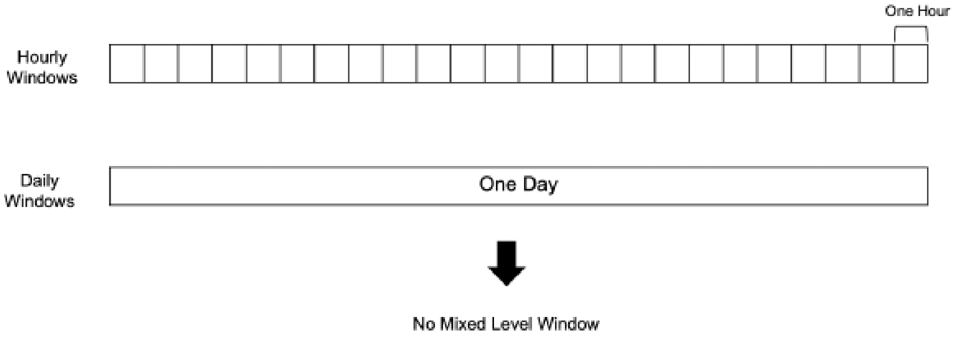
- Different levels of granularity combined into one level.
- Mixed level of granularity can be two types:
  - Temporal-based Mixed Levels of Granularity
    - Time based.
  - Spatial-based Mixed Levels of Granularity
    - Space or location based.

Mixed Levels of Granularity

- Temporal-based Mixed Levels of Granularity
  - Time based.



(a) Mixed Level of Granularity between Day and Night



(b) No Mixed Level

Sensor Arrays

- A sensor array is a group of sensors, usually deployed in a certain geometry pattern.
- A network of distributed sensors.
- They add new dimension to the observation, and hence it helps to estimate more parameters, to have better picture of the environment being observed, and improve accuracy.
- Two categories:

    1. Multiple sensors measuring the same things, and
    2. Multiple sensors measuring different things, but they are grouped together.

Sensor Arrays

- Multiple sensors measuring the same things
- **Two methods to lower the granularity of sensor arrays that measure the same thing:**
  - Method 1: Reduce and then Merge
  - Method 2: Merge and then Reduce

Sensor Arrays

- Multiple sensors measuring different things

    - Sensors arrays can be a collection of sensors measuring different things within the same environment.

- Example: A simple indoor sensor array, containing three sensors: air quality, temperature, and humidity.

- **Two methods to lower the granularity of sensor arrays that measure the different thing:**
    - Method 1: Reduce, Normalize, and then Merge
    - Method 2: Normalize, Merge and then Reduce

Unit Summary

1. **Volume** ➔ Sessions 1, 2, 3, 4
   – How to process Big Data Volume?

2. **Complexity** ➔ Sessions 5, 6, 7, 8
   – How to apply machine learning algorithms to every aspect of Big Data?

3. **Velocity** ➔ Sessions 9, 10, 11
   – How to handle and process Fast Streaming Data?

Thank You

Questions?