MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT5202 – Data Processing for Big Data

Stream Join Processing
Edited by
Chee-Ming Ting

Developed by
Prajwol Sangat

GROUP OF EIGHT AUSTRALIA

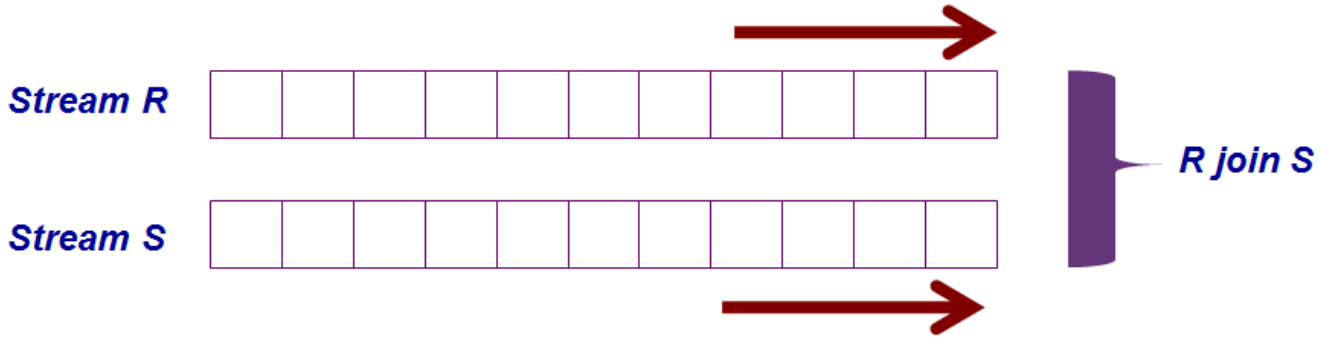
Last Week

- Streaming Data Processing
- Streaming Processing Technology

This week

- Overview of Stream join
- Time based window stream join (Unbounded)
  - Tuple slide
  - Time slide
- Tuple based window stream join (Unbounded)
- Bounded stream join

Overview of Stream Join



Bounded Stream Join

- If both streams R and S are bounded streams
- There is a start and an end

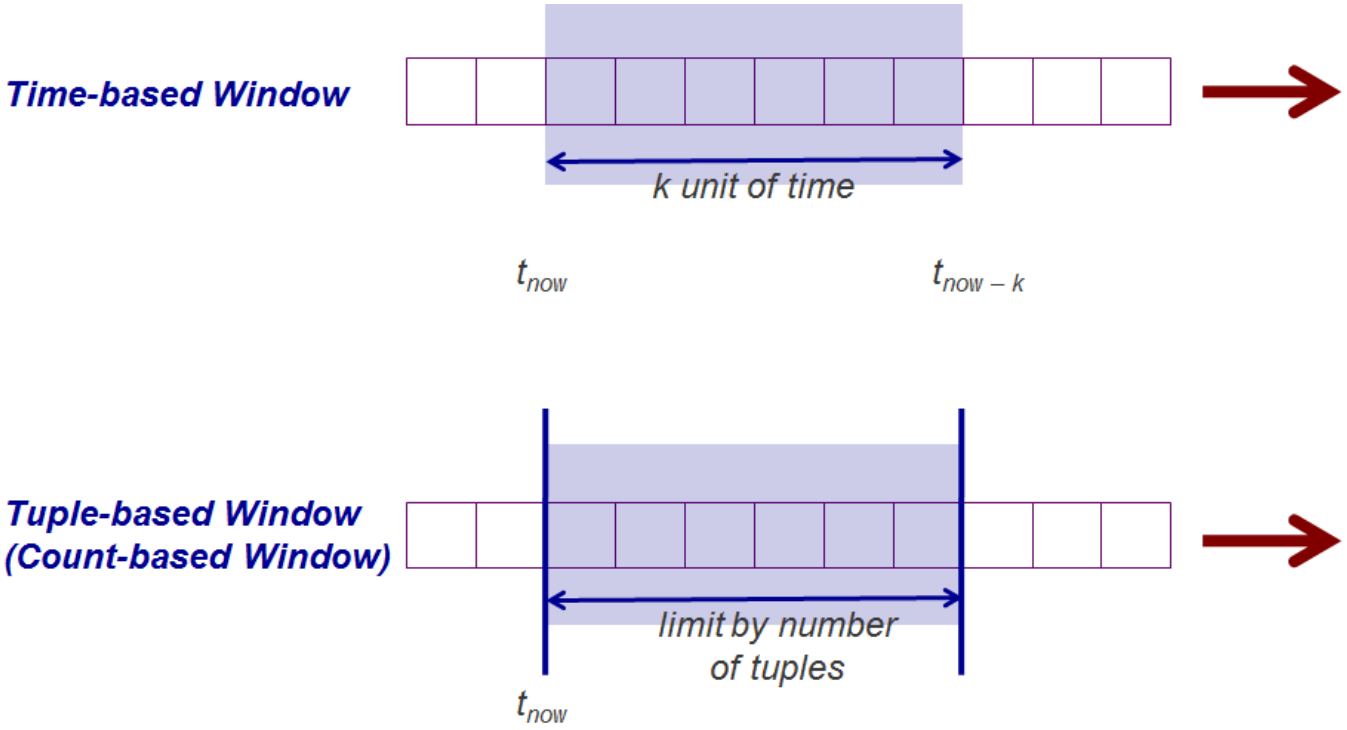
Unbounded Stream Join

- If any of the streams is unbounded
- There is a start but no end

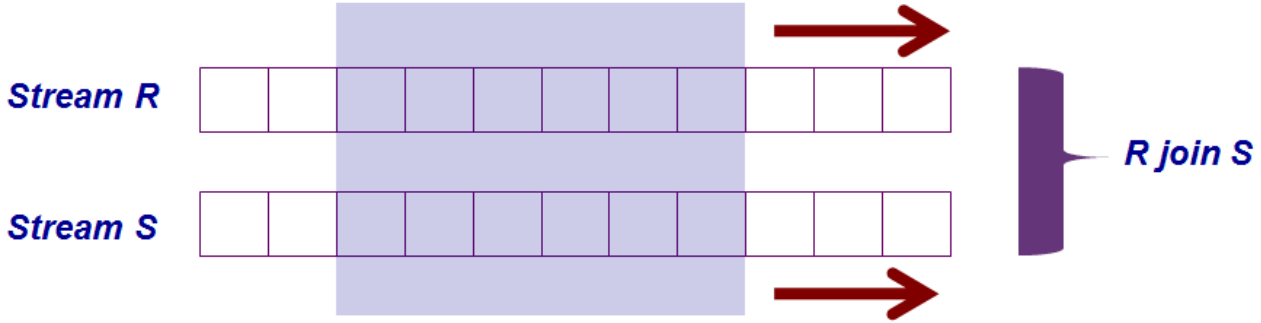
Challenges:

- How to join streams with no end?
- Due to network latency, some data arrives late compared to other related tuples

Recap - Windowing System in Unbounded Streams



Window-based Stream Join



A General Stream Join Process:

- When a tuple *r* arrives from input stream *R*:
  - Scan stream *S*'s window to find tuples matching *r,* and get join result
  - Insert new tuple *r* into window for stream *R*
  - Invalidate all expired tuples in stream *R*'s window

Hence, stream join is based on the data in the current window!!

- Tuples in the window cannot be matched with expired tuples (discarded from window) and with incoming tuples (not arrived yet)

Review – Hash Join

Hash S Table

Probe Table R into hash table

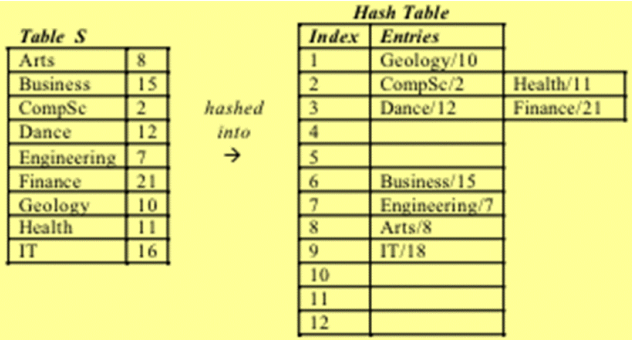


Figure 5.6. Hashing Table *S*

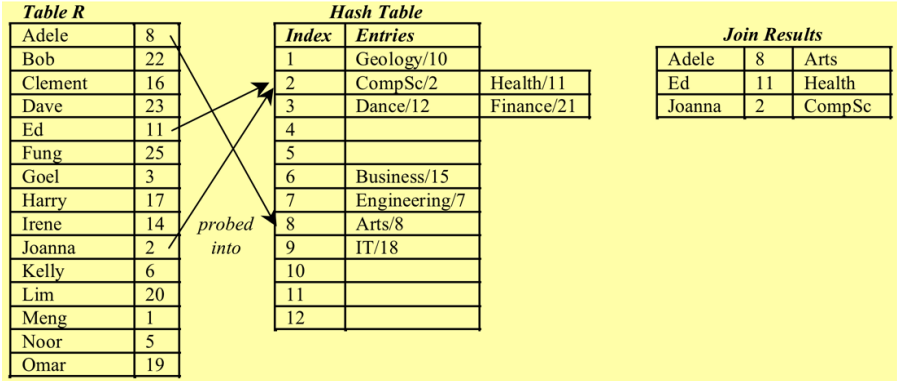
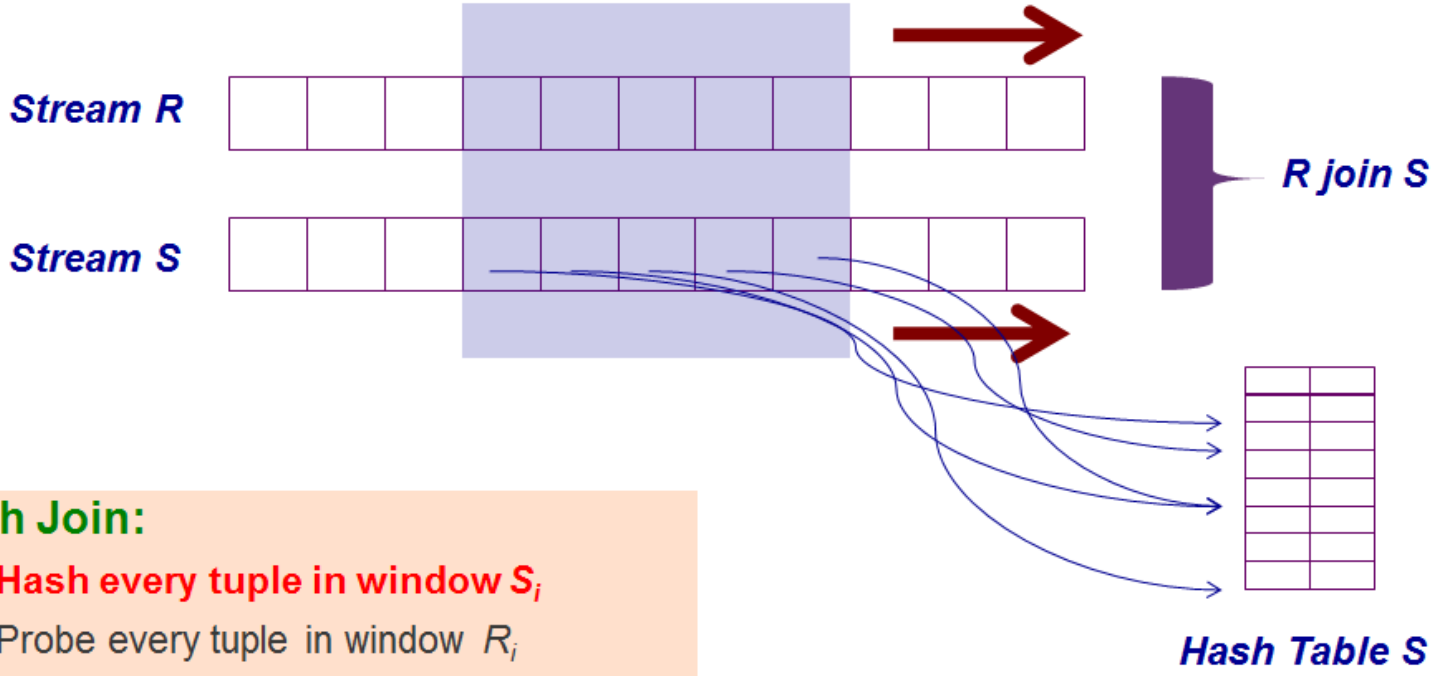


Figure 5.7. Probing Table *R*

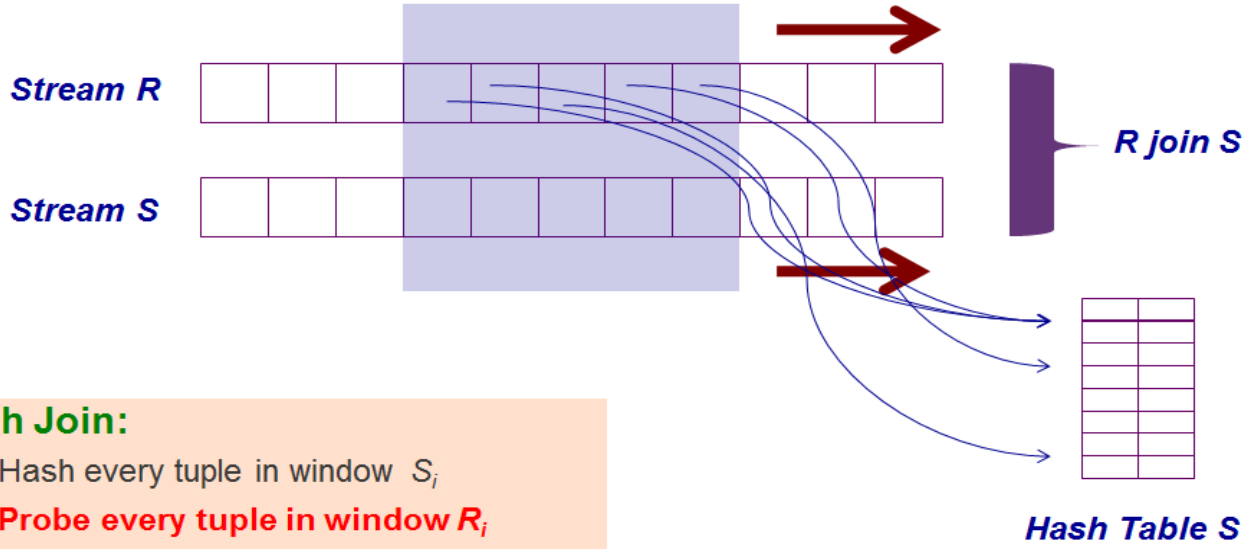
Hash Join



Hash Join:

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

Hash Join



Hash Join:

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

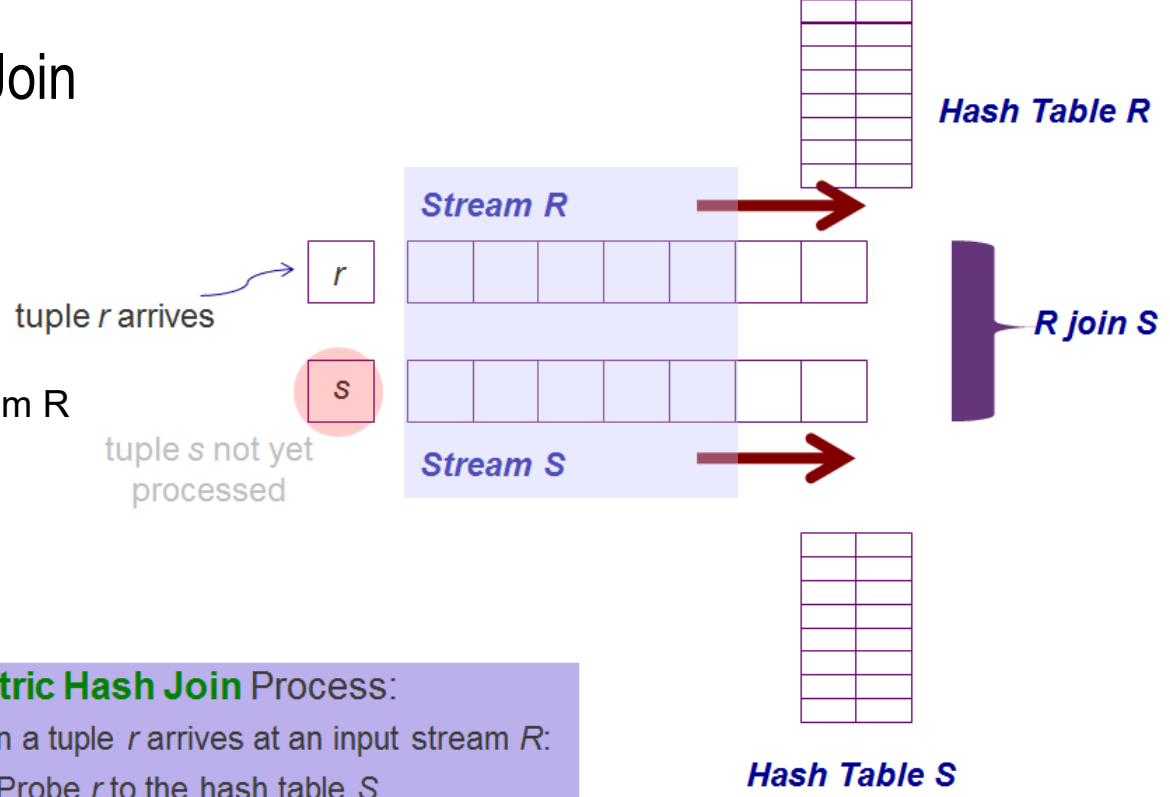
What's the problem?

Problem: If *r* comes in first and then *s* comes in later, then *r* will not be compared with the *s*

→ *If r and s has same key, we will miss join operation*

Symmetric Hash Join

When r arrives at input stream R



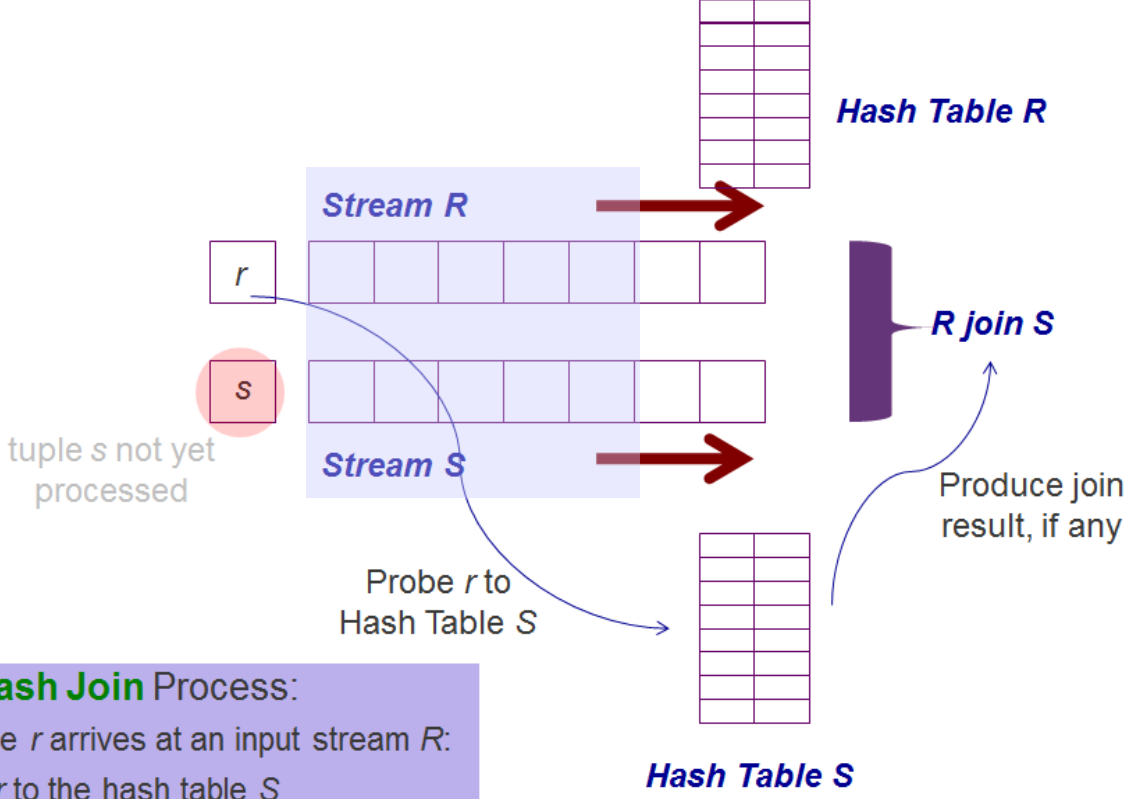
Symmetric Hash Join Process:

- 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

Step 2:

Probe r into hash table S



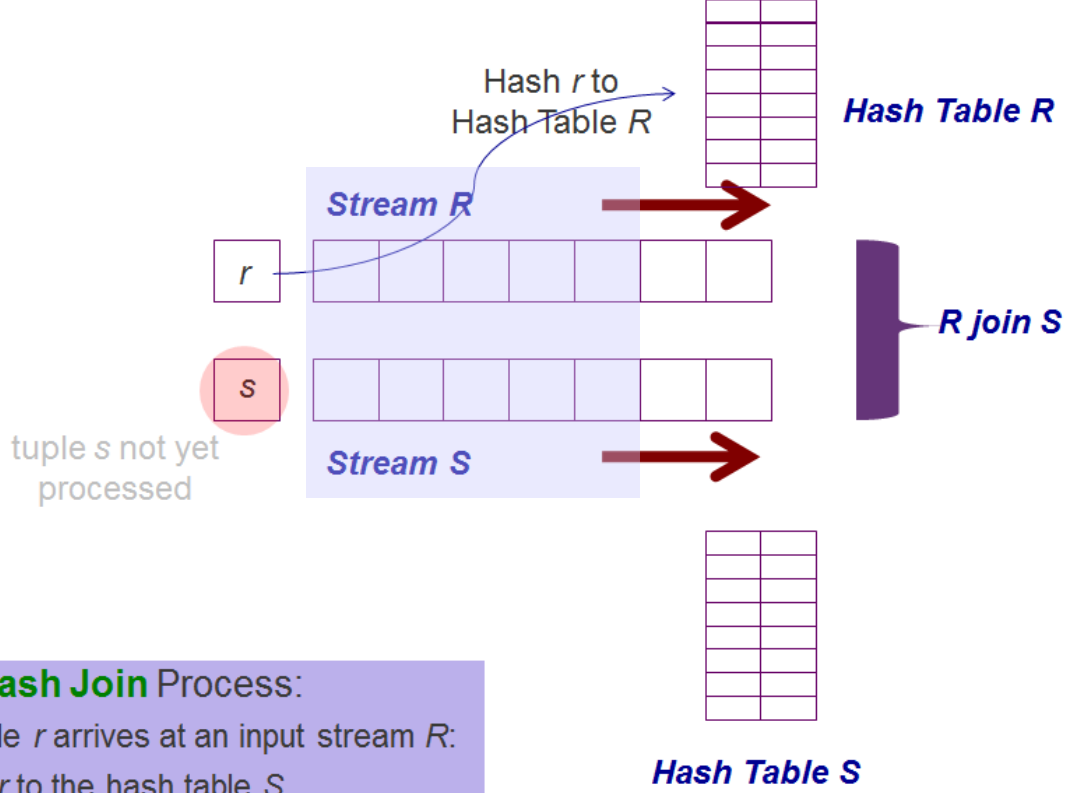
Symmetric Hash Join Process:

- 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

Step 3:

Hash r into hash table R



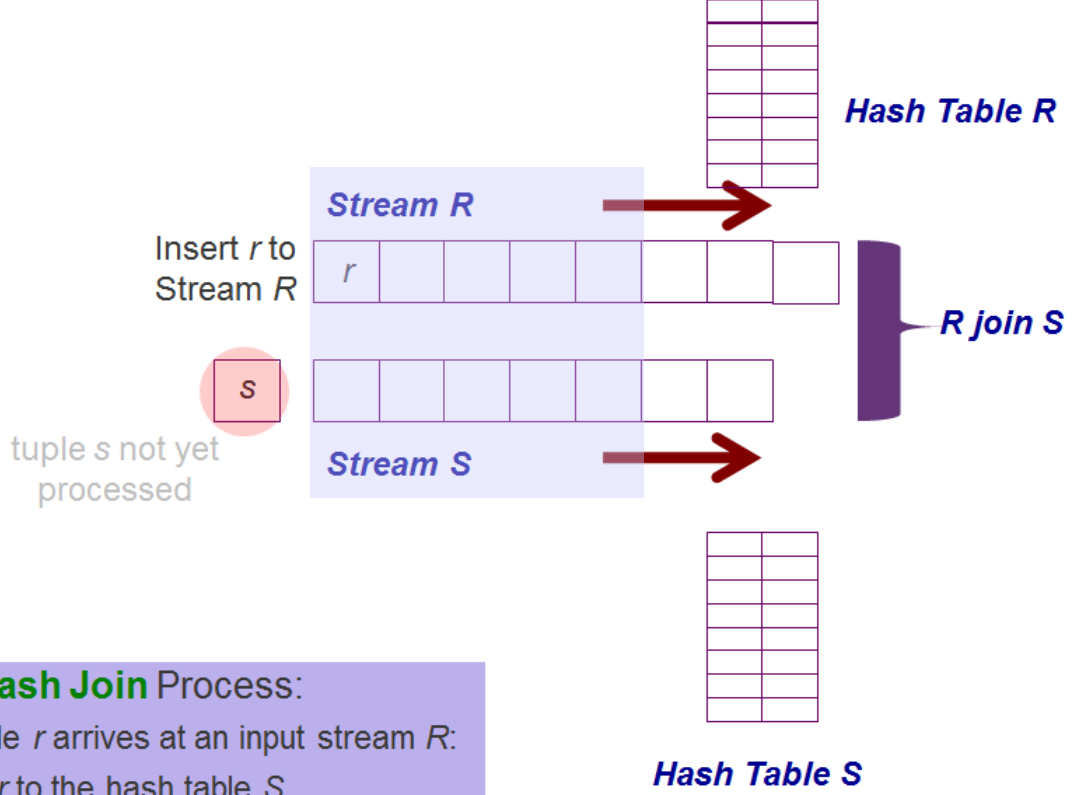
Symmetric Hash Join Process:

- 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

Step 4:

- When *s* comes in, probe into hash table R to get join results
- After that, hash s into own hash table, and insert into stream S



Symmetric Hash Join Process:

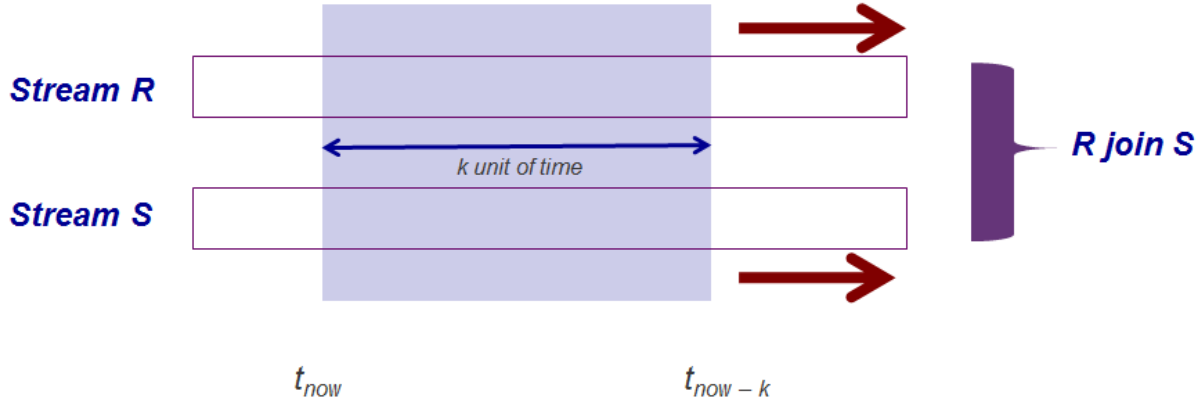
- When a tuple *r* arrives at an input stream *R*:
  - Probe *r* to the hash table *S*
  - Hash tuple *r* into hash table *R*
  - Insert new tuple *r* into stream *R*

By using two hash tables (symmetric), we don't miss the join of any incoming tuple.

This week

- Overview of Stream join
- Time based window stream join (Unbounded)
  - Tuple slide
  - Time slide
- Tuple based window stream join (Unbounded)
- Bounded stream join

Time-based Window Stream Join



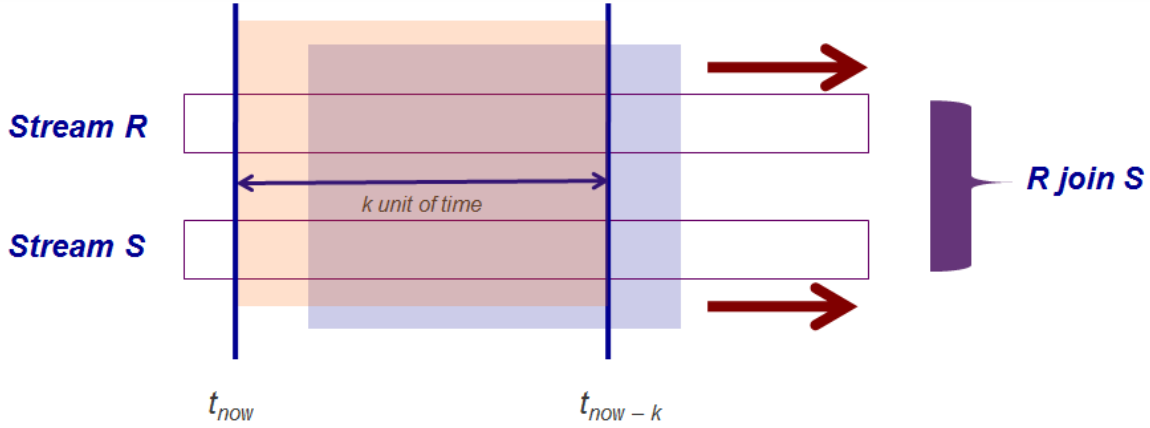
Unbounded Stream Join (Window-based):

- Join is only applied to tuples in the window
- **But window is a running window…**

Each window has fixed time duration

- Any numbers of tuples within that window will be considered for the join operation

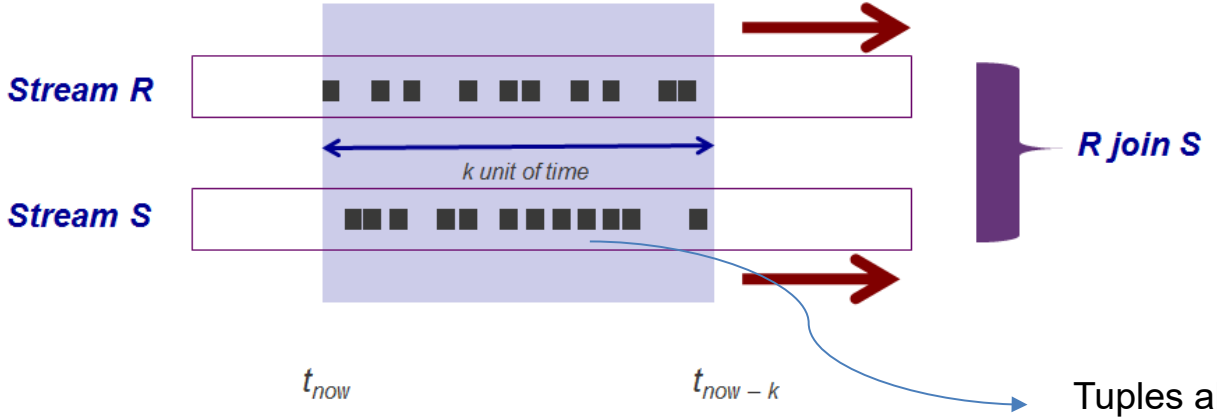
Time-based Window Stream Join



Unbounded Stream Join (Window-based):

- Join is only applied to tuples in the window
- **But window is a running window…**
- **How to slide the window?**
  - Tuple **Slide** - Slide window based on number of tuples (move window when tuples come in)
  - Time **Slide** - Slide window based on time interval (e.g., move window every 30s)

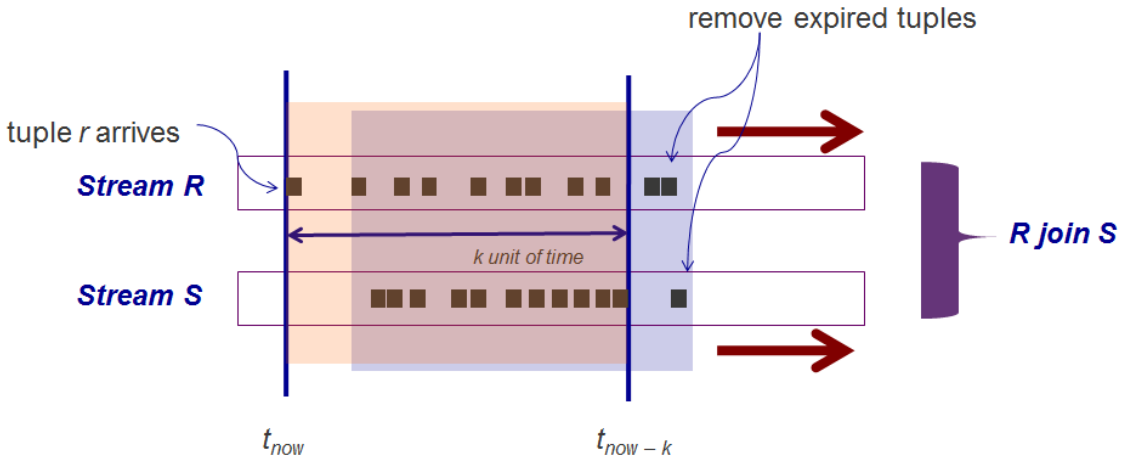
Tuple Slide Stream Join



Tuple Slide Stream Join:

- When a new tuple *r* arrives at stream *R*
  - Slide all the windows (remove expired tuples)
  - Join *r* with all tuples in the new window in stream *S*
  - Add *r* to the window in stream *R*

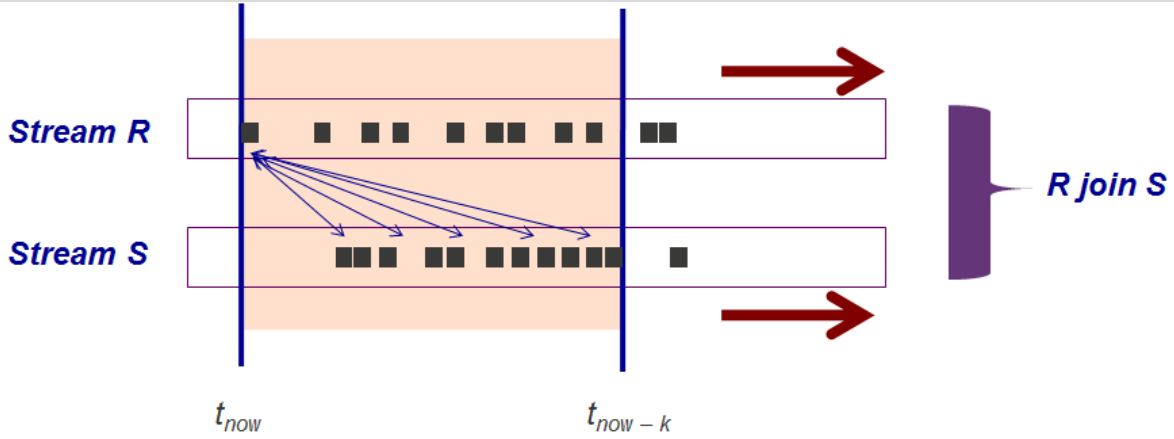
Tuple Slide Stream Join



Tuple Slide Stream Join:

- When a new tuple *r* arrives at stream *R*
  - **Slide all the windows (remove expired tuples)**
  - Join *r* with all tuples in the new window in stream *S*
  - Add *r* to the window in stream *R*

Tuple Slide Stream Join



Tuple Slide Stream Join:

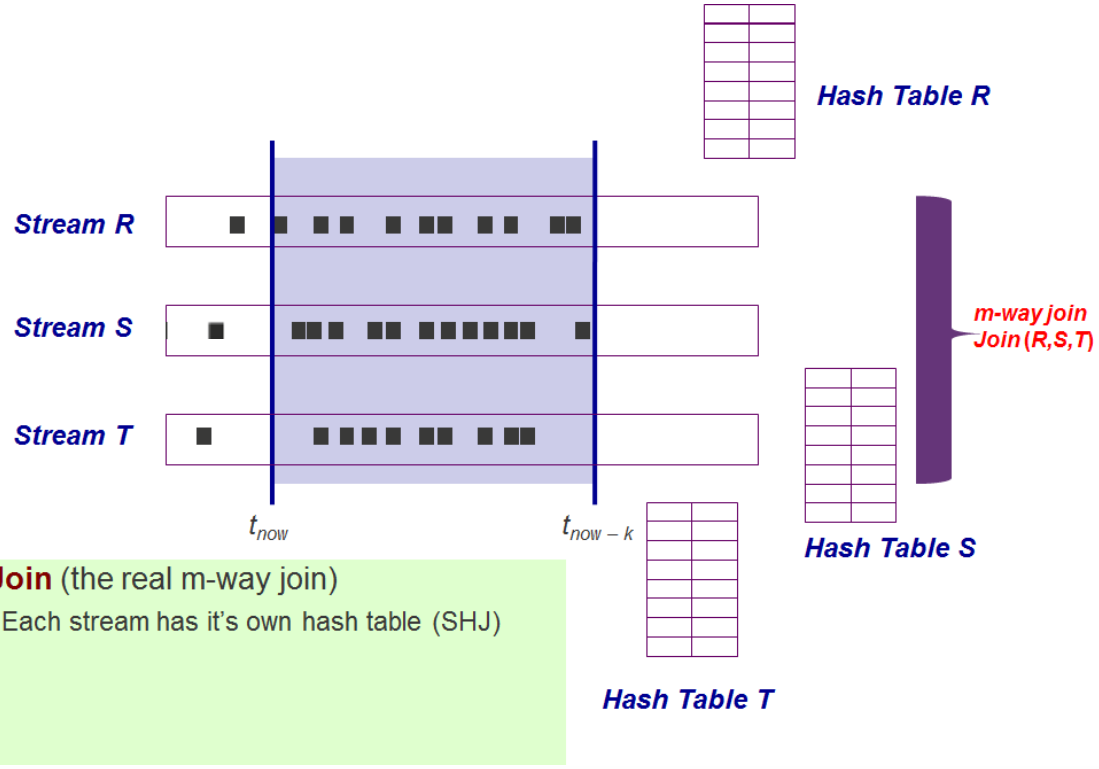
- When a new tuple *r* arrives at stream *R*
  - **Slide all the windows (remove expired tuples)**
  - **Join *r* with all tuples in the new window in stream *S* ➔ which join method?** → Symmetric Hash join
  - Add *r* to the window in stream *R*

Tuple Slide (Using M–Join)

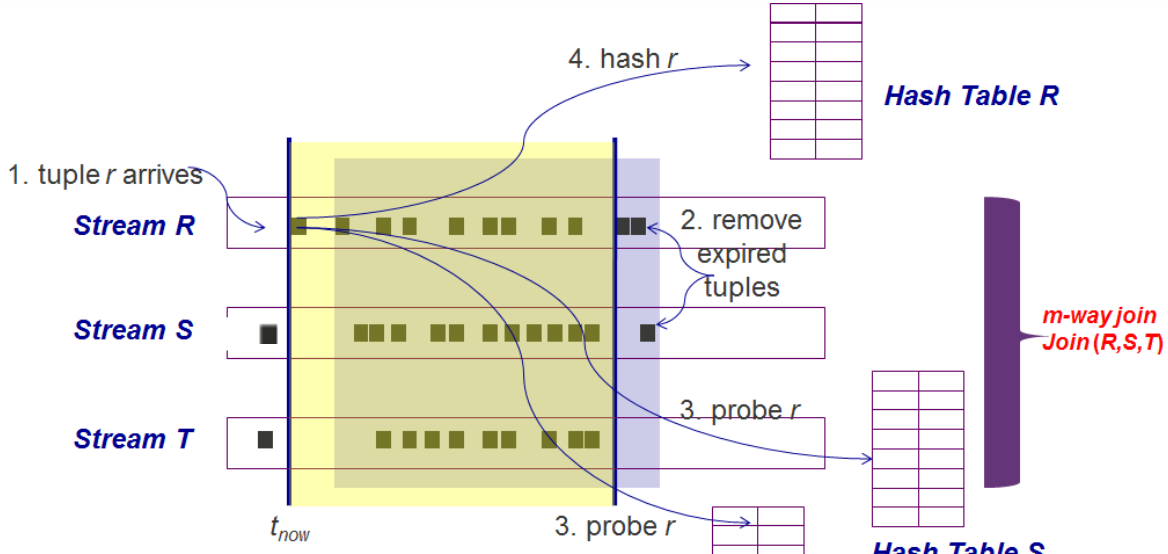
How to join more than two streams at the same time?

- Use M-Join – a multiway streaming join
- Based on symmetric hash join

(work similarly to two-stream)



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*

Repeat same procedure when tuple *s* arrives

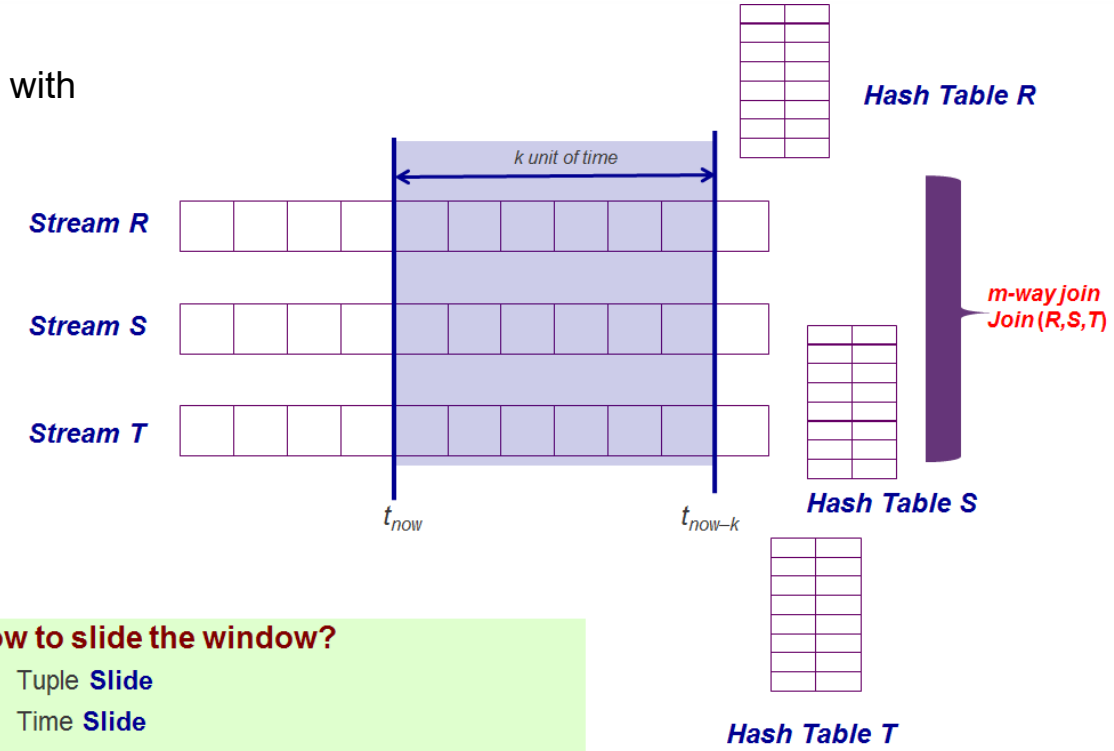
- Probe s into hash table R & T, and find the match
- Then, hash s to hash table S

Advantages: can still match *r* and *s* although s arrive later than *r*

Repeat the same when tuple *t* arrives

Time Slide (Using M-Join)

How M-join works with time-slide window



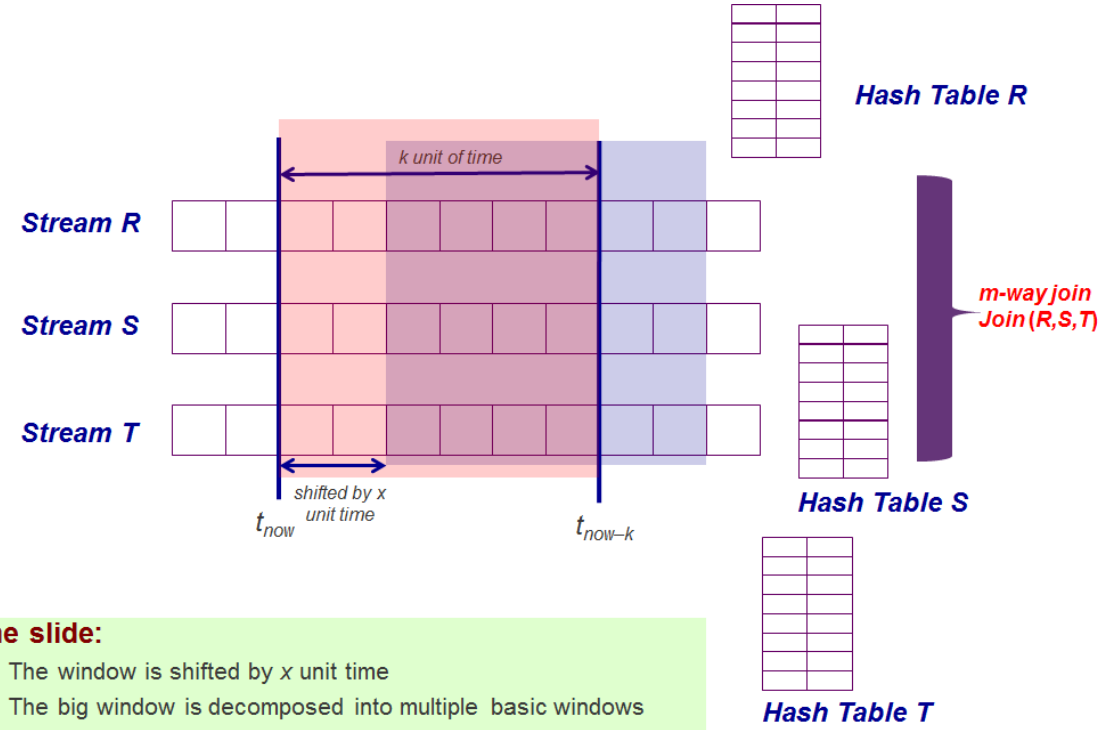
How to slide the window?

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

Time Slide (Using M-Join)

- Let one box here represents one unit time or one basic window.

Ex: 1 unit time = 1 minute
- Window size = 6 mins
- slide = 2mins

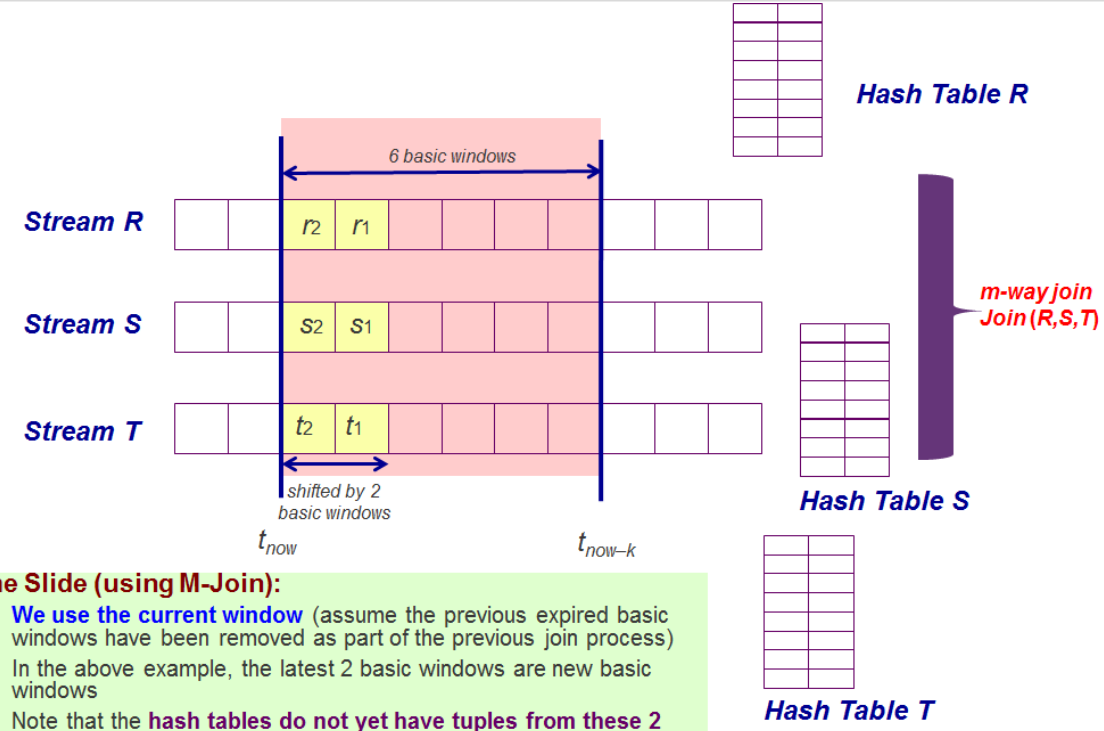


Time slide:
- The window is shifted by *x* unit time
- The big window is decomposed into multiple basic windows
- The big window is then **shifted by 1 or more basic windows**

Time Slide (Using M-Join)

Question:

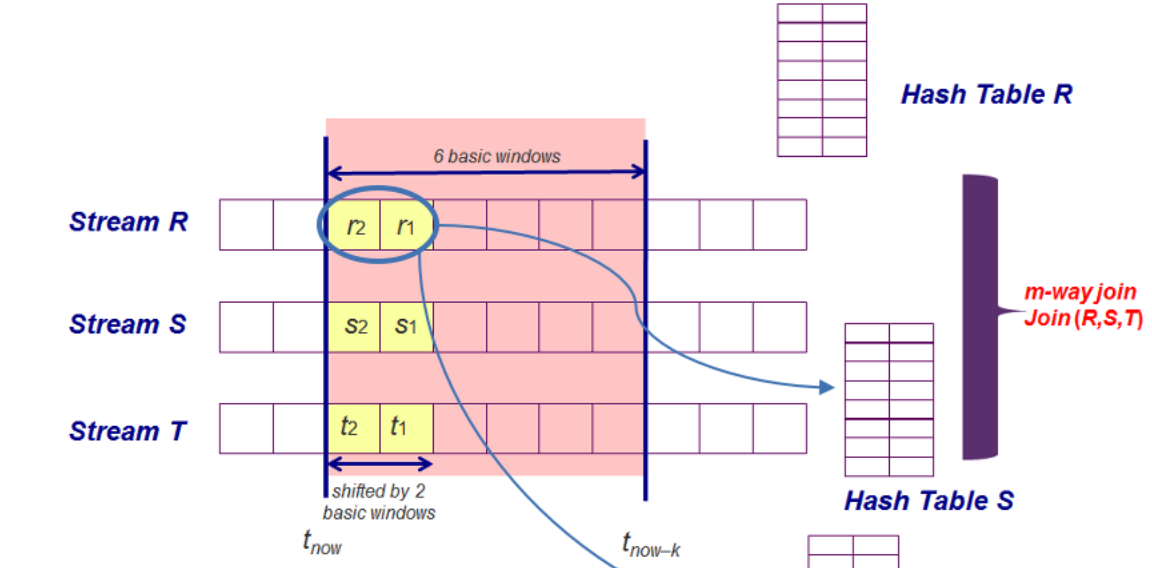
How to process or join tuples in the new basic windows?



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**

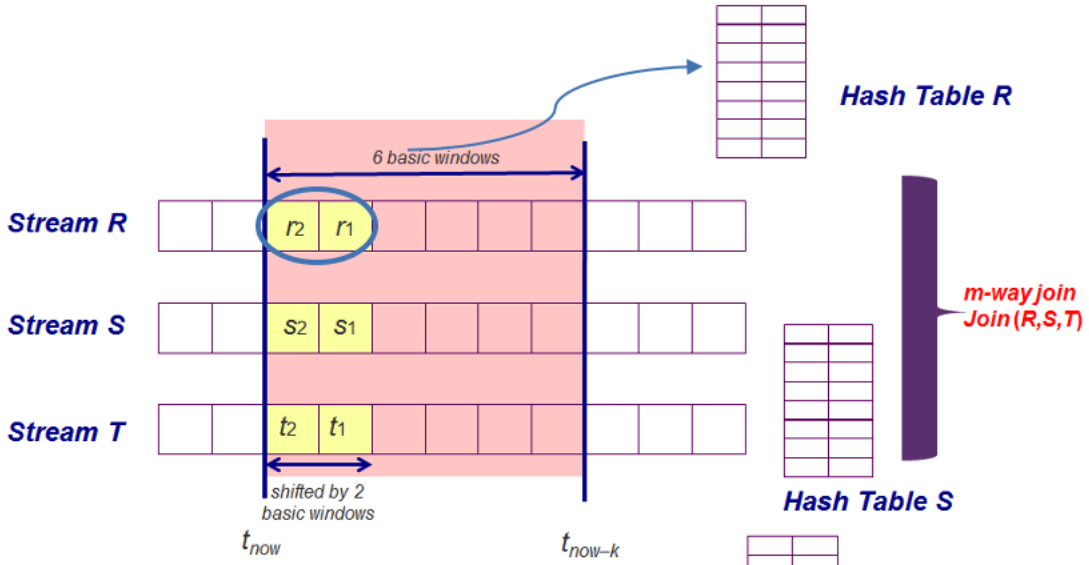
Time Slide (Using M-Join)



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*

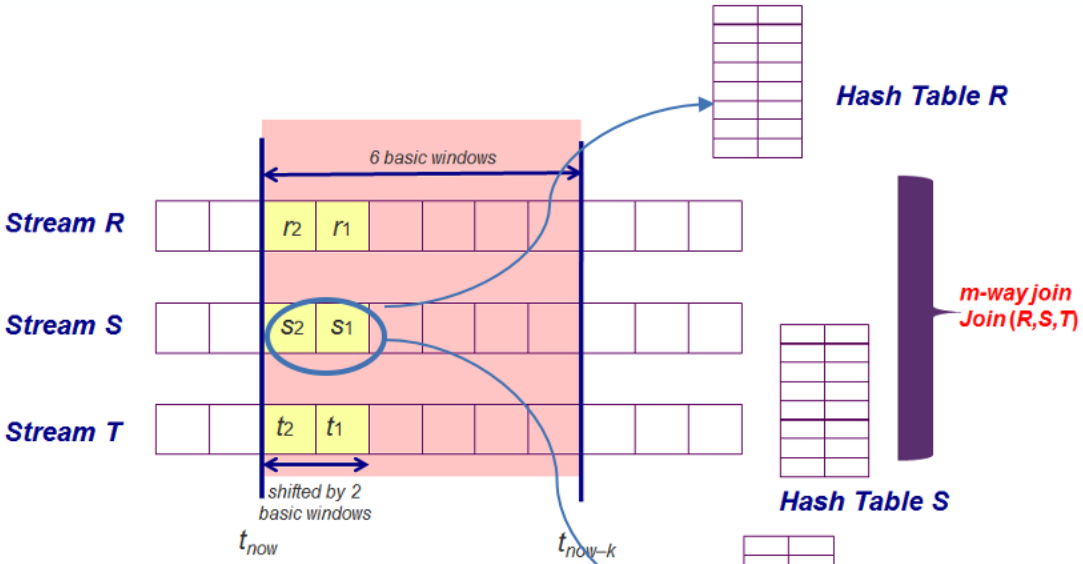
Time Slide (Using M-Join)



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*

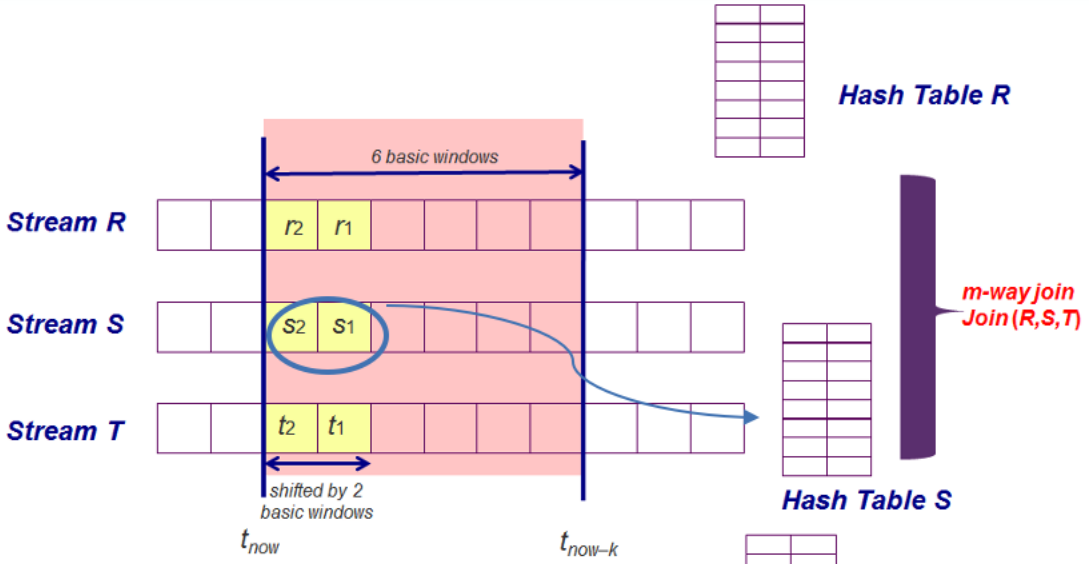
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *S*:**
  - Take all tuples from basic windows *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*

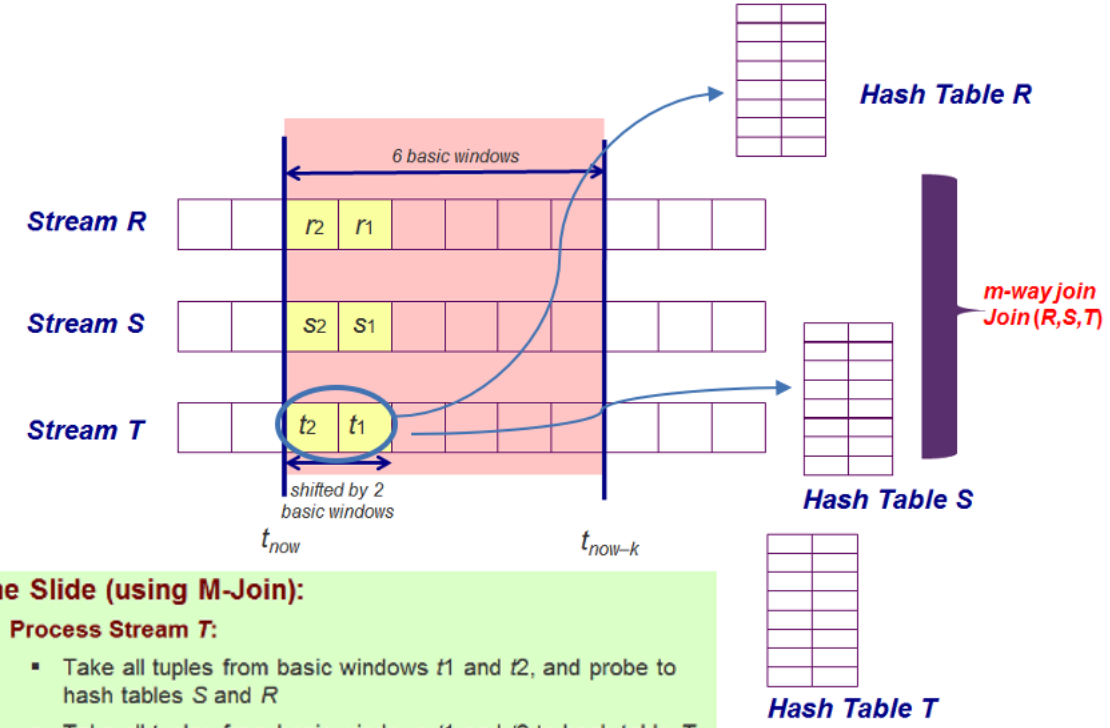
Time Slide (Using M-Join)



Time Slide (using M-Join):

- **Process Stream *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*

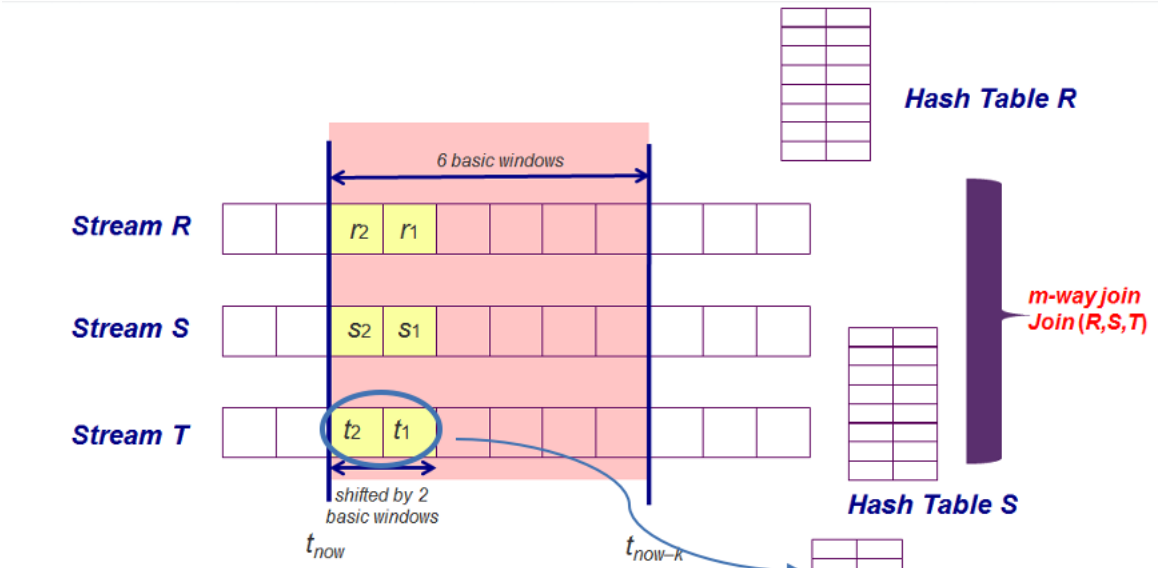
Time Slide (Using M-Join)



Time Slide (using M-Join):

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

Time Slide (Using M-Join)



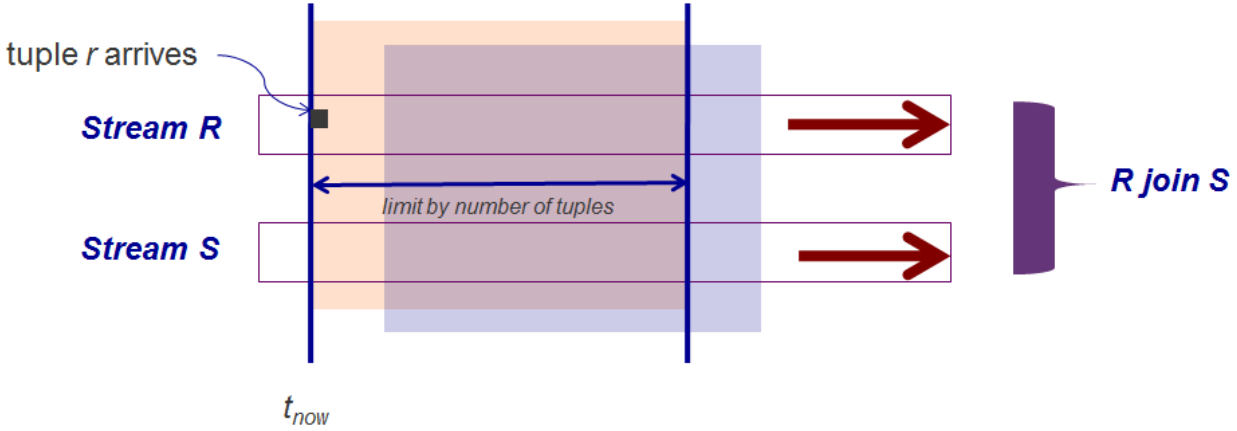
Time Slide (using M-Join):

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

This week

- Overview of Stream join
- Time based window stream join (Unbounded)
  - Tuple slide
  - Time slide
- Tuple based window stream join (Unbounded)
- Bounded stream join

Tuple-based Window Stream Join

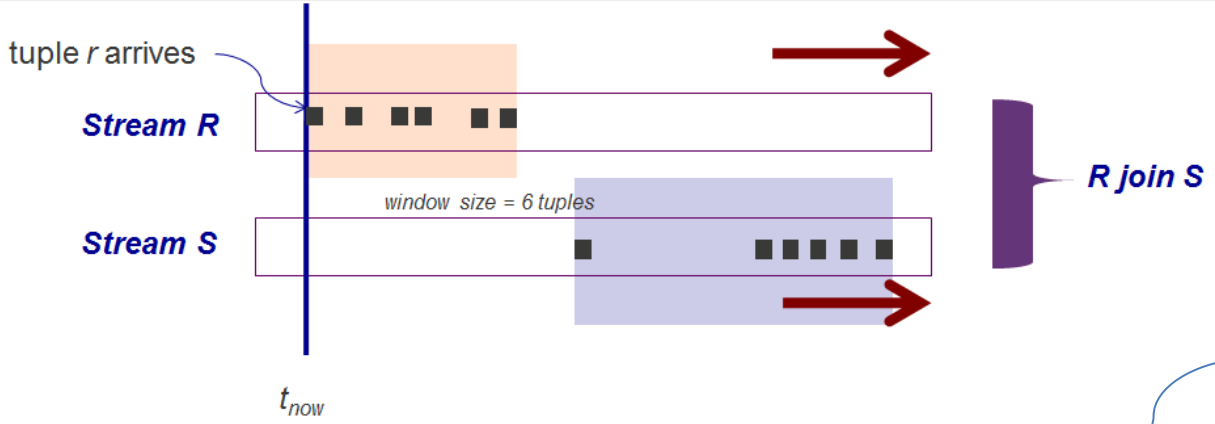


- **Size of the window** is *k* number of tuples
- When a tuple *r* arrives in stream *R*, we readjust the window size of stream *R*. What about stream *S*? If the size of the window is based on stream *R*, window in stream *S* might have less number of tuples (thus violates the tuple-based window rule)
- If we allow **window size to be different**, what is the **semantic of the window**?

Ex: window size =100 tuples

- When 1 tuple comes in , oldest tuple needs to be removed to include it in the window because the size has to be 100

Tuple-based Window Stream Join



- Should we have the **same window size** for all streams in the join?
- If this is the case, in an extreme case, windows among streams will not overlap
- What is the semantic of the window, and hence the join?

Not many research in Tuple-based Window Join, because lack of understanding on how tuple-based window may be applied to stream join.

- If we slide windows for both R & S at the same time → window size will be different
- OR, we have different sliding mechanism for S, e.g., no slide until it has new tuples → same window size

- Problem: If two streams have different arrival rate (e.g. R is faster than S)
- Tuples in S will get very old

- When performing join, you join new data in R with old data in S

Inspired by how soccer players handshake

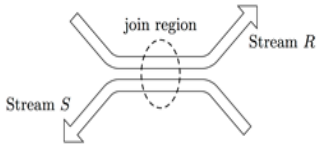


Figure 1: Handshake join idea.

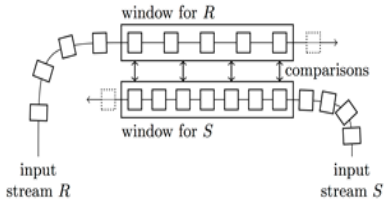


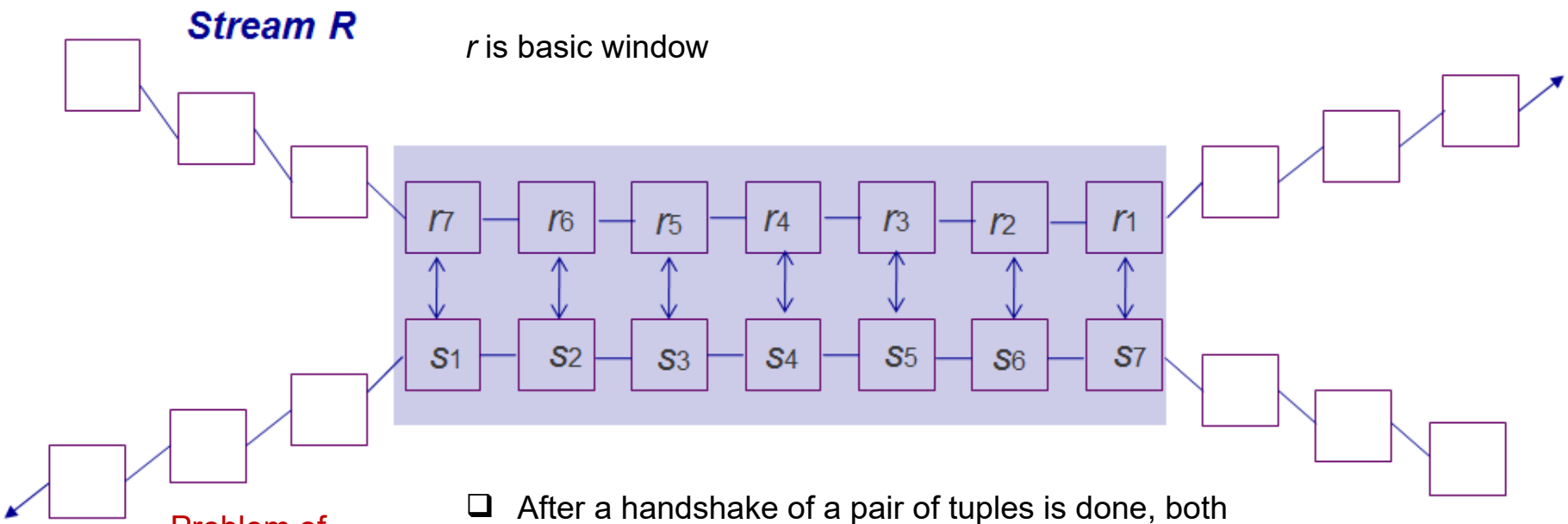
Figure 3: Handshake join sketch. Streams flow by each other in opposite directions; comparisons (and result generation) happens in parallel as the streams pass by.

- Even then… this is not a Tuple-based Window Stream Join
- It is still a Time-based Window Stream Join
- But monitoring each tuple (and when it is expired) looks more natural in this Soccer Handshake method.

Looks nice, but there are problems…

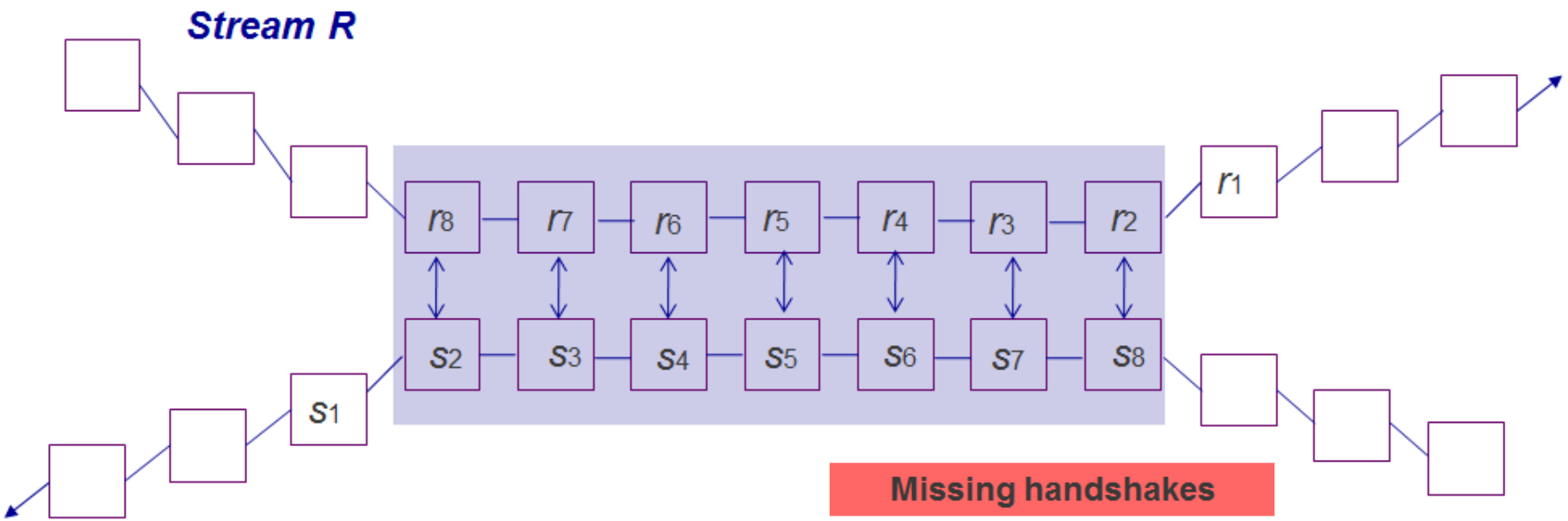
- ❑ Streams flow by each other in opposite direction.
- ❑ Ex: stream R moves from left to right and stream S moves from right to left
- ❑ Comparison/join occurs when streams pass by

Handshake Join



- After a handshake of a pair of tuples is done, both streams R and S move forward at the same time
- As a result, we will miss a handshake because one tuple from R moves to the right, and one tuple from S moves to the left, and one handshake will be missed

Handshake Join

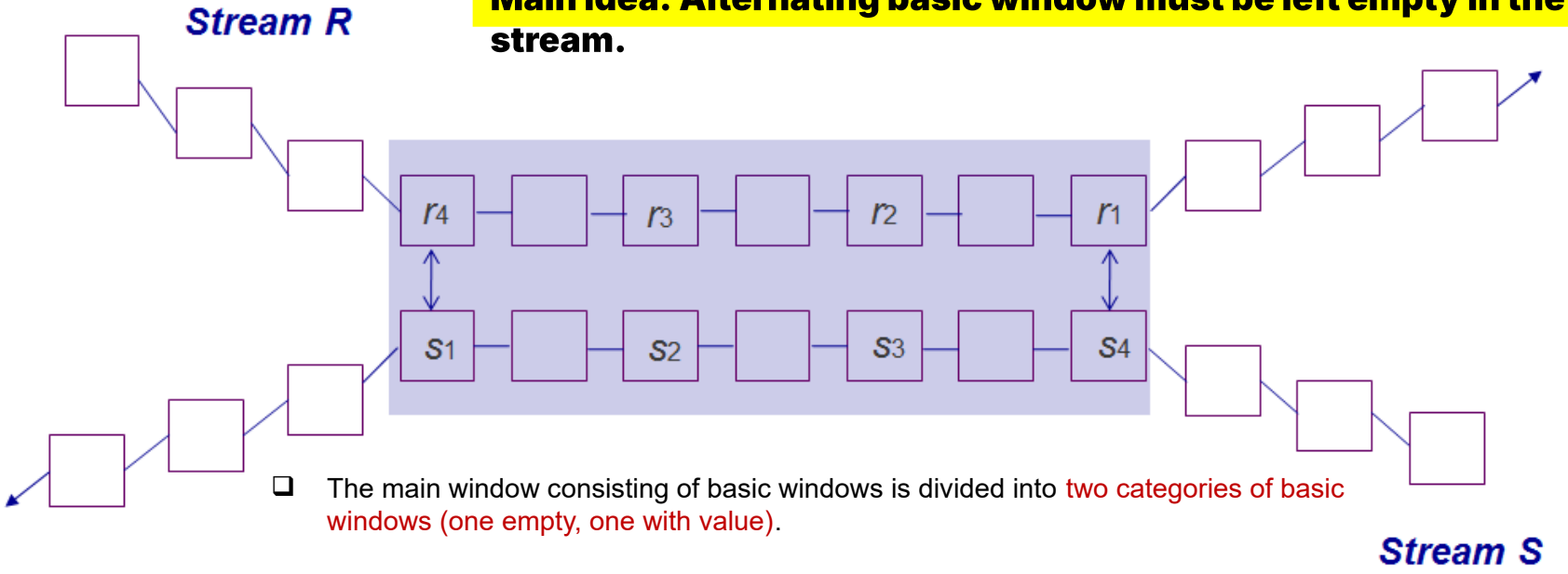


- r4 hand shakes with s6 (not s5) and we miss one handshake

Handshake Join (Solution 1)

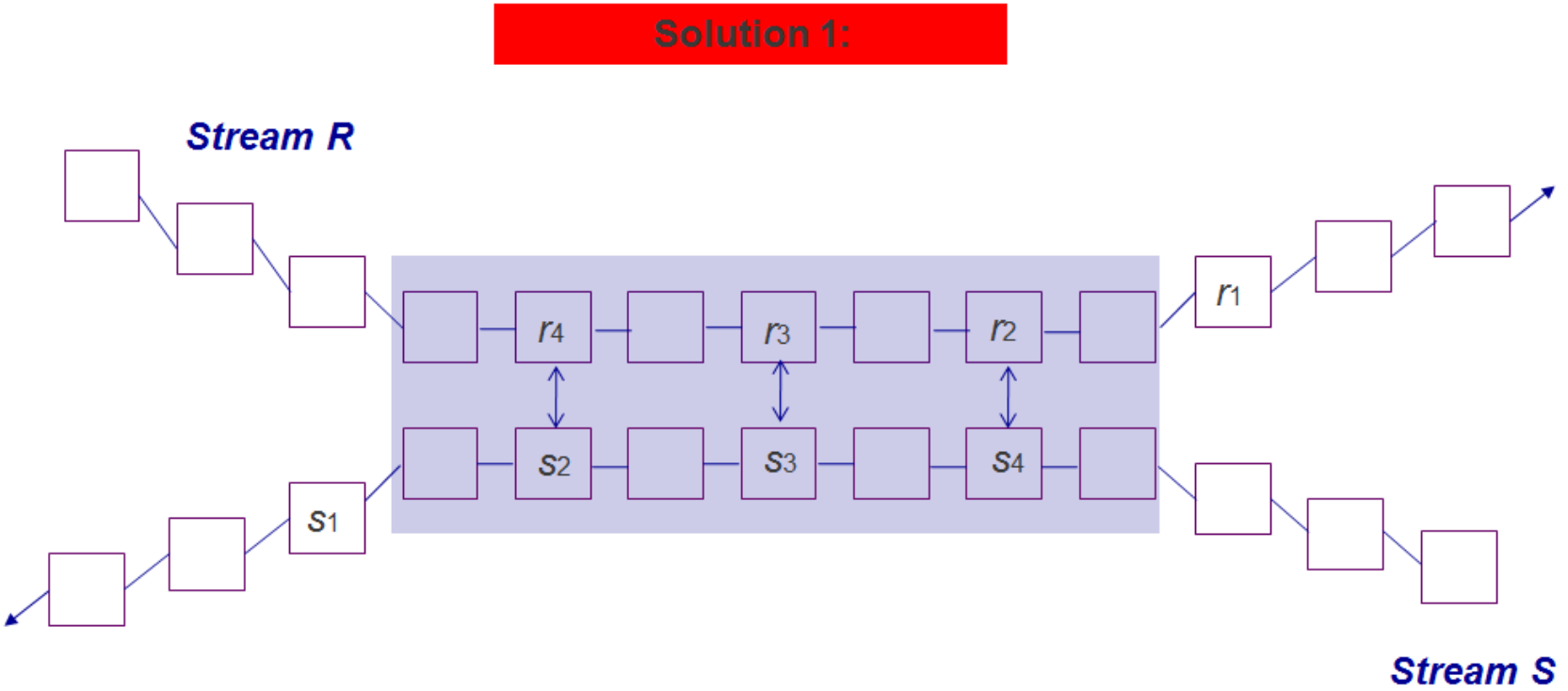
Solution 1:

Main Idea: Alternating basic window must be left empty in the stream.

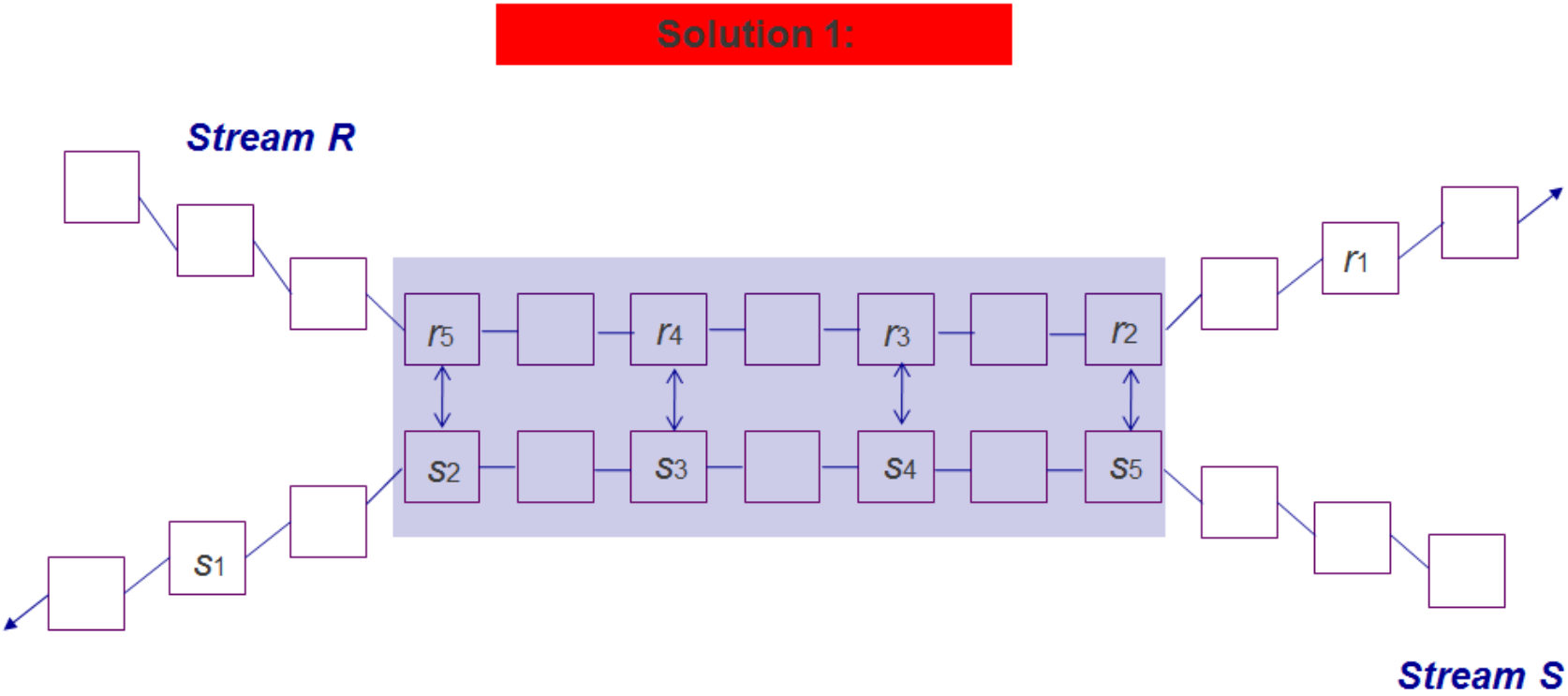


- The main window consisting of basic windows is divided into two categories of basic windows (one empty, one with value).
- Each basic window will be used in an alternate fashion. When the streams move, they will move to the empty basic window, and hence no missing handshake will occur.

Handshake Join (Solution 1)

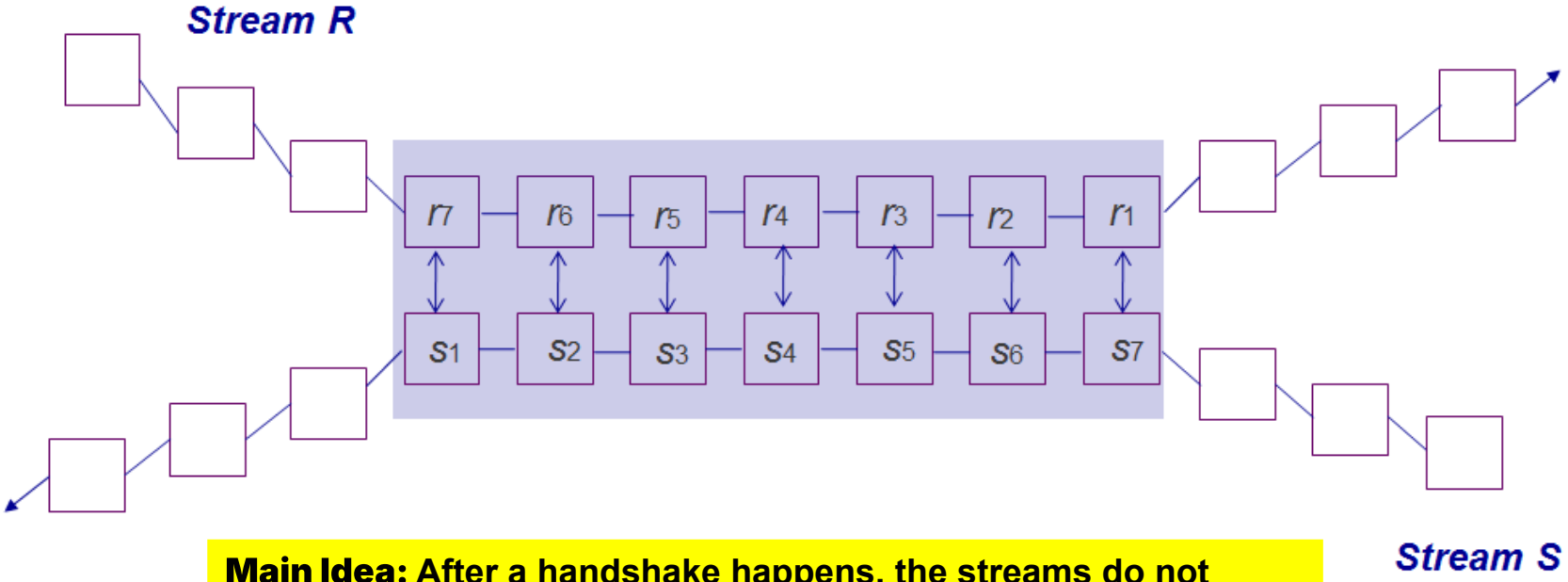


Handshake Join (Solution 1)



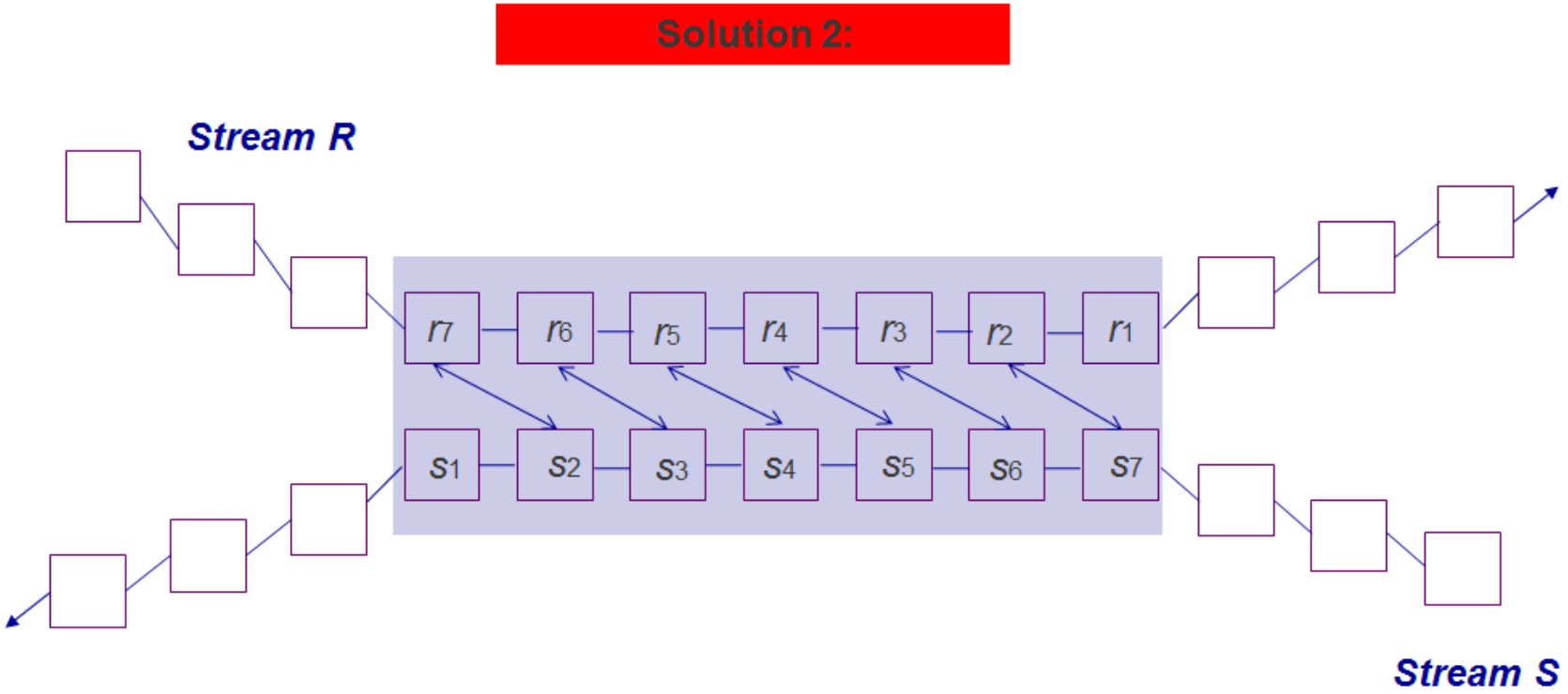
Handshake Join (Solution 2)

Solution 2:

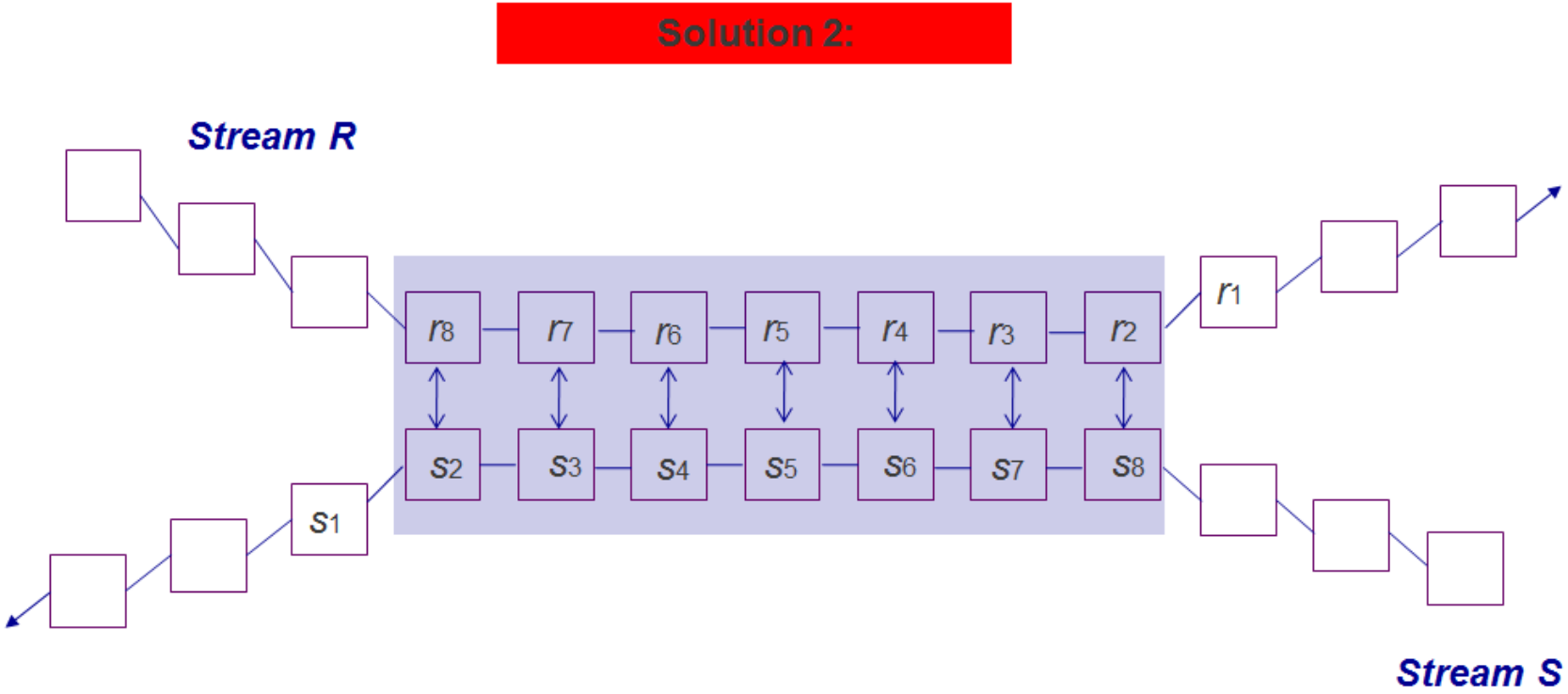


Main Idea: After a handshake happens, the streams do not move forward, but each tuple in the stream performs a handshake with the next tuple, and then after than both streams move forward.

Handshake Join (Solution 2)



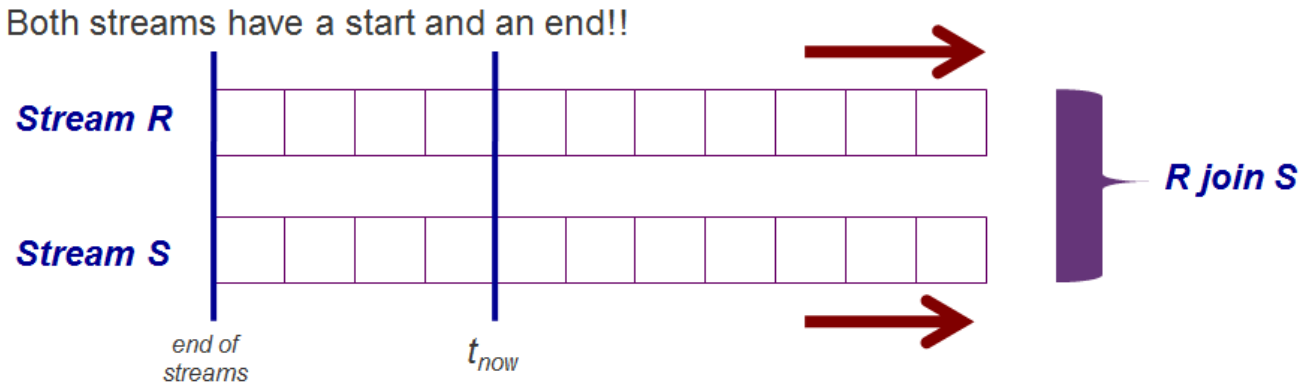
Handshake Join (Solution 2)



This week

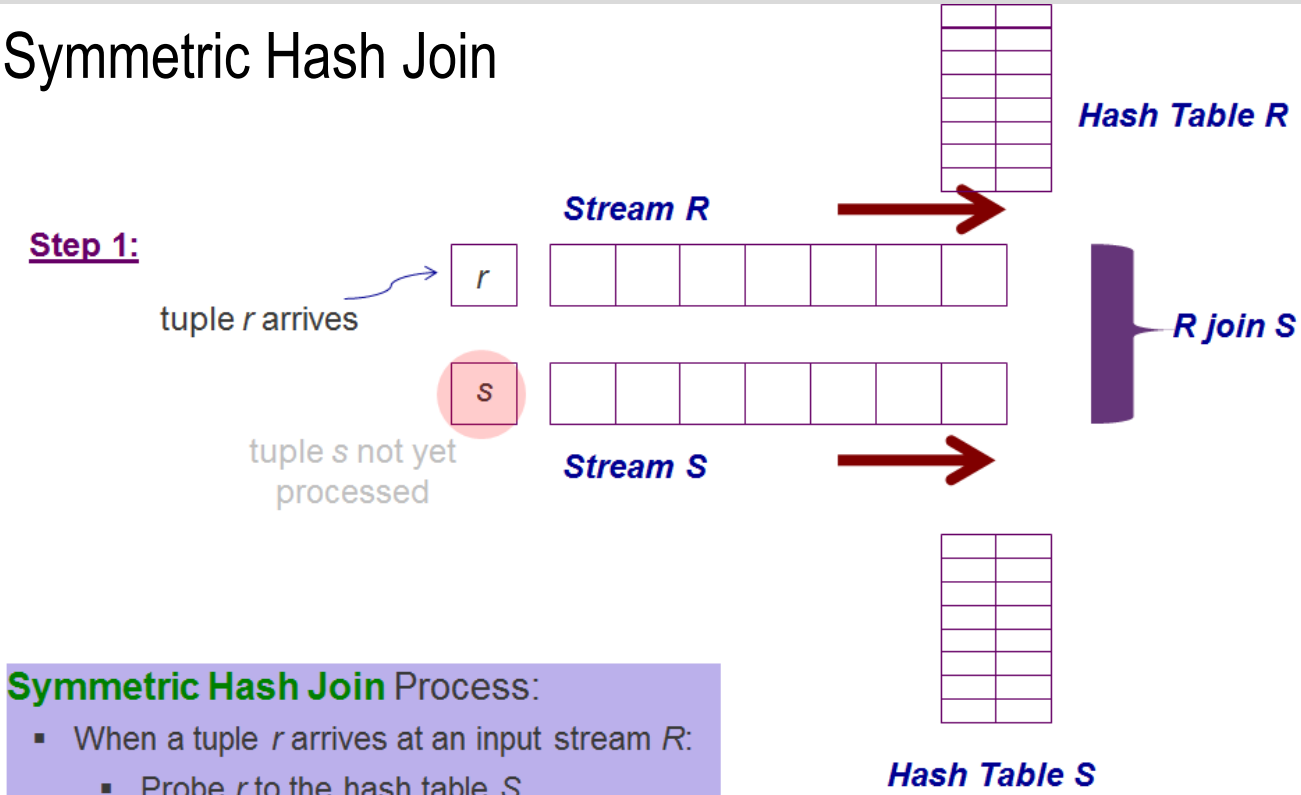
- Overview of Stream join
- Time based window stream join (Unbounded)
  - Tuple slide
  - Time slide
- Tuple based window stream join (Unbounded)
- Bounded stream join
  - How to join two data streams that has end?
    e.g. data from railway sensor, when train stops, data is not streaming

Bounded Stream Join



- No window ➔ it's easy…
- The semantic is hence the same as that of relational join
- It is also called a "Pipelining Join"
- Processing options:
  - Offline (the same as relational join processing) → Wait until all data come in, and perform traditional relational join processing
  - Online (this is pipelining join) → Perform join while data is streaming

Symmetric Hash Join



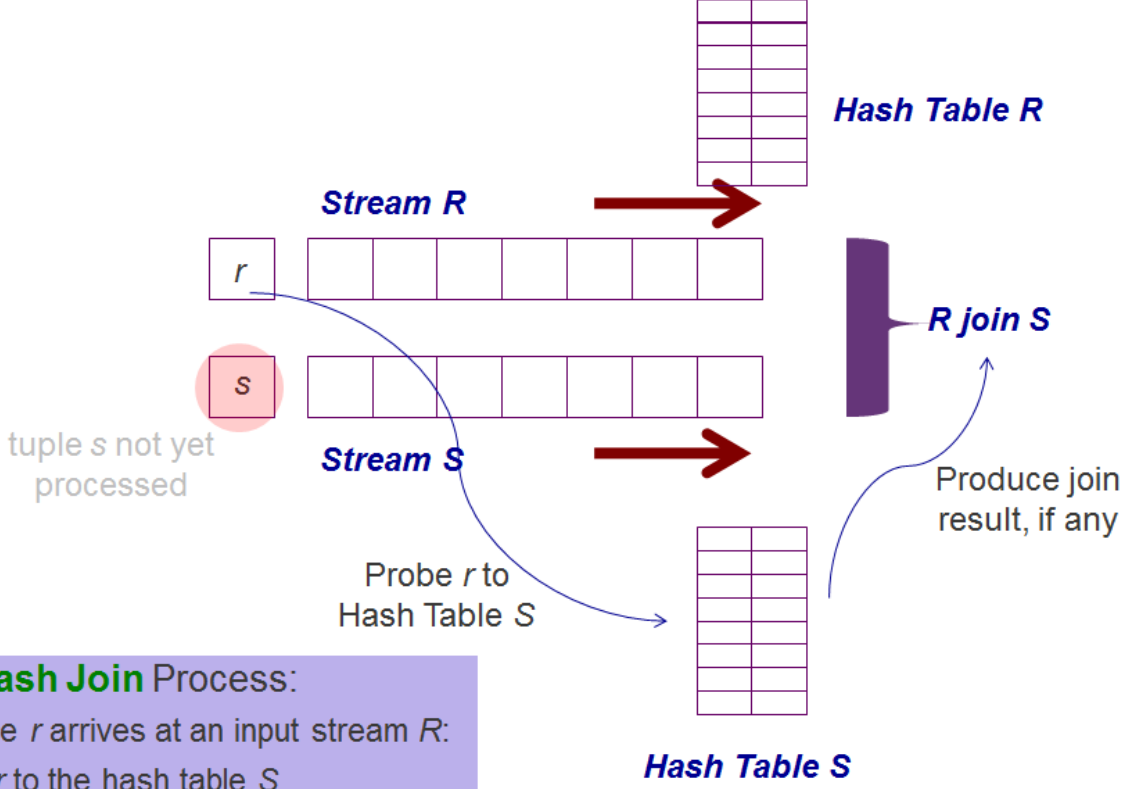
Symmetric Hash Join Process:

- 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*

How is it different from window-based join for unbounded stream?

- Window-based join: Remove entries from hash table when tuple expire
- Bounded stream join:
  - No removal of entries from hash table
  - Hash tables R & S will hold entire stream data
  - because stream will end at some point

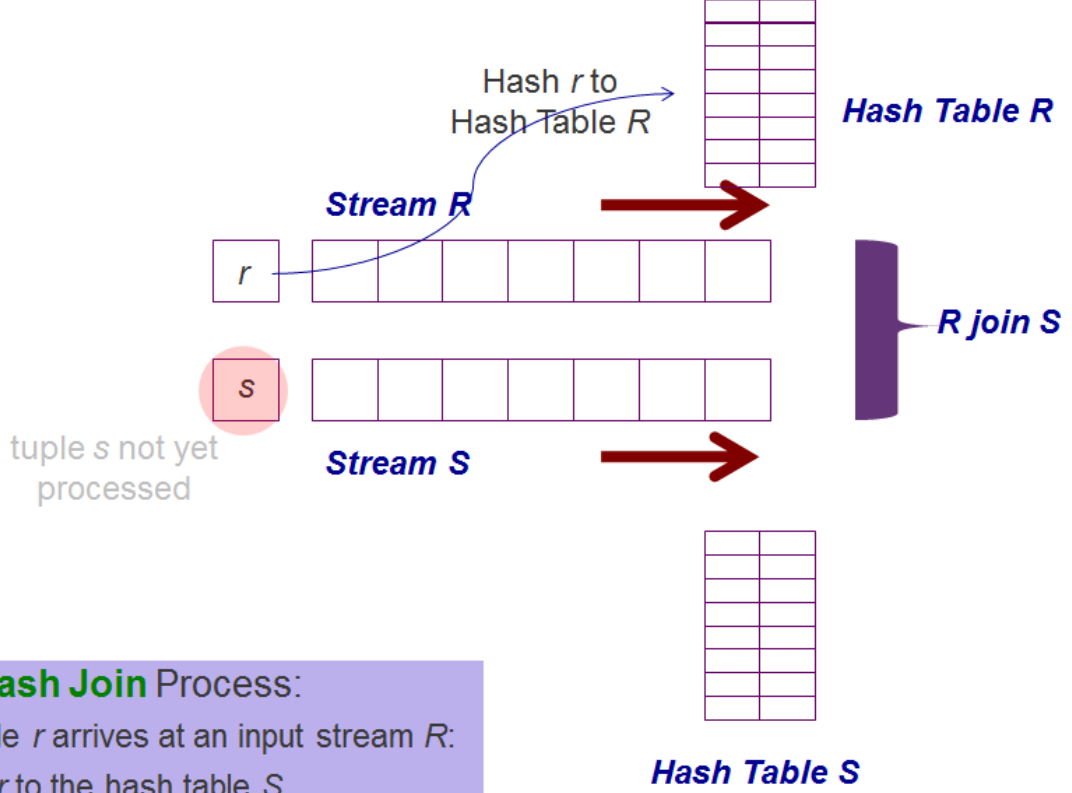
Symmetric Hash Join



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*

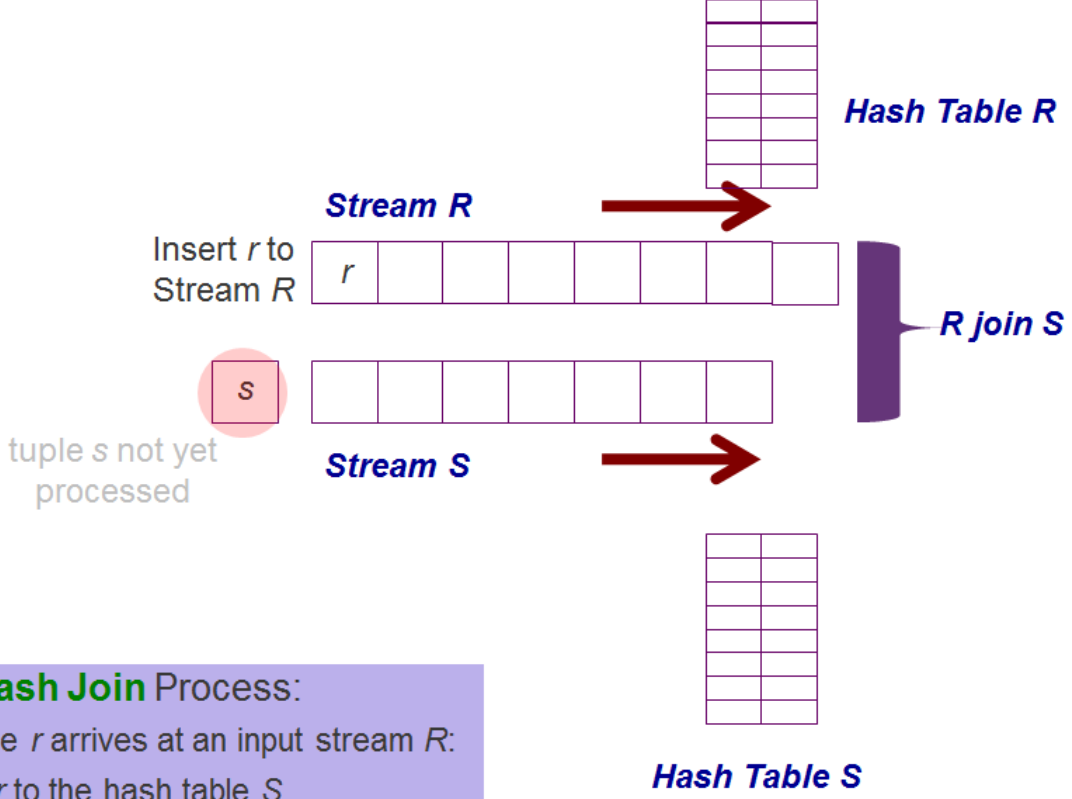
Symmetric Hash Join



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

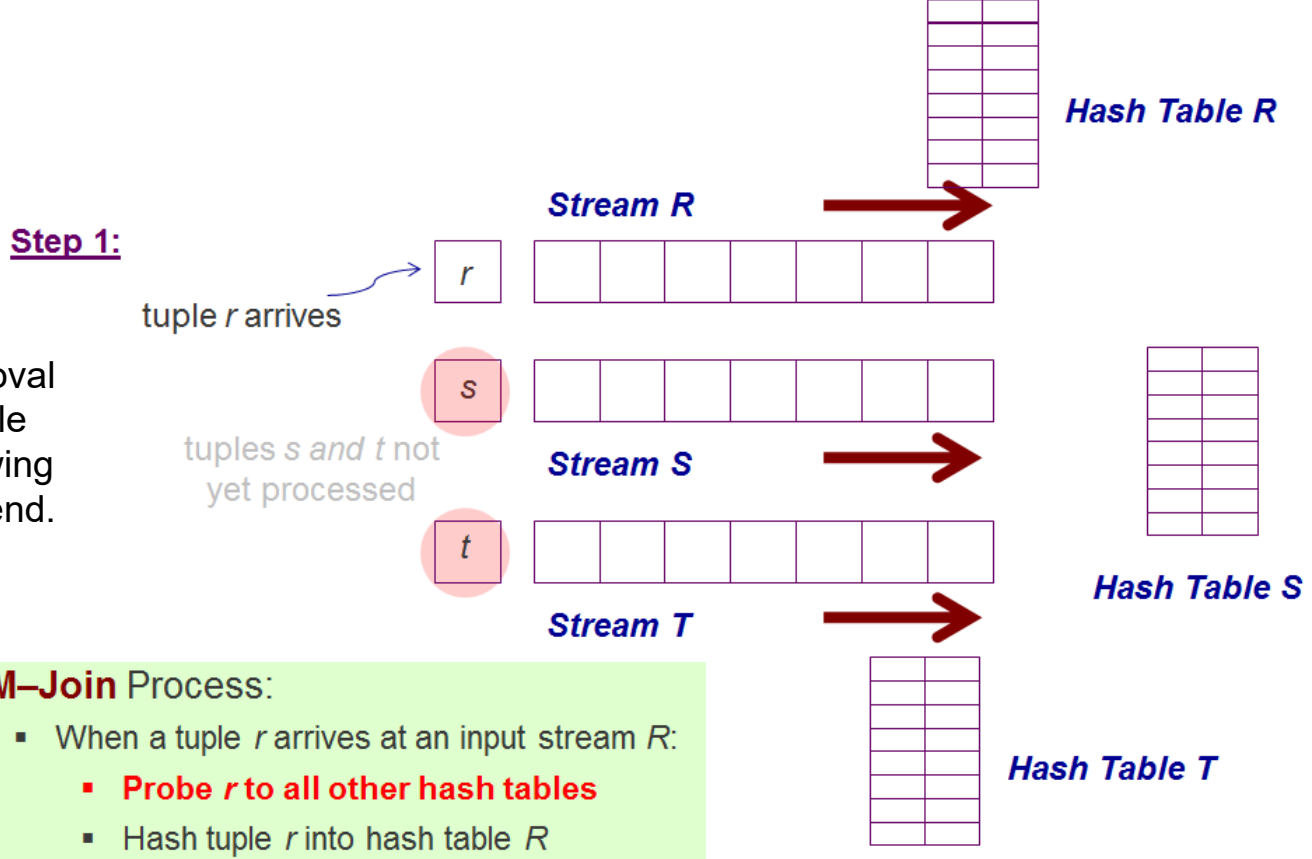


Symmetric Hash Join Process:

- 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

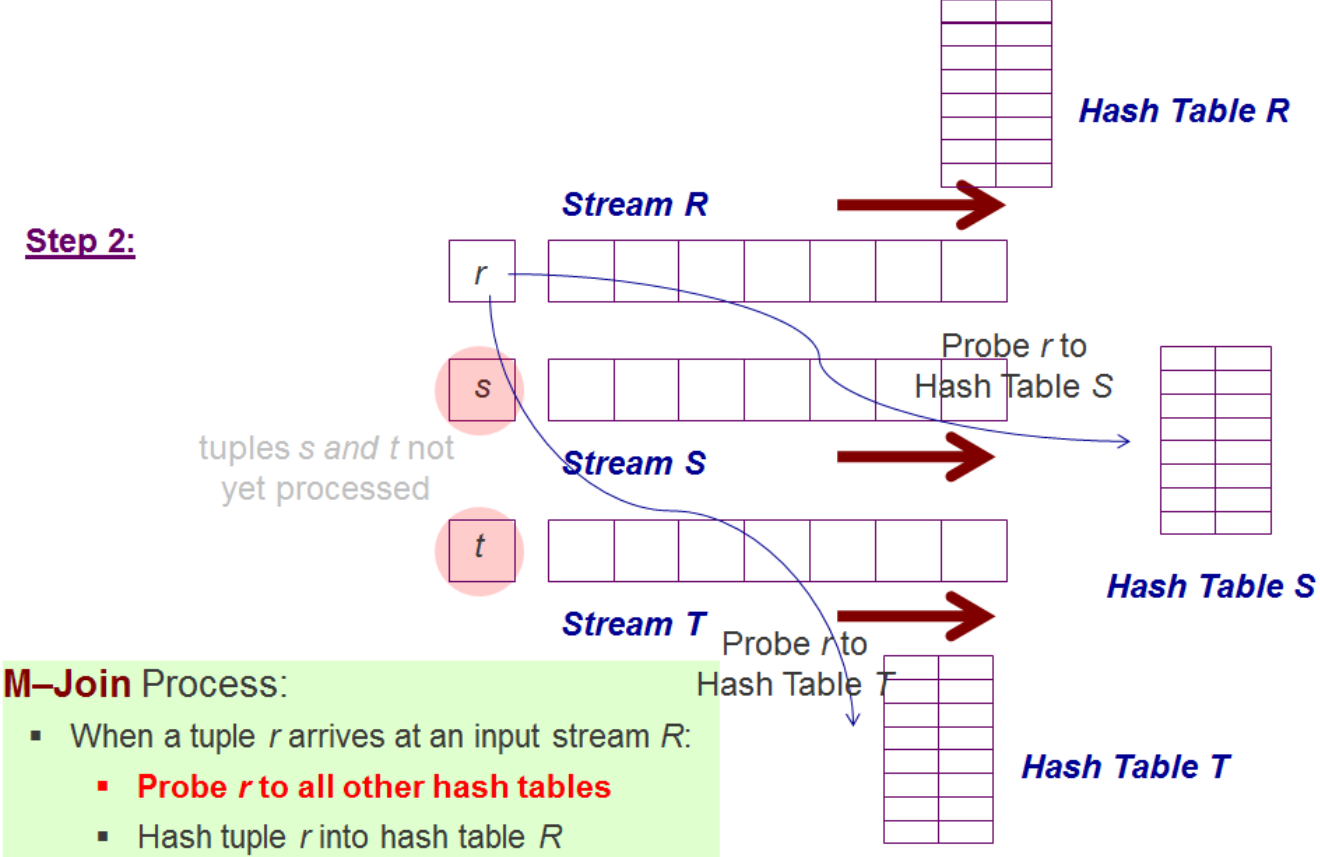
Similarly for M-join, no removal of entries from the hash table because there is no windowing and the stream is going to end.



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*

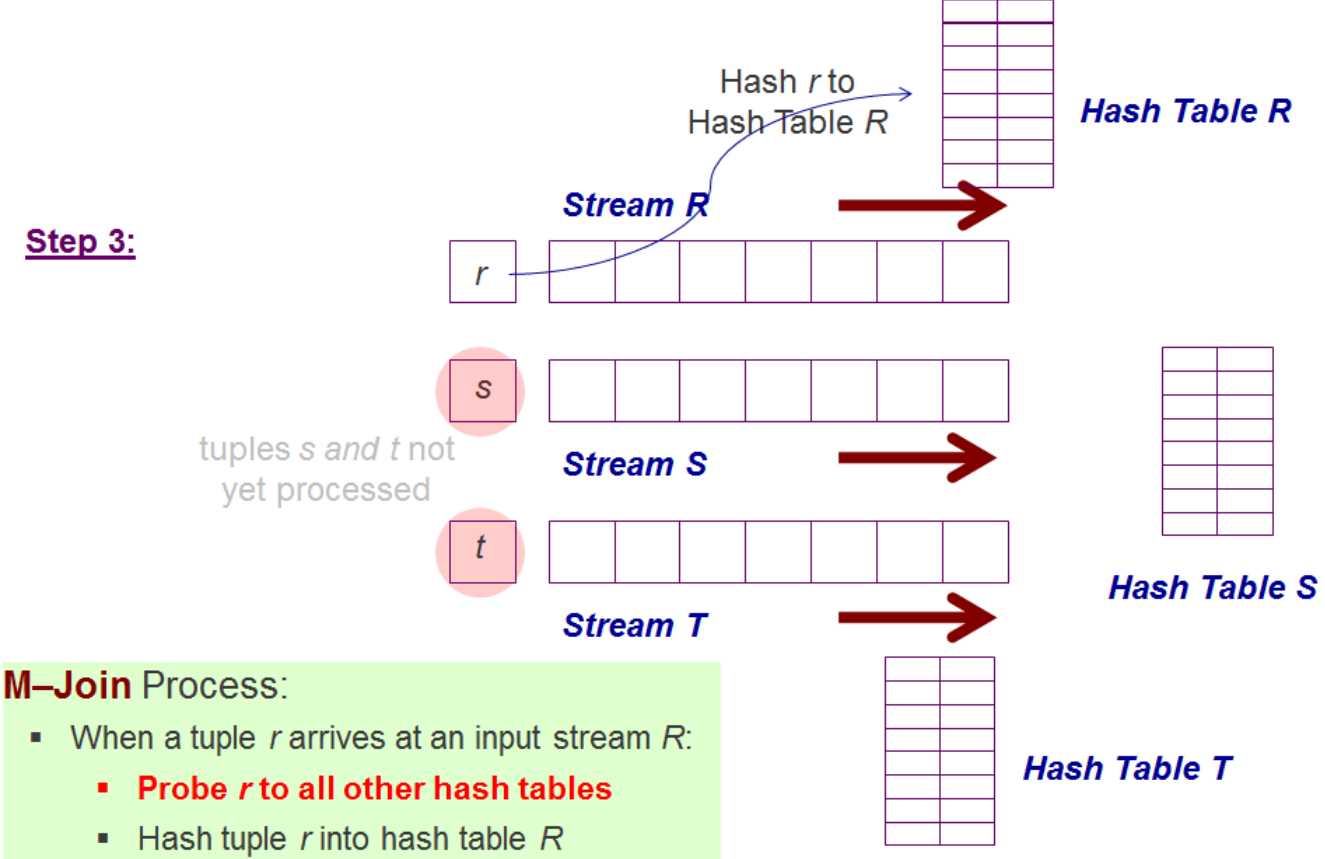
M-Join



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*

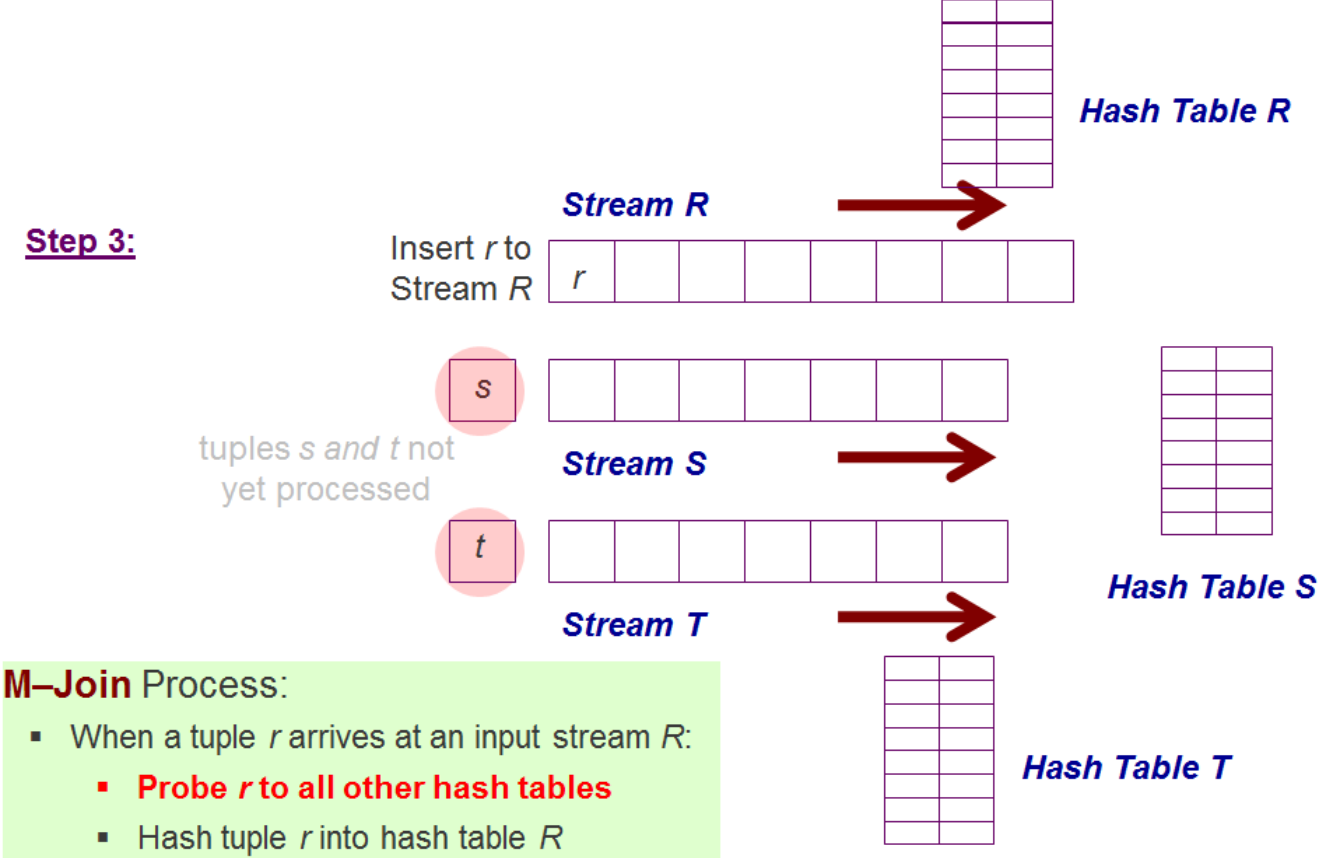
M-Join



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*

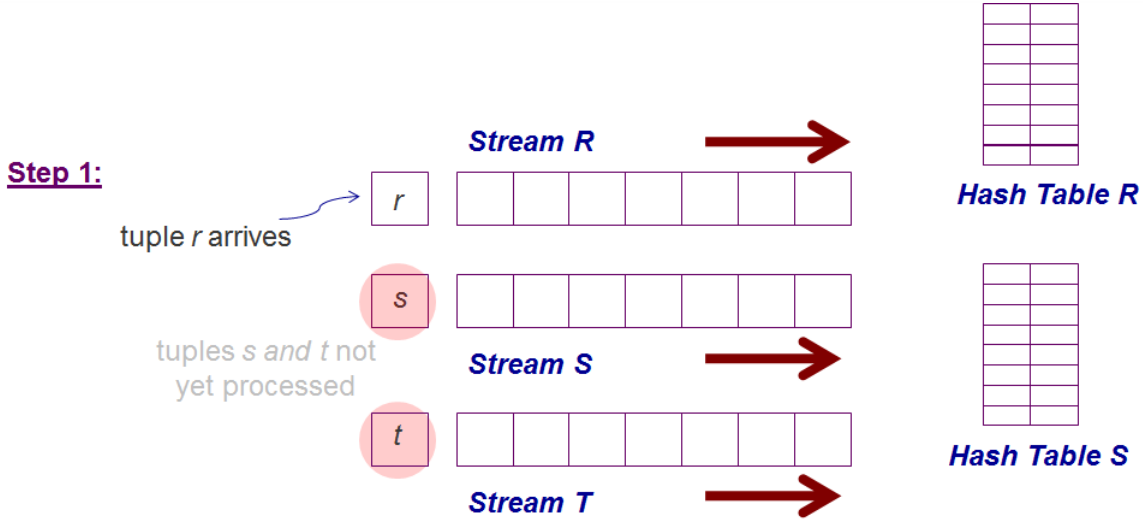
M-Join



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*

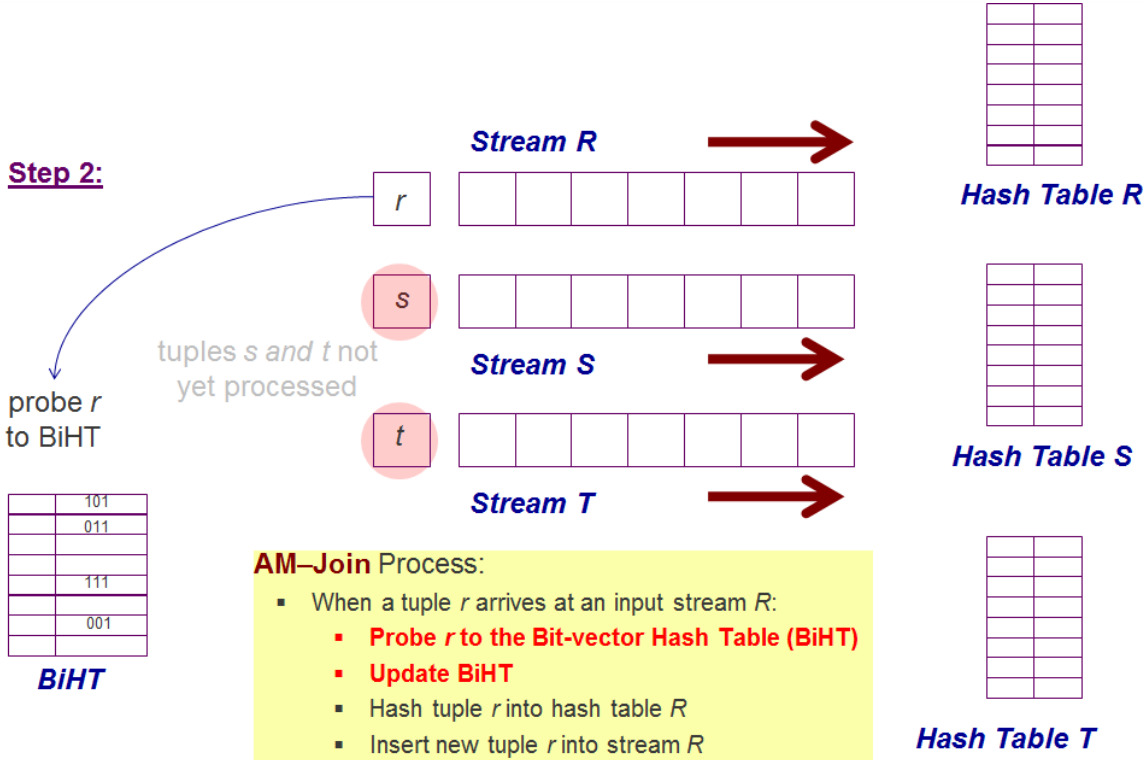
AM-Join



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*

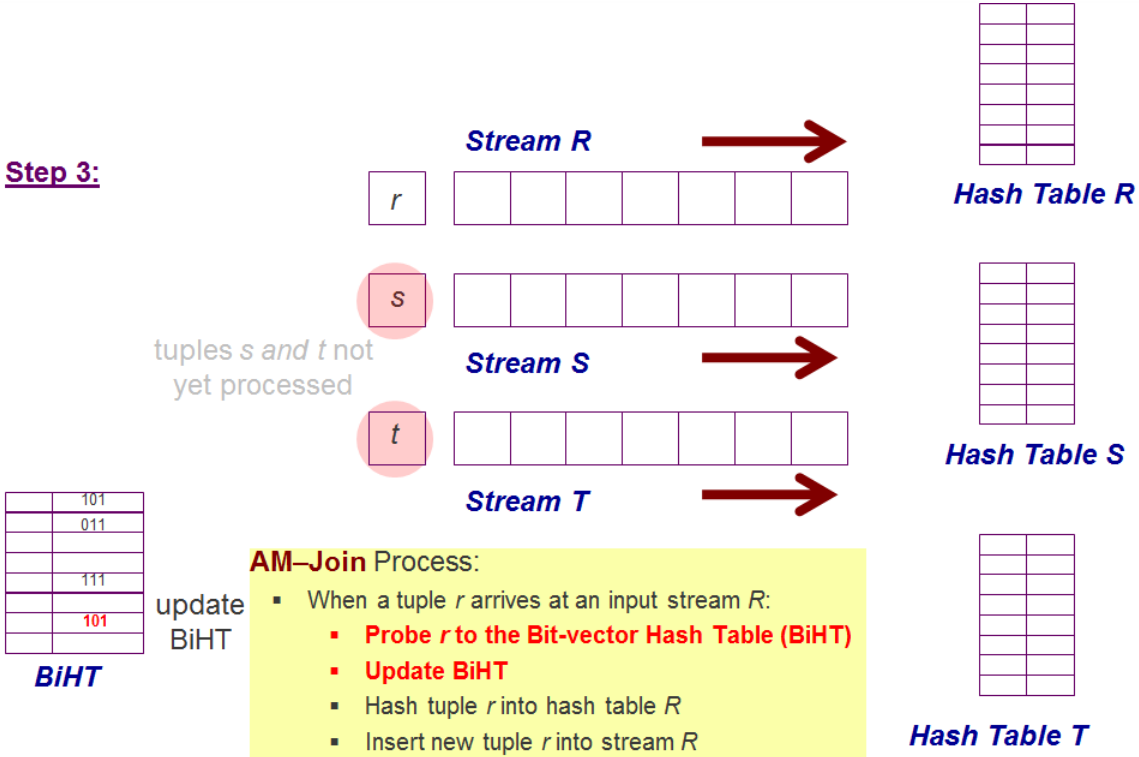
AM-Join



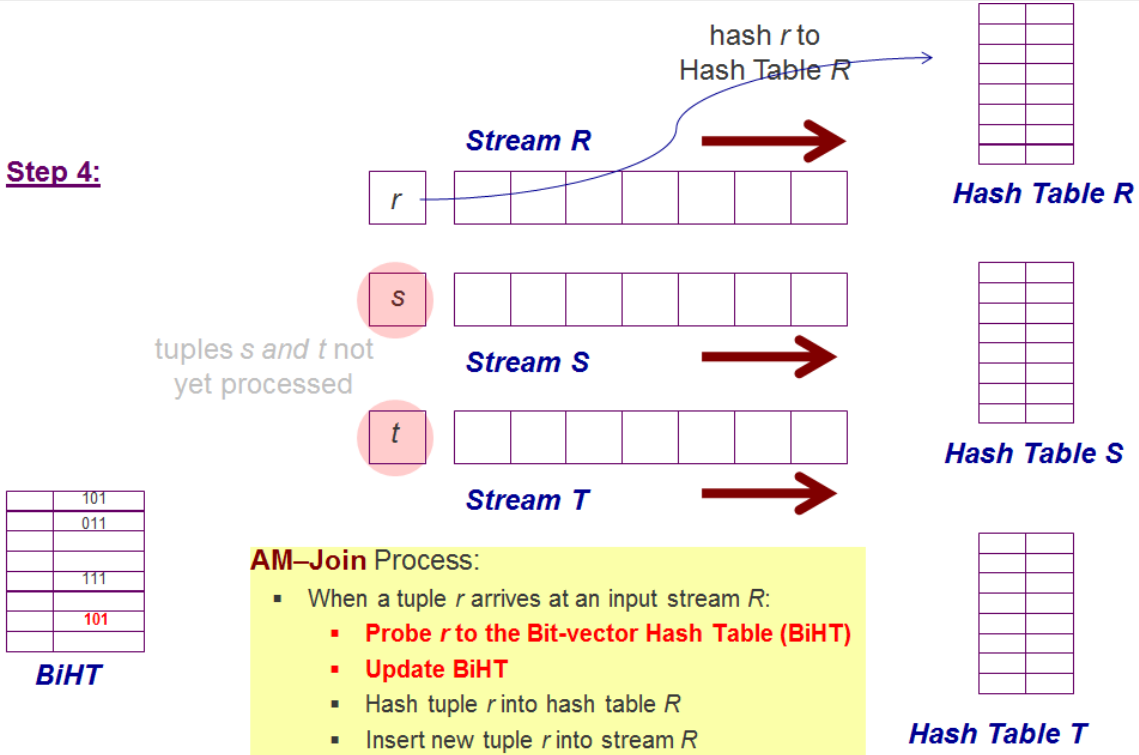
AM–Join Process:

- 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*

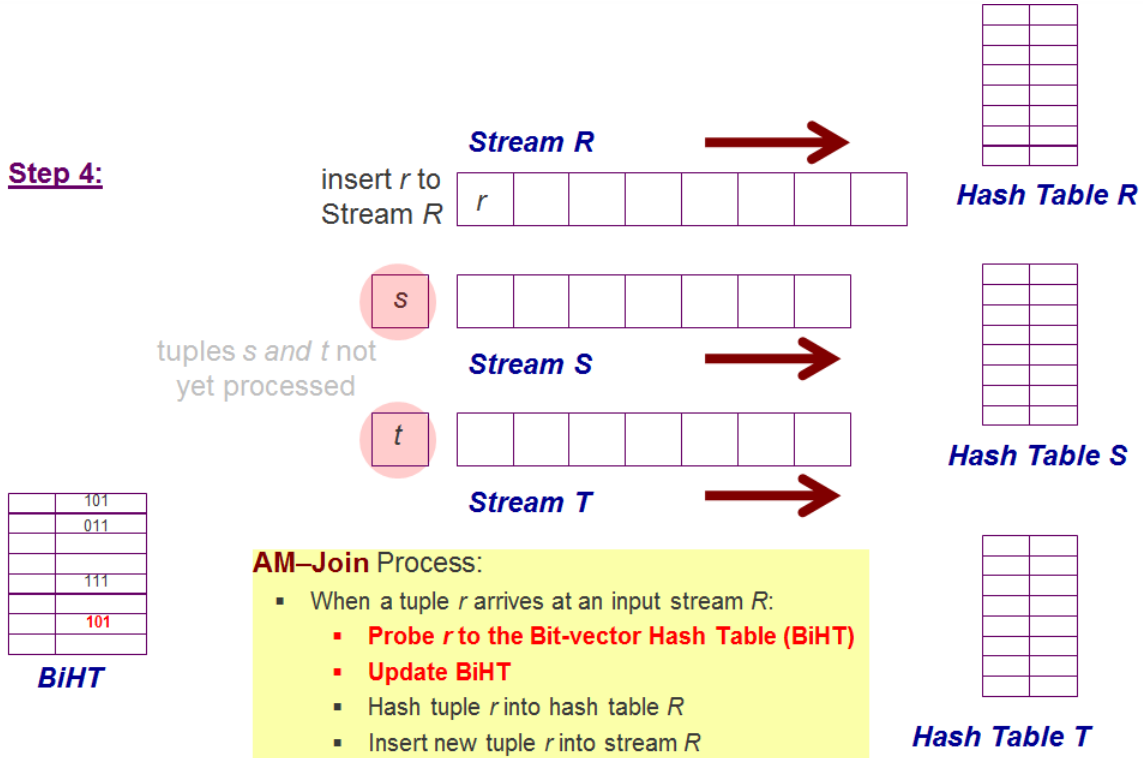
AM-Join


Step 3:
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
update BiHT
Hash Table T
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

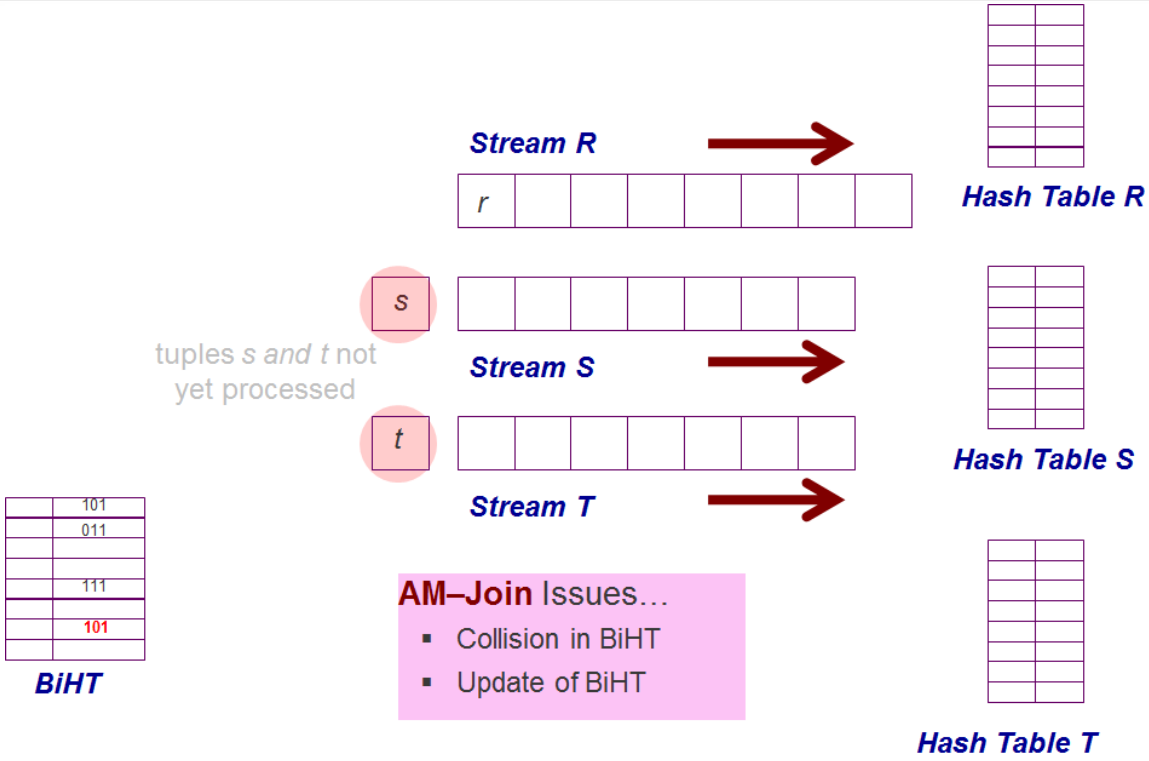
AM-Join


Step 4:
hash r to
Hash Table R
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

AM-Join

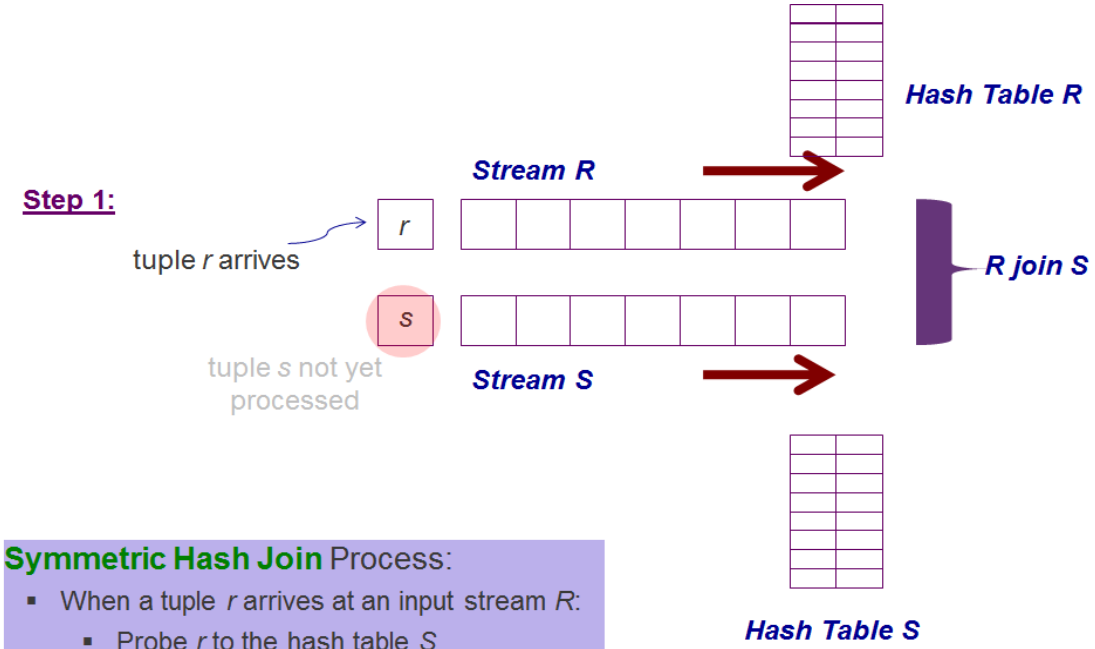

Step 4:
insert r to Stream R
Stream R
r
Hash Table R
s
tuples s and t not yet processed
Stream S
Hash Table S
t
Stream T
101
011
111
101
BiHT
AM–Join Process:
When a tuple r arrives at an input stream R:
Probe r to the Bit-vector Hash Table (BiHT)
Update BiHT
Hash tuple r into hash table R
Insert new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

AM-Join


Stream R
r
Hash Table R
s
tuples s and t not yet processed
Stream S
Hash Table S
t
Stream T
Hash Table T
101
011
111
101
BiHT
AM–Join Issues…
Collision in BiHT
Update of BiHT

DEMO

- 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*

Thank You

MONASH University

Questions?