# NiagaraCQ: A Scalable Continuous Query System for Internet Databases

M.L.Narasimham

## Presentation Outline

- What's NiagaraCQ
- General strategy of incremental group optimization
- Query split scheme with materialized intermediate files
- Incremental grouping of selection and join operators
- Experimental Details

## Continuous Queries

- A triple ( Q, A, Stop)
- Example

  *Inform me when ever the price of Dell  stock drops by more than 5%*
- A broad Classification
  - Change Based
  - Timer Based

## NiagaraCQ

- A CQ system for the Internet
- Continuous Queries on XML data sets
- Scalable CQ processing
- Incremental group optimization
- Handles both change based and timer based queries in a uniform way

## NiagaraCQ command language

- Creating a *CQ*

  **Create** *CQ_name*
  *XML-QL query*
  **Do** *action*
  { **START** *start_time*} { **EVERY** *time_interval*}
  { **EXPIRE** *expiration_time*}

- Delete *CQ_name*

# Advantages of Grouping

- Group optimization has the following benefits:
  - Grouped queries can share computations.
  - Common execution plans of grouped queries can reside in memory, significantly saving on I/O costs compared to executing each query separately.
  - Grouping makes it possible to test the “firing” conditions of many continuous queries together, avoiding unnecessary invocations.

Can this traditional grouping technique be applied into Continuous Query directly?

NO!!!

## *Why not?*

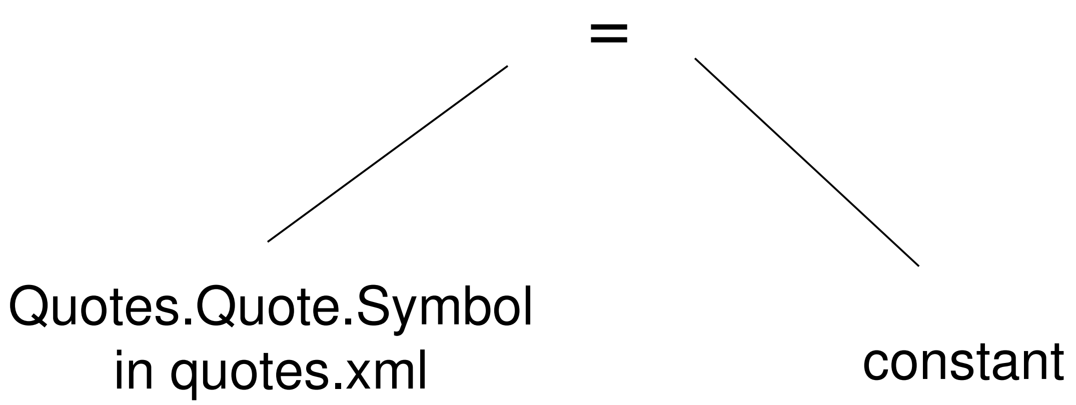
- Previous group optimization efforts focused on finding an optimal plan for a small number of similar queries.
- Not applicable to a continuous query system for the following reasons:
  - Computationally too expensive to handle a large number of queries.
  - Not designed for an environment like the web where CQ s are added or removed dynamically.

# Incremental group optimization

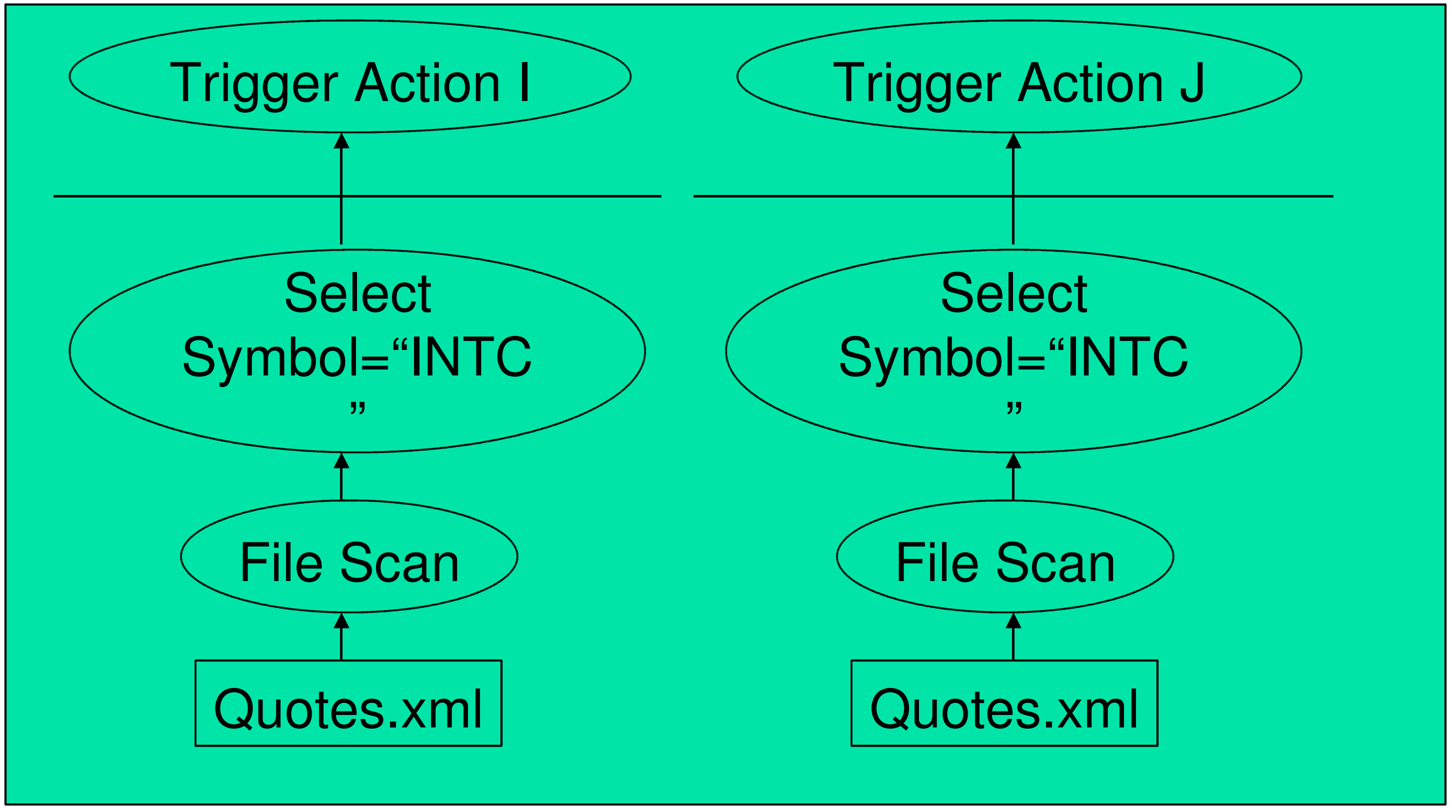
- Expression Signature
  - Query examples

*Where <Quotes><Quote><Symbol>INTC</></></>*
*element_as $g in “http://www.stock.com/quotes.xml”*
*construct $g*

*Where <Quotes><Quote><Symbol>MSFT</></></>*
*element_as $g in “http://www.stock.com/quotes.xml”*
*construct $g*



# Query plans


Trigger Action I
Trigger Action J
Select Symbol="INTC"
Select Symbol="INTC"
File Scan
File Scan
Quotes.xml
Quotes.xml

## Group

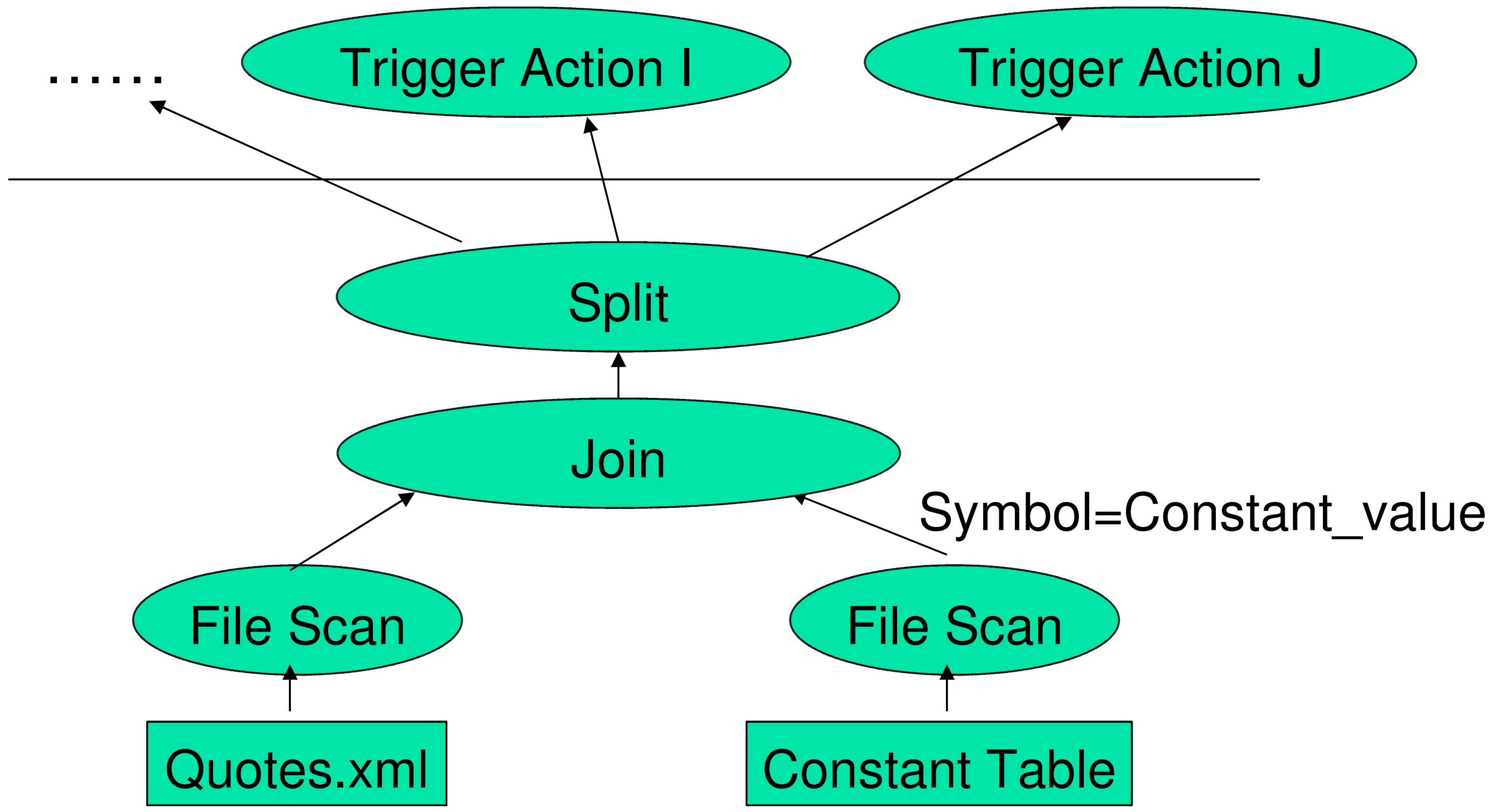
- Group Signature

  Common Signature of all queries in the group

- Group constant table

| Constant_value | Destination_buffer |
|---|---|
| …. | …. |
| INTC | Dest.i |
| MSFT | Dest.j |
| …. | …. |

# Group plan


……
Trigger Action I
Trigger Action J
Split
Join
Symbol=Constant_value
File Scan
File Scan
Quotes.xml
Constant Table

# Incremental Grouping Algorithm

- When a new query is submitted
  - *If the expression signature of the new query matches that of existing groups*
    - *Break the query plan into two parts*
    - *Remove the lower part*
    - *Add the upper part onto the group plan*
  - *else create a new group*

# query-split scheme

- Incremental group optimization scheme employs a query-split scheme.
  - After the signature of a new query is matched, the sub-plan corresponding to the signature is replaced with a scan of the output file produced by the matching group.
  - Optimization process then continues with the remainder of the query tree in a bottom-up fashion until the entire query has been analyzed.
  - If no group “matches” a signature of the new query, a new query group for this signature is created in the system.
  - Thus, each continuous query is split into several smaller queries such that inputs of each of these queries are monitored using the same techniques that are used for the inputs of user-defined continuous queries.

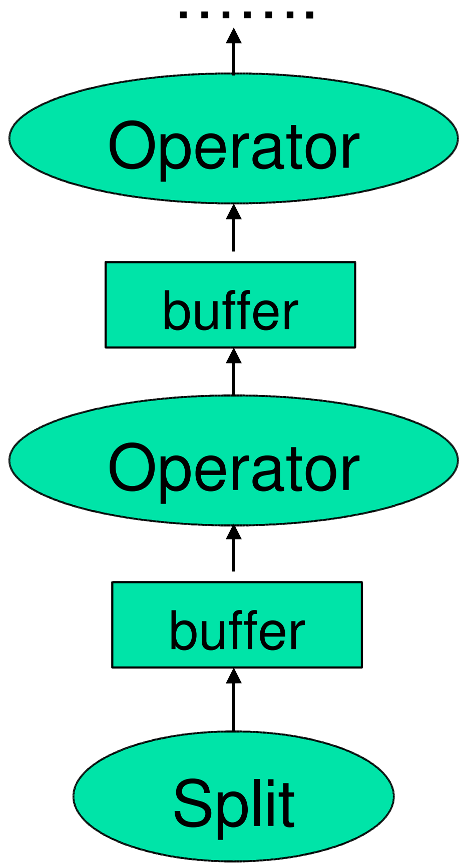
## query-split scheme (contd..)

- Advantages for query-split scheme:
  - Main advantage is that it can be implemented using a general query engine with only minor modifications.
  - Anther advantage is that the approach is very scalable.

## Buffer design

- The destination buffer for the split operator is needed.
- Pipelined scheme
- Intermediate scheme

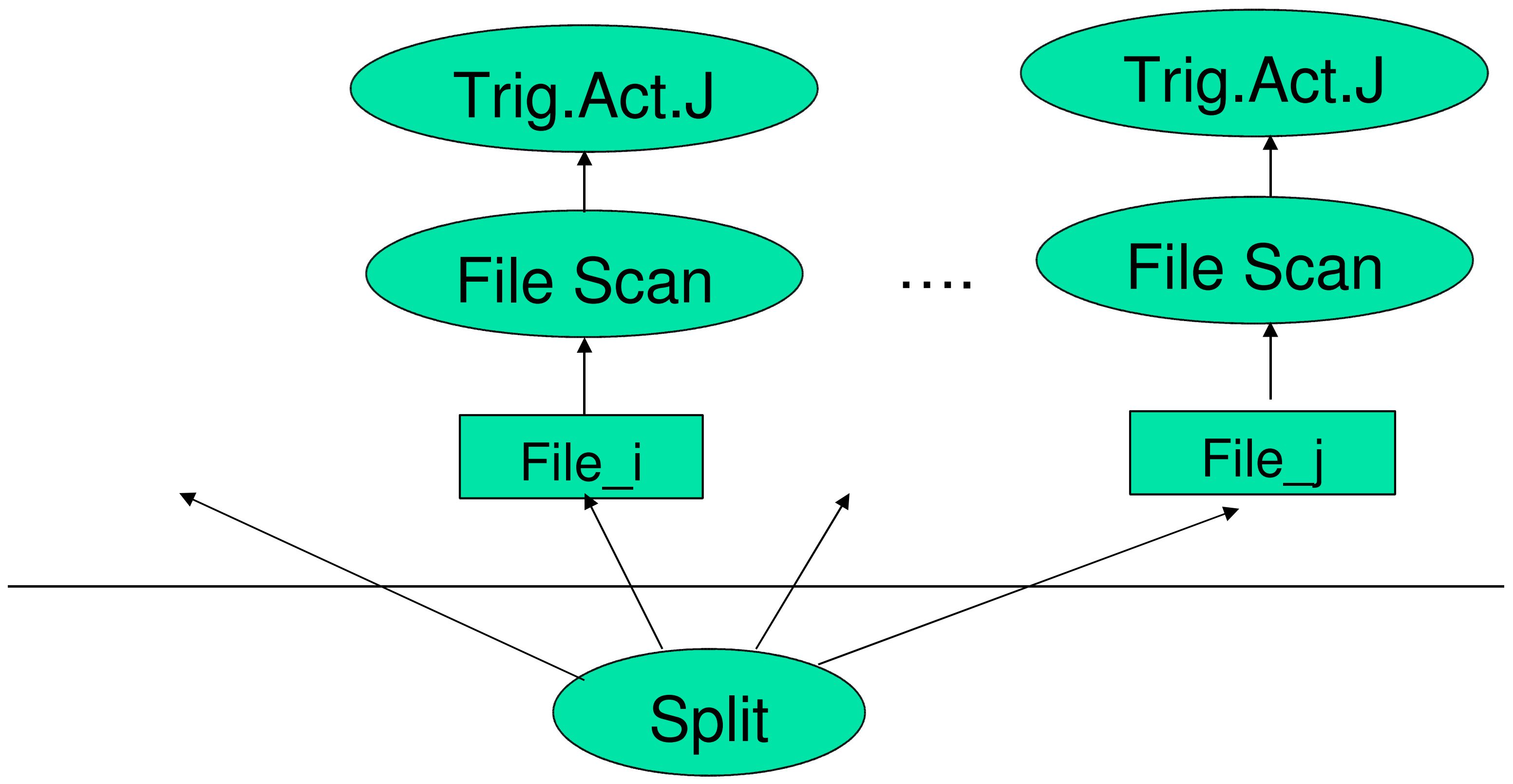
## Pipelined scheme



## Disadvantage of pipeline

- Such a scheme doesn’t work for grouping timer-based CQ’s. It’s difficult for a split operator to determine which tuple should be stored and how long they should be stored for.
- A large portion of the query plan may not need to be executed at each query invocation.
- One query may block many other queries.

# Query Split with Materialized Intermediate Files

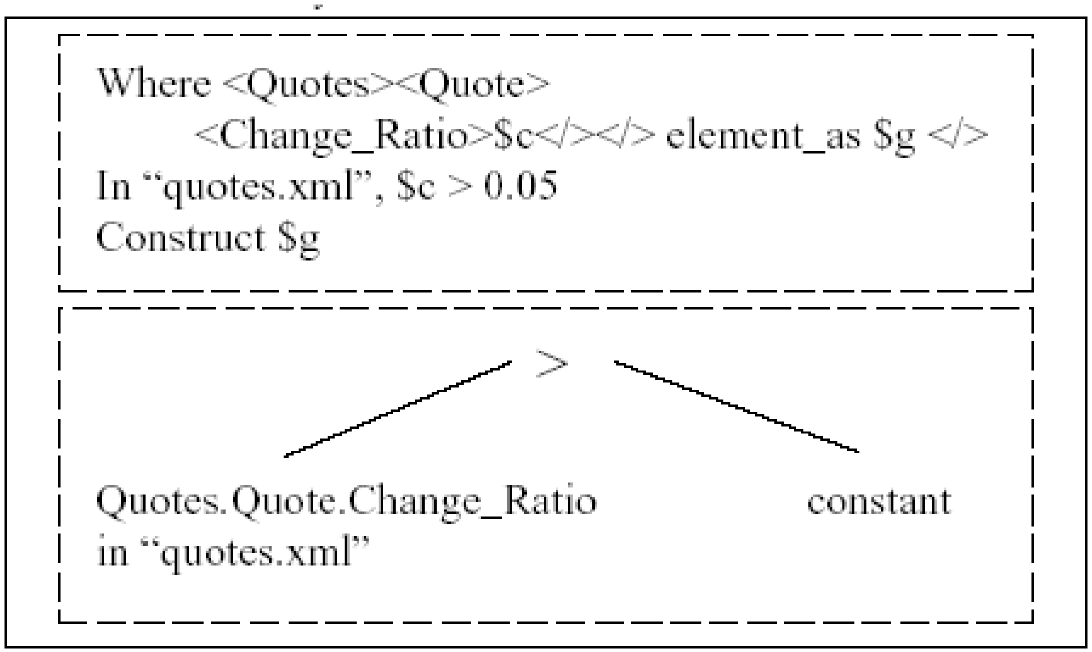


## Advantages of this design

- Every query is scheduled independently. Only the necessary queries are executed.
- This approach handles intermediate files and original data files uniformly.
- The potential bottleneck of pipelined approach is avoided.

# Incremental grouping of Selection predicates

- Format: “Attribute op Constant”
  - Attribute is a path expression without wildcards in it.
  - Op includes “=”, “<”, “>”, because these formats dominate in the selection queries.

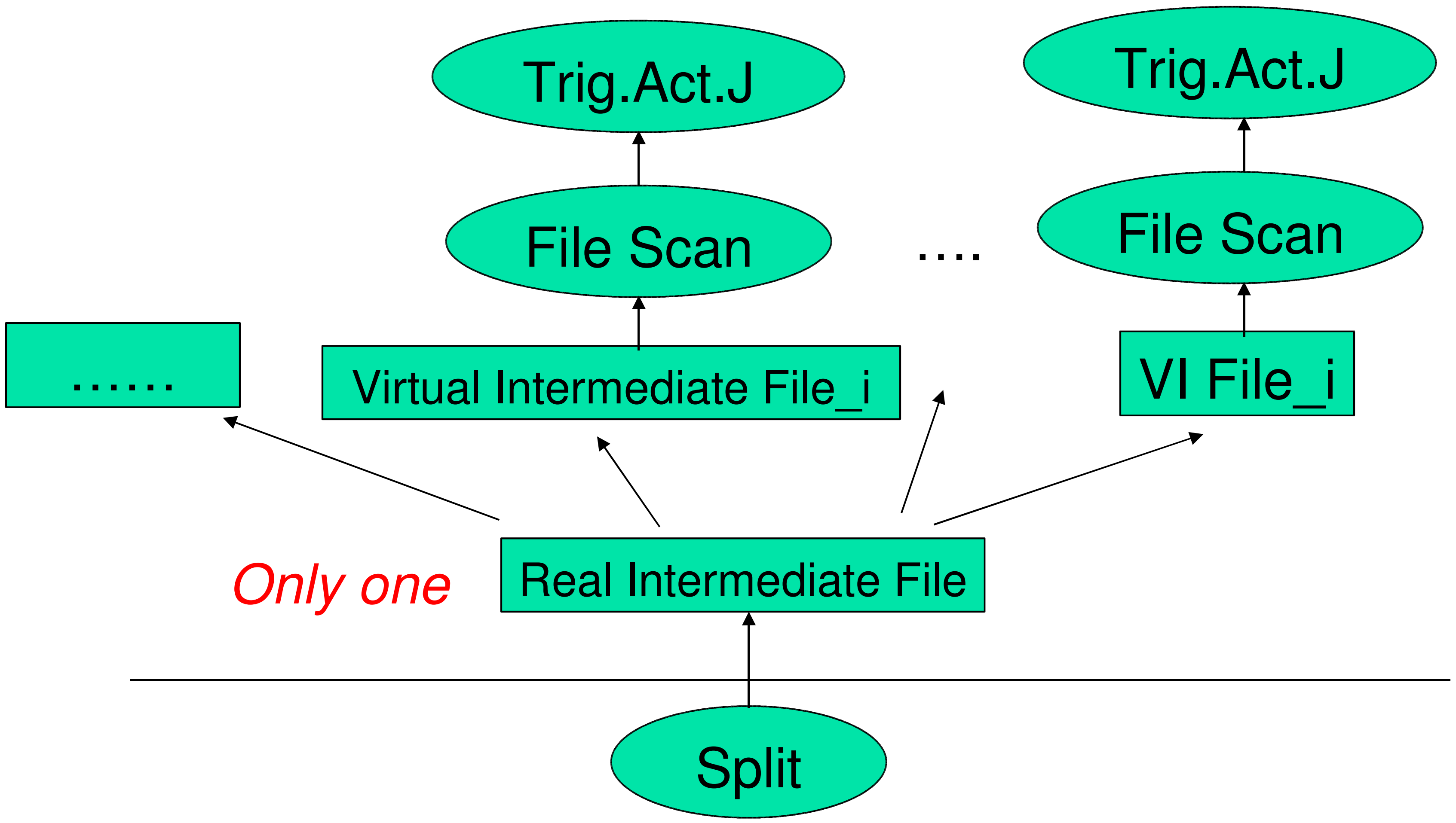


Where <Quotes><Quote>
<Change_Ratio>$c</></> element_as $g </>
In “quotes.xml”, $c > 0.05
Construct $g
>
Quotes.Quote.Change_Ratio
in “quotes.xml”
constant

## Problem for range-query groups

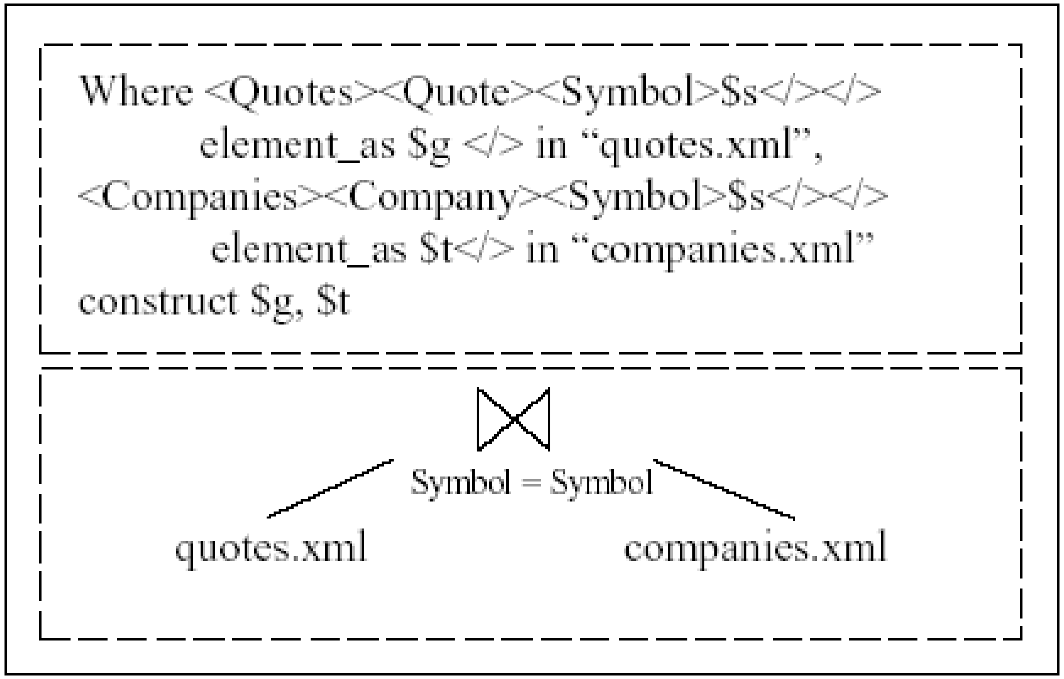
- One potential problem for range-query groups is that the intermediate files may contain a large number of duplicated tuples because range predicate of the different queries might overlap.

## Solution: Virtual Intermediate Files



# Incremental grouping of Join Operators

- Join operators are usually expensive, sharing common join operations can significantly reduce the amount of computation.



```
Where <Quotes><Quote><Symbol>$s</></>
          element_as $g </> in "quotes.xml",
<Companies><Company><Symbol>$s</></>
          element_as $t</> in "companies.xml"
construct $g, $t
```

## Queries that contain both join and selection

- Example query :

  *Where <Quotes><Quote><Symbol>$s</>*

  *<Industry>"Computer Service"</></>*

  *element_as $g </> in "quotes.xml",*

  *<Companies><Company><Symbol>$s</></>*

  *element_as $t</> in "companies.xml"*

  *construct $g, $t*

- Where to place the selection operator ?
  - Below the join
  - Above the join

## Join first? Or selection first?

- Advantage
- Disadvantage
- Solution: choose a better one based on a cost model

## Incremental Evaluation

- Incremental evaluation allows queries to be invoked only on the changed data.
- For each file, on which CQ’s are defined, NiagaraCQ keeps a “delta file” that contains recent changes.
- Queries are run over the delta files whenever possible instead of their original files.
- A time stamp is added to each tuple in the delta file. NiagaraCQ fetches only tuples that were added to the delta file since the query’s last firing time.

## Memory Caching

- Caching is used to obtain good performance with a limited amount of memory.
- Caches query plans, system data structures, and data files for better performance.

## What should be cached?

- Grouped query plans. We assume that the number of query groups is relatively small.
- Recently accessed files, delta files.
- The event list for monitoring the timer-based events. But it can be large, so we keep only a “time window” of this list

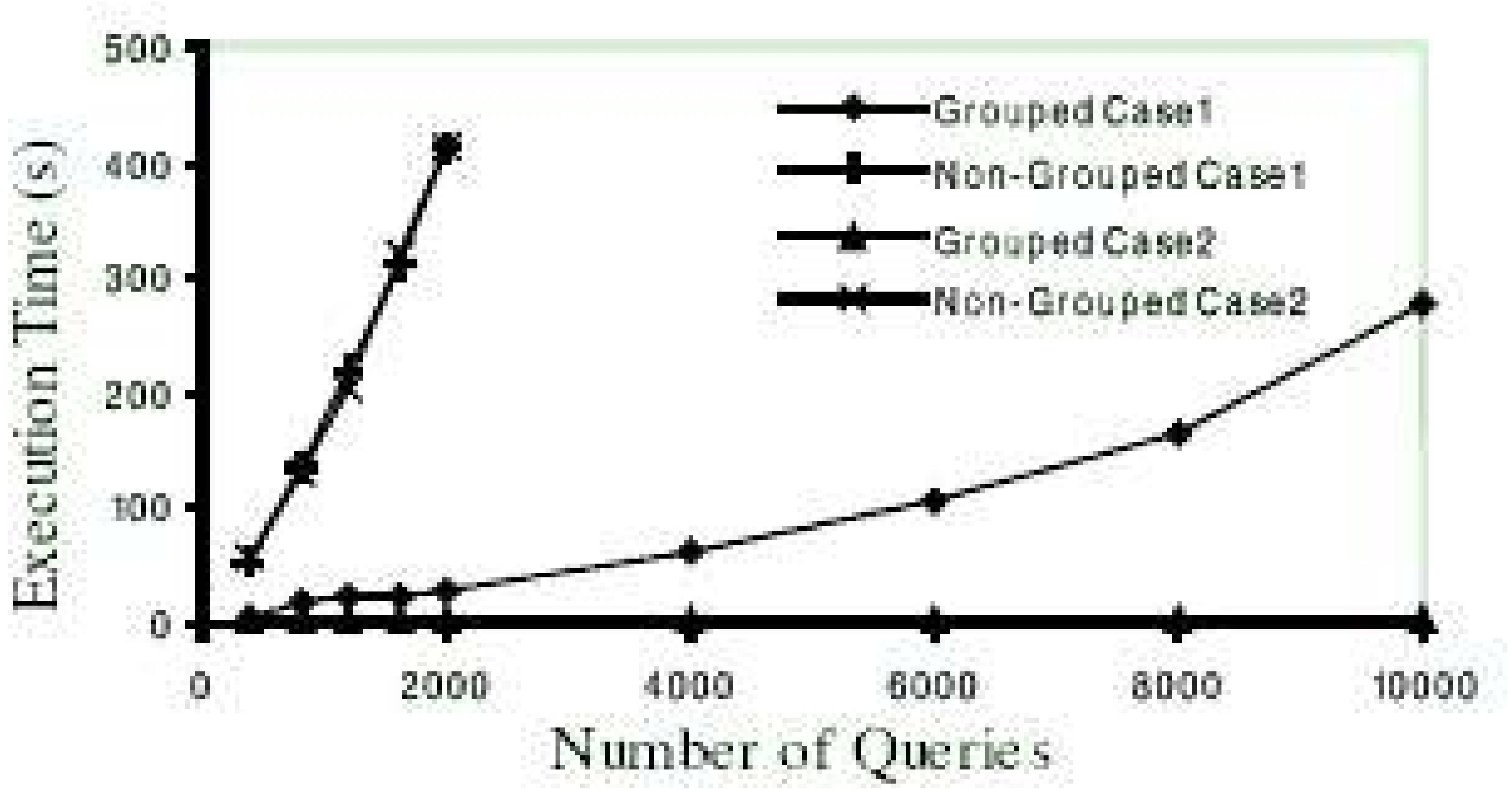
    e.g : events in next 24 hrs

# Experimental Results

*Example query :*

*Where <Quotes><Quote><Symbol>"INTC"</></>*

*element_as $g </> in "quotes.xml", construct $g*



## Conclusion

- Incremental grouping methodology makes group optimization more scalable
- Grouping method using query split scheme requires minimal changes to general purpose query engine.
- Incremental evaluation and caching techniques make NiagaraCQ more scalable.

Thank You.. ❏