Multi query optimization support for CEP

Background

In contrast to normal stream processing, CEP generally deploys stateful operators (SEQ, NEG etc.). Scaling CEP has several dimensions:

1. Handling large number of queries.(Focus of this report)
2. Queries that needs large working memory.
3. Handling a complex query that might not fit within a single machine.
4. Handling large number of events.

Motivation

This report focused on analysis the techniques for "Handling large number of queries."

Noted: Intuitively, one can build CEP engine as "share nothing" architecture(like what Esper does), and execute many queries seprately in different engine instances, and intuitively, just replicate every event to those engines, a more fine-grand way is to leverage on data partition strategy. This will require data partition mechanism(see another report), in this report, we however are interested in how MQO can be done in sharing architecture.

Some possible challenges:

1. How to efficiently support data sharing between multiple concurrently continuously running queries. (And when can we discard the data safely?)
2. How to share the intermediate result efficiently (may across machines).
3. How to allow mix of run-time adding of continuous query.
4. How to guarantee fairness/priority.

1. Multi Query Optimization on traditional databases

1.1 Core idea: Common subexpression Elimination

1. In TODS'88, [Multiple query optimization], The main idea is to identify common subexpressions across the set of running queries. Once common subexpressions are detected, the system can replace the original subqueries with a broader subquery that subsumes all of them or rearrange running queries such that common subexpressions are executed together.
2. In Sigmod'07,[Efficient Exploitation of Similar Sub-expressions for Query Processing], the author introduced a novel technique for detecting common subexpressions which allows MQO to be integrated in production systems. The experiments show that applying MQO across a few tens of queries is extremely efficient.

1.1.1 Extended idea 1: Materialized views

1. In Sigmod'10,[Efficient and Extensible Algorithms for Multi Query Optimization] extends the idea of MQO by introducing **materialized views** to further benefit from commonalities across queries.

> Nevertheless, the additional overhead of maintaining the materialized views limits the applicability of this technique.

1.1.2 Extended idea 2: Cache intermediate results

1. In Proc.DE'01, [Cache-on-Demand:Recycling with Certainty], the author suggested cache intermediate results among queries.
2. In Sigmod'09, [An Architecture for Recycling Intermediates in a Column-Store], the author further improved this suggestion.

> "Caching (recycling) intermediate results of common operators(i.e. a very frequent join), has been shown to increase the performance as it comes with a number of advantages. Nonetheless, result caching cannot be applied to systems with high update loads, as the caches are invalidated very frequently. In order to support high update loads, the proposed algorithm does not rely on result caching at all."

2. Many query optimization on traditional databases (Shared workload optimization)

2.1 Drawback of MQO

1. MQO limits the amount of queries that can share work.
   > (In contrast, the goal of SW systems is to process hundreds to thousands of queries concurrently)
2. Additionally, subexpressions have to be detected every time a new set of queries arrive in the system, because the sharing of the previous set of queries is independent of the sharing of the next set of queries.
   > (In contrast, SWO avoids this limitations by not detecting common subexpressions at all. Instead, rely on identifying common operators across prepared statements.)
3. Further more, SWO has to be executed only when the workload evolves(i.e. more prepared statements are added). This allows SWO to be applied not only to analytical workloads(TPC-H), but also to transnational workloads (TPC-W).

2.2 Shared workload optimization

1. In ICDE'07, [Increasing Buffer-Locality for Multiple Relational Table Scans through Grouping and Throttling], the author uses a cooperative scan operator where each table scan answers for more than one query at the same time, independently of their selection predicates.
2. In VLDB'09, [Predictable Performance for Unpredictable Workloads], allows thousands of queries to share the same scan, while maintaining predictable performance by ***indexing*** the running queries.
3. In VLDB'14 [Shared Workload Optimizaiton], The author pointed out that *load interaction* problem can be caused by individual optimization for each query, especially in multi core architectures.

> "Does not optimise queries individually; Does not look for common predicates; Focuses on running queries concurrently over a pool of **Shared operators**; Consequently, must identify which operators to share; and how to organize these operators into a global access plan."

3. Stream processing system(those potentially support multi-query on CEP)

1. In Sigmod'09 [ZStream: A Cost-based Query Processor for Adaptively Detecting Composite Events] *Zstream* adopt query-plan-tree approach to support CEP pattern detection, and it has the potential to fully enjoy the multi query optimization techniques from traditional database research.

   > The author however, does not fully examine the opportunity of performing multi-query optimization strategy on CEP. Also, there might be some reasons that all CEP paper does not follow their idea of "query plan tree approach on CEP"

2. In CIDR'07 [Cayuga: A General Purpose Event Monitoring System]*Cayuga* intended to apply multi-query optimization on CEP, but the core engine is single threaded, thus the improvement is quite limited. Two major categories of MQO techniques in Cayuga:

   1. state merging.
   2. indexing.
   3. channel.

   > details will be covered in MQO on CEP together in Chapter 4.

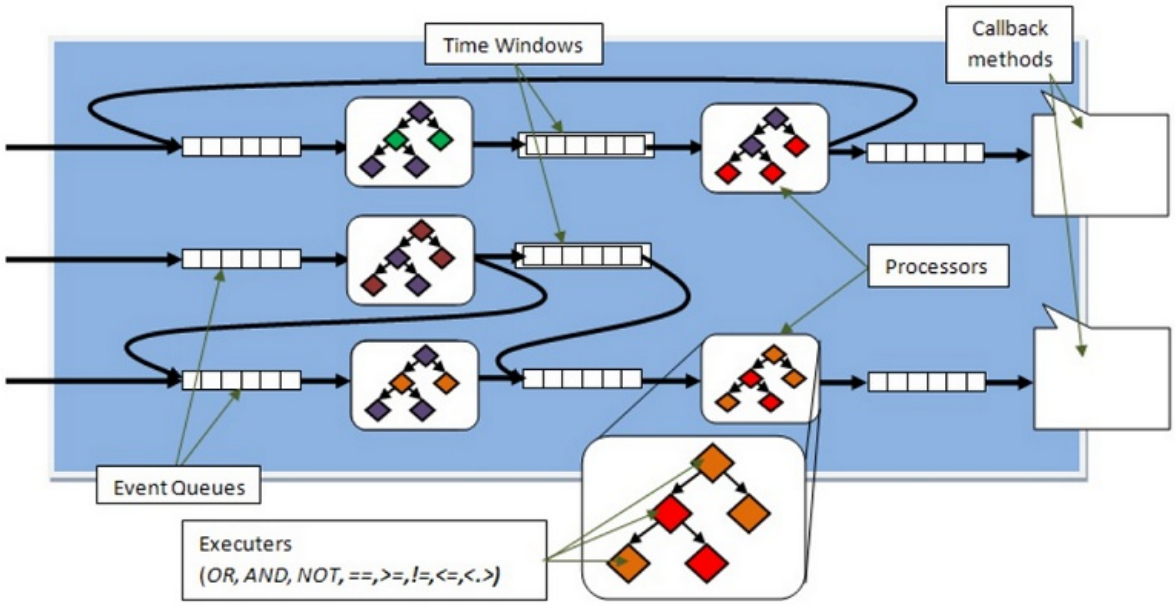
3. In Sigmod'07 [Massively Multi-Query Join Processing in Publish/Subscribe Systems], the author (Same team from Cayuga) proposed a set of novel query processing techniques, referred to as *Massively Multi-Query Join Processing* Techniques, for processing **a large number of XML stream queries** involving value **joins** over Mulitiple XML streams and documents. The proposed techniques are as name suggested primary focus on **joins** (Not other operators).

   > Publish-subscribe system are somehow related to CEP, but not exactly same, the proposed solution may/maynot need to applied into CEP (Only if **Join** is used in CEP).

4. In IPDPS'07 [Optimizing Multiple Distributed Stream Queries Using Hierarchical Network Partitions], this paper consider query optimization in general(traditional) stream processing systems, while, take network partition into consideration, they proposed two methods to efficiently search for optimal plan.
   > Not very related to CEP but may be a useful future work.So I didn't fully expand it in detail here.

5. In GCE'11 [Siddhi: A Second Look at Complex Event Processing Architectures] Siddhi is a Complex Event Processing Implementation that incorporates most of the state of the art advances, done by a group of undergraduate students in their final year project.

   > Although no much novelty there, they emphasized the importance of **multi-thread** demanded in CEP:



6. In SoCC'10, [Comet:Batched Stream Processing for Data Intensive Distributed Computing], the author introduced *Batched Stream Processing (BSP)* to model recurring(batch) computations on incrementally bulk-appended data streams. The core ideas for optimization are coming from varies query processing techniques in database systems, including sharing intermediate results among queries.

4. MQO on CEP

NOTED: The summerize of MQO on CEP is categorized by research group instead of ideas(or papers). Because the efforts in this topic is very limited, that's why we need to do something.

4.1 Cayuga(Hong Ming Sheng, Cornell, project stopped)

4.1.1 Related papers:

1. In EDBT'06, [Towards Expressive Publish/Subscribe Systems], the author extended publish-subscribe system inorder to support stateful operators(like those heavily used in CEP).

   > A very impactful thing is they emphsized the strenth of both MQO(from publish-subscribe system) and stateful operators(from CEP). Also, they have introduced the idea of extending original MQO techniques to support CEP. A very interesting experiment result shows that <State-merge> is not that useful compare to *efficient indexing*.

2. In EDBT'09, [Rule-based Multi-query optimization], the author proposed a **unified** framework to support multi-query optimization for both stream processing engine(SE) and event detection engine(EE).

   > For EE, the main idea is to translate the original automata into query plan tree with some additional operators then re-use MQO techniques from publish subscribe system, they cannot be used in real CEP system (As E-Cube does).

4.1.2 Core ideas:

4.1.2.1 State merging(common)

1. The first type is **prefix state merging**. Intuitively, given an existing automaton a, and a new input automaton b, b can be merged into a by *identifying the longest prefixes of a and b that are identical, and then share the two prefixes in the merged automaton.

   > In other words, they translate the *MQO problem on automata* into the well known MQO technique on query plans:**Common Subexpression Elimination(CSE)**.

2. In addition(mentioned by author), **resubscription** in CEP that must need **two** NFA, that make them not able to inlineing, are now possible to inline as they can be expressed by **one** query plan tree. In other words, the author prefer query plan tree, instead of NFA.

4.1.2.2 Automaton indexing(First *applied* in Cayuga)

> Indexing techniques are studied in publish-subscribe system for many years, it's still a very hot topic in publish-subscribe system area, VLDB'14 got one paper in this topic by the way. But Cayuga is the first who considering indexing in CEP area, but the contribution is very limited and not completed:

```
1. They only consider optimization for queries with same sequence pattern, or say, they ignored the fact that different CEP qu
2. Their indexing techniques can only work for "ESP Style" filters. CEP Sylte filters like "parameterized filter"(Introduced
```

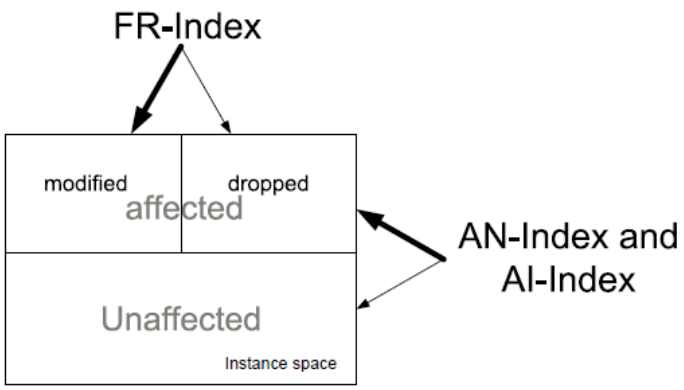


Fig. 3. Instance Search Space

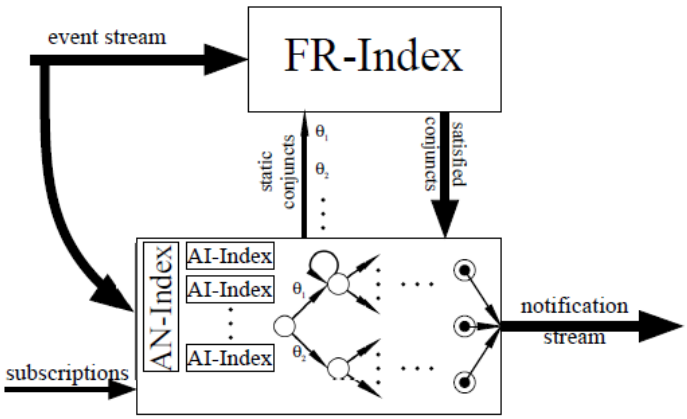


Fig. 4. Cayuga architecture

1. Active Node Index & Active Instance Index
    1. ***motivation***: the number of active automaton instances and number of edges that each instance could potentially traverse can be very large. Hence, evaluating all these edge predicates for each incoming event is not feasible.
    2. ***goal*** : to efficiently identify the instances (query instances/active automaton instance)that are affected by an incoming event.

        > ***NOTICE***: an instance is affected iff it cannot filter the coming event (equivalently to say is the filter predicate is satisfied).

    3. **(filter predicate (Key), automaton instances (value))** structure.
2. Forward-Rebind(FR) index
    1. ***motivation*** : Knowing the instances affected by an incoming event is not sufficient.
    2. ***goal*** : need to determine, **which forward and rebind edges** these instances will traverse.
        1. Traverse a FR edge modifies an instance bindings (e.g. the instance are looking for another event), affecting the instance content.
        2. If no edge can be traversed, the instance is deleted.

> ***NOTICE:*** Unfortunately, I don't see any particular strong reason to have this index. And in the original paper, there's no workload that specially need FR index, a even more important question would be: why do we need a predicate on the edge, instead of just on the node itself.

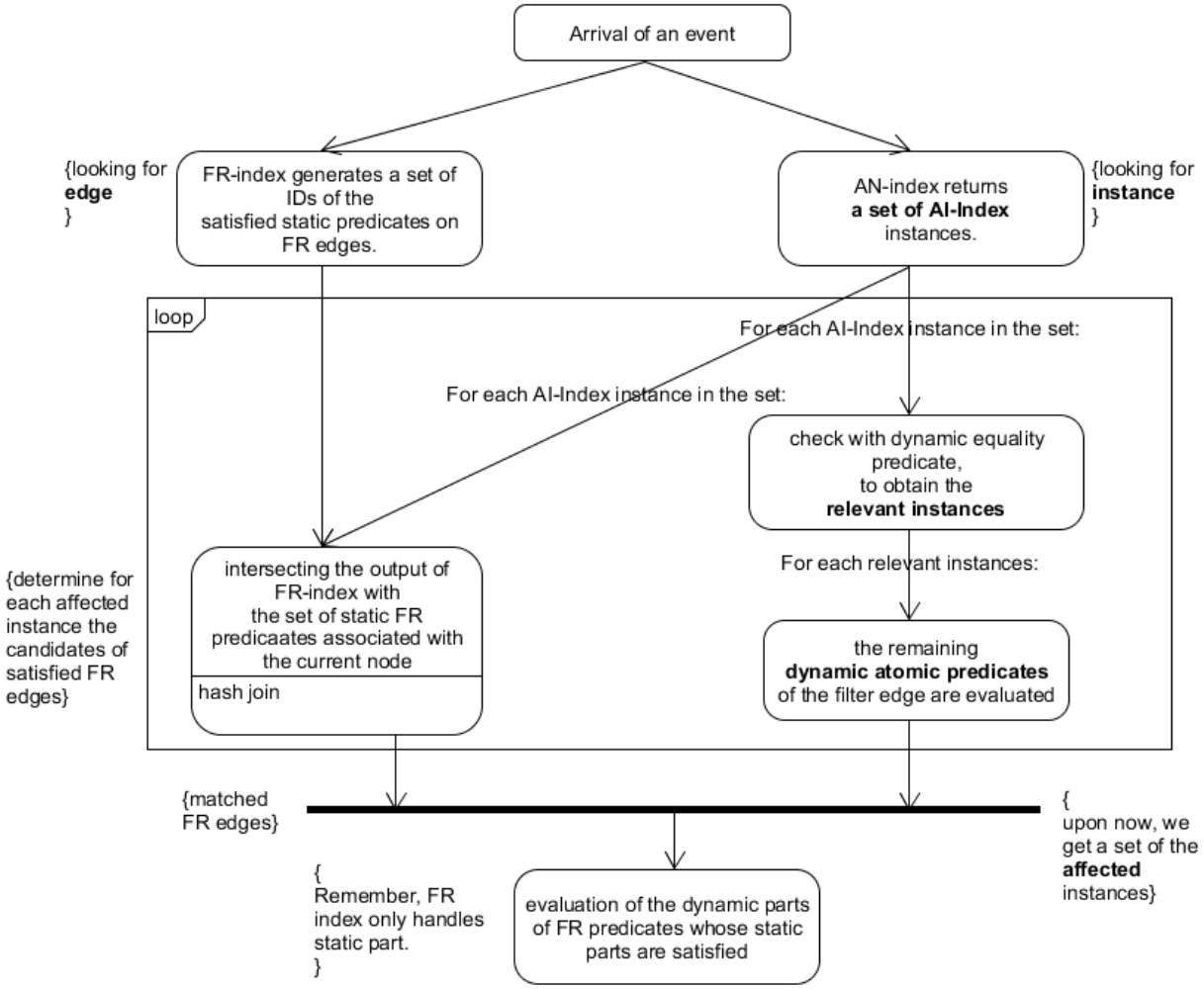
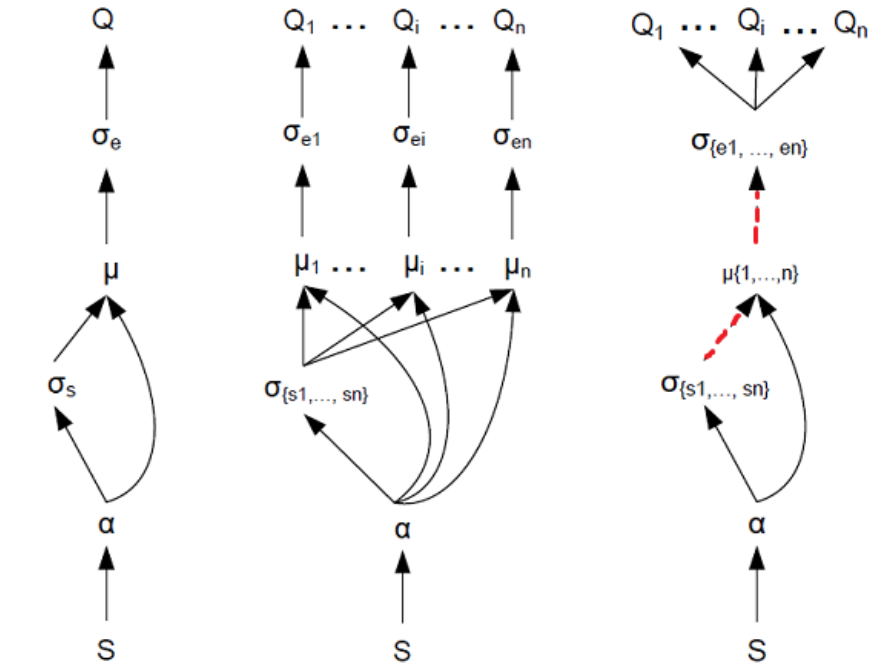


Figure:activity of cayuga indexing upon arrival of an event.

4.1.2.3 Channel on CEP(first *proposed* by Cayuga):

In the above techniques, operators that have *same definition* cannot share because the input stream are different, in order to cope with this constraint, the author proposed a technique called **Channel** (I call it **label** previously).



(a) Query Plan for Query 1 (b) Query Plan for n Instances of Query 2 Without Channel (c) Query Plan for n Instances of Query 2 With Channel

Some good comments from the paper:

Channels are a powerful mechanism that allows us to aggressively share work among operators that read even **different** streams... Trade-offs that we need to bear in mind:

> 1. if two streams are encoded into the same channel, then stream tuples with the **same content** can share storage by being represented as the came channel tuples.
> 2. if the consumer operators of them have the same definition the evaluation on channel tuples will be more efficient than evaluating tuples from them separately.
> 3. mapping multiple streams to the same channel creates overhead:
>   1. Time-wise, with multiple streams being mapped to the same channel, the consumer of this channel now has to process the membership component of each input tuple(label).
>   2. Space-wise, each channel tuple has to carry the membership component.

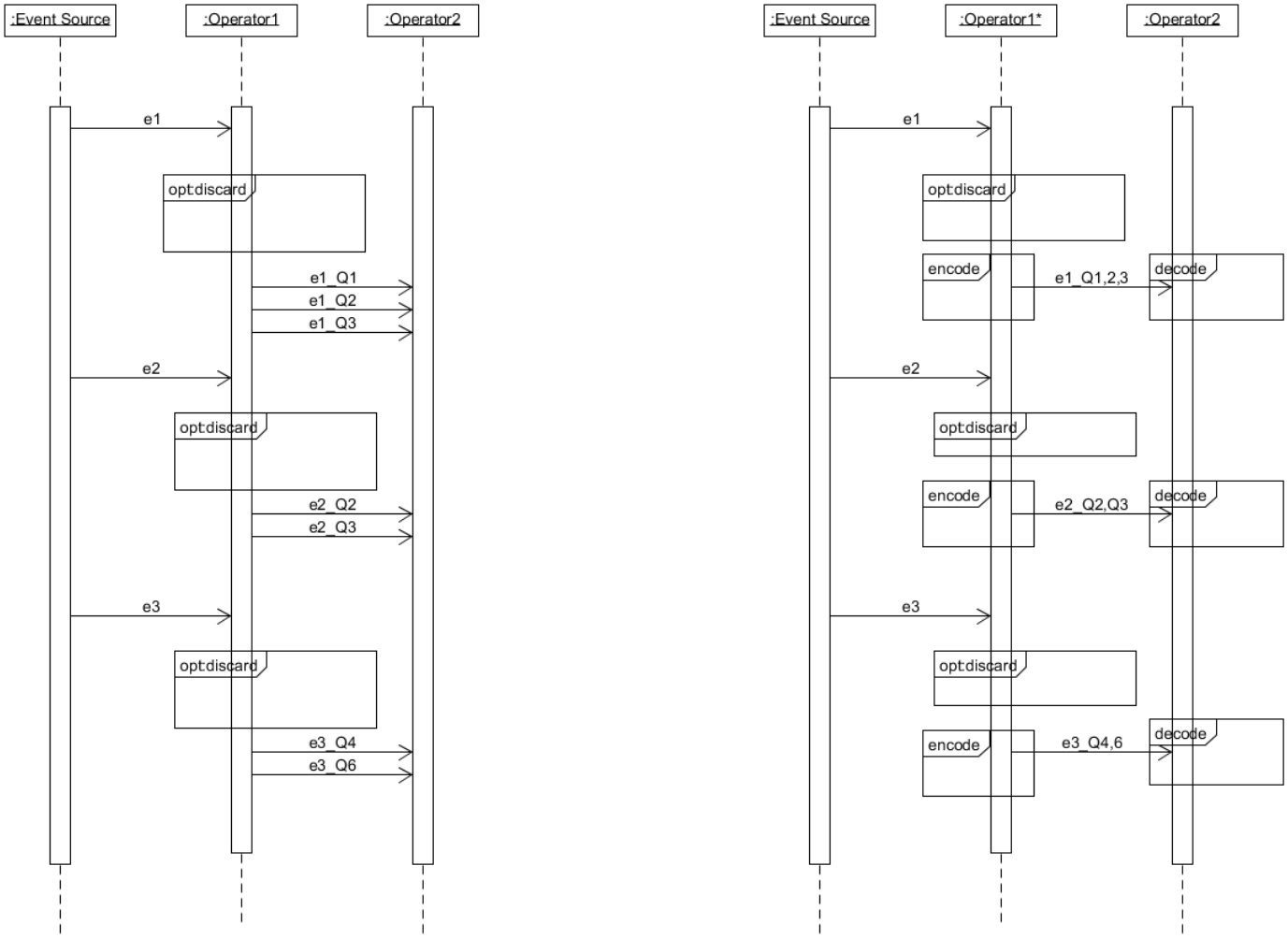
Thus, two stream S1,S2, are *sharable*, denoted as S1~S2.

```
1. S1=S2 => S1~S2.
2. S1 and S2 are produced by two stream sources that are labeled to be sharable then S1~S2.
3. For any *unary operators o1, o2, if S1 :=o1(T1), S2 :=o2(T2), and T1~T2, then S1~S2.
4. For any binary operators o1, o2, if S1 :=o1(T1,U1),S2:=o2(T2,U2), and o1=o2, T1~T2, U1~U2, then S1~S2.
5. For selection operator that read T and produces S, S~T.
6. S1~S2 => S2~S1.
7. S1~S2 ^ S2~S3 => S1~S3.
```

But, only put them into same channel IF following conditions are all true:

```
1. S1~S2
2. S1 and S2 are produced by the same **m-op**
3. the consumers of the s1 and s2 have the same definition.
```

> as pointed out by the author, those constraint only make the channel more effective, and applicable in many practical use-case.



without channel(left), with channel(Right)

4.2 HP-Chaos(Elke A. Rundensteiner, WPI, Research still very active)

4.2.1 Related papers:

1. In VLDB'09 [A Shared Execution Strategy for Multiple Pattern Mining Requests over Streaming Data]
2. In sigmod'11 [E-Cube: Multi-Dimensional Event Sequence Analysis Using Hierarchical Pattern Query Sharing], the author proposed hierarchical topology execution strategy on CEP. which is state-of-the-art multi-query sharing strategy.
3. In EDBT'11 [An Optimal Strategy for Monitoring Top-k Queries in Streaming Windows]
4. In SIGMOD'14[Complex Event Analytics: Online Aggregation of Stream Sequence Patterns]

4.3 SASE(Diao yan lei, UMASS, Research still very active)

Unfortunately, SASE seems not interested in MQO topic, thus their paper never consider MQO yet.