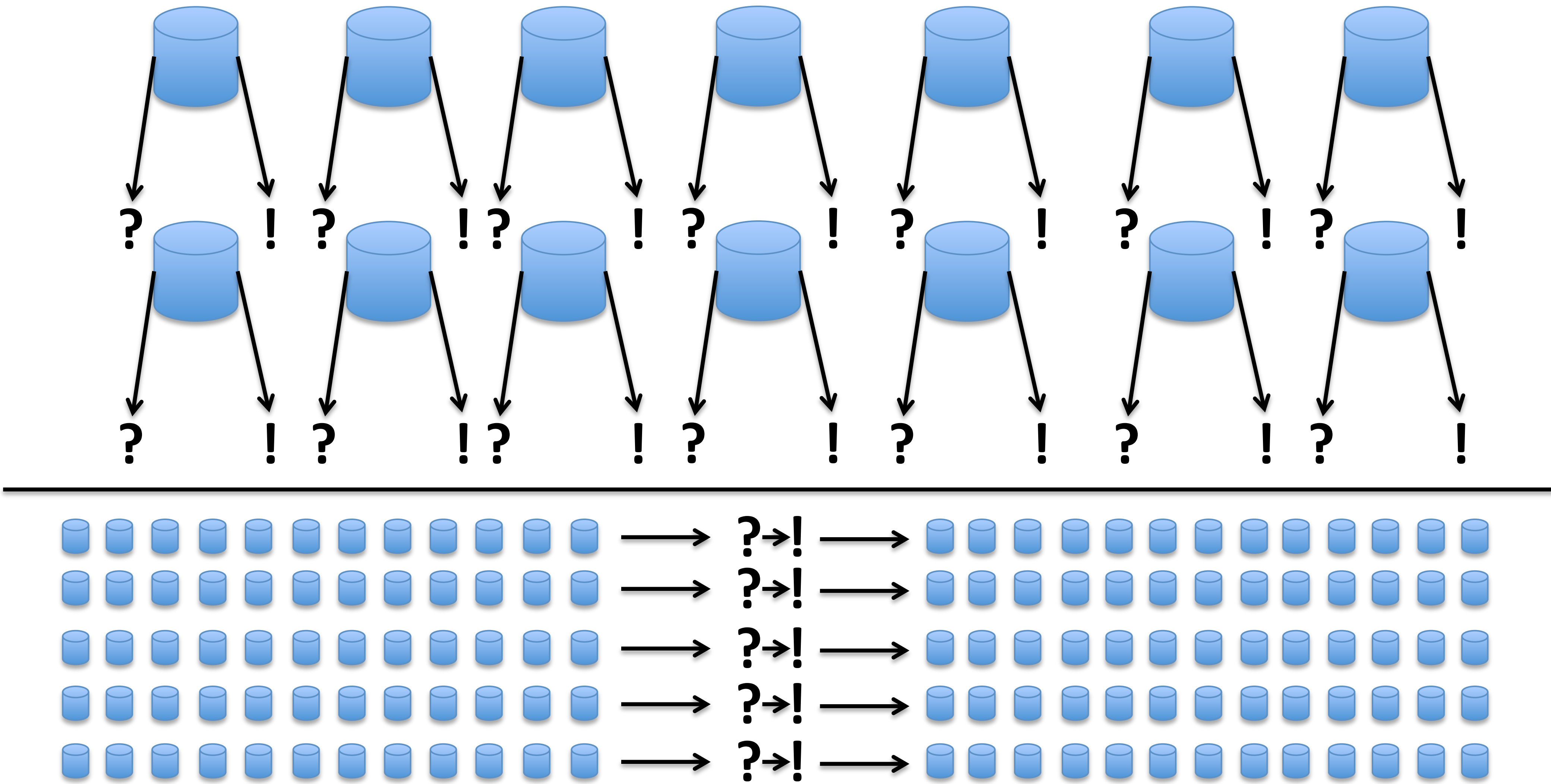
Auto-Parallelizing Stateful Distributed Streaming Applications

Scott Schneider[*], Martin Hirzel[*], Bugra Gedik[+] and Kun-Lung Wu[*]

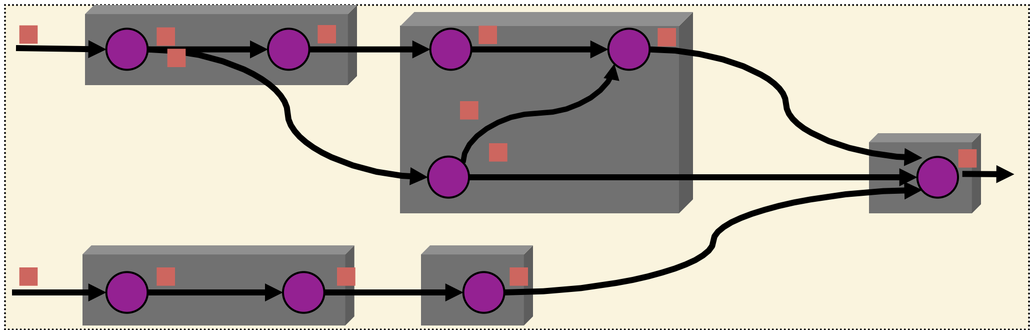
[*]*IBM Research* [+]*Bilkent University*

Big Data



Programming Model

- Streams applications
  - Described as data-flow graphs
    - An instance of a flow graph is a job in the system
  - Flow graphs consist of
    - **Tuples**: structured data item
    - **Operators**: Reusable stream analytics
    - **Streams**: Series of tuples with a given schema



Streaming Programming Models

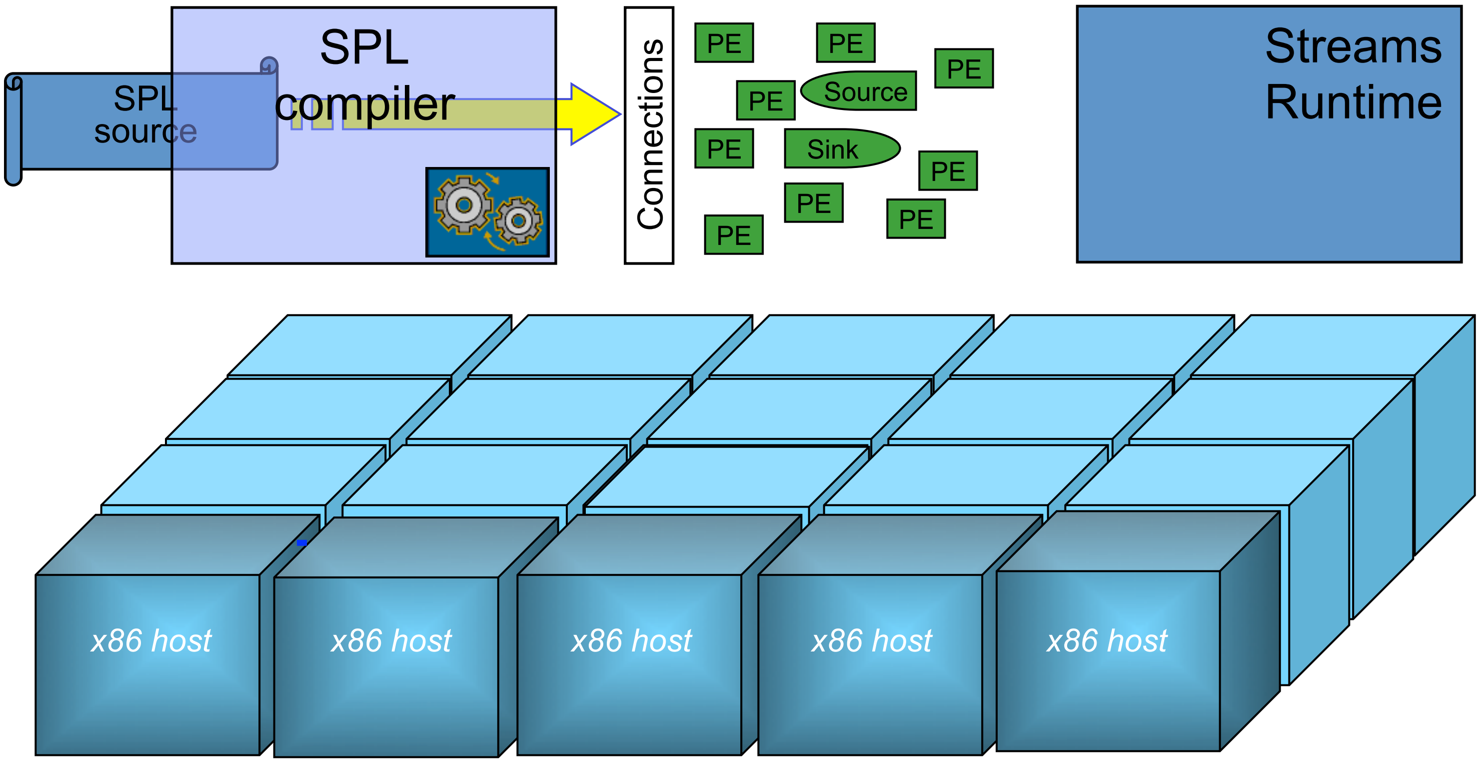
Synchronous

- Static selectivity
  - e.g., 1 : 3
    ```
    for i in range(3):
      result = f(i)
      submit(result)
    ```
  - In general, *m* : *n* where *m* and *n* are statically known
- Always has static schedule

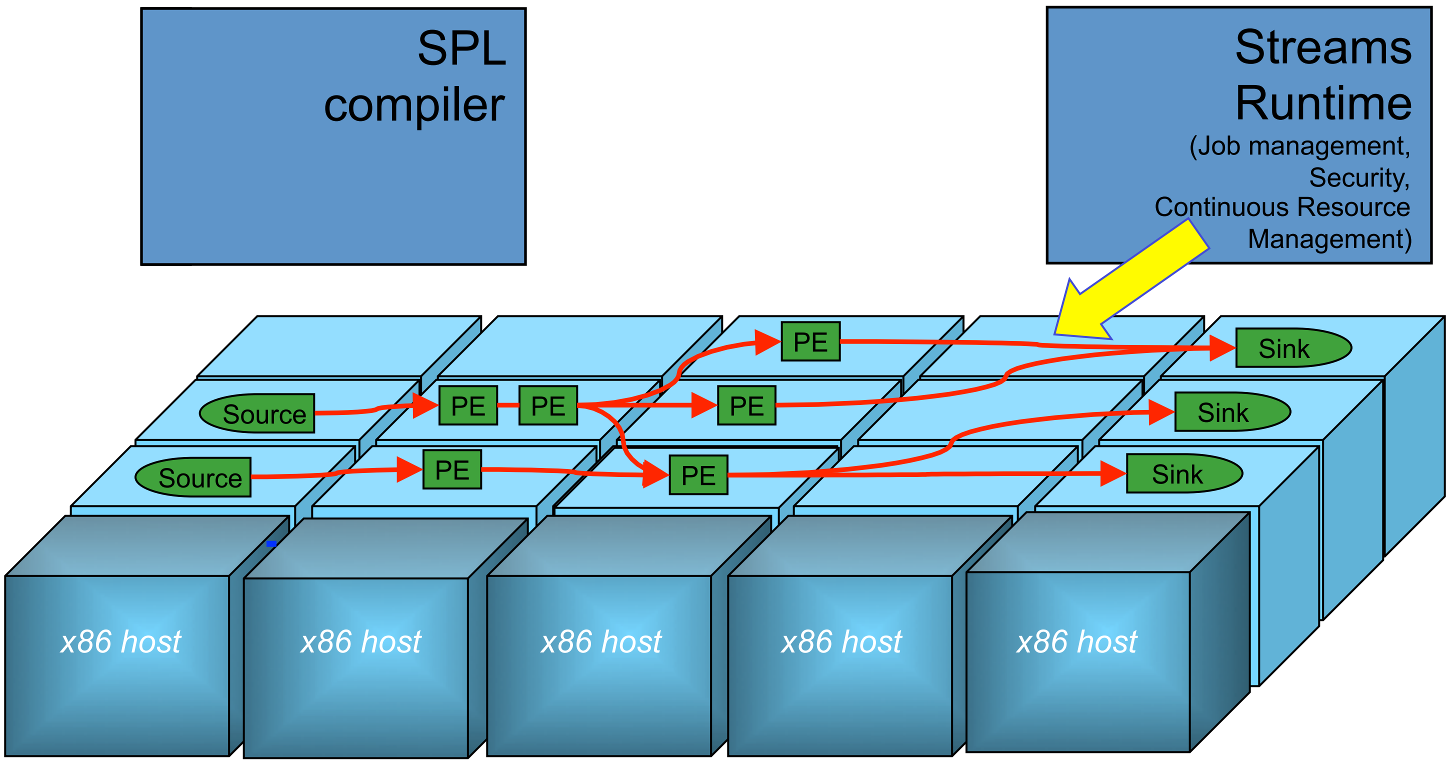
Asynchronous

- Dynamic selectivity
  - e.g., 1 : [0,1]
    ```
    if input.value > 5:
      submit(result)
    ```
  - In general, 1 : *
- In general, schedules cannot be static

InfoSphere Streams Runtime



InfoSphere Streams Runtime



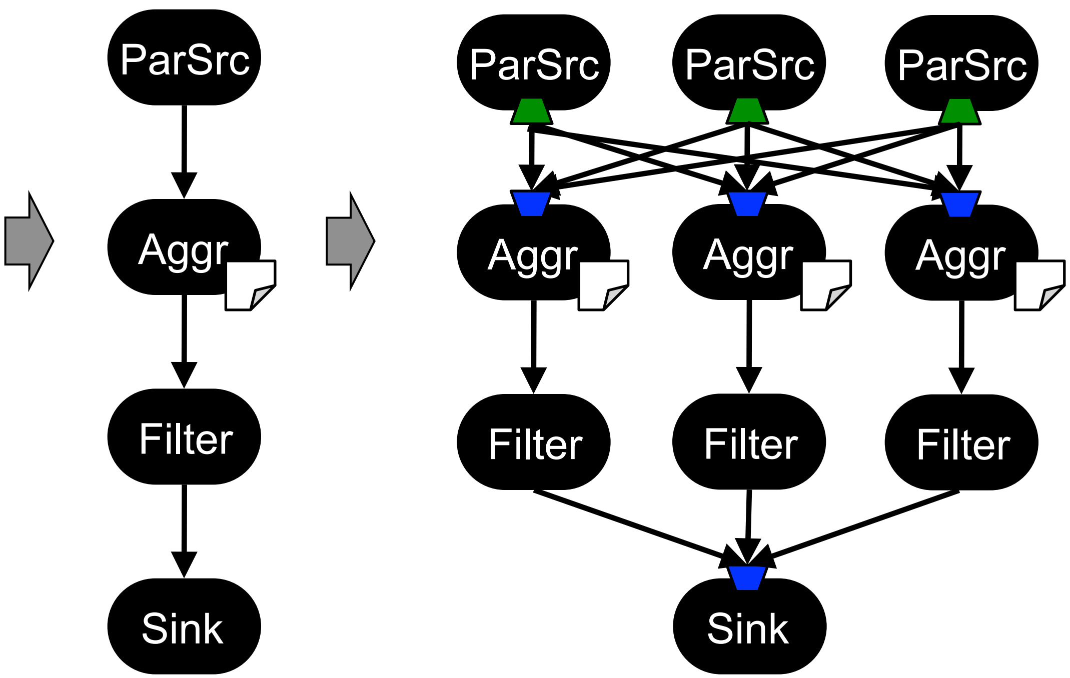
What we do

```
composite Main {
type
  Entry = int32 uid, rstring server,
          rstring msg;
  Sum = uint32 uid, int32 total;
graph
  stream<Entry> Msgs = ParSource() {
    param servers: "logs.*.com";
          partitionBy: server;
  }

  stream<Sum> Sums = Aggregate(Msgs) {
    window Msgs: tumbling, time(5),
                 partitioned;
    param partitionBy: uid;
  }

  stream<Sum> Suspects = Filter(Sums) {
    param filter: total > 100;
  }

  () as Sink = FileSink(Suspects) {
    param file: "suspects.csv";
  }
}
```



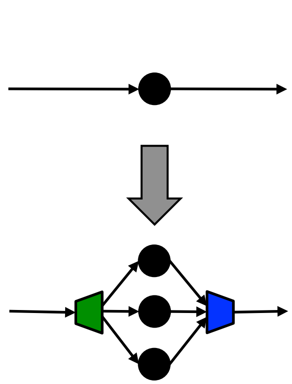
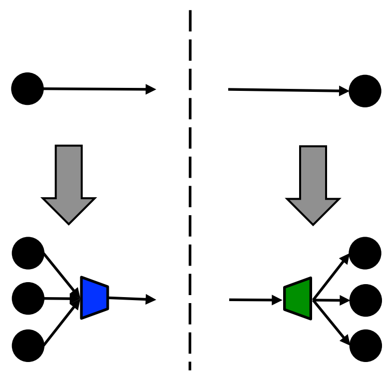
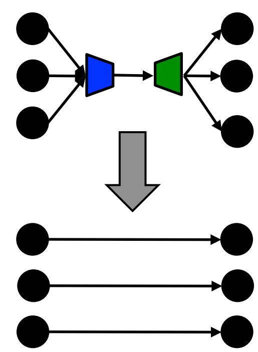
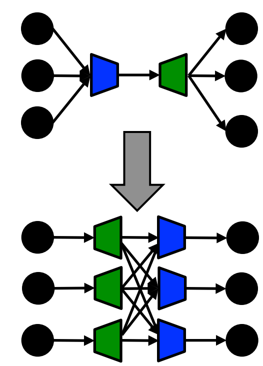
Overview

Compiler:
- Apply parallel transformations
- Pick routing mechanism (e.g., hash by key)
- Pick ordering mechanism (e.g., seq. numbers)

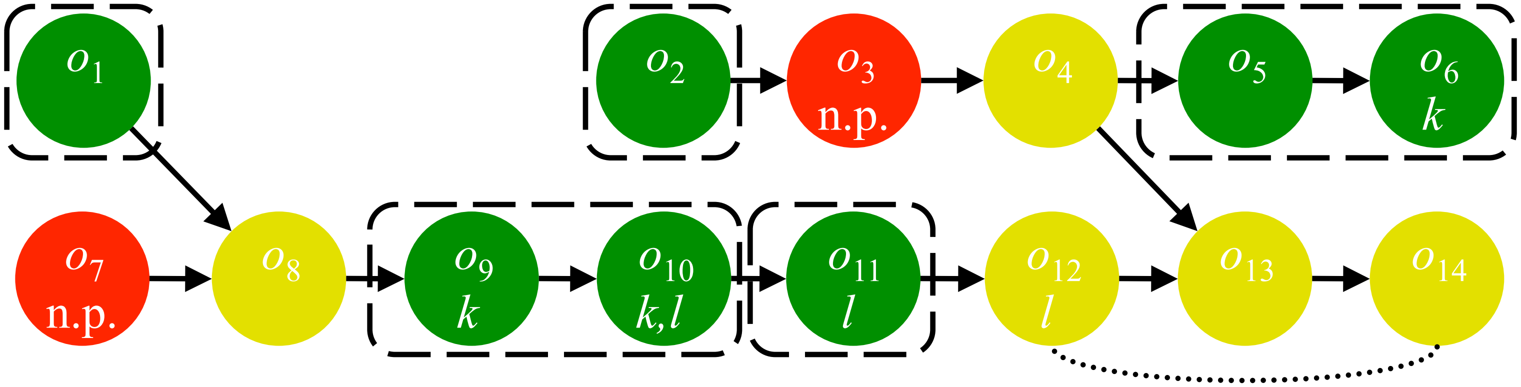
↓ Stream graph description

Runtime:
- Replicate segment into channels
- Add split/merge/shuffle as needed
- Enforce ordering

Transformations & Safety Conditions

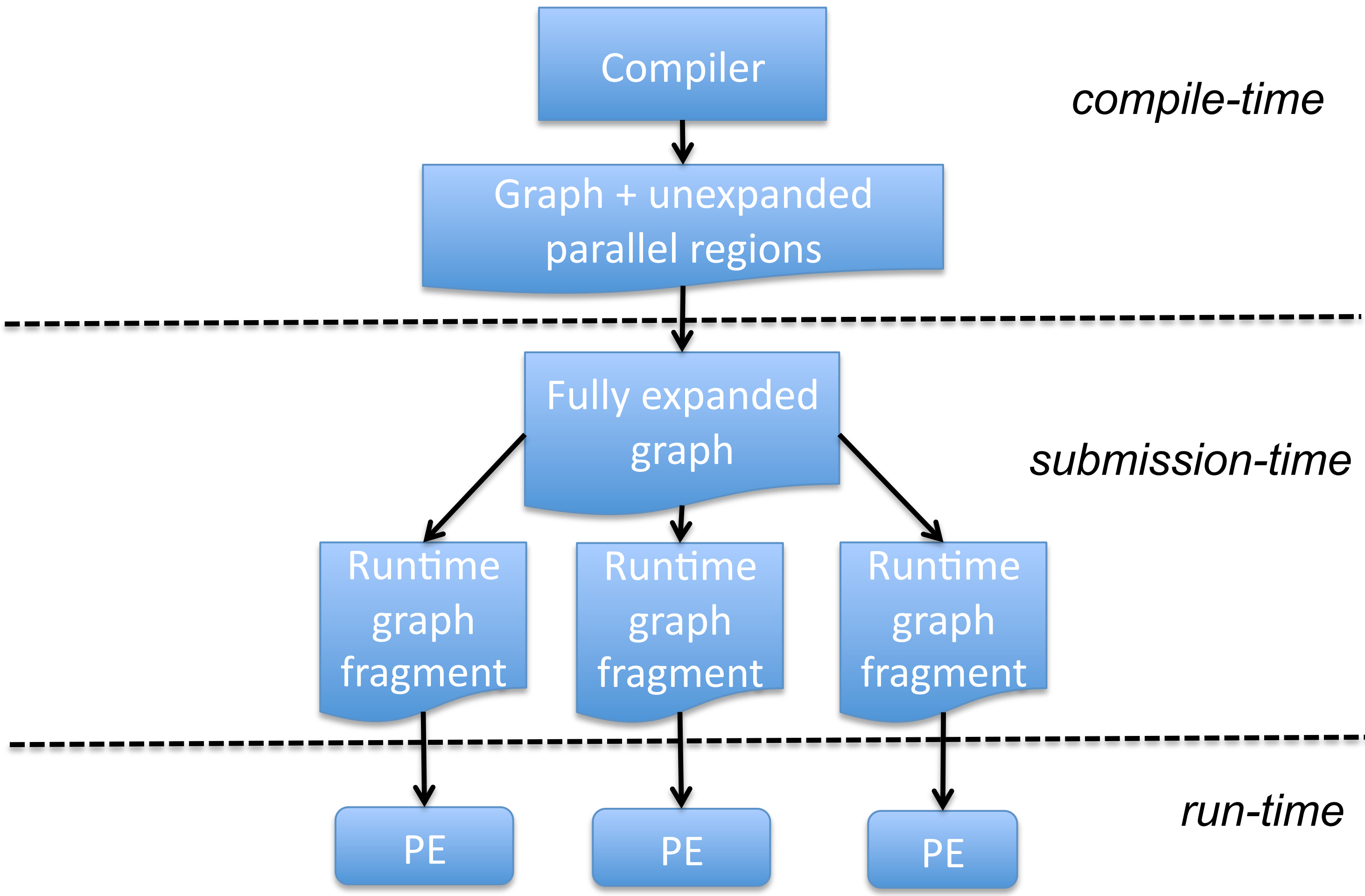
Parallelize non-source/sink	Parallelize sources and sinks	Combine parallel regions	Rotate merge and split
			
• stateless *or* partitioned state • selectivity ≤ 1 • simple chain	• stateless *or* partitioned state	• stateless *or* • compatible keys • forwarding	• incompatible keys

Select Parallel Segments



- Can't parallelize
  - Operators with >1 fan-in or fan-out
  - Punctuation dependecy later on
- Can't add operator to parallel segment if
  - Another operator in segment has co-location constraint
  - Keys don't match

Compiler *to* Runtime

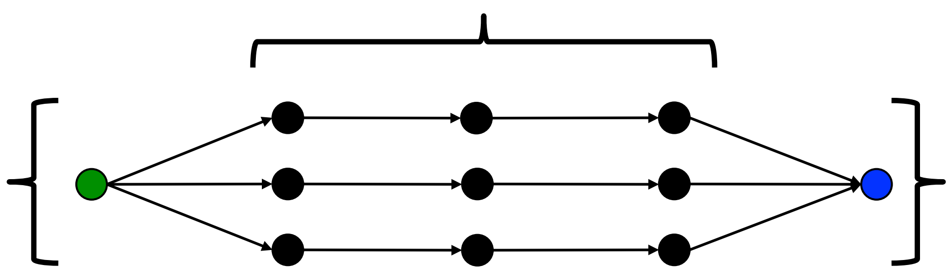


Runtime

	selectivity = 1	selectivity ≤ 1	selectivity unknown
no state			*don't parallelize*
partitioned state			*don't parallelize*
unknown state	*don't parallelize*	*don't parallelize*	*don't parallelize*

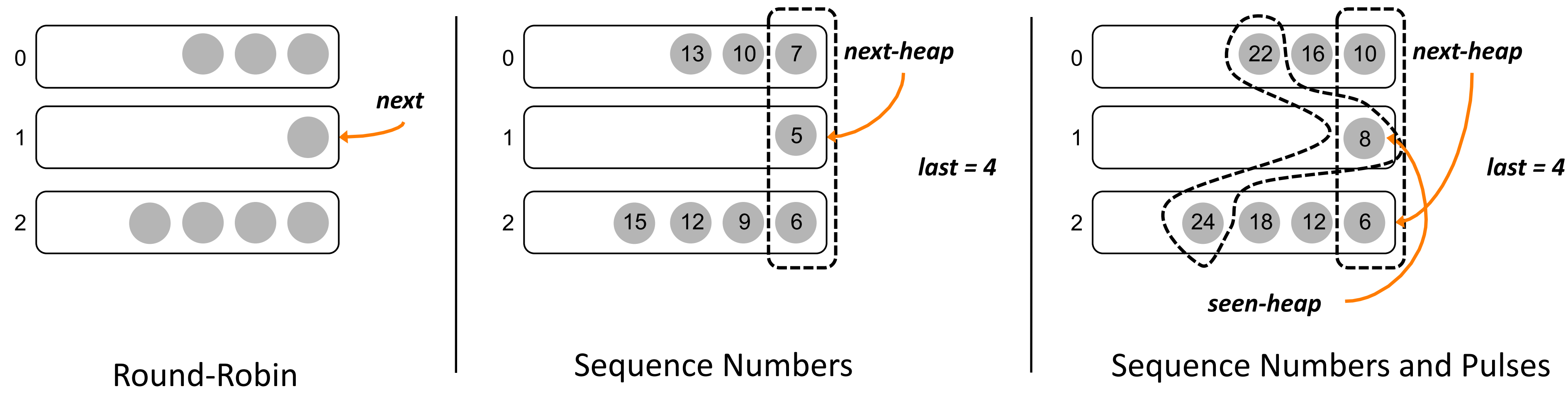
Operators in parallel segments:
- Forward seqno & pulse

Split:
- Insert seqno & pulse
- Routing

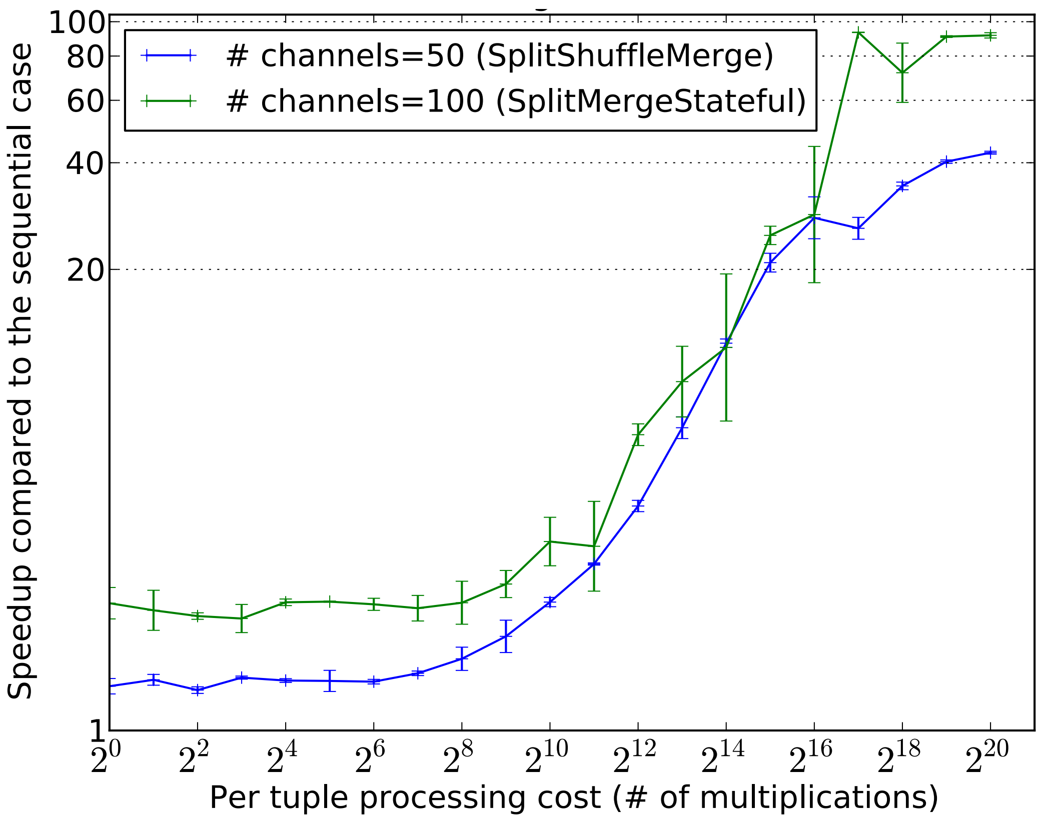


Merge:
- Apply ordering policy
- Remove seqno (if there) and drop pulse (if there)

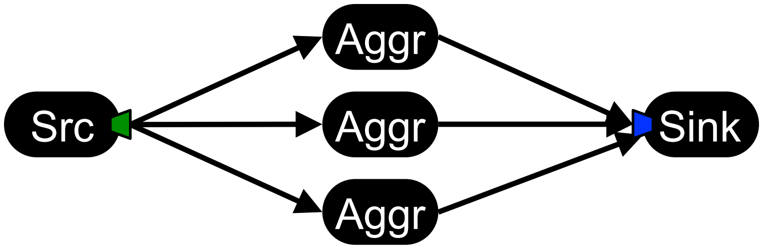
Merger Ordering

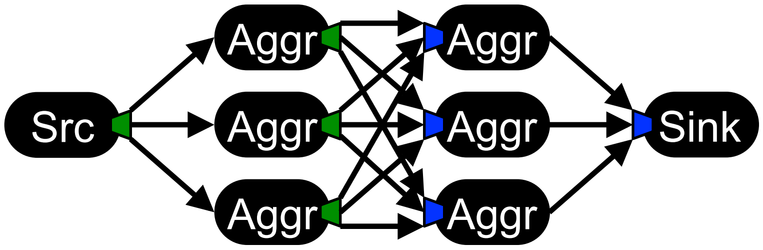


Scalability



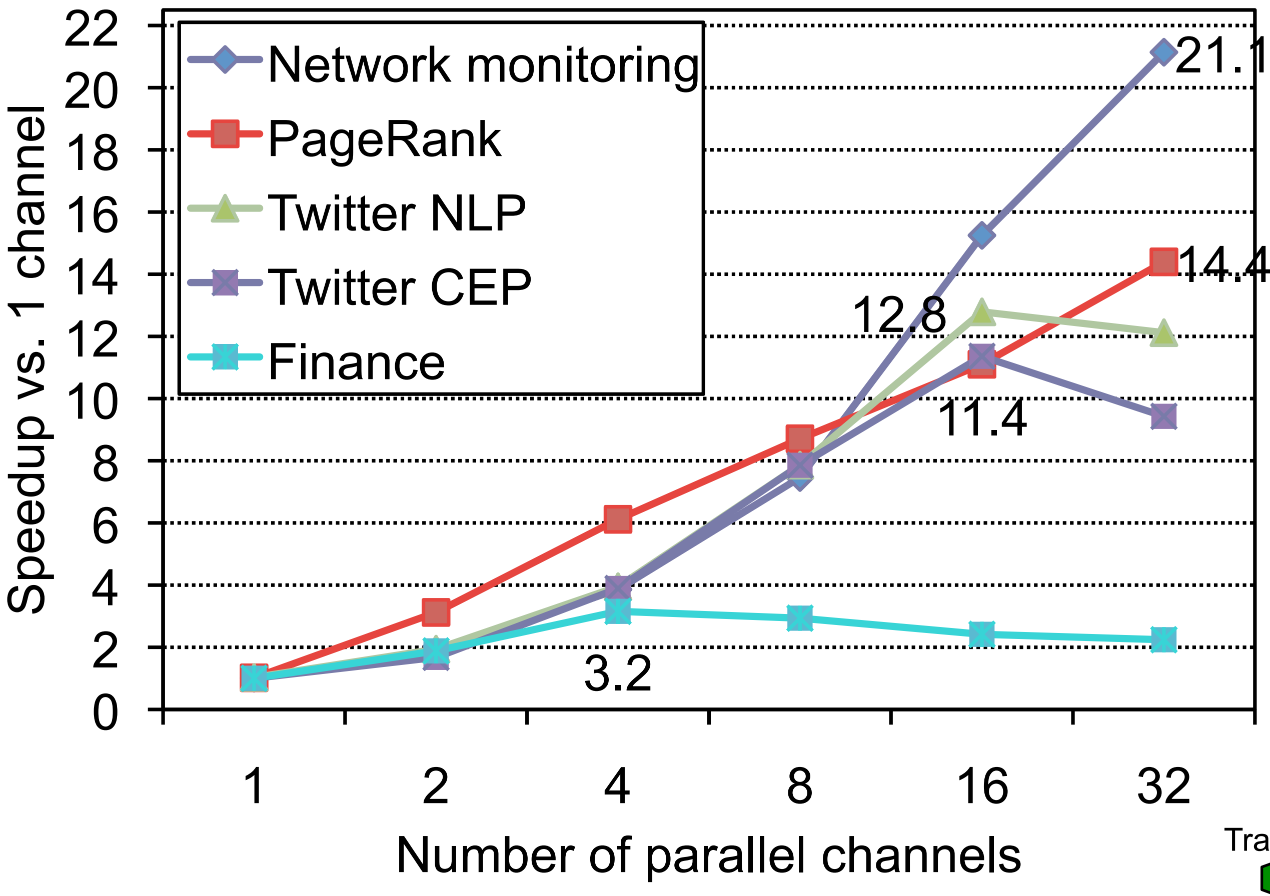
SplitMergeStateful

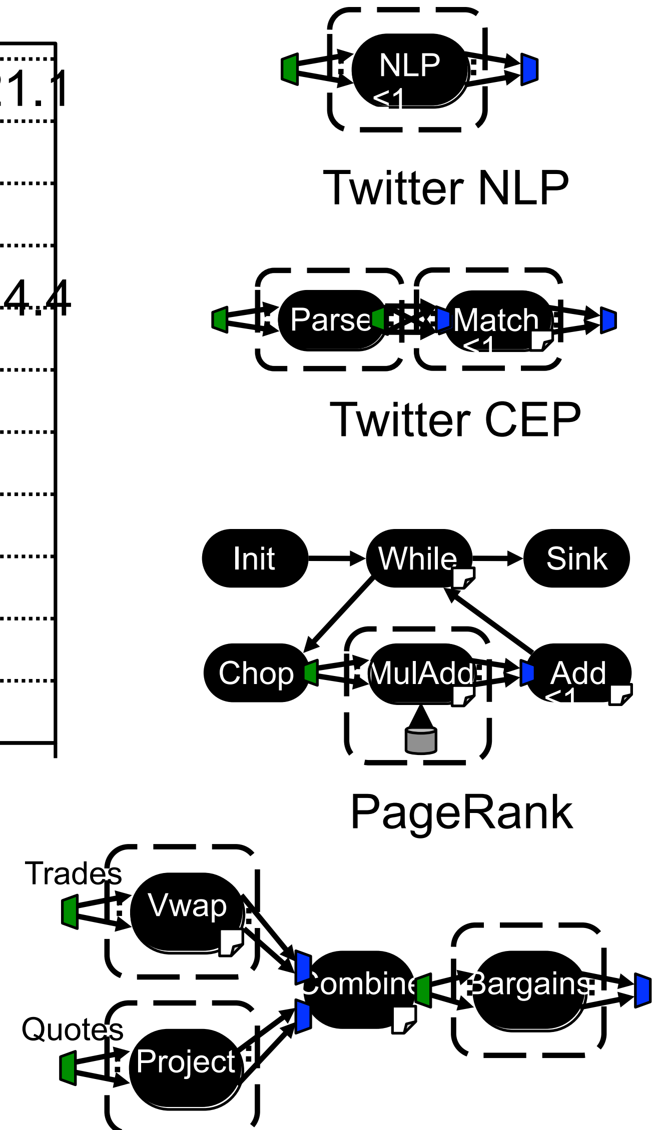


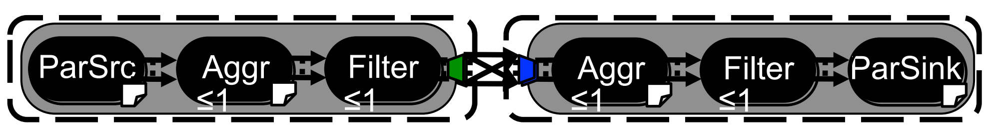


SplitShuffleMerge

Application Kernels



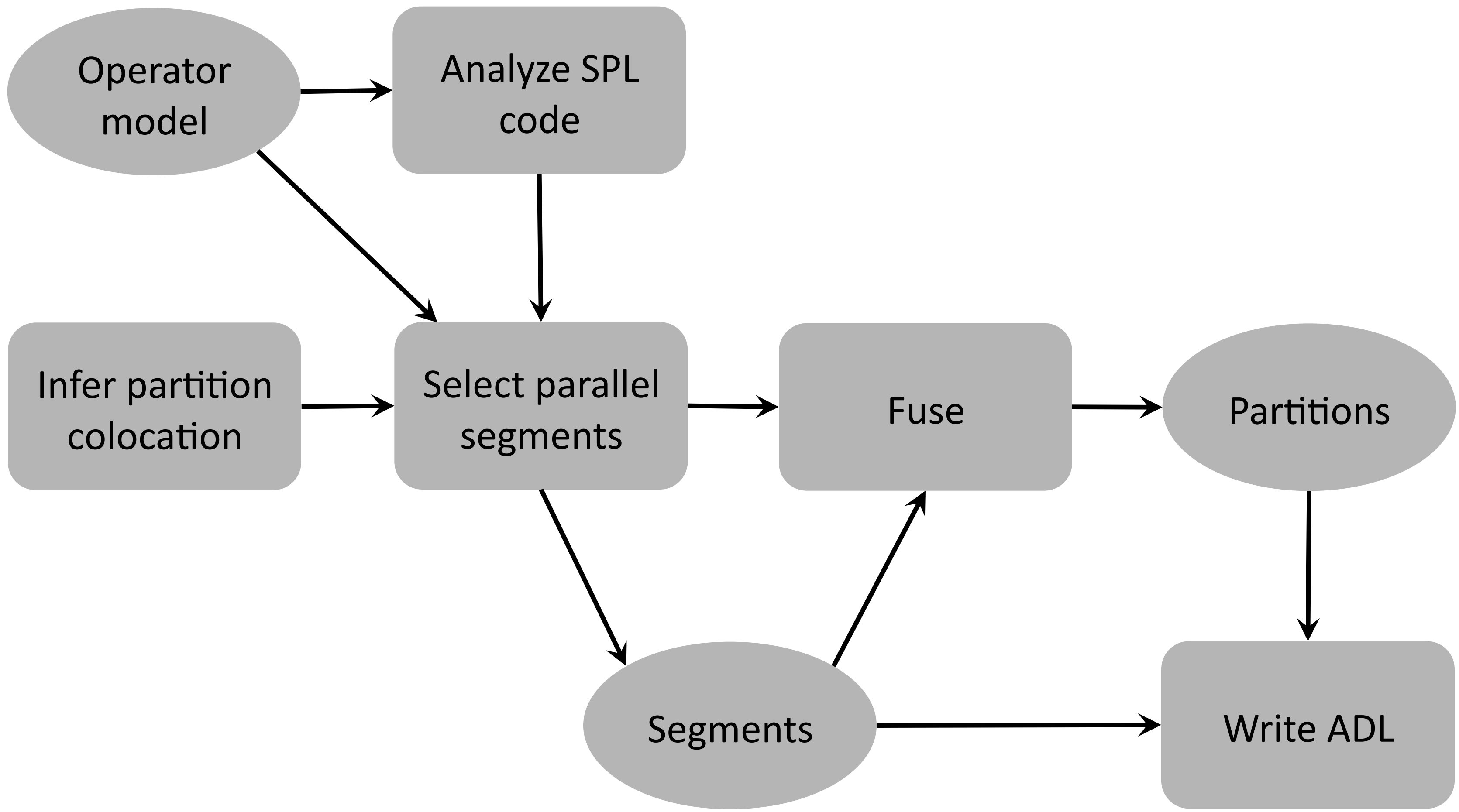




Questions?

Backups

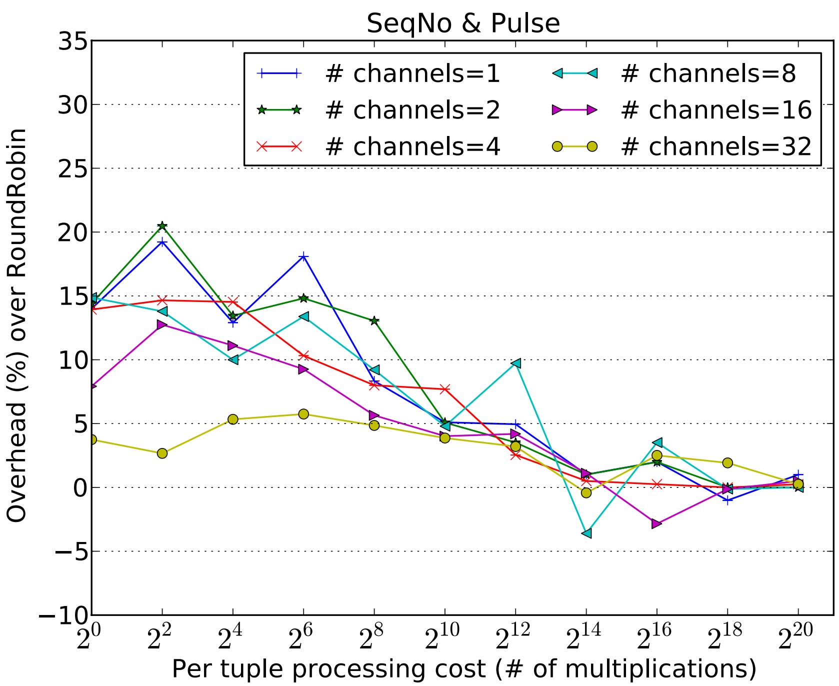
Compiler Changes



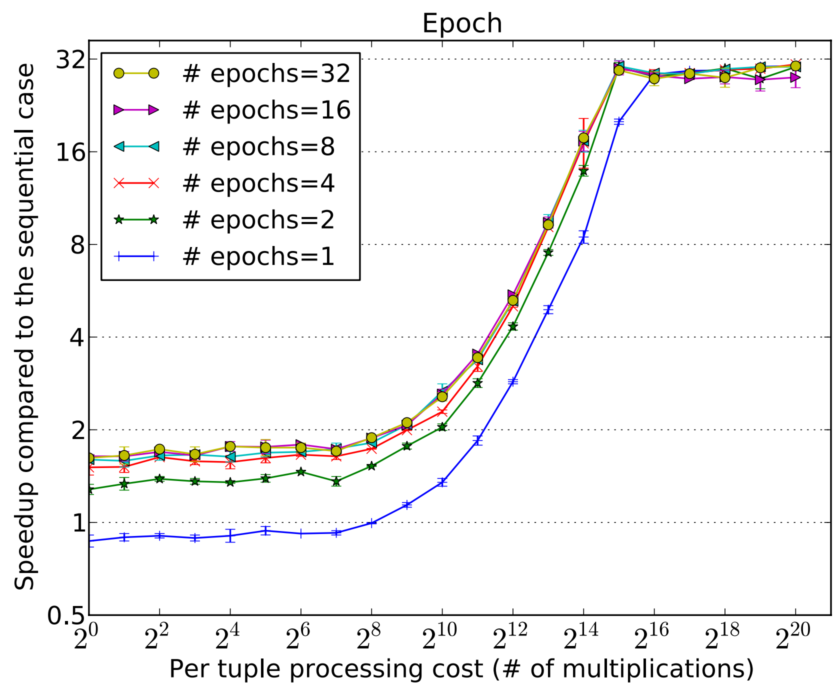
Transport Changes

- TCPSender
  - Added the ability to send to a subset of connections on an output port
- Handshake
  - Modified to include sender identities
- TCPReciever
  - Added support for identifying which connection has delivered a tuple

Overhead

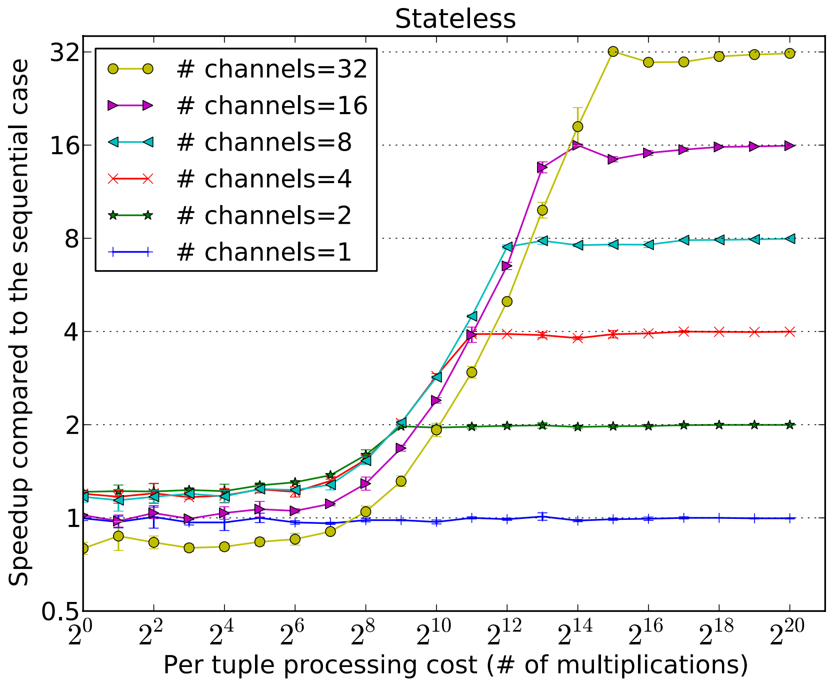


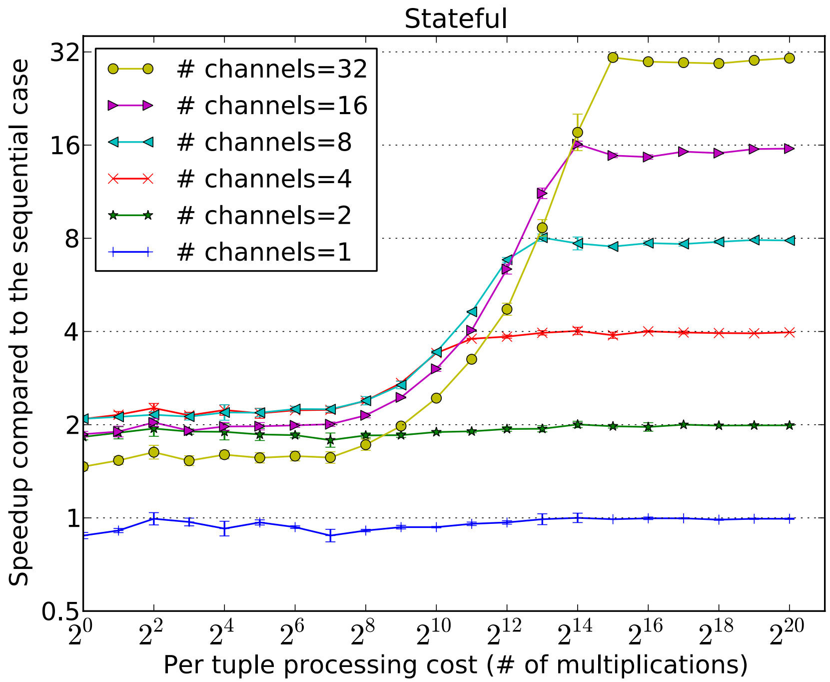
SeqNo & Pulse
Overhead (%) over RoundRobin
Per tuple processing cost (# of multiplications)
channels=1
channels=2
channels=4
channels=8
channels=16
channels=32
35
30
25
20
15
10
5
0
−5
−10
2^0 2^2 2^4 2^6 2^8 2^{10} 2^{12} 2^{14} 2^{16} 2^{18} 2^{20}



Epoch
Speedup compared to the sequential case
Per tuple processing cost (# of multiplications)
epochs=32
epochs=16
epochs=8
epochs=4
epochs=2
epochs=1
32
16
8
4
2
1
0.5
2^0 2^2 2^4 2^6 2^8 2^{10} 2^{12} 2^{14} 2^{16} 2^{18} 2^{20}

Scalability





Scalability