Low-Synchronization, Mostly Lock-Free, Elastic Scheduling for Streaming Runtimes

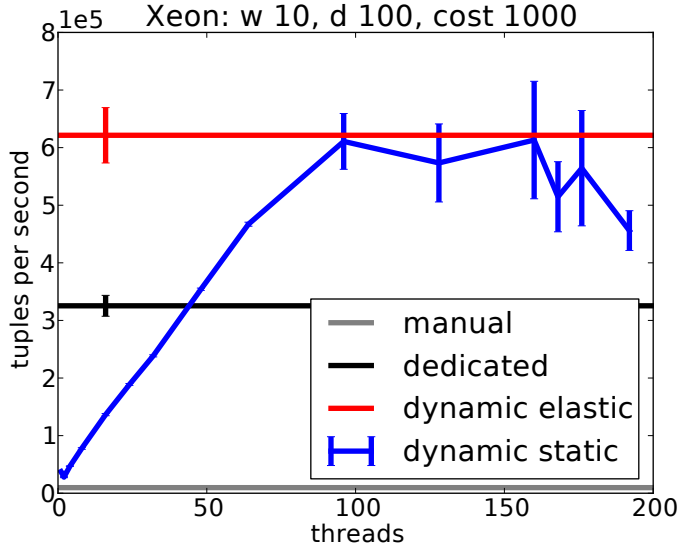
Scott Schneider and Kun-Lung Wu

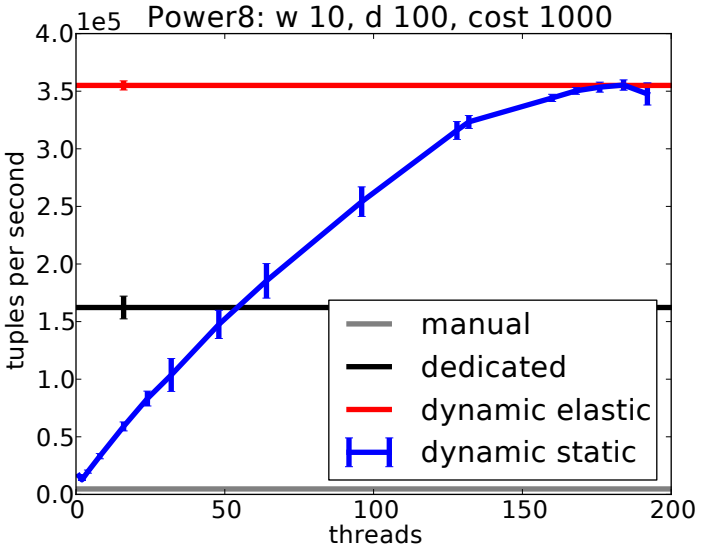
IBM Research

{scott.a.s, klwu}@us.ibm.com

Conclusions First

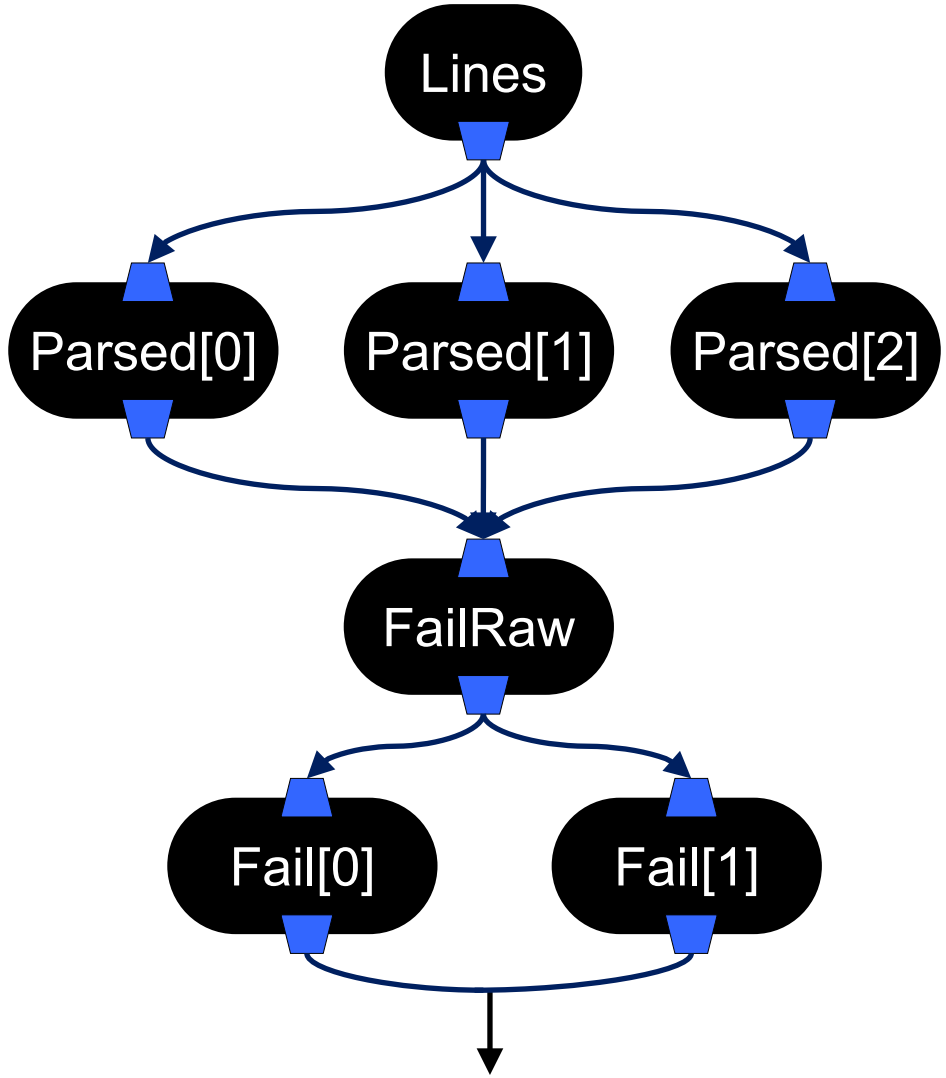
- Streaming scheduler that scales to thousands of operators and hundreds of threads
  - minimizes locks, synchronization and global data access
  - interface must maintain backwards compatibility with previous versions
- Elasticity algorithms seek out “good enough” thread levels
  - establishes performance improvement trends; trust (or not) of prior data
  - seeks highest thread level with no further improvement trend that it trusts



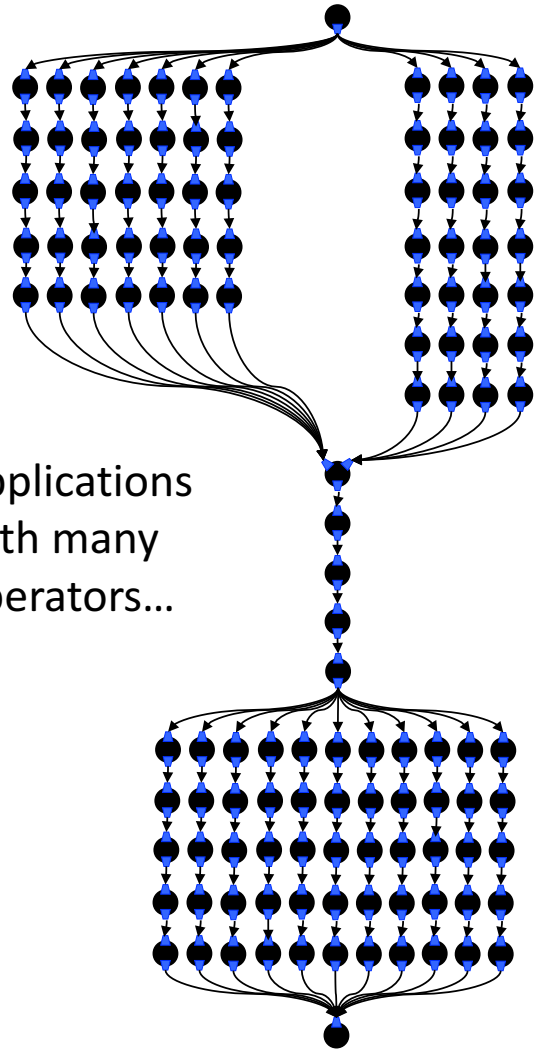


SPL: Tuples, Streams and Operators

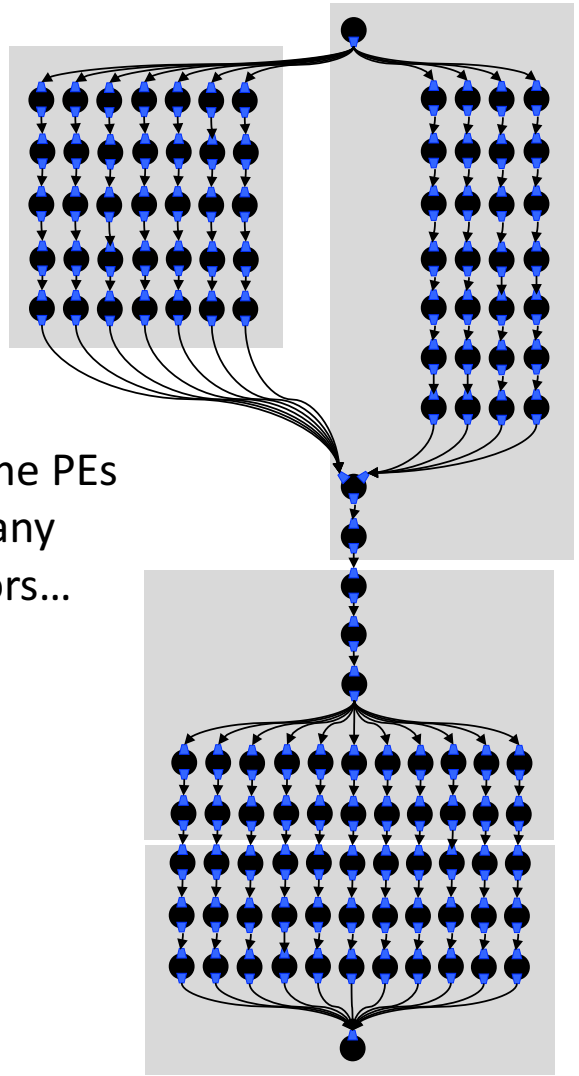
```
composite LoginFailures(output Fail) {
type LogLine = timestamp time, rstring hostname,
               rstring srvc, rstring msg;
     Failure = timestamp time, rstring uid, rstring euid,
               rstring tty, rstring rhost,
               rstring user;
graph
  stream<rstring line> Lines = FileSource() {
    param format: line;
          file: "/var/log/messages";
  }
  @parallel(width=3)
  stream<LogLine> Parsed = Custom(Lines) {
    logic onTuple Lines: submit(parseLog(line), ParsedLines);
  }
  stream<LogLine> FailRaw = Filter(ParsedLines) {
    param filter: match(srvc, "sshd") &&
                  match(msg, "authentication failure");
  }
  @parallel(width=2)
  stream<Failure> Fail = Custom(FailuresRaw) {
    logic onTuple FailRaw: submit(parseMsg(msg), Failures);
  }
}
```



The Problem

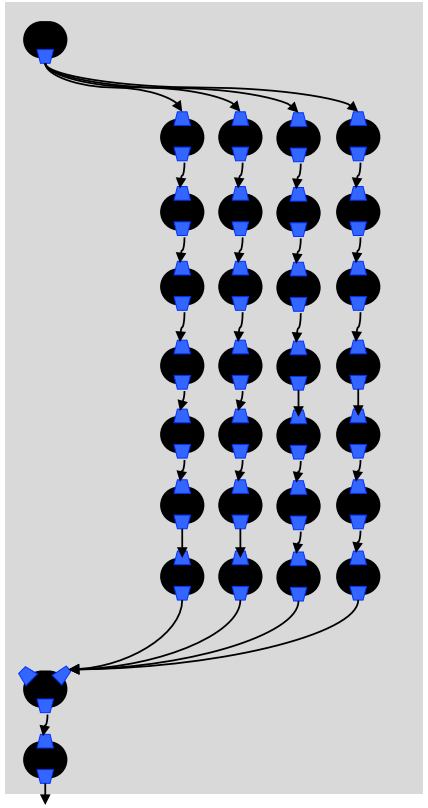


Applications with many operators…



…become PEs with many operators…

…and given an individual PE with many operators on a multicore host, how do we automatically take advantage of thread parallelism?

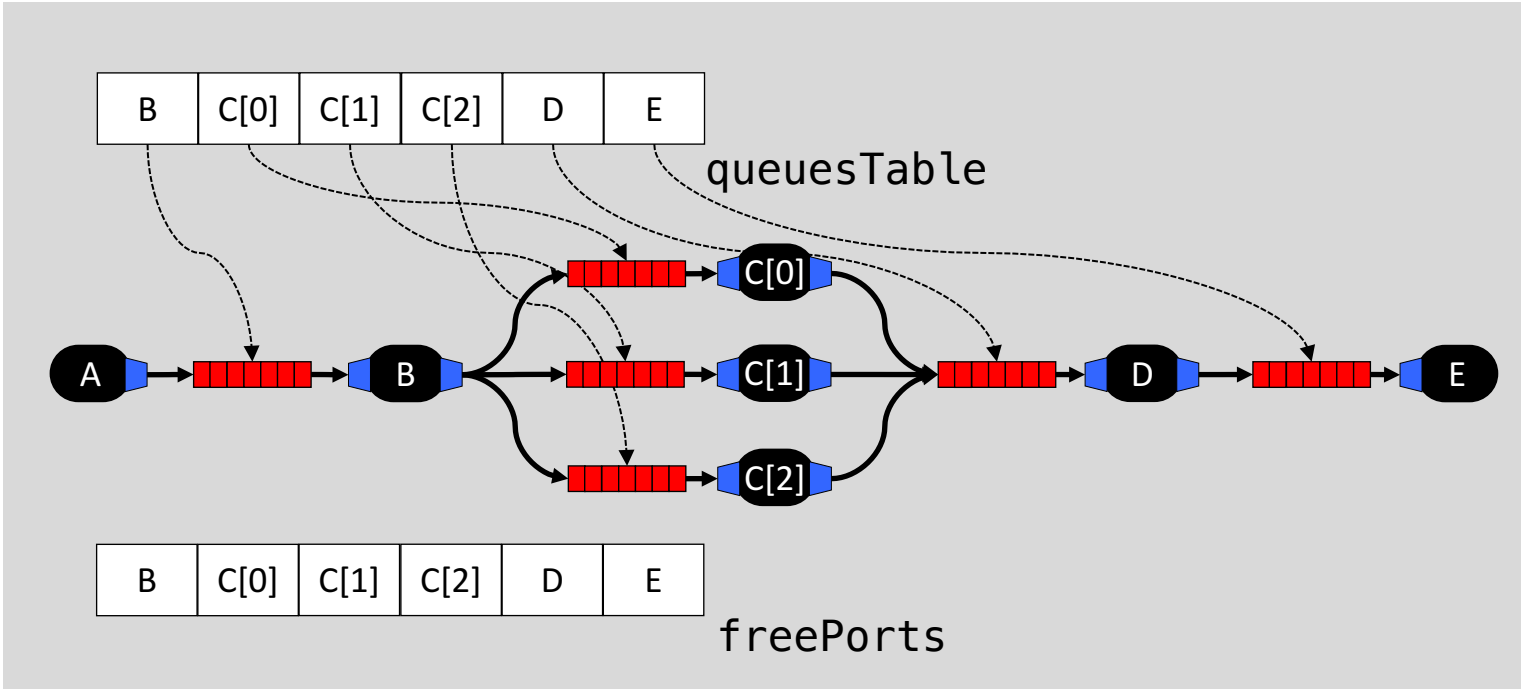


Solution Requirements

- Must be able to add and remove threads dynamically
  - enables elasticity
  - implies threads cannot be statically assigned to operators
- Must maintain tuple order
  - if operator *A* emits tuples a_0, a_1, …, a_n, and operator *B* consumes those tuples, it must see them in the order a_0, a_1, …, a_n
- Must scale as we add threads
  - no global locks
  - minimize global data access
- Cannot change existing API
  - customers have existing applications which we cannot break
  - all changes must be in the SPL runtime *without* changing the operator API

Solution Overview: Startup

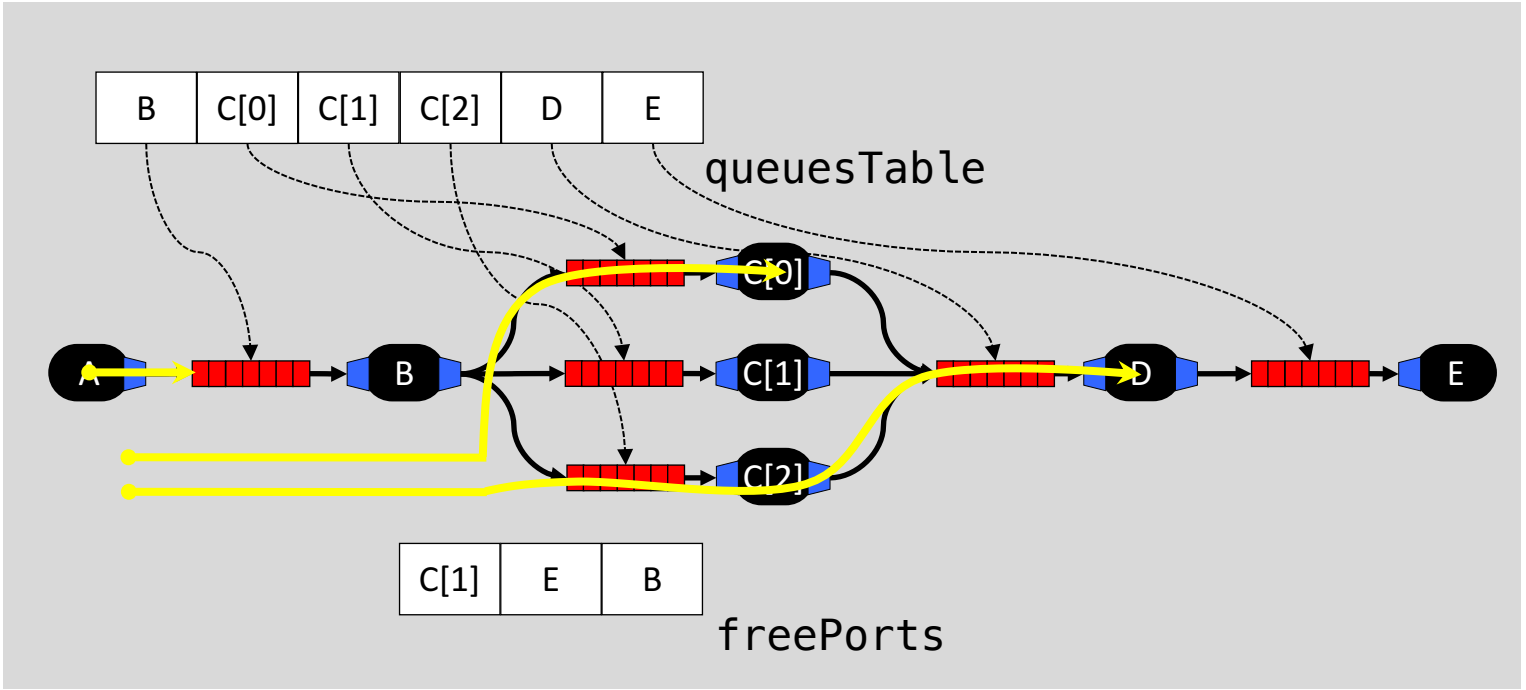
`queuesTable` is a static map from an operator input port to its lock-free, FIFO queue



`freePorts` is a lock-free, FIFO list of all operators input ports which *may* be free to execute; initialized to all operators in PE

Solution Overview: Execution

`queuesTable` does not change

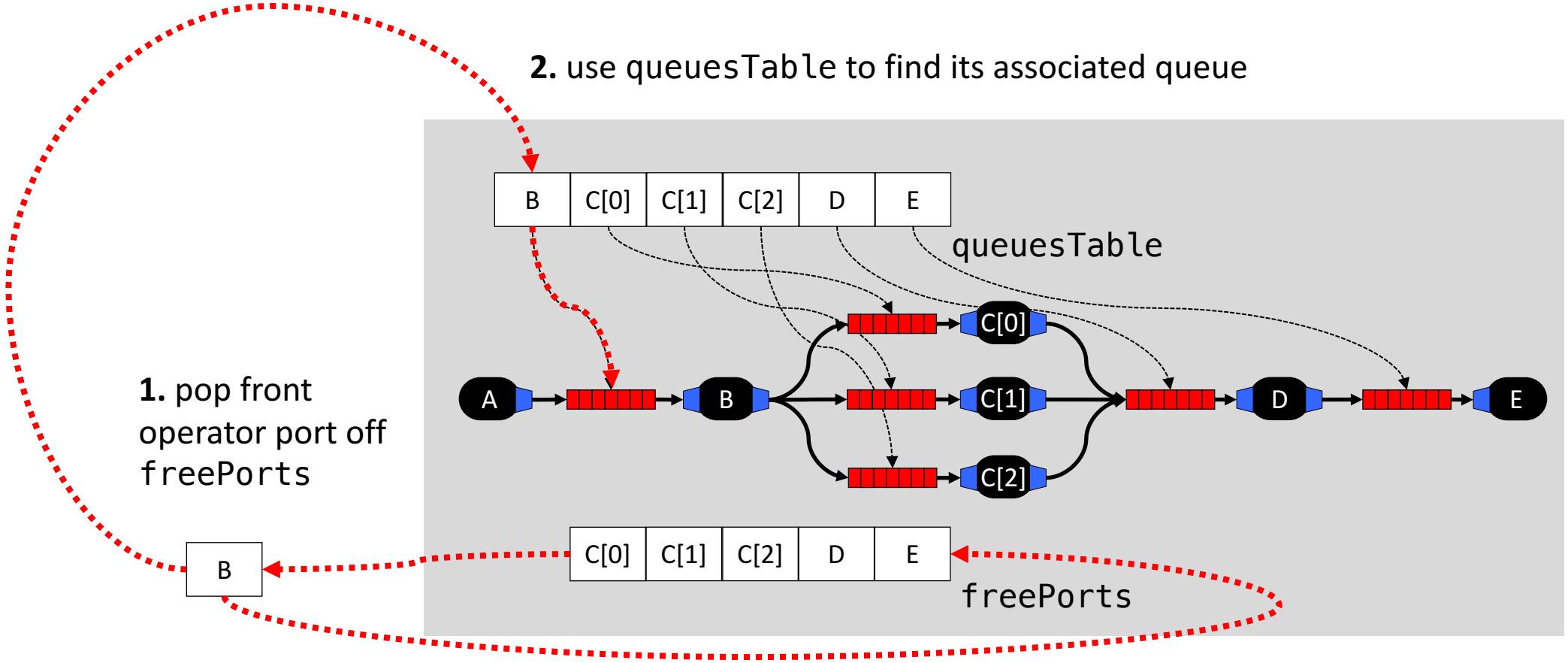


PE maintains a dynamic thread pool; threads will try to clear queues before going back to `freePorts`

threads that hit a full queue may execute that operator to free up the queue

PE pops operators to execute from front of `freePorts`, pushes previously executed or not-actually-free operators to back

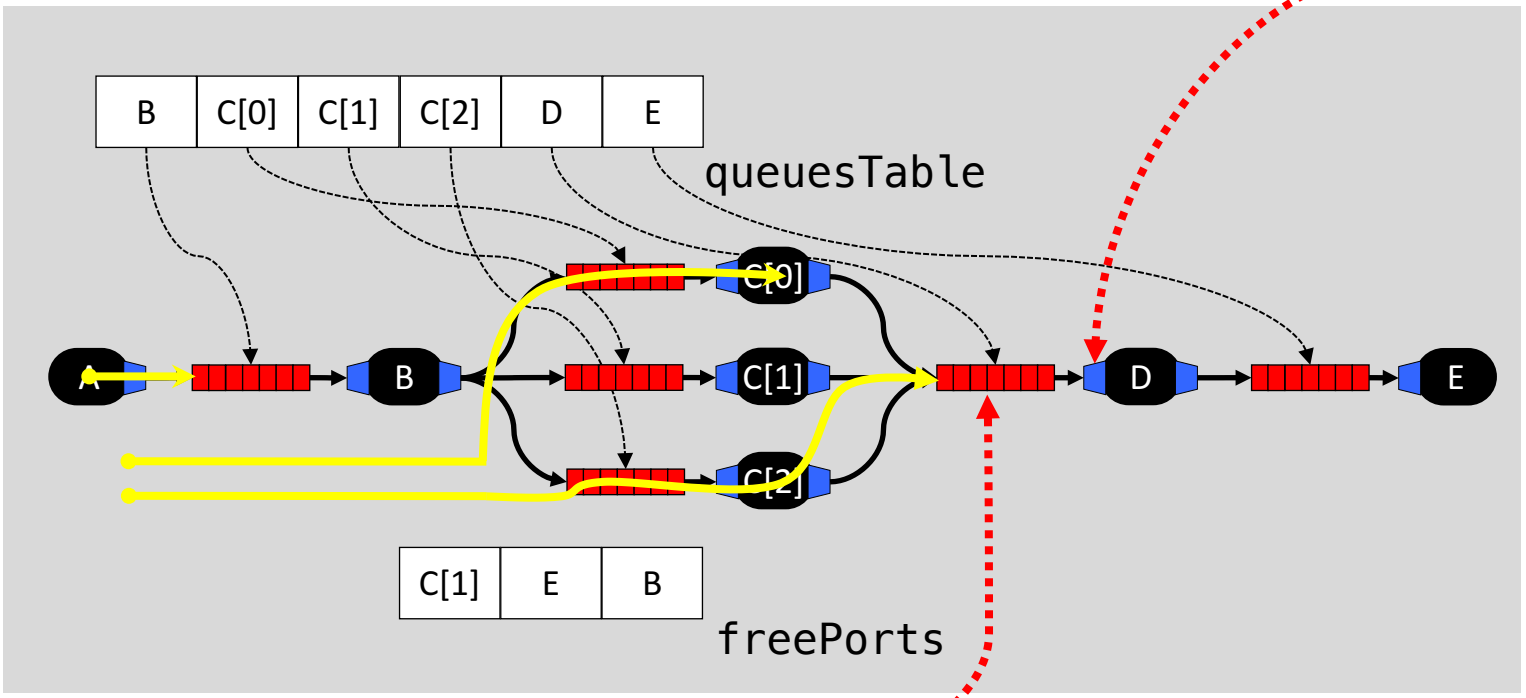
Scheduling: Finding Work



3. ask: is it actually free? yes, drain queue by executing port, then push to back of `freePorts`; no, just push to back of `freePorts`; go to **1.**

Scheduling: Pushing Tuples

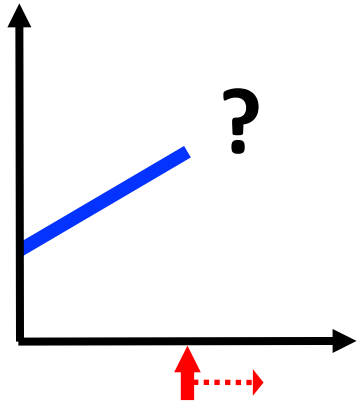
2. …try to push the tuple into the queue; if it succeeds we're done; if not…



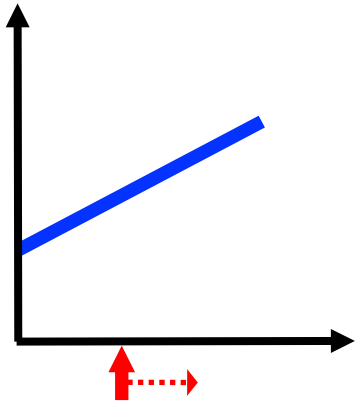
1. thread tries to push tuple into queue, but queue is full; loop over…

3. …check if the port is free; if yes, pop a bunch of tuples and execute port to free up space; in either case, go to **1**

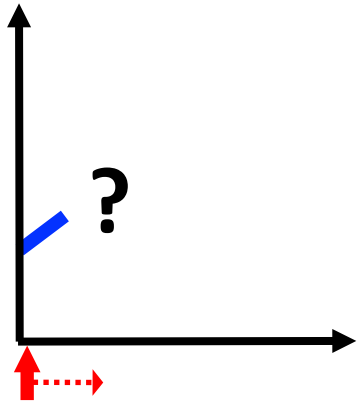
Elasticity Rules



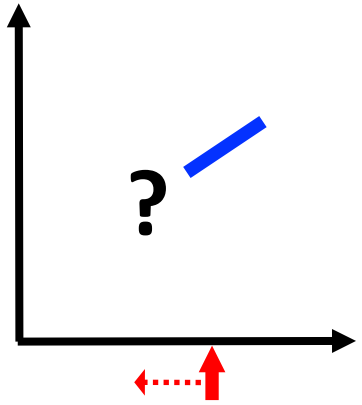
if upward trend below, and don't trust data above, go up



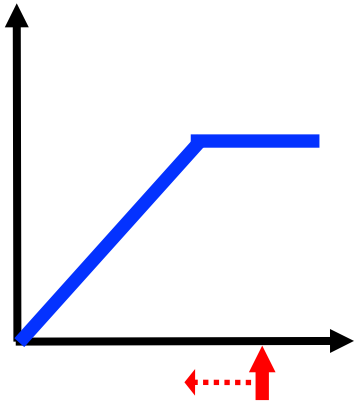
if upward trend above, and do trust data above, go up



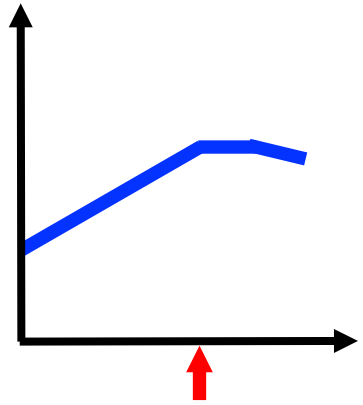
at level 1, if don't trust data above, go up



if don't trust data below, go down

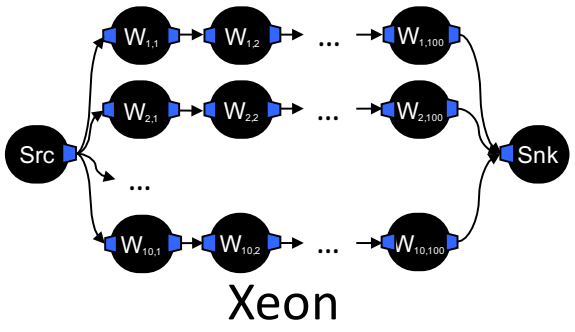


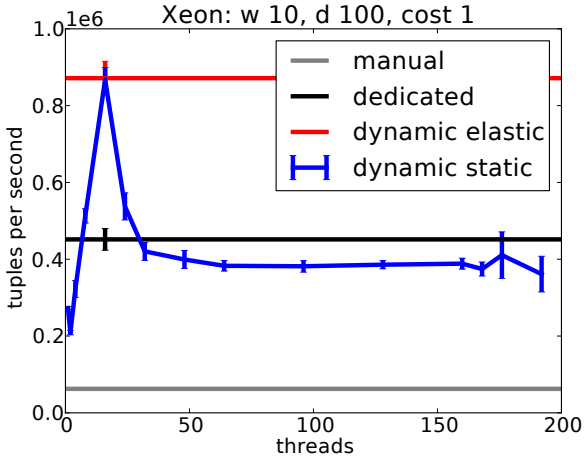
if there is no upward trend from below, go down

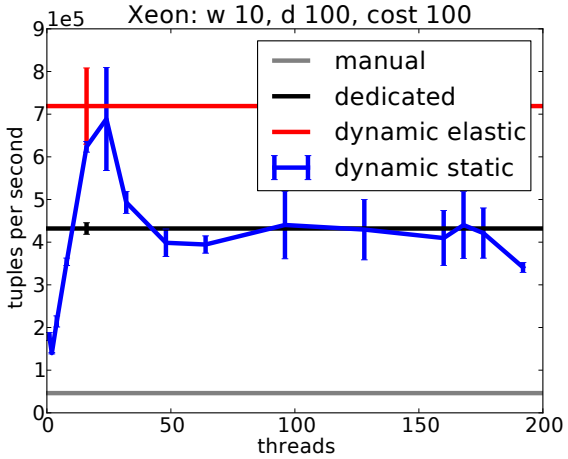


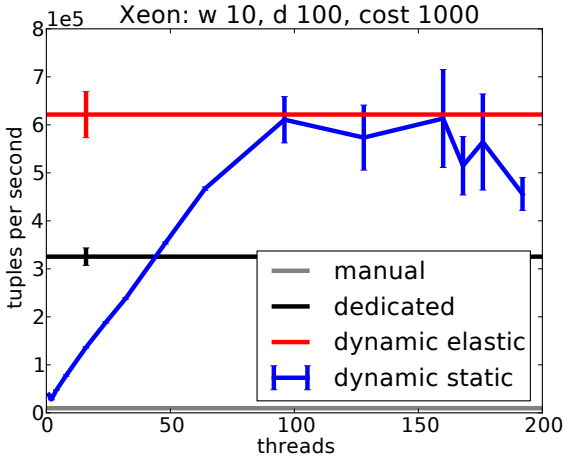
if other conditions are not met, stay at same level

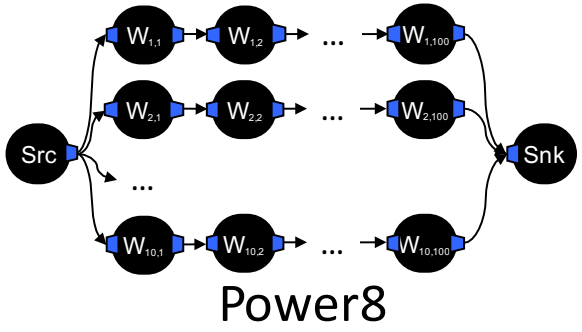
Experiments: Mix

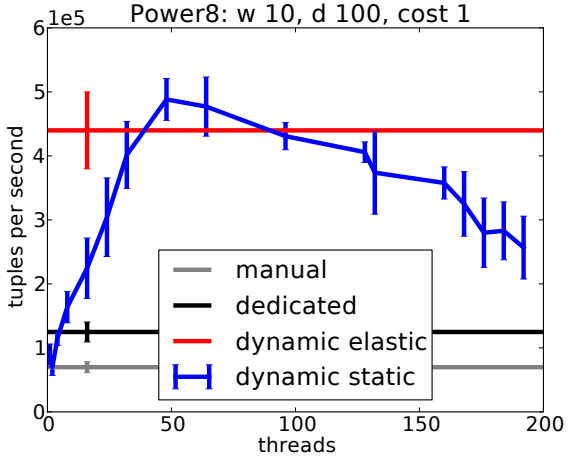


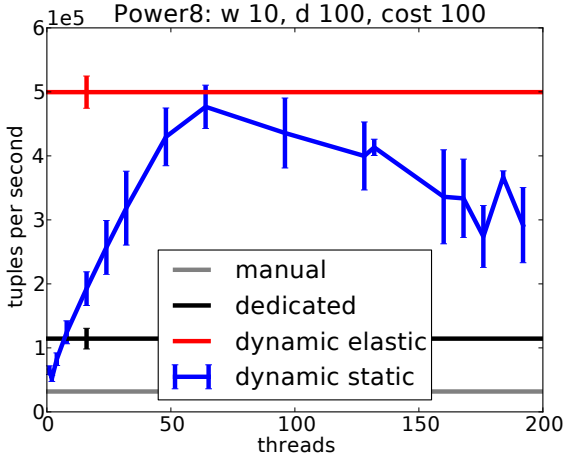


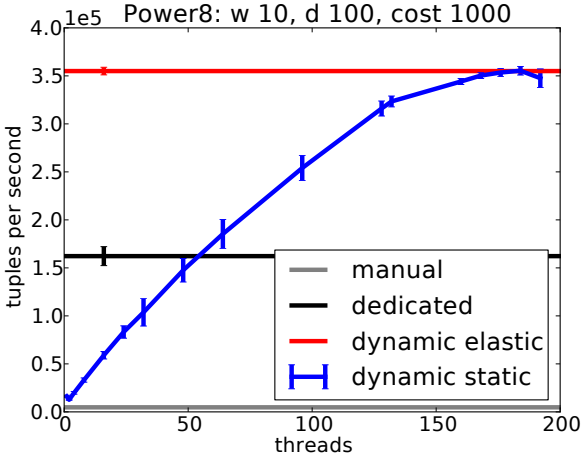




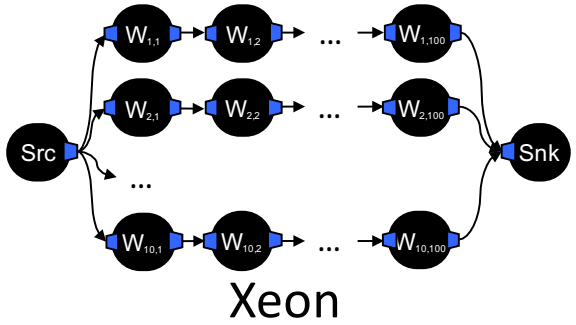


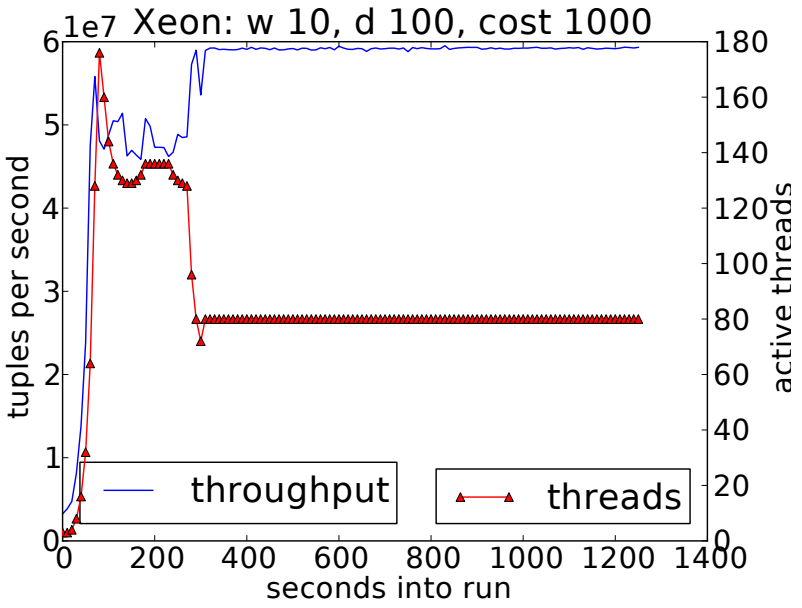


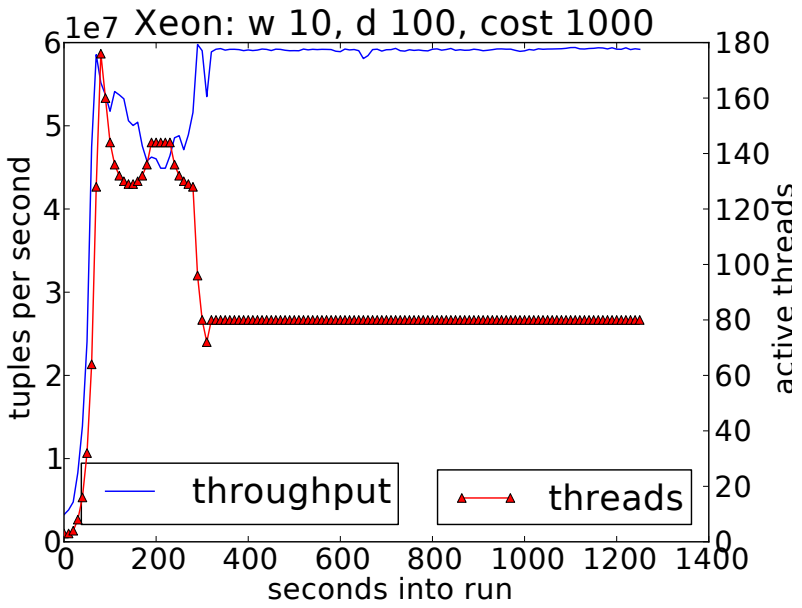


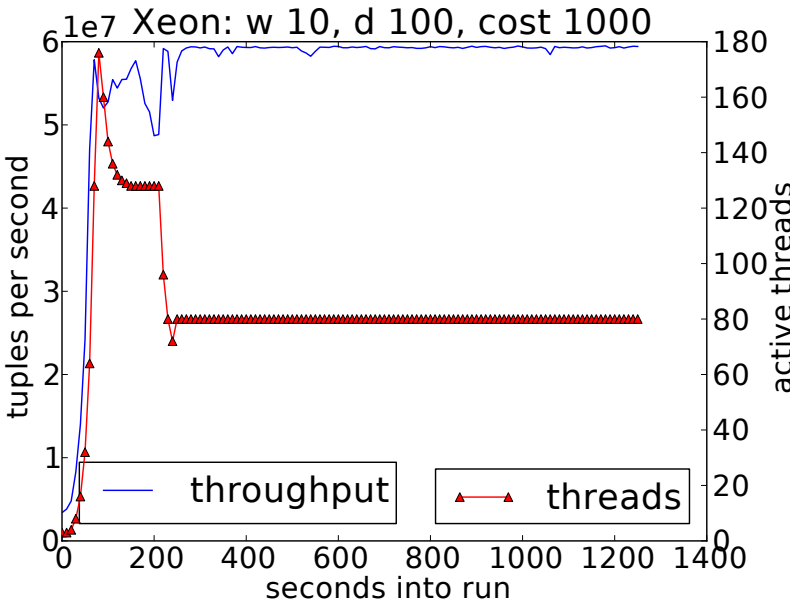


Experiments: Mix, Elasticity

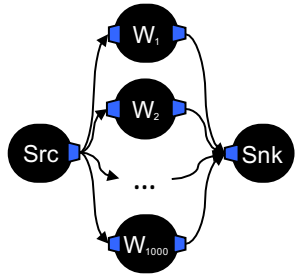




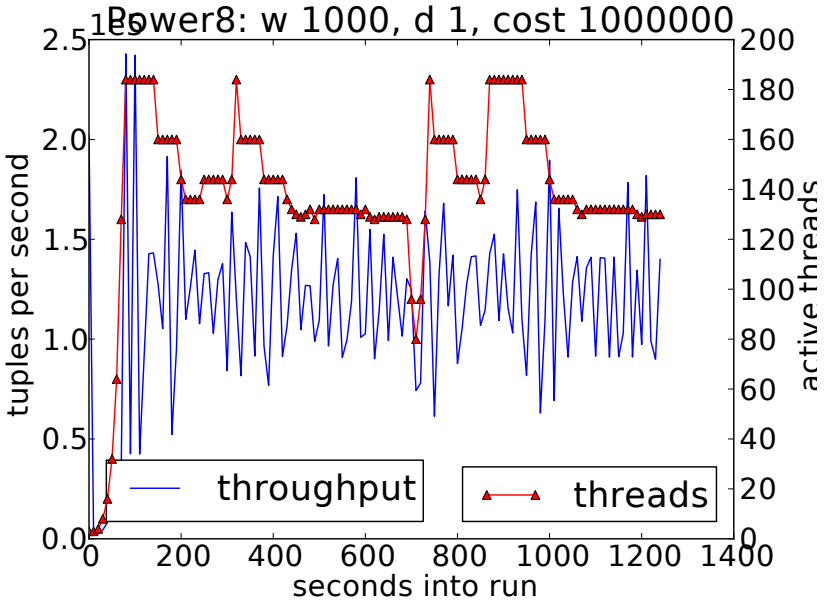


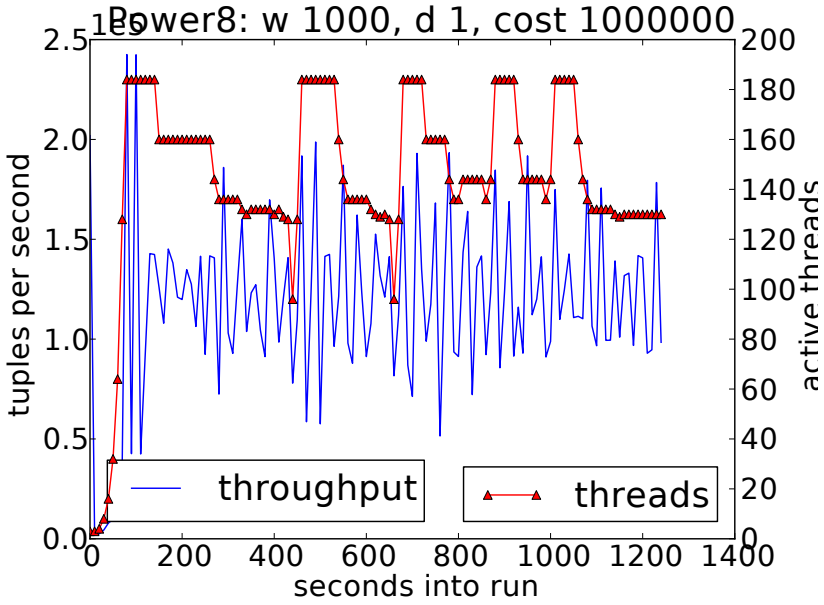


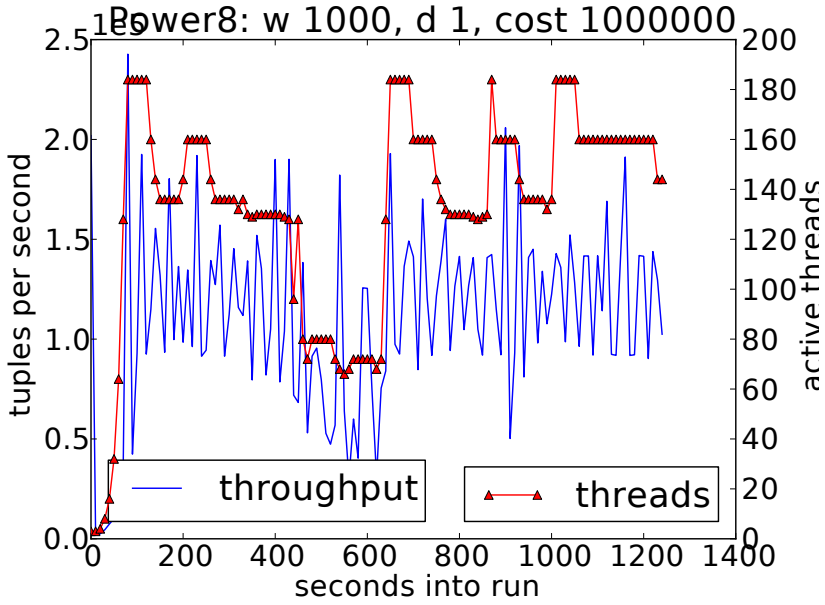
Experiments: Pure Data-Parallel, Elasticity



Power8

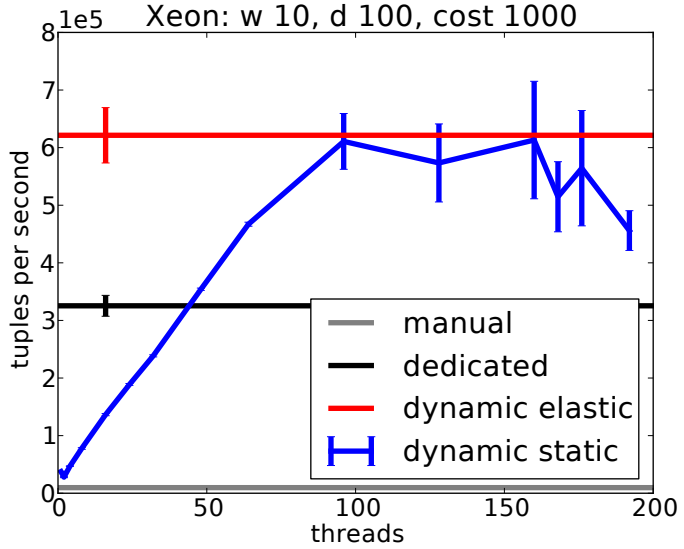


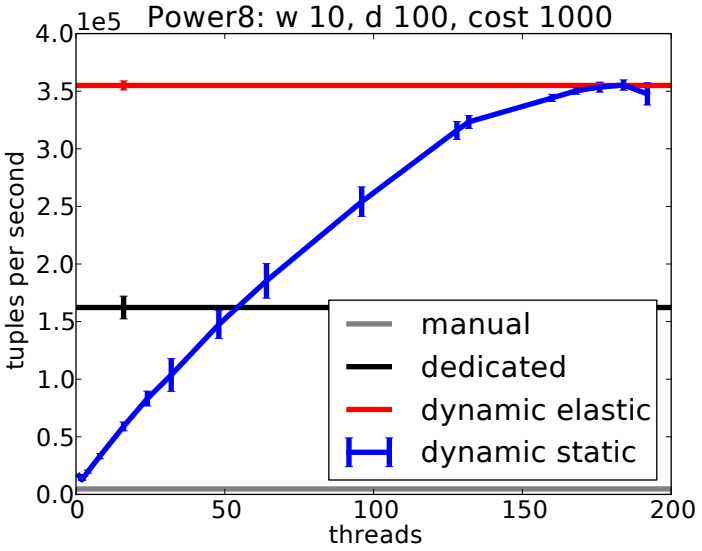




Conclusions Again

- Streaming scheduler that scales to thousands of operators and hundreds of threads
  - minimizes locks, synchronization and global data access
  - interface must maintain backwards compatibility with previous versions
- Elasticity algorithms seek out “good enough” thread levels
  - establishes performance improvement trends; trust (or not) of prior data
  - seeks highest thread level with no further improvement trend that it trusts





Backup

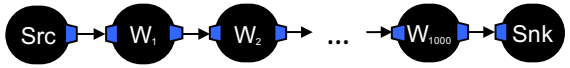
SPL: Threading Models

```
@threading(model=dynamic, threads=8, elastic=false)
composite Main {
graph
  stream<Data> Src = Beacon() {}
  stream<Data> Res = ManyOperators(Src) {}
  () as Sink = Custom(Res) {}
}
```

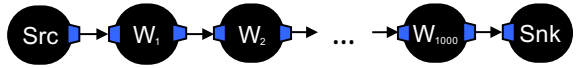
```
@threading(model=manual)
composite Ingest(output Out) {
graph
  stream<Data> Src = SpecializedSource() {
    config placement: host(InHosts), partitionColocation("ingest");
  }
  stream<Data> Parsed = SpecializedHeavyParsing(Src) {
    config threadedPort: queue(Src, Sys.Wait);
           placement: partitionColocation("ingest");
  }
  stream<Data> Filtered = Filter(Parsed) {
    param filter: Parsed.age > 13;
    config placement: partitionColocation("ingest");
  }
  stream<Data> Out = SpecializedLightParsing(Filtered) {
    config placement: partitionColocation("ingest");
  }
}

@threading(model=automatic)
composite Main {
graph
  stream<Data> In = Ingest() {}
  stream<Data> Processed = Processing(In) {}
  () as Sink = Egress(Processed) {}
  config hostPool: InHosts = createPool({size=1u, tags=["ingest"]},
                                        Sys.Exclusive);
}
```

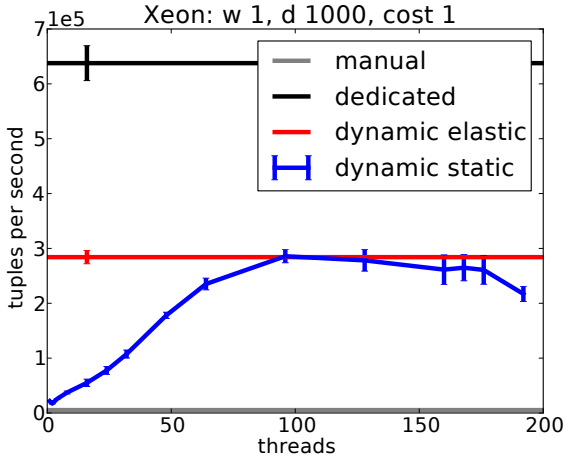
Experiments: Pure Pipeline

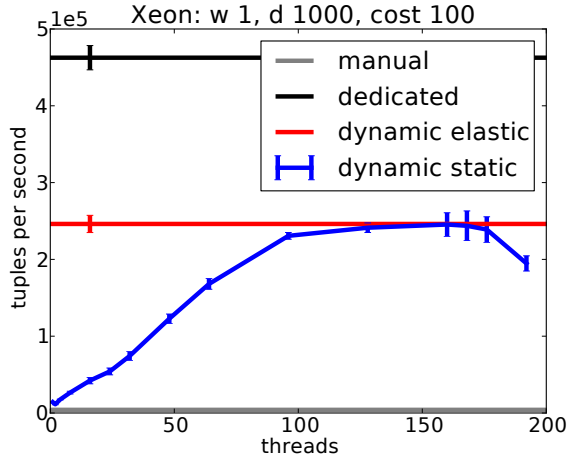


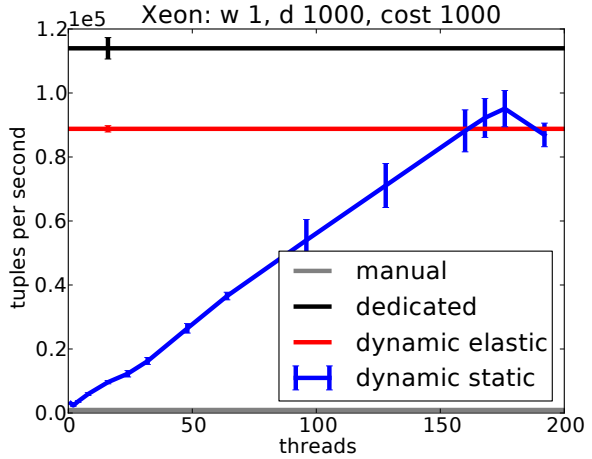
Xeon

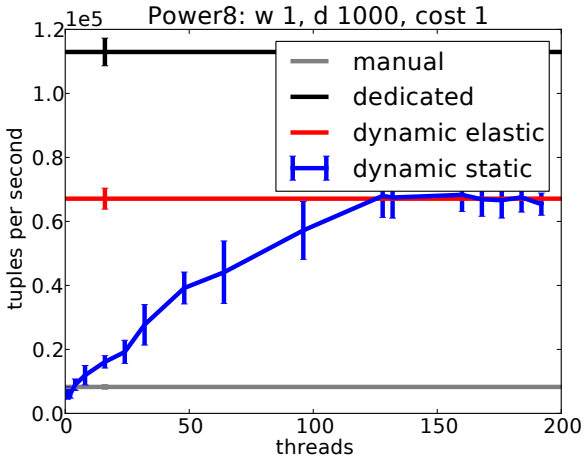


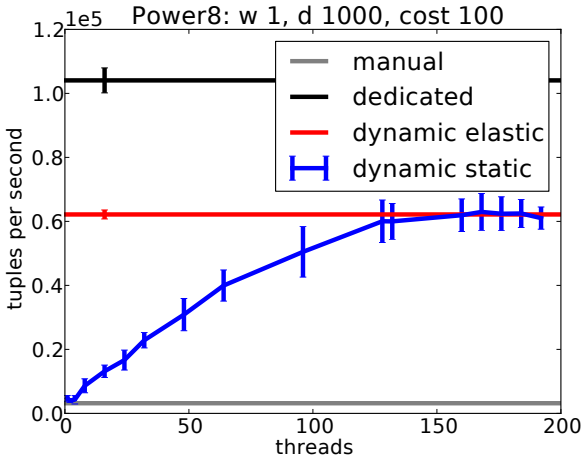
Power8

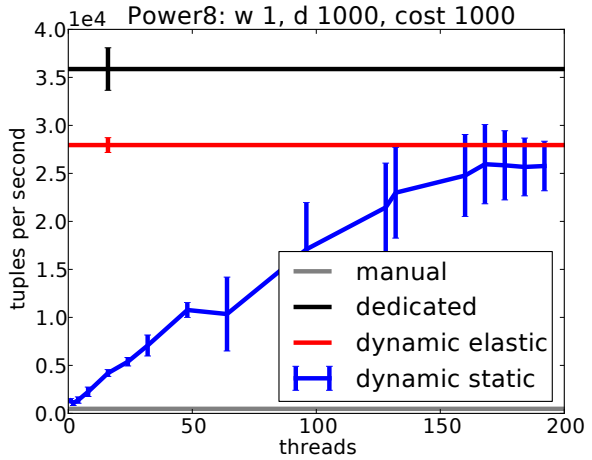




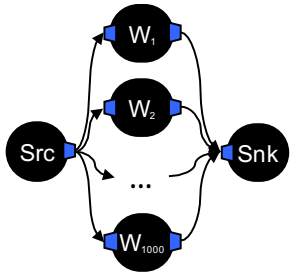




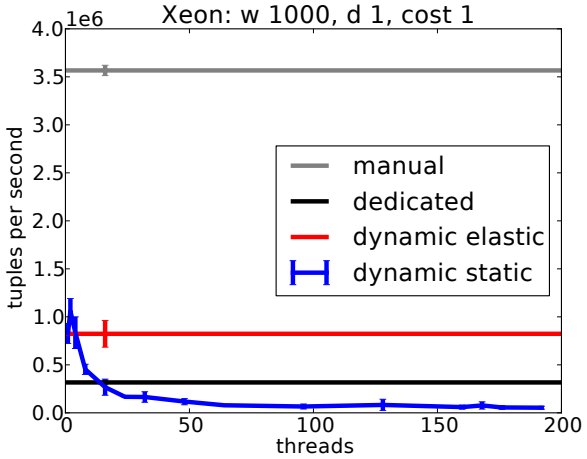


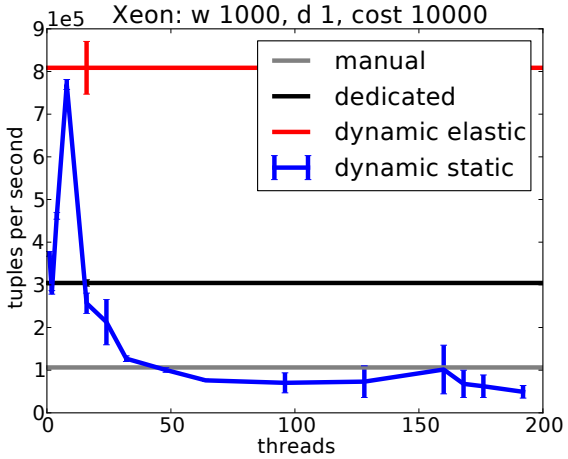


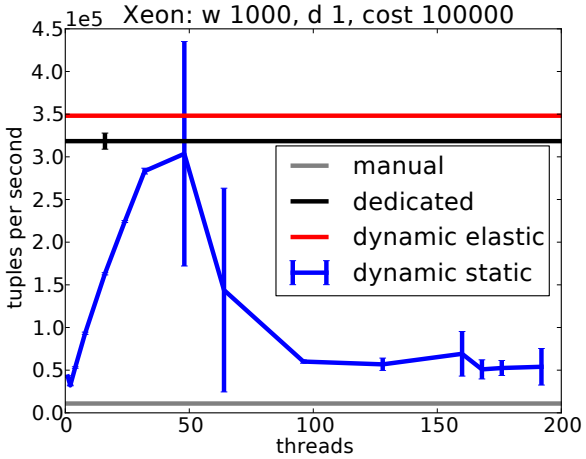
Experiments: Pure Data-Parallel

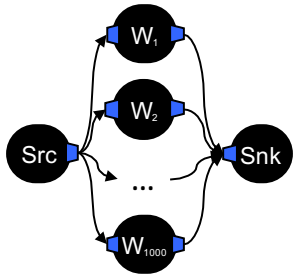


Xeon

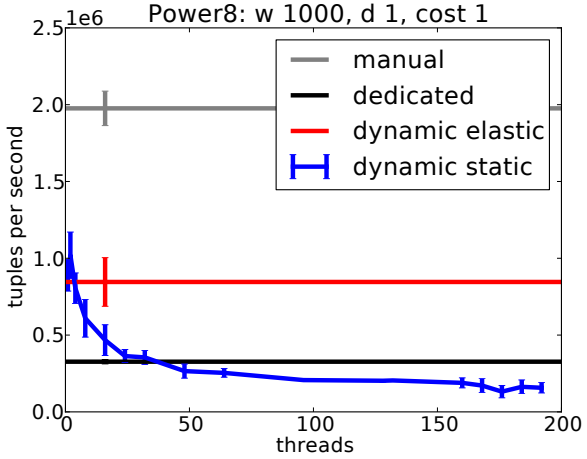


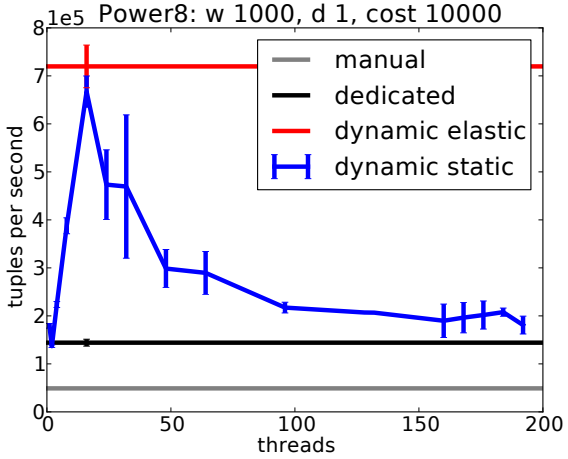


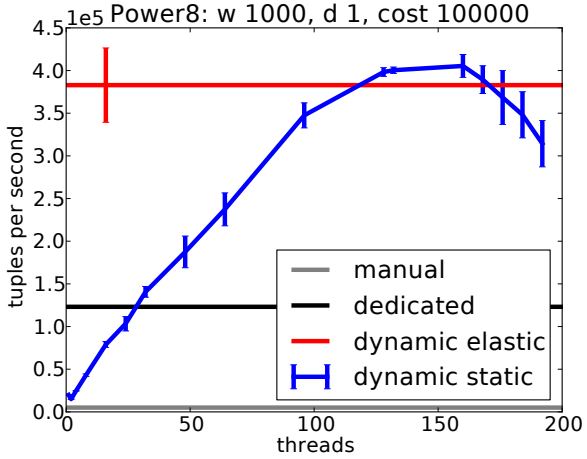




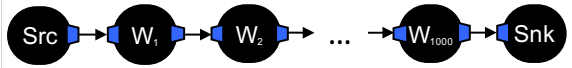
Power8







Experiments: Pure Pipeline, Elasticity



Xeon