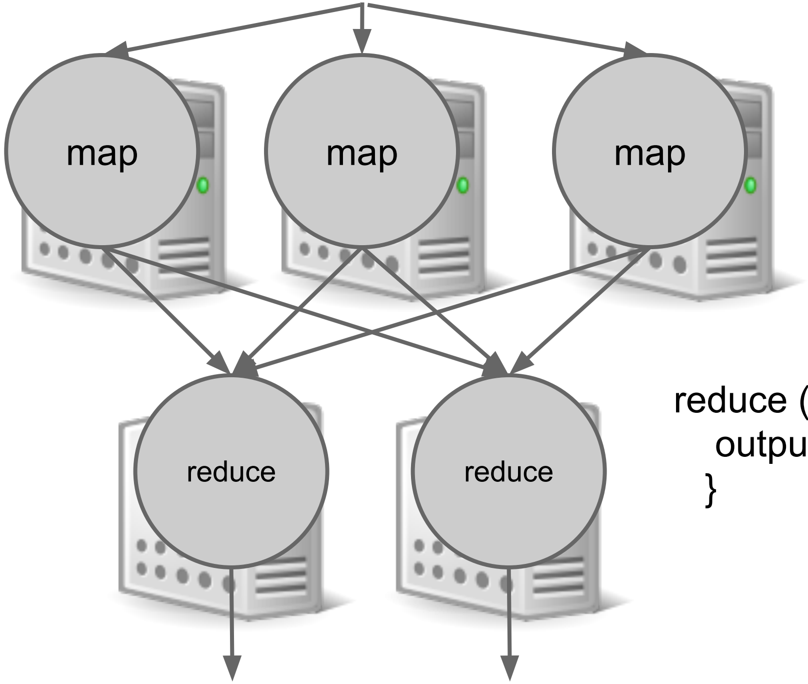
Naiad: Timely Dataflow

Frank McSherry, Rebecca Isaacs, Michael Isard, and Derek G. Murray, Composable Incremental and Iterative Data-Parallel Computation with Naiad, MSR-TR-2012-105, 2012.

Derek Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, M. Abadi: Naiad: A Timely Dataflow System, SOSP, 2013.

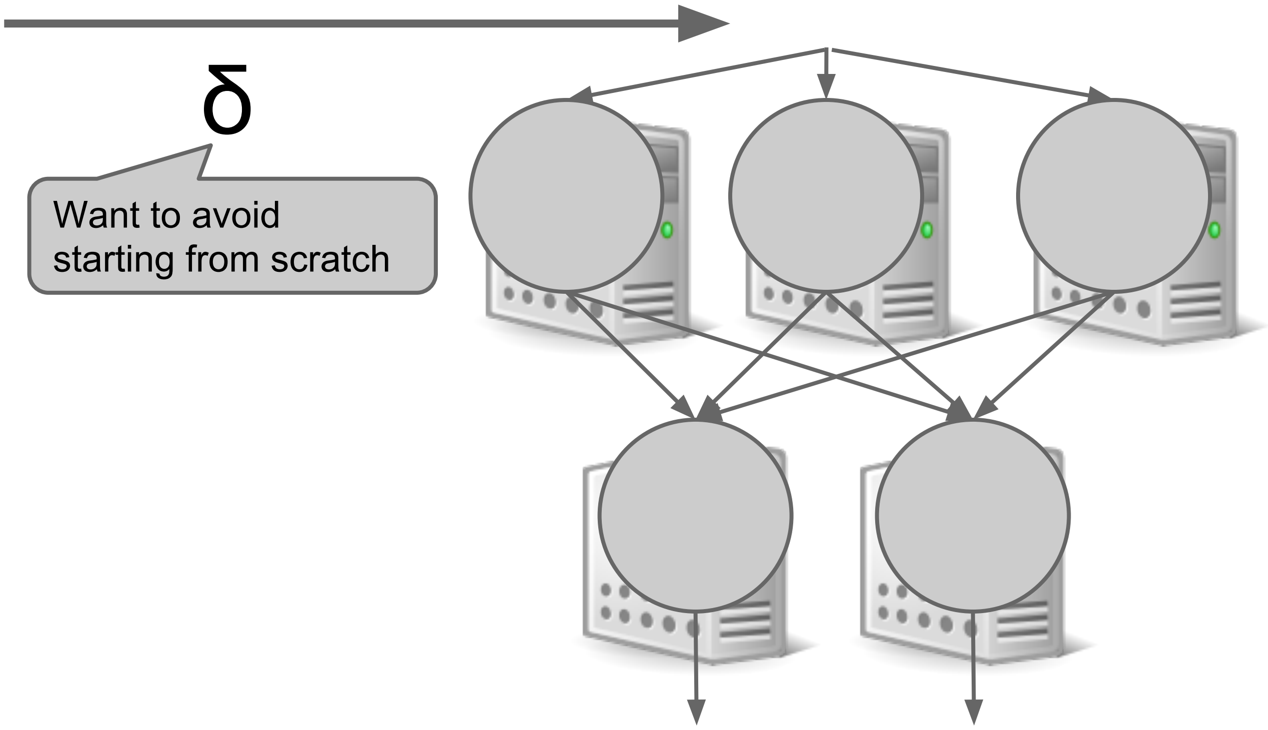
Distributed Dataflow Programming

```
map (key, value, context)  {
    words =value.split(‘ ‘);
    foreach (word in words) {
     context.write(word, 1);
    }
  }
```

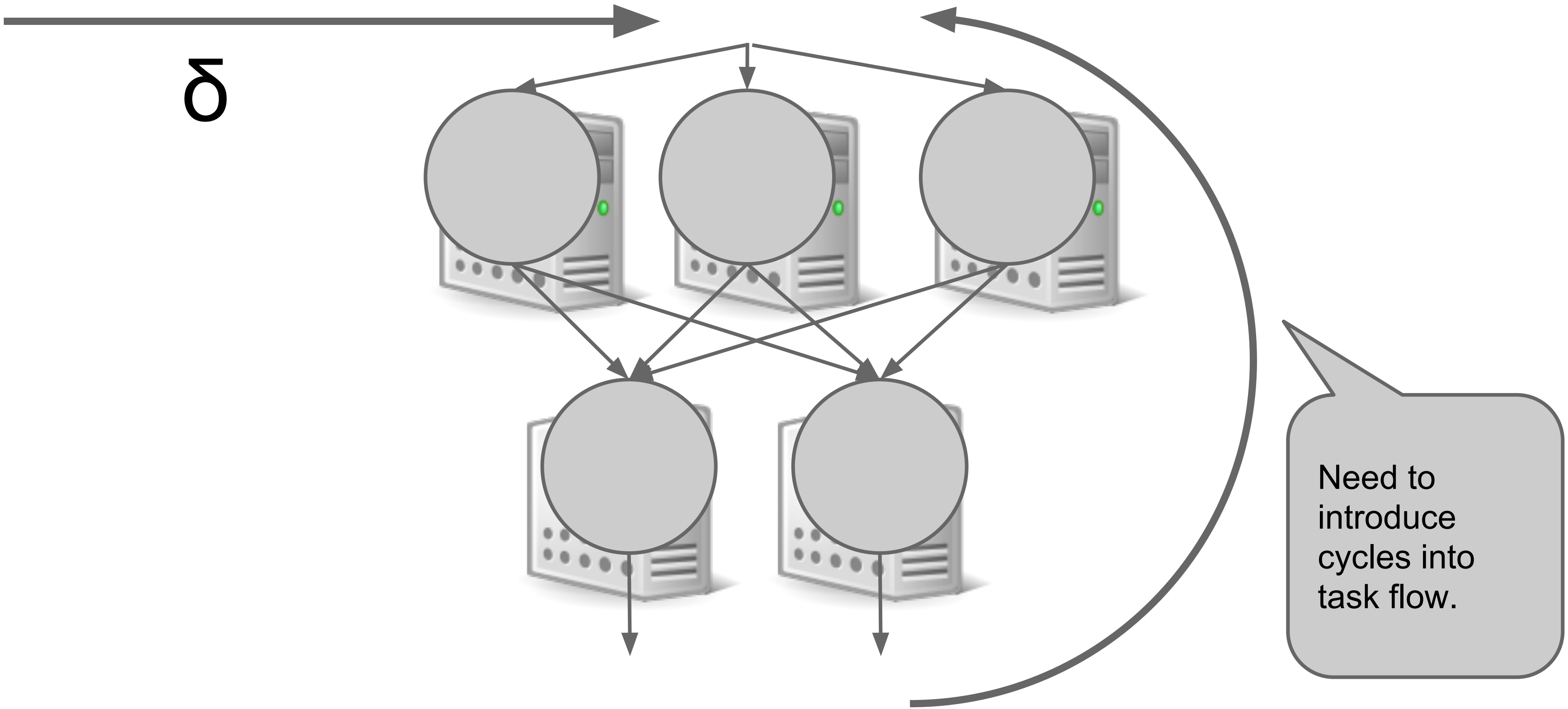



```
reduce (key, values, output)  {
    output.collect(key, values.length);
   }
```

Incremental and Iterative Processing



Incremental and Iterative Processing



Incremental Computation

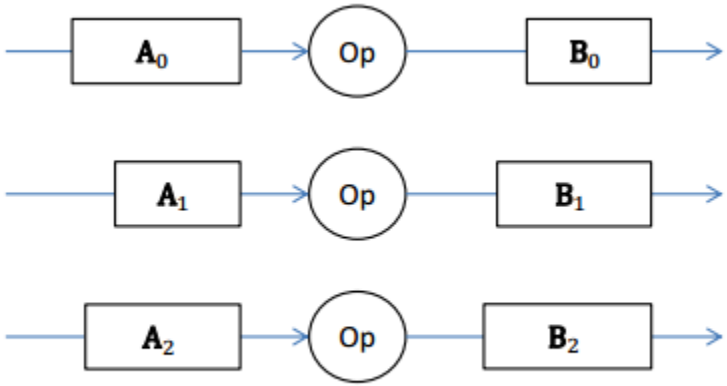


Figure 2: A sequence of input collections $\mathbf{A}_0, \mathbf{A}_1, \ldots$ and the corresponding output collections $\mathbf{B}_0, \mathbf{B}_1, \ldots$. Each is defined independently as $\mathbf{B}_t = \mathrm{Op}(\mathbf{A}_t)$.

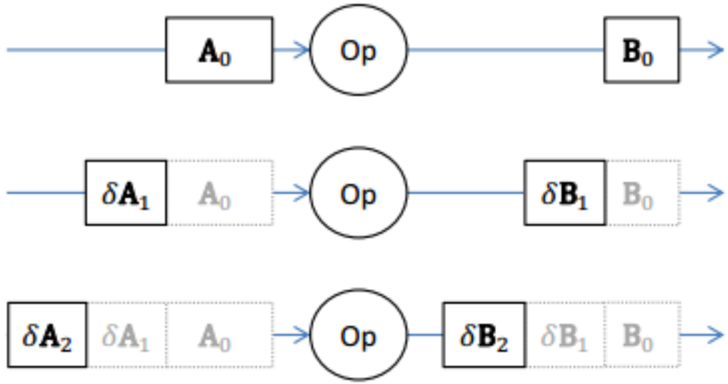
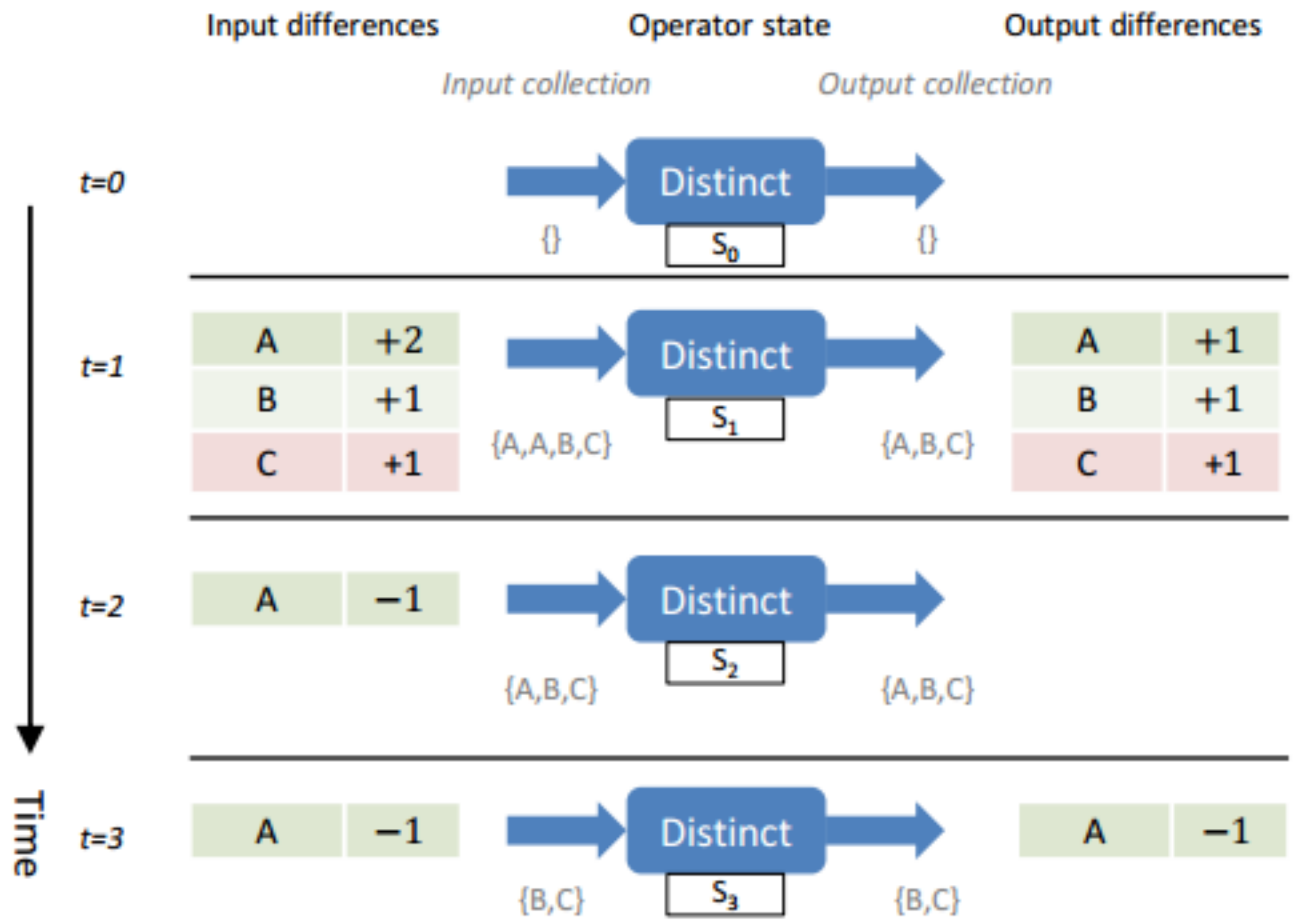
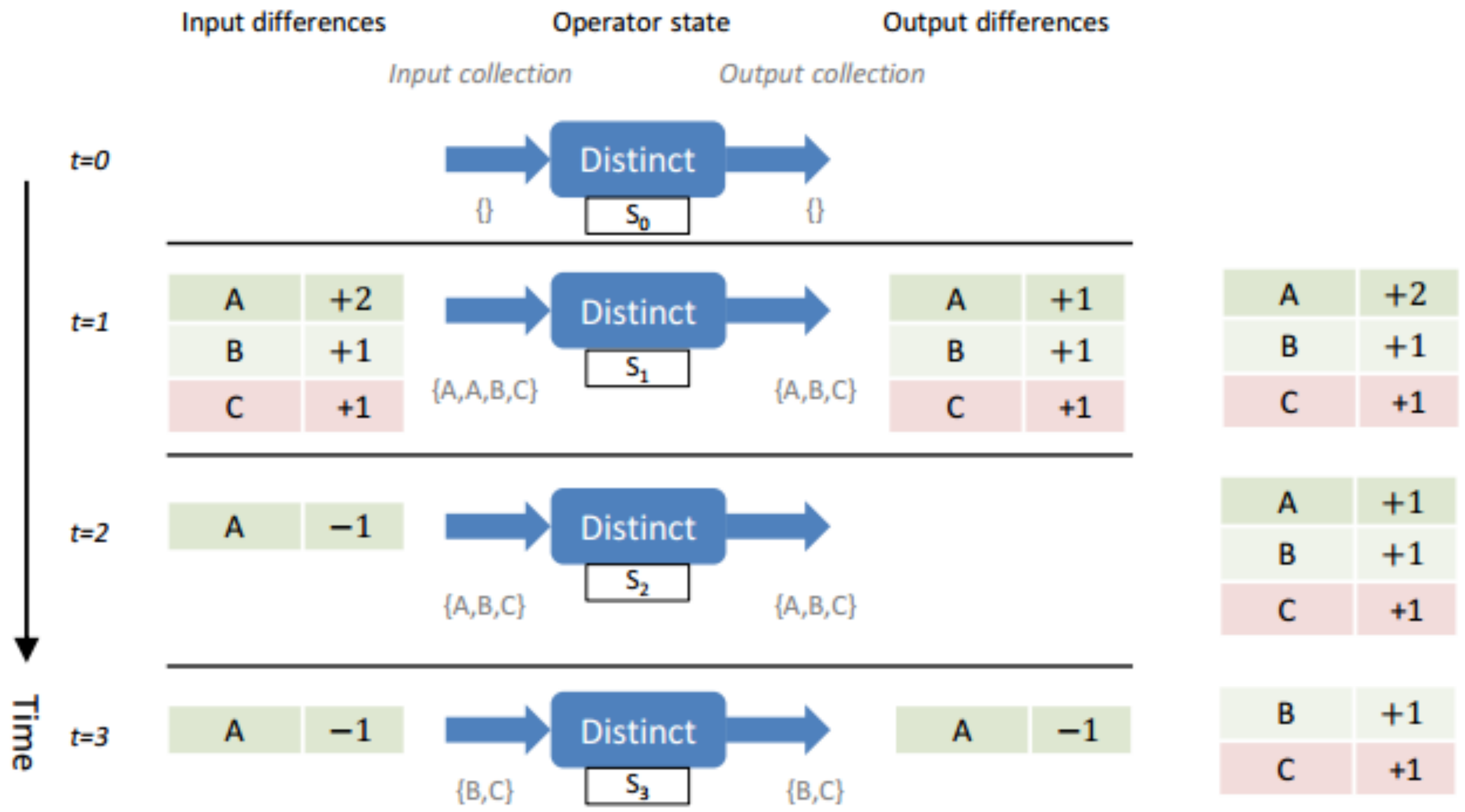


Figure 3: The same sequence of computations as in Figure 2, presented as differences from the previous collections. The outputs still satisfy $\mathbf{B}_t = \mathrm{Op}(\mathbf{A}_t)$, but are represented as differences $\delta\mathbf{B}_t = \mathbf{B}_t - \mathbf{B}_{t-1}$.



Input differences
Operator state
Output differences
Input collection
Output collection
t=0
Distinct
{}
S0
{}
t=1
A +2
B +1
C +1
Distinct
{A,A,B,C}
S1
{A,B,C}
A +1
B +1
C +1
t=2
A −1
Distinct
{A,B,C}
S2
{A,B,C}
t=3
A −1
Distinct
{B,C}
S3
{B,C}
A −1
Time

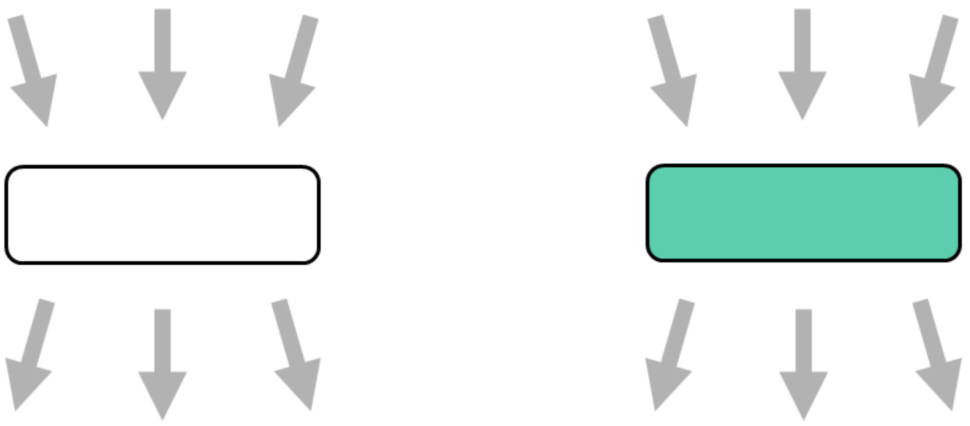


Input differences
Operator state
Output differences
Input collection
Output collection
t=0
Distinct
{}
S0
{}
t=1
A +2
B +1
C +1
Distinct
{A,A,B,C}
S1
{A,B,C}
A +1
B +1
C +1
A +2
B +1
C +1
t=2
A −1
Distinct
{A,B,C}
S2
{A,B,C}
A +1
B +1
C +1
t=3
A −1
Distinct
{B,C}
S3
{B,C}
A −1
B +1
C +1
Time

Synchronous vs Asynchronous

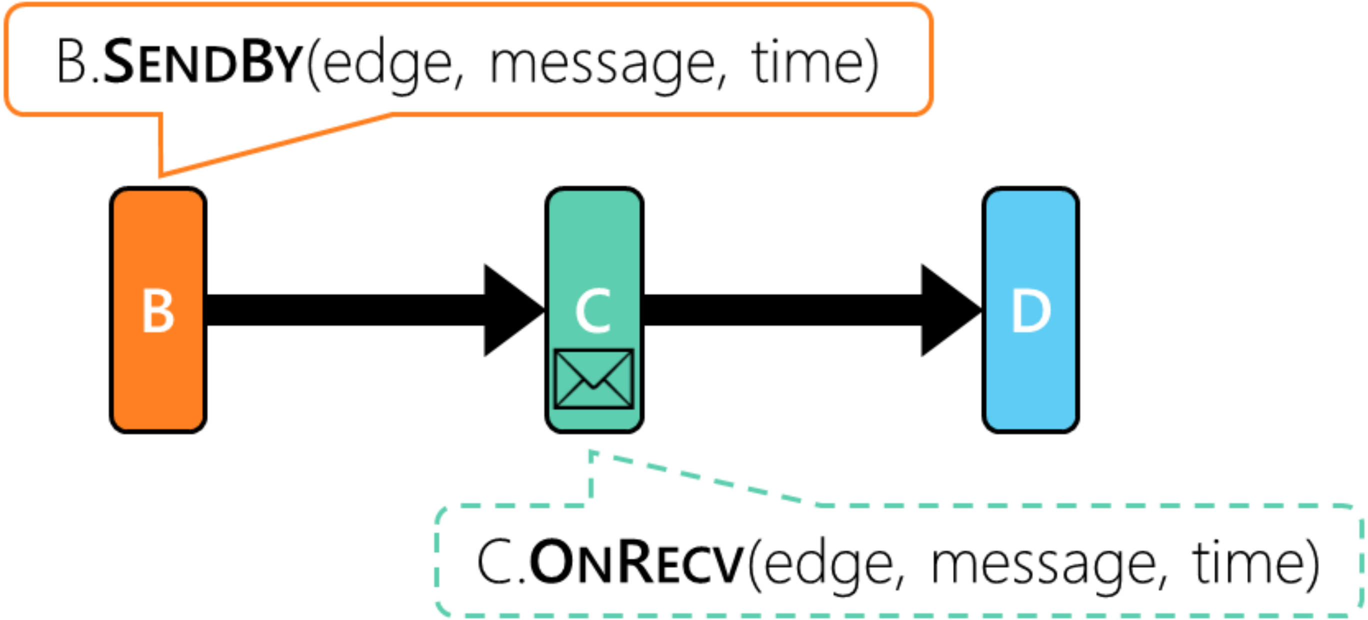
Batching vs. Streaming

(synchronous) (asynchronous)



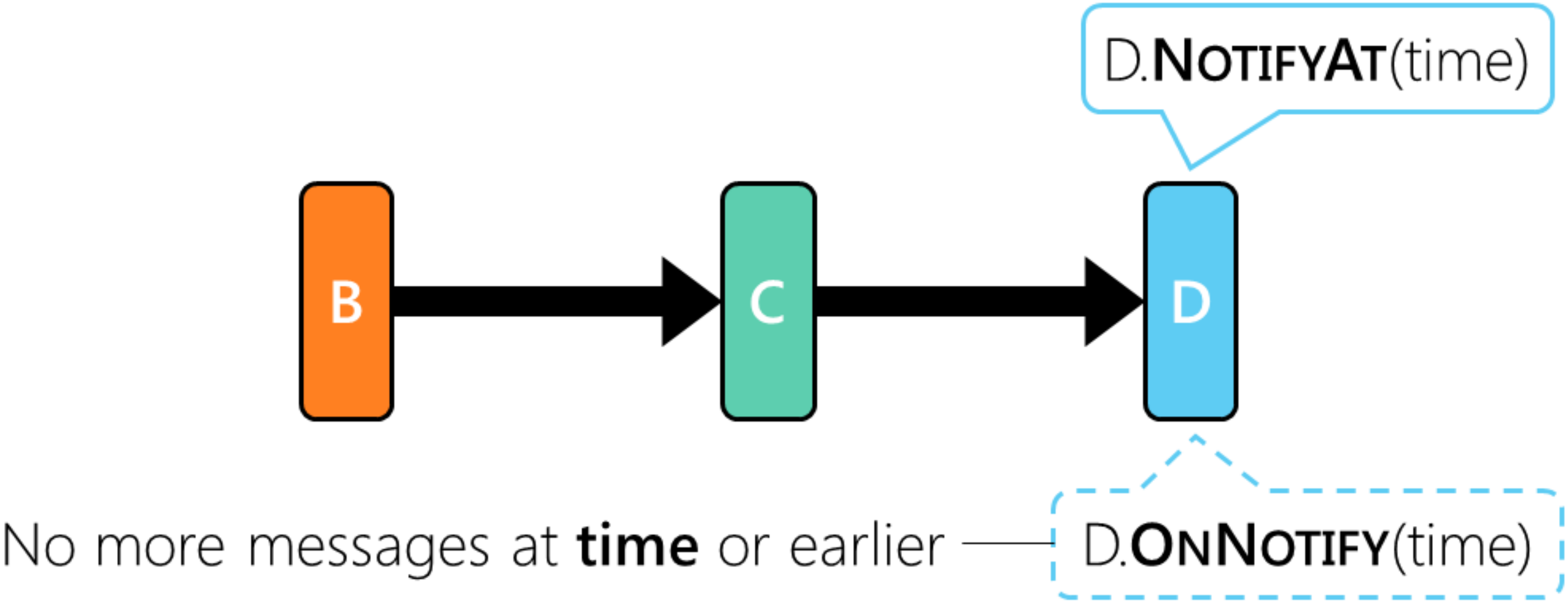
Batching (synchronous)	Streaming (asynchronous)
× Requires coordination	✓ No coordination needed
✓ Supports aggregation	× Aggregation is difficult

Programming Model: Messages

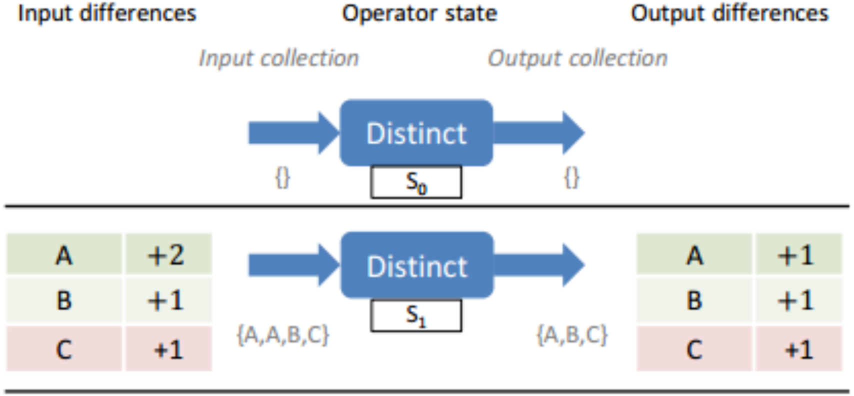


Messages are delivered asynchronously

Programming Model: Notifications



Notifications support batching




```
class DistinctCount<S,T> : Vertex<T>
{
  Dictionary<T, Dictionary<S,int>> counts;
  void OnRecv(Edge e, S msg, T time)
  {
    if (!counts.ContainsKey(time)) {
      counts[time] = new Dictionary<S,int>();
      this.NotifyAt(time);
    }

    if (!counts[time].ContainsKey(msg)) {
      counts[time][msg] = 0;
      this.SendBy(output1, msg, time);
    }

    counts[time][msg]++;
  }

  void OnNotify(T time)
  {
    foreach (var pair in counts[time])
      this.SendBy(output2, pair, time);
    counts.Remove(time);
  }
}
```

Differential Computation

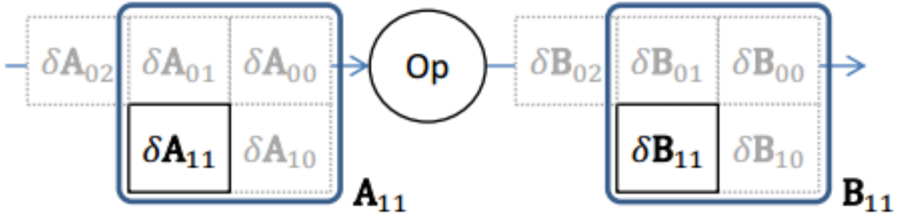
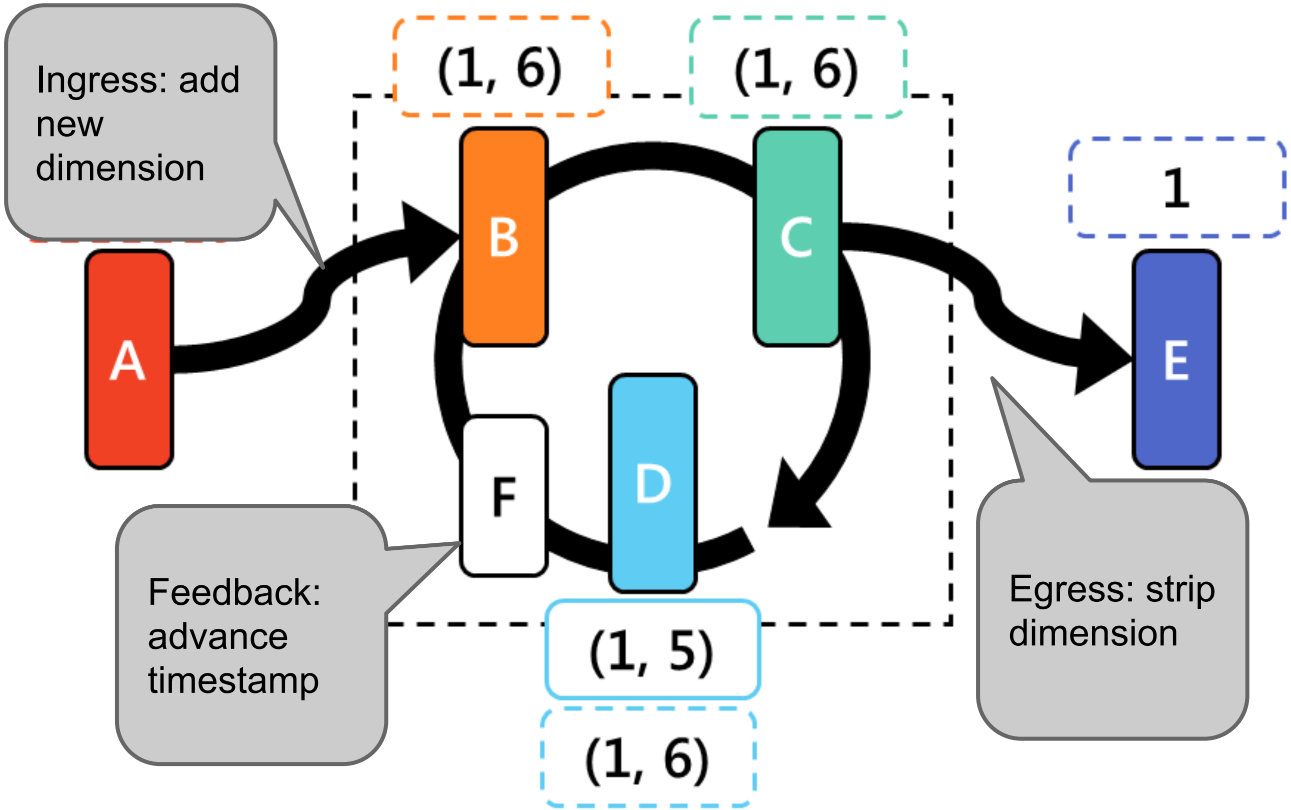


Figure 4: Differential computation in which multiple independent collections $B_{ij} = \text{Op}(A_{ij})$ are computed. The rounded boxes indicate the differences that are accumulated to form the collections A_{11} and B_{11}.

Revisiting Iteration

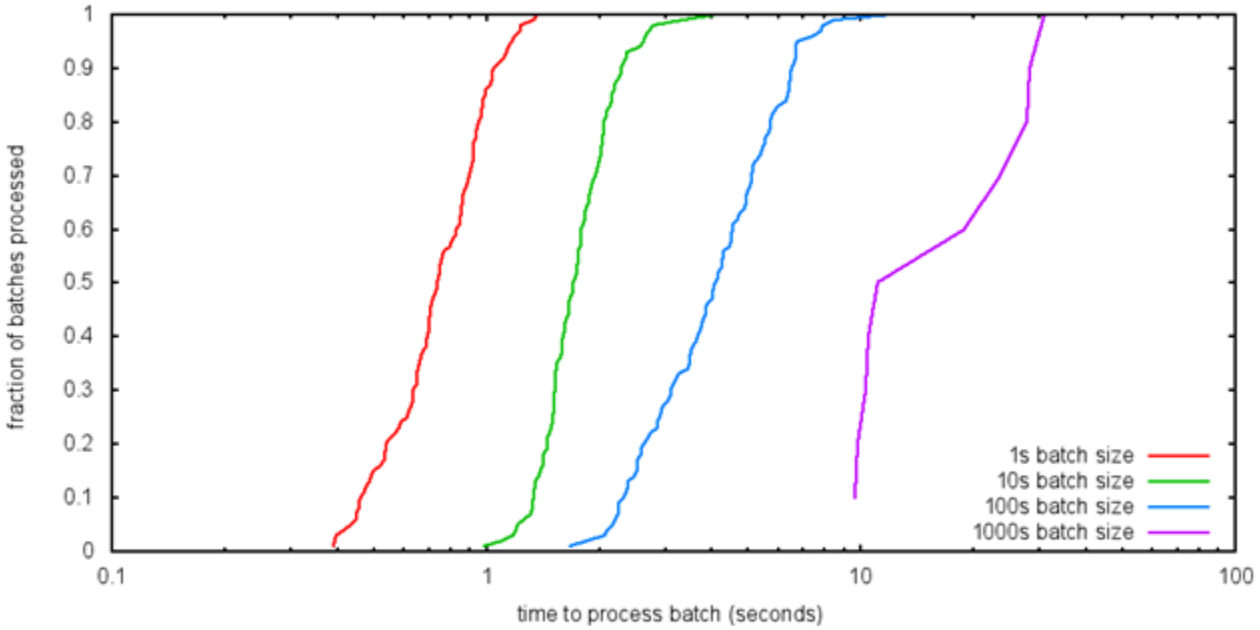


Engineering for low latency

- Reduce TCP delayed ACK and retransmit times
- Use finer grain scheduling timers to reduce impact of data structure contention
- Reduce impact of garbage collection by modifying GC parameters and utilising reusable types.

Evaluation

CDFs for 24 hour windowed SCC of @mention graph.

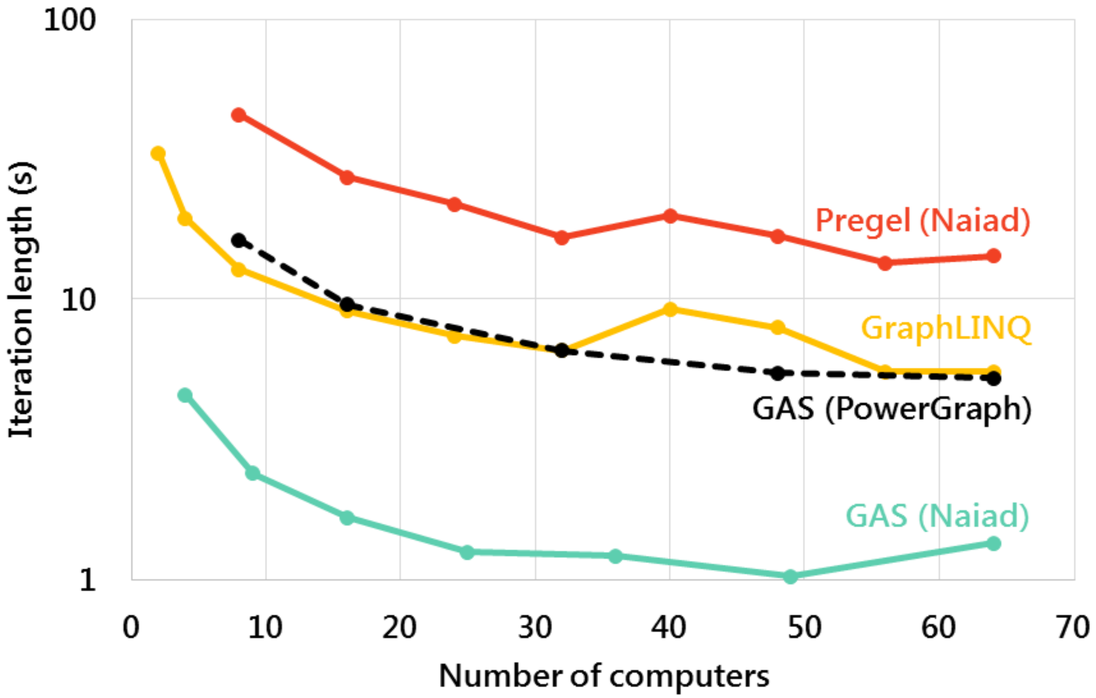


Evaluation

PageRank

Twitter graph
42 million nodes
1.5 billion edges

64 × 8-core 2.1 GHz AMD Opteron
16 GB RAM per server
Gigabit Ethernet



Evaluation

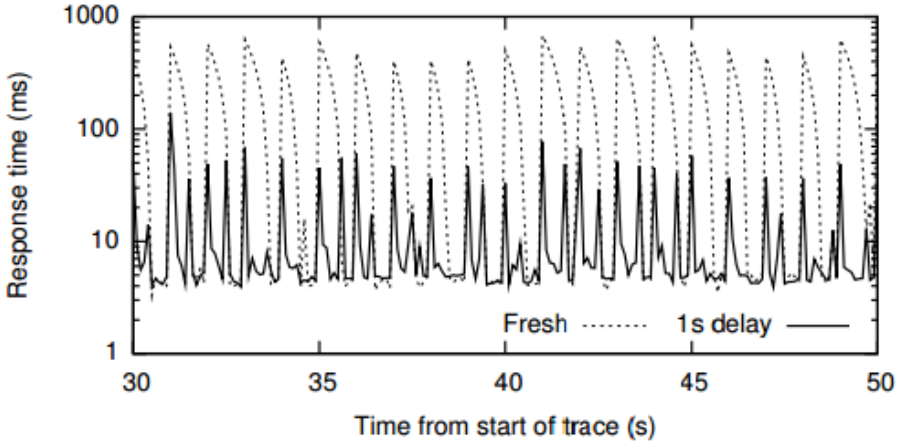


Figure 8: Time series of response times for interactive queries on a streaming iterative graph analysis (§6.4). The computation receives 32,000 tweets/s, and 10 queries/s. "Fresh" shows queries being delayed behind tweet processing; "1 s delay" shows the benefit of querying stale but consistent data.