

# Naiad: A Timely Dataflow System

Derek G. Murray  
Michael Isard

Frank McSherry  
Paul Barham


Rebecca Isaacs  
Martin Abadi

Microsoft Research Silicon Valley

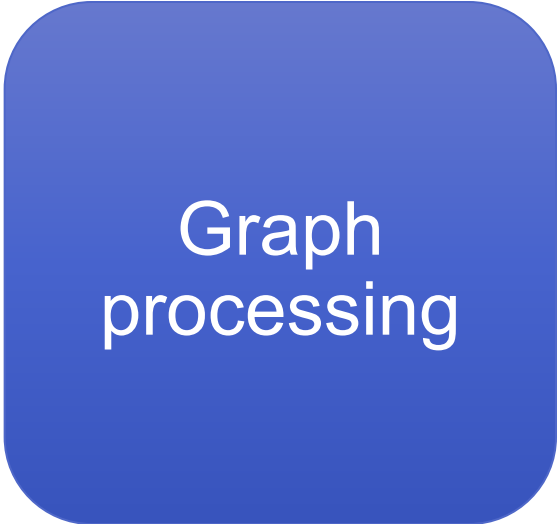
Presented by Braden Ehrat

A red rounded square with a gradient from light red at the top to dark red at the bottom.

Batch  
processing

A teal rounded square with a gradient from light teal at the top to dark teal at the bottom.

Stream  
processing

A blue rounded square with a gradient from light blue at the top to dark blue at the bottom.

Graph  
processing

## Batch processing

- Hadoop
- Dryad

## Stream processing

- Storm
- MillWheel

## Graph processing

- GraphLab
- PowerGraph

Batch  
processing

Stream  
processing

Graph  
processing

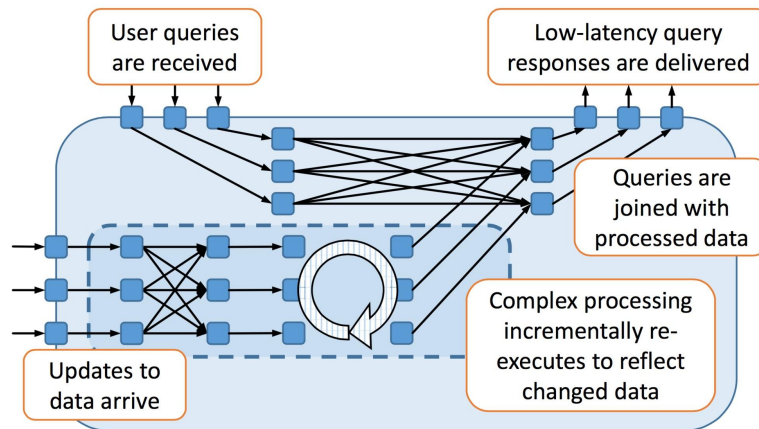
Timely dataflow with Naiad

```
graph TD; A[Batch processing] --> D[Timely dataflow with Naiad]; B[Stream processing] --> D; C[Graph processing] --> D;
```

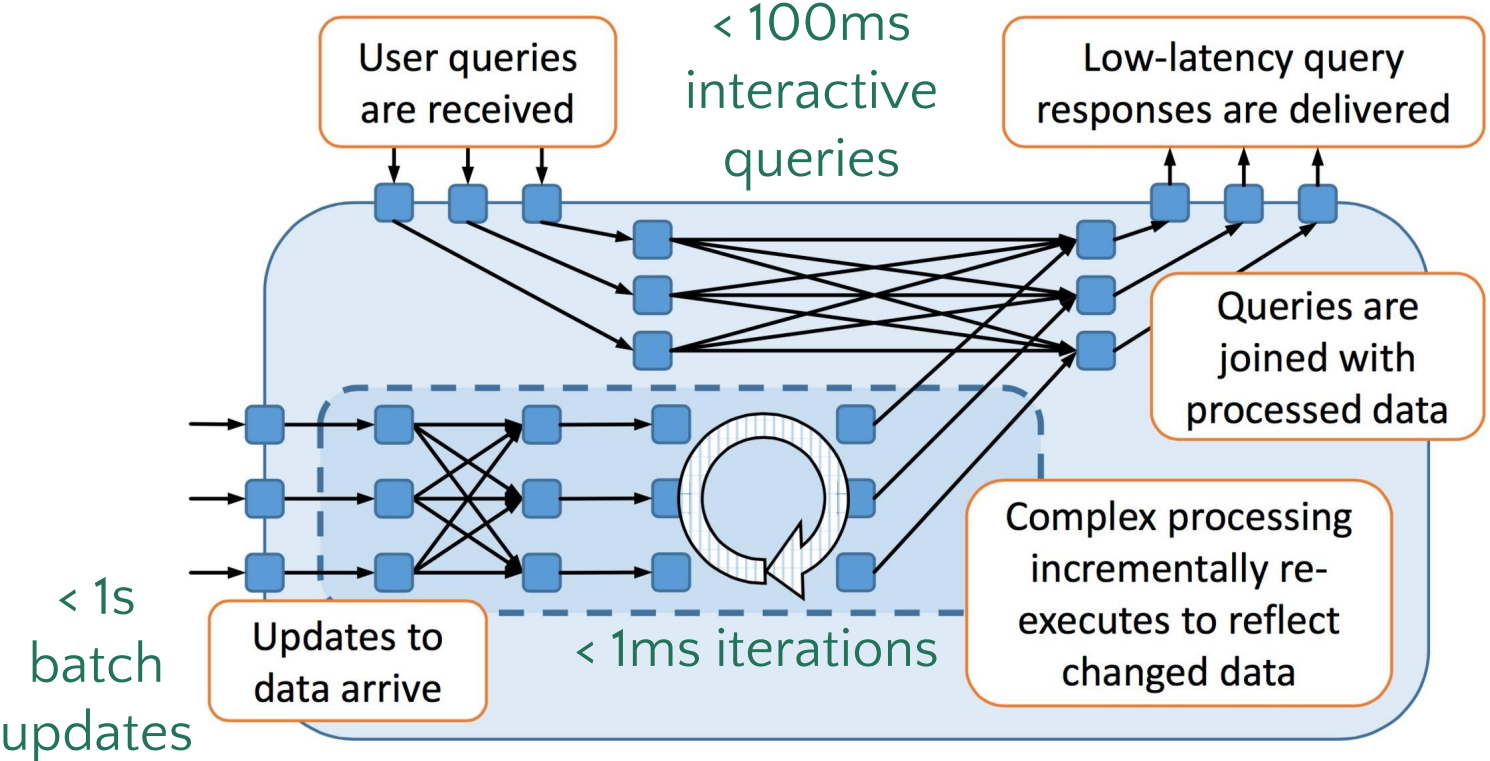
# Timely dataflow

A new computational model for stream processing

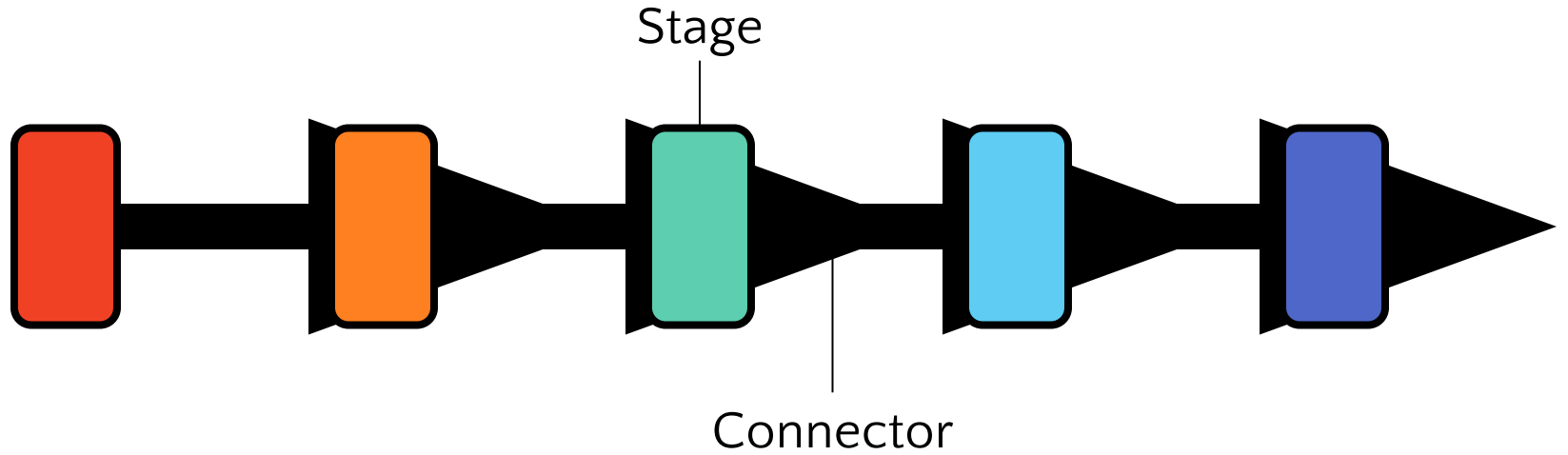
- Supports feedback loops
- Stateful vertices with arbitrary data
- Notifications for end of epoch



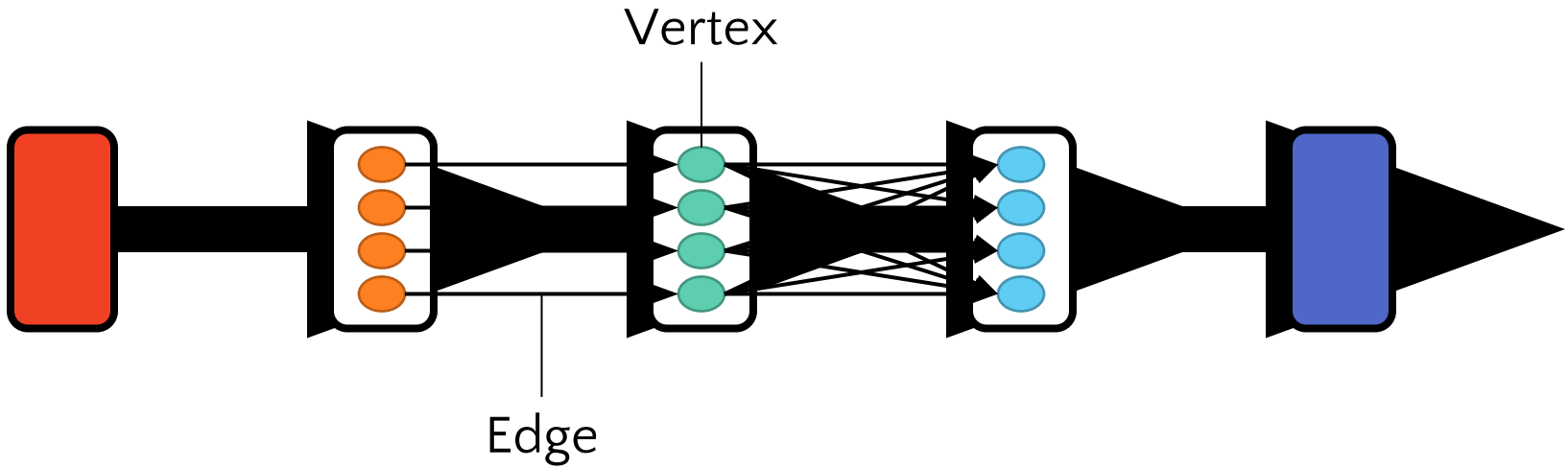
# Low-latency, incremental stream processing



# Dataflow

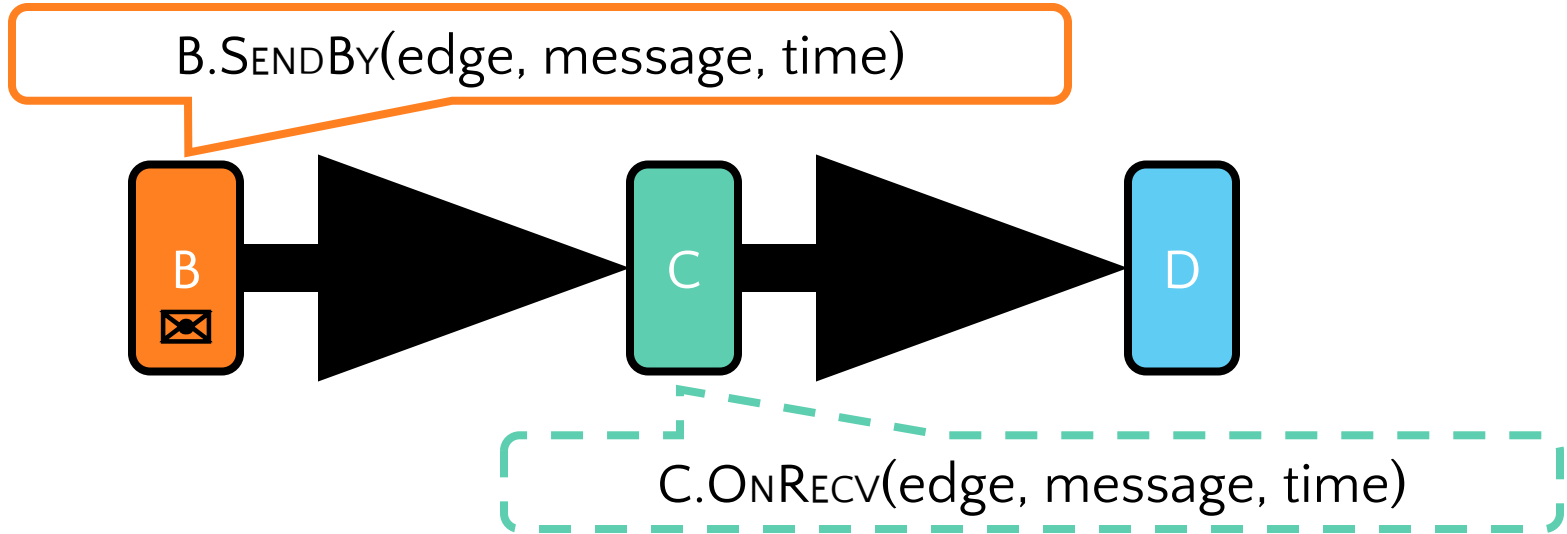


# Dataflow: parallelism



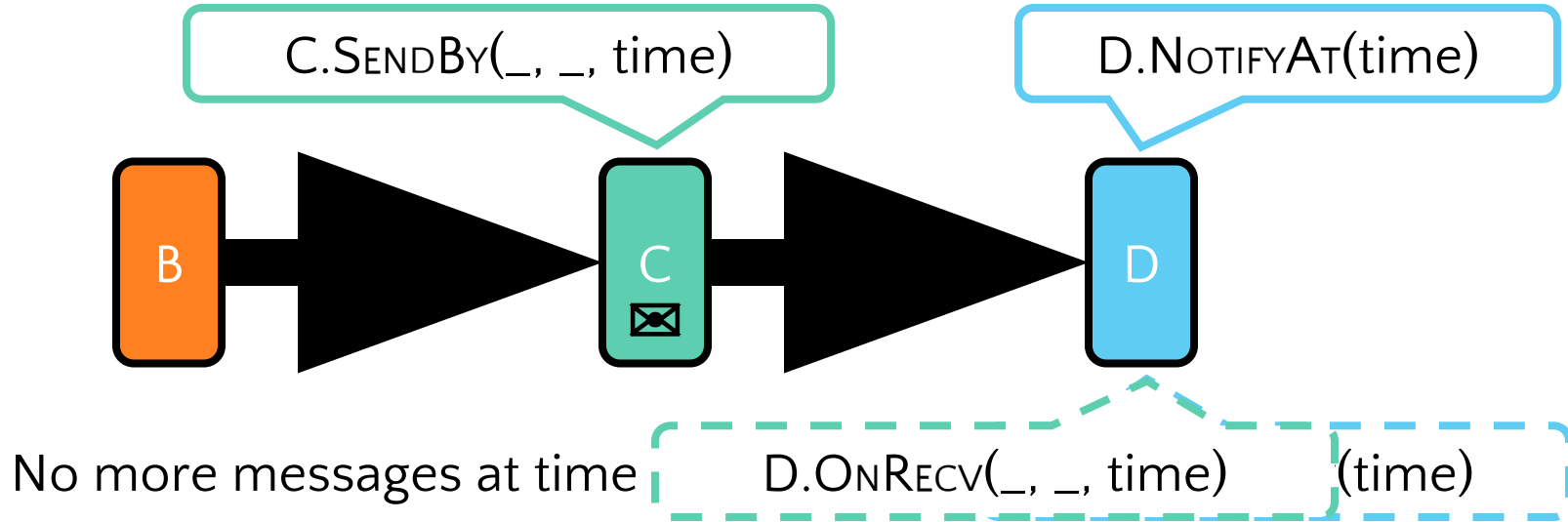


# Messages



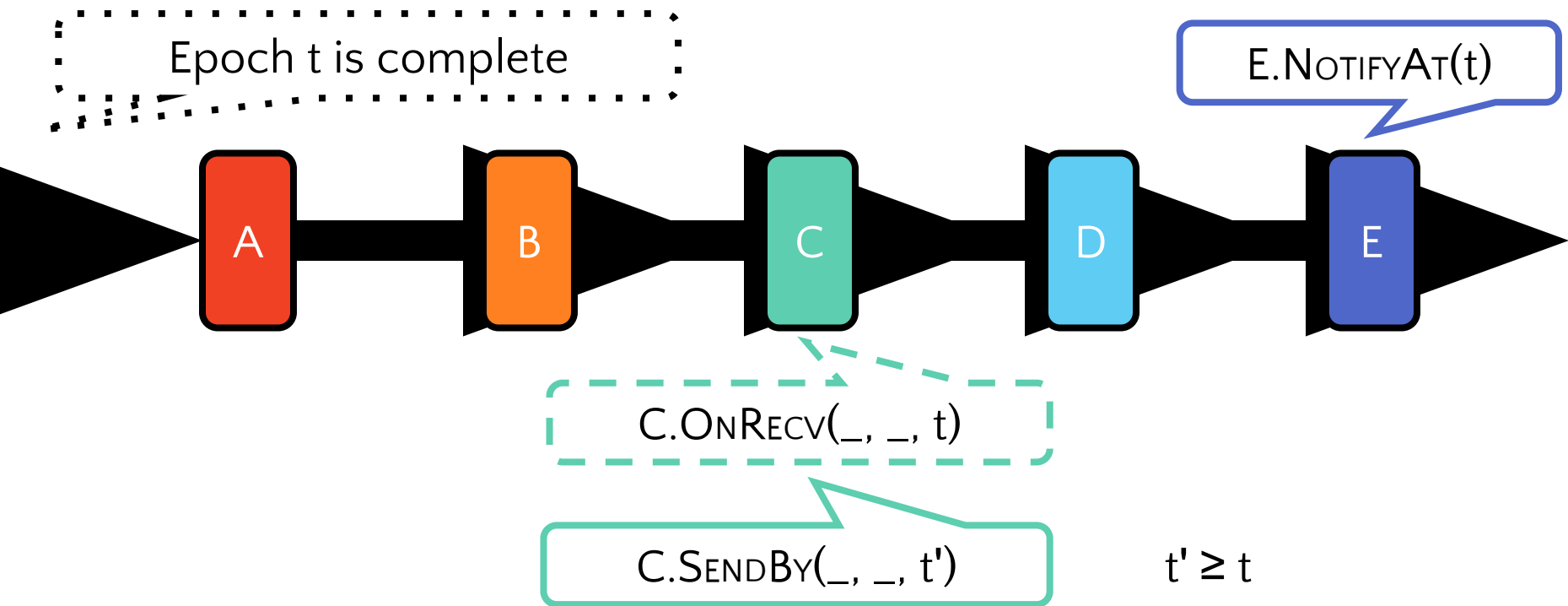
Messages are delivered asynchronously

# Notifications

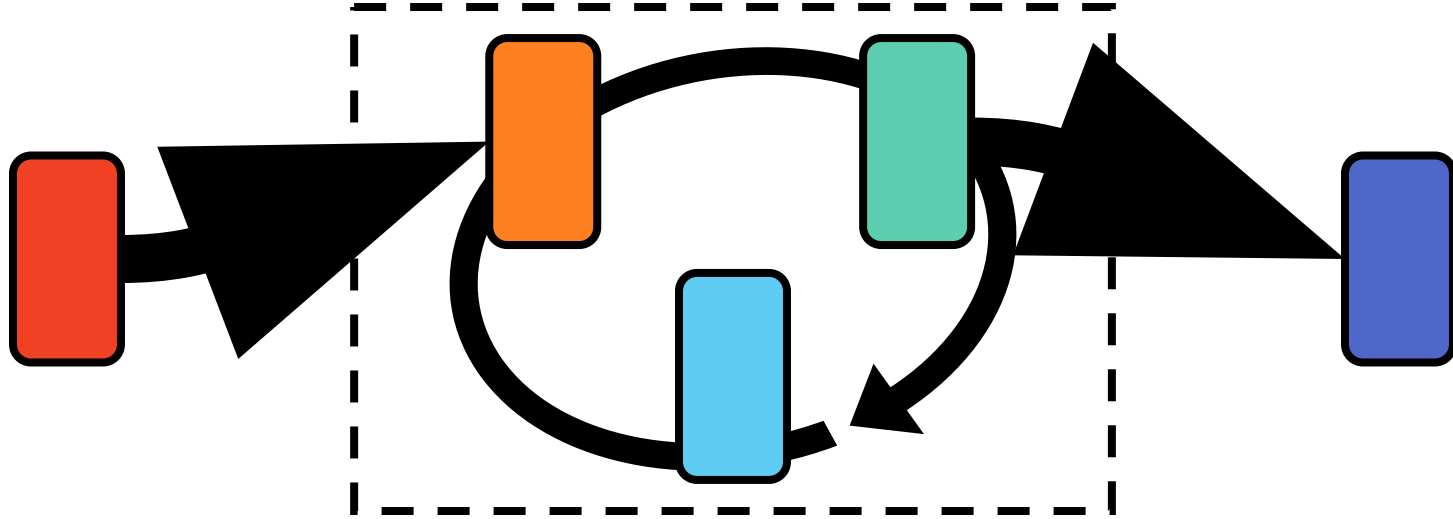


Notifications support batching

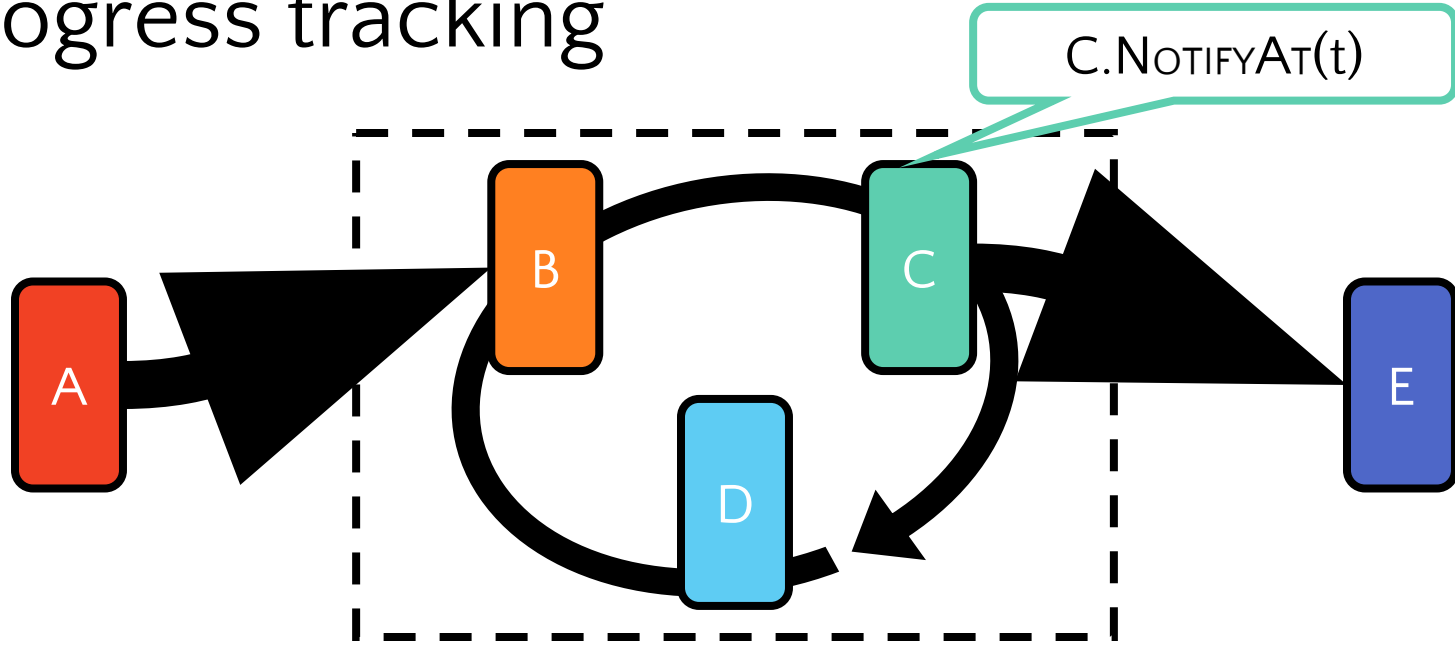
# Progress tracking



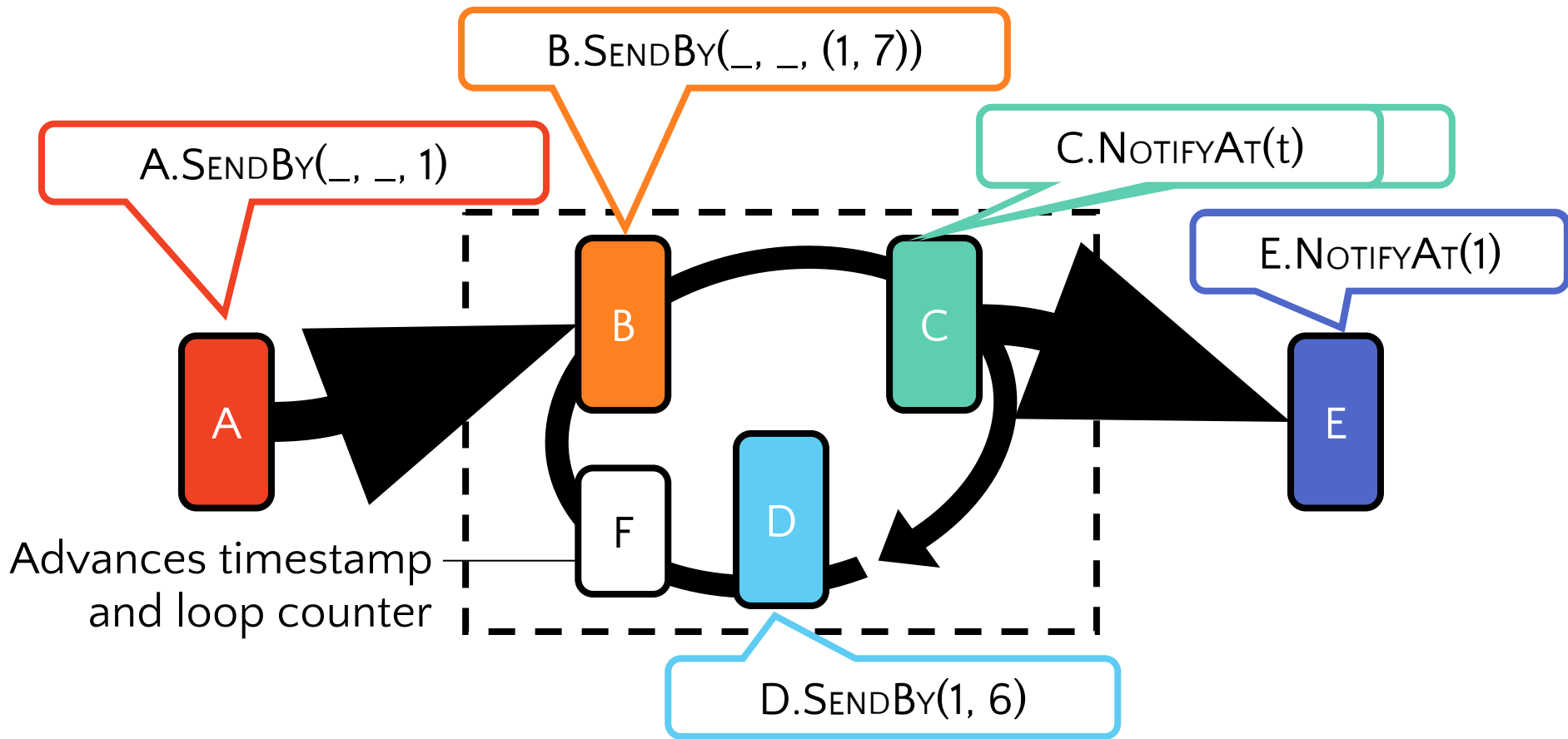
# Dataflow: iteration



# Progress tracking



Problem: C depends on its own output



Solution: structured timestamps in loops

# Simple API

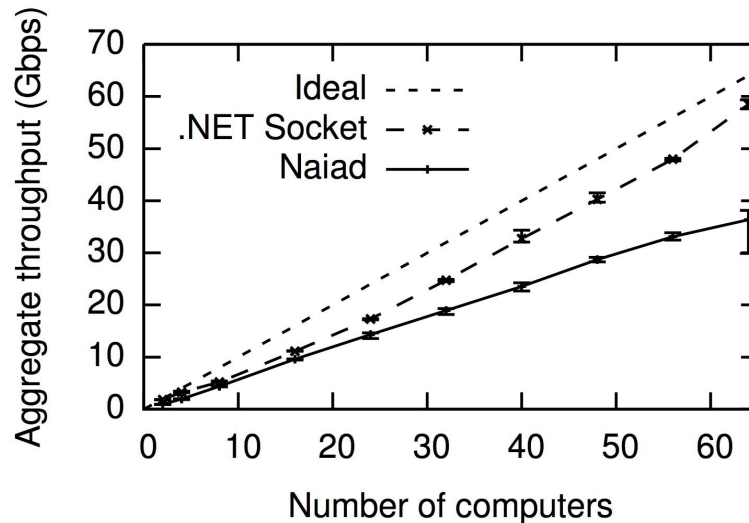
```
class DistinctCount<S,T> : Vertex<T> {
    Dictionary<T, Dictionary<S,int>> counts;
    void OnRecv(Edge e, S msg, T time) {
        if (!counts.ContainsKey(time)) {
            counts[time] = new Dictionary<S,int>();
            this.NotifyAt(time);
        }
        if (!counts[time].ContainsKey(msg))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

# Evaluation

All-to-all exchange  
throughput

Naiad exchanges 8-byte  
records between all  
processes

Shows low, linear overhead



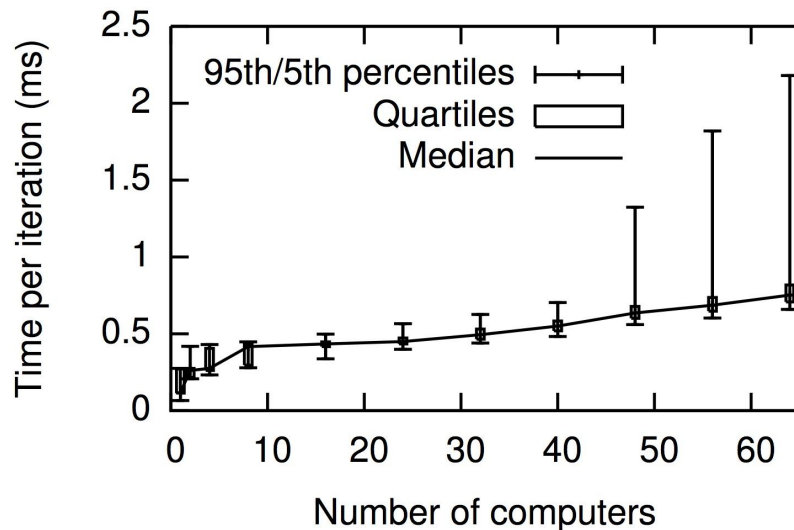


# Global barrier (Iteration) latency

Evaluates time to achieve global coordination

No data was exchanged

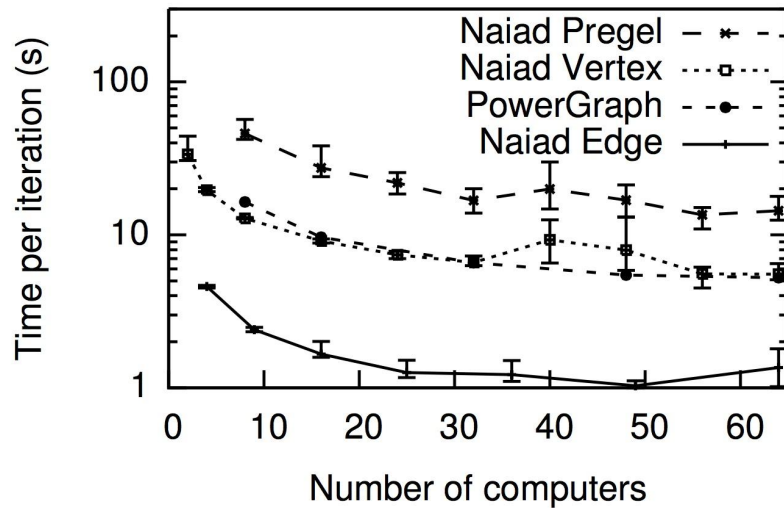
Effect of micro-stragglers seen at 50-60 nodes



# Real world calculations

## Twitter follower graph

- 42M nodes
- 1.5B Edges
- 6GB on disk

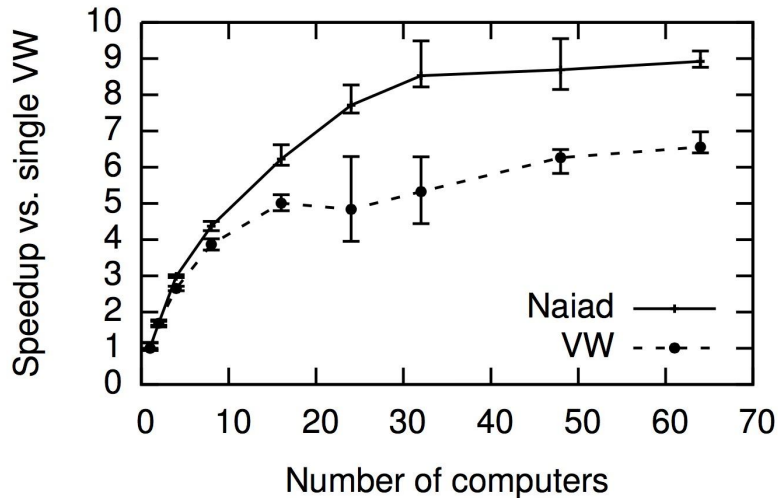


PageRank on Twitter followers

# Real world calculations

Vowpal Wabbit: Open-source distributed machine learning

Naiad is on-par with specialized implementations



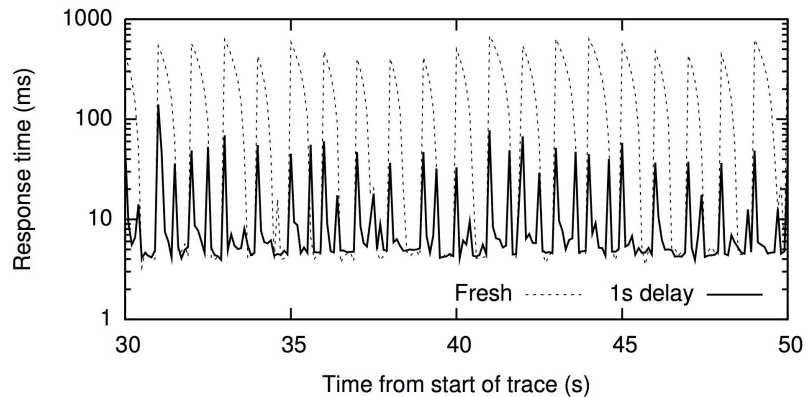
# Query Latency

Compute connected components and top tweets

- 32,000 tweets/s
- 10 queries/s

Fresh: queries delayed behind updates

1s delay: querying stale but consistent data



# Conclusions

Timely Dataflow in Naiad achieves:

- The performance of specialized frameworks
- Generic flexibility

Open source: <http://github.com/MicrosoftResearchSVC/naiad/>