What is ‘Timely’ Dataflow?! What is its significance?
DATAFLOW ?!
Dataflow?!
**Dataflow?!**

*Stages* makeup a directed graph where data flows along *connectors*.
**Dataflow?!**

*Stages* makeup a directed graph so it allows *iterations*
Dataflow?!

Physical **stages** is where systems exploit opportunities for **parallelism**
Also **stages** can run on different machines in the cluster.
Dataflow?! With **stages** parallelized as such. **Connectors** presenting the flow, have different types of **mapping**.
Dataflow

- Batch Processing e.g. MapReduce, Spark
- Asynchronous Processing e.g. Storm, MillWheel
- Variations for Graph Processing e.g. Pregel, GraphLab
Dataflow: Batch Processing

Stages require synchronization
Dataflow: Batch Processing

stages require synchronization
Dataflow: Batch Processing

**stages** require synchronization

1. **(1)**
2. **(2)**
3. **(3)**
4. **(4)**
5. **(5)**

**state**

input
**Dataflow: Batch Processing**

- **Stages** require synchronization

Diagram:
- Input
- Stages 1, 2, 3, 4, 5
- State (2)
Dataflow: Batch Processing

**stages** require synchronization

1. (1) input
2. (2)
3. (3)
4. (4)
5. (5)
Dataflow: Batch Processing

- Iterations make use of synchronization.
- The cost is latency.
**Dataflow: Asynchronous Processing**

**stages** do NOT require synchronization
all stages are active and output data after processing input data.
Dataflow: Asynchronous Processing

- Compared with batch:
  - latency is lower.
  - Aggregations are incremental and data changes over time.

- More efficient for distributed systems.
  - Stages do not need coordination.

- Correspondence between input & output is lost.
So, what is (Naiad) Timely dataflow ?!
Timely Dataflow?!

**stages** are asynchronous. They can synchronize if needed. Make use of **logical timestamps**.
Timely Dataflow

- Reconcile both models batch and async.
- Low-latency and high-throughput.
Where does Naiad fit?!
NAIAD?! 

- It is the prototype built by Microsoft Research underlying **Timely dataflow** Computational model.
- Iterative and incremental computations.
- The logical timestamps allow **coordination**.
- Provides efficiency, maintainability and simplicity.
Let's look at a computational example
NAIAD?! 

- It is the prototype built by Microsoft Research underlying **Timely dataflow** Computational model.
The Timely Dataflow Graph Structure
Graph Structure

(input, integer epoch)
Graph Structure

External Producer

(input, integer epoch)

(message, epoch)

External Consumers
Graph Structure
Graph Structure

- input comes in as (data, 0), (data, 1), (data, 2)
  - Within a loop, I adds a loop counter so it is (data, epoch, 0)
  - F in each iteration increments the loop counter (data, epoch, 1) etc.
  - E removes the loop counter and it is back to (data, epoch)
Programming Model
Using the timestamps
Programming Model

External Producer

External Consumers

Root Context

(input, epoch)

(input, timestamp)
**Programming Model**

![Diagram of a programming model](image-url)
Programming Model
Programming Model

Diagram showing a sequence of steps starting from an "External Producer" node, followed by nodes labeled "In", "A", "B", and "Out", leading to an "External Consumers" node. The diagram includes a "Root Context" and a "NotifyAt" node with a timestamp parameter.
Programming Model
Programming Model Summary

- SendBy(edge, message, timestamp)
- OnRecv(edge, message, timestamp)
- NotifyAt(timestamp)
- OnNotify(timestamp)
Programming Model In Practice
Notice

- Project was discontinued in 2014. Silicon Valley lab closed.

- The paper uses C#. The latest one is open sourced and is in Rust.
**Word Count Example**

Class $V<\text{Msg, Time}>$: $\text{Vertex<Time>}$ { ... }
WORD COUNT EXAMPLE

{ Dict<Time, Dict<Msg, int> > counts; ... }
Word Count Example (2 Different Implementations)

```c
{ void OnRecv (Edge e, Msg m, Time t) { ... } 
  void OnNotify (Time t) { ... } }
```
Writing Programs in General

- It is possible to write programs against the Timely Dataflow abstraction.

- It is possible to use libraries (MapReduce, Pregel, PowerGraph, LINQ etc.)

In General:
- Define Input, computational & Output vertices.
- Create a timely dataflow graph using the appropriate interface.
- Supply labeled data to input stages.
- Stages follow a push-based model.
Timely Guarantees
How is timely dataflow achieved
**How is timely dataflow achieved**

- **Key point:** timestamps at which future message can occur depends on: 1. Unprocessed events & 2. Graph Structure.
How is timely dataflow achieved

- **Pointstamp** of an event (timestamp, location: E or V)
  - SendBy -> Msg event of pointstamp \((t, e)\)
  - NotifyAt -> Notif event of pointstamp \((t, v)\)
How is timely dataflow achieved

- Pointstamp(t₁, l₁) could-result-in Pointstamp(t₂, l₂) If there is a path between l₁ and l₂ presented by f() i.e. f(t₁) ≤ t₂
How is timely dataflow achieved (Correctness Guarantees)

- Path Summary between A and C: “”
How is timely dataflow achieved (Correctness Guarantees)

- Path Summary between A and C: “add” or “add-increment(n)”
**Single-Threaded Implementation**

- Scheduler that needs to deliver events.
**Single-Threaded Implementation**

- Scheduler has active pointstamps <-> unprocessed events.
Single-Threaded Implementation

- Scheduler has active pointstamps <-> unprocessed events.
- Scheduler has two counts:
  - Occurrence count of not resolved event.
  - Precursor count of how many active pointstamps precede it in the could-result-in order.
Single-Threaded Implementation

- **Pointstamp**\((t, l)\) becomes **active**.
  
  Precursor count to number of existing active pointstamps that could result in it.
  Increment precursor count of any pointstamp it could-result-in.
  Becomes not active when occurrence is zero.
  When not active, decrement the precursor count for any pointstamp that it could-result-in.
The Distributed Environment
Distributed Implementation
Distributed Progress Tracking

- Initial protocol: same as single multi-threaded.
  - Broadcast occurrence count updates.

- Do not immediately update local occurrence count.
  - Broadcast progress updates to all workers including myself.
  - Broadcast from a worker to another delivered in a FIFO manner.

- Use of a projected timestamp.
- A technique to buffer and accumulate updates.
Micro-Stragglers

- Have a big effect on overall performance.
  - Packet Loss (Networking)
  - Contention on concurrent data
  - Garbage collection
Performance Evaluation
Performance Evaluation

- I invite you to read: “Scalability! BUT at what Cost”
Performance Evaluation

● Comparison with:
  ○ SQL Server Parallel Data Warehouse (RDBMS)
  ○ Scalable HyperLink Store (distributed in-memory DB for storing large portions of the web graph)
  ○ DryadLINQ (data parallel computing using a declarative / high level programming language)

● Algos i.e. PageRank, SCC etc.
Conclusion: “Our prototype outperforms general-purpose batch processors and often outperforms state-of-the-art async systems which provide few semantic guarantees.”
Conclusion: “Our prototype outperforms general-purpose batch processors and often outperforms state-of-the-art async systems which provide few semantic guarantees.”
Streaming Systems
as of today
Streaming Systems

- Systems that have unbounded data in mind.
- They are a superset of batch processing systems.
Streaming Systems

Reference: Fig-1: Example of time domain mapping. Streaming 101
Streaming Systems

Design Questions:

● **What** results are calculated?
The types of *transformations* within the pipeline.

● **Where** in event time are results calculated?
The use of *event-time windowing* within the pipeline.

● **When** in processing time are results materialized? The use of watermarks and triggers.

● **How** do refinements of results relate?
Discard or accumulate or accumulate and retract.
Thank you!
**Resources**

- Link to transcribed talk in pdf format.
- Timely Dataflow ([Rust Implementation](#))
- Frank blog posts:
  - Timely dataflow
  - Differential dataflow
- The world beyond batch: Streaming 101
- The world beyond batch: Streaming 102