Differential Dataflow

McSherry, Frank D., Murray, Derek G., Isaacs, Rebecca, Isard, Michael

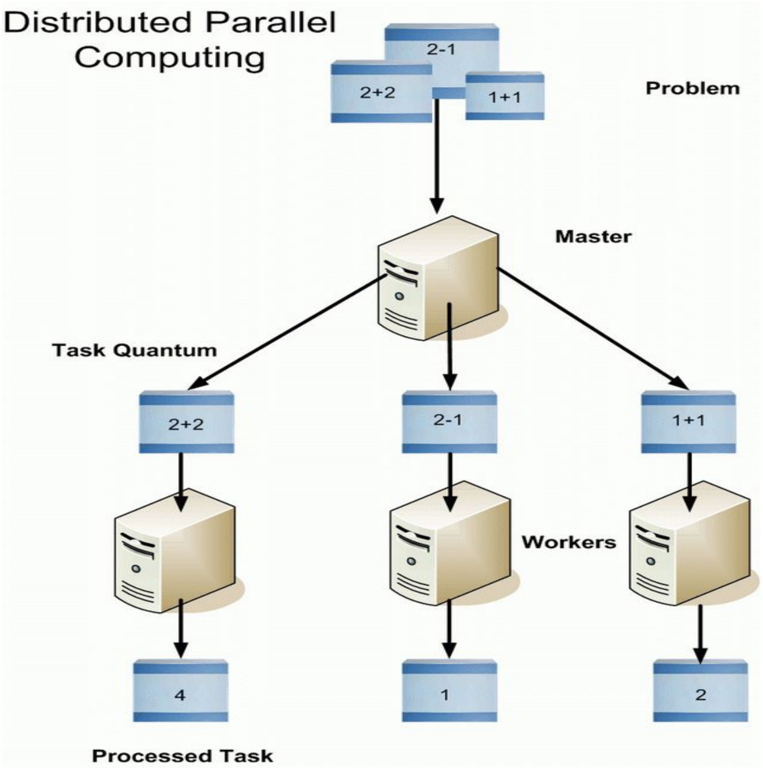
Chathura Kankanamge
08th November 2016

Outline

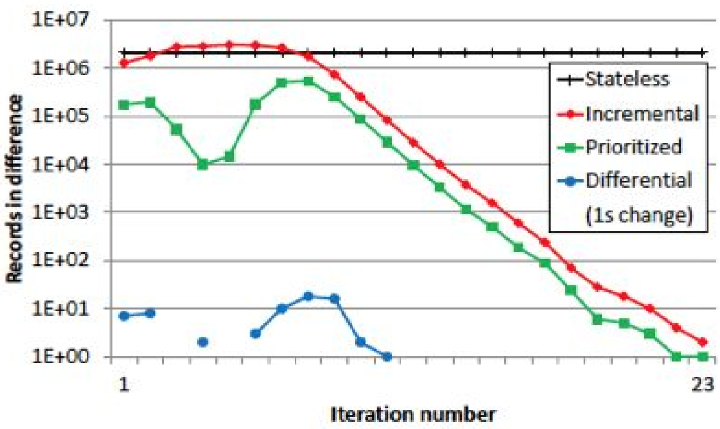
- Motivation for Differential Dataflow
- Key Concepts
- Differential Dataflow in practice
- Discussion

Motivation

Traditional data parallel processing



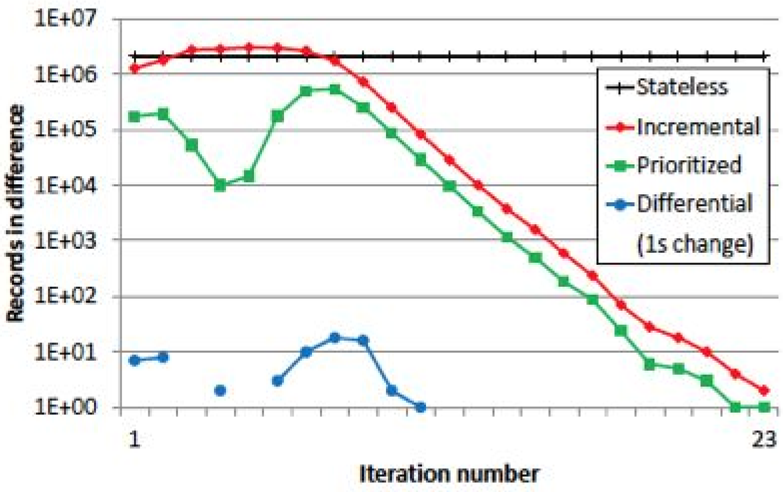
- Take input data in batches.
- Process and output.
- Highly evolved - Hadoop, Spark.
- Mostly stateless.



Interactive - Twitter Mention Graph

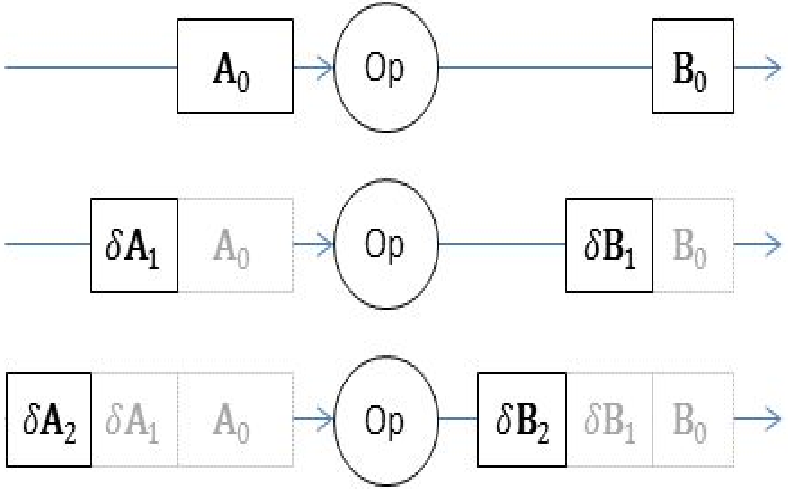
- Used to find trending #hashtags.
- Billions of vertices and edges.
- Millions of updates per second (storm).
- Needs low latency of streaming and throughput of spark.
- Similar issue with interactive analytics

Loop Processing



- Some algorithms require iterations
  - Pagerank
  - Connected components
- Usually requires transferring entire state between iterations
- Spark, Hadoop etc execution times ~ stateless

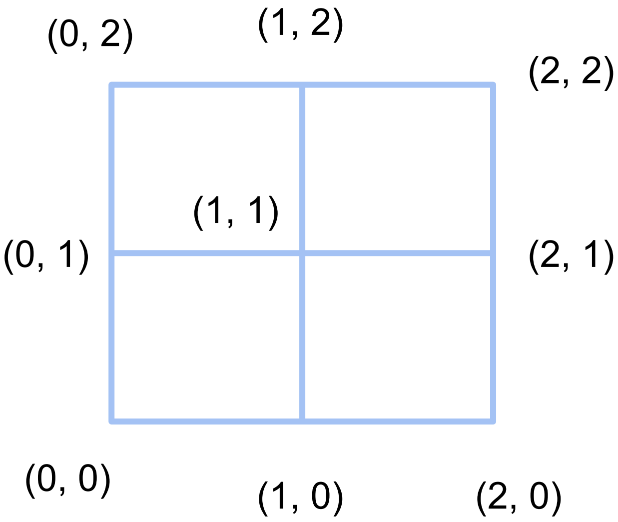
Incremental Dataflow



- Stateful.
- Get the differences of collections.
- Only calculate changes.
- Example
  - Wordcount in Hadoop Online.
- Can deal with changes due to,
  - Loops
  - New Data
- But NOT both!!

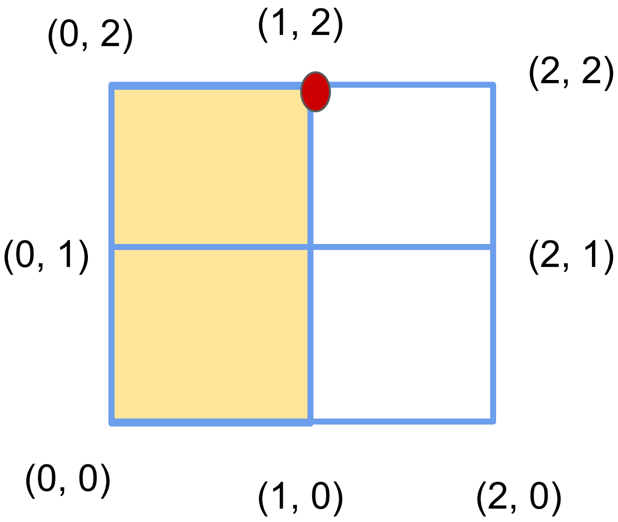
Concepts

Total vs Partial Ordering



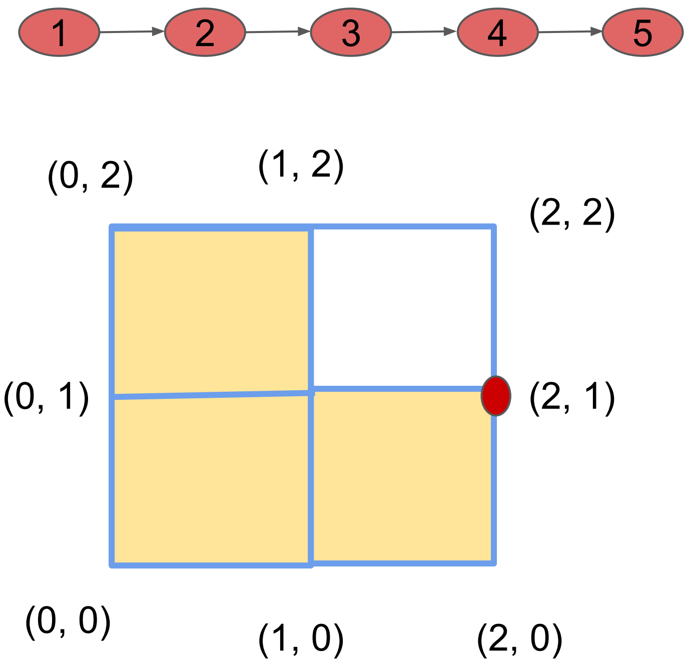
- Traditional dataflow systems expect total ordering
    - Multiple variables are a problem
- A partial ordering uses a time vector for ordering
    - Deals well with multiple variables
- Partial because ordering by variable x gives only a partial ordering

Total vs Partial Ordering



- Traditional dataflow systems expect total ordering
  - Multiple variables are a problem
- A partial ordering uses a time vector for ordering
  - Deals well with multiple variables
- Partial because ordering by variable x gives only a partial ordering for x

Total vs Partial Ordering



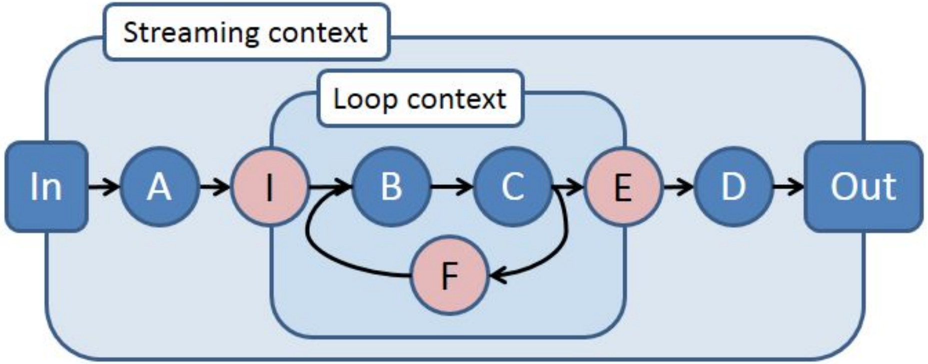
- Traditional dataflow systems expect total ordering
  - Multiple variables are a problem
- A partial ordering uses a time vector for ordering
  - Deals well with multiple variables
- Partial because ordering by variable x gives only a partial ordering

Differential Dataflow

- Computational Model
  - Defines how to process partially ordered data.
  - Defines state between iterations

- Goals
  - Do less calculation per change
  - Converge quicker per iteration

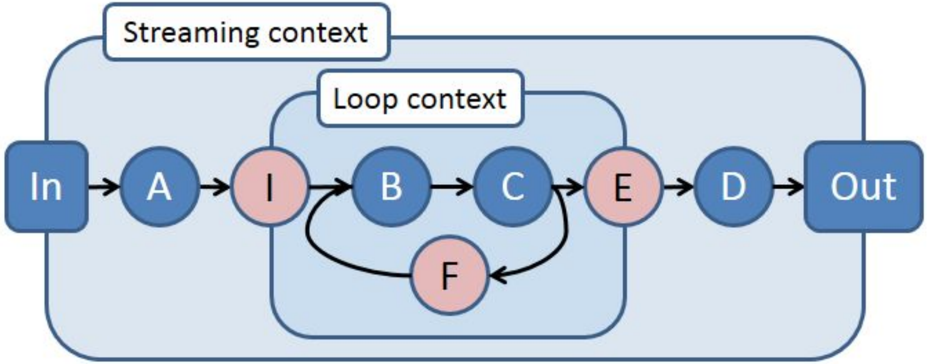
Timely Dataflow



- Performs Iterative Calculations
- Computational model with directed graph
- Vertices exchange messages
- Logical Timestamps for messages

$$: (\overbrace{e \in \mathbb{N}}^{\text{epoch}}, \overbrace{\langle c_1, \ldots, c_k \rangle \in \mathbb{N}^k}^{\text{loop counters}})$$

Timely Dataflow

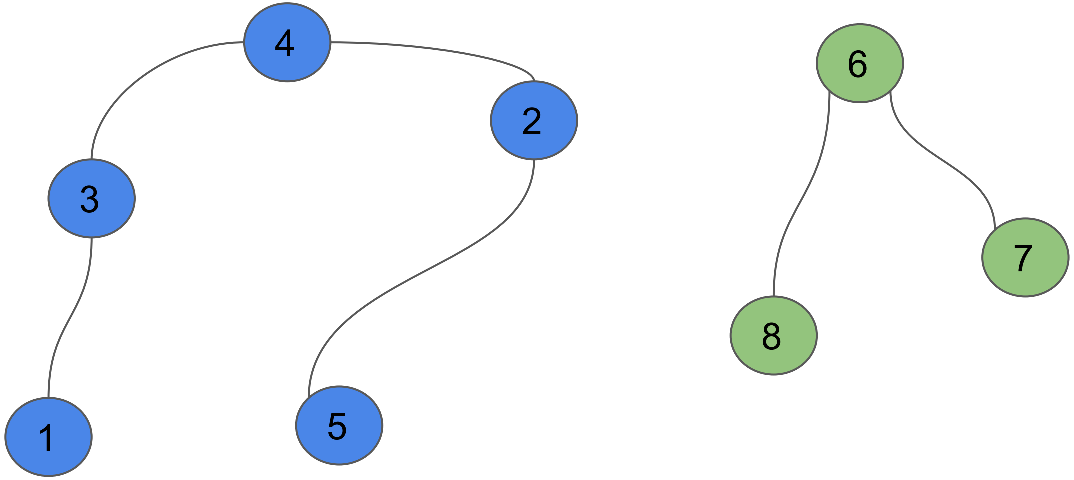


- Loops denoted by,
  - Ingress - adds a counter
  - Feedback - increments a counter
  - Egress - removes a counter
- Pointstamps - events at location and time

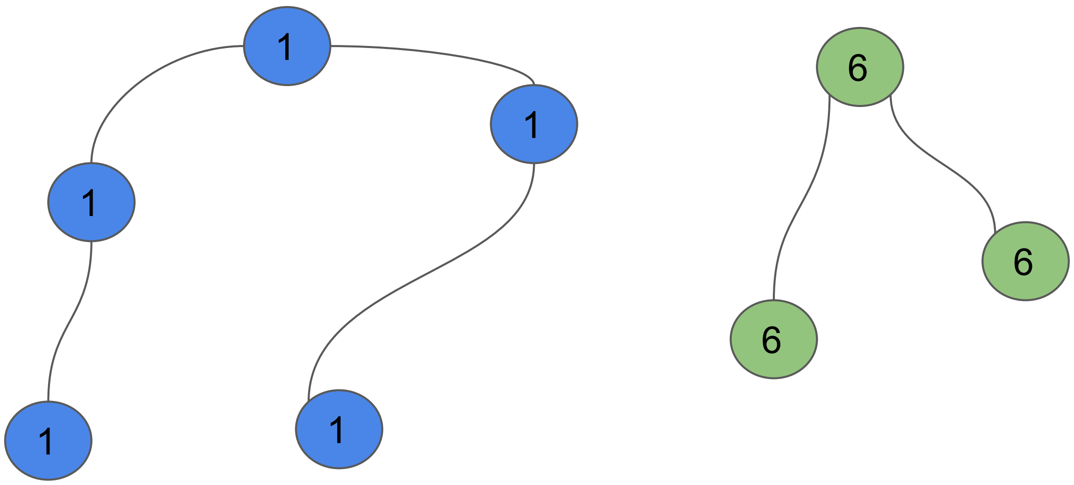
$$(t \in \text{Timestamp}, \overbrace{l \in \text{Edge} \cup \text{Vertex}}^{\text{location}})$$

Differential Dataflow in practise

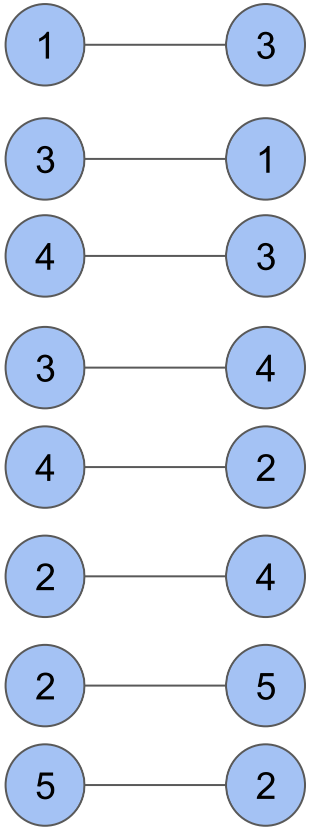
The Connected Graph Problem

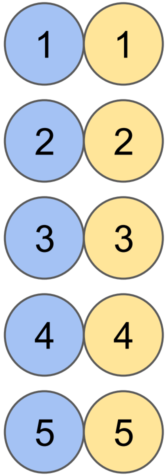


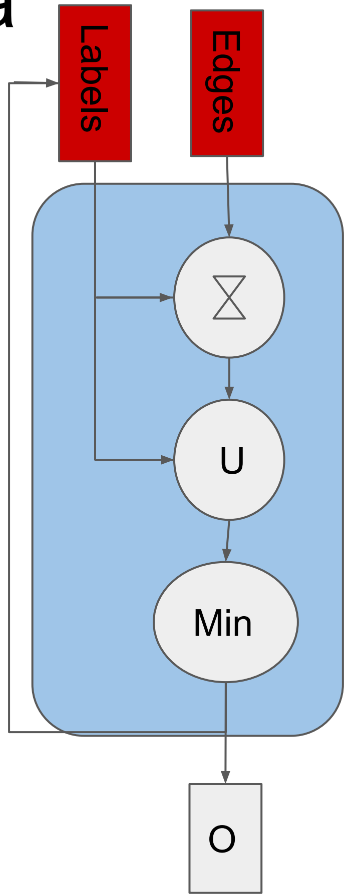
The Connected Graph Problem



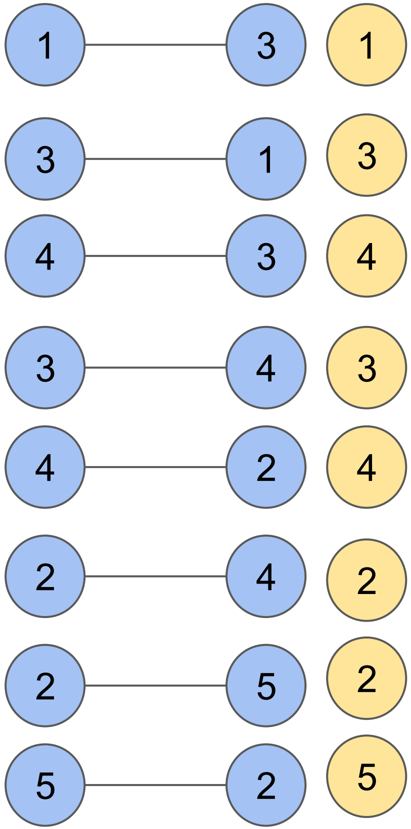
Connected Graph with Relational Algebra

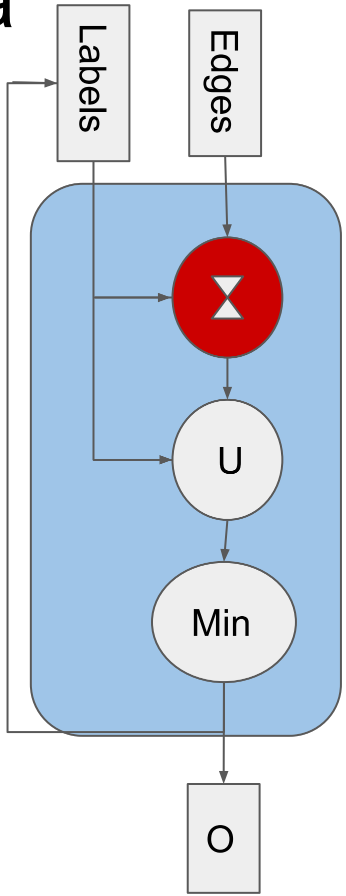




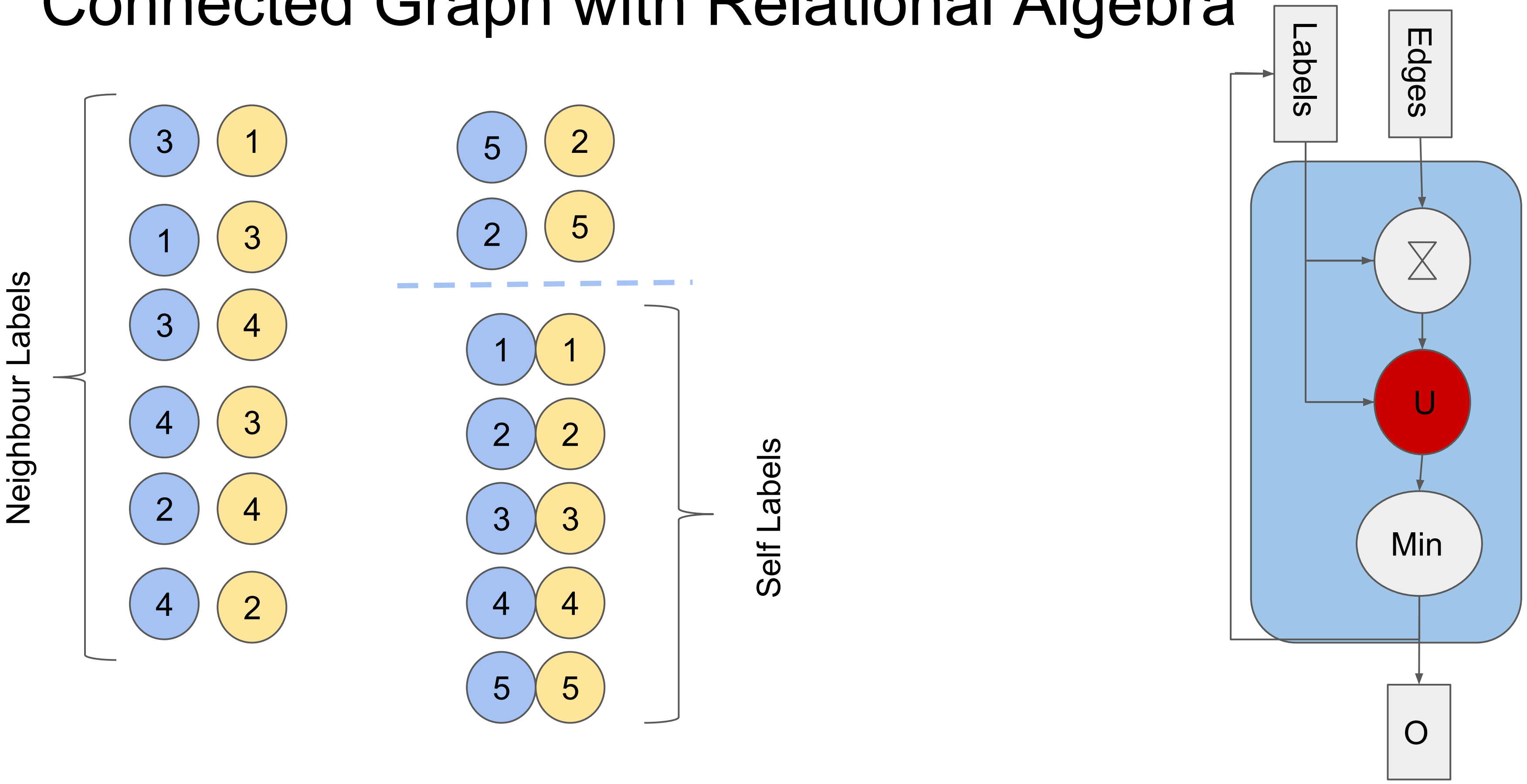


Connected Graph with Relational Algebra

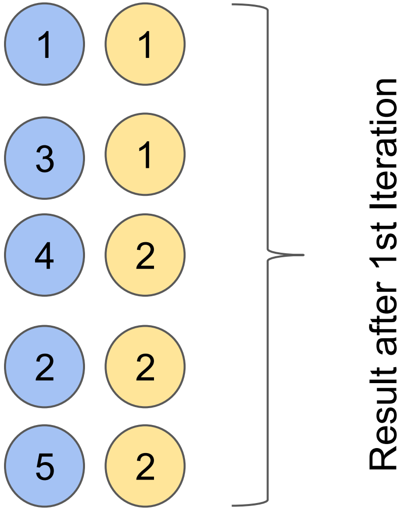


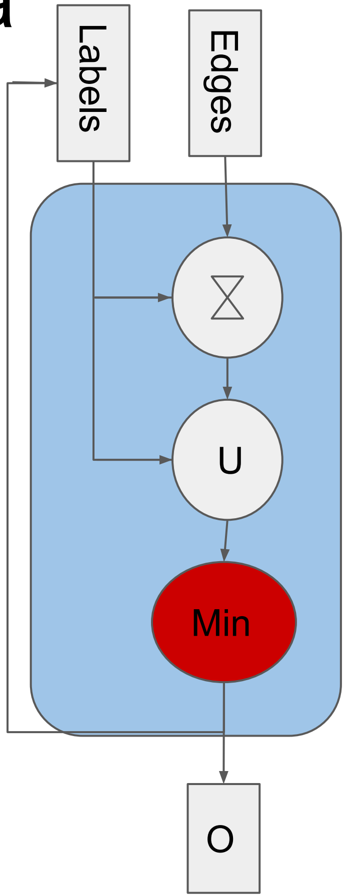


Connected Graph with Relational Algebra

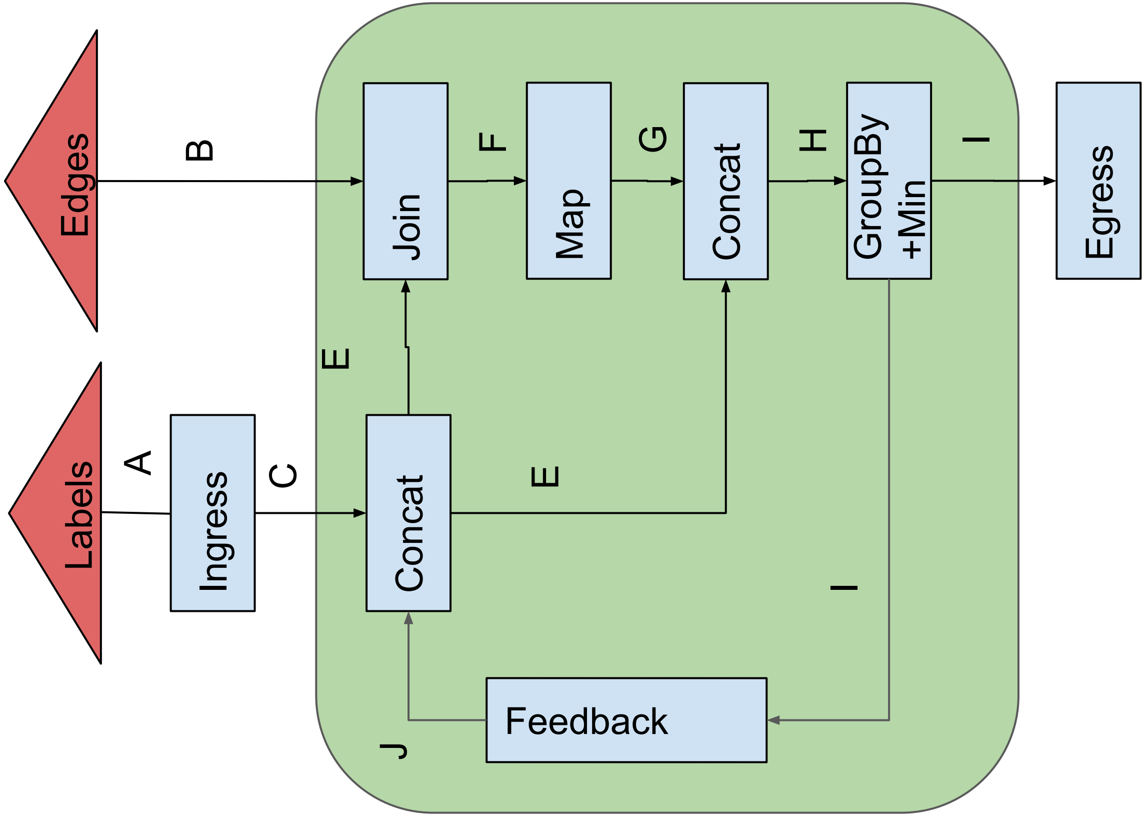


Connected Graph with Relational Algebra





Connected Graph in Timely



- Edges are available constantly
- Add counter at Ingress
- Remove Counter at egress
- Increment counter at feedback
- Map converts joined tuples into node/label tuples
- Concat performs the union

Maintaining State in Differential Dataflow

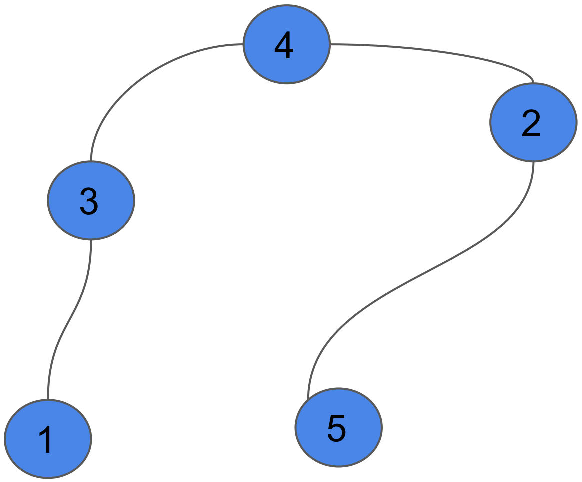
$$\delta\mathbf{B}_t = \mathrm{Op}\left(\sum_{s \leq t} \delta\mathbf{A}_s\right) - \sum_{s<t} \delta\mathbf{B}_s$$

Change in state at node *b* at *t*

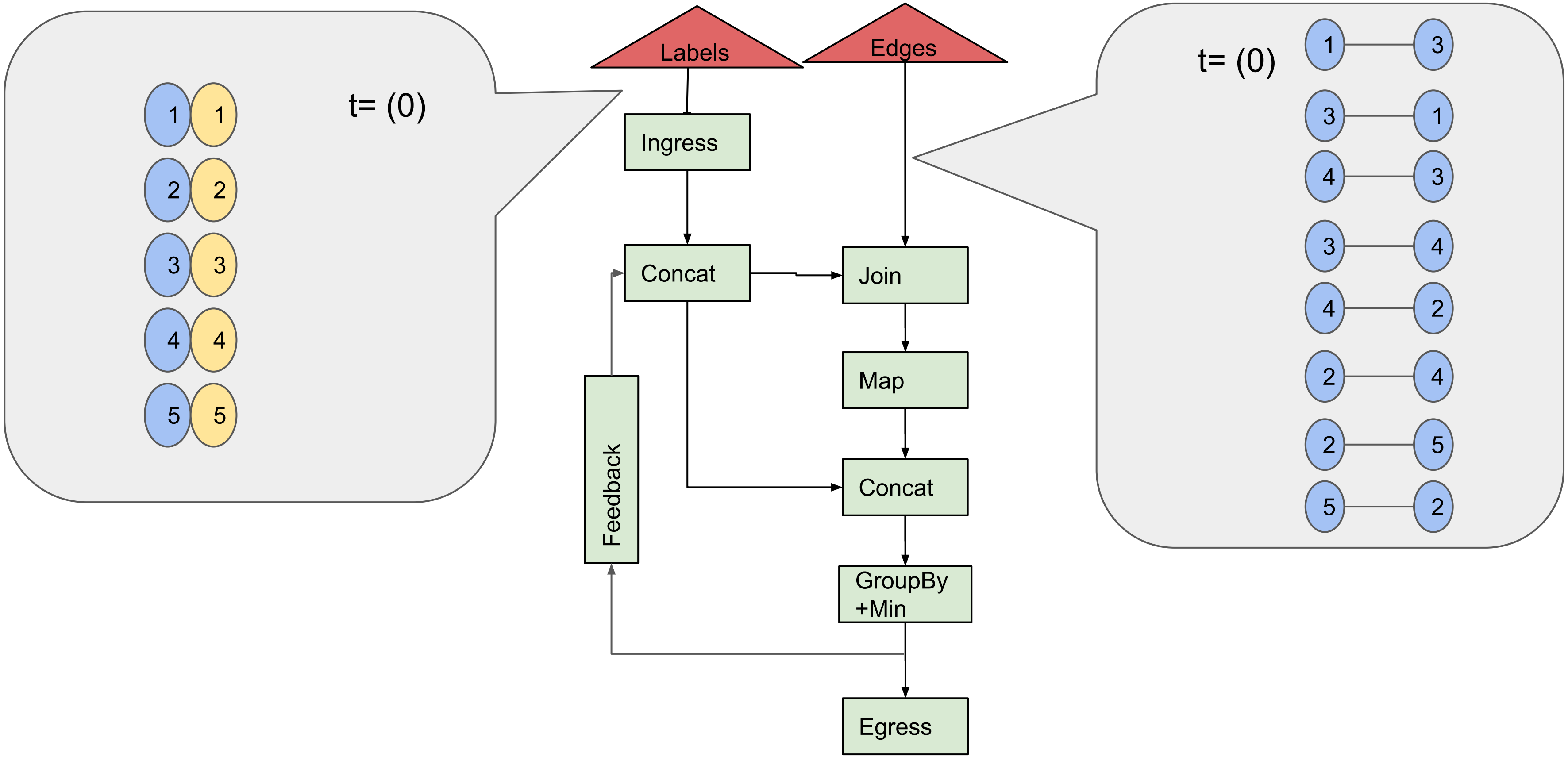
Cumulative state at *b* upto *t*

Sum of all states at *b* before *t*

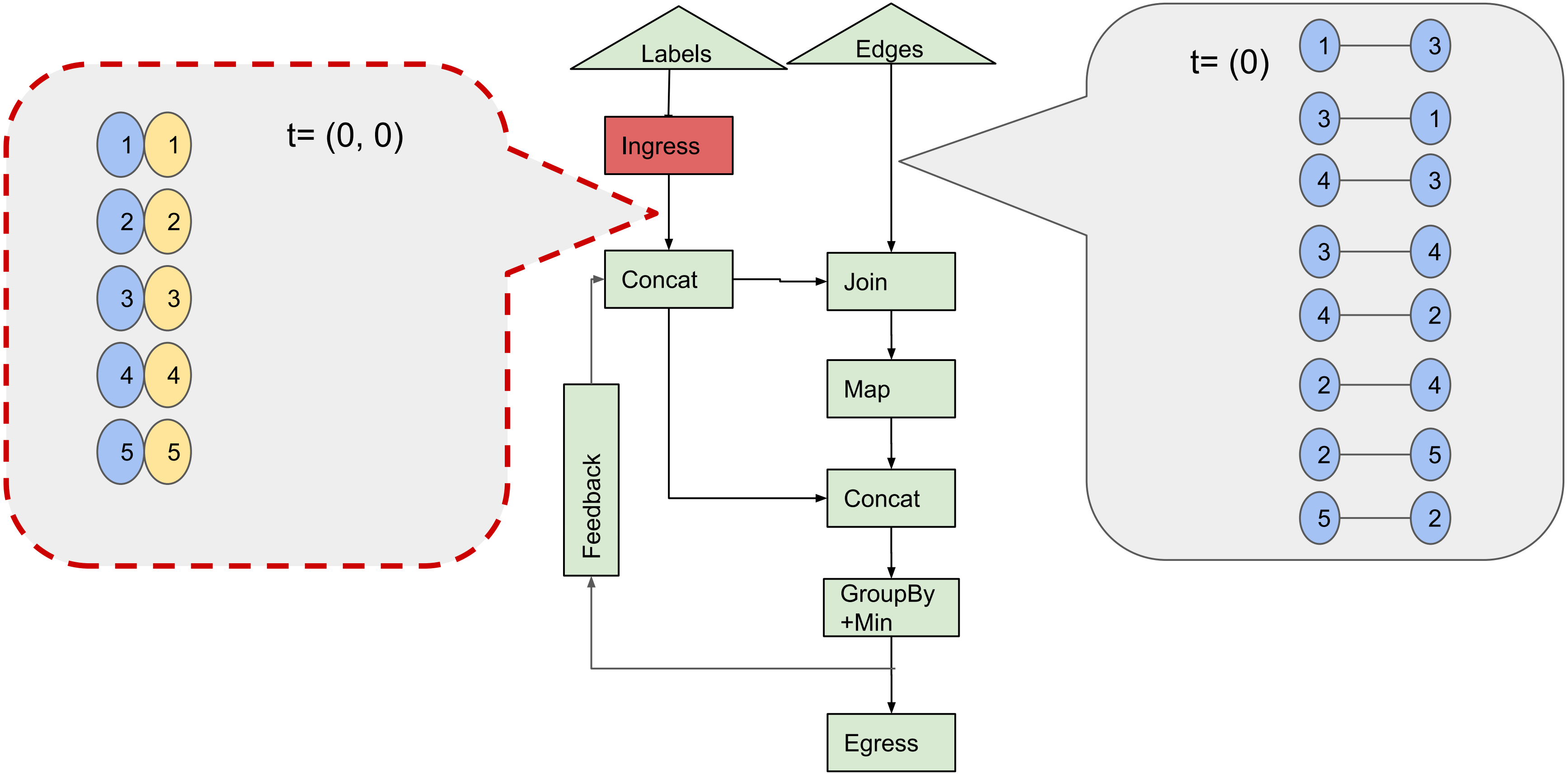
Connected Graph



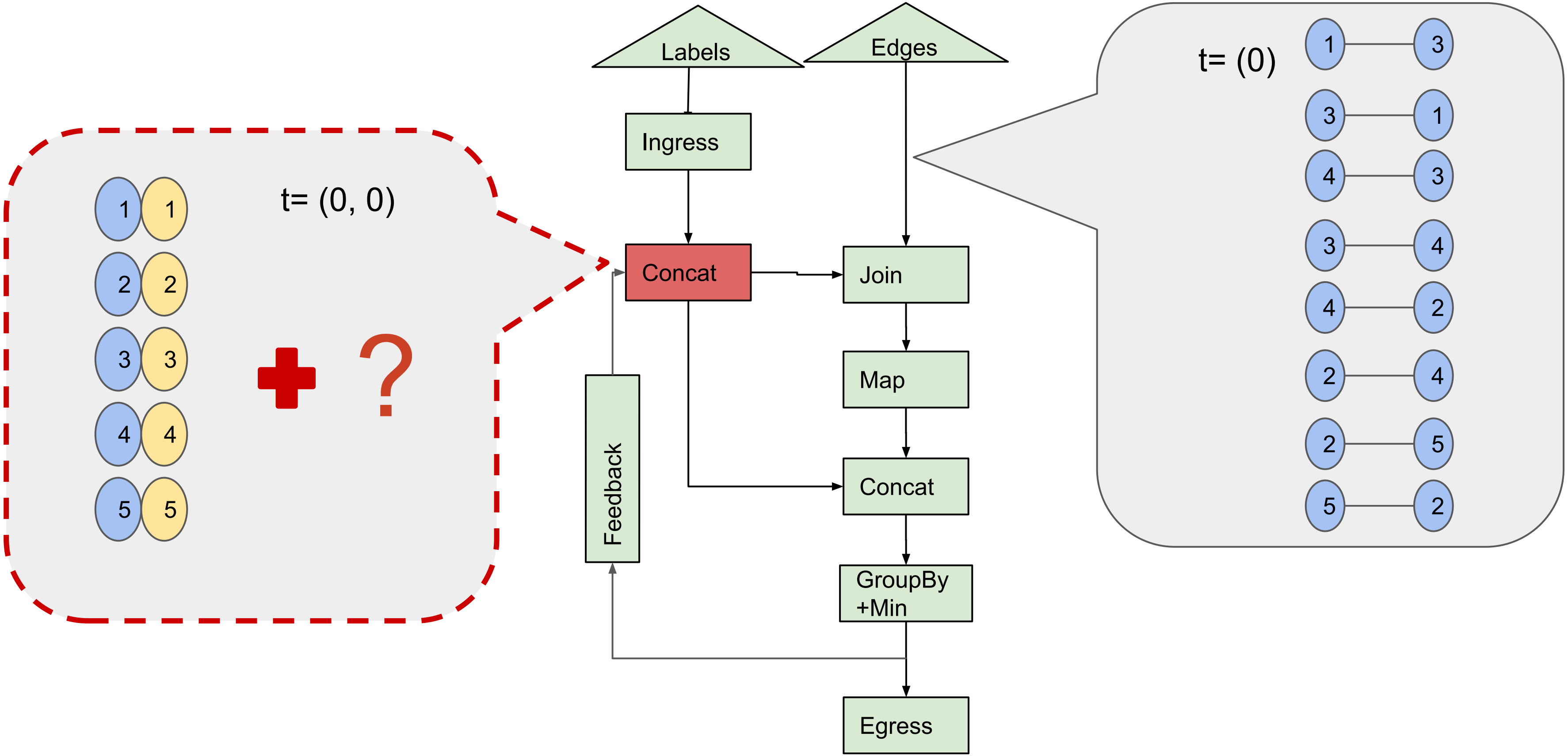
Connected Graph in Differential



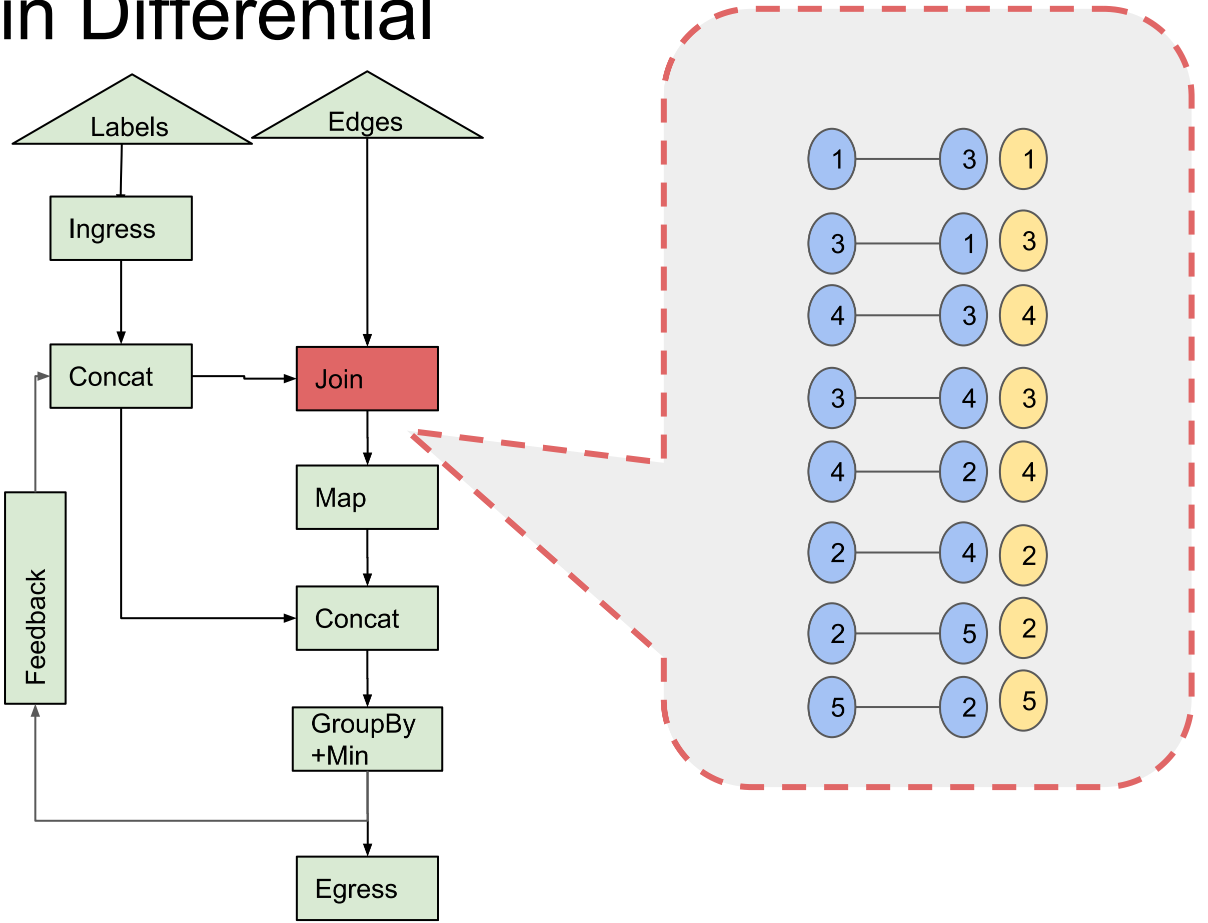
Connected Graph in Differential



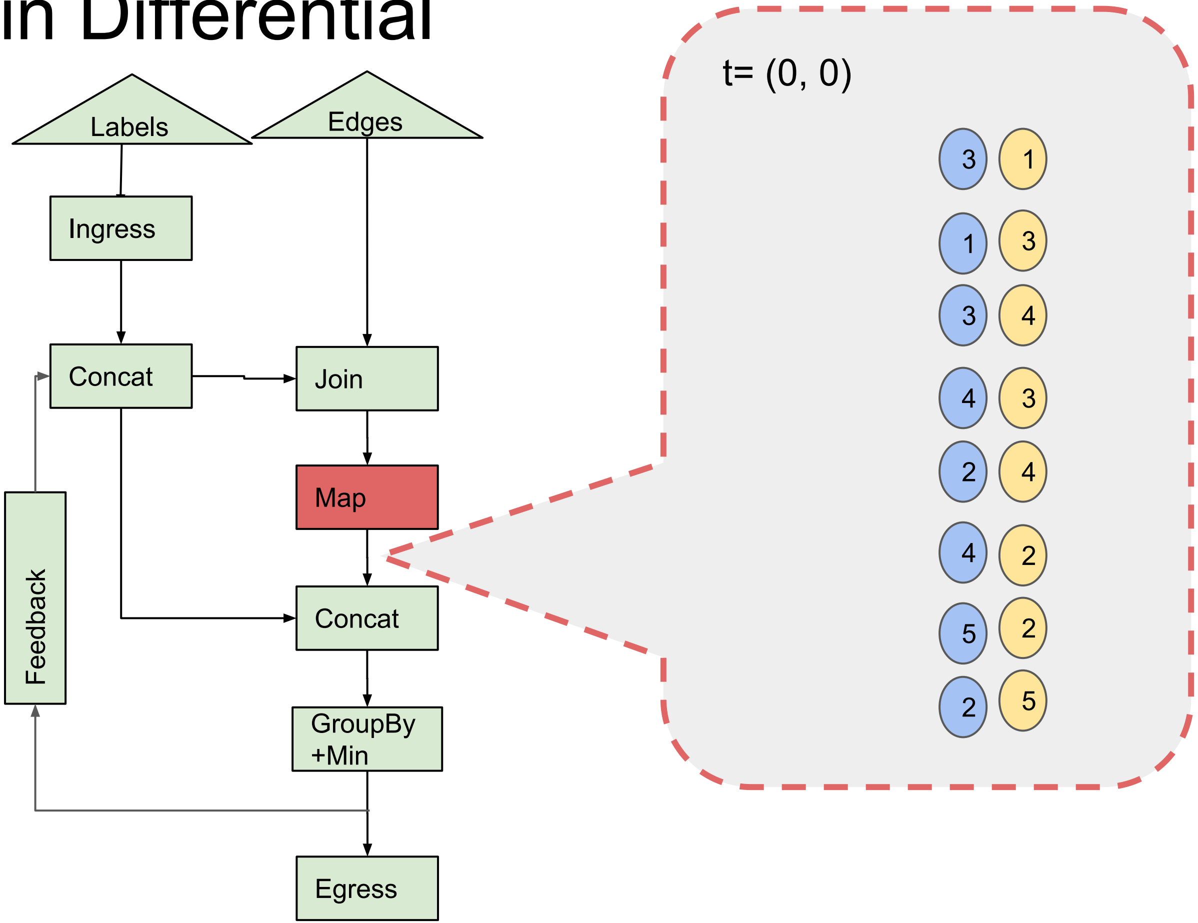
Connected Graph in Differential



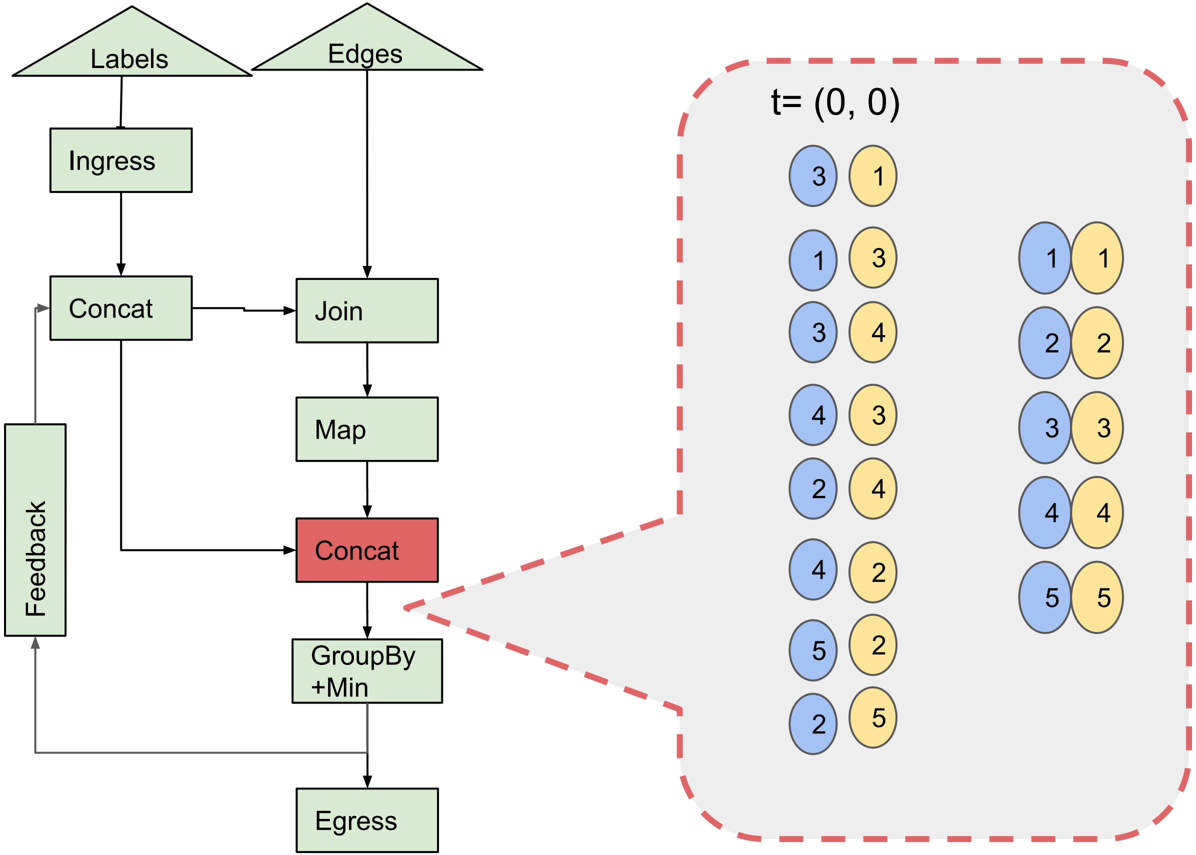
Connected Graph in Differential



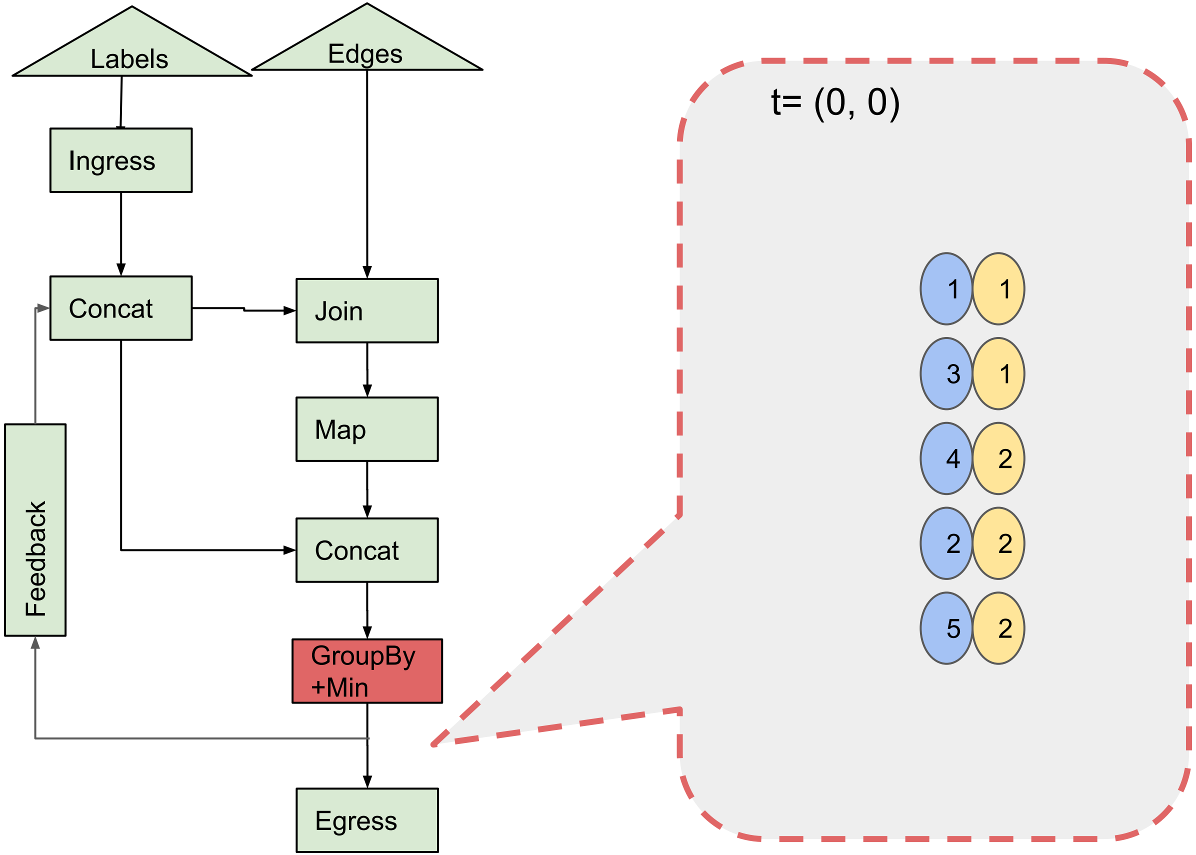
Connected Graph in Differential



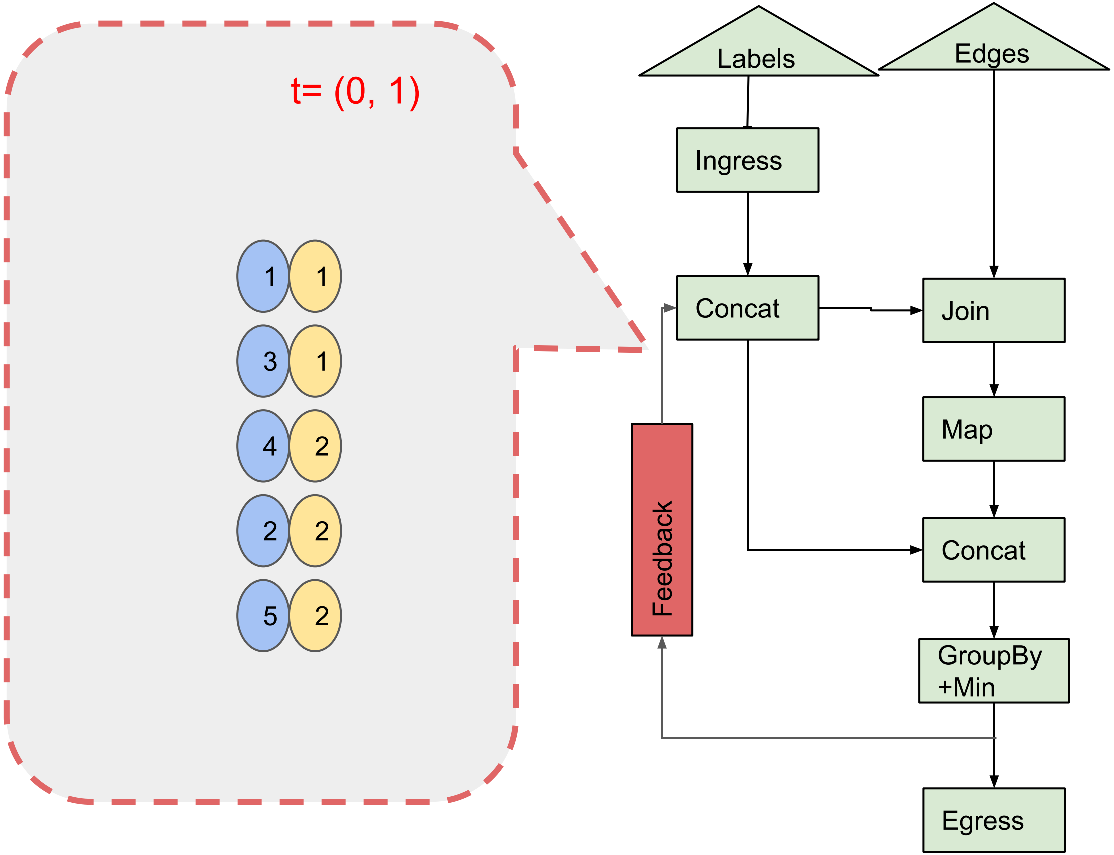
Connected Graph in Differential



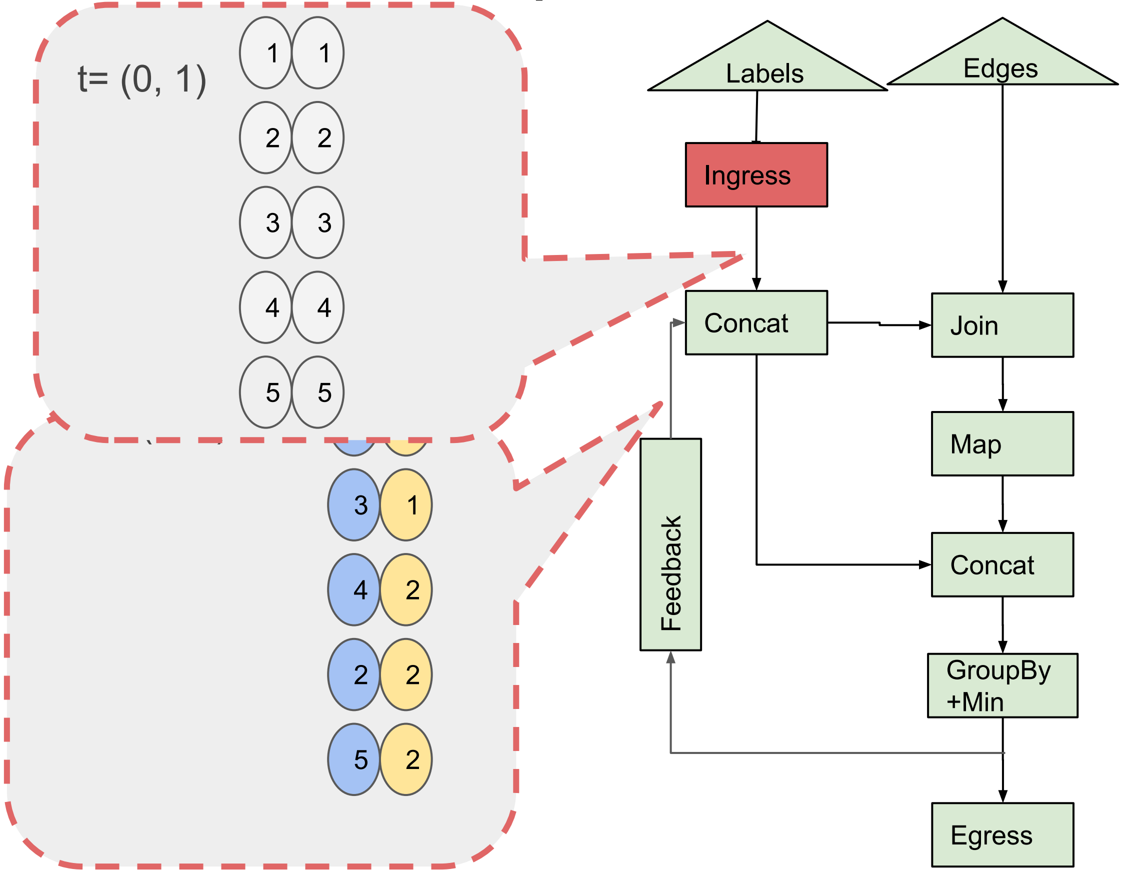
Connected Graph in Differential



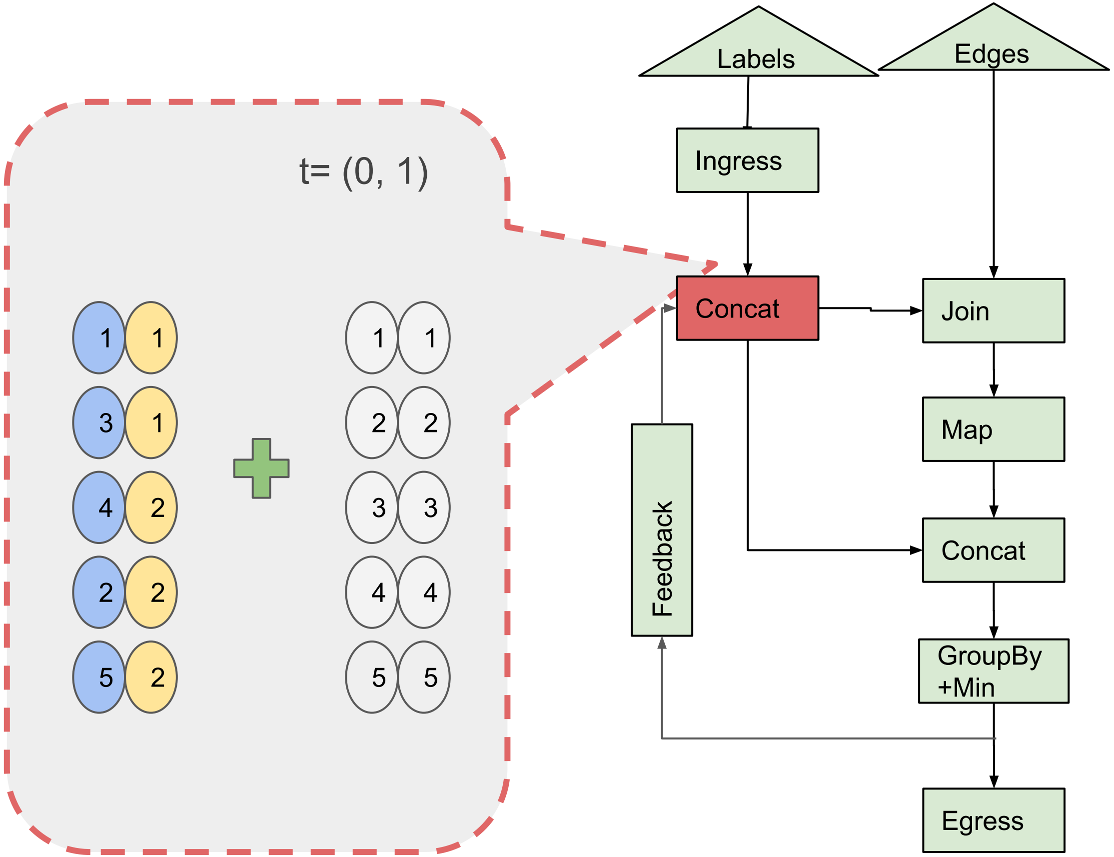
Connected Graph in Differential



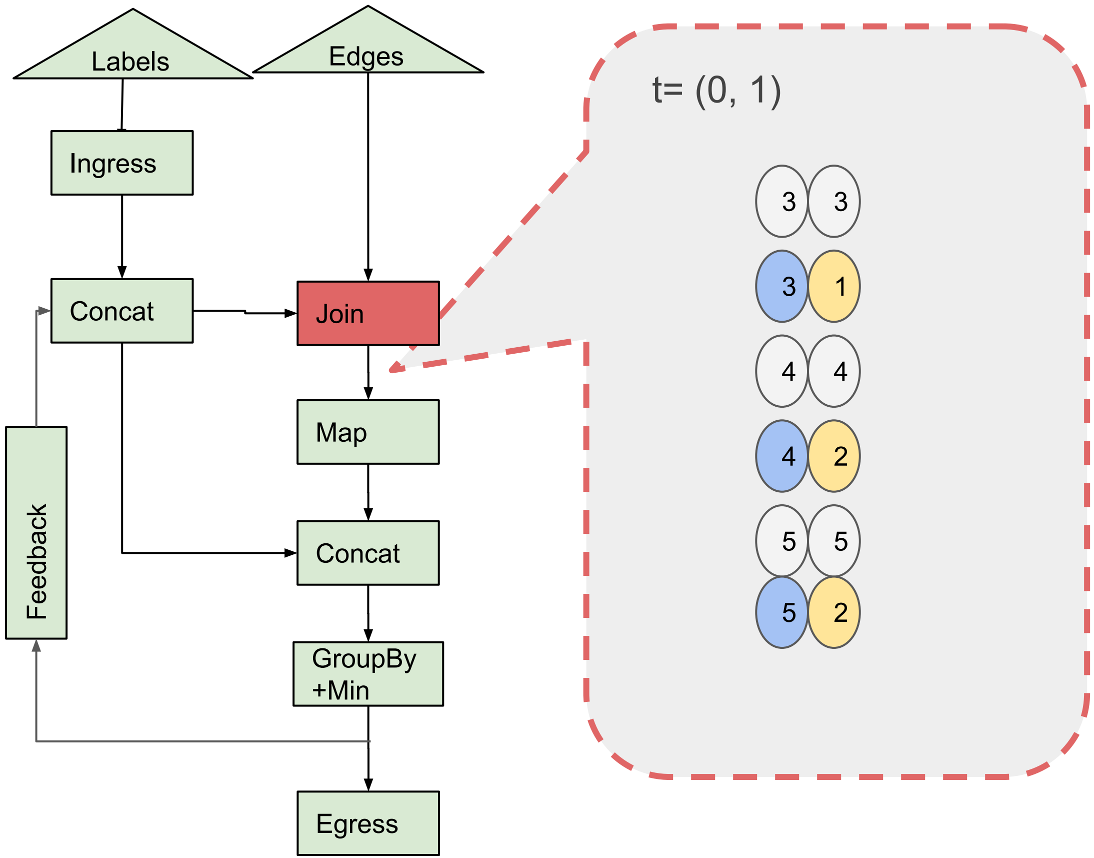
Connected Graph in Differential



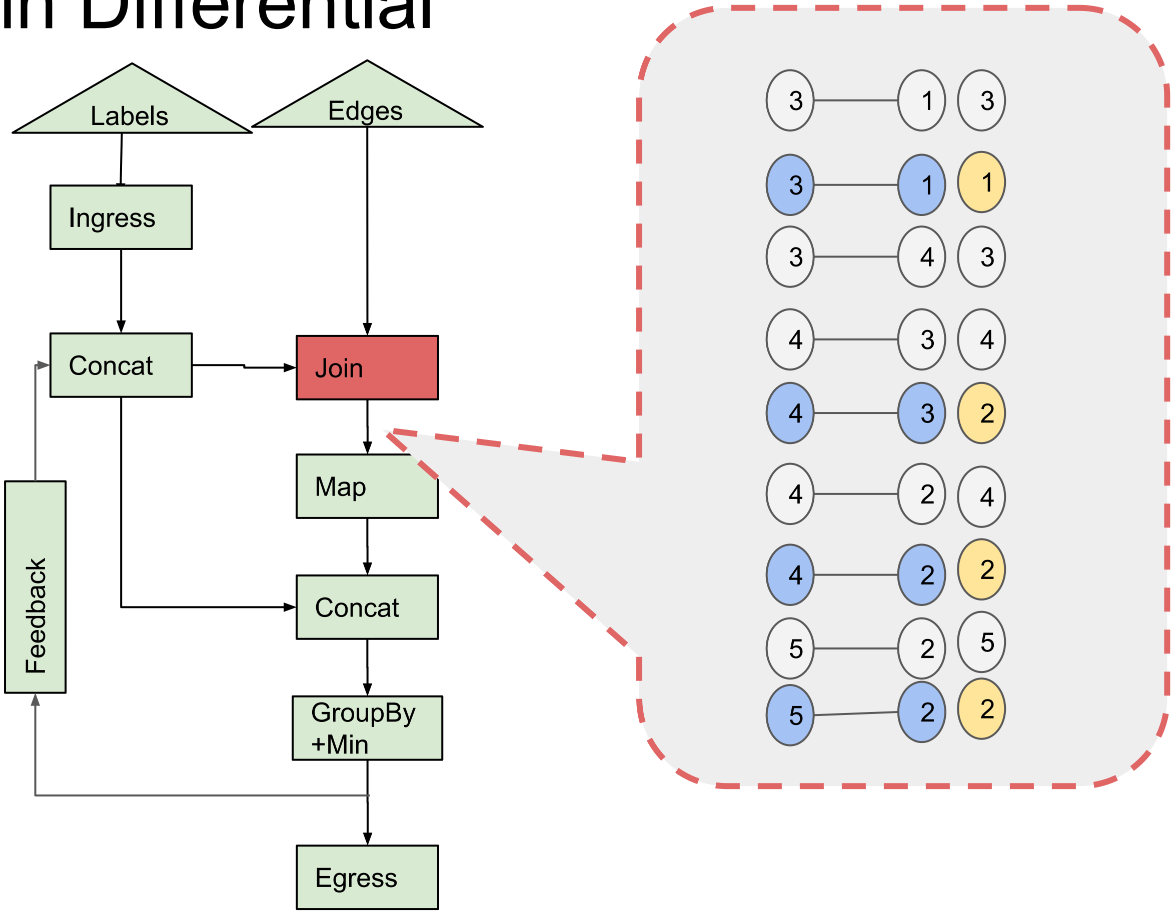
Connected Graph in Differential



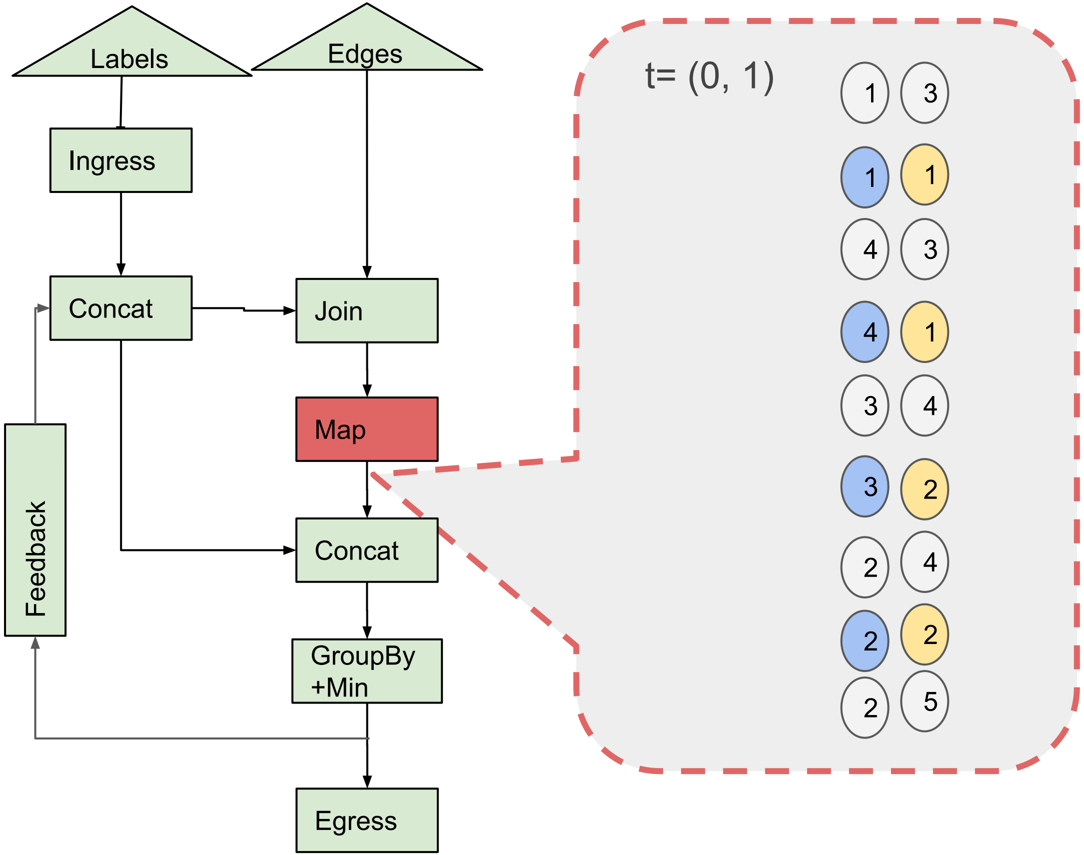
Connected Graph in Differential



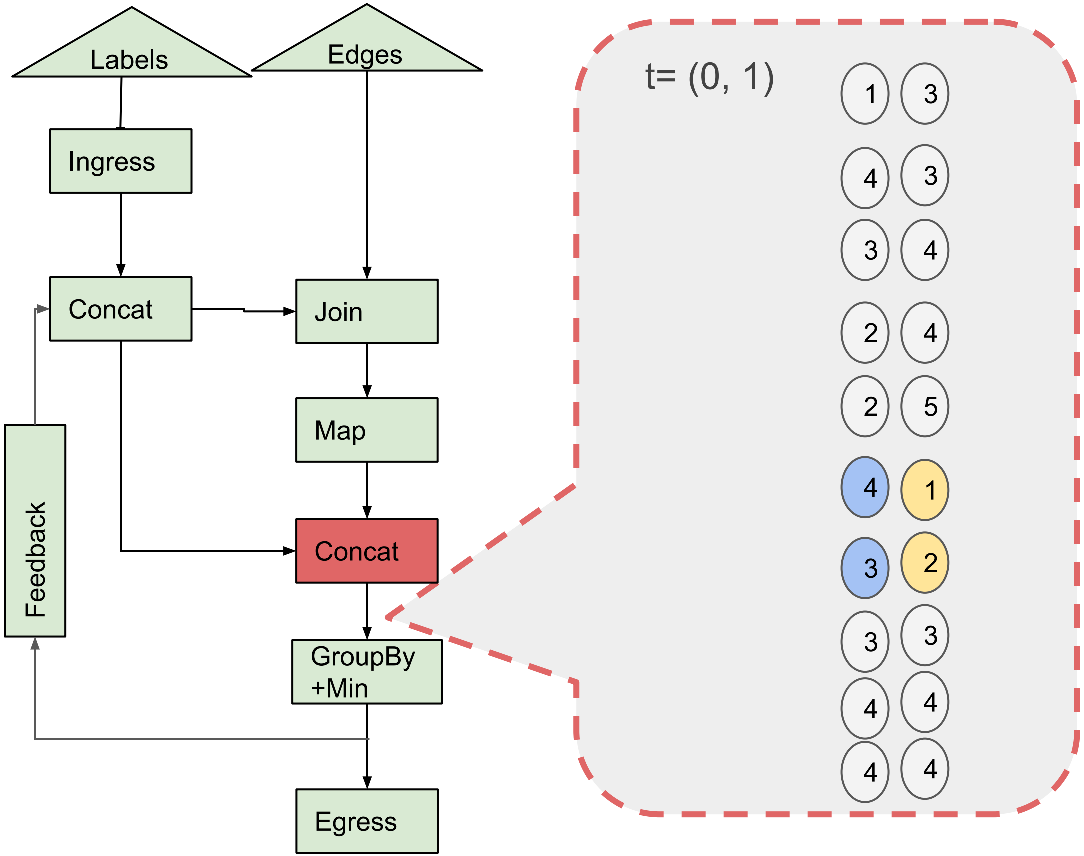
Connected Graph in Differential



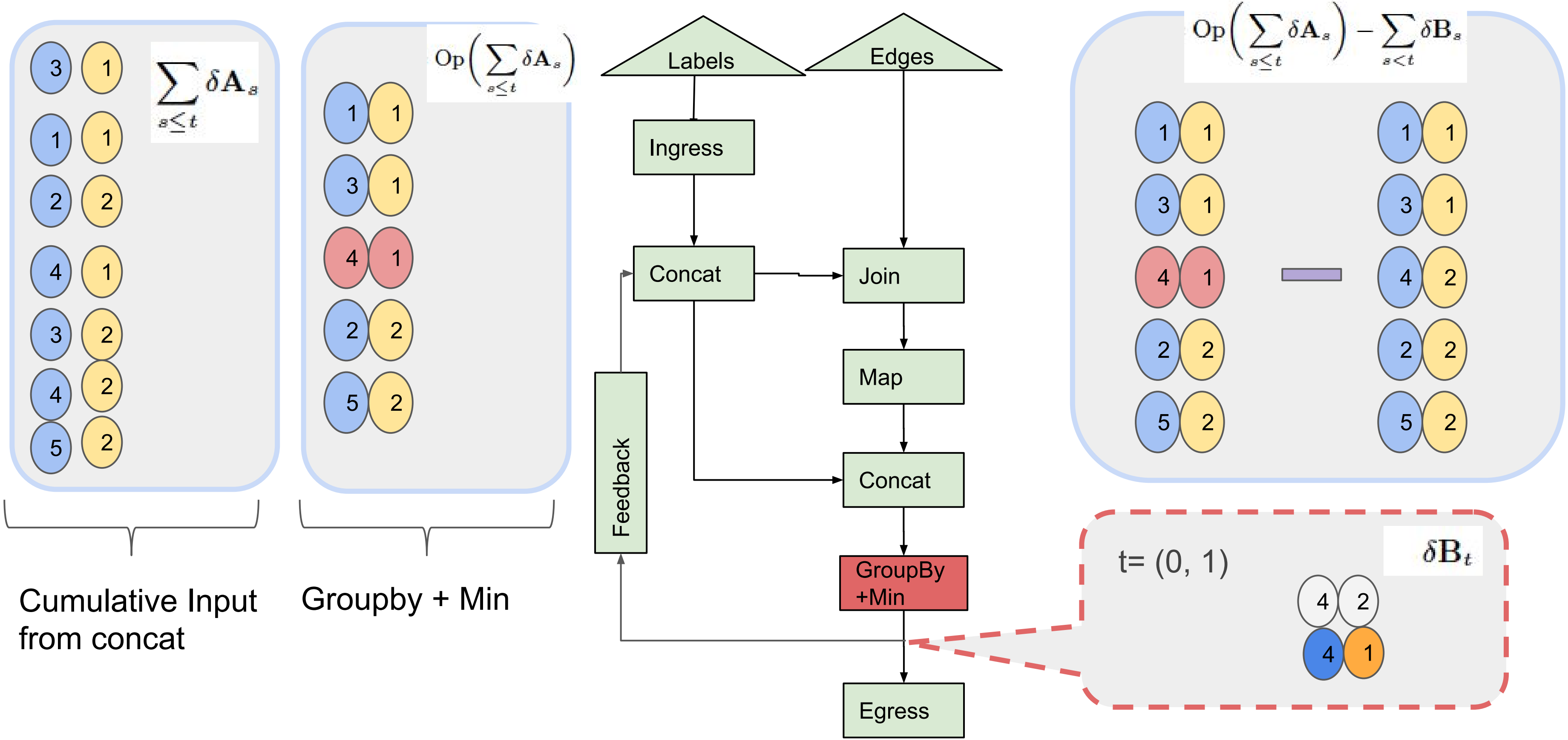
Connected Graph in Differential



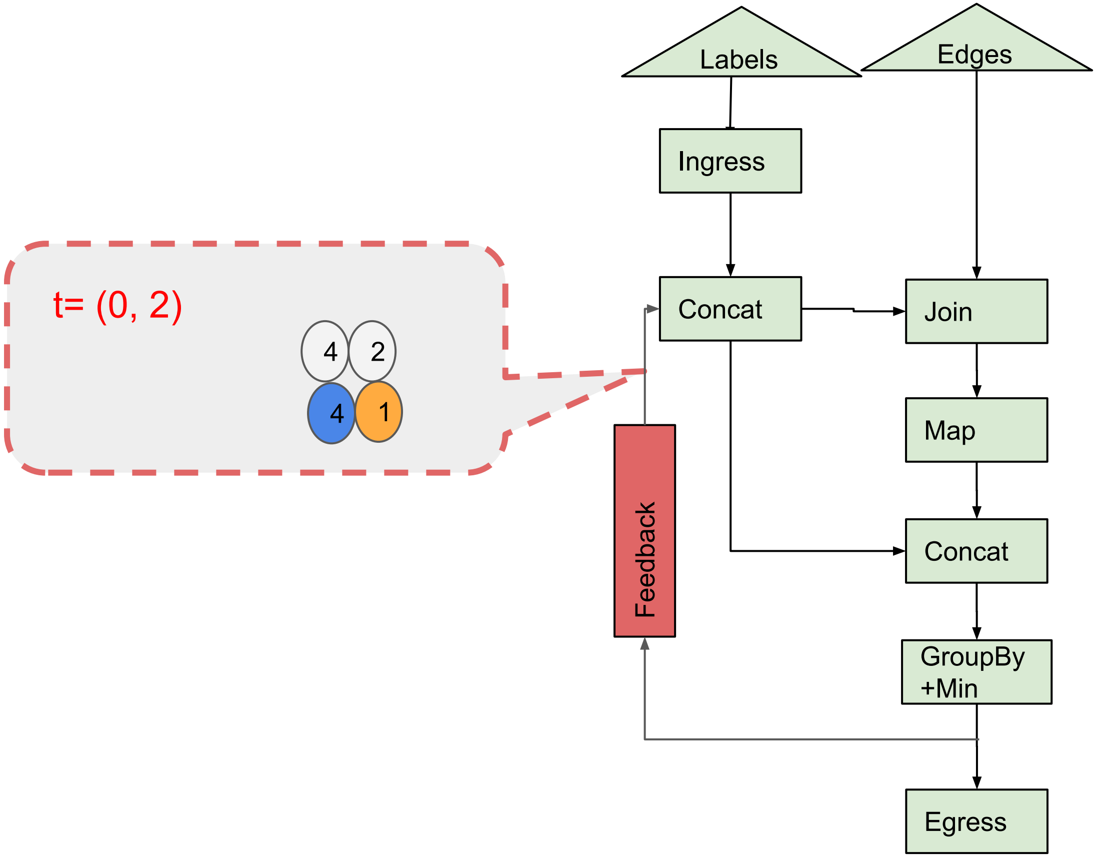
Connected Graph in Differential



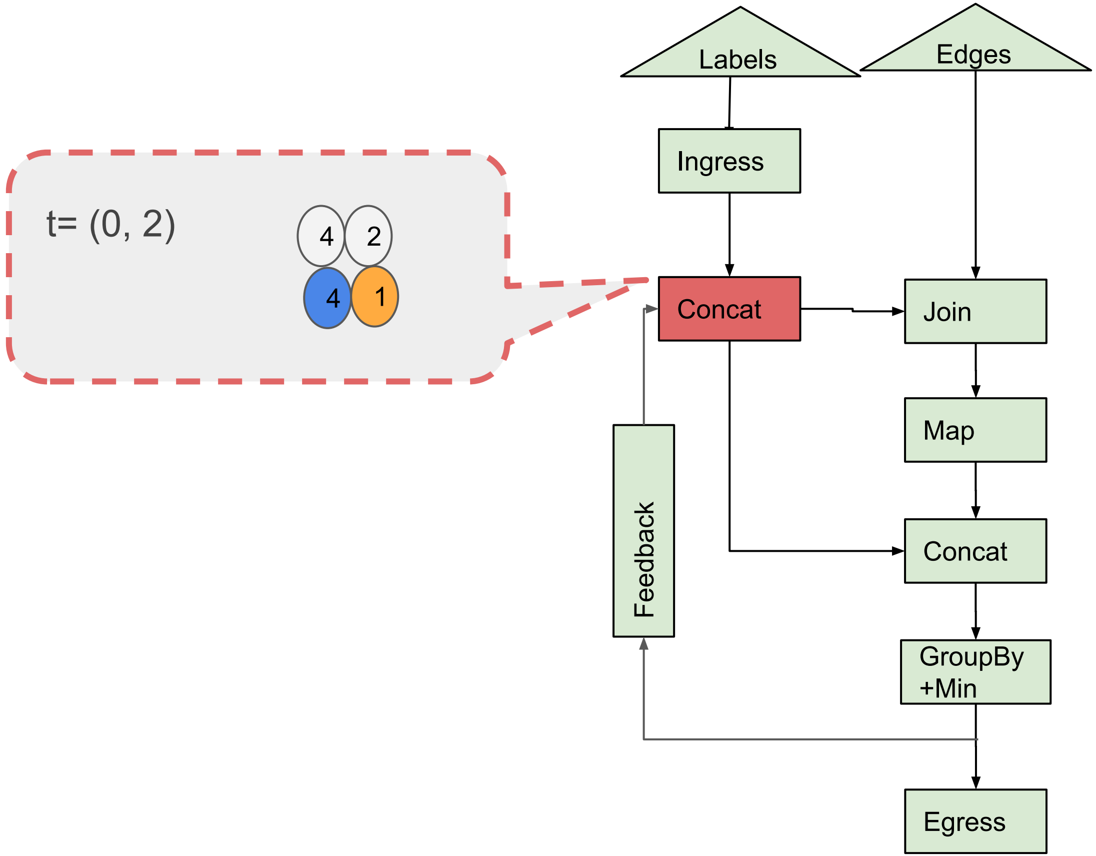
Connected Graph in Differential



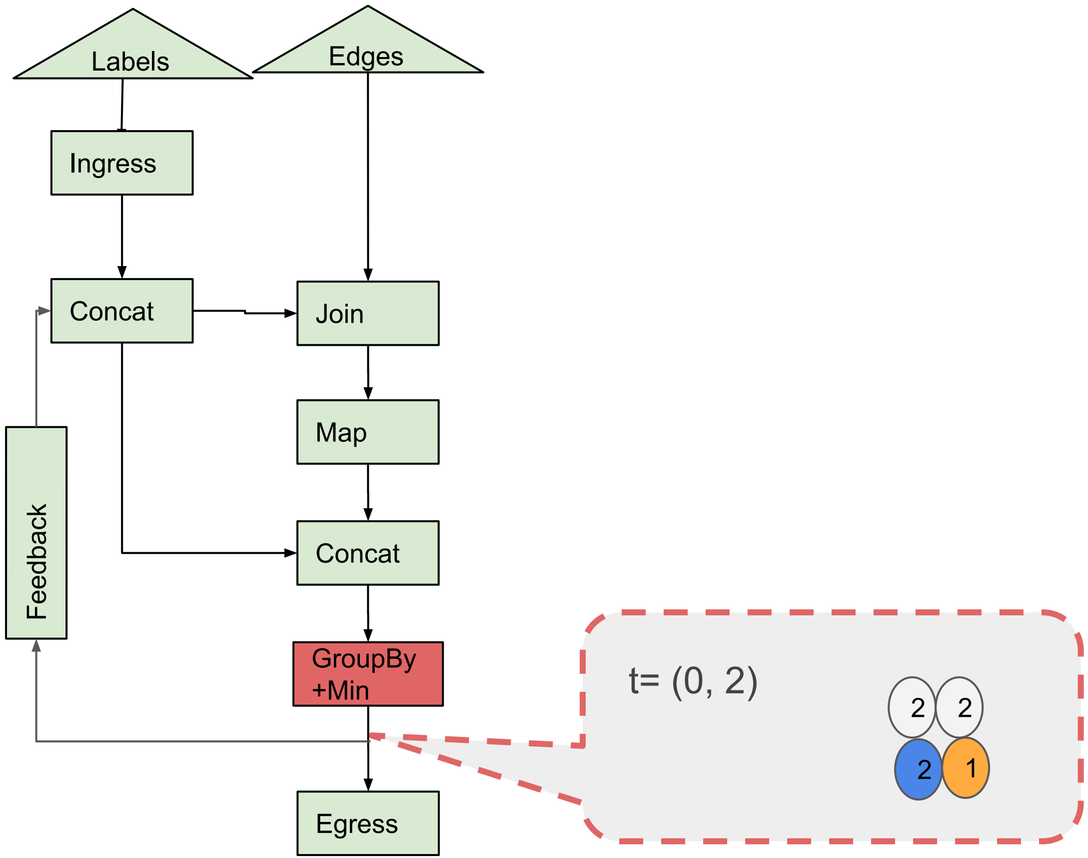
Connected Graph in Differential



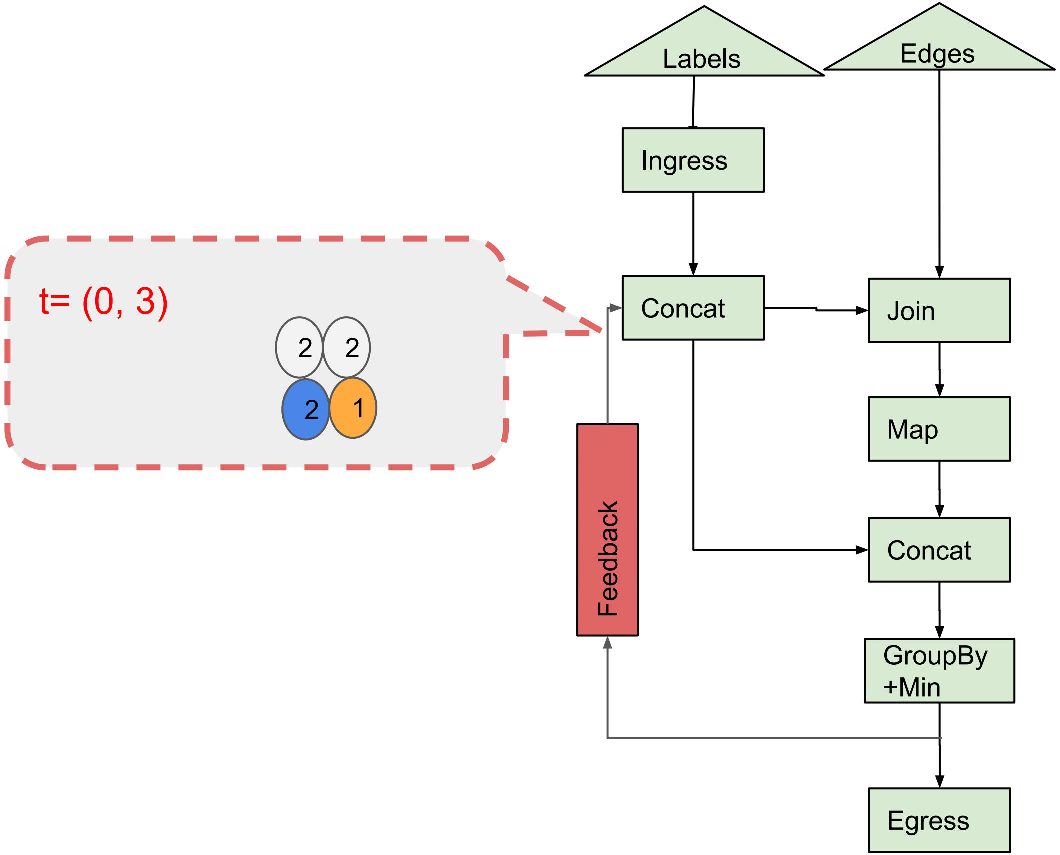
Connected Graph in Differential



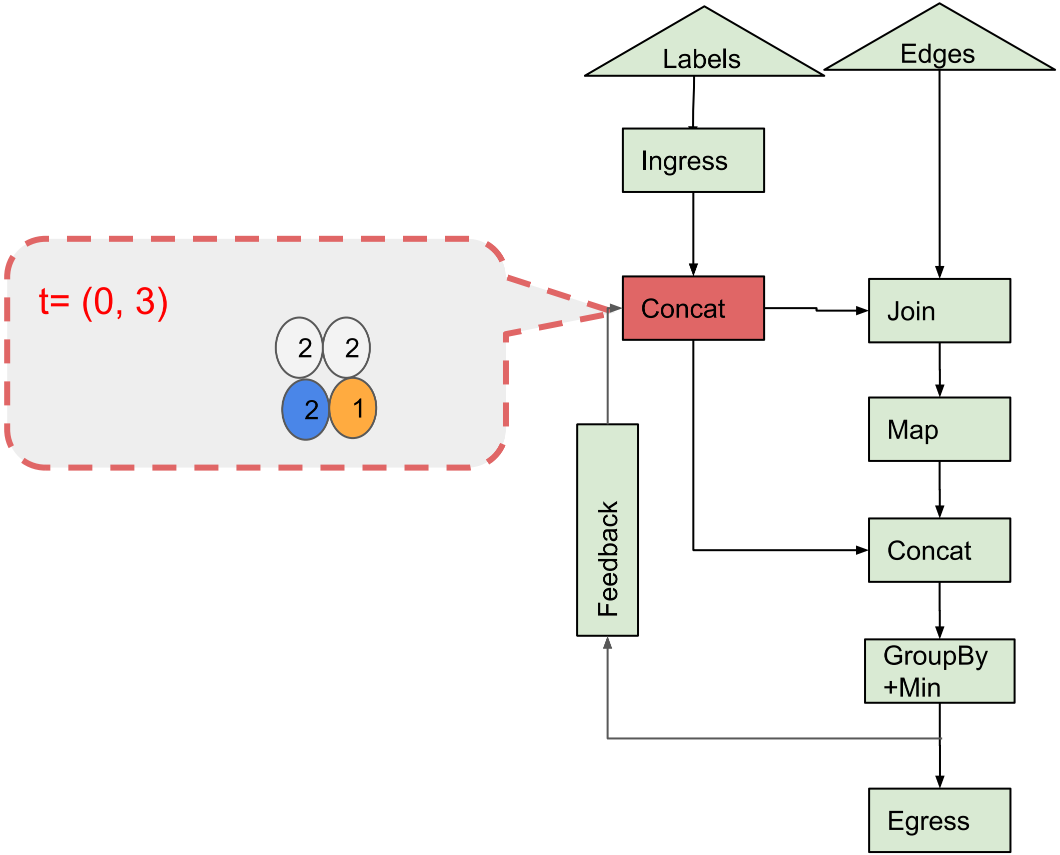
Connected Graph in Differential



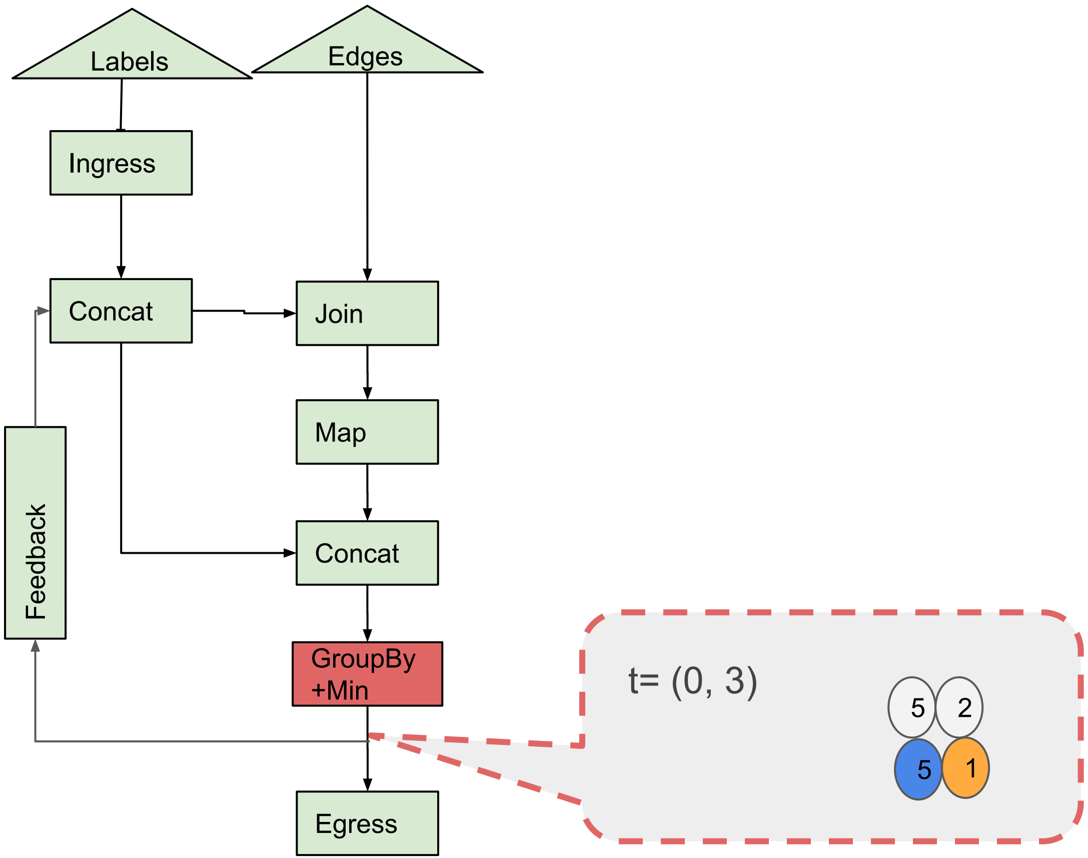
Connected Graph in Differential



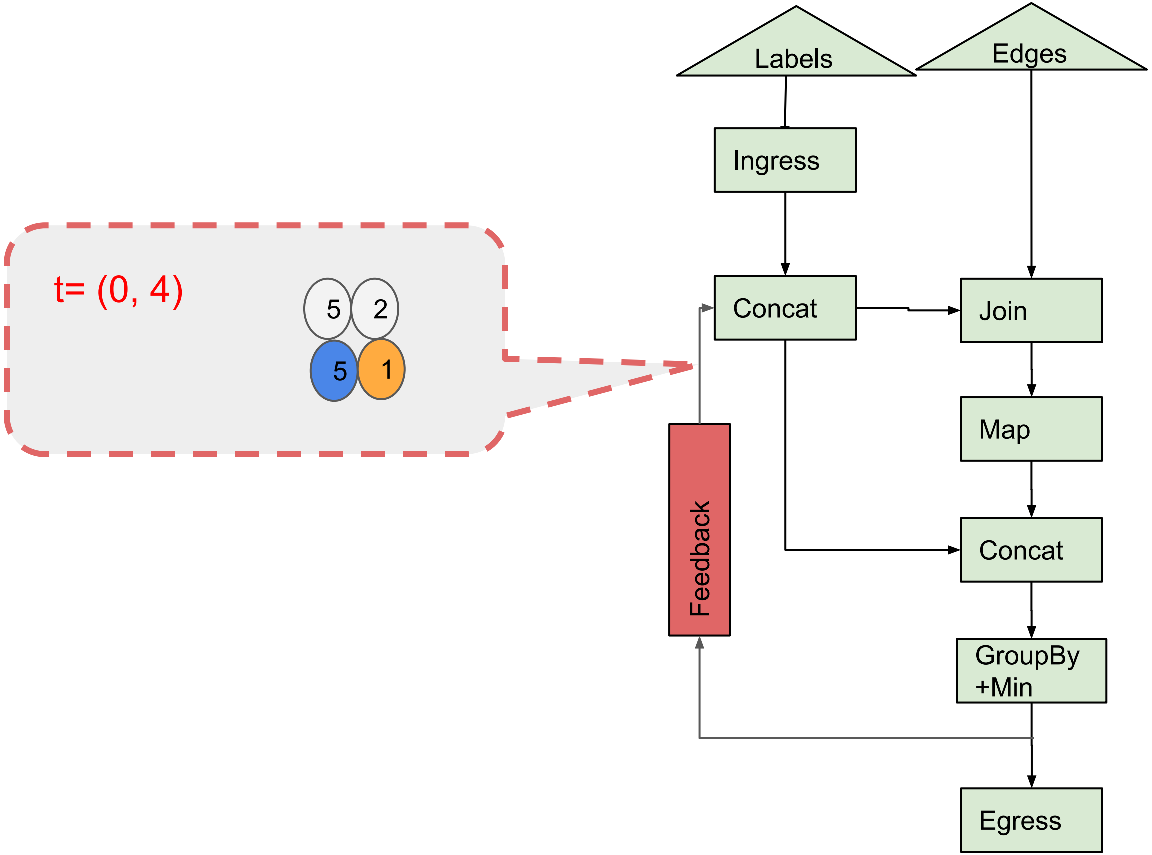
Connected Graph in Differential



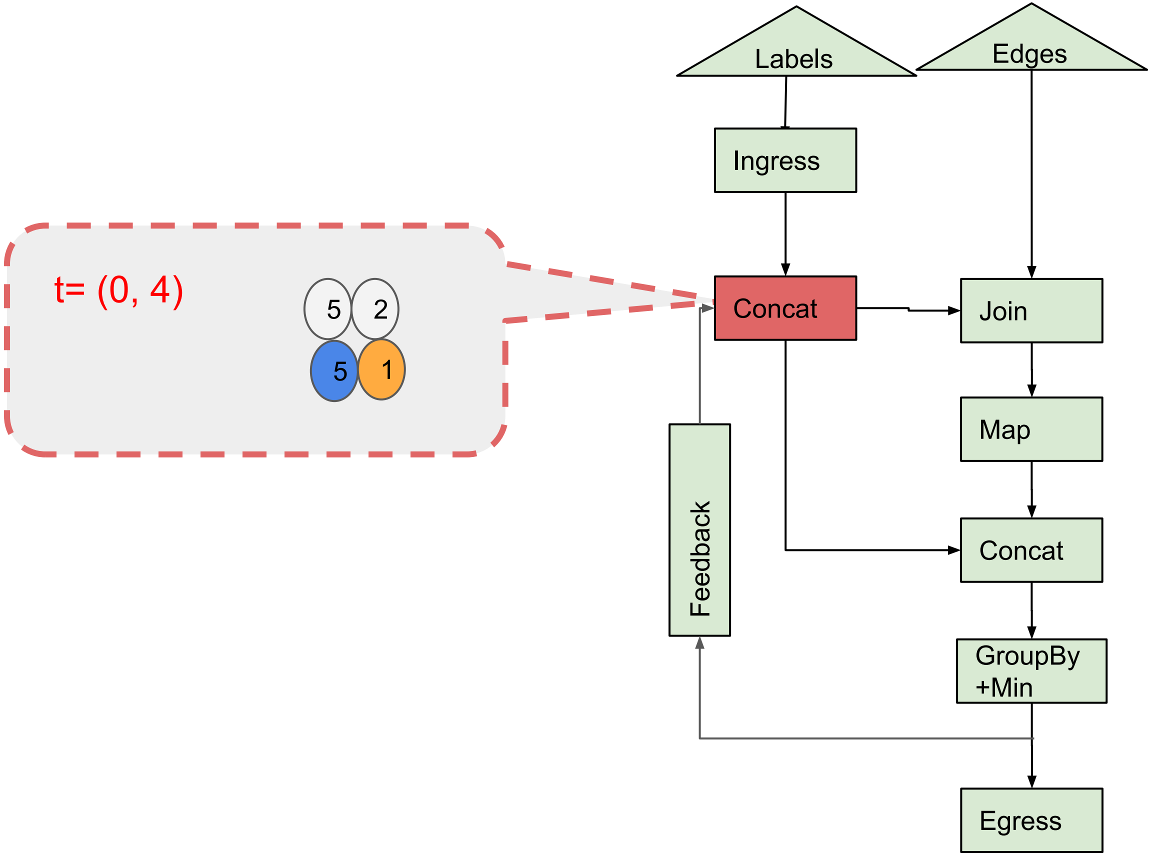
Connected Graph in Differential



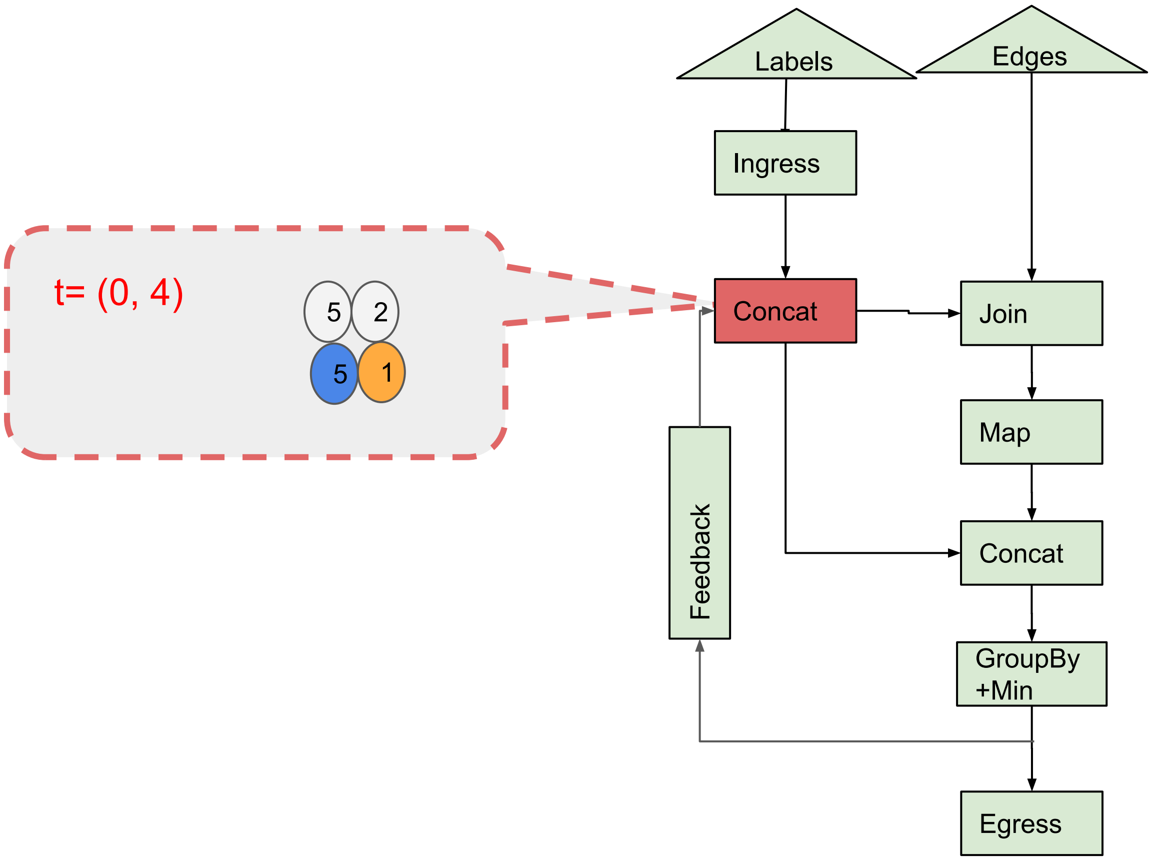
Connected Graph in Differential



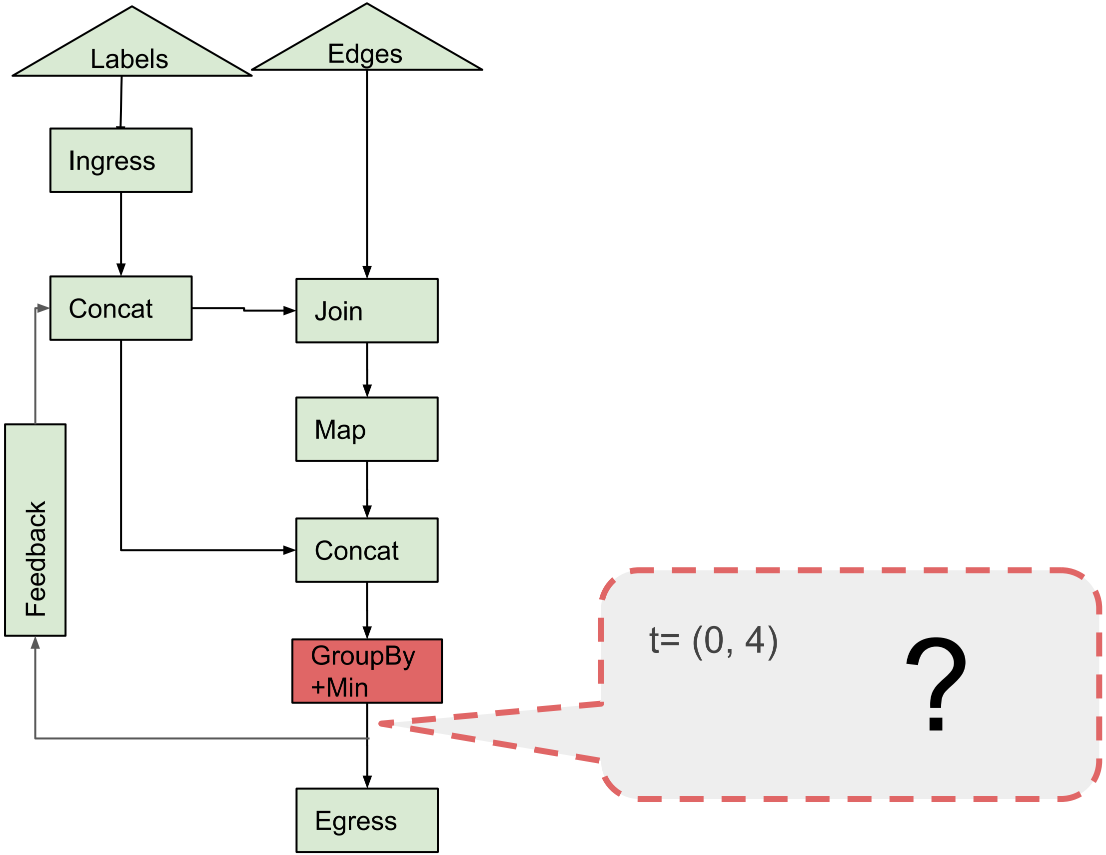
Connected Graph in Differential



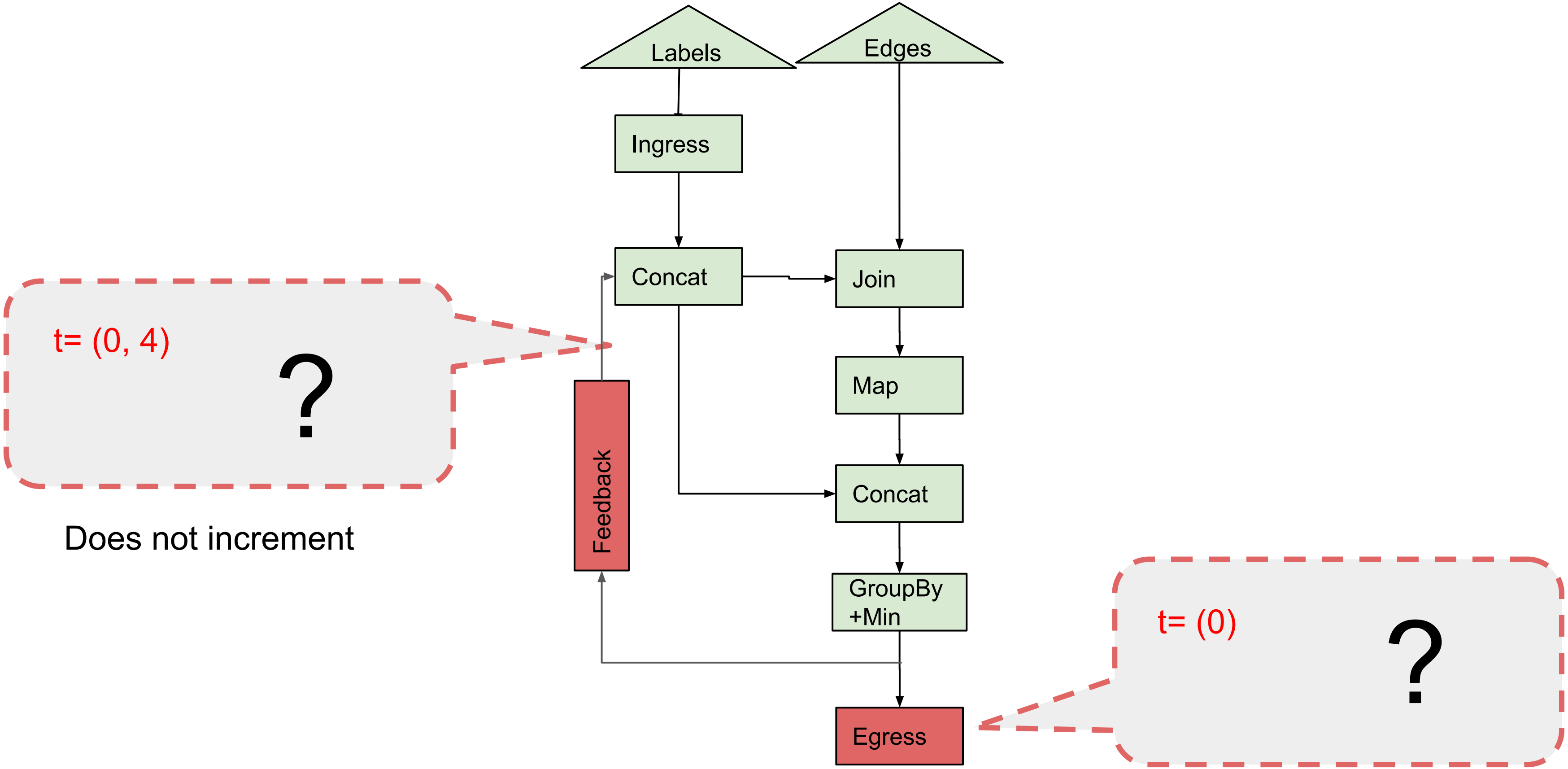
Connected Graph in Differential



Connected Graph in Differential

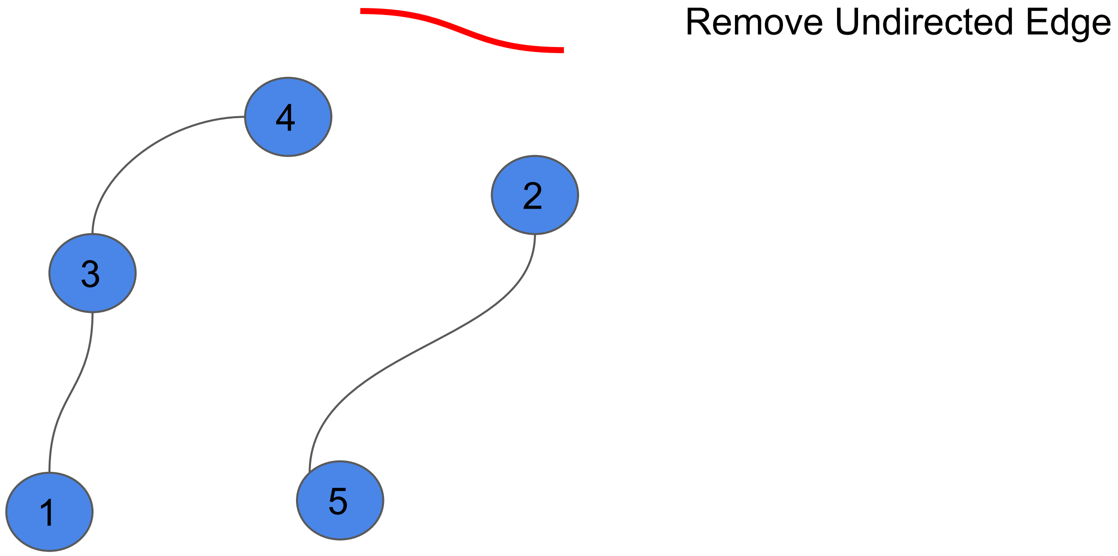


Connected Graph in Differential

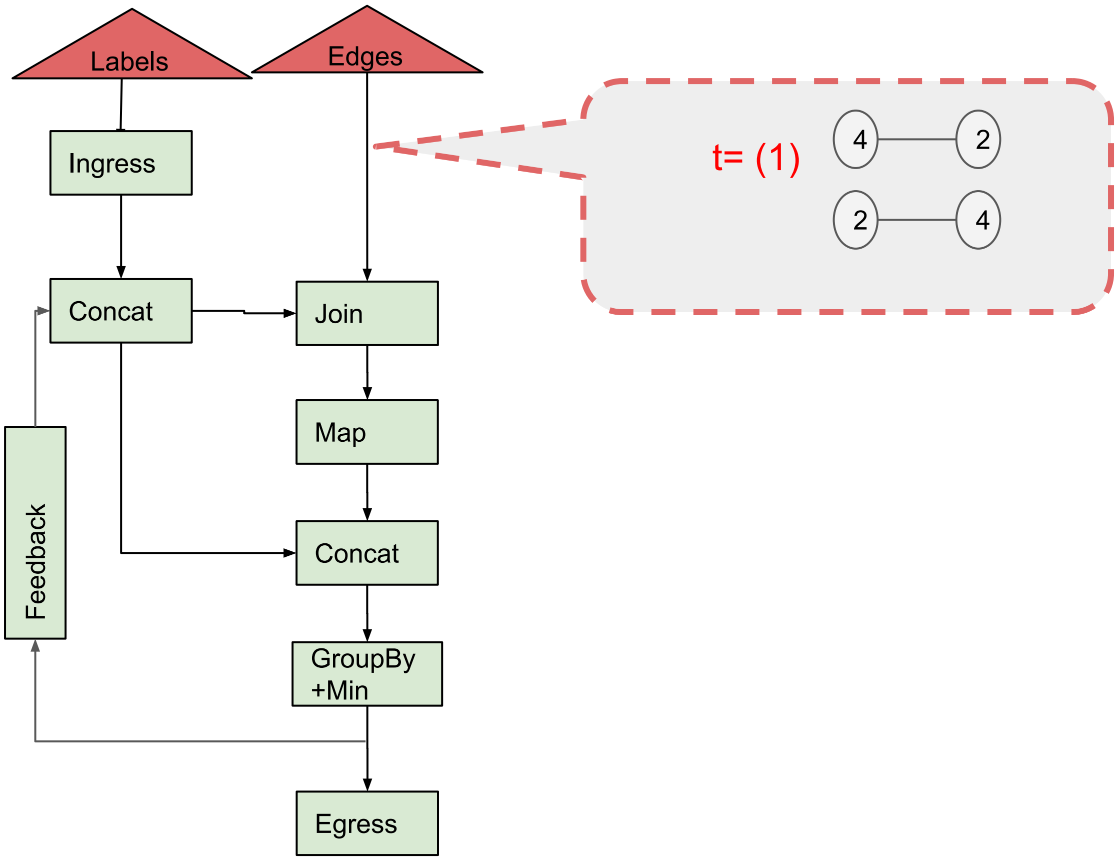


Does not increment

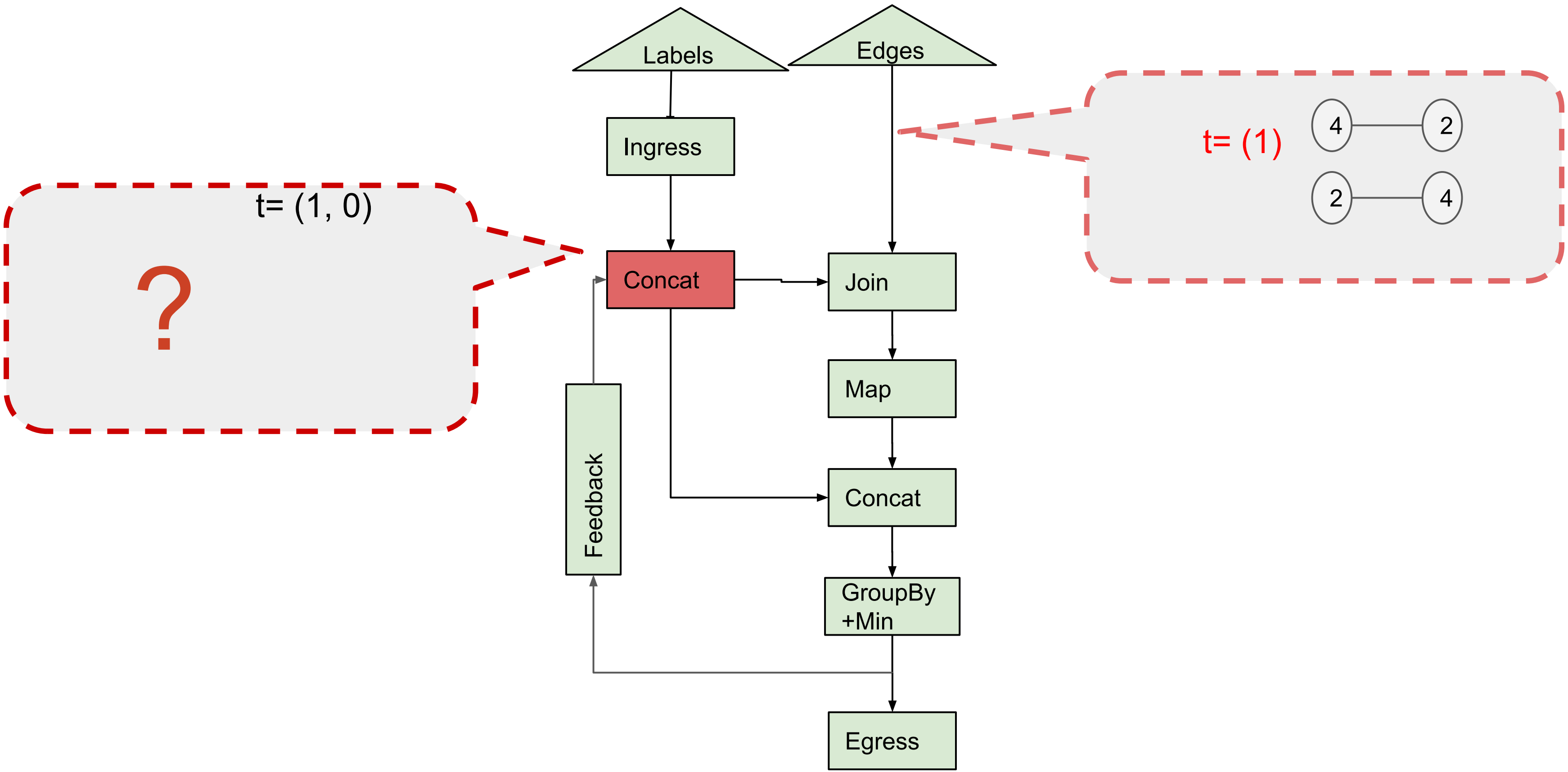
Changes to Connected Graph - I



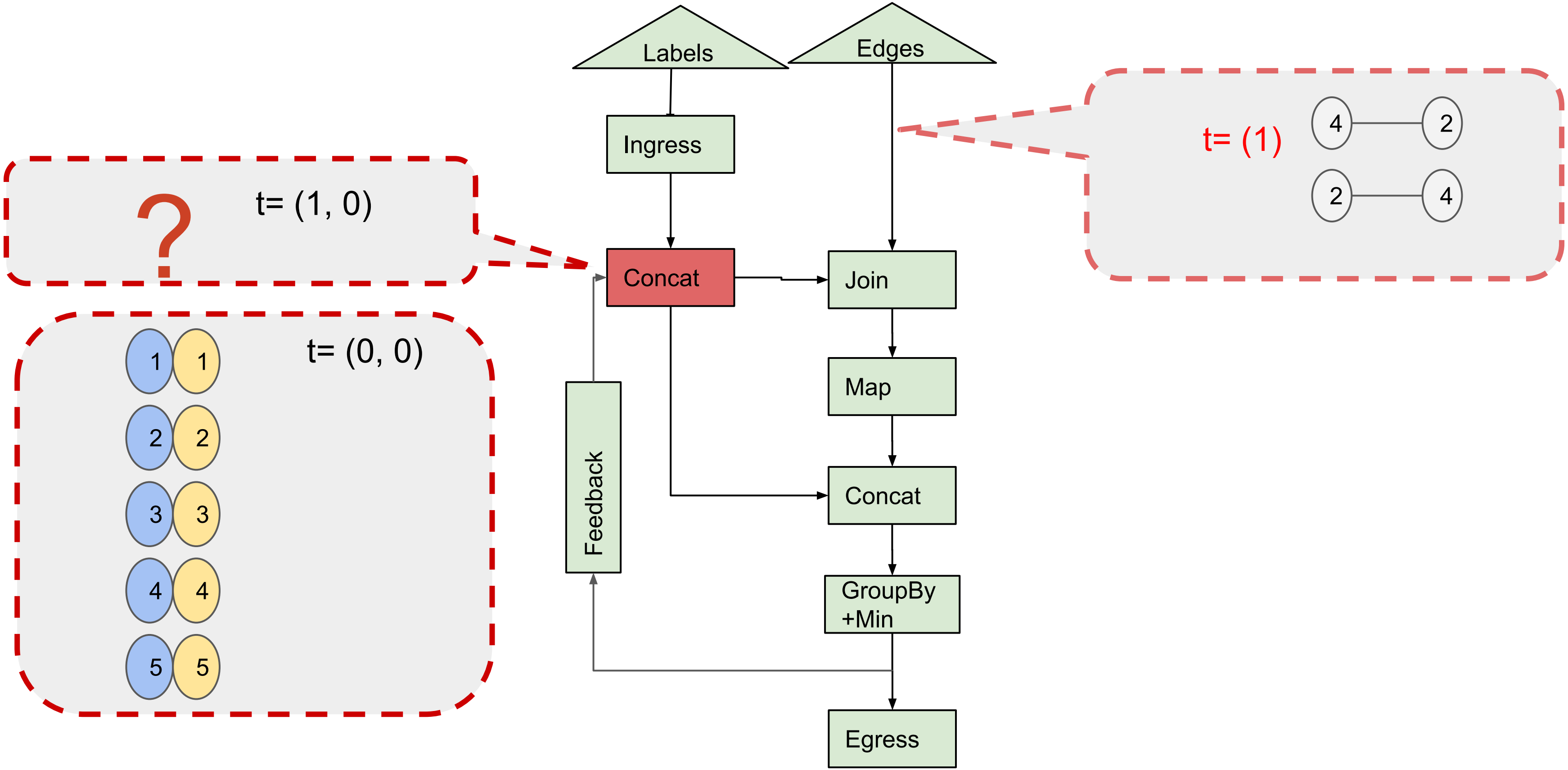
Changes to Connected Graph - I



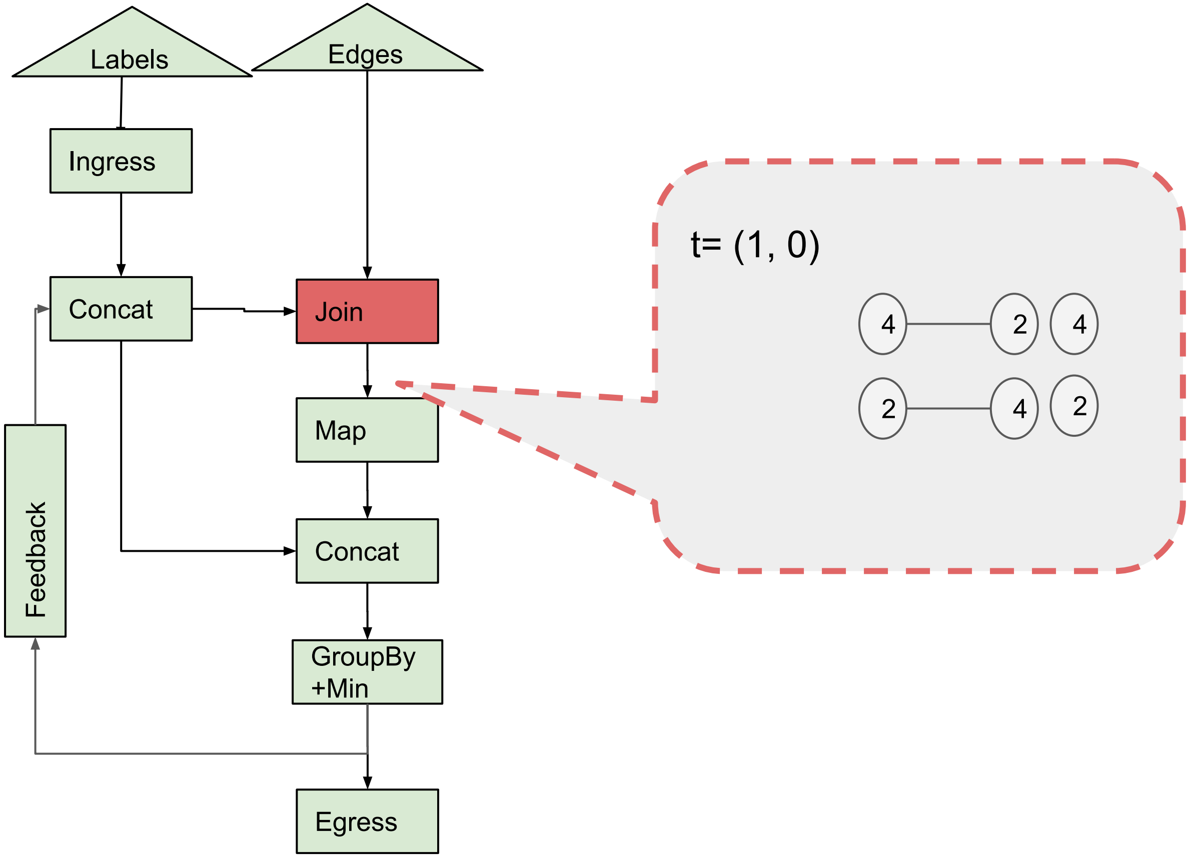
Changes to Connected Graph - I



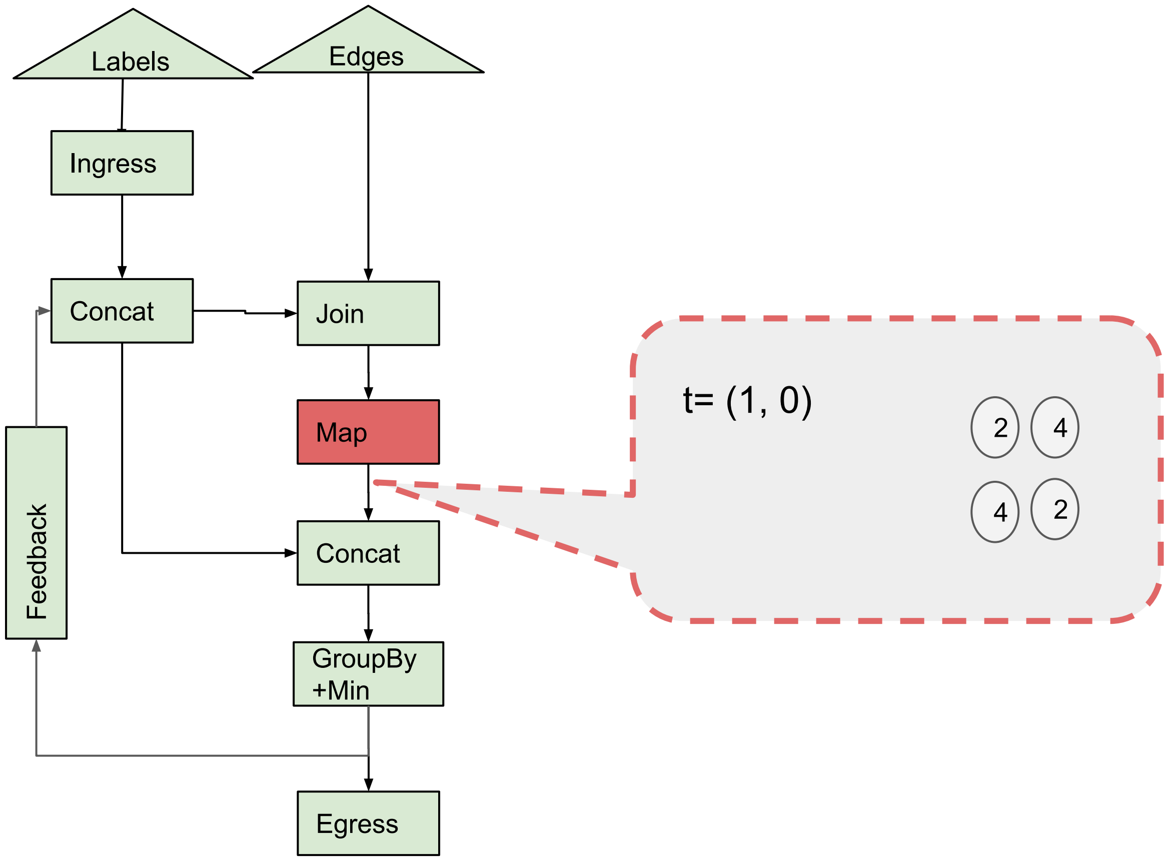
Changes to Connected Graph - I



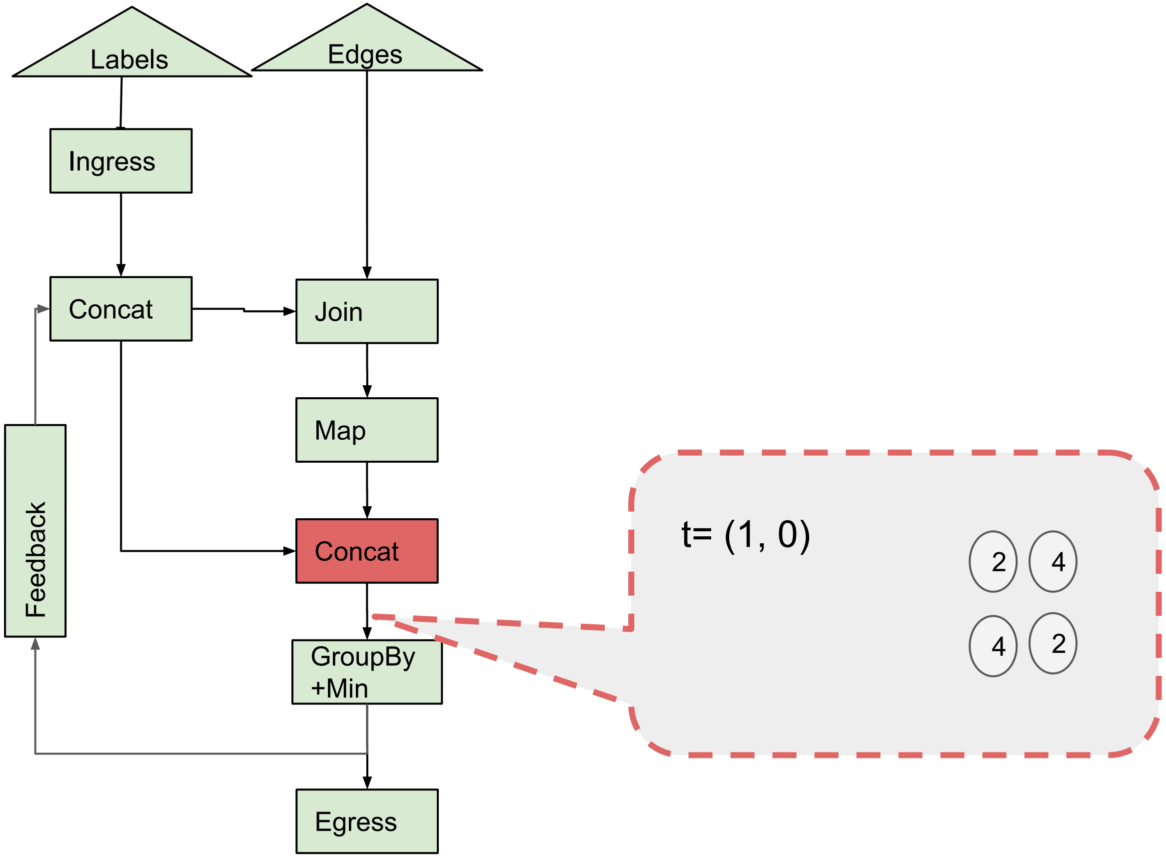
Changes to Connected Graph - I



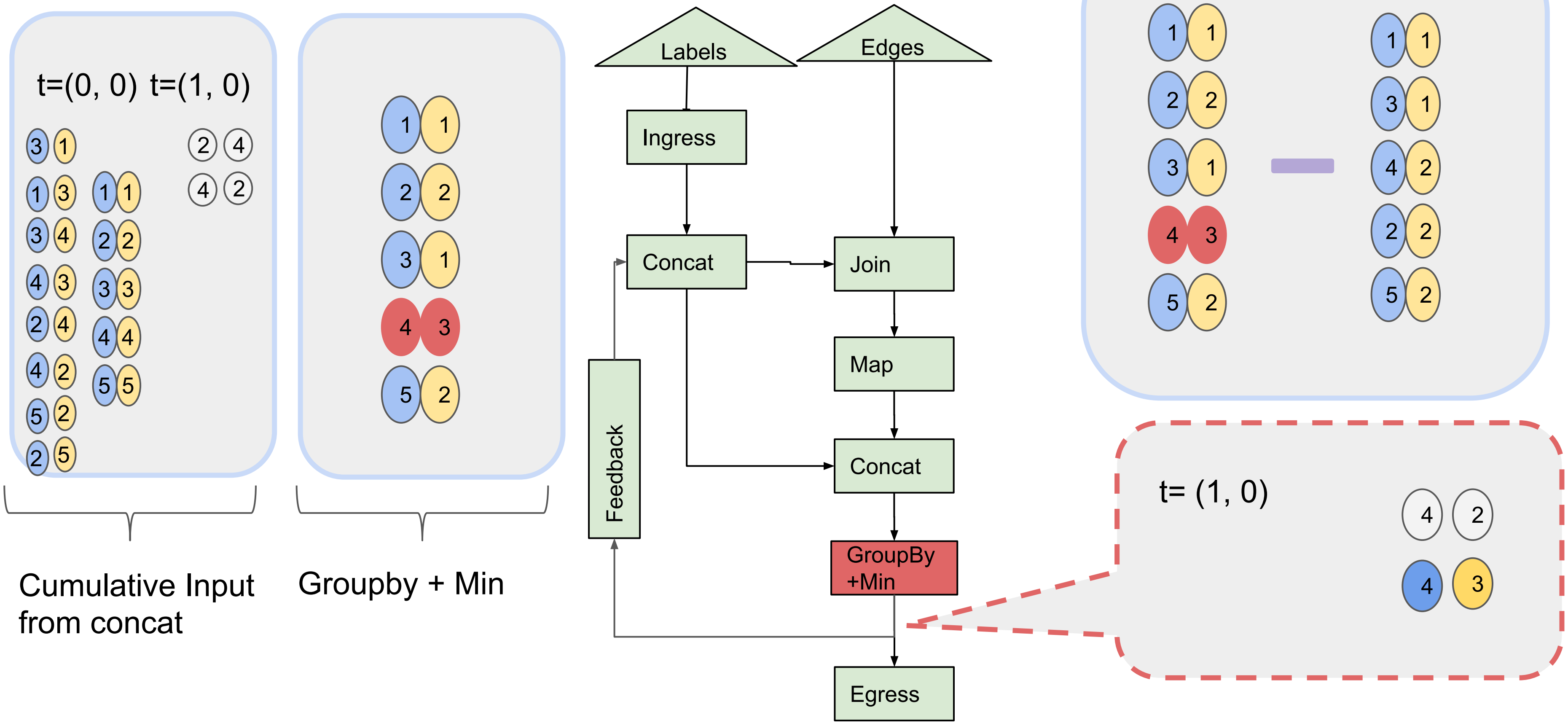
Changes to Connected Graph - I



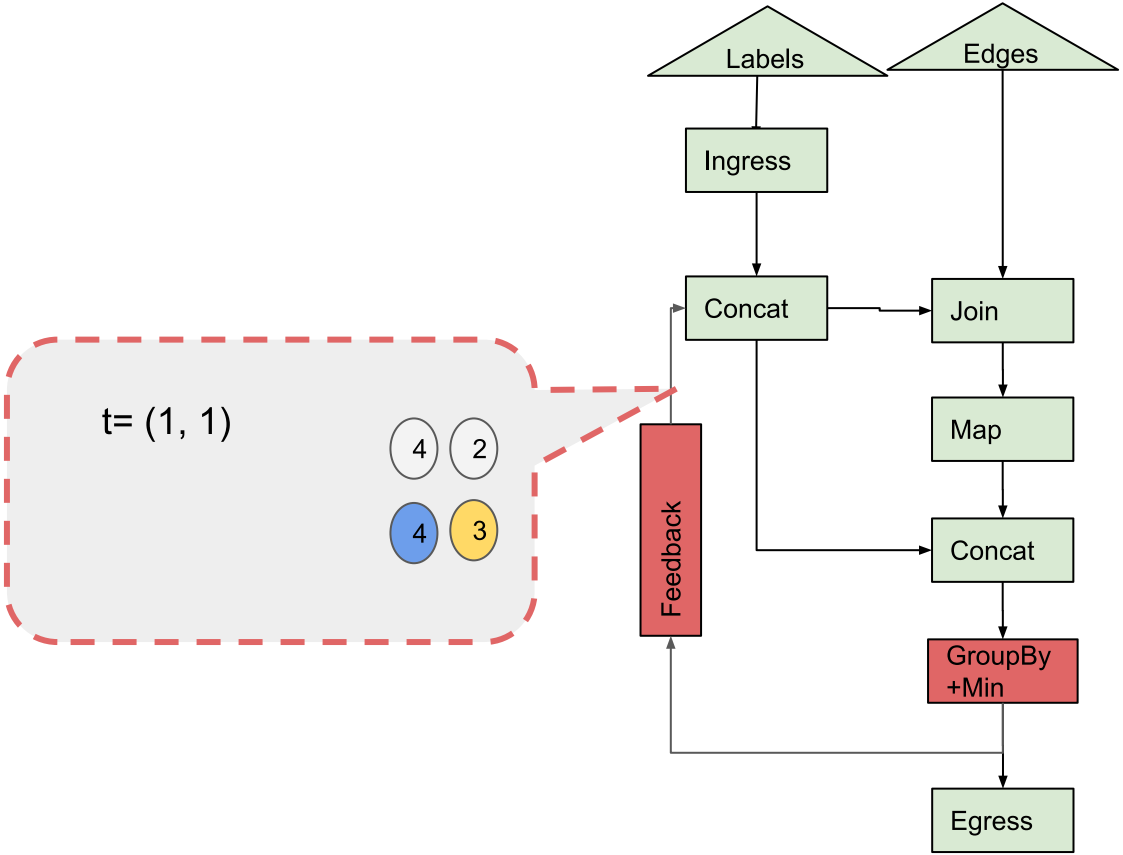
Changes to Connected Graph - I



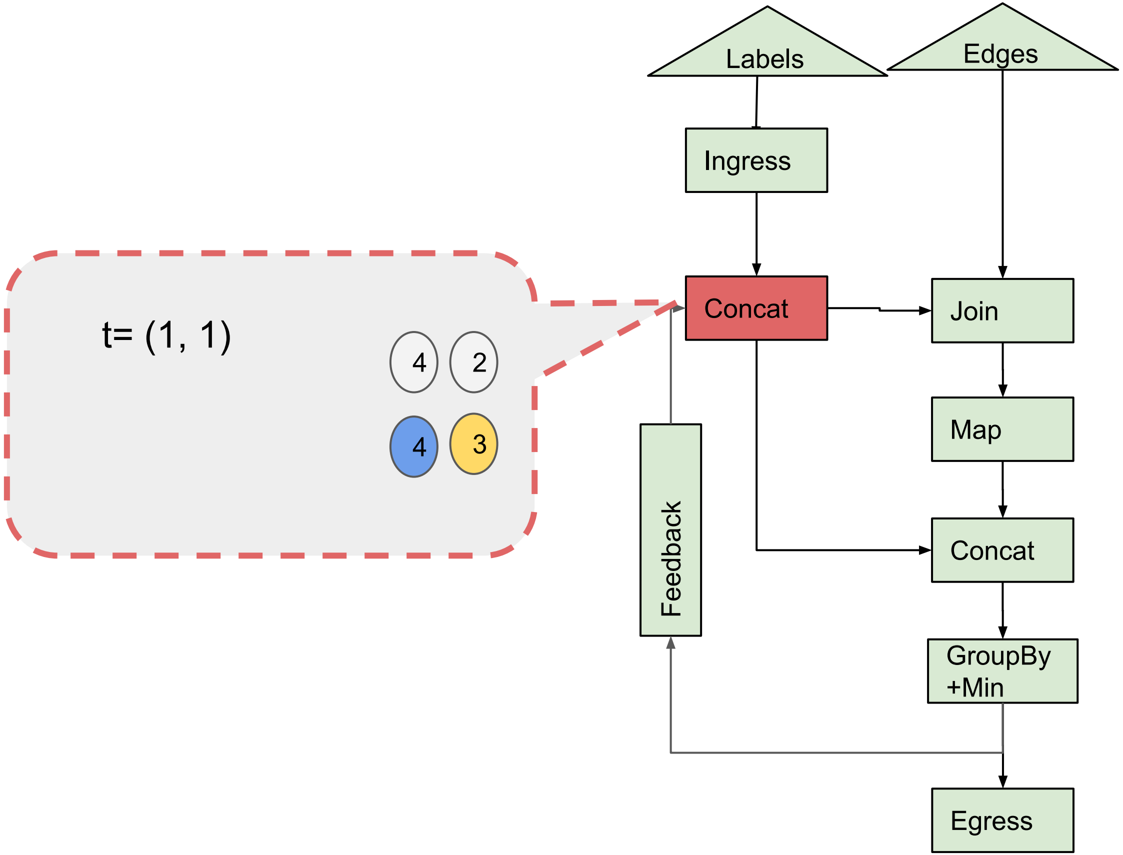
Changes to Connected Graph - I



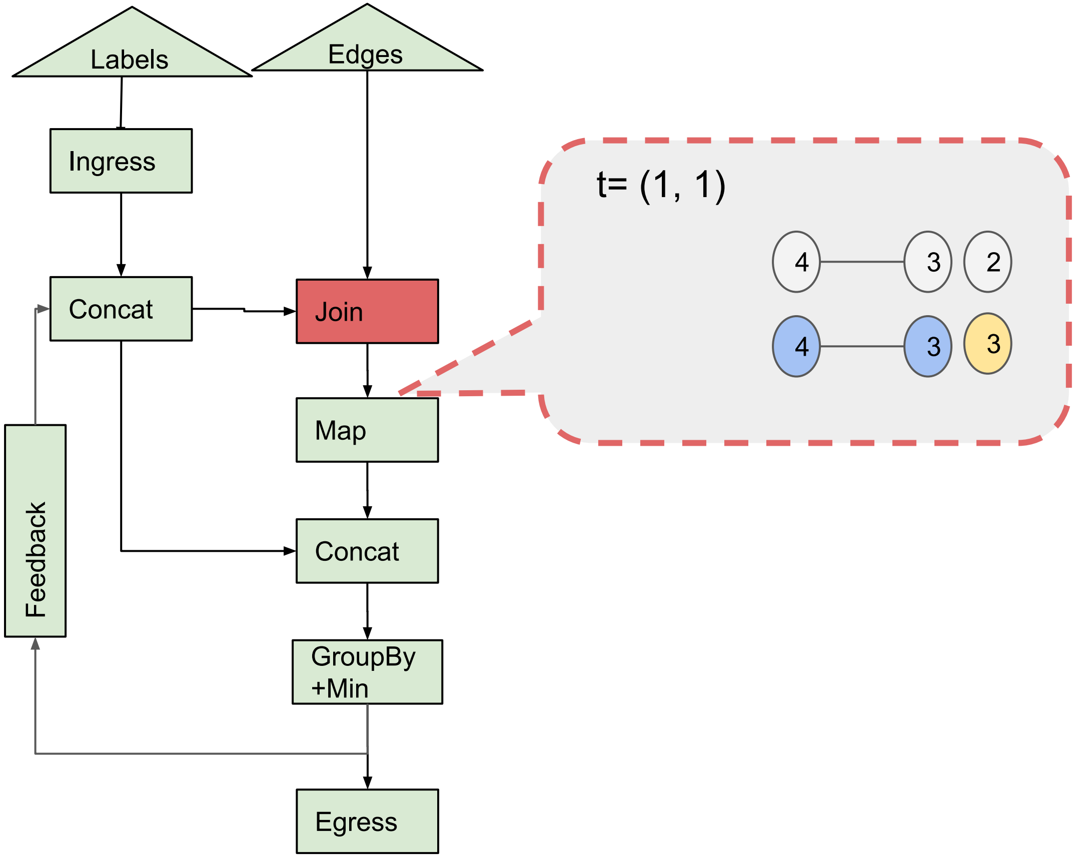
Changes to Connected Graph - I



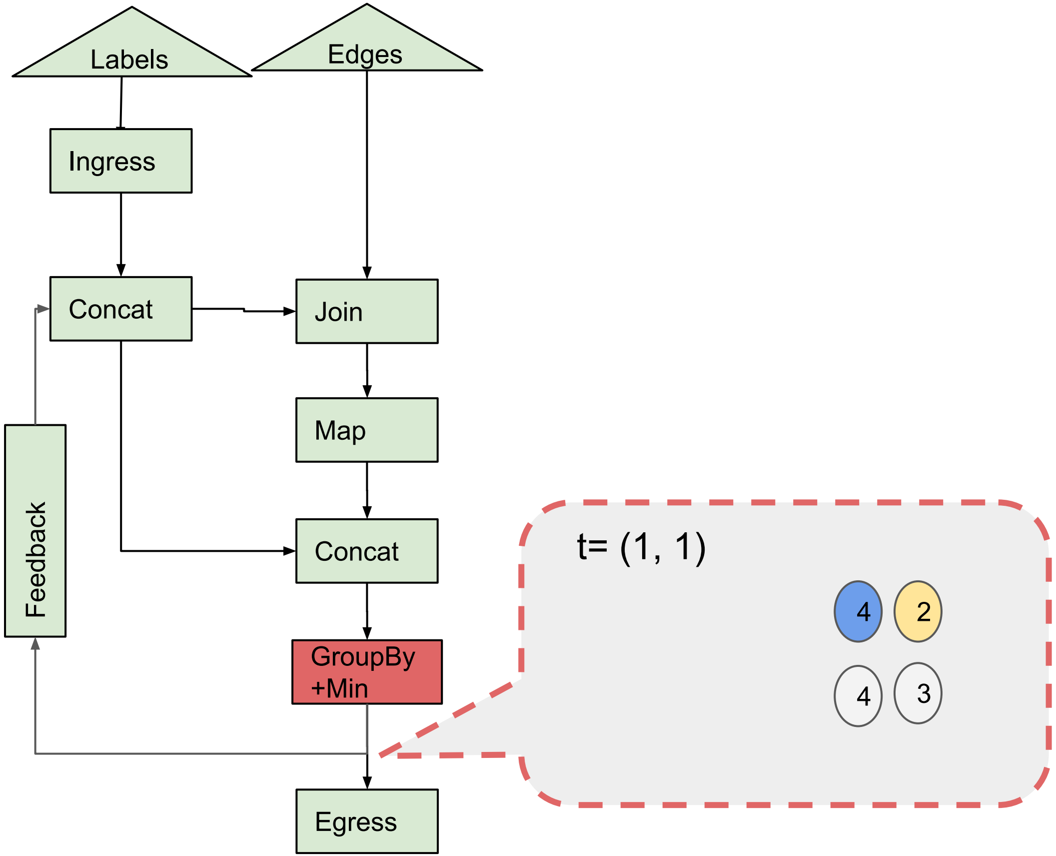
Changes to Connected Graph - I



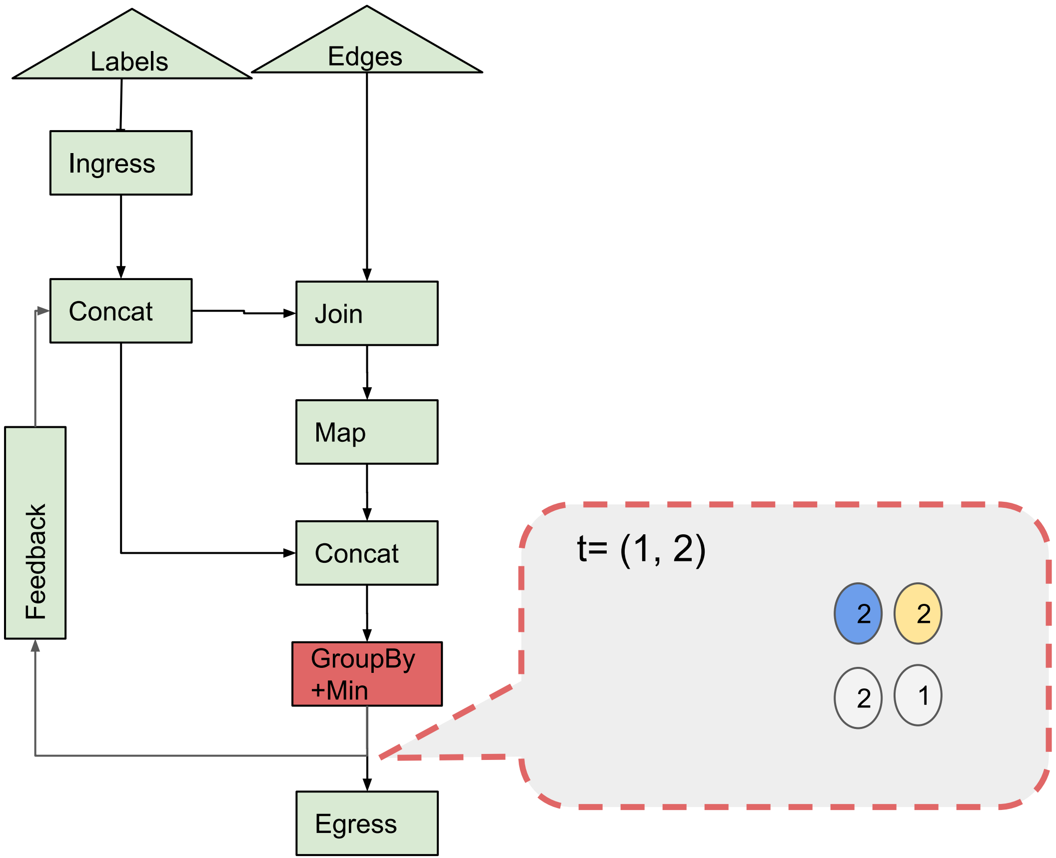
Changes to Connected Graph - I



Changes to Connected Graph - I

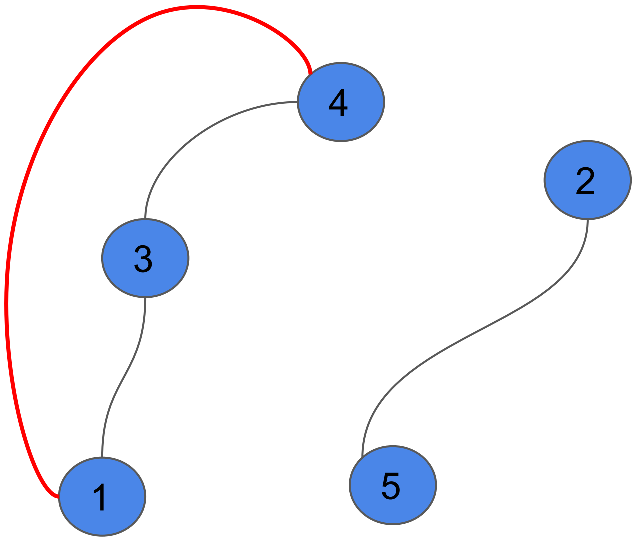


Changes to Connected Graph - I

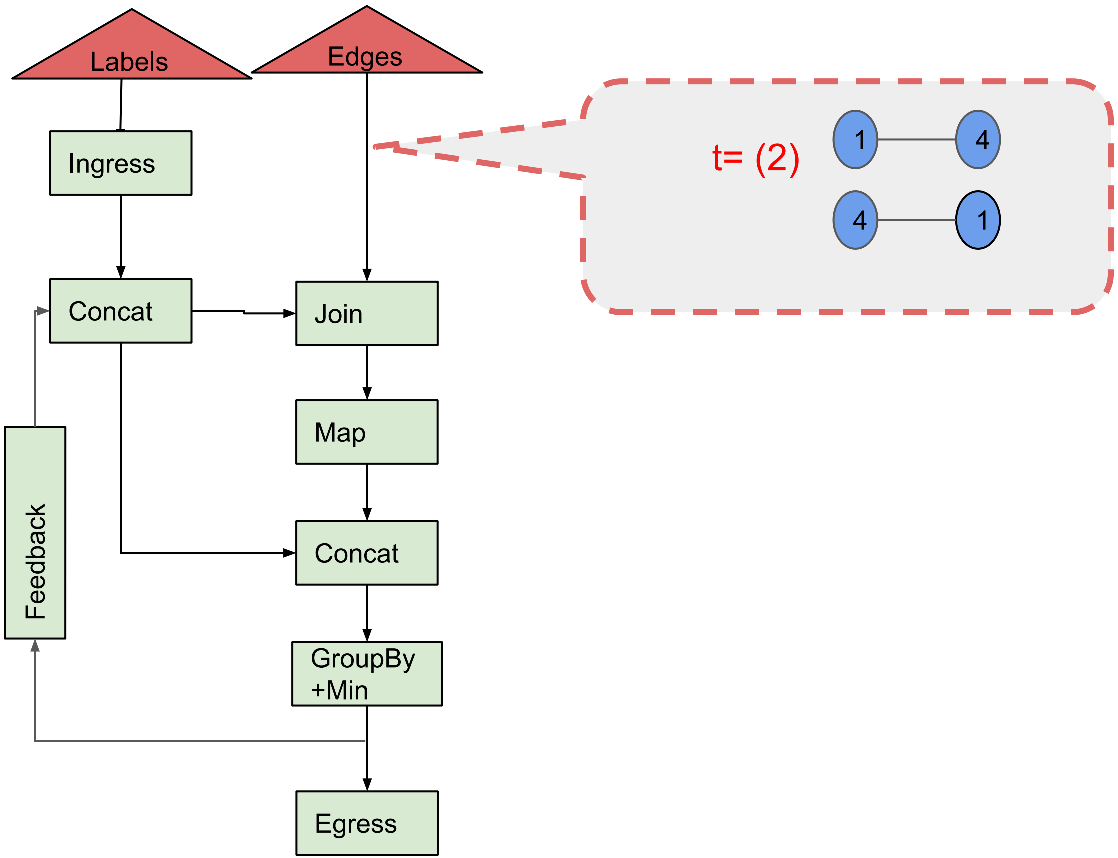


Changes to Connected Graph - II

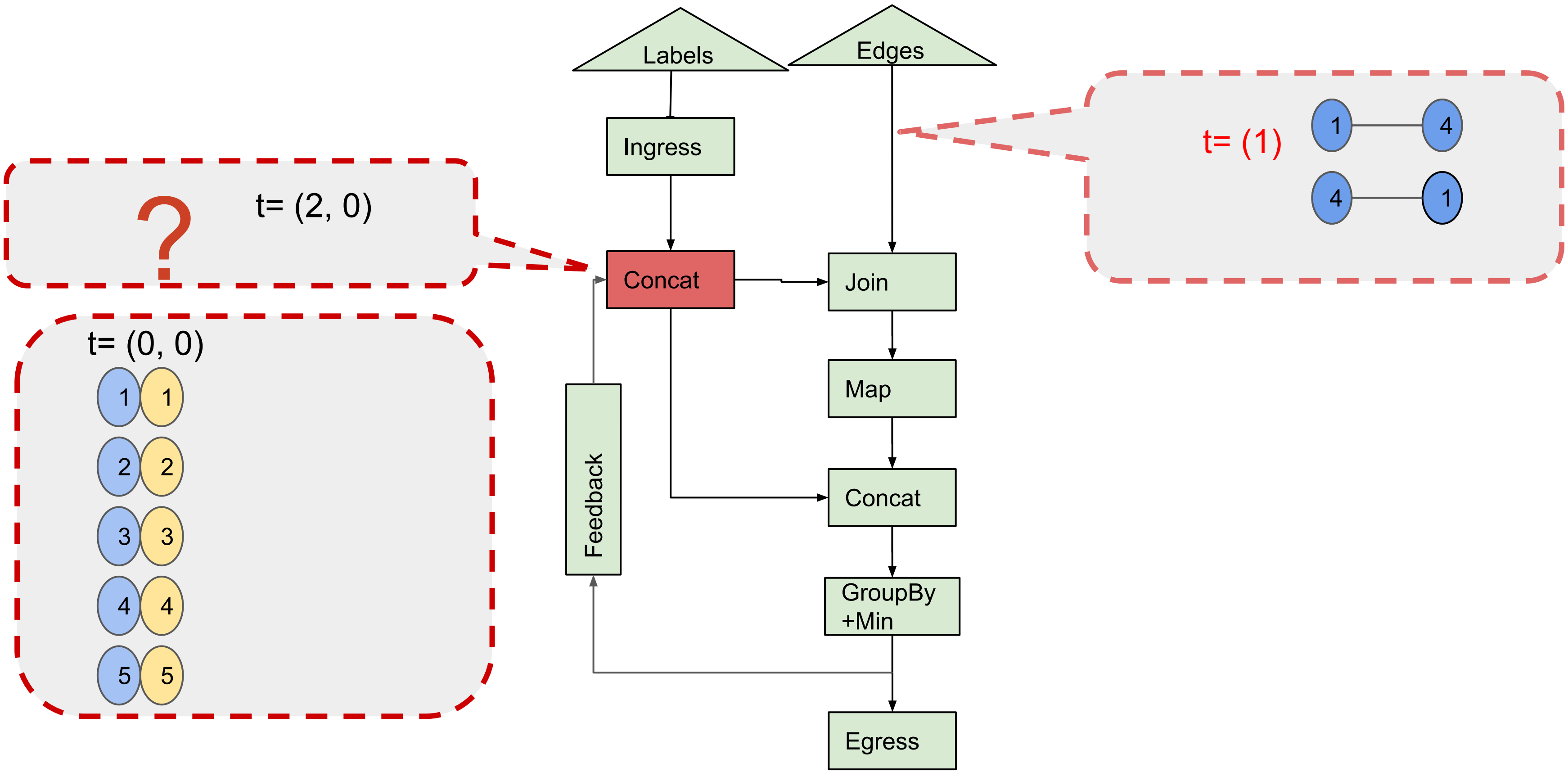
Add Undirected Edge



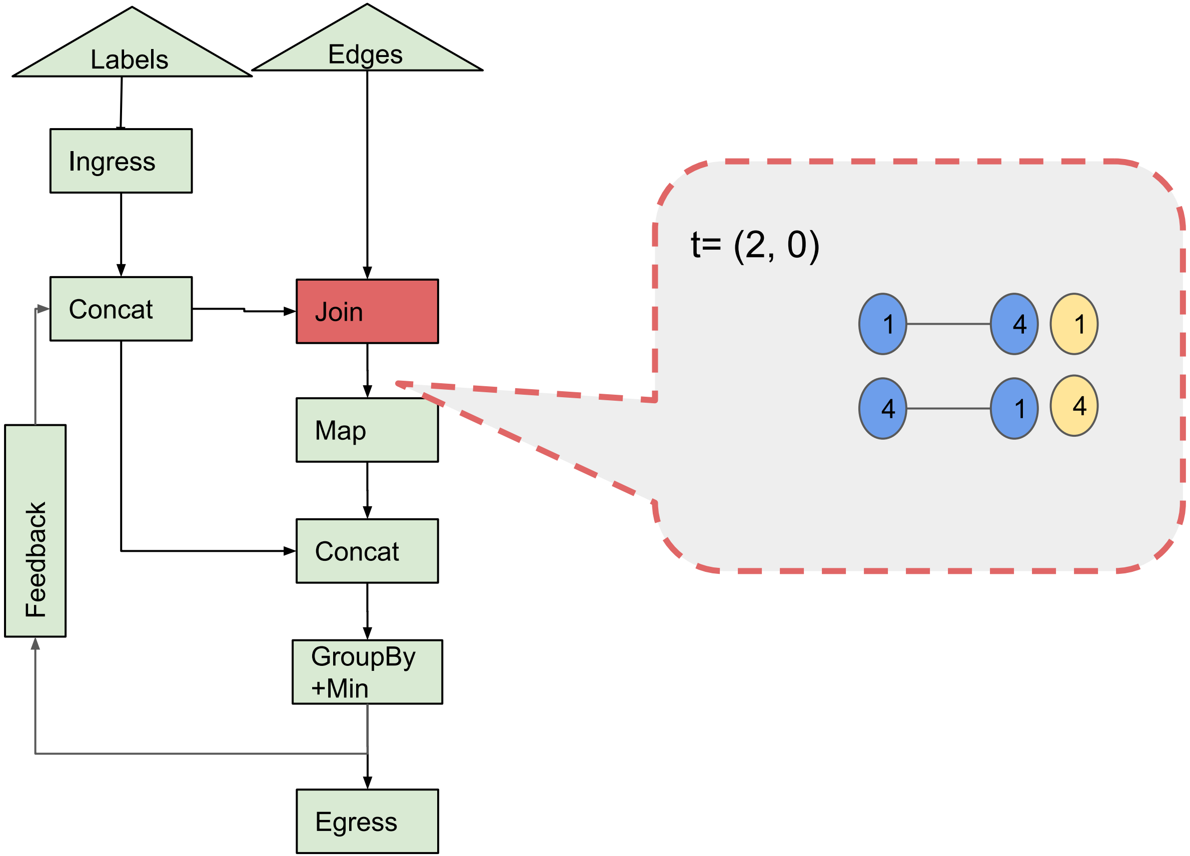
Changes to Connected Graph - II



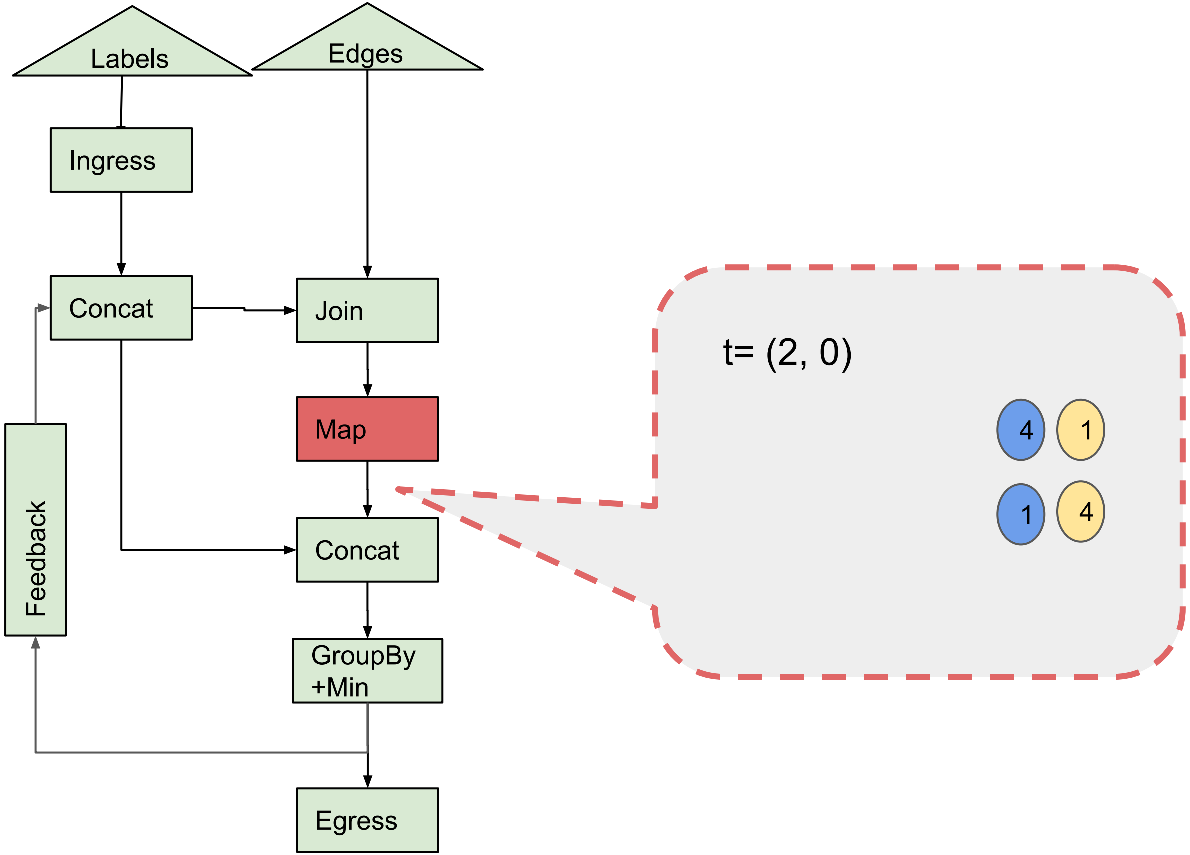
Changes to Connected Graph - II



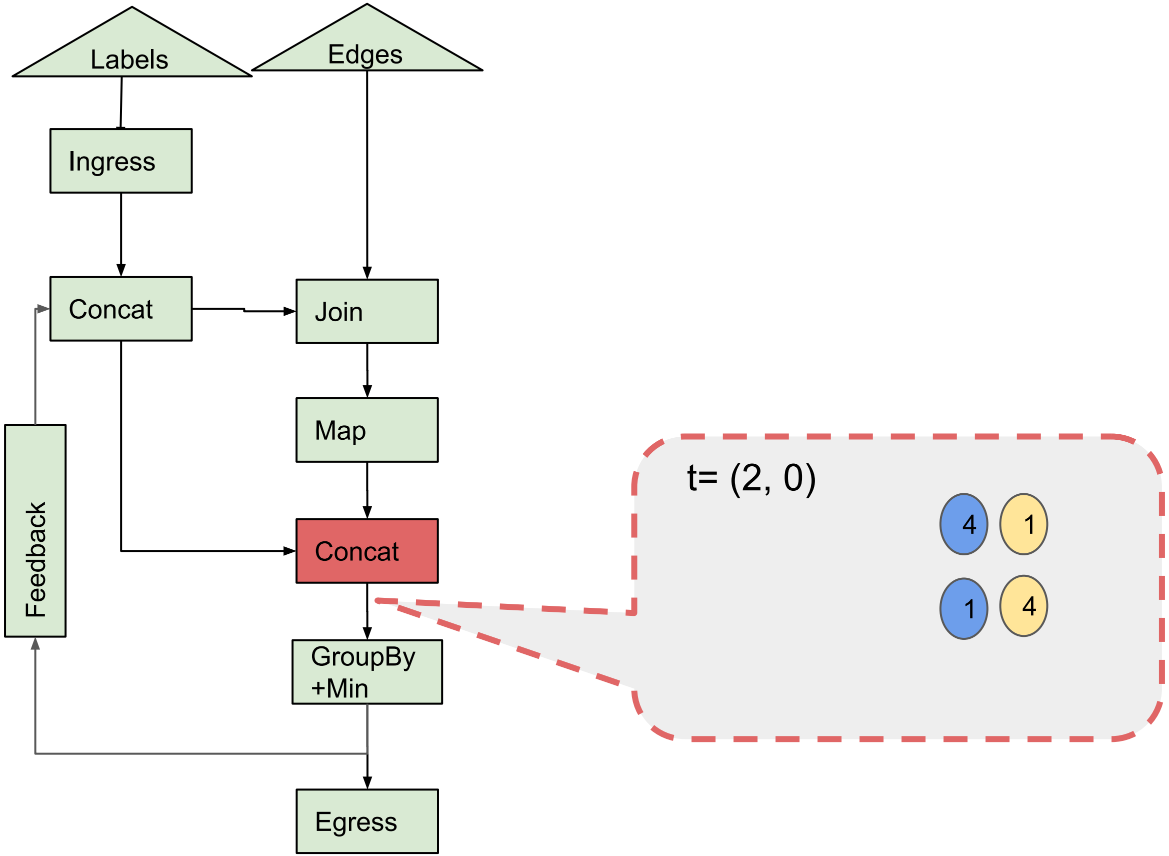
Changes to Connected Graph - II



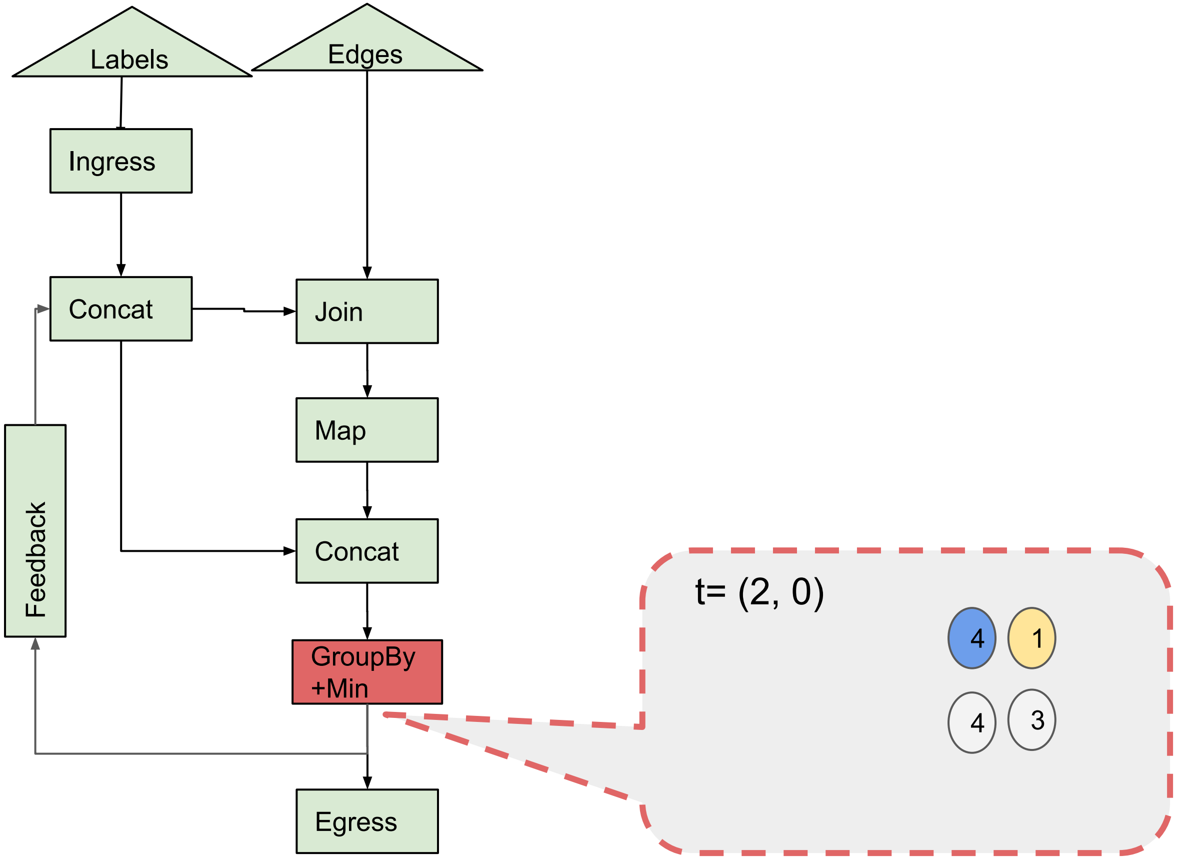
Changes to Connected Graph - II



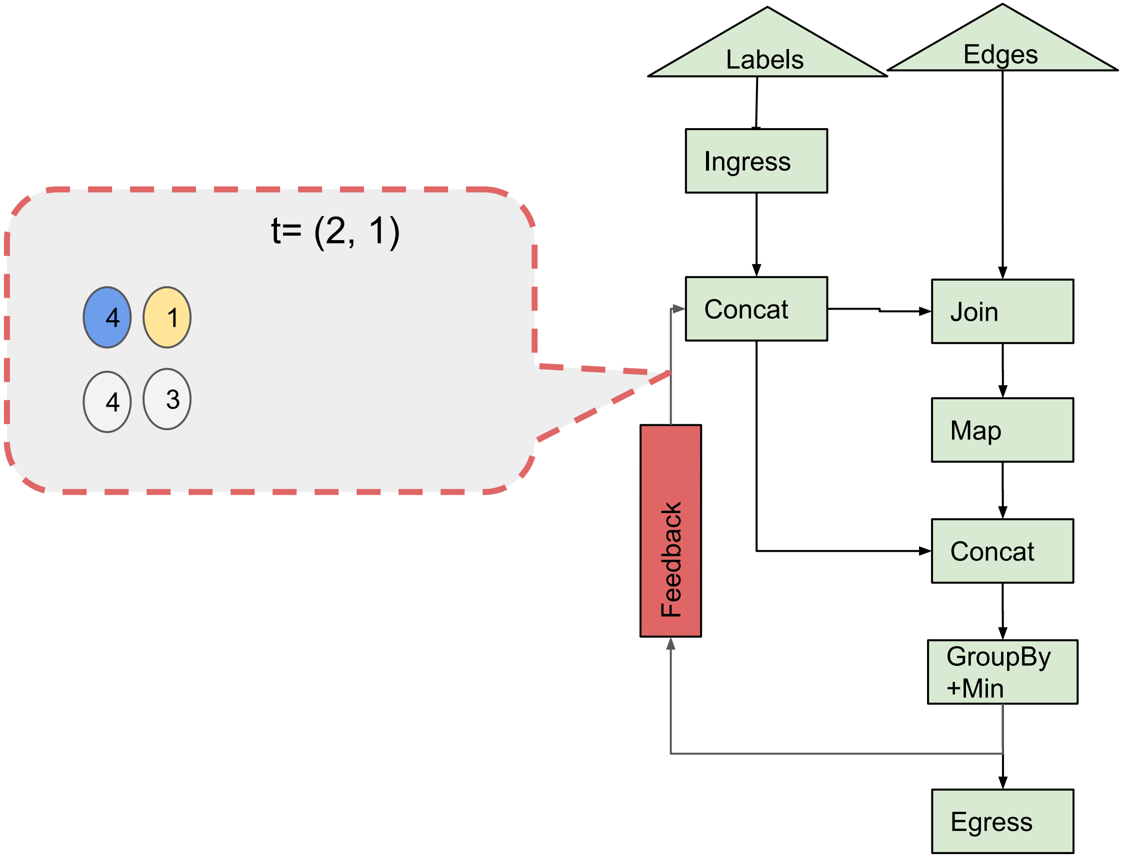
Changes to Connected Graph - II



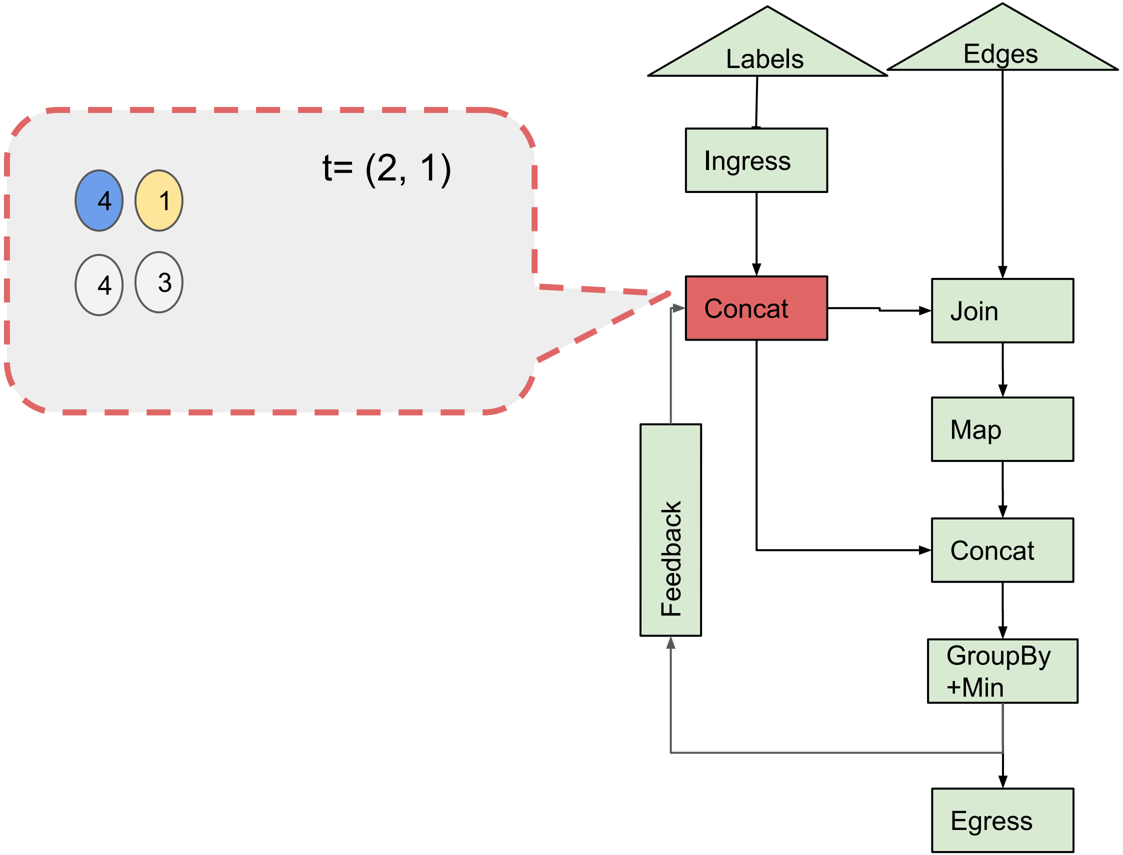
Changes to Connected Graph - II



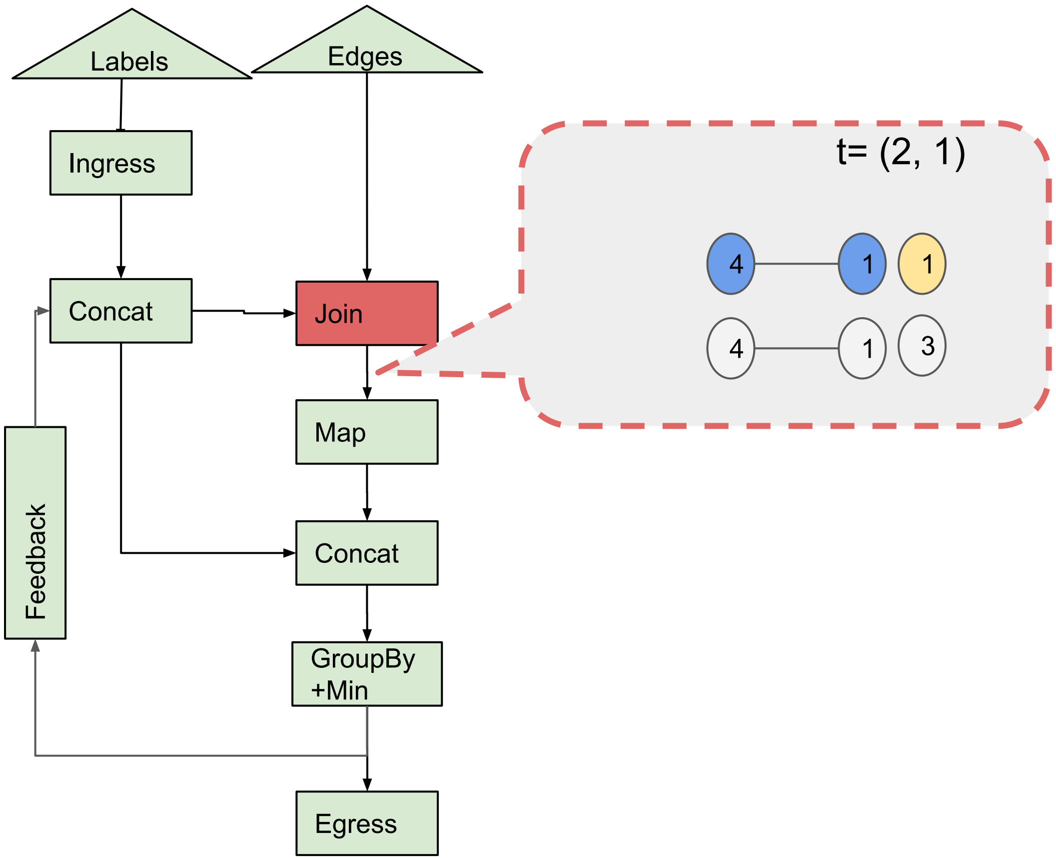
Changes to Connected Graph - II



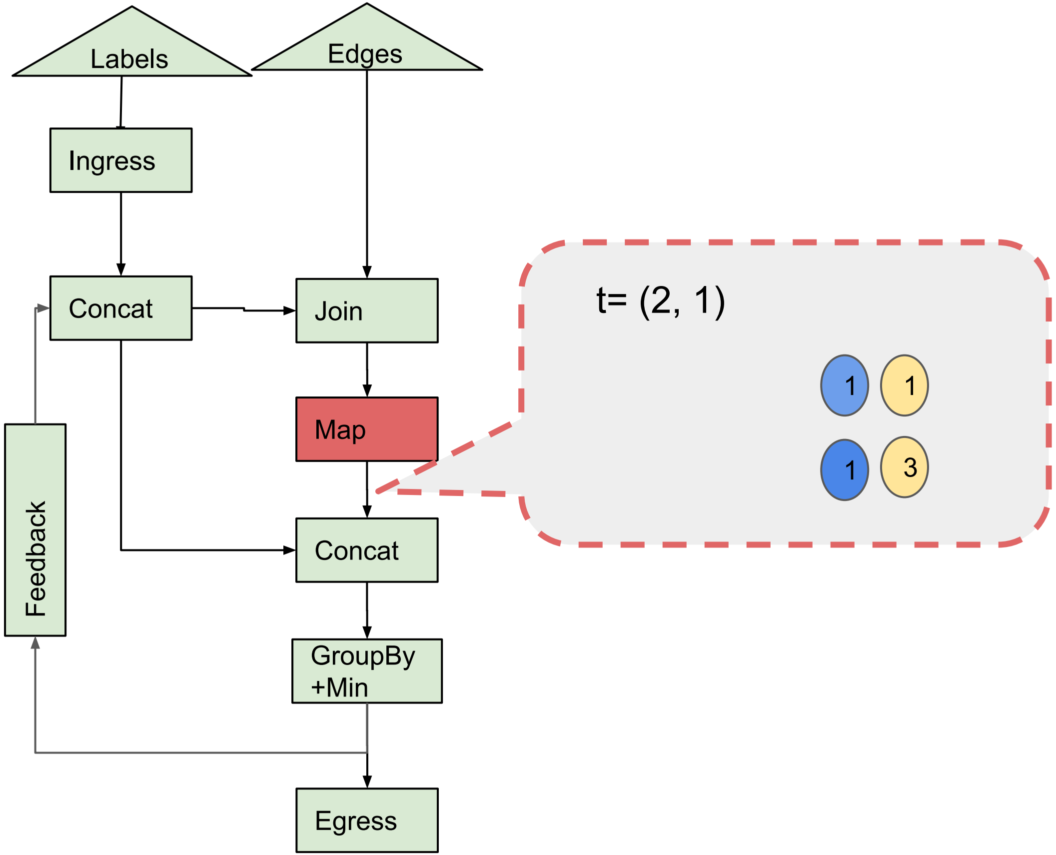
Changes to Connected Graph - II



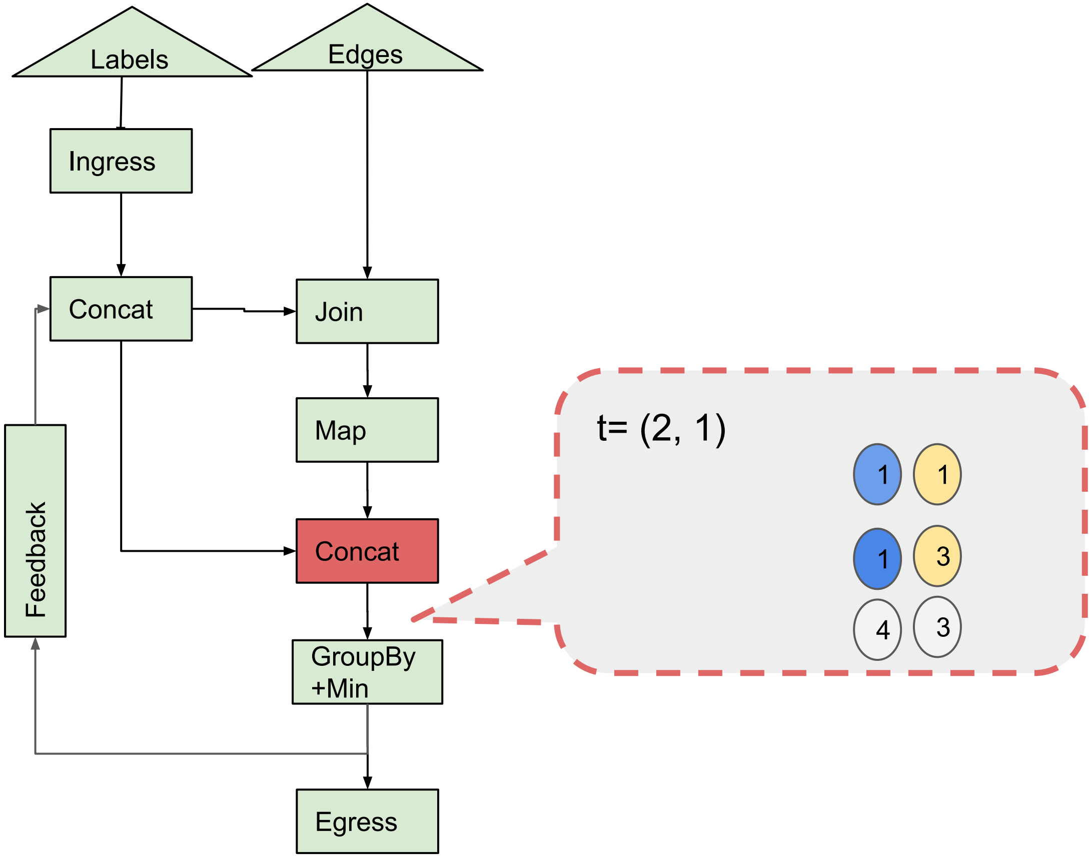
Changes to Connected Graph - II



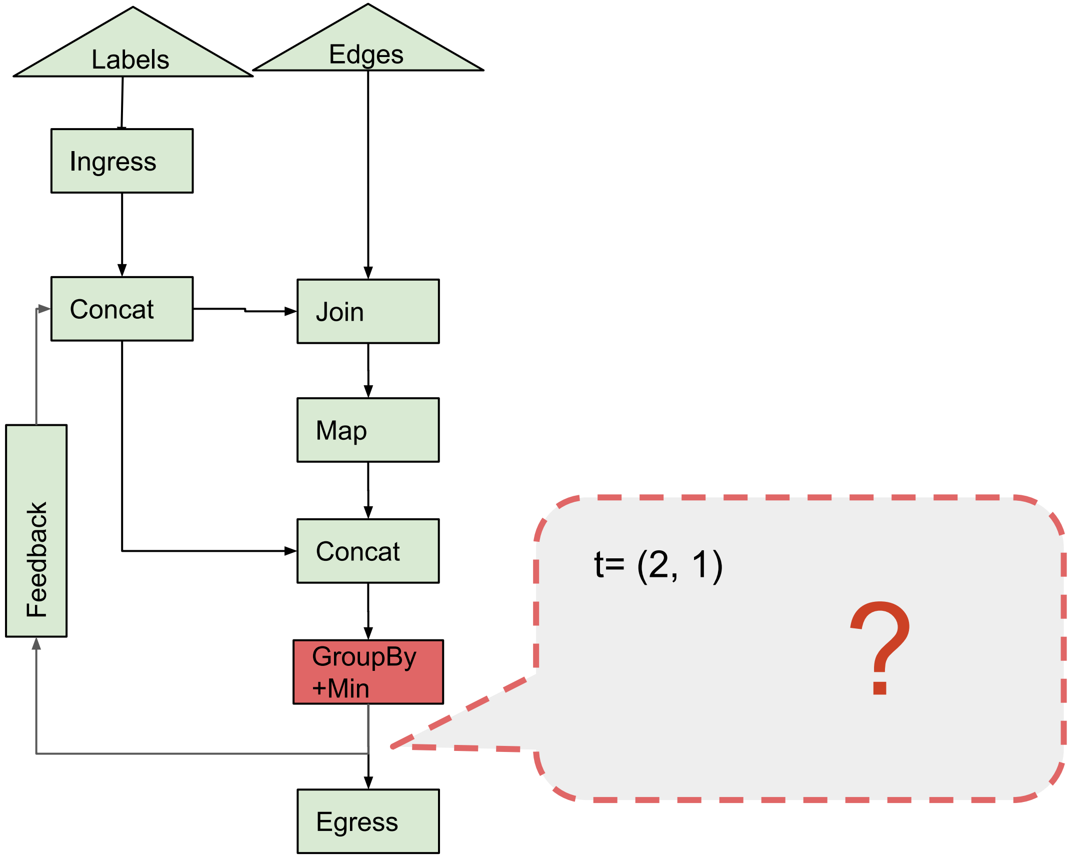
Changes to Connected Graph - II



Changes to Connected Graph - II



Changes to Connected Graph - II



Discussion

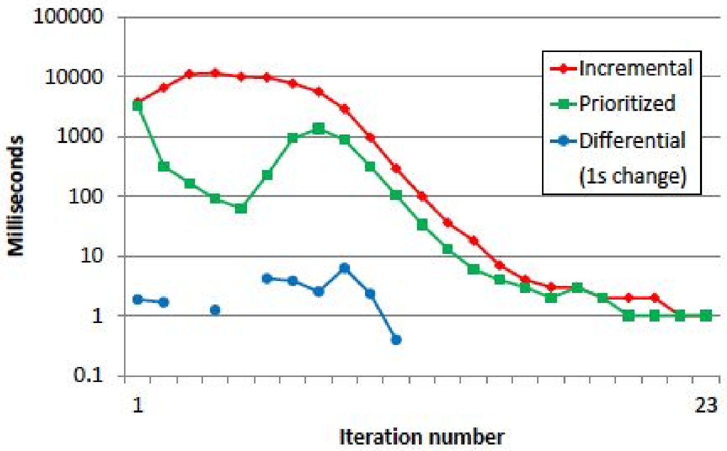
Tradeoffs in Iterative Systems

How do you deal with new data when you are iterating with old data?

- How much state to keep (Memory)
  - Do you keep all data in memory
  - What about intermediate calculations
  - Incremental View Maintenance
- How much work to do
  - Do everything from the beginning
  - Do work only in the nodes where new data came in

Iterative vs Differential Dataflow

- The flexibility of partial Ordering.
  - Iterative Ordering - $(i_1, j_1) \leq (i_2, j_2)$ iff $i_1 \leq i_2$ and $j_1 \leq j_2$.
  - Lexicographic Ordering - $(i_1, j_1) \leq (i_2, j_2)$ if $i_1 < i_2$ or $i_1 = i_2$ and $j_1 \leq j_2$.
  - Programmer can choose the partial ordering.
- Communication via Diffs
  - Only the diffs are sent around as messages
  - Nodes on both sides know the previous calculations



Discussion

- Is the memory for performance tradeoff justified?

Thank You