# Communicating Beam pipeline graphs

cmaden@google.com

June 2019

This is a proposal to standardize the mental model we use to communicate Beam pipeline concepts. We currently use different mental models throughout the documentation, which can be confusing to readers.

The purpose of this proposal is to create a canonical mental model. The canonical mental model should:

- Provide intuition to new users about basic pipelines
- Scale to accurately diagram longer, complicated pipelines
- Help readers debug issues at execution time
- Represent Beam pipelines generally (not a specific execution engine)

Readers should be able to use the mental model to quickly sketch representative pipelines, and contributors should be able to easily integrate it into the docs (it shouldn't require long captions or preambles).

Various mental models will be useful for different situations, so the purpose of this proposal is not to disregard these other approaches. There are many possible pipelines and a canonical mental model may not be completely accurate in every case. My goal is to create a model that clearly and accurately represents most pipelines.

Currently, Beam pipelines are represented as directed acyclic graphs. The most accurate pipeline representation is a bigraph. I suggest we simplify the bigraph representation and:

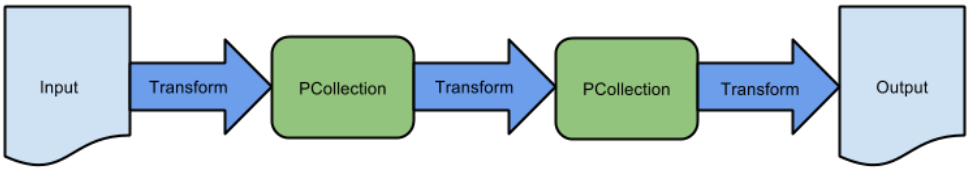
- Describe PTransforms and PCollections both as nodes in directed acyclic graphs.
- Describe PCollections functionally, as inputs and outputs for PTransforms.
- Diagram PTransforms and PCollections as parts of the same unit. Each node includes a description of the PTransform *and* the output PCollection.

**See Beam pipeline graphs for example diagrams based off of this pipeline representation.**

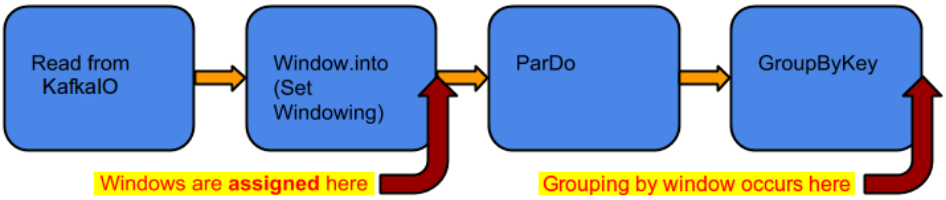
The mental model that these graphs emphasize is that PTransforms manipulate PCollections, and pipelines direct PCollections through PTransforms to process data.

## Current state of the docs

We currently use two different directed acyclic graphs to model pipeline diagrams. One is the "functional programming" model. In this model, PCollections are represented as nodes and PTransforms are edges.




The other is the "Dataflow graph" model. In this model, PTransforms are represented as nodes and PCollections are edges.




We can also conceptually describe PTransform or PCollection objects as nodes or edges in order to emphasize their properties. We currently use both mental models . In the pydoc, we describe PTransforms as nodes to emphasize that they accept PValues as inputs and emit PValues as outputs. But in some cases, we also describe PValues as nodes.

## Beam pipeline bigraph

The most accurate conceptual pipeline representation is a bigraph. The two sets of nodes in the bigraph are PTransforms and PCollections, and the expanded graph follows these rules:

- Each PTransform has any number of outgoing and incoming edges
- Each PCollection has any number of outgoing edges but only one incoming edge

Here's an example of a diagram based on this pipeline representation. This pipeline representation is accurate, but:

- It's hard to use it to create simple pipelines because there are a lot of nodes to keep track of
- It doesn't emphasize the functional behavior of PCollections (that pipelines direct PCollections through PTransforms to process data)

These issues can make it hard for new users to conceptualize this pipeline representation and debug issues at execution time. To simplify the bigraph, I suggest that we group PTransforms and output PCollections into the same unit.

## PTransforms as nodes

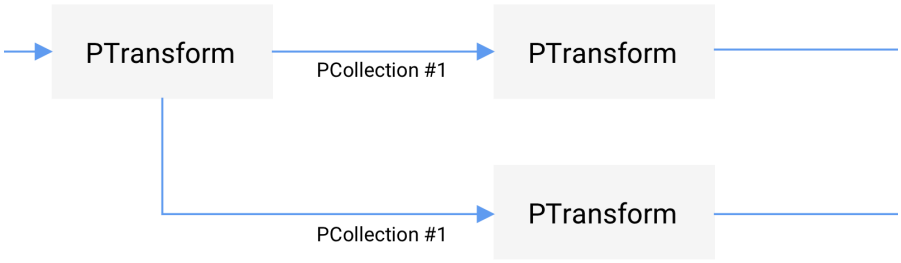
I suggest that we describe PTransforms as nodes throughout the docs. I think that representing PTransforms as nodes is the most accurate. At first, it's less intuitive than representing them as

edges, but representing them as nodes is makes it easier to build complicated pipelines because:

- It more accurately represents how to interact with PCollection objects. PCollections don't "contain" values, so you can't iterate over a PCollection to read or "take" the values outside of a pipeline (like you can with the collections module). Instead, you manipulate PCollections with PTransforms, and pipelines pass PCollections through a series of PTransforms.
- It's better at representing unbounded data. Data flows through the pipeline and is continuously and successively manipulated by PTransforms.

## PTransforms and PCollections in the same unit

Although representing PTransforms as nodes is accurate, it can sometimes be inaccurate to represent PCollections as edges. For instance, a branching pipelines with two outgoing edges is represented by a PTransform that outputs two copies of a PCollection. In this example, a single PCollection is represented by multiple edges.




I suggest that we don't describe PCollections as edges and instead describe them functionally, as inputs and outputs for PTransforms. A Beam pipeline graph would therefore group PTransforms and output PCollections in the same unit. Each node would represent a PTransform and output PCollection, and each PTransform and PCollection would be labeled and described.

The following graph is an example diagram for this mental model. See the next section for more diagrams.


In this mental model, pipelines are represented via PTransforms, input PCollections, output PCollections. All PTransforms are nodes. PTransforms are described with their output

PCollections, and each output PCollection is passed to (or “flows into”) subsequent PTransforms.

# Diagramming branching pipelines

Diagrams of Beam pipeline graphs emphasize a mental model where:

- PTransforms manipulate PCollections
- Pipelines direct PCollections through PTransforms to process data.

The following table links to examples of branching pipeline diagrams that emphasize this mental model:

| Branching pipeline | Representation of mental model | Sample diagram |
|---|---|---|
| PTransforms with multiple output PCollections | The PTransform block (node) has multiple outgoing arrows (edges) | Link |
| PTransforms with multiple input PCollections | The PTransform block (node) has multiple incoming arrows (edges) | Link |
| Multiple PTransforms with the same input PCollection | The PTransform block (node) has multiple incoming arrows (edges), each from different PCollection blocks (nodes) | Link |

## Representing joining transforms

Joining transforms are diagrammed as PTransforms with more incoming edges than outgoing edges. To represent the input PCollections, the tail of each incoming edge connects to a PTransform node with a distinct output PCollection. Here’s a sample diagram.

## Representing side inputs

Side inputs are diagrammed as distinct PCollections nodes with outgoing arrows that direct to the target PTransform. Although side inputs are not themselves PCollections (but rather runner materializations of PCollections), I think that this simplified representation is more useful for new users and simple pipelines. Here’s an example.

## Representing the source database

I suggest that we don’t diagram the source database and instead start pipelines at the first read transform. It is neither a PTransform nor a PCollection (and shouldn’t be conceptualized as such), so removing it will simplify diagrams.

### Representing composite transforms

One way to represent composite transforms is by expanding the graph and diagramming each component part. Showing a composite transform's expanded structure is useful because you can diagram the component parts once and later diagram them as a single node. Composite parts can be diagrammed like other pipeline components (each nested PTransform is its own node) and then encapsulated by a distinguishing visual feature. Incoming and outgoing edges would connect directly to the nested nodes.

Another way to represent composite transforms is by collapsing all component parts into one node collapsing all component parts into one node. Like other PTransforms in this mental model, composite transforms can be represented with any number of input and output PCollections. If the composite transform has more input PCollections than output PCollections, it's diagrammed as a joining transform.

Both ways to represent composite transforms are useful and should be integrated into the documentation together. Some documentation may want to compare the collapsed and expanded versions of the same composite transform. We can distinguish between composite transforms by color-coding them. I recommend that we use colors instead of numbers because colors are unordered.

## Other solutions

TensorFlow and Cloud Dataflow also use directed acyclic graphs to represent data and data processing functions.

TensorFlow computation graphs represent both data objects and data processing functions as nodes. However, unlike Beam pipelines, all TensorFlow nodes have only one outgoing edge. In Beam, representing both PTransforms and PCollections as nodes would be similar to representing the PTransforms as nodes and PCollections as edges. An output PCollection might be an input to multiple PTransforms, so the diagram might incorrectly imply that the pipeline is processing multiple copies of the same PCollection.

Dataflow execution graphs represent PTransforms as nodes. Each node is associated with metadata that pops open when you click on it. The metadata includes the input and output PCollections. This is similar to the suggested diagrams because the PCollections are represented as neither nodes nor edges. Instead, PCollections are described functionally, as only the inputs and outputs to PTransforms.

Lastly, I recommend that, no matter the diagram style, we prime the reader by explicitly describing how the diagrams work and what they represent (including the specific ways in which

they simplify the bigraph representation). I think this is a good idea that we can incorporate with the suggested solution.