Scaling Autonomous Driving with Apache Beam

Sayat Satybaldiyev
Senior ML System Engineer II

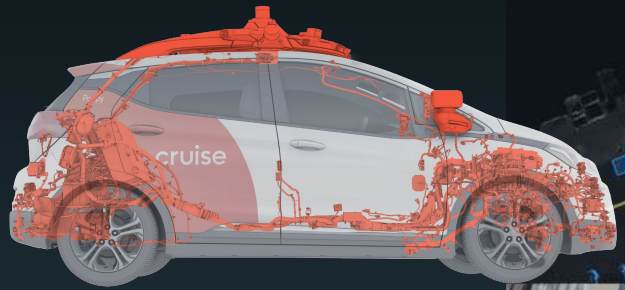
Arwin Tio
Senior ML System Engineer II

BEAM SUMMIT

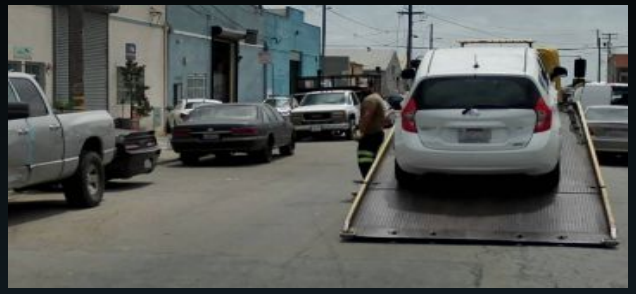
September 4-5, 2024

Sunnyvale, CA. USA

Introduction to Cruise



A normal day in the life of a Cruise AV



Introduction to Terra Platform

How Terra was developed to address Cruise's data processing challenges.

Terra’s data processing platform on Apache Beam.

The flexibility and scalability of Apache Beam for handling Cruise’s data needs.

Why Terra?

Hidden technical debt in Machine learning systems Sculley et.

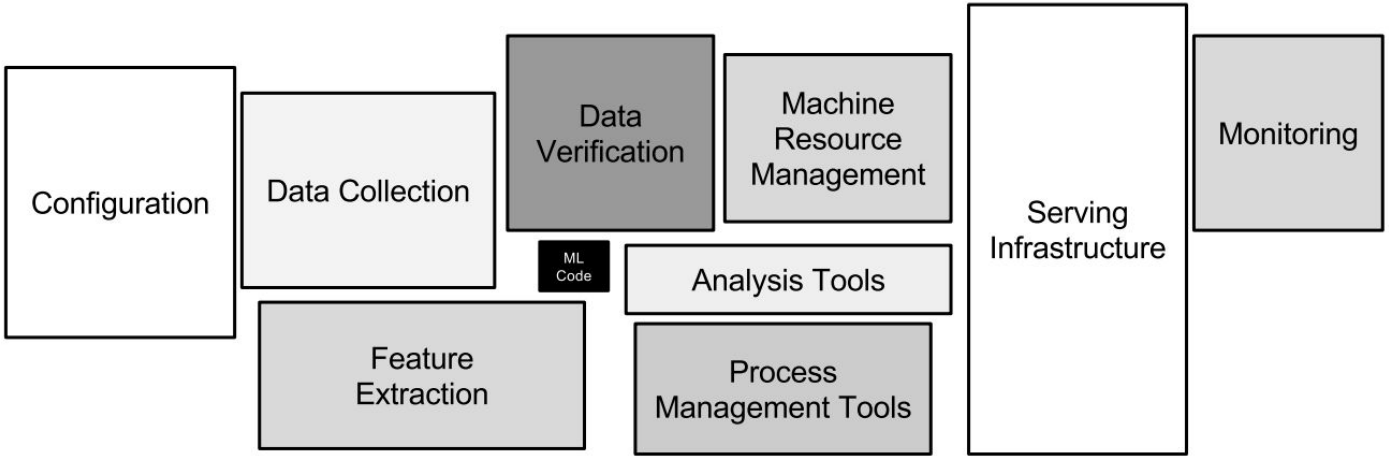


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

What Terra provides?

IO connectors to Cruise data sources

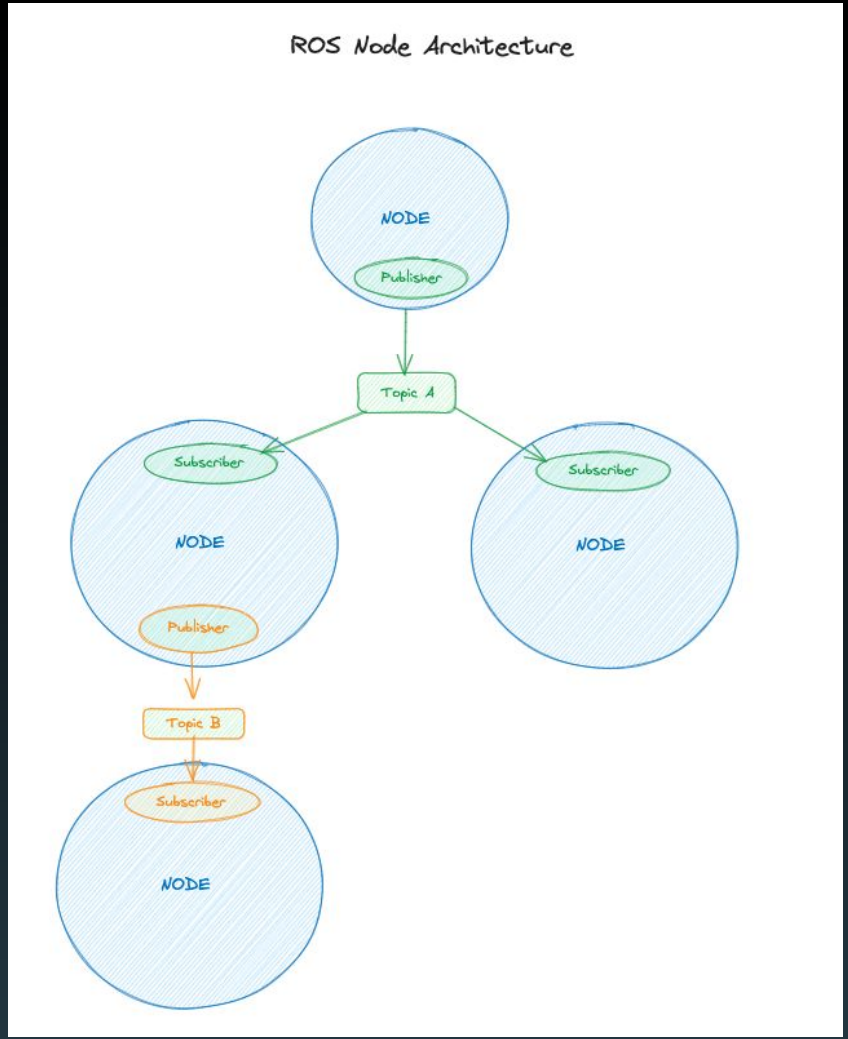
Common Transforms and ML specific Ops

Dependency management

Optimization etc

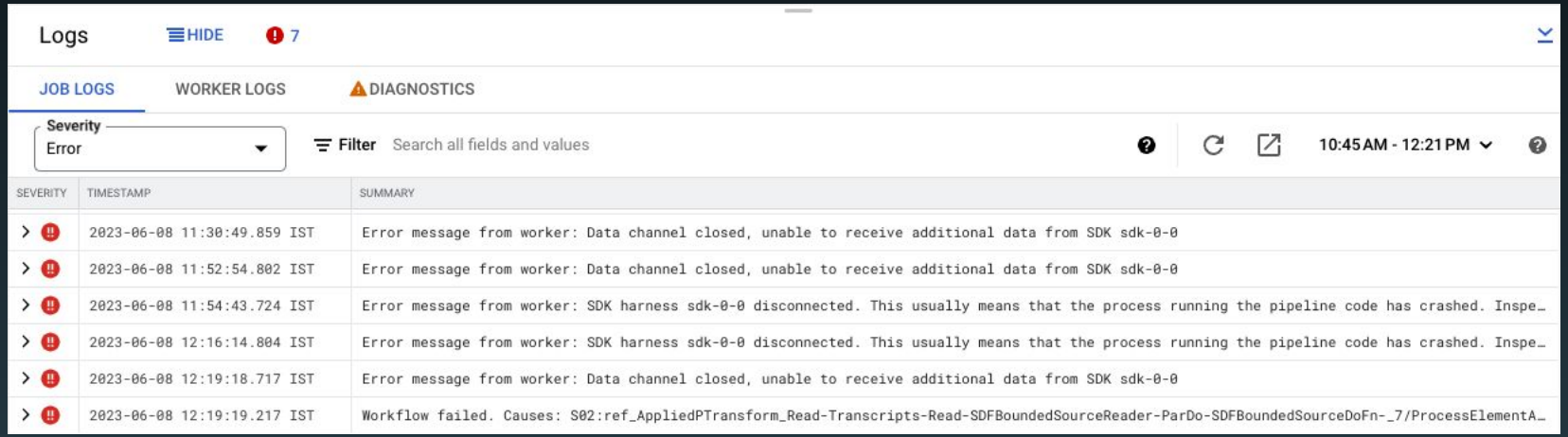
AV stack replay in Apache Beam

- We use the Beam Python SDK, but sometimes our ML feature generation requires (partially) rerunning our AV stack on Apache Beam, which means C++
- Our AV Stack is built on the ROS (Robot Operating System) framework
- To oversimplify, our ROS AV stack consists of modular nodes that send asynchronous messages to each other
- Concretely, this means running one of those ROS nodes in a Beam DoFn with C++ extension pybindings



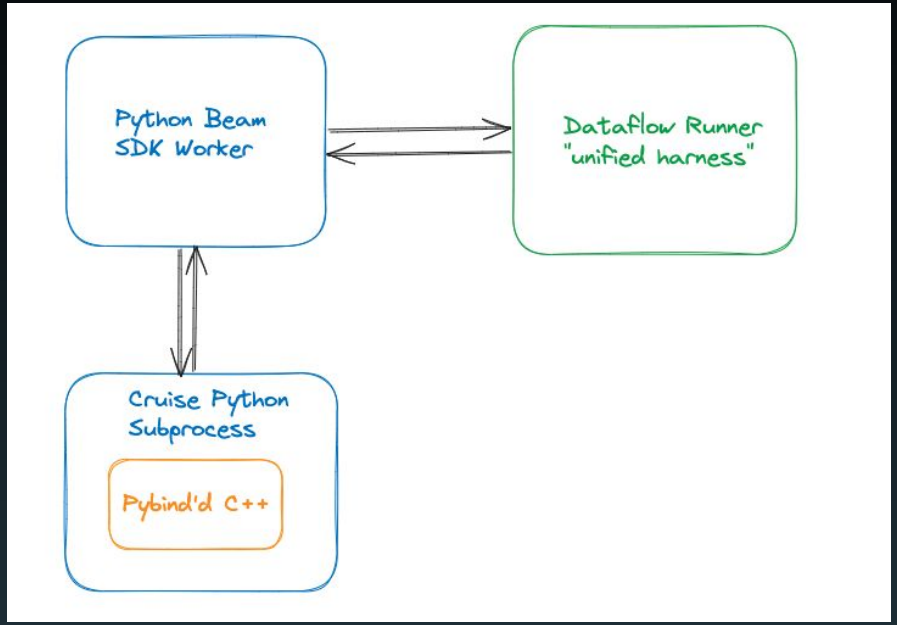
AV stack replay in Beam – C++ Sandboxing

- Another challenge: abrupt process crashes from C++ (i.e. segfaults)
- C++ code can crash in ways that bypass Beam Python SDK's exception handling, resulting in the dreaded "SDK harness disconnected" or "Timed out waiting for an update from the worker" error!
- Moreover, logs from C++ go straight to stdout/stderr and doesn't go through the Fn Logging API, which are harder to find in the Dataflow UI



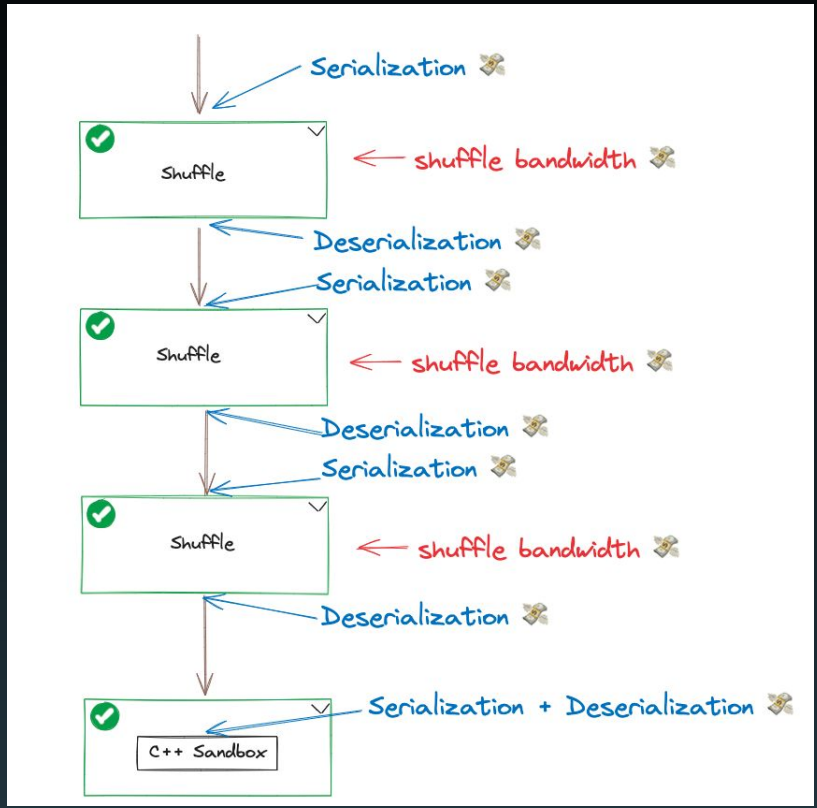
AV stack replay in Beam – C++ Sandboxing

- We solve this problem by sandboxing C++ execution inside a subprocess
- When C++ code crashes, only the subprocess terminates, not the main Beam SDK Python process
- Also: subprocess fork() causes problems for us. We use spawn(). See https://github.com/grpc/grpc/blob/master/doc/fork_support.md



Shuffle and SerDe costs

- Note that a running in subprocess adds overheads, since data needs to be serialized and deserialized as it crosses the process boundary.
- Since we have large input data consisting of sensor and trajectory data, SerDe overheads account for a large part of our processing costs, sometimes more than the actual feature computation!
- Our Beam pipelines usually has several shuffles - each shuffle boundary also introduces a SerDe cycle. For example, a pipeline with 3 shuffles will cause your data to be serialized and deserialized 3 times, in addition to the subprocess boundary.
- In addition, shuffle bandwidth costs are also significant

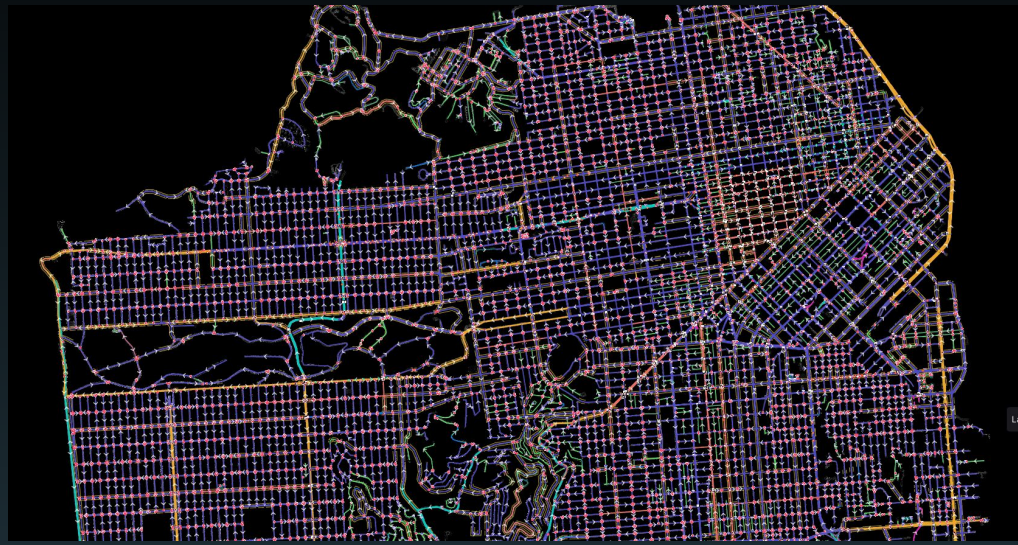


Shuffle and SerDe costs – some solutions

- **Deserialization pushdown** - by keeping input data serialized as raw bytes until the last possible moment (i.e. until we need to manipulate it). If the data is stored as serialized bytes, Beam Coders don't need to do anything (i.e. pickle) as the data passes through shuffle and process boundaries.
- **Projection pushdown** - don't expand the dimensionality of the data until the last possible step where you need it. For example, we enrich our scenes with ground truth object labels only after all shuffle steps. This avoids the cost of shuffle bandwidth and serialization of that extra data.
- **Compression** - we compress all of our data using zstd before a shuffle. Your mileage may vary as this is a bandwidth vs compute trade off but for us it is a significant net positive.

AV stack replay in Beam – Map loading challenge I

- Running the AV Stack in Beam introduces some interesting challenges.
- For example, the AV will load the entire map of the city it's currently in, as it has strong spatial locality and does not jump between different cities.
- Our feature generation pipelines however, processes scenes from different cities and spanning many dates over the past years. Since maps change over time, this means loading different versions of the map.



Semantic Map of San Francisco

AV stack replay in Beam – challenge II Map loading

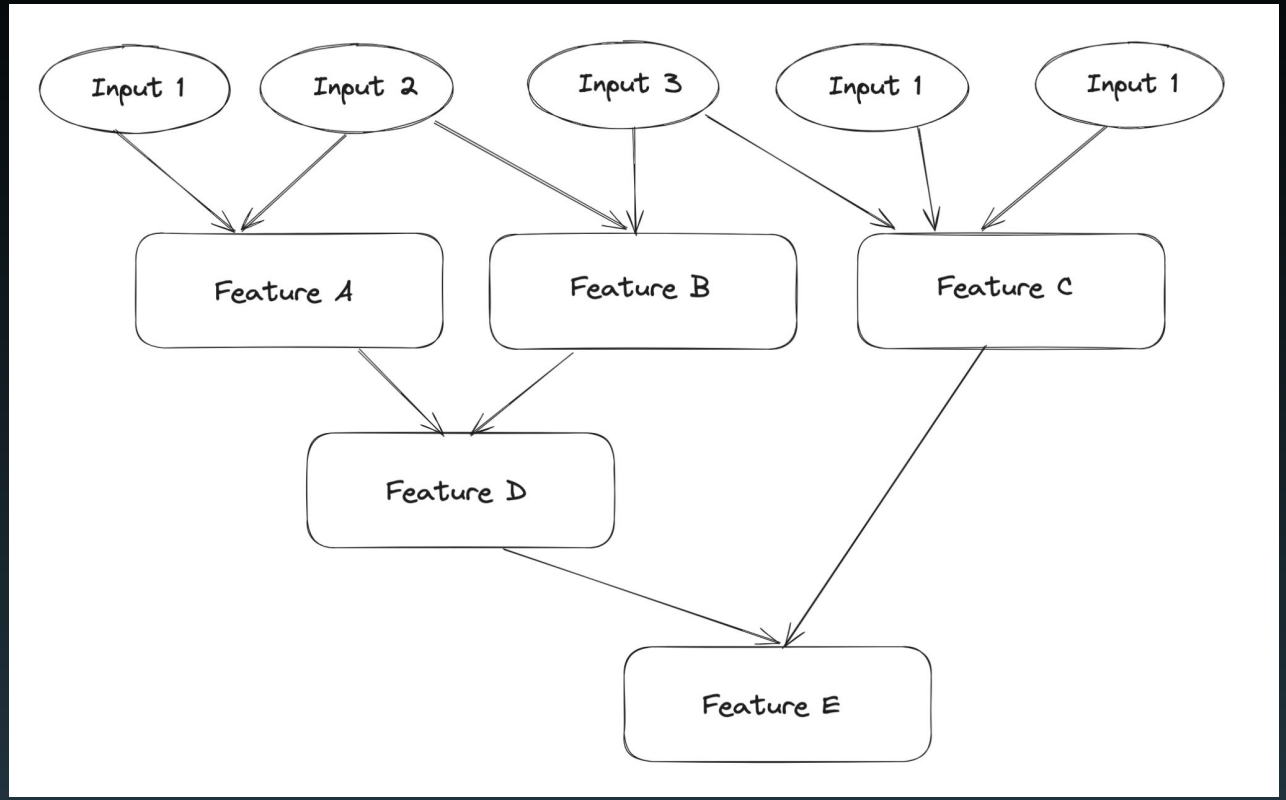
- Map loading is an expensive operation, as it downloads a large amount of data and has to build spatial indexes, so we cannot afford to do it for every element in the PCollection.
- So, we need to group by our data by city/map version in order to amortize the cost of map loading over many different elements.
- In practice, the snippet below is not enough, as it causes data skew and limits the parallelism of scenes from each city/map version. But that is the general idea.

```
pcoll
| Map(lambda scene: ((scene["city"], scene["map_version"]), scene))
| "Partition by map" >> GroupByKey()
| "Process" >> ParDo(Process())
```

Optimizing Feature Development

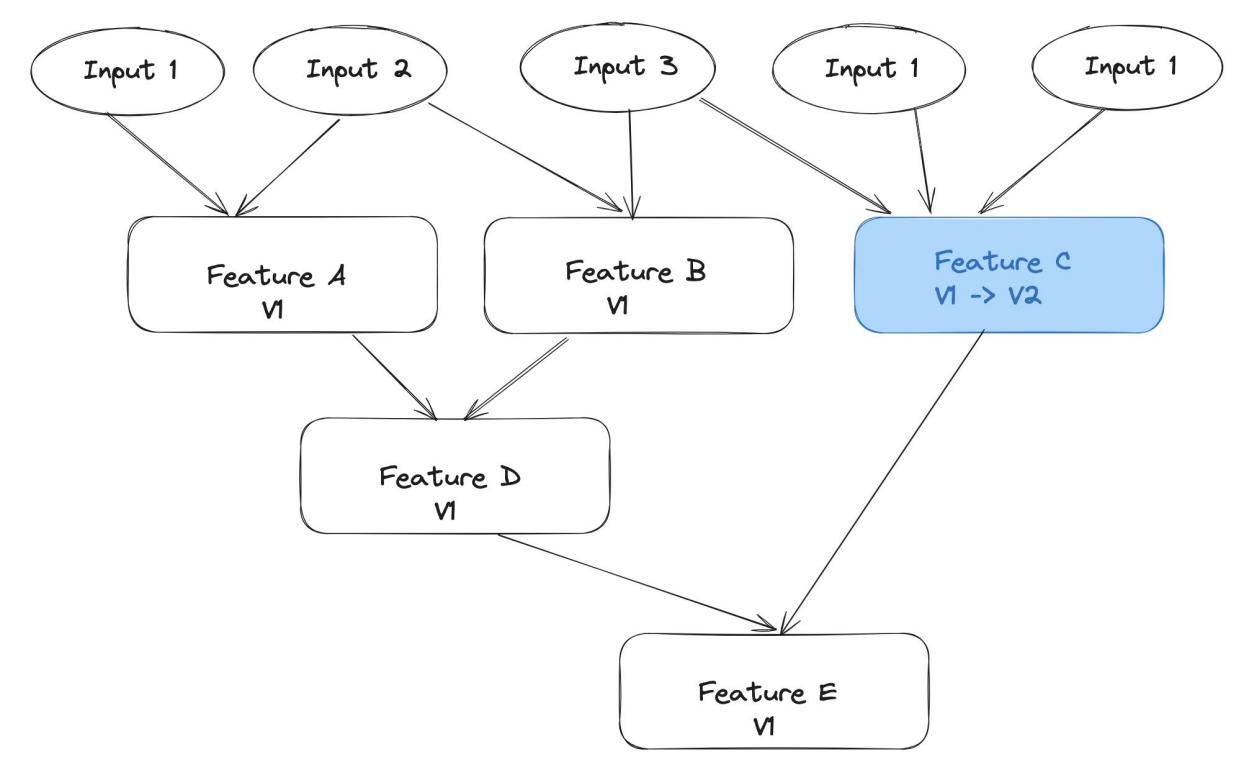
- Most "business logic" and processing power of our Apache Beam pipelines is spent on feature computation
- Our ML engineers iterate on feature code, they add or modify existing features as they perform experiments
- Because of this, it's important to make feature iteration cheap and fast
- We solve this with "feature caching"

Feature Graph



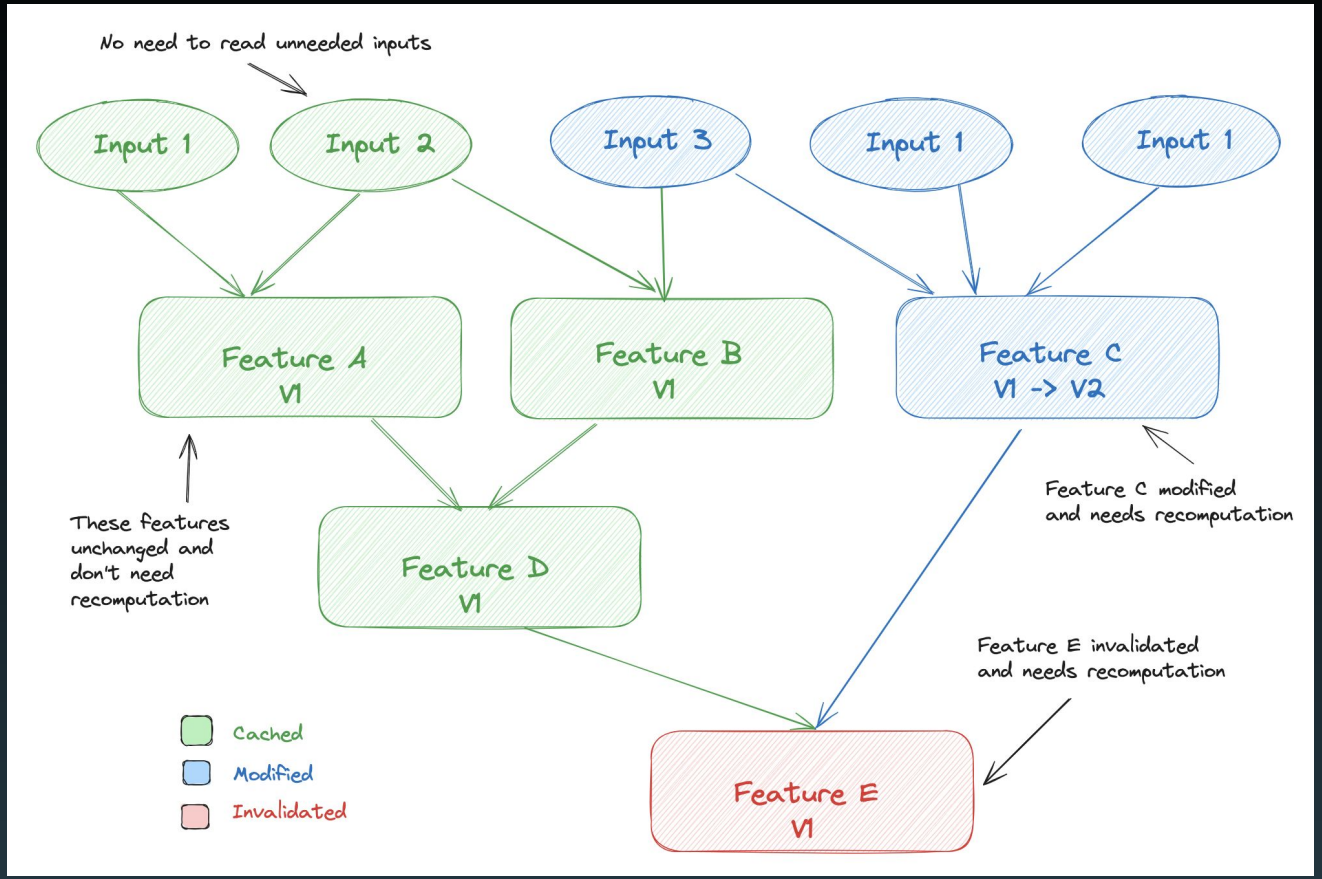
- Our features can be organized into a directed graph
- Features can have raw inputs or other features as dependencies

Feature Graph – Recomputation



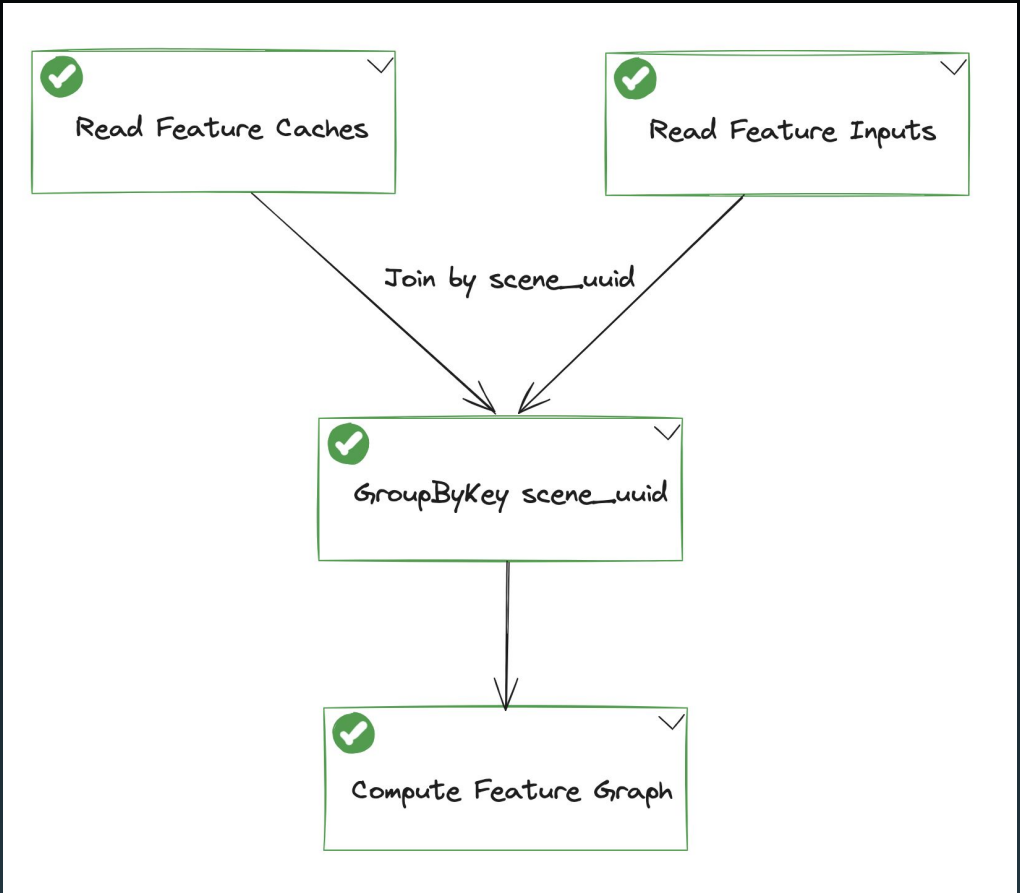
- Let's say that an MLE wants to modify feature C (in blue) - with a naive implementation we would need to recompute the entire feature graph
- However many of the feature nodes (A, B, D) will produce the exact same results

Feature Graph – Caching



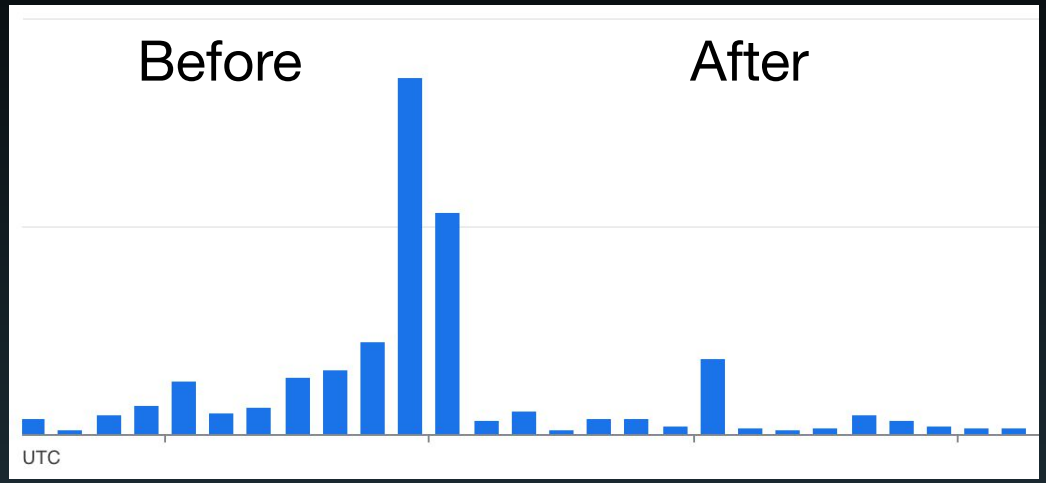
- We can walk through the graph and determine which features are modified, invalidated, or unaffected.
- The features that are unaffected can be cached and reused!
- We can also skip reading raw inputs that are unneeded - this ties to the earlier theme of minimizing shuffle and SerDE costs.

Feature Caching in Beam



- We key our inputs and feature cache by "scene uuid", which we join with a GroupByKey
- Feature computation only has to recompute invalidated features by checking the feature versions (i.e. V1 -> V2)
- Note: caching is a **storage vs compute tradeoff** - some of our features produces outputs that are so large that is cheaper for us to recompute than pay for storage - measure your storage costs!

BigQuery Cache



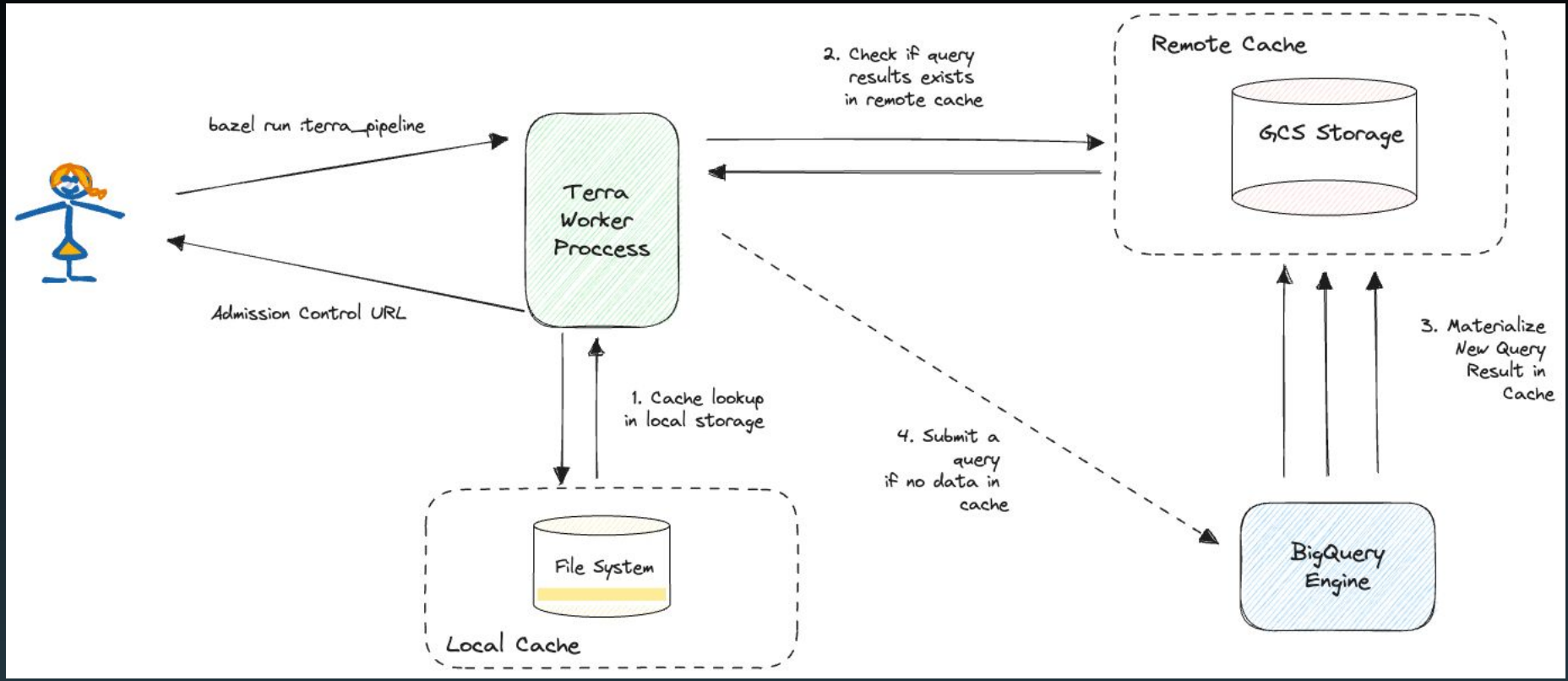
BigQuery Job Count

Speedup Local Iterations

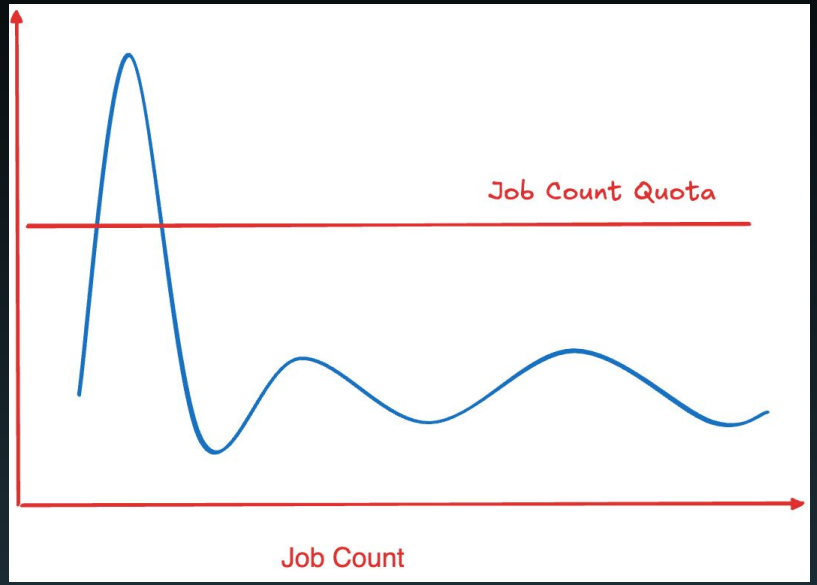
Avoid BQ Slot Contentions Issues

Saves: 60 hours+/week

BigQuery Cache

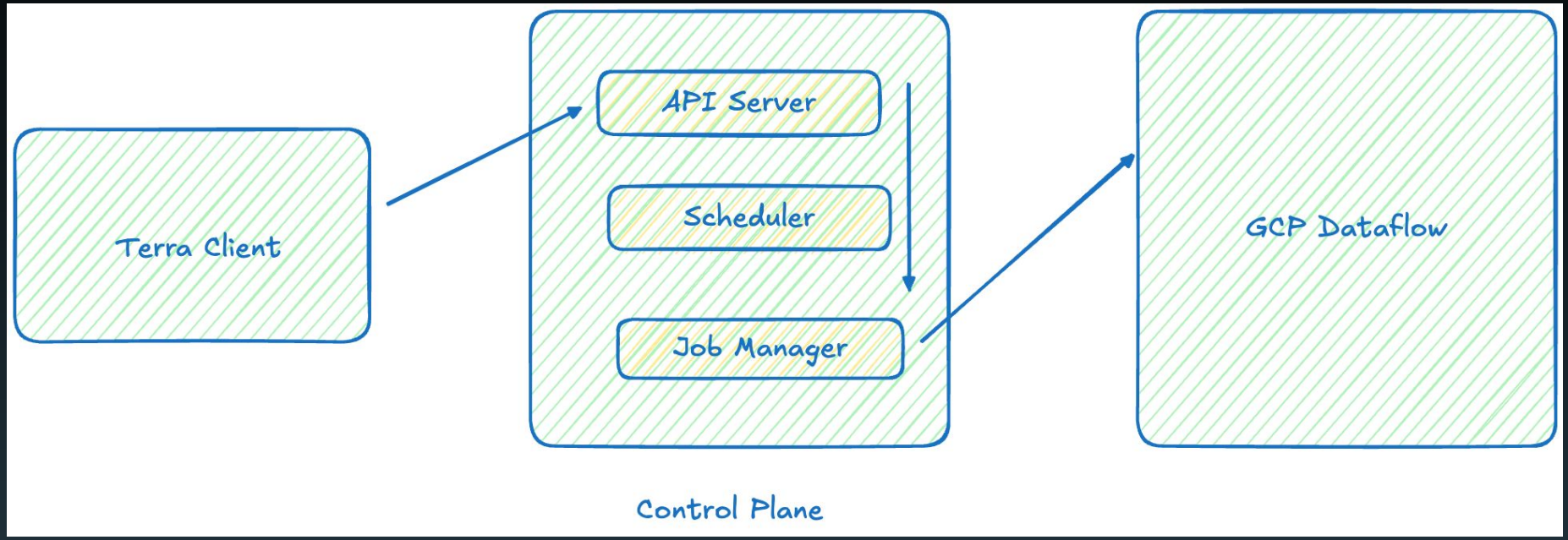


Control Plane



- Terra’s control plane for job submission and execution.
- Scheduling policies to ensure fair resource allocation and capacity utilization

Control Plane



Thank you!

Do you have any questions?

https://www.linkedin.com/in/satybald/
https://www.linkedin.com/in/arwin-tio-83b69557/