Human Centric

Machine Learning Infrastructure

@

NETFLIX

Ville Tuulos

QCon SF, November 2018

Caveman Cupcakes

Caveman Cupcakes

Meet Alex, a new chief data scientist at Caveman Cupcakes

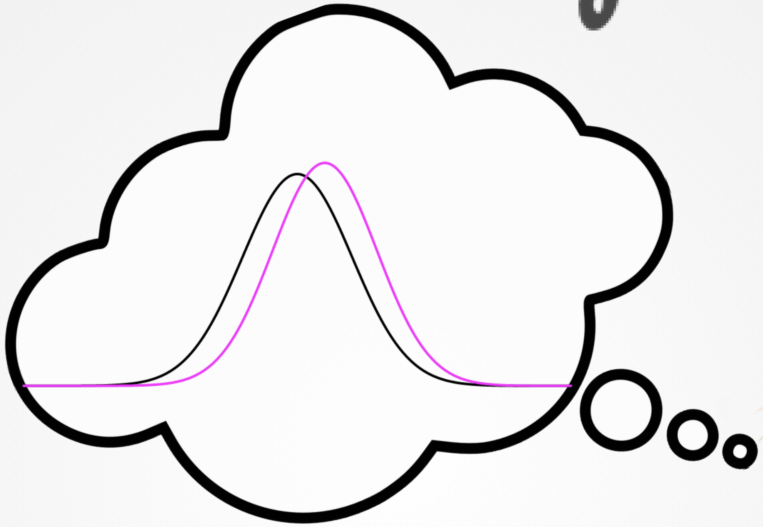
Caveman Cupcakes

We need a dynamic pricing model.

Caveman Cupcakes

Caveman Cupcakes

Caveman Cupcakes

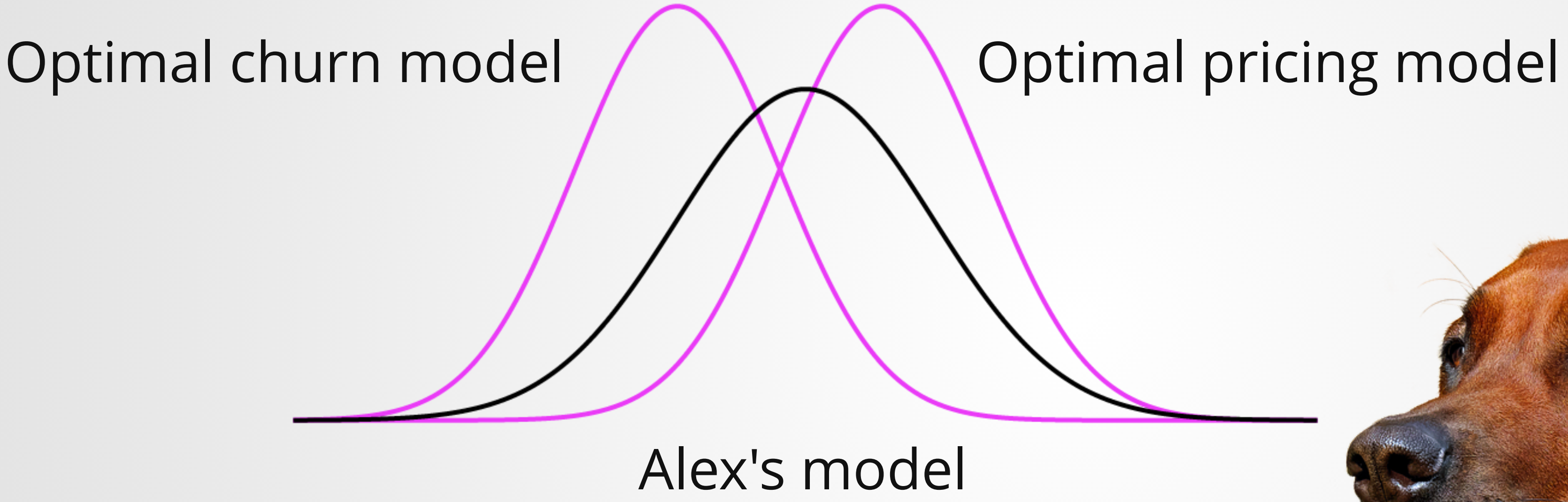


```
#include "main.h"

template<int Alignment,typename VectorType> void map_class_vector(const VectorType&
{
  typedef typename VectorType::Index Index;
  typedef typename VectorType::Scalar Scalar;

  Index size = m.size();

  VectorType v = VectorType::Random(size);

  Index arraysize = 3*size;

  Scalar* a_array = internal::aligned_new<Scalar>(arraysize+1);
  Scalar* array = a_array;
  if(Alignment!=Aligned)
    array = (Scalar*)(internal::IntPtr(a_array) + (internal::packet_traits<Scalar>::AlignedOnScalar?sizeof(Scalar):sizeof(type

  {
    Map<VectorType, Alignment, InnerStride<3> > map(array, size);
    map = v;
    for(int i = 0; i < size; ++i)
    {
      VERIFY(array[3*i] == v[i]);
      VERIFY(map[i] == v[i]);
    }
  }
```

Caveman Cupcakes

Caveman Cupcakes

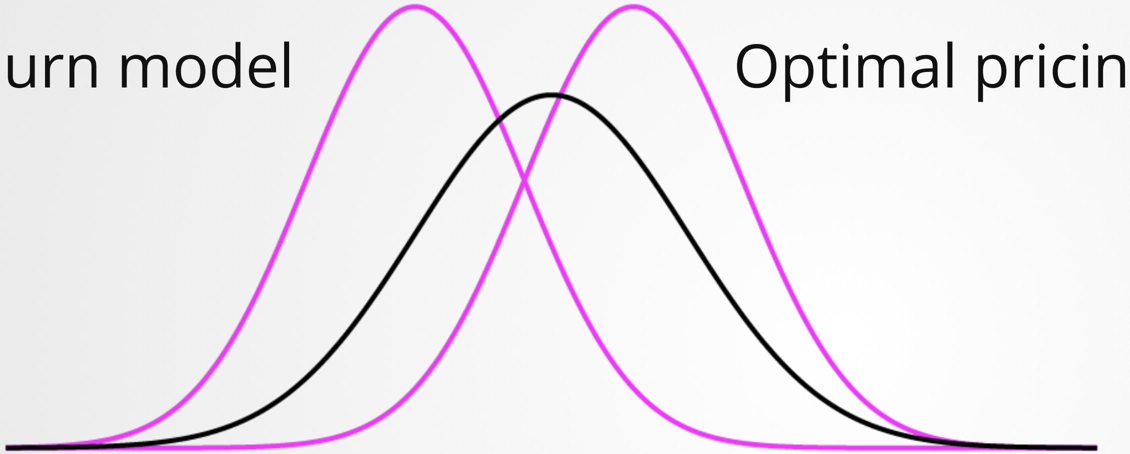
Caveman Cupcakes



Caveman Cupcakes

Optimal churn model

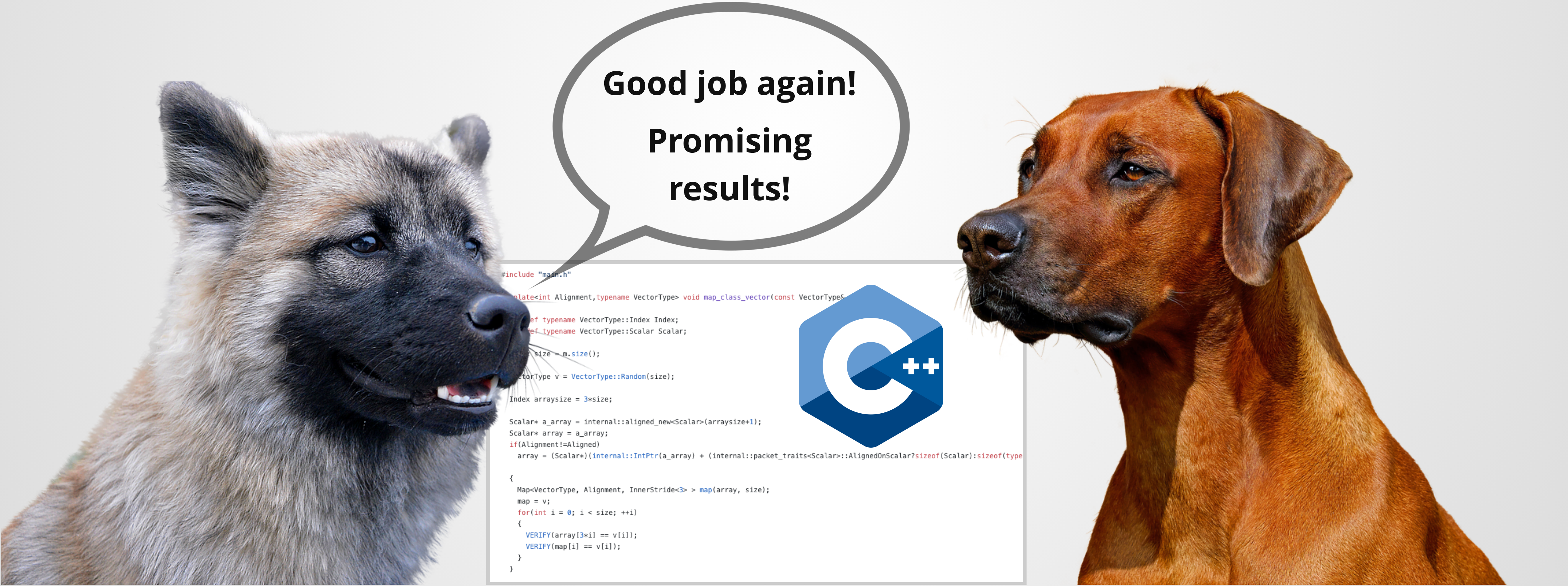
Optimal pricing model



Alex's model

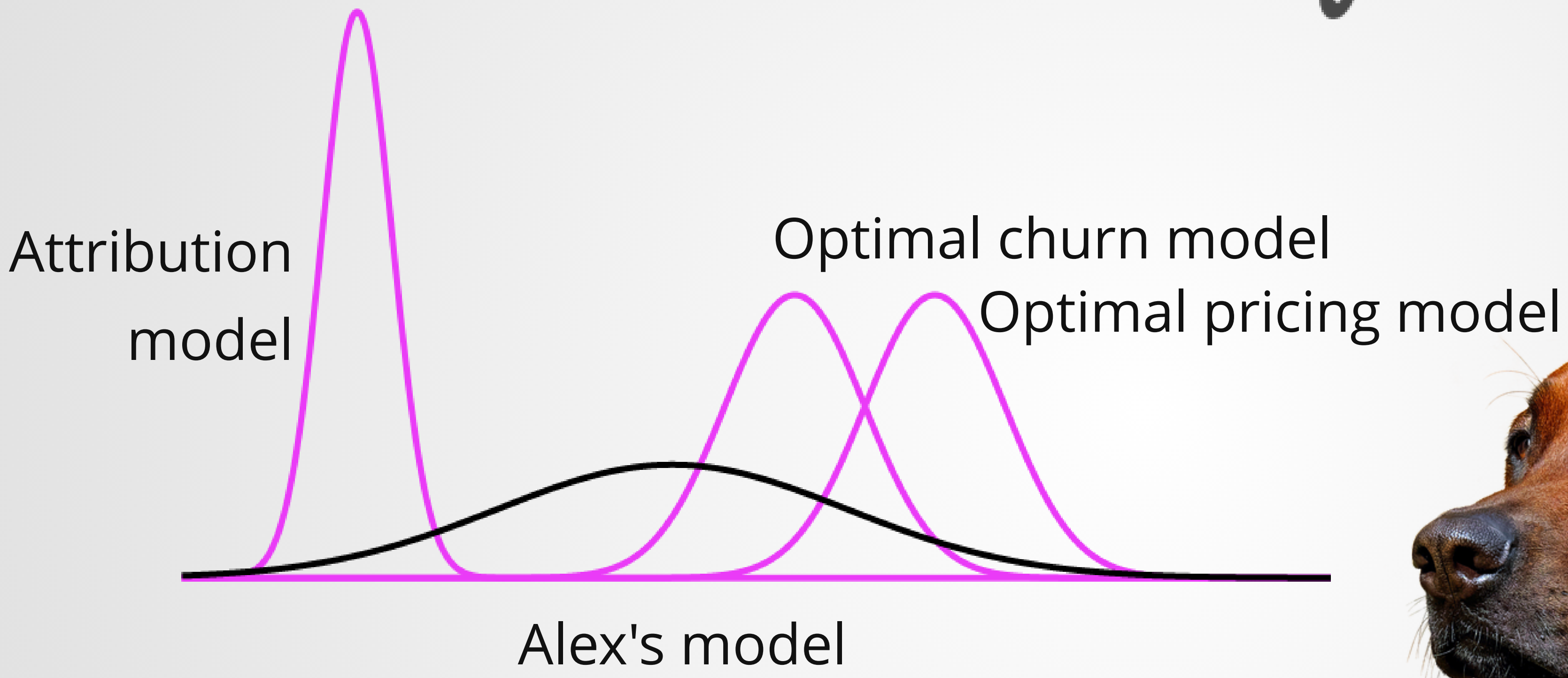
```
#include "main.h"

template<int Alignment,typename VectorType> void map_class_vector(const VectorType&
{
  typedef typename VectorType::Index Index;
  typedef typename VectorType::Scalar Scalar;

  Index size = m.size();

  VectorType v = VectorType::Random(size);

  Index arraysize = 3*size;

  Scalar* a_array = internal::aligned_new<Scalar>(arraysize+1);
  Scalar* array = a_array;
  if(Alignment!=Aligned)
    array = (Scalar*)(internal::IntPtr(a_array) + (internal::packet_traits<Scalar>::AlignedOnScalar?sizeof(Scalar):sizeof(type

  {
    Map<VectorType, Alignment, InnerStride<3> > map(array, size);
    map = v;
    for(int i = 0; i < size; ++i)
    {
      VERIFY(array[3*i] == v[i]);
      VERIFY(map[i] == v[i]);
    }
  }
```

Caveman Cupcakes

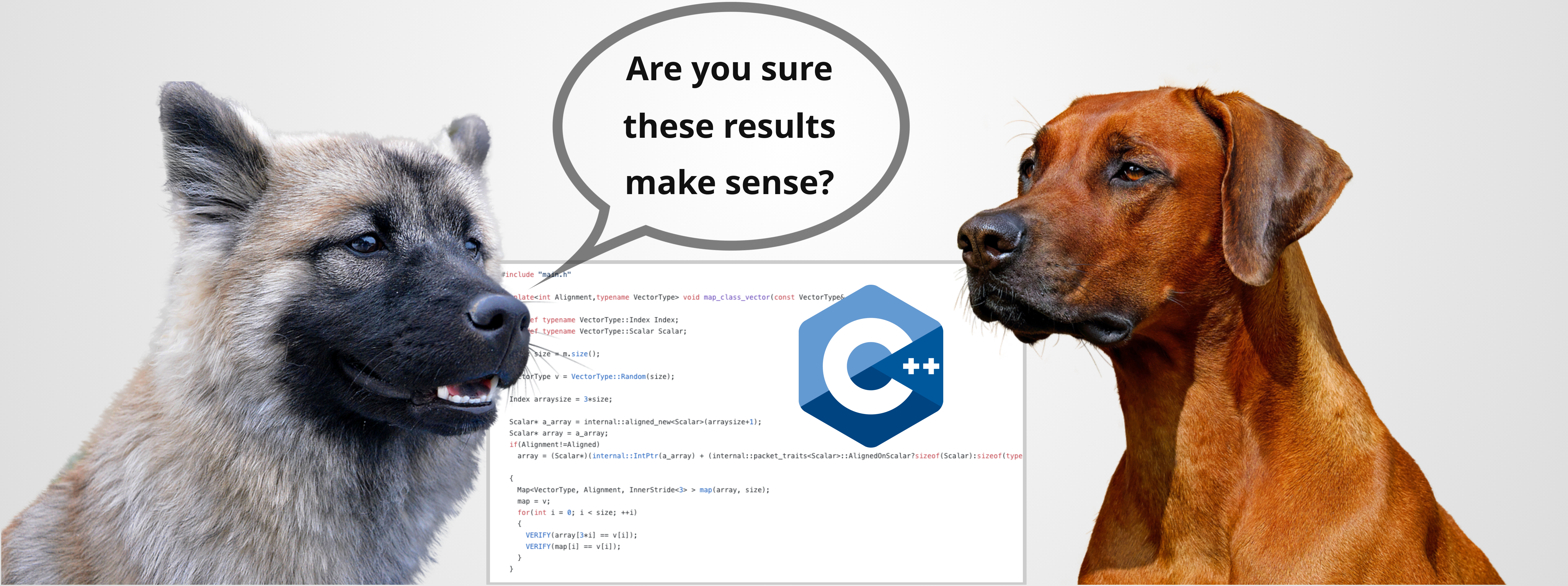


Caveman Cupcakes

Caveman Cupcakes



Caveman Cupcakes



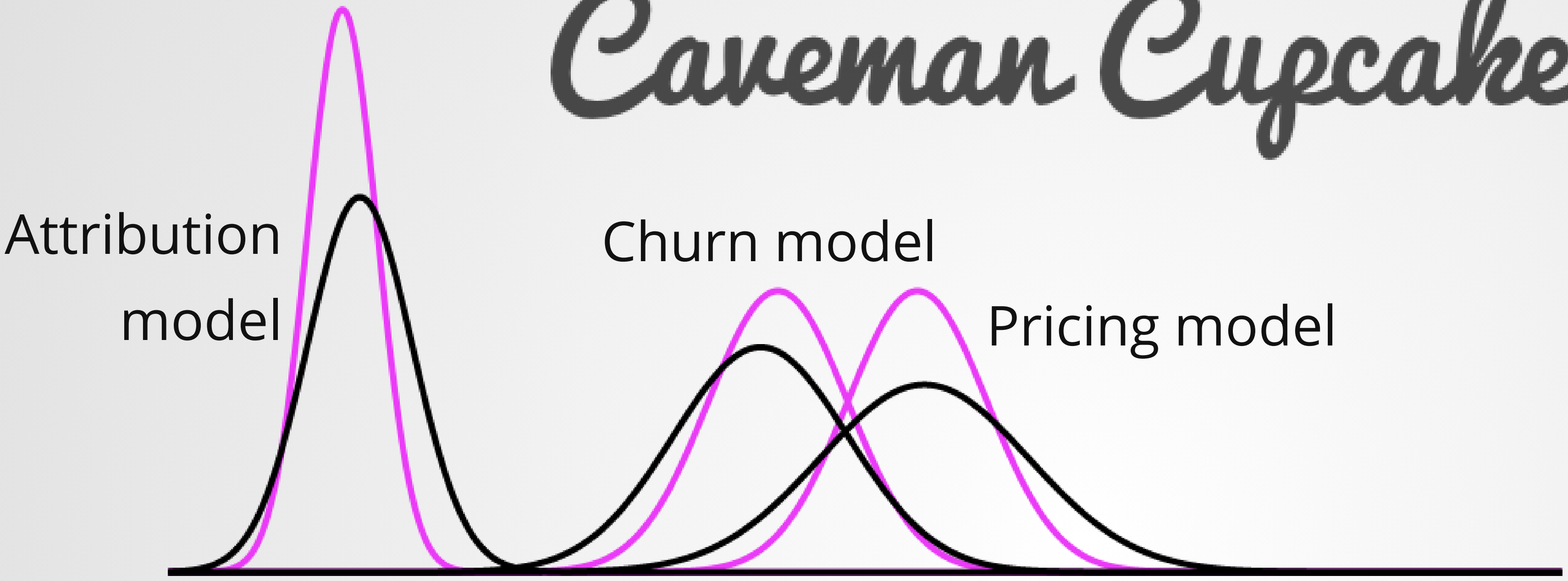
Caveman Cupcakes

Take two

Caveman Cupcakes

Meet the new data science team at Caveman Cupcakes

Caveman Cupcakes




```python
@step
def train_simple_gbrt(self):
    """
    Train a GBRT model with your best guess parameters
    """
    from sklearn import ensemble
    from sklearn.model_selection import train_test_split

    x_train , x_test , y_train , y_test = train_test_split(
          self.features, self.labels, test_size = 0.20, random_state
    self.x_test = x_test
    self.y_test = y_test

    estimator = ensemble.GradientBoostingRegressor(
        n_estimators = 300, max_depth = 4, min_samples_split = 2, learn


    estimator.fit(x_train, y_train)
    self.first_estimator = estimator

    self.next(self.param_search)


@step
def param_search(self):
    """
    Enumerate the list of paramet
    """

    from sklearn.model_selection import

    self.parameter_grid = ParameterGrid({
        'n_estimators':[400, 500],
        'learning_rate': [0.1, 0.05],
        'max_depth':[6, 4],
    })
    self.next(self.fit_gbrt_for_given_param, foreach='parameter_grid')
```

```r
clean_data_set <- function(self) {
  suppressPackageStartupMessages(library(data.table))
  data <- self$data %>%
    py_to_r() %>%
    data.table()
  data[, c("date", "id") := NULL]
  char_cols <- names(data)[(sapply(data, class) == "character")]
  for (col in char_cols) set(data, j = col, value = factor(data[[col]]))
  self$labels <- data[, price] %>%
    r_to_py()
  self$features <- data[, !"price", with = FALSE] %>%
    r_to_py()
}

parameter_grid <- function(self) {
  parameters <- list(
    n.trees = 100 * 1:3,
    shrinkage = .01,
    interaction.depth = c(5, 10),
    n.minobsinnode = 1:3
  )
  parameter_grid <- expand.grid(parameters) %>%
    split(1:nrow(.))
  names(parameter_grid) <- NULL
  self$parameters <- parameter_grid
}
```

VS

VS

the human is the bottleneck

VS

the human is the bottleneck

the human is the bottleneck

Build

Data Warehouse

Compute Resources

Data Warehouse

Job Scheduler

Compute Resources

Data Warehouse

Versioning

Job Scheduler

Compute Resources

Data Warehouse

Collaboration Tools

Versioning

Job Scheduler

Compute Resources

Data Warehouse

Model Deployment

Collaboration Tools

Versioning

Job Scheduler

Compute Resources

Data Warehouse

Feature Engineering

Model Deployment

Collaboration Tools

Versioning

Job Scheduler

Compute Resources

Data Warehouse

ML Libraries

Feature Engineering

Model Deployment

Collaboration Tools

Versioning

Job Scheduler

Compute Resources

Data Warehouse

How much data scientist cares

ML Libraries

Feature Engineering

Model Deployment

Collaboration Tools

Versioning

Job Scheduler

Compute Resources

Data Warehouse

How much data scientist cares

ML Libraries

Feature Engineering

Model Deployment

Collaboration Tools

Versioning

Job Scheduler

Compute Resources

Data Warehouse

How much infrastructure is needed

Deploy


Deploy
No plan survives contact with enemy

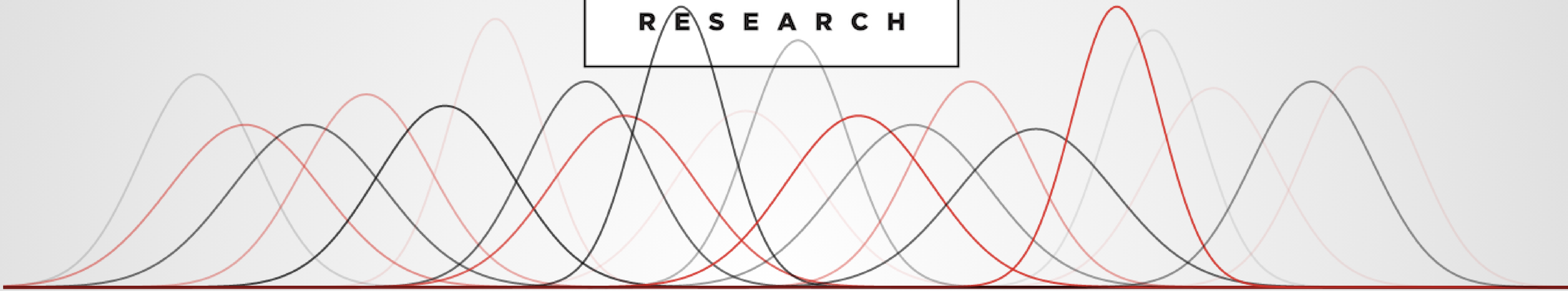
Deploy

No plan survives contact with enemy

No model survives contact with reality

our ML infra supports
two human activities:
building and **deploying**
data science workflows.

NETFLIX



Content Valuation

Optimize Production Schedules

Screenplay Analysis Using NLP

Predict Quality of Network

Intelligent Infrastructure

Machine Translation

Fraud Detection

Content Tagging

Predict Churn

Classify Support Tickets

Title Portfolio Optimization

Incremental Impact of Marketing

Estimate Word-of-Mouth Effects

Cluster Tweets

Optimal CDN Caching

ML Libraries: R, XGBoost, TF etc.

Notebooks: **Nteract**

Job Scheduler: **Meson**

Compute Resources: **Titus**

Query Engine: **Spark**

Data Lake: **S3**

ML Libraries: R, XGBoost, TF etc.

Notebooks: **Nteract**

Job Scheduler: **Meson**

Compute Resources: **Titus**

Query Engine: **Spark**

Data Lake: **S3**

Airflow

kubernetes

models

ML Libraries: R, XGBoost, TF etc.

prototyping

Notebooks: **Nteract**

compute

Job Scheduler: **Meson**

Compute Resources: **Titus**

Query Engine: **Spark**

data

Data Lake: **S3**

Airflow

kubernetes

Bad Old Days

Data Scientist built an NLP model in Python. Easy and fun!

Bad Old Days

Data Scientist built an NLP model in Python. Easy and fun!

How to run at scale?

Custom Titus executor.

Bad Old Days

Data Scientist built an NLP model in Python. Easy and fun!

How to run at scale?

Custom Titus executor.

How to access data at scale?

Slow!

Bad Old Days

Data Scientist built an NLP model in Python. Easy and fun!

How to run at scale?

Custom Titus executor.

How to access data at scale?

Slow!

How to schedule the model to update daily?

Learn about the job scheduler.

Bad Old Days

Data Scientist built an NLP model in Python. Easy and fun!

How to run at scale?

Custom Titus executor.

How to access data at scale?

Slow!

How to schedule the model to update daily?

Learn about the job scheduler.

How to expose the model to a custom UI?

Custom web backend.

Bad Old Days

Data Scientist built an NLP model in Python. Easy and fun!

How to run at scale?

Custom Titus executor.

How to access data at scale?

Slow!

How to schedule the model to update daily?

Learn about the job scheduler.

How to expose the model to a custom UI?

Custom web backend.

Bad Old Days

Time to production:

4 months

Data Scientist built an NLP model in Python. Easy and fun!

How to monitor models in production?

Ho ... at scale?

Custom Titus executor.

How to access data at sc...

Sl...

How to let another data scientist iterate on her version of the model safely?

H... schedule the model to update daily?

...t the job scheduler.

How ... del to a custom UI?

Custom web bu...

How to debug yesterday's failed production run?

How to iterate on a new version without breaking the production version?

Bad Old Days

...production:

4 months

How to backfill historical data?

How to make this faster?

ML Wrapping: **Metaflow**

models — ML Libraries: R, XGBoost, TF etc.

prototyping — Notebooks: **Nteract**

compute — Job Scheduler: **Meson**

compute — Compute Resources: **Titus**

compute — Query Engine: **Spark**

data — Data Lake: **S3**

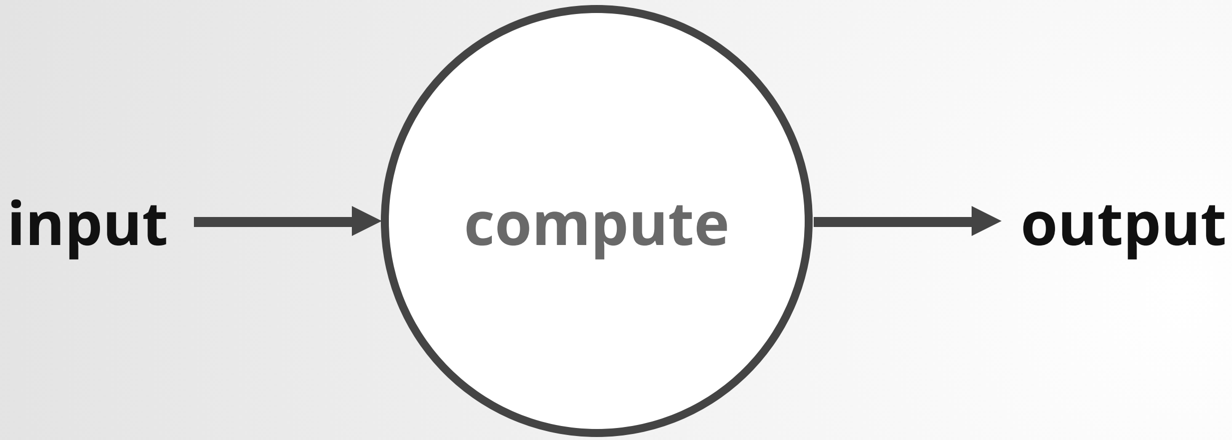
Airflow

kubernetes

Metaflow

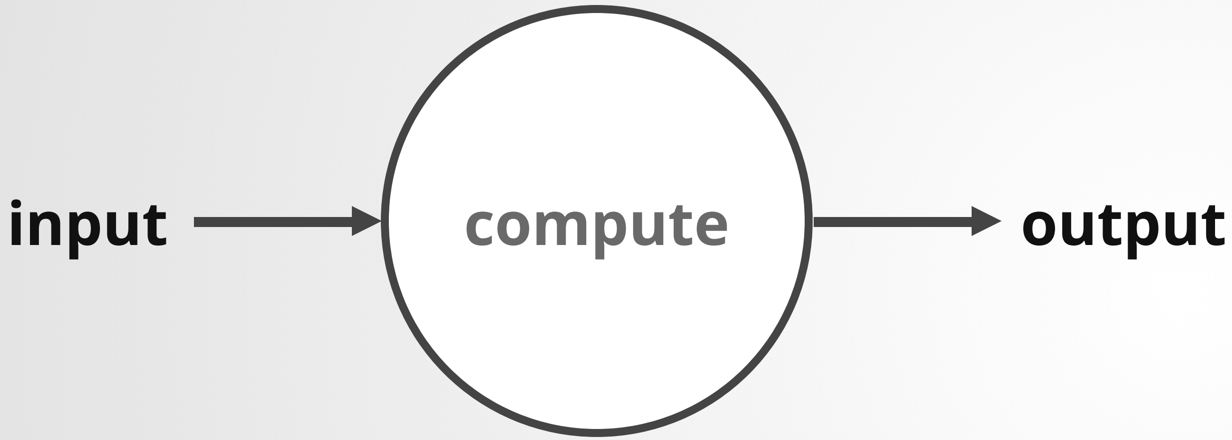
Build

How to get started?




```
def compute(input):
    output = my_model(input)
    return output
```

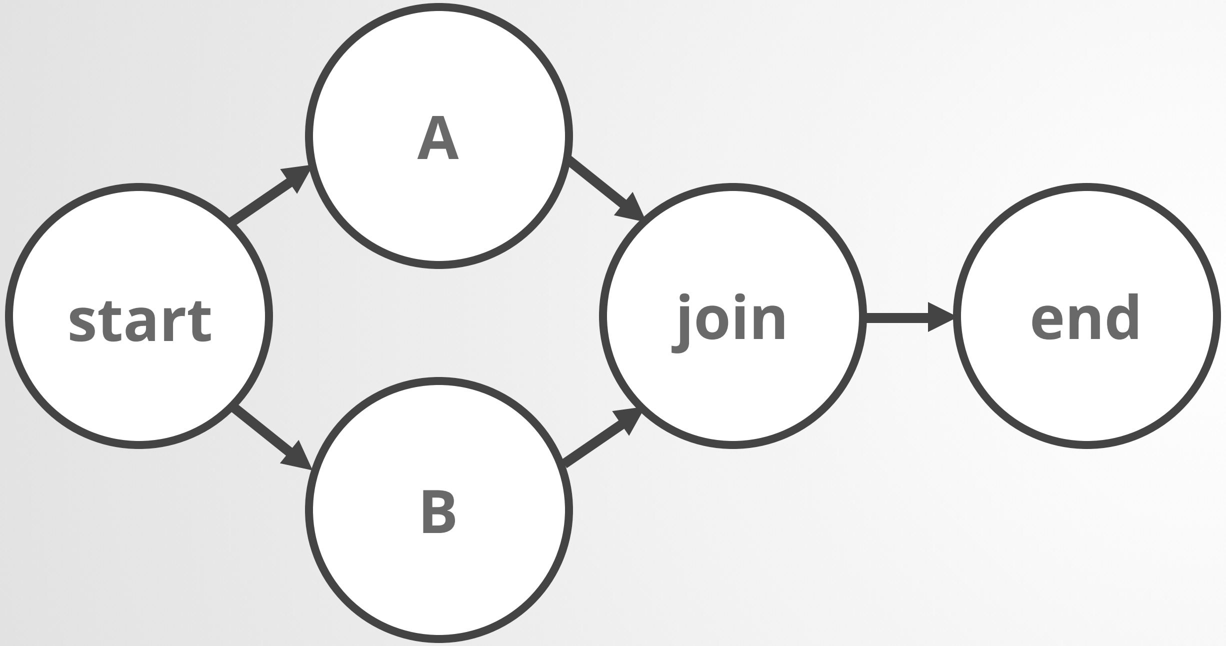
How to get started?




```python
def compute(input):
    output = my_model(input)
    return output
```

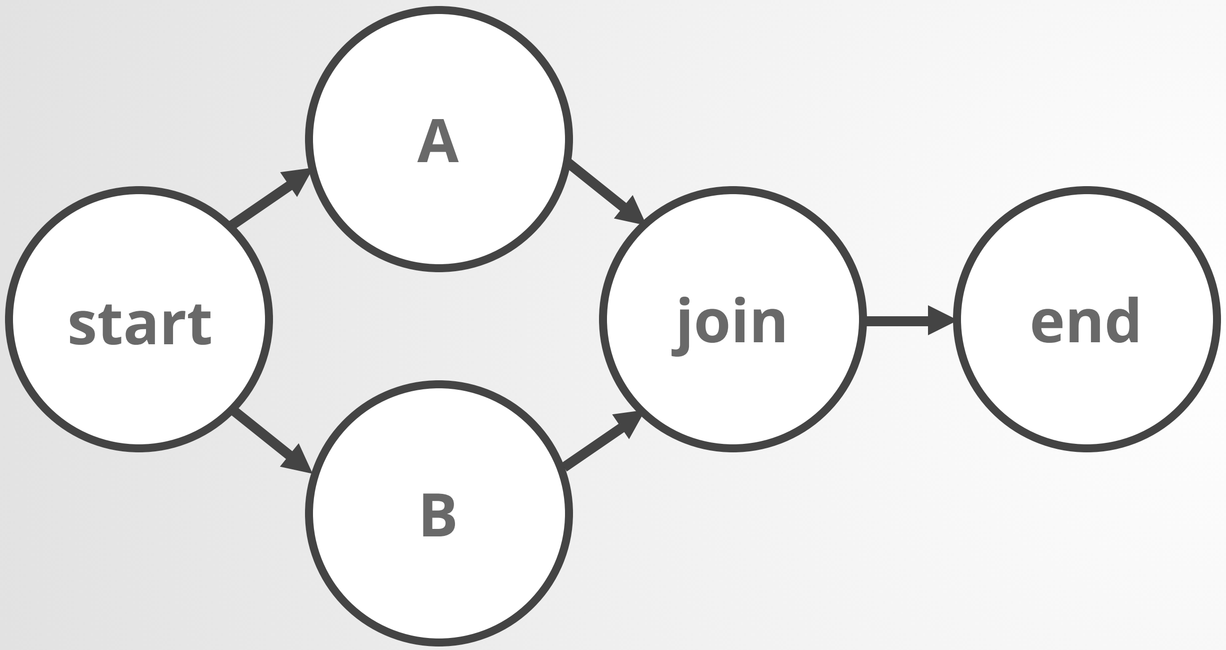
```
# python myscript.py
```

How to structure my code?




```python
from metaflow import FlowSpec, step

class MyFlow(FlowSpec):

  @step
  def start(self):
    self.next(self.a, self.b)

  @step
  def a(self):
    self.next(self.join)

  @step
  def b(self):
    self.next(self.join)

  @step
  def join(self, inputs):
    self.next(self.end)

MyFlow()
```

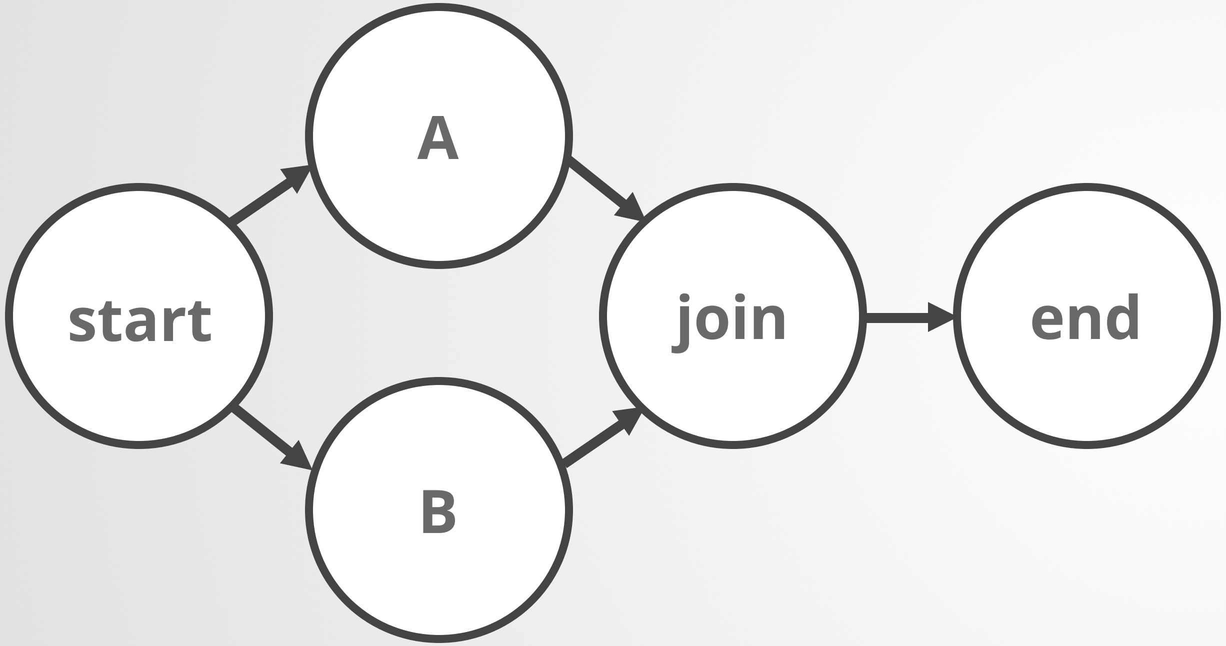
How to structure my code?




```
# python myscript.py run
```

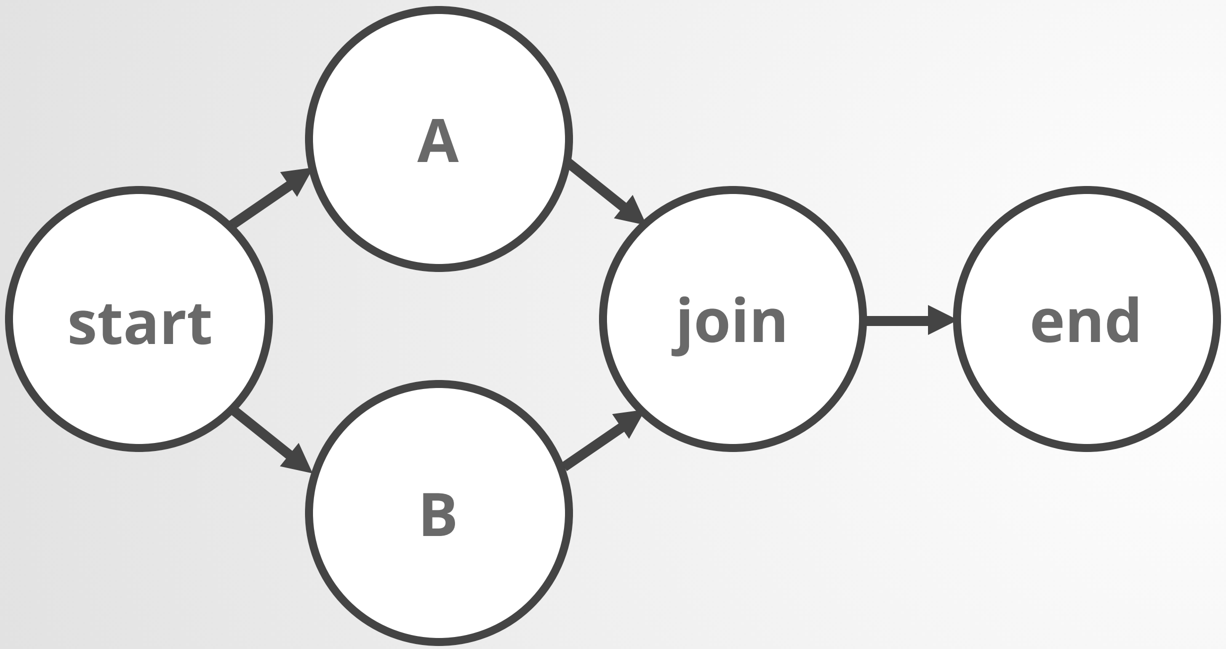
```python
from metaflow import FlowSpec, step

class MyFlow(FlowSpec):

  @step
  def start(self):
    self.next(self.a, self.b)

  @step
  def a(self):
    self.next(self.join)

  @step
  def b(self):
    self.next(self.join)

  @step
  def join(self, inputs):
    self.next(self.end)

MyFlow()
```

How to deal with models written in R?




```
metaflow("MyFlow") %>%
  step(
    step = "start",
    next_step = c("a", "b")
  ) %>%
  step(
    step = "A",
    r_function = r_function(a_func),
    next_step = "join"
  ) %>%
  step(
    step = "B",
    r_function = r_function(b_func),
    next_step = "join"
  ) %>%
  step(
    step = "Join",
    r_function = r_function(join,
                    join_step = TRUE),
```

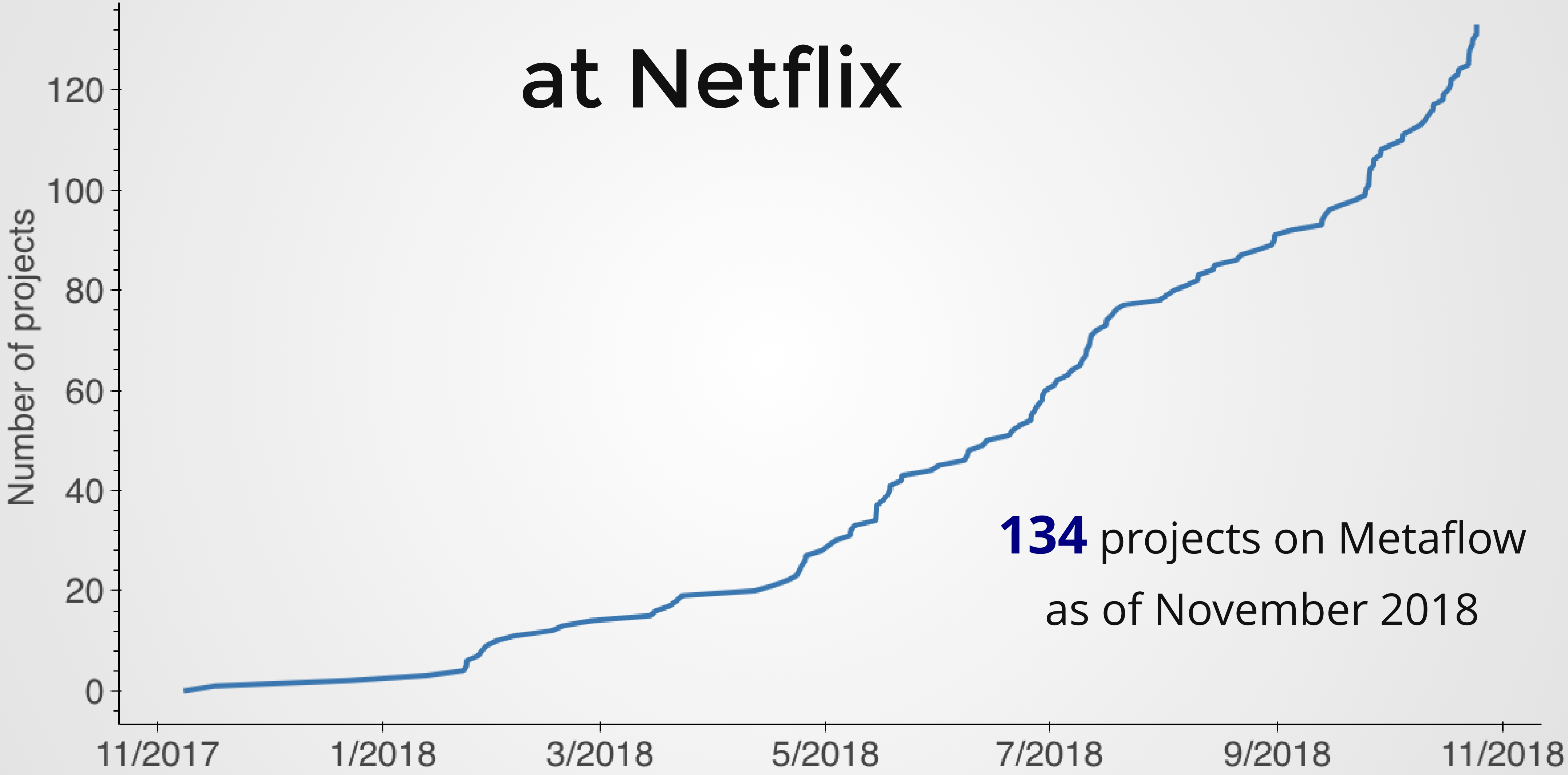
How to deal with models written in R?




```
# RScript myscript.R
```

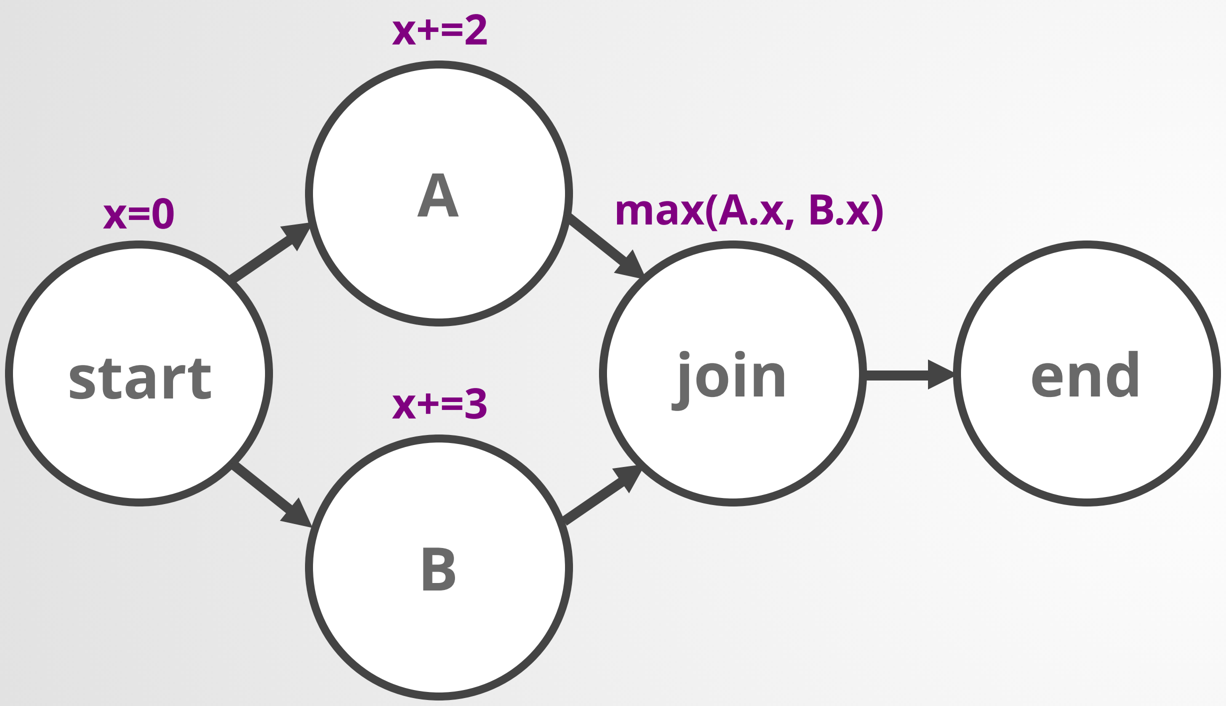
```
metaflow("MyFlow") %>%
  step(
    step = "start",
    next_step = c("a", "b")
  ) %>%
  step(
    step = "A",
    r_function = r_function(a_func),
    next_step = "join"
  ) %>%
  step(
    step = "B",
    r_function = r_function(b_func),
    next_step = "join"
  ) %>%
  step(
    step = "Join",
    r_function = r_function(join,
                            join_step = TRUE),
```

Metaflow adoption at Netflix



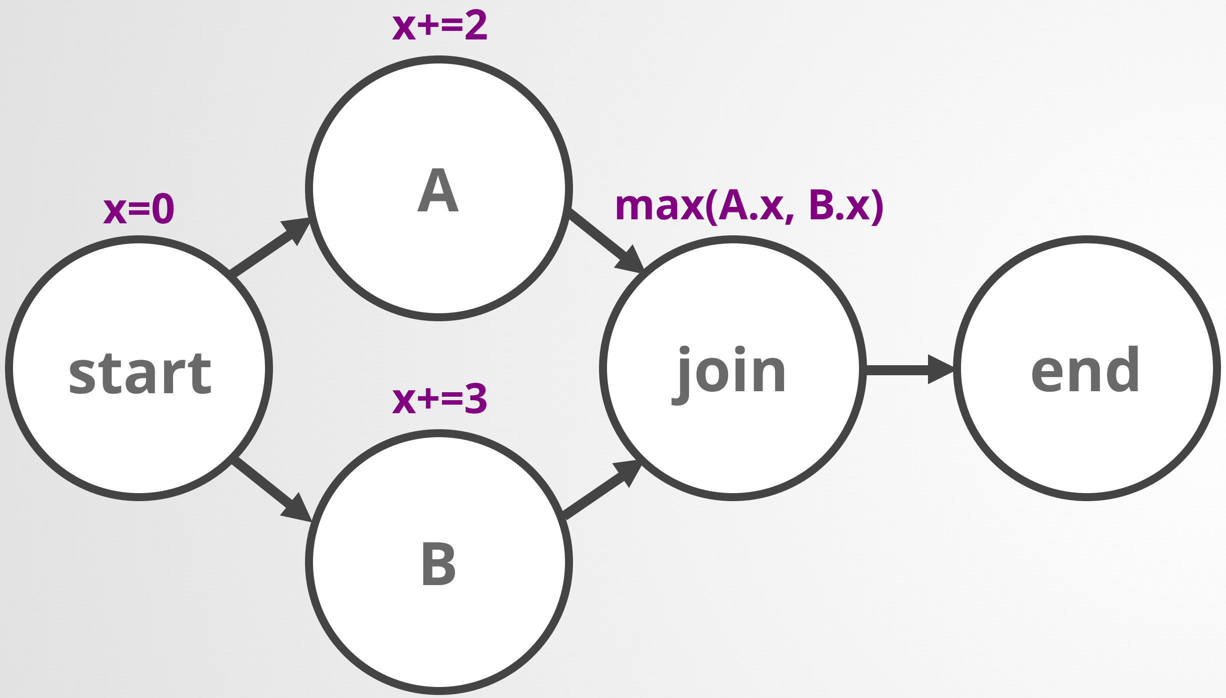
134 projects on Metaflow
as of November 2018

How to prototype and test my code locally?




```
@step
def start(self):
  self.x = 0
  self.next(self.a, self.b)

@step
def a(self):
  self.x += 2
  self.next(self.join)

@step
def b(self):
  self.x += 3
  self.next(self.join)

@step
def join(self, inputs):
  self.out = max(i.x for i in inputs)
  self.next(self.end)
```

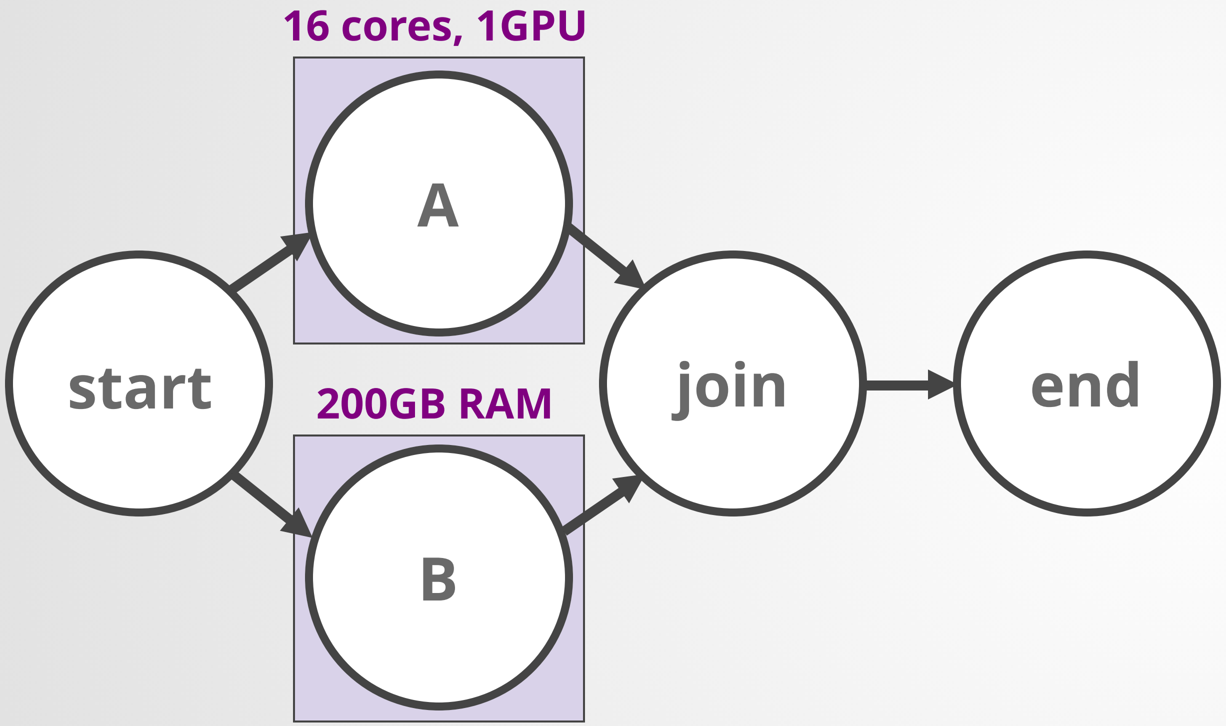
How to prototype and test my code locally?

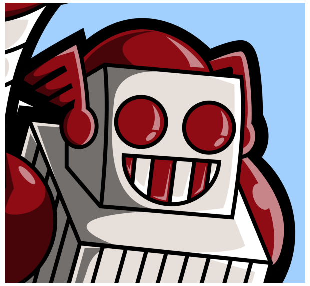



```
# python myscript.py resume B
```

```
@step
def start(self):
  self.x = 0
  self.next(self.a, self.b)

@step
def a(self):
  self.x += 2
  self.next(self.join)

@step
def b(self):
  self.x += 3
  self.next(self.join)

@step
def join(self, inputs):
  self.out = max(i.x for i in inputs)
  self.next(self.end)
```

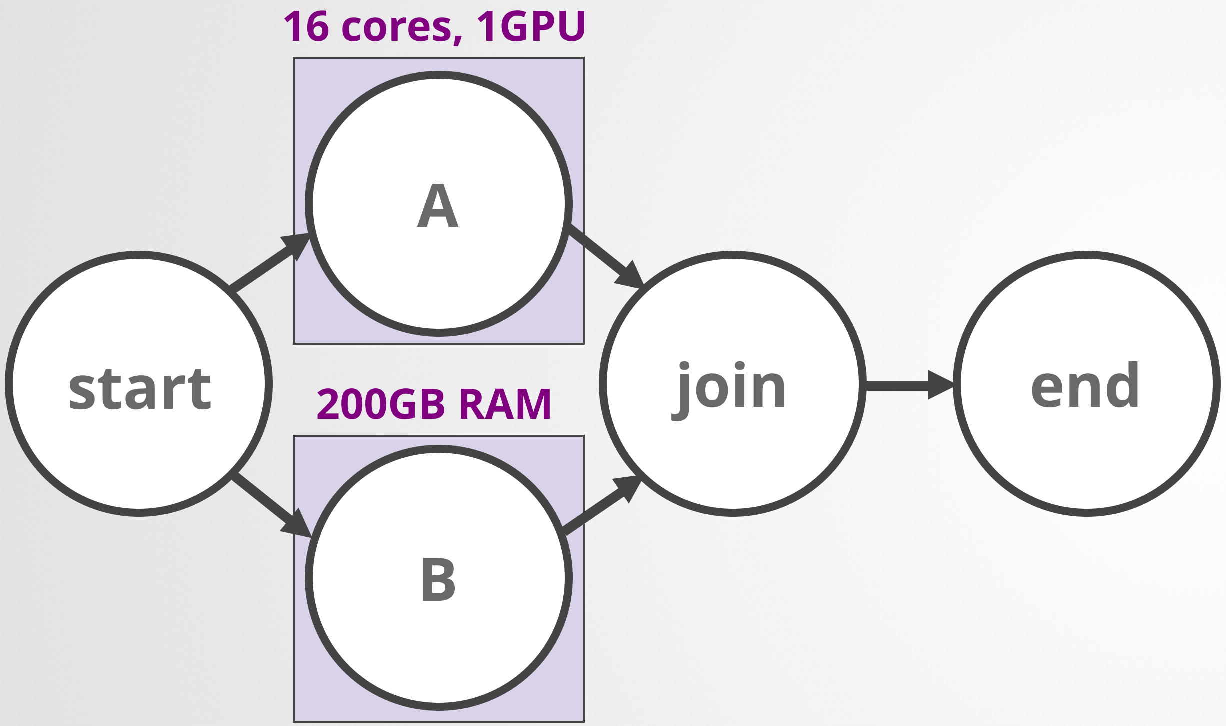
How to get access to more CPUs, GPUs, or memory?



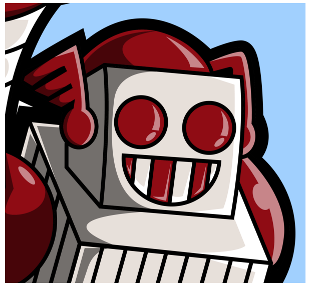


```python
@titus(cpu=16, gpu=1)
@step
def a(self):
  tensorflow.train()
  self.next(self.join)

@titus(memory=200000)
@step
def b(self):
  massive_dataframe_operation()
  self.next(self.join)
```

How to get access to more CPUs, GPUs, or memory?

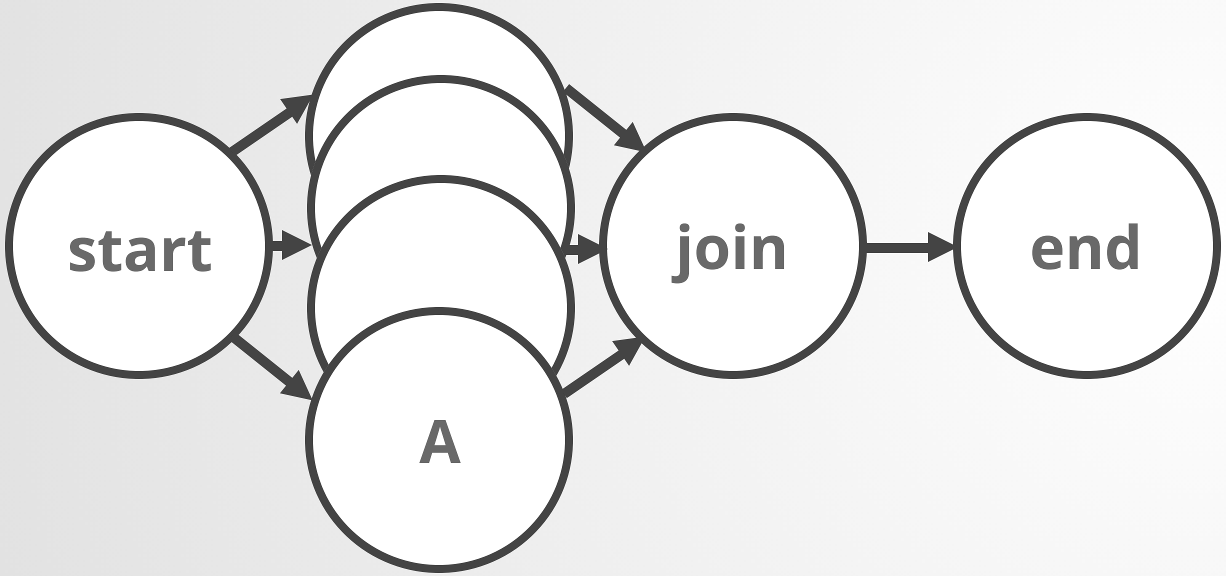



```
# python myscript.py run
```



```
@titus(cpu=16, gpu=1)
@step
def a(self):
  tensorflow.train()
  self.next(self.join)

@titus(memory=200000)
@step
def b(self):
  massive_dataframe_operation()
  self.next(self.join)
```

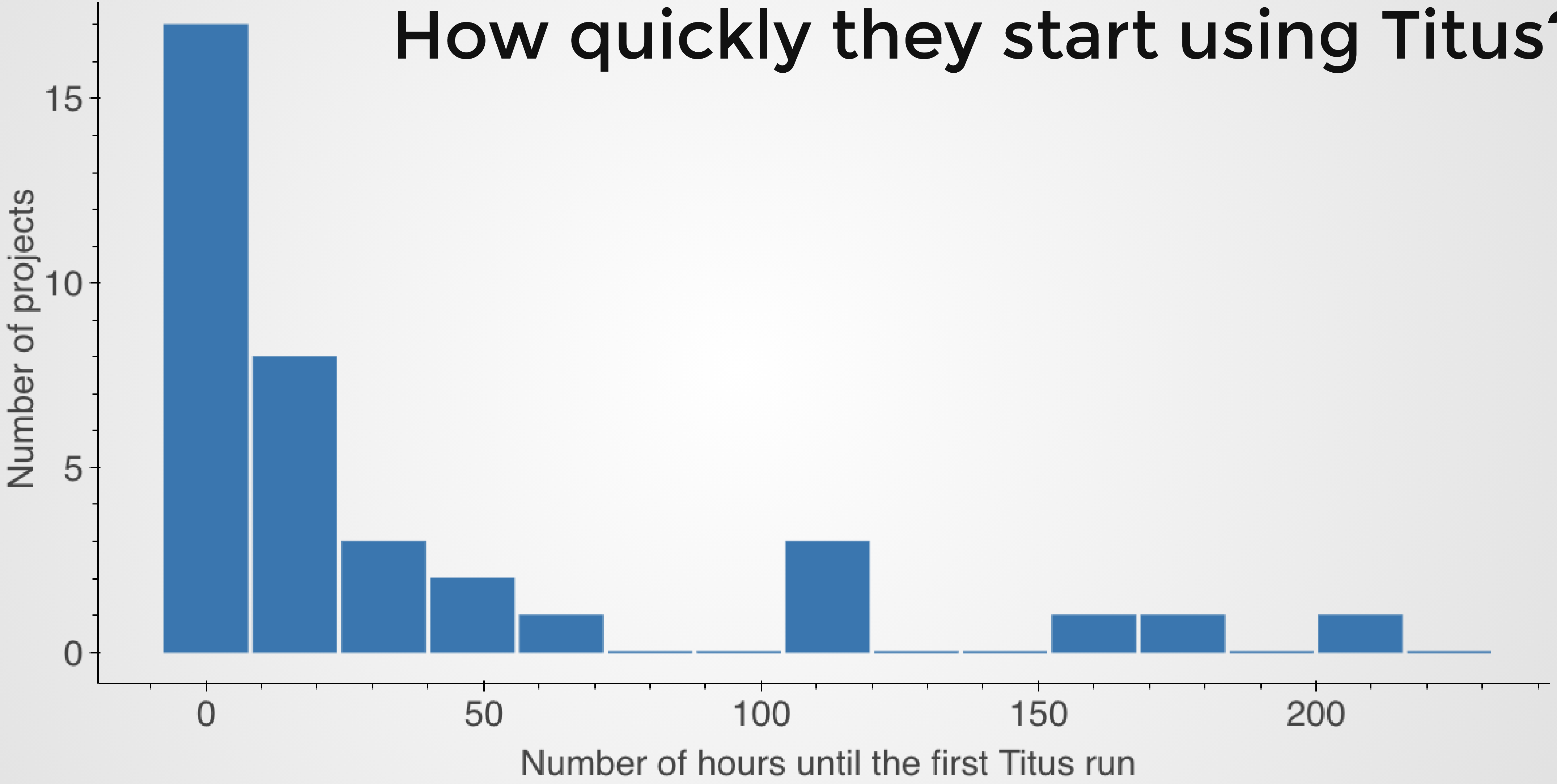
How to distribute work over many parallel jobs?



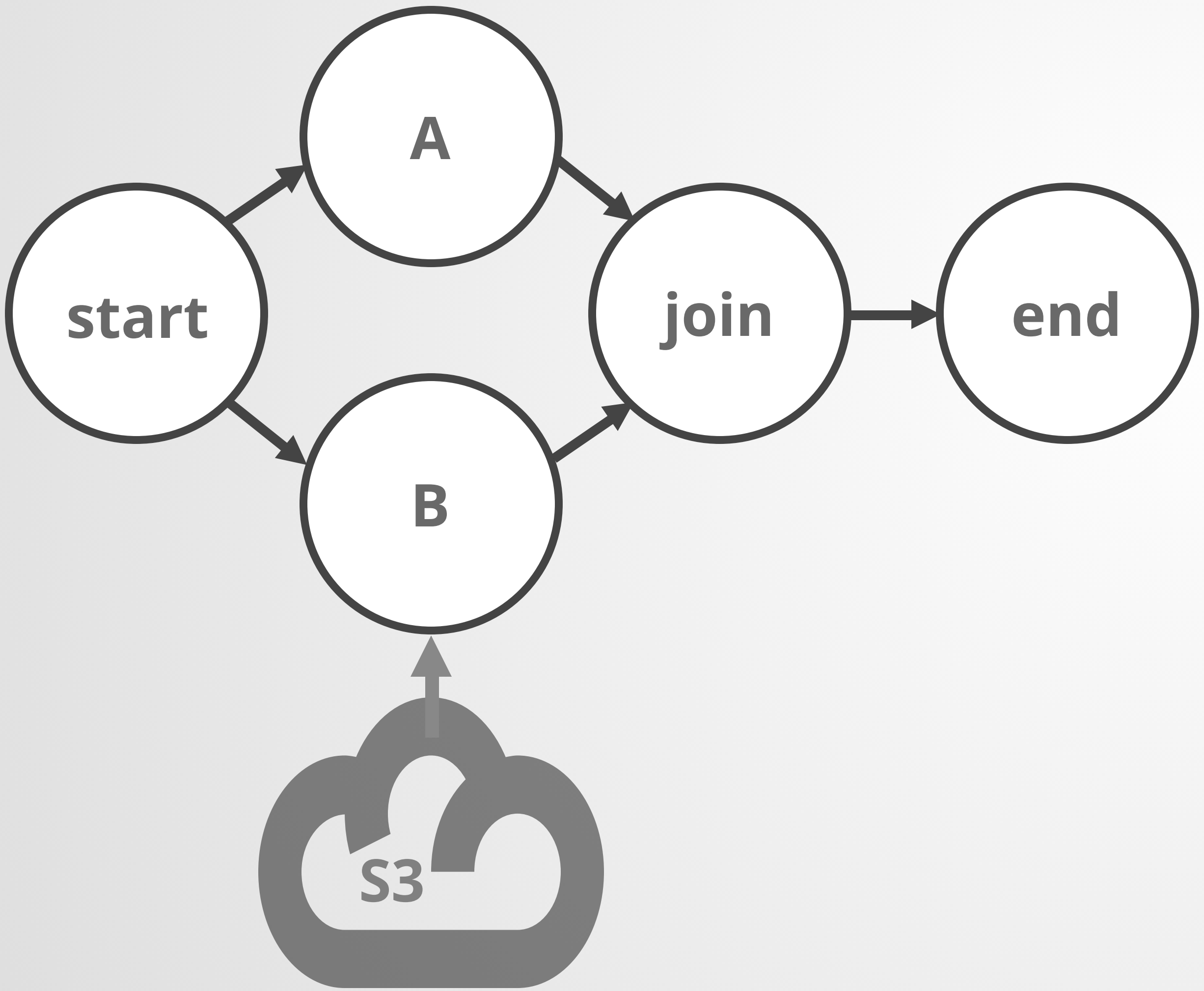

```
@step
def start(self):
  self.grid = ['x','y','z']
  self.next(self.a, foreach='grid')

@titus(memory=10000)
@step
def a(self):
  self.x = ord(self.input)
  self.next(self.join)

@step
def join(self, inputs):
  self.out = max(i.x for i in inputs)
  self.next(self.end)
```

40% of projects run steps outside their dev environment.

How quickly they start using Titus?



How to access large amounts of input data?




```
from metaflow import Table

@titus(memory=200000, network=20000)
@step
def b(self):
    # Load data from S3 to a dataframe
    # at 10Gbps
    df = Table('vtuulos', 'input_table')
    self.next(self.end)
```

Case Study: Marketing Cost per Incremental Watcher

1. Build a separate model for every new title with marketing spend.
   Parallel foreach.

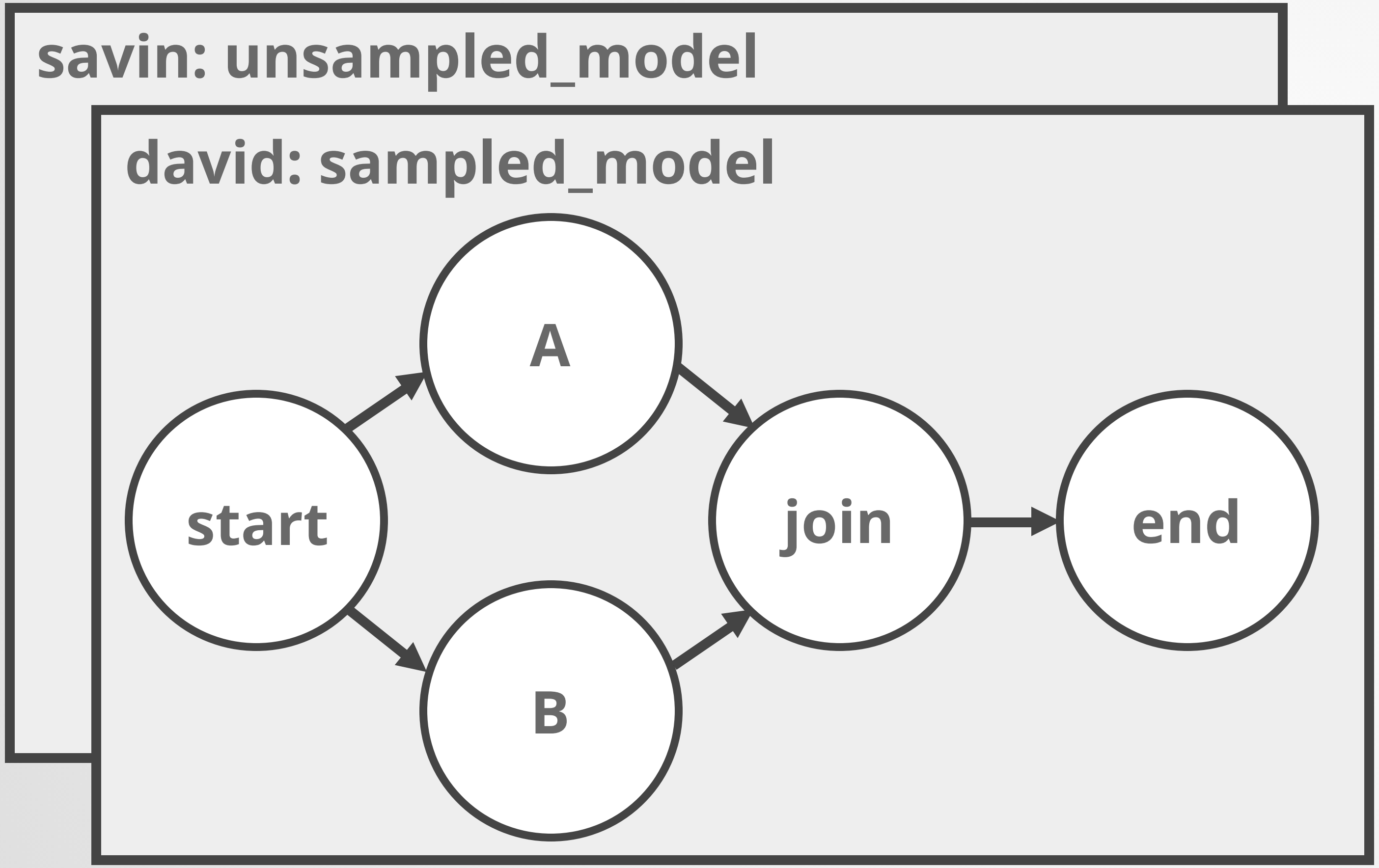
2. Load and prepare input data for each model.
   Download Parquet **directly from S3**.
   Total amount of model input data: **890GB**.

3. Fit a model.
   Train each model on an instance with **400GB of RAM, 16 cores**.
   The model is **written in R.**

4. Share updated results.
   Collect results of individual models, write to a table.
   Results shown on **a Tableau dashboard.**

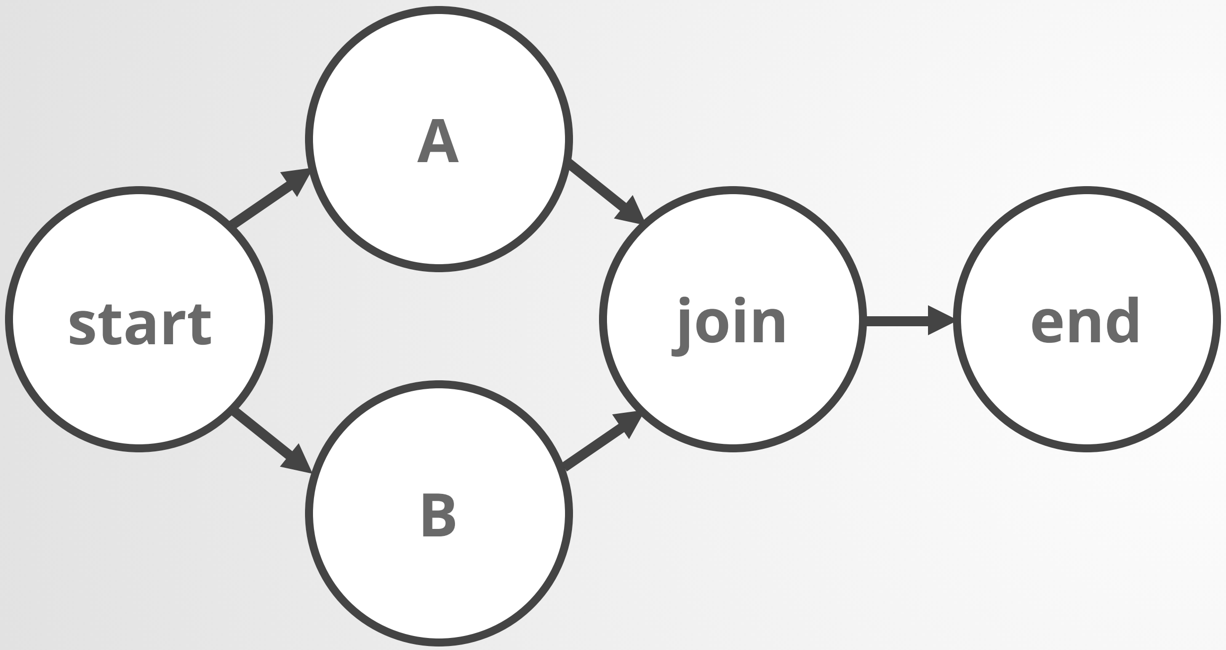
Deploy

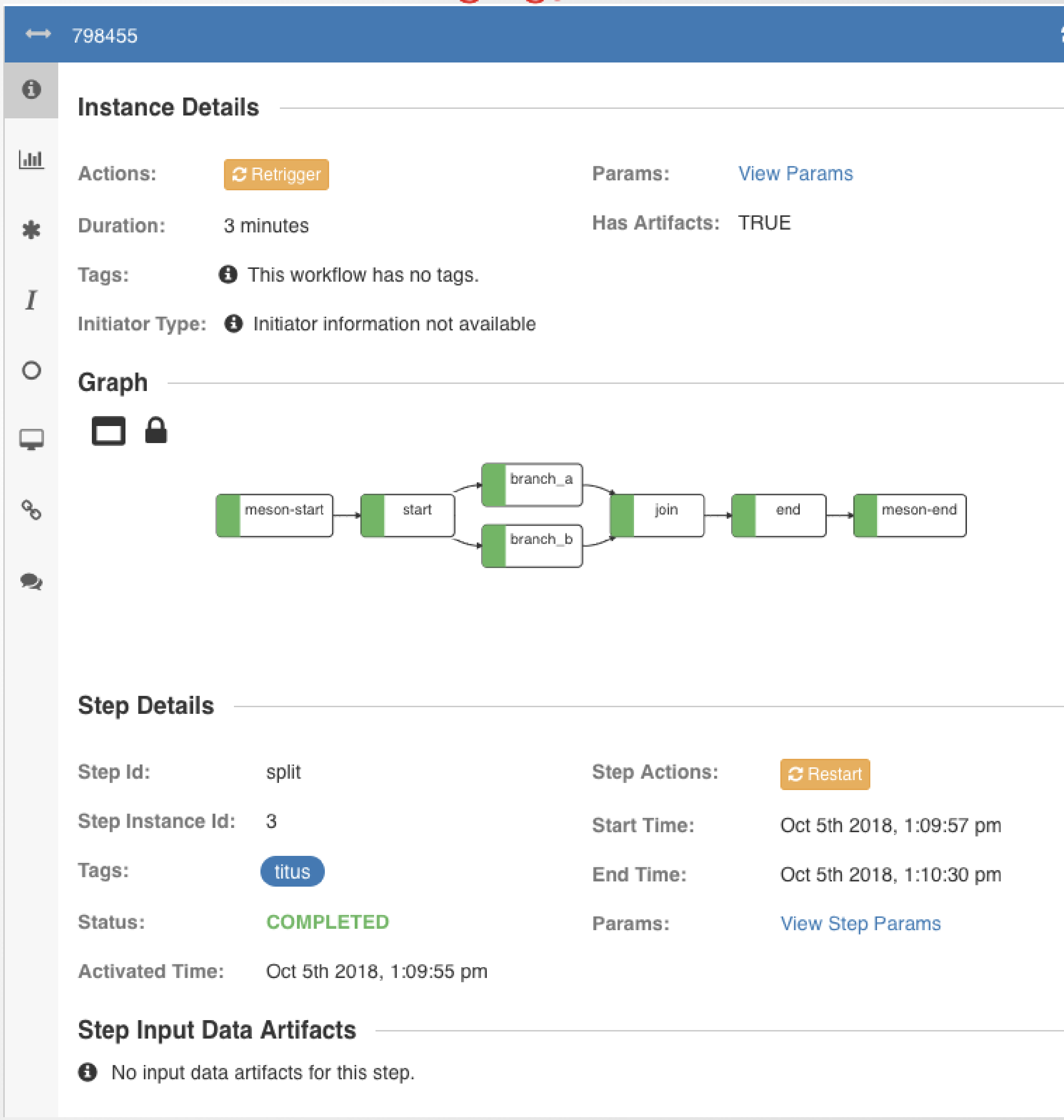
How to version my results and access results by others?



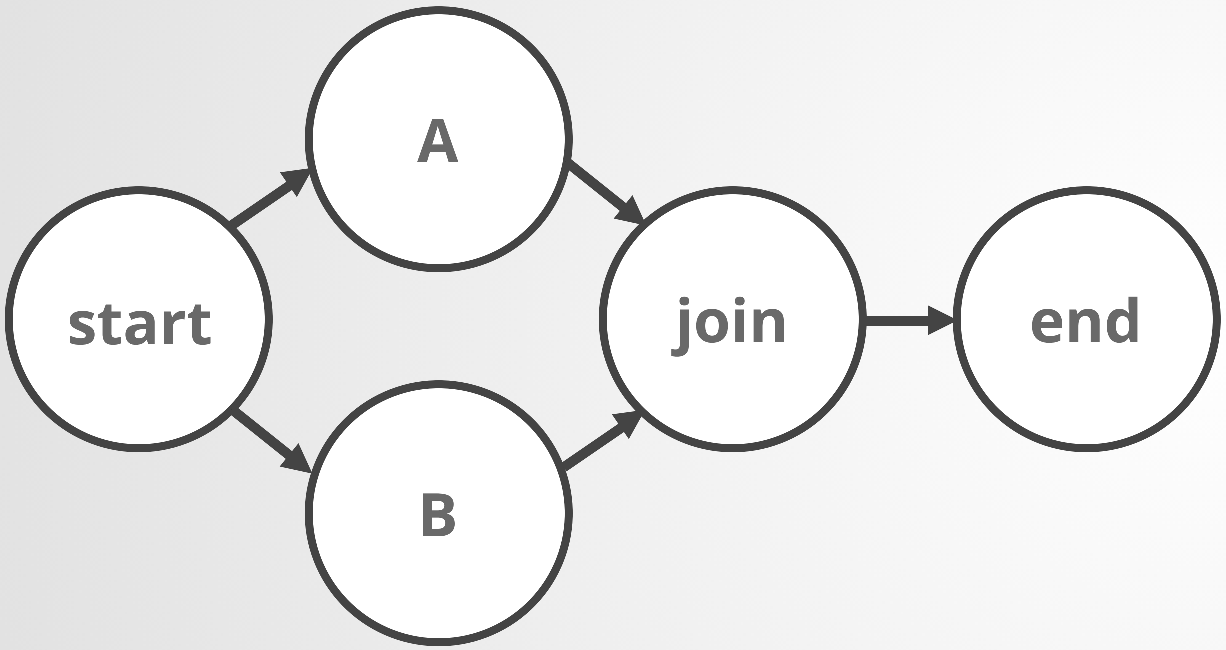

```python
# Access Savin's runs
namespace('user:savin')
run = Flow('MyFlow').latest_run
print(run.id) # = 234
print(run.tags) # = ['unsampled_model']

# Access David's runs
namespace('user:david')
run = Flow('MyFlow').latest_run
print(run.id) # = 184
print(run.tags) # = ['sampled_model']

# Access everyone's runs
namespace(None)
run = Flow('MyFlow').latest_run
print(run.id) # = 184
```

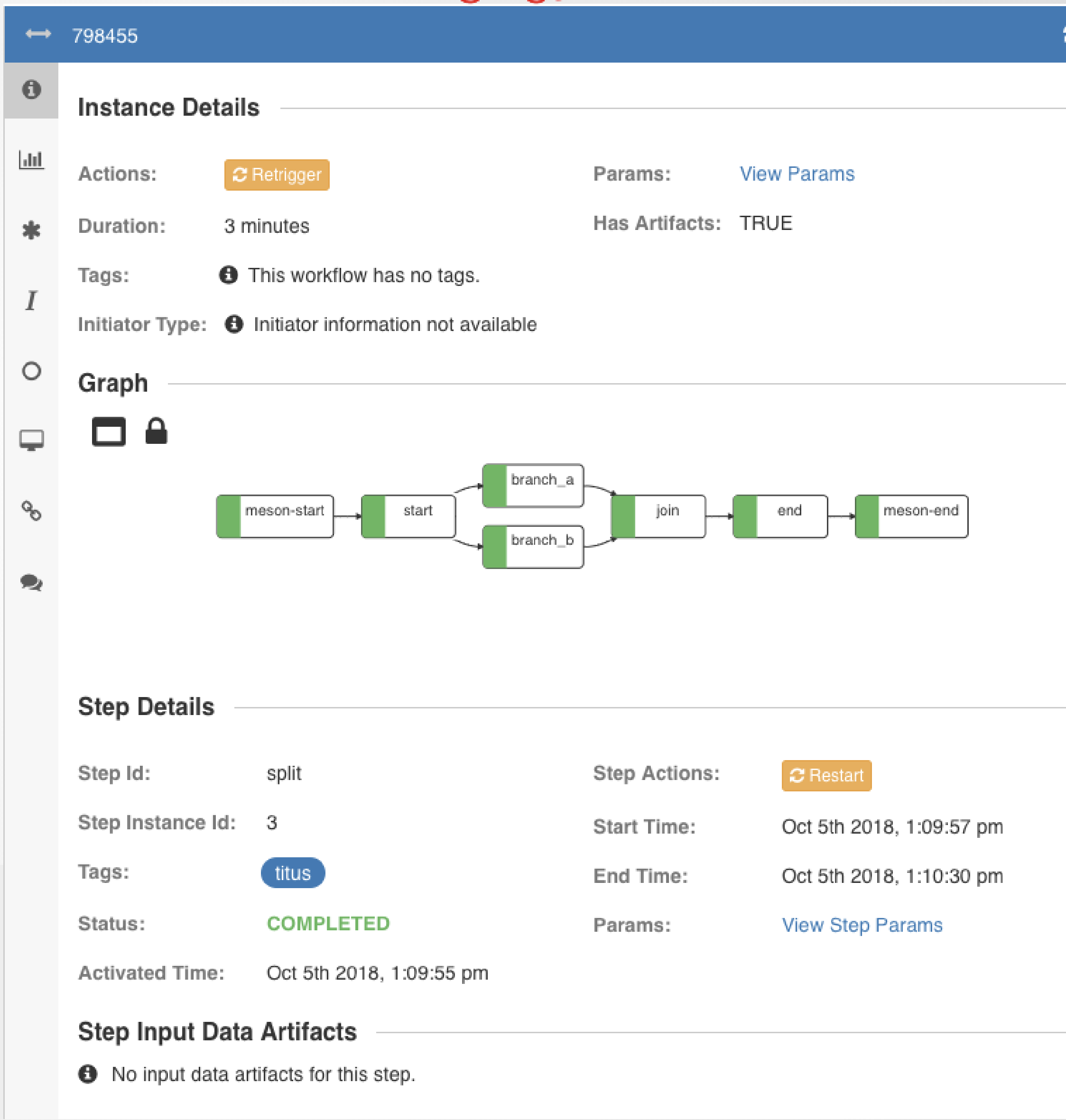
How to deploy my workflow to production?





How to deploy my workflow to production?

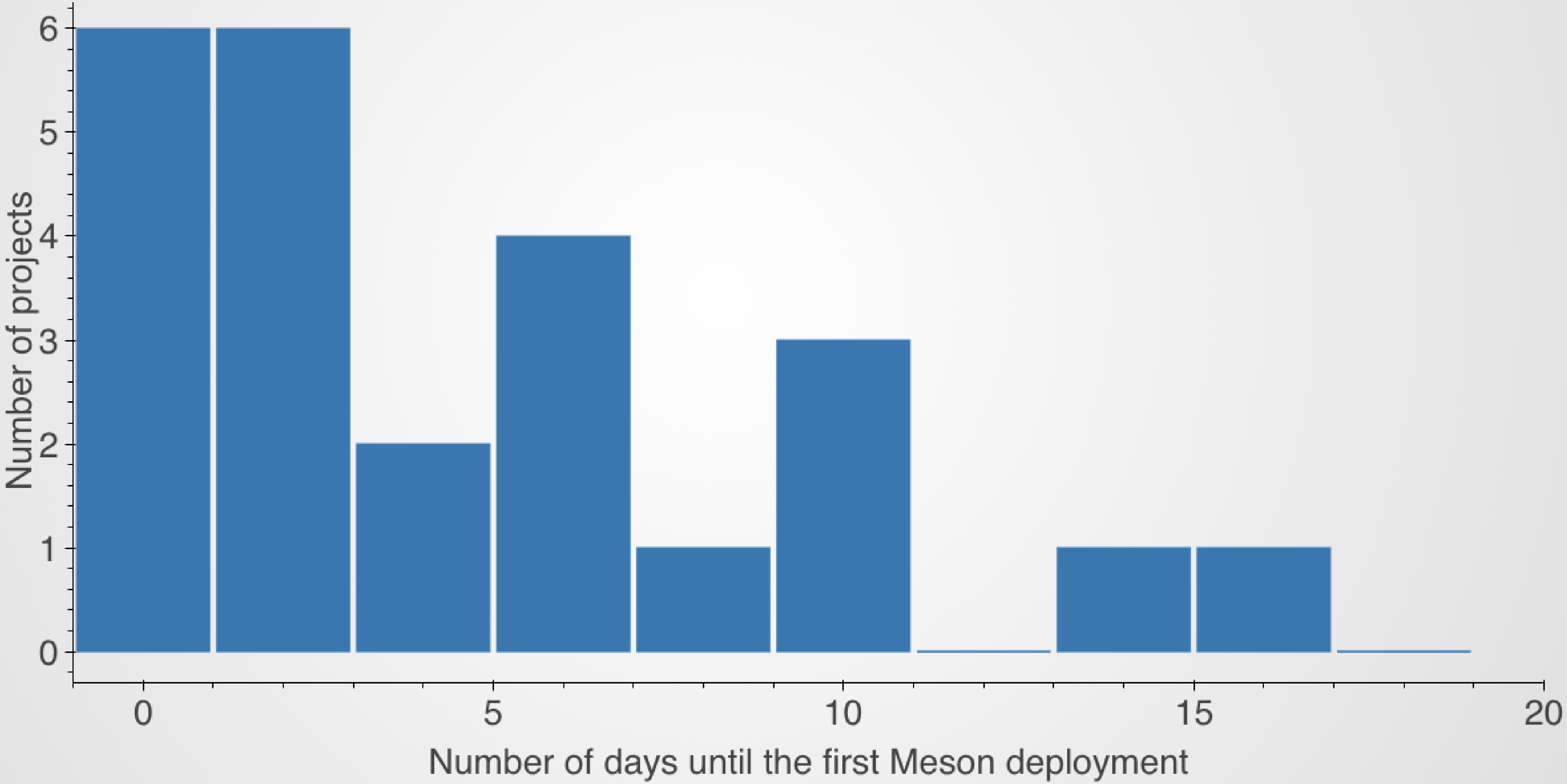




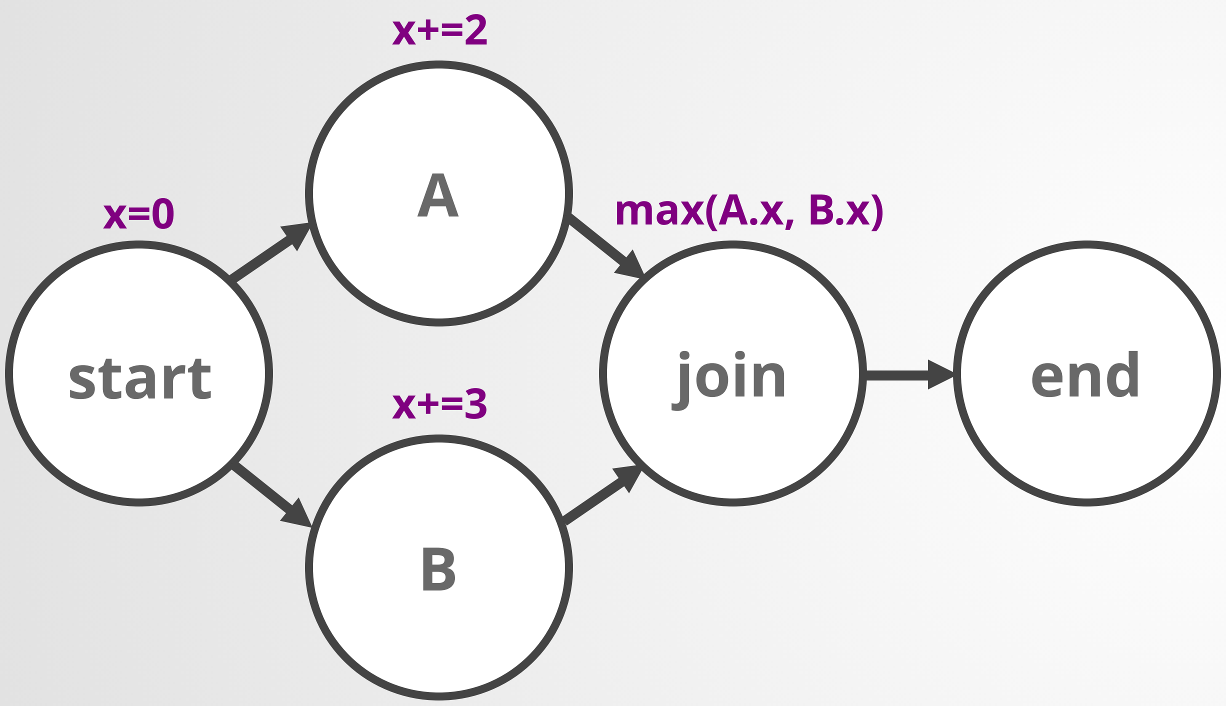
#python myscript.py meson create

26% of projects get deployed to the production scheduler.

How quickly the first deployment happens?

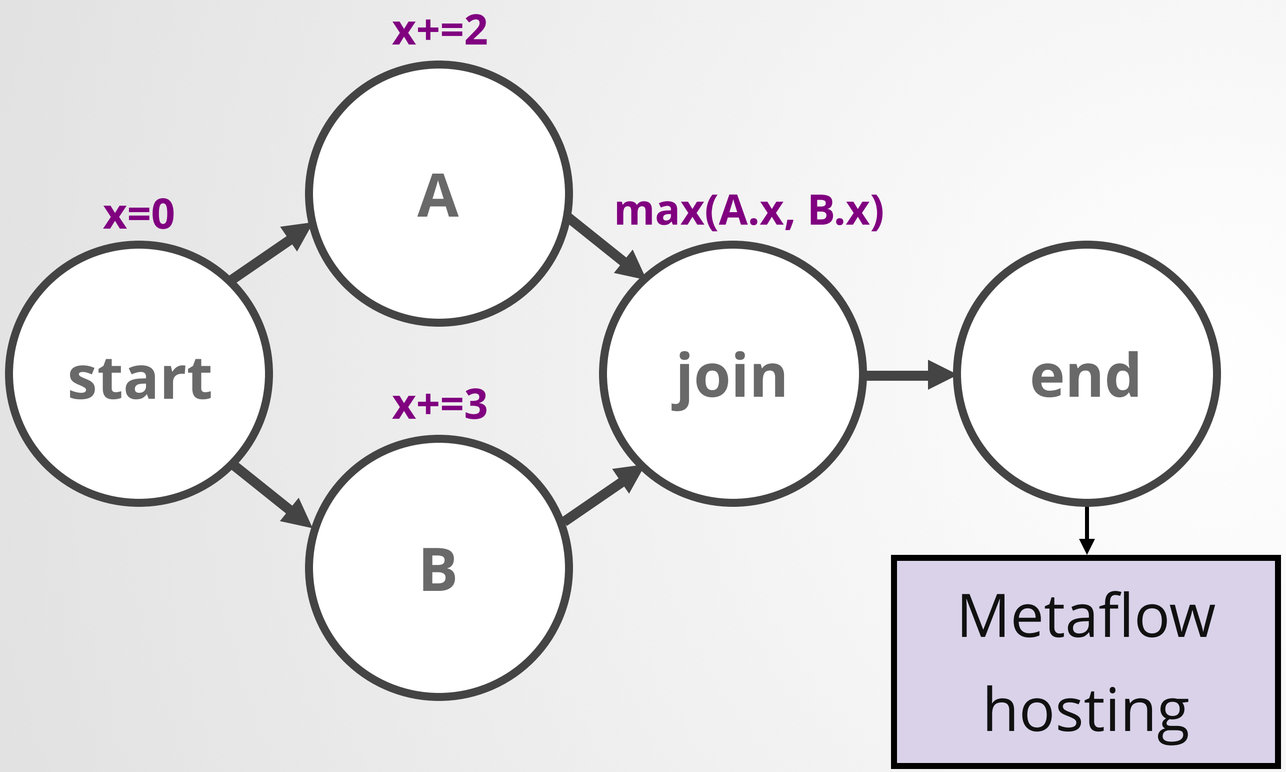


How to monitor models and examine results?



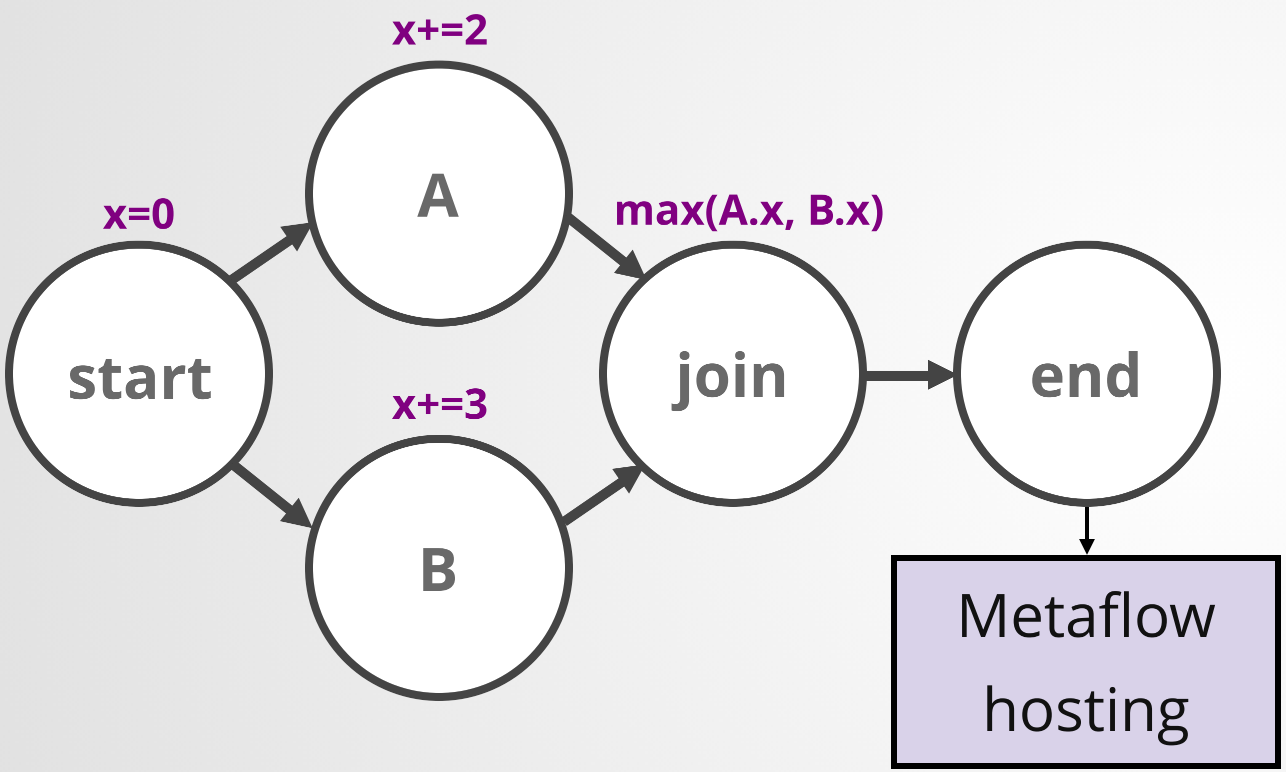

```
In [145]: from metaflow import Flow
          run = Flow('MyFlow').latest_run
          run

Out[145]: Run('MyFlow/3')

In [152]: list(run)

Out[152]: [Step('MyFlow/3/end'),
           Step('MyFlow/3/join'),
           Step('MyFlow/3/b'),
           Step('MyFlow/3/a'),
           Step('MyFlow/3/start')]

In [150]: run['start'].task.data.x

Out[150]: 0

In [151]: run['a'].task.data.x

Out[151]: 2
```

How to deploy results as a microservice?




```python
from metaflow import WebServiceSpec
from metaflow import endpoint

class MyWebService(WebServiceSpec):

    @endpoint
    def show_data(self, request_dict):
        # TODO: real-time predict here
        result = self.artifacts.flow.x
        return {'result': result}
```

How to deploy results as a microservice?




```
from metaflow import WebServiceSpec
from metaflow import endpoint

class MyWebService(WebServiceSpec):

    @endpoint
    def show_data(self, request_dict):
        # TODO: real-time predict here
        result = self.artifacts.flow.x
        return {'result': result}
```

```
# curl http://host/show_data
```

{"result": 3}

Case Study: Launch Date Schedule Optimization

1. Batch optimize launch date schedules for new titles daily.

   Batch optimization deployed on **Meson.**

2. Serve results through a custom UI.

   Results deployed on **Metaflow Hosting**.

3. Support arbitrary what-if scenarios in the custom UI.

   Run optimizer **in real-time in a custom web endpoint.**

Metaflow

diverse problems

diverse problems

↓

diverse people

diverse problems

↓

diverse people

↓

diverse models

diverse problems

↓

diverse people

↓

diverse models

↓

help people build

diverse problems

diverse people

diverse models

help people build

help people deploy

diverse problems

↓

diverse people

↓

diverse models

↓

help people build

↓

help people deploy

↓

happy people, healthy business

thank you!

@vtuulos
vtuulos@netflix.com

Photo Credits

https://www.maxpixel.net/Isolated-Animal-Hundeportrait-Dog-Nature-3234285

Bruno Coldiori

https://www.flickr.com/photos/br1dotcom/8900102170/