How Beam ML Serves Large Models

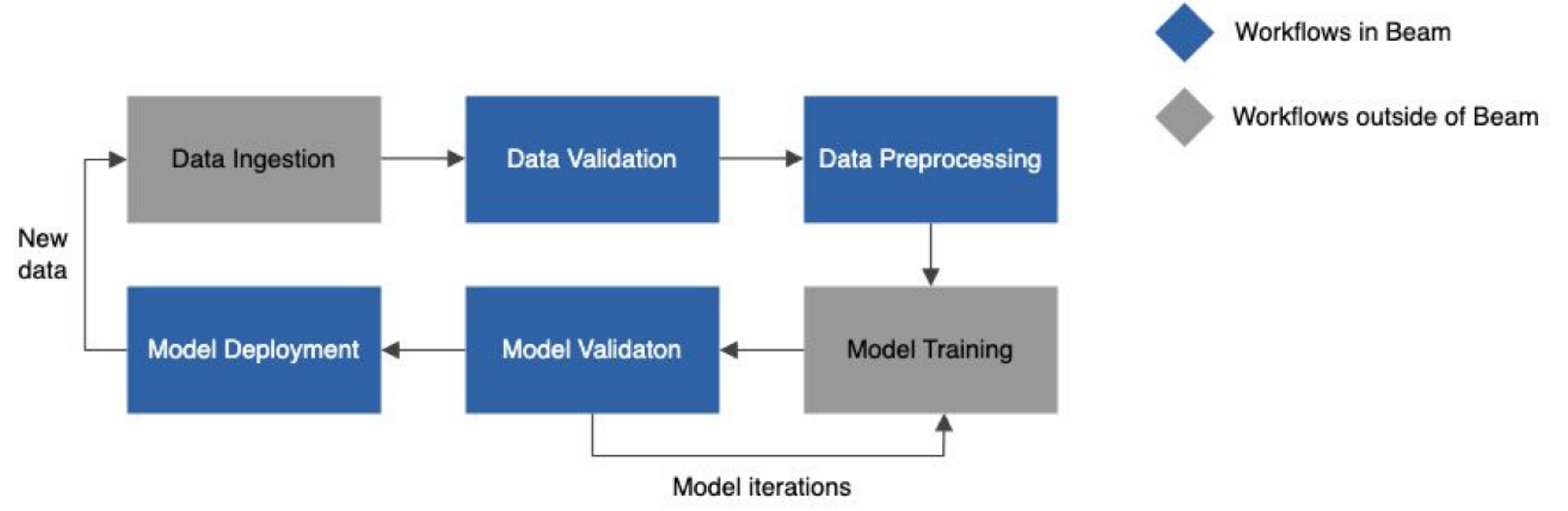
Danny McCormick

BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

The ML lifecycle



Inference with Beam

Challenges of Distributed Inference

- Efficiently loading models
- Batching
- Model Updates
- Using multiple models

- Beam takes care of all of this with the RunInference transform
- Loads model, batches inputs, handles updates, and plugs into DAG

```
RunInference(model_handler=<config>)
```

RunInference

```
>>> data = numpy.array([10, 40, 60, 90],
...                           dtype=numpy.float32).reshape(-1, 1)

>>> model_handler = PytorchModelHandlerTensor(
...     model_class=LinearRegression,
...     model_params={'input_dim': 1, 'output_dim': 1},
...     state_dict_path='gs://path/to/model.pt')

>>> with beam.Pipeline() as p:
...    predictions = (
...        p
...        | beam.Create(data)
...        | beam.Map(torch.Tensor) # Map np array to Tensor
...        | RunInference(model_handler=model_handler)
...        | beam.Map(print))
```

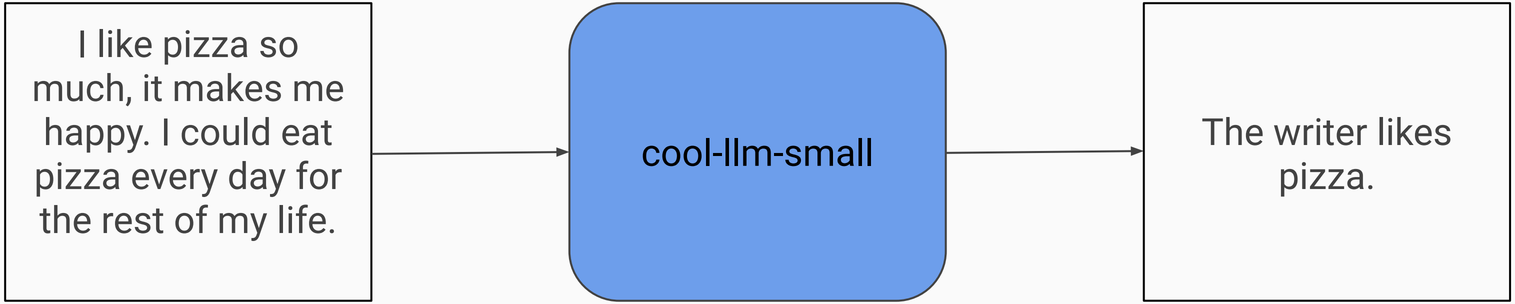
Basic Inference Demo

colab.sandbox.google.com/github/apache/beam/blob/master/examples/notebooks/beam-ml/run_inference_huggingface.ipynb

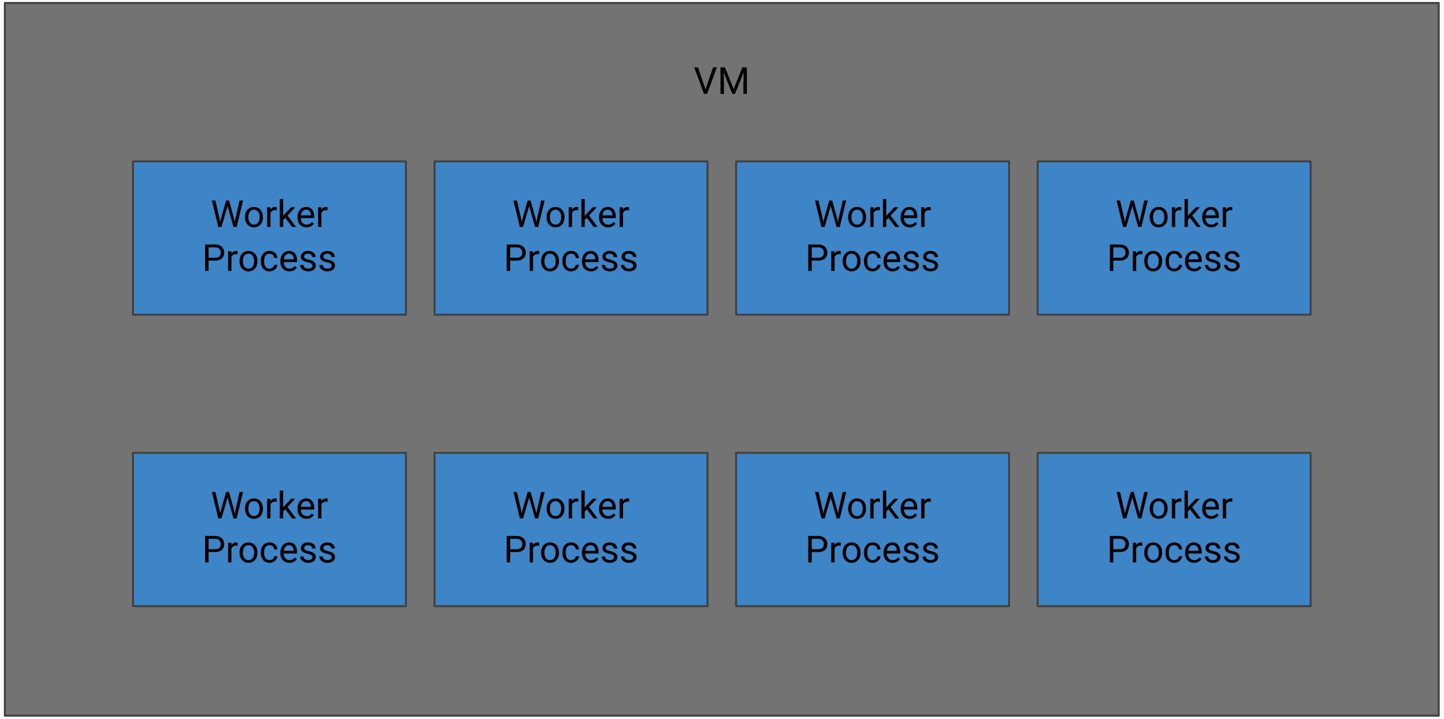
Large Models

Motivating Use Case

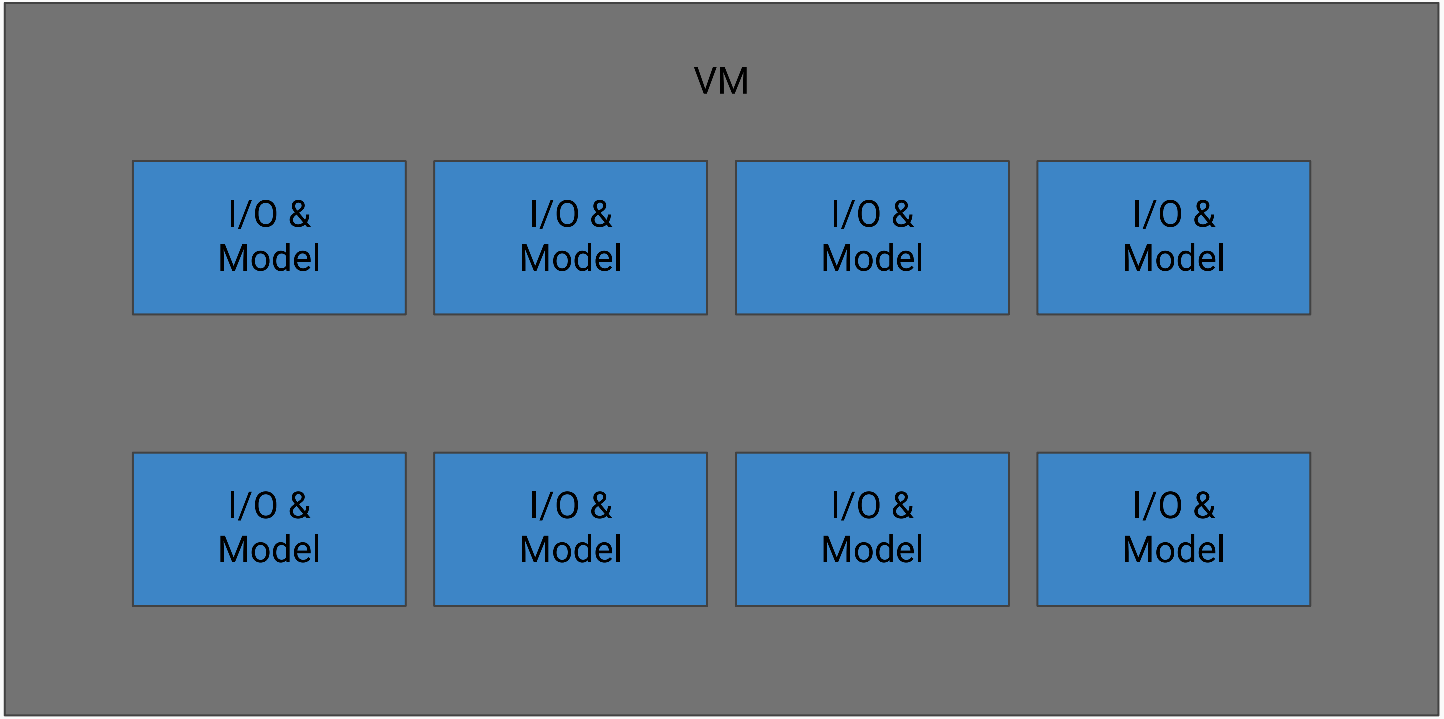
- Text summarization with a small-medium sized LLM
- Can fit many copies of model in memory



Distributed Runner Architecture*



Ideal small model configuration

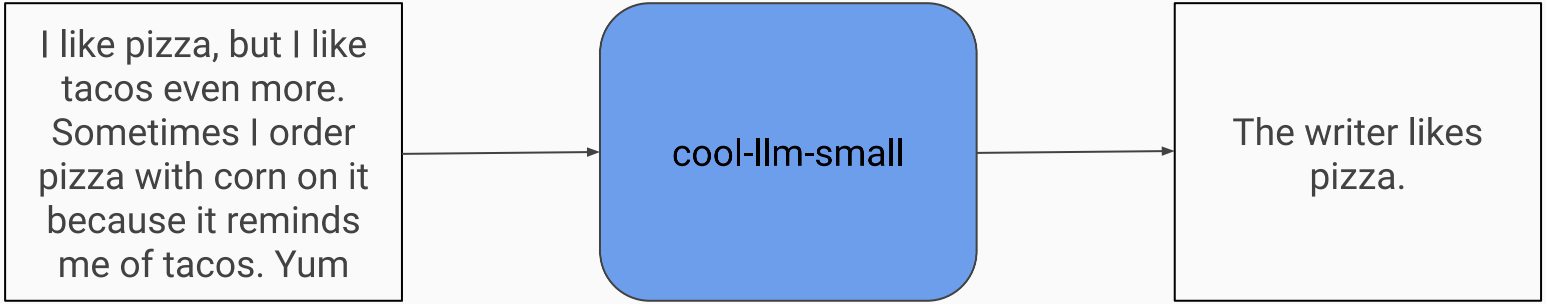


Default configuration: share model across threads

- Aka the easy case
- Uses Beam's shared.py module

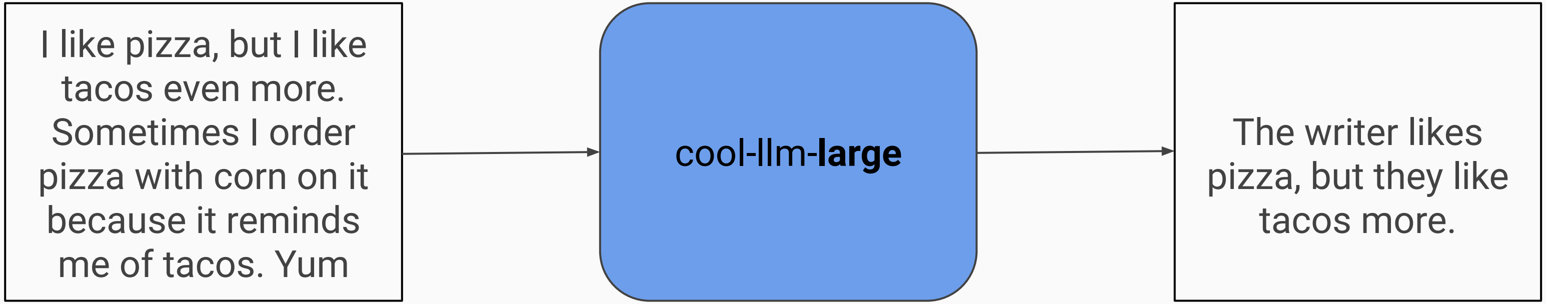
Motivating Use Case - revisited

- Text summarization with a small-medium sized LLM **isn't good enough**

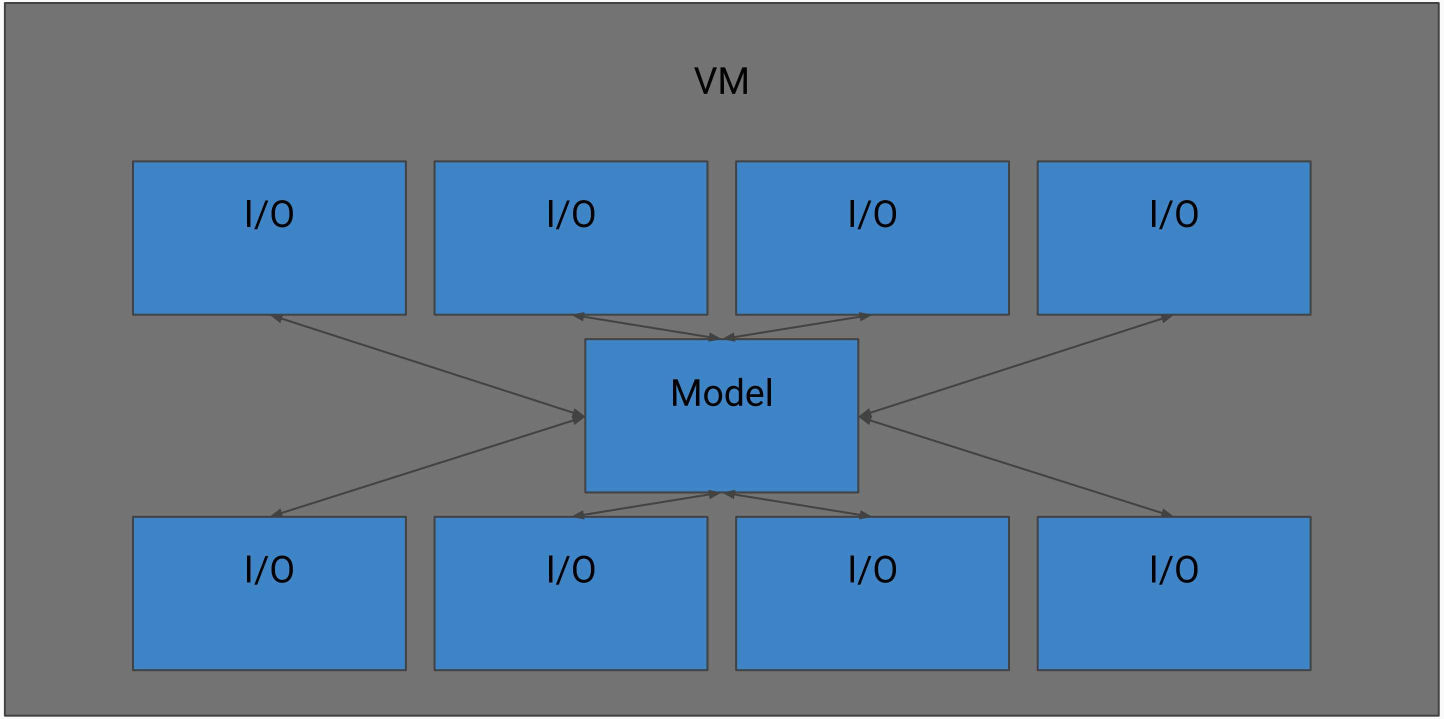


Motivating Use Case - revisited

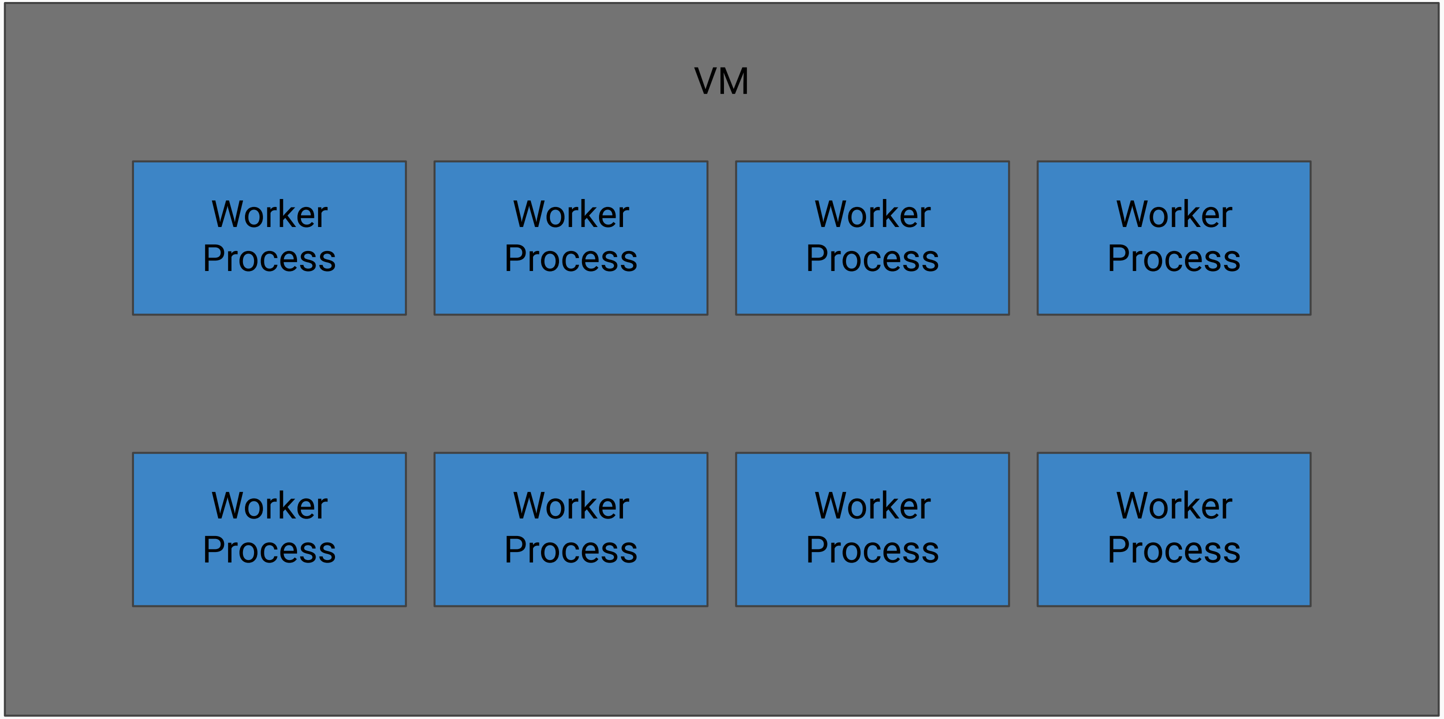
- Many options, one is to switch to a larger model
- Can only fit one (or few) copies in memory



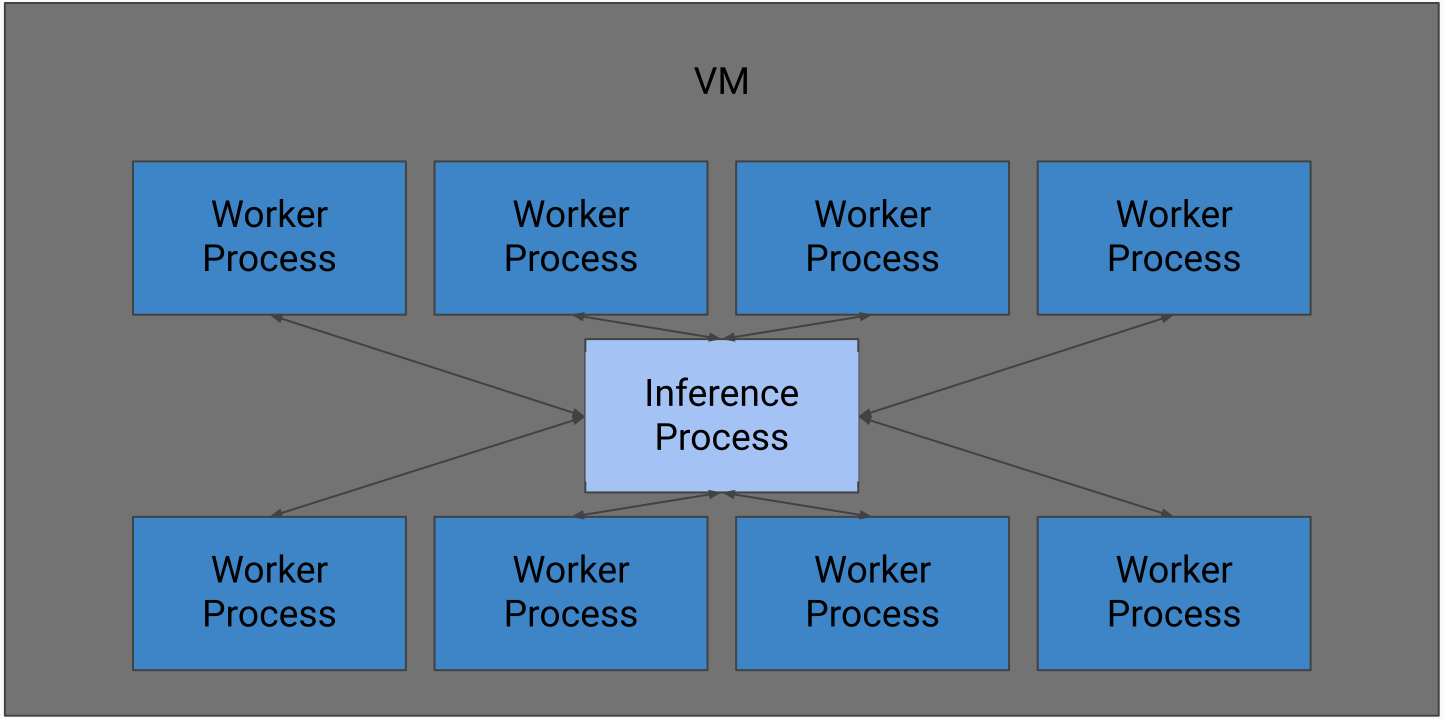
Ideal Large Model Configuration



How do we map ideal model configurations to this?



Ideal Large Model Configuration



Optional: serve a single model for all processes

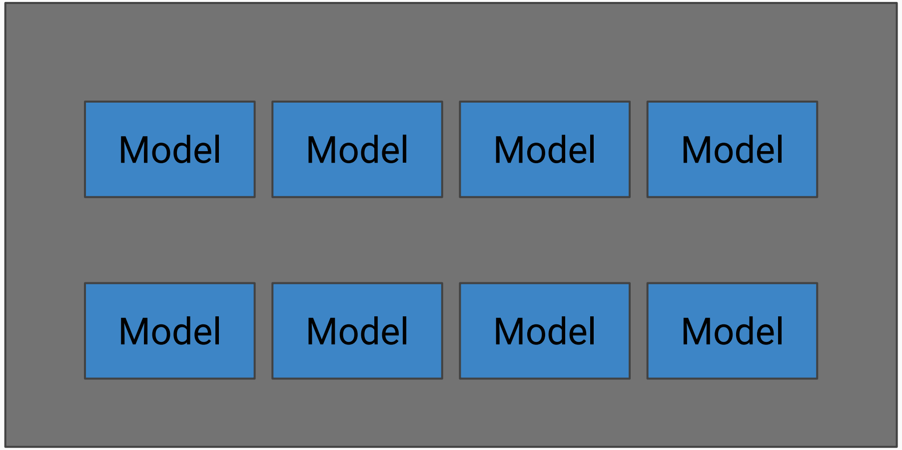
- Reduce memory at cost of interprocess communication, minimized parallelism
- Uses beam’s multi_process_shared library

Sounds like a lot of work to switch serving configurations, right?

- If you’re spinning this yourself, you need to set up a new serving topology, but Beam can make it easy

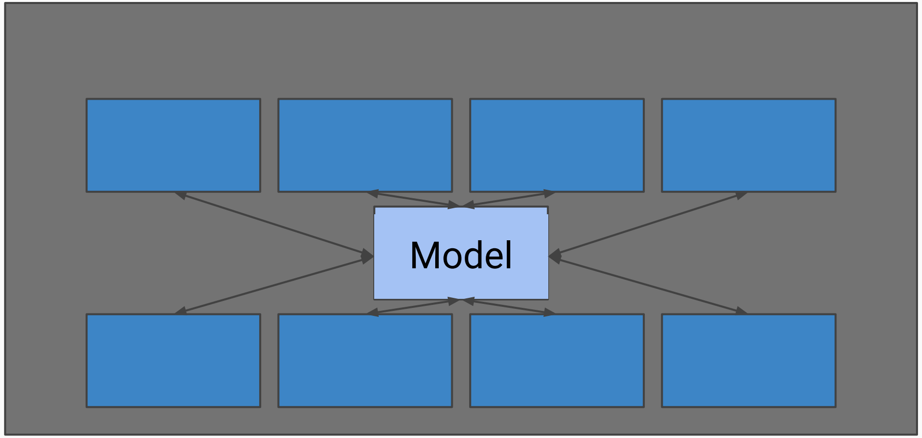
Built in model handler using default configuration

```
>>> model_handler = PytorchModelHandlerTensor(
...     model_class=LinearRegression,
...     model_params={'input_dim': 1, 'output_dim': 1},
...     state_dict_path='gs://path/to/model.pt')

>>> pcoll | RunInference(model_handler=model_handler)
```



Built in model handler using large model configuration

```
>>> model_handler = PytorchModelHandlerTensor(
...     model_class=LinearRegression,
...     large_model=True,
...     model_params={'input_dim': 1, 'output_dim': 1},
...     state_dict_path='gs://path/to/model.pt')

>>> pcoll | RunInference(model_handler=model_handler)
```



Custom Model Handler configuration (default, share across threads)

```
>>> def run_inference(model, batch, …):
...   model.predict(batch)
```

Custom Model Handler configuration (large model configuration)

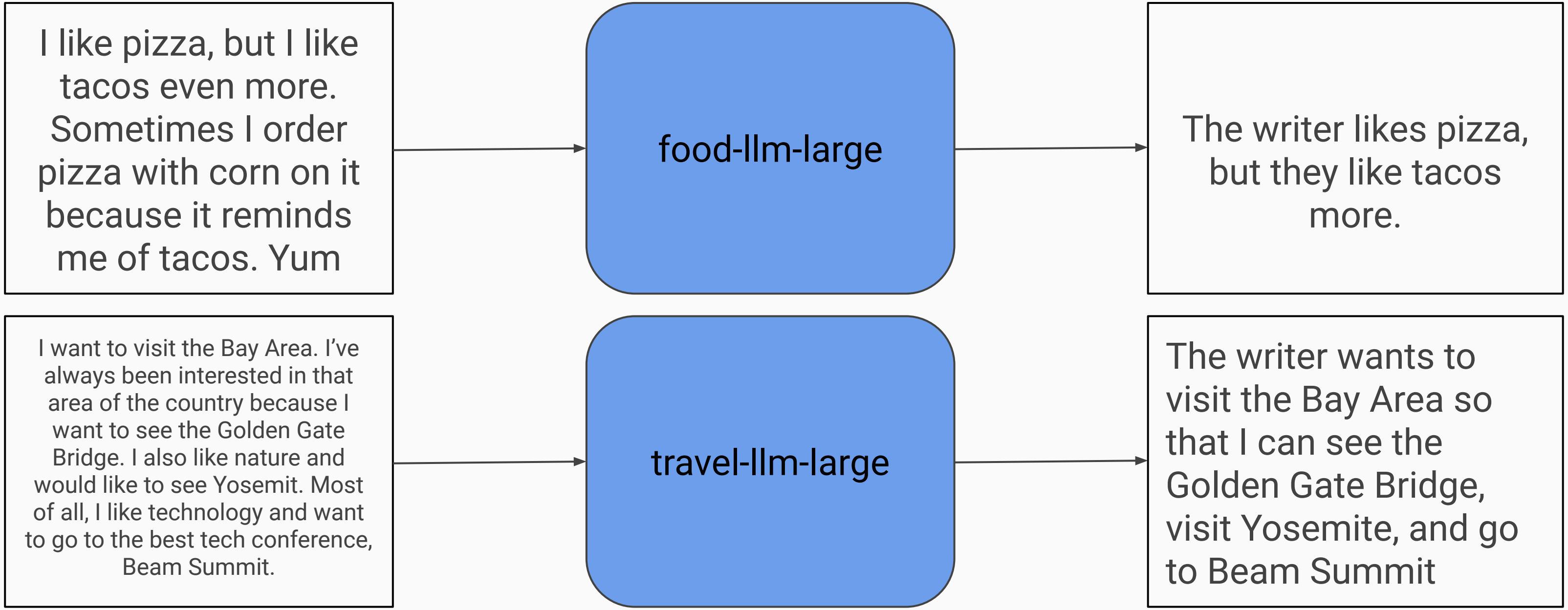
```
>>> def run_inference(model, batch, …):
...   model.predict(batch)

>>> def share_model_across_processes(self) -> bool:
...   return true
```

Custom Model Handler configuration (medium model configuration)

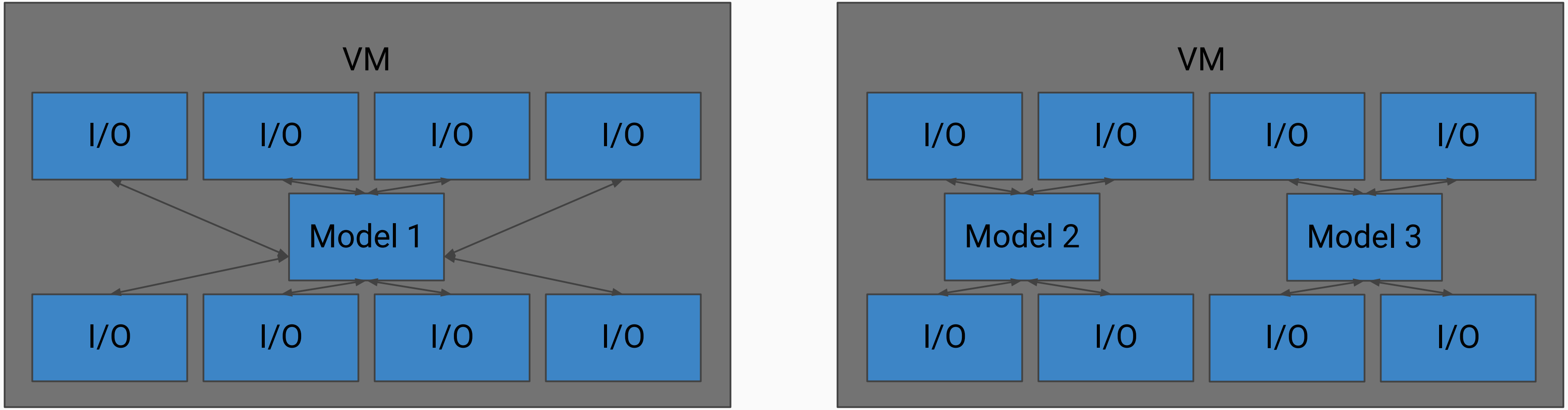
```
>>> def run_inference(model, batch, …):
...   model.predict(batch)

>>> def share_model_across_processes(self) -> bool:
...   return true

>>> def model_copies(self) -> int:
...   return 4
```

Motivating Use Case - revisited

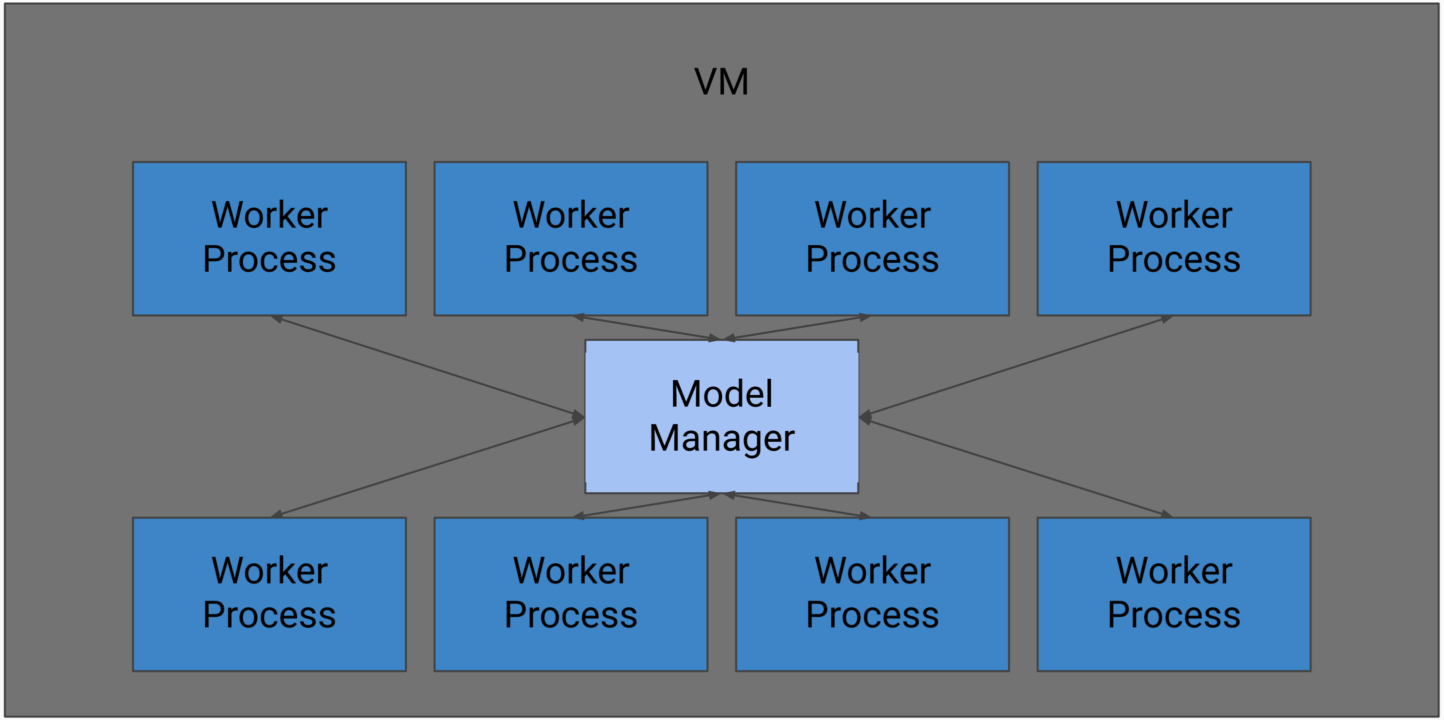
- What if I need a model per customer?



Ideal Multi Large Model Configuration



Ideal Large Model Configuration



Optional: serve a single model for all processes

- Model Manager empowered to load/unload models in order to make optimal use of memory

Again, this is just configuration

```
>>> mh1 = PytorchModelHandlerTensor(
...   model_class=LinearRegression,
...   model_params={'input_dim': 1, 'output_dim': 1},
...   state_dict_path='gs://path/to/model.pt')
```

```
>>> pcoll | RunInference(mh1)
```

Again, this is just configuration

```
>>> mh1 = PytorchModelHandlerTensor(
...   model_class=LinearRegression,
...   model_params={'input_dim': 1, 'output_dim': 1},
...   state_dict_path='gs://path/to/model.pt')
>>> mh2 = <...>

>>> per_key_mhs = [
...   KeyModelMapping(['key1', 'key2', 'key3'], mh1),
...   KeyModelMapping(['foo', 'bar', 'baz'], mh2)]
>>> mh = KeyedModelHandler(per_key_mhs)

>>> pcoll | RunInference(mh)
```

Per Key Model Demo

colab.sandbox.google.com/github/apache/beam/blob/master/examples/notebooks/beam-ml/per_key_models.ipynb

Specialty Hardware

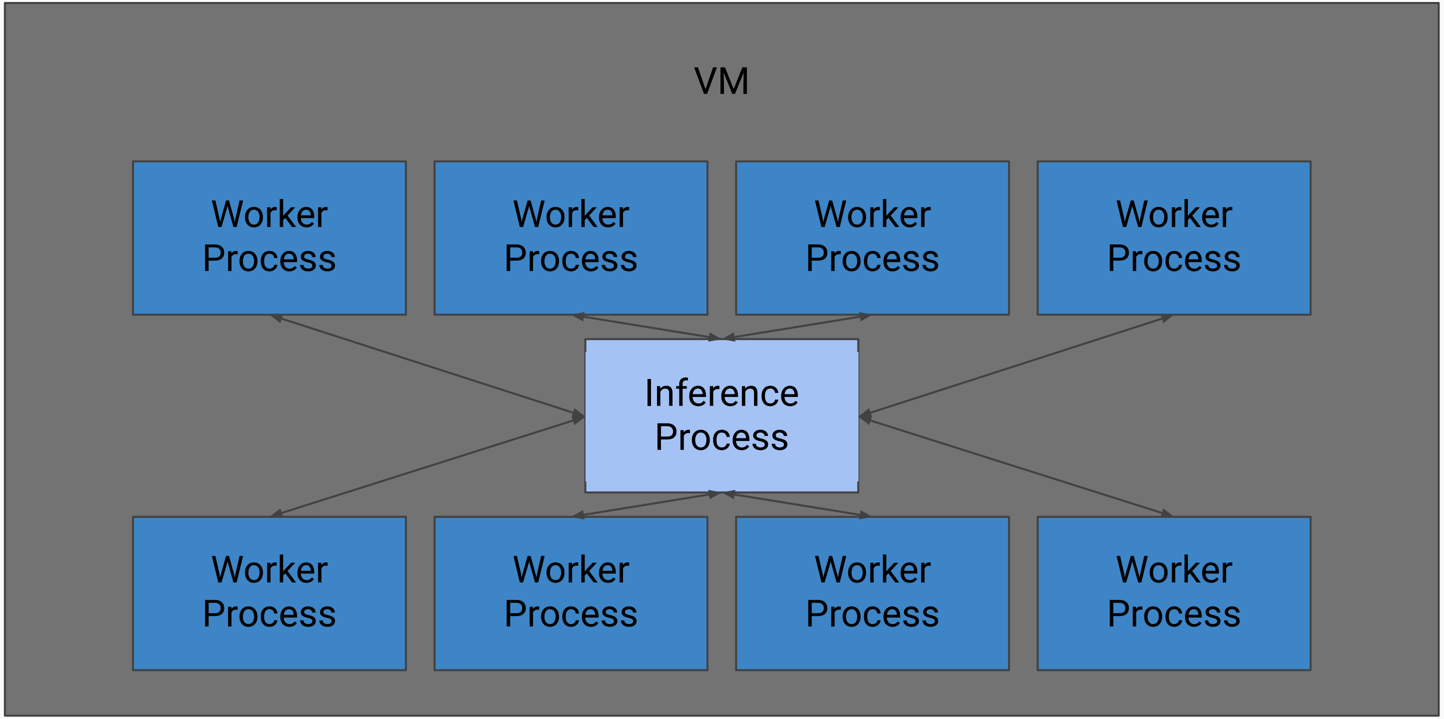
GPU/TPU Support

- Hardware availability dependent on runner
- Beam has some primitives that help

Beam Primitives for GPUs

- Resource hints for heterogeneous pools
- Built in detection + framework specific responses to GPUs at the ModelHandler level
- Large model setting helps

Central Inference Process provides a single point of interaction with GPU



Try it yourself

https://github.com/apache/beam/tree/master/examples/notebooks/beam-ml

Thank you!

Questions?

dannymccormick@google.com

Github - damccorm

https://www.linkedin.com/in/danny-mccormick-a044b1103/