

Working with Tensorflow from the JVM

How Big Data and Deep Learning can be BFFs
[@holdenkarau](https://twitter.com/holdenkarau)



Holden:

- My name is Holden Karau
- Preferred pronouns are she/her
- Developer Advocate at Google
- Apache Spark PMC, Beam contributor
- previously IBM, Alpine, Databricks, Google, Foursquare & Amazon
- co-author of Learning Spark & High Performance Spark
- Twitter: [@holdenkarau](https://twitter.com/holdenkarau)
- Slide share <http://www.slideshare.net/hkarau>
- Code review livestreams: <https://www.twitch.tv/holdenkarau> / <https://www.youtube.com/user/holdenkarau>
- Spark Talk Videos <http://bit.ly/holdenSparkVideos>





Who is Boo?



@booprogrammer
Drawn by @impurepics

- Boo uses she/her pronouns (as I told the Texas house committee)
- Best doge
- Lot's of experience barking at computers to make them go faster
- Author of “Learning to Bark” & “High Performance Barking”
- On twitter [@BooProgrammer](https://twitter.com/BooProgrammer)

Why glorious employer (Google) cares

- We have lots of JVM Big Data tools in our ecosystems
 - Apache Beam w/Dataflow, Apache Spark w/Dataproc and more!
- Also GKE can run Apache Spark and Apache Flink, etc.
 - The Spark side still needs some work, but were getting there
- Something something cloud
- This is not like an official position per-se



Who I think you wonderful humans are?

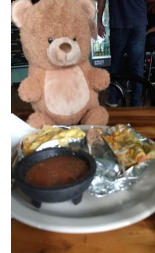
- Nice enough people
- Don't mind pictures of cats
- Probably pretty familiar with Spark
- Maybe somewhat familiar with Beam?



What will be covered?

- Quick: Big Data Outside the JVM in general
- TensorFlowOn{Spark, Beam+{Spark**, Flink**}}
- Apache Arrow - How this changes “everything”*
- Where we are today in non-JVM support in Beam
 - And why this matters for Tensorflow

** Doesn't like *work* yet.



PySpark



- The Python interface to Spark
- Same general technique used as the bases for the C#, R, Julia, etc. interfaces to Spark
- Fairly mature, integrates well-ish into the ecosystem, less a Pythonrific API
- Has some serious performance hurdles from the design

A quick detour into PySpark's internals



+

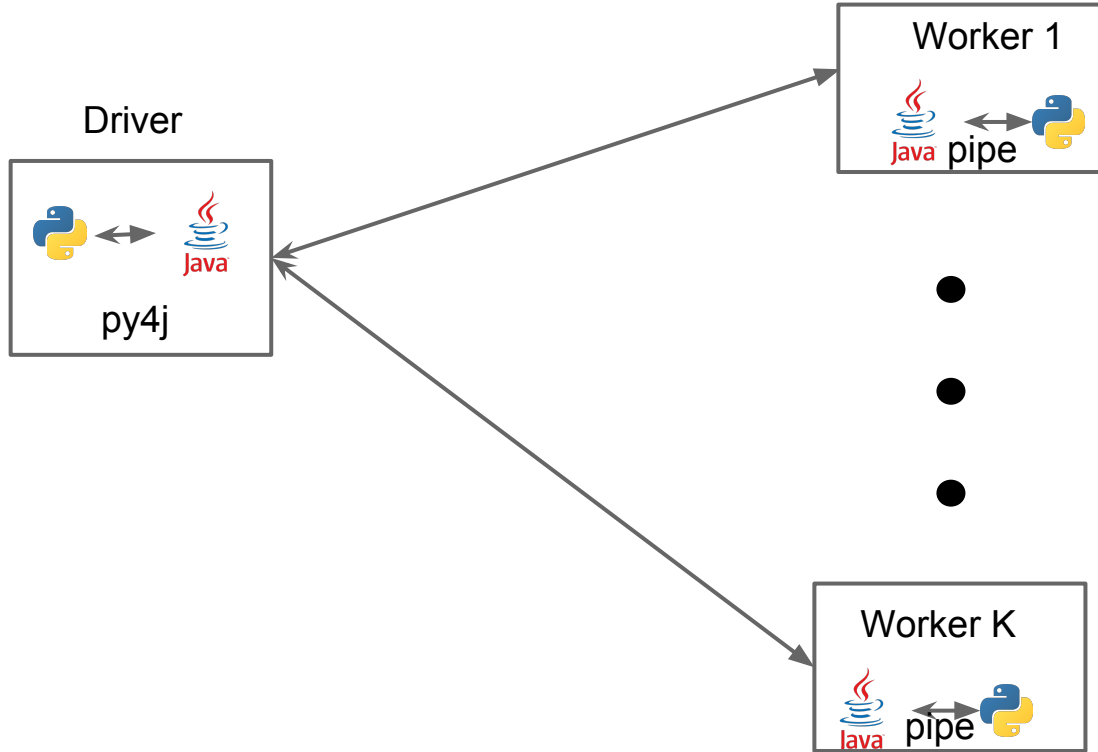


+

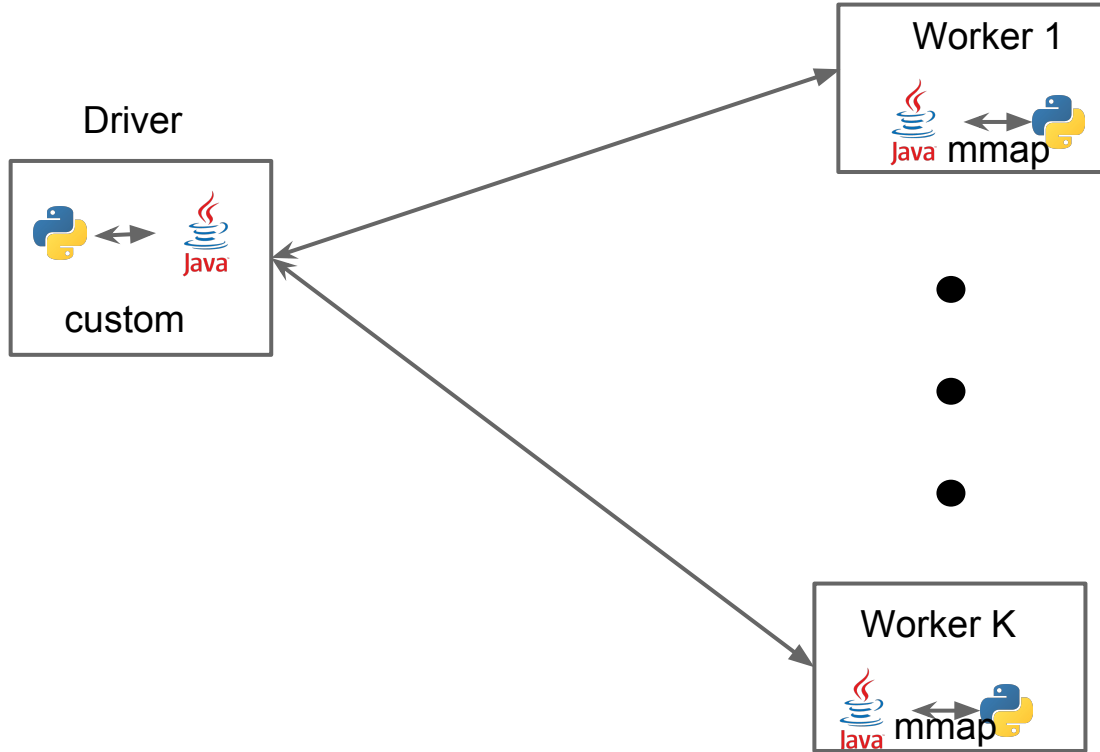
JSON



So what does that look like?



And in flink....



So how does that impact Py[X]

\forall \text{all } X \text{ in } \{\text{Big Data}\}-\{\text{Native Python Big Data}\}



- Double serialization cost makes everything more expensive
- Python worker startup takes a bit of extra time
- Python memory isn't controlled by the JVM - easy to go over container limits if deploying on YARN or similar
- Error messages make ~0 sense
- Dependency management makes limited sense
- features aren't automatically exposed, but exposing them is normally simple

What's the rest of big data outside the JVM look like?



Most of the tools are built in the JVM, so how do we play together?

- Pickling, Strings, JSON, XML, oh my!
- Unix pipes
- Sockets

What about if we don't want to copy the data all the time?

- Or standalone “pure”* re-implementations of everything
 - Reasonable option for things like Kafka where you would have the I/O regardless.
 - Also cool projects like dask (pure python) -- but hard to talk to existing ecosystem

TensorFlowOnSpark, everyone loves mnist!



```
cluster = TFCluster.run(sc, mnist_dist_dataset.map_fun, args,  
args.cluster_size, num_ps, args.tensorboard,  
TFCluster.InputMode.SPARK)  
if args.mode == "train":  
    cluster.train(dataRDD, args.epochs)
```

The “future”*: faster interchange



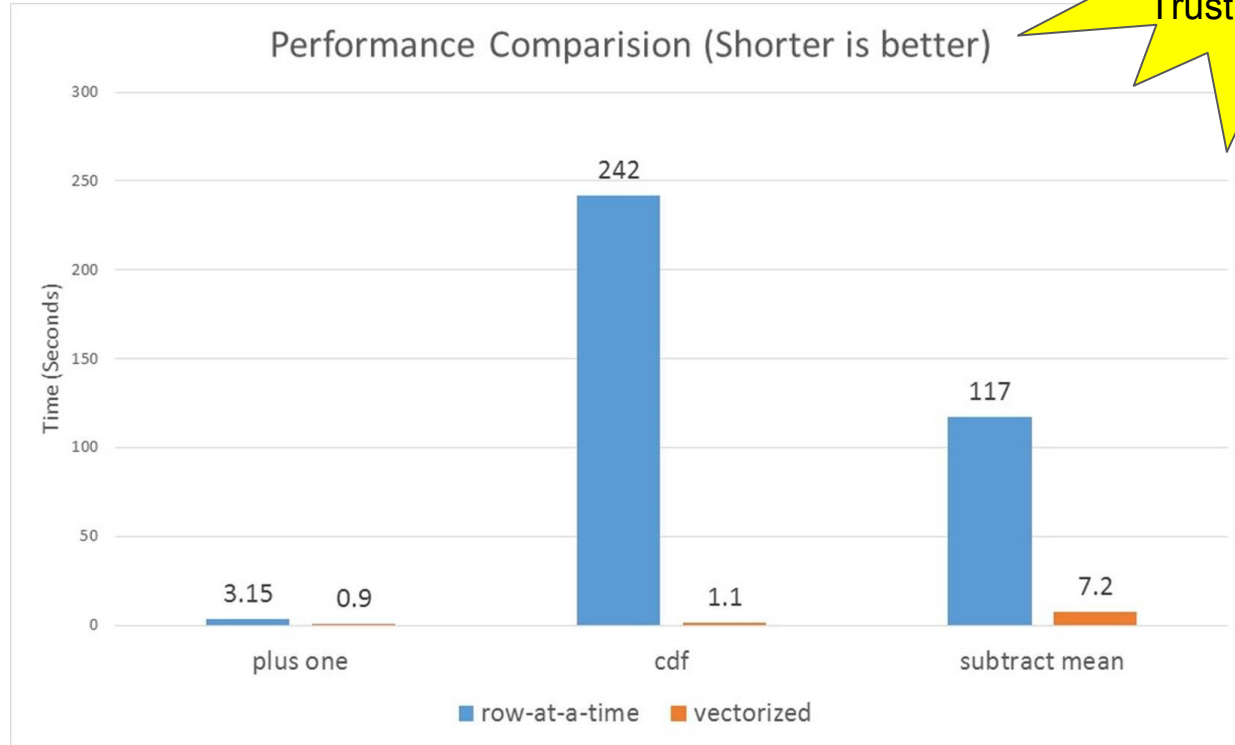
- By future I mean availability today but running it in production is “adventurous”
- Unifying our cross-language experience
 - And not just “normal” languages, CUDA counts yo



*Arrow: Spark 2.3 and beyond & GPUs & R & Python &

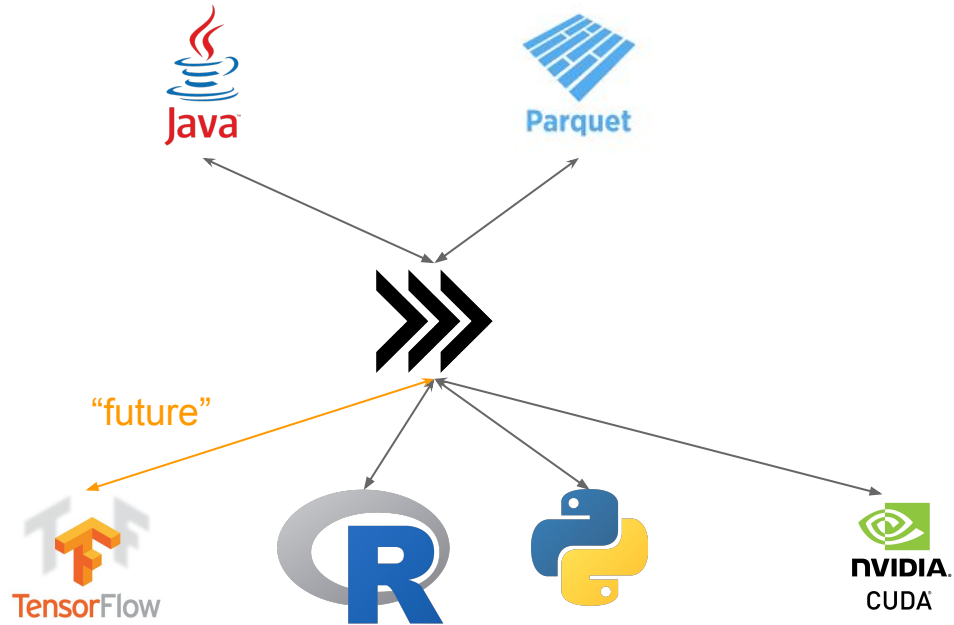
What does the future look like?*

*Vendor benchmark.
Trust but verify.



*Source: <https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html>.

Arrow (a poorly drawn big data view)

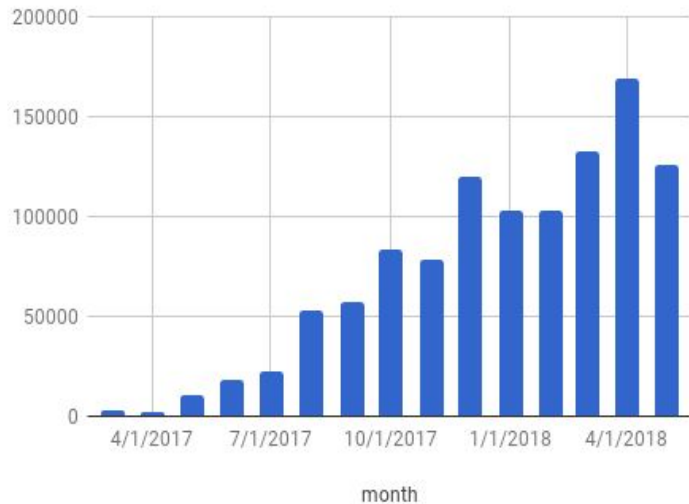


Logos trademarks of their respective projects

Apache Arrow Adoption



PyArrow Monthly Downloads



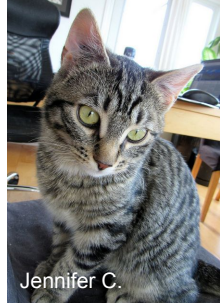
[@kstirman](#)

Rewriting your code because why not

```
spark.catalog.registerFunction(  
    "add", lambda x, y: x + y, IntegerType())
```

=>

```
add = pandas_udf(lambda x, y: x + y, IntegerType())
```



And we can do this in TFOOnSpark*:



```
unionRDD.foreachPartition(TFSparkNode.train(self.cluster_info,  
self.cluster_meta, qname))
```

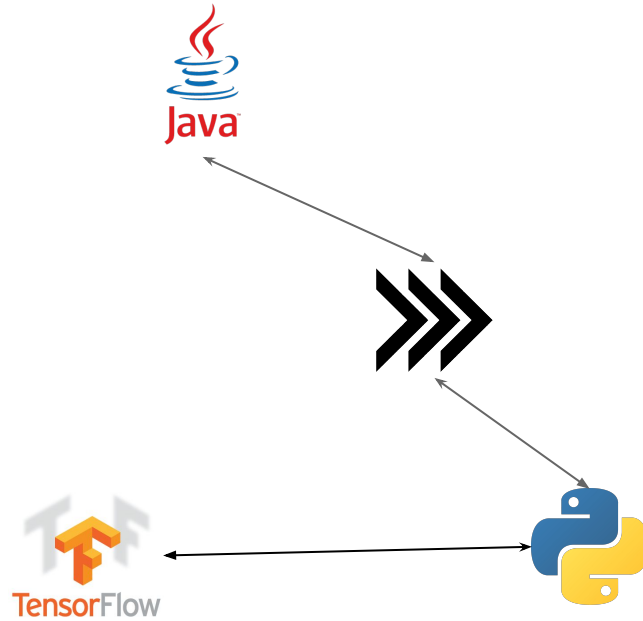
Will Transform Into something magical (aka fast but unreliable)
on the next slide!

Which becomes



```
train_func = TFSparkNode.train(self.cluster_info,
self.cluster_meta, qname)
@pandas_udf("int")
def do_train(inputSeries1, inputSeries2):
    # Sad hack for now
    modified_series = map(lambda x: (x[0], x[1]),
zip(inputSeries1, inputSeries2))
    train_func(modified_series)
    return pandas.Series([0] * len(inputSeries1))
```

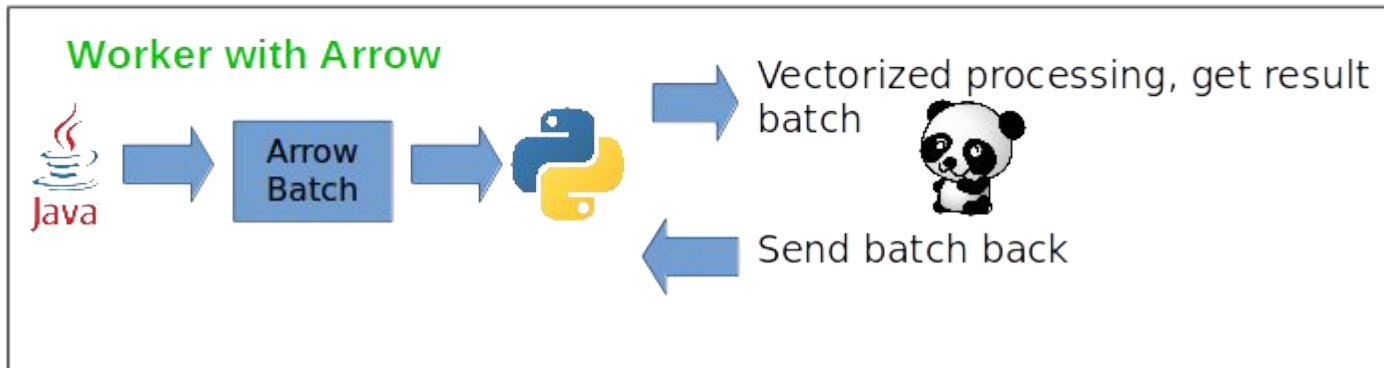
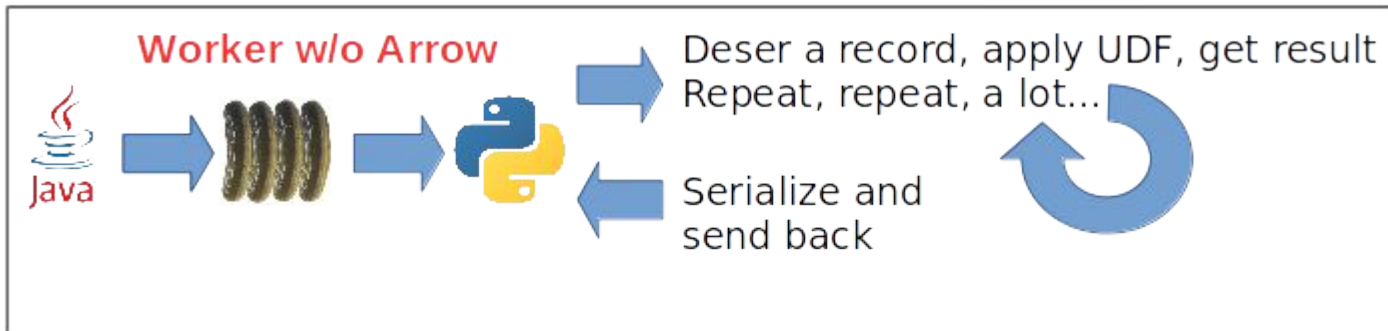
And this now looks like:



Logos trademarks of their respective projects



How a Python worker looks with Arrow



TFOnSpark Possible vNext+1?



- Avoid funneling the data through Python native types
 - For now the Spark Arrow UDFS aren't perfect for this
 - But we can (and are) improving them
- mmaped Arrow?
- Skip Python on the workers handling data entirely (idk I'm lazy so probably not)

Multi-language pipelines?!?



- Totally! I mean kind-of. Spark is ok right?
 - Beam is working on it :) #comejoinus
 - Easier for Python calling Java
 - If SQL counts for sure. If not, yes*.
- Technically what happens when you use Scala Spark + Tensorflow
- And same with Beam, etc.
- Right now painful to write in practice in OSS land outside of libraries (commercial vendors have some solutions but I stick to OSS when I can)



What is/why Sparkling ML

- A place for useful Spark ML pipeline stages to live
 - Including both feature transformers and estimators
- The why: Spark ML can't keep up with every new algorithm
- Lots of cool ML on Spark tools exist, but many don't play nice with Spark ML or together.
- We make it easier to expose Python transformers into Scala land and vice versa.
- Our repo is at: <https://github.com/sparklingpandas/sparklingml>

So what goes in startup.py?



- A class for our Java code to call with parameters & request functions
- Code to take the Python UDFS and construct/return the underlying Java UDFS
- A main function to startup the Py4J gateway & Spark context to serialize our functions in the way that is expected
- Pretty much it's just boilerplate but [you can take a look if you want.](#)

So what goes in startup.py?



```
class PythonRegistrationProvider(object):
```

```
class Java:
```

```
    package = "com.sparklingpandas.sparklingml.util.python"  
    className = "PythonRegistrationProvider"  
    implements = [package + "." + className]
```

So what goes in startup.py?



Jennifer C.

```
def registerFunction(self, ssc, jsession, function_name,
params):
    setup_spark_context_if_needed()
    if function_name in functions_info:
        function_info = functions_info[function_name]
        evaledParams = ast.literal_eval(params)
        func = function_info.func(*evaledParams)
        udf = UserDefinedFunction(func, ret_type,
make_registration_name())
        return udf._judf
    else:
        print("Could not find function")
```

What's the boilerplate in Java?

- Call Python
- A trait representing the Python entry point
- Wrapping the UDFS in Spark ML stages (optional buuut nice?)
- Also kind of boring, its [in a few files if you want to look.](#)

Ok first more wordcount



With Spacy now! Non-English language support!

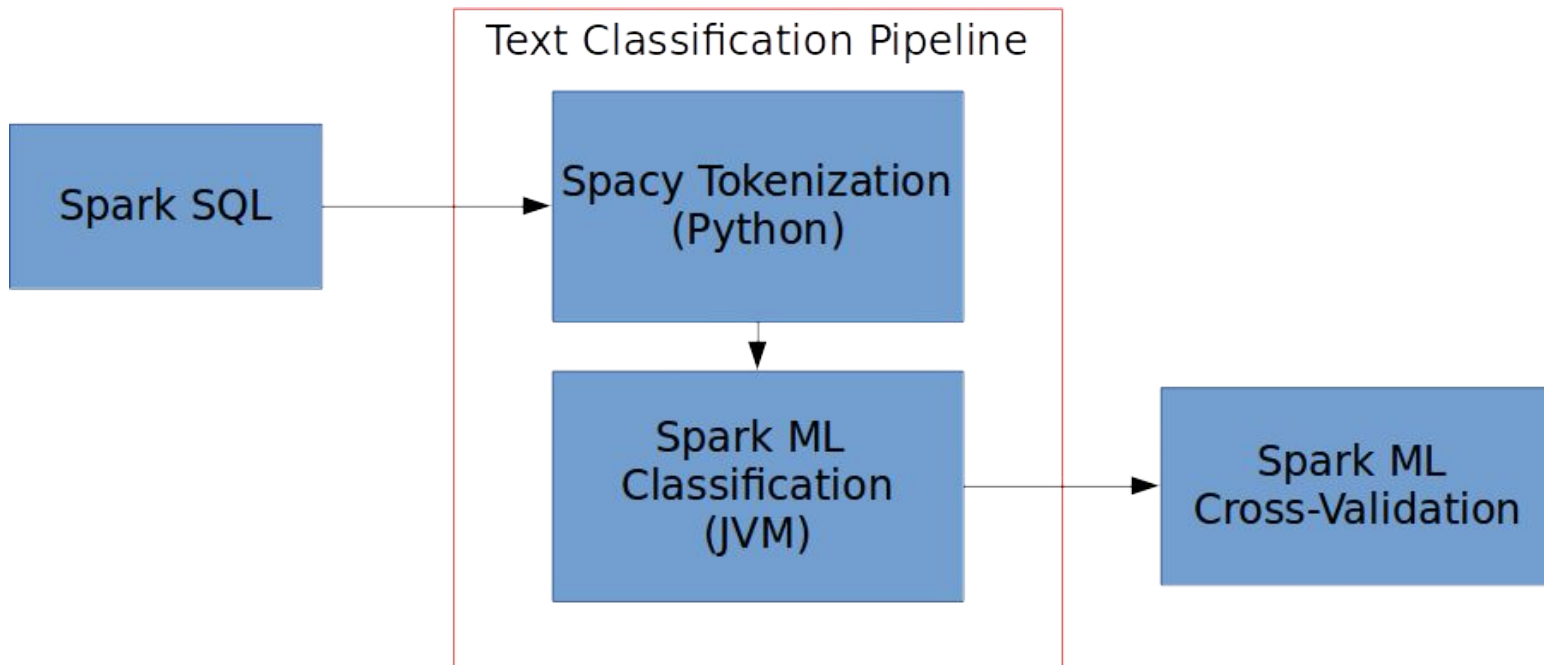
```
def inner(inputSeries):  
    """Tokenize the inputString using spacy for  
    the provided language."""  
    nlp = SpacyMagic.get(lang)  
  
    def tokenizeElem(elem):  
        return list(map(lambda token: token.text,  
list(nlp(unicode(elem)))))  
  
    return inputSeries.apply(tokenizeElem)
```


And from the JVM:

```
val transformer = new SpacyTokenizePython()
transformer.setLang("en")
val input = spark.createDataset(
    List(InputData("hi boo"), InputData("boo")))
transformer.setInputCol("input")
transformer.setOutputCol("output")
val result = transformer.transform(input).collect()
```



Sparkling ML Mixed Language Pipeline



SparkDeepLearning Pipelines



- Easier to work with than TensorFlowOnSpark for many people
- Good for “transfer learning”, not for completely new models
 - This thing is good at finding cats, maybe we can find dogs with it!
- Also powered by DataFrames (yay!)
- We can do the same tricks to make it work with Arrow
- You can use it with Spark’s feature prep stages if you try hard enough
- We can expose it in Scala in a few ways....
 - See “Deploying models as SQL functions” + Sparkling ML + PyArrow :p
 - Future/other ways: Tensorflow Scala + compile graph + pass down?

SparkDeepLearning + Sparkling ML*

```
class DLImagePredictor(BasicTransformationFunction):  
  
    @classmethod  
    def func(cls, *args):  
        deepimagepredictor = DeepImagePredictor(*args)  
        class MyFunction(LegacyTransformationFunction):  
  
            def transform_df(self, jdf):  
                df = javadf_to_pydf(jdf)  
                return deepimagepredictor.transform(df)._jdf
```

*Emphasis on the * here, needs some more details

Scala side*

```
class SDLImagePredictorPython(override val uid: String) extends
BasicPythonTransformer {
```

```
    final val modelName = new Param[String](this, "modelName",
"model")
```

```
    /** @group getParam */
```

```
    final def getModelName: String = $(modelName)
```

*Emphasis on the * here, needs some more details

Scala side*

```
final def setName(value: String): this.type =  
set(this.modelName, value)
```

```
def this() =  
this(Identifiable.randomUUID("SDLImagePredictorPython"))
```

```
override val pythonFunctionName = "sdldip"
```

```
override def copy(extra: ParamMap) = {  
  defaultCopy(extra)  
}
```

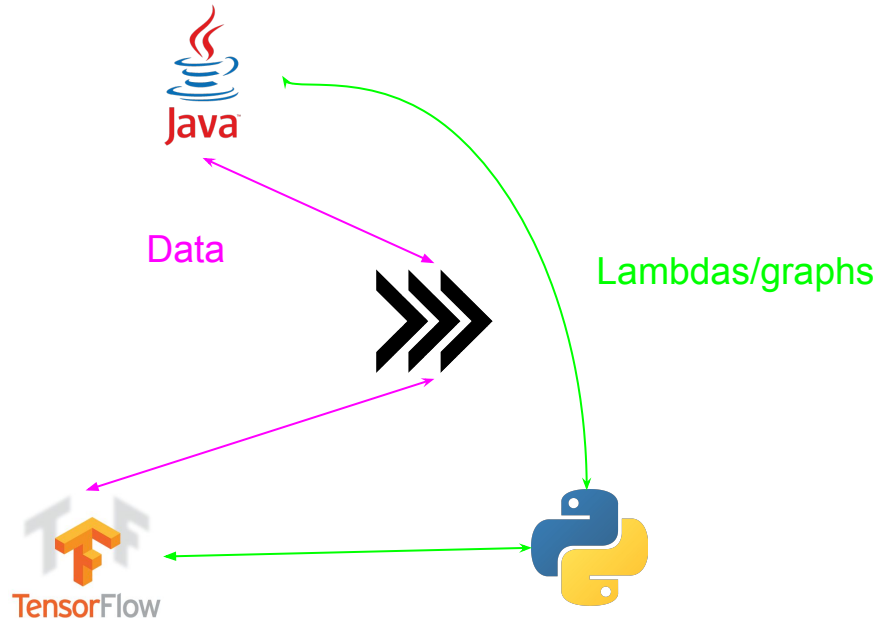
*Emphasis on the * here, needs some more details

Scala side*

```
def miniSerializeParams() = {  
    "[\"" + $(inputCol) + "\",\"" + $(outputCol) + "\",\"" +  
$(modelName) + "\"]"  
}
```

*Emphasis on the * here, needs some more details

A possible future



Logos trademarks of their respective projects

DL4J & Friends



- Totally an option too
- I'm like a solid 50% sure they support Arrow some of the time (although not completely sure - [take a look in the repo](#)).
- More than just transfer learning

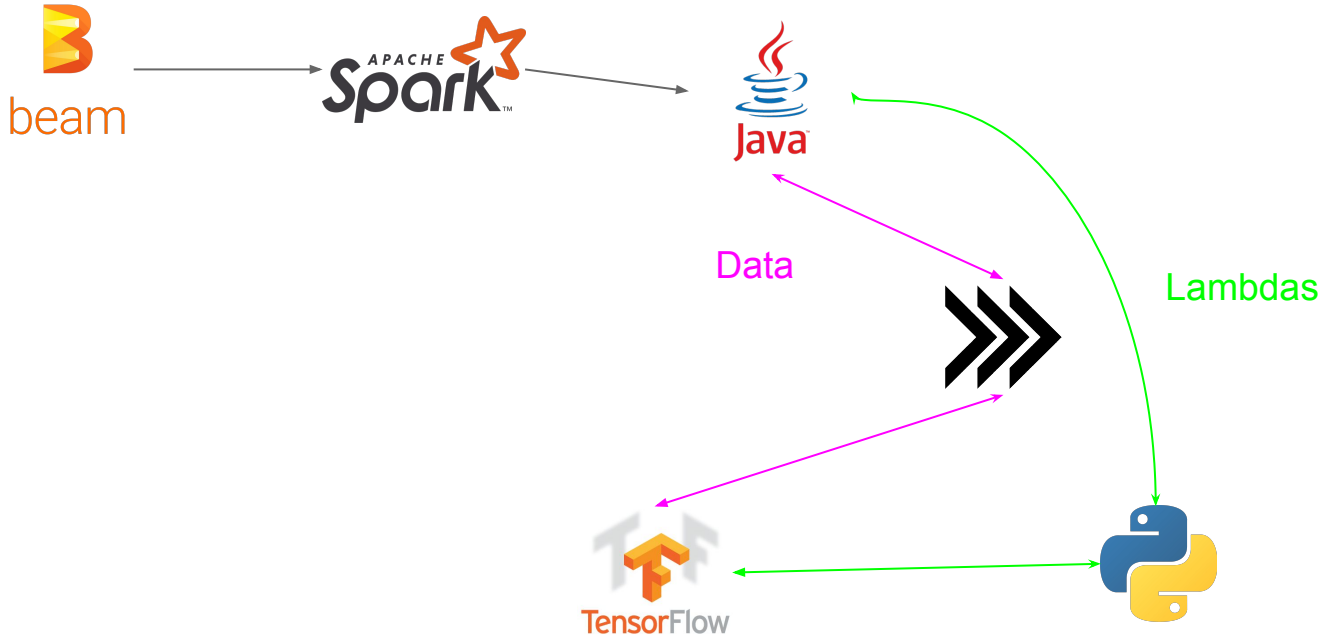
TensorFlow isn't enough on its own



PROJennifer C.

- Enter [TFX](#) & friends like [Kubeflow](#)
 - Current related TFX OSS components: [TF.Transform](#) [TF.Serving](#) (with more coming)
- Alternatives: piles of custom code re-created at serving time.
 - Yay job security?

Another possible future



Logos trademarks of their respective projects

TF.Transform



PROKathryn Yengel

- For pre-processing of your data
 - e.g. where you spend 90% of your dev time anyways
- Integrates into serving time :D
- OSS
- Runs on top of Apache Beam but doesn't (currently) work outside of GCP
 - [#wereworkingonitipromise](#)

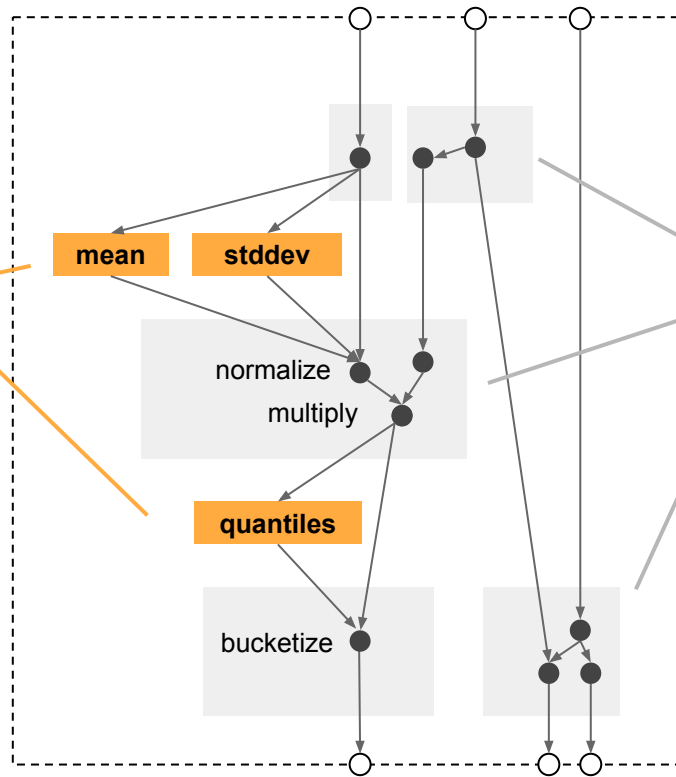
Defining a Transform processing function

```
def preprocessing_fn(inputs):  
    x = inputs['x']  
    y = inputs['y']  
    s = inputs['s']  
    x_centered = x - tft.mean(x)  
    y_normalized = tft.scale_to_0_1(y)  
    s_int = tft.string_to_int(s)  
    return { 'x_centered': x_centered,  
            'y_normalized': y_normalized, 's_int': s_int }
```

Analyzers

Reduce (full pass)

Implemented as a distributed data pipeline

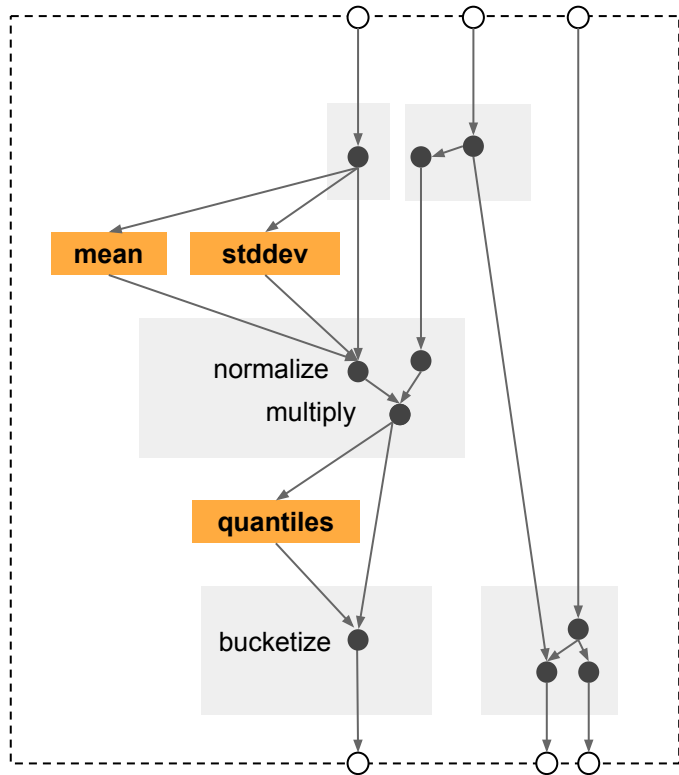


Transforms

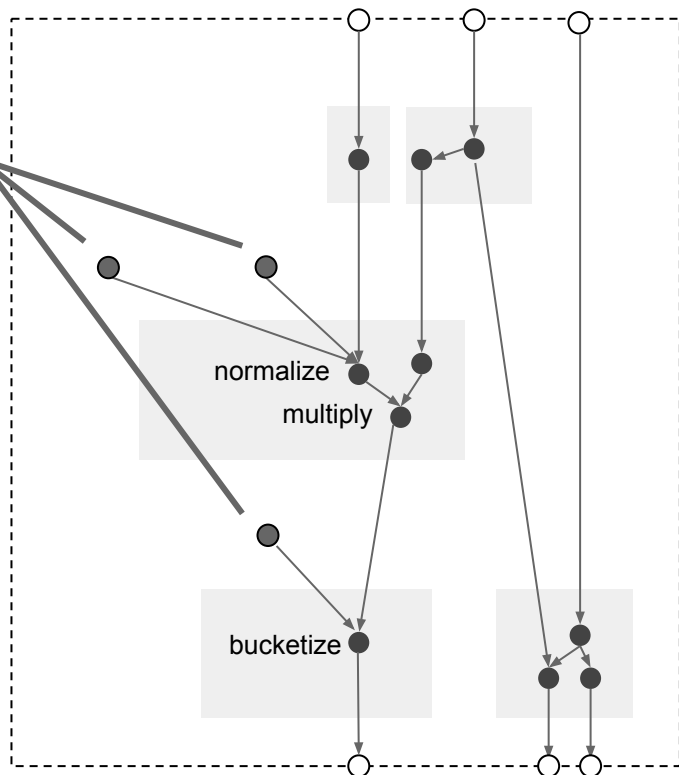
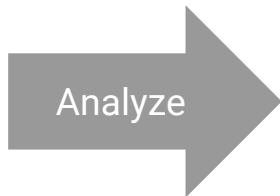
Instance-to-instance (don't change batch dimension)

Pure TensorFlow

data



constant tensors



Some common use-cases...

Scale to ...

```
tft.scale_to_z_score
```

```
...
```

Bucketization

```
tft.quantiles
```

```
tft.apply_buckets
```

Bag of Words / N-Grams

```
tf.string_split
```

```
tft.ngrams
```

```
tft.string_to_int
```

Feature Crosses

```
tf.string_join
```

```
tft.string_to_int
```


BEAM Beyond the JVM: Current release



- Non JVM BEAM doesn't work outside of Google's environment yet
- tl;dr : uses grpc / protobuf
 - Similar to the common design but with more efficient representations (often)
- But exciting new plans to unify the runners and ease the support of different languages (called SDKS)
 - See <https://beam.apache.org/contribute/portability/>
- If this is exciting, you can come join me on making BEAM work in Python3
 - Yes we still don't have that :(
 - But we're getting closer & you can come join us on [BEAM-2874](#) :D



beam

BEAM Beyond the JVM: Master branch

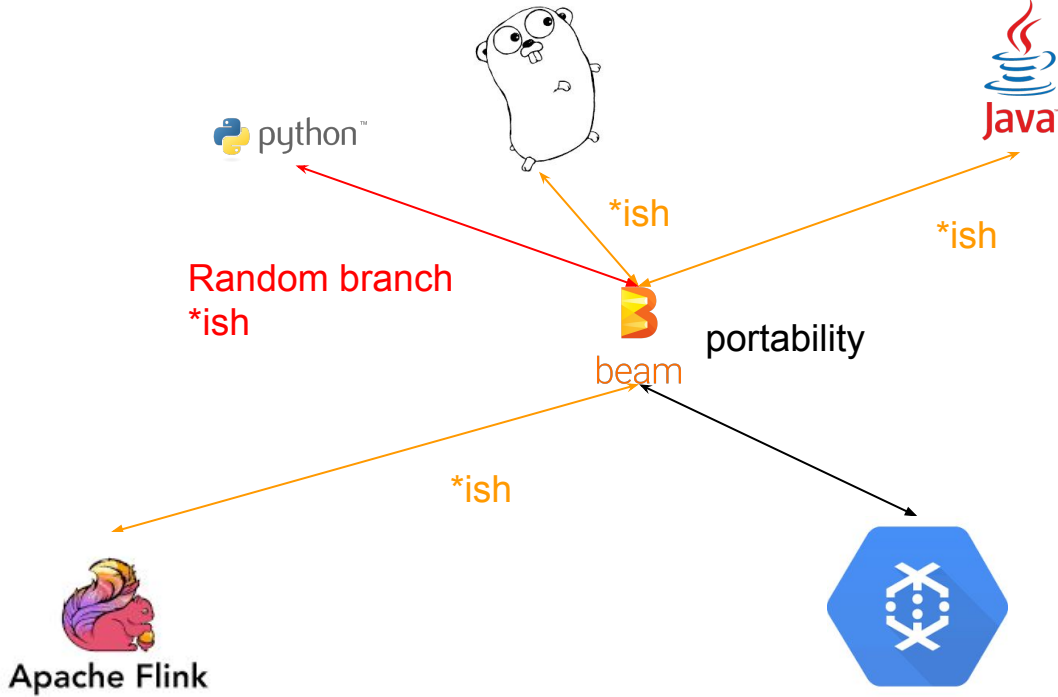
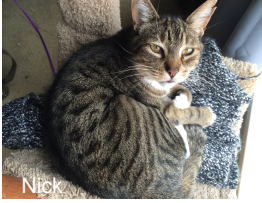


- Common interface for setting up jobs
- Portability framework allows SDK harnesses in arbitrary to be kicked off
- Runners ship in their own docker containers (goodbye dependency hell, hello container hell)
- Hacked up Python SDK to sort of talk to the new interface



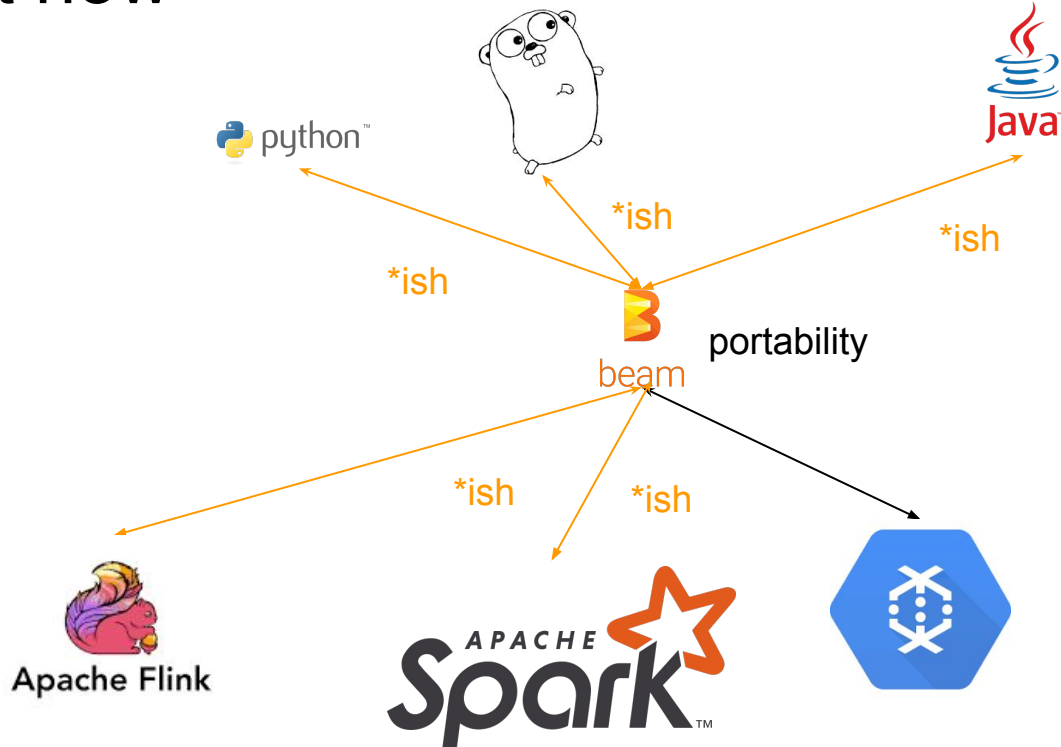
beam

BEAM Beyond the JVM: Master branch





BEAM Beyond the JVM: The “future” e.g. not now





So how TF does this relate to TF?

- Tensorflow is in Python (kind of)
- Once we finish the Python SDK on Beam on Flink adventure you can use all sorts of cool libraries (like TFX) to do your tensorflow work
 - You can use them today too if your use case is on [Dataflow](#)
- You will be able manage your dependencies
- You will be able to (in theory) re-use dataprep code at serving time
 - 80% less copy n' paste code with slight mistakes that get out of date!**
- No that doesn't work today
- Or tomorrow
- But... eventually
 - Standard OSS excuse “patches welcome” (sort of if you can find the branch :p)

**Not a guarantee, see your vendor for details.



beam

References



PROR. Crap Mariner

- TensorFlowOnSpark - <https://github.com/yahoo/TensorFlowOnSpark>
- Spark Deep Learning Pipelines - <https://github.com/databricks/spark-deep-learning>
- flink-tensorflow - <https://github.com/FlinkML/flink-tensorflow>
- TF.Transform - <https://github.com/tensorflow/transform>
- Beam portability design: <https://beam.apache.org/contribute/portability/>
- Beam on Flink + portability <https://issues.apache.org/jira/browse/BEAM-2889>
& <https://github.com/bsidhom/beam/tree/hacking-job-server>

High Performance Spark!

You can buy it today! On the internet!

Only one chapter focused on non-JVM stuff, I'm sorry. But don't let that stop you.

Cats love it*

*Or at least the box it comes in. If buying for a cat, get print rather than e-book.



And some upcoming talks:



- June

- FOSS Backstage - Dealing with contributor overload
- Scala Days NYC - Missed out on Scala Days EU? Come to NYC!

- July

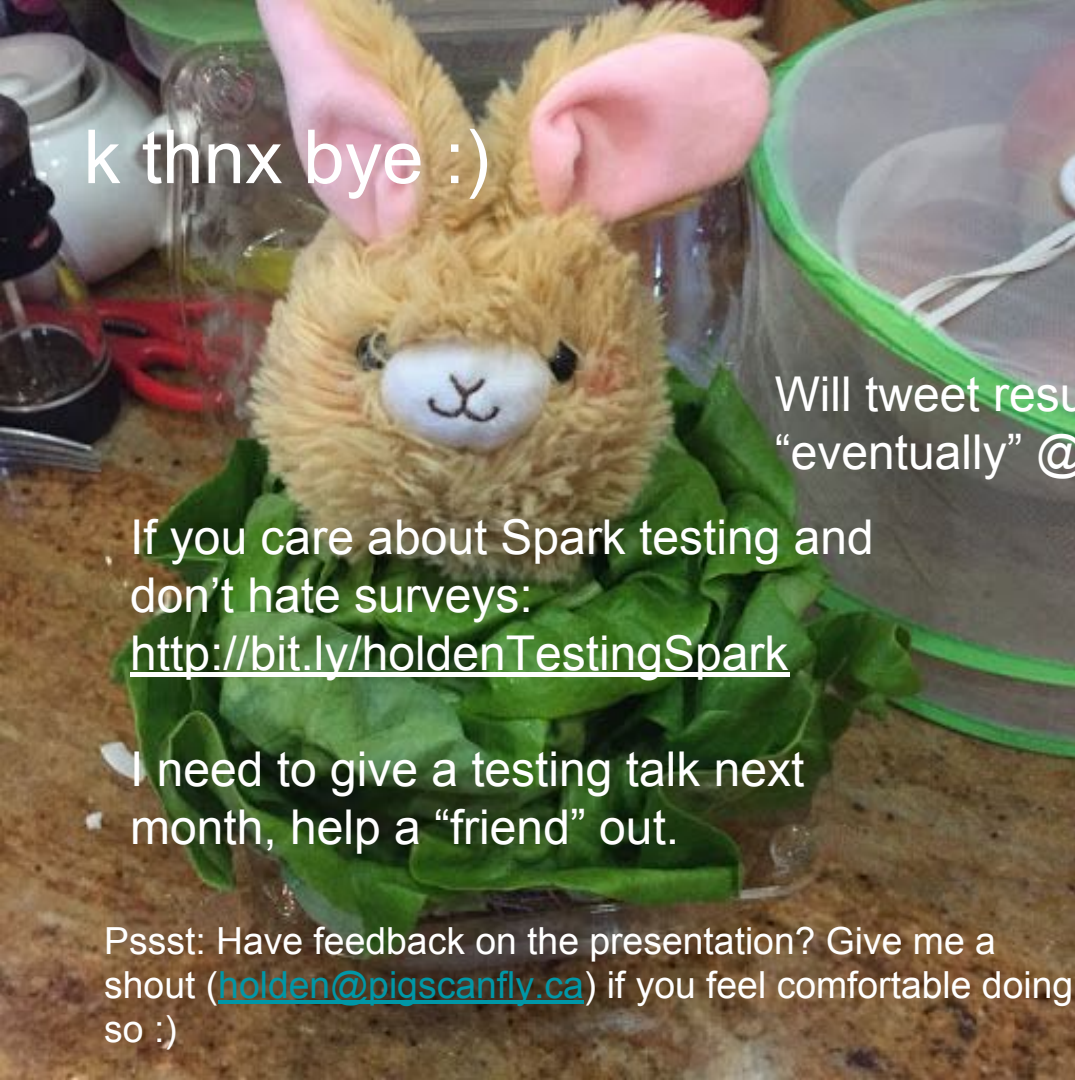
- Possible PyData Meetup in Amsterdam (tentative)
- Curry on Amsterdam
- OSCON Portland

- August

- JupyterCon NYC

- September

- Strata NYC
- Strangeloop STL



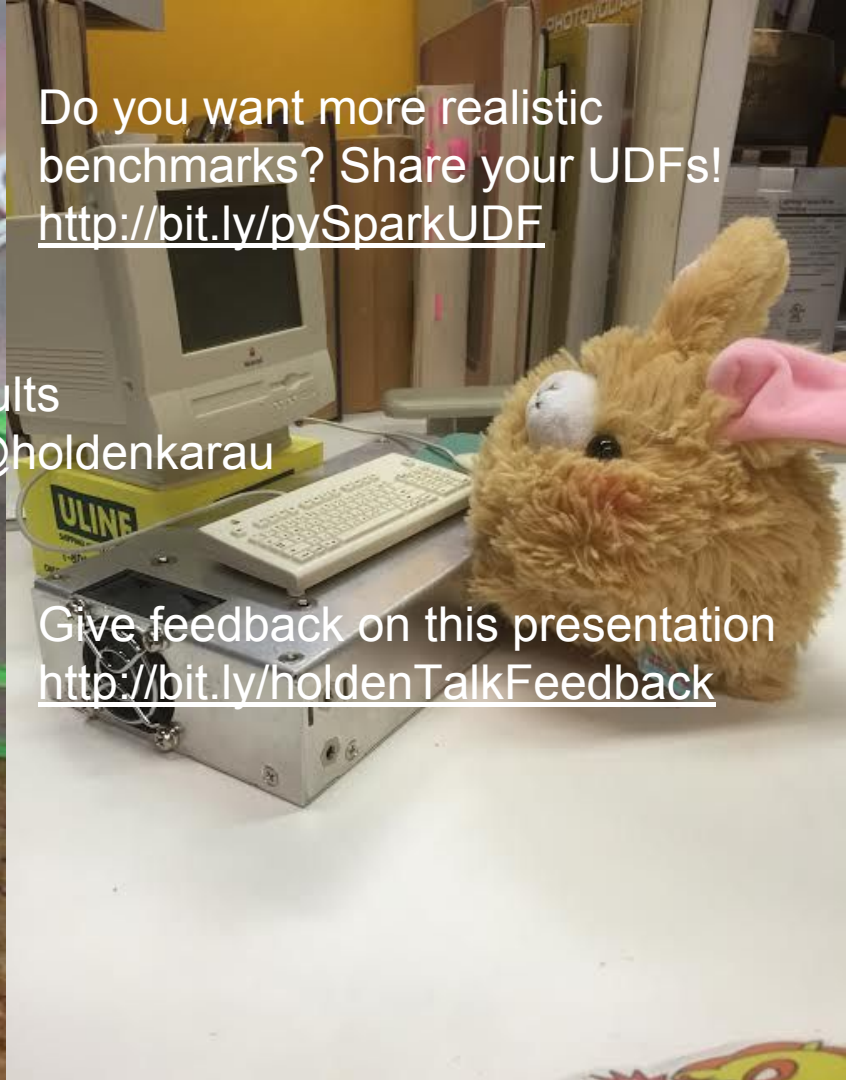
k thnx bye :)

If you care about Spark testing and don't hate surveys:
<http://bit.ly/holdenTestingSpark>

I need to give a testing talk next month, help a "friend" out.

Pssst: Have feedback on the presentation? Give me a shout (holden@pigscanfly.ca) if you feel comfortable doing so :)

Will tweet results "eventually" @holdenkarau



Do you want more realistic benchmarks? Share your UDFs!
<http://bit.ly/pySparkUDF>

Give feedback on this presentation
<http://bit.ly/holdenTalkFeedback>

(Optional) Beam Demo Time!!!!

- Word count!!!!!! So amazing!!!!!!!!!!
- Based on my testing last Friday there is a 2 in 3 chance this will hard lock my computer
- That's your friendly reminder not to run any of this in production
- Demo [shell script of fun](#) (go only) & [python + go](#)



Hadoop “streaming” (Python/R)

- Unix pipes!
- Involves a data copy, formats get sad
- But the overhead of a Map/Reduce task is pretty high anyways...



Lisa Larsson



Kafka: re-implement all the things

- Multiple options for connecting to Kafka from outside of the JVM (yay!)
- They implement the protocol to talk to Kafka (yay!)
- This involves duplicated client work, and sometimes the clients can be slow (solution, FFI bindings to C instead of Java)
- Buuuut -- we can't access all of the cool Kafka business (like Kafka Streams) and features depend on client libraries implementing them (easy to slip below parity)



Dask: a new beginning?

- Pure* python implementation
- Provides *real enough* DataFrame interface for distributed data
- Also your standard-ish distributed collections
- Multiple backends
- Primary challenge: interacting with the rest of the big data ecosystem
 - Arrow & friends might make this better with time too, buuut....
- See <https://dask.pydata.org/en/latest/> & <http://dask.pydata.org/en/latest/spark.html>