Accelerating Big Data Beyond the JVM

Engine noises go here

@holdenkarau

Holden:

- My name is Holden Karau
- Prefered pronouns are she/her
- Developer Advocate at Google
- Apache Spark PMC :)
- previously IBM, Alpine, Databricks, Google, Foursquare & Amazon
- co-author of Learning Spark & High Performance Spark
- @holdenkarau
- Slide share http://www.slideshare.net/hkarau
- Linkedin https://www.linkedin.com/in/holdenkarau
- Github https://github.com/holdenk
- Spark Videos http://bit.ly/holdenSparkVideos

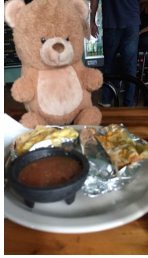
Who I think you wonderful humans are?

- Nice enough people
- Don’t mind pictures of cats
- Might know some the different distributed systems talked about
- Possibly know some Python or R
- Or are tired of scala/ java

What are these systems for?

- “Big Data”
  - Which varies from: it doesn’t fit on your macbook to the largest AWS/GCP server you can rent)
- Involve both the distribution of Data & Processing
- Most are written in the JVM (but not all)

What will be covered?



- A more detailed look at the current state of PySpark
- Why it isn’t good enough
- Why things are finally changing
- A brief tour of options for non-JVM languages in the Big Data space
- My even less subtle attempts to get you to buy my new book
- Pictures of cats & stuffed animals
- tl;dr - We’ve* made some bad** choices historically, and projects like Arrow & friends can save us from some of these (yay!)

What’s the state of non-JVM big data?

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

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

Yes, we have wordcount! :p


```
lines = sc.textFile(src)
words = lines.flatMap(lambda x: x.split(" "))
word_count =
  (words.map(lambda x: (x, 1))
        .reduceByKey(lambda x, y: x+y))
word_count.saveAsTextFile(output)
```

These are still combined and executed in one python executor

No data is read or processed until after this line

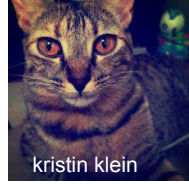
This is an “action” which forces spark to evaluate the RDD

A quick detour into PySpark's internals

+

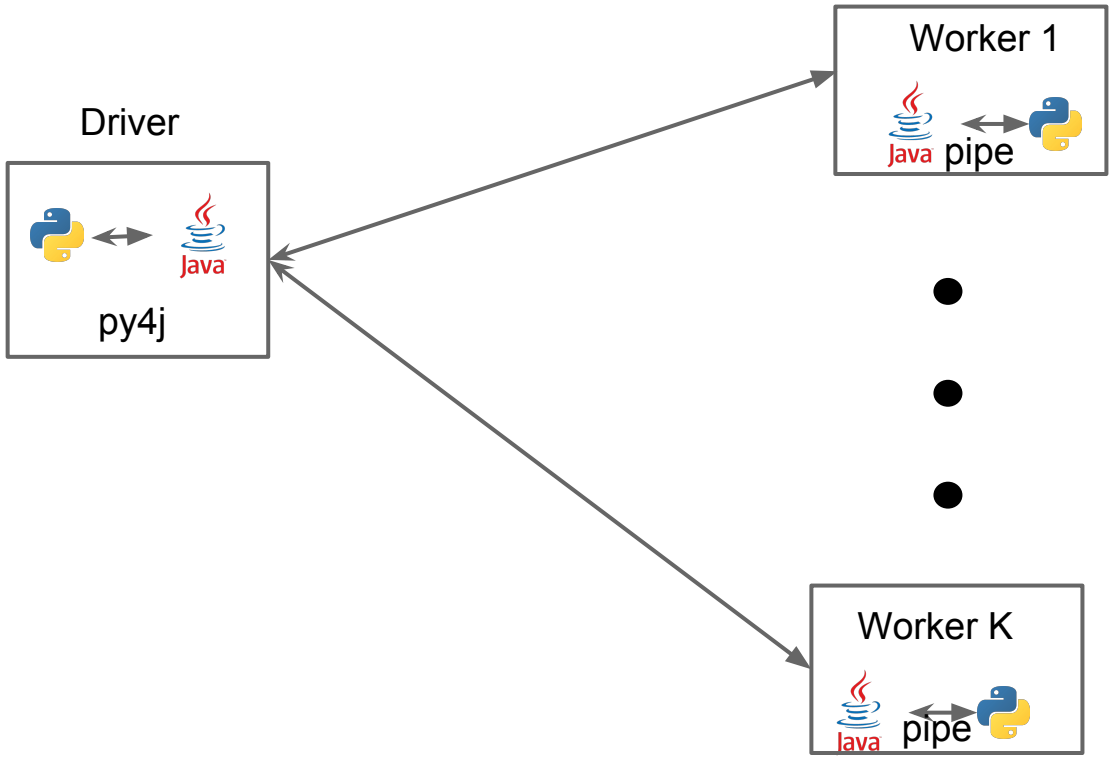
+ JSON

Spark in Scala, how does PySpark work?

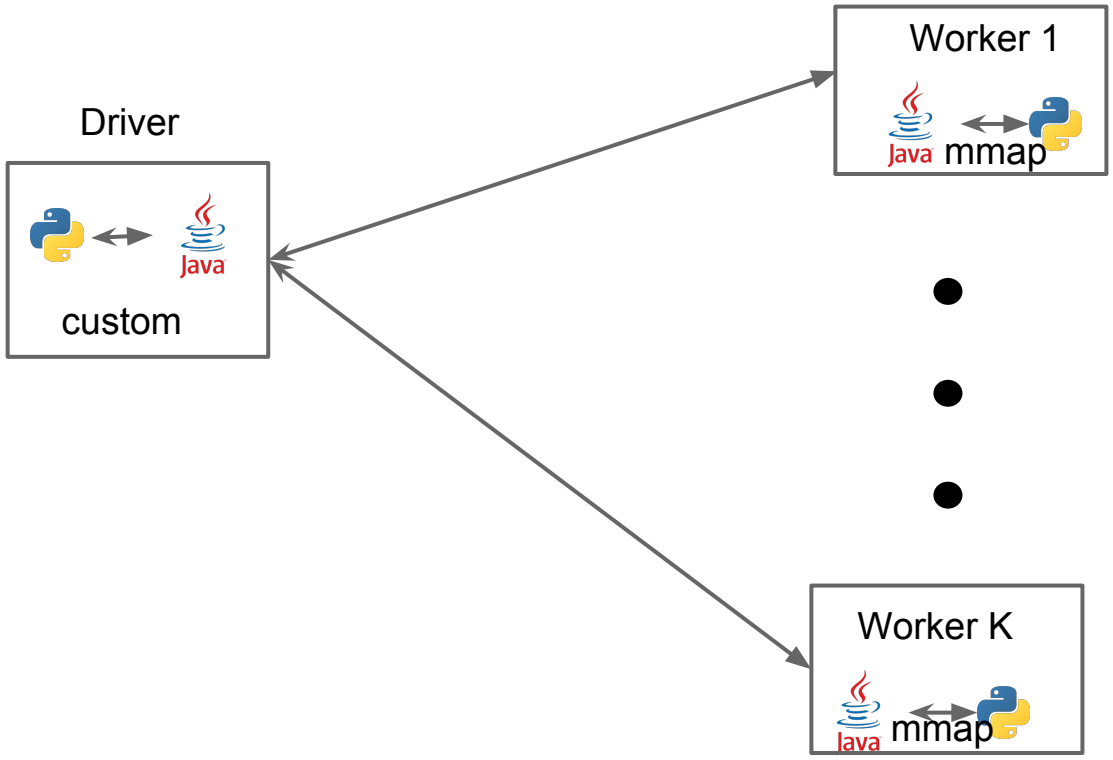


- Py4J + pickling + JSON and magic
  - Py4j in the driver
  - Pipes to start python process from java exec
  - cloudPickle to serialize data between JVM and python executors (transmitted via sockets)
  - Json for dataframe schema
- Data from Spark worker serialized and piped to Python worker --> then piped back to jvm
  - Multiple iterator-to-iterator transformations are still pipelined :)
  - So serialization happens only once per stage
- Spark SQL (and DataFrames) avoid some of this

So what does that look like?



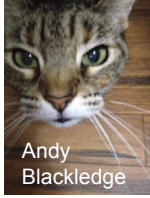
And in flink….



So how does that impact PySpark?

- 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
- Spark Features aren’t automatically exposed, but exposing them is normally simple

Our saviour from serialization: DataFrames



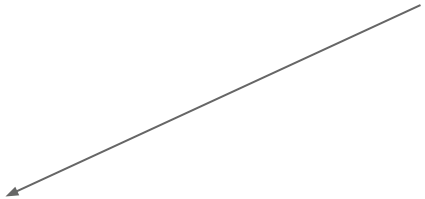
- For the most part keeps data in the JVM
  - Notable exception is UDFs written in Python
- Takes our python calls and turns it into a query plan if we need more than the native operations in Spark's DataFrames
- be wary of Distributed Systems bringing claims of usability….

So what are Spark DataFrames?

- More than SQL tables
- Not Pandas or R DataFrames
- Semi-structured (have schema information)
- tabular
- work on expression as well as lambdas
  - e.g. df.filter(df.col("happy") == true) instead of rdd.filter(lambda x: x.happy == true))
- Not a subset of Spark "Datasets" - since Dataset API isn't exposed in Python yet :(

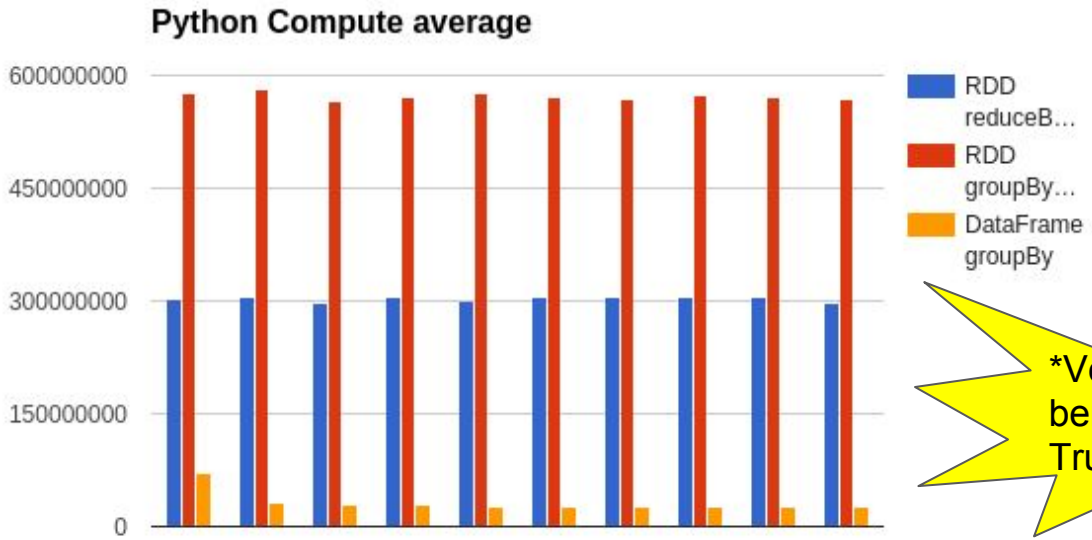
Word count w/Dataframes

Still have the double serialization here :(



```
df = sqlCtx.read.load(src)
# Returns an RDD
words = df.select("text").flatMap(lambda x: x.text.split(" "))
words_df = words.map(
  lambda x: Row(word=x, cnt=1)).toDF()
word_count = words_df.groupBy("word").sum()
word_count.write.format("parquet").save("wc.parquet")
```

We can see the difference easily:



*Vendor benchmark. Trust but verify.

*

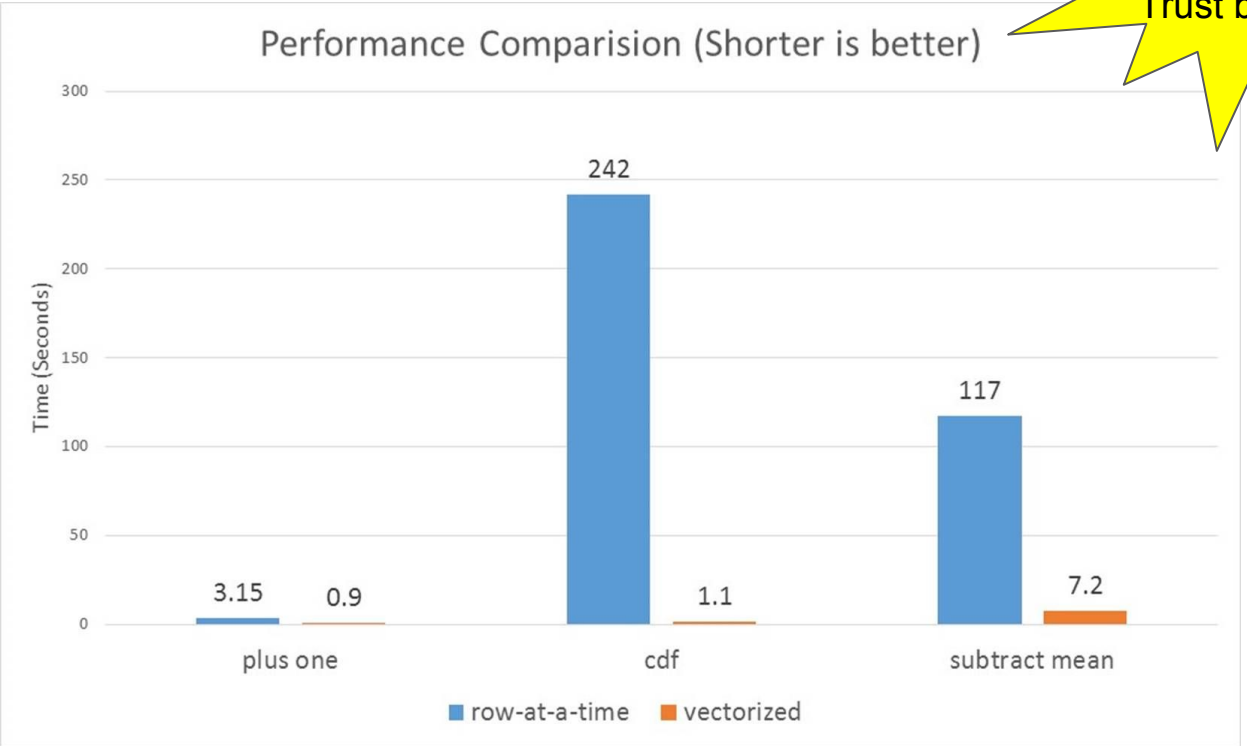
The “future”*: faster interchange

- By future I mean availability starting in the next 3-6 months (with more improvements after).
  - Yes much of this code exists, it just isn’t released yet so I’m sure we’ll find all sorts of bugs and ways to improve.
  - Relatedly you can help us test in Spark 2.3 when we start the RC process to catch bug early!
- Unifying our cross-language experience
  - And not just “normal” languages, CUDA counts yo


*Arrow: likely the future. I really hope so. Spark 2.3 and beyond!

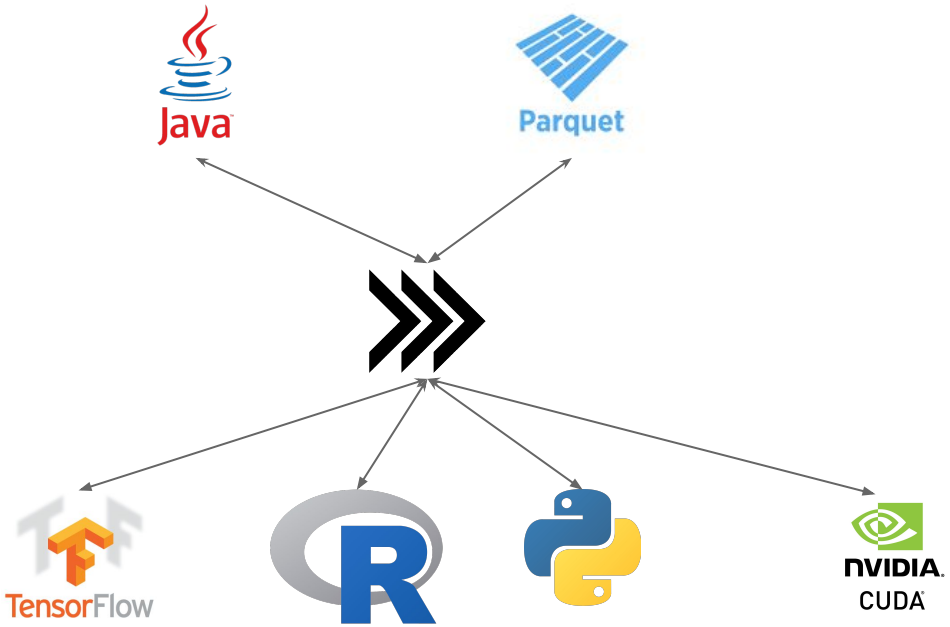
What does the future look like?*

*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

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())
```

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...

What do the rest of the systems do?

- Spoiler: mostly it’s not better
- Different tradeoffs, maybe better for your use case but all tradeoffs

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?

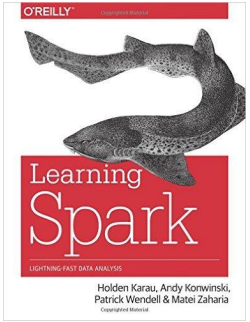
- Pure* python implementation
- Provides *real enough* DataFrame interface for distributed data
- Also your standard-ish distributed collections
- Multiple backends
- Primary challenge: interacting with the rest of the big data ecosystem
  - Arrow & friends might make this better with time too, buuut….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

BEAM Beyond the JVM

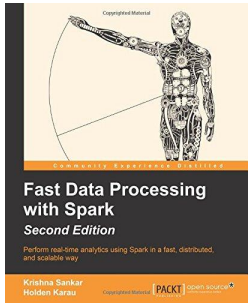
- Non JVM BEAM doesn’t work outside of Google’s environment yet
- tl;dr : uses grpc / protobuf
  - Similar to the common design but with more efficient representations (often)
- But exciting new plans to unify the runners and ease the support of different languages (called SDKS)
  - See https://beam.apache.org/contribute/portability/
- 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

References

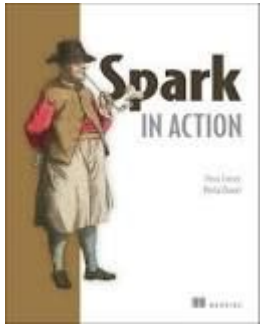
- Apache Arrow: https://arrow.apache.org/
- Brian (IBM) on initial Spark + Arrow https://arrow.apache.org/blog/2017/07/26/spark-arrow/
- Li Jin (two sigma) https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html
- Bill Maimone https://blogs.nvidia.com/blog/2017/06/27/gpu-computation-visualization/



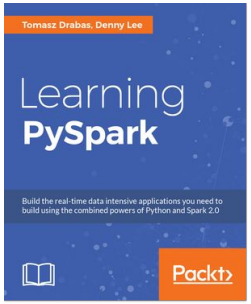
Learning Spark



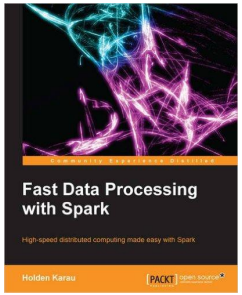
Fast Data Processing with Spark (2nd edition)



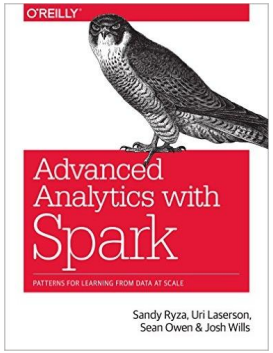
Spark in Action



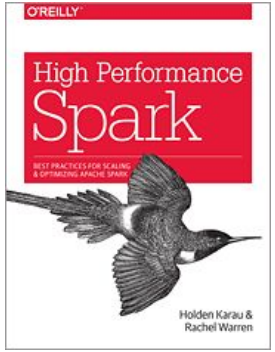
Learning PySpark



Fast Data Processing with Spark (Out of Date)

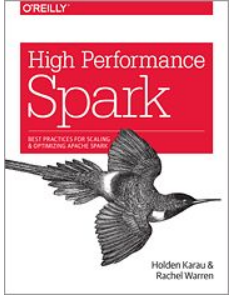


Advanced Analytics with Spark



High Performance Spark

High Performance Spark!



You can buy it today, the O'Reilly folks have it upstairs (& so does Amazon).

Only one chapter on non-JVM and nothing on Arrow, I'm sorry.

Cats love it*

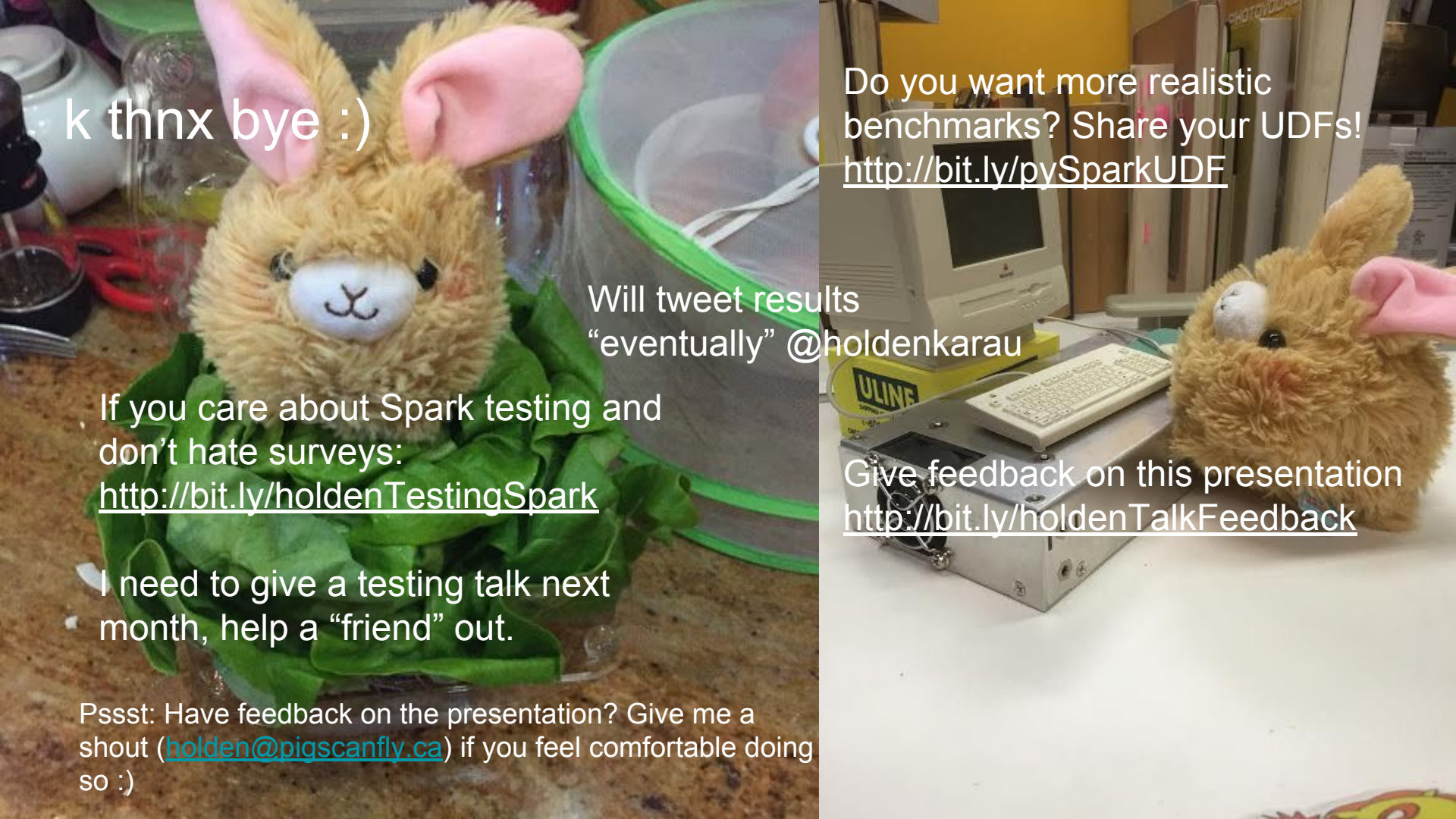
*Or at least the box it comes in. If buying for a cat, get print rather than e-book.

Spark Videos

- Apache Spark Youtube Channel
- My Spark videos on YouTube -
  - http://bit.ly/holdenSparkVideos
- Spark Summit 2014 training
- Paco's Introduction to Apache Spark

And some upcoming talks:

- Maybe office hours @ 6pm today if interest?*
  - Tweet at me (@holdenkarau) if interested+free, otherwise I'll go find chocolate :)
- JFokus - Surprisingly mostly talking about Python….
- Strata San Jose
- Strata London
- QCon Brasil
- QCon AI SF
- Know of interesting conferences/webinar things that should be on my radar? Let me know!


k thnx bye :)
Will tweet results “eventually” @holdenkarau
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 :)
Do you want more realistic benchmarks? Share your UDFs! http://bit.ly/pySparkUDF
Give feedback on this presentation http://bit.ly/holdenTalkFeedback
ULINE

Beyond wordcount: depencies?

- Your machines probably already have pandas
  - But maybe an old version
- But they might not have “special_business_logic”
  - Very special business logic, no one wants change fortran code*.
- Option 1: Talk to your vendor**
- Option 2: Try some open source software
- Option 3: Containers containers containers***
- Option 4: Any of this mornings talks
- We’re going to focus on option 2!

*Because it’s perfect, it is fortran after all.

** I don’t like this option because the vendor I work for doesn’t have an answer.

*** Great for your resume!

coffee_boat to the rescue*


```
# You can tell it's alpha cause were installing from github
!pip install --upgrade
git+https://github.com/nteract/coffee_boat.git
# Use the coffee boat
from coffee_boat import Captain
captain = Captain(accept_conda_license=True)
captain.add_pip_packages("pyarrow", "edtf")
captain.launch_ship()
sc = SparkContext(master="yarn")
# You can now use pyarrow & edtf
captain.add_pip_packages("yourmagic")
# You can now use yourmagic in transformations!
```

Bonus Slides

Maybe you ask a question and we go here :)

Why now?

- There’s been better formats/options for a long time
- JVM devs want to use libraries in other languages with lots of data
  - e.g. startup + Deep Learning + ? => profit
- Arrow has solved the chicken-egg problem by building not just the chicken & the egg, but also a hen house

What’s still going to hurt?

- Per-record streaming
  - Arrow is probably less awesome for serialization
  - But it’s still better than we had before
- Debugging is just going to get worse
- Custom data formats
  - Time to bust out the C++ code and a bottle of scotch / matte as appropriate
  - Or just accept the “legacy” performance

We can do that w/Kafka streams..

- Why bother learning from our mistakes?
- Or more seriously, the mistakes weren’t that bad...

Our “special” business logic

```
def transform(input):
    """
    Transforms the supplied input.
    """
    return str(len(input))
```

Let’s pretend all the world is a string:

```
override def transform(value: String): String = {
    // WARNING: This may summon cuthuluhu
    dataOut.writeInt(value.getBytes.size)
    dataOut.write(value.getBytes)
    dataOut.flush()
    val resultSize = dataIn.readInt()
    val result = new Array[Byte](resultSize)
    dataIn.readFully(result)
    // Assume UTF8, what could go wrong? :p
    new String(result)
  }
```

From https://github.com/holdenk/kafka-streams-python-cthulhu

Then make an instance to use it...

```
val testFuncFile =
  "kafka_streams_python_cthulhu/strlen.py"
stream.transformValues(
    PythonStringValueTransformerSupplier(testFuncFile))
// Or we could wrap this in the bridge but thats effort.
```

From https://github.com/holdenk/kafka-streams-python-cthulhu

Let’s pretend all the world is a string:

```
def main(socket):
    while (True):
        input_length = _read_int(socket)
        data = socket.read(input_length)
        result = transform(data)
        resultBytes = result.encode()
        _write_int(len(resultBytes), socket)
        socket.write(resultBytes)
        socket.flush()
```

From https://github.com/holdenk/kafka-streams-python-cthulhu

What does that let us do?

- You can add a map stage with your data scientists Python code in the middle
- You're limited to strings*
- Still missing the "driver side" integration (e.g. the interface requires someone to make a Scala class at some point)

What about things other than strings?

Use another system

- Like Spark! (oh wait) or BEAM* or FLINK*?

Write it in a format Python can understand:

- Pickling (from Java)
- JSON
- XML

Purely Python solutions

- Currently roll-your-own (but not that bad)

*These are also JVM based solutions calling into Python. I'm not saying they will also summon Cuthulhu, I'm just saying hang onto your souls.