

Terabytes at your Fingertips

Codifying the practice of distributed coding at scale

Sam Penrose

Sam Penrose, Data Engineer at Mozilla

- Your ~~teacher~~ fellow student
- Does “data engineering” for Mozilla (distributed, terabyte-scale)
- Sam / Mozilla: github.com/SamPenrose
- Sam / life: www.sampenrose.net | twitter.com/sampenrose

We will learn

1. How distributed coding at scale (“DCS”) is **new**
2. The **three problems** you encounter and their **five root causes**
3. A **core technique** for interactive DCS coding

“Normal” coding

```
>>> def extract_and_transform(d):
...     value = d['field']
...     return float(value)
...
>>> def test():
...     data = [{'field': '1.1'}, {'field': '1.02'}]
...     expected = [1.1, 1.02]
...     actual = map(extract_and_transform, data)
...     assert actual == expected
...
>>> test()
>>>
```

Tests pass—ship it!

```
>>> import json
>>> with open('production_data.json') as f:
...     production_data = json.load(f)
...
>>> loaded = []
>>> for d in production_data:
...     value = extract_and_transform(d)
...     loaded.append(value)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    File "<stdin>", line 3, in extract_and_transform
      TypeError: float() argument must be a string or a number
```

Let's debug interactively

```
>>> def debug_etl_with_print(d):
...     value = d['field']
...     print "value is:", value
...     return float(value)
...
>>> for d in production_data:
...     _ = debug_etl_with_print(d)
...
value is: 1.07
value is: 1.13
value is: None
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    File "<stdin>", line 4, in debug_etl_with_print
      TypeError: float() argument must be a string or a number
```

What did we just do?

1. Noticed a problem.
2. **Displayed the program's state.**
3. Identified the problem.

How old is this technique?

1989: Python

1970: Unix shell

1962: Lisp REPL

1951: add print statements



Ritchie and Thompson(sitting) at PDP-11 in Bell Labs

No, really: 1951

5-2 Location of mistakes in a program.

It might be thought that a good way of finding errors in a program would be to make the machine proceed order by order under the control of the "Single E.P." button (see Section 6-4), and to study the numbers in the machine by watching the monitors attached to the arithmetical unit and store. This, however, usually turns out to be a very slow and inefficient process, especially as the numbers are displayed in binary form. Methods have therefore been developed which permit the machine to proceed unhindered by the operator, whilst printing on the teleprinter a permanent record that can be studied at leisure, and that will assist in understanding the nature of the mistake.

One such method is to wait until the machine has stopped (or to stop it deliberately) and then, without clearing the whole store, to insert (by pressing the starter button again) a small program which will print, in suitable form, the contents of part of the store. This has come to be known as the "post-mortem" method. Tapes are kept available near the EDSAC for printing the function letters, or address parts, of orders in consecutive storage locations. Programmers may also prepare their own post-mortem tapes.

This method yields only a static picture of the store as it was when the calculation stopped. Other methods have been derived to provide information about the whole course of the calculation. These necessarily involve modifying the program to cause the extra printing, and therefore a new tape must be prepared and presented to the machine. This, however, is no hardship, since the machine will read an average tape in about a minute, and the preparation need not take more than a few minutes of the programmer's time.

5-21 Method using extra output orders. One simple and very useful plan is to place an output order at the beginning of the master routine and in front of each subroutine so that the completion of the various stages of the program

Notebooks: brave new (?) world

```
>>> import json
>>> with open('production_data.json') as f:
...     production_data = json.load(f)
...
>>> loaded = []
>>> for d in production_data:
...     value = extract_and_transform(d)
...     loaded.append(value)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    File "<stdin>", line 3, in extract_and_transform
TypeError: float() argument must be a string or a number
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Jupyter Not just Python, stateful Python in a Web Page! Last Checkpoint: 05/26/2017 (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, various icons for file operations.
- In [1]:**

```
def extract_and_transform(d):
    value = d['field']
    return float(value)
```
- In [2]:**

```
import json
with open('/Users/spenrose/production_data.json') as f:
    production_data = json.load(f)
loaded = []
for d in production_data:
    value = extract_and_transform(d)
    loaded.append(value)
```
- Stack Trace:** A dashed red line separates the code from the error output.

```
TypeError                                 Traceback (most recent call last)
<ipython-input-2-4366f9deab51> in <module>()
      4     loaded = []
      5     for d in production_data:
----> 6         value = extract_and_transform(d)
      7         loaded.append(value)

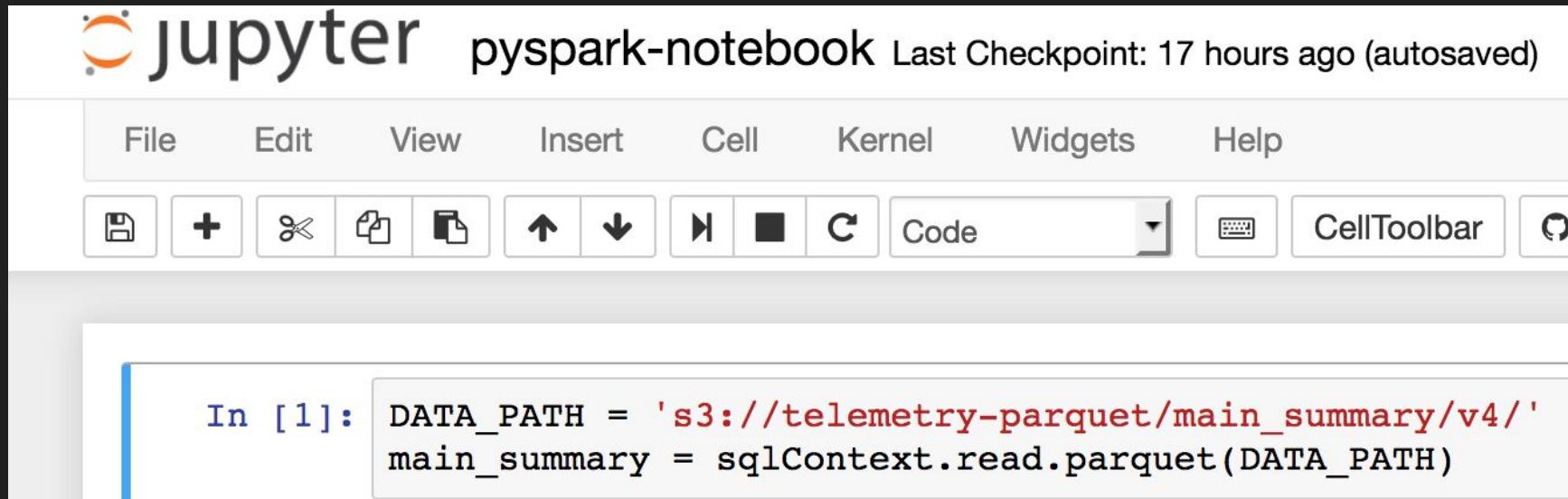
<ipython-input-1-71e89f87416b> in extract_and_transform(d)
      1     def extract_and_transform(d):
      2         value = d['field']
----> 3         return float(value)

TypeError: float() argument must be a string or a number
```

Coding: 1951 -> 201x

1. Run
2. Observe
 - a. By print() if necessary
3. Edit
 - a. GOTO: 1

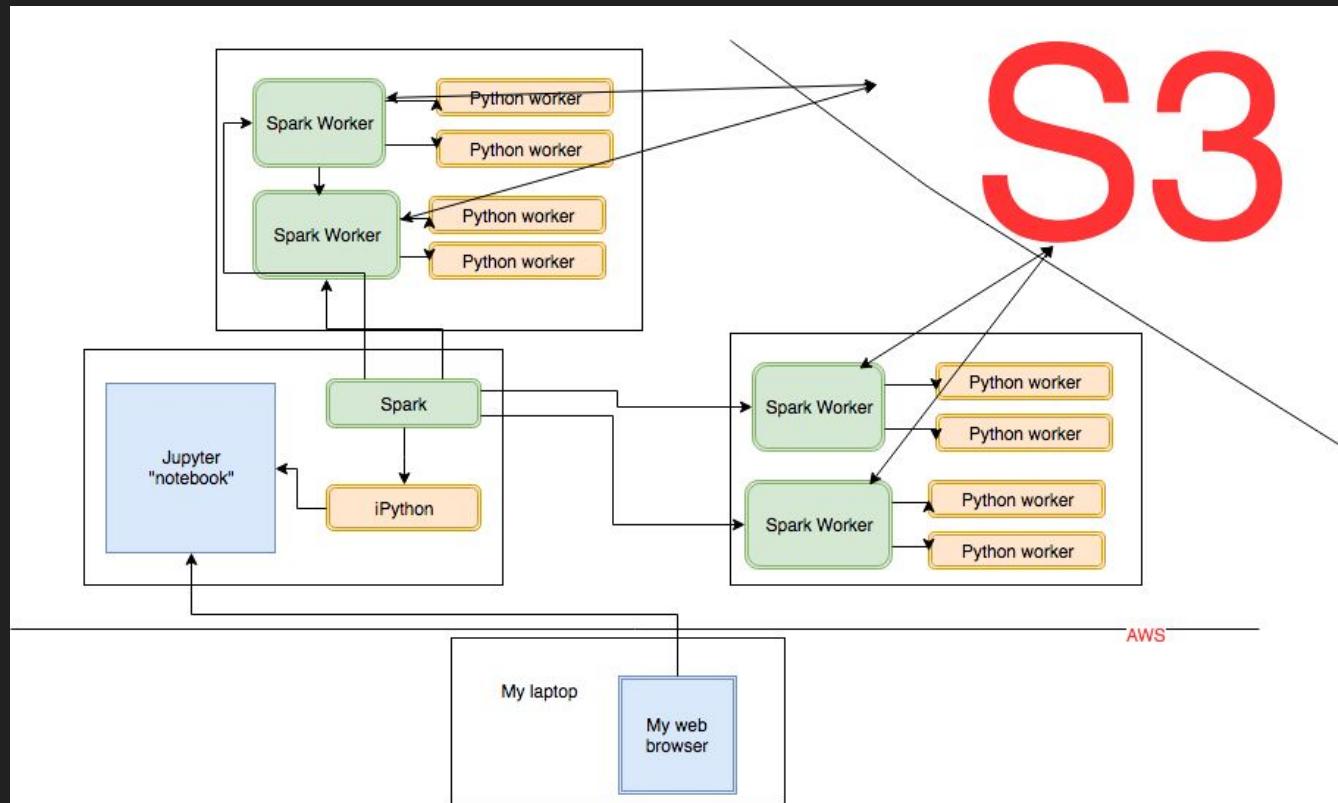
2017: 2 lines -> 500TB on 30 nodes



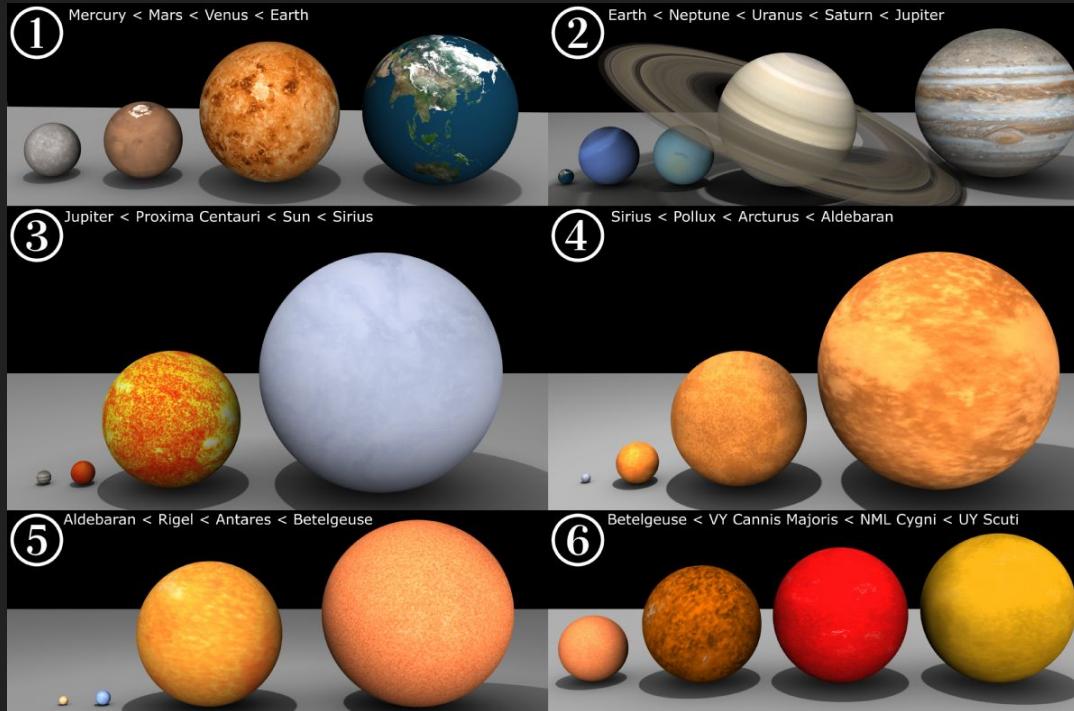
The screenshot shows a Jupyter Notebook interface with the title "jupyter pyspark-notebook Last Checkpoint: 17 hours ago (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with various icons for file operations like save, new, cut, copy, paste, and navigation. A dropdown menu is open next to the "Code" button. The main area displays a code cell labeled "In [1]:" containing the following Python code:

```
In [1]: DATA_PATH = 's3://telemetry-parquet/main_summary/v4/'  
main_summary = sqlContext.read.parquet(DATA_PATH)
```

Everything has changed ... except the UI



Our minds don't do scale



"Relative sizes of the planets in the solar system and some of the largest stars" by Jcpag2012

Not Python (or Scala), not a variable, not printable

jupyter pyspark-notebook Last Checkpoint: 17 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Python [conda root]

In [1]: `DATA_PATH = 's3://telemetry-parquet/main_summary/v4/'
main_summary = sqlContext.read.parquet(DATA_PATH)`

In [3]: `main_summary`

Out[3]: `DataFrame[document_id: string, client_id: string, channel: string, normalized_channel: string,
, country: string, city: string, os: string, os_version: string, os_service_pack_major: bigint,
os_service_pack_minor: bigint, windows_build_number: bigint, windows_uber: bigint, install_
year: bigint, profile_creation_date: bigint, subsession_start_date: string, subsession_length:
: bigint, distribution_id: string, submission_date: string, sync_configured: boolean, sync_co
unt_desktop: int, sync_count_mobile: int, app_build_id: string, app_display_version: string,
app_name: string, app_version: string, timestamp: bigint, env_build_id: string, env_build_ver
sion: string, env_build_arch: string, e10s_enabled: boolean, e10s_cohort: string, locale: str
ing, active_experiment_id: string, active_experiment_branch: string, reason: string, timezone
_offset: int, plugin_hangs: int, aborts_plugin: int, aborts_content: int, aborts_gmplugin: in
t, crashes_detected_plugin: int, crashes_detected_content: int, crashes_detected_gmplugin: in
t, crash_submit_attempt_main: int, crash_submit_attempt_content: int, crash_submit_attempt_pl
ugin: int, crash_submit_success_main: int, crash_submit_success_content: int, crash_submit_su`

Enormously powerful pseudo-statements

```
In [5]: from datetime import date, timedelta  
week_ago = date.today() - timedelta(days=7)  
start = week_ago.isoformat()  
clause = "submission_date > '{}'.format(start)  
recent = main_summary.where(clause)
```

```
In [6]: recent.count()
```

```
Out[6]: 53866405276
```

Well invalidate my priors and call me perplexed!

```
In [5]: from datetime import date, timedelta  
week_ago = date.today() - timedelta(days=7)  
start = week_ago.isoformat()  
clause = "submission_date > '{}'".format(start)  
recent = main_summary.where(clause)
```

```
In [6]: recent.count()
```

```
Out[6]: 53866405276
```

```
In [7]: main_summary.count()
```

```
Out[7]: 164284288426
```

Wait, 1/3 of data in last week?
Something's wrong.

Using the DCS¹ prompt: take()

```
Out[18]: [Row(active_experiment_id=u'e10s-enabled-beta-20151214@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),  
          Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org')]
```

```
In [22]: e10s = recent.rdd.filter(lambda row: row['active_experiment_id'].startswith('e10s'))  
ten_e10s = e10s.take(10)
```

1. “DCS”: distributed computing at scale

Tracebacks at the DCS prompt (1)

Out[18]:

```
[Row(active_experiment_id=u'e10s-enabled-beta-20151214@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org'),
 click to expand output; double click to hide output
 Row(active_experiment_id=u'e10s-beta45-withoutaddons@experiments.mozilla.org')]
```

In [22]: e10s = recent.rdd.filter(lambda row: row['active_experiment_id'].startswith('e10s'))
ten_e10s = e10s.take(10)

```
Py4JJavaErrorTraceback (most recent call last)
<ipython-input-22-334b6583877b> in <module>()
      1 e10s = recent.rdd.filter(lambda row: row['active_experiment_id'].startswith('e10s'))
----> 2 ten_e10s = e10s.take(10)
```

Tracebacks at the DCS prompt (2)

Tracebacks at the DCS prompt (3)

```
File "<ipython-input-22-334b6583877b>", line 1, in <lambda>
AttributeError: 'NoneType' object has no attribute 'startswith'
```

This reminds me of something ...

```
File "<ipython-input-22-334b6583877b>", line 1, in <lambda>
AttributeError: 'NoneType' object has no attribute 'startswith'

>>> import json
>>> with open('production_data.json') as f:
...     production_data = json.load(f)
...
>>> loaded = []
>>> for d in production_data:
...     value = extract_and_transform(d)
...     loaded.append(value)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    File "<stdin>", line 3, in extract_and_transform
      TypeError: float() argument must be a string or a number
```

Root cause #1: data variance

```
In [16]: def is_e10s_experiment(row):
    if row['active_experiment_id'] is None:
        return False
    return row['active_experiment_id'].startswith('e10s')
e10s = recent.rdd.filter(is_e10s_experiment)
ten_e10s = e10s.take(10)
```

```
In [17]: ten_e10s[0]['active_experiment_id']
```

```
Out[17]: u'e10s-beta45-withoutaddons@experiments.mozilla.org'
```

Why was I able to debug that?

Did I display the program state? **No.**



Ritchie and Thompson(sitting) at PDP-11 in Bell Labs

What about the previous issue?

```
In [5]: from datetime import date, timedelta  
week_ago = date.today() - timedelta(days=7)  
start = week_ago.isoformat()  
clause = "submission_date > '{}'".format(start)  
recent = main_summary.where(clause)
```

```
In [6]: recent.count()
```

```
Out[6]: 53866405276
```

```
In [7]: main_summary.count()
```

```
Out[7]: 164284288426
```

Wait, 1/3 of data in last week?
Something's wrong.|

Maybe I can do something about that, too!

```
Wait, 1/3 of data in last week?  
Something's wrong.  
  
In [8]: tinytinylittleeensyweensybit = main_summary.take(10)  
  
In [9]: for row in tinytinylittleeensyweensybit:  
        print row['submission_date'],  
  
        20160620 20160620 20160620 20160620 20160620 20160620 20160620 201  
  
In [10]: start  
  
Out[10]: '2017-05-19'  
  
In [11]: properly_formatted = week_ago.isoformat().replace('-', '')  
properly_formatted  
  
Out[11]: '20170519'  
  
In [12]: clause = "submission_date > '{}'".format(properly_formatted)|  
recent = main_summary.where(clause)  
  
In [13]: recent.count()  
  
Out[13]: 2119866345
```

Why was I able to debug these two problems?

1. I displayed the DCS **program state**.
2. In the first case, the **traceback** displayed the **problem**.
3. In the second case, the **problem** was displayable in **every row**.

The core technique*

1. **Localize** the **problem** state to observe it.

*part one

Problem states, localized and observed

```
File "<ipython-input-22-334b6583877b>", line 1, in <lambda>
AttributeError: 'NoneType' object has no attribute 'startswith'
```

```
Wait, 1/3 of data in last week?  
Something's wrong.
```

```
In [8]: tinytinylittleeensyweensybit = main_summary.take(10)
```

```
In [9]: for row in tinytinylittleeensyweensybit:  
    print row['submission_date'],
```

```
20160620 20160620 20160620 20160620 20160620 20160620 20160620 20160620 2011
```

```
In [10]: start
```

```
Out[10]: '2017-05-19'
```

```
In [11]: properly_formatted = week_ago.isoformat().replace('-', '')  
properly_formatted
```

```
Out[11]: '20170519'
```

```
In [12]: clause = "submission_date > '{}'".format(properly_formatted)|  
recent = main_summary.where(clause)
```

```
In [13]: recent.count()
```

```
Out[13]: 2119866345
```

Modularize fix, because software engineering

```
In [16]: def is_e10s_experiment(row):
    if row['active_experiment_id'] is None:
        return False
    return row['active_experiment_id'].startswith('e10s')
e10s = recent.rdd.filter(is_e10s_experiment)
ten_e10s = e10s.take(10)
```

```
In [17]: ten_e10s[0]['active_experiment_id']
```

```
Out[17]: u'e10s-beta45-withoutaddons@experiments.mozilla.org'
```

The screenshot shows a Jupyter Notebook interface with a toolbar at the top and a code cell below it. The code cell contains Python code for filtering rows based on the 'active_experiment_id' field. The output of the code is displayed below the cell.

```
hadoop@i...-31-0-97:~ bash ... java
File Edit Options Buffers Tools Python Help
def is_e10s_experiment(row):
    if row['active_experiment_id'] is None:
        return False
    return row['active_experiment_id'].startswith('e10s')
```

Test, because software engineering

```
In [19]: # Modularize and test module locally
import my_local_utilities
worked = filter(my_local_utilities.is_e10s_experiment,
                tinytinylittleeensyweensybit) == []
worked

Out[19]: True
```

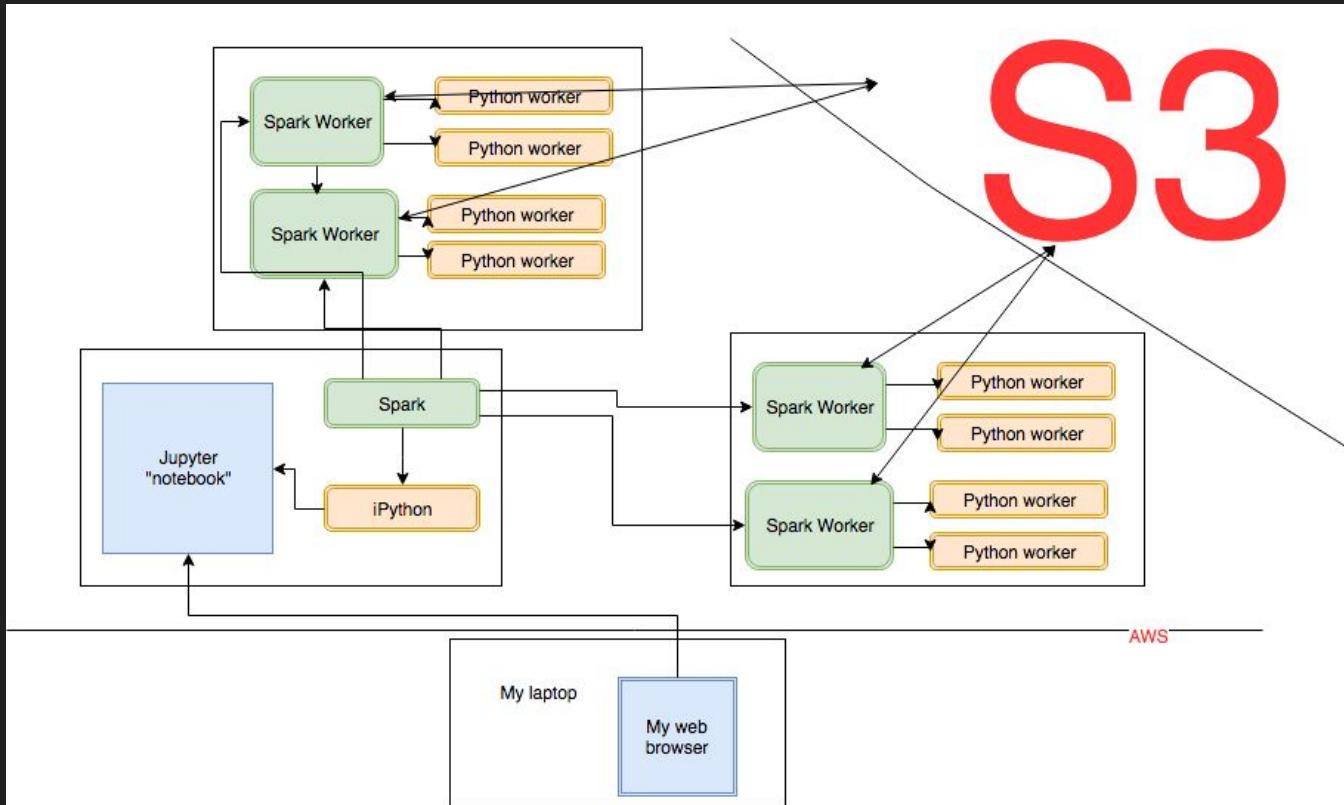
Engineering problem (1)

```
In [20]: # Now distribute *the test data* and try the code on it.  
tiny_distributed = sc.parallelize(tinytinylittleeensyweensybit)  
works_distributed = tiny_distributed.filter(  
    my_local_utilities.is_e10s_experiment).collect() == []  
  
Py4JJavaErrorTraceback (most recent call last)  
<ipython-input-20-db53628f212c> in <module>()  
      2 tiny_distributed = sc.parallelize(tinytinylittleeensyweensybit)  
click to scroll output; double click to hide  3 works_distributed = tiny_distributed.filter(  
----> 4     my_local_utilities.is_e10s_experiment).collect() == []
```

Engineering problem (2)

```
Py4JJavaError: An error occurred while calling z:  
.  
: org.apache.spark.SparkException: Job aborted due  
led 4 times, most recent failure: Lost task 0.3 i  
s-west-2.compute.internal): org.apache.spark.api.  
nt call last):  
    File "/mnt/yarn/usercache/hadoop/appcache/app  
12025_0001_01_000164/pyspark.zip/pyspark/worker.p  
        func, profiler, deserializer, serializer = re  
        File "/mnt/yarn/usercache/hadoop/appcache/app  
12025_0001_01_000164/pyspark.zip/pyspark/worker.p  
        command = serializer._read_with_length(file)  
        File "/mnt/yarn/usercache/hadoop/appcache/app  
12025_0001_01_000164/pyspark.zip/pyspark/serializ  
        return self.loads(obj)  
        File "/mnt/yarn/usercache/hadoop/appcache/app  
12025_0001_01_000164/pyspark.zip/pyspark/serializ  
        return pickle.loads(obj)  
ImportError: No module named my_local_utilities
```

I just imported it ... locally



Root cause #2: heterogenous framework state

```
In [19]: # Modularize and test module locally
import my_local_utilities
worked = filter(my_local_utilities.is_e10s_experiment,
                tinytinylittleeensyweensybit) == []
worked
```

```
Out[19]: True
```

```
In [20]: # Now distribute *the test data* and try the code on it.
tiny_distributed = sc.parallelize(tinytinylittleeensyweensybit)
works_distributed = tiny_distributed.filter(
    my_local_utilities.is_e10s_experiment).collect() == []
```

```
Py4JJavaErrorTraceback (most recent call last)
<ipython-input-20-db53628f212c> in <module>()
      2     tiny_distributed = sc.parallelize(tinytinylittleeensyweensybit)
      3     works_distributed = tiny_distributed.filter(
----> 4         my_local_utilities.is_e10s_experiment).collect() == []
```

The core technique, completed

1. **Localize problem** state to observe it.
2. Scale out and up in **steps**.

Scaling out and up in steps

1. Get the code working with a small local dataset.
2. Scale **out**: distribute **that small local dataset** and **that code**. Still works?
3. Scale **up**: iteratively run that code on more and more of the full dataset.
 - a. E.g. 1%, 10%, 20%, 40%
 - b. On each loop, look for a **scale fail**.

Mo' data, mo' problems

Three flavors of scale fail:

1. Traceback reporting a **your-code-vs-your-data** problem.
2. Exponential **slowdown**.
3. Traceback reporting a **toolchain panic**.

Your code vs. your data: “only bring me problems”

```
In [14]: def fragile_is_e10s_experiment(row):
    """I will raise an AttributeError."""
    return row['active_experiment_id'].startswith('e10s')
def is_this_row_naughty(row):
    try:
        row['active_experiment_id'].startswith('e10s')
    except AttributeError:
        return True # or row
    return False
problem_rows = recent.rdd.filter(is_this_row_naughty)
problem_made_locally_observable = problem_rows.take(10)
```

Root cause #3: mental models of control flow

```
def to_sessions((client_id, pinglist)):
    client = {
        'client_id': client_id,
        'known_bad_pings': [],
        'sessions': [],
        'breaks_by_psc': 0,
        'breaks_by_ssid': 0,
        'breaks_by_sid': 0
    }
    def mark_is_finished(session):
        try:
            session['to_end'] = session['pings'][-1]['meta']['reason'] in ("shutdown", "aborted-session")
            session['complete'] = session['from_start'] and session['to_end']
        except Exception:
            session['to_end'] = False
            session['complete'] = False
    def make_session(first, previousSessionId=None):
        session = {'pings': [first]}
        try:
            session['from_start'] = first.get('payload', {}).get('info', {}).get(
                'subsessionCounter') == 1
        except Exception:
            session['from_start'] = False
        mark_is_finished(session)
        if previousSessionId:
            if first['payload']['info']['previousSessionId'] != previousSessionId:
                client['breaks_by_sid'] += 1
        return session
    def psc(d):
        result = -99
        try:
            result = d.get('payload', {}).get('info', {}).get(
                'profileSubsessionCounter', result)
        except Exception:
            pass
        return result
    pinglist.sort(key=psc)
    previous = pinglist[0]
    session = make_session(previous)
    if not pinglist[1:]:
        mark_is_finished(session)
        client['sessions'].append(session)
    for d in pinglist[1:]:
        try:
            if previous['payload']['info']['profileSubsessionCounter'] + 1 != \
                    d['payload']['info']['profileSubsessionCounter']:
                client['breaks_by_psc'] += 1
            if previous['payload']['info']['subsessionId'] != \
                    d['payload']['info']['previousSubsessionId']:
                client['breaks_by_ssid'] += 1
            if previous['payload']['info']['sessionId'] == \
                    d['payload']['info']['sessionId']:
                session['pings'].append(d)
            else:
                mark_is_finished(session)
                client['sessions'].append(session)
                session = make_session(d, previous['payload']['info']['sessionId'])
            previous = d
        except Exception:
            client['known_bad_pings'].append(d)

    return client
```

If the results surprise you, quantify the control flow

You have a mental model of your code's control flow.

Localize and observe it to see if it is the problem.

```
In [21]: def quantify_control_flow():
    almost_every_time = sc.accumulator(0)
    once_in_a_blue_moon = sc.accumulator(0)
    def process(row):
        if row['active_experiment_id'] is not None:
            almost_every_time.add(1)
            pass # Do the real work.
        else:
            once_in_a_blue_moon.add(1)
            pass
    return process, almost_every_time, once_in_a_blue_moon
quantifier, almost_every_time, once_in_a_blue_moon = quantify_control_flow()

In [22]: _ = recent.rdd.map(quantifier).collect()

In [23]: print almost_every_time # -> "should" be huge, is tiny
print once_in_a_blue_moon # "should" be tiny, is huge
12781730
2107084615
```

Slowness

Two new root causes:

- #4: exponential behavior
- #5: heavy load (high ratio of data to computational resources)

Reallocate

1. Are you comparing every row to every row?
2. Yes: you are (probably) running an exponential algorithm.
 - a. First step: repartition on the comparison key(s).
 - b. (Make sure the algorithm lacks “inner loop: `fetch_across_network()`”)
3. No: see if the data is skewed

Repartition on your comparison key

```
def partitioner(d):
    return hash(d.get('meta', d).get('clientId', 'MISSING'))
def get_key(d):
    return d.get('meta', d).get('clientId', 'MISSING')
PARTITION_FACTOR = 10
def group_by_client_id(rdd):
    size = rdd.getNumPartitions() * PARTITION_FACTOR
    rdd.repartitionAndSortWithinPartitions(size, partitioner, True, get_key)
    return rdd.groupBy(lambda d: d.get('clientId', 'MISSING'))
```

Divide and conquer

1. Is your processing function associative?
2. If so, turn that frown upside down!
 - a. $\text{data}^*2 \rightarrow \text{delay}^*4$:-(
 - b. $\text{data}/2 \rightarrow \text{delay}/4$:-)



Sample

1. 10^6 rows: 800,000 ‘a’, 200,000 ‘b’ or ‘c’
2. Take 10% sample -> 10^5 rows
 - a. “What is the ratio of ‘a’ to (‘b’ or ‘c’)?” -> “About 4:1.”
 - b. Sampling worked great!
3. 10^1 rows: [a, a, a, c, a, b, a, a, a]
- a. Take 10% sample: useless.
4. More information:
 - a. ~~Read an article.~~
 - b. ~~Read a book.~~
 - c. Take a class.

Spend



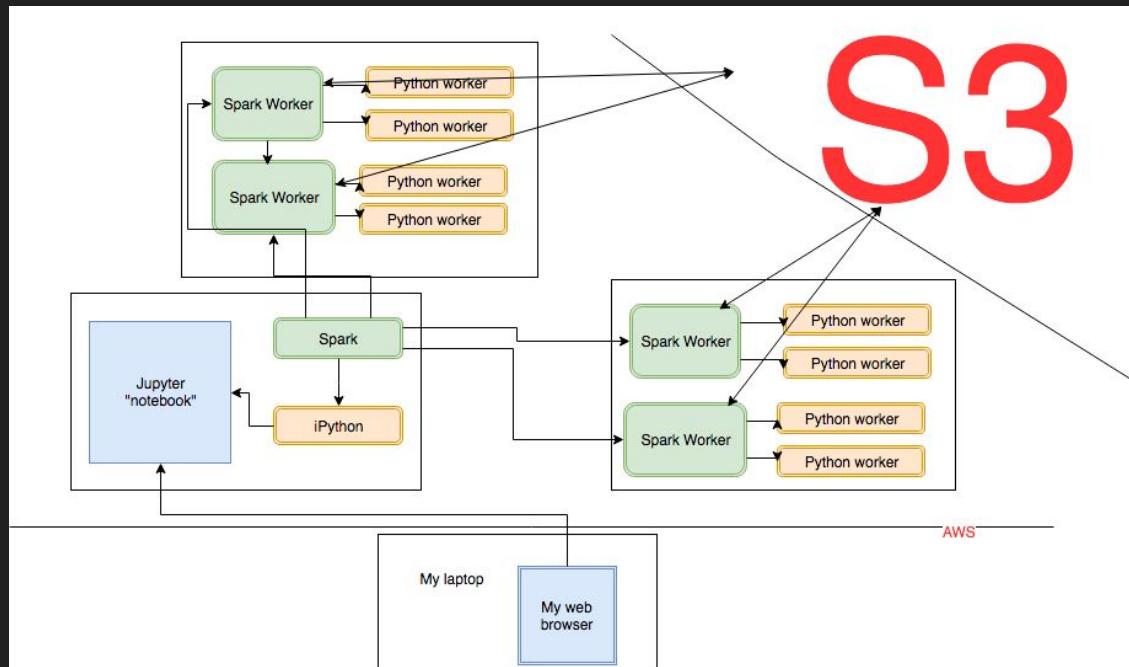
By Argonne National Laboratory's Flickr page [CC BY-SA 2.0 (<http://creativecommons.org/licenses/by-sa/2.0>)], via Wikimedia Commons

Toolchain panics: recognition

- **Py4JNetworkError**: An error occurred while trying to connect to the Java server
- **org.apache.spark.SparkException**: Job 330 cancelled; SparkContext was shut down
- **org.apache.spark.sql.catalyst.errors.package\$TreeNodeException**
- **org.apache.spark.SparkException**: Kryo serialization failed: Buffer overflow.

Toolchain panics: recovery

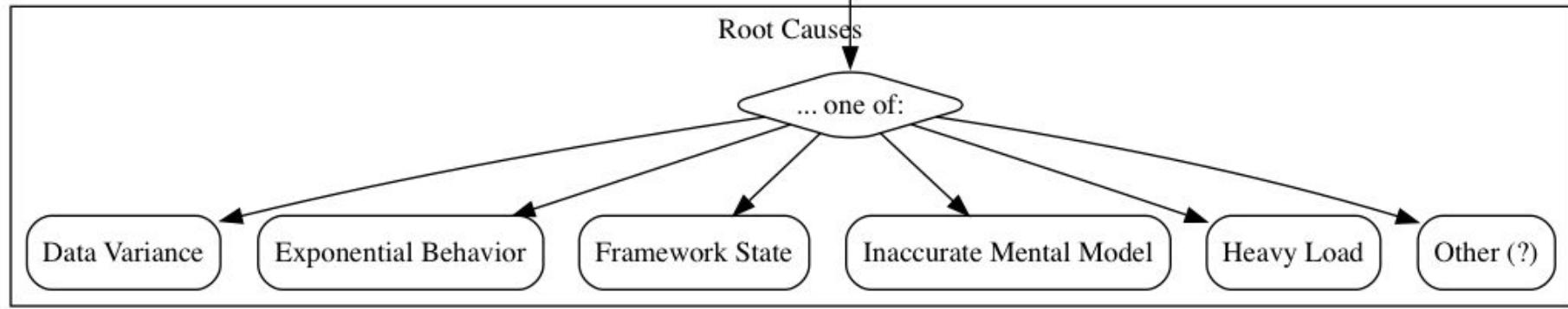
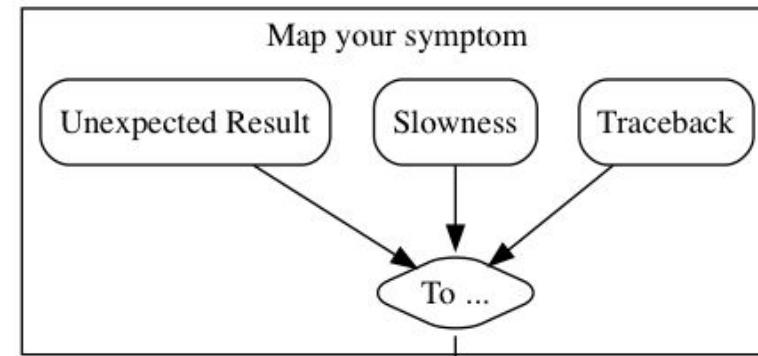
- **Lighten** the load
 - Reallocate, sample, ...
- Check toolchain **edges**
 - Spark console, etc.



Synopsis (1)

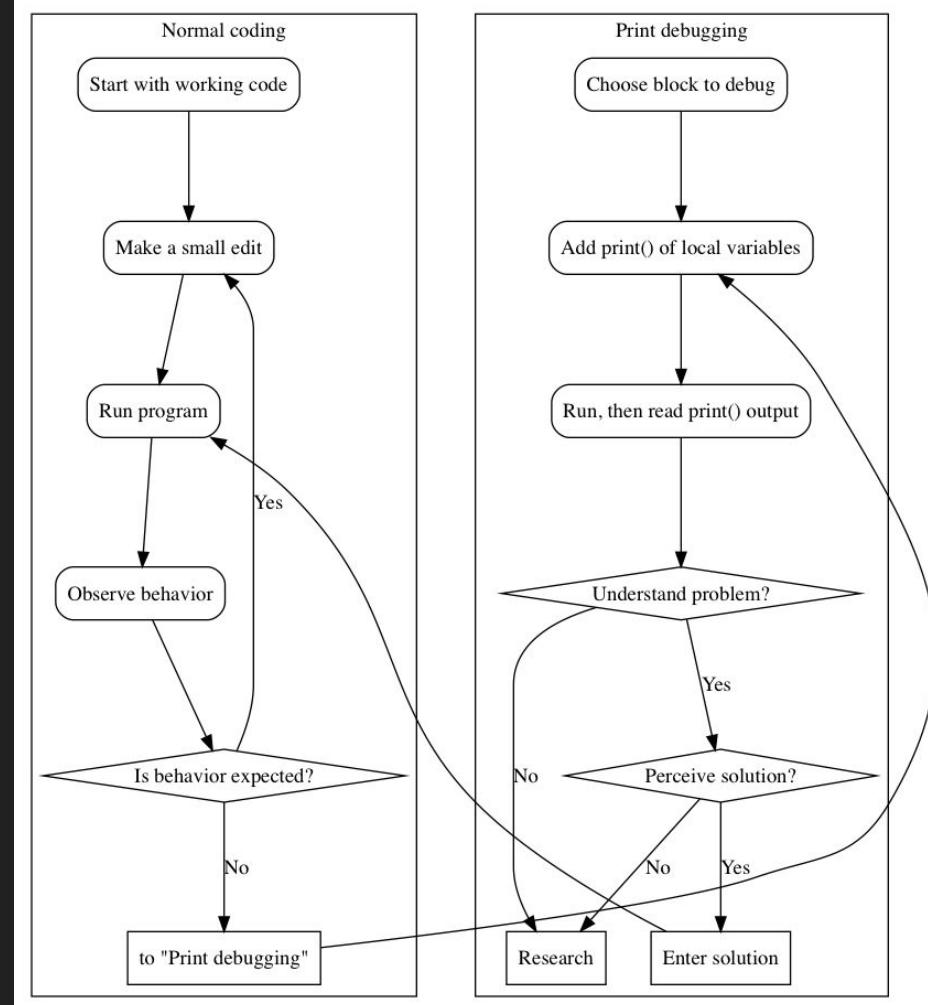
- Three **symptoms**:
 - Crashes, slowness, unexpected results
- Five **causes**:
 - Variance, framework state, mental models, exponential behavior, heavy load
- One **core technique**:
 - **Localize** the **problem state** to observe it
 - **Scale** out and up in **steps**

Synopsis (2)



<https://git.io/decision-tree>

"Start with working code" -> "Make a small edit"
"Make a small edit" -> "Run program"
"Run program" -> "Observe behavior"
"Observe behavior" -> "Is behavior expected?"
"Is behavior expected?" ->
 "Make a small edit"[label="Yes"]
"Is behavior expected?" ->
 "to \"Print debugging\"""[label="No"]



In conclusion

A **possibility** you may not have considered ...

Thank you

www.sampenrose.net | @sampenrose