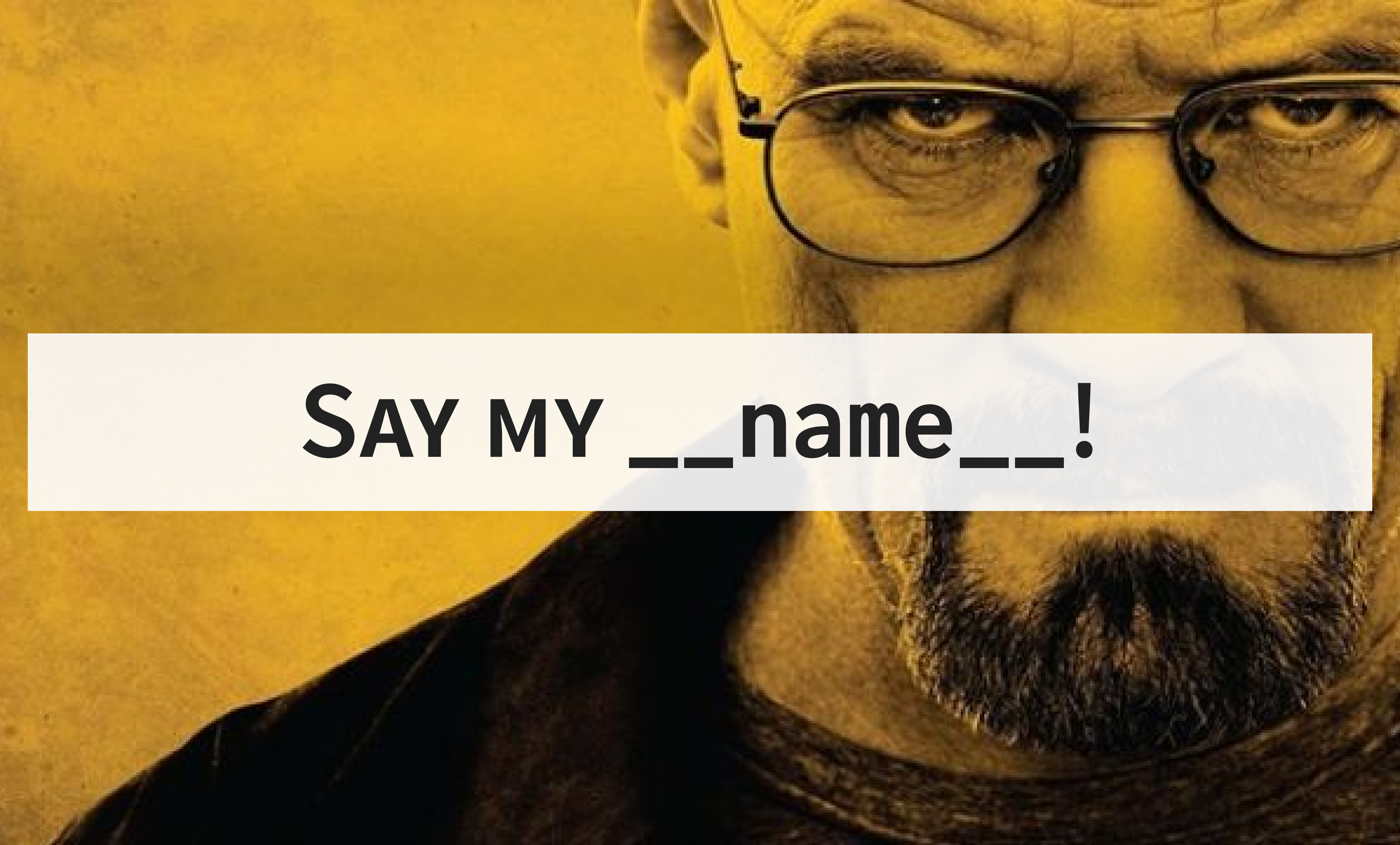
Graphs Redux

Dr. Scott Gorlin
Harvard University
Fall 2018

Agenda

- Say my __name__!
- Data Pipeline
- Advanced Luigi
- Where We Are
- Dask
- Parquet
- Readings


Say my __name__!

A Misunderstood Idiom

```
if __name__ == '__main__':
    ...
```

This may not be what you want!

What's in a __name__?

Files:

```
main.py
package/
    __init__.py
    module.py
```

python main.py:

```
if __name__ == '__main__':
    import sys
    print(sys.path[0]) # /Users/scott/repos/package
    print(__name__) # __main__
    import package
    from package import module
    print(package.__name__) # package
    print(module.__name__)  # package.module
```

Looks ok so far!

What's in a __name__?

package.module:

```
import logging

logger =
logging.getLogger(__name__)

if __name__ == '__main__':
    import sys
    print(sys.path[0])
    import package
```

python package/module.py:

```
/Users/scott/repos/package/package # !!!
Traceback (most recent call last):
  File "package/module.py", line 7, in <module>
    import package
ModuleNotFoundError: No module named 'package'
```

Python changed the python path, and altered the working code!

`module` is treated like a top level module, not part of a package

What's in a __name__?

package.module:

```
import logging

logger =
logging.getLogger(__name__)

if __name__ == '__main__':
    import sys
    print(sys.path[0])
    import package
```

Other programatic usages of __name__ will break!

eg anything any other `from package.module import logger`

Do not directly run packages

Never directly run `python some_package/__init__.py` or any python file that is part of a library. It is not designed for that.

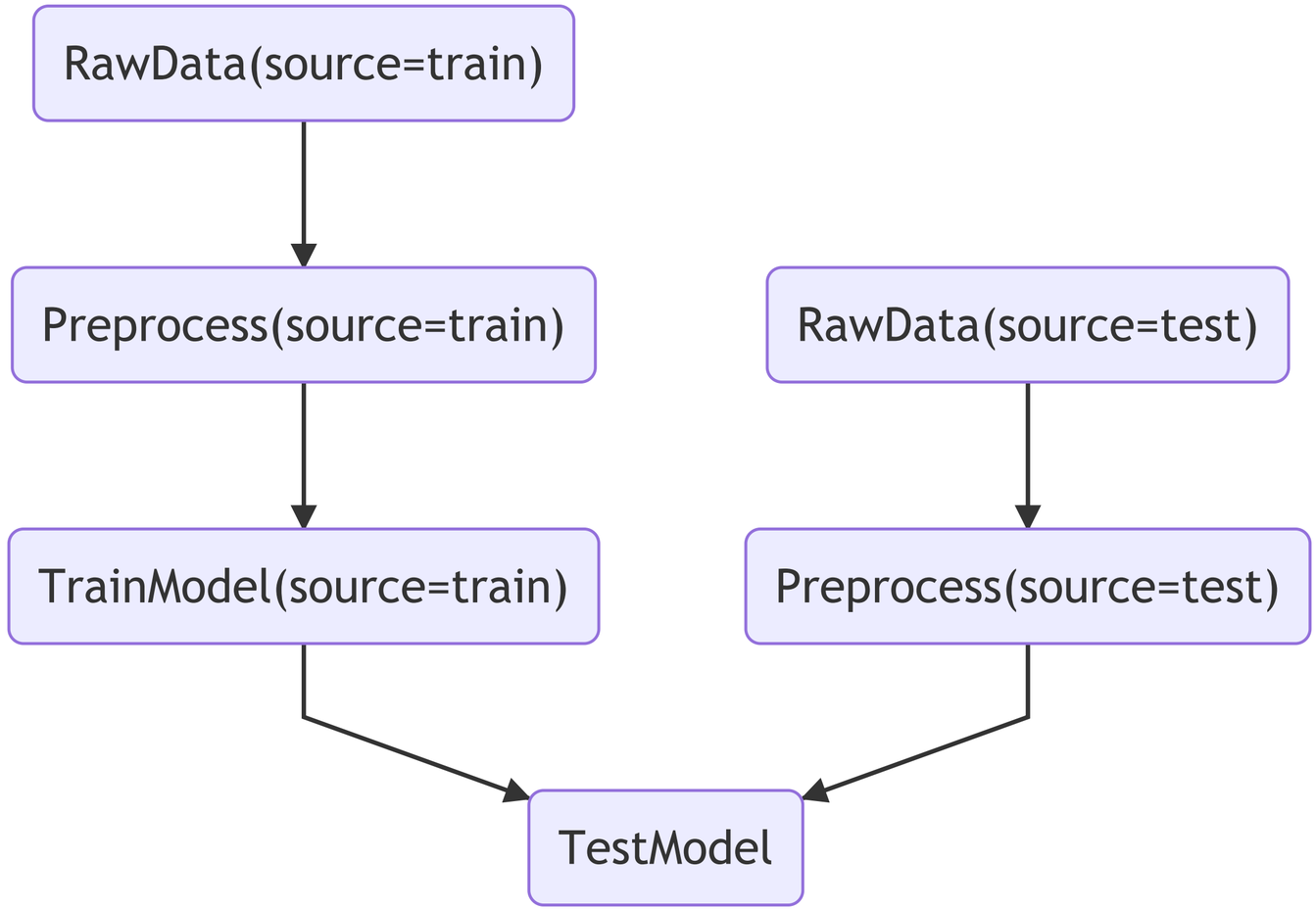
Your IDE can trick you: never ‘debug package/module.py’ directly; use a scratch file, main.py, junk.py, etc at the top of the repo

You can directly run a package with `python -m package` if you implement `package/__main__.py` in a similar form.

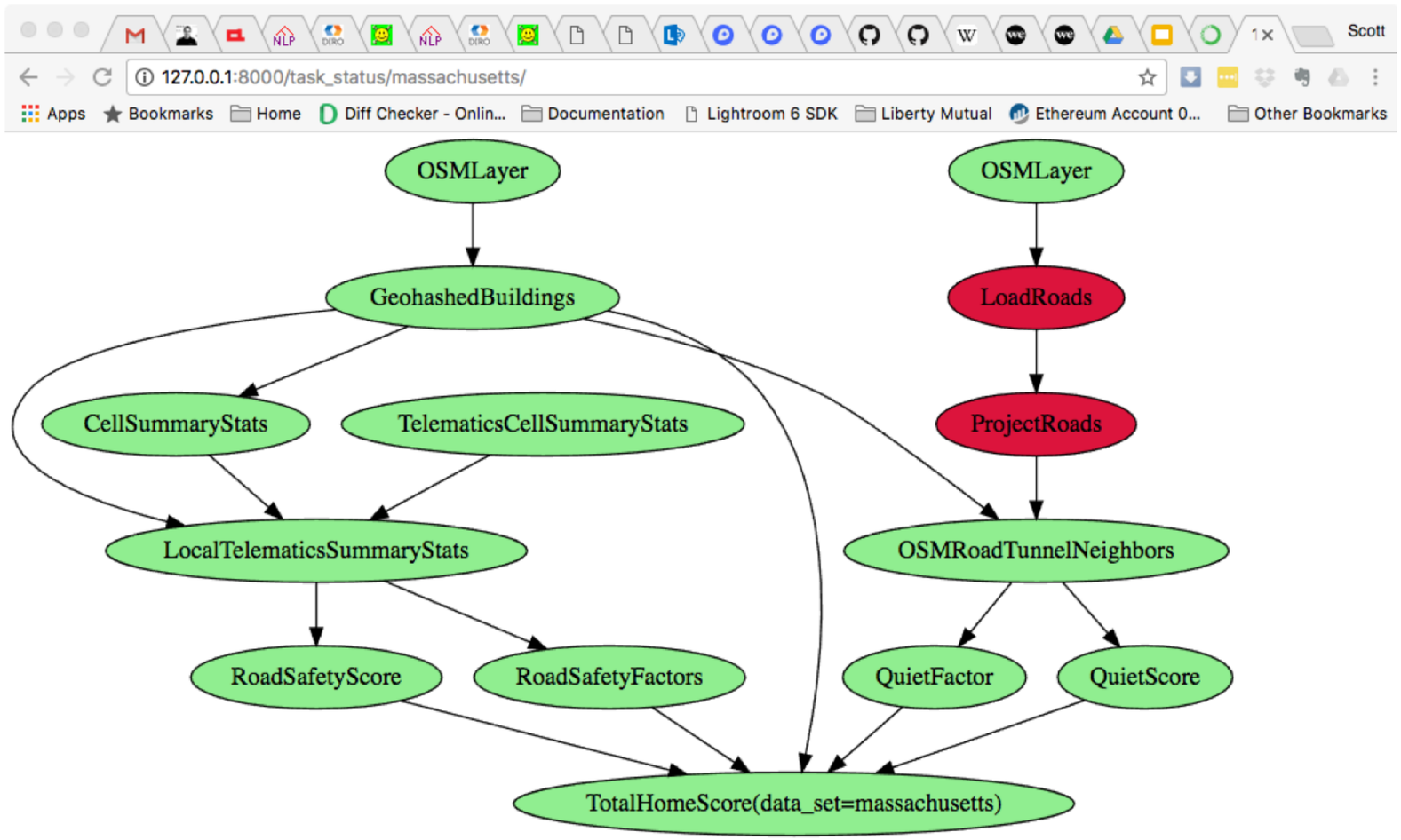
Data Pipeline

The Big Picture

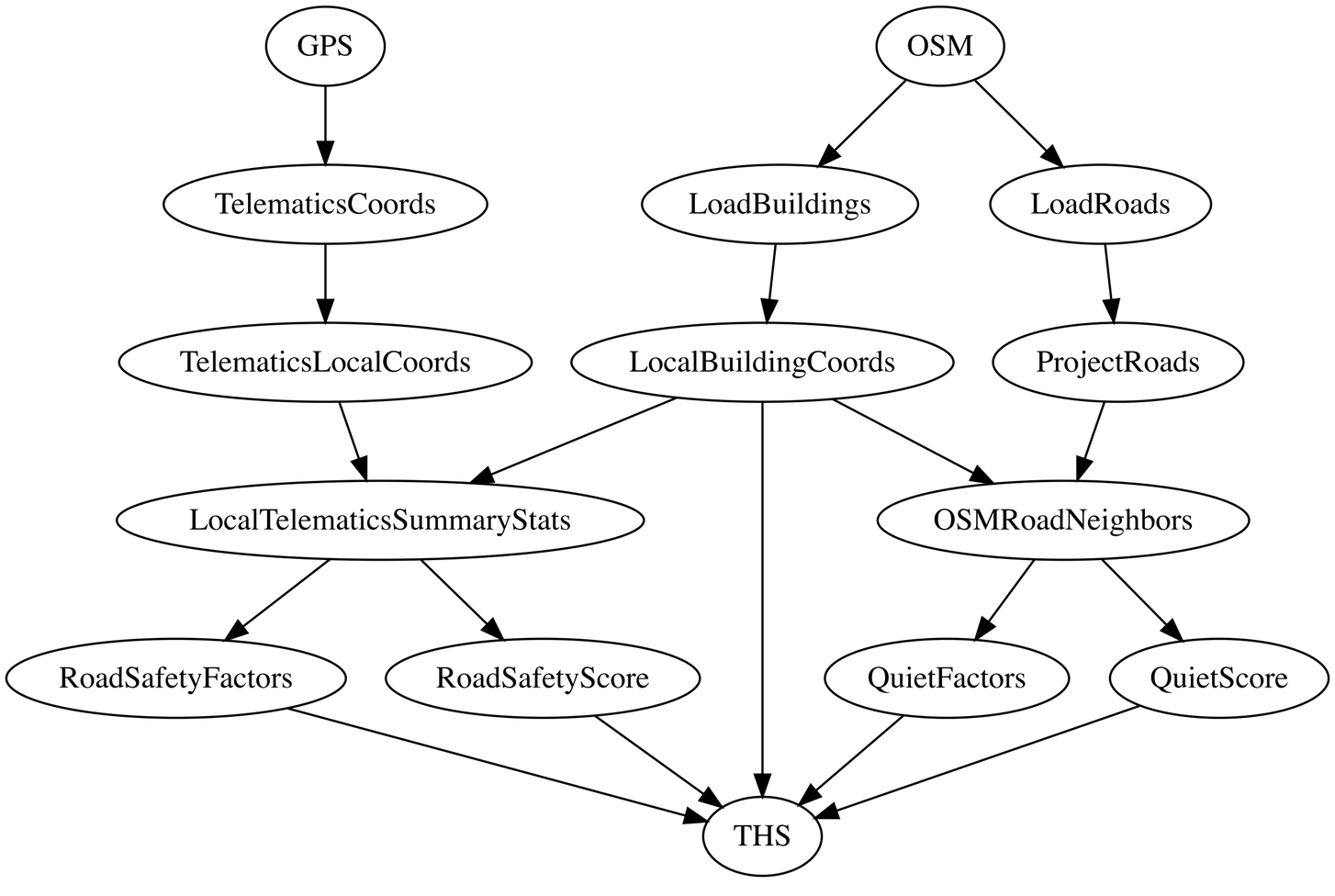
Using Task's, we now can see our entire project from above



The Big Picture

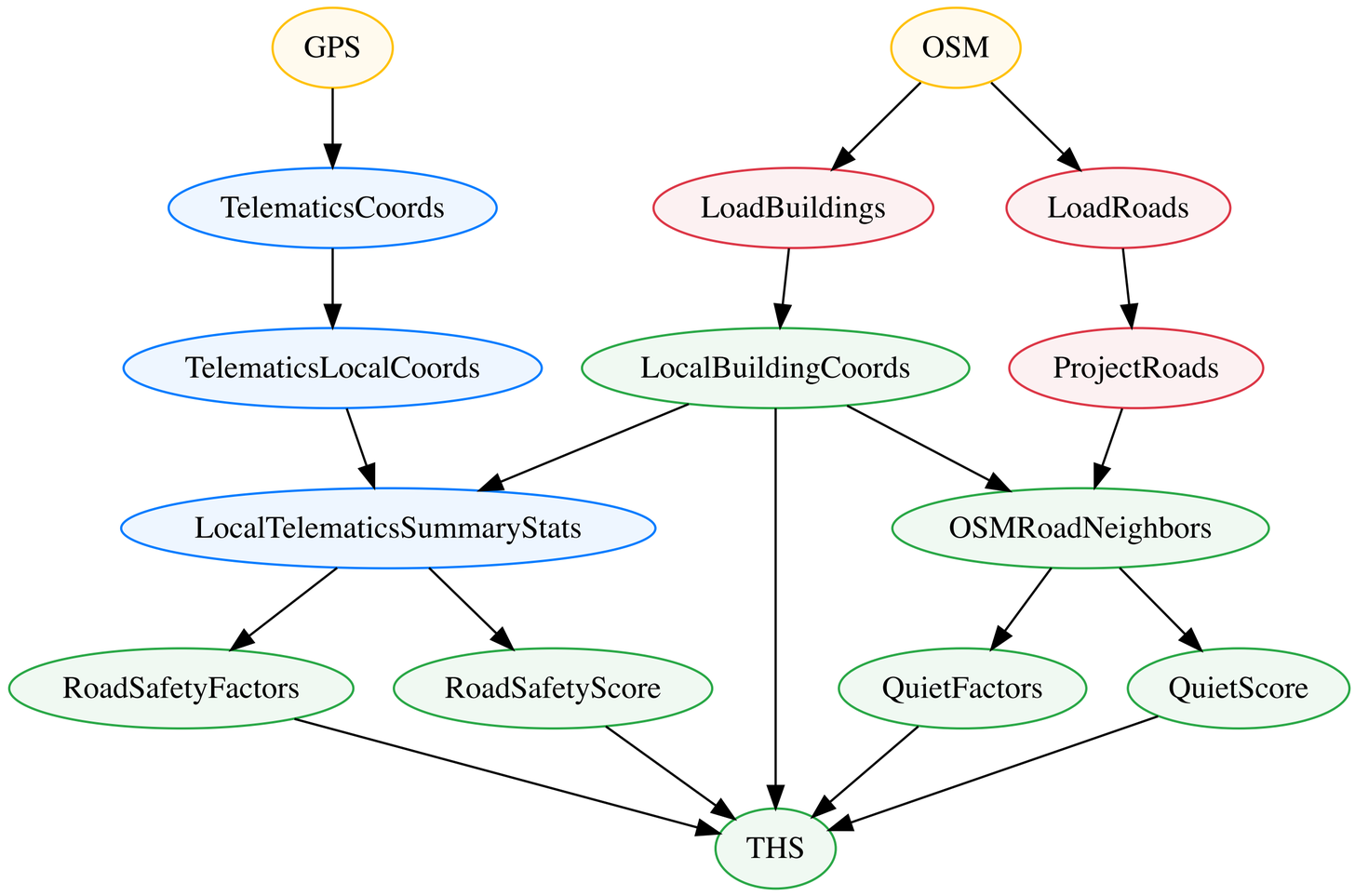


The Big Picture



Total Home Score

The Big Picture



Spark/Parquet **Dask/Parquet** **External** **Django/SQL**

Scheduling

Luigi

Requirements

Including dynamic dependencies

Outputs

Including logic, salting, etc

Executors

Running everything

Delegated: `.run()`

Just pass the target paths!

Spark

Kick off MR/ETL

Oozie

Another scheduling system!

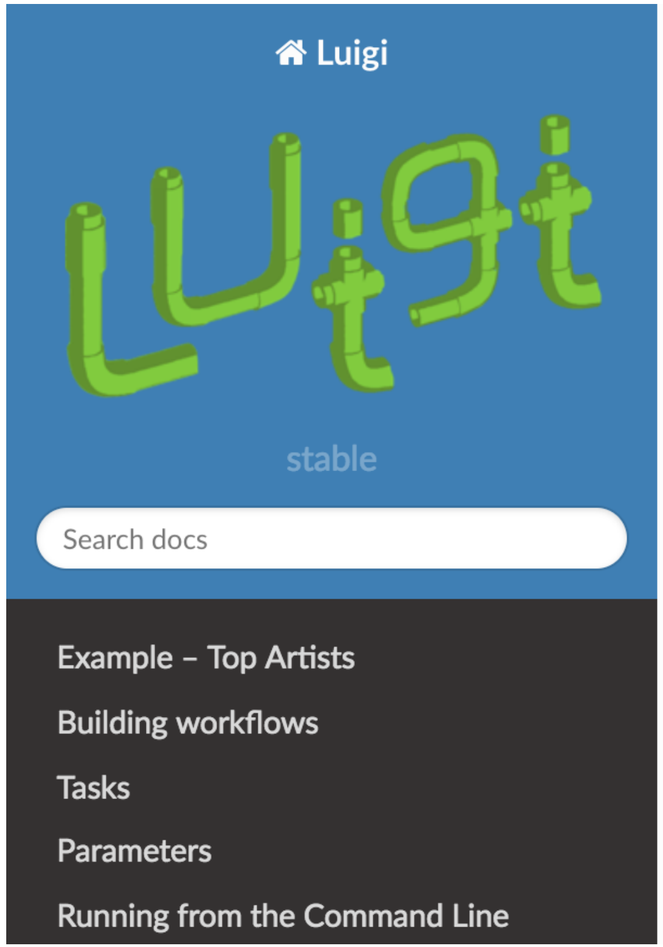
…

Advanced Luigi

Task Types

So far we've seen `Task` and `ExternalTask`, but many more exist!

Luigi has a rich set of contributions, mostly for running non-python tasks



Submodules

- luigi.contrib.batch module
- luigi.contrib.bigquery module
- luigi.contrib.bigquery_avro module
- luigi.contrib.dataproc module
- luigi.contrib.docker_runner module
- luigi.contrib.ecs module
- luigi.contrib.esindex module
- luigi.contrib.external_program module
- luigi.contrib.ftp module
- luigi.contrib.gcp module
- luigi.contrib.gcs module
- luigi.contrib.hadoop module
- luigi.contrib.hadoop_jar module
- luigi.contrib.hive module
- luigi.contrib.kubernetes module
- luigi.contrib.lsf module

External Programs

```
# luigi.contrib.external_program
class ExternalProgramTask(Task):

    def program_args(self):
        raise NotImplementedError()


    def run(self):
        args = self.program_args()

        logger.info(
            'Running command: %s',
            ' '.join(args))
        subprocess.check_call(args)
```

```
# luigi.contrib.spark
class SparkSubmitTask(ExternalProgramTask):
    def program_args(self):
        return [
            'spark-submit',
            '--name', self.name,
            self.app
            ...
        ]

class PySparkSubmitTask(SparkSubmitTask):
    ...
```

External Programs

Your luigi target doesn't control how an external program writes data - it can only point to the data once written.

You should choose a target that matches the atomicity pattern of the program.

Eg, `luigi.contrib.S3FlagTarget` which looks for a `_SUCCESS` flag left by many hadoop/EMR jobs, including spark.

External Programs

Programs come in two flavors:

Local/Wrapped Runner

The job process starts and locally succeeds or fails

Examples: `python main.py`, `spark --deploy-mode client`

Dispatched Runner

Process submits a remote job and successfully terminates

The task output could be job id; next task queries job status

(out of scope for this class)

Examples: Oozie, EMR step, `spark --deploy-mode cluster`

The Inevitable Creep

Task Executor Types

- External tasks
- Spark tasks
- SQL tasks
- Kube tasks
- Docker tasks

Output Types

- Local targets
- S3/hdfs Targets
- Partitioned Targets
- DB Targets

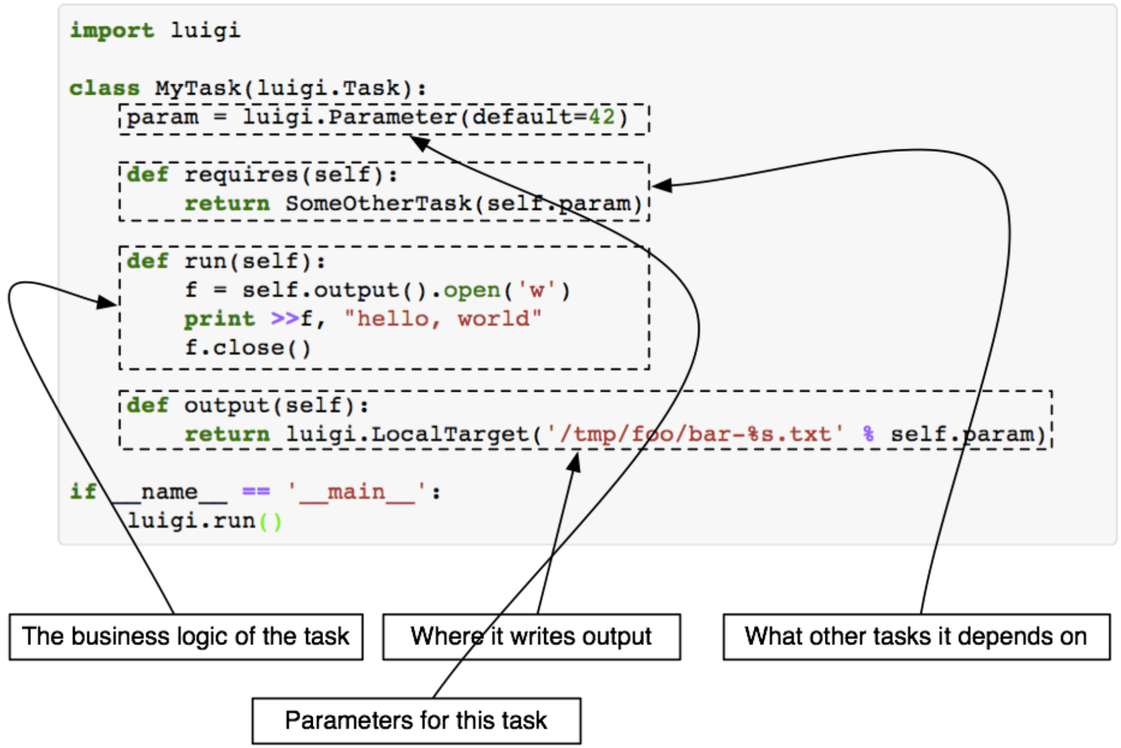
Variants

- Salted targets
- Your task hierarchy

Your tasks have a high risk of multi inheritance!

Task Skeleton

- Parameters
  - Already declarative
- Run
  - Seems like the main job of the task, keep as a method
- Requires, Output
  - Seem ripe for declarative composition



Requirements

Requirements can be a single task, a list, or a dict

They often share some parameters with the task, and should be formed with `task.clone(requirement_class, **param_overrides)`

Requirements

We can implement this as a descriptor and a registry!

```
class Requirement:
    def __init__(self, task_class, **params):
        ...

    def __get__(self, task, cls):
        return task.clone(
            self.task_class,
            **self.params)
```

```
class Requires:
    def __get__(self, task, cls):
        return lambda : self(task)

    def __call__(self, task):
        # Search task.__class__ for Requirements
        # return instances
```

```
class MyTask(Task):
    # Replace task.requires()
    requires = Requires()
    other = Requirement(OtherTask)

    def run(self):
        # Convenient access here...
        with self.other.output().open('r') as f:
            ...

>>> MyTask().requires()
{'other': OtherTask()}
```

This will be on your next pset!

Output

Output tends to follow a few rules you can choose from:

1. Is there a relationship between the file path and the task name?
    - eg, `{DATA_ROOT}/{param}/{task_name}.txt`, with a configurable root directory
2. Is the output salted?
3. Do you follow defaults, like encrypting things?
4. Is output all local, S3 targets, etc? Do you need special authentication?

The output logic probably isn't all that related to your task classes, and likely can be solved using composition

Output

```
class LocalTargetOutput:
    def __init__(self, file_pattern='{task}.txt'):
        ...
    def __get__(self, task, cls):
        return lambda: LocalTarget(...)
```

```
class MyTask:
    # Compose output
    output = LocalTargetOutput()

>>> MyTask().output()
LocalTarget()
```

Where We Are

```cython
def primes(int nb_primes):
    cdef int n, i, len_p
    cdef int p[1000]
    if nb_primes > 1000:
        nb_primes = 1000

    len_p = 0  # The current number of elements in p.
    n = 2
    while len_p < nb_primes:
        # Is n prime?
        for i in p[:len_p]:
            if n % i == 0:
                break
        else:
            p[len_p] = n
            len_p += 1
        n += 1

    # Let's return the result in a python list:
    result_as_list  = [prime for prime in p[:len_p]]
    return result_as_list
```

“Python”

- We have repeatable and appropriately specified virtual environments
- We can choose between minimal reqs and frozen dependencies
- We understand the role of `pipenv` for library and app development

CODE

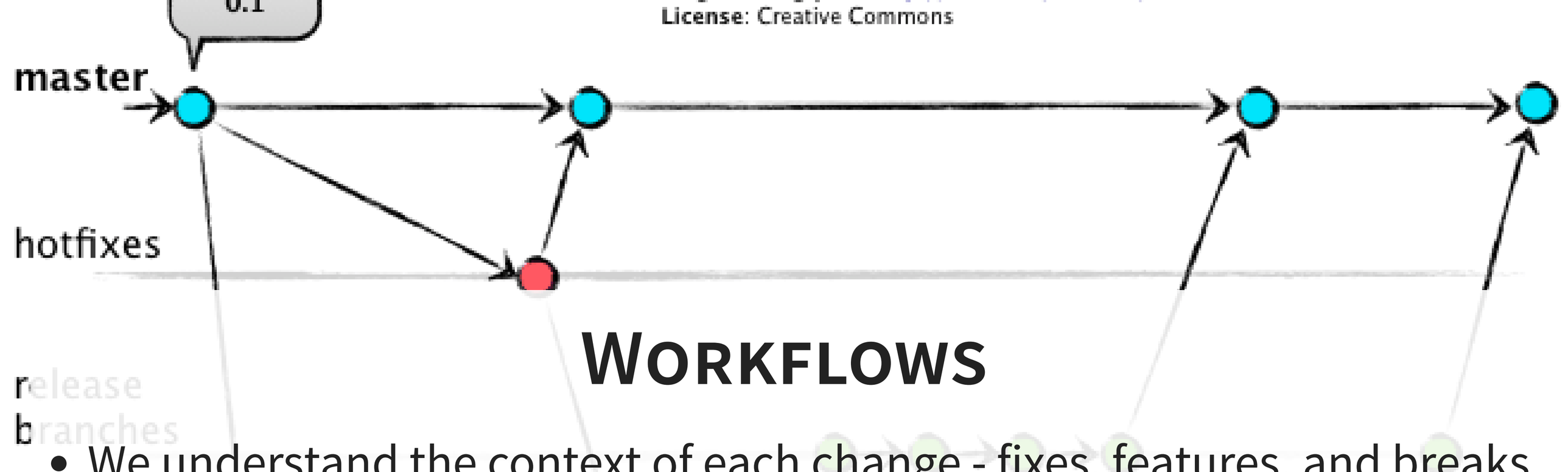
DEPLOY

Testing

- We know the importance of unit testing, how to test, and what to test
- We can measure coverage, including code branches
- We know the environment matters, and test libraries where it counts
- We can mock out code, even faking integration with other systems

TEST

MONITOR



Workflows

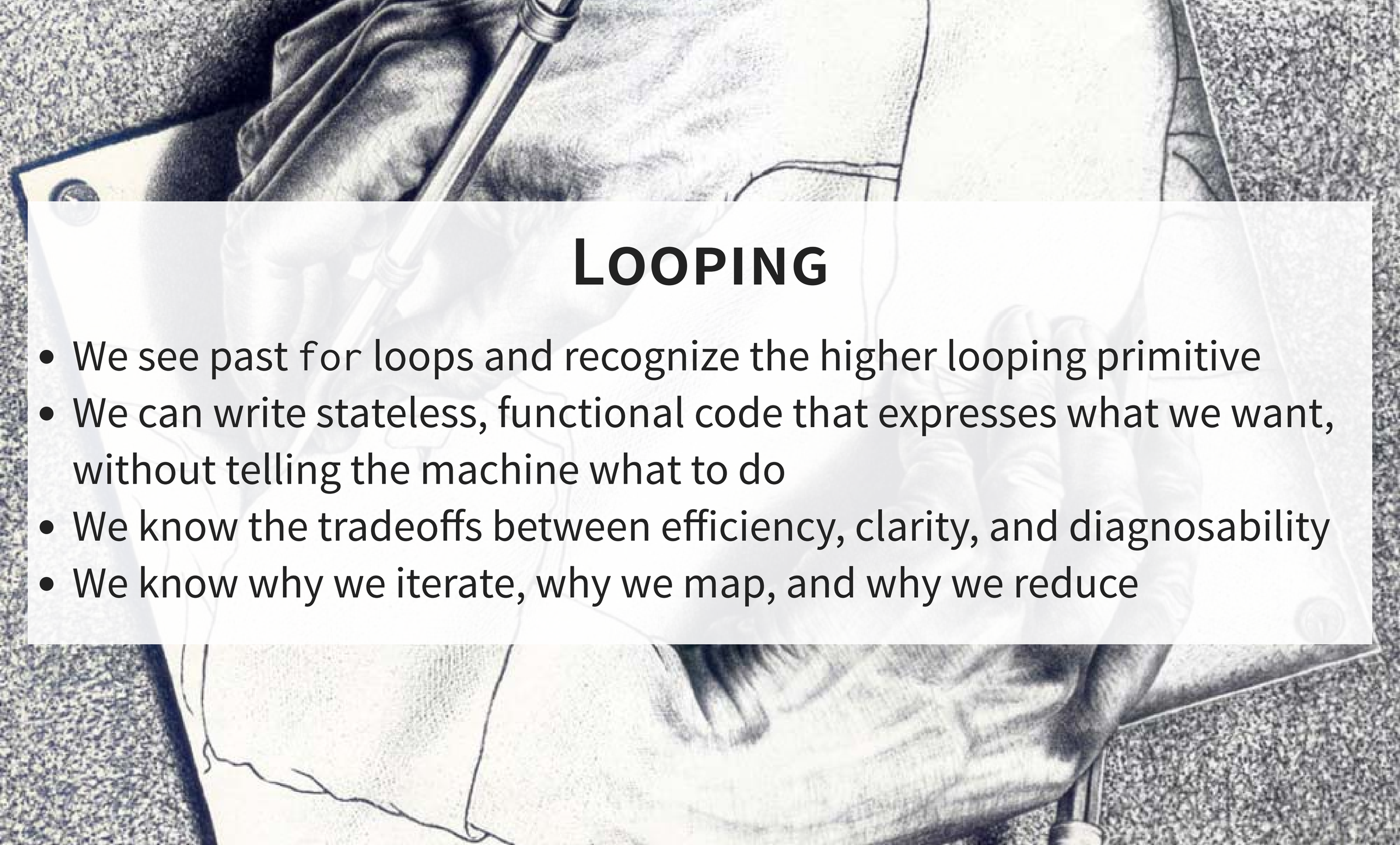
- We understand the context of each change - fixes, features, and breaks
- We track history linearly and meaningfully using semantic versions
- We bootstrap our work and codify best practices with templates

Higher Levels

- We recognize meta-patterns in code at levels higher than a function
- We wrap, register, and alter functions using decorators
- We provide reusable context to code using context managers

Deployment

- We know what is part of our code, and what is not
- We configure our deployments with environment variables, ensuring our code is useable anywhere
- We handle data and secrets with discretion and privacy

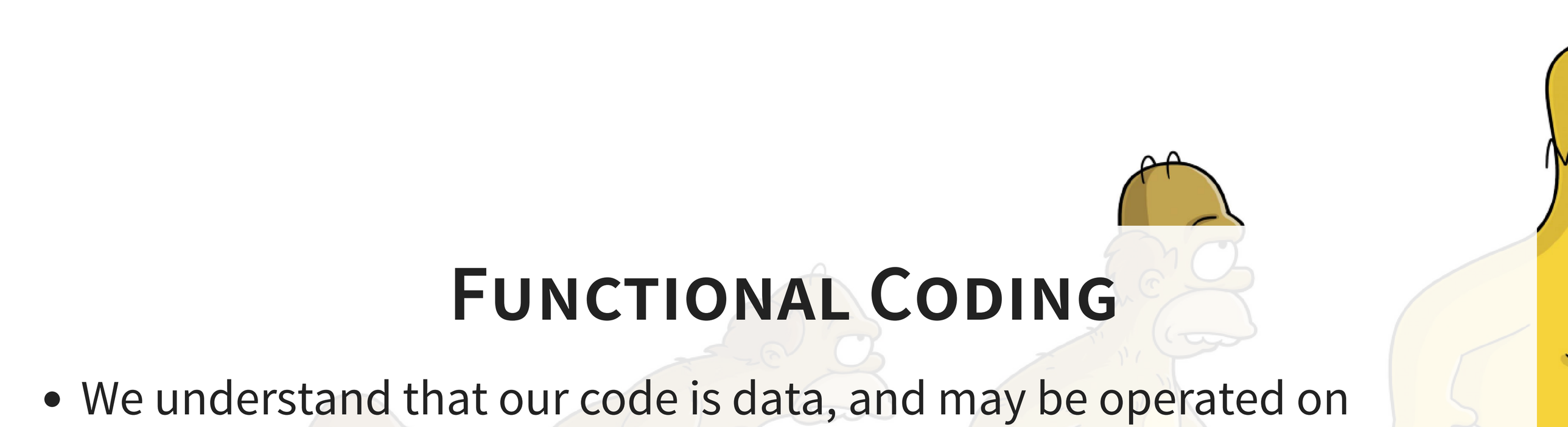


Looping

- We see past `for` loops and recognize the higher looping primitive
- We can write stateless, functional code that expresses what we want, without telling the machine what to do
- We know the tradeoffs between efficiency, clarity, and diagnosability
- We know why we iterate, why we map, and why we reduce

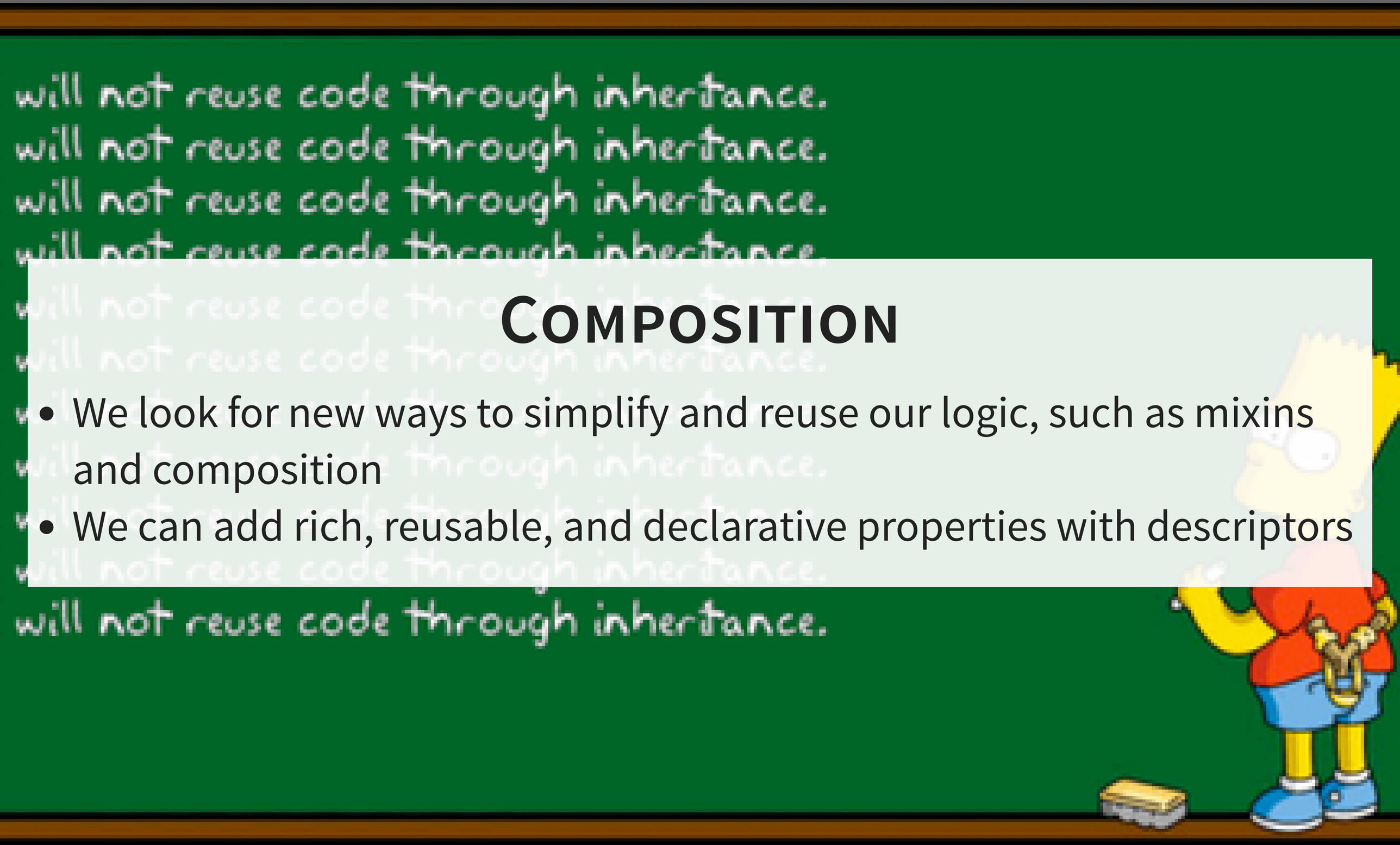
Functional Coding

- We understand that our code is data, and may be operated on
- We know when state is valuable, and when it fails us
- We can encapsulate logic in functions that we pass as arguments to higher frameworks
- We strive to be declarative in all that we do



Composition

- We look for new ways to simplify and reuse our logic, such as mixins and composition
- We can add rich, reusable, and declarative properties with descriptors



Graphical Programs

- We recognize our programs, literally, as directed graphs
- We can directly visualize, debug, and optimize a graph
- We can construct graphs to represent our ideas and programs

Graphical Programs

- We can scaffold our application at the highest level using dataflows
- We can solve new and extremely important problems using the graph, such as seen with Salted

END OF LIST!

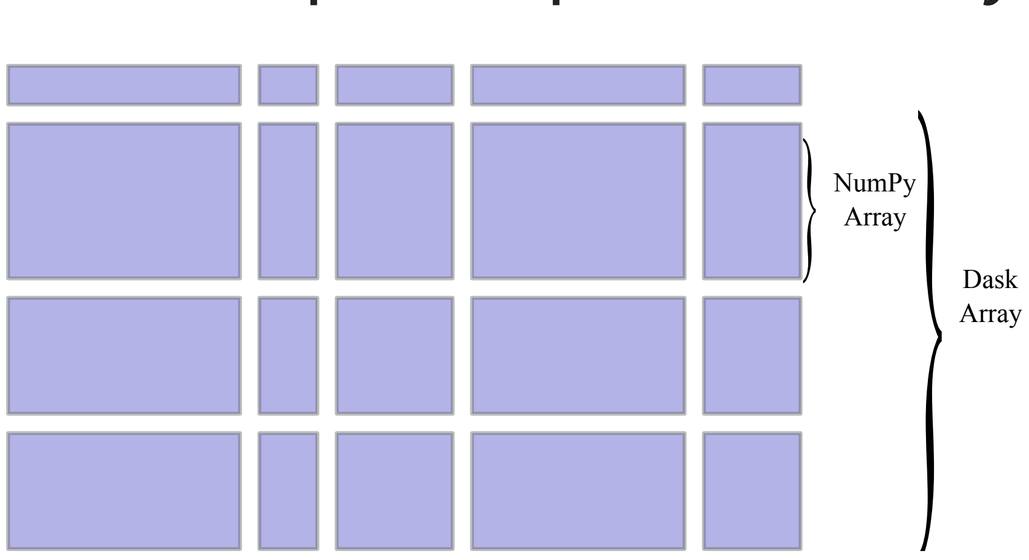
Midterm scope ends here

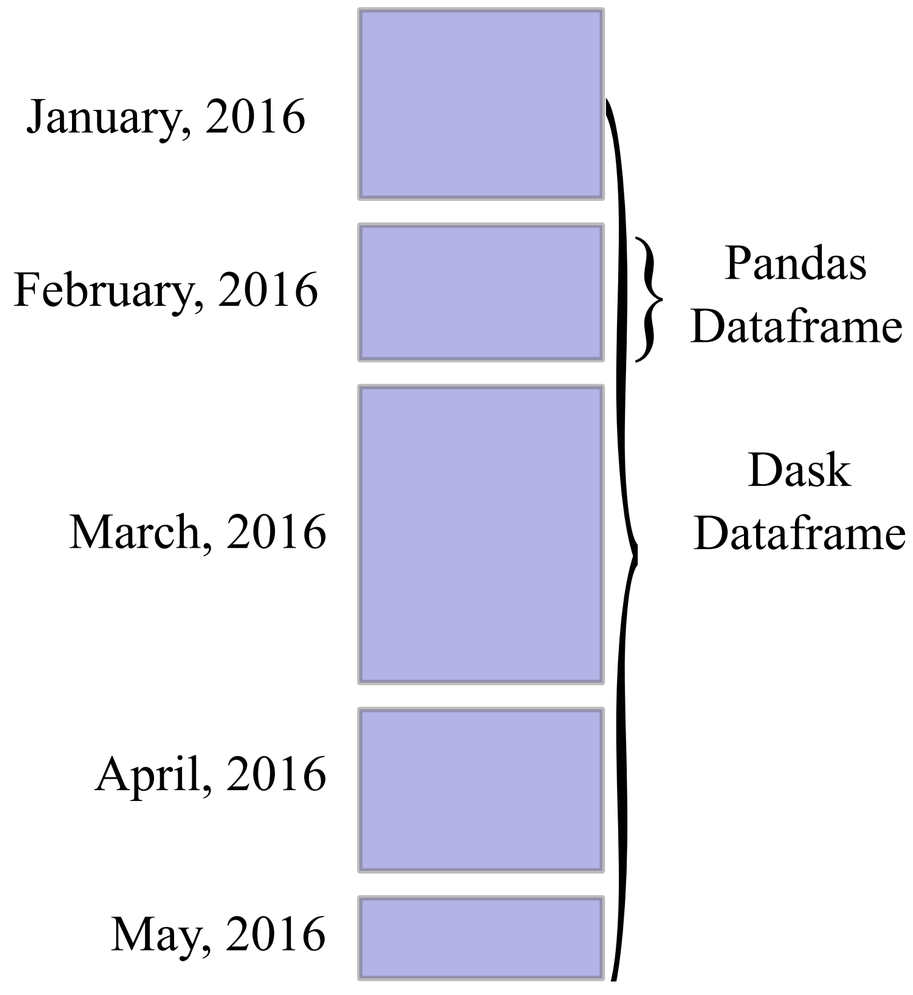
Dask

Dask

Computational Graphs for Python

- Out-Of-Memory numpy, pandas, text
- Chunked, distributed, delayed
- Can replace Spark for many workflows





dask.DataFrame

A partitioned, symbolic pandas frame

```
>>> import dask.dataframe as dd
>>> df = dd.read_csv('2014-*.csv')
>>> df.head()  # Implicit 'compute'
   x  y
0  1  a
1  2  b
2  3  c
3  4  a
4  5  b
5  6  c
```

```
>>> df
Dask DataFrame Structure:
                   x       y
npartitions=3
None             int64  object
None               ...     ...
None               ...     ...
None               ...     ...
Dask Name: from-delayed, 9 tasks
```

We don't notice it, but dask ***has not*** read the data yet. It did some brief checking to get row headers and column types, but data is not yet in memory.

dask.DataFrame

A partitioned, symbolic pandas frame

```
>>> pdf = pd.read_excel('hashed.xlsx')
>>> df = dd.from_pandas(pdf, chunksize=10)
>>> df
Dask DataFrame Structure:
                learn project
npartitions=5
0d733444       object  object
33174d90          ...     ...
...               ...     ...
d9a774d1          ...     ...
ffc21800          ...     ...
Dask Name: from_pandas, 5 tasks
```

```
>>> df.loc['57019809']
Dask DataFrame Structure:
                learn project
npartitions=1
57019809       object  object
57019809          ...     ...
Dask Name: loc, 6 tasks
```

Notice the ***predicate pushdown***. This row index must fall within a single partition, so dask only returns that.

dask.DataFrame

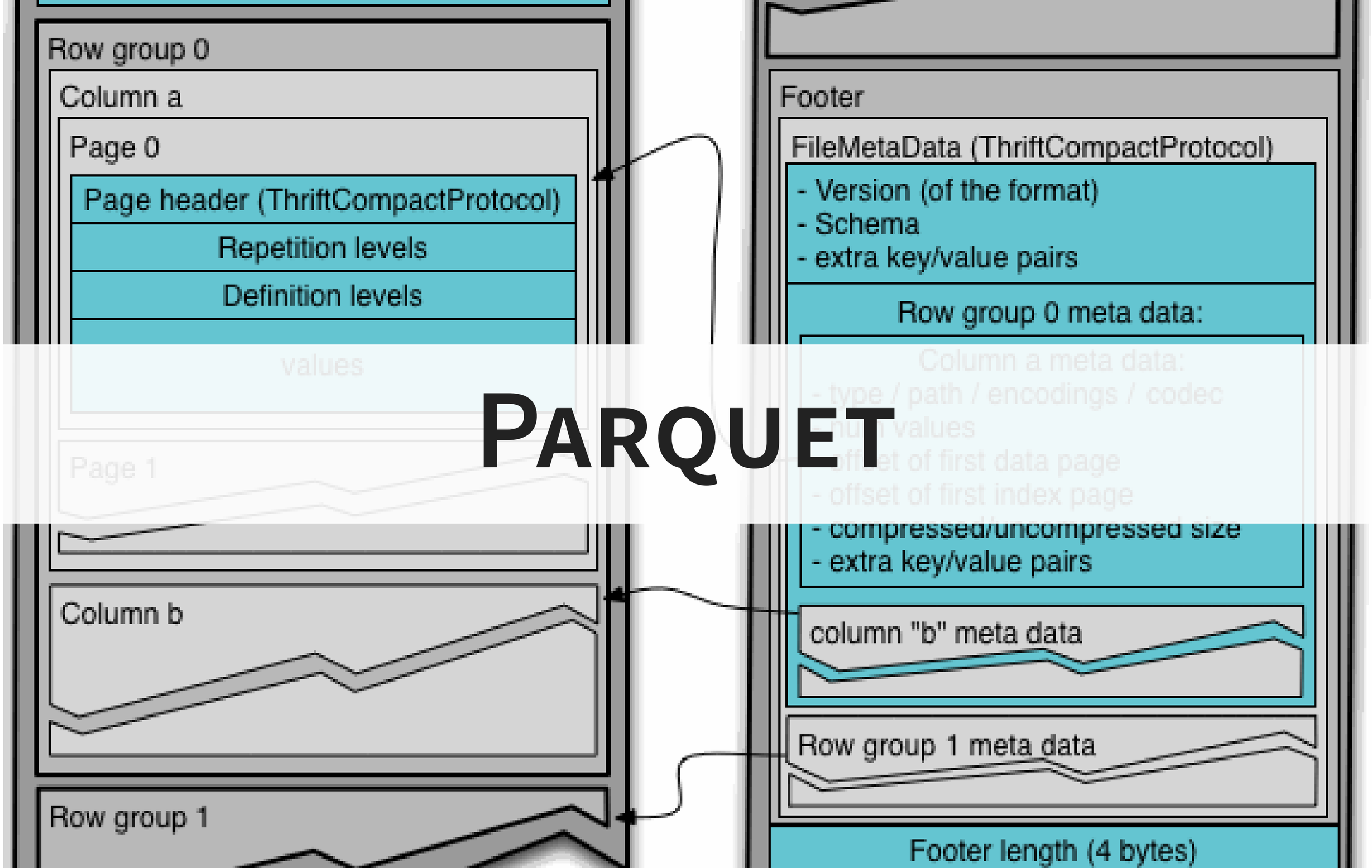
Note the _meta!

```
>>> df = pd.DataFrame(
    {'a': [1, 2, 3],
    'b': ['x', 'y', 'z']})
>>> ddf = dd.from_pandas(
    df, npartitions=2)
>>> ddf._meta
Empty DataFrame
Columns: [a, b]
Index: []
>>> ddf._meta.dtypes
a     int64
b    object
dtype: object
```

```
# Apply function to every partition
ddf.map_partitions(
    foo,
    # Provide meta if necessary
    meta=pd.DataFrame(
        {'a': [1], 'b': [2]}
    )
) # cf ddf.apply()
```

Dask must know the shape of your delayed data, and can't always guess.

Provide it when you can


Row group 0
Column a
Page 0
Page header (ThriftCompactProtocol)
Repetition levels
Definition levels
values
Page 1
Column b
Row group 1
Footer
FileMetaData (ThriftCompactProtocol)
- Version (of the format)
- Schema
- extra key/value pairs
Row group 0 meta data:
Column a meta data:
- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs
column "b" meta data
Row group 1 meta data
Footer length (4 bytes)

PARQUET

Column Stores

Row Store

Vertical Partitions

id	name	grade
1	Scott	89
2	Andrew	91
99	Mary	87

Column Store

Vertical *and* horizontal partitions

id	name
1	Scott
2	Andrew
99	Mary

id	grade
1	89
2	91
99	87

Most operations only care about a few columns

Don’t read them all if you don’t need to!

Column Stores

Any table...

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

Row Storage

Any csv:

A1	B1	C1	A2	B2	C2	A3	B3	C3

Column Storage

A better format:

A1	A2	A3	B1	B2	B3	C1	C2	C3

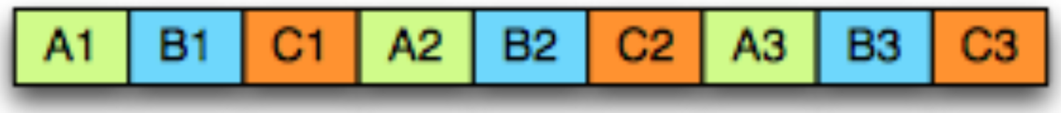
Column Stores

Column stores offer superior:

- Compression, due to homogeneity
- Selective column reads
- IO, due to buffering

Row Storage

Any csv:



Column Storage

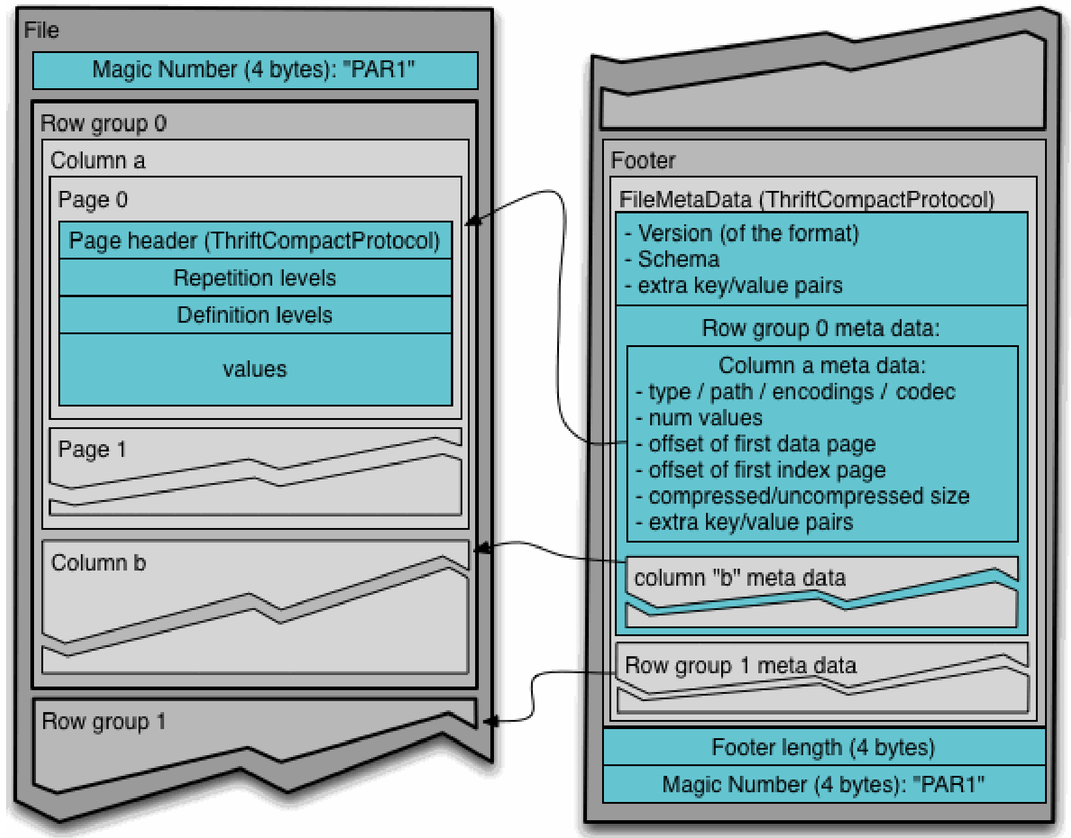
A better format:

Apache Parquet

Emerging standard for distributed, columnar binary data

Statistics for predicate pushdown

http://parquet.apache.org/



Dataframes ♥ Parquet

```
read_parquet(
    path,
    # Only read what you want!
    columns=['a', 'b'],

    # Filter what you read!
    filters=[
        ('a', '>', 5)
    ],
)
```

Statistics

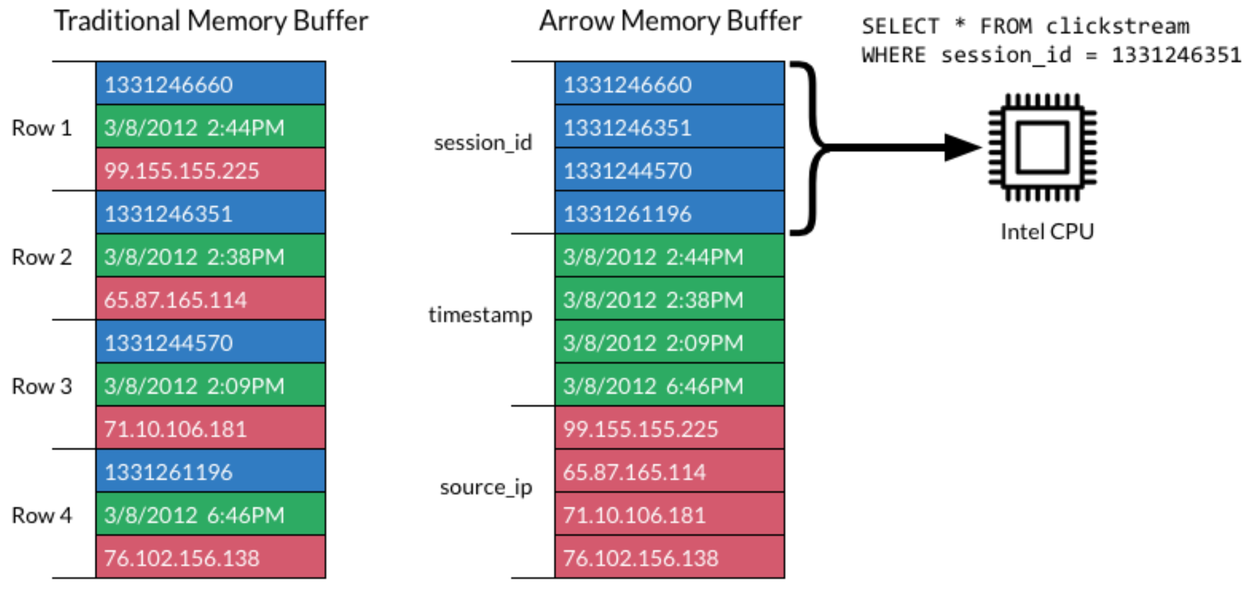
Parquet stores partition min/max values to allow for index and block filtering too!

Apache Arrow

Unification of column stores - Pandas, Parquet, Spark, …

http://arrow.apache.org/

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138



The Players

Parquet

Established on-disk columnar standard

You will use directly and extensively

Arrow

Emerging in-memory standard

Basis for Pandas2

Integrated into Parquet code base

Exciting, but you likely won't touch directly

Feather

POC minimal disk format for arrow

We don't care

Readings

For post midterm:

- Dask Overviews (esp DataFrame)
- Apache Arrow and the “10 Things I Hate About pandas”
- Apache Arrow and Apache Parquet
- Dremel made simple with parquet
- Skim slides: The Columnar Roadmap