PONDER

Scaling up your pandas workflows with Modin

Devin Petersohn

Co-founder and CTO, Ponder

About me

- Active Duty U.S. Marine
  - Korean Crypto-Linguist
  - 3d Radio BN (2008-2012)
- BS - University of Missouri (2016)
- MS - UC Berkeley (2018)
- PhD - UC Berkeley (2021)
  - NSF Graduate Research Fellow
  - Chancellor's Fellow
  - Modin started as my PhD project
- Currently: Cofounder and CTO of Ponder

PONDER

A dataframe built from first principles

Data Science Organization problems

What problems do data science teams face?

Data Science has a scalability problem

Data Science has a scalability problem

Data ~~Science~~ Teams* have a scalability problem

Data ~~Science~~ Teams* have a scalability problem

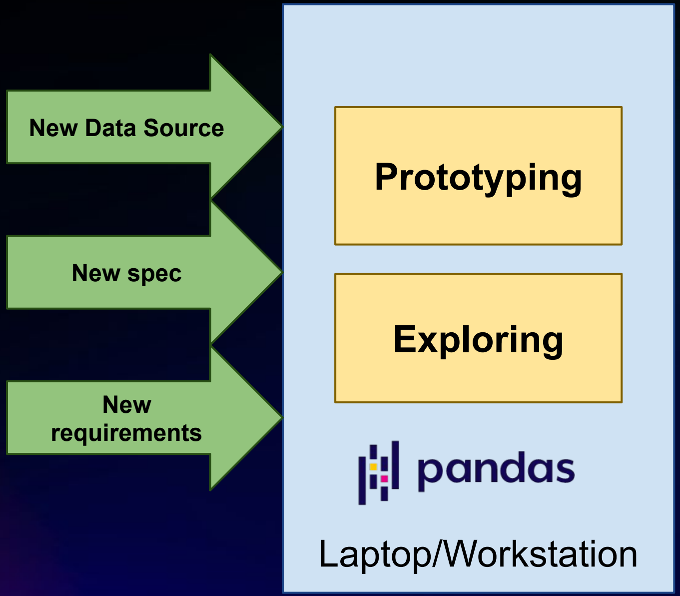
PONDER

MORE DATA SCIENTISTS != MORE INSIGHTS

Data ~~Science~~ Teams* have a scalability problem

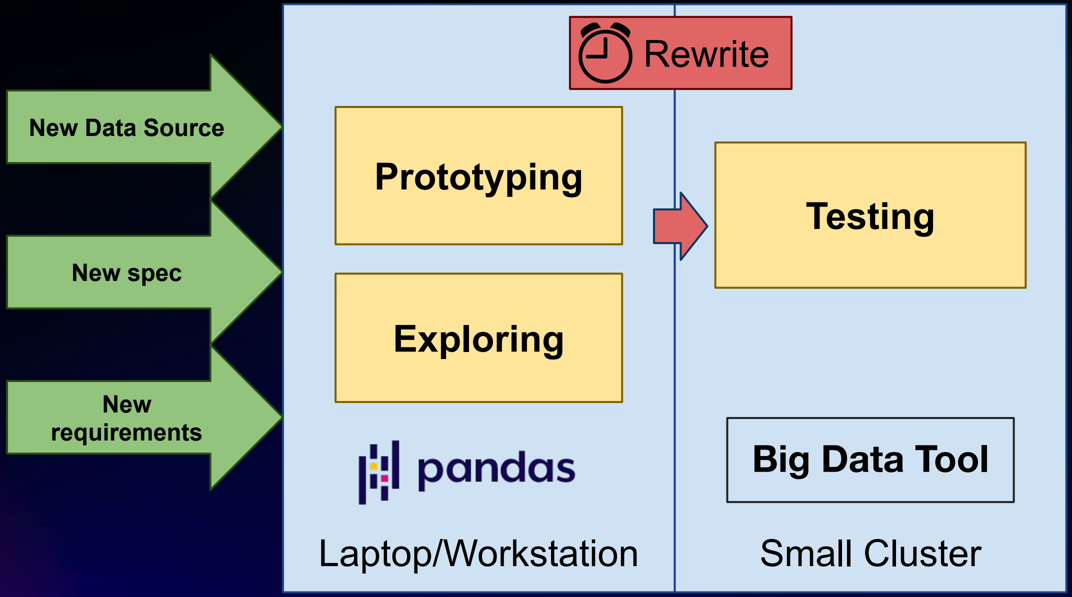
Many organizations look like this

PONDER

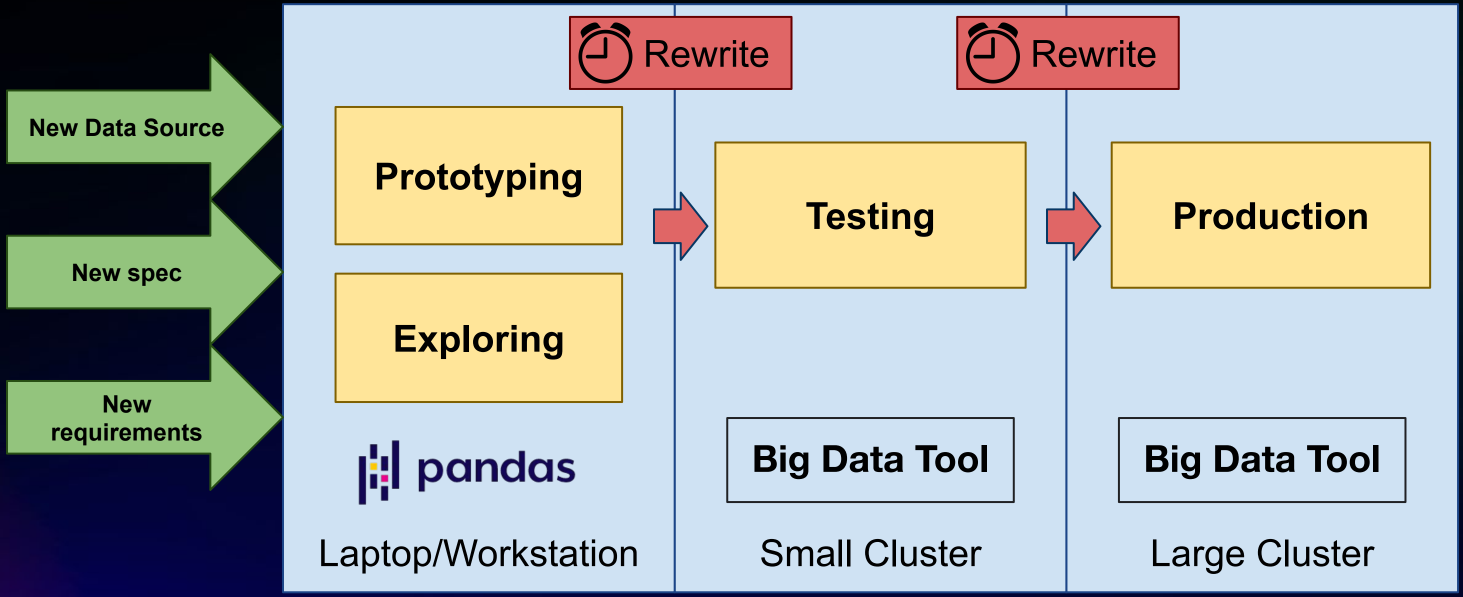


Many organizations look like this

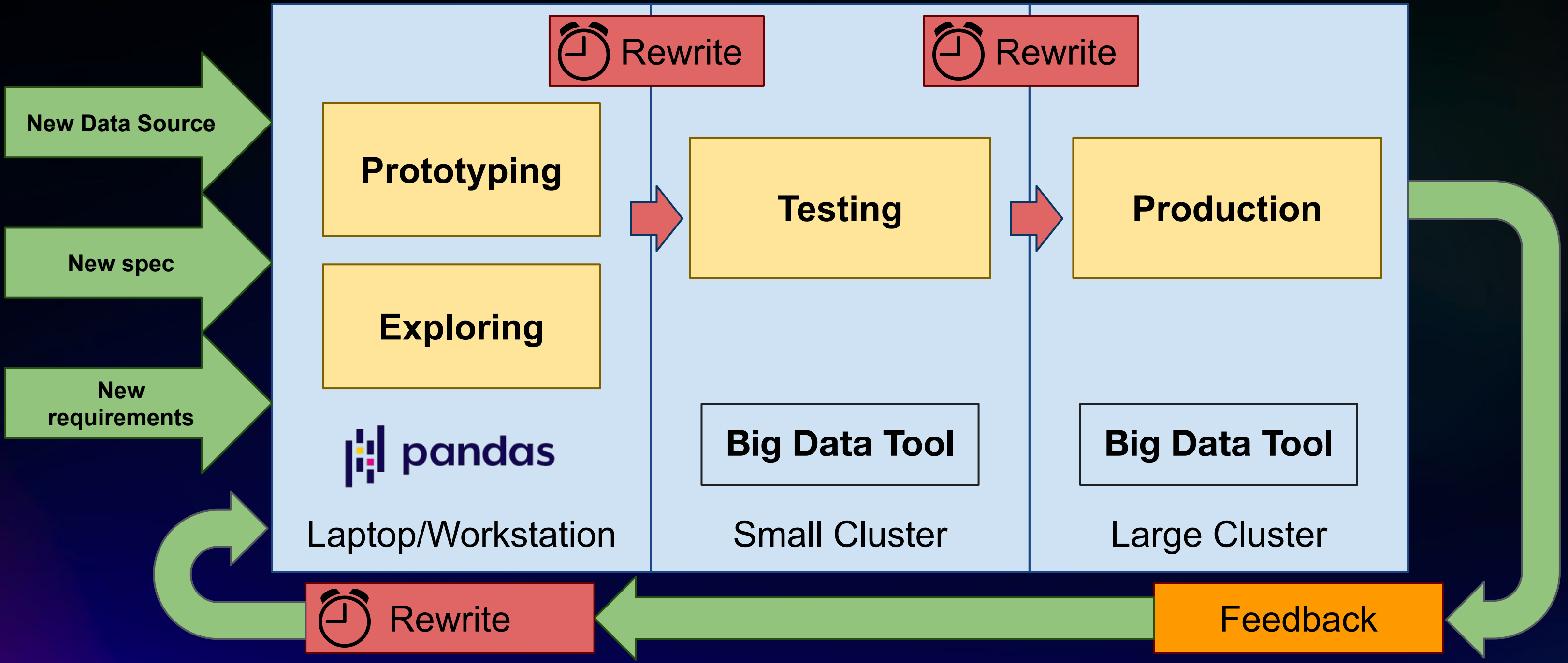
PONDER

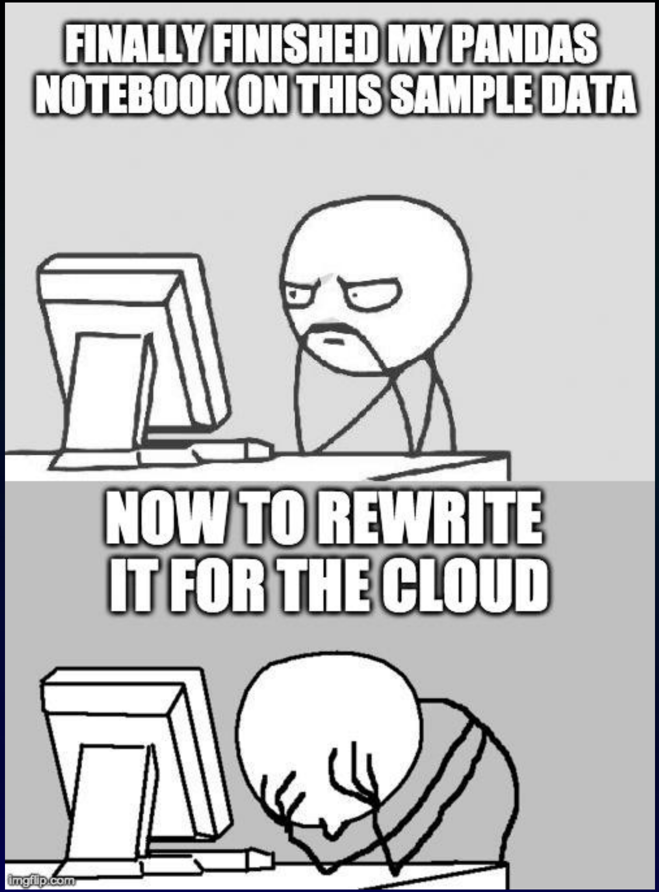


Many organizations look like this



Many organizations look like this




FINALLY FINISHED MY PANDAS NOTEBOOK ON THIS SAMPLE DATA
NOW TO REWRITE IT FOR THE CLOUD
imgflip.com

PONDER

Data Science scalability is human scalability

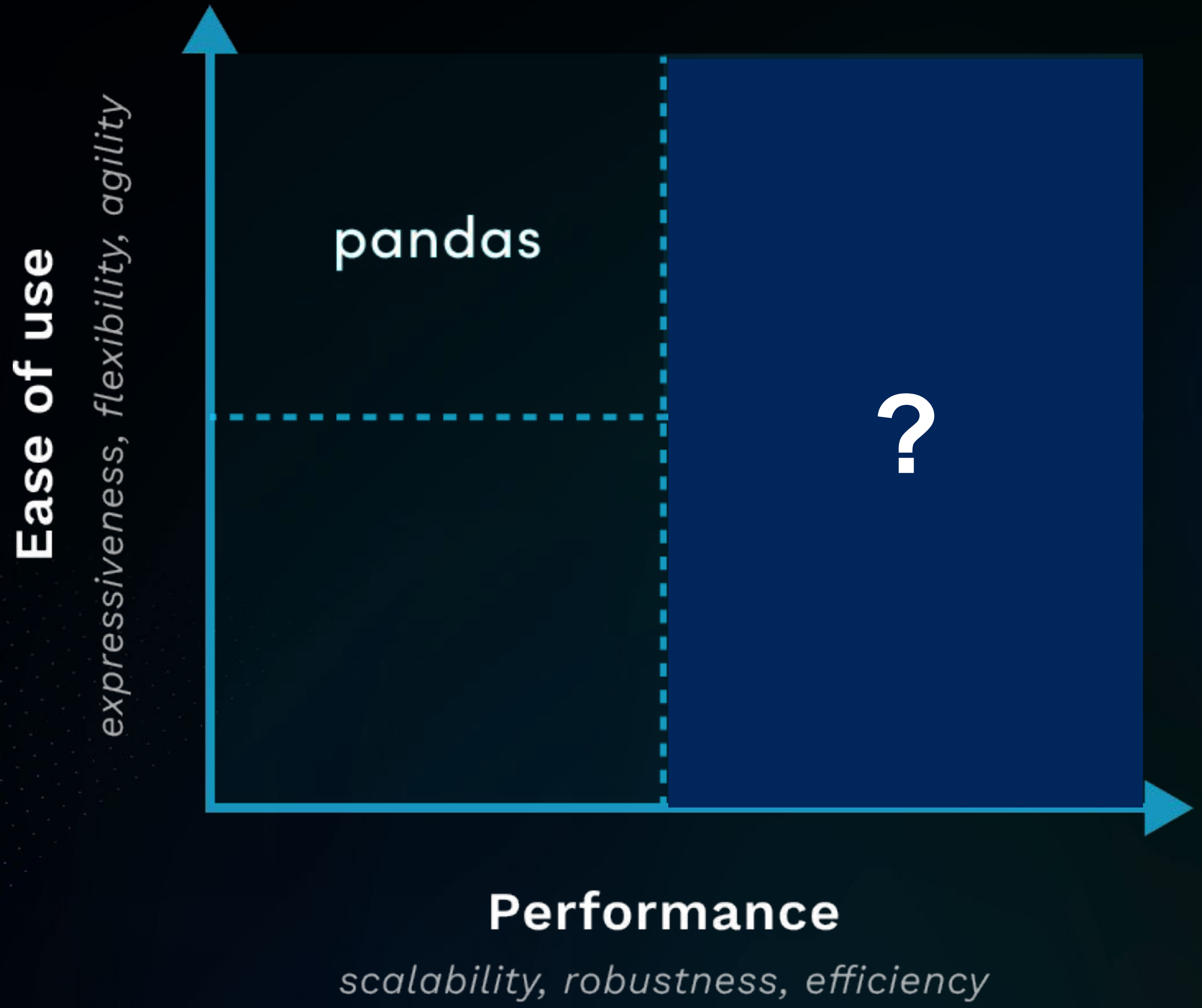
There is no service that can spin up more **Data Scientists**, so we must treat them like the finite resource they are

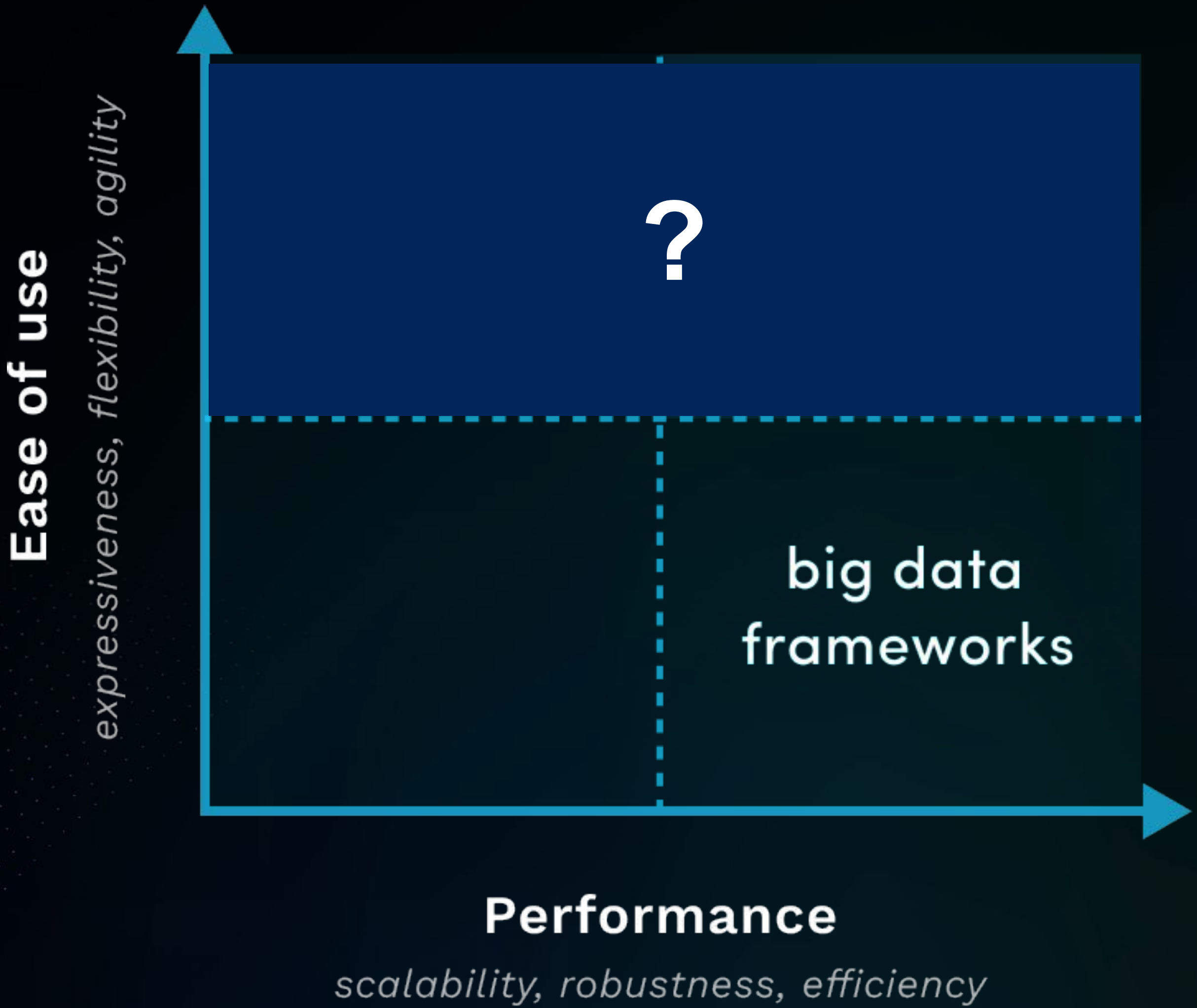
PONDER

Data Science scalability is a human scal

But why?

There is n … can spin up more **Data Scientists**,
so we mu … at them like the finite resource they are


pandas
?
Ease of use
expressiveness, flexibility, agility
Performance
scalability, robustness, efficiency


Ease of use
expressiveness, flexibility, agility
?
big data frameworks
Performance
scalability, robustness, efficiency

Shifting the focus from the machine to the user

PONDER

Tools should work for data scientists

Data Scientists shouldn't have to work for their tools

PONDER

Ponder's work:

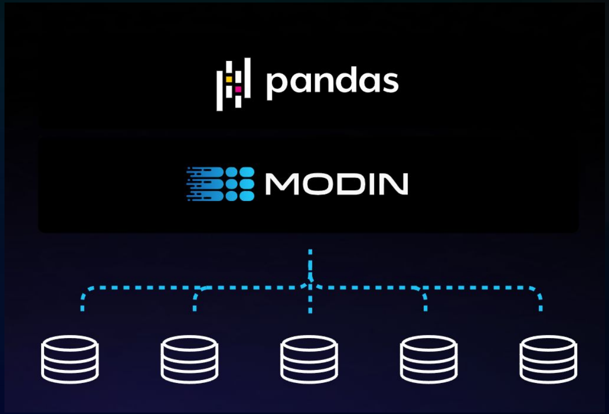
Transparently scale existing tools

Abstract away all of the components of the system that data scientists don't care about, only expose details they do care about.

Ponder's work:

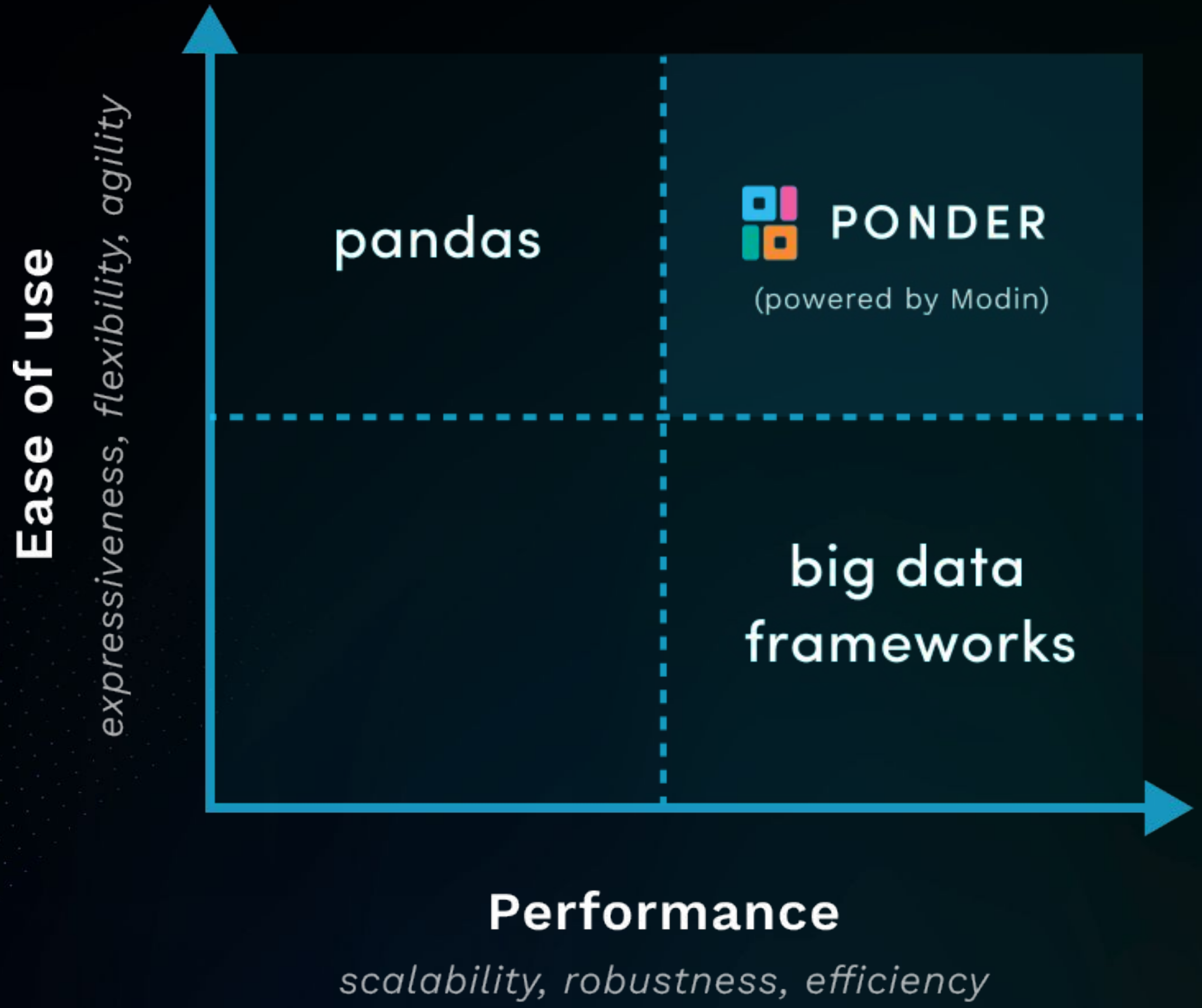
Transparently scale existing tools

pandas

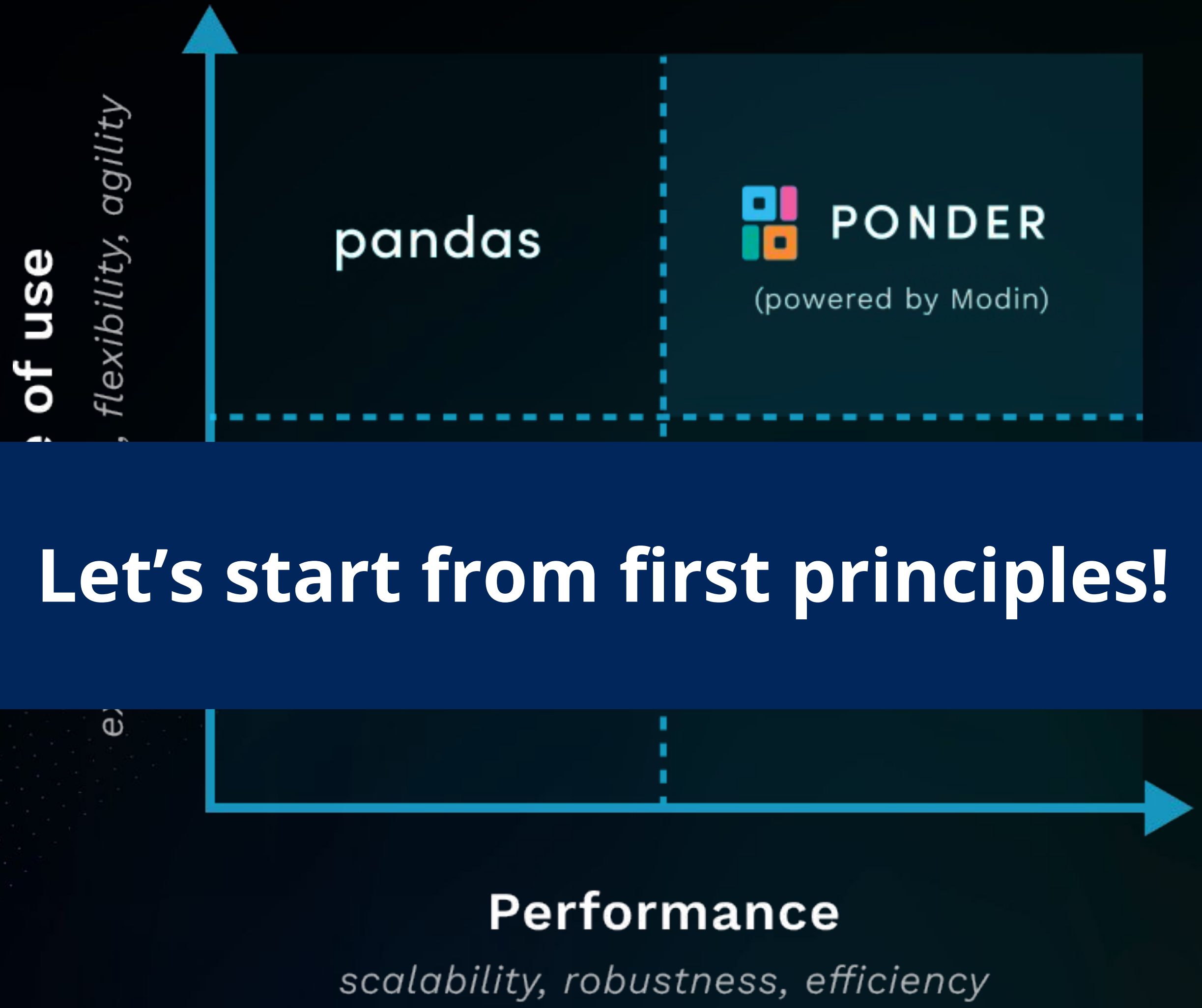


Abstract away all of the components of the system that data scientists don't care about, only expose details they do care about.

PONDER



pandas
PONDER
(powered by Modin)
big data frameworks
Ease of use
expressiveness, flexibility, agility
Performance
scalability, robustness, efficiency



Let’s start from first principles!

But the Pandas API doesn't scale!

The API is simply an expression of what to do

Solving dataframes from first principles

- Dataframe Data Model
- Dataframe Algebra
- Parallelism / Decomposition Rules
- Type System
- Operator Semantics
- Implementation

First Steps: Formalize the Dataframe

PONDER

Towards Scalable Dataframe Systems

Devin Petersohn, William Ma, Doris Lee, Stephen Macke, Doris Xin, Xiangxi Mo
Joseph E. Gonzalez, Anthony D. Joseph, Joseph M. Hellerstein, Aditya Parameswaran
UC Berkeley
{devin.petersohn, williamma, dorislee, smacke, dorx, xmo, jegonzal, adj, hellerstein, adityagp} @berkeley.edu

ABSTRACT

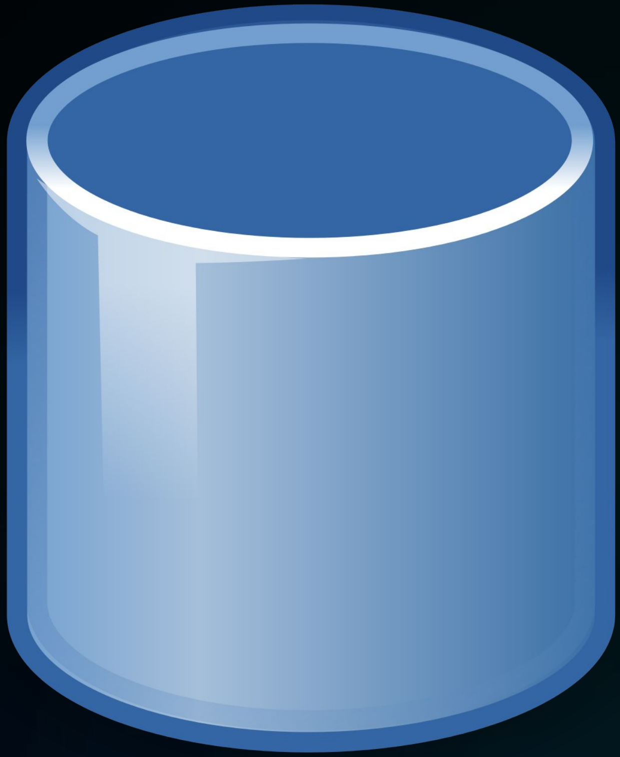
Dataframes are a popular and convenient abstraction to represent, structure, clean, and analyze data during exploratory data analysis. Despite the success of dataframe libraries in R and Python (pandas), dataframes face performance issues even on moderately large datasets. In this vision paper, we take the first steps towards formally defining dataframes, characterizing their properties, and outlining a research agenda towards making dataframes more interactive at scale. We draw on tools and techniques from the database community, and describe ways they may be adapted to serve dataframe systems, as well as the new challenges therein. We also describe our current progress toward a scalable dataframe system, MODIN, which is already up to 30× faster than pandas in preliminary case studies, while enabling unmodified pandas code to run as-is. In its first 18 months, MODIN is already used by over 60 downstream projects, has over 250 forks, and 3,900 stars on GitHub, indicating the pressing need for pursuing this agenda.

1 INTRODUCTION

Characteristics such as these have helped dataframes become incredibly popular for EDA; for instance, the dataframe abstraction provided by pandas within Python (pandas.pydata.org), has, as of 2019, been downloaded over 200 million times, served as a dependency for over 160,000 repositories in GitHub, and starred on GitHub more 22,000 times. Python's own popularity has been attributed to the success of pandas for data exploration and data science [7,9]. Due to its ubiquity, we focus on pandas for concreteness.

Pandas has been developed from the ground up via open-source contributions; pandas DataFrame API operators and their implementations have been provided by dozens of contributors to satisfy immediate or ad-hoc needs, spanning capabilities that mimic relational algebra, linear algebra, and spreadsheet formula computation. To date, the pandas DataFrame API has ballooned to over 200 distinct operators [14]. R, which is both more mature and more carefully curated than pandas, has only 70 operators—but this is still far more than, say, relational and linear algebra combined [15].

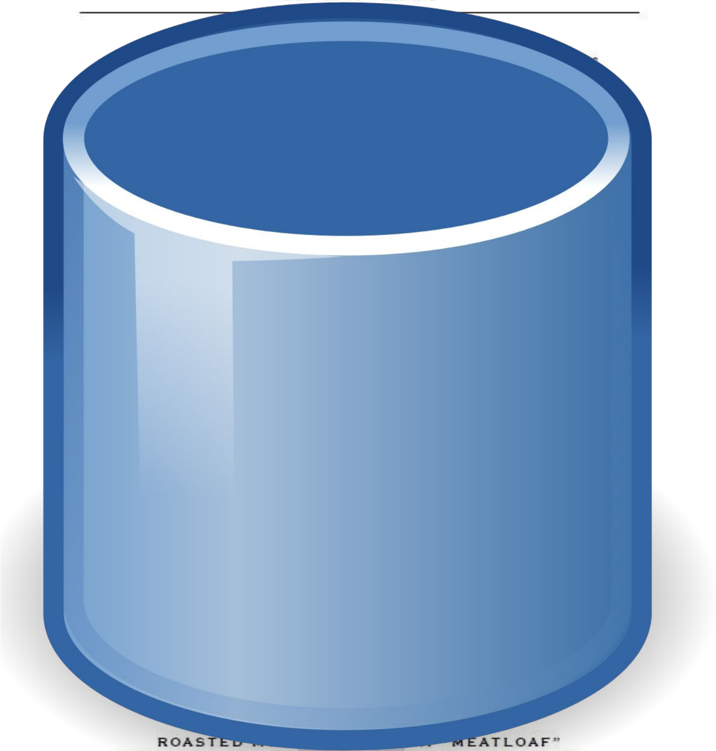
While this rich API is sometimes cited as a reason for pandas' attractiveness, the set of operators has significant redundancies, often with different performance implications. These redundancies



THE FRENCH LAUNDRY

TASTING OF VEGETABLES

28 MAY 2021

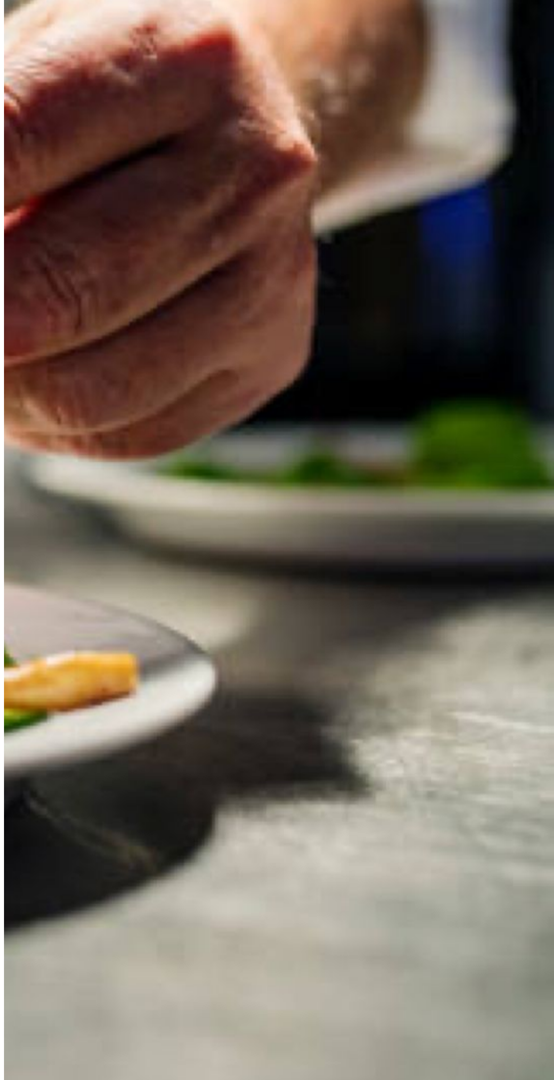


ROASTED … “MEATLOAF”

La Ratte Potato Purée, Crispy Cipollini Onions, Smoked Tomato Glaze, Watercress Leaves and Whole Grain Mustard Gravy

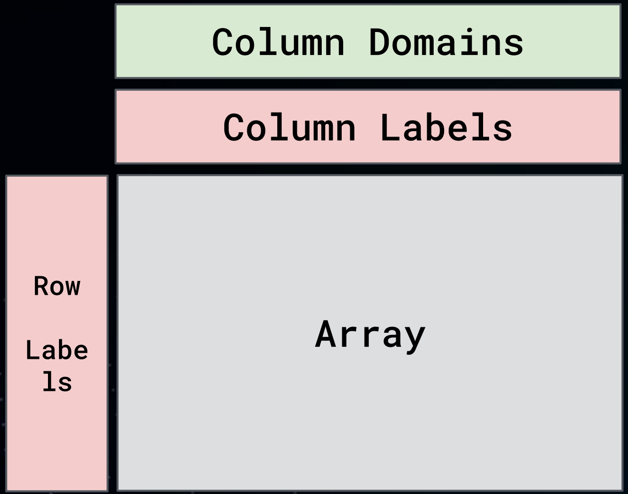
“GOUGÈRE”

Andante Dairy “Etude” and Preserved Australian Black Winter Truffle “Fondue”



pandas

Dataframes: A New Data Model and API



Dataframes: Mixed type-array, w/ row and column labels

`df.groupby(..)`

`df.drop(..)`

`pd.merge(..)`

`df.describe(..)`

`pd.concat(..)`

`df.pivot(..)`

`df.explode(..)`

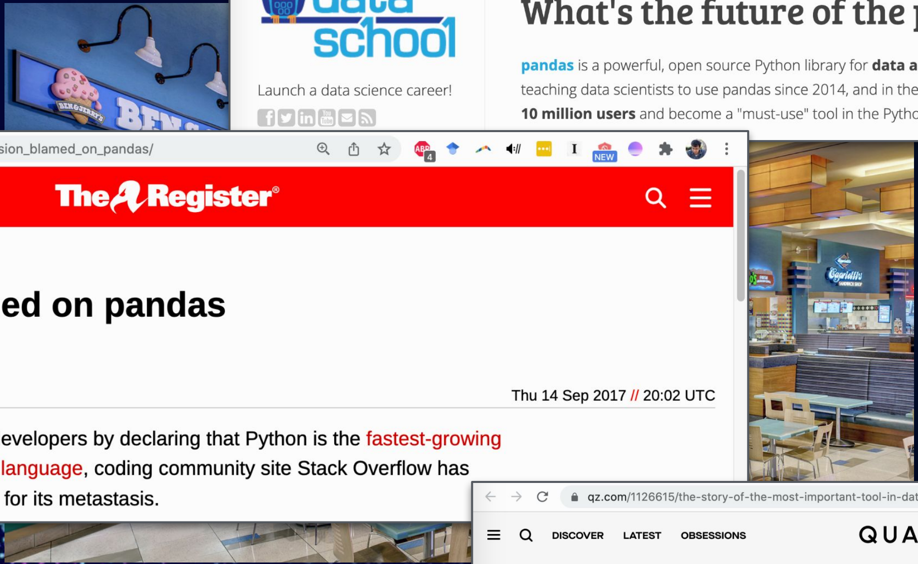
600+ functions to clean, reshape, explore, and summarize data spanning rel., linear, & spreadsheet algebra


BEN&JERRY'S
BEN&JERRY'S
FATBURGER
THE LAST GREAT HAMBURGER STAND
PANDA EXPRESS
13 PROGRESSIVE JACKPOTS!
red rock

PONDER

Convenience	Entire query at once	Incremental + inspection
Flexible	Strict schema	Mixed types, R/C and data/metadata equiv.
Versatility	SFW or bust	600+ functions

Everybody loves pandas!



dataschool.io/future-of-pandas/

data school

Launch a data science career!

December 12, 2018 · PYTHON

What's the future of the pandas library?

pandas is a powerful, open source Python library for **data analysis, manipulation, and visualization**. I've been teaching data scientists to use pandas since 2014, and in the years since, it has grown in popularity to an estimated **5 to 10 million users** and become a "must-use" tool in the Python data science toolkit.

theregister.com/2017/09/14/python_explosion_blamed_on_pandas/

SIGN IN — The Register®

{* DEVOPS *}

Python explosion blamed on pandas

Data science fad just won't die

Thomas Claburn in San Francisco — Thu 14 Sep 2017 // 20:02 UTC

34

Not content to bait developers by declaring that Python is the fastest-growing major programming language, coding community site Stack Overflow has revealed the reason for its metastasis.

qz.com/1126615/the-story-of-the-most-important-tool-in-data-science/

DISCOVER LATEST OBSESSIONS — QUARTZ — FEATURED EMAILS BECOME A MEMBER

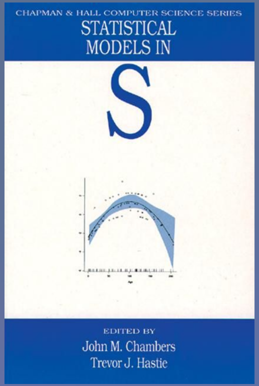
PANDAS

Meet the man behind the most important tool in data science

Inherited Data Model

PONDER

Small group development



1990

~70 operators

R

1995
2000 (stable)

Large community development

>200 operators

pandas

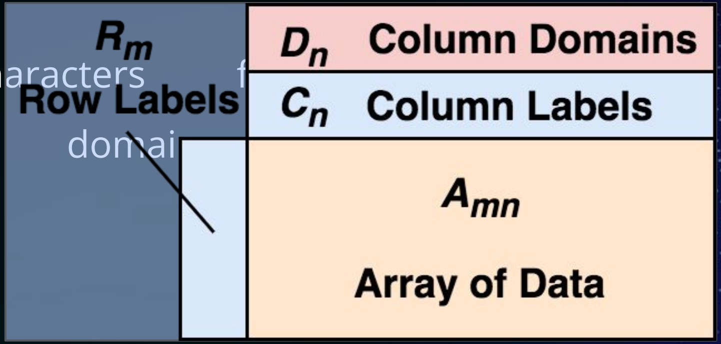
2008

Dataframe formal definition (VLDB 2020)

Let Σ^* be the finite set of characters f

Let *Dom* be a finite set of domai

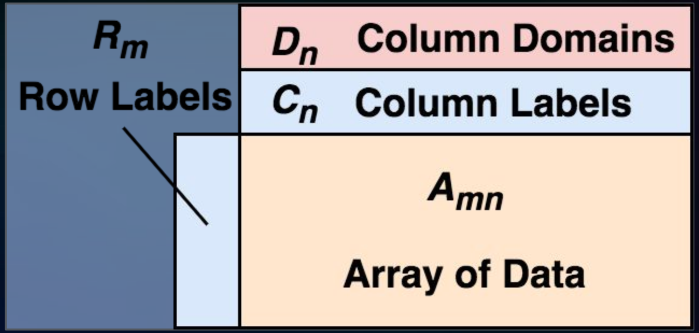
Let each $dom_i \in Dom$ have a mapping p_i: $\Sigma^* \rightarrow dom_i$.



A **dataframe** is a tuple (A_{mn}, R_m, C_n, D_n), where A_{mn} is an arrangement of entries in columns and rows from the domain Σ^*, R_m is a vector of row labels from Σ^*, C_n is a vector of column labels from Σ^*, and D_n is a vector of *n* domains from some finite set of domains *Dom*, one per column, each of which can also be left unspecified. We call D_n the schema of the dataframe. If any of the *n* entries within D_n is left unspecified, then that domain can be induced by applying a schema induction function $S(\cdot)$ to the corresponding column of A_{mn}. The schema induction function S: $\Sigma^* \rightarrow Dom$, assigns an arrangement of *m* strings to a domain in *Dom*.

Dataframe data model

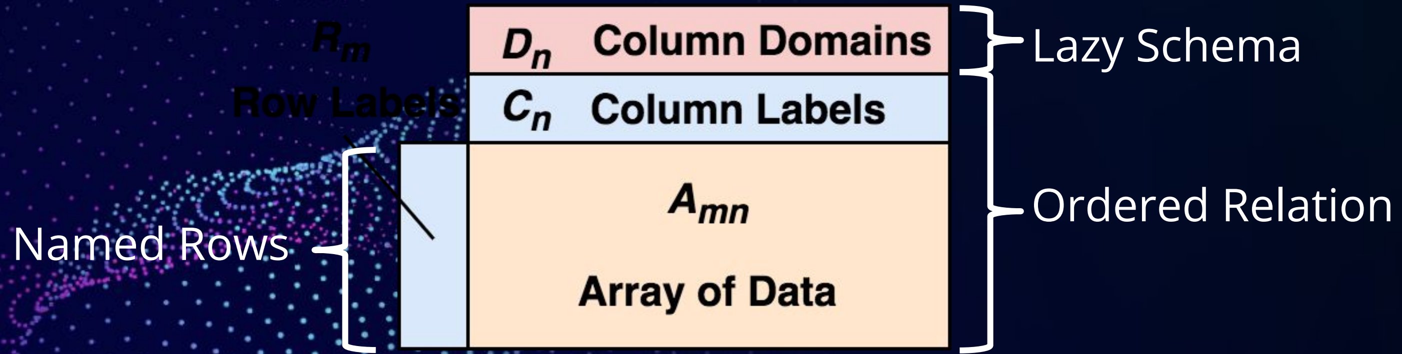
- Ordered, but not necessarily sorted
  - Rows **and** columns
- No predefined schema necessary
  - Types can be induced at runtime
- Typed Row/column labels
  - Labels can become data
- Indexing by label or by row/column numeric index
  - “Named notation” or “Positional notation”



Dataframes from two perspectives

From a **relational algebra** perspective, dataframes contain:

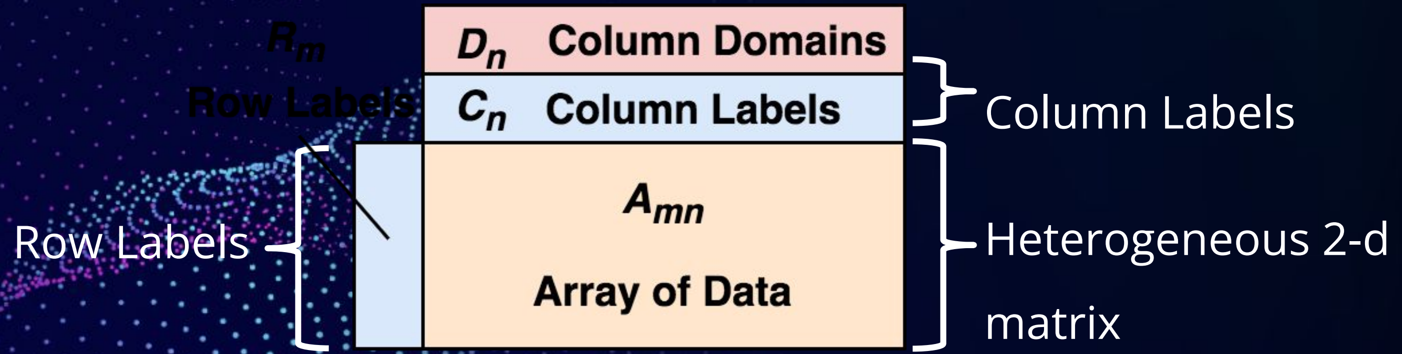
- An ordered table
- Named rows of arbitrary type
- A lazily-induced schema
- Column names of arbitrary type
- Column and row symmetry
- Support for linear algebra operators (e.g. matrix multiply)



Dataframes from two perspectives

From a **linear algebra** perspective:

- Heterogeneous matrix-like data structure
- Both numeric and non-numeric types
- Explicit row and column labels
- Indexing by label in addition to position
- Support for relational algebra operators (e.g. join)



- Dataframe Data Model ✔
- Dataframe Algebra
- Parallelism / Decomposition Rules
- Type System
- Operator Semantics
- Implementation

Next: What can a dataframe do?

First Principles next steps:

define an algebra

What can Pandas Do?

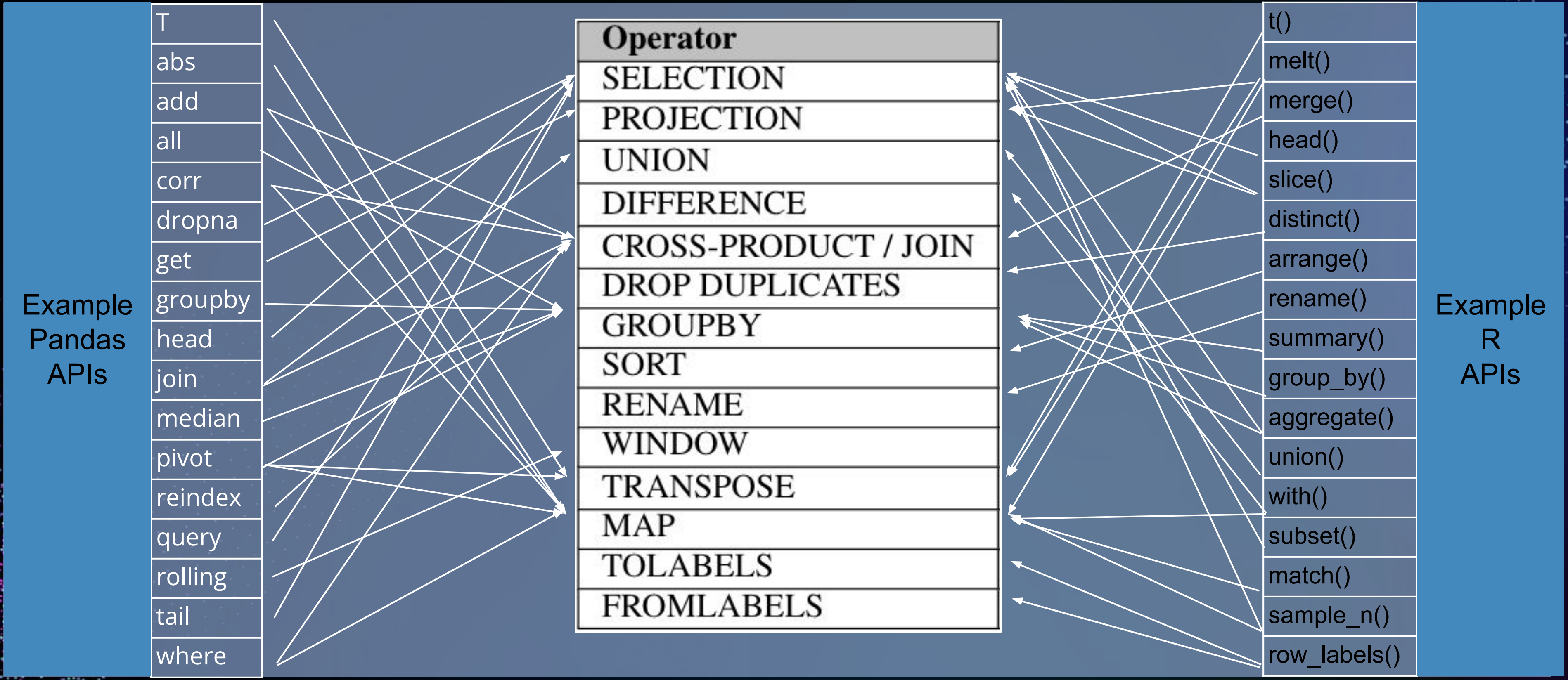
PONDER

pandas

pd.DataFrame	280+ methods
pd.Series	280+ methods
Convenience methods (e.g. concat)	40+ APIs

Total: 600+ operators in pandas

PONDER


Example Pandas APIs
T
abs
add
all
corr
dropna
get
groupby
head
join
median
pivot
reindex
query
rolling
tail
where
Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS
t()
melt()
merge()
head()
slice()
distinct()
arrange()
rename()
summary()
group_by()
aggregate()
union()
with()
subset()
match()
sample_n()
row_labels()
Example R APIs

Proof by exhaustion that all pandas APIs are covered

Operator Pattern	Modin Syntax	Exhaustive list of pandas public API	Count
Applying a user-defined function uniformly element-wise, column-wise, or row-wise	map(df, axis, f)	abs, astype, clip, eval, isna, isnull, notna, notnull, fillna, isin, apply, applymap, transform, fillna, rank, round, unary version of {add, subtract, etc.}, string manipulations (str), datetime manipulation (dt), replace	42
Binary function between two dataframes	map(join(df1, df2), f)	add, subtract, multiply, divide, eq, floordiv, where, update, corr, corrwith, combine, combine_first, cov, dot, ge, gt, lt, pow, radd, rdiv, rfloordiv, rmod, rmul, rpow, rsub, rtruediv, sub, truediv	28
Indexing: Queries on the row labels	mask(df, row_indices=query)	as_freq, asof, at, at_time, between_time, drop_level, drop, first, first_valid_index, iloc, last, last_valid_index, loc, mask, pop, get, head, tail, iat, lookup, resample, sample, select, take, truncate	26
Treating metadata as data, metadata manipulation, and querying	to_labels(map(from_labels(df), f), label)	add_prefix, add_suffix, swap_level, melt, reorder_levels, shift, slice_shift, tshift, tz_convert, tz_localize	10
Reshaping, transposing, pivot	[map/groupby](transpose(df), f)	pivot, pivot_table, stack, unstack, transpose, T	6
One-hot (dummy) encoding	transpose(to_labels(transpose(map(df, f, axis=1))))	get_dummies	1
User-defined aggregation of values per-column	reduction(df, f)	all, any, count, agg, aggregate, compound, describe, duplicated, equals, idxmax, idxmin, kurtosis, mad, max, mean, median, min, mode, memory_usage, prod/product, nunique, quantile, sem, skew, std, sum, var	28
Aligning and joining two dataframes on row or column labels	join(from_labels(df1), from_labels(df2), on="index", axis)	align, concat, join, reindex, reindex_axis, reindex_like	6
Window user-defined functions (window size < length of dataframe)	window(df, axis, f, size)	bfill, cumsum, cumprod, cummax, cummin, diff, ewm, expanding, ffill, fillna, interpolate, nLargest, nsmallest, pct_change, rolling	15
Conditional filter	filter(df, f)	dropna, drop_duplicates, filter, query	4
Queries on the schema of the dataframe	filter_by_types(df, types)	select_dtypes	1
Type Inference and induction	infer_types(df, columns)	infer_objects, convert_objects	2
Column/row insertion and assignment, appending columns/rows	concat(df1, [df2])	append, assign, concat, insert	4
Groupby with a user-defined aggregation or function	groupby(df, by, f)	groupby	
Join on an attribute	join(df1, df2, condition, how)	merge, merge_asof	2
Sorting on labels or column values	sort_by(df)	sort_index, sort_values	2
Expand the number of rows or columns	explode(df, f)	explode	1

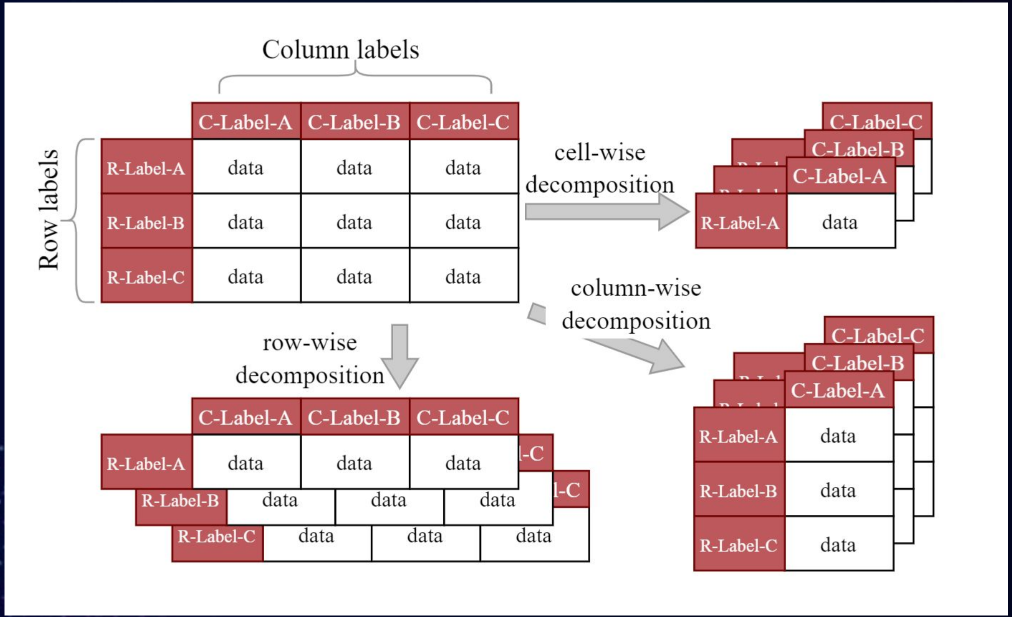
- Dataframe Data Model ✔
- Dataframe Algebra ✔
- Parallelism / Decomposition Rules
- Type System
- Operator Semantics
- Implementation

Decomposition Rules -> Formalize parallelism

Cell wise: An operator can be applied to a “unit dataframe” independently

Row-wise: An operator can be applied to each row independently

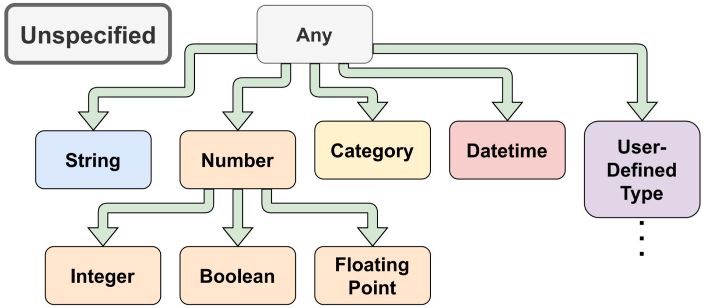
Column-wise: An operator can be applied to each column independently



- Dataframe Data Model ✔
- Dataframe Algebra ✔
- Parallelism / Decomposition Rules ✔
- Type System
- Operator Semantics
- Implementation

Formalization of other components of the dataframe

Type System



Order Semantics

Operator	Input Order	Position	Output Order & Position
mask	N	Y*	Parameter-Dependent
filter_by_types	N	N	Inherited | Updated
map	N	Y⋄	Inherited from Inputs
filter	N	Y⋄	Inherited | Updated
explode	N	Y⋄	Inherited | Updated
reduce	N	Y⋄	Inherited
window	Y	N	Inherited
groupby	N	N	Data-dependent
infer_types	N	N	Inherited
join	N	N	Inherited*
concat	N	N	Inherited*
transpose	N	N	Inherited
to_labels	N	N	Inherited
from_labels	N	Y	Inherited
sort	N	N	Data-dependent
rename	N	N	Inherited

- Dataframe Data Model ✓
- Dataframe Algebra ✓
- Parallelism / Decomposition Rules ✓
- Type System ✓
- Operator Semantics ✓
- Implementation

So now we know:

- What a dataframe is (formally)
- What operators a dataframe supports
- How these operators map back to pandas
- How to handle dataframe types
- How to decouple logical and physical order
- How to maximally parallelize each operator

MODIN

A dataframe built from first principles

Modin

Accelerate your pandas workloads by changing one line of code

PONDER

To use Modin:

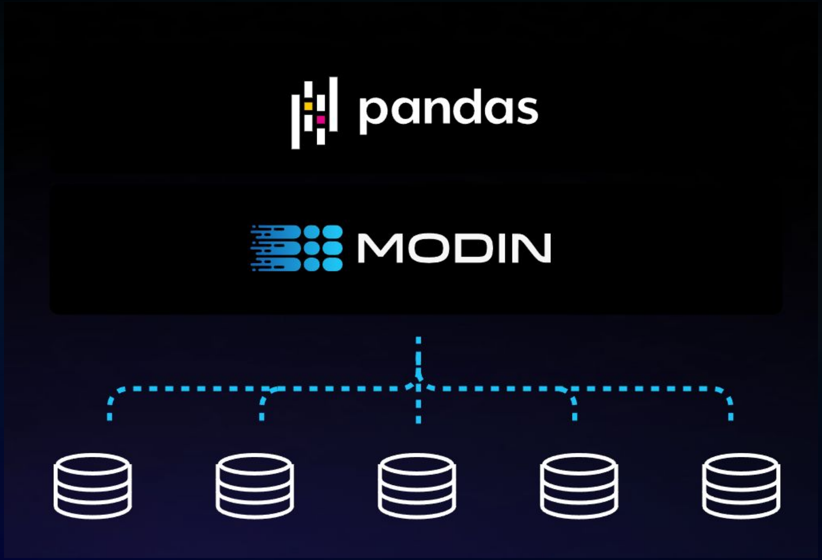
```
import pandas as pd
```

To install Modin:

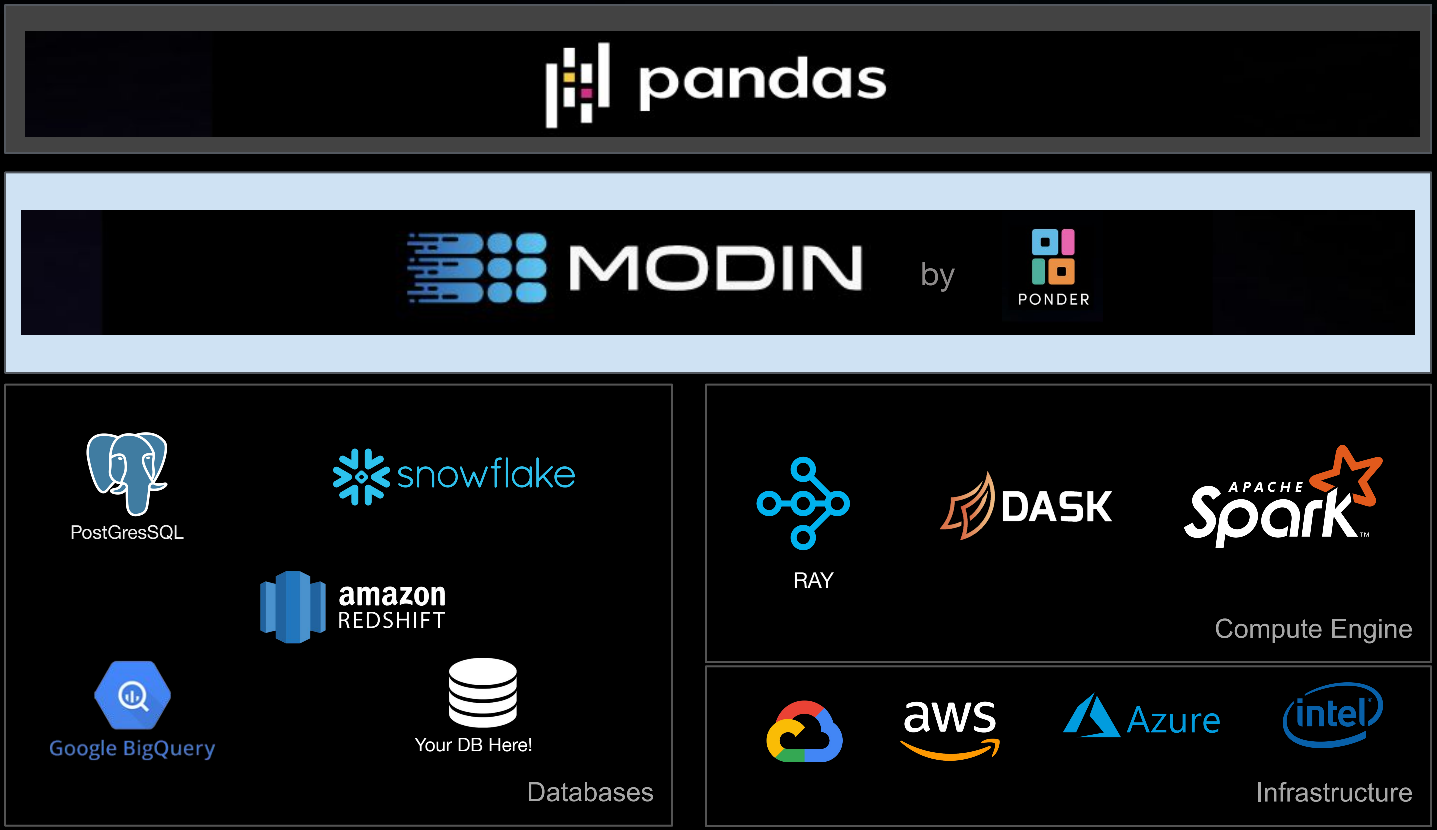
```
pip install modin
```

What can we do with the formalism?

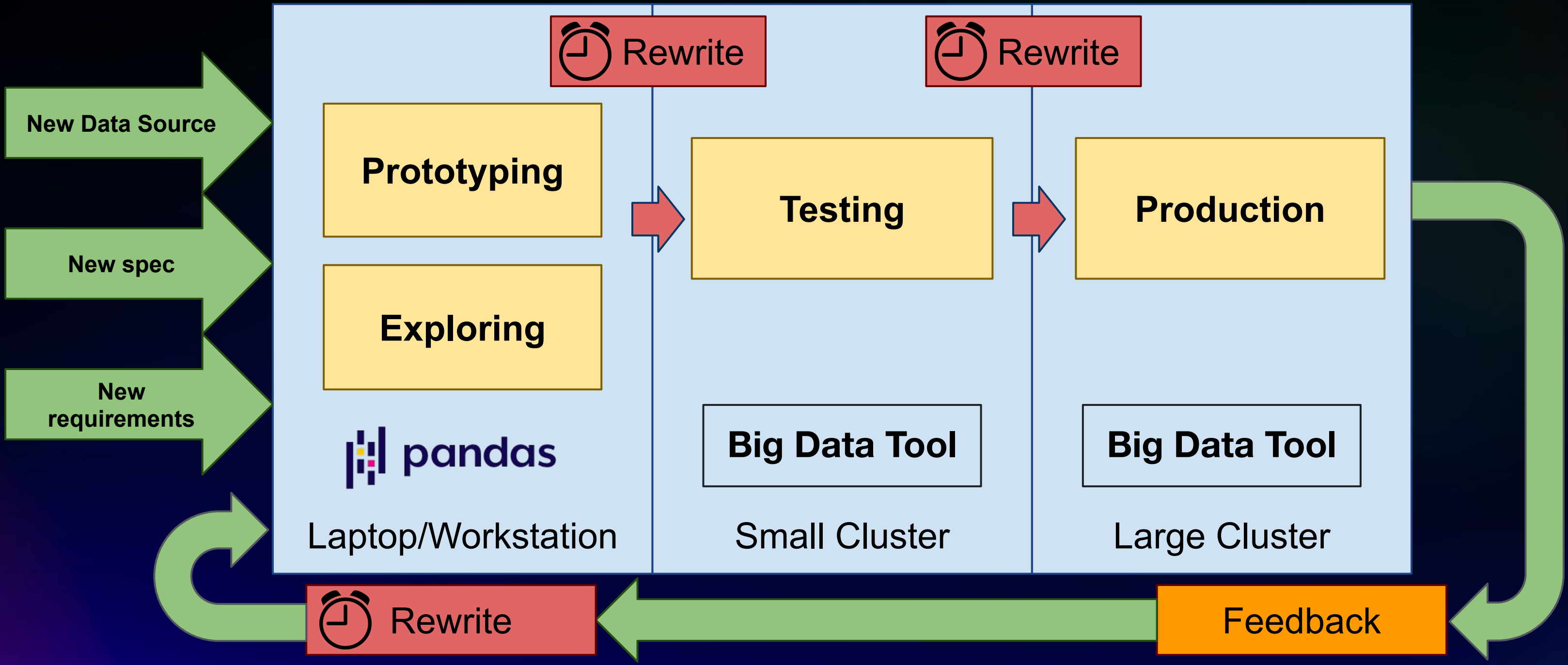
- Data model
  - Expose the “feel” of pandas without the baggage
- Dataframe Algebra
  - Smaller surface to implement
  - Map operators to other systems



Pandas on everything with Ponder

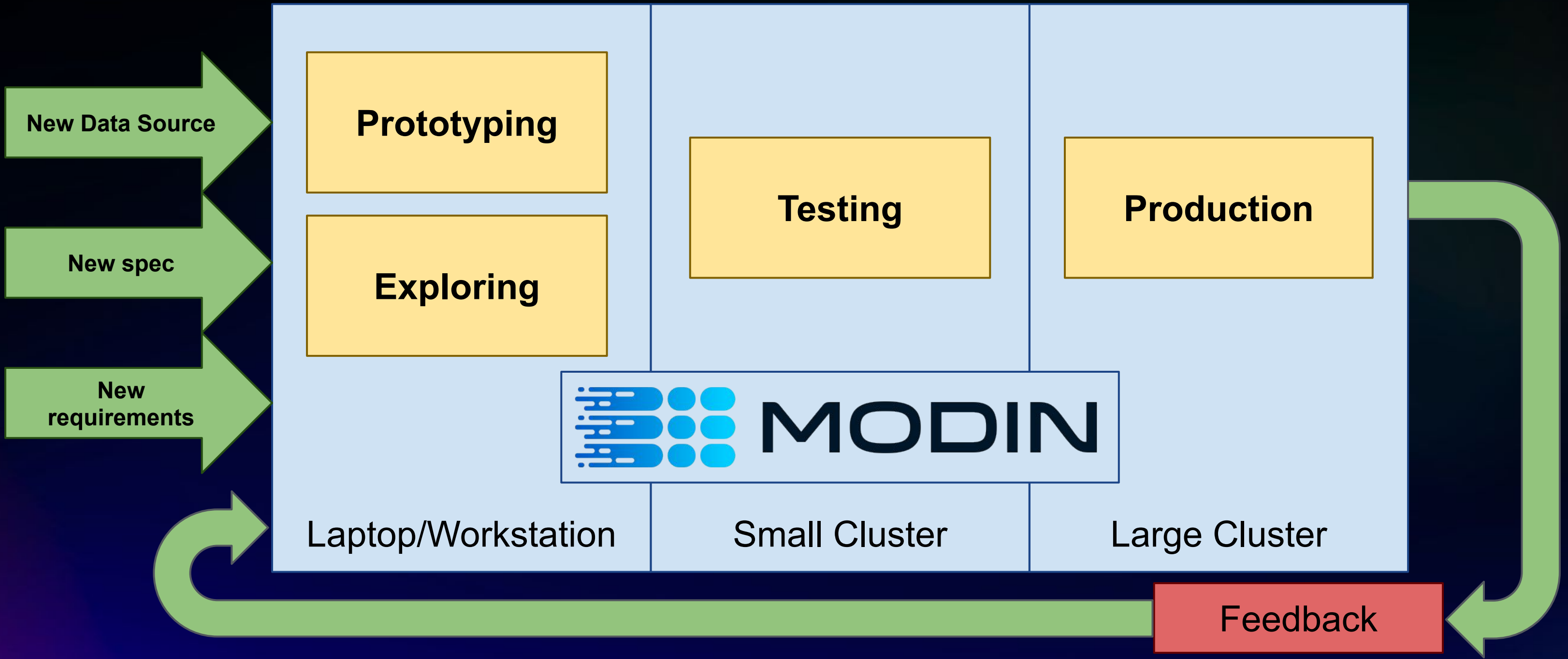


Many organizations look like this



How Modin is being used in practice:

PONDER



PONDER

Modin Open Source Impact

Used by 10% Fortune 100 companies & more!

BOSCH
vmware
Bristol Myers Squibb
lululemon
LogMeIn
intel
cisco
gsk
TESLA
U.S. Department of Defense
Ford
Allianz
VOXMEDIA
NOKIA
ATLASSIAN
UK FLOORING DIRECT
Azure
BERKELEY LAB
kevala

2.6M+
Downloads to date

7k
Github stars

90+
Open Source Contributors

E-Commerce Case Study

PONDER

Current State

Pricing + Recommendation
Pipelines in Production

150M Events
Streamed Daily

50k Records
Pandas Limit

With Ponder

Modin

1000X
More Data with Modin

< 20 Sec.
Faster with Modin

Finance Case Study

PONDER

Current State

Regulatory Reporting
Pipelines Migration

10k+ Lines
Of Code to change est.

2k+ Hours
Human time to rewrite est.

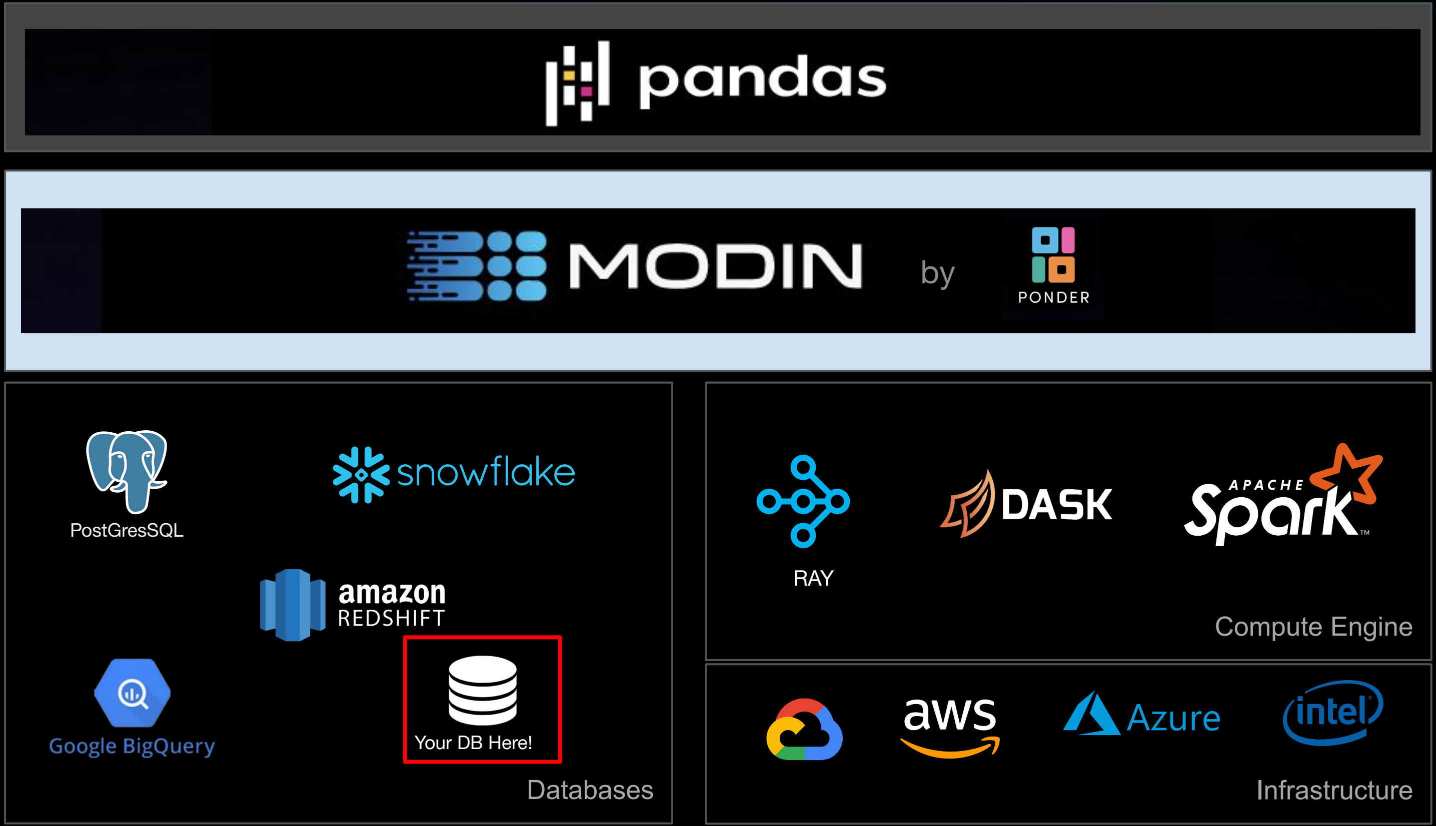
With Ponder

Modin

100X
Reduction in code reduction w/ Modin

100X
Reduction in human time with Modin

Bring your database!



In summary...

- Formal dataframe data model & algebra
  - Dataframes are newly defined structures with a lot of open problems!
  - Dataframe algebra can express all of pandas
- Reference implementation: Modin
  - High impact -> problems are real and pressing need
- Deep technical problems still exist!
  - It is an exciting time to be working on dataframes

Thank you!

Devin Petersohn
devin@ponder.io

Backup

Current state of affairs

Current state of affairs

PONDER

- Coverage of pandas API
    - pandas.DataFrame - 83% (93% based on usage)
    - pandas.Series - 77% (86% based on usage)
    - pandas.read_* - 42% (>90% based on usage)
-

API Vision - 5 years

- More complete pandas API - 95%
- All common interactive data processing modalities
  - Spreadsheet API
  - SQL API
  - New Modin API
- Hooks for SQL systems to implement parts of the API
  - A partial pandas API for relational systems
- Preliminary numpy API - more implementation proof that dataframes can act as matrices
  - Plug in to sklearn

Engineering Vision - 5 years

PONDER

- Query planning/optimization
  - Optimize for user’s time
  - Some research involved here (more later)
  - Extending Calcite?
- GPU integration
- MPI, other compute engines
- Serverless

Case study: Pivot

PONDER

transpose + groupby + map

Narrow Table (SALES)

Year	Month	Sales
2001	Jan	100
2001	Feb	110
2001	Mar	120
2002	Jan	150
2002	Feb	200
2002	Mar	250
2003	Jan	300
2003	Feb	310

Wide Table of MONTHS

Month	2001	2002	2003
Jan	100	150	300
Feb	110	200	310
Mar	120	250	NULL

Pivot ⟶

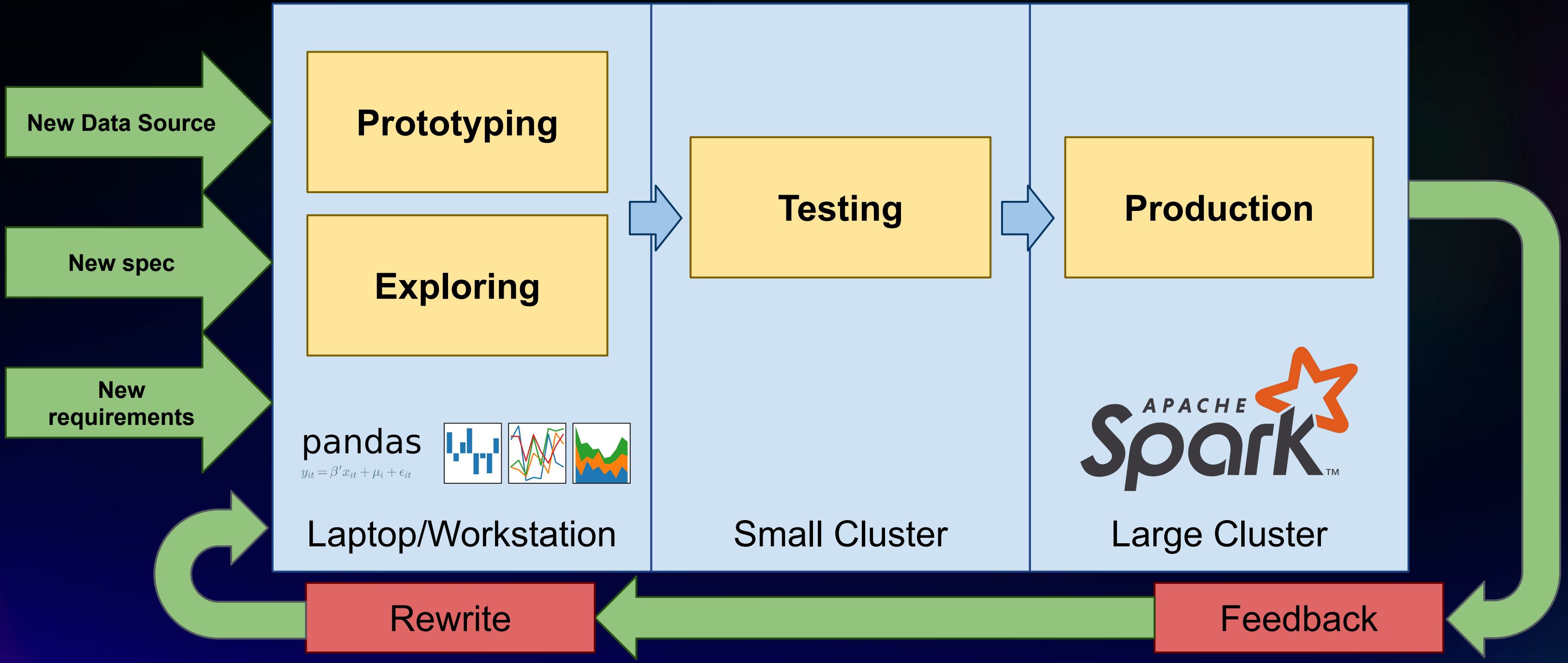
⟵ Unpivot

Year	Jan	Feb	Mar
2001	100	110	120
2002	150	200	250
2003	300	310	NULL

Wide Table of YEARS

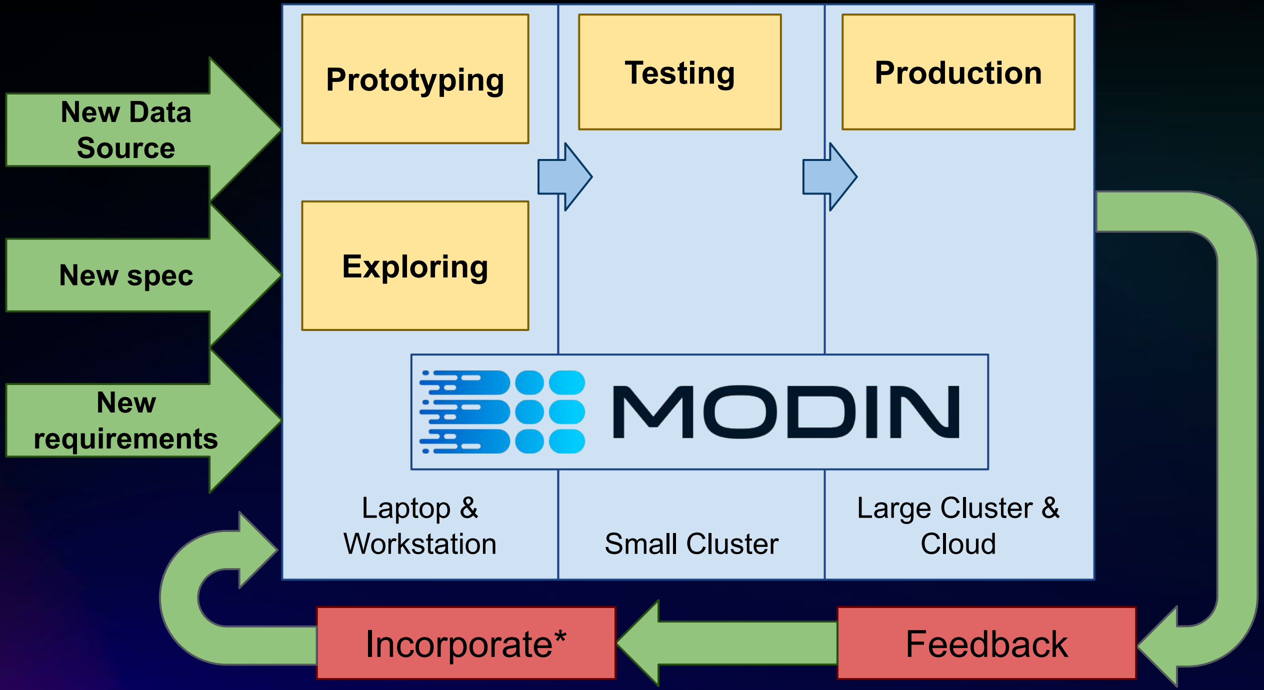
Transpose + map

Current Machine Learning Lifecycle



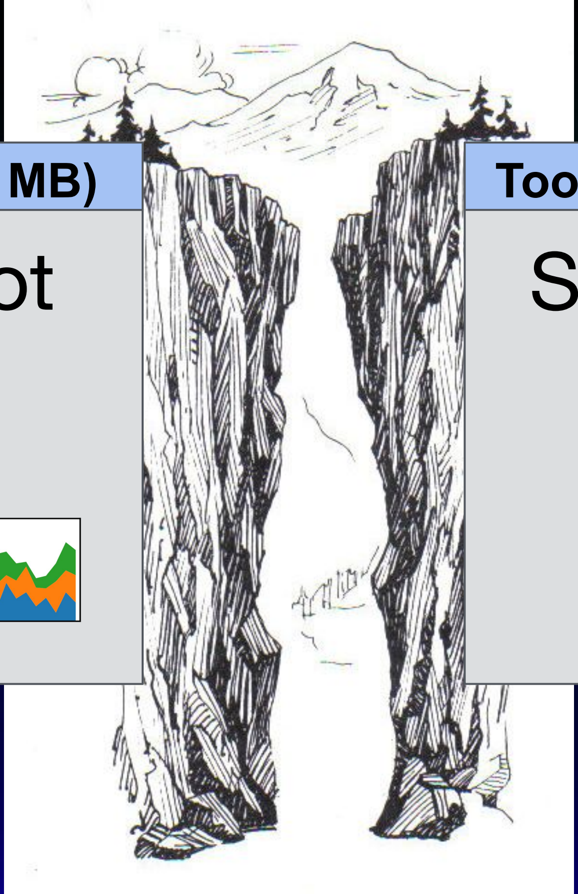
One API, all scales (think SQL)

PONDER



Data Science Landscape: Today

PONDER

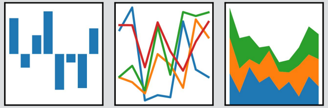


Tools efficient for O(1MB)

Usable but not scalable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

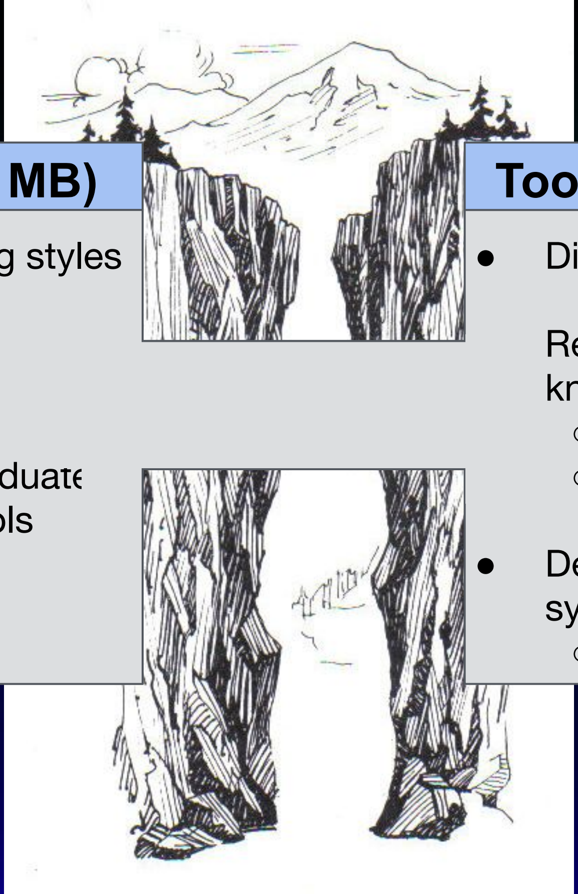


Tools efficient for O(100s GB+)

Scalable but not usable

APACHE Spark™

Data Science Landscape: Today



Tools efficient for O(1MB)

- Follow typical programming styles
- Tools are widely used and understood - in production
- The majority of college graduate will already know these tools
- No scalability

Tools efficient for O(100s GB+)

- Difficult to debug
- Requ uted g knowled
  - Must artitioning
  - Lazy ated
- Desig ter for system le
  - New APIs that do t e same thing

Data Science Landscape: Today

PONDER

Tools efficient for O(1MB)

Usable and scalable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

pandas++

Tools efficient for O(100s GB+)

Scalable and usable

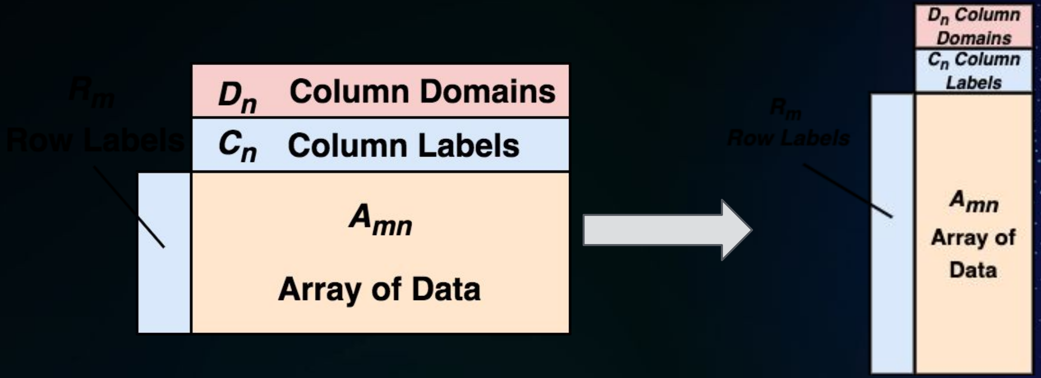
pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

PONDER

Dataframe Algebra

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS



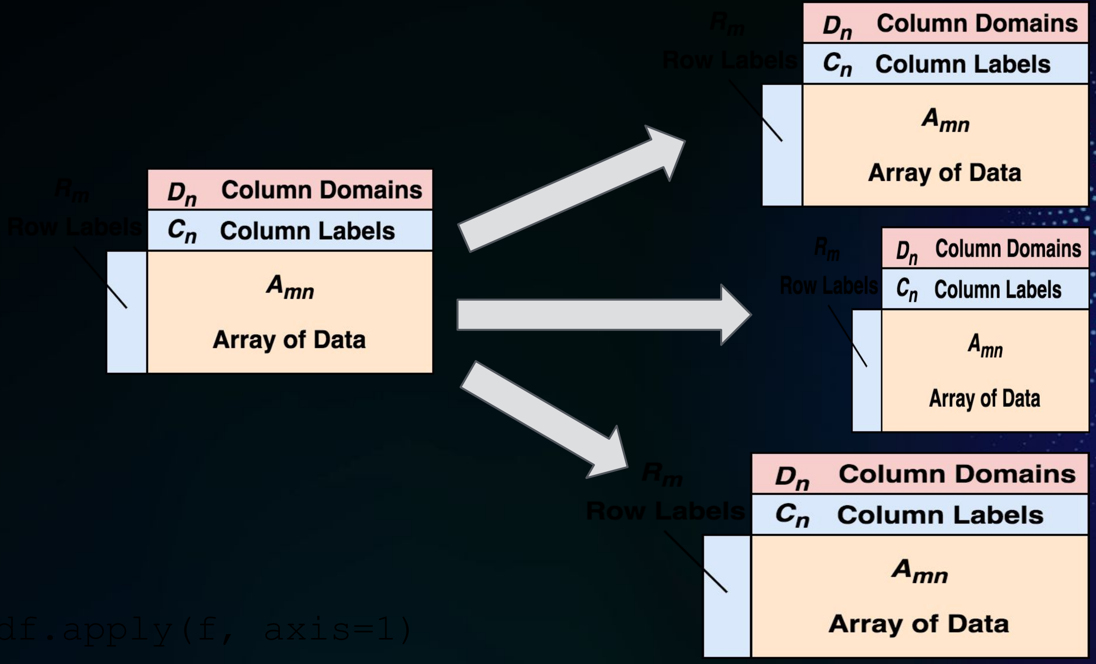
Columns become rows/rows become columns

```
df.T
```

Dataframe Algebra

PONDER

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS



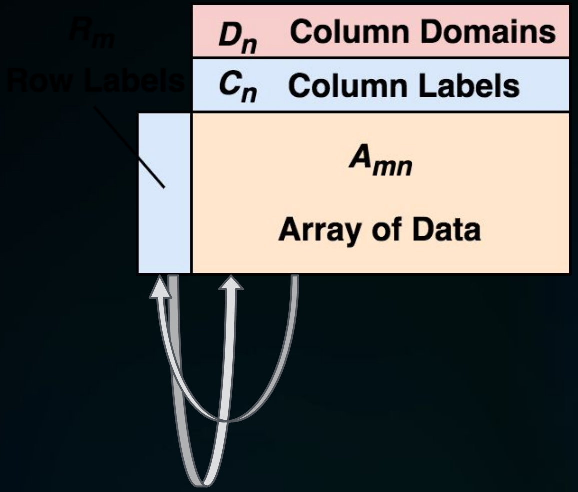

```
df.apply(f, axis=1)
```

User defined function across all rows

Dataframe Algebra

PONDER

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS




```
df.set_index(df["col"])
df.reset_index(drop=False)
```

Elevating data into metadata, or moving metadata into the data

Working with tabular data

A (very) brief history

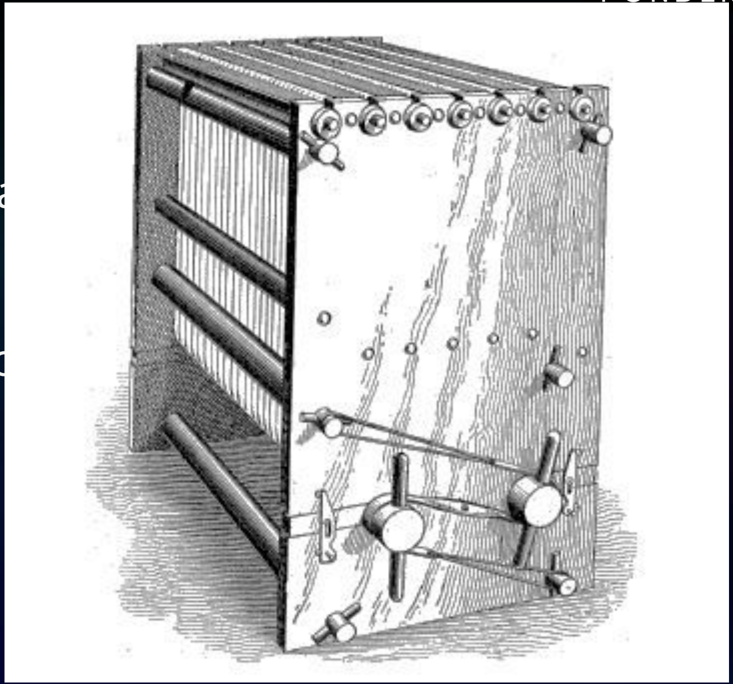
Relational Databases

PONDER

- Invented in the 1970's by Edgar F. Codd
  - Defined a data model for **structured data**
  - Schema must be known before input into DBMS
  - Decoupling of the physical representation from logical
- Popularized in the 1980's
  - Data model has stood the test of time, still in use!
  - SQL

Census - the original “Big Data”

- 1880’s - Seaton Device, manual intervention a
  10 years
- 1890’s - Herman Hollerith’s machine, punch c
- 1940’s - First use of electronic computers



Dataframes

PONDER

- Emerged from a real-world need at the time
  - No way of handling unstructured or semi-structured data
  - Matrices and Tables did not fit their need, something new needed
- Not formalized!
- Dataframes have an origin in S

Focus: pandas

pandas success

PONDER

- 23 million installs/month
- Over 300 million total installs
- Used by over 209k projects in GitHub
- 24.6k GitHub stars

Pandas API

PONDER

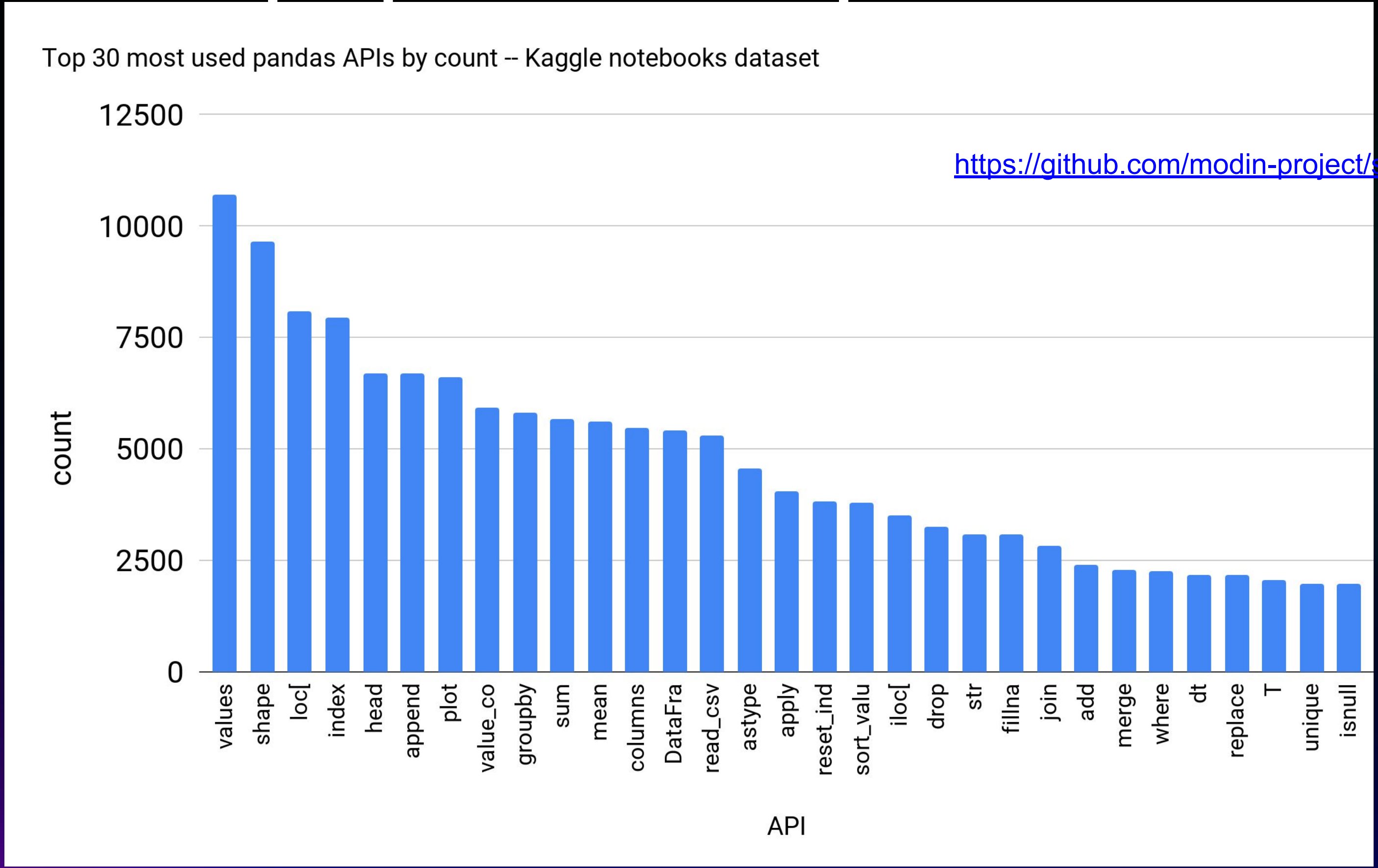
pd.DataFrame	280+ methods
pd.Series	280+ methods
Convenience methods (e.g. concat)	40+ APIs

pandas API is huge and expressive

pd.DataFrame	280+ methods
pd.Series	280+ methods
Convenience methods (e.g. concat)	40+ APIs

PONDER

What do people use within pandas API?



https://github.com/modin-project/study_kaggle_usage

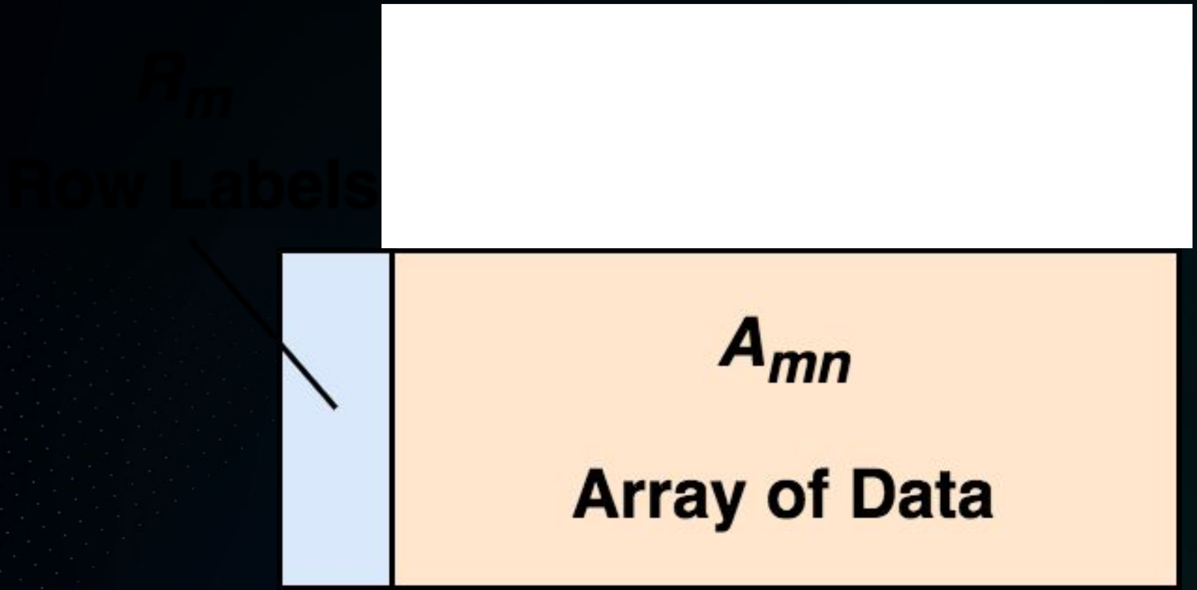
PONDER

Definition: Dataframes

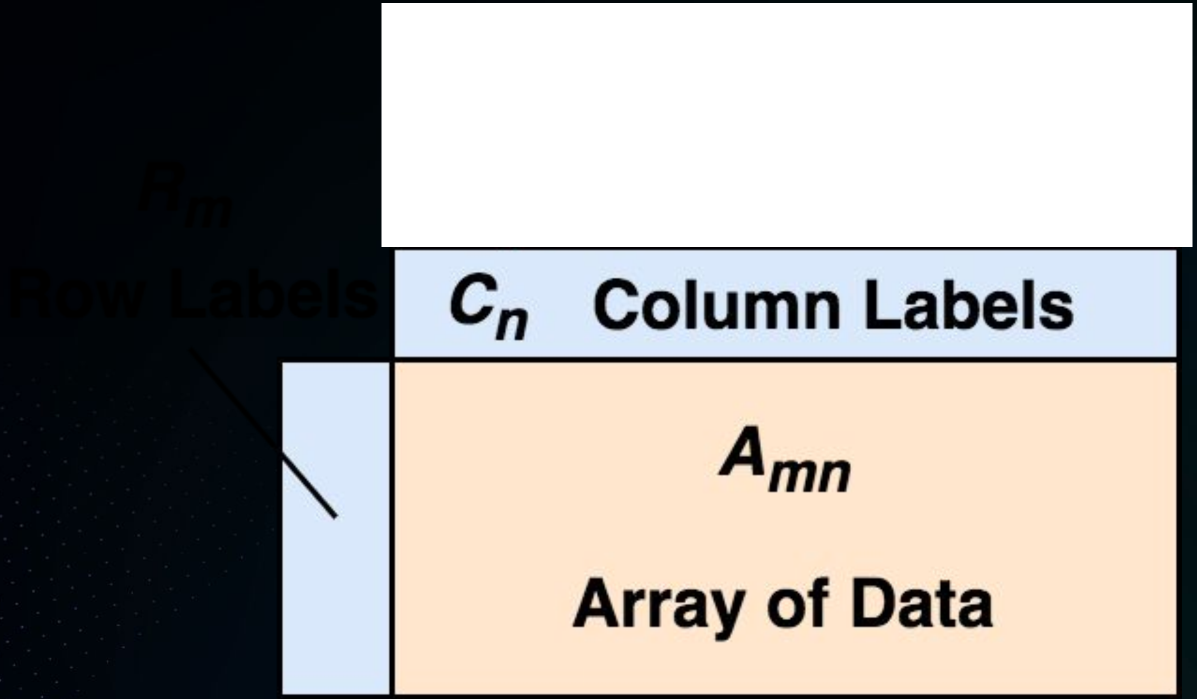
A_{mn}

Array of Data

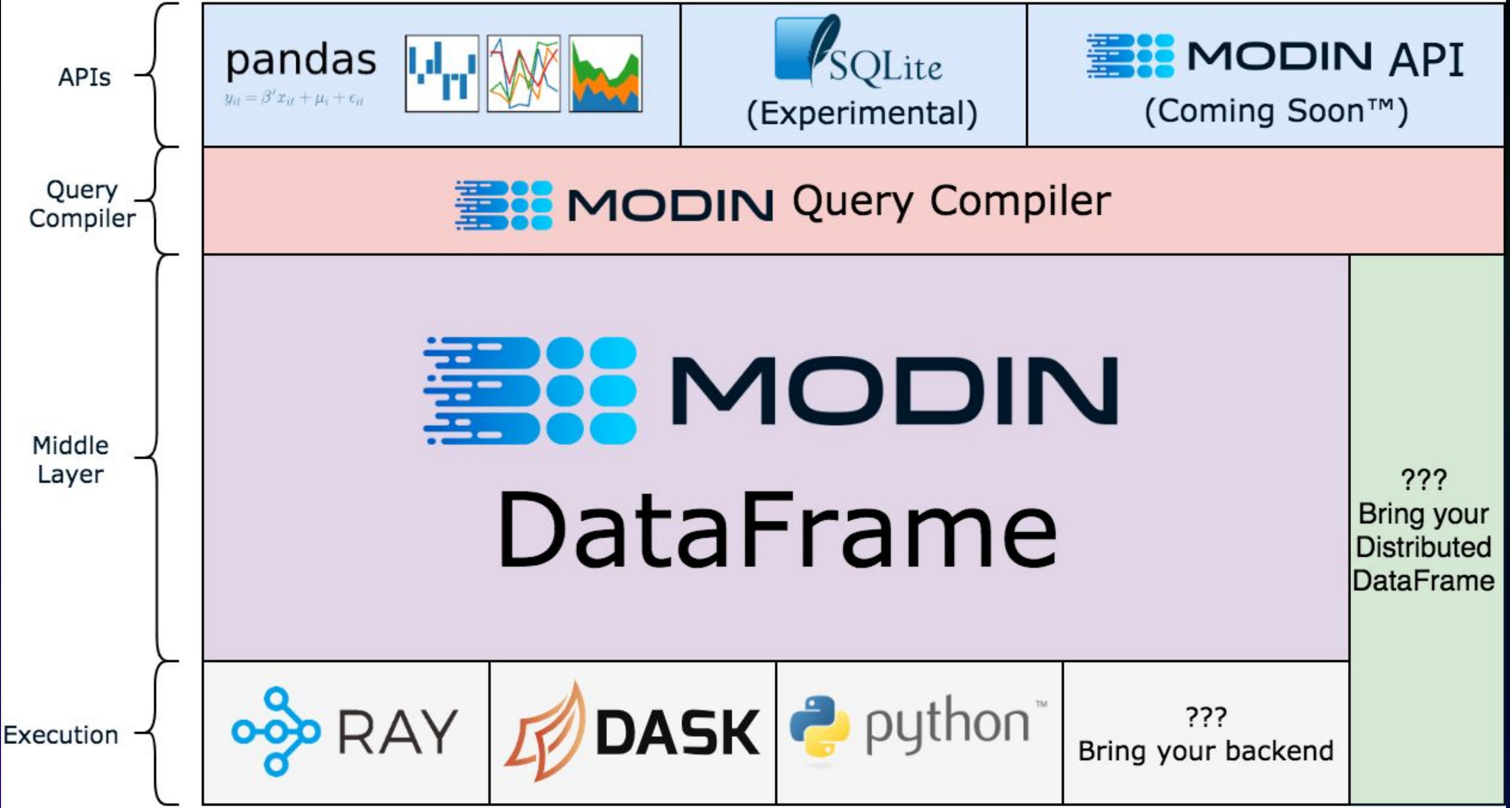
Definition: Dataframes



Definition: Dataframes

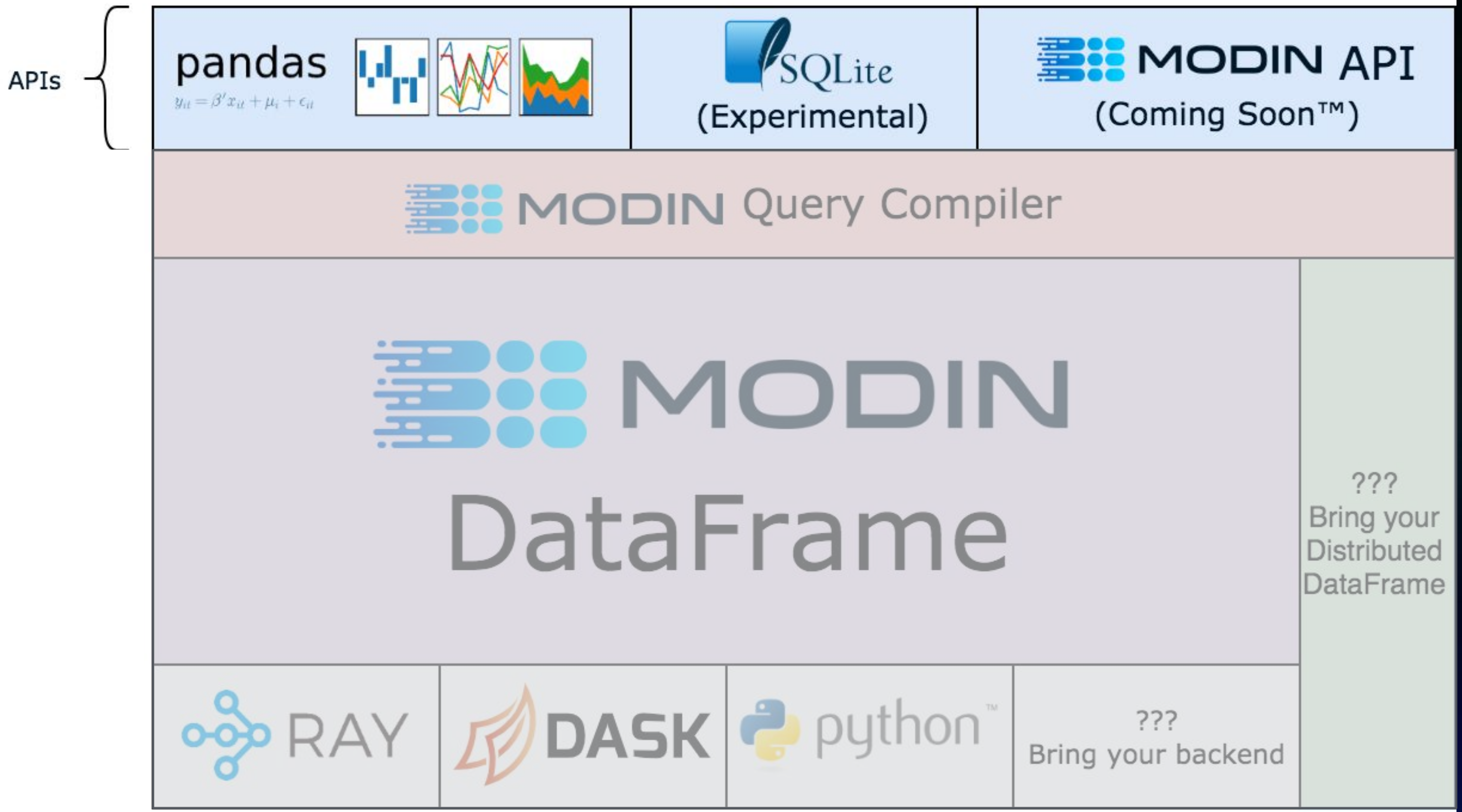


Mo


APIs
Query Compiler
Middle Layer
Execution
pandas
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python™
???
Bring your backend

PONDER

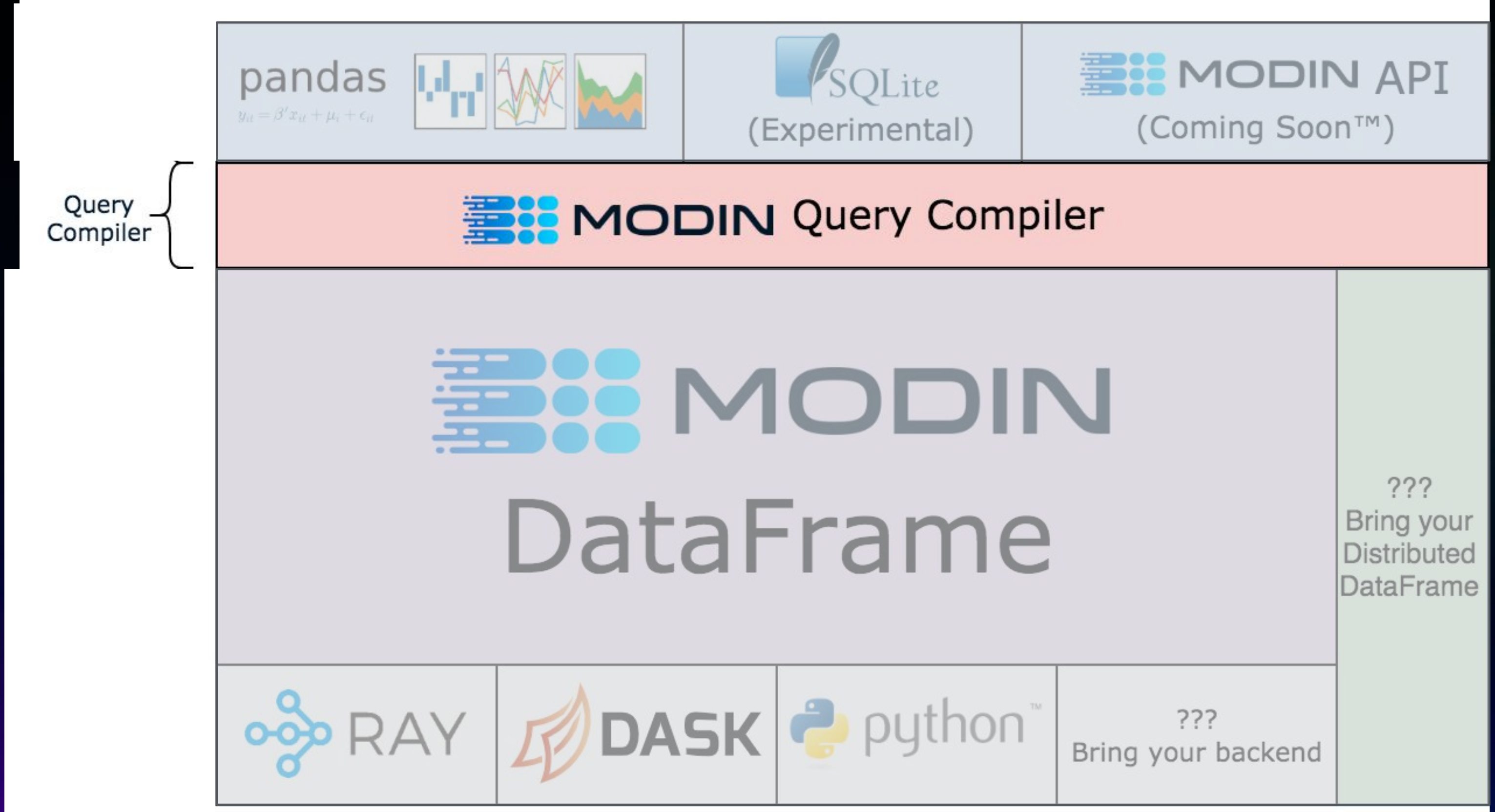
Mo



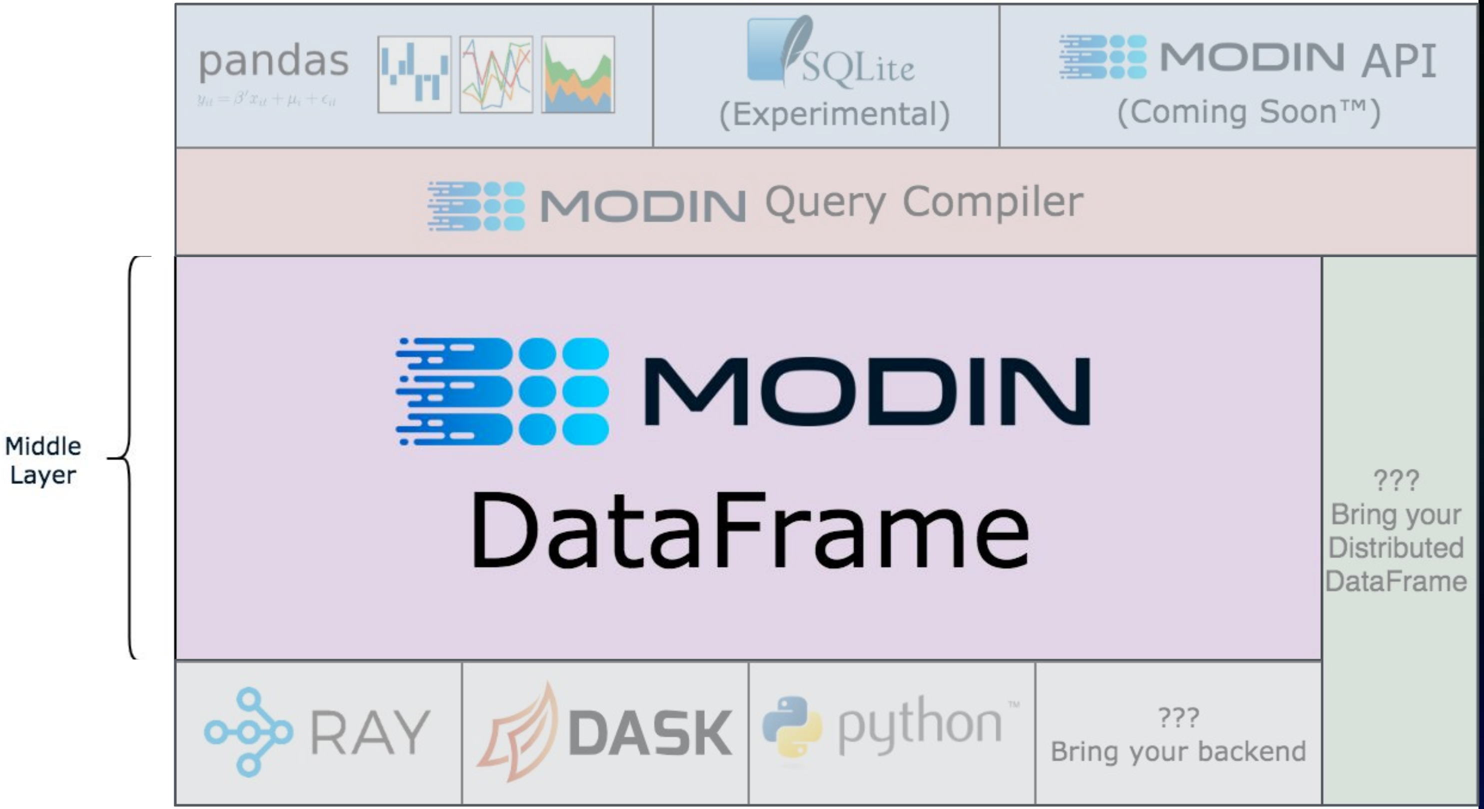
APIs
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python
???
Bring your backend

PONDER

Mo


pandas
SQLite
(Experimental)
MODIN API
(Coming Soon™)
Query Compiler
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python
???
Bring your backend

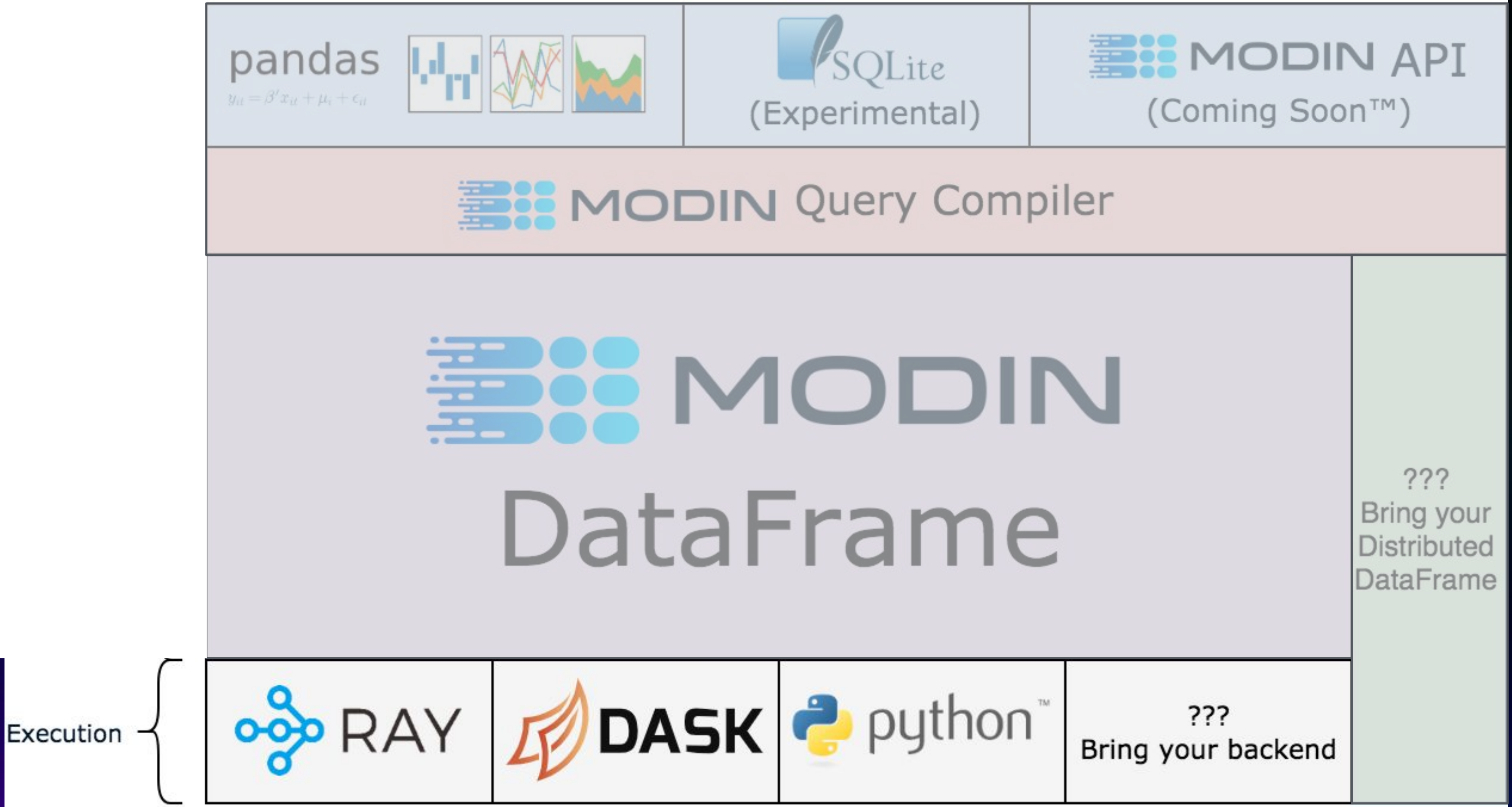
Mo



pandas
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
Middle Layer
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python
???
Bring your backend

PONDER

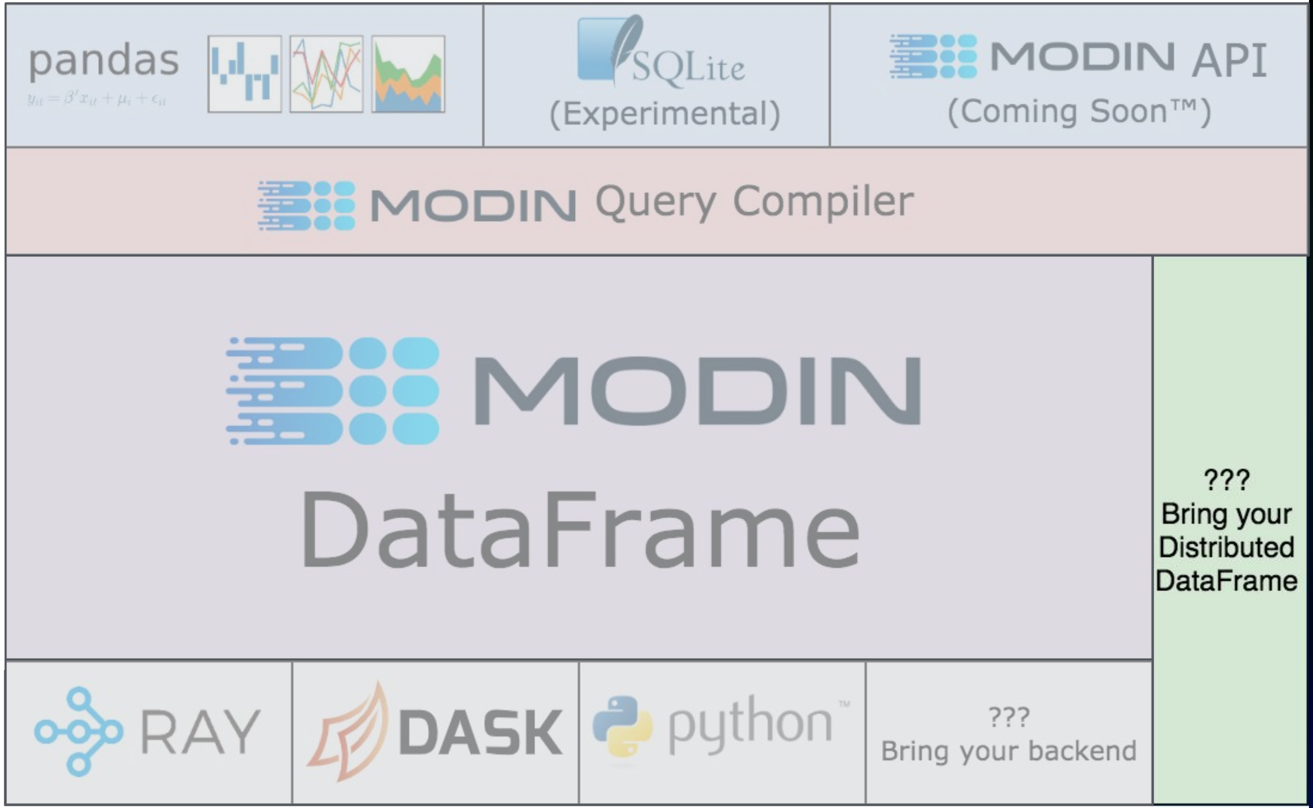
Mo



pandas
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
Execution
RAY
DASK
python™
???
Bring your backend

PONDER

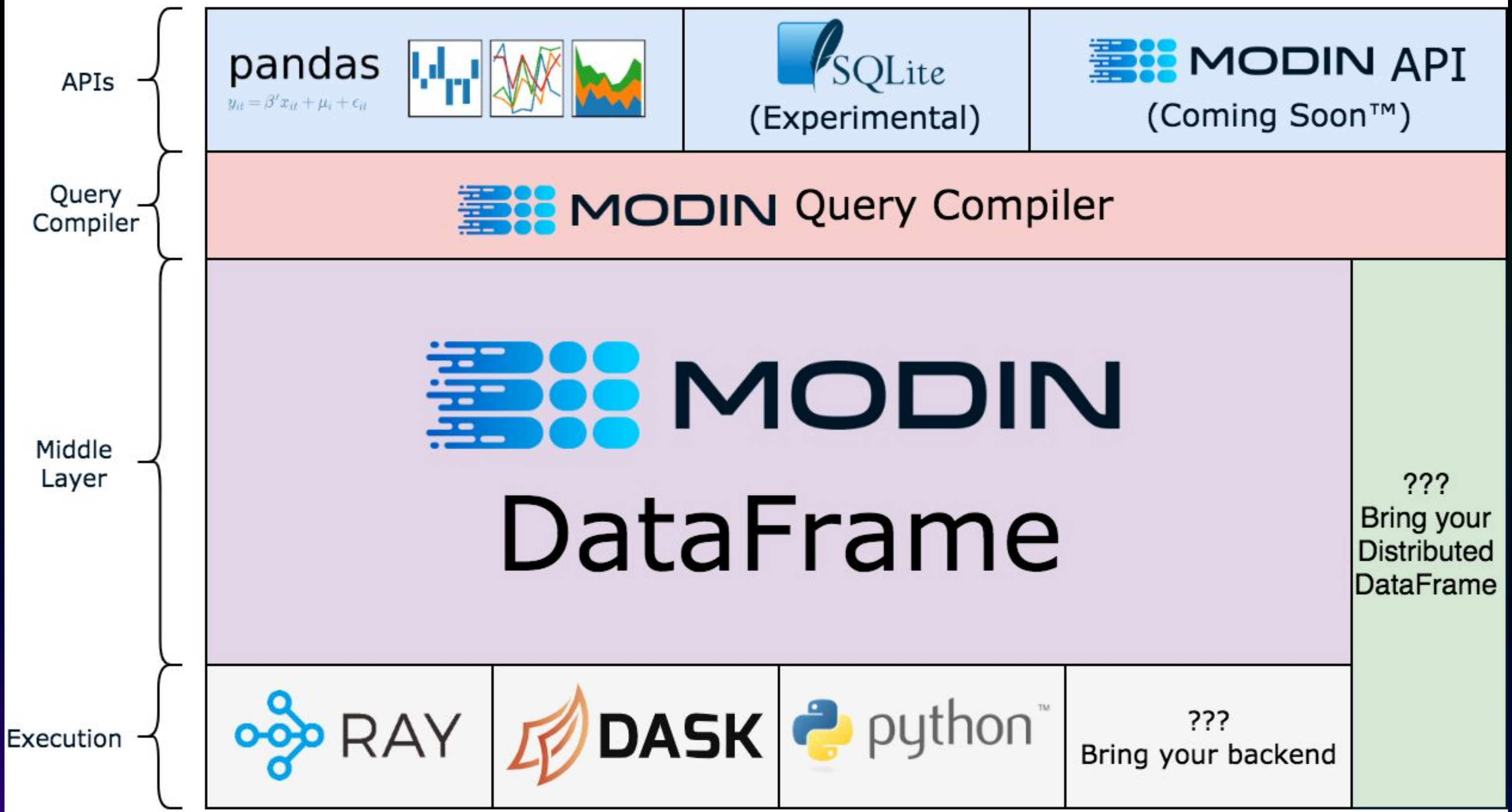
Mo



pandas
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python
???
Bring your backend

PONDER

Mo

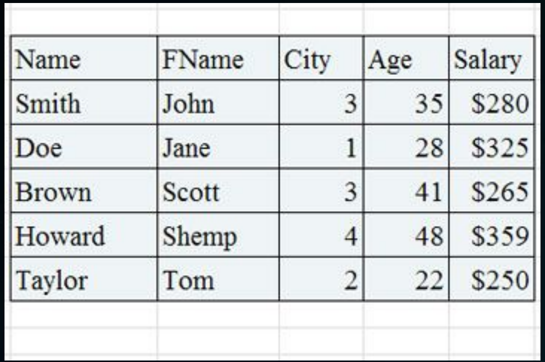


APIs
Query Compiler
Middle Layer
Execution
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python™
???
Bring your backend

Dataframes emerged from a need to hybridize

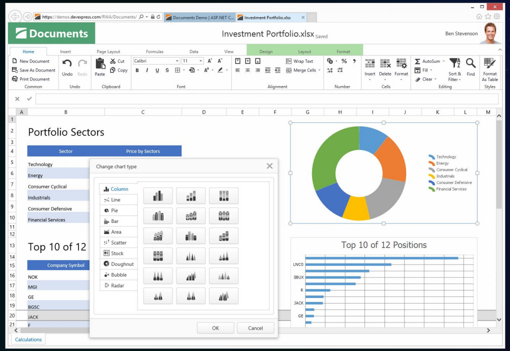
Matrix

Relational Table



Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Spreadsheet



PONDER

Dataframes emerged from a need to hybridize

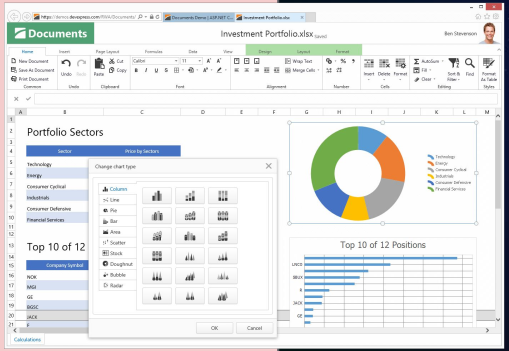
Matrix

$$\begin{pmatrix} 0 & 3 & 1 & 0 & 2 & 3 & 8 & 1 & 1 & 3 \\ 1 & 1 & 0 & 0 & 7 & 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 2 & 0 & 0 & 6 & 7 & 1 & 2 & 2 \\ 1 & 2 & 3 & 10 & 0 & 4 & 6 & 1 & 0 & 5 \\ 3 & 2 & 2 & 1 & 4 & 3 & 2 & 1 & 6 & 0 \\ 7 & 4 & 4 & 5 & 3 & 9 & 6 & 1 & 6 & 1 \\ 7 & 1 & 1 & 5 & 2 & 8 & 9 & 1 & 3 & 6 \\ 5 & 0 & 1 & 6 & 2 & 0 & 0 & 0 & 1 & 5 \\ 1 & 6 & 3 & 3 & 4 & 6 & 2 & 0 & 1 & 1 \\ 1 & 2 & 2 & 4 & 1 & 1 & 3 & 0 & 8 & 2 \end{pmatrix}$$

Relational Table

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Spreadsheet



Dataframes

Dataframe Algebra

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS

"Position notation"
or
"Named notation"

```
df.iloc[row_pos, col_pos]
df.loc[row_lab, col_lab]
```

Dataframe Algebra

PONDER

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS

Ordered union

```
df.append(df2)
```

Dataframe Algebra

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS

Ordered Joins bring fundamentally new challenges

```
df.merge(df2, how="inner")
df2.merge(df, how="inner")
```

Dataframe Algebra

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS

This is a generalized definition of groupby

```
df.count()
df.groupby(df.columns).count()
```

Dataframe Algebra

PONDER

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS

Conceptually a rolling function, can output same table shape or smaller (groupby)

```
df.cumsum()
df.rolling
```

Dataframe Algebra

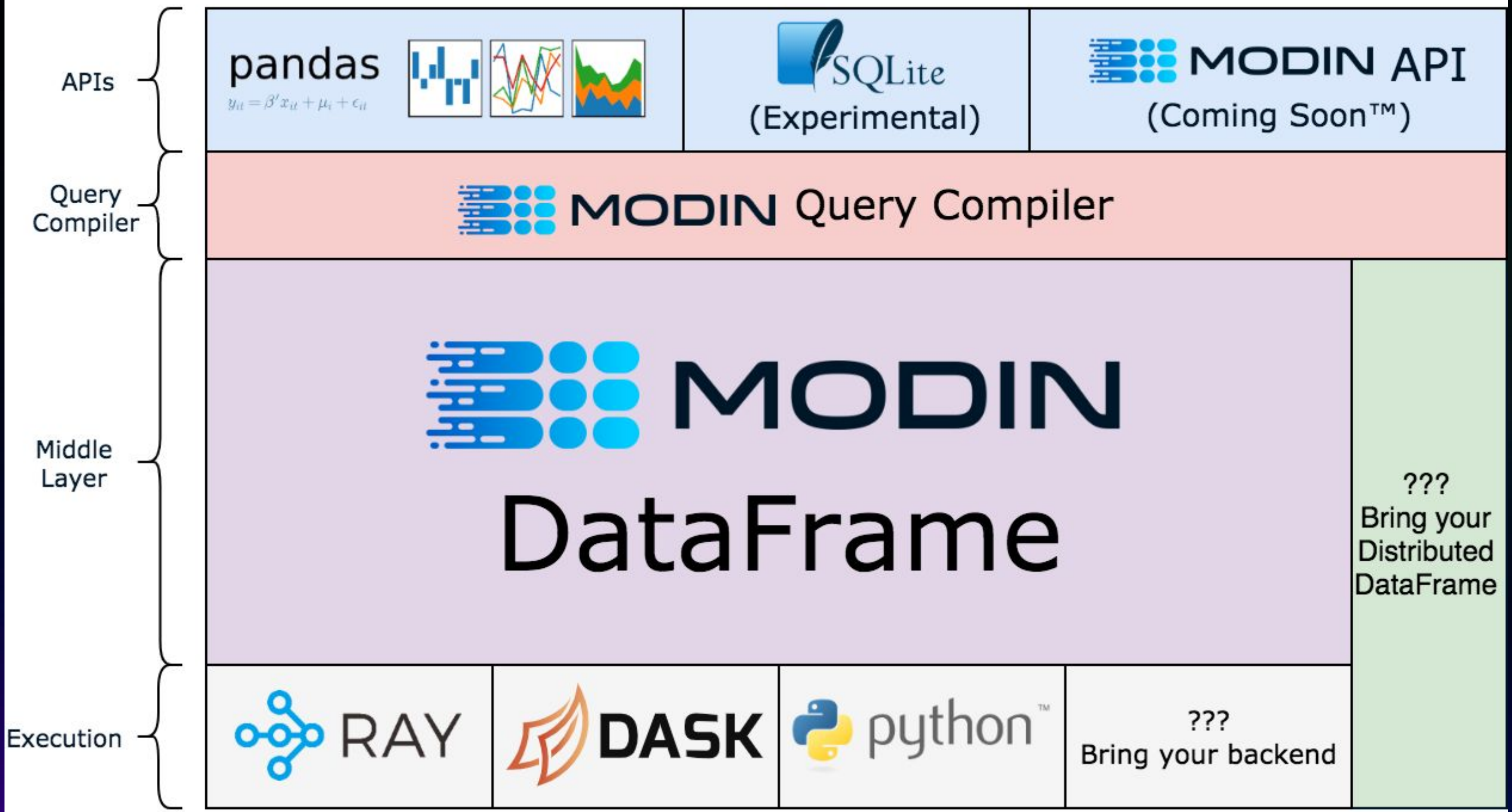
PONDER

Operator
SELECTION
PROJECTION
UNION
DIFFERENCE
CROSS-PRODUCT / JOIN
DROP DUPLICATES
GROUPBY
SORT
RENAME
WINDOW
TRANSPOSE
MAP
TOLABELS
FROMLABELS

```
df.reset_index(drop=False)
```

Insert the row labels into the data and reset the row labels to the positional notation

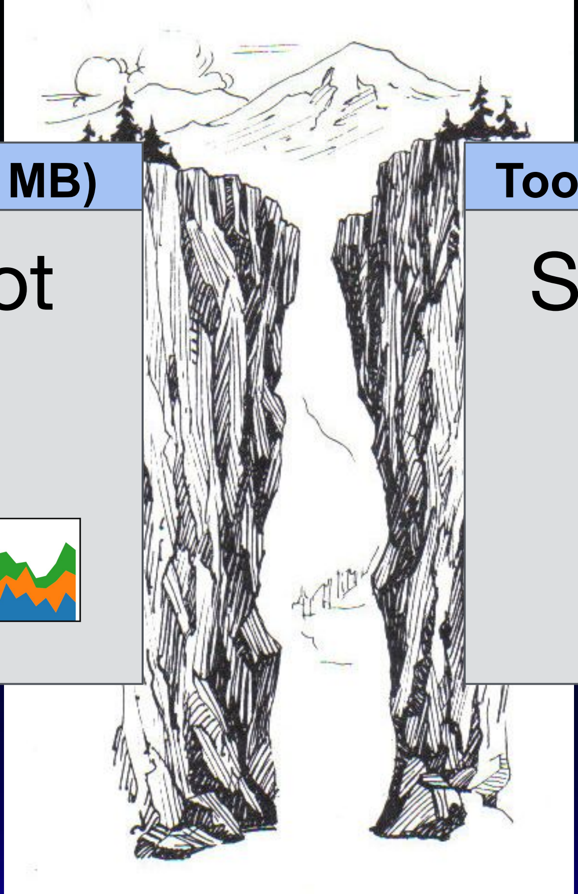
Mo


APIs
Query Compiler
Middle Layer
Execution
pandas
SQLite
(Experimental)
MODIN API
(Coming Soon™)
MODIN Query Compiler
MODIN
DataFrame
???
Bring your Distributed DataFrame
RAY
DASK
python
???
Bring your backend

PONDER

Data Science Landscape: Today

PONDER

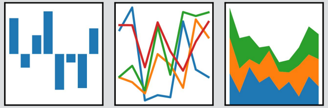


Tools efficient for O(1MB)

Usable but not scalable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Tools efficient for O(100s GB+)

Scalable but not usable

APACHE Spark™

Data Science Landscape: Today

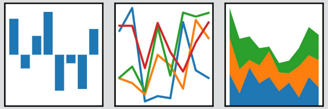
PONDER

Tools efficient for O(1MB)

Usable and scalable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



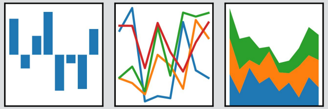
MODIN

Tools efficient for O(100s GB+)

Scalable and usable

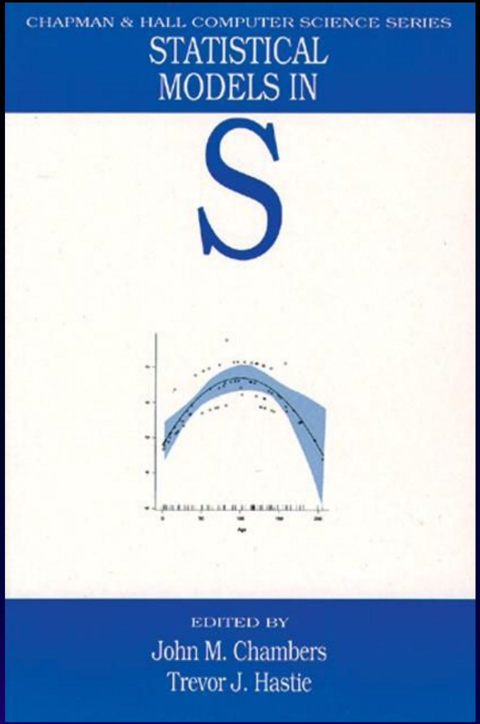
pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Dataframe origin

A (not so) long time ago, at Bell Labs

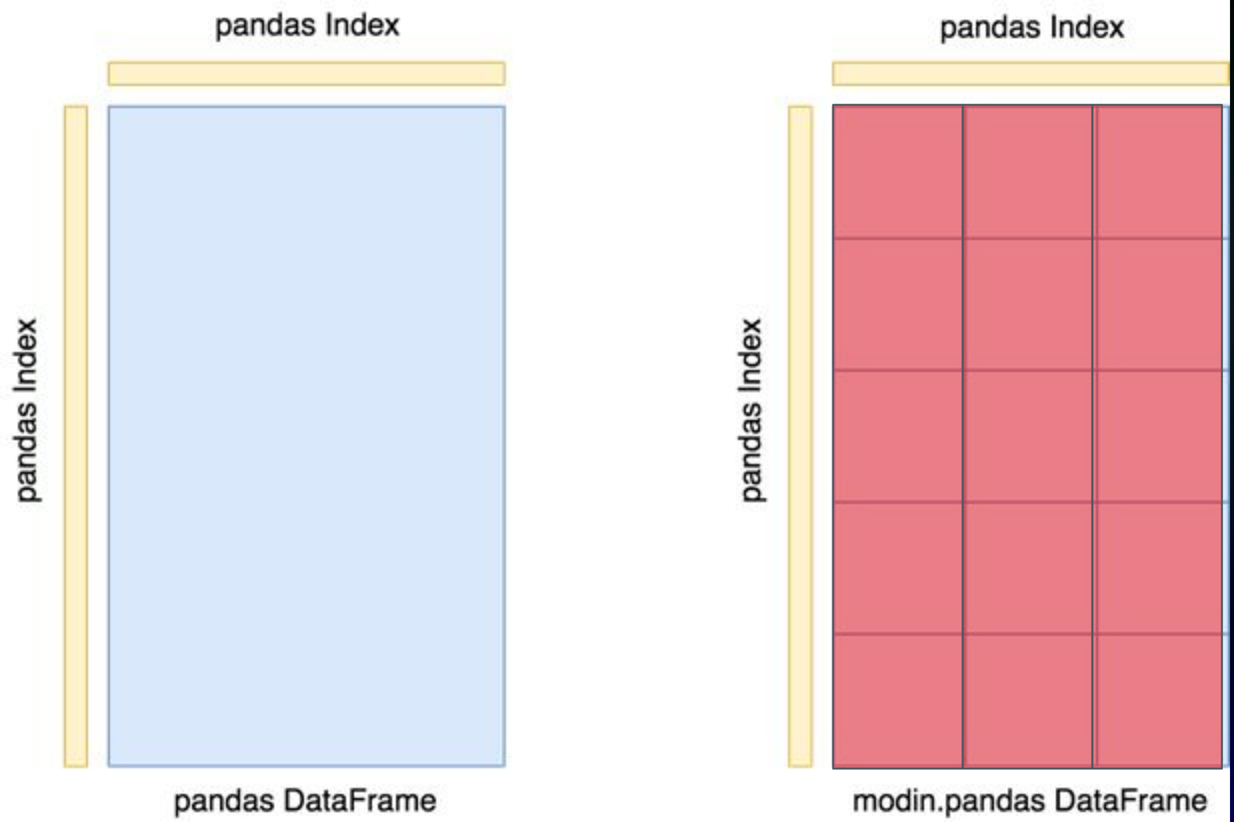


Chapter 3: Data for Models

“Dataframes are more general than matrices in the sense that matrices in S assume all elements to be of the same mode—all numeric, all logical, all character string, etc.”

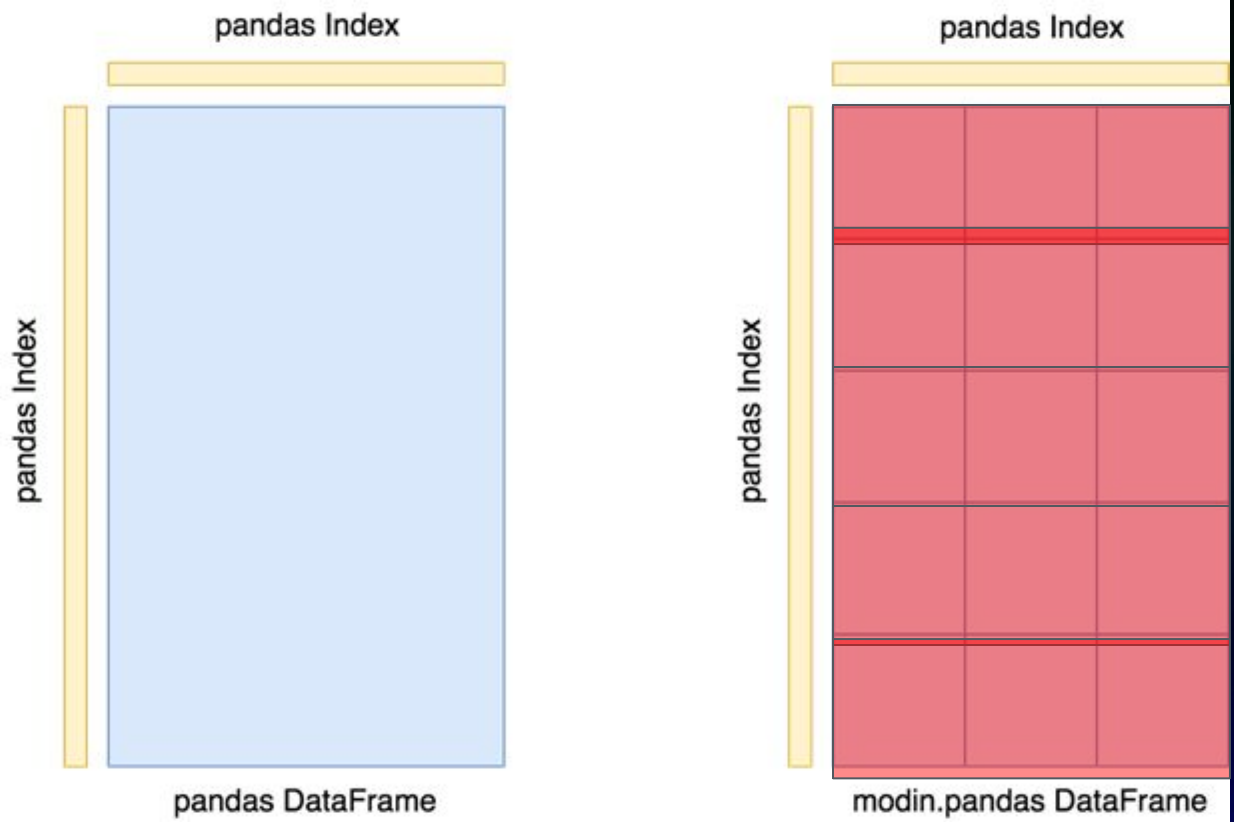
“**... data frames support matrix-like computation,** with variables as columns and observations as rows,
and, in addition, they allow computations in which the variables actas separate objects, referred to by name.”

Modin partitioning - logical column partitioning



Modin partitioning - logical row partitioning

PONDER



Modin architecture

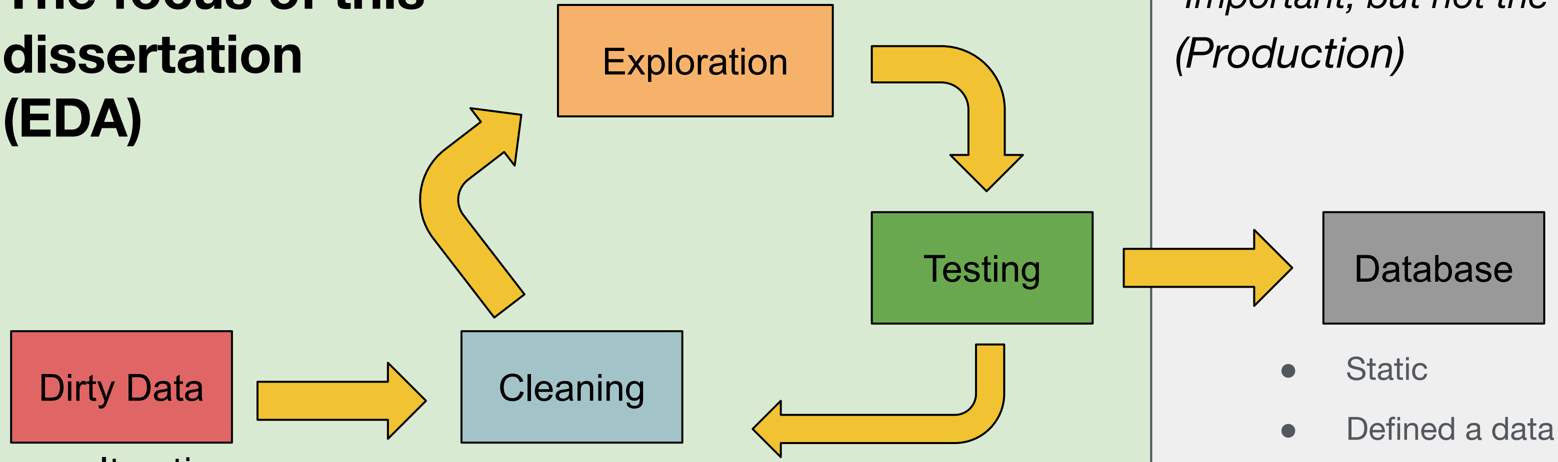
PONDER

- Highly flexible
- Layered architecture
  - New optimizations can be implemented as they are developed
- Support for dataframe algebra
- Partitioning approach lends itself to allowing us to use optimizations from multiple domains

Data Science LifeCycle

PONDER

The focus of this dissertation (EDA)



- Iterative
- **Unstructured** data model
- Schema is lazily induced
- No decoupled physical representation from logical

Important, but not the focus (Production)

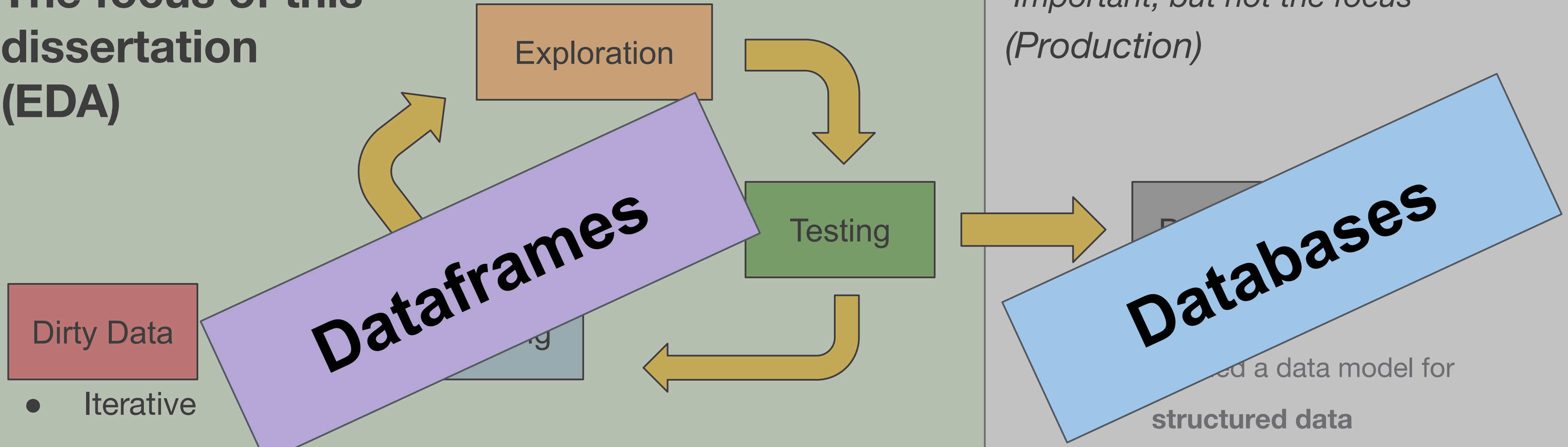
- Static
- Defined a data model for **structured data**
- Schema must be known before input into DBMS
- Decoupling of the physical representation from logical

Data Science LifeCycle

PONDER

The focus of this dissertation (EDA)

Important, but not the focus (Production)



- Iterative
- **Unstructured** data model
- Schema is lazily induced
- No decoupled physical representation from logical

- ed a data model for **structured data**
- Schema must be known before input into DBMS
- Decoupling of the physical representation from logical