Strongly Typed Datasets in a Weakly Typed World

Marco Neumann
@crepererum

PyCon.DE 2018

BlueYonder
a jda. company

Data Engineering

For Production

In [1]:
```
import pandas as pd
```

In [2]:
```
ser1 = pd.Series([True, 2, '3'], dtype=object)
ser1 + ser1
```

Out[2]:
```
0     2
1     4
2    33
dtype: object
```

In [3]:
```
ser2 = pd.Series([True], dtype=object)
~ser2
```

Out[3]:
```
0    -2
dtype: object
```

In [4]:
```
ser3 = pd.Series([1 << 53, (1 << 53) + 1], dtype=float)
ser3.nunique()
```

Out[4]:
```
1
```

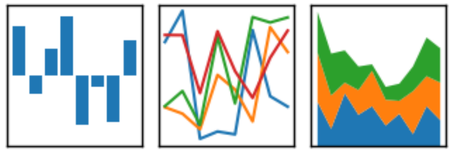
In [5]:
```
ser4 = pd.Series(['2018-01-01', pd.Timestamp('2018-01-02')], dtype=object)
ser4.dt.month
```

```
---------------------------------------------------------------------
--
AttributeError                                Traceback (most recent call las
```

Dataflow

pandas

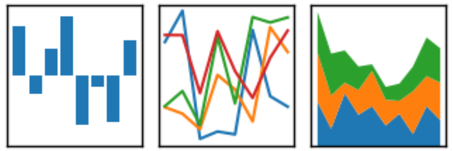
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

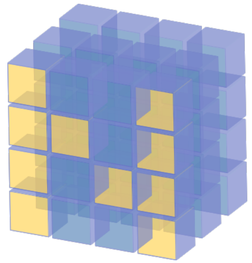


Dataflow

pandas

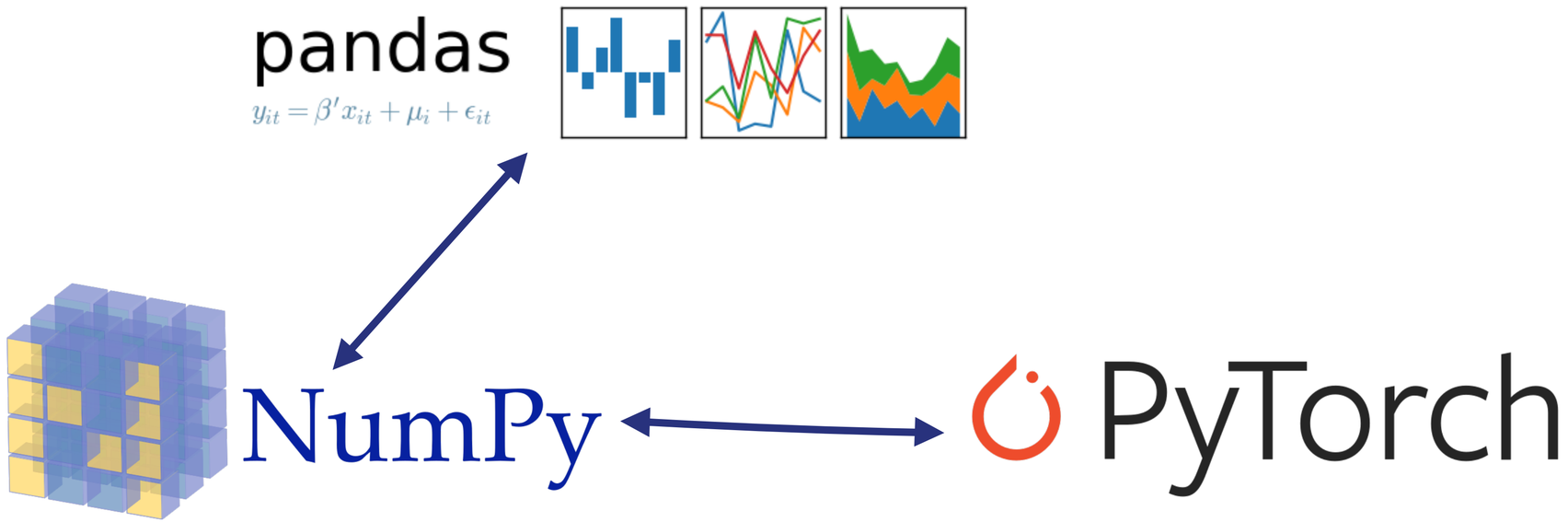
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



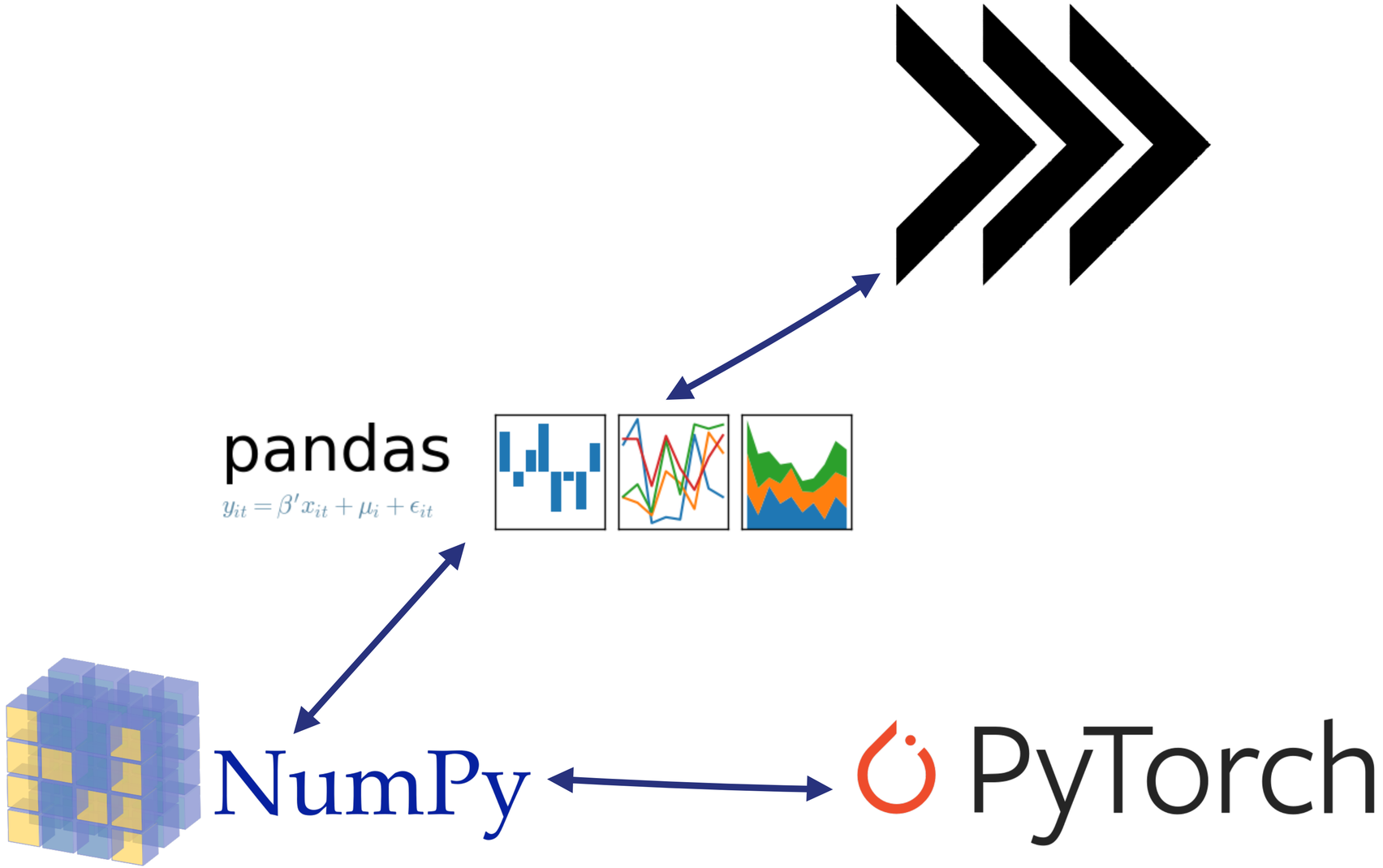


NumPy

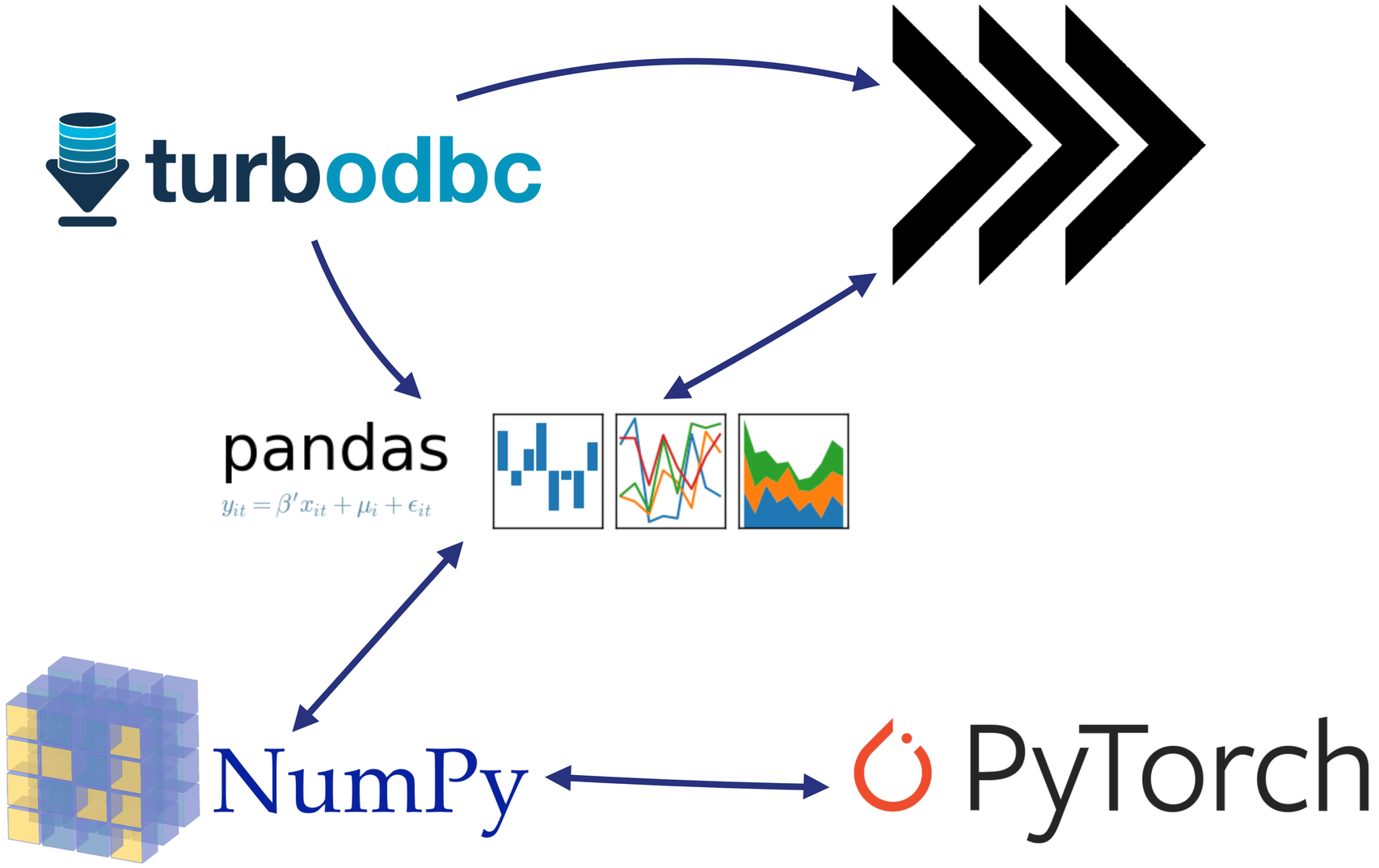
Dataflow



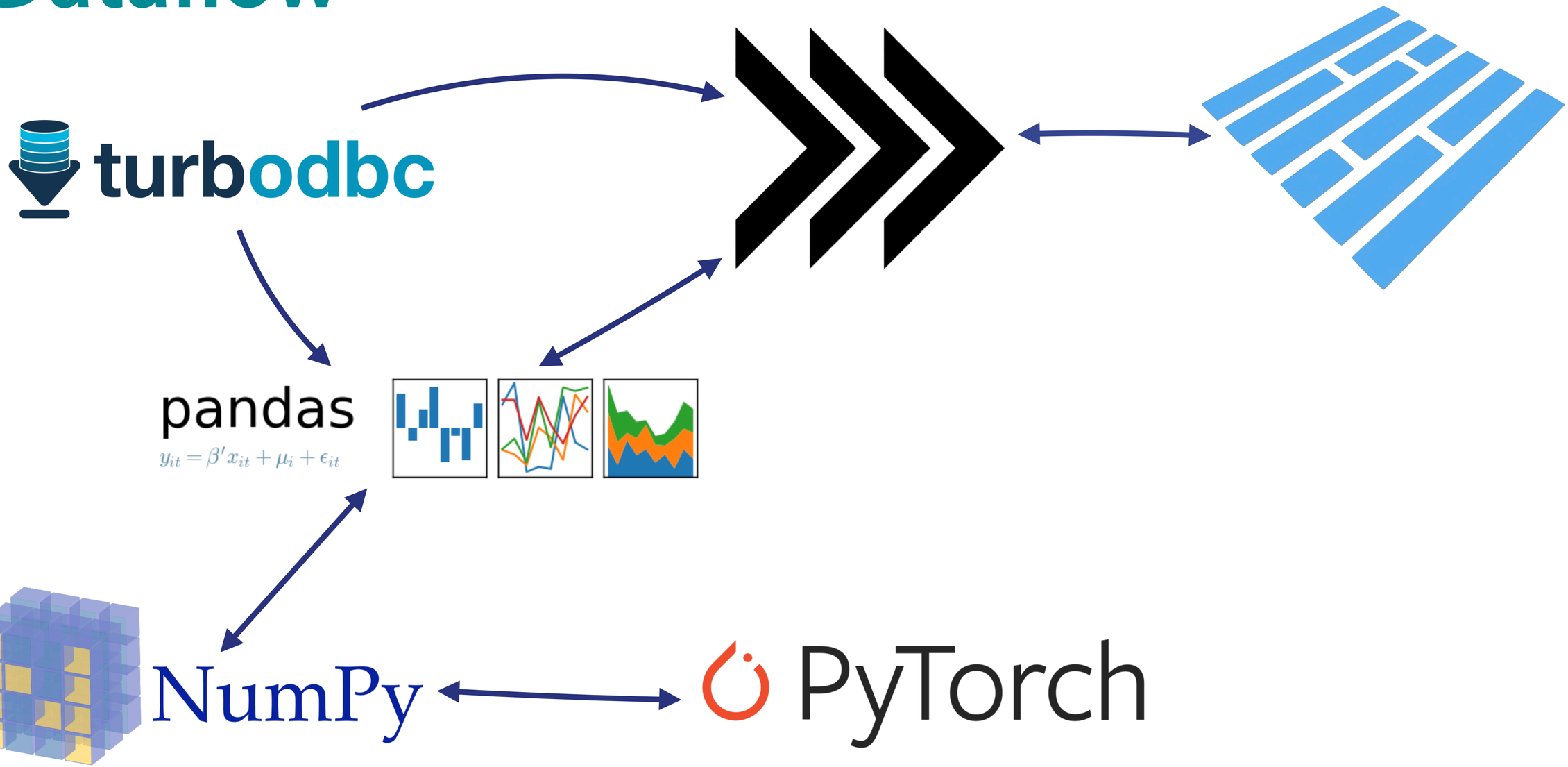
Dataflow



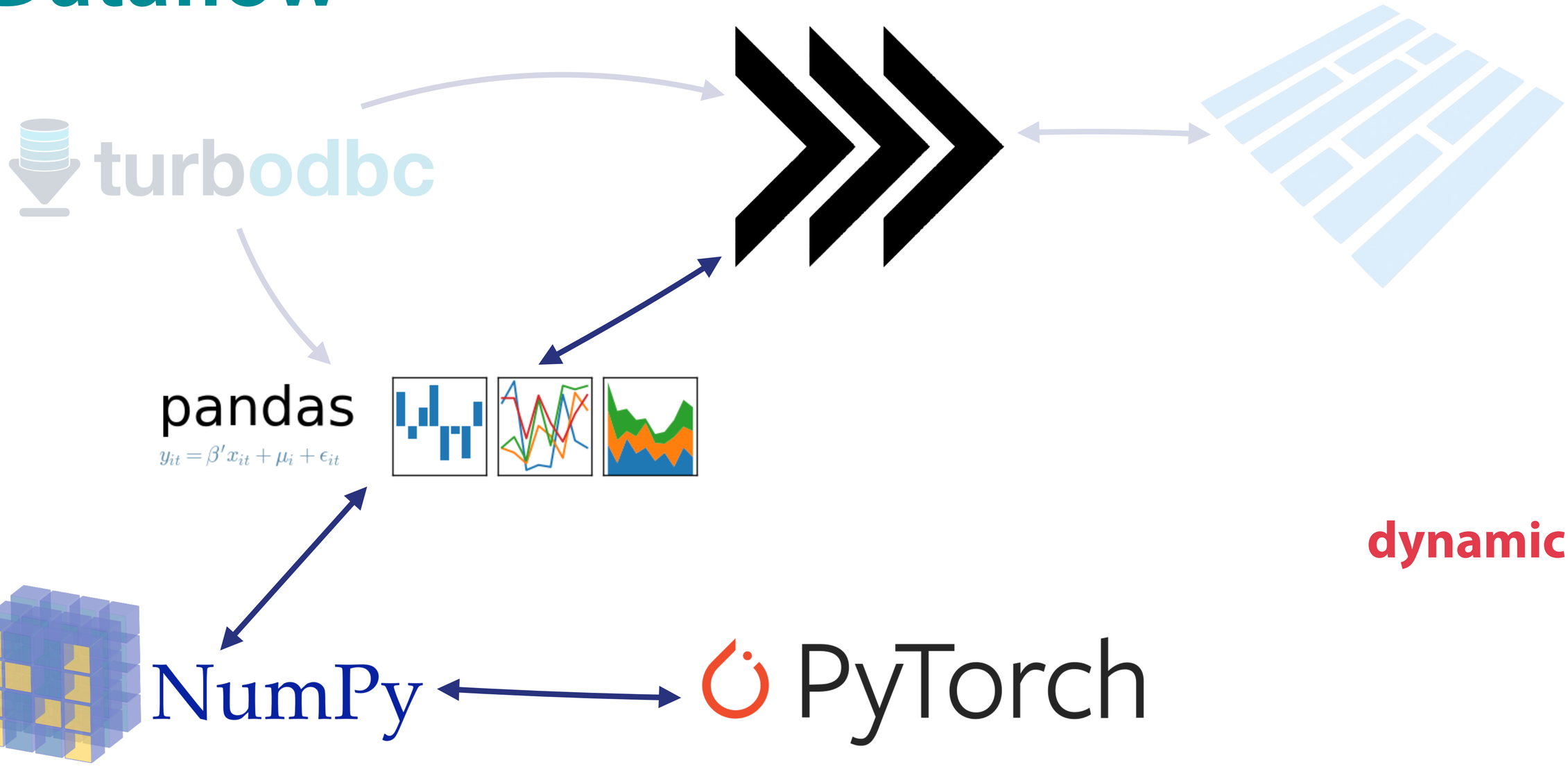
Dataflow



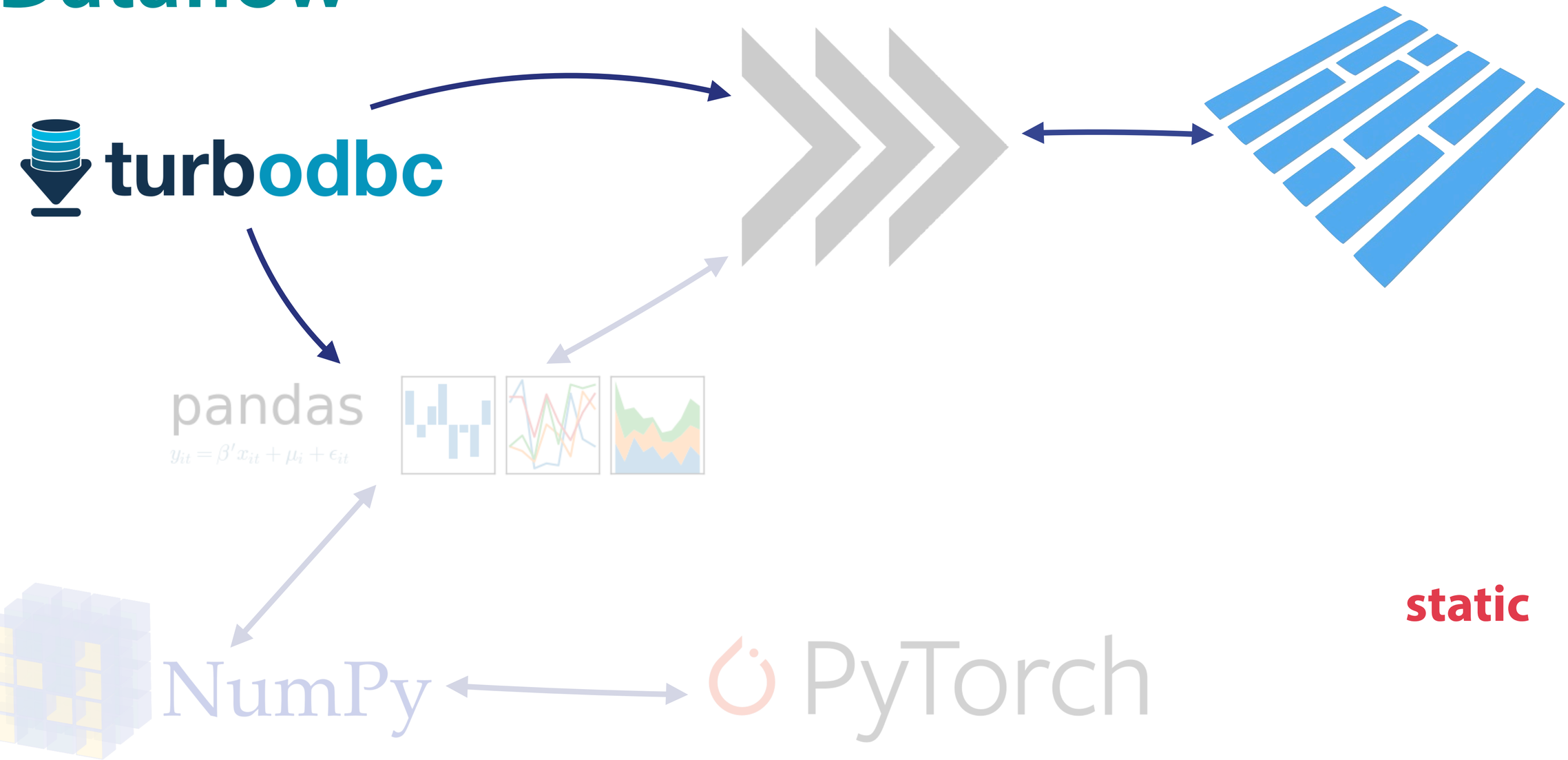
Dataflow



Dataflow


turbodbc
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
NumPy
PyTorch
dynamic

Dataflow



static

https://docs.scipy.org/doc/numpy/user/basics.types.html
https://docs.scipy.org/doc/numpy/reference/arrays.dtypes.html

Numpy

Data types

See also:
Data type objects

Array types and conversions between types

NumPy supports a much greater variety of numerical types than Python does. This section shows which are available, and how to modify an array's data-type.

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
intc	Identical to C `int` (normally `int32` or `int64`)
intp	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)

Pandas

https://pandas.pydata.org/pandas-docs/stable/basics.html#dtypes
https://pandas.pydata.org/pandas-docs/stable/extending.html
https://pandas.pydata.org/pandas-docs/stable/categorical.html

dtypes

The main types stored in pandas objects are `float`, `int`, `bool`, `datetime64[ns]` and `datetime64[ns, tz]`, `timedelta[ns]`, `category` and `object`. In addition these dtypes have item sizes, e.g. `int64` and `int32`. See Series with TZ for more detail on `datetime64[ns, tz]` dtypes.

Extension Types

New in version 0.23.0.

Warning: The **`pandas.api.extension.ExtensionDtype`** and **`pandas.api.extension.ExtensionArray`** APIs are new and experimental. They may change between versions without warning.

Pandas defines an interface for implementing data types and arrays that *extend* NumPy's type system. Pandas itself uses the extension system for some types that aren't built into NumPy (categorical, period, interval, datetime with

Categorical Data

This is an introduction to pandas categorical data type, including a short comparison with R's `factor`.

PyTorch

https://pytorch.org/docs/stable/tensors.html

torch.Tensor

A `torch.Tensor` is a multi-dimensional matrix containing elements of a single data type.

Torch defines eight CPU tensor types and eight GPU tensor types:

Data type	dtype	CPU tensor	GPU tensor
32-bit floating point	`torch.float32` or `torch.float`	`torch.FloatTensor`	`torch.cuda.Flo`
64-bit floating point	`torch.float64` or `torch.double`	`torch.DoubleTensor`	`torch.cuda.Dou`
16-bit floating point	`torch.float16` or `torch.half`	`torch.HalfTensor`	`torch.cuda.Hal`
8-bit integer (unsigned)	`torch.uint8`	`torch.ByteTensor`	`torch.cuda.Byt`
8-bit integer (signed)	`torch.int8`	`torch.CharTensor`	`torch.cuda.Cha`
16-bit integer (signed)	`torch.int16` or `torch.short`	`torch.ShortTensor`	`torch.cuda.Sho`
32-bit integer (signed)	`torch.int32` or `torch.int`	`torch.IntTensor`	`torch.cuda.Int`
64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.LongTensor`	`torch.cuda.Lon`

Turbodbc

https://turbodbc.readthedocs.io/en/latest/pages/getting_started.html
https://turbodbc.readthedocs.io/en/latest/pages/advanced_usage.html

Database type(s)	Python type
Integers, `DECIMAL(<19,0)`	`int`
`DOUBLE`, `DECIMAL(<19, >0)`	`float`
`DOUBLE`, `DECIMAL(>18, 0)`	`unicode` (`str`) or `int` *
`DOUBLE`, `DECIMAL(>18, >0)`	`unicode` (`str`) or `float` *
`BIT`, boolean-like	`bool`
`TIMESTAMP`, `TIME`	`datetime.datetime`
`DATE`	`datetime.date`
`VARCHAR`, strings	`unicode` (`str`)

*) The conversion depends on turbodbc's `large_decimals_as_64_bit_types` option.

https://arrow.apache.org/docs/python/api.html

Pyarrow

API Reference

Type and Schema Factory Functions ¶

`null` ()	Create instance of null type
`bool_` ()	Create instance of boolean type
`int8` ()	Create instance of signed int8 type
`int16` ()	Create instance of signed int16 type
`int32` ()	Create instance of signed int32 type
`int64` ()	Create instance of signed int64 type
`uint8` ()	Create instance of unsigned int8 type
`uint16` ()	Create instance of unsigned uint16 type
`uint32` ()	Create instance of unsigned uint32 type
`uint64` ()	Create instance of unsigned uint64 type
`float16` ()	Create half-precision floating point type

https://github.com/apache/parquet-format/blob/master/LogicalTypes.md

Parquet

Parquet Logical Type Definitions

Logical types are used to extend the types that parquet can be used to store, by specifying how the primitive types should be interpreted. This keeps the set of primitive types to a minimum and reuses parquet's efficient encodings. For example, strings are stored as byte arrays (binary) with a UTF8 annotation.

This file contains the specification for all logical types.

Metadata

The parquet format's `LogicalType` stores the type annotation. The annotation may require additional metadata fields, as well as rules for those fields. There is an older representation of the logical type annotations called `ConvertedType`. To support backward compatibility with old files, readers should interpret `LogicalTypes` in the same way as `ConvertedType`, and writers should populate `ConvertedType` in the metadata according to well defined conversion rules.

Perfect World

Preservation

```
In [1]: import dask.dataframe as dd
        import numpy as np
        import pandas as pd

In [2]: df = pd.DataFrame({
            'date': pd.date_range('2018-01-01', '2018-02-01'),
        })
        df['month'] = df['date'].dt.month
        df['x'] = df['date'].apply(lambda x: np.random.normal())

In [3]: ddf = dd.from_pandas(df, chunksize=100)
        ddf.to_parquet('./dask_dataframe_1', partition_on=['month'])
```

```
dask_dataframe_1
├── _common_metadata
├── month=1
│   └── 552da9ba018c48c1b8722643af3e081d.parquet
└── month=2
    └── cc686be9ffb54f21a5aaca7e9502e3c6.parquet
```

Dask Types

```
dask_dataframe_1
├── _common_metadata
├── month=1
│   └── 552da9ba018c48c1b8722643af3e081d.parquet
└── month=2
    └── cc686be9ffb54f21a5aaca7e9502e3c6.parquet
```

```
In [1]: import pyarrow.parquet as pq

In [2]: part1 = pq.read_table('./dask_dataframe_1/month=1/552da9ba018c48c1b8722643
        print(part1.num_rows)
        print(part1.schema.names)
        print(part1.schema.types)

        31
        ['date', 'x', '__index_level_0__']
        [TimestampType(timestamp[us]), DataType(double), DataType(int64)]
```

Dask Types

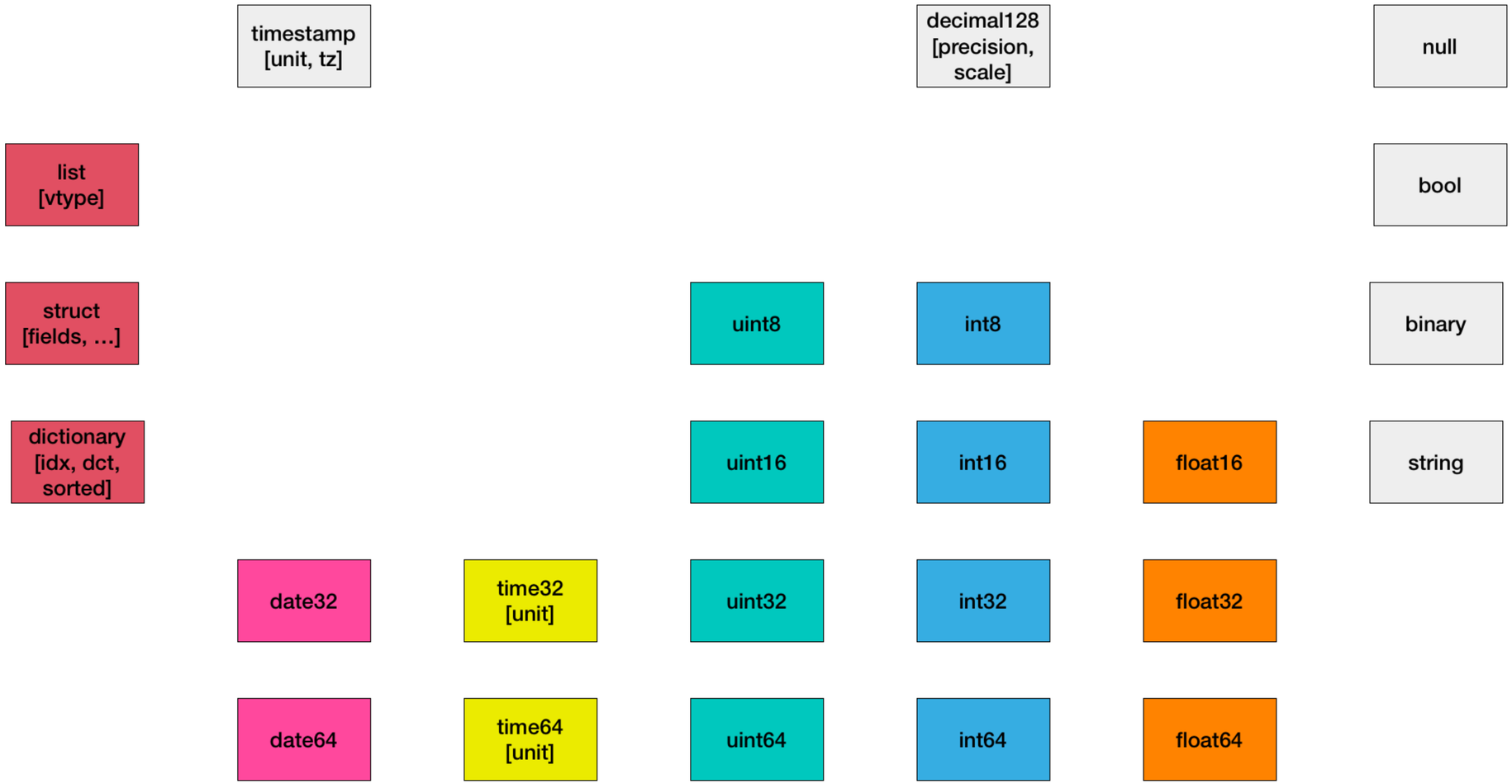
```
dask_dataframe_1
├── _common_metadata
├── month=1
│   └── 552da9ba018c48c1b8722643af3e081d.parquet
└── month=2
    └── cc686be9ffb54f21a5aaca7e9502e3c6.parquet
```

In [1]:
```
import pyarrow.parquet as pq
```

In [2]:
```
common = pq.read_table('./dask_dataframe_1/_common_metadata')
print(common.num_rows)
print(common.schema.names)
print(common.schema.types)
```

```
0
['date', 'month', 'x', '__index_level_0__']
[TimestampType(timestamp[us]), DataType(int64), DataType(double), DataTyp
e(int64)]
```

Arrow Types


timestamp
[unit, tz]
decimal128
[precision,
scale]
null
list
[vtype]
bool
struct
[fields, ...]
uint8
int8
binary
dictionary
[idx, dct,
sorted]
uint16
int16
float16
string
date32
time32
[unit]
uint32
int32
float32
date64
time64
[unit]
uint64
int64
float64

Type Variance

Turbodbc

```
>>> cursor.execute("SELECT * FROM (VALUES(1), (2), (3))")
>>> table = cursor.fetchallarrow()
>>> table
pyarrow.Table
__COL0__: int64
```

```
>>> cursor.execute("SELECT * FROM (VALUES(1), (2), (3))")
>>> table = cursor.fetchallarrow(adaptive_integers=True)
>>> table
pyarrow.Table
__COL0__: int8
```

Pandas

```
In [1]: import pandas as pd

In [2]: df1 = pd.DataFrame({
            'a': [1, 2, 3],
            'x': [10, 20, 30],
        })
        df2 = pd.DataFrame({
            'a': [2, 3, 4],
            'y': [-2, -3, -4],
        })

In [3]: df1.merge(df2, how='left')

Out[3]:
```

	a	x	y
0	1	10	NaN
1	2	20	-2.0
2	3	30	-3.0

dtype	cast to
integer	float
boolean	object

Pandas

```
In [1]: import pandas as pd

In [2]: df = pd.DataFrame({
            's': pd.Series(['one', 'two', 'three'], dtype='category'),
        })

In [3]: df['s'].str.upper()

Out[3]: 0       ONE
        1       TWO
        2     THREE
        Name: s, dtype: object

In [4]: df.dtypes

Out[4]: s    category
        dtype: object
```

Numpy

```
In [1]: import numpy as np

In [2]: a = np.arange(10 * 10, dtype=np.int64).reshape((10, 10))

In [3]: b = np.mean(a, axis=1)
        print(b)
        print(b.dtype)

        [ 4.5 14.5 24.5 34.5 44.5 54.5 64.5 74.5 84.5 94.5]
        float64
```

Type Compatibility

int16 => int64

[-32768, 32767]

[-9223372036854775808, 9223372036854775807]

int16 <= int64

[-32768, 32767]

[-9223372036854775808, 9223372036854775807]

int64 => int64

[-9223372036854775808, 9223372036854775807]

[-9223372036854775808, 9223372036854775807]

int64 => float64

[-9223372036854775808, 9223372036854775807]

[-1.7976931348623157e+308, 1.7976931348623157e+308]

int64 => float64

```
In [1]: import numpy as np

In [2]: x = np.int64((1 << 53) + 1)
        y = np.int64(np.float64(x))
        x, y

Out[2]: (9007199254740993, 9007199254740992)
```

int64 => float64

```
In [1]: import numpy as np

In [2]: x = np.int64((1 << 53) + 1)
        y = np.int64(np.float64(x))
        x, y

Out[2]: (9007199254740993, 9007199254740992)
```

int64 => float64

```
In [1]: import numpy as np

In [2]: x = np.int64((1 << 53) + 1)
        y = np.int64(np.float64(x))
        x, y

Out[2]: (9007199254740993, 9007199254740992)
```

✓ numeric information

✗ IDs

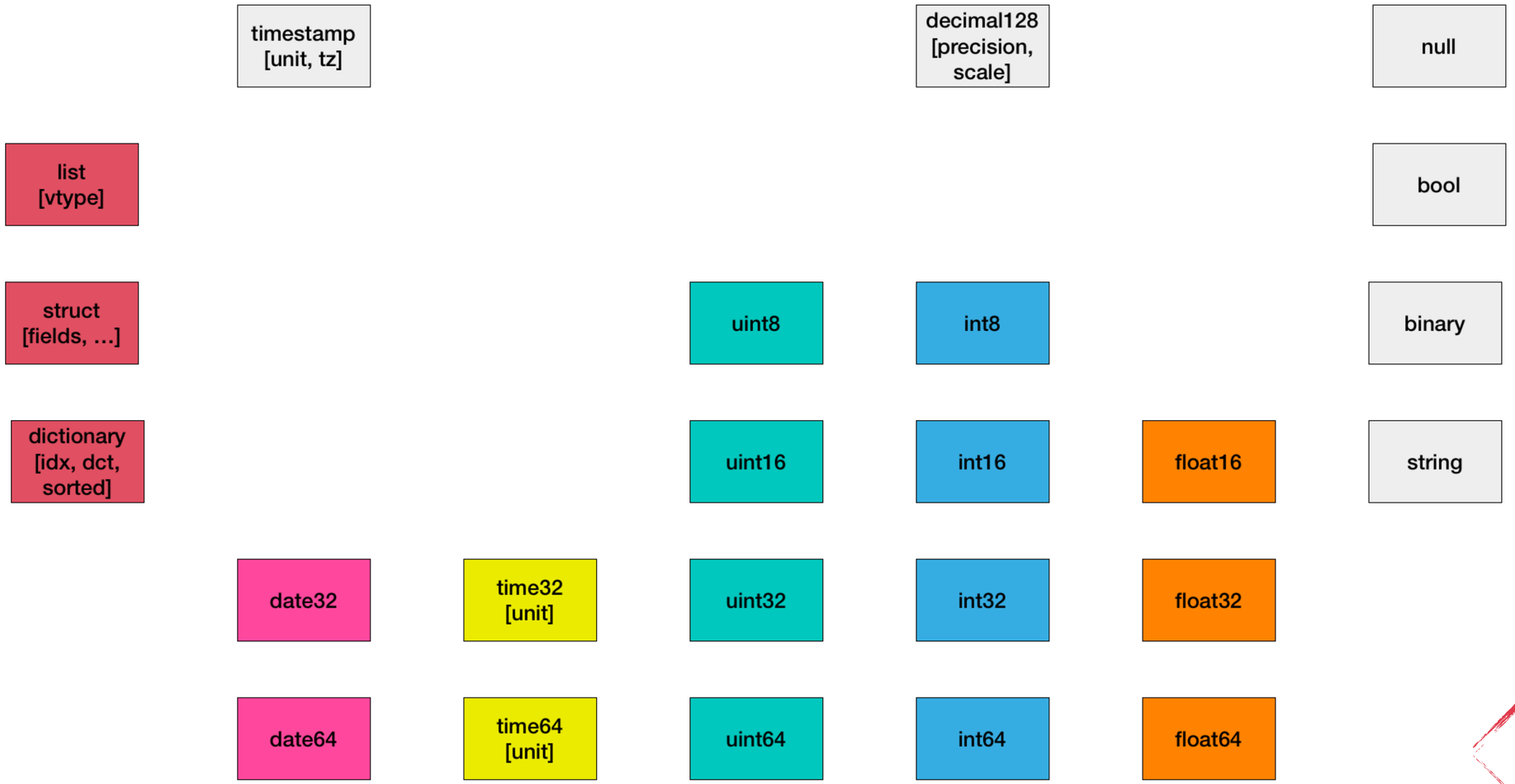
unicode => binary

“München”

“M\xc3\xbcnchen”

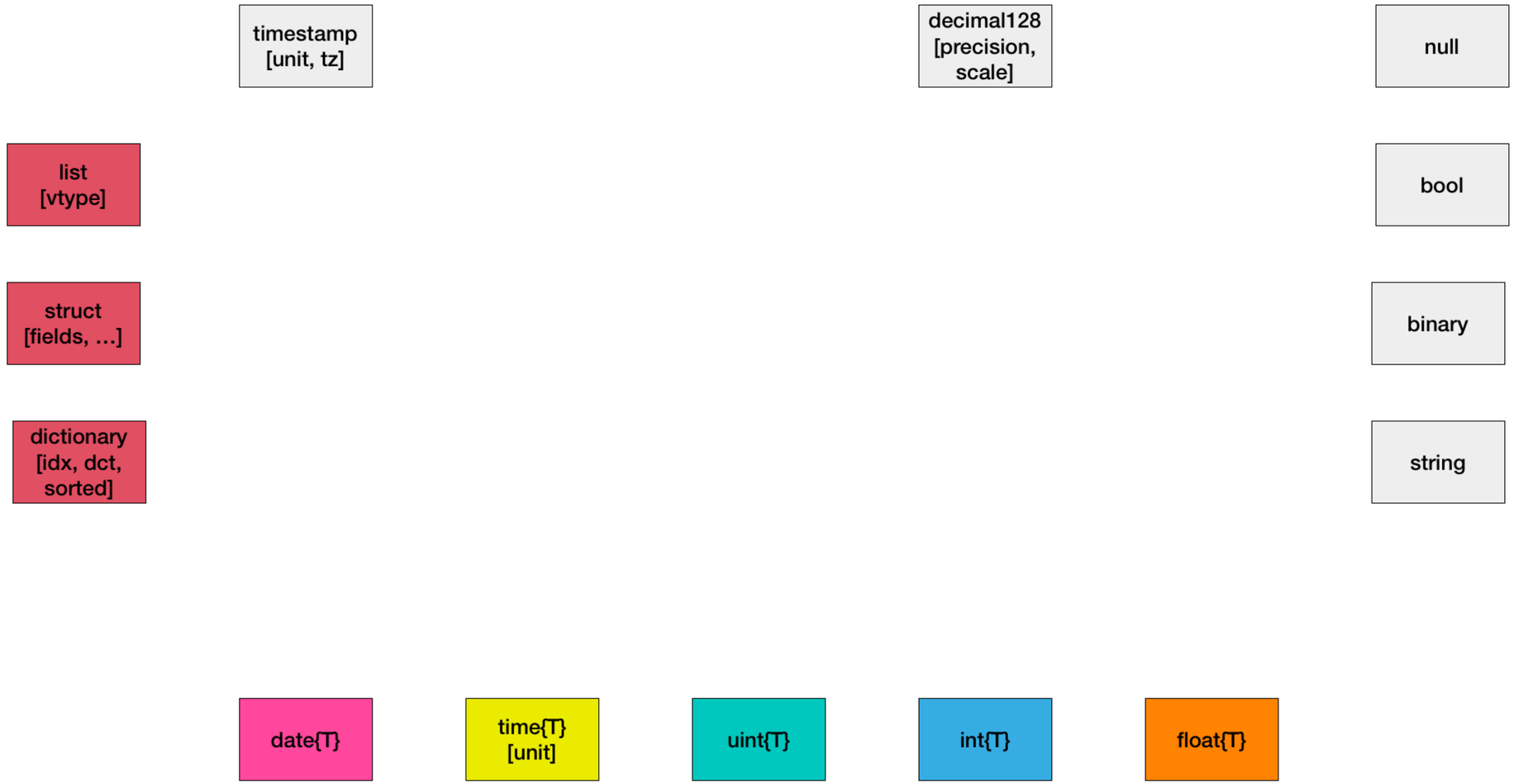
Type Preservation

Arrow Types



RECAP

Arrow Types (modified)



Dask Types

```
dask_dataframe_1
├── _common_metadata
├── month=1
│   └── 552da9ba018c48c1b8722643af3e081d.parquet
└── month=2
    └── cc686be9ffb54f21a5aaca7e9502e3c6.parquet
```

In [1]:
```
import pyarrow.parquet as pq
```

In [2]:
```
common = pq.read_table('./dask_dataframe_1/_common_metadata')
print(common.num_rows)
print(common.schema.names)
print(common.schema.types)
```

```
0
['date', 'month', 'x', '__index_level_0__']
[TimestampType(timestamp[us]), DataType(int64), DataType(double), DataTyp
e(int64)]
```

Type Normalization

```
data AType =
    ABinary |
    ABool |
    ADate Int |
    ADecimal128 Int Int |
    ADictionary AType AType Bool |
    AFloat Int |
    AInt Int |
    AList AType |
    ANull |
    AString |
    ATime Int String |
    ATimestamp String String |
    AUInt Int
    deriving (Show)
```

```
convert :: AType -> AType

convert (AInt w)              = AInt   64
convert (AUInt w)             = AUInt  64
convert (AFloat w)            = AFloat 64
convert (ADate w)             = ADate  64
convert (ATime w u)           = ATime  64 u

convert (AList t)             = AList  (convert t)
convert (ADictionary t i s)   = t

convert x                     = x
```

Takeaways

1. Dynamic Typed Runtime, but Statically Typed Datasets
2. Bit-Width:
    1. Allow Variance during Read/Write
    2. (if required) Store Maximum-Width Type
3. Categoricals:
    1. Allow “Compressed” and “Uncompressed” Data
    2. (if required) Store “Uncompressed”/Internal Type

Thank You

Marco Neumann
@crepererum

1. Bottles, portions, liquid and perfume HD
by Alex Sajan
https://unsplash.com/photos/BNuxGwj-a24

2. Sunset boulevard
by Simon Matzinger
https://unsplash.com/photos/twukN12EN7c

3. Stages of a Monarch Butterfly
by Suzanne D. Williams
https://unsplash.com/photos/VMKBFR6r_jg

4. Lovers bike ride
by Sabina Ciesielska
https://unsplash.com/photos/-2PrzYfgotw

5. Jar, jam, market food HD
by Viktor Forgacs
https://unsplash.com/photos/5mGGOWD-Ths

BlueYonder
a jda. company