PyData Global

Reactive data processing in Python

Adrian Kosowski, PhD
.pathway

What you would want to do:

1. Write your program

```
my_model = train_classifier (X_train, y_train)
my_model.predict (X_test)
```

2. Run some experiments

3. It’s working. Let’s have a coffee ☕ .

What life has in store:

1. Write your program

```
my_model = train_classifier (X_train, y_train)
my_model.predict (X_test)
```

2. Run some experiments

3. ~~It's working. Let's have a coffee ☕.~~
Handle live data updates:

- Redo all the code logic in a streaming architecture.
- Prepare to handle every imaginable (and unimaginable) data input change event ever, in conjunction with your classification & prediction logic 🙃.

Some of the joys of streaming data

1. Data updates are full of surprises

Day 1: *We just need to handle **new** data points as they arrive at input, and we are done.*

Day 77: *By the way, could we please **update** all the data points that arrived in the last hour? Somebody entered distances in miles rather than meters into the* `distance_m feature` *this morning.*
*Our models have already got trained on that data, we will need to **un-learn** that.*

Some of the joys of streaming data

2. Unknown unknowns are real fun.

Over the last week my program has been churning data that it has never seen before.

- *I cannot restart it.*
- *I cannot easily look inside it.*
- *I am not quite sure of what state my program is in, and how it got there.*

Today, one of our models seems to be classifying all objects as: "cow".
What do I do?

Answer

Some of the joys of streaming data

3. Consistency

We have deployed our system for ***live temperature monitoring of a power plant****.*
In the last half minute, the "critical alert" flag has flashed "on" and "off" at least a dozen times.
It seems to be off now. Is our system done with computing yet, or do we wait just a minute longer to be on the safe side?

Falling back to batch execution: a frequent (sad) outcome

"That's enough. I will just rerun every 30 minutes the batch prototype I wrote in pandas / pyspark on all the data we've seen so far. At least it works."

Checklist:

- ❑ **Latency:** will the rerun be fast enough for business needs?
- ❑ **Computing cost:** how many cores will that take?
- ❑ **Reproducibility:** do I need to ensure the results of my re-run are somehow "close" to the previous run?
- ❑ **Future-proofing:** there will be more & bigger data in the future.

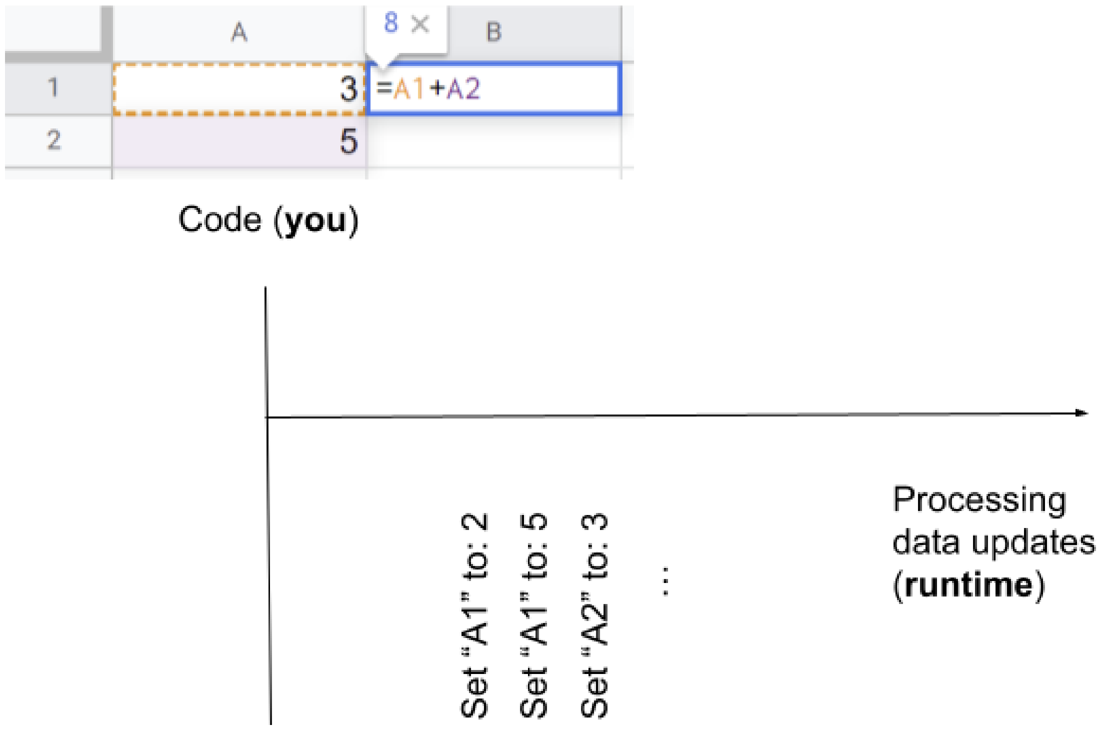
Outcome: A lot of **projects fail to check at least one box** BUT deploy in batch mode anyway, delivering partial value. We have seen it a lot in the enterprise context.

The barrier to deliver streaming data projects in production is often just too high, mostly due to problems with tooling.

Reactive design to the rescue: uncoupling logic from data updates

- You (the Developer): describe your logic as you would for a batch system.
- Let the reactive framework handle data streams and propagate all changes.

We all know what this means in a spreadsheet.

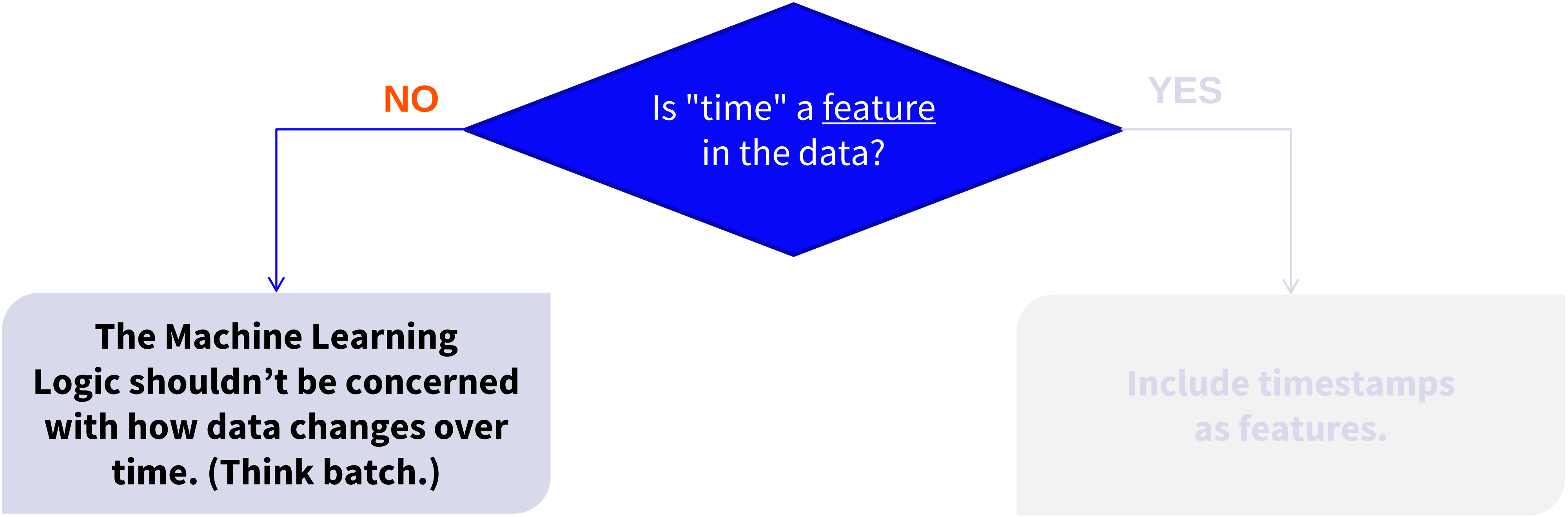


Today we will see what this means:

- **for a Machine Learning pipeline**
- **when working with Python dataframes (tables).**

Preparing our first Machine Learning model, reactively

Task: *"I am interested in classifying handwritten digits. (How original!)*
The training data is changing over time.
I want my model to improve as new training data becomes available."

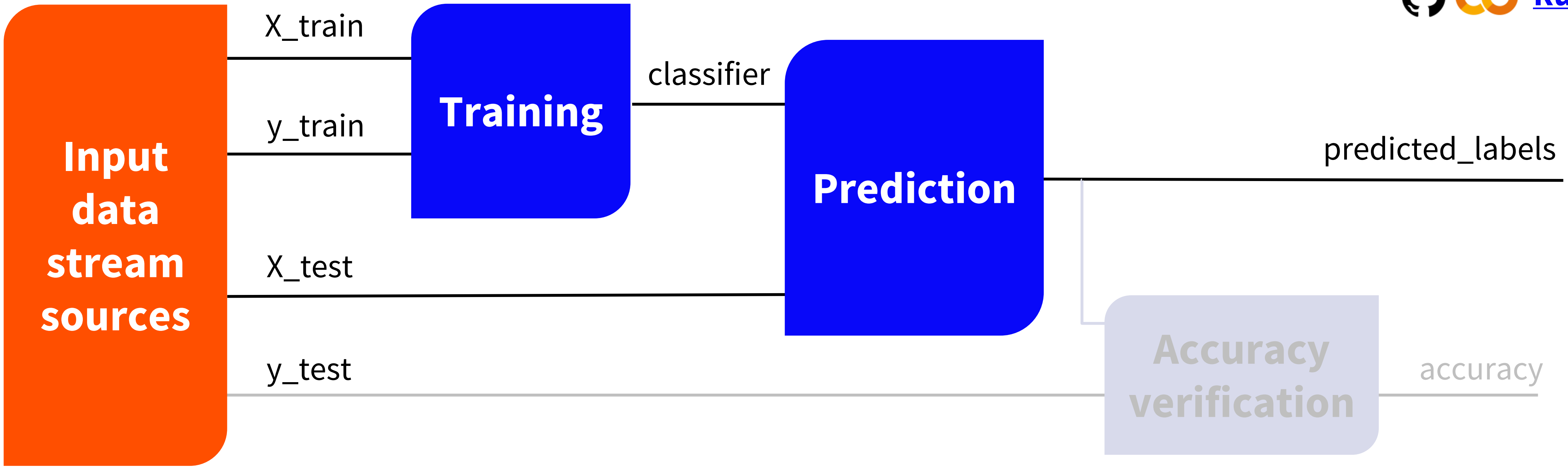


The reactive framework will take care of updating the model as data changes.

Classification Example in Pathway

```
import pathway as pw
X_train, y_train, X_test, y_test = pw.ml.datasets.classification.load_mnist_stream()
classifier = pw.ml.classifiers.knn_lsh_train(X_train, d=28*28), y_train
predicted_labels = pw.ml.classifiers.knn_lsh_classify(*classifier, X_test, k=3)
# accuracy = pw.ml.utils.classifier_accuracy(predicted_labels, y_test)
```

Run it



Classification decisions for incoming data: when and how?

X_test is the table of points for which predictions are needed.
We expect the decisions in X_test to update themselves as the trained model changes over time.
The contents of table X_test are updated, as needed.

Need to take decisions on point *x* ➔ we insert x into X_test
No longer need to take decisions on point x ➔ we delete x from X_test

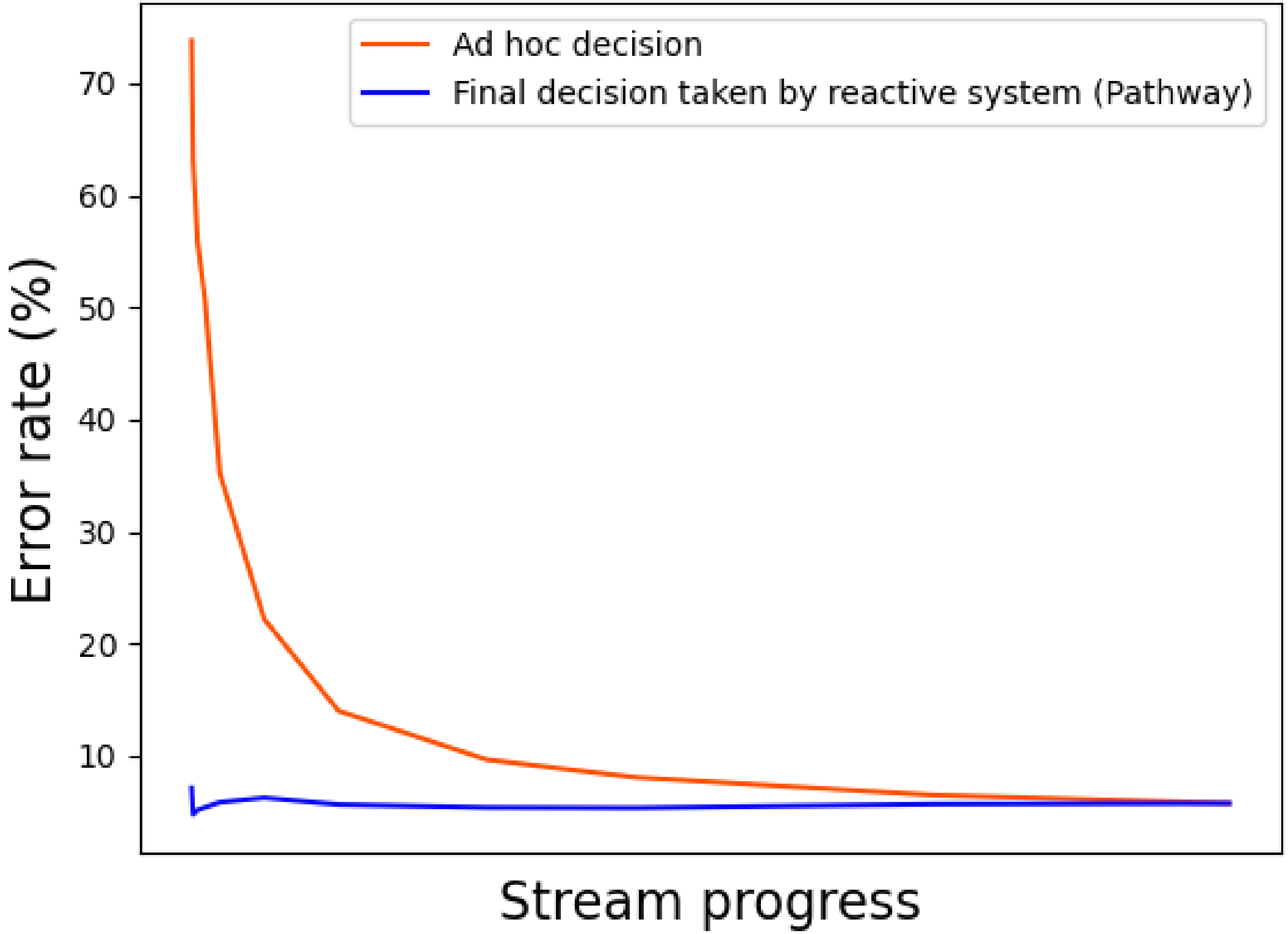
How do I control which decisions should be computed by the system?

We use the setup from the previous slide.
In it, we make sure to define **X_test** correctly “upstream” (to the left of) all the prediction logic.

Decisions in Pathway

How do they play out over time?

Stream behavior example ("streaming MNIST dataset", training set size 60k, test set size 10k), all reshuffled, arriving in a stream, insert-only. Error rates for the basic KNN+LSH classifier.



Pathway: a code primer (1/4)

Tables are at the heart of everything.

You can connect an input table to a **static data source**…

```
table_dogs = pw.debug.table_from_markdown(
    """
    | name  | age
  1 | Ace   | 8
  2 | Bella | 5
  3 | Coco  | 13
 """
)
```

Or to a **stream connector**…

```
table_dogs = pw.csv.read("example-stream.csv")
```

Run it

Pathway: a code primer (2/4)

Table operations can be done like on dataframes.

Example: **filter**

```
table_dogs_young = table_dogs.filter(
    table_dogs.age <= 10
)  # table_dogs['age'] also works
pw.debug.compute_and_print(table_dogs_young)
```

```
            | name  | age
^2TMTFGY... | Ace   | 8
^YHZBTNY... | Bella | 5
```

[Run it](#)

Pathway: a code primer (3/4)

Joins, joins, and more joins.

```
table_dogs_owners = pw.debug.table_from_markdown(
    """
    | name  | owner
  1 | Ace   | Alice
  2 | Bella | Bob
  3 | Coco  | Alice
 """
)
```

```
table_dogs_full = table_dogs.join(
    table_dogs_owners, table_dogs.name == table_dogs_owners.name
).select(table_dogs.name, table_dogs.age, table_dogs_owners.owner)
pw.debug.compute_and_print(table_dogs_full)
```

```
            | name  | age | owner
^VJ3K9DF... | Ace   | 8   | Alice
^V1RPZW8... | Bella | 5   | Bob
^R0GE4WM... | Coco  | 13  | Alice
```

Run it

Pathway: a code primer (4/4)

Functions on tables

use **apply** for the easy cases (same row)

```
table_dogs_corrected = table_dogs.select(
    table_dogs.name, age=pw.apply((lambda x: x - 1), table_dogs["age"])
)
pw.debug.compute_and_print(table_dogs_corrected)
```

```
            | name  | age
^2TMTFGY... | Ace   | 7
^YHZBTNY... | Bella | 4
^SERVYWW... | Coco  | 12
```

Run it

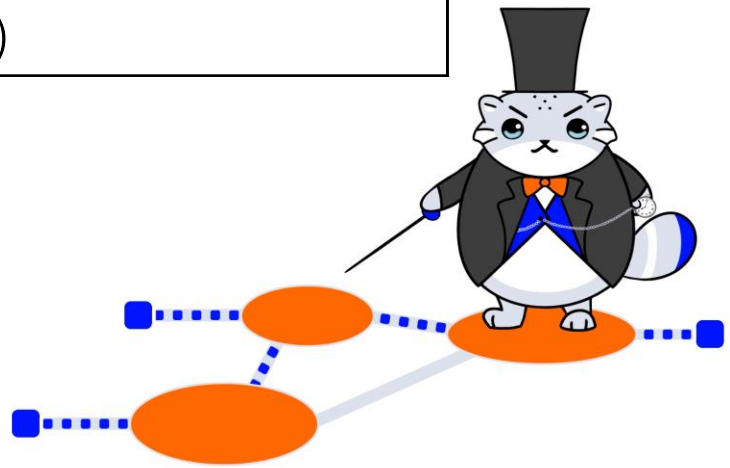
Harder cases: use **transformer class syntax** (to reference other rows or tables).

What are the main building blocks when programming in Pathway?

Transformers: they transform tables into tables.

Data Engineering **built-ins**	**Machine Learning "smart replacement"** **(library functions)**
.filter	Smart filter
.join	Fuzzy join / fuzzy matching
.groupby	Clustering Classification
sorted index	Item ranking (MLR)

To build your program in Pathway, build a computation flow graph (data pipeline) out of such operations.



Making your own transformers:

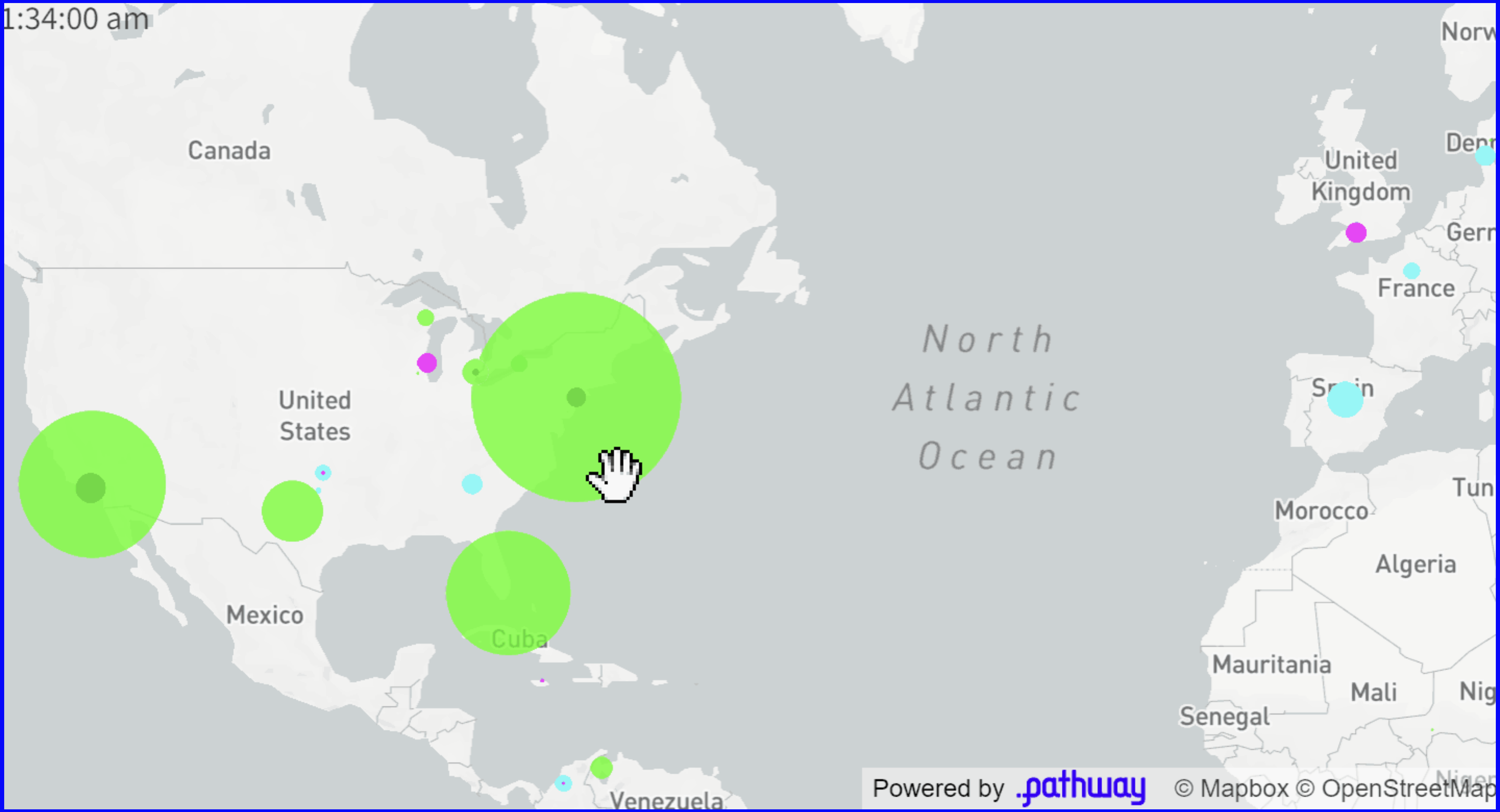
- You can write code using row **pointers** in Pathway (in Python class syntax)
- Write iterative and recursive **row-centric logic**: build and traverse lists, graphs,...
- You can run transformers in a **loop** (e.g.: iterate until convergence).
- **Be compositional**: you can build transformers out of other transformers.

A couple of tutorial notebooks you will find in our examples repo:

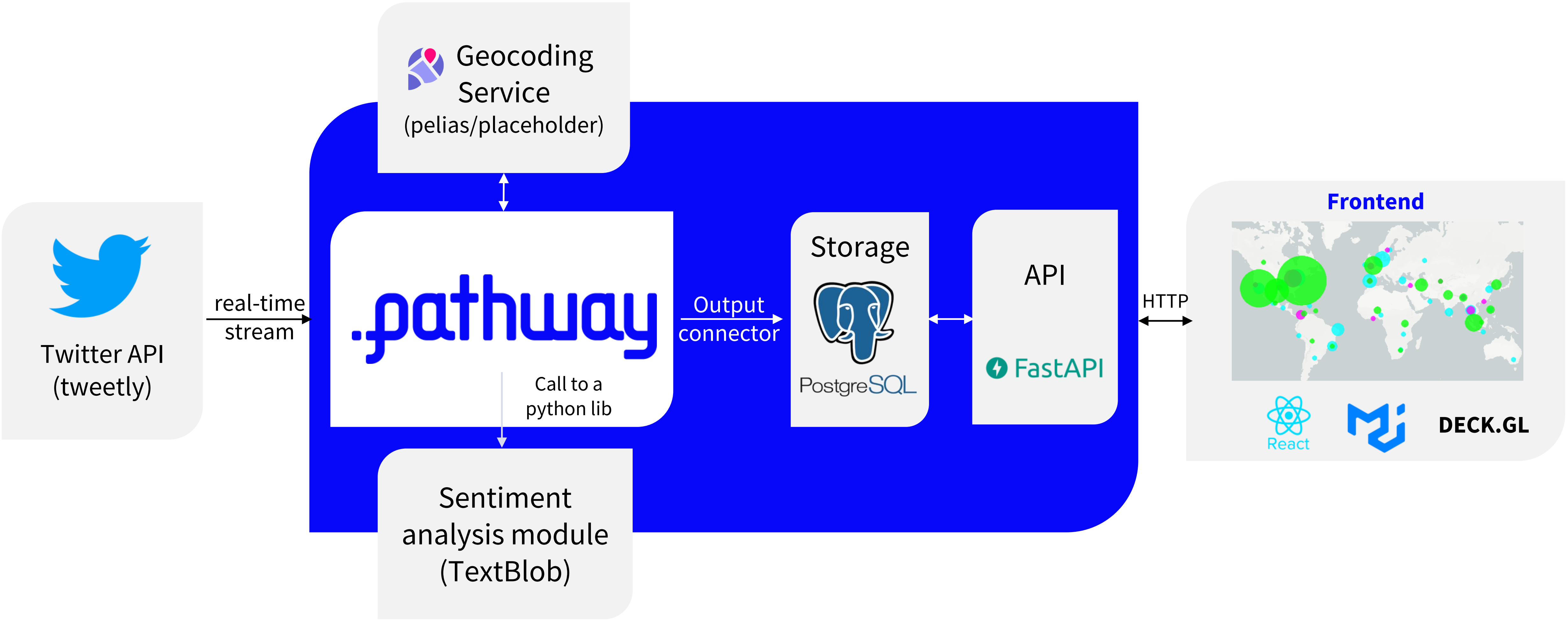
- Detecting suspicious user activity with Tumbling Window group-by
- Time between events in a multi-topic event stream
- Mining hidden user pair activity with Fuzzy Join
- Bellman-Ford Algorithm
- Computing PageRank

https://github.com/**pathwaycom**/pathway-examples

DEMO: Real time Twitter sentiment analysis app with Pathway



Application architecture



Get it on github: bit.ly/pathway-twitter

Run it with **docker-compose**

Twitter app logic: the pipeline in Pathway

Step 1: Preprocessing tweets ingested from tweepy

- **Filtering**: we are interested only in retweets and replies to other tweets.
- For each tweet, we **join** it with the data of its user and lookup the “location” field.
- We then obtain users’ coordinates by calling **placeholder** - the free coarse geocoder – through Pathway’s “apply” function in one line of code.

Twitter app logic: the pipeline in Pathway

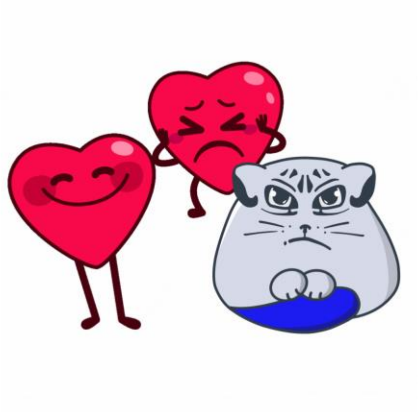
Step 2: Iterative geolocation cleaning with Pathway

Some Twitter users put weird locations like “*turn on notifications*” for place name.
To filter these out, we **use an iterative process**:

<u>REPEAT</u> UNTIL CONVERGENCE:

- For each user
  - compute “close_fraction” as the fraction of nearby retweets (<200km)
  - IF “close_fraction” < CUTOFF
    - make sure there is no other user in the same location, with its “close_fraction” > CUTOFF
    - filter out all tweets and retweets with this location

Twitter app logic: the pipeline in Pathway



Step 3: Sentiment analysis

- A number between [-1,1] is computed for each tweet with a one-liner: **calling the <u>TextBlob</u>** library through an "apply" in Pathway.

- The aggregation takes place in another line with a call to Pathway's "group-by".

Step 4: Computing influence

- First try: count the number of retweets: a one-line "group-by" aggregation in Pathway.

- Also taking into account the number of followers and the overall activity of the retweeting users gives us a fair "predictor" for the number of <u>upcoming retweets</u> - we can typically say **which tweets are likely to create a significant buzz before this actually happens**.

Main takeaway

What life has in store:

1. **Write your program**

```
my_model = pw.train_classifier (X_train, y_train)
my_model.predict (X_test)
```

2. **Run some experiments**

3. ~~It's working. Let's have a coffee ☕ .~~
   Handle live data updates:

   - Redo all the code logic in a streaming architecture.
   - Prepare to handle every imaginable (and unimaginable) data input change event ever, in conjunction with your classification & prediction logic 🙃.

Things are easier when you have pw's:

1. Write your program

```
my_model = pw.train_classifier (X_train, y_train)
pw.predict (my_model, X_test)
```

2. Run some experiments

3. Let Pathway handle live data updates for you.

4. It’s working. Let’s have a coffee ☕ .

Happy to have you in the **Pathway** community!

- **Join us** at https://pathway.com/developers, we are around on Discord.

- Run all **examples** from this talk directly from https://github.com/pathwaycom/pathway-examples/

Backup slides:
* Some caveats & questions
* Further reading not related to Pathway

[BACKUP] Caveats & Questions:

Is it possible to make the presented approach serverless?
[No way, stateless joins are provably impossible.]

If time actually IS a feature in my data, how do I model it? I’m doing time series data.
[Pick the most relevant notions of time as a feature. Happy to discuss this further.]

How do I do data schema updates?
[A tough one, but we have it on our roadmap, the blueprints are there.]

How can storage and persistence be handled?
[We have working setups with enterprise clients - another great topic for a chat!]

[BACKUP] Further reading not related to Pathway

- Twitter - more advanced metrics for evaluating possible impact of a tweet:
  a nice paper: Prediction of Retweet Cascade Size over Time
  a nice video: Analyzing Big Data with Twitter: Stan Nikolov on Information Diffusion at Twitter

- A nice overview of cool frameworks capable of working with data updates without a full recompute: Incremental computing - Wikipedia

- Batch data pipelines & orchestration using Python:
  Airflow, Dagster, Luigi, now dbt,...

[BACKUP] Reactive Data Processing

Where it helps:

Incoming live events data

Interactive work with the user

Reactive data processing

What it is meant to provide:

Scalable distributed runtime

Asynchronous dataflow

Incremental computation

Consistency of results