

# Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang and Chen Li



UCIRVINE

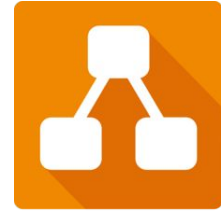
# Tutorial Outline

1. Overview of collaborative data analytics
2. Challenges and solutions:
  - a. Shared workflow editing
  - b. Shared workflow execution
  - c. Interacting with runtime execution
  - d. Runtime co-debugging on Python UDFs
3. Open challenges

# Tutorial Outline

1. → Overview of collaborative data analytics (Chen)
2. Challenges and solutions:
  - a. Shared workflow editing (Chen)
  - b. Shared workflow execution (Zuozhi)
  - c. Interacting with runtime execution (Zuozhi)
  - d. Runtime co-debugging on Python UDFs (Zuozhi)
3. Open challenges (Chen and Zuozhi)

# Popularity of Collaboration Cloud Services



**draw.io**

Benefits:

- ✓ Cloud services
- ✓ Shared editing
- ✓ Version control
- ✓ Sharing

# Python Notebooks

- Support real-time collaborations
- For programmers

The screenshot shows a Deepnote notebook titled "Predicting customer churn". The notebook is running on a Snowflake warehouse. The code cell contains a SQL query that selects columns from a table, filtered by DSL and Fiber optic services. The query is executed, and the results are displayed in a table.

```
column_names=['monthlycharges', 'tenuremonths', 'contract', 'churnvalue']  
where_filter=['DSL', 'Fiber optic']
```

```
SELECT {{ ' ', 'join(column_names) | sqlsafe }}  
FROM demo_db.demo_schema.telco_dataset  
SAMPLE(42)  
WHERE internetervice in {{ where_filter | inclause }}
```

monthlycharges	tenuremonths int	Contract object	churnvalue float64
			1



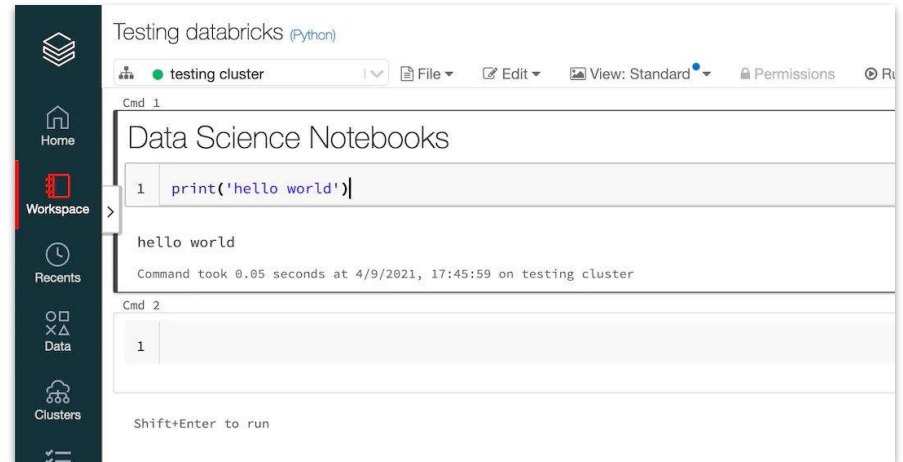
# Big Data Processing Systems

Efficient and scalable



# Big Data Processing Systems

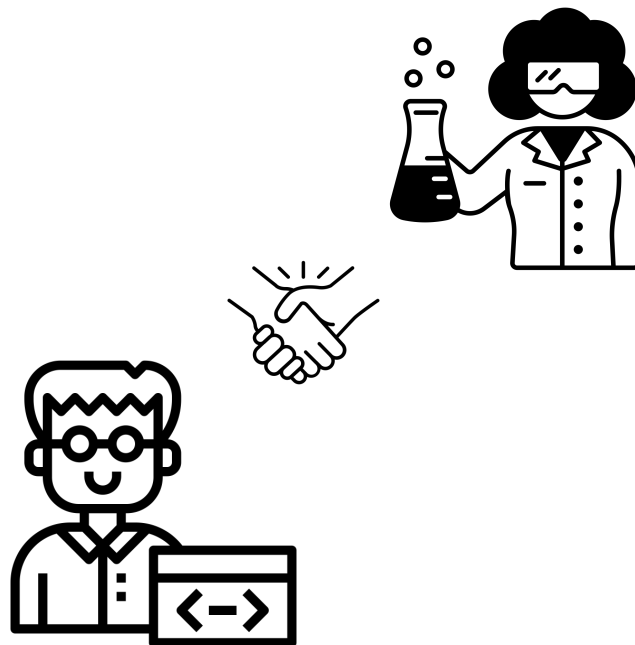
- Start to support collaborations
- For SQL experts



Databricks notebooks

# Need of Multi-disciplinary Collaborations

- Domain experts:
  - Rich domain knowledge
  - Limited IT/coding skills
- IT experts
  - Limited domain knowledge
  - Strong coding skills





# Workflow Systems

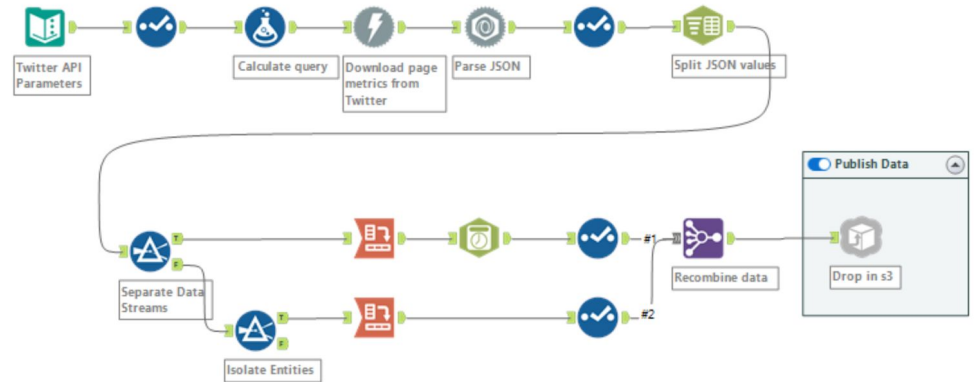


Pros:

- Easy-to-use interface
- No coding required

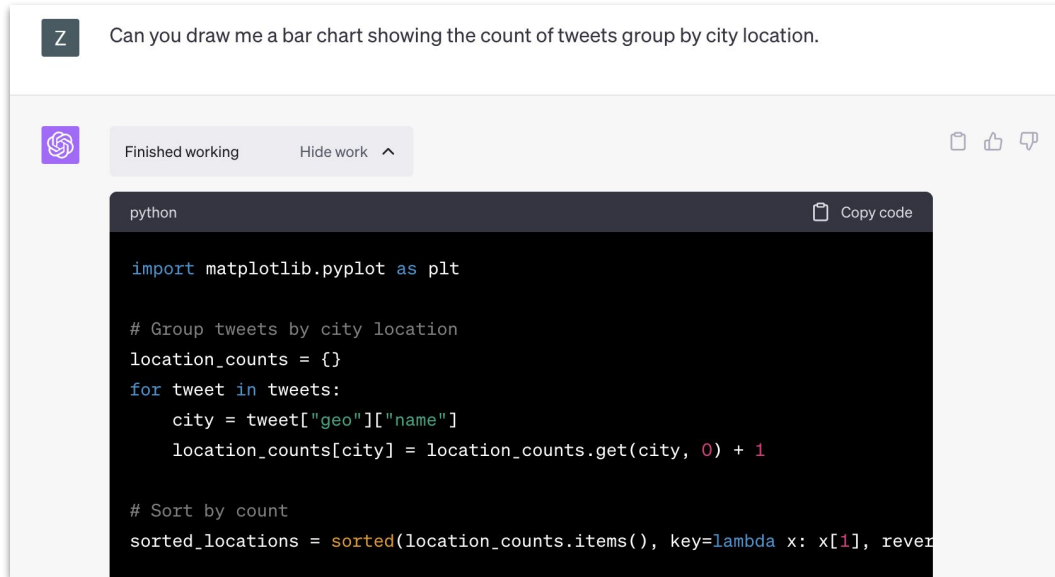
Cons:

- Pre-cloud architecture
- No collaboration features
- Limited scalability



# ChatGPT Code Interpreter

- Easy to use, no programming
- Integratable into a workflow system



The screenshot shows the ChatGPT Code Interpreter interface. At the top, a user prompt asks: "Can you draw me a bar chart showing the count of tweets group by city location." Below the prompt, the AI response is displayed in a code editor window titled "python". The code is as follows:

```
python
import matplotlib.pyplot as plt

# Group tweets by city location
location_counts = {}
for tweet in tweets:
    city = tweet["geo"]["name"]
    location_counts[city] = location_counts.get(city, 0) + 1

# Sort by count
sorted_locations = sorted(location_counts.items(), key=lambda x: x[1], reverse=True)
```

# System Requirements for Collaborations

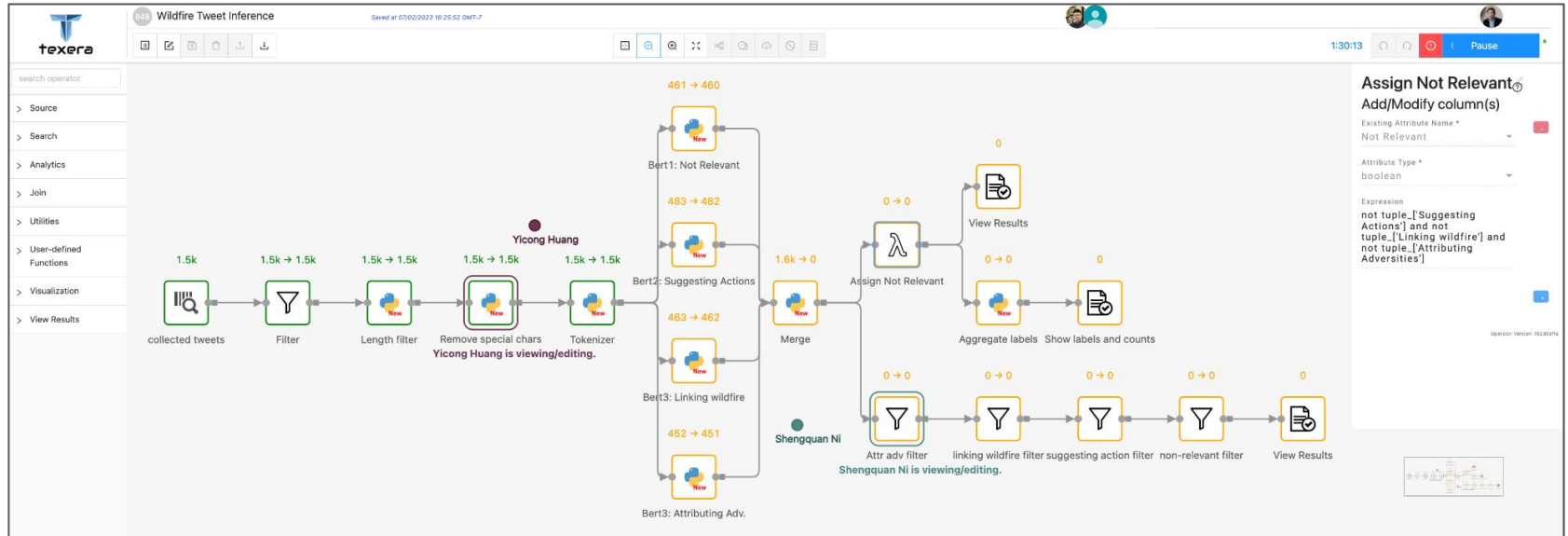
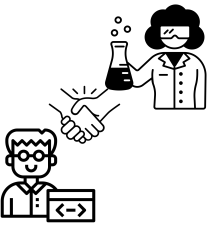
1. GUI-based workflows for non-IT people
2. Collaborative editing
3. Interactions during runtime
4. Supporting multiple languages: Python, R, ...
5. Supporting machine learning (training, inference, ...)
6. Scalable!

# A system for collaborative data analytics

- Started in 2016; open source
- Used by many research projects
- Powerful features:
  - Cloud services
  - Version control
  - Shared editing
  - Commenting
  - Sharing
  - ...



# Demo!



# Collaboration Feature: Version Control



Version Diff

Version#	Timestamp
15	08/17/2023 02:57:54 GMT-7
14	08/17/2023 02:57:53 GMT-7
13	08/17/2023 02:57:52 GMT-7
12	08/17/2023 02:57:51 GMT-7
11	08/17/2023 02:57:50 GMT-7
10	08/17/2023 02:57:49 GMT-7
9	08/17/2023 02:57:47 GMT-7

Historical Version List

# Collaboration Feature: Resource Sharing

Share this file with others ×

Target User

\* E-mail:

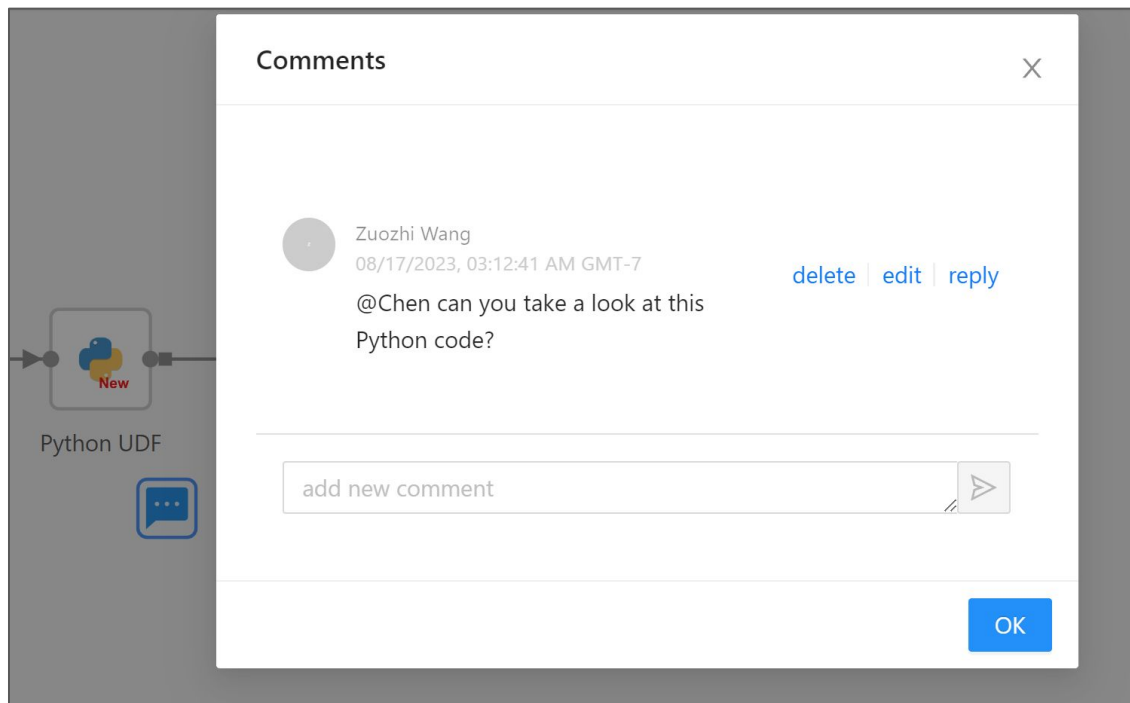
Share

Access Level:  ▾

Access:

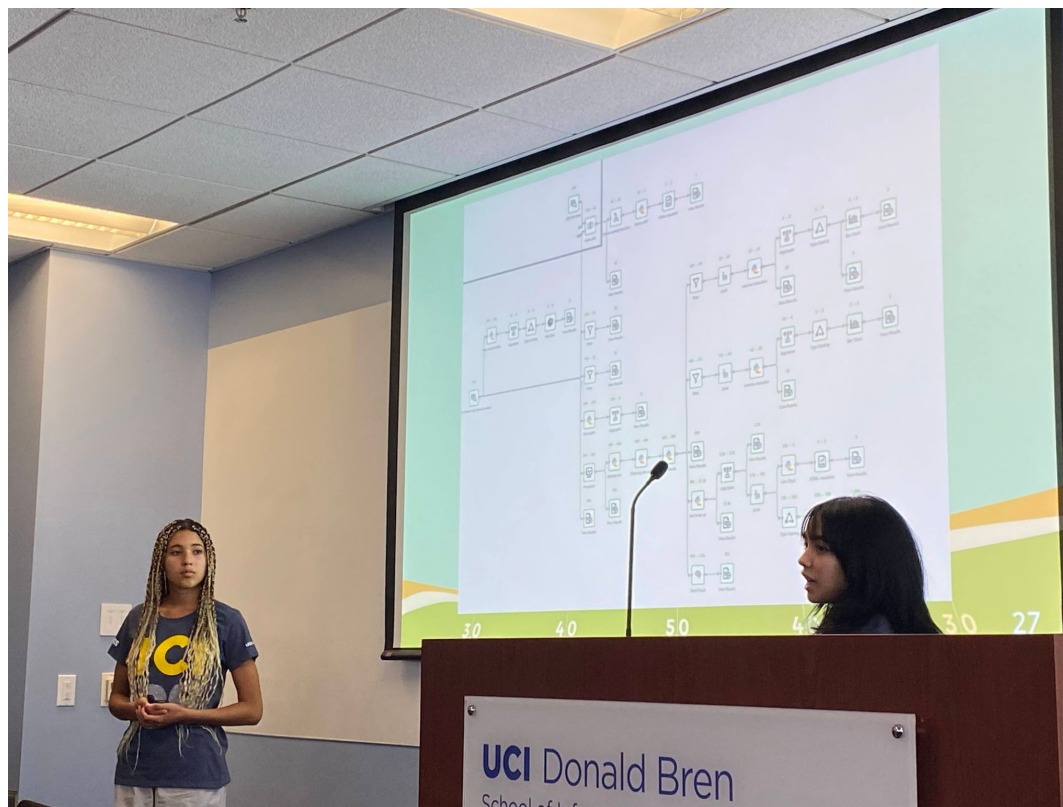
- zuozhiw@gmail.com

# Collaboration feature: commenting





# High School Students Using Texera!

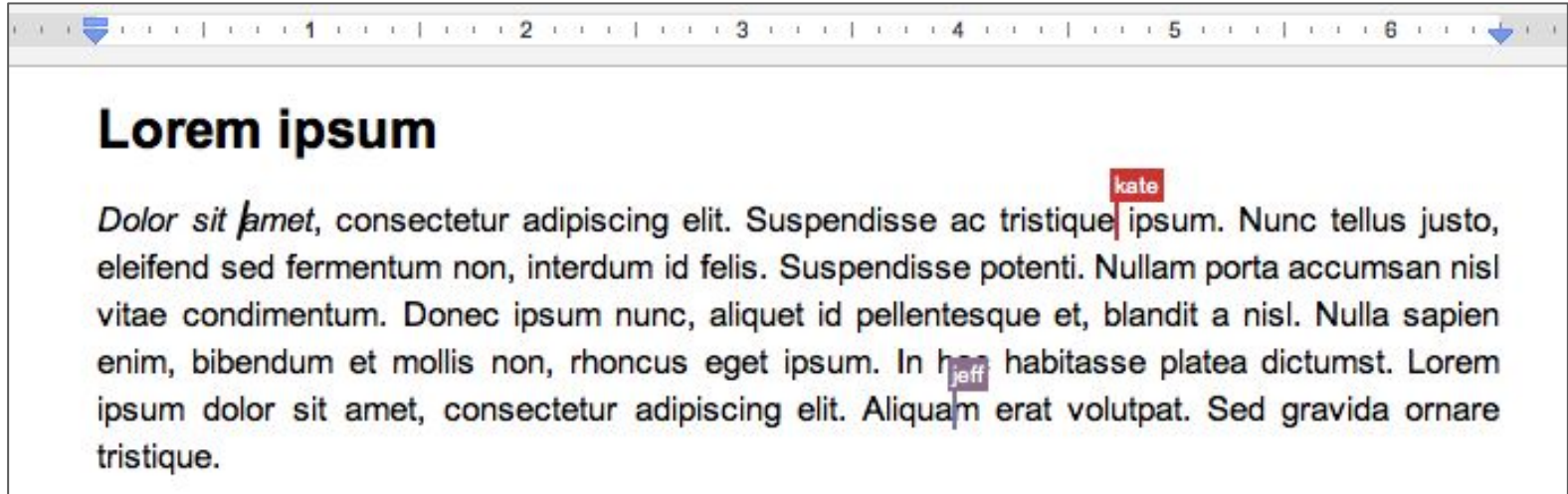


Building a Collaborative Data Analytics System. Zuozhi Wang and Chen Li

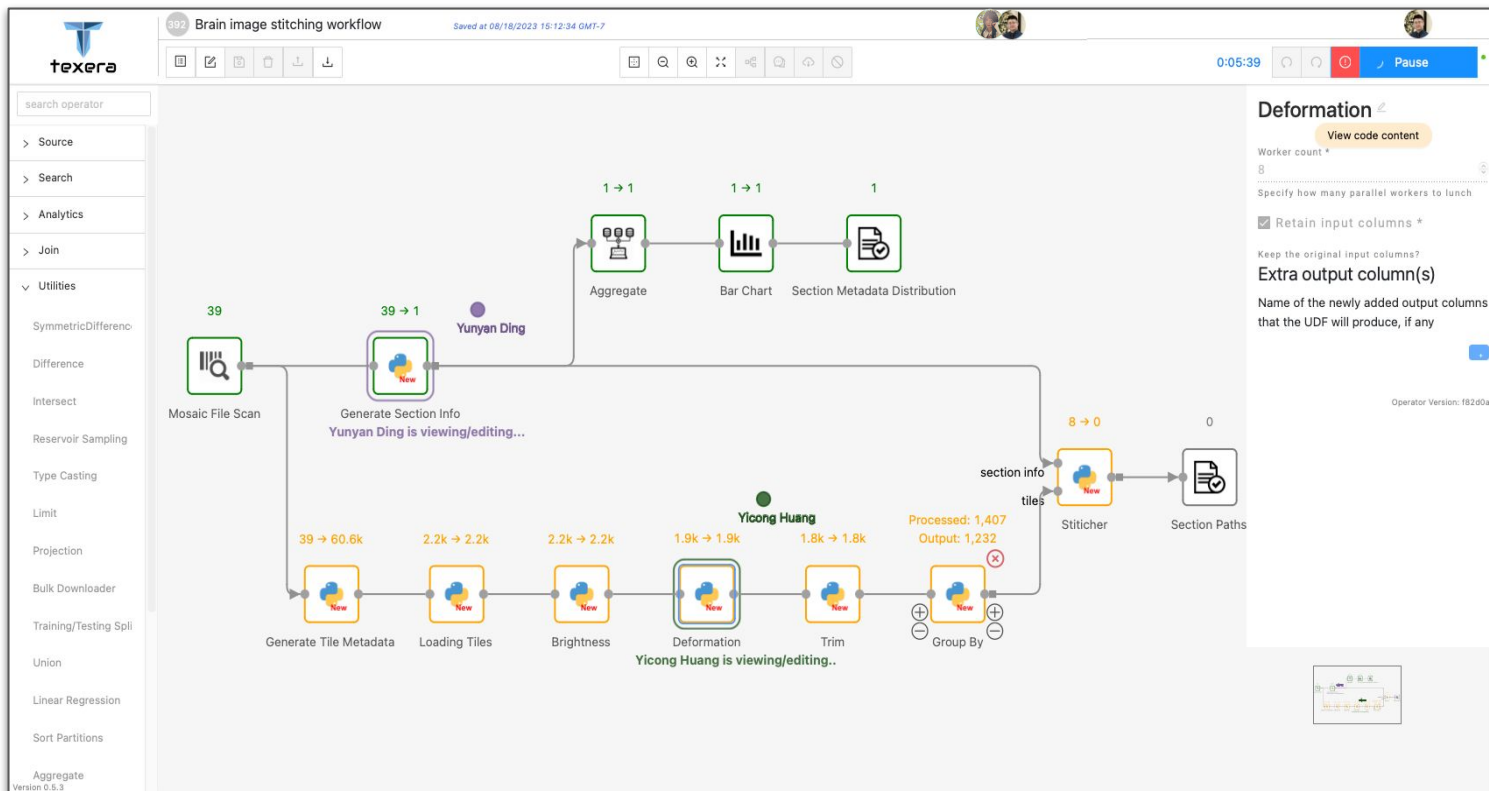
# Tutorial Outline

1. Overview of collaborative data analytics (Chen)
2. Challenges and solutions:
  - a. → Shared workflow editing (Chen)
  - b. Shared workflow execution (Zuozhi)
  - c. Interacting with runtime execution (Zuozhi)
  - d. Runtime co-debugging on Python UDFs (Zuozhi)
3. Open challenges (Chen and Zuozhi)

# Goal: enabling shared editing similar to Google Docs



# Workflow shared editing



# Shared Editing



Alice

U C I



Bob

U C I

# Two users doing concurrent edits



U C X I



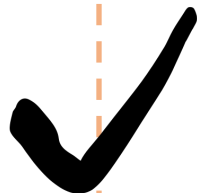
U C Y I

# Shared Editing: Good (same result)



U C X Y I

U C X Y I

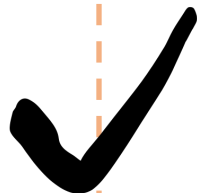


## Shared Editing: Also good (same result)



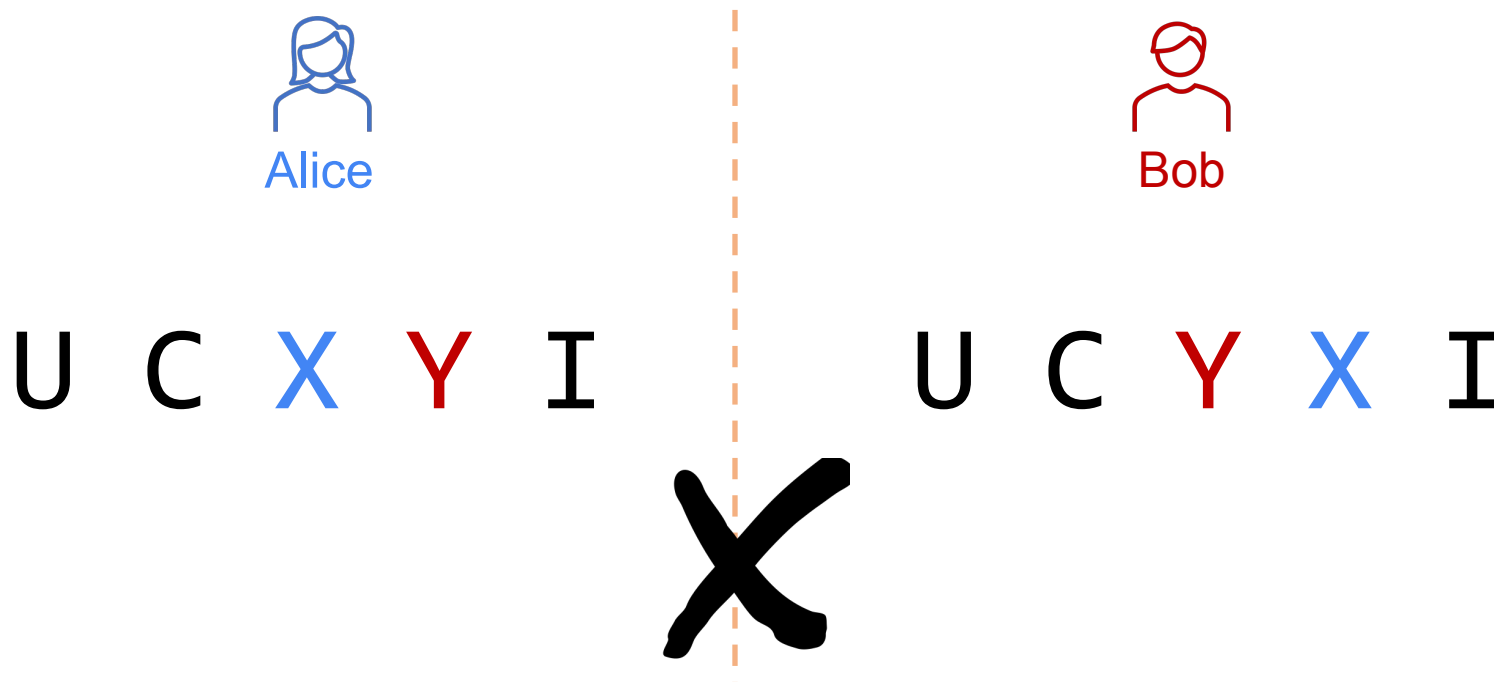
U C Y X I

U C Y X I





# Shared Editing: Bad (different results)



# Shared Editing: How to determine an order

1st method: Operational Transformation (OT)

- Using a central server

☰ **Central Server**

Document:

**UCI**

Version: R0

Operations:

**Alice**

**UCI**

**Bob**

**UCI**

☰ **Central Server**

Document: **UCI** Version: R0

Operations:

R0: insert("X", pos=2)

Alice

UCXI

R0: insert("Y", pos=2)

Bob

UCYI

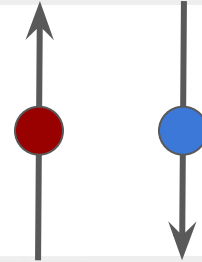
☰ Central Server

Document:

UCXI

Operations: ●

Received Alice's request  
Update R0 to R1



Propagate  
Alice's Edit

Alice

UCXI

Bob

UCXYI

☰ Central Server

Document:

UCXYI

Operations: ● ●

Received Bob's request  
on R0, insert("Y", pos=2)



on R1: insert("Y", pos=3)



Alice

UCXYI

Bob

UCXYI

# Final Result

☰ **Central Server**

Document:

UCXYI

Operations: ● ●

**Alice**

UCXYI

**Bob**

UCXYI

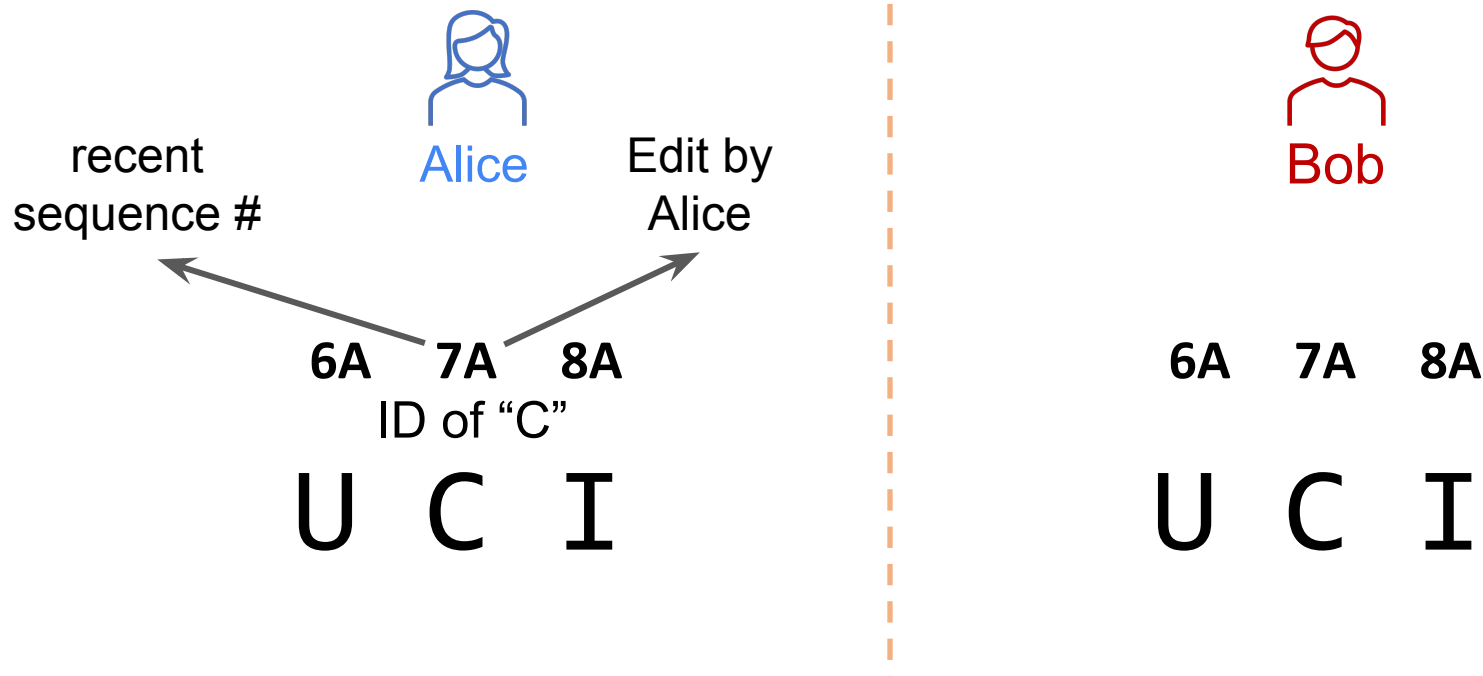
# Shared Editing: How to Determine Order

2nd method: Conflict-free Replicated Data Type (CRDT)

- Peer-to-peer communication
- Everyone follows the same policy.



# CRDT (Conflict-free Replicated Data Type)



# CRDT: concurrent edits



Alice

9A: { 7A, ins("X") }  
new id    pos    operation

6A	7A	9A	8A
U	C	X	I

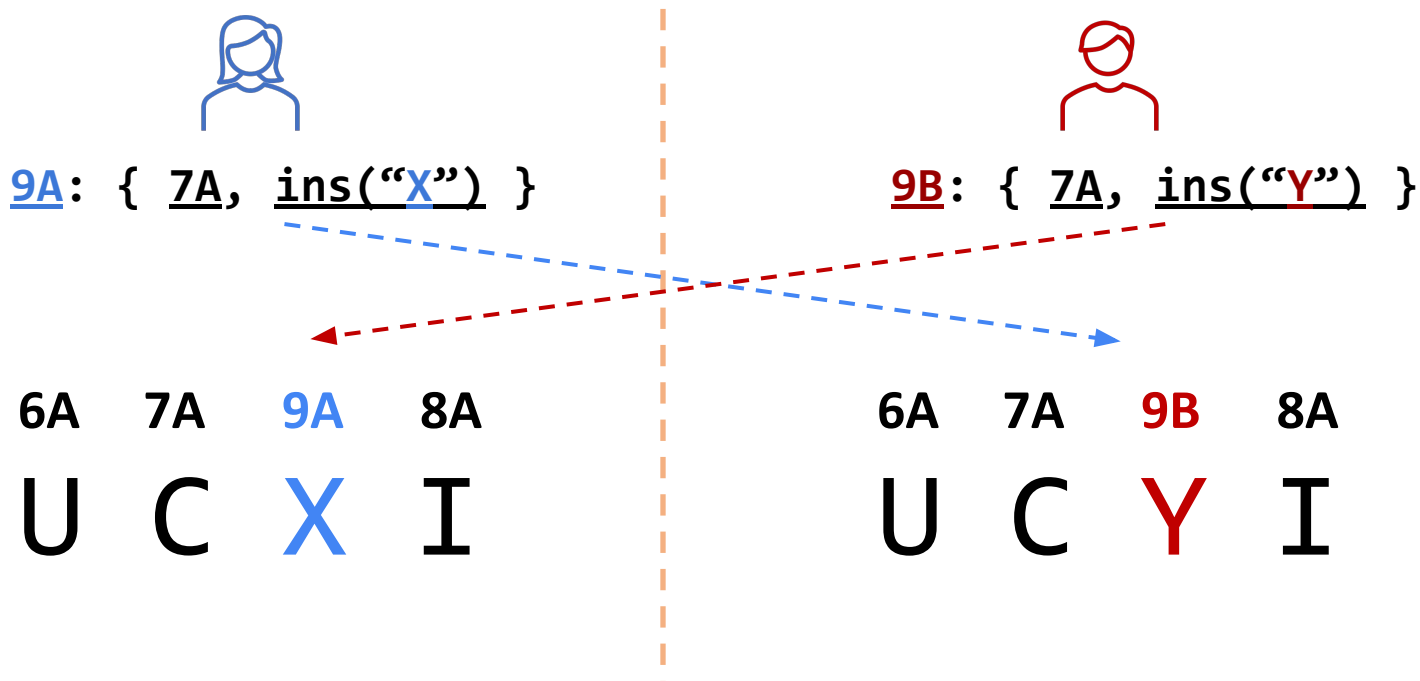


Bob

9B: { 7A, ins("Y") }  
new id    pos    operation

6A	7A	9B	8A
U	C	Y	I

# CRDT: transmitting edits to other users



# CRDT: independently comparing edits



Alice

9B: { 7A, ins("Y") }

9A > 9B

6A	7A	9A	8A
U	C	X	I

A red dashed arrow points from the '9A' label to the 'X' character.



Bob

9A: { 7A, ins("X") }

9A > 9B

6A	7A	9B	8A
U	C	Y	I

A blue dashed arrow points from the '9B' label to the 'Y' character.

# CRDT: following same policy to decide



Alice



Bob

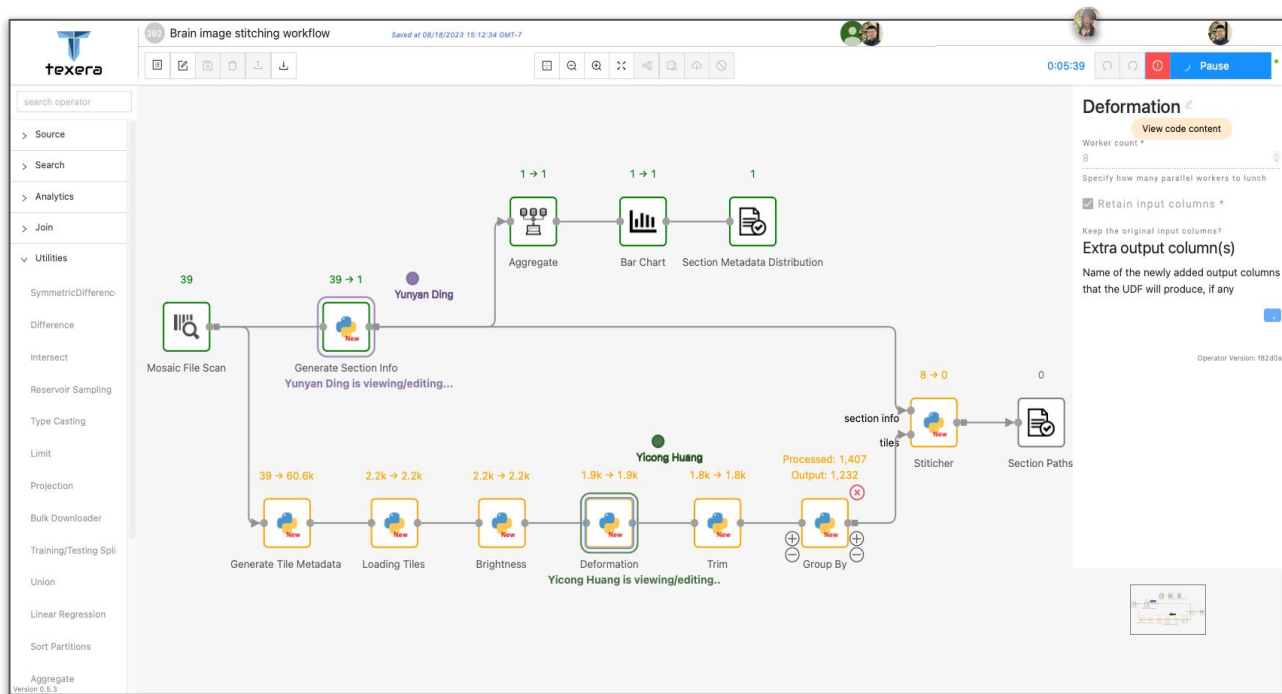
6A 7A 9A 9B 8A  
U C X Y I

6A 7A 9A 9B 8A  
U C X Y I

# Comparison of the Two Methods

	<b>Central server</b>	<b>Implementation complexity</b>	<b>Performance</b>	<b>Open-source ecosystem</b>
OT	Yes	Complex	Good	Less rich
CRDT	No	Simple	Acceptable	Rich

# Workflow Shared Editing



CRDT “documents” are:

- Workflow DAG
- Properties

# Adding Two Operators Concurrently: no conflicts

Alice



CSV File Scan 1

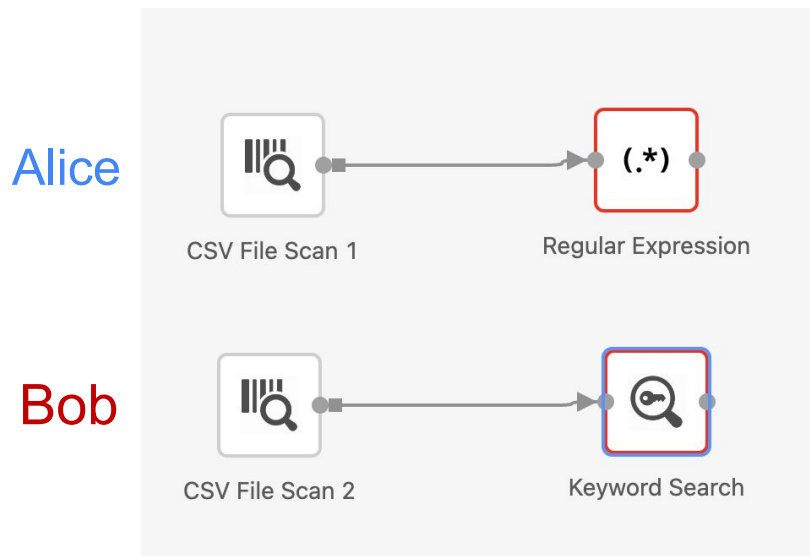
Bob



CSV File Scan 2



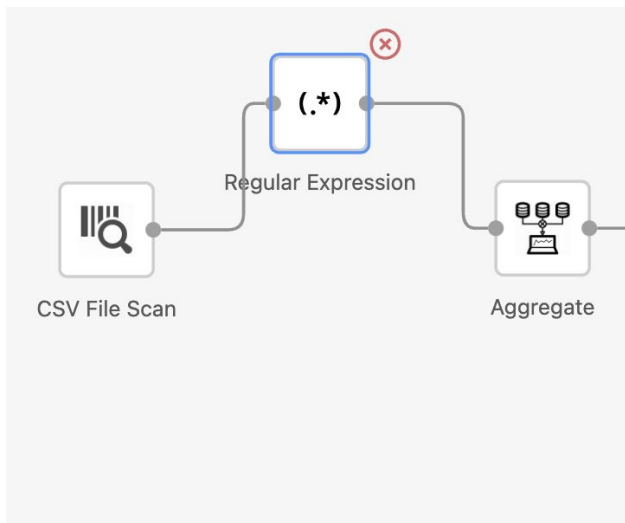
# Adding Two Operators Concurrently: no conflicts



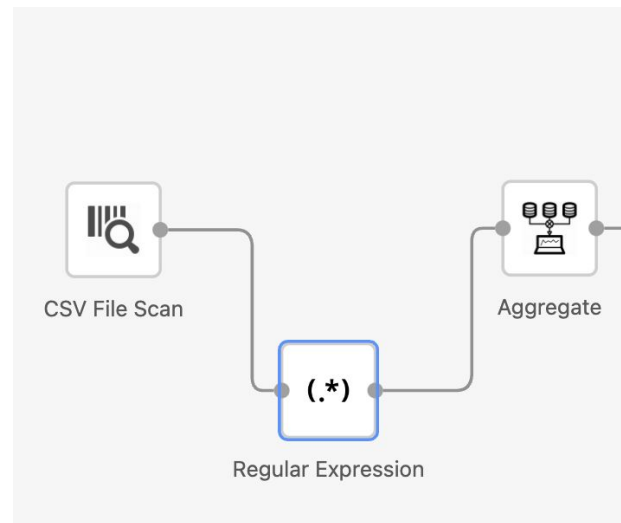
# Moving Operators to Different Positions: conflicts

CRDT resolves it based on user ID

Alice



Bob



# Changing Operator's Properties: conflicts

CRDT resolves it based on user ID

## Regular Expression

Case Insensitive \*

regex match is case sensitive  
Attribute \*

text ▼

column to search regex on

Regex \*

foo|

regular expression

Alice

## Regular Expression

Case Insensitive \*

regex match is case sensitive  
Attribute \*

text ▼

column to search regex on

Regex \*

bar

regular expression

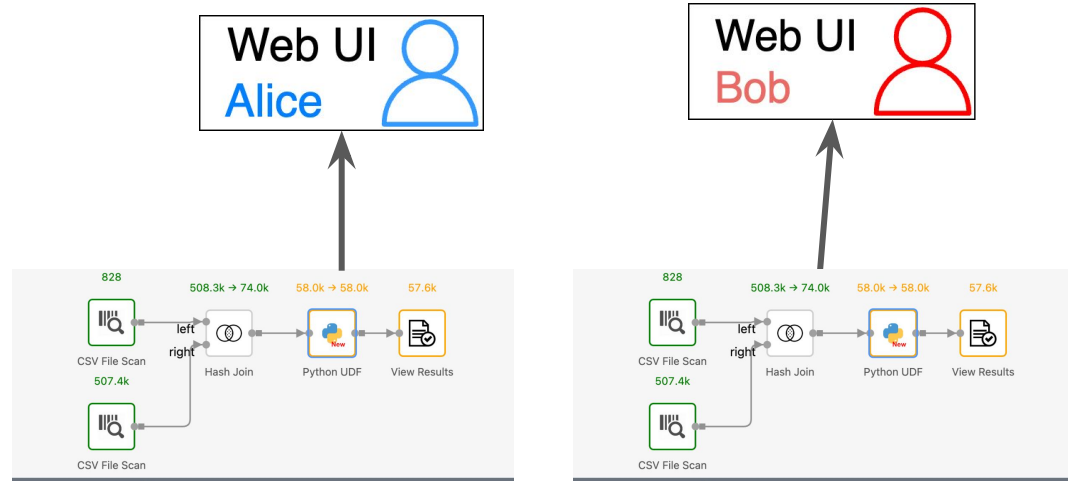
Bob

# Tutorial Outline

1. Overview of collaborative data analytics (Chen)
2. Challenges and solutions:
  - a. → Shared workflow editing (Chen)
  - b. → Shared workflow execution (Chen and Zuozhi)
  - c. Interacting with runtime execution (Zuozhi)
  - d. Runtime co-debugging on Python UDFs (Zuozhi)
3. Open challenges (Chen and Zuozhi)

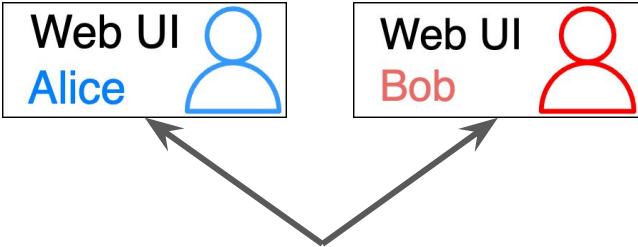
# Need of Shared Execution

If users have their own executions, then collaboration is not possible.

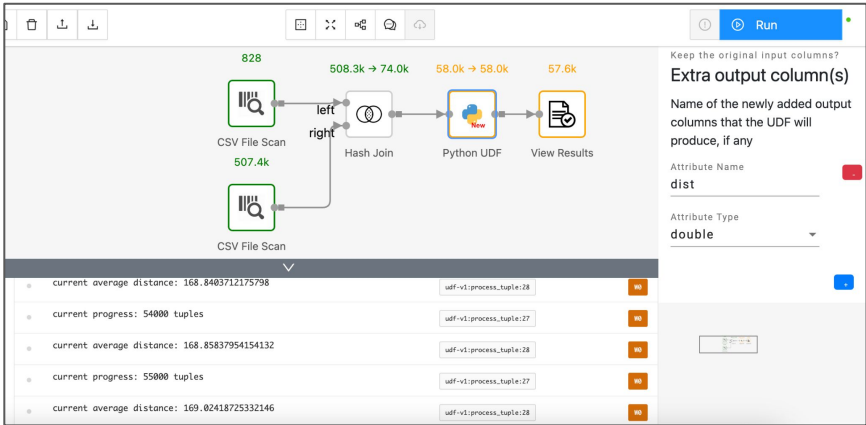


# Benefits of Shared Execution

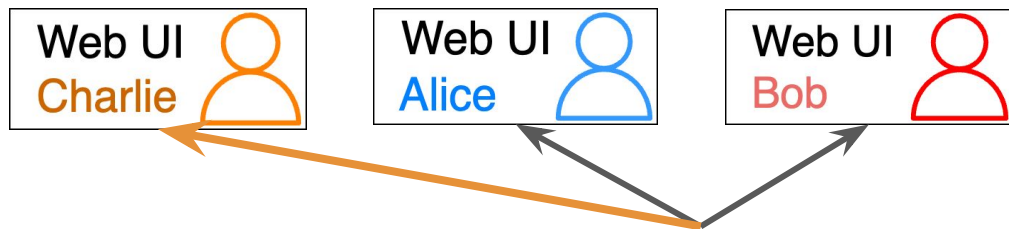
- View same results
- Shared control
- Co-debugging
- .....



Shared Execution State

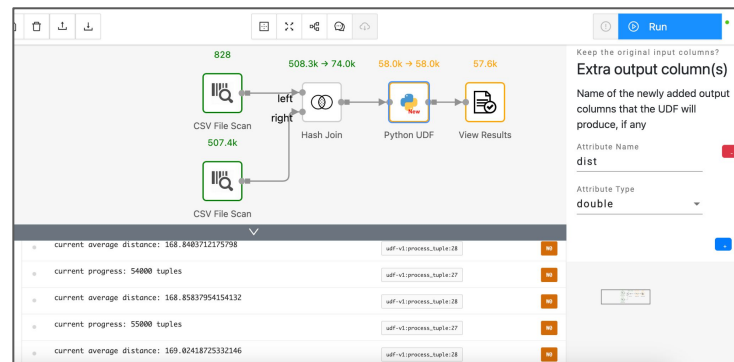


# Benefits of Shared Execution (cont.)



Shared Execution State

Invite new collaborators at runtime



# Shared Execution: Demo

The screenshot displays a data pipeline interface with the following components:

- Top Toolbar:** Includes icons for trash, upload, download, grid, zoom, refresh, chat, and a blue 'Run' button.
- Main Workspace:** Shows a workflow with four nodes:
  - CSV File Scan (top):** 828 rows, 507.4k size.
  - CSV File Scan (bottom):** 507.4k size.
  - Hash Join:** Receives 'left' and 'right' inputs. Statistics: 508.3k → 74.0k.
  - Python UDF:** Labeled 'New'. Statistics: 58.0k → 58.0k.
  - View Results:** 57.6k rows.
- Bottom Console:** A log of execution events for the Python UDF:
  - current average distance: 168.8403712175798 (udf-v1:process\_tuple:28)
  - current progress: 54000 tuples (udf-v1:process\_tuple:27)
  - current average distance: 168.85837954154132 (udf-v1:process\_tuple:28)
  - current progress: 55000 tuples (udf-v1:process\_tuple:27)
  - current average distance: 169.02418725332146 (udf-v1:process\_tuple:28)
- Right Panel:** Configuration for the Python UDF:
  - Keep the original input columns? (checkbox)
  - Extra output column(s) (text input)
  - Name of the newly added output columns that the UDF will produce, if any (text input)
  - Attribute Name: `dist` (text input)
  - Attribute Type: `double` (dropdown menu)



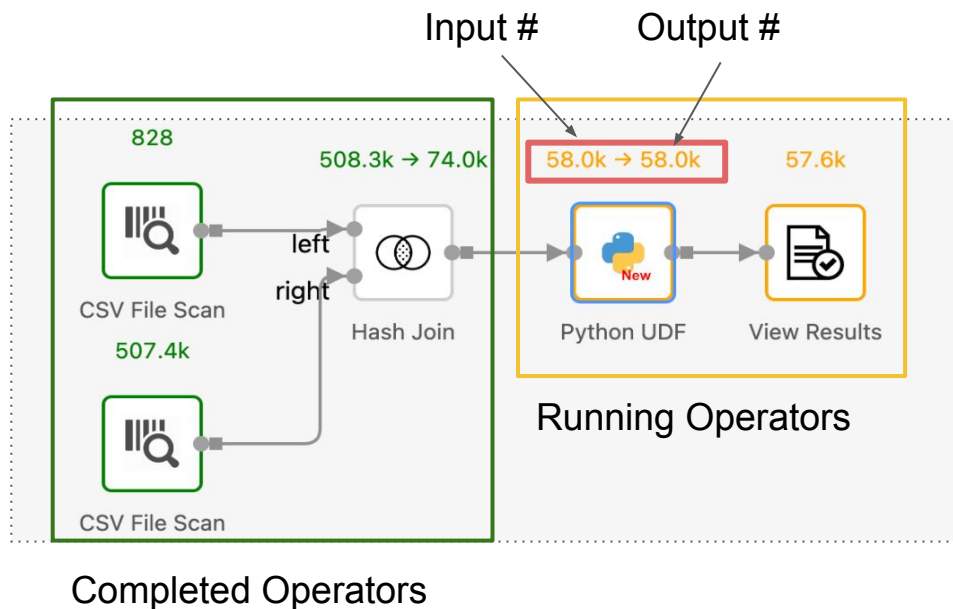
# Workflow Status: Running/Paused/Failed/Completed

The screenshot displays a workflow management interface. At the top, there are icons for trash, upload, and download. Below these are icons for a grid, zoom, refresh, chat, and share. The main workflow area shows a sequence of steps: two 'CSV File Scan' nodes (one with 828 records, the other with 507.4k), a 'Hash Join' node (508.3k → 74.0k), a 'Python UDF' node (58.0k → 58.0k), and a 'View Results' node (57.6k). The 'Python UDF' node is highlighted with a blue border and a 'New' label. A red box highlights the 'Run' button in the top right corner. Below the workflow, there is a table with columns for status, progress, and actions.

Status	Progress	Action
●	current average distance: 168.8403712175798	udf-v1:process_tuple:28 <span>WO</span>
●	current progress: 54000 tuples	udf-v1:process_tuple:27 <span>WO</span>
●	current average distance: 168.85837954154132	udf-v1:process_tuple:28 <span>WO</span>
●	current progress: 55000 tuples	udf-v1:process_tuple:27 <span>WO</span>
●	current average distance: 169.02418725332146	udf-v1:process_tuple:28 <span>WO</span>

Keep the original input columns?  
Extra output column(s)  
Name of the newly added output columns that the UDF will produce, if any  
Attribute Name **dist**  
Attribute Type **double**

# Operator Status: Input/Output Tuple Count



# Execution Results

The screenshot shows a data pipeline execution interface. The pipeline consists of four stages: CSV File Scan (828 rows), Hash Join (508.3k → 74.0k rows), Python UDF (58.0k → 58.0k rows), and View Results (57.6k rows). The Python UDF stage is marked as 'New'. The View Results stage displays a table of results. A red box highlights the table, which has 8 columns: id, long\_low, lat\_low, long\_high, lat\_high, id#@1, and an unlabeled column. The table contains 6 rows of data.

id	long_low	lat_low	long_high	lat_high	id#@1	
3b77caf94bfc81fe	-118.668404	33.704538	-118.155409	34.337041	1476765525984174000	Yes, let @FoxNews use your footage to add
cea1f774c62bb6fc	-119.583292	36.677705	-119.538486	36.721241	1475578758006337500	@ChaoticCoochiie @PFleeceman If you're a
3b77caf94bfc81fe	-118.668404	33.704538	-118.155409	34.337041	1475586396811264000	🇺🇸🇺🇸🇺🇸 We are collaborating with @carlosfig
a592bd6ceb1319f7	-117.282538	32.53962	-116.9274403	33.0804044	1475976318529388500	@Greenpeace 2.4 degrees in temperature r
3b77caf94bfc81fe	-118.668404	33.704538	-118.155409	34.337041	1476346495280443400	Articles like this (along with music news) is

# User Interaction History

The screenshot displays a data analytics workflow with the following components and data:

- CSV File Scan** (828 rows, 507.4k size)
- CSV File Scan** (508.3k → 74.0k size)
- Hash Join** (58.0k → 58.0k size)
- Python UDF** (57.6k size)
- View Results**

The Python UDF configuration panel shows the following settings:

- Keep the original input columns?
- Extra output column(s): **dist**
- Attribute Name: **dist**
- Attribute Type: **double**

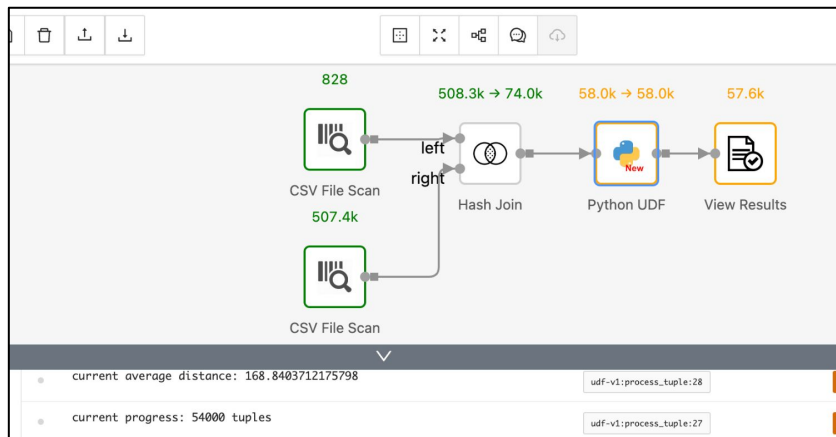
The User Interaction History panel (highlighted in red) contains the following entries:

- `self._context.operator_manager.operator.average_distance` (USER-1)
- `155.36079087094146` ((Pdb))
- `self._context.tuple_processing_manager.current_input_tuple` (USER-1)
- `Tuple['id': 'e1e35d357ceefa52', 'long_low': -118.4019312, 'lat_low': -300.0, 'long_high': -118.352695, 'lat_high': -300.0, 'id#@1': 1471642336094396417, 'text': 'My number 1 motivation to stop climate change is so that I can make my son Cesar so he can be CP3']` ((Pdb))

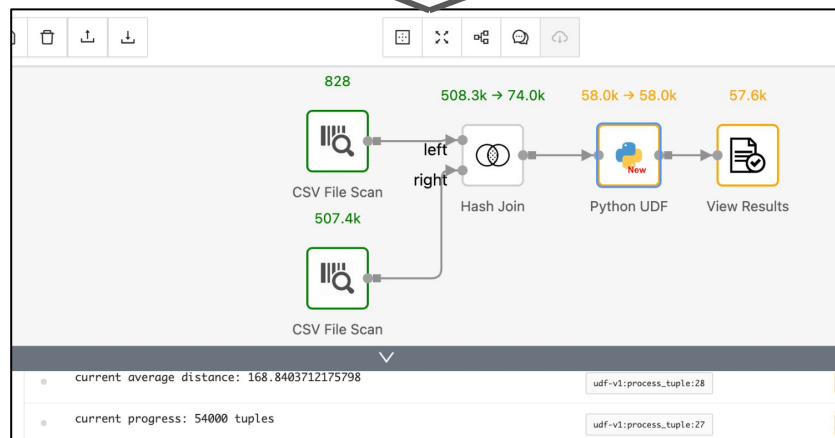
# Shared Execution: How to Share?



## Execution State

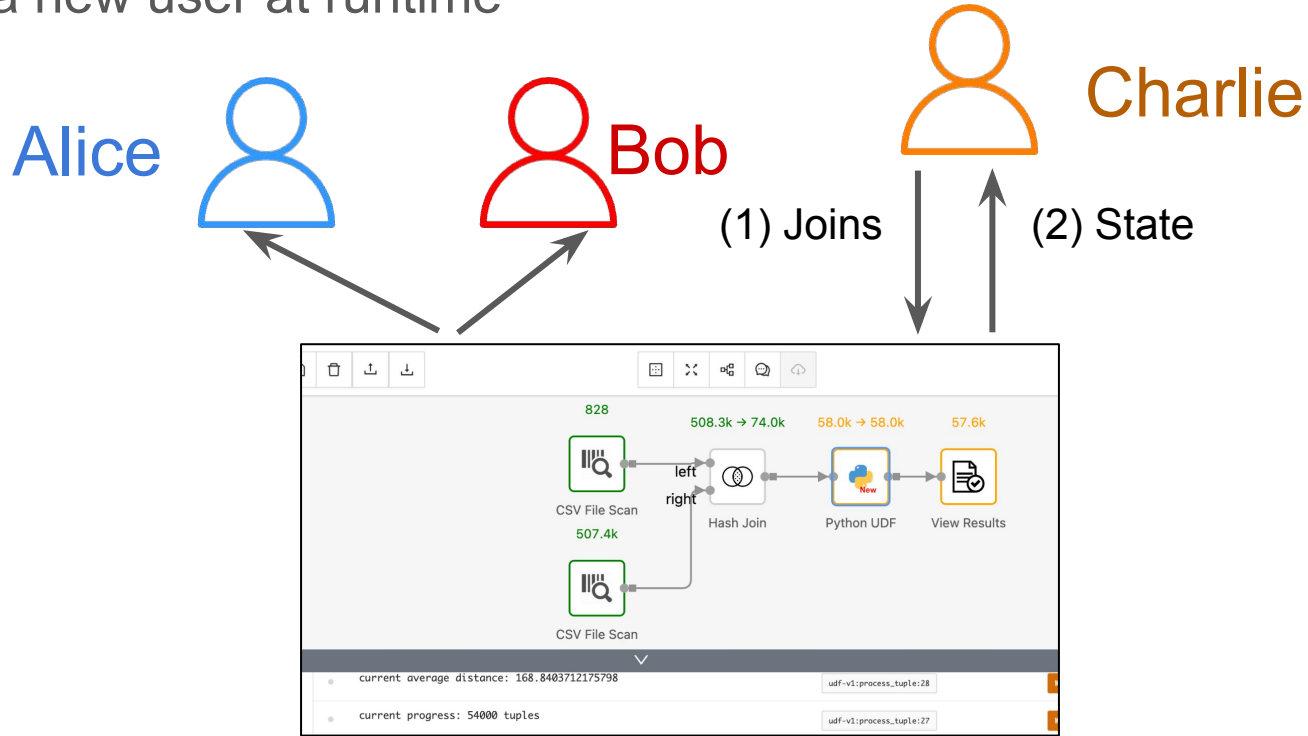


# Approach 1: Periodically Broadcast State to All Frontends



# Approach 1: Periodically Broadcast State to All Frontends

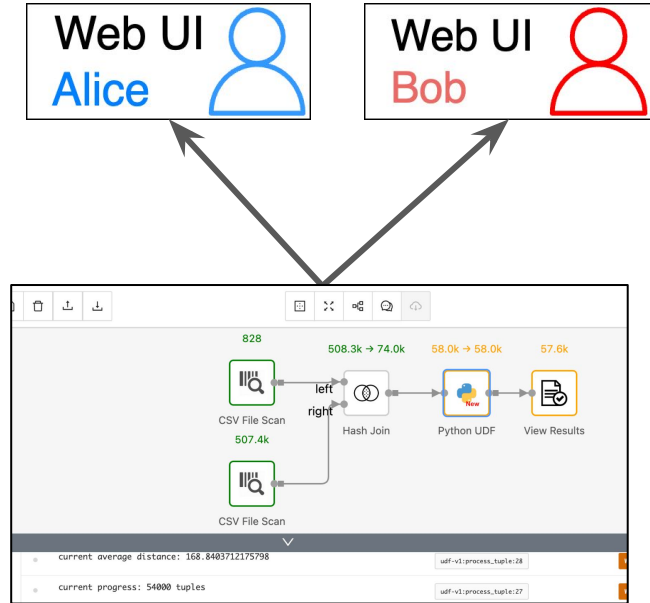
Adding a new user at runtime



# Problems with Approach 1

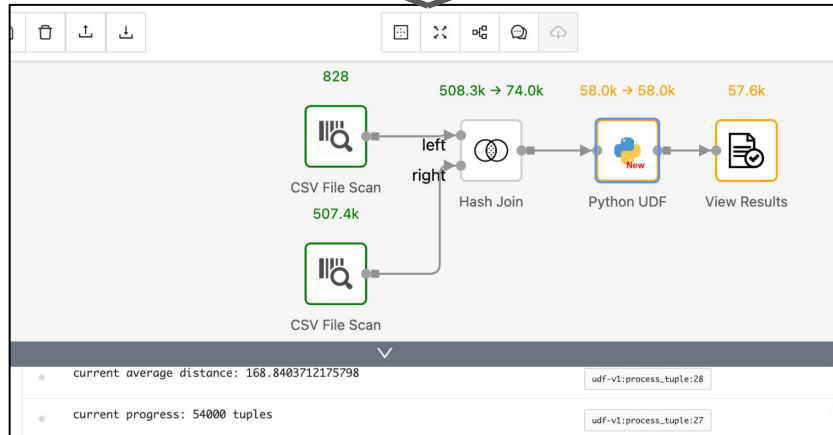
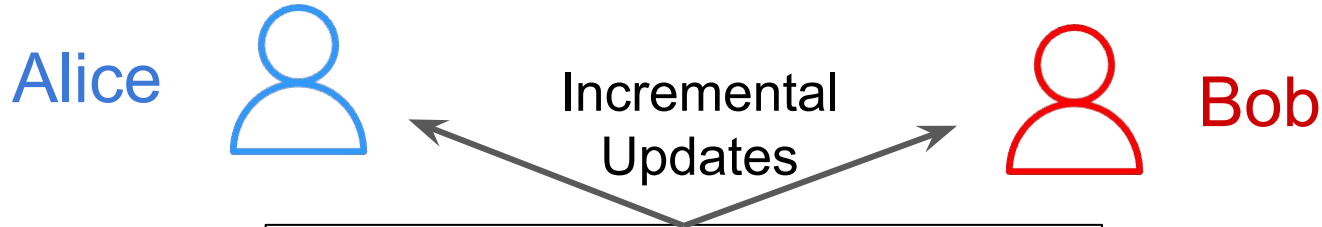


- Interaction history: growing
- Execution results: large



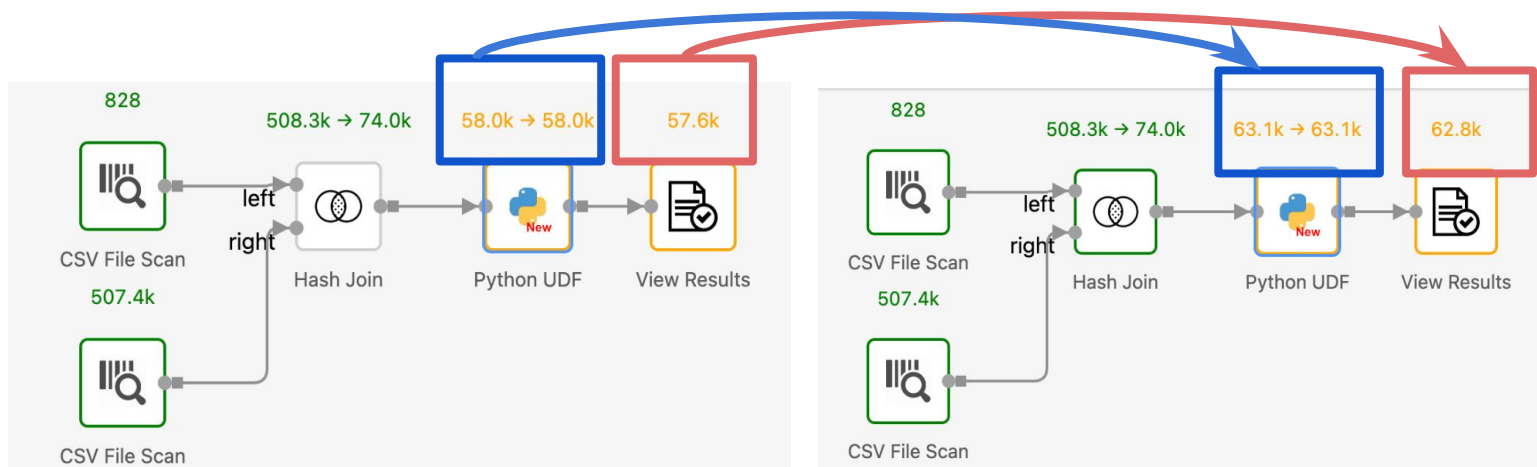


# Approach 2: Periodically Send Incremental Updates



# Approach 2: Periodically Send Incremental Updates

An example



Before

After

# Approach 2: Periodically Send Incremental Updates

## Another example

•	<code>self._context.operator_manager.operator.average_distance</code>
•	155.36079087094146
•	<code>self._context.tuple_processing_manager.current_input_tuple</code>

All Workers ▾

Before

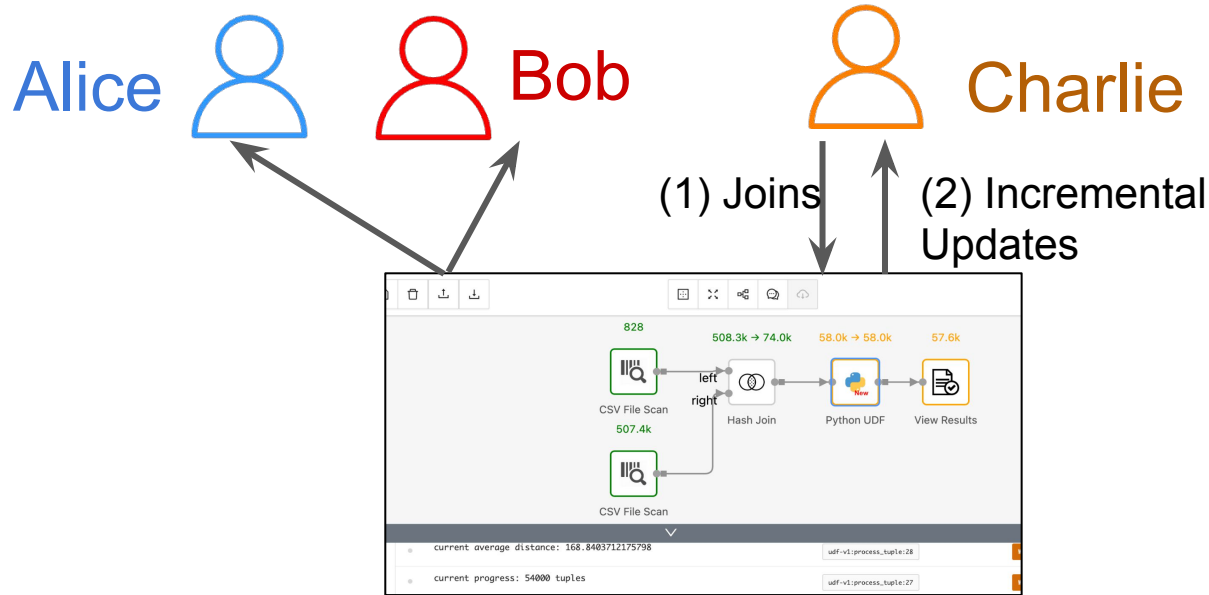
•	<code>self._context.operator_manager.operator.average_distance</code>
•	155.36079087094146
•	<code>self._context.tuple_processing_manager.current_input_tuple</code>
•	<code>Tuple['id': 'e1e35d357ceefa52', 'long_low': -118.4019312, 'lat_low': -300.0, 'long_high': -118.352695, 'lat_high': -300.0, 'id#@1': 1471642336094396417, 'text': 'My number 1 motivation to stop climate change is so that I can make my son Cesar so he can be CP3']</code>

All Workers ▾

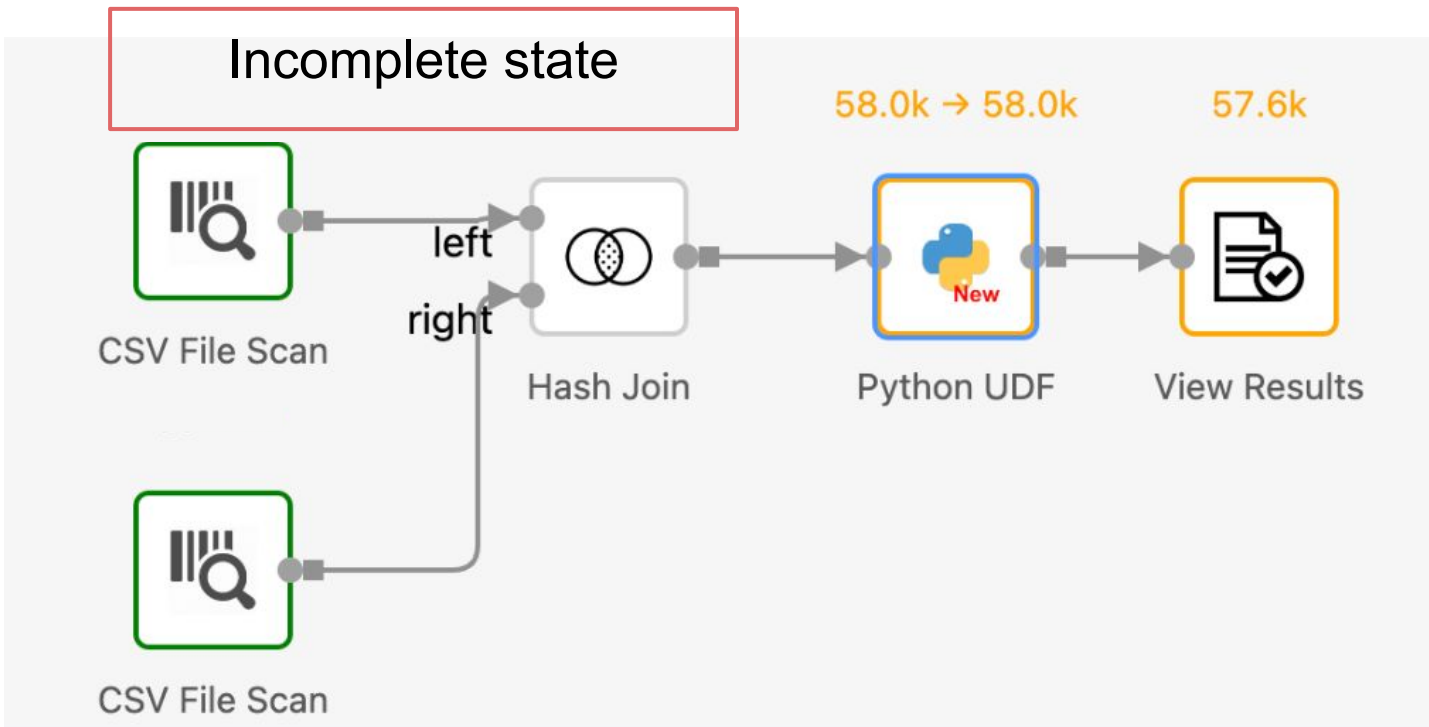
After

# Approach 2: Periodically Send Incremental Updates

Adding a new user



# Problem with Approach 2 When Adding a New User



## Problem with Approach 2 When Adding a New User

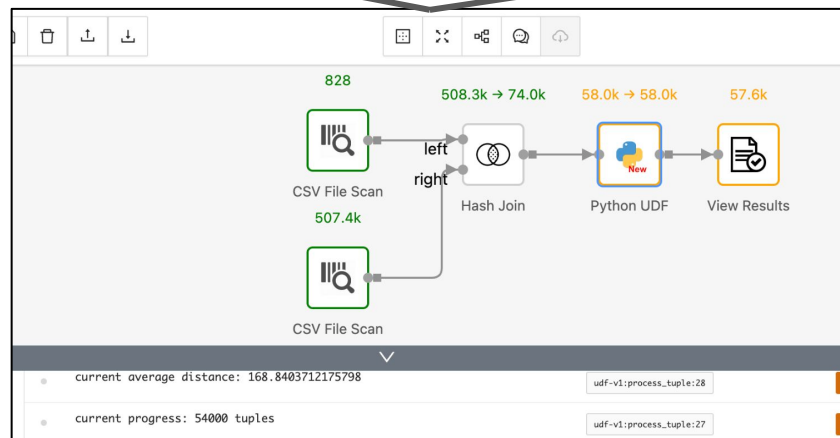
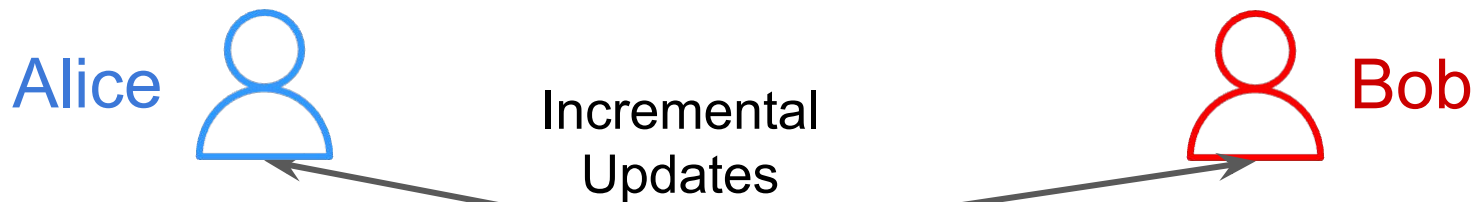


Incomplete interaction history

- `Tuple['id': 'e1e35d357ceefa52', 'long_low': -118.4019312, 'lat_low': -300.0, 'long_high': -118.352695, 'lat_high': -300.0, 'id#@1': 1471642336094396417, 'text': 'My number 1 motivation to stop climate change is so that I can make my son Cesar so he can be CP3']`

All Workers ▾

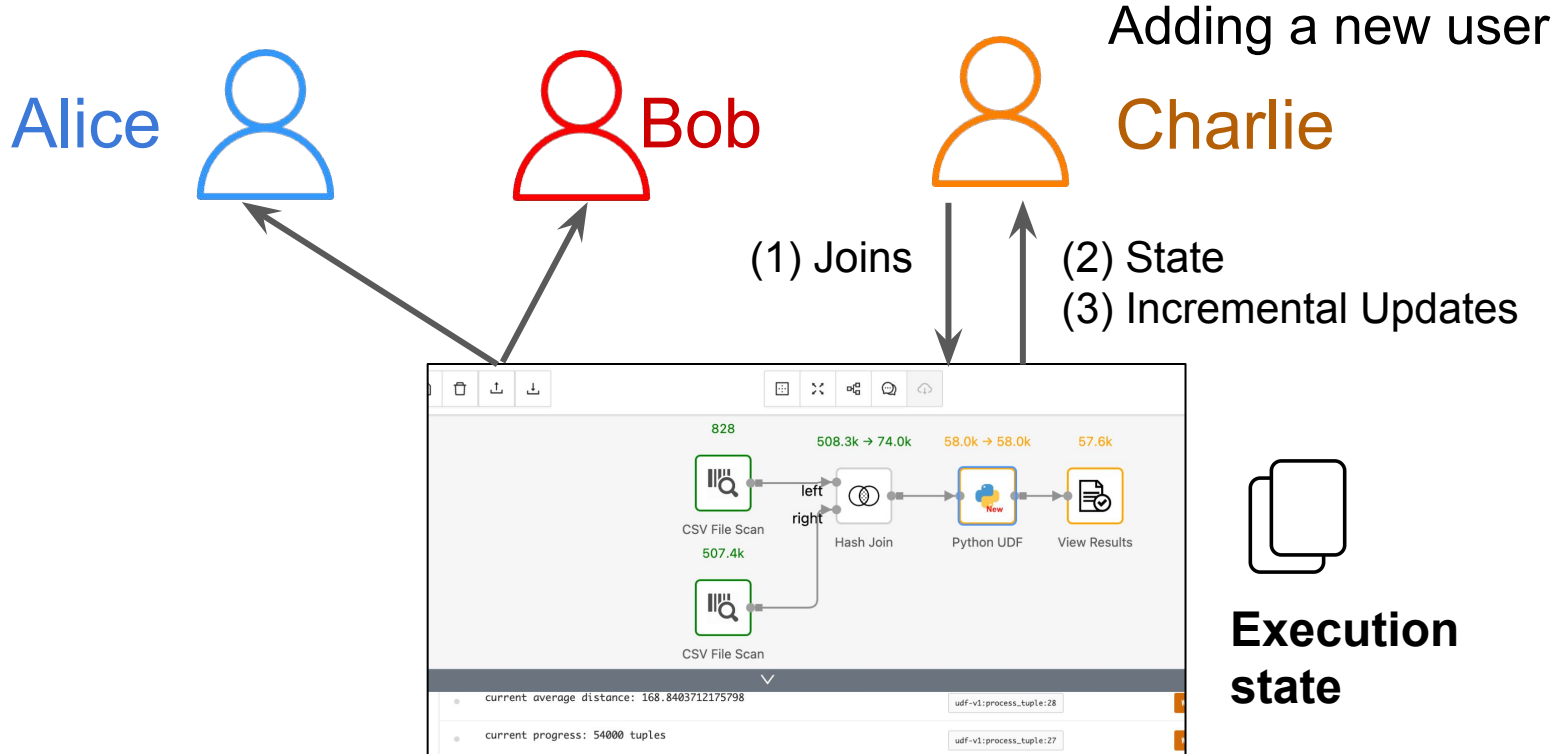
# Approach 3: Keep State and Send Incremental Updates



**Execution state**

The icon consists of two overlapping rounded rectangles, representing a state or a set of data. Below it, the text "Execution state" is written in a bold, black font.

# Approach 3: Keep State and Send Incremental Updates





# Tutorial Outline

1. Overview of collaborative data analytics (Chen)
2. Challenges and solutions:
  - a. → Shared workflow editing (Chen)
  - b. → Shared workflow execution (Chen and Zuozhi)
  - c. → Interacting with runtime execution (Zuozhi)
  - d. Runtime co-debugging on Python UDFs (Zuozhi)
3. Open challenges (Chen and Zuozhi)

# Supporting Interactions at Runtime

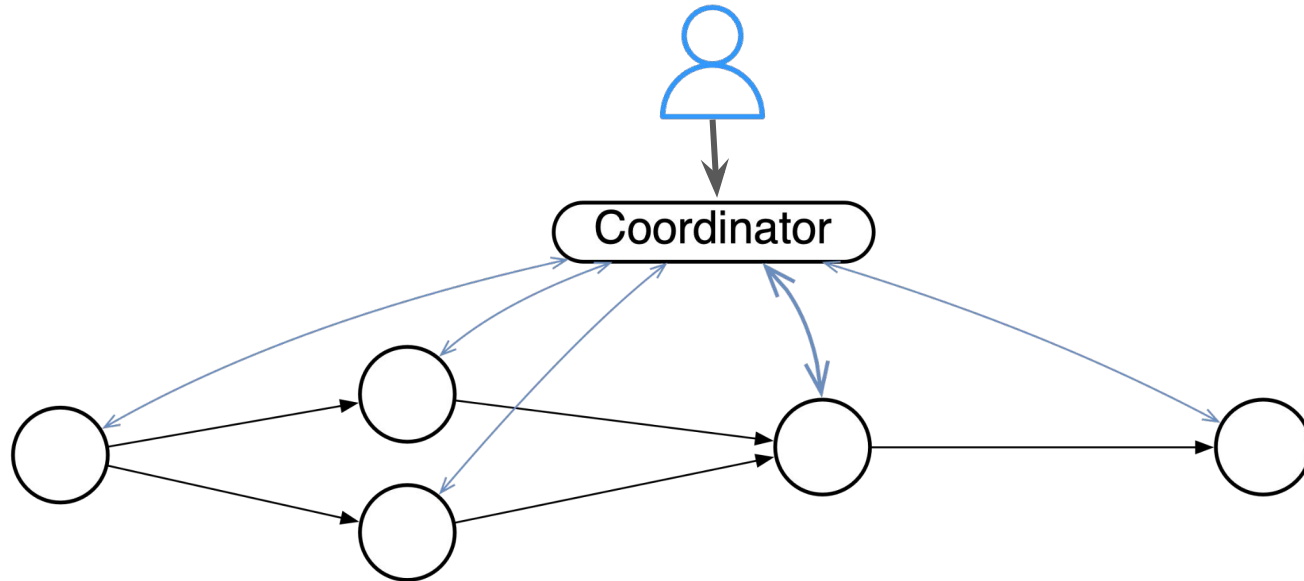
## Requirements of Pause:

- Fast
- Interact with an operator after pause

The screenshot displays a data pipeline interface. At the top, there is a toolbar with various icons and a timer showing 0:05:39. A red box highlights a 'Pause' button, which is a blue button with a red power icon. Below the toolbar, the pipeline consists of three operators: 'Aggregate' (1 → 1), 'Bar Chart' (1 → 1), and 'Section Metadata Distribution' (1). The 'Aggregate' operator is highlighted with a green box. On the right side, there is a 'Deformation' configuration panel with a 'View code content' button. The panel includes a 'Worker count' field set to 8, a 'Specify how many parallel workers to lunch' label, a checked 'Retain input columns' checkbox, and an 'Extra output column(s)' field.

# Approach 1: Using OS Signals

**SIGSTOP** (signal stop) / **SIGCONT** (signal continue)



# Using OS Signals to Pause

User

Operator

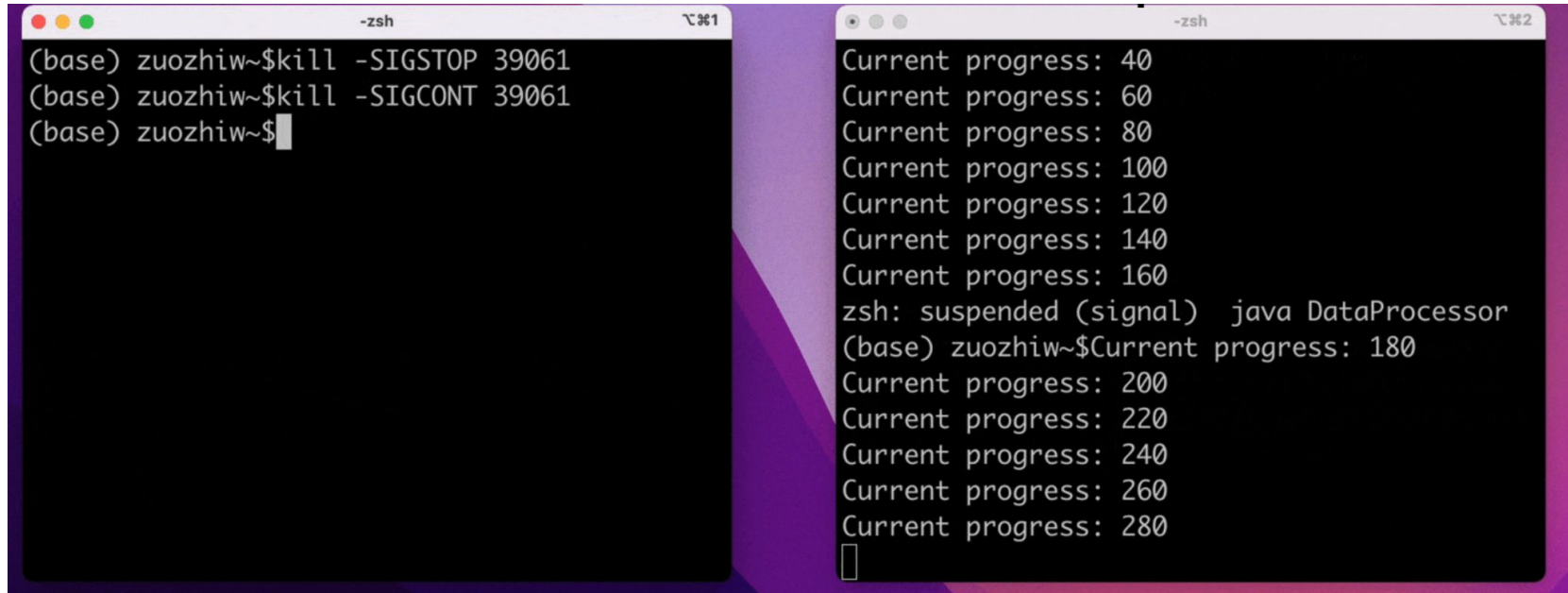
```
(base) zuozhiw~$kill -SIGSTOP 39061
(base) zuozhiw~$

(base) zuozhiw~$java DataProcessor
Current Process ID (PID): 39061
Current progress: 20
Current progress: 40
Current progress: 60
Current progress: 80
Current progress: 100
Current progress: 120
Current progress: 140
Current progress: 160
zsh: suspended (signal) java DataProcessor
(base) zuozhiw~$
```

# Using OS Signals to Resume

User

Operator



```
(base) zuozhiw~$kill -SIGSTOP 39061
(base) zuozhiw~$kill -SIGCONT 39061
(base) zuozhiw~$
```

```
Current progress: 40
Current progress: 60
Current progress: 80
Current progress: 100
Current progress: 120
Current progress: 140
Current progress: 160
zsh: suspended (signal) java DataProcessor
(base) zuozhiw~$Current progress: 180
Current progress: 200
Current progress: 220
Current progress: 240
Current progress: 260
Current progress: 280
```

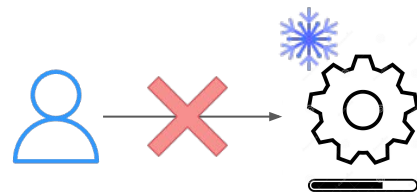
# Approach 1: Using OS Signals

Pros: 😊

- Suspends immediately
- OS native support - little implementation effort

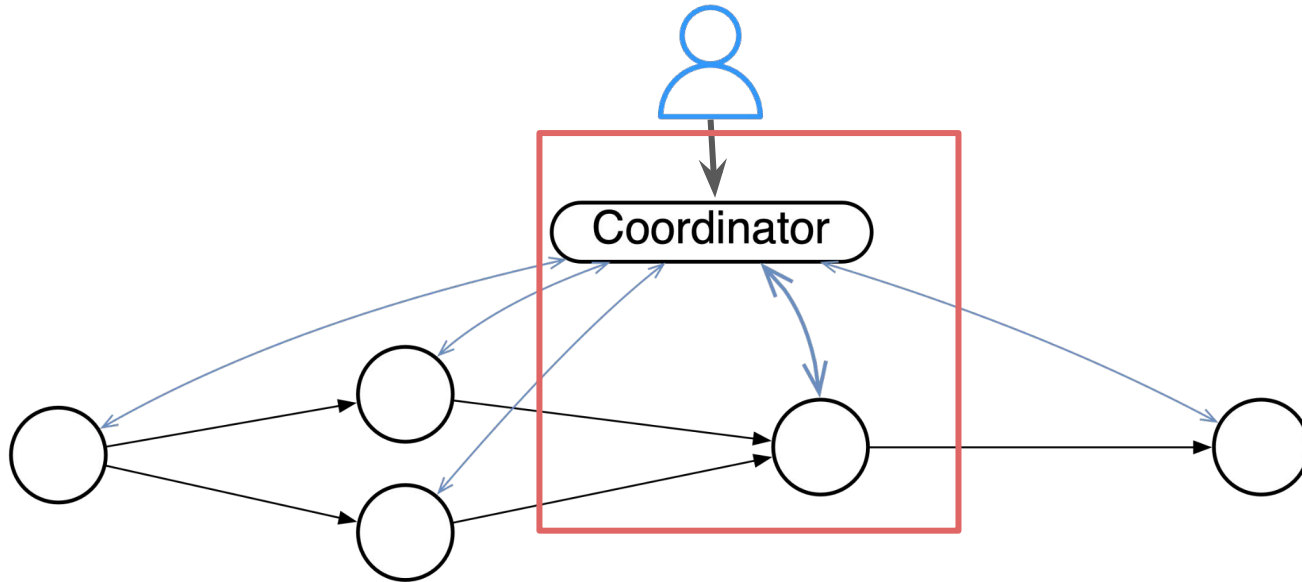
Cons: 😞

- Unable to interact with an operator

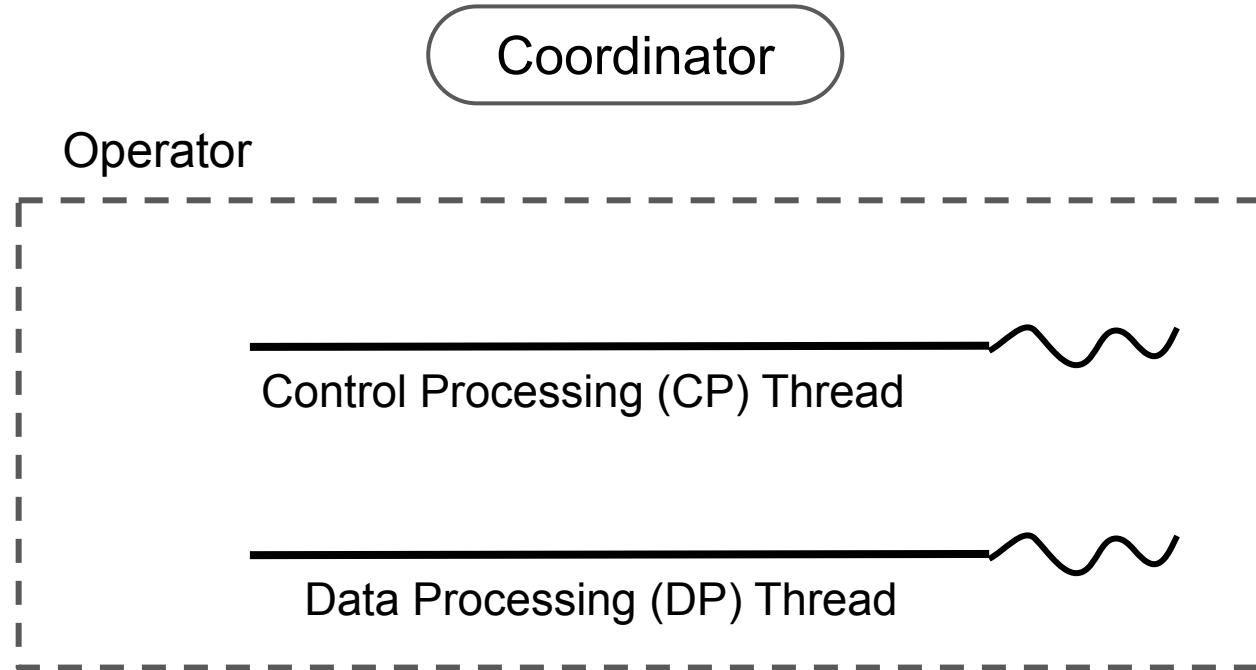


## Approach 2: Using Thread-level Signal

In Java: **Thread.Suspend()** / **Thread.Resume()**



## Approach 2: Using Thread-level Signal

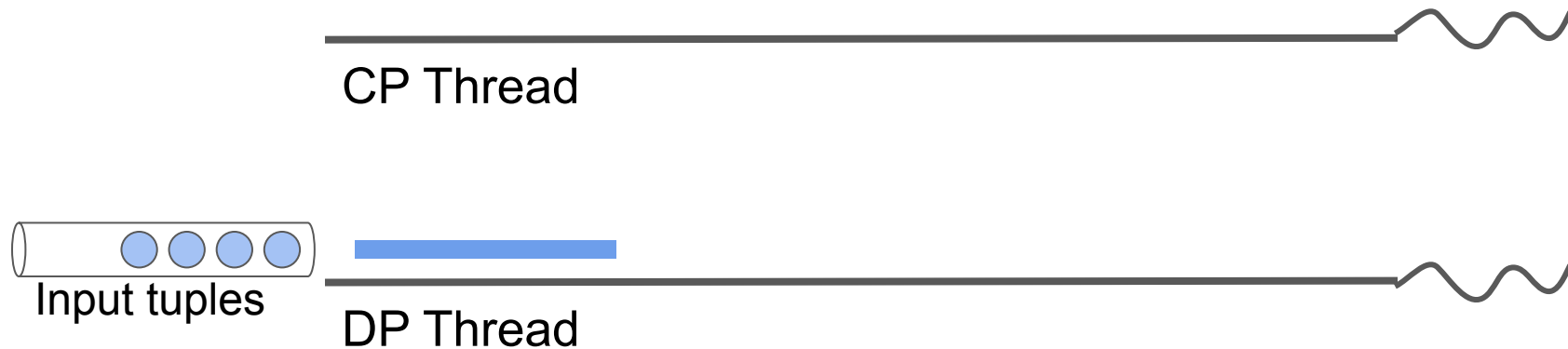




# Approach 2: Using Thread-level Signal

Pause operator execution

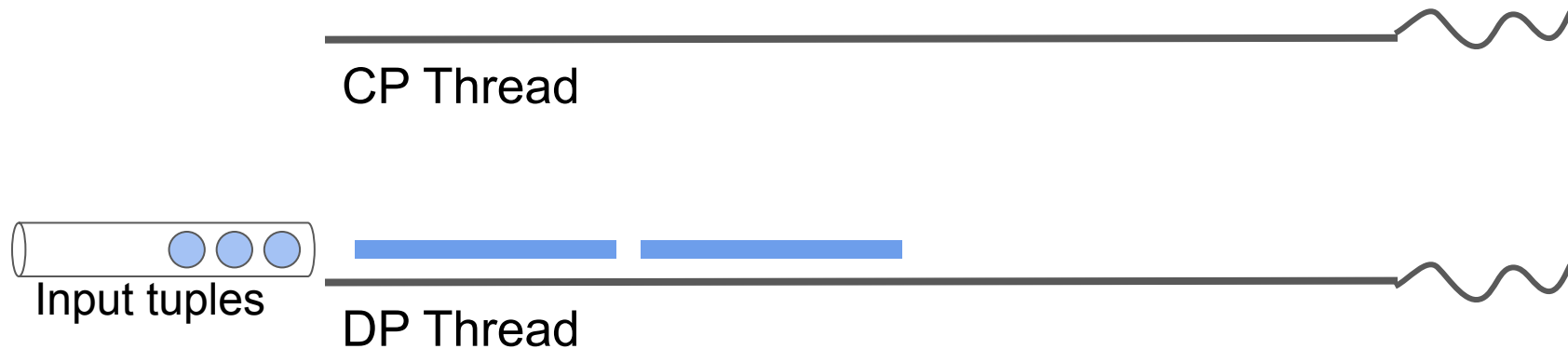
Coordinator



# Approach 2: Using Thread-level Signal

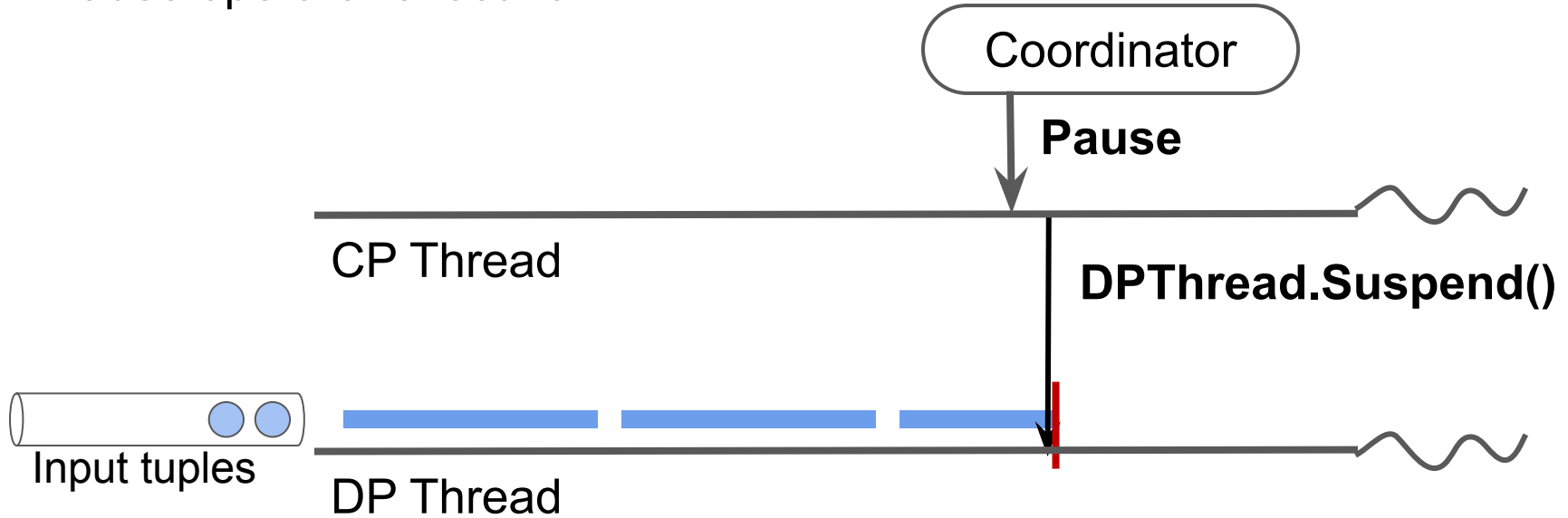
Pause operator execution

Coordinator



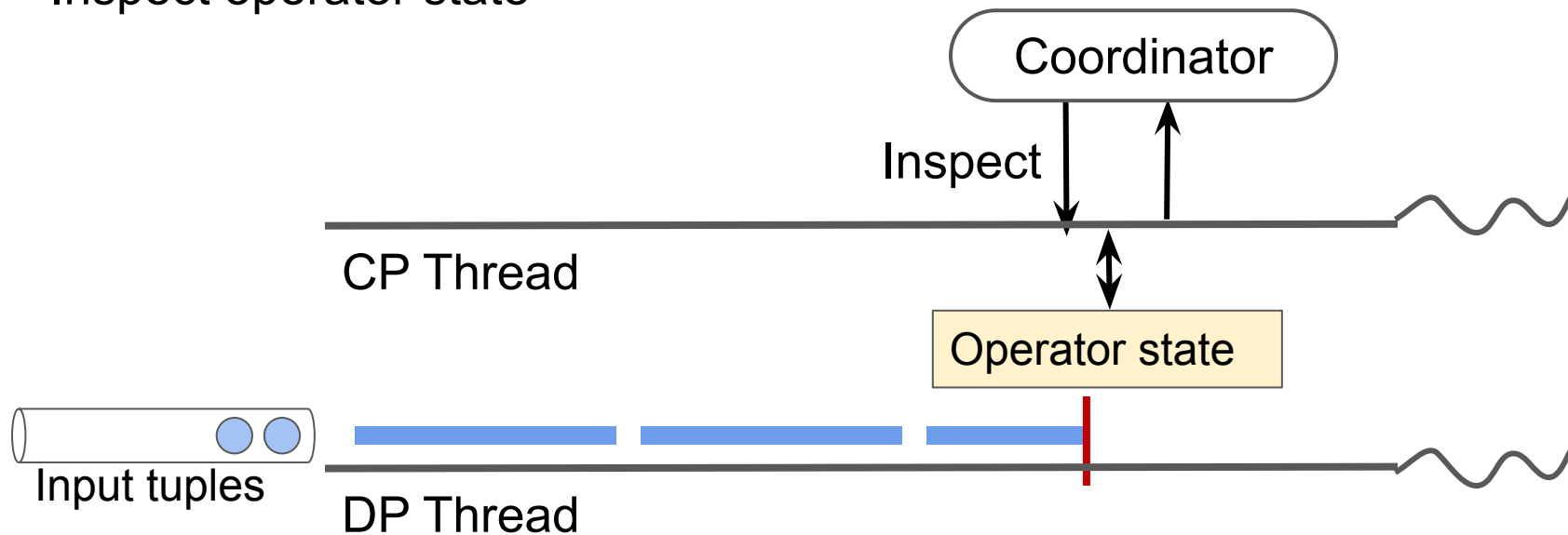
# Approach 2: Using Thread-level Signal

Pause operator execution



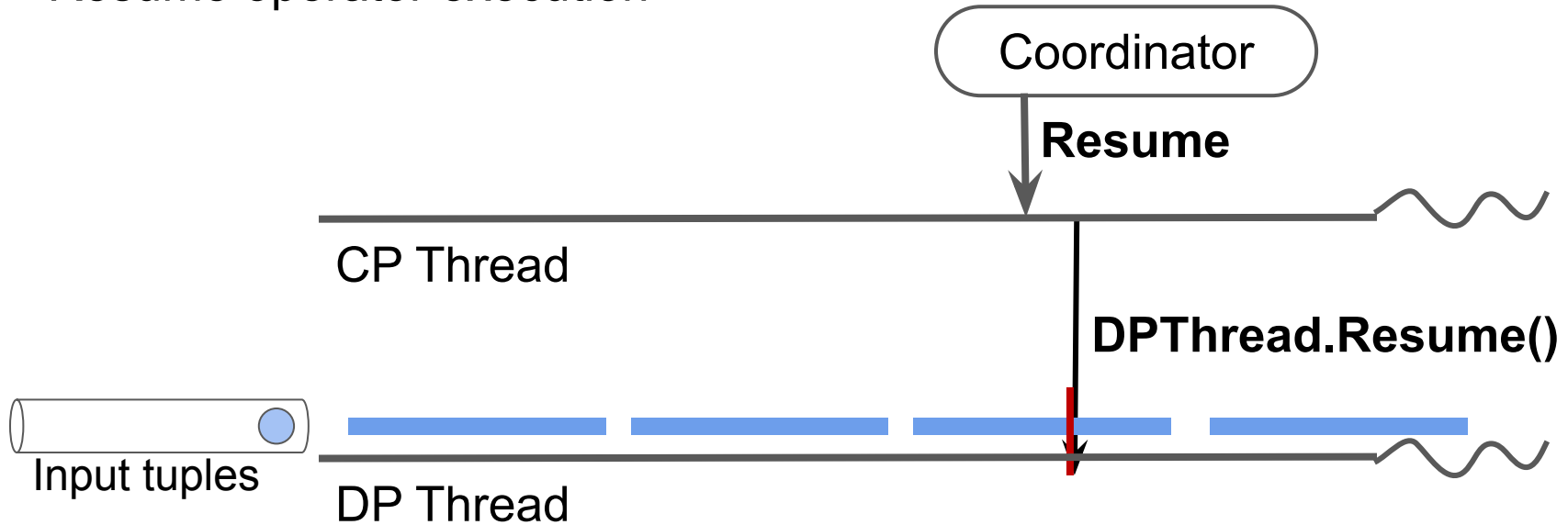
## Approach 2: Using Thread-level Signal

Inspect operator state




# Approach 2: Using Thread-level Signal

Resume operator execution



## Approach 2: Using Thread-level Signal

Pros: 

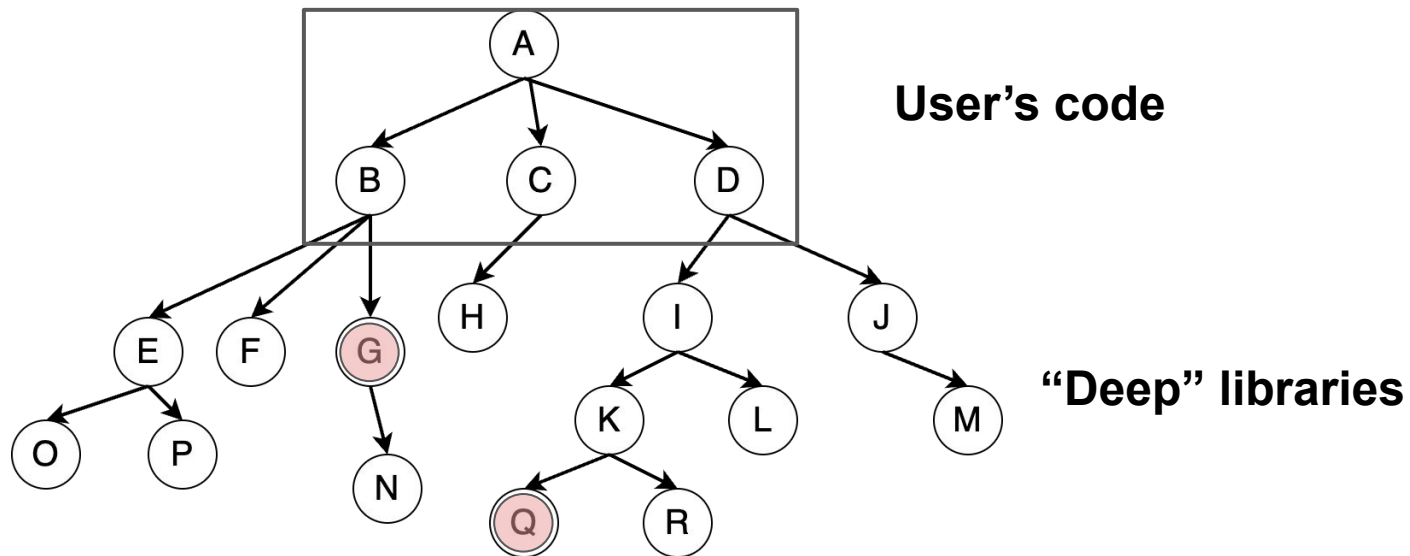
- Suspends immediately
- Able to interact with operator after pause

# Problem with Approach 2

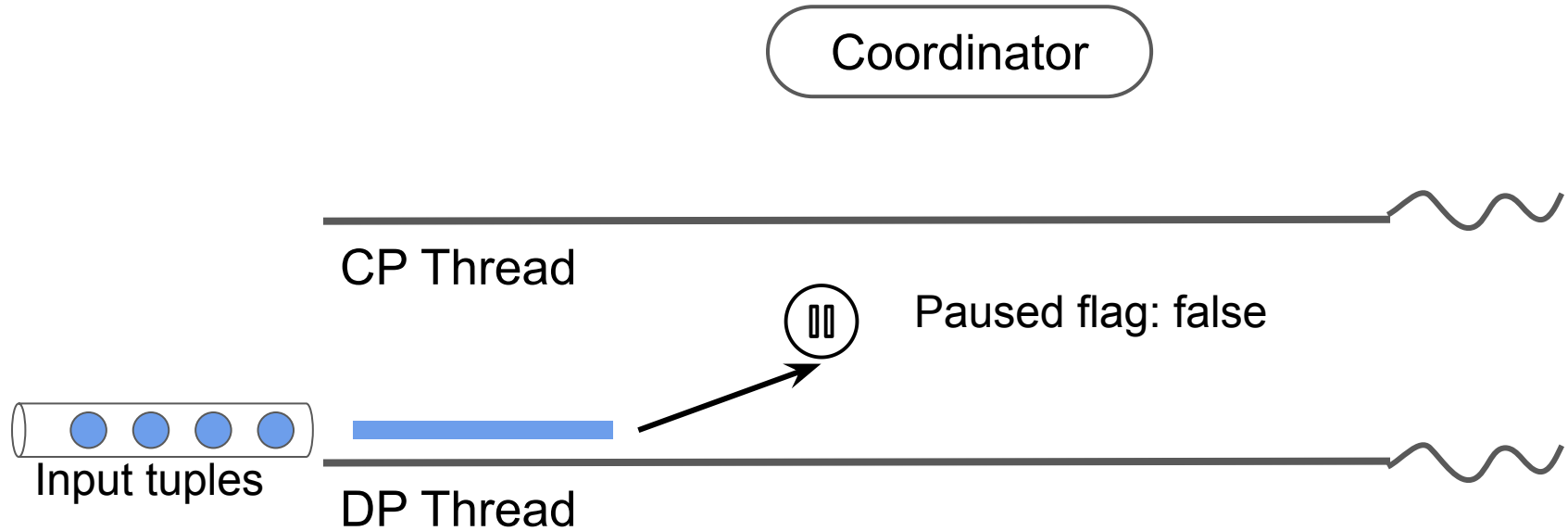


Stopping point is arbitrary and incomprehensible to users.

Operator's call graph

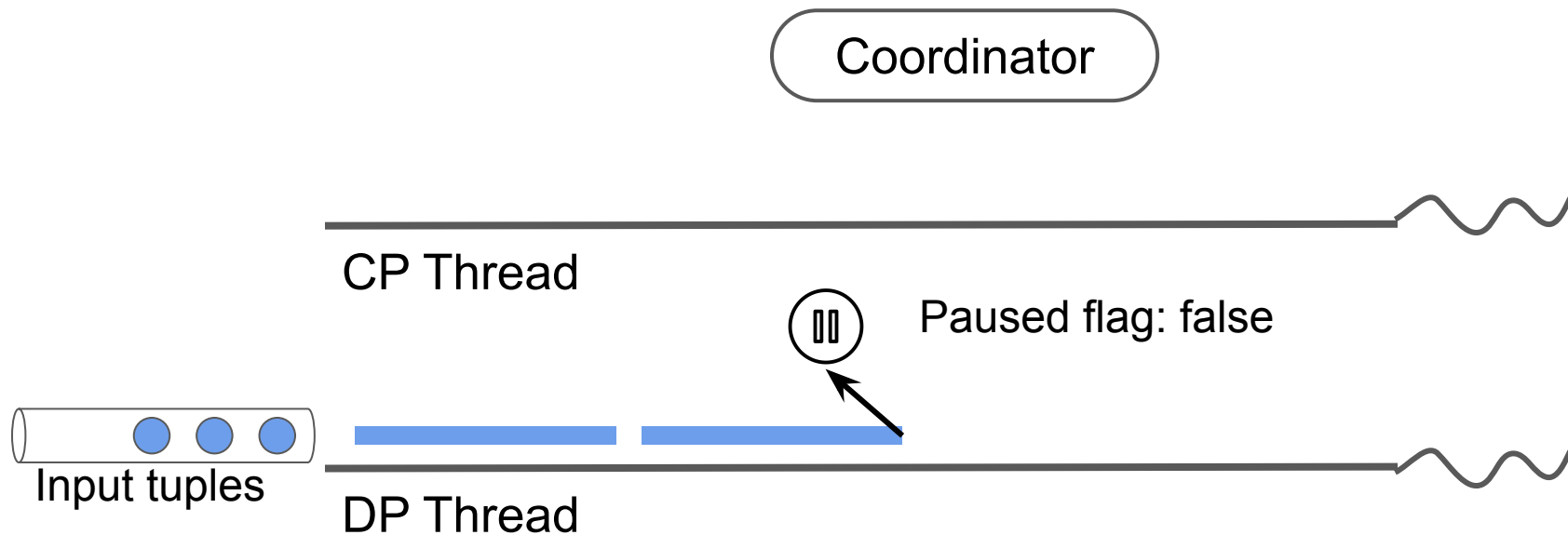


# Approach 3: Checking for Pause between Tuples

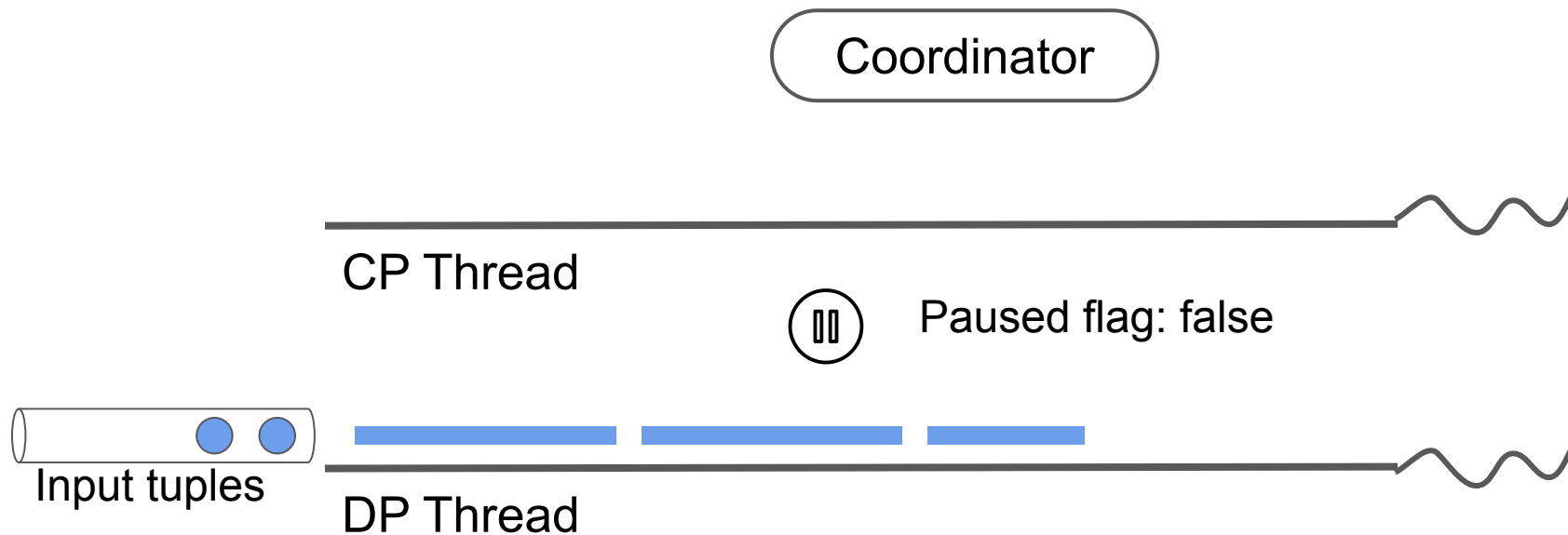




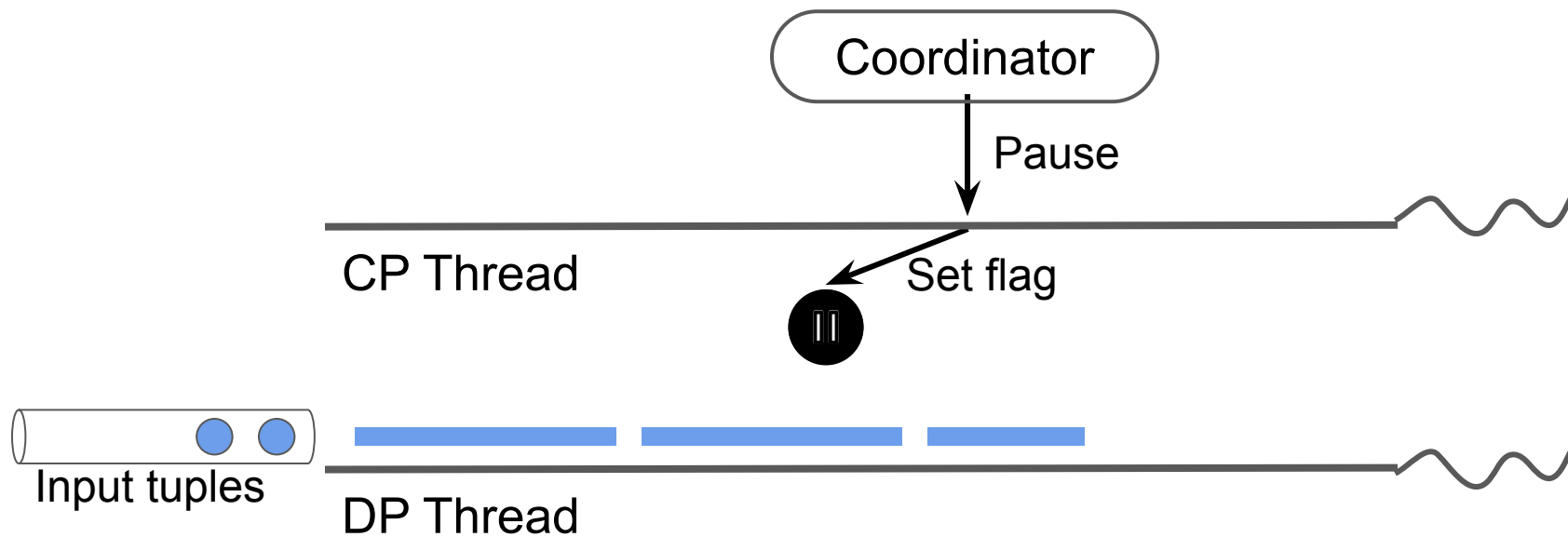
# Approach 3: Checking for Pause between Tuples



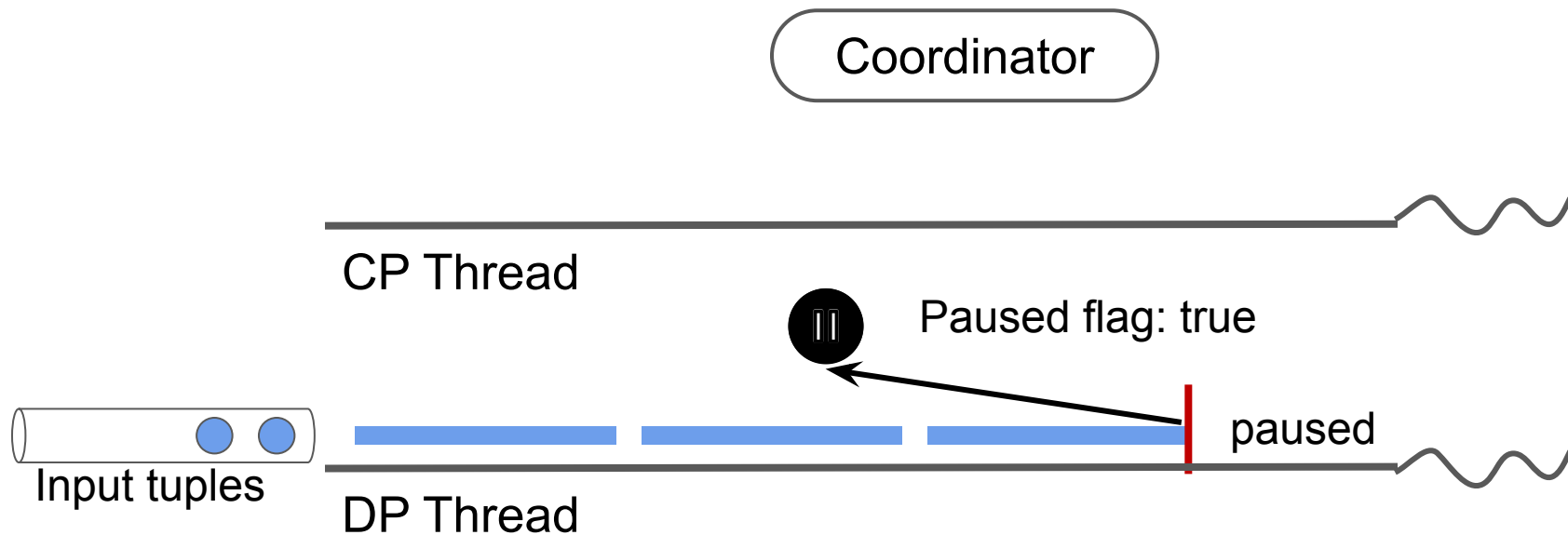
# Approach 3: Checking for Pause between Tuples




## Approach 3: Checking for Pause between Tuples



# Approach 3: Checking for Pause between Tuples



## Approach 3: Checking for Pause between Tuples

Pros: 

- Meaningful stopping points

Cons: 

- Slower than approaches 1 and 2
- Runtime overhead of checking (10%)

# Comparison of three approaches

	Approach 1 (OS signals)	Approach 2 (Thread signals)	Approach 3 (Between tuples)
Pause Speed	Fast	Fast	OK
Interactions	No	Yes	Yes
Meaningful Stopping Point	No	No	Yes

# Tutorial Outline

1. Overview of collaborative data analytics (Chen)
2. Challenges and solutions:
  - a. → Shared workflow editing (Chen)
  - b. → Shared workflow execution (Chen and Zuozhi)
  - c. → Interacting with runtime execution (Zuozhi)
  - d. → Runtime co-debugging on Python UDFs (Zuozhi)
3. Open challenges (Chen and Zuozhi)

# User Defined Functions in Workflows

The screenshot displays the PyTera workflow editor. On the left, a sidebar lists various workflow components: Search operator, Source, Search, Analytics, Join, Utilities, User-defined Functions, Visualization, and View Results. The main workspace is titled 'Python Script' and contains the following code:

```
1 # Choose from the following templates:
2 #
3 from pyterera import *
4 from datetime import datetime
5
6
7 class ProcessTupleOperator(UDFOperatorV2):
8
9     @overrides
10    def process_tuple(self, tuple_: Tuple, port: int) -> Iterator[Optional[TupleLike]]:
11
12        # Extract the month as an integer
13        timestamp_str = tuple_['created_at']
14
15        timestamp = self.parse_time(timestamp_str)
16
17        # Extract the month as an integer
18        tuple_['month'] = str(timestamp.month)
19        yield tuple_
20
21    def parse_time(self, timestamp_str: str) -> datetime:
22        formats = ["%Y-%m-%d %H:%M:%S.%f"]
23
24        # Convert the timestamp string to a datetime object
25        for timestamp_format in formats:
26            try:
27                timestamp = datetime.strptime(timestamp_str, timestamp_format)
28                return timestamp
29            except:
30                continue
31
```

On the right, the configuration panel for the 'Month Extractor' UDF is visible. It includes an 'Edit code content' button, a 'Worker count' field set to 1, a 'Specify how many parallel workers to launch' section, a checked 'Retain input columns \*' option, and an 'Extra output column(s)' section where the attribute name is 'month' and the attribute type is 'string'.



# Runtime Co-debugging on Python UDFs

## Data errors

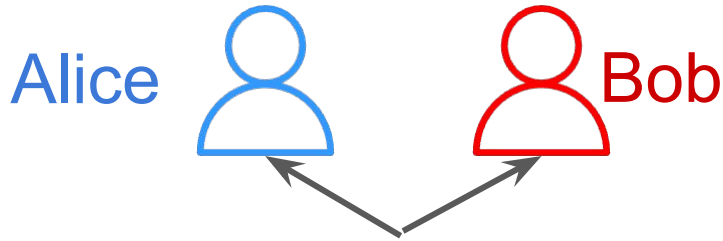
- misformatted data
- unexpected nulls
- corrupted data
- ...

## UDF errors

- corner cases
- wrong parameters
- ...

# Co-debugging Demo

Domain experts need technical assistance in debugging.

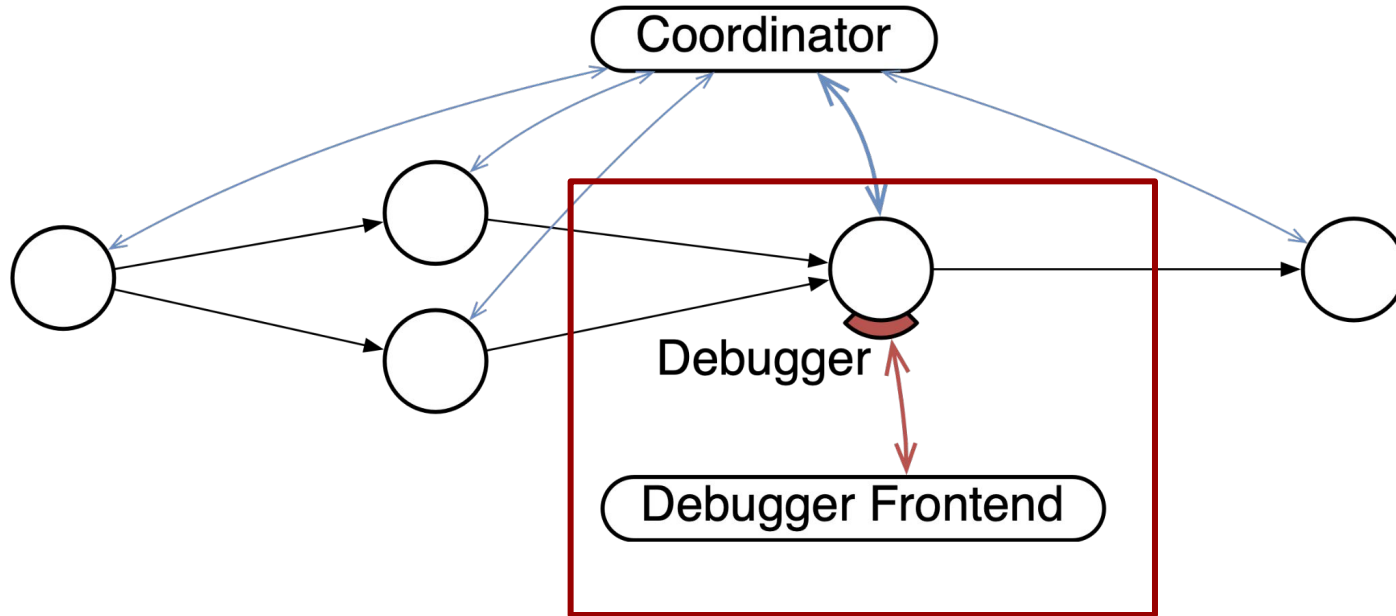


The screenshot displays a data pipeline with the following components: two 'CSV File Scan' nodes (507.4k each), a 'Hash Join' node (508.3k → 74.0k), a 'Python UDF' node (58.0k → 58.0k), and a 'View Results' node (57.6k). A red error message is visible at the bottom of the interface:

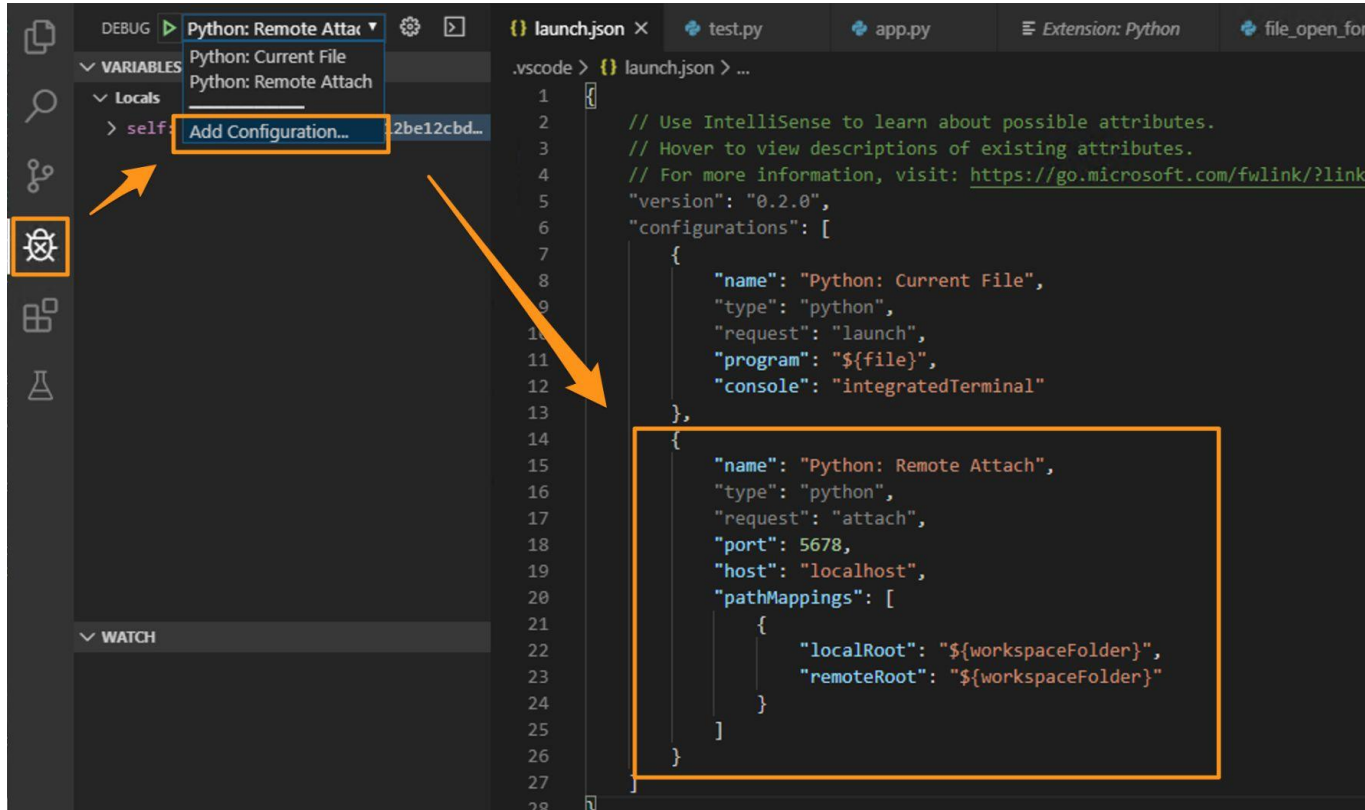
```
Worker ActorVirtualIdentity(Worker:WF5-PythonUDFV2-operator-b9f560ac-d404-4ebe-8...
Traceback (most recent call last): File "/Users/zuozhiw/workspace/texera-
dev/texera/core/amber/src/main/python/core/runnables/data_processor.py", line 50
"/var/folders/nv/p64_15js3m5dtbxbw7x73bm8000gn/T/tmpdeqbfuffstTempFS/udf-v1.py"
min(dist, hs.haversine((target).loc, unit=Unit.MILES, check = True)) File "/Users/
packages/haversine/haversine.py", line 219, in haversine _ensure_lat_lon(lat2, 1
packages/haversine/haversine.py", line 93, in _ensure_lat_lon raise ValueError(f
Latitude -300.0 is out of range [-90, 90]
```

Runtime Error

# Approach 1: Attaching a Remote Debugger to a UDF



# Approach 1: Attaching a Remote Debugger to a UDF



# Approach 1: Attaching a Remote Debugger to a UDF

Pro:



- Works out-of-the-box. (Adopted by PyFlink, PySpark)

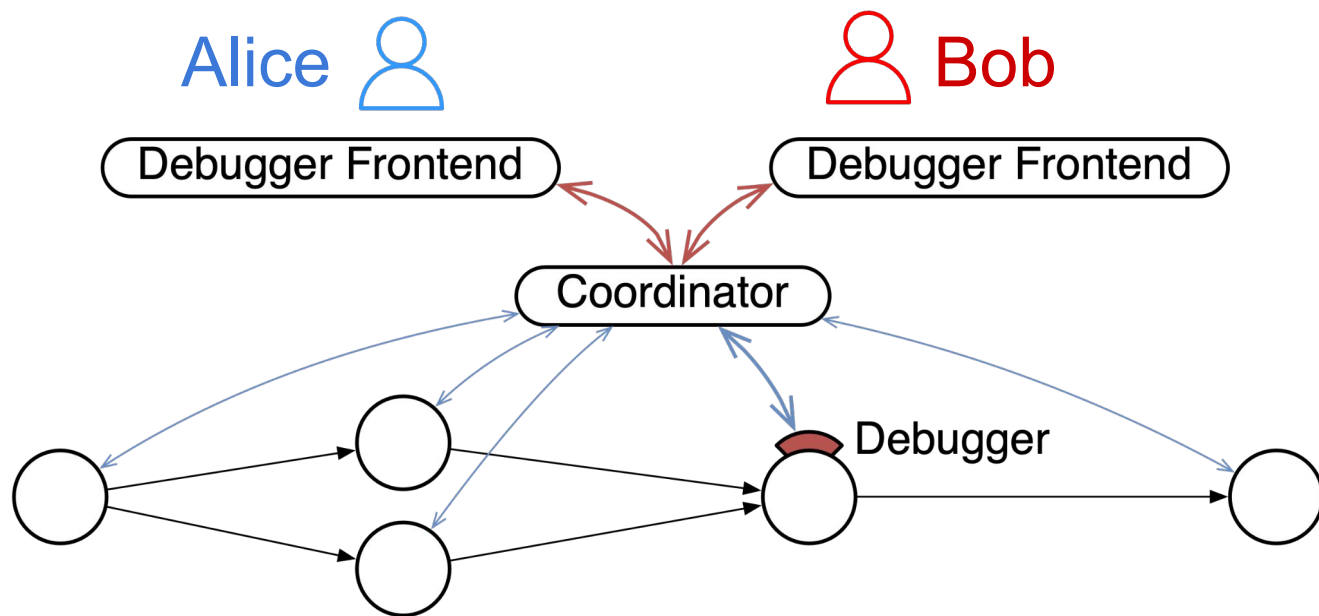
Con:



- Cannot co-debug: only a single debugger can be attached

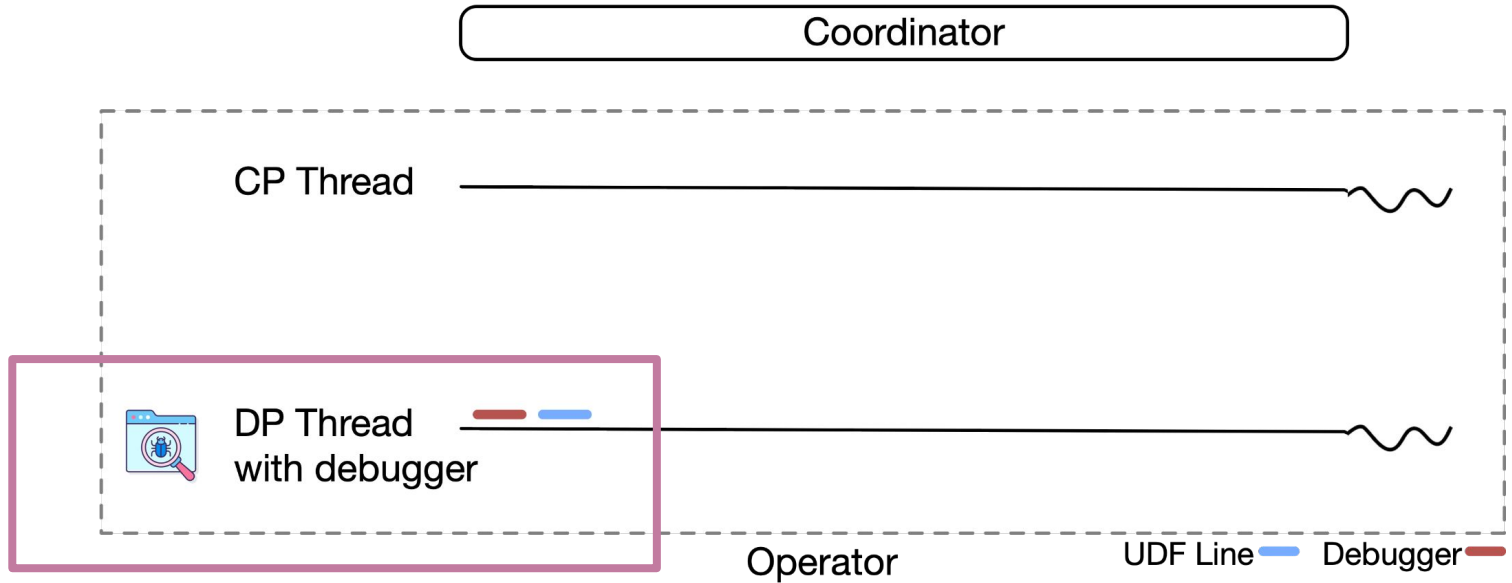
## Approach 2: Coordinator Controls Debuggers

Co-debugging: coordinator manages multiple debug frontends



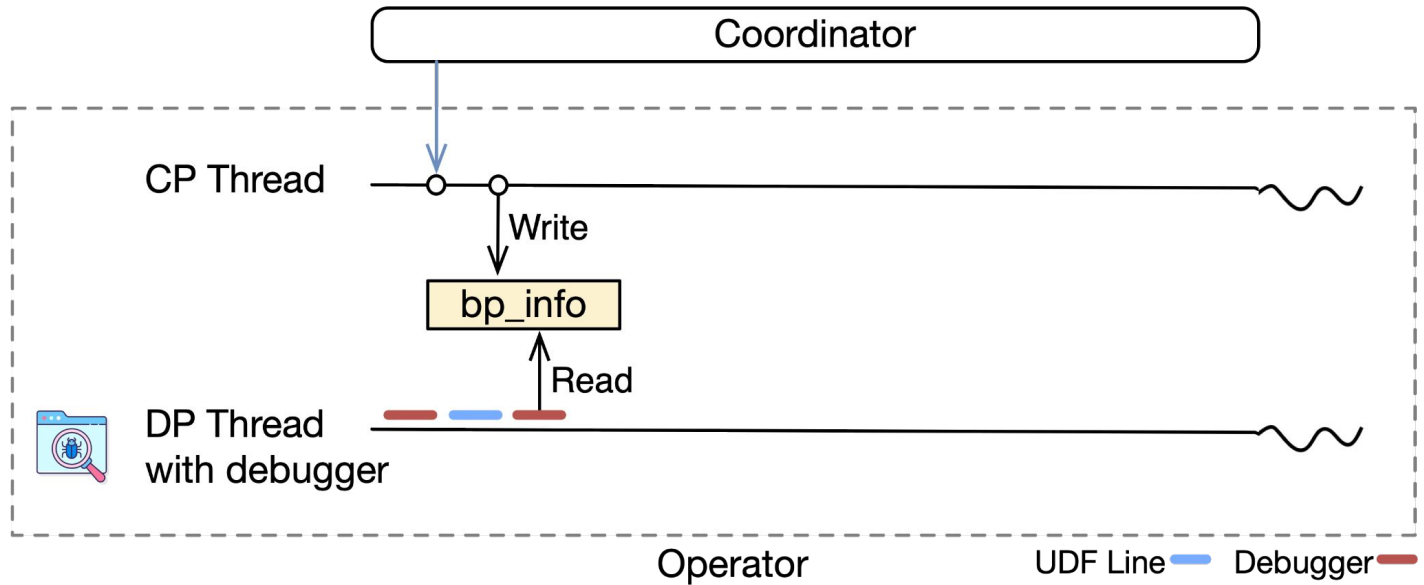
# Approach 2: Coordinator Controls Debuggers

## Execution model



# Approach 2: Coordinator Controls Debuggers

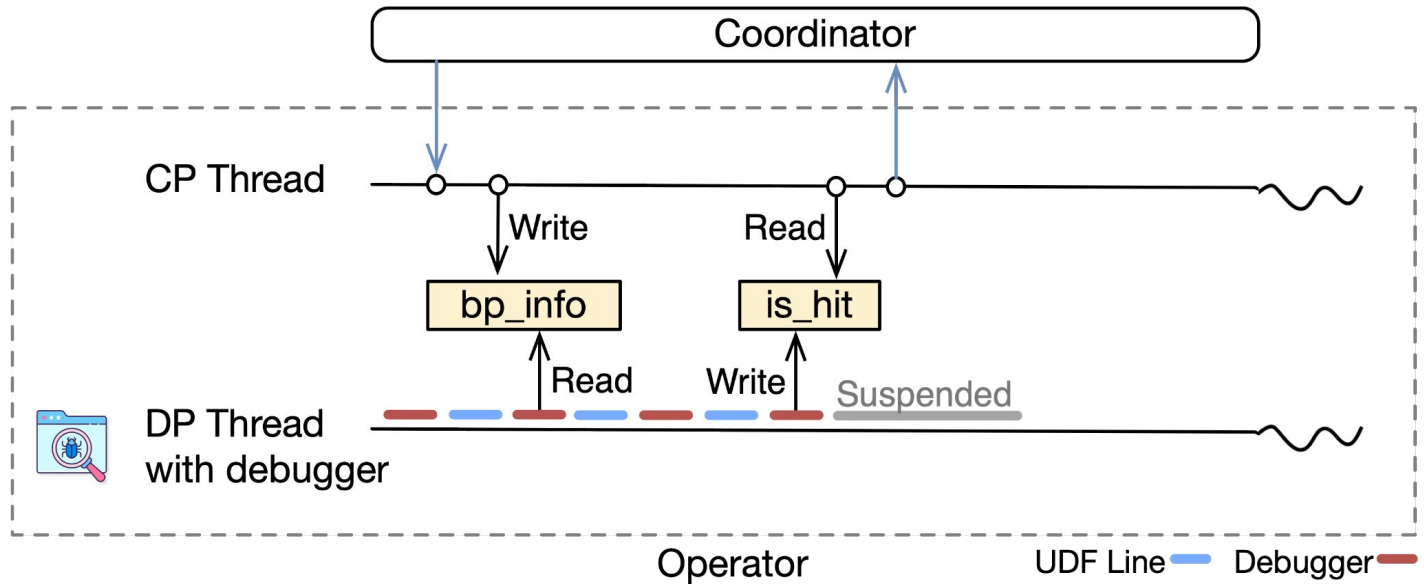
## Setting a breakpoint





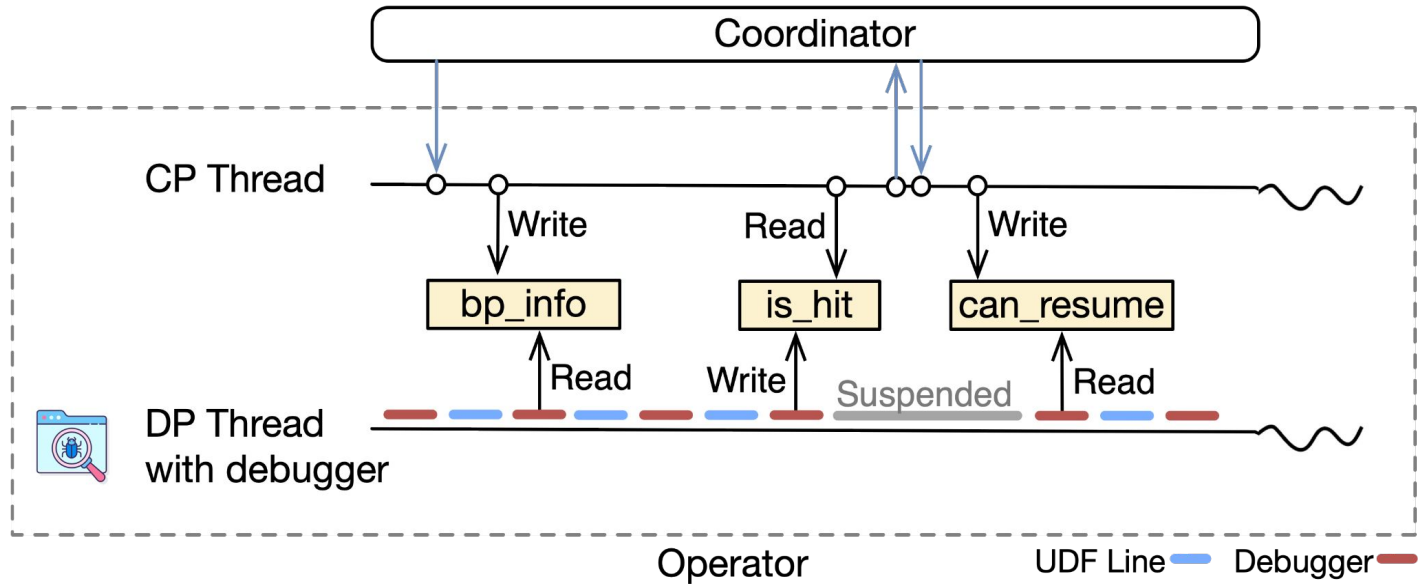
# Approach 2: Coordinator Controls Debuggers

## Hitting a breakpoint



# Approach 2: Coordinator Controls Debuggers

## Resuming from the breakpoint



## Approach 2: Coordinator Controls Debuggers

Pro: 

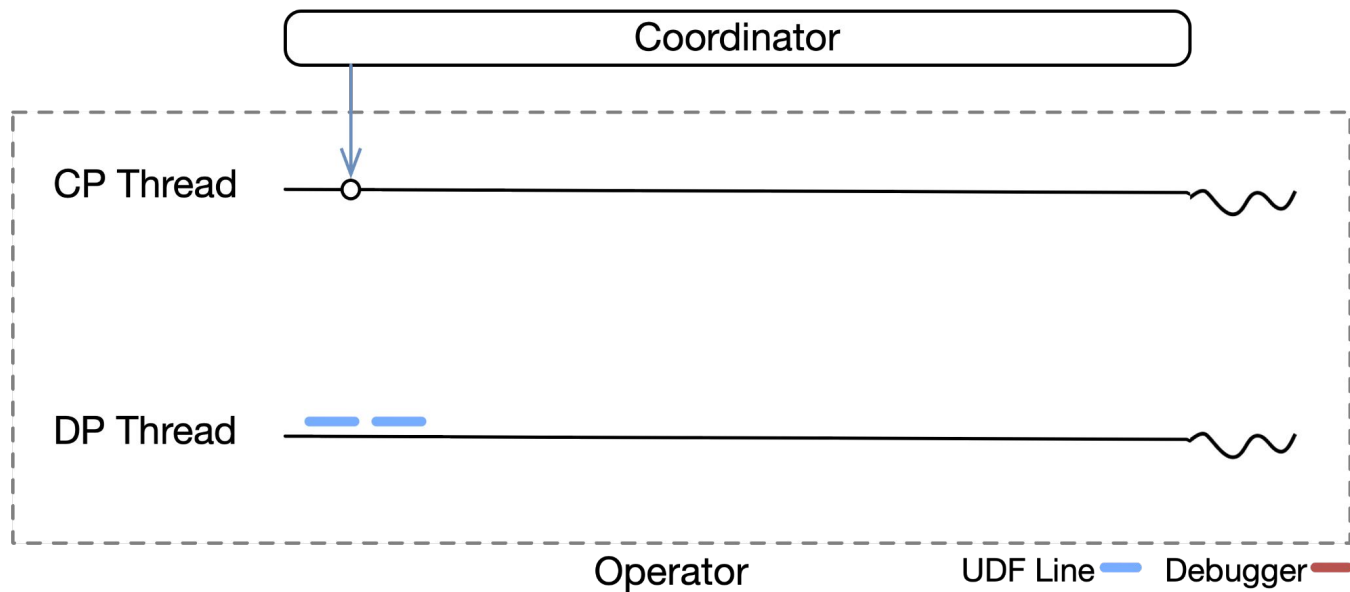
- Supports co-debug

Con: 

- High overhead (5x): always running in debug mode

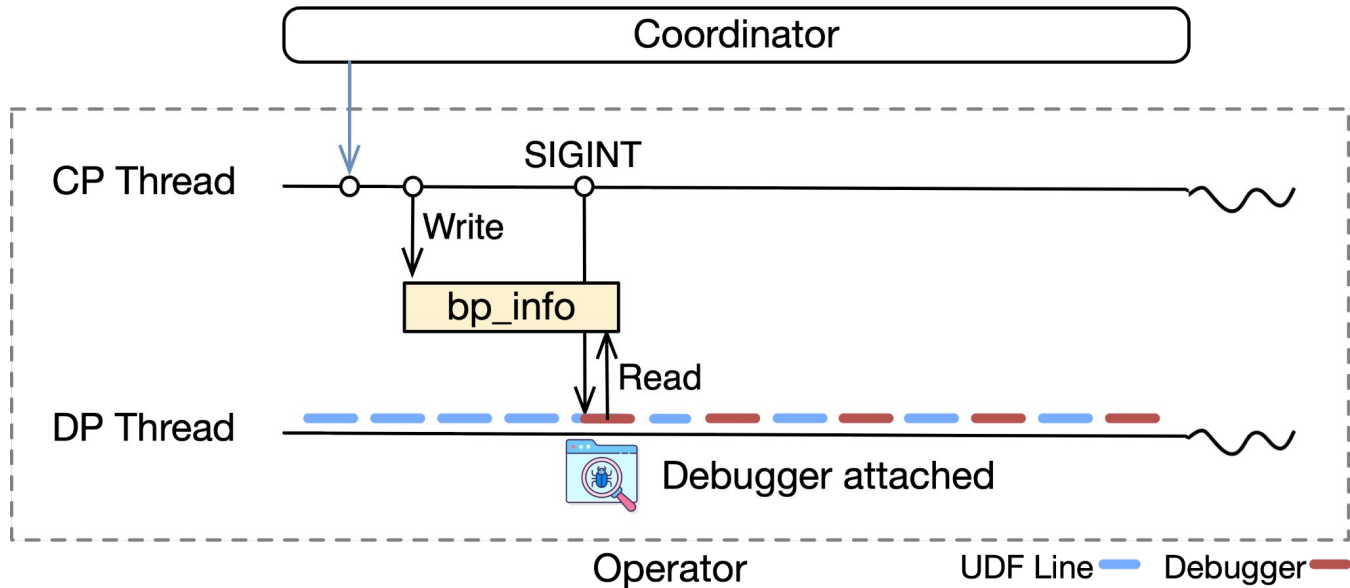
# Reducing Overhead: Invoke Debugger on Demand

Starting without a debugger



# Reducing Overhead: Invoke Debugger on Demand

## Attaching a debugger



# Adoption of Texera

<b>Deployments</b>	<b>Users</b>	<b>Projects</b>	<b>Workflows</b>	<b>Executions</b>	<b>Versions</b>	<b>Contributors</b>
4	100+	30+	1000+	10,000+	100,000+	100+

Statistics of Texera Service as of May 2023

# Tutorial Outline

1. Overview of collaborative data analytics
2. Challenges and solutions:
  - a. Shared workflow editing
  - b. Shared workflow execution
  - c. Interacting with runtime execution
  - d. Runtime co-debugging on Python UDFs
3. Open challenges

# Open Challenges

- Fault tolerance
- Serverless computation for elasticity
- Workflow ecosystem for knowledge sharing in scientific communities
- Reproducibility of workflows
- Dynamic reconfiguration: our research talk “Fries”: 3:30 pm, C8, Junior Ballroom AB
- Debugging Python UDFs: our SIGMOD 2024 “Udon” paper



# Acknowledgements



National Science Foundation (NSF)  
III 1745673, III 2107150.

## Core Texera Team:



Dr. Sadeem Alsudais



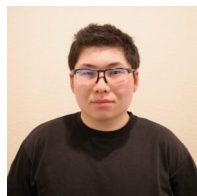
Yunyan Ding



Yicong Huang



Dr. Avinash Kumar



Xinyuan Lin



Xiaozhen Liu



Raj Mohanty

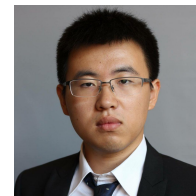


Shengquan Ni

## Speakers:



Dr. Chen Li



Dr. Zuozhi Wang

# Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang and Chen Li



UCIRVINE



Texera GitHub Repo