Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang and Chen Li

UCIRVINE

texera

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)

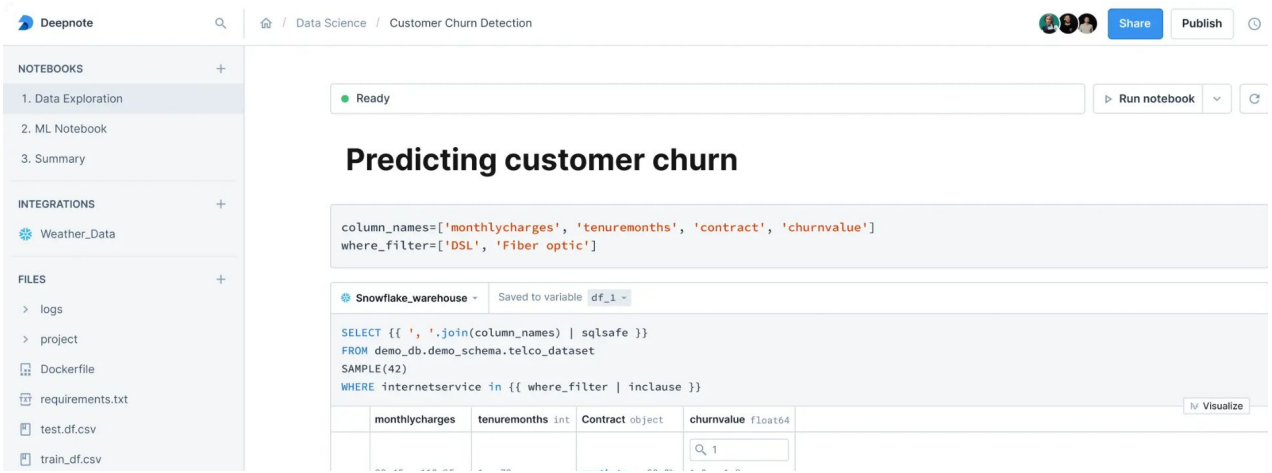
Popularity of Collaboration Cloud Services

Benefits:

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

Python Notebooks

- Support real-time collaborations
- For programmers



jupyter

colab

Deepnote

hyperquery

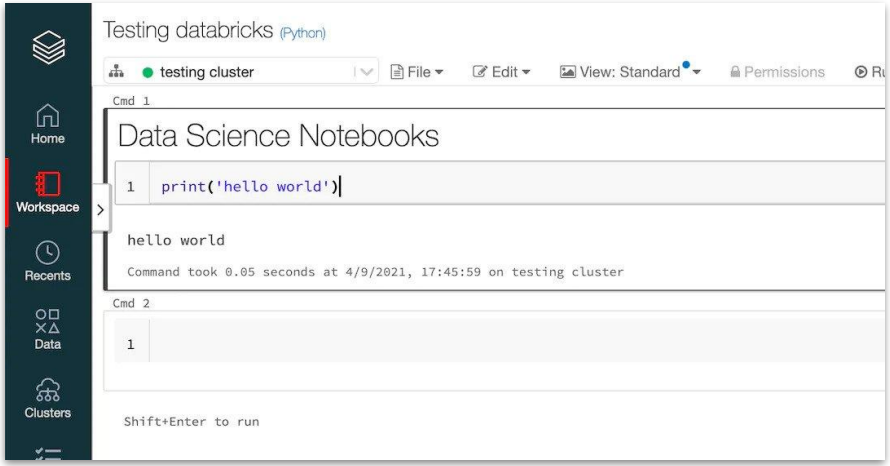
HEX

Big Data Processing Systems

Efficient and scalable

Big Data Processing Systems

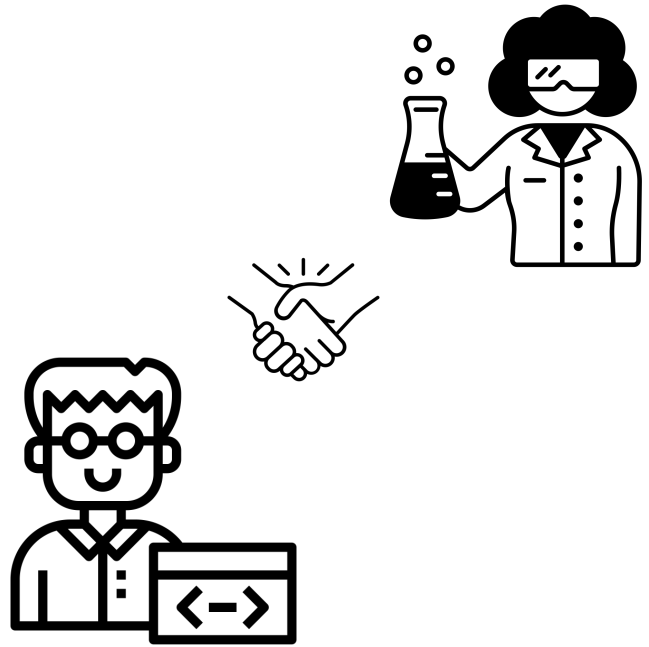
- Start to support collaborations
- For SQL experts



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

RAPIDMINER

alteryx

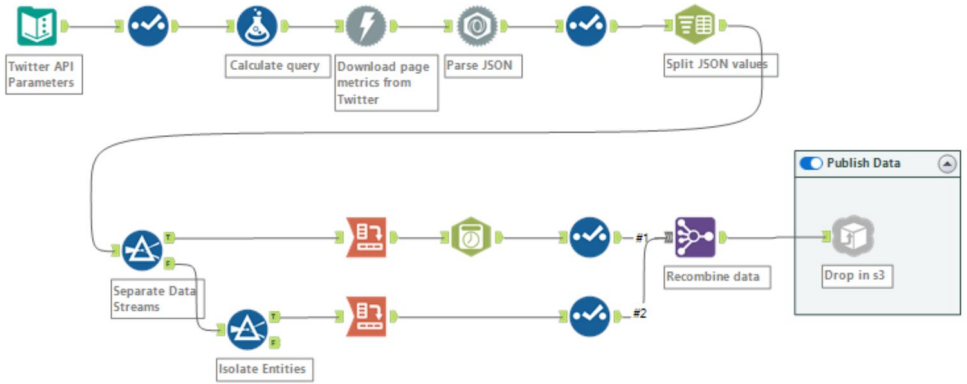
Open for Innovation KNIME

Pros:

- Easy-to-use interface
- No coding required

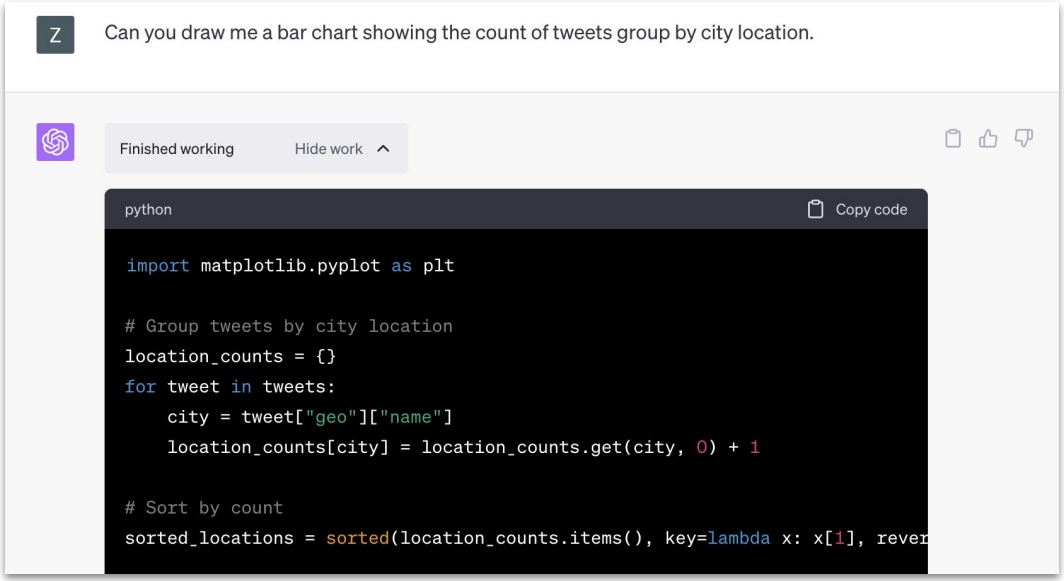
Cons:

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



ChatGPT Code Interpreter

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



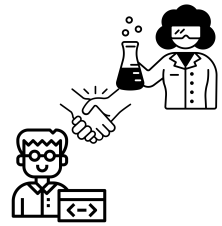
System Requirements for Collaborations

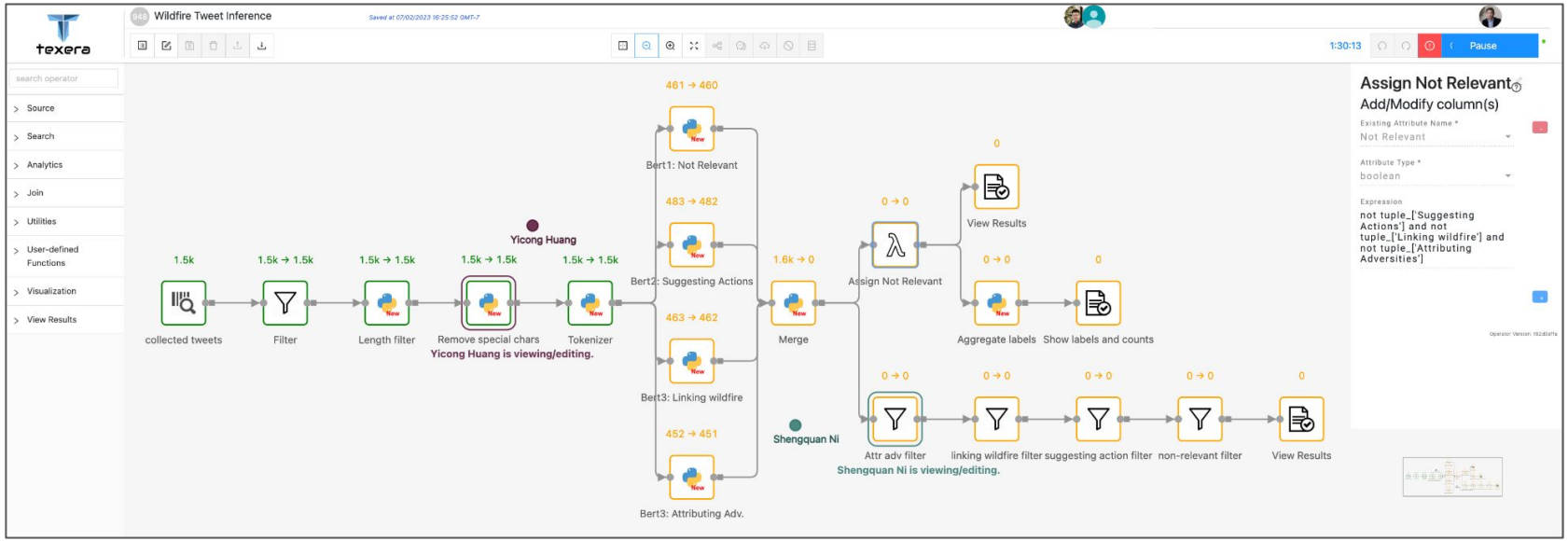
1. GUI-based workflows for non-IT people
2. Collaborative editing
3. Interactions during runtime
4. Supporting multiple languages: Python, R, …
5. Supporting machine learning (training, inference, …)
6. Scalable!

A system for collaborative data analytics

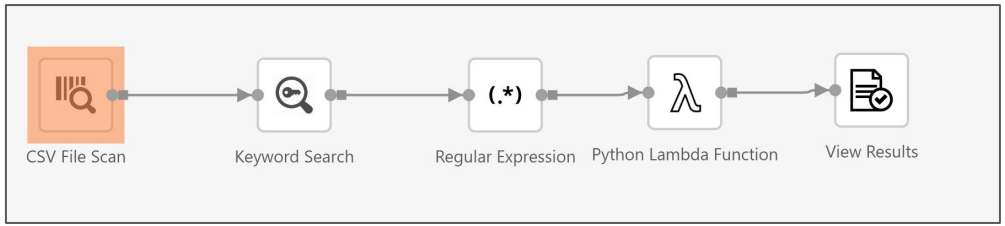
- Started in 2016; open source
- Used by many research projects
- Powerful features:
  - Cloud services
  - Version control
  - Shared editing
  - Commenting
  - Sharing
  - ...

Demo!




texera
Wildfire Tweet Inference
Pause
Source
Search
Analytics
Join
Utilities
User-defined Functions
Visualization
View Results
collected tweets
Filter
Length filter
Remove special chars
Tokenizer
Yicong Huang
Yicong Huang is viewing/editing.
Bert1: Not Relevant
Bert2: Suggesting Actions
Bert3: Linking wildfire
Bert3: Attributing Adv.
Merge
Shengquan Ni
Shengquan Ni is viewing/editing.
Assign Not Relevant
View Results
Aggregate labels
Show labels and counts
Attr adv filter
linking wildfire filter
suggesting action filter
non-relevant filter
View Results
Assign Not Relevant
Add/Modify column(s)
Existing Attribute Name *
Not Relevant
Attribute Type *
boolean
Expression
not tuple_['Suggesting Actions'] and not tuple_['Linking wildfire'] and not tuple_['Attributing Adversities']

Collaboration Feature: Version Control

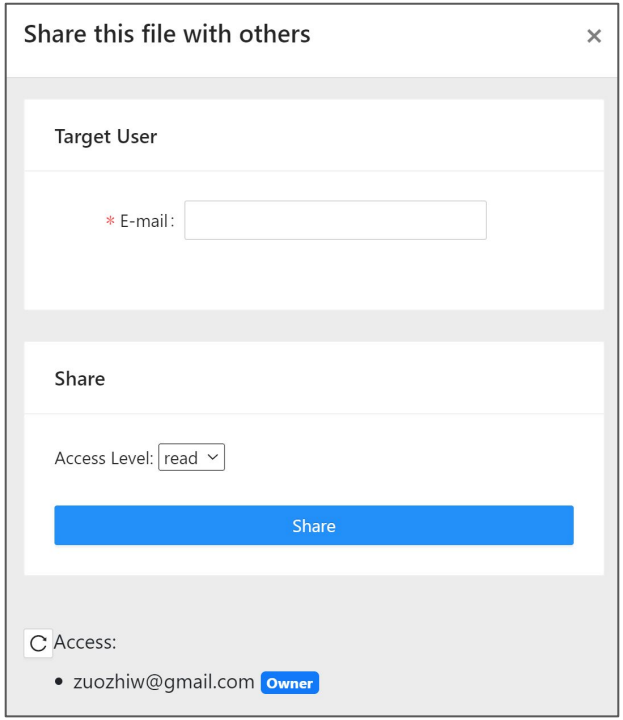


Version Diff

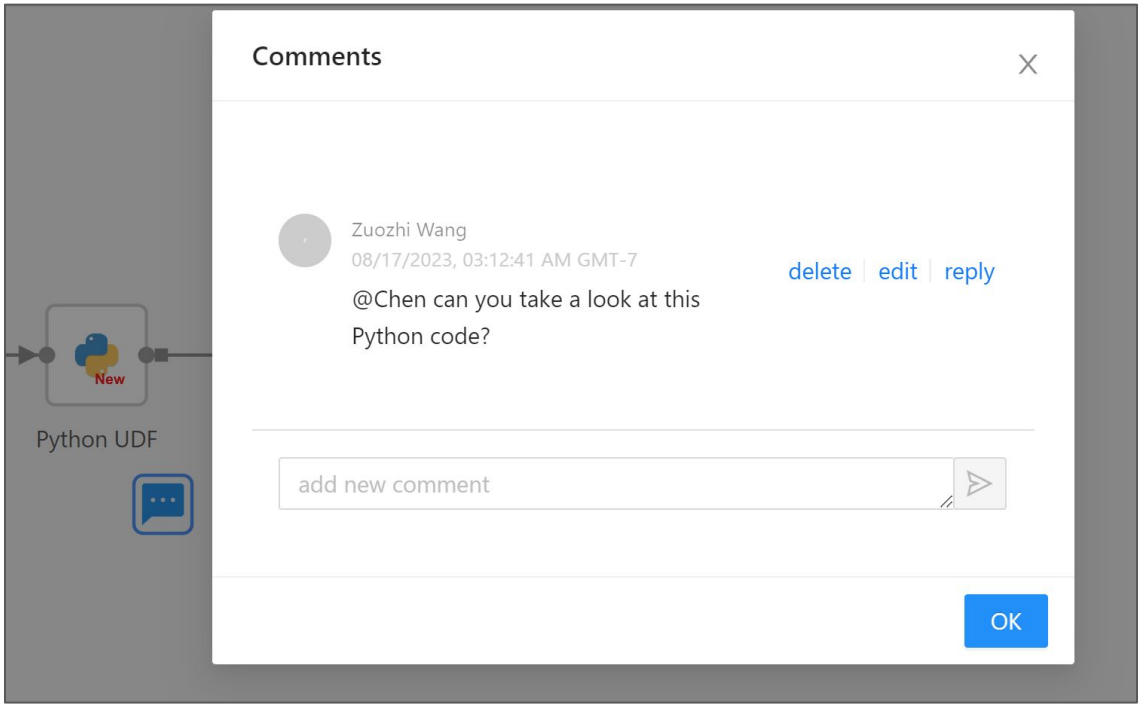
Version#	Timestamp
15	08/17/2023 02:57:54 GMT-7
14	08/17/2023 02:57:53 GMT-7
13	08/17/2023 02:57:52 GMT-7
12	08/17/2023 02:57:51 GMT-7
11	08/17/2023 02:57:50 GMT-7
10	08/17/2023 02:57:49 GMT-7
9	08/17/2023 02:57:47 GMT-7

Historical Version List

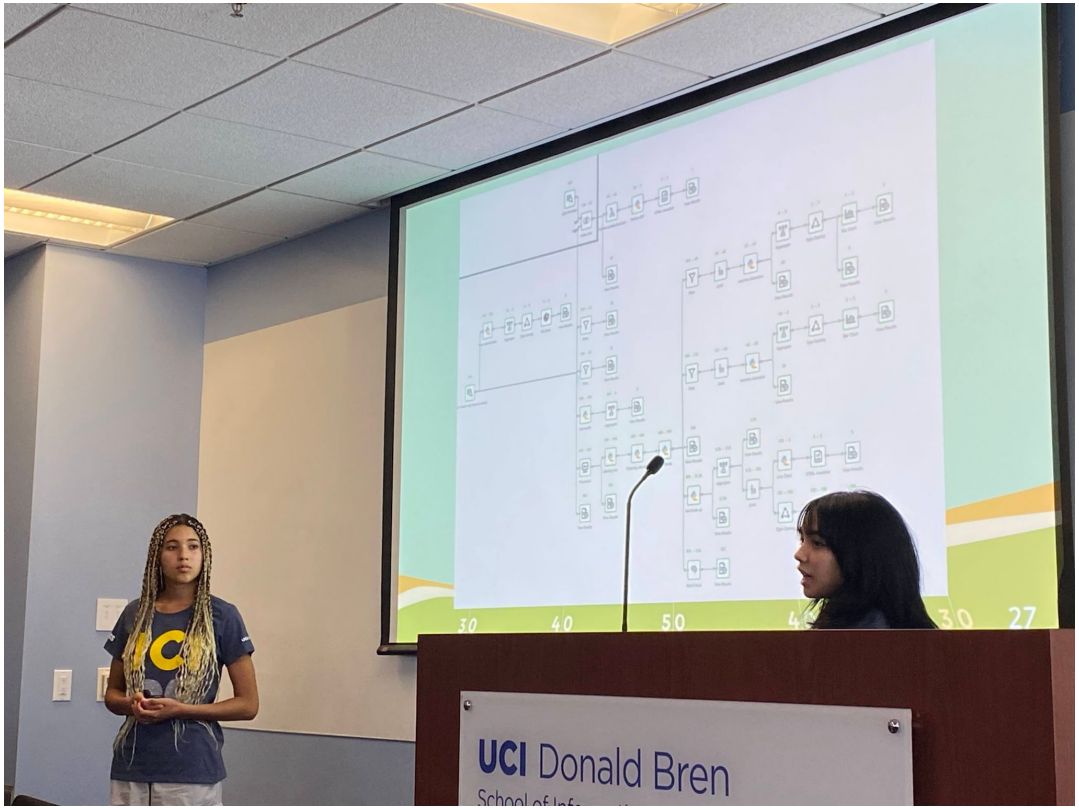
Collaboration Feature: Resource Sharing



Collaboration feature: commenting



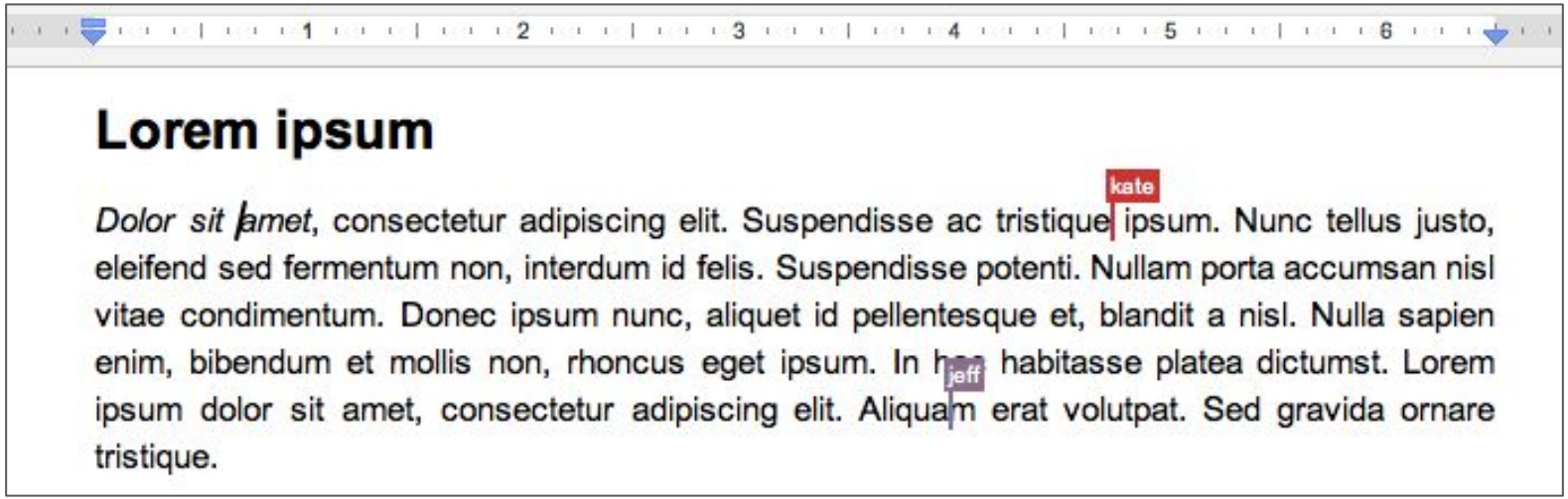
High School Students Using Texera!



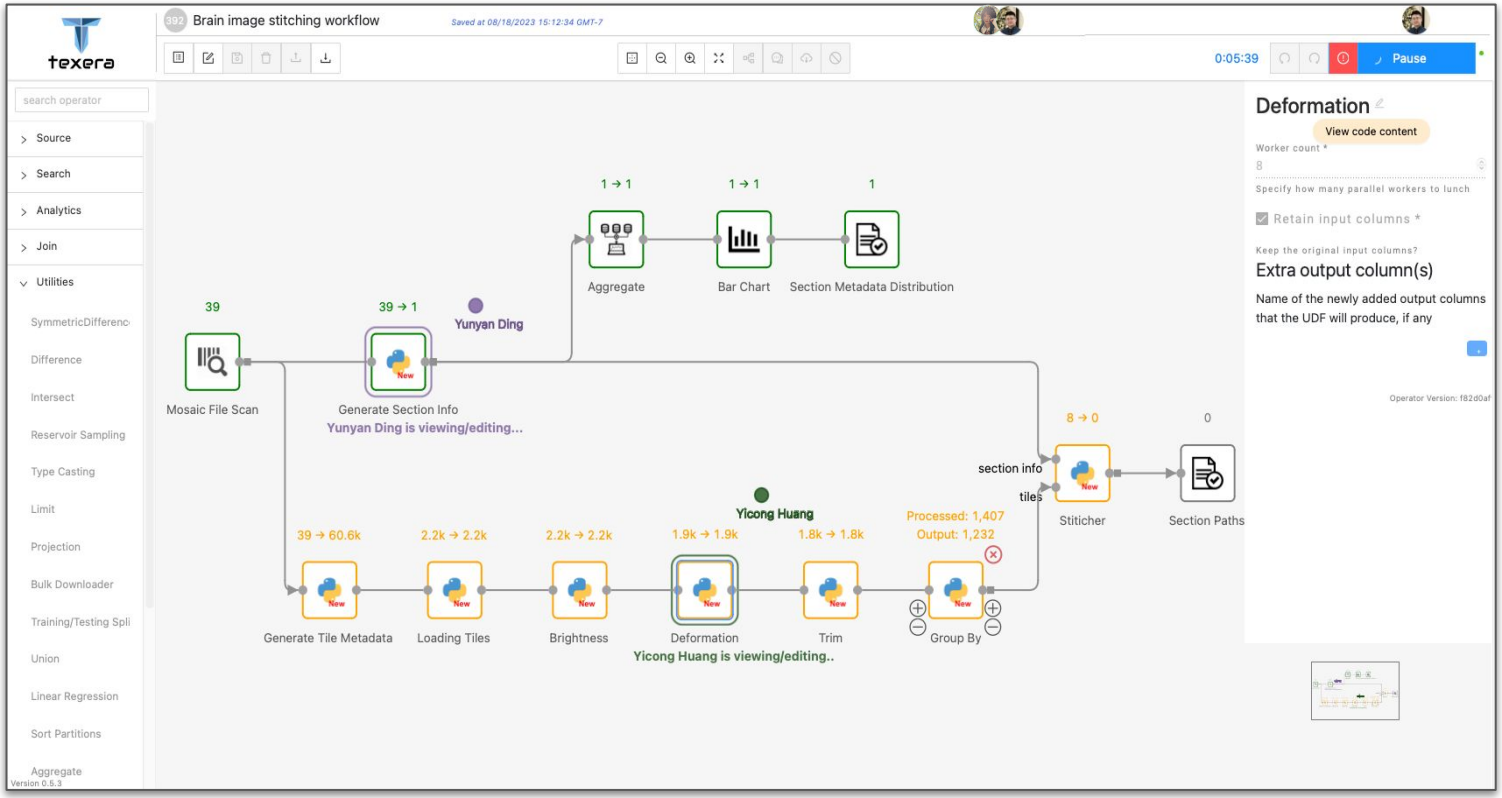
Tutorial Outline

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

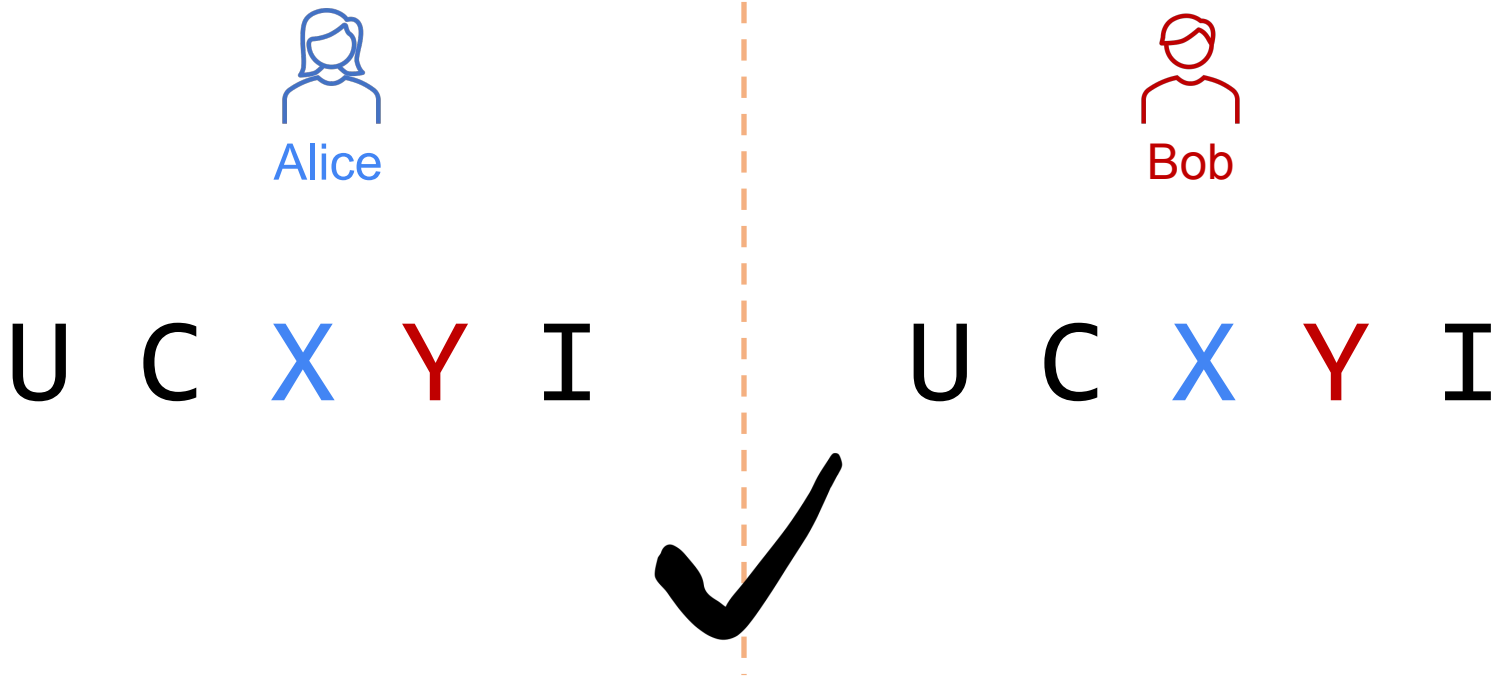
Alice

U C X I

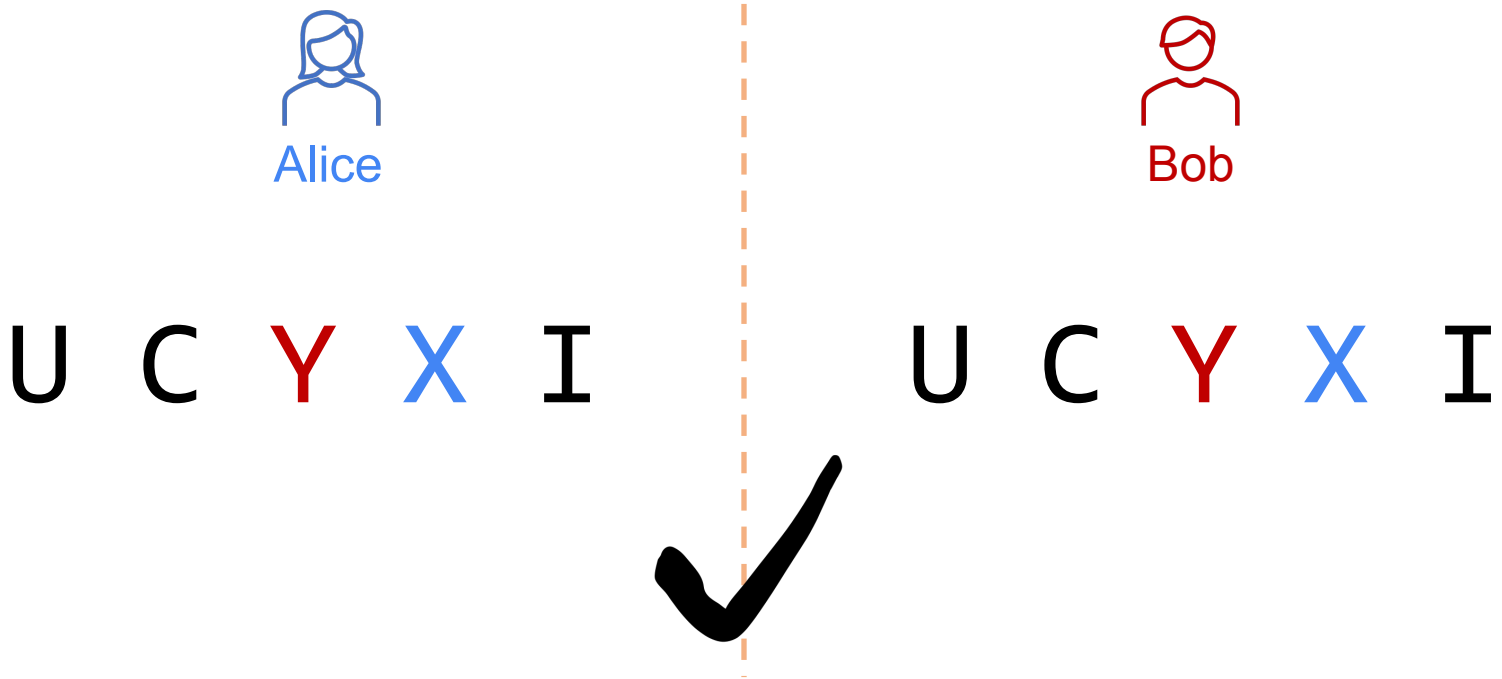
Bob

U C Y I

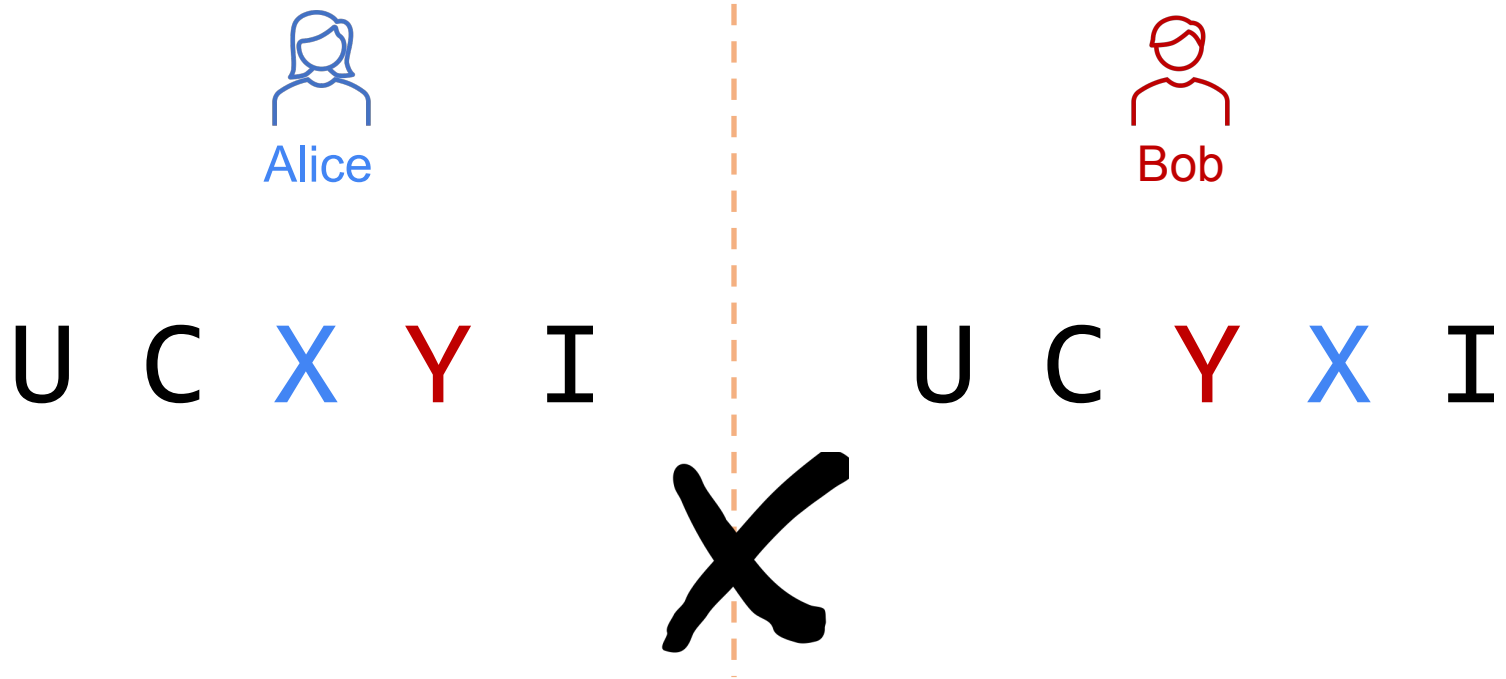
Shared Editing: Good (same result)



Shared Editing: Also good (same result)



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

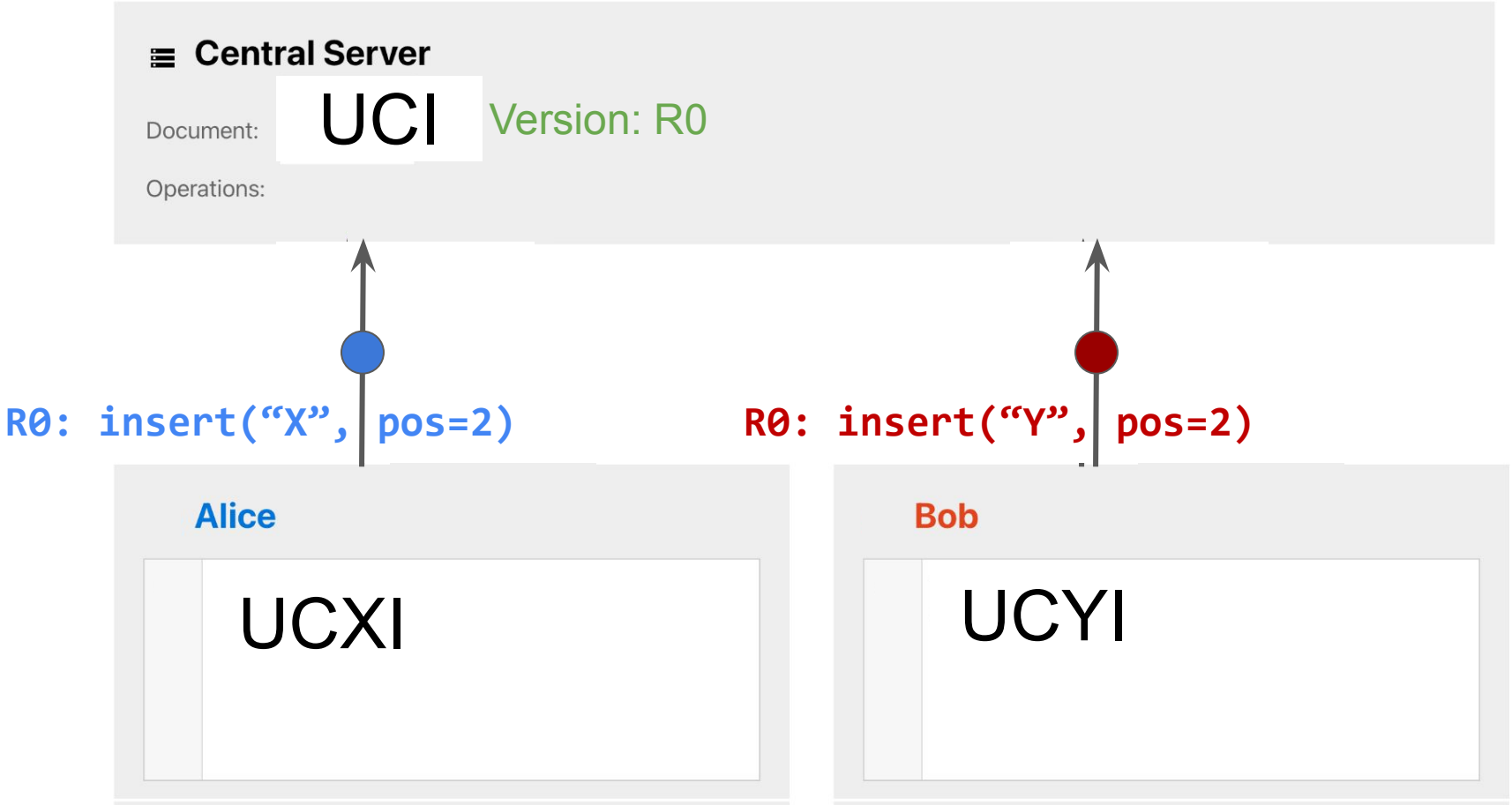
Operations:

Alice

UCI

Bob

UCI

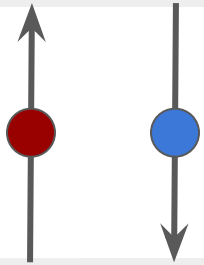

Central Server
Document:
UCI
Version: R0
Operations:
R0: insert("X", pos=2)
R0: insert("Y", pos=2)
Alice
UCXI
Bob
UCYI

Central Server

Document: UCXI

Operations:

Received Alice's request
Update R0 to R1



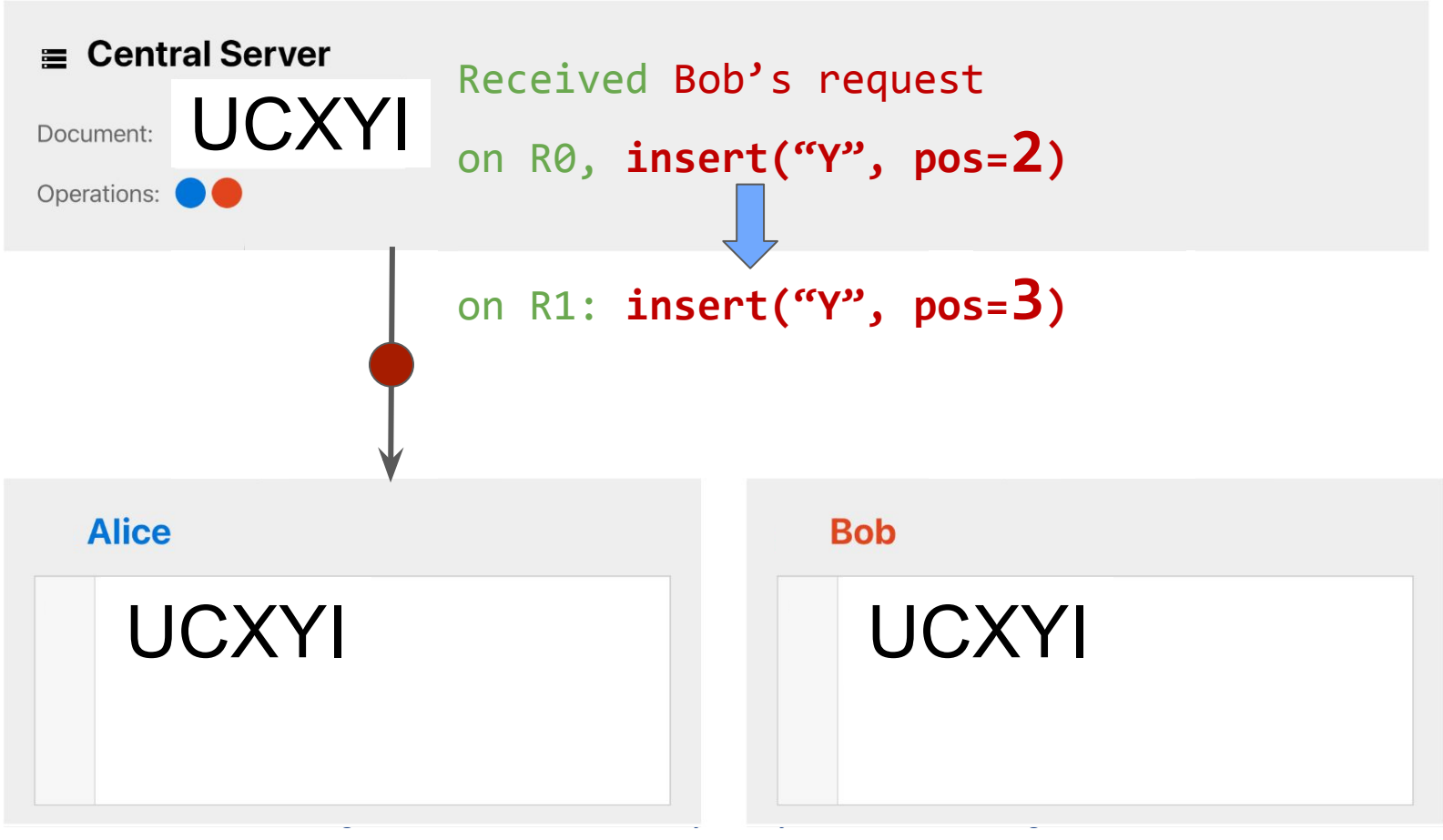
Propagate
Alice's Edit

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

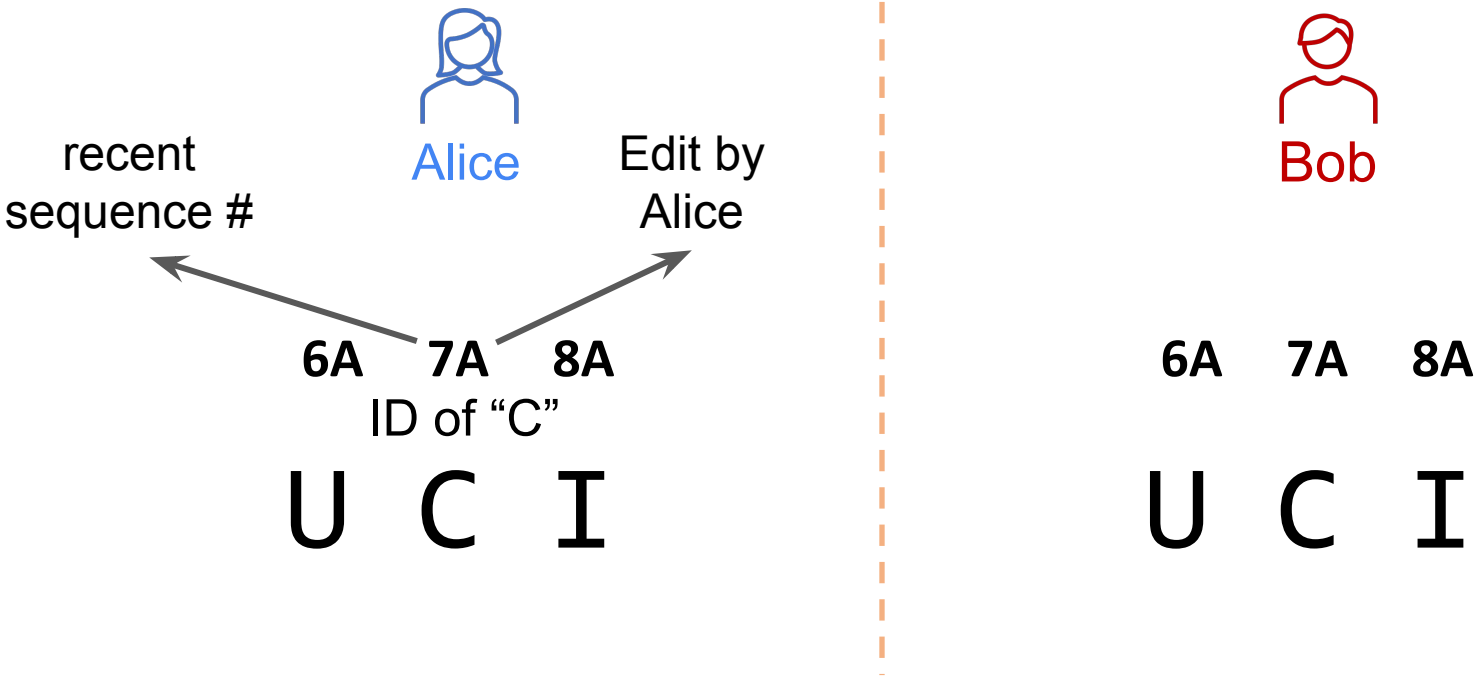
UCXYI

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

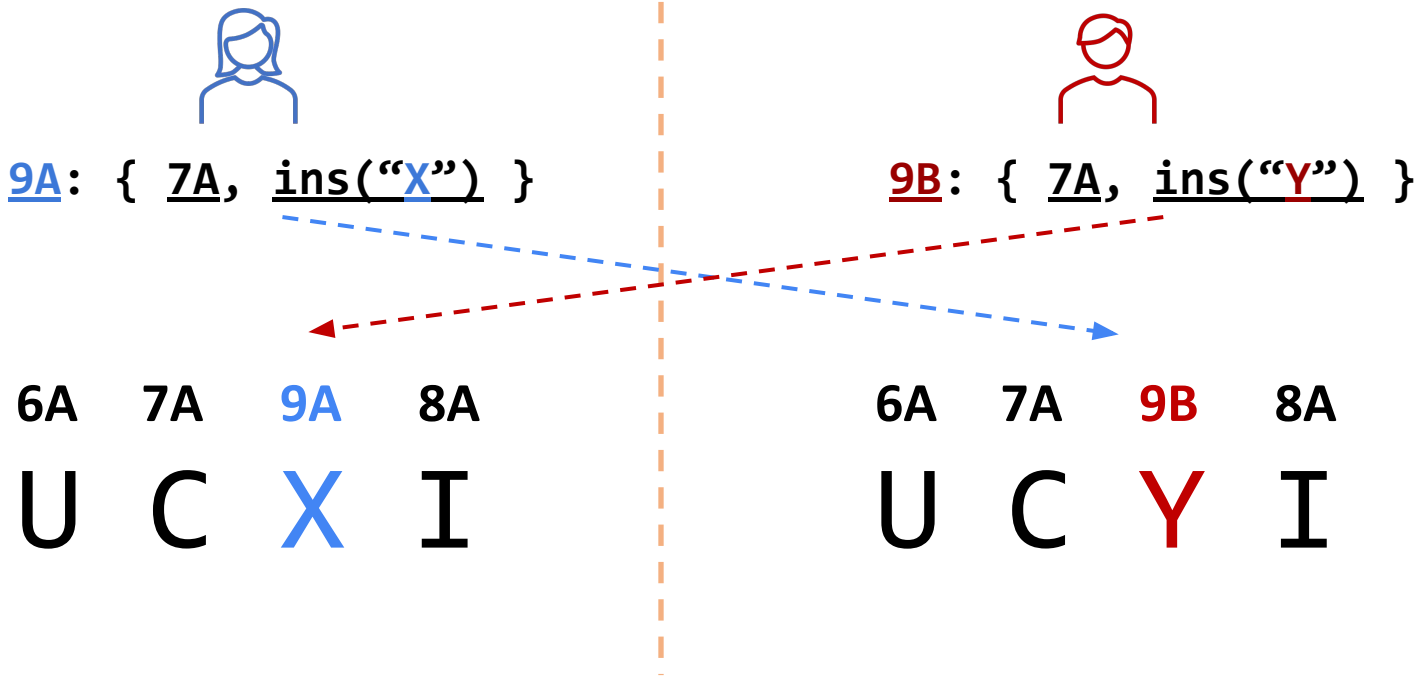
Bob

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

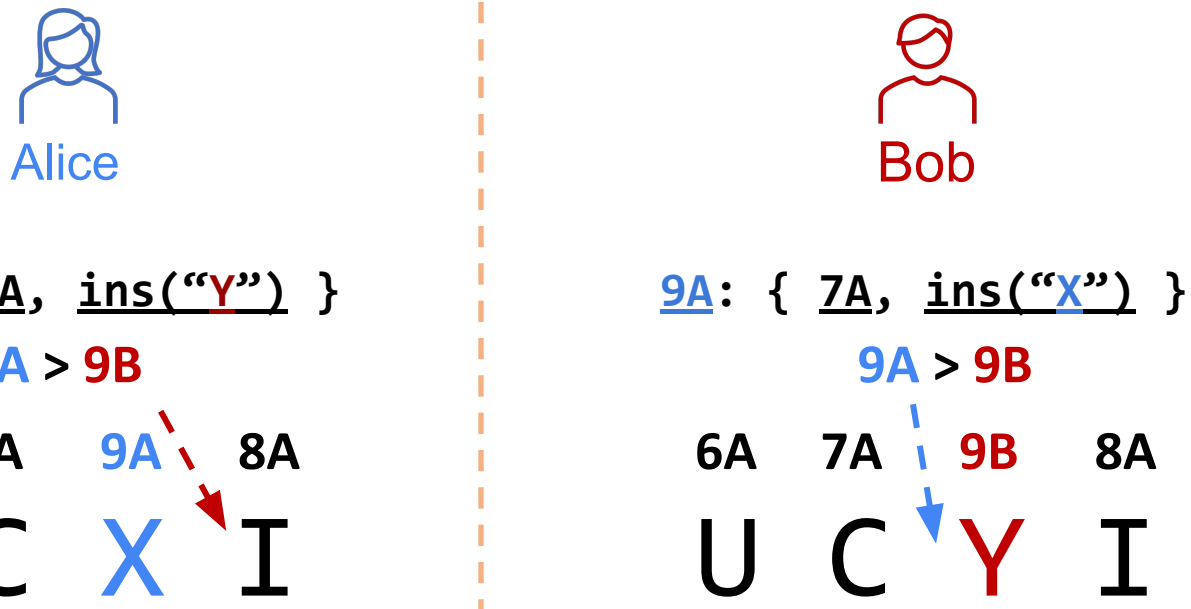
new id pos operation

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

CRDT: transmitting edits to other users



CRDT: independently comparing edits



CRDT: following same policy to decide

Alice

Bob

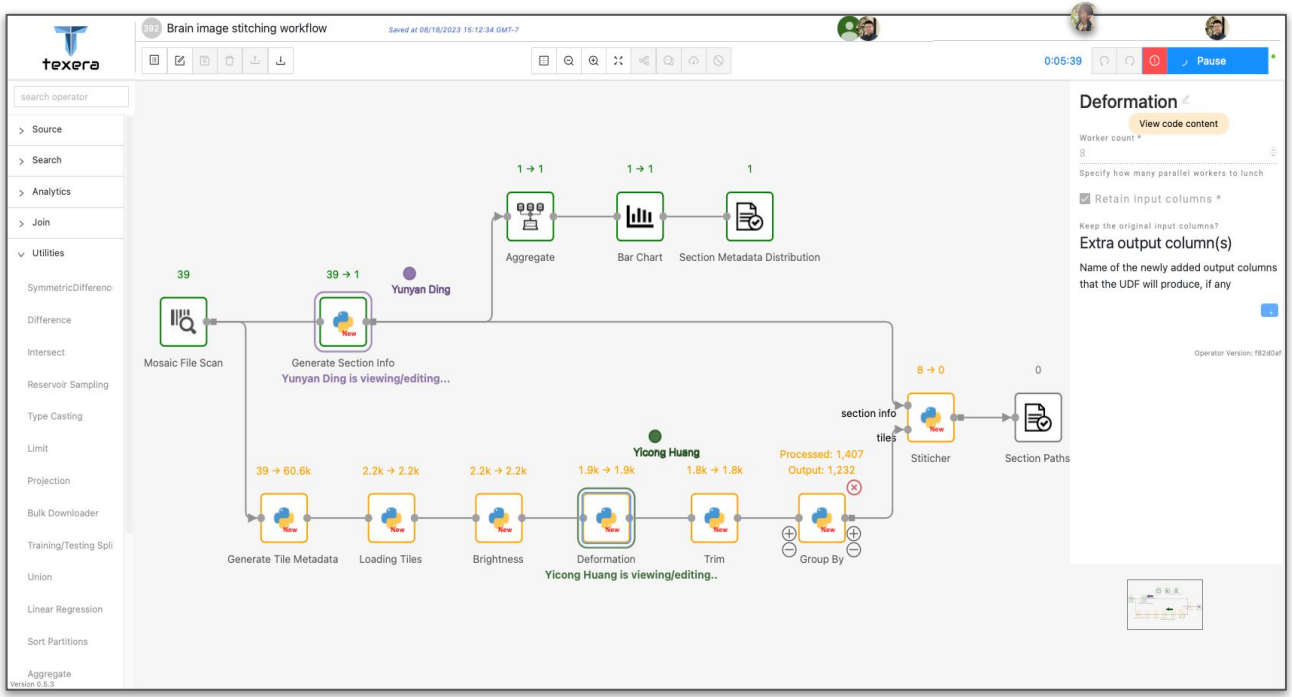
6A	7A	9A	9B	8A
U	C	X	Y	I

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

Comparison of the Two Methods

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

Workflow Shared Editing



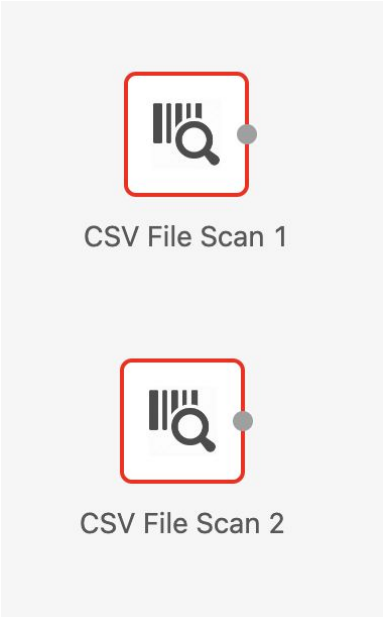
CRDT “documents” are:

- Workflow DAG
- Properties

Adding Two Operators Concurrently: no conflicts

Alice

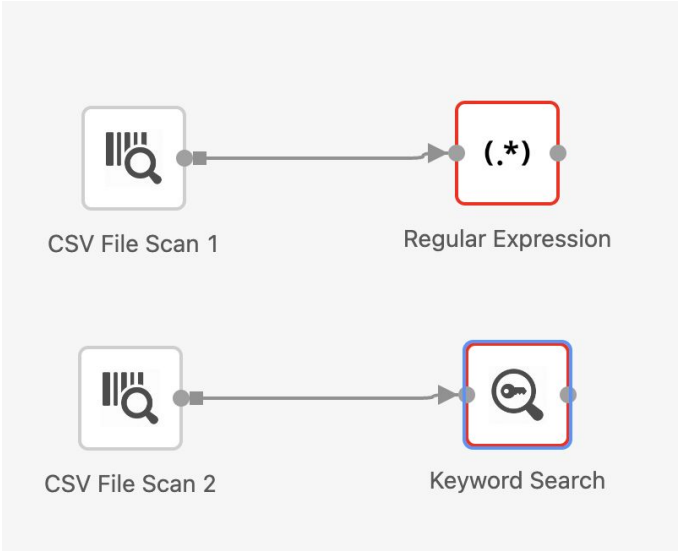
Bob



Adding Two Operators Concurrently: no conflicts

Alice

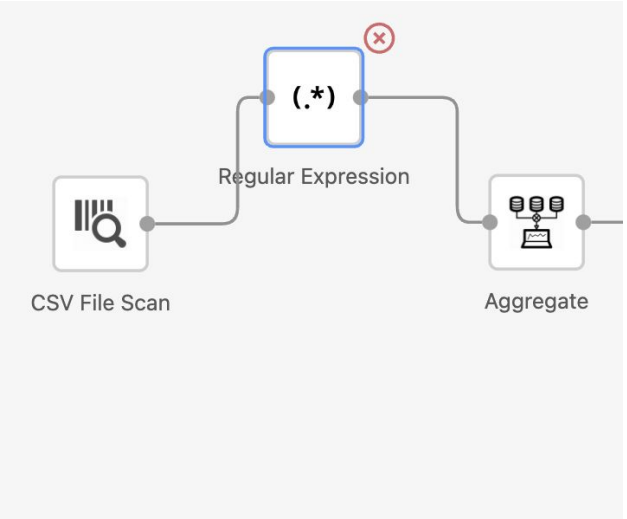
Bob



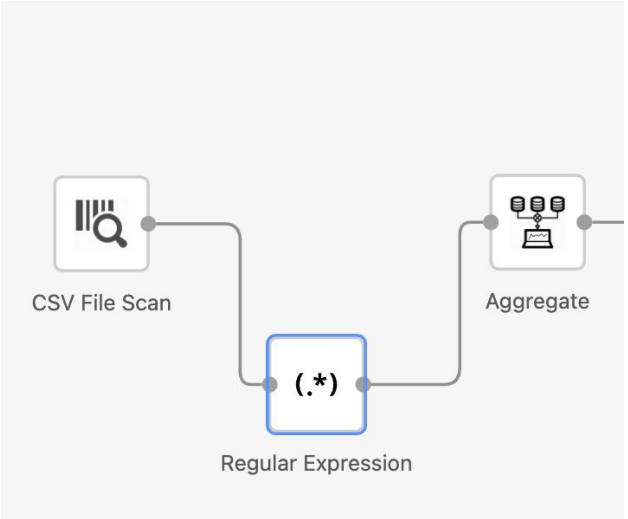
Moving Operators to Different Positions: conflicts

CRDT resolves it based on user ID

Alice

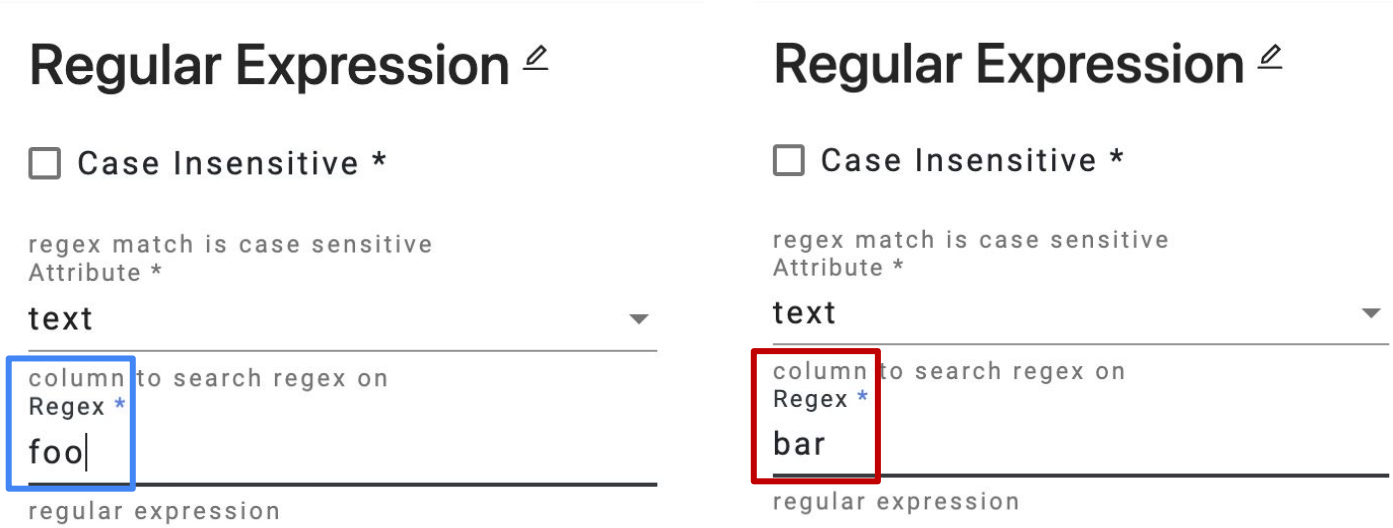


Bob



Changing Operator's Properties: conflicts

CRDT resolves it based on user ID

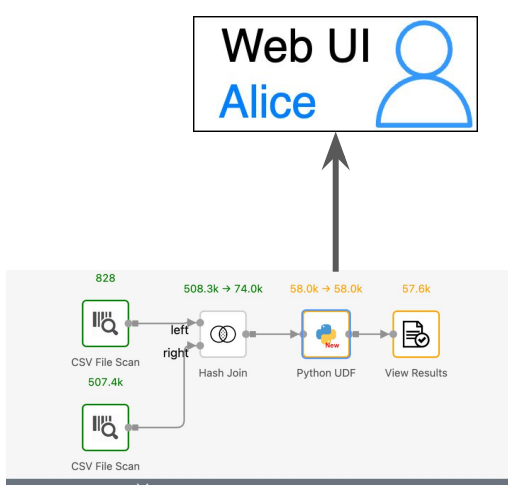


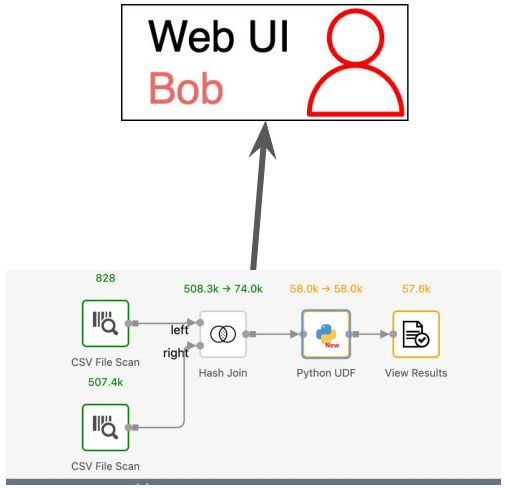
Tutorial Outline

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

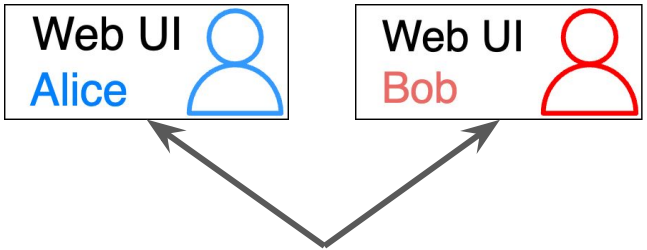
If users have their own executions, then collaboration is not possible.

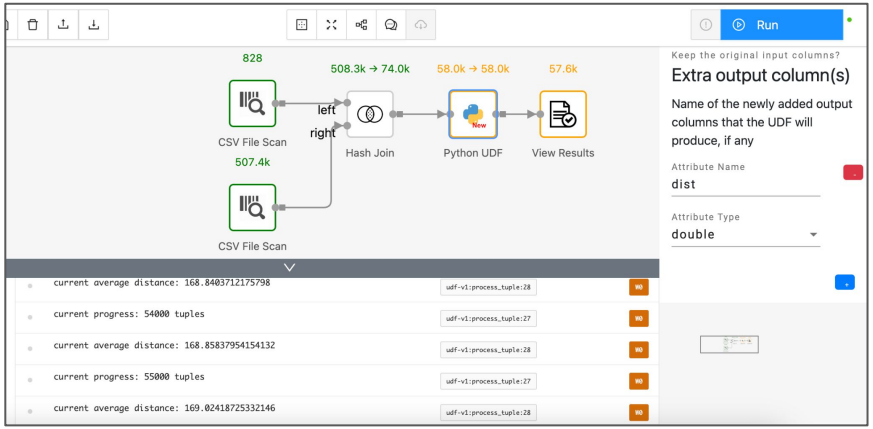




Benefits of Shared Execution

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

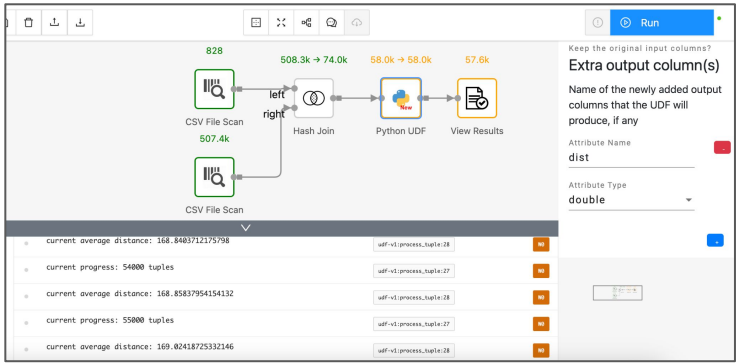




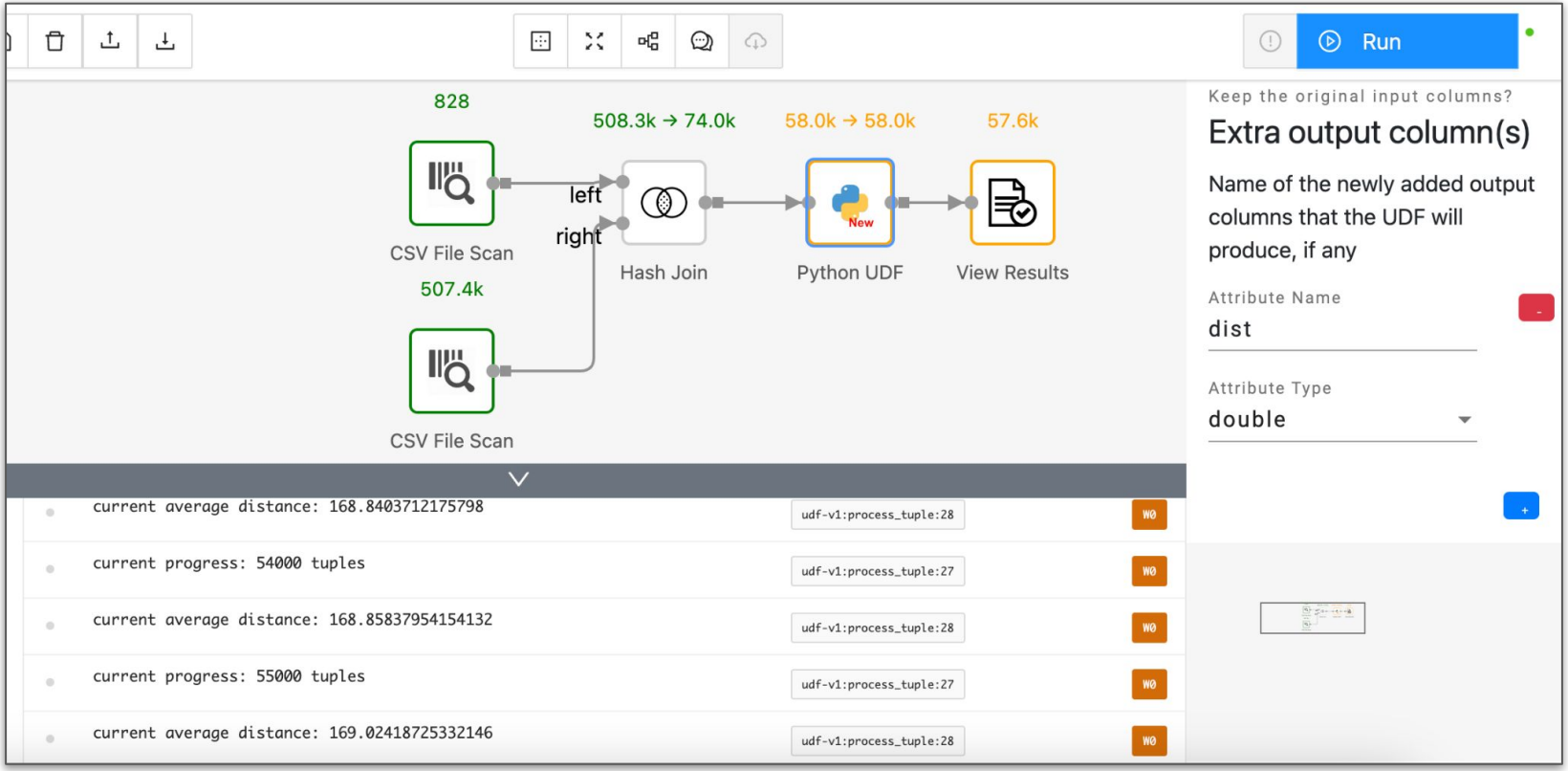
Benefits of Shared Execution (cont.)

Shared Execution State

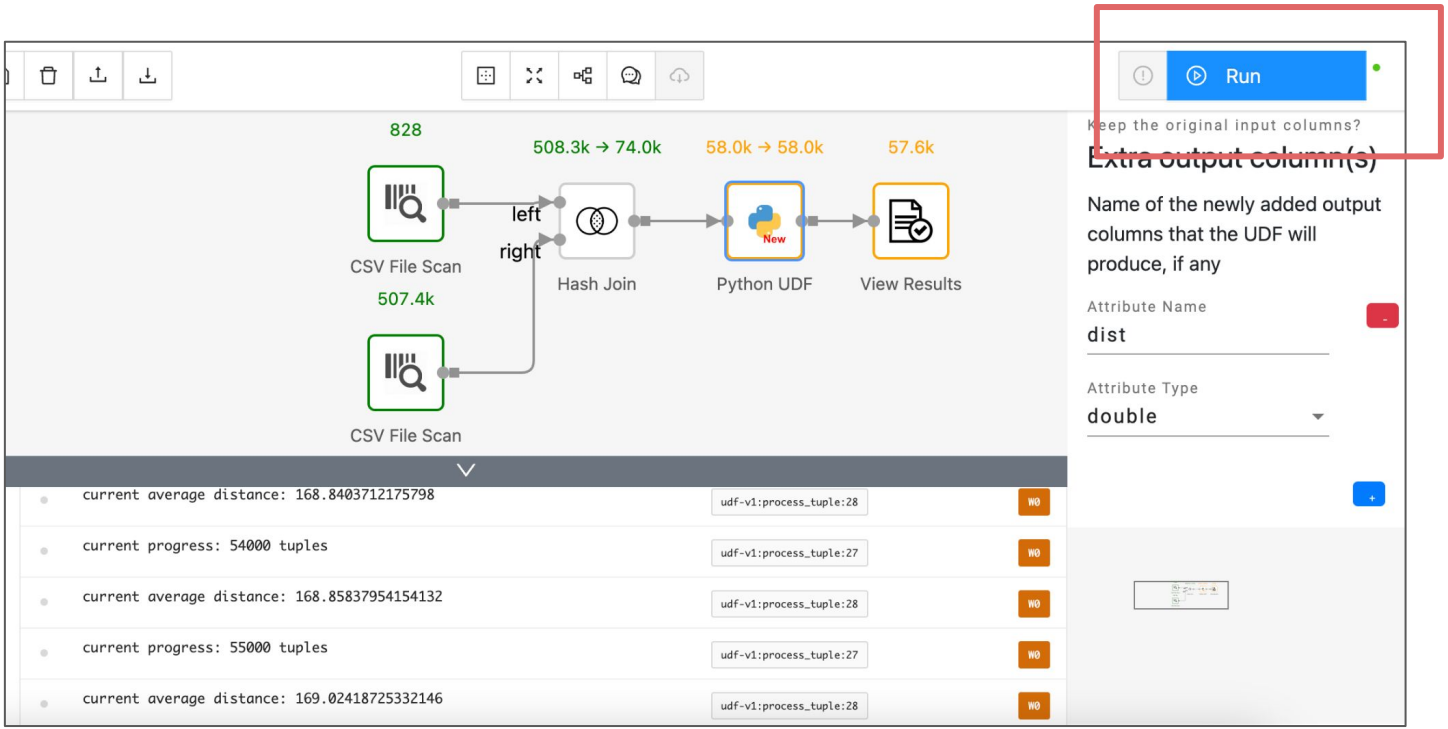
Invite new collaborators at runtime



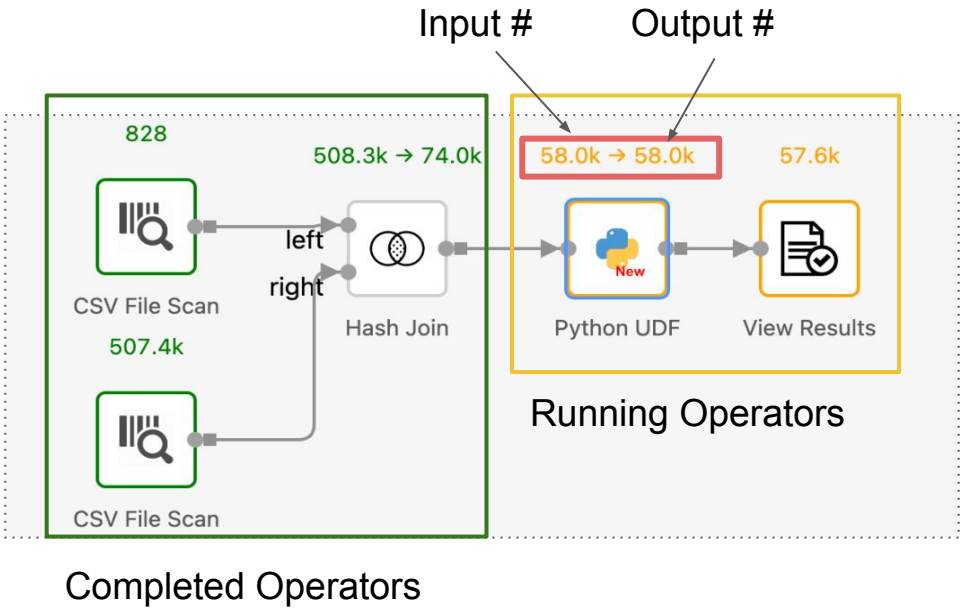
Shared Execution: Demo



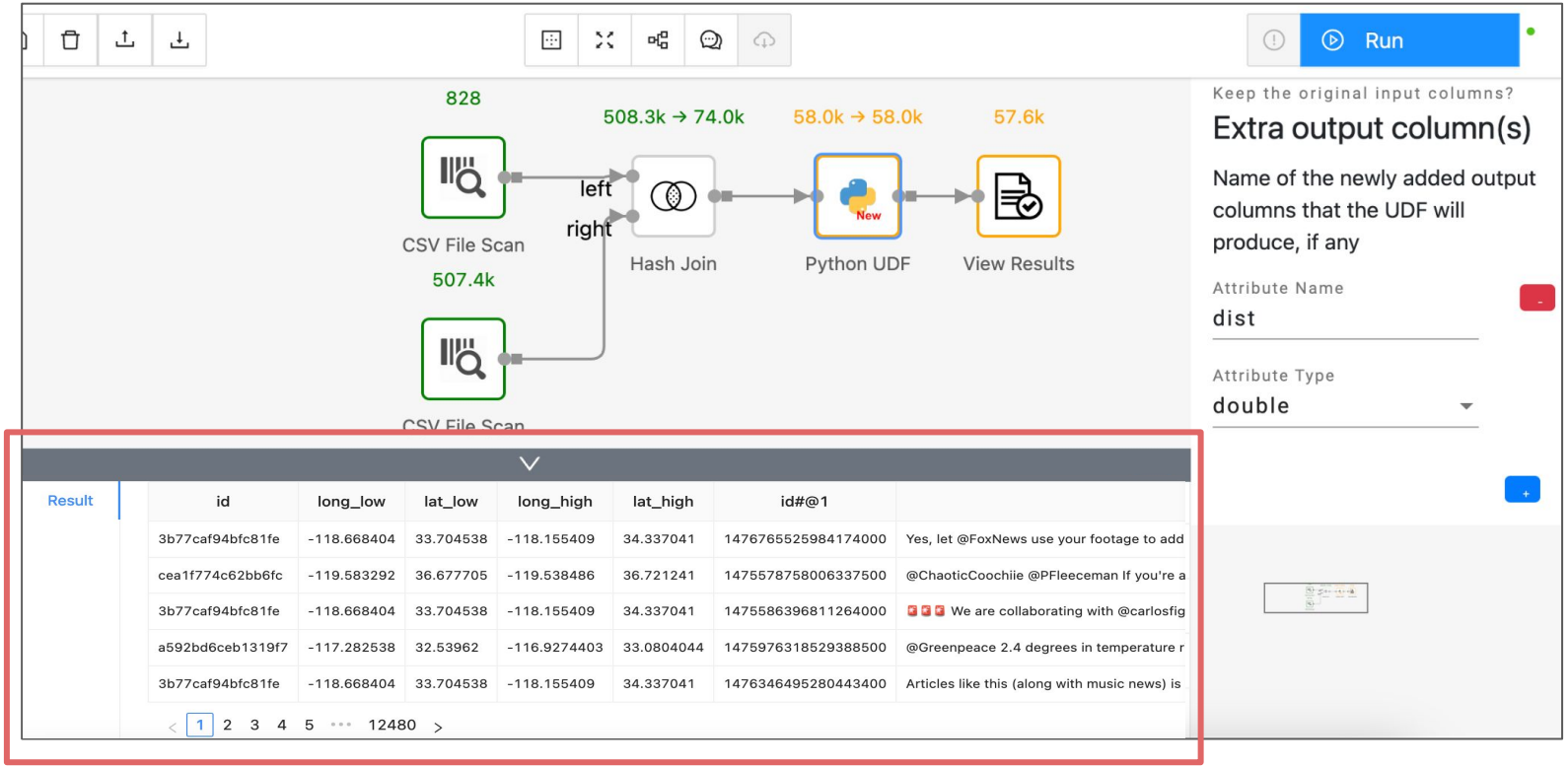
Workflow Status: Running/Paused/Failed/Completed



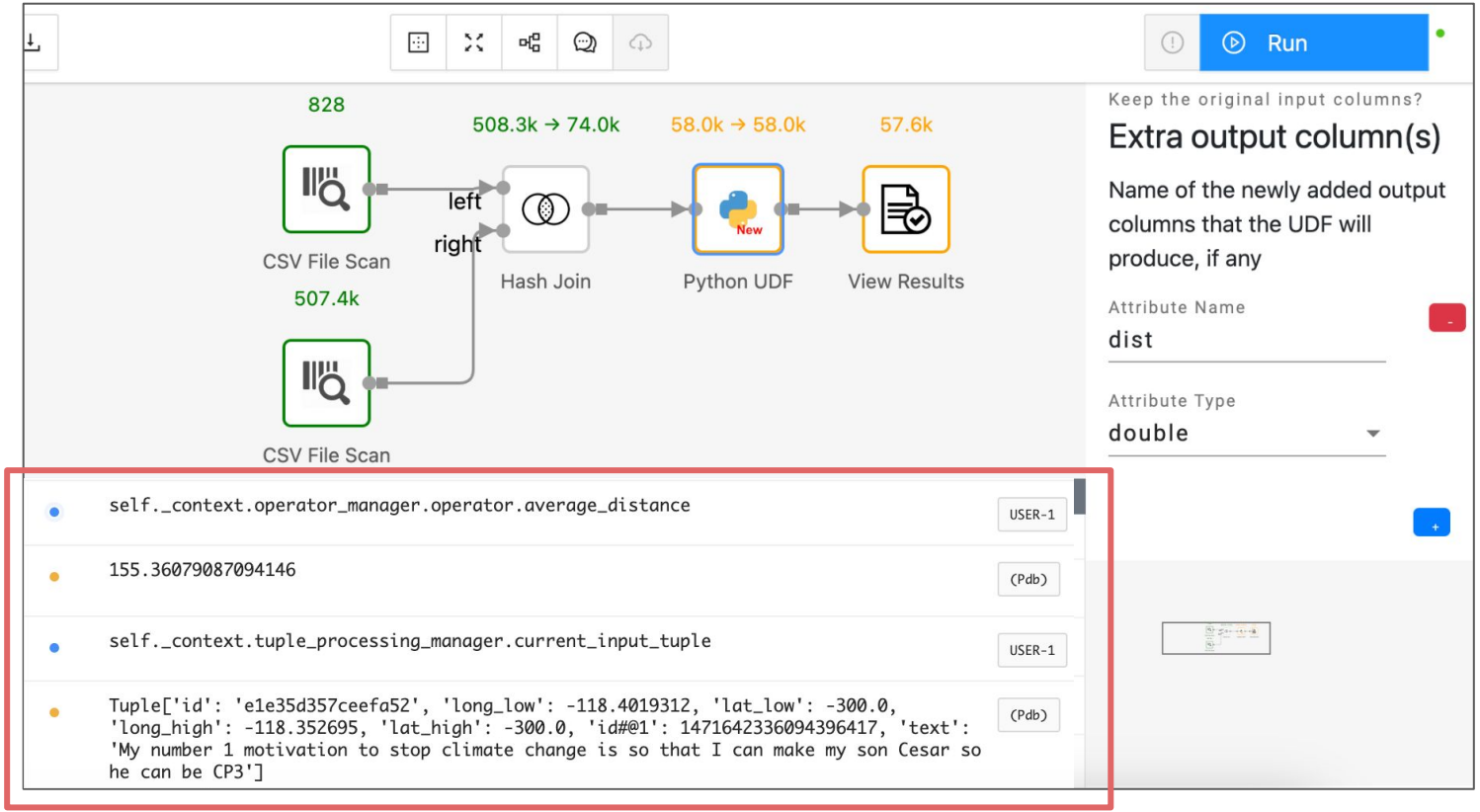
Operator Status: Input/Output Tuple Count



Execution Results



User Interaction History

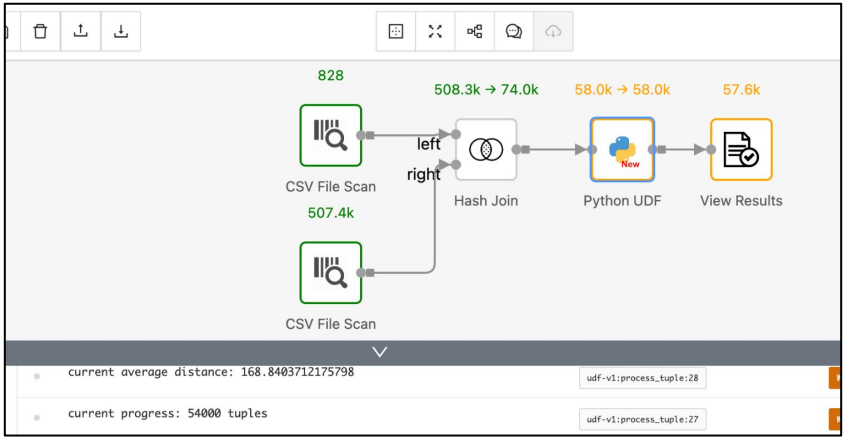


Shared Execution: How to Share?

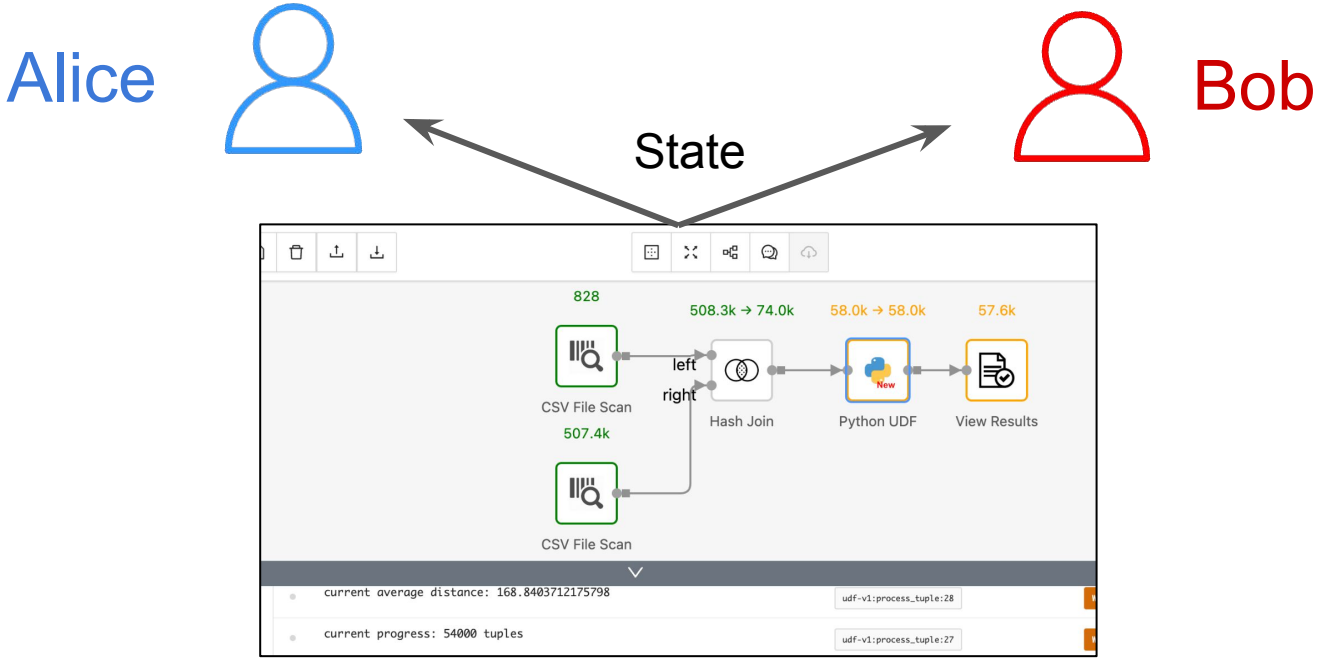
Alice

Bob

Execution State

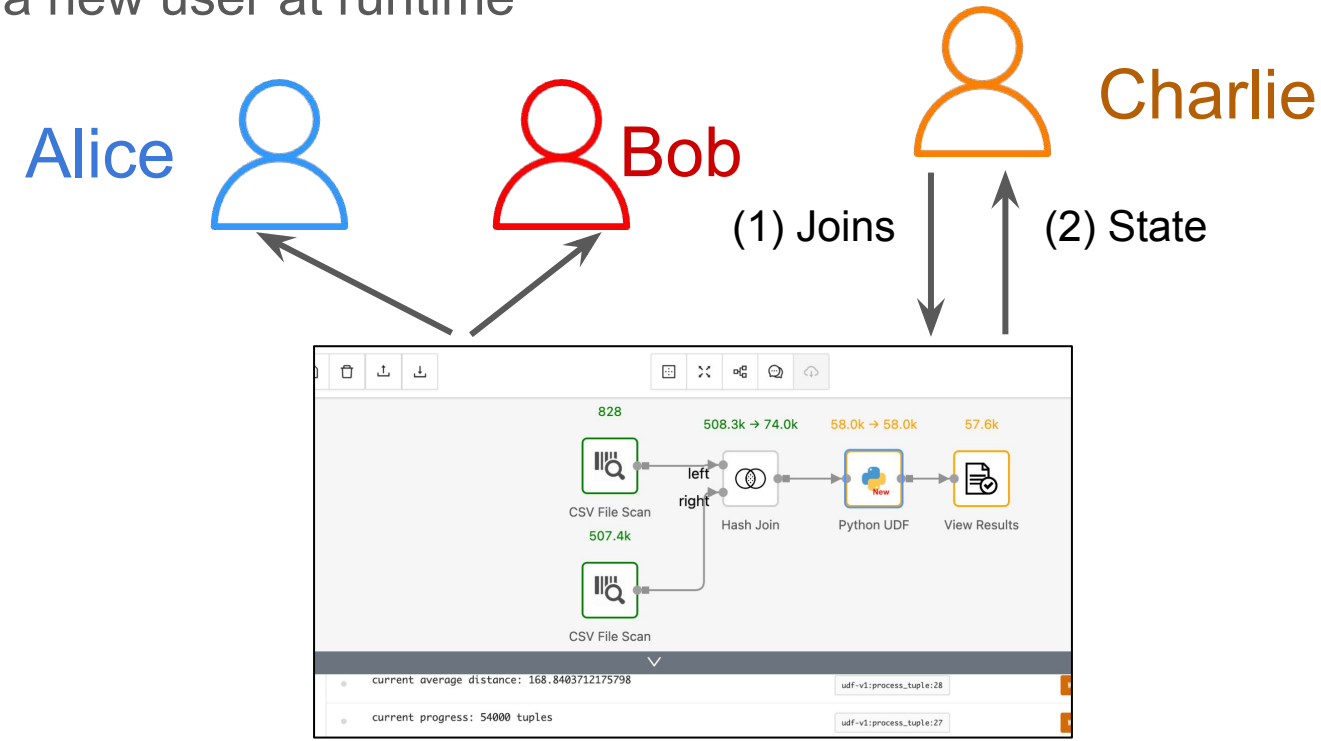


Approach 1: Periodically Broadcast State to All Frontends



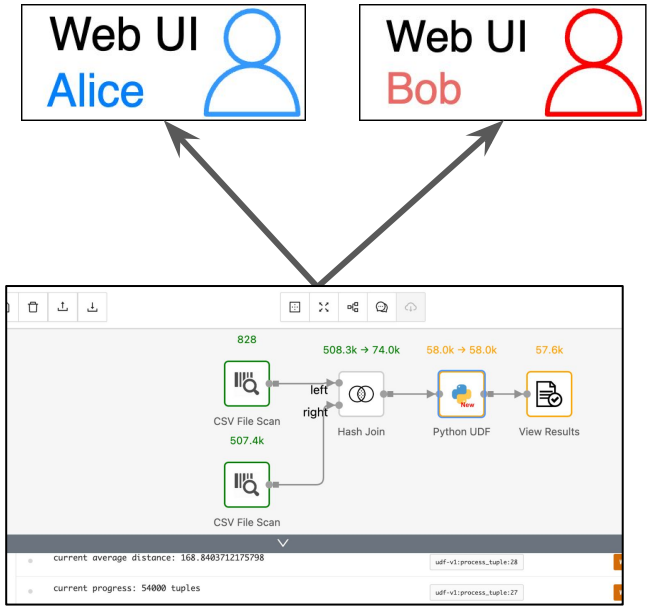
Approach 1: Periodically Broadcast State to All Frontends

Adding a new user at runtime

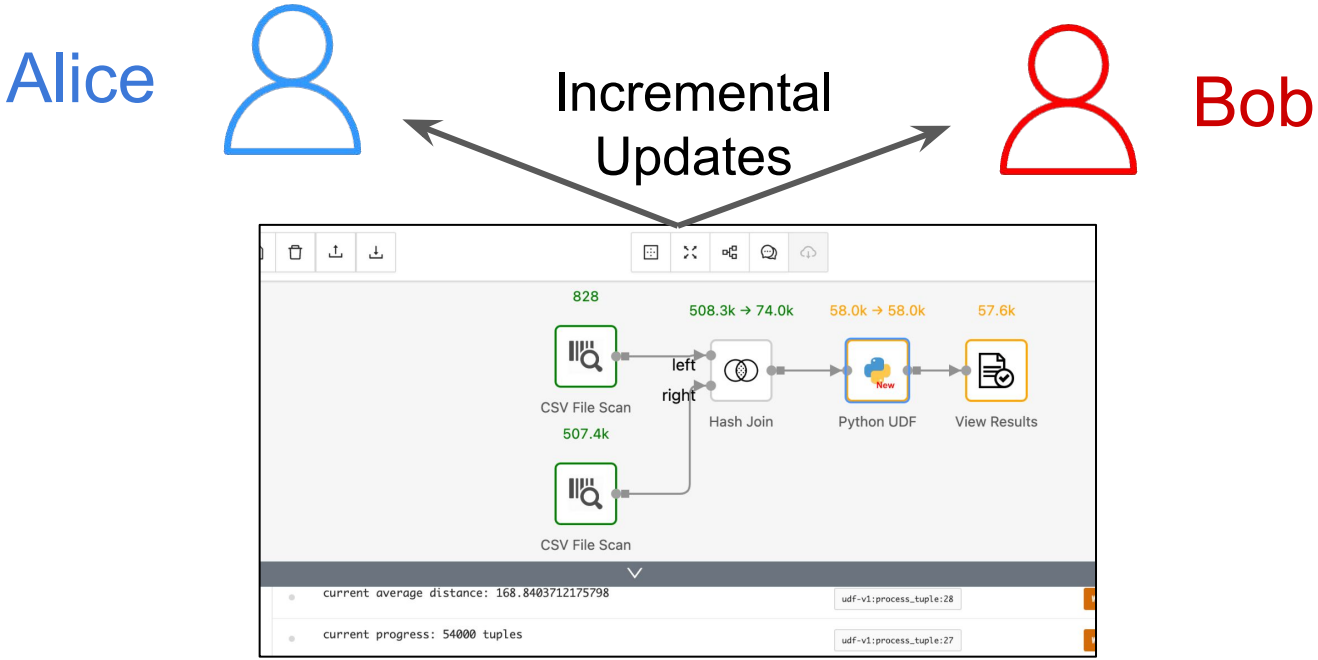


Problems with Approach 1

- Interaction history: growing
- Execution results: large

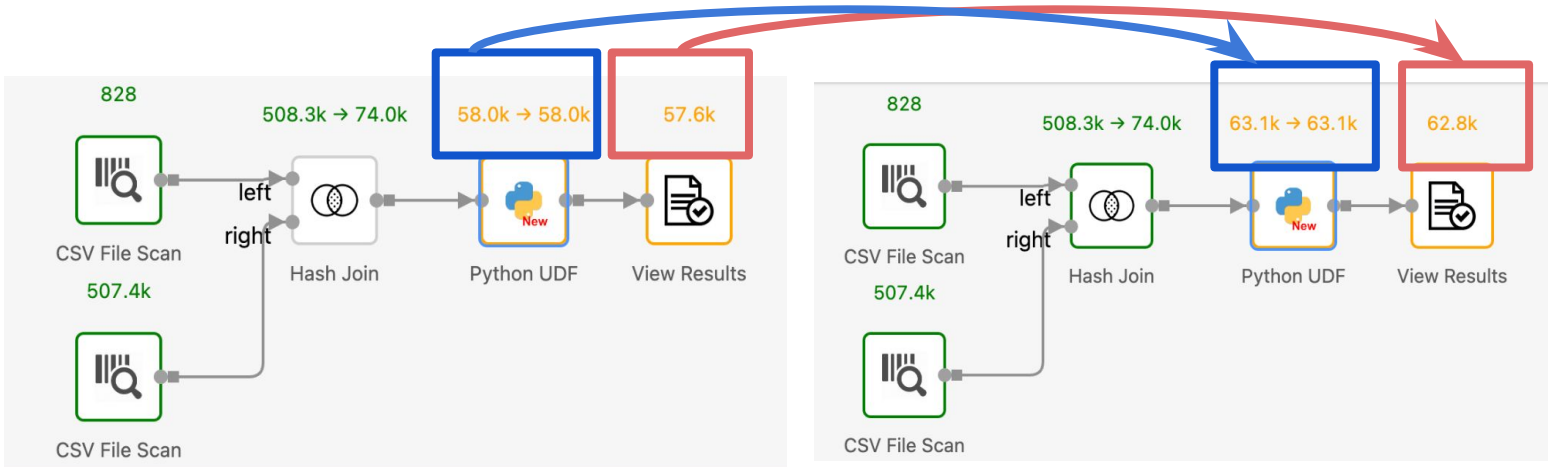


Approach 2: Periodically Send Incremental Updates



Approach 2: Periodically Send Incremental Updates

An example

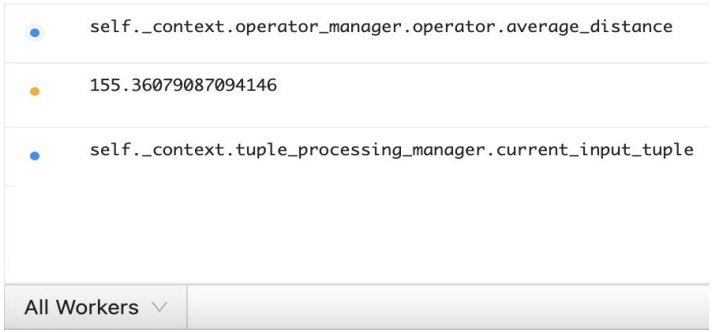


Before

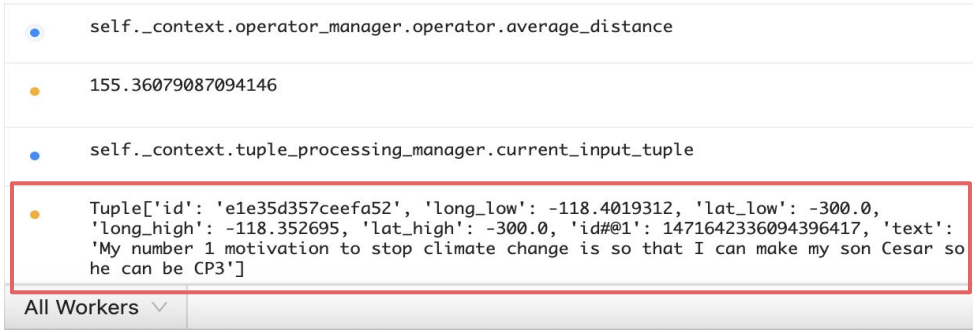
After

Approach 2: Periodically Send Incremental Updates

Another example



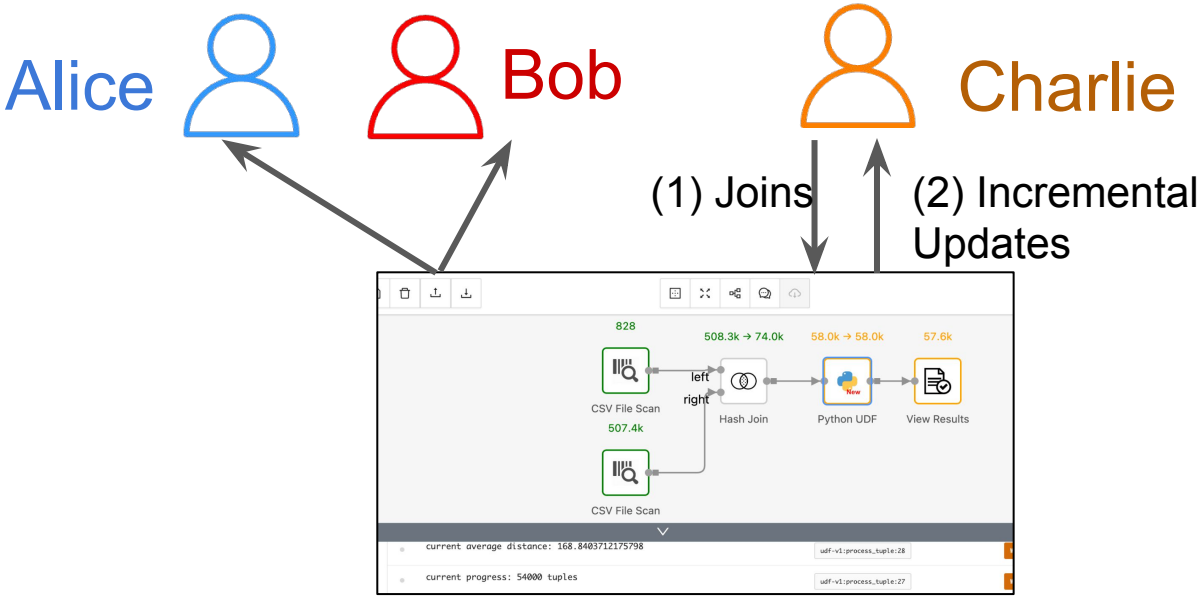
Before



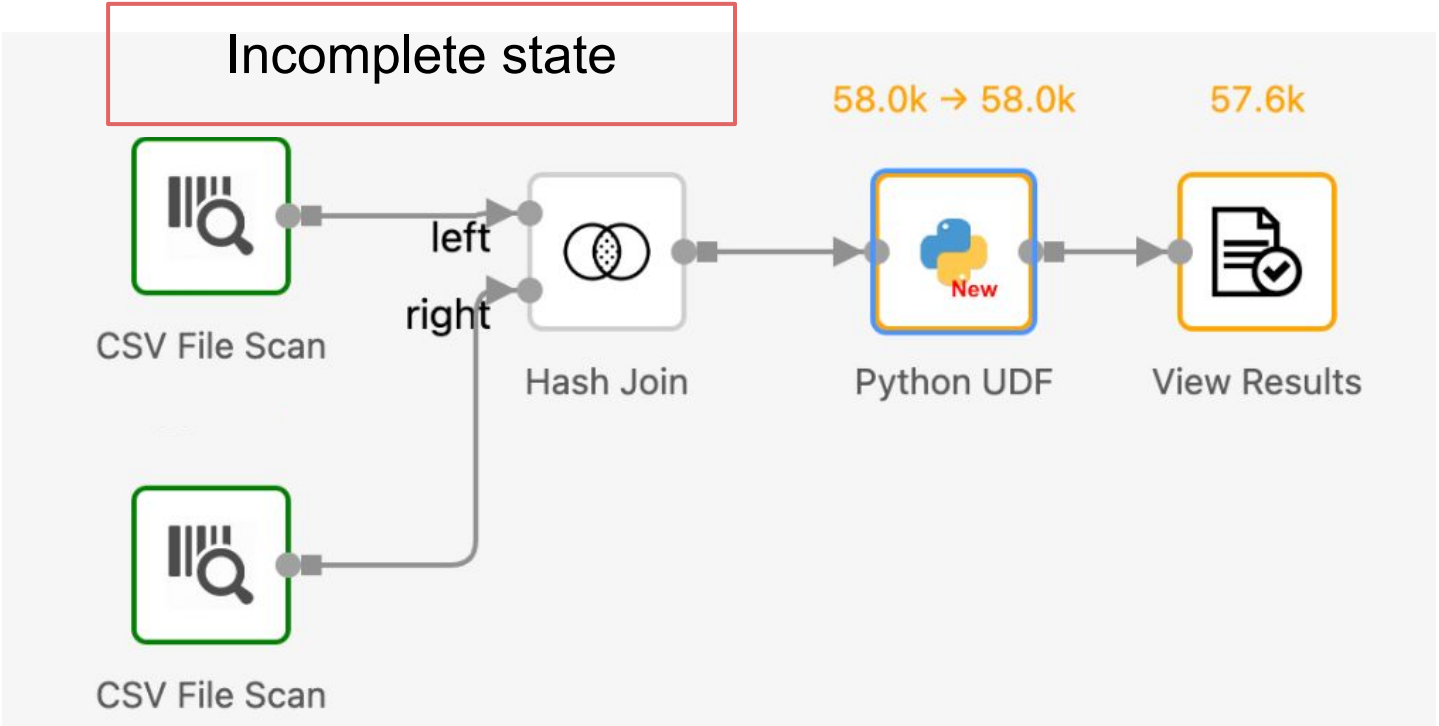
After

Approach 2: Periodically Send Incremental Updates

Adding a new user



Problem with Approach 2 When Adding a New User



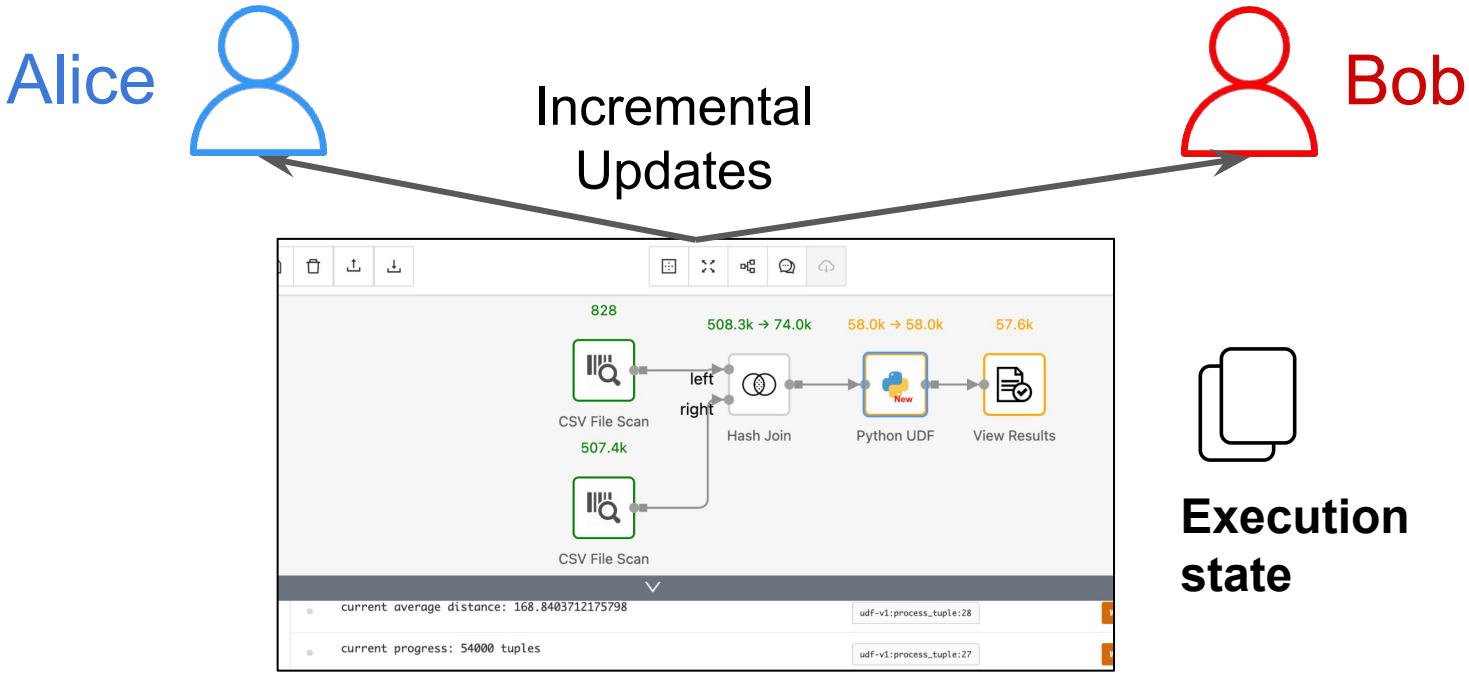
Problem with Approach 2 When Adding a New User

Incomplete interaction history

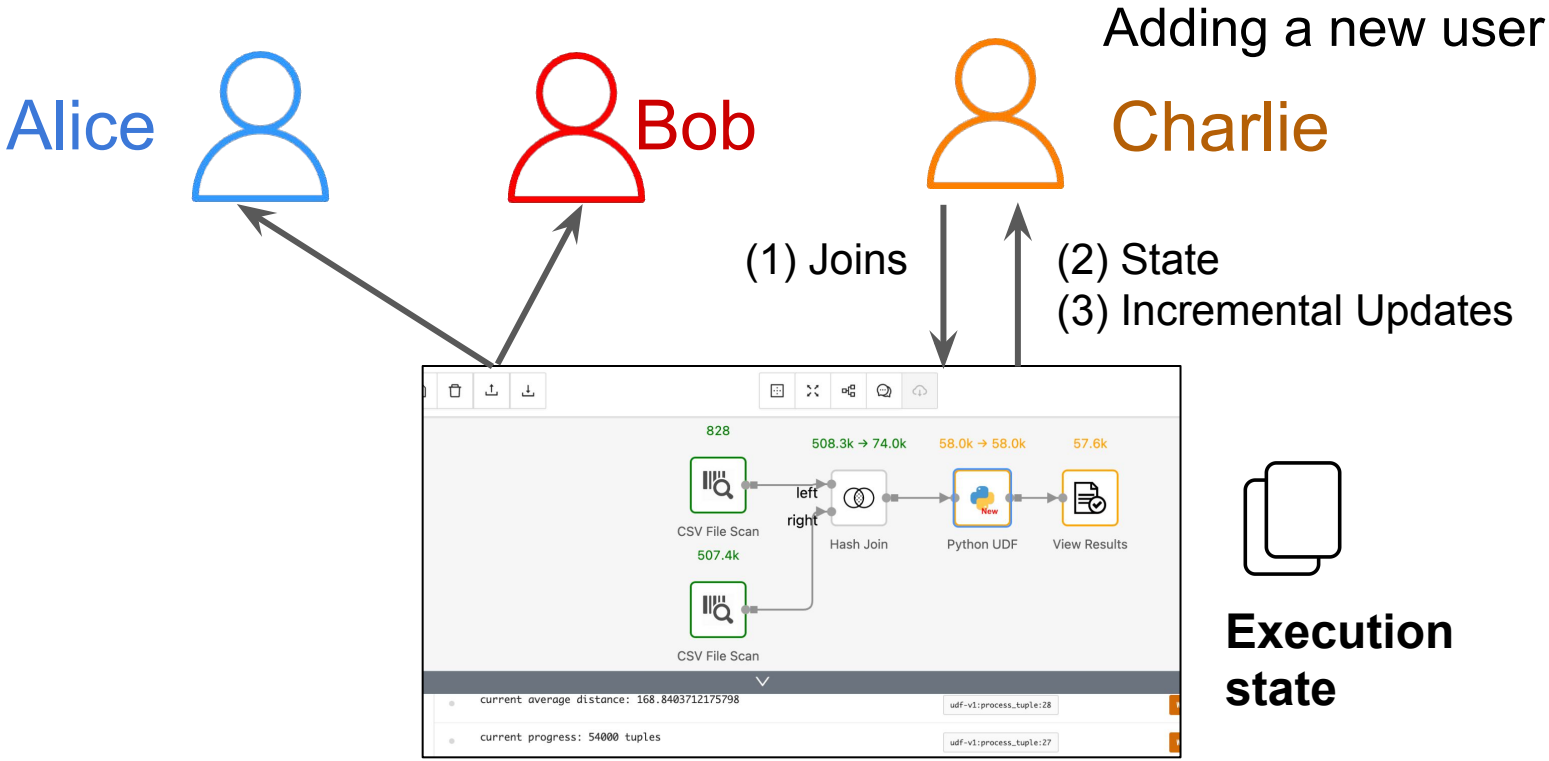
```
Tuple['id': 'e1e35d357ceefa52', 'long_low': -118.4019312, 'lat_low': -300.0,
'long_high': -118.352695, 'lat_high': -300.0, 'id#@1': 1471642336094396417, 'text':
'My number 1 motivation to stop climate change is so that I can make my son Cesar so
he can be CP3']
```

All Workers

Approach 3: Keep State and Send Incremental Updates



Approach 3: Keep State and Send Incremental Updates



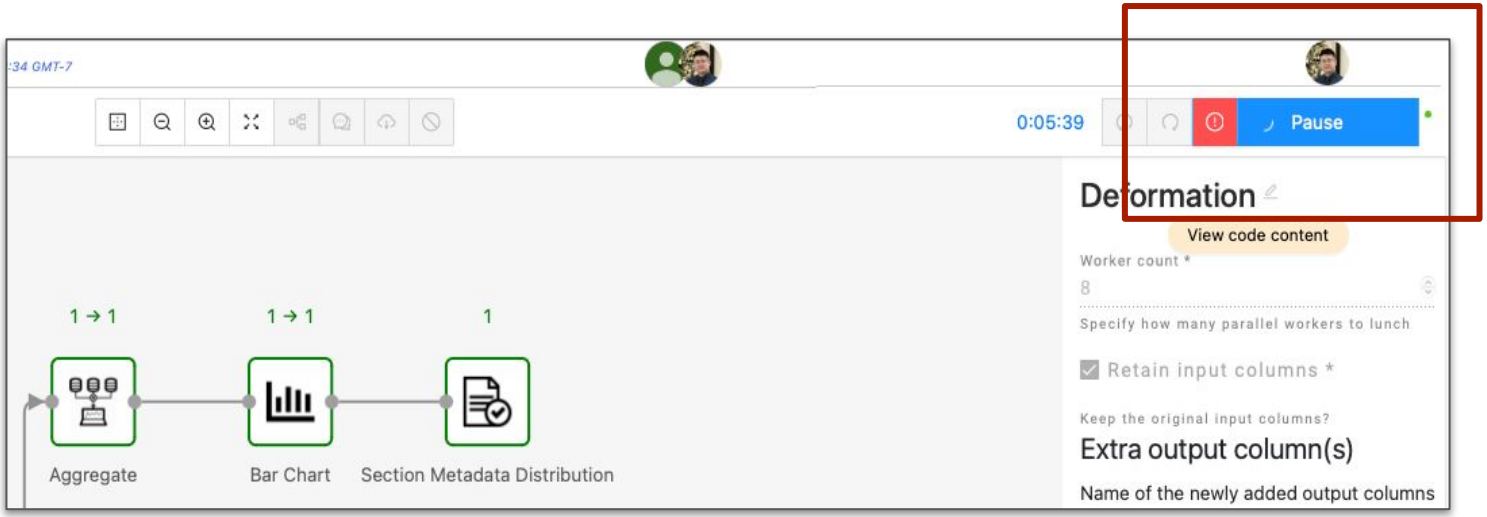
Tutorial Outline

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

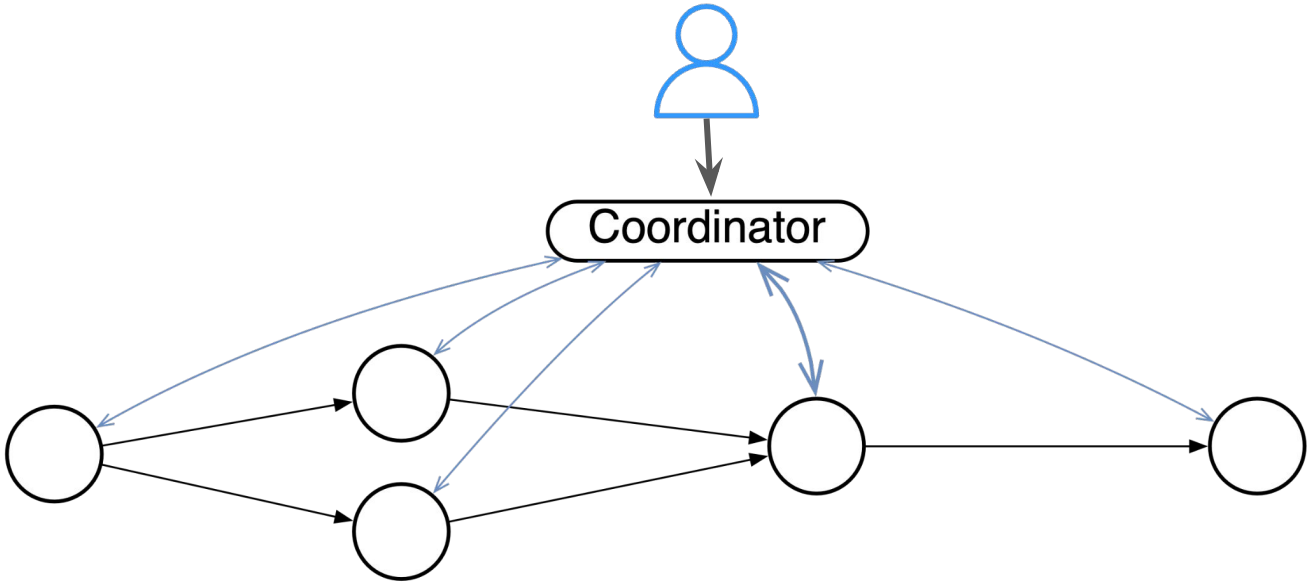
Requirements of Pause:

- Fast
- Interact with an operator after pause



Approach 1: Using OS Signals

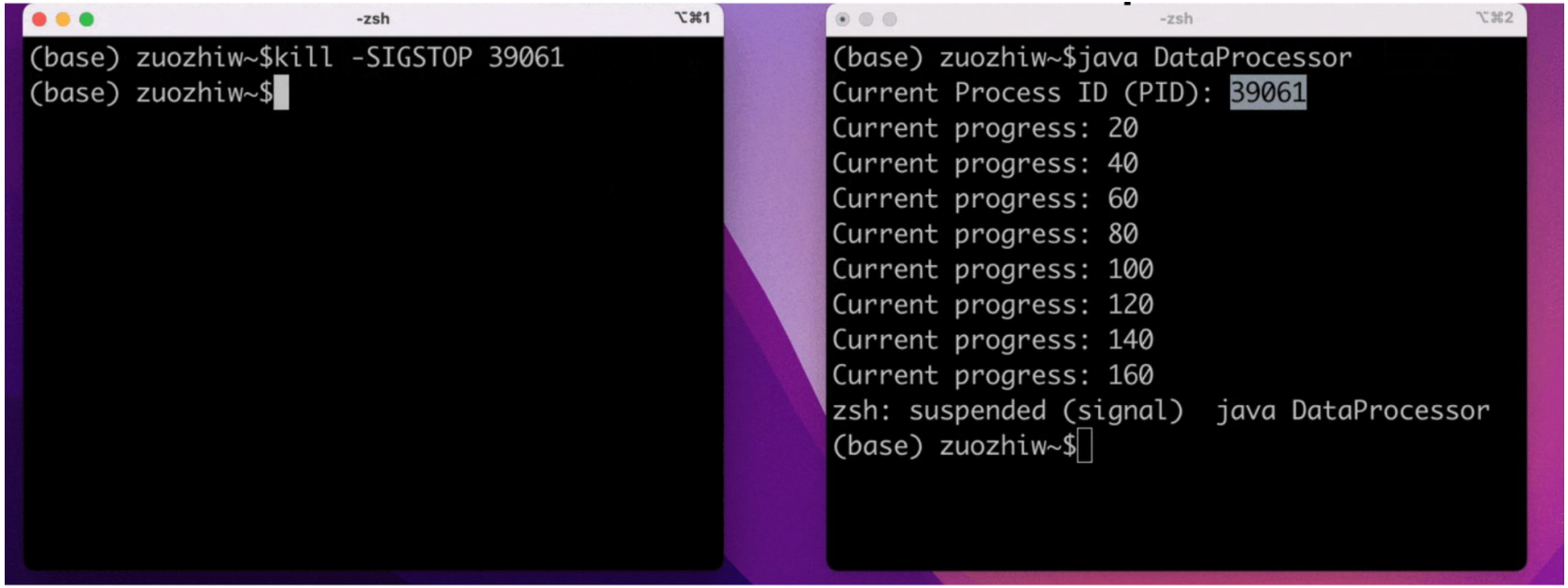
SIGSTOP (signal stop) / **SIGCONT** (signal continue)



Using OS Signals to Pause

User

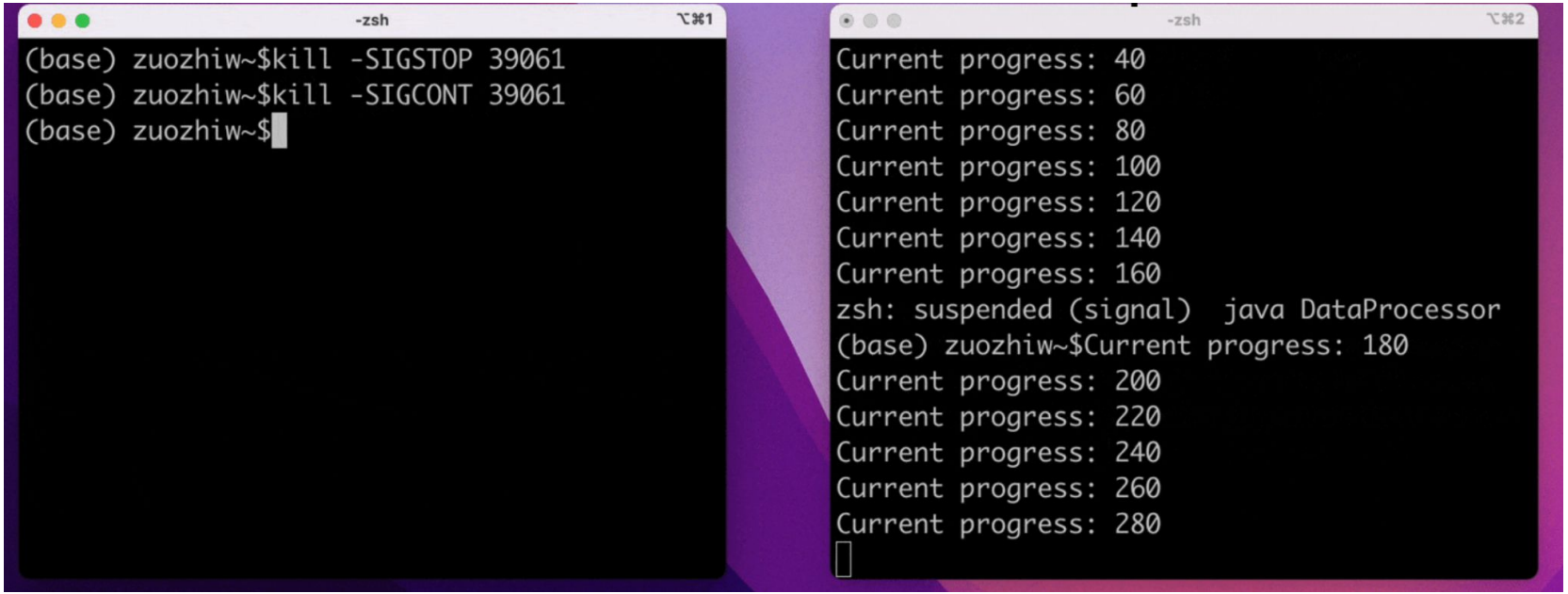
Operator



Using OS Signals to Resume

User

Operator



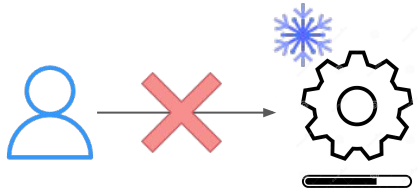
Approach 1: Using OS Signals

Pros: 

- Suspends immediately
- OS native support - little implementation effort

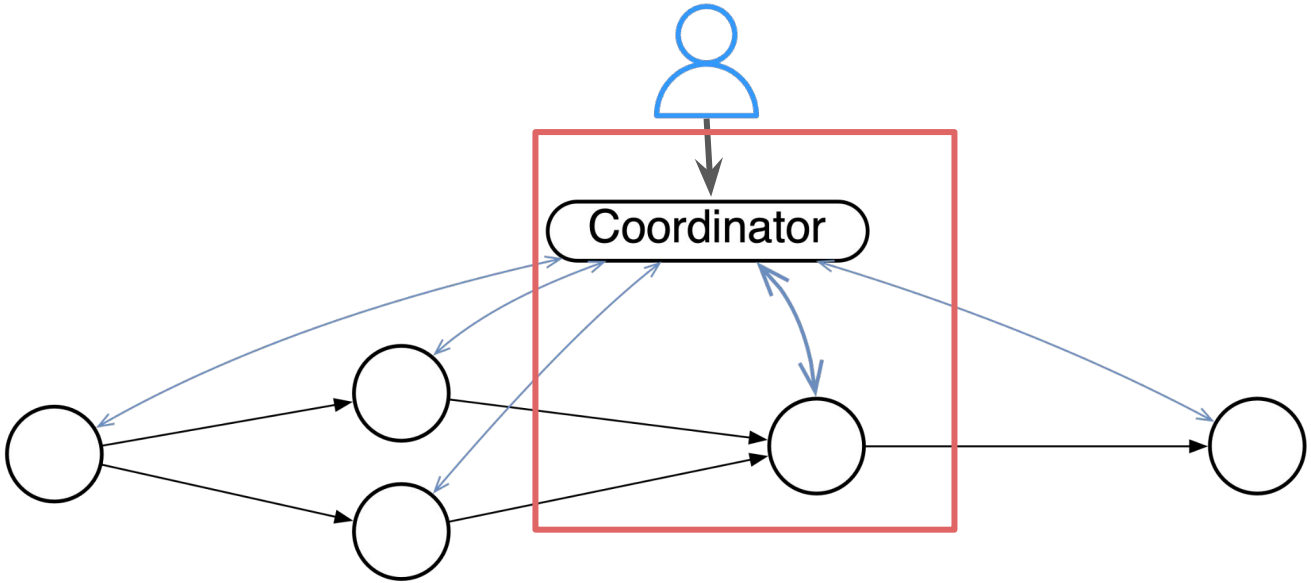
Cons: 

- Unable to interact with an operator

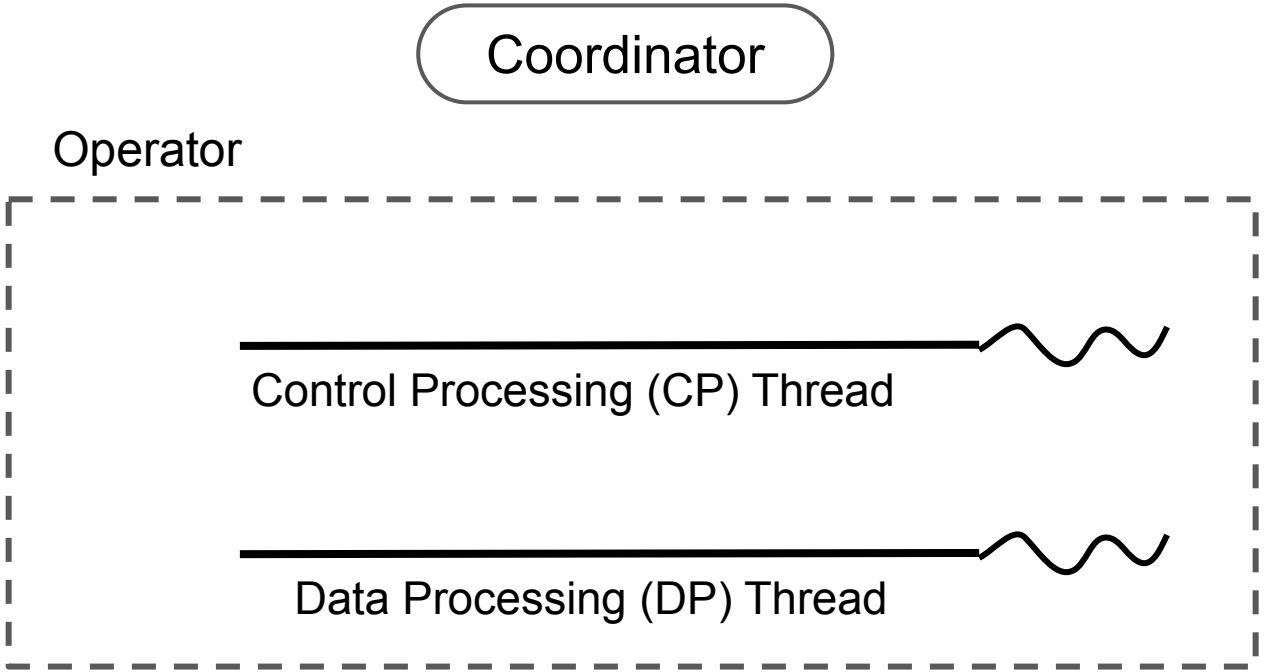


Approach 2: Using Thread-level Signal

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

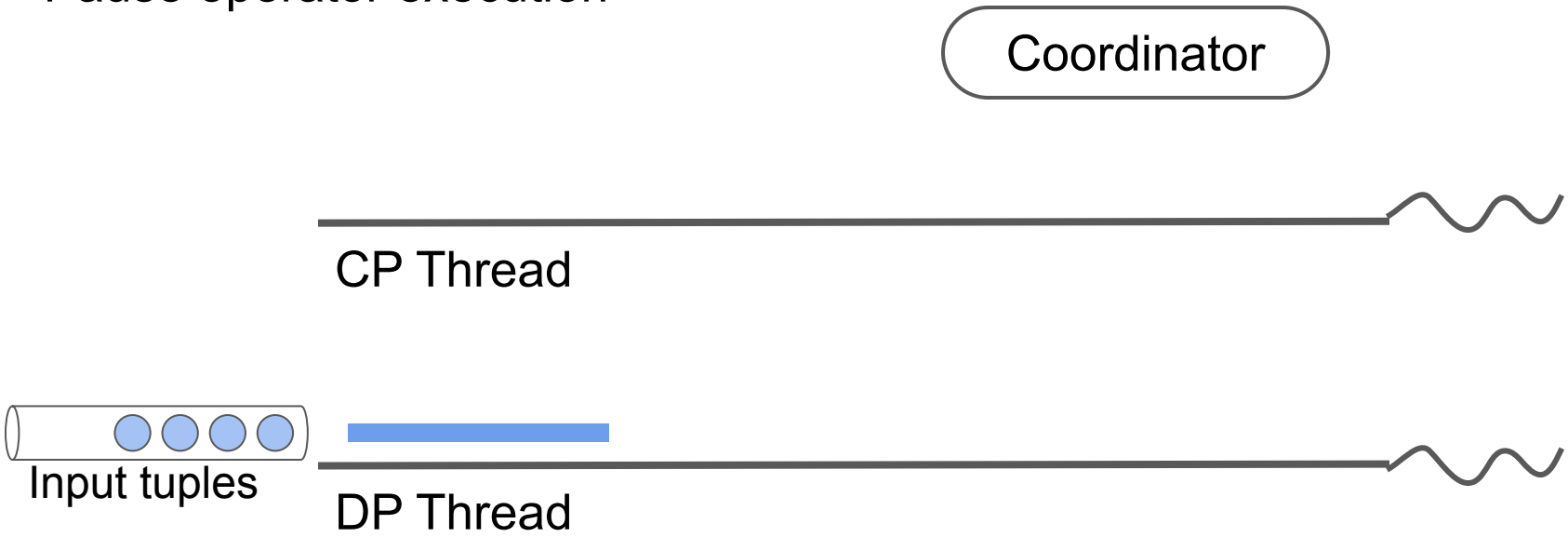


Approach 2: Using Thread-level Signal



Approach 2: Using Thread-level Signal

Pause operator execution

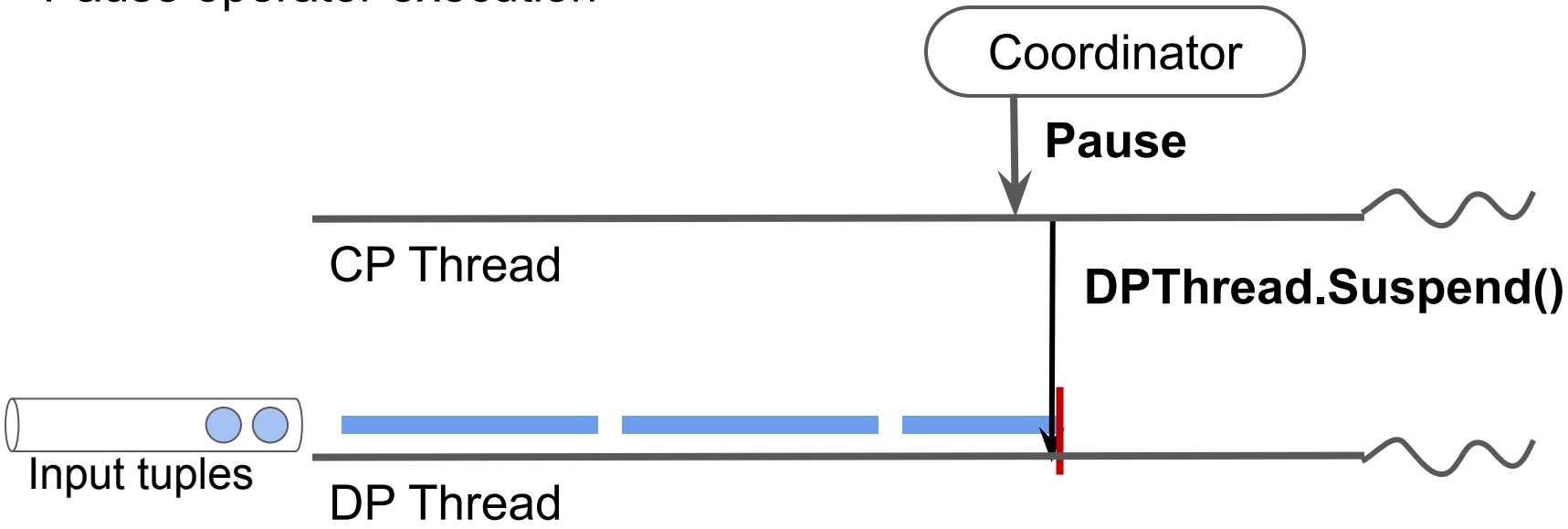


Approach 2: Using Thread-level Signal

Pause operator execution

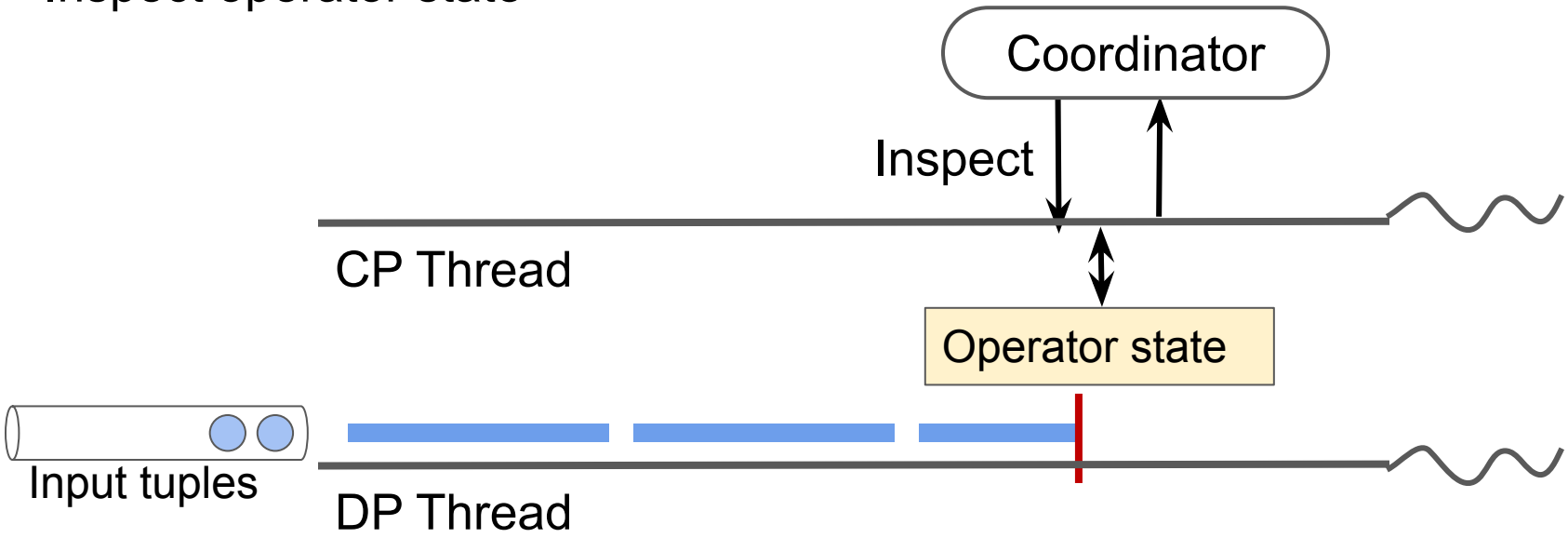
Approach 2: Using Thread-level Signal

Pause operator execution



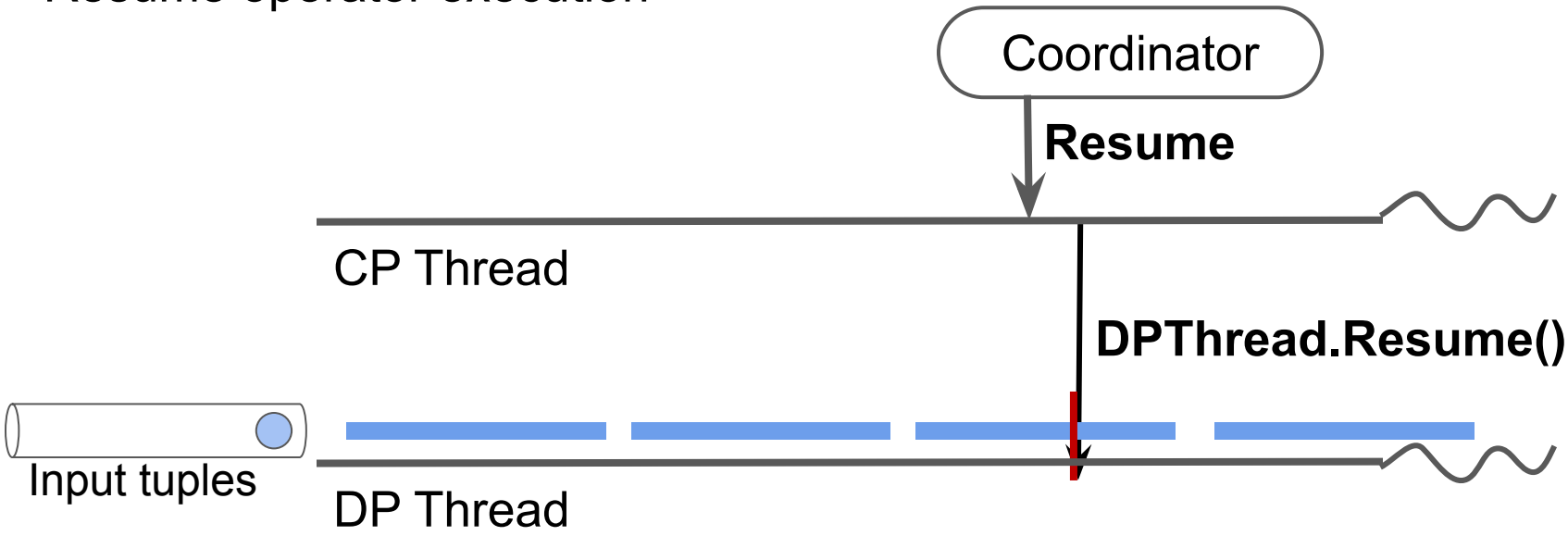
Approach 2: Using Thread-level Signal

Inspect operator state



Approach 2: Using Thread-level Signal

Resume operator execution



Approach 2: Using Thread-level Signal

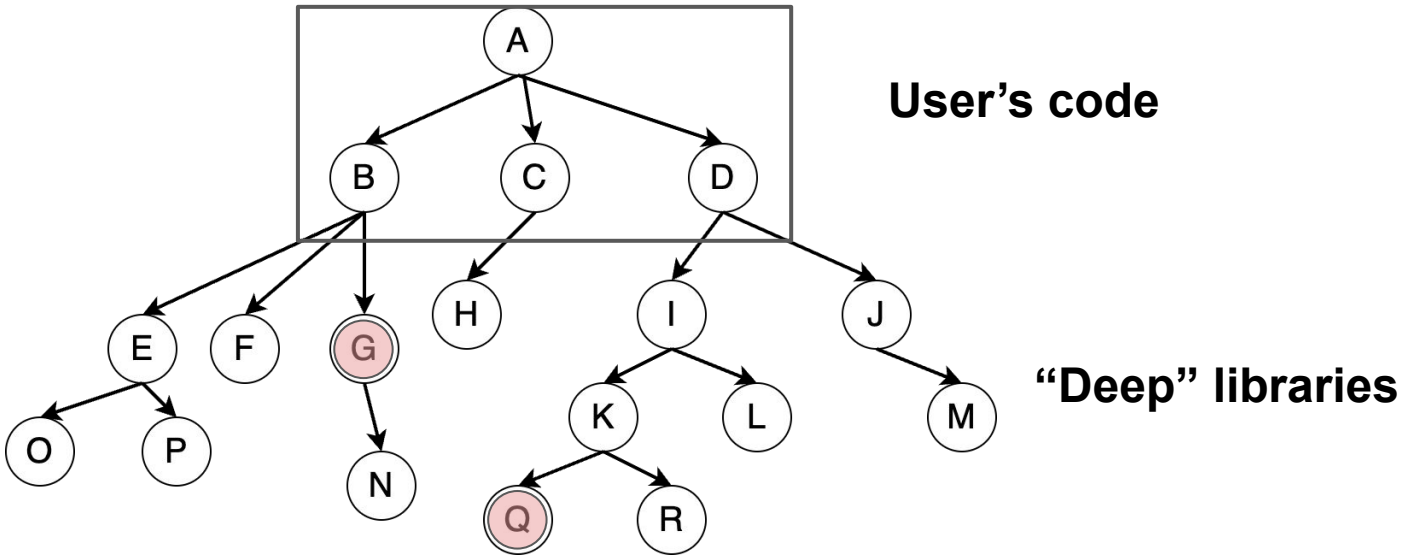
Pros: 

- Suspends immediately
- Able to interact with operator after pause

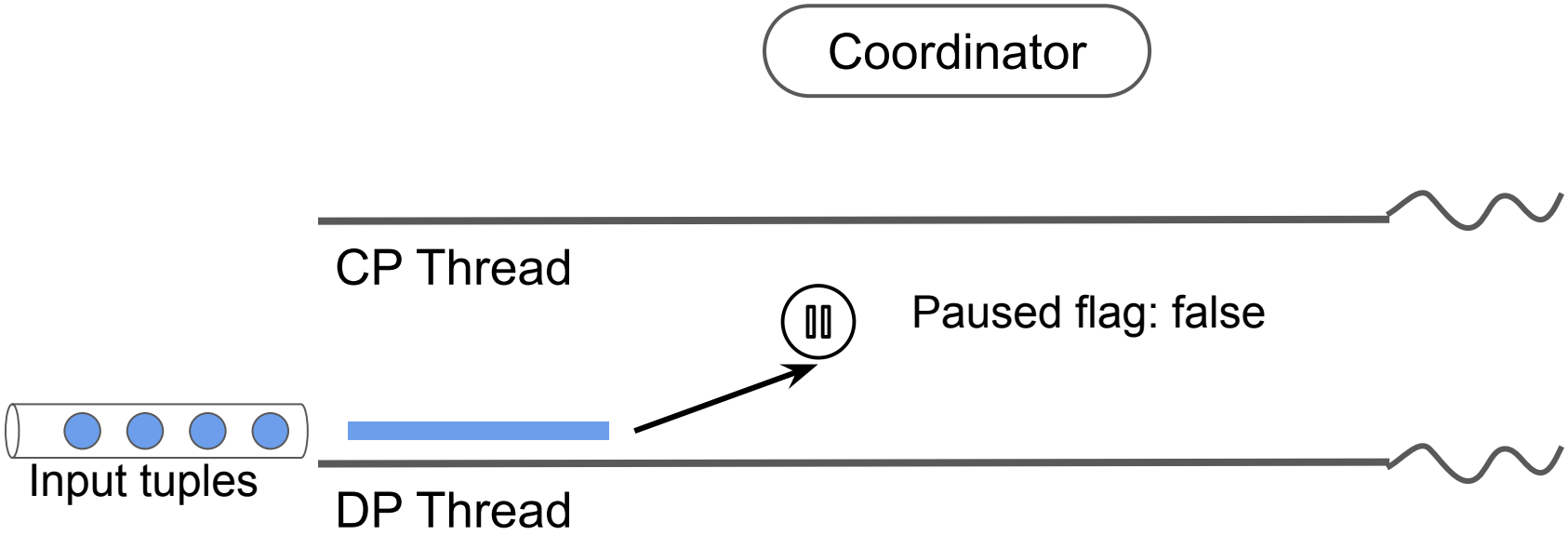
Problem with Approach 2

Stopping point is arbitrary and incomprehensible to users.

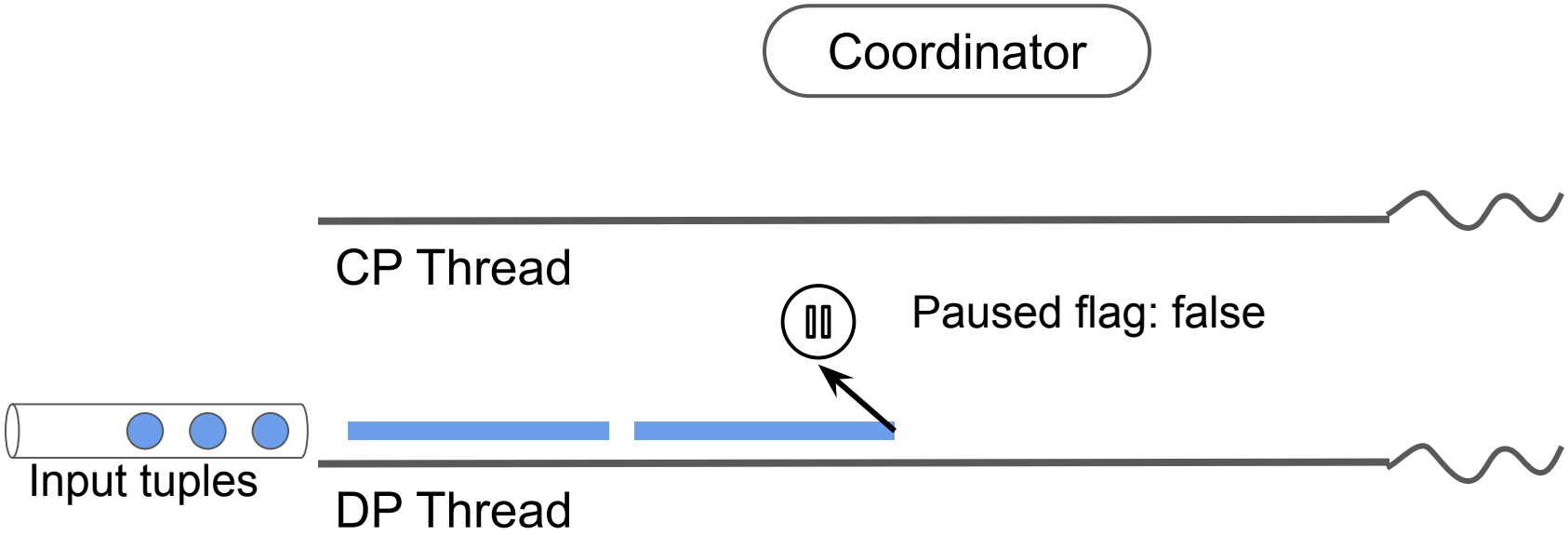
Operator's call graph



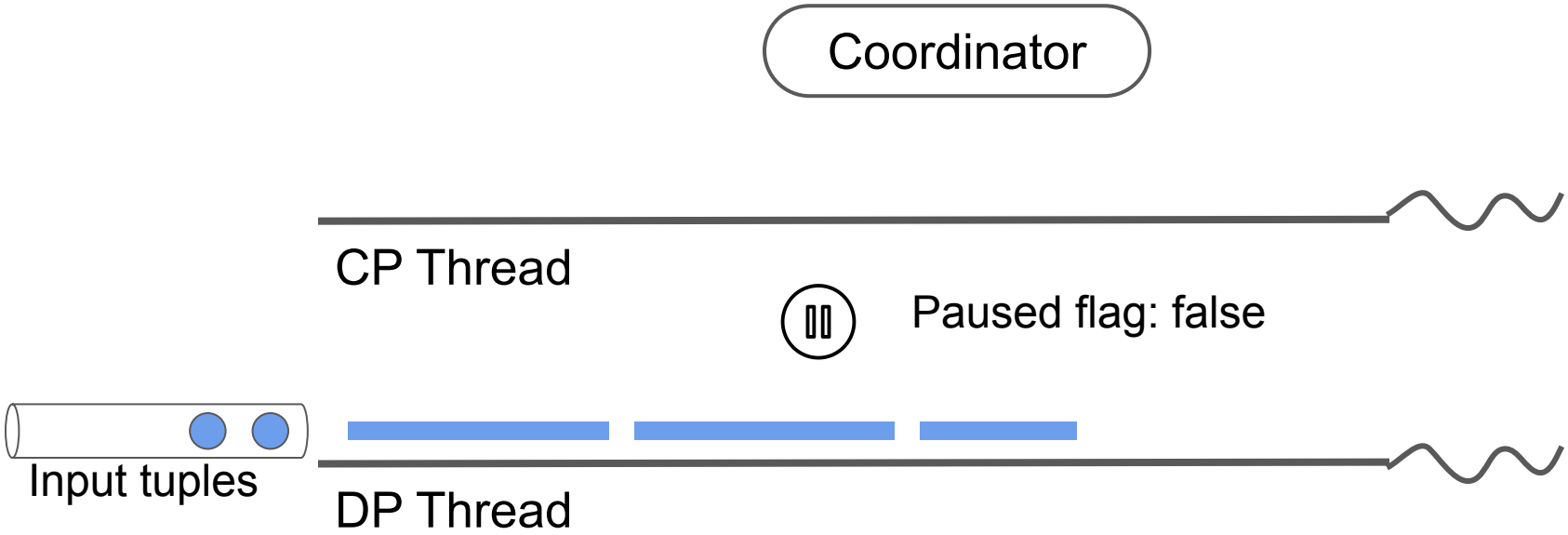
Approach 3: Checking for Pause between Tuples



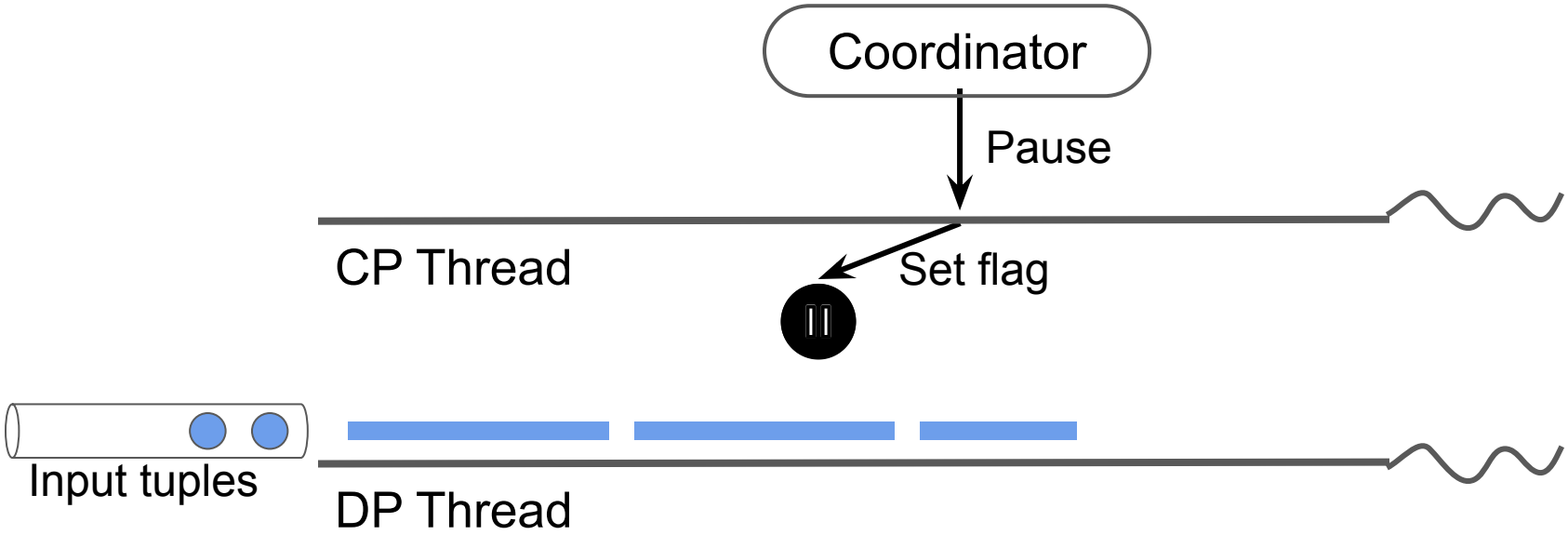
Approach 3: Checking for Pause between Tuples



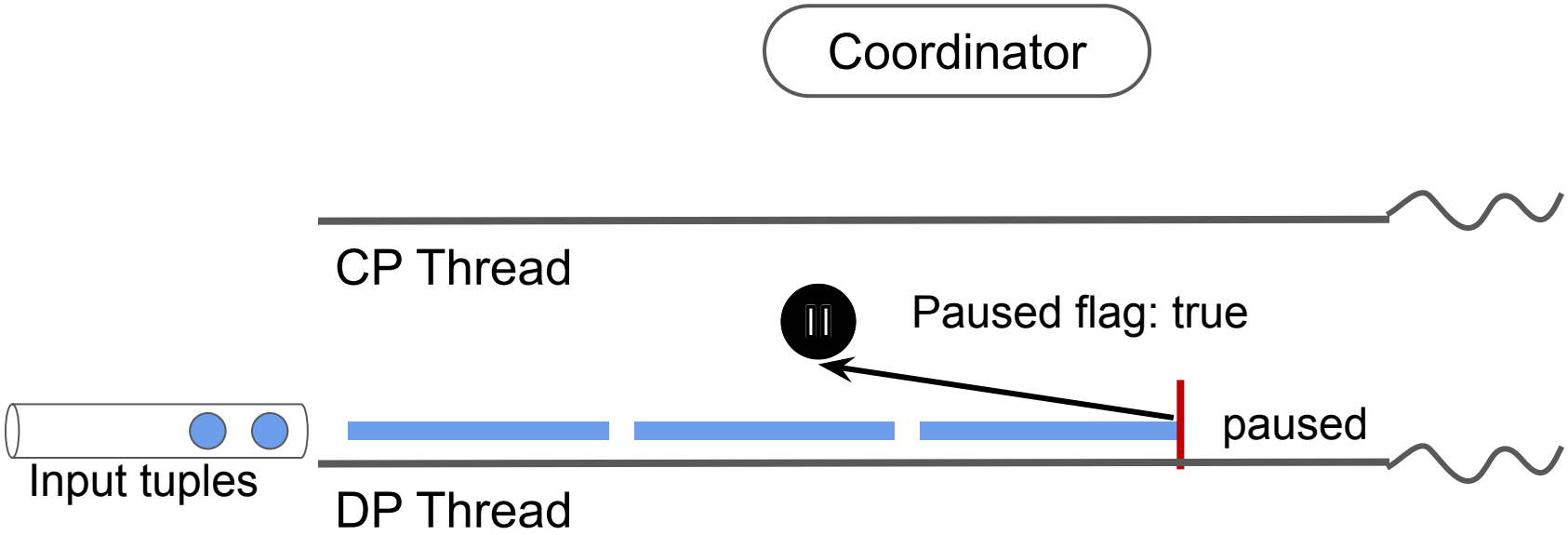
Approach 3: Checking for Pause between Tuples



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%)

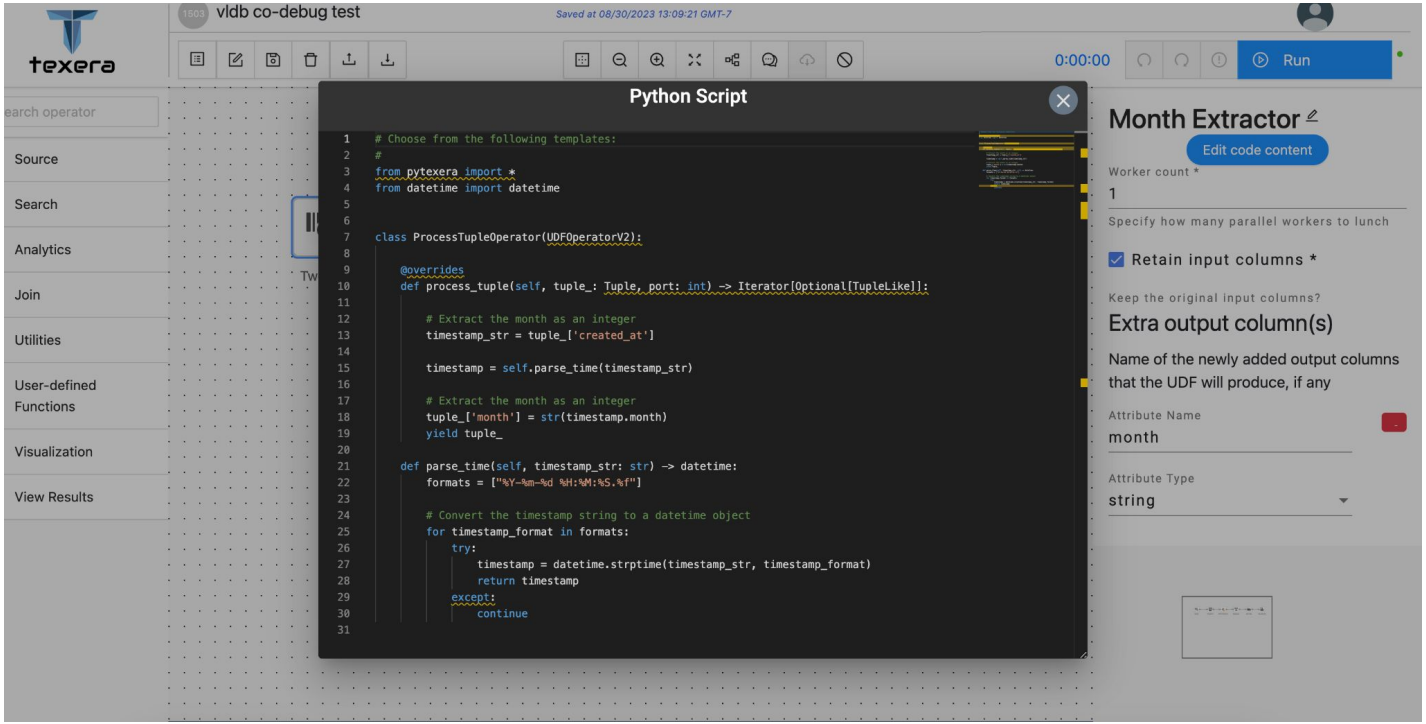
Comparison of three approaches

	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



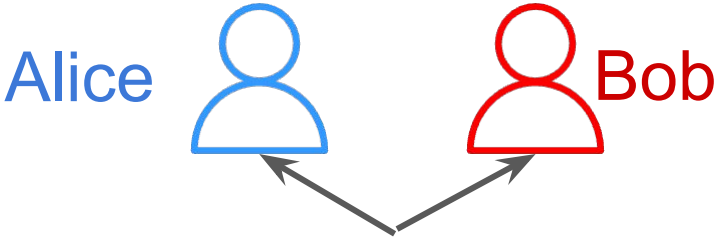
Runtime Co-debugging on Python UDFs

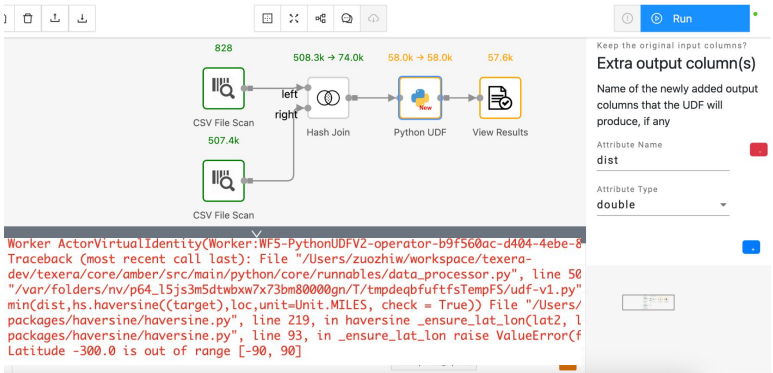
Data errors
- misformated data
- unexpected nulls
- corrupted data
- ...

UDF errors
- corner cases
- wrong parameters
- ...

Co-debugging Demo

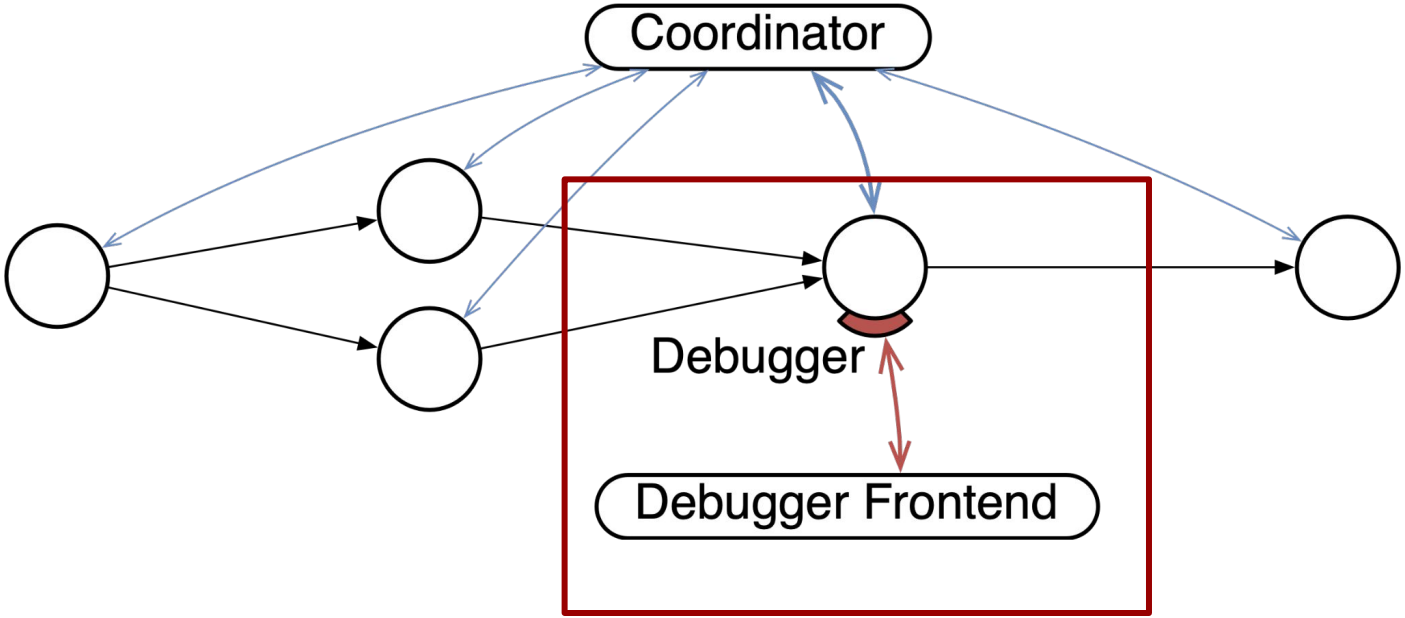
Domain experts need technical assistance in debugging.



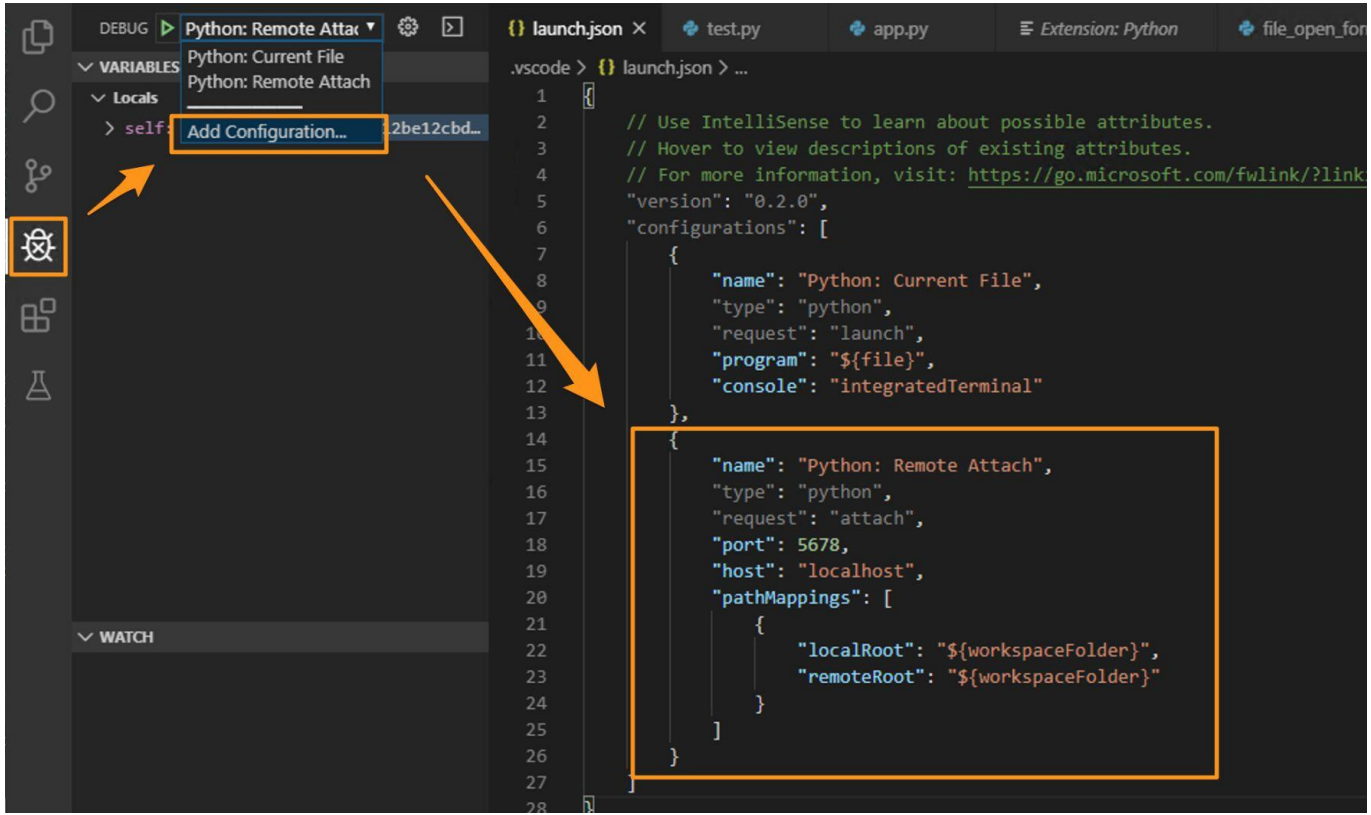


Runtime Error

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: 

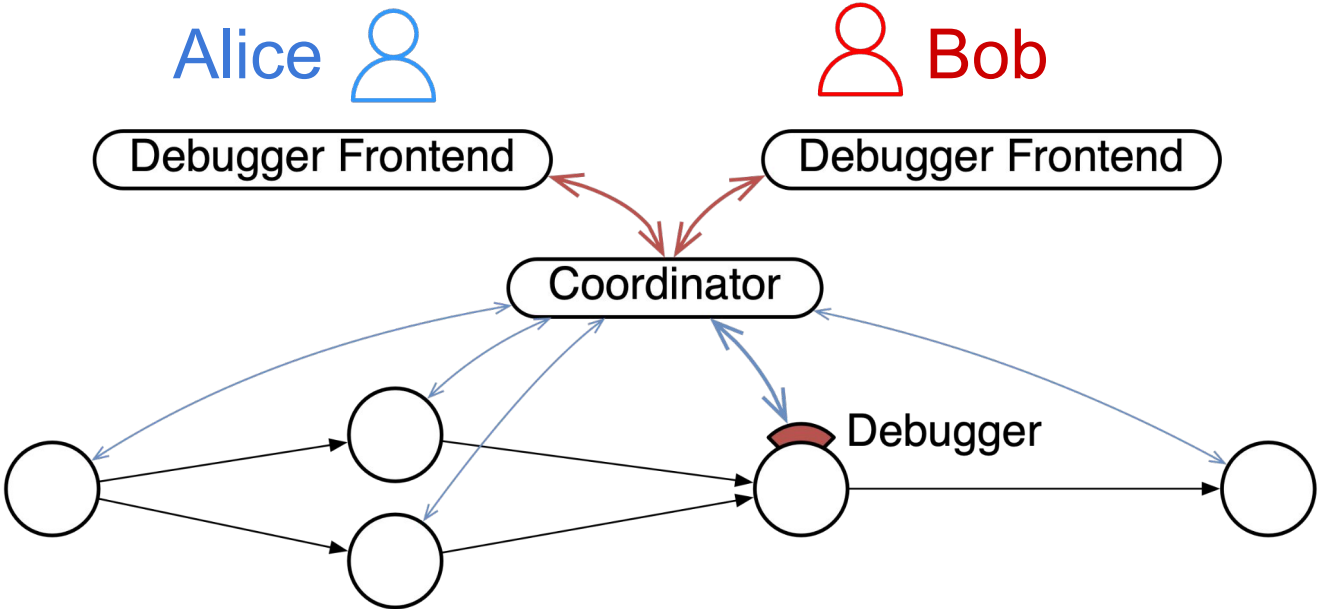
- Works out-of-the-box. (Adopted by PyFlink, PySpark)

Con: 

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

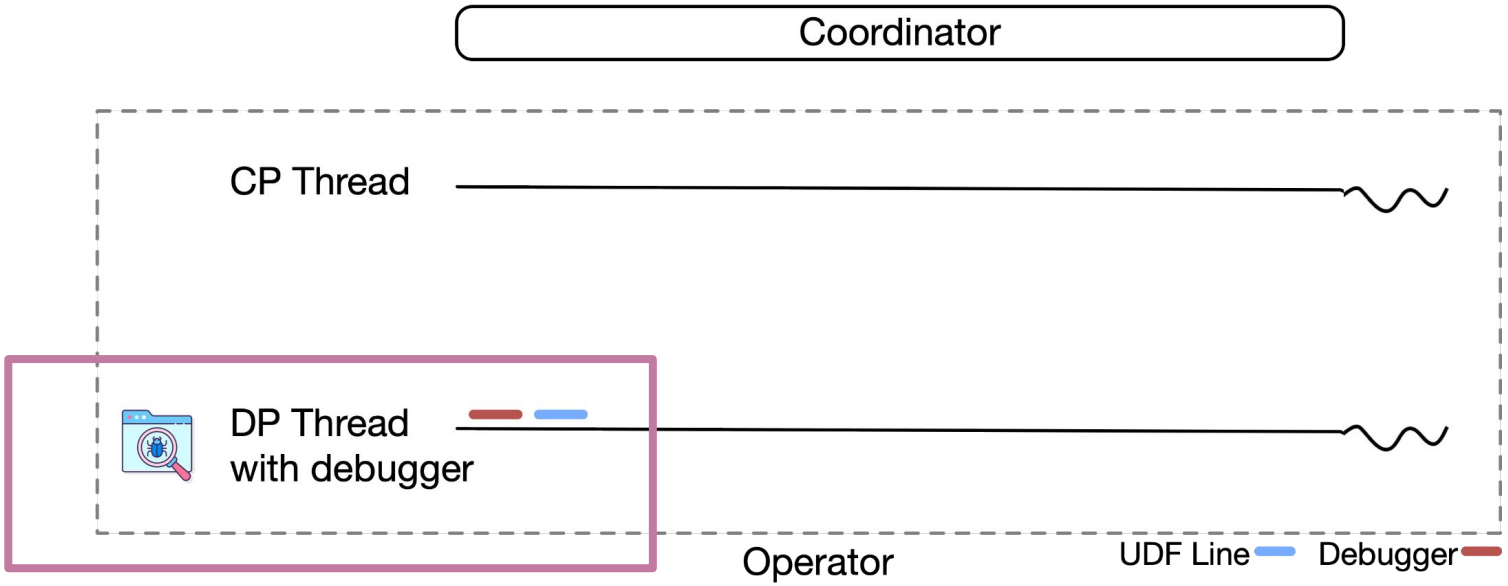
Approach 2: Coordinator Controls Debuggers

Co-debugging: coordinator manages multiple debug frontends



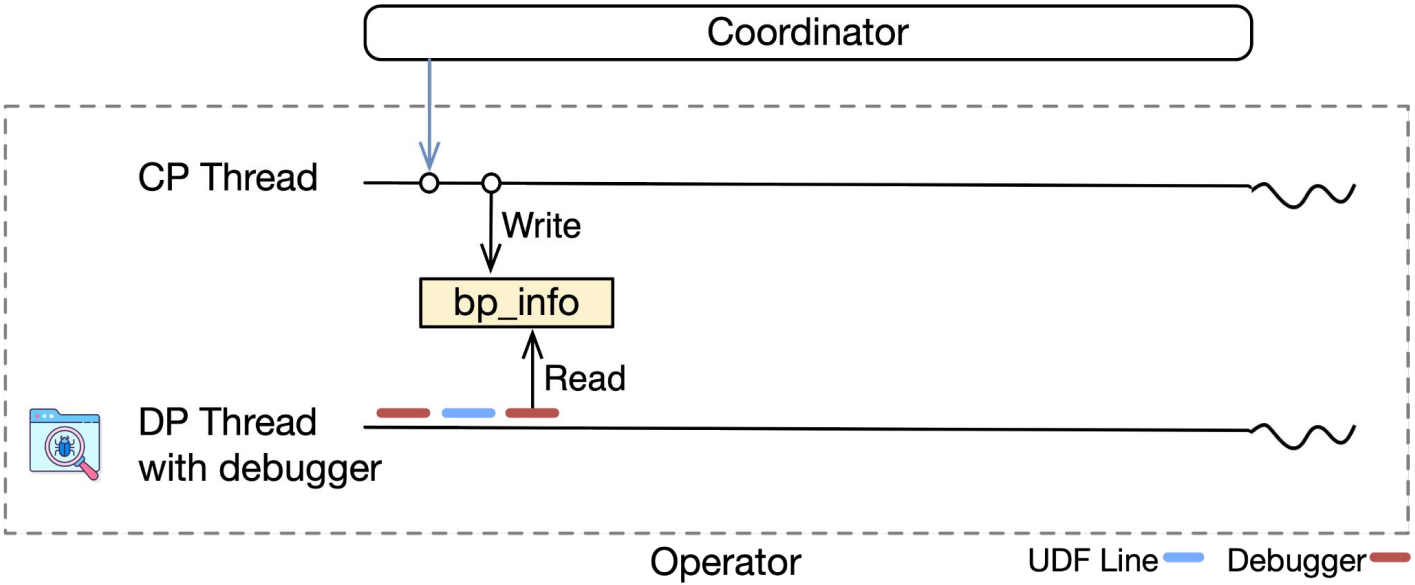
Approach 2: Coordinator Controls Debuggers

Execution model



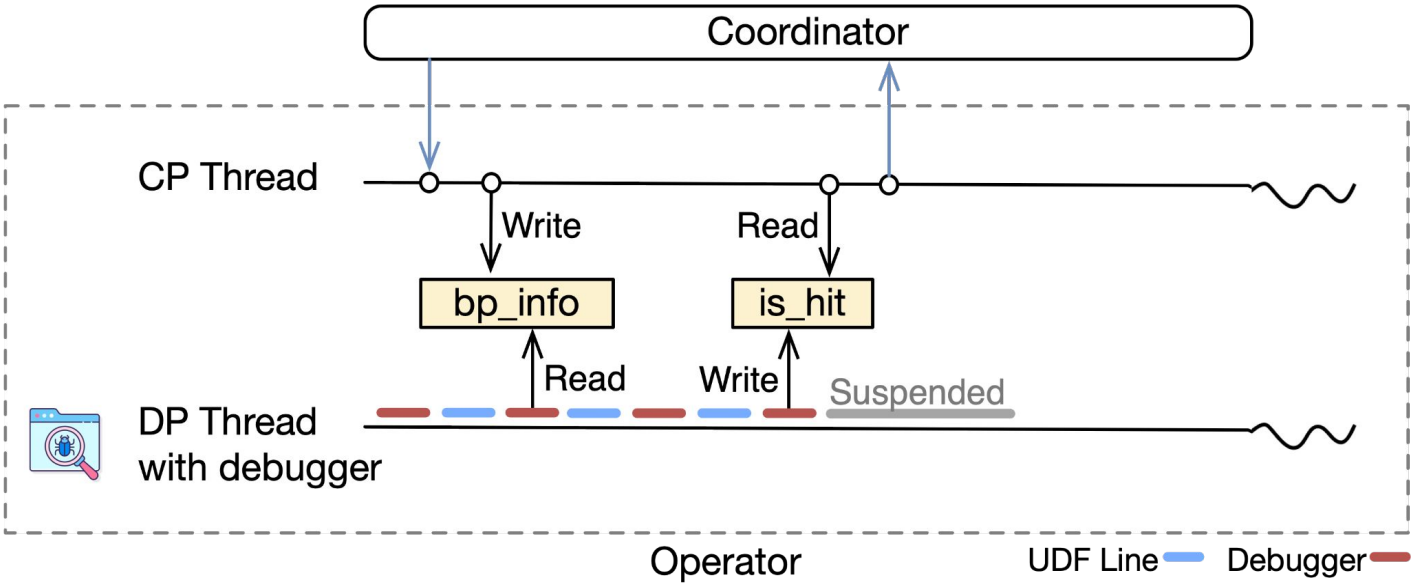
Approach 2: Coordinator Controls Debuggers

Setting a breakpoint



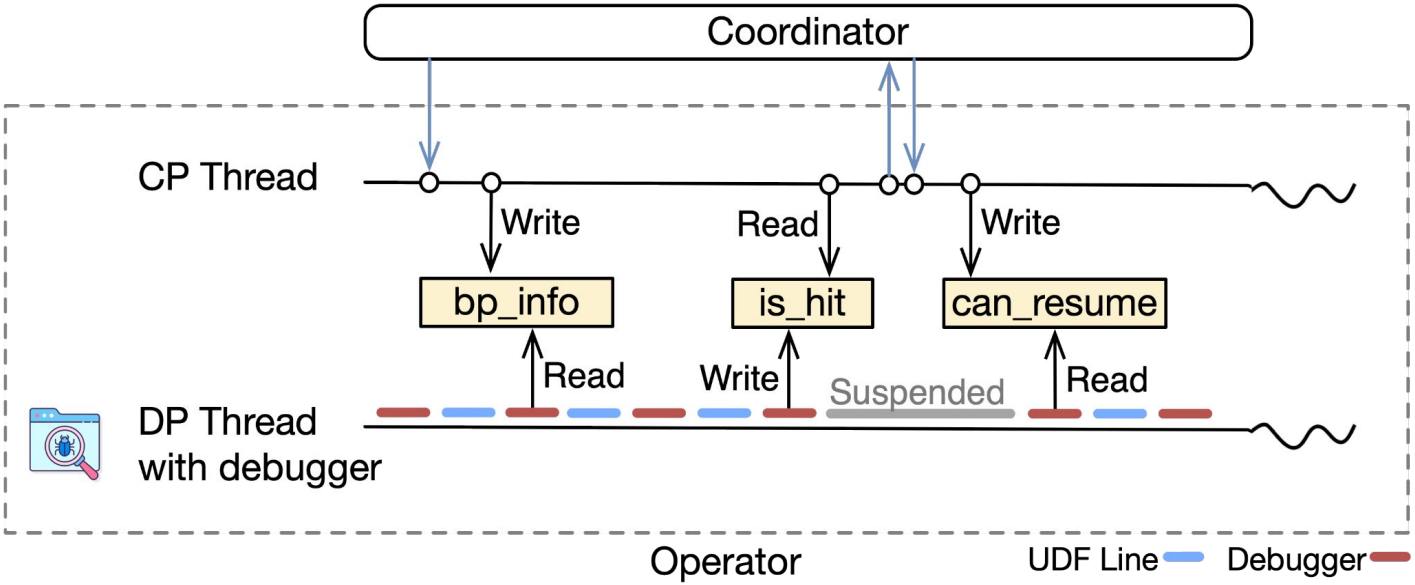
Approach 2: Coordinator Controls Debuggers

Hitting a breakpoint



Approach 2: Coordinator Controls Debuggers

Resuming from the breakpoint



Approach 2: Coordinator Controls Debuggers

Pro: 

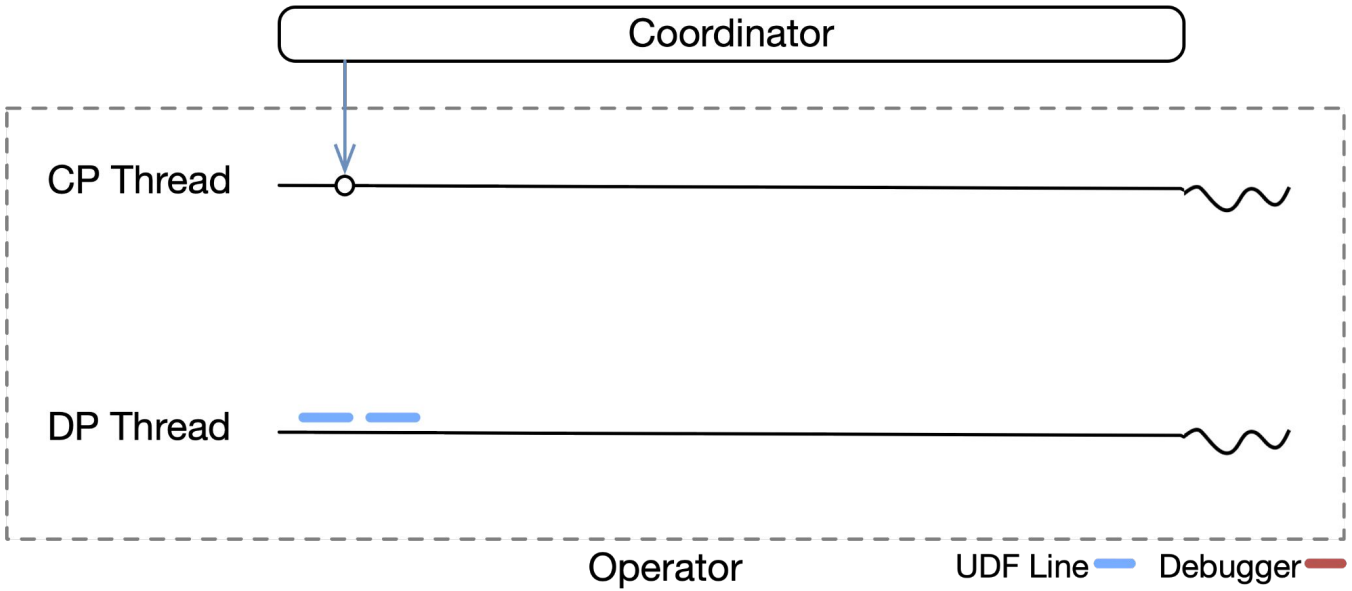
- Supports co-debug

Con: 

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

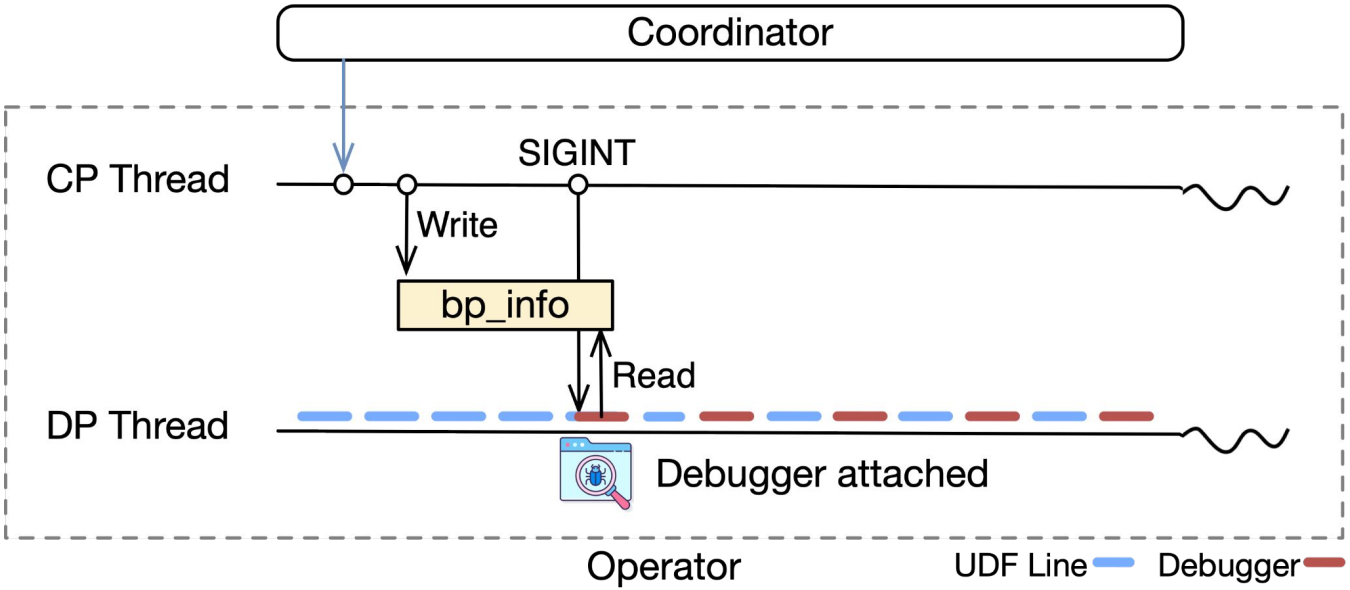
Reducing Overhead: Invoke Debugger on Demand

Starting without a debugger



Reducing Overhead: Invoke Debugger on Demand

Attaching a debugger



Adoption of Texera

Deployments	Users	Projects	Workflows	Executions	Versions	Contributors
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.

Speakers:

Dr. Chen Li

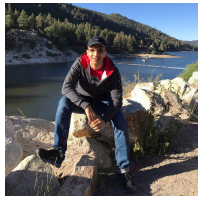
Dr. Zuozhi Wang

Core Texera Team:

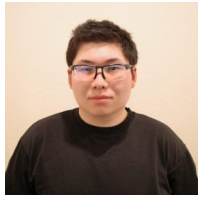
Dr. Sadeem Alsudais

Yunyan Ding

Yicong Huang



Dr. Avinash Kumar



Xinyuan Lin

Xiaozhen Liu

Raj Mohanty

Shengquan Ni

Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang and Chen Li

UCIRVINE

Texera GitHub Repo

texera