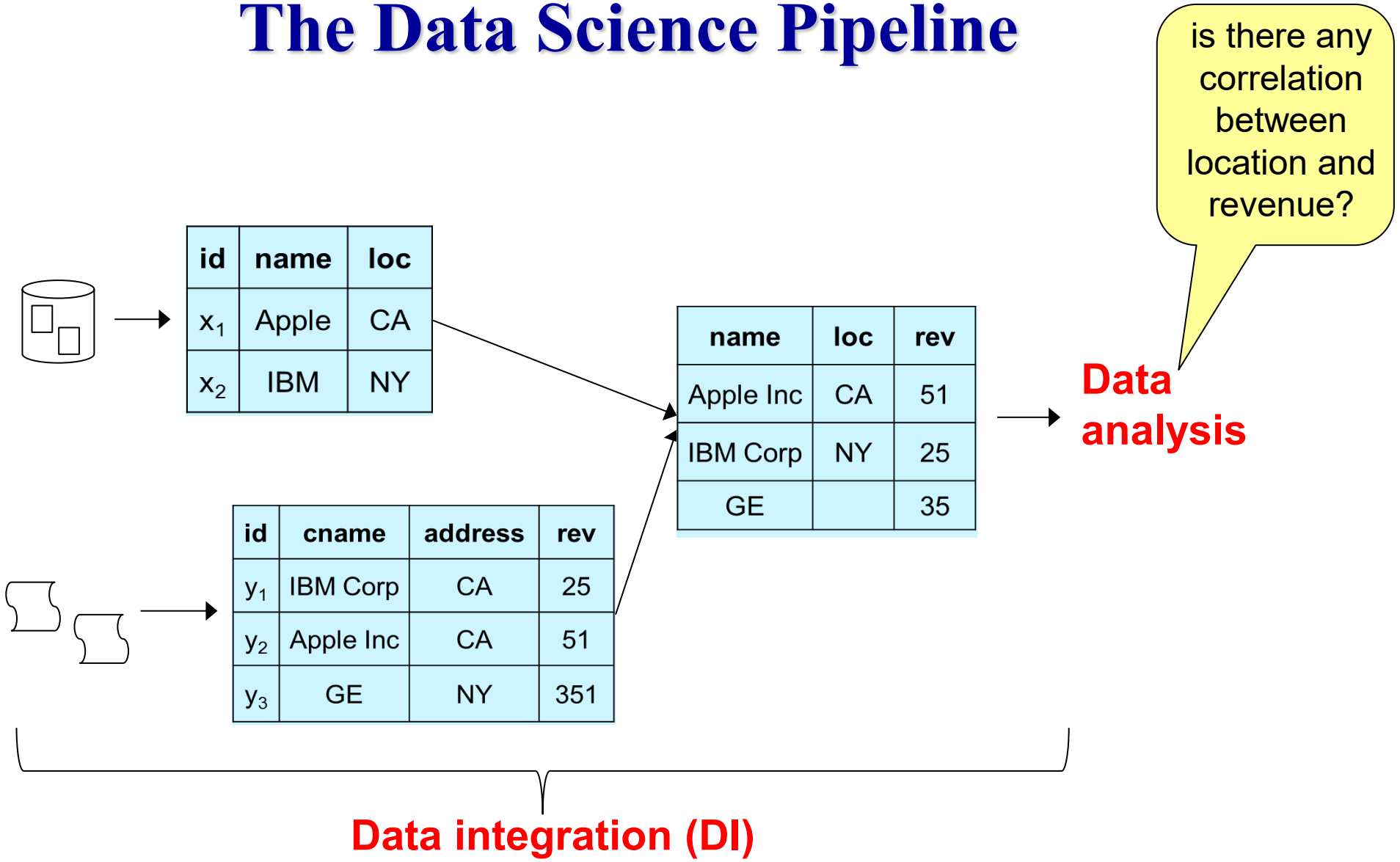
Cymphony: Toward a Crowdsourcing Platform for Data Science

Amanpreet Singh Saini

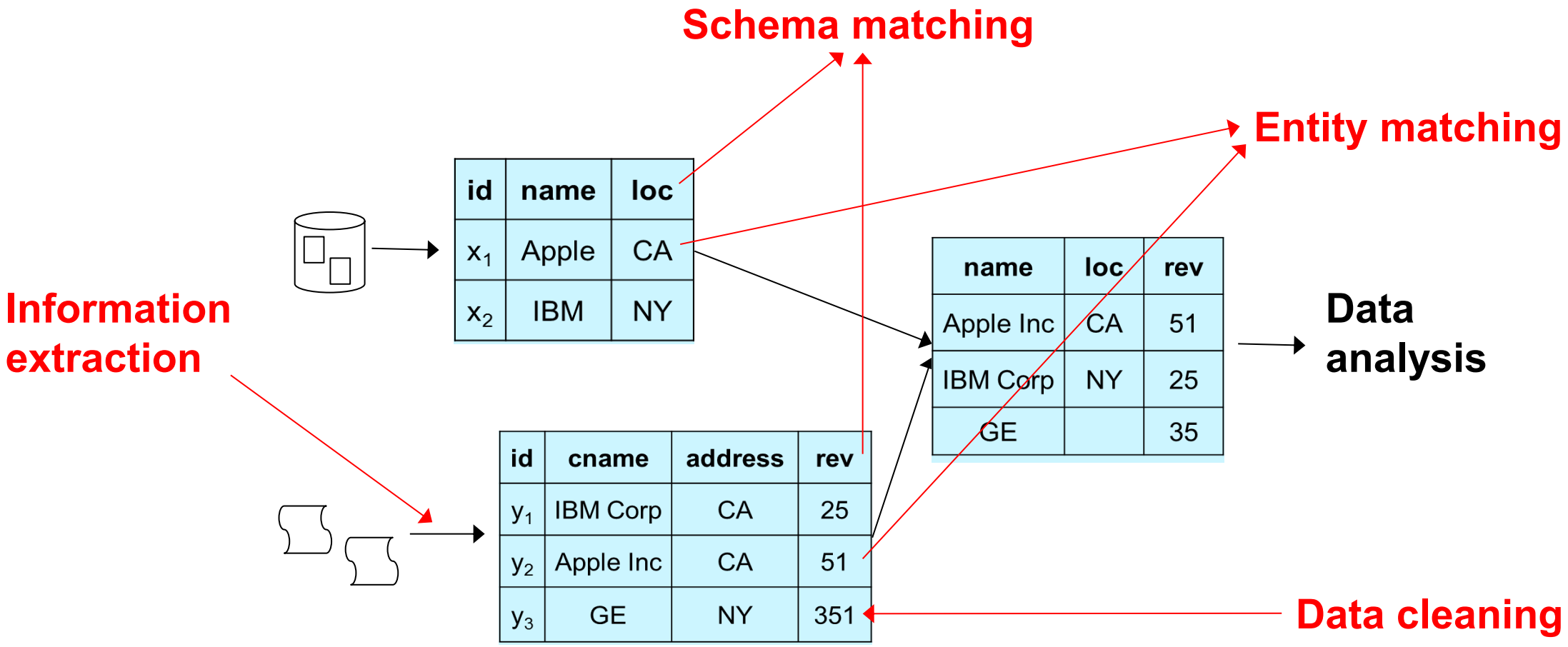
PhD Defense

March 6, 2026

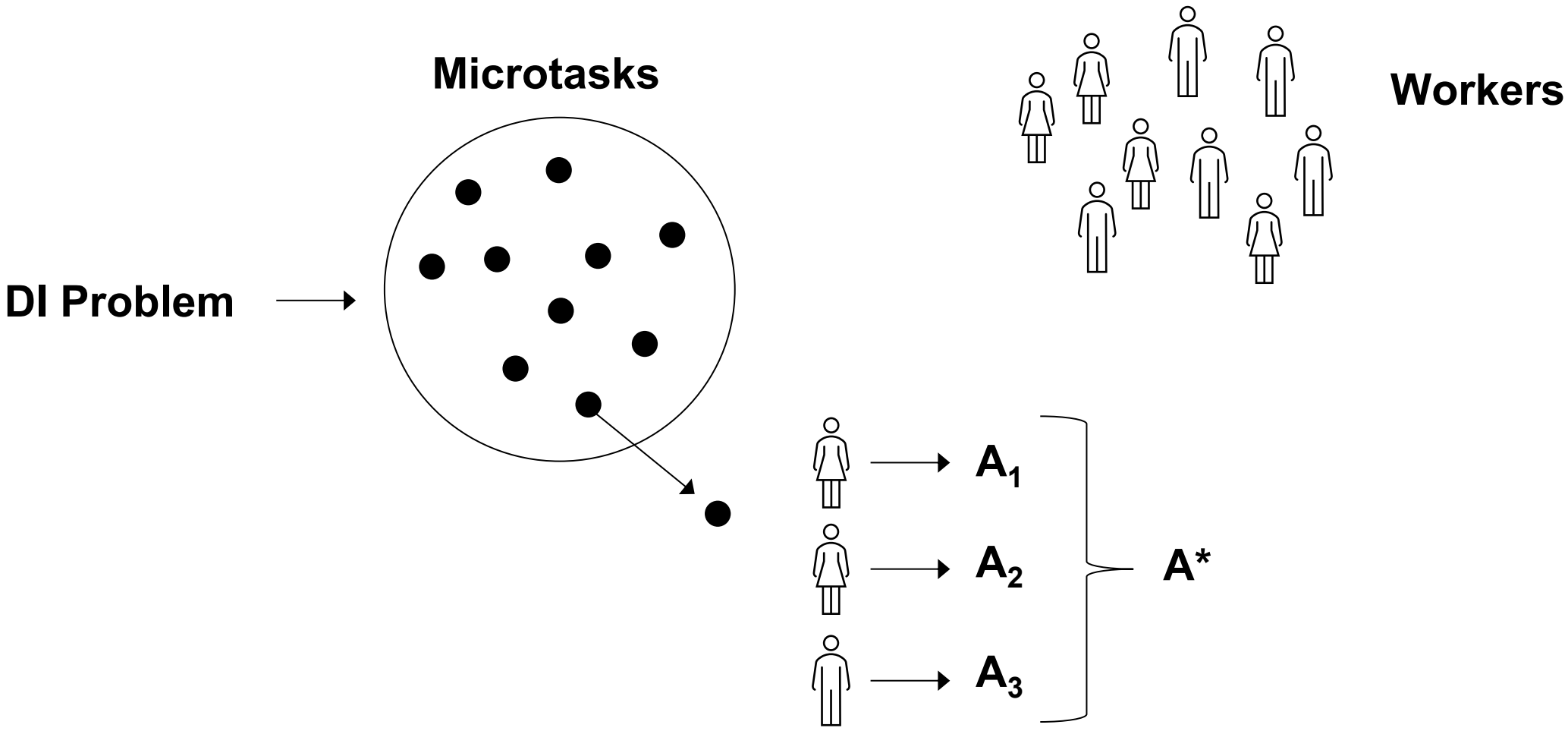
The Data Science Pipeline



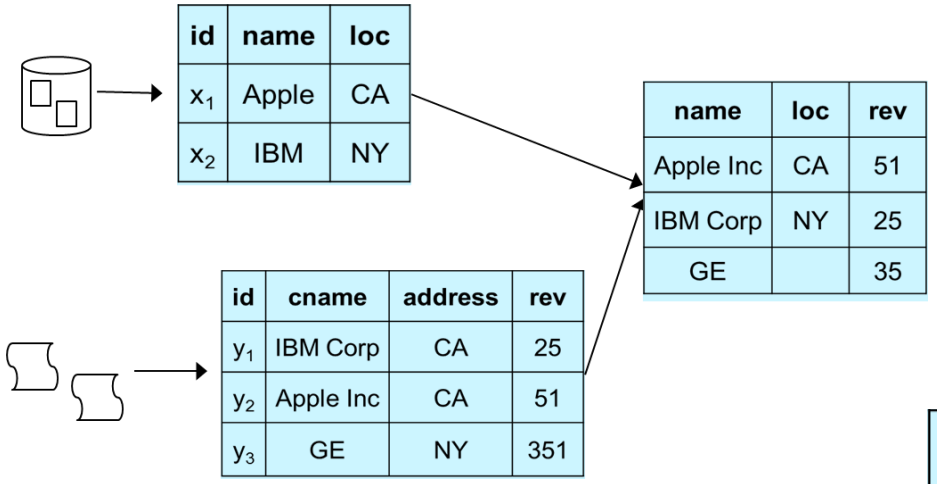
Example Data Integration Problems



These Problems Are Often Solved Using Crowdsourcing



Example 1: Data Cleaning



id	name	loc
x_1	Apple	CA
x_2	IBM	NY

id	cname	address	rev
y_1	IBM Corp	CA	25
y_2	Apple Inc	CA	51
y_3	GE	NY	351

name	loc	rev
Apple Inc	CA	51
IBM Corp	NY	25
GE		35

id	cname	address	rev
y_1	IBM Corp	CA	25
y_2	Apple Inc	CA	51
y_3	GE	NY	351

Example 2: Creating Labeled Data for Entity Matching

id	name	loc
x_1	Apple	CA
x_2	IBM	NY

id	cname	address	rev
y_1	IBM Corp	CA	25
y_2	Apple Inc	CA	51
y_3	GE	NY	351

Do these two tuples refer to the same entity?

x_1	Apple	CA	
y_2	Apple Inc	CA	51

Yes | No | Not Sure

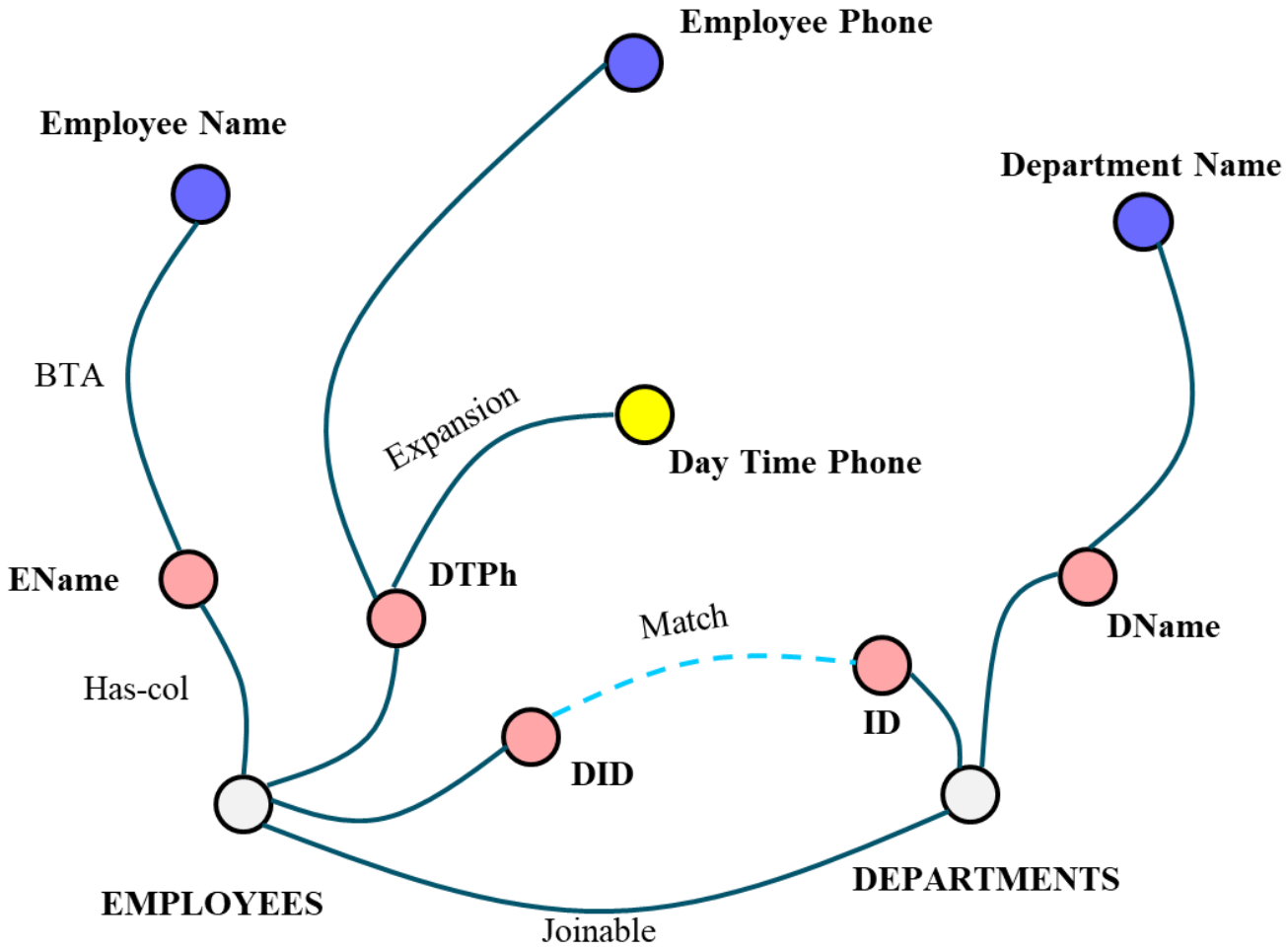
Example 3: Crowdsourcing for Data Catalog

EMPLOYEES

EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

DEPARTMENTS

ID	DName
d_1	Sales
d_2	Legal



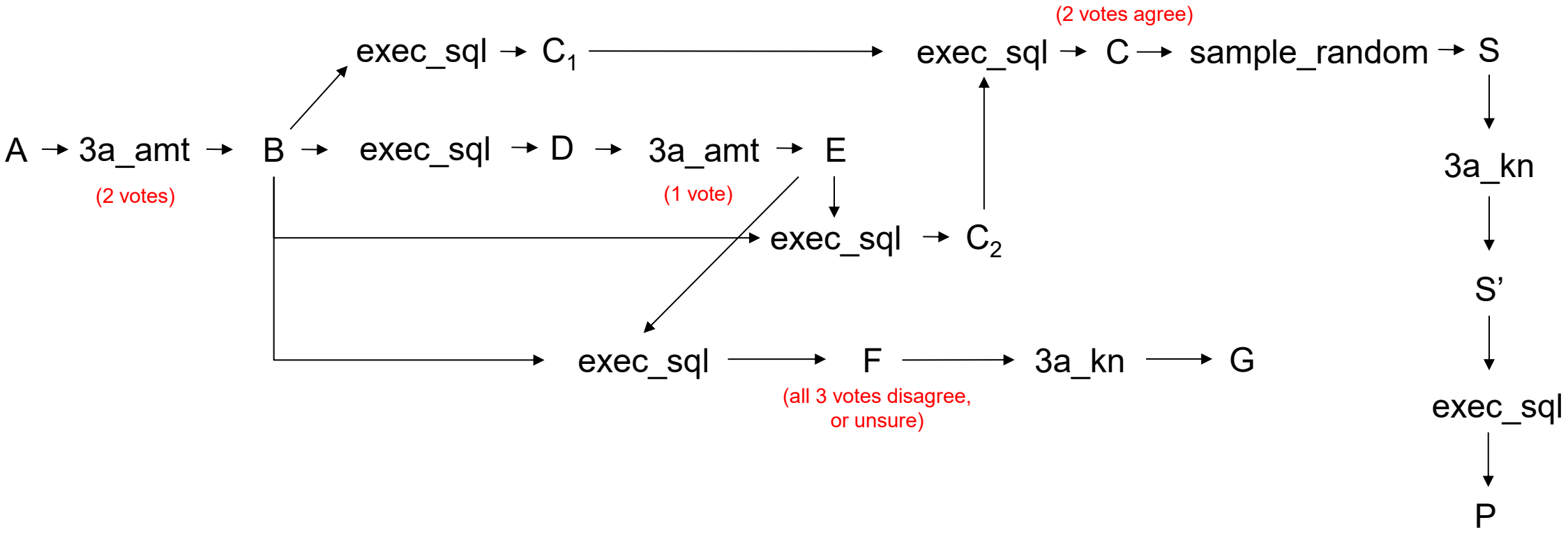
State of the Art

- **Academia**
  - Pioneering work in our group in 2003-2010
  - Super hot in the DB community in 2011-2020
  - Lot of papers, mostly solving “point problems”
    - E.g., develop algorithmic solutions to solve entity matching, low CS cost, high accuracy
  - **No system development, no open-source software**
- **Industry**
  - **Solutions baked into bigger systems**, e.g., CS solutions inside a data catalog system
  - Cloud-hosted self-service solutions
    - Amazon Mechanical Turk, Appen, Clickworker, Microworkers
  - Contractors
    - Sama, iMerit, Appen, Toloka

Very Hard for Data Science Teams to Crowdsource

- **Can't use industrial solutions**
  - Hosted CS services: limited capabilities, hard to customize/extend, can't send sensitive data
  - Contractors: very expensive, take too long
    - These are best suited for big companies, e.g., Google, Microsoft, Walmart
- **Can't use academic solutions**
  - No general-purpose software
  - Nothing that
    - can be deployed quickly and inexpensively
    - easy to customize/extend
    - Saves time by already taking care of mundane stuff
- **So today teams do CS in ad-hoc cumbersome ways**
  - Most common method: emailing around Excel spreadsheets, writing custom code
  - Very time consuming, error prone

Especially Because CS Workflows Are Often Quite Complex



Output = (C \ S) union S' union G

My Dissertation

- **Introducing the problem of building a CS platform for DI** 
- **Designing Cymphony**
  - Define problem, operators, workflows
- **Implementation**
- **Applying Cymphony to DI problems & evaluation**
  - Usability, accuracy, scaling
- **Customizing Cymphony for Data Catalog Systems**
  - User registration, operators, program, API
- **Releasing Cymphony**
  - Stand-alone, as a system component

Cymphony: A Crowdsourcing Platform for Data Integration, in preparation
Toward data cleaning with a target accuracy: a case study for value normalization, Ardalan et al., IEEE Big Data 2022
Entity matching meets data science: a progress report from the Magellan project, Govind et al., SIGMOD 2019

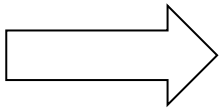
Building a CS Platform for DI

- **Requirements**
  - End-to-end system
  - Can solve many DI problems
  - Easy to use, customize, and extend
- **Envisioned usage**
  - Stand-alone
  - As component of a bigger system

Challenge: Define CS Problems for DI

- **Wide variety of DI problems**
  - Extraction, cleaning, classification, matching, etc.
  - Very different in nature
- **How to define a simple CS problem template that can solve most of them?**
- **Solution**

Product Title	Price
"ASUS X205TA 11.6 Inch Laptop (Intel Atom, 2 GB, 32GB SSD, Gold) - Free Upgrade to Windows 10"	$199.00
"Lenovo G50 Entertainment Laptop - Black: DOORBUSTER - Intel Core i7-5500U, 8GB RAM, 1TB HDD, 15.6\ FHD 1080P Display, USB3.0, HDMI,Bluetooth, Windows 8.1"	$799.77
...	...



Product Title	Price	Final Label
"ASUS X205TA 11.6 Inch Laptop (Intel Atom, 2 GB, 32GB SSD, Gold) - Free Upgrade to Windows 10"	$199.00	32GB
"Lenovo G50 Entertainment Laptop - Black: DOORBUSTER - Intel Core i7-5500U, 8GB RAM, 1TB HDD, 15.6\ FHD 1080P Display, USB3.0, HDMI,Bluetooth, Windows 8.1"	$799.77	1TB
...	...	...

Challenge: Define CS Operators & Workflows

- **This was difficult, taking a fair amount of time**
- **Consider a simple CS problem**
  - Given a table T, output T with a new column final_annotation
  - For each tuple x in T, get 3 annotations from 3 workers, then take majority vote
- **How to create a workflow of operators to model this problem?**

- **Initial idea**
  - Use three operators: assign, annotate, aggregate
  - Use three tables A(x,w), B(x,w,a), C(x,a*)
  - Assign(T,w,A) => x
  - Annotate(x,w) => a
  - Aggregate(all tuples <x,a> in C) => a*
  - This is intuitive, but can't define a clean workflow involving these three operators

Complex Interactions Among Three Operators

- **Input: table T**
- **Define empty tables A(x,w), B(x,w,a), C(x,a*)**
- **Stopping-cond = all tuples in T have annotation a* in C**

- **While (stopping-cond = false) and (a new worker w arrives)**
  - x = assign(T,w,A)
  - Remove x from T and add (x,w) to A

  - a = annotate(x,w);
  - If a = null then {put x back into T; break} else {add (x,w,a) to B}

  - a* = aggregate(x, B);
  - If a* = null then {put x back into T} else {add (x,a*) to C}

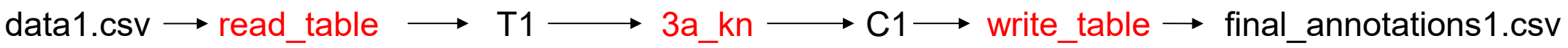
Other Problems

- **How to model crowdsourcing using hosted services, e.g., AMT?**
- **CS workflows are often complex, involving non-crowd operations**
  - Table T => crowdsource using 30 workers => table C
  - Table C => random sample => Table S => crowdsource using 2 data stewards => table D
  - Table C, table D => SQL query => precision of table C
- **How do we model these?**
- **Also want to ensure developers can easily customize and extend**

Our Solution

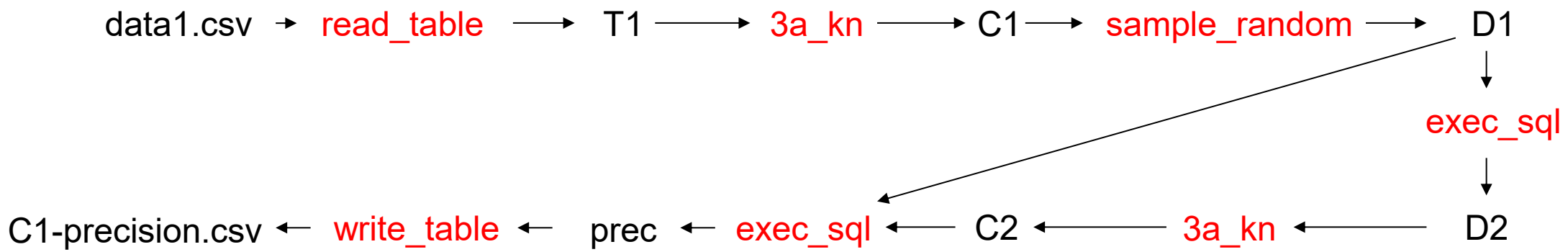
- **Cymphony workflow = DAG**
  - Nodes = operators, edges = flows of input/output data
- **Cymphony operator: X = H(Y)**
  - X and Y refer to tables, files, and other typical values (e.g., int, real, string, etc.)
- **Cymphony program**
  - Specifies the workflow

- **Example**
  - T1 = read_table(“data1.csv”);
  - (B1,C1) = 3a_kn(T1, “instruction1.html”, k=2, n=3);
  - write_table(C1,”final_annotations1.csv”);



Another Example

```
T1 = read_table("data1.csv");
(B1,C1) = 3a_kn(T1,"instruction1.html", k=2,n=3);
D1 = sample_random(C1,500);
D2 = exec_sql(SQL query to drop column final_label from D1 and make a new copy, D1)
(B2,C2) = 3a_kn(D2,"instruction1.html", …);
prec = exec_sql(SQL query to compute precision, D1, C2);
write_table(prec,"C1-precision.csv");
```



More on Cymphony Operators

- **We provide built-in operators**
  - Human operators: take a table I, crowdsource, produce a table O
  - Machine operators: any operator that automatically transforms table/files and other value types
    - Sample_random, arbitrary SQL queries
- **Developers can**
  - Customize existing built-in operators
  - Provide new operators
- **For Cymphony 1.0**
  - Human operators: 3a_kn, 3a_amt
  - Machine operators: sample_random, exec_sql

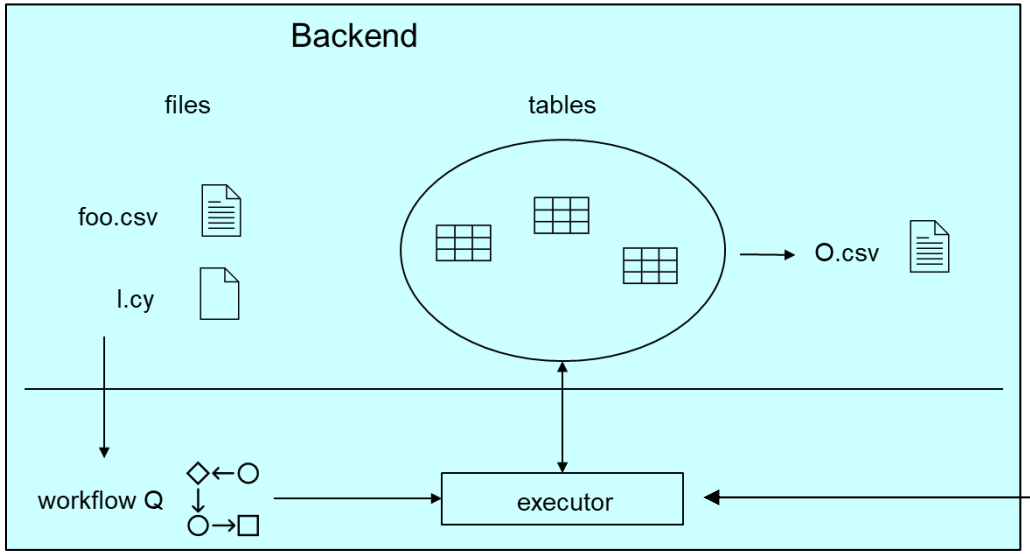
Human Operator 3a_kn

- **For each microtask t, get up to n votes**
  - If at least k agree, then return that as “aggregated annotation”
  - Otherwise return null (undecided)
- **Can capture many common crowdsourcing scenarios**
  - Get up to 3 votes then take majority: k = 2, n = 3
  - Get up to 5 votes then take majority: k = 3, n = 5
  - Using 1 trustworthy data steward: k= 1, n = 1
- **Implement this operator using functions assign/annotate/aggregate discussed earlier**

- **Input: table T**
- **Define empty tables A(t,w), B(t,w,a), C(t,a*)**
- **Stopping-cond = all tuples in T have annotation a* in C**
- **While (stopping-cond = false) and (a new worker w arrives)**
  - t = assign(T,w,A)
  - Remove t from T and add (t,w) to A
  - a = annotate(t,w);
  - If a = null then {put t back into T; break} else {add (t,w,a) to B}
  - a* = aggregate(t, B);
  - If a* = null then {put t back into T} else {add (t,a*) to C}

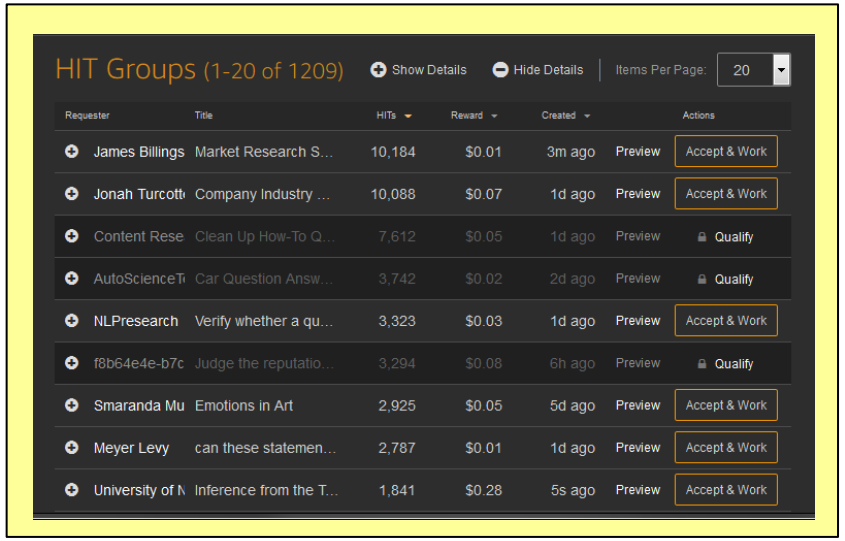
Human Operator 3a_amt

- **For each microtask t**
  - Get n votes from AMT workers
  - If at least k votes agree, then Cymphony returns that as “aggregated annotation”
- **Example**
  - T1 = read_table(“data1.csv”);
  - (B1,C1) = 3a_amt(T1, “instruction.html”, k=2, n=3, workers_from_location=[“US-CA”, “IN”], ...);



Cymphony

API



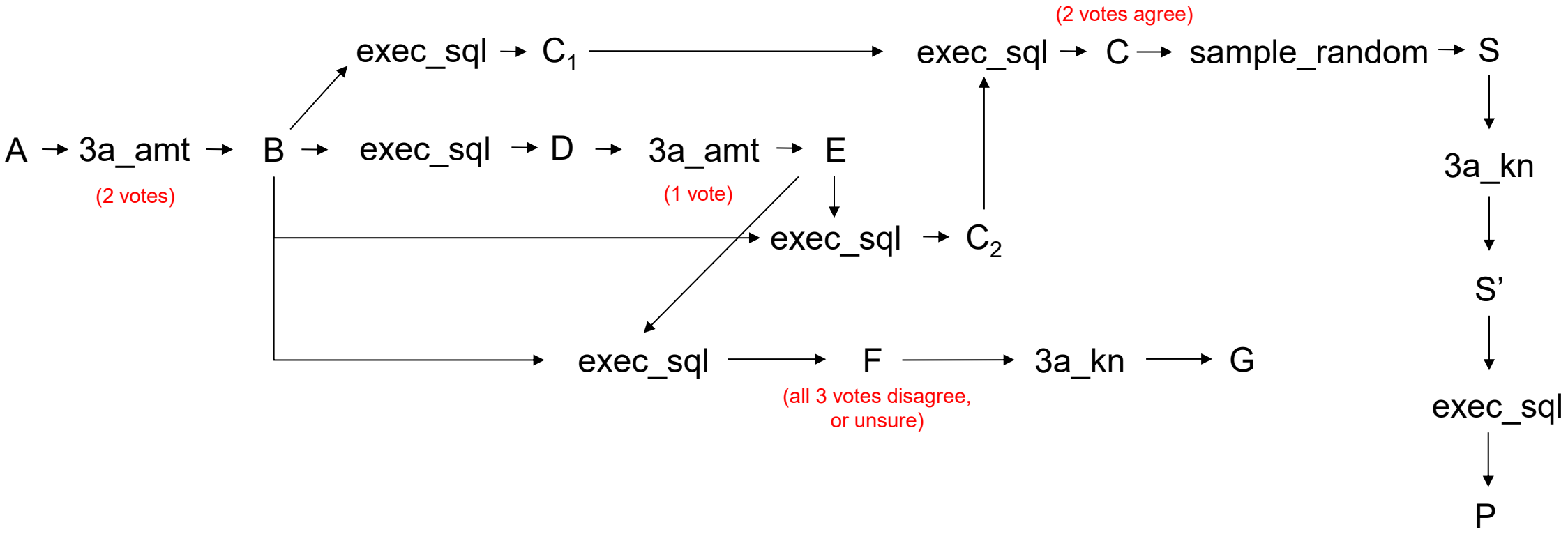
Amazon Mechanical Turk

Built-in Machine Operators

- **Sample_random(D, k) => E**
  - Very common operation in crowdsourcing workflows
- **Exec_sql(a SQL query, tables) => output table**
  - This allows user to execute arbitrary SQL queries to process the data

```
/* queries to attach tuples to the labeled ids and separate out tuples that need more annotations */
(
        AGREEMENTS_WITH_LABELS,
        DISAGREEMENTS_WITHOUT_LABELS
) = exec_sql(
        ORIGINAL_DATA,
        C_1,
        queries = "
                CREATE TABLE temp as (
                        SELECT _id,product_title,label
                        FROM ORIGINAL_DATA
                        INNER JOIN C_1
                        USING(_id)
                );
                CREATE TABLE agreements as (
                        select * from temp where label!='undecided'
                );
                CREATE TABLE disagreements as (
                        select * from temp where label='undecided'
                );
                ALTER TABLE disagreements DROP COLUMN label;
        ",
        mapping_to_output_variables = [
                "temp:None",
                "agreements: AGREEMENTS_WITH_LABELS",
                "disagreements: DISAGREEMENTS_WITHOUT_LABELS"
        ]
);
```

Sample Complex Cymphony Workflows

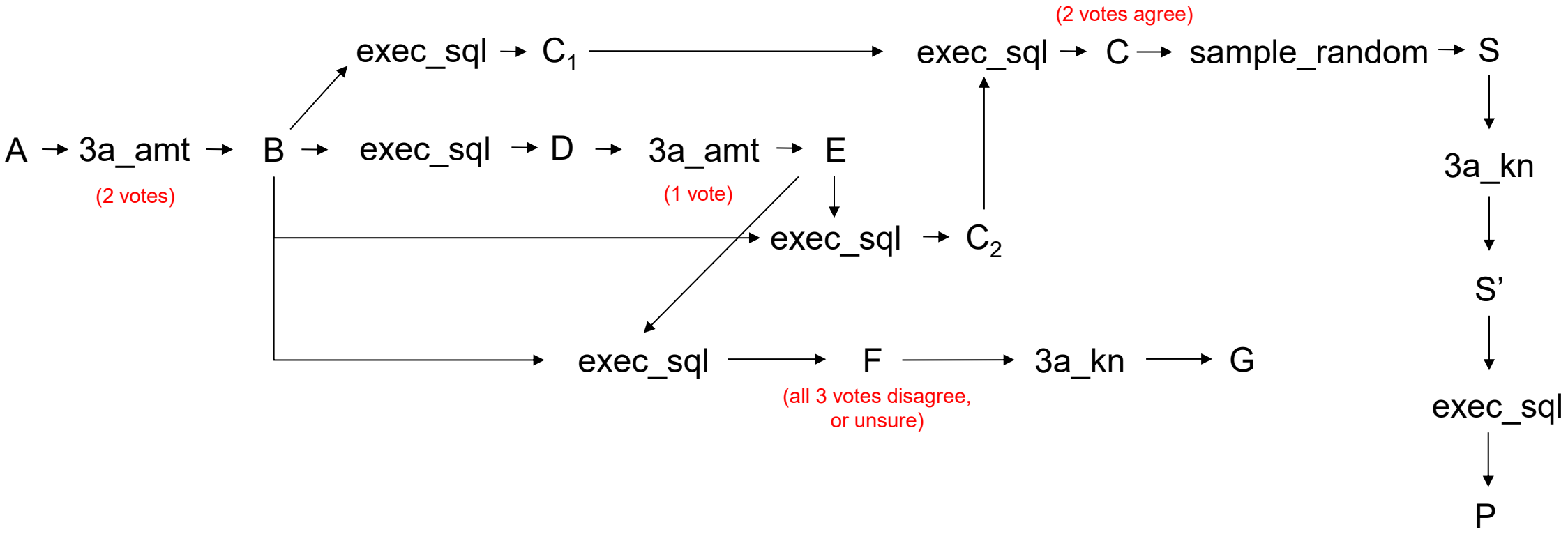


Output = (C \ S) union S' union G

Executing Cymphony Workflows

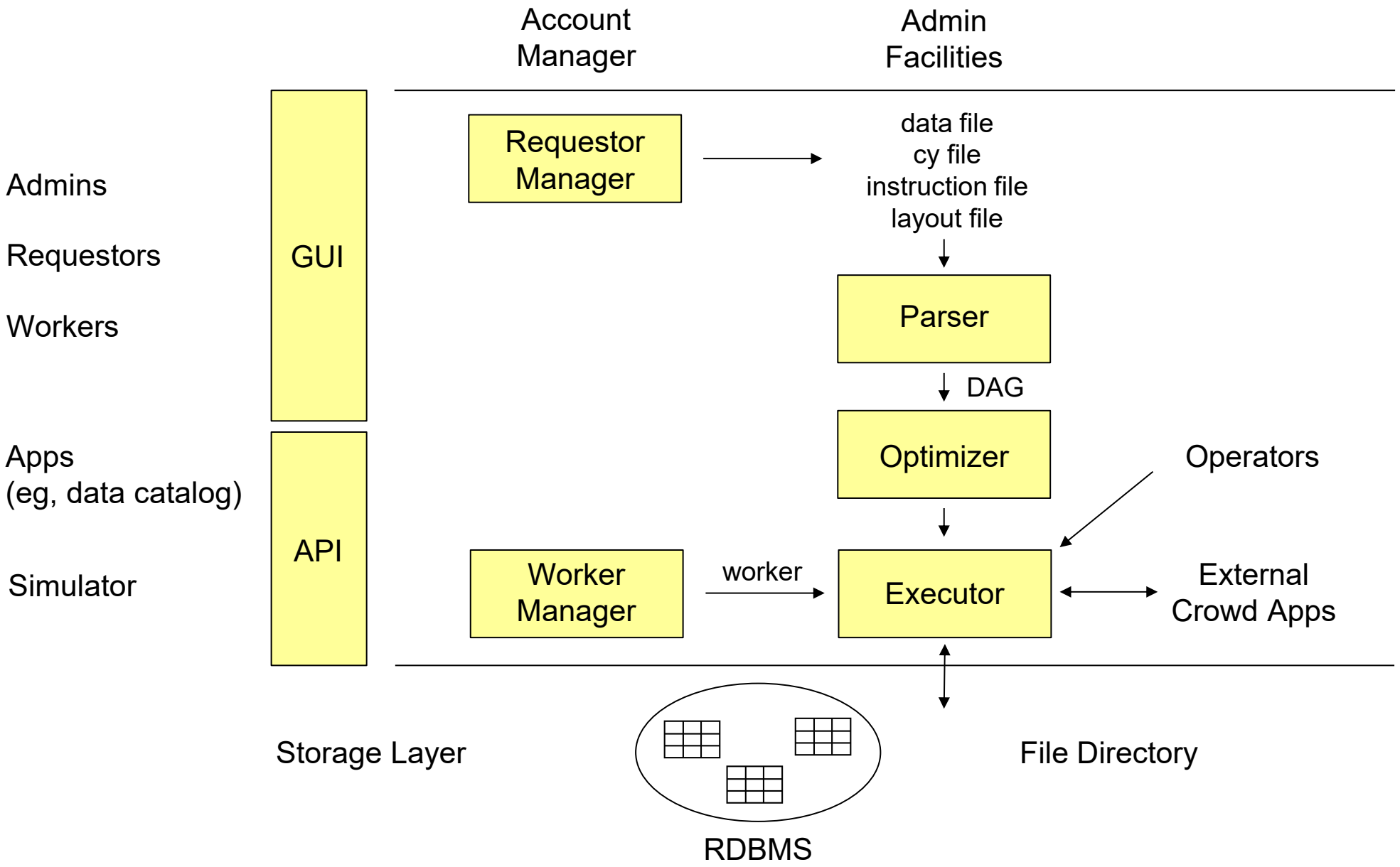
- **Executable operators**
  - A machine operator is executable if its inputs are available
  - A human operator is executable if its non-human inputs (e.g., tables, files) are available
- **For each time epoch (e.g., each second)**
  - Execute all machine operators that are executable
  - If a worker w arrives
    - Assign w to (or let w choose) an executable human operator
    - Execute that human operator
- **Terminate when all operators in the workflow have been executed**

Executing Cymphony Workflows



Output = (C \ S) union S' union G

System Architecture



Current Status

- **Coding from 03/2020 – 03/2023**
- **16,816 lines of code**
- **69,716 total lines**
  - including Python, HTML, JS, CSS, MD files
- **Deployed at Qatar Computing Research Institute in 04/2022**

Applying Cymphony to DI Problems & Experiments

- **Consider 3 problems**
  - Color extraction from textual product descriptions
  - Column classification in data catalog
  - Entity matching
- **Showed that CS workflows to solve them can be very complex**
- **Showed that Cymphony can easily solve these complex CS workflows**
  - Saving significant user effort, achieving state-of-the-art accuracy at a reasonable cost
  - Demonstrating the versability of Cymphony

- **Demonstrated scalability to realistically large dataset and worker sizes**

My Dissertation

- **Introducing the problem of building a CS platform for DI**
- **Designing Cymphony**
  - Define problem, operators, workflows
- **Implementation**
- **Applying Cymphony to DI problems & evaluation**
  - Usability, accuracy, scaling
- **Customizing Cymphony for Data Catalog Systems** 
  - User registration, operators, program, API
- **Releasing Cymphony**
  - Stand-alone, as a system component

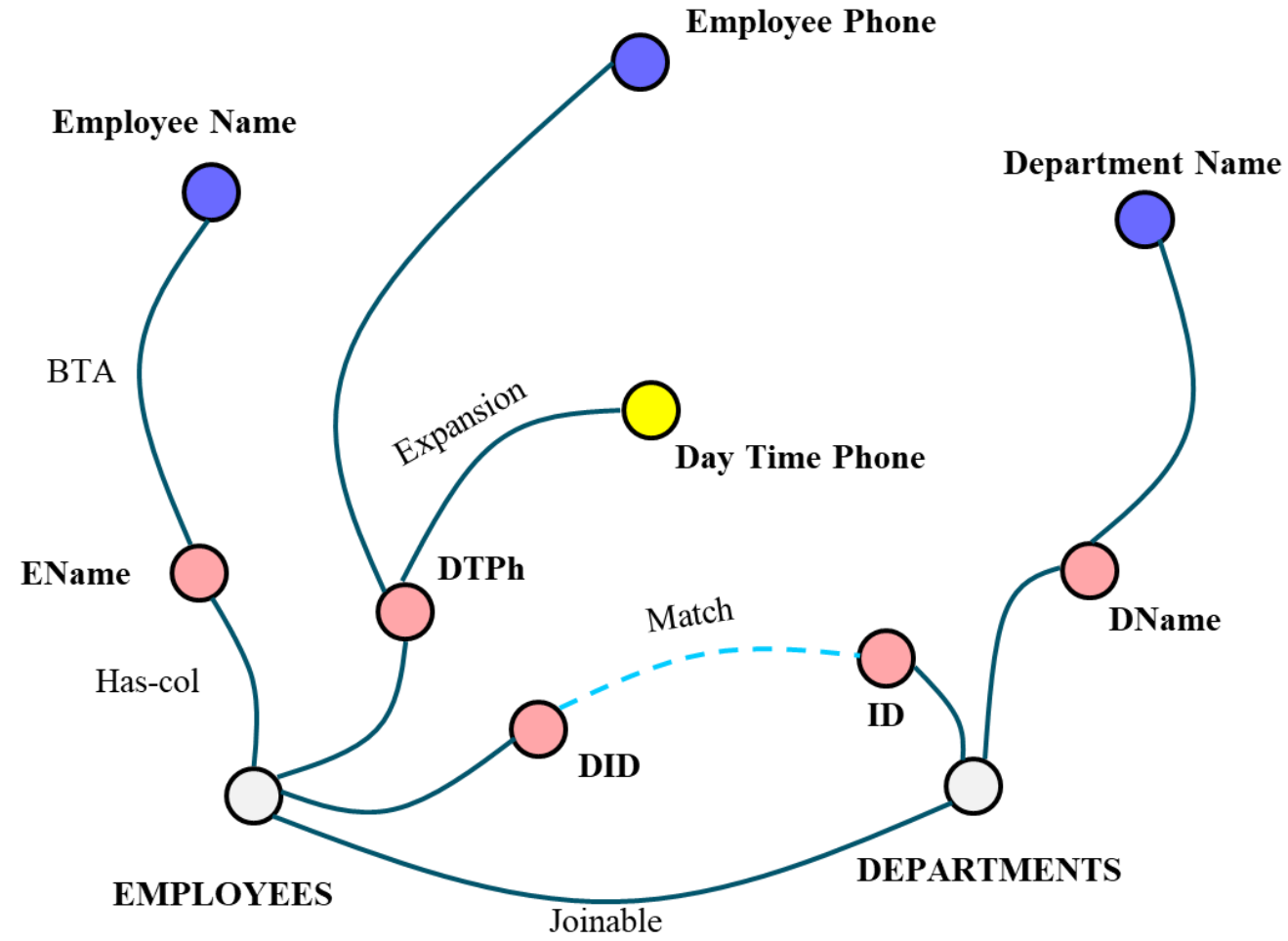
Data Catalog Systems

EMPLOYEES

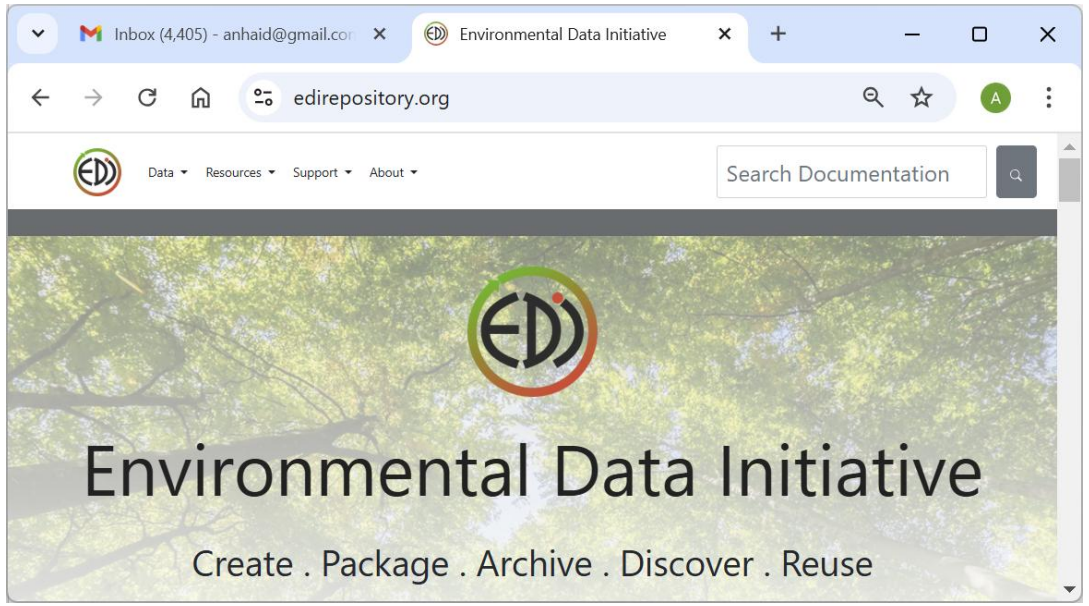
EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

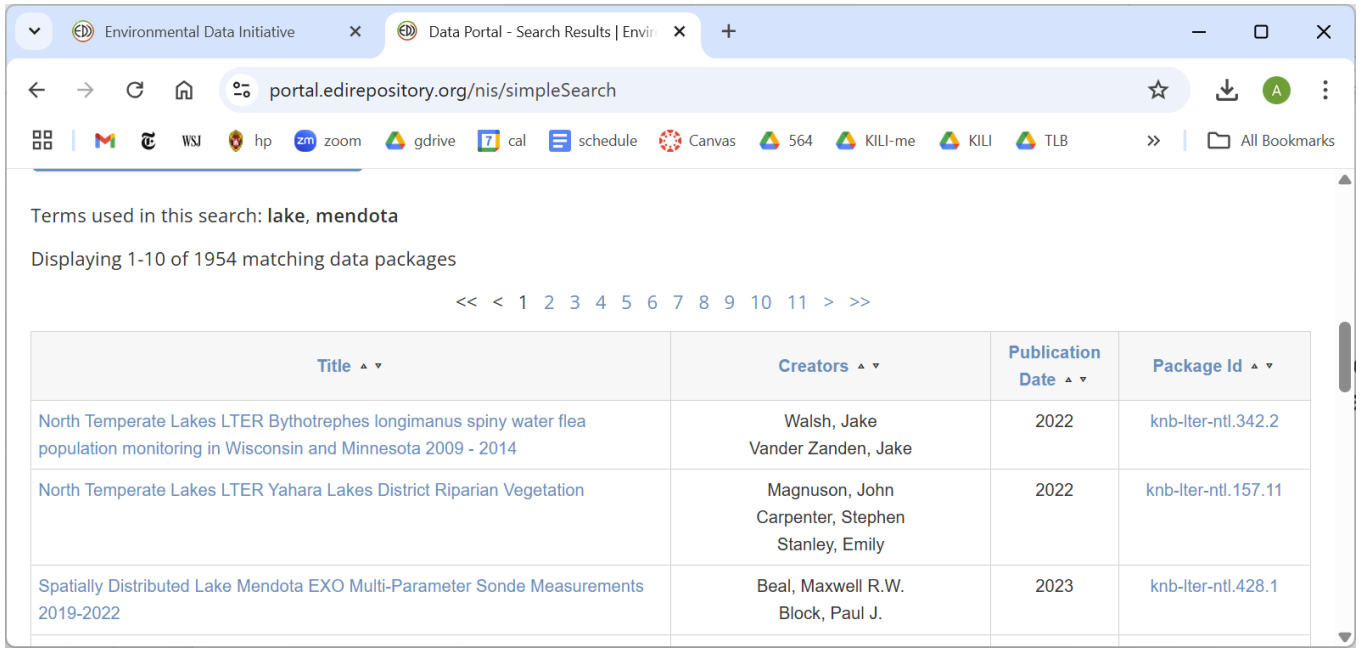
DEPARTMENTS

ID	DName
d_1	Sales
d_2	Legal



Example: Environmental Sciences





Since 2007
87K data packages, 18K tables
60TB storage, 10K downloads / week

- **Other lakes in environmental science**
  - Dryad, CUAHSI, Zenodo, Figshare, BCO-DMO, Artic Data Center, IDigBio

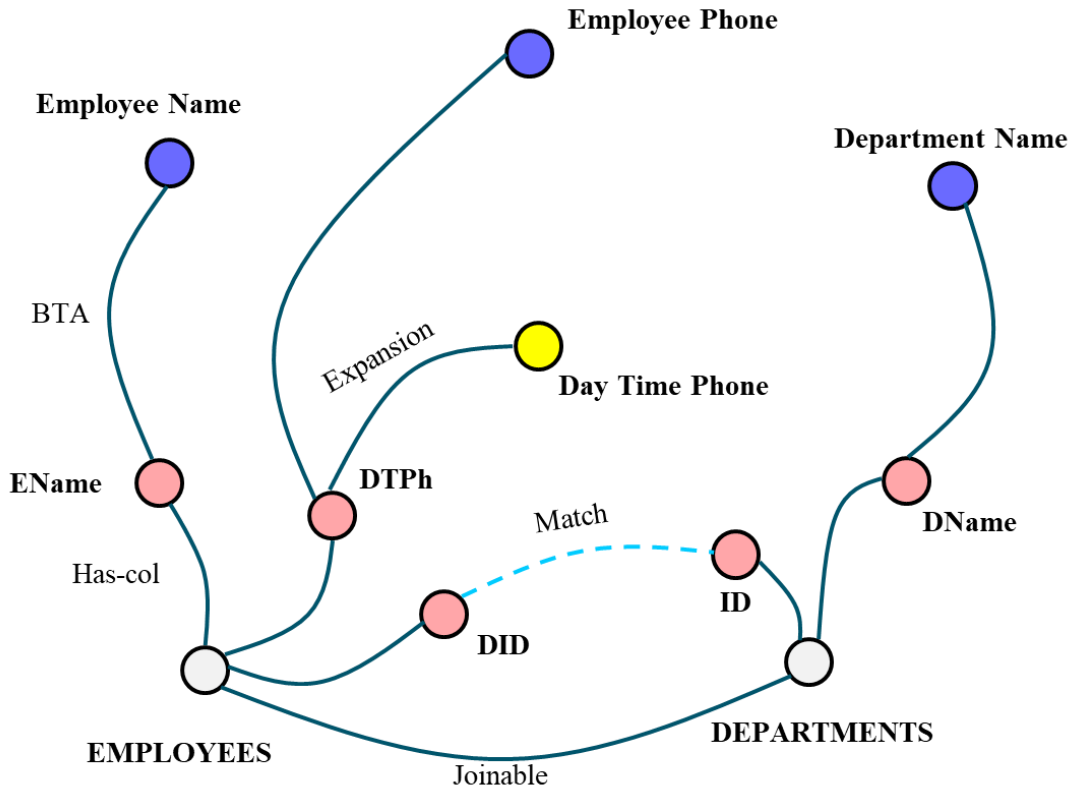
Crowdsourcing Needs for Data Catalog Systems

EMPLOYEES

EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

DEPARTMENTS

ID	DName
d_1	Sales
d_2	Legal



Often called curation

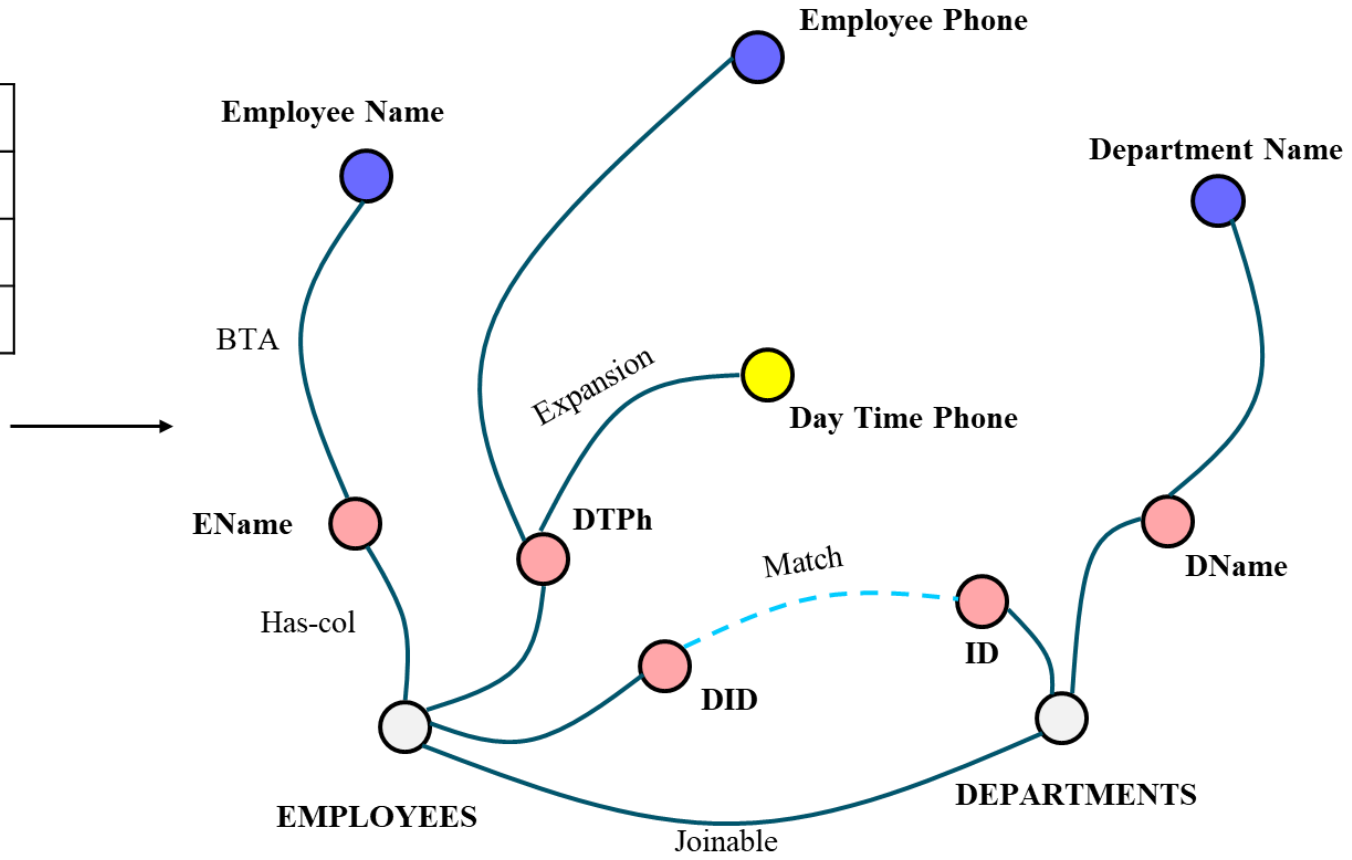
Bulk Curation & Drive-By Curation

EMPLOYEES

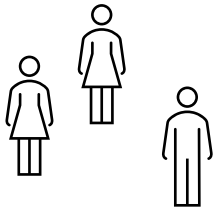
EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

DEPARTMENTS

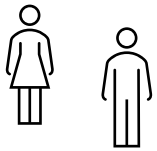
ID	DName
d_1	Sales
d_2	Legal



Users



Data stewards

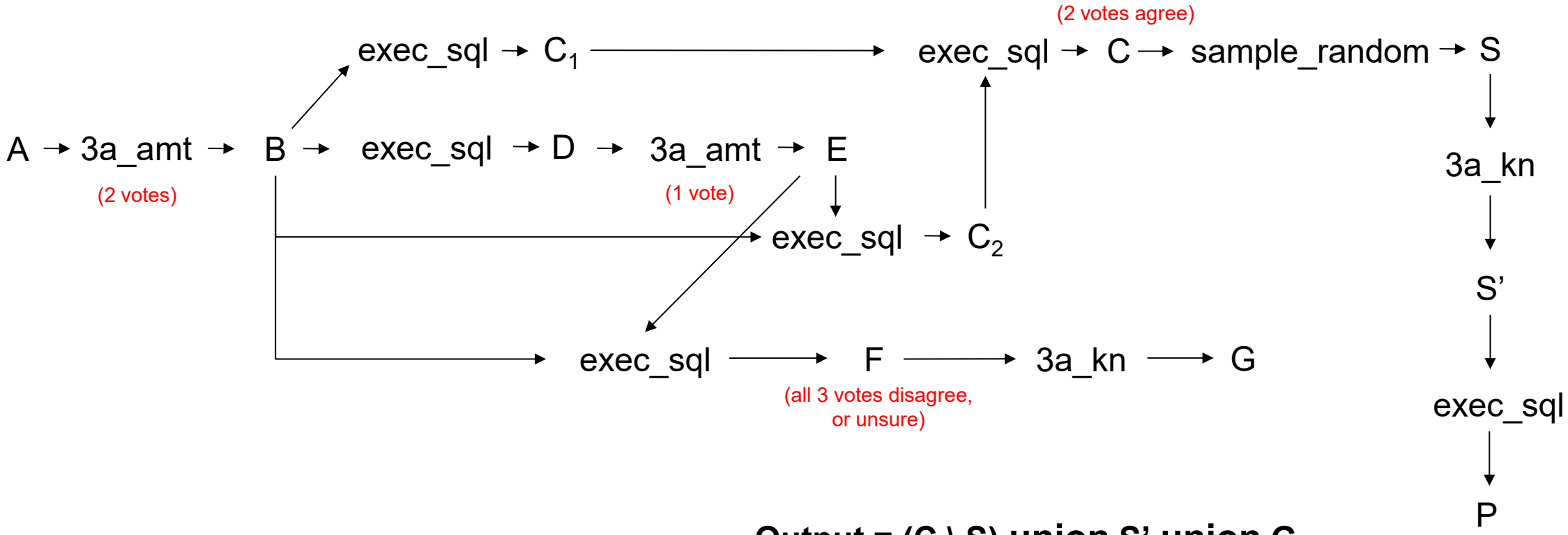


External
crowd platforms

amazon
mechanical turk

Curation Challenges

- **Often need many different curation workflows**
  - Table name expansion: 3 users agree OR 1 data steward agrees
  - Column name expansion: 2 users agree OR 1 data steward agrees
- **Workflows can be quite complex**



Output = (C \ S) union S' union G

No Good Solutions Today

- Curation workflows are often hardwired into the code of data catalogs
- Very difficult to implement, debug, maintain, modify

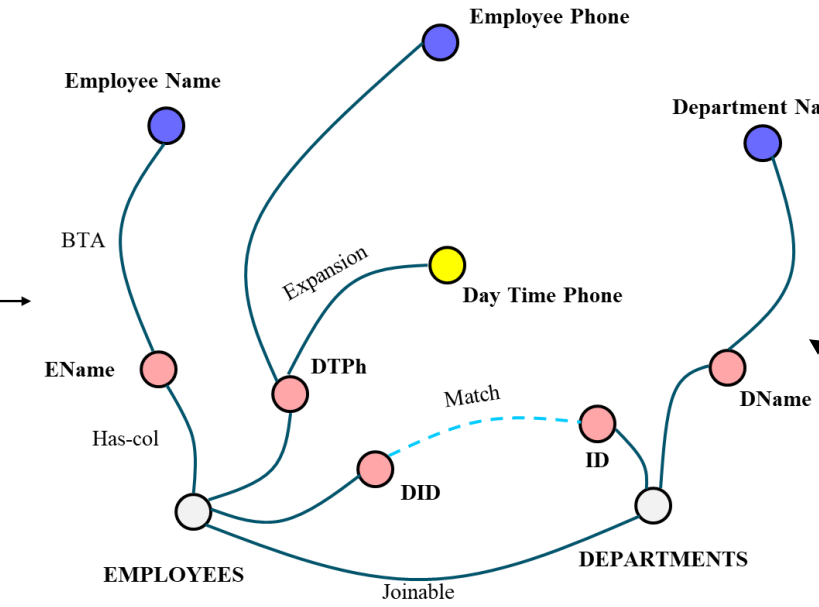
Our Proposal: Using Cymphony

EMPLOYEES

EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

DEPARTMENTS

ID	DName
d_1	Sales
d_2	Legal

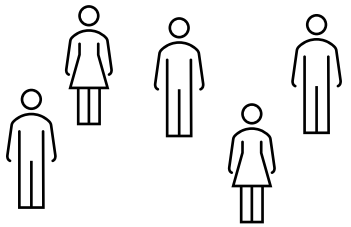



```
T1 = read_table("data1.csv");
(B1,C1) = 3a_kn(T1,"instruction1.html", k=2,n=3);
D1 = sample_random(C1,500);
D2 = exec_sql(SQL query to drop column final_label from D1 and make a new copy, D1)
(B2,C2) = 3a_kn(D2,"instruction1.html", …);
prec = exec_sql(SQL query to compute precision, D1, C2);
write_table(prec,"C1-precision.csv");
```

data1.csv → read_table → T1 → 3a_kn → C1 → sample_random → D1

exec_sql

D2 → 3a_kn → C2 → exec_sql → prec → write_table → C1-precision.csv



amazon mechanical turk

- **Will study this in the context of SmartCat**

Customizing SmartCat for Curation

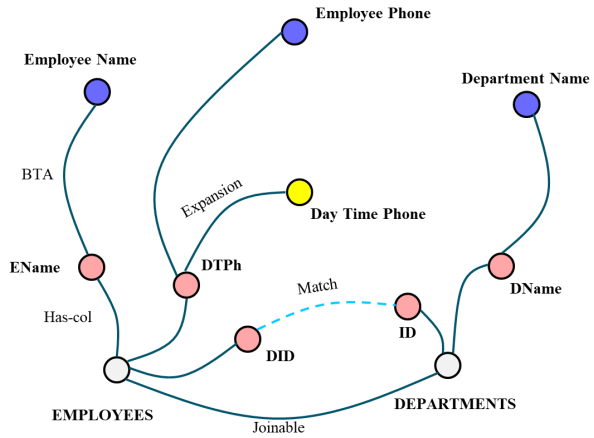
- **SmartCat operates in epochs**
- **In each epoch**
  - Runs feeders, enrichers, consumers
- **Now in each epoch**
  - Runs feeders, enrichers, curators, consumers
- **System admin can write two kinds of curators**
  - Bulk and drive-by

EMPLOYEES

EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

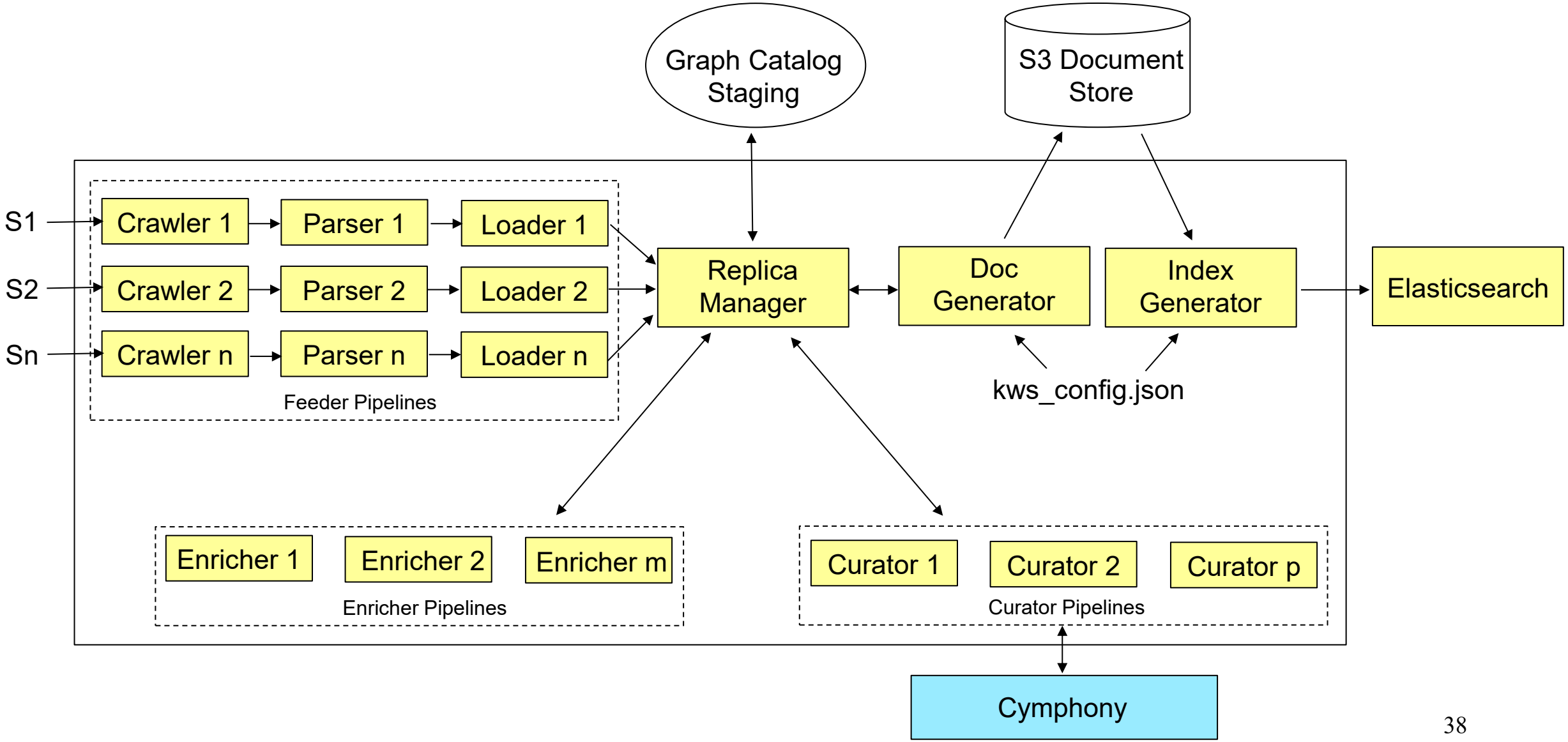
DEPARTMENTS

ID	DName
d_1	Sales
d_2	Legal



Keyword search
NL querying
Browsing

The Smartcat Data Catalog System



Bulk Curators

- **Consider a bulk curator X that curates column name expansion**
- **Admin obtains a table E of column name expansions**
  - E.g., 200 expansions
- **Admin writes a Cymphony program F to curate these expansions**
  - E.g., for each expansion, solicit up to 3 yes/no votes, take majority

- **In an epoch**
  - SmartCat sends table E and program F to Cymphony (via APIs)
  - Cymphony starts a run R to curate table E

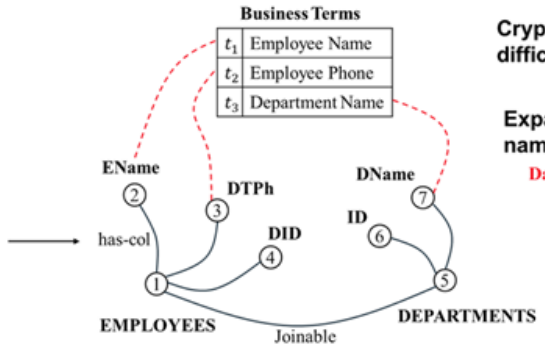
- **In subsequent epochs**
  - SmartCat can ping Cymphony for status on run R

EMPLOYEES

EName	DTPh	DID
Dave Smith	4399	d_1
Jane Miller	5603	d_2
Mike Davis	2862	d_1

DEPARTMENTS

ID	DName
d_1	Sales
d_2	Legal



Cryptic names, difficult to search

Expand column names

Day Time Phone

Workers = data stewards (1 vote), catalog users (3 votes)

Table E + Workflow F

Run ID

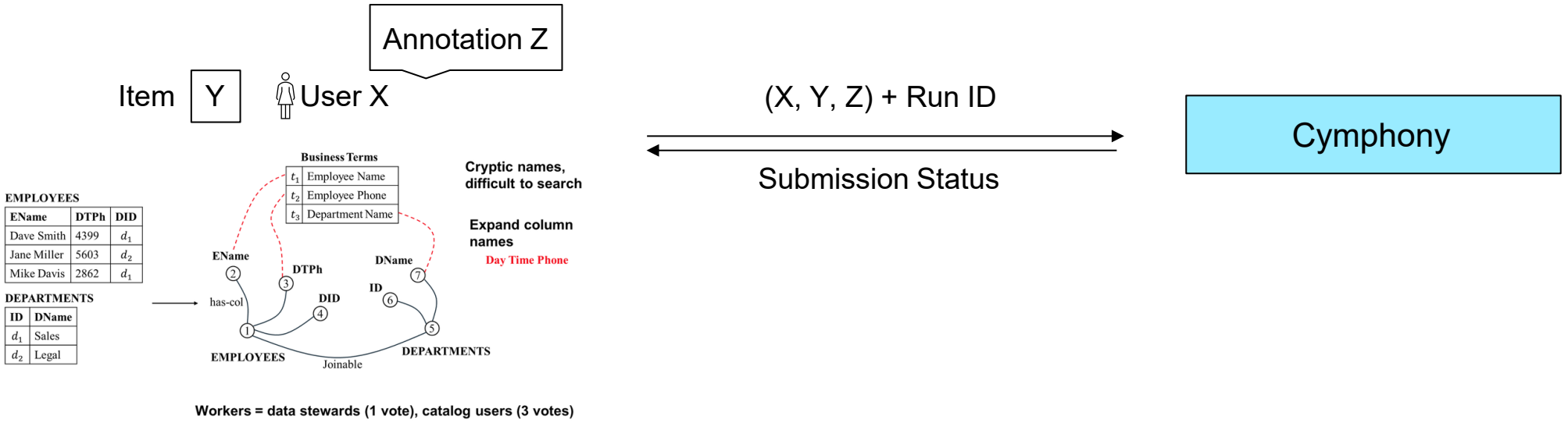
Check Run Status

Notify Completion

Cymphony

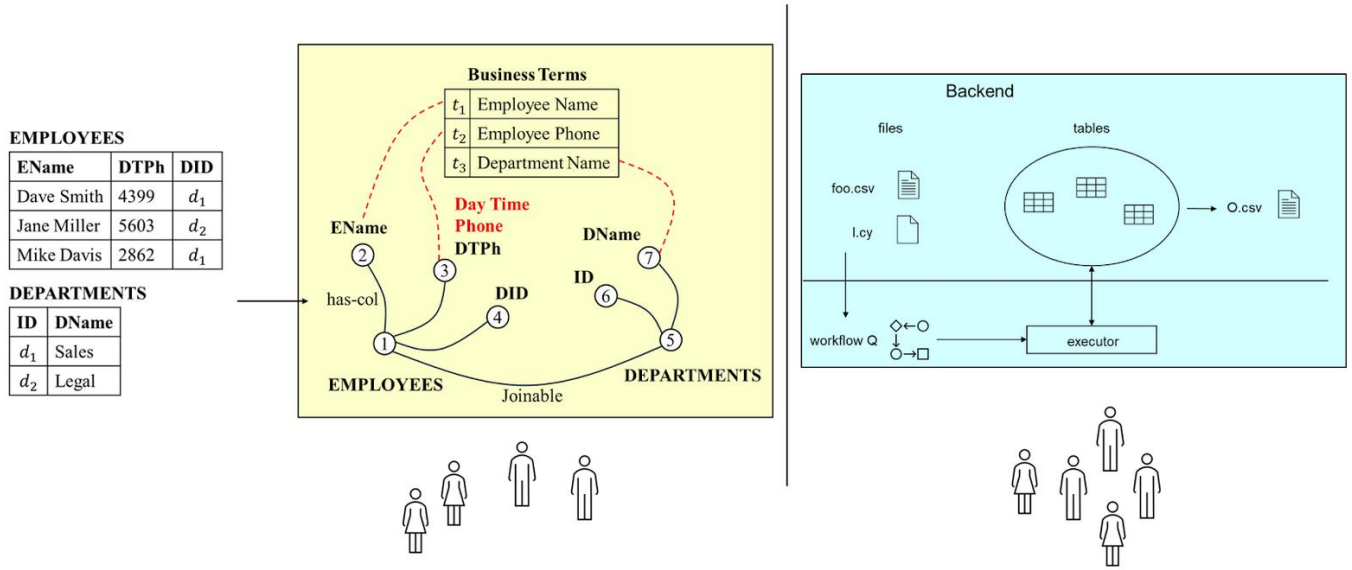
Drive-By Curators

- Similar to Bulk Curators
- But allows catalog users to provide feedback on the GUI
- Feedback is collected in a table G
- Occasionally G is sent to Cymphony to be pushed into run R



Realizing Our Proposed Solution

- **Support multiple worker roles**
  - Customize Cymphony's user management model
- **Support SC-style curation workflows**
  - Customize Cymphony's operators and program model
- **Support different forms of interaction required by SC**
  - API-driven integration of SC with Cymphony



Customize Cymphony's User Management: Summary

- **The original Cymphony supports only a single user type**
  - Called "regular users", can give noisy curations
- **We modified it to support two more user types for SmartCat**
  - Stewards: give correct curations
  - Admins: responsible for setup and maintenance

Customize Cymphony's User Management

- **Motivation**
  - Original Cymphony platform primarily designed around a uniform user population
  - SC explicitly requires multiple human roles with different expertise assumptions
- **Regular users**
  - correspond to SC business users
  - characterized by noisy annotations
- **Steward users**
  - correspond to SC stewards
  - characterized by trustworthy annotations
- **Admins**
  - correspond to SC admins
  - responsible for setup and system maintenance

Design of User Management Customizations

- **Conceptual user onboarding workflow**
  - Admin deploys Cymphony.
  - Admin accesses a special admin-only interface or programmatic path to create accounts and assign roles such as admin, steward, or (regular) user.
  - Admin shares initial account details with these users.
  - Users accessing the public signup flow automatically assigned the “user” role by default.
- **Design decisions to support the workflow**
  - Use built-in Django role groups.
  - Two ways for admin to manage users: GUI and programmatic approach.
  - Automated default role assignment.
  - Must be able to recognize user roles when they participate in labeling tasks later.

Implementation of User Management Customizations

- **Create three Django role groups:**
  - "admin", "steward", and "user"
- **Enable and configure the Django admin interface**
  - Admins can create users, assign users to groups, and manage permissions
- **Implement a Django management command: "manage_roles_and_users"**
  - Enables admins to create role groups, create user accounts, and assign users to groups
- **Implement a "post_save" signal handler**
  - Automatically assigns newly registered users to the "user" group
  - Ensures self-service signups are categorized without manual intervention

Implementation of User Management Workflow

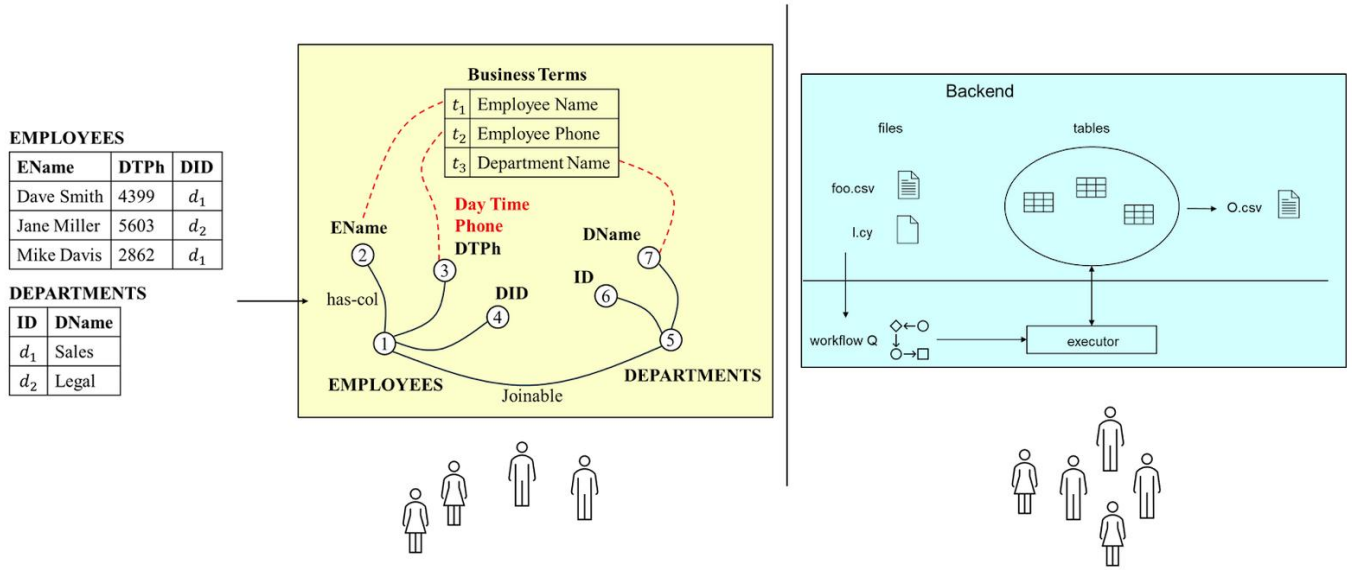
- **GUI-based setup**
  - Admin creates a superuser and logs into the Django admin panel.
  - Admin creates the three role groups.
  - Admin creates privileged accounts (admins and stewards) and assigns them to appropriate groups.
  - Public users sign up via the standard Cymphony registration flow.
- **Programmatic setup**
  - Admin runs “manage_roles_and_users” to create role groups.
  - Admin creates users and assigns them to roles via command-line flags.
  - New users signing up through the public interface are still auto-assigned to the “user” group via the signal handler.
- **Customizations fully integrated with original user management system**
  - Preserving standard security mechanisms such as password hashing and session management.

Underlying Data Structures

- **AUTH_USER(id, username, password, first_name, last_name, email)**
  - Stores core user identity and credential information used for authentication.
- **AUTH_GROUP(id, name)**
  - Stores the three role groups (“admin”, “steward”, “user”).
- **AUTH_USER_GROUPS(id, user_id, group_id)**
  - Join table that stores assignment of users to groups.

Realizing Our Proposed Solution

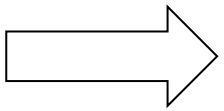
- **Support multiple worker roles**
  - Customize Cymphony's user management model
- **Support SC-style curation workflows**
  - Customize Cymphony's operators and program model
- **Support different forms of interaction required by SC**
  - API-driven integration of SC with Cymphony



Motivation

- **As a reminder, many DI tasks follow a common structure:**
  - Input table → Human annotation → Aggregated output table
  - Examples include information extraction, classification, and entity matching

Product Title	Price
"ASUS X205TA 11.6 Inch Laptop (Intel Atom, 2 GB, 32GB SSD, Gold) - Free Upgrade to Windows 10"	$199.00
"Lenovo G50 Entertainment Laptop - Black: DOORBUSTER - Intel Core i7-5500U, 8GB RAM, 1TB HDD, 15.6\ FHD 1080P Display, USB3.0, HDMI,Bluetooth, Windows 8.1"	$799.77
...	...



Product Title	Price	Final Label
"ASUS X205TA 11.6 Inch Laptop (Intel Atom, 2 GB, 32GB SSD, Gold) - Free Upgrade to Windows 10"	$199.00	32GB
"Lenovo G50 Entertainment Laptop - Black: DOORBUSTER - Intel Core i7-5500U, 8GB RAM, 1TB HDD, 15.6\ FHD 1080P Display, USB3.0, HDMI,Bluetooth, Windows 8.1"	$799.77	1TB
...	...	...

Motivation

- **Cymphony abstracts this process using operators**
- **Baseline Operator: 3a_kn**
  - Implements assign–annotate–aggregate pattern with (k, n) as parameters
  - Collect up to n votes; if ≥ k agree → finalize label; otherwise → NULL (undecided)
  - Treats all workers identically
- **Smartcat distinguishes regular users (noisy) and stewards (trusted)**
- **Realistic requirement:**
  - Either majority of regular workers, or fewer high-trust steward votes
  - For example, 2 out of 3 regular workers OR 1 steward vote
  - Generally, (k out of n regular votes) OR (l out of m steward votes)

Design of Human Operator 3a_knlm

- **For each task t, get up to n regular votes or up to m steward votes**
  - If at least k regular worker votes agree or at least l steward votes agree, then return that as “aggregated annotation”.
  - Otherwise return null (undecided).
- **Example: k = 2, n = 3 ; l = 1, m = 1**
  - Case 1: w1 = “Yes”, w2 = “Yes” → finalize “Yes”
  - Case 2: s1 = “Yes” → finalize “Yes”
  - Case 3: w1 = “Yes”, w2 = “No”, w3 = “Cannot Determine” → finalize “NULL”

Implementation of Human Operator 3a_knlm

- **Implement using functions assign/annotate/aggregate discussed earlier**
  - with modification only in aggregate step
- **For each task t:**
  - Collect annotations
  - Identify worker role (r)
  - Trigger role-specific aggregation for task t and role r
    - Join annotations with user-group tables
    - Filter by task t and role r
    - Group by annotation, and count votes
    - Compare vote count to corresponding threshold, and aggregate as before
- **No cross-pool mixing**
  - regular and steward votes are evaluated independently

Underlying Job-Scoped Tables (Unchanged Structure)

- **Tuples table T:**
  - ui_pj_wk_rm_jn_tuples(task_id, …)
  - Stores the input tuples to be curated as part of the job

- **Tasks table:**
  - ui_pj_wk_rm_jn_tasks(task_id, total_assigned, abandoned, pending_annotations, done)
  - Maintains task-level execution metadata required to coordinate concurrent worker arrivals

- **Assignments table A:**
  - ui_pj_wk_rm_jn_assignments(task_id, worker_id, status, timeout_threshold_at, abandoned_at, completed_at)
  - Records task assignments to individual workers and tracks their lifecycle

Underlying Job-Scoped Tables (Unchanged Structure)

- **Outputs table B:**
  - ui_pj_wk_rm_jn_outputs(task_id, worker_id, annotation)
  - Persists raw annotations submitted by workers
  - Also serves as the input relation for role-aware aggregation in the 3a_knlm operator

- **Final labels table C:**
  - ui_pj_wk_rm_jn_final_labels(task_id, label)
  - Stores the aggregated label produced for each task once a consensus condition is met

Why 3a_knlm Is a Meaningful Extension

- Enables heterogeneous trust models
- Accelerates convergence via expert votes
- Demonstrates operator-level extensibility

Do We Need to Customize the CY Program?

- We introduced a new role-aware operator (3a_knlm)
- Question: does Smartcat require changes to the program model?
- Or is operator-level customization sufficient?

Reminder: The Cymphony Programming Model

- **Cymphony workflow = DAG**
  - Nodes = operators, edges = flows of input/output data
- **Cymphony operator: X = H(Y)**
  - X and Y refer to tables, files, and other typical values (e.g., int, real, string, etc.)
- **Cymphony program**
  - Specifies the workflow

- **Example**
  - T1 = read_table(“data1.csv”);
  - (B1,C1) = 3a_kn(T1, “instruction1.html”, k=2, n=3);
  - write_table(C1,”final_annotations1.csv”);

Reminder: The Cymphony Programming Model

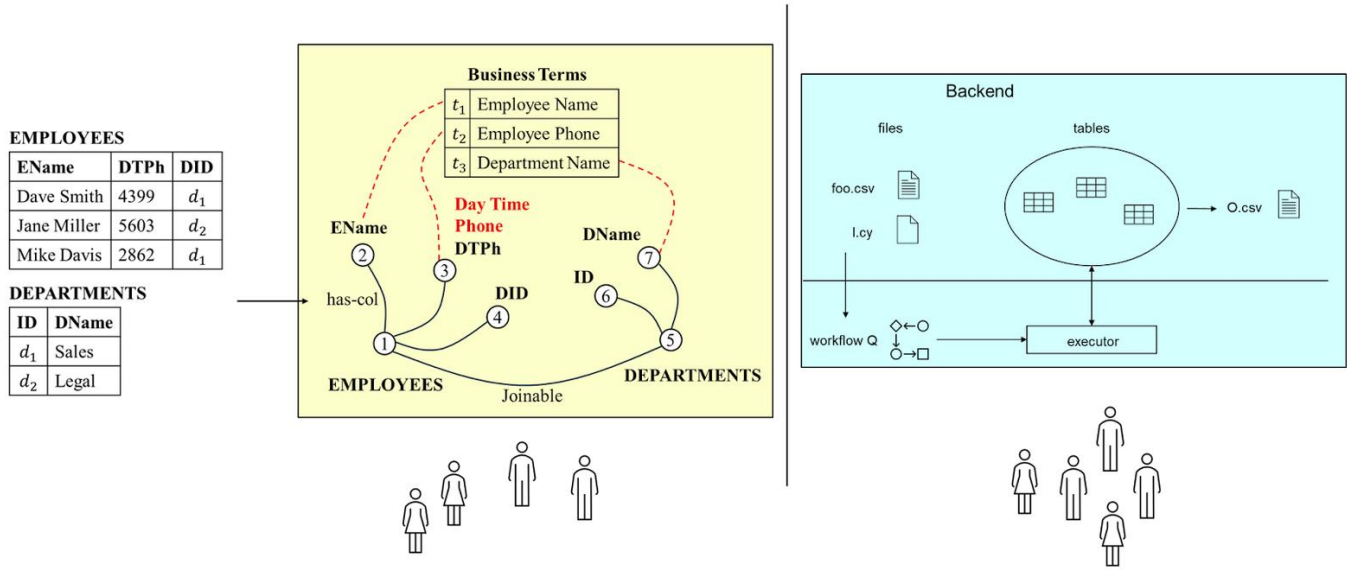
- **Example**
  - T1 = read_table(“data1.csv”);
  - (B1,C1) = 3a_kn(T1, “instruction1.html”, k=2, n=3);
  - write_table(C1,”final_annotations1.csv”);
- **What Does This Program Assume?**
  - Single worker population with uniform trust
  - Single (k, n) aggregation policy with no role differentiation
- **Smartcat Needs Role-Aware Aggregation**
  - Regular workers and stewards
  - Mixed voting policy with k-of-n regular votes OR l-of-m steward votes
- **Question:**
  - Does this require new syntax or control structures?

Using 3a_knlm in the Program

- **Workflow Using 3a_knlm**
  - T1 = read_table("data1.csv");
  - (B1, C1) = 3a_knlm(T1, "instruction1.html", k = 2, n = 3, l = 1, m = 1);
  - write_table(C1, "final_annotations1.csv");
- **Observations**
  - No changes to workflow semantics
  - No changes to the underlying DAG abstraction
  - Only the operator changes
- **Discussion**
  - Behavior evolves at operator layer
  - Developers customize without rewriting system
  - Heterogeneous trust integrates cleanly

Realizing Our Proposed Solution

- **Support multiple worker roles**
  - Customize Cymphony's user management model
- **Support SC-style curation workflows**
  - Customize Cymphony's operators and program model
- **Support different forms of interaction required by SC**
  - API-driven integration of SC with Cymphony



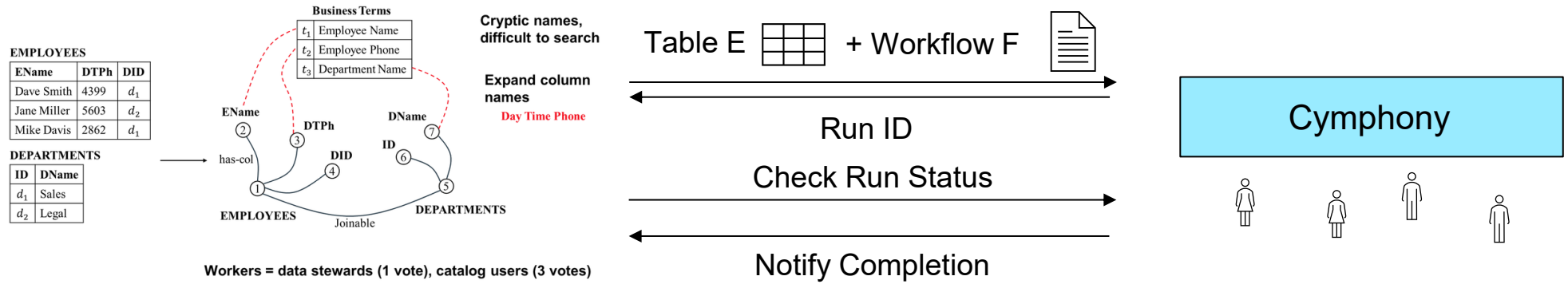
Motivation from Practical Catalog Workflows

- **Two Realistic Curation Modes**
  - Drive-by curation: happens inline while browsing/searching
  - Bulk curation: dedicated session with explicitly assigned tasks

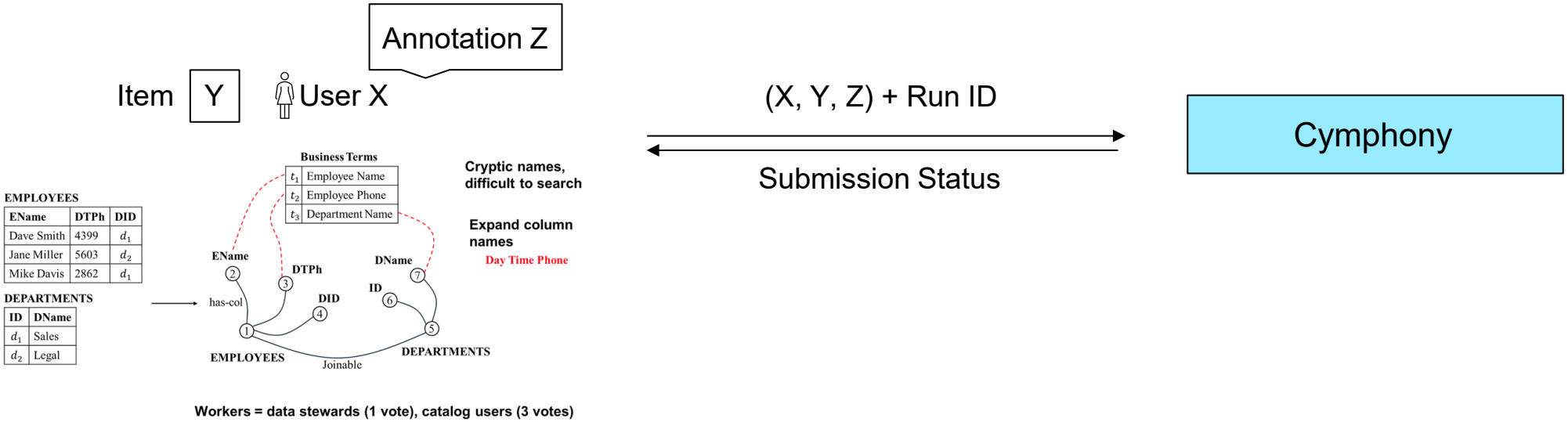
- **We need the CN–SC interface to support:**
  - Submitting individual curation actions as they occur (drive-by)
  - Supporting long-running sessions with workers/stewards (bulk)
  - Exchanging identifiers, annotations, and aggregated outputs

Motivation from Practical Catalog Workflows

- **Customize the CN–SC interface around two complementary modes:**
  - Bulk curation:



  - Drive-by curation:



Challenges in Integration Design

- **Drive-By Before a Run Exists**
  - Smartcat may accumulate candidate expansions in table E
  - Users might start validating items
  - But no CN run exists yet

- **Decision:**
  - Drive-by enabled only after workflow F is published to CN
  - Only for tuples in submitted table E

Challenges in Integration Design

- **Excess Drive-By Annotations**
  - In 3a_kn, at most n annotations per task needed
  - But SC may receive more drive-by annotations than needed

- **Design Choice:**
  - SC forwards all annotations
  - CN applies first-come-first-served and enforces vote thresholds internally

Challenges in Integration Design

- **Concurrent Drive-By and Bulk Annotations**
  - Annotations may arrive from SC (drive-by), or from workers interacting with CN (bulk)
  - How do we combine them?

- **Decision:**
  - First-come-first-served semantics
  - Early annotations can finalize a task
  - Later ones ignored if consensus already reached

Challenges in Integration Design

- **Identity of Drive-By Users**
  - Option 1: Unified authentication between SC and CN
  - Option 2: CN treats drive-by annotations as originating from a single logical entity

- **We choose:**
  - All drive-by annotations submitted under a single SC-controlled CN account

- **Benefits:**
  - No shared identity infrastructure
  - No cross-system auth coupling
  - Easier deployment

Challenges in Integration Design

- **Multiple Human Operators per Run**
  - If a workflow contains multiple human operators, a single run id, but multiple internal job ids
  - Drive-by routing becomes ambiguous

- **Design Choice**
  - Although CN fully supports multiple human operators
  - Restrict the SC–CN boundary to avoid API complexity
  - Each Smartcat-issued curation run should contain exactly one human operator

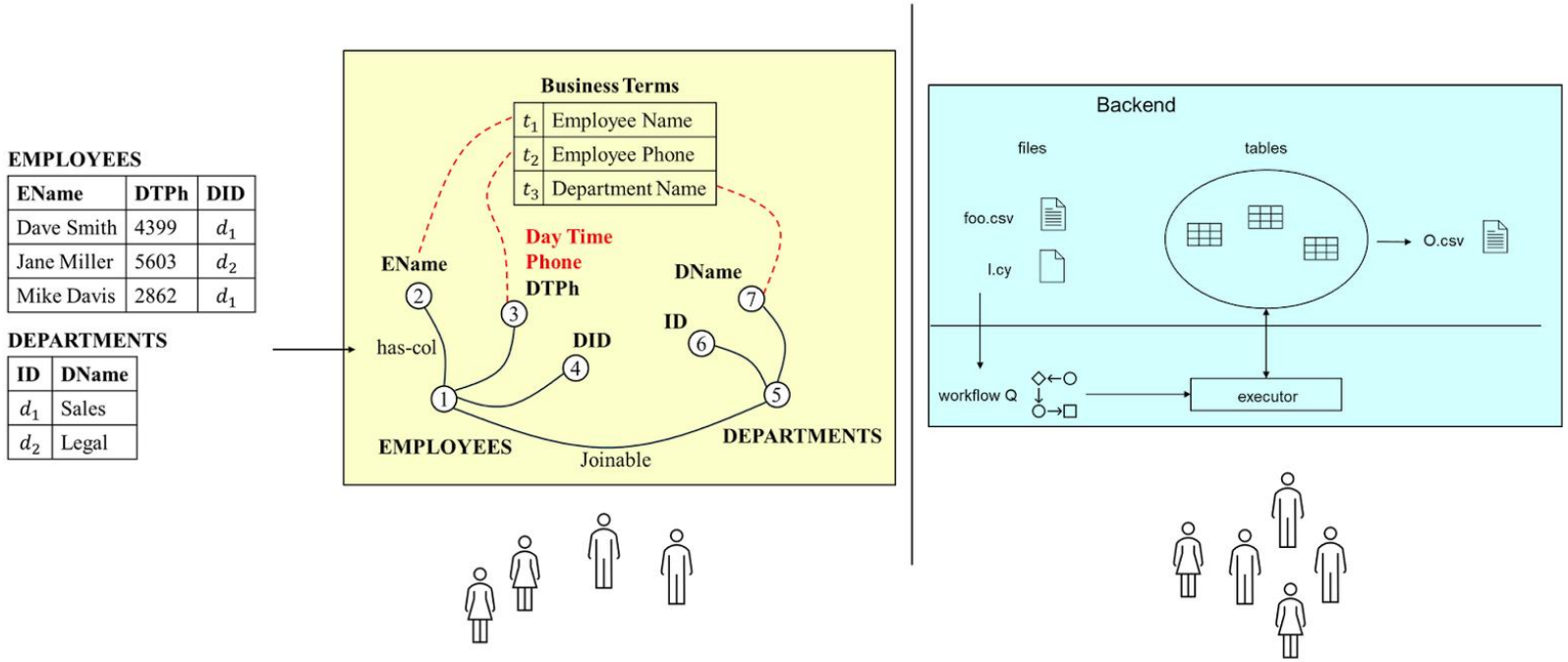
- **Avoids**
  - Exposing job ids
  - Workflow-specific mappings in SC
  - Tight coupling

Integration Design Principles

- Scope out catalog-internal lifecycle complexity
- Push curation and aggregation semantics into CN
- Preserve loose coupling
- Avoid exposing execution artifacts
- Keep routing unambiguous via run identifier

API Surface Overview

- **Define a small, focused set of APIs:**
  - Create Curation Run
  - Submit Drive-By Curations
  - Monitor Run Execution
  - Retrieve Execution Artifacts
  - Run Completion Notification

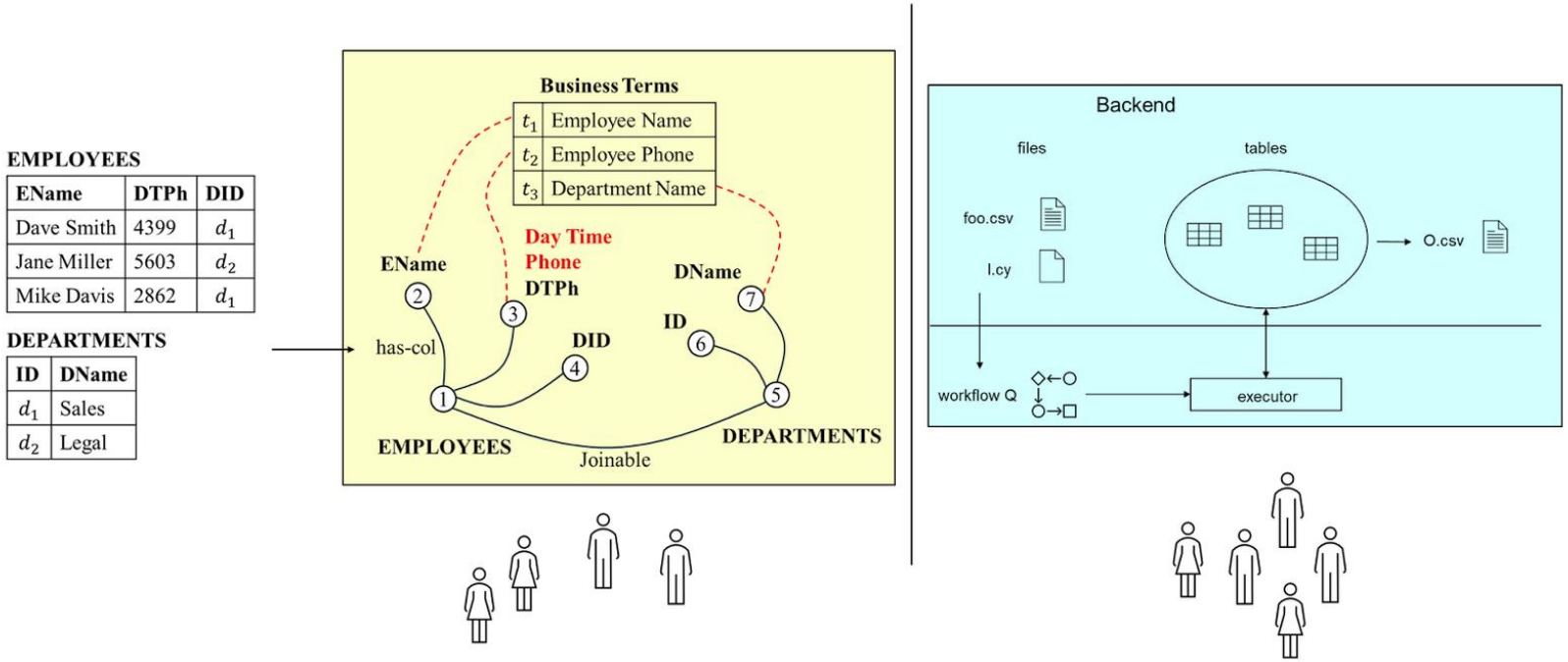


API Surface Overview (Cymphony)

API	Input	Response	Description
Create Curation Run	• Input table T (CSV) • Tuple identifier field • Workflow F (.cy program) • Instruction + layout HTML • Callback URL (webhook)	• Run identifier • Status code • Diagnostic message	Instantiates a run as a first-class execution object in Cymphony
Submit Drive-By Curations	• Run identifier • List of Triples ⟨x,u,v⟩	• Status code • Diagnostic message	Records annotations, updates execution state, and immediately attempts aggregation
Monitor Run Execution	Run identifier	Run Status	
Retrieve Execution Artifacts	• Run identifier • List of logical table names	CSV files, bundled in ZIP archive	

API Surface Overview (Smartcat)

API	Input	Response	Description
Notify (Smartcat) about Run Completion	• Run identifier • Run status • Timestamp		Avoids polling delays by sending a run completion callback to Smartcat.



Integration Design Summary

- **Cymphony owns:**
  - Run lifecycle
  - Execution semantics
  - Aggregation

- **Smartcat owns:**
  - UI + user interaction
  - Catalog state

- **Boundary:**
  - Clean
  - API-driven

Integration Design Summary

- Supports heterogeneous roles
- Supports dual interaction modes
- Preserves execution model integrity
- Avoids identity coupling
- Avoids exposing internal DAG/job structure
- Demonstrates real embedding of CN in production-like system

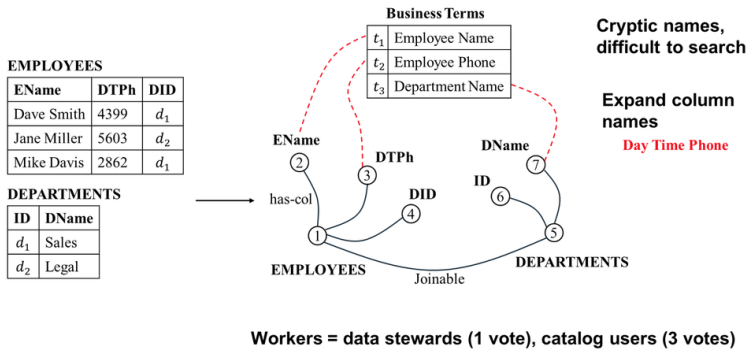
Integration Implementation

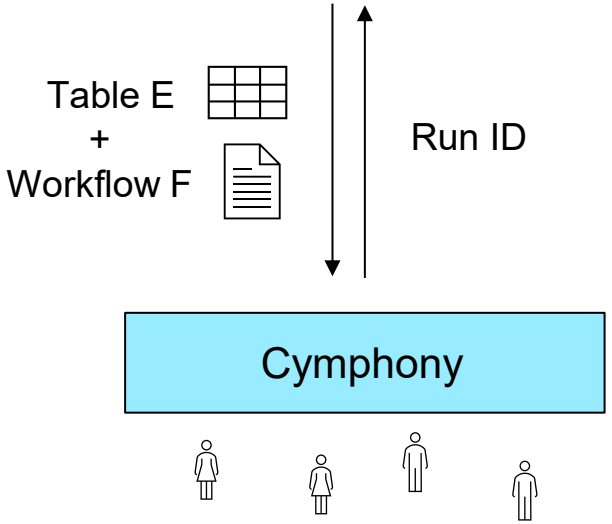
- **Two Smartcat-Driven Schema Extensions**
- **Add Column id_field_name to the Table workflow_input_files:**
  - Smartcat uses a logical id (e.g., column/asset id).
  - Cymphony uses internal task_id.
  - id_field_name enables stable mapping from Smartcat tuple id → CN task id when drive-by curations come in.
- **Add Column notification_url to the Table all_runs:**
  - Persist Smartcat webhook endpoint per run
  - CN can notify SC immediately when run terminates
  - Avoids external state / shared coordination services
  - Preserves loose coupling
- **Everything else:**
  - Projects/workflows/runs/jobs semantics unchanged
  - Operator implementations unchanged

Create Curation Run API

- **Input:**
  - CSV Input table T
  - Tuple identifier field (e.g., tid)
  - Workflow spec F (.cy)
  - Instruction HTML (+ optional layout HTML)
  - Optional Callback URL

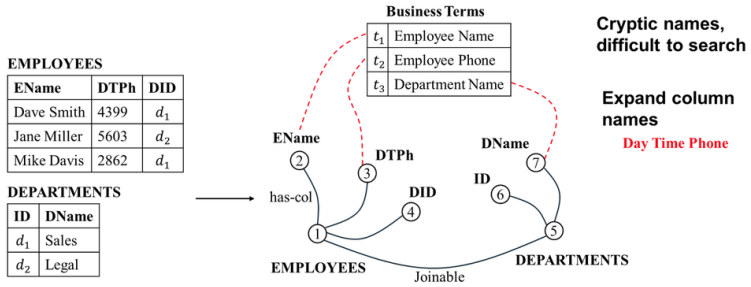
- **Step 1: Create Logical Containers**
  - all_projects(p_id, u_id, p_name, p_desc)
  - all_workflows(w_id, p_id, u_id, w_name, w_desc)



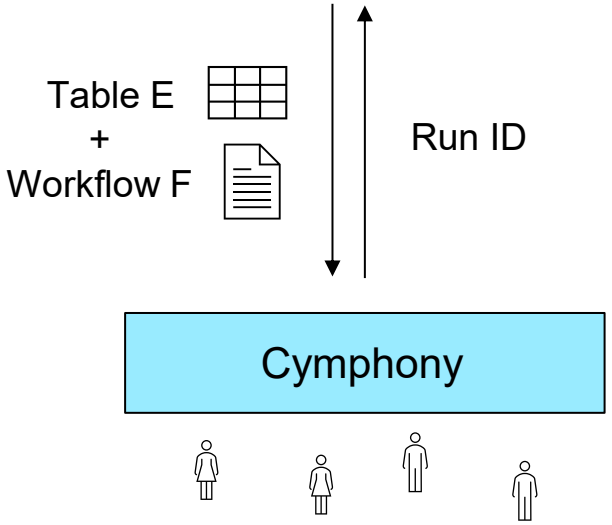


Create Curation Run API

- **Step 2: Register Workflow Artifacts (Ledger Tables)**
  - Artifact ledger tables:
    - workflow_input_files(..., id_field_name)
    - workflow_inst_files(...)
    - workflow_layout_files(...)
    - workflow_cy_files(...)
  - Files stored on disk:
    - $HOME/u<u_id>/p<p_id>/w<w_id>/

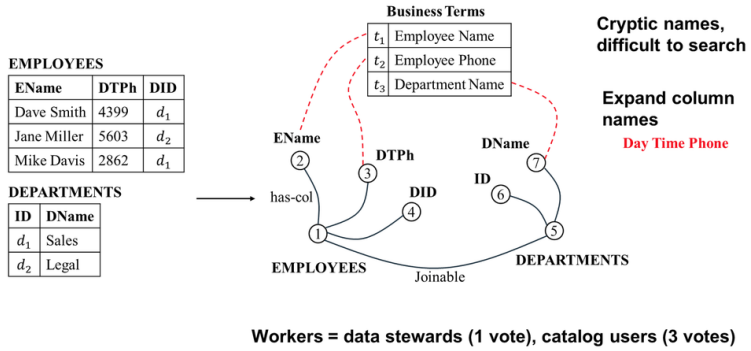


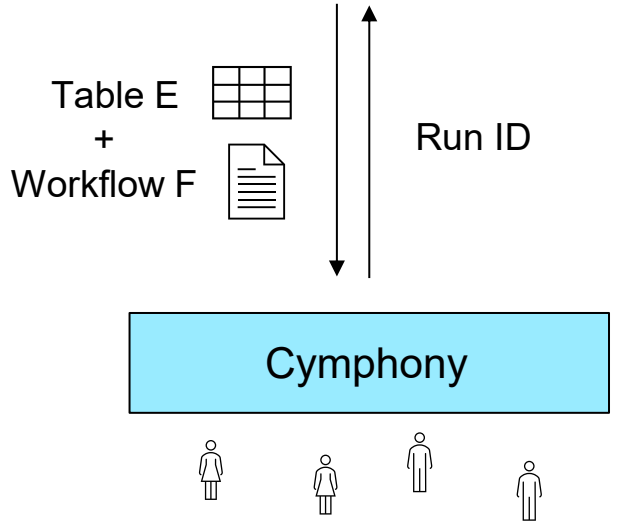
Workers = data stewards (1 vote), catalog users (3 votes)



Create Curation Run API

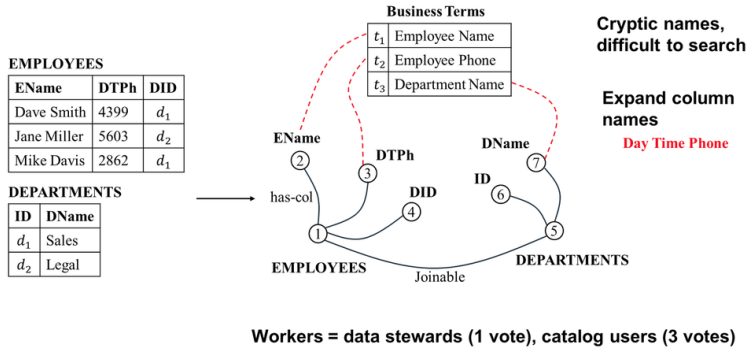
- **Step 2: Register Workflow Artifacts (Ledger Tables)**
  - Artifact ledger tables:
    - workflow_input_files(..., id_field_name)
    - workflow_inst_files(...)
    - workflow_layout_files(...)
    - workflow_cy_files(...)
  - Files stored on disk:
    - $HOME/u<u_id>/p<p_id>/w<w_id>/
- **Step 3: Instantiate Run + Isolate Run Directory**
  - CN creates run record in
    - all_runs(..., r_status, notification_url)
  - CN creates run directory
    - $HOME/.../w<w_id>/r<r_id>/
  - Copies workflow artifacts into run directory to isolate execution state.



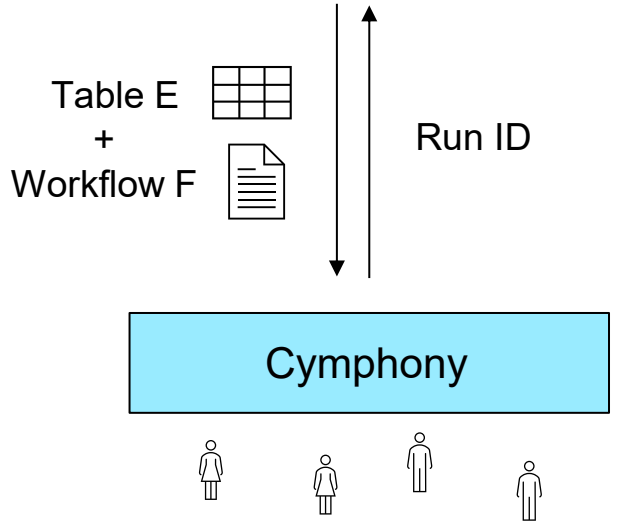


Create Curation Run API

- **Step 4: Parse .cy → DAG + Execution Order**
  - CN constructs in-memory DAG and persists:
    - ui_pj_wk_rm_nodes(n_id, name, type)
    - ui_pj_wk_rm_edges(src_id, dst_id)
    - ui_pj_wk_rm_nodes_execution_order(n_id, position)

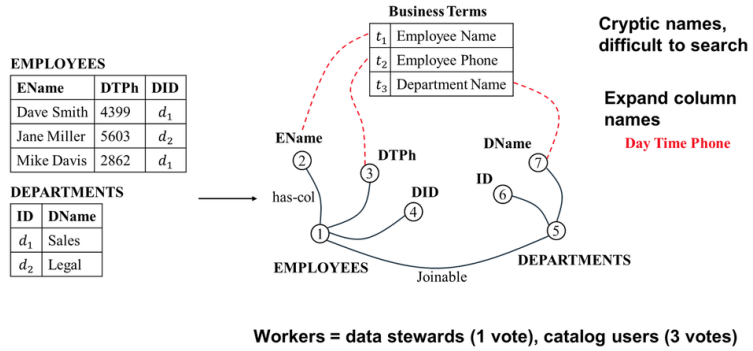


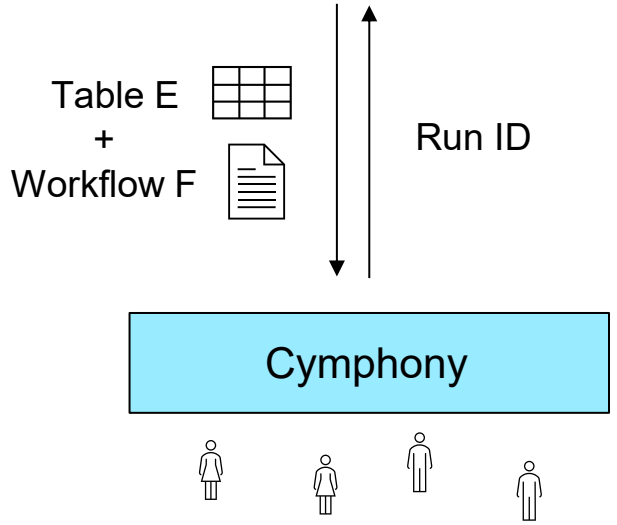
- **Step 5: Initialize Job Metadata for Each Operator**
  - For each operator node:
    - Create entry in all_jobs(..., j_type, j_status)
    - Automatic vs human
    - Initial status: IDLE
  - Maintain mapping:
    - ui_pj_wk_rm_mapping_operator_node_vs_job(n_id, j_id)



Create Curation Run API

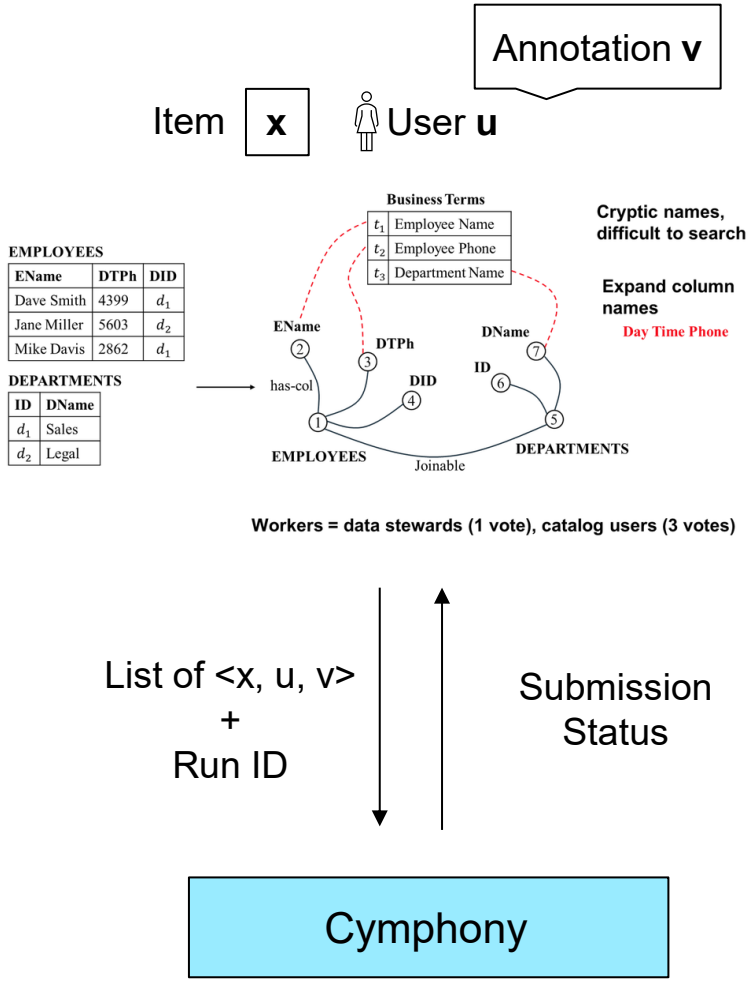
- **Step 6: Begin Execution**
  - CN executes DAG in topological order:
    - Runs automatic operators immediately
    - Halts at first human operator
    - Transitions that job to RUNNING
  - Returns:
    - Run id: user_id.project_id.workflow_id.run_id
    - Status code
    - Diagnostic message





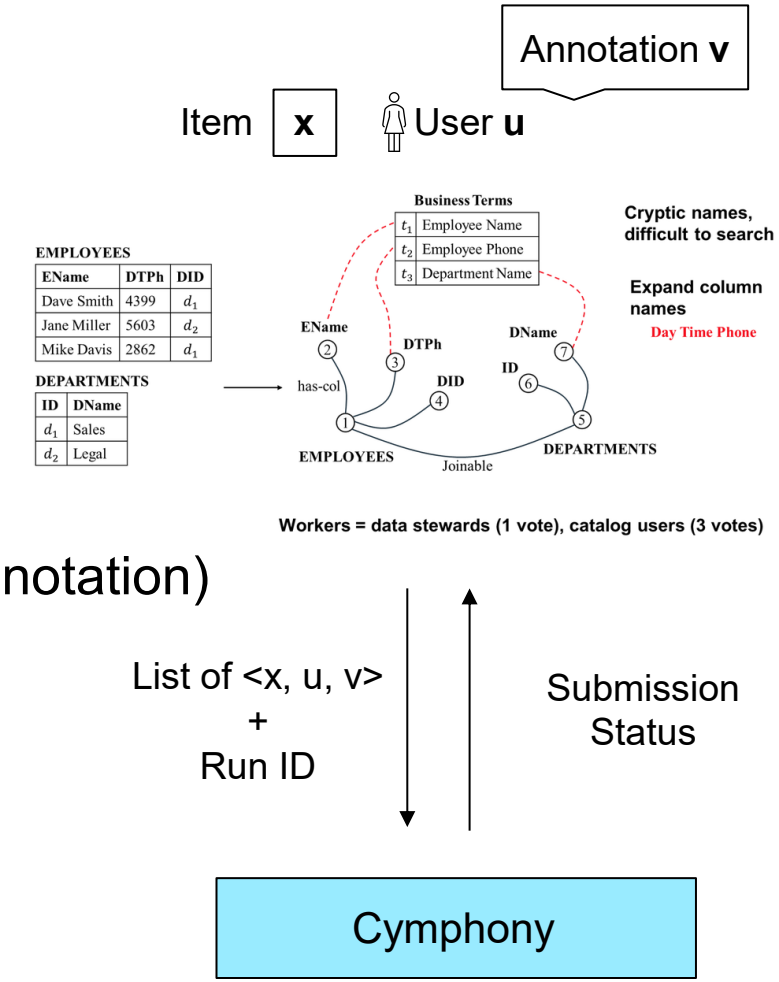
Drive-by Curation API

- **Input:**
  - Run identifier
  - Curations ⟨x,u,v⟩ (SC tuple id, SC user id, vote)
- **CN must:**
  - Locate human job for run
  - Map SC tuple id → CN task_id
  - Append into job outputs
  - Attempt aggregation immediately
- **Response:**
  - Status code
  - Diagnostic message



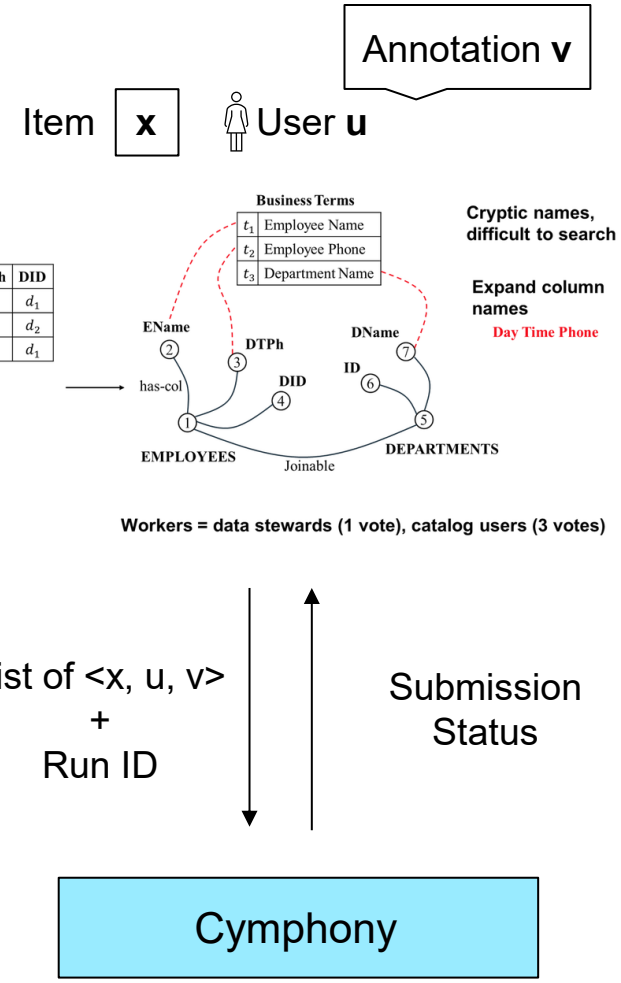
Drive-by Curation API

- **Step 1–2: Resolve Job + Resolve id_field_name**
  - CN parses composite run id → (u_id, p_id, w_id, r_id)
  - CN finds unique human job via:
    - all_jobs(...) filtering to 3a_kn / 3a_knlm
  - CN loads id_field_name from:
    - workflow_input_files(..., id_field_name)
- **Step 3–4: Auditability + Loose Coupling**
  - First write raw votes to:
    - ..._drive_by_curation_votes(<id_field_name>, worker_id, annotation)
  - Then rewrite SC user id u to:
    - a single CN “Smartcat drive-by” account id



Drive-by Curation API

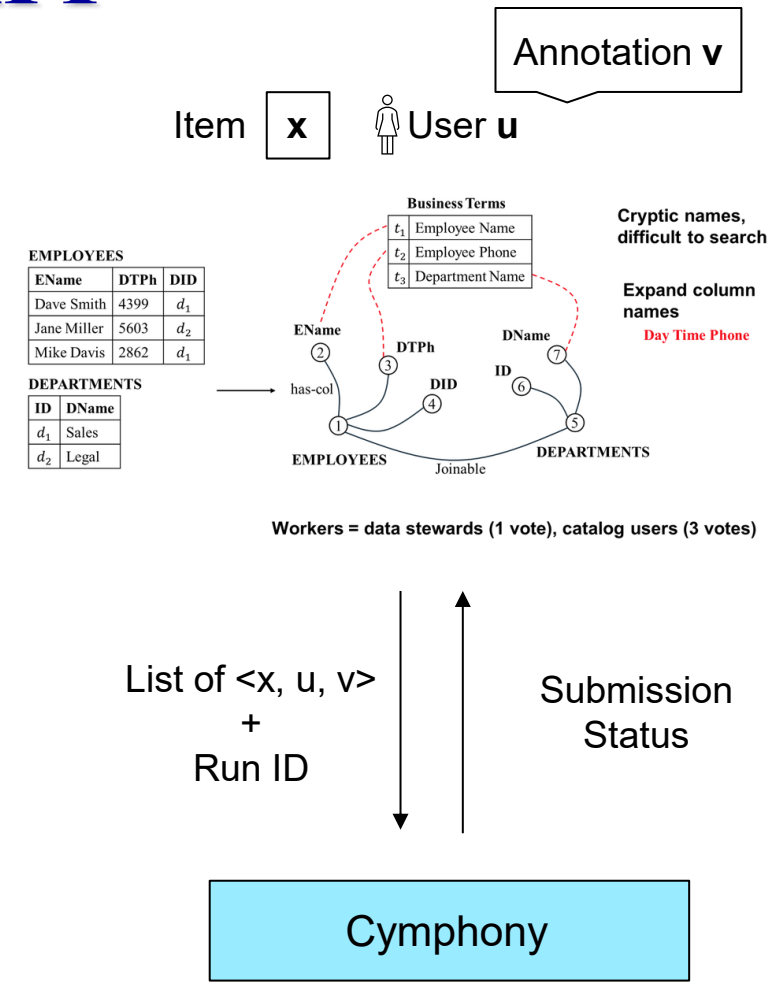
- **Step 5: Convert SC Tuple IDs to CN Task IDs**
  - Join against:
    - ..._tuples(task_id, ..., <id_field_name>, ...)
  - Produce:
    - ⟨task_id, rewritten_user, annotation⟩
  - Append to standard outputs:
    - ..._outputs(task_id, worker_id, annotation)
- **Step 6: Immediate Aggregation with Row Locks**
  - For each affected task_id:
    - If already in ..._final_labels, skip
    - Acquire row-level lock on ..._tasks(task_id, ...)
    - Check convergence using votes in ..._outputs
    - If converged, write ..._final_labels(task_id, label) and mark task as done
    - Release lock + update metadata



Drive-by Curation API

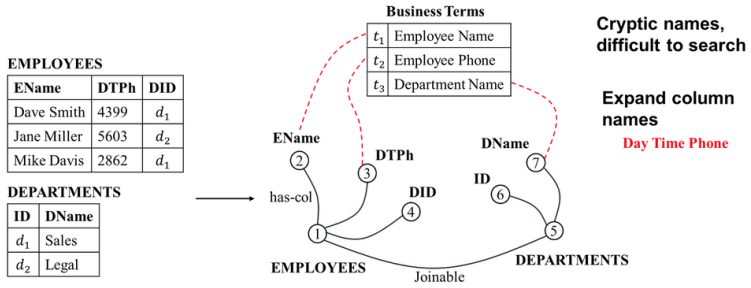
- **Step 6: Immediate Aggregation with Row Locks**
  - For each affected task_id:
    - If already in ..._final_labels, skip
    - Acquire row-level lock on ..._tasks(task_id, ...)
    - Check convergence using votes in ..._outputs
    - If converged, write ..._final_labels(task_id, label) and mark task as done
    - Release lock + update metadata

- **Returns:**
    - Status code
    - Diagnostic message



Run Status API

- **Input:**
  - Run identifier
- **CN:**
  - Retrieves run status from
    - all_runs(..., r_status, notification_url)
  - Possible states
    - IDLE
    - RUNNING
    - COMPLETED
    - ABORTED
- **Response:**
  - Run Status



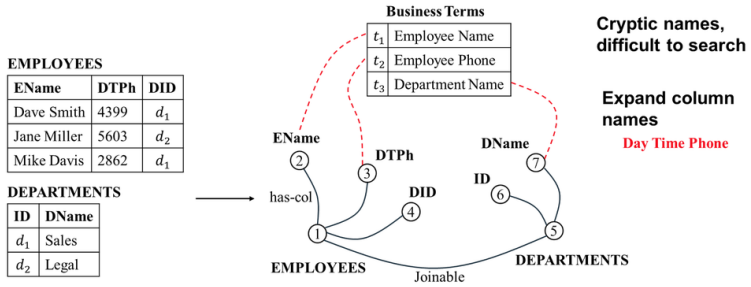
Workers = data stewards (1 vote), catalog users (3 votes)

Run ID ↓ ↑ Run Status

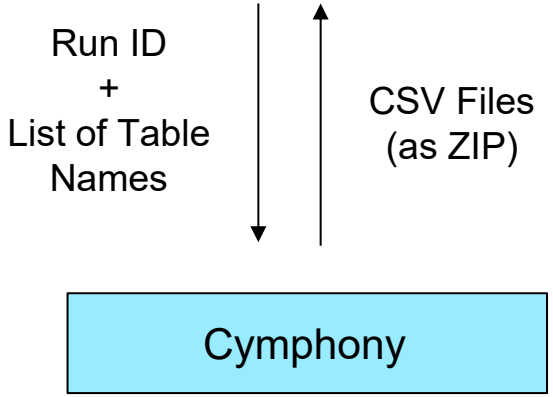
Cymphony

Download Tables API

- **Input:**
  - Run identifier
  - List of logical table names
    - Assignments
    - Annotations
    - Aggregations
- **CN:**
  - Locates run directory via all_runs
  - Locates human job via all_jobs
  - Resolves job-scoped relations by namespace
  - Joins with ..._tuples to restore original input fields
  - Exports CSVs + bundles into ZIP
- **Response:**
  - ZIP archive

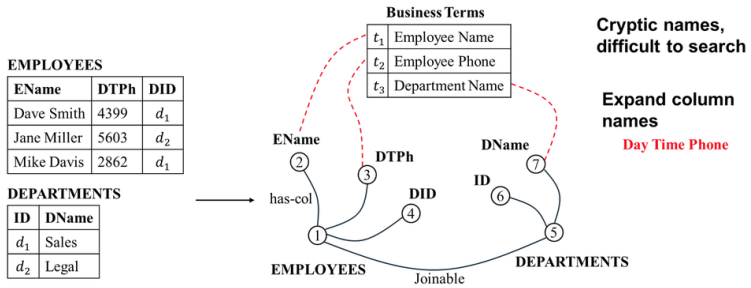


Workers = data stewards (1 vote), catalog users (3 votes)



Run Completion Webhook Notification

- **At run creation:**
  - SC supplies webhook endpoint
- **Upon run completion:**
  - Update all_runs → COMPLETED
  - Read notification_url
  - CN invokes callback and sends payload:
    - Run id
    - Terminal status
    - Completion timestamp
  - Failures logged but do not change terminal state.
- **Benefits:**
  - No polling delay
  - Immediate catalog update



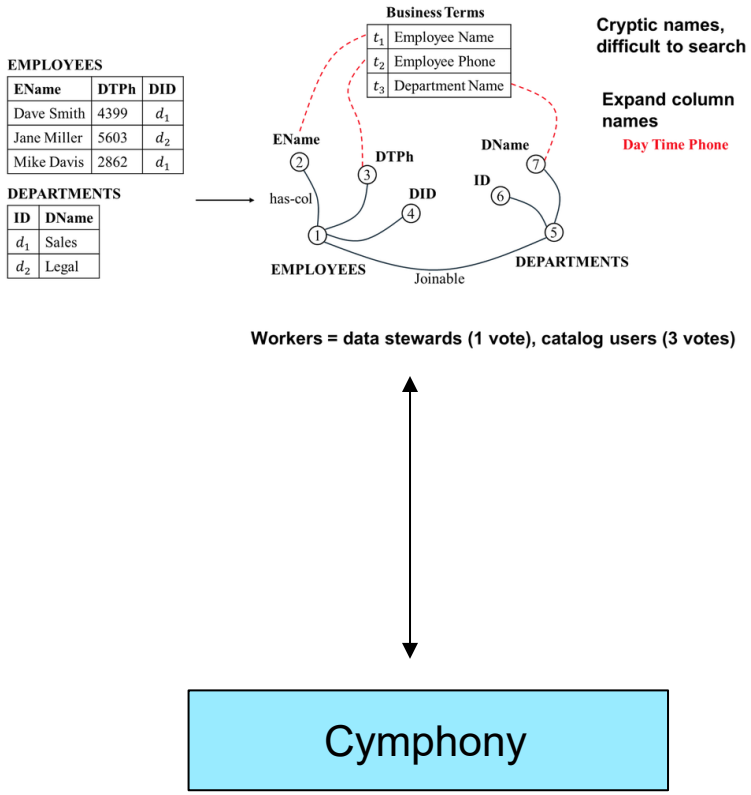
Workers = data stewards (1 vote), catalog users (3 votes)

Run ID
+
Run Status
+
Completion Timestamp

Cymphony

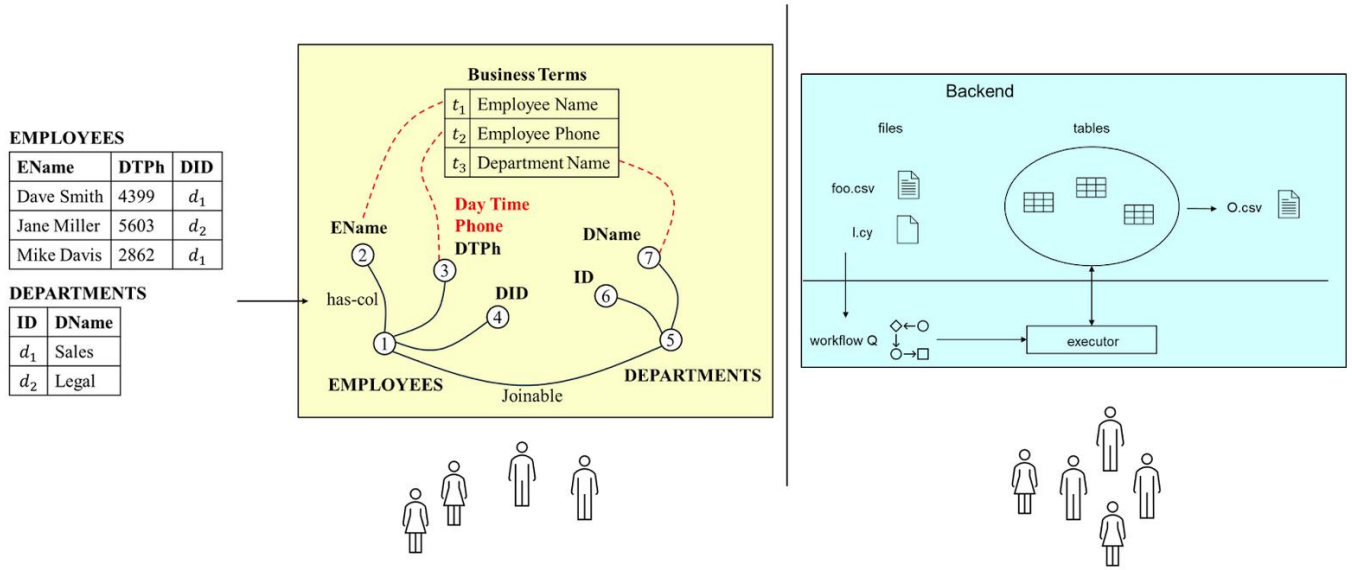
Implementation Takeaways

- **What This Implementation Demonstrates**
  - Minimal schema evolution
  - Drive-by votes become first-class annotations
  - Concurrency safety via task-level row locks
  - Smartcat integration uses only public APIs
  - Execution semantics stay unchanged



Realizing Our Proposed Solution

- **Support multiple worker roles**
  - Customize Cymphony's user management model
- **Support SC-style curation workflows**
  - Customize Cymphony's operators and program model
- **Support different forms of interaction required by SC**
  - API-driven integration of SC with Cymphony



Experiments

Experiment Plan

- **Consider 3 experiment classes**
  - Experiments 1–3: bulk curation baselines (synthetic workers/stewards)
  - Experiments 4–6: introduce drive-by + mixed interaction
  - Experiment 7: large-scale external worker population

- **Evaluate that Cymphony can serve as a curation backend for SC that**
  - Supports multiple worker types including stewards, organization users, and turkers
  - Supports multiple interaction modalities including drive-by, bulk, and mixed interaction patterns

Dataset

- **Environmental Data Initiative (EDI)**
- **Enriched using:**
  - Smartcat CNE pipeline
  - Maverick LLM
  - 3,830 candidate expansions
- **Dataset Attributes**
  - table_name
  - column_name
  - gt_label
  - expansion
  - gold_label (1.0 correct, 0.0 incorrect)
    - Used to evaluate annotation correctness.

```
id,table_name,column_name,gt_label,expansion,gold_label
61,1997lgextnuts.csv,CAN#,can #,Canister Number,1.0
62,1997lgextnuts.csv,CANWT,can weight,Canister Weight,1.0
63,1997lgextnuts.csv,WETWT,wet weight,Wet Weight,1.0
64,1997lgextnuts.csv,DRYWT,dry weight,Dry Weight,1.0
65,1997lgextnuts.csv,KCLWT,,Potassium Chloride Weight,
```

Dataset

- **Data Preprocessing**
  - Some records missing gold_label
    - Gold expansion unavailable
    - LLM output invalid
  - Filter rows with missing gold labels
- **Final dataset:**
  - 2,918 valid records

```
id,table_name,column_name,gt_label,expansion,gold_label
61,1997lgextnuts.csv,CAN#,can #,Canister Number,1.0
62,1997lgextnuts.csv,CANWT,can weight,Canister Weight,1.0
63,1997lgextnuts.csv,WETWT,wet weight,Wet Weight,1.0
64,1997lgextnuts.csv,DRYWT,dry weight,Dry Weight,1.0
65,1997lgextnuts.csv,KCLWT,,Potassium Chloride Weight,
```

Experiments 1–3 Setup (Held Constant)

- **Dataset**
  - 2,918 records
  - filtered EDI expansions

id,table_name,column_name,gt_label,expansion,gold_label
61,1997lgextnuts.csv,CAN#,can #,Canister Number,1.0
62,1997lgextnuts.csv,CANWT,can weight,Canister Weight,1.0
63,1997lgextnuts.csv,WETWT,wet weight,Wet Weight,1.0
64,1997lgextnuts.csv,DRYWT,dry weight,Dry Weight,1.0

- **Task**
  - Is the LLM-generated expansion semantically correct?
- **Workflow**
  - Single human operator 3a_knlm
- **Bulk Curation**
- **Worker Characteristics**
  - Synthetic regular workers
  - Arrival: 1 worker / 2 min, up to 100
  - Speed: 1 task / 10 sec
  - Accuracy: uniform in [0.80, 0.85]

Experiment 1: Bulk Workers (2-out-of-3)

- **Workflow**
  - Single human operator 3a_knlm
  - Aggregation policy:
    - Regular workers: k/n = 2/3
    - Steward policy configured as 1/1 (although no stewards present in the experiment)

- **Results**
  - Active workers: 34
  - Per-worker annotations:
    - Range = 3 – 393
    - Avg = 198.2
  - Observed worker accuracy:
    - Range = 0.71 – 1.0
    - Avg = 0.8165

Experiment 1: Bulk Workers (2-out-of-3)

- **Results**
  - Active workers: 34
  - Per-worker annotations:
    - Range = 3 – 393
    - Avg = 198.2
  - Observed worker accuracy:
    - Range = 0.71 – 1.0
    - Avg = 0.8165
  - Task outcomes:
    - 2,918/2,918 tasks converge (0 undecided)
    - Output labels: 2,333 positive, 585 negative
  - Accuracy: 0.9146
  - Run completion time: 1:06:59

Experiment 2: Bulk Workers with Higher Redundancy (3-out-of-5)

- **Workflow**
  - Single human operator 3a_knlm
  - Aggregation policy:
    - Regular workers: k/n = 3/5
    - Steward policy configured as 1/1 (although no stewards present in the experiment)

- **Hypothesis: Redundancy**
  1. Increases accuracy, but
  2. Increases vote volume
  3. Increases completion time

Experiment 2: Bulk Workers with Higher Redundancy (3-out-of-5)

- **Results**
  - Active workers: 42
  - Per-worker annotations:
    - Range = 6 – 489
    - Avg = 247.8 (↑ vs Exp. 1, confirming hypothesis 2)
  - Observed worker accuracy:
    - Range = 0.74 – 0.92
    - Avg = 0.8186
  - Task outcomes:
    - 2,918/2,918 converge (0 undecided)
    - Output labels: 2,422 positive, 496 negative
  - Accuracy: 0.9513 (↑ vs Exp. 1, confirming hypothesis 1)
  - Run completion time: 1:23:25 (↑ vs Exp. 1, confirming hypothesis 3)

Experiment 3: Bulk Workers and Stewards (2-out-of-3; 1-out-of-1)

- **Workflow**
  - Single human operator 3a_knlm
  - Aggregation policy:
    - Regular workers: k/n = 2/3
    - Steward policy configured as 1/1 (stewards present in this experiment)
- **Worker Characteristics**
  - Synthetic regular workers
    - Arrival: 1 worker / 2 min, up to 100
    - Speed: 1 task / 10 sec
    - Accuracy: uniform in [0.80, 0.85]
  - 2 stewards, accuracy ≈ 95%
    - Steward 1: active immediately for 1 hour
    - Steward 2: joins at 30 min, active for 2 hours

Experiment 3: Bulk Workers and Stewards (2-out-of-3; 1-out-of-1)

- **Hypothesis: Without increasing redundancy**
  1. Improve quality
  2. Reduce latency

- **Results**
  - Regular workers:
    - Active: 31
    - Per-worker annotations: 7 – 362 (avg 184.2)
    - Observed accuracy avg: 0.8301
  - Stewards:
    - Steward annotations: 355 and 183
    - Observed accuracy: 0.9352, 0.9398

Experiment 3: Bulk Workers and Stewards (2-out-of-3; 1-out-of-1)

- **Results**
  - Regular workers:
    - Active: 31
    - Per-worker annotations: 7 – 362 (avg 184.2)
    - Observed accuracy avg: 0.8301
  - Stewards:
    - Steward annotations: 355 and 183
    - Observed accuracy: 0.9352, 0.9398
  - Task outcomes:
    - 2,918/2,918 converge (0 undecided)
    - Output labels: 2,361 positive, 557 negative
  - Accuracy: 0.9263 (↑ vs Exp. 1, confirming hypothesis 1)
  - Run completion time: 1:01:38 (↓ vs Exp. 1, confirming hypothesis 2)

Experiment Plan

- **Consider 3 experiment classes**
  - Experiments 1–3: bulk curation baselines (synthetic workers/stewards)
  - Experiments 4–6: introduce drive-by + mixed interaction 
  - Experiment 7: large-scale external worker population

- **Evaluate that Cymphony can serve as a curation backend for SC that**
  - Supports multiple worker types including stewards, organization users, and turkers
  - Supports multiple interaction modalities including drive-by, bulk, and mixed interaction patterns

Experiments 4–6 Setup (Held Constant)

- **Dataset**
  - 2,918 records
  - filtered EDI expansions

```
id,table_name,column_name,gt_label,expansion,gold_label
61,1997lgextnuts.csv,CAN#,can #,Canister Number,1.0
62,1997lgextnuts.csv,CANWT,can weight,Canister Weight,1.0
63,1997lgextnuts.csv,WETWT,wet weight,Wet Weight,1.0
64,1997lgextnuts.csv,DRYWT,dry weight,Dry Weight,1.0
```

- **Task**
  - Is the LLM-generated expansion semantically correct?

- **Workflow**
  - Single human operator 3a_knlm
  - Aggregation policy:
    - Regular workers: k/n = 2/3
    - Steward policy configured as 1/1

Experiments 4–6 Setup (Held Constant)

- **Worker Interaction Pattern: Beyond Bulk Curation**
  - Exp 4: Pure drive-by (only SC drive-by API submissions)
  - Exp 5: Drive-by + bulk regular workers
  - Exp 6: Drive-by + bulk regular + high-trust stewards

- **Worker Characteristics**
  - Drive-by simulation (Smartcat-side):
    - 50 synthetic SC users, 1 user / 2 min, 1 task / 10 sec
    - Each arrives once, labels 10 tuples, never returns
    - Batch submissions to Drive-by Curation API
    - Per-user accuracy sampled in [0.80, 0.85]

Experiment 4: Pure Drive-by (2-out-of-3)

- **Annotation Source**
  - Smartcat drive-by

- **Worker Characteristics**
  - Drive-by simulation (Smartcat-side):
    - 50 synthetic SC users, 1 user / 2 min, 1 task / 10 sec
    - Each arrives once, labels 10 tuples, never returns
    - Batch submissions to Drive-by Curation API
    - Per-user accuracy sampled in [0.80, 0.85]

- **Results**
  - Total drive-by annotations: 500
  - Distinct tuples touched: 467

Experiment 4: Pure Drive-by (2-out-of-3)

- **Results**
  - Total drive-by annotations: 500
  - Distinct tuples touched: 467
  - Tuples converged: 20 / 2,918
  - Convergence limited by insufficient redundancy, not conflicts
  - Accuracy on converged subset: 1.0000*
  - Completion time: 1:40:15
- **Discussion**
  - Sparse, uneven coverage across tuples
  - Results retrievable even when most tuples stay unresolved
- **Conclusion**
  - Integration remains stable with partial convergence

Footnote: * Accuracy computed only on tuples that received enough votes to converge.

Experiment 5: Mixed Drive-by + Bulk (2-out-of-3)

- **Annotation Source**
  - Smartcat drive-by
  - Bulk curation by regular workers on CN
- **Worker Characteristics**
  - Drive-by simulation from Smartcat-side (same as Exp. 4)
    - 50 synthetic SC users, 1 user / 2 min, 1 task / 10 sec
    - Each arrives once, labels 10 tuples, never returns
    - Batch submissions to Drive-by Curation API
    - Per-user accuracy sampled in [0.80, 0.85]
  - Synthetic regular workers (same as Exp. 3)
    - Arrival: 1 worker / 2 min, up to 100
    - Speed: 1 task / 10 sec
    - Accuracy: uniform in [0.80, 0.85]

Experiment 5: Mixed Drive-by + Bulk (2-out-of-3)

- **Hypothesis**
  1. Bulk guarantees full coverage
  2. No quality degradation vs bulk-only baseline of Exp. 1
  3. Drive-by helps early progress
- **Results**
  - Drive-by:
    - Before completion: 33 users, 330 annotations, 309 tuples
    - Full simulation: 50 users, 500 annotations
  - Bulk:
    - 33 regular workers
    - Avg accuracy ≈ 0.82
  - 2,918/2,918 tasks converge with 0 undecided (confirming hypothesis 1)
  - Accuracy: 0.9242 (≈ vs Exp. 1, confirming hypothesis 2)
  - Run completion time: 1:06:03 (slightly ↓ vs Exp. 1, confirming hypothesis 3)

Experiment 5: Mixed Drive-by + Bulk (2-out-of-3)

- **Discussion**
  - Drive-by annotations are:
    - Ingested via API
    - Written into same job-scoped outputs table as bulk annotations
    - Aggregation treats both modes uniformly
  - Confirms a key design decision:
    - Unify at the job-scoped output layer
    - Keep interaction modality at the API boundary

- **Conclusion**
  - Drive-by + bulk interaction modalities coexist cleanly

Experiment 6: Mixed Drive-by, Bulk, and Steward (2-out-of-3; 1-out-of-1)

- **Annotation Source**
  - Smartcat drive-by
  - Bulk curation by regular workers and stewards on CN
- **Worker Characteristics**
  - Drive-by simulation from Smartcat-side (same as Exp. 4)
    - 50 synthetic SC users, 1 user / 2 min, 1 task / 10 sec
    - Each arrives once, labels 10 tuples, never returns
    - Batch submissions to Drive-by Curation API
    - Per-user accuracy sampled in [0.80, 0.85]
  - Synthetic regular workers (same as Exp. 3)
    - Arrival: 1 worker / 2 min, up to 100
    - Speed: 1 task / 10 sec
    - Accuracy: uniform in [0.80, 0.85]

Experiment 6: Mixed Drive-by, Bulk, and Steward (2-out-of-3; 1-out-of-1)

- **Worker Characteristics**
  - Drive-by simulation from Smartcat-side (same as Exp. 4)
    - 50 synthetic SC users, 1 user / 2 min, 1 task / 10 sec
    - Each arrives once, labels 10 tuples, never returns
    - Batch submissions to Drive-by Curation API
    - Per-user accuracy sampled in [0.80, 0.85]
  - Synthetic regular workers (same as Exp. 3)
    - Arrival: 1 worker / 2 min, up to 100
    - Speed: 1 task / 10 sec
    - Accuracy: uniform in [0.80, 0.85]
  - 2 stewards, accuracy ≈ 95% (same as Exp. 3)
    - Steward 1: active immediately for 1 hour
    - Steward 2: joins at 30 min, active for 2 hours

Experiment 6: Mixed Drive-by, Bulk, and Steward (2-out-of-3; 1-out-of-1)

- **Hypothesis**
  1. Stewards resolve tuples early
  2. Faster convergence without increasing redundancy
  3. Maintains high accuracy
- **Results**
  - Drive-by: 50 users
  - Bulk:
    - 31 regular workers
    - 2 stewards
  - Tuples converged: 2,918 / 2,918 (0 undecided)
  - Accuracy: 0.9208 (≈ vs Exp. 5, confirming hypothesis 3)
  - Run completion time: 1:00:54 (↓ vs Exp. 5, confirming hypothesis 1 and 2)

Experiment 7: Large-Scale Turker Deployment

- **Prior Experiments:**
  - Synthetic workers with controlled simulation

- **Evaluate:**
  - Real-world worker behavior
  - Cost
  - Convergence behavior
  - Label quality

- **Dataset Preparation:**
  - Start: 2,918 labeled EDI records
    - 2,508 positive
    - 410 negative

Experiment 7: Large-Scale Turker Deployment

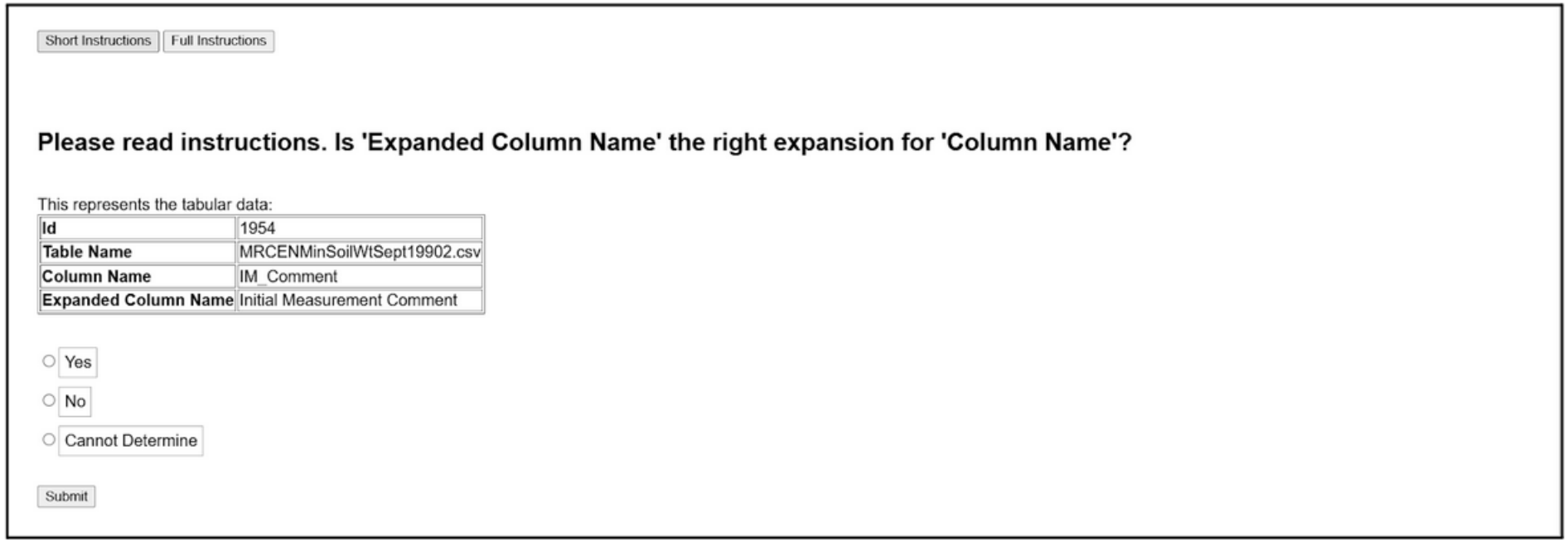
- **Dataset Preparation**
  - Start: 2,918 labeled EDI records
    - 2,508 positive
    - 410 negative
  - Construct evaluation dataset: 1,100 records
    - Step 1 — Tightly Representative (100 records):
    - One tuple per unique table name
    - Balanced to ~50–50 positive/negative
    - Step 2 — Loosely Representative (1,000 records):
    - Random shuffle
    - Preserve natural skew
  - Final Composition: 1,100 records
    - 915 positive
    - 185 negative

Experiment 7: Large-Scale Turker Deployment

- **Task Design**
  - Gold labels removed before submission
  - Worker sees:
    - Column name
    - Proposed expansion
  - Question:
    - “Is this expansion correct?”
  - Response options:
    - Yes
    - No
    - Cannot Determine

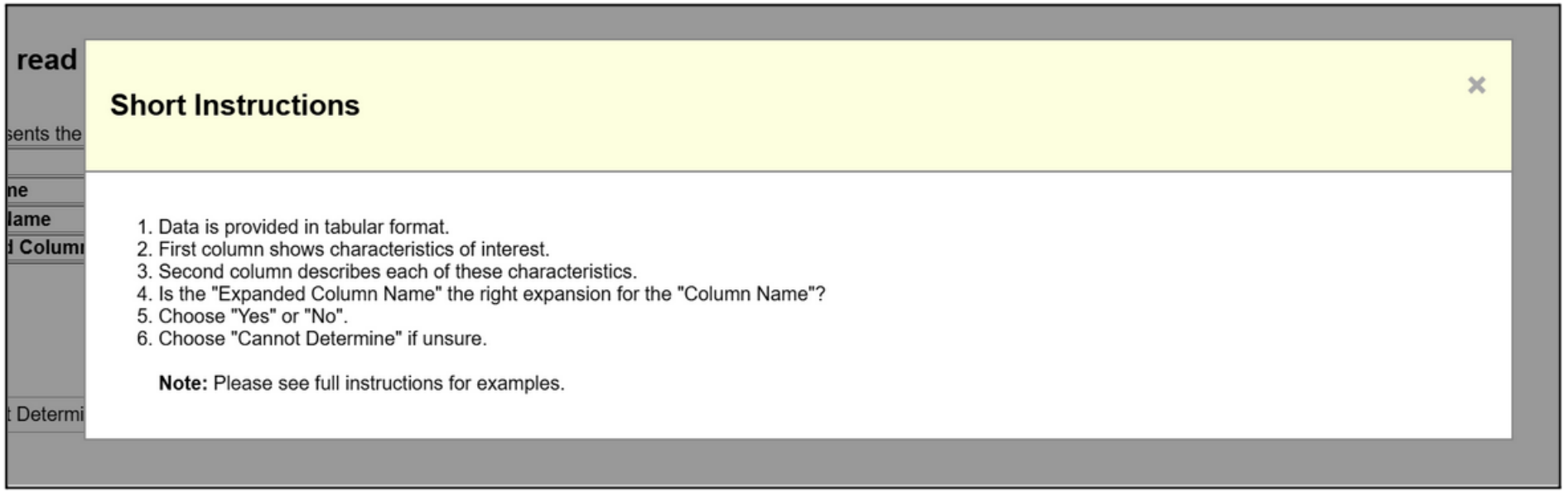
Experiment 7: Large-Scale Turker Deployment

- **Interface**
  - Design Goal
    - Minimize ambiguity
    - Allow explicit uncertainty
  - Tabular Layout



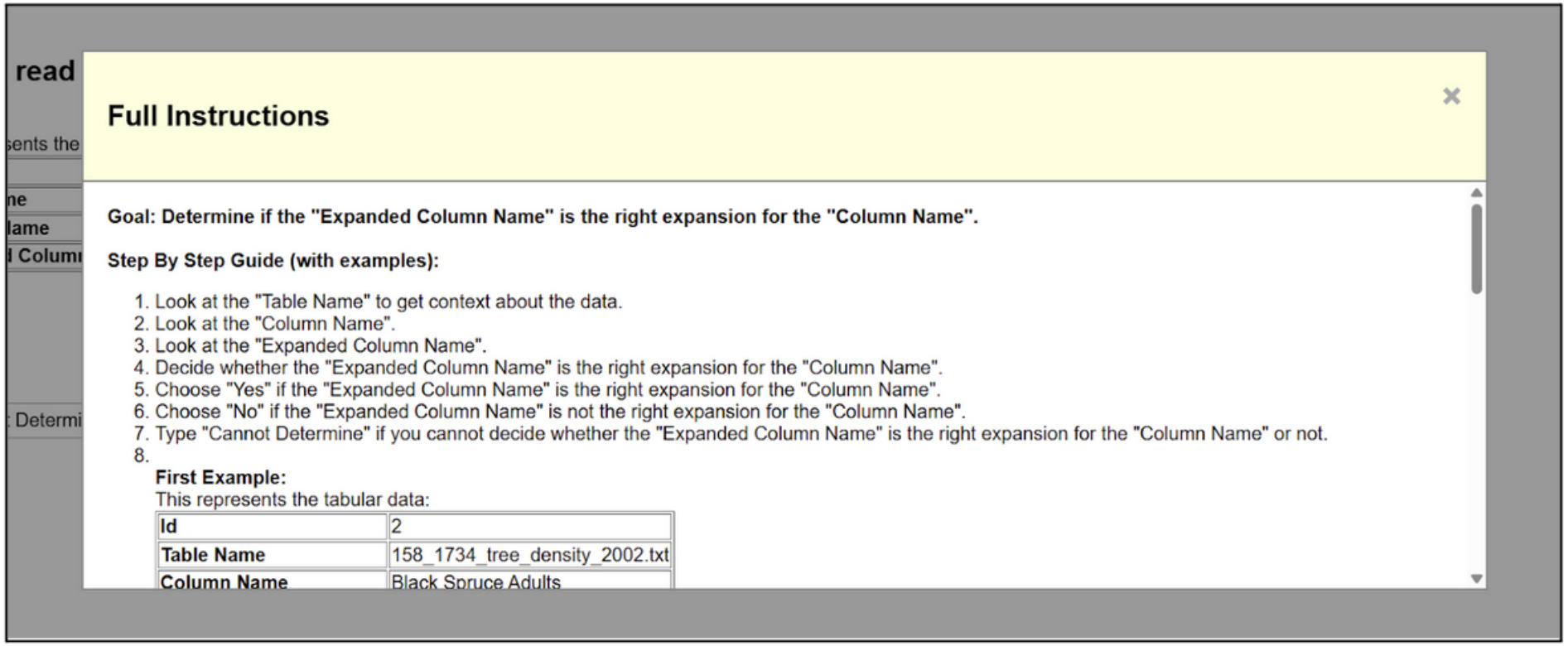
Experiment 7: Large-Scale Turker Deployment

- **Interface**
  - Short Instructions



Experiment 7: Large-Scale Turker Deployment

- **Interface**
  - Full Instructions



Step By Step Guide (with examples):

1. Look at the "Table Name" to get context about the data.
2. Look at the "Column Name".
3. Look at the "Expanded Column Name".
4. Decide whether the "Expanded Column Name" is the right expansion for the "Column Name".
5. Choose "Yes" if the "Expanded Column Name" is the right expansion for the "Column Name".
6. Choose "No" if the "Expanded Column Name" is not the right expansion for the "Column Name".
7. Type "Cannot Determine" if you cannot decide whether the "Expanded Column Name" is the right expansion for the "Column Name" or not.
8.

First Example:
This represents the tabular data:

Id	2
Table Name	158_1734_tree_density_2002.txt
Column Name	Black Spruce Adults
Expanded Column Name	Black Spruce Adults

- From the table name, we can infer that the data seems to be about tree density in 2002.
- From the column name, we can infer that the column does not have any abbreviation, and seems to be about Black Spruce Adults.
- From the expanded column name, we can infer that the expanded column name is the same as the column name.
- **Since the column name did not have any abbreviation, and the expanded column name is the same as the column name, we conclude that the expanded column name is the right expansion for the column name.**
- Hence, we will choose "Yes".

Second Example:
This represents the tabular data:

Id	42
Table Name	1988gssabm.csv
Column Name	Average g/m^2
Expanded Column Name	Average Grams per Square Meter

- From the table name, we cannot infer anything since it is cryptic.
- From the column name, we can see some abbreviations, and it seems to be saying Average gram/meter^2.
- From the expanded column name, we can infer that the expanded column name is "Average Grams per Square Meter" which seems to be the correct expansion for the column name.
- **Since the expanded column name seems to be the right expansion for the column name, we will choose "Yes".**

Third Example:
This represents the tabular data:

Id	404
Table Name	410_LogDecompDynamicsIntAK3_DiskData.txt
Column Name	LD0
Expanded Column Name	LD0

- From the table name, we can infer that the data seems to be about log decomposition dynamics for disk data.
- From the column name, we can infer that the column name is LD0 which seems to be abbreviated, but doesn't tell us anything more.
- From the expanded column name, we can infer that the expanded column name is the same as the abbreviated column name. It did not expand the abbreviation correctly.
- **Since the expanded column name does not seem to expand the column name correctly, we will choose "No".**

Fourth Example:
This represents the tabular data:

Id	3643
Table Name	occurrences.csv
Column Name	verbatimEventDate
Expanded Column Name	Unprocessed Event Date

- From the table name, we can infer that the data seems to be about occurrences.
- From the column name, we can infer that the column name is verbatimEventDate, and that it does not have any abbreviation.
- From the expanded column name, we can infer that the expanded column name is Unprocessed Event Date.
- Since the column name was not abbreviated, its expansion should have been "verbatimEventDate" or "verbatim event date", and not "Unprocessed Event Date".
- **Since the expanded column name does not seem to expand the column name correctly, we will choose "No".**

Experiment 7: Large-Scale Turker Deployment

- **Workflow**
  - Single human operator: 3a_amt
  - Aggregation policy: 2-out-of-3
  - Each HIT:
    - 1 task
    - $0.03 per annotation
  - Restrictions:
    - US-based
    - Master workers
  - All logic handled by:
    - Cymphony's AMT Manager

Experiment 7: Large-Scale Turker Deployment

- **Cost and Time**
  - Workers:
    - 15 Turkers
  - Votes per worker:
    - Range: 2 – 759
    - Avg: 220 votes
  - Cost: $132
  - Run completion time: 6h 7m
- **Accuracy**
  - Avg worker accuracy: 0.8687 (excluding “Cannot Determine”)
  - Tasks with decisive labels: 1,074
    - Undecided: 22
    - Resolved as “Cannot Determine”: 4

Experiment 7: Large-Scale Turker Deployment

- **Asymmetry Analysis**
  - When turkers say “Yes”, they are saying yes to:
    - 98.58% of positive examples

column_name	expansion
DO%	Dissolved Oxygen Percentage

    - 70.81% of negative examples (misclassify)

column_name	expansion
Len_PSMY_cm	Length PSMY Centimeters

  - When turkers say “No”/ ”Cannot Determine”/ ”Undecided”, they are saying so to:
    - 29.19% of negative examples (capture ≈ 30% negatives)

column_name	expansion
bodyclean_TransportMode	Transport Mode

    - 1.42% of positive examples (misclassify)

column_name	expansion
HOBO_Depth	HOBO Depth

- **Discussion**
  - Negative cases are inherently more ambiguous
  - “Yes” votes are highly accurate for positives, but also misclassify a bunch of negative cases
  - Non-affirmative votes act as a trustworthy filter for negative cases

Summary

- **Cymphony can serve as a curation backend for SC**
  - Supports multiple worker types including stewards, organization users, and turkers
  - Supports multiple interaction modalities including drive-by, bulk, and mixed interaction patterns

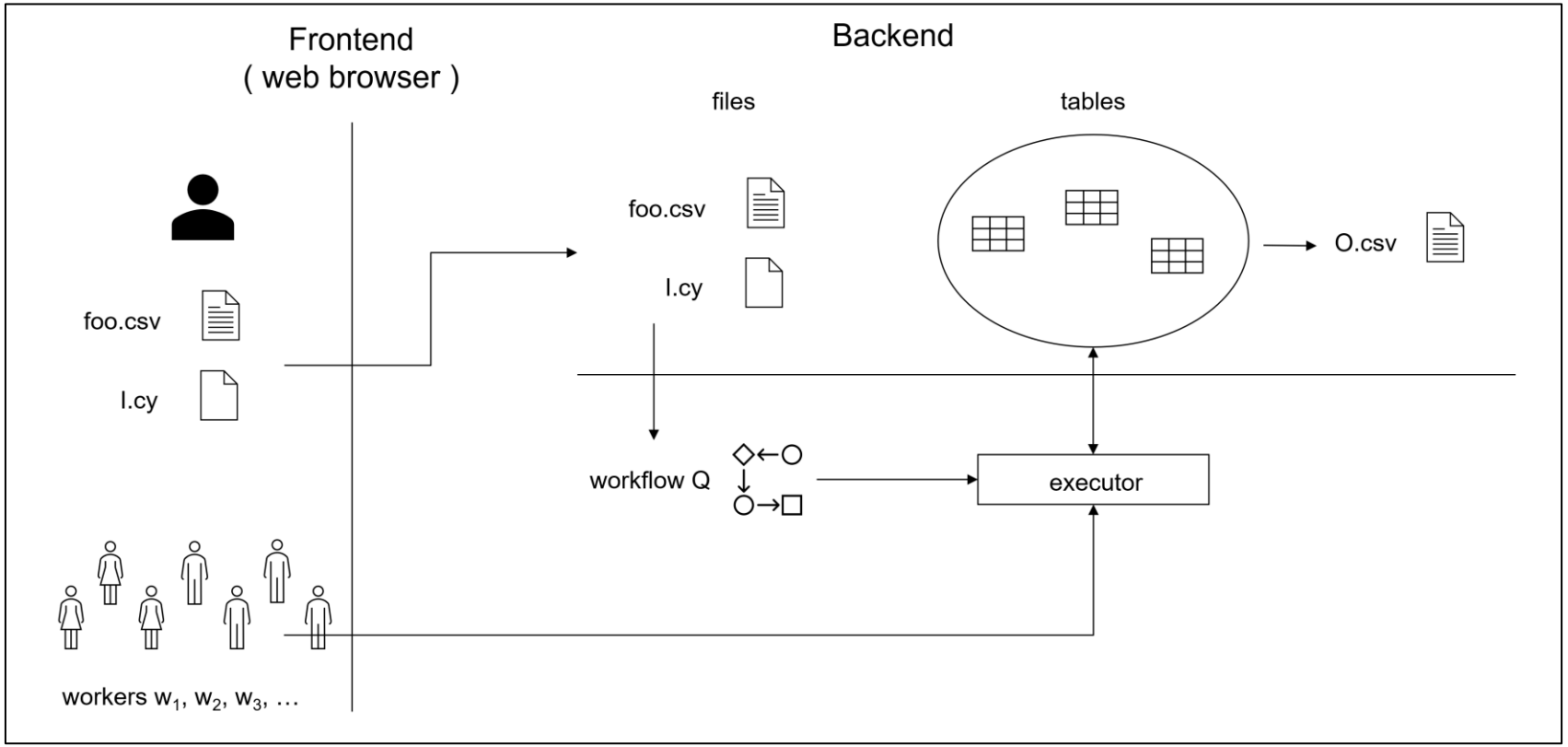
- **Experiments with real data and realistic catalog-style scenarios**
  - Show that Cymphony workflows can reach high accuracy
  - While supporting the curation requirements of SC

My Dissertation

- **Introducing the problem of building a CS platform for DI**
- **Designing Cymphony**
  - Define problem, operators, workflows
- **Implementation**
- **Applying Cymphony to DI problems & evaluation**
  - Usability, accuracy, scaling
- **Customizing Cymphony for Data Catalog Systems**
  - User registration, operators, program, API
- **Releasing Cymphony** 
  - Stand-alone, as a system component

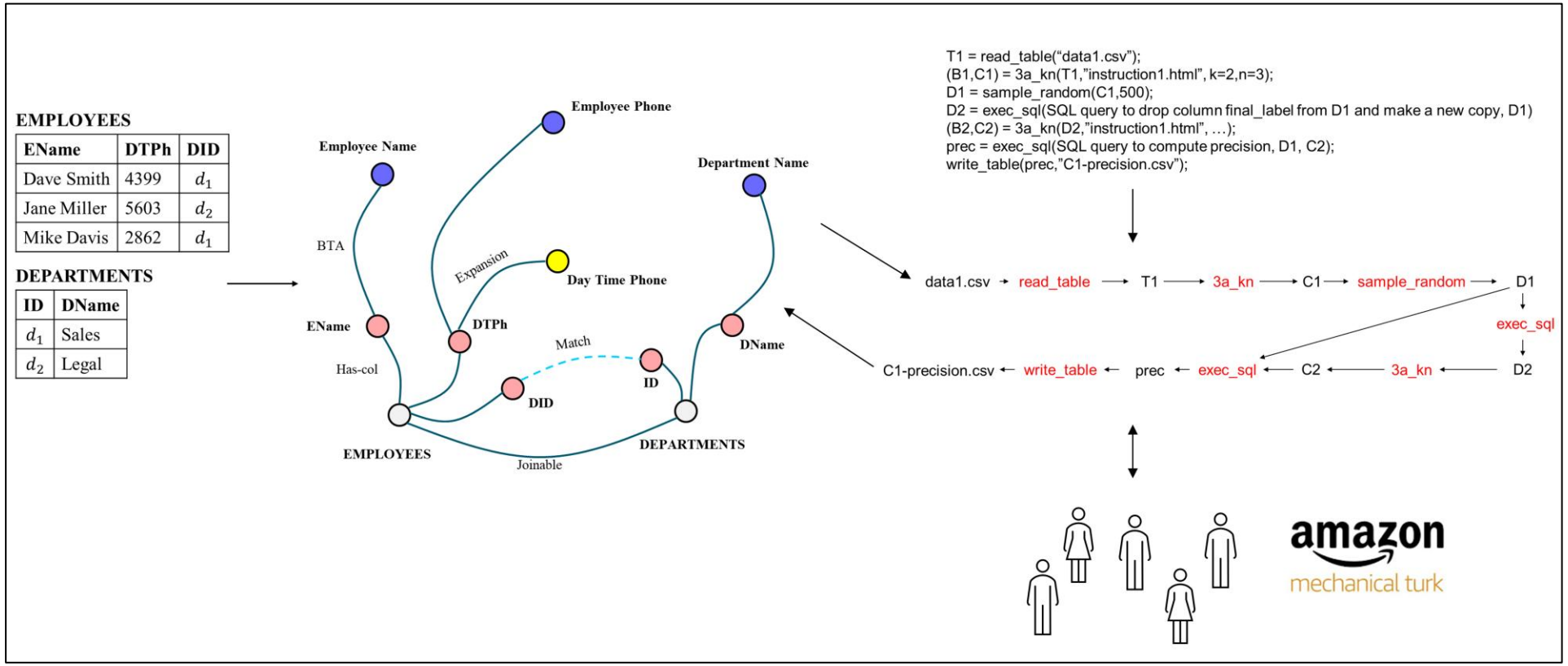
Releasing Cymphony: Stand-Alone

- **Released as open-source platform**
  - Public repo: github.com/anhaidgroup/cymphony
  - Deployed in production at QCRI since April 2022



Releasing Cymphony: As a System Component (Smartcat Integration)

- **Integrated as backend execution service for Smartcat**
  - Customized Cymphony for Smartcat
  - Public repo: https://github.com/saini5/cymphony4cs_customized



Conclusions

- **Crowdsourcing is ubiquitous in data integration and data science**
  - Yet current solutions remain fragmented and ad hoc

- **We have made significant contributions toward systematizing human-in-the-loop DI**
  - Cymphony: a declarative, operator-based crowdsourcing platform
  - End-to-end execution model for human–machine workflows
  - Scalable architecture validated with real workers and simulations

- **Provided a reusable human-in-the-loop backend for larger systems**
  - Enabling multiple interaction modalities
  - Supporting heterogeneous worker populations
  - Foundation for future human–machine data systems research