Magellan: Toward Building Entity Matching Management Systems

Presented by: Pradap Konda

February 27, 2018

Entity Matching

Table A

Name	City	State
Dave Smith	Madison	WI
Joe Wilson	San Jose	CA
Dan Smith	Middleton	WI

Table B

Name	City	State
David D. Smith	Madison	WI
Daniel W. Smith	Middleton	WI

- **Lot of work in this area over the past few decades**
- **Mainly focus on developing algorithms**

Need More Effort on Building EM Systems

- **Truly critical to advance the field**

- **EM is engineering by nature**
- **Can't keep developing EM algorithms in vacuum**
  - Akin to continuing to develop join algorithms without rest of RDBMS
- **Must build systems to evaluate algorithms, integrate R&D efforts, make practical impacts**

- **As examples, RDBMSs and Big Data systems were critical to advancing their respective fields**

But what kind of systems we should build, and how?

Current Research / System Building Agenda for Entity Matching

- **Two fundamental steps: blocking and matching**

Table A

	Name	City	State
a_1	Dave Smith	Madison	WI
a_2	Joe Wilson	San Jose	CA
a_3	Dan Smith	Middleton	WI

Table B

	Name	City	State
b_1	David D. Smith	Madison	WI
b_2	Daniel W. Smith	Middleton	WI

block on state = state → (a_1, b_1), (a_1, b_2), (a_3, b_1), (a_3, b_2) → **match** → (a_1, b_1) +, (a_1, b_2) -, (a_3, b_1) -, (a_3, b_2) +

Current Research / System Building Agenda for Entity Matching

Table A

	Name	City	State
a_1	Dave Smith	Madison	WI
a_2	Joe Wilson	San Jose	CA
a_3	Dan Smith	Middleton	WI

Table B

	Name	City	State
b_1	David D. Smith	Madison	WI
b_2	Daniel W. Smith	Middleton	WI

block on state = state → (a_1, b_1), (a_1, b_2), (a_3, b_1), (a_3, b_2) → **match** → (a_1, b_1) +, (a_1, b_2) -, (a_3, b_1) -, (a_3, b_2) +

Research

Focus on these two steps
- Develop algorithms
- Maximize accuracy, minimize cost

Assume other steps are trivial

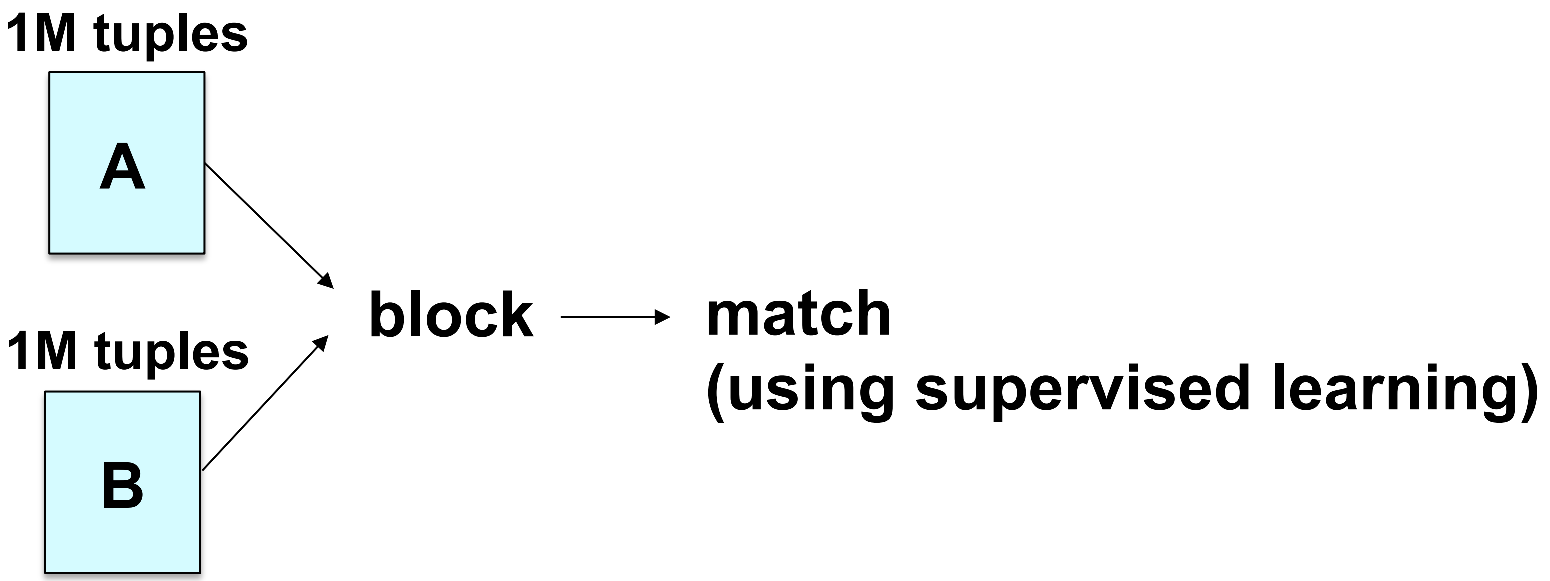
Build stand-alone monolithic systems (e.g., in Java)

Blocker 1	Matcher 1
Blocker 2	Matcher 2
…	…

This is Far from Enough for Handling EM in Practice

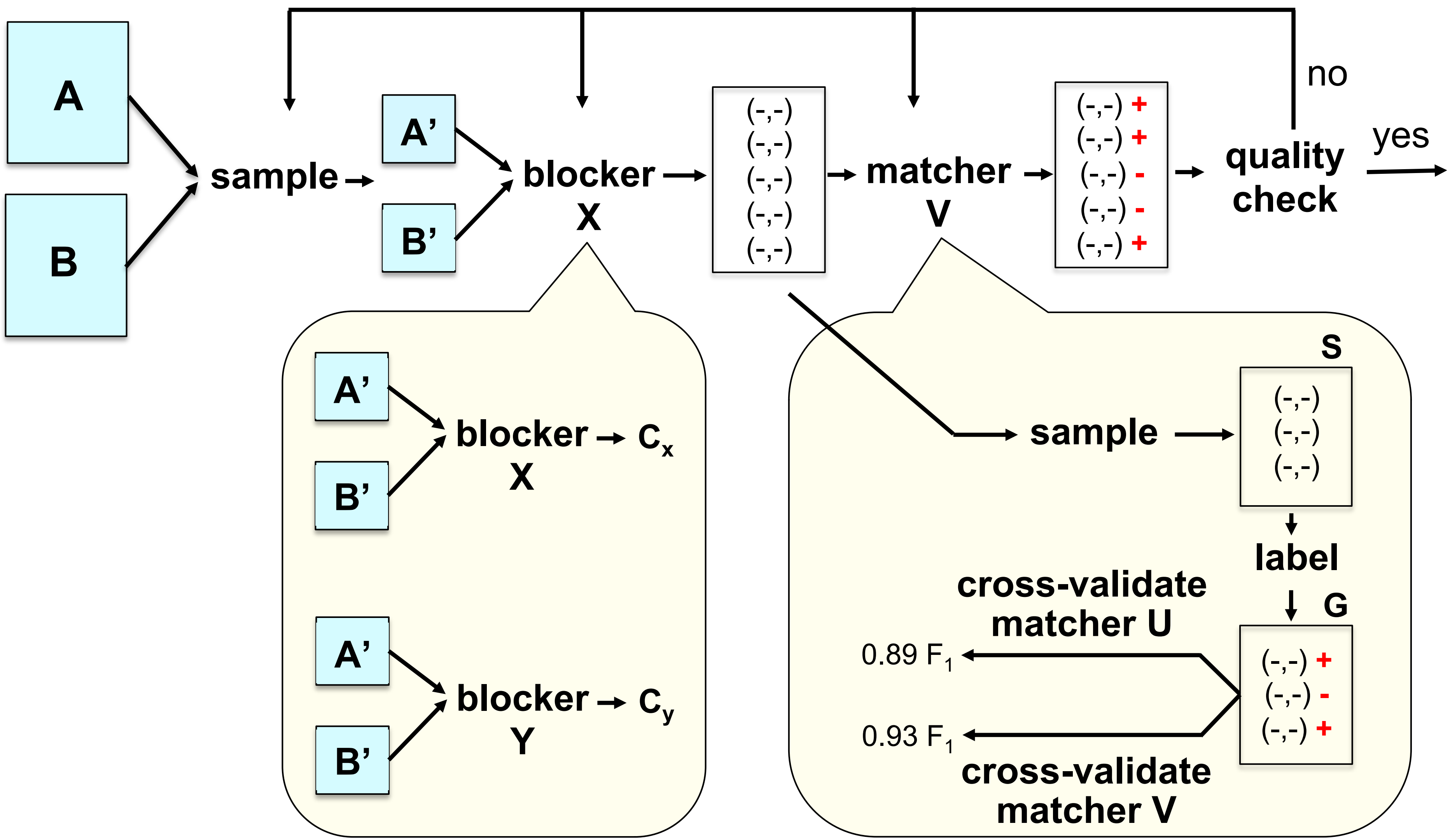
- **EM in practice is significantly more complex**
  - A messy, iterative, multiple-step process
  - Many steps perceived trivial are actually quite difficult to do
- **Even if we let a human user be in charge of the whole EM process, he/she often doesn't know what to do**
- **Will illustrate in the next few slides**
  - Using an example of applying supervised learning to do EM

How Is EM Done Today in Practice?



- **Development stage**
  - finds an accurate workflow, using data samples
- **Production stage**
  - executes workflow on entirety of data
  - focuses on scalability

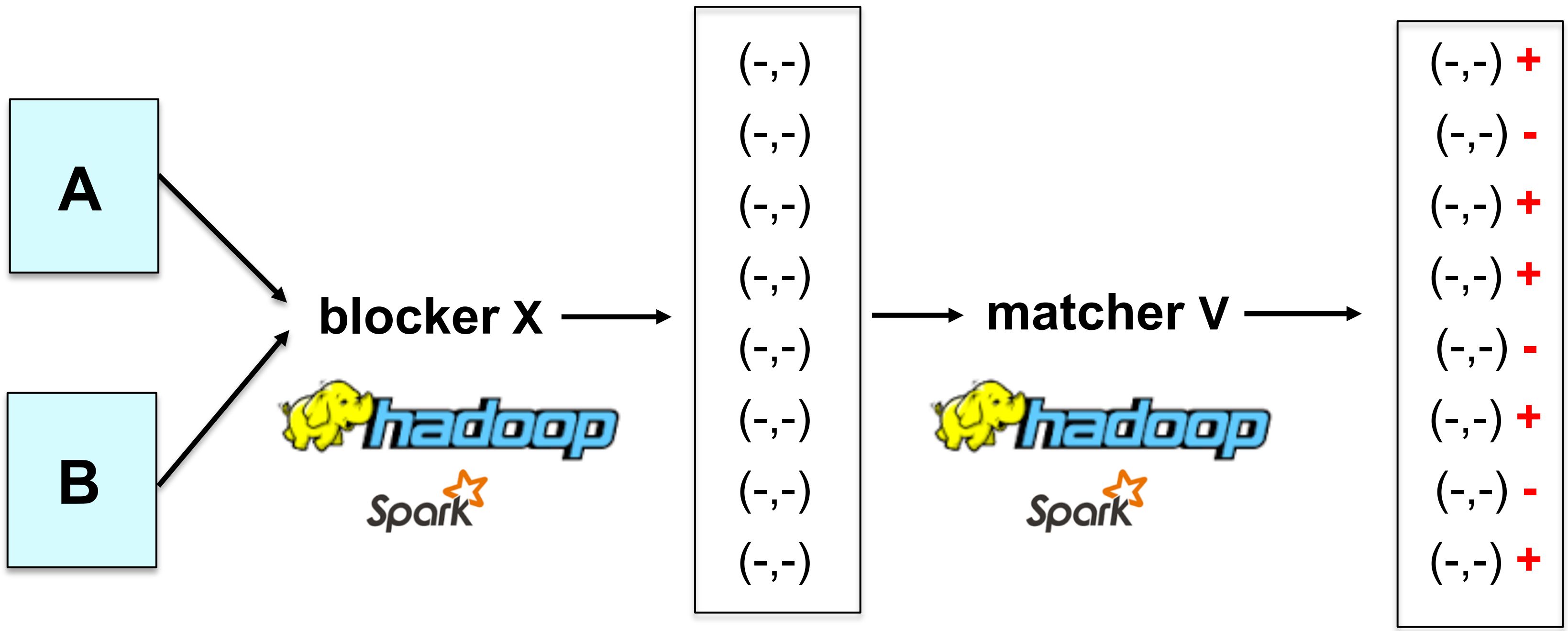
Development Stage



Select the best blocker: X, Y

Select the best matcher: U, V

Production Stage



Scaling, quality monitoring, exception handling, crash recovery, …

Limitations of Current EM Systems

- **Examined 33 systems**
  - 18 non-commercial and 15 commercial ones

1. Do not cover the entire EM workflow

2. Hard to exploit a wide range of techniques

- Visualization, learning, crowdsourcing, etc.

3. Do not distinguish development vs production stages

4. Very little guidance for users

5. Not designed from scratch for extendability

Characteristics of 18 Non-Commercial Systems

Name	Affiliation	Scenarios	Blocking	Matching	Exploration, cleaning	User interface	Language	Open source	Scaling
Active Atlas	University of Southern California	Single table, two tables	Hash-based	ML-based (decision tree)	No	GUI, commandline	Java	No	No
BigMatch	US Census Bureau	Single table, two tables	Attribute equivalence, rule-based	Not supported	No	Commandline	C	No	Yes (supports parallelism on a single node)
D-Dupe	University of Maryland	Single table, two tables	Attribute equivalence	Relational clustering		GUI	C#	No	No
Dedoop	University of Leipzig	Single table	Attribute equivalence, sorted neighborhood	ML-based (decision tree, logistic regression, SVM etc.)	No	GUI	Java	No	Yes (Hadoop)
Dedupe	DataMade	Single table, two tables	Canopy clustering, predicate-based	Agglomerative hierarchical clustering-based	Yes	Commandline	Python	Yes	Yes
DuDe	University of Potsdam	Single table, two tables	Sorted neighborhood	Rule-based	Yes	Commandline	Java	Yes	No
Febrl	Australian National University	Single table, two tables	Full index, blocking index, sorting index, suffixarray index, qgram index, canopy index, stringmap index	Fellegi-Sunter, optimal threshold, k-means, FarthestFirst, SVM, TwoStep	Yes	GUI, commandline	Python	Yes	No
FRIL	Emory University	Single table, two tables	Attribute equivalence, sorted neighborhood	Expectation maximization	Yes	GUI	Java	Yes	Yes (supports parallelism on a single node)
MARLIN	University of Texas at Austin		Canopy clustering	ML-based (decision tree, SVM)					No
Merge Toolbox	University of Duisburg-Eissen	Single table, two tables	Attribute equivalence, canopy clustering	Probabilistic, expectation maximization	No	GUI	Java	No	No
NADEEF	Qatar Computing Research Institute	Single table, two tables		Rule-based	No	GUI	Java	No	No
OYSTER	University of Arkansas	Single table, two tables	Attribute equivalence	Rule-based	Yes	Commandline	Java	Yes	No
pydedupe	GPoulter (GitHub username)	Single table, two tables	Attribute equivalence	ML-based, rule-based	Yes	Commandline	Python	Yes	No
RecordLinkage	Institute of Medical Biostatistics, Germany	Single table, two tables	Attribute equivalence	ML-based, probabilistic	Yes	Commandline	R	Yes	No
SERF	Stanford University	Single table		R-Swoosh algorithm	No	Commandline	Java	No	No
Silk	Free University of Berlin	RDF data		Rule-based	Yes	GUI	Java	Yes	Yes (supports parallelism on a single node, Hadoop)
TAILOR	Purdue University	Single table, two tables	Attribute equivalence, sorted neighborhood	Probabilisitic, clustering, hybrid, induction	No	GUI	Java	No	No
WHIRL	William Cohen			Vector space model		Commandline	C++	No	No

Characteristics of 15 Commercial Systems

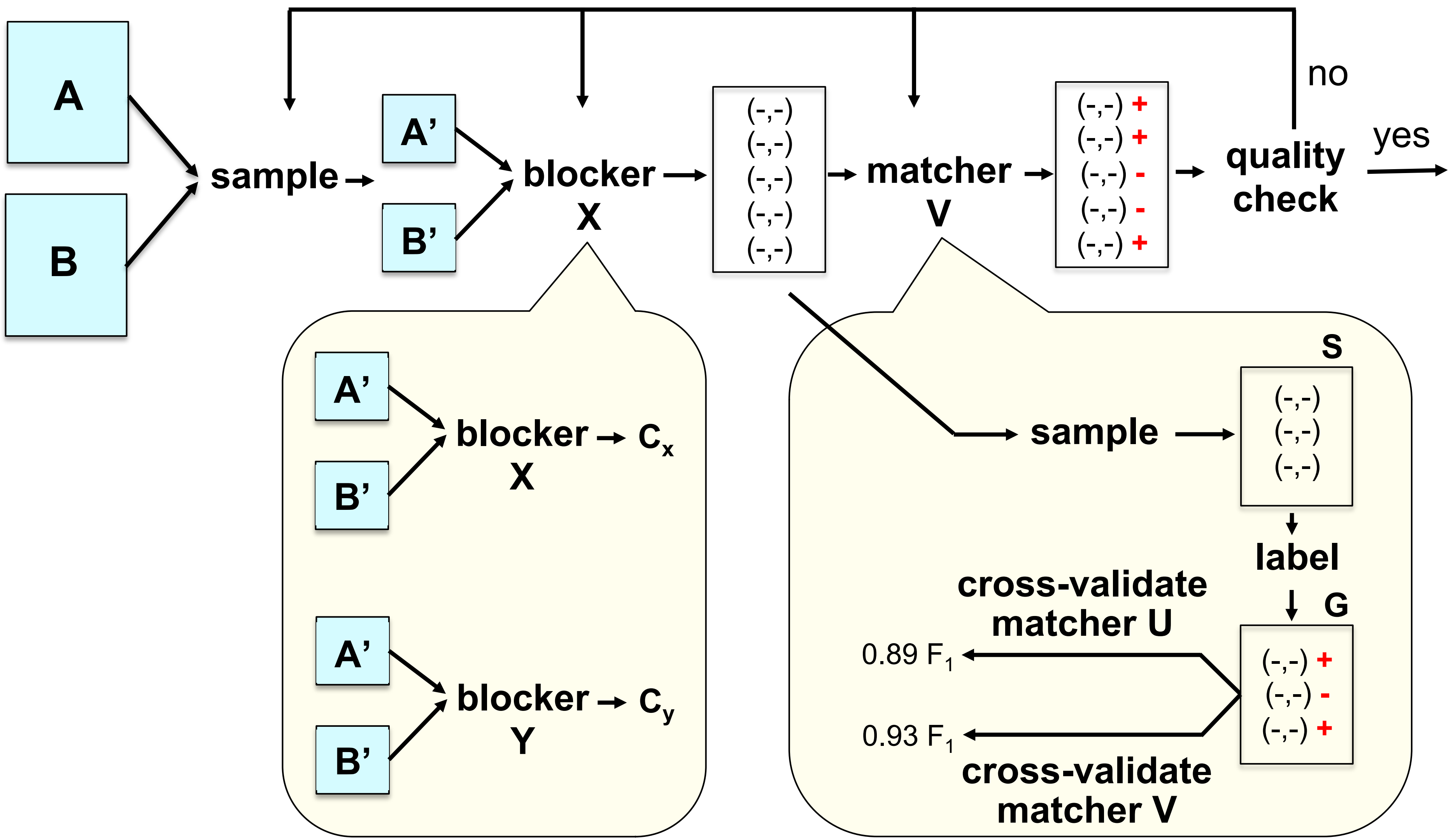
Name	Purpose and how EM fits in	Supported EM scenarios	Main user interface	Distinction between dev. and prod. stages	Language	Scripting environment
DataMatch from Data Ladder	Data cleaning, data matching. EM forms the core of their solution	Multiple tables	GUI	No		No
Dedupe.io	Record linkage, deduplication. EM forms the core of their solution	Single table, two tables	Web-based	No		No
FuzzyDupes	Duplicate detection, data cleaning. EM forms the core of their solution	Single table, two tables	GUI	No		No
Graphlab Create	EM is offered as a service on top of their GraphLab platform	Single table, two tables, linking records to a KB	Web-based		C++	Yes
IBM InfoSphere	Customer data analytics. EM is supported by a component (BigMatch) in the product	Single table, two tables	Web-based		Java	No
Informatica Data Quality	Improve data quality. EM forms a part of data quality pipeline	Single table, two tables	GUI			No
LinkageWiz	Data matching and data cleaning. EM forms the core of their solution	Single table, two tables	GUI	No		No
Oracle Enterprise Data Quality	Improve data quality. EM forms a part of data quality pipeline	Single table, two tables	GUI			No
Pentaho Data Integration	ETL, data integration. EM forms a part of ETL/data integration pipe line	Single table, two tables	GUI		Java	No
SAP Data Services	Improve data quality, data integration. EM forms a part of data integration pipeline	Single table, two tables	GUI	No		
SAS Data Quality	Improve data quality. EM forms a part of data quality pipeline	Single table, multiple tables	Web-based			Limited support
Strategic Matching	Data matching and data cleaning. EM forms the core of their solution	Single table, two tables	GUI	No		No
Talend Data Quality	Improve data quality. EM forms a part of data quality pipeline	Single table, two tables	GUI			No
Tamr	Data curation. EM forms a part of data curation pipeline	Multiple tables	Web-based	No	Java	No
Trillium Data Quality	Improve data quality. EM forms a part of data quality pipeline	Single table, multiple tables	GUI			No

1. Do Not Cover the Entire EM Workflow

- **Focus on blocking and matching**
  - Develop ever more complex algorithms
  - Maximize accuracy and minimize costs

- **Assume other steps are trivial**
  - **In practice these steps raise serious challenges**
  - Example 1: sampling to obtain two smaller tables A’ and B’
  - Example 2: sample a set of tuple pairs to label
  - Example 3: label the set

Development Stage



Select the best blocker

Select the best matcher

Example 1: Sampling Two Smaller Tables

- **Tables A and B each has 1M tuples**
  - Very difficult to experiment with them directly in development stage
  - Way too big, so too time consuming
- **Need to sample smaller tables**
  - A' from A, B' from B, say 100K tuples for each table
- **How to sample?**
  - Random sampling from A and B may result in very few matching tuple pairs across A' and B'
  - How to resolve this?

Example 2: Take a Sample from the Candidate Set (for Subsequent Labeling)

- **Let C be the set of candidate tuple pairs produced by applying a blocker to two tables A' and B'**

- **We need to take a sample S from C, label S, then use the labeled set to find the best matcher and train it**

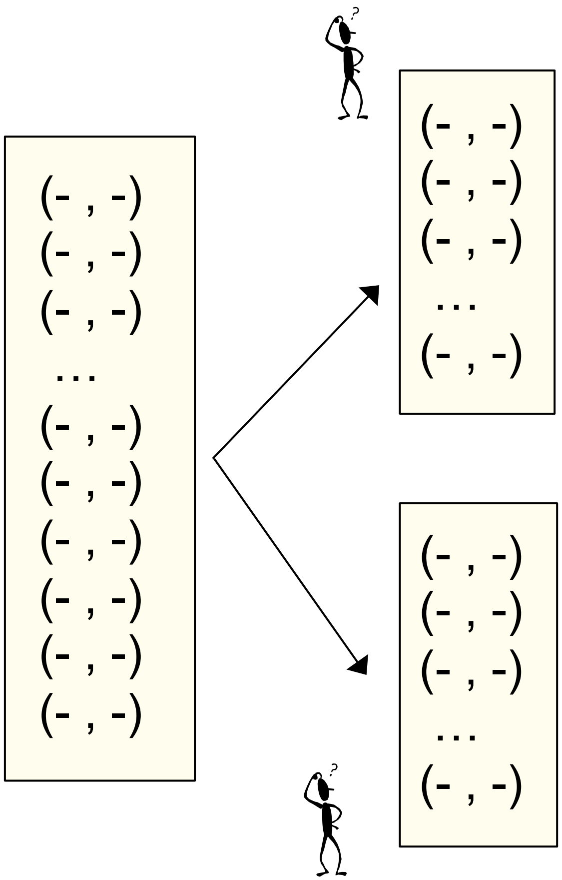
- **How to take a sample S from C?**
  - Random sampling often does not work well if C contains few matches
  - In such cases S contains no or very few matches

Example 3: Labeling the Sample

- **This task is often divided between two or more people**
- **As they label their set of tuple pairs, they may follow very different notions of matching**
  - E.g., given two restaurants with same names, different locations
  - A person may say “match”, another person may say “not a match”
- **At the end, it becomes very difficult to reconcile different matching notions and relabel the sample**
- **This problem becomes even worse when we crowdsource the labeling process**

An Illustrating Example for Distributed Labeling

Two restaurants match if they refer to the same real-world restaurant



([Laura's, 23 Farewell Str], [Laura, 23 Farewell]) +

([Palmyra, 46 Main St], [Palmyra, 15 Broadway]) -

([KFC, 24 Main St], [KFC, 41 Johnson Ave]) +

2. Hard to Exploit a Wide Range of Techniques

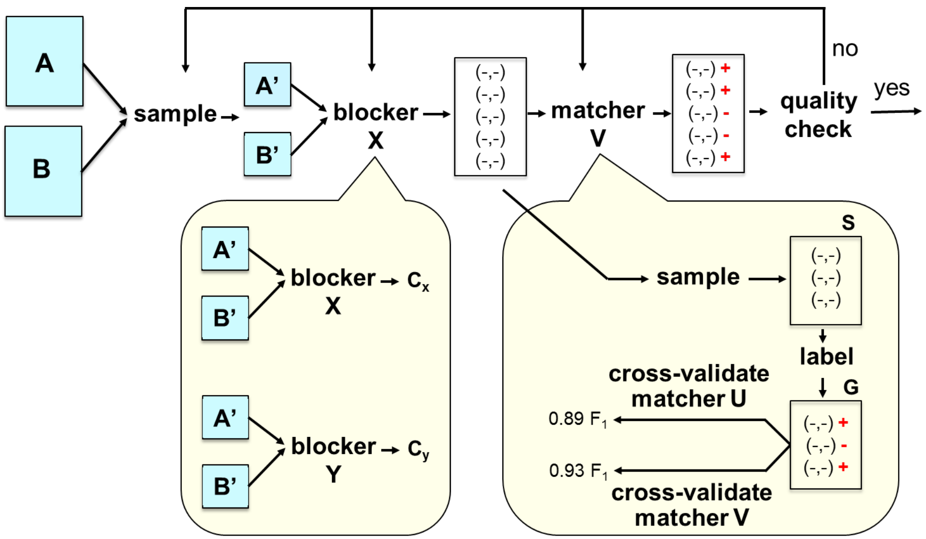
- **EM steps often exploit many techniques**
  - SQL querying, keyword search, learning, visualization, information extraction, outlier detection, crowdsourcing, etc.

- **Difficult to incorporate all into a single system**

- **Difficult to move data repeatedly across systems**
  - An EM system, a visualization system, an extraction system, etc.

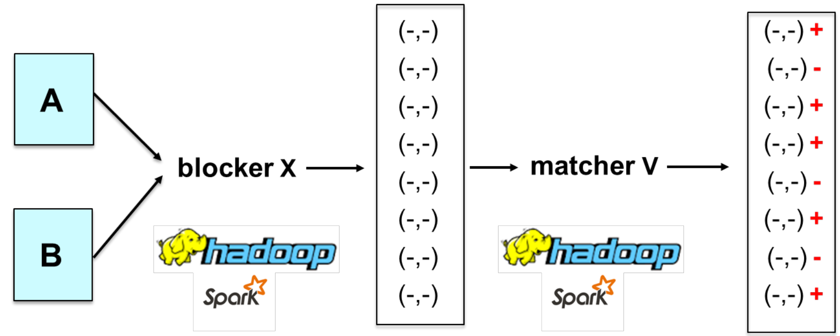
- **Problem: most systems are stand-alone monoliths, not designed to play well with other systems**

3. Do Not Distinguish Dev vs Prod Stages



Select the best blocker

Select the best matcher



Scaling, quality monitoring, exception handling, crash recovery, …

- **Current systems**
  - Provide a set of blockers / matchers
  - Provide a way to specify / optimize / execute workflows
  - Pay very little attention to the development stage

4. Little Guidance for Users

- **Suppose user wants at least 95% precision & 80% recall**
- **How to start? With rule-based EM first? Learning-based EM first?**
- **What step to take next?**
- **How to do a step?**
  - E.g., how to label a sample?
- **What to do if after much effort, still hasn't reached desired accuracy?**

5. Not Designed from Scratch for Extendability

- **Can we build a single system that solves all EM problems?**
  - No
- **In practice, users often want to**
  - Customize, e.g., to a particular domain
  - Extend, e.g., with latest technical advances
  - Patch, e.g., writing code to implement lacking functionalities
- **Users also want interactive scripting environments**
  - For rapid prototyping, experimentation, iteration

- **Most current EM systems**
  - Are not designed so that users can easily customize, extend, patch
  - Are not situated in interactive scripting environments

Summary

Many serious limitations:

1. **Do not cover the entire EM workflow**
2. **Hard to exploit a wide range of techniques**
3. **Do not distinguish development vs production stages**
4. **Very little guidance for users**
5. **Not designed from scratch for extendability**

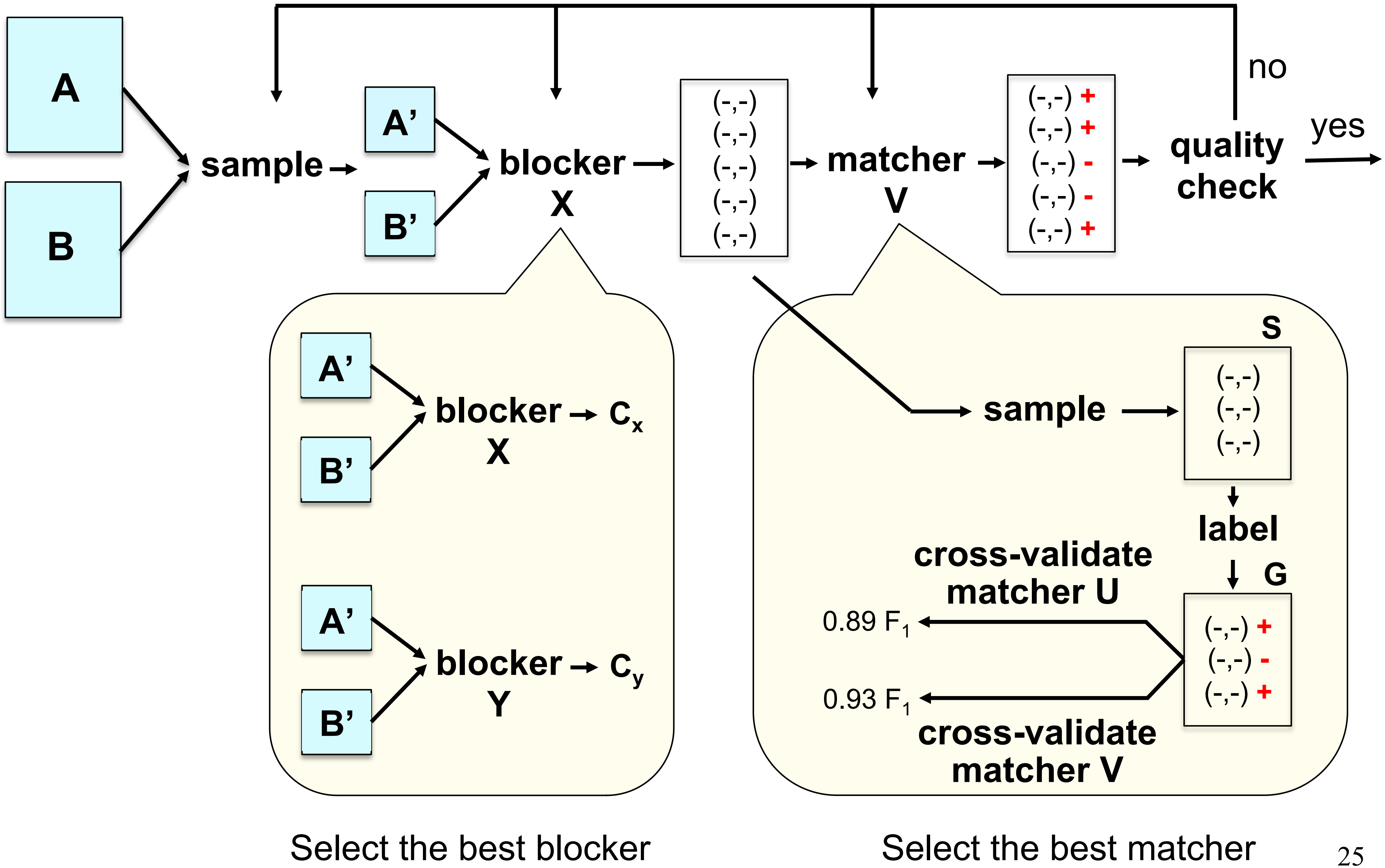
Our Solution: The Magellan Approach

- **Define a clear scope**
  - Each system targets **a set of EM scenarios** and **power users**

- **Solve the development stage**
  - Develop a **how-to guide**
    - Helps users discover accurate workflow
    - Must cover all steps
    - Tells users what to do, step by step
  - Develop **tools for pain points** in the guide
    - On top of **PyData ecosystem**

- **Solve the production stage in a similar way**
  - But focus on scalability, crash recovery, etc.

How-to Guide/Tools for Development Stage



Example How-to Guide for Matching Tables Using Supervised Learning

1. Load tables A and B into Magellan. Downsample if necessary.

2. Perform blocking on the tables to obtain a set of candidate tuple pairs C.

3. Take a random sample S from C and label pairs in S as matched / non-matched.

4. Create a set of features then convert S into a set of feature vectors H. Split H into a development set I and an evaluation set J.

5. Repeat until out of debugging ideas or out of time:

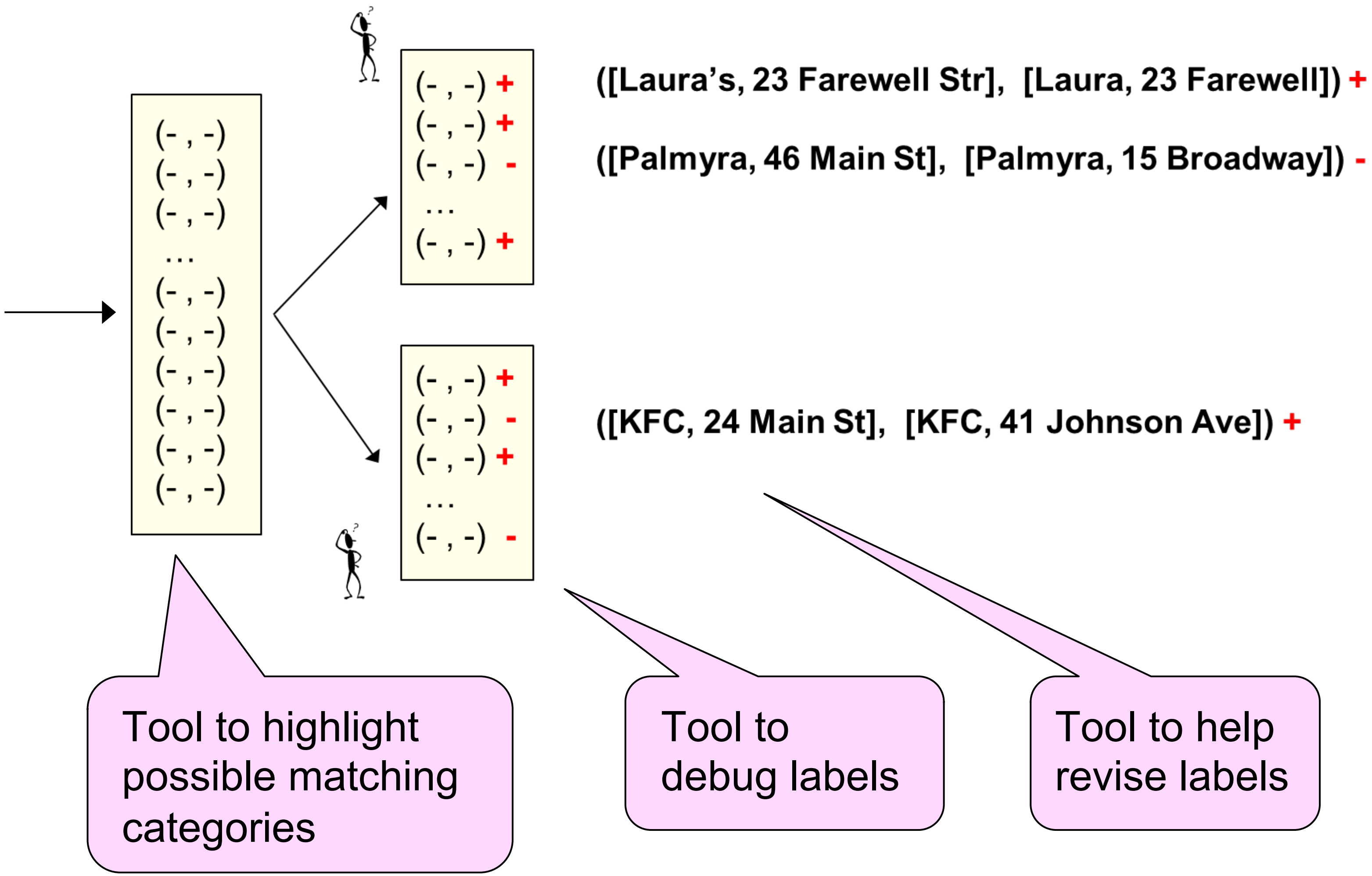
   (a) Perform cross validation on I to select the best matcher. Let this matcher be X.

   (b) Debug X using I. This may change the matcher X, the data, labels, and the set of features, thus changing I and J.

6. Let Y be the best matcher obtained in Step 5. Train Y on I, then apply to J and report the matching accuracy on J.

How-to Guide/Tools for Development Stage

Users want step-by-step guide on how to take a sample then label it



Build Tools on the PyData Ecosystem

- **Key observation**
  - **Development stage does a lot of data analysis**
    - E.g., analyzing data to discover EM matching rules
    - Often requires cleaning, visualizing, finding outliers, etc.
  - Very hard to incorporate all such techniques into a single EM system
  - **Better to build on top of an open-source data ecosystem**

- **Two major current ecosystems**
  - Python and R

- **PyData ecosystem**
  - Used extensively by data scientists
  - > 100K packages (in PyPI)
  - Data analysis stack / big data stack

The Magellan Architecture

Match
two tables

Power
Users

Development Stage

How-to guide

Tools for pain points
(as Python commands)

Data samples

→ EM Workflow →

Production Stage

How-to guide

Tools for pain points
(as Python commands)

Original data

PyData eco system

Data Analysis Stack

pandas, scikit-learn, matplotlib,
numpy, scipy, pyqt, seaborn,
…

Big Data Stack

PySpark, mrjob, Pydoop,
pp, dispy,
…

Python Interactive Environment
Script Language

Raises Numerous R&D Challenges

- **Developing good how-to guides is very difficult**
  - Even for very simple EM scenarios
- **Developing tools to support how-to guides raises many research challenges**
  - Accuracy, scaling
- **Novel challenges for designing open-world systems**

Examples of Current Research Problems

- Profile the two tables to be matched,
  to understand different matching definitions
- Normalize attribute values using machine and humans
- Verify attribute values using crowdsourcing
- Debug the blocking / labeling / matching process
- Scale up blocking to 100s of millions of tuples
- Apply Magellan template to string similarity joins
- …

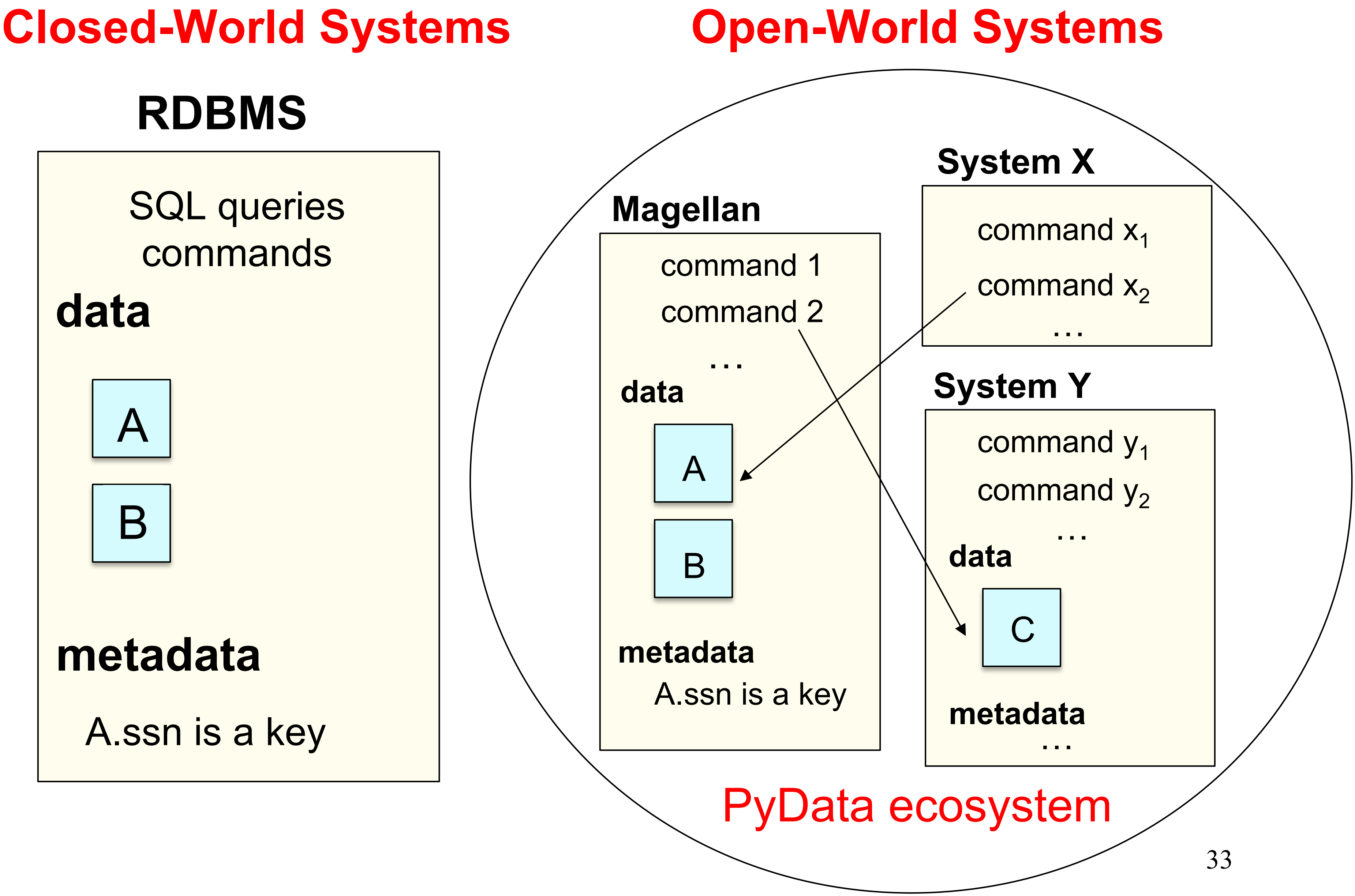
- Our group is working on many of the above challenges

Designing for an Open World

- **Magellan has been built as an open-world system**
  - On top of Python data ecosystem
  - Relies on external systems to supply tools in visualization, mining, IE, etc.

- **Raises many non-trivial challenges**
  - Managing metadata
  - Designing data structures
  - Handling missing values
  - Package version incompatibilities
  - Data type mismatch
  - …

Metadata Management



Metadata Management: Naïve Solution

- **Rewrite the external commands to be metadata aware**

- **Issues:**
  - Need a lot of developer effort
    - Impractical given the large number of commands and packages that users can use
  - Cannot force the user to wait till a developer has made an external command metadata aware

Metadata Management: Current Solution

- **Design each command in Magellan to be metadata aware**
- **Each command at the start, checks for all the metadata constraints that it requires to be true**
  - E.g. primary key constraint must be satisfied to operate on Table A
  - Command will not proceed until all the required constraints are satisfied
- **During its execution it will try to manage metadata properly**
- **If it encounters an invalid constraint, it will alert the user**
  - But will continue execution as the constraint is not critical for the correct execution

Designing Data Structures

- **At the heart of Magellan is a set of tables**
  - Tuples to be matched are stored in two tables A and B
  - Intermediate and final results can also be stored in tables
  - Need to store metadata
  - Important to study how to implement tables
- **Design alternatives**
  - Use Pandas data frame to store and process tables
  - Define a new class with multiple fields.
    - One field stores the data frame and other fields store metadata
  - Subclass Pandas data frame and add fields to store metadata

Alternative 1: Use Pandas Data Frame

- **Pandas is a popular package to store and process tables using data frame data structure**

- **Naïve solution is to implement Magellan tables as Pandas data frames**

- **Issue: cannot store metadata**
  - E.g. primary key of a table

Alternative 2:
Include Pandas Data Frame in Another Class

- **Define a new class, MTable and include a field for Pandas data frame and other fields for metadata**

- **Issues:**
  - Makes it difficult for other packages operate on Magellan's data
    - Existing packages are completely unaware of MTable
    - Commands in these packages cannot operate on MTable objects directly
  - Need to redefine commands from other packages, a time-consuming and brittle process

Alternative 3: Inherit Pandas Data Frame

- **Subclass Pandas data frame to define a new class MDataframe**
  - Include fields to store metadata
- **Any existing command that knows data frames can operate on MDataframe objects**

- **Issue:**
  - Commands may return inappropriate type of objects
    - Can be quite confusing to users

Current Solution: Pandas Data Frame + Catalog

- **Store Magellan tables as Pandas data frames**
  - Any existing Python package that manipulates data frames will work
  - Maximizes the chances of interoperating with other packages seamlessly
- **Store metadata in a separate object, catalog**
  - Similar to RDBMS
  - Stores metadata for each table in Magellan
  - Magellan commands which require metadata can probe the catalog
- **General principle**
  - Use data structures that are most common to other systems to store its data
  - When not possible, provide procedures to convert between its own type and the ones commonly used by other systems

Current Status of Magellan

- **Has been in development since June 2015**
  - ~18 months
  - 1 main developer + 2 contributors
- **Contains 7 major new tools for how-to guides**
- **Built on top of 11 different packages from PyData ecosystem**
  - E.g., Pandas, Scikit-learn, etc.
- **Exposes 104 commands to users**
- **Codebase includes 87 Python files with ~14K lines of Python code**

Current Status of Magellan

- **Package is comprehensively tested**
  - 1136 unit test cases
  - 90 performance test cases
- **Codebase is extensively documented**
  - 5K lines of comments
- **Most advanced and comprehensive open-source EM system available today**
- **Used extensively in education, science and at several companies**

Current Status of Magellan

- **Used as a teaching tool for data science classes at UW**
  - CS 638: 83 students
  - CS 784: 44 students
- **Used in biomedicine domain to match drugs**
  - 2 accepted posters
  - Highlighted in CPCP 2017
- **Used at companies**

@WalmartLabs Johnson Controls RECRUIT RECRUIT Institute of Technology

Marshfield Clinic

- **Resulted in a research paper and a demonstration at VLDB '16**

Experiments with 44 Students

Team	Domain	Table A Size	Table B Size	Cand. set Size	First learning Iteration (A)			Final best learning matcher (B)			Num. of Iterations	Final matcher (C)			Num. of Iterations	Diff. in F1 between (C) and (A)
					P	R	F1	P	R	F1		P	R	F1		
1	Vehicles	4786	9003	8009	71.2	71.2	71.2	91.43	94.12	92.75	4	100	100	100	2	28.8
2	Movies	7391	6408	78079	99.28	95.13	97.04	98.21	100	99.1	2	100	100	100	1	2.01
3	Movies	3000	3000	1000000	98.9	99.44	99.5	98.63	98.63	98.63	1	98.63	98.63	98.63	0	-0.87
4	Movies	3000	3000	36000	68.2	69.16	68.6	98	100	98.99	3	98	100	98.99	1	30.39
5	Movies	6225	6392	54028	100	95.23	97.44	100	100	100	3	100	100	100	1	2.56
6	Restaurants	6960	3897	10630	100	37.5	54.55	100	88.89	94.12	3	100	88.89	94.12	1	39.57
7	Electronic Products	4559	5001	823832	73	51	59	73.3	64.71	68.75	2	100	64.71	78.57	1	19.57
8	Music	6907	55923	58692	92	79.31	85.19	90.48	82.61	86.36	2	100	92.16	95.92	2	10.73
9	Restaurants	9947	28787	400000	100	78.5	87.6	94.44	97.14	95.77	4	94.44	97.14	95.77	0	8.17
10	Cosmetic	11026	6445	36026	56	56	56	96.67	87.88	92.06	3	96.43	87.1	91.53	4	35.53
11	EBooks	6482	14110	13652	96.67	96.67	96.67	100	95.65	97.78	4	100	98.33	99.13	1	2.46
12	Beer	4346	3000	4334961	84.5	59.6	65.7	100	60.87	75.68	4	91.3	91.3	91.3	4	25.6
13	Books	3506	3508	2016	93.46	100	96.67	91.6	100	95.65	2	91.6	100	95.65	0	-1.02
14	Books	3967	3701	4029	74.17	82.2	82.5	100	84.85	91.8	3	100	84.85	91.8	5	9.3
15	Anime	4000	4000	138344	95.9	88.9	92.2	100	100	100	2	100	100	100	1	7.8
16	Books	3021	3098	931	74.2	100	85.2	96.34	84.95	90.29	2	94.51	92.47	93.48	1	8.28
1																
1																
1																
2																
2																
2																
2																
24	Baby Products	10000	5000	11000	78.6	44.8	57.7	96.43	72.97	83.08	5	100	72.97	84.37	2	26.67

- **Baseline: P = 56-100%, R = 37-100%, F1 = 56-99%**
- **Magellan: P = 91-100%, R = 64-100%, F1 = 78-100%**
  - 20 teams out of 24 achieved recall above 90%

Experiments with 44 Students

- **Tools for pain points were highly effective**
- **Debugging blockers**
  - 18 out of 24 teams used the debugger, for 5 iterations on average
  - Debugger helps in (a) cleaning data
    (b) finding correct blocker types/attributes
    (c) tuning blocker parameters
    (d) knowing when to stop
- **Debugging matchers**
  - Teams performed 3 debugging iterations on average
  - Actions performed include (a) feature selection
    (b) data cleaning
    (c) parameter tuning
- **Students extensively used visualization, extraction, cleaning, etc. (using PyData packages)**

Magellan "in the Wild"

- **WalmartLabs**
  - Helped improve a system already in production
- **Johnson Controls**
  - Matched hundreds of thousands of suppliers for JCI
  - Precision above 95%, recall above 92% (across many data sets)
- **Marshfield Clinic**
  - Matched 18M pairs of drugs
  - Precision: 99.18% Recall: 95.29%
- **Raised additional interesting challenges**
  - Data can be very dirty, need far more cleaning tools

Novelties in the Current Work

- **Conceptual novelties:**
  - Radically different from current EM systems
  - Conceptually novel architecture and methodology
    - Distinguish between development & production stages
    - Provide how-to guides
    - Identify pain points and develop supporting tools
    - Implement tools on top of the PyData ecosystem
- **Technical novelties:**
  - Realizing such conceptual novelties raises many research problems
  - Many of them are pursued by members of our group
  - Provided preliminary solution for some of the problems
    - Metadata management, designing data structures
- **Practical impact:**
  - Magellan has been released as an open-source tool
  - Used in education, science and companies

For More Details

- http://www.vldb.org/pvldb/vol9/p1197-pkonda.pdf
- Check out Magellan under
  http://pages.cs.wisc.edu/~anhai/
- GitHub: github.com/anhaidgroup