



Semantische Daten- und Prozessintegration

ALSAYED ALGERGAWY
BIRGITTA KÖNIG-RIES

Hallo

- Dr. Ing. Alsayed Albergawy
- Postdoc at Heinz Nixdorf Endowed Chair for Distributed Information Systems
 - Birgitta König-Ries
- Research Interests:
 - Schema/data integration
 - Schema/ontology matching
 - Scientific (semantic) data integration
 - Biodiversity data management
- Room: Leutragraben 1, Jentower Room: 17N05
- alsayed.albergawy@uni-jena.de



And you....

- Your name?
 - Your specilaziation?
 - Your institute/department?
 - Your backgorund/SW skills
-
- Your expectation from this course?

Lecture

- Introducing principle methods of data integration
- discussing how to evaluate data integration results
- Topics
 1. Introduction to data integration
 2. String, schema, and data matching
 3. Query processing
 4. Web data: XML and ontology
 5. Semantic data integration
 6. Uncertainty
 7. Semantic web services

Schedule

| Week | Topic |
|-------------------------|---|
| Tues. 03.11.2020 | Ch. 1: Introduction |
| Tues. 10.11.2020 | Ch. 2: Query expression |
| Tues. 17.11 | Ch. 3: Data sources description |
| Tues. 24.11 | Ch. 4: String matching |
| Tues. 01.12 | Ch. 5: Schema matching & mapping |
| Tues. 08.12 | Ch. 7: Data matching (Project discussion) |
| Tues. 15.12 | Ch. 8: Query processing -1 |
| Tues. 22.12 | Ch. 8: Query processing -2 |
| 24.12-02.01.2019 | Charismas break |
| Tues. 05.01.2021 | Ch. 9: Wrapper |
| Tues. 12.01.2021 | Ch. 10: XML |
| Tues. 19.01 | Ch. 11: Uncertainty |
| Tues. 26.01 | Ch. 12: Ontology |
| Tues. 02.02 | Ch. : Web data integration |
| Tues. 09.02 | Ch. : Semantic web services (Prof. König-Ries) |

Course organization

- Course Webpage
 - <https://fusion.cs.uni-jena.de/fusion/classes/semantische-daten-und-prozessintegration-6/>
- Location and time
 - Online course
 - ❖ <https://bbb.fmi.uni-jena.de/b/als-iin-aeck-fbd>
 - Tuesday 10:00 bis 12:00 (typically 10:15 bis 11:45)
- Overall grading
 - Oral exam (30 min)
 - 15 min for a project presentation
 - 15 min for topic questions/discussion

Course organization

- All materials will be made available in “moodle”
 - <https://moodle.uni-jena.de/>
 - WS2020-23004 (ONLINE-PLUS im WS 2020/21: (Semantische) Daten- und Prozessintegration)
- Material for the upcoming week will be available in “moodle”
- Moodle forum for questions and discussions
- Two-session per lecture (~ 40 min web conference / week) to deepen the content (~10 min break in-between)

Similar (Online) courses

- Data Warehouse Concepts, Design, and Data Integration ([link](#))
- Big Data Integration and Processing ([link](#))
- Web Data Integration @Mannheim University ([link](#))
- Online book: Principles of data integration ([link](#))

Introduction and Course Outline

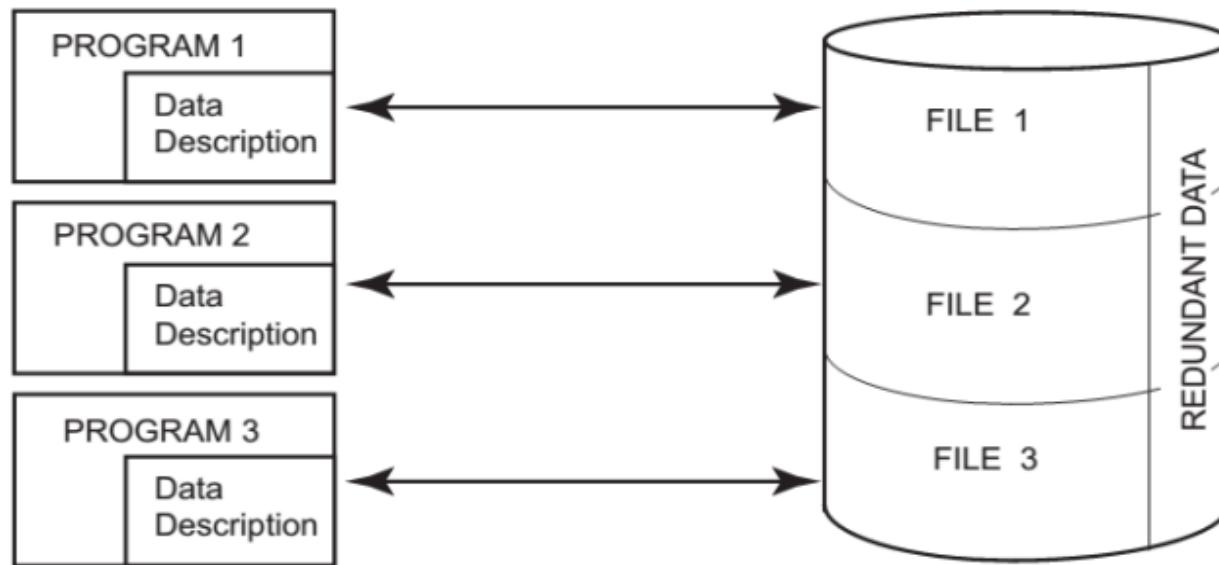
- ✓ Course outline and organization
- Introduction
- What is data integration?
- Why is data integration important?
- Schema heterogeneity
- Goal of data integration, why it's a hard problem
- Data integration architectures
- Course outline and organization



Semantische Daten- und Prozessintegration:

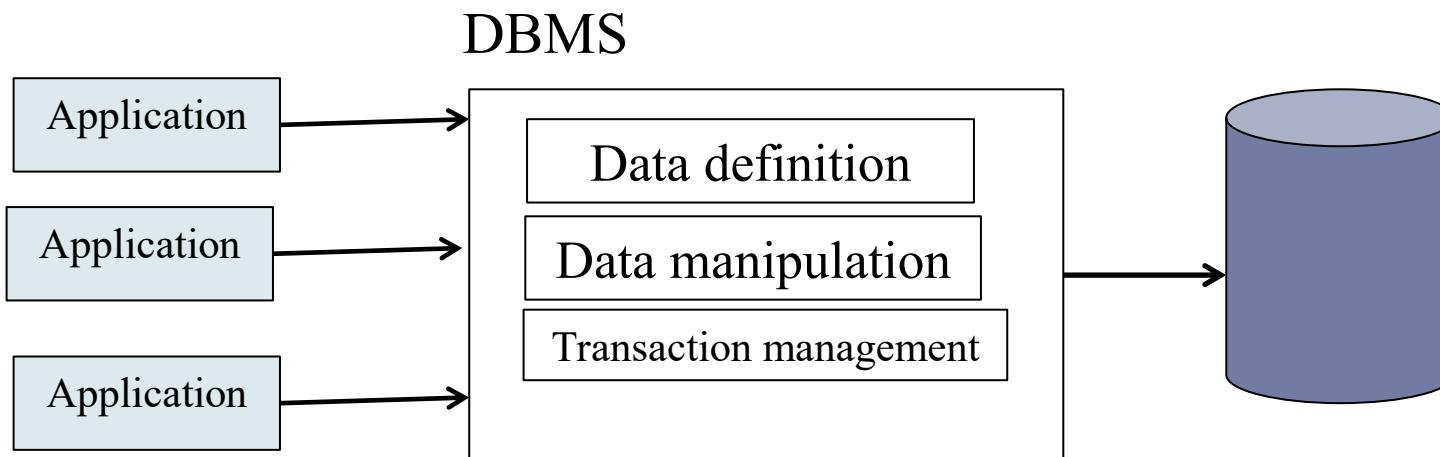
Introduction

File data management



- Each application defines and maintains its own data

Centralized data management



- New requirements:
 - Support for de-centralized organization structures
 - High availability
 - High performance
 - Scalability

Database management system

- What is (are) data?

Definition

A **database (DB)** is an organized collection of related data.

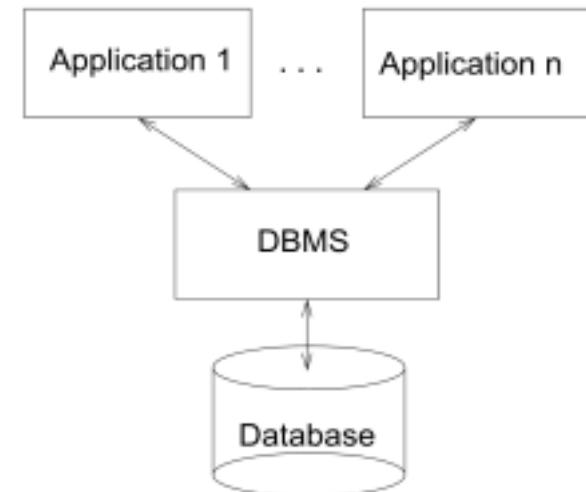
Definition

A **database management system (DBMS)** is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

- A database system (DBS)= DB+DBMS

Architecture of a DBS

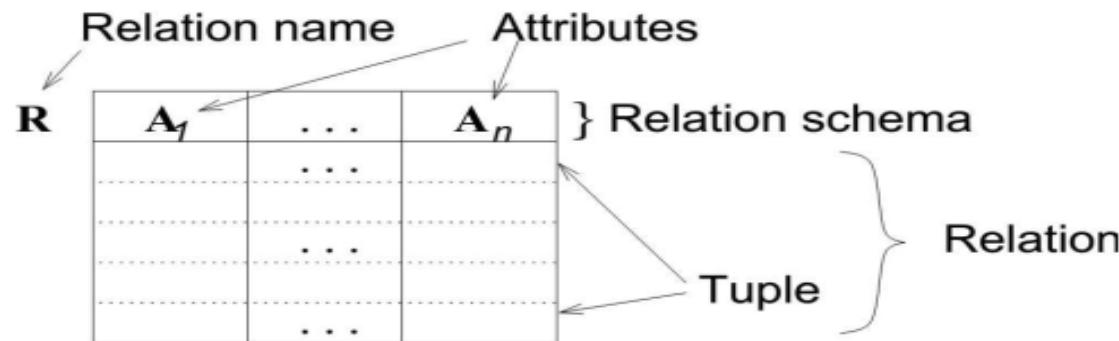
- **Applications** access database using specific views (external schema)
- The **DBMS** provides access for all applications using the logical schema
- The **database** is stored on secondary storage according to an internal schema



Relational model

Definition

A **relational database** is a database that is structured according to the relational database model. It consists of a set of relations.



- A relation schema R is denoted by $R(A_1, A_2, \dots, A_n)$, where:
 - ▶ R is the relation name
 - ▶ A_1, A_2, \dots, A_n is a set of attributes
 - ▶ The *degree* of a relation is the number of attributes n of its relation schema
- A relation (or relation state) r of the relation schema $R(A_1, A_2, \dots, A_n)$, also denoted by $r(R)$, is a set of $n - tuples$
 $r = \{t_1, t_2, \dots, t_m\}$
 - ▶ the number of tuples of the relation defines its *cardinality*

DBMS: it's all about abstraction

- *Logical vs. Physical; What vs. How.*

Students:

| SSN | Name | Category |
|-------------|---------|-----------|
| 123-45-6789 | Charles | undergrad |
| 234-56-7890 | Dan | grad |
| | ... | ... |

Takes:

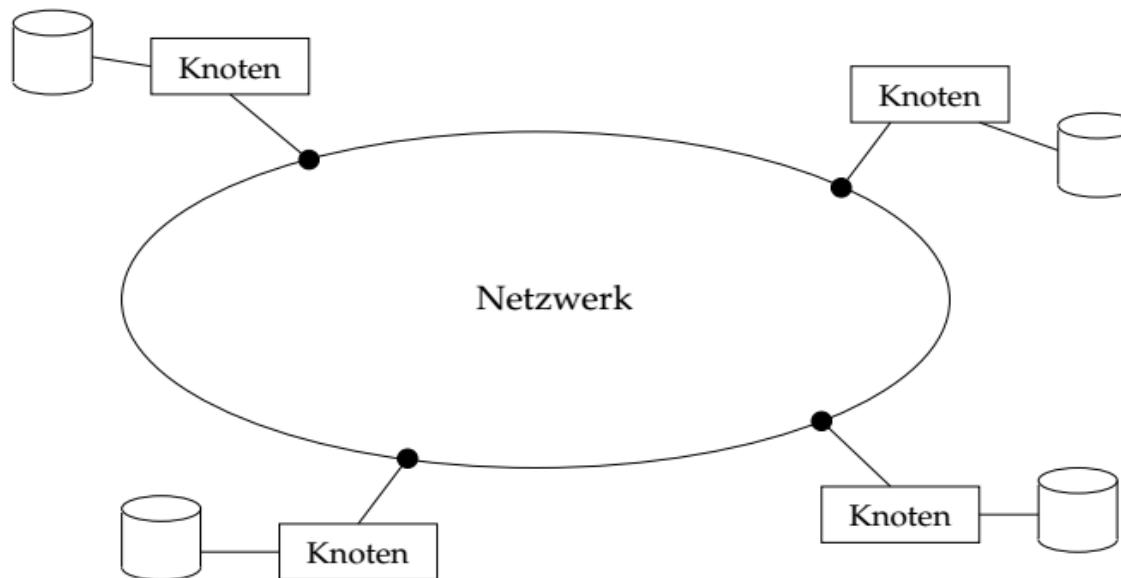
| SSN | CID |
|-------------|--------|
| 123-45-6789 | CSE444 |
| 123-45-6789 | CSE444 |
| 234-56-7890 | CSE142 |
| | ... |

Courses:

| CID | Name | Quarter |
|--------|-------------------|---------|
| CSE444 | Databases | fall |
| CSE541 | Operating systems | winter |

```
SELECT C.name  
FROM Students S, Takes T, Courses C  
WHERE S.name="Mary" and  
S.ssn = T.ssn and T.cid = C.cid
```

Distributed data management



Distributed computing

Definition

A **distributed computing system** is a collection of autonomous processing elements that are interconnected by a computer network

- The elements cooperate in order to perform the assigned task.
- The term **distributed** is used very broadly. The exact meaning of the word depends on the context. What can be distributed?
 - ▶ Processing logic
 - ▶ Functions
 - ▶ Data
 - ▶ Control
- Why do we distribute at all?
 - ▶ Current applications are inherently distributed
 - ▶ Corresponding to organizational structure of widely distributed enterprises
 - ▶ ...

Distributed database system

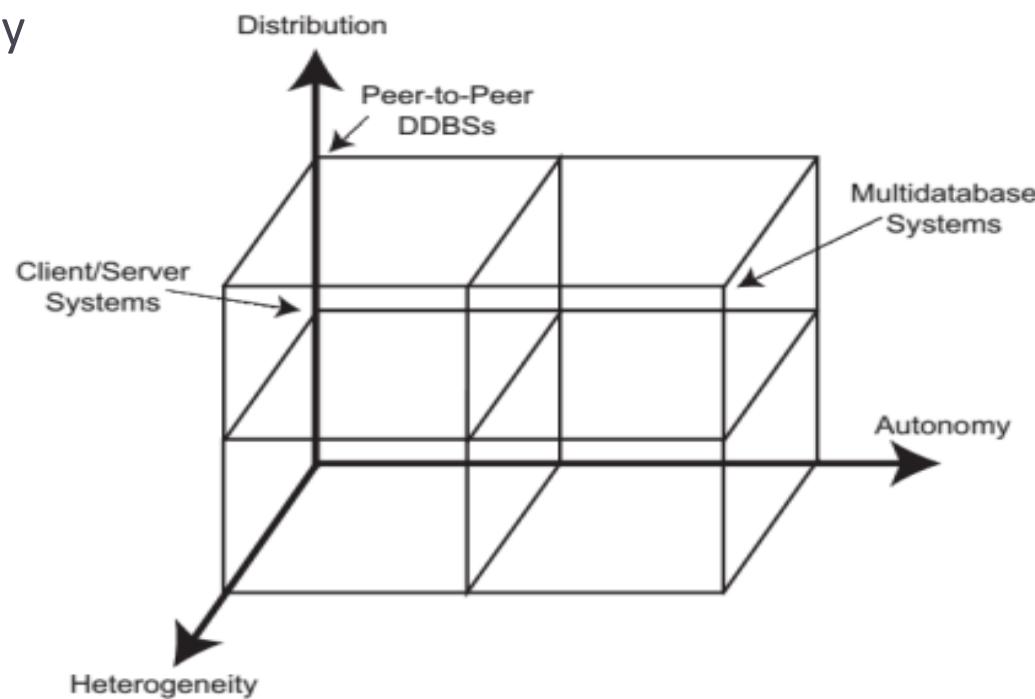
Definition

A **distributed database (DDB)** is a collection of multiple, *logically interrelated* databases *distributed* over a computer network

- A **distributed database management system (DDBMS)** is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users
- DDBS (distributed database system) = DDB + DDBMS
- Assumptions
 - ▶ Data stored at a number of sites each site logically consists of a single processor
 - ▶ Processors at different sites are interconnected by a computer network
 - ▶ DDB is a database, not a collection of files. Data is logically related; structured into multiple files; accessed through common interface.
 - ▶ DDBMS is a collection of DBMSs.

Architecture model for DDBS

- Architecture models for DDBSs can be classified along three dimensions:
 - Autonomy
 - Distribution
 - Heterogeneity



Architecture for DDBS

- **Autonomy**; Refers to the distribution of control (not of data) and indicates the degree to which individual DBMSs can operate independently (*tight integration, semiautonomous systems, or total isolation*)
- **Distribution**; Refers to the physical distribution of data over multiple sites (*no distribution, client/server, fully distribution*)
- **Heterogeneity**; Refers to heterogeneity of the components at various levels(*hardware, operating systems, communication, databases*)



Semantische Daten- und Prozessintegration



CHAPTER 1: INTRODUCTION TO DATA INTEGRATION



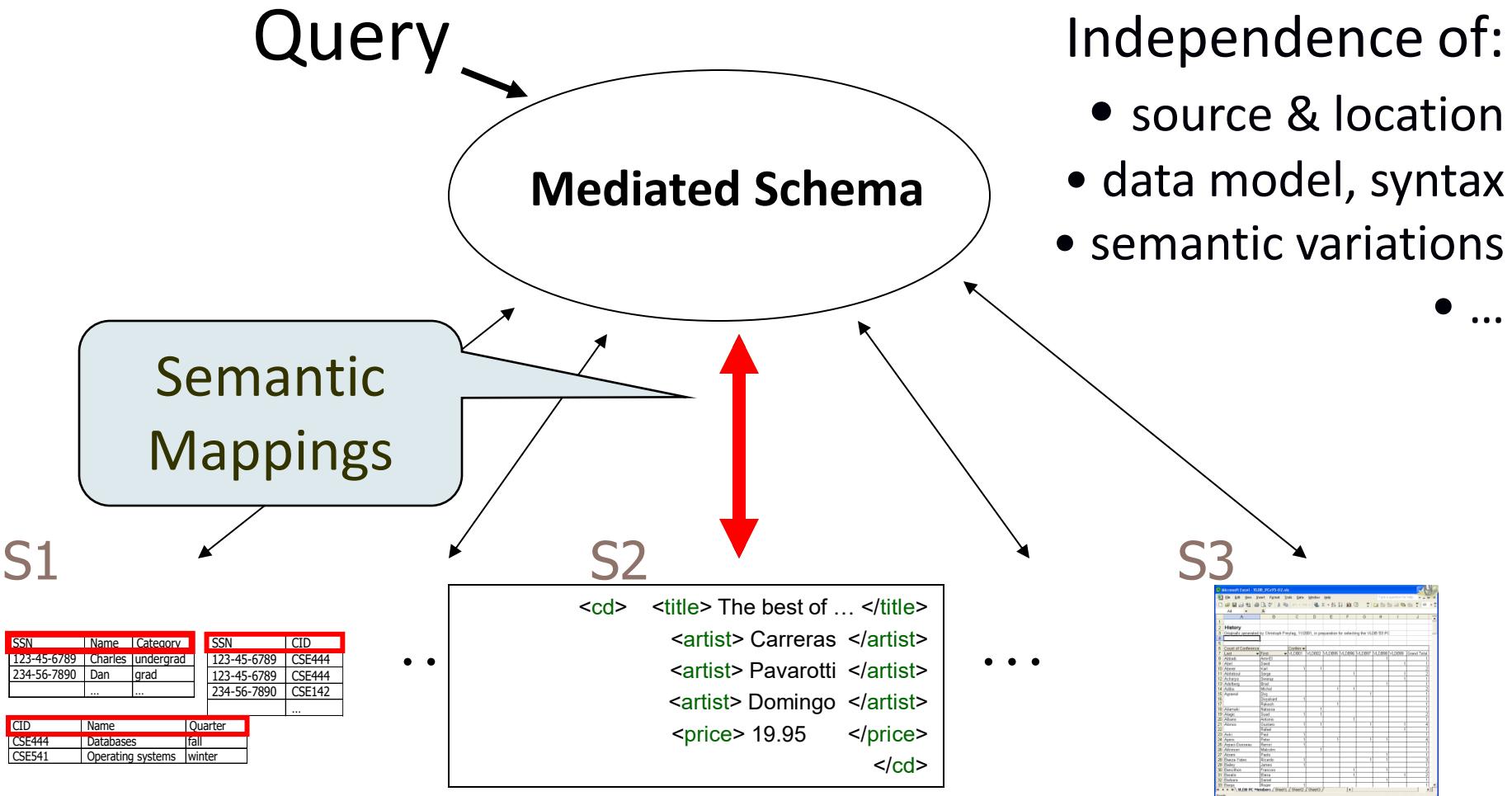
PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

Data integration

- Databases are great: they let us manage huge amounts of data
 - Assuming you've put it all into your schema.
- In reality, data sets are often created independently
 - Only to discover later that they need to combine their data!
 - At that point, they're using different systems, different schemata and have limited interfaces to their data.
- The goal of data integration: *tie together different sources, controlled by many people, under a common schema.*

Data integration



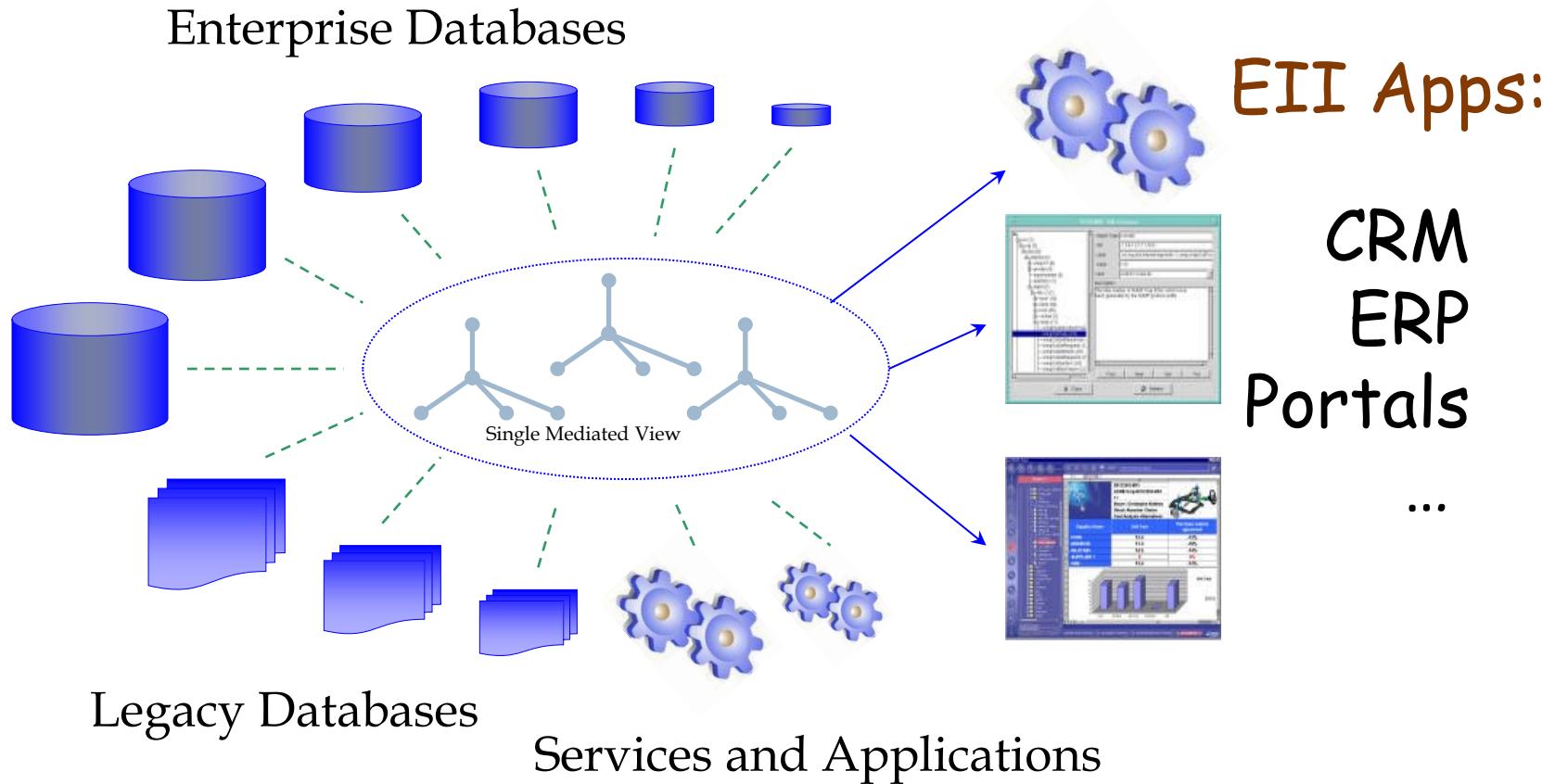
Outline

- ✓ Course outline and organization
- ✓ What is data integration?
- Why is data integration important?
 - Schema heterogeneity
 - Goal of data integration, why it's a hard problem
 - Data integration architectures

Applications of data integration

- Business
- Science
- Government
- The Web
- Pretty much everywhere

Application area 1: Business



50% of all IT \$\$\$ spent here!

Application area 2: The Web



 books & charts

 Quilt-Books.Com

 KATSUKI
BOOKS 日本書店

 Shelf
Books.com

 FOREIGN
BOOKS
www.foreignbooks.com

 COMPU-BOOKS

 Afrikan
Books.com

 USBORNE
CHILDREN BOOKS

 God Gave His Word

 Providence-Books
www.providence-books.com



 NGHOXI
books



 BORDERS.com
10 million books, CDs, and videos

 metro-Books

 BARNES
& NOBLE
www.bn.com

 StoneCreekBooks.com

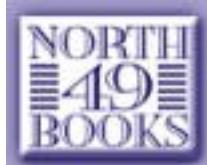
 ASTROLOGY-BOOKS.COM
Brought to you by SuperStar Books

 SeeMe4Books

 amazon.com.

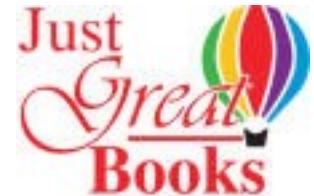
 HALF PRICE COMPUTER BOOKS DANCE BOOKS

 acma
BOOKS
.com

 NORTH
49
BOOKS

 funny face
Bookstore
SE BARENTHOUSE

 Professional
Books Inc.

 Just
Great
Books

Application area 2: The Web

<https://www.airbnb.de/getaways/Leipzig?destination=Berlin>

Wochenendausflug

Entdecke, welche Reiseziele für Wochenendausflüge in der Nähe von Leipzig wir zusammengestellt haben.

2 STUNDEN Berlin



1 / 20

So. | Mo. 19°C

Gründe, hierher zu kommen:

Die Berliner Mauer, historische Bedeutung, Bauhaus, Currywurst, Provokante Street Art, unnachgiebige Selbstdarstellung, Parties, die das ganze Wochenende dauern, Elektronische Musik, Künstler, die Hausbesetzer sind.



Verfügbar in Berlin



25qm/5.OG mit Dachterrasse...
Privatzimmer · Berlin



Wonderful flat with big roof-...
Ganze Unterkunft · Berlin



The Manhattan of Berlin App...
Ganze Unterkunft · Berlin



As a thank-you bonus, [site members](#) have access to a banner-ad-free version of the site, with print-friendly pages.

(Already a member? [Click here.](#))



[US Flags](#)

[EnchantedLearning.com](#)

US History



[US Geography](#)

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

[African-Americans](#)

[Artists](#)

[Explorers of the US](#)

[Inventors](#)

[US Presidents](#)

[US Symbols](#)

[US States](#)



[President's Day Activities](#)

[EnchantedLearning.com](#)

The Presidents of the United States of America

[In the order in which they served](#)

[Alphabetical order](#)

[Short table of Data](#)



[Abraham Lincoln](#)

The President and Vice-President are elected every four years. They must be at least 35 years of age, they must be native-born citizens of the United States, and they must have been residents of the U.S. for at least 14 years. (Also, a person cannot be elected to a third term as President.)

| President | Party | Term as President | Vice-President |
|--|-----------------------|-------------------|----------------------------------|
| 1. George Washington (1732-1799) | None, Federalist | 1789-1797 | John Adams |
| 2. John Adams (1735-1826) | Federalist | 1797-1801 | Thomas Jefferson |
| 3. Thomas Jefferson (1743-1826) | Democratic-Republican | 1801-1809 | Aaron Burr, George Clinton |
| 4. James Madison (1751-1836) | Democratic-Republican | 1809-1817 | George Clinton, Elbridge Gerry |
| 5. James Monroe (1758-1831) | Democratic-Republican | 1817-1825 | Daniel Tompkins |
| 6. John Quincy Adams (1767-1848) | Democratic-Republican | 1825-1829 | John Calhoun |
| 7. Andrew Jackson (1767-1845) | Democrat | 1829-1837 | John Calhoun, Martin van Buren |
| 8. Martin van Buren (1782-1862) | Democrat | 1837-1841 | Richard Mentor Johnson |
| 9. William H. Harrison (1773-1841) | Whig | 1841 | John Tyler |
| 10. John Tyler (1790-1862) | Whig | 1841-1845 | |
| 11. James K. Polk (1795-1849) | Democrat | 1845-1849 | |
| 12. Zachary Taylor (1784-1850) | Whig | 1849-1850 | |
| 13. Millard Fillmore (1800-1874) | Whig | 1850-1853 | |
| 14. Franklin Pierce (1804-1869) | Democrat | 1853-1857 | William King |
| 15. James Buchanan (1791-1868) | Democrat | 1857-1861 | John Breckinridge |

*Hundreds of millions of high-quality
tables on the Web*

The Deep Web

- Millions of high quality HTML forms out there
- Each form has its own special interface
 - Hard to explore data across sites.
- Goal (for some domains):
 - A single interface into a multitude of deep-web sources.

The Deep Web is accessible via HTML Forms

YAHOO! hotjobs®

Home Job Search My Searches My Jobs My Resumes Care

Search for Jobs Across the Web

Keyword(s)
(e.g. Job title, company, occupation)

City & State or Zip

Include surrounding cities

Job Category

Job Search Saved Searches

Find a job

Advanced search | Search help

Enter keywords

Select an employer

Select a job type

USAJOBS® USAJOBS is the official job site of the United States Federal Government. It's your one-stop source for Federal jobs and employment information.

Search Jobs **My USAJOBS** **Info Center** **Veterans** **Forms**

Basic Search | Agency Search | Series Search | Advanced Search | Search Help

Keyword Search ?

(e.g.: Job Title, Agency Name, Vacancy Announcement #, Control #) [More Tips](#)

Location Search ?

For multiple selections, hold down **Ctrl** (**Command** for Macs) while clicking selections.

----- Select all -----

US
AK
AK-Aleutian Islands
AK-Anchorage

Job Category Search ?

For multiple selections, hold down **Ctrl** (**Command** for Macs) while clicking selections.

----- SELECT ALL -----

Accounting, Budget and Finance
Biological Sciences
Business, Industry, and Procurement
Copyright, Patent, and Trademark

Salary Range ?

from to

OR

Pay Grade (GS) ?

from to

Every Classified

<http://www.everyclassified.com/>

University o...Engineering Interesting sites ▾ News ▾ Celidon ▾ Shopping ▾ Yahoo! Tools and Reference ▾ Apple .Mac Amazon eBay

EveryClassified beta

About EveryClassified Search

AUTOS



Research new and used makes and models to find the right car for you.

Make:

Model:

Year:

Location:
(Enter City, State or Zip Code)

RENTALS



Search for apartments or houses for rent.

Type:

From: to

Bed:

Location:
(Enter City, State or Zip Code)

REAL ESTATE



Make finding your next home easier. Search homes for sale.

From: to

Bed: Bath:

Location:
(Enter City, State or Zip Code)

JOBs



Find a job.

Keywords:
(ie. Retail, Sales, Management)

Location:
(Enter City, State or Zip Code)

PERSONALS



Find that special someone.

I am a:

seeking a:

Age: to

Location:
(Enter City, State or Zip Code)

GENERAL



Search for miscellaneous items for sale.

Keywords:
(ie. furniture, Wood, Bookshelf)

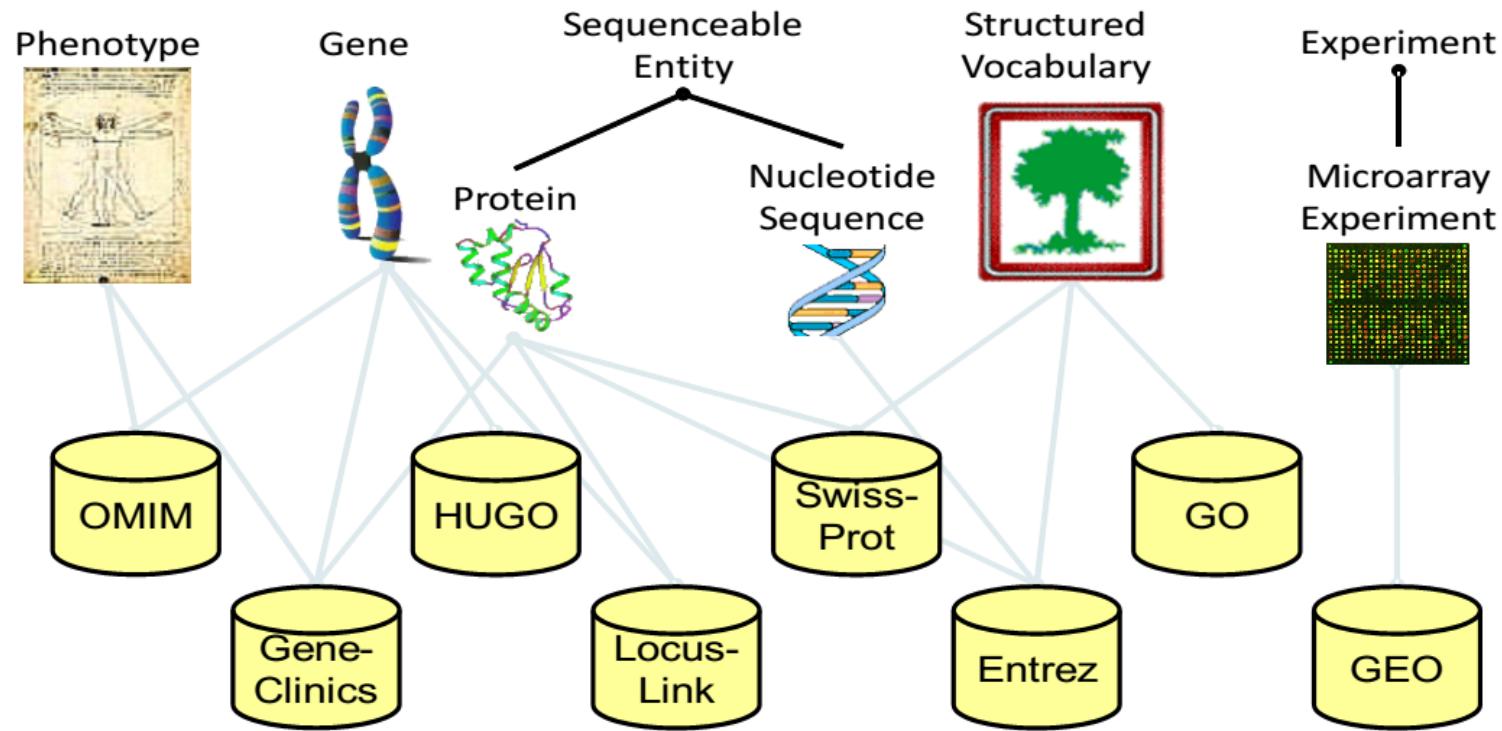
Location:
(Enter City, State or Zip Code)

[Autos](#) | [General](#) | [Jobs](#) | [Personals](#) | [Real Estate](#) | [Rentals](#) | 2005 EveryClassified [Submit a Site](#) Created by 

Create a single site to search for jobs/rentals/...

Application area 3: Science

Example: Gen ontology



Hundreds of biomedical data sources available; growing rapidly!

Application area 3: Science

Example: Biodiversity Research



Image: [AcidFlask](#) via flickr [CC license](#)



iDiv Data requirements along the way

How many and which?

- Taxonomic reference databases
- Plot inventory data (local)
- Remote sensing data (large scales)

Why do they co-occur?

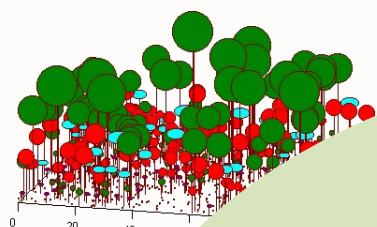
- Range distribution data
- Dated phylogenies
- Fossil data

Why do they coexist?

- Functional plant trait data
- Phylogenetic distances
- Herbivore count data

Big Challenge

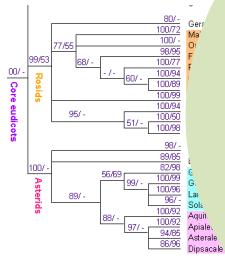
find and integrate all these data types:



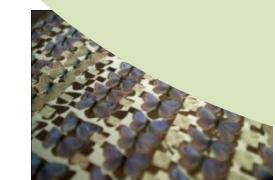
Field Inventory



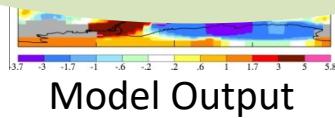
Satellite Data



iDiv



Collection Data



Functional Traits



Ecosystem Data

So, what is data integration?

Data integration is the process of consolidating data from a set of heterogeneous data sources into a single uniform data set or view on the data.

- The integrated data set should:
 1. Correctly and completely represent the content of all data sources.
 2. Use a single data model and a single schema.
 3. Only contain a single representation of every real-world entity.
 4. Not contain any conflicting data about single entities.
- To achieve this, data integration needs to **resolve various types of heterogeneity** that exist between data sources.

Outline

- ✓ Course outline and organization
- ✓ What is data integration
- ✓ Why is data integration important?
- Schema heterogeneity
- Goal of data integration, why it's a hard problem
- Data integration architectures

Enterprise data integration

Employees

FullTimeEmp

Hire

TempEmployees

Training

Courses

Enrollments

Sales

Products

Sales

Resumes

Interview
CV

Services

Services
Customers
Contracts

HelpLine

Calls

Employees

Employees
Hire

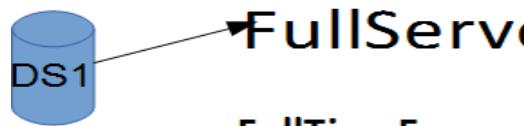
Credit Cards

Customer
CustDetail

Resumes
Interview

HelpLine
Calls

Examples of heterogeneity



FullTimeEmp

ssn, empld, firstName
middleName, lastName

Hire

empld, hireDate, recruiter

TempEmployees

ssn, hireStart, hireEnd



Employees

ID, firstNameMiddleInitial,
lastName

Hire

ID, hireDate, recruiter

HelpLine

Calls (date, agent, custID,
text, action)

HelpLine

Calls(date, agent, custID,
Description, followup)

File Home Insert Page Layout Formulas Data Review View Developer Load Test Acrobat Team

Record Macro Use Relative References Add-Ins COM Add-Ins Insert Design Mode View Code Properties View Code Source Map Properties Import Expansion Packs Export Refresh Data XML Document Panel Modify

N61

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | |
|----|----------|---------|------|-------|---------|------|------------|------|---------------|-----------|-----------------------|---------------------|------------------|--------|------------------|
| 1 | Transect | Section | Plot | Time | Heather | Fern | Blackberry | Tree | Tree Regrowth | Dead Wood | Litter, no vegetation | Moss, no vegetation | Bare Soil / Rock | Other | Comments |
| 71 | T003 | S018 | EI | 11:10 | 10 | | | | 40 | | 50 | | | | |
| 72 | T003 | S018 | EO | 11:10 | | 10 | | | 20 | | 70 | | | | |
| 73 | T003 | S018 | WI | 11:11 | 70 | 20 | | | | | 10 | | | | Farn (3 Knospen) |
| 74 | T003 | S018 | WO | 11:11 | 30 | | | | 10 | | 60 | | | | |
| 75 | T003 | S019 | EI | 11:13 | | | | | | | 100 | | | | |
| 76 | T003 | S019 | EO | 11:13 | 20 | 10 | | | 40 | | 30 | | | | |
| 77 | T003 | S019 | WI | 11:15 | 70 | | | | | | 30 | | | | |
| 78 | T003 | S019 | WO | 11:15 | 30 | | | | | | 70 | | | | |
| 79 | T003 | S020 | EI | 11:17 | | 20 | | | | | 80 | | | | |
| 80 | T003 | S020 | EO | 11:17 | | 20 | | | | | 80 | | | | |
| 81 | T003 | S020 | WI | 11:19 | 10 | 10 | | | | | 80 | | | | |
| 82 | T003 | S020 | WO | 11:19 | | 30 | | | | | 70 | | | | |
| 83 | T003 | S021 | EI | 11:27 | | | | | 10 | | 90 | | | | |
| 84 | T003 | S021 | EO | 11:27 | | | | | 20 | | 80 | | | | |
| 85 | T003 | S021 | WI | 11:28 | 80 | | | 20 | | | | 100 | | | |
| 86 | T003 | S021 | WO | 11:28 | | | | | | | | | | | |
| 87 | T003 | S022 | EI | 11:30 | | | | | 100 | | | | | | |
| 88 | T003 | S022 | EO | 11:30 | | | | | 50 | | 50 | | | | |
| 89 | T003 | S022 | WI | 11:32 | 50 | 10 | | | 10 | 10 | 20 | | | | |
| 90 | T003 | S022 | WO | 11:32 | | | | | | | 100 | | | | |
| 91 | T003 | S023 | EI | 11:33 | | 10 | | | | 20 | | 70 | | | |
| 92 | T003 | S023 | EO | 11:33 | | | | | | 30 | | 70 | | | |
| 93 | T003 | S023 | WI | 11:35 | 20 | 20 | | | | | 60 | | | | |
| 94 | T003 | S023 | WO | 11:35 | | | | | | | 100 | | | | |
| 95 | T003 | S024 | EI | 11:37 | | | | | 30 | | 50 | | 20* | *Gras | |
| 96 | T003 | S024 | EO | 11:37 | | 10 | | | | | 90 | | | | |
| 97 | T003 | S024 | WI | 11:40 | | | | | | 40 | | 60 | | | |
| 98 | T003 | S024 | WO | 11:40 | | 20 | | | | | 50 | | 30* | * Gras | |
| 99 | T003 | S025 | EI | 11:43 | | | | | 80 | | | 20 | | | |

File Home Insert Page Layout Formulas Data Review View Developer Load Test Acrobat Team

Record Macro Use Relative References Add-Ins COM Add-Ins Insert Design Mode View Code Properties View Code Source Map Properties Import Expansion Packs Export Refresh Data XML Document Panel Modify

A163

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | |
|-----|----------|---------|------|-------|---------|------|------------|------|---------------|-----------|-----------------------|---------------------|------------------|-------|----------|
| 1 | Transect | Section | Plot | Time | Heather | Fern | Blackberry | Tree | Tree Regrowth | Dead Wood | Litter, no vegetation | Moss, no vegetation | Bare Soil / Rock | Other | Comments |
| 92 | T003 | S023 | EO | 11:33 | | | | | | 20 | | 80 | | | |
| 93 | T003 | S023 | WI | 11:35 | 10 | 20 | | | | | | 70 | | | |
| 94 | T003 | S023 | WO | 11:35 | | | | | | | | 100 | | | |
| 95 | T003 | S024 | EI | 11:37 | | | | | 60 | | | | | | |
| 96 | T003 | S024 | EO | 11:37 | | | | | | | | 40 | | | |
| 97 | T003 | S024 | WI | 11:40 | | | | | | | 50 | | | | |
| 98 | T003 | S024 | WO | 11:40 | | 20 | | | | | 10 | | 60 | | |
| 99 | T003 | S025 | EI | 11:43 | | 10 | | | | | | 60 | | | |
| 100 | T003 | S025 | EO | 11:43 | | | | | 30 | 30 | | | | | |
| 101 | T003 | S025 | WI | 11:45 | | 10 | | | | 20 | | | | | |
| 102 | T003 | S025 | WO | 11:45 | | | | | | | 40 | | | | |
| 103 | T003 | S026 | EI | 11:49 | | | | | 50 | 30 | | | | | |
| 104 | T003 | S026 | EO | 11:49 | | | | | 80 | | | | | | |
| 105 | T003 | S026 | WI | 11:51 | | 10 | | | 10 | | 70 | | | | |
| 106 | T003 | S026 | WO | 11:52 | | 20 | | | | | 20 | | | | |
| 107 | T003 | S027 | EI | 12:48 | | 20 | | | 50* | | 20 | | | | |
| 108 | T003 | S027 | EO | 12:48 | 20 | | | | | 50 | | | | | |
| 109 | T003 | S027 | WI | 12:51 | | 20 | | | 20 | 10 | 30 | | | | |
| 110 | T003 | S027 | WO | 12:51 | | | | | | 20 | 10 | | | | |
| 111 | T003 | S028 | EI | 12:54 | | | | | | | | 100 | | | |
| 112 | T003 | S028 | EO | 12:54 | 10 | | | | | | | 70 | | | |
| 113 | T003 | S028 | WI | 12:56 | | | | | | | | 50 | | | |

Outline

- ✓ Course outline and organization
- ✓ What is data integration
- ✓ Why is data integration important?
- ✓ Schema heterogeneity
- Goal of data integration, why it's a hard problem
- Data integration architectures

Goal of data integration

- Uniform query access to a set of data sources
- Handle:
 - Scale of sources: from tens to millions
 - Heterogeneity
 - Autonomy
 - Semi-structure

Why is it hard?

- Systems-level reasons:
 - Managing different platforms
 - SQL across multiple systems is not so simple
 - Distributed query processing
- Logical reasons:
 - Schema (and data) heterogeneity
- ‘Social’ reasons:
 - Locating and capturing relevant data in the enterprise.
 - Convincing people to share (data fiefdoms)
 - ❖ Security, privacy and performance implications.

Setting expectations

Data integration is **AI-Complete**.

- Completely automated solutions unlikely.

Goal 1:

- Reduce the effort needed to set up an integration application.

Goal 2:

- Enable the system to perform gracefully with uncertainty (e.g., on the web)

Data integration smorgasbord

Something for everyone:

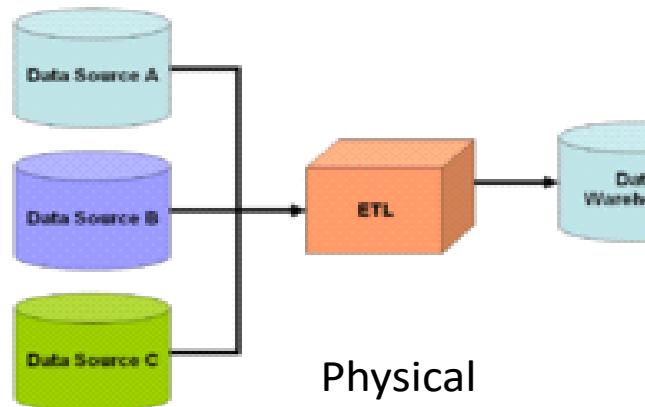
- **Theory** of modeling data sources
- **Systems** aspects of data integration
- **Architectural** issues: e.g., P2P data sharing
- **AI @ work**: automated schema matching
- **Web**: latest on data integration & web
- **Commercial** products: BEA, IBM
- **Semantic Web**: what does it have to offer?
- New trends in DBMS: **uncertainty, dataspaces**

Outline

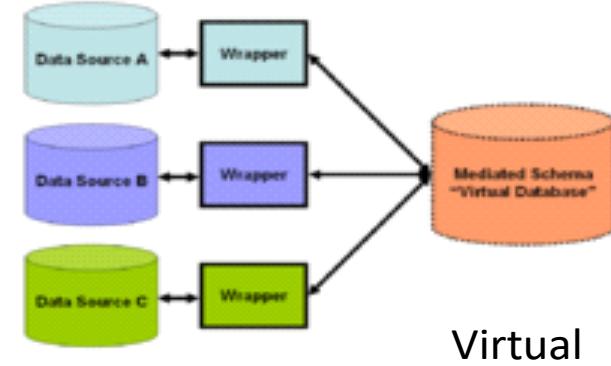
- ✓ Introduction: data integration as a new abstraction
- ✓ Examples of data integration applications
- ✓ Schema heterogeneity
- ✓ Goal of data integration, why it's a hard problem
- Data integration architectures
- Course Outline and Organization

Data integration architecture

- Data integration approaches:
 - physical data integration ([warehousing](#))
 - virtual data integration ([mediated schema](#))



Physical

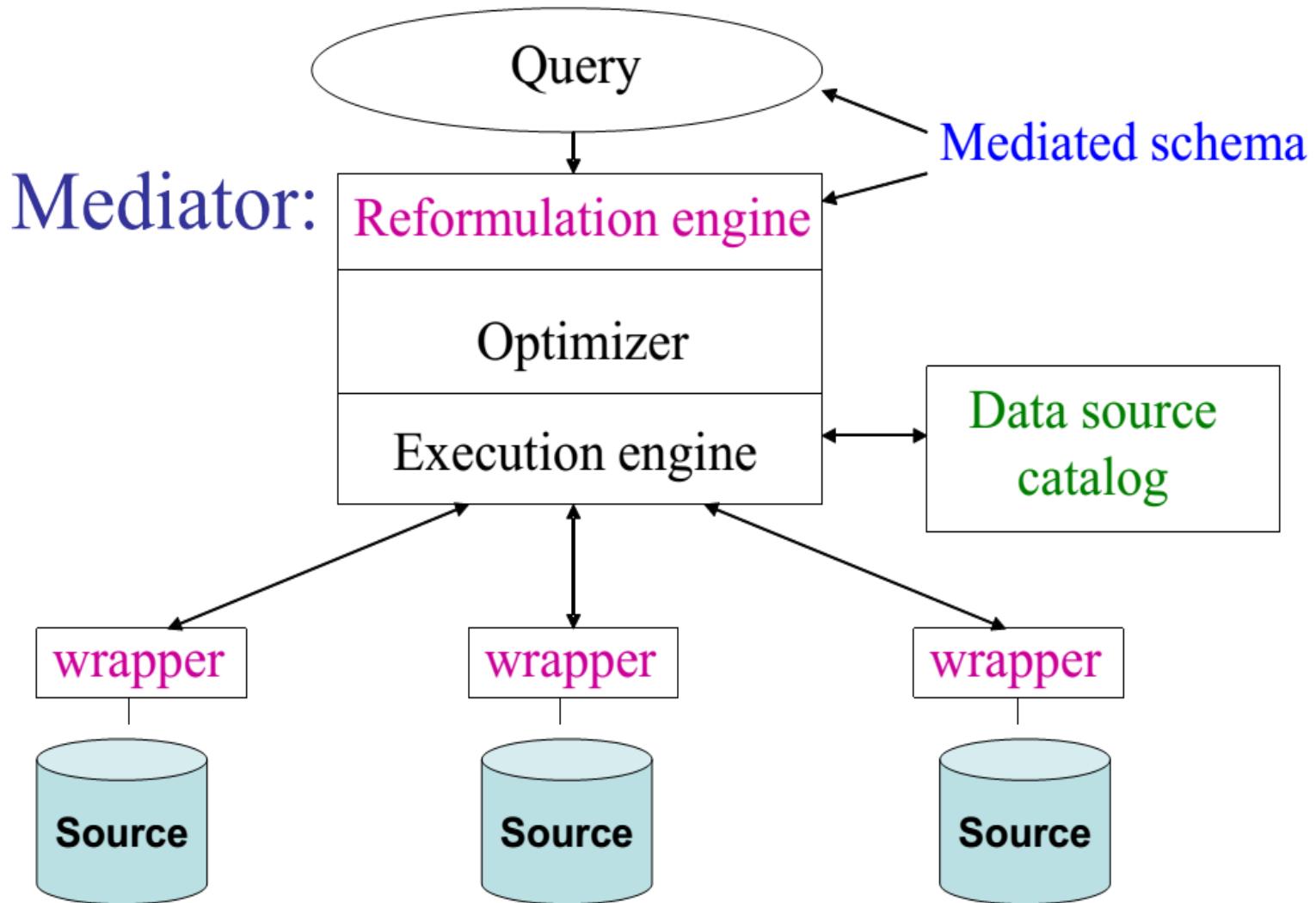


Virtual

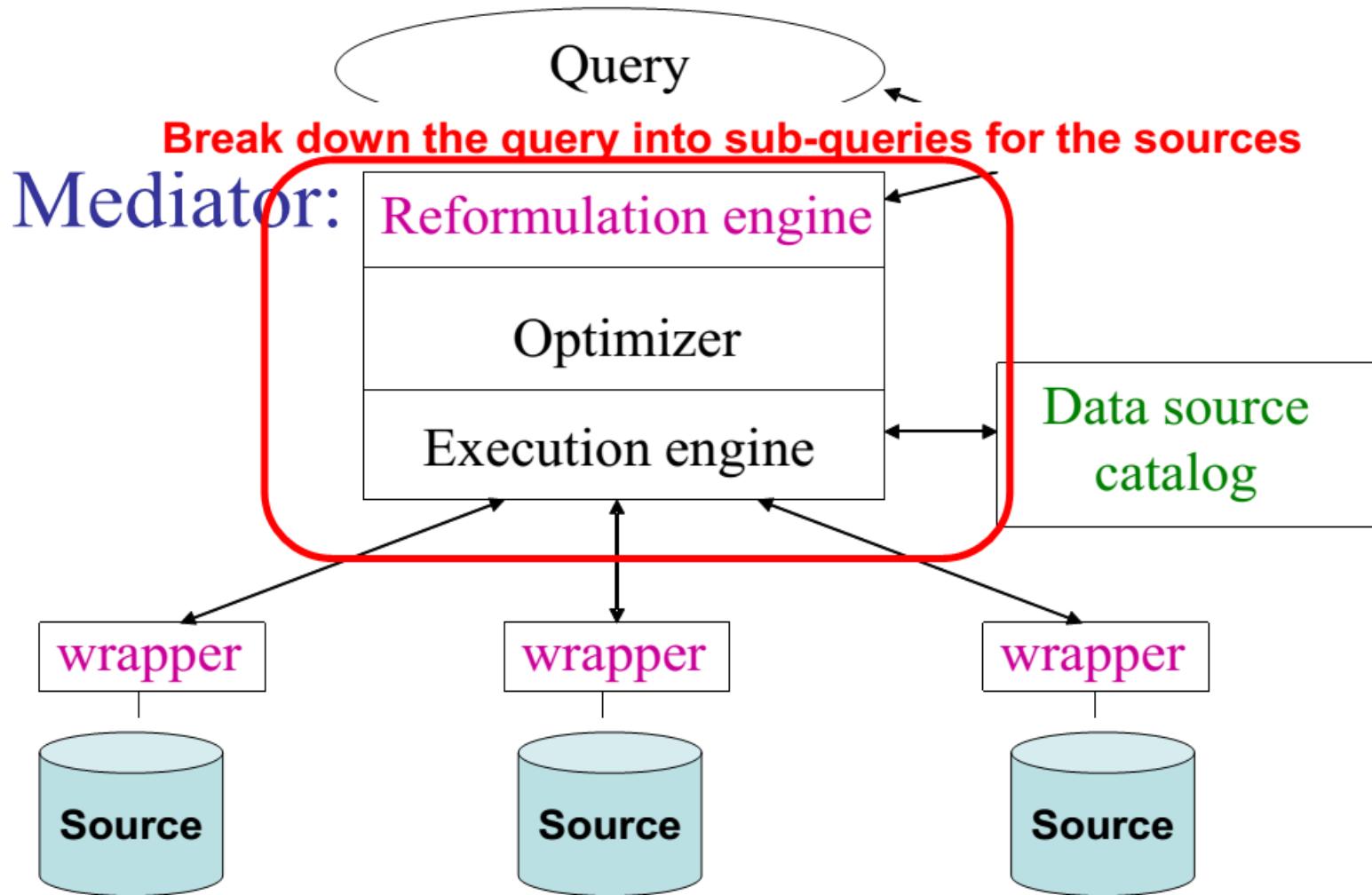
Virtual data integration

- Leave the data in local sources
- When a query comes in:
 - Determine the relevant sources to the query
 - Break down the query into sub-queries for the sources
 - Get the answers from the sources, and combine them appropriately
- Data is fresh

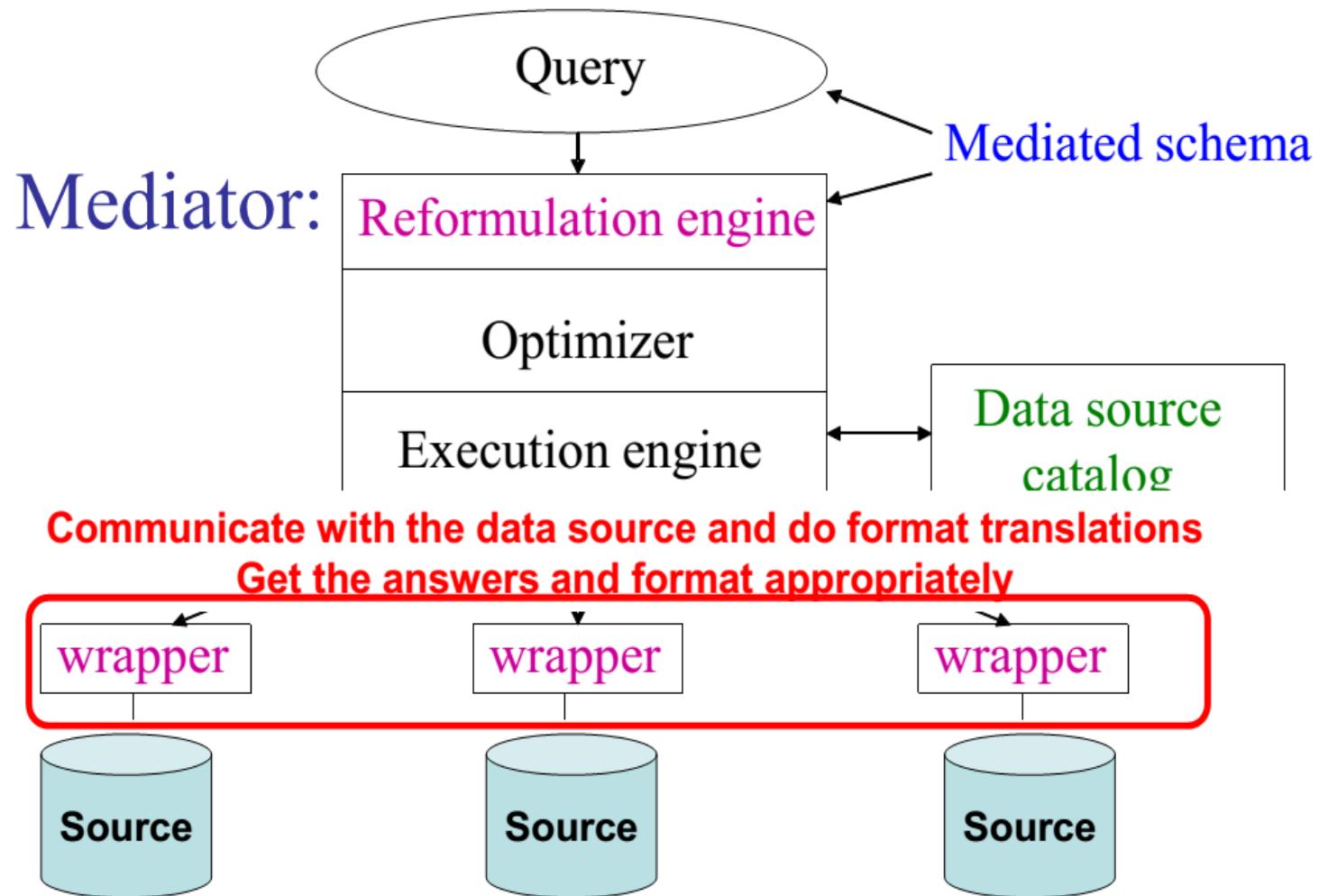
Virtual data integration architecture



Virtual data integration architecture

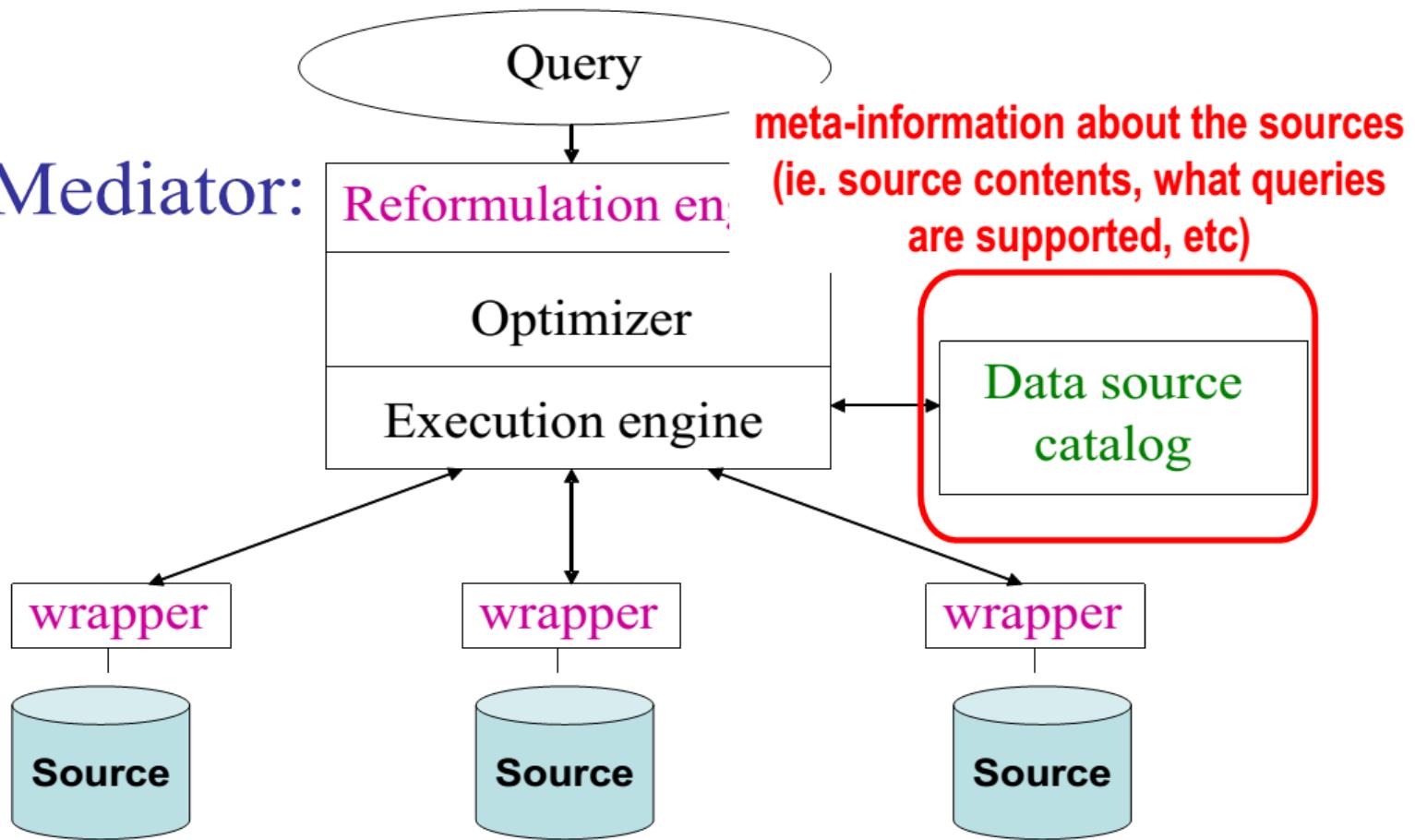


Virtual data integration architecture



Virtual data integration architecture

Mediator:



Virtual data integration: An example

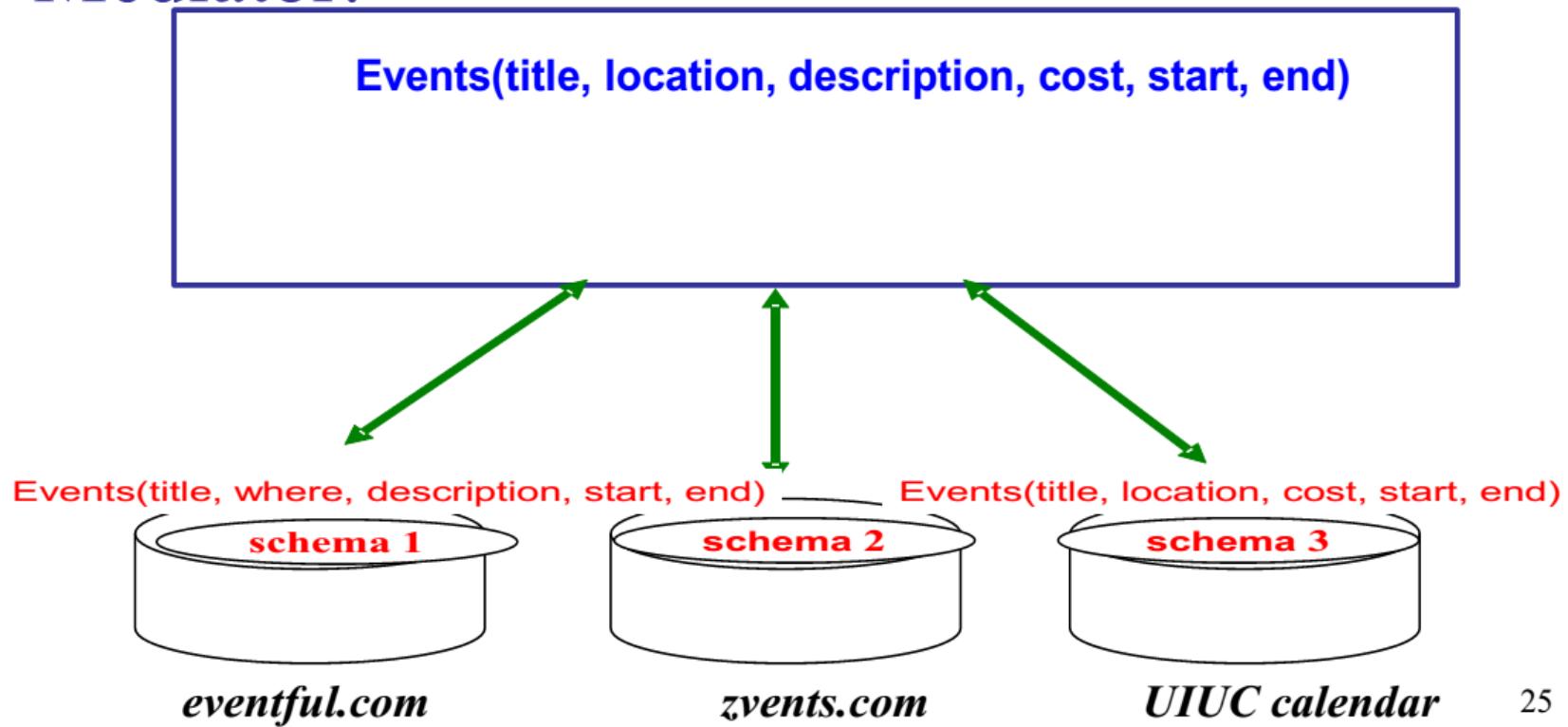
Mediator:

Events(title, location, description, cost, start, end)

Virtual data integration: An example

Find upcoming events in Champaign-Urbana

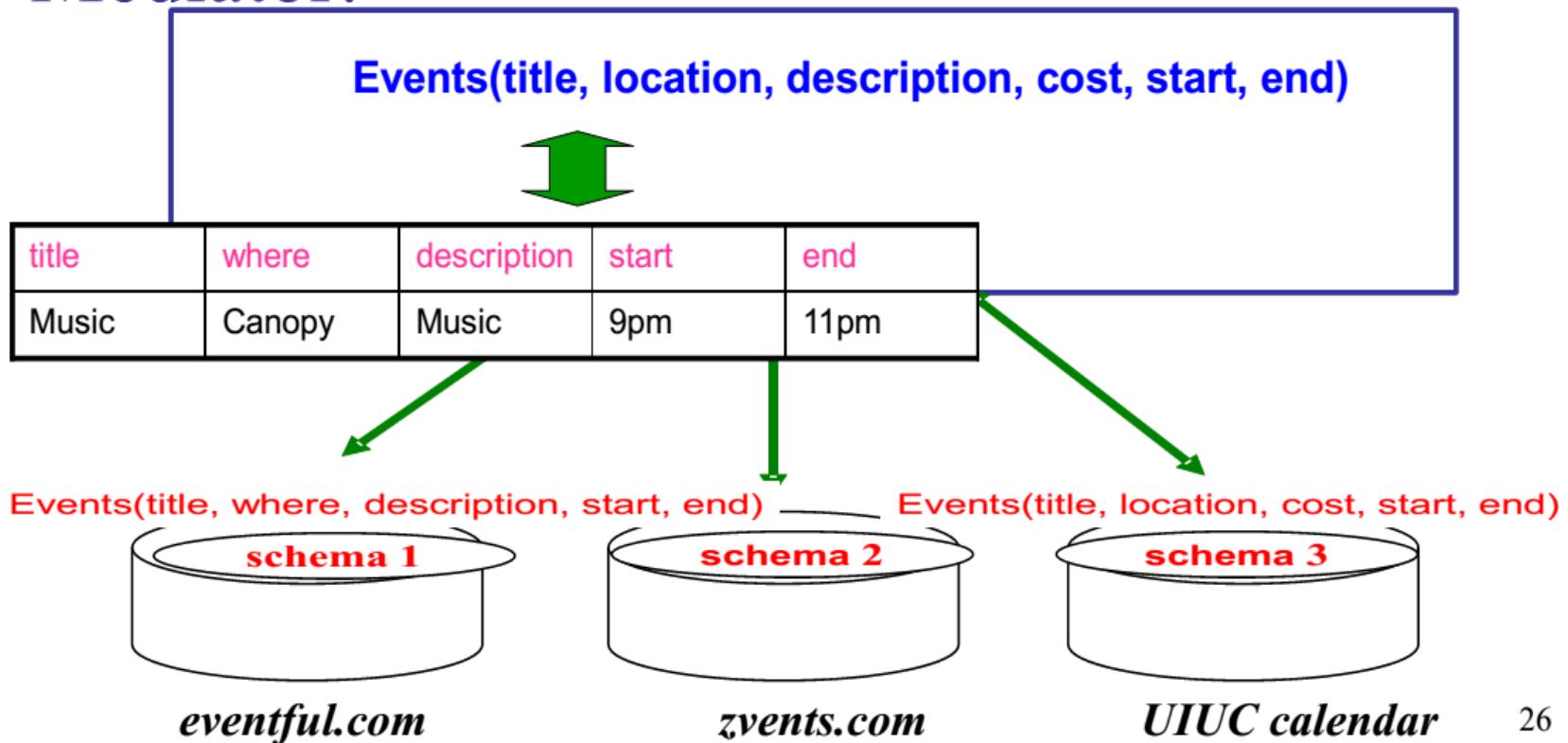
Mediator:



Virtual data integration: An example

Find upcoming events in Champaign-Urbana

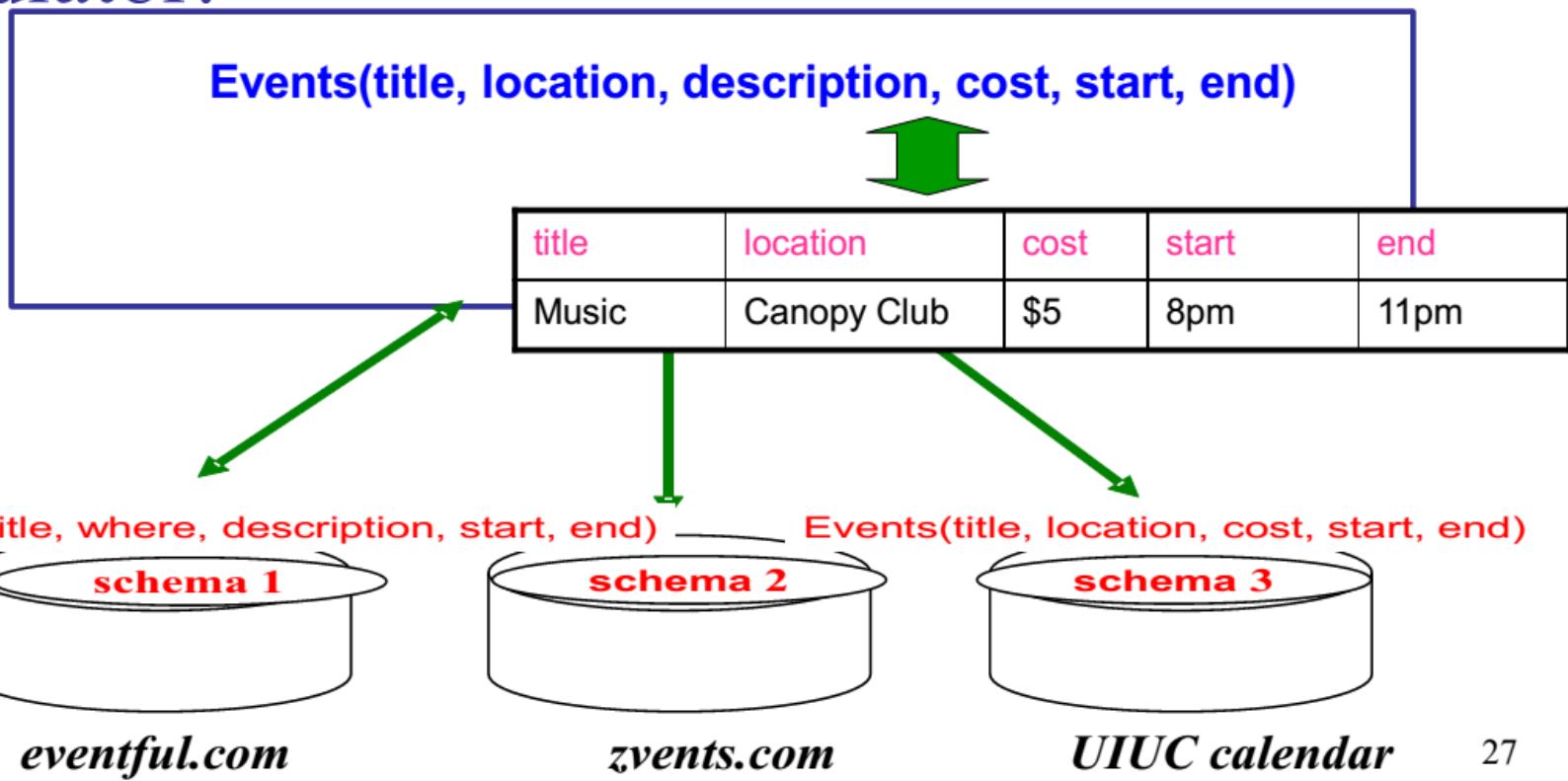
Mediator:



Virtual data integration: An example

Find upcoming events in Champaign-Urbana

Mediator:



Virtual data integration: The challenge

Data integration: is the process of combining data from different data sources and representing them in a unified form

| title | where | description | start | end |
|-------|--------|-------------|-------|------|
| Music | Canopy | Music | 9pm | 11pm |

| title | location | Cost | start | end |
|-------|-------------|------|-------|------|
| Music | Canopy Club | \$5 | 8pm | 11pm |

Virtual data integration: The challenge

Data Integration: the process of combining data from different data sources and presenting a unified view of these data

| title | location | description | cost | start | end |
|-------|----------|-------------|------|-------|-----|
| | | | | | |



Events(title, location, description, cost, start, end)



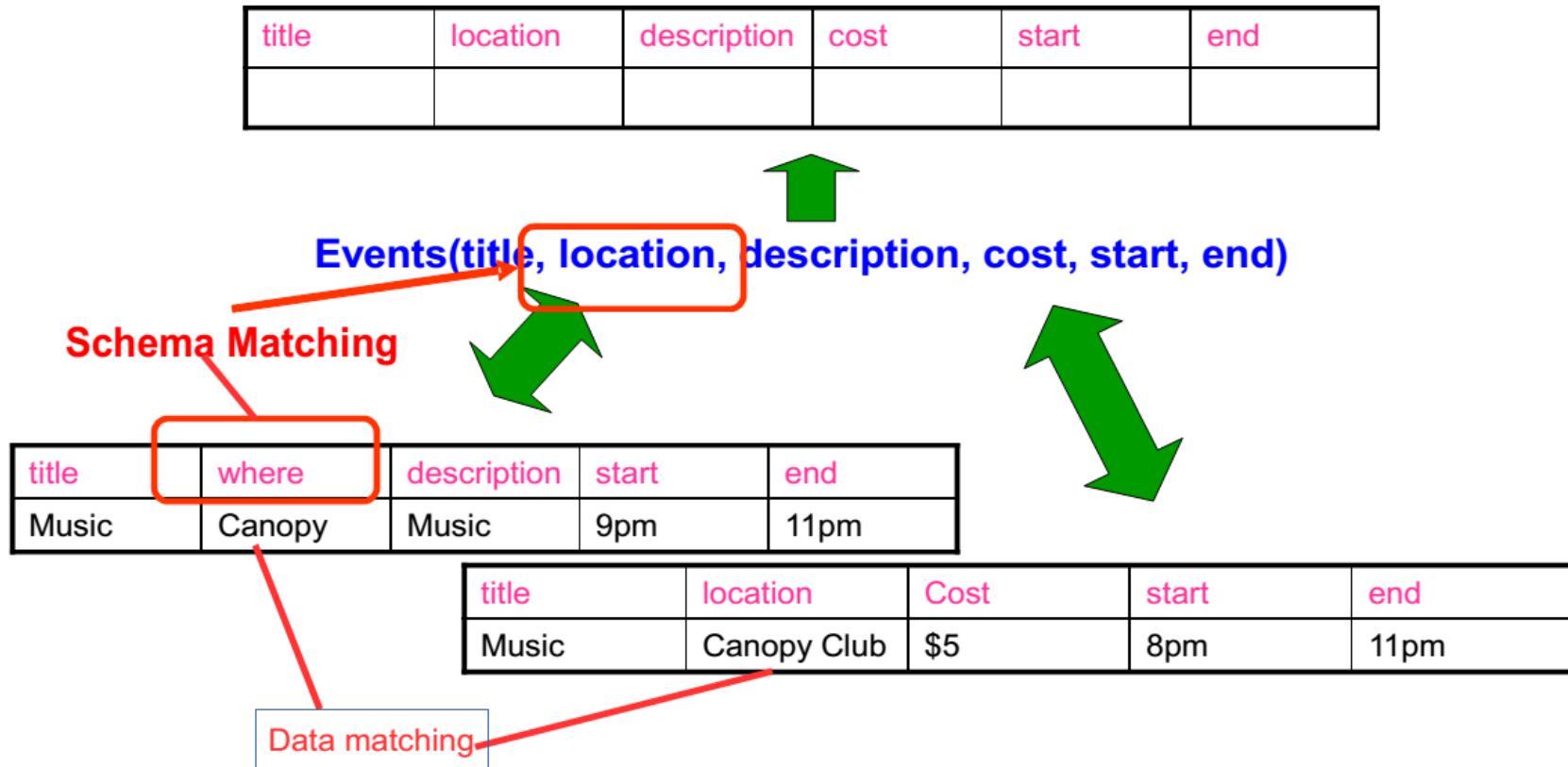
| title | where | description | start | end |
|-------|--------|-------------|-------|------|
| Music | Canopy | Music | 9pm | 11pm |



| title | location | Cost | start | end |
|-------|-------------|------|-------|------|
| Music | Canopy Club | \$5 | 8pm | 11pm |

Virtual data integration: The challenge

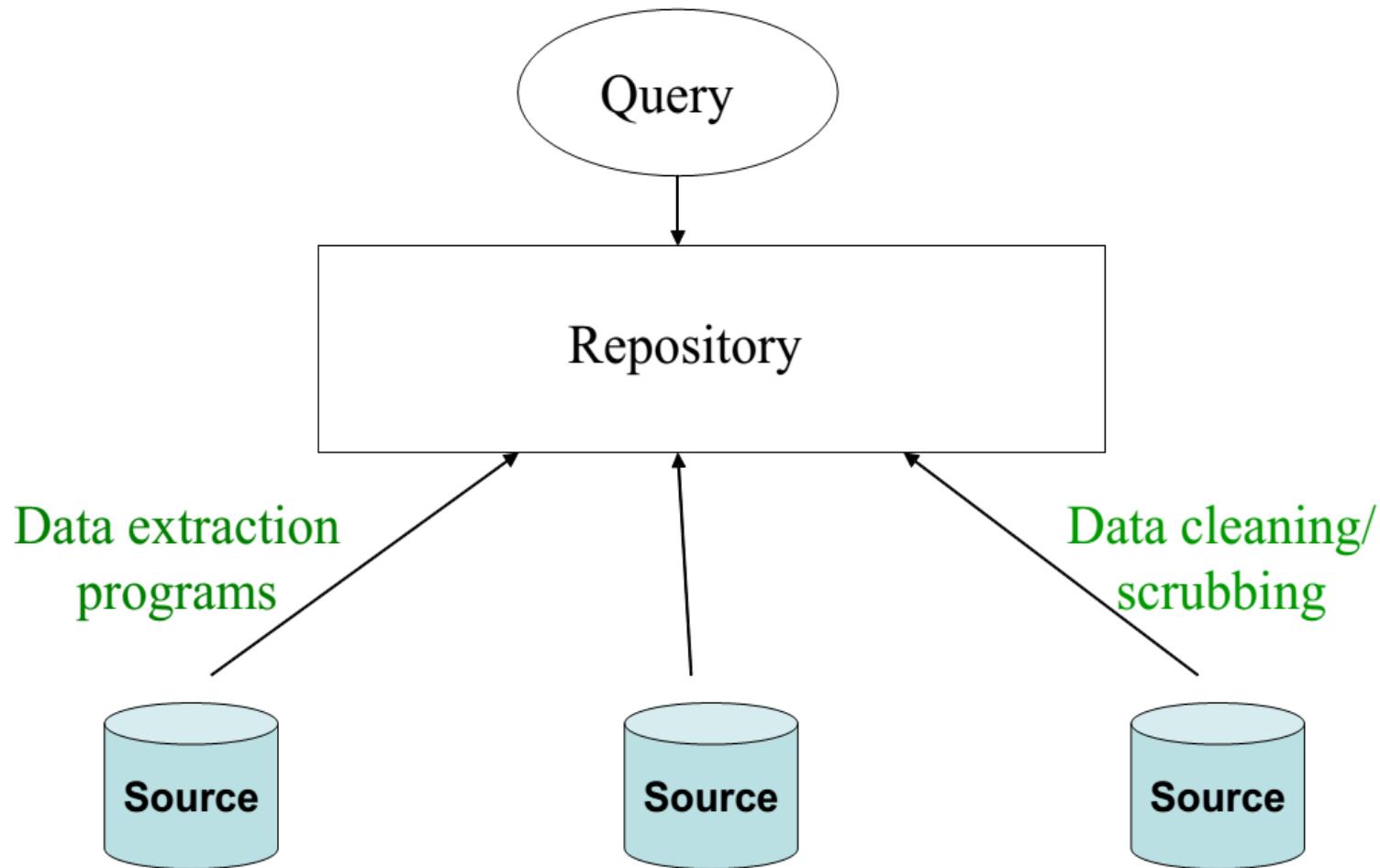
Data Integration: the process of combining data from different data sources and presenting a unified view of these data



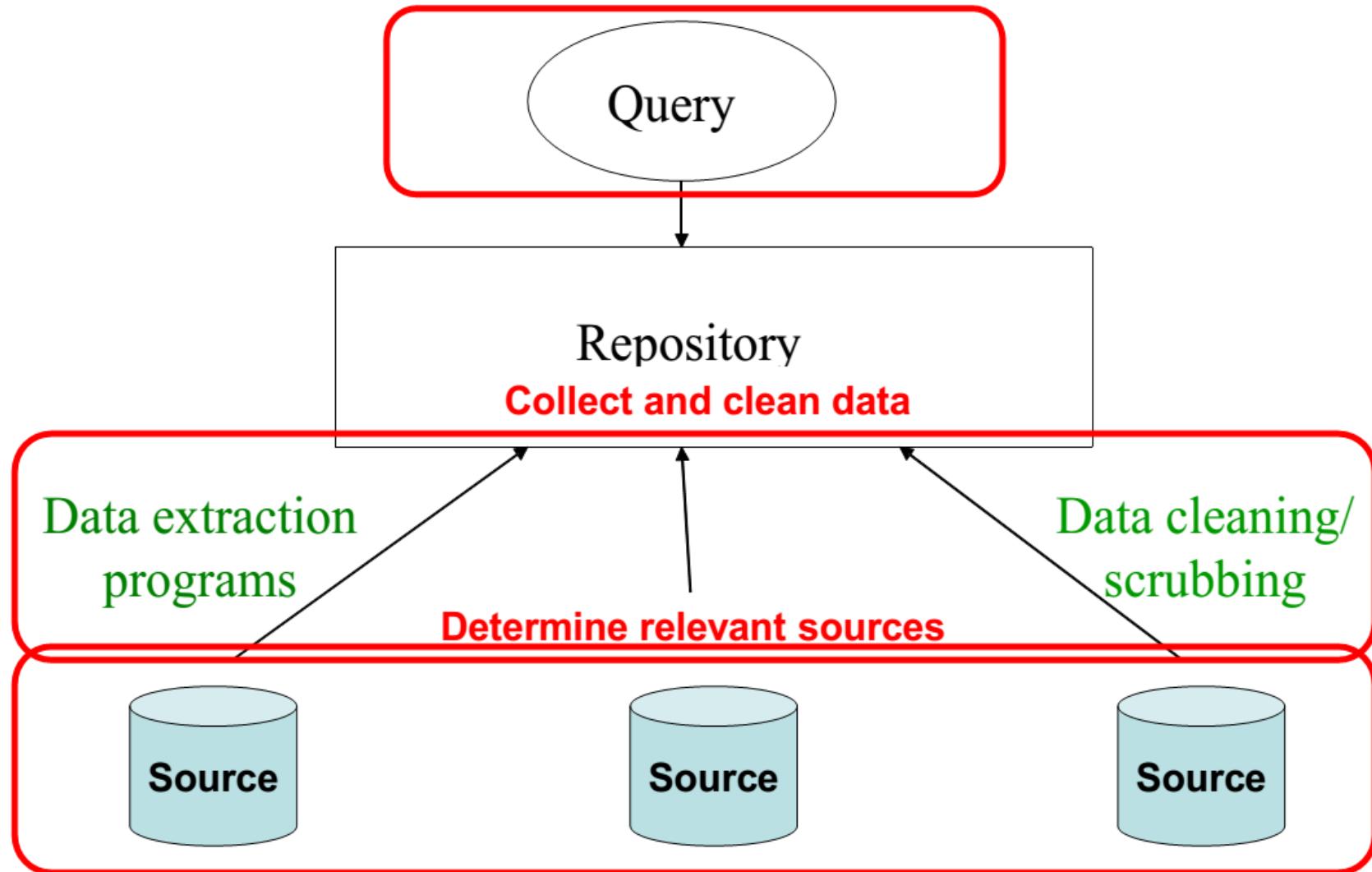
Physical data integration: Data warehouse

- Load all the data periodically into a central database (warehouse)
 - Performance is good
 - Data may not be fresh
 - Need to clean, scrub your data

Physical data integration: Data warehouse



Physical data integration: Data warehouse



Physical data integration: Data warehouse

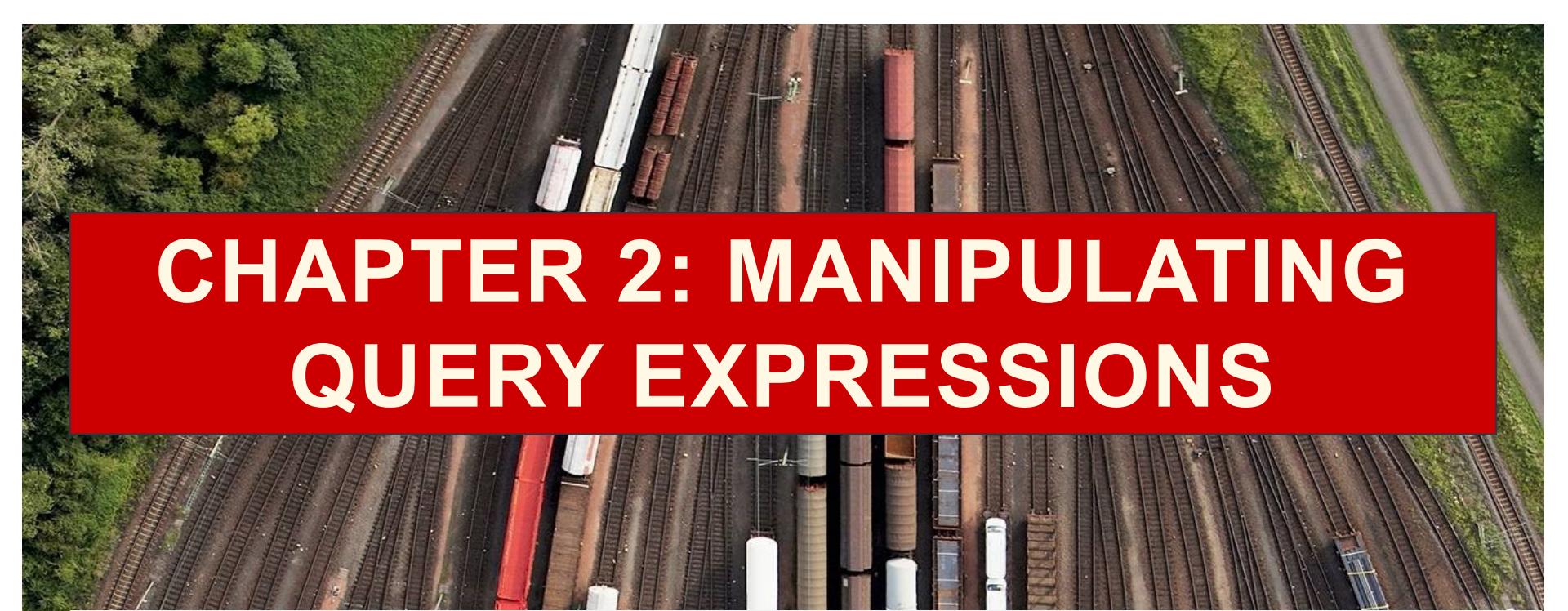
- Virtual integration
 - No data are collected offline
 - On a search, data are collected and processed from various sources at runtime
- Data warehouse
 - Data are collected offline and stored in a central repository
 - Search is performed on the repository
- When should we take the virtual integration approach?
- When should we take the warehousing approach?

Summary of Chapter 1

- Data integration: abstract away the fact that data comes from multiple sources in varying schemata.
- Problem occurs everywhere: it's key to business, science, Web and government.
- Goal: reduce the effort involved in integrating.
- Regardless of the architecture, heterogeneity is a key issue.
- Architectures range from warehousing to virtual integration.

Literature

- AnHai Doan, Alon Halevy, Zachary Ives: **Principles of Data Integration**. Morgan Kaufmann, 2012.
- Ulf Leser, Felix Naumann: **Informationsintegration**. DBunkt Verlag, 2007. (<http://www.teletask.de/archive/series/overview/892/>)
- Xin Luna Dong, Divesh Srivastava: **Big Data Integration**, Morgan & Claypool, 2015
- Jérôme Euzenat, Pavel Shvaiko: **Ontology Matching**. Springer, 2014.
- Felix Naumann: **An Introduction to Duplicate Detection**. Morgan & Claypool, 2012.
- M. Tamer Özsu and Patrick Valduriez: **Principles of Distributed Database Systems**. Third Edition, 2011



CHAPTER 2: MANIPULATING QUERY EXPRESSIONS



PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

Introduction

- How does a data integration system decide which sources are relevant to a query? Which are redundant? How to combine multiple sources to answer a query?
- Answer: by reasoning about the contents of data sources.
 - Data sources are often described by queries / views.
- This chapter describes the fundamental tools for manipulating query expressions and reasoning about them.

Outline

- Review of basic database concepts
- Query unfolding
- Query containment and equivalence
- Answering queries using views

Basic database concepts

- Relational data model
- Integrity constraints
- Queries and answers
- Conjunctive queries

Basic database concepts

- A **database (DB)** is an organized collection of related data.
- A **database management system (DBMS)** is a suite of computer programs designed to manage a database and run operations on the data requested by numerous clients.
- A **database system (DBS)** is the concrete instance of a database managed by a database management system.

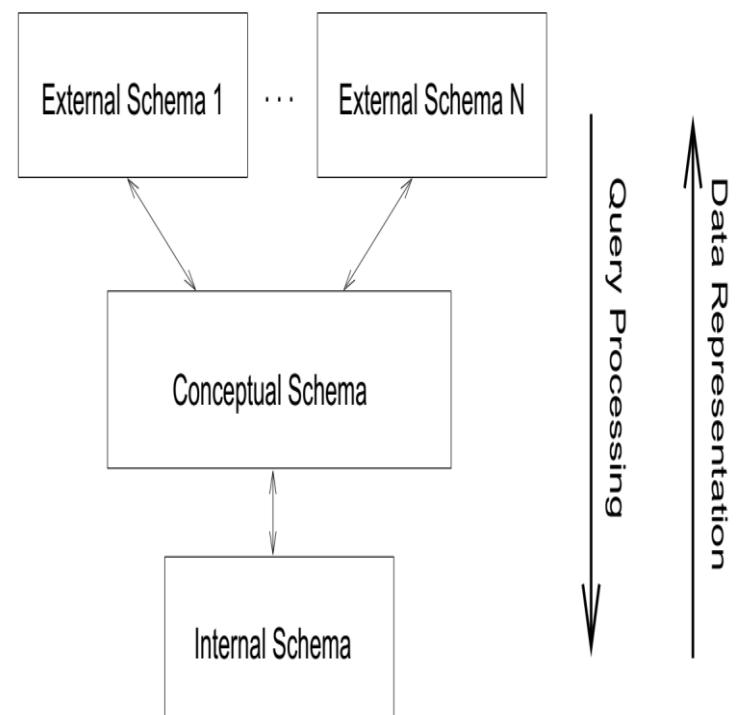
Data independency

- **Logical data independence:**

Changes to the logical schema level must not require a change to an application (external schema) based on the structure.

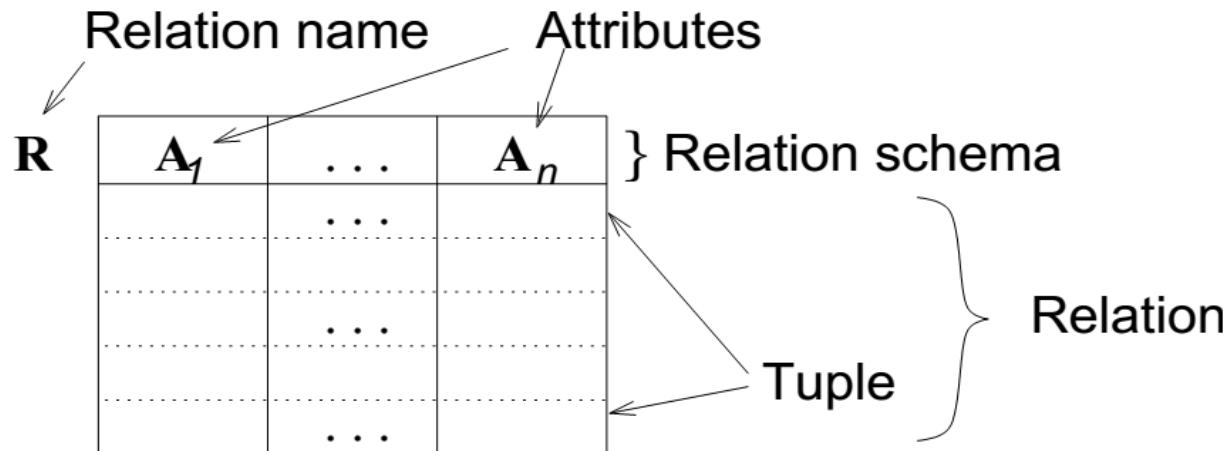
- **Physical data independence:**

Changes to the physical schema level (how data is stored) must not require a change to the logical schema.



Relation model

A **relational database** is a database that is structured according to the relational database model. It consists of a set of relations.



Employee: emplID, name, hireDate, manager

Interview: candidate, date, recruiter, hireDecision, grade

EmployeePerf: emplID, name, reviewQuarter, grade, reviewer

Interview

| <i>Candidate</i> | <i>Date</i> | <i>Recruiter</i> | <i>hierDecision</i> | <i>grade</i> |
|------------------|-------------|------------------|---------------------|--------------|
| John Smith | 08.10.2018 | | Accepted | 1.5 |
| Lin Yi | 10.10.2018 | | Not | 3.5 |
| Rolle Sing | 11.10.2018 | | Not | 2.5 |

Table/relation name

Attribute names

Tuples or rows

Notations

- A database state (instance) **D**
- Attributes with letters from the beginning of the Alphabet, e.g. **A, B, C, ...** And sets or lists of attributes \overline{A}
- Relation names with letter **R, S, T**, and sets of relations T
- Tuples with lowercase letters e.g., s, t

Integrity constraints

- Integrity constraints describe valid tuples of a relation
 - *Primary key* constraint
 - *Foreign key* constraint (referential integrity)
 - *Value range* constraints
 - ...
- A *key* is a set of columns that uniquely determine a row in the database:
 - There do not exist two tuples, t_1 and t_2 such that $t_1 \neq t_2$ and t_1 and t_2 have the same values for the key columns.
 - (*EmplID*, *reviewQuarter*) is a key for **EmployeePerf**

IC: Functional dependencies

- A set of attribute **A** functionally determines a set of attributes **B** if: whenever t_1 and t_2 agree on the values of **A**, they must also agree on the values of **B**.
 - For example, (EmpID, reviewQuarter) functionally determine (grade).
- Note: a key dependency is a functional dependency where the key determines all the other columns.

IC: Foreign keys

- Given table **T** with key **B** and table **S** with key **A**:
 - **A** is a foreign key of **B** in **T** if whenever a **S** has a row where the value of **A** is **v**, then **T** must have a row where the value of **B** is **v**.
- Example: the *empID* attribute of **EmployeePerf** is a foreign key for attribute *emp* of **Employee**.

Employee: empID, name, hireDate, manager

Interview: candidate, date, recruiter, hireDecision, grade

EmployeePerf: emplID, name, reviewQuarter, grade, reviewer

Relational algebra

- A **relational algebra** is a set of operations that are **closed** over relations.
 - Each operation has one or more relations as input
 - The output of each operation is a relation

Primitive operations:

- Selection σ
- Projection π
- Cartesian product (cross product) \times
- Set union \cup
- Set difference $-$
- Rename β

Non-primitive operations

- Natural Join \bowtie
- θ -Join and Equi-Join \bowtie_φ
- Semi-Join \ltimes
- Outer-Joins $= \times$
- Set intersection \cap
- ...

Cartesian product

| R | |
|---|---|
| A | B |
| 1 | 2 |
| 3 | 4 |

$$r(R) \times r(S)$$

| S | | |
|----|----|----|
| C | D | E |
| 5 | 6 | 7 |
| 8 | 9 | 10 |
| 11 | 12 | 13 |

| A | B | C | D | E |
|---|---|----|----|----|
| 1 | 2 | 5 | 6 | 7 |
| 1 | 2 | 8 | 9 | 10 |
| 1 | 2 | 11 | 12 | 13 |
| 3 | 4 | 5 | 6 | 7 |
| 3 | 4 | 8 | 9 | 10 |
| 3 | 4 | 11 | 12 | 13 |

The Natural Join Operation

| R | |
|---|---|
| A | B |
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

$r(R) \bowtie r(S)$

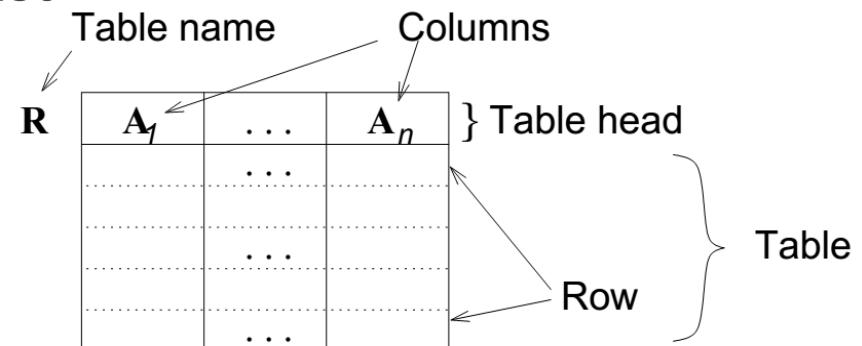
| A | B | C | D |
|---|---|---|---|
| 3 | 4 | 5 | 6 |
| 5 | 6 | 7 | 8 |

| S | | |
|---|---|----|
| B | C | D |
| 4 | 5 | 6 |
| 6 | 7 | 8 |
| 8 | 9 | 10 |

- Combine tuples from two relations $r(R)$ and $r(S)$ where for
 - all attributes $a = R \cap S$ (defined in both relations)
 - is $t(R.a) = t(S.a)$.

SQL data model

- Said to implement relational database model
- Defines own terms
- Some significant differences exist



- **Structured Query Language (SQL)**: declarative language to describe requested query results
- Realizes relational operations (with the mentioned discrepancies)
- Basic form: *SELECT-FROM-WHERE*-block (SFW)

SQL (very basic)

Interview:

candidate, date, recruiter, hireDecision, grade

EmployeePerf:

emplID, name, reviewQuarter, grade, reviewer

```
select recruiter, candidate  
from Interview, EmployeePerf  
where recruiter=name AND  
EmployeePerf.grade < 2.5
```

Query answers

- $Q(D)$: the **set** (or **multi-set**) of rows resulting from applying the query Q on the database D .
- Unless otherwise stated, we will consider **sets** rather than **multi-sets**.

SQL (w/aggregation)

EmployeePerf:

emplD, name, reviewQuarter, grade, reviewer

```
select reviewer, Avg(grade)
from EmployeePerf
where reviewQuarter=“1/2007”
```

Conjunctive queries

- Conjunctive queries are the simplest form of queries that can be expressed over a database.
- There are many ways to define a conjunctive query, and we will do it from a logical perspective first, using Datalog notation

Conjunctive queries

$$Q(\bar{X}) :- R_1(\bar{X}_1), R_2(\bar{X}_2), \dots, R_n(\bar{X}_n), c_1, \dots, c_m$$

head of the query

body of the query (each called
sub-goal or conjunction)

where:

- \bar{X} head variables
- \bar{X}_i existential variables
- c_j interpreted atoms $X\theta Y$

Conjunctive queries

```
select recruiter, candidate  
from Interview, EmployeePerf  
where recruiter=name AND  
      EmployeePerf.grade < 2.5
```

Q(X,T) :-

Interview(X,D,**Y**,H,F), EmployeePerf(E,**Y**,T,W,Z), W < 2.5.

Joins are expressed with multiple occurrences
of the same variable

Conjunctive queries (negated subgoals)

Q(X,T) :-

Interview(X,D,**Y**,H,F), EmployeePerf(E,**Y**,T,W,Z),
¬OfferMade(X, date).

Safety: every head variable must appear in a positive subgoal.

Unions of conjunctive queries

Multiple rules with the same head predicate
express a union

Q(R,C) :-

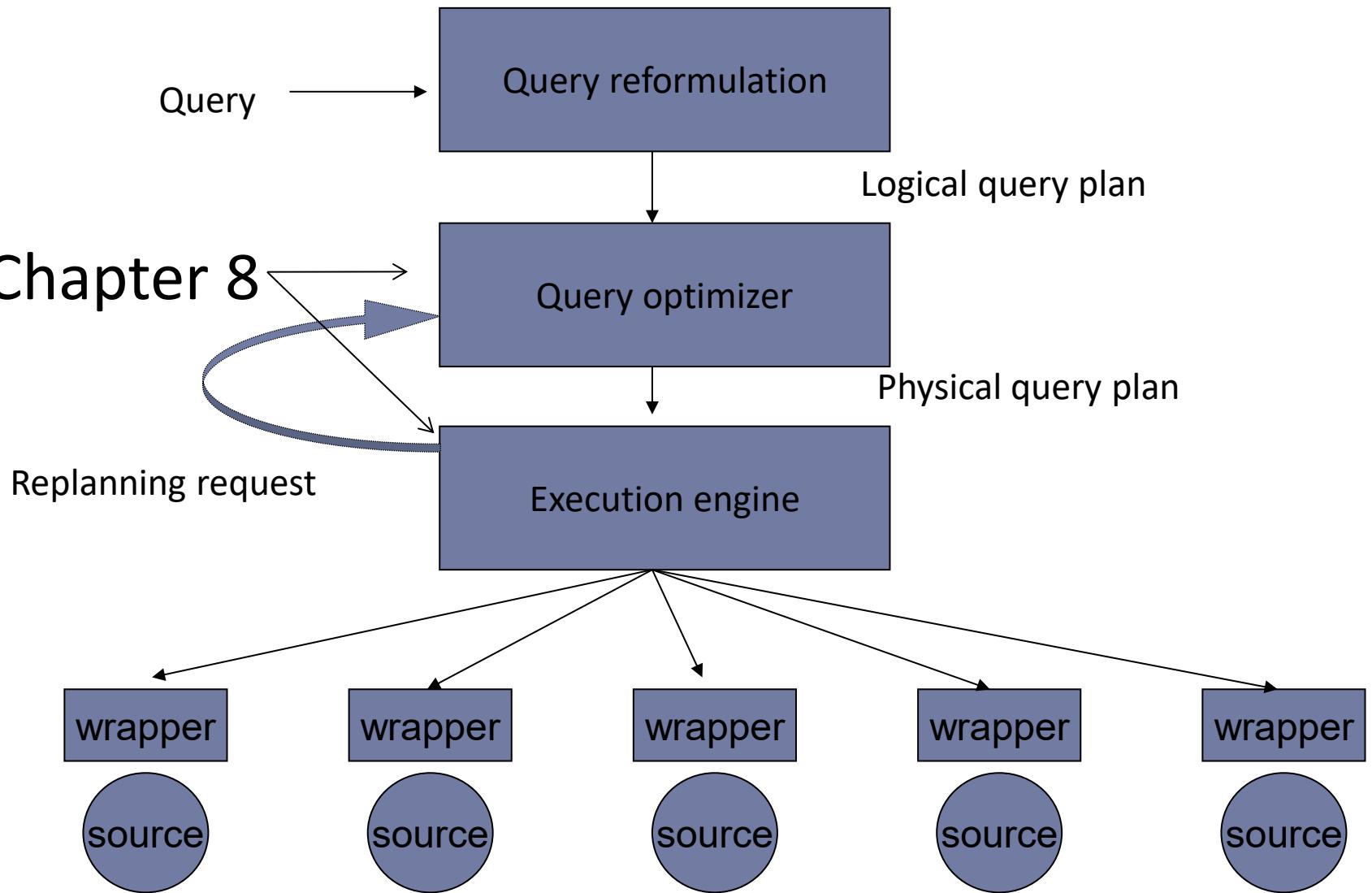
Interview(X,D,**Y**,H,F), EmployeePerf(E,**Y**,T,W,Z),
W < 2.5.

Q(R,C) :-

Interview(X,D,**Y**,H,F), EmployeePerf(E,**Y**,T,W,Z), Manager(y),
W > 3.9.

Query processing

Chapter 8



Outline

- ✓ Review of basic database concepts
- Query unfolding
- Query containment and equivalence
- Answering queries using views

Query unfolding

- **Compositionality**: we can write queries that refer to views (named queries) in their body
 - Advantage of declarative languages
- **Query unfolding**: process of undoing the composition of queries
 - Given a query that refer to views, query unfolding rewrites the query so that it refers database tables

Query unfolding

- *Query composition* is an important mechanism for writing complex queries.
 - Build query from views in a bottom up fashion.
- Query unfolding “unwinds” query composition.
- Important for:
 - Comparing between queries expressed with views
 - Query optimization (to examine all possible join orders)
 - Unfolding may even discover that the composition of two satisfiable queries is unsatisfiable!

Query unfolding: Example

- Given the following two relations
 - *Flight (dept, arrival)*; stores pairs of cities between there is a direct flight
 - *Hub(city)*; stores the set of hub cities of the airline
 - Query Q1 asks for pairs of cities between which there is a flight that goes through a hub
 - Query Q2 asks for pairs of cities that are on the same outgoing path from a hub

$$Q_1(X, Y) : - \text{Flight}(X, Z), \text{Hub}(Z), \text{Flight}(Z, Y)$$
$$Q_2(X, Y) : - \text{Hub}(Z), \text{Flight}(Z, X), \text{Flight}(X, Y)$$
$$Q_3(X, Z) : - Q_1(X, Y), Q_2(Y, Z)$$

- The unfolding of Q3 is:

$$Q'_3(X, Z) : - \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y), \text{Hub}(W), \text{Flight}(W, Y), \text{Flight}(Y, Z)$$

Query unfolding algorithm

- Find a subgoal $p(X_1, \dots, X_n)$ such that p is defined by a rule r .
- Unify $p(X_1, \dots, X_n)$ with the head of r .
- Replace $p(X_1, \dots, X_n)$ with the result of applying the unifier to the subgoals of r (use fresh variables for the existential variables of r).
- Iterate until no unifications can be found.
- If p is defined by a union of r_1, \dots, r_n , create n rules, for each of the r 's.

Query unfolding summary

- Unfolding does not necessarily create a more efficient query!
 - Just lets the optimizer explore more evaluation strategies.
 - Unfolding is the opposite of rewriting queries using views (see later).
- The size of the resulting query can grow exponentially

Outline

- ✓ Review of basic database concepts
- ✓ Query unfolding
- Query containment and equivalence
- Answering queries using views

Query containment: Motivation (1)

Intuitively, the unfolding of \mathbf{Q}_3 is *equivalent* to \mathbf{Q}_4 :

$$\begin{aligned} Q'_3(X, Z) :- & \neg Flight(X, U), Hub(U), Flight(U, Y), \\ & Hub(W), Flight(W, Y), Flight(Y, Z) \end{aligned}$$
$$\begin{aligned} Q_4(X, Z) :- & \neg Flight(X, U), Hub(U), Flight(U, Y), \\ & Flight(Y, Z) \end{aligned}$$

How can we justify this intuition formally?

Query containment: Motivation (2)

Furthermore, the query Q_5 that requires going through two hubs is *contained* in Q'_3

$$Q_5(X, Z) :- \neg Flight(X, U), Hub(U), Flight(U, Y), \\ Hub(Y), Flight(Y, Z)$$
$$Q'_3(X, Z) :- \neg Flight(X, U), Hub(U), Flight(U, Y), \\ Flight(X, Z)$$

We need algorithms to detect these relationships.

Query containment and equivalence

Query Q_1 *contained in* query Q_2 if
for **every** database D

$$Q_1(D) \sqsubseteq Q_2(D)$$

Query Q_1 is *equivalent* to query Q_2 if
 $Q_1(D) \sqsubseteq Q_2(D)$ and $Q_2(D) \sqsubseteq Q_1(D)$

Note: containment and equivalence are properties of the queries, not of the database!

Why do we need It?

- When sources are described as views, we use containment to compare among them.
- If we can remove sub-goals from a query, we can evaluate it more efficiently.
- Actually, containment arises everywhere...

Reconsidering the Example

Relations: *Flight (dept, arrival)*
Hub(city)

Views:

$$Q_1(X, Y) :- \text{Flight}(X, Z), \text{Hub}(Z), \text{Flight}(Z, Y)$$
$$Q_2(X, Y) :- \text{Hub}(Z), \text{Flight}(Z, X), \text{Flight}(X, Y)$$

Query:

$$Q_3(X, Z) :- Q_1(X, Y), Q_2(Y, Z)$$

Unfolding:

$$Q'_3(X, Z) :- \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y),$$
$$\quad \quad \quad \text{Hub}(W), \text{Flight}(W, Y), \text{Flight}(Y, Z)$$

Remove redundant subgoals

Redundant subgoals?

$$Q'_3(X, Z) :- \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y), \\ \text{Hub}(W), \text{Flight}(W, Y), \text{Flight}(Y, Z)$$

\Rightarrow

$$Q''_3(X, Z) :- \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y), \\ \text{Flight}(Y, Z)$$

Is Q''_3 equivalent to Q'_3 ?

$$Q'_3(X, Z) :- \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y) \\ \text{Hub}(W), \text{Flight}(W, Y), \text{Flight}(Y, Z)$$

Containment: conjunctive queries

$$Q_1(\bar{X}) :- g_1(\bar{X}_1), \dots, g_n(\bar{X}_n)$$

No interpreted predicates (\geq, \neq)
or negation for now.

Recall semantics:

if φ maps the body subgoals to tuples in D
then, $\varphi(\bar{X})$ is an answer.

Containment mappings

$$Q_1(\bar{X}) :- g_1(\bar{X}_1), \dots, g_n(\bar{X}_n)$$

$$Q_2(\bar{Y}) :- h_1(\bar{Y}_1), \dots, h_m(\bar{Y}_m)$$

$$\varphi: \text{Vars}(Q_1) \rightarrow \text{Vars}(Q_2)$$

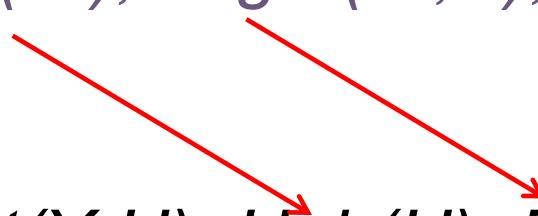
is a containment mapping if:

$$\varphi(g_i(\bar{X}_i)) \sqsubset \text{Body}(Q_2)$$

and

$$\varphi(\bar{X}) = \bar{Y}$$

Containment mapping: Example

$$Q'_3(X, Z) :- \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y), \\ \text{Hub}(W), \text{Flight}(W, Y), \text{Flight}(Y, Z)$$

$$Q''_3(X, Z) :- \text{Flight}(X, U), \text{Hub}(U), \text{Flight}(U, Y), \\ \text{Flight}(Y, Z)$$

Identity mapping on all variables, except:

$$W \rightarrow U$$

Theorem

[Chandra and Merlin, 1977]

Q_1 contains Q_2 if and only if there is a containment mapping from Q_1 to Q_2 .

Deciding whether Q_1 contains Q_2 is NP-complete.

Two Views of this Result

- Variable mapping:
 - a condition on variable mappings that guarantees containment
- Representative (canonical) databases:
 - We found a (single) database that would offer a counter example if there was one
- Containment results typically fall into one of these two classes.

Union of Conjunctive Queries

$\uparrow Q_1(X,Y) :- \text{Flight}(X,Z), \text{Flight}(Z,Y)$

{ $Q_1(X,Y) :- \text{Flight}(X,Z), \text{Flight}(Y,Z), \text{Hub}(Z)$

$\downarrow Q_2(X,Y) :- \text{Flight}(X,Z), \text{Flight}(Z,Y), \text{Hub}(Z)$

Theorem: a CQ is contained in a union of CQ's if and only if it is contained in *one* of the conjunctive queries.

Outline

- ✓ Review of basic database concepts
- ✓ Query unfolding
- ✓ Query containment
- Answering queries using views

Motivating example (part 1)

Movie(*ID, title, year, genre*)

Director(*ID, director*)

Actor(*ID, actor*)

$Q(T, Y, D) :- \neg \text{Movie}(I, T, Y, G), Y \sqsupseteq 1950, G = "comedy"$
 $\quad \quad \quad \text{Director}(I, D), \text{Actor}(I, D)$

$V_1(T, Y, D) :- \neg \text{Movie}(I, T, Y, G), Y \sqsupseteq 1940, G = "comedy"$
 $\quad \quad \quad \text{Director}(I, D), \text{Actor}(I, D)$

$V_1 \supseteq Q \quad \Rightarrow \quad Q(T, Y, D) :- \neg V_1(T, Y, D), Y \sqsupseteq 1950$

Containment is enough to show that V_1 can be used to answer Q.

Motivating example (part 2)

$Q(T, Y, D) :- \text{Movie}(I, T, Y, G), Y \square 1950, G = "comedy"$
 $\quad \quad \quad \text{Director}(I, D), \text{Actor}(I, D)$

$V_2(I, T, Y) :- \text{Movie}(I, T, Y, G), Y \square 1950, G = "comedy"$
 $V_3(I, D) :- \text{Director}(I, D), \text{Actor}(ID, D)$

Containment does not hold, but intuitively, V_2 and V_3 are useful for answering Q .

$Q'(T, Y, D) :- V_2(I, T, Y), V_3(I, D)$

How do we express that intuition?
Answering queries using views!

Problem definition

Input: Query Q

View definitions: V_1, \dots, V_n

A rewriting: a query Q' that refers only to the views and interpreted predicates

An equivalent rewriting of Q using V_1, \dots, V_n :
a rewriting Q' , such that $Q' \Leftrightarrow Q$.

Motivating example (part 3)

Movie(*ID, title, year, genre*)

Director(*ID, director*)

Actor(*ID, actor*)

$Q(T, Y, D) :- \neg \text{Movie}(I, T, Y, G), Y \square 1950, G = "comedy"$
 $\quad \quad \quad \text{Director}(I, D), \text{Actor}(I, D)$

$V_4(I, T, Y) :- \neg \text{Movie}(I, T, Y, G), \underline{Y \square 1960}, G = "comedy"$

$V_3(I, D) :- \neg \text{Director}(I, D), \text{Actor}(I, D)$

$Q''(T, Y, D) :- \underline{V_4(I, T, Y)}, V_3(I, D)$

maximally-contained rewriting

Maximally-contained rewritings

Input: Query Q

Rewriting query language L

View definitions: V_1, \dots, V_n

Q' is a maximally-contained rewriting of Q given V_1, \dots, V_n and L if:

1. $Q' \in L$,
2. $Q' \subseteq Q$, and
3. *there is no Q'' in L such that $Q'' \subseteq Q$ and $Q' \subset Q''$*

Motivation (in words)

- LAV-style data integration
 - Need maximally-contained rewritings
- Query optimization
 - Need equivalent rewritings
 - Implemented in most commercial DBMS
- Physical database design
 - Describe storage structures as views

Exercise: which of these views can be used to answer Q?

$Q(T, Y, D) :- \neg \text{Movie}(I, T, Y, G), Y \square 1950, G = "comedy"$
 $\quad \quad \quad \text{Director}(I, D), \text{Actor}(I, D)$

$V_2(I, T, Y) :- \neg \text{Movie}(I, T, Y, G), Y \square 1950, G = "comedy"$

$V_3(I, D) :- \neg \text{Director}(I, D), \text{Actor}(I, D)$

$V_6(T, Y) :- \neg \text{Movie}(I, T, Y, G), Y \square 1950, G = "comedy"$

$V_7(I, T, Y) :- \neg \text{Movie}(I, T, Y, G), Y \square 1950,$
 $\quad \quad \quad G = "comedy", \text{Award}(I, W)$

$V_8(I, T) :- \neg \text{Movie}(I, T, Y, G), Y \square 1940, G = "comedy"$

View-Based Query Answering

- Maximally-contained rewritings are parameterized by query language.
- More general question:
 - Given a set of view definitions, view instances and a query, what are **all** the answers we can find?
- We introduce **certain answers** as a mechanism for providing a formal answer.

View Instances = Possible DB's

Consider the two views:

$V_8(dir) :- Movie(ID, dir, actor)$

$V_9(actor) :- Movie(ID, dir, actor)$

And suppose the extensions of the views are:

$V_8: \{Allen, Copolla\}$

$V_9: \{Keaton, Pacino\}$

Possible Databases

There are multiple databases that satisfy the above view definitions: (we ignore the first argument of *Movie* below)

DB1. $\{(Allen, Keaton), (Coppola, Pacino)\}$

DB2. $\{(Allen, Pacino), (Coppola, Keaton)\}$

If we ask whether Allen directed a movie in which Keaton acted, we can't be sure.

Certain answers are those true in *all* databases that are consistent with the views and their extensions.

Certain Answers: Formal Definition

[Open-world Assumption]

- Given:
 - Views: V_1, \dots, V_n
 - View extensions v_1, \dots, v_n
 - A query Q
- A tuple t is a certain answer to Q under the open-world assumption if $t \in Q(D)$ for all databases D such that:
 - $V_i(D) \supseteq v_i$ for all i .

Certain Answers

[Closed-world Assumption]

- Given:
 - Views: V_1, \dots, V_n
 - View extensions v_1, \dots, v_n
 - A query Q
- A tuple t is a certain answer to Q under the open-world assumption if $t \in Q(D)$ for all databases D such that:
 - $V_i(D) = v_i \text{ for all } i.$

Certain Answers: Example

| | |
|----------------------------------|--------------|
| $V_8(dir) :- Director(ID, dir)$ | V8: {Allen} |
| $V_9(actor) :- Actor(ID, actor)$ | V9: {Keaton} |

$Q(dir, actor) :- Director(ID, dir), Actor(ID, actor)$

Under closed-world assumption:
single DB possible \Rightarrow (Allen, Keaton)

Under open-world assumption:
no certain answers.

The Good News

- The MiniCon and Inverse-rules algorithms produce all certain answers
 - Assuming no interpreted predicates in the query (ok to have them in the views)
 - Under open-world assumption
 - Corollary: they produce a maximally-contained rewriting

In Other News...

- Under closed-world assumption finding all certain answers is co-NP hard!

Proof: encode a graph -- $G = (V, E)$

$v_1(X) :- \text{color}(X, Y)$ $I(V_1) = V$

$v_2(Y) :- \text{color}(X, Y)$ $I(V_2) = \{\text{red}, \text{green}, \text{blue}\}$

$v_3(X, Y) :- \text{edge}(X, Y)$ $I(V_3) = E$

$q() :- \text{edge}(X, Y), \text{color}(X, Z), \text{color}(Y, Z)$

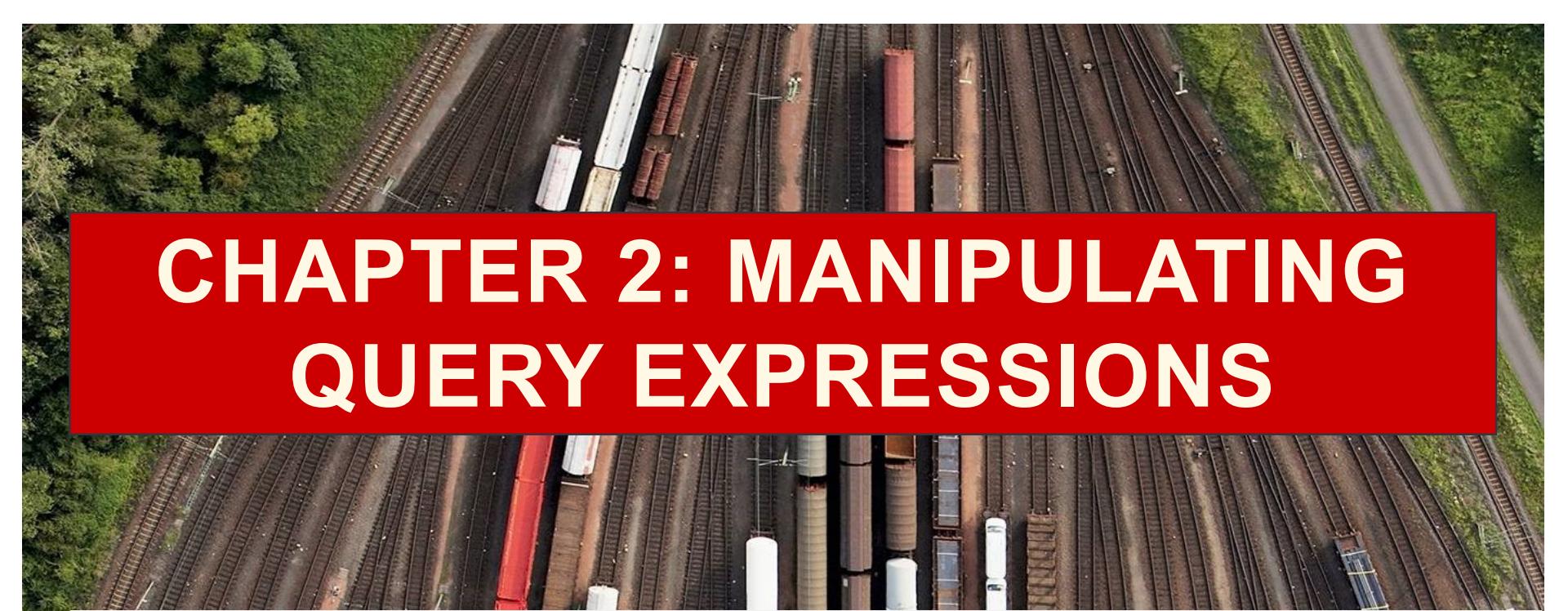
q has a certain tuple iff G is not 3-colorable

Interpreted Predicates

- In the views: no problem (all results hold)
- In the query Q:
 - If the query contains interpreted predicates, finding all certain answers is co-NP-hard even under open-world assumption
 - Proof: reduction to CNF.

Summary of Chapter 2

- Query containment and answering queries using views are fundamental tools in our arsenal.
- In general, they are NP-complete (or worse), but in practice they are not the bottleneck.
- Certain answers are the formalism we use to model answers in data integration systems:
 - They capture our knowledge about what tuples *must* be in the instance of the mediated schema.



CHAPTER 2: MANIPULATING QUERY EXPRESSIONS



PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

Introduction

- How does a data integration system decide which sources are relevant to a query? Which are redundant? How to combine multiple sources to answer a query?
- Answer: by reasoning about the contents of data sources.
 - Data sources are often described by queries / views.
- This chapter describes the fundamental tools for manipulating query expressions and reasoning about them.

More slides of Chapter 2

- Further detailed slides on query containments and equivalence

CQ's with Comparison Predicates

A tweak on containment mappings provides a sufficient condition:

$$\begin{aligned} Q_1(\overline{\underline{X}}) &:- g_1(\overline{\underline{X}_1}), \dots, g_n(\overline{\underline{X}_n}), C_1 \\ Q_2(\overline{Y}) &:- h_1(Y_1), \dots, h_m(Y_m), C_2 \end{aligned}$$

$\varphi: Vars(Q_1) \rightarrow Vars(Q_2)$:

$$\varphi(g_i(\overline{\underline{X}_i})) \sqsubset Body(Q_2)$$

$$\varphi(\overline{\underline{X}}) = \overline{Y}$$

and $C_2 \models \varphi(C_1)$

Example Containment Mapping

$Q_1(X, Y) :- \neg Flight(X, Z), Flight(Z, Y),$

$Population(Z, P), P \leq 100,000$

$Q_2(U, V) :- \neg Flight(U, W), Flight(W, V), Hub(W),$

$Population(W, S), S \leq 500,000$

$X \rightarrow U, Y \rightarrow V, Z \rightarrow W$

$P \leq 100,000 \models P \leq 500,000$

Containment Mappings are not Sufficient

$$Q_1(X,Y) :- R(X,Y), S(U,V), U \leq V$$

$$Q_2(X,Y) :- R(X,Y), S(U,V), S(V,U)$$

No containment mapping, but

$$Q_1 \supseteq Q_2$$

Query Refinements

$$Q_1(X,Y) :- R(X,Y), S(U,V), U \leq V$$
$$Q_2(X,Y) :- R(X,Y), S(U,V), S(V,U)$$

We consider the refinements of Q_2

$$Q_2(X,Y) :- R(X,Y), S(U,V), S(V,U), U \leq V$$
$$Q_2(X,Y) :- R(X,Y), S(U,V), S(V,U), V < U$$

The red containment mapping applies for the first refinement and the blue to the second.

Constructing Query Refinements

- Consider all complete orderings of the variables and constants in the query.
- For each complete ordering, create a conjunctive query.
- The result is a union of conjunctive queries.

Complete Orderings

Given

a conjunction C of interpreted atoms over
a set of variables X_1, \dots, X_n
a set of constants a_1, \dots, a_m

C_T is a complete ordering if: $C_T \models C$, and

$$\forall d_1, d_2 \quad \{X_1, \dots, X_n, a_1, \dots, a_m\}$$

$$C_T \models d_1 < d_2 \quad or \quad C_T \models d_1 > d_2 \quad or \quad C_T \models d_1 = d_2$$

Query Refinements

$$Q_1(\bar{X}) :- g_1(\bar{X}_1), \dots, g_n(\bar{X}_n), C_1$$

Let C_T be a complete ordering of C_1

Then:

$$Q_1(\bar{X}) :- g_1(\bar{X}_1), \dots, g_n(\bar{X}_n), C_T$$

is a refinement of Q_1

Theorem:

[Queries with Interpreted Predicates]

[Klug, 88, van der Meyden, 92]

Q_1 contains Q_2 if and only if there is a containment mapping from Q_1 to every refinement of Q_2 .

Deciding whether Q_1 contains Q_2 is Σ_2^p – *complete*.

In practice, you can do much better (see CQIPCContainment Algorithm in Sec. 2.3.4)

Queries with Negation

$$Q_1(\bar{X}) :- \neg g_1(\bar{X}_1), \dots, g_n(\bar{X}_n),$$
$$\quad \square h_1(\bar{Y}_1), \dots, \square h_k(\bar{Y}_k)$$

Queries assumed safe:

every head variable appears in
a positive sub-goal in the body.

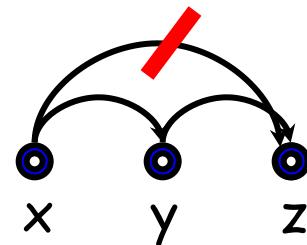
Revised containment mappings:

map negative subgoals in Q_1
to negative subgoals in Q_2 .

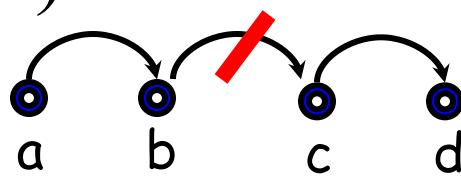
→ Sufficient condition, but not necessary.

Containment with no Containment Mapping

$Q_2() :- \neg a(X, Y), a(Y, Z), \square a(X, Z)$



$Q_1() :- \neg a(A, B), a(C, D), \square a(B, C)$



$Q_1 \supseteq Q_2$

Theorem

[Queries with Negation]

B: total number of variables and constants in Q_2 .

Q_1 contains Q_2 if and only if
 $Q_1(D) \supseteq Q_2(D)$ for all databases D with
at most B constants.

Deciding whether Q_1 contains Q_2
is Σ_2^p – complete.

Bag Semantics

| Origin | Destination | Departure Time |
|---------|-------------|----------------|
| SF | Seattle | 8AM |
| SF | Seattle | 10AM |
| Seattle | Anchorage | 1PM |

$$Q_1(X,Y) :- \text{Flight}(X, Z, W), \text{Flight}(Z, Y, W_1)$$

Set semantics: $\{(SF, Anchorage)\}$

Bags: $\{(SF, Anchorage), (SF, Anchorage)\}$

Theorem

[Conjunctive Queries, Bag Semantics]

Q_1 is equivalent to Q_2 if and only if there is a 1-1 containment mapping.

Trivial example on non-equivalence:

$Q_1(X) :- P(X)$

$Q_2(X) :- P(X), P(X)$

Query containment?

Grouping and Aggregation

- Count queries are sensitive to multiplicity
- Max queries are not sensitive to multiplicity
- and many more results (see Section 2.3.6).

Algorithms for answering queries using views

- Step 1: we'll bound the space of possible query rewritings we need to consider (no interpreted predicates)
- Step 2: we'll find efficient methods for searching the space of rewritings
 - Bucket Algorithm, MiniCon Algorithm
- Step 2b: we consider “logical approaches” to the problem:
 - The Inverse-Rules Algorithm
- We'll consider interpreted predicates, ...

Bounding the Rewriting Length

Theorem: if there is an equivalent erwriting,
there is one with at most n subgoals.

Query: $Q(\bar{X}) :- p_1(\bar{X}_1), \dots, p_n(\bar{X}_n)$

Rewriting: $Q'(\bar{X}) :- V_1(\bar{X}_1), \dots, V_m(\bar{X}_m)$

Expansion: $Q''(\bar{X}) :- \underbrace{\sigma^1}_{\sigma^1}, \dots, \underbrace{\sigma^m}_{\sigma^m}$

φ

σ^m

Proof: Only n subgoals in Q can contribute to
the image of the containment mapping φ

Complexity Result

[LMSS, 1995]

- Applies to queries with no interpreted predicates.
- Finding an equivalent rewriting of a query using views is NP-complete
 - Need only consider rewritings of query length or less.
- Maximally-contained rewriting:
 - Union of all conjunctive rewritings of length n or less.

The Bucket Algorithm

Key idea:

- Create a bucket for each subgoal g in the query.
- The bucket contains view atoms that contribute to g .
- Create rewritings from the Cartesian product of the buckets.

Bucket Algorithm in Action

$Q(ID, Dir) :- Movie(ID, title, year, genre), Revenues(ID, amount),$
 $Director(ID, dir), amount \square \$100M$

$V_1(I, Y) :- Movie(I, T, Y, G), Revenues(I, A), I \square 5000, A \square \$200M$

$V_2(I, A) :- Movie(I, T, Y, G), Revenues(I, A)$

$V_3(I, A) :- Revenues(I, A), A \leq \$50M$

$V_4(I, D, Y) :- Movie(I, T, Y, G), Director(I, D), I \leq 3000$

View atoms that can contribute to *Movie*:

$V_1(ID, year), V_2(ID, A'), V_4(ID, D', year)$

The Buckets and Cartesian product

| Movie(ID,title, year,genre) | Revenues(ID, amount) | Director(ID,dir) |
|--------------------------------|-------------------------|------------------|
| $V_1(ID,year)$ | $V_1(ID,Y')$ | $V_4(ID,Dir,Y')$ |
| $V_2(ID,A')$ | $V_2(ID,amount)$ | |
| $V_4(ID,D',year)$ | | |

Consider first candidate rewriting: first $V1$ subgoal is redundant, and $V1$ and $V4$ are mutually exclusive.

$$q_1'(ID,dir) : -V_1(ID, \cancel{year}), V_1(ID, y'), V_4(ID, dir, y')$$

Next Candidate Rewriting

| Movie(ID,title, year,genre) | Revenues(ID, amount) | Director(ID,dir) |
|--------------------------------|-------------------------|------------------|
| $V_1(ID,year)$ | $V_1(ID,Y')$ | $V_4(ID,Dir,Y')$ |
| $V_2(ID,A')$ | $V_2(ID,amount)$ | |
| $V_4(ID,D',year)$ | | |

$q_2'(ID,dir) :- V_2(ID,A'), V_2(ID,amount), V_4(ID,dir,y')$

$q_2'(ID,dir) :- V_2(ID,amount), V_4(ID,dir,y'),$

$amount \square \$100M$

The Bucket Algorithm: Summary

- Cuts down the number of rewriting that need to be considered, especially if views apply many interpreted predicates.
- The search space can still be large because the algorithm does not consider the interactions between different subgoals.
 - See next example.

The MiniCon Algorithm

$$Q(title, year, dir) :- Movie(ID, title, year, genre),$$
$$\qquad\qquad\qquad Director(ID, dir), Actor(ID, dir)$$
$$V_5(D, A) :- Director(I, D), Actor(I, A)$$


Intuition: The variable I is not in the head of V_5 , hence V_5 cannot be used in a rewriting.
MiniCon discards this option early on, while the Bucket algorithm does not notice the interaction.

MinCon Algorithm Steps

- Create MiniCon descriptions (MCDs):
 - Homomorphism on view heads
 - Each MCD covers a set of subgoals in the query with a set of subgoals in a view
- Combination step:
 - Any set of MCDs that covers the query subgoals (without overlap) is a rewriting
 - No need for an additional containment check!

MiniCon Descriptions (MCDs)

An atomic fragment of the ultimate containment mapping

$$Q(title, act, dir) :- Movie(ID, title, year, genre), \\ Director(ID, dir), Actor(ID, act)$$
$$V(I, D, A) :- Director(I, D), Actor(I, A)$$

MCD: $ID \rightarrow I$

mapping: $dir \rightarrow D$
 $act \rightarrow A$

covered subgoals of Q: {2,3}

MCDs: Detail 1

$$Q(title, year, dir) :- Movie(ID, title, year, genre), \\ Director(ID, dir), Actor(ID, dir)$$
$$V(I, D, A) :- Director(I, D), Actor(I, A)$$

Need to specialize the view first:

$$V'(I, D, D) :- Director(I, D), Actor(I, D)$$

MCD:
 $ID \rightarrow I$

mapping:
 $dir \rightarrow D$

covered subgoals of Q: {2,3}

MCDs: Detail 2

$$\begin{aligned} Q(title, year, dir) :- & \text{Movie}(ID, title, year, genre), \\ & \text{Director}(ID, dir), \text{Actor}(ID, dir) \\ V(I, D, D) :- & \text{Director}(I, D), \text{Actor}(I, D), \\ & \text{Movie}(I, T, Y, G) \end{aligned}$$

Note: the third subgoal of the view is *not* included in the MCD.

MCD:
 $ID \rightarrow I$

mapping:
 $dir \rightarrow D$

covered subgoals of Q *still*: {2,3}

Inverse-Rules Algorithm

- A “logical” approach to AQUV
- Produces maximally-contained rewriting in polynomial time
 - To check whether the rewriting is equivalent to the query, you still need a containment check.
- Conceptually simple and elegant
 - Depending on your comfort with Skolem functions...

Inverse Rules by Example

Given the following view:

$V_7(I,T,Y,G) :- Movie(I,T,Y,G), Director(I,D), Actor(I,D)$

And the following tuple in V_7 :

$V_7(79, \text{Manhattan}, 1979, \text{Comedy})$

Then we can infer the tuple:

$Movie(79, \text{Manhattan}, 1979, \text{Comedy})$

Hence, the following ‘rule’ is sound:

IN₁: $Movie(I,T,Y,G) :- V_7(I,T,Y,G)$

Skolem Functions

$V_7(I, T, Y, G) :- Movie(I, T, Y, G), Director(I, D), Actor(I, D)$

Now suppose we have the tuple

$V_7(79, \text{Manhattan}, 1979, \text{Comedy})$

Then we can infer that there exists *some* director. Hence, the following rules hold (note that they both use the same Skolem function):

IN₂: $Director(I, f_1(I, T, Y, G)) :- V_7(I, T, Y, G)$

IN₃: $Actor(I, f_1(I, T, Y, G)) :- V_7(I, T, Y, G)$

Inverse Rules in General

Rewriting = Inverse Rules + Query

$Q_2(title, year, genre) :- Movie(ID, title, year, genre)$

Given Q2, the rewriting would include:

IN₁, IN₂, IN₃, Q₂.

Given input: V₇(79,Manhattan,1979,Comedy)

Inverse rules produce:

Movie(79,Manhattan,1979,Comedy)

Director(79,*f₁(79,Manhattan,1979,Comedy)*)

Actor(79,*f₁(79,Manhattan,1979,Comedy)*)

Movie(Manhattan,1979,Comedy)

(the last tuple is produced by applying Q₂).

Comparing Algorithms

- Bucket algorithm:
 - Good if there are many interpreted predicates
 - Requires containment check. Cartesian product can be big
- MiniCon:
 - Good at detecting interactions between subgoals

Algorithm Comparison (Continued)

- Inverse-rules algorithm:
 - Conceptually clean
 - Can be used in other contexts (see later)
 - But may produce inefficient rewritings because it “undoes” the joins in the views (see next slide)
- Experiments show MiniCon is most efficient.
- Recently: MiniCon improved by using constraint satisfaction techniques.

Inverse Rules Inefficiency Example

Query and view:

$$Q(X, Y) :- e_1(X, Z), e_2(Z, Y)$$

$$V(A, B) :- e_1(A, C), e_2(C, B)$$

Inverse rules:

$$e_1(A, f_1(A, B)) :- V(A, B)$$

$$e_2(f_1(A, B), B) :- V(A, B)$$

Now we need to re-compute the join...



CHAPTER 3: DESCRIBING DATA SOURCES



PRINCIPLES OF DATA INTEGRATION

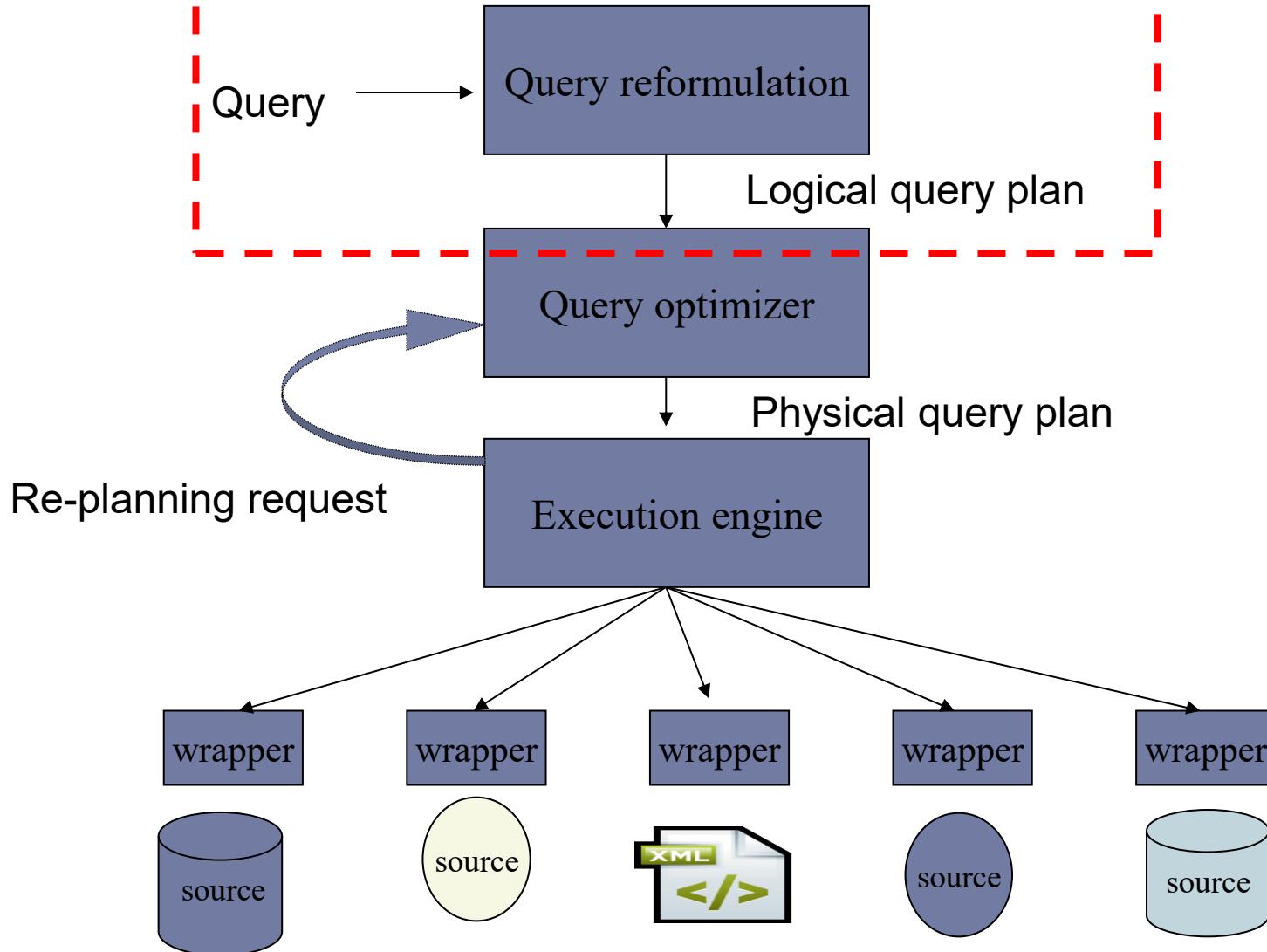
ANHAI DOAN ALON HALEVY ZACHARY IVES

Motivation and Outline

- **Descriptions of data sources** enable the data integration system to:
 - determine which sources are relevant to a query
 - access the sources appropriately
 - combine data from multiple sources
 - overcome limitations that specific sources may have
 - identify opportunities for more efficient query evaluation
- **Source descriptions** are a formalism for specifying the important aspects of data sources.

Outline

- Introduction to semantic heterogeneity
- Schema mapping languages
- Access pattern limitations
- Integrity constraints on the mediated schema
- Answer completeness
- Data-level heterogeneity



Schema Heterogeneity

- Schema heterogeneity is a fact of life.
 - Whenever schemas are designed by different people/organizations, they will be *different*, even if they model the *same* domain!
- The goal of schema mappings is to reconcile schema heterogeneity:
 - Mostly between the mediated schema and the schema of the data sources.

Schema Heterogeneity: An example

Mediated schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Local sources

S1

Movie(title, director, year, genre)
Actor(AID, firstName, lastName,
nationality, yearofBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2

Cinemas(place, movie, start)

S3

NYCCinemas(name, title, startTime)

S5

MovieGenres(title, genre)

S6

MovieDirectors(title, dir)

S7

MovieYears(title, year)

S4

Reviews(title, date, grade, review)

1. Table and Attribute Naming

Mediated Schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Sources

Table and attribute names

S1

Movie(title, director, year, genre)
Actor(AID, firstName, lastName,
nationality, yearOfBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2

Cinemas(place, movie, start)

S3

NYCCinemas(name, title, startTime)

S5

MovieGenres(title, genre)

S6

MovieDirectors(title, dir)

S7

MovieYears(title, year)

S4

Reviews(title, date, grade, review)

2. Tabular Organization of Schema

Mediated Schema

Movie: title, director, year, genre

Actors: title, name

Plays: movie, location, startTime

Reviews: title, rating, description

Sources

Different tabular organization

S1

Movie(title, director, year, genre)

Actor(AID, firstName, lastName,
nationality, yearofBirth)

ActorPlays(AID, MID)

MovieDetails(MID, director, genre, year)

S2

Cinemas(place, movie, start)

S3

NYCCinemas(name, title, startTime)

S5

MovieGenres(title, genre)

S6

MovieDirectors(title, dir)

S7

MovieYears(title, year)

S4

Reviews(title, date, grade, review)

3. Schema Coverage

Mediated Schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Sources

Different coverage and detail

S1

Movie(title, director, year, genre)
Actor(AID, firstName, lastName,
nationality, yearOfBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2

Cinemas(place, movie, start)

S3

NYCCinemas(name, title, startTime)

S5

MovieGenres(title, genre)

S6

MovieDirectors(title, dir)

S7

MovieYears(title, year)

S4

Reviews(title date, grade, review)

Semantic Heterogeneity Summary

- Differences in:
 - Naming of schema elements
 - Organization of tables
 - Coverage and detail of schema
 - Data-level representation (IBM vs. International Business Machines)
- Reason:
 - Schemas probably designed for different applications getContexts.

More in Source Descriptions

- Possible access patterns to the data source:
 - Are there required inputs? (e.g., web forms, web services)
 - Can the source process complex database queries?
- Source completeness:
 - Is the source complete, or partially complete?
- Reliability, load restrictions, mirror sites, ...

Outline

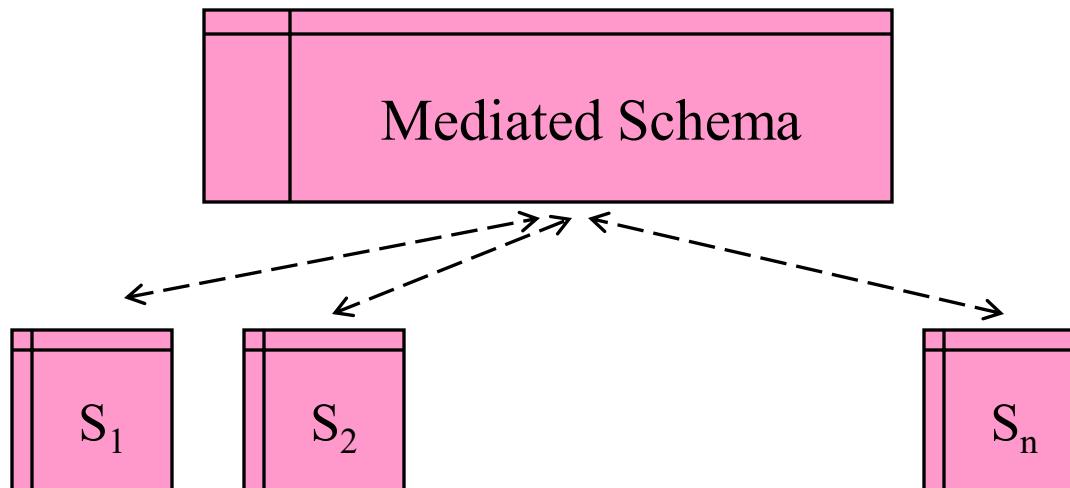
- ✓ Introduction to semantic heterogeneity
- Schema mapping languages
 - Access pattern limitations
 - Integrity Constraints on the mediated schema
 - Answer completeness
 - Data-level heterogeneity

Principles of Schema Mappings

Schema mappings describe the relationship between:

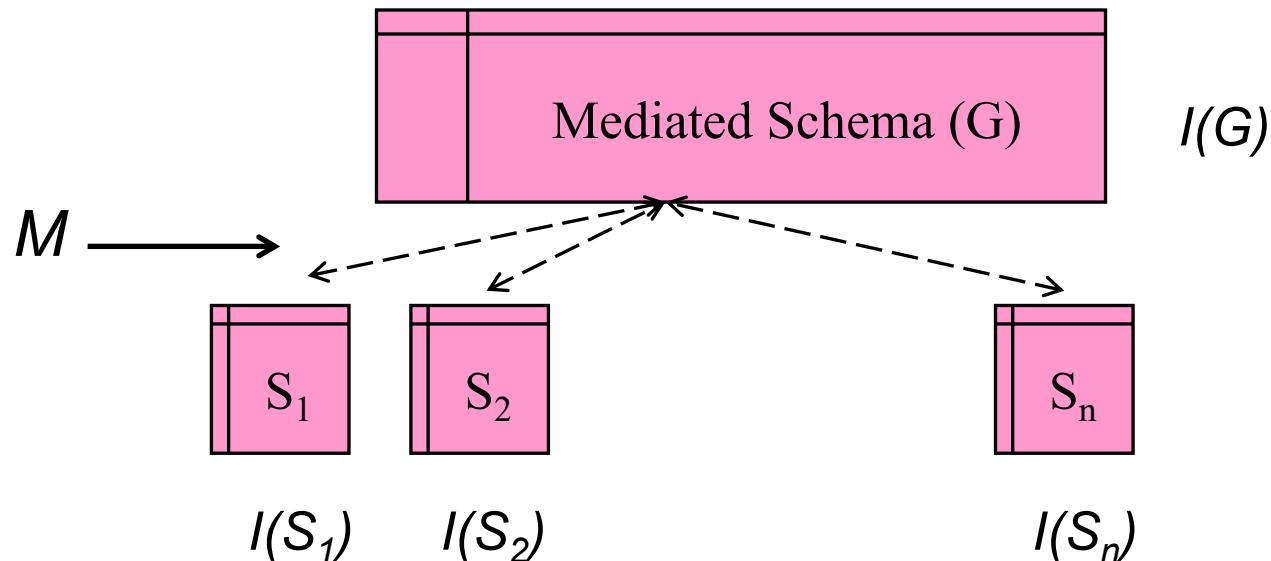


Or between:



Semantics of Schema Mappings

Formally, schema mappings describe a relation: which instances of the mediated schema are consistent with the current instances of the data sources.



$I(G)$ ($I(S_i)$): the set of possible instances of the schema G (S_i).

$$M_R \subseteq I(G) \times I(S_1) \times \dots \times I(S_n)$$

Relations, explained

- A relation is a subset of the Cartesian product of its columns' domains:

| | |
|---|----|
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |

- The table on the left is a subset of the Cartesian product of $\text{Int} \times \text{Int}$.
- The table describes the Squared relation.
- Similarly, Mr specifies the possible instances of the mediated schema, given instances of the sources.

Possible Instances of Mediated Schema: An example

- Source 1: *(Director, Title, Year)* with tuples
 - {(Allen, Manhattan, 1979),}
 - (Coppola, GodFather, 1972)}
- Mediated schema: *(Title, Year)*
 - Simple projection of Source 1
 - Only one possible instance:
 - {**(Manhattan, 1979)**, **(GodFather, 1972)**}

Possible Instances of Mediated Schema: Second example

- Source 1: *(Title, Year)* with tuples
 - $\{(\text{Manhattan}, 1979), (\text{GodFather}, 1972)\}$

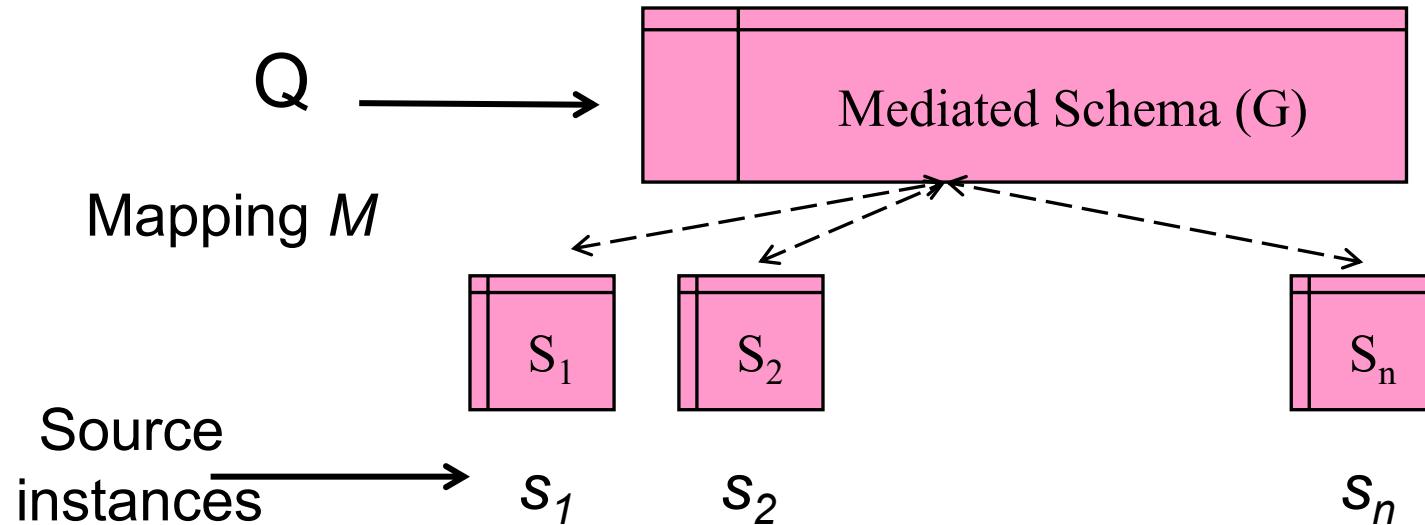
- Mediated schema: *(Director, Title, Year)*
 - Possible instance 1: $\{(\text{Allen}, \text{Manhattan}, 1979), (\text{Coppola}, \text{GodFather}, 1972)\}$
 - Another possible instance 2: $\{(\text{Halevy}, \text{Manhattan}, 1979), (\text{Stonebraker}, \text{GodFather}, 1972)\}.$

Answering Queries over Possible Instances of Mediated Schema

- Mediated schema: *(Director, Title, Year)*
 - Possible instance 1: {(Allen, Manhattan, 1979), (Coppola, GodFather, 1972)}
 - Another possible instance 2: {(Halevy, Manhattan, 1979), (Stonebraker, GodFather, 1972)}.
- *Query Q1: return all years of movies*
 - Answer: (1979, 1972) are **certain answers**.
- *Query Q2: return all directors*
 - **No certain answers** because no directors appear in all possible instances of the mediated schema.

Certain Answers Makes this Formal

An answer is *certain* if it is true in every instance of the mediated schema that is consistent with: (1) the instances of the sources and (2) the mapping M .



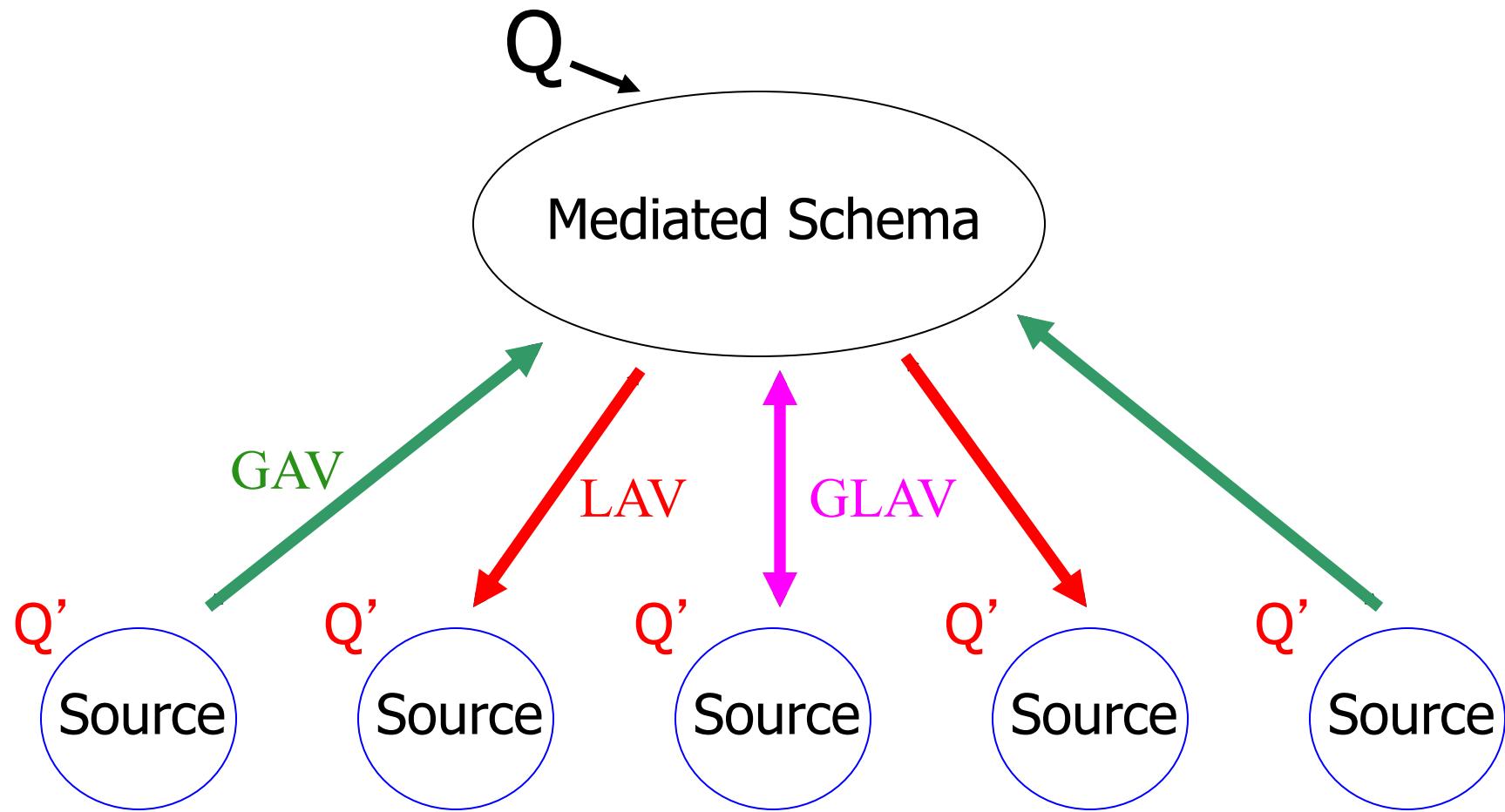
$t \in Q(s_1, \dots, s_n)$ iff
 $t \in Q(g)$ for $\forall g$, s.t. $(g, s_1, \dots, s_n) \in M_R$

Source Description Languages

Features

- *Flexibility:*
 - Should be able to express relationships between real schemata
- *Efficient reformulation:*
 - Computational complexity of reformulation and finding answers
- *Easy update:*
 - Should be easy to add and delete sources

Languages for Schema Mapping



Global-as-View (GAV)

- Mediated schema defined as a set of views over the data sources

Movie: title, director, year, genre

S1
Movie(MID,title)
Actor(AID, firstName, lastName, nationality, yearOfBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

$$\begin{aligned} \text{Movie}(title, director, year, genre) &\supseteq \\ S1.\text{Movie}(MID, title), \\ S1.\text{MovieDetail}(MID, director, genre, year) \end{aligned}$$

GAV: Formal Definition

A set of expressions of the form:

$$G_i(\bar{X}) \supseteq Q(\bar{S}) \quad \text{or} \quad G_i(\bar{X}) = Q(\bar{S})$$

open-world
assumption

closed-world
assumption

- G_i : relation in mediated schema
- $Q(\bar{S})$:query over source relations

GAV Example

Movie: title, director, year, genre

Movie(title, director, year, genre) ⊇

S1.Movie(MID, title), S1.MovieDetail(MID, director, genre, year)

Movie(title, director, year, genre) ⊇

S5.MovieGenres(title, genre),

S6.MovieDirectors(title, director),

S7.MovieYears(title, year)

S5

MovieGenres(title, genre)

S6

MovieDirectors(title, dir)

S7

MovieYears(title, year)

S1

Movie(MID, title)

*Actor(AID, firstName, lastName,
nationality, yearOfBirth)*

ActorPlays(AID, MID)

MovieDetails(MID, director, genre, year)

GAV Example (cont.)

Plays: movie, location, startTime

$Plays(movie, location, startTime) \supseteq$

$S2.Cinemas(location, movie, startTime)$

$Plays(movie, location, startTime) \supseteq$

$S3.NYCCinemas(location, movie, startTime)$

S2

Cinemas(place, movie, start)

S3

NYCCinemas(name, title, startTime)

Reformulation in GAV

- Given a query Q on the mediated schema:
 - Return the best query possible on the data sources.

Reformulation in GAV

= Query/View Unfolding

$Q(title, location, startTime) :-$

$Movie(title, director, year, "comedy"),$
 $Plays(title, location, st), st \geq 8pm$

$Movie(title, director, year, genre) \supseteq$

$S1.Movie(MID, title), S1.MovieDetail(MID, director, genre, year)$

$Plays(movie, location, startTime) \supseteq$

$S2.Cinemas(location, movie, startTime)$

First Reformulation

Q(title, location, startTime) :-

*Movie(title, director, year, “comedy”),
Plays(title, location, st), st ≥ 8pm*



Q'(title, location, startTime) :-

S1.Movie(MID, title),

S1.MovieDetail(MID, director, “comedy”, year)

S2.Cinemas(location, title, st), st ≥ 8pm

Another Reformulation

Q(title, location, startTime) :-

Movie(title, director, year, “comedy”),

Plays(title, location, st), st ≥ 8pm



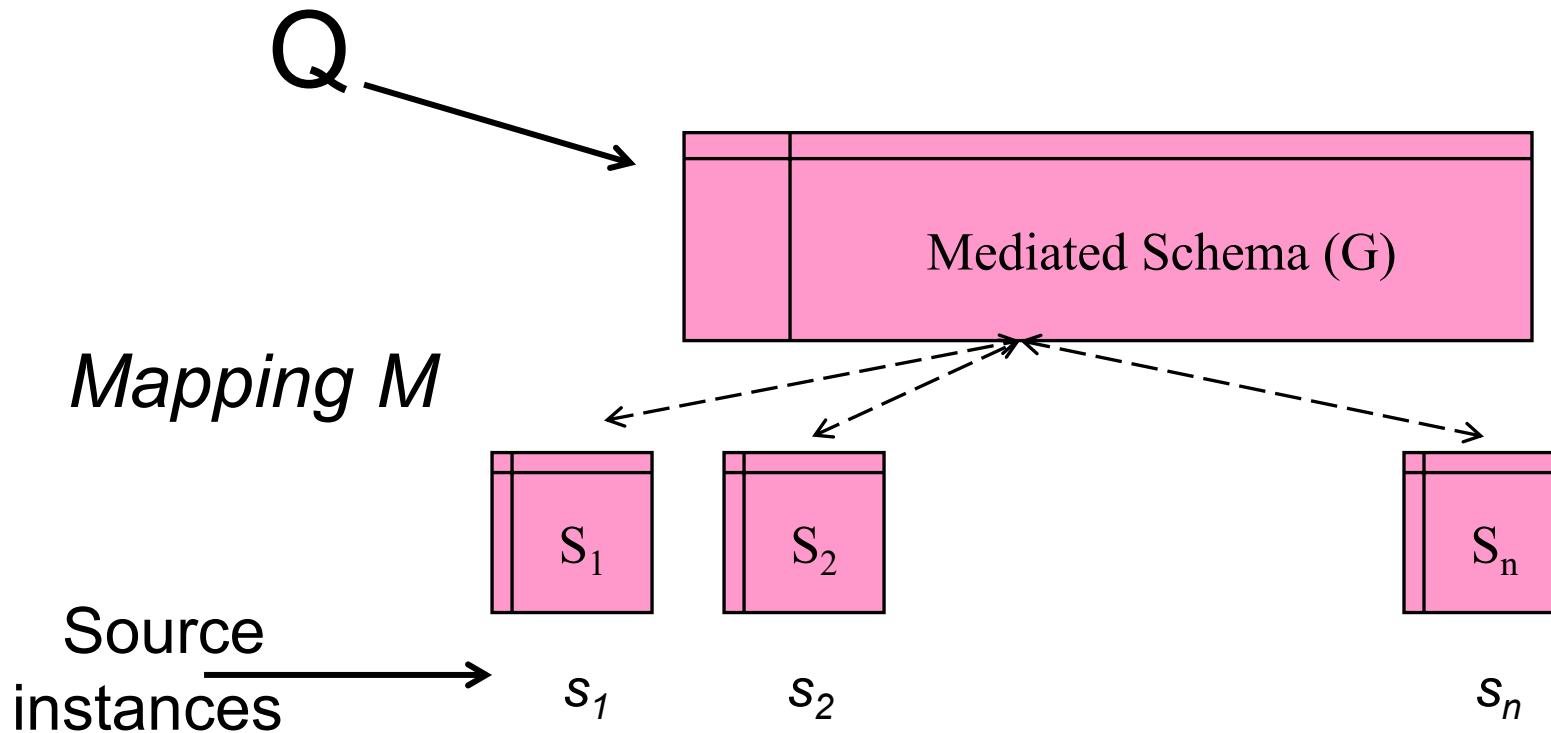
Q'(title, location, startTime) :-

S1.Movie(MID, title),

S1.MovieDetail(MID, director, “comedy”, year)

S3.NYCCinemas(location, title, st), st ≥ 8pm

Certain Answers (recall definition)

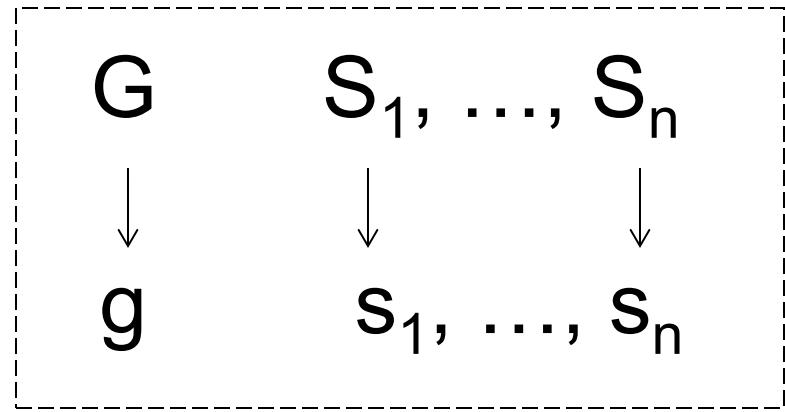


$t \in Q(s_1, \dots, s_n)$ iff
 $t \in Q(g)$ for $\forall g$, s.t. $(g, s_1, \dots, s_n) \in M_R$

Semantics of GAV

$(g, s_1, \dots, s_n) \in M_R$ if:

$$G_i(\bar{X}) \supseteq Q(\bar{S}) \quad \Rightarrow$$



The extension of G_i in g is a **super-set** of evaluating Q_i on the sources.

$$G_i(\bar{X}) = Q(\bar{S}) \quad \square$$

The extension of G_i in g is **equal** to evaluating Q_i on the sources.

Tricky Example for GAV

S8: stores pairs of (actor, director)

Movie: title, director, year, genre

Actors: title, name

Plays: movie, location, startTime

Reviews: title, rating, description

$Actors(NULL, actor) \supseteq S8(actor, director)$

$Movie(NULL, director, NULL, NULL) \supseteq S8(actor, director)$

Tricky Example for GAV

$Actors(NULL, actor) \supseteq S8(actor, director)$

$Movie(NULL, director, NULL, NULL) \supseteq S8(actor, director)$

Given the S8 tuples:

({Keaton, Allen}, {Pacino, Coppola})

We'd get tuples for the mediated schema:

$Actors: (\{NULL, Keaton\}, \{NULL, Pacino\})$

$Movie: (\{NULL, Allen, NULL, NULL\},$
 $\{NULL, Coppola, NULL, NULL\})$

Tricky Example (2)

Actors: ({NULL, Keaton}, {NULL, Pacino})

*Movie: ({NULL, Allen, NULL, NULL},
 {NULL, Coppola, NULL, NULL})*

Can't answer the query:

Q(actor, director) :-

Actors(title, actor),

Movie(title, director, genre, year)

LAV (Local as View) will solve this problem

GAV Summary

- Mediated schema is defined as views over the sources.
- Reformulation is conceptually easy
 - Polynomial-time reformulation and query answering.
- GAV forces everything into the mediated schema's perspective:
 - Cannot capture a variety of tabular organizations.

Local-as-View (LAV)

- Data sources defined as views over mediated schema!

S5

MovieGenres(title, genre)

S6

MovieDirectors(title, dir)

S7

MovieYears(title, year)

Movie: title, director, year, genre

Actors: title, name

Plays: movie, location, startTime

Reviews: title, rating, description

$S5.MovieGenres(title, genre) \subseteq Movie(title, dir, year, genre)$

$S6.MovieDirectors(title, dir) \subseteq Movie(title, dir, year, genre)$

Local-as-View (LAV)

- Data sources defined as views over mediated schema!

s8

ActorDirectors(actor,dir)

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

S8.ActorDirectors(actor,dir) ⊆

Movie(title,dir,year,genre), Actor(title,actor)

year ≥ 1980

LAV: Formal Definition

A set of expressions of the form:

$$S_i(\bar{X}) \subseteq Q_i(G)$$

open-world
assumption

$$\text{or} \quad S_i(\bar{X}) = Q_i(G)$$

closed-world
assumption

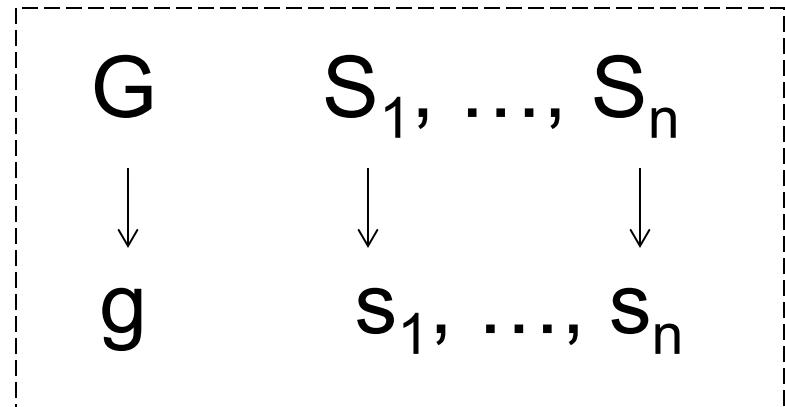
- $S_i(\bar{X})$: source relation
- $Q_i(G)$: query over mediated schema

Semantics of LAV

$(g, s_1, \dots, s_n) \in M_R$ if:

$$S_i(\overline{X}) \subseteq Q_i(G) \quad \Rightarrow$$

The result of Q_i over g is a **superset** of s_i .



$$S_i(\overline{X}) = Q_i(G) \quad \square$$

The result of Q_i over g **equals** s_i .

Possible Databases

Unlike GAV, LAV definitions imply a *set* of possible databases for the mediated schema.

$$\begin{aligned} S8.\textit{ActorDirectors}(actor, dir) \subseteq \\ \textit{Movie}(title, dir, year, genre), \textit{Actor}(title, actor) \end{aligned}$$

$$S8 : \{(Keaton, Allen)\}$$

Two possible databases for the mediate schema are:

Movie: {("manhattan", allen, 1979, comedy)}
Actor:{("manhattan", keaton)}

Movie: {("foobar", allen, 1979, comedy)}
Actor:{("foobar", keaton)}

Possible Databases

Since the source may be incomplete, other tuples may be in the instance of the mediated schema:

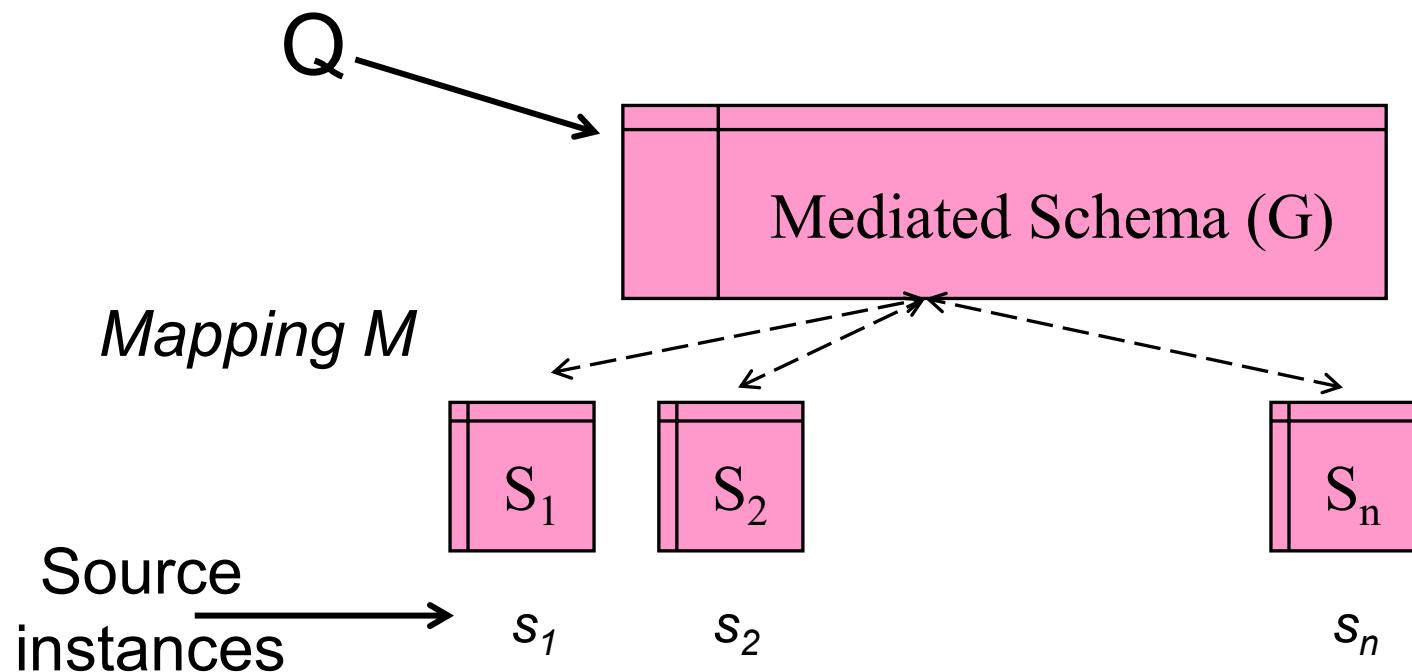
$$S8.\textit{ActorDirectors}(\textit{actor},\textit{dir}) \subseteq \\ \textit{Movie}(\textit{title},\textit{dir},\textit{year},\textit{genre}), \textit{Actor}(\textit{title},\textit{actor})$$

$$S8 : \{(Keaton, Allen)\}$$

Movie: { (manhattan, allen, 1981, comedy),
 (leatherheads, clooney, 2008, comedy) }

Actor:{ (manhattan, keaton),
 (the godfather, keaton) }

Certain Answers



$t \in Q(s_1, \dots, s_n) \text{ iff}$

$t \in Q(g) \text{ for } \forall g, \text{ s.t. } (g, s_1, \dots, s_n) \in M_R$

Certain Answers: Example 1

```
S8.ActorDirectors(actor,dir) ⊆  
Movie(title,dir,year,genre),Actor(title,actor)  
year ≥ 1980
```

$S8 : \{(Keaton, Allen)\}$

```
Q(actor,dir) :-  
Movie(title,dir,year,genre),Actor(title,actor)
```

Only one certain answer: *(Keaton, Allen)*

Certain Answers: Example 2

$$V_8(dir) = DirectorActor(ID, dir, actor)$$
$$V_9(actor) = DirectorActor(ID, dir, actor)$$
$$Q(dir, actor) :- \neg Director(ID, dir), Actor(ID, actor)$$

{Allen}
{Keaton}

Under closed-world assumption:

single DB possible \Rightarrow (Allen, Keaton)

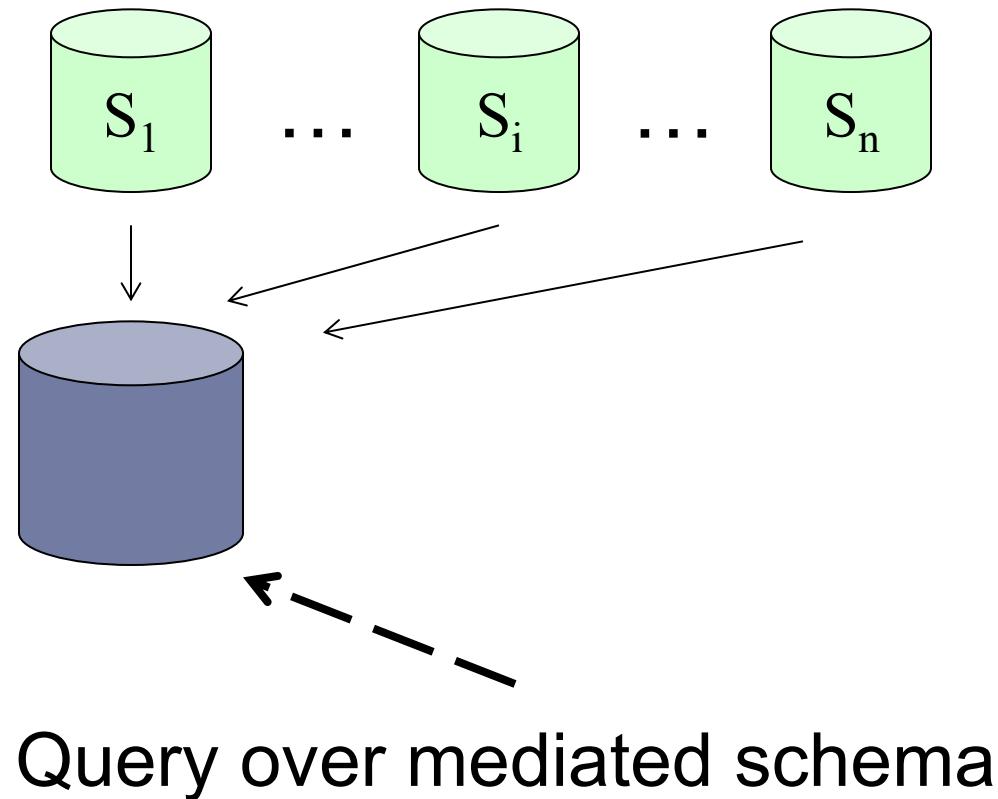
Under open-world assumption:

no certain answers.

Reformulation in LAV

We're given tuples for
sources
(expressed as views)

Mediated schema
(but no tuples)



This is exactly the problem of:...
Answering queries using views!

Local-as-View Summary

- Reformulation = answering queries using views
- Algorithms work well in practice:
 - Reformulation is not the bottleneck
- Under some conditions, guaranteed to find all certain answers
 - In practice, they typically do.
- LAV expresses incomplete information
 - GAV does not. Only a single instance of the mediated schema is consistent with sources.

LAV Limitation

Movie: title, director, year, genre

S1

Movie(*MID*,title)

Actor(AID, firstName, lastName, nationality, yearofBirth)

ActorPlays(AID, *MID*)

MovieDetails(*MID*, director, genre, year)

If a key is internal to a data source, LAV cannot use it.

So...

GLAV

A set of expressions of the form:

$$Q^S(\bar{X}) \subseteq Q^G(\bar{X}) \text{ or } Q^S(\bar{X}) = Q^G(\bar{X})$$

- Q^G : query over mediated schema
- Q^S : query over data sources

GLAV Example

$$\begin{aligned} & S1.Movie(MID, title), S1.MovieDetail(MID, dir, genre, year) \\ & \subseteq \\ & Movie(title, dir, "comedy", year), year \geq 1970 \end{aligned}$$

Reformulation in GLAV

- Given a query Q
- Find a rewriting Q' using the views

$$Q^S(\bar{X}) \subseteq Q^G(\bar{X})$$

$$Q_1^G, \dots, Q_n^G$$

- Create Q'' by replacing: $Q_i^G \rightarrow Q_i^S$
- Unfold Q_1^S, \dots, Q_n^S

Outline

- ✓ Introduction to semantic heterogeneity
- ✓ Schema mapping languages
- Access pattern limitations
- Integrity Constraints on the mediated schema
- Answer completeness
- Data-level heterogeneity

Access-Pattern Limitations

- Often we can only access the data sources in specific ways:
 - Web forms: require certain inputs
 - Web services: interface definitions
 - Controlling load on systems: allow only limited types of queries
- Model limitations using adornments:
 - $V^{bf}(X,Y)$: first argument must be bound

Example Binding Patterns

$Cites(X, Y)$
 $DBPapers(X)$
 $AwardPaper(X)$

$S1 : CitationDB^{bf}(X, Y) \subseteq Cites(X, Y)$

$S2 : CitingPapers^f(X) \subseteq Cites(X, Y)$

$S3 : DBSource^f(X) \subseteq DBpapers(X)$

$S4 : AwardDB^b(X) \subseteq AwardPaper(X)$

Executable Plans

conjunctive plan:

$$q_1(\overline{X_1}), \dots, q_n(\overline{X_n})$$

adornments: BF_i for q_i

Plan is **executable** if there are $bf_i \in BF_i$ s.t.

If X is in position k of q_i , and bf_i has ‘ b ’ in k ’ the position, then X occurs earlier in the plan.
i.e., every variable that must be bound has a value.

Example (1)

$S1 : CitationDB^{bf}(X, Y) \subseteq Cites(X, Y)$

$S2 : CitingPapers^f(X) \subseteq Cites(X, Y)$

$S3 : DBSource^f(X) \subseteq DBpapers(X)$

$S4 : AwardDB^b(X) \subseteq AwardPaper(X)$

Cites(X,Y)
DBPapers(X)
AwardPaper(X)

$Q(X) :- Cites(X, 001)$

X $Q'(X) :- CitationDB(X, 001)$

$Q'(X) :- CitingPapers(X), CitationDB(X, 001)$

Example (2): cannot bound the length of a possible rewriting!

$S1 : CitationDB^{bf}(X, Y) \subseteq Cites(X, Y)$

$S3 : DBSource^f(X) \subseteq DBpapers(X)$

$S4 : AwardDB^b(X) \subseteq AwardPaper(X)$

Cites(X,Y)
DBPapers(X)
AwardPaper(X)

$Q(X) :- AwardPaper(X)$

$Q'(X) :- DBSource(X), AwardDB(X)$

$Q'(X) :- DBSource(Y), CitationDB(Y, X), AwardDB(X)$

$Q'(X) :- DBSource(Y), CitationDB(Y, X_1), \dots, CitationDB(X_n, X), AwardDB(X)$

Recursive Plans to the Rescue

```
papers(X) :- DBSource(X)
```

```
papers(X) :- papers(Y), CitationDB(Y,X)
```

```
Q'(X) :- papers(X), AwardDB(X)
```

Theorem: this can be done in general

[Duschka, Genesereth & Levy, 1997]

Outline

- ✓ Introduction to semantic heterogeneity
- ✓ Schema mapping languages
- ✓ Access pattern limitations
- Integrity Constraints on the mediated schema
(further reading)
- Answer completeness (further reading)
- Data-level heterogeneity (further reading)

Summary of Chapter 3

- Source descriptions include:
 - Schema mappings
 - Completeness, access-pattern limitations
 - Data transformations
 - Additional query-processing capabilities
- Schema mapping languages
 - GAV: reformulation by unfolding
 - LAV/GLAV: reformulation by answering queries using views
- Binding patterns and integrity constraints can lead to tricky cases:
 - Recursive rewritings can often address these.



CHAPTER 3: DESCRIBING DATA SOURCES



PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

Motivation and Outline

- **Descriptions of data sources** enable the data integration system to:
 - determine which sources are relevant to a query
 - access the sources appropriately
 - combine data from multiple sources
 - overcome limitations that specific sources may have
 - identify opportunities for more efficient query evaluation
- **Source descriptions** are a formalism for specifying the important aspects of data sources.

Outline

- Introduction to semantic heterogeneity
- Schema mapping languages
- Access pattern limitations
- Integrity constraints on the mediated schema
- Answer completeness
- Data-level heterogeneity

Outline

- ✓ Introduction to semantic heterogeneity
- ✓ Schema mapping languages
- ✓ Access pattern limitations
- Integrity Constraints on the mediated schema
(further reading)
- Answer completeness (further reading)
- Data-level heterogeneity (further reading)

Integrity Constraints

- In the presence of integrity constraints on the mediated schema, the reformulation algorithms may need to be modified.
- Next, an example in the context of Local-as-View.
- See an example for Global-as-view in Chapter 3.4.2.

Integrity Constraints with LAV

Mediated schema:

$\text{Schedule}(\text{airline}, \text{flightNum}, \text{date}, \text{pilot}, \text{aircraft})$

Functional dependencies:

$\text{Pilot} \rightarrow \text{Airline}$ and $\text{Aircraft} \rightarrow \text{Airline}$

Data source:

$S(\text{date}, \text{pilot}, \text{aircraft}) \subseteq$
 $\text{schedule}(a, fN, date, pilot, aircraft)$

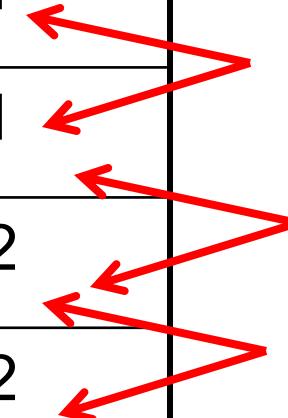
Query: (pilots in same airline as Mike)

$q(p) :- \text{schedule}(a, fN, date, "mike", aircraft)$
 $\text{schedule}(a, f, d, p, a)$

Without functional dependencies, no answers!

But We Do Have Answers...

| Date | Pilot | Aircraft |
|------|-------|----------|
| 1/1 | Mike | 111 |
| 5/2 | Ann | 111 |
| 1/3 | Ann | 222 |
| 4/3 | John | 222 |



Mike and Ann work for the same airline

111 and 222 belong to the same airline

John and Ann work for the same airline, ...

No Limit to Rewriting Size

```
q'_n(p) :- S(D1, "mike", C1), S(D2, p2, C1),  
          S(D3, p2, C2), S(D4, p3, C2), ...,  
          S(D2n-2, pn, Cn-1), S(D2n-1, pn, Cn)  
          S(D2n, p, Cn)
```

Does this sound familiar?

Recursive plans to the rescue!

Recursive Plan

Start with inverse rules and get:

$schedule(f_1(1/1, Mike, \#111), f_2(1/1, Mike, \#111), 1/1, Mike, \#111)$

$schedule(f_1(5/2, Ann, \#111), f_2(5/2, Ann, \#111), 5/2, Ann, \#111)$

$schedule(f_1(1/3, Ann, \#222), f_2(1/3, Ann, \#222), 1/3, Ann, \#222)$

$schedule(f_1(4/3, John, \#222), f_2(4/3, John, \#222), 4/3, John, \#222)$

But we know that:

$$f_1(1/1, Mike, \#111) = f_1(5/2, Ann, \#111)$$

$$f_1(5/2, Ann, \#111) = f_1(1/3, Ann, \#222)$$

$$f_1(1/3, Ann, \#222) = f_1(4/3, John, \#222)$$

The e relation

- Let's define a relation $e(X,Y)$ that stores all pairs that are actually equal.
- $e(x,x)$ – holds for every x .
- The other rules for inferring $e(X,Y)$ are derived from the functional dependencies.
- Add: $e(X,Y) :- e(X,Z), e(Z,Y)$

The Rules

Aircraft → Airline: So...

```
e(X,Y) :- schedule(X,F,P,D,A),  
          schedule(Y,F',P',D',A'),  
          e(A,A')
```

Pilot → Airline: So...

```
e(X,Y) :- schedule(X,F,P,D,A),  
          schedule(Y,F',P',D',A'),  
          e(P,P')
```

One More Step

Reformulate the query to use $e(X, Y)$:

$$q(P) :- \text{schedule}(AL, F', D', Mike, A), \\ \text{schedule}(AL, F, D, P, A')$$

$$q(P) :- \text{schedule}(AL, F', D', P', A), \\ \text{schedule}(AL', F, D, P, A'), \\ e(AL, AL'), e(P', Mike)$$

Integrity Constraints in GAV

$schedule(flNum) \subseteq$
 $flight(flightNum)$

| Airline | flNum | Date | Pilot | Craft |
|---------|-------|------|-------|-------|
| United | 111 | 1/1 | Mike | 15 |
| SAS | 222 | 1/3 | Ann | 17 |

| flightNum | Origin | Dest |
|-----------|---------|------|
| 222 | Seattle | SFO |
| 333 | SFO | CPH |

$q(fn) :- schedule(\text{airline}, fN, \text{date}, \text{pilot}, \text{aircraft})$
 $\quad flight(fN, \text{origin}, \text{destination})$

111 should be in the answer, but won't.

Outline

- ✓ Introduction to semantic heterogeneity
- ✓ Schema mapping languages
- ✓ Access pattern limitations
- ✓ Integrity Constraints on the mediated schema
- Answer completeness
- Data-level heterogeneity

Local Completeness

- We've modeled complete sources with GAV/LAV/GLAV
- Often, sources are only **partially** complete:
 - Online movie db is complete only w.r.t recent movies
 - My bib file is complete only w.r.t. data integration
- We need to model local completeness
- Question: given partially complete sources:
 - Is the answer to a query Q complete?

Modeling Local Completeness

$S5.MovieGenres(title, genre) \subseteq Movie(title, dir, year, genre)$

$S6.MovieDirectors(title, dir) \subseteq Movie(title, dir, year, genre)$

$S7.MovieYear(title, year) \subseteq Movie(title, dir, year, genre)$

We model local completeness with *LC* statements of the form:

$LC(S5.MovieGenres(title, genre), genre = "comedy")$

$LC(S6.MovieDirectors(title, dir), American(dir))$

$LC(S7.MovieYears(title, year), year \square 1980)$

Formally

Given a LAV description: $S(\bar{X}) \subseteq Q(\bar{X}), C_1$

We *define LC with a constraint C on the arguments of the source S*: $LC(S, C)$.

Semantics: add the following to source description:

$$S(\bar{X}) = Q(\bar{X}), C, C_1$$

Answer Completeness

Is the answer to Q complete, given the sources?

$LC(S5.MovieGenres(title, genre), genre = "comedy")$

$LC(S6.MovieDirectors(title, dir), American(dir))$

$LC(S7.MovieYears(title, year), year \sqsupseteq 1980)$

- ✓ $q_1(title :- Movie(title, dir, genre, "comedy"), year \sqsupseteq 1990, American(dir))$
- ✗ $q_2(title :- Movie(title, dir, genre, "comedy"), year \sqsupseteq 1970, American(dir))$

Detecting answer completeness can be reduced to a query containment problem.

Algorithm for Detecting Answer Completeness

Given: $S_i(\overline{X}_i) \subseteq R(\overline{X}_i), C_i$ $LC(S_i, C_i)$

Assume E_i are new relation names, and define

$$\begin{aligned} V_i(\overline{X}_i) &:- E_i(\overline{X}_i), \square C_i \\ V_i(\overline{X}_i) &:- S_i(\overline{X}_i) \end{aligned}$$

Let Q be a conjunctive query Define Q' by replacing S_i in Q by V_i

Q is answer-complete iff Q is equivalent to Q' .

Outline

- ✓ Introduction to semantic heterogeneity
- ✓ Schema mapping languages
- ✓ Access pattern limitations
- ✓ Integrity Constraints on the mediated schema
- ✓ Answer completeness
- Data-level heterogeneity

Data-Level Heterogeneity

- Huge problem in practice:
 - Data coming from different sources rarely joins perfectly.
- Differences of scale:
 - Celsius vs. Fahrenheit
 - Numeric vs. letter grades
 - First Name, Last Name vs. FullName
 - Prices with taxes or without
 - ...

Mappings with Transformations

$$S(city, temp - 32 * 5/9, month) \subseteq \\ Weather(city, temp, humidity, month)$$
$$CDStore(cd, price) \subseteq CDPrices(cd, state, price * (1 + rate)), \\ LocalTaxes(state, rate)$$

Reference Reconciliation

- Multiple ways to refer to the same real-world entity:
 - John Smith vs. J. R. Smith
 - IBM vs. International Business Machines
 - Alon Halevy vs. Alon Levy
 - South Korea vs. Republic of Korea
- Create concordance tables:
 - Pairs of corresponding values
 - How? See the next chapters!

Summary of Chapter 3

- Source descriptions include:
 - Schema mappings
 - Completeness, access-pattern limitations
 - Data transformations
 - Additional query-processing capabilities
- Schema mapping languages
 - GAV: reformulation by unfolding
 - LAV/GLAV: reformulation by answering queries using views
- Binding patterns and integrity constraints can lead to tricky cases:
 - Recursive rewritings can often address these.

Chapter 4: String Matching

PRINCIPLES OF
DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

Introduction

- Find strings that refer to same real-world entities
 - “David Smith” and “David R. Smith”
 - “1210 W. Dayton St Madison WI” and “1210 West Dayton Madison WI 53706”
- Play critical roles in many data integration tasks
 - Schema matching, data matching, information extraction
- This chapter
 - Defines the string matching problem
 - Describes popular similarity measures
 - Discusses how to apply such measures to match a large number of strings

Outline

- Problem description
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size filtering, prefix filtering, position filtering, bound filtering

Problem Description

- Given two sets of strings X and Y
 - Find all pairs $x \in X$ and $y \in Y$ that refer to the same real-world entity
 - We refer to (x,y) as a match
 - Example

| Set X | Set Y | Matches |
|--|---|------------------------------|
| $x_1 = \text{Dave Smith}$ $x_2 = \text{Joe Wilson}$ $x_3 = \text{Dan Smith}$ | $y_1 = \text{David D. Smith}$ $y_2 = \text{Daniel W. Smith}$ | (x_1, y_1) (x_3, y_2) |
| (a) | (b) | (c) |

- Two major challenges: *accuracy & scalability*

Accuracy Challenges

- Matching strings often appear quite differently
 - Typing and OCR (Optical Character recognition) errors: *David Smith* vs. *Davod Smith*
 - Different formatting conventions: *10/8* vs. *Oct 8*
 - Custom abbreviation, shortening, or omission:
Daniel Walker Herbert Smith vs. *Dan W. Smith*
 - Different names, nick names: *William Smith* vs. *Bill Smith*
 - Shuffling parts of strings: *Dept. of Computer Science*, UW-Madison vs. *Computer Science Dept., UW-Madison*

Accuracy Challenges

- Solution:
 - Use a similarity measure $s(x,y) \in [0,1]$
 - ❖ The higher $s(x,y)$, the more likely that x and y match
 - Declare x and y matched if $s(x,y) \geq t$
- Distance measure/cost measure have also been used
 - Same concept
 - But smaller values → higher similarities

Scalability Challenges

- Applying $s(x,y)$ to all pairs is impractical
 - Quadratic in size of data
- Solution: apply $s(x,y)$ to only most promising pairs, using a method *FindCands*
 - For each string $x \in X$
use method FindCands to find a candidate set $Z \subseteq Y$
for each string $y \in Z$
if $s(x,y) \geq t$ then return (x,y) as a matched pair
- We discuss ways to implement FindCands later

Outline

- Problem description
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size filtering, prefix filtering, position filtering, bound filtering

Edit Distance

- Also known as Levenshtein distance
- $d(x,y)$ computes minimal cost of transforming x into y , using a sequence of operators, each with cost 1
 - delete a character
 - insert a character
 - substitute a character with another
- Example: $x = David\ Smiths$, $y = Davidd\ Simth$,
 $d(x,y) = 4$, using following sequence
 - ❖ Inserting a character d (after David)
 - ❖ Substituting m by i
 - ❖ Substituting i by m
 - ❖ Deleting the last character of x, which is s

Edit Distance

- Models common editing mistakes
 - Inserting an extra character, swapping two characters, etc.
 - So smaller edit distance → higher similarity
- Can be converted into a similarity measure
$$s(x,y) = 1 - d(x,y) / [\max(\text{length}(x), \text{length}(y))]$$
- Example
 - ❖ $s(\text{David Smiths}, \text{Davidd Simth}) = 1 - 4 / \max(12, 12) = 0.67$

Computing Edit Distance using Dynamic Programming

- Define $x = x_1 x_2 \cdots x_n$, $y = y_1 y_2 \cdots y_m$
 - $d(i,j)$ = edit distance between $x_1 x_2 \cdots x_i$ and $y_1 y_2 \cdots y_j$,
the i-th and j-th prefixes of x and y
- Recurrence equations

$$d(i,j) = \min \begin{cases} d(i-1, j-1) & \text{if } x_i = y_j \text{ // copy} \\ d(i-1, j-1) + 1 & \text{if } x_i \neq y_j \text{ // substitute} \\ d(i-1, j) + 1 & \text{// delete } x_i \\ d(i, j-1) + 1 & \text{// insert } y_j \end{cases}$$

$$d(i,j) = \min \begin{cases} d(i-1, j-1) + c(x_i, y_j) & \text{// copy or substitute} \\ d(i-1, j) + 1 & \text{// delete } x_i \\ d(i, j-1) + 1 & \text{// insert } y_j \end{cases}$$

$$c(x_i, y_j) = \begin{cases} 0 & \text{if } x_i = y_j, \\ 1 & \text{otherwise} \end{cases}$$

Example

- $x = dva, y = dave$

| | | y0 | y1 | y2 | y3 | y4 |
|----|---|----|----|----|----|----|
| | | d | a | v | e | |
| x0 | | 0 | 1 | 2 | 3 | 4 |
| x1 | d | 1 | 0 | 1 | | |
| x2 | v | 2 | | | | |
| x3 | a | 3 | | | | |

| | | y0 | y1 | y2 | y3 | y4 |
|----|---|----|----|----|----|----|
| | | d | a | v | e | |
| x0 | | 0 | 1 | 2 | 3 | 4 |
| x1 | d | 1 | 0 | 1 | 2 | 3 |
| x2 | v | 2 | 1 | 1 | 2 | |
| x3 | a | 3 | 2 | 1 | 2 | 2 |

$x = d - v a$
 | | | |
 $y = d a v e$

substitute a with e
 insert a (after d)

- Cost of dynamic programming is $O(|x| |y|)$
- Termination: $d(n,m)$ is the distance

Needleman-Wunch Measure

- Generalizes Levenshtein edit distance
- Basic idea
 - defines notion of *alignment* between x and y
 - assigns *score* to alignment
 - return the alignment with highest score
- Alignment: set of correspondences between characters of x and y, allowing for gaps

d - - v a
| ||
d e e v e

Scoring an Alignment

- Use a *score matrix* and a *gap penalty*
- Example

d -- v a
| | |
d e e v e

| | d | v | a | e |
|----------|----------|----------|----------|----------|
| d | 2 | -1 | -1 | -1 |
| v | -1 | 2 | -1 | -1 |
| a | -1 | -1 | 2 | -1 |
| e | -1 | -1 | -1 | 2 |

$$c_g = 1$$

- alignment score = sum of scores of all correspondences - sum of penalties of all gaps
 - ❖ e.g., for the above alignment, it is 2 (for d-d) + 2 (for v-v) -1 (for a-e) -2 (for gap) = 1
 - ❖ this is the alignment with the highest score, it is returned as the Needleman-Wunch score for dva and deeve.

Needleman-Wunch Generalizes Levenshtein in Three Ways

- Computes similarity scores instead of distance values
- Generalizes edit costs into a score matrix
 - allowing for more fine-grained score modeling
 - e.g., $\text{score}(o,0) > \text{score}(a,0)$
 - e.g., different amino-acid pairs may have different semantic distance
- Generalizes insertion and deletion into gaps, and generalizes their costs from 1 to C_g

Computing Needleman-Wunch Score with Dynamic Programming

$$s(i,j) = \max \begin{cases} s(i-1,j-1) + c(x_i,y_j) \\ s(i-1,j) - c_g \\ s(i,j-1) - c_g \end{cases}$$

$$s(0,j) = -j c_g$$

$$s(i,0) = -i c_g$$

$c(x_i,y_j)$

| | d | v | a | e |
|---|----|----|----|----|
| d | 2 | -1 | -1 | -1 |
| v | -1 | 2 | -1 | -1 |
| a | -1 | -1 | 2 | -1 |
| e | -1 | -1 | -1 | 2 |

$$c_g = 1$$

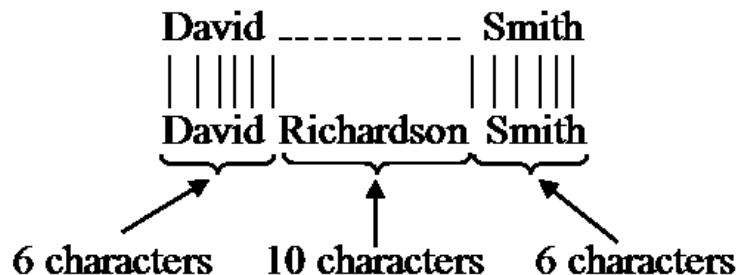
| | | | | | | |
|---|----|----|----|----|----|----|
| | | d | e | e | v | e |
| | 0 | -1 | -2 | -3 | -4 | -5 |
| d | -1 | 2 | 1 | 0 | -1 | -2 |
| v | -2 | 1 | 1 | 0 | 2 | 1 |
| a | -3 | 0 | 0 | 0 | 1 | 1 |

d - - v a
d e e v e

The Affine Gap Measure: Motivation

- An extension of Needleman-Wunch that handles longer gap more gracefully
- E.g., “*David Smith*” vs. “*David R. Smith*”
 - Needleman-Wunch well suited here
 - opens gap of length 2 right after “David”

- E.g.,



- Needleman-Wunch not well suited here, gap cost is too high
- If each char correspondence has score 2, $c_g = 1$, then the above has score $6*2 - 10 = 2$

The Affine Gap Measure: Solution

- In practice, gaps tend to be longer than 1 character
- Assigning same penalty to each character unfairly punishes long gaps
- Solution: define cost of opening a gap vs. cost of continuing the gap
 - $\text{cost}(\text{gap of length } k) = c_0 + (k-1)c_r$
 - c_0 = cost of opening gap
 - c_r = cost of continuing gap, $c_0 > c_r$
- E.g., “David Smith” vs. “David Richardson Smith”
 - $c_0 = 1, c_r = 0.5$, alignment cost = $6*2 - 1 - 9*0.5 = 6.5$

Computing Affine Gap Score using Dynamic Programming

$$s(i,j) = \max \{M(i,j), I_x(i,j), I_y(i,j)\}$$

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + c(x_i, y_j) \\ I_x(i-1,j-1) + c(x_i, y_j) \\ I_y(i-1,j-1) + c(x_i, y_j) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1,j) - c_o \\ I_x(i-1,j) - c_r \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i,j-1) - c_o \\ I_y(i,j-1) - c_r \end{cases}$$

- The notes detail how these equations are derived

The Smith-Waterman Measure: Motivation

- Previous measures consider *global alignments*
 - attempt to match *all characters of x* with *all characters of y*
- Not well suited for some cases
 - e.g., “Prof. John R. Smith, Univ of Wisconsin” and “John R. Smith, Professor”
 - similarity score here would be quite low
- Better idea: find *two substrings of x and y* that are most similar
 - e.g., find “John R. Smith” in the above case → *local alignment*

The Smith-Waterman Measure: Basic Ideas

- Find the best local alignment between x and y , and return its score as the score between x and y
- Makes two key changes to Needleman-Wunch
 - allows the match to restart at any position in the strings (no longer limited to just the first position)
 - ❖ if global match dips below 0, then ignore prefix and restart the match
 - after computing matrix using recurrence equation, retracing the arrows from the largest value in matrix, rather than from lower-right corner
 - ❖ this effectively ignores suffixes if the match they produce is not optimal
 - ❖ retracing ends when we meet a cell with value 0 → start of alignment

Computing Smith-Waterman Score using Dynamic Programming

$$s(i,j) = \max \begin{cases} 0 \\ s(i-1,j-1) + c(x_i,y_j) \\ s(i-1,j) - c_g \\ s(i,j-1) - c_g \end{cases}$$

$$s(0,j) = 0$$

$$s(i,0) = 0$$

| | | d | a | v | e |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| d | 0 | 0 | 2 | 1 | 0 |
| a | 0 | 0 | 1 | 4 | 3 |
| v | 0 | 0 | 1 | 4 | 3 |
| e | 0 | 2 | 1 | 3 | 3 |

| | d | e | e | v | e |
|---|----|----|----|----|----|
| 0 | -1 | -2 | -3 | -4 | -5 |
| d | -1 | 2 | 1 | 0 | -1 |
| v | -2 | 1 | 1 | 0 | 2 |
| a | -3 | 0 | 0 | 0 | 1 |

The Jaro Measure

- Mainly for comparing short strings, e.g., first/last names
- To compute $jaro(x,y)$
 - find common characters x_i and y_j such that $x_i = y_j$ and $|i-j| \leq \min\{|x|, |y|\}/2$
 - intuitively, common characters are identical and positionally “close to each other”
 - if the i -th common character of x does not match the i -th common character of y , then we have a transposition
 - return $jaro(x,y) = 1 / 3[c/|x| + c/|y| + (c - t/2)/c]$, where c is the number of common characters, and t is the number of transpositions

The Jaro Measure: Examples

- $x = \text{jon}$, $y = \text{john}$
 - $c = 3$ because the common characters are j, o, and n
 - $t = 0$
 - $\text{jaro}(x,y) = 1 / 3(3/3 + 3/4 + 3/3) = 0.917$
 - contrast this to 0.75, the sim score of x and y using edit distance
- $x = \text{jon}$, $y = \text{ojhn}$
 - common char sequence in x is jon
 - common char sequence in y is ojn
 - $t = 2$
 - $\text{jaro}(x,y) = 0.81$

The Jaro-Winkler Measure

- Captures cases where x and y have a low Jaro score, but share a prefix → still likely to match
- Computed as
 - $\text{jaro-winkler}(x,y) = (1 - PL \cdot PW) * \text{jaro}(x,y) + PL \cdot PW$
 - PL = length of the longest common prefix
 - PW is a weight given to the prefix

Outline

- Problem description
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size filtering, prefix filtering, position filtering, bound filtering

Set-based Similarity Measures

- View strings as sets or multi-sets of tokens
- Use set-related properties to compute similarity scores
- Common methods to generate tokens
 - consider words delimited by space
 - ❖ possibly stem the words (depending on the application)
 - ❖ remove common stop words (e.g., the, and, of)
 - ❖ e.g., given “david smith” → generate tokens “david” and “smith”
 - consider q-grams, substrings of length q
 - ❖ e.g., “david smith” → the set of 3-grams are {##d, #da, dav, avi, ..., h##}
 - ❖ special character # is added to handle the start and end of string

The Overlap Measure

- Let B_x = set of tokens generated for string x
- Let B_y = set of tokens generated for string y
- $O(x,y) = |B_x \cap B_y|$
 - returns the number of common tokens
- E.g., $x = \text{dave}$, $y = \text{dav}$
 - $B_x = \{\#\text{d}, \text{da}, \text{av}, \text{ ve}, \text{e}\#\}$, $B_y = \{\#\text{d}, \text{da}, \text{av}, \text{ v}\#\}$
 - $O(x,y) = 3$

The Jaccard Measure

- $J(x,y) = |B_x \cap B_y| / |B_x \cup B_y|$
- E.g., $x = \text{dave}$, $y = \text{dav}$
 - $B_x = \{\#\text{d}, \text{da}, \text{av}, \text{ ve}, \text{e}\#\}$, $B_y = \{\#\text{d}, \text{da}, \text{av}, \text{v}\#\}$
 - $J(x,y) = 3/6$
- Very commonly used in practice

The TF/IDF Measure: Motivation

- uses the TF/IDF notion commonly used in IR
 - two strings are similar if they share distinguishing terms
 - e.g., $x = \text{Apple Corporation, CA}$
 $y = \text{IBM Corporation, CA}$
 $z = \text{Apple Corp}$
 - $s(x,y) > s(x,z)$ using edit distance or Jaccard measure, so x is matched with $y \rightarrow$ incorrect
 - TF/IDF measure can recognize that Apple is a distinguishing term, whereas Corporation and CA are far more common \rightarrow correctly match x with z

Term Frequencies and Inverse Document Frequencies

- Assume x and y are taken from a *collection of strings*
- Each string is converted into a *bag of terms called a document*
- Define term frequency $tf(t,D) = \text{number of times term } t \text{ appears in document } D$
- Define inverse document frequency $\text{idf}(t) = N / N_d$,
number of documents in collection divided by number of documents that contain t
 - note: in practice, $\text{idf}(t)$ is often defined as $\log(N / N_d)$, here we will use the above simple formula to define $\text{idf}(t)$

Example

$$x = aab \Rightarrow B_x = \{a, a, b\}$$

$$y = ac \Rightarrow B_y = \{a, c\}$$

$$z = a \Rightarrow B_z = \{a\}$$

$$\text{tf}(a, x) = 2 \quad \text{idf}(a) = 3/3 = 1$$

$$\text{tf}(b, x) = 1 \quad \text{idf}(b) = 3/1 = 3$$

$$\dots \quad \text{idf}(c) = 3/1 = 3$$

$$\text{tf}(c, z) = 0$$

Feature Vectors

- Each document d is converted into a feature vector \mathbf{v}_d
- \mathbf{v}_d has a feature $v_d(t)$ for each term t
 - value of $v_d(t)$ is a function of TF and IDF scores
 - here we assume $v_d(t) = \text{tf}(t,d) * \text{idf}(t)$

$$x = aab \Rightarrow B_x = \{a, a, b\}$$

$$y = ac \Rightarrow B_y = \{a, c\}$$

$$z = a \Rightarrow B_z = \{a\}$$

$$\text{tf}(a, x) = 2 \quad \text{idf}(a) = 3/3 = 1$$

$$\text{tf}(b, x) = 1 \quad \text{idf}(b) = 3/1 = 3$$

$$\dots \quad \text{idf}(c) = 3/1 = 3$$

$$\text{tf}(c, z) = 0$$

| | a | b | c |
|----------------|---|---|---|
| \mathbf{v}_x | 2 | 3 | 0 |
| \mathbf{v}_y | 3 | 0 | 3 |
| \mathbf{v}_z | 3 | 0 | 0 |

TF/IDF Similarity Score

- Let p and q be two strings, and T be the set of all terms in the collection
- Feature vectors v_p and v_q are vectors in the $|T|$ -dimensional space where each dimension corresponds to a term
- TF/IDF score of p and q is the cosine of the angle between v_p and v_q
 - $s(p,q) = \sum_{t \in T} v_p(t) * v_q(t) / [\sqrt{\sum_{t \in T} v_p(t)^2} * \sqrt{\sum_{t \in T} v_q(t)^2}]$

TF/IDF Similarity Score

- Score is high if strings share many frequent terms
 - terms with high TF scores
- Unless these terms are common in other strings
 - i.e., they have low IDF scores
- Dampening TF and IDF as commonly done in practice
 - use $v_d(t) = \log(tf(t,d) + 1) * \log(idf(t))$ instead of $v_d(t) = tf(t,d) * idf(t)$
- Normalizing feature vectors
 - $v_d(t) = v_d(t) / \sqrt{\sum_{t \in T} v_d(t)^2}$

Outline

- Problem description
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size filtering, prefix filtering, position filtering, bound filtering

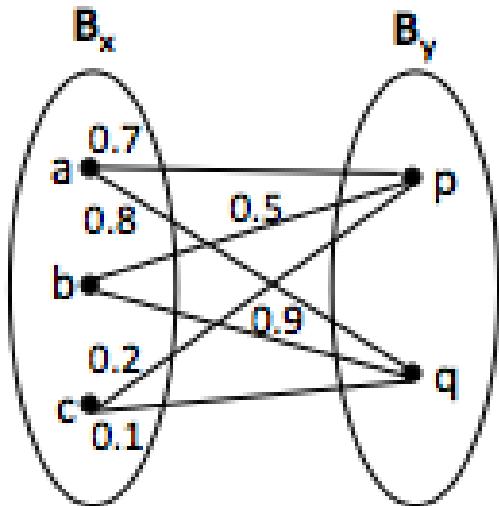
Generalized Jaccard Measure

- Jaccard measure
 - considers overlapping tokens in both **x** and **y**
 - a token from **x** and a token from **y** **must be identical** to be included in the set of overlapping tokens
 - this can be **too restrictive** in certain cases
- Example:
 - matching taxonomic nodes that describe companies
 - “Energy & Transportation” vs. “Transportation, Energy, & Gas”
 - in theory Jaccard is well suited here, in practice Jaccard may not work well if tokens are commonly misspelled
 - ❖ e.g., energy vs. energy
 - generalized Jaccard measure can help such cases

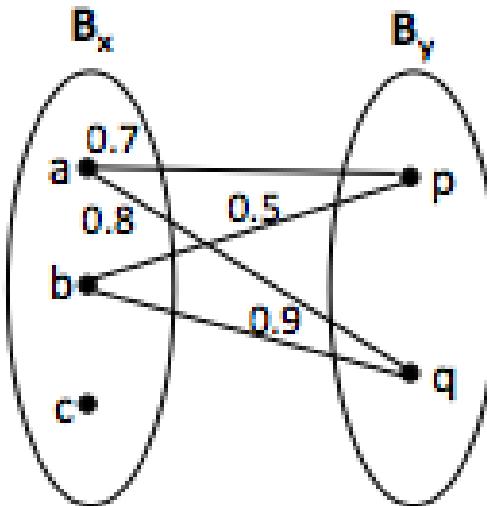
Generalized Jaccard Measure

- Let $B_x = \{x_1, \dots, x_n\}$, $B_y = \{y_1, \dots, y_m\}$
- Step 1: find token pairs that will be in the “softened” overlap set
 - apply a similarity measure s to compute sim score for each pair (x_i, y_j)
 - keep only those score \geq a given threshold α , this forms a bipartite graph G
 - find the maximum-weight matching M in G
- Step 2: return normalized weight of M as generalized Jaccard score
 - $GJ(x,y) = \sum_{(x_i,y_j) \in M} s(x_i, y_j) / (|B_x| + |B_y| - |M|)$

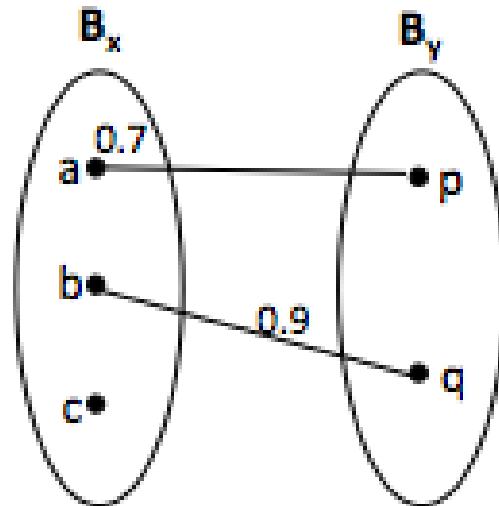
An Example



(a)



(b)



(c)

- Generalized Jaccard score: $(0.7 + 0.9)/(3 + 2 - 2) = 0.53$

The Soft TF/IDF Measure

- Similar to generalized Jaccard measure, except that it uses TF/IDF measure as the “higher-level” sim measure
 - e.g., “Apple Corporation, CA”, “IBM Corporation, CA”, and “Aple Corp”, with Apple being mispelt in the last string
- Step 1: compute $\text{close}(x,y,k)$: set of all terms $t \in B_x$ that have at least one close term $u \in B_y$, i.e., $s'(t,u) \geq k$
 - s' is a basic sim measure (e.g., Jaro-Winkler), k prespecified
- Step 2: compute $s(x,y)$ as in traditional TF/IDF score, but weighing each TF/IDF component using s'
 - $s(x,y) = \sum_{t \in \text{close}(x,y,k)} v_x(t) * v_y(u^*) * s'(t,u^*)$
 - $u^* \in B_y$ maximizes $s'(t,u) \forall u \in B_y$

An Example

$x = abcd$

$y = aa'b'cd'$

$$B_x = \{a, b, c, d\}$$

$$B_y = \{a, a', b', c, d'\}$$

$$s(x,y) = v_x(a) \cdot v_y(a) \cdot 1 + \\ v_x(b) \cdot v_y(b') \cdot 0.8 + \\ v_x(c) \cdot v_y(c) \cdot 1$$

$$\text{close}(x, y, 0.75) = \{a, b, c\}$$

(a)

(b)

(c)

Outline

- Problem description
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size filtering, prefix filtering, position filtering, bound filtering

Phonetic Similarity Measures

- Match strings based on their sound, instead of appearances
- Very effective in matching names, which often appear in different ways that sound the same
 - e.g., Meyer, Meier, and Mire; Smith, Smithe, and Smythe
- Soundex is most commonly used

The Soundex Measure

- Used primarily to match surnames
 - maps a surname x into a 4-letter code
 - two surnames are judged similar if share the same code
- Algorithm to map x into a code:
 - Step 1: keep the first letter of x , subsequent steps are performed on the rest of x
 - Step 2: remove all occurrences of W and H. Replace the remaining letters with digits as follows:
 - ❖ replace B, F, P, V with 1, C, G, J, K, Q, S, X, Z with 2, D, T with 3, L with 4, M, N with 5, R with 6
 - Step 3: replace sequence of identical digits by the digit itself
 - Step 4: Drop all non-digit letters, return the first four letters as the soundex code

The Soundex Measure

- Example: x = Ashcraft
 - after Step 2: A226a13, after Step 3: A26a13, Step 4 converts this into A2613, then returns A261
 - Soundex code is padded with 0 if there is not enough digits
- Example: Robert and Rupert map into R163
- Soundex fails to map Gough and Goff, and Jawornicki and Yavornitzky
 - designed primarily for Caucasian names, but found to work well for names of many different origins
 - does not work well for names of East Asian origins
 - ❖ which uses vowels to discriminate, Soundex ignores vowels

Online tools

- <https://asecuritysite.com/forensics/simstring>
- <https://github.com/rrice/java-string-similarity>
- <https://www.npmjs.com/package/similarity>
- <http://www.mlsec.org/harry/>

Outline

- Problem description
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size filtering, prefix filtering, position filtering, bound filtering

Scalability Challenges

- Applying $s(x,y)$ to all pairs is impractical
 - Quadratic in size of data
- Solution: apply $s(x,y)$ to only most promising pairs, using a method *FindCands*
 - For each string $x \in X$
use method *FindCands* to find a candidate set $Z \subseteq Y$
for each string $y \in Z$
if $s(x,y) \geq t$ then return (x,y) as a matched pair
 - This is often called a **blocking solution**
 - Set Z is often called the umbrella set of x
- We now discuss ways to implement *FindCands*
 - using Jaccard and overlap measures for now

Inverted Index over Strings

- Converts each string $y \in Y$ into a document, builds an inverted index over these documents
- Given term t , use the index to quickly find documents of Y that contain t

Example

Set X

- 1: {lake, mendota}
- 2: {lake, monona, area}
- 3: {lake, mendota, monona, dane}

Set Y

- 4: {lake, monona, university}
- 5: {monona, research, area}
- 6: {lake, mendota, monona, area}

(a)

| Terms in Y | ID Lists |
|------------|----------|
| area | 5 |
| lake | 4, 6 |
| mendota | 6 |
| monona | 4, 5, 6 |
| research | 5 |
| university | 4 |

(b)

Limitations

- The inverted list of some terms (e.g., stop words) can be very long → costly to build and manipulate such lists
- Requires enumerating all pairs of strings that share at least one term. This set can still be very large in practice.

Size Filtering

- Retrieves only strings in Y whose sizes make them match candidates
 - given a string $x \in X$, infer a constraint on the size of strings in Y that can possibly match x
 - uses a B-tree index to retrieve only strings that satisfy size constraints
- E.g., for Jaccard measure $J(x,y) = |x \cap y| / |x \cup y|$
 - assume two strings x and y match if $J(x,y) \geq t$
 - can show that given a string $x \in X$, only strings y such that $|x|/t \geq |y| \geq |x| * t$ can possibly match x

Example

Set X

1: {lake, mendota}
2: {lake, monona, area}
3: {lake, mendota, monona, dane}

Set Y

4: {lake, monona, university}
5: {monona, research, area}
6: {lake, mendota, monona, area}

(a)

| Terms in Y | ID Lists |
|------------|----------|
| area | 5 |
| lake | 4, 6 |
| mendota | 6 |
| monona | 4, 5, 6 |
| research | 5 |
| university | 4 |

(b)

- Consider $x = \{\text{lake, mendota}\}$. Suppose $t = 0.8$
- If $y \in Y$ matches x , we must have
 - $2/0.8 = 2.5 \geq |y| \geq 2 * 0.8 = 1.6$
 - no string in Set Y satisfies this constraint \rightarrow no match

Prefix Filtering

- Key idea: if two sets share many terms → large subsets of them also share terms
- Consider overlap measure $O(x,y) = |x \cap y|$
 - if $|x \cap y| \geq k \rightarrow$ any subset $x' \subseteq x$ of size at least $|x| - (k - 1)$ must overlap y
- To exploit this idea to find pairs (x,y) such that $O(x,y) \geq k$
 - given x , construct subset x' of size $|x| - (k - 1)$
 - use an inverted index to find all y that overlap x'

Example

x: {lake, monona, area}
x'

y: {lake, mendota, monona, area}

Set X

- 1: {lake, mendota}
- 2: {lake, monona, area}
- 3: {lake, mendota, monona, dane}

Set Y

- 4: {lake, monona, university}
- 5: {monona, research, area}
- 6: {lake, mendota, monona, area}
- 7: {dane, area, mendota}

(a)

(b)

(c)

| Terms in Y | ID Lists |
|------------|----------|
| area | 5, 6, 7 |
| lake | 4, 6 |
| mendota | 6, 7 |
| monona | 4, 5, 6 |
| research | 5 |
| university | 4 |
| dane | 7 |

- Consider matching using $O(x,y) \geq 2$
- $x_1 = \{\text{lake, mendota}\}$, let $x_1' = \{\text{lake}\}$
- Use inverted index to find $\{y_4, y_6\}$ which contain at least one token in x_1'

Selecting the Subset Intelligently

- Recall that we select a subset x' of x and check its overlap with the entire set y
- We can do better by selecting a particular subset x' and checking its overlap with only a particular subset y' of y
- How?
 - impose an ordering O over the universe of all possible terms
 - ❖ e.g., in increasing frequency
 - reorder the terms in each $x \in X$ and $y \in Y$ according to O
 - refer to subset x' that contains the first n terms of x as the prefix of size n of x

Selecting the Subset Intelligently

- How? (continued)
 - can prove that if $|x \cap y| \geq k$, then x' and y' must overlap, where x' is the prefix of size $|x| - (k - 1)$ of x and y' is the prefix of size $|y| - (k - 1)$ of y (see notes)
- Algorithm
 - reorder terms in each $x \in X$ and $y \in Y$ in increasing order of their frequencies
 - for each $y \in Y$, create y' , the prefix of size $|y| - (k - 1)$ of y
 - build an inverted index over all prefixes y'
 - for each $x \in X$, create x' , the prefix of size $|x| - (k - 1)$ of x , then use above index to find all y such that x' overlaps with y'

Example

university < research
< dane < area
< mendota < monona < lake

Reordered Set X

- 1: {mendota, lake}
- 2: {area, monona, lake}
- 3: {dane, mendota, monona, lake}

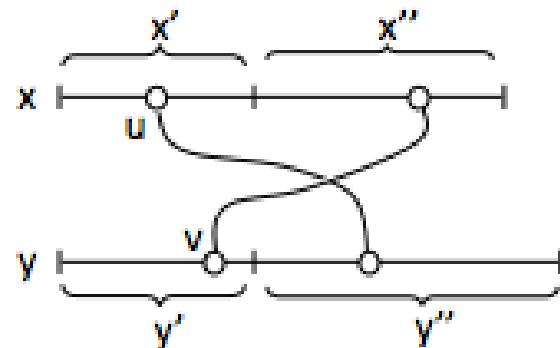
Reordered Set Y

- 4: {university, monona, lake}
- 5: {research, area, monona}
- 6: {area, mendota, monona, lake}
- 7: {dane, area, mendota}

(a)

(b)

(c)



- $x = \{\text{mendota, lake}\} \rightarrow x' = \{\text{mendota}\}$

Example

| Terms in Y | ID Lists |
|------------|----------|
| area | 5, 6, 7 |
| mendota | 6 |
| monona | 4, 6 |
| research | 5 |
| university | 4 |
| dane | 7 |

(a)

| Terms in Y | ID Lists |
|------------|----------|
| area | 5, 6, 7 |
| lake | 4, 6 |
| mendota | 6, 7 |
| monona | 4, 5, 6 |
| research | 5 |
| university | 4 |
| dane | 7 |

(b)

- See the notes for applying prefix filtering to Jaccard measure

Position Filtering

- Further limits the set of candidate matches by deriving an upper bound on the size of overlap between x and y
- e.g., $x = \{\text{dane, area, mendota, monona, lake}\}$
 $y = \{\text{research, dane, mendota, monona, lake}\}$
- Suppose we consider $J(x,y) \geq 0.8$, in prefix filtering we consider $x' = \{\text{dane, area}\}$ and $y' = \{\text{research, dane}\}$ (see notes)
- But we can do better than this. Specifically, we can prove that $O(x,y) \geq [t/(1+t)] * (|x| + |y|) = 4.44$ (see notes)
 - so can immediately discard the above (x,y) pair

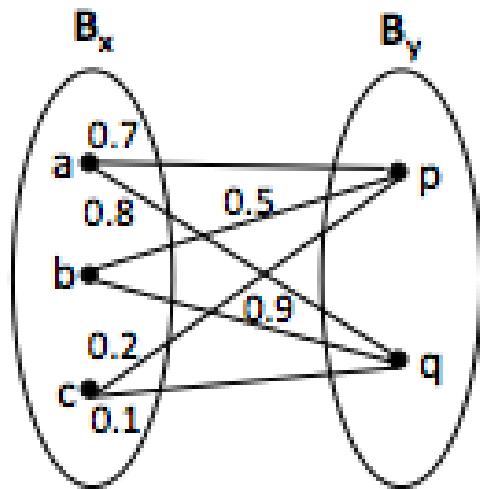
Bound Filtering

- Used to optimize the computation of generalized Jaccard similarity measure
- Recall that
 - $GJ(x,y) = \sum_{(x_i,y_j) \in M} s(x_i,y_j) / (|B_x| + |B_y| - |M|)$
- Algorithm
 - for each (x,y) compute an upper bound $UB(x,y)$ and a lower bound $LB(x,y)$ on $GJ(x,y)$
 - if $UB(x,y) \leq t \rightarrow (x,y)$ can be ignored, it is not a match
 - if $LB(x,y) \geq t \rightarrow$ return (x,y) as a match
 - otherwise compute $GJ(x,y)$

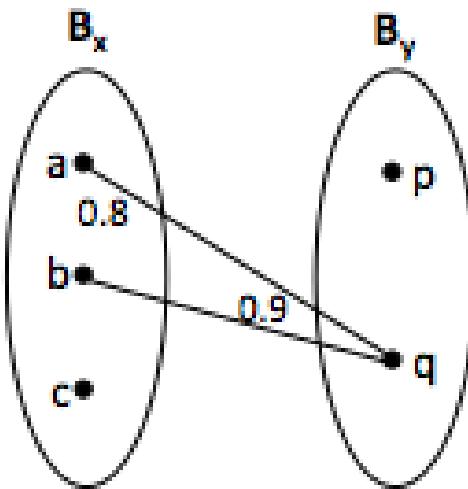
Computing $\text{UB}(x,y)$ and $\text{LB}(x,y)$

- For each $x_i \in B_x$, find $y_j \in B_y$ with the highest element-level similarity, such that $s(x_i, y_j) \geq \alpha$. Call this set of pairs S_1 .
- For each $y_j \in B_y$, find $x_i \in X$ with the highest element-level similarity, such that $s(x_i, y_j) \geq \alpha$. Call this set of pairs S_2 .
- Compute
 - $\text{UB}(x,y) = \sum_{(x_i, y_j) \in S_1 \cup S_2} s(x_i, y_j) / (|B_x| + |B_y| - |S_1 \cup S_2|)$
 - $\text{LB}(x,y) = \sum_{(x_i, y_j) \in S_1 \cap S_2} s(x_i, y_j) / (|B_x| + |B_y| - |S_1 \cap S_2|)$

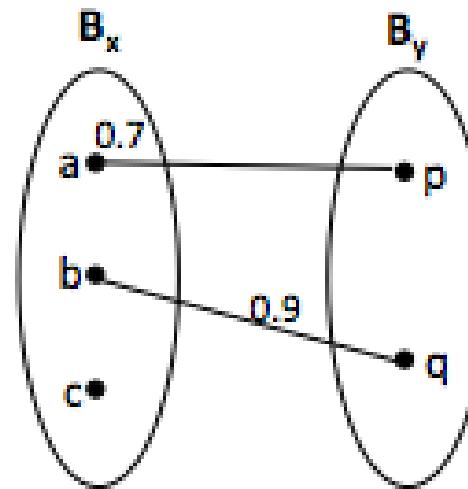
Example



(a)



(b)



(c)

- $S_1 = \{(a,q), (b,q)\}$, $S_2 = \{(a,p), (b,q)\}$
- $UB(x,y) = (0.8+0.9+0.7+0.9)/(3+2-3) = 1.65$
- $LB(x,y) = 0.9/(3+2-1) = 0.225$

Extending Scaling Techniques to Other Similarity Measures

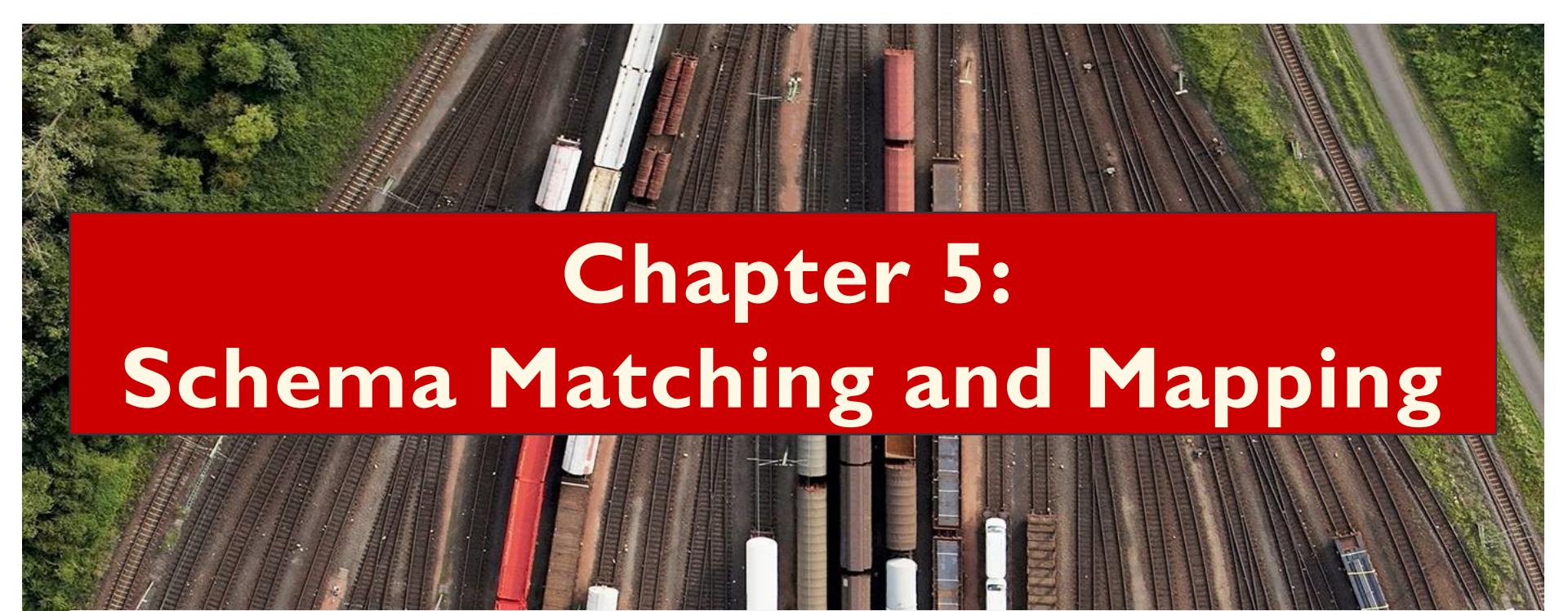
- Discussed Jaccard and overlap so far
- To extend a technique T to work for a new similarity measure $s(x,y)$
 - try to translate $s(x,y)$ into constraints on a similarity measure that already works well with T
- The notes discuss examples that involve edit distance and TF/IDF

Summary

- String matching is pervasive in data integration
- Two key challenges:
 - what similarity measure and how to scale up?
- Similarity measures
 - Sequence-based: edit distance, Needleman-Wunch, affine gap, Smith-Waterman, Jaro, Jaro-Winkler
 - Set-based: overlap, Jaccard, TF/IDF
 - Hybrid: generalized Jaccard, soft TF/IDF, Monge-Elkan
 - Phonetic: Soundex
- Scaling up string matching
 - Inverted index, size/prefix/position/bound filtering

Acknowledgment

- Slides in the scalability section are adapted from
<http://pike.psu.edu/p2/wisc09-tech.ppt>



Chapter 5: **Schema Matching and Mapping**



**PRINCIPLES OF
DATA INTEGRATION**

ANHAI DOAN ALON HALEVY ZACHARY IVES

Introduction

- We have described
 - formalisms to specify source descriptions
 - algorithms that use these descriptions to reformulate queries
- How to create the source descriptions?
 - often begin by creating semantic matches
 - ❖ name = title, location = concat(city, state, zipcode)
 - then elaborate matches into semantic mappings
 - ❖ e.g., structured queries in a language such as SQL
- Schema matching and mapping are often quite difficult
- This chapter describes matching and mapping tools
 - that can significantly reduce the time it takes for the developer to create matches and mappings

Outline

- Problem definition, challenges, and overview
- Schema matching
 - Matchers
 - Combining match predictions
 - Enforcing domain integrity constraints
 - Match selector
 - Reusing previous matches
 - Many-to-many matches
- Schema mapping

Semantic mappings

DVD-VENDOR

Movies(id, title, year)

Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)

Locations(lid, name, taxRate)

AGGREGATOR

Items(name, releaseInfo, classification, price)

- Let **S** and **T** be two relational schemas
 - refer to the attributes and tables of **S** and **T** as their elements
- A semantic mapping is a query expression that relates a schema **S** with a schema **T**
 - the following mapping shows how to obtain *Movies.title*
 - ❖ **SELECT name as title**
 - FROM Items**

Semantic mappings

DVD-VENDOR

Movies(id, title, year)

Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)

Locations(lid, name, taxRate)

AGGREGATOR

Items(name, releaseInfo, classification, price)

- More examples of semantic mappings

- the following mapping shows how to obtain *Items.price*
 - ❖ **SELECT (basePrice * (1 + taxRate)) AS price**
FROM Products, Locations
WHERE Products.saleLocID = Locations.lid
- the following mapping shows how to obtain an entire tuple for Items table of AGGREGATOR
 - ❖ **SELECT title AS name, releaseDate AS releaseInfo, rating AS classification, basePrice * (1 + taxRate) AS price**
FROM Movies, Products, Locations
WHERE Movies.id = Products.mid AND Products.saleLocID = Locations.lid

Example of the need to create semantic mappings for DI systems

DVD-VENDOR

Movies(id, title, year)

Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)

Locations(lid, name, taxRate)

AGGREGATOR

Items(name, releaseInfo, classification, price)

- Consider building a data integration system
 - over two sources, with schemas **DVD-VENDOR** & **BOOK-VENDOR**
 - assume the mediated schema is **AGGREGATOR**
- If we use *Global-as-View* approach to relate schemas
 - must describe Items in **AGGREGATOR** as a query over sources
 - to do this, create semantic mappings m_1 and m_2 that specify how to obtain tuples of Items from **DVD-VENDOR** and **BOOK-VENDOR**, respectively, then return semantic mapping $(m_1 \text{ UNION } m_2)$ as the GAV description of Items table.

Example of the need to create semantic mappings for DI systems

DVD-VENDOR

Movies(id, title, year)

Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)

Locations(lid, name, taxRate)

AGGREGATOR

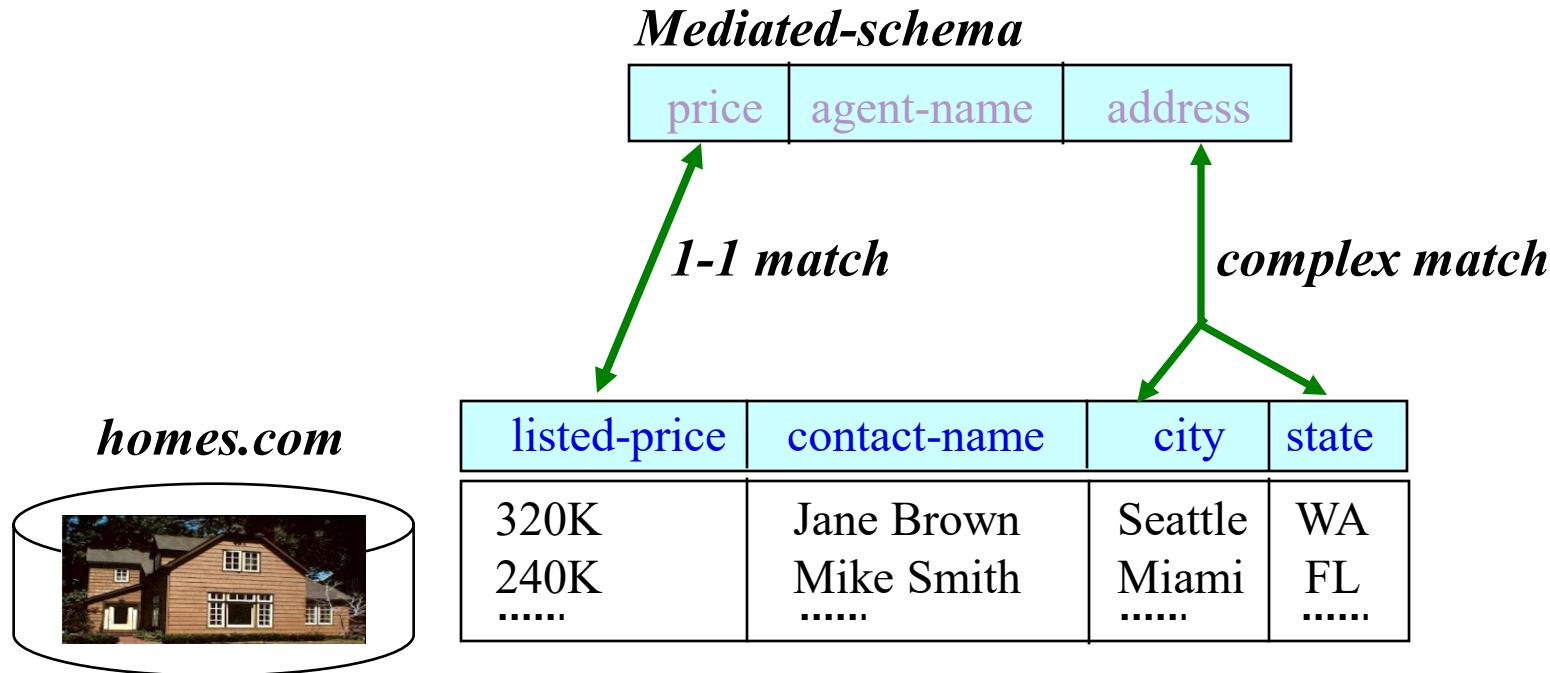
Items(name, releaseInfo, classification, price)

- If we use *Local-as-View* approach to relate schemas
 - for each table in **DVD-VENDOR** and **BOOK-VENDOR**, must create a semantic mapping that specifies how to obtain tuples for that table from schema **AGGREGATOR** (i.e., from table Items)
- If we use GLAV approach
 - there are semantic mappings going in both directions

Semantic matches

- A semantic match relates a set of elements in a schema **S** to a set of elements in schema **T**
 - without specifying in detail (to the level of SQL queries) the exact nature of the relationship (as in semantic mappings)
- *One-to-one matches*
 - `Movies.title = Items.name`
 - `Products.rating = Items.classification`
- *One-to-many matches*
 - `Items.price = Products.basePrice * (1 + Locations.taxRate)`
- *Other types of matches*
 - many-to-one, many-to-many

Semantic Matches between Schemas



Relationship between Schema Matching and Mapping

- To create source description
 - often start by creating semantic matches
 - then elaborate matches into mappings
- Why start with semantic matches?
 - they are often easier to elicit from designers
 - ❖ e.g., can specify $\text{price} = \text{basePrice} * (1 + \text{taxRate})$ from domain knowledge
- Why the need to elaborate matches into mappings?
 - matches often specify functional relationships
 - but they cannot be used to obtain data instances
 - need SQL queries, that is, mappings for that purpose
 - so matches need to be elaborated into mappings

Relationship between Schema Matching and Mapping

- Example: elaborate the match
 - $\text{price} = \text{basePrice} * (1 + \text{taxRate})$
 - into mapping
 - ```
SELECT (basePrice * (1 + taxRate)) AS price
FROM Product, Location
WHERE Product.saleLocID = Location.lid
```
- Another reason for starting with matches
  - break the long process in the middle
  - allow designer to verify and correct the matches
  - thus reducing the complexity of the overall process

# Challenges of Schema Matching and Mapping

---

- Matching and mapping systems must reconcile semantic heterogeneity between the schemas
- Such semantic heterogeneity arise in many ways
  - same concept, but different names for tables and attributes
    - ❖ rating vs classification
  - multiple attributes in 1 schema relate to 1 attribute in the other
    - ❖ basePrice and taxRate relate to price
  - tabular organization of schemas can be quite different
    - ❖ one table in AGGREGATOR vs three tables in DVD-VENDOR
  - coverage and level of details can also differ significantly
    - ❖ DVD-VENDOR also models releaseDate and releaseCompany

# Challenges of Schema Matching and Mapping

---

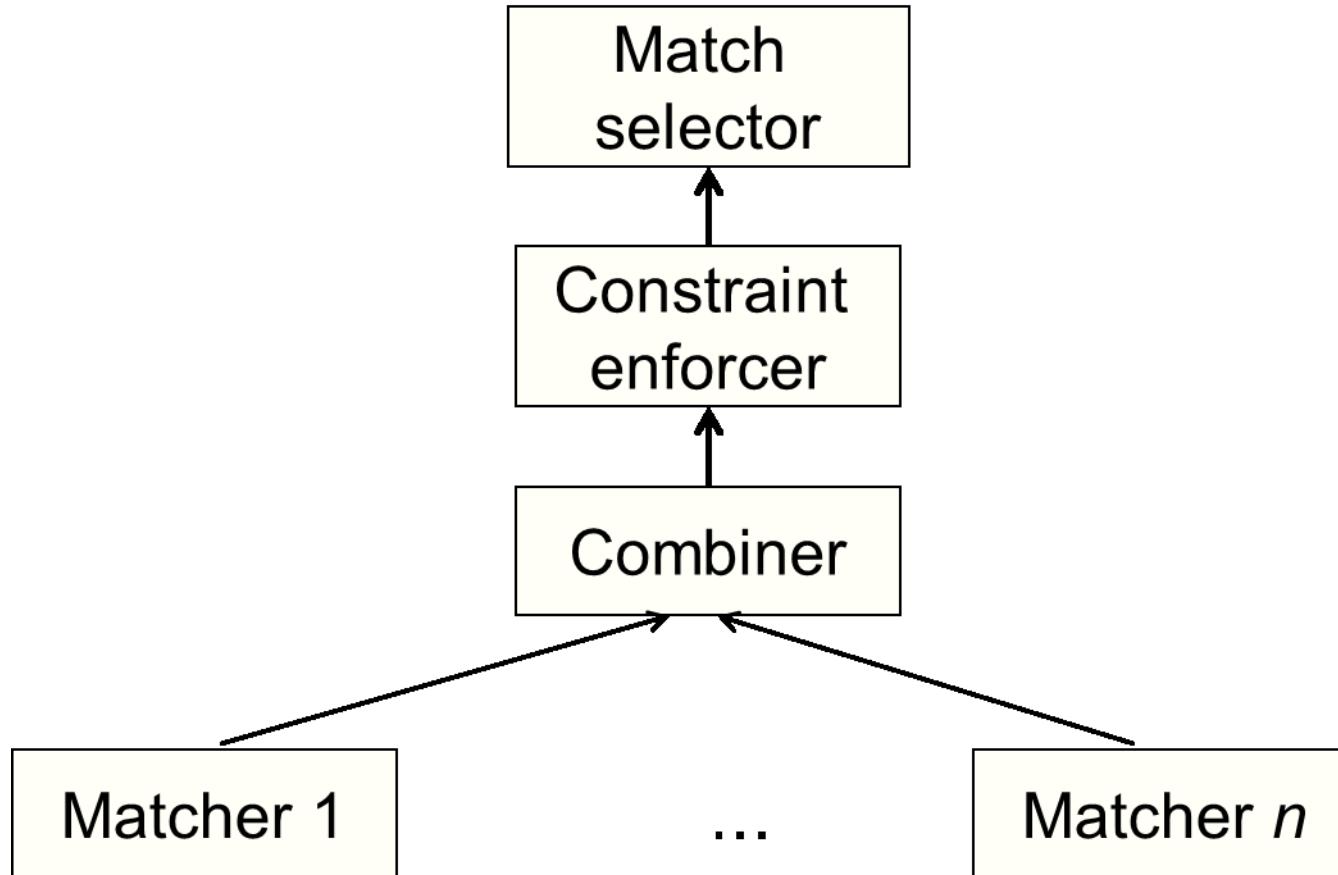
- Why do we have semantic heterogeneity?
  - schemas are created by different people whose states and styles are different
  - disparate databases are rarely created for exact same purposes
- Why reconciling semantic heterogeneity is hard
  - the semantics is not fully captured in the schemas
  - schema clues can be unreliable
  - intended semantics can be subjective
  - correctly combining the data is difficult
- Standard is not a solution!
  - works for limited use cases where number of attributes is small and there is strong incentive to agree on them

# Overview of Matching Systems

---

- For now we consider only 1-1 matching systems
  - will discuss finding complex matches later
- Key observation: need multiple heuristics / types of information to maximize matching accuracy
  - e.g., by **matching the names**, can infer that *releaseInfo = releaseDate* or *releaseInfo = releaseCompany*, but do not know which one
  - by **matching the data values**, can infer that *releaseInfo = releaseDate* or *releaseInfo = year*, but do not know which one
  - by combining both, can infer that *releaseInfo = releaseDate*

# Matching System Architecture



# Overview of Mapping Systems

---

- **Input:** matches, output: actual mappings
- **Key challenge:** find how tuples from one source can be transformed and combined to produce tuples in the other
  - which data transformation to apply?
  - which joins to take?
  - and many more possible decisions

# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - *Matchers*
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

# Matchers

---

- schemas → similarity matrix
- **Input:** two schemas **S** and **T**, plus any possibly helpful auxiliary information (e.g., data instances, text descriptions)
- **Output:** *sim matrix* that assigns to each element pair of **S** and **T** a number in [0,1] predicting whether the pair match
- Numerous matchers have been proposed
- We describe a few, in two classes:  
*name matchers* and *data matchers*

# Name-Based Matchers

---

- Use string matching techniques
  - e.g., edit distance, Jaccard, Soundex, etc.
- Often have to pre-process names
  - split them using certain delimiters
    - ❖ e.g., saleLocID → sale, Loc, ID
  - expand known abbreviations or acronyms
    - ❖ loc → location, cust → customer
  - expand a string with synonyms / hypernyms
    - ❖ add cost to price, expand product into book, dvd, cd
  - remove stop words
    - ❖ in, at, and

# Example

DVD-VENDOR

**Movies**(id, title, year)

**Products**(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)

**Locations**(lid, name, taxRate)

AGGREGATOR

**Items**(name, releaseInfo, classification, price)

**name-base matcher:**

name = <name: 1, title: 0.2>

releaseInfo = <releaseDate: 0.5, releaseCompany: 0.5>

price = <basePrice: 0.8>

# Instance-Based Matchers

---

- When schemas come with data instances, these can be extremely helpful in deciding matches
- Many instance-based matchers have been proposed
- Some of the most popular
  - recognizers
    - ❖ use dictionaries, regexes, or simple rules
  - overlap matchers
    - ❖ examine the overlap of values among attributes
  - classifiers
    - ❖ use learning techniques

# Building Recognizers

---

- Use dictionaries, regexes, or rules to recognize data values of certain kinds of attributes
- Example attributes for which recognizers are well suited
  - country names, city names, US states
  - person names (can use dictionaries of last and first names)
  - color, rating (e.g., G, PG, PG-13, etc.), phone, fax, soc sec
  - genes, protein, zip codes

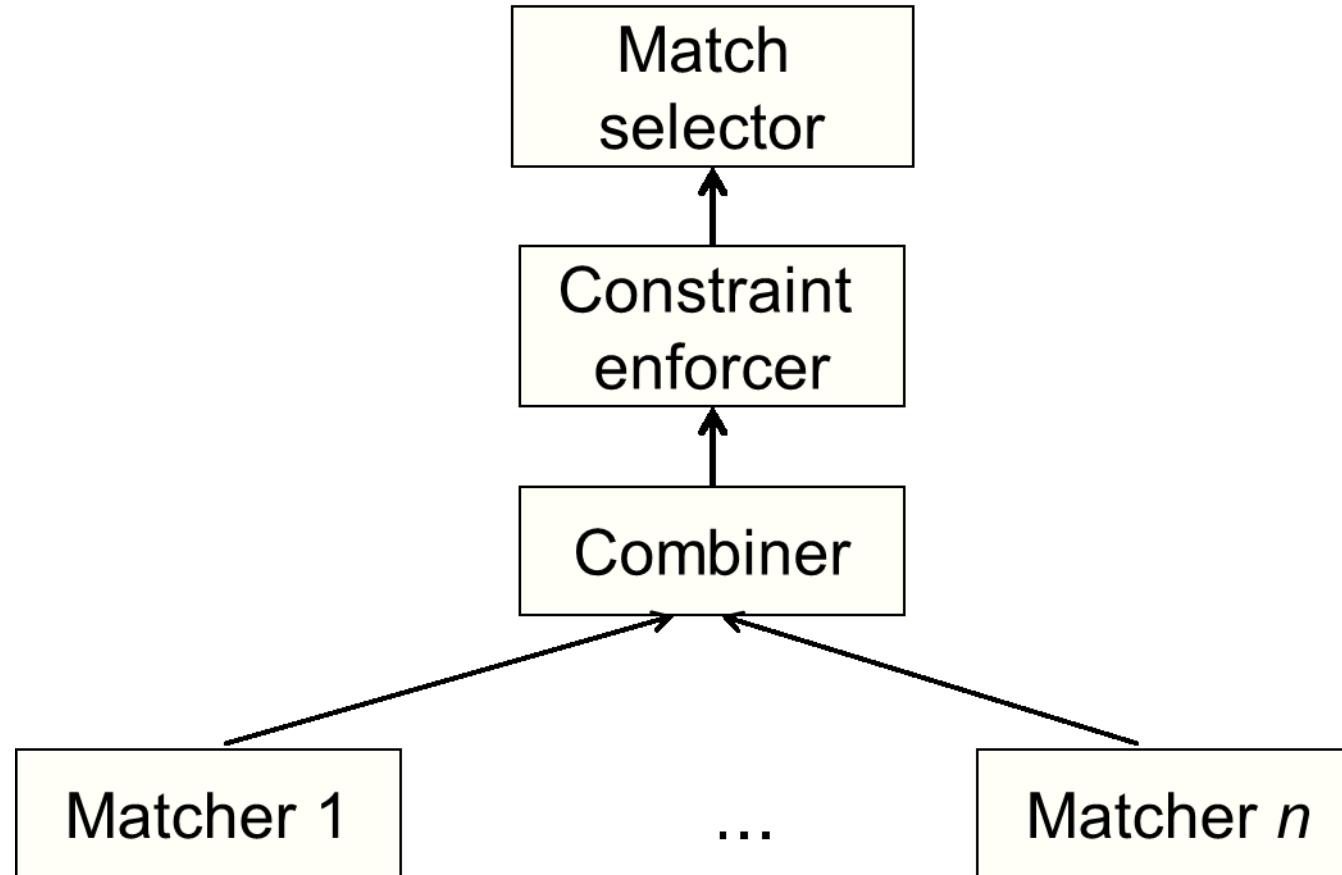
# Measuring the Overlap of Values

- Typically applies to attributes whose values are drawn from some finite domain
  - e.g., movie ratings, movie titles, book titles, country names
- Jaccard measure is commonly used
- Example:
  - use Jaccard measure to build a data-based matcher between **DVD-VENDOR** and **AGGREGATOR**
  - **AGGREGATOR.name** refers to *DVD titles*,  
**DVD-VENDOR.name** refers to *sale locations*,  
**DVD-VENDOR.title** refers to *DVD titles*  
→ low score for (name, name), high score for (name, title)

## **data-based matcher:**

name = <name: 0.2, title: 0.5>  
releaseInfo = <releaseDate: 0.7>  
classification = <rating: 0.6>  
price = <basePrice: 0.2>

# Reminder: Matching System Architecture



# Combining Match Predictions

---

- A combiner merges the sim matrices output by the matchers into a single matrix
- Simplest types of combiner return average, minimum, or maximum of scores
- For example, an average combiner computes the score between  $s_i$  and  $t_j$  to be
  - $\text{combined}(i,j) = [\sum_{m=1}^k \text{matcherScore}(m, i, j)]/k$
  - here k matchers are used
  - $\text{matcherScore}(m,i,j)$  is score between  $s_i$  and  $t_j$  as produced by the m-th matcher

# Combining Match Predictions: Another Example of the Average Combiner

DVD-VENDOR

**Movies**(id, title, year)

**Products**(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)

**Locations**(lid, name, taxRate)

AGGREGATOR

**Items**(name, releaseInfo, classification, price)

**name-base matcher:**

name = <name: 1, title: 0.2>

releaseInfo = <releaseDate: 0.5, releaseCompany: 0.5>

price = <basePrice: 0.8>

**data-based matcher:**

name = <name: 0.2, title: 0.5>

releaseInfo = <releaseDate: 0.7>

classification = <rating: 0.6>

price = <basePrice: 0.2>

**average-combiner:**

name = <name: 0.6, title: 0.5>

releaseInfo = <releaseDate: 0.6, releaseCompany: 0.25>

classification = <rating: 0.3>

price = <basePrice: 0.5>

# Combining Match Predictions

---

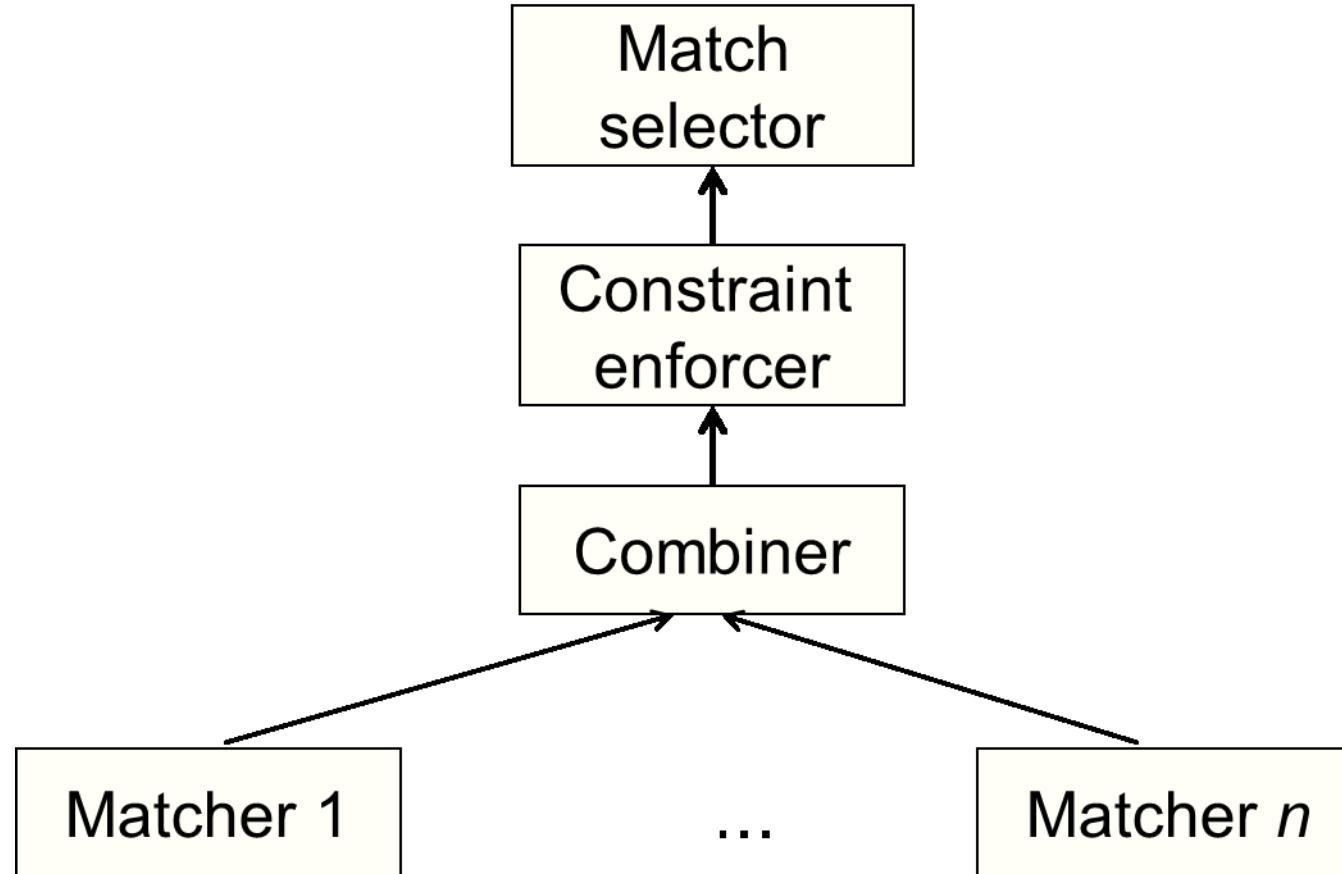
- When to use which combiner?
  - average combiner : when we do not have any reason to trust one matcher over the others
  - maximum combiner: when we trust a strong signal from matchers, i.e., if a matcher outputs a high value, we are relatively confident that the two elements match
  - minimum combiner: when we want to be more conservative
- More complex types of combiners
  - use hand-crafted scripts
    - ❖ e.g., if  $s_i$  is address, return the score of the data-based matcher otherwise, return the average score of all matchers

# Combining Match Predictions

---

- More complex types of combiners (cont.)
  - weighted-sum combiners
    - ❖ give weights to each matcher, according to its importance
    - ❖ may learn the weights from training data
    - ❖ can combine the weights in many ways: linear regression, logistic regression, etc.
  - the combiner itself can be a learner, which learns how to combine the scores of the matchers
    - ❖ e.g., decision tree, logistic regression, etc.

# Reminder: Matching System Architecture



# Enforcing Domain Integrity Constraints

---

- Designer often has knowledge that can be naturally expressed as domain integrity constraints
- Constraint enforcer exploits these to prune certain match combinations
  - searches through the space of all match combinations produced by the combiner
  - finds one combination with the highest aggregated confidence score that satisfies the constraints

# Illustrating Example

**average-combiner:**

name = <name: 0.6, title: 0.5>

releaseInfo = <releaseDate: 0.6, releaseCompany: 0.25>

classification = <rating: 0.3>

price = <basePrice: 0.5>

- Here we have four match combinations  $M_1 - M_4$ 
  - $M_1 = \{\text{name} = \text{name}, \text{releaseInfo} = \text{releaseDate}, \text{classification} = \text{rating}, \text{price} = \text{basePrice}\}$
  - For each  $M_i$ , can compute an aggregated score
    - ❖ e.g., by multiplying the individual scores,  
so  $\text{score}(M_1) = 0.6 * 0.6 * 0.3 * 0.5$

# Illustrating Example (Cont.)

---

- Suppose designer knows that
  - **AGGREGATOR.name** refers to movie titles
  - many movie titles contain at least four words
- Designer can specify a constraint such as
  - if an attribute A matches **AGGREGATOR.name**, then in any random sample of 100 data values of A, at least 10 values must contain four words or more
- Now the constraint enforcer can search for the best match combination that satisfies this constraint

# Illustrating Example (Cont.)

- How to search?
  - conceptually, check the combination with the highest score,  $M_1$ : it does not satisfy the constraint
  - check the combination with the next highest score,  $M_2$ : this one satisfies the constraint, so return it as the desired match combination
    - ❖ name = title, releaseInfo = releaseDate, classification = rating, price = basePrice
- In practice exploiting constraints is quite hard
  - must handle a variety of constraints
  - must find a way to search efficiently

# Domain Integrity Constraints

---

- Two kinds of constraints: hard and soft
- Hard constraints
  - must be enforced
  - no output match combination can violate them
- Soft constraints
  - of more heuristic nature, may actually be violated
  - we try to minimize the degree to which extent these constraints are violated
- Each constraint is associated with a cost
  - for hard constraints, the cost is  $\infty$
  - for soft constraints, the cost can be any positive number

# Example

BOOK-VENDOR

**Books**(ISBN, publisher, pubCountry, title, review)

**Inventory**(ISBN, quantity, location)

DISTRIBUTOR

**Items**(code, name, brand, origin, desc)

**InStore**(code, availQuant)

|                | Constraints                                                                                                                                                                                                                                                                                                                             | Costs |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| c <sub>1</sub> | If A = Items.code, then A is a key                                                                                                                                                                                                                                                                                                      | ∞     |
| c <sub>2</sub> | If A = Items.desc, then any random sample of 100 data instances of A must have an average length of at least 20 words                                                                                                                                                                                                                   | 1.5   |
| c <sub>3</sub> | If A <sub>1</sub> = B <sub>1</sub> , A <sub>2</sub> = B <sub>2</sub> , B <sub>2</sub> is next to B <sub>1</sub> in the schema, but A <sub>2</sub> is not next to A <sub>1</sub> , then there is no A* next to A <sub>1</sub> such that  sim(A*,B <sub>2</sub> ) – sim(A <sub>2</sub> ,B <sub>2</sub> )  ≤ t for a small pre-specified t | 2     |
| c <sub>4</sub> | If more than half of the attributes of Table U match those of Table V, then U = V                                                                                                                                                                                                                                                       | 1     |

# Domain Integrity Constraints

---

- Each constraint is specified only once by the designer
- Key requirement
  - given a constraint  $c$  and a match combination  $M$ , the enforcer must be able to efficiently decide whether  $M$  violates  $c$ , given all the available data instances of the schemas
- If the enforcer cannot detect a violation, that does not mean that the constraint indeed holds, may just mean that there is not enough data to verify
  - e.g., if all current data instances of  $A$  are distinct, that does not mean  $A$  is a key

# Searching the Space of Match Combinations

---

- There are many ways to do this, depending on the application and the types of constraints involved
- We describe here two methods
  - an adaptation of A\* search
    - ❖ guaranteed to find the optimal solution
    - ❖ but computationally more expensive
  - local propagation
    - ❖ faster
    - ❖ but performs only local optimizations

# Review: A\* Search

---

- A\* searches for a goal state within a set of states, beginning from an initial state
- Each path through the search space is assigned a cost
- A\* finds the goal state with the cheapest path from the initial state
- Performs best-first search
  - starts with the initial state, expand this state into a set of states
  - selects the state with the smallest estimated cost
  - expands the selected state into a set of states
  - again selects the state with the smallest estimated cost, etc.

# Review: A\* Search

---

- Estimated cost of a state  $n$  is  $f(n) = g(n) + h(n)$ 
  - $g(n)$  = cost of path from initial state to  $n$
  - $h(n)$  = a lower bound on cost from  $n$  to a goal state
  - $f(n)$  = a lower bound on the cost of the cheapest solution via  $n$
- A\* terminates when reaching a goal state, returning path
  - guaranteed to find a solution if exists, and the cheapest one

# Applying Constraints with A\* Search

---

- Goal: apply A\* to match schemas  $S_1$  and  $S_2$ 
  - $S_1$  has attributes  $A_1, \dots, A_n$
  - $S_2$  has attributes  $B_1, \dots, B_m$
- A state = a tuple of size n
  - the i-th element either specifies a match for  $A_i$ , or a wildcard \*, representing that the match for  $A_i$  is yet undetermined
  - a state can be viewed as a set of match combinations that are consistent with the specifications
    - ❖ e.g.,  $(B_2, *, B_1, B_3, B_2)$
  - a state is abstract if it contains wildcards, is concrete otherwise

# Applying Constraints with A\* Search

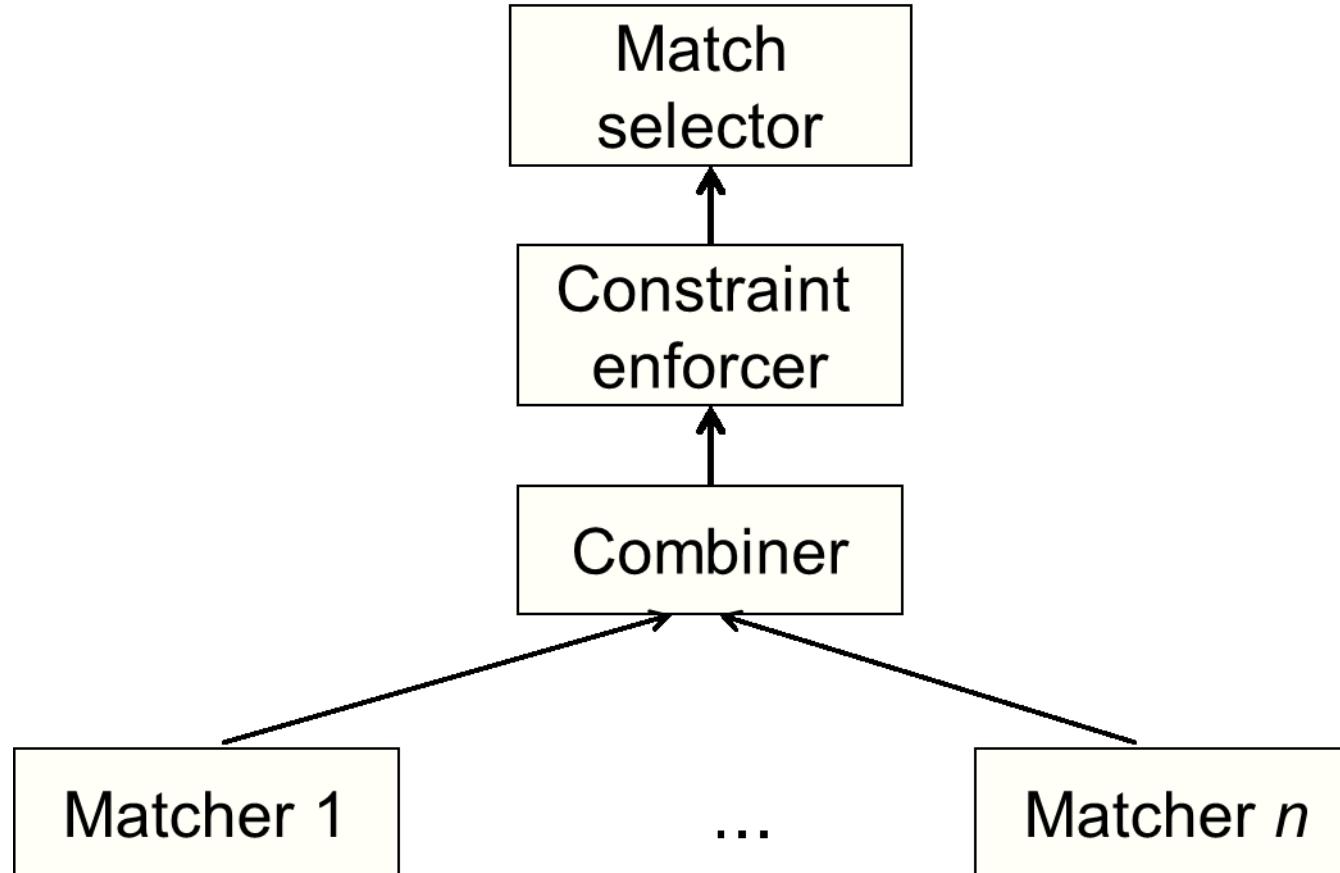
---

- Initial state:  $(*, *, \dots, *)$  = all match combinations
- Goal states: those that do not contain any \*
- Expanding states:
  - can only expand an abstract state
  - choose a \* and replace it with all possible matches
  - a key decision is which \* to expand

# Applying Constraints with A\* Search

- Cost of goal states:
  - combines our estimate of the likelihood of the combination and the degree to which it violates the constraints
  - $\text{cost}(M) = -\text{LH}(M) + \text{cost}(M, c_1) + \dots + \text{cost}(M, c_p)$
  - $\text{LH}(M) = \text{likelihood of } M \text{ according to the sim matrix} = \log \text{conf}(M)$ 
    - ❖ if  $M = (B_{k1}, \dots, B_{kn})$  then  
 $\text{conf}(M) = \text{combined}(1, k_1) * \dots * \text{combined}(1, k_n)$
  - $\text{cost}(M, c_i)$  the degree to which  $M$  violates constraint  $c_i$
- Cost of abstract states:
  - estimating this is quite involved, using approximation over the unknown wildcards (see notes)

# Reminder: Matching System Architecture



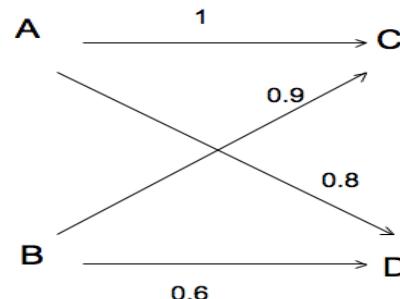
# Match Selector

---

- Selects matches from the *sim matrix*
- Simplest strategy: *thresholding*
  - all attribute pairs with *sim* not less than a threshold are returned as matches
  - e.g., given the matrix    name = <title: 0.5>  
                      releaseInfo = <releaseDate: 0.6>  
                      classification = <rating: 0.3>  
                      price = <basePrice: 0.5>  
                      given threshold 0.5, return matches name = title, etc.
- More complex strategies return the top few match combinations

# A Common Strategy to Select a Match Combination: Use Stable Marriage

- Elements of  $\mathbf{S}$  = men, elements of  $\mathbf{T}$  = women
- $sim(i,j)$  = the degree to which  $A_i$  and  $B_j$  desire each other
- Find a stable match combination between men and women
- A match combination would be unstable if
  - there are two couples  $A_i = B_j$  and  $A_k = B_l$  such that  $A_i$  and  $B_l$  want to be with each other, i.e.,  $sim(i,l) > sim(i, j)$  and  $sim(i,l) > sim(k,l)$
- Other algorithms exist to select a match combination



# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - **Reusing previous matches**
  - Many-to-many matches
- Schema mapping

# Reusing Previous Matches

---

- Schema matching tasks are often repetitive
  - e.g., keep matching new sources into the mediated schema
- Can a schema matching system improve over time? Can it learn from previous experience?
- Yes, one way to do this is to use machine learning techniques
  - consider matching sources  $\mathbf{S}_1, \dots, \mathbf{S}_n$  into a mediated schema  $\mathbf{G}$
  - we manually match  $\mathbf{S}_1, \dots, \mathbf{S}_m$  into  $\mathbf{G}$  (where  $m \ll n$ )
  - the system generalizes from these matches to predict matches for  $\mathbf{S}_{m+1}, \dots, \mathbf{S}_n$ 
    - ❖ use a technique called multi-strategy learning

# Multi-Strategy Learning: Training Phase

---

- Employ a set of learners  $L_1, \dots, L_k$ 
  - each learner creates a classifier for an element  $e$  of the mediated schema  $G$ , from training examples of  $e$
  - these training examples are derived using semantic matches between the training sources  $S_1, \dots, S_m$  and  $G$
- Use a meta-learner to learn a weight  $w_{e,L_i}$  for each element  $e$  of the mediated schema and each learner  $L_i$ 
  - these weights will be used later in the matching phase to combine the predictions of the learners  $L_i$
- See notes on examples of learners and how to train meta-learner

# Example of Training Phase

- Mediated schema  $\mathbf{G}$  has three attributes:  $e_1, e_2, e_3$
- Use two learners: Naive Bayes and Decision Tree
- NB learner creates three classifiers:  $C_{e1,NB}, C_{e2,NB}, C_{e3,NB}$ 
  - e.g.,  $C_{e1,NB}$  will decide if a given data instance belongs to  $e_1$
- To train  $C_{e1,NB}$ , use training sources  $S_1, \dots, S_m$ 
  - suppose when matching these to  $\mathbf{G}$ , we found that only two attributes a and b matches  $e_1$ 
    - ❖ use data instances of a and b as positive examples
    - ❖ use data instances of other attributes of  $S_1, \dots, S_m$  as negative examples
- Training other classifiers proceeds similarly
- Training meta-learner produces 6 weights:
  - $W_{e1,NB}, W_{e1,DT}, \dots, W_{e3,NB}, W_{e3,DT}$

# Multi-Strategy Learning: Matching Phase

- Given a new schema  $S$  with attributes  $e'_1, \dots, e'_n$
- Apply learners  $L_1, \dots, L_k$  to  $e'_1, \dots, e'_n$
- Combine the predictions of the learners
  - $p_e(e') = \sum_{i=1}^k w_{\{e, L_i\}} * p_{\{e, L_i\}}(e')$
  - $p_e(e')$ : the overall sim score between attribute  $e$  of mediated schema and attribute  $e'$  (which is  $e'_1, e'_2, \dots$ , or  $e'_n$ )
  - $p_{\{e, L\}}$ : the sim score learner  $L_i$  computes between  $e$  and  $e'$

# Example of Matching Phase

- Recall from the example of training phase
  - G has three attributes  $e_1, e_2, e_3$ ; two learners NB and DT
    - ❖ with classifiers  $C_{e1,NB}, C_{e2,NB}, C_{e3,NB}$  and  $C_{e1,DT}, C_{e2,DT}, C_{e3,DT}$
  - meta-learner has six weights  $w_{e1,NB}, w_{e1,DT}, \dots, w_{e3,NB}, w_{e3,DT}$
- Let S be a new source with attributes  $e'_1$  and  $e'_2$
- NB learner produces a 3\*2 matrix of sim scores
  - ❖  $p_{e1, NB}(e'_1), p_{e1, NB}(e'_2)$  [by classifier  $C_{e1, NB}$ ]
  - ❖  $p_{e2, NB}(e'_1), p_{e2, NB}(e'_2)$  [by classifier  $C_{e2, NB}$ ]
  - ❖  $p_{e3, NB}(e'_1), p_{e3, NB}(e'_2)$  [by classifier  $C_{e3, NB}$ ]
- DT learner produces a similar sim matrix
- Meta-learner combines the predictions
  - ❖  $p_{e1}(e'_1) = w_{e1, NB} * p_{e1, NB}(e'_1) + w_{e1, DT} * p_{e1, DT}(e'_1)$

# Discussion

---

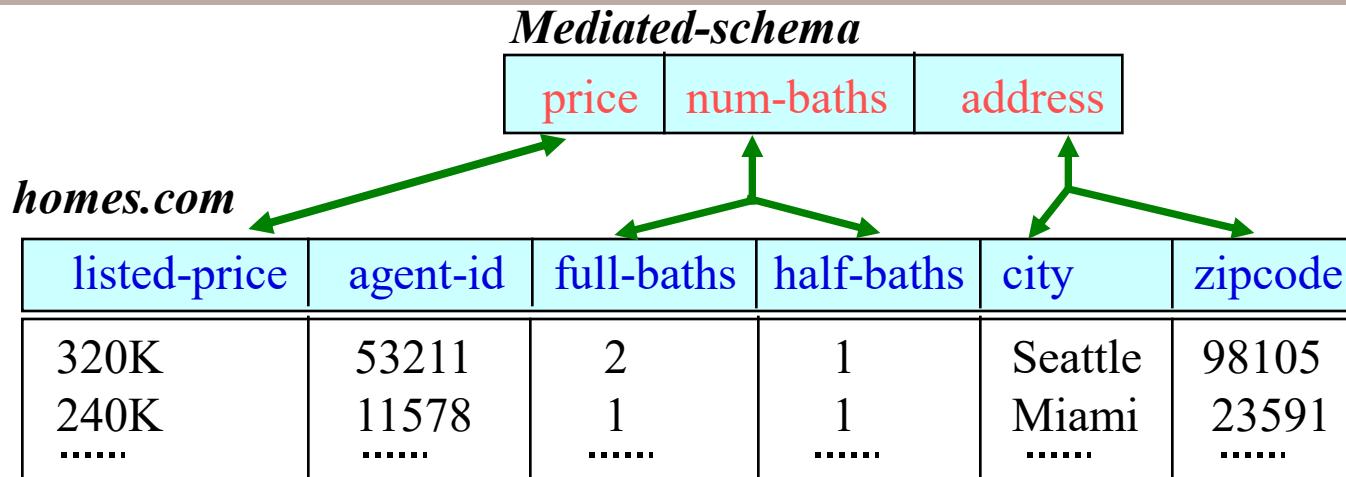
- Mapping to the generic schema matching architecture
  - learners = matchers
  - meta-learner = combiner
- Here the matchers and combiner use machine learning techniques → enable them to learn from previous matching experiences (of sources  $S_1, \dots, S_m$ )
- Note that even when we match just two sources S and T, we can still use machine learning techniques in the matchers and combiners
  - e.g., if the data instances of source S are available, they can be used as training data to build classifiers over S

# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

# Many-to-Many Matching



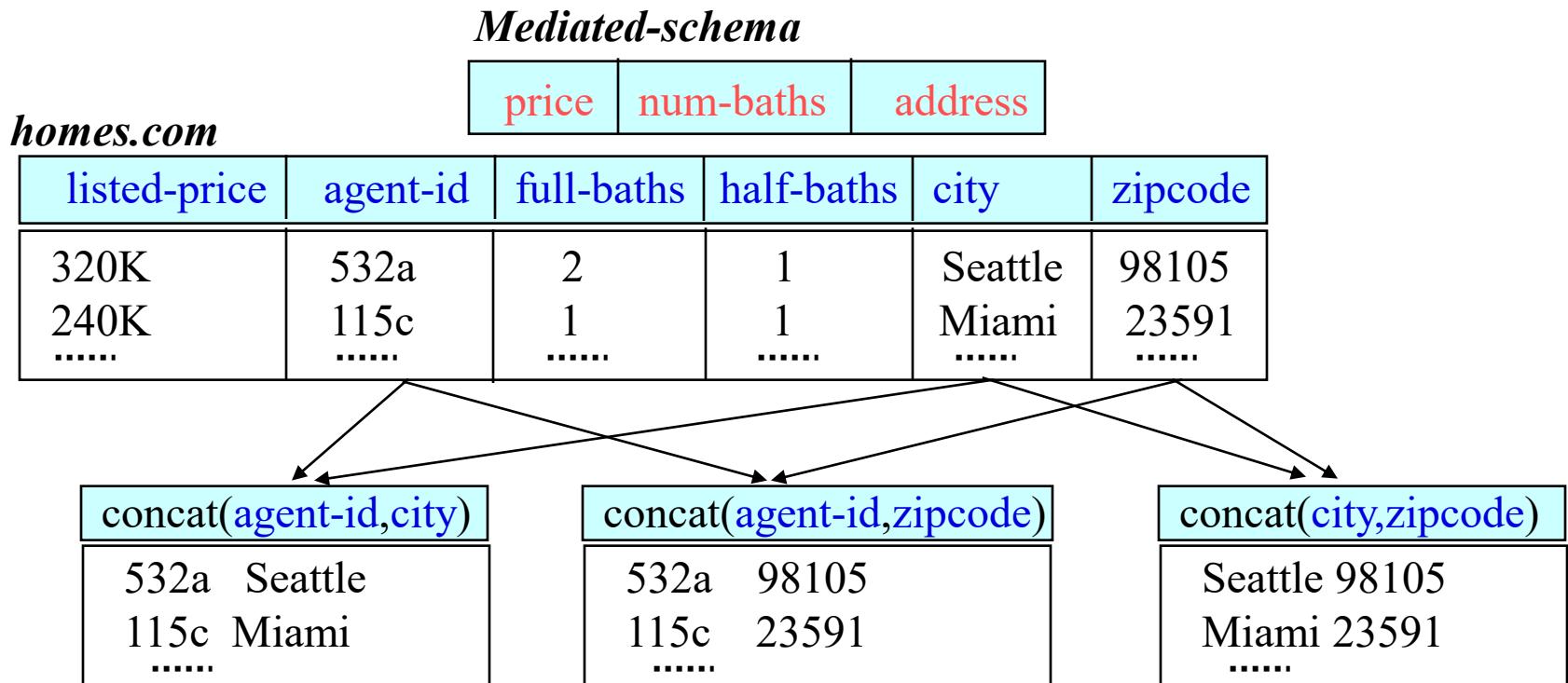
- Consider matches between *combinations* of columns
  - ... unlimited search space!
- Key challenge: control the search.

# Search for Complex Matches

---

- Employ specialized searchers:
  - Text searcher: concatenations of columns
  - Numeric searcher: arithmetic expressions
  - Date searcher: combine month/year/date
- Evaluate match candidates:
  - Compare with learned models
  - Statistics on data instances
  - Typical heuristics

# An Example:Text Searcher



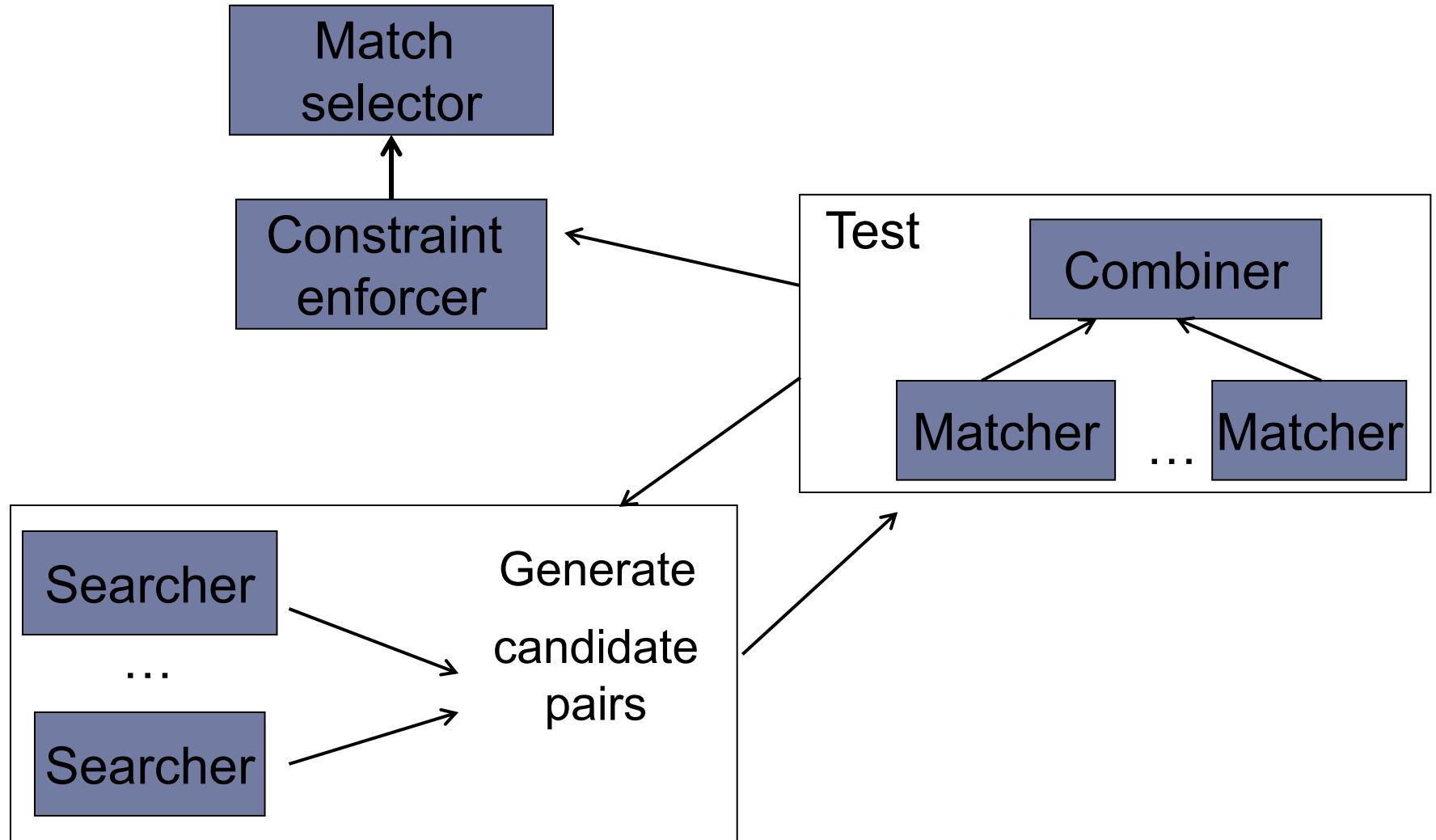
- Best match candidates for **address**
  - $(\text{agent-id}, 0.7)$ ,  $(\text{concat}(\text{agent-id}, \text{city}), 0.75)$ ,  $(\text{concat}(\text{city}, \text{zipcode}), 0.9)$

# Controlling the Search

---

- Limit the search with beam search:
  - Consider only top  $k$  candidates at every level of the search
- Termination based on diminishing returns:
  - Estimate of quality does not change much between iterations
- Details of a system that did this:
  - iMap [Doan et al., SIGMOD, 2004]

# Modified Architecture



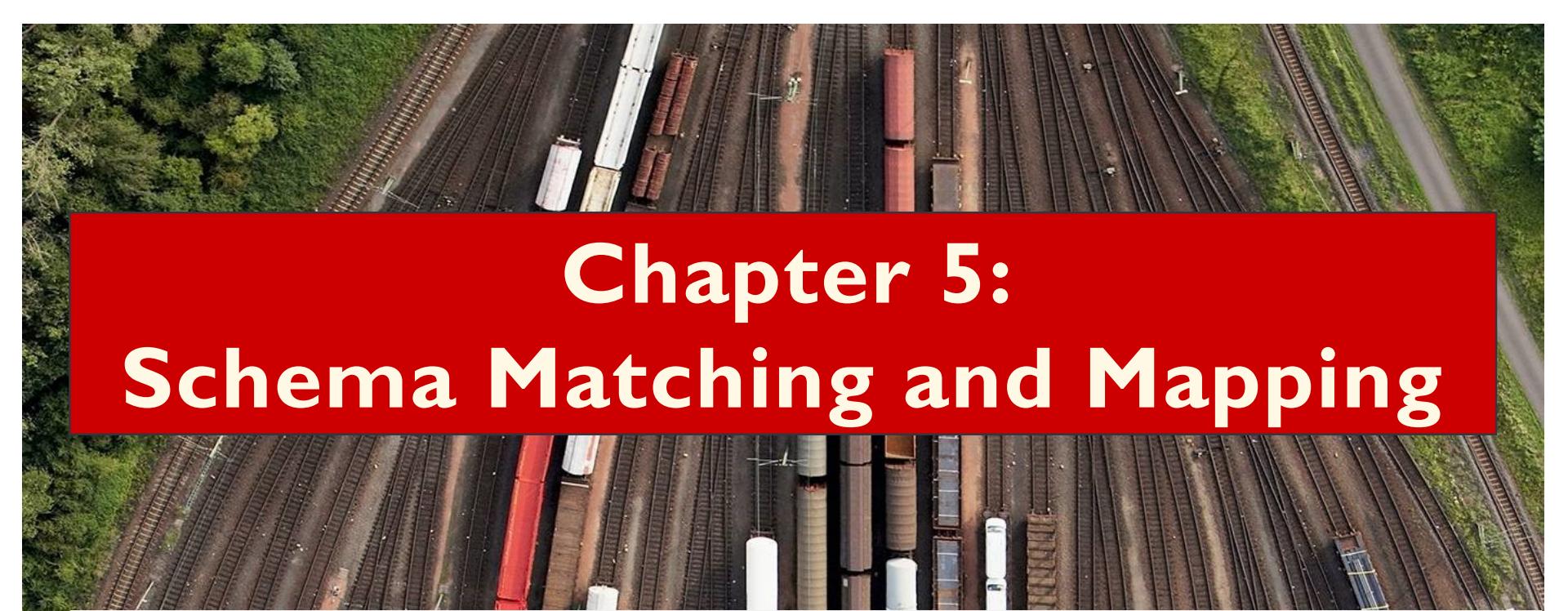
# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

# Summary

- Schema matching:
  - Use multiple matchers and combine results
  - Learn from the past
  - Incorporate constraints and user feedback
- From matching to mapping:
  - Search through possible queries
  - Principles from database design guide search
  - User interaction is key



# **Chapter 5:** **Schema Matching and Mapping**



**PRINCIPLES OF  
DATA INTEGRATION**

**ANHAI DOAN ALON HALEVY ZACHARY IVES**

# Introduction

- We have described
  - formalisms to specify source descriptions
  - algorithms that use these descriptions to reformulate queries
- How to create the source descriptions?
  - often begin by creating semantic matches
    - ❖ name = title, location = concat(city, state, zipcode)
  - then elaborate matches into semantic mappings
    - ❖ e.g., structured queries in a language such as SQL
- Schema matching and mapping are often quite difficult
- This chapter describes matching and mapping tools
  - that can significantly reduce the time it takes for the developer to create matches and mappings

# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

# Using Classifiers

---

- Builds classifiers on one schema and uses them to classify the elements of the other schema
  - e.g., use Naïve Bayes, decision tree, rule learning, SVM
- A common strategy
  - for each element  $s_i$  of schema  $S$ , want to train classifier  $C_i$  to recognize instances of  $s_i$
  - to do this, need positive and negative training examples
    - ❖ take all data instances of  $s_i$  (that are available) to be positive examples
    - ❖ take all data instances of other elements of  $S$  to be negative examples
  - train  $C_i$  on the positive and negative examples

# Using Classifiers

---

- A common strategy (cont.)
  - now we can use  $C_i$  to compute sim score between  $s_i$  and each element  $t_j$  of schema T
  - to do this, apply  $C_i$  to data instances of  $t_j$ 
    - ❖ for each instance,  $C_i$  produces a number in [0,1] that is the confidence that the instance is indeed an instance of  $s_i$
  - now need to aggregate the confidence scores of the instances (of  $t_j$ ) to return a single confidence score (as the sim score between  $s_i$  and  $t_j$ )
  - a simple way to do so is to compute the average score over all instances of  $t_j$

# Using Classifiers: An Example

## SCHEMA S

| <b>currentShowing</b> | <b>address</b>   | <b>phone</b>   |
|-----------------------|------------------|----------------|
| Lord of the Rings     | Madison WI       | (608) 695 2311 |
|                       | Mountain View CA | (650) 277 1358 |

## SCHEMA T

| <b>name</b> | <b>location</b> | <b>phone</b> |
|-------------|-----------------|--------------|
| ...         | Milwaukee WI    | ...          |
|             | Palo Alto CA    |              |
|             | Philadelphia PA |              |

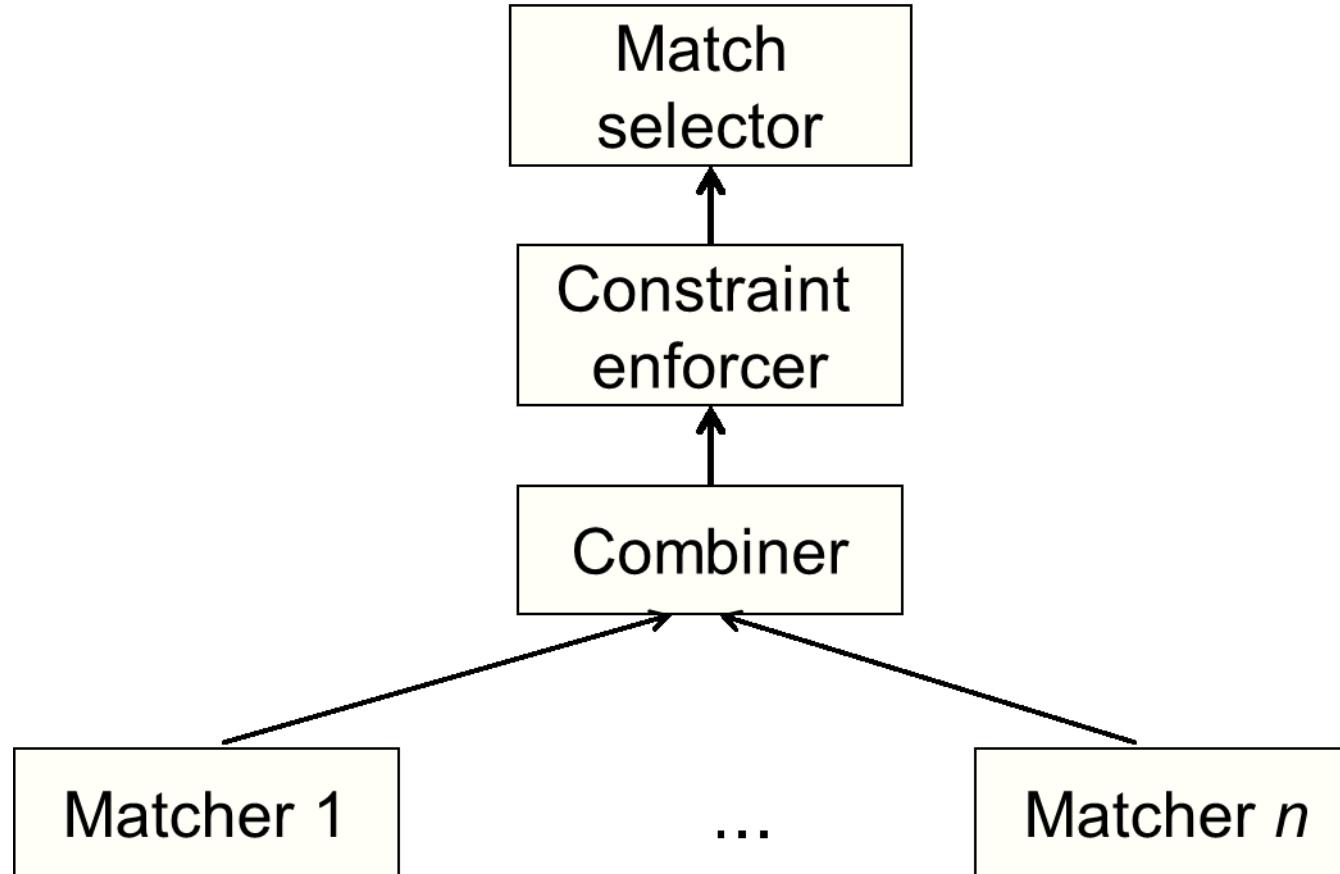
- $s_i$  is address,  $t_j$  is location
- Sim scores are 0.9, 0.7, and 0.5, respectively for the three instances of T.location → return average score of 0.7 as sim score between address and location

# Using Classifiers

---

- Designer decides which schema should play the role of schema S (on which to build classifiers)
  - typically chooses the mediated schema to be S, so that can reuse the classifiers to match the schemas of new data sources
- May want to do it both ways
  - build classifiers on S and use them to classify instances of T
  - then build classifiers on T and use them to classify instances of S
  - e.g., when both S and T are taxonomies of concepts
    - ❖ see the bibliographic notes

# Reminder: Matching System Architecture



# Enforcing Domain Integrity Constraints

---

- Designer often has knowledge that can be naturally expressed as domain integrity constraints
- Constraint enforcer exploits these to prune certain match combinations
  - searches through the space of all match combinations produced by the combiner
  - finds one combination with the highest aggregated confidence score that satisfies the constraints

# Applying Constraints with Local Propagation

- Propagate constraints locally from schema elements to their neighbors until we reach a fixed point
- First select constraints that involve element's neighbors
- Then rephrase them to work with local propagation

|                | Constraints                                                                                                                                                                                                                                                                                                                             | Costs    |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| c <sub>1</sub> | If A = Items.code, then A is a key                                                                                                                                                                                                                                                                                                      | $\infty$ |
| c <sub>2</sub> | If A = Items.desc, then any random sample of 100 data instances of A must have an average length of at least 20 words                                                                                                                                                                                                                   | 1.5      |
| c <sub>3</sub> | If A <sub>1</sub> = B <sub>1</sub> , A <sub>2</sub> = B <sub>2</sub> , B <sub>2</sub> is next to B <sub>1</sub> in the schema, but A <sub>2</sub> is not next to A <sub>1</sub> , then there is no A* next to A <sub>1</sub> such that  sim(A*,B <sub>2</sub> ) – sim(A <sub>2</sub> ,B <sub>2</sub> )  ≤ t for a small pre-specified t | 2        |
| c <sub>4</sub> | If more than half of the attributes of Table U match those of Table V, then U = V                                                                                                                                                                                                                                                       | 1        |

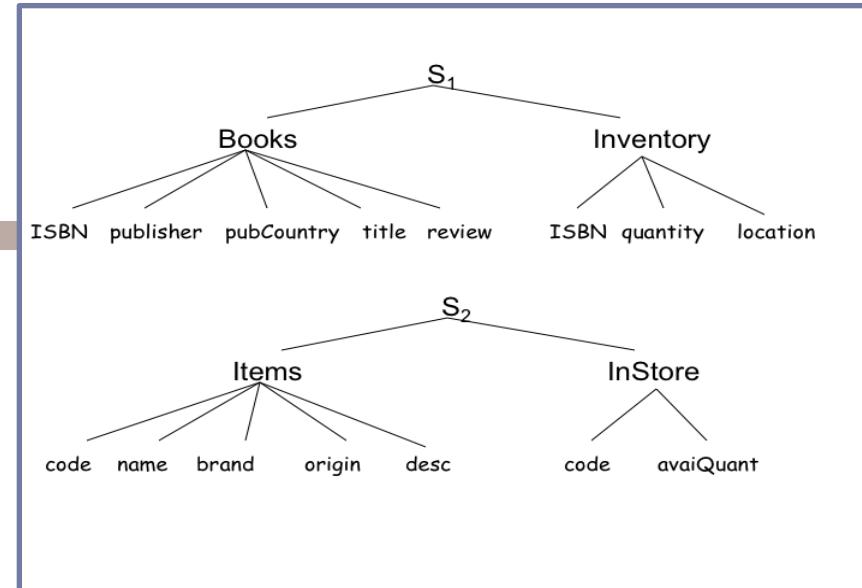
# An Example

|       | Constraints                                                                                                                                                                                                                         | Costs    |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| $c_1$ | If $A = \text{Items.code}$ , then $A$ is a key                                                                                                                                                                                      | $\infty$ |
| $c_2$ | If $A = \text{Items.desc}$ , then any random sample of 100 data instances of $A$ must have an average length of at least 20 words                                                                                                   | 1.5      |
| $c_3$ | If $A_1 = B_1, A_2 = B_2, B_2$ is next to $B_1$ in the schema, but $A_2$ is not next to $A_1$ , then there is no $A^*$ next to $A_1$ such that $ \text{sim}(A^*, B_2) - \text{sim}(A_2, B_2)  \leq t$ for a small pre-specified $t$ | 2        |
| $c_4$ | If more than half of the attributes of Table U match those of Table V, then $U = V$                                                                                                                                                 | 1        |

- rephrasing  $c_3$ :
  - ❖ if  $\text{sim}(A_1, B_1) \leq 0.9$  and  $A_1$  has a neighbor  $A_2$  such that  $\text{sim}(A_2, B_2) \geq 0.75$ , and  $B_1$  is a neighbor of  $B_2$ , then increase  $\text{sim}(A_1, B_1)$  by  $\alpha$
- constraint  $c_4$  can also be rephrased (see notes)

# Local Propagation Algorithm

- Initialization:
  - represent  $S_1$  and  $S_2$  as graphs
  - algorithm computes a *sim matrix SIM* which is initialized to be the combined matrix (output by the combiner)
- Iteration:
  - select a node  $s_1$  in graph of  $S_1$ , update the values in *SIM* based on similarities computed for its neighbors
  - if perform tree traversal, go bottom-up, starting from the leaves
- Termination:
  - after either a fixed number of iterations or when the changes to *SIM* are smaller than a pre-defined threshold



# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - **Reusing previous matches**
  - Many-to-many matches
- Schema mapping

# Reusing Previous Matches

---

- Schema matching tasks are often repetitive
  - e.g., keep matching new sources into the mediated schema
- Can a schema matching system improve over time? Can it learn from previous experience?
- Yes, one way to do this is to use machine learning techniques
  - consider matching sources  $\mathbf{S}_1, \dots, \mathbf{S}_n$  into a mediated schema  $\mathbf{G}$
  - we manually match  $\mathbf{S}_1, \dots, \mathbf{S}_m$  into  $\mathbf{G}$  (where  $m \ll n$ )
  - the system generalizes from these matches to predict matches for  $\mathbf{S}_{m+1}, \dots, \mathbf{S}_n$ 
    - ❖ use a technique called multi-strategy learning

# Multi-Strategy Learning: Training Phase

---

- Employ a set of learners  $L_1, \dots, L_k$ 
  - each learner creates a classifier for an element  $e$  of the mediated schema  $G$ , from training examples of  $e$
  - these training examples are derived using semantic matches between the training sources  $S_1, \dots, S_m$  and  $G$
- Use a meta-learner to learn a weight  $w_{e,L_i}$  for each element  $e$  of the mediated schema and each learner  $L_i$ 
  - these weights will be used later in the matching phase to combine the predictions of the learners  $L_i$
- See notes on examples of learners and how to train meta-learner

# Example of Training Phase

- Mediated schema  $\mathbf{G}$  has three attributes:  $e_1, e_2, e_3$
- Use two learners: Naive Bayes and Decision Tree
- NB learner creates three classifiers:  $C_{e1,NB}, C_{e2,NB}, C_{e3,NB}$ 
  - e.g.,  $C_{e1,NB}$  will decide if a given data instance belongs to  $e_1$
- To train  $C_{e1,NB}$ , use training sources  $S_1, \dots, S_m$ 
  - suppose when matching these to  $\mathbf{G}$ , we found that only two attributes a and b matches  $e_1$ 
    - ❖ use data instances of a and b as positive examples
    - ❖ use data instances of other attributes of  $S_1, \dots, S_m$  as negative examples
- Training other classifiers proceeds similarly
- Training meta-learner produces 6 weights:
  - $W_{e1,NB}, W_{e1,DT}, \dots, W_{e3,NB}, W_{e3,DT}$

# Multi-Strategy Learning: Matching Phase

- Given a new schema  $S$  with attributes  $e'_1, \dots, e'_n$
- Apply learners  $L_1, \dots, L_k$  to  $e'_1, \dots, e'_n$
- Combine the predictions of the learners
  - $p_e(e') = \sum_{i=1}^k w_{\{e, L_i\}} * p_{\{e, L_i\}}(e')$
  - $p_e(e')$ : the overall sim score between attribute  $e$  of mediated schema and attribute  $e'$  (which is  $e'_1, e'_2, \dots$ , or  $e'_n$ )
  - $p_{\{e, L\}}$ : the sim score learner  $L_i$  computes between  $e$  and  $e'$

# Example of Matching Phase

- Recall from the example of training phase
  - G has three attributes  $e_1, e_2, e_3$ ; two learners NB and DT
    - ❖ with classifiers  $C_{e1,NB}, C_{e2,NB}, C_{e3,NB}$  and  $C_{e1,DT}, C_{e2,DT}, C_{e3,DT}$
  - meta-learner has six weights  $w_{e1,NB}, w_{e1,DT}, \dots, w_{e3,NB}, w_{e3,DT}$
- Let S be a new source with attributes  $e'_1$  and  $e'_2$
- NB learner produces a 3\*2 matrix of sim scores
  - ❖  $p_{e1, NB}(e'_1), p_{e1, NB}(e'_2)$  [by classifier  $C_{e1, NB}$ ]
  - ❖  $p_{e2, NB}(e'_1), p_{e2, NB}(e'_2)$  [by classifier  $C_{e2, NB}$ ]
  - ❖  $p_{e3, NB}(e'_1), p_{e3, NB}(e'_2)$  [by classifier  $C_{e3, NB}$ ]
- DT learner produces a similar sim matrix
- Meta-learner combines the predictions
  - ❖  $p_{e1}(e'_1) = w_{e1, NB} * p_{e1, NB}(e'_1) + w_{e1, DT} * p_{e1, DT}(e'_1)$

# Discussion

---

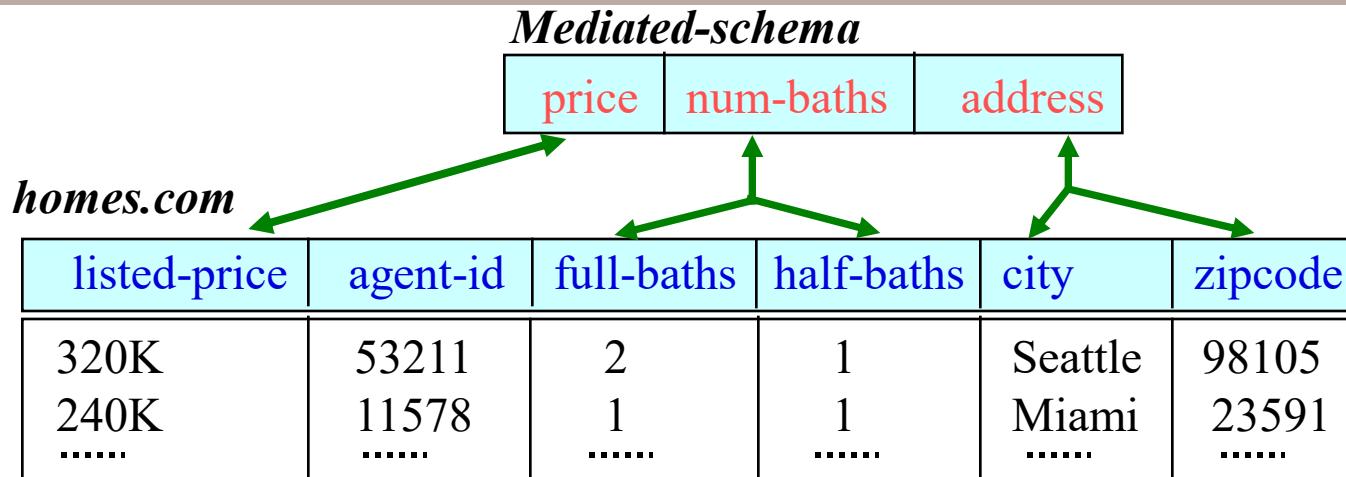
- Mapping to the generic schema matching architecture
  - learners = matchers
  - meta-learner = combiner
- Here the matchers and combiner use machine learning techniques → enable them to learn from previous matching experiences (of sources  $S_1, \dots, S_m$ )
- Note that even when we match just two sources S and T, we can still use machine learning techniques in the matchers and combiners
  - e.g., if the data instances of source S are available, they can be used as training data to build classifiers over S

# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

# Many-to-Many Matching



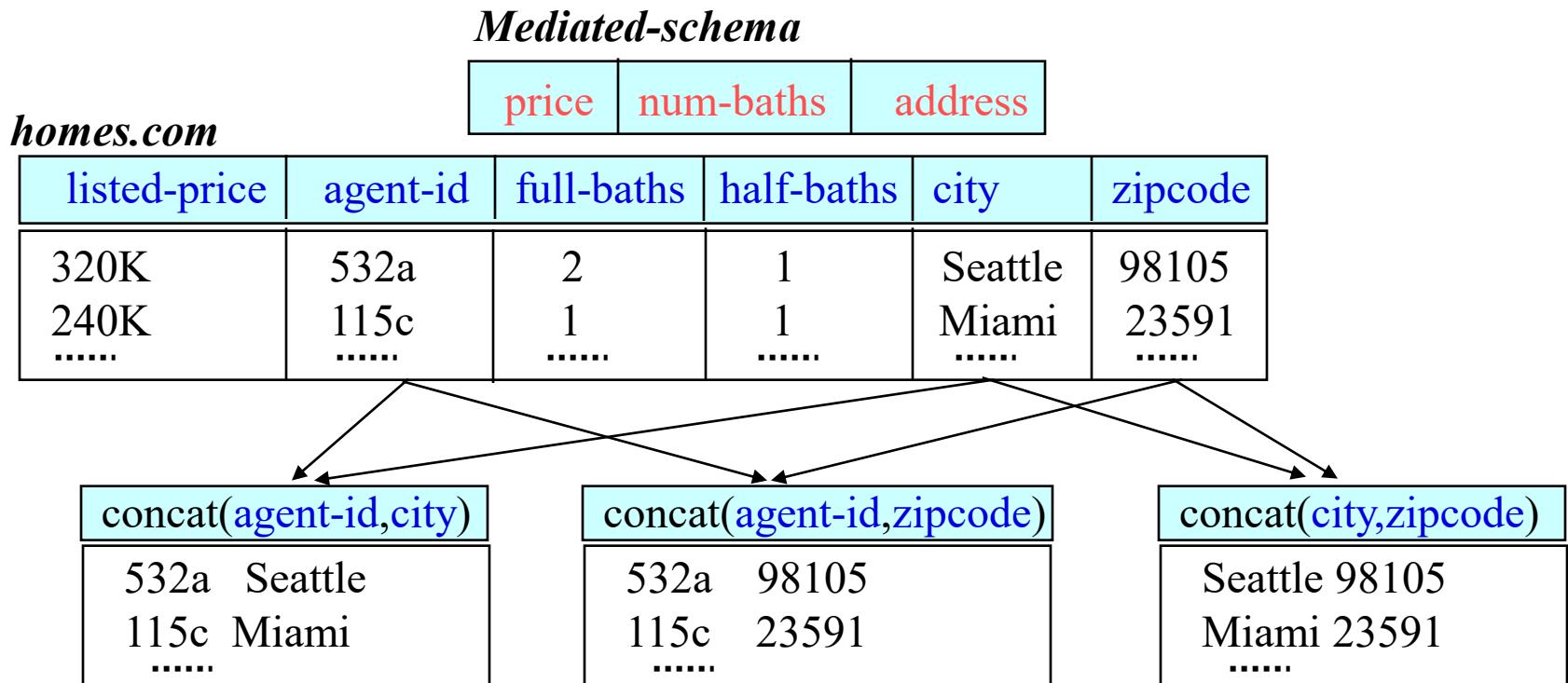
- Consider matches between *combinations* of columns
  - ... unlimited search space!
- Key challenge: control the search.

# Search for Complex Matches

---

- Employ specialized searchers:
  - Text searcher: concatenations of columns
  - Numeric searcher: arithmetic expressions
  - Date searcher: combine month/year/date
- Evaluate match candidates:
  - Compare with learned models
  - Statistics on data instances
  - Typical heuristics

# An Example:Text Searcher



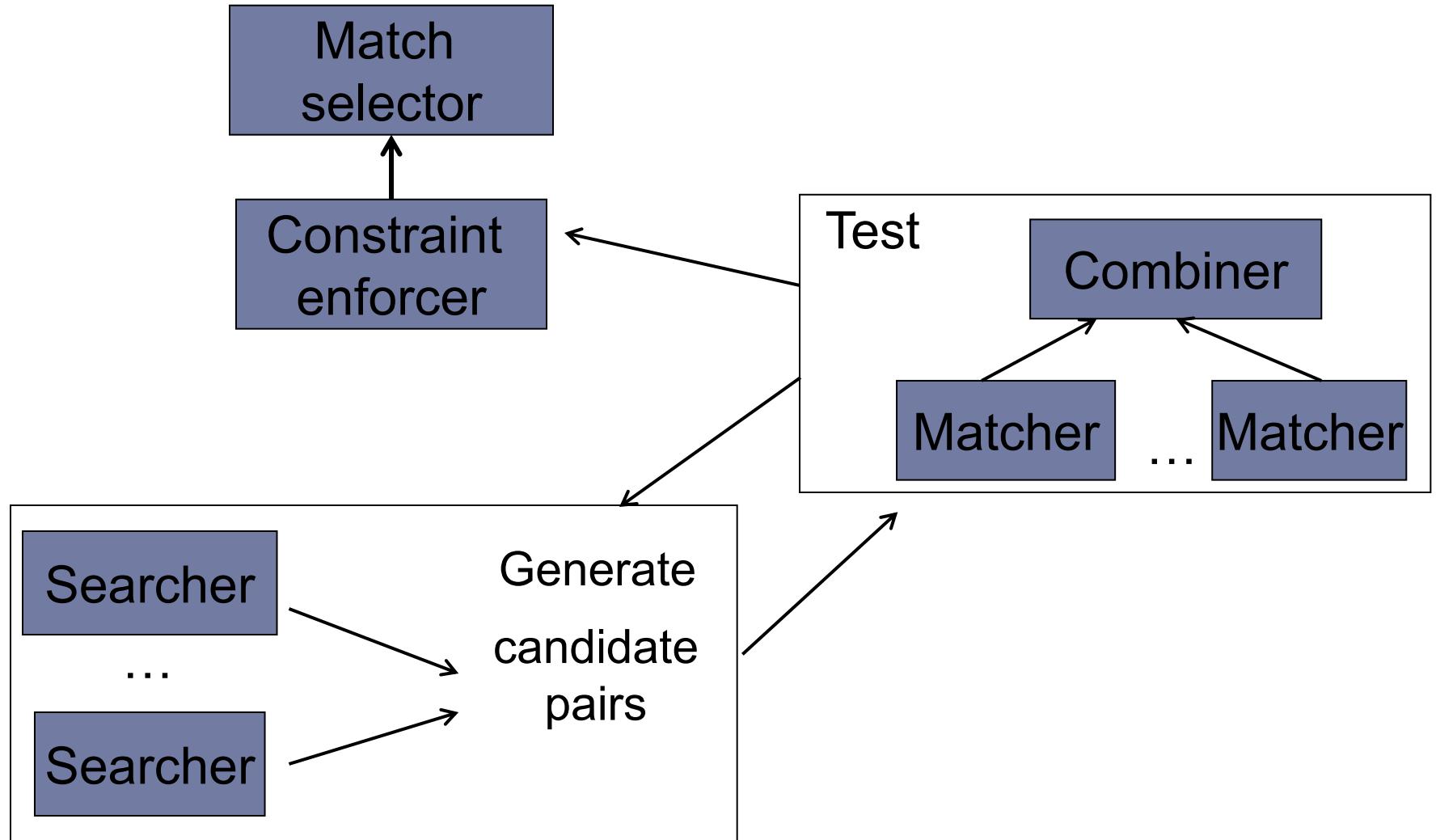
- Best match candidates for **address**
  - $(\text{agent-id}, 0.7)$ ,  $(\text{concat}(\text{agent-id}, \text{city}), 0.75)$ ,  $(\text{concat}(\text{city}, \text{zipcode}), 0.9)$

# Controlling the Search

---

- Limit the search with beam search:
  - Consider only top  $k$  candidates at every level of the search
- Termination based on diminishing returns:
  - Estimate of quality does not change much between iterations
- Details of a system that did this:
  - iMap [Doan et al., SIGMOD, 2004]

# Modified Architecture



# Outline

---

- Problem definition, challenges, and overview
- Schema matching
  - Matchers
  - Combining match predictions
  - Enforcing domain integrity constraints
  - Match selector
  - Reusing previous matches
  - Many-to-many matches
- Schema mapping

# From Matching to Mapping

- Input:
  - Schema matches
  - Constraints (if available)
- Output:
  - Schema mappings
  - (for now, let's do SQL)
- Let's look at the choices we need to make
  - A solution will emerge...
  - Based on the IBM Clio Project

# Multiple Join Paths

Address

|    |      |
|----|------|
| id | Addr |
|----|------|

Professor

|    |      |        |
|----|------|--------|
| id | name | salary |
|----|------|--------|

Student

|     |     |    |
|-----|-----|----|
| nam | GPA | Yr |
|-----|-----|----|

PayRate

|      |        |
|------|--------|
| Rank | HrRate |
|------|--------|

WorksOn

|      |      |     |          |
|------|------|-----|----------|
| name | Proj | hrs | ProjRank |
|------|------|-----|----------|

Personnel

|     |
|-----|
| Sal |
|-----|

$$f1: \text{PayRate}(\text{HrRate}) * \text{WorksOn}(\text{Hrs}) = \text{Personnel}(\text{Sal})$$

# Two Possible Queries

```
select P.HrRate * W.hrs
from PayRate P, WorksOn W
where P.Rank = W.ProjRank
```

```
select P.HrRate * W.hrs
from PayRate P, WorksOn W, Student S
where W.Name=S.Name and
S.Yr = P.Rank
```

We could also consider the Cartesian product  
but that seems intuitively wrong.

# Horizontal partitioning

Address

|    |      |
|----|------|
| id | Addr |
|----|------|

Professor

|    |      |        |
|----|------|--------|
| id | name | salary |
|----|------|--------|

Personnel

Student

|     |     |    |
|-----|-----|----|
| nam | GPA | Yr |
|-----|-----|----|

Sal

PayRate

|      |        |
|------|--------|
| Rank | HrRate |
|------|--------|

WorksOn

|      |      |     |          |
|------|------|-----|----------|
| name | Proj | hrs | ProjRank |
|------|------|-----|----------|

$f_2: Professor(Sal) \rightarrow Personnel(Sal)$

# What Kind of Union?

```
select P.HrRate * W.hrs
from PayRate P, WorksOn W
where P.Rank = W.ProjRank
```

UNION ALL

```
select Sal
from Professor
```

Could also do an outer-union ...  
and even a join.

# Two Sets of Decisions

- What join paths to choose?
  - (we'll call these 'candidate sets')
- How to combine the results of the joins?
- Underlying database-design principles:
  - Values in the source should appear in the target
  - They should only appear once
  - We should not lose information

# Join Paths

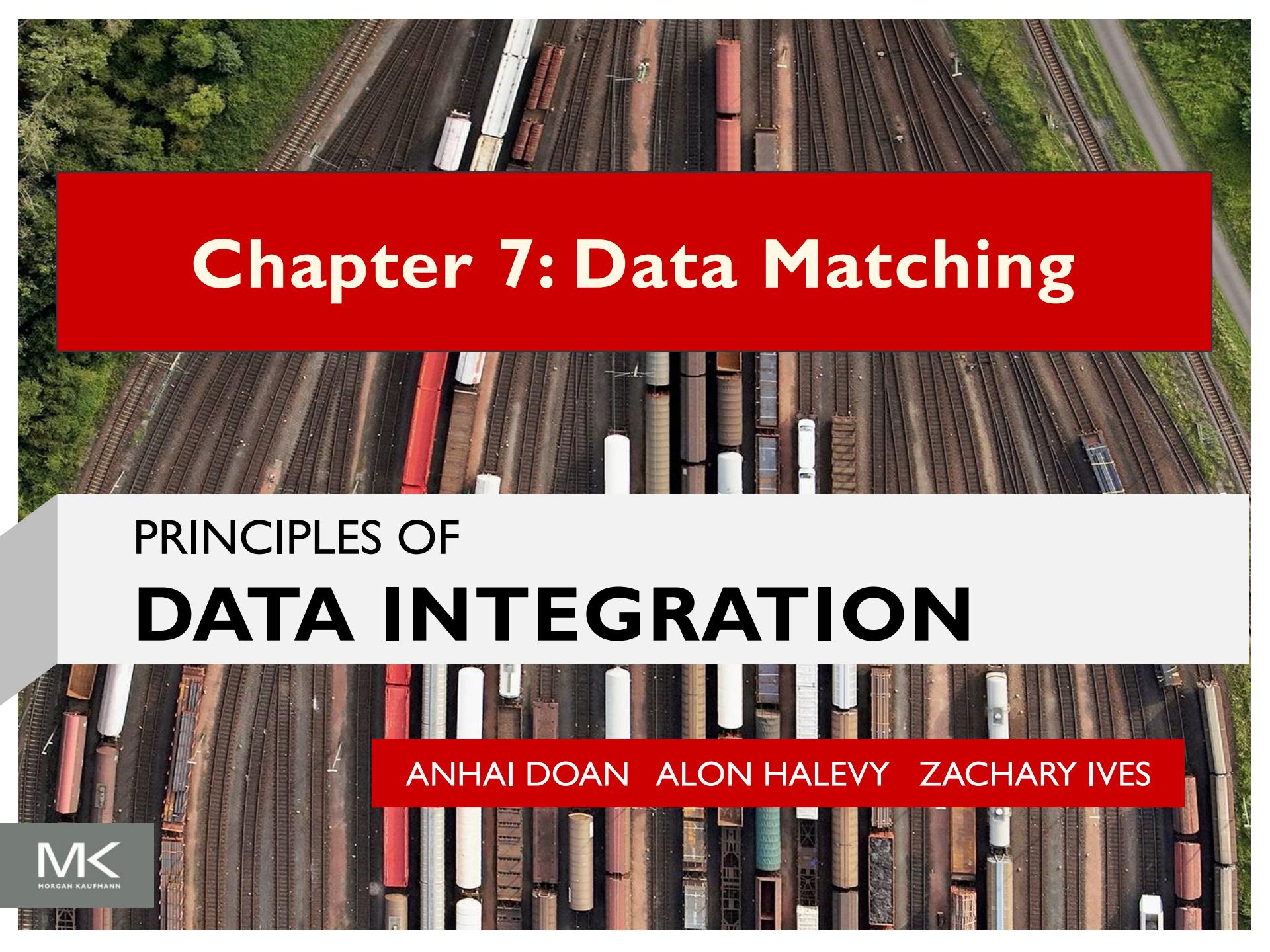
- Discover candidate join paths by:
  - following foreign keys
  - look at paths used in queries
  - paths discovered by mining data for joinable columns.
- Select paths by:
  - Prefer foreign keys
  - Prefer ones that involve a constraint
  - Prefer smaller difference between inner and outer joins.
- Result of this step: candidate sets.

# Selecting Covers

- Candidate cover: a minimal set of candidate sets that covers all the input correspondences
- Select best cover:
  - Prefer with fewest candidate paths
  - Prefer one that covers more attributes of target
- Express mapping as union of candidate sets in selected cover.

# Summary

- Schema matching:
  - Use multiple matchers and combine results
  - Learn from the past
  - Incorporate constraints and user feedback
- From matching to mapping:
  - Search through possible queries
  - Principles from database design guide search
  - User interaction is key



# Chapter 7: Data Matching

## PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Introduction

- **Data matching**: find structured data items that refer to the same real-world entity
  - **entities** may be represented by tuples, XML elements, or RDF triples, not by strings as in string matching
  - e.g., (David Smith, 608-245-4367, Madison WI)  
vs (D. M. Smith, 245-4367, Madison WI)
- Data matching arises in many integration scenarios
  - merging multiple databases with the same schema
  - joining rows from sources with different schemas
  - matching a user query to a data item
- One of the most fundamental problems in data integration

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

# Problem definition

---

- Given two relational tables  $X$  and  $Y$  with identical schemas
  - assume each tuple in  $X$  and  $Y$  describes an entity (e.g., person)
- We say tuple  $x \in X$  matches tuple  $y \in Y$  if they refer to the same real-world entity
  - $(x,y)$  is called a match
- Goal: find all matches between  $X$  and  $Y$

# Example

Table X

|                | Name       | Phone          | City      | State |
|----------------|------------|----------------|-----------|-------|
| X <sub>1</sub> | Dave Smith | (608) 395 9462 | Madison   | WI    |
| X <sub>2</sub> | Joe Wilson | (408) 123 4265 | San Jose  | CA    |
| X <sub>3</sub> | Dan Smith  | (608) 256 1212 | Middleton | WI    |

(a)

Table Y

|                | Name            | Phone    | City    | State |
|----------------|-----------------|----------|---------|-------|
| Y <sub>1</sub> | David D. Smith  | 395 9426 | Madison | WI    |
| Y <sub>2</sub> | Daniel W. Smith | 256 1212 | Madison | WI    |

(b)

Matches

(x<sub>1</sub>, y<sub>1</sub>)  
(x<sub>3</sub>, y<sub>2</sub>)

(c)

- Other variations
  - Tables X and Y have different schemas
  - Match tuples within a single table X
  - The data is not relational, but XML or RDF
- These are not considered in this chapter (see bib notes)

# Why is this different than string matching?

---

- In theory, can treat each tuple as a string by concatenating the fields, then apply string matching techniques
- But doing so makes it hard to apply sophisticated techniques and domain-specific knowledge
- E.g., consider matching tuples that describe persons
  - suppose we know that in this domain two tuples match if the names and phone match exactly
  - this knowledge is hard to encode if we use string matching
  - so it is better to keep the fields apart

# Challenges

---

- Same as in string matching
- How to match accurately?
  - difficult due to variations in formatting conventions, use of abbreviations, shortening, different naming conventions, omissions, nicknames, and errors in data
  - several common approaches: rule-based, learning-based, clustering, probabilistic, collective
- How to scale up to large data sets?
  - again many approaches have been developed, as we will discuss

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

# Rule-based matching

---

- The developer writes rules that specify when two tuples match
  - typically after examining many matching and non-matching tuple pairs, using a development set of tuple pairs
  - rules are then tested and refined, using the same development set or a test set
- Many types of rules exist, we will consider
  - linearly weighted combination of individual similarity scores
  - logistic regression combination
  - more complex rules

# Linearly weighted combination rules

- Compute the sim score between tuples  $x$  and  $y$  as a linearly weighted combination of individual sim scores
  - $\text{sim}(x,y) = \sum_{i=1}^n \alpha_i * \text{sim}_i(x, y)$
  - $n$  is number of attributes in each table
  - $\text{sim}_i(x,y)$  is a sim score between the  $i$ -th attributes of  $x$  and  $y$
  - $\alpha_i \in [0,1]$  is a pre-specified weight that indicates the importance of the  $i$ -th attribute to  $\text{sim}(x,y)$ , such that  $\sum_{i=1}^n \alpha_i = 1$
- We declare  $x$  and  $y$  matched if  $\text{sim}(x,y) \geq \beta$  for a pre-specified  $\beta$ , and not matched otherwise
  - in another variation: declare  $x$  and  $y$  matched if  $\text{sim}(x,y) \geq \beta$ , not matched if  $\text{sim}(x,y) < \gamma$ , and subject to human review

# Example

Table X

|                | Name       | Phone          | City      | State |
|----------------|------------|----------------|-----------|-------|
| X <sub>1</sub> | Dave Smith | (608) 395 9462 | Madison   | WI    |
| X <sub>2</sub> | Joe Wilson | (408) 123 4265 | San Jose  | CA    |
| X <sub>3</sub> | Dan Smith  | (608) 256 1212 | Middleton | WI    |

(a)

Table Y

|                | Name            | Phone    | City    | State |
|----------------|-----------------|----------|---------|-------|
| Y <sub>1</sub> | David D. Smith  | 395 9426 | Madison | WI    |
| Y <sub>2</sub> | Daniel W. Smith | 256 1212 | Madison | WI    |

(b)

Matches

(x<sub>1</sub>, y<sub>1</sub>)  
(x<sub>3</sub>, y<sub>2</sub>)

(c)

- $\text{sim}(x,y) =$

$$0.3s_{\text{name}}(x,y) + 0.3s_{\text{phone}}(x,y) + 0.1s_{\text{city}}(x,y) + 0.3s_{\text{state}}(x,y)$$

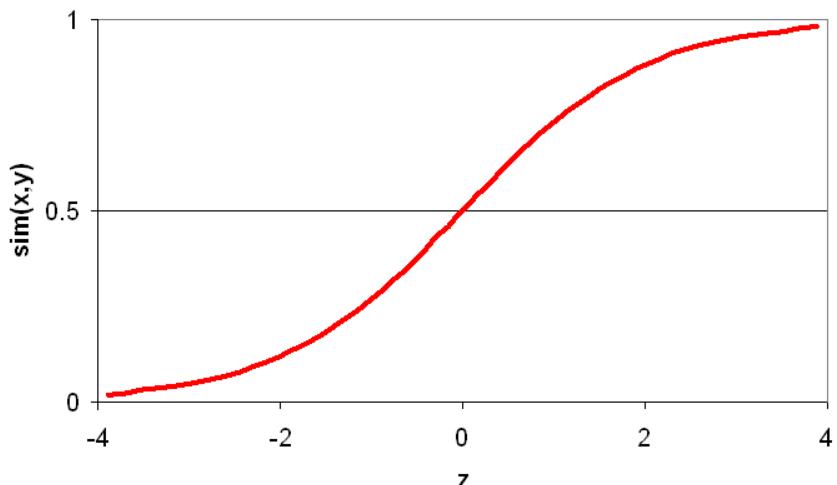
- $s_{\text{name}}(x,y)$ : based on Jaro-Winkler
- $s_{\text{phone}}(x,y)$ : based on edit distance between x's phone (after removing area code) and y's phone
- $s_{\text{city}}(x,y)$ : based on edit distance
- $s_{\text{state}}(x,y)$ : based on exact match; yes → 1, no → 0

# Pros and Cons

- Pros
  - conceptually simple, easy to implement
  - can learn weights  $\alpha_i$  from training data
- Cons
  - an increase  $\delta$  in the value of any  $s_i$  will cause a linear increase  $\alpha_i * \delta$  in the value of  $s$
  - in certain scenarios this is not desirable, thereafter a certain threshold an increase in  $s_i$  should count less (i.e., “diminishing returns” should kick in)
  - e.g., if  $s_{name}(x,y)$  is already 0.95 then the two names already very closely match
    - ❖ so any increase in  $s_{name}(x,y)$  should contribute only minimally

# Logistic regression rules

- address the diminishing-returns problem
- $\text{sim}(x,y) = 1 / (1 + e^{-z})$ , where  $z = \sum_{i=1}^n \alpha_i * \text{sim}_i(x, y)$ 
  - here  $\alpha_i$  are not constrained to be in  $[0,1]$  and sum to 1
  - so  $z$  goes from  $-\infty$  to  $+\infty$ , in which case  $\text{sim}(x,y)$  gradually increases, but minimally so after  $z$  has exceeded a certain value  
→ ensuring diminishing returns



# Logistic regression rules

---

- Are also very useful in situations where
  - there are many “signals” (e.g., 10-20) that can contribute to whether two tuples match
  - we don’t need all of these signals to “fire” in order to conclude that the tuples match
  - as long as a reasonable number of them fire, we have sufficient confidence
- Logistic regression is a natural fit for such cases
- Hence is quite popular as a first matching method to try

# More complex rules

---

- Appropriate when we want to **encode more complex matching knowledge**
  - e.g., two persons match if names match approximately and either phones match exactly or addresses match exactly
    1. If  $s_{name}(x,y) < 0.8$  then return “not matched”
    2. Otherwise if  $e_{phone}(x,y) = \text{true}$  then return “matched”
    3. Otherwise if  $e_{city}(x,y) = \text{true}$  and  $e_{state}(x,y) = \text{true}$  then return “matched”
    4. Otherwise return “not matched”

# Pros and Cons of Rule-based approaches

---

- Pros
  - easy to start, conceptually relatively easy to understand, implement, debug
  - typically run fast
  - can encode complex matching knowledge
- Cons
  - can be labor intensive, it takes a lot of time to write good rules
  - can be difficult to set appropriate weights
  - in certain cases it is not even clear how to write rules
  - **learning-based approaches** address these issues

# Outline

---

- Problem definition
- Rule-based matching
- **Learning- based matching**
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

# Learning-based matching

---

- Here we consider **supervised** learning
  - learn a matching model **M** from training data, then apply **M** to match new tuple pairs
  - will consider unsupervised learning later
- Learning a matching model **M** (the training phase)
  - start with training data:  $T = \{(x_1, y_1, l_1), \dots (x_n, y_n, l_n)\}$ , where each  $(x_i, y_i)$  is a tuple pair and  $l_i$  is a label: “yes” if  $x_i$  matches  $y_i$  and “no” otherwise
  - define a set of features  $f_1, \dots, f_m$ , each quantifying one aspect of the domain judged possibly relevant to matching the tuples

# Learning-based matching

- Learning a matching model  $\mathbf{M}$  (continued)
  - convert each training example  $(x_i, y_i, l_i)$  in  $T$  to a pair  $(\langle f_1(x_i, y_i), \dots, f_m(x_i, y_i) \rangle, c_i)$ 
    - ❖  $v_i = \langle f_1(x_i, y_i), \dots, f_m(x_i, y_i) \rangle$  is a feature vector that encodes  $(x_i, y_i)$  in terms of the features
    - ❖  $c_i$  is an appropriately transformed version of label  $l_i$  (e.g., yes/no or 1/0, depending on what matching model we want to learn)
  - thus  $T$  is transformed into  $T' = \{(v_1, c_1), \dots, (v_n, c_n)\}$
  - apply a learning algorithm (e.g. decision trees, SVMs) to  $T'$  to learn a matching model  $\mathbf{M}$

# Learning-based matching

---

- Applying model **M** to match new tuple pairs
  - given pair  $(x,y)$ , transform it into a feature vector
    - ❖  $v = \langle f_1(x,y), \dots, f_m(x,y) \rangle$
  - apply **M** to  $v$  to predict whether  $x$  matches  $y$

# Example: Learning a linearly weighted rule

$\langle a_1 = \text{(Mike Williams, (425) 247 4893, Seattle, WA), } b_1 = \text{(M. Williams, 247 4893, Redmond, WA), yes} \rangle$

$\langle a_2 = \text{(Richard Pike, (414) 256 1257, Milwaukee, WI), } b_2 = \text{(R. Pike, 256 1237, Milwaukee, WI), yes} \rangle$

$\langle a_3 = \text{(Jane McCain, (206) 111 4215, Renton, WA), } b_3 = \text{(J. M. McCain, 112 5200, Renton, WA), no} \rangle$

- $s_1$  and  $s_2$  use Jaro-Winkler and edit distance
- $s_3$  uses edit distance (ignoring area code of  $a$ )
- $s_4$  and  $s_5$  return 1 if exact match, 0 otherwise
- $s_6$  encodes a heuristic constraint

# Example:

## Learning a linearly weighted rule

- Goal: learn rule  $s(a,b) = \sum_{i=1}^6 \alpha_i s_i(a, b)$
- Perform a least-squares linear regression on training data

$$v_1 = \langle [s_1(a_1, b_1), s_2(a_1, b_1), s_3(a_1, b_1), s_4(a_1, b_1), s_5(a_1, b_1), s_6(a_1, b_1)], 1 \rangle$$

$$v_2 = \langle [s_1(a_2, b_2), s_2(a_2, b_2), s_3(a_2, b_2), s_4(a_2, b_2), s_5(a_2, b_2), s_6(a_2, b_2)], 1 \rangle$$

$$v_3 = \langle [s_1(a_3, b_3), s_2(a_3, b_3), s_3(a_3, b_3), s_4(a_3, b_3), s_5(a_3, b_3), s_6(a_3, b_3)], 0 \rangle$$

to find weights  $\alpha_i$  that minimizes the squared error

$$\sum_{i=1}^3 (c_i - \sum_{j=1}^6 \alpha_j s_j(v_i))^2$$

- $c_i$  is the label associated with  $v_i$

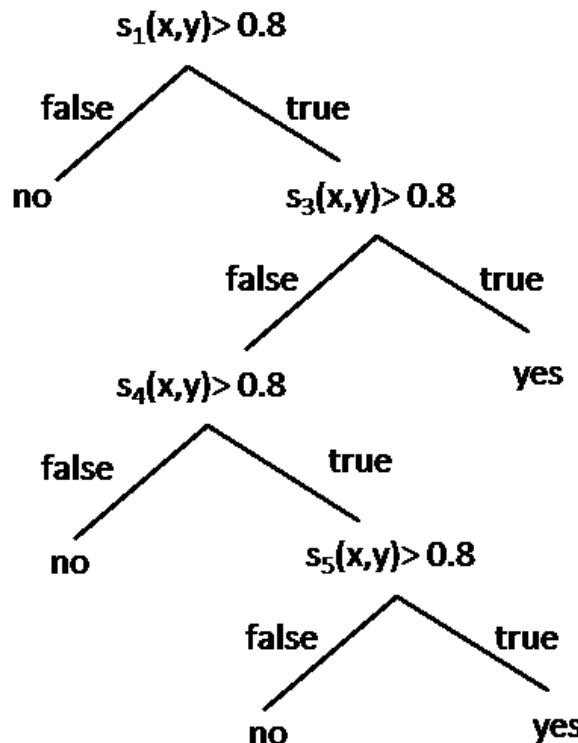
# Example: Learning a decision tree

match names    match phones    match cities    match states    check area code against city

$v_1 = \langle [s_1(a_1, b_1), s_2(a_1, b_1), s_3(a_1, b_1), s_4(a_1, b_1), s_5(a_1, b_1), s_6(a_1, b_1)], \text{yes} \rangle$

$v_2 = \langle [s_1(a_2, b_2), s_2(a_2, b_2), s_3(a_2, b_2), s_4(a_2, b_2), s_5(a_2, b_2), s_6(a_2, b_2)], \text{yes} \rangle$

$v_3 = \langle [s_1(a_3, b_3), s_2(a_3, b_3), s_3(a_3, b_3), s_4(a_3, b_3), s_5(a_3, b_3), s_6(a_3, b_3)], \text{no} \rangle$



Now the labels are yes/no, not 1/0

# Pros and Cons of Learning-based approach

---

- Pros compared to rule-based approaches
  - in rule-based approaches must manually decide if a particular feature is useful → labor intensive and limit the number of features we can consider
  - learning-based ones can automatically examine a large number of features
  - learning-based approaches can construct very complex “rules”
- Cons
  - still require training examples, in many cases a large number of them, which can be hard to obtain
  - clustering addresses this problem

# Outline

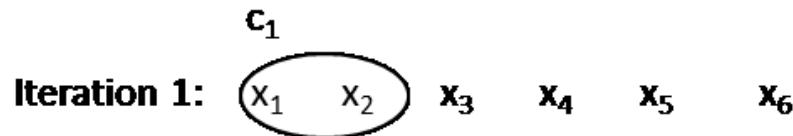
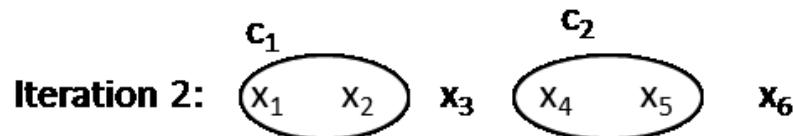
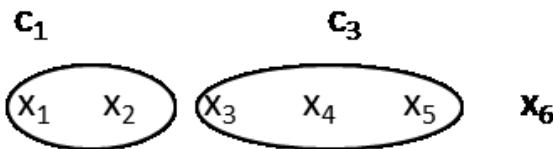
---

- Problem definition
- Rule-based matching
- Learning- based matching
- **Matching by clustering**
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

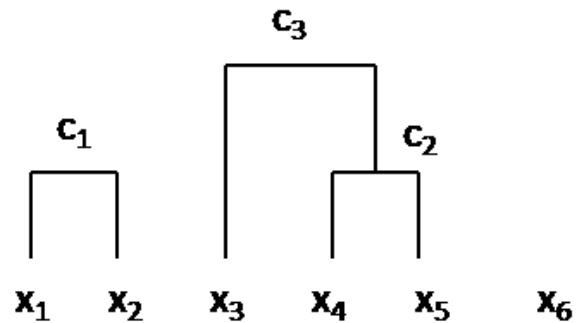
# Matching by clustering

- Many common clustering techniques have been used
  - agglomerative hierarchical clustering (AHC), k-means, graph-theoretic, ...
  - here we focus on AHC, a simple yet very commonly used one
- AHC
  - partitions a given set of **tuples D** into a **set of clusters**
    - ❖ all tuples in a cluster refer to the same real-world entity, tuples in different clusters refer to different entities
  - begins by putting each tuple in D into a single cluster
  - iteratively merges the two most similar clusters
  - stops when a desired number of clusters has been reached, or until the similarity between two closest clusters falls below a pre-specified threshold

# Example



$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6$



- $\text{sim}(x,y) =$   
 $0.3s_{\text{name}}(x,y) + 0.3s_{\text{phone}}(x,y) + 0.1s_{\text{city}}(x,y) + 0.3s_{\text{state}}(x,y)$

# Computing a similarity score between two clusters

---

- Let **c** and **d** be two clusters
- Single link:  $s(c,d) = \min_{x_i \in c, y_j \in d} \text{sim}(x_i, y_j)$
- Complete link:  $s(c,d) = \max_{x_i \in c, y_j \in d} \text{sim}(x_i, y_j)$
- Average link:  $s(c,d) = [\sum_{x_i \in c, y_j \in d} \text{sim}(x_i, y_j)] / [\# \text{ of } (x_i, y_j) \text{ pairs}]$
- Canonical tuple
  - create a canonical tuple that represents each cluster
  - sim between **c** and **d** is the sim between their canonical tuples
  - canonical tuple is created from attribute values of the tuples
    - ❖ e.g., “Mike Williams” and “M. J. Williams” → “Mike J. Williams”
    - ❖ (425) 247 4893 and 247 4893 → (425) 247 4893

# Key ideas underlying the clustering approach

---

- View matching tuples as the problem of constructing entities (i.e., clusters)
- The process is iterative
  - leverage what we have known so far to build “better” entities
- In each iteration merge all matching tuples within a cluster to build an “entity profile”, then use it to match other tuples → merging then exploiting the merged information to help matching
- These same ideas appear in subsequent approaches that we will cover

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- **Probabilistic approaches to matching**
- Collective matching
- Scaling up data matching

# Probabilistic approaches to matching

- Model matching domain using a probability distribution
- Reason with the distribution to make matching decisions
- Key benefits
  - provide a principled framework that can naturally incorporate a variety of domain knowledge
  - can leverage the **wealth of prob representation and reasoning techniques** already developed in the **AI** and **DB** communities
  - provide a frame of reference for comparing and explaining other matching approaches
- Disadvantages
  - computationally expensive
  - often hard to understand and debug matching decisions

# What we discuss next

- Most current probabilistic approaches employ generative models
  - these encode full prob distributions and describe how to generate data that fit the distributions
- Some newer approaches employ discriminative models (e.g., conditional random fields)
  - these encode only the probabilities necessary for matching (e.g., the probability of a label given a tuple pair)
- Here we focus on **generative model based approaches**
  - first we explain Bayesian networks, a simple type of generative models
  - then we use them to explain more complex ones

# Bayesian networks: Motivation

- Let  $X = \{x_1, \dots, x_n\}$  be a set of variables
  - e.g.,  $X = \{\text{Cloud}, \text{Sprinkler}\}$
- A state = an assignment of values to all variables in  $X$ 
  - e.g.,  $s = \{\text{Cloud} = \text{true}, \text{Sprinkler} = \text{on}\}$
- A probability distribution  $P$  assigns to each state  $s_i$  a value  $P(s_i)$  such that  $\sum_{s_i \in S} P(s_i) = 1$ 
  - $S$  is the set of all states
  - $P(s_i)$  is called the probability of  $s_i$

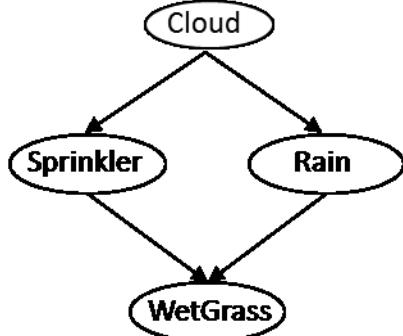
| States |           | Probabilities |
|--------|-----------|---------------|
| Cloud  | Sprinkler |               |
| t      | on        | 0.3           |
| t      | off       | 0.3           |
| f      | on        | 0.3           |
| f      | off       | 0.1           |

# Bayesian networks: Motivation

- Reasoning with prob models: to answer queries such as
  - $P(A = a)? P(A = a | B = b) = ?$  where A and B are subsets of vars
- Examples
  - $P(\text{Cloud} = t) = 0.6$   
(by summing over first two rows)
  - $P(\text{Cloud} = t | \text{Sprinkler} = \text{off}) = 0.75$
- Problems: can't enumerate all states, too many of them
  - real-world apps often use hundreds or thousands of variables
- Bayesian networks solve this by providing a compact representation of a probability distribution

| States |           | Probabilities |
|--------|-----------|---------------|
| Cloud  | Sprinkler |               |
| t      | on        | 0.3           |
| t      | off       | 0.3           |
| f      | on        | 0.3           |
| f      | off       | 0.1           |

# Baysian networks: Representation



| Cloud |     |
|-------|-----|
| t     | f   |
| 0.3   | 0.7 |

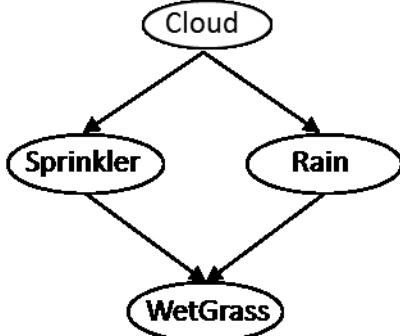
| Cloud | Sprinkler |     |
|-------|-----------|-----|
|       | on        | off |
| t     | 0.2       | 0.8 |
| f     | 0.8       | 0.2 |

| Cloud | Rain |     |
|-------|------|-----|
|       | t    | f   |
| t     | 0.6  | 0.4 |
| f     | 0.3  | 0.7 |

| Sprinkler | Rain | WetGrass |     |
|-----------|------|----------|-----|
|           |      | t        | f   |
| on        | t    | 1        | 0   |
| on        | f    | 1        | 0   |
| off       | t    | 1        | 0   |
| off       | f    | 0.1      | 0.9 |

- nodes = variables, edges = probabilistic dependencies
- Key assertion: each node is probabilistically independent of its non-descendants given the values of its parents
  - e.g., WetGrass is independent of Cloud given Sprinkler & Rain
  - Sprinkler is independent of Rain given Cloud

# Baysian networks: Representation



| Cloud |     |
|-------|-----|
| t     | f   |
| 0.3   | 0.7 |

| Cloud | Sprinkler |     |
|-------|-----------|-----|
|       | on        | off |
| t     | 0.2       | 0.8 |
| f     | 0.8       | 0.2 |

| Cloud | Rain |     |
|-------|------|-----|
|       | t    | f   |
| t     | 0.6  | 0.4 |
| f     | 0.3  | 0.7 |

| Sprinkler | Rain | WetGrass |     |
|-----------|------|----------|-----|
|           |      | t        | f   |
| on        | t    | 1        | 0   |
| on        | f    | 1        | 0   |
| off       | t    | 1        | 0   |
| off       | f    | 0.1      | 0.9 |

- The key assertion allows us to write
  - $P(C,S,R,W) = P(C).P(S|C).P(R|C).P(W|R)$
  - Thus, to compute  $P(C,S,R,W)$ , need to know only four local probability distributions, also called conditional probability tables (CPTs)
  - use only 9 statements to specify the full PD, instead of 16

# Bayesian networks: Reasoning

---

- Also called performing inference
  - computing  $P(A)$  or  $P(A|B)$ , where A and B are subsets of vars
- Performing exact inference is NP-hard
  - taking time exponential in number of variables in worst case
- Data matching approaches address this in three ways
  - for certain classes of BNs there are polynomial-time algorithms or closed-form equations that return exact answers
  - use standard approximate inference algorithms for BNs
  - develop approximate algorithms tailored to the domain at hand

# Learning bayesian Networks

- To use a BN, current data matching approaches
  - typically require a domain expert to create the graph
  - then learn the CPTs from training data
- Training data: set of states we have observed
  - e.g.,  $d_1 = (\text{Cloud}=t, \text{Sprinkler}=\text{off}, \text{Rain}=t, \text{WetGrass}=t)$
  - $d_2 = (\text{Cloud}=t, \text{Sprinkler}=\text{off}, \text{Rain}=f, \text{WetGrass}=f)$
  - $d_3 = (\text{Cloud}=f, \text{Sprinkler}=\text{on}, \text{Rain}=f, \text{WetGrass}=t)$
- Two cases
  - training data has no missing values
  - training data has some missing values
    - ❖ greatly complicates learning, must use EM algorithm
  - we now consider them in turn

# Learning with no missing values



**Training data D**

$d_1 = (1,0)$   
 $d_2 = (1,0)$   
 $d_3 = (1,1)$   
 $d_4 = (0,1)$

**CPTs to be learned**

| A |   |
|---|---|
| 1 | 0 |
| ? | ? |

| A | B |   |
|---|---|---|
|   | 1 | 0 |
| 1 | ? | ? |
| 0 | ? | ? |

**CPTs learned from training data**

| A    |      |
|------|------|
| 1    | 0    |
| 0.75 | 0.25 |

| A | B    |      |
|---|------|------|
|   | 1    | 0    |
| 1 | 0.33 | 0.67 |
| 0 | 1    | 0    |

- $d_1 = (1,0)$  means  $A = 1$  and  $B = 0$

# Learning with No Missing Values

- Let  $\theta$  be the probabilities to be learned. Want to find  $\theta^*$  that maximizes the prob of observing the training data D
  - $\theta^* = \arg \max_{\theta} P(D | \theta)$
- $\theta^*$  can be obtained by simple counting over D
- E.g., to compute  $P(A = 1)$ : count # of examples where  $A = 1$ , divide by total # of examples
- To compute  $P(B = 1 | A = 1)$ : divide # of examples where  $B = 1$  and  $A = 1$  by # of examples where  $A = 1$
- What if not having sufficient data for certain states?
  - e.g., need to compute  $P(B=1|A=1)$ , but # states where  $A = 1$  is 0
  - need smoothing of the probabilities (see notes)

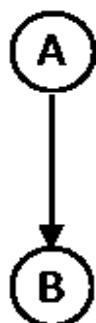
# Learning with Missing Values

---

- Training examples may have missing values
  - $d = (\text{Cloud}=? , \text{Sprinkler}=\text{off}, \text{Rain}=? , \text{WetGrass}=t)$
- Why?
  - we failed to observe a variable
    - ❖ e.g., slept and did not observe whether it rained
  - the variable by its nature is unobservable
    - ❖ e.g., werewolves who only get out during dark moonless night  
→ can't never tell if the sky is cloudy
- Can't use counting as before to learn (e.g., infer CPTs)
- Use EM algorithm

# The Expectation-Maximization (EM) Algorithm

- Key idea:
  - two unknown quantities:  $\theta$  and missing values in D
  - iteratively estimates these two, by assigning initial values, then using one to predict the other and vice versa, until convergence



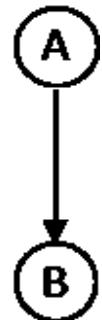
Training data D

$d_1 = (?, 0)$   
 $d_2 = (?, 0)$   
 $d_3 = (?, 1)$

The EM algorithm

1. Initialize  $\theta$  and let it be  $\theta^0$ . Set  $n = 0$ .
2. (Expectation) Use  $\theta^n$  to estimate missing values of D.  
Let the resulting set be  $D^n$ .
3. (Maximization) Compute  $\theta^{n+1}$  using counting over  $D^n$ .
4. Exit and return  $\theta^n$  if no increase is achieved for  $P(\theta^n | D^n)$ .  
Otherwise repeat Steps 2-3 with  $n = n + 1$ .

# An Example



**Training data D**

$$\begin{aligned} d_1 &= (? , 0) \\ d_2 &= (? , 0) \\ d_3 &= (? , 1) \end{aligned}$$

| A   |     |
|-----|-----|
| 1   | 0   |
| 0.5 | 0.5 |

| A | B   |     |
|---|-----|-----|
|   | 1   | 0   |
| 1 | 0.6 | 0.4 |
| 0 | 0.5 | 0.5 |

$\theta^0$

$D^0$

$\theta^1$

$$d_1 = \begin{bmatrix} P(A=1) = 0.44, B=0 \\ P(A=0) = 0.56 \end{bmatrix}$$

$$d_2 = \begin{bmatrix} P(A=1) = 0.44, B=0 \\ P(A=0) = 0.56 \end{bmatrix}$$

$$d_3 = \begin{bmatrix} P(A=1) = 0.54, B=1 \\ P(A=0) = 0.46 \end{bmatrix}$$

| A    |      |
|------|------|
| 1    | 0    |
| 0.47 | 0.53 |

| A | B    |      |
|---|------|------|
|   | 1    | 0    |
| 1 | 0.38 | 0.62 |
| 0 | 0.29 | 0.71 |

- EM also aims to find  $\theta$  that maximizes  $P(D|\theta)$ 
  - just like the counting approach in case of no missing values
- It may not find the globally maximal  $\theta^*$ 
  - converging instead to a local maximum

# Bayesian Networks as Generative Models

---

- Generative models
  - encode full probability distributions
  - specify how to generate data that fit such distributions
- Bayesian networks: well-known examples of such models
- A perspective on how the data is generated helps
  - guide the construction of the Bayesian network
  - discover what kinds of domain knowledge to be naturally incorporated into the network structure
  - explain the network to users
- We now examine three prob approaches to matching that employ increasingly complex generative models

# Data Matching with Naïve Bayes

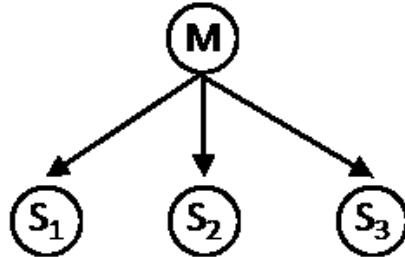
---

- Define variable  $M$  that represents whether  $a$  and  $b$  match
- Our goal is to compute  $P(M|a,b)$ 
  - declare  $a$  and  $b$  matched if  $P(M=t|a,b) > P(M=f|a,b)$
- Assume  $P(M|a,b)$  depends only on  $S_1, \dots, S_n$ , features that are functions that take as input  $a$  and  $b$ 
  - e.g., whether two last names match, edit distance between soc sec numbers, whether the first initials match, etc.
- $P(M|a,b) = P(M|S_1, \dots, S_n)$ , using Bayes Rule, we have
  - $P(M|S_1, \dots, S_n) = P(S_1, \dots, S_n|M)P(M)/P(S_1, \dots, S_n)$

# Data Matching with Naïve Bayes

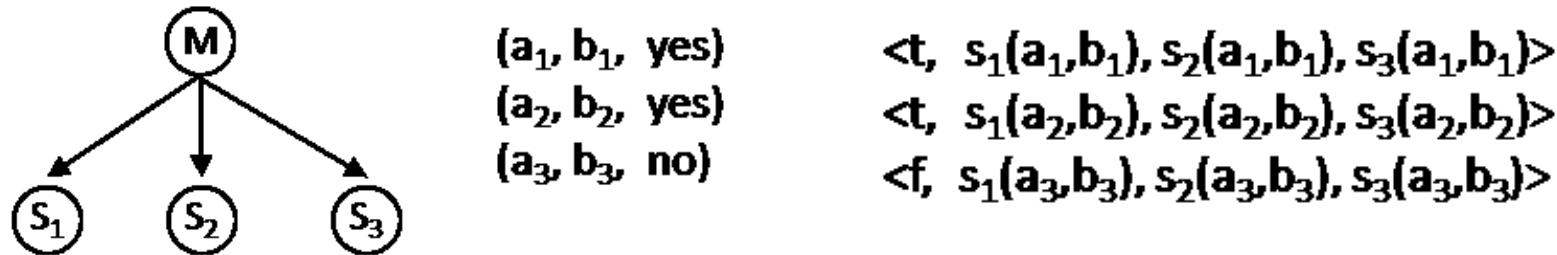
- $P(M|S_1, \dots, S_n) = P(S_1, \dots, S_n|M)P(M)/P(S_1, \dots, S_n)$
- Assume  $S_1, \dots, S_n$  are independent given M
  - $P(S_1, \dots, S_n|M) = \prod_{i=1}^n P(S_i|M)$
- We also have
  - $P(S_1, \dots, S_n) = P(S_1, \dots, S_n|M=t)P(M=t) + P(S_1, \dots, S_n|M=f)P(M=f)$
- So, to compute  $P(M|S_1, \dots, S_n)$ , i.e.,  $P(M|a,b)$ , we only need to know  $P(S_1|M), \dots, P(S_n|M)$ , and  $P(M)$
- The above model is captured in a Bayesian network called a Naïve Bayes model

# The Naïve Bayes Model



- The assumption that  $S_1, \dots, S_n$  are independent of one another given  $M$  is called the Naïve Bayes assumption
  - which often does not hold in practice
- Computing  $P(M | S_1, \dots, S_n)$  is performing an inference on the above Bayesian network
- Given the simple form of the network, this inference can be performed easily, if we know the CPTs

# Learning the CPTs Given Training Data



- Convert training examples into feature vectors
- Then apply learning with no missing values as described earlier (i.e., counting) to feature vectors to learn the CPTs
- Once learned, can apply the BN to match a new pair of tuples ( $a, b$ ) by comparing  $P(M=t|a,b)$  and  $P(M=f|a,b)$ 
  - this reduces to comparing  $\prod_{i=1}^n P(S_i|M = t)P(M = t)$  and  $\prod_{i=1}^n P(S_i|M = f)P(M = f)$
  - no need to compute  $P(S_1, \dots, S_n)$

# Learning the CPTs Given No Training Data

$(a_4, b_4)$

$\langle ?, s_1(a_4, b_4), s_2(a_4, b_4), s_3(a_4, b_4) \rangle$

$(a_5, b_5)$

$\langle ?, s_1(a_5, b_5), s_2(a_5, b_5), s_3(a_5, b_5) \rangle$

$(a_6, b_6)$

$\langle ?, s_1(a_6, b_6), s_2(a_6, b_6), s_3(a_6, b_6) \rangle$

- Assume  $(a_4, b_4), \dots, (a_6, b_6)$  are tuple pairs to be matched
- Convert these pairs into training data with missing values
  - the missing value is the correct label for each pair (i.e., the value for variable M: “matched”, “not matched”)
- Now apply EM algorithm to learn both the CPTs and the missing values at the same time
  - once learned, the missing values are the labels (i.e., “matched”, “not matched”) that we want to see

# Summary

- The developer specifies the network structure, i.e., the directed acyclic graph
  - which is a Naïve Bayesian network structure in this case
- If given training data in form of tuple pairs together with their correct labels (matched, not matched), we can learn the CPTs of the Naïve Bayes network using counting
  - then we use the trained network to match new tuple pairs (which means performing exact inferences to compute  $P(M|a,b)$ )
- People also refer to the Naïve Bayesian network as a Naïve Bayesian classifier

# Summary (cont.)

- If no training data is given, but we are given a set of tuple pairs to be matched, then we can use these tuple pairs to construct training data with missing values
  - we then apply EM to learn the missing values and the CPTs
  - the missing values are the match predictions that we want
- The above procedures (for both cases of having and not having training data) can be generalized in a straightforward fashion to arbitrary Bayesian network cases, not just Naïve Bayesian ones

# Modeling Feature Correlations

- Naïve Bayes assumes no correlations among  $S_1, \dots, S_n$

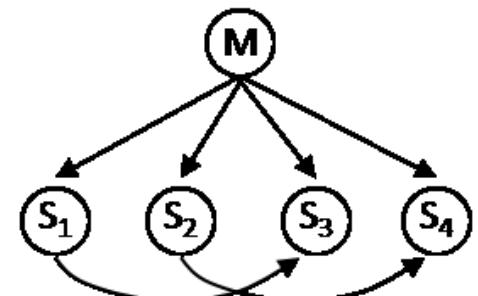
- We may want to model such correlations

- e.g., if  $S_1$  models whether soc sec numbers match, and  $S_3$  models whether last names match, then there exists a correlation between the two

- We can then train and apply this more expressive BN to match data

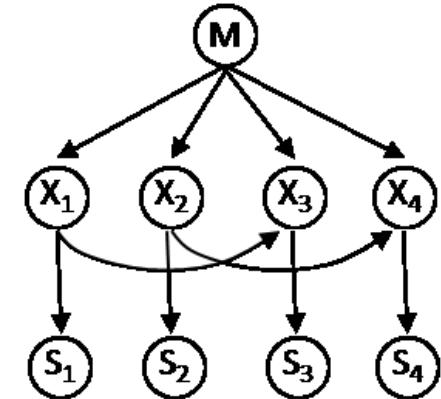
- Problem: “blow up” the number of probs in the CPTs

- assume  $n$  is # of features,  $q$  is the # of parents per node, and  $d$  is the # of values per node  $\rightarrow O(nd^q)$  vs.  $2dn$  for the comparable Naïve Bayesian



# Modeling Feature Correlations

- A possible solution
  - assume each tuple has  $k$  attributes
  - consider only  $k$  features  $S_1, \dots, S_k$ ,  
the  $i$ -th feature compares only values of  
the  $i$ -th attributes
  - introduce binary variables  $X_i$ ,  $X_i$  models whether the  $i$ -th  
attributes should match, given that the tuples match
  - then model correlation only at the  $X_i$  level, not at  $S_i$  level
- This requires far fewer probs in CPTs
  - assume each node has  $q$  parents, and each  $S_i$  has  $d$  values,  
then we need  $O(k2^q + 2kd)$  probs



# Key Lesson

---

- Constructing a BN for a matching problem is an art that must consider the trade-offs among many factors
  - how much domain knowledge to be captured
  - how accurately we can learn the network
  - how efficiently we can do so
- The notes present an even more complex example about matching mentions of entities in text

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- **Collective matching**
- Scaling up data matching

# Collective Matching

---

- Matching approaches discussed so far make independent matching decisions
  - decide whether **a** and **b** match independently of whether any two other tuples **c** and **d** match
- Matching decisions however are often correlated
  - exploiting such correlations can improve matching accuracy

# An Example

**W. Wang, C. Chen, A. Ansari, A mouse immunity model**

**W. Wang, A. Ansari, Evaluating immunity models**

**L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine**

**W. J. Wang, A. Ansari, Autoimmunity in biliary cirrhosis**

|                       | First initial | Middle initial | Last name |
|-----------------------|---------------|----------------|-----------|
| <b>a<sub>1</sub></b>  | W             |                | Wang      |
| <b>a<sub>2</sub></b>  | C             |                | Chen      |
|                       | ...           | ...            | ...       |
| <b>a<sub>9</sub></b>  | W             | J              | Wang      |
| <b>a<sub>10</sub></b> | A             |                | Ansari    |

- Goal: match authors of the four papers listed above
- Solution
  - extract their names to create the table above
  - apply current approaches to match tuples in table
- This fails to exploit co-author relationships in the data

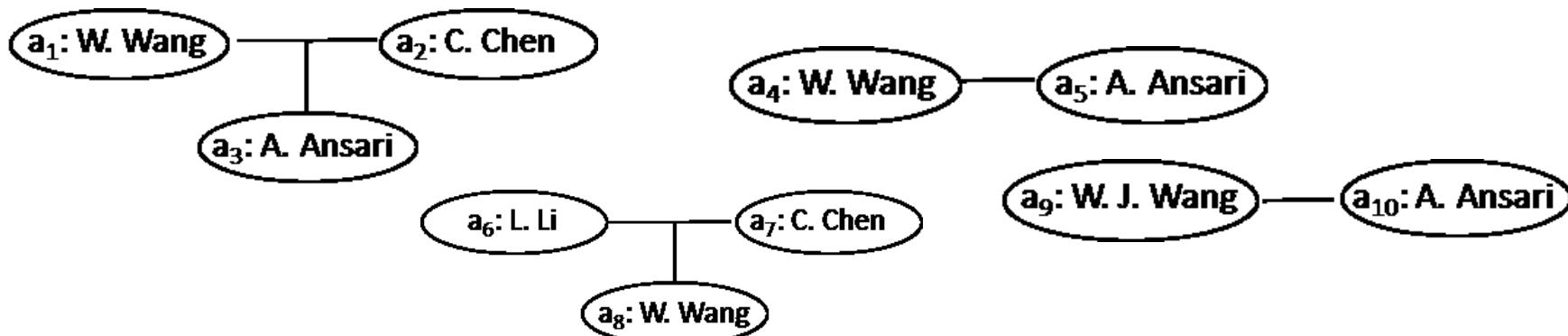
# An Example (cont.)

W. Wang, C. Chen, A. Ansari, A mouse immunity model

W. Wang, A. Ansari, Evaluating immunity models

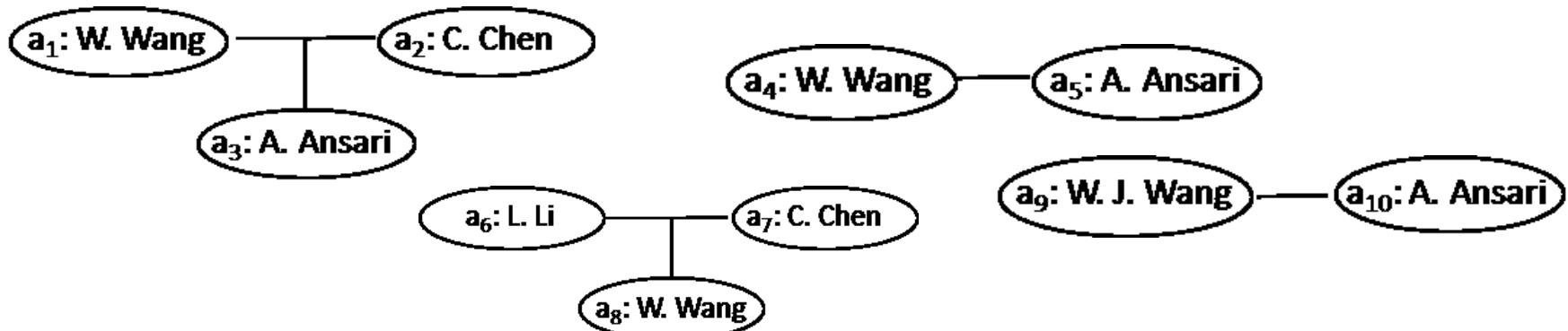
L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine

W. J. Wang, A. Ansari, Autoimmunity in biliary cirrhosis



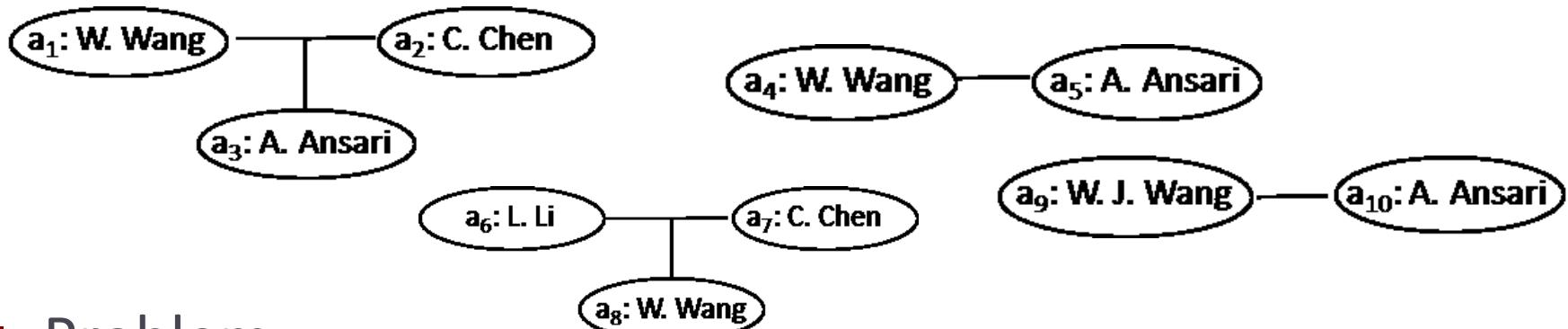
- nodes = authors, hyperedges connect co-authors
- Suppose we have matched  $a_3$  and  $a_5$ 
  - then intuitively  $a_1$  and  $a_4$  should be more likely to match
  - they share the same name and same co-author relationship to the same author

# An Example (cont.)



- First solution:
  - add coAuthors attribute to the tuples
    - ❖ e.g., tuple a<sub>1</sub> has coAuthors = {C. Chen, A. Ansari}
    - ❖ tuple a<sub>4</sub> has coAuthors = {A. Ansari}
  - apply current methods, use say Jaccard measure for coAuthors

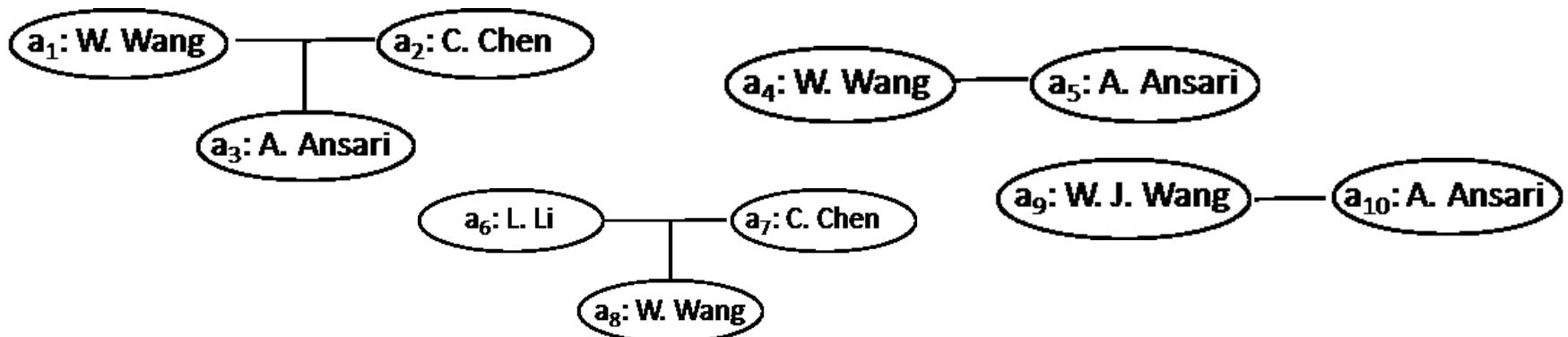
# An Example (cont.)



- Problem:
  - suppose  $a_3$ : A. Ansari and  $a_5$ : A. Ansari share same name but do not match
  - we would match them, and incorrectly boost score of  $a_1$  and  $a_4$
- How to fix this?
  - want to match  $a_3$  and  $a_5$ , then use that info to help match  $a_1$  and  $a_4$ ; also want to do the opposite
  - so should match tuples collectively, all at once and iteratively

# Collective Matching using Clustering

- Many collective matching approaches exist
  - clustering-based, probabilistic, etc.
- Here we consider clustering-based (see notes for more)
- Assume input is graph
  - nodes = tuples to be matched
  - edges = relationships among tuples



# Collective Matching using Clustering

---

- To match, perform agglomerative hierarchical clustering
  - but modify sim measure to consider correlations among tuples
- Let A and B be two clusters of nodes, define
  - $\text{sim}(A,B) = \alpha * \text{sim}_{\text{attributes}}(A,B) + (1 - \alpha) * \text{sim}_{\text{neighbors}}(A,B)$
  - $\alpha$  is pre-defined weight
  - $\text{sim}_{\text{attributes}}(A,B)$  uses only attributes of A and B, examples of such scores are single link, complete link, average link, etc.
- $\text{sim}_{\text{neighbors}}(A,B)$  considers correlations
  - we discuss it next

# An Example of $\text{sim}_{\text{neighbors}}(A, B)$

---

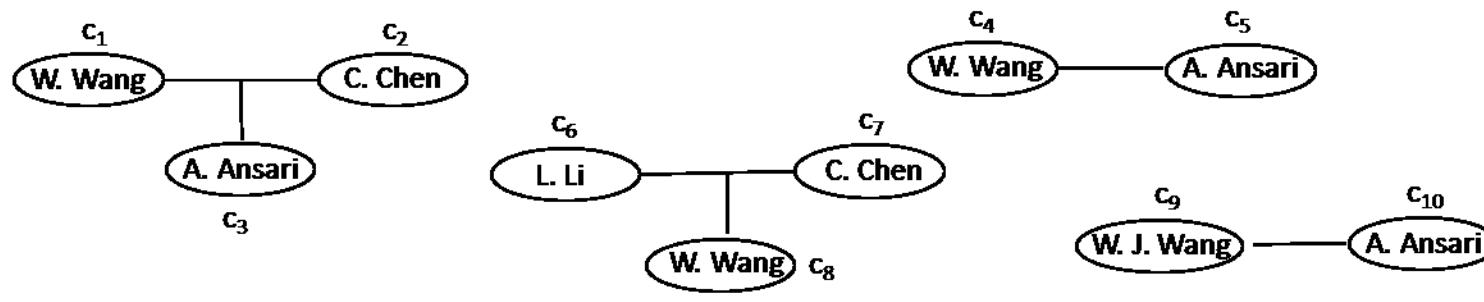
- Assume a single relationship R on the graph edges
  - can generalize to the case of multiple relationships
- Let  $N(A)$  be the bags of the cluster IDs of all nodes that are in relationship R with some node in A
  - e.g., cluster A has two nodes a and a',  
a is in relationship R with node b with cluster ID 3, and  
a' is in relationship R with node b' with cluster ID 3  
and another node b'' with cluster ID 5  
 $\rightarrow N(A) = \{3, 3, 5\}$
- Define  $\text{sim}_{\text{neighbors}}(A, B) =$   
$$\text{Jaccard}(N(A), N(B)) = |N(A) \cap N(B)| / |N(A) \cup N(B)|$$

# An Example of $\text{sim}_{\text{neighbors}}(A, B)$

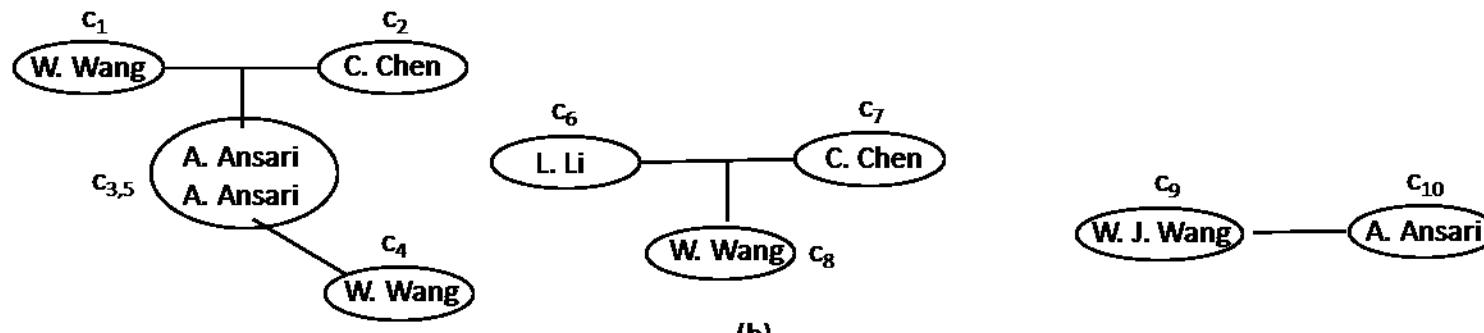
---

- Recall that earlier we also define a Jaccard measure as
  - $\text{JaccardSim}_{\text{coAuthors}}(a, b) = \frac{|\text{coAuthors}(a) \cap \text{coAuthors}(b)|}{|\text{coAuthors}(a) \cup \text{coAuthors}(b)|}$
- Contrast that to
  - $\text{sim}_{\text{neighbors}}(A, B) = \text{Jaccard}(N(A), N(B)) = \frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|}$
- In the former, we assume two co-authors match if their “strings” match
- In the latter, two co-authors match only if they have the same cluster ID

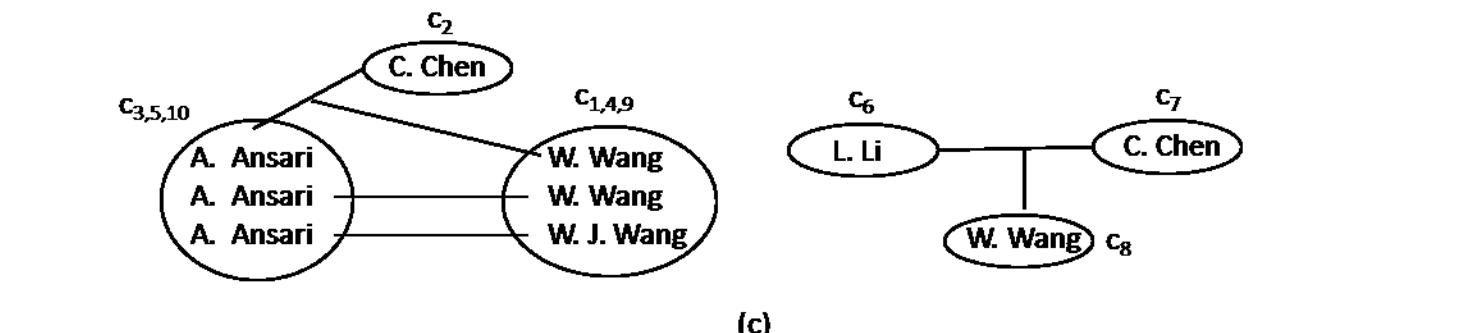
# An Example to Illustrate the Working of Agglomerative Hierarchical Clustering



(a)



(b)



(c)

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

# Scaling up Rule-based Matching

- Two goals: minimize # of tuple pairs to be matched and minimize time it takes to match each pair
- For the first goal:
  - hashing
  - sorting
  - indexing
  - canopies
  - using representatives
  - combining the techniques
- Hashing
  - hash tuples into buckets, match only tuples within each bucket
  - e.g., hash house listings by zipcode, then match within each zip

# Scaling up Rule-based Matching

## ■ Sorting

- use a key to sort tuples, then scan the sorted list and match each tuple with only the previous  $(w-1)$  tuples, where  $w$  is a pre-specified window size
- key should be strongly “discriminative”: brings together tuples that are likely to match, and pushes apart tuples that are not
  - ❖ example keys: soc sec, student ID, last name, soundex value of last name
- employs a stronger heuristic than hashing: also requires that tuples likely to match be within a window of size  $w$ 
  - ❖ but is often faster than hashing because it would match fewer pairs

# Scaling up Rule-based Matching

- Indexing
  - index tuples such that given any tuple  $a$ , can use the index to quickly locate a relatively small set of tuples that are likely to match  $a$ 
    - ❖ e.g., inverted index on names
- Canopies
  - use a computationally cheap sim measure to quickly group tuples into overlapping clusters called canopies (or umbrella sets)
  - use a different (far more expensive) sim measure to match tuples within each canopy
  - e.g., use TF/IDF to create canopies

# Scaling up Rule-based Matching

---

- Using representatives
  - applied during the matching process
  - assigns tuples that have been matched into groups such that those within a group match and those across groups do not
  - create a representative for each group by selecting a tuple in the group or by merging tuples in the group
  - when considering a new tuple, only match it with the representatives
- Combining the techniques
  - e.g., hash houses into buckets using zip codes, then sort houses within each bucket using street names, then match them using a sliding window

# Scaling up Rule-based Matching

---

- For the second goal of minimizing time it takes to match each pair
  - no well-established technique as yet
  - tailor depending on the application and the matching approach
  - e.g., if using a simple rule-based approach that matches individual attributes then combines their scores using weights
    - ❖ can use short circuiting: stop the computation of the sim score if it is already so high that the tuple pair will match even if the remaining attributes do not match

# Scaling up Other Matching Methods

- Learning, clustering, probabilistic, and collective approaches often face similar scalability challenges, and can benefit from the same solutions
- Probabilistic approaches raise additional challenges
  - if model has too many parameters → difficult to learn efficiently, need a large # of training data to learn accurately
  - make independence assumptions to reduce # of parameters
- Once learned, inference with model is also time costly
  - use approximate inference algorithms
  - simplify model so that closed form equations exist
- EM algorithm can be expensive
  - truncate EM, or initializing it as accurately as possible

# Scaling up Using Parallel Processing

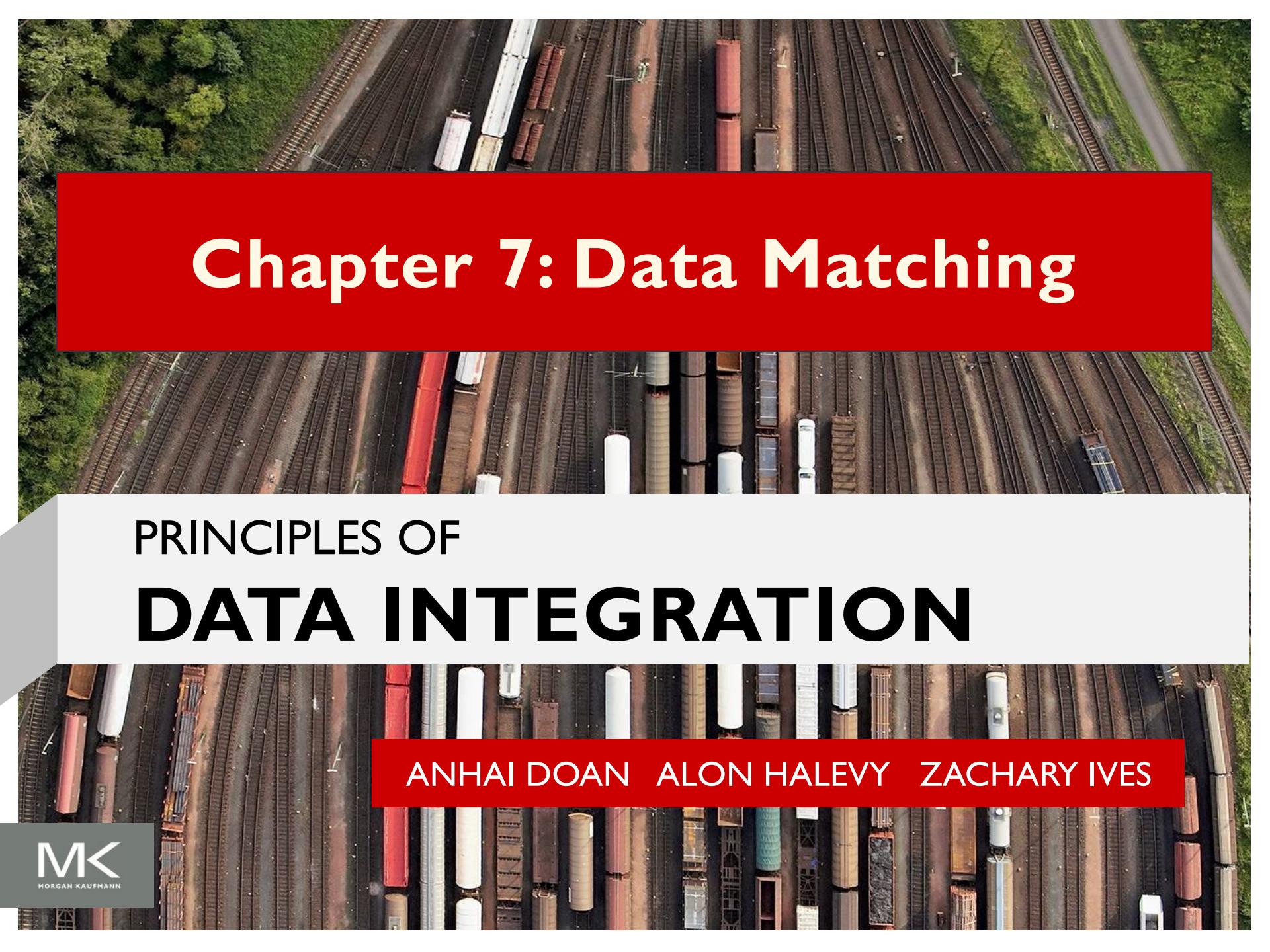
---

- Commonly done in practice
- Examples
  - hash tuples into buckets, then match each bucket in parallel
  - match tuples against a taxonomy of entities (e.g., a product or Wikipedia-like concept taxonomy) in parallel
    - ❖ two tuples are declared matched if they match into the same taxonomic node
    - ❖ a variant of using representatives to scale up, discussed earlier

# Summary

---

- Critical problem in data integration
- Huge amount of work in academia and industry
  - Rule-based matching
  - Learning- based matching
  - Matching by clustering
  - Probabilistic approaches to matching
  - Collective matching
- This chapter has covered only the most common and basic approaches
- The bibliography discusses much more



# Chapter 7: Data Matching

## PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Introduction

- **Data matching**: find structured data items that refer to the same real-world entity
  - **entities** may be represented by tuples, XML elements, or RDF triples, not by strings as in string matching
  - e.g., (David Smith, 608-245-4367, Madison WI)  
vs (D. M. Smith, 245-4367, Madison WI)
- Data matching arises in many integration scenarios
  - merging multiple databases with the same schema
  - joining rows from sources with different schemas
  - matching a user query to a data item
- One of the most fundamental problems in data integration

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

# Collective Matching

---

- Matching approaches discussed so far make independent matching decisions
  - decide whether **a** and **b** match independently of whether any two other tuples **c** and **d** match
- Matching decisions however are often correlated
  - exploiting such correlations can improve matching accuracy

# An Example

**W. Wang, C. Chen, A. Ansari, A mouse immunity model**

**W. Wang, A. Ansari, Evaluating immunity models**

**L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine**

**W. J. Wang, A. Ansari, Autoimmunity in biliary cirrhosis**

|                       | First initial | Middle initial | Last name |
|-----------------------|---------------|----------------|-----------|
| <b>a<sub>1</sub></b>  | W             |                | Wang      |
| <b>a<sub>2</sub></b>  | C             |                | Chen      |
|                       | ...           | ...            | ...       |
| <b>a<sub>9</sub></b>  | W             | J              | Wang      |
| <b>a<sub>10</sub></b> | A             |                | Ansari    |

- Goal: match authors of the four papers listed above
- Solution
  - extract their names to create the table above
  - apply current approaches to match tuples in table
- This fails to exploit co-author relationships in the data

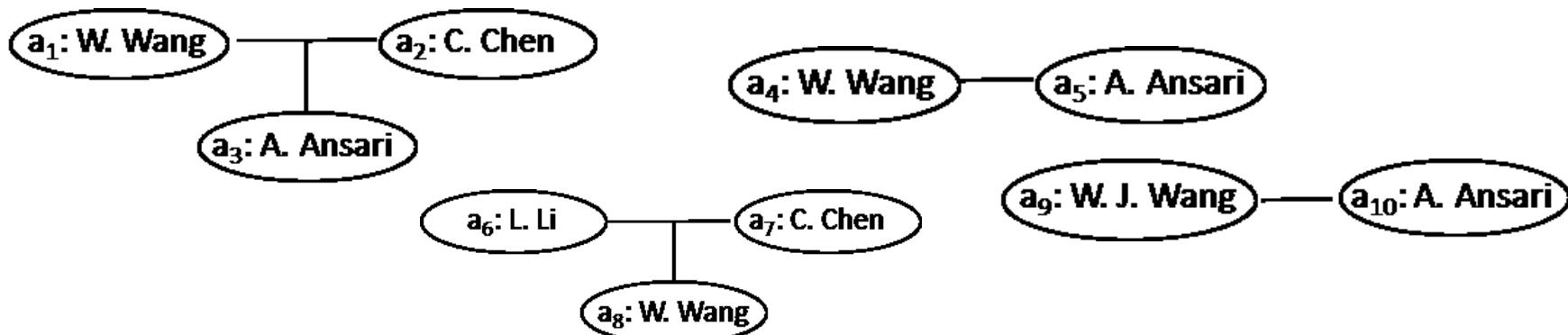
# An Example (cont.)

W. Wang, C. Chen, A. Ansari, A mouse immunity model

W. Wang, A. Ansari, Evaluating immunity models

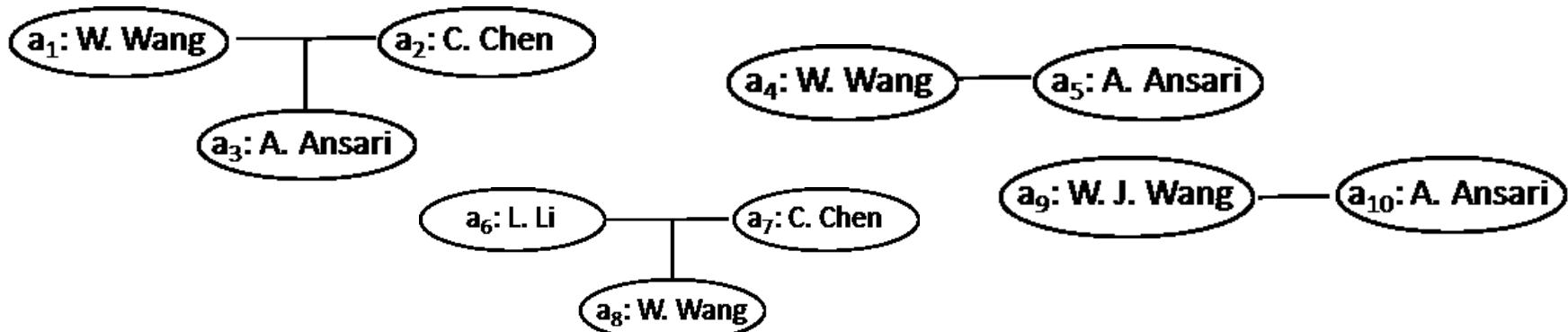
L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine

W. J. Wang, A. Ansari, Autoimmunity in biliary cirrhosis



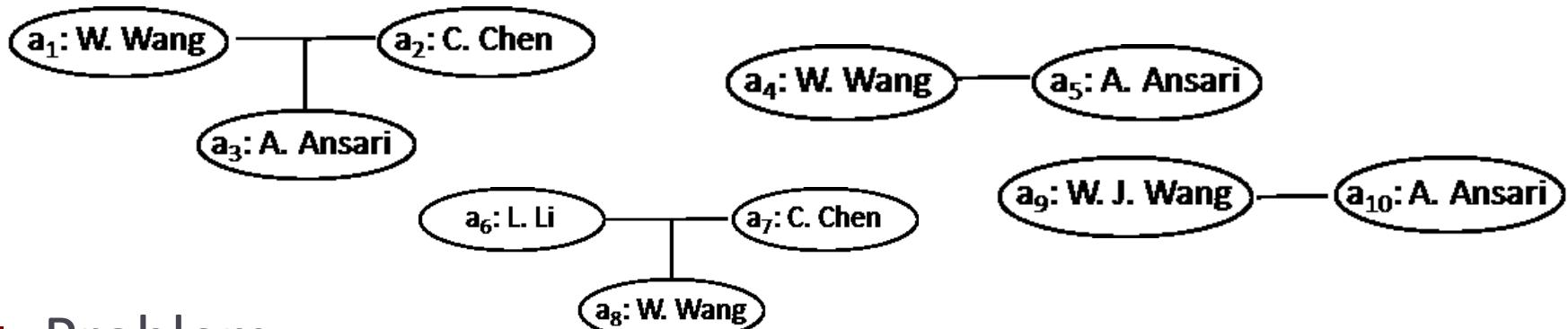
- nodes = authors, hyperedges connect co-authors
- Suppose we have matched  $a_3$  and  $a_5$ 
  - then intuitively  $a_1$  and  $a_4$  should be more likely to match
  - they share the same name and same co-author relationship to the same author

# An Example (cont.)



- First solution:
  - add coAuthors attribute to the tuples
    - ❖ e.g., tuple a<sub>1</sub> has coAuthors = {C. Chen, A. Ansari}
    - ❖ tuple a<sub>4</sub> has coAuthors = {A. Ansari}
  - apply current methods, use say Jaccard measure for coAuthors

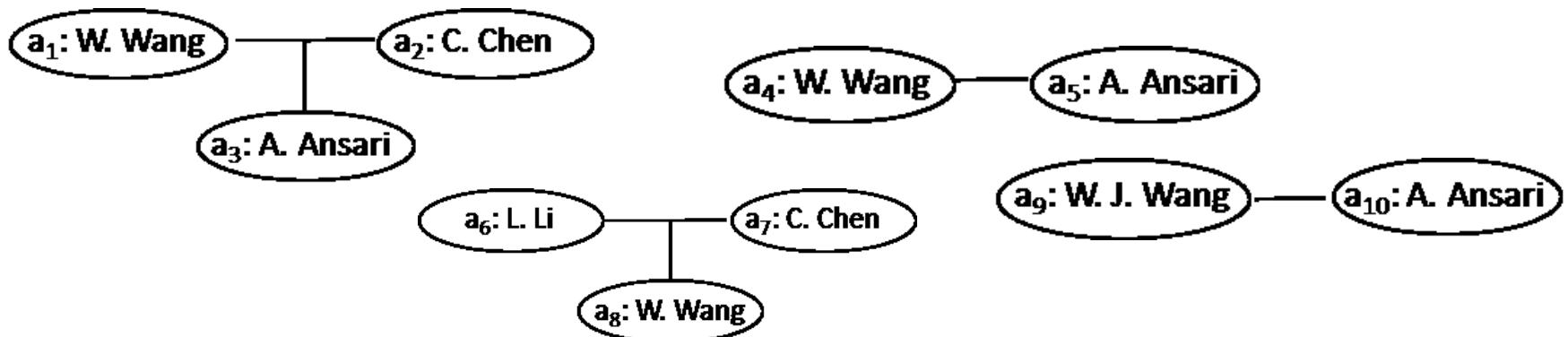
# An Example (cont.)



- Problem:
  - suppose  $a_3$ : A. Ansari and  $a_5$ : A. Ansari share same name but do not match
  - we would match them, and incorrectly boost score of  $a_1$  and  $a_4$
- How to fix this?
  - want to match  $a_3$  and  $a_5$ , then use that info to help match  $a_1$  and  $a_4$ ; also want to do the opposite
  - so should match tuples collectively, all at once and iteratively

# Collective Matching using Clustering

- Many collective matching approaches exist
  - clustering-based, probabilistic, etc.
- Here we consider clustering-based (see notes for more)
- Assume input is graph
  - nodes = tuples to be matched
  - edges = relationships among tuples



# Collective Matching using Clustering

---

- To match, perform agglomerative hierarchical clustering
  - but modify sim measure to consider correlations among tuples
- Let A and B be two clusters of nodes, define
  - $\text{sim}(A,B) = \alpha * \text{sim}_{\text{attributes}}(A,B) + (1 - \alpha) * \text{sim}_{\text{neighbors}}(A,B)$
  - $\alpha$  is pre-defined weight
  - $\text{sim}_{\text{attributes}}(A,B)$  uses only attributes of A and B, examples of such scores are single link, complete link, average link, etc.
- $\text{sim}_{\text{neighbors}}(A,B)$  considers correlations
  - we discuss it next

# An Example of $\text{sim}_{\text{neighbors}}(A, B)$

---

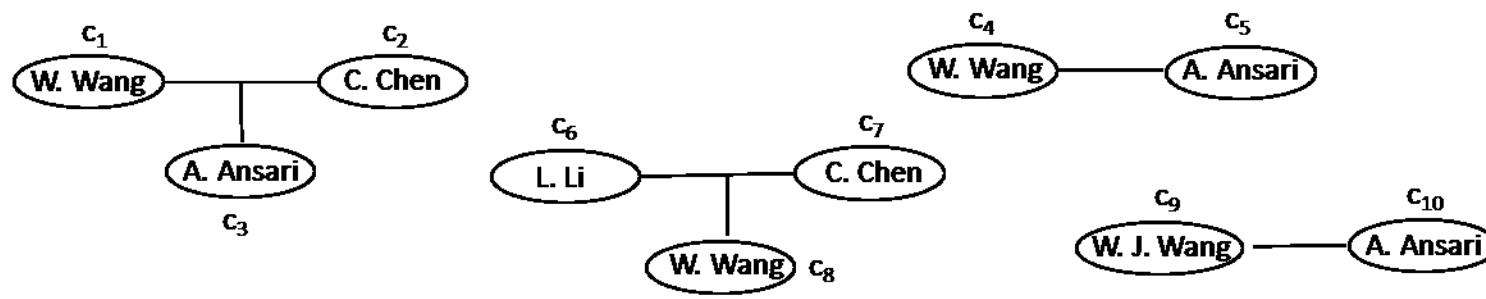
- Assume a single relationship R on the graph edges
  - can generalize to the case of multiple relationships
- Let  $N(A)$  be the bags of the cluster IDs of all nodes that are in relationship R with some node in A
  - e.g., cluster A has two nodes a and a',  
a is in relationship R with node b with cluster ID 3, and  
a' is in relationship R with node b' with cluster ID 3  
and another node b'' with cluster ID 5  
 $\rightarrow N(A) = \{3, 3, 5\}$
- Define  $\text{sim}_{\text{neighbors}}(A, B) =$   
$$\text{Jaccard}(N(A), N(B)) = |N(A) \cap N(B)| / |N(A) \cup N(B)|$$

# An Example of $\text{sim}_{\text{neighbors}}(A, B)$

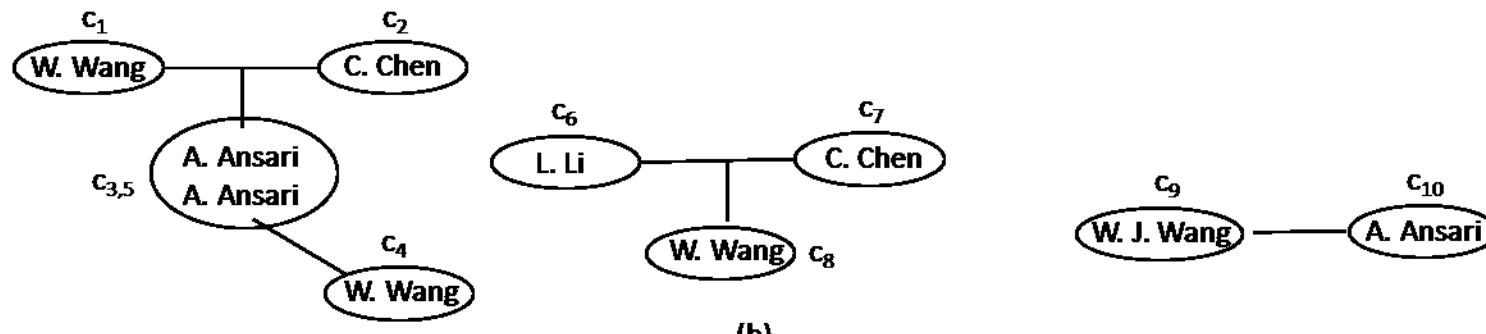
---

- Recall that earlier we also define a Jaccard measure as
  - $\text{JaccardSim}_{\text{coAuthors}}(a, b) = \frac{|\text{coAuthors}(a) \cap \text{coAuthors}(b)|}{|\text{coAuthors}(a) \cup \text{coAuthors}(b)|}$
- Contrast that to
  - $\text{sim}_{\text{neighbors}}(A, B) = \text{Jaccard}(N(A), N(B)) = \frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|}$
- In the former, we assume two co-authors match if their “strings” match
- In the latter, two co-authors match only if they have the same cluster ID

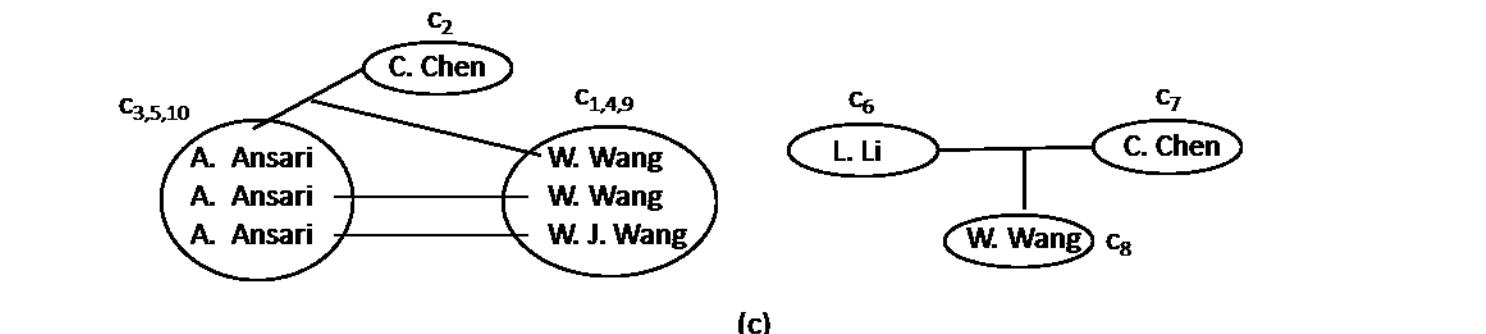
# An Example to Illustrate the Working of Agglomerative Hierarchical Clustering



(a)



(b)



(c)

# Outline

---

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

# Scaling up Rule-based Matching

- Two goals: minimize # of tuple pairs to be matched and minimize time it takes to match each pair
- For the first goal:
  - hashing
  - sorting
  - indexing
  - canopies
  - using representatives
  - combining the techniques
- Hashing
  - hash tuples into buckets, match only tuples within each bucket
  - e.g., hash house listings by zipcode, then match within each zip

# Scaling up Rule-based Matching

---

- Sorting
  - use a key to sort tuples, then scan the sorted list and match each tuple with only the previous  $(w-1)$  tuples, where  $w$  is a pre-specified window size
  - key should be strongly “discriminative”: brings together tuples that are likely to match, and pushes apart tuples that are not
    - ❖ example keys: soc sec, student ID, last name, soundex value of last name
  - employs a stronger heuristic than hashing: also requires that tuples likely to match be within a window of size  $w$ 
    - ❖ but is often faster than hashing because it would match fewer pairs

# Scaling up Rule-based Matching

---

- Indexing
  - index tuples such that given any tuple  $a$ , can use the index to quickly locate a relatively small set of tuples that are likely to match  $a$ 
    - ❖ e.g., inverted index on names
- Canopies
  - use a computationally cheap sim measure to quickly group tuples into overlapping clusters called canopies (or umbrella sets)
  - use a different (far more expensive) sim measure to match tuples within each canopy
  - e.g., use TF/IDF to create canopies

# Scaling up Rule-based Matching

---

- Using representatives
  - applied during the matching process
  - assigns tuples that have been matched into groups such that those within a group match and those across groups do not
  - create a representative for each group by selecting a tuple in the group or by merging tuples in the group
  - when considering a new tuple, only match it with the representatives
- Combining the techniques
  - e.g., hash houses into buckets using zip codes, then sort houses within each bucket using street names, then match them using a sliding window

# Scaling up Rule-based Matching

---

- For the second goal of minimizing time it takes to match each pair
  - no well-established technique as yet
  - tailor depending on the application and the matching approach
  - e.g., if using a simple rule-based approach that matches individual attributes then combines their scores using weights
    - ❖ can use short circuiting: stop the computation of the sim score if it is already so high that the tuple pair will match even if the remaining attributes do not match

# Scaling up Other Matching Methods

- Learning, clustering, probabilistic, and collective approaches often face similar scalability challenges, and can benefit from the same solutions
- Probabilistic approaches raise additional challenges
  - if model has too many parameters → difficult to learn efficiently, need a large # of training data to learn accurately
  - make independence assumptions to reduce # of parameters
- Once learned, inference with model is also time costly
  - use approximate inference algorithms
  - simplify model so that closed form equations exist
- EM algorithm can be expensive
  - truncate EM, or initializing it as accurately as possible

# Scaling up Using Parallel Processing

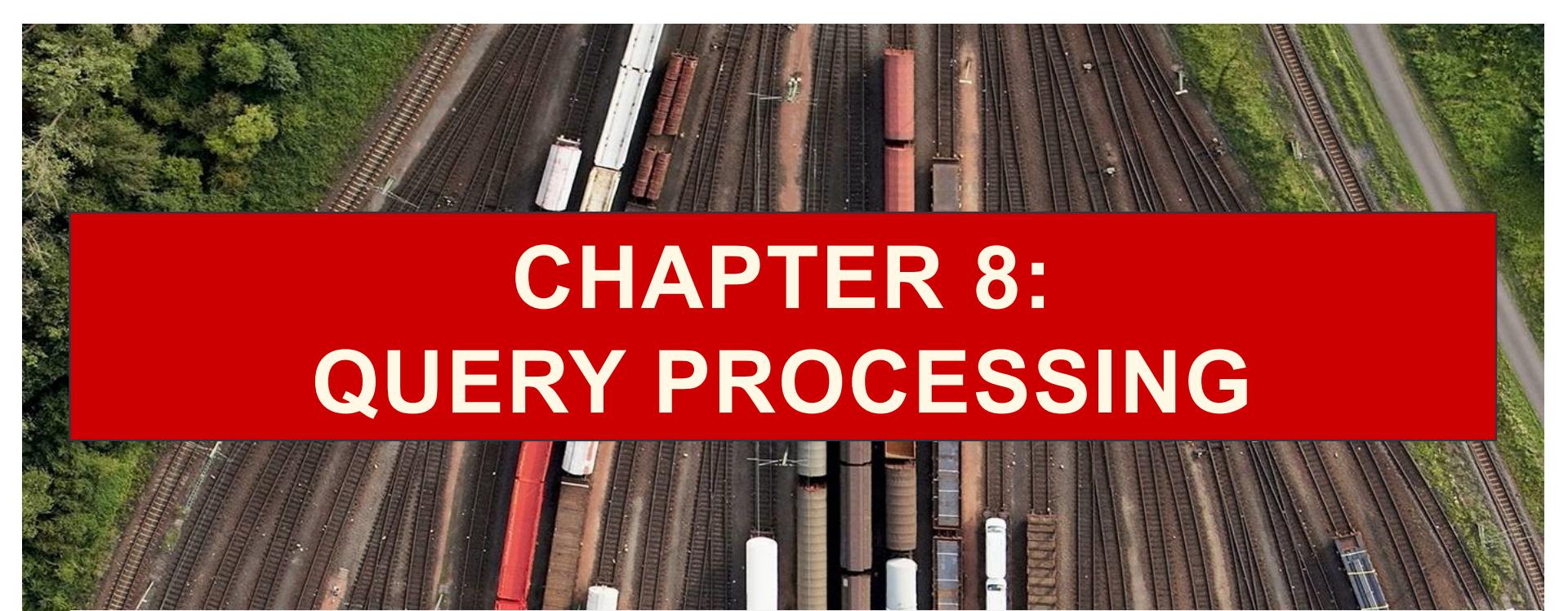
---

- Commonly done in practice
- Examples
  - hash tuples into buckets, then match each bucket in parallel
  - match tuples against a taxonomy of entities (e.g., a product or Wikipedia-like concept taxonomy) in parallel
    - ❖ two tuples are declared matched if they match into the same taxonomic node
    - ❖ a variant of using representatives to scale up, discussed earlier

# Summary

---

- Critical problem in data integration
- Huge amount of work in academia and industry
  - Rule-based matching
  - Learning- based matching
  - Matching by clustering
  - Probabilistic approaches to matching
  - Collective matching
- This chapter has covered only the most common and basic approaches
- The bibliography discusses much more



# CHAPTER 8: QUERY PROCESSING



## PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Query processing

To this point, we have seen the upper levels of the data integration problem:

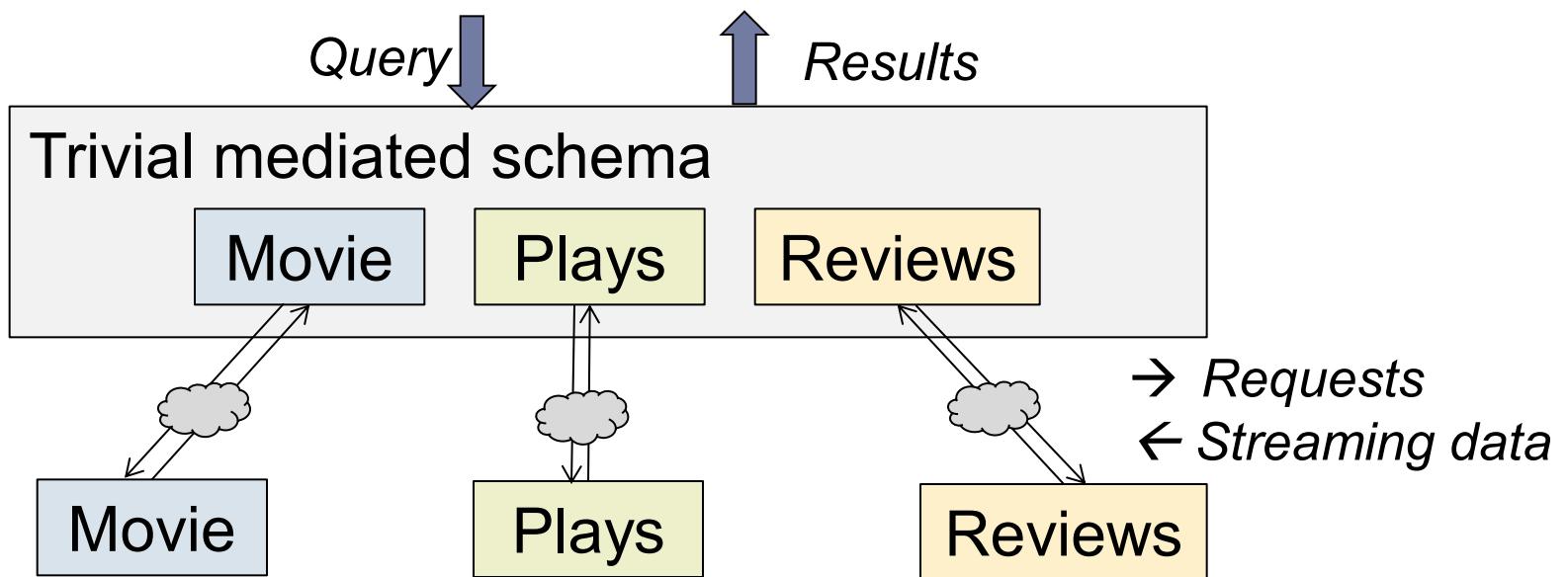
- How to construct **mappings** from **sources** to a single **mediated schema**
- How queries posed over the mediated schema are reformulated over the sources



The key question considered in this chapter: how do we **efficiently execute the reformulated query**, in distributed / streaming fashion?

# An example query over the mediated schema

```
SELECT title, startTime
FROM Movie M, Plays P, Reviews R
WHERE M.title = P.movie AND M.title = R.movie
AND P.location = "New York" AND M.actor = "Morgan
Freeman" AND R.avgStars >= 3
```



# Query processing for integration: Challenges beyond those of a DBMS

## Source query capabilities and limited data statistics

- Statistics on underlying data are often unavailable
- Many sources aren't even databases!



## Input binding restrictions

- May need to provide inputs before we can see matching data



## Variable network connectivity and source performance

- Performance is not always predictable



## Need for fast initial answers

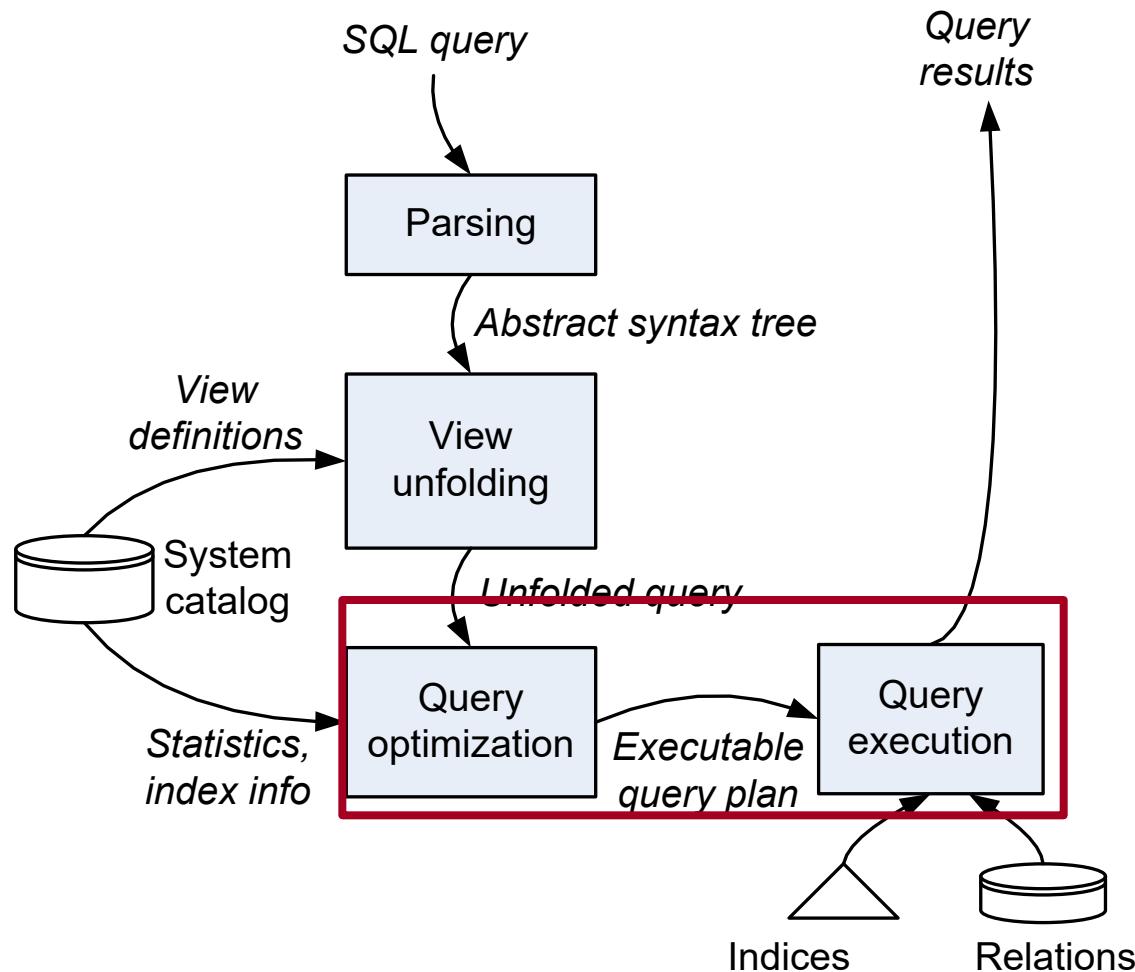
- Emphasis is often on pipelined results, not overall query completion time – harder to optimize for!

# Outline

---

- ✓ What makes query processing hard
- Review of DBMS query processing
  - Review of distributed query processing
  - Query processing for data integration

# A DBMS query processor



# Our example query in the DBMS setting

---

**SELECT** title, startTime

**FROM** Movie M, Plays P, Reviews R

**WHERE** M.title = P.movie AND M.title = R.movie

AND P.location = “New York” AND M.actor =

“Morgan Freeman” AND R.avgStars >= 3

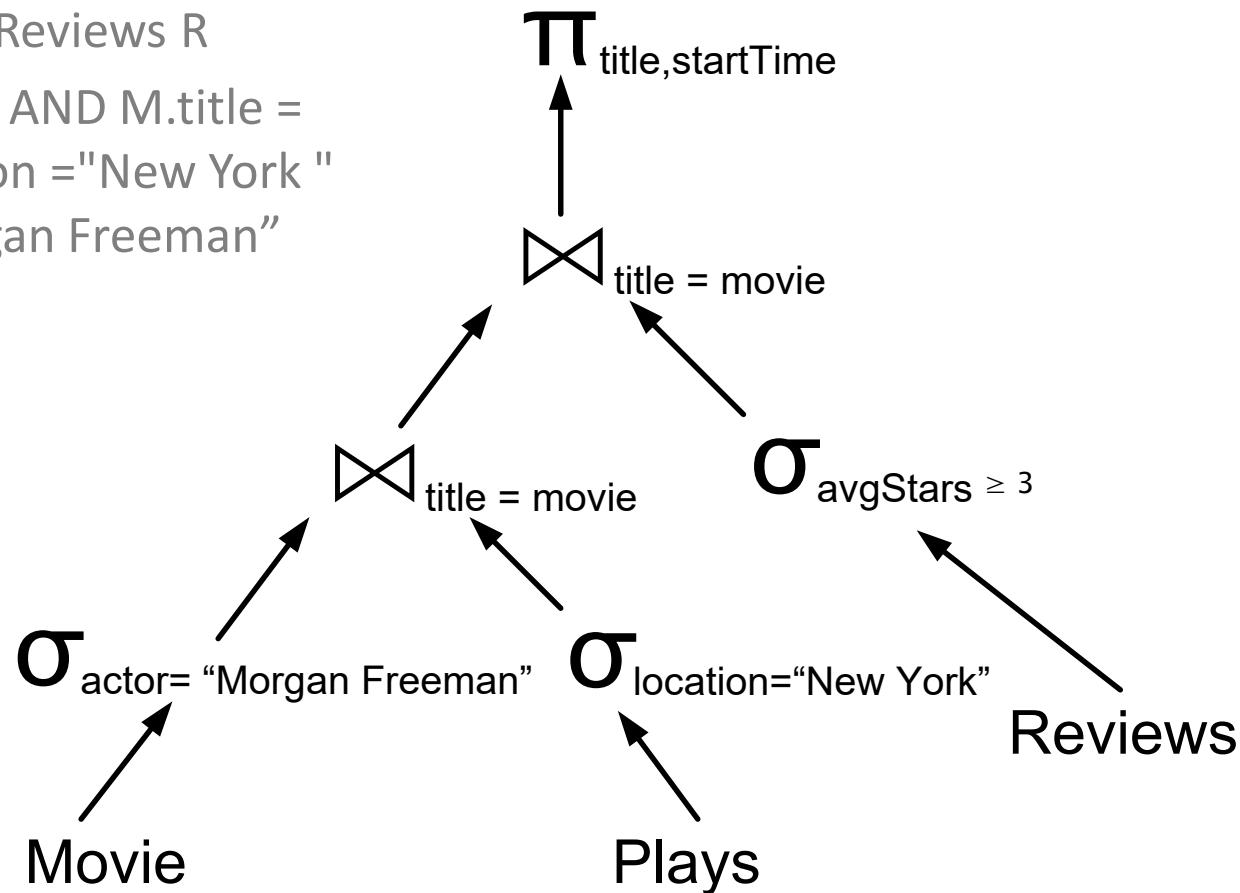
- Query is parsed, generally broken into predicates, **unfolded as necessary** (Section 2.2), then converted into a **logical query plan used by the optimizer...**

# Example logical query plan

**SELECT** title, startTime

**FROM** Movie M, Plays P, Reviews R

**WHERE** M.title = P.movie AND M.title =  
R.movie AND P.location = "New York"  
AND M.actor = "Morgan Freeman"  
AND R.avgStars >= 3



# Query optimization

---

- Goal: compare all **equivalent** query expressions and their low-level implementations (**operators**)
- Can be divided into two main aspects:
  - Plan enumeration (“search”)
  - Cost estimation

# Foundations of query plan enumeration

- Exploit properties of the relational algebra to find equivalent expressions, e.g.:

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$\sigma_\alpha(R \bowtie S) = (\sigma_\alpha(R) \bowtie S) \quad \text{if } \alpha \text{ only refers to } R$$

- Assume **selection, projection** are always done as early as possible
- Joins (generally) satisfy **principle of optimality**:

Best way to compute  $R \bowtie S \bowtie T$  takes advantage of the best way of doing a 2 way join, followed by one additional join:

$$(R \bowtie S) \bowtie T, \quad (R \bowtie T) \bowtie S, \quad \text{or } (S \bowtie T) \bowtie R$$

*(uses optimal way of computing this expression)*

# The Ordering exception

---

- Sorting (or hash-partitioning) data sometimes adds extra cost in the short term
- ... but a sort might make it more efficient to do multiple joins, thus pay off in the long term
  - e.g., by using a merge join operator or an index
- This does not follow principle of optimality!
  - Consider each ordering (including “no ordering”) to be a **separate optimization** space
  - “Interesting orders” are those that might help query processing

# Enumerating plans

Can formulate as a **dynamic programming problem**:

1. **Base case**: consider all possible ways of accessing the base tables, with all **selections & projections** pushed down
2. **Recursive case** ( $i=2 \dots \text{number of joins}+1$ ): explore all ways to join results with  $(i-1)$  tables, with one additional table
  - ❖ Common heuristic – only considers **linear plans**
3. Then repeat process for all **Cartesian products**
4. Apply **grouping** and aggregation

Alternatively: use top-down (recursion + memoization) exploration – enables pruning

# Query plan cost estimation

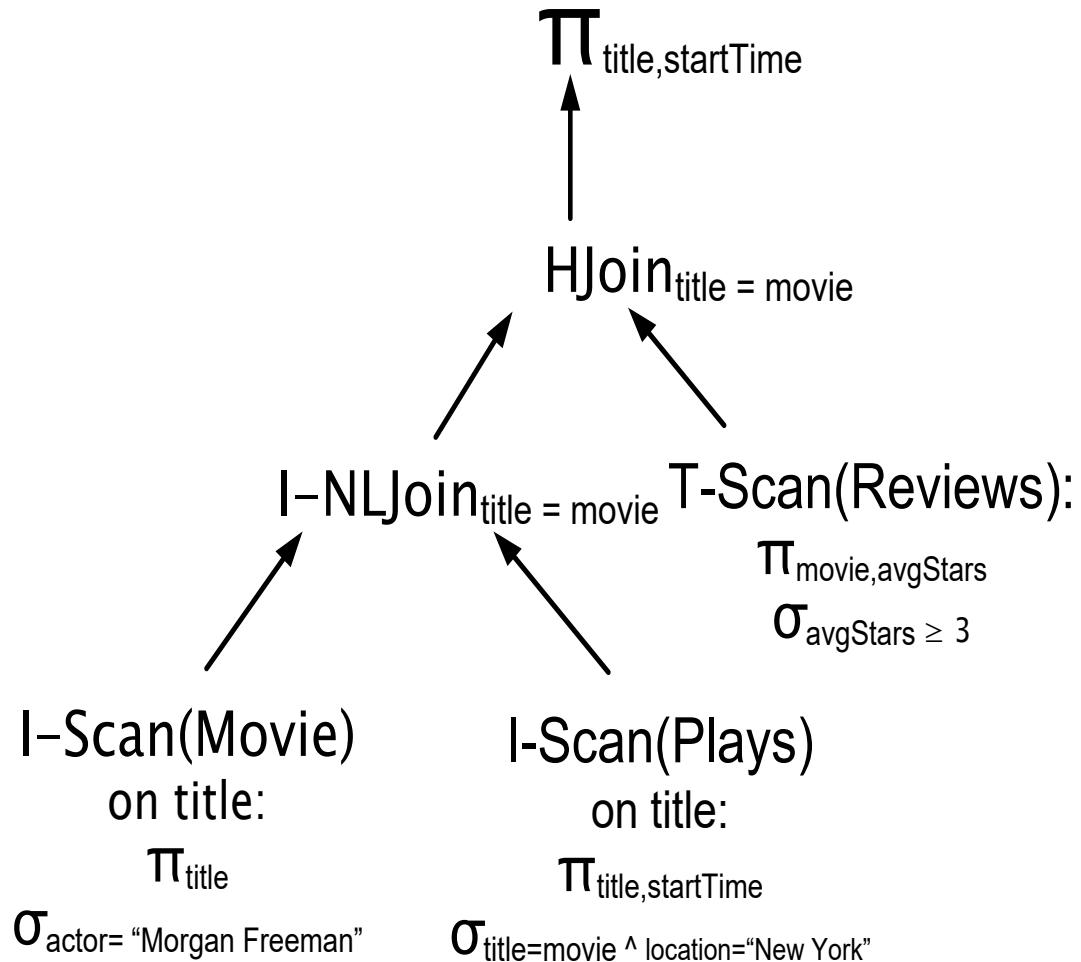
---

For each expression, predict the **cost and output size** given what we know about the inputs

Requires significant information:

- A **cost formula** for each algorithm, in terms of disk I/Os, CPU speed, ...
- **Calibration parameters** for host machine performance
- Information about the **distributions** of join and grouping columns
  - Used to estimate the selectivity of joins, number of unique values, and so on

# Example physical plan produced by an optimizer



**Hjoin:** Hash join has two phases: build (a hash table for the first relations) and probe (scan tuples from the other relation)

**T-Scan:** Table scan

**I-Scan:** Index scan

**I-NLJoin:**

# Query execution

---

The query execution engine is like a **virtual machine** that runs physical query plans

Two architectural considerations:

- Granularity of processing
- Control flow

# Granularity of processing

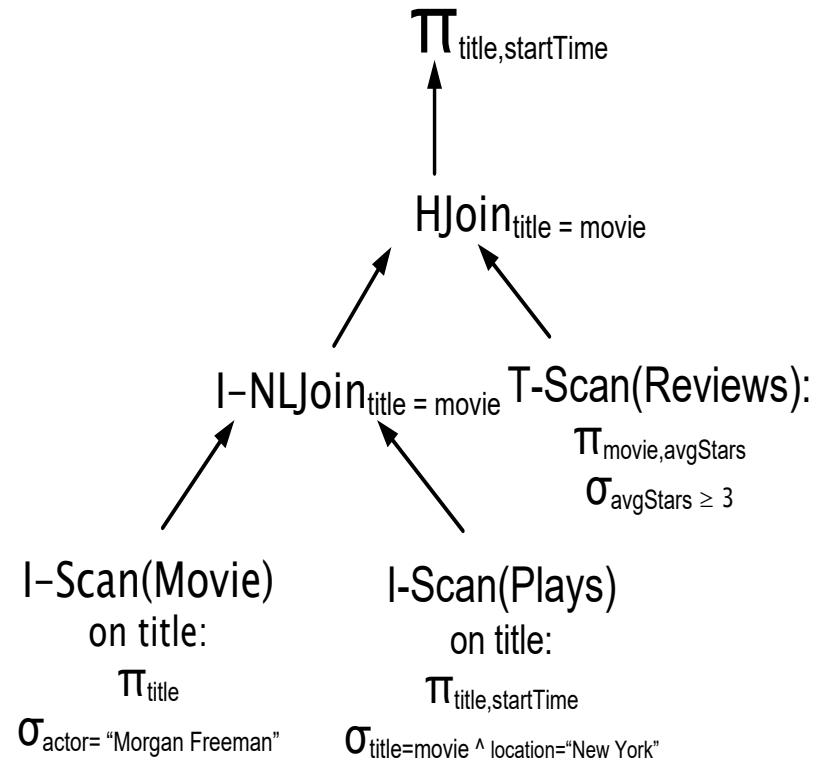
## Options for granularity:

- Operators process one relation at a time
- Operators process one tuple at a time (**pipelining**) – better responsiveness and parallelism

**Blocking** algorithms can't pipeline output, e.g., **GROUP BY** over unsorted data

Some operators are **partly blocking**:

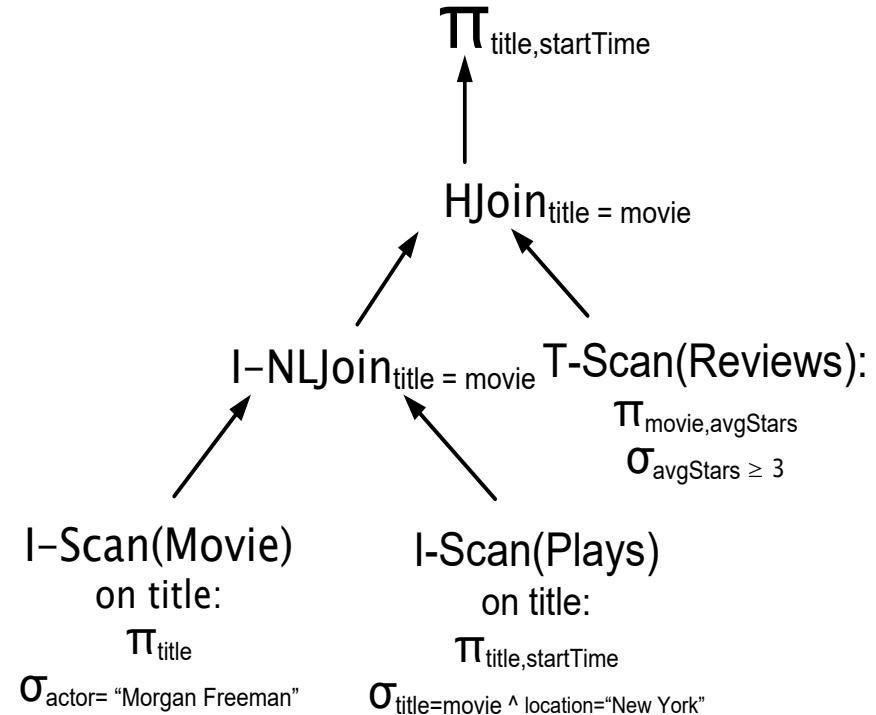
e.g., hash join (HJoin) must buffer one input before joining



# Control flow

Options for control flow:

- **Iterator-based** – execution begins at root
  - *open, next, close*
  - Propagate calls to children
  - May call multiple child *nests*
- **Dataflow-driven** – execution driven by tuple arrival at leaves
- Hybrids of the two



# Relational DBMSs set the stage

---

1. An array of algorithms for relational operations – physical plans
2. Cost-based query optimization
  - Enumeration, possibly with pruning heuristics
    - ❖ Exploit principle of optimality
    - ❖ Also search interesting orders
  - Cost estimation using statistics
3. Basic execution models
  - Iterator
  - Data-driven
  - Hybrid

# Outline

---

- ✓ What makes query processing hard
- ✓ Review of DBMS query processing
- **Review of distributed query processing**
- Query processing for data integration

# Distributed query processing

Suppose our data is distributed across multiple machines and we need to process queries

- **Parallel DBMSs** assume homogeneous nodes and fast networks (sometimes even shared memory)
    - ❖ Major goal: efficiently utilize all resources, balance load
  - **Distributed DBMSs** assume heterogeneous nodes, slower networks, some sources only available on some machines
    - ❖ Major goal: determine what computation to place where
- Our focus here is on the latter (**Distributed DBMSs**)

# Challenges of distributed query processing

- Many more options, with different costs to enumerate during database design and query optimization
- New **two-phase join** algorithms to reduce communications

Original Table

| CUSTOMER_ID | FIRST_NAME | LAST_NAME | FAVORITE_COLOR |
|-------------|------------|-----------|----------------|
| 1           | TAEKO      | OHNUKI    | BLUE           |
| 2           | O.V.       | WRIGHT    | GREEN          |
| 3           | SELDÄ      | BAĞCAN    | PURPLE         |
| 4           | JIM        | PEPPER    | AUBERGINE      |

Vertical Partitions

VP1                    VP2

| CUSTOMER_ID | FIRST_NAME | LAST_NAME |
|-------------|------------|-----------|
| 1           | TAEKO      | OHNUKI    |
| 2           | O.V.       | WRIGHT    |
| 3           | SELDÄ      | BAĞCAN    |
| 4           | JIM        | PEPPER    |

Horizontal Partitions

HP1

| CUSTOMER_ID | FIRST_NAME | LAST_NAME | FAVORITE_COLOR |
|-------------|------------|-----------|----------------|
| 1           | TAEKO      | OHNUKI    | BLUE           |
| 2           | O.V.       | WRIGHT    | GREEN          |

HP2

| CUSTOMER_ID | FIRST_NAME | LAST_NAME | FAVORITE_COLOR |
|-------------|------------|-----------|----------------|
| 3           | SELDÄ      | BAĞCAN    | PURPLE         |
| 4           | JIM        | PEPPER    | AUBERGINE      |

# The Data placement problem

Performance depends on where the data is placed

- Partition by **relations**: each machine gets some tables
- **Vertical partitioning**: each machine gets different columns from the same table
- **Horizontal partitioning**: each machine gets different rows from the same table

Also an option to **replicate** data partitions

- Trade-offs between query and update performance!

As with order in a centralized DBMS, data's **location** becomes a **property** the optimizer needs to explore

# The Data placement problem

Original Table

| CUSTOMER ID | FIRST NAME | LAST NAME | FAVORITE COLOR |
|-------------|------------|-----------|----------------|
| 1           | TAEKO      | OHNUKI    | BLUE           |
| 2           | O.V.       | WRIGHT    | GREEN          |
| 3           | SELDAA     | BAĞCAN    | PURPLE         |
| 4           | JIM        | PEPPER    | AUBERGINE      |

Vertical Partitions

VP1

| CUSTOMER ID | FIRST NAME | LAST NAME |
|-------------|------------|-----------|
| 1           | TAEKO      | OHNUKI    |
| 2           | O.V.       | WRIGHT    |
| 3           | SELDAA     | BAĞCAN    |
| 4           | JIM        | PEPPER    |

VP2

| CUSTOMER ID | FAVORITE COLOR |
|-------------|----------------|
| 1           | BLUE           |
| 2           | GREEN          |
| 3           | PURPLE         |
| 4           | AUBERGINE      |

Horizontal Partitions

HP1

| CUSTOMER ID | FIRST NAME | LAST NAME | FAVORITE COLOR |
|-------------|------------|-----------|----------------|
| 1           | TAEKO      | OHNUKI    | BLUE           |
| 2           | O.V.       | WRIGHT    | GREEN          |

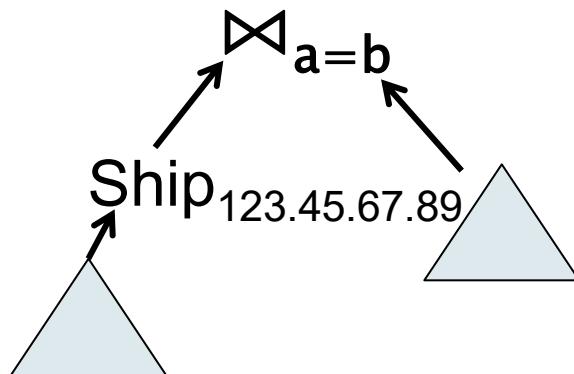
HP2

| CUSTOMER ID | FIRST NAME | LAST NAME | FAVORITE COLOR |
|-------------|------------|-----------|----------------|
| 3           | SELDAA     | BAĞCAN    | PURPLE         |
| 4           | JIM        | PEPPER    | AUBERGINE      |

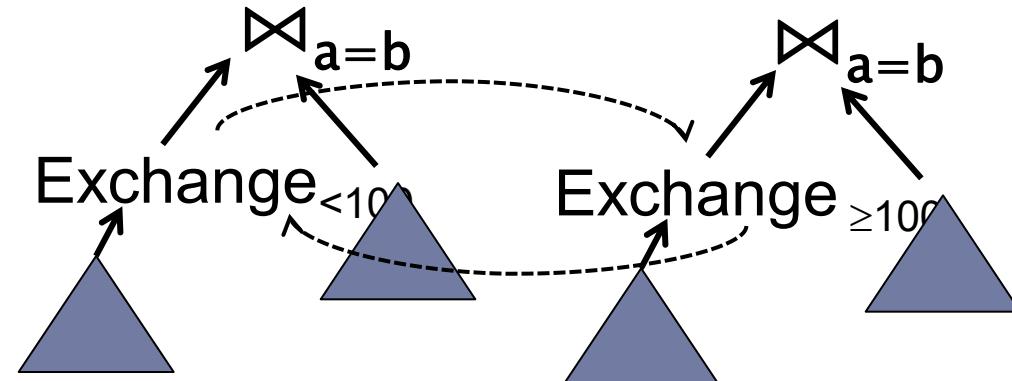
# New query operators for distributed databases

## 1. Operators to transfer information across nodes

**ship** routes a stream of tuples from one machine to another, named machine



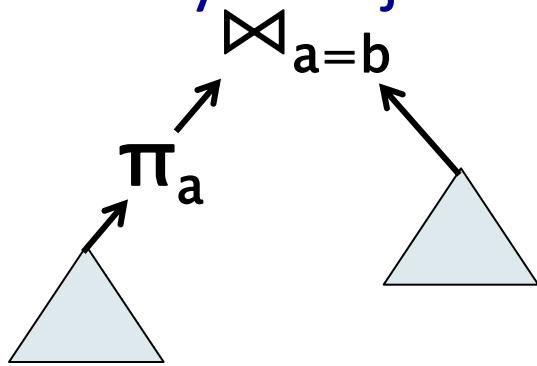
**rehash** or **exchange** swaps tuples among machines running the same plan: each machine gets tuples within some range of keys



# Joining in two phases

## 2. Operators that perform distributed joins...

### Two-way semijoin

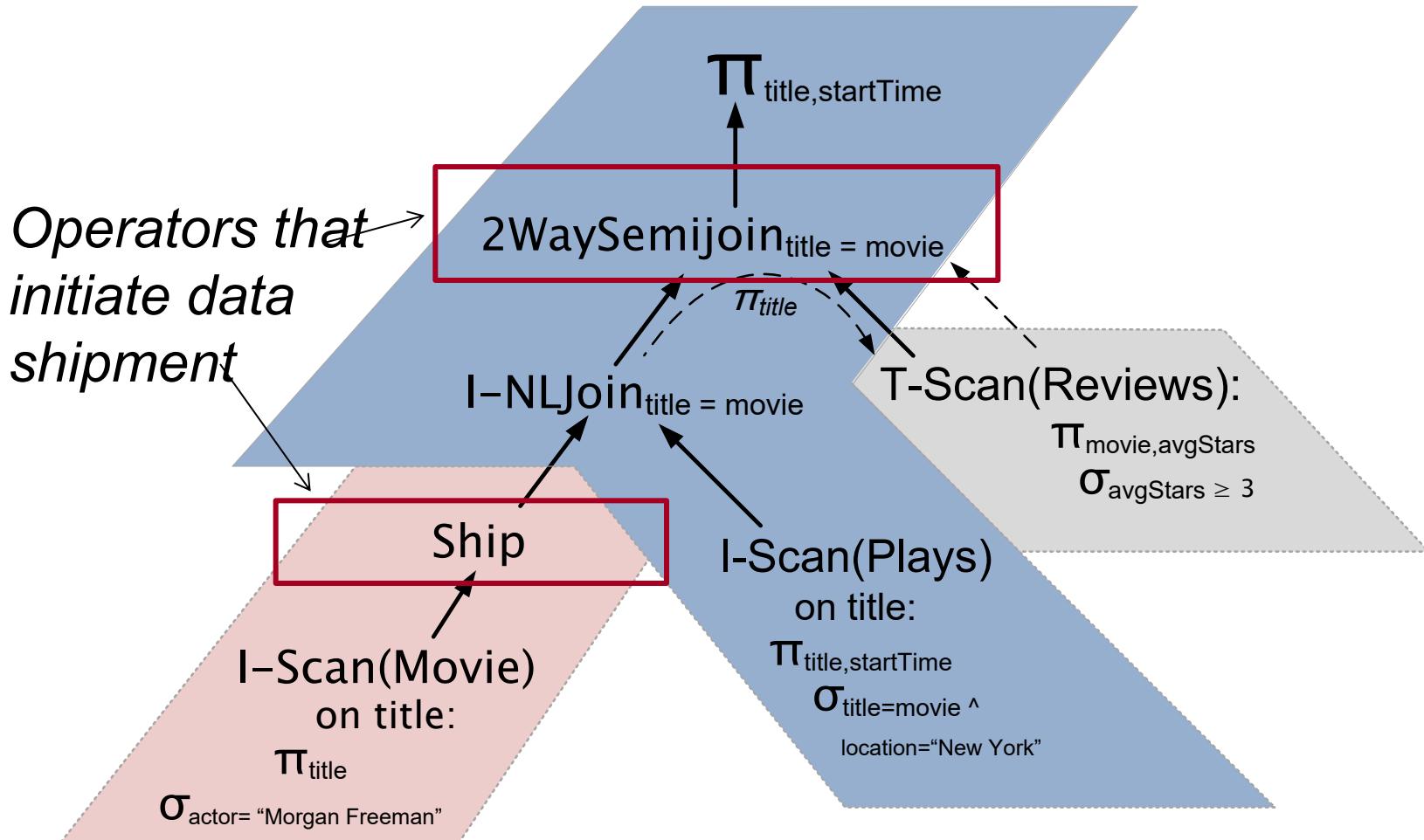


- Take R, project join key attributes
- Ship to S
- Fetch all matching S tuples
- Ship to destination
- Join R tuples with S tuples

### Bloomjoin

- Take R, build *Bloom filter*
  - Build a large bit vector
  - Take each value of R.a, hash with multiple functions  $h_i(a)$
  - Set bits for each  $h_i(a)$
- Ship to S
- Fetch all matching S tuples
  - For each S.b, ensure a match to each  $h_i(b)$  in Bloom filter
- Ship to destination
- Join R tuples with S tuples

# An example distributed query plan (Shades indicate machines)



# Distributed DBMSs Generalize DB Techniques to Handle Communication

---

Distributed databases contributes three basic sets of techniques:

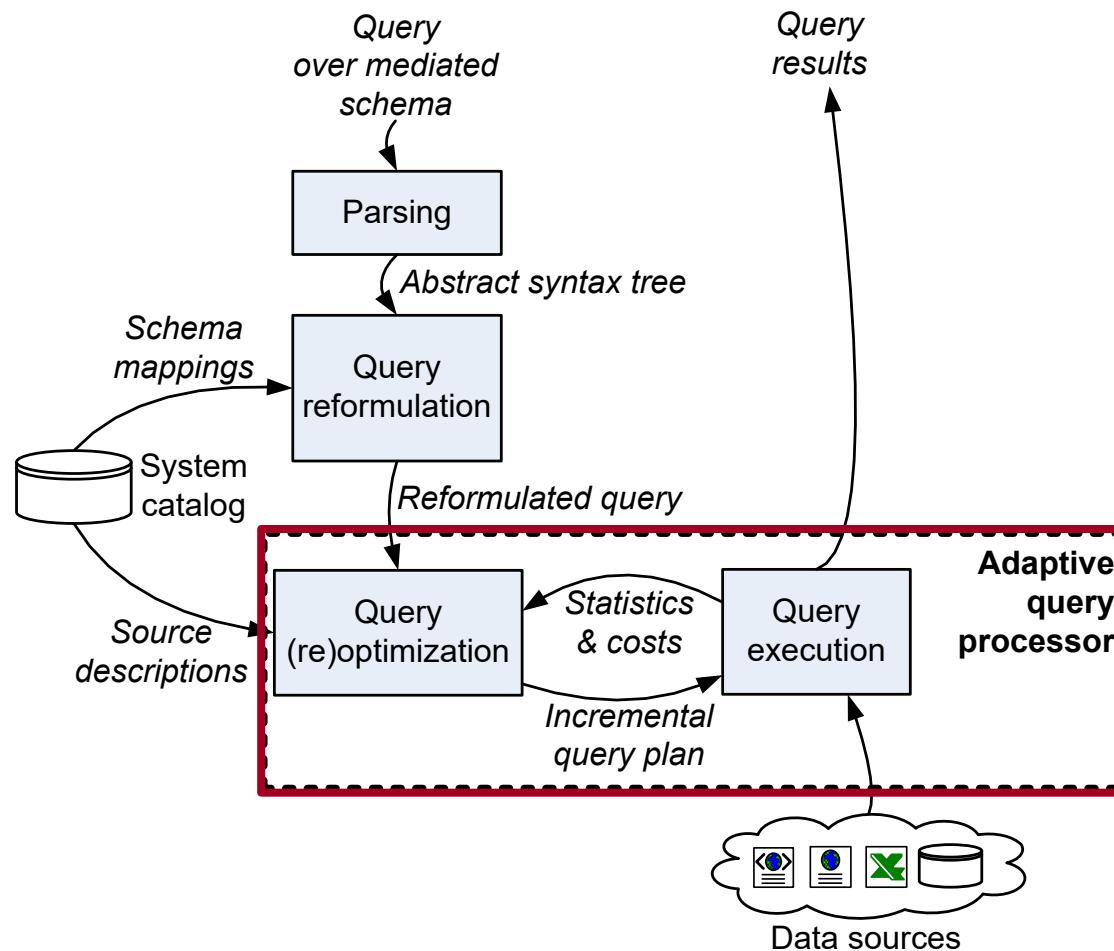
1. Different **data partitioning** methods and their impact on performance
2. Incorporating **data location** into cost-based optimization
3. New operators for shipping data and for multi-stage joins

# Outline

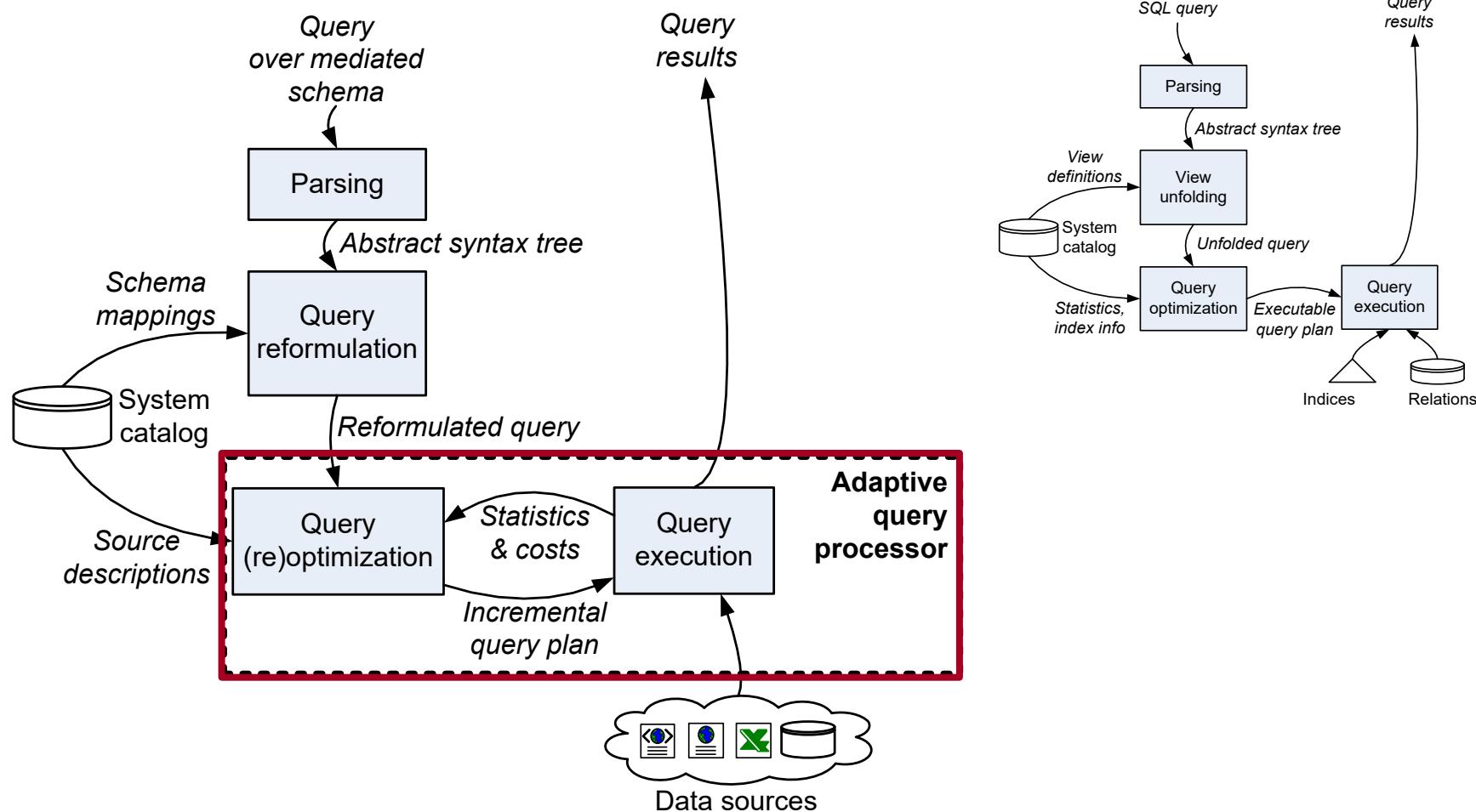
---

- ✓ What makes query processing hard
- ✓ Review of DBMS query processing
- ✓ Review of distributed query processing
- **Query processing for data integration**
  - Generating initial plans
  - Query execution for Internet data
  - Adaptive query processing overview
  - Event-driven adaptivity
  - Performance-driven adaptivity

# Query Processing in Data Integration



# Query Processing in Data Integration



# Query Processing for Data Integration

Suppose we want to do distributed query processing where:

- We have no indices and no statistics
- Data is all remote, and may be restricted in how it can be accessed
- Results must be presented as soon as available



Such challenges are encountered in data integration

- The query processor is given a reformulated query and little else to work with
- Existing optimization techniques don't work!
- We need **adaptive** and **fully pipelined** techniques...

# Query Processing for Data Integration

---

We'll discuss in several steps:

1. How to generate initial query plans
2. Query execution for integration
3. Adaptive query processing
  - ❖ Event-driven adaptivity
  - ❖ Performance-driven adaptivity

# Initial Plan Generation

---

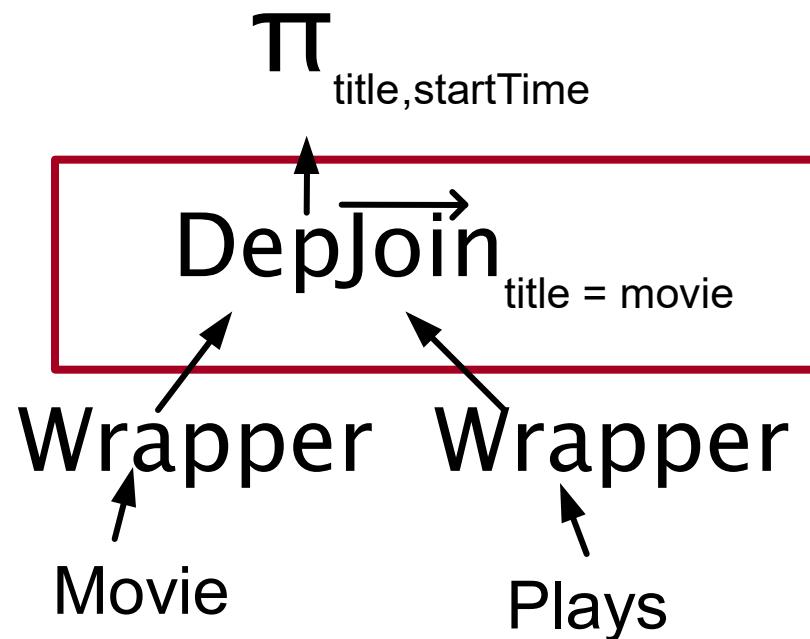
- Generally very much like query optimization for distributed databases, except that we may be more reliant on heuristics:
  - Use rough profiles of source performance and bandwidth
  - Use conservative estimates (i.e., over-estimates) of intermediate result sizes
- A few extra introduced by **wrappers** to access sources (See Chapter 9)
  - Access-pattern restrictions
  - Source query capabilities

# Wrinkle 1: Access-Pattern Restrictions

- Assume every source has a **wrapper**
  - Some sources require certain access patterns  
(Section 3.3) or input bindings
    - ❖ we represent with adornments, e.g., CitationDB<sup>bf</sup>(X,Y)
- The optimizer must find a plan that joins CitationDB's X attribute with another relation expression
  - ... for which we don't have binding restrictions!
  - Requires a **dependent join** operation, which is implemented like the **2-way semijoin** but feeds data to the wrapper

Search CIS

# Plan with a Dependent Join



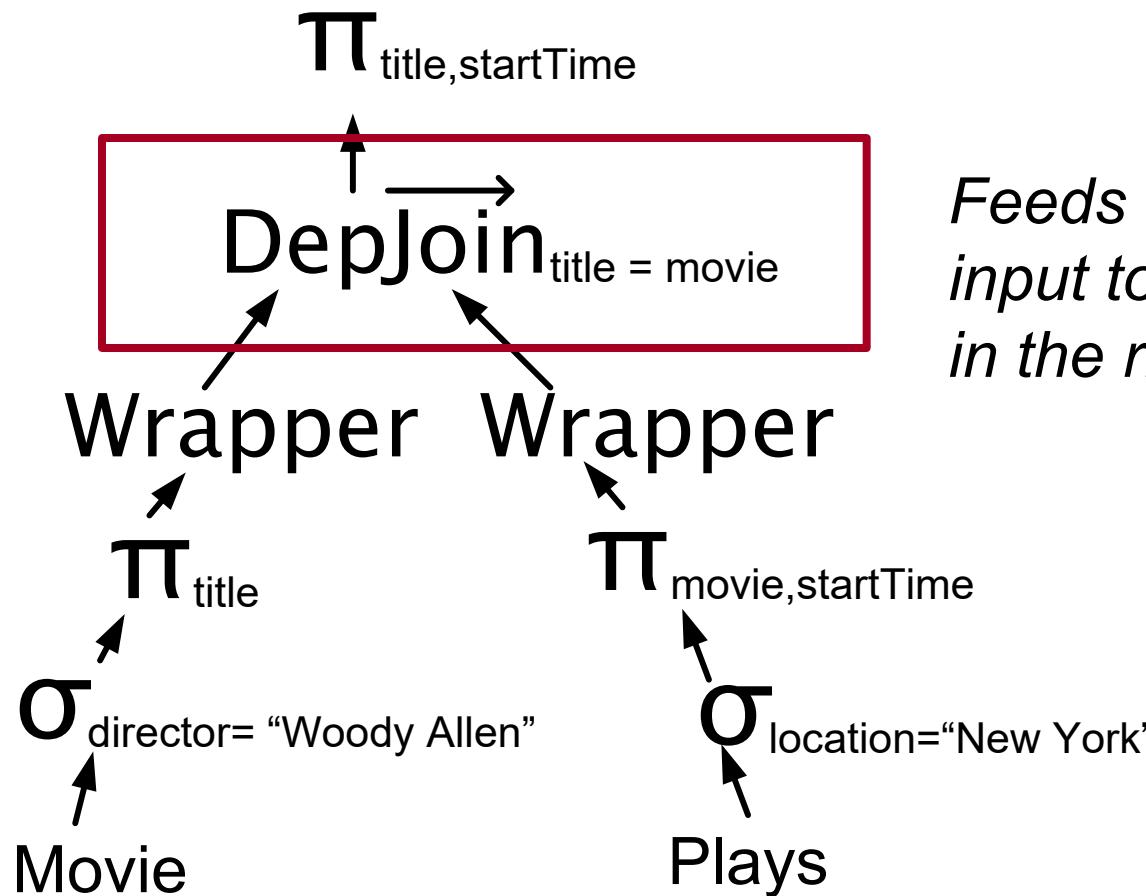
*Feeds data from left input to the wrapper in the right input*

# Wrinkle 2: Source Query Capabilities

---

- Some sources can execute certain operations (selection, project, sometimes join or more)
  - e.g., an SQL database as a source
  - ... or even a Web form that applies filter predicates
- The data integration system optimizer must be able to estimate the cost of pushing operations into the source
  - ... which in turn may vary depending on the source's load, which we may not know...

# Plan with a Dependent Join

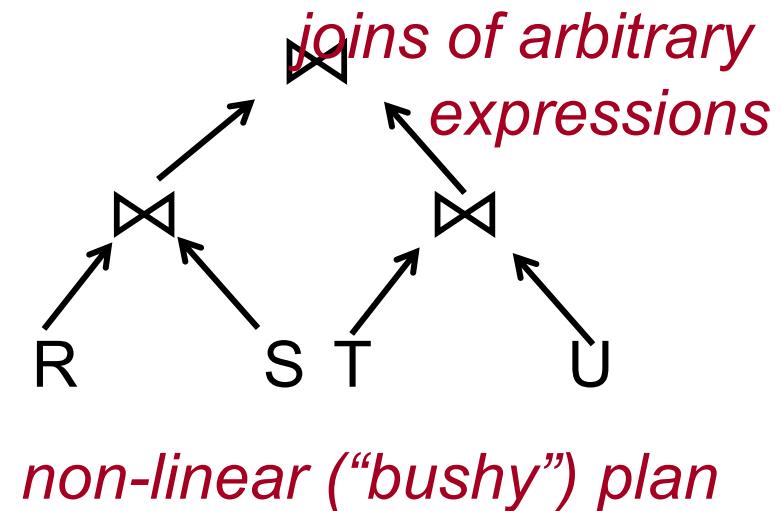
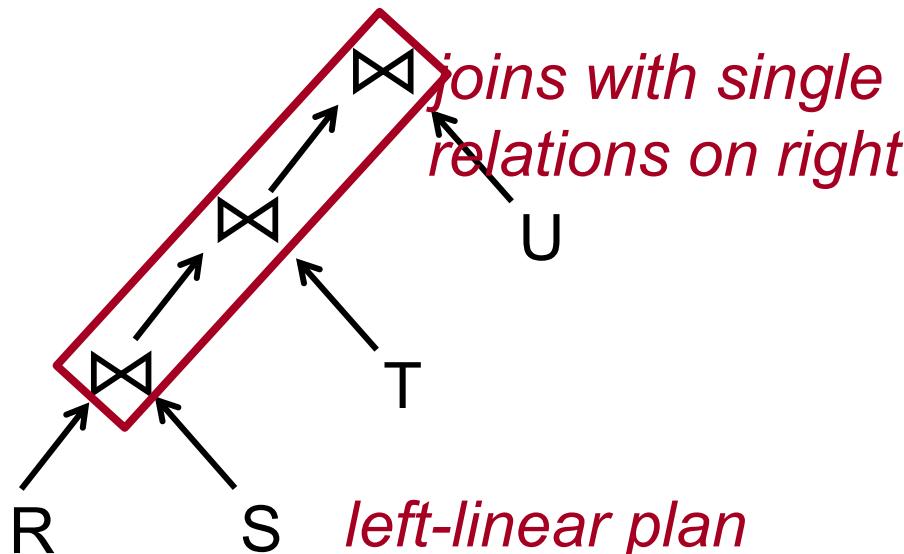


*Feeds data from left input to the wrapper in the right input*

# The Basic Approach

Enumerate all possible query expressions

- “bushy” plan enumeration instead of “left-linear” as was illustrated for relational DBMS optimization



Use a different set of cost formulas for pushed-down operations, custom to the wrapper and the source

# Changes to Query Execution

---

- Internet-based sources present challenges:
  - Slow and unpredictable data transfer rates
  - Source failures
- Many data integration applications also emphasize early results!
- Emphasize maximally pipelined query processing + failure handling...

# Threads to the Rescue

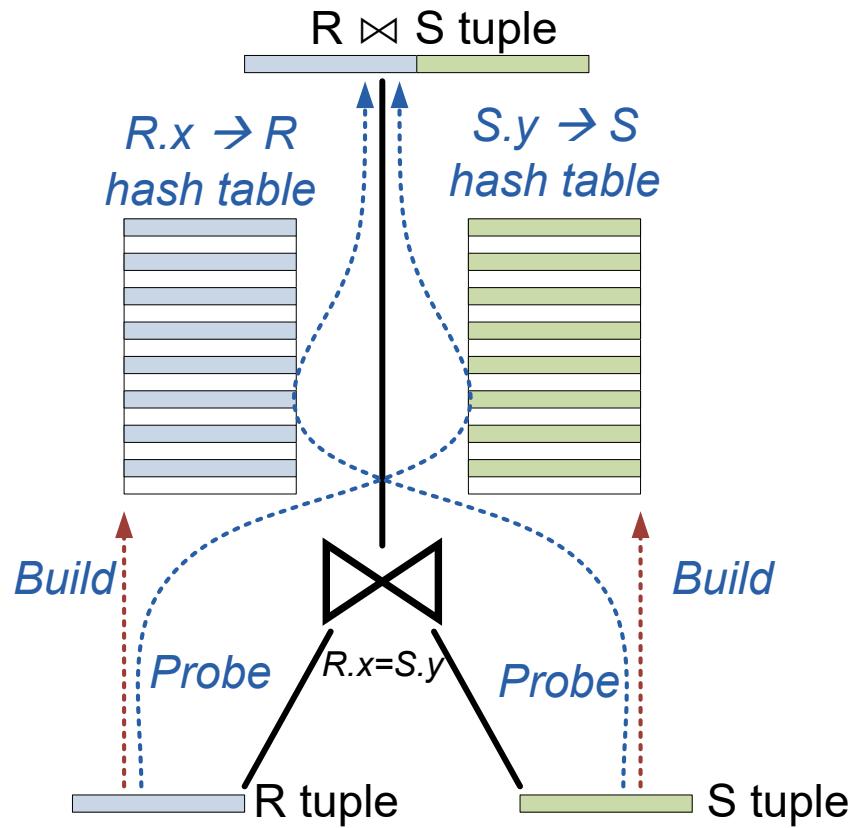
---

- Different sources will send data at different rates – an architecture based on iterators may hold up the CPU when it's waiting for I/O
  - e.g., we are trying to read one table but no data is yet ready
- Develop fully pipelined join operators with multiple threads (one to process each input, one for the computation)...
  - An instance of the **pipelined hash join**

# Pipelined Hash Join (aka Symmetric Hash Join)

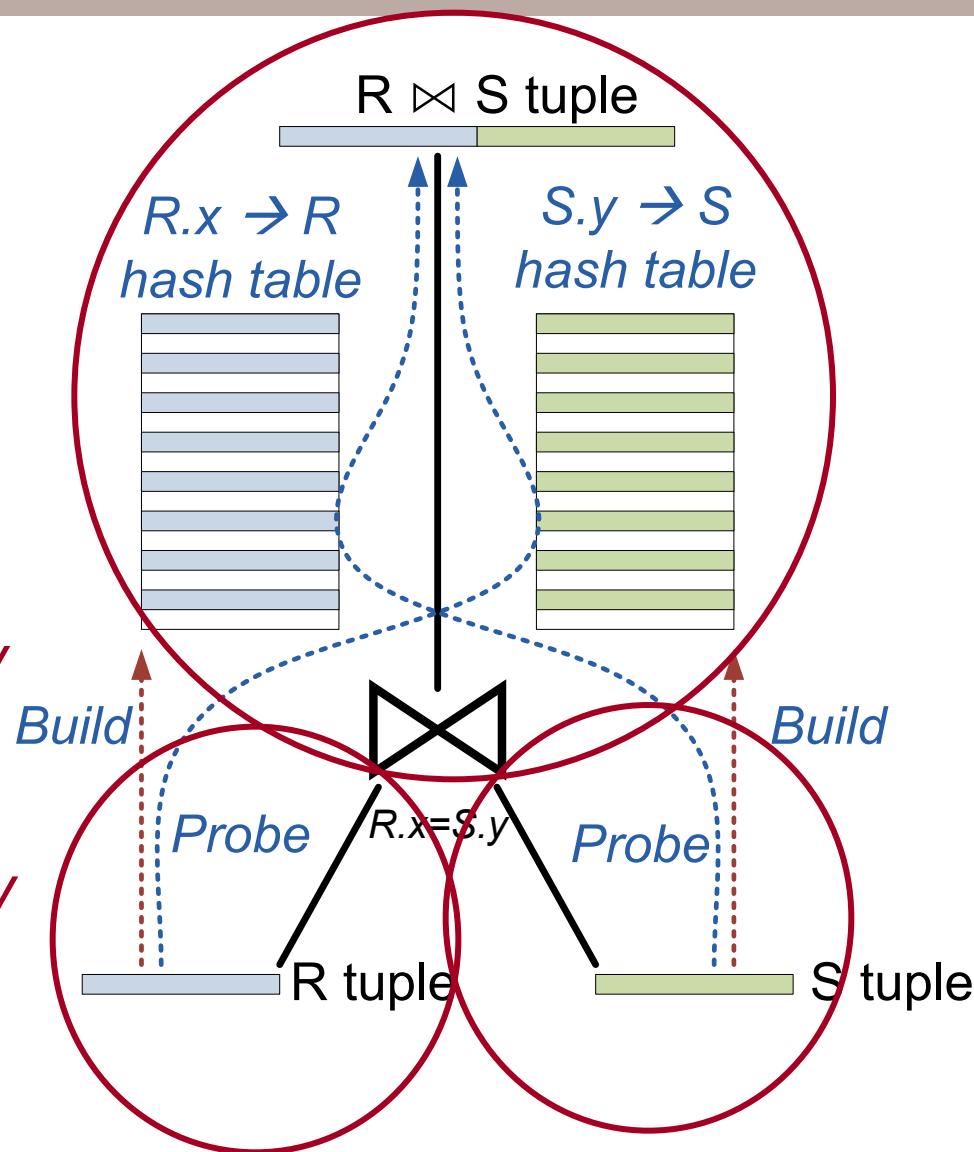
Operates symmetrically,  
and is fully pipelined:

1. Read from either input
2. Add tuple to input's hash table
3. Probe against opposite hash table
4. Return resulting output

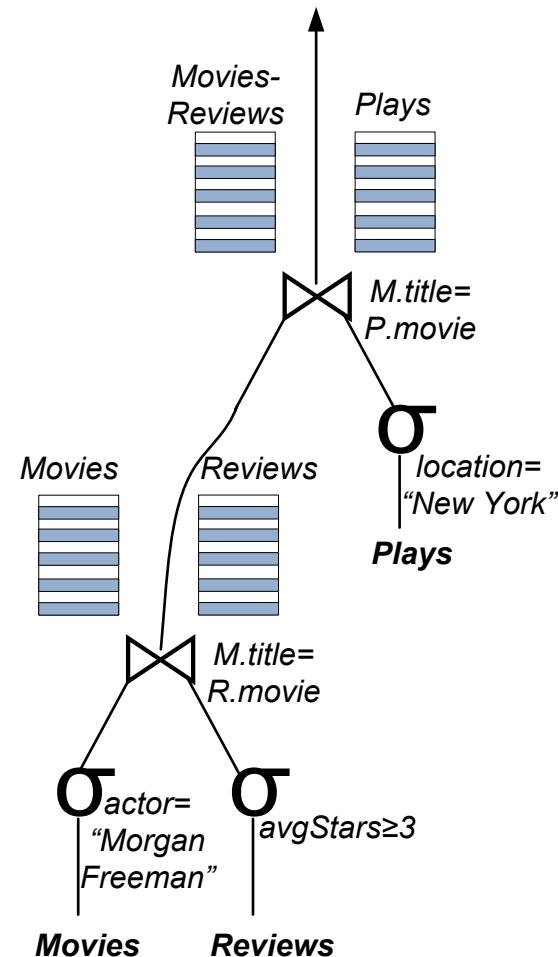


# Pipelined Hash Join

*Each red ellipse represents a separate thread of control, scheduled by the CPU based on availability*



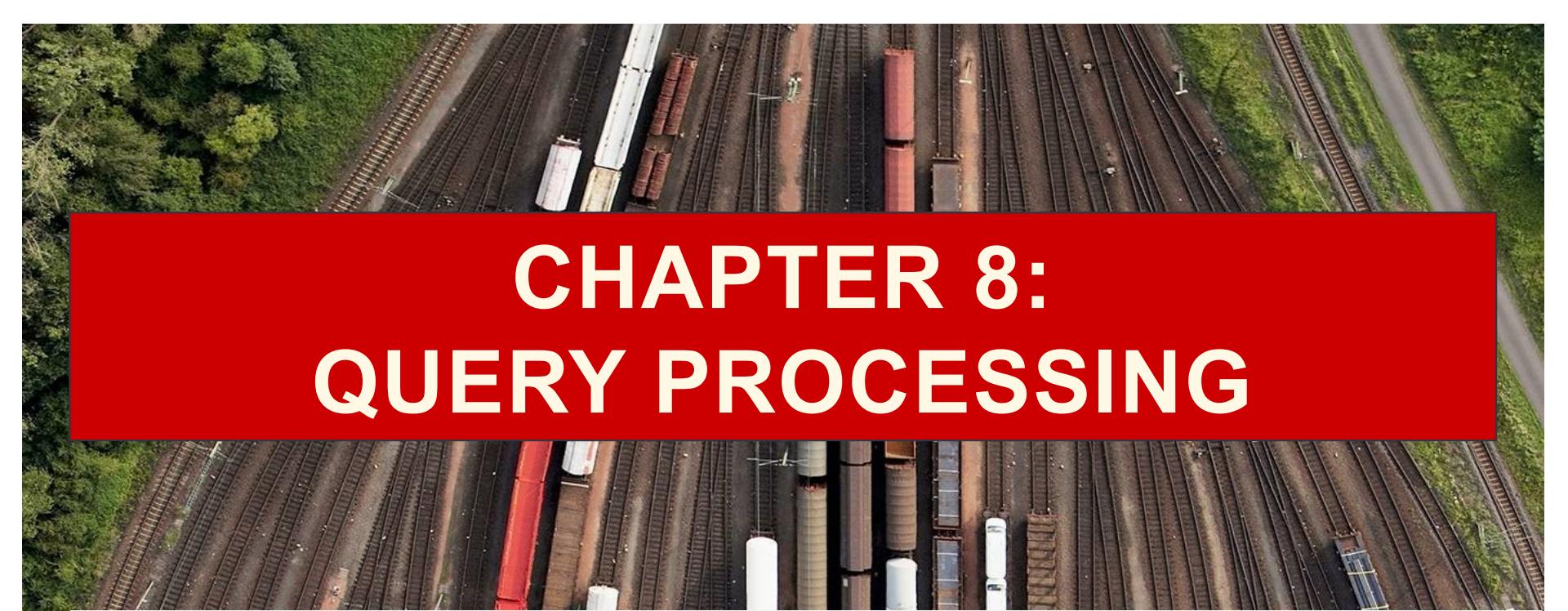
# Pipelined Join Plan



# Summary

---

- Query processing for data integration builds upon past models
  - From centralized DBMSs: plan enumeration, cost modeling, basic query execution architectures
  - From distributed DBMSs: modeling communication cost, data distribution algorithms
- But a new emphasis on:
  - Wrappers, access patterns, and query subexpression push-down
  - Fully pipelined algorithms, multithreaded operation
  - Adaptivity – event-based and performance-based
- Also, we may encounter XML (see Chapter 11)



# CHAPTER 8: QUERY PROCESSING



## PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Query processing

To this point, we have seen the upper levels of the data integration problem:

- How to construct **mappings** from **sources** to a single **mediated schema**
- How queries posed over the mediated schema are reformulated over the sources



The key question considered in this chapter: how do we **efficiently execute the reformulated query**, in distributed / streaming fashion?

# Query processing for integration: Challenges beyond those of a DBMS

## Source query capabilities and limited data statistics

- Statistics on underlying data are often unavailable
- Many sources aren't even databases!



## Input binding restrictions

- May need to provide inputs before we can see matching data



## Variable network connectivity and source performance

- Performance is not always predictable



## Need for fast initial answers

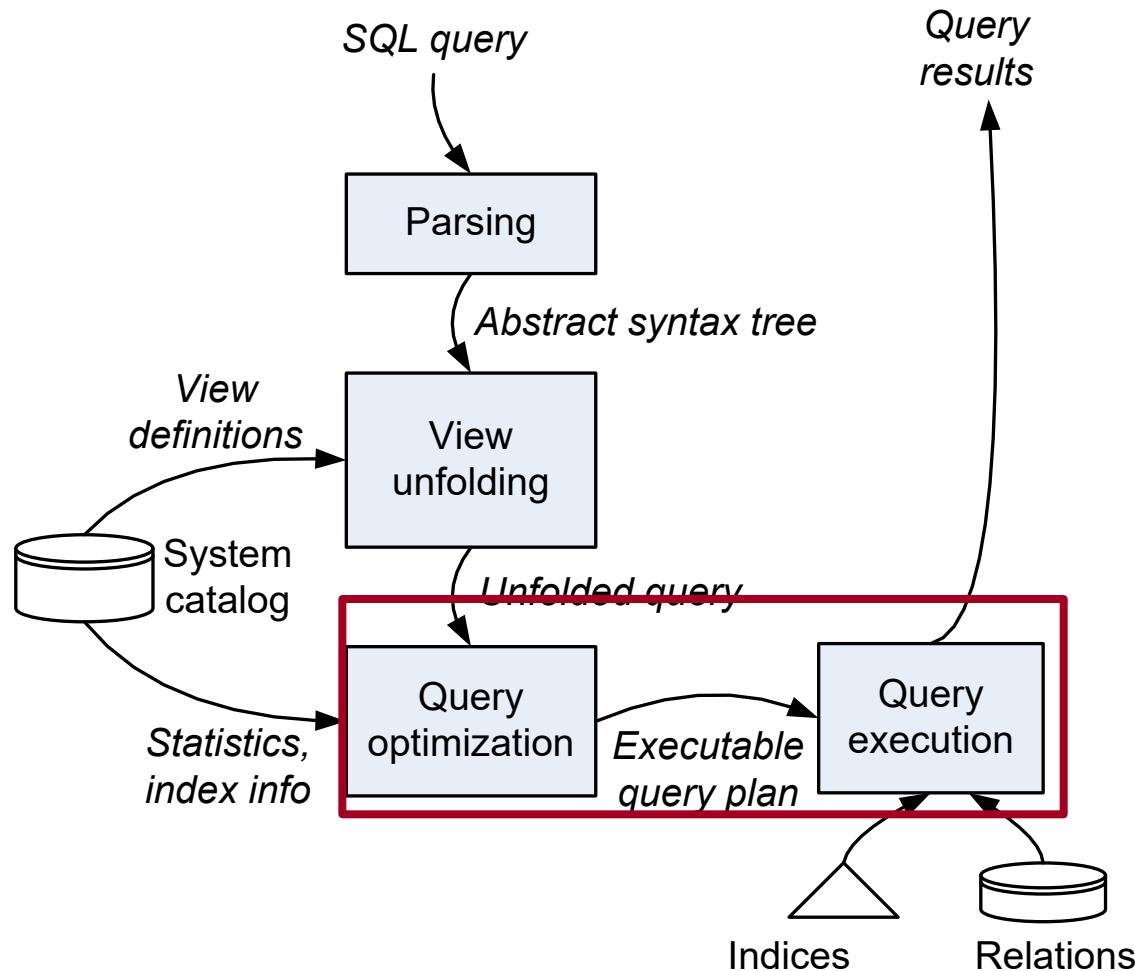
- Emphasis is often on pipelined results, not overall query completion time – harder to optimize for!

# Outline

---

- ✓ What makes query processing hard
- Review of DBMS query processing
  - Review of distributed query processing
  - Query processing for data integration

# A DBMS query processor

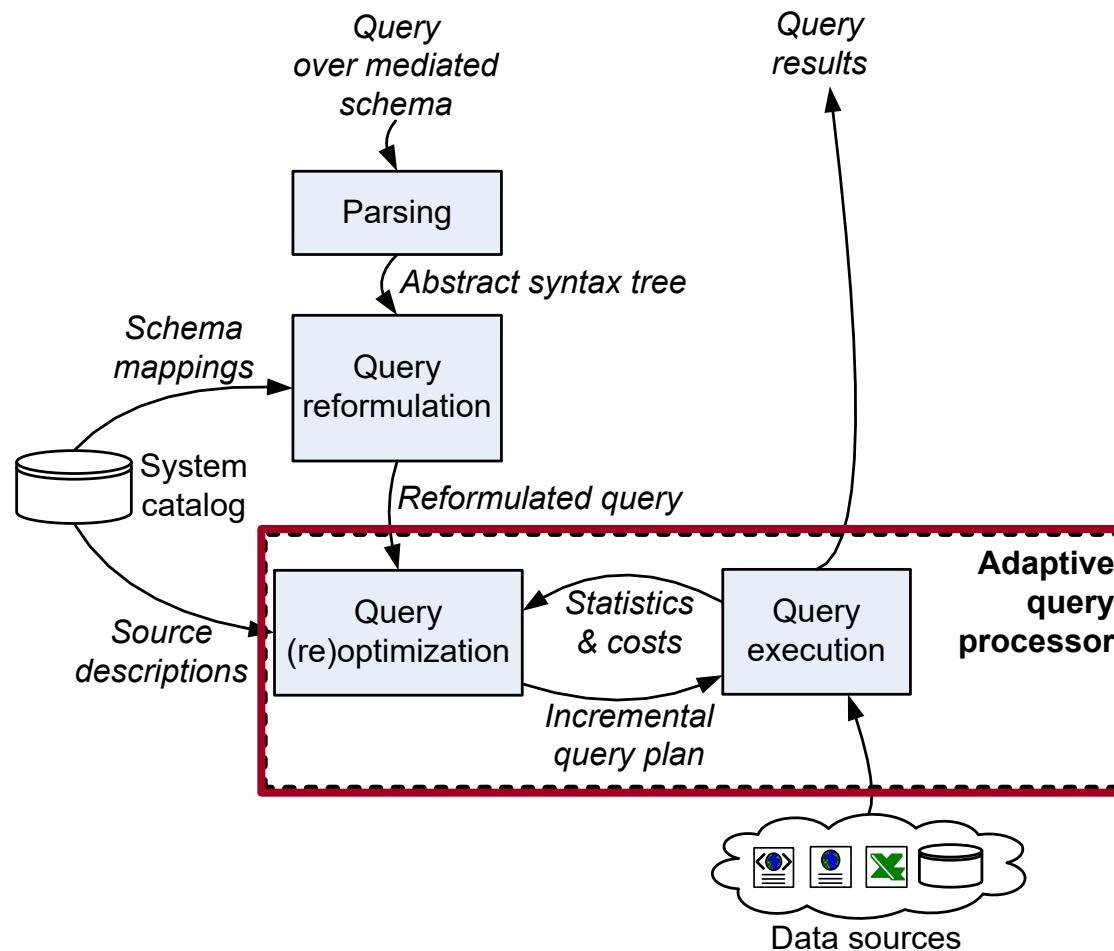


# Outline

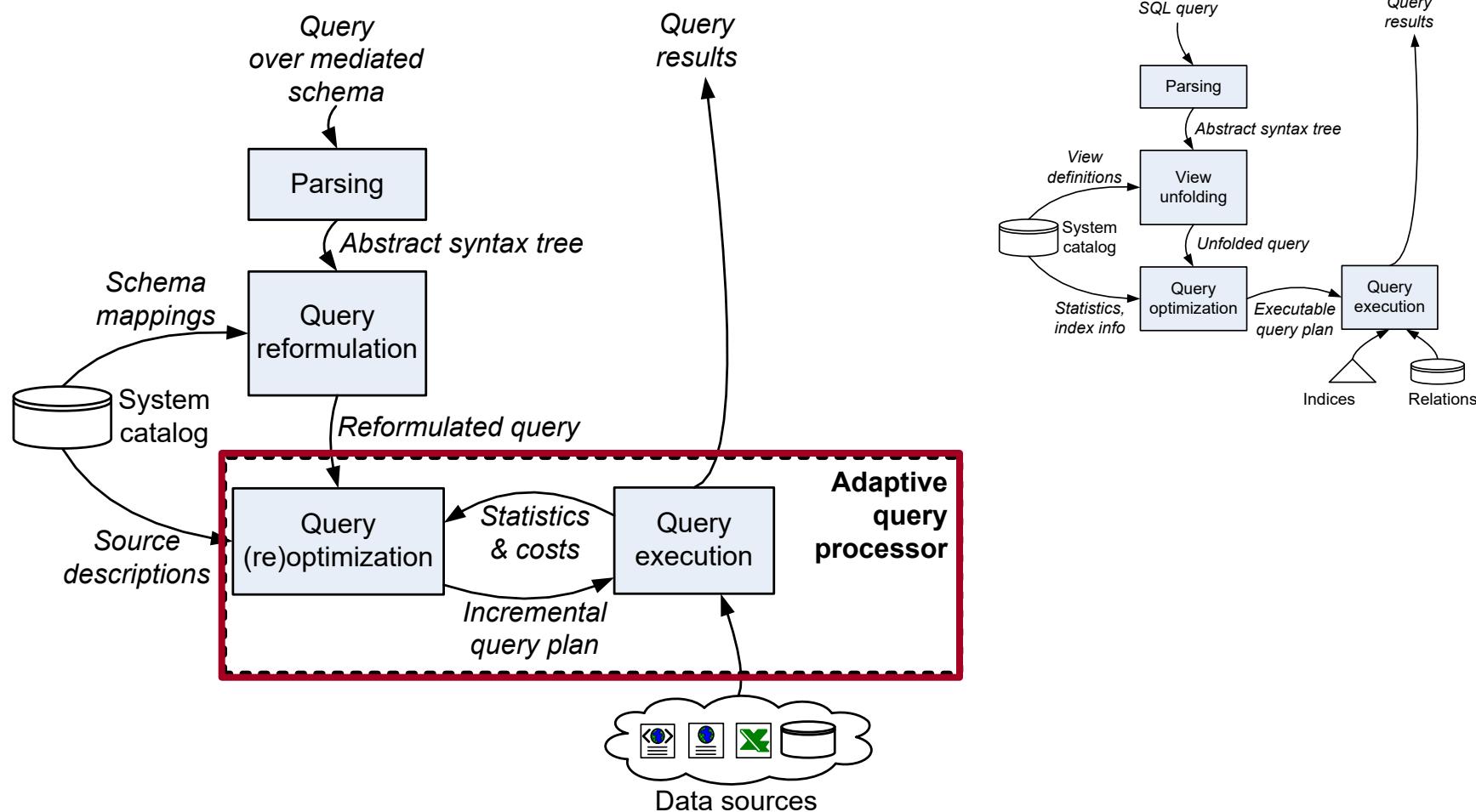
---

- ✓ What makes query processing hard
- ✓ Review of DBMS query processing
- ✓ Review of distributed query processing
- **Query processing for data integration**
  - Generating initial plans
  - Query execution for Internet data
  - Adaptive query processing overview
  - Event-driven adaptivity
  - Performance-driven adaptivity

# Query Processing in Data Integration



# Query Processing in Data Integration



# Query Processing for Data Integration

Suppose we want to do distributed query processing where:

- We have no indices and no statistics
- Data is all remote, and may be restricted in how it can be accessed
- Results must be presented as soon as available



Such challenges are encountered in data integration

- The query processor is given a reformulated query and little else to work with
- Existing optimization techniques don't work!
- We need **adaptive** and **fully pipelined** techniques...

# Query Processing for Data Integration

---

We'll discuss in several steps:

1. How to generate initial query plans
2. Query execution for integration
3. Adaptive query processing
  - ❖ Event-driven adaptivity
  - ❖ Performance-driven adaptivity

# Initial Plan Generation

---

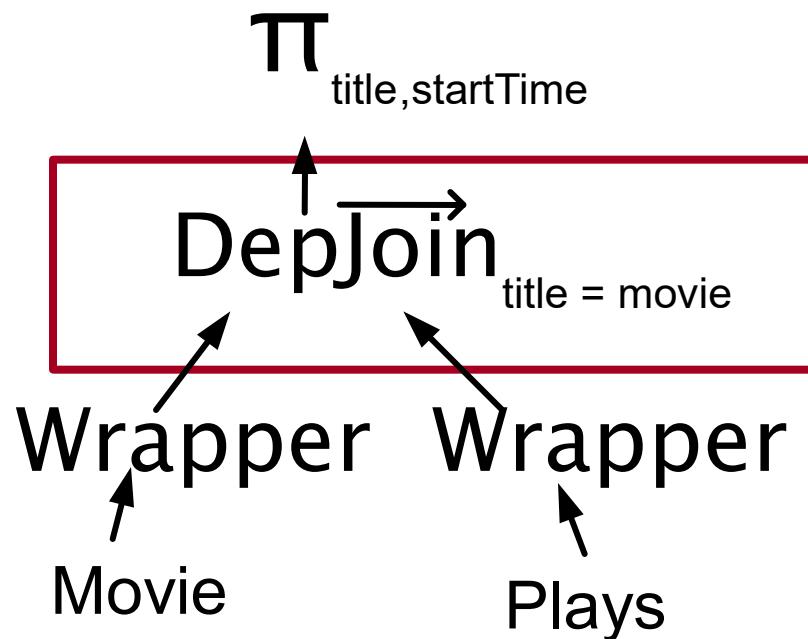
- Generally very much like query optimization for distributed databases, except that we may be more reliant on heuristics:
  - Use rough profiles of source performance and bandwidth
  - Use conservative estimates (i.e., over-estimates) of intermediate result sizes
- A few extra introduced by **wrappers** to access sources (See Chapter 9)
  - Access-pattern restrictions
  - Source query capabilities

# Wrinkle 1: Access-Pattern Restrictions

- Assume every source has a **wrapper**
  - Some sources require certain access patterns  
(Section 3.3) or input bindings
    - ❖ we represent with adornments, e.g., CitationDB<sup>bf</sup>(X,Y)
- The optimizer must find a plan that joins CitationDB's X attribute with another relation expression
  - ... for which we don't have binding restrictions!
  - Requires a **dependent join** operation, which is implemented like the **2-way semijoin** but feeds data to the wrapper

Search CIS

# Plan with a Dependent Join



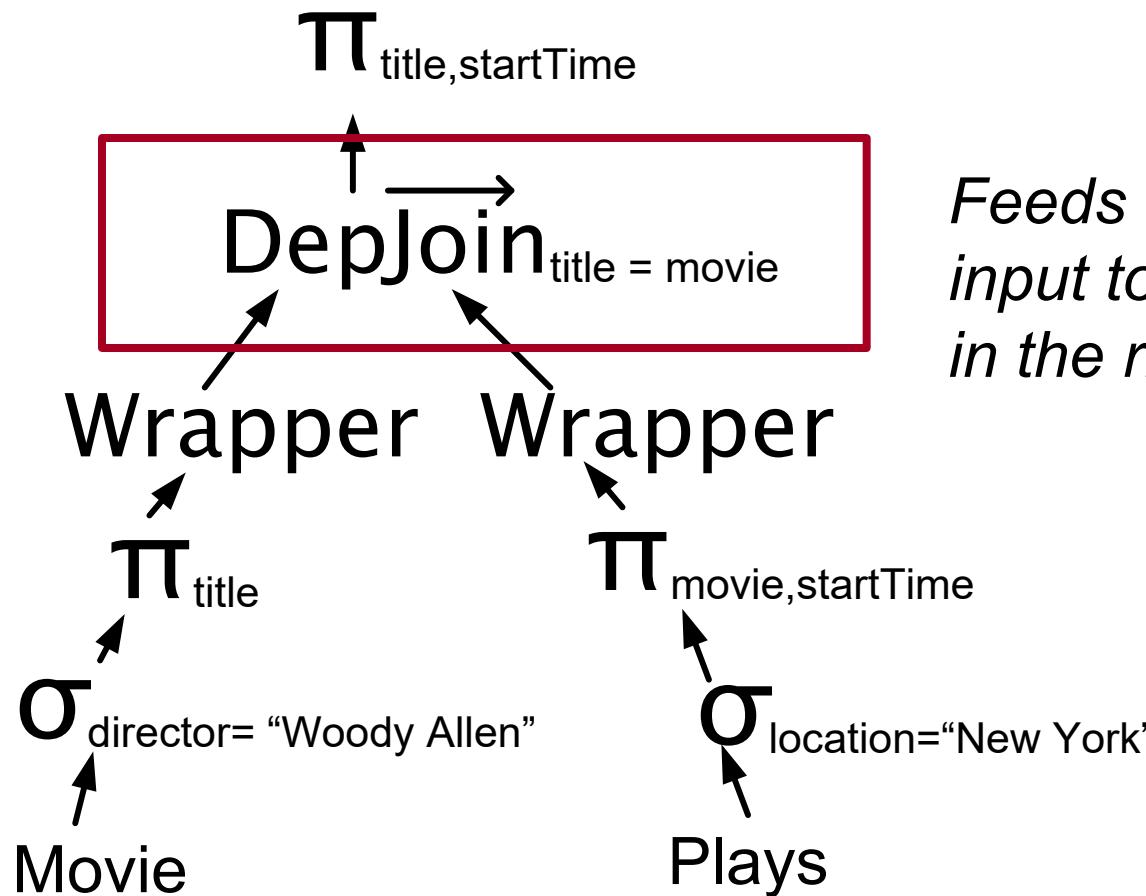
*Feeds data from left input to the wrapper in the right input*

# Wrinkle 2: Source Query Capabilities

---

- Some sources can execute certain operations (selection, project, sometimes join or more)
  - e.g., an SQL database as a source
  - ... or even a Web form that applies filter predicates
- The data integration system optimizer must be able to estimate the cost of pushing operations into the source
  - ... which in turn may vary depending on the source's load, which we may not know...

# Plan with a Dependent Join

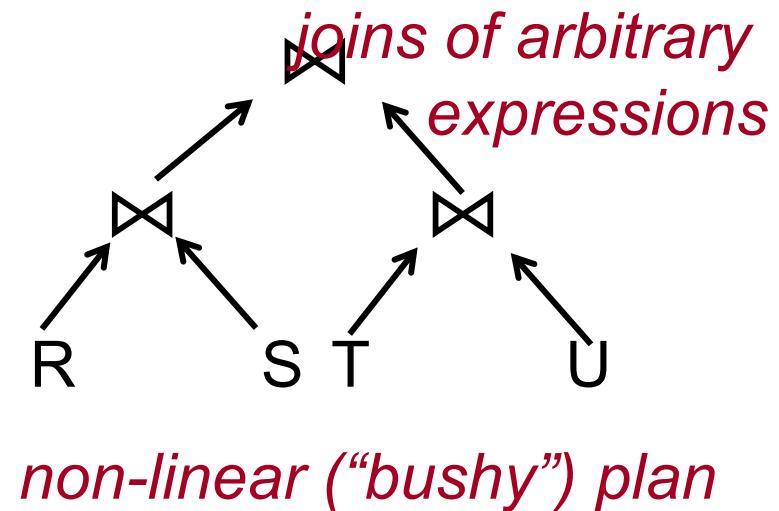
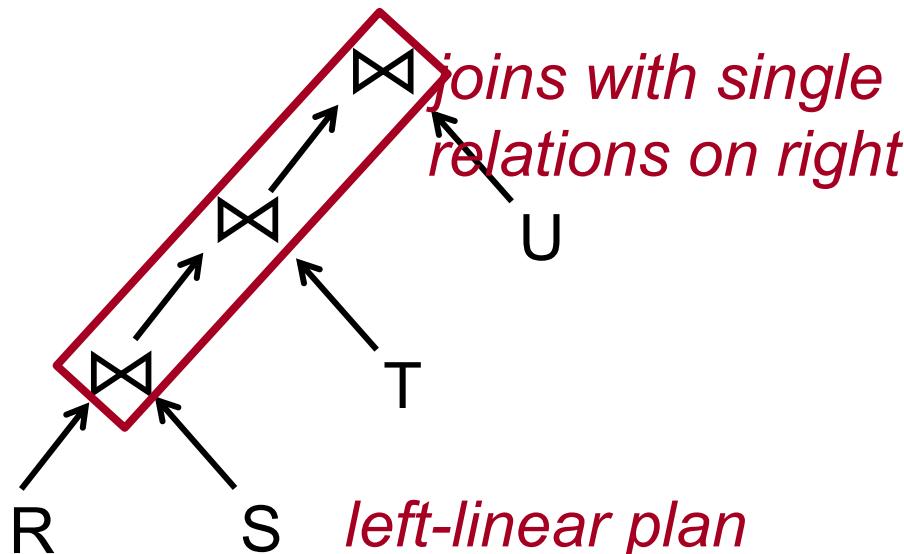


*Feeds data from left input to the wrapper in the right input*

# The Basic Approach

Enumerate all possible query expressions

- “bushy” plan enumeration instead of “left-linear” as was illustrated for relational DBMS optimization



Use a different set of cost formulas for pushed-down operations, custom to the wrapper and the source

# Changes to Query Execution

---

- Internet-based sources present challenges:
  - Slow and unpredictable data transfer rates
  - Source failures
- Many data integration applications also emphasize early results!
- Emphasize maximally pipelined query processing + failure handling...

# Threads to the Rescue

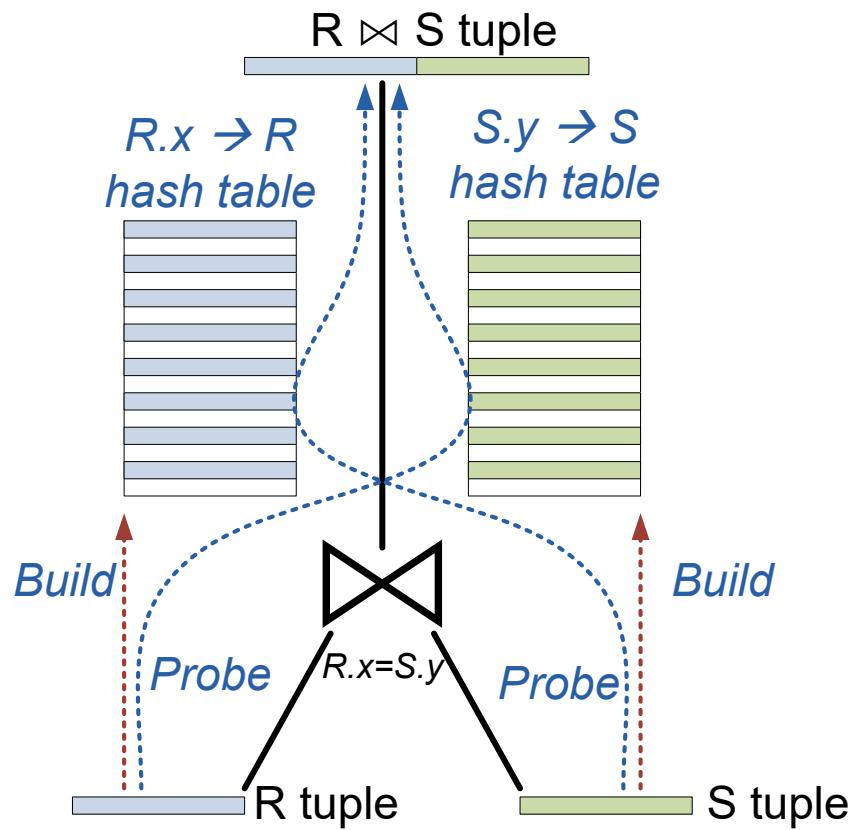
---

- Different sources will send data at different rates – an architecture based on iterators may hold up the CPU when it's waiting for I/O
  - e.g., we are trying to read one table but no data is yet ready
- Develop fully pipelined join operators with multiple threads (one to process each input, one for the computation)...
  - An instance of the **pipelined hash join**

# Pipelined Hash Join (aka Symmetric Hash Join)

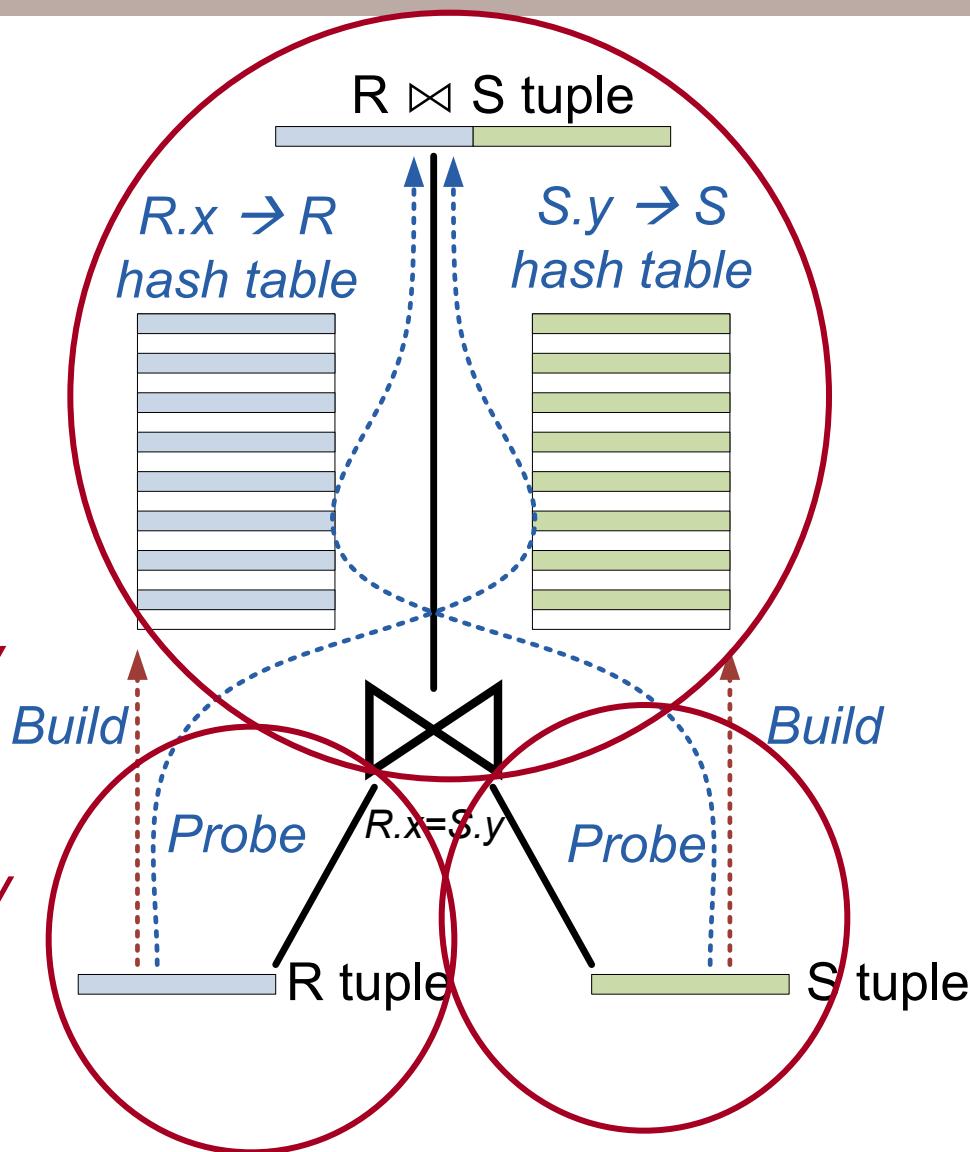
Operates symmetrically,  
and is fully pipelined:

1. Read from either input
2. Add tuple to input's hash table
3. Probe against opposite hash table
4. Return resulting output

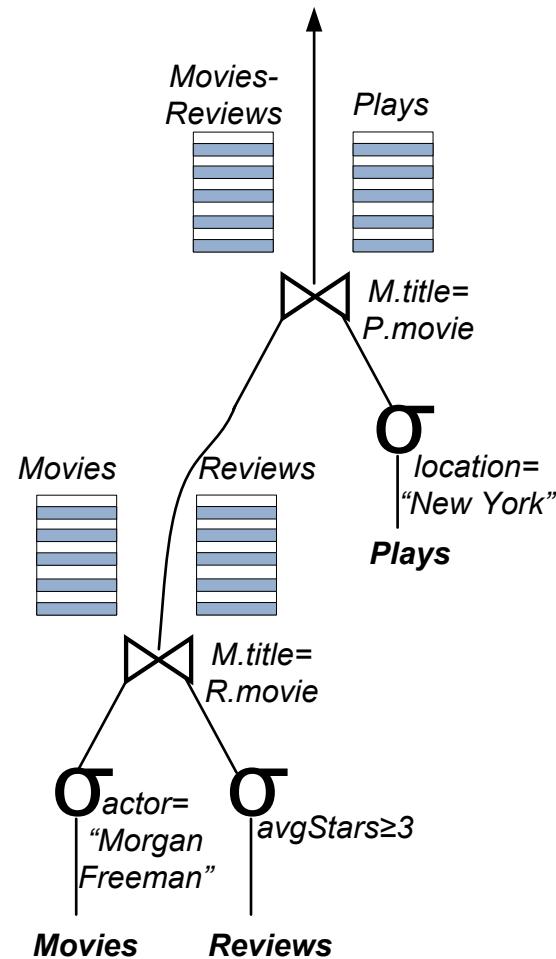


# Pipelined Hash Join

*Each red ellipse represents a separate thread of control, scheduled by the CPU based on availability*



# Pipelined Join Plan



# Other Pipelined Operators

---

- In general, all algorithms will be hash-based as opposed to index- or sort-based
  - This supports pipelined execution in a networked setting
- Sometimes this requires techniques for when hash tables exceed memory capacity
  - Spill to disk, process by recursively applying hashing or by using nested loops algorithms

# A Problem: Problems Can Arise!

---

- It's not uncommon for a data source to become unavailable
  - The server crashes, the network becomes unavailable, a job gets preempted, etc.
- Operators in a data integration query plan need to generate *events*, much like exceptions in a programming environment
  - These need to be handled by the query processor, as we describe next...

# Adaptive Query Processing

---

Distributed, pipelined execution is a necessity to handle data integration queries

However, we can still end up with plans that:

- Encounter massive **delays** for some sources
- Run into **errors** – e.g., a source fails
- Are highly **inefficient** – e.g., produce huge amounts of intermediate state due to bad cost estimates

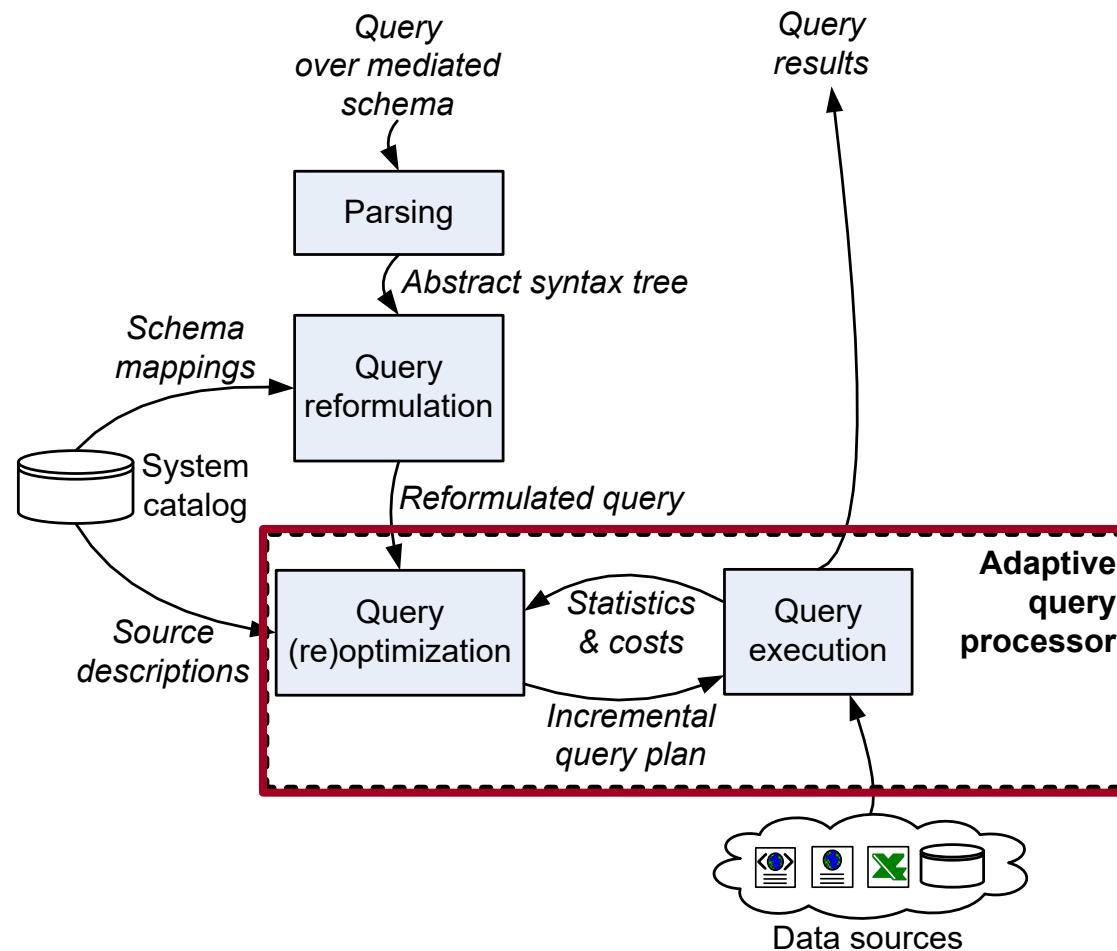
This motivates **adaptive** query processing

# The First Step: A Unified Query Processor

---

- Most relational DB engines have separate query optimization and execution modules
  - (This is in fact changing)
- Our goal here: start executing a query and **adapt** to conditions
- Requires a unified query processor that loops between decision-making and execution

# Query Processing in Data Integration



# Challenges of Adaptive Query Processing

---

- Generally a balance among many factors:
  - How much information to collect – exploration vs. exploitation
  - Analysis time vs. execution time – more optimization results in better plans, but less time to spend running them
  - Space of adaptations – how much can we change execution?
  - Short-term vs. long-term – balancing between making the best progress **now**, and incurring state that might cost a lot **later**.
- We divide into **event-** and **performance-driven** types

# Event-Driven Adaptivity

---

Consider changing the plan at designated points where the query processor is stopped

- When a source fails or times out
- When a segment of the query plan complete and is materialized
  - ❖ ... and we verify at runtime that the materialized result size is significantly different from expectations
- We can encode decision-making for the execution system through a rule mechanism – **on** event **if** condition **then** take action... including re-running the optimizer!

# Source Failure Events and Rules

---

- Source failures can occur in two main ways:
  - a source times out
  - a source is unreachable

on timeout(wrapper1, 10msec) ...

- There can be multiple kinds of responses
  - Find an alternative source

on timeout(wrapper1, 10msec) if true then activate(coll1, A)
  - Reschedule (suspend until later) a subplan or operator

on timeout(wrapper1, 10msec) if true then reschedule(op2)

# How Do We Generate the Rules?

---

- For source failures, we need a partial ordering of data sources with alternates
  - The optimizer can generate rules to access the alternates if the primaries are down
- For rescheduling, the optimizer can defer parts of the plan that are dependent on a blocked data source

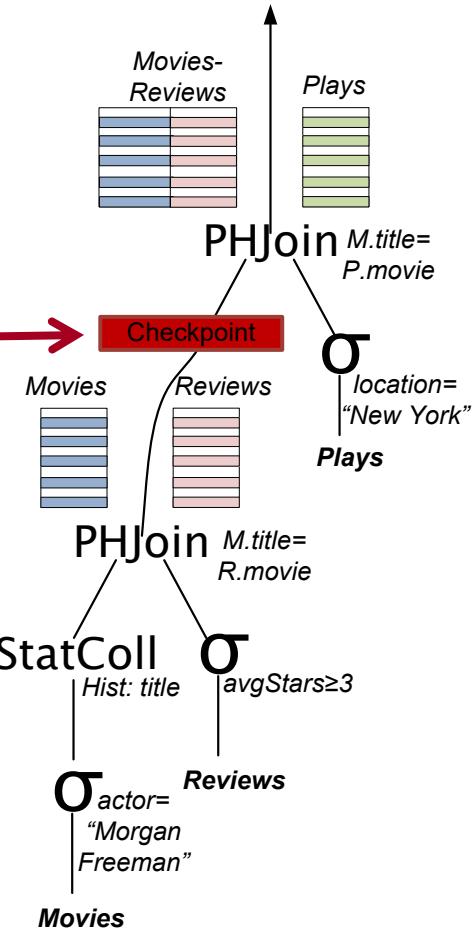
# Cost Mis-estimation Events

---

- Another source of problems: the optimizer mis-estimates the cost or cardinality of a result
  - ... and in turn this may cause sub-optimal execution of the rest of the plan!
- Can we find points at which to **change** the rest of the plan, based on what we discover in early execution?
  - Gather information during execution – special operators to track cardinalities, build histograms
  - Periodically “checkpoint” and consider re-optimizing
  - If necessary, re-optimize with the results we’ve computed

# Mid-query Re-optimization

- Break the plan into several pipelines, e.g., based on uncertainty of results
  - Write the results to disk, place a “checkpoint”
- During each stage, add statistics collection operations
  - e.g., histograms on join attributes
- At the checkpoint, re-estimate costs
  - If they are beyond a threshold, re-run the optimizer



# Re-Optimization

---

- In an event-driven setting, the optimizer only gets re-run if it looks likely that costs for the remaining plan will be significantly different
- The existing result is typically treated as a **materialized view** and the initial query can be re-optimized
  - If useful, the materialized view will be directly used by the optimizer
  - In the worst case, the size of the view, plus the statistics gathered, result in a better cost estimate

# Challenges of Event-Driven Adaptivity

---

- Checkpoints are expensive – they prevent fully pipelined execution
  - It is hard to decide where to put the checkpoint!
  - Sometimes we will waste a lot of work in getting to the checkpoint
- 
- This motivates **performance-driven adaptivity...**

# Performance-Driven Adaptivity

- Suppose we want to change query plans in mid-execution, without re-reading data
- Let's exploit the fact that multiple relational expressions can do equivalent computation
  - Break up the data into **segments or phases**
  - Run a different **plan** over each phase
  - Put the results together!
- By the relational algebra:

$$\Pi_{\bar{A}}(\sigma(R_1 \bowtie \dots \bowtie R_m)) \equiv \bigcup_{1 \leq c_1 \leq n, \dots, 1 \leq c_m \leq n} \Pi_{\bar{A}}(\sigma(R_1^{c_1} \bowtie \dots \bowtie R_m^{c_m}))$$

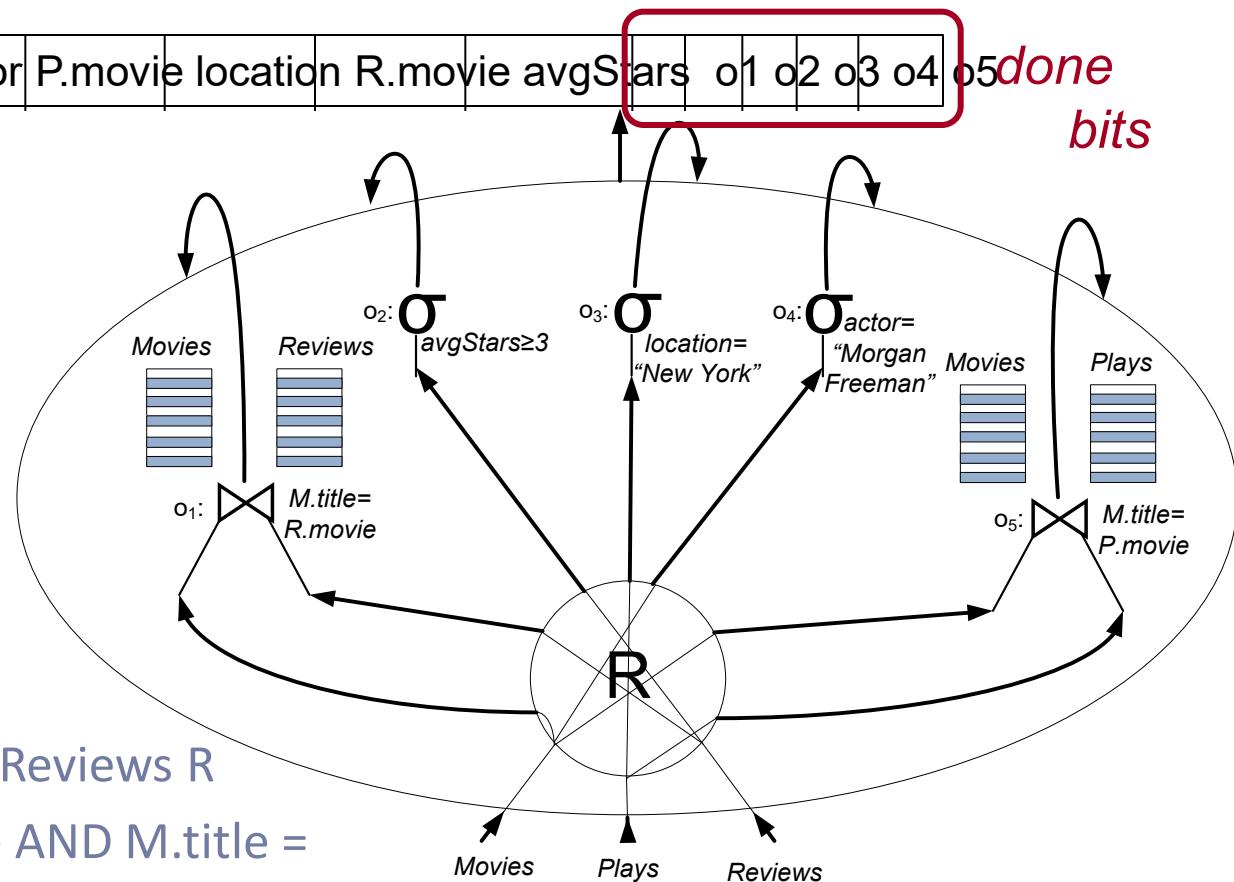
# Flow-Based Plan Selection: Eddies

---

- Suppose we want to get rid of the optimizer altogether, except possibly to create a very basic initial plan
  - Use **network routing** ideas to process tuples
  - Treat the problem as one of **routing** tuples among different **operators**
    - ❖ Flow control – send tuples to operators that are not busy
- A (select-project-join) plan becomes an **eddy** with
  - A **tuple router** that controls where tuples go
  - A set of operators
  - A set of tuples, marked with where they have been

# Eddy

*Tuple:* title|actor|P.movie|location|R.movie|avgStars|o1|o2|o3|o4|o5|done  
bits



```

SELECT title, startTime
FROM Movie M, Plays P, Reviews R
WHERE M.title = P.movie AND M.title =
 R.movie AND P.location ="New York "
 AND M.actor = "Morgan Freeman"
 AND R.avgStars >= 3

```

# Routing Policies

- An eddy routes a tuple to an operator, which
  - May drop it (as with a selection)
  - May “shrink” it (as with a projection) or “expand” it (as with a join)
  - May produce multiple output tuples (as with a join)
- Given an input (partial) tuple, how do we determine which operator to route it to?
  - This is the **routing policy**. Consider “lottery scheduling”:
    - ❖ Give each operator a series of “tickets” – more for highly selective operators
    - ❖ For any tuple, randomly choose a ticket for where to go next
      - ◆ Ignore operators that are busy (queue is full) or already done

# Issues with Eddies... And Enhancements

---

- Lottery scheduling works well for selection operators
- But join is “weird”:
  - It filters (or multiplies) tuples, but also accumulates **state**
  - Response: state modules (STeMs)
    - ❖ Separate the state management and filter logic into two logical operations – helps the eddy track behavior
- The eddy can “make mistakes”
  - Sometimes it’s easy to accumulate lots of state early – and later find that we have to join a lot of tuples with the state
  - Response: STAIRs
    - ❖ “Migrate” state by disassembling joined results, sending them to other operators in the eddies

# Another Concern

---

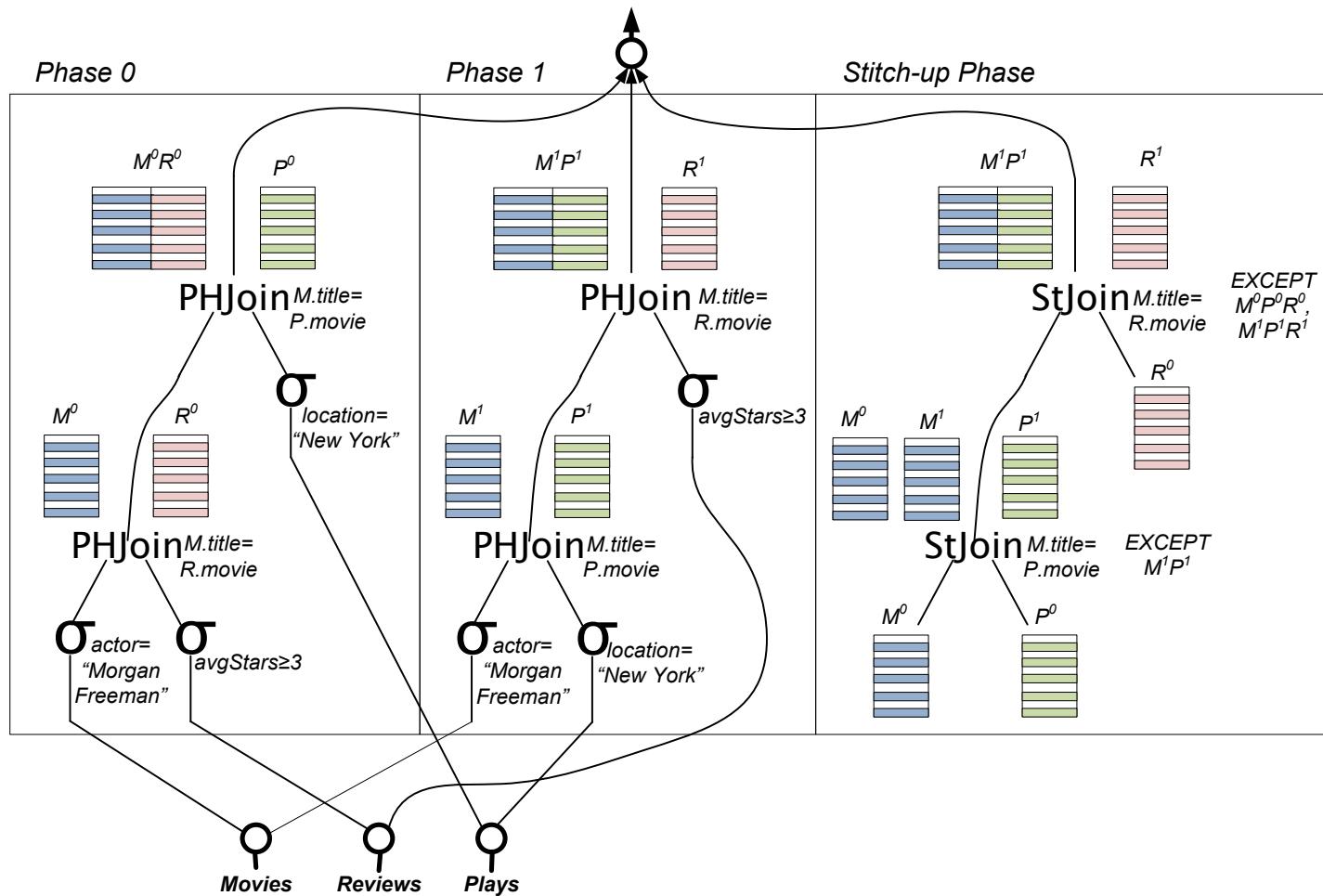
- The eddy makes use of flow control to “steer” towards an optimal plan
- This means it will quickly adapt to the plan that currently results in best performance
- But we also constantly send tuples through sub-optimal paths as a result
  
- Can we do something reactive like eddies, which uses cost estimation to guide the paths that tuples follow?

# Corrective Query Processing

---

- The basic idea of **corrective query processing**:
  - Frequent optimize-execute loop
  - We monitor progress and continuously re-estimate cost
  - If cost exceeds some threshold, we **suspend** the current query plan, use the optimizer to choose a new plan
- The new plan “resumes” from the input streams that the old plan was using
- Ultimately we need to combine data across these plans (phases) – this is what we call the **stitch-up phase**

# Corrective Query Processing



# How Cost Re-Estimation and Re-optimization Work

---

- Every few seconds, plan status is polled
  - Cardinalities are re-estimated based on “progress so far” (based on an estimate of how much more data remains)
- The query optimizer’s cost estimator is re-run over the current plan\*
  - If it diverges by a threshold, re-optimization is triggered in the background (while execution continues)
  - Re-optimization uses updated selectivity + cost values
- If a “significantly better” plan is found, the current plan is suspended and the new one is started

\* The optimizer remains in memory at all times

# Stitch-up Plans

---

- A challenge: “new” results need to be joined with “old ones”
- We can do this in three ways:
  - Feed new results into old plans, join with old data
  - Feed old results into new plans, join new data with old
  - **Do all of the cross-plan joining in a separate stage (phase)**
- The last one is called a **stitch-up plan** and can be done with special join algorithms that efficiently compute only those answers that remain

# Corrective QP versus Eddies

---

Easy to extend to more complex queries

- Aggregation, grouping, subqueries, etc.

Separates two factors, **conservatively** creates state:

- Scheduling is handled by pipelined operators
- CQP chooses plans using long-term cost estimation
- Postpones cross-phase results to final phase

Assumes settings where computation cost, state are the bottlenecks

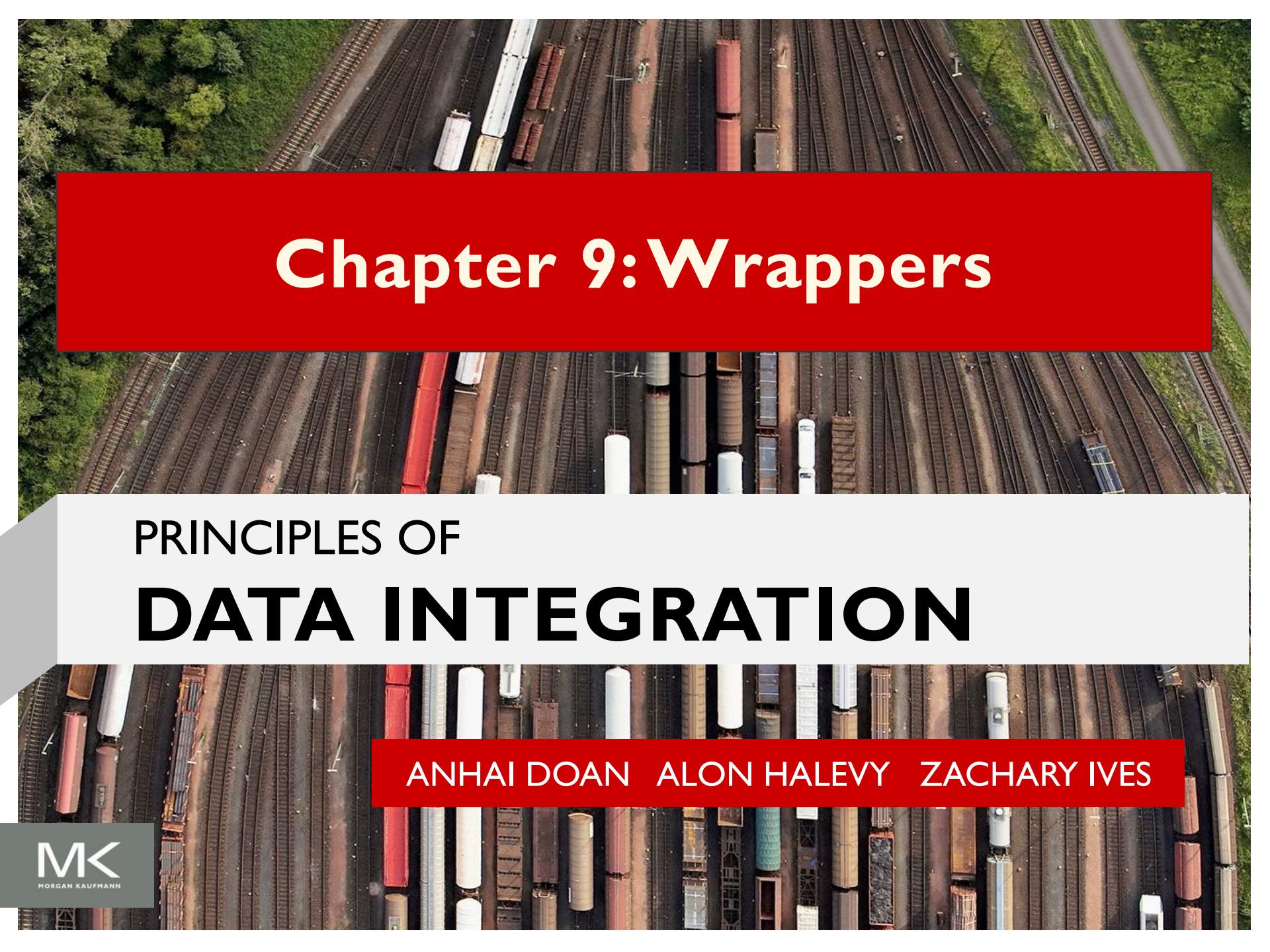
Note this is not a requirement of the method – but one implementation

- Contrast with eddies + STAIRS, which eagerly do computation, and move state around once it's created!

# Summary

---

- Query processing for data integration builds upon past models
  - From centralized DBMSs: plan enumeration, cost modeling, basic query execution architectures
  - From distributed DBMSs: modeling communication cost, data distribution algorithms
- But a new emphasis on:
  - Wrappers, access patterns, and query subexpression push-down
  - Fully pipelined algorithms, multithreaded operation
  - Adaptivity – event-based and performance-based
- Also, we may encounter XML (see Chapter 11)

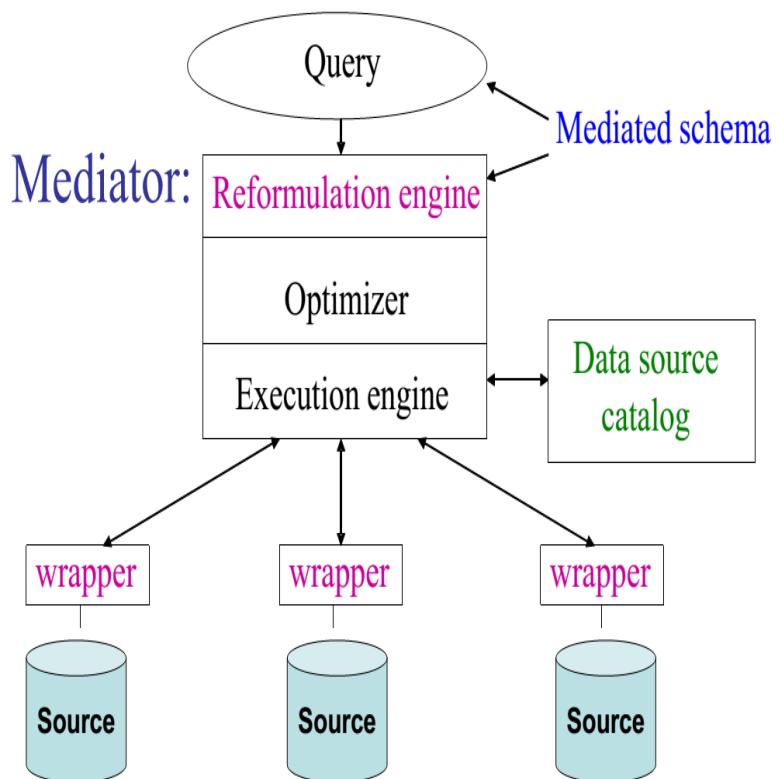


# Chapter 9: Wrappers

PRINCIPLES OF  
**DATA INTEGRATION**

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Introduction



| Week                    | Topic                                                     |
|-------------------------|-----------------------------------------------------------|
| Tues. 03.11.2020        | Ch. 1: Introduction                                       |
| Tues. 10.11.2020        | Ch. 2: Query expression                                   |
| Tues. 17.11             | Ch. 3: Data sources description                           |
| Tues. 24.11             | Ch. 4: String matching                                    |
| Tues. 01.12             | Ch. 5: Schema matching & mapping                          |
| Tues. 08.12             | Ch. 7: Data matching ( <b>Project discussion</b> )        |
| <u>Tues. 15.12</u>      | <u>Ch. 8: Query processing - I</u>                        |
| Tues. 22.12             | Ch. 8: Query processing - 2                               |
| <b>24.12-02.01.2019</b> | <b>Charismas break</b>                                    |
| Tues. 05.01.2021        | <u>Ch. 9:Wrapper</u>                                      |
| Tues. 12.01.2021        | <u>Ch. 10: XML : Ch. 9:Wrapper</u>                        |
| Tues. 19.01             | Ch. 11: Uncertainty: Ch. 10: XML                          |
| Tues. 26.01             | <u>Ch. 11: Ontology</u>                                   |
| Tues. 02.02             | Ch. 12: Web data integration: Ch. 11: Uncertainty         |
| Tues. 09.02             | Ch. 13: Semantic web services ( <b>Prof. König-Ries</b> ) |

# Introduction

---

- Wrappers are components of DI systems that communicate with the data sources
  - sending queries from higher levels in the system to the sources
  - converting replies to a format that can be manipulated by query processor
- Complexity of wrapper depends on nature of data source
  - e.g., source is RDBMS, wrapper's task is to interact with JDBC driver
  - in many cases, wrapper must parse semi-structured data such as HTML pages and transform it into a set of tuples
  - we focus on this latter case

# Outline

---

- Problem definition
- Manual wrapper construction
- Learning-based wrapper construction
- Wrapper learning without schema
  - a.k.a., automatic approaches
- Interactive wrapper construction (for further reading)

# Data sources

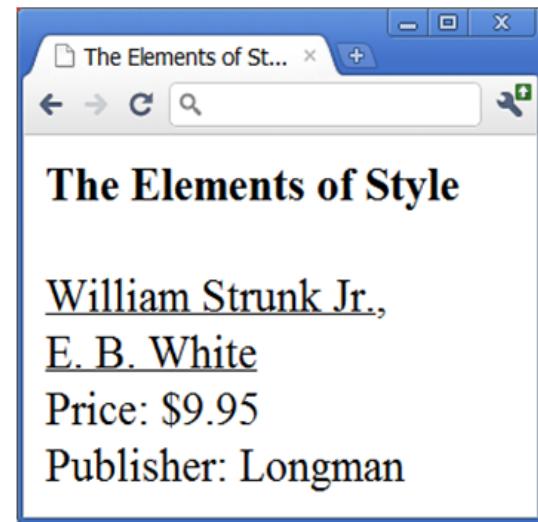
- Consider data sources that consist of a **set of Web pages**
- For each source **S**, assume each Web page displays structured data using a schema **T<sub>S</sub>** and a format **F<sub>S</sub>**
  - these are common across all pages of the source



(a) [countries.com](http://countries.com)



(b) [easycalls.com](http://easycalls.com)



(c) [greatbooks.com](http://greatbooks.com)

# Data sources

- These kinds of pages are very common on the Web
  - often created in sites that are powered by database systems
  - user sends a query to the database system
    - ❖ e.g., list all countries and calling codes in the continent Australia
  - system produces a set of tuples
  - a scripting program creates an HTML page that embeds the tuples , using a schema  $T_s$  and a format  $F_s$
  - the HTML is sent to the user



# Wrapper

---

- A wrapper  $W$  extracts structured data from pages of  $S$
- Formally,  $W$  is a tuple  $(T_W, E_W)$ 
  - $T_W$  is a target schema
    - ❖ this needs not be the same as the schema  $T_S$  used on the page, because we may want to extract only a subset of attributes of  $T_S$
  - $E_W$  is an extraction program that uses format  $F_S$  to extract from each page a data instance conforming to  $T_W$ 
    - ❖  $T_W$  is typically written in a script language (e.g., Perl) or in some higher-level declarative language that an execution engine can interpret

# Example 1

- Consider a **wrapper** that extracts all attributes from pages of *countries.com*
  - target schema  $T_W$  is the source schema  $T_S = (\text{country}, \text{capital}, \text{population}, \text{continent})$
  - extraction program  $E_W$  may be a Perl script that specifies that given a page P from this source
    - ❖ return the first fully capitalized string as country
    - ❖ return the string immediately following “Capital:” as capital
    - ❖ etc.



*countries.com*

# Example 2

- Consider a **wrapper** that extracts only the first two attributes from pages of **countries.com**
  - target schema  $T_W$  is *(country, capital)*
  - extraction program  $E_W$  may be a Perl script that specifies that given a page P from this source
    - ❖ return the first fully capitalized string as country
    - ❖ return the string immediately following “Capital:” as capital



*countries.com*

# Wrapper construction problem

- Construct  $(T_w, E_w)$  by inspecting the pages of  $S$ 
  - also called **wrapper learning**
- Two main variants
  - given schema  $T_w$ , construct extraction program  $E_w$ 
    - ❖ e.g., given  $T_w = (\text{country}, \text{capital})$ , construct  $E_w$  that extracts these two attributes from source countries.com
    - ❖ manual/learning/interactive approaches address this problem (see later)
  - no  $T_w$  is given, instead, construct the source schema  $T_s$  and take it to be the target schema  $T_w$ , then construct  $E_w$ 
    - ❖ e.g., given pages from source countries.com, learn their schema  $T_s$ , then learn  $E_w$  program that extracts the attributes of  $T_s$
    - ❖ the automatic approach addresses this problem (see later)

# Challenges of wrapper construction

## 1. Learning source schema $T_s$ is very difficult

- typically view each page of  $S$  as a string generated by a grammar  $G$
- learn  $G$  from a set of pages of  $S$ , then use  $G$  to infer  $T_s$
- e.g., pages of countries.com may be generated by  
 $R = <\text{html}>.+\?<\text{hr}><\text{br}>(.+?)<\text{br}>\text{Capital}: (.+?)<\text{br}>\text{Population}: (.+?)<\text{br}>\text{Continent}: (.+?)</\text{html}>$   
which encodes a regular grammar
- inferring a grammar from positive examples (i.e., pages of  $S$ ) is well-known to be difficult
  - ❖ regular grammars cannot be correctly identified from positive examples
  - ❖ even with both positive and negative examples, there is no efficient algorithm to identify a reasonable grammar (i.e., one that is minimal)

# Challenges of wrapper construction

1. Learning source schema  $T_S$  is very difficult (cont.)
  - current solutions consider only relatively simple regular grammars that encode either flat or nested tuple schemas
  - even learning these simple schemas has proven difficult
  - typically use various heuristics to search a large space of candidate schemas
    - ❖ incorrect heuristics often lead to incorrect schemas
    - ❖ increasing the complexity of the schema even slightly can lead to an exponential increase in size of search space, resulting in an intractable searching process

# Challenges of wrapper construction

## 2. Learning the extraction program $E_w$ is difficult

- ideally,  $E_w$  should be Turing complete (e.g., as a Perl script) to have maximal expressive power, but impractical to learn such programs
- so assume  $E_w$  to be of a far more restricted computational model, then learn only the limited set of parameters of model
  - ❖ e.g., learning to extract country and capital from pages of countries.com
  - ❖ assume  $E_w$  is specified by a tuple  $(s_1, e_1, s_2, e_2)$ :  $E_w$  always extract the first string between  $s_1$  and  $e_1$  as country, and the first string between  $s_2$  and  $e_2$  as capital
  - ❖ so here learning  $E_w$  reduces to learning the above four parameters
- even just learning a limited set of parameters has proven quite difficult, for reasons similar to those of learning schema  $T_s$

# Challenges of wrapper construction

- Coping with myriad exceptions
  - there are often many exceptions in how data is laid out and formatted
    - ❖ e.g., (title, author, price) in the normal cases, but attributes (e.g., price) may be missing, attribute order may be reversed (e.g., (author, title, price)), or attribute format may be changed (e.g., price in red font)
  - when inspecting a small number of pages to create wrapper, such exceptions may not be apparent (yet)
  - thus exceptions cause many problems
    - ❖ invalidate assumptions on schema/data format, thus producing incorrect wrappers
    - ❖ force us to revise source schema  $T_S$  and extraction program  $E_W$ , such revisions blow up the search space

# Main solution approaches

---

- Manual
  - developer manually creates  $T_w$  and  $E_w$
- Learning
  - developer highlights the attributes of  $T_w$  in a set of Web pages, then applies a learning algorithm to learn  $E_w$
- Automatic
  - automatically learn both  $T_s$  and  $E_w$  from a set of Web pages
- Interactive
  - combines aspects of learning and automatic approaches
  - developer provides feedback to a program until convergence

# Outline

---

- Problem definition
- Manual wrapper construction
- Learning-based wrapper construction
- Wrapper learning without schema
  - a.k.a., automatic approaches
- Interactive wrapper construction

# Manual wrapper construction

- Developer examines a set of Web pages
  - manually creates target schema  $T_W$  and extraction program  $E_W$
  - often writes  $E_W$  using a procedural language such as Perl

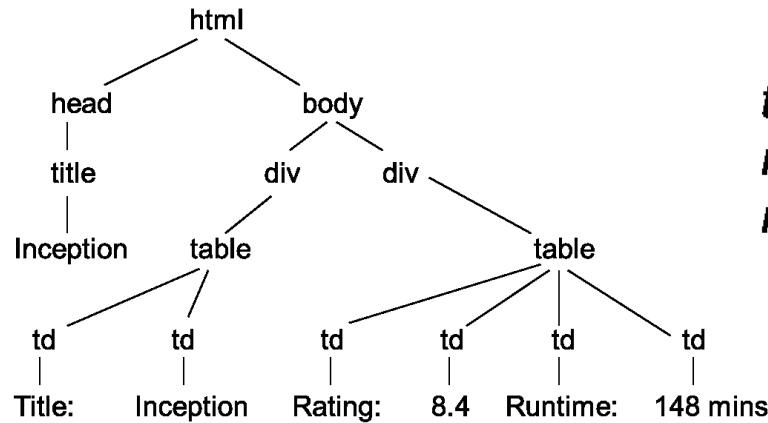


```
#!/usr/bin/perl -w

open(INFILE, $ARGV[0]) or die "can't open file\n";
while ($line = <INFILE>) {
 if ($line =~ m/(.+?)<VB>\s+?</>(\d+?)</V>
/) {
 print "($1,$2)\n";
 }
}
close(INFILE);
```

# Manual wrapper construction

- There are multiple ways to view a page
  - as a string → can write wrapper as Perl program
  - as a DOM tree → can write wrapper using XPath language



*title* = /html/body/div[1]/table/td[2]/text()  
*rating* = /html/body/div[2]/table/td[2]/text()  
*runtime* = /html/body/div[2]/table/td[4]/text()

- as a visual page, consisting of blocks

# Manual wrapper construction

---

- Regardless of page model (string, DOM tree, visual, etc.), using a low-level procedural language to write  $E_W$  can be very laborious
- High-level wrapper languages have been proposed
- E.g., **HLRT language**
  - see the next part on learning
- Using high-level language often result in loss of expressiveness
- But they are often easier to understand, debug, and maintain

# Outline

---

- Problem definition
- Manual wrapper construction
- Learning-based wrapper construction
- Wrapper learning without schema
  - a.k.a., automatic approaches
- Interactive wrapper construction

# Learning-based wrapper construction

---

- Consider more limited wrapper types (compared to the manual approach)
- But can automatically learn these using training examples
- Providing such examples typically involve marking up Web pages
  - can be done by technically naïve users
  - often requires far less work than manually writing wrappers
- We explain learning approaches using two wrapper types
  - HLRT
  - Stalker

# HLRT wrappers

- Use string delimiters to specify how to extract tuples



```
<HTML>
<TITLE>Countries in Australia (Continent)</TITLE>
<BODY>
Countries in Australia (Continent)<P>
Australia <I>61</I>

East Timor <I>670</I>

Papua New Guinea <I>675</I>

<HR>
Copyright easycalls.com
</BODY>
</HTML>
```

} head  
} data region  
} tail

- To extract *(country, code)*, an HLRT wrapper can
  - chop off the head using `<P>`, chop off the tail using `<HR>`
  - extract strings between `<B>` and `</B>` in the data region as countries, and between `<I>` and `</I>` as codes

# HLRT wrappers



```
<HTML>
<TITLE>Countries in Australia (Continent)</TITLE>
<BODY>
Countries in Australia (Continent)<P>
Australia <I>61</I>

East Timor <I>670</I>

Papua New Guinea <I>675</I>

<HR>
Copyright easycalls.com
</BODY>
</HTML>
```

} *head*  
} *data*  
} *region*  
} *tail*

- Thus, **HLRT = Head-Left-Right-Tail**
- Above wrapper can be represented as tuple  
(<P>, <HR>, <B>, </B>, <I>, </I>)
- Formally, an HLRT wrapper that extracts **n** attributes is a tuple of  $(2n + 2)$  strings  $(h, t, l_1, r_1, \dots, l_n, r_n)$

# Learning HLRT wrappers

- Suppose
  - D wants to extract  $n$  attributes  $a_1, \dots, a_n$  from source S
  - after examining pages of S, D has established that an HLRT wrapper  $W = (h, t, l_1, r_1, \dots, l_n, r_n)$  will do the job
- Our goal: learn  $h, t, l_1, r_1, \dots, l_n, r_n$ 
  - to do this, label a set of pages  $T = \{p_1, \dots, p_m\}$ 
    - ❖ i.e., identifying in  $p_i$  the start and end positions of all values of attributes  $a_1, \dots, a_n$ , typically done using a GUI
  - feed the labeled pages  $p_1, \dots, p_m$  into a learning module
  - learning module produces  $h, t, l_1, r_1, \dots, l_n, r_n$

# Example of a learning module for HLRT

---

- A simple module systematically searches the space of all possible HLRT wrappers

## 1. Find all possible values for $h$

- let  $x_i$  be the string from the beginning of page  $p_i$  (a labeled page) until the first occurrence of the very first attribute  $a_1$
- then  $x_1, \dots, x_m$  contains the correct  $h$
- thus, take the set of all common substrings of  $x_1, \dots, x_m$  to be candidate values for  $h$

## 2. Find all possible values for $t$ :

- similar to the case of finding all possible values for  $h$

# Example of a learning module for HLRT

### 3. Find all possible values for each $\mathbf{l}_i$

- e.g., consider  $\mathbf{l}_1$ , the left delimiter of  $\mathbf{a}_1$
- $\mathbf{l}_1$  must be a common suffix of all strings (in labeled pages) that end right before a marked value of  $\mathbf{a}_1$
- can take the set of all such suffixes to be candidate values for  $\mathbf{l}_1$

### 4. Find all possible values for each $\mathbf{r}_i$

- similar to case of  $\mathbf{l}_i$ , but consider prefixes instead of suffixes

### 5. Search in the combined space of the above values

- combine above cand values to form cand wrappers
- if a cand wrapper  $W$  correctly extracts all values of  $\mathbf{a}_1, \dots, \mathbf{a}_n$  from all labeled pages  $\mathbf{p}_1, \dots, \mathbf{p}_m$ , then return  $W$
- The notes discuss optimizing the above learning module

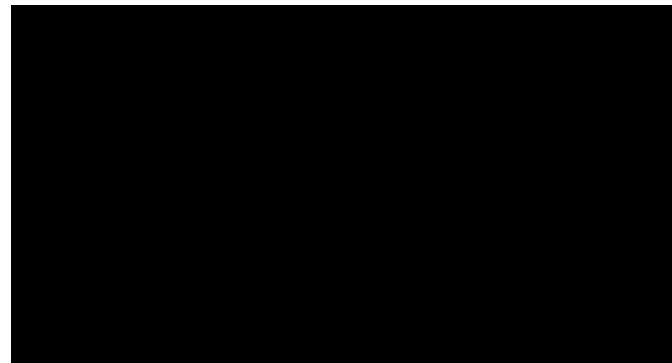
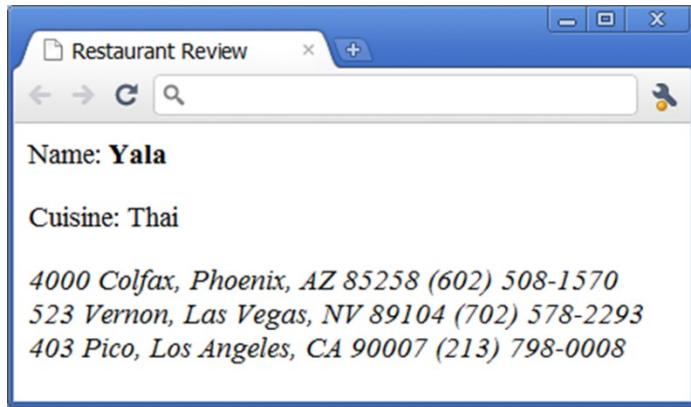
# Limitations of HLRT wrappers

---

- HLRT wrappers are easy to understand and implement
- But have limited applicability
  - assume a **flat tuple schema**
  - assume all attributes can be extracted using delimiters
- In practice
  - many sources use more complex schemas, e.g., nested tuples
    - ❖ book is modeled as a tuple (title, authors, price), where authors is a list of tuples (first-name, last-name)
  - may not be able to extract using delimiters
    - ❖ extracting zip codes from “40 Colfax, Phoenix, AZ 85258”
- Stalker wrappers address these issues

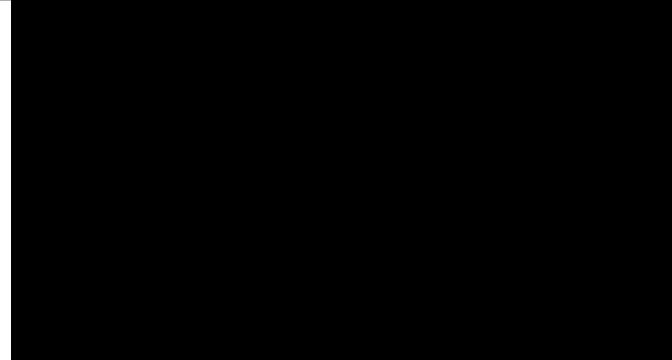
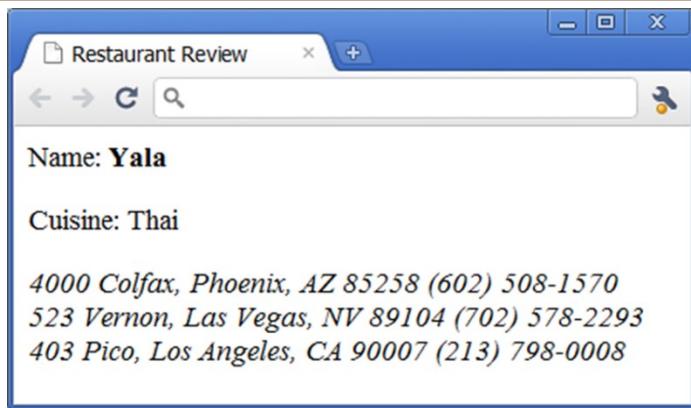
# Nested tuple schemas

- Stalker wrappers use nested tuple schemas



- here each page is a tuple (name, cuisine, addresses), where **addresses** is a list of tuples (street, city, state, zipcode, phone)
- Nested tuple schemas are very commonly used in Web pages
  - capture how many people think about the data
  - convenient for visual representation

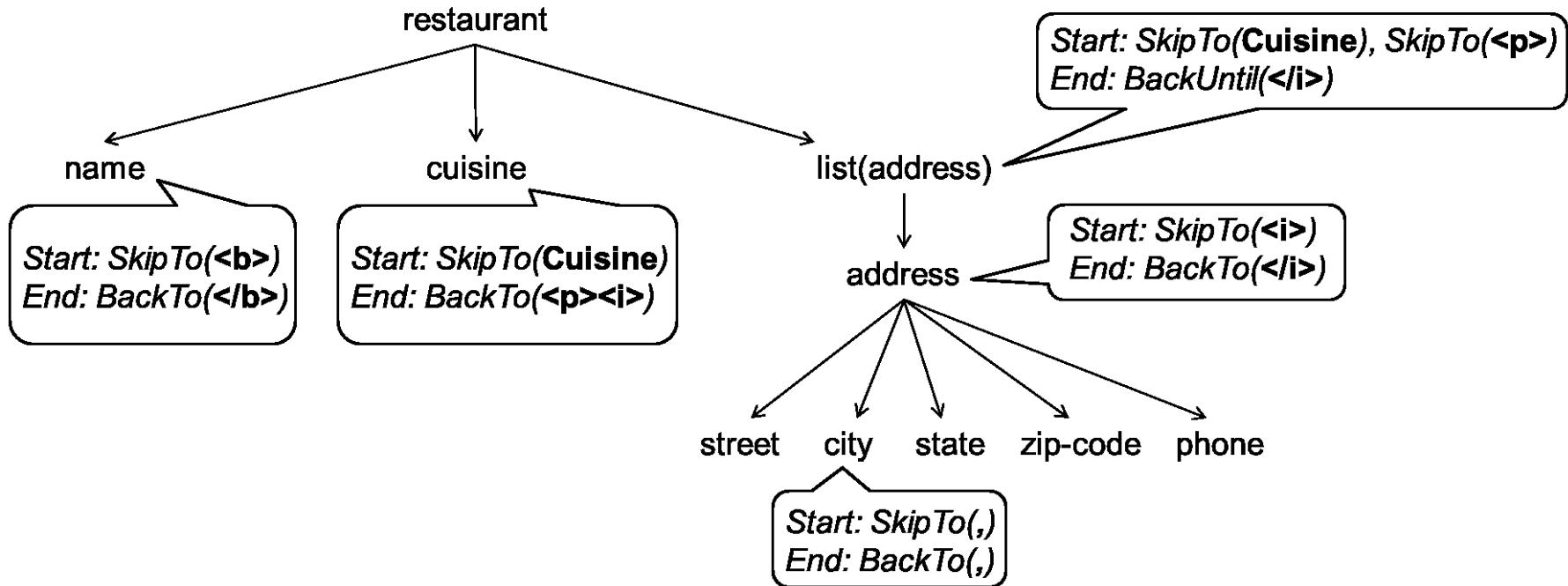
# Nested Tuple Schemas



- Definition: let  $\mathbf{N}$  be the set of all nested tuple schemas
  - the schema displaying data as a single string belongs to  $\mathbf{N}$
  - if  $T_1, \dots, T_n$  belong to  $\mathbf{N}$ , then the tuple schema  $(T_1, \dots, T_n)$  belongs to  $\mathbf{N}$
  - if  $T$  belongs to  $\mathbf{N}$ , then the list schema  $\langle T \rangle$  also belongs to  $\mathbf{N}$
- A nested tuple schema can be visualized as a tree
  - leaves are strings; internal nodes are tuple or list nodes

# The Stalker wrapper model

- A Stalker wrapper
  - specifies a nested-tuple schema in form of a tree
  - assigns to each tree node a set of rules that show how to extract values for that node

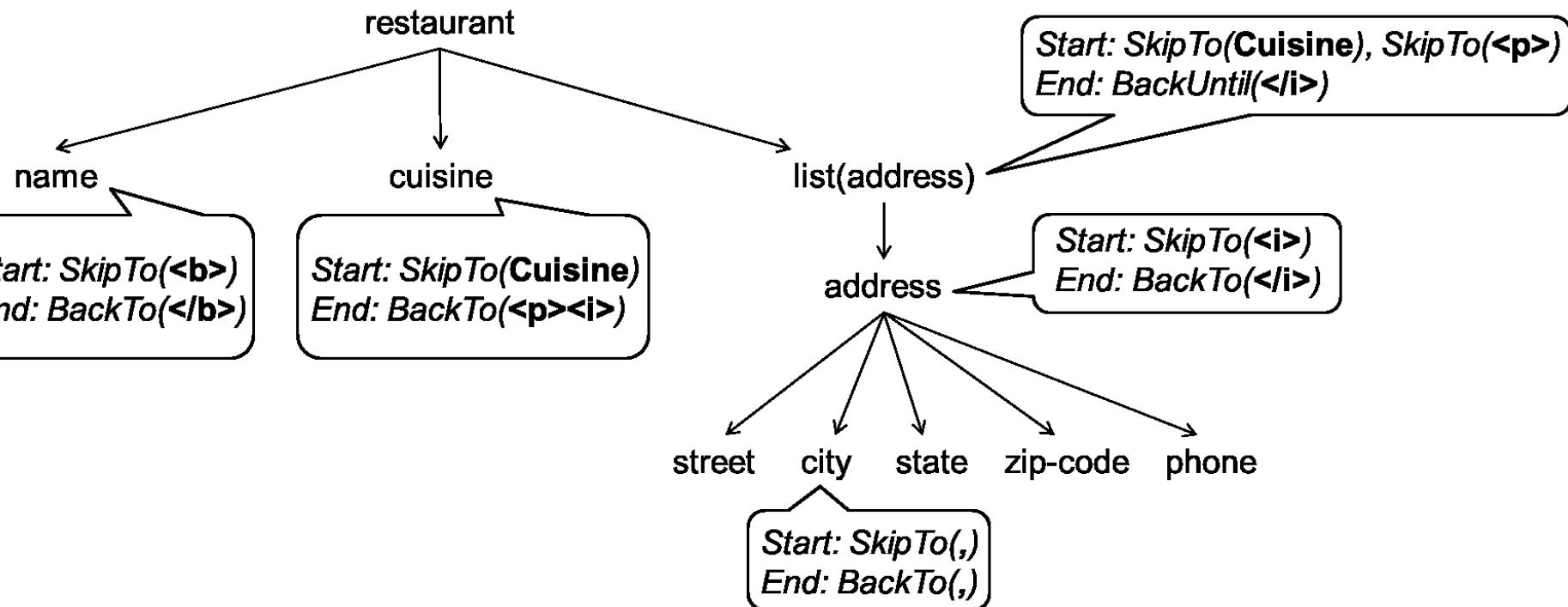


# Example of executing the wrapper

```
<p> Name: Yala<p>Cuisine: Thai<p>
<i>4000 Colfax, Phoenix, AZ 85258 (602) 508-1570</i>

<i>523 Vernon, Las Vegas, NV 89104 (702) 578-2293</i>

<i>403 Pico, LA, CA 90007 (213) 798-0008</i>
```



# Stalker extraction rules

---

- Each rule = context: sequence of commands
- Context: Start, End, etc.
- Sequences of commands: SkipTo(<b>)  
SkipTo(Cuisine:), SkipTo(<p>)
- Each command inputs a landmark
  - e.g., <b>, Cuisine:, <p>, or triple (Name Punctuation HTMLTag)
- A landmark = sequence of tokens and wildcards
  - each wildcard refers to a class of tokens
    - ❖ e.g., Punctuation, HTMLTag
  - a landmark = a restricted kind of regex

# Stalker extraction rules

---

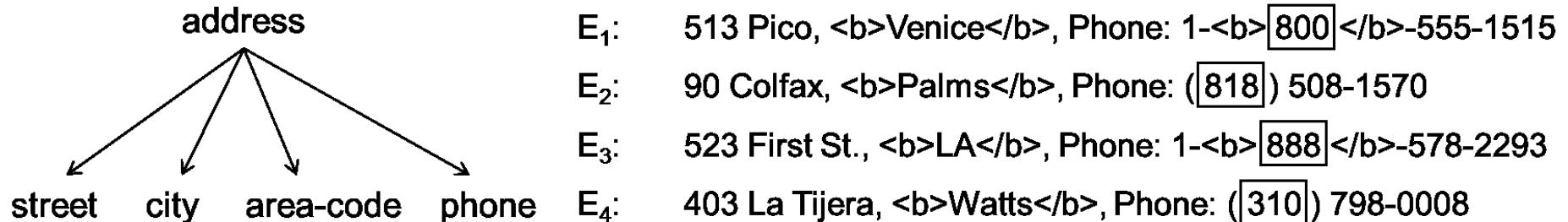
- Each rule = context:sequence of commands
- Executing rule = executing commands sequentially
- Executing command = consuming text until reaching a string that matches the input landmark
- Stalker also considers rules that contain disjunction of sequences of commands
  - Start: either SkipTo(<b>) or SkipTo(<i>)

# Learning stalker wrappers

---

- Input (of the learner):
  - a nested tuple schema in form of a tree
  - a set of pages where the instances of the tree nodes have been marked up
- Output:
  - use the marked-up pages to learn the rules for the tree nodes
  - for each leaf node: learn a start rule and an end rule
  - for each internal node, e.g., `list(address)`, learn a start rule and an end rule to extract the entire list
- We now illustrate the learning process by considering learning a start rule for a leaf node

# Learning a start rule for area code



- Use a learning technique called sequential covering
  - 1<sup>st</sup> iteration: find a rule that covers a subset of training examples
    - ❖ e.g.,  $R_1 = \text{SkipTo}( )$ , which covers  $E_2$  and  $E_4$
  - 2<sup>nd</sup> iteration: find a rule that covers a subset of remaining exams
    - ❖ e.g.,  $R_7 = \text{SkipTo}(-<b>)$ , which covers all remaining examples
  - and so on, the final rule is a disjunction of all rules found so far
    - ❖ e.g., Start: either  $\text{SkipTo}( )$  or  $\text{SkipTo}(-<b>)$

# Learning a start rule for area code

- Sequential covering can consider a huge number of rules
- Example: consider these rules during the 2<sup>nd</sup> iteration before selecting the best rule (rule R<sub>7</sub>):

Wildcards: *Anything, Numeric, AlphaNumeric, Alphabetic, Capitalized, AllCaps, HTMLTag, nonHTML, Punctuation*

R<sub>7</sub>: *SkipTo(- <b>)*

R<sub>8</sub>: *SkipTo(Punctuation <b>)*

R<sub>9</sub>: *SkipTo(Anything <b>)*

R<sub>10</sub>: *SkipTo(Venice) SkipTo(<b>)*

R<sub>11</sub>: *SkipTo(</b>) SkipTo(<b>)*

R<sub>12</sub>: *SkipTo(:) SkipTo(<b>)*

R<sub>13</sub>: *SkipTo(-) SkipTo(<b>)*

R<sub>14</sub>: *SkipTo(,) SkipTo(<b>)*

R<sub>15</sub>: *SkipTo(Phone) SkipTo(<b>)*

R<sub>16</sub>: *SkipTo(1) SkipTo(<b>)*

R<sub>17</sub>: *SkipTo(Numeric) SkipTo(<b>)*

R<sub>18</sub>: *SkipTo(Punctuation) SkipTo(<b>)*

R<sub>19</sub>: *SkipTo(HTMLTag) SkipTo(<b>)*

R<sub>20</sub>: *SkipTo(AlphaNum) SkipTo(<b>)*

R<sub>21</sub>: *SkipTo(Alphabetic) SkipTo(<b>)*

R<sub>22</sub>: *SkipTo(Capitalized) SkipTo(<b>)*

R<sub>23</sub>: *SkipTo(NonHTML) SkipTo(<b>)*

R<sub>24</sub>: *SkipTo(Anything) SkipTo(<b>)*

# Discussion

- The wrapper model of Stalker subsumes that of HLRT
  - nested tuple schemas are more general than flat tuple schemas
- Both can be viewed as modeling finite state automata
- Both illustrate how imposing structure on the target schema language makes learning practical
  - structure can be simple as flat tuple schema, or more complex, as nested tuple schemas
  - significantly restrict target language, and transform general learning into a far easier problem of learning a relatively small set of parameters: delimiting strings or extraction rules
- Even with such restricted problem settings, learning is still very difficult: large search space, use of heuristics

# Outline

---

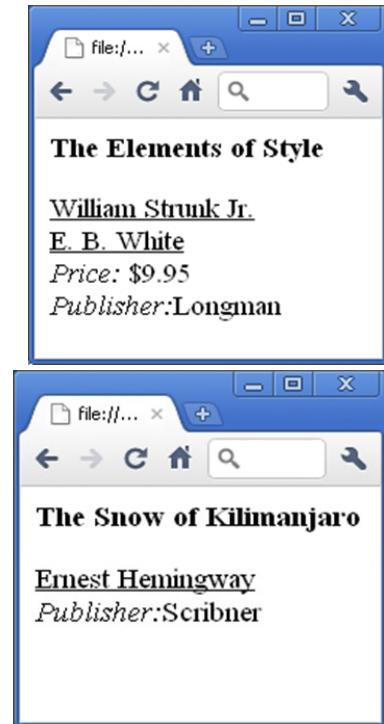
- Problem definition
- Manual wrapper construction
- Learning-based wrapper construction
- Wrapper learning without schema
  - a.k.a., automatic approaches
- Interactive wrapper construction

# Wrapper learning without schema

- Also called automatic approaches to wrapper learning
  - input a set of Web pages of source  $S$
  - examine similarities and dissimilarities across pages
  - automatically infer schema  $T_S$  of pages and extraction program  $E_W$  that extracts data conforming to  $T_S$

AB+C?D

```
<HTML>#PCDATA<P>(<U>#PCDATA</U>
)+
(<I>Price:</I>#PCDATA
)?<I>Publisher:</I>#PCDATA
</HTML>
```



|                         |                                   |        |          |
|-------------------------|-----------------------------------|--------|----------|
| The Elements of Style   | William Strunk Jr.<br>E. B. White | \$9.95 | Longman  |
| The Snow of Kilimanjaro | Ernest Hemingway                  |        | Scribner |

# RoadRunner: A representative approach

---

- Web pages of source  $S$  use schema  $T_S$  to display data
- RoadRunner models  $T_S$  as a nested tuple schema
  - allows optionals (e.g., C in AB+C?D)
  - but does not allow disjunctions (would blow up run time)
  - so  $T_S$  here is union-free regular expressions
- Roadrunner models extraction program  $E_W$  as a regex that when evaluated on a Web page will extract attributes of  $T_S$ 
  - e.g.,  
`<HTML><B>#PCDATA</B><P>(<U>#PCDATA</U><BR>)+<I>Price:</I>#PCDATA<BR>)?<I>Publisher:</I>#PCDATA<BR></HTML>`
  - #PCDATA are slots for values, which can't contain HTML tags

# Inferring Schema $T_S$ and Program $E_W$

---

- Given set of Web pages  $P = \{p_1, \dots, p_n\}$ , examine  $P$  to infer  $E_W$ , then infer  $T_S$  from  $E_W$
- To infer  $E_W$ , iterate
  - initializing  $E_W$  to page  $p_1$  (which can be viewed as a regex)
  - generalize  $E_W$  to also match  $p_2$ , and so on
  - return an  $E_W$  that has been generalized (minimally) to match all pages in  $P$
- Generalization step is the key, which we discuss next

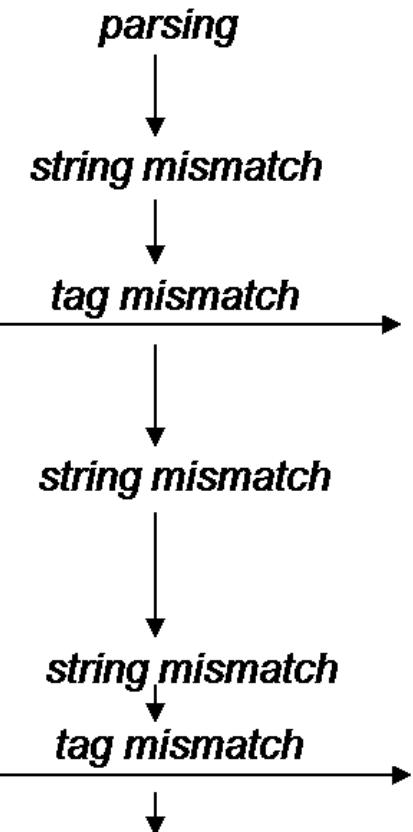
# The Generalization Step

---

- Assume E\_W has been initialized to page p\_1
- Now generalize to match page p\_2
- Tokenize pages into tokens (string or HTML tag)
- Compare two pages, starting from the first token
- Eventually, will likely to run into a mismatch (of tokens)
  - string mismatch: “Database” vs. “Data Integration”
  - tag mismatch: 2 tags, or 1 tag and 1 string
    - ❖ e.g., <UL> vs. <IMG ...>
  - resolving a string mismatch is not too hard, resolving a tag mismatch is far more difficult

### Wrapper (initially page $p_1$ ):

```
01: <HTML>
02: Books on the Topic:
03:
04: Database
05:
06:
|_
07:
08-10: <|>Title:</|>
11: Introduction to Databases
12:
13:
14-16: <|>Title:</|>
17: Relational Databases
18:
19:
20: </HTML>
```



### Target page (page $p_2$ ):

```
01: <HTML>
02: Books on the Topic:
03:
04: Data Integration
05:
06:
07:
08:
09-11: <|>Title:</|>
12: Integrating XML
13:
14:
15-17: <|>Title:</|>
18: Data Integration
19:
20:
21-23: <|>Title:</|>
24: Information Fusion
25:
26:
27: </HTML>
```

### The wrapper after matching page $p_2$ :

```
<HTML>Books on the Topic:#PCDATA
()?

 (<|>Title:</|>#PCDATA)+
</HTML>
```

# Handling Tag Mismatch

---

- Due to either an iterator or an optional
  - <UL> vs. <IMG src=.../> is due to an optional image on  $p_2$
  - </UL> on line 19 of  $p_1$  vs. <LI> on line 20 of  $p_2$  is due to an iterator (2 books in  $p_1$  vs. 3 books in  $p_2$ )
- When tag mismatch happens
  - try to find if it's due to an iterator
  - if yes, generalize  $E_W$  to incorporate iterator
  - otherwise generalize  $E_W$  to incorporate optional
  - there is a reason why we look for iterator before looking for optional
    - ❖ if we don't do so, everything will be thought of as optional, and be generalized accordingly

# Handling Tag Mismatch

---

- Generalize  $E_w$  to incorporate an optional
  - detect which page includes the optional
    - ❖ in the running example, `<IMG src=.../>` is the optional string
  - generalize  $E_w$  accordingly
    - ❖ e.g., introducing the pattern `(<IMG src=.../>)?`
- Generalize  $E_w$  to incorporate an iterator
  - an iterator repeats a pattern, which we call a square
    - ❖ e.g., each book description is a square
  - find the squares, use them to find the lists, then generalize  $E_w$

# Handling Tag Mismatch

---

- Resolving an iterator mismatch often involves recursion
  - while resolving an outer mismatch, may run into an inner mismatch
  - mismatches must be resolved from inside out, recursively

```

 Information Fusion
 David Smith
 Jane Lee

```

```

 Data Integration
 James Madison

```

# Summary

- To generalize  $E_w$  to match a page  $p$ 
  - must detect and resolve all mismatches
  - for each mismatch, must decide if it is a string mismatch, iterator mismatch, or optional mismatch
  - for an iterator or optional mismatch, can search on either the side of  $E_w$  (e.g., page  $p_1$ ) or the side of the target page  $p$ 
    - ❖ e.g., for optional mismatch, the optional can be on either  $E_w$  or  $p$
  - for an iterator or optional mismatch, even when we limit the search to just one side, there are often many square candidates and optional candidates to consider
  - to resolve an iterator mismatch, it may be necessary to recursively resolve many inner mismatches first

# Reducing Runtime Complexity

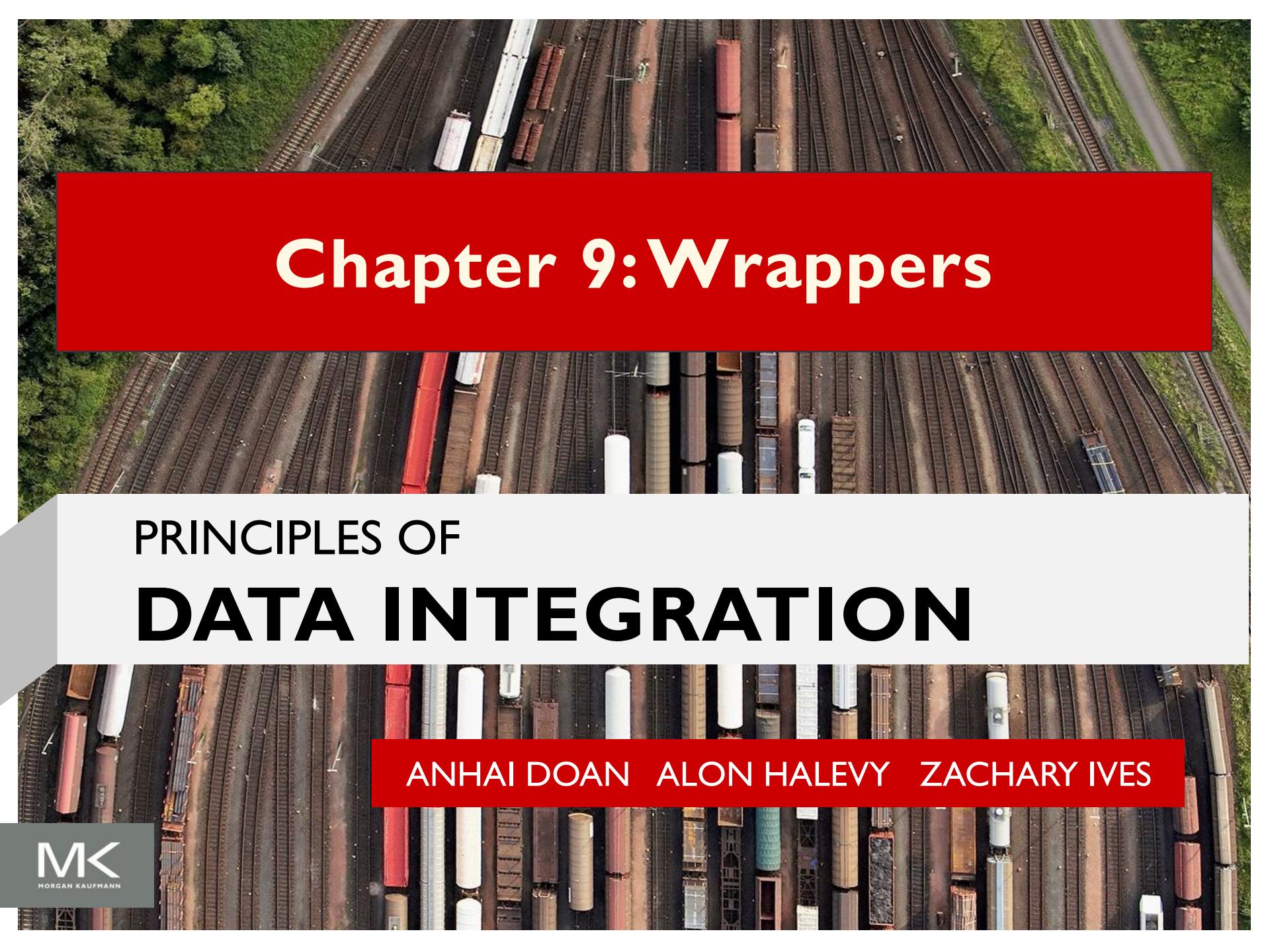
---

- From summary, it is clear that the search space is vast
  - multiple options at each decision point
  - when dead end, must backtrack to the closest decision point and try another option
  - the generalization algorithm incurs exponential time in the length of the inputs
- RoadRunner uses three heuristics to reduce runtime
  - limits # of options at each decision point, consider only top k
  - does not allow backtracking at certain decision points
  - ignores certain iterator/optional patterns judged to be highly unlikely

# Summary

---

- Critical problem in data integration
  - where many sources produce HTML data
- Huge amount of literature
- Remains very difficult
- Common approaches
  - Manual wrapper construction
  - Learning-based wrapper construction
  - Wrapper learning without schema
    - ❖ a.k.a., automatic approaches
  - Interactive wrapper construction

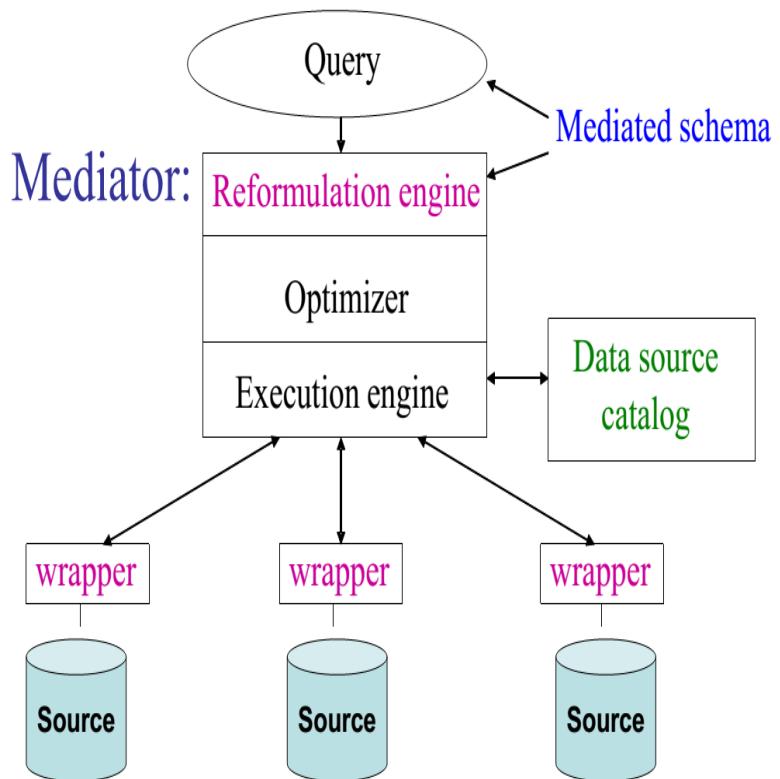


# Chapter 9: Wrappers

PRINCIPLES OF  
**DATA INTEGRATION**

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Introduction



| Week                    | Topic                                                         |
|-------------------------|---------------------------------------------------------------|
| Tues. 03.11.2020        | Ch. 1: Introduction                                           |
| Tues. 10.11.2020        | Ch. 2: Query expression                                       |
| Tues. 17.11             | Ch. 3: Data sources description                               |
| Tues. 24.11             | Ch. 4: String matching                                        |
| Tues. 01.12             | Ch. 5: Schema matching & mapping                              |
| Tues. 08.12             | Ch. 7: Data matching ( <b>Project discussion</b> )            |
| Tues. 15.12             | Ch. 8: Query processing - I                                   |
| <del>Tues. 22.12</del>  | <del>Ch. 8: Query processing - II</del>                       |
| <b>24.12-02.01.2019</b> | <b>Charismas break</b>                                        |
| Tues. 05.01.2021        | Ch. 9: Wrapper                                                |
| Tues. 12.01.2021        | <del>Ch. 10: XML</del> : Ch. 9: Wrapper                       |
| Tues. 19.01             | <del>Ch. 11: Uncertainty</del> : Ch. 10: XML                  |
| Tues. 26.01             | Ch. 11: Ontology                                              |
| Tues. 02.02             | <del>Ch. 12: Web data integration</del> : Ch. 11: Uncertainty |
| Tues. 09.02             | <del>Ch. 13: Semantic web services (Prof. König-Ries)</del>   |

# Outline

---

- Problem definition
- Manual wrapper construction
- Learning-based wrapper construction
- Wrapper learning without schema
  - a.k.a., automatic approaches
- **Interactive wrapper construction**

# Motivation

---

- Limitations of learning and automatic approaches
  - use heuristics to reduce search time in huge space of cands
  - such heuristics are not perfect, so approaches are brittle
    - ❖ we have no idea when they produce correct wrappers
  - even with heuristics, still takes way too long to search
- Interactive approaches address these problems
  - start with little or no user input, search until uncertainty arises
  - ask user for feedback, then resume searching
  - repeat until converging to a wrapper that user likes

# Motivation

---

- User feedback can take many forms
  - label new pages, identify correct extraction result, visually create extraction rules, answer questions posed by system, identify page patterns, etc.
- Key challenges
  - decide when to solicit feedback
  - what feedback to solicit
- Will describe three representative systems
  - interactive labeling of pages with Stalker
  - Identifying correct extraction results with Poly
  - Creating extraction rules with Lixto

# Interactive Labeling of Pages with Stalker

---

- Modify Stalker so that it ask user to label pages during the search process (not “before”, as discussed so far)
  - asks user to label a page (or a few)
  - uses this page to build an initial wrapper
  - interleaves search with soliciting user feedback until finding a satisfactory wrapper
- How to find which page to ask user to label next?
  - maintain two candidate wrappers
  - find pages on which they disagree
  - ask user to label one of these problematic pages
  - this is a form of active learning called co-testing

# Detailed Algorithm

## 1. User labels one or several Web pages

Name:<i>Savory</i><p>Phone:<i> (608) 263-4567 </i><p>Fax:(608) 523-4917

## 2. Learn two wrappers

- e.g., learning to mark the start of a phone number: we can learn a forward rule as well as a backward rule
  - ❖ forward rule       $R_1$ : SkipTo(Phone:<i>)
  - ❖ backward rule       $R_2$ : BackTo(Fax), BackTo()

## 3. Apply learned wrappers to find a problematic page

- apply them to a large set of unlabeled pages
- if they disagree in extraction results on a page → problematic

## 4. Ask user to label a problematic page

## 5. Repeat Steps 2-4 until no more problematic pages

# Identifying Correct Extraction Results with Poly

---

- Also uses co-testing, but differs from Stalker
  - maintains multiple cand wrappers instead of just two
  - asks user to identify correct extraction results
  - instead of using string model, uses DOM tree and visual model

## 1. Initialization

- assumes multiple tuples per page, assume user wants to extract a subset of these tuples
- thus, asks user to label a target tuple on a page by highlighting the attributes of the tuple
  - ❖ e.g., extracting all tuples (title, price, rating) with rating 4 in Table Books
  - ❖ user highlights the first tuple (a, 7, 4) in Table Books

# Example

The screenshot shows a software window with a blue header bar containing standard window controls (minimize, maximize, close) and application-specific icons (back, forward, search, refresh). Below the header is a toolbar with similar icons.

**Books**

| title | price | rating |
|-------|-------|--------|
| a     | 7     | 4      |
| b     | 9     | 4      |
| c     | 12    | 3      |

**DVDs**

| title | price | rating |
|-------|-------|--------|
| d     | 13    | 3      |
| e     | 10    | 2      |

The screenshot shows a software window with a blue header bar containing standard window controls (minimize, maximize, close) and application-specific icons (back, forward, search, refresh). Below the header is a toolbar with similar icons.

**Books**

| title | price | rating |
|-------|-------|--------|
| k     | 11    | 4      |
| l     | 6     | 3      |

**DVDs**

| title | price | rating |
|-------|-------|--------|
| m     | 15    | 4      |
| n     | 7     | 2      |

The screenshot shows a software window with a blue header bar containing standard window controls (minimize, maximize, close) and application-specific icons (back, forward, search, refresh). Below the header is a toolbar with similar icons.

**Special Offers**

| title | price | rating |
|-------|-------|--------|
| t     | 5     | 2      |
| u     | 7     | 3      |

**Books**

| title | price | rating |
|-------|-------|--------|
| v     | 17    | 4      |
| x     | 8     | 3      |

# Identifying Correct Extraction Results with Poly

---

## 2. Using the labeled tuple to generate multiple wrappers

- generate multiple wrappers, each extracts from current page a set of tuples that contain the highlighted tuple
- example wrappers
  - ❖ extracts all book and DVD tuples, just book tuples, book and DVD tuples with rating 4, just book tuples with rating 4, the first tuple of all tables, just the first tuple of the first table
  - ❖ all of these wrappers extract the highlighted tuple (a, 7, 4)

## 3. Soliciting the correct extraction result

- shows user extraction result produced by cand wrappers on the page, and asks user to identify the correct result
- remove all cand wrappers that do not produce that result

# Identifying Correct Extraction Results with Poly

## 3. Soliciting the correct extraction result (cont.)

- example

- ❖ user wants all books with rating 4, so identifies the set  $\{(a, 7, 4), (b, 9, 4)\}$  as correct
- ❖ this removes several wrappers, still leaves those that extract all book and DVD tuples with rating 4, book tuples with rating 4, and all tuples with rating 4 from the first table
- ❖ all of the remaining wrappers still produce correct results on the highlighted page; this page is no longer useful

The screenshot shows a software window with a blue header bar containing standard icons like minimize, maximize, and close. Below the header is a toolbar with icons for back, forward, search, and other navigation functions. The main area is divided into two sections: 'Books' and 'DVDs'. Each section contains a table with three columns: title, price, and rating.

| title | price | rating |
|-------|-------|--------|
| a     | 7     | 4      |
| b     | 9     | 4      |
| c     | 12    | 3      |

| title | price | rating |
|-------|-------|--------|
| d     | 13    | 3      |
| e     | 10    | 2      |

# Identifying Correct Extraction Results with Poly

## 4. Evaluating the remaining wrappers on verification pages

- applies all remaining wrappers to a large set of unlabeled pages to see if wrappers disagree
  - ❖ e.g., “extract all book and DVD tuples with rating 4” and “extract all book tuples with rating 4” disagree on the first page here
  - ❖ “extract book tuples with rating 4” and “extract all tuples with rating 4 in the first table” disagree on the second page
- if finds a disagreement on a page q, repeat Steps 3-4: asks user to select the correct result on q, etc.

The image shows two separate browser windows side-by-side. Both windows have a blue header bar with standard browser controls (back, forward, search, etc.).

**Left Window (Verification Page 1):**

- Books:** A table with columns title, price, rating. Rows contain (k, 11, 4) and (1, 6, 3).
- DVDs:** A table with columns title, price, rating. Rows contain (m, 15, 4) and (n, 7, 2).

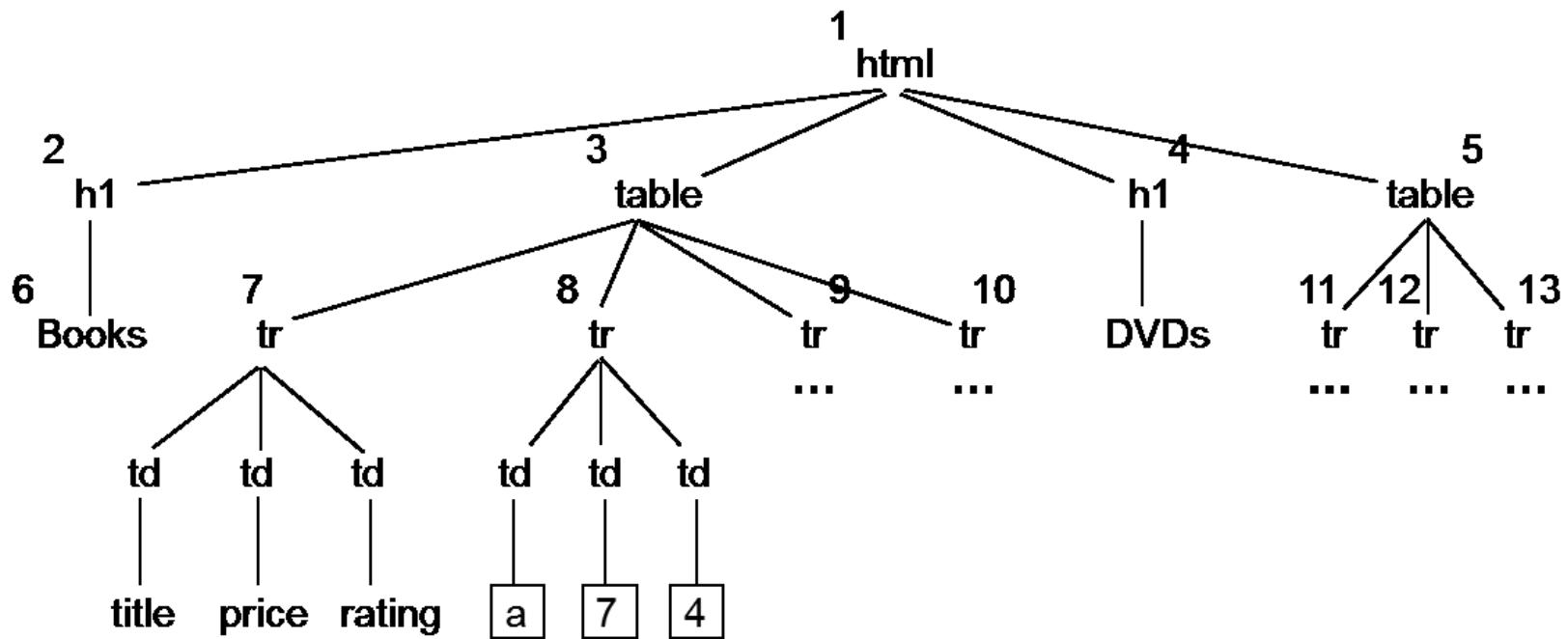
**Right Window (Verification Page 2):**

- Special Offers:** A table with columns title, price, rating. Rows contain (t, 5, 2) and (u, 7, 3).
- Books:** A table with columns title, price, rating. Rows contain (v, 17, 4) and (x, 8, 3).

## 5. Return all cand wrappers when they no longer disagree on unlabeled pages

# Generating the Wrappers in Poly

- Convert page into a DOM tree
- Identifies nodes that map to highlighted attributes
- Create XPath-like expressions from root to these nodes
- ~~Screenshots for details~~



# Creating Extraction Rules with Lixto

- Lixto vs. Poly and Stalker
  - user visually create extraction rules using highlighting and dialog boxes
    - ❖ instead of labeling pages or identifying extraction results
  - encodes extraction rules internally using a Datalog-like language, defined over DOM tree and string models of pages
- Creating the extraction rules visually
  - Web page lists books being auctioned
  - user can create 4 rules
    - ❖ Rule 1 extracts books themselves
    - ❖ Rules 2-4 extract title/price/# of bids of each book, respectively

The screenshot shows a web browser window with a blue header bar. The title bar says 'Database Books'. Below the header is a toolbar with icons for back, forward, search, and other functions. The main content area displays a table with the following data:

| Title                            | Price   | Bids              |
|----------------------------------|---------|-------------------|
| <a href="#">Databases</a>        | \$15.00 | <a href="#">1</a> |
| <a href="#">Data Mining</a>      | \$13.99 | <a href="#">3</a> |
| <a href="#">Data Integration</a> | \$21.99 | <a href="#">2</a> |

# Creating the Extraction Rules with Lixto

- To create Rule 1, which extracts books
  - user highlights a book tuple
    - ❖ e.g., the first one “Databases”
  - Lixto maps this tuple to corresponding subtree of the DOM tree of page, extrapolates to create Rule 1, shows result of Rule 1 on the page
  - user accepts Rule 1
    - ❖ can also refine the rule

| Title                            | Price   | Bids |
|----------------------------------|---------|------|
| <a href="#">Databases</a>        | \$15.00 | 1    |
| <a href="#">Data Mining</a>      | \$13.99 | 3    |
| <a href="#">Data Integration</a> | \$21.99 | 2    |

# Creating the Extraction Rules with Lixto

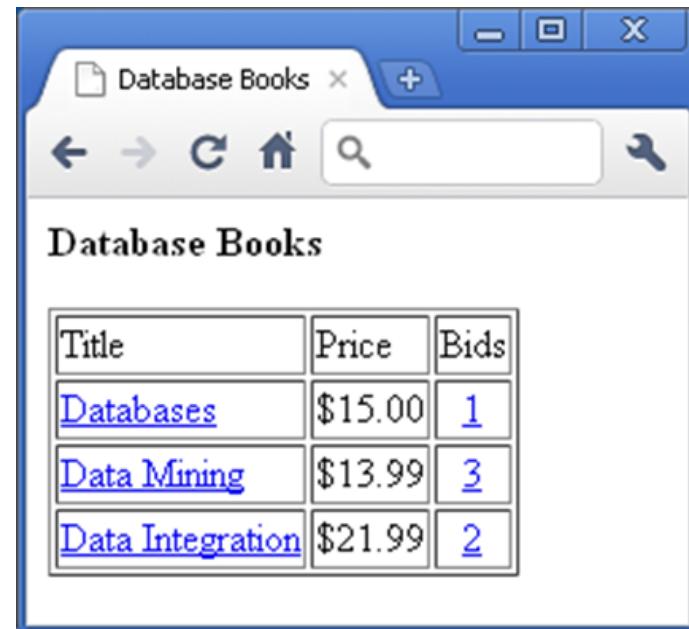
- To create Rule 2, which extracts titles
  - user specifies that this rule will extract from the book instances identified by Rule 1
  - user highlights a title
  - Lixto uses this to create a rule and shows user all extraction results of this rule
  - user realizes this rule is too general (e.g., extracting both titles and bids), so user refines the rule using dialog boxes

| Title            | Price   | Bids |
|------------------|---------|------|
| Databases        | \$15.00 | 1    |
| Data Mining      | \$13.99 | 3    |
| Data Integration | \$21.99 | 2    |

# Creating the Extraction Rules with Lixto

- What can user do?

- highlighting a tuple or a value
- using dialog boxes to restrict or relax a rule
- write regular expressions
- refers to real-world concepts defined by Lixto

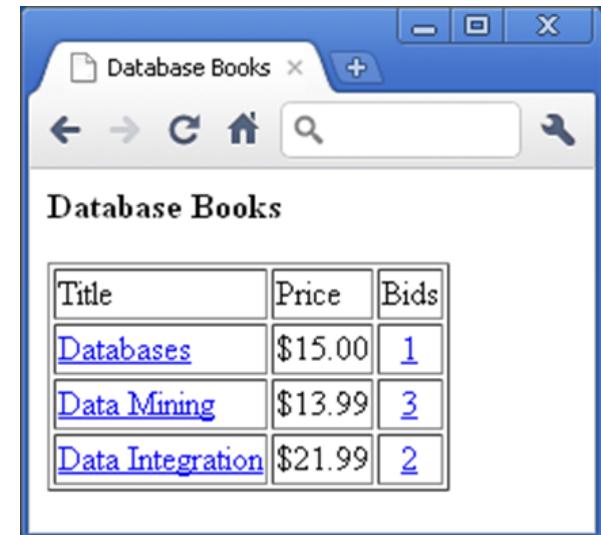


The screenshot shows a software application window titled "Database Books". The window has a toolbar with icons for back, forward, search, and other functions. Below the toolbar is a header bar with the title "Database Books". The main area displays a table with three columns: "Title", "Price", and "Bids". There are four rows of data:

| Title                            | Price   | Bids |
|----------------------------------|---------|------|
| <a href="#">Databases</a>        | \$15.00 | 1    |
| <a href="#">Data Mining</a>      | \$13.99 | 3    |
| <a href="#">Data Integration</a> | \$21.99 | 2    |

# Representing the Extraction Rules

- Using a Datalog-like internal language
- User is not aware of this language
- Example of the four rules discussed so far



The screenshot shows a web browser window with a title bar 'Database Books'. Below the title bar is a toolbar with icons for back, forward, search, and refresh. The main content area displays a table with three columns: 'Title', 'Price', and 'Bids'. The table contains four rows of data:

| Title            | Price   | Bids |
|------------------|---------|------|
| Databases        | \$15.00 | 1    |
| Data Mining      | \$13.99 | 3    |
| Data Integration | \$21.99 | 2    |

$R_1: book(S,X) \leftarrow page(S), subelem(S,.table,X)$

$R_2: title(S,X) \leftarrow book(-,S), subelem(S,(*.td.*.content,[(href,,substr)])),X), notbefore(S,X,.td,100)$

$R_3: price(S,X) \leftarrow book(-,S), subelem(S,(*.td,[elementtext,\var[Y].*,regvar])),X), isCurrency(Y)$

$R_4: bids(S,X) \leftarrow book(-,S), subelem(S,*.td,X), before(S,X,.td,0,30,Y,-), price(-,Y)$

# Summary

---

- Critical problem in data integration
  - where many sources produce HTML data
- Huge amount of literature
- Remains very difficult
- Common approaches
  - Manual wrapper construction
  - Learning-based wrapper construction
  - Wrapper learning without schema
    - ❖ a.k.a., automatic approaches
  - Interactive wrapper construction

# CHAPTER II: XML

## PRINCIPLES OF DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

# Gaining access to diverse data

We have focused on data integration  
in the **relational model**

Simplest model to understand

| A              | B              |
|----------------|----------------|
| a <sub>1</sub> | b <sub>1</sub> |
| a <sub>2</sub> | b <sub>2</sub> |

Real-world data is often not in relational form

e.g., Excel spreadsheets, Web tables, Java objects, RDF, ...

- One approach: convert using custom wrappers (Ch. 9)
- But suppose tools would adopt a **standard** export (and import) mechanism?
  - ... This is the role of XML, the eXtensible Markup Language



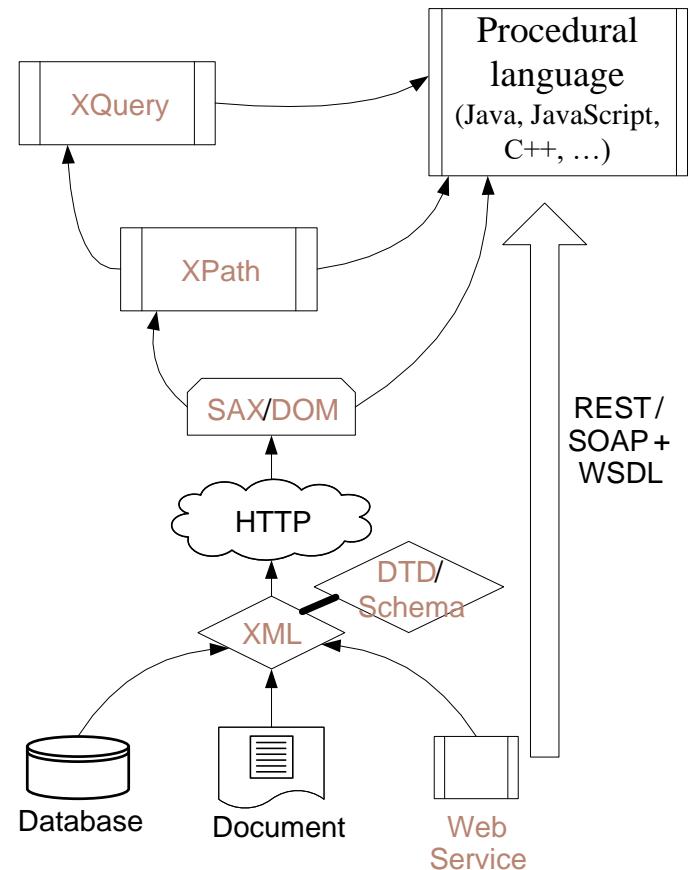
# What is XML?

Hierarchical, human-readable format

- A “sibling” to HTML, **always parseable**
- “Lingua franca” of data: encodes documents and structured data
- Blends data and schema (structure)

Core of a broader ecosystem

- Data – **XML** (also RDF, Ch. 12)
- Schema – **DTD** and **XML Schema**
- Programmatic access – **DOM** and **SAX**
- Query – **XPath**, **XSLT**, **XQuery**
- Distributed programs – **Web services**



# XML anatomy

```
<?xml version="1.0" encoding="ISO-8859-1" ?> ← Processing Instr.
<dblp> ← Open-tag
 <mastersthesis mdate="2002-01-03" key="ms/Brown92">
 <author>Kurt P. Brown</author>
 <title>PRPL: A Database Workload Specification Language</title>
 <year>1992</year> ← Element
 <school>Univ. of Wisconsin-Madison</school>
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
 <editor>Paul R. McJones</editor> ← Attribute
 <title>The 1995 SQL Reunion</title>
 <journal>Digital System Research Center Report</journal>
 <volume>SRC1997-018</volume> ← Close-tag
 <year>1997</year>
 <ee>db/labs/dec/SRC1997-018.html</ee>
 <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

# XML data components

XML includes two kinds of data items:

Elements

```
<article mdate="2002-01-03" ...>
 <editor>Paul R. McJones</editor> ...
</article>
```

- ❖ Hierarchical structure with open tag-close tag pairs
- ❖ May include nested elements
- ❖ May include attributes within the element's open-tag
- ❖ Multiple elements may have same name
- ❖ Order matters

Attributes

```
mdate="2002-01-03"
```

- ❖ Named values – not hierarchical
- ❖ Only one attribute with a given name per element
- ❖ Order does NOT matter

# Well-formed XML: Always parseable

Any legal XML document is always parseable by an XML parser, without knowledge of tag meaning

- The start – *preamble* – tells XML about the char. encoding  
`<?xml version="1.0" encoding="utf-8"?>`
- There's a single root element
- All open-tags have matching close-tags (unlike many HTML documents!), or a special:  
`<tag/>` shortcut for empty tags (equivalent to `<tag></tag>`)
- Attributes only appear once in an element
- XML is case-sensitive

# Outline

---

## ➤ XML data model

- Node types
- Encoding relations and semi-structured data
- Namespaces
- XML schema languages
- XML querying
- XML query processing
- XML schema mapping

# XML as a data model

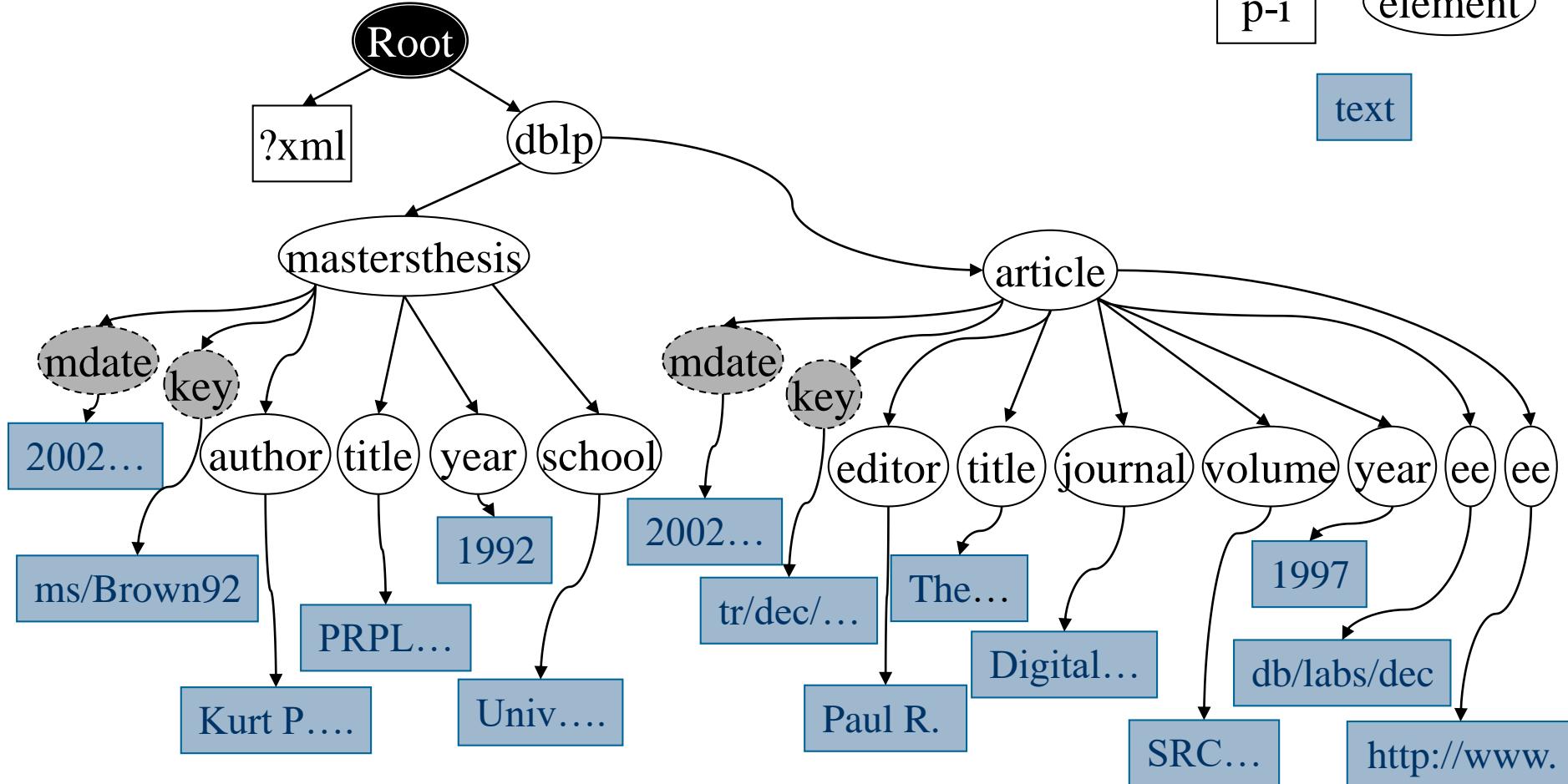
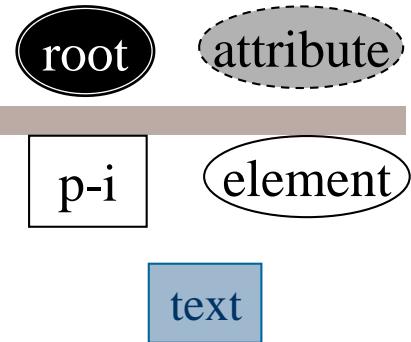
---

XML “information set” includes 7 types of nodes:

- Document (root)
- Element
- Attribute
- Processing instruction
- **Text (content)**
- Namespace
- Comment

XML data model includes this, plus typing info, plus order info and a few other things

# XML data model visualized



# XML easily encodes relations

Student-course-grade

| sid | cid    | exp-grade |
|-----|--------|-----------|
| 1   | 570103 | B         |
| 23  | 550103 | A         |

```
<student-course-grade>
 <tuple><sid>1</sid><cid>570103</cid>
 <exp-grade>B</exp-grade></tuple>
 <tuple><sid>23</sid><cid>550103</cid>
 <exp-grade>A</exp-grade></tuple>
</student-course-grade>
```

*OR*

---

```
<student-course-grade>
 <tuple sid="1" cid="570103" exp-grade="B"/>
 <tuple sid="23" cid="550103" exp-grade="A"/>
</student-course-grade>
```

# XML is “Semi-Structured”

```
<parents>
 <parent name="Jean">
 <son>John</son>
 <daughter>Joan</daughter>
 <daughter>Jill</daughter>
 </parent>
 <parent name="Feng">
 <daughter>Ella</daughter>
 </parent>
...
```

# Combining XML from multiple sources with the same tags: Namespaces

- *Namespaces* allow us to specify a context for different tags
- Two parts:
  - Binding of namespace to URI
  - Qualified names

The diagram illustrates the XML code with annotations:

- An annotation *Default namespace for non-qualified names* points to the first `xmlns` declaration in the root tag: `<root xmlns="http://www.first.com/aspace" ...>`.
- An annotation *Defines "otherns" qualifier* points to the second `xmlns` declaration in the root tag: `<root ... xmlns:otherns="...">`.
- The XML code itself shows:
  - The root tag has two `xmlns` declarations: one for the default namespace (`http://www.first.com/aspace`) and one for the `otherns` namespace (`...otherns="..."`).
  - A child tag `<myns:tag ...>` uses the `myns` namespace (`http://www.fictitious.com/mypath`).
  - Content within the `myns:tag` includes:
    - A tag `<thistag>` is in the default namespace (`http://www.first.com/aspace`).
    - A tag `<myns:thistag>` is in the `myns` namespace.
    - A tag `<otherns:thistag>` is in the `otherns` namespace.

# Outline

---

- ✓ XML data model
- XML schema languages
  - DTDs
  - XML Schema (XSD)
- XML querying
- XML query processing
- XML schema mapping

# XML Isn't enough on its own

---

It's too unconstrained for many cases!

- How will we know when we're getting garbage?
- How will we know what to query for?
- How will we understand what we received?

We also need:

- An idea of (at least part of) the structure
- Some knowledge of how to interpret the tags...

# Document type definitions (DTDs)

The DTD is an EBNF<sup>1</sup> grammar defining XML structure

- The XML document specifies an associated DTD, plus the root element of the document
- DTD specifies children of the root (and so on)

DTD also defines special attribute types:

- IDs – special attributes that are analogous to keys for elements
- IDREFs – references to IDs
- IDREFS – a list of IDREFs, space-delimited (!)
- All other attributes are essentially treated as strings

# An example DTD and how to reference it from XML

Example DTD:

```
<!ELEMENT dblp((mastersthesis | article)*)
<!ELEMENT
 mastersthesis(author,title,year,school,committeemember*)
<!ATTLIST mastersthesis(mdate CDATA #REQUIRED
 key ID #REQUIRED
 advisor CDATA #IMPLIED)
<!ELEMENT author(#PCDATA)>
...
...
```

Example use of DTD in XML file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE dblp SYSTEM "my.dtd">
<dblp>...
```

# Links in XML: Restricted foreign keys

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE graph SYSTEM "special.dtd">
<graph>
 <author id="author1">
 <name>John Smith</name>
 </author>
 <article>
 <author ref="author1" /> <title>Paper1</title>
 </article>
 <article>
 <author ref="author1" /> <title>Paper2</title>
 </article>
 ...

```

*Suppose we have defined  
this to be of type ID*

*Suppose we have defined  
this to be of type IDREF*

# Limitations of DTDs

---

DTDs capture grammatical structure, but have some drawbacks:

- Don't capture database datatypes' domains
- IDs aren't a good implementation of keys
  - ❖ Why not?
- No way of defining OO-like inheritance
  
- “Almost XML” syntax – inconvenient to build tools for them

# XML Schema (XSD)

Aims to address the shortcomings of DTDs

- XML syntax
- Can define keys using XPaths (we'll discuss later)
- Type subclassing that also includes restrictions on ranges
  - ❖ “By extension” (adds new data) and “by restriction” (adds constraints)
- ... And, of course, domains and built-in datatypes

# Basics of XML schema

Need to use the XML Schema namespace (generally named xsd)

- **simpleTypes** are a way of restricting domains on scalars
  - Can define a **simpleType** based on integer, with values within a particular range
- **complexTypees** are a way of defining element/attribute structures
  - Basically equivalent to **!ELEMENT**, but more powerful
  - Specify sequence, choice between child elements
  - Specify **minOccurs** and **maxOccurs** (default 1)
- Must associate an **element/attribute** with a **simpleType**, or an **element** with a **complexType**

# Simple XML schema example

*Associates “xsd” namespace with XML Schema*

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="mastersthesis" type="ThesisType"/>
 <xsd:complexType name="ThesisType">
 <xsd:attribute name="mdate" type="xsd:date"/>
 <xsd:attribute name="key" type="xsd:string"/>
 <xsd:attribute name="advisor" type="xsd:string"/>
 <xsd:sequence>
 <xsd:element name="author" type="xsd:string"/>
 <xsd:element name="title" type="xsd:string"/>
 <xsd:element name="year" type="xsd:integer"/>
 <xsd:element name="school" type="xsd:string"/>
 <xsd:element name="committeemember" type="CommitteeType"
 minOccurs="0"/>
 </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

*This is the root element, with type specified below*

# Designing an XML Schema/DTD

Not as formalized as relational data design

- Typically based on an existing underlying design, e.g., relational DBMS or spreadsheet

We generally orient the XML tree around the “central” objects

Big decision: element vs. attribute

- Element if it has its own properties, or if you *might* have more than one of them
- Attribute if it is a single property – though element is OK here too!

# Outline

---

- ✓ XML data model
- ✓ XML schema languages
- XML querying
  - DOM and SAX
  - XPath
  - XQuery
- XML query processing
- XML schema mapping

# XML Parser

- A **parser** is a software component that sits between the application and the XML files.
- It reads a **text-formatted XML file** or stream and converts it to a document to be manipulated by the application.
- **Well-formed** documents respect the syntactic rules.
- **Valid** documents not only respect the syntactic rules but also conform to a structure as described in a DTD.
- **Tree-based parsers** map an XML document into an internal tree structure, and then allow an application to navigate that tree.
- **Event-based parsers** report parsing events (such as the start and end of elements) directly to the application through callbacks.

# Document Object Model (DOM) and Simple API for XML (SAX)

- **DOM**: an object-oriented representation of the XML parse tree (roughly like the Data Model graph)
  - **DOM objects** have methods like “getFirstChild()”, “getNextSibling”
  - Common way of traversing the tree
  - Can also **modify** the DOM tree – alter the XML – via `insertAfter()`, etc.
- Sometimes we don't want **all** of the data: **SAX**
  - **Parser interface** that calls a function each time it parses a processing-instruction, element, etc.
  - Your code can determine what to do, e.g., build a data structure, or discard a particular portion of the data

# Querying XML

Alternate approach to processing the data: a *query* language

- Define some sort of a *template* describing traversals from the *root* of the directed graph
- Potential benefits in parallelism, views, schema mappings, and so on
- In XML, the basis of this template is called an XPath
  - ❖ Can also declare some constraints on the values you want
  - ❖ The XPath returns a *node set* of matches

# XPaths

---

In its simplest form, an Xpath looks like a path in a file system:

`/mypath/subpath/*/morepath`

- But XPath returns a *node set* representing the XML nodes (and their subtrees) at the end of the path
- XPaths can have *node tests* at the end, filtering all except node types
  - ❖ `text()`, `processing-instruction()`, `comment()`, `element()`, `attribute()`
- XPath is fundamentally an ordered language: it can query in order-aware fashion, and it returns nodes in order

# Recall our sample XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
 <mastersthesis mdate="2002-01-03" key="ms/Brown92">
 <author>Kurt P. Brown</author>
 <title>PRPL: A Database Workload Specification
Language</title>
 <year>1992</year>
 <school>Univ. of Wisconsin-Madison</school>
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
 <editor>Paul R. McJones</editor>
 <title>The 1995 SQL Reunion</title>
 <journal>Digital System Research Center Report</journal>
 <volume>SRC1997-018</volume>
 <year>1997</year>
 <ee>db/labs/dec/SRC1997-018.html</ee>
 <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

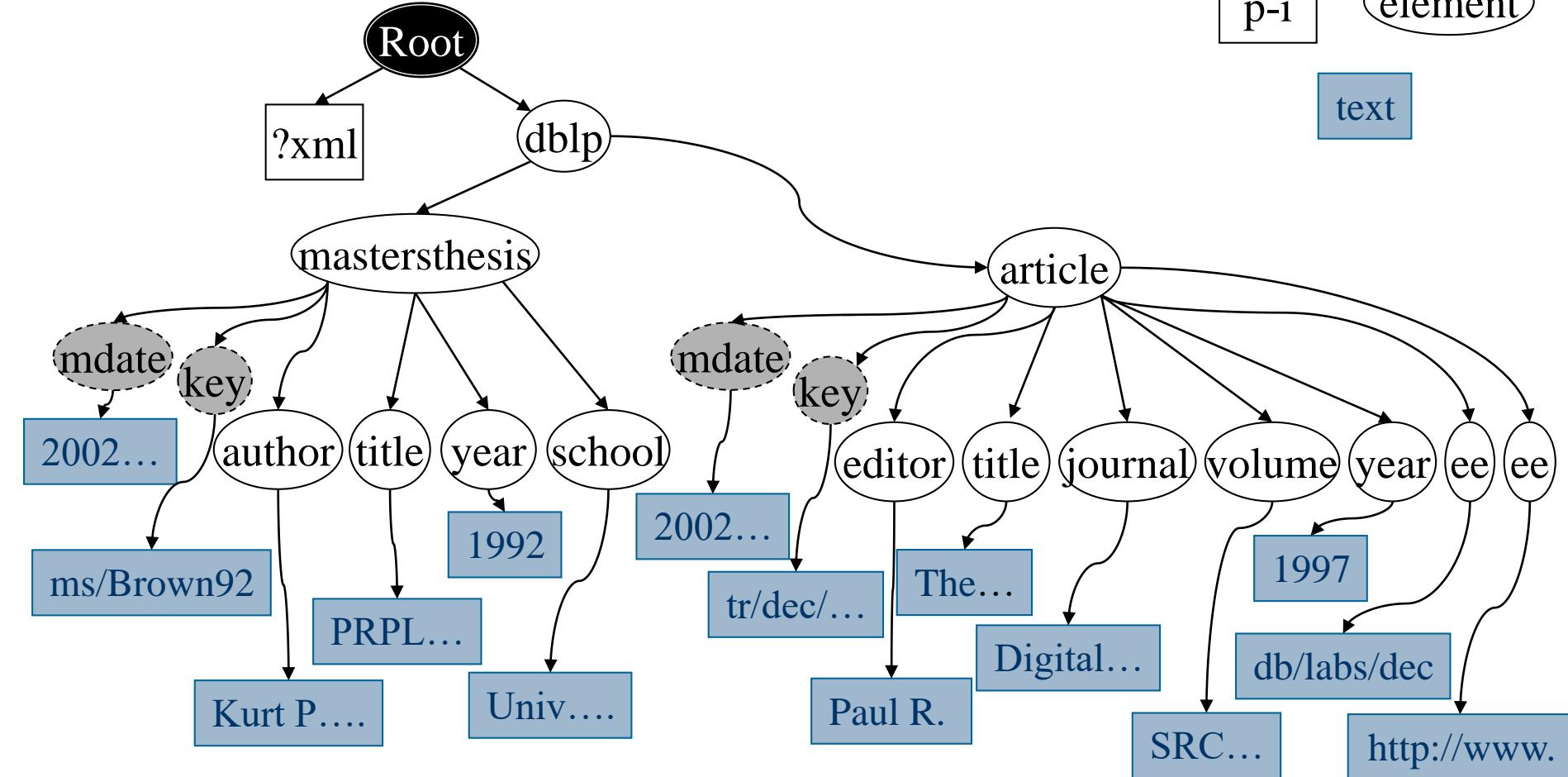
# Recall our XML tree

root  
attribute

p-i

element

text



# XPath Queries: Some examples

---

- `/dblp/mastersthesis/title`
- `/dblp/*/editor`
- `//title`
- `//title/text()`

# Context nodes and relative paths

XPath has a notion of a *context node*: it's analogous to a current directory

- “.” represents this context node
  - “..” represents the parent node
  - We can express relative paths:  
`subpath/sub-subpath/.../...` gets us back to the context node
- By default, the document root is the context node

# Predicates – Selection operations

---

A *predicate* allows us to filter the node set based on selection-like conditions over sub-XPaths:

```
/dblp/article[title = “Paper1”]
```

which is equivalent to:

```
/dblp/article[./title/text() = “Paper1”]
```

# Axes: More complex traversals

Thus far, we've seen XPath expressions that go *down* the tree (and up one step)

- But we might want to go up, left, right, etc. via axes:
    - ❖ `self::path-step`
    - ❖ `child::path-step`
    - ❖ `descendant::path-step`
    - ❖ `descendant-or-self::path-step`
    - ❖ `preceding-sibling::path-step`
    - ❖ `preceding::path-step`
    - ❖ `parent::path-step`
    - ❖ `ancestor::path-step`
    - ❖ `ancestor-or-self::path-step`
    - ❖ `following-sibling::path-step`
    - ❖ `following::path-step`
  - The previous XPaths we saw were in “abbreviated form”  
`/child::dblp/child::mastersthesis/child::title`  
`/descendant-or-self::title`

# Querying order

- We saw in the previous slide that we could query for preceding or following siblings or nodes
- We can also query a **node's position** according to some index:
  - `fn::first()`, `fn::last()` index of 0<sup>th</sup> & last element matching the last step
  - `fn::position()` relative count of the current node  
`child::article[fn::position() = fn::last()]`

# XPath is used within many standards

---

- XML Schema uses simple **XPaths** in defining keys and uniqueness constraints
- XQuery
- XSLT
- XLink and Xpointer – hyperlinks for XML

# XPath is used to express XML schema keys & foreign keys

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="ThesisType">
 <xsd:attribute name="key" type="xsd:string"/>
 <xsd:sequence>
 <xsd:element name="author" type="xsd:string"/> ...
 <xsd:element name="school" type="xsd:string"/> ...
 </xsd:sequence>
</xsd:complexType>
<xsd:element name="dblp"> <xsd:sequence>
 <xsd:element name="mastersthesis" type="ThesisType">
 <xsd:keyref name="schoolRef" refer="schoolId">
 <xsd:selector xpath=".//school"/> <xsd:field xpath=".//text()"/>
 </xsd:keyref> </xsd:element>
 <xsd:element name="university" type="SchoolType">...</xsd:element>
 </xsd:sequence>
 <xsd:key name="schoolId">
 <xsd:selector xpath="university"/><xsd:field xpath="@key"/>
 </xsd:key> </xsd:element> </xsd:schema>
```

*Foreign key refers to key by its ID*

*Item w/key = selector  
Field is its key*

# XQuery

- A XQuery is *the* language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is defined by the W3C
- XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.)
- XQuery is a W3C recommendation (Jan 2007; latest 14 Dec 2010) thus a standard

# XQuery

---

- XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.
- XQuery 1.0 is a *strict superset* of XPath 2.0
- $\forall$  XPath 2.0 expression is directly an XQuery 1.0 expression (a query)
- The extra expressive power is the ability to:
  - *Join* information from different sources and
  - *Generate* new XML fragments

# How to *Select* nodes with XQuery?

## ■ Functions

- XQuery uses functions to extract data from XML documents.
- `doc()`: function to open a file
- **Example:** `doc("books.xml")`

## ■ (X)Path Expressions

- XQuery uses path expressions to navigate through elements in an XML document.
- **Example:** select all the title elements in the "books.xml" file:  
`doc("books.xml")/bookstore/book/title`

## ■ Predicates

- XQuery uses predicates to limit the extracted data from XML documents.
- **Example:** select all the book elements under the bookstore element that have a price element with a value that is less than 30 :  
`doc("books.xml")/bookstore/book[price<30]`

# XQuery's basic form

- Has an analogous form to SQL's  
**SELECT..FROM..WHERE..GROUP BY..ORDER BY**
- The model: bind nodes (or node sets) to variables; operate over each legal combination of bindings; produce a set of nodes
- “**FLWOR**” statement [note case sensitivity!]:  
    **for** {iterators that bind variables}  
    **let** {collections}  
    **where** {conditions}  
    **order by** {order-paths}  
    **return** {output constructor}
- Mixes XML + XQuery syntax; use {} as “escapes”

# XQuery's basic form

- select all the title elements under the book elements that are under the bookstore element that have a price element with a value that is higher than 30.

- Path expression :

```
doc("books.xml") /bookstore/book[price>30]/title
```

- FLWOR expression :

```
for $x in doc("books.xml") /bookstore/book
where $x/price>30
return $x/title
```

# Present the result in an HTML list

```

{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return {$x}
}

```

# Difference between **for** and **let**

```
for $x in (1, 2, 3, 4)
let $y := ("a", "b", "c")
return ($x, $y)
```



```
1, a, b, c, 2, a, b, c, 3, a, b, c, 4, a, b, c
```

```
let $x in (1, 2, 3, 4)
for $y := ("a", "b", "c")
return ($x, $y)
```



```
1, 2, 3, 4, a, 1, 2, 3, 4, b, 1, 2, 3, 4, c
```

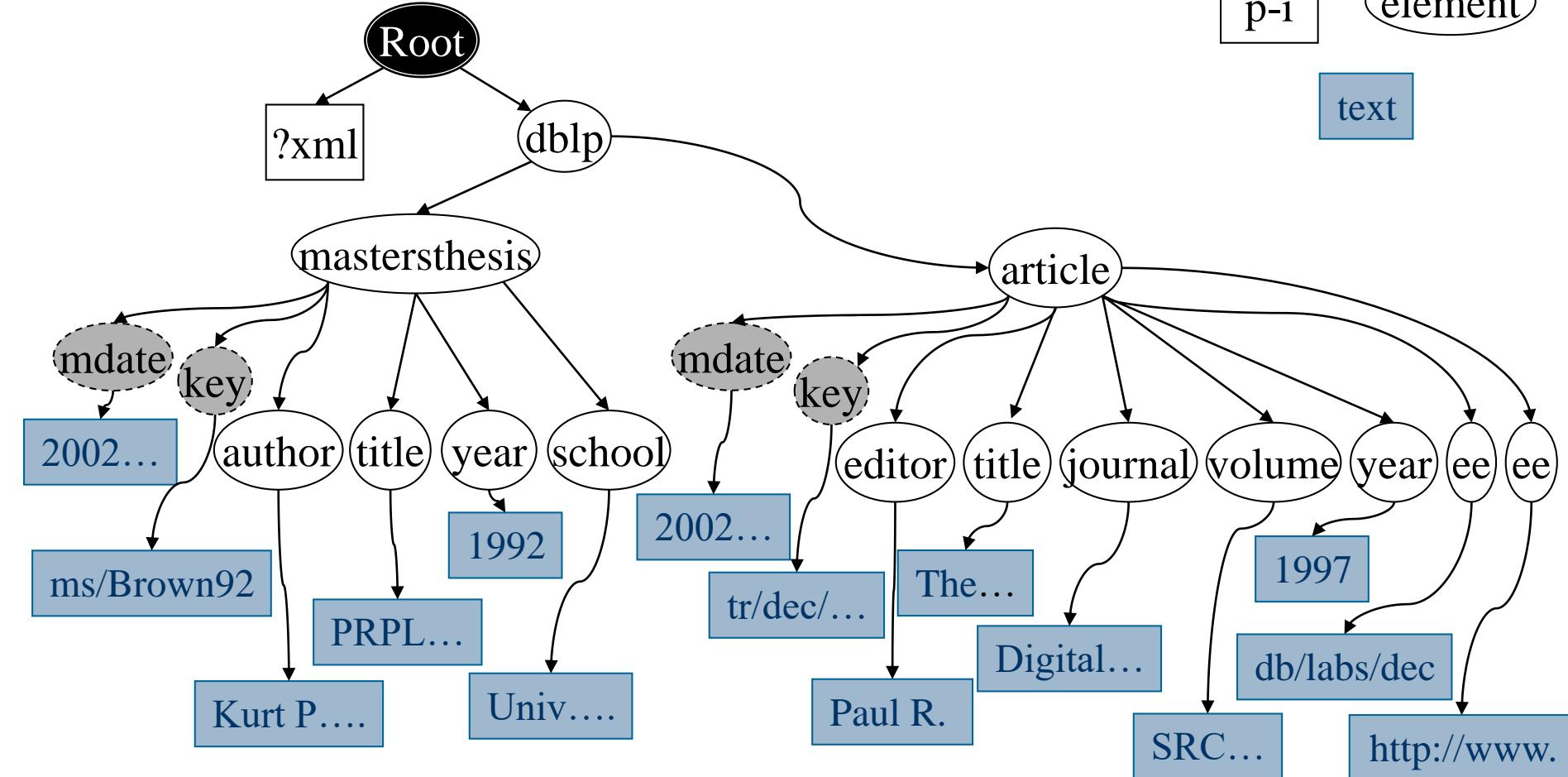
# Recall Our XML Tree

root  
attribute

p-i

element

text



# “Iterations” in XQuery

A series of (possibly nested) FOR statements assigning the results of XPaths to variables

```
for $root in doc ("http://my.org/my.xml")
 for $sub in $root/rootElement,
 $sub2 in $sub/subElement, ...
```

- Something like a template that pattern-matches, produces a “binding tuple”
- For each of these, we evaluate the WHERE and possibly output the RETURN template
- `document()` or `doc()` function specifies an input file as a URI
  - Early versions used “document”; modern versions use “doc”

# Two XQuery examples

```
<root-tag> {
 for $p in doc (“dblp.xml”)/dblp/article,
 $yr in $p/yr
 where $yr = “1997”
 return <paper> { $p/title } </paper>
} </root-tag>
```

---

```
for $i in doc (“dblp.xml”)/dblp/article[author/text() = “John Smith”]
return <smith-paper>
 <title>{ $i/title/text() }</title>
 <key>{ $i/@key }</key>
 { $i/crossref }
</smith-paper>
```

# Restructuring data in XQuery

Nesting XML trees is perhaps the most common operation

In XQuery, it's easy – put a subquery in the `return` clause where you want things to repeat!

```
for $u in doc("dblp.xml")/dblp/university
 where $u/country = "USA"
 return <ms-theses-99>
 { $u/name } {
 for $mt in doc("dblp.xml")/dblp/mastersthesis
 where $mt/year/text() = "1999" and $mt/school = $u/name
 return $mt/title }
 </ms-theses-99>
```

# Collections & aggregation in XQuery

In XQuery, many operations return **collections**

- XPaths, sub-XQueries, functions over these, ...
- The **let** clause assigns the results to a **variable**

Aggregation simply applies a function over a collection, where the function returns a value (very elegant!)

```
let $allpapers := doc ("dblp.xml")/dblp/article
return <article-authors>
 <count> { fn:count(fn:distinct-values($allpapers/authors)) } </count>
{ for $paper in doc("dblp.xml")/dblp/article
 let $pauth := $paper/author
 return <paper> {$paper/title}
 <count> { fn:count($pauth) } </count>
 </paper>
} </article-authors>
```

# Collections, Ctd.

Unlike in SQL, we can compose aggregations and create new collections from old:

```
<result> {
let $avgItemsSold := fn:avg(
 for $order in doc("my.xml")/orders/order
 let $totalSold = fn:sum($order/item/quantity)
 return $totalSold)
 return $avgItemsSold
} </result>
```

# Distinct-ness

In XQuery, DISTINCT-ness happens as a **function over a collection**

- But since we have nodes, we can do duplicate removal according to value or node
- Can do `fn:distinct-values(collection)` to remove duplicate values, or `fn:distinct-nodes(collection)` to remove duplicate nodes

```
for $years in fn:distinct-values(doc("dblp.xml")//year/text())
return $years
```

# Sorting in XQuery

---

- In XQuery, what we order is the sequence of “result tuples” output by the `return` clause:

```
for $x in doc (“dblp.xml”)/proceedings
order by $x/title/text()
return $x
```

# Querying & defining metadata

Can get a node's name by querying `name()`:

```
for $x in doc ("dblp.xml")/dblp/*
return name($x)
```

Can construct elements and attributes using **computed names**:

```
for $x in doc ("dblp.xml")/dblp/*,
 $year in $x/year,
 $title in $x/title/text()
return
element { name($x) } {
 attribute { "year-" + $year } { $title }
}
```

# Views in XQuery

- A view is a **named query**
- We use the name of the view to invoke the query  
(treating it as if it were the relation it returns)

XQuery:

```
declare function V() as element(content)* {
 for $r in doc("R")/root/tree,
 $a in $r/a, $b in $r/b, $c in $r/c
 where $a = "123"
 return <content>{$a, $b, $c}</content>
}
```

*Using the view:*

```
for $v in V()/content,
$r in doc("r")/root/tree
where $v/b = $r/b
return $v
```

# Outline

---

- ✓ XML data model
- ✓ XML schema languages
- ✓ XML querying
- XML query processing
- XML schema mapping

# Streaming Query Evaluation

---

- In data integration scenarios, the query processor must fetch **remote** data, parse the XML, and process
- Ideally: we can **pipeline** processing of the data as it is “streaming” to the system

“Streaming XPath evaluation”

... which is also a building block to pipelined XQuery evaluation...

# Main Observations

---

- XML is sent (serialized) in a form that corresponds to a **left-to-right depth-first traversal** of the parse tree
- The “core” part of XPath (child, descendant axes) essentially corresponds to **regular expressions** over edge labels

# The First Enabler: SAX (Simple API for XML)

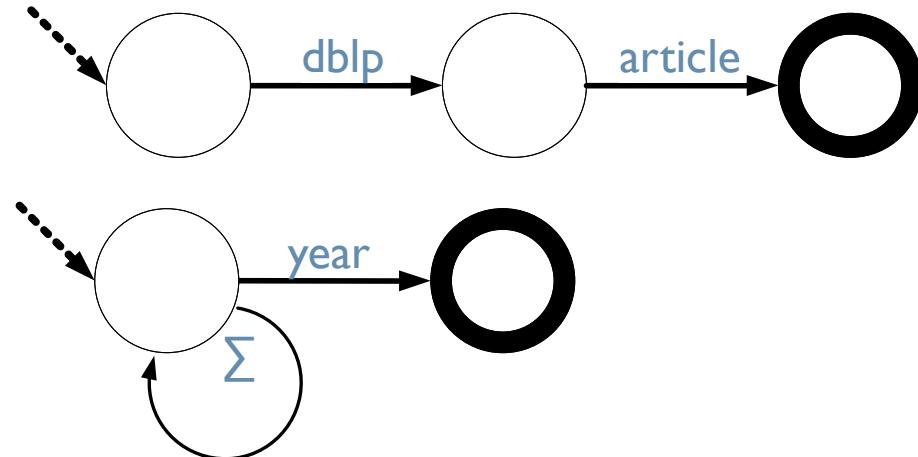
---

- If we are to match XPaths in streaming fashion, we need a stream of XML nodes
- SAX provides a series of event notifications
  - Events include open-tag, close-tag, character data
  - Events will be fired in depth-first, left-to-right traversal order of the XML tree

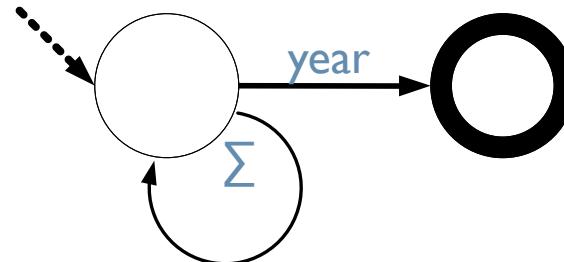
# The Second Key: Finite Automata

- Convert each XPath to an equivalent regular expression
- Build a finite automaton (NFA or DFA) for the regexp

/dblp/article

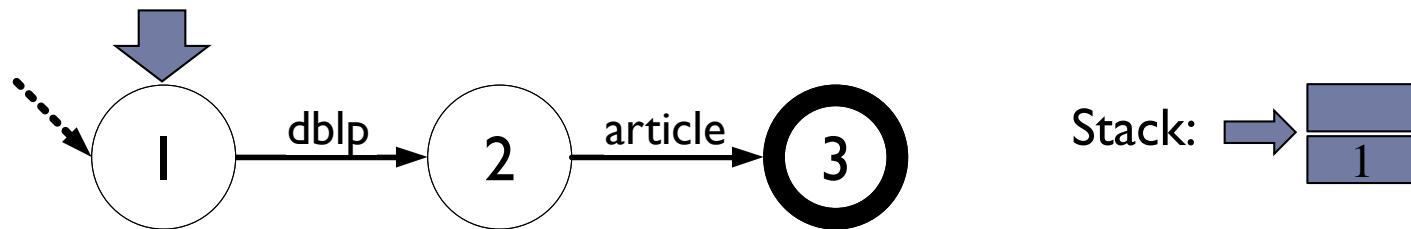


//year



# Matching an XPath

- Assume a “cursor” on active state in the automaton
- On matching open-tag: push advance active state
- On close-tag: pop active state

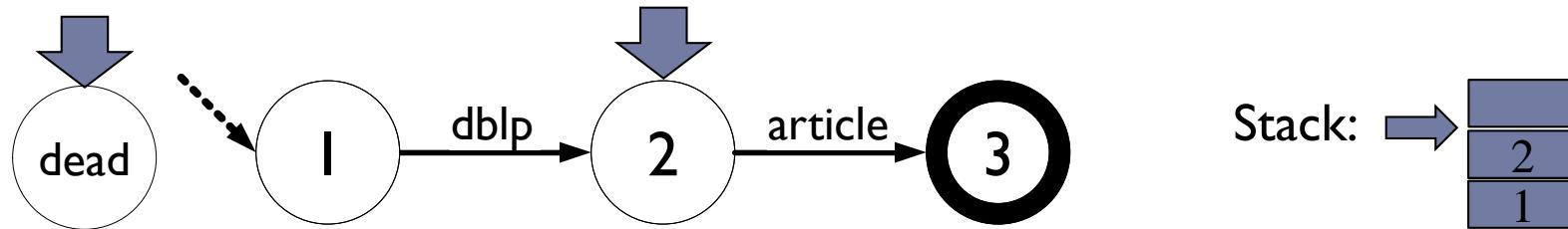


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
 <mastersthesis>
 ...
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
 <editor>Paul R. McJones</editor>
 <title>The 1995 SQL Reunion</title>
 <journal>Digital System Research Center Report</journal>
 <volume>SRC1997-018</volume>
 <year>1997</year>
 <ee>db/labs/dec/SRC1997-018.html</ee>
 <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

← event: start-element “dblp”

# Matching an XPath

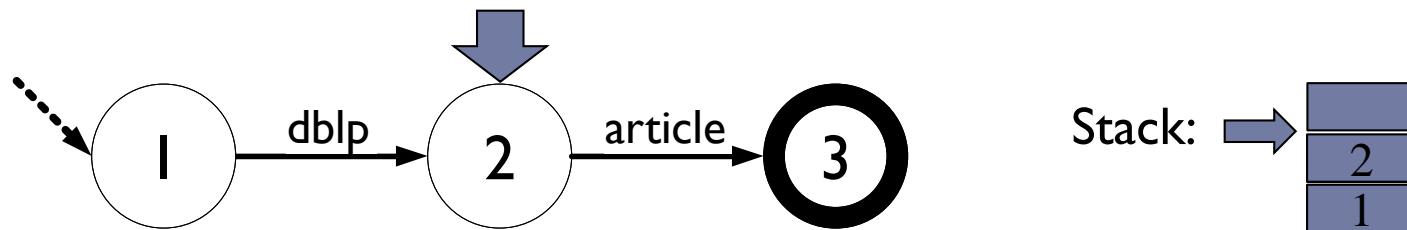
- Assume a “cursor” on active state in the automaton
  - On matching open-tag: push advance active state
  - On close-tag: pop active state



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
 <mastersthesis>
 ...
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
 <editor>Paul R. McJones</editor>
 <title>The 1995 SQL Reunion</title>
 <journal>Digital System Research Center Report</journal>
 <volume>SRC1997-018</volume>
 <year>1997</year>
 <ee>db/labs/dec/SRC1997-018.html</ee>
 <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

# Matching an XPath

- Assume a “cursor” on active state in the automaton
- On matching open-tag: push advance active state
- On close-tag: pop active state

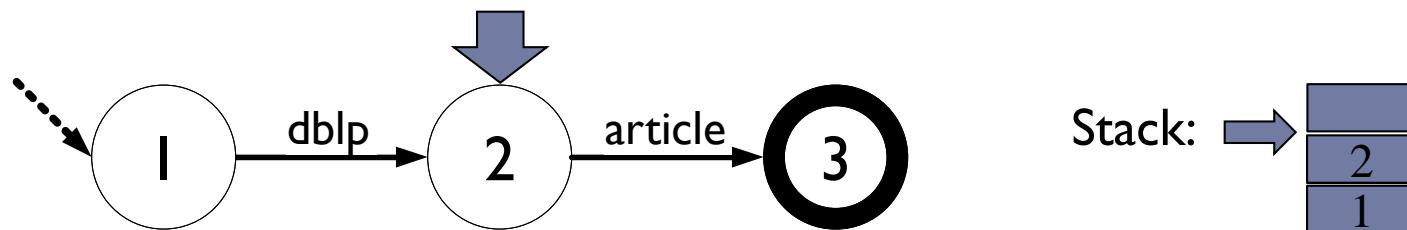


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
 <mastersthesis>
 ...
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
 <editor>Paul R. McJones</editor>
 <title>The 1995 SQL Reunion</title>
 <journal>Digital System Research Center Report</journal>
 <volume>SRC1997-018</volume>
 <year>1997</year>
 <ee>db/labs/dec/SRC1997-018.html</ee>
 <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

event: end-element “*mastersthesis*”

# Matching an XPath

- Assume a “cursor” on active state in the automaton
- On matching open-tag: push advance active state
- On close-tag: pop active state



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
 <mastersthesis>
 ...
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018"> ←
 <editor>Paul R. McJones</editor>
 <title>The 1995 SQL Reunion</title>
 <journal>Digital System Research Center Report</journal>
 <volume>SRC1997-018</volume>
 <year>1997</year>
 <ee>db/labs/dec/SRC1997-018.html</ee>
 <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

event: *start-element “article”*

*match!*

# Different Options

---

- Many different “streaming XPath” algorithms
  - What kind of automaton to use
    - ❖ DFA, NFA, lazy DFA, PDA, proprietary format
  - Expressiveness of the path language
    - ❖ Full regular path expressions, XPath, ...
    - ❖ Axes
- Which operations can be pushed into the operator
  - ❖ XPath predicates, joins, position predicates, etc.

# From XPaths to XQueries

---

- An XQuery takes multiple XPaths in the FOR/LET clauses, and iterates over the elements of each XPath (binding the variable to each)

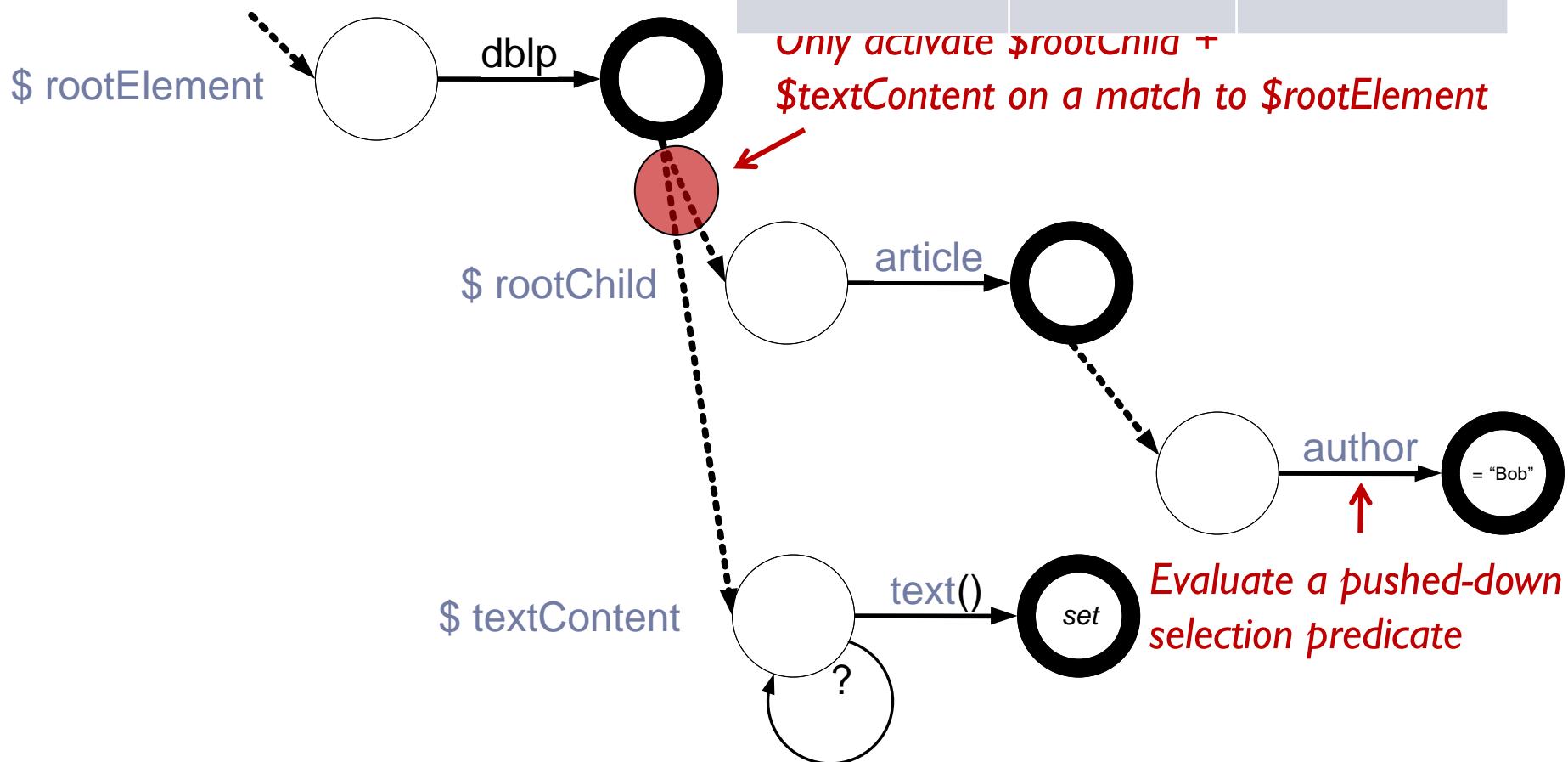
```
FOR $rootElement in doc("dblp.xml")/dblp,
 $rootChild in $rootElement/article[author="Bob"],
 $textContent in $rootChild/text()
```

- We can think of an XQuery as doing **tree matching**, which returns tuples (\$i, \$j) for each tree matching \$i and \$j in a document
- Streaming XML path evaluator that supports a hierarchy of matches over an XML document

# XQuery Path Evaluation

```
FOR $rootElement in doc("dblp.xml")/dblp,
 $rootChild in $rootElement/article[author="Bob"],
 $textContent in $rootChild/text()
```

- Multiple, dependent state machines outputting *binding tuples*



# Beyond the Initial FOR Paths

- The streaming XML evaluator operator returns tuples of bindings to nodes

| \$rootElement | \$rootChild | \$textContent |
|---------------|-------------|---------------|
|               |             |               |

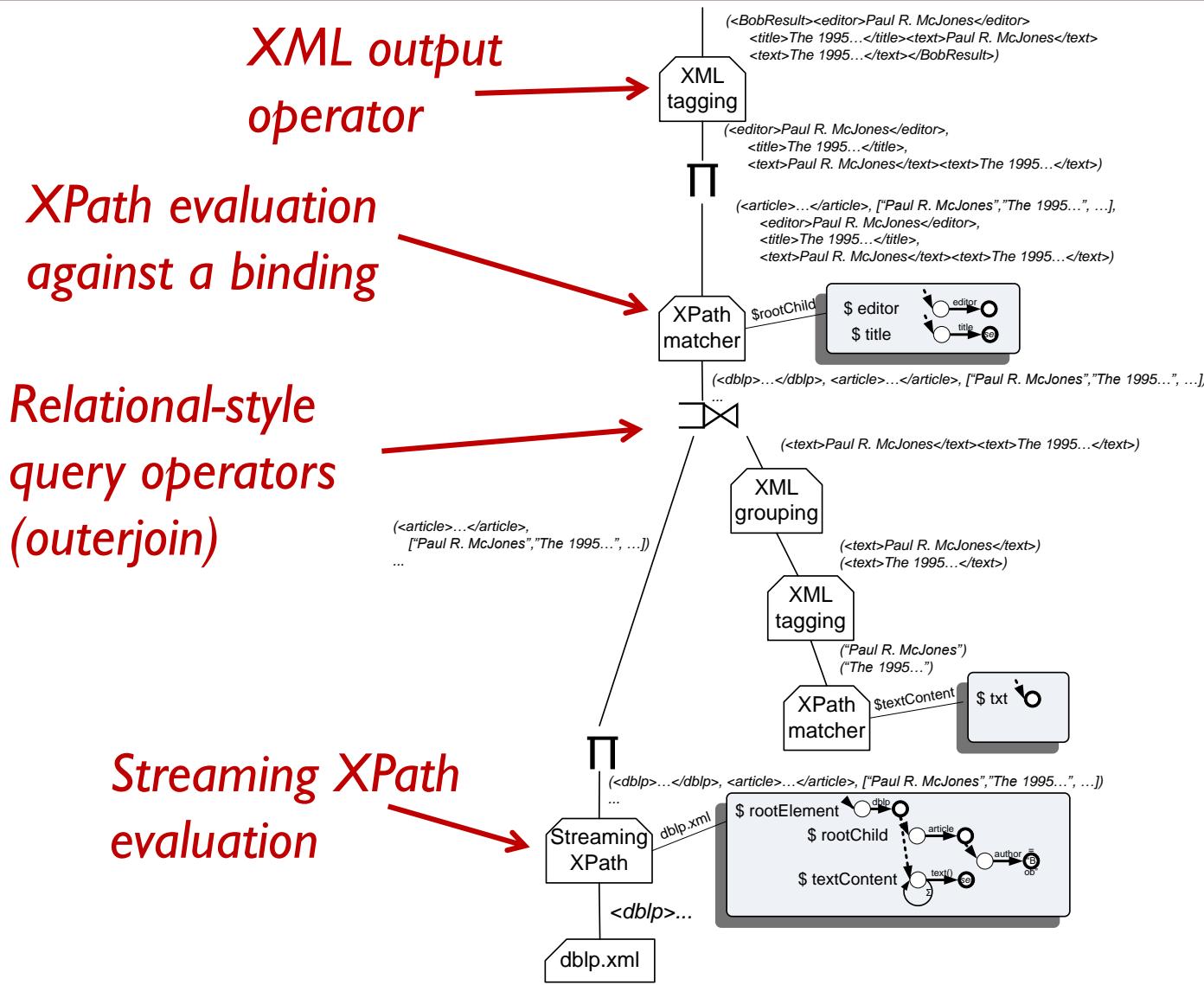
- We can now use standard relational operators to join, sort, group, etc.
- Also in some cases we may want to do further XPath evaluation against one of the XML trees bound to a variable

# Creating XML

---

- To return XML, we need to be able to take streams of binding tuples and:
  - Add tags around certain columns
  - Group tuples together and nest them under tags
- Thus XQuery evaluators have new operators for performing these operations

# An Example XQuery Plan



# Optimizing XQueries

---

- An entire field in and of itself
- A major challenge versus relational query optimization: estimating the “fan-out” of path evaluation
- A second major challenge: full XQuery supports arbitrary recursion and is Turing-complete

# Outline

---

- ✓ XML data model
- ✓ XML schema languages
- ✓ XML querying
- ✓ XML query processing
- XML schema mapping

# Schema Mappings for XML

---

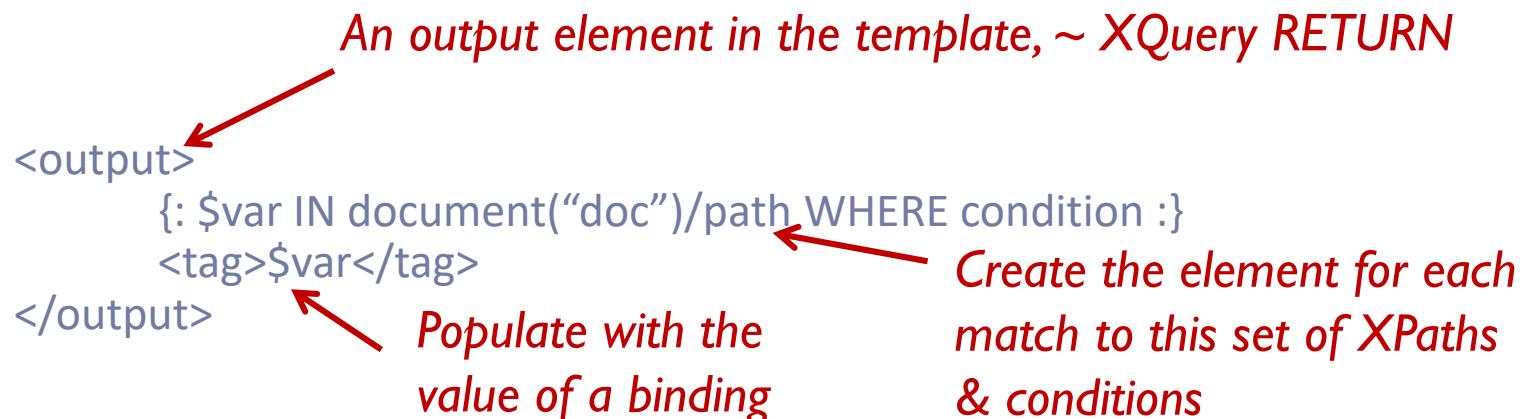
- In Chapter 3 we saw how schema mappings were described for relational data
  - As a set of **constraints** between source and target databases
- In the XML realm, we want a similar constraint language, but must address:
  - Nesting – XML is hierarchical
  - Identity – how do we *merge* multiple partial results into a single XML tree?

# One Approach: Piazza XML Mappings

Derived from a subset of XQuery extended with node identity

- The latter is used to merge results with the same node ID

Directional mapping language based on **annotations to XML templates**



- Translates between parts of data instances
- Supports special annotations and object fusion

# Mapping Example between Two XML Schemas

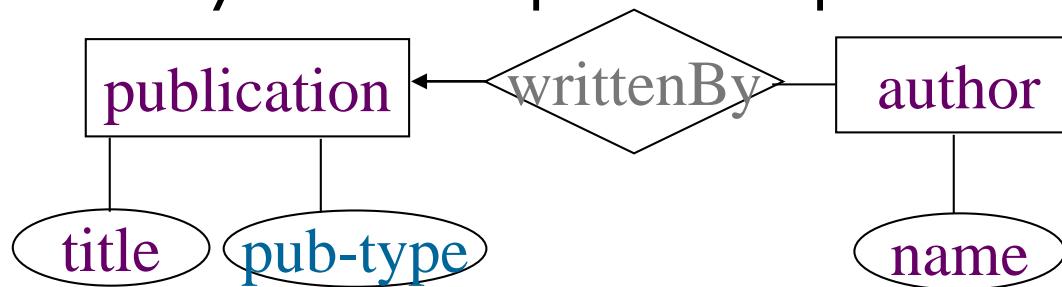
Target: *Publications by book*

<pubs>  
<book>\*  
<title>  
<author>\*  
<name>

Source: *Publications by author*

<authors>  
<author>\*  
<full-name>  
<publication>\*  
<title>  
<pub-type>

Has an entity-relationship model representation like:



# Example Piazza-XML Mapping

```
<pubs>
 <book>
 {_: $a IN document("...")/authors/author,
 $an IN $a/full-name,
 $t IN $a/publication/title,
 $typ IN $a/publication/pub-type
 WHERE $typ = "book" :}

 <title>{$t}</title>
 <author><name>{$an}</name></author>
 </book>
</pubs>
```

Output one book per match to author

Insert title and author name subelements

# Example Piazza-XML Mapping

```
<pubs>
 <book piazza:id={$t}><!--
 { : $a IN document("...")/authors/author,
 $an IN $a/full-name,
 $t IN $a/publication/title,
 $typ IN $a/publication/pub-type
 WHERE $typ = "book" :} -->
 <title piazza:id={$t}>{$t}</title>
 <author><name>{$an}</name></author>
 </book>
</pubs>
```

*Merge elements if they are for the same value of \$t*

*Output one book per match to author*

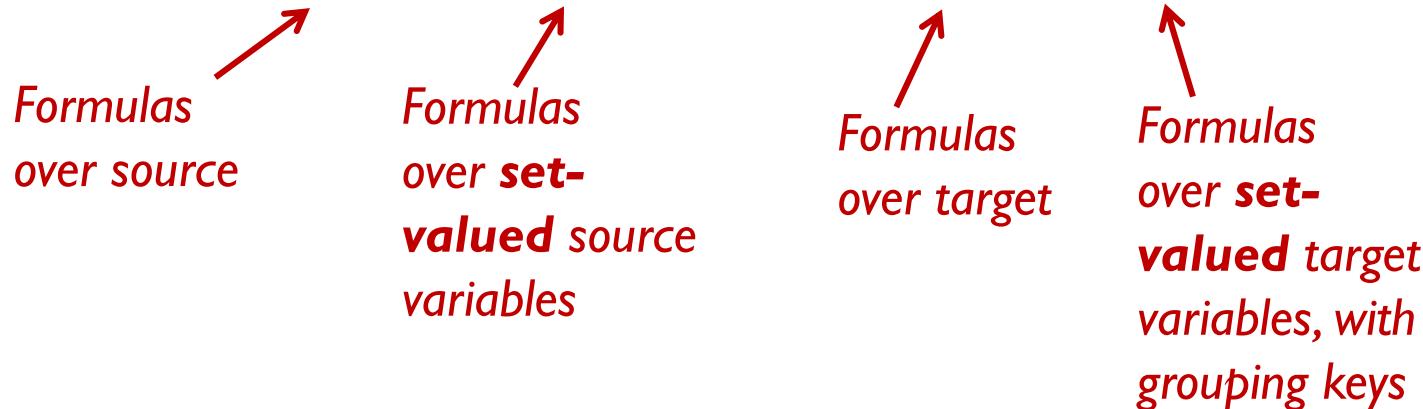
*Insert title and author name subelements*

# A More Formal Model: Nested TGDs

The underpinnings of the Piazza-XML mapping language can be captured using *nested tuple-generating dependencies* (nested TGDs)

- Recall relational TGDs from Chapter 3

$$\forall \bar{X}, \bar{Y}, \bar{S} (\phi(\bar{X}, \bar{Y}) \wedge \Phi(\bar{S}) \rightarrow \exists \bar{Z}, \bar{T} (\psi(\bar{X}, \bar{Z}) \wedge \Psi(\bar{T})))$$



- As before, we'll typically omit the  $\forall$  quantifiers...

# Example Piazza-XML Mapping as a Nested TGD

```
<pubs>
 <book piazza:id={$t}>
 { : $a IN document("...")/authors/author,
 $an IN $a/full-name,
 $t IN $a/publication/title,
 $typ IN $a/publication/pub-type
 WHERE $typ = "book" :}

 <title piazza:id={$t}>{$t}</title>
 <author><name>{$an}</name></author>
 </book>
 </pubs>
```

$\text{authors}(\text{author}) \wedge \text{author}(f, \text{publication}) \wedge \text{publication}(t, \text{book}) \rightarrow$

$\exists p (\text{pubs}(\text{book}) \wedge \text{book}_t(t, \text{author}', \text{publisher}) \wedge \text{author}'_{t,f}(f) \wedge \text{publisher}_t(p))$

*Grouping keys in target*

# Query Reformulation for XML

---

- Two main versions:
  - Global-as-view-style:
    - ❖ Query is posed over the target of a nested TGD, or a Piazza-XML mapping
    - ❖ Can answer the query through standard XQuery view unfolding
  - Bidirectional mappings, more like GLAV mappings in the relational world:
    - ❖ An advanced topic – see the bibliographic notes

# XML Wrap-up

---

- XML forms an important part of the data integration picture – it's a “bridge” enabling rapid connection to external sources
  
- It introduces new complexities in:
  - Query processing – need streaming XPath / XQuery evaluation
  - Mapping languages – must support identity and nesting
  - Query reformulation
- It also is a bridge to RDF and the Semantic Web (Chapter 12)