Storing Entities and Relationships

Susan B. Davidson and Zachary G. Ives

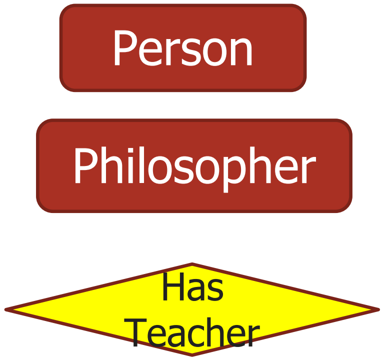
University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

How Do We Store Entities & Relationships?



Entity set: represents all of the entities of a type, and their properties

- Person: ID, name, birth, death
- Philosopher: inherits the same fields, possibly adds new ones

Relationship set: represents a link between people

- *HasTeacher*(teacher: ID of Person, student: ID of Person)

Person (Also: Philosopher)

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

Key

Foreign keys

HasTeacher

Teacher	Student
1233	1234
1232	1233

https://tinyurl.com/cis545-lecture-02-07-22

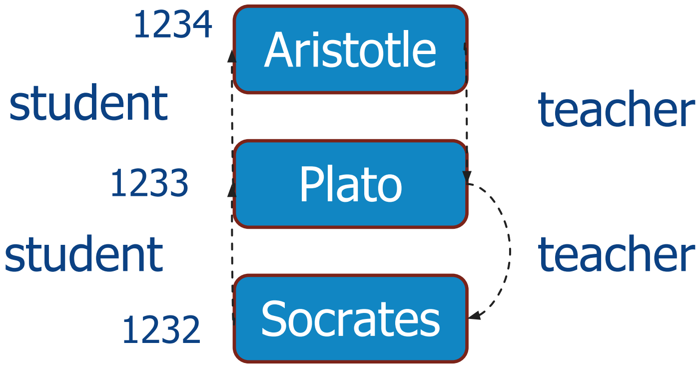
The Tables Represent a Graph of Connections

Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

Teacher	Student
1233	1234
1232	1233



https://tinyurl.com/cis545-lecture-02-07-22

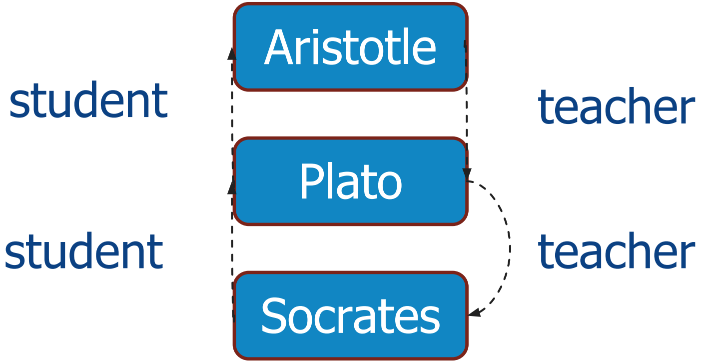
Who Is the Teacher of Aristotle's Teacher?

Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

Teacher	Student
1233	1234
1232	1233



SELECT
FROM
WHERE

https://tinyurl.com/cis545-lecture-02-07-22

Who Is the Teacher of Aristotle's Teacher?

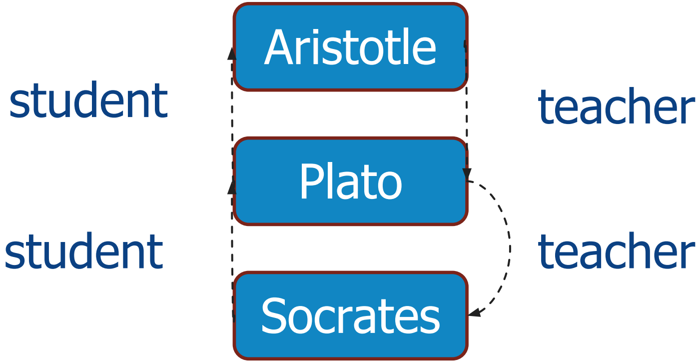
Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

A

HasTeacher

Teacher	Student
1233	1234
1232	1233



SELECT
FROM Person A
WHERE A.name='Aristotle'

https://tinyurl.com/cis545-lecture-02-07-22

Who Is the Teacher of Aristotle's Teacher?

Person

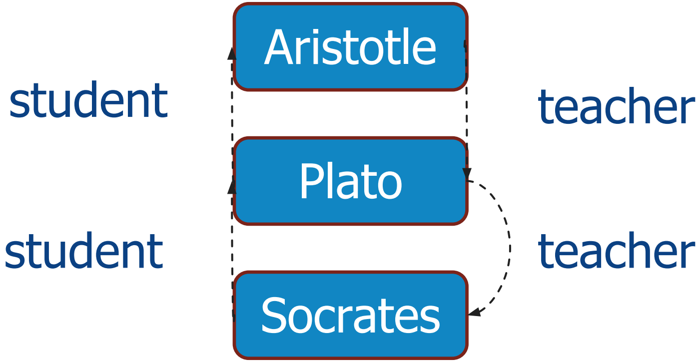
A

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

PI

Teacher	Student
1233	1234
1232	1233




```
SELECT
FROM Person A JOIN HasTeacher PI
      ON ID=Student
WHERE A.name='Aristotle'
```

https://tinyurl.com/cis545-lecture-02-07-22

Who Is the Teacher of Aristotle's Teacher?

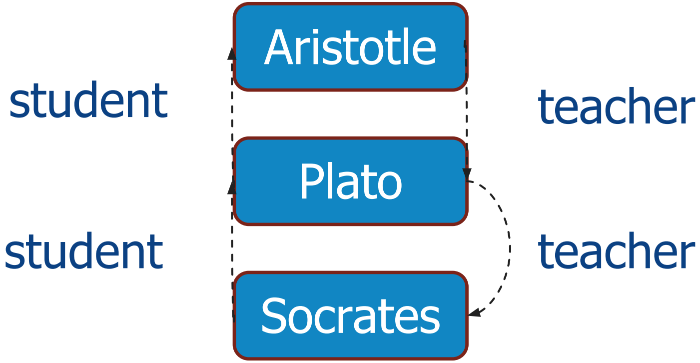
Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

A

HasTeacher

	Teacher	Student
Pl	1233	1234
So	1232	1233




```
SELECT
FROM Person A JOIN HasTeacher Pl
        ON A.ID=Student JOIN HasTeacher So
        ON Pl.teacher = So.student
WHERE A.name='Aristotle'
```

https://tinyurl.com/cis545-lecture-02-07-22

Who Is the Teacher of Aristotle's Teacher?

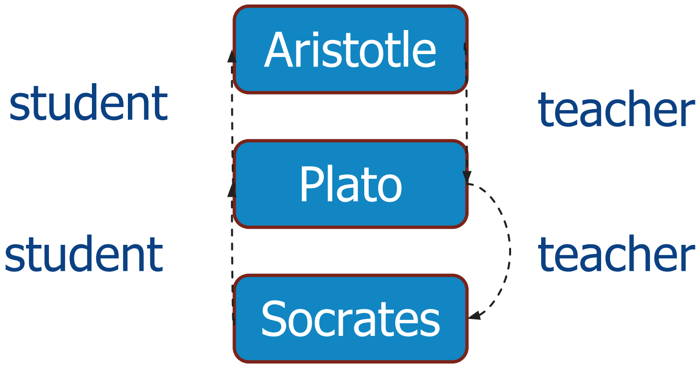
Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

A

HasTeacher

	Teacher	Student
Pl	1233	1234
So	1232	1233




```
SELECT So.teacher
FROM Person A JOIN HasTeacher Pl
      ON A.ID=Student JOIN HasTeacher So
      ON Pl.teacher = So.student
WHERE A.name='Aristotle'
```

https://tinyurl.com/cis545-lecture-02-07-22

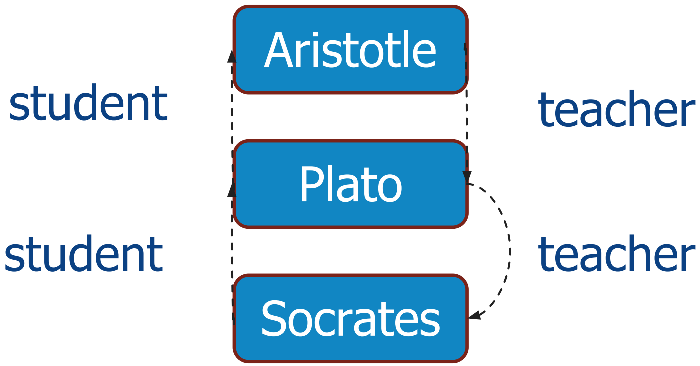
Who Are Socrates' Academic Descendants?

Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

Teacher	Student
1233	1234
1232	1233



SELECT
FROM
WHERE

https://tinyurl.com/cis545-lecture-02-07-22

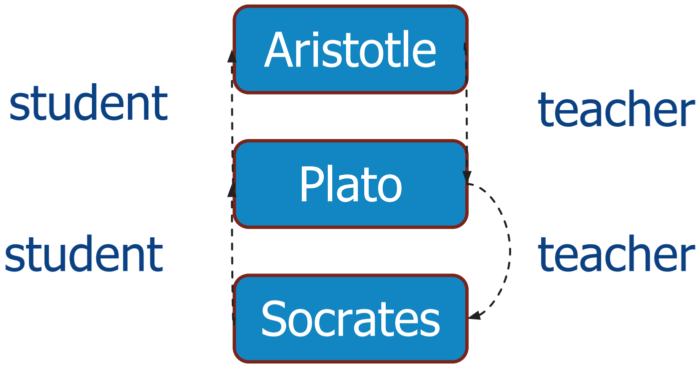
Who Are Socrates' Academic Descendants?

Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

Teacher	Student
1233	1234
1232	1233



SELECT
FROM Person So
WHERE So.Name='Socrates'

https://tinyurl.com/cis545-lecture-02-07-22

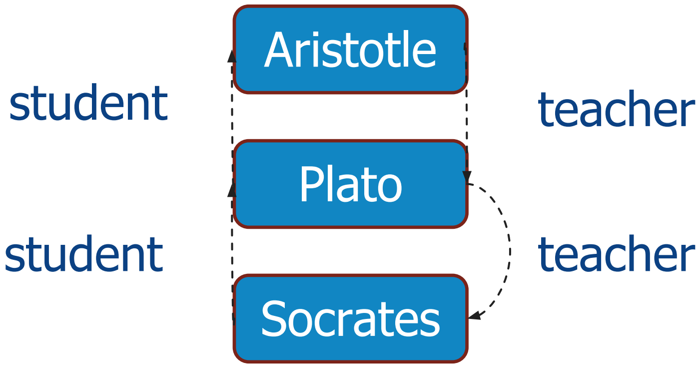
Who Are Socrates' Academic Descendants?

Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

Teacher	Student
1233	1234
1232	1233




```
SELECT Pl.Student
FROM Person So JOIN HasTeacher Pl
   ON So.ID = Pl.teacher
WHERE So.Name='Socrates'
```

https://tinyurl.com/cis545-lecture-02-07-22

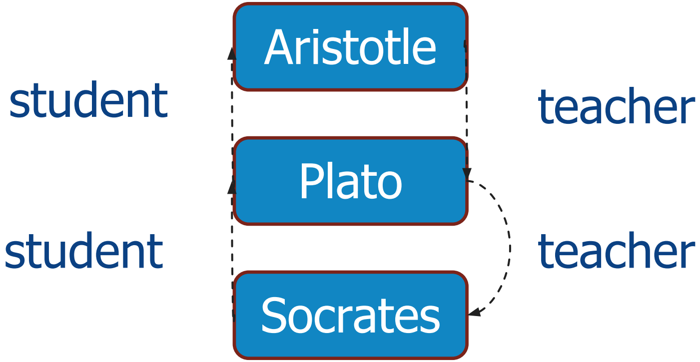
Who Are Socrates' Academic Descendants?

Person

ID	Name	Birth	Death
1234	Aristotle	384 BC	322 BC
1233	Plato	428 BC	348 BC
1232	Socrates	470 BC	399 BC

HasTeacher

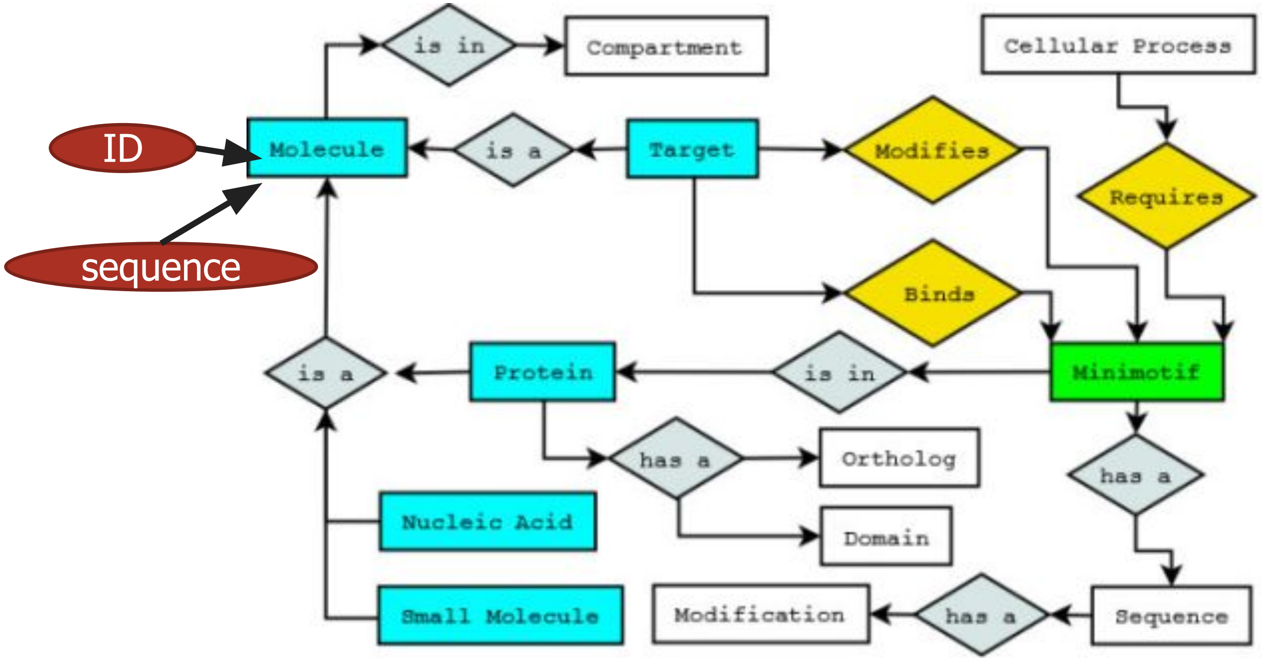
Teacher	Student
1233	1234
1232	1233




```
SELECT Pl.Student
FROM Person So JOIN HasTeacher Pl
  ON So.ID = Pl.teacher
WHERE So.Name='Socrates'
UNION
SELECT Ar.Student
FROM Person So JOIN HasTeacher Pl
  ON So.ID = Pl.teacher JOIN HasTeacher Ar
  ON Ar.teacher = Pl.student
WHERE So.Name='Socrates'
```

https://tinyurl.com/cis545-lecture-02-07-22

ER is a General Model:
A Graph of Entities & Relationships



https://tinyurl.com/cis545-lecture-02-07-22

General Database Design

Deciding on the entities, relationships, and constraints is part of *database design*

- There are ways to do this to minimize the errors in the database, and make it easiest to keep *consistent*

- See CIS 450/550 for details

For this class: we'll assume we do simple E-R diagrams with properties

… and that each node becomes a Dataframe

https://tinyurl.com/cis545-lecture-02-07-22

Recap: Basic Concepts in Data Modeling

Knowledge represented as **concepts or classes**, which can correspond to tables

- But there is also a notion of **subclassing** (inheriting fields)
- And of **instances** (rows in the tables)

Knowledge representation often describes these relationships as constraints

We can capture knowledge using graphs with nodes (entity sets, concepts) and edges (relationship sets)

- Entity-relationship diagrams show this
- Entity sets and relationship sets can both become tables!
- Graphs + queries can be used to capture any kind of data and relationships (not always conveniently)

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771559 (06D)

- A foreign key:
  a. takes on a value from a key in another table
  b. is a C++ pointer
  c. must be unique within its own table
  d. has multiple values per row
- The data in a relational database can be modeled as a graph of tuple relationships, as we saw in the slides. How do we traverse edges in this graph?
  a. filtering / selection
  b. applymap
  c. joins
  d. unions

https://tinyurl.com/cis545-lecture-02-07-22

Hierarchical Data and NoSQL

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

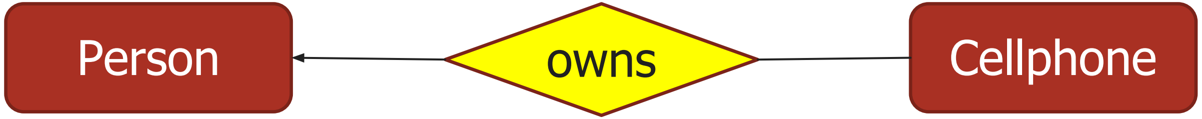
CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

Hierarchy vs Relations
(“NoSQL” vs “SQL”)

Sometimes it’s convenient to take data we could codify as a graph:



And instead save it as a *tree* or *forest*:

[{‘person’: {‘name’: ‘jai’, phones: [{‘mfr’: ‘Apple’, ‘model’: …},
{‘mfr’: ‘Samsung’, ‘model’: …}},
{‘person’: {‘name’: ‘kai’, phones: [{‘mfr’: ‘Apple’, ‘model’: …}]}]

This is what NoSQL databases do!

https://tinyurl.com/cis545-lecture-02-07-22

Let's Now Look at a Working Example: *Social Network Analysis*

https://tinyurl.com/cis545-notebook-03

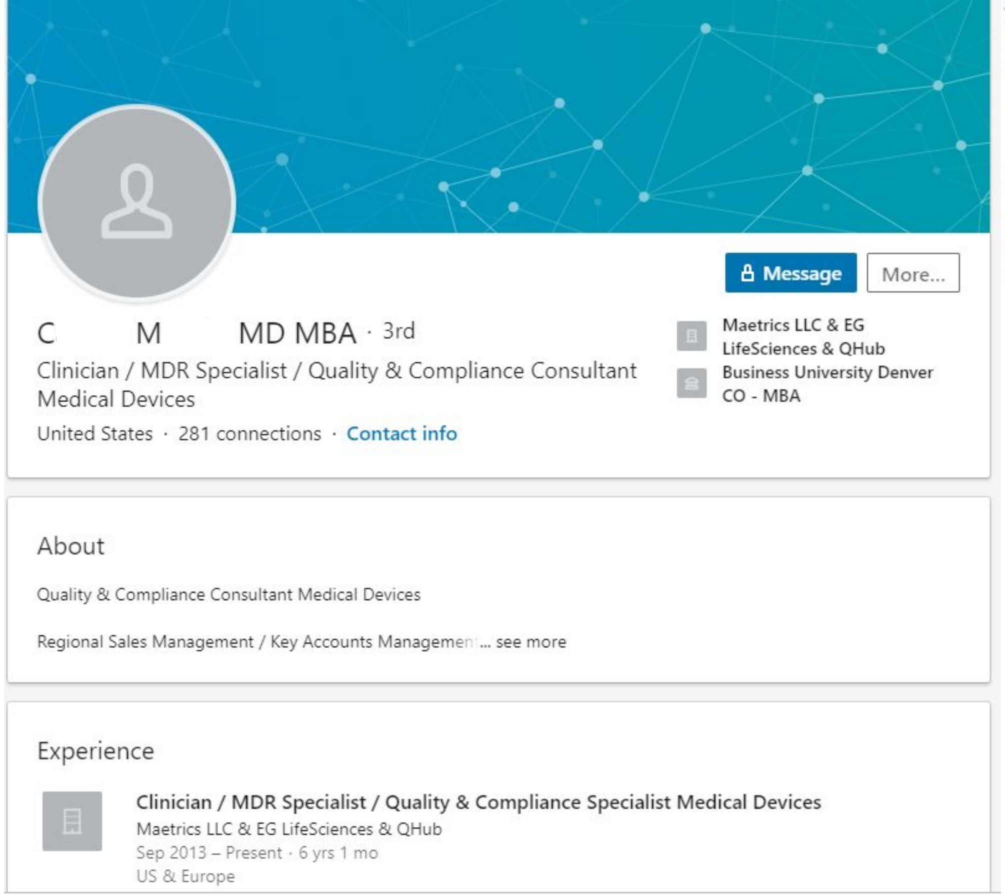
Extracted data from LinkedIn, was in https://www.kaggle.com/linkedindata/linkedin-crawled-profiles-dataset

~3M people, stored as a ~9GB list of lines made up of JSON

(For the Colab version we've cut to 10,000 lines so it executes quickly enough.)

JSON is nested dictionaries and lists – i.e., NoSQL-style !

https://tinyurl.com/cis545-lecture-02-07-22




```
{
"_id":"in-00000001",
"name":{
  "family_name":"M ",
  "given_name":"Dr C "
},
"locality":"United States",
"skills":[
  "Key Account Development",
  "Strategic Planning",
  "Market Planning",
  "Team Leadership",
  "Negotiation",
  "Forecasting",
  "Key Account Management",
  "Sales Management",
  "New Business Development",
  "Business Planning",
  "Cross-functional Team Leader
  "Budgeting",
  "Strategy Development",
  "Business Strategy",
  "Consultative Selling",
  "Medical Devices",
  "Customer Relations",
  "Contract Negotiation"
```


https://tinyurl.com/cis545-lecture-02-07-22

NoSQL Databases (We'll See Details in a Bit)

- Originally, indeed stood for "no-SQL", now "not-only-SQL"
- Typically store **nested objects**, or possibly binary objects, by IDs or keys

Note that a nested object can be captured in relations, via multiple tables!

Some well-known NoSQL systems:

- MongoDB: stores JSON, i.e., lists and dictionaries
- Google Bigtable: stores tuples with irregular properties
- Amazon S3: stores binary files by key

Major differences from SQL databases:

- Querying is often much simpler, e.g. they often don't do joins!
- They support limited notions of **consistency** when you update [we'll discuss later]

https://tinyurl.com/cis545-lecture-02-07-22

Parsing Even Not-So-Big Data Is Painfully Slow!

```
%%time
# 100,000 records from linkedin
linked_in = open('linkedaa')

people = []

for line in linked_in:
    person = json.loads(line)
    people.append(person)

people_df = pd.DataFrame(people)
people_df[people_df['industry'] == 'Medical Devices']
```

```
CPU times: user 58.2 s, sys: 1min 57s, total: 2min 55s
Wall time: 3min 19s
```

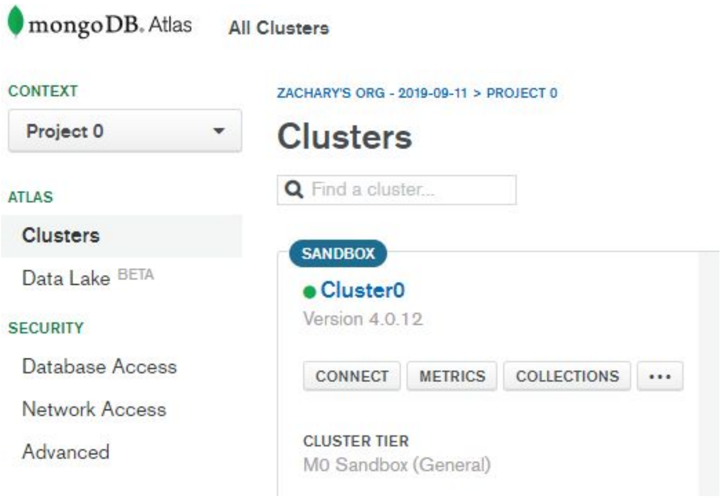
	_id	name	locality	skills	industry	summary
0	in-00000001	{'family_name': 'Mazalu MBA', 'given_name': 'D...	United States	[Key Account Development, Strategic Planning, ...	Medical Devices	SALES MANAGEMENT / BUSINESS DEVELOPMENT / PROJ...
161	in-13806219531	{'family_name': 'Gao', 'given_name': 'Tony'}	China	[ISO 13485, Medical Devices]	Medical Devices	NaN

https://tinyurl.com/cis545-lecture-02-07-22

Can We Do Better?

Maybe save the data in a way that doesn't require parsing of strings?

https://cloud.mongodb.com



https://tinyurl.com/cis545-lecture-02-07-22

MongoDB NoSQL DBMS
Lets Us Store + Fetch Hierarchical Data

```
client =
MongoClient('mongodb+srv://cis545:1course4all@cluster0-cy1yu.mongodb.
net/test?retryWrites=true&w=majority')

linkedin_db = client['linkedin']
linked_in = open('linkedin.json')

for line in linked_in:
    person = json.loads(line)
    linkedin_db.posts.insert_one(person)
```

https://tinyurl.com/cis545-lecture-02-07-22

Data in MongoDB


```
_id: "in-00001"
> education: Array
> group: Object
> name: Object
  overview_html: "<dl id="overview"><dt id="overview-summary-current-title" class="summa..."
  locality: "Antwerp Area, Belgium"
> skills: Array
  industry: "Pharmaceuticals"
  interval: 20
v experience: Array
  > 0: Object
  > 1: Object
  v 2: Object
      org: "Columbia University"
      title: "Associate Research Scientist"
      start: "August 2006"
      desc: "Work on peptide to restore wt p53 function in cancer."
  > 3: Object
  > 4: Object
  summary: "Ph.D. scientist with background in cancer research, translational medi..."
  url: "http://be.linkedin.com/in/00001"
> also_view: Array
  specilities: "Biomarkers in Oncology, Cancer Genomics, Molecular Profiling of Cancer..."
> events: Array
```


https://tinyurl.com/cis545-lecture-02-07-22

Finding Things: In-Memory List vs in MongoDB

```
def find_skills_in_list(skill):
    for post in list_for_comparison:
        if 'skills' in post:
            skills = post['skills']
            for this_skill in skills:
                if this_skill == skill:
                    return post
    return None

def find_skills_in_mongodb(skill):
    return linkedin_db.posts.find_one({'skills': skill})
```

Similar to an XPath
posts[skills=mySkill]

https://tinyurl.com/cis545-lecture-02-07-22

The Story So Far

For hierarchical data, it's often useful to use a NoSQL database

- Natural mapping for nested dictionaries, JSON, etc.

Such systems often support queries that are in terms of nested content

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771563 (06E)

- NoSQL systems are distinct from relational database management systems in that they:
  a. provide improved consistency
  b. typically support key-based lookup of values
  c. emphasize joins
  d. cannot support SQL
- Hierarchical JSON data ("forests") may be faster to access from MongoDB, versus parsing a file into a dataframe and querying, because
  a. we can query for individual JSON trees
  b. it's faster to parse everything in MongoDB
  c. the network is slow
  d. MongoDB operates faster servers

https://tinyurl.com/cis545-lecture-02-07-22

Hierarchy and Relations

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

How Do We Convert Hierarchical Data to Dataframes (Tables)?

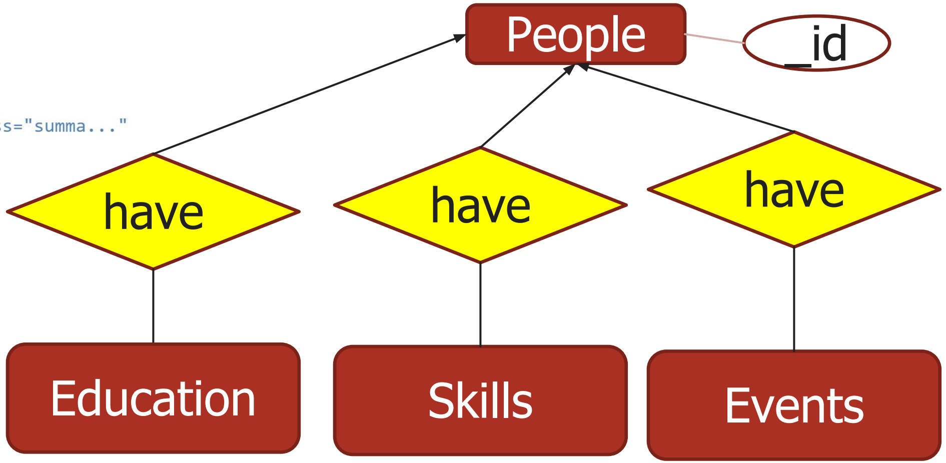
Hierarchical data doesn't work well for data visualization or machine learning


```
_id: "in-00001"
> education: Array
> group: Object
> name: Object
  overview_html: "<dl id="overview"><dt id="overview-summary-current-title" class="summa..."
  locality: "Antwerp Area, Belgium"
> skills: Array
  industry: "Pharmaceuticals"
  interval: 20
v experience: Array
  > 0: Object
  > 1: Object
  v 2: Object
      org: "Columbia University"
      title: "Associate Research Scientist"
      start: "August 2006"
      desc: "Work on peptide to restore wt p53 function in cancer."
  > 3: Object
  > 4: Object
  summary: "Ph.D. scientist with background in cancer research, translational medi..."
  url: "http://be.linkedin.com/in/00001"
> also_view: Array
  specilities: "Biomarkers in Oncology, Cancer Genomics, Molecular Profiling of Cancer..."
> events: Array
```


https://tinyurl.com/cis545-lecture-02-07-22

The Basic Idea:
Split into Tables When There Isn’t 1:1

_id: "in-00001"
› education: Array
› group: Object
› name: Object
overview_html: "<dl id="overview"><dt id="overview-summary-current-title" class="summa..."
locality: "Antwerp Area, Belgium"
› skills: Array
industry: "Pharmaceuticals"
interval: 20
⌄ experience: Array
› 0: Object
› 1: Object
⌄ 2: Object
org: "Columbia University"
title: "Associate Research Scientist"
start: "August 2006"
desc: "Work on peptide to restore wt p53 function in cancer."
› 3: Object
› 4: Object
summary: "Ph.D. scientist with background in cancer research, translational medi..."
url: "http://be.linkedin.com/in/00001"



https://tinyurl.com/cis545-lecture-02-07-22

The Basic Idea: Nesting Becomes Links (“Key/Foreign Key”)

```
_id: "in-00001"
> education: Array
> group: Object
> name: Object
  overview_html: "<dl id="overview"><dt id="overview-summary-cu
  locality: "Antwerp Area, Belgium"
> skills: Array
  industry: "Pharmaceuticals"
  interval: 20
v experience: Array
  > 0: Object
  > 1: Object
  v 2: Object
      org: "Columbia University"
      title: "Associate Research Scientist"
      start: "August 2006"
      desc: "Work on peptide to restore wt p53 function in ca
  > 3: Object
  > 4: Object
  summary: "Ph.D. scientist with background in cancer research,
  url: "http://be.linkedin.com/in/00001"
> also_view: Array
  specilities: "Biomarkers in Oncology, Cancer Genomics, Molecu
> events: Array
```

people

_id	Overview_html	locality	industry	...
in-00001	<dl id=...	Antwerp Area	Pharmaceu	

experience

person	org	title	start	desc
in-00001	Columbia	Assoc…	August	Wor…
in-00001	…	…	…	…

https://tinyurl.com/cis545-lecture-02-07-22

Reassembling through (Left Outer) Joins

```
pd.read_sql_query("select _id, org" +\
                  " from people left join experience on _id=person ",\
                  conn)
```

_id	org
in-00001	Albert Einstein Medical Center
in-00001	Columbia University
in-00001	Johnson and Johnson

```
pd.read_sql_query("select _id, \'[\' + group_concat(org) + \']\'" +\
                  " from people left join experience on _id=person "+\
                  " group by _id", conn)
```

_id	experience
in-00000001	None
in-00001	Albert Einstein Medical Center,Columbia Univer...
in-00006	UCSF,Wyss Institute for Biologically Inspired ...

https://tinyurl.com/cis545-lecture-02-07-22

Views

Sometimes we use a query enough that we want to give its results a name, and make it essentially a table

```
conn.execute("create view people_experience as " +\
             " select _id, group_concat(org) as experience " +\
             " from people left join experience on _id=person group by _id")

pd.read_sql_query('select * from people_experience', conn)
```

https://tinyurl.com/cis545-lecture-02-07-22

Occasional Data Storage Considerations: Access and Consistency

Sometimes we may need to allow for failures and "undo"…

- We saw "BEGIN TRANSACTION … COMMIT"
- There is also "ROLLBACK"

Relational DBMS typically provide atomic **transactions** for this; most NoSQL DBMSs don't

A second consideration when the data is shared: what happens when multiple users are editing and querying at the same time?

- **Concurrency control** (how do we handle concurrent updates) and **consistency** (when do I see changes)
- The focus of other courses, e.g. CIS 450/550…

https://tinyurl.com/cis545-lecture-02-07-22

Recap

We can model hierarchical data in relations

- Using separate relations for each 1:many or many:many nesting level

Need to (left outer)join it to reassemble the hierarchy

- *Views* let us give a name to the reassembled results

If data isn't static, we should consider **transactions** and **concurrency**

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771582 (06F)

- We split hierarchical data into tables when
  a. the data has multiple values per parent item
  b. the data has a separate column name
  c. the data is a nested dictionary
  d. the data can be represented as an ER diagram
- We may want to use left outerjoins to reassemble hierarchical data because
  a. there may be parent items with no children
  b. outerjoin is faster than innerjoin
  c. outerjoin is hierarchical
  d. outerjoin returns a subset of the answers of innerjoin

https://tinyurl.com/cis545-lecture-02-07-22

Summary of Data Modeling

Representing data's classes and properties is essential

- We can do this via logical constraints (including queries)
- And by diagrams

Two main kinds of data models for databases

- NoSQL – largely hierarchical
- Relational

With joins, each can encode graphs – thus they are equivalent in what they capture, but the convenience of querying differs!

https://tinyurl.com/cis545-lecture-02-07-22

Efficient Data Processing

(Or, How to Not Crash Your Python Kernel)

Susan B. Davidson and Zachary G. Ives

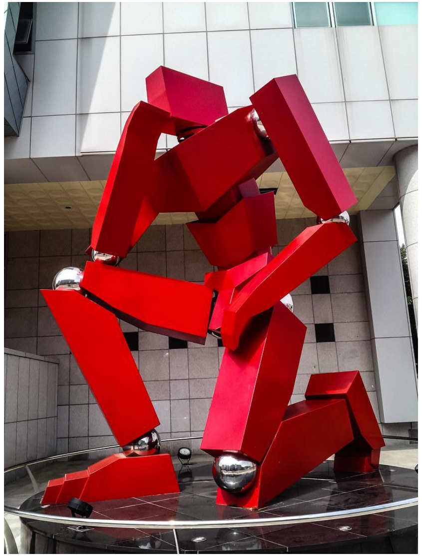
University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

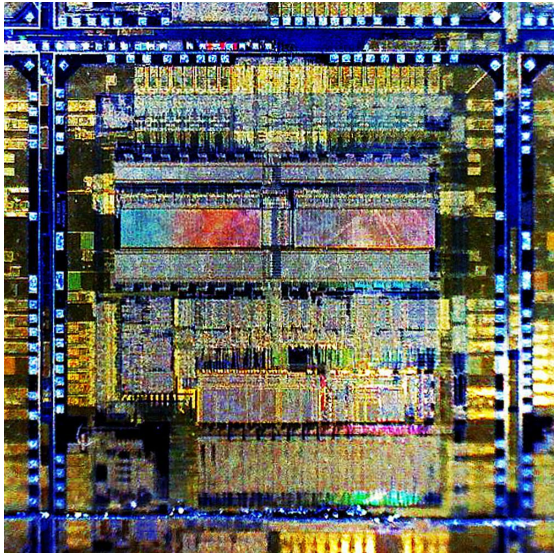
https://tinyurl.com/cis545-lecture-02-07-22

Last time: *conceptual* data representation



"Thinking Robot" by purunuri is licensed under CC BY-NC-SA 2.0

This time: *physical* data representation and processing



"Chip Geometry" by byzantiumbooks is licensed under CC BY 2.0

https://tinyurl.com/cis545-lecture-02-07-22

As We Get to *Big* Data…

How we encode and index data affects how expensive it is to process

Choice (and order) of *algorithms* affects performance!

Informed decisions can make the difference between un-answerable computations (kernel crashes) and quick results!

https://tinyurl.com/cis545-lecture-02-07-22

Data Engineering

- At the beginning, we mentioned the field has begun to segment into *data science* vs *data engineering*

- An understanding of how to handle scale is squarely in the purview of data engineering – and critical to many real tasks

https://tinyurl.com/cis545-lecture-02-07-22

Roadmap for Data Processing

- How computer architecture and memory affect performance
- Optimizing relational algebra to minimize memory costs
- Optimizing join orders
- Algorithmic techniques

https://tinyurl.com/cis545-lecture-02-07-22

Essentials of Computer Architecture

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

Understanding a Bit of Computer Architecture Is Key

A few ideas we'll need from Computer Architecture:

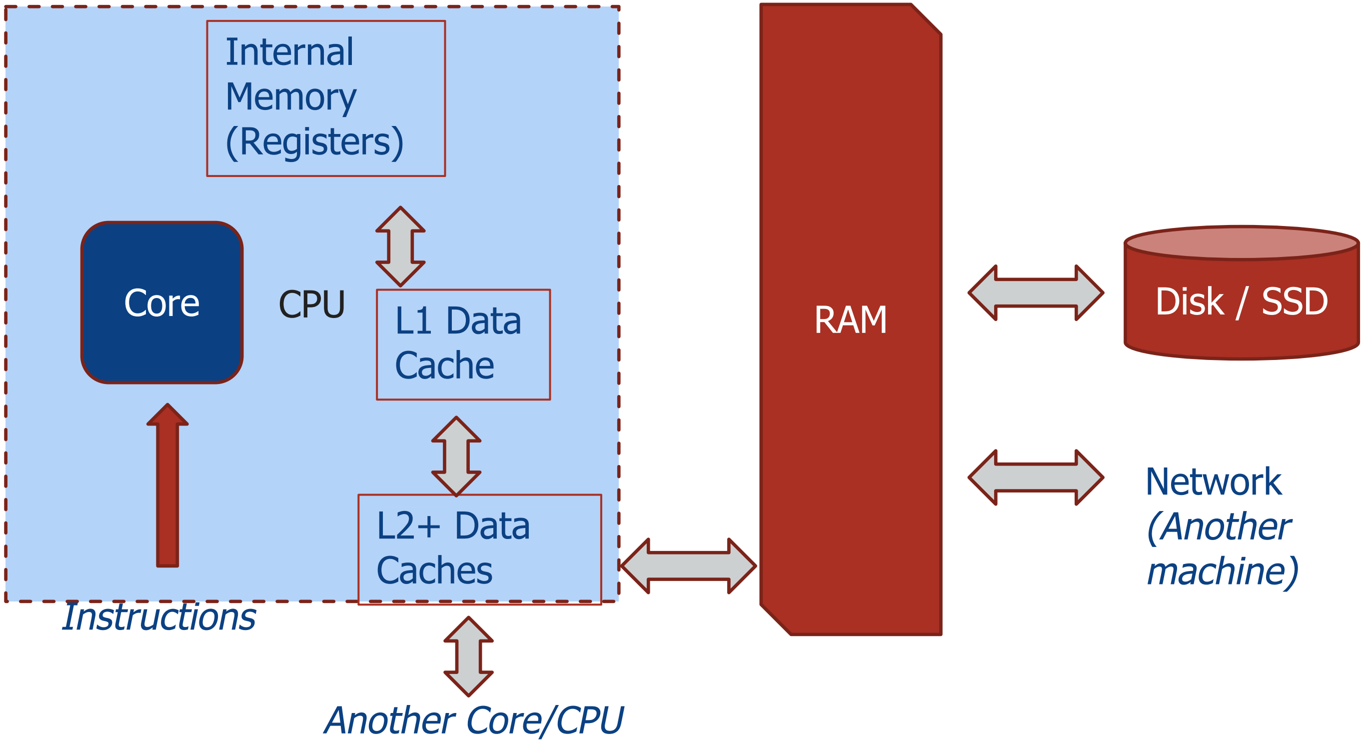
1) Different program instructions take different amounts of work and time
   - Lines of code (in Python / C / etc) turn into multiple *machine instructions*
   - Machine instructions don't all have equal performance!

2) The computer's *memory hierarchy* means accessing data is also not uniform!

3) Modern hardware includes special optimizations to:
   - Do the same operation on multiple data items simultaneously
   - Run multiple independent pieces of code at the same time

https://tinyurl.com/cis545-lecture-02-07-22

Key Aspects of a Computer: Simplified View of a Microprocessor



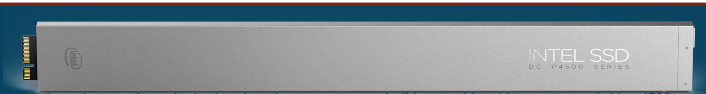
https://tinyurl.com/cis545-lecture-02-07-22

The Memory Hierarchy from One X86's Perspective

Ryzen 2000 vs Core 8000: https://www.anandtech.com/show/12625/amd-second-generation-ryzen-7-2700x-2700-ryzen-5-2600x-2600/3

Registers	16 int, 16-32 double, 0.2 ns
L1 cache	32KB, 1ns
L2 cache	256-512KB 2-3ns
L3 cache	2MB/core, 8-9ns
RAM	TB+, 75ns
SSD	PB+, 10us

https://www.newegg.com/crucial-16gb-260-pin-ddr4-so-dimm/p/N82E16820156081



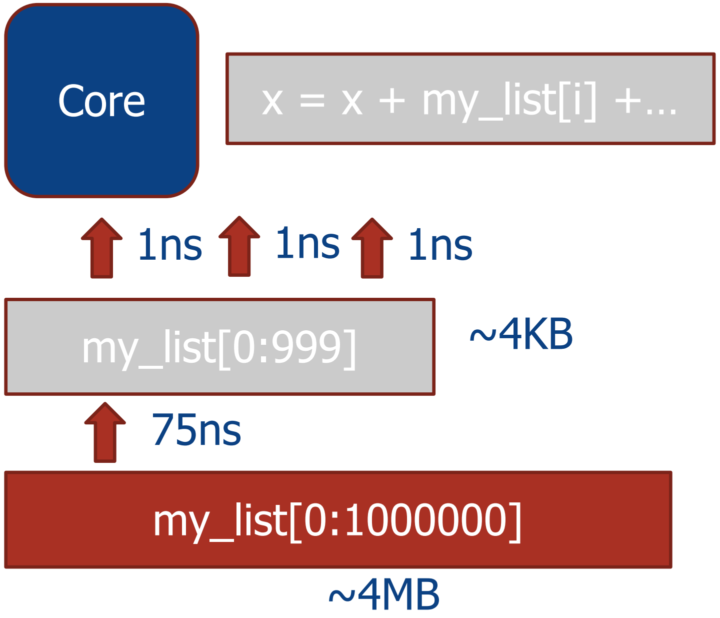
https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds/dc-p4500-series/

https://tinyurl.com/cis545-lecture-02-07-22

How Are Caches Used? (We Are Simplifying to 1 Level)

```
# Create an array 1,000,000 by 1
my_list = np.empty(1000000)

for i in range(0, len(my_list)):
  x = x + my_list[i] + \
    random.random()
```

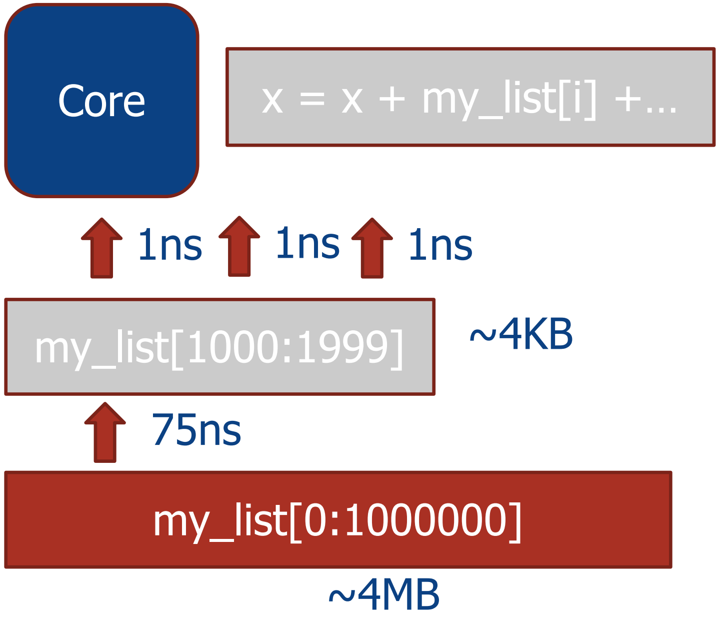


https://tinyurl.com/cis545-lecture-02-07-22

How Are Caches Used? (We Are Simplifying to 1 Level)

```
# Create an array 1,000,000 by 1
my_list = np.empty(1000000)

for i in range(0, len(my_list)):
  x = x + my_list[i] + \
    random.random()
```

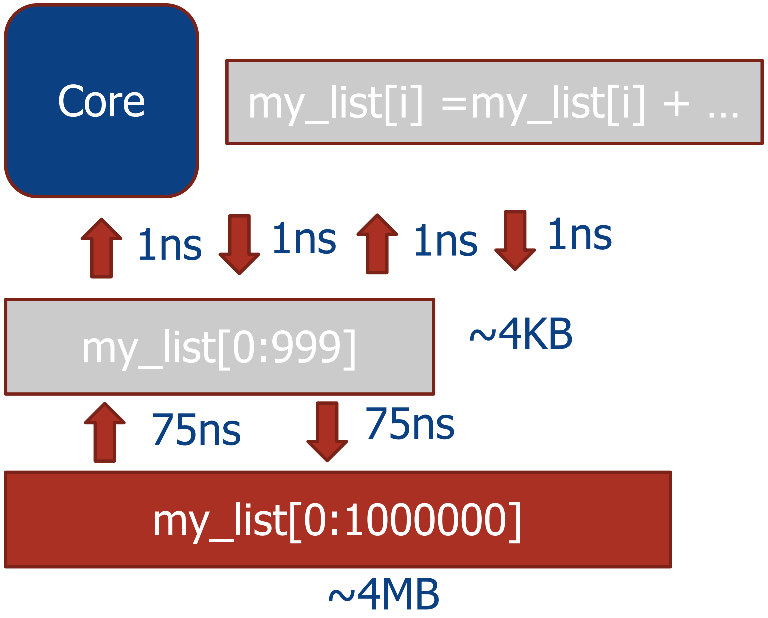


https://tinyurl.com/cis545-lecture-02-07-22

How Are Caches Used? (We Are Simplifying to 1 Level)

```
# Create an array 1,000,000 by 1
my_list = np.empty(1000000)

for i in range(0, len(my_list)):
  my_list[i] = my_list[i] + \
    random.random()
```



https://tinyurl.com/cis545-lecture-02-07-22

Interactive Visualizer (Colin Scott):

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Original list from Jeff Dean and Peter Norvig (Google)

Latency Numbers Every Programmer Should Know

2019

1ns

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■

Main memory reference: 100ns

1,000ns ≈ 1μs

Compress 1KB wth Zippy: 2,000ns ≈ 2μs

10,000ns ≈ 10μs = ■

Send 2,000 bytes over commodity network: 62ns

SSD random read: 16,000ns ≈ 16μs

Read 1,000,000 bytes sequentially from memory: 4,000ns ≈ 4μs

Round trip in same datacenter: 500,000ns ≈ 500μs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 62,000ns ≈ 62μs

Disk seek: 3,000,000ns ≈ 3ms

Read 1,000,000 bytes sequentially from disk: 947,000ns ≈ 947μs

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

How Do We Take Advantage of the Memory Hierarchy?

- Focus on data reduction – filter rows and columns
- Put the most frequently used items together
- Find ways to pass over the data once, instead of many times
- Some of this is done by our big data engines – but there are other factors in *optimization* we'll see shortly

https://tinyurl.com/cis545-lecture-02-07-22

Recap: Consider the Memory Hierarchy When Thinking about Scale

- Accessing data in **predictable ways** is better
  - CPU predicts and “pre-fetches” the data into caches
  - Repeated requests to related data keep memory in the caches
- Smaller “memory footprints” are better (at tension with the previous)
- Packing multiple data values into the same memory, and applying a repeated computation, is better

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771492/ (07B)

How many instructions can a typical machine run in one second, at peak?

a. billions (10^9)
b. trillions (10^{12})
c. thousands (10^3)
d. millions (10^6)

How does a cache help with performance?

e. it stores all of system memory in a faster place
f. it stores the contents of the registers
g. it allows us to amortize expensive memory fetches across multiple requests
h. it stores all of the disk state

https://tinyurl.com/cis545-lecture-02-07-22

Optimizing Relational Expressions to Improve Performance

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

Roadmap for Data Processing

https://tinyurl.com/cis545-notebook-03

- How computer architecture and memory affect performance
- Optimizing relational algebra to minimize memory costs
- Optimizing join orders
- Algorithmic techniques

https://tinyurl.com/cis545-lecture-02-07-22

Big Data Takes A Long Time to Process

Even simple operations are expensive at scale…

```
%%time
# 100,000 records from linkedin
linked_in = open('linkedaa')

people = []

for line in linked_in:
    person = json.loads(line)
    people.append(person)

people_df = pd.DataFrame(people)
people_df[people_df['industry'] \
          == 'Medical Devices']
```

_id	name	locality	skills	industry	
in-00000001	{'family_name': 'M ', 'given_name': 'D...	United States	[Key Account Development, Strategic Planning, ...	Medical Devices	E
in-13806219531	{'family_name': 'G ', 'given_name': 'T }'	China	[ISO 13485, Medical Devices]	Medical Devices	

```
CPU times: user 58.2 s, sys: 1min 57s, total: 2min 55s
Wall time: 3min 19s
```

What makes this so slow?

- Scanning through memory
- Parsing strings
- Updating the list data structure
- Converting to Pandas DF

https://tinyurl.com/cis545-lecture-02-07-22

Optimizing for Costs

1. Reduce data as soon as possible… "Push down" operations!

   Select (filter) at the earliest point possible

   - Less fetching from disk (if relevant), updating of data structures, processing of rows!

   Project once columns aren't needed: Data better fits into the CPU cache

```
%%time
linked_in = open('linkedaa')
people = []
for line in linked_in:
    person = json.loads(line)
    if 'industry' in person and \
       person['industry'] == \
               'Medical Devices':
        people.append(person)

people_df = pd.DataFrame(people)
```

_id	name	locality	skills	industry	
in-00000001	{'family_name': 'M ', 'given_name': 'D...	United States	[Key Account Development, Strategic Planning, ...	Medical Devices	E
in-13806219531	{'family_name': 'G ', 'given_name': 'T }	China	[ISO 13485, Medical Devices]	Medical Devices	

```
CPU times: user 30.2 s, sys: 11.5 s, total: 41.7 s
Wall time: 43.2 s
```

https://tinyurl.com/cis545-lecture-02-07-22

SQL Does this Automatically (and Doesn’t Parse Every Time)

```
%%time

pd.read_sql_query('select * from people where industry="Medical Devices"', conn)
```

```
CPU times: user 15.6 ms, sys: 109 ms, total: 125 ms
Wall time: 137 ms
```

_id	locality	industry	summary
in-00000001	United States	Medical Devices	SALES MANAGEMENT / BUSINESS DEVELOPMENT / PROJ...
in-13806219531	China	Medical Devices	

https://tinyurl.com/cis545-lecture-02-07-22

Even Better: Indexing Speeds Selection

Can we speed up fetches for specific data, e.g., people by industry?

An **index** is a map from an **index key** to a **set of values**

- It allows us to directly find matches to the key without scanning the data
- Can be in-memory or on disk

Two types of indices:

- **Tree** indices (B+ Trees) allow us to find all values <= key, >= key, = key
- **Hash** indices allow us to look up all values equal to the key

Q: Which is a **dict** in Python?

https://tinyurl.com/cis545-lecture-02-07-22

```
conn = sqlite3.connect('linkedin.db')

conn.execute('begin transaction')
conn.execute("create index people_industry on people(industry)")
conn.execute('commit')
```

```
%%time

pd.read_sql_query('select * from people where industry="Medical Devices"', conn)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 26.9 ms
```

To Consider

Can indexing also speed projection?

https://tinyurl.com/cis545-lecture-02-07-22

The Summary So Far: Selections and Projections

- A good rule of thumb is to “push down” selection and projection operations
- SQL DBMSs do this automatically
- Index data structures allow us to evaluate predicates on a **key** and directly return the matches
  - Hash indices help with exact-matches (we’ll revisit this soon)
  - Tree indices (B+ Trees in relational DBMSs) help with equality and inequality

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771528 (07C)

How does "pushing down" filter conditions typically help performance? Choose the best answer.

a. Reduces memory "footprint" and number of operations
b. Requires an index
c. Reduces number of operations only
d. Reduces memory "footprint" only

A typical index:

e. requires time to access data that's linear in the number of rows in the table
f. requires roughly constant time for every access
g. requires time to access data that's linear in the number of columns in the table
h. doesn't work unless all data fits in system memory

https://tinyurl.com/cis545-lecture-02-07-22

Join Ordering and Optimization

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

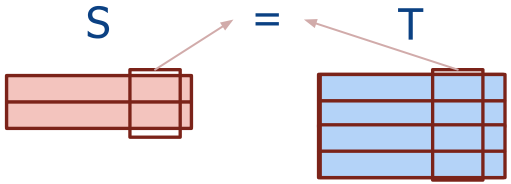
CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

Recall Join
Compares all Pairs of Tuples for a Match

S join T on s_on = t_on:



How we implement it:

```
for every tuple s in S
   for every tuple t in T
      if s[s_on] == t[t_on] then combine & return
```

https://tinyurl.com/cis545-lecture-02-07-22

Joins: Expensive Operations

```
def merge(S,T,s_on,t_on):
    ret = pd.DataFrame()

    for s_index in range(0, len(S)):                  # Rows in S
        for t_index in range(0, len(T)):              # Rows in T
          if S.loc[s_index, s_on] == T.loc[t_index, t_on]:
                ret = ret.append(S.loc[s_index].append(T.loc[t_index]),\
                                 ignore_index=True)
    return ret
```

How many operations does this require?

How big is the output?

https://tinyurl.com/cis545-lecture-02-07-22

Using Our Intuitions in Order of Evaluation (Abstract Example)

Suppose we have three simple Dataframes describing people at Penn (roughly 10.1K undergrad, 10K grad/professional, 0.1K both, 10K faculty/staff):

people(id, name) **category**(id, grad_undergrad_facstaff) **grade**(id, sem, score)

30K people *30.1K entries (G, U, F/S)* *450K entries*

1. Roughly how much work is it to do:

~10K

300M comp, ~10K res

```
people.merge(category[category['id']=='G']).merge(grade)
```

4.5B comp, ~150K res

2. Roughly how much work is it to do:

~10K

```
people.merge(grade).merge(category[category['id']=='G']))
```

13.5B comp, ~450K res

4.5B comp, ~150K res

https://tinyurl.com/cis545-lecture-02-07-22

Looking at Order of Evaluation on Our Real, LinkedIn Dataset

```
people_df = pd.read_sql_query('select * from people limit 500', conn)
experience_df = pd.read_sql_query('select * from experience limit 5000', conn)
skills_df = pd.read_sql_query('select * from skills limit 8000', conn)

temp = merge(people_df, experience_df, '_id', 'person')  # 2228 rows
mktg_df = skills_df[skills_df['value'] == 'Marketing'][['person']] # 23 rows
```

```
%%time
merge(merge(people_df, exp

Merge compared 2500000 tup
Merge compared 51244 tuple
CPU times: user 32.4 s, sy
Wall time: 32.4 s
```

```
%%time
merge(merge(people_df, mktg_df, '_id', 'person'), experience_df, '_id', 'person')

Merge compared 11500 tuples
Merge compared 85000 tuples
CPU times: user 1.23 s, sys: 31.2 ms, total: 1.27 s
Wall time: 1.23 s
```

https://tinyurl.com/cis545-lecture-02-07-22

Do We Have to Choose Orders Manually?

Order of evaluation matters because intermediate result sizes matter

SQL (and query optimization) decide for us!

```
conn.execute('create view people500 as select * from people limit 500')
conn.execute('create view experience5000 as select * from experience limit 5000')
conn.execute('create view skills8000 as select * from skills limit 8000')

pd.read_sql_query('select * from (people500 join experience5000 on _id=person) pe join ' + \
                  'skills8000 sk on pe._id=sk.person and sk.value="Marketing"', conn)
```

```
CPU times: user 0 ns, sys: 15.6 ms, total: 15.6 ms
Wall time: 20.9 ms
```

https://tinyurl.com/cis545-lecture-02-07-22

Do We Have to Do this Manually?

Order of evaluation matters because intermediate result sizes matter

SQL (and query optimization) decide for us!

```
conn.execute('create view people500 as select * from people limit 500')
conn.execute('create view experience5000 as select * from experience limit 5000')
conn.execute('create view skills8000 as select * from skills limit 8000')

pd.read_sql_query('select * from (people500 join skills8000 on _id=person) ps join ' + \
                  'experience5000 ex on ps._id=ex.person and value="Marketing"',
conn)
```

```
CPU times: user 15.6 ms, sys: 0 ns, total: 15.6 ms
Wall time: 18 ms
```

https://tinyurl.com/cis545-lecture-02-07-22

Recap

- Joins are expensive, requiring a quadratic (in the number of inputs) number of comparisons
- Cleverly ordering our joins to reduce intermediate result sizes makes a huge performance difference
- SQL databases can choose this automatically for us

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771532 (07D)

If we join R (1,000 tuples) and S (1,000) tuples, our join result will be:

a. anywhere from 0 to 1 million tuples
b. 1,000 tuples
c. 10,000 tuples
d. 25,000 tuples

If we are joining three tables, our best strategy is to

e. choose a join order that minimizes the intermediate result
f. choose a join order by random sampling
g. join the first table with the second, then the third
h. choose a join order that maximizes the intermediate result

https://tinyurl.com/cis545-lecture-02-07-22

Improving Algorithmic Efficiency

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics

Portions of this lecture have been contributed to the OpenDS4All project, piloted by Penn, IBM, and the Linux Foundation

https://tinyurl.com/cis545-lecture-02-07-22

Two Key Ideas

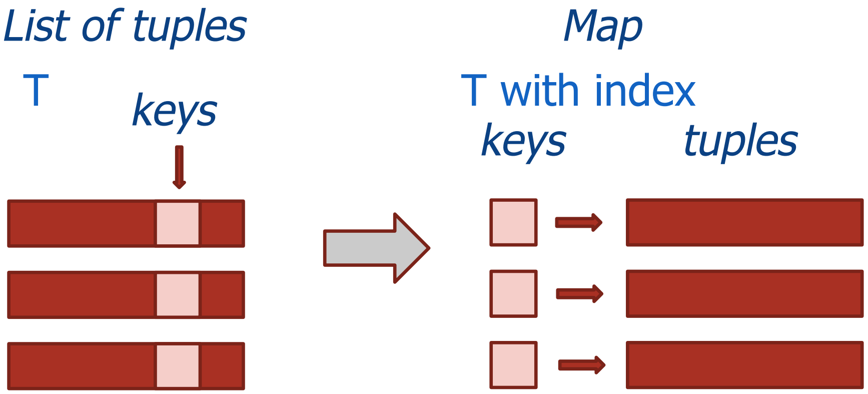
- To this point, we've looked at reducing data
- Now: two ideas for improving performance by changing how we access data
  - Dictionaries / in-memory indices / maps
  - Buffering / blocks

https://tinyurl.com/cis545-lecture-02-07-22

Pandas Merge Is Based on Exact-Matches

Can we find exact-matches between the values in one tuple (from some table S) with another tuple (from some table T)?

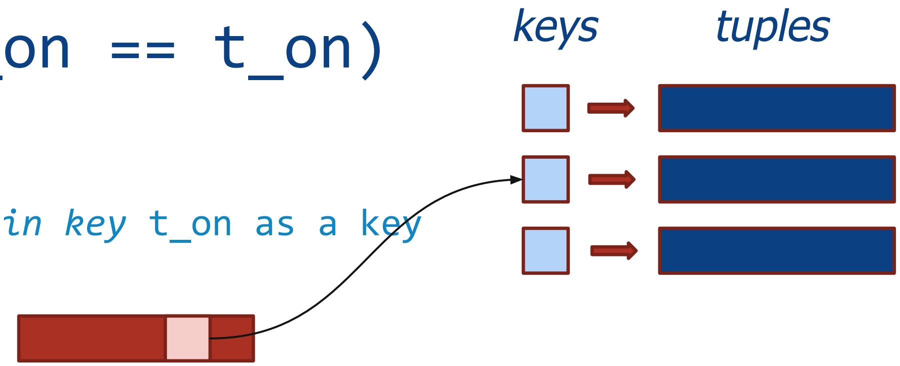
We had a way of doing fast lookups: maps from keys to values (i.e., dicts, indices)



https://tinyurl.com/cis545-lecture-02-07-22

Rethinking Join
(For Equality)

Given `S join T on (s_on == t_on)`




```
index Dataframe T, using the join key t_on as a key
for each tuple s in S:
   Pull out s_on.
   Find matches in the index where t_on = s_on.
   Combine s with the matches and return.
```

https://tinyurl.com/cis545-lecture-02-07-22

A More Efficient Equality Join

```
def merge_map(S,T,s_on,t_on):
    ret = pd.DataFrame()
    T_map = {}
    # Take
    # make
    for t_i
      T_map[

    # Now f
    for s_i
        cou
        if S.loc[s_index, s_on] in T_map:
                ret = ret.append(S.loc[s_index].\
       appe
       igno

return ret
```

```
%%time

# Here's a test join, with people and their experiences.  We can see how many
# comparisons are made
merge_map(experience_df, people_df, 'person', '_id')
```

```
Merge compared 5500 tuples
CPU times: user 7.12 s, sys: 0 ns, total: 7.12 s
Wall time: 7.14 s
```

```
merge(people_df, experience_df, '_id', 'person')
```

```
Merge compared 2500000 tuples
CPU times: user 30.8 s, sys: 0 ns, total: 30.8 s
Wall time: 30.9 s
```

https://tinyurl.com/cis545-lecture-02-07-22

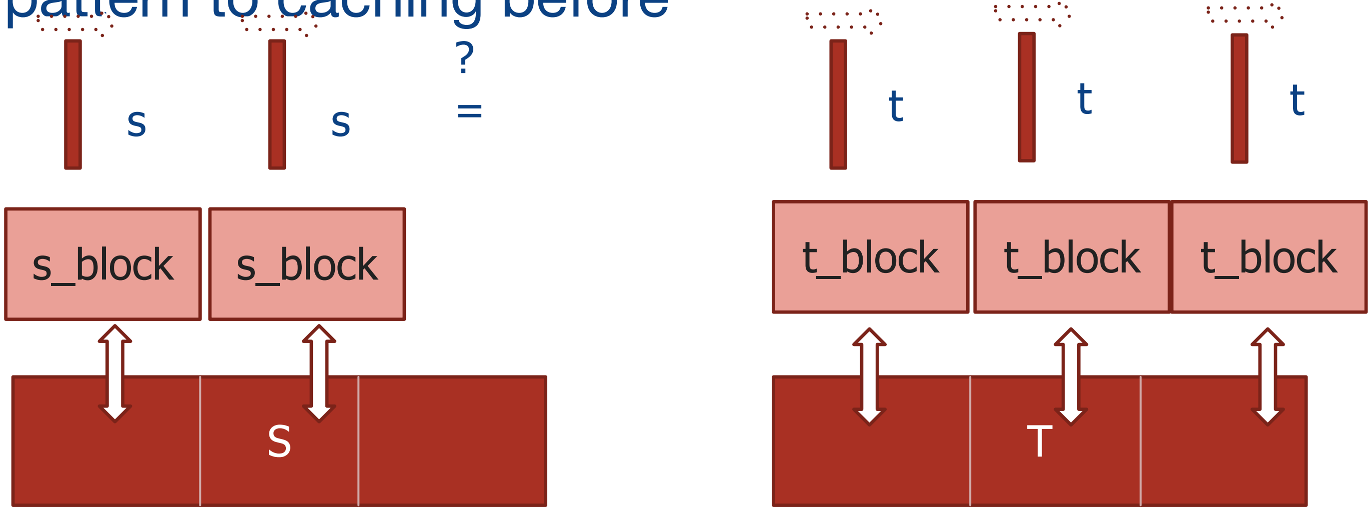
For Further Thought

How does this work if you are joining on more than one field? (first + last name)?

https://tinyurl.com/cis545-lecture-02-07-22

Suppose We Have Big Data, so S and T Don't Fit in Memory!

We need to read a *block* at a time, following a similar pattern to caching before



https://tinyurl.com/cis545-lecture-02-07-22

Joins with Big Data
Need to Read a Table in Blocks

```
# Pseudocode
def merge_on_disk(S,T,s_on,t_on):
    ret = []
    for s_block in blocks(S):
         for t_block in blocks(T):
              for s in s_block:
                  for t in t_block:
                   if s[s_on] == t[t_on]:
                          ret.add(s.append(t))

    return ret
```

Now we read blocks(S) * blocks(T) pages, and compare each s and t from these
100us per block read, 75ns per comparison!!!

https://tinyurl.com/cis545-lecture-02-07-22

Beyond the Relational Algebra

- For big data, sometimes we'll need to supply our own operations
  - Functions to be called via apply (or applymap)
  - Functions that take collections of data
- Again, we'll want to rely on using maps and indices to reduce data usage, and intelligent buffering

https://tinyurl.com/cis545-lecture-02-07-22

Brief Review

https://canvas.upenn.edu/courses/1636888/quizzes/2771493 (07E)

If we use a dictionary / map / hash table to join two tables `S` and `T`, our cost is proportional to

a. the product of the cardinalities (numbers of rows) of `S` and `T`
b. the cardinality (number of rows) of the bigger of `S` and `T`
c. the sum of the cardinalities (numbers of rows) of `S` and `T`
d. the square root of the sums of the sizes of the tables, squared

How does buffering or blocked access help speed performance when tables are bigger than memory?

e. we reduce the size of the index
f. we reduce the overall number of disk fetches
g. we reduce the size of the join output
h. we reduce the overall number of join comparisons

https://tinyurl.com/cis545-lecture-02-07-22

Summary: Making Dataframes / Queries Efficient

General rule of thumb for efficiency: consider order of evaluation

- Minimize intermediate results
- For Pandas, "ballpark estimate" which order is better
- SQL database optimizes using statistical information it collects on tables!

For some joins, can use faster algorithm:

- Joins, by default, iterate over both tables
- An *index* makes the lookups more efficient IF the join is on the *index key*

https://tinyurl.com/cis545-lecture-02-07-22