THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

12. Transactions and Beyond RDBMS

CSCI 2541W Database Systems & Team Projects

Gabe

Includes slides from Prof. Bhagi Narahari and from Wood

Class support

We have ~12 hours of office hours every week

If you need help, come to us!

Find the balance:

- Problem solve, learn things on your own, practice debugging
- but get help when you aren't making progress or aren't sure what to try!

The TAs/UTAs/LAs are an amazing resource for you

- Maybe you should consider being one next year?!

Virtual Environments (venv)

Create a new virtual environment

```
# macOS/Linux
# You may need to run sudo apt-get install python3-venv first
python3 -m venv .venv

# Windows
# You can also use py -3 -m venv .venv
python -m venv .venv
```

Load the environment

```
# macOS/Linux
source .venv/bin/activate
# Windows
.venv\Scripts\activate.bat
```

Flask Auto-Reload

Do you save file, kill flask server, start flask server after every change? Try this!

FLASK_ENV=development python3 main.py

```
 * Serving Flask app "app" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 259-217-934
127.0.0.1 - - [19/Mar/2021 15:36:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Mar/2021 15:36:24] "GET /favicon.ico HTTP/1.1" 404 -
 * Detected change in '/Users/timwood/flask-data/main.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 259-217-934
127.0.0.1 - - [19/Mar/2021 15:36:35] "GET / HTTP/1.1" 200 -
^C
```

VS Code Tips

Get familiar with this editor! It’s pretty great!

Do you know how to?

1. Jump to the definition of a function?
2. Find all the references to a variable?
3. Rename a variable in all places?
4. Select the next occurrence of a highlighted word?
5. Quickly switch between files?
6. Autocomplete code snippets?
7. Share your environment with teammate?
8. Comment out the current line?
9. View two files side-by-side?
10. View the changes you’ve made to a file since last commit?

Python Debugger

Bold but true? Using a debugger is the single best way to quickly become a better developer and save yourself lots of time

Easy to use:

- Click left of line numbers to set a break point
- Press F5 to start debugger (or use menus)
- Step through code with buttons
- Use Debug Console to view/edit variables
- Sad: difficult to debug how data gets rendered in templates

Why Relational Databases are great… and awful

Relational Databases

Relational databases are the dominant form of database and apply to many data management problems.

- Over $30 billion annual market in 2017.

Relational databases are not the only way.

Other models:

- Hierarchical model
- Object-oriented
- JSON/YAML
- Graphs
- Key-value stores
- Document models

Relational Database Model

Well developed data model – gained widespread acceptance…eventually!

- Started gaining acceptance in 80’s…took off in 90’s

Many benefits:

- Data-program independence
- Persistence of data – data ‘stays’ on storage
- Manage concurrency in transactions – **transaction processing**
- SQL programming became a standard
- Growth of online businesses / e-commerce meant greater demand for recording and reporting data

What is transaction processing?

A user's program may carry out many operations on the data retrieved from the database; but the DBMS is only concerned about what data is read/written from/to the database

A transaction is the DBMS's abstract view of the user program: sequence of Read/Write to DB

- Ex: Withdraw from bank account: update balance in SQL

Concurrent execution of user programs essential for good performance.

- Keep CPU humming when disk IO takes place.

What is transaction processing?

Purchase Queries (cost = 20):

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

```
# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost
```

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

rry out many operations on the
abase; but the DBMS is only
a is read/written from/to the

s abstract view of the user
d/Write to DB
count: update balance in SQL

ser programs essential for

disk IO takes place.

What is transaction processing?

Purchase Queries (cost = 20):

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

```
# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost
```

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

balance = 100

rry out many operations on the
abase; but the DBMS is only
a is read/written from/to the

s abstract view of the user
d/Write to DB
count: update balance in SQL

er programs essential for

disk IO takes place.

What is transaction processing?

Purchase Queries (cost = 20):

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

```
# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost
```

balance = 100

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

balance = 80

rry out many operations on the
abase; but the DBMS is only
a is read/written from/to the

s abstract view of the user
d/Write to DB
count: update balance in SQL

programs essential for

k IO takes place.

What is transaction processing?

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

```
# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost
```

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

```
# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost
```

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

What is transaction processing?

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -=

balance = 100

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -=

balance = 100

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

What is transaction processing?

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

balance = 80

What is transaction processing?

Purchase Queries:

SELECT balance FROM bank_accnt AS ba WHERE ba.uid = 10

```
# python code gets pybalance
cost = 20
if pybalance > cost:
    pybalance -= cost
```

UPDATE bank_account AS ba SET **ba.balance = pybalance** WHERE ba.uid = 10

Free Stuff is great!

Until you lose your job cause you’re giving customers free stuff ;-(

balance = 80

Concurrency ….

Users submit Transactions; assume each executes by itself

- Concurrency achieved by DBMS which interleaves actions (read/write of DB objects) of different Transactions
- Each Transaction must leave the DB in a consistent state (if DB is consistent when Transaction begins)

How to interleave operations from different Transactions (programs) which may share the same data?

- Ex: Two (or more) students registering for same course

What happens if system crashes – how to recover to a consistent state?

Big idea: ACID Properties in RDBMs

A______________

C______________

I______________

D______________

What properties are important for Transactions?

Big idea: ACID Properties in RDBMs

Atomicity: all actions in Transaction happen or none happen

Consistency: if each Transaction is consistent (maintains data integrity rules), and DB starts in consistent state then it ends consistent

Isolation: Execution of one Transaction isolated from others (they act like they execute one after the other)

Durability: if a Transaction commits (completes), its effects persist

Meeting the **ACID** Test:

- Concurrency controller guarantees **consistency** and **isolation**
- Logging & recovery for **atomicity** and **durability**

Concurrency Control..How? Locks!

Conflict occurs when two Transactions try to access the same data item

Associate a "**lock**" for each shared data item

- Similar to mutual exclusion (MUTEX)
- To access a data item, check if it is unlocked else wait
- Need to worry about the type of operation: Read or Write
  - Leads to Lock Modes: Shared Lock(S) for Reads only and Exclusive Lock(X) for Writes
- Providing both consistency and performance is hard!

You'll learn more in OS

Recoverability: Logging

Record the operations of each transaction into a log

- Only consider a transaction complete if a "commit" operation is appended to the log
- After a commit, we can update the actual data file

If system crashes, read from log file to rollback to a consistent state

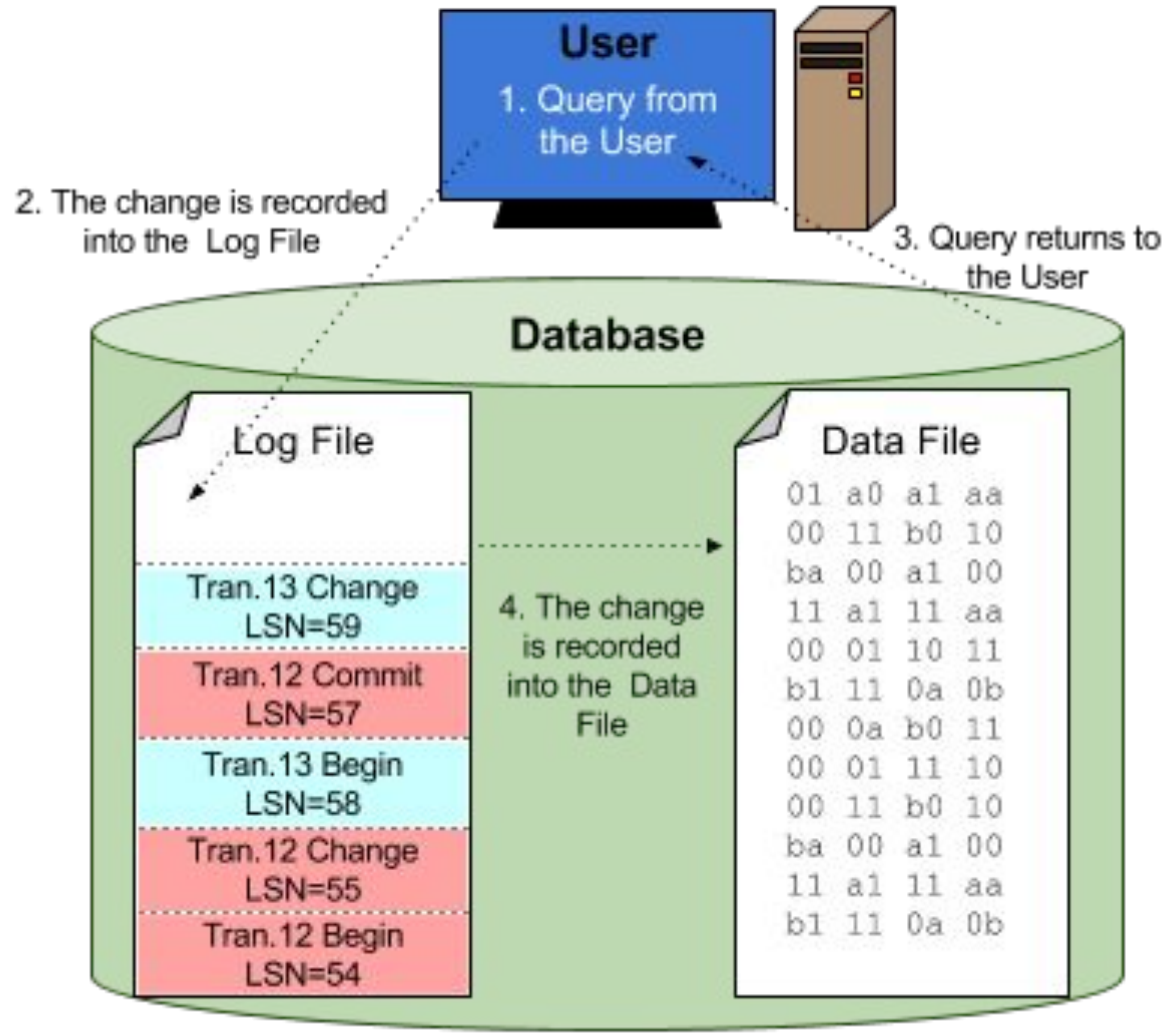


image from
https://sqlbak.com/academy/transaction-log

So why do we need something other than Relational DBs?

Database application trends?

Data trends?

Any guesses for how data or applications have been changing in last 10 years?

How to store a Customer…?

In 1990

In 2000

In 2005

In 2020

For each change:

- ALTER TABLE Customer… add columns
- Take DB offline, change schema, repopulate DB, fix any inconsistencies…

Instead of adding Columns…

How could we add new information such as mobile phone to our DB without adding columns to an existing table?

Instead of adding Columns…

Could create separate tables and use Joins to combine them

- Customer JOIN Phone JOIN MobilePhone JOIN Gender JOIN Email JOIN Twitter JOIN Instagram JOIN …

But doing lots of joins is expensive and messy

- Lots of fields may be NULL, need to be careful about consistency

If our data is constantly evolving or every record has a variable structure, RDBMS may not be the right choice!

RDBMS Pros and Cons

Strengths of Relational DBs?

Weaknesses of Relational DBs?

RDBMS Pros and Cons

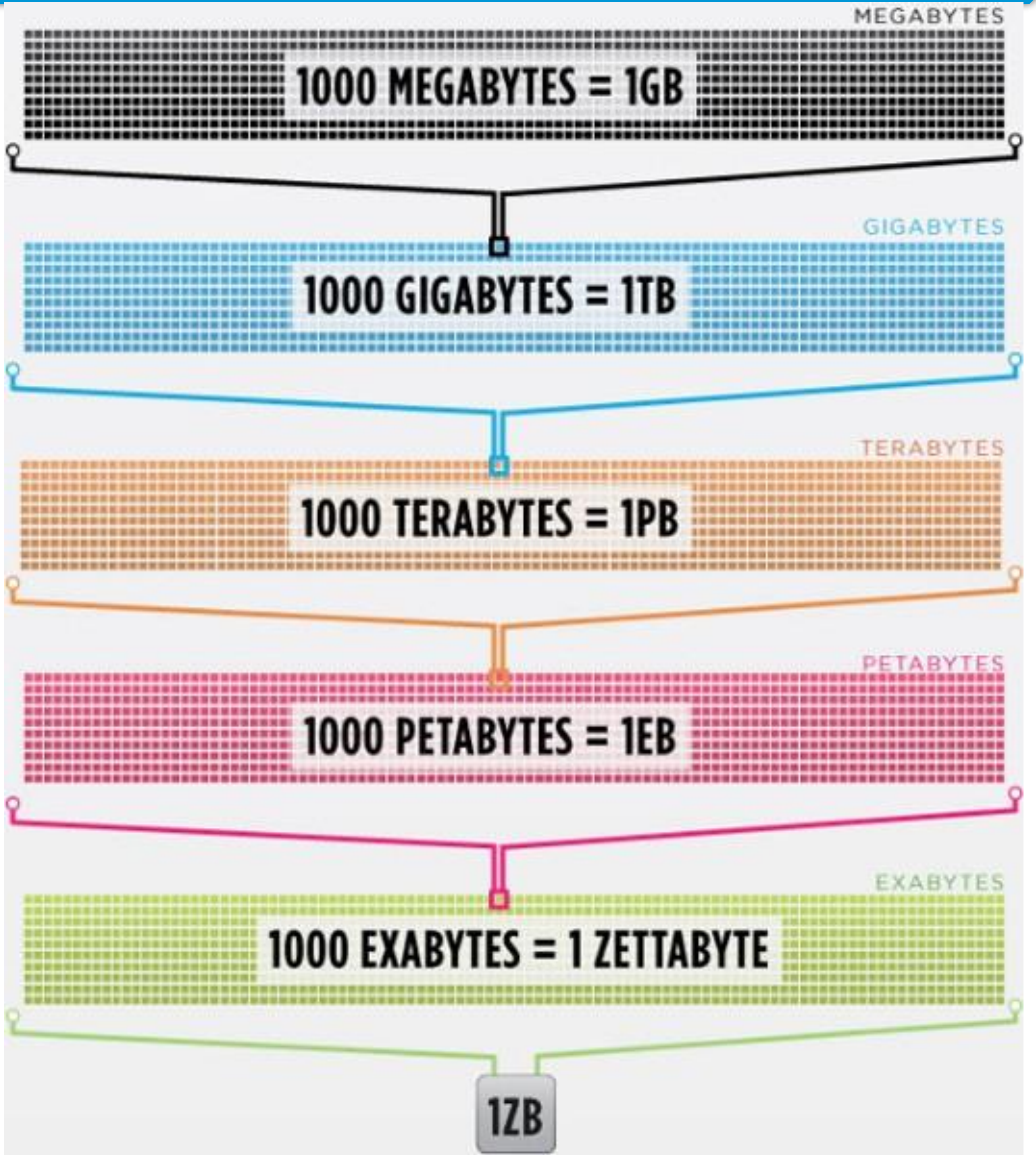
Strengths	Weaknesses
ACID properties (Atomic, Consistent, Isolated, Durable)	Strong consistency properties are expensive to enforce
Widespread/standardized	Strict structure is difficult to adapt
	Some expensive features are not needed by some apps

Trend 1

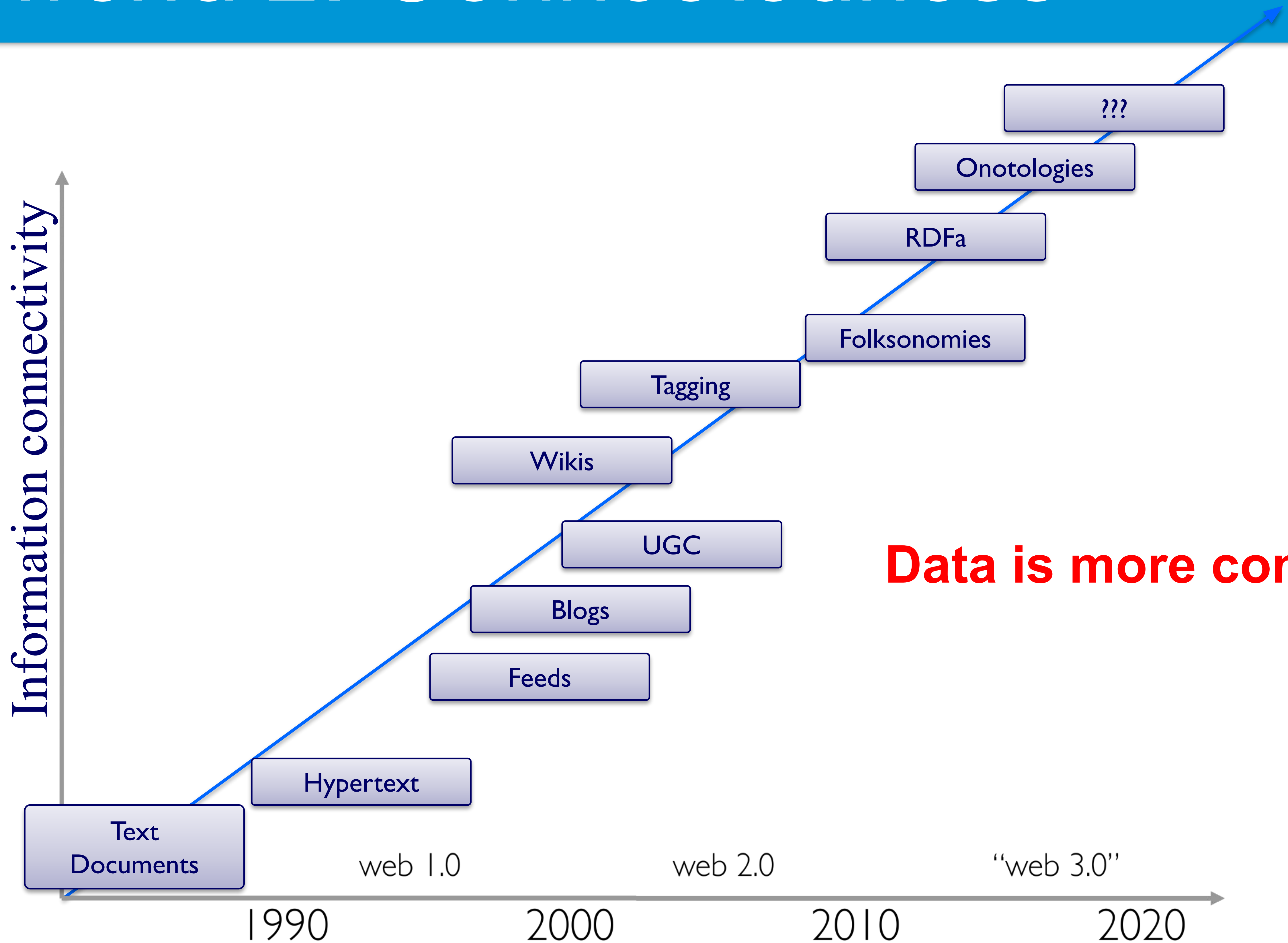
Data is getting bigger:

"Every 2 days we create as much information as we did up to 2003"
– Eric Schmidt, Google in **2010**

Facebook generates
4 Petabytes per day! (2020)



Trend 2: Connectedness



Data is more connected

Trend 3: Data is often Semi-Structured (or no structure)

If you tried to collect all the data of every movie ever made, how would you model it?

Actors, Characters, Locations, Dates, Costs, Ratings, Showings, Ticket Sales, etc.

Relational Databases Challenges

Features of relational databases make them "challenging" for certain problems:

1. Fixed schemas – defined ahead of time, changes are difficult, and lots of real-world data is “messy”. Relational design requires lots of Joins. **So get rid of schemas**
2. Complicated queries – SQL is declarative and powerful but may be overkill. **Instead, do the work in application code**
3. Transaction overhead – Not all data and query answers need to be perfect. **Close enough is sometimes good enough**
4. Scalability – Relational databases may not scale sufficiently to handle high data and query loads or this scalability comes with a very high cost. **Find new ways to scale**

Database Scaling

RDBMS are "scaled up" by adding hardware processing power

- Need more performance? upgrade your machine!

Why is it difficult to **replicate** or **partition** an RDBMS to improve performance **by using multiple computers?**

Let's consider the Python Dictionary

```
myDict = {
  "name": "Maya",
  "address": "156 East 24th street",
  "city": "New York",
  "state":"New York",
  "cars": ["Ford","Honda"]
}
```

Access any **Value** from the dictionary using its **Key**

– Dictionary = Key/Value Store = Hash Table

Suppose we have to add lots and lots more fields...

How could we scale this "database"?

Scaling a Dictionary (KV Store)

A Dictionary (or Key-Value store) can be:

Scaled UP by getting a more powerful server

- Just like RDBMS

Scaled OUT by adding another server and *partitioning* the data

- KV store doesn't need to support queries across objects!
- Consistency is not a problem, easy to exploit parallelism from many servers

Dictionaries can be Nested

A "value" can be a complex data structure of its own!

Each Employee can have several fields within its own dictionary

We can partition the KV store so each server holds a set of Employees

```
employees = {}
employees["Brenda"] = {
  "name": "Brenda Kali",
  "address": "156 East 24th St",
  "city": "New York",
  "state":"New York",
  "cars": ["Ford","Honda"]
}
employees["Jose"] = {
  "name": "Jose Constantino",
  "address": "231 West 181st St",
  "city": "New York",
  "state":"New York",
  "cars": ["Tesla"]
}
...
```

Be careful - key must be unique!

Employee Database

Which is better?!

Two possible structures

RDBMS / SQL

ID	name	address	...
Brenda	Brenda Kali	156 E. 24th St	...
Jose	Jose Constantino	231 W. 181st St	...
...	...	...	...

ID	car
Brenda	Ford
Brenda	Honda
Jose	Tesla

KV Store / Not SQL

```
employees = {}
employees["Brenda"] = {
   "name": "Brenda Kali",
   "address": "156 East 24th St",
   "city": "New York",
   "state":"New York",
   "cars": ["Ford","Honda"]
}
employees["Jose"] = {
   "name": "Jose Constantino",
   "address": "231 West 181st St",
   "city": "New York",
   "state":"New York",
   "cars": ["Tesla"]
}
...
```

It depends!

Do you need to filter employees by where they live?

- Use RDBMS! KV store just knows about the key!

What if each employee has unique set of fields that must be stored?

- Use KV store since internals of an employee are entirely customizable

What if scale of data is really really big?

- Use KV store **IF** you don't need to worry about cross-record consistency or queries

Does this look familiar to anyone?

(Reformatted slightly)

```
{'Brenda': {
  'name': 'Brenda Kali',
  'address': '156 East 24th St',
  'city': 'New York',
  'state': 'New York',
  'cars': ['Ford', 'Honda']},
  'Jose': {
  'name': 'Jose Constantino',
  'address': '231 West 181st St',
  'city': 'New York',
  'state': 'New York',
  'cars': ['Tesla']}
}
```

JSON, XML, etc

'Schema-less' data structure definitions

- Data format, not a full DBMS!

JavaScript Object Notation (**JSON**, pronounced "Jason")

- Serializes (saves) data objects into text form
- Human-readable
- Semi-structured
- Pervasively used in many languages (beyond JS)

Used to transmit most data to/between web services over AJAX/REST interfaces

- Client-side javascript makes a request to server, server responds with JSON data, client updates local browser view

JSON Example

JSON constructs:

- **Values**: number, strings (double quoted), true, false, null
- **Objects**: enclosed in { } and consist of set of key-value pairs (dictionary)
- **Arrays**: enclosed in [ ] and are lists of values
- Objects and arrays can be nested

Example:

```
{"Employees": [{"eno":"E1", "ename":"J. Doe", "title":"EE", "salary":30000, "WorksOn":["P1"]},
               {"eno":"E2", "ename":"M. Smith", "title":"SA", "salary":50000, "WorksOn":["P1","P2"]},
               {"eno":"E3", "ename":"A. Lee", "title":"ME", "salary":40000, "WorksOn":["P3"]}
],
"Projects": [{"pno":"P1", "pname":"Instruments", "budget":150000},
               {"pno":"P2", "pname":"DB Develop", "budget":135000},
               {"pno":"P3", "pname":"Budget", "budget":250000}
]
}
```

JSON Parsers

JSON parser converts JSON file (or string) into program objects (checks syntax)

- In javascript, can call eval() method on variable containing a JSON string

Many languages have APIs to allow for creation and manipulation of JSON objects

Common use:

- JSON data provided from a server (NoSQL or relational) and sent to web client
- Web client uses javascript to convert JSON into objects and manipulate as required

Converters for csv to json

What is NoSQL?

Stands for No-SQL or Not Only SQL??

What is definition….No definition!! But common some characteristics:

Class of non-relational data storage systems

Schema-less: usually do not require a fixed schema nor do they use the concept of joins

Cluster friendliness – ability to run on large number of servers (distributed system / cluster)

All NoSQL offerings relax one or more of the ACID properties

NoSQL - advantages

NoSQL databases are useful for several problems not well-suited for relational databases:

- Variable data: semi-structured, evolving, or has no schema
- Massive data: terabytes or petabytes of data from new applications (web analysis, sensors, social graphs)
- Parallelism: large data requires architectures to handle massive parallelism, scalability, and reliability
- Simpler queries: may not need full SQL expressiveness
- Relaxed consistency: more tolerant of errors, delays, or inconsistent results ("eventual consistency")
- Easier/cheaper: less initial cost to get started

NoSQL is not really about SQL but instead developing data management systems that are not relational.

CAP Theorem..getting around ACID

The CAP Theorem (proposed by Eric Brewer) states that there are three properties of a data system:

- Consistency
- Availability
- Partitions

but you can have at most two of the three properties at a time

- Since scaling out requires partitioning, many NoSQL systems sacrifice consistency for availability/partitioning.

Eventual Consistency - weaker than ACID

- Kind of what it sounds like
- Does not guarantee updates are immediately visible
- But eventually all nodes will agree on a final value

NoSQL (Data) Models

There are a variety of models/systems that are not relational:

- **Column Stores** – represent data in columns rather than rows.
  - Examples: Google BigTable, HBase, Cassandra
- **Key-value stores** – ideal for retrieving specific data records from a large set of data
- **Document stores** – similar to key-value stores except value is a document in some form (e.g. JSON)
- **Graph databases** – represent data as graphs

Related:

- **MapReduce** – technique for large scale data analysis provided by many NoSQL DBMSs

Typical NoSQL API

Basic API access:

- **get**(key) -- Extract the value given a key
- **put**(key, value) -- Create or update the value given its key
- **delete**(key) -- Remove the key and its associated value
- **execute**(key, operation, parameters) -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc).

What is missing compared to SQL?

What do you lose with NoSQL systems?

Joins, group by; order by

- Implement this logic in the application layer (eg Python)

ACID transactions

SQL

Enterprise integration with other relational and SQL-based systems

JDBC/ODBC APIs

familiarity and standards compliance

1. Key-Value Data Model

Key-value stores store and retrieve data using keys. The data values are arbitrary. Designed for "web sized" data sets.

Operations:

- insert(key, value), fetch(key), update(key), delete(key)

Basically just a remote Dictionary / Hash Table / Hashmap

Benefits: high-scalability, availability, and performance

Limitations: single record transactions, eventual consistency, simple query interface

Examples: Cassandra, Amazon Dynamo, Google BigTable, HBase, Redis, memcached

2. Document Store Data model

Document stores are similar to key-value stores but the value stored as a structured document (e.g. JSON, XML).

Can store and query documents by key as well as retrieve and filter documents by their properties

- Not as powerful as SQL

Benefits: high-scalability, availability, and performance

Limitations: same as key-value stores, may cause redundancy and more code to manipulate documents

Examples: CouchDB, SimpleDB, MongoDB, Document DB

3. Graph Databases

Model the data as a graph

Why graph databases? We'll use an example you've come across….

Examples: Neo4J, Flock, ArangoDB.

Question: You want to find the cheapest flight, regardless of number of stops, from Montreal to Seattle

Flight Data stored as Relational Table

Flight_ID	Start_Airport	End_Airport	Cost
1231	Montreal	Seattle	700
1234	Montreal	Chicago	200
1235	Montreal	Boston	100
2123	Boston	Seattle	400
2124	Boston	Chicago	50
3123	Chicago	Seattle	200
3124	Chicago	Boulder	50
4123	Boulder	Seattle	100
….			

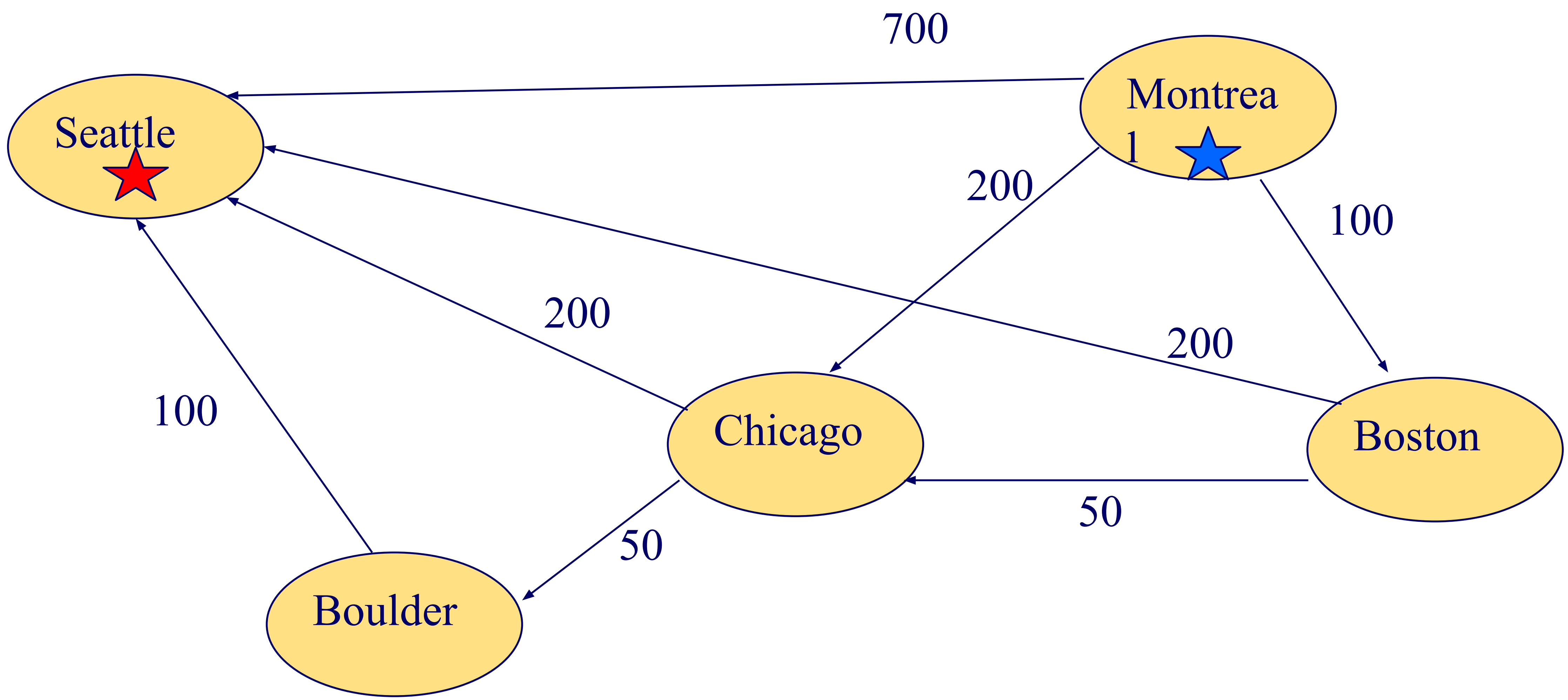
Query for direct flight

```
SELECT Cost
FROM Flights
WHERE Start_Airport='Montreal'
And End_Airport='Seattle';
```

Query for 1-stop flight

```
SELECT (A.Cost + B.Cost)
FROM Flights A,B
WHERE A.Start_Airport='Montreal'
AND A.End_Airport=B.Start_Airport
B.End_Airport= 'Seattle';
```

An Alternate Data Model



How do you find the cheapest flight plan from Montreal to Seattle ?

– Do you know of any algorithms to do this ?

What is a Graph Database?

A database with an explicit graph structure

Each node knows its adjacent nodes

- As the number of nodes increases, the cost of a local step (or hop) remains the same

Captures the richness in connectedness of data

- Social network analytics – much easier when modeled as a graph
- Many problems can be represented as graphs (supply chains, transportation, software function call chains, ...)

Graph Examples

Average number of "hops" between two random Twitter users?

Is Prof. Wood related to....?

Graph Examples

Average number of "hops" between two random Twitter users? **3.43**

Is Prof. Wood related to....?

Mao Zedong 毛澤東 is your 9th great uncle's second great nephew's wife's niece's husband's great grandson's wife's sister's husband's wife's half sister's husband's father's wife's brother's sister's husband's great nephew's ex-partner's 4th husband.

You → **Robert Wood** your father → **John Wood** his father → **Everett Wood** his father → **Hudson Wood** his father → **Julius Wood** his father → **Asenath Caulkins** his mother →
Rufus Caulkins her father → **Judith Tinker** his mother → **Samuel Tinker** her father → **Amos Tinker, Sr.,** his father → **Rebecca DeWolf (Tinker)** his sister →
Edward Nathan DeWolfe her husband → **Mary Lee (de Wolfe)** his sister → **Hannah Griswold** her daughter → **Phoebe "Phebe, Phoeby" Griswold Parsons** her daughter →
Maj. Gen. Samuel H. Parsons her son → **[Mrs.] Parsons** his wife → **Samuel Lee** her brother → **Jeanette Manierre** his daughter → **John Manierre** her husband →
Edward Manierre, tailor his son → **Eva Dawson** his daughter → **Mitchell Dawson, lawyer & poet** her son → **Rose Dawson** his wife →
Emily "Mickey" Hahn, journalist & author her sister → **Shao Xunmei (邵洵美), poet** her husband → **盛佩玉(五)** his wife → **盛昌頤(一)** her father → **盛毓菊(四)** his daughter →
李國芝 her husband → **李經馥** his father → **曾廣璇** his wife → **曾廣銓** her brother → **曾廣珊** his sister → **俞明頤** her husband → **俞明震** his brother → **俞大純** his son →
俞啟威（黃敬） his son → **江青 Jiang Qing** his ex-partner → **Mao Zedong 毛澤東** her husband

Should I be using NoSQL Databases?

NoSQL Data storage systems makes sense for applications that need to deal with **very very large** semi-structured data

- Social Networking Feeds, Data analytics

For most organizational (ecommerce) databases, which are not that large and have low update/query rates, **regular relational databases are usually the right solution**

- Standards, reliable, ACID

Databases for Analytics

Transactional RDMSes:

- Transactions
- Joins
- Retrieving multiple tuple fields
- OLTP – online transaction processing

Transactions for digesting data – analytics!

- Range queries, aggregation
- Operations on a small number of attributes
- OLAP – online analytics processing

Most operations, single attribute?

SELECT AVG(price) FROM purchases
WHERE price > 100 * 60

price	product_name	…	…	…	…	…	…	…	…	purchase _location
200										
…	…	…	…	…	…	…	…	…	…	…
20										

```
struct purchases_tuple {
    int64_t cents;
    char    product_name[1024];
    …
}
struct purchases {
    // this would be dynamic alloc
    struct purchases_tuple rows[N];
}

struct purchases relation;
```

```
int64_t tot = 0, cnt = 0, p;

for (int i = 0; i < N; i++) {
    p = relation.rows[i].cents;

    // What code goes here?


}
return tot / cnt;
```

Most operations, single attribute?

SELECT AVG(price) FROM purchases

WHERE price > 100 * 60

price	product_name	…	…	…	…	…	…	…	…	purchase _location
200										
…	…	…	…	…	…	…	…	…	…	…
20										

```
struct purchases_tuple {
    int64_t cents;
    char    product_name[1024];
    …
}
struct purchases {
    // this would be dynamic alloc
    struct purchases_tuple rows[N];
}

struct purchases relation;
```

```
int64_t tot = 0, cnt = 0, p;

for (int i = 0; i < N; i++) {
    p = relation.rows[i].cents;

    if (p > 100 * 60) {
        tot += p;
        cnt++;
    }
}
return tot / cnt;
```

Most operations, single attribute?

SELECT AVG(price) FROM purchases
WHERE price > 100 * 60

price	product_name	…	…	…	…	…	…	…	…	purchase _location
200										
…	…								…	…
20										

Inefficiencies in this code?

```
struct purchases
    int64_t cent
    char    prod
    …
}
struct purchases {
    // this would be dynamic alloc
    struct purchases_tuple rows[N];
}

struct purchases relation;
```

```
= 0, p;

N; i++) {
s[i].cents;

    if (p > 100 * 60) {
        tot += p;
        cnt++;
    }
}
return tot / cnt;
```

Most operations, single attribute?

SELECT AVG(price) FROM purchases

						..	…	…	…	purchase _location
…		…	…	…	…	…	…	…	…	…
20										

What if
`sizeof(struct purchase_tuple) == BLOCK_SIZE`
… but cents is small (8)!

```
struct purchases_tuple {
    int64_t cents;
    char    product_name[1024];
    …
}
```

```
int64_t tot = 0, cnt = 0, p;

for (int i = 0; i < N; i++) {
    p = relation.rows[i].cents;
```

This accesses memory like this – each memory access is a new block!

`relation`

Most operations, single attribute?

SELECT AVG(price) FROM purchases

What if
`sizeof(struct purchase_tuple) == BLOCK_SIZE`
… but cents is small (8)!

…	…	…	…	purchase_location
…			…	…
20				

```
struct purchases
    int64_t cent
    char    prod
    …
}
struct purchases {
```

```
= 0, p;
N; i++) {
s[i].cents;
if (            {
```

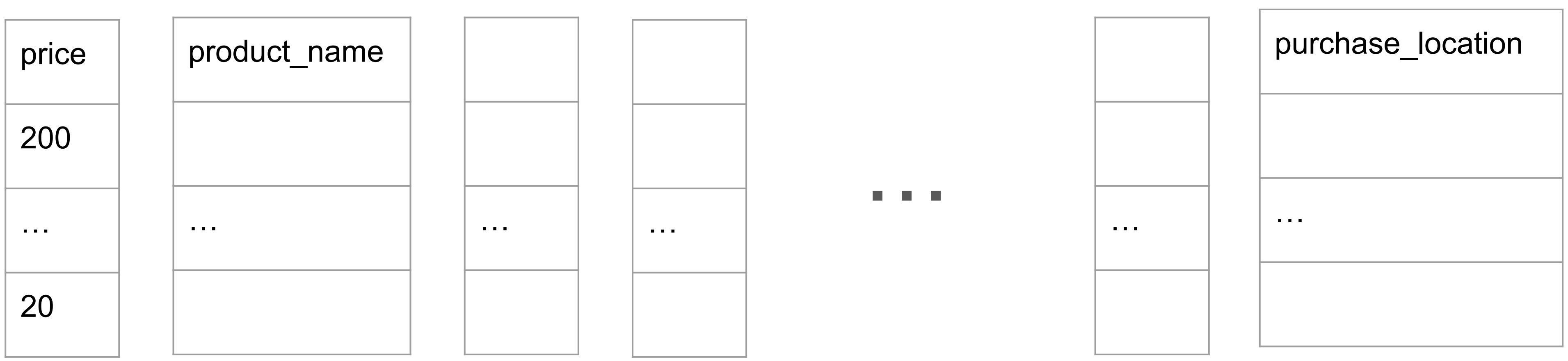
We don't want every access to be on a new block!
Can we make this faster?

This accesses memory like this – each memory access is a new block!

relation

Most operations, single attribute?

SELECT AVG(price) FROM purchases
WHERE price > 100 * 60

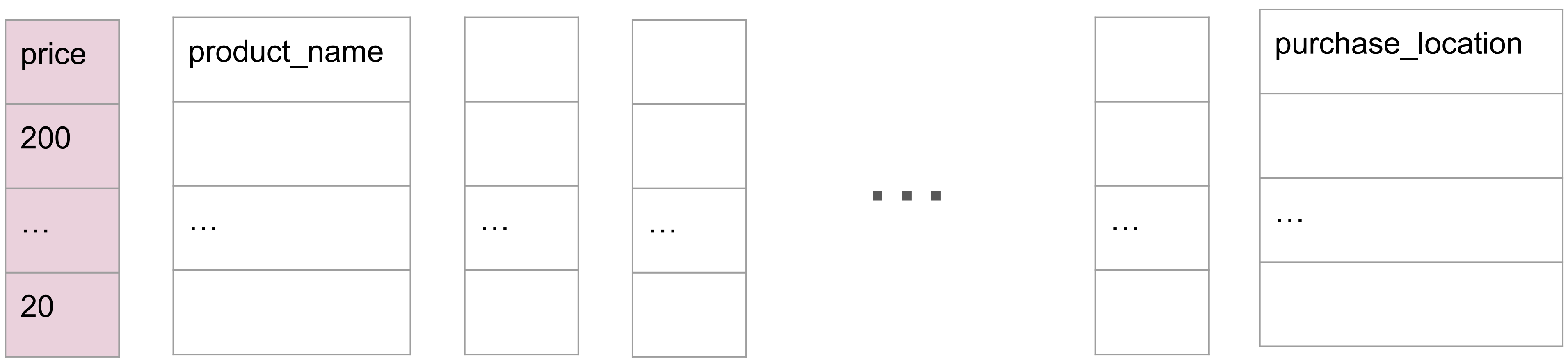



```
struct purchases_cents {
    int64_t cents[N];
}
struct purchases {
    struct purchases_cents cents;
    struct product_name prodname;
    …
}

struct purchases relation;
```

```
int64_t tot = 0, cnt = 0, p;

for (int i = 0; i < N; i++) {
    p = relation.cents.cents[i];

    if (p > 100 * 60) {
        tot += p;
        cnt++;
    }
}
return tot / cnt;
```

Most operations, single attribute?

SELECT AVG(price) FROM purchases
WHERE price > 100 * 60



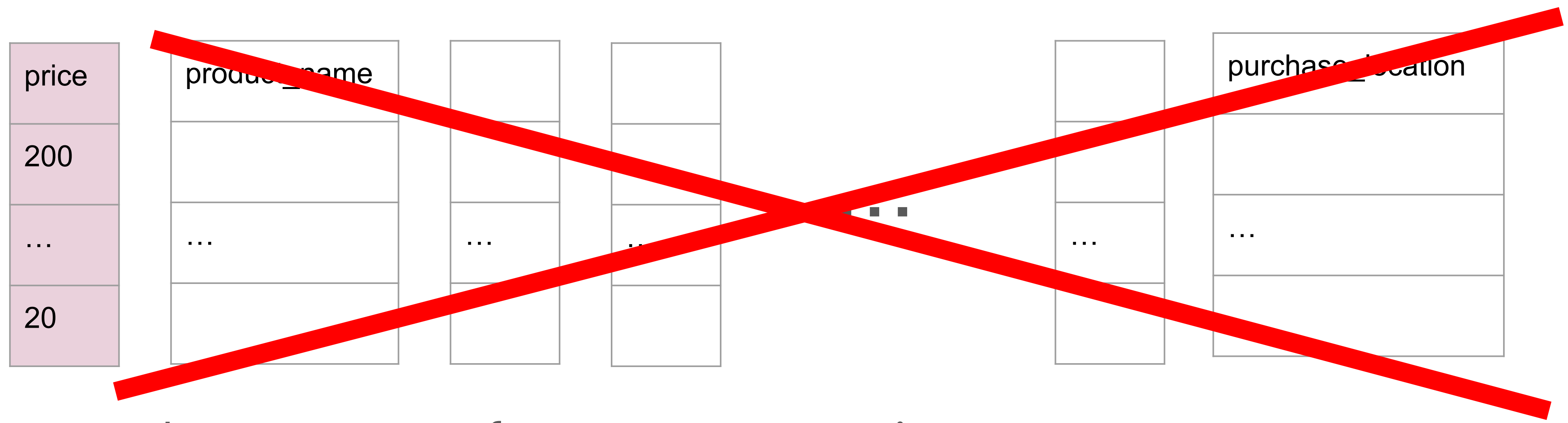

```
struct purchases_cents {
    int64_t cents[N];
}
struct purchases {
    struct purchases_cents cents;
    struct product_name prodname;
    …
}

struct purchases relation;
```

```
int64_t tot = 0, cnt = 0, p;

for (int i = 0; i < N; i++) {
    p = relation.cents.cents[i];

    if (p > 100 * 60) {
        tot += p;
        cnt++;
    }
}
return tot / cnt;
```

Most operations, single attribute?

SELECT AVG(price) FROM purchases

WHERE price > 100 * 60




```
struct purchases_cents {
    int64_t cents[N];
}
struct purchases {
    struct purchases_cents cents;
    struct product_name prodname;
    …
}

struct purchases relation;
```

```
int64_t tot = 0, cnt = 0, p;

for (int i = 0; i < N; i++) {
    p = relation.cents.cents[i];

    if (p > 100 * 60) {
        tot += p;
        cnt++;
    }
}
return tot / cnt;
```

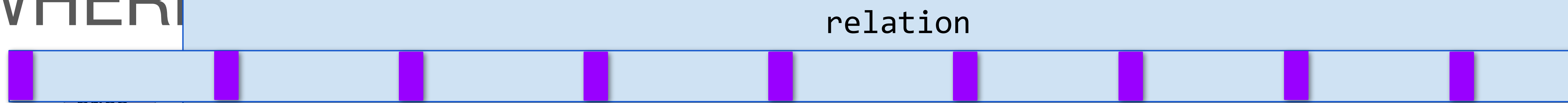
Most operations, single attribute?

SELEC

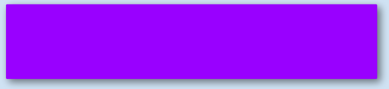
WHERI

price
200
…
20

Previous memory access pattern:



versus using this **column store layout**



Might fit all accesses in a *single block*!

```
struct purc
    int64_t
}
struct purchases {
    struct purchases_cents cents;
    struct product_name prodname;
    …
}

struct purchases relation;
```

```
        p = relation.cents.cents[i];

        if (p > 100 * 60) {
            tot += p;
            cnt++;
        }
    }
    return tot / cnt;
```

OLAP: Column Stores for Analytics

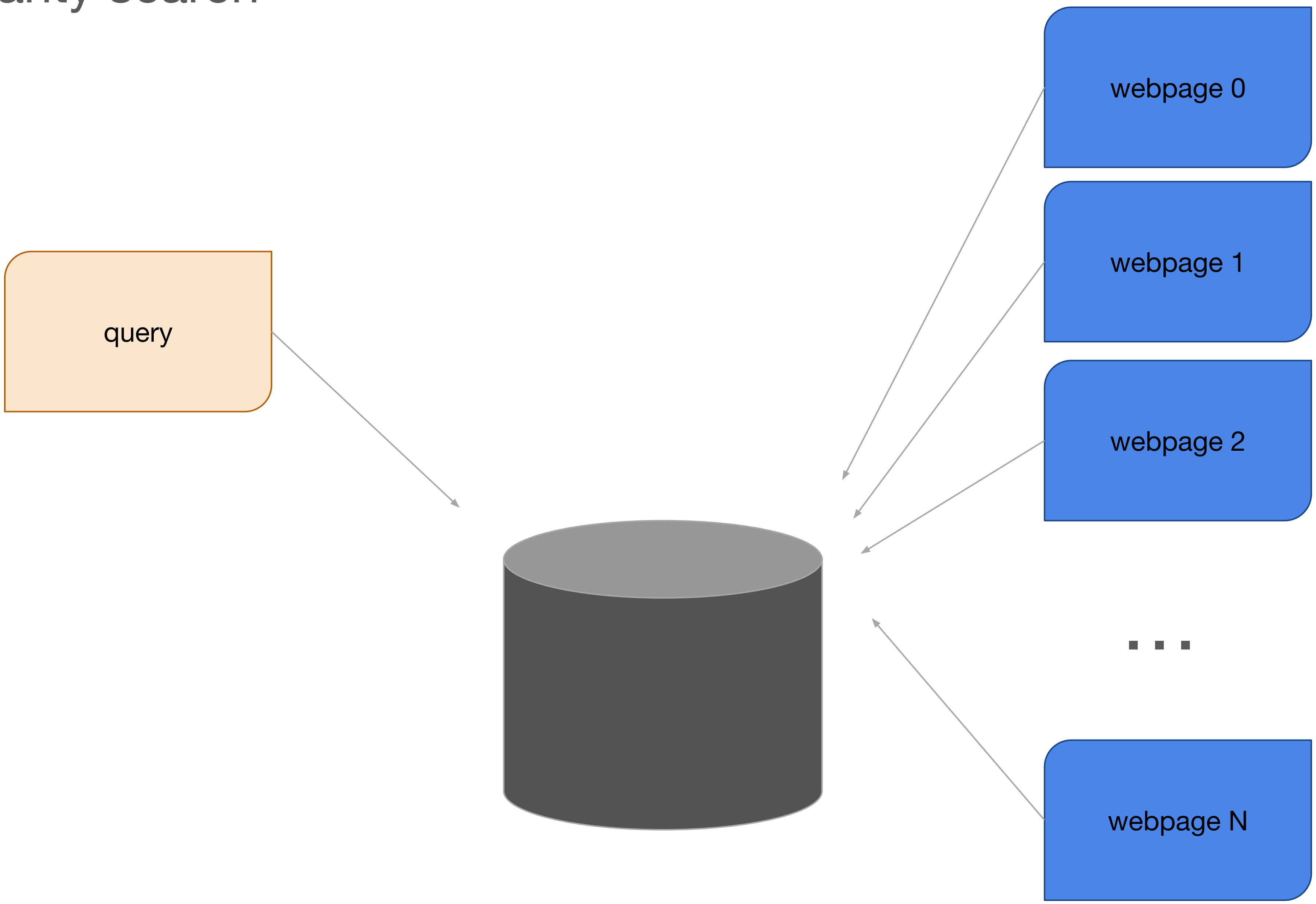
Column stores:

- When queries focus on a small number of columns…
  - store the data in columns!
- Better layout of relation
  - across blocks
  - across cache-lines
- Recall: DBMS *storage engine* optimizes for the hardware!
  - Abstraction is powerful!

Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages “closest” in content to the query!

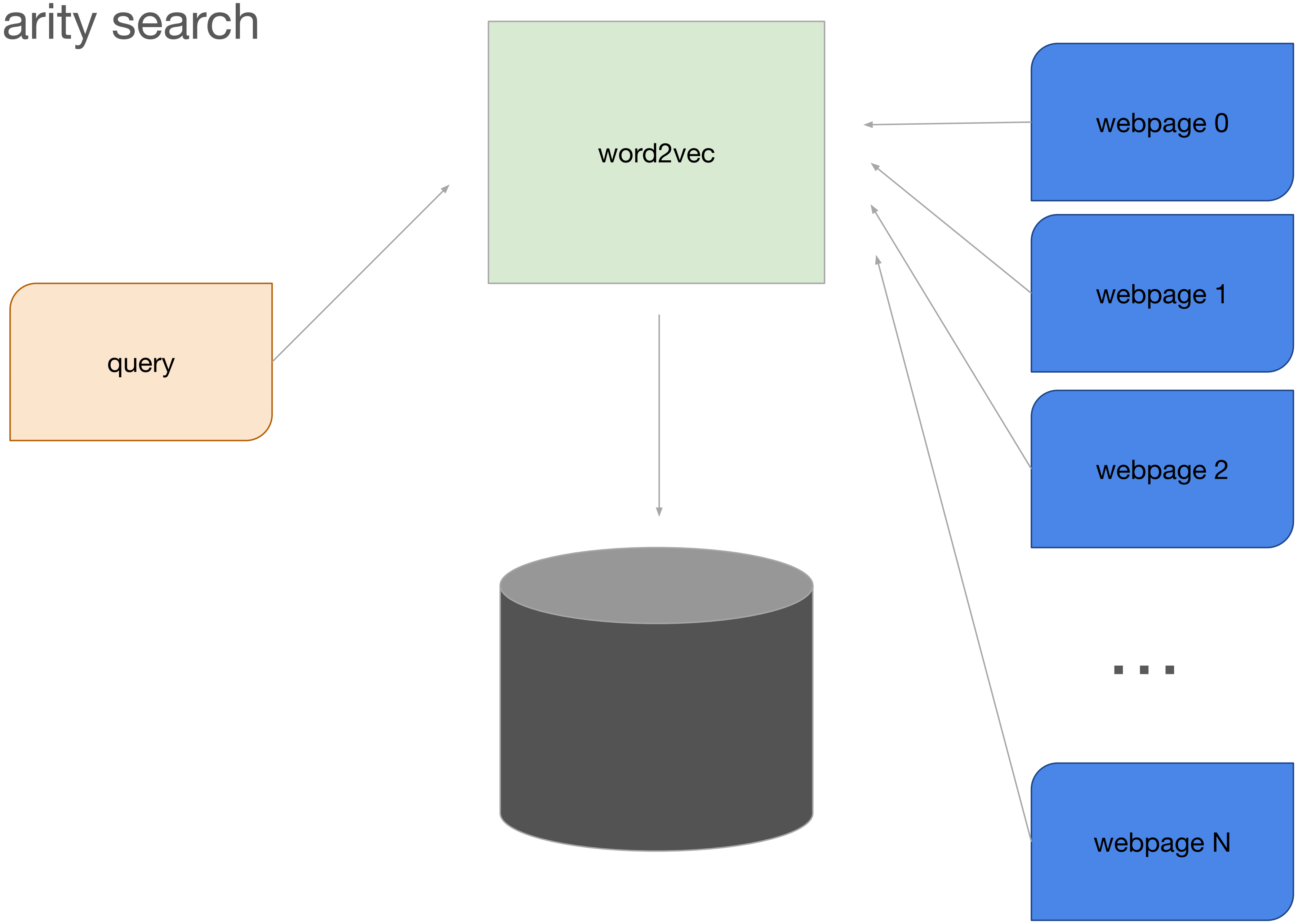
- Similarity search



Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages “closest” in content to the query!

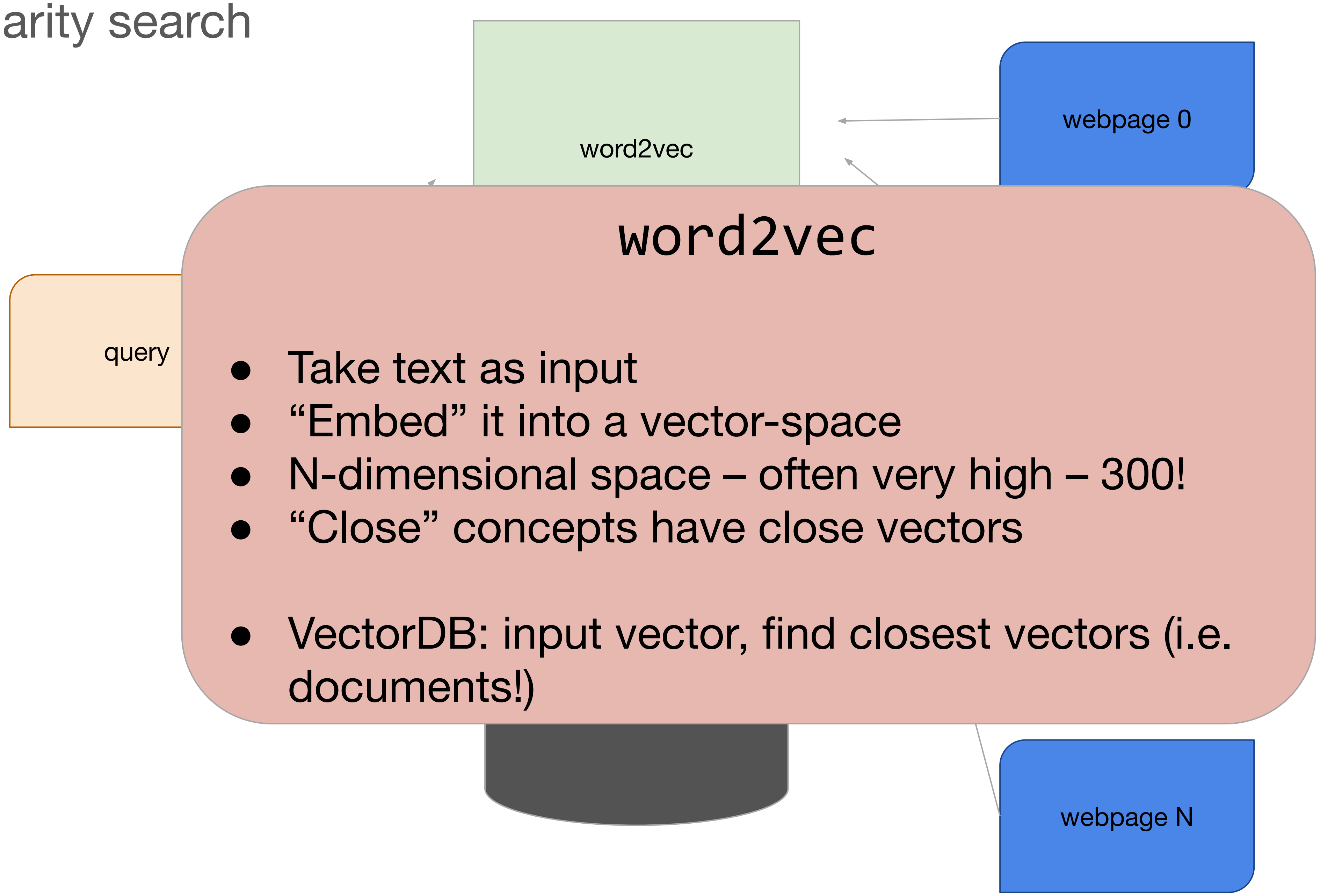
- Similarity search



Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages "closest" in content to the query!

- Similarity search



word2vec

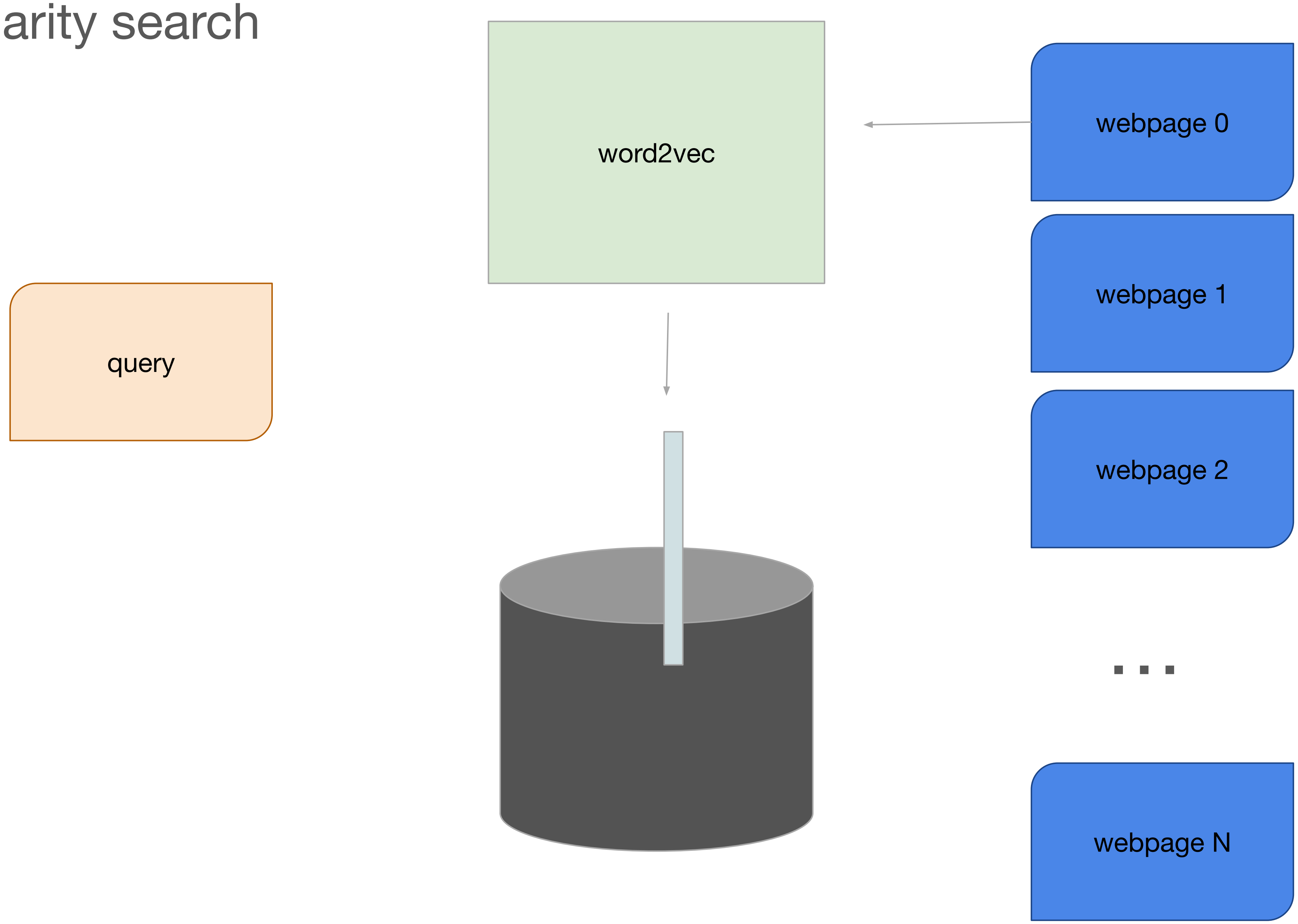
- Take text as input
- "Embed" it into a vector-space
- N-dimensional space – often very high – 300!
- "Close" concepts have close vectors

- VectorDB: input vector, find closest vectors (i.e. documents!)

Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages “closest” in content to the query!

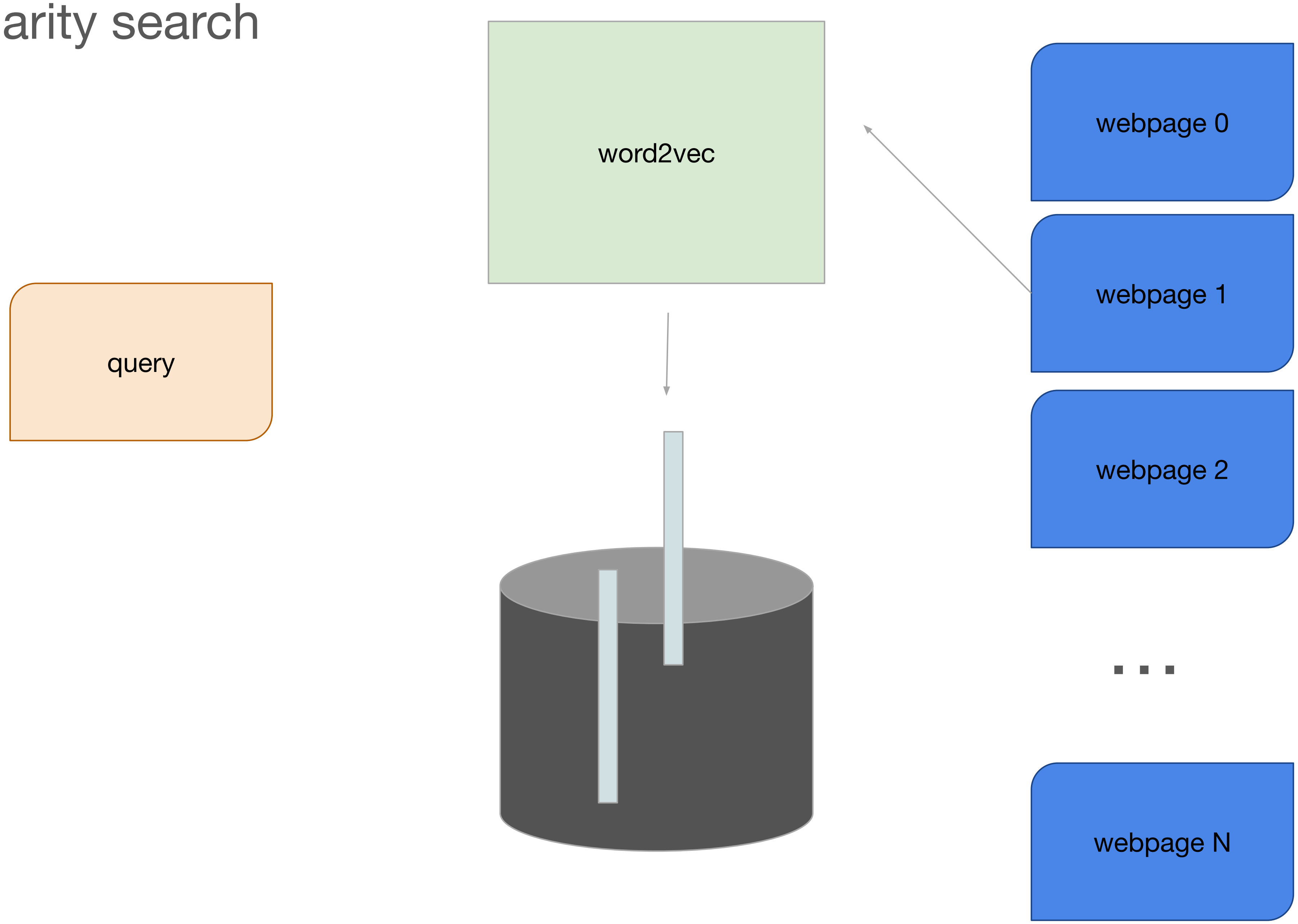
- Similarity search



Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages “closest” in content to the query!

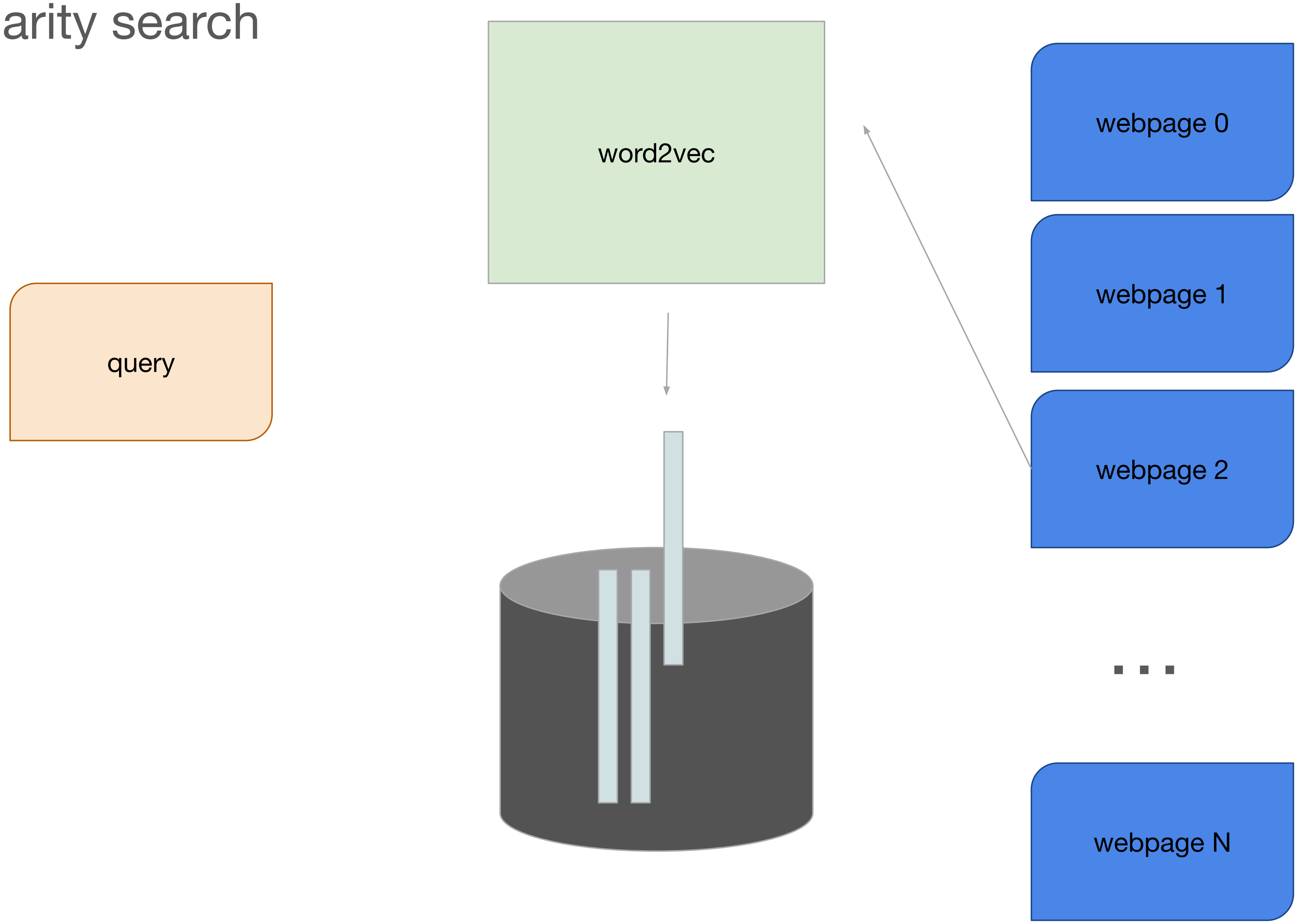
- Similarity search



Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages “closest” in content to the query!

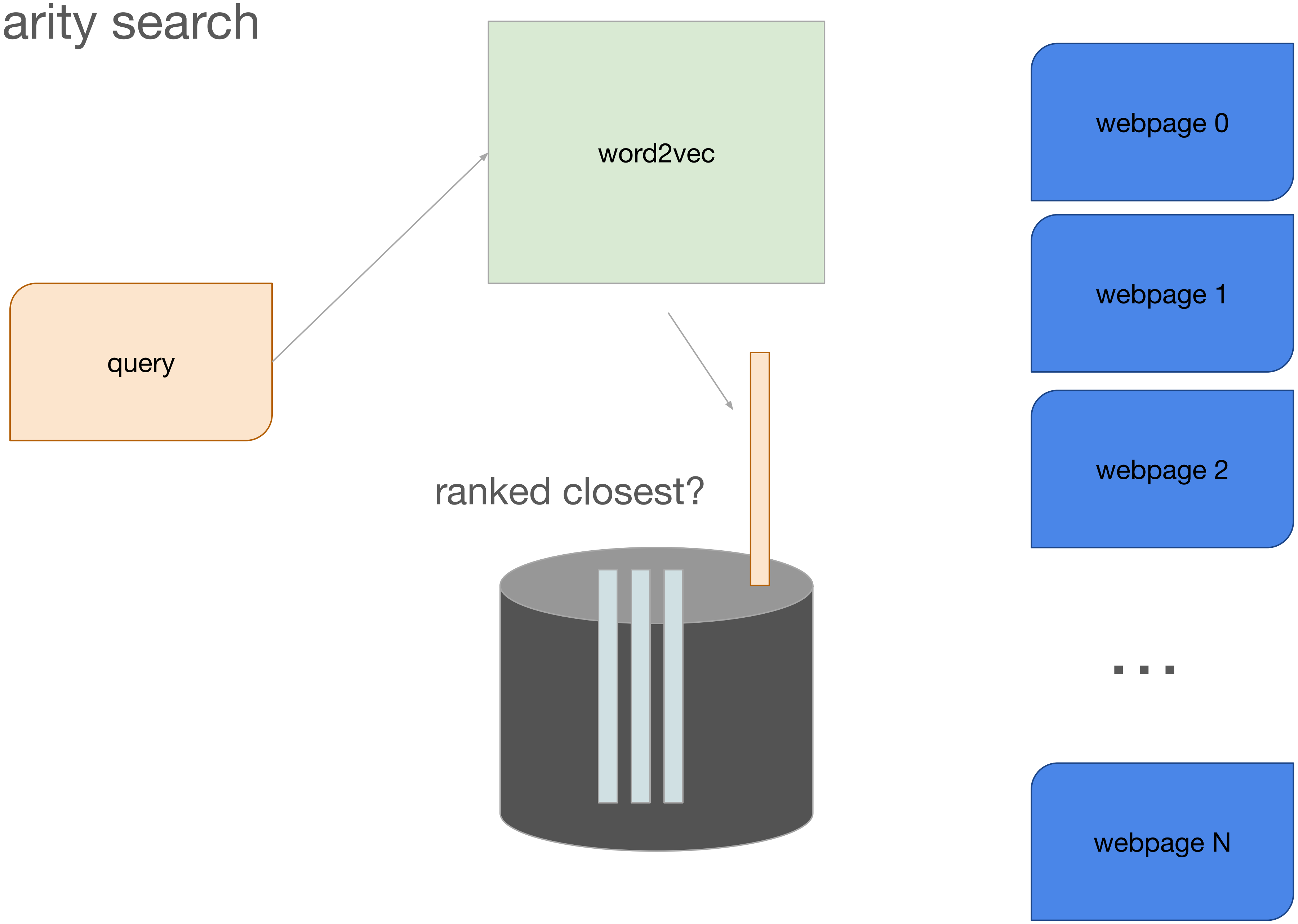
- Similarity search



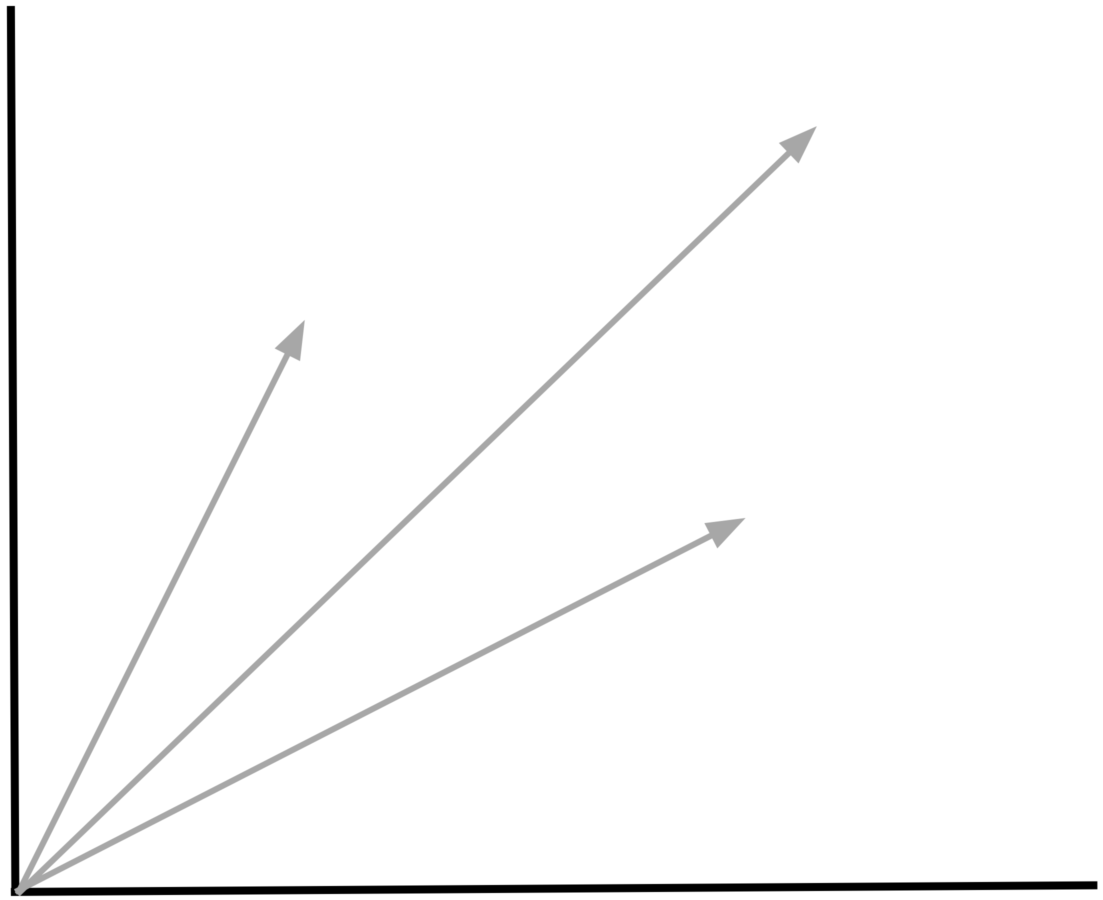
Vector DB

Let's imagine a program that tracks all webpages, takes a user query in natural language, and gives a list of webpages "closest" in content to the query!

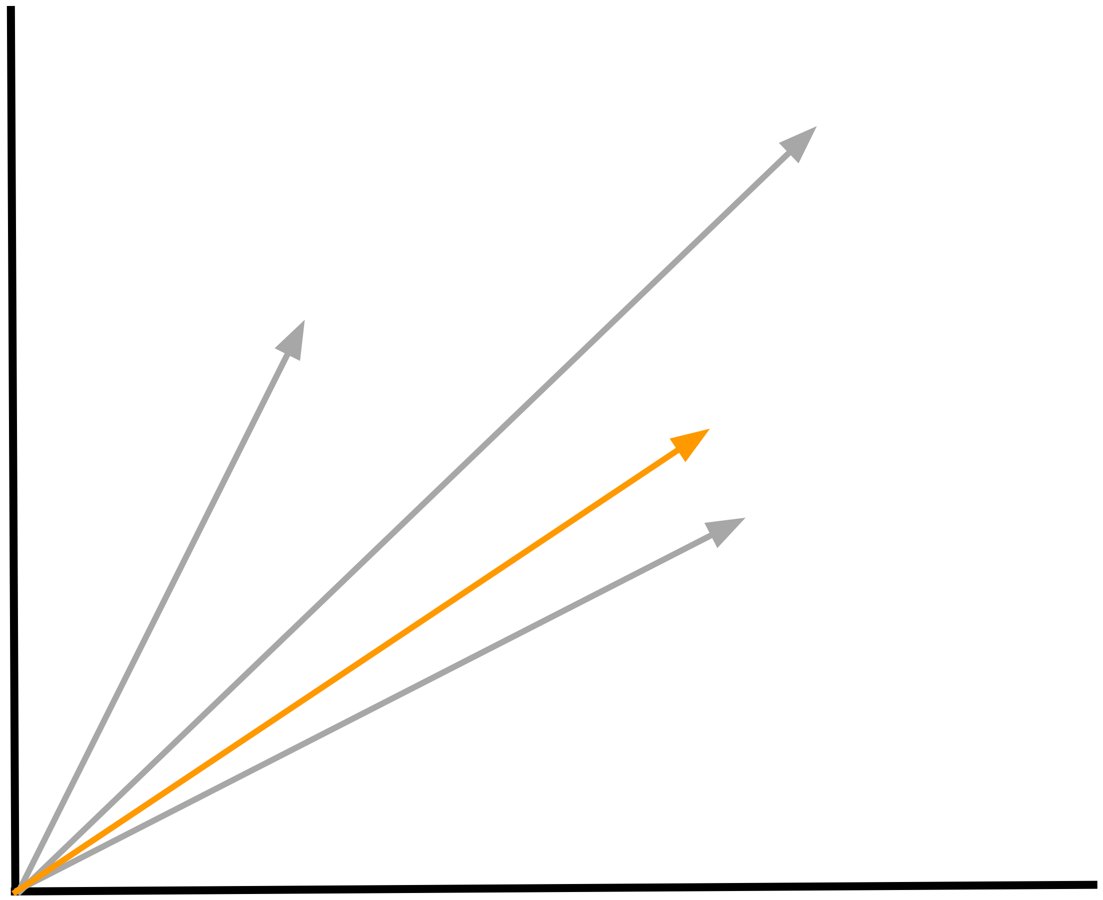
- Similarity search



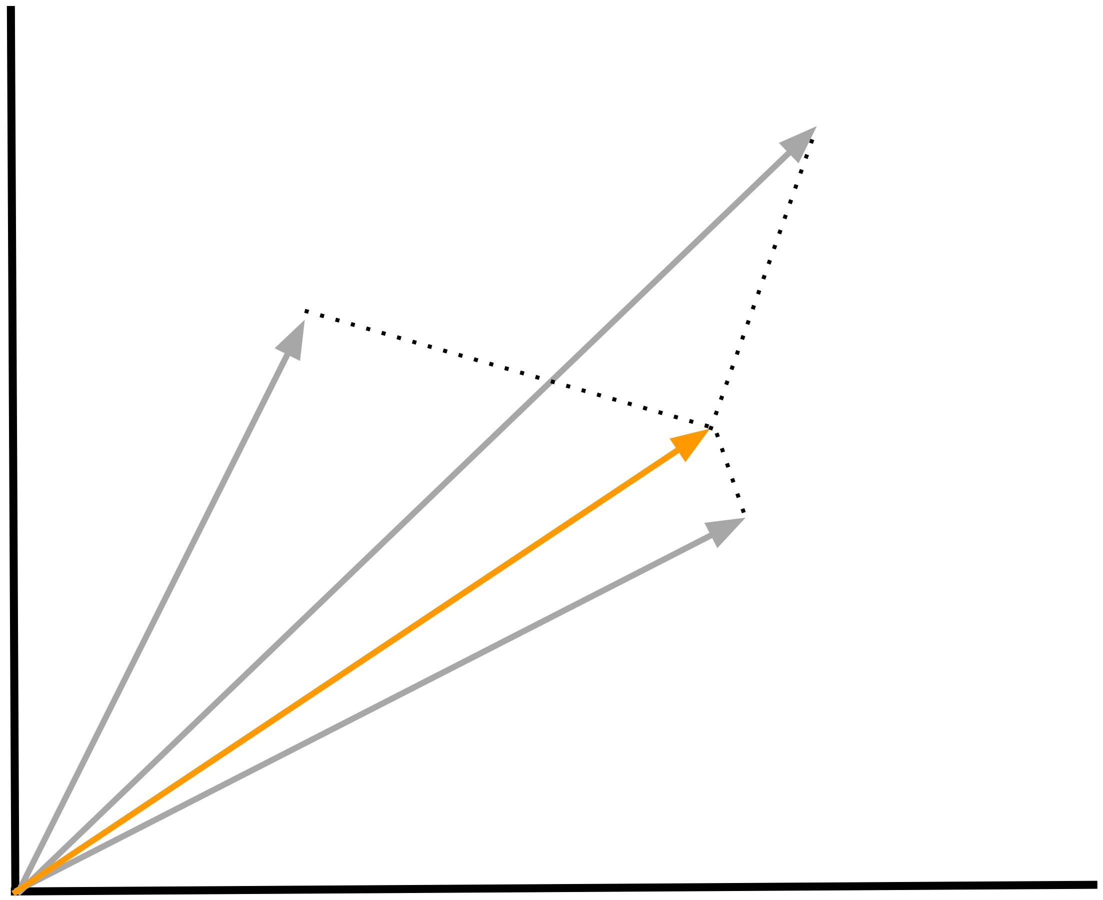
Vector DB



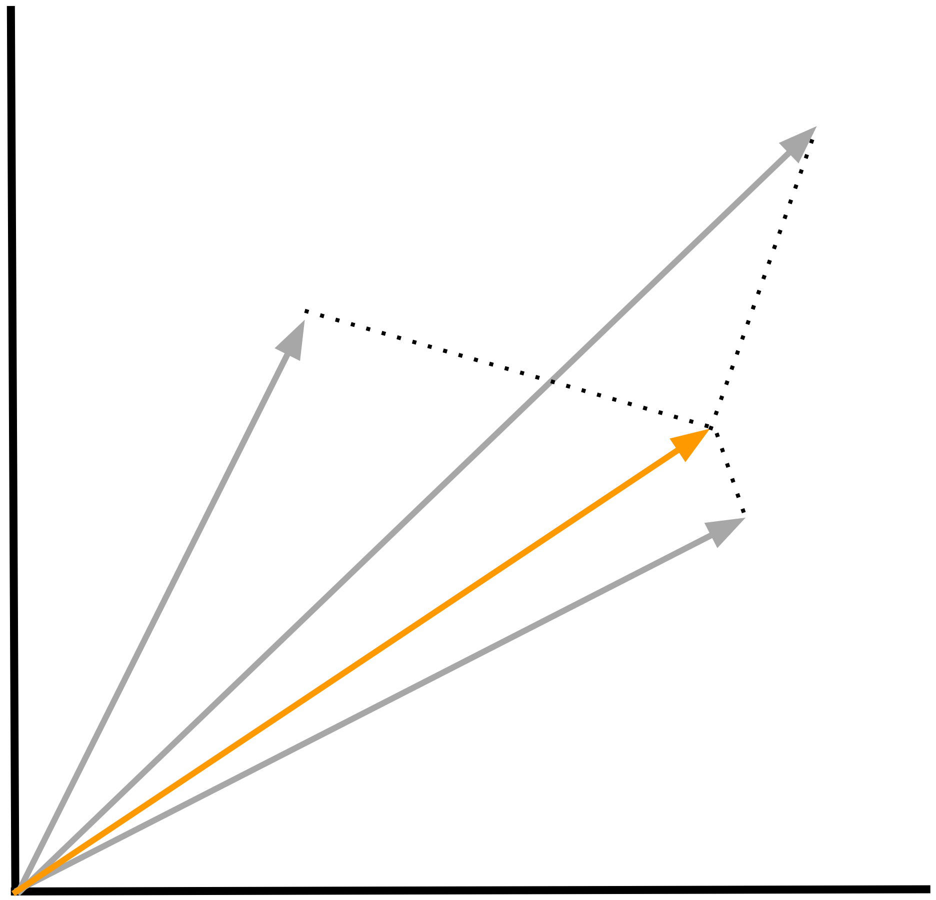
Vector DB



Vector DB

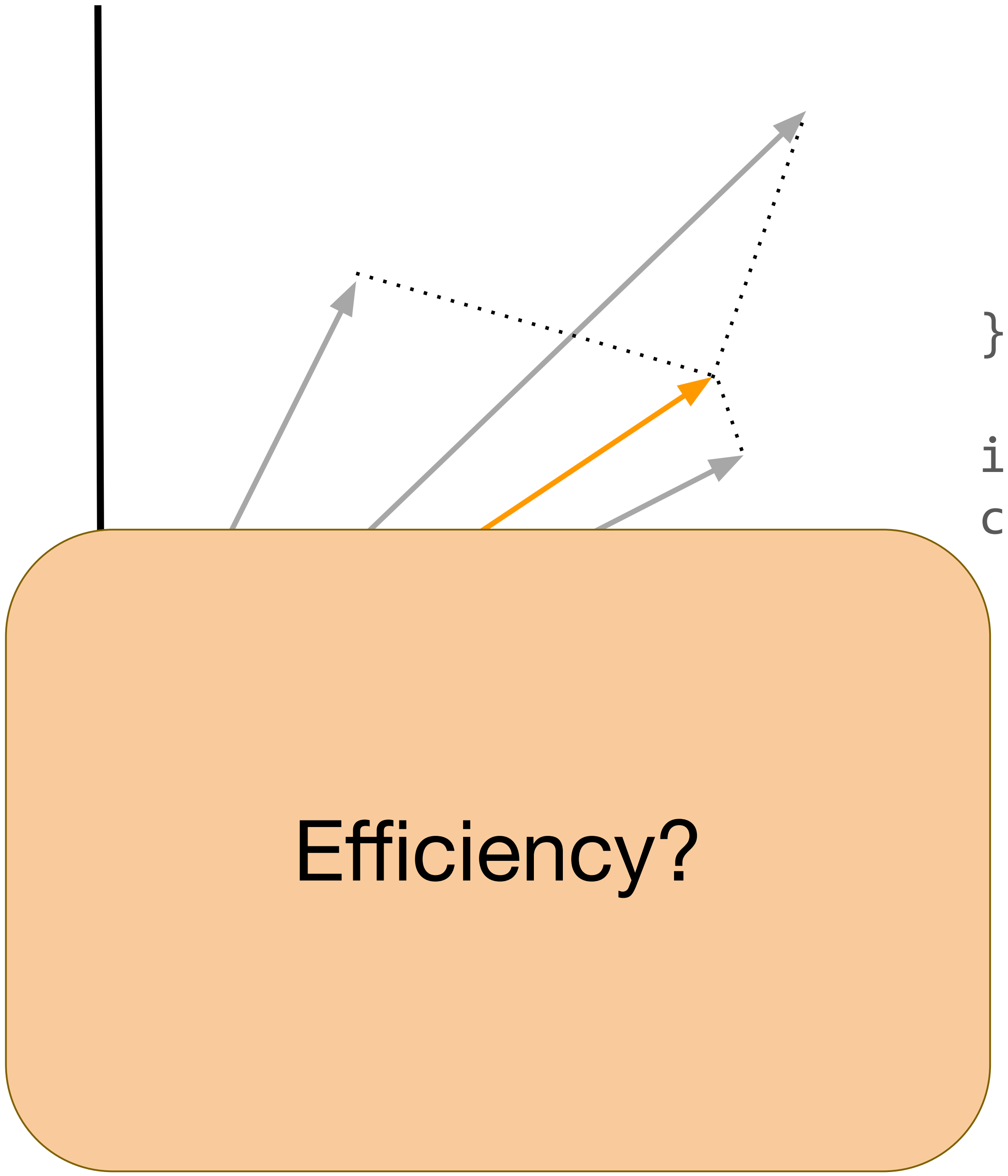


Vector DB



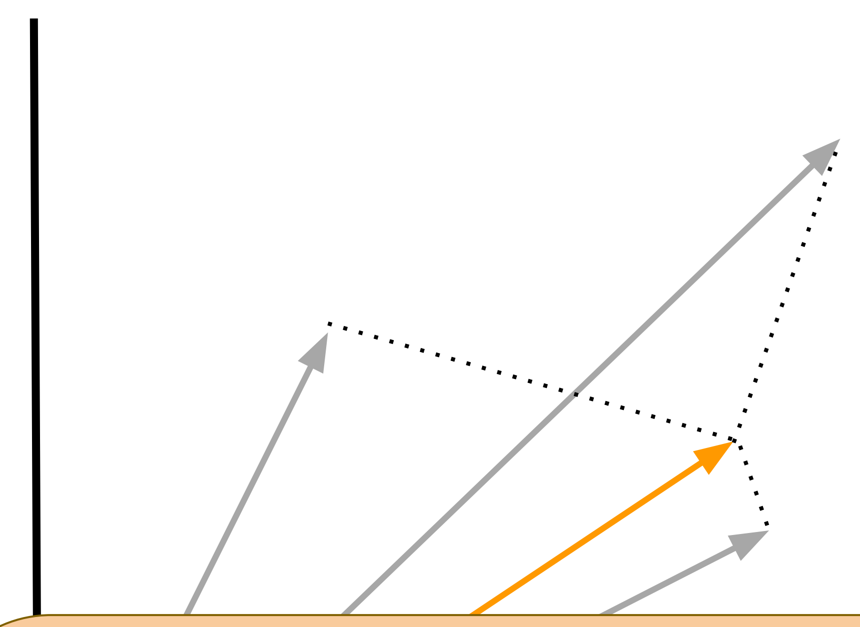
```
// euclidean distance
float
distance(float *v0, float *v1) {
    float sum = 0;
    for (int i = 0; i < DIM; i++) {
        sum += pow(v0[i] - v1[i], 2);
    }
    return sqrt(sum);
}

int
closest(float **vs, float *q) {
    float min = FLT_MAX;
    int min_off = 0;
    for (int i = 0; vs[i]; i++) {
        float d = distance(vs[i], q);
        if (min > d) {
            min = d;
            min_off = i;
        }
    }
    return min_off;
}
```

Vector DB




```
// euclidean distance
float
distance(float *v0, float *v1) {
    float sum = 0;
    for (int i = 0; i < DIM; i++) {
        sum += pow(v0[i] - v1[i], 2);
    }
    return sqrt(sum);
}

int
closest(float **vs, float *q) {
    float min = FLT_MAX;
    int min_off = 0;
    for (int i = 0; vs[i]; i++) {
        float d = distance(vs[i], q);
        if (min > d) {
            min = d;
            min_off = i;
        }
    }
    return min_off;
}
```

Vector DB



Vector DBs:

- Efficient similar search
- Hierarchical – compare distance to a small set of vectors, then to a set close to the closest, then …

```
// euclidean distance
float
distance(float *v0, float *v1) {
    float sum = 0;
    for (int i = 0; i < DIM; i++) {
        sum += pow(v0[i] - v1[i], 2);
    }
    return sqrt(sum);
}

int
closest(float **vs, float *q) {
    float min = FLT_MAX;
    int min_off = 0;
    for (int i = 0; vs[i]; i++) {
        float d = distance(vs[i], q);
        if (min > d) {
            min = d;
            min_off = i;
        }
    }
    return min_off;
}
```

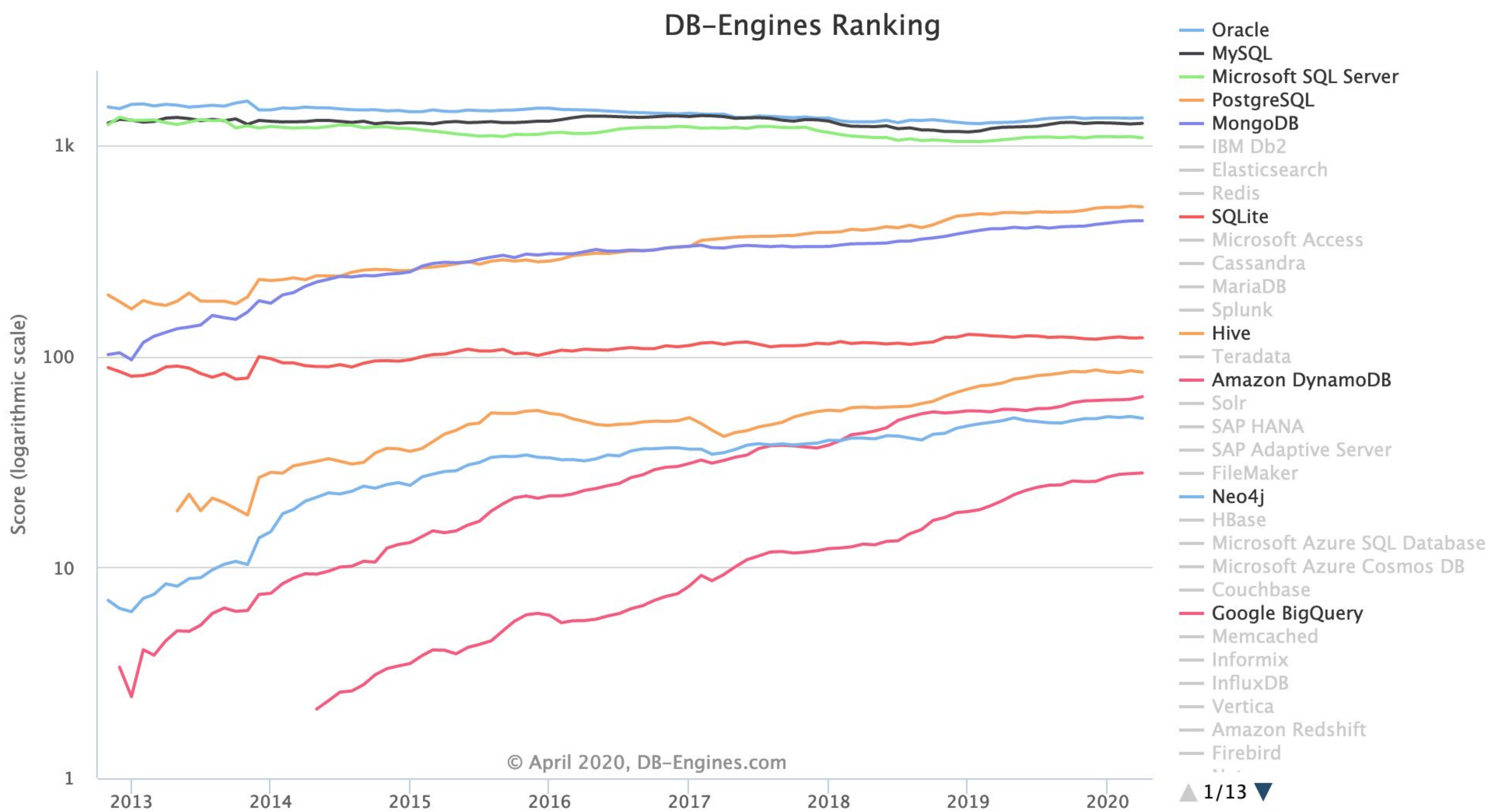
Which DB to use?

- Transactional, relational DB?
- No-SQL?
- Graph?
- Analytics DB?
- Vector DB?

Good news: Postgres supports all of the options!

- You need to know what you want,
- and why you want it.

DB Engines Ranking: DBMS systems by popularity



https://db-engines.com/en/ranking_trend

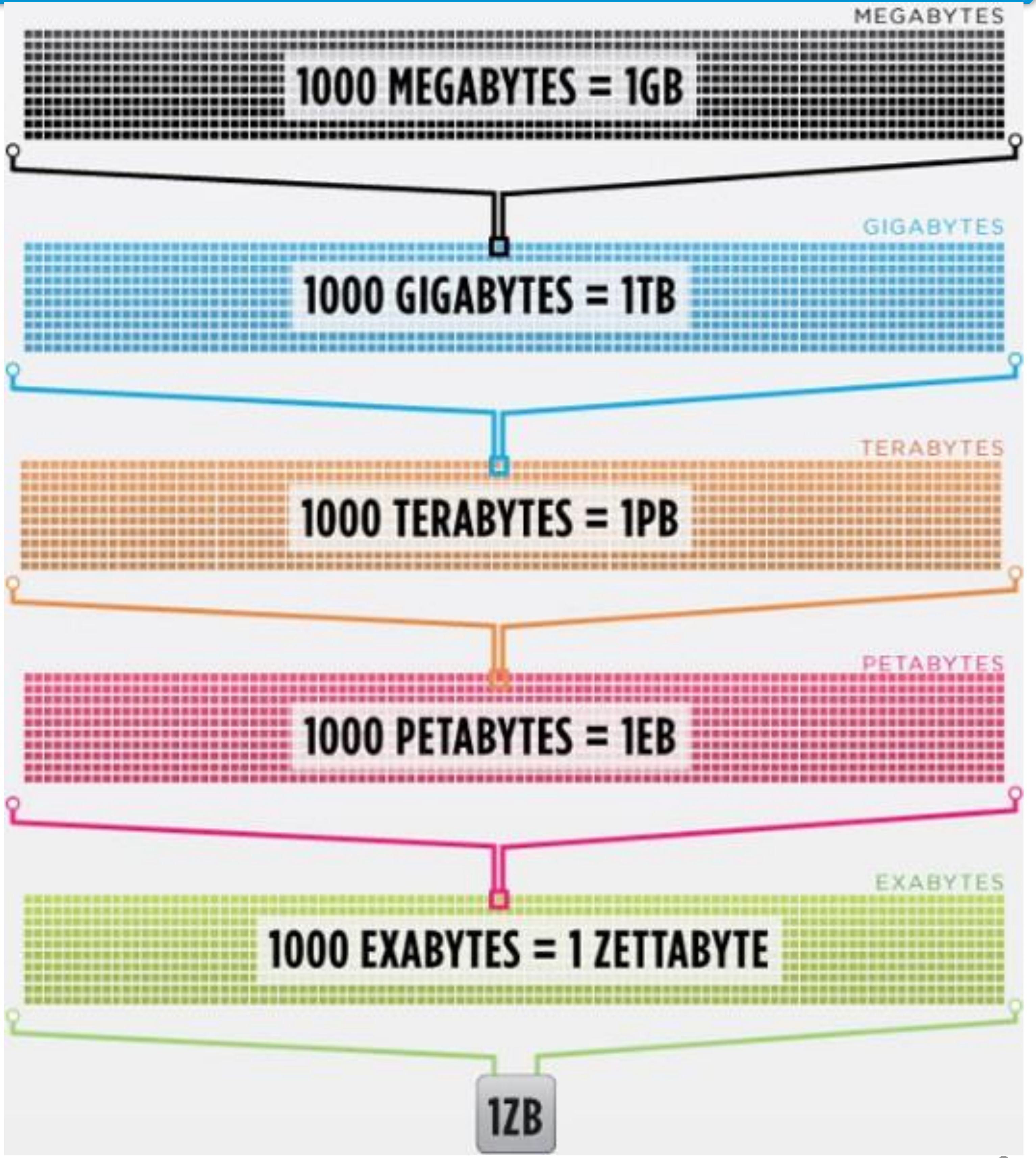
stopped here in Monday lecture

Trend 1

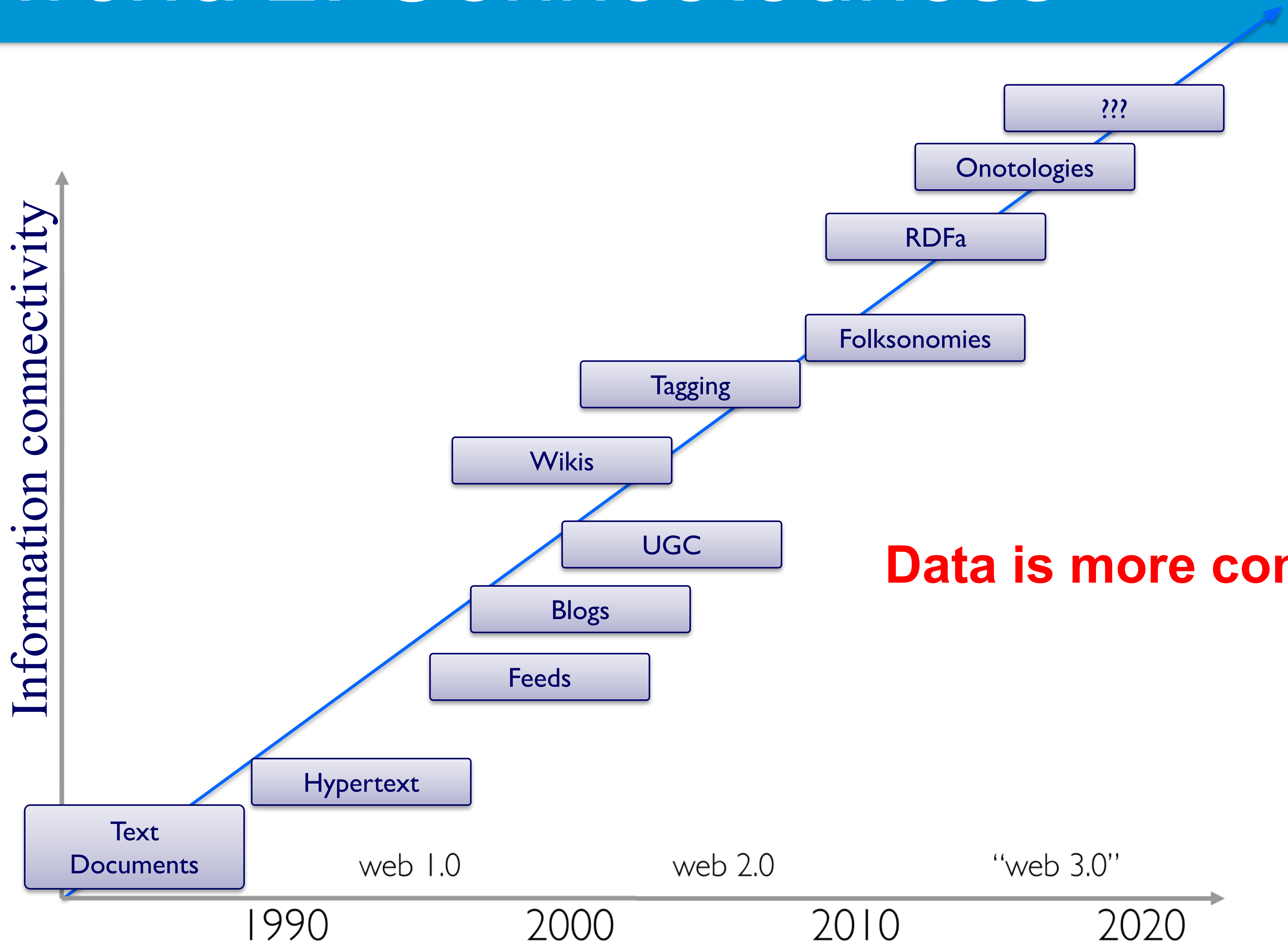
Data is getting bigger:

"Every 2 days we create as much information as we did up to 2003"
– Eric Schmidt, Google in **2010**

Facebook generates
4 Petabytes per day! (2020)



Trend 2: Connectedness



Data is more connected

Trend 3: Data is often Semi-Structured (or no structure)

If you tried to collect all the data of every movie ever made, how would you model it?

Actors, Characters, Locations, Dates, Costs, Ratings, Showings, Ticket Sales, etc.

Relational Databases Challenges

Some features of relational databases make them "challenging" for certain problems:

- 1) Fixed schemas – defined ahead of time, changes are difficult, and lots of real-world data is “messy”. Relational design requires lots of Joins. **So get rid of schemas**
- 2) Complicated queries – SQL is declarative and powerful but may be overkill. **Instead, do the work in application code**
- 3) Transaction overhead – Not all data and query answers need to be perfect. **Close enough is sometimes good enough**
- 4) Scalability – Relational databases may not scale sufficiently to handle high data and query loads or this scalability comes with a very high cost. **Find new ways to scale**