NoSQL Databases

Data Modeling
Aggregates + Graphs

Rodrygo L. T. Santos

rodrygo@dcc.ufmg.br

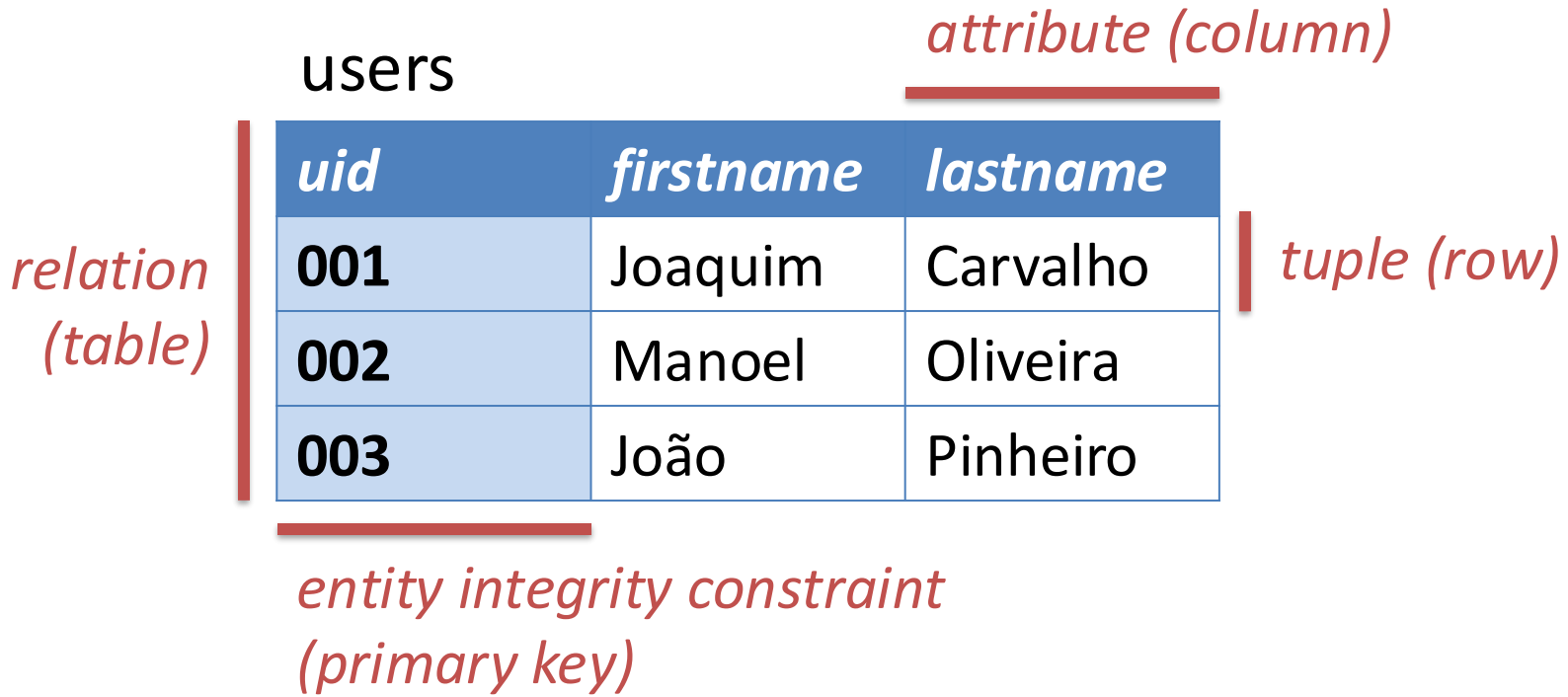
About data models

"A data model [is] a collection of concepts that can be used to describe the structure of a database."

[Elmasri & Navathe, 2011]

data types + relationships + constraints + basic operations

The good ol' relational model



The good ol' relational model

users

uid	*firstname*	*lastname*
001	Joaquim	Carvalho
002	Manoel	Oliveira
003	João	Pinheiro

telephones

uid	*number*	*type*
001	3409-1234	work
001	8888-1234	mobile
002	3409-4321	work

referential integrity constraint (foreign key)

The good ol’ relational model

```
SELECT
  firstname, lastname, number, type
FROM
  users, telephones
WHERE
  users.uid = telephones.uid AND
  lastname = ‘Oliveira’
```

Data modeling goals

- Data-oriented modeling
  - Driven by the structure of the data

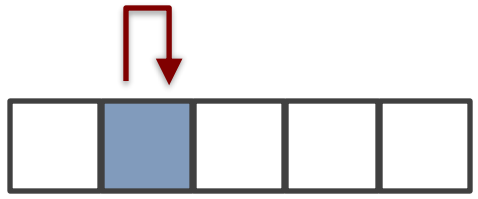
"What answers do I have?"

- Query-oriented modeling
  - Driven by access patterns

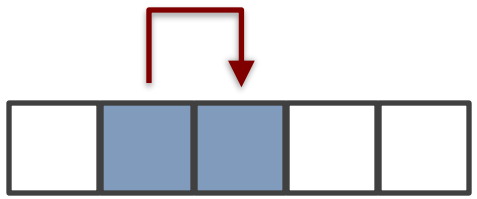
"What questions do I have?"

How to take advantage?

- **Principle of locality:** *"programs tend to use data and instructions with addresses* ***near or equal*** *to those they have used recently"*

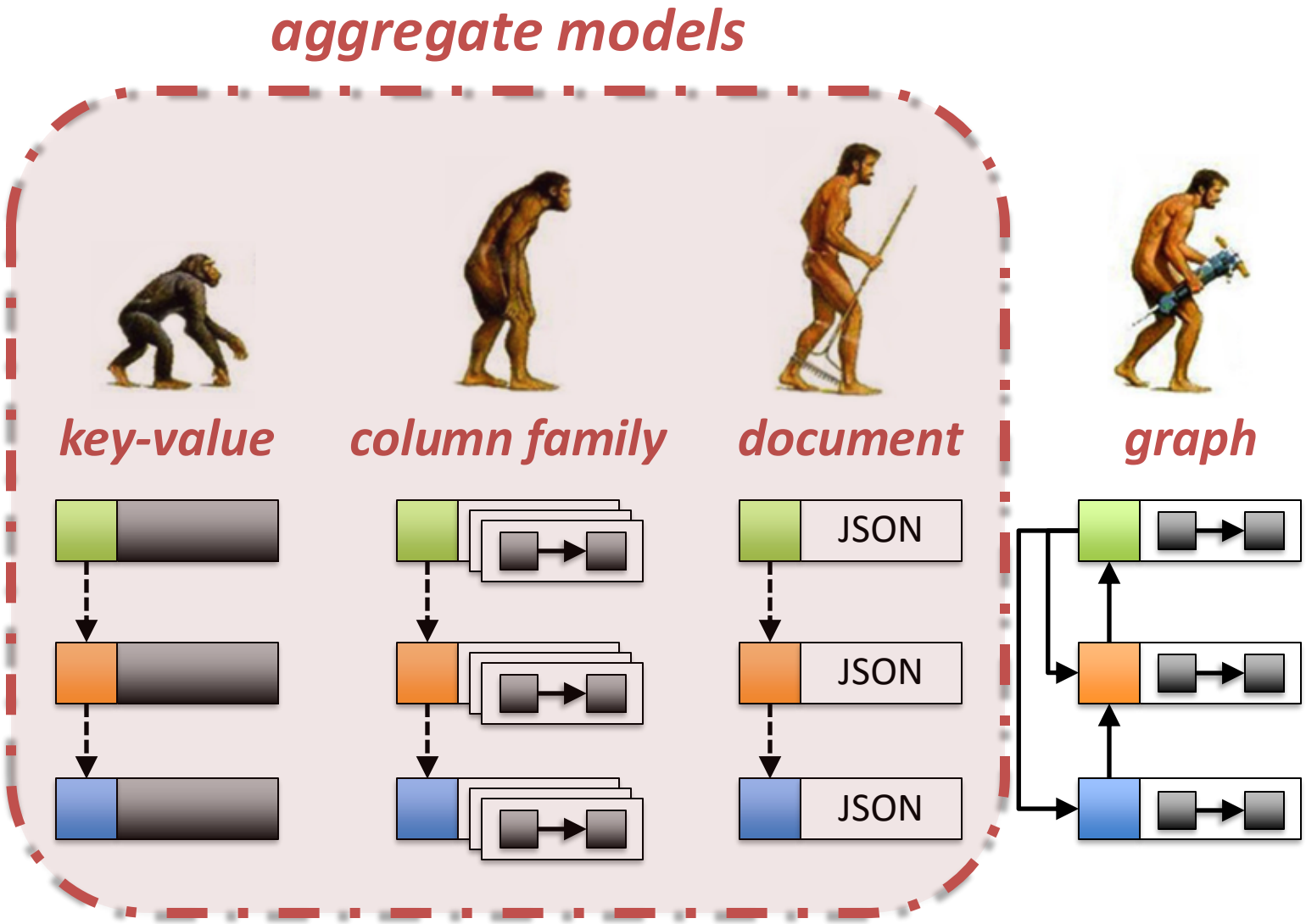


temporal locality



spatial locality

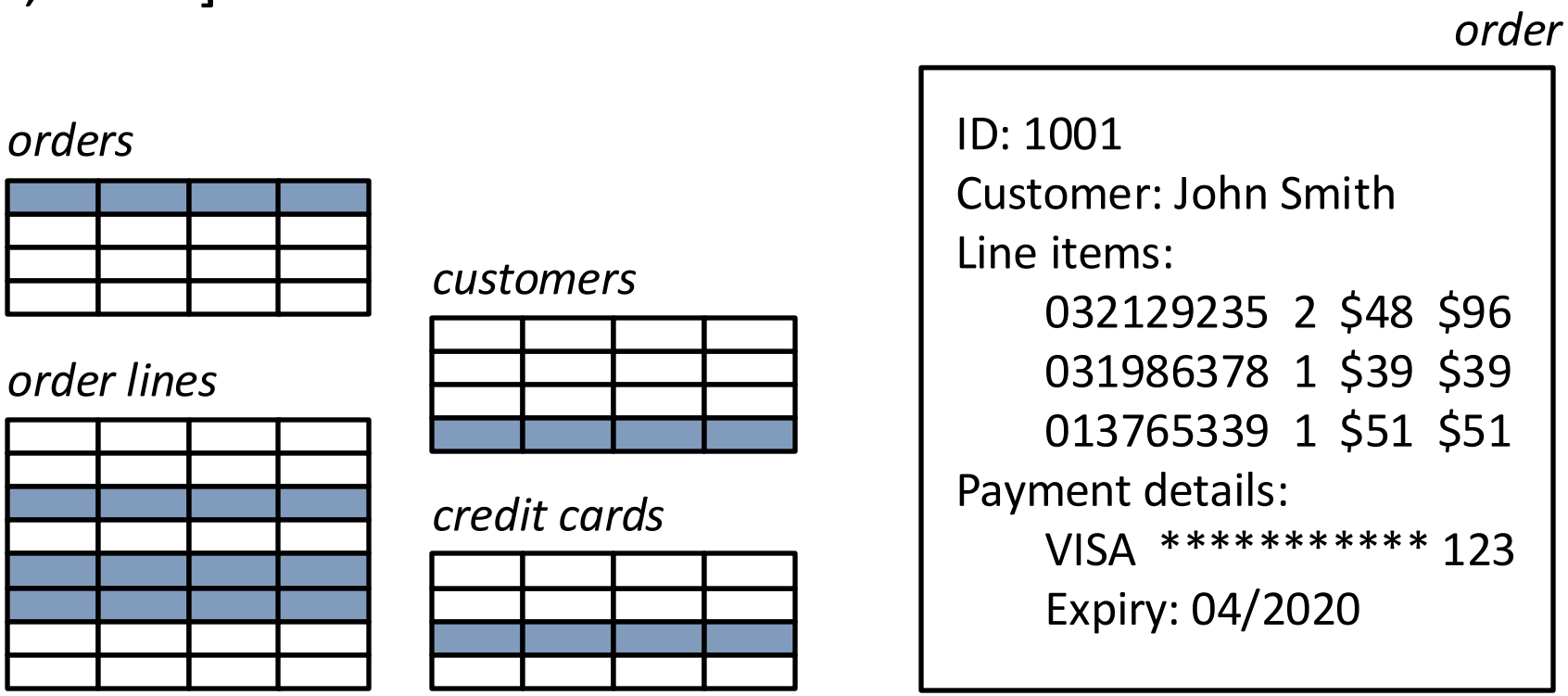
NoSQL medicine



Aggregates

"An aggregate is a collection of related objects that we wish to treat as a unit"

[Fowler, 2013]



Why aggregates?

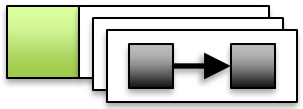
- Aggregates are not a logical data property
  - Remember query-driven modeling?
- Help greatly with running on a cluster
  - Minimize the number of nodes queried for data
- Help transaction processing
  - An aggregate is a natural transaction boundary
  - *Cross-aggregate atomicity needs application code*

So, what flavors are there?

Opaque aggregates

- Aggregate is a blob
  - Store whatever you like

key-value



column family

- Can only query by key
  - Retrieve the whole thing

Transparent aggregates

- Aggregate has structure
  - Must respect the underlying data structure (JSON, XML)

document

- Can query by key or value
  - Can retrieve parts or whole

Key-value API

- Operations
  - get(key)
  - set(key, value, [ttl])
  - del(key)
- Data types
  - blob
  - list, set, hash of blob

Key-value API

- Example (Riak)

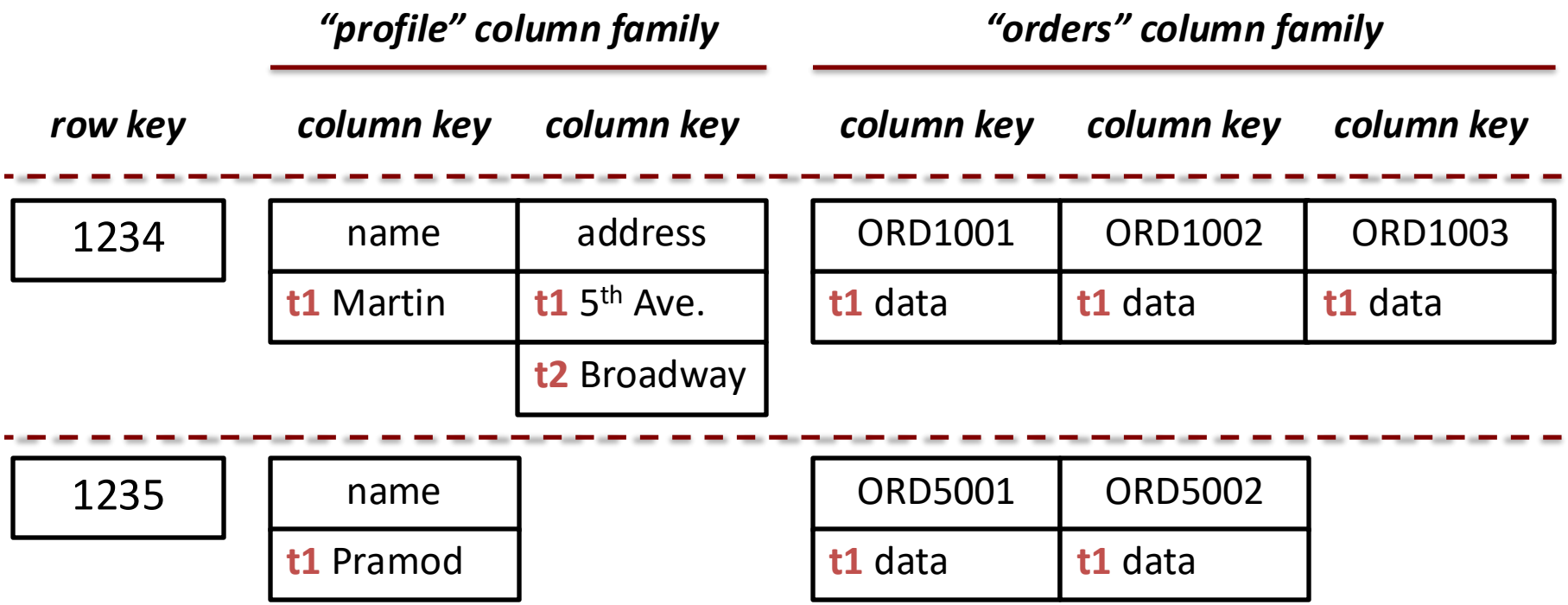
```
curl -v -X POST -d '
 { "lastVisit":1324669989288,
   "user":{"customerId":"91cfdf5bcb7c",
   "name":"buyer",
   "countryCode":"US",
   "tzOffset":0}
 }' \
 -H "Content-Type: application/json" \
 http://localhost:8098/buckets/session/keys/a7e618d9db25

curl -i http://localhost:8098/buckets/session/keys/a7e618d9db25
```

Key-value DOs

- Session data
  - Session ID as the key
- Shopping cart data
  - Session ID as the key
- User profile data
  - User ID as the key
- … and any other scenario that requires concurrently looking up things by key

Column family model



row is the aggregate!

Column family API

- Operations
  - get(cfamily, row, [column])
  - set(cfamily, row, column, value)
  - del(cfamily, row, [column])

    or
  - CQL: a simple SQL flavor (Cassandra)
    - No joins, no subqueries, simple selection conditions
- Data types
  - blob

Column family API

- Example (Cassandra CQL)

```
CREATE COLUMNFAMILY Customer (
  KEY varchar PRIMARY KEY,
  name varchar,
  city varchar,
  web  varchar);

INSERT INTO Customer (KEY, name, city, web)
  VALUES ('mfowler', 'Martin Fowler', 'Boston',
        'www.martinfowler.com');

SELECT name, web FROM Customer WHERE city='Boston'
```

Column family DOs

- Event logging
  - Scalable writes
- Content management systems
  - Tags, categories, links, trackbacks as columns
- Counters
  - INCR Visits['mfowler'][home] BY 1;
- Expiring data
  - SET Customer['mfowler']['demo_access'] = 'allowed' WITH ttl=2592000;

Document API

- Operations
  - get(*XPath-like expression*)
  - put(document)
  - del(docid)
- Data types
  - Structured document (JSON, BSON, XML)

Document API

- Example (MongoDB)

```
{
 "userId":"883c2c5”,
 "firstname": "Pramod",
 "citiesvisited": [ "Chicago", "London” ],
 "addresses": [
  { "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
  },
  { "state": "MH",
    "city": "PUNE",
    "type": "R" }
 ],
 "lastcity": ”London"
}
```

```
db.users.insert(document)
db.users.find({"userId":"883c2c5"})
db.users.find({"addresses.state":/AK/})
```

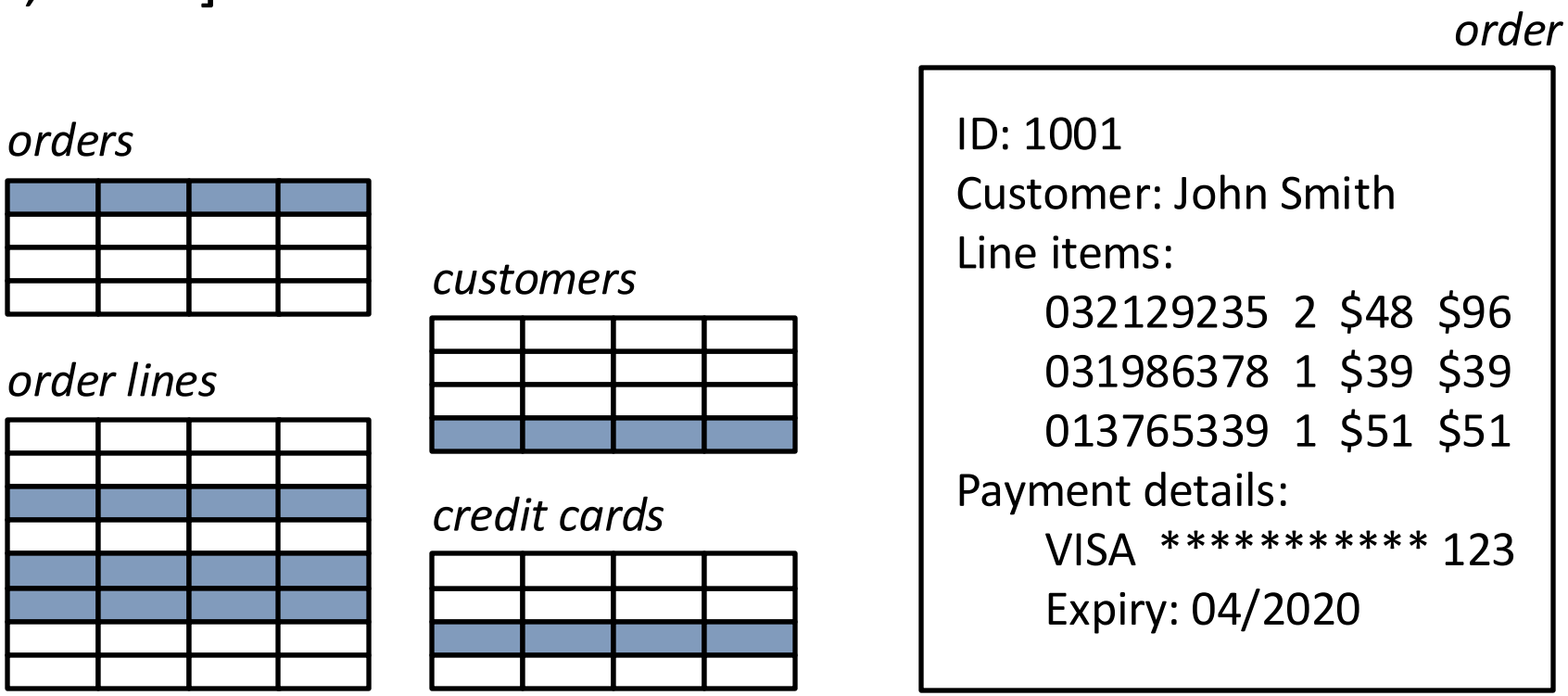
Document DOs

- Content management systems
  - Values are documents
- Search engines
  - Elasticsearch is a prominent example
- Data analytics
  - New bits can be added efficiently
- E-commerce applications
  - Flexible schema and cheap refactoring

Aggregates

"An aggregate is a collection of related objects that we wish to treat as a unit"

[Fowler, 2013]



Aggregates

- Process customer together with an order
  - Order is the aggregate
- ***Process each customer individually?***
  - Customer and order are the aggregates
  - Provide links between customer and order
  - Links can even be indexed
    - Link in transparent document fields
    - Link in metadata for opaque values

WAIT!

Customer and order in separate structures…
Implicit links between the two…

This is not NoSQL…
… this is OldSQL!

OldSQL to the rescue?

- Relationships in the relational world
  - Customer(customer_id, customer_name)

    Order(order_id, ..., customer_id)

    Order(customer_id) **REFERENCES** Customer(customer_id)
- Pull out your join hammer
  - SELECT *

    FROM Customer **NATURAL JOIN** Order

Writability

- Complex relationships = long SQL expressions
  - Customers who bought same product?

    SELECT C1.customer_id, C2.customer_id

    FROM Customer C1 **NATURAL JOIN** Order O1

    NATURAL JOIN Product P1 **NATURAL JOIN** Order O2

    NATURAL JOIN Customer C2

    WHERE C1.customer_id <> C2.customer_id
  - Customer and FOAF who bought same product?
    - Good luck…

Efficiency

- Complex relationships = many index lookups
  - Path exists between two vertices?
    (each person with an average of 50 friends)

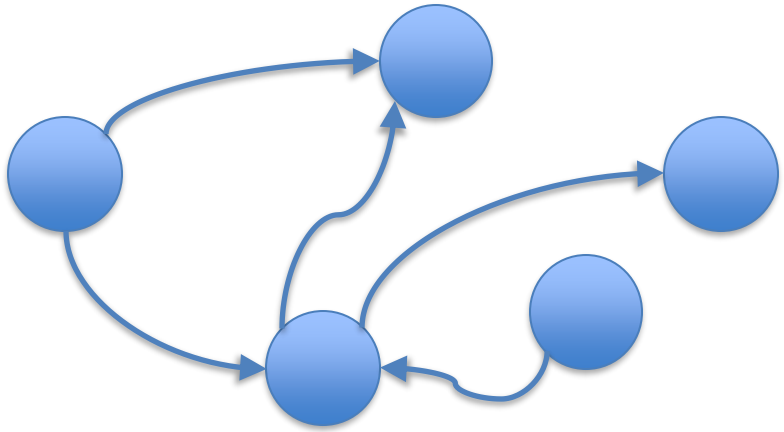
# persons	OldSQL
1,000	2,000 ms
1,000,000	?

Enter graph databases

- OldSQL isn’t cluster-friendly
  - Bring in aggregate databases
- OldSQL struggles with complex relationships
  - Bring in graph databases

Remember graphs?

- The most flexible data structure



vertices

+

edges

- But I could model a graph in OldSQL…

  … and in aggregate NoSQL databases as well

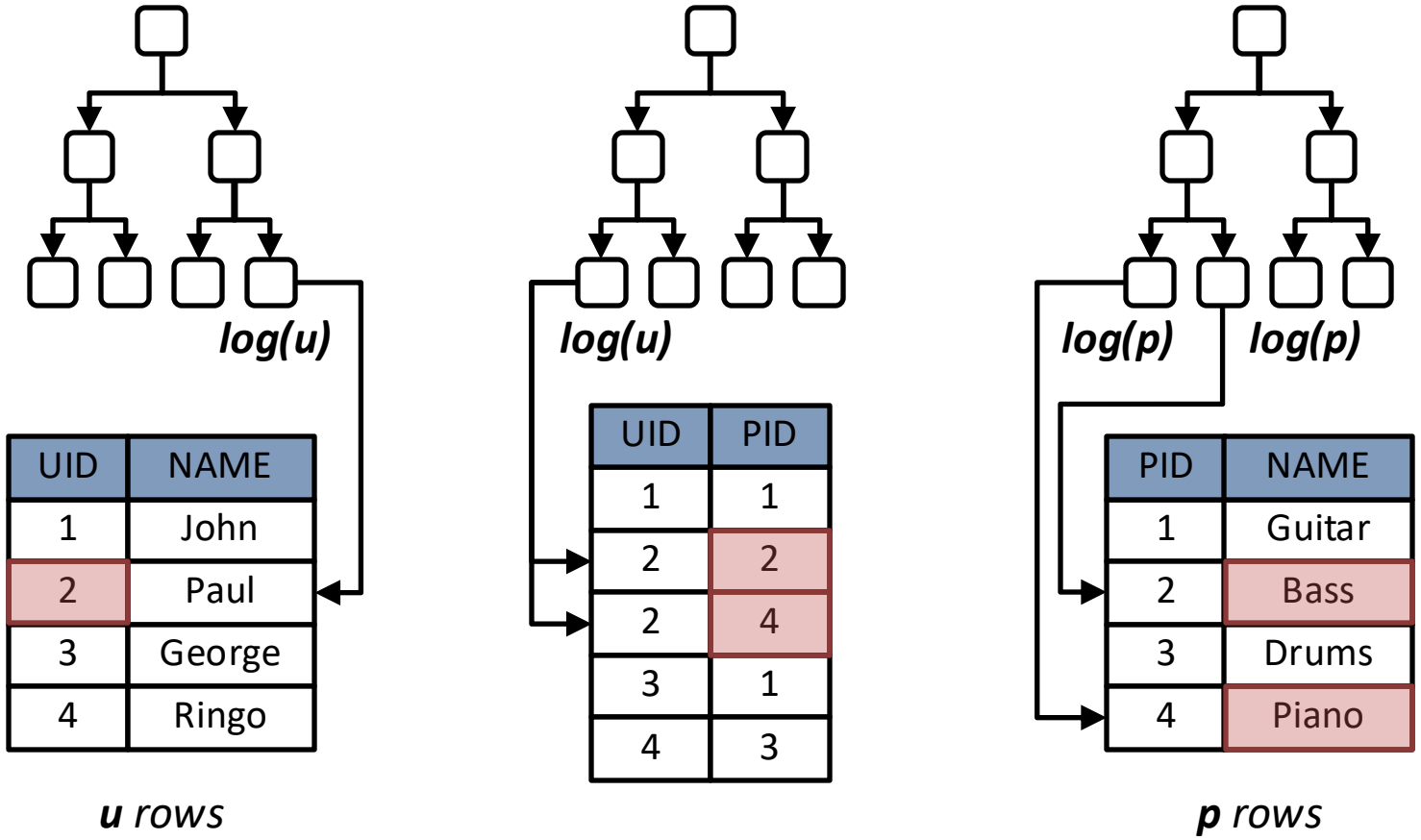
What is a graph database?

"A graph database [management system] is any storage system data provides ***index-free*** *adjacencies."*

[Rodriguez and Neubauer, CoRR 2010]

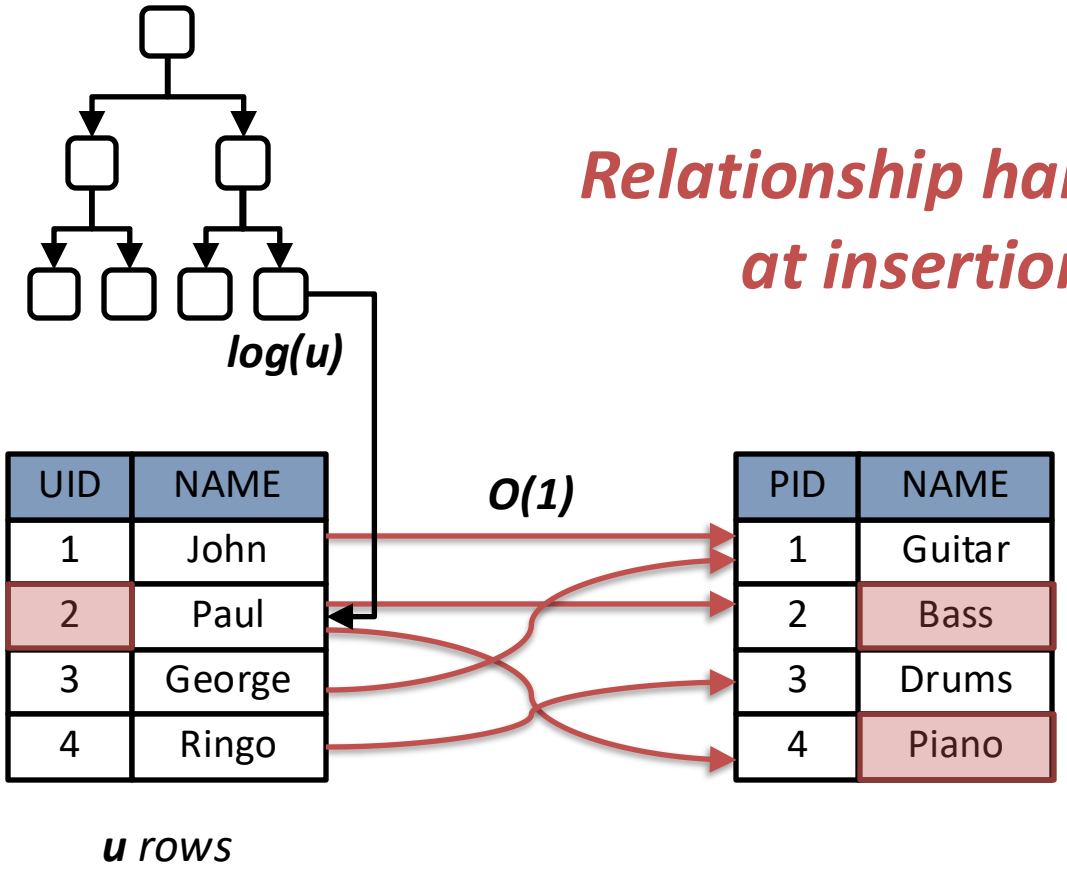
Relational traversals

- What instruments does Paul have?



Graph traversals

- What instruments does Paul have?

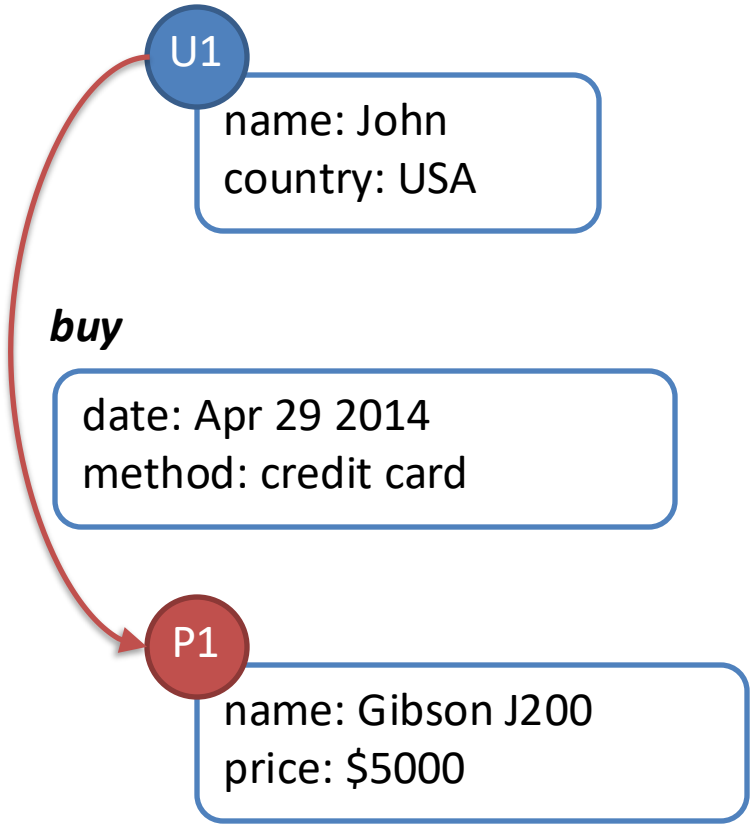


Why graph databases?

- Complex relationships = many index lookups
  - Path exists between two vertices?
    (each person with an average of 50 friends)

# persons	OldSQL	Graph DB
1,000	2,000 ms	2 ms
1,000,000	Way too long…	2 ms

The property graph model



- Directed, edge-labeled, attributed graph
  - Aka the property graph
- G = (V, E, λ, μ)
  - V: set of vertices
  - E: $E \subseteq (V \times V)$
  - λ: $E \rightarrow \Sigma$
  - Σ: set of labels
  - μ: $(V \cup E) \times R \rightarrow S$
  - R: set of keys
  - S: set of values

Graph API

- Create, Update, Delete operations
  - add/del(vertex)
  - add/del(edge, vertex-in, vertex-out)
  - set/del(vertex/edge, key, value)
- Retrieve operation
  - Look up start vertex (O(log n))
  - Traverse edges (O(1))

Graph API

- Cypher (declarative)
  - Bob's friends and their location

```
START b = node:nodeIndex(name = "bob")
MATCH (b)-[:FRIEND]->(f)
RETURN f.name, f.location
```

  - Bob's friends' friends' friends

```
START b = node:nodeIndex(name = "bob")
MATCH path = (b)-[:FRIEND*1..3]->(f)
RETURN b.name, f.name, length(path)
```

Graph API

- Gremlin (imperative)
  - Collaborative filtering for vertex 1

```
m = [:];
g.v(1).out('likes').in('likes')
  .out('likes').groupCount(m);
m.sort{-it.value};
```

  - Vendor-agnostic standard
  - Extremely powerful
    - Transform, filter, side effect, branch predicates

Graph DO's

- Connected data
  - Multi-relational graphs
- Location-based services
  - Place is a vertex
  - Edges have a distance property
- Recommendation engines
  - User and item as vertices
  - Click, rate, like, buy as edges