Pocket Data

The Case for TPC-MOBILE

Oliver Kennedy, Jerry Ajay, Geoff Challen, Lukasz Ziarek

http://odin.cse.buffalo.edu/research/astral

Big Data!

Big Data!

Big Data!

- GB, TB or PB of data!
- Hundreds of thousands of updates per second
- Thousands of nodes computing together!
- “Virtually” infinite resources!

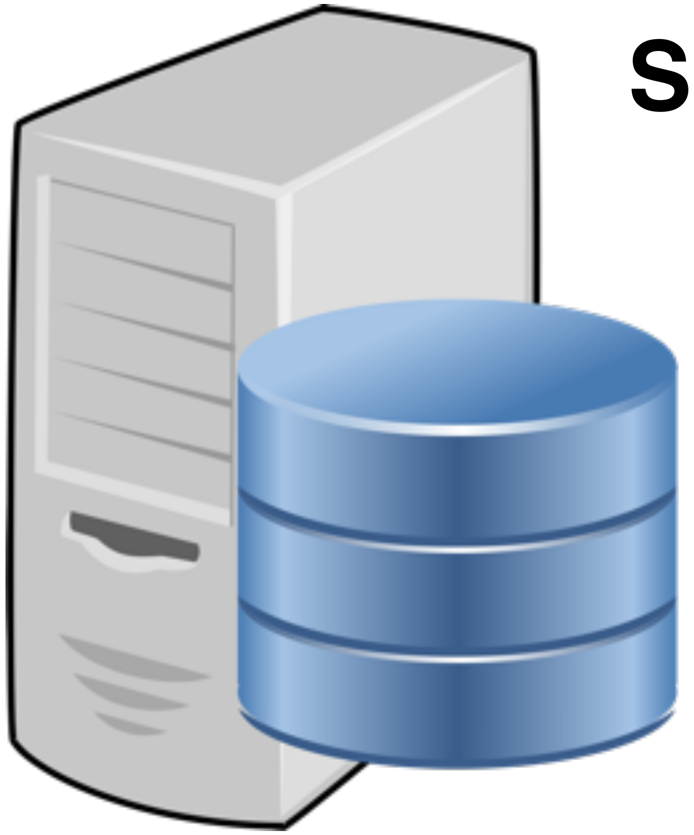
Big Data!

Big Data!

TPC-H

TPC-C

SSB



TPC-E

TPC-DI

YCSB

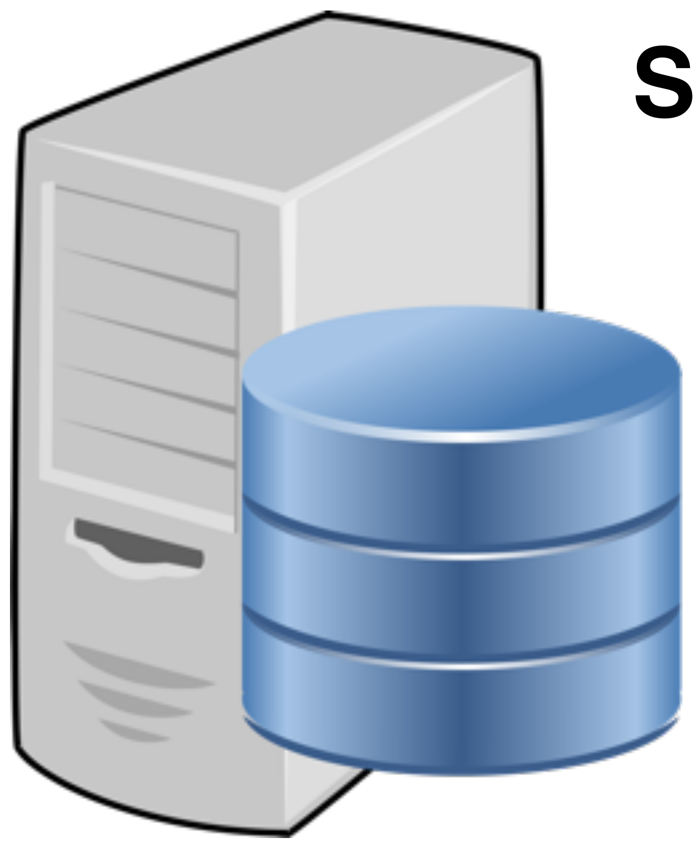
TPC-DS

Big Data!

TPC-H

TPC-C

SSB



TPC-E

TPC-DI

YCSB

TPC-DS

What about other types of databases?

The average smartphone processes almost 180 thousand queries per day

That's about **2 queries per second**

2 Queries per Second

2 Queries per Second

- Is this Big Data? **No!**

2 Queries per Second

- Is this Big Data? **No!**
- Is this Important?

2 Queries per Second

- Is this Big Data? **No!**
- Is this Important?
    - **Multi-Tenancy**: The phone is more than just a DB.

2 Queries per Second

- Is this Big Data? **No!**
- Is this Important?
  - **Multi-Tenancy**: The phone is more than just a DB.
  - **Power**: 1-2 days of battery life under ideal circumstances.

2 Queries per Second

- Is this Big Data? **No!**
- Is this Important?
  - **Multi-Tenancy**: The phone is more than just a DB.
  - **Power**: 1-2 days of battery life under ideal circumstances.
  - **It's Everywhere**: Odds are that your phone is running some queries right now!

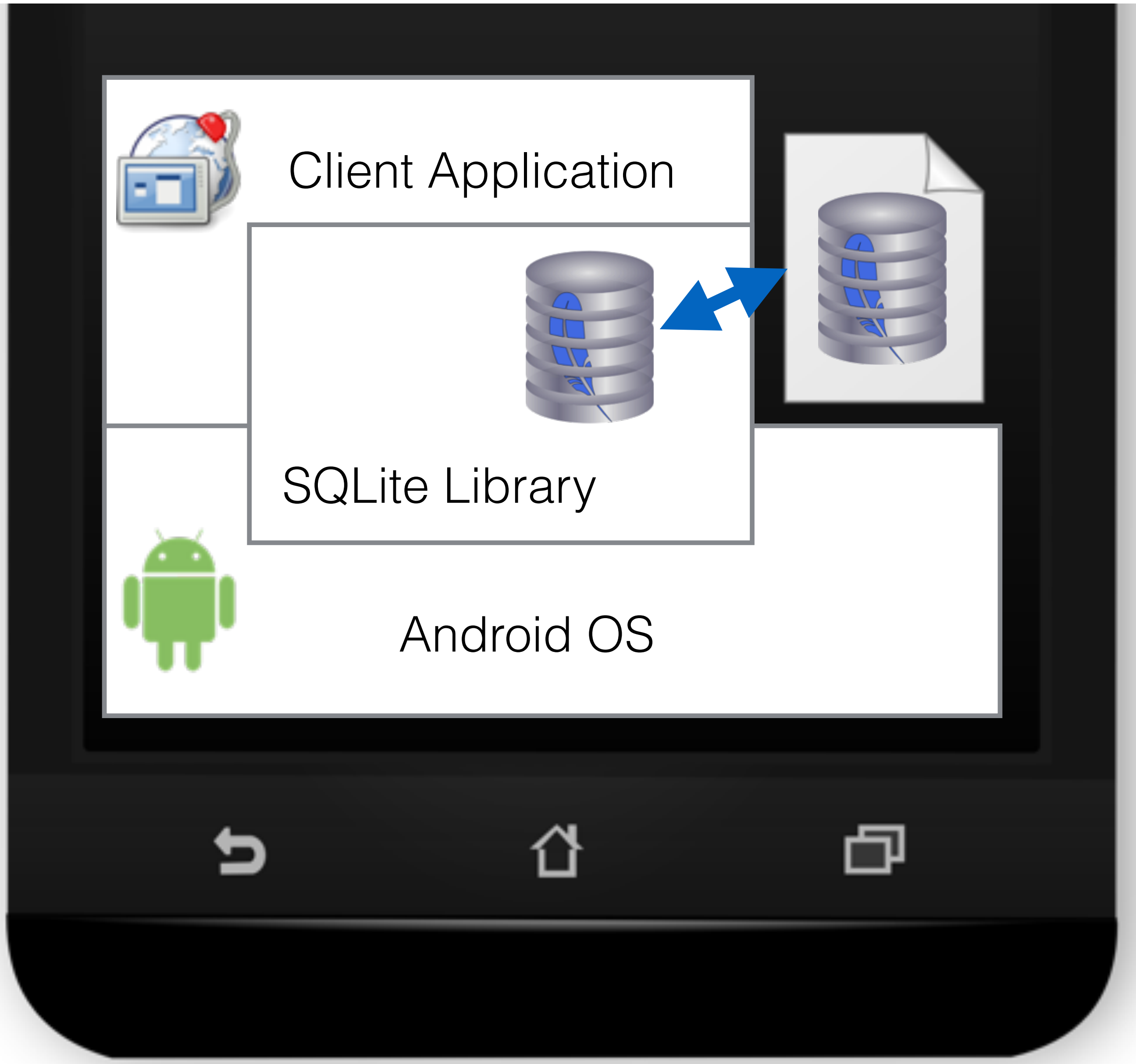
2 Queries per Second

- Is this Big Data? **No!**
- Is this Important? **YES!**
    - **Multi-Tenancy**: The phone is more than just a DB.
    - **Power**: 1-2 days of battery life under ideal circumstances.
    - **It's Everywhere**: Odds are that your phone is running some queries right now!

We need to better understand pocket-scale data

SQLite

- **Embedded**: SQLite is a library
- **Un-shared**: SQLite DBs are specific to one client “app”.
- **Lightweight**: Entire SQLite DB is backed to one file.
- **Universal**: SQLite client library is available by default in nearly all major OSes.
- **“Easy”**: Duck Typing, Relaxed SQL Syntax, One Big Lock (file)



How do developers and users use Pocket Scale Data?

PhoneLab

A Smartphone Platform Testbed

~200 UB students, faculty, and staff using instrumented LG Nexus 5 smartphones in exchange for discounted service.

PhoneLab

A Smartphone Platform Testbed

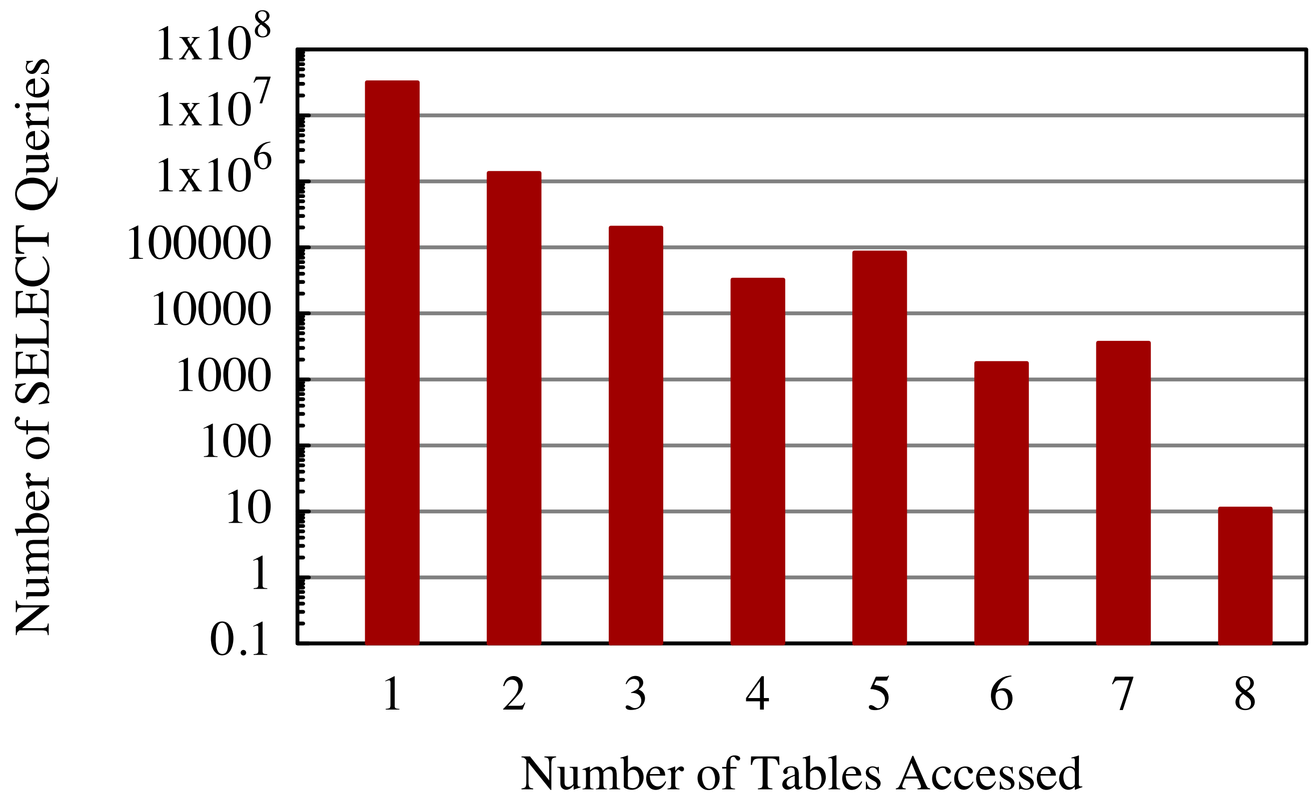
- **Preliminary Trial**: 11 phones for ~1 month (254 phone/days)
- Instrumented SQLite logs **all** statements (~45 mil statements)
  - ~33.5 million `SELECT` statements
  - ~9.4 million `INSERT` statements
  - ~1 million `UPDATE` statements
  - ~1.2 million `DELETE` statements
  - 179 distinct ‘apps’ issuing statements

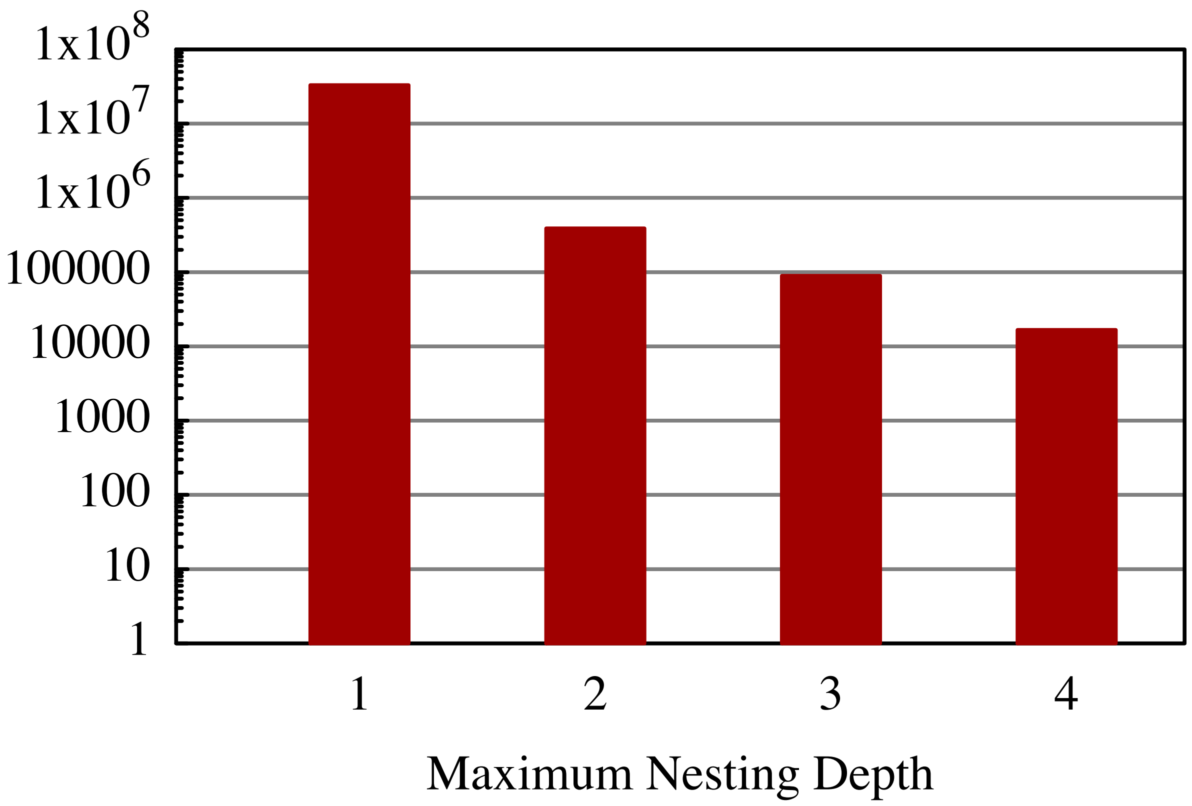
https://phone-lab.org/experiment/request

- SELECT Complexity
- ORM Effects
- Function Usage
- Read/Write Ratios
- Query Periodicity

- SELECT Complexity
- ORM Effects
- Function Usage
- Read/Write Ratios
- Query Periodicity

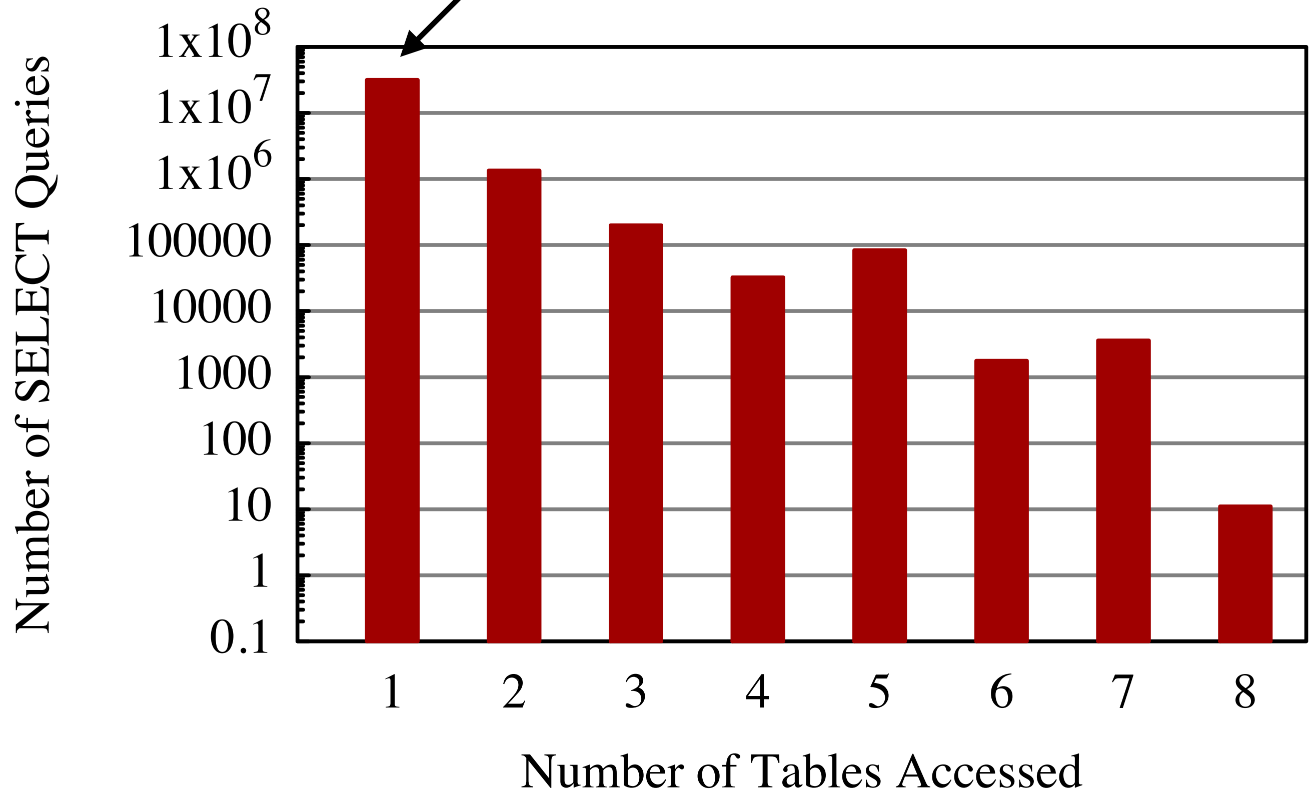
SELECT Complexity

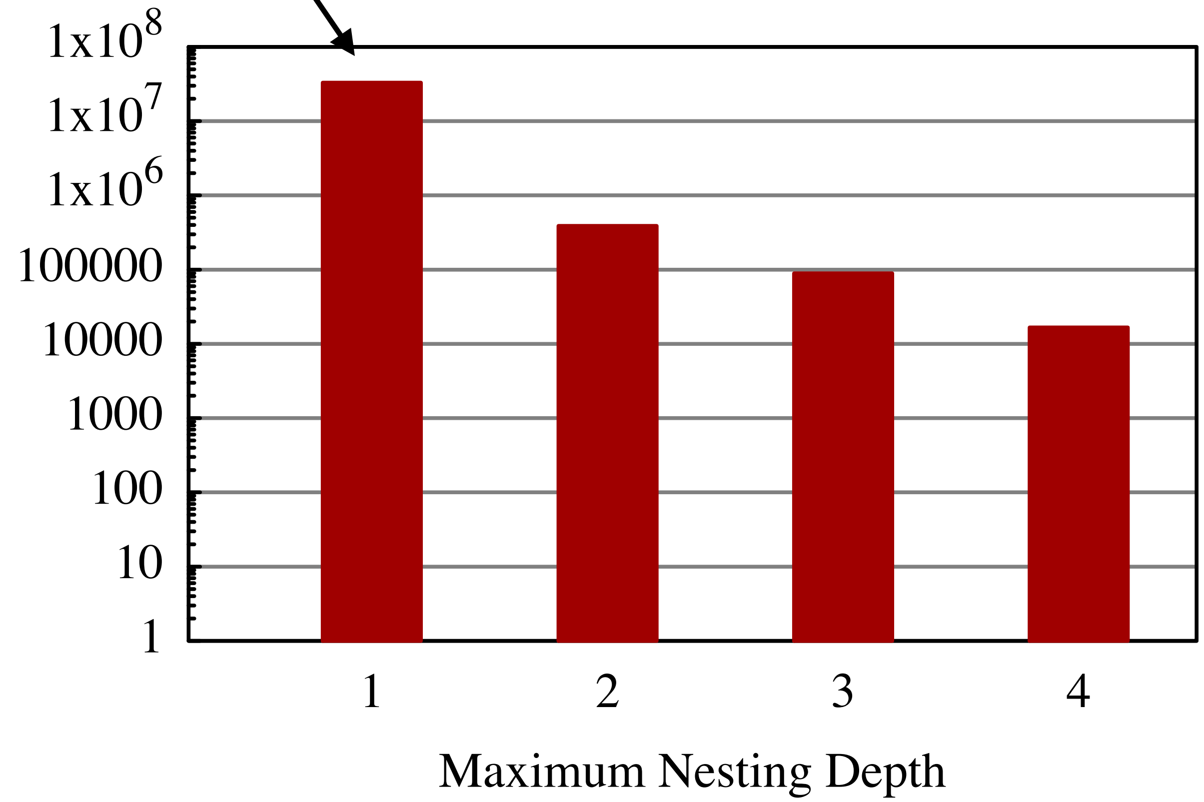




SELECT Complexity

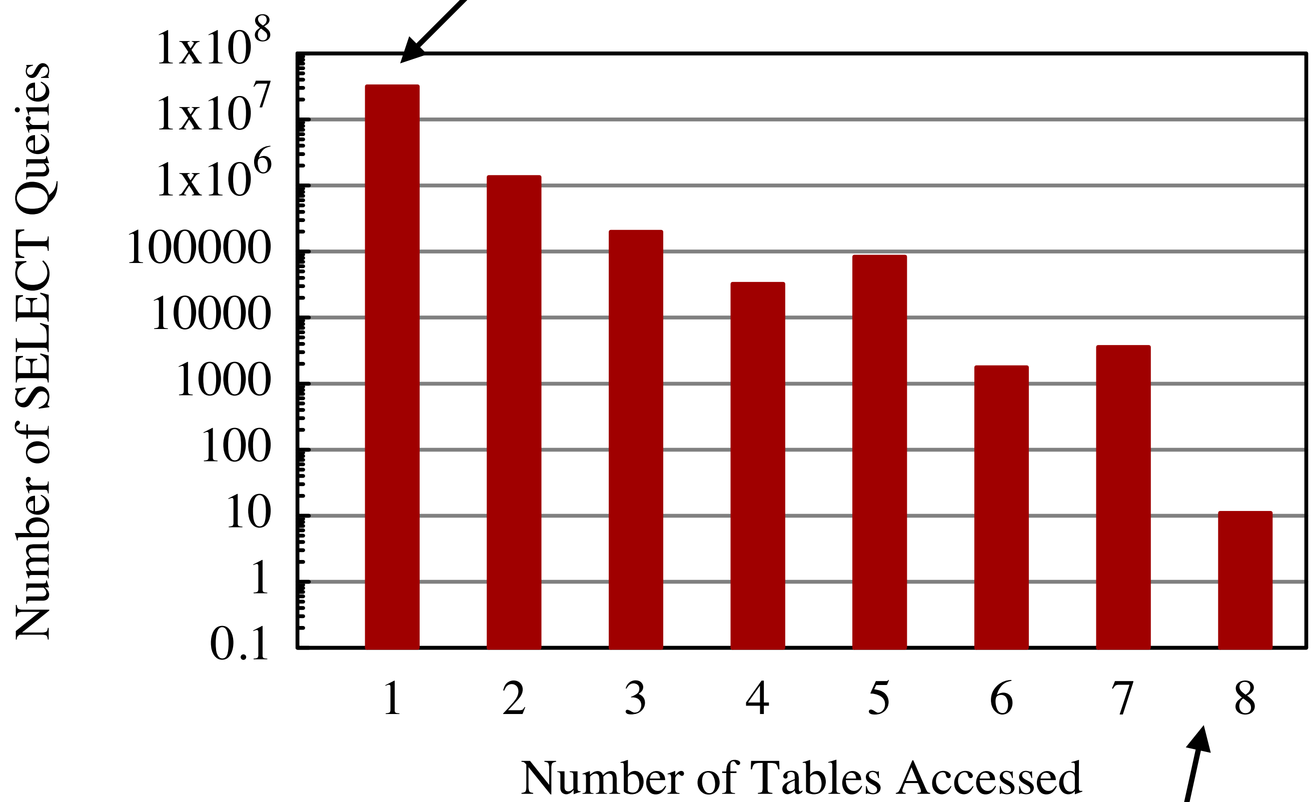
30 million simple "SPA" queries

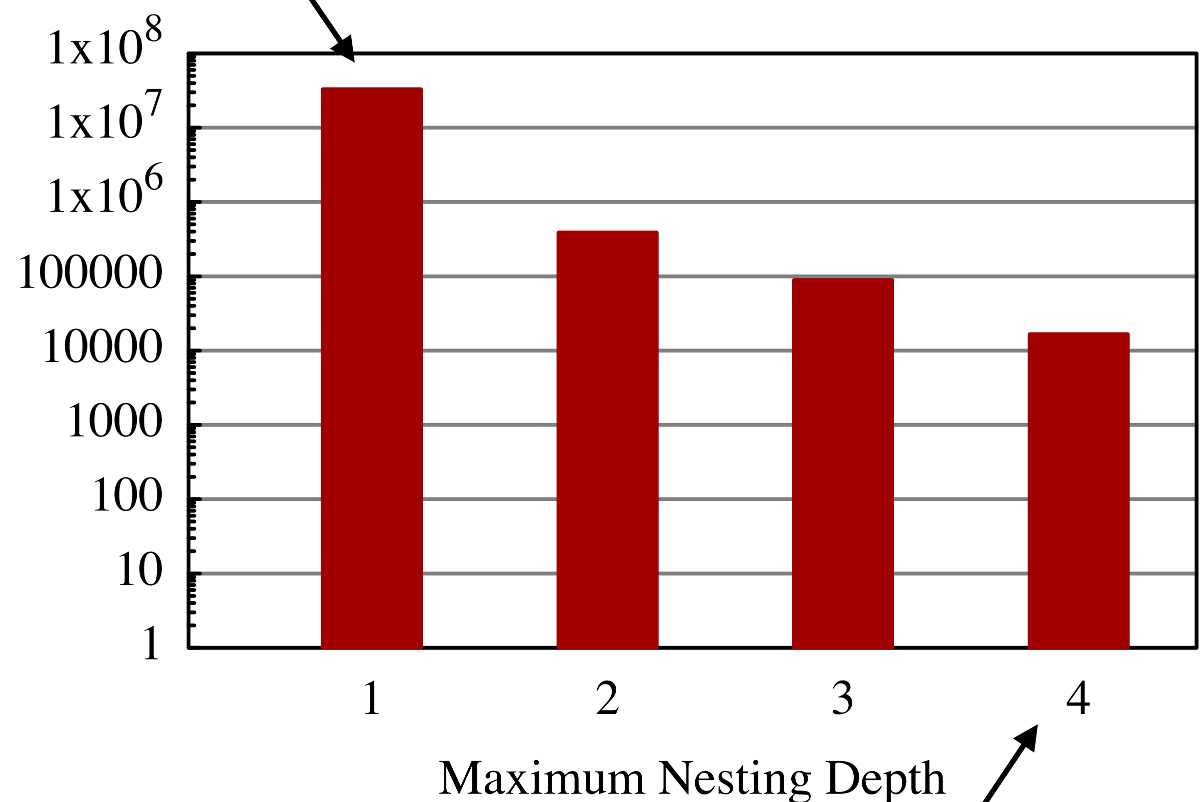




SELECT Complexity

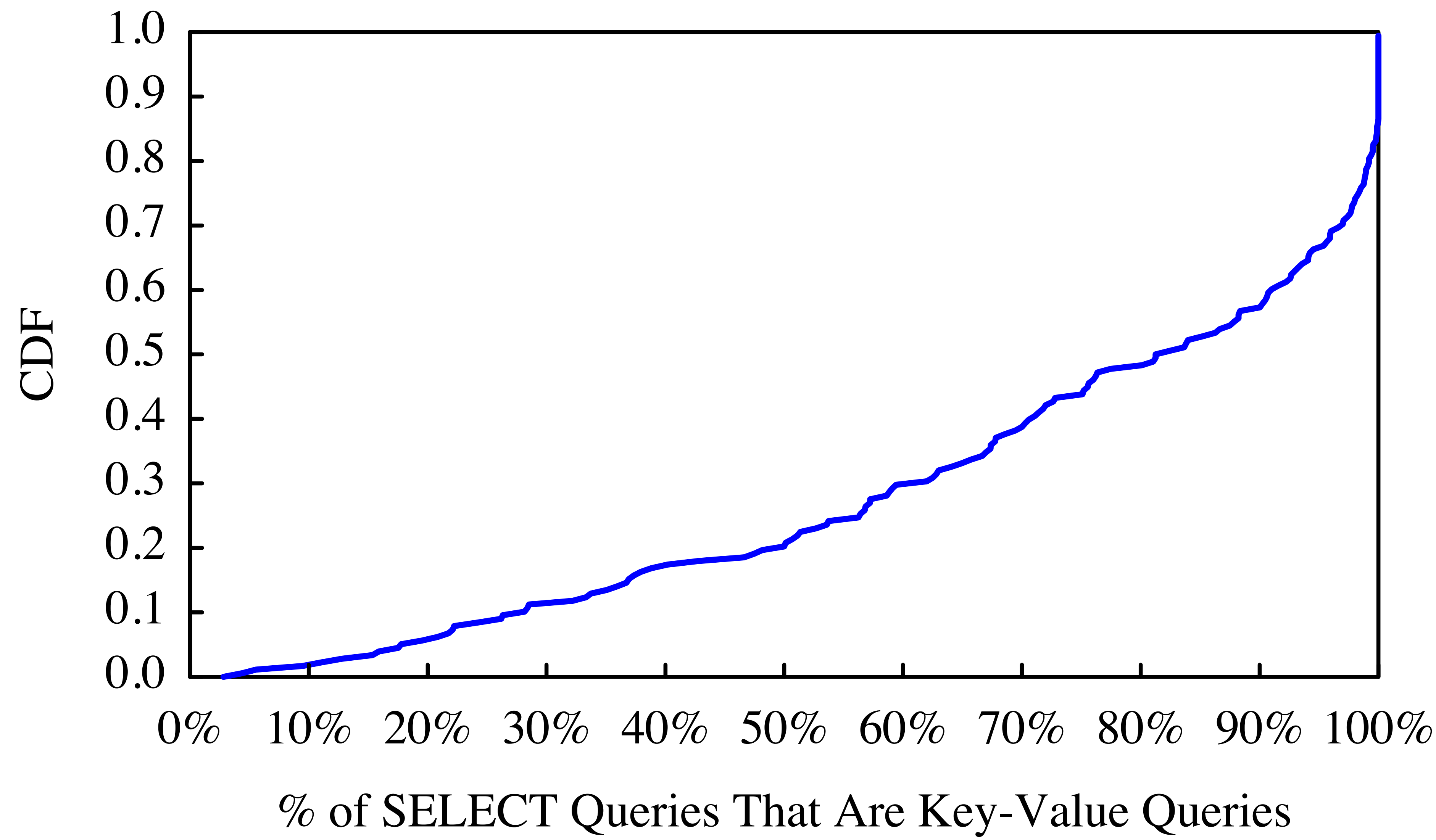
30 million simple "SPA" queries



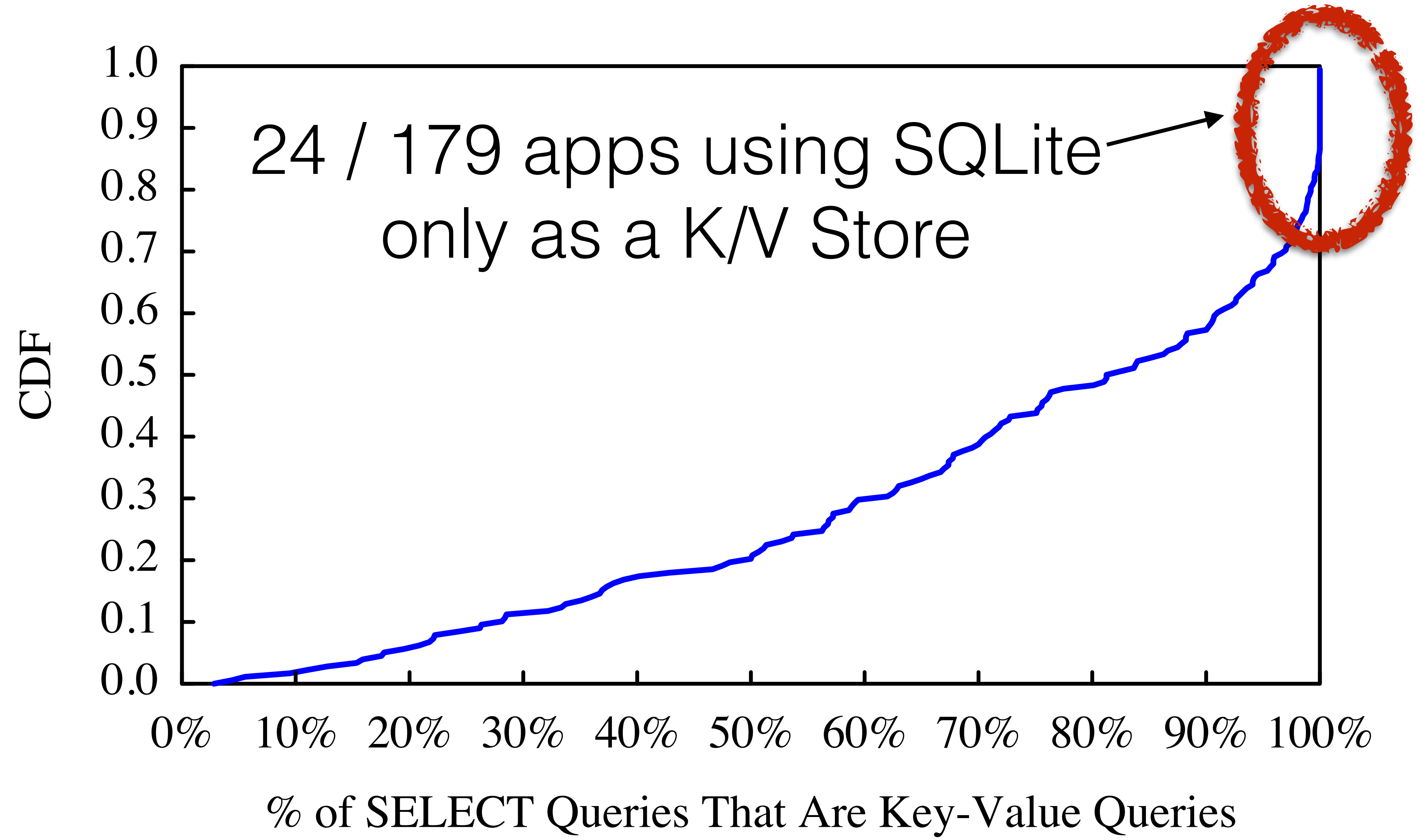


Infrequent, but extremely complex queries

SELECT Complexity
(by app)



SELECT Complexity
(by app)

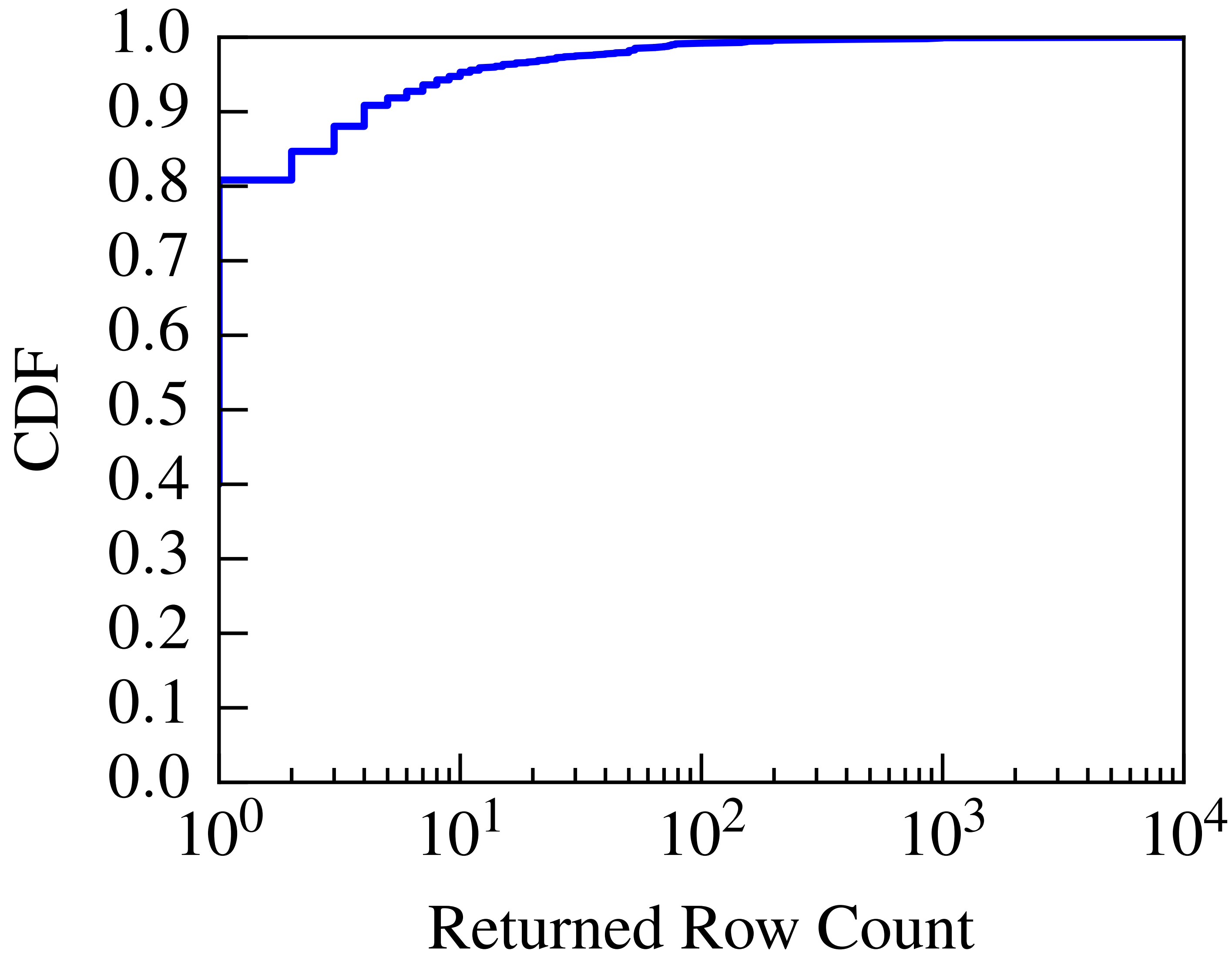


SELECT Complexity

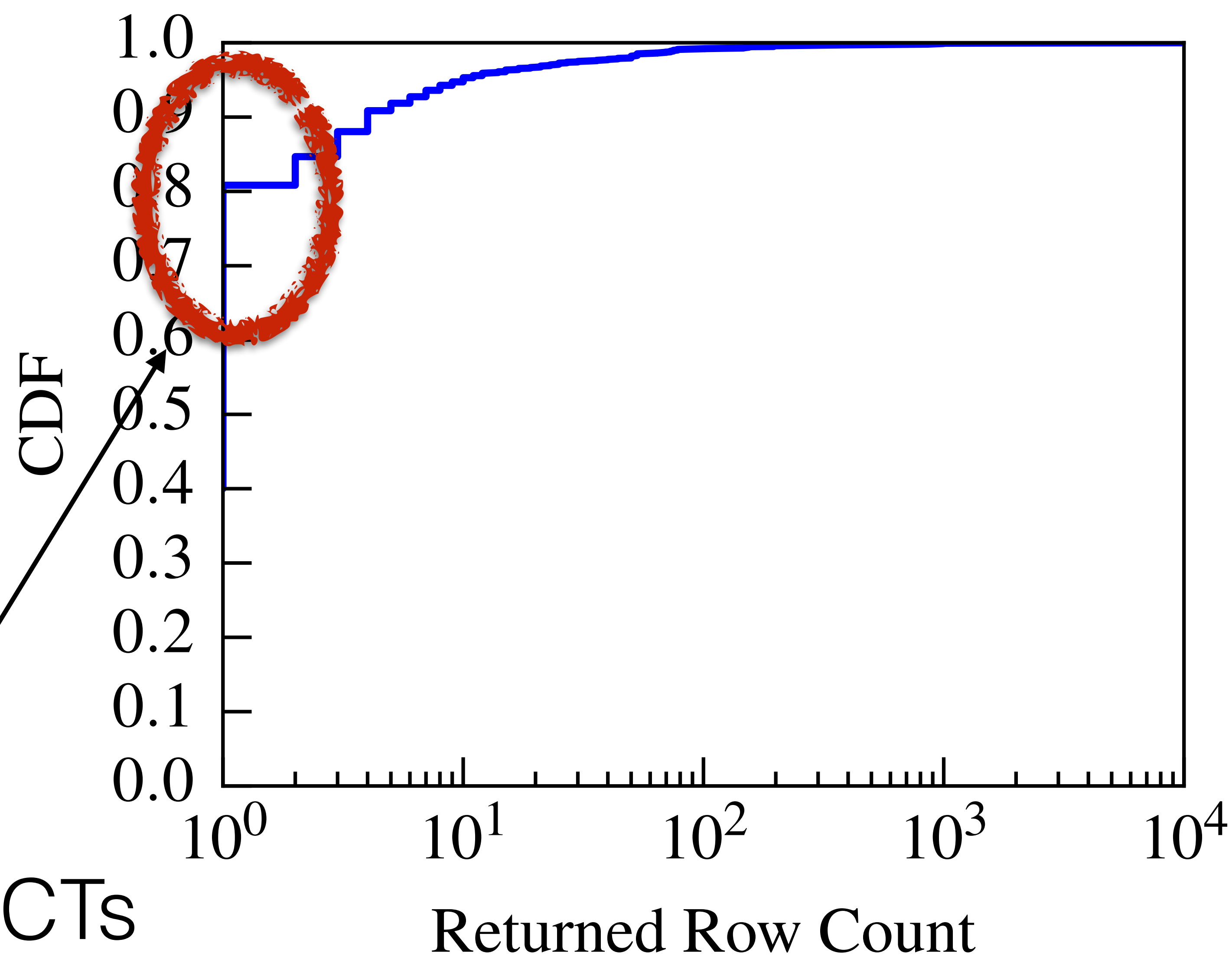
```
INSERT OR REPLACE INTO
    properties(property_key,property_value)
    VALUES (?,?);

SELECT property_value
    FROM properties
    WHERE property_key=?;
```

(These are actual real queries from the trace)

SELECT Complexity

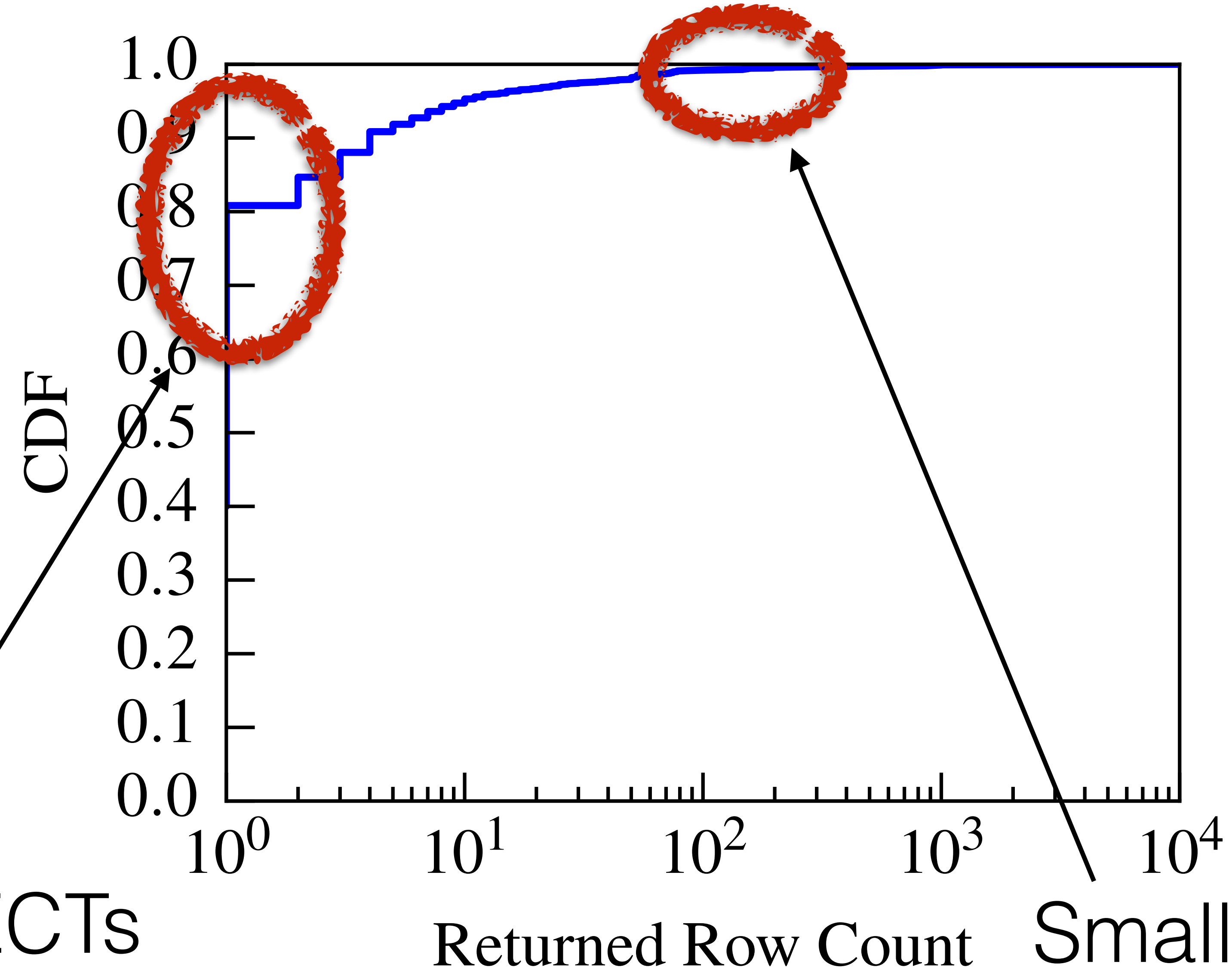


SELECT Complexity



80% of SELECTs return one row

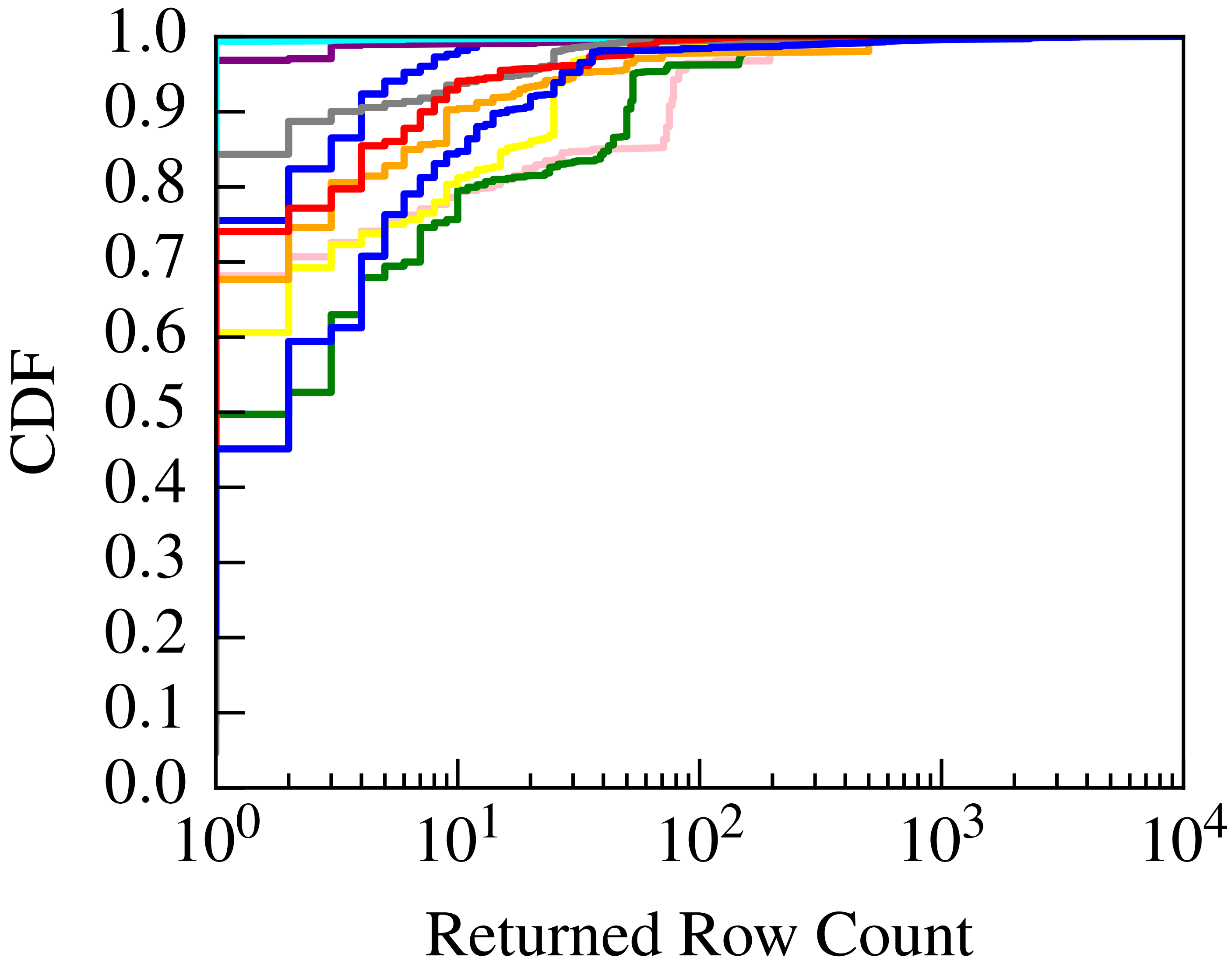
SELECT Complexity



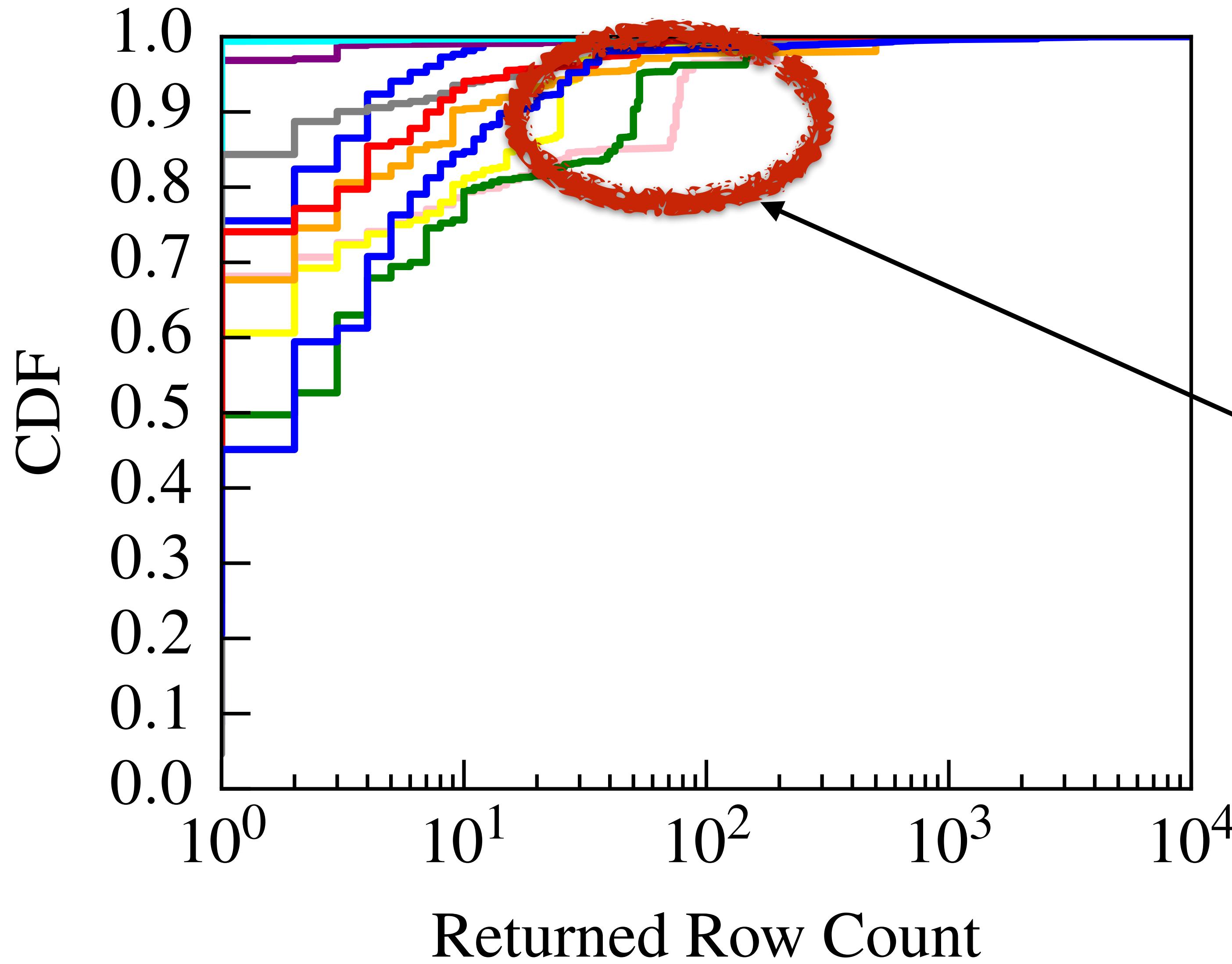
80% of SELECTs return one row

Small % of SELECTs return 100s of rows

SELECT Complexity
(by app)

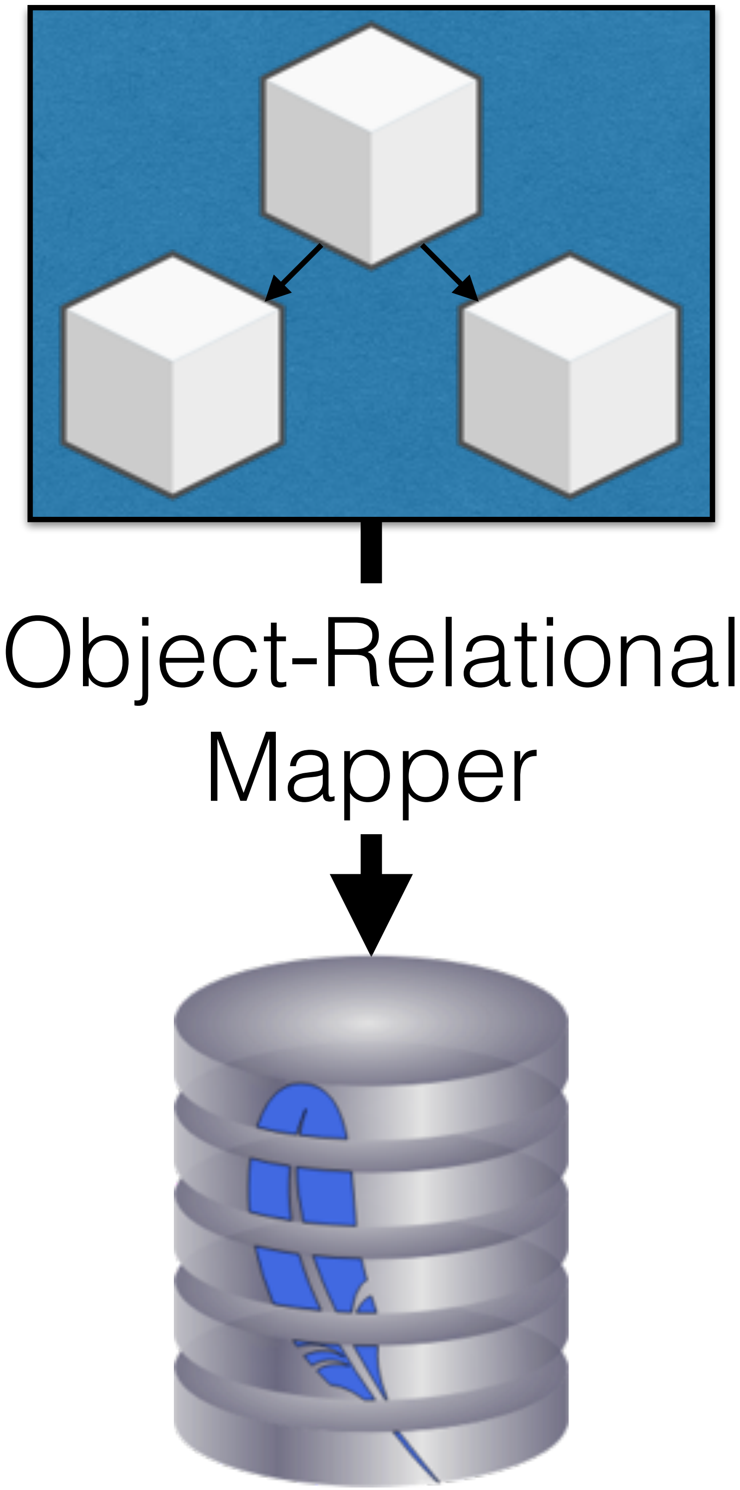


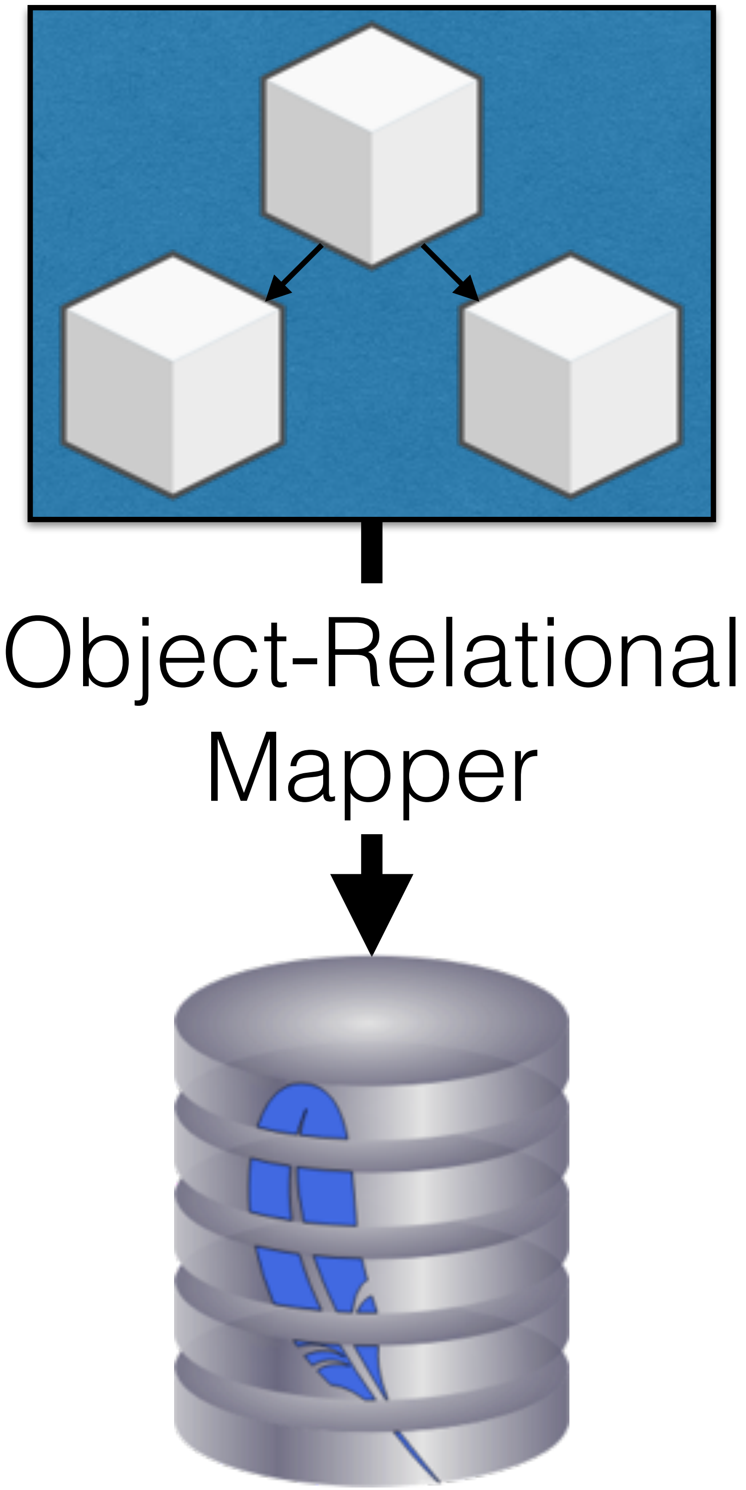
SELECT Complexity
(by app)



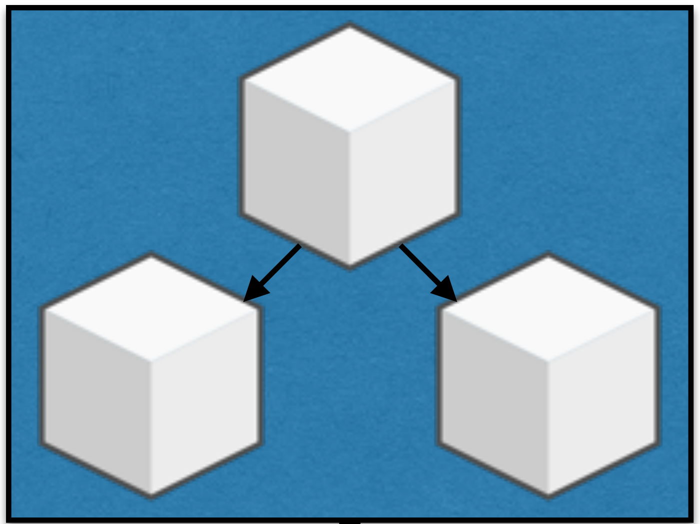
Google+, GMail, Facebook, Contacts

- SELECT Complexity
- ORM Effects
- Function Usage
- Read/Write Ratios
- Query Periodicity

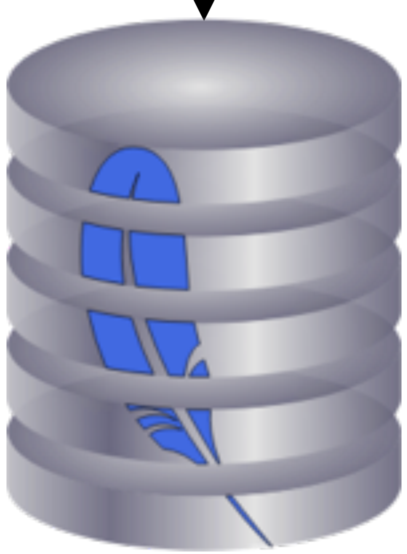

Object-Relational
Mapper




```
pers = Persons.get(10)
name = pers.firstName()
```



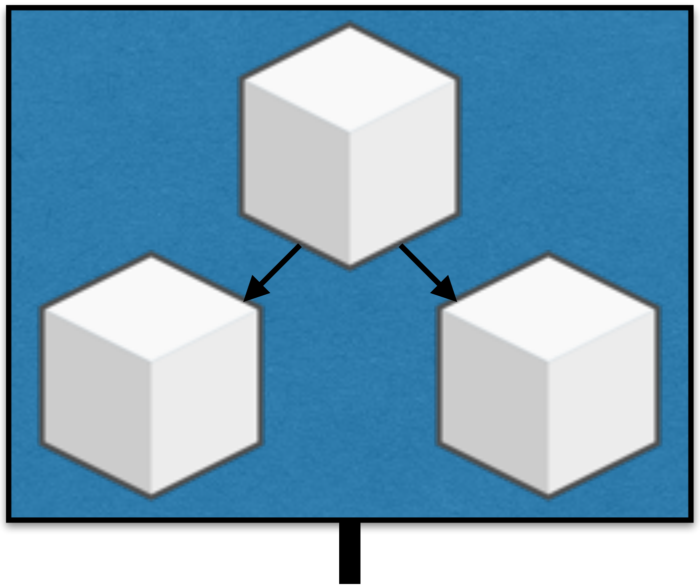
Object-Relational Mapper



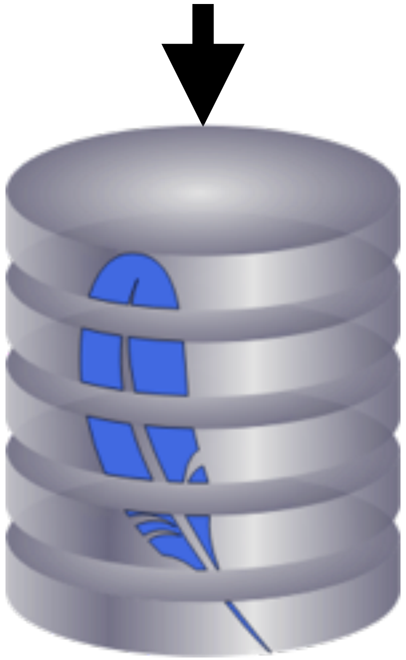
```
pers = Persons.get(10)
name = pers.firstName()
```

```
SELECT first_name
FROM Persons
WHERE id = 10;
```

Image courtesy of http://openclipart.org



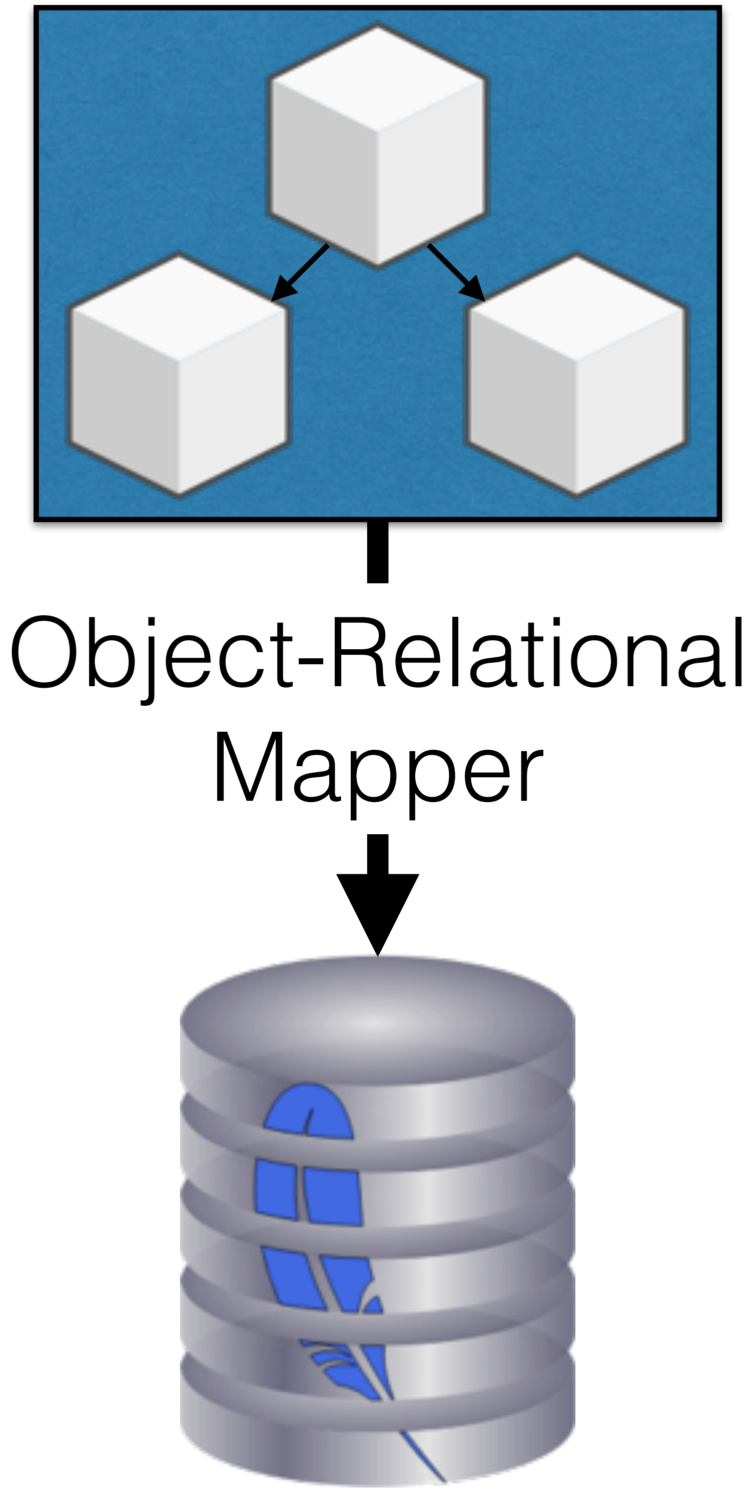
Object-Relational Mapper



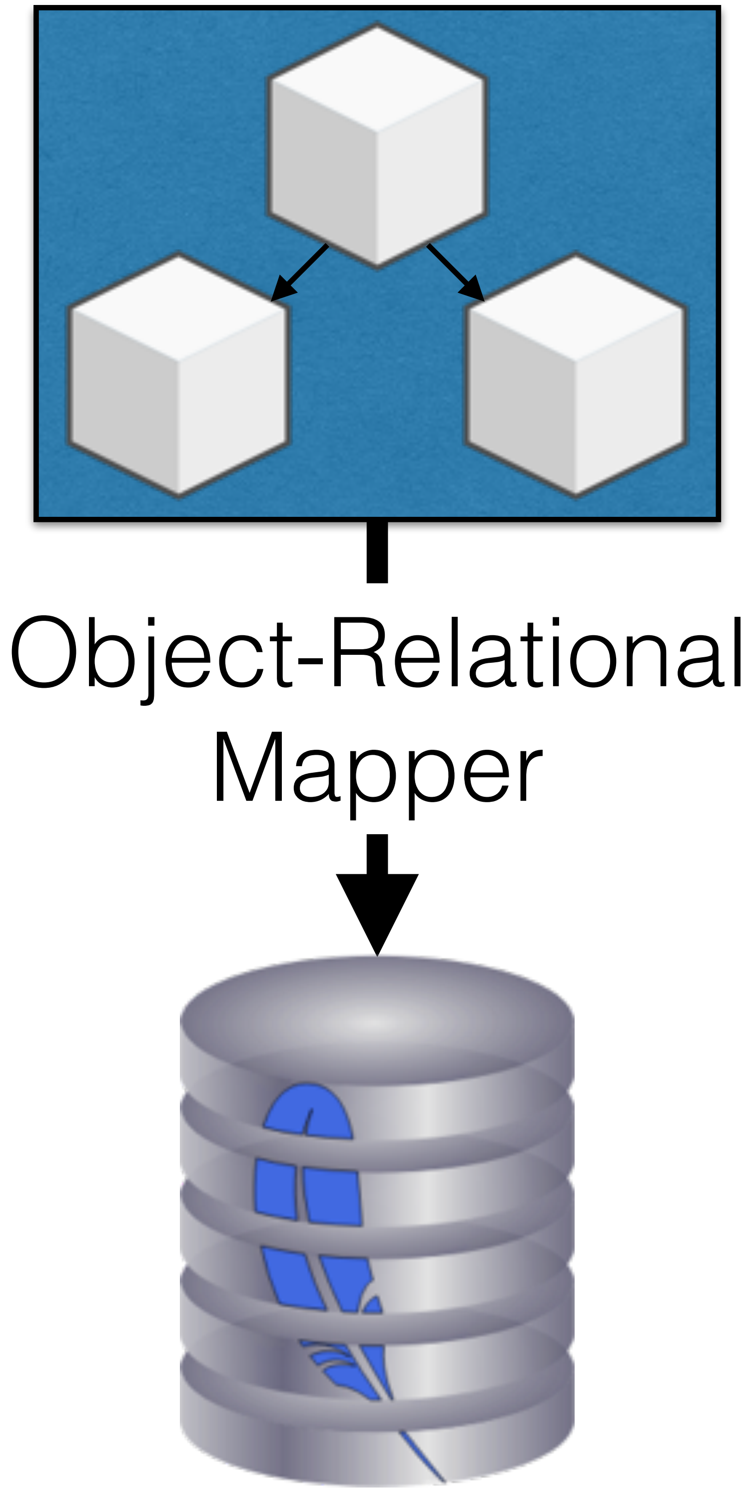
```
pers = Persons.get(10)
name = pers.firstName()
```

```
SELECT first_name
FROM Persons
WHERE id = 10;
```

SQL DB used for persisting objects

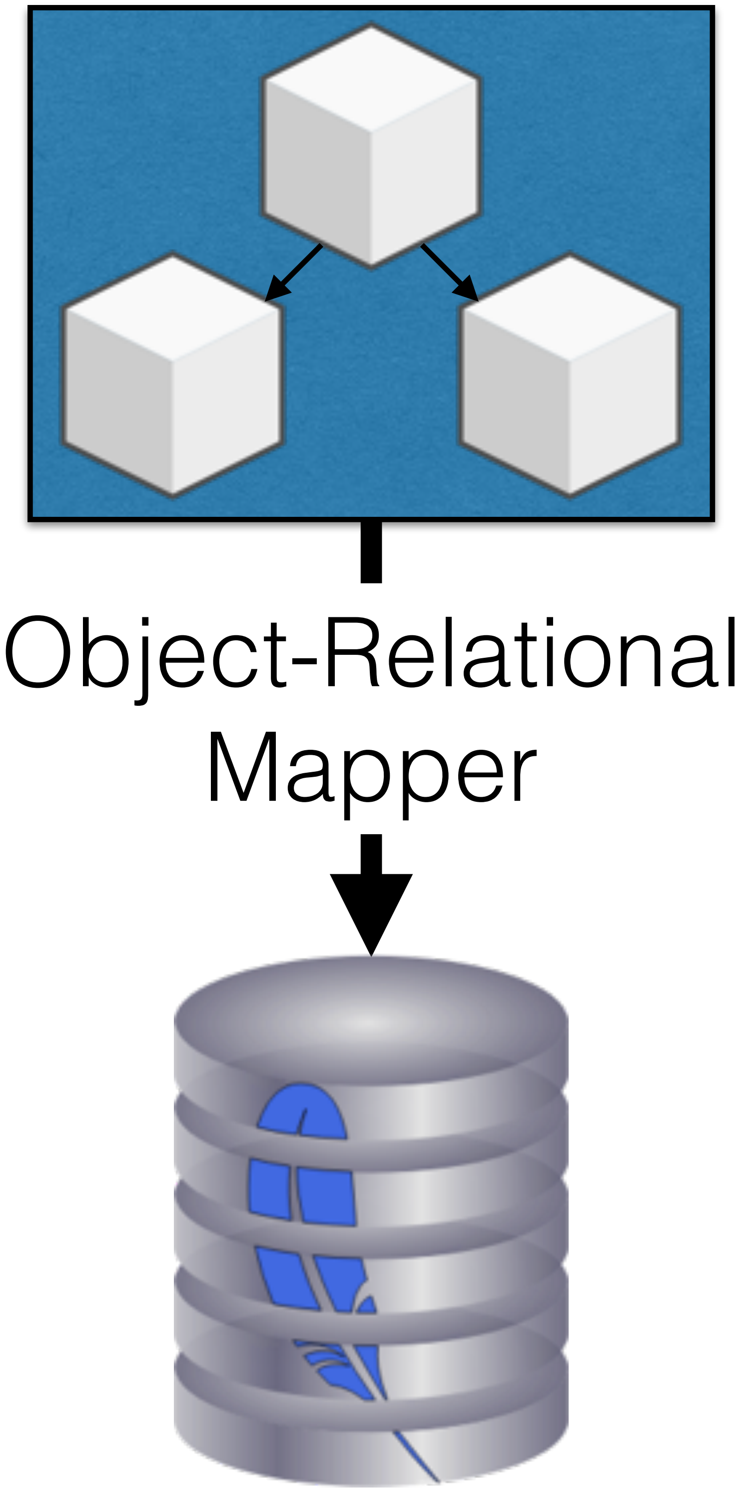



```
pers = Persons.get(10)
org = pers.employer()
name = org.name()
```




```
pers = Persons.get(10)
org = pers.employer()
name = org.name()
```

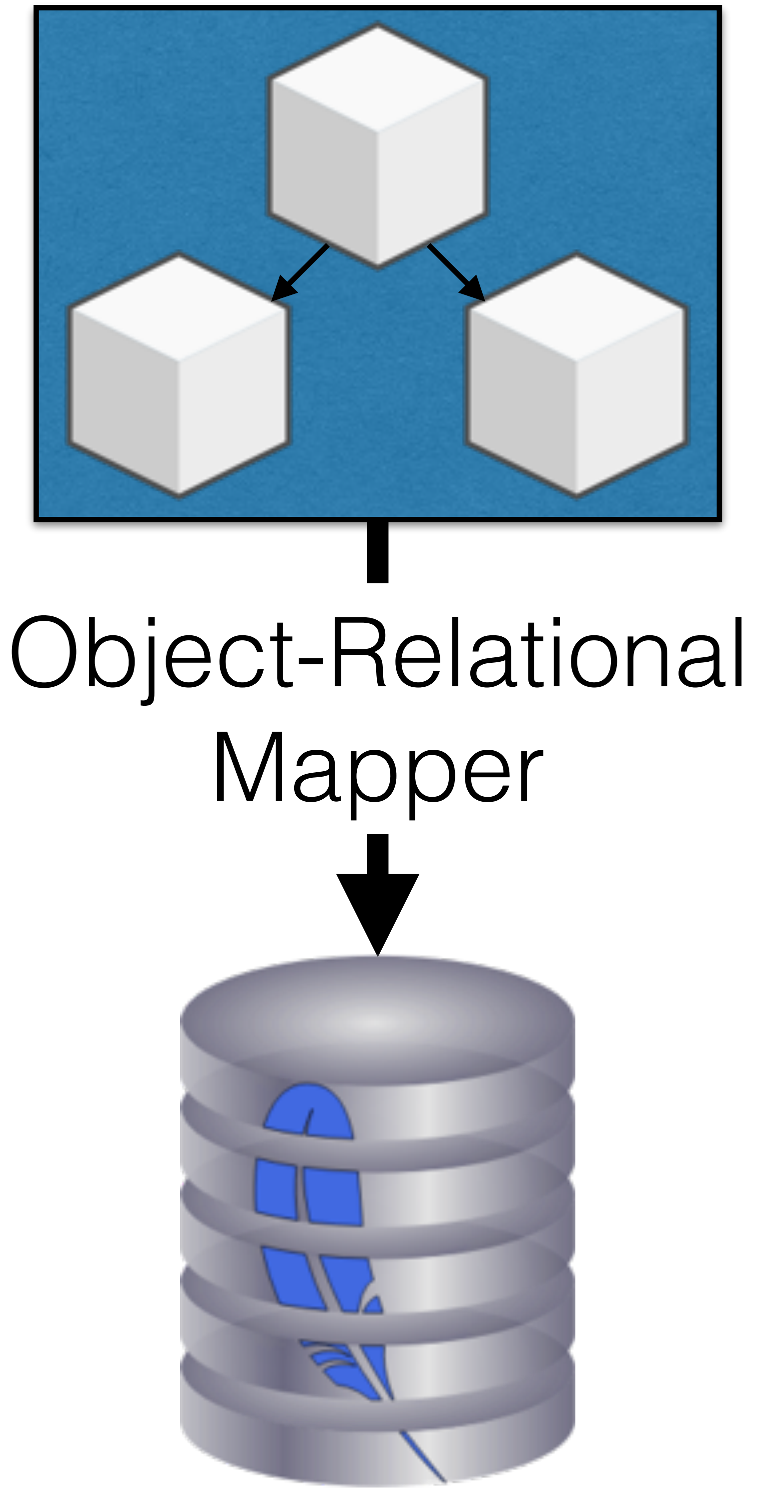
```
SELECT employer_id
FROM Persons
WHERE id = 10;

SELECT name
FROM Organizations
WHERE id = ?;
```



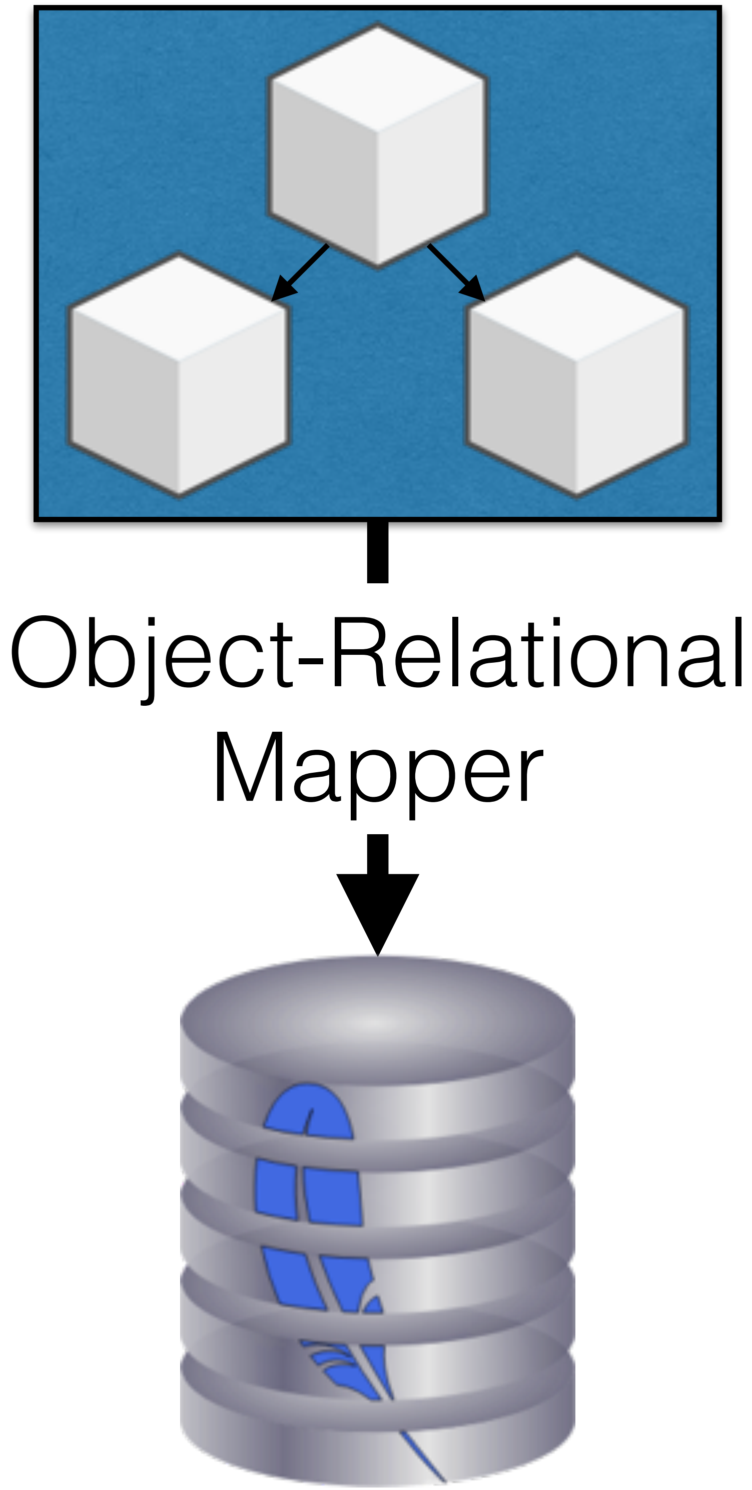

```
pers = Persons.get(10)
org = pers.employer()
name = org.name()
```

```
SELECT employer_id
FROM Persons
WHERE id = 10;

SELECT name
FROM Organizations
WHERE id = ?;
```

ORMs are not always efficient



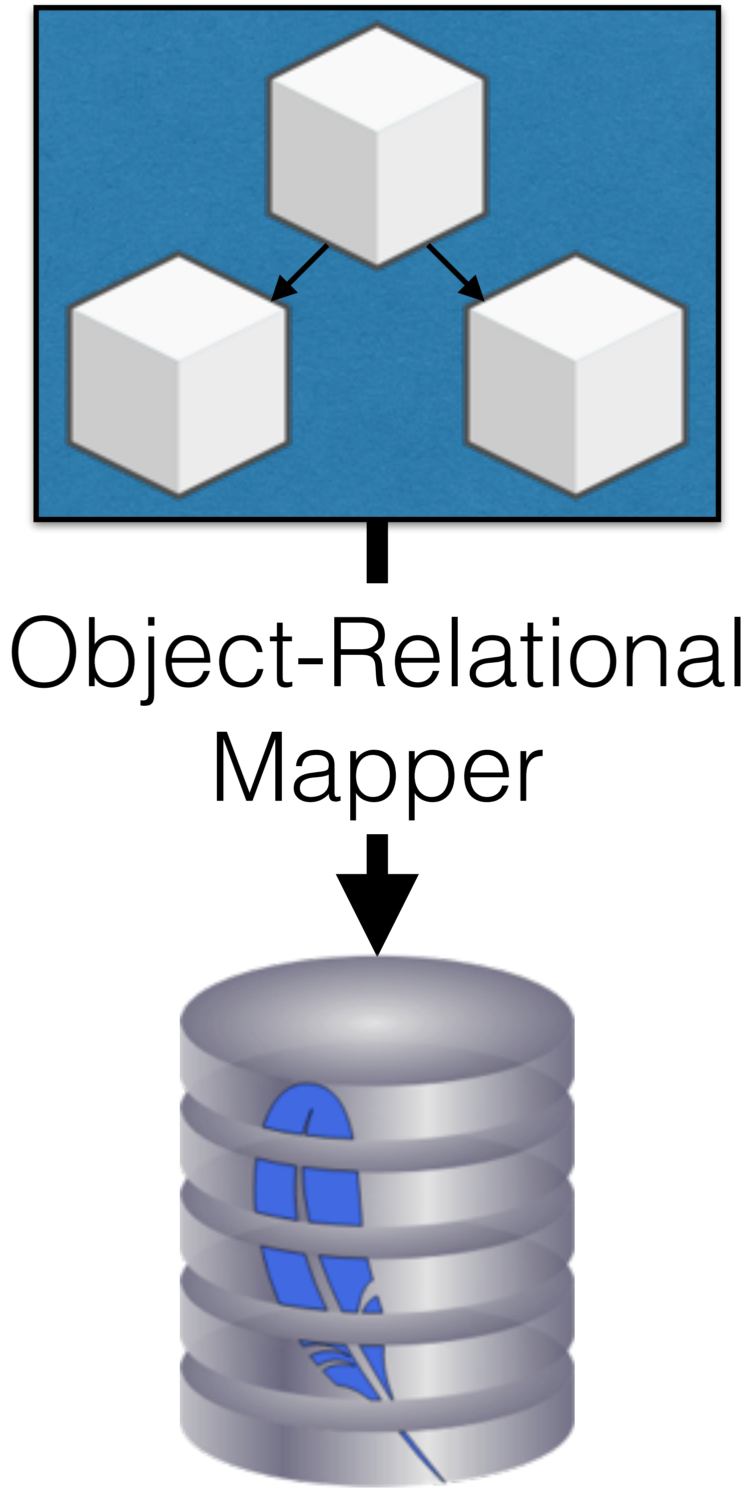

```
pers = Persons.get(10)
pers.setSalary(
  pers.salary() * 1.1
)
```




```
pers = Persons.get(10)
pers.setSalary(
  pers.salary() * 1.1
)
```

```
SELECT salary
FROM Persons
WHERE id = 10;

UPDATE Persons
SET salary = ?
WHERE id = 10;
```

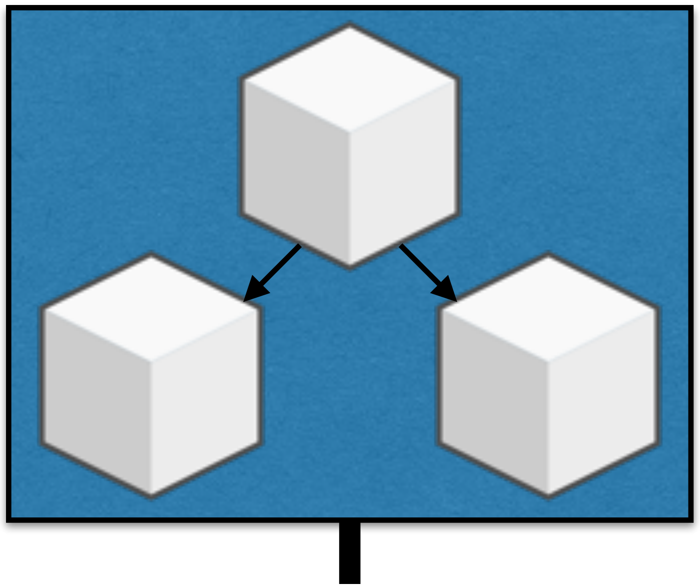
Image courtesy of http://openclipart.org



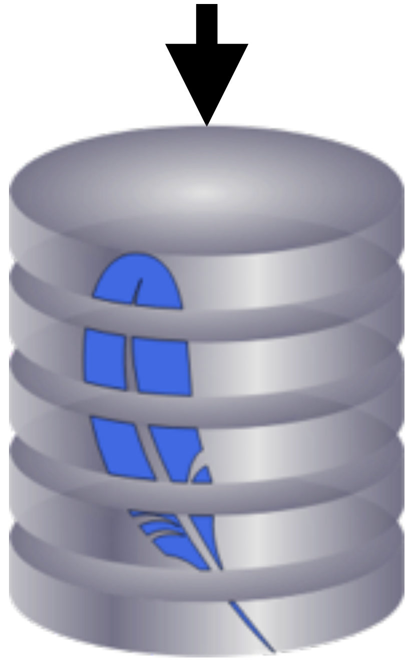

```
pers = Persons.get(10)
pers.setSalary(
  pers.salary() * 1.1
)
```

```
SELECT salary
FROM Persons
WHERE id = 10;

UPDATE Persons
SET salary = ?
WHERE id = 10;
```

We saw NO update value computations in SQL

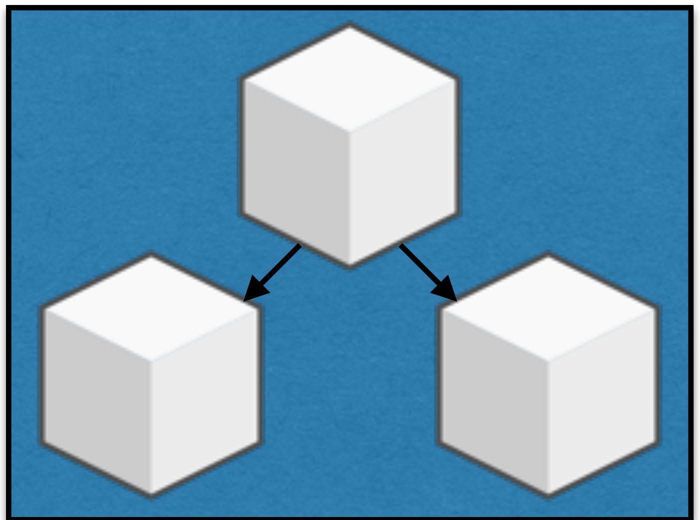


Object-Relational
Mapper

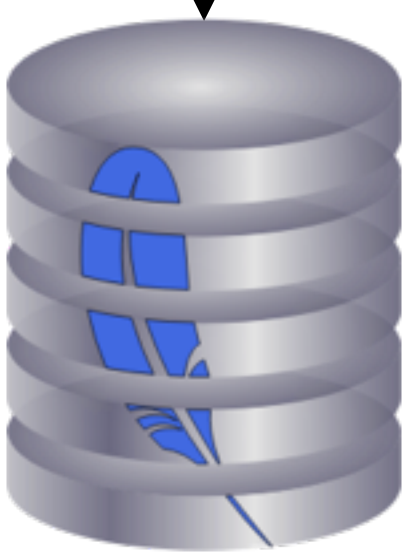


```
pers = Persons.get(10)
pers.setSalary(
  pers.salary() * 1.1
)
```

```
SELECT salary
FROM Persons
WHERE id = 10;

INSERT OR REPLACE INTO
 Persons(id, salary)
VALUES (?, 10);
```



Object-Relational Mapper



```
pers = Persons.get(10)
pers.setSalary(
  pers.salary() * 1.1
)
```

```
SELECT salary
FROM Persons
WHERE id = 10;

INSERT OR REPLACE INTO
 Persons(id, salary)
VALUES (?, 10);
```

Insert or Replace used very frequently

- SELECT Complexity
- ORM Effects

Function Usage

- Read/Write Ratios
- Query Periodicity

Aggregates

Function	Call Sites
GROUP_CONCAT	583,474
SUM	321,387
MAX	314,970
COUNT	173,031
MIN	19,566
AVG	15

Aggregates

Function	Call Sites
GROUP_CONCAT	583,474
SUM	321,387
MAX	314,970
COUNT	173,031
MIN	19,566
AVG	15

Aggregates most common function type

Aggregates

Function	Call Sites
GROUP_CONCAT	583,474
SUM	321,387
MAX	314,970
COUNT	173,031
MIN	19,566
AVG	15

Concatenate all strings in a column: Non-algebraic

Other Functions

Other Functions

- Mostly string manipulation (`length, substr`)

Other Functions

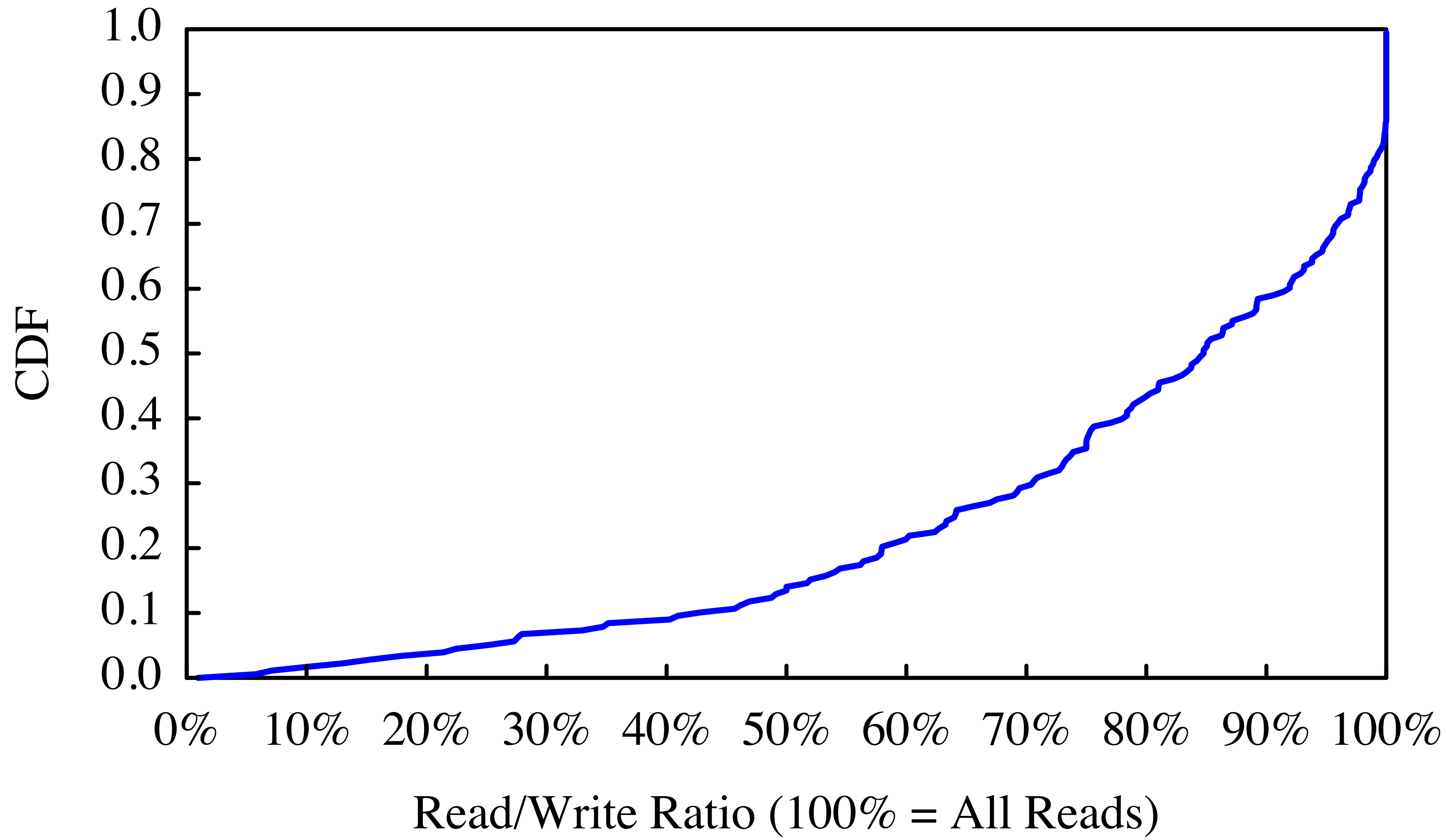
- Mostly string manipulation (`length, substr`)
- Some Android-Specific (`phone_numbers_equal`)

Other Functions

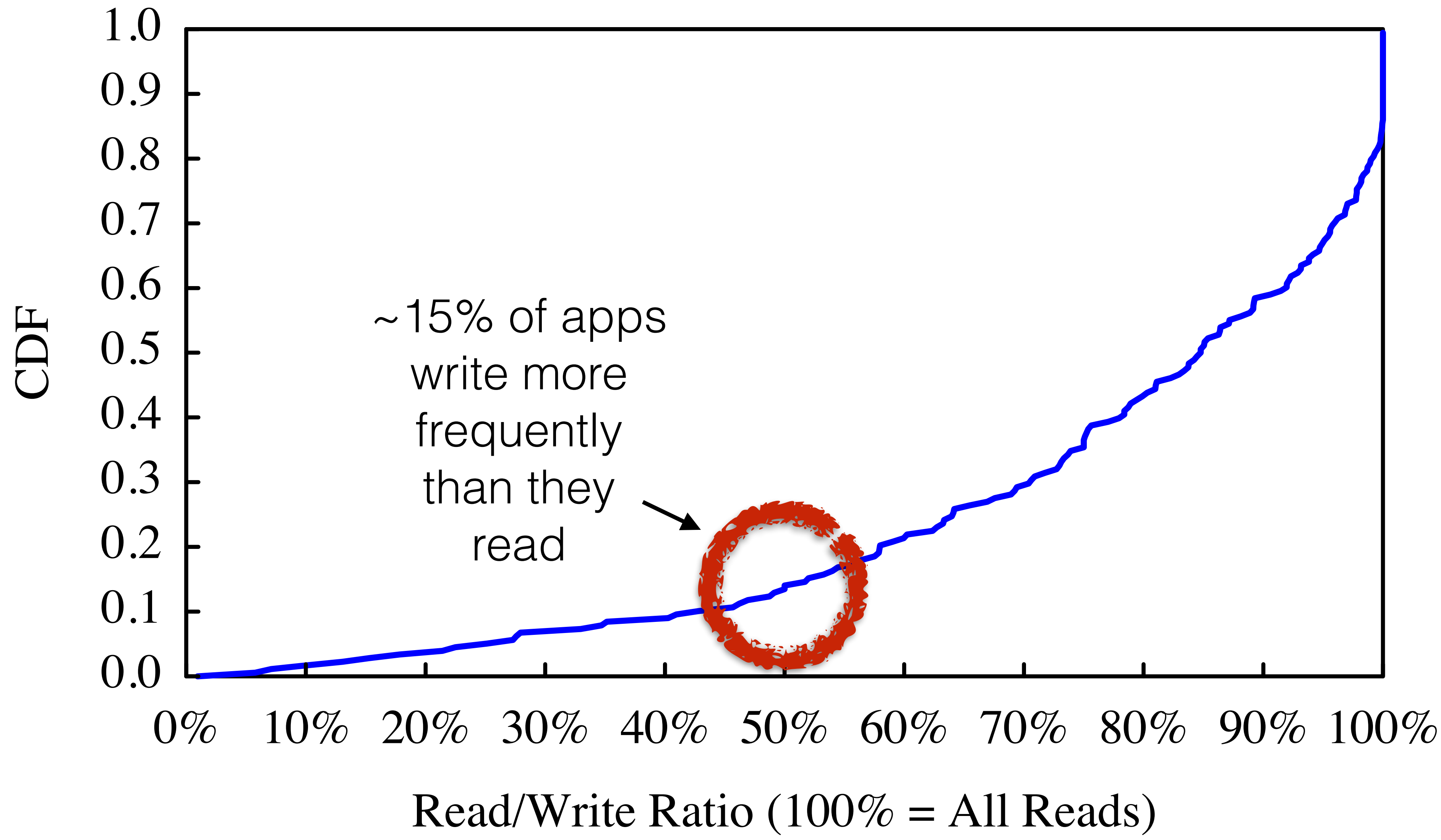
- Mostly string manipulation (`length, substr`)
- Some Android-Specific (`phone_numbers_equal`)
- **NO** UDFs at all

- SELECT Complexity
- ORM Effects
- Function Usage
- Read/Write Ratios
- Query Periodicity

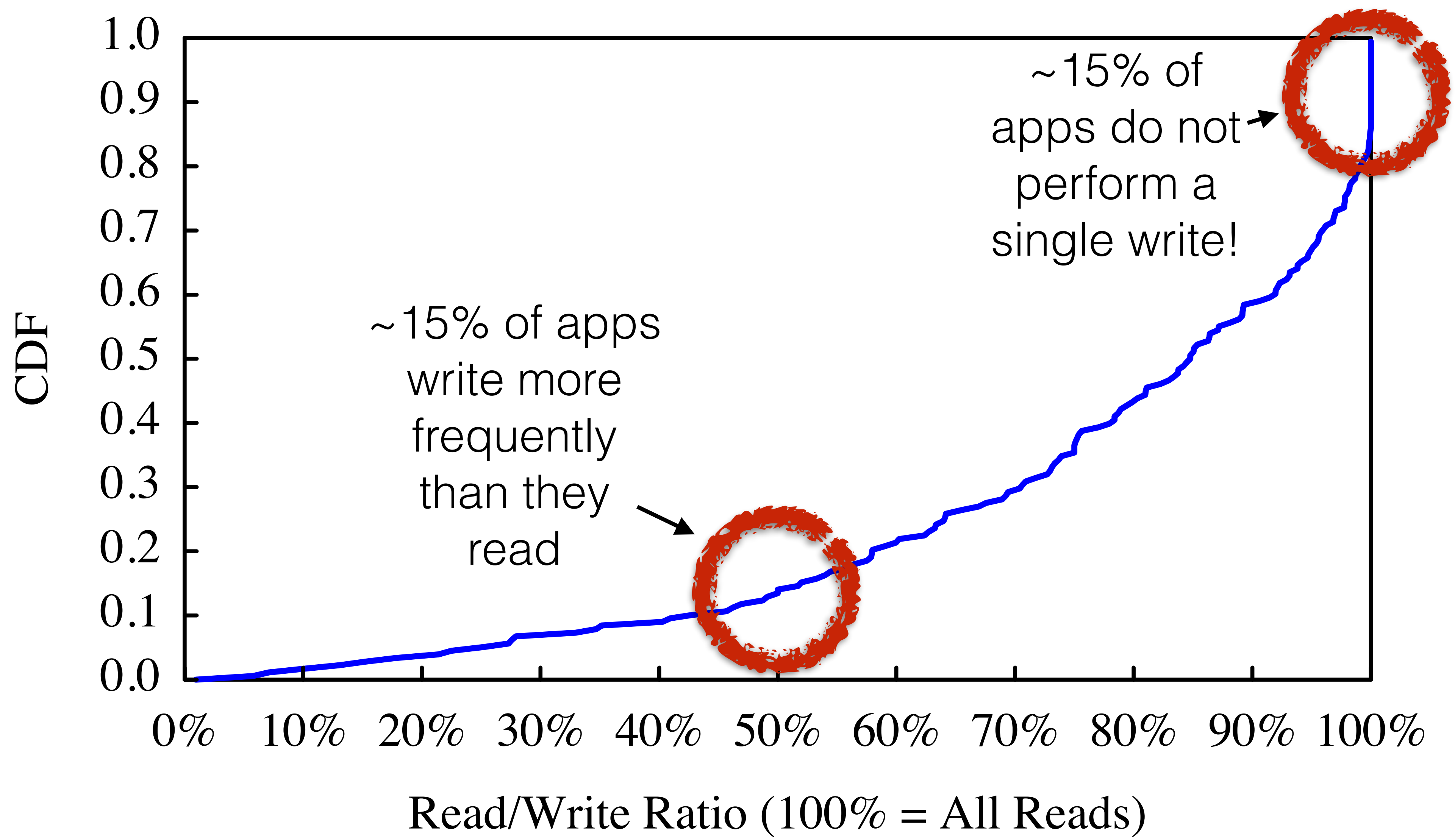
Reads vs Writes



Reads vs Writes



Reads vs Writes



Read-Only Workloads

Read-Only Workloads

- *JuiceSSH, Key Chain*
  - Credential store, infrequent writes

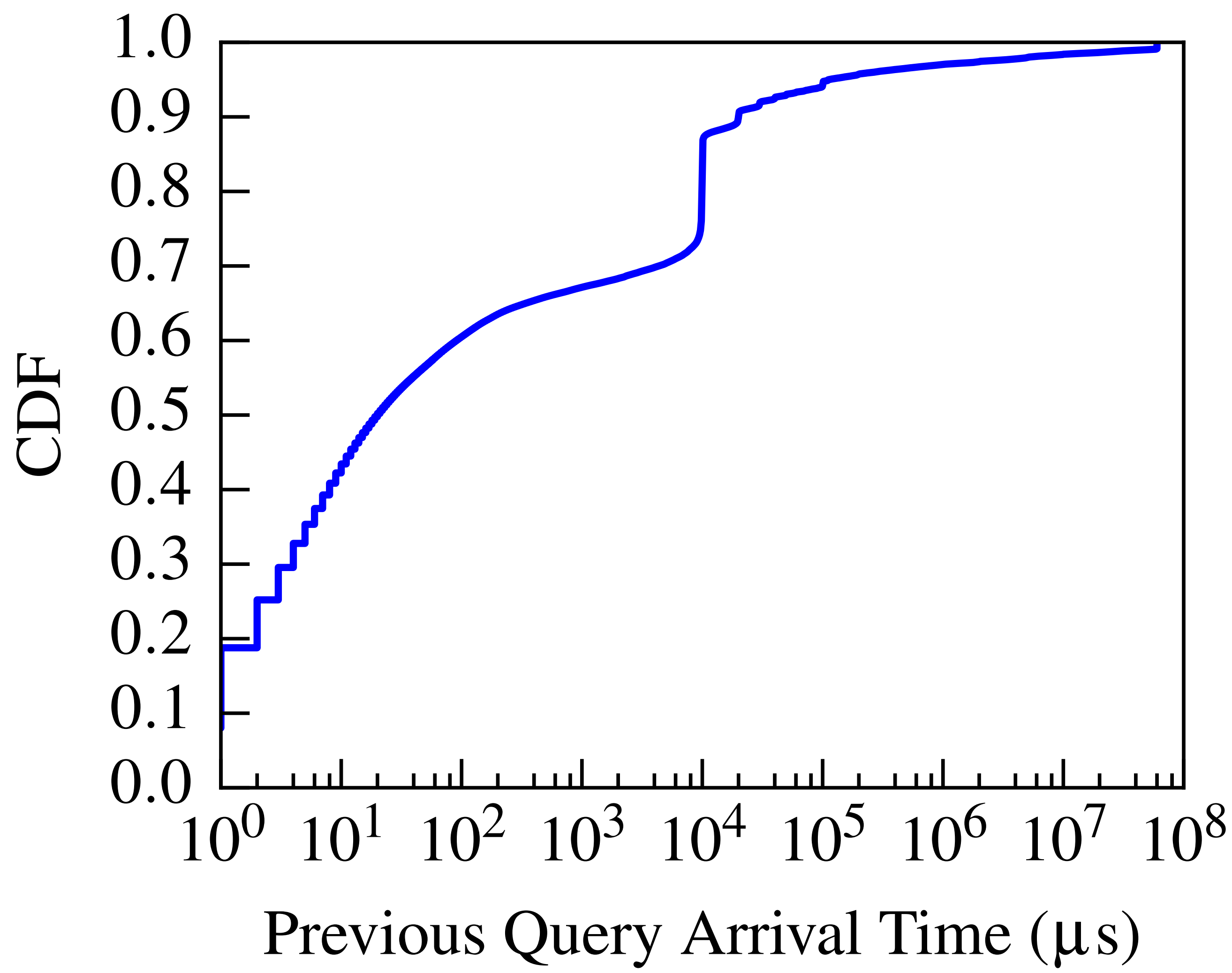
Read-Only Workloads

- *JuiceSSH, Key Chain*
  - Credential store, infrequent writes
- *Google Play Newsstand, Eventbrite, …*
  - Frequent queries over changing data
  - Data bulk updated by **copying entire SQLite DB**

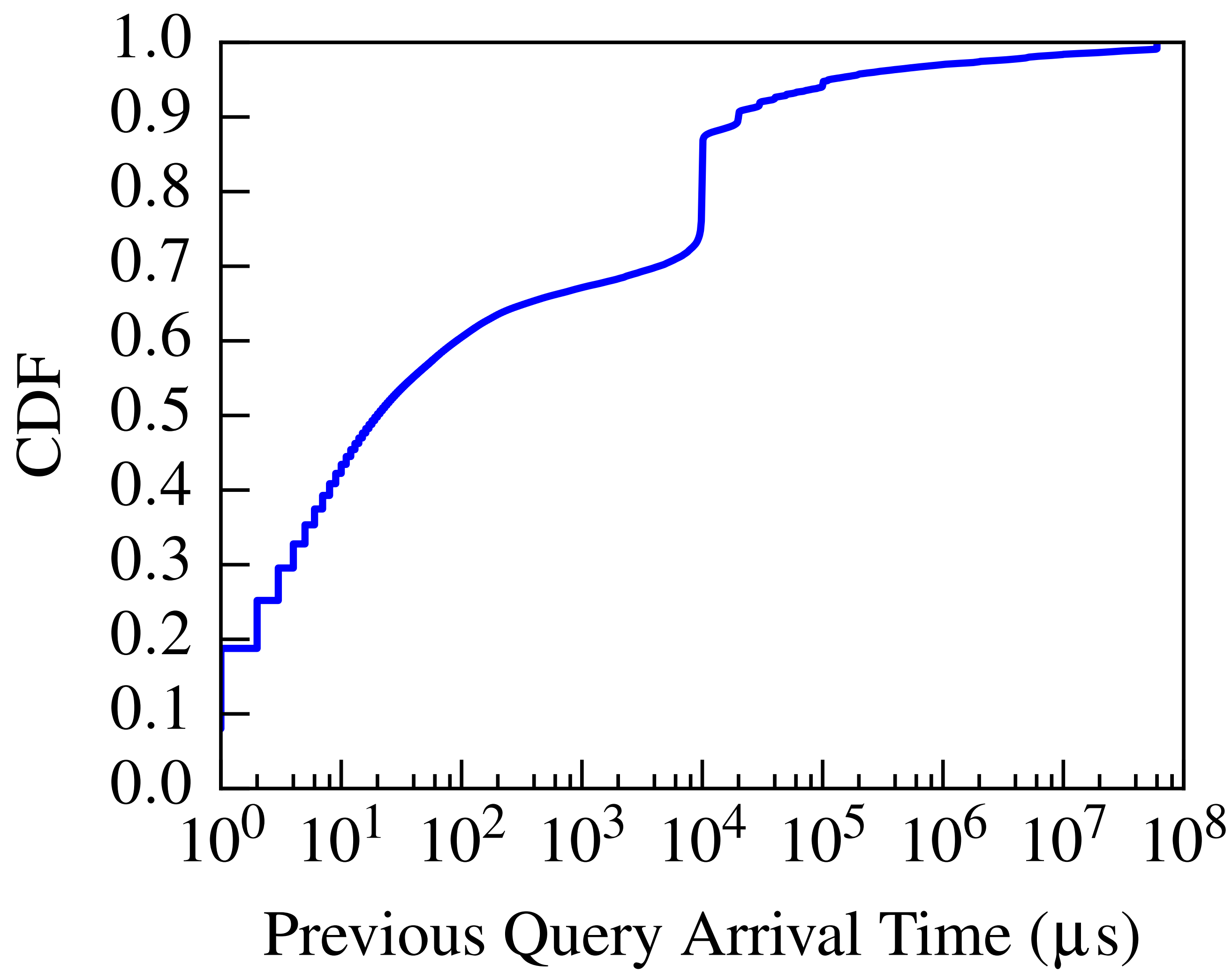
- SELECT Complexity
- ORM Effects
- Function Usage
- Read/Write Ratios

Query Periodicity

Query Arrival Frequency

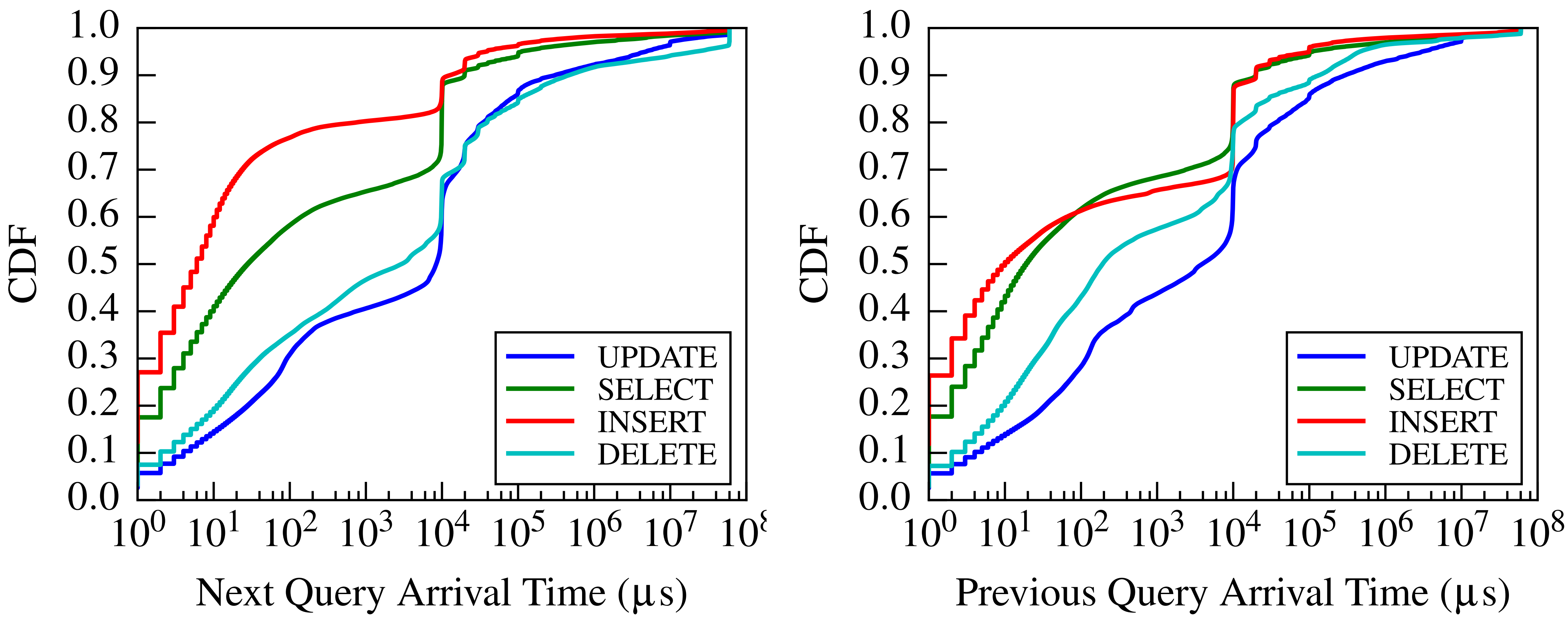


Query Arrival Frequency



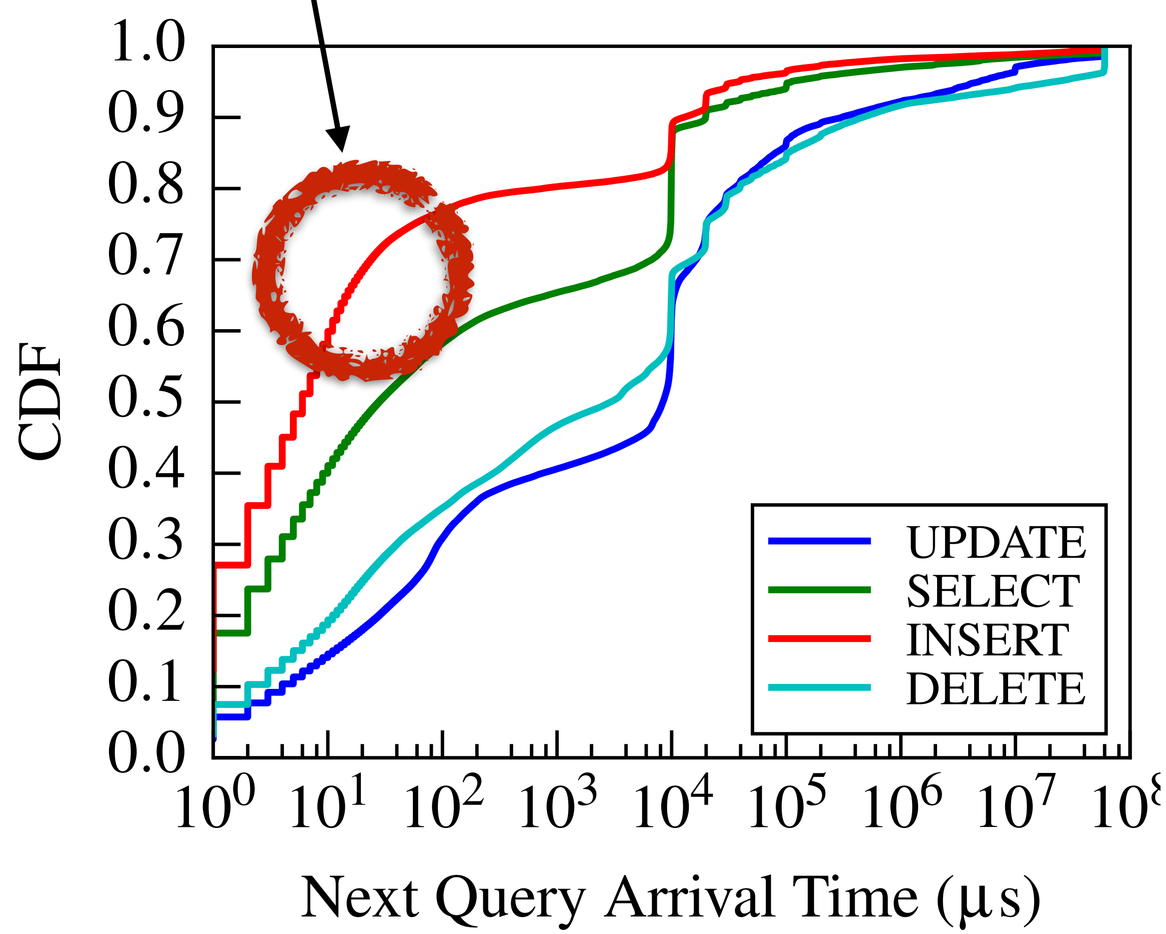
15-20% of queries arrive ~10ms after last query

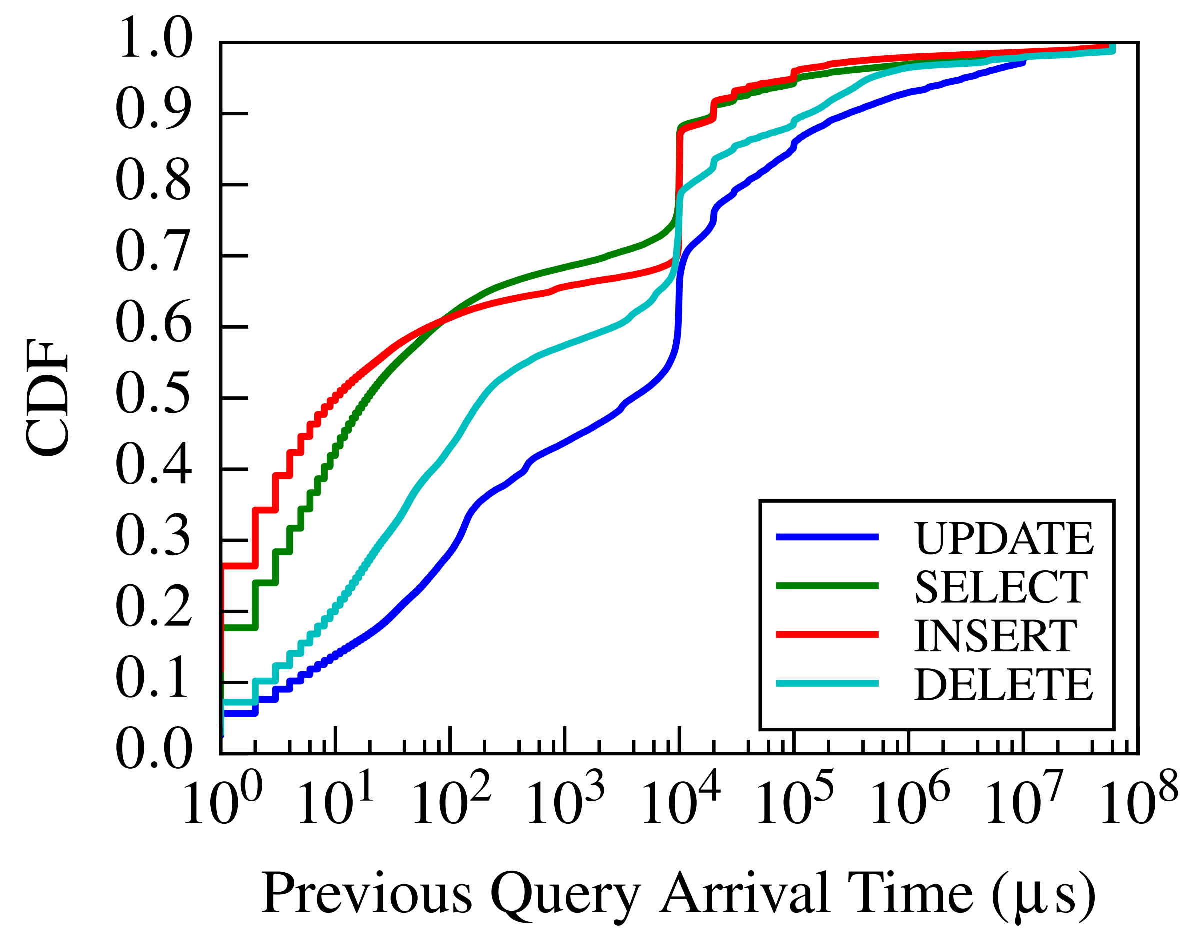
By Query Type



By Query Type

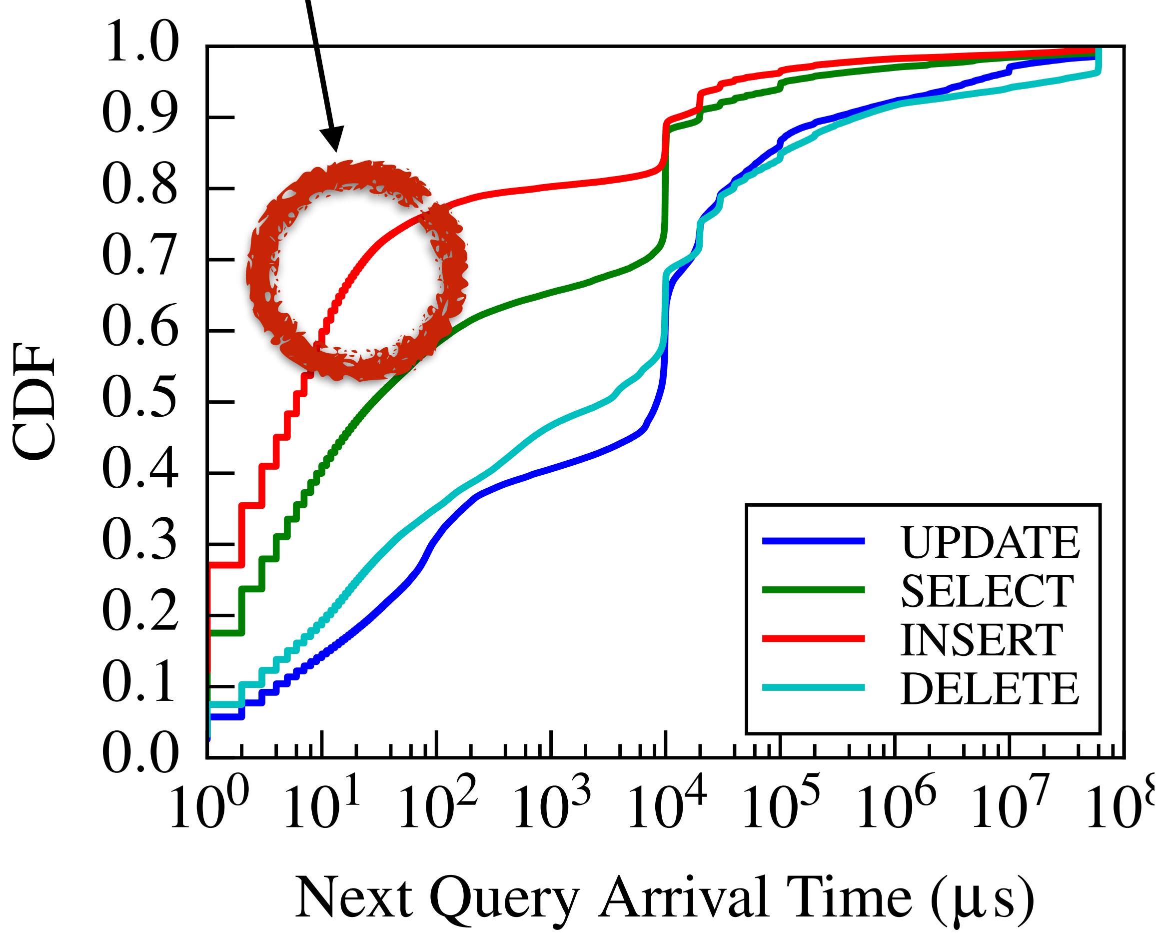
70% of inserts come less than
0.1 ms before another query



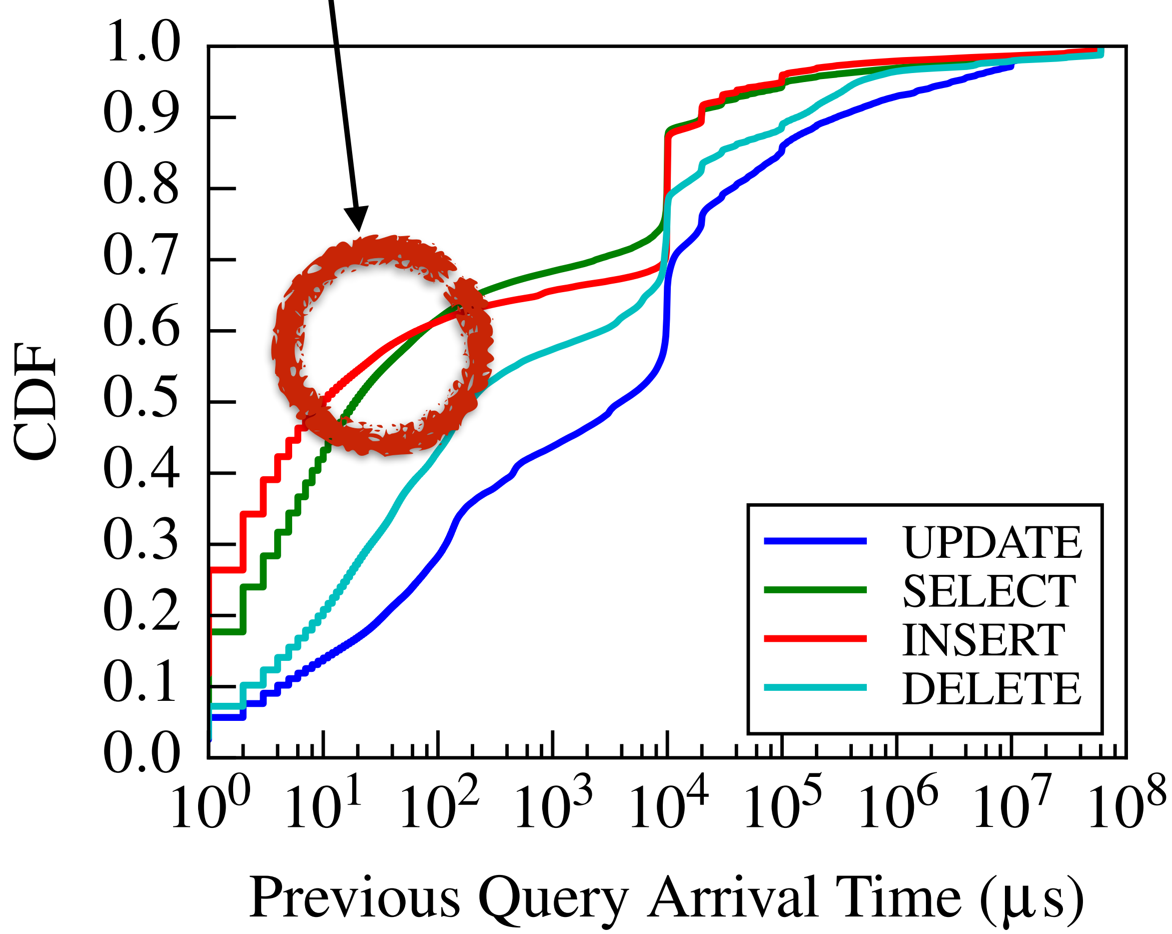


By Query Type

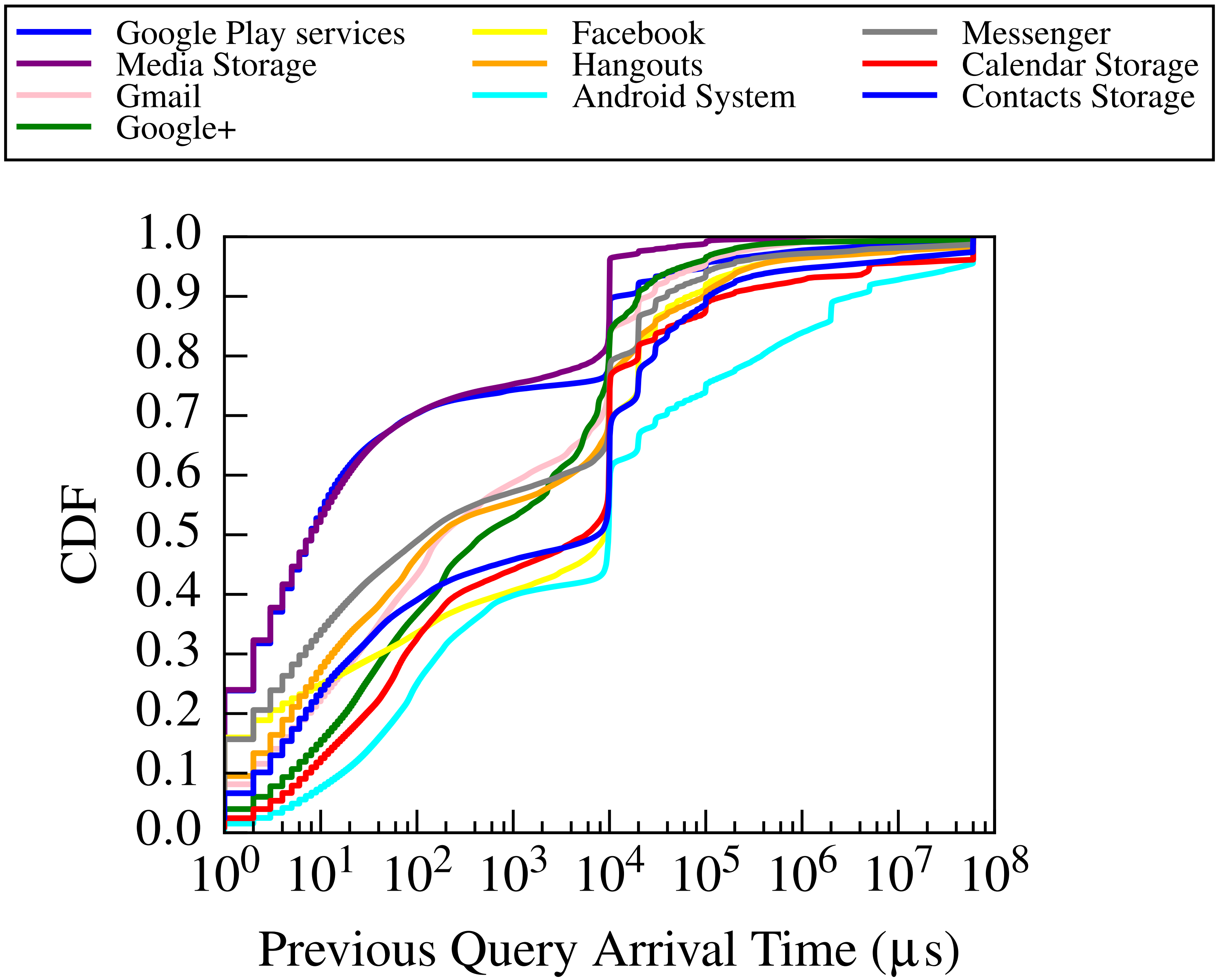
70% of inserts come less than 0.1 ms before another query



Most sequences consist of INSERTs and SELECTs



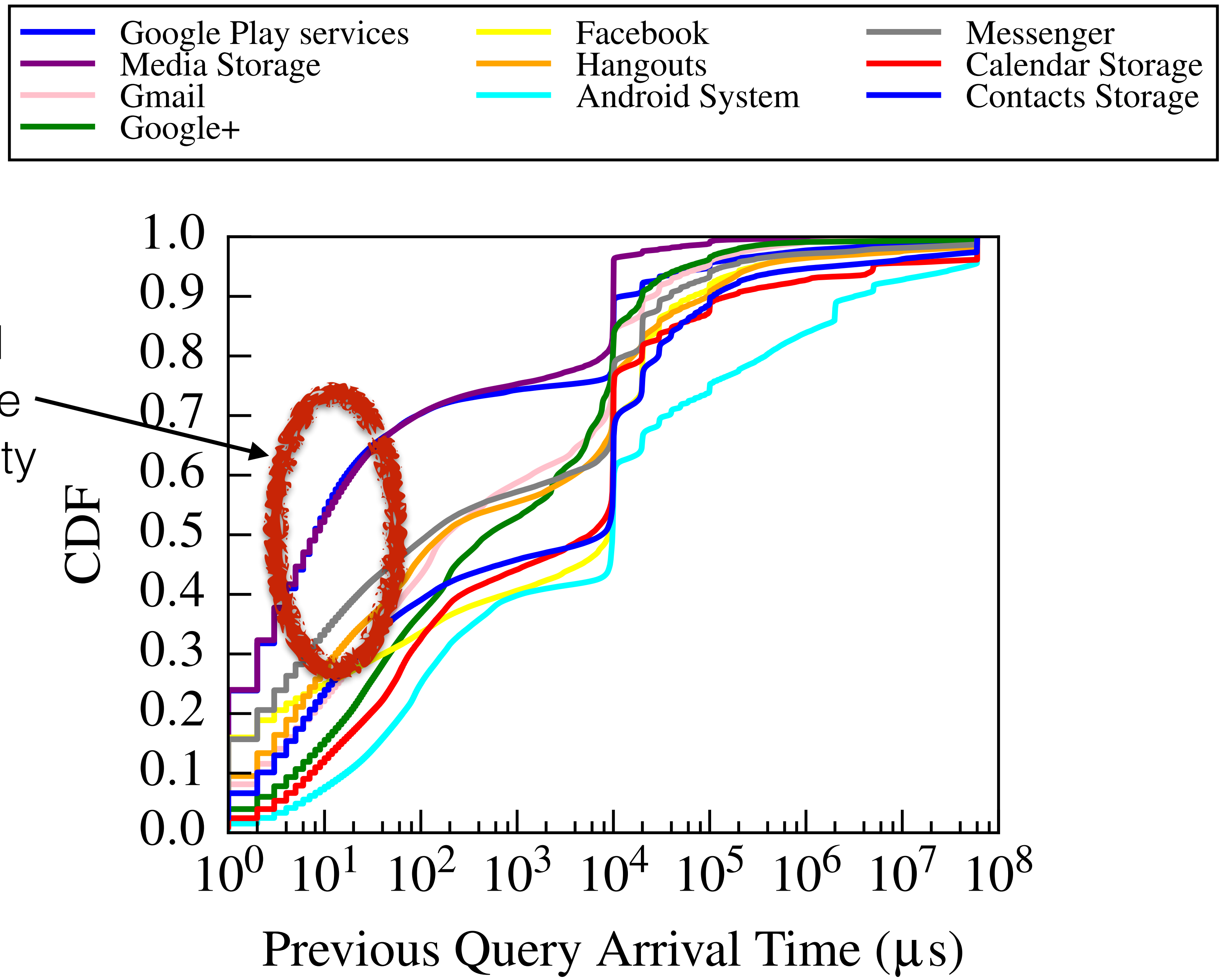
By App



Google Play services
Media Storage
Gmail
Google+
Facebook
Hangouts
Android System
Messenger
Calendar Storage
Contacts Storage
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
CDF
10^0 10^1 10^2 10^3 10^4 10^5 10^6 10^7 10^8
Previous Query Arrival Time (µs)

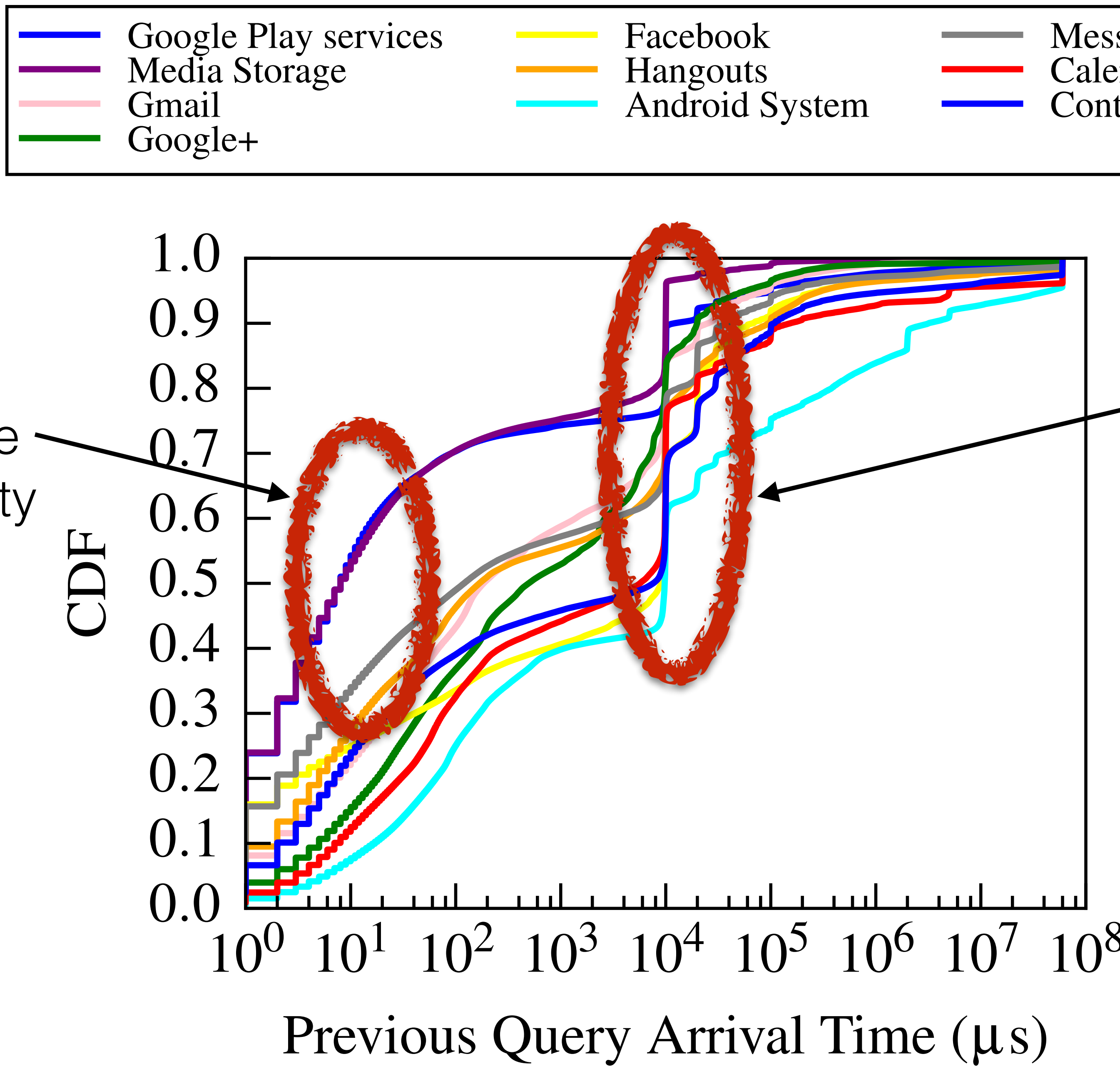
By App

Google Play Services and Media Storage are VERY bursty



By App

Google Play Services and Media Storage are VERY bursty



Same 10 ms periodicity evident across all apps

A Call to Action!

- Mobile phones process ~2 queries/second
  - DB performance important for power, latency, …
- Embedded DBs used differently than Server DBs.
  - We need to understand these access patterns before we can optimize for them.

We need a TPC-MOBILE for pocket-scale data!