Outlook for tomorrow: Cloudy with a chance of PIQLs

Tim Kraska

kraska@cs.berkeley.edu

Storing Data in the Cloud is GREAT

- The Cloud Computing era promises
  - Scalability
  - Fault-tolerance
  - Pay-as-you-go
- All big players and more and more startups have Cloud Storage/DB products

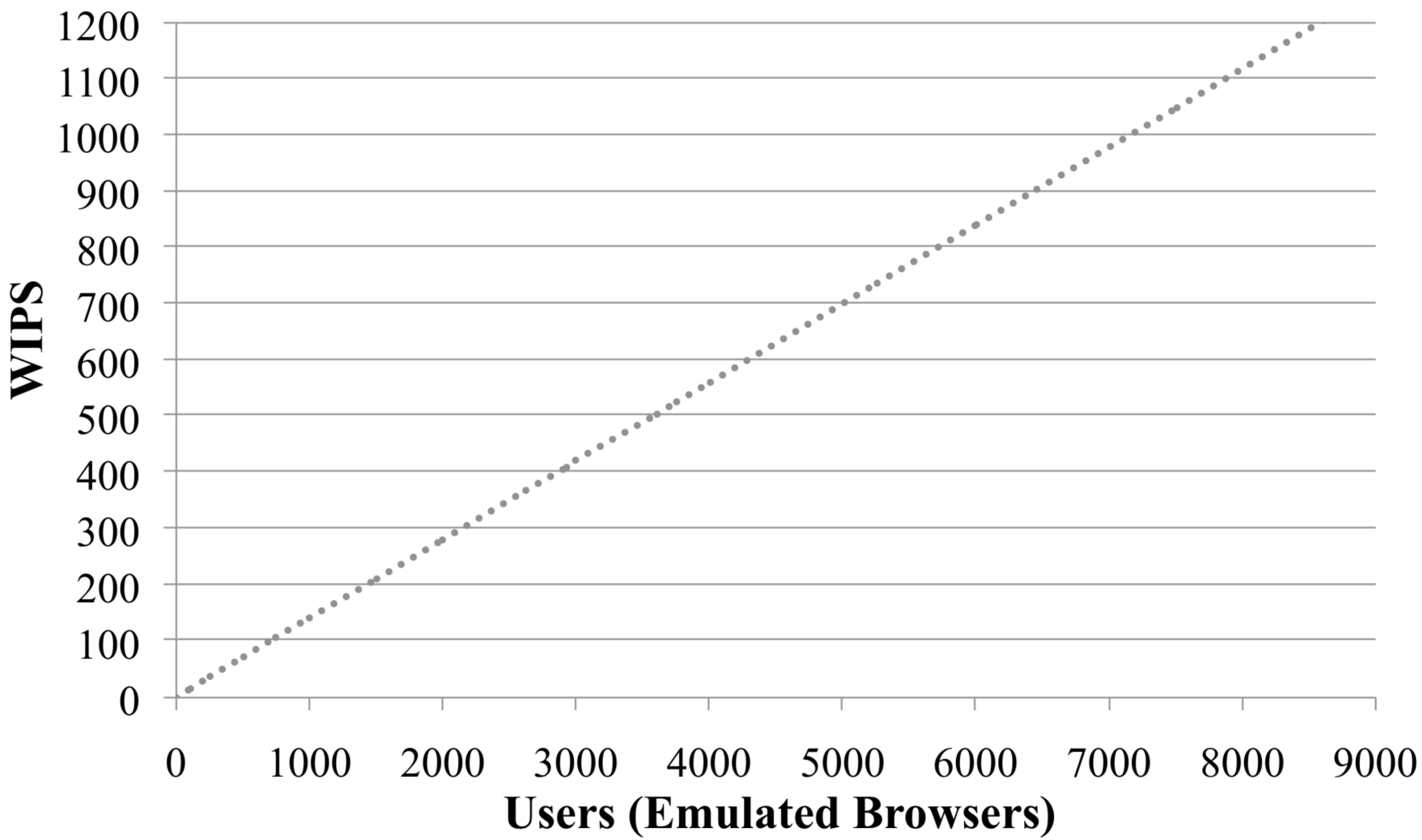
amazon web services™

rackspace®

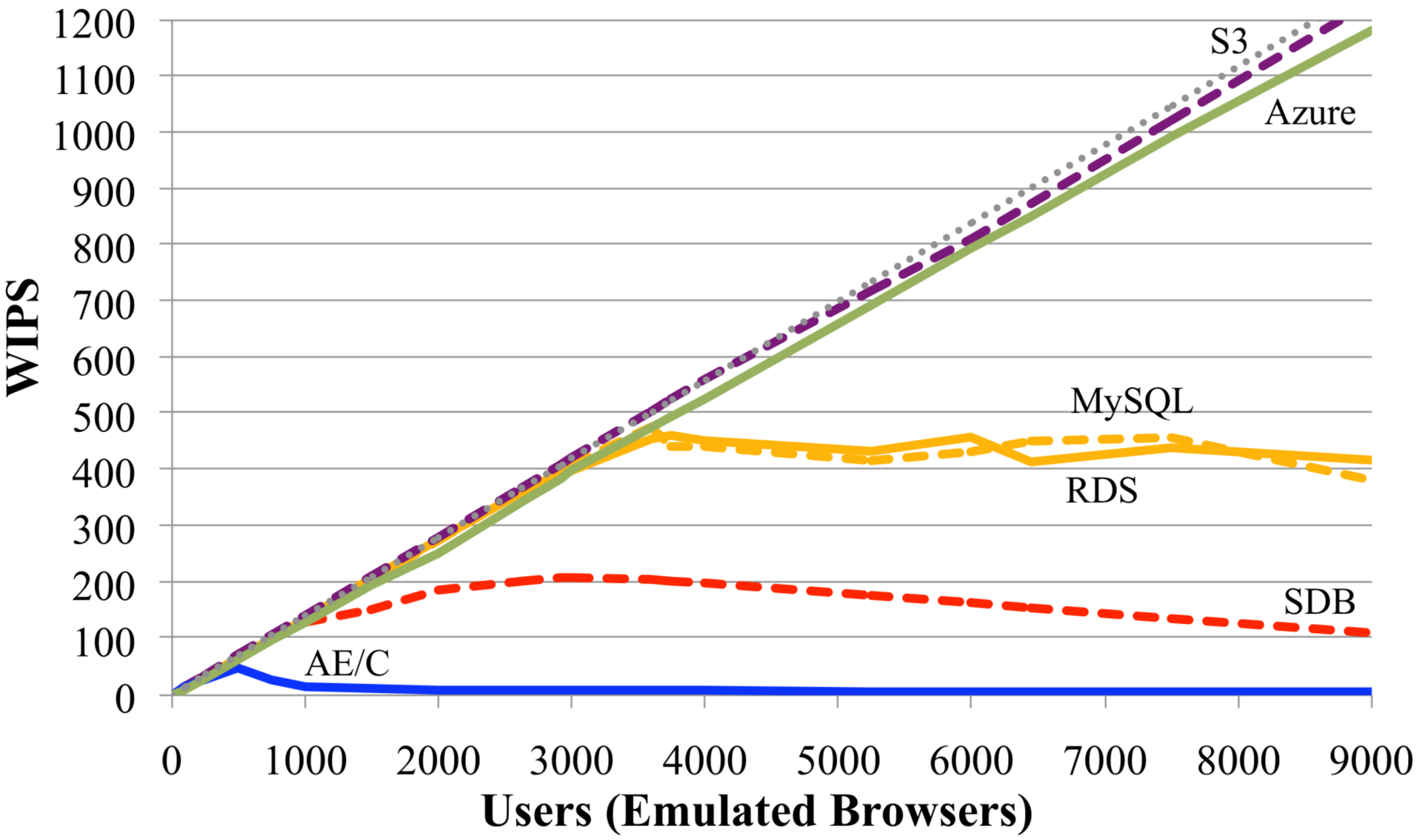
salesforce.com SOFTWARE

Success On Demand.™

Scalability of existing products

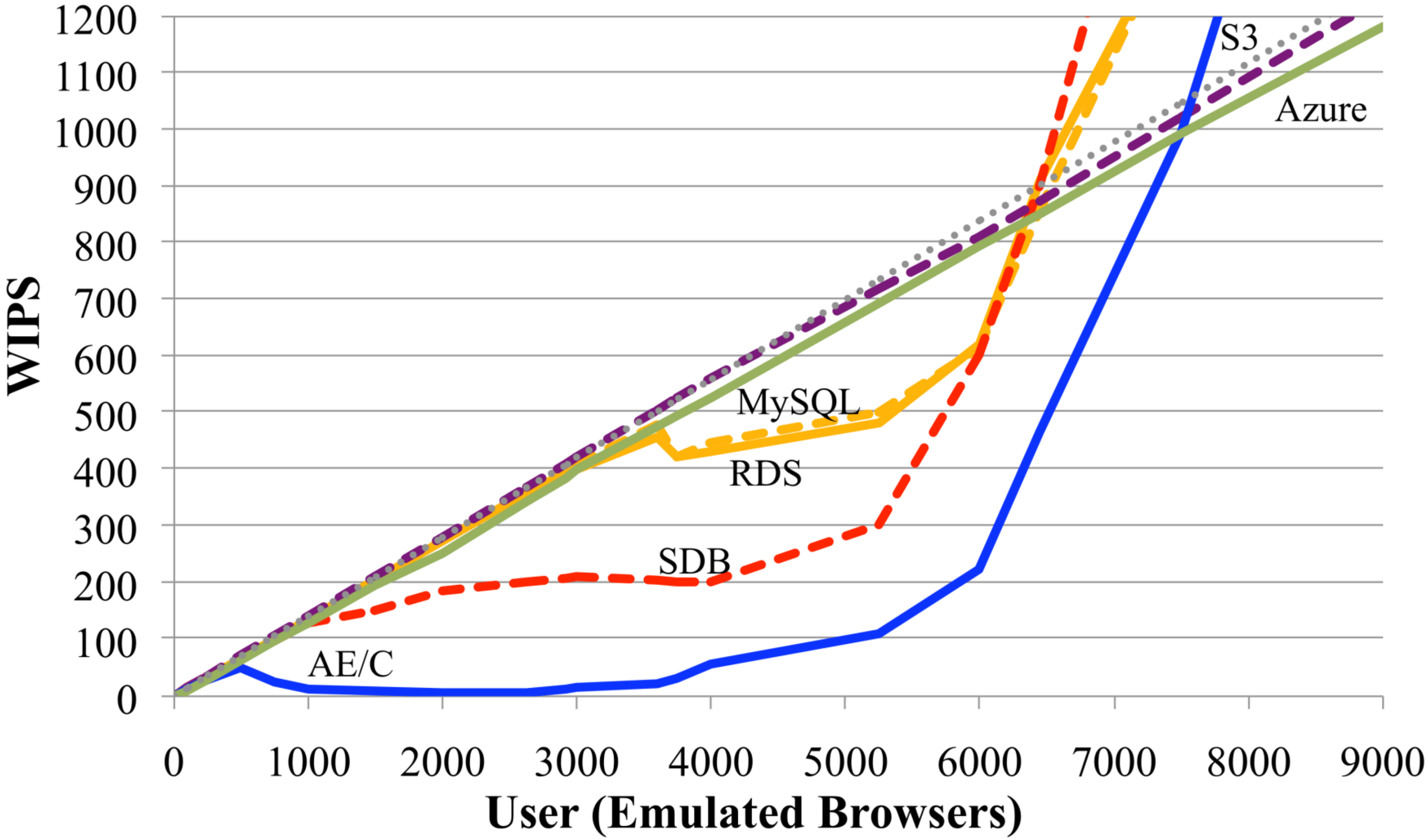


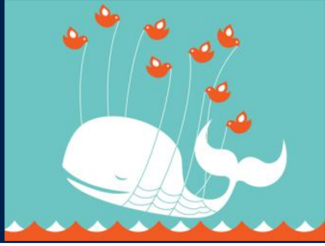
Scalability



[SIGMOD09]

Solution: Sharding???

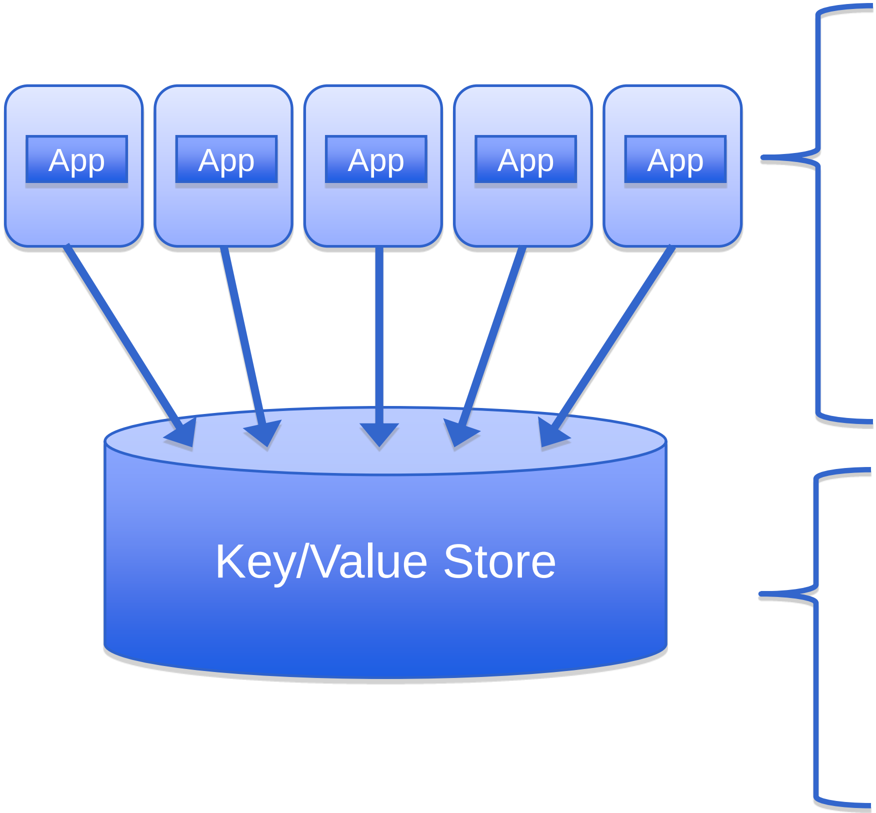




The Success of Key/Value-Stores

Developers find it difficult to write fast/scalable sites using a traditional RDBMS

- Ex: Many of Twitter's "Fail Whales" caused by unintentionally slow DB queries [Chirp 2010]



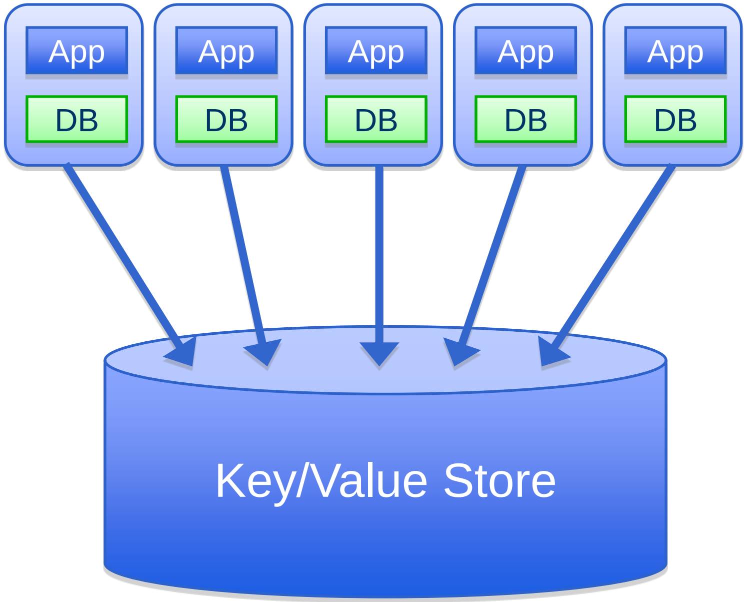
- Stateless
- Easy to scale
- Requires to re-invent DB functionality
  - DB operators (Complex queries are expressed as imperative programs)
  - Consistency
  - Physical (hard-coded) schemas
- Service lock-In

- Simple Query Interface
- Reduced consistency
- Easy to scale (no sharding required)
- Predictable performance
- Easy to price
- High availability (even across data centers)

A Scalable Architecture

Providing a highly scalable DB-Layer on top of the Key/Value store

- Combines database and application Layer
- Scales with the application
- Provides carefully crafted DB-functionality to the developer



- Stateless
- Easy to scale
- DB Library
  - Higher declarative language
  - Consistency a la Card
  - Physical/Logical data independence
  - Independent of Key/Value store

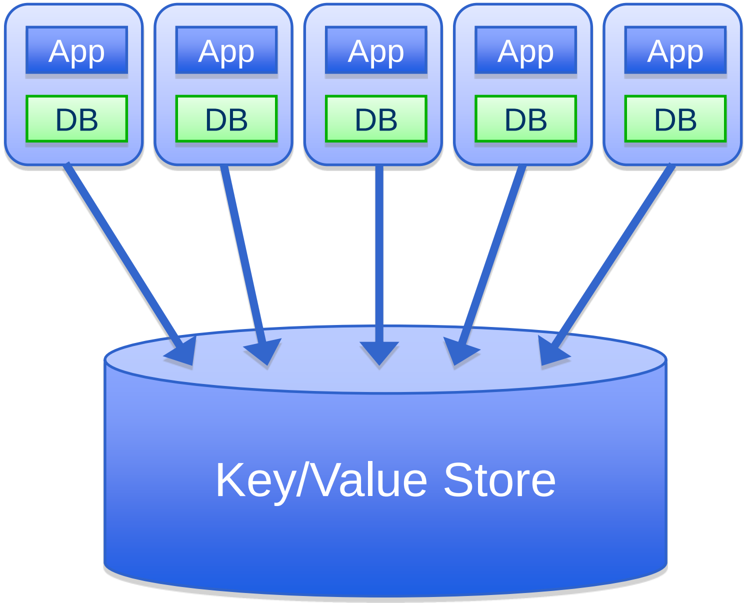
- Simple Query Interface
- Reduced consistency
- Easy to scale (no sharding required)
- Predictable performance
- Easy to price
- High availability (even across data centers)

[Building a DB on S3, SCADS]

A Scalable Architecture

Providing a highly scalable DB-Layer on top of the Key/Value store

- Combines database and application Layer
- Scales with the application
- Provides carefully crafted DB-functionality to the developer



- Stateless
- Easy to scale
- DB Library
  - **Higher declarative language**
  - **Consistency a la Card**
  - Physical/Logical data independence
  - Independent of Key/Value product

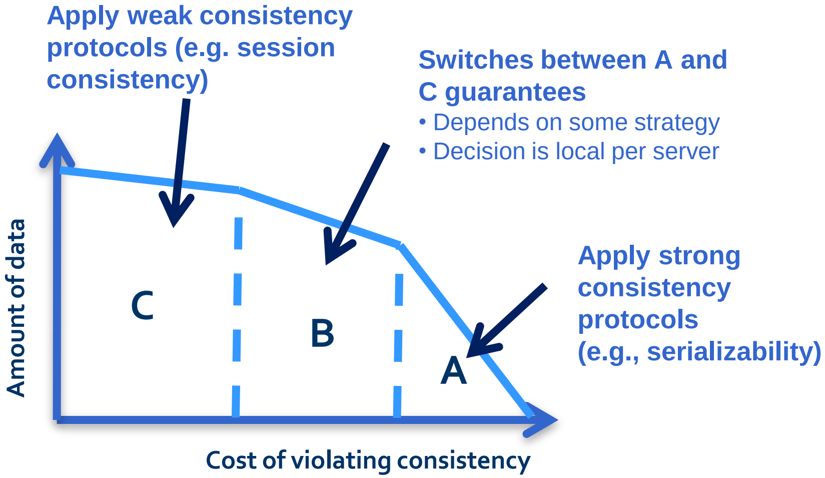
- Simple Query Interface
- Reduced consistency
- Easy to scale (no sharding required)
- Predictable performance
- Easy to price
- High availability (even across data centers)

[Building a DB on S3, SCADS]

Consistency a la Card

Idea: Choose consistency depending on the data requirements

- Logging: Append-Only, no CC needed
- Customer profiles: Single owner, no CC needed
- Product Stock: Commutative updates, CC needed if risking of overselling products
- Ticket Reservation: Commutative updates, CC only needed if close to be sold out
- Bank Transaction: Commutative updates, CC depends on account type
- Access rights at Facebook: Strong CC required, you never want your mother (or your boss) to see your party pictures

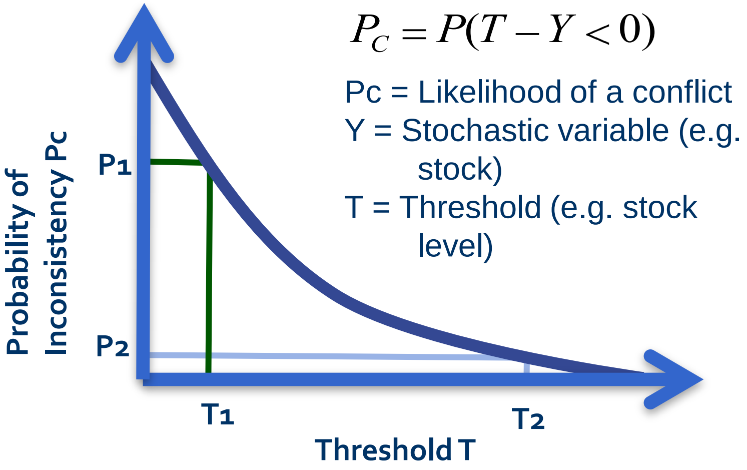


Apply weak consistency protocols (e.g. session consistency)

Switches between A and C guarantees

- Depends on some strategy
- Decision is local per server

Apply strong consistency protocols (e.g., serializability)

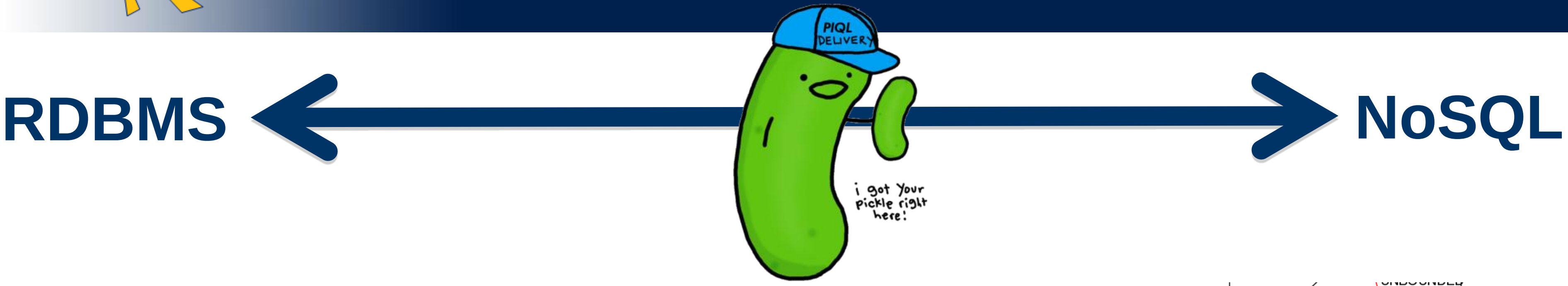


$$P_C = P(T - Y < 0)$$

Pc = Likelihood of a conflict
Y = Stochastic variable (e.g. stock)
T = Threshold (e.g. stock level)

[Consistency Rationing: VLDB09]

PIQL: Know when to say No

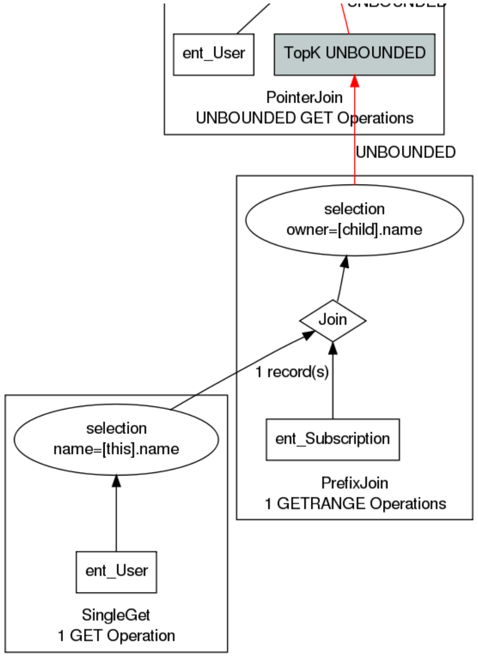


RDBMS

NoSQL

Performance Insightful Query Language

- Scale-independent **declarative language**
  - Only allows developers to write queries with a **data-size independent upper bound**
  - Provides optimization / data independence
- Performance feedback given to developer at compile time
  - Potentially slow queries are prohibited



UNBOUNDED

ent_User

TopK UNBOUNDED

PointerJoin
UNBOUNDED GET Operations

UNBOUNDED

selection
owner=[child].name

Join

1 record(s)

ent_Subscription

PrefixJoin
1 GETRANGE Operations

selection
name=[this].name

ent_User

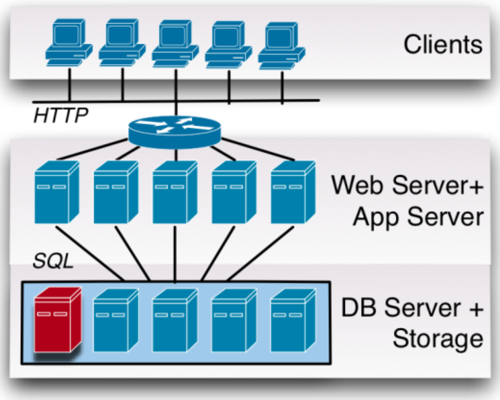
SingleGet
1 GET Operation

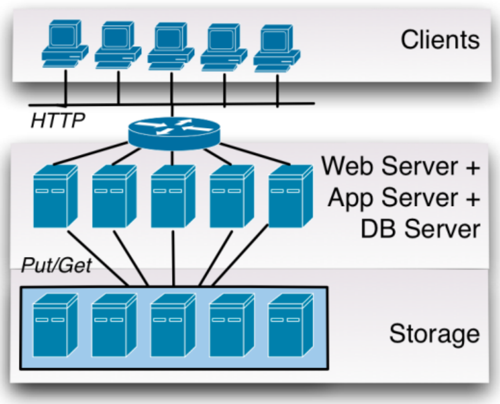
[PIQL: SOCC09 and SIGMOD09]

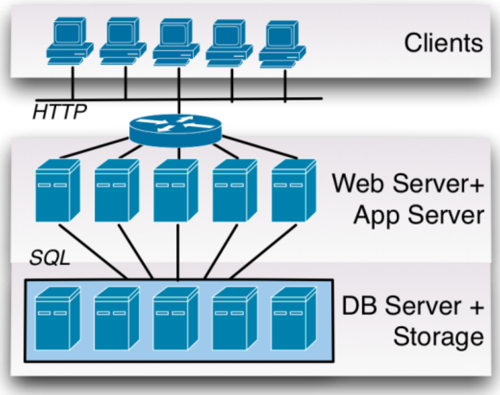
Future Directions

- Challenge 1: Architecture and functionality per layer
- Challenge 2: New/changing workload patterns
- Challenge 3: Data model and language support
- …

Challenge 1: Architecture and functionality per layer







Missing: Reference architecture for Cloud-DB

- What is the right functionality per layer
- How to efficiently push down or up operators between layers
- Support for multi-data centers
  - Increasingly important
  - Helps to increase availability and to decrease response time

[Started to explore with Cloudy]

Challenge 2: Workload Patterns

- Many systems today are over-customized
- More and more highly specific DB systems are built (and only half working)

→ **One size fits not all to the *EXTREME!!!***

→ ***Service Lock-In***

- Workloads change over time
- Instead of predicting HW needs, predict usage needs
- A new system for every new workload?
  - Transaction
  - Analytics/ML
  - Reports
  - Graph Traversal
  - …

→ New (self-made) integration problem

Challenge 3: Data Model and Language Support

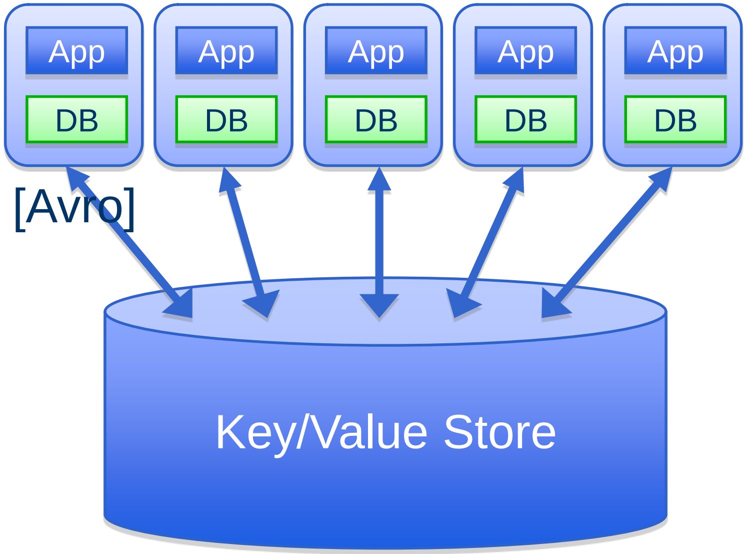
- Data model
  - Document oriented
  - Relational
  - XML
  - Objects
- Language integration (Linq, Ruby,….)
- How to develop with different consistency models

The Resurrection of

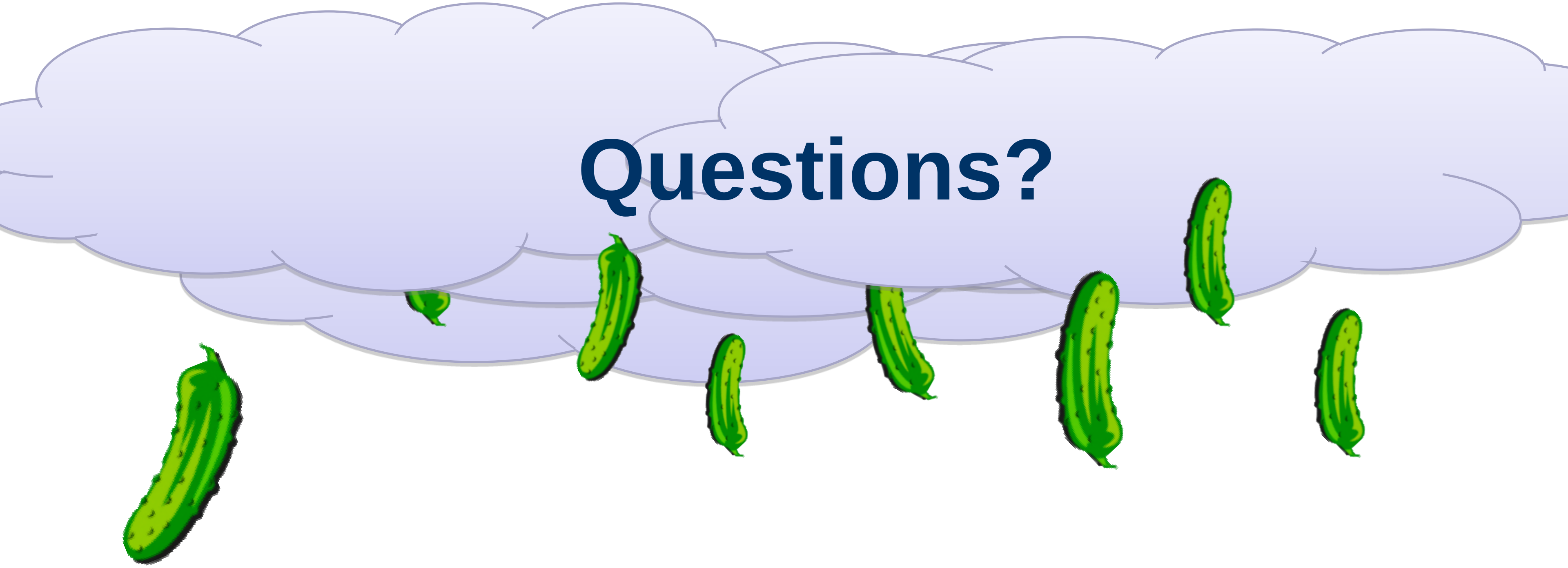
OODBMS?

SCADS Example

```
case class Cust(var name:String, var salary:Int, ….)  with AvroRecord {…}
val ns = cluster.createTable[IntRec, Cust](”test”))
ns.insert(1, new Cust(“Jim”, 10000,…))
val custs= cluster.get(“test”)
val result = custs.map(a => a.name == “Jim” && a.incrSalary(10) > 100)
```



Questions?



Tim Kraska

kraska@cs.berkeley.edu