Java is a trademark of Sun Microsystems, Inc.

Java

JavaOne℠

JDBC – We don't need no stinking JDBC. How LinkedIn uses memcached, a spoonful of SOA, and a sprinkle of SQL to scale.

David Raccah & Dhananjay Ragade

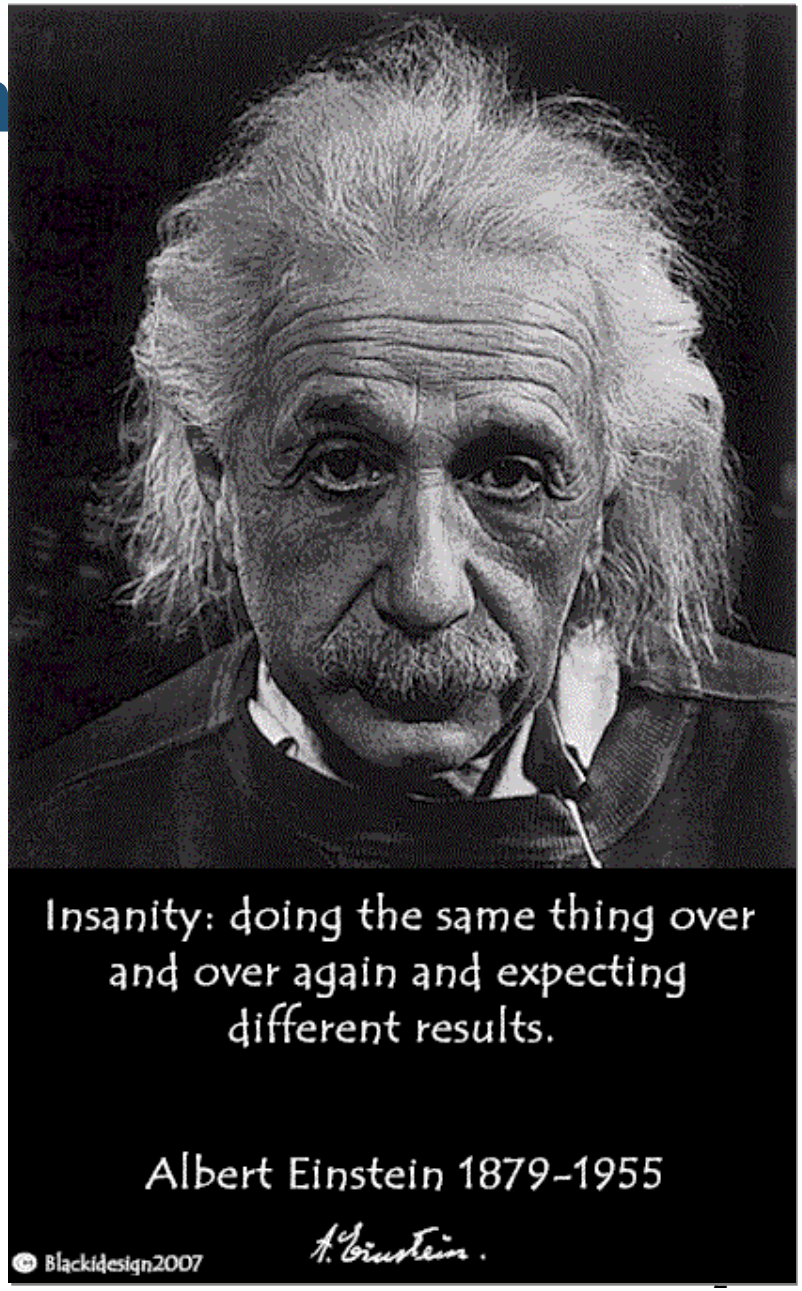
LinkedIn Corporation

LinkedIn

Goal of this Presentation

What you will learn

How LinkedIn built a cheap and scalable system to store our member's profiles, and how you can do the same



Agenda

- **Review system *ilities***
- What happened to databases?
- SOA What
- Discuss existing Best Practices
- Pixie Dust and Kool-Aid are not so bad
- What LinkedIn's got up their sleeve
- How it all came together…
- Q&A

Terminology of the *ilities*
the terms of large successful systems

- Performance
  - Not an “ility” but without it, no ility will save you
- Availability
  - Availability is the proportion of time a system is in a functioning condition
- Reliability
  - The probability that a functional unit will perform its required function for a specified interval under stated conditions.
  - The ability of something to "fail well" (fail without catastrophic consequences)

Terminology of the *ilities*
the terms of large successful systems

- Scalability
  - Slow with multiple users vs. single user
- Manageability
  - The ability to manage all parts of a large moving system
- Serviceability
  - The ability to service an arm of the system without bleeding to death (e.g. change out a database from a working system). Bleeding is OK in a high performance system – death is NOT acceptable.

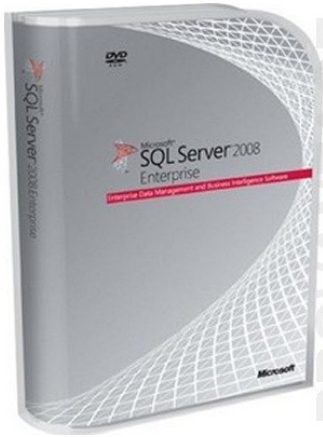
Agenda

- Review system *ilities*
- **What happened to databases?**
- SOA What
- Discuss existing Best Practices
- Pixie Dust and Kool-Aid are not so bad
- What LinkedIn's got up their sleeve
- How it all came together…
- Q&A

Databases

The systems that drive the enterprise … or….

- RDBMS – Relational Data Base Management System Attribute
- KVSS – Key Value Storage System
- Enterprise Search Engines

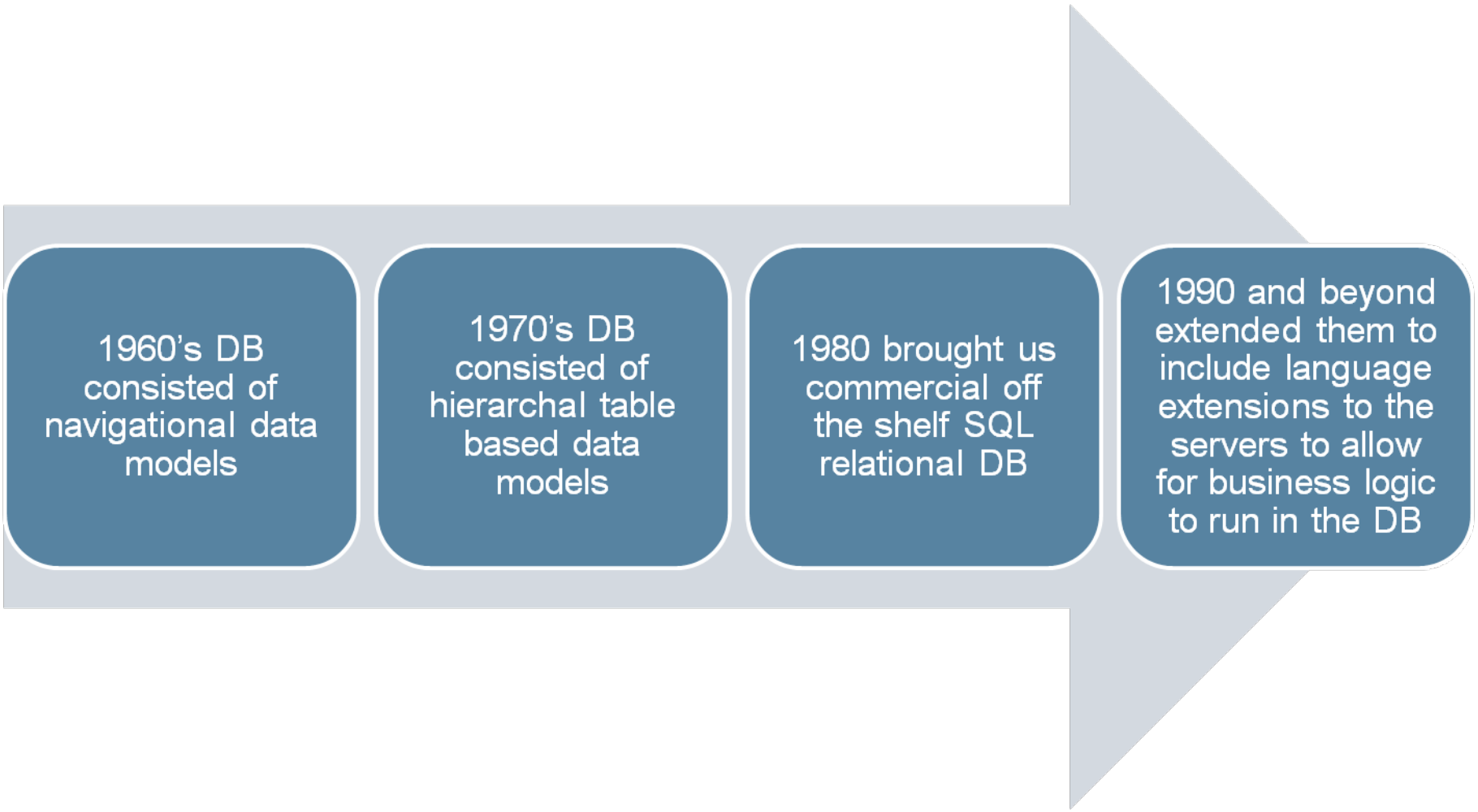


MySQL®

OR XML?

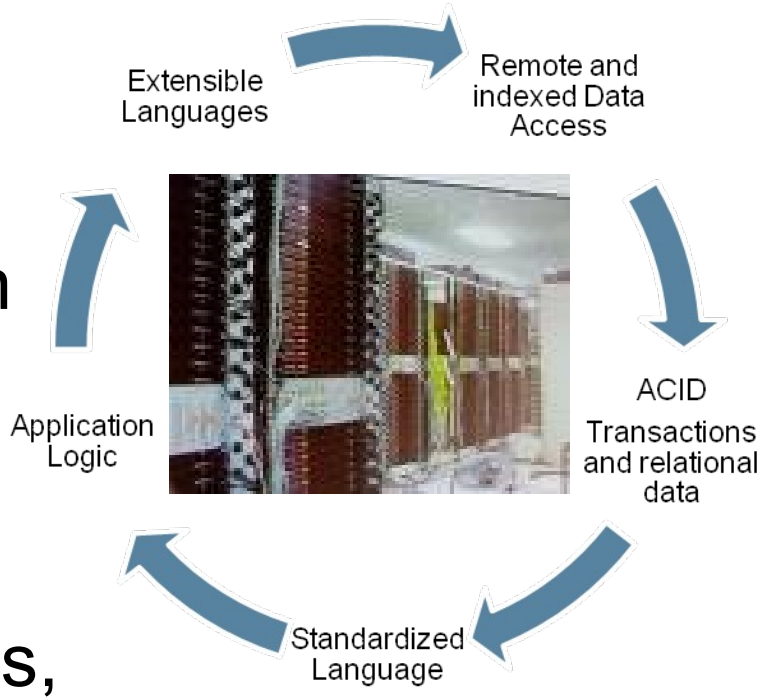
amazon web services™

Database Server History….



Database mind set has changed…
From data access to data management to….

- Initially it was all about remote data access with an index
- Then it moved to ACID data management and tooling
- Then it became an Application Server with data affinity
- Now we have come full circle and people have figured out that scaling is more important than relationships, transactions, and data and behavioral affinity.



Database Mantra that Rule the Roost
ACID

- Atomicity – All or nothing
- Consistency – Data in the system should never get in a contradictory state.
- Isolation: Two requests cannot interfere with one another.
- Durability: No do over – once the data is persisted, it cannot change.

Anti-Database Rules
BASE

- Basically Available
  - Support partial failures within your architecture (e.g. sharding)
- Soft state
  - State may be out of synch for some time
- Eventually consistent
  - Eventually all data is made consistent (as long as the hardware is reliable)

Database Scalability
Or lack thereof…

- Databases work. Look at:
  - Hotmail
  - Facebook
  - eBay
- Databases scale with hardware
- They do not scale horizontally well
  - Partition management is nonexistent and RYO is a mess
  - Many use them as ISAM and not even relational

Database Tools and language
Duh…

- Defacto standards for tools and languages abound for relational databases
- Easy to manage the data within a partition and easy to write code to operate on said data
- Terrifying but nice to use extensions include running Java within the Data Engine, so that you could run your application within the big iron

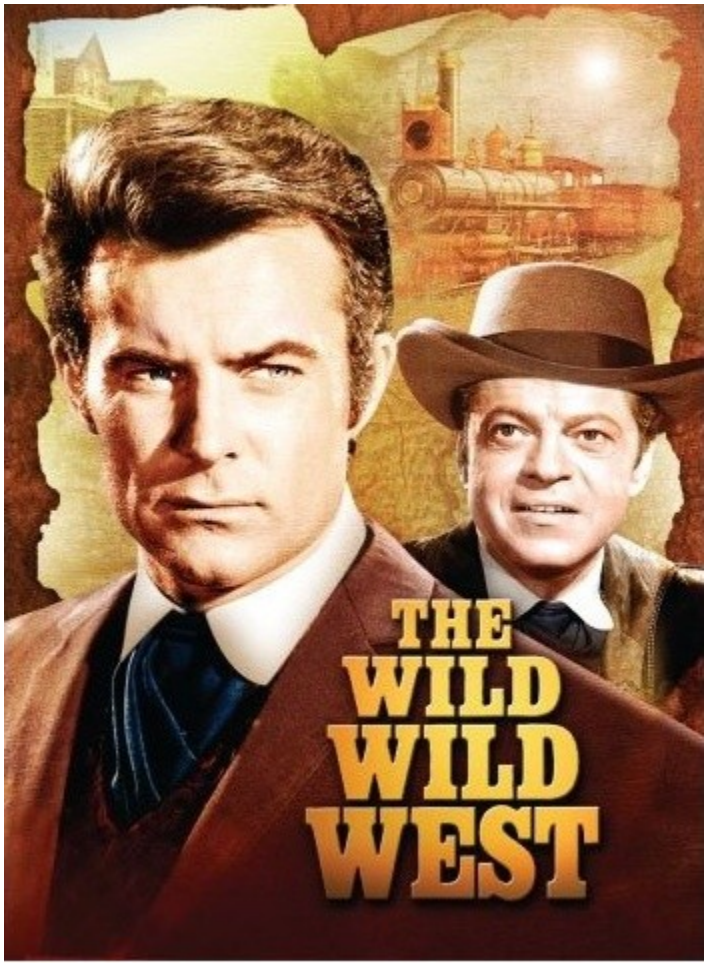
Database's other features

Which are the pain points….

- Constraints – Nice idea until you start partitioning.
  2PC is the anti-scalability pattern (Pat Helland)
- Computation – this feature turns out to cause more pain as cost rises with scale and are incompatible with most languages and tools.
- Replication & backup
  - Nice tools that are indeed important and useful
- ACL support & Data Engine optimizations
  - Used for sure, but exist to circumvent deficiencies

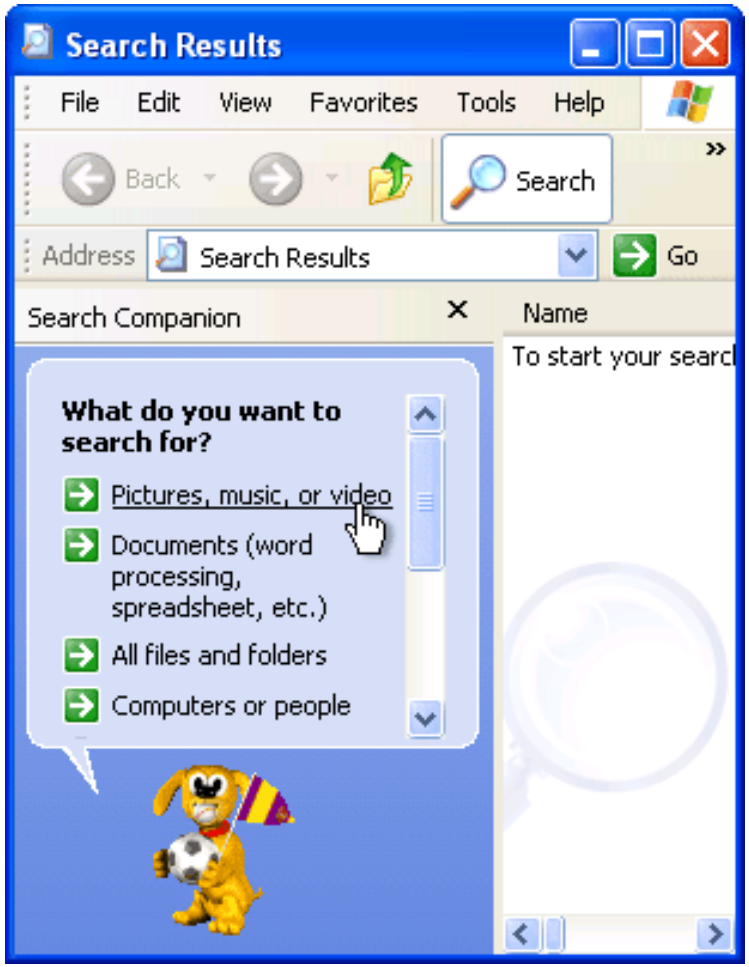
Key Value Storage Systems
BigTable, Hive, Dynamo– the Wild Wild West

- Reliable – Proven on web
- Available – redundant (locally)
- Scalable – no constraints
- Limited ACIDity
- No Standard and not portable
- Almost no:
  - Constraints or relationships
  - Computation or transactions



Enterprise Search Engines
Index yes – storage device no

- A great inverted index
- Finds data quickly
- However, what it returns is commonly an ID to the entity(s) in question
- Real-Time solutions are available but not fully deployed today
- Limited ACIDity/transactions
- Scalable, available, reliable

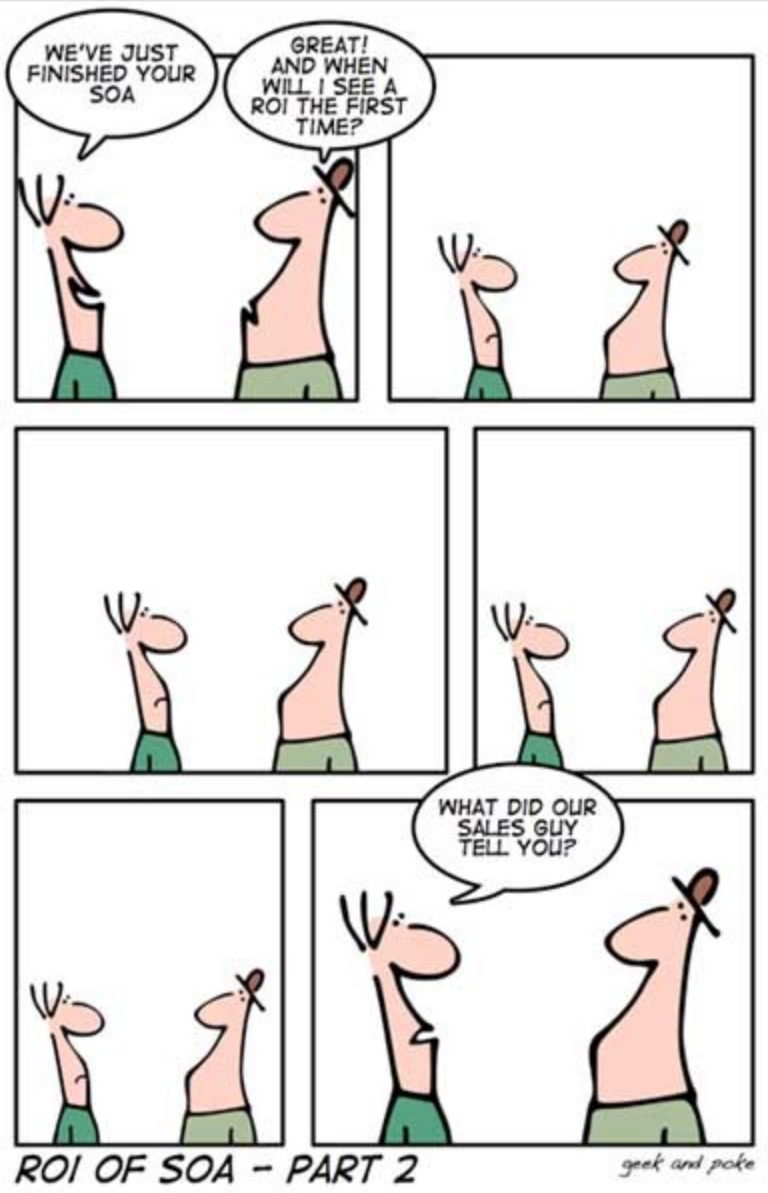


Agenda

- Review system *ilities*
- What happened to databases?
- **SOA What**
- Discuss existing Best Practices
- Pixie Dust and Kool-Aid are not so bad
- What LinkedIn's got up their sleeve
- How it all came together…
- Q&A

SOA
Service Oriented Architecture

- SOA may be overkill for most enterprises
- Still a Tiered and layered architecture – which is what SOA hoped to formulate and standardize is a solid approach
- Services (not SOA) allow for efficient reuse of business processes and aggregation services within a complex development organization



Agenda

- Review system *ilities*
- What happened to databases?
- SOA What
- **Discuss existing Best Practices**
- Pixie Dust and Kool-Aid are not so bad
- What LinkedIn's got up their sleeve
- How it all came together…
- Q&A

Best Practices
Storage and architecture

- Store critical data redundantly and reliably with a cluster
  - Google via BigTable, Facebook via MySQL, eBay via replicated & sharded DB
- Layer services on top of the storage device to manage data integrity and complexity
  - LinkedIn, Amazon, eBay

Best Practices

Storage and architecture

- Create a bus to route replicated data to consumers – e.g. search, data mining, etc.
  - Almost all sites
- Parallelization via things like scatter/gather
  - Almost all search topologies (Google, Yahoo, Live),
  - Facebook, etc.

Best Practices
Storage and architecture

- Keep the system stateless
  - eBay, Google, etc.
- Partition data and services
  - Facebook, eBay
- Cache data
- Replicate your data
- Route requests to where the behavior and/or data exists
- Degrade gracefully with load

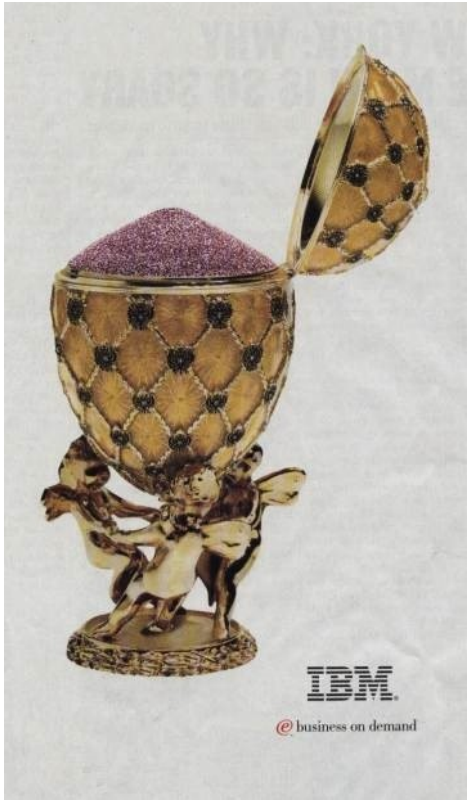
Best Practices
Storage and architecture

- Tiering systems
  - Latency vs. Affinity
    - Traversal versus affinity – you need to analyze the cost and make a decision
  - Scaling vs. parallelizing
    - Do you need to keep tiering all systems to keep the scalability uniform?
  - Complexity vs. diminished dependencies
    - Does the reduced dependencies make up for the increased system complexity?

Agenda

- Review system *ilities*
- What happened to databases?
- SOA What
- Discuss existing Best Practices
- **Pixie Dust and Kool-Aid are not so bad**
- What LinkedIn's got up their sleeve
- How it all came together…
- Q&A

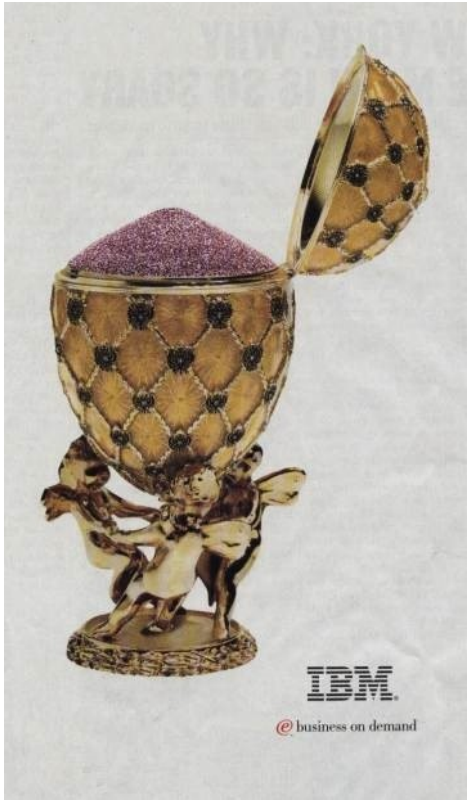
Pixie Dust and Kool-Aid
Building on the past



Pixie Dust and Kool-Aid

Building on the past

- So what do we want:
  - Reliable
  - Available
  - Scalable
  - ACIDity on simple transactions
  - Standard and portable interface
  - Data Optimizations
  - Cache and replicate
  - Low cost BASE architecture



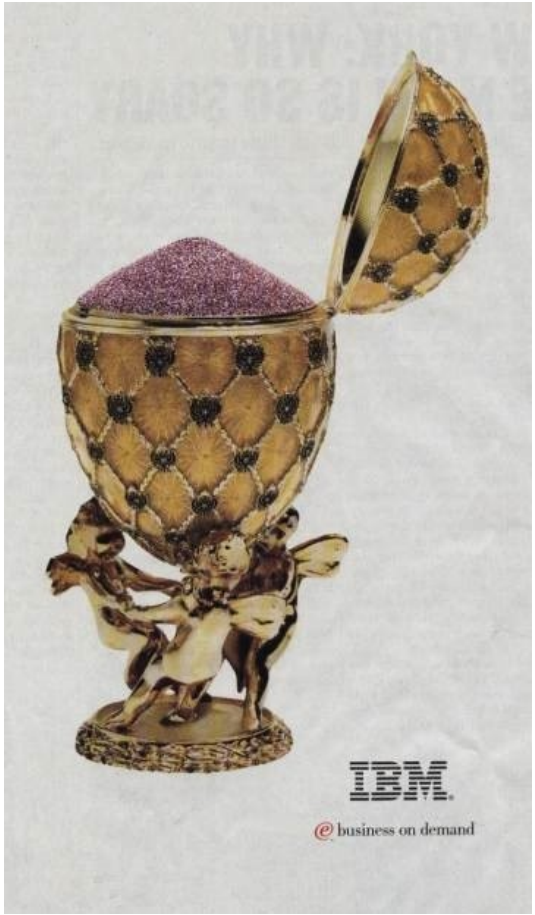
Agenda

- Review system *ilities*
- What happened to databases?
- SOA What
- Discuss existing Best Practices
- Pixie Dust and Kool-Aid are not so bad
- **What LinkedIn's got up their sleeve**
- How it all came together…
- Q&A

LinkedIn's Data Services

Mixture of standards and pixie dust

- Front a database with a service
- Cache data
- Route to and partition the data service
- Scale and replicate services in a horizontal manner
- Keep all writes ACID and subsequent reads ACID as well



LinkedIn's Data Services

Mixture of standards and pixie dust

- Databases are reliable
- Scale out at the service
- Replicate and cache
- Partitioning comes from the front tier and business servers that front the data services

LinkedIn's Data Services

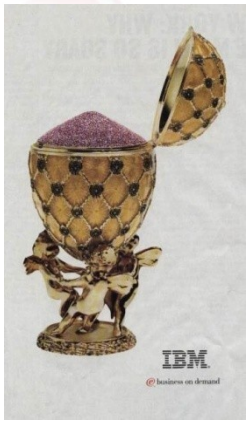
Immediate replication vs. eventual replication

- Caching needs a consistency algorithm
- Techniques for immediate replication
  - Paxos
    - Chubby, Microsoft AutoPilot, Zoo Keeper
  - N Phase Commit (2PC and 3PC)
- Techniques for eventual consistency
  - BASE (Basically Available, Soft-state, Eventual Consistency
    - Inktomi, Dynamo, AWS

LinkedIn's Data Services

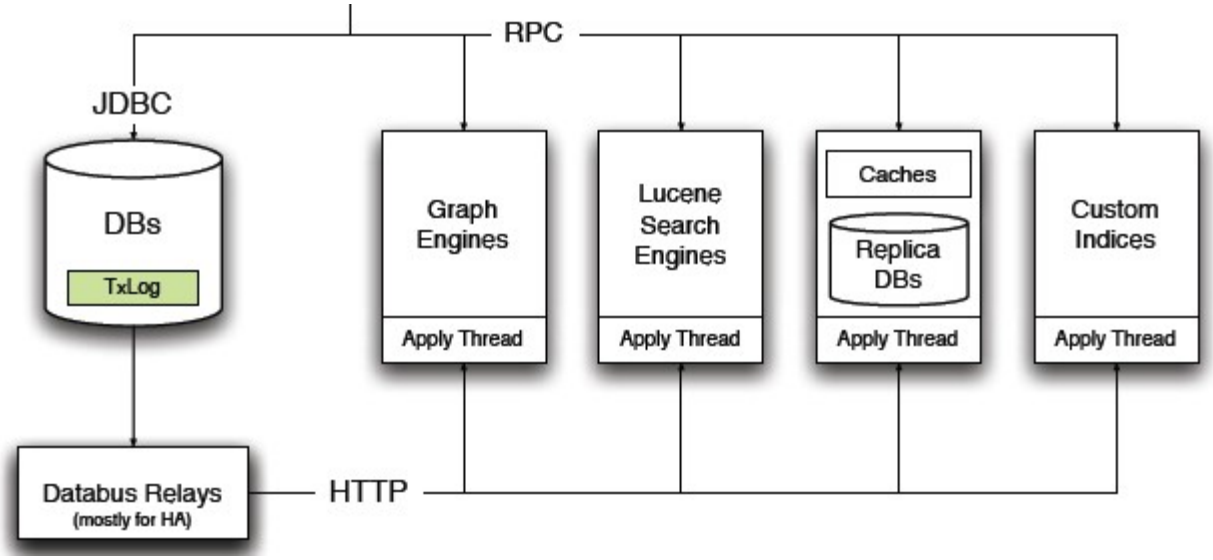
LinkedIn's approach

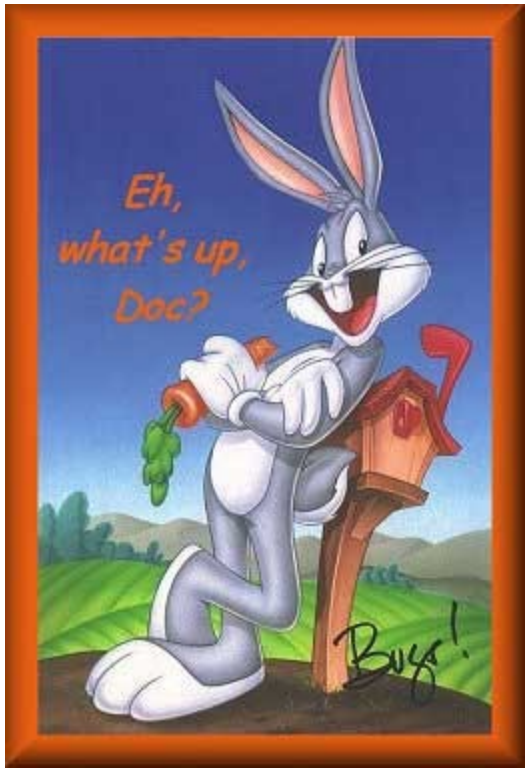
- Keep core data ACID
- Keep replicated and cached data BASE
- Replicate data via the data bus
- Cache data on a cheap memory (memcached)
- Use a hint to route the client to his / her's ACID data



LinkedIn's Data Services

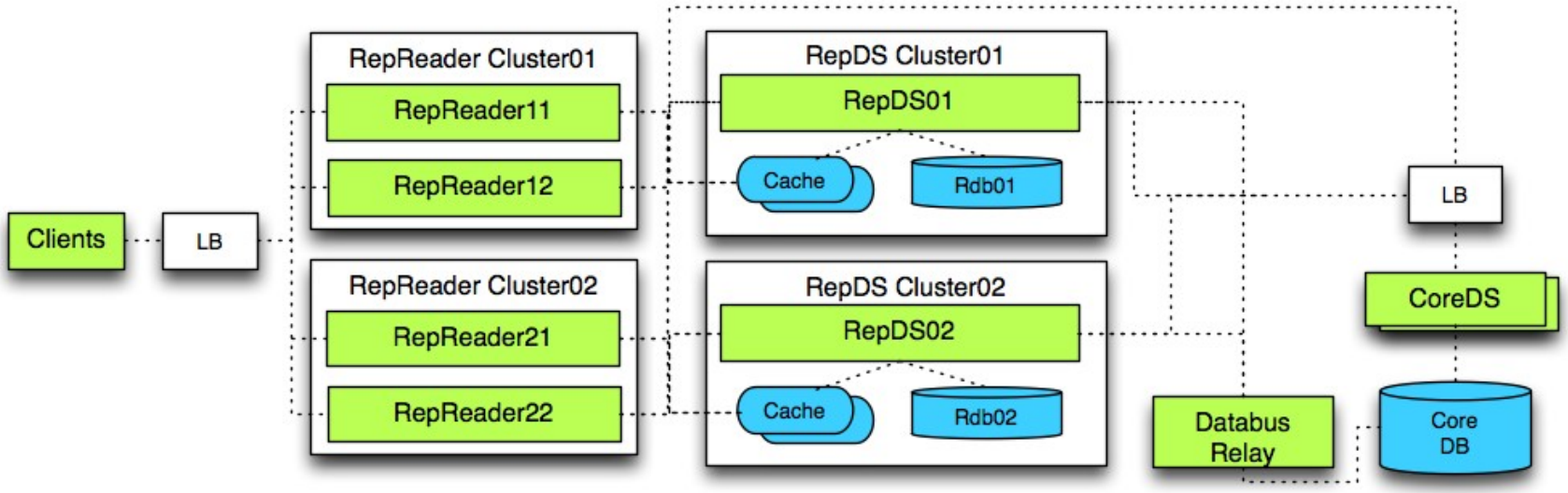
Databus – the linchpin of our replication





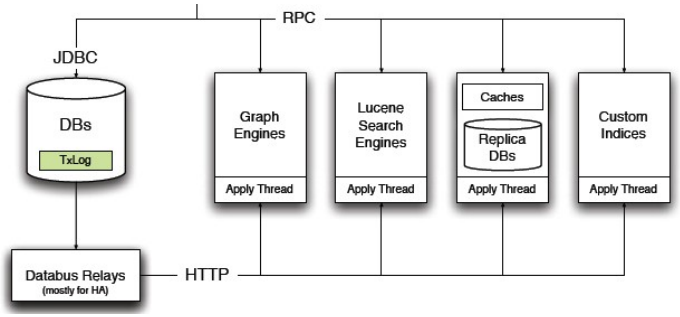
LinkedIn's Data Services

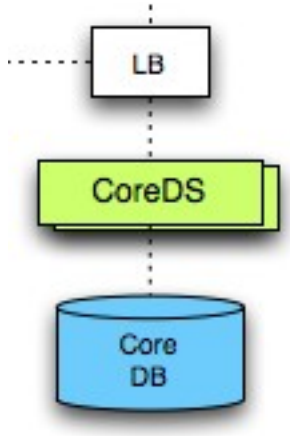
LinkedIn's approach



LinkedIn's Data Services
Core DS

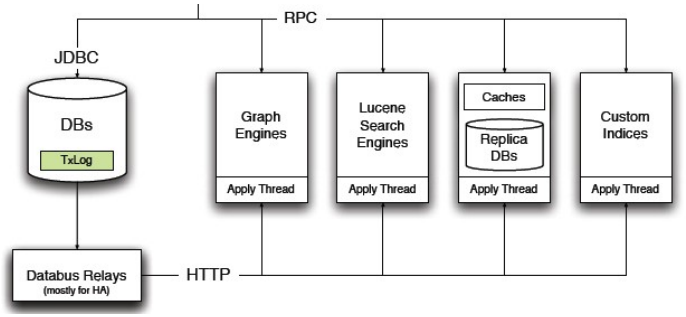
- Keep core data ACID in the DB
- All writes come here.
- Databus source for all replication
- The last line of defense for a cache miss
- Also manages sharding and partitioning

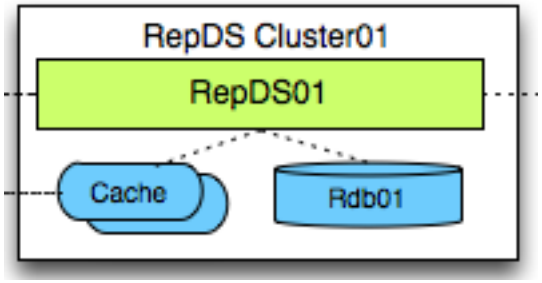




LinkedIn's Data Services
RepDS

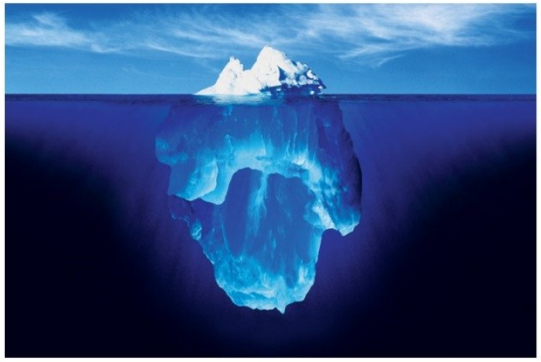
- Manages cache consistency and replication
- Manages the freshness of the caller
- Reads come from cache

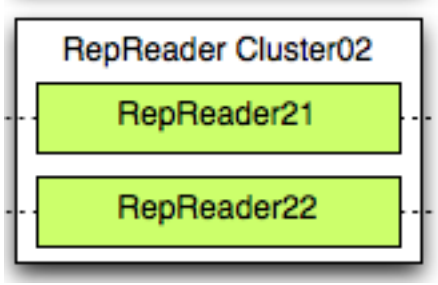


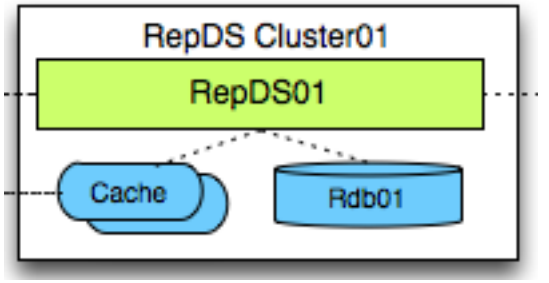


LinkedIn's Data Services
RepReader

- RepReader is the typical tip of the iceberg problem
- All read operations are sourced from the cache unless the caller's freshness token is out of the window

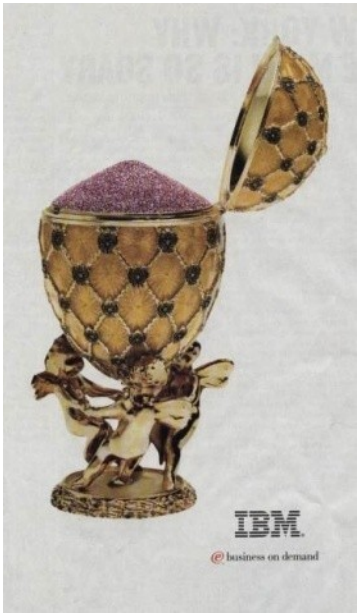


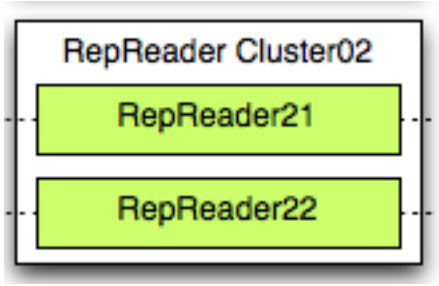




LinkedIn's Data Services
Freshness Token (AKA Pixie Dust)

- The freshness token = Pixie Dust for CUD operations
- It also allows us to give the caller control over whether they are content with BASE data, even if they did no CUD operation.

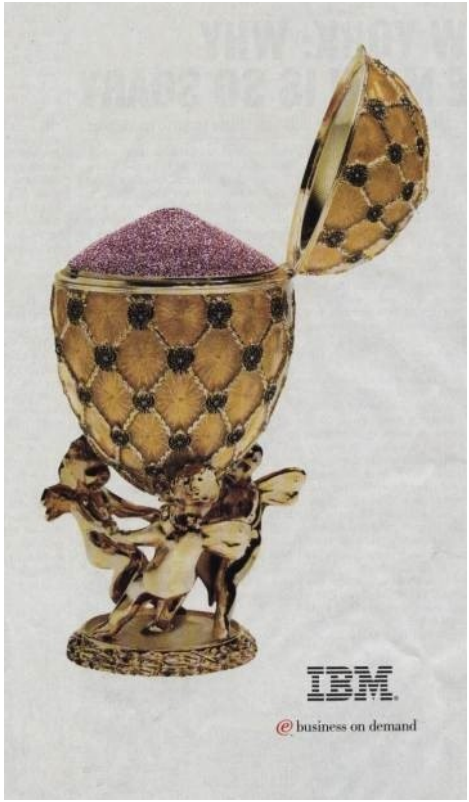




LinkedIn's Data Services

For the love of Pixie dust and Kool-Aid

- We use commodity hardware and software to run our service
- We use Pixie Dust to keep costs down and keep our customer happy
- We keep OPS and the exec-staff happy with our special brand of Kool-Aid



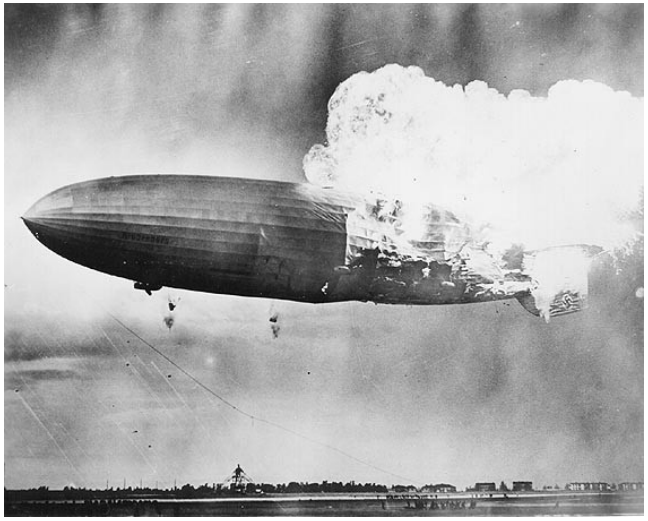
Agenda

- Review system *ilities*
- What happened to databases?
- SOA What
- Discuss existing Best Practices
- Pixie Dust and Kool-Aid are not so bad
- What LinkedIn's got up their sleeve
- **How it all came together…**
- Q&A

Profile Re-architecture

Changing planes in mid-flight

- Original LinkedIn System
- Use of XML for i18n
- Phased Transition



Problems from the original system
Anthropology 101

- Be fair… it worked well for a startup
- Many tables in one big DB
- Too many similar object hierarchies
- No well defined domains

Why XML?
Flexibility

- Profile has many fields
- 1NF for I18n ==> too many tables
- StAX for fast parsing
- Easier to version the profile
- Human readable
- JSON? ProtoBuf?

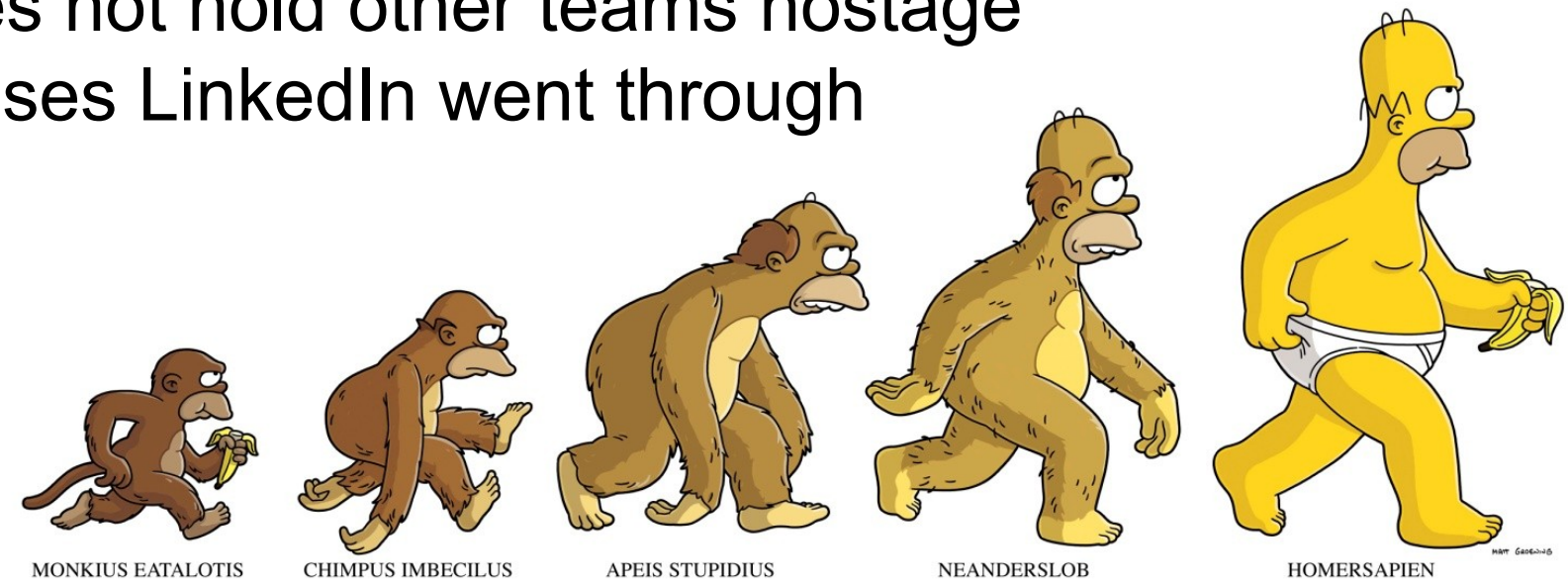
Issues with XML <good/> <bad/> <ugly/>

- XML schema design tradeoffs and analytics impact
- XML is verbose
- StAX is unfriendly
- XML in the DB caused us some performance headaches

Phased Transition

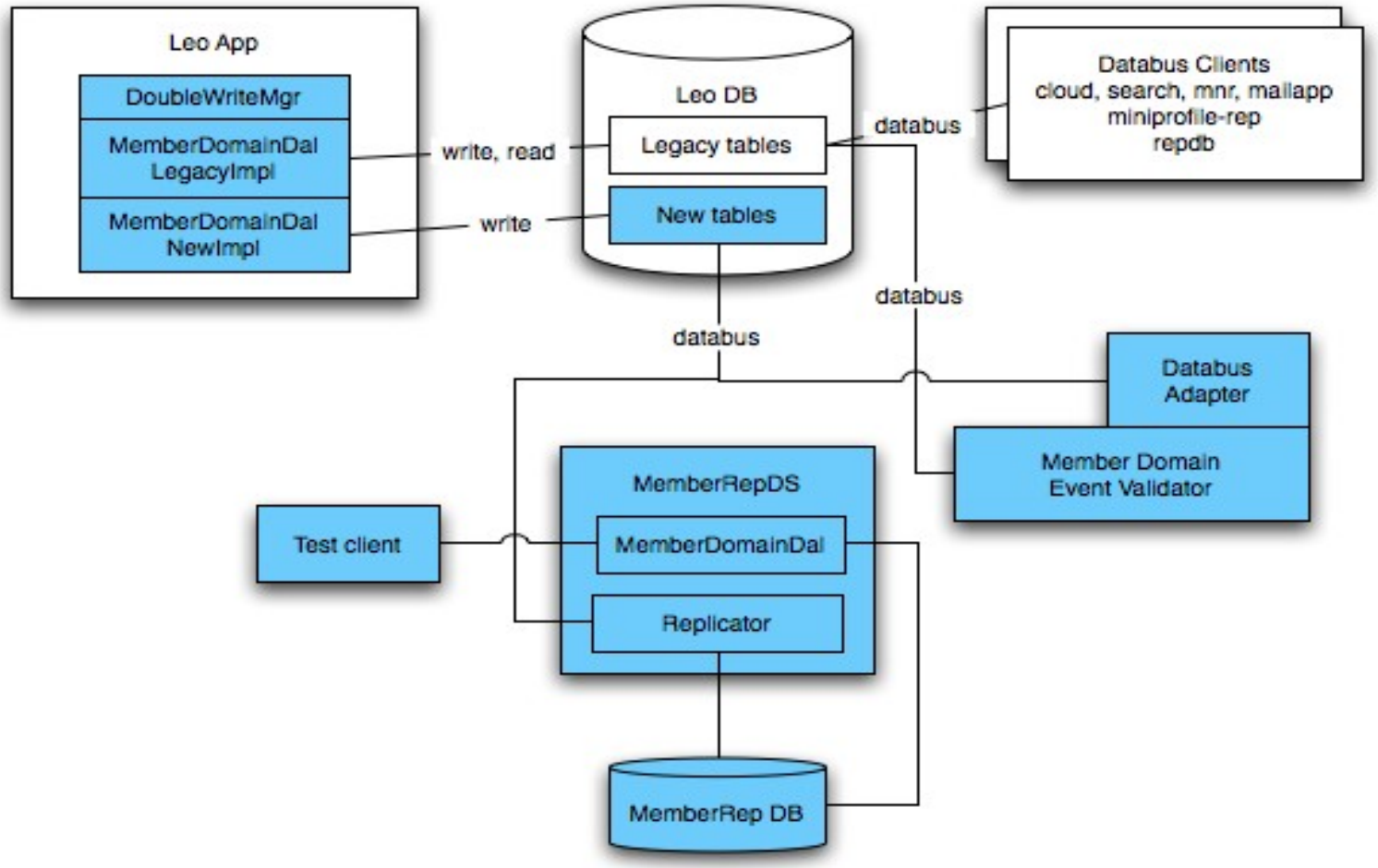
Evolving a living, breathing organism

- Successive iterations avoid breakages
- No major site downtime
- Easier to sanity check
- Does not hold other teams hostage
- Phases LinkedIn went through



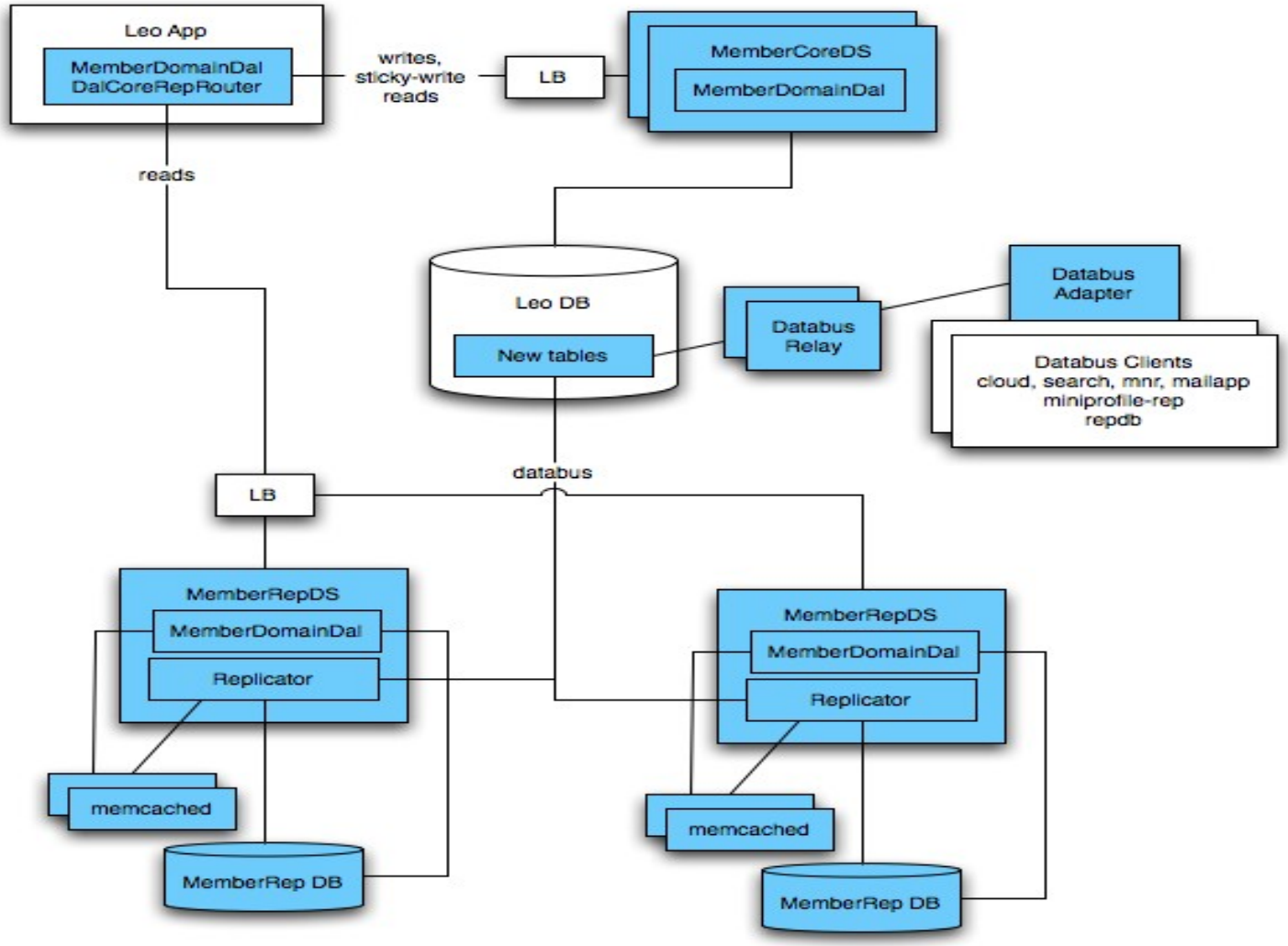
HOMERSAPIEN

Double Writes Topology
Safety first



After Legacy Tables Dropped

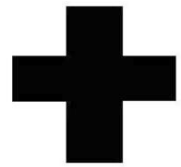
Auld Lang Syne

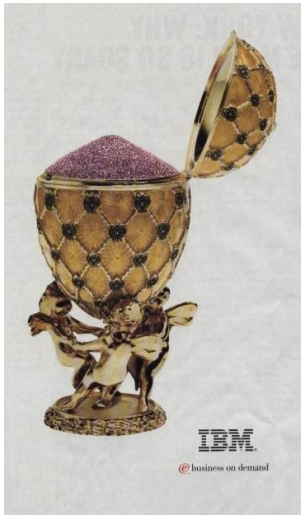


Wrap up

The moral of the story is…

- Keep your system BASE
- Use commodity hardware
- Use pixie dust (AKA data freshness token)
- Evolve slowly - no big bang!





Q&A

Thank You

David Raccah & Dhananjay Ragade
draccah@linkedin.com
dragade@linkedin.com

Linkedin Corporation

Appendix

Performance
Often mixed up with scalability

- Performance
  - A numerical value given to a single system when asked to do a task under nominal load
  - If the system responds poorly without load, it will assuredly continue its molasses response time under load

Availability
Often mixed up with reliability

- Availability
  - A numerical value given to a system that defines the proportion of time a system is in a functioning condition.
  - Most common scoring system is called nines – which is defined as the uptime versus the uptime and downtime – five nines = 0.99999

Reliability

The ability for a system to perform its functionality

- Reliability
  - A system can be 100% available and still be 100% unreliable (e.g. non consistent caching)
  - A person can consistently give you the wrong answer
  - Architecture is defined as the balance of the ilities and cost

Scalability

the term that many think is the holy grail

- Scalability
  - The ability for a system to manage more traffic or to be “scaled” as more traffic appears
  - System slows with multiple users vs. single user
  - Route, Partition, Orchestrate, replicate, and go asynch
  - Split the system horizontally
  - Rarely scale vertically

The rest of the *ilities*

the ones that people tend to ignore till its too late

- Manageability
  - It is a double-edged sword which can be easily ignored
- Serviceability
  - Here complexity starts to rear its ugly head
- Maintainability
  - Of course maintainability tends to run upstream of complexity