ebay inc
GLOBAL PLATFORM
& INFRASTRUCTURE

# Deep dive - Oracle sharding at eBay Inc.

John Kanagaraj, Data Architect, eBay Inc.

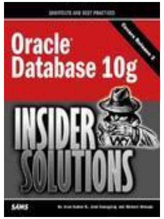
November 2014

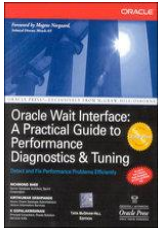
# Agenda

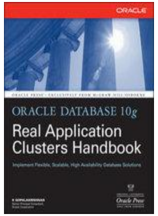
- eBay Scale – A quick glance
- Problems at scale
- The eBay Approach
- Q & A

# Speaker Qualifications

- Data Architect / Database Engineer at eBay Inc.
- Worked with Oracle Databases and UNIX for too long ☺
- Frequent speaker, Author and Technical editor
- Started working with NoSQL recently
- [http://www.linkedin.com/in/johnkanagaraj](http://www.linkedin.com/in/johnkanagaraj)
- See my “Oracle vs NoSQL” slide deck on LinkedIn







SELECT JOURNAL

ORACLE ACE

Offline

Online

| Volume | Velocity | Variety |
|---|---|---|
| 5-15 Years of History<br>4PB Largest Table<br>16M Analytic Queries<br>14K Users<br>4M Batch Queries<br>900K Ad Hoc Queries | 37PB Read<br>3PB Write<br>16+TB/day Semi-Structured Data<br>36 TB/hour x-Platform Data Transfers | 3.5PB+ Structured Data<br>10PB Semi-Structured Data (80% compressed)<br>10K+ Name/Value Pairs |
| 6PB Consumed<br>2TB Daily Average<br>700M Active Items<br>300M Active Site Users<br>8K Average Application Connections/DB | 200B+ eBay Queries/day<br>4K eBay Batch Runs/day<br>25GB/sec Peak Site Traffic | 800+ Oracle Instances<br>300+ MongoDB Nodes<br>300+ MySql Nodes<br>200+ Cassandra Nodes |

# Architectural Forces at Internet Scale

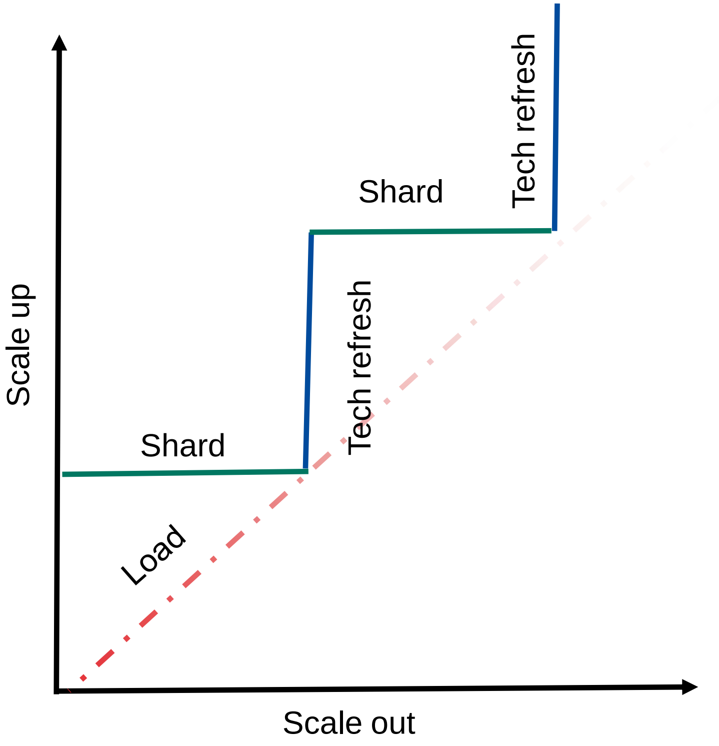
- Scalability driven by unpredictability
  - Capacity needs to increase linearly with load: usually not the case
  - Designing for 5-10x growth in data, traffic, users, etc. costs $$$$$$
- Availability: True 24x7x365
  - Resilience to failure (MTBF)
  - Rapid recoverability from failure (MTTR)
  - Graceful degradation and appropriate timeouts
- Latency
  - User experience latency, esp. with Multi Data Center
  - Data latency and the CAP Theorem in play
- Manageability
  - Simplicity leads to Maintainability and better MTTR
  - In-depth diagnostics at all layers and levels
- Cost
  - Development effort and complexity
  - Operational cost (TCO)

# Best Practices for Scaling

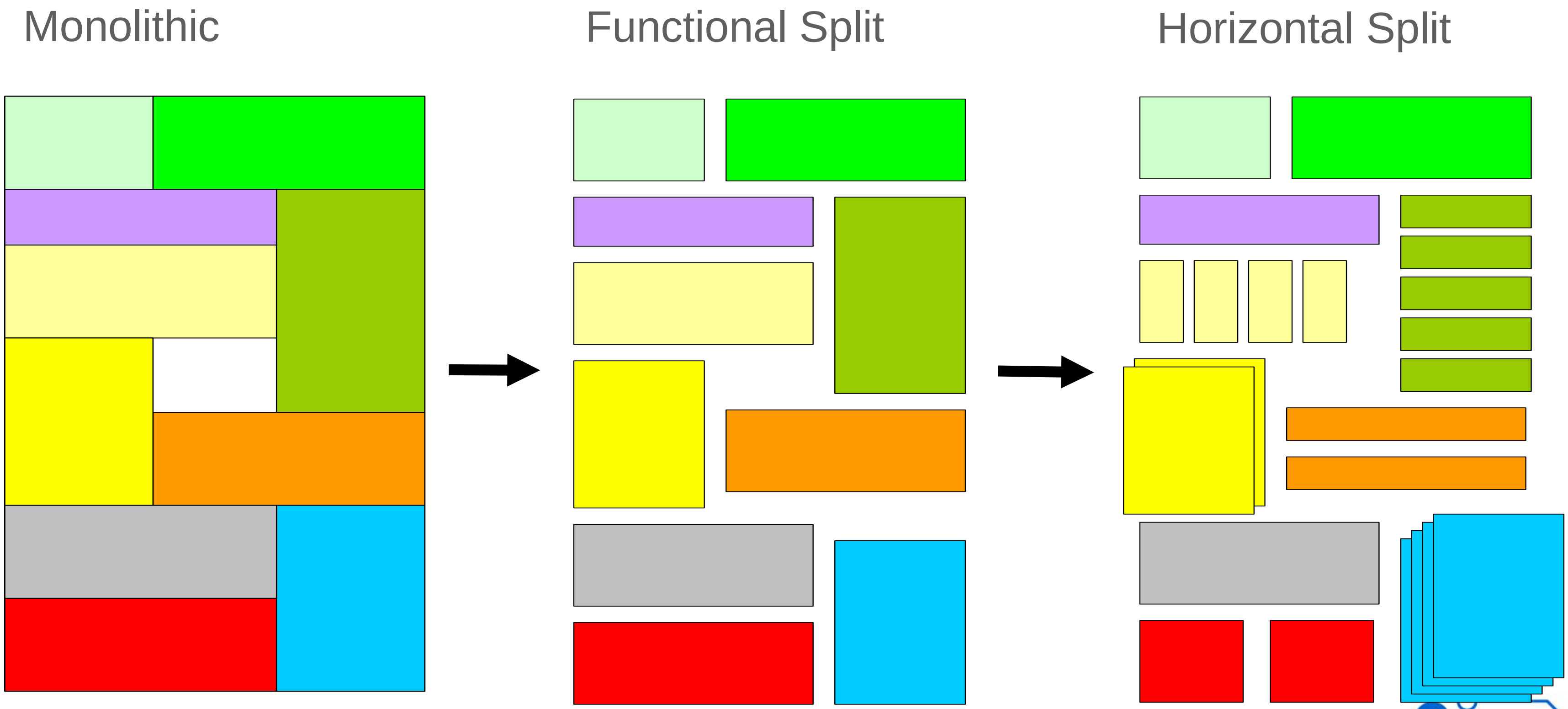
- Partition Everything (Functional/Horizontal)
- Asynch as much as possible
- Automate Everything
- Plan for Failure
- Expect (and cater for) Inconsistency

# Scaling Patterns

- Split every problem into manageable chunks
  - By data, load, and/or usage pattern
  - Repeat after me: *If you can’t split it, you can’t scale it*
- Motivations for splitting
  - Scalability: Horizontally and independently
  - Availability:  Isolate failures to specific segments
  - Manageability:  Decouple different segments and functional areas
  - Cost:  can use less expensive hardware, both for deployments and tech refreshes
  - Survivability: Test out new hardware/upgrades, etc. in “small batches”



# Database splits



Monolithic

Functional Split

Horizontal Split

# Partitioning Everything

## *Pattern:  Functional Segmentation*

- – Segment processing into pools, services, and stages
- – Segment data along usage boundaries (beware: PL/SQL and Transactions!)

## *Pattern:  Horizontal Split*

- – Load-balance processing
  - Within a pool, all servers are created equal
- – Split (or "*shard*") data along primary access path
  - Partition by range, modulo of a key, lookup, etc.

## *Corollary:  No Session State and Relationships**

- – User session flow moves through multiple application pools
- – Absolutely no session state in application tier
- – Breaks relationships (joins, enforcement of foreign keys)

# Partition Everything:  Databases

- *Pattern:  Functional Segmentation*
  - Segment databases into functional areas
  - Group data using standard data modeling techniques
    - Cardinality (1:1, 1:N, M:N)
    - Data relationships
    - Usage characteristics
  - Logical hosts
    - Abstract application’s logical representation from host’s physical location
    - Support colocating and separating hosts without code change

*Over 1000+ logical databases on ~400 physical hosts*

# Partition Everything:  Databases

- *Pattern:  Horizontal Split*
  - Split (or *“shard”*) databases horizontally along primary access path
  - Different sharding strategies for different use cases
    - Deterministic function on key
    - Lookup- or range-based
    - Aligned on stronger key
  - Aggregation / routing in eBay’s custom built Data Access Layer (DAL)
    - Abstracts developers from split logic, logical-physical mapping
    - Routes CRUD operation(s) to appropriate shard(s)
    - Supports rebalancing and read routing through configuration change

# Partition Everything: Application Tier

- *Pattern: Functional Segmentation*
  - Segment functions into separate application pools
  - Minimizes DB / resource dependencies, esp. number of connections!
  - Allows for parallel development, deployment, and monitoring
- *Pattern: Horizontal Split*
  - Within pool, all application servers are created equal
  - Routing through standard load-balancers
  - Allows for rolling updates
- Needed when dealing with thousands of App servers in hundreds of pools

# Why Oracle?

- Store and persist data - That’s what databases are for !!
- Relational Database - Tables, Columns, Constraints… (within database)
- ACID properties – Sets Oracle Apart from NoSQL…
  - Atomic
  - Consistent
  - Isolation
  - Durable
- Interface to access and manipulate data via SQL (which many NoSQL's lack…)
- High performance and scalability Within certain well-known boundaries…
- Backup, Redundancy and Data Movement DR/Active Standbys, GoldenGate,…
- Procedural options Only when necessary

# Data modeling: Core strength and necessity!

- 400+ Site Data Models
- > 200 Logical host families
- > 1000 expanded logical hosts
- ~ 75,000 columns (excluding 3rd-parties)
  - Mostly NUMBER or VARCHAR2, some DATE and few LOB
- ~ 2000 sequences
- Very small number of Views, Triggers, and Stored Procedures
  - Vastly reduced PL/SQL dependency
  - Always keep functional segmentation in mind!

# What’s in a Logical Host?

- eBay construct denoting a data source
- Maps to one or more physical databases
- Enables data source abstraction for code
- Structural unit for availability and scalability
- Groups objects along functions and products
- Groups database objects in the data models
  - One logical host = One data model
  - 400 + and counting

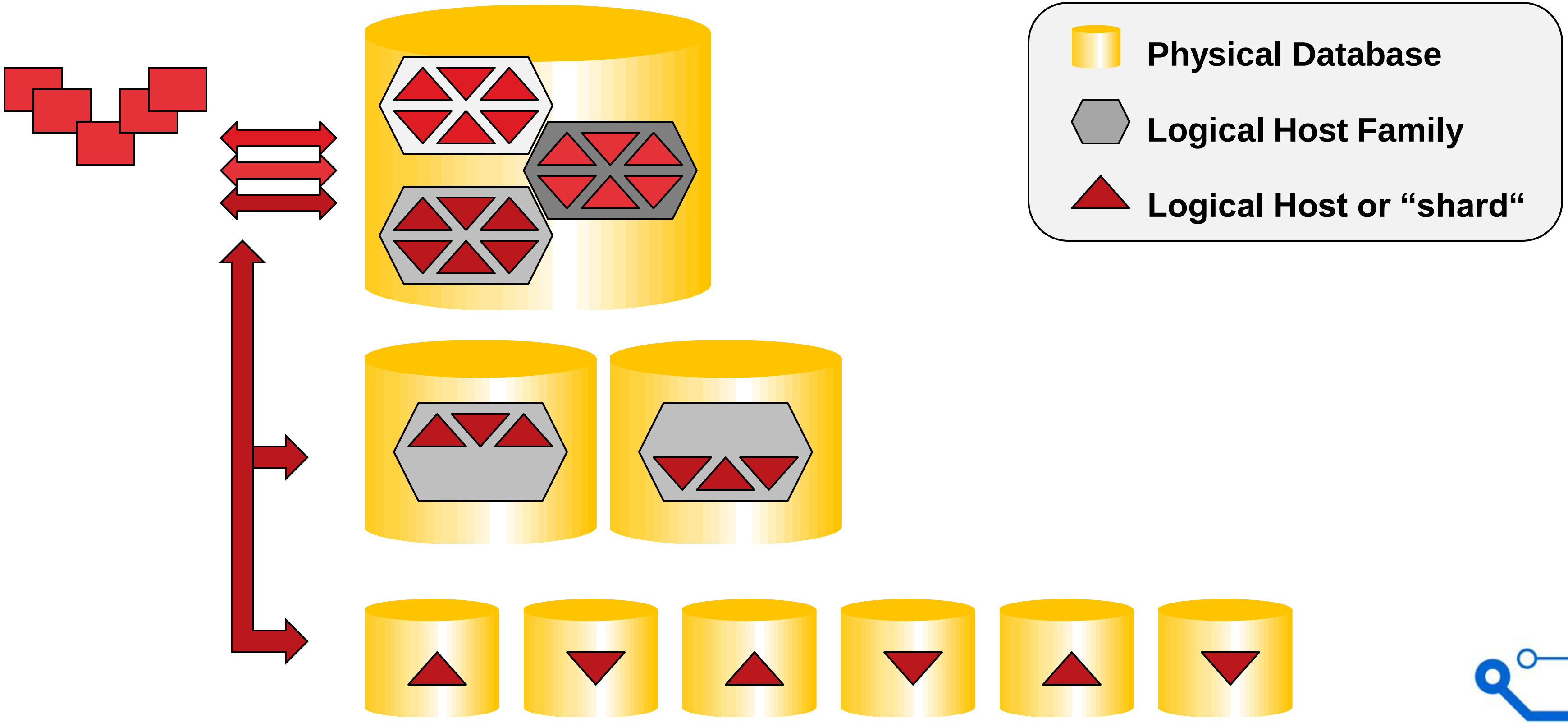
# eBay Sharding Patterns

- Deterministic Function
- Lookup based
- Aligned Host

# Deterministic Function

- Location is based on a deterministic function of a primary key or “hint” from a previous interaction
- Sequence-controlled PK ensures global uniqueness and host number fidelity
- eBay’s custom Data Access Layer (DAL) understands/supports this
- Pros:
  - Simple (single key access pattern)
  - No performance overhead
- Cons:
  - Physical scale-out’s upper-bound limited by the function’s boundaries
  - Re-sharding requires code changes and data re-organization
  - Records cannot be relocated since PK decides shard location

# Logical to Physical: Scale-Out Scenario



Physical Database

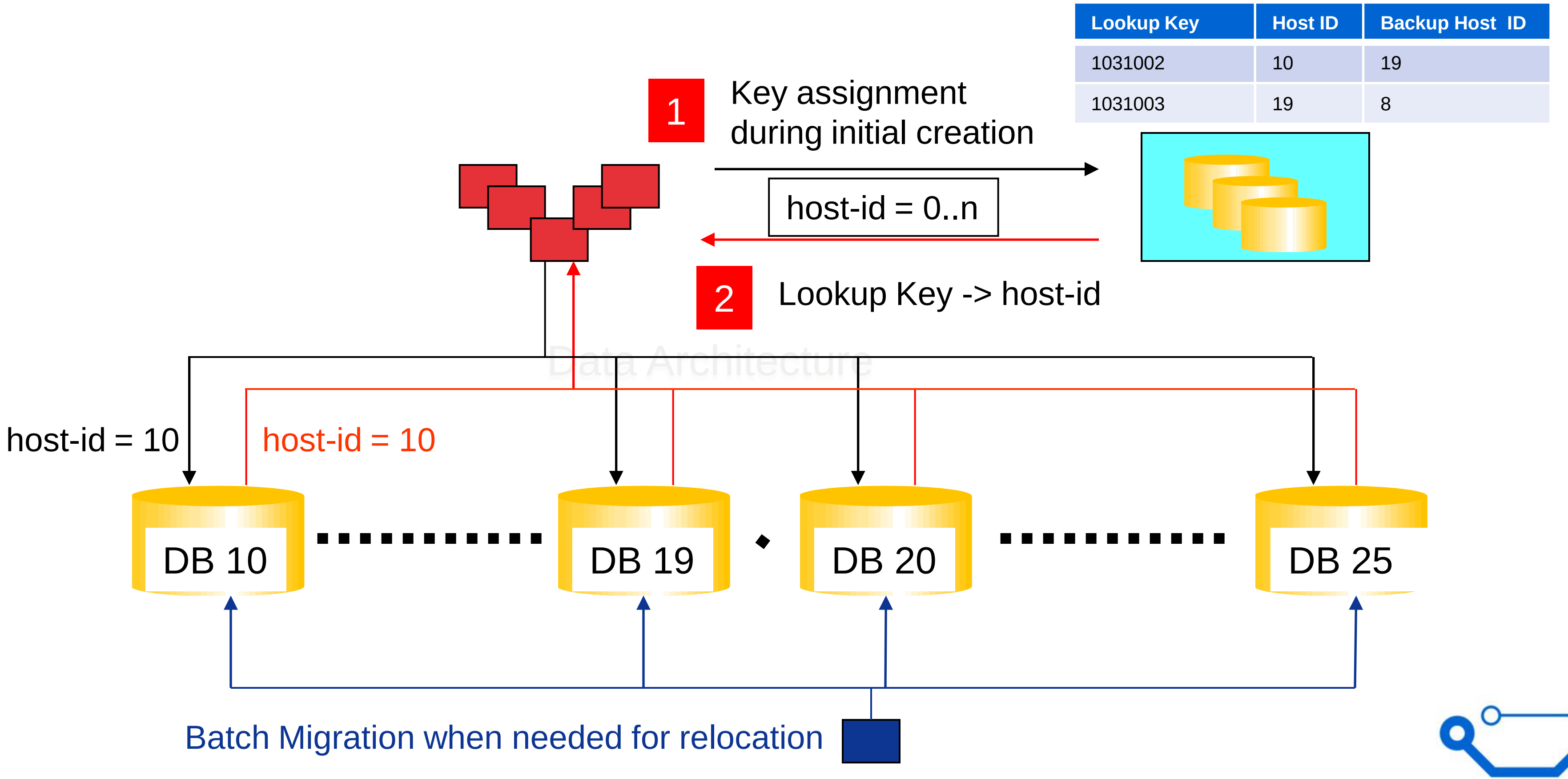
⬢ Logical Host Family

▲ Logical Host or “shard“

# Lookup Based

- Needed a custom balanced pattern as well for uneven patterns
- Extra Indirection from a Lookup database
- Requires read lookup table to map ID to host numbers
- Allocation can be round-robin or modulo
- Location is based on lookup using ID
- Uses sequence-controlled PK to ensure global uniqueness
- Pros:
  - User records can be relocated without impacting code (not tied to key)
  - Scale-out as needed. Practically, no preset upper limit on host count
- Cons:
  - Additional round trip to lookup. Thread Local Caching used to mitigate
  - Additional layer to be managed
  - Useful only for Read-mostly scenarios

# Typical Lookup Architecture



| Lookup Key | Host ID | Backup Host ID |
|---|---|---|
| 1031002 | 10 | 19 |
| 1031003 | 19 | 8 |

# Aligned Host

- Paradigm for storing ***related*** data aligned along the “stronger” key
- Motivations:
  - Multi-Data-Center compliant
  - Provides locality of reference for stronger key
  - Improved capacity management
- Key Concepts
  - Related items cannot be relocated
  - Allows for Tiered solutions for selected data
  - Location continues to be based on ID range lookups

# Wrap up

- Breaks traditional understanding of relational concepts
- Data should allow isolation and segregation
- Multiple sharding patterns are needed for various access patterns
- Needs strong routing and data access layer support
- Accepted, *understood* and implemented Data Architecture standards and patterns
- Scale up via Tech Refresh is necessary but much more manageable
- ***Real*** Scale-out via Functional segmentation and Sharding is possible!

# Q & A