Lessons from Data Management
Oracle Database and Google Cloud Spanner

Wei Hu
Vice President, High Availability Technologies

21-Feb-2018

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Outline

- Large scale OLTP customers
- Innovation in Databases
- Google Spanner
- Cloud
- Customer case study

OLTP transactions on a single Oracle database

- Futures Exchange – 200k txn/second
- Science – 150k changes/second
- User experience company – 125k operations/second
- Telecom - 13.8 million rows/second ingest

- These are single database numbers (RAC)
  - These systems have more room to scale
  - With Oracle Sharding, they will scale up even more
- Txn rates and rows/second can be very abstract, following is some context

All 3 Telco Operators in China use Oracle

- The largest, China Mobile, is also largest in the world by subscribers
- China has 22 provinces
  - China Mobile Jiangsu (80 million population), China Mobile Guangdong (108 million), China Mobile Zhejiang (55.5 million)
  - Biggest EU country, Germany, has 82 million people
- Core billing and CRM systems
- China Mobile is an early adopter of Oracle 12c
- China Telecom is early adopter of Oracle Sharding

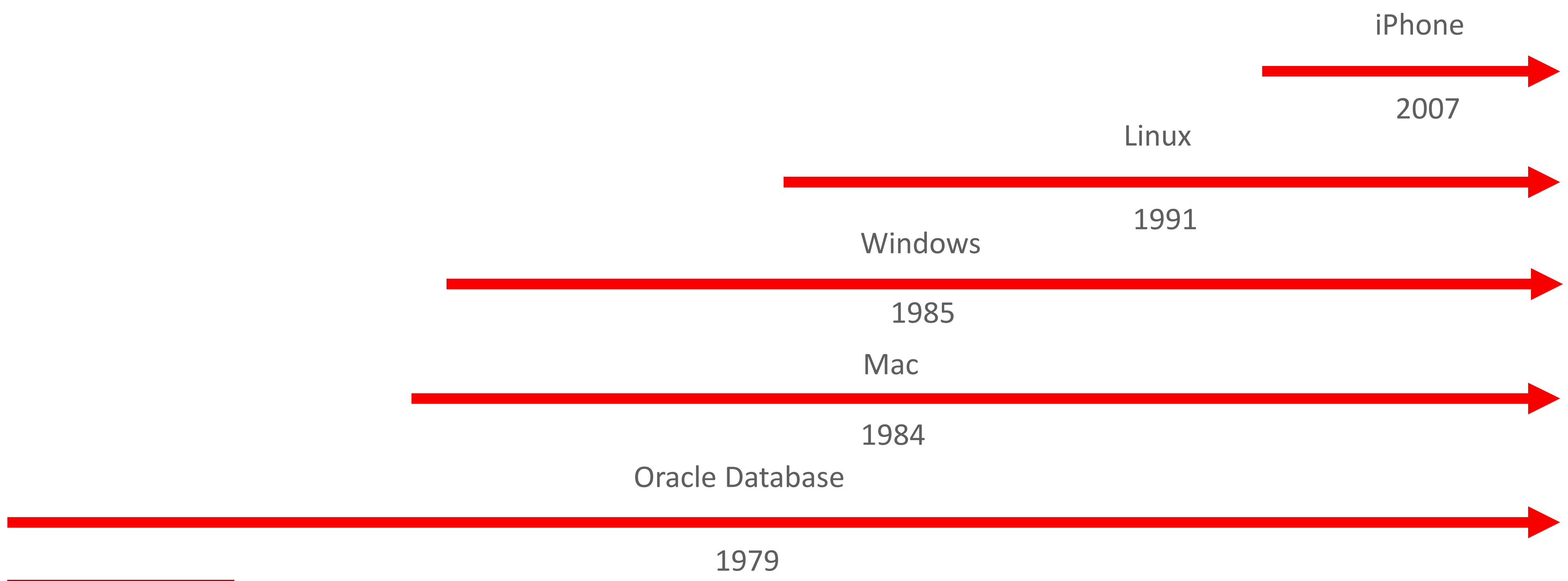
ORACLE

Biggest e-Government System Built on Oracle

- China has and is expanding the world's largest e-Government system to serve 1.38 billion population
- Consists of 12 Golden projects – Golden Finance, Golden Water, Golden Human Resources and Social Security, Golden Customs, …
- All except 1 are built on Oracle
- Some stats
  - Over 1 billion social security cards
  - Medical insurance covers 1.3 billion people (Obamacare covers ~10 million in 2017)
- Database EE, RAC, Partioning, Active Data Guard, In-memory, Audit Vault, Exadata, …
- This is China, with reputation for blocking foreign tech companies

Innovation

Relational Database is One of the Most Successful Software Platforms



ORACLE

Technological Innovation and Assimilation in Databases

- Database management undergoing constant innovation
- Database industry also has a long history of assimilating and incorporating the best ideas
- The mainstream databases (Oracle, SQL Server, DB2) have been the most successful because of their ability to incorporate the best ideas (both internally and externally developed)

Example: NoSQL databases

- NoSQL database have some significant innovations
  - Built-in JSON that makes it easy to persist application objects
  - Automated sharding makes it easy to support geo-replication and scale-out
  - Developer friendliness by integrating with latest development environments (Go, Ruby, Python,..)
  - These are really good ideas
- NoSQL databases also have bad ideas and deficiencies
- Bad ideas
  - No transactions, lack of consistency (wrong results), weak query language, proprietary interfaces instead of open standards
- Deficiencies
  - Lack of centralized security policy enforcement, poor separation between data and application logic, and poor SMP scalability
  - Lack of enterprise-grade capabilities around availability and manageability

Proliferation of One-Trick Ponies

- *Most non-mainstream (niche) databases are one-trick ponies – person or thing with only one special feature, talent, or area of expertise*
- Data management is very complicated
  - Databases are at the heart of enterprises. Mission-critical software is incredibly demanding and difficult to get right
  - Takes highly-skilled team years to build the mission-critical platform capabilities for availability, security, and manageability
  - Effort is beyond the resources and expected lifetime of any startup company
- Consequently, niche vendors focus on implementing a key idea, hoping to survive long enough to build out the rest of the product later
- Unfortunately, the rest of the product takes orders of magnitude more time than the key idea
- By comparison, it takes less time for mainstream databases to incorporate the key idea

Oracle's Strategy

- Oracle does a huge amount of innovation on its own
  - The Oracle database is widely acknowledged as the gold standard in data management
  - Autonomous database, Engineered systems, Cloud, Consistent Read, Real Application Clusters, Active Data Guard, Flashback, Edition-based redefinition, Application Continuity, Database Vault, …
  - Consistent data architecture across on-premises, cloud@customer, public cloud
- Plus, Oracle assimilates and innovates upon the best ideas in the industry
  - Dual-format in-memory column store, Integrated Sharding, native JSON, object-relational data format, XMLtype, Automatic Storage Management…
- Oracle picks up the good ideas, without the bad ideas
  - From NoSQL, Oracle picks up the good ideas of JSON and Sharding; without the bad ideas and deficiencies
- Once this is done, there is no reason to accept the compromises and deficiencies of NoSQL databases
  - All the mainstream databases: Oracle, SQL Server, and DB2, pursue this strategy

This is a trend over the past 30 years

- Over the past 3 decades, numerous startups have come out with good ideas, then died as these ideas were assimilated by the mainstream databases
  - Star schema (Red Brick, 1986)
  - Database appliances (Netezza, 1999)
  - Object database (Gemstone, Objectivity, ObjectStore, Versant, Illustra, … 1988-1990s)
  - XML database (Tamino, MarkLogic, eXist-db, … 2000-2004)
  - NoSQL database
  - Columnar (Vertica, VectorWise, ParAccel)
  - In-memory
  - Cloud
  - …

Google Cloud Spanner

Spanner Borrowed Good Ideas

- Good ideas borrowed from relational databases
  - Schema for separation between data and code, SQL, strong consistency, transactions
- Interesting turn-around for Google and strong statement about viability of NoSQL / NewSQL approaches
  - Map Reduce (Hadoop) 2004-2005 - Non-relational, key-value pairs
  - BigTable (Hbase) 2006 - Non-relational, No SQL, data as uninterpreted strings, no type columns, secondary indexes, triggers, advanced query language, …
  - Unfortunately, these out-dated ideas (Non-relational, Key-value pairs, Eventual consistency, Proprietary APIs) infected other NoSQL/NewSQL databases: Dynamo (2007), Cassandra (2010), …
- Spanner re-discovers relational data management and SQL as cutting edge technologies

Relational model, Consistency, Transactions, and SQL are Good Ideas

- *NoSQL datastores … limited API and loose consistency models complicate application development,* Megastore paper, Google, January 2011
- complexity of dealing with a non-ACID data store in every part of our business logic would be too great, and there was simply no way our business could function without SQL queries, F1 paper, Google, 2013

- Developing good applications is hard
- Dealing with relaxed consistency (wrong results) is *very* hard; lack of transactions means application has more difficulty handling failure cases
- Programming to non-standard, less powerful interfaces makes it harder (and locks you into proprietary APIs and programming models)

Other Data Management Good Ideas Forgotten by Many

- Program-Data Independence and Schema
  - Allows data to be shared by multiple applications
  - Can understand structure of data without reading source code
  - Can change data definition without changing every application using the data
- Normalization means you do not need to duplicate the data
- Can query across all your data sources
- Enforcing security at the database is more secure – you are not vulnerable to your most poorly-written application
- Use standard interfaces - there is no NoSQL standard

Do You Think This SQL Runs on Spanner?

Transfer $500 from Savings Account to Checking Account

```
UPDATE savings_accounts
  SET balance = balance - 500
  WHERE cust_id = 12345 AND account = 3209;
```

Decrement Savings Acct
(Savings table constraint: amount > $100)

```
UPDATE checking_accounts
  SET balance = balance + 500
  WHERE cust_id = 12345 AND account = 3208;
```

Increment Checking Acct

```
INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (13, cust_id, account, balance, -500)
    FROM savings_accounts WHERE cust_id = 12345 AND account = 3209;

INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (12, cust_id, account, balance, 500)
   FROM checking_accounts WHERE cust_id = 12345 AND account = 3208;
```

Journal the transfer operations
(tran_cod 13 – transfer from
tran_code 12 – transfer to)

```
COMMIT WORK;
```

End transaction

Spanner has Rudimentary SQL Support

- Spanner SQL only has SELECT and some DDLs. INSERT, UPDATE, MERGE *not* supported
- Very limited datatypes – BOOL, INT64, FLOAT64, STRING, BYTES, ARRAY, STRUCT, DATE, TIMESTAMP
  - Far less than what one expects from real RDBMS
  - Missing important modern datatypes such as JSON, XML, LOBs, Spatial, CHAR, VARCHAR…
- Restrictive and limited data model
  - Every table must have a primary key
  - No Views, synonyms, materialized views, foreign key constraints, triggers, windowing functions, temporal, default column values…
  - Other limitations imposed by Spanner's replication implementation (to be discussed later)
- Partial SQL implementation means you do not get the benefits of standard interfaces and must use proprietary APIs
  - Developers have to do more work, and applications are less portable

ORACLE

Highly Restrictive Transactions

- Cannot use SQL with Spanner transactions
  - Must implement as custom logic calling Spanner's proprietary API (Instead of SET TRANSACTION, COMMIT, ABORT)
- Cannot read your own writes
  - If you read a value you updated before you commit, you get wrong results
- Locks are not persistent
  - Might lose a lock, and only find out during transaction commit
  - Can workaround this by verifying your reads, but that is expensive
- Spanner transactions are very slow because of synchronous communication and locking across regions
  - Google recommends doing stale reads (at least 10 seconds of staleness) to bypass locking overhead
  - Google's own applications do a lot of additional work to cope with this high latency
    - Read in batches, read asynchronously in parallel, buffer writes; use coarse schemas so that there are fewer tables and columns, and fewer joins
- These and other limitations place a huge burden on application developers

Oracle SQL vs Spanner Program

```
/* transfer $500 from savings to checking */
/* all tables are sharded by cust_id */
/* savings table constraint: amount > $100 */
UPDATE savings_accounts
  SET balance = balance - 500
  WHERE cust_id = 12345 AND account = 3209;
UPDATE checking_accounts
  SET balance = balance + 500
  WHERE cust_id = 12345 AND account = 3208;

/* log new balances (read your own writes) */
/* tran_code = 13 – transfer from */
INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (13, cust_id, account, balance, -500)
    FROM savings_accounts WHERE cust_id = 12345 AND account = 3209;
/* tran_code = 12 – transfer to */
INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (12, cust_id, account, balance, 500)
   FROM checking_accounts WHERE cust_id = 12345 AND account = 3208;

COMMIT WORK;
```

```
static void writeWithTransaction(DatabaseClient dbClient) {
  dbClient
    .readWriteTransaction()
    .run(
      new TransactionCallable<Void>() {
        @Override
        public Void run(TransactionContext transaction) throws Exception {
          // Transfer $500 from SAVINGS to CHECKING. We do the transfer in a transaction to
          // ensure that the transfer is atomic.
          Struct row =
              transaction.readRow("SAVINGS_ACCOUNTS", Key.of(12345, 3209),
                                  Arrays.asList("SavingsRow"));
          long savingsBalance = row.getLong(2);
          // Transaction will only be committed if sufficient funds (> $100) constraint is met
          // at the time of commit. Otherwise it will be aborted and the callable will be rerun
          // by the client library.
          if (savingsBalance >= 600) {
            long checkingBalance =
                transaction
                    .readRow("CHECKING_ACCOUNTS", Key.of(12345, 3208),
                             Arrays.asList("CheckingsRow"))
                    .getLong(2);
            long transfer = 500;
            checkingBalance += transfer;
            savingsBalance -= transfer;
            transaction.buffer(
                Mutation.newUpdateBuilder("CHECKING_ACCOUNTS")
                    .set("cust_id")
                    .to(12345)
                    .set("account")
                    .to(3208)
                    .set("CheckingsRow")
                    .to(checkingBalance)
                    .build());
            transaction.buffer(
                Mutation.newUpdateBuilder("SAVINGS_ACCOUNTS")
                    .set("cust_id")
                    .to(12345)
                    .set("account")
                    .to(3209)
                    .set("SavingsRow")
                    .to(savingsBalance)
                    .build());
            // Log new balances, cannot read your own writes
            // mutations: checkingBalance, savingsBalance
            // must fit in memory!
            transaction.buffer(
              Mutation.newInsertBuilder("JOURNAL")
                .set("tran_code")
                .to(13)
                .set("cust_id")
                .to(12345)
                .set("account")
                .to(3209)
                .set("balance")
                .to(savingsBalance)
                .set("change_amount")
                .to(transfer)
                .build());
            transaction.buffer(
              Mutation.newInsertBuilder("JOURNAL")
                .set("tran_code")
                .to(12)
                .set("cust_id")
                .to(12345)
                .set("account")
                .to(3208)
                .set("balance")
                .to(checkingBalance)
                .set("change_amount")
                .to(transfer)
                .build());
          }
          return null;
        }
      });
} // writeWithTransaction
```

Primitive SQL Execution Engine

- Lacks access structures, optimizer, and adaptive plans for both serial and parallel query
  - Lacks advanced access structures (bitmap indexes, zonemaps, materialized views)
  - Does not provide in-memory capabilities or take advantage of vector instructions
  - No support for data compression (useful for IO acceleration)
  - Must add FORCE_INDEX hint for execution plan to use a global secondary index
  - Essentially no optimizer
- Oracle uses dynamic sampling, cost-based optimization, and adaptive plans to automatically generate the best execution plan
  - This type of advanced capabilities takes decades to implement and refine

Big Gaps in SQL Functionality

- Aside from SELECT-only…
- Spanner does not support analytics
  - Statistical functions, data mining, analytic functions, pattern matching, sampling of tables, partition wise joins, lateral views, SQL model clause for analytics, extensions to GROUP BY with rollup and grouping sets, aggregations like LIST_AGG, …
- Combining OLTP and analytics in the same system delivers real-time data and simplifies development and management

Replication

- Spanner uses synchronous replication
  - Consistency achieved at cost of huge latencies when the replicas are distributed
  - These latencies cannot be hidden and must be handled by the application
- Oracle supports synchronous and asynchronous replication
  - Active Data Guard with synchronous replication that can adaptively transition to async replication when there is a network failure
  - Active Data Guard Far Sync that allows you to achieve zero data loss over long distances using sync + async
  - GoldenGate supports active/active multi-master updates with automatic conflict resolution
- Oracle Global Data Services load balances to all available replicas based on locality and load
- Spanner replicates data to all the regions
  - Does not work in new world where data sovereignty is the rule
  - China, Germany, France, Russia, … require data about their citizens to be stored in their country
- Oracle Sharding supports data sovereignty by allowing you to shard across regions where each region has a set of replicas for its own data

Monitoring, Tuning, Security

- Spanner lacks monitoring and tuning tools
  - Oracle has wealth of features to monitor and tune the application
  - Access advisors (index, partition, and materialized view advisors)
  - SQL monitoring and tuning, AWR, ASH, v$ performance views
  - Real Application Testing with DB replay
- Spanner lacks enterprise-level security
  - Spanner has basic notion of users, roles, and encryption
  - Lacks powerful mechanisms for separation-of-duties, least privilege, monitoring, and advanced auditing capabilities as provided by Data Vault, Audit Vault, Real Application Security, …
  - Particularly important in the cloud where you relies on others for your security
    - Data Vault ensures that even Cloud DBAs cannot access your credentials or data
    - Real Application Security greatly limit damage caused by common phishing attacks

Cloud Spanner is Proprietary Lock-in to Google Cloud

- Cloud Spanner is available only on the Google Cloud
  - Not available on-premises or on any other cloud
  - Data formats and APIs are proprietary and incompatible with other databases
    - Difficult to migrate off
- This is a problem with IBM mainframes and now with many cloud vendors
- Oracle has a strong hybrid cloud offering
  - Data is 100% compatible on Oracle Cloud and on-premises
  - Easy to migrate data between Oracle Cloud and on-premises
  - Oracle sharding allows some shards to be on-premises, in the cloud, or have replicas of the same shard be both on-premises and cloud

99.999% SLA

- 99.999% uptime over a year means 5 minutes downtime per year or 26 seconds per month
- Per Google:
  - **"Downtime Period"** means a period of five consecutive minutes of Downtime with a minimum of 60 requests per minute. Intermittent Downtime for a period of less than five minutes will not be counted towards any Downtime Periods
- If the system is down for 4 consecutive minutes, it’s considered up. You can have a whole series of 4 minute downtimes, and still not be considered down
- Google caps damage at 10% of monthly bill until you get down to 99.0%. You need to be down more than 7 hour 18 minutes before you get more than 10% damages

What is Google Spanner's *trick*

- If Spanner is a one-trick pony, what is that trick?
- We've seen Spanner's limitations
  - You probably don't want Oracle to move to eliminate all SQL except SELECTs, eliminate datatypes, eliminate true transactions, …
- Which innovative aspects of Spanner should mainstream databases adopt?
  - Google name?
  - Synchronous replication over wide area networks (?)
  - What else?
- Please think about it and let us know

Summary of Cloud Spanner Limitations

- Very limited support for SQL, datatypes, and relational model
- High-overhead and difficult-to-use transactions
- Primitive query engine and lack analytics support
- Synchronous, high-latency copy-everywhere replication that does not meet data sovereignty requirements
- Limited security capabilities
- Proprietary lock-in to Google Cloud

- *Not proven to run any really complex systems like ERP, CRM, etc. with hundreds of thousands of tables*

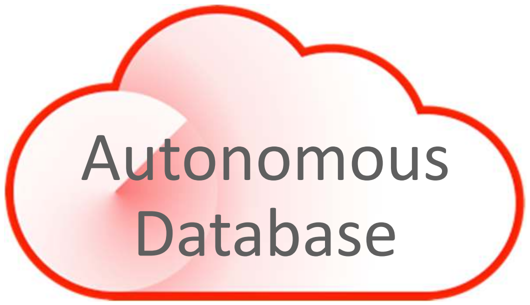
Back to Innovation

Mainstream Databases vs One-Trick Ponies

- Mainstream databases like Oracle have been very successful in innovating and incorporating the best ideas from the entire database community
- As these ideas are incorporated in mainstream products, the niche vendors inevitably die out
- The accumulation of capabilities makes the mainstream databases ever stronger; making it even harder for niche vendors to be competitive

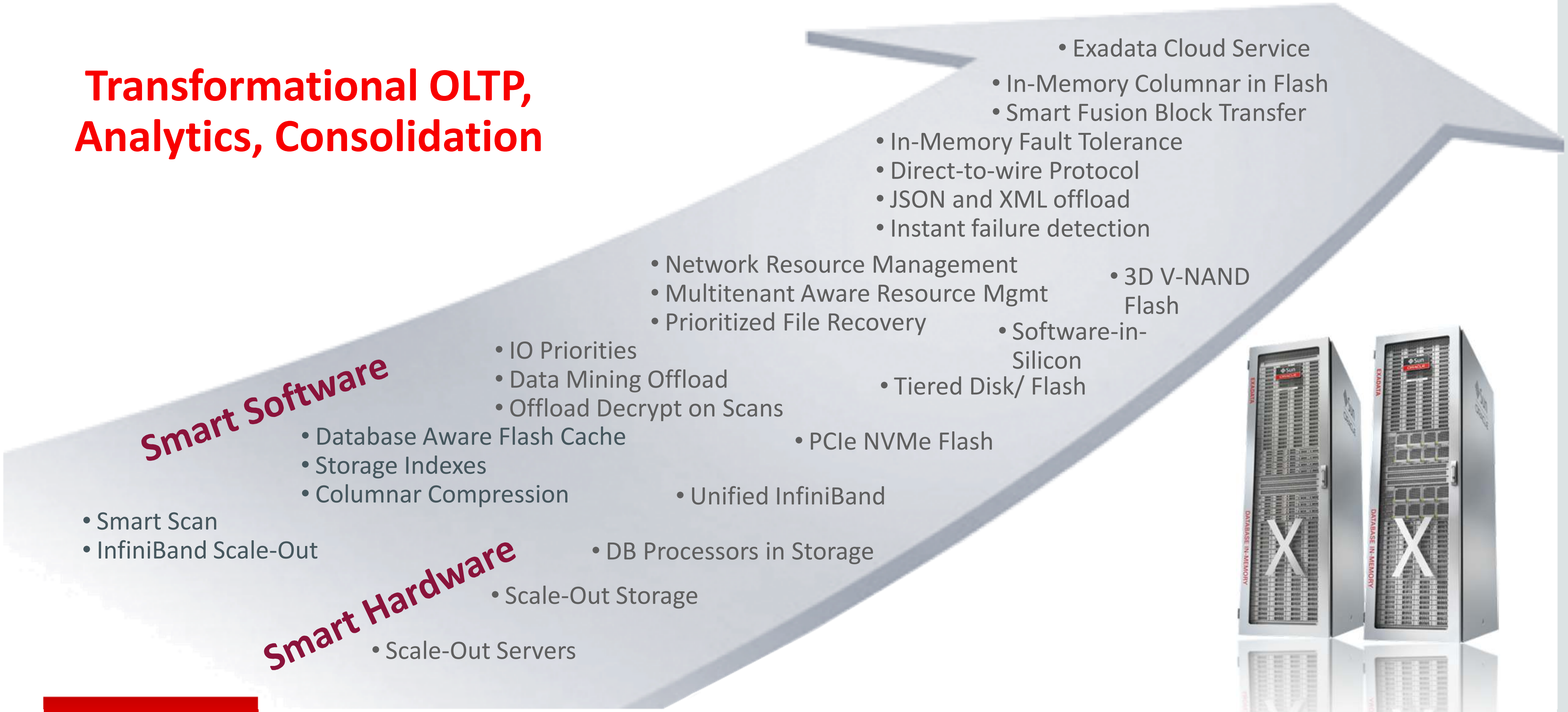
Oracle’s Vision for Autonomous Database

- Self-Driving
  - User defines service levels, database makes them happen
- Self-Securing
  - Protection from both external attacks and malicious internal users
- Self-Repairing
  - Automated protection from all downtime



Exadata Innovations over the Years

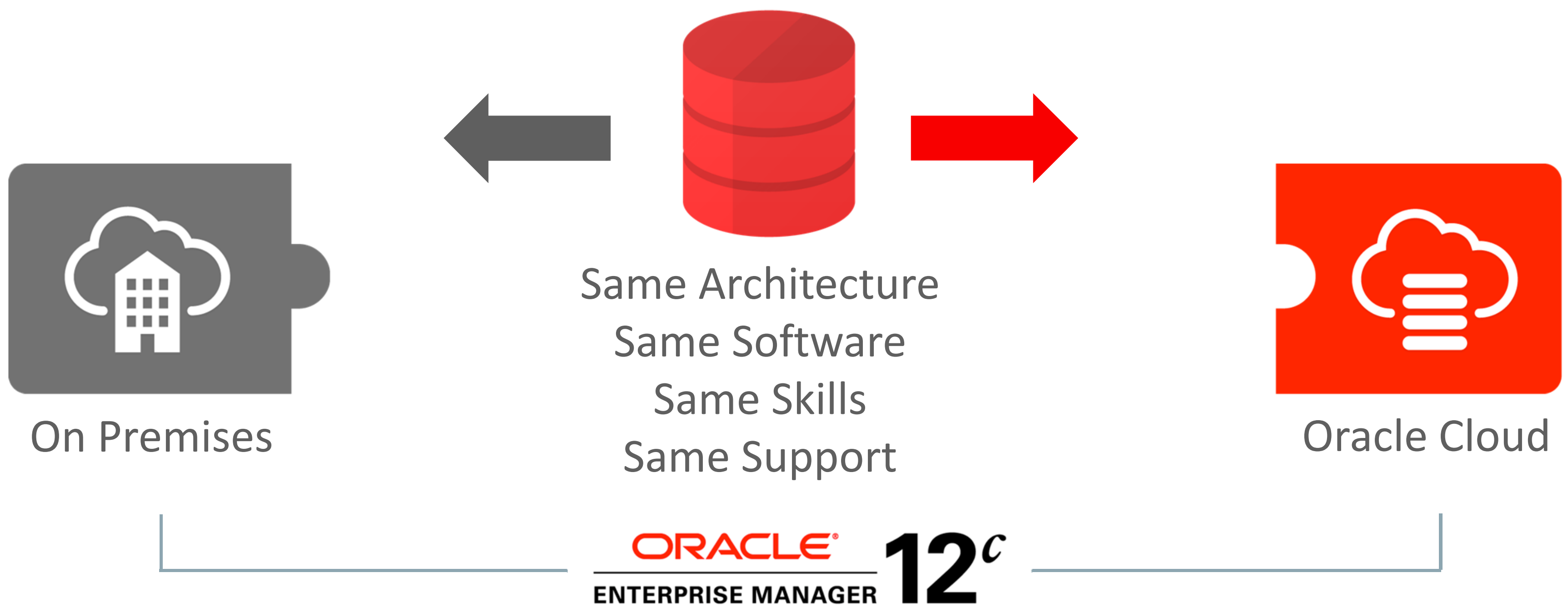
Transformational OLTP, Analytics, Consolidation



- Smart Scan
- InfiniBand Scale-Out
- Scale-Out Servers
- Scale-Out Storage
- Database Aware Flash Cache
- Storage Indexes
- Columnar Compression
- DB Processors in Storage
- IO Priorities
- Data Mining Offload
- Offload Decrypt on Scans
- Unified InfiniBand
- PCIe NVMe Flash
- Network Resource Management
- Multitenant Aware Resource Mgmt
- Prioritized File Recovery
- Tiered Disk/ Flash
- Software-in-Silicon
- 3D V-NAND Flash
- In-Memory Fault Tolerance
- Direct-to-wire Protocol
- JSON and XML offload
- Instant failure detection
- In-Memory Columnar in Flash
- Smart Fusion Block Transfer
- Exadata Cloud Service

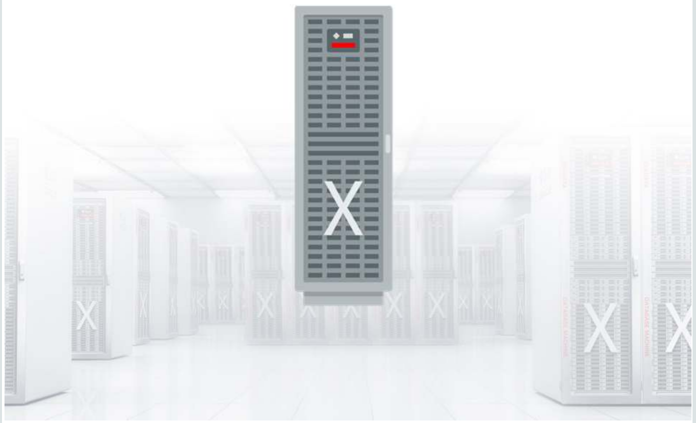
Oracle Platform-as-a-Service Strategy

Full portability across the hybrid cloud



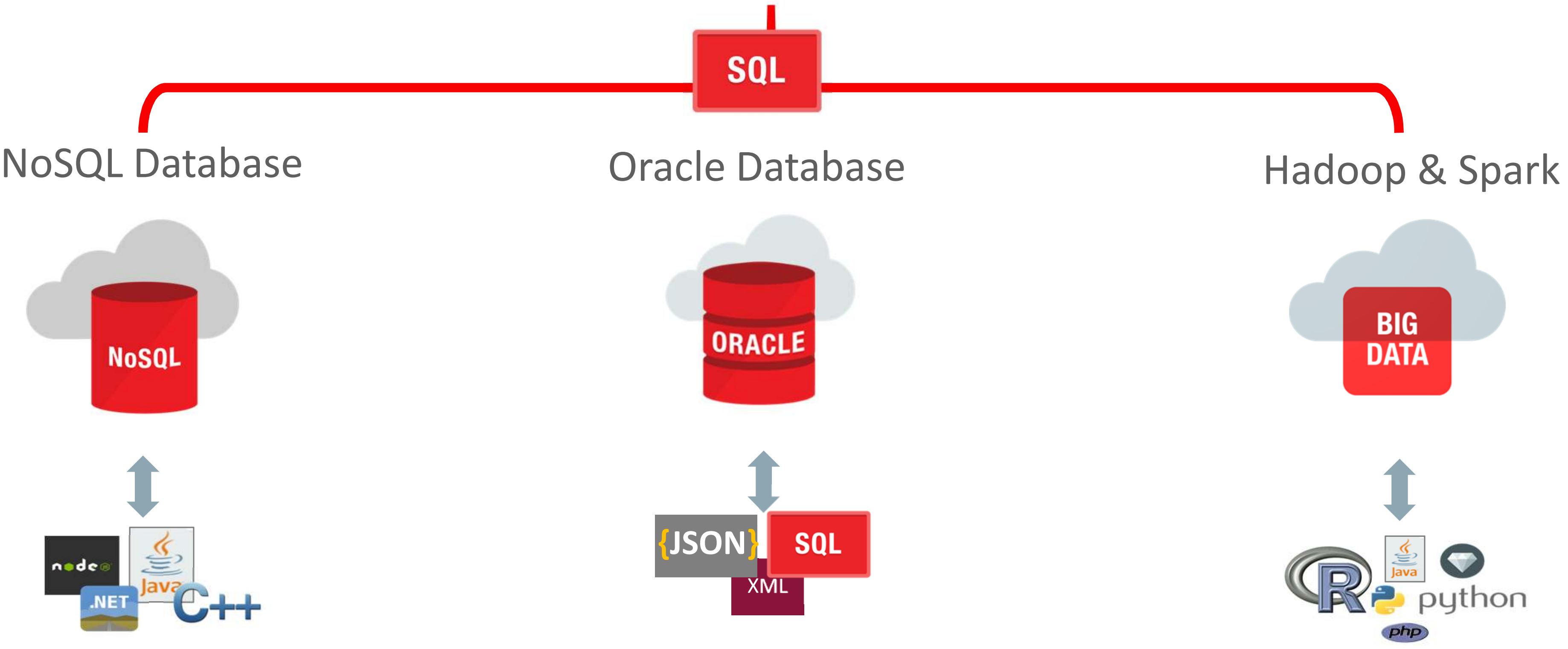
Enterprise Manager manages both On Premises and Cloud

Choice of Exadata Deployment Models

Exadata Database Machine	Exadata Cloud Machine	Exadata Cloud Service
		
Customer Data Center	Customer Data Center	Oracle Cloud
Purchased	Subscription	Subscription
Customer Managed	Oracle Managed	Oracle Managed

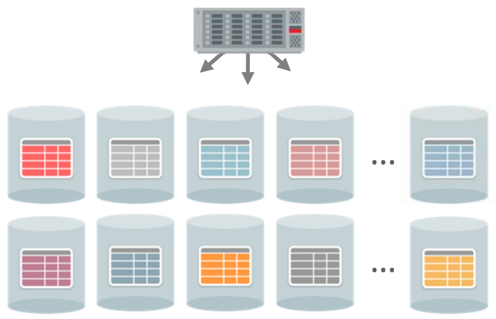
Strategy – Unified Data Management Platform Services

Special purpose and standards based access



Sharding

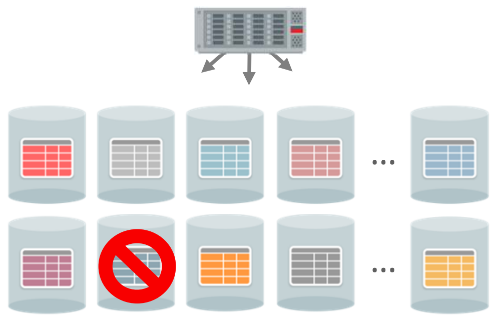
Linear Scalability



Add shards online to increase database throughput

Online split and rebalance

Fault Tolerant

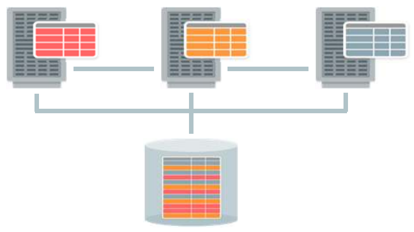


No shared hardware or software to isolate faults.

Shards may run different Oracle releases

RAC Sharding

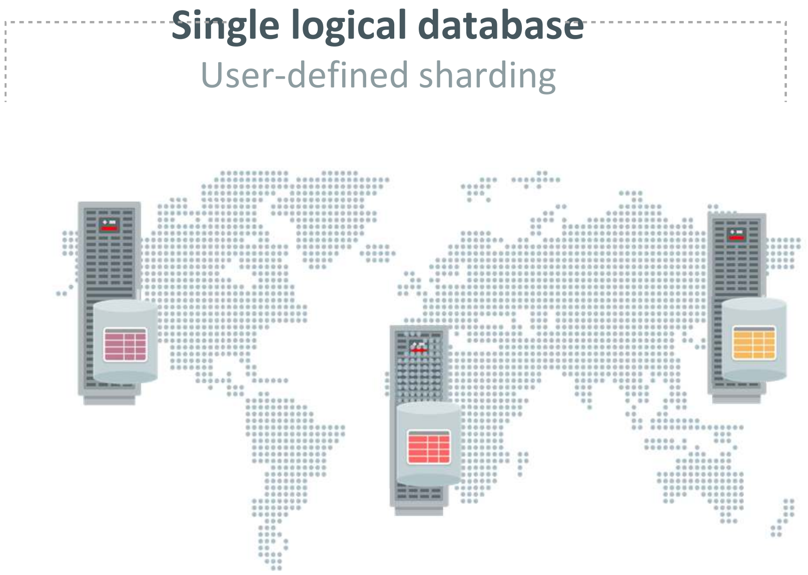
NEW IN 18c



Combines sharding with RAC to allow sharding to work with complex applications

Global-scale Distributed Databases

- Multi-year effort
  - Oracle 12*c* Global Data Services load balances requests across globally distributed replicas
  - Composite sharding in 12.2 and user-defined sharding in 18c extends this further
- Some customers have databases across regions (lower latency, regulatory needs, …)
  - Data mostly accessed within the region
  - Also need view of data across all regions
- Cross-region Composite and User-defined sharding
  - Single sharded database; shards *can* have different data; can have replicas within region
  - Local, fast access to shard in region
  - Multi-shard queries access data on all shards
  - Each shard can be further scaled using sharding or RAC
- Supports hybrid cloud or cloud bursting



Amazon.com

- AWS (Amazon Web Services) competes with Oracle
  - Supports open source databases – MySQL, Postgres
  - Develops own databases – Aurora, Redshift, Dynamo
- Promotes migration from Oracle to AWS
- They have the technology, they have a lot of people, understand Open Source, and highly motivated
- Amazon uses Dynamo to store customer shopping carts
- What databases does Amazon use for its core systems for payments, retail, and digital commerce?

Amazon.com runs Oracle

- Amazon Payments Platform runs on Oracle
  - This is the eCommerce payments system supporting millions of Amazon customers (developers, buyers, and sellers)
- Amazon Retail Systems run on Oracle
  - Multiple large databases supporting OLTP and DSS
- Amazon Digital Commerce Platform runs Oracle
  - Supports the services behind the Kindle and other Amazon digital businesses
- By Amazon’s own admission, these are all large-scale, massively concurrent, highly available distributed Oracle systems

ORACLE

Integrated Cloud

Applications & Platform Services