What's New in the Next Generation of Oracle Database

Gerald Venzl (@GeraldVenzl)

Senior Principal Product Manager
Database Development
June 08, 2017

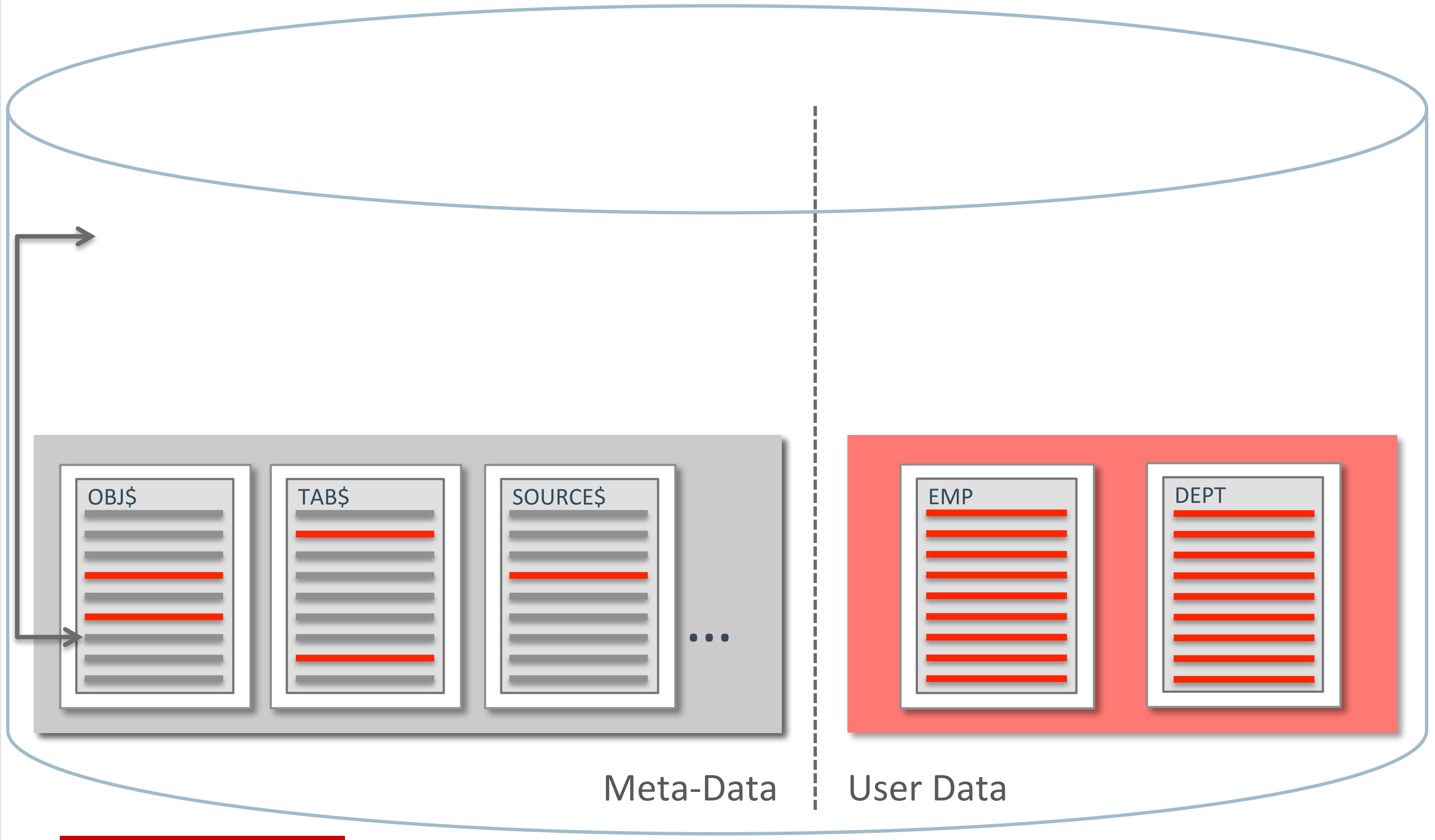
Safe Harbor Statement

Program Agenda

Program Agenda

Oracle Data and User Data

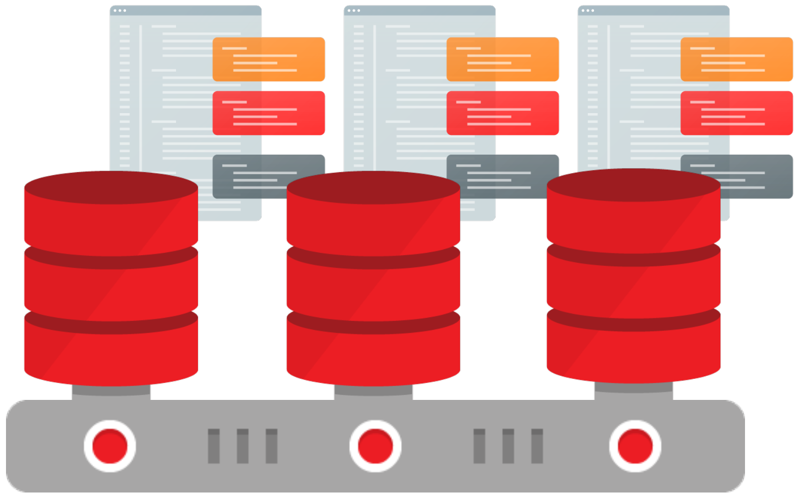
Before 12.1: Oracle and user data intermingle over time



- New database contains Oracle meta-data only
- Populate database with user data
  - Oracle and customer meta-data intermingled
  - Portability challenge!
- Multitenant fix: *Horizontally-partitioned data dictionary*
  - Only Oracle-supplied meta-data remains in root

Application Containers
Programs replicated across PDBs

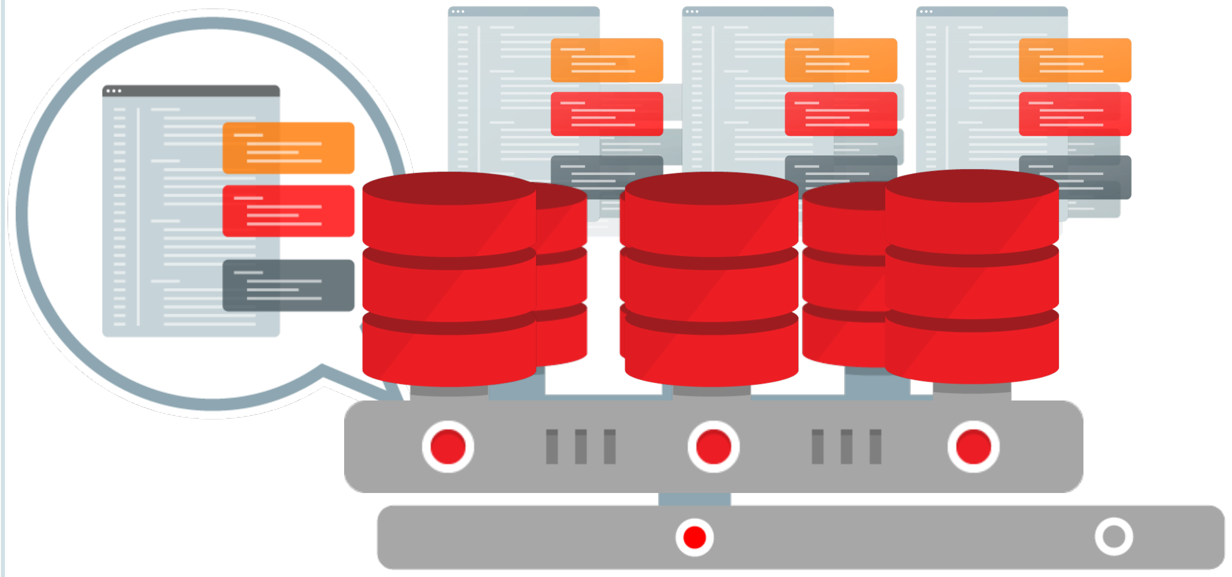
the brooklyn bean V2



Application Containers

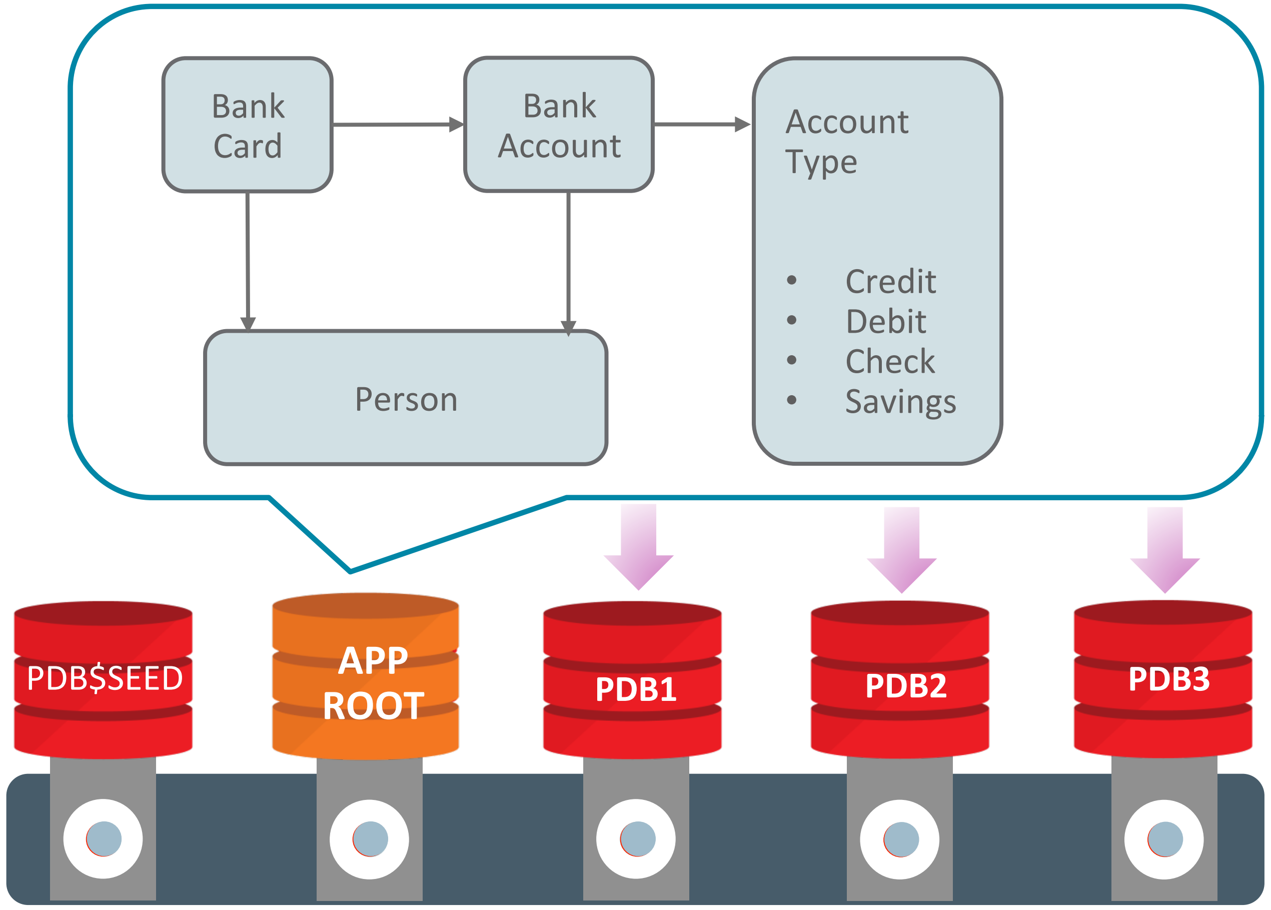
Root container for your applications

- Application Container comprises
  - Application Root (Master)
  - Application PDBs (for each Tenant)
  - Application Seed (for provisioning)
- PDBs share application objects
  - Code, metadata and data
- Further simplifies management
  - Apply updates to application container
  - Sync tenant PDBs from central master
- Suitable for all applications
  - SaaS, franchise, divisional, etc.



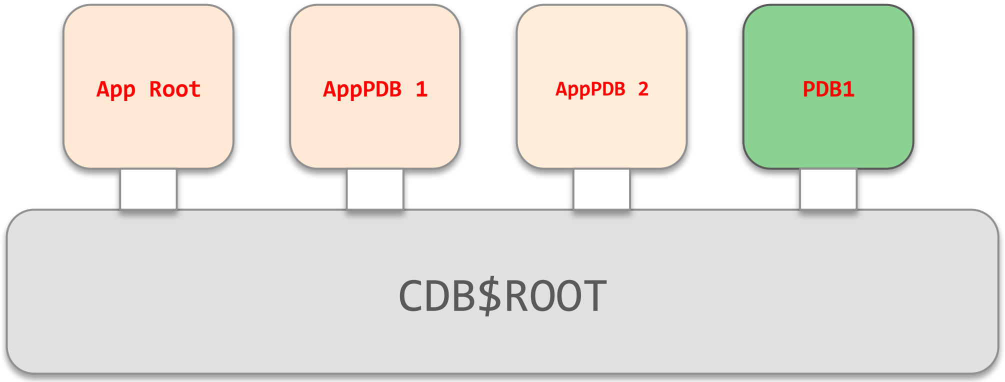
Application Containers

Share & propagate across multiple PDBs

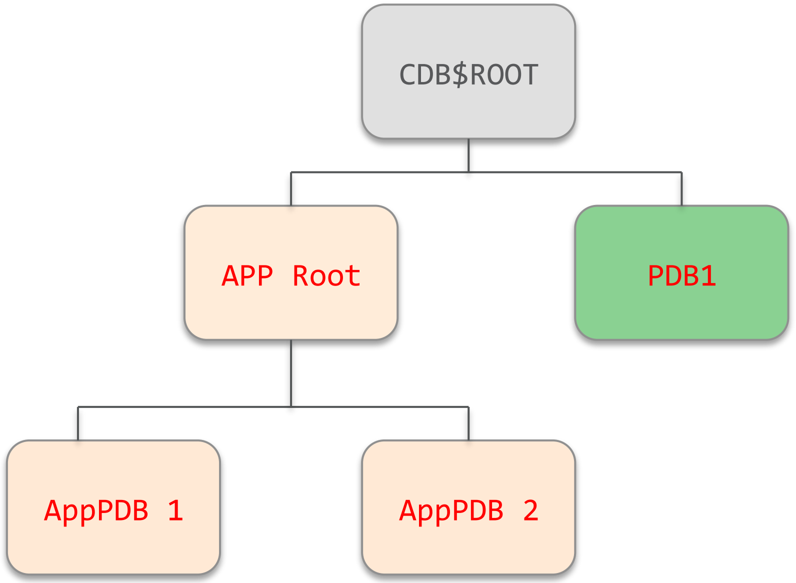


What is an Application Container ?

- An Application container is a collection of PDBs consisting of Application Root and all Application PDBs associated with it



Physical Representation



Logical Representation

Application Containers

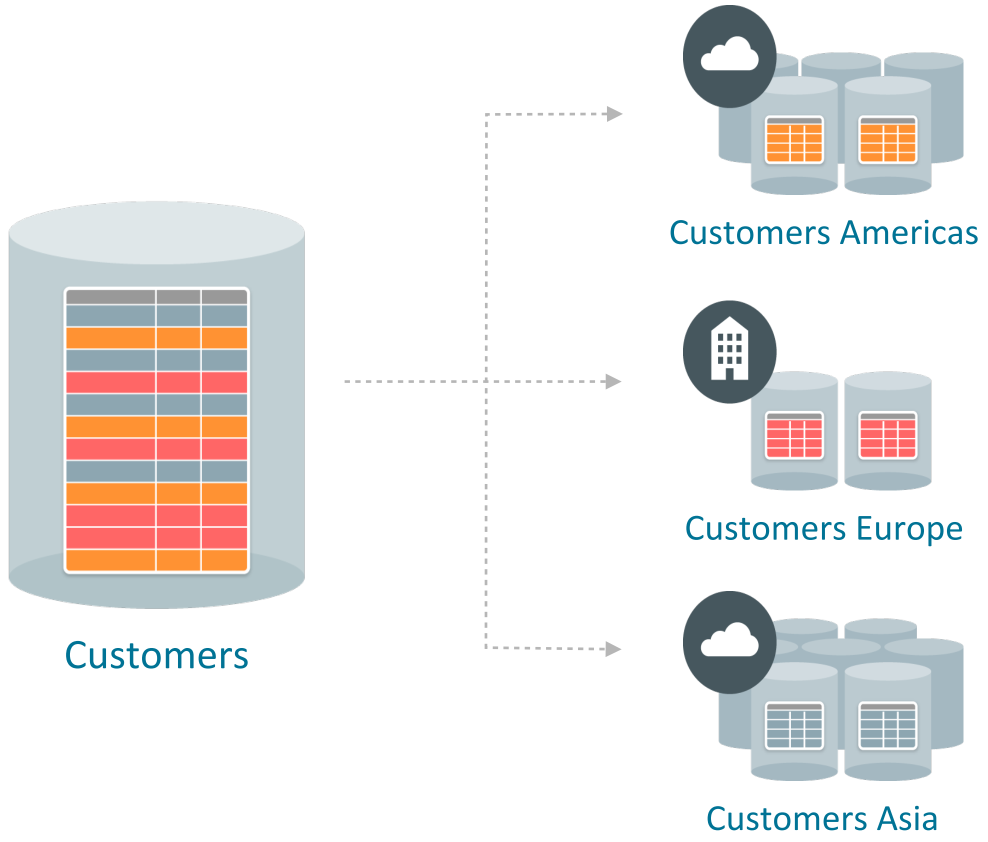
The future of Database Application Development

- Application Root PDB for defining application master
  - Metadata and common data shared across tenant PDBs
- Install one copy of your application
- Instant provisioning of an Application PDB/Tenant (with a seed PDB)
- Container Data views for reporting across PDBs (CONTAINERS clause based)
- Supports in-place simple patching
- Supports Unplug/Plug upgrade across Application Root

Program Agenda

Oracle Database Sharding

Oracle Database for web-scale applications



One giant database partitioned into many small databases (shards)

- **RAC and Data Guard meet needs of over 99% of applications** while preserving application transparency
- Some **Global-Scale OLTP applications** prefer to **shard** massive databases into a farm of smaller databases
  - Avoid scalability or availability edge cases of a single large database
  - Willing to customize data model and applications to enable transactions to be automatically routed to the right shard
- Native SQL for sharding tables across up to 1000 Shards
  - Routing of SQL based on shard key, and cross shard queries
  - Online addition and reorganization of shards
  - Linear scalability of data, workload, users with isolation

Application Suitability for Sharding

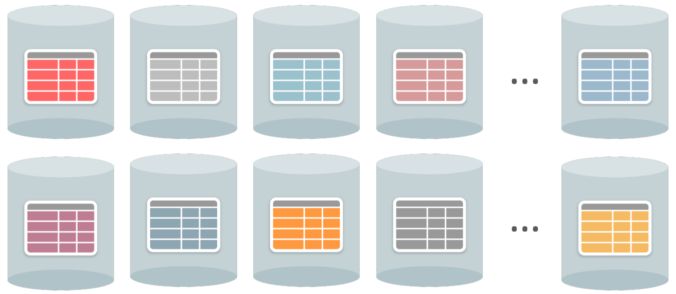
OLTP Applications with the Following Characteristics

- Applications for massive scale
  - E.g. e-commerce, mobile, social etc.
- Applications must be shard-aware
- Primary usage pattern
  - Single-shard operations based on shard key , e.g. customer_id, account_id etc.

Oracle Sharding Automated Distribution

Enhanced SQL syntax for Sharding

```
…

CREATE SHARDED TABLE Customers
( CustId     VARCHAR2(60) NOT NULL,
  FirstName VARCHAR2(60),
  LastName  VARCHAR2(60),
  …
    PRIMARY KEY(CustId),
)
PARTITION BY CONSISTENT HASH (CustId)
 …
```



- SQL syntax for creating sharded tables
  - Not proprietary APIs as with NoSQL
- Creation of a sharded table automatically partitions data across shards
  - Transparent resharding as data grows
- Choice of sharding methods:
  - System managed - consistent hash
  - User defined - range, list
  - Composite - range-hash, list-hash
- Common reference data (e.g. Price List) is automatically duplicated on all shards
- Supports shard placement in specific geographies to satisfy government data privacy

Sharded Schema

Customers

Customer	Name
123	Mary
456	John
999	Peter

Orders

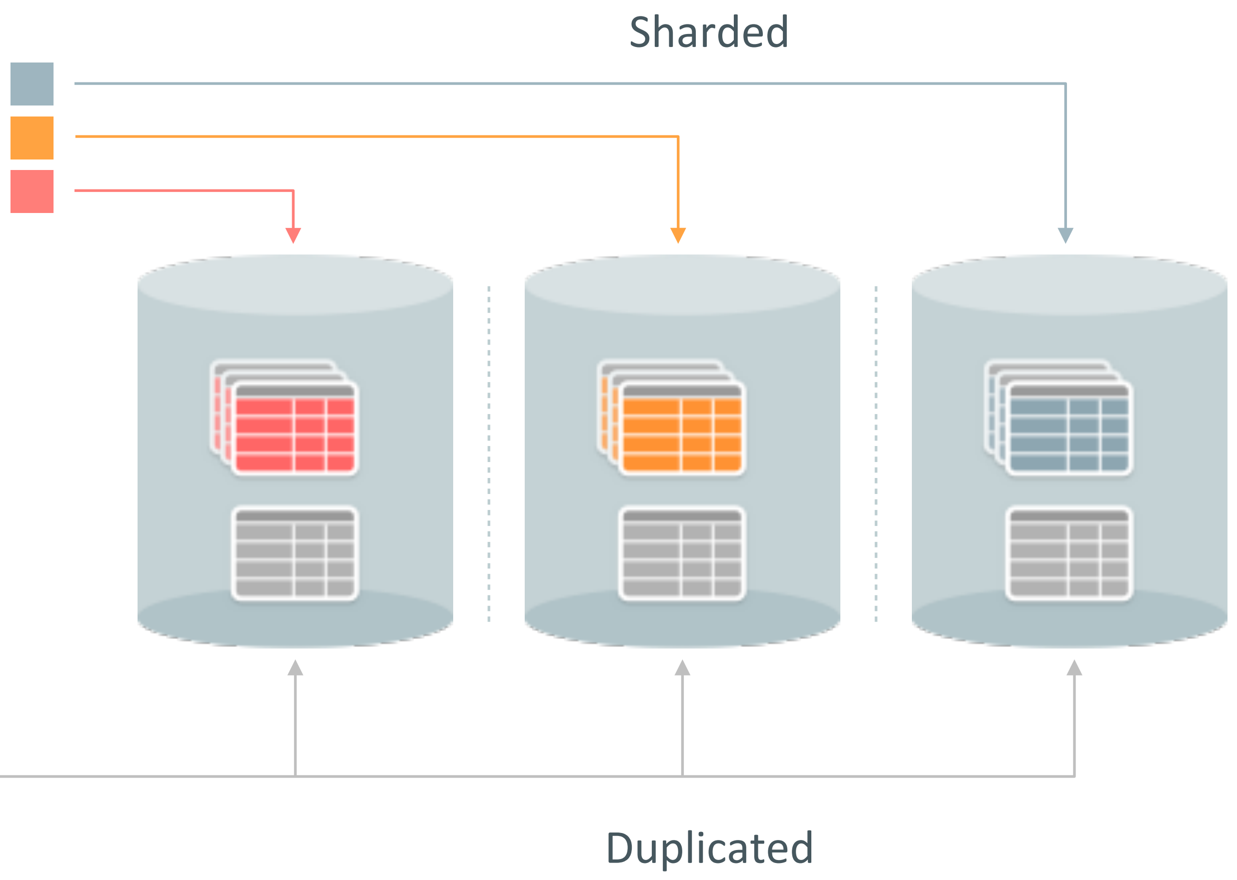
Order	Customer
4001	123
4002	456
4003	999
4004	456
4005	456

Line Items

Line	Order
40011	1001
40012	4003
40013	4001
40014	4004
40015	4003
40016	4003

Products

SKU	Product
100	Coil
101	Piston
102	Belt



Sharded Table Family – Enhanced SQL DDL Syntax

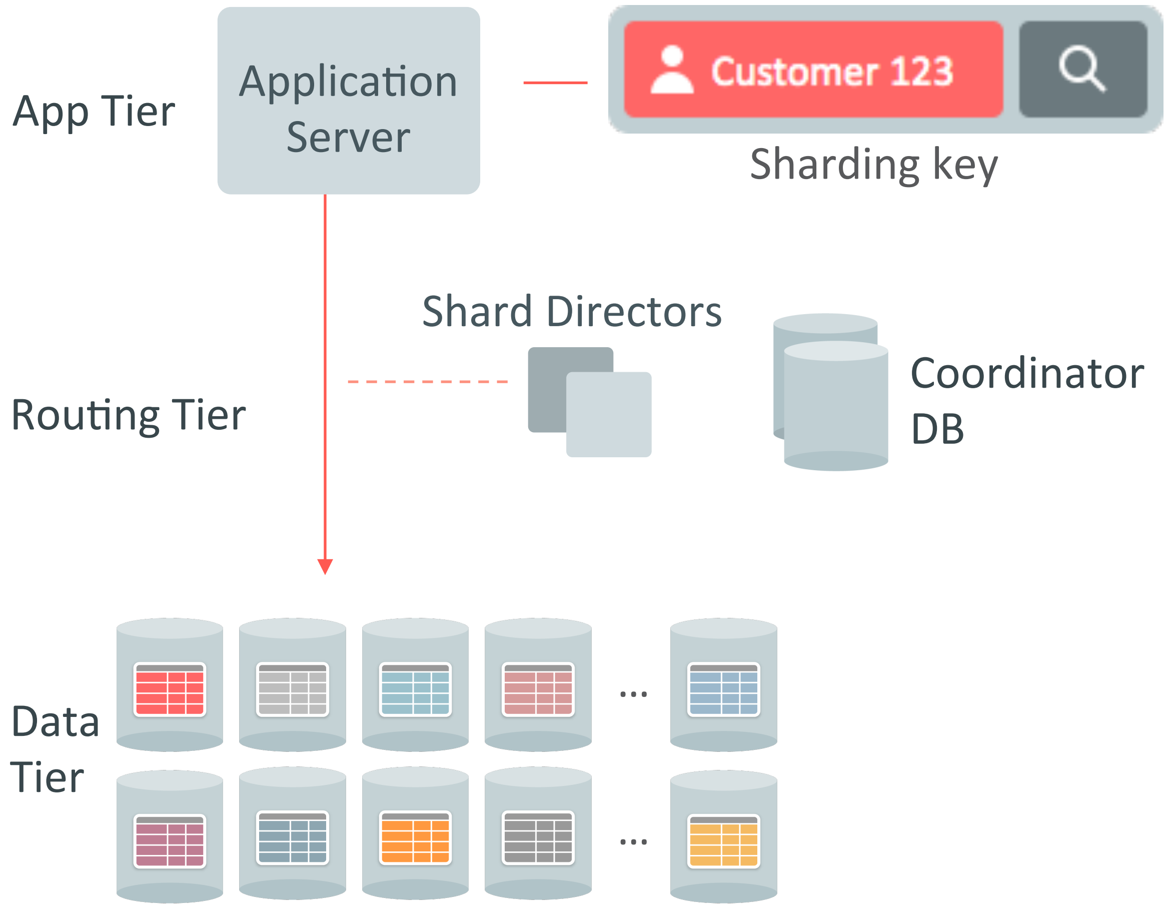
```
CREATE SHARDED TABLE Customers
( CustNo NUMBER NOT NULL,
  Name VARCHAR2(50),
 ….
  Class VARCHAR2(3),
  CONSTRAINT RootPK PRIMARY KEY(CustNo)
)
PARTITION BY CONSISTENT HASH (CustNo)
PARTITIONS AUTO
TABLESPACE SET ts1 ;
```

```
CREATE SHARDED TABLE Orders
( OrderNo NUMBER(5),
  CustNo NUMBER(3),
  OrderDate DATE ,
  …
  CONSTRAINT CustFK FOREIGN KEY
(CustNo)
    REFERENCES Customers(CustNo)
)
PARTITION BY REFERENCE (CustFK) ;
```

```
CREATE LOOKUP TABLE Products(
SKU NUMBER(4) PRIMARY KEY,
Product VARCHAR2(20),
Price NUMBER(6,2))
)
TABLESPACE dupl ;
```

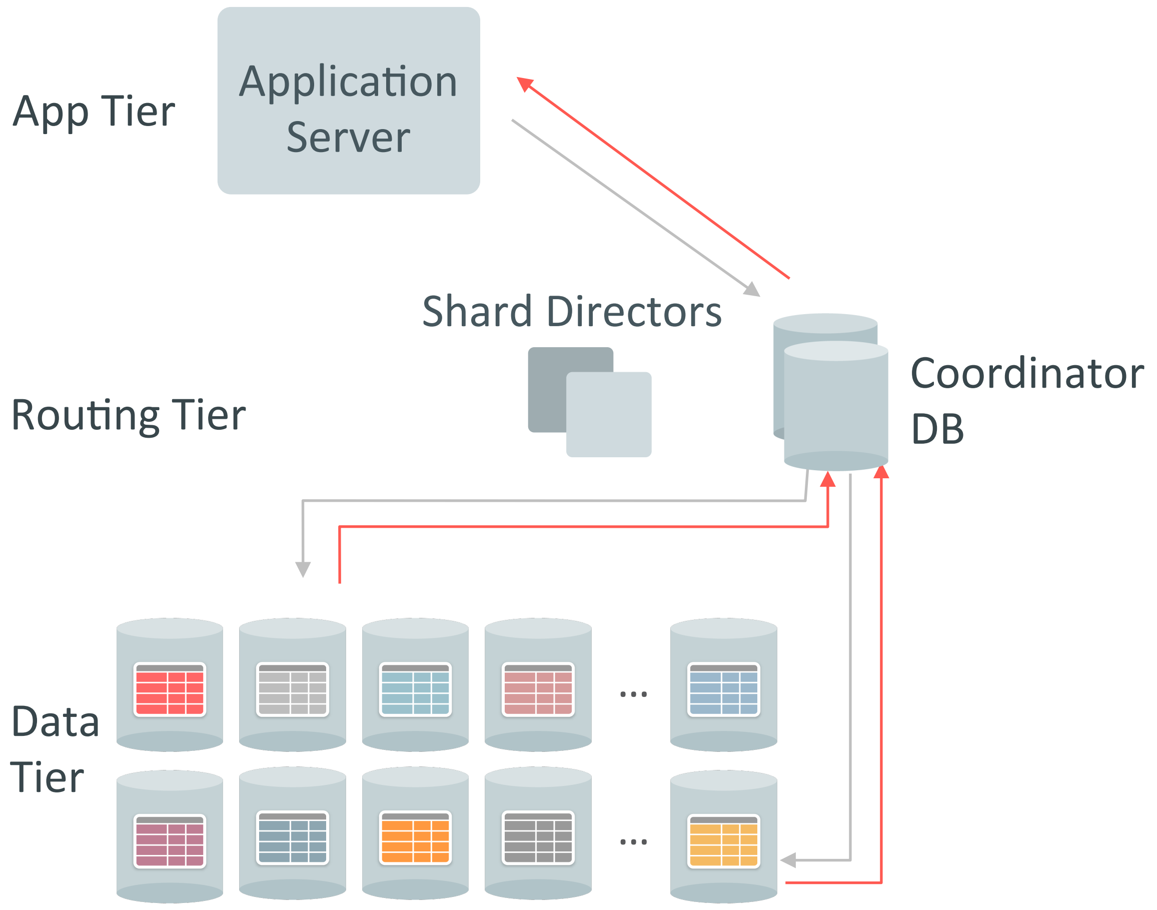
Routing Support on Client for Highest Speed

- Clients pass sharding key (e.g. Customer ID) to Connection pool, connection is routed to the right shard
- Fast: caching key ranges on client ensures that most accesses go directly to the shard
- Scalable: easily scales with more clients and shards
- Supports UCP, OCI, ODP.NET, and JDBC



Non-Shard Key Access & Cross-Shard Queries

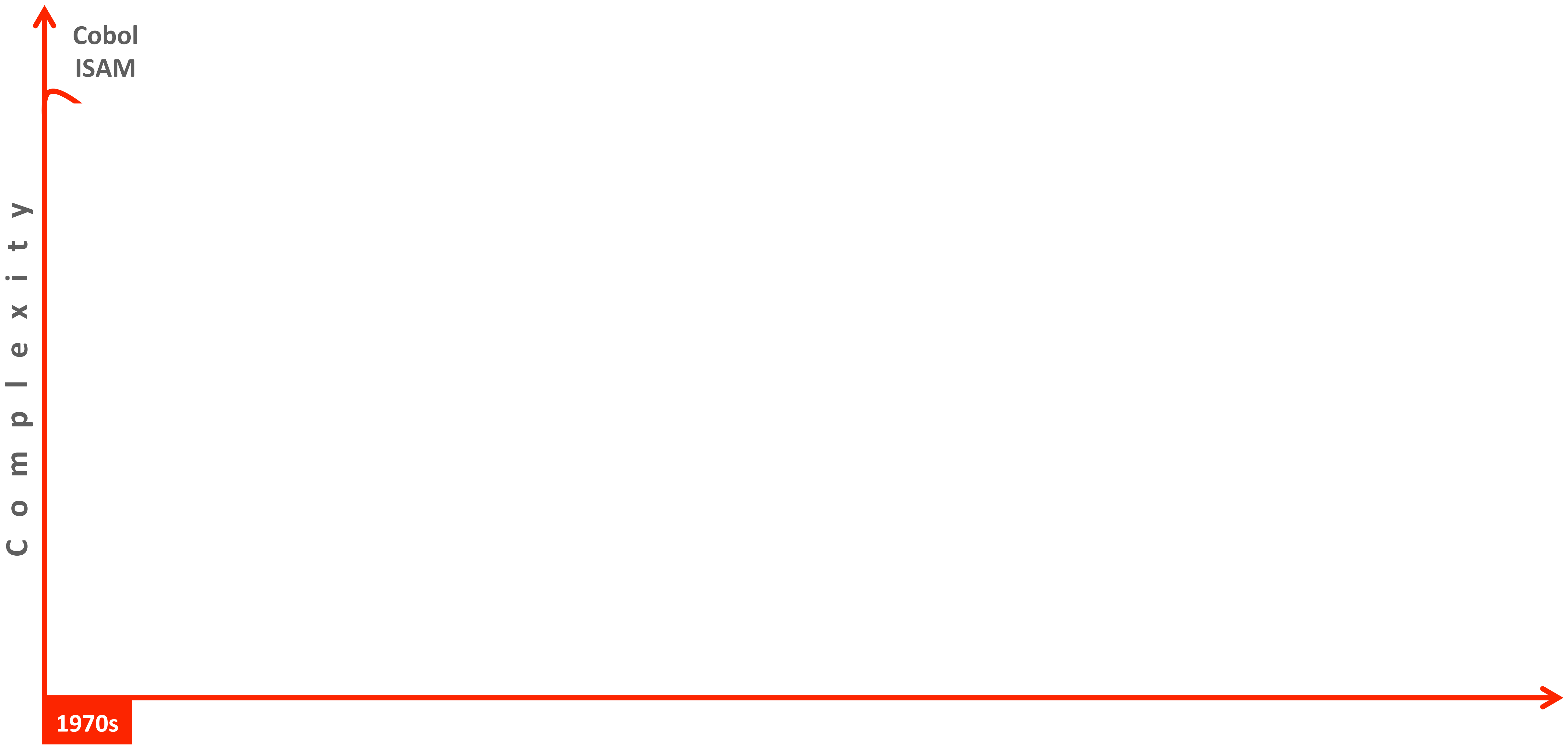
- If client does not pass shard key to Connection pool, the connection is made to the coordinator database
- Coordinator parses SQL and will proxy/ route request to one or more shards
  - Supports shard pruning and scatter-gather
- For developer convenience and not for high performance
- Supports many but not all Queries
- No Update support



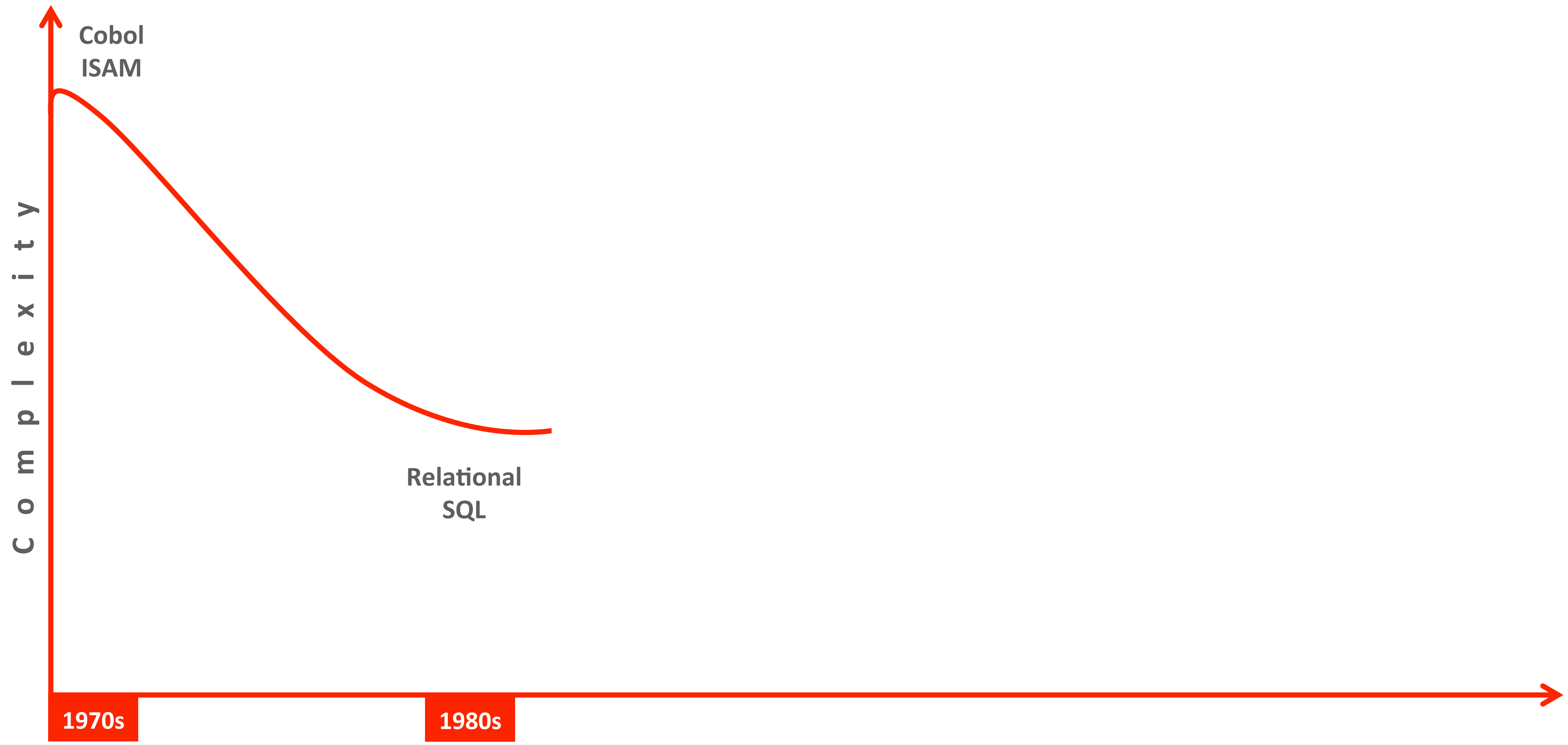
Program Agenda

BACK TO THE FUTURE

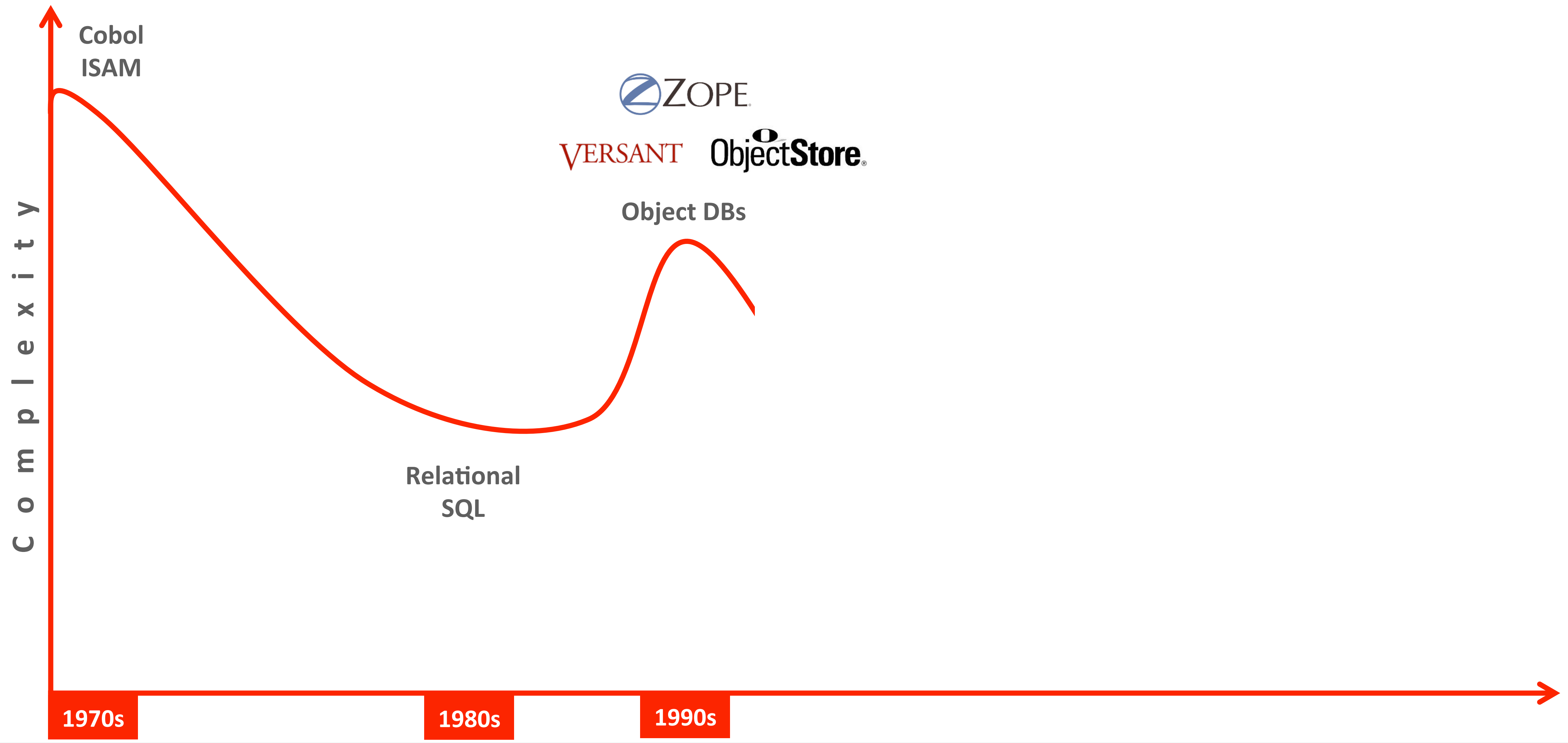
Evolution of data management



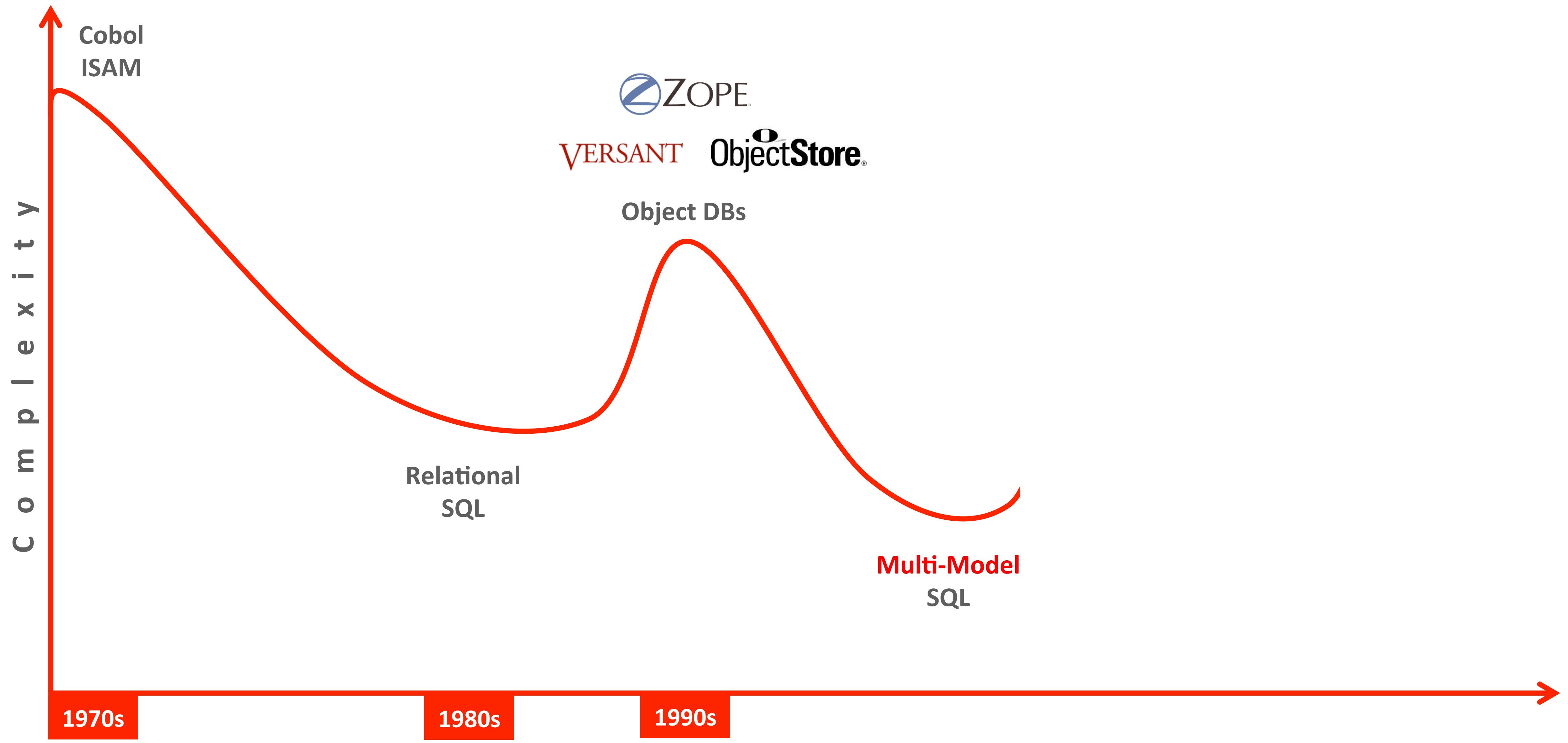
Evolution of data management



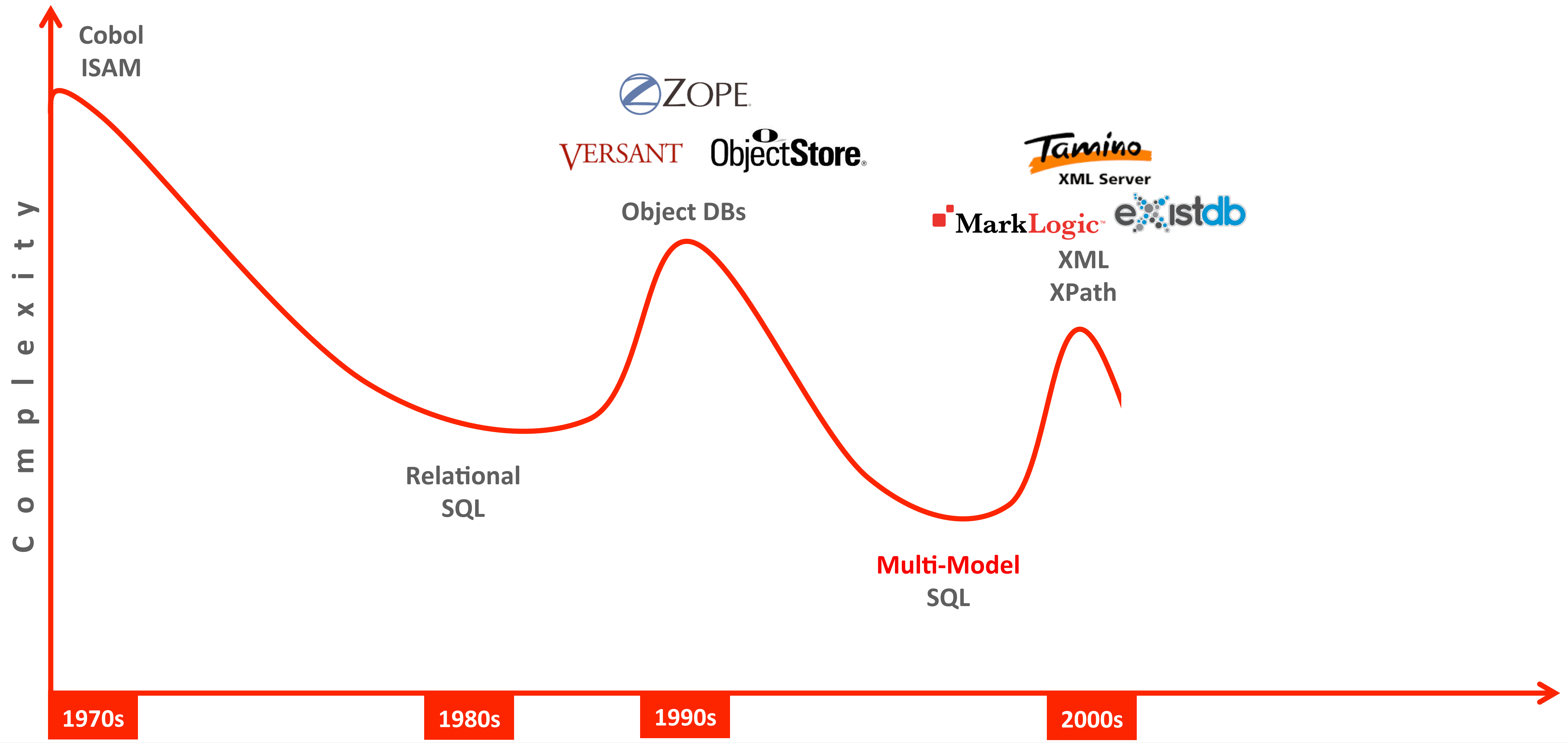
Evolution of data management



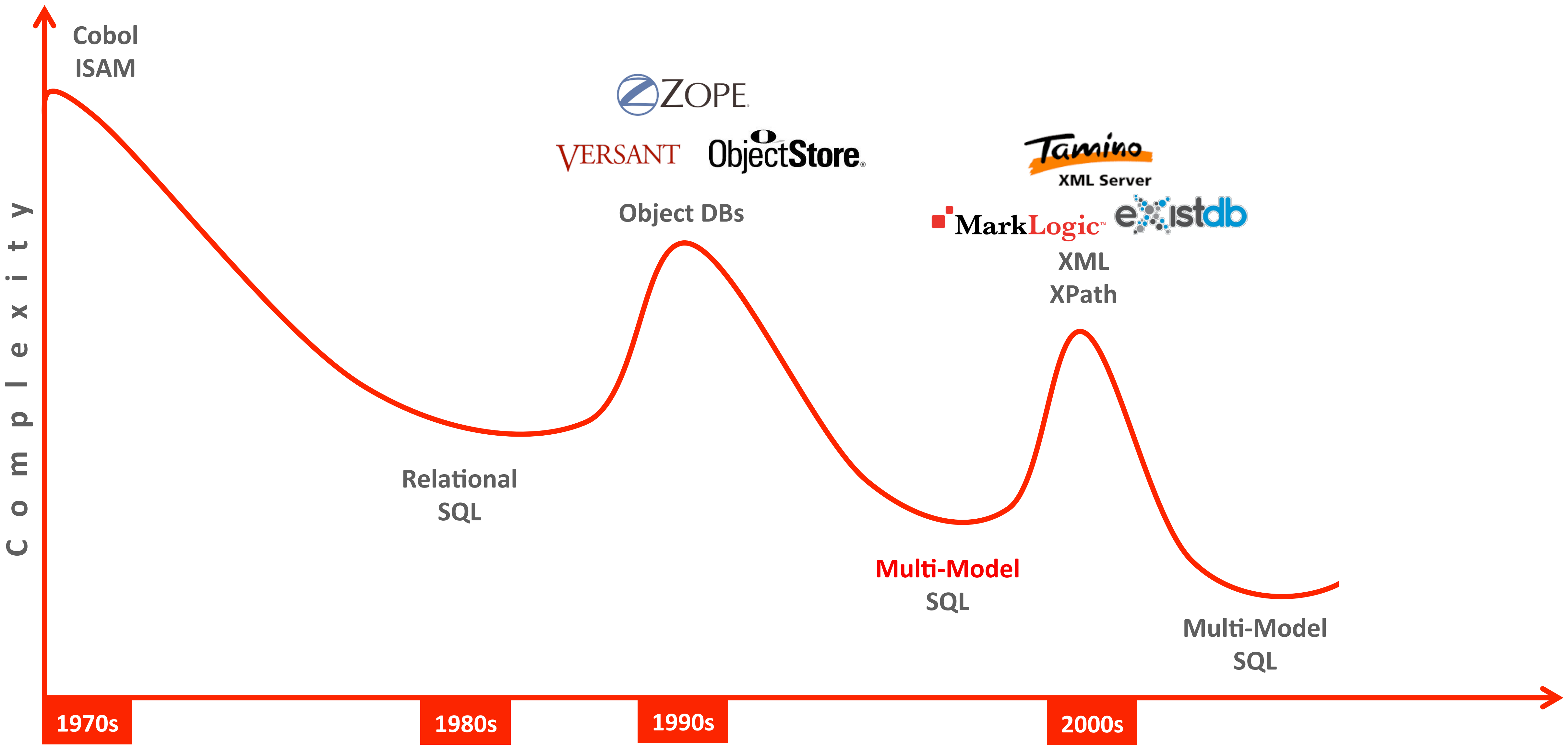
Evolution of data management



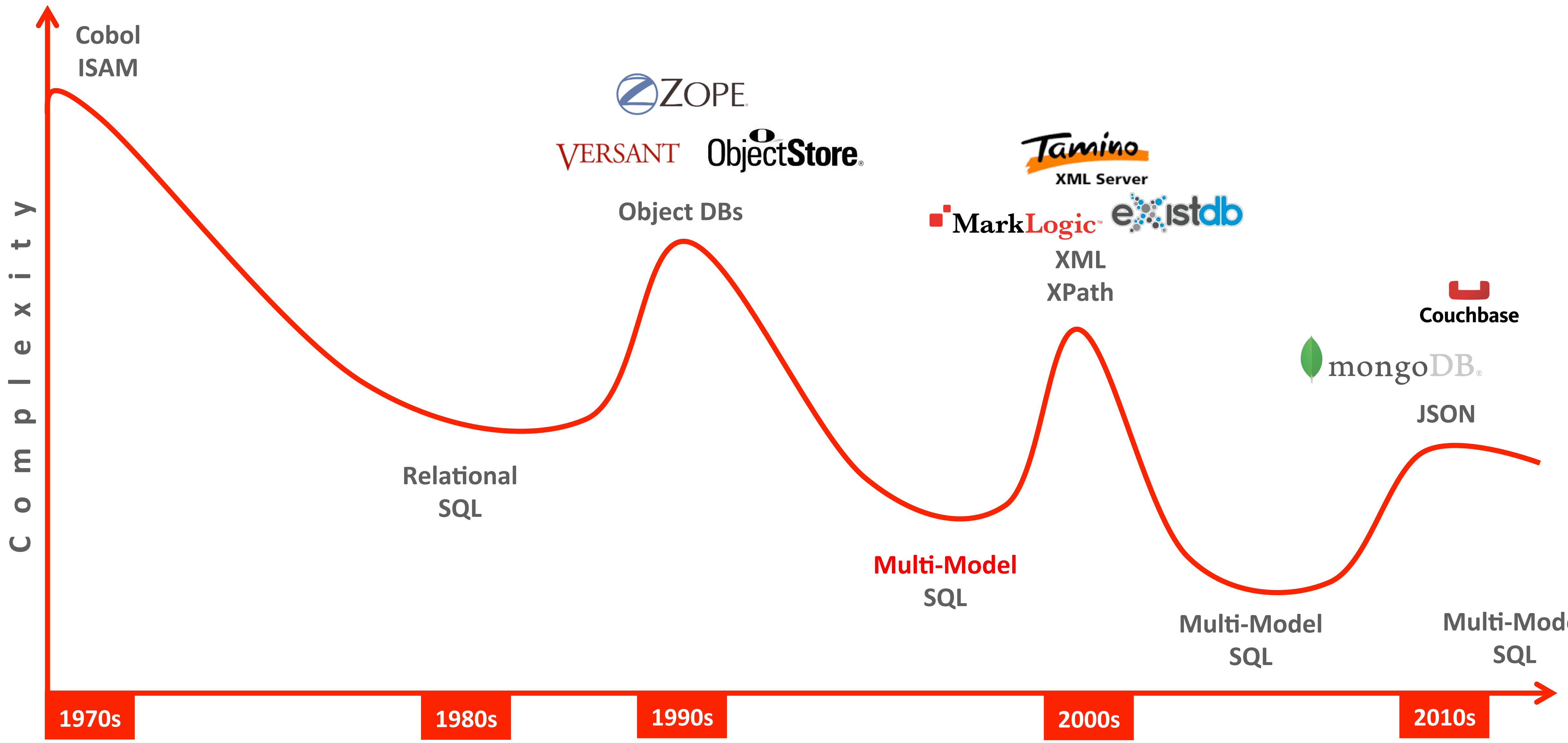
Evolution of data management



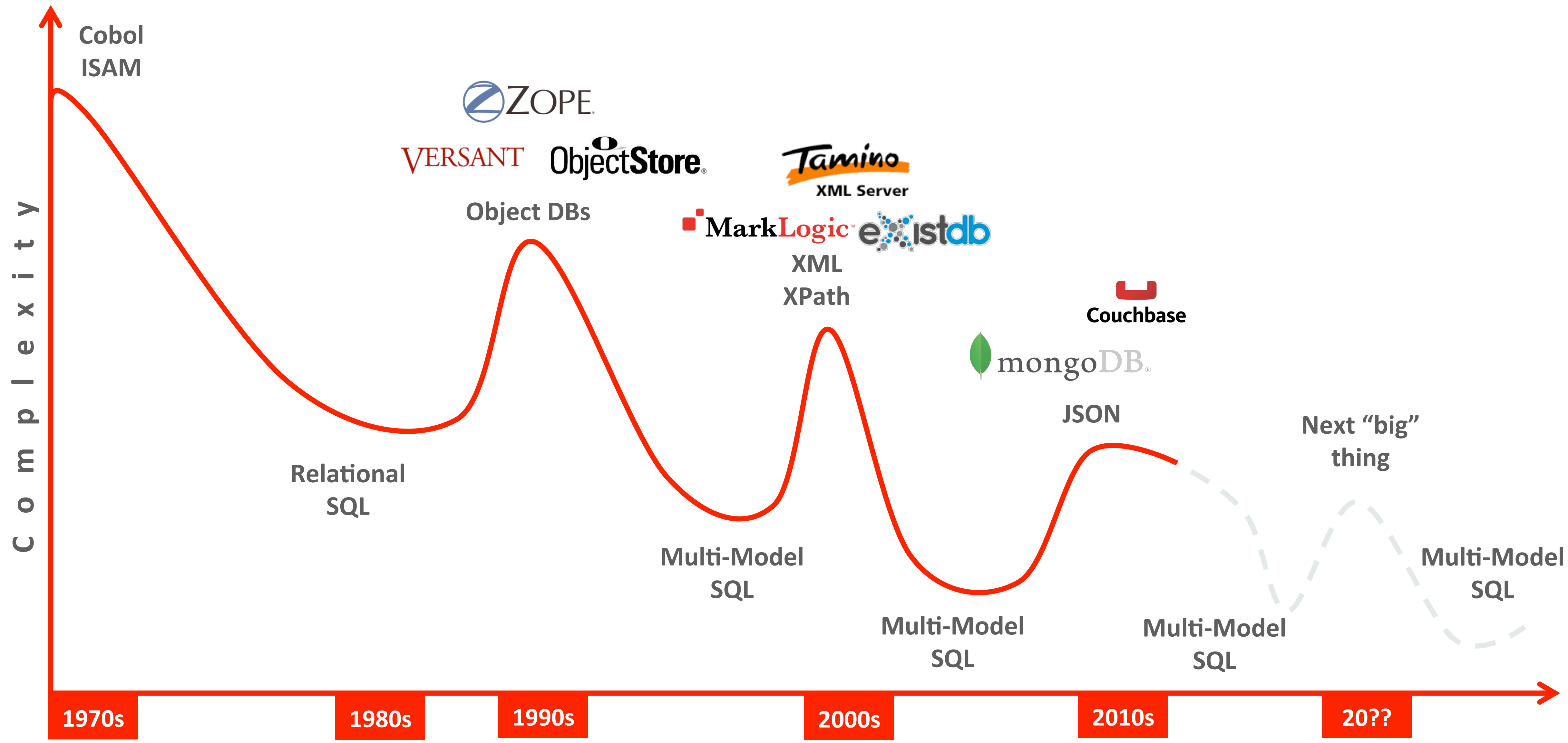
Evolution of data management



Evolution of data management

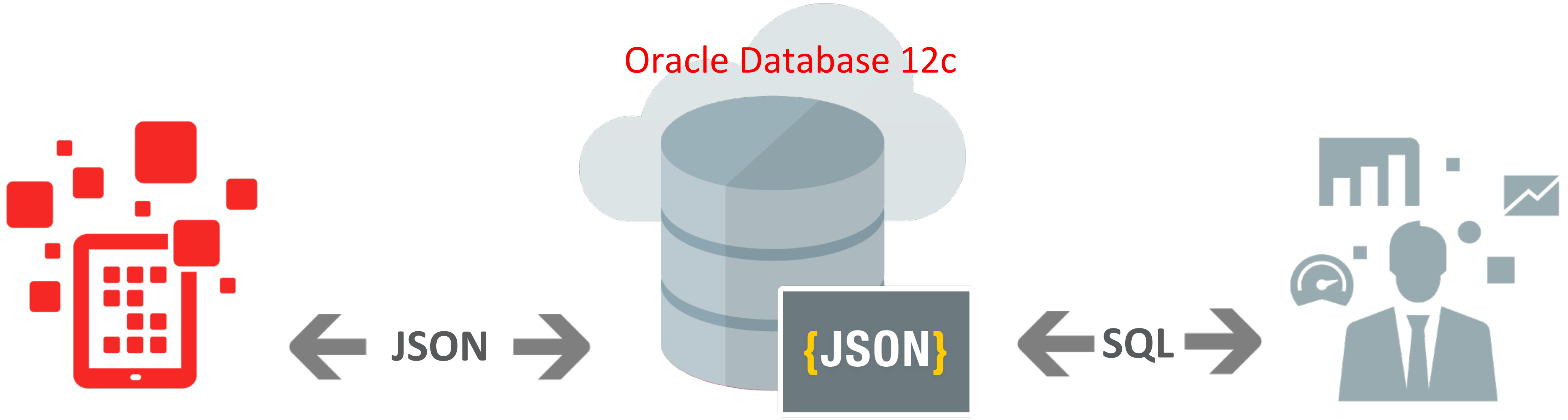


Multi-model prevails over time



Oracle 12c JSON document store

Simple NoSQL Development experience



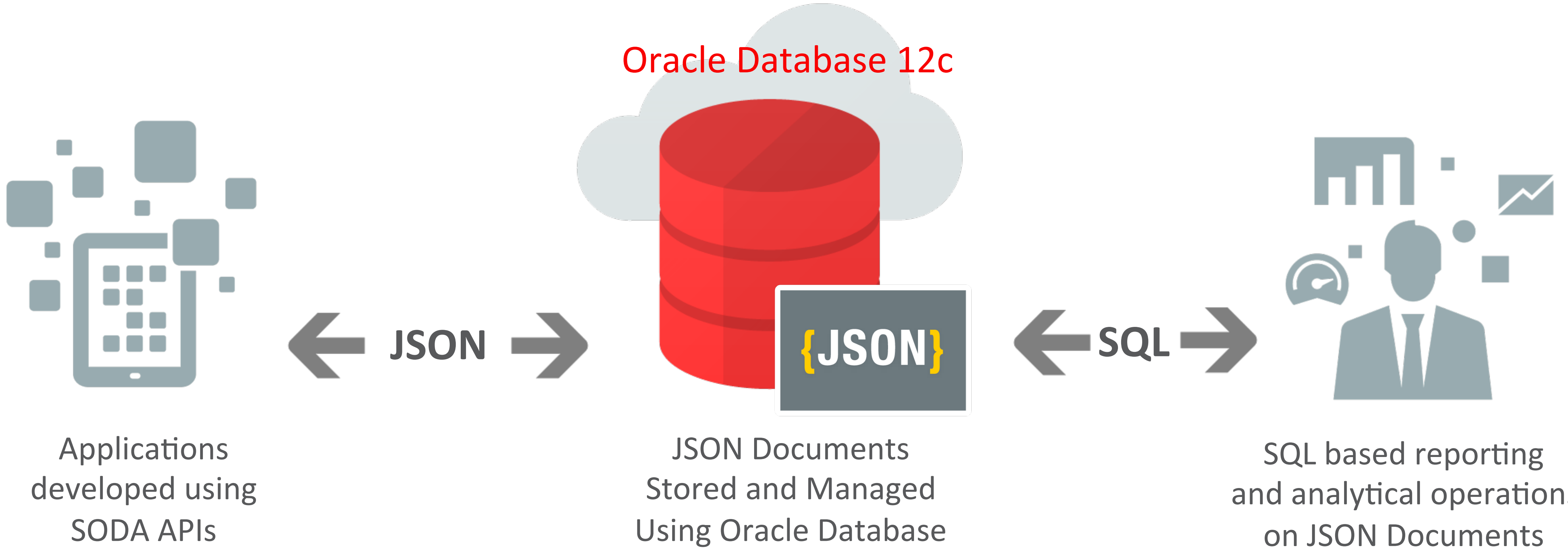
Applications developed using SODA APIs

JSON Documents Stored and Managed Using Oracle Database

SQL based reporting and analytical operations on JSON Documents

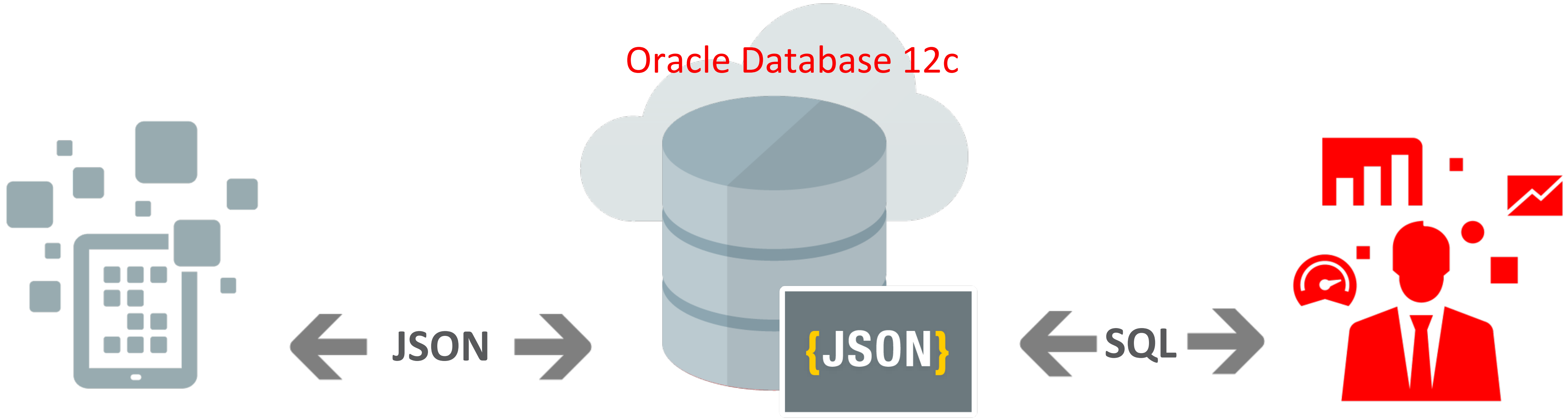
Oracle 12c JSON document store

Enterprise Data Management


Oracle Database 12c
JSON
{JSON}
SQL
Applications developed using SODA APIs
JSON Documents Stored and Managed Using Oracle Database
SQL based reporting and analytical operations on JSON Documents

Oracle 12c JSON document store

All the power of SQL when needed



Applications developed using SODA APIs

JSON Documents Stored and Managed Using Oracle Database

SQL based reporting and analytical operations on JSON Documents

JSON Support in Oracle Database

Fast Application Development + Powerful SQL Access

Application developers:
Access JSON documents using REST API

```
POST /my_database/my_schema/customers HTTP/1.0
Content-Type: application/json
Body:
{
 "firstName": "John",
 “lastName”: "Smith",
 "age": 25,
 "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021“,
      "isBusiness" : false  },
  "phoneNumbers": [
      {"type": "home",
       "number": "212 555-1234“ },
      {"type": "fax",
       "number": "646 555-4567“ }  ]
}
```

Analytical tools and business users:
Query JSON using SQL

```
select
  c.json_document.firstName,
  c.json_document.lastName,
  c.json_document.address.city
from customers c;


firstName     lastName      address.city
-----------   -----------   --------------
“John”        “Smith”       “New York”
```

JSON integration with PL/SQL

- New PL/SQL objects enable fine grained manipulation of JSON content
  - JSON_OBJECT_T : for working with JSON objects
  - JSON_ARRAY_T : for working with JSON Arrays
  - JSON_OBJECT_T and JSON_ARRAY_T are subtypes of JSON_ELEMENT_T
- These objects provide a set of methods for manipulating JSON
- Piecewise updates of JSON documents now supported in PL/SQL

JSON integration with PL/SQL

```
WITH FUNCTION updateTax(JSON_DOC in VARCHAR2 ) RETURN VARCHAR2 IS
    jo JSON_OBJECT_T;
    price NUMBER;
    taxRate NUMBER;
BEGIN
    jo := JSON_OBJECT_T(JSON_DOC);
    taxRate := jo.get_Number('taxRate');
    price := jo.get_Number('total');
    jo.put('totalIncludingTax', price * (1+taxRate));
    RETURN jo.to_string();
END;
ORDERS AS (
    SELECT '{"taxRate":0.175,"total":10.00}' JSON_DOCUMENT
      FROM dual
)
SELECT JSON_DOCUMENT, updateTax(JSON_DOCUMENT)
  FROM ORDERS;

JSON_DOCUMENT                   UPDATETAX(JSON_DOCUMENT)
------------------------------- -----------------------------------------------------------
{"taxRate":0.175,"total":10.00} {"taxRate":0.175,"total":10.00,"totalIncludingTax":11.75}
```

Data Guide: Understanding your JSON documents

- Metadata discovery: discovers the structure of collection of JSON documents
  - Optional: deep analysis of JSON for List of Values, ranges, sizing etc.
- Automatically Generates
  - Virtual columns
  - Relational views
    - De-normalized relational views for arrays
  - Reports/Synopsis of JSON structure

Data Guide: Automatic Schema Inference

Table containing JSON documents

```
SQL> desc MOVIE_TICKETS
NAME                      TYPE
----------------------    -----------
BOOKING_ID                RAW(16)
BOOKING_TIME              TIMESTAMP(6)
BOOKING_DETAILS           VARCHAR2(4000)
```

```
{
  "Theater":"AMC 15",
  "Movie":"Jurrasic World 3D",
  "Time":2015-11-26T18:45:00",
  "Tickets":{
     "Adults":2
   }
}
```

JSON DataGuide

```
DBMS_JSON.AddVC(
   "MOVIE_TICKETS",
   "BOOKING_DETAILS");
```

Table enhanced with virtual columns

```
SQL> desc MOVIE_TICKETS
NAME                         TYPE
----------------------       -----------
BOOKING_ID                   RAW(16)
BOOKING_TIME                 TIMESTAMP(6)
BOOKING_DETAILS              VARCHAR2(4000)
BOOKING_DETAILS$Movie        VARCHAR2(16)
BOOKING_DETAILS$Theater      VARCHAR2(16)
BOOKING_DETAILS$Adults       NUMBER
BOOKING_DETAILS$Time         VARCHAR2(32)
```

JSON Search Index : A universal index for JSON content

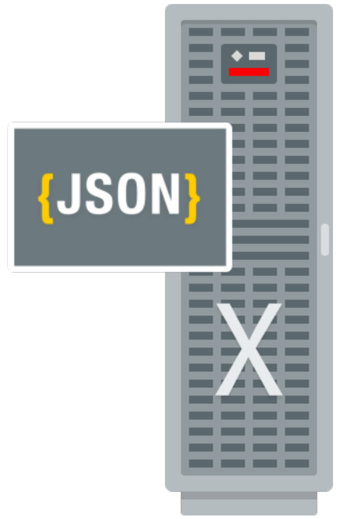
```
CREATE SEARCH INDEX JSON_SEARCH_INDEX
  ON J_PURCHASEORDER (PO_DOCUMENT) FOR JSON;
```

- Supports searching on JSON using key, path and value
- Supports range searches on numeric values
- Supports full text searches:
  - Full boolean search capabilities (and, or, and not)
  - Phrase search, proximity search and "within field" searches.
  - Inexact queries: fuzzy match, soundex and name search.
  - Automatic linguistic stemming for 32 languages
  - A full, integrated ISO thesaurus framework

Query Optimizations for JSON

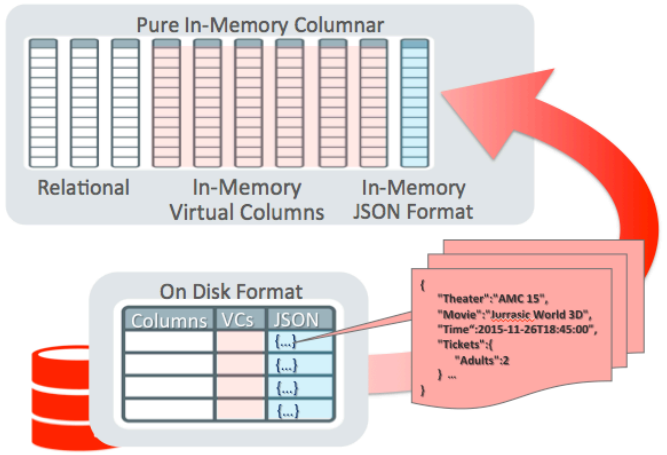
Exadata Smart Scans

- Exadata Smart Scans execute portions of SQL queries on Exadata storage cells
- JSON query operations 'pushed down' to Exadata storage cells
  - Massively parallel processing of JSON documents



In-Memory Columnar Store

- Virtual columns, included those generated using JSON Data Guide loaded into In-Memory Virtual Columns
- JSON documents loaded using a highly optimized In-Memory binary format
- Query operations on JSON content automatically directed to In-Memory



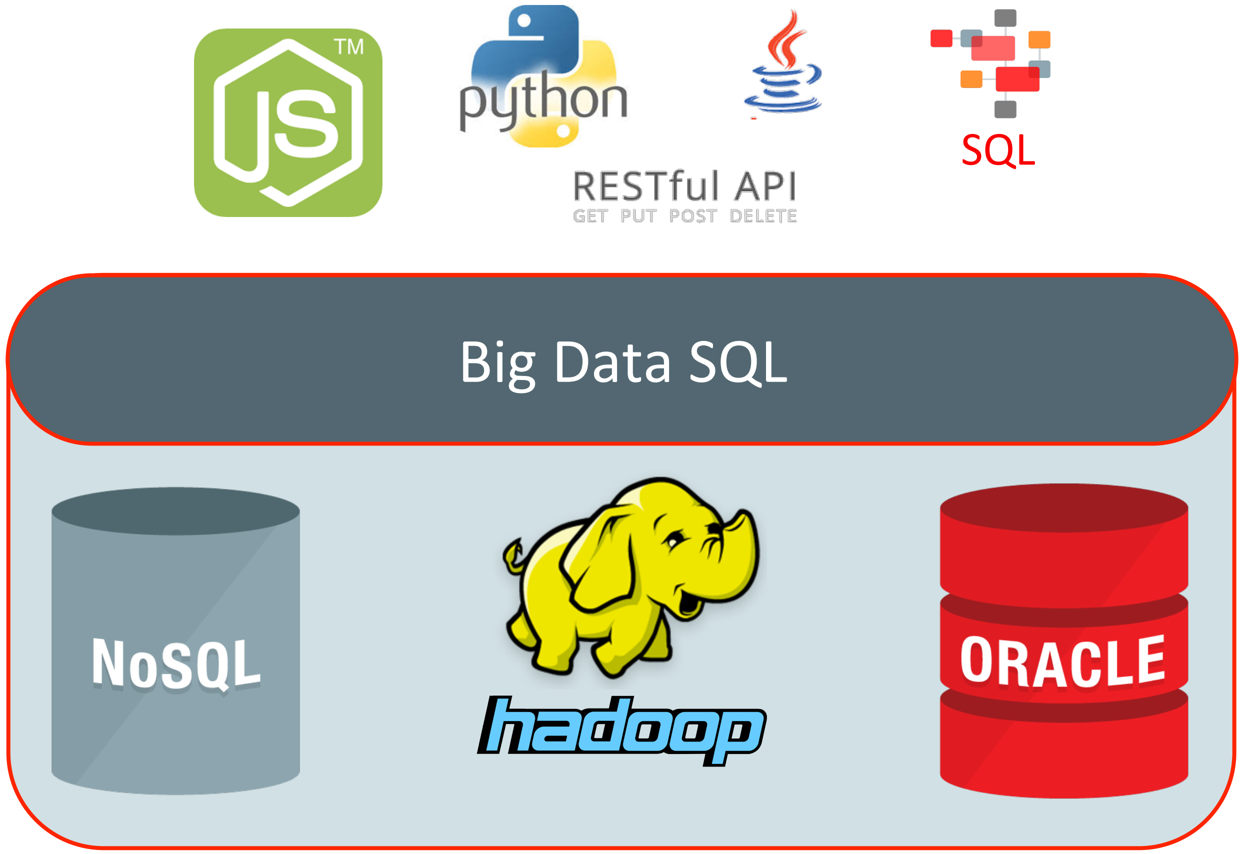
Native JSON Generation

```
SQL> SELECT JSON_OBJECT('Id' is EMPLOYEE_ID, 'FirstName' is FIRST_NAME,
  2                     'LastName' is LAST_NAME) JSON
  3        FROM HR.EMPLOYEES
  4          WHERE EMPLOYEE_ID = 100;

JSON
------------------------------------------------------------------------------
{ "Id" : 100 , "FirstName" : "Steven" , "LastName" : "King" }

SQL>
```

- JSON generation functions available:
  - JSON_OBJECT / JSON_OBJECTAGG
  - JSON_ARRAY / JSON _ARRAYAGG

Program Agenda

The Best of Both Worlds



The Best of Both Worlds

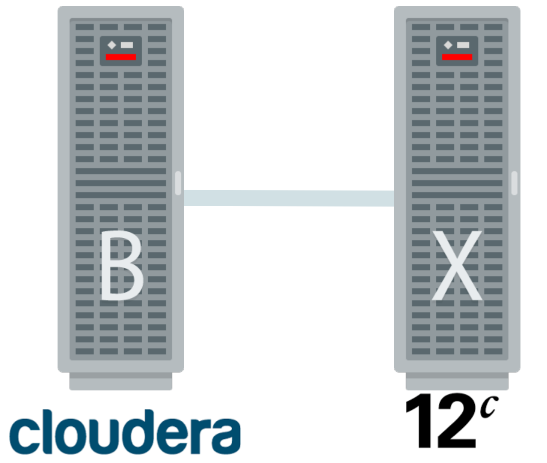
➕

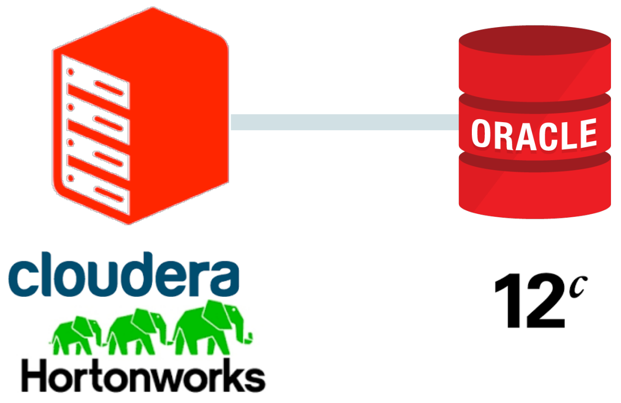
- Simplicity
- Specialization
- Performance

➖

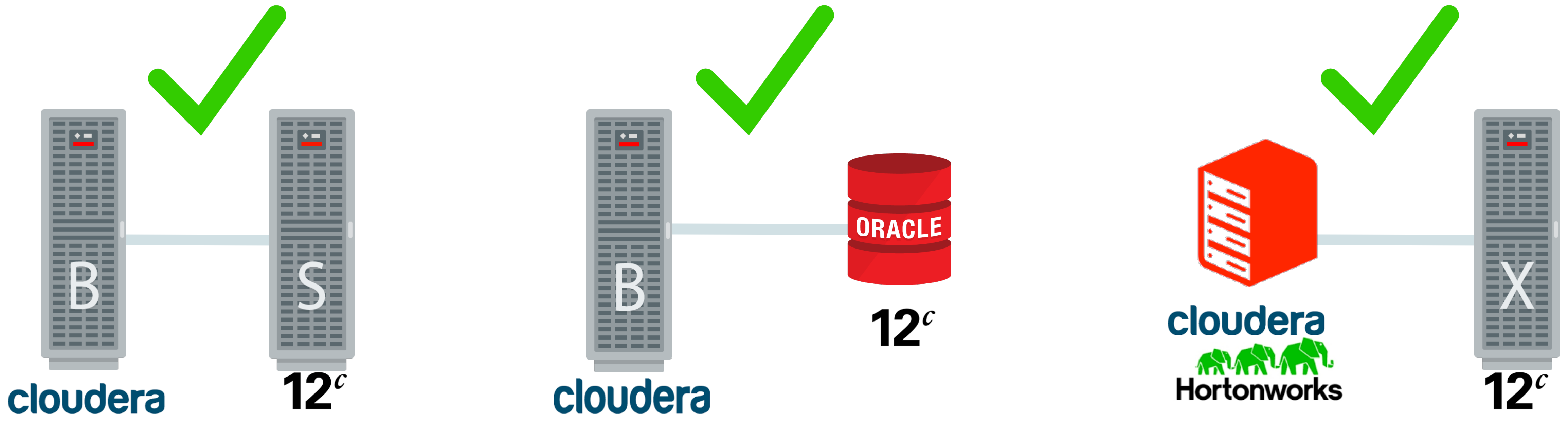
- Complexity
- Fragmentation
- Delays

Yesterday’s On-Premises Deployment Models





Today More Deployment Options for Big Data SQL



Program Agenda

Case-insensitive Database and Column-level Collation

Greatly simplifies migration of case-insensitive functionality of 3rd-party products

```
CREATE TABLE product
( id        NUMBER,
  name      VARCHAR2(50) COLLATE BINARY_CI,
  comments  VARCHAR2(500)
) DEFAULT COLLATION BINARY;

SELECT name, comments FROM product
 WHERE name LIKE '%BASE%' OR
    comments COLLATE BINARY_CI LIKE '%REPORT%';

NAME                         COMMENTS
Oracle Database
Activity-Based Management
Business Intelligence        Replaces Reports
```

_CI = case-insensitive

Inherits BINARY

- Linguistic-sensitive operations, e.g., comparison and sorting, on the column honor the declared collation
- Unspecified column collation is inherited from the default collation property of the parent table or schema
- COLLATE operator can be used to cast an explicit collation anywhere in an expression

Approximate Query Processing

Not every query requires a completely accurate result

- 12.1.0.2 APPROX_COUNT_DISTINCT
- 12.2.0.1 adds:
- APPROX_PERCENTILE
- APPROX_MEDIAN
  - Find the value for a given percentile, e.g. what is the amount sold that represents the 90% percentile of all sales
  - 6-13X faster with error typically < 1%

- Approximate functions used without any application changes
  - Queries automatically re-written to use approximate functions
  - approx_for_aggregation = TRUE

- Store approximate aggregates in materialized views with query rewrite
  - Not previously possible to use MV's with distinct and percentile aggregates

Property Graph Support

- Massively-Scalable Graph Database
  - Scales to **trillions** of edges
- Memory-based Graph Analytics
  - More than 35 graph analysis algorithms
- Simple Standard interfaces
  - SQL, Java
  - Tinkerpop: Blueprints, Gremlin, Rexster
  - Groovy, Python

PL/SQL deprecate pragma

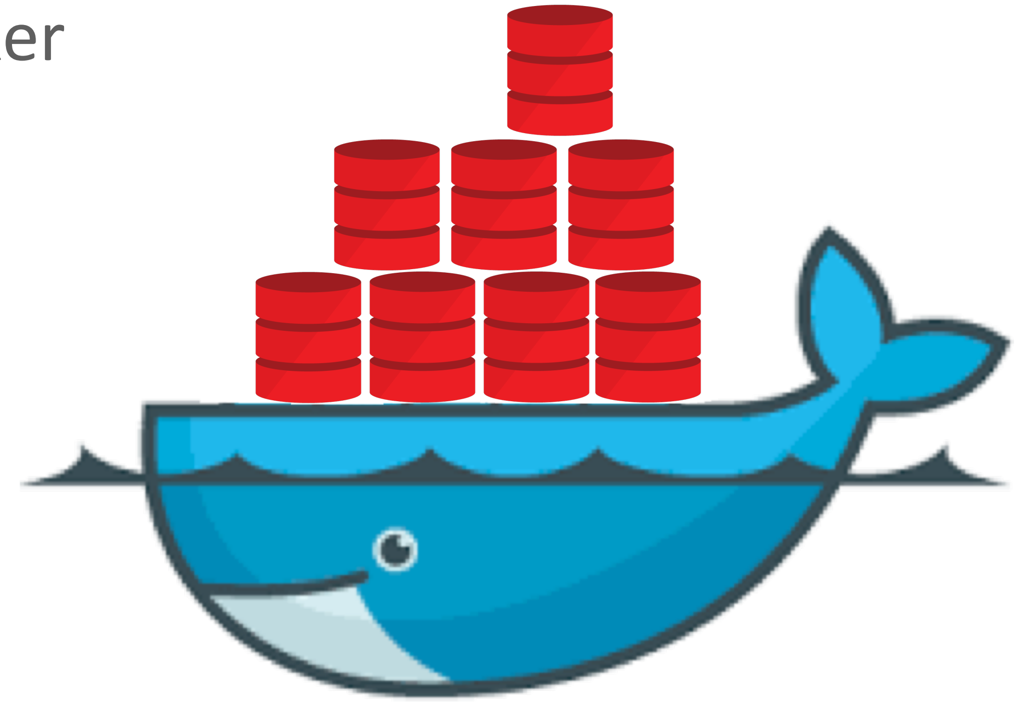
```
create procedure p authid Definer is
  pragma deprecate(p, 'p is deprecated. You must use p2 instead.');
begin
  DBMS_Output.Put_Line('p');
end p;

PLW-06019: entity P is deprecated

create procedure q authid Definer is
begin
  p();
  DBMS_Output.Put_Line('q');
end q;

PLW-06020:
reference to a deprecated entity: p is deprecated. You must use p2 instead.
```

128-byte identifiers for objects

```
CREATE TABLE VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG
(
  VERY_VERY_LONG_TEXT_COLUMN_WITH_DATA_TYPE_VARCHAR2_THAT_IS_72_BYTES_LONG VARCHAR2(25)
);

Table VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG created.

INSERT INTO VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG
   VALUES ('Hello OOW attendees!');

1 row inserted.

SELECT * FROM VERY_VERY_LONG_TABLE_NAME_IDENTIFIER_THAT_IS_58_BYTES_LONG;

VERY_VERY_LONG_TEXT_COLUM
-------------------------
Hello OOW attendees!
```

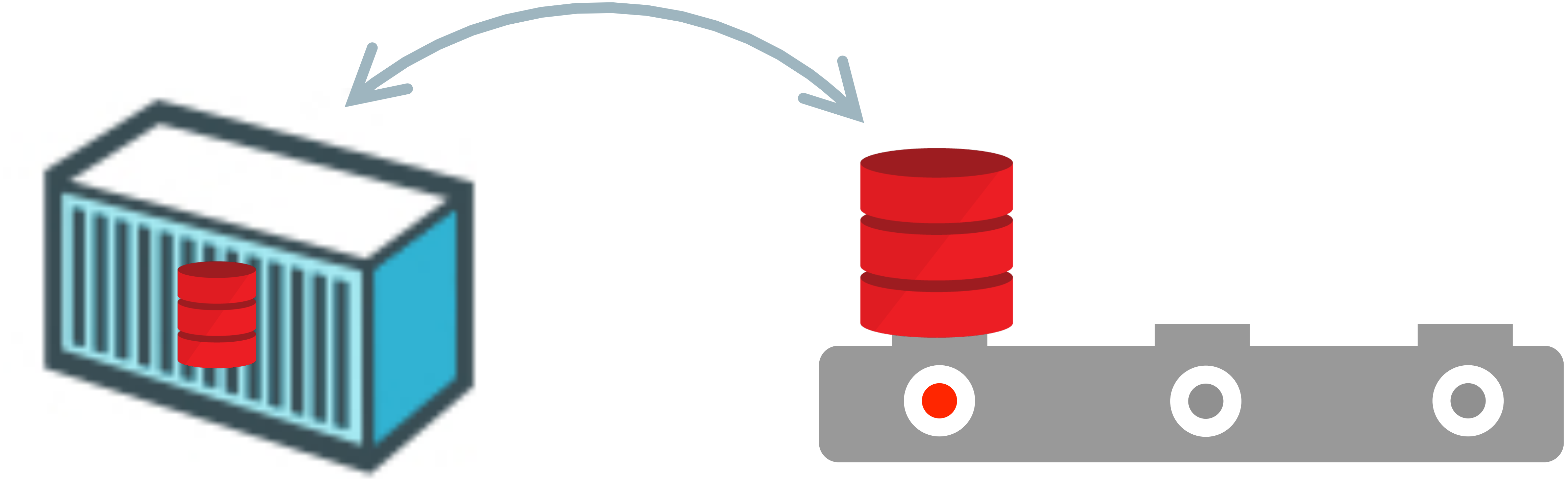
Oracle on Docker

- Oracle Database is fully supported on Docker
  - Oracle Linux 7
  - Red Hat Enterprise Linux 7
- Oracle image on Docker Store
- Docker build files on GitHub



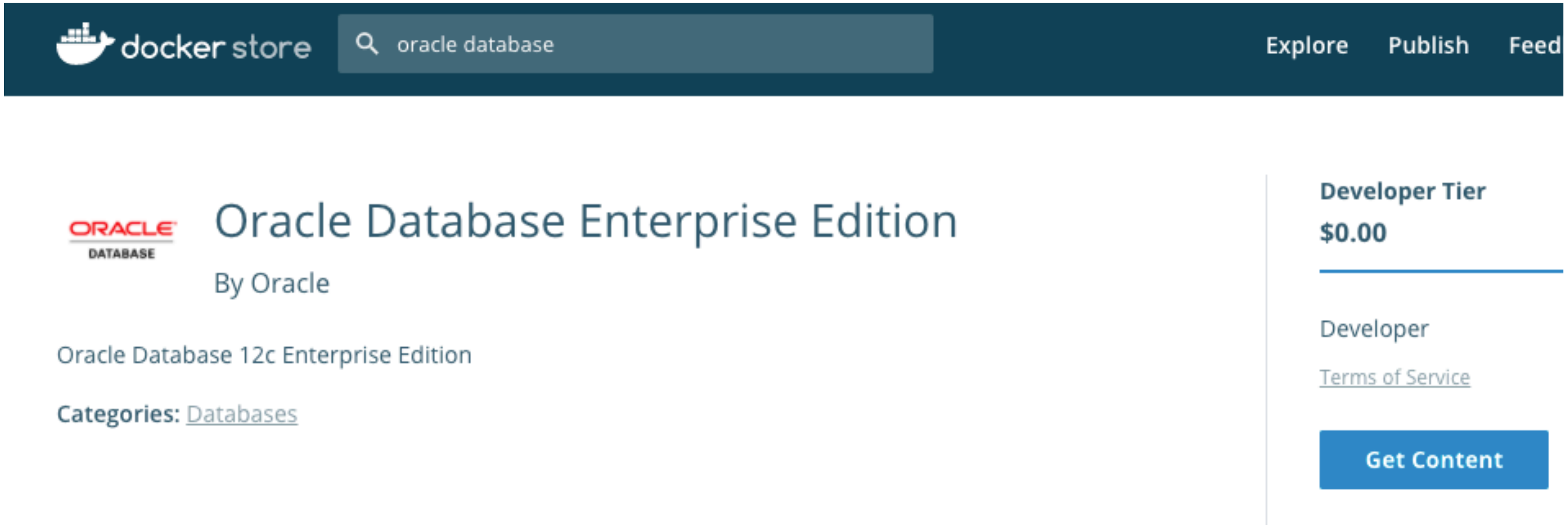
Oracle on Docker

- Docker container contains single-PDB CDB
- PDB can be plugged, unplugged, etc.
- PDB can move bi-directional



Docker Store

- Oracle 12.1.0.2 images are available on Docker Store Registry
  - https://store.docker.com
  - 12.2.0.1 coming soon (currently going through testing)



Oracle Database Enterprise Edition

By Oracle

Oracle Database 12c Enterprise Edition

Categories: Databases

Developer Tier

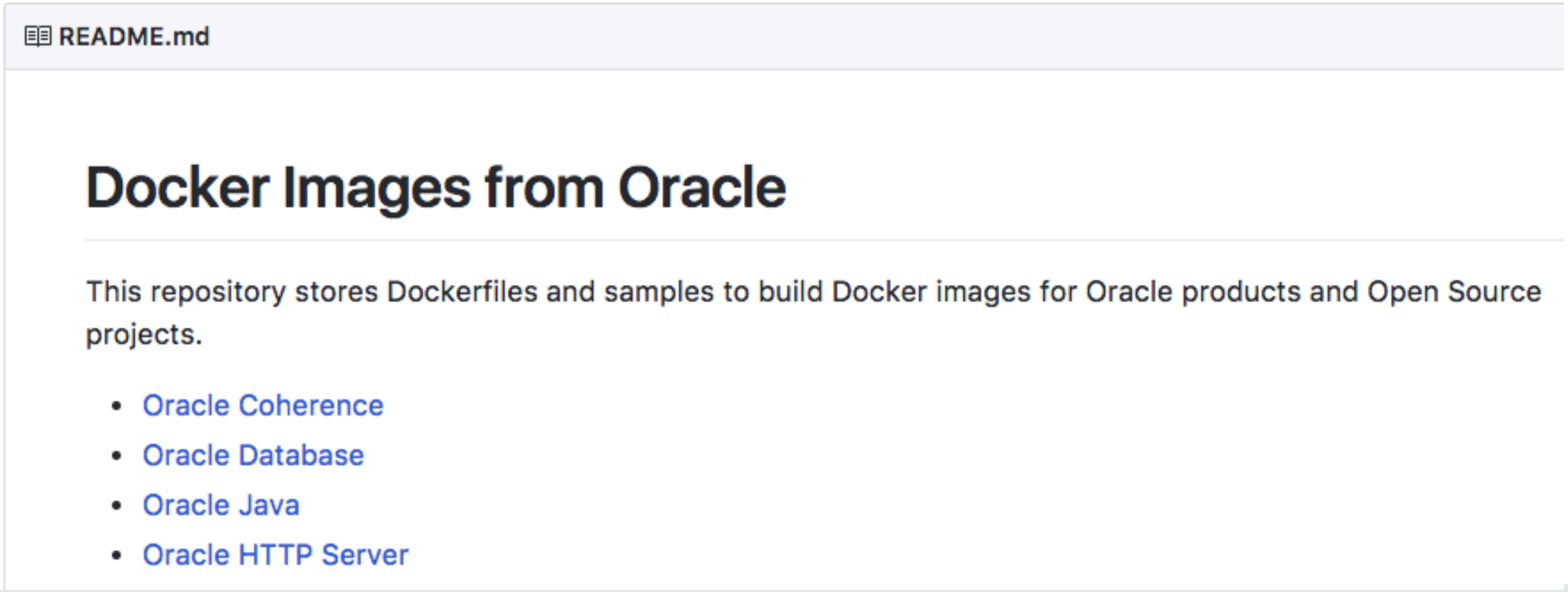
$0.00

Developer

Terms of Service

Docker build files available on GitHub

- Repository: https://github.com/oracle/docker-images
- Build files for 12.2.0.1 EE/SE2, 12.1.0.2 EE/SE2, 11.2.0.2 XE



README.md

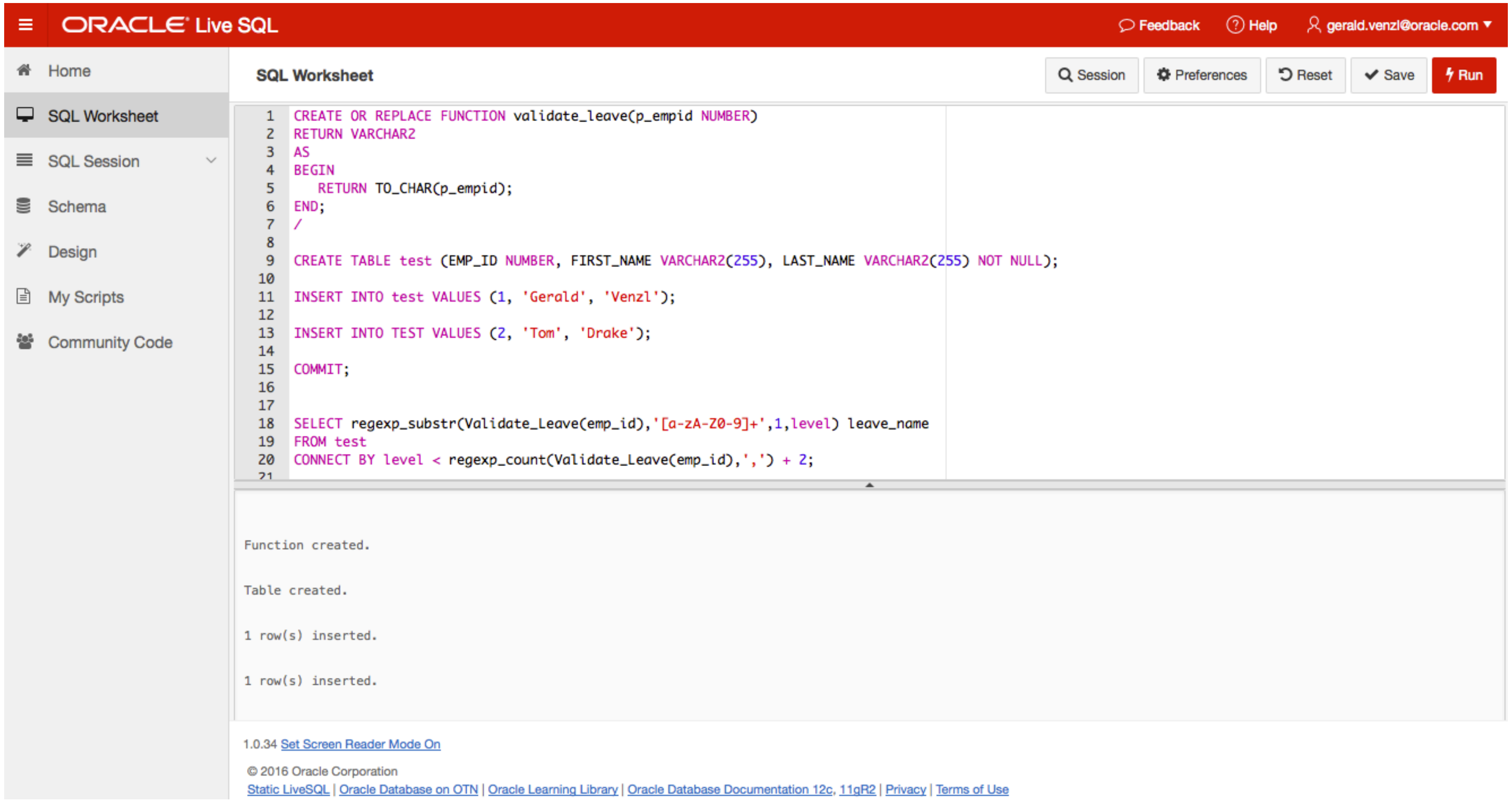
Docker Images from Oracle

This repository stores Dockerfiles and samples to build Docker images for Oracle products and Open Source projects.

- Oracle Coherence
- Oracle Database
- Oracle Java
- Oracle HTTP Server

LiveSQL.oracle.com

The full power of Oracle SQL in your browser



```
CREATE OR REPLACE FUNCTION validate_leave(p_empid NUMBER)
RETURN VARCHAR2
AS
BEGIN
   RETURN TO_CHAR(p_empid);
END;
/

CREATE TABLE test (EMP_ID NUMBER, FIRST_NAME VARCHAR2(255), LAST_NAME VARCHAR2(255) NOT NULL);

INSERT INTO test VALUES (1, 'Gerald', 'Venzl');

INSERT INTO TEST VALUES (2, 'Tom', 'Drake');

COMMIT;


SELECT regexp_substr(Validate_Leave(emp_id),'[a-zA-Z0-9]+',1,level) leave_name
FROM test
CONNECT BY level < regexp_count(Validate_Leave(emp_id),',') + 2;
```

```
Function created.

Table created.

1 row(s) inserted.

1 row(s) inserted.
```

1.0.34 Set Screen Reader Mode On

© 2016 Oracle Corporation
Static LiveSQL | Oracle Database on OTN | Oracle Learning Library | Oracle Database Documentation 12c, 11gR2 | Privacy | Terms of Use

Thank you!

https://developer.oracle.com

Safe Harbor Statement

Integrated Cloud

Applications & Platform Services

ORACLE®