
ORACLE®

Zero Downtime: Hiding Planned Maintenance and Unplanned Outages from Applications

Carol Colrain
Consulting Member of Technical Staff,
Technical Lead for Client-Failover, RAC Development

ORACLE DATABASE

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.

Program Agenda

1	Problems to Solve	10
2	Fast Application Notification	15
3	Continuous Connections	15
4	Hiding Planned Maintenance	15
5	Hiding Unplanned Outages	30
6	Success Stories	

1 What problems confront applications at database outages?

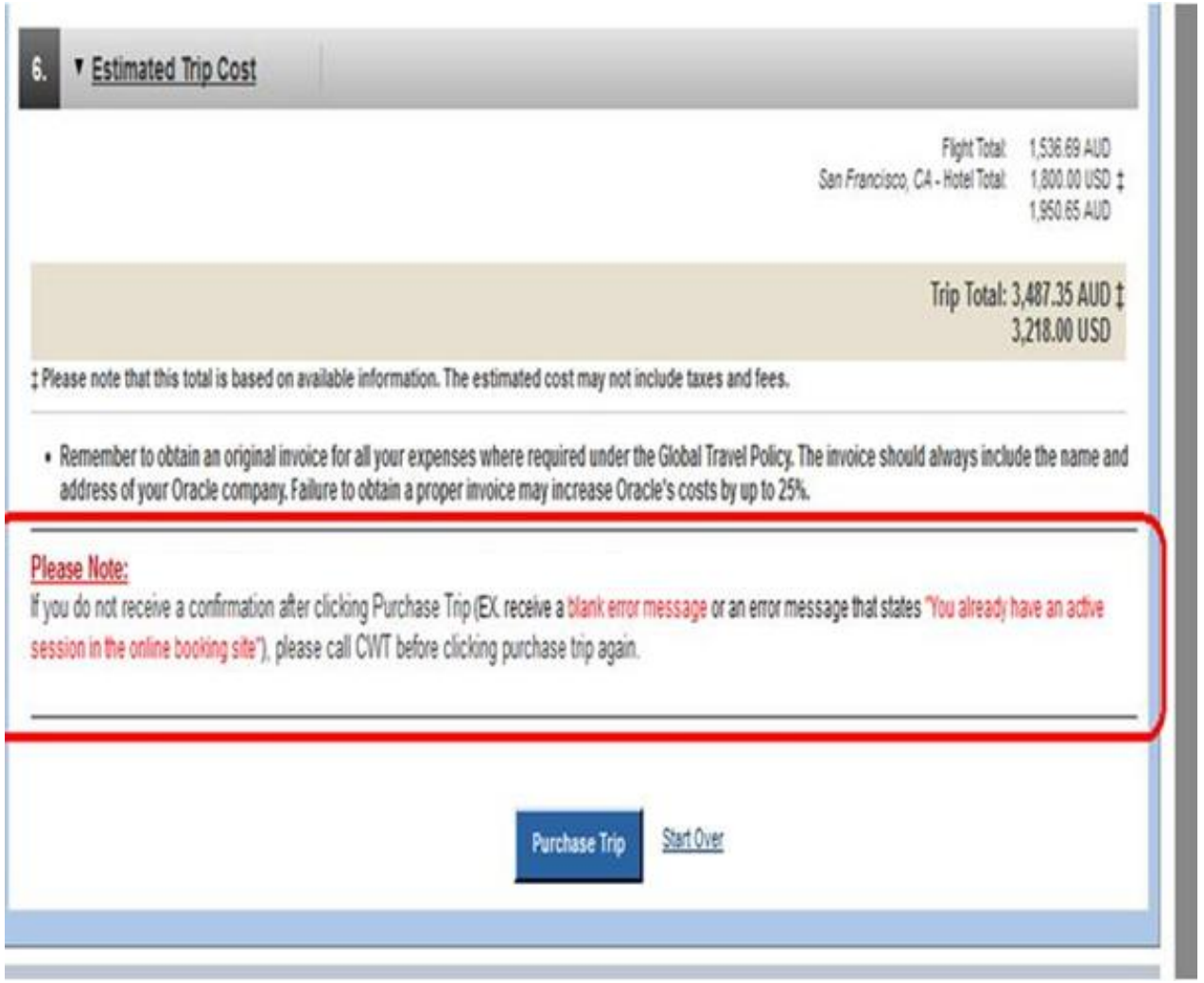
In-Flight Work

Sorry. Internal Server Error - 500 Error
We are currently experiencing an issue with our servers on coolcar.com. Please come back later.

Pre-12c Situation

Database outages cause in-flight work to be lost, leaving users and applications in-doubt

- Restart applications and mid-tiers
- User frustration
- Cancelled work
- Duplicate submissions
- Errors even when planned
- Developer pains



How do we reach all applications?

- Move work to different instance/database with no errors reported to applications at planned maintenance
- Hide unplanned database outages from the applications
- Take adoption out of the developers hands to configuration/operation only
- Work with current drivers and older database, whenever possible

2 Outage Detection

The dead thing cannot tell you that it's dead

Applications Waste Time

- Hanging on TCP/IP timeouts
- Connecting when services are down
- Not connecting when services resume
- Receiving errors during planned maintenance
- Processing partial results when server is down
- Attempting work at slow, hung, or dead nodes

Fast Application Notification

Proven since 10g

- **Down** – received in low ms to invoke failover
- **Planned Down** – drains sessions for planned maintenance with no user interruption whatsoever
- **Up** – Re-allocates sessions when services resume
- **Load %** - Advice to balance sessions for RAC locally and GDS globally
- **Affinity** - Advice when to keep conversation locality

12c: Auto-Configuration + Global Data Services

12c FAN: Standardized, Auto-Configured

Client	10g	11g	12c
JDBC Implicit Connection Cache	ONS	ONS	desupport
JDBC Universal Connection Pool		ONS	**ONS**
OCI/OCCI driver	AQ	AQ	**ONS**
ODP.NET Unmanaged Provider (OCI)	AQ	AQ	**ONS**
ODP.NET Managed Provider (C#)		ONS	**ONS**
OCI Session Pool	AQ	AQ	**ONS**
WebLogic Active GridLink		ONS	**ONS**
Tuxedo		ONS	**ONS**
Listener	ONS	ONS	**ONS**

12c JDBC FAN Auto-Configures

- **12c JDBC clients and 12c Oracle database**
  - Check ons.jar is included in the class path
  - To enable FAN set the pool property
    - **fastConnectionFailoverEnabled=true**

- **Before 12c - JDBC clients or database**
  - also set the pool property for remote ons
    - oracle.ons.nodes **=mysun05:6200,mysun06:6200, mysun07:6200,mysun08:6200**

or via autoons

- oracle.ons.nodes.001=node1a,node1b,node1c... (site 1 nodes here)
  oracle.ons.nodes.002=node2a,node2b,node2c... (site 2 nodes here)

12c OCI FAN Auto-Configures

- **12c OCI clients and 12c Oracle database**

  Use srvctl to configure the service for AQ HA Notification:

  srvctl modify service -db EM -service GOLD **-notification TRUE**

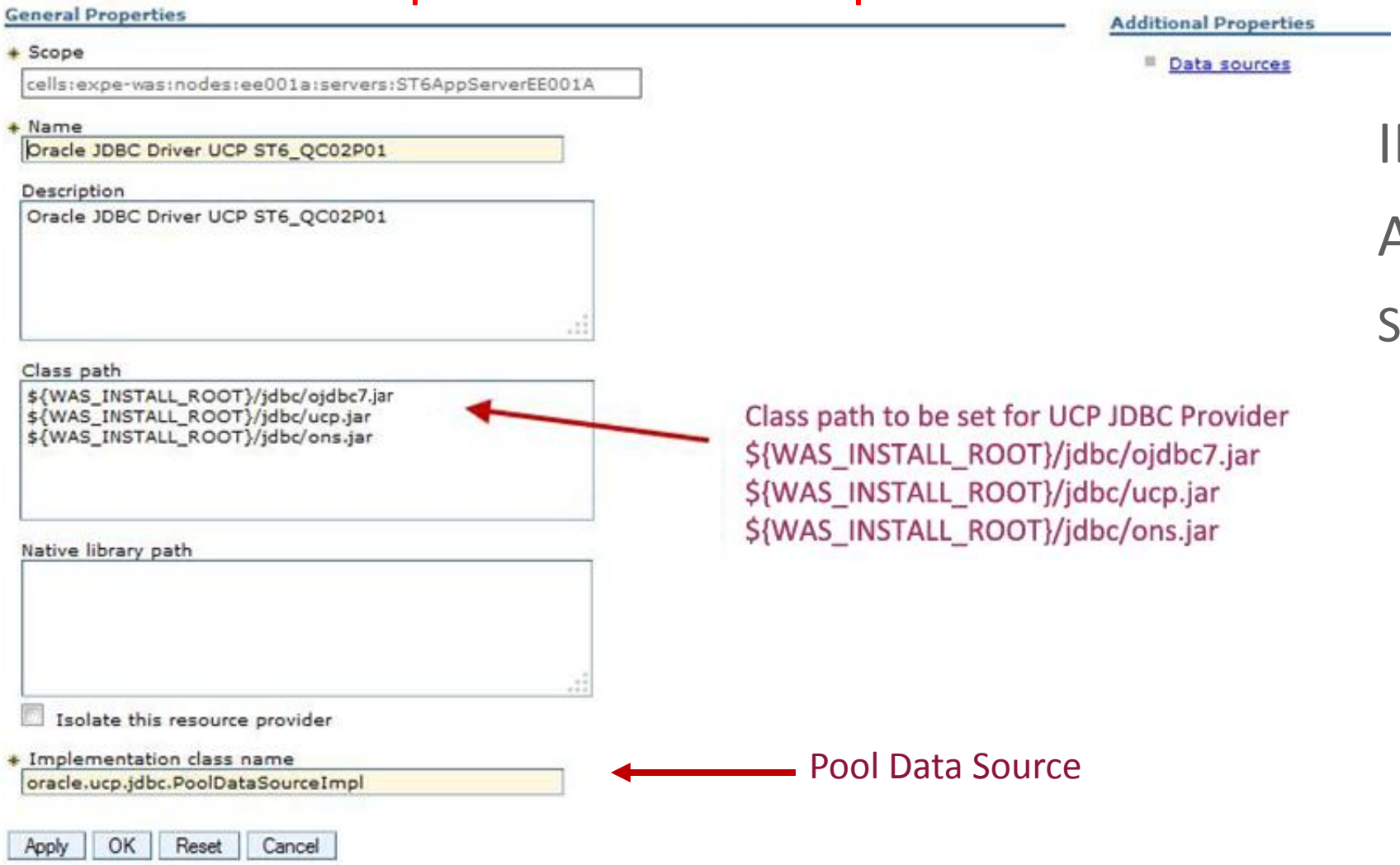
  For the client, enable in oraaccess.xml

```
<oraaccess>
  <default_parameters>
    <events>true</events>
  </default_parameters>
</oraaccess>
```

- **Before 12c OCI clients or database**
  - Enable OCI_EVENTS at environment creation OCIEnvCreate(..)
  - Link the app with the client thread o/s library.

FAN with other Java Application Servers

Use UCP – a simple DataSource replacement



Class path to be set for UCP JDBC Provider
${WAS_INSTALL_ROOT}/jdbc/ojdbc7.jar
${WAS_INSTALL_ROOT}/jdbc/ucp.jar
${WAS_INSTALL_ROOT}/jdbc/ons.jar

Pool Data Source

IBM WebSphere

Apache Tomcat

See OTN.

Monitor FAN

- **Create a FAN callout in ..$GRID_HOME/racg/userco**
- **Download FANwatcher from OTN RAC page**

FANwatcher

..

VERSION=1.0 **event_type=SERVICEMEMBER** service=orcl_swing_pdb2 instance=orcl1 database=orcl db_domain= host=sun01 **status=down reason=USER** timestamp=2014-07-30 12:02:51 timezone=-07:00

VERSION=1.0 event_type=**SERVICEMEMBER** service=orcl_swing_pdb10 instance=orcl1 database=orcl db_domain= host=sun01 **status=down reason=USER** timestamp=2014-07-30 12:02:52 timezone=-07:00

VERSION=1.0 event_type=SERVICE service=orcl_swing_pdb10 database=orcl db_domain= host=sun01 status=down **reason=USER**

21

Continuous Connections

Applications should see no errors while services relocate.

Connections Appear Continuous **for OCI - 12102**

while a service is temporarily unavailable

Safe for failover + storms

Retry while service is unavailable

New

```
alias =(DESCRIPTION =
    (CONNECT_TIMEOUT=90) (RETRY_COUNT=20)(RETRY_DELAY=3)
        (TRANSPORT_CONNECT_TIMEOUT=3)
     (ADDRESS_LIST =
         (LOAD_BALANCE=on)
         ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
     (ADDRESS_LIST =
         (LOAD_BALANCE=on)
         ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))
    (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

OCI Only

Balance scan

Connections Appear Continuous **for Java - 12102**

while a service is temporarily unavailable

Lower skips down IPs but can cause timeouts on failover and storms

```
(DESCRIPTION =
   (CONNECT_TIMEOUT= 4)  (RETRY_COUNT=20)(RETRY_DELAY=3)
    (ADDRESS_LIST =
        (LOAD_BALANCE=on)
        ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
    (ADDRESS_LIST =
        (LOAD_BALANCE=on)
        ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))
   (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

Balance scan

Connections Appear Continuous **for ODP.NET - 12102**

while a service is temporarily unavailable

Increase ODP.NET "connection timeout" connection attribute for failover to complete – e.g. 90s to accommodate login storms.

Safe for failover + storms

```
alias =(DESCRIPTION =
    (TRANSPORT_CONNECT_TIMEOUT=3) (RETRY_COUNT=20)(RETRY_DELAY=3)
     (ADDRESS_LIST =
          (LOAD_BALANCE=on)
          ( ADDRESS = (PROTOCOL = TCP)(HOST=primary-scan)(PORT=1521)))
     (ADDRESS_LIST =
          (LOAD_BALANCE=on)
          ( ADDRESS = (PROTOCOL = TCP)(HOST=secondary-scan)(PORT=1521)))
    (CONNECT_DATA=(SERVICE_NAME = gold-cloud)))
```

Lessons Learned – New Connections

- **ALWAYS use application services to connect to the database.**
  - Do not use the database service or PDB service – these are for administration only, not HA
- Use current client driver (12102) with current or older RDBMS
- Use one DESCRIPTION – more cause long delays connecting
- Set CONNECT_TIMEOUT=90 or higher to prevent logon storms (OCI and ODP)
  - Do not also set JDBC property oracle.net.ns.SQLnetDef.TCP_CONNTIMEOUT_STR as it overrides
- LOAD_BALANCE=on per address list balances SCANs
- Do not use retry count without retry delay
- **Do not use Easy*Connect – it has no HA capabilities.**

Patches before 12.2

For Java Net Connections only:

- RETRY_COUNT must apply when service is down (19154304)
  - PSE 21439688 on 12.1.3.1 WebLogic Server
- Set LOAD_BALANCE=on per address to balance the SCAN (18057904)
- 11 Databases - NO DELAY PARAMETER FOR RETRYING INCOMING CONNECTIONS (16618074)
- TRANSPORT_CONNECT_TIMEOUT (19000803)

Transparent Planned Maintenance

Applications should see no errors during maintenance.

Transparent Planned Maintenance

- Drains work away from instances targeted for maintenance initiated by FAN
  - Supports well behaved applications using Oracle pools
    - WebLogic Active GridLink, UCP, ODP.NET unmanaged and managed, OCI Session Pool, PHP
    - 3rd party application servers using UCP DataSource: IBM Websphere, Apache Tomcat,..
- Failover at transactional disconnect
    - applications adapted for TAF SELECT with OCI or ODP.Net unmanaged provider
    - applications with own/custom failover

DBA steps - Drain Work at Safe Places

Repeat for each service allowing time to drain

- **Stop service (no –force)**

```
srvctl stop service –db .. -instance .. [–service] .. (omitting –service stops all)
```

- or **Relocate service (no –force)**

```
srvctl relocate service –db .. -service .. –oldinst .. –newinst
srvctl relocate service –db .. -service .. –currentnode.. –targetnode
```

- **Wait to allow sessions and XA branches to drain.** (see notes)
- **For remaining sessions, stop transactional per service**

```
exec dbms_service.disconnect_session('[service]', DBMS_SERVICE.POST_TRANSACTION);
```

- **Now stop the instances using your preferred method including opatch**
- **For major maintenance operations, disable to prevent restarts**

```
srvctl disable instance –db .. -instance
```

How it works

Applications using …	Oracle pools or drivers – WebLogic Active GridLink, UCP, ODP.NET managed/unmanaged, OCI, Tuxedo 3rd party App Servers using UCP: IBM WebSphere, Apache Tomcat
DBA Step	**srvctl [relocate|stop] service (no –force)**
Sessions Drain	Immediately New work is redirected by listeners Idle sessions are released Active sessions are released when returned to pools

FAN Planned

Pools drain sessions as work completes

Planned Maintenance at NEC

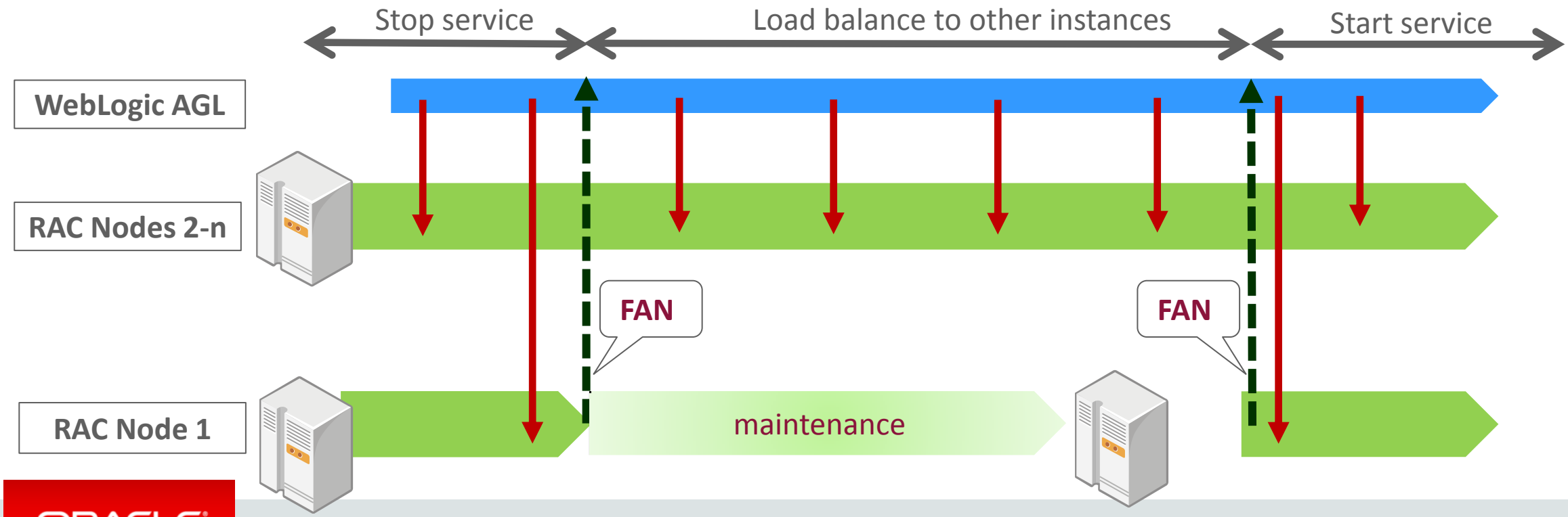
WebLogic Active GridLink and Real Application Clusters

Empowered by Innovation

NEC

1. srvctl stop services at one instance & drain **(e.g. 5-7s)**
2. Instance shutdown
3. Apply patch or change parameter or other maintenance
4. Restart instance & service

No errors, application continues



ORACLE

Planned Maintenance at NEC

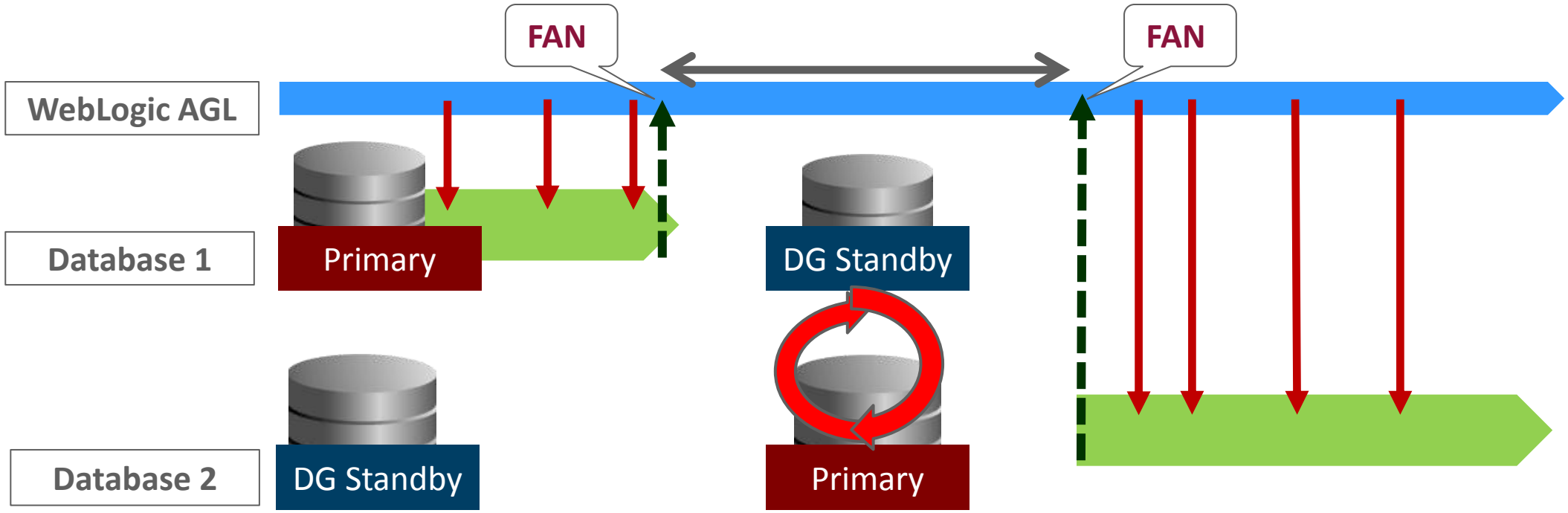
WebLogic Active GridLink and Data Guard

Empowered by Innovation

NEC

1. srvctl stop services on primary site & drain **(e.g. 25s – 30s)**
2. Data Guard switchover
3. New primary database open, start service, rebalance

No errors, application continues



High Availability by Patch Type

	One- Off	PSU/CPU	Bundle Patch	Patch Set
RAC Rolling	96%	All	Most	No
Standby First	98%	All	All	No
Out of Place	All	All	Exadata bundles	No
Online - Hot	82%*	No	No	No

** Available from 11.2.0.2 onward*

Enterprise Applications

Application	DBA operation at planned maintenance	Configuration Setting
Siebel	disconnect sessions transactional	NET
PeopleSoft		NET and TAF SELECT
JD Edwards		NET
Informatica		NET

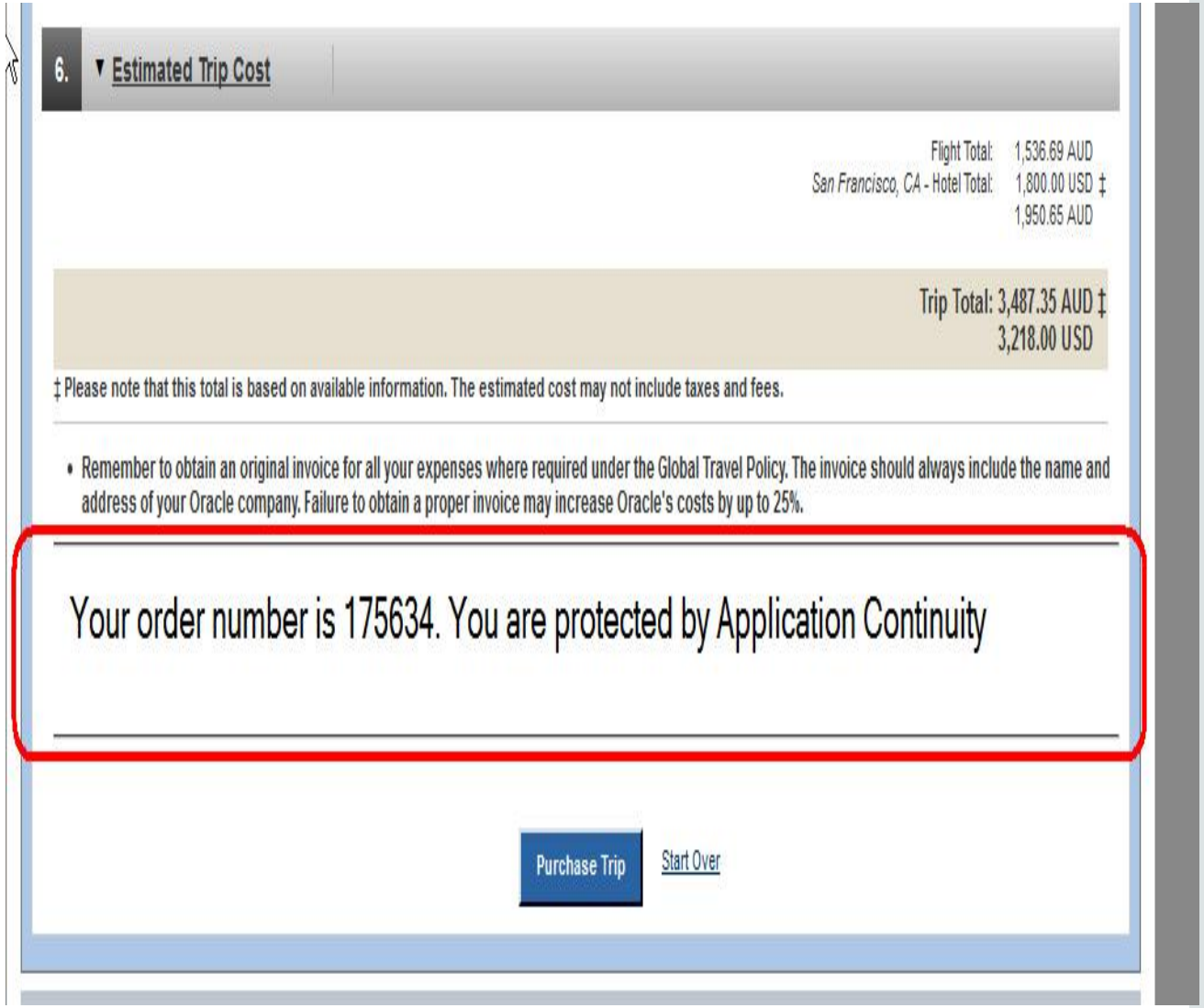
Application Continuity

Unplanned outages should be hidden from applications

Application Continuity

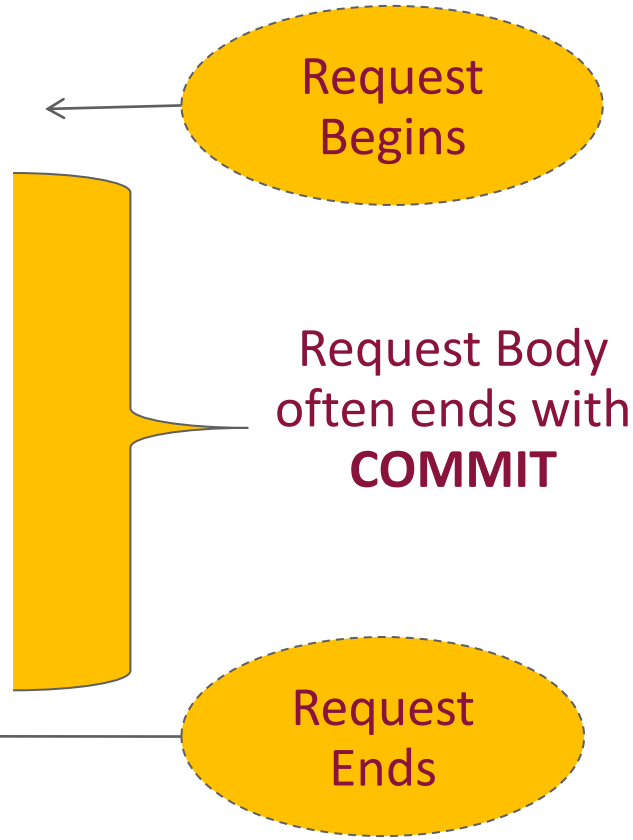
In-flight work continues

- Replays in-flight work on recoverable errors
- Masks most hardware, software, network, storage errors and outages
- Supports JDBC-Thin, UCP, WebLogic Server, 3rd Party Java app servers
- RAC, RAC One, & Active Data Guard
- Improves end user experience



Database Request – UCP example

```
PoolDataSource pds = GetPoolDataSource();
Connection conn = pds.getConnection();
PreparedStatement pstmt = …
   …
    SQL, PL/SQL, local calls, RPC
     …
conn.commit();
conn.close();
```



Phases in Application Continuity

1 – Normal Operation

- Client marks database requests
- Server decides which calls can & cannot be replayed
- Directed, client holds original calls, their inputs, and validation data

2 – Outage Phase 1: Reconnect

- Checks replay is enabled
- Verifies timeliness
- Creates a new connection
- Checks target database is valid
- Uses Transaction Guard to force last outcome

3 – Outage Phase 2: Replay

- Replays captured calls
- Ensures results returned to application match original
- On success, returns control to the application

Exclusions

When replay is not enabled

Application Level

- Default database or default PDB service
- Deprecated, non-standard JDBC classes
- XA in 12.1

Request Level

- Admin actions
  - Alter system
  - Alter database
  - Alter session (subset)
- Best effort for streams; OCI only – no ADT's or AQ
- Active Data Guard with read/write DB links

Target Database

- Databases able to diverge
  - Logical Standby
  - Golden Gate
  - PDB Clone

Steps to use Application Continuity

Check	What to do
Request Boundaries	UCP, WebLogic, and supported 3rd Party App servers – return connections to pool
JDBC Deprecated Classes	Replace non-standard classes (MOS 1364193.1); use assessment to know
Side Effects	Use disable API if a request has a call that should not be replayed
Callbacks	Register a callback for applications that change state outside requests For WebLogic Active Gridlink and UCP labels – do nothing
Mutable Functions	Grant keeping mutable values, e.g. sequence.nextval

Request Boundaries

Let the database know that it has a request

- Oracle Pools – JDBC UCP and WebLogic
  - **Return connections to pool**

- 3rd Party Java Application Servers 

  IBM WebSphere, Apache Tomcat, your own
  - Use UCP – a simple DataSource switch
  - **Return connections to pool**

- Custom - Standalone Java, 3rd Party

Disabling Replay

Use disableReplay API for requests that should not be replayed.

Make a conscious decision to replay side effects

e.g. Autonomous Transactions
UTL_HTTP
UTL_URL
UTL_FILE
UTL_FILE_TRANSFER
UTL_SMTP
UTL_TCP
UTL_MAIL
DBMS_JAVA callouts
EXTPROC

Grant Mutables

Keep original function results at replay

For owned sequences:

ALTER SEQUENCE.. [sequence object] [KEEP|NOKEEP];

CREATE SEQUENCE.. [sequence object] [KEEP|NOKEEP];

Grant and Revoke for other users:

GRANT [KEEP DATE TIME | KEEP SYSGUID].. [to USER]

REVOKE [KEEP DATE TIME | KEEP SYSGUID][from USER]

GRANT KEEP SEQUENCE on [sequence object] [to USER] ;

REVOKE KEEP SEQUENCE on [sequence object] [from USER]

Callbacks

For applications that set state outside database requests

- WebLogic and UCP Connection Labeling
  - Do nothing
- Custom
  - Register Connection Initialization Callback
  - Sets initial state for a session at BOTH runtime and replay
  - Available with WebLogic, UCP, JDBC-Thin driver

Configuration at Database

Set Service Attributes

FAILOVER_TYPE = TRANSACTION for Application Continuity

Review the service attributes:

- COMMIT_OUTCOME = TRUE for Transaction Guard
- REPLAY_INITIATION_TIMEOUT = 300 after which replay is canceled
- FAILOVER_RETRIES = 30 for the number of connection retries per replay
- FAILOVER_DELAY = 3 for delay in seconds between connection retries

Configuration at Client

Use JDBC Replay Data Source

At WebLogic Console or UCP, or your own property file –

```
Select new 12.1 datasource

replay datasource=oracle.jdbc.replay.OracleDataSourceImpl
```

Use JDBC statement cache rather than the WLS Statement Cache

Killing Sessions - Extended

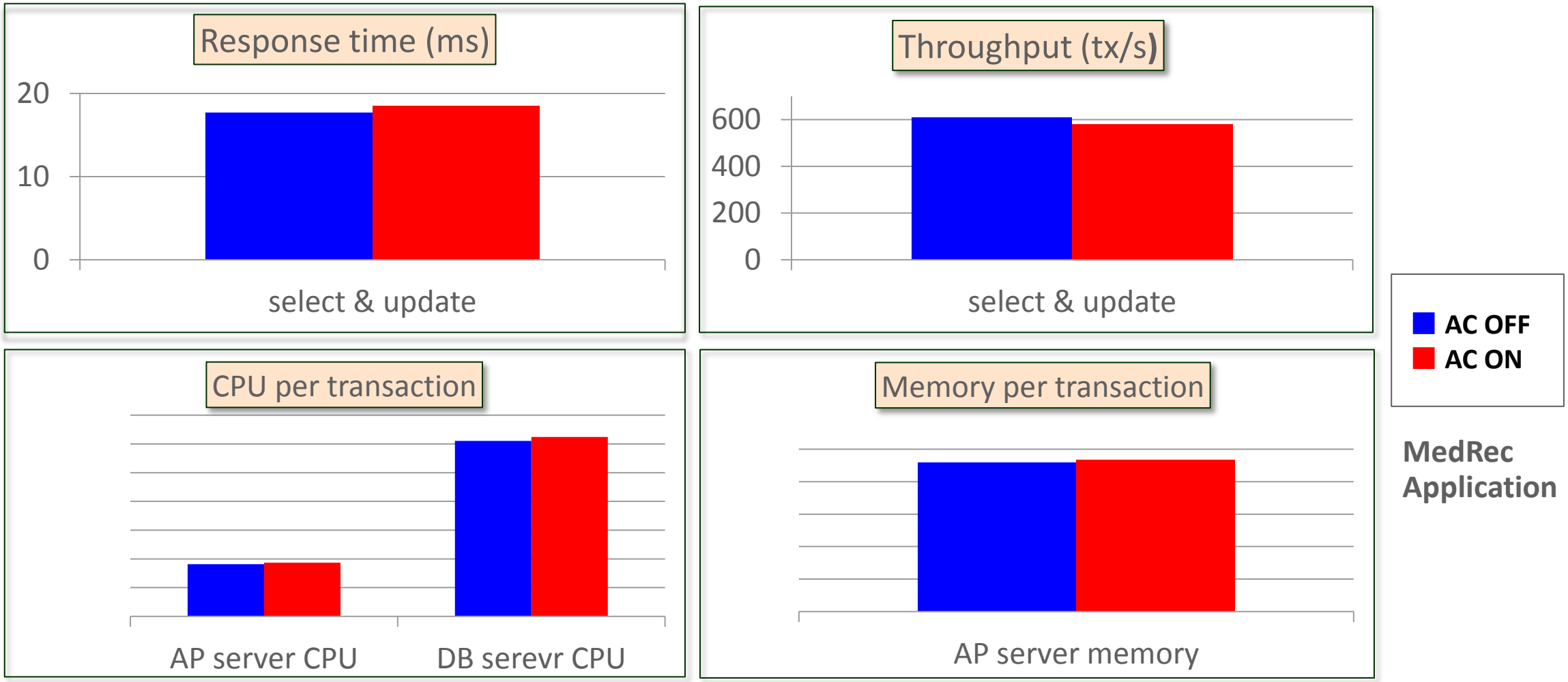
DBA Command	Replays
srvctl stop service -db orcl -instance orcl2 -force	**YES**
srvctl stop service -db orcl -node rws3 -force	**YES**
srvctl stop service -db orcl -instance orcl2 **–noreplay** -force	
srvctl stop service -db orcl -node rws3 **–noreplay** -force	
alter system kill session … immediate	**YES**
alter system kill session … **noreplay**	
dbms_service.disconnect_session([service], dbms_service. **noreplay**)	

Application Continuity Performance

WebLogic Server Active GridLink and Real Application Clusters

Empowered by Innovation

NEC



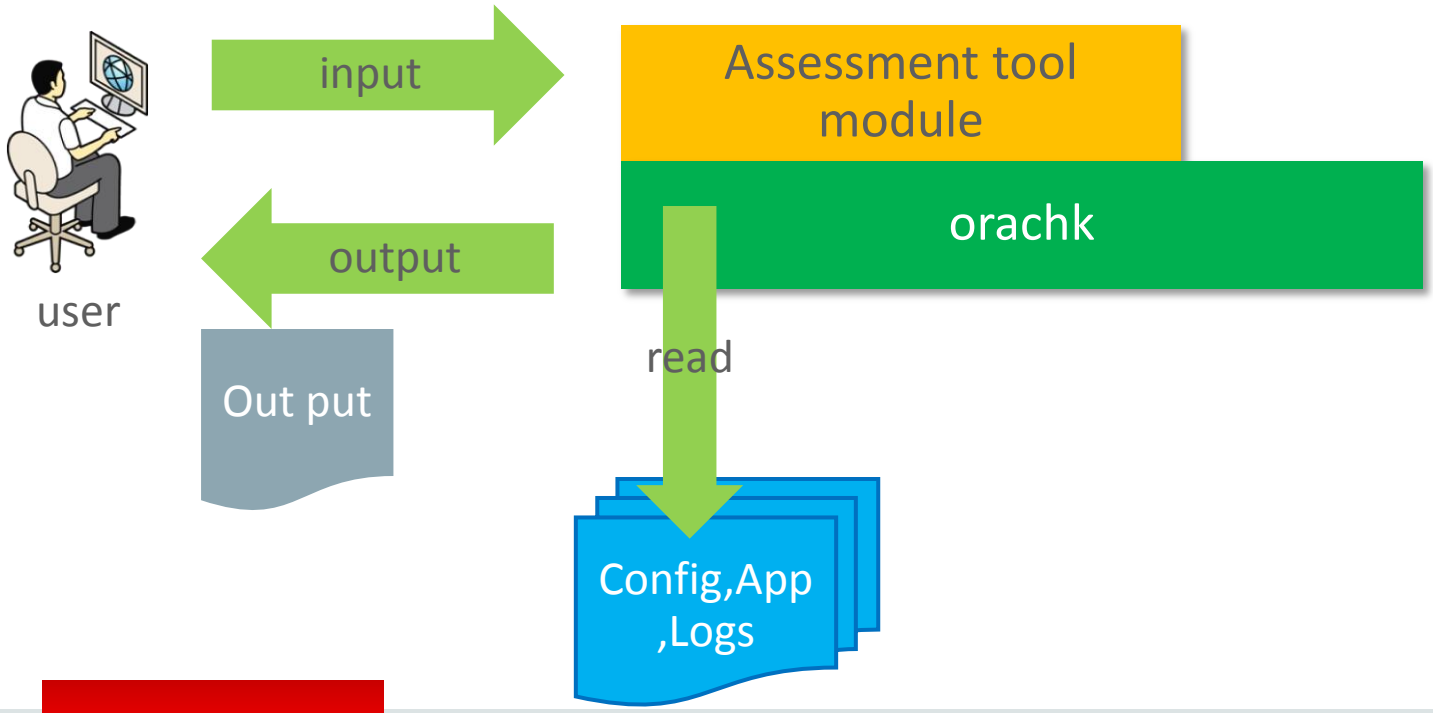
ORACLE

AC Assessment – in ORAchk

How effective is Application Continuity for user application

Where Application Continuity is not in effect - what steps need to be taken

When Application Continuity cannot be used and why due to a global restriction



No	Assessment functions
0	Pretest(sanity check)
1	JDBC Concrete Classes
2	Request Boundaries and Protection Level
3	Decide to Disable
4	Callbacks
5	Mutable Functions

Available May 2015 ORAchk

AC Statistics

Supported for Oracle JDBC replay driver

Statistics are client-side, cumulative per-connection or total for all pooled connections using oracle.jdbc.replay.ReplayableConnection

ReplayableConnection.getReplayStatistics (FOR_CURRENT_CONNECTION) returns statistics for current connection

ReplayableConnection.getReplayStatistics (FOR_ALL_CONNECTIONS) returns statistics for all connections in the pool

ReplayableConnection. clearReplayStatistics(StatisticsReportType) clears replay statistics – per connection or all connections

Runtime

- TotalRequests = 1
- TotalCompletedRequests = 1
- TotalCalls = 19
- TotalProtectedCalls = 19

Replay

- TotalCallsAffectedByOutages = 3
- TotalCallsTriggeringReplay = 3
- TotalCallsAffectedByOutagesDuringReplay = 0
- SuccessfulReplayCount = 1
- FailedReplayCount = 0
- ReplayDisablingCount = 0
- TotalReplayAttempts = 3

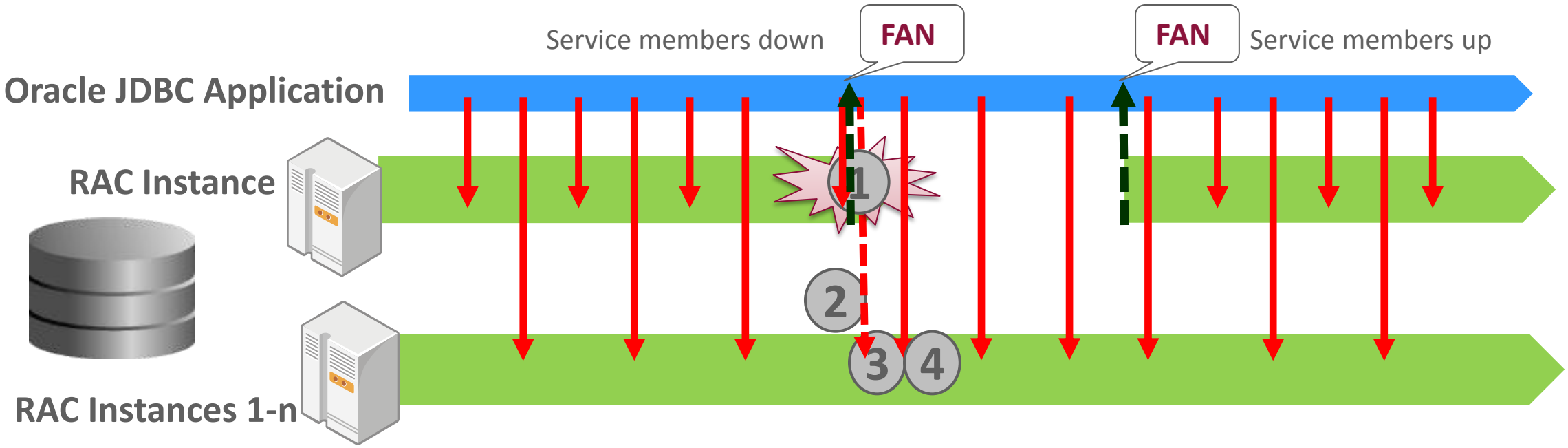
Case Study – Instance Outage

Empowered by Innovation

NEC

Application Continuity replays – application sees no errors

1. Instance outage *
2. Replay driver receives error/FAN and connects to another RAC instance
3. Application Continuity replays
4. Application continues and returns to client



* Similar for session exit without FAN.

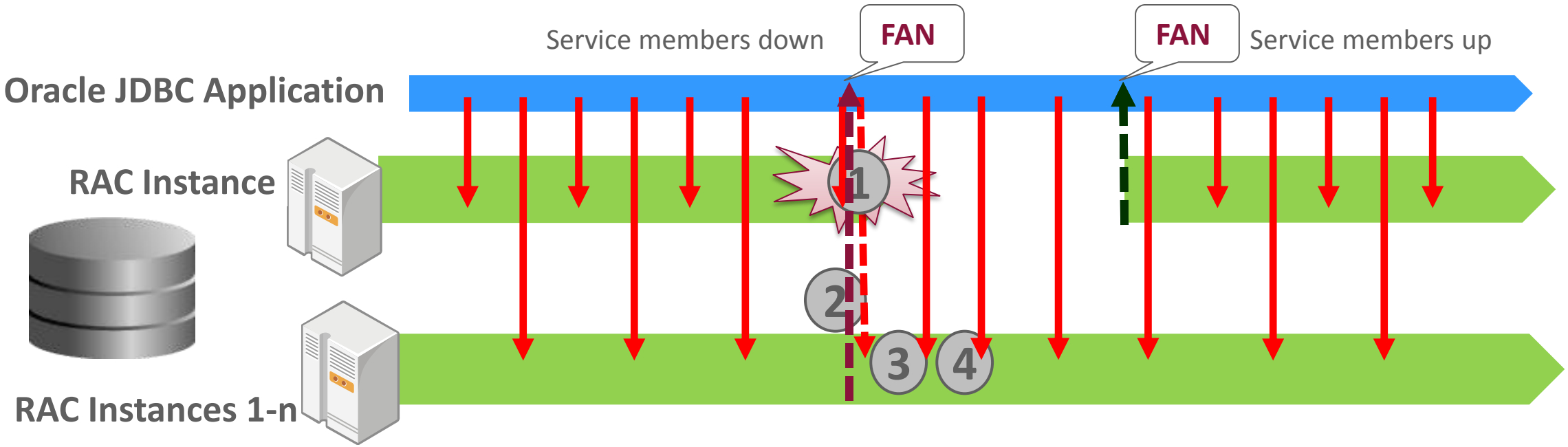
Case Study – Public Network Down

Empowered by Innovation

NEC

Application Continuity replays – application sees no errors

1. Public Network Down
2. Replay driver receives FAN from survivor and connects to another RAC instance
3. Application Continuity replays
4. Application continues and returns to client



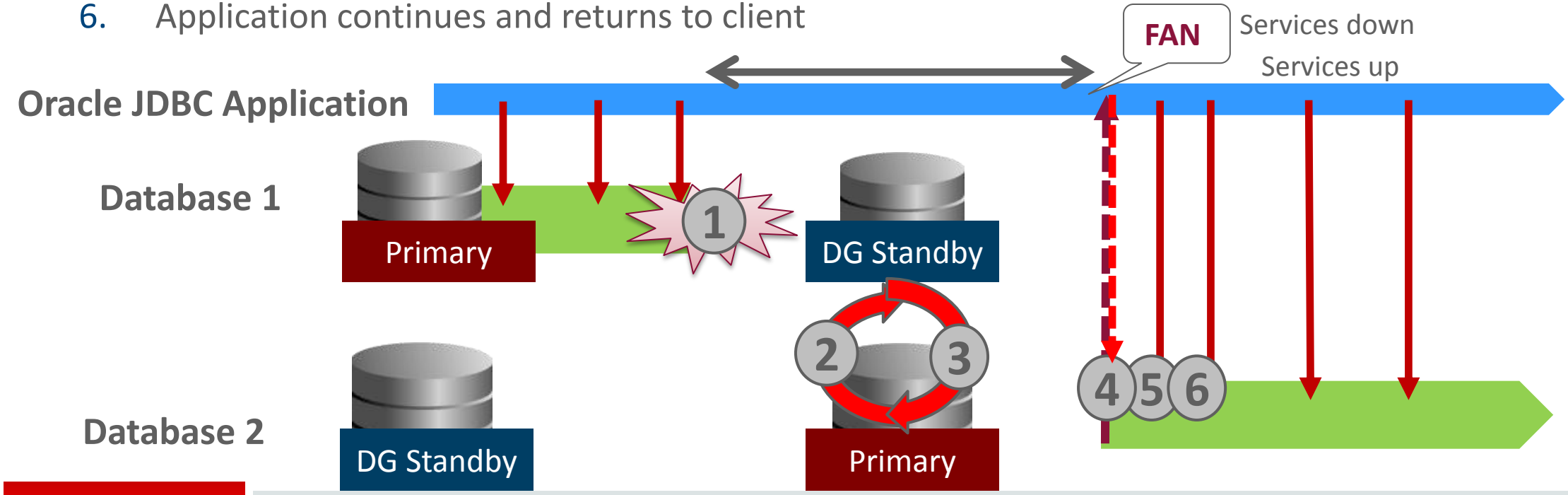
Case Study – Site Down

Application Continuity replays – application sees no errors

Empowered by Innovation

NEC

1. Site or database down
2. FSFO observer waits FastStartFailoverThreshold
3. FSFO observer automated failover
4. Replay driver receives FAN from secondary site and connects to another RAC instance*
5. Application Continuity replays
6. Application continues and returns to client



ORACLE

*** Tuning tip :** Set RETRY_COUNT and RETRY_DELAY to prevent errors for incoming connection requests

Recommended Patches

Component	Bug #	Description
Java Net	19154304	RETRY_COUNT did not include services
JavaNet	19000803	Provide TRANSPORT_CONNECT_TIMEOUT
RDBMS	19152020	PMON fast cleanup
RDBMS	19174056	Hang Manager extension
WebLogic	19587233	FAN and Application Continuity + JTS integration
WebLogic	20907322	FAN Autoons support

Lessons Learned

- Return connections to the connection pool between requests.
- Set http_request_timeout to allow the replay to occur
- Set REPLAY_INITIATION_TIMEOUT, RETRY_COUNT, and RETRY_DELAY
- Use mutable values. Think of mutables in terms of delayed execution.
- If the application sets values after creating a connection outside the application – repeat these settings in the callback.
- If the application is using XA datasource – check why. Most apps do not need XA.
- If testing and using V$instance etc, put in the callback to prevent mismatch as replay.

Transaction Guard

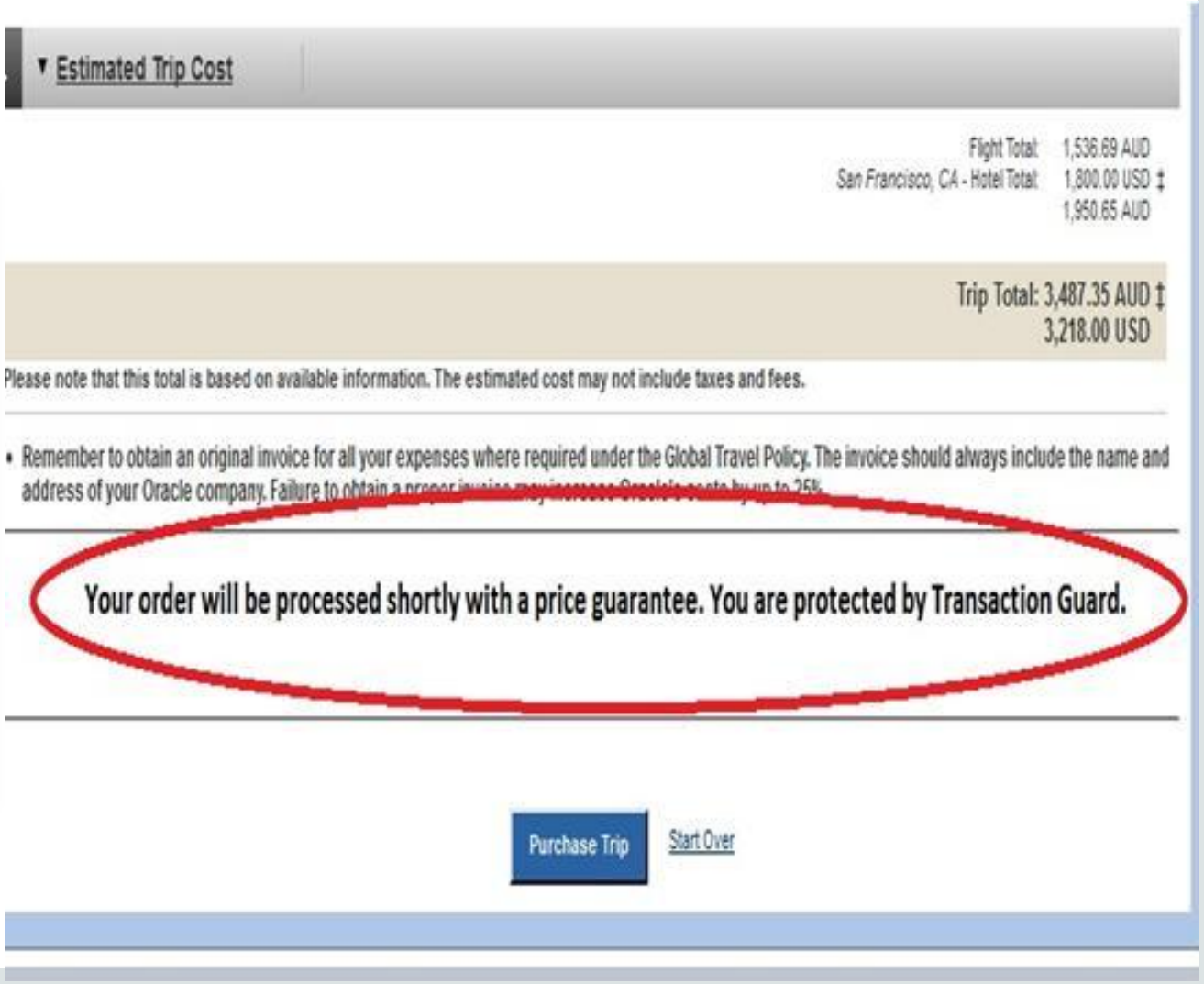
Unplanned outages should be hidden from applications

Transaction Guard

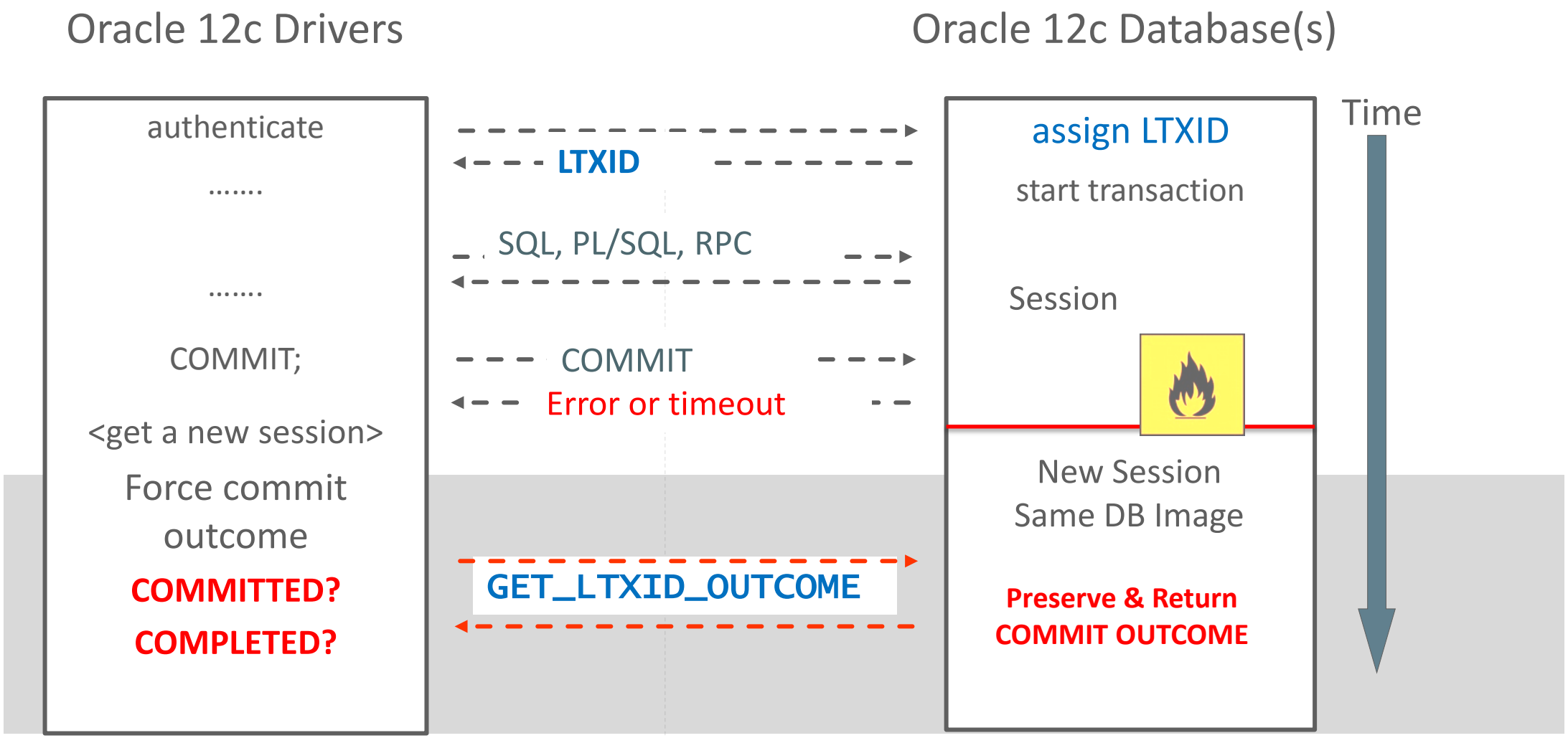
First RDBMS to preserve **COMMIT** Outcome

Reliable transaction outcome after outages

- Allows applications to deal with failures and timeouts correctly
- Without Transaction Guard, retrying can cause logical corruption
- Application Continuity uses Transaction Guard
- API available with JDBC-thin, OCI/OCCI, ODP.NET



How Transaction Guard Works



Transaction Coverage

Inclusions

- Local
- Commit on Success (auto-commit)
- Distributed and Remote
- DDL, DCL, parallel DDL
- PL/SQL with embedded COMMIT
- PL/SQL with COMMIT as last call
- Read-only (allowed for)

Exclusions

- XA in 12.1
- Active Data Guard with database links used to commit at primary

Database Target - Coverage

Inclusions 12.1

Single Instance Oracle RDBMS

RAC One Node

Real Application Clusters

Data Guard

Active Data Guard

Multitenant including unplug/plug

plus Transparent Application Failover (pre-integrated)

Exclusions Database Failed Over To -

Logical Standby

PDB Clones

Golden Gate and third party replication

Forcing Commit Outcome

DBMS_APP_CONT.GET_COMMIT_OUTCOME forces the commit outcome, returning -

- COMMITTED
  - TRUE the user call executed at least one commit
  - FALSE the user call is uncommitted and stays that way

- USER_CALL_COMPLETED
  - TRUE the user call ran to completion.
  - FALSE the user call is not known to have finished

    e.g. use if app expects return data – e..g commit on success, commit embedded in PL/SQL

Exceptions

- SERVER_AHEAD
  - the server is ahead of the client.
  - the transaction is an old transaction and must have already been committed
- CLIENT_AHEAD
  - the client is ahead of the server.
  - This can happen if the server has been flashed backed or using commit nowait
- ERROR
  - During processing an error happened.

Use Case - Unambiguous Outcome

Database session outage

FAN aborts dead session FAST

Application receives an error

Get last LTXID from dead session

Obtain a new database session

// Force commit outcome

execute **DBMS_APP_CONT.GET_LTXID_OUTCOME** using last LTXID

Add this part in the error handling routine

If **committed** then {

process committed ; // let user or app know it committed

if user_call_completed then application may continue

else application may not be able to continue}

Else process **uncommitted** // let user know its safe to resubmit or resubmit automatically

What **NOT** to do – assume it did not commit

Expert Level

```
Connection jdbcConnection = getConnection();
boolean isJobDone = false;
while(!isJobDone) {
  try {
    // apply the raise (DML + commit):
    giveRaiseToAllEmployees(jdbcConnection,5);
    // no exception, we consider the job as done:
    isJobDone = true;
  } catch (SQLRecoverableException recoverableException) {
    // On SQLRecoverableException, retry until isJobDone is true.
try {
      jdbcConnection.close();
    } catch(Exception ex) {} // ignore any exception
    // Now reconnect so that we can retry:
    jdbcConnection = getConnection();
  }
}
```

Incorrect logic is here.
An error does not mean it did not commit

ORACLE

What **NOT** to do – continued

Expert Level

```
void giveRaiseToAllEmployees(Connection conn, int percentage) throws SQLException {
  Statement stmt = null;
  try {
    stmt = conn.createStatement();
    stmt.executeUpdate("UPDATE emp SET sal=sal+(sal*"+percentage+"/100)");
  } catch (SQLException sqle ) {
    throw sqle;
  }
  finally {
    if(stmt != null)
      stmt.close();
  }
  // At the end of the request commit the changes:
  conn.commit();   // commit can succeed but return is lost
} .... (continued)
```

Problem occurs here

Solve with Transaction Guard - JDBC

Expert Level

```
Connection jdbcConnection = getConnection();
boolean isJobDone = false;
while(!isJobDone) {
  try {
    // apply the raise (DML + commit):
    giveRaiseToAllEmployees(jdbcConnection, 5);
    // no exception, the procedure completed:
    isJobDone = true;
  } catch (SQLRecoverableException recoverableException) {
    // Retry only if the error was recoverable.
    try {
      jdbcConnection.close();   // close old  connection:
    } catch (Exception ex) {}
    Connection newJDBCConnection = getConnection();   // reconnect to allow retry
    // Use Transacton Guard to force last outcome : committed or uncommitted
    LogicalTransactionId ltxid
      = ((OracleConnection)jdbcConnection).getLogicalTransactionId();
    isJobDone = getTransactionOutcome(newJDBCConnection, ltxid);
    jdbcConnection = newJDBCConnection;
  }
```

Catch recoverable exception

Use Transaction Guard

Solve with Transaction Guard – ODP.NET

Expert Level

```
catch(Exception ex)
{
   OracleLogicalTransaction olt = con.OracleLogicalTransaction;
   olt.GetOutcome();    // obtains new connection

   if (!olt.Committed)  // guaranteed uncommitted
   {
         // safe for application or user to resubmit here
   }
  else
   {  // transaction committed
      // test for completion  – This part is not needed for top level commit, and when states are not needed
      if (olt.UserCallCompleted)
      {
        // return committed status
      else
      {
        // return committed status - and warn that return states are unavailable
      }
    }
```

Transaction Guard

Required if using TAF Basic or TAF SELECT

- TAF handles Transaction Guard for OCI and ODP.NET apps
  - Set a boolean at connect and in TAF callback for TGenabled

```
catch (Exception ex)
    {
      //ONLY resubmit for the listed TAF errors and ONLY when TG is enabled

       if (TGEnabled &&  (ex.Number == 25402 || ex.Number == 25408 || ex.Number == 25405 ))
      {
         // application may cleanup, then rollback and re-submit the current transaction
      }
      else
      {
         // handle the error as before; do not resubmit
      }
```

Refer MOS 2011697.1

Use Case - Application Resubmits until Committed

Force Commit Outcome	Application Step
Recoverable error occurs	Obtain **LTXID-A-n,** Get a new session, Execute GET_LTXID_OUTCOME
COMMITTED and COMPLETED	**Return committed and continue**
COMMITTED AND NOT COMPLETED	Return committed, some apps cannot continue
UNCOMMITTED	Resubmit with a new session with **LTXID-B-0**
Recoverable error	Obtain **LTXID-B-n,** Get a new session, Execute GET_LTXID_OUTCOME
UNCOMMITTED	Resubmit with a new session with **LTXID-C-0**
COMMITTED and COMPLETED	**Return committed and continue**

Server-side settings for Transaction Guard

- **On Service**
  - COMMIT_OUTCOME
    - Values – TRUE and FALSE
    - Default – FALSE
    - Applies to new sessions
- **GRANT EXECUTE** ON DBMS_APP_CONT TO <user>;

Transaction Guard – Key Takeaway

First RDBMS to preserve commit outcome

- Users should not see misleading errors when a transaction really did commit.
- Driver receives an LTXID at authentication and on every commit.
- Once the commit outcome is returned, **the result never changes**.
- Safe for applications and mid-tiers to return success or resubmit themselves.

Success Stories
Out of the Box

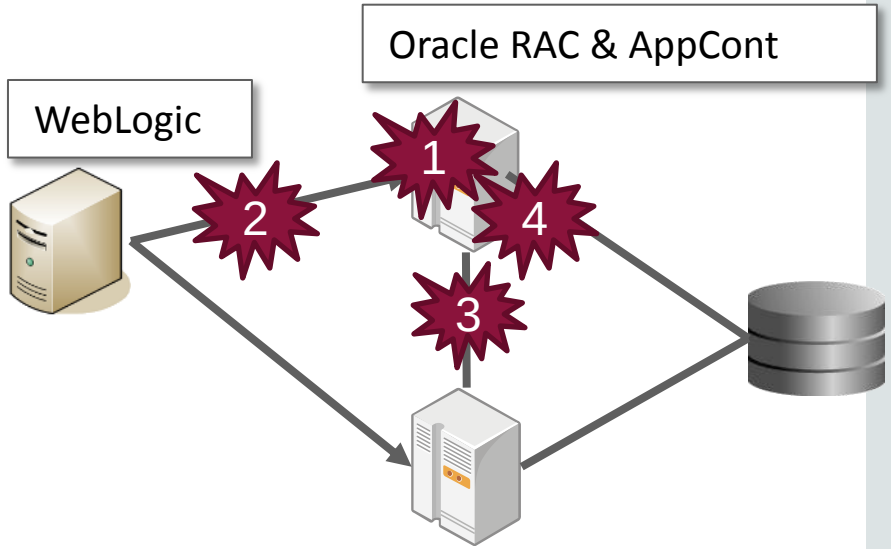
Unplanned Failover with Application Continuity

WebLogic Active GridLink and Real Application Clusters

Empowered by Innovation

NEC

	BEFORE	AFTER
1	DB 11gR2+WLS Generic DS Error AP wait time:1s	DB12c+ GridLink+AppCont No errors, App Continues AP wait time:1s
2	DB 11gR2+WLS Generic DS TIMEOUT 900s (TCP keep-alive)	DB12c+ GridLink+AppCont No errors, App Continues AP wait time:1s
3	DB 11gR2+WLS Generic DS Error AP wait time: 30s	DB12c+ GridLink+AppCont No errors, App Continues AP wait time:30s
4	DB 11gR2+WLS Generic DS Hang AP wait time: minutes	DB12c+ GridLink+AppCont + NEC Monitor : No errors, App Continues



1. Instance down
2. Public network down
3. Interconnect down
4. Background process hang

ORACLE

Planned Failover with FAN

WebLogic Server Active GridLink, RAC and Data Guard

Empowered by Innovation

NEC

DBA Operation	Maintenance	Result	Time to Drain all Sessions
RAC rolling	PSU apply using opatch	No errors to application	5s
RAC rolling	Instance parameter change	No errors to application	7s
Data Guard switchover	Site maintenance	No errors to application	29s
Data Guard switchover	Site maintenance fallback	No errors to application	25s

ORACLE

Planned and Unplanned Failover

RAC One Node, IBM WebSphere, Universal Connection Pool

Swiss Re

Maintenance	Result	Time allowed
Planned with FAN + Net	No errors to application	4 hours
Unplanned with Application Continuity + Net	No errors to application	10 minutes

ORACLE

Runtime, Planned, & Unplanned

ODP.NET Unmanaged Provider, RAC, and Data Guard

EPSILON

Database Method	Client Method	Example	Result	Time to Drain Sessions
RAC rolling upgrade/change	Drain with FAN + TNS	PSU / CPU	No errors to application	5s
Data Guard Switchover	Drain with FAN + TNS	Standby first PSU/CPU	No errors to application	25s
RAC Failover	Failover with FAN + TNS + TAF SELECT	Node outage	Errors for transactions	5s
Data Guard Failover	Failover with FAN + TNS + TAF SELECT	Site outage	Errors for transactions	-

ORACLE

Pythian
love your data®

For Developers : Application Continuity offloads the challenging work of transaction resubmission during failure events, allowing developers to focus on functionality.

Christo Kutrovsky – ATCG Principal Consultant, Oracle ACE

For Enterprise Architects : Application Continuity is a major step towards the holy grail of a continuously available, consistent, and highly performing database cluster

Marc Fielding – ATCG Principal Consultant, Oracle

ORACLE

Empowered by Innovation

NEC

The combinatorial solution with Application Continuity, Real Application Clusters, Data Guard, WebLogic Server Active GridLink and NEC hardware and middleware enables us to provide incredibly high available system for our Mission Critical customers. This solution will become our primary solution for cloud and big data areas.

Yuki Moriyama

Senior Manager, NEC Corporation

ORACLE

Safe Harbor Statement

The preceding 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.


ORACLE®