



# ORACLE®

## 11 things about Oracle Database 11g Release 2

Thomas Kyte

<http://asktom.oracle.com/>





# 1 Do it yourself Parallelism

# Incrementally modify a table in parallel

- Used to do this manually all of the time
  - Search for ‘diy parallel’ on asktom...
  - Spent part of a chapter on ‘how to’ in Expert Oracle Database Architecture
- I split by rowid ranges
  - Split table into N equi-sized, non-overlapping chunks
  - Create a job passing in the low and high rowids for each range
  - Job would process “where rowid between :lo and :hi”
- Or by primary key ranges using NTILE()
- DBMS\_PARALLEL\_EXECUTE automates both approaches and makes it easy (and more functional)

# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> create table t
  2  as
  3  select *
  4      from all_objects
  5  /
```

Table created.

```
ops$tkyte%ORA11GR2> exec dbms_stats.gather_table_stats( user, 'T' );
```

PL/SQL procedure successfully completed.

```
ops$tkyte%ORA11GR2> select blocks, blocks/10 from user_tables where
  table_name = 'T';
```

BLOCKS	BLOCKS/10
1044	104.4

# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> create table log
 2  ( lo_rowid  rowid,
 3    hi_rowid  rowid,
 4    nrows     number,
 5    stime     timestamp,
 6    etime     timestamp
 7  )
 8  /
```

Table created.

# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> create or replace
 2  procedure do_update( p_lo_rowid in rowid, p_hi_rowid in rowid )
 3  as
 4      l_rid rowid;
 5      l_cnt number;
 6  begin
 7      insert into log (lo_rowid,hi_rowid,stime)
 8      values (p_lo_rowid,p_hi_rowid,systimestamp)
 9      returning rowid into l_rid;
10
11      update t set object_name = lower(object_name)
12      where rowid between p_lo_rowid and p_hi_rowid;
13      l_cnt := sql%rowcount;
14
15      update log
16      set etime = systimestamp,
17      nrows = l_cnt
18      where rowid = l_rid;
19  end;
20  /
Procedure created.
```

# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> begin
  2          dbms_parallel_execute.create_task('update t');
  3          dbms_parallel_execute.create_chunks_by_rowid
  4          ( task_name    => 'update t',
  5            table_owner => user,
  6            table_name  => 'T',
  7            by_row      => false,
  8            chunk_size  => 100);
  9  end;
 10  /
```

PL/SQL procedure successfully completed.

```
ops$tkyte%ORA11GR2> select chunk_id, status, start_rowid, end_rowid
  2    from dba_parallel_execute_chunks
  3   where task_name = 'update t'
  4  /
```

CHUNK_ID	STATUS	START_ROWID	END_ROWID
194	UNASSIGNED	AAASTlAAEAAAAAdkAAA	AAASTlAAEAAAAAd/CcP
193	UNASSIGNED	AAASTlAAEAAAAAcAAAA	AAASTlAAEAAAAAdjCcP

...

# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> begin
  2      dbms_parallel_execute.run_task
  3      ( task_name      => 'update t',
  4        sql_stmt       => 'begin do_update( :start_id, :end_id ); end;',
  5        language_flag  => DBMS_SQL.NATIVE,
  6        parallel_level => 2 );
  7 end;
  8 /
```

PL/SQL procedure successfully completed.

```
ops$tkyte%ORA11GR2> select chunk_id, status, start_rowid, end_rowid
  2      from dba_parallel_execute_chunks
  3      where task_name = 'update t'
  4      /
```

CHUNK_ID	STATUS	START_ROWID	END_ROWID
195	PROCESSED	AAASTlAAEAAAAeAAAA	AAASTlAAEAAAAfjCcP
196	PROCESSED	AAASTlAAEAAAAfkAAA	AAASTlAAEAAAAf/CcP

...



# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> begin
  2          dbms_parallel_execute.drop_task('update t');
  3  end;
  4  /
```

PL/SQL procedure successfully completed.

```
ops$tkyte%ORA11GR2> select chunk_id, status, start_rowid, end_rowid
  2      from dba_parallel_execute_chunks
  3  where task_name = 'update t'
  4  /
```

no rows selected

# Incrementally modify a table in parallel

```
ops$tkyte%ORA11GR2> select nrows, stime, etime-stime ela from log;
```

NROWS	STIME	ELA
1950	07-OCT-09 11.38.38.441904 AM	+0000000000 00:00:00.056220
6747	07-OCT-09 11.38.38.499673 AM	+0000000000 00:00:00.571049
1911	07-OCT-09 11.38.39.072111 AM	+0000000000 00:00:00.060847
6662	07-OCT-09 11.38.37.364203 AM	+0000000000 00:00:00.150791
1952	07-OCT-09 11.38.37.519093 AM	+0000000000 00:00:00.057181
6920	07-OCT-09 11.38.37.577507 AM	+0000000000 00:00:00.146901
1999	07-OCT-09 11.38.37.725649 AM	+0000000000 00:00:00.008060
6997	07-OCT-09 11.38.37.734748 AM	+0000000000 00:00:00.152851
...		
6663	07-OCT-09 11.38.38.069751 AM	+0000000000 00:00:00.533909
1914	07-OCT-09 11.38.38.605693 AM	+0000000000 00:00:00.029193
6653	07-OCT-09 11.38.38.635749 AM	+0000000000 00:00:00.447706

32 rows selected.



2 Analytics are the coolest thing to happen to SQL since the keyword SELECT

# More Analytics!

- Long awaited LISTAGG
  - First did STRAGG in 9iR2 with user defined aggregates
  - Oracle Database 10g gave us a sys\_connect\_by\_path 'trick'
  - Oracle Database 11g Release 2 makes it 'easy'

# Analytics Rock and Roll

```
SQL> select deptno,
2         substr(
3         max(sys_connect_by_path(ename, ';' )),
4         3) enames
5   from (
6   select deptno,
7         ename,
8         row_number()
9         over
10        (partition by deptno
11         order by ename) rn
12   from emp
13        )
14   start with rn = 1
15   connect by prior deptno = deptno
16         and prior rn+1 = rn
17   group by deptno
18   order by deptno
19  /
```

DEPTNO ENAMES

```
-----
10 CLARK; KING; MILLER
20 ADAMS; FORD; JONES;
   SCOTT; SMITH

30 ALLEN; BLAKE;
   JAMES; MARTIN;
   TURNER; WARD
```

# Analytics Rock and Roll

```
SQL> select deptno,  
2         listagg( ename, ';' )  
3         within group  
4         (order by ename) enames  
5   from emp  
6   group by deptno  
7   order by deptno  
8   /
```

DEPTNO ENAMES

-----  
10 CLARK; KING; MILLER

20 ADAMS; FORD; JONES;  
SCOTT; SMITH

30 ALLEN; BLAKE;  
JAMES; MARTIN;  
TURNER; WARD

# Analytics Rock and Roll

```
SQL> select deptno,
2         ename,
3         row_number()
4         over (partition by deptno
5               order by ename) rn,
6         first_value(ename)
7         over (partition by deptno
8               order by ename) "1st ename",
9         nth_value(ename,3)
10        over (partition by deptno
11              order by ename) "3rd ename",
12         last_value(ename)
13        over (partition by deptno
14              order by ename
15              rows between current row
16                    and unbounded following) "last ename"
17    from emp
18   order by deptno, ename
19  /
```

# Analytics Rock and Roll

	DEPTNO	ENAME	RN	1st e	3rd ena	last en
	-----	-----	--	-----	-----	-----
	10	CLARK	1	CLARK		MILLER
		KING	2	CLARK		MILLER
		MILLER	3	CLARK	MILLER	MILLER
	20	ADAMS	1	ADAMS		SMITH
		FORD	2	ADAMS		SMITH
		JONES	3	ADAMS	JONES	SMITH
		SCOTT	4	ADAMS	JONES	SMITH
		SMITH	5	ADAMS	JONES	SMITH
	30	ALLEN	1	ALLEN		WARD
		BLAKE	2	ALLEN		WARD
		JAMES	3	ALLEN	JAMES	WARD
		MARTIN	4	ALLEN	JAMES	WARD
		TURNER	5	ALLEN	JAMES	WARD
		WARD	6	ALLEN	JAMES	WARD

```

SQL> select deptno,
2      ename,
3      row_number()
4      over (partition by deptno
5            order by ename) rn,
6      first_value(ename)
7      over (partition by deptno
8            order by ename) "1st ename",
9      nth_value(ename,3)
10     over (partition by deptno
11           order by ename) "3rd ename",
12      last_value(ename)
13     over (partition by deptno
14           order by ename
15           rows between current row
16               and unbounded following) "last ename"
17   from emp
18  order by deptno, ename
19 /

```





## 3 Execute on a directory

## External Tables can run code now

- External tables allow for a preprocessor
  - Program is run when you SELECT from external table
  - The 'location' is passed to the script/executable
  - The executable does whatever it wants and writes to stdout
  - Stdout is treated as the input file
- We need a way to control who can do what
- GRANT EXECUTE ON DIRECTORY handles that

# EXECUTE and PREPROCESSOR

```
ops$tkyte%ORA11GR2> CREATE or replace DIRECTORY load_dir  
2 AS '/mnt/hgfs/docs/Presentations/Seminar/11gr2'  
3 /
```

Directory created.

```
ops$tkyte%ORA11GR2> CREATE or replace DIRECTORY exec_dir  
2 AS '/mnt/hgfs/docs/Presentations/Seminar/11gr2'  
3 /
```

Directory created.

# EXECUTE and PREPROCESSOR

```
ops$tkyte%ORA11GR2> CREATE TABLE EMP_ET
 2  (
 3      "EMPNO" NUMBER(4) ,
 4      "ENAME" VARCHAR2(10) ,
 5      "JOB" VARCHAR2(9) ,
 6      "MGR" NUMBER(4) ,
 7      "HIREDATE" DATE ,
 8      "SAL" NUMBER(7,2) ,
 9      "COMM" NUMBER(7,2) ,
10      "DEPTNO" NUMBER(2)
11  )
12  ORGANIZATION external
13  ( TYPE oracle_loader
14      DEFAULT DIRECTORY load_dir
15      ACCESS PARAMETERS
16      ( RECORDS DELIMITED BY NEWLINE
17          preprocessor exec_dir: 'run_gunzip.sh'
18          FIELDS TERMINATED BY "|" LDRTRIM
19      )
20      location ( 'emp.dat.gz')
21  )
22  /
```

Table created.

# EXECUTE and PREPROCESSOR

```
ops$tkyte%ORA11GR2> !file emp.dat.gz
```

```
emp.dat.gz: gzip compressed data, was "emp.dat", from Unix, last  
modified: Wed Oct  7 12:48:53 2009
```

```
ops$tkyte%ORA11GR2> !cat run_gunzip.sh
```

```
#!/bin/bash
```

```
/usr/bin/gunzip -c $*
```

```
ops$tkyte%ORA11GR2> select empno, ename from emp_et where rownum <= 5;
```

EMPNO	ENAME
7369	SMITH
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN

-----

# EXECUTE and PREPROCESSOR, interesting idea...

```
ops$tkyte%ORA11GR2> CREATE TABLE ls
 2  (
 3    line varchar2(255)
 4  )
 5  ORGANIZATION external
 6  ( TYPE oracle_loader
 7    DEFAULT DIRECTORY load_dir
 8    ACCESS PARAMETERS
 9    ( RECORDS DELIMITED BY NEWLINE
10      preprocessor exec_dir:'run_ls.sh'
11      FIELDS TERMINATED BY "|" LDRTRIM
12    )
13    location ( 'run_ls.sh')
14  )
15  /
```

Table created.

# EXECUTE and PREPROCESSOR, interesting idea...

```
ops$tkyte%ORA11GR2> select * from ls;
```

```
LINE
```

```
-----  
11 things about 11gr2.ppt
```

```
diyp.sql
```

```
ebr.old.sql
```

```
ebr.sql
```

```
emp.ctl
```

```
emp.dat.gz
```

```
EMP_ET_26122.log
```

```
emp_et.sql
```

```
LS_26122.log
```

```
run_gunzip.sh
```

```
run_ls.sh
```

```
11 rows selected.
```



## 4 Recursive Subquery Factoring



# Recursive Subquery Factoring

- ANSI SQL replacement for connect by
- Can be
  - Easier to understand than connect by
  - Unless of course, you have been using connect by for 22 years – in which case it looks confusing

# Recursive Subquery Factoring

```
ops$tkyte%ORA11GR2> with emp_data(ename,empno,mgr,1)
 2  as
 3  (select ename, empno, mgr, 1 lvl from emp where mgr is null
 4    union all
 5     select emp.ename, emp.empno, emp.mgr, ed.lvl+1
 6       from emp, emp_data ed
 7       where emp.mgr = ed.empno
 8  )
 9  SEARCH DEPTH FIRST BY ename SET order_by
10  select 1,
11          lpad('*',2*1,'*')||ename nm
12    from emp_data
13   order by order_by
14  /
```

# Recursive Subquery Factoring

L	NM
1	**KING
2	****BLAKE
3	*****ALLEN
3	*****JAMES
3	*****MARTIN
3	*****TURNER
3	*****WARD
2	****CLARK
3	*****MILLER
2	****JONES
3	*****FORD
4	*****SMITH
3	*****SCOTT
4	*****ADAMS

14 rows selected.

# Recursive Subquery Factoring

```
ops$tkyte%ORA11GR2> with data(r)
  2  as
  3  (select 1 r from dual
  4    union all
  5    select r+1 from data where r < 5
  6  )
  7  select r, sysdate+r
  8    from data;
```

```
      R SYSDATE+R
-----
1 08-OCT-09
2 09-OCT-09
3 10-OCT-09
4 11-OCT-09
5 12-OCT-09
```

# Recursive Subquery Factoring

- ANSI SQL replacement for connect by
- Can be
  - Easier to understand than connect by
  - Unless of course, you have been using connect by for 22 years – in which case it looks confusing
  - Used to solve Sudoku puzzles!



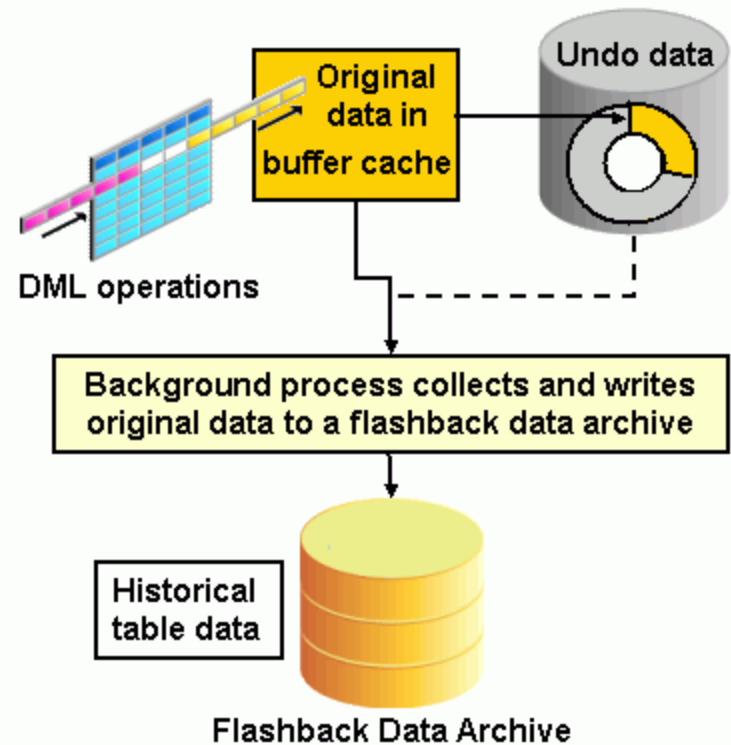
## 5 Improved Time Travel

# Improved Time Travel

- Flashback Data Archive
  - Query data as of 5 days, 5 weeks, 5 months, 5 years – whatever – in the past
  - <http://www.oracle.com/technology/oramag/oracle/08-jul/o48totalrecall.html>
    - Article by Jonathan Gennick on this feature for more info
- How does it work...

# How Does Flashback Data Archive Work?

- Primary source for history is the undo data
- History is stored in automatically created history tables inside the archive
- Transactions and its undo records on tracked tables marked for archival
  - Undo records not recycled until history is archived
- History is captured asynchronously by new background process (fbda)
  - Default capture interval is 5 minutes
  - Capture interval is self-tuned based on system activities
  - Process tries to maximize undo data reads from buffer cache for better performance
  - INSERTs do not generate history records

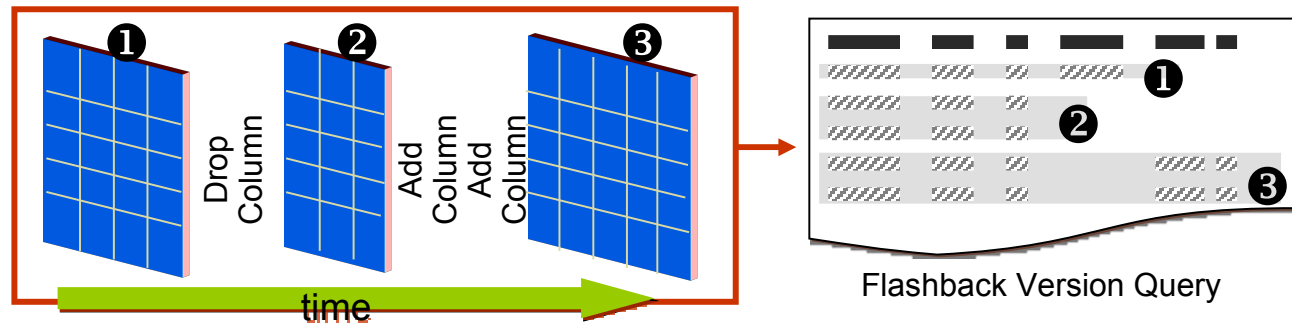




# Oracle Database 11g Release

## Total Recall Schema Evolution Support

- Alter base table – history table automatically adjusts
  - Drop, Rename, Modify Column
  - Drop, Truncate Partition
  - Rename, Truncate Table
- Flashback query supported across DDL changes



- Complex DDL changes (e.g. table split) accommodated
  - Associate/Dissociate history table via DBMS\_FLASHBACK\_ARCHIVE package



6 You've got Mail

# File Watchers

- As files arrive in some directory
  - An event is generated
  - And your code can be invoked to deal with it...

# File Watchers

```
ops$tkyte%ORA11GR2> begin
  2      dbms_scheduler.create_credential(
  3          credential_name => 'watch_credential',
  4          username         => 'tkyte',
  5          password         => 'foobar');
  6  end;
  7  /
```

PL/SQL procedure successfully completed.

# File Watchers

```
ops$tkyte%ORA11GR2> create or replace directory MY_FILES as  
  '/home/tkyte/files'  
2  /
```

Directory created.

```
ops$tkyte%ORA11GR2> create table files  
2  (  
3      file_name varchar2(100),  
4      loaded timestamp,  
5      contents  clob  
6  );
```

Table created.

# File Watchers

```
ops$tkyte%ORA11GR2> create or replace procedure process_files
  2  (p_payload in sys.scheduler_filewatcher_result)
  3  is
  4      l_clob clob;
  5      l_bfile bfile;
  6  begin
  7      insert into files
  8          (loaded, file_name, contents )
  9      values (p_payload.file_timestamp,
 10          p_payload.directory_path || '/' || p_payload.actual_file_name,
 11          empty_clob()
 12      ) returning contents into l_clob;
 13
 14      l_bfile := bfilename( 'MY_FILES', p_payload.actual_file_name );
 15      dbms_lob.fileopen( l_bfile );
 16      dbms_lob.loadfromfile( l_clob, l_bfile, dbms_lob.getlength(l_bfile) );
 17      dbms_lob.fileclose( l_bfile );
 18  end;
 19  /
```

Procedure created.

# File Watchers

```
ops$tkyte%ORA11GR2> begin
  2      dbms_scheduler.create_program(
  3          program_name          => 'file_watcher',
  4          program_type          => 'stored_procedure',
  5          program_action        => 'Process_Files',
  6          number_of_arguments  => 1,
  7          enabled                => false);
  8      dbms_scheduler.define_metadata_argument(
  9          program_name          => 'file_watcher',
10          metadata_attribute    => 'event_message',
11          argument_position     => 1);
12      dbms_scheduler.enable('file_watcher');
13  end;
14  /
```

PL/SQL procedure successfully completed.

# File Watchers

```
ops$tkyte%ORA11GR2> begin
  2      dbms_scheduler.create_file_watcher(
  3          file_watcher_name => 'my_file_watcher',
  4          directory_path     => '/home/tkyte/files',
  5          file_name           => '*',
  6          credential_name     => 'watch_credential',
  7          destination         => null,
  8          enabled              => false);
  9  end;
10  /
```

PL/SQL procedure successfully completed.



# File Watchers

```
ops$tkyte%ORA11GR2> begin
  2      dbms_scheduler.create_job(
  3          job_name          => 'my_file_job',
  4          program_name      => 'file_watcher',
  5          event_condition    => 'tab.user_data.file_size > 10',
  6          queue_spec        => 'my_file_watcher',
  7          auto_drop         => false,
  8          enabled            => false);
 10  end;
 11  /
```

PL/SQL procedure successfully completed.

```
ops$tkyte%ORA11GR2> exec
  dbms_scheduler.enable('my_file_watcher,my_file_job');
```

PL/SQL procedure successfully completed.

# File Watchers

```
ops$tkyte%ORA11GR2> select * from files;
```

FILE_NAME	LOADED	CONTENTS
/home/tkyte/files/file4.txt	07-OCT-09 07.37.22.000000 PM	hello world, how are you hello world, how are you hello world, how are you hello world, how are you hello world, how are you



## 7 Deferred Segment Creation

# Deferred Segment Creation

- Segments (tables, indexes, etc) normally allocate an initial extent
- They might be small, but they exist
- If you do something “small” (or fast) over and over a lot – it gets “big” (or slow)
- Many third party applications create thousands of tables
  - And then use 100 of them
- Deferred segment creation allows us to put off initial extent allocation until the first row is put into a segment.

# Deferred Segment Creation

```
SQL> alter session set
      2 deferred_segment_creation=false;
Session altered.
```

```
SQL> create table t1
      2 ( x int
      3     constraint t1_pk
      4     primary key,
      5     y int
      6     constraint t1_y
      7     unique,
      8     z clob
      9 )
     10 lob( z )
     11 store as t1_z_lob
     12 (index t1_z_lobidx);
Table created.
```

```
SQL> select segment_name,
      2         extent_id,
      3         bytes
      4     from user_extents
      5     order by segment_name;
```

SEGMENT_NAM	EXTENT_ID	BYTES
-----	-----	-----
T1	0	65536
T1_PK	0	65536
T1_Y	0	65536
T1_Z_LOB	0	65536
T1_Z_LOBIDX	0	65536

# Deferred Segment Creation

```
SQL> alter session set
      2 deferred_segment_creation=true;
Session altered.
```

No Change!

```
SQL> create table t2
      2 ( x int
      3     constraint t2_pk
      4     primary key,
      5     y int
      6     constraint t2_y
      7     unique,
      8     z clob
      9 )
     10 lob( z )
     11 store as t2_z_lob
     12 (index t2_z_lobidx);
Table created.
```

```
SQL> select segment_name,
      2         extent_id,
      3         bytes
      4     from user_extents
      5     order by segment_name;
```

SEGMENT_NAM	EXTENT_ID	BYTES
-----	-----	-----
T1	0	65536
T1_PK	0	65536
T1_Y	0	65536
T1_Z_LOB	0	65536
T1_Z_LOBIDX	0	65536

# Deferred Segment Creation

```
SQL> insert into t2 values ( 1, 2, 'hello world' );  
1 row created.
```

```
SQL> select segment_name,  
2          extent_id,  
3          bytes  
4      from user_extents  
5      order by segment_name;
```

SEGMENT_NAM	EXTENT_ID	BYTES
-----	-----	-----
T1	0	65536
T1_PK	0	65536
T1_Y	0	65536
T1_Z_LOB	0	65536
T1_Z_LOBIDX	0	65536
T2	0	65536
T2_PK	0	65536
T2_Y	0	65536
T2_Z_LOB	0	65536
T2_Z_LOBIDX	0	65536

10 rows selected.



## 8 Flash Cache



# Oracle Database 11g Release 2

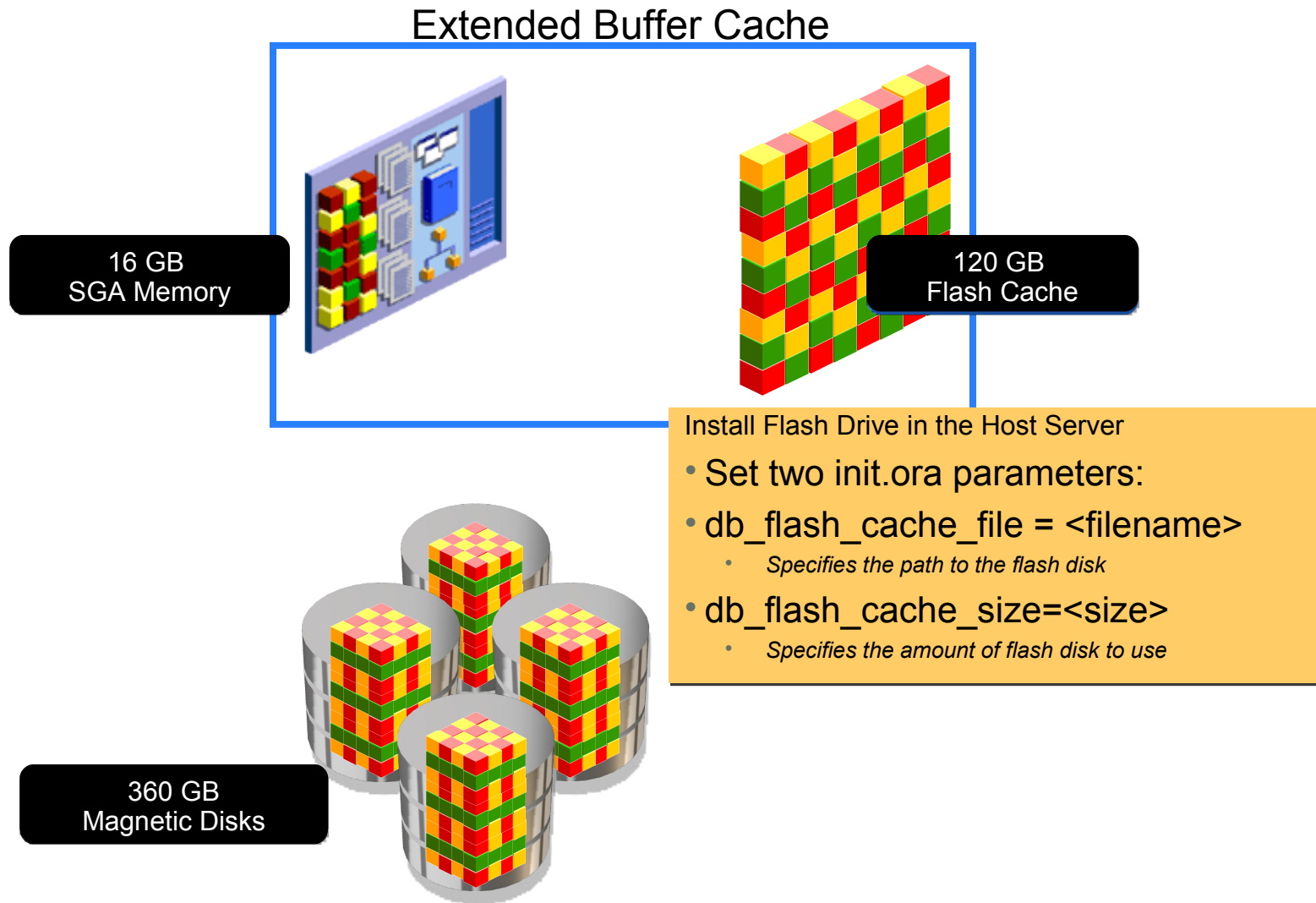
## Reduce I/O bandwidth requirement with Flash Cache

- A transparent extension of the database buffer cache using solid-state disk (SSD) technology
  - SSD acts as a Level 2 cache (SGA is Level 1)
    - Faster than disk (100x faster for reads)
    - Cheaper than memory (\$50 per gigabyte)
    - Large capacity (hundreds of gigabytes per flash disk)
- Fewer drives and better performance
  - For I/O throughput, users often use hundreds of drives today
  - Flash enables I/O throughput without all the drives
  - Large jobs complete faster



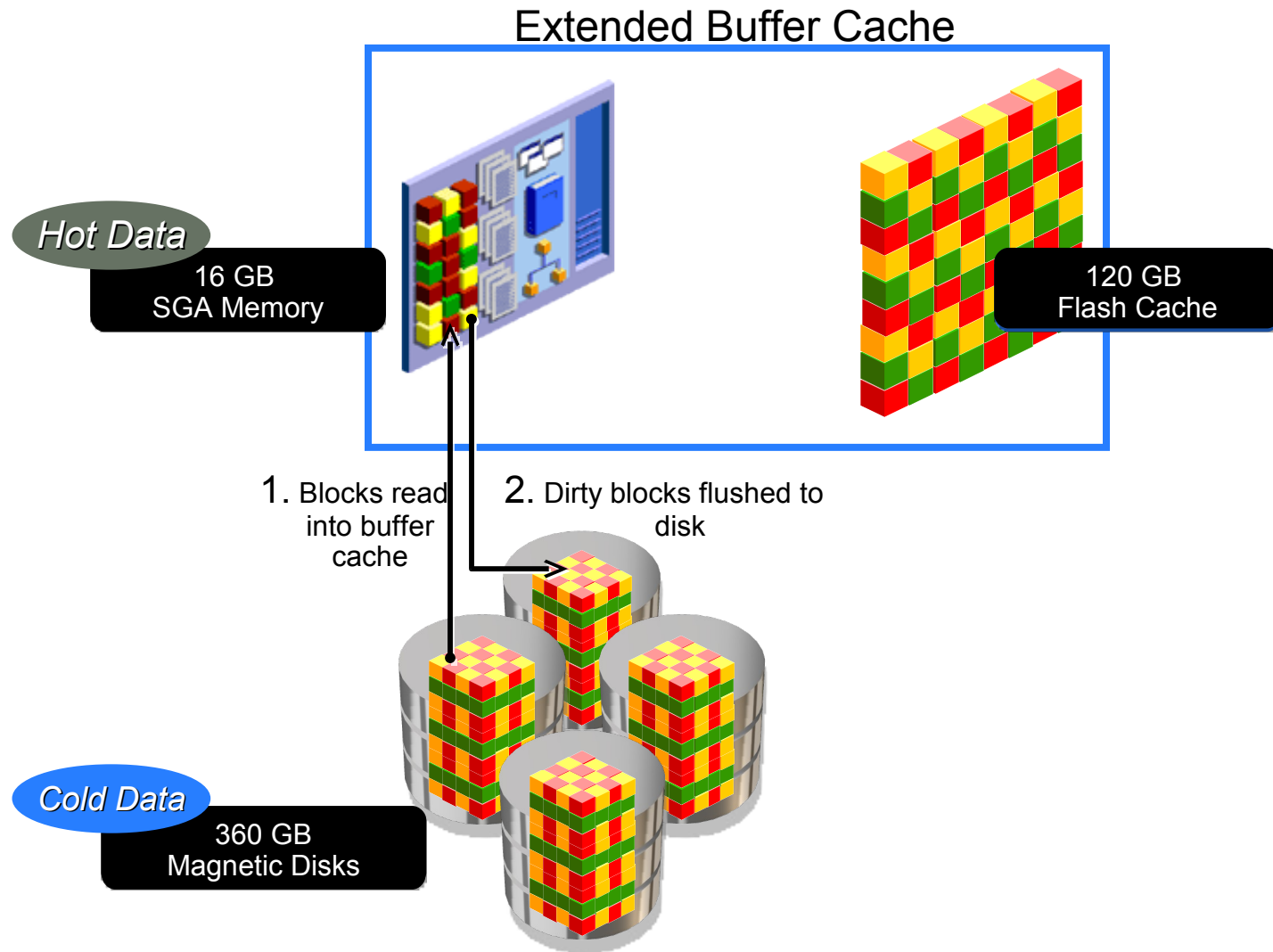
# Flash Cache

## How it works



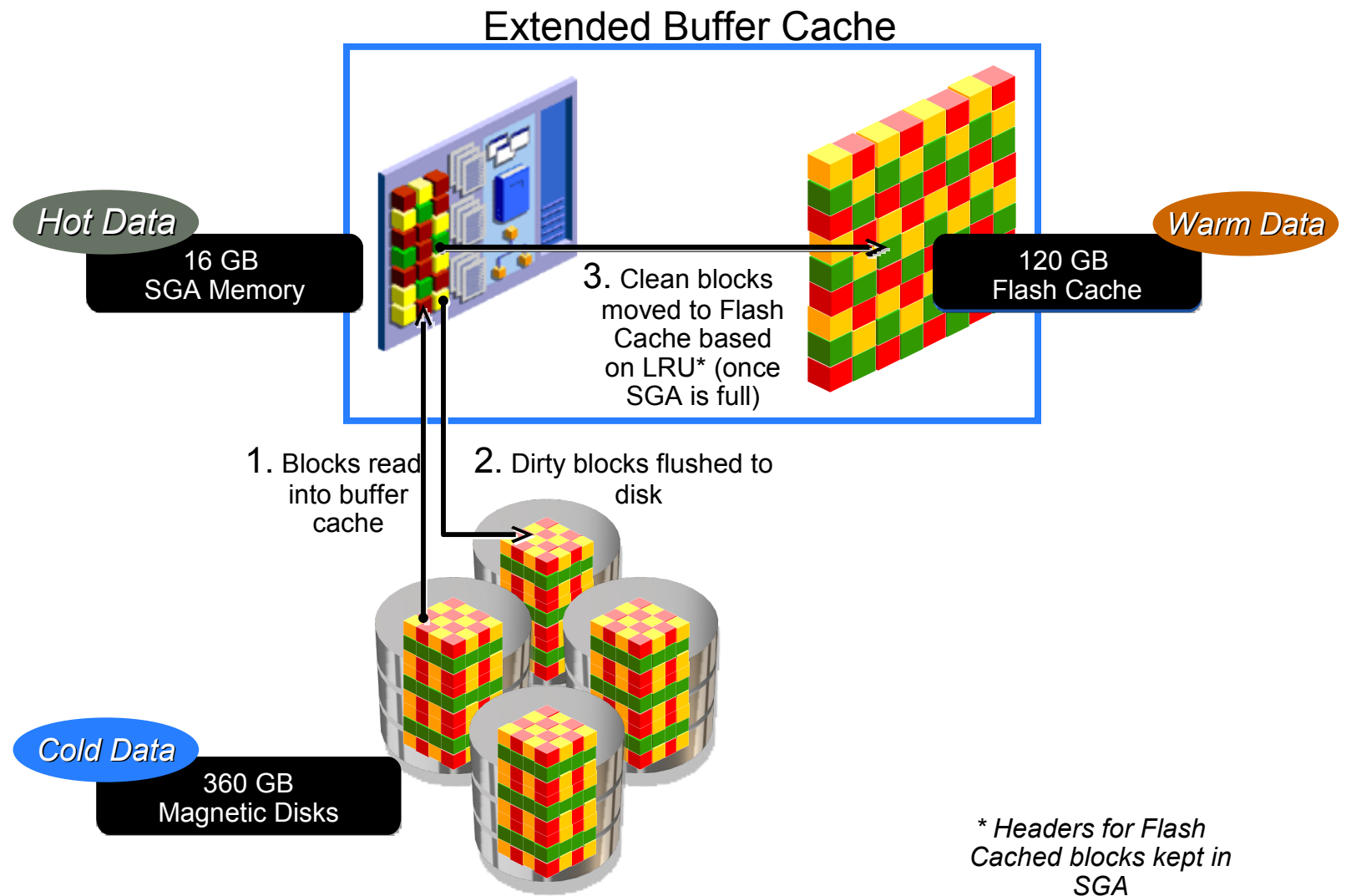
# Flash Cache

## How it works



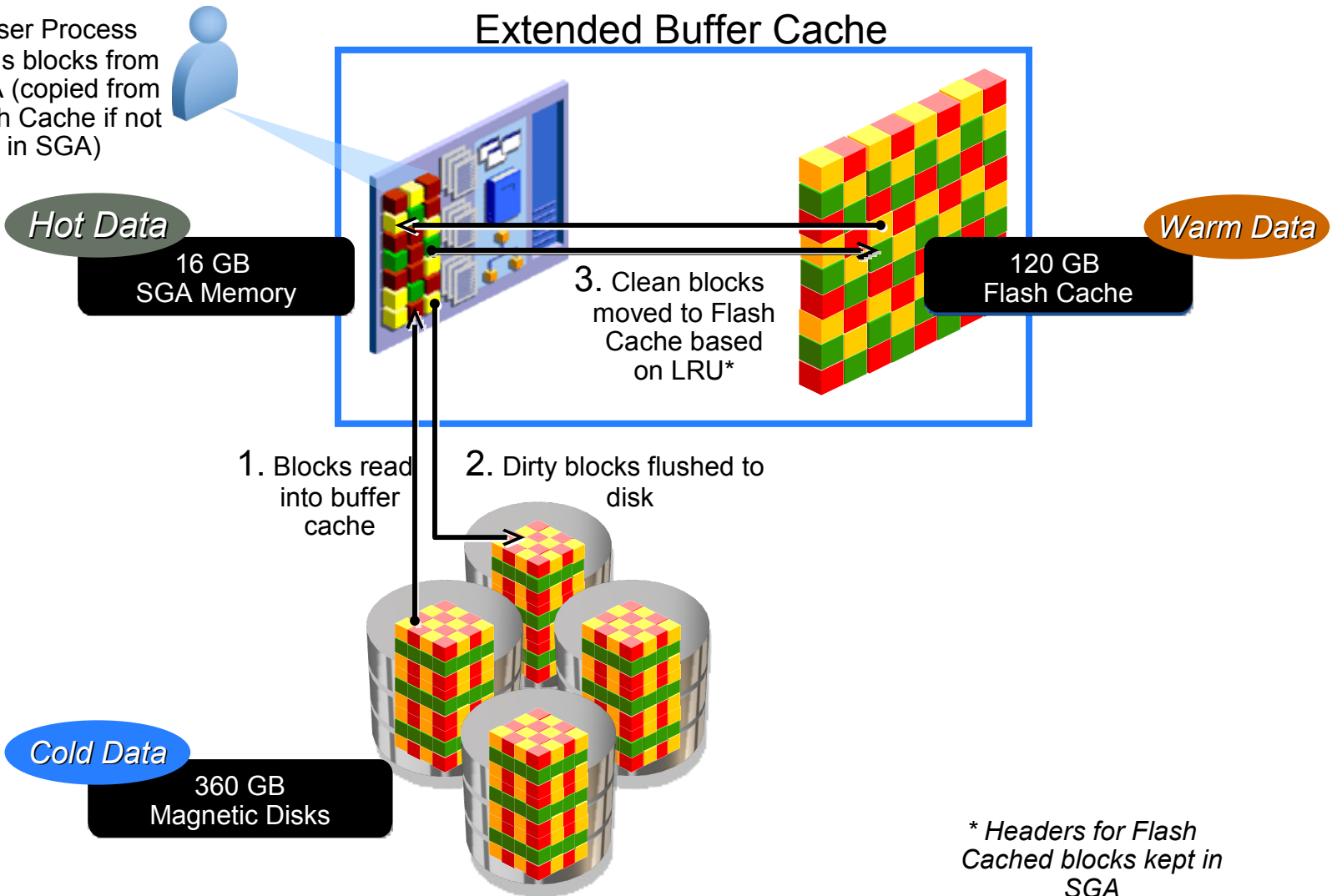
# Flash Cache

## How it works



# Flash Cache

4. User Process reads blocks from SGA (copied from Flash Cache if not in SGA)

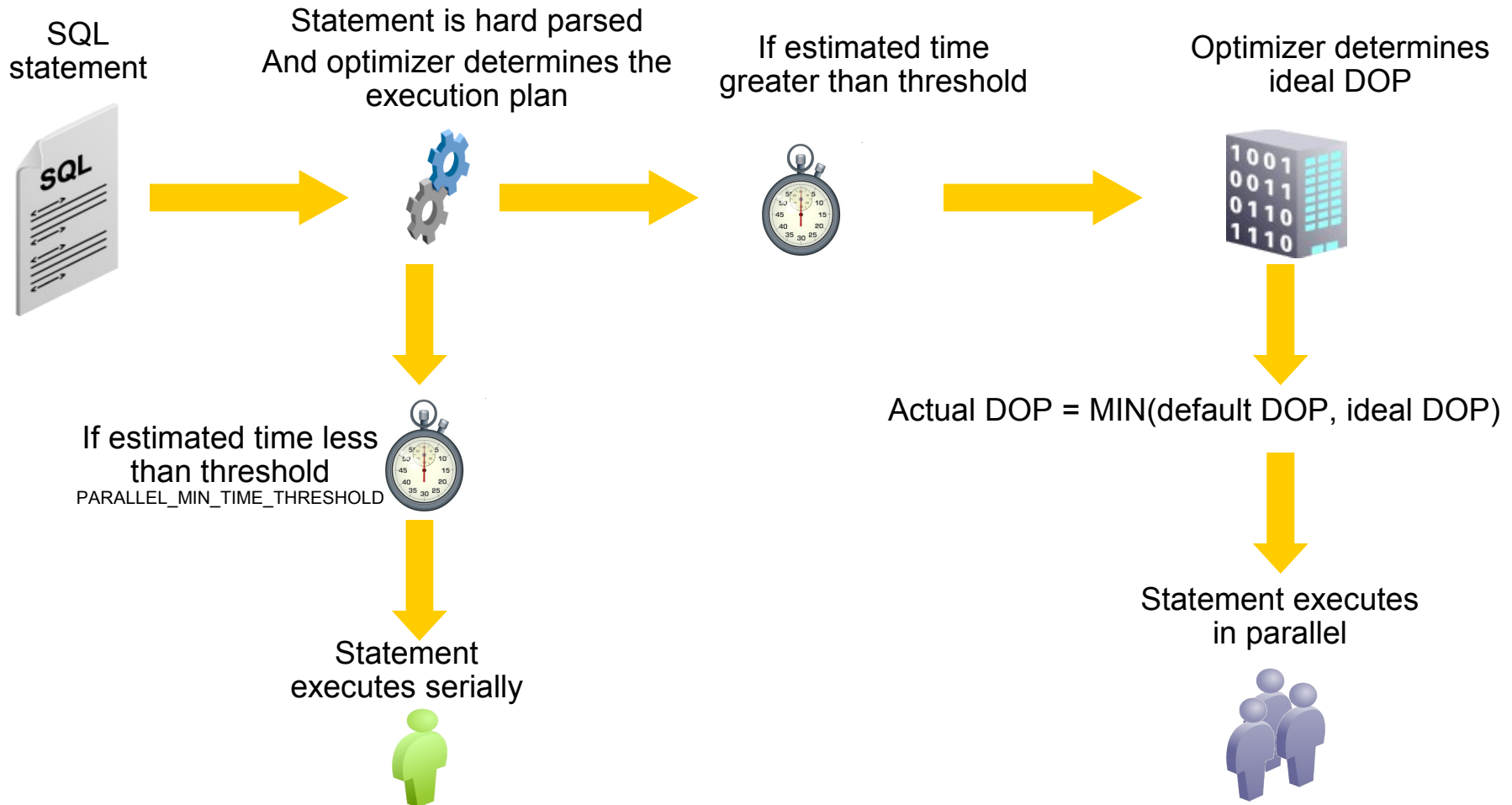




## 9 Parallel Improved

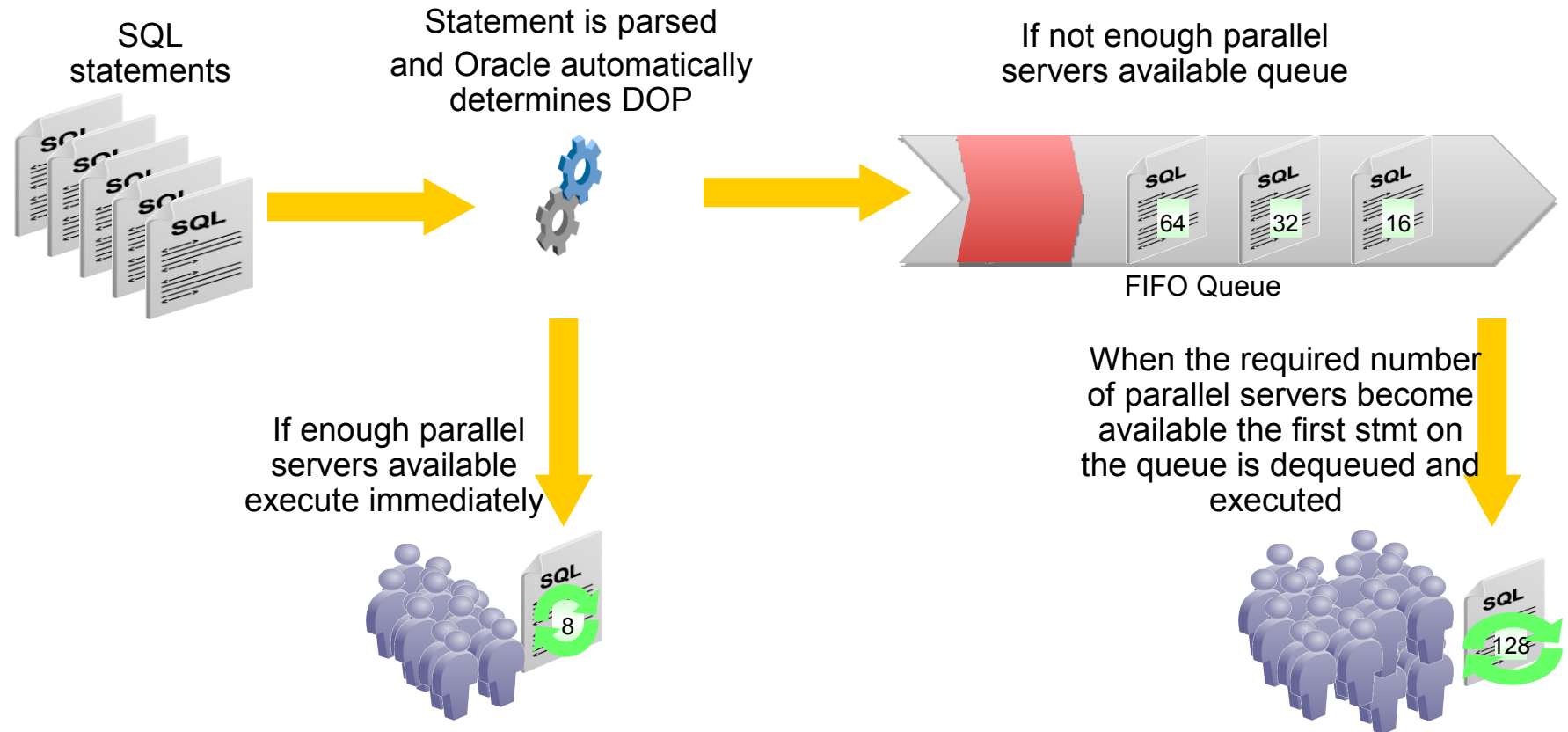
# Automated Degree of Parallelism

## How it works



# Parallel Statement Queuing

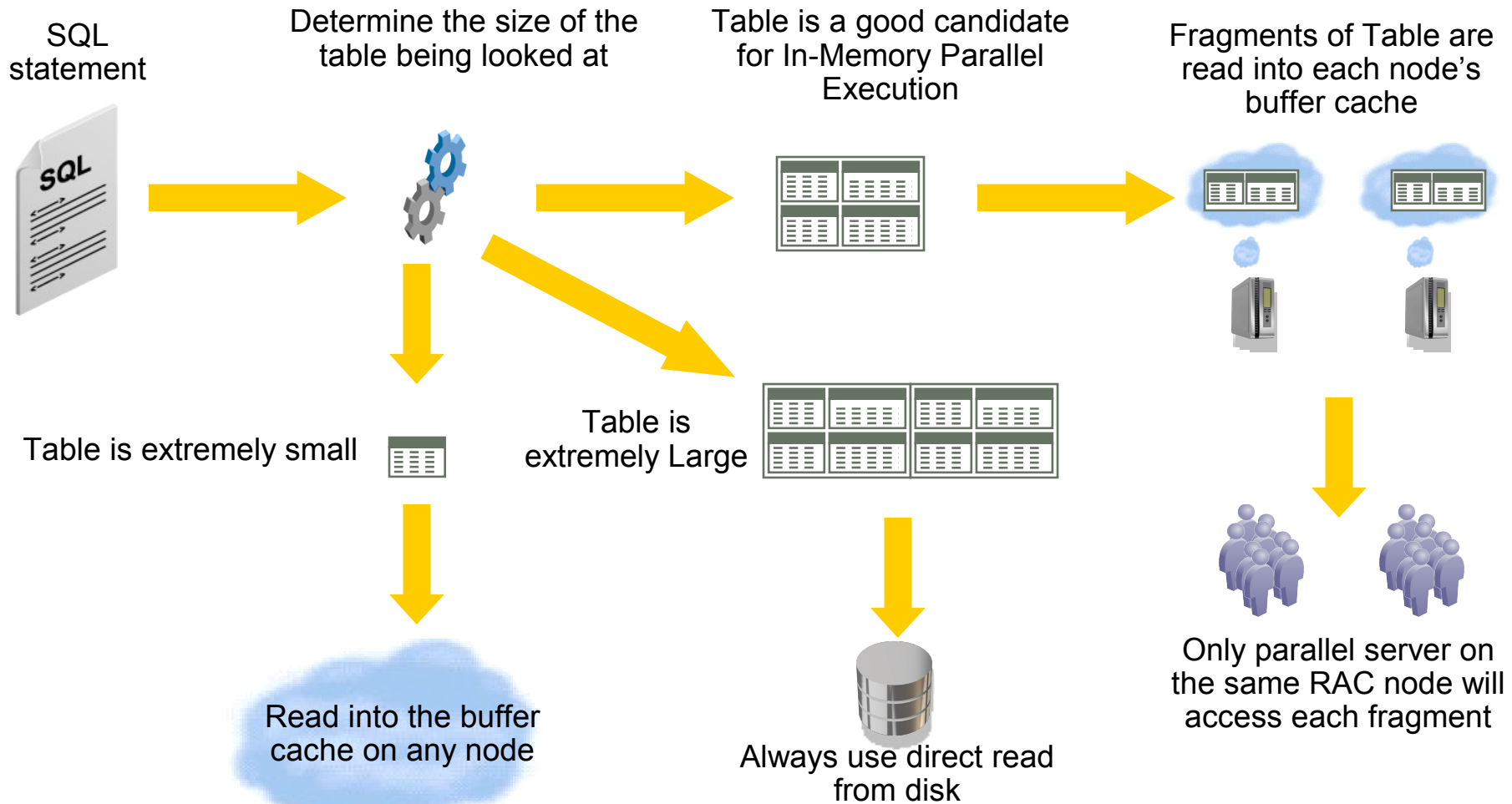
## How it works





# In-Memory Parallel Execution

## How it works





# 10 Edition-based Redefinition



Yes, this is here twice

But only because

It is the *killer feature*

Of Oracle Database 11g Release 2

It is worth 2 features

10+Edition-based  
Redefinition!



**ORACLE®**

**Edition Based Redefinition**

# Why? It is all about online change...

things from the past include

- Online Parameter Changes
- Online Major Memory Changes
- Online Schema Evolution
- Online Index Creates
- Quiesce
- Rolling Upgrades
- Online Disk reconfiguration (ASM)
- Online Cross Platform Tablespace Transport
- Full Database Transports
- And more....

# Oracle Database 11g Release 2

## Online Application Upgrade

- Large, mission critical applications are often unavailable for tens of hours while a patch or an upgrade is installed
- Oracle Database 11g Release 2 introduces revolutionary new capabilities that allow online application upgrade with uninterrupted availability of the application
- The pre-upgrade application and the post-upgrade application can be used concurrently

# Online Application Upgrade

## Requirements

- The installation of the upgrade into the production database must not perturb live users of the pre-upgrade application
  - Many objects must be changed in concert. The changes must be made in privacy
- Transactions done by the users of the pre-upgrade application must be reflected in the post-upgrade application
- For hot rollover, we also need the reverse of this:
  - Transactions done by the users of the post-upgrade application must be reflected in the pre-upgrade application

# The solution

## Edition-based redefinition

- Revolutionary new features:
  - edition
  - editioning view
  - crossedition trigger
- Code changes are installed in the privacy of a new edition
- Data changes are made safely by writing only to new columns or new tables not seen by the old edition
  - An editioning view exposes a different projection of a table into each edition to allow each to see just its own columns
  - A crossedition trigger propagates data changes made by the old edition into the new edition's columns, or (in hot-rollover) vice-versa



# The challenge

## Application versioning

- Scenario
  - The application has 1,000 mutually dependent tables, views, PL/SQL units, and other objects
  - There's more than one schema
  - They refer to each other by name – and often by schema-qualified name
  - The upgrade needs to change 10 of these

# The challenge

## Application versioning

- Can't change the 10 objects in place because the other 990 refer to them – and this would require changing the pre-upgrade application
- Through 11.1 object determined by name and owner
- naming mechanisms through 11.1 are not rich enough to support online application upgrade

# The solution

## Editions

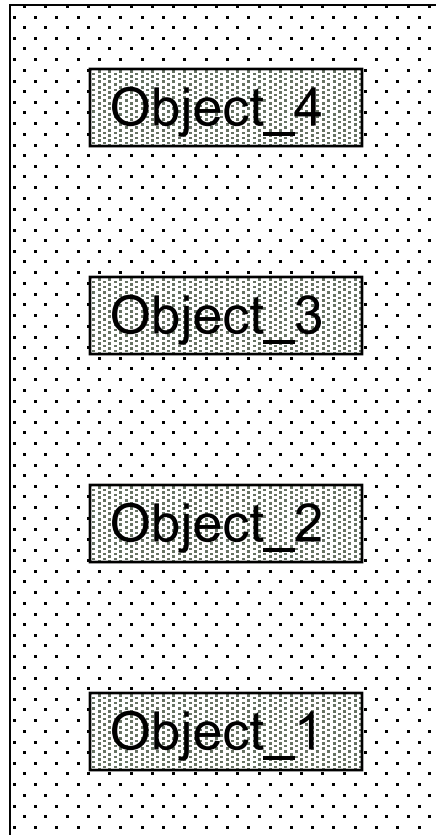
- Edition - new nonschema object type in 11.2
- An edition is the child of an existing edition
  - An edition inherits all objects from its parent
  - Each edition can have its own private occurrence of “the same” object
  - An edition can’t have more than one child
- A database must have at least one edition
- A database session always has a current edition

# The solution

## Editions

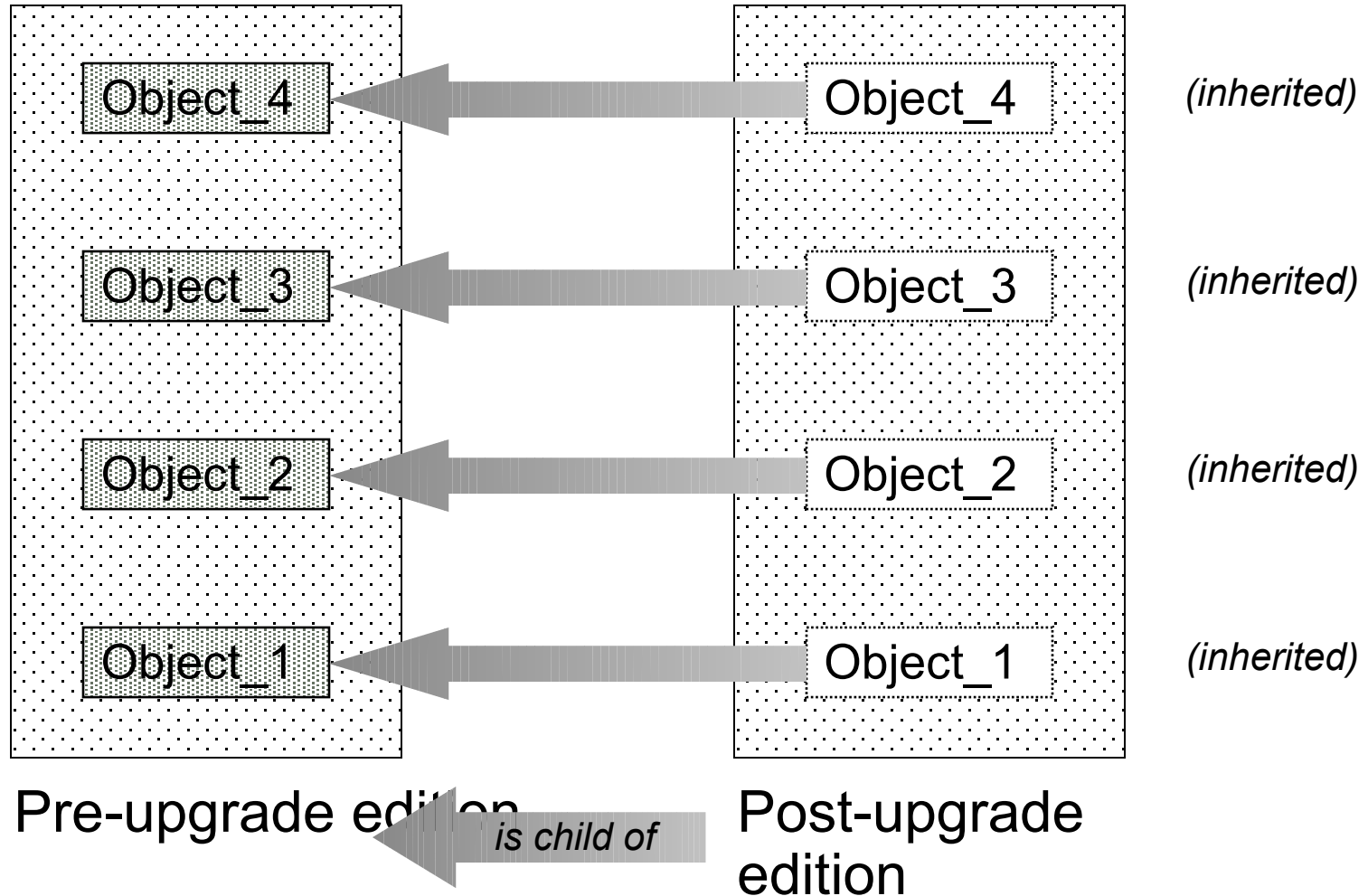
- From 11.2 editioned objects identified by name, owner and the edition in which it was created
- Name and owner are interpreted in the context of a current edition of the current session
- Visibility rules: find the object with the given name and owner in the closest ancestor edition

# Editions & object visibility

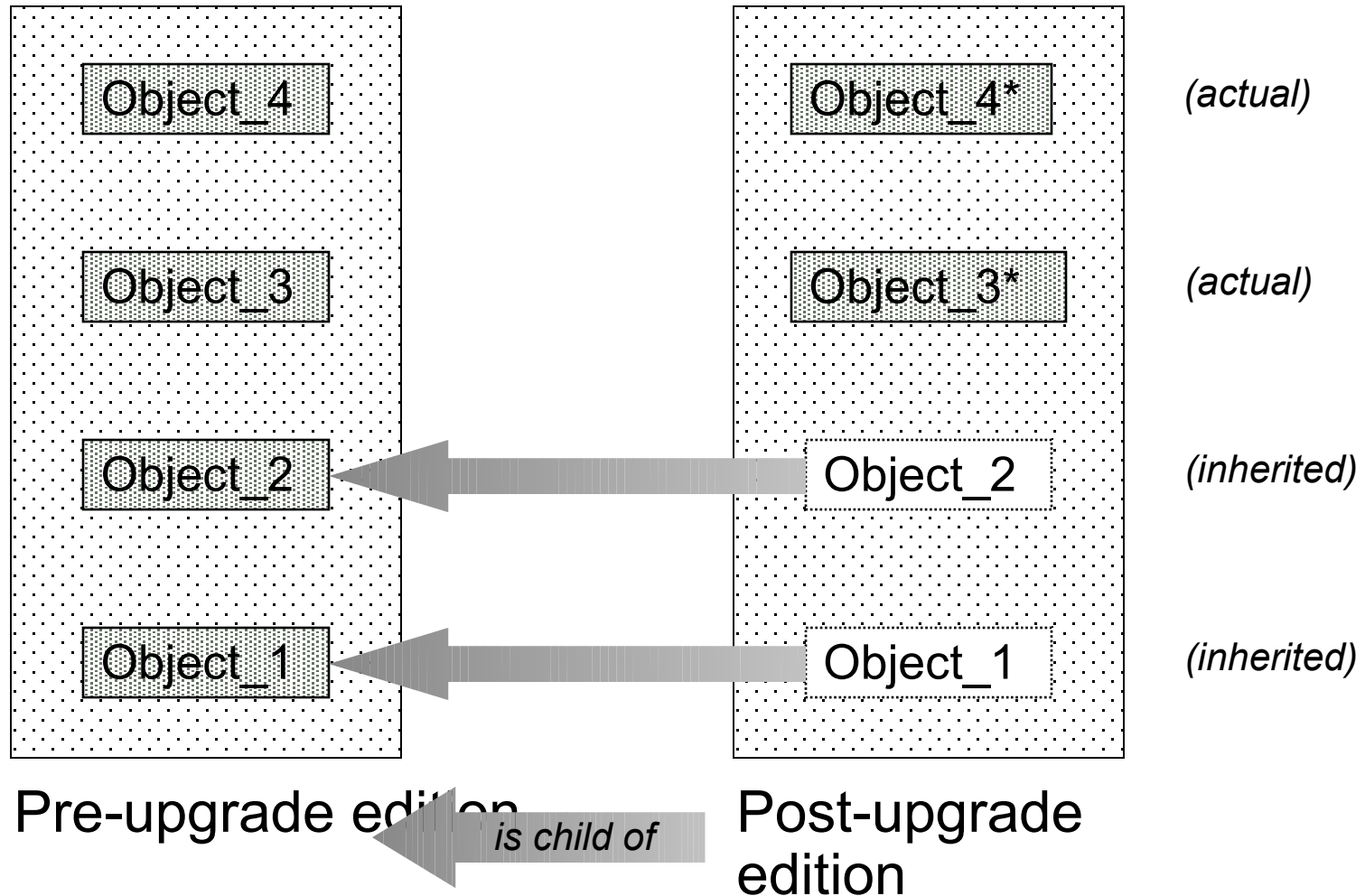


Pre-upgrade edition

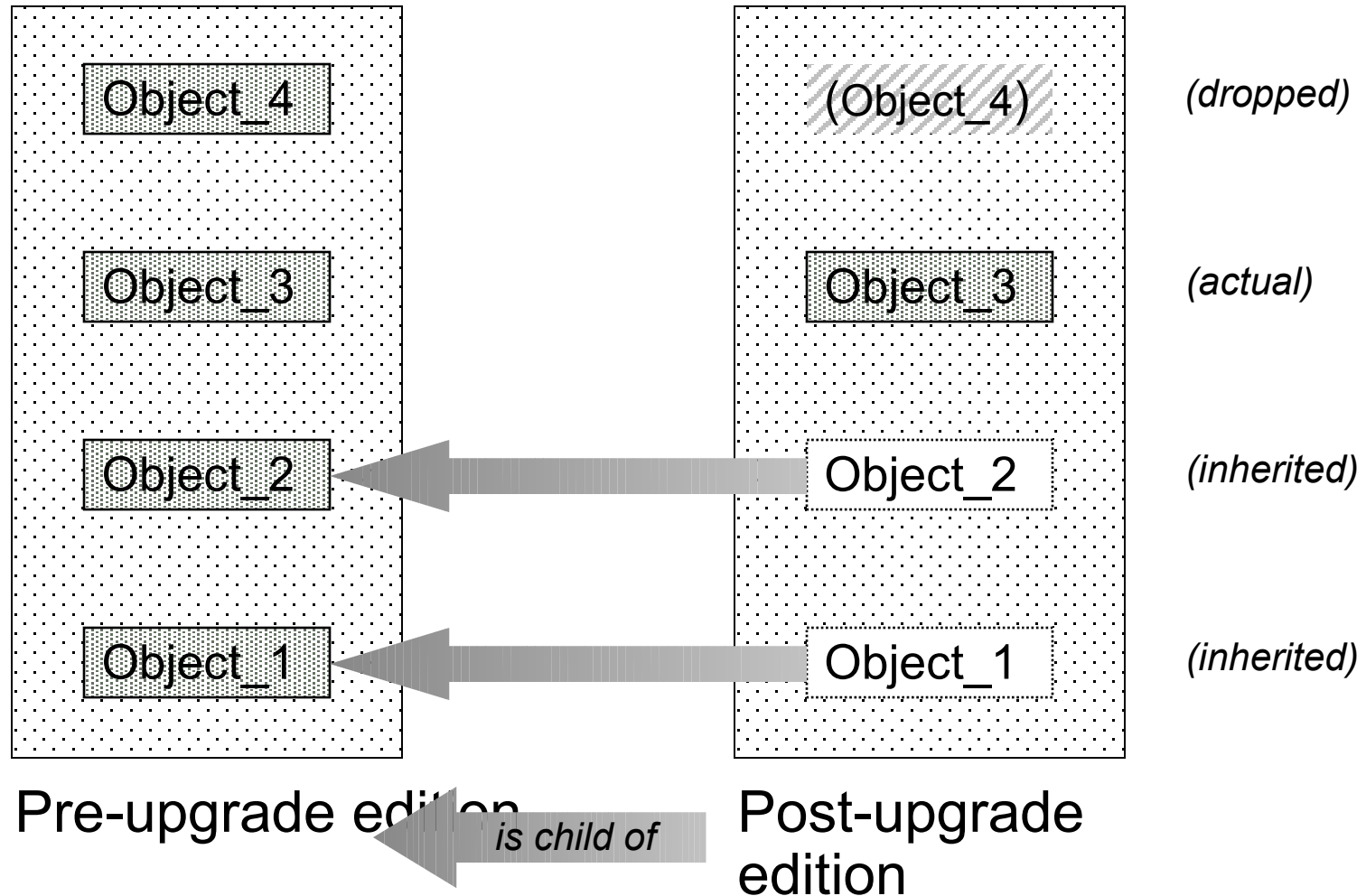
# Editions & object visibility



# Editions & object visibility



# Editions & object visibility





# Editions

- If your upgrade needs only to change synonyms, views, or PL/SQL units, you now have all the tools you need



DEMONSTRATION

# Edition-based Redefinition

ebr.sql

# Editable and noneditable object types

- Not all object types are editable
  - Synonyms, views, and PL/SQL units of all kinds (including, therefore, triggers and libraries) are editable
  - Objects of all other object types e.g. tables are noneditable
- Version the structure of a table manually
  - Instead of changing a column, you add a replacement column
  - Then you rely on the fact that a view is editable

## Editing views – dealing with data across an Application upgrade

- An editing view may only project and rename columns
- The EV must be owned by the table's owner
- Application code should refer only to the logical world
- Like all views, an editing view can be read-only
- If you can tolerate only read access to the underlying data for an editing view that the upgrade will change, you now have all the tools you need

## What if data is changing during upgrade?

- If schema changes are required then both old and new structures must be maintained during upgrade process
- Triggers have the ideal properties to do this safely
- Each trigger must fire appropriately to propagate changes to pre-upgrade columns into the post-upgrade columns – and vice versa

# The solution

## Crossedition triggers

- Crossedition triggers directly access the table.
- The new crossedition trigger has special firing rules
- You create crossedition triggers in the Post\_Upgrade edition
  - The paradigm is: don't interfere with the Pre\_Upgrade edition
- The firing rules rules assume that
  - Pre-upgrade columns are changed – by ordinary application code – only by sessions using the Pre\_Upgrade edition
  - Post-upgrade columns are changed only by sessions using the Post\_Upgrade edition

# The solution

## Crossedition triggers

- A forward crossedition trigger is fired by application DML issued by sessions using the Pre\_Upgrade edition
- A reverse crossedition trigger is fired by application DML issued by sessions using the Post\_Upgrade edition
- The SQL that a crossedition trigger issues always executes in the edition that owns it: the Post\_Upgrade edition

## Case study

- The HR sample schema, as shipped by Oracle Corp, has a single column for phone number
  - Diana Lorentz      590.423.5567
  - John Russell      011.44.1344.429268
- It must be split into two new columns
  - Country code
  - Number within country
  - Diana Lorentz      + 1-590-423-5567
  - John Russell      + 44-134-442-9268

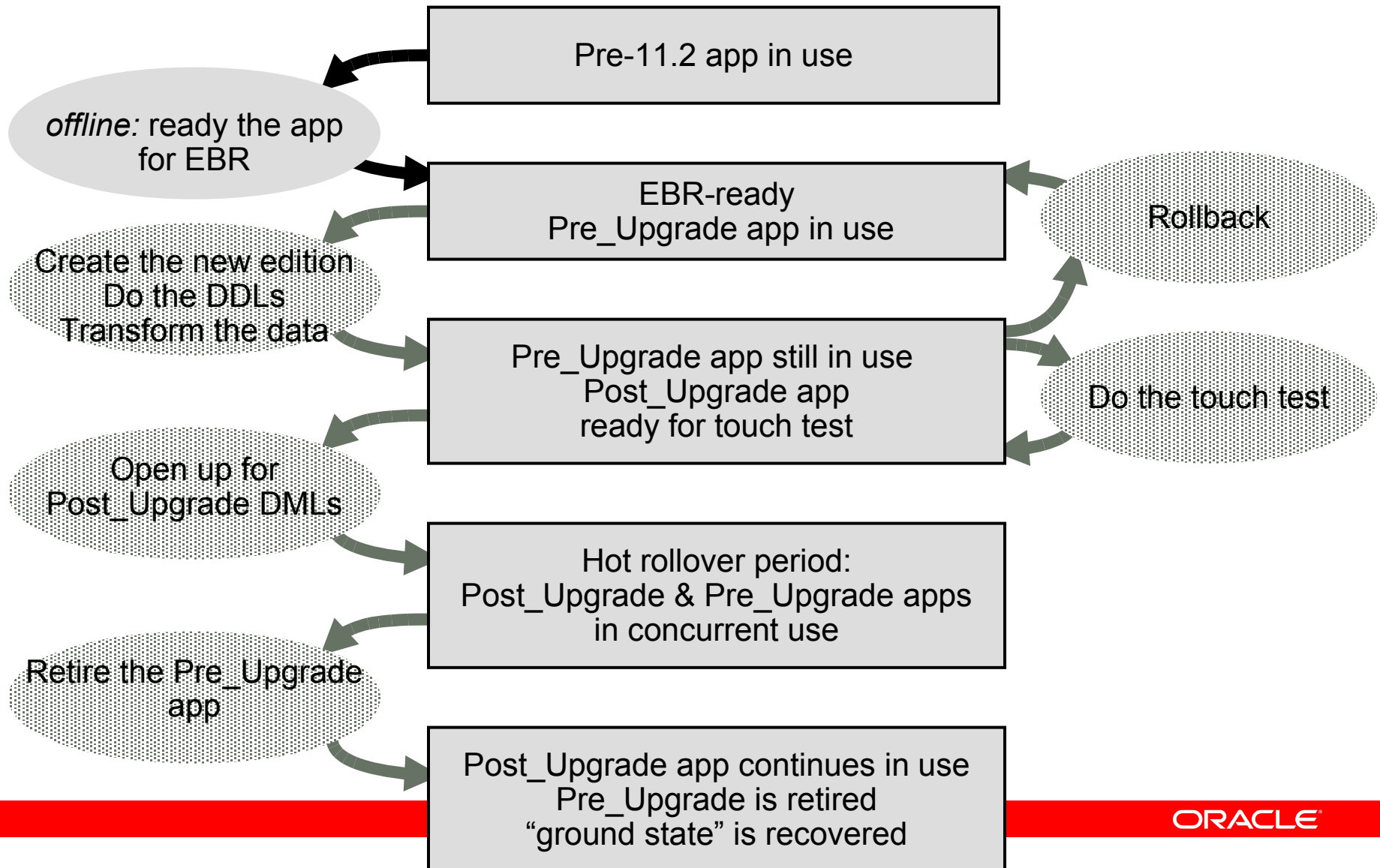


# Case study

First Name	Last Name	Phone
Steven	King	011.32.242.647.4719
Neena	Kochhar	708.108.8233
Lex	De Haan	205.621.9819
Alexander	Hunold	011.38.209.317.1291

First Name	Last Name	Cntry #
Steven	King	+32 242-647-4719
Neena	Kochhar	+1 708-108-8233
Lex	De Haan	+1 205-621-9819
Alexander	Hunold	+38 209-317-1291
Bruce	Ernst	+1 431-800-6569
David	Austin	+1 207-718-4492
Valli	Pataballa	+45 662-851-6340
Diana	Lorentz	+47 951-260-7204
Nancy	Greenberg	+1 434-531-4024
Daniel	Faviet	+1 665-985-7057
John	Chen	+1 189-665-7741
Ismael	Sciarra	+1 235-670-4647
Jose Manuel	Urman	+1 782-701-7504
Luis	Popp	+40 224-526-8013
Den	Raphaely	+35 385-817-3247
Alexander	Khoo	+1 754-359-1977
Shelli	Baida	+36 949-238-6312
Sigal	Tobias	+40 440-870-9983
Guy	Himuro	+1 550-189-7906
Karen	Colmenares	+1 411-573-4711

# EBR task flow – summary



## Pre-11.2 design

- Application code accesses tables directly, in the ordinary way



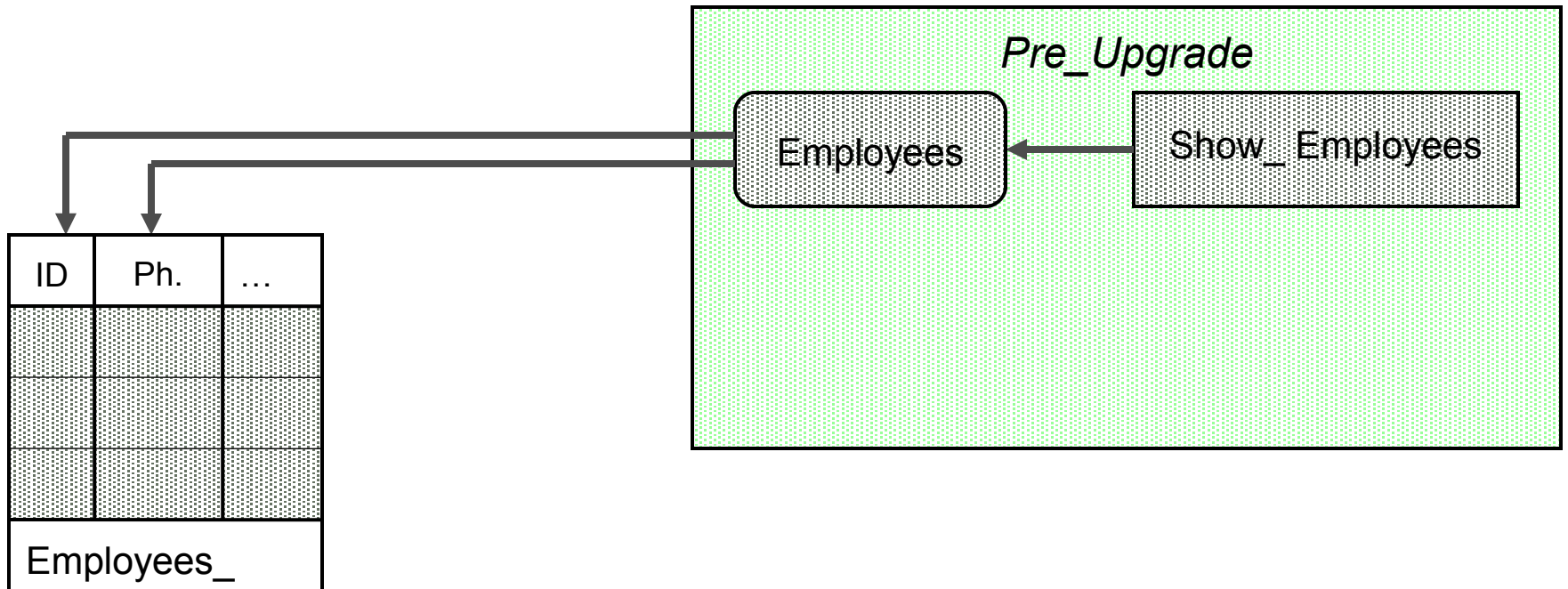
# Readying the application for editions

- “Slide in” an editioning view in front of every table
  - Rename each table to an obscure but related name (e.g. an exotic name that ends with underscore)
  - Create an editioning view for each table that has the same name that the table originally had
  - “Move” triggers to the editioning views ... (next slide)
  - Revoke privileges from the tables and grant them to the editioning views
  - Move VPD policies to the editioning views
- There may be a need for schema reorganization to accommodate the rule that a noneditioned object cannot depend on an editioned object

# Readying the application for editions

- Of course,
  - All indexes on the original Employees remain valid but User\_Ind\_Columns now shows the new values for Table\_Name and Column\_Name
  - All constraints (foreign key and so on) on the original Employees remain in force for Employees\_
- However,
  - Triggers don't fully "follow" the rename – and anyway that's not what you want
  - Rather, just drop the trigger and re-run the original create trigger statement to "move" the trigger onto the editioning view

Starting point.  
Pre-upgrade app in normal use.

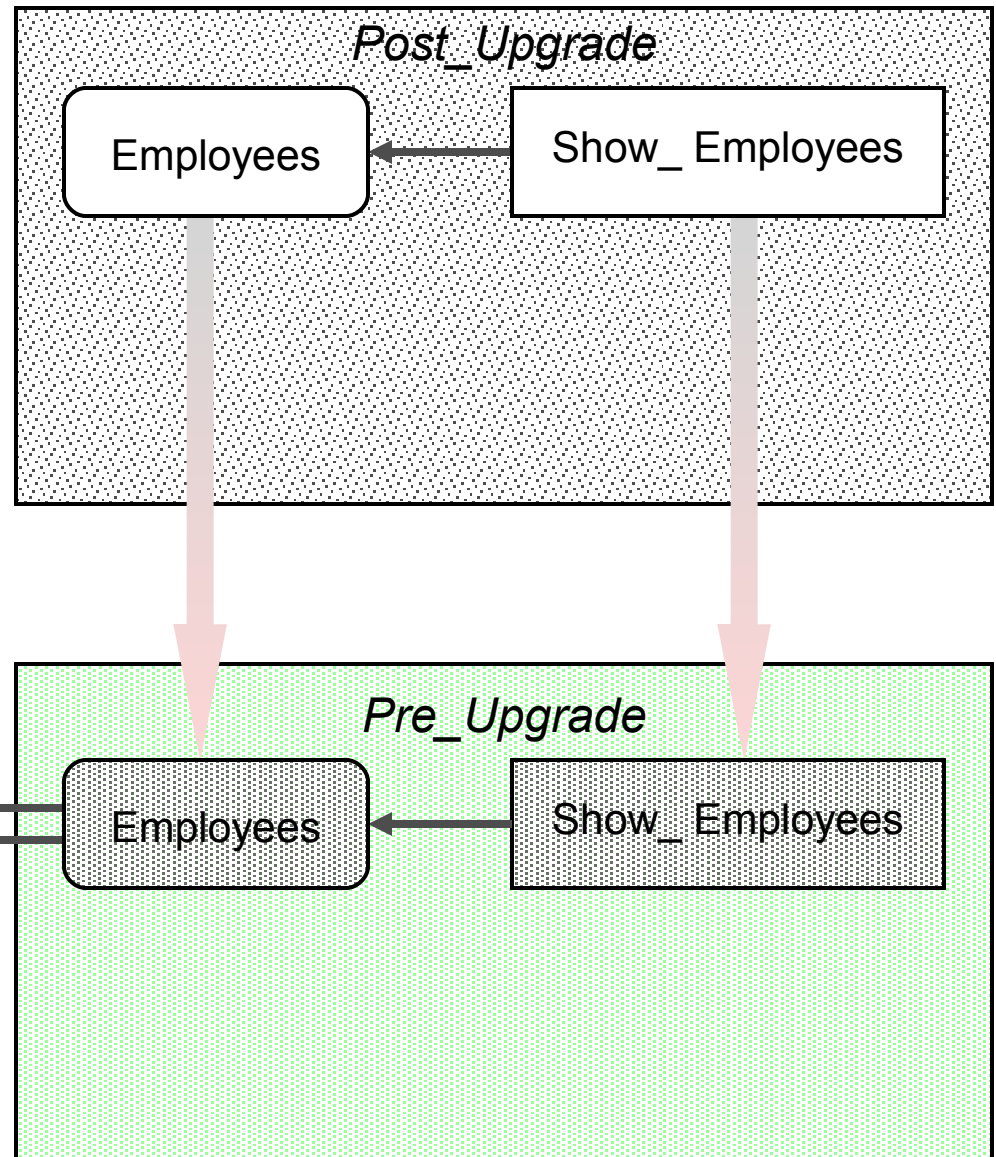


Start the edition-based redefinition exercise.

Create the new edition as the child of the existing one.

This is fast because initially all the editioned objects are just inherited.

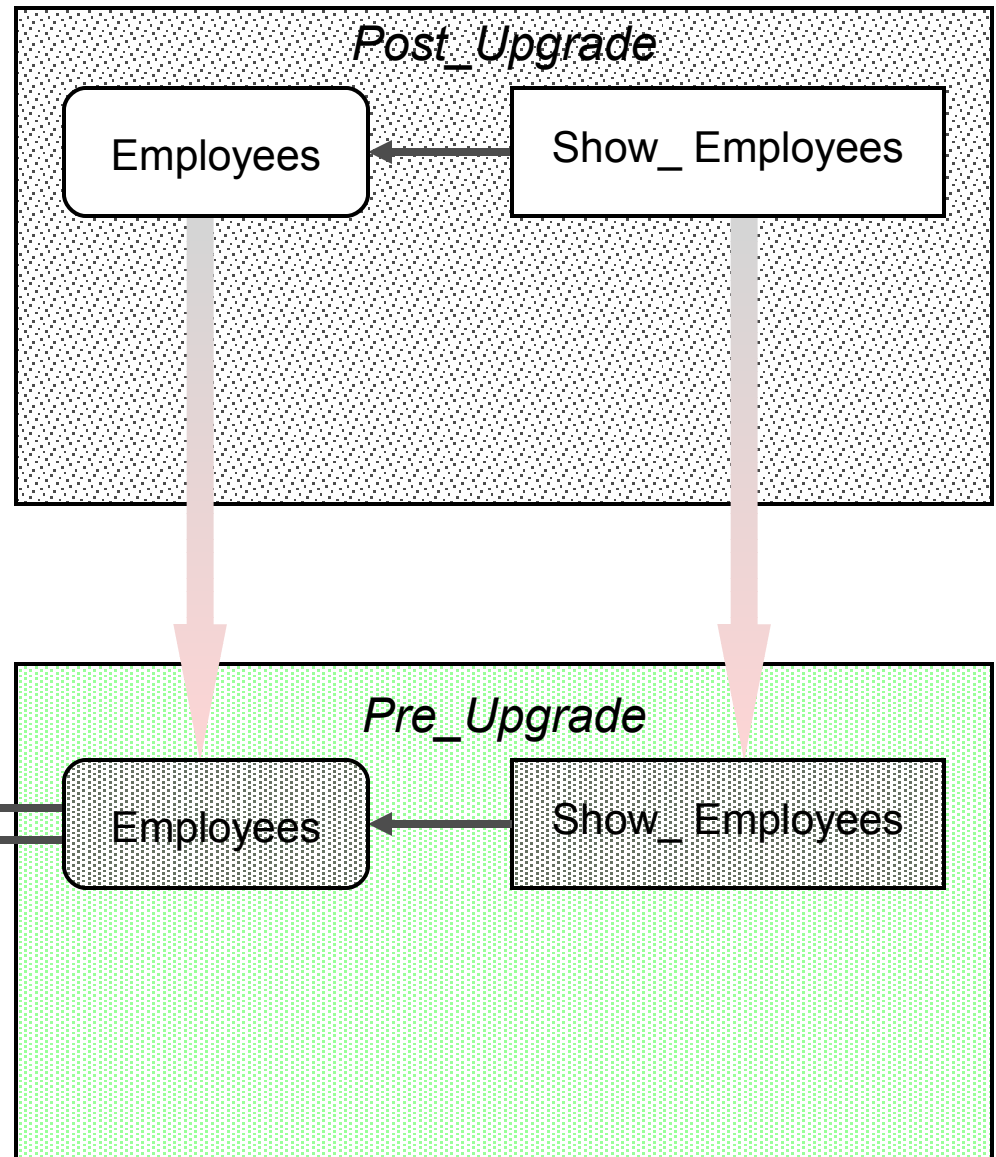
ID	Ph.	...
Employees_		



Create the replacement columns  
in the underlying table.

The editioning view shields  
the app from this change.

ID	Ph.	...	Cntry	#
Employees_				

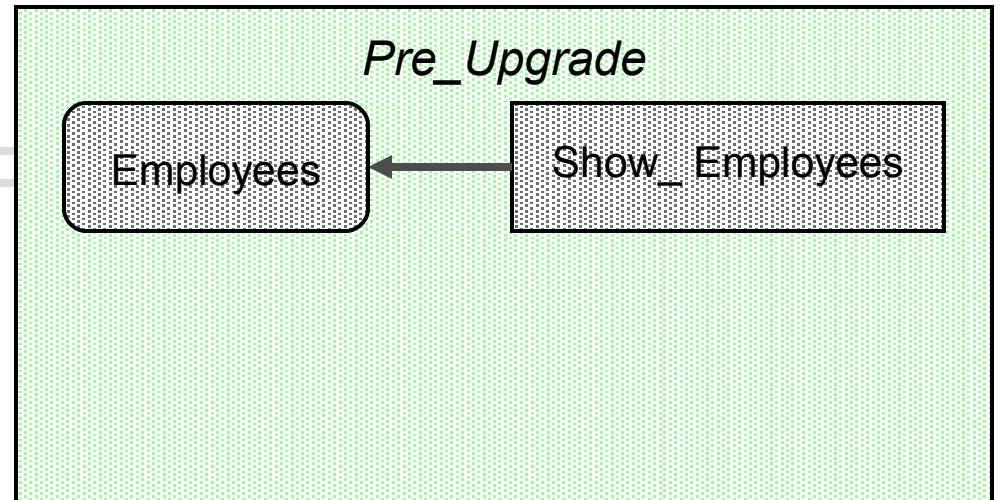
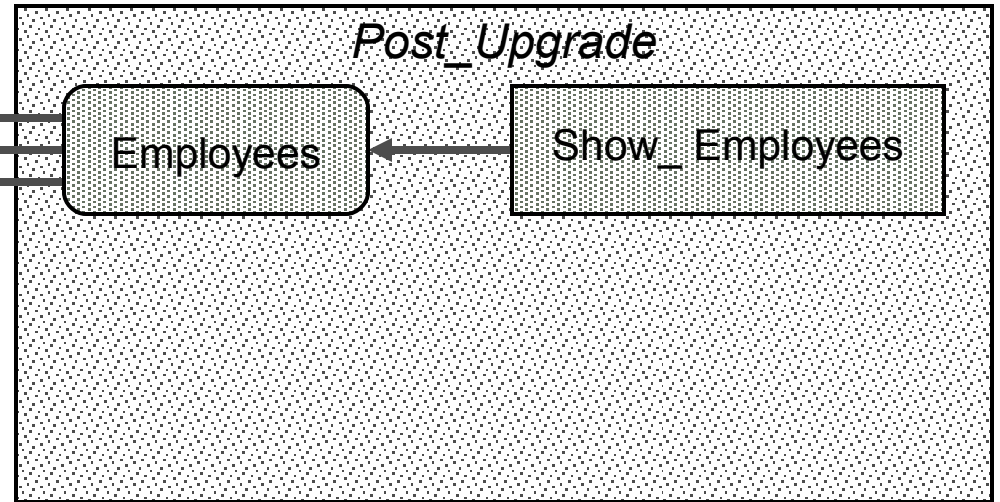


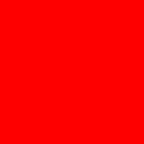


Change *Employees* to select the new columns.

Change *Show\_Employees* to implement the new behavior.

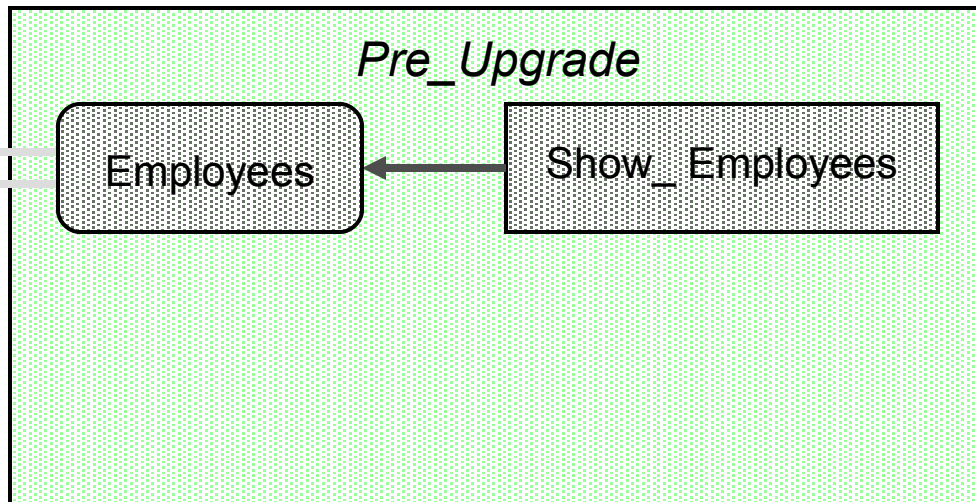
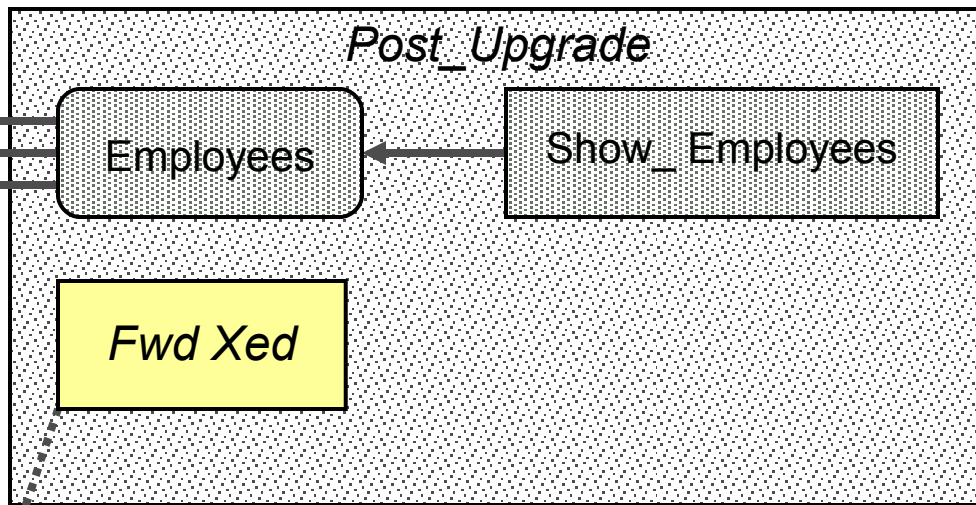
ID	Ph.	...	Cntry	#
Employees_				





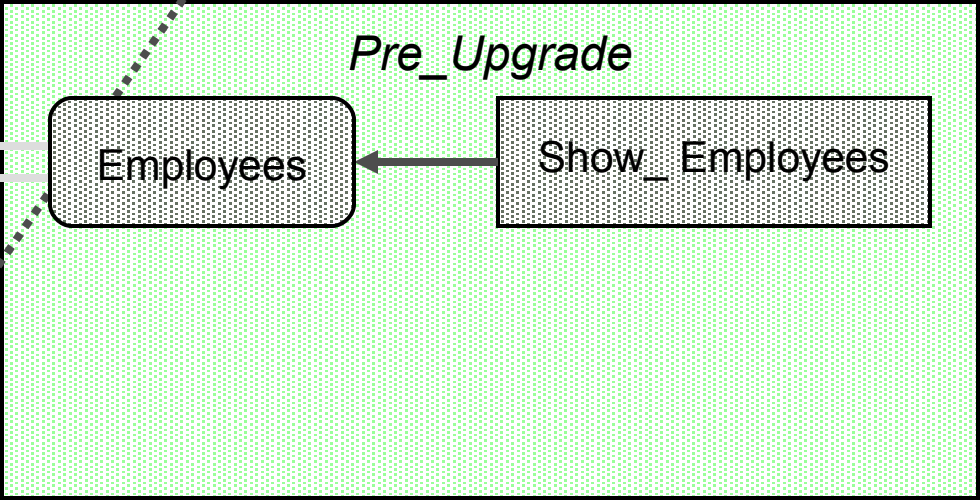
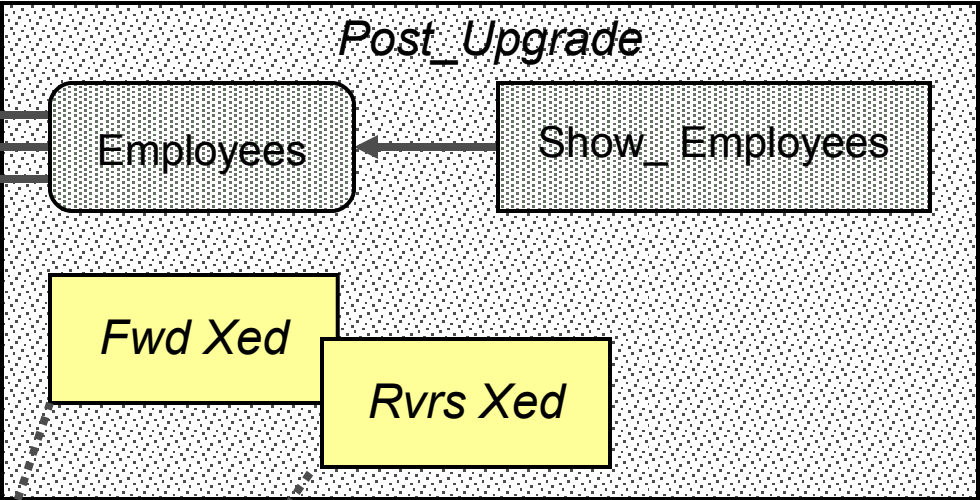
Create the forward crossedition trigger.

ID	Ph.	...	Cntry	#
Employees_				





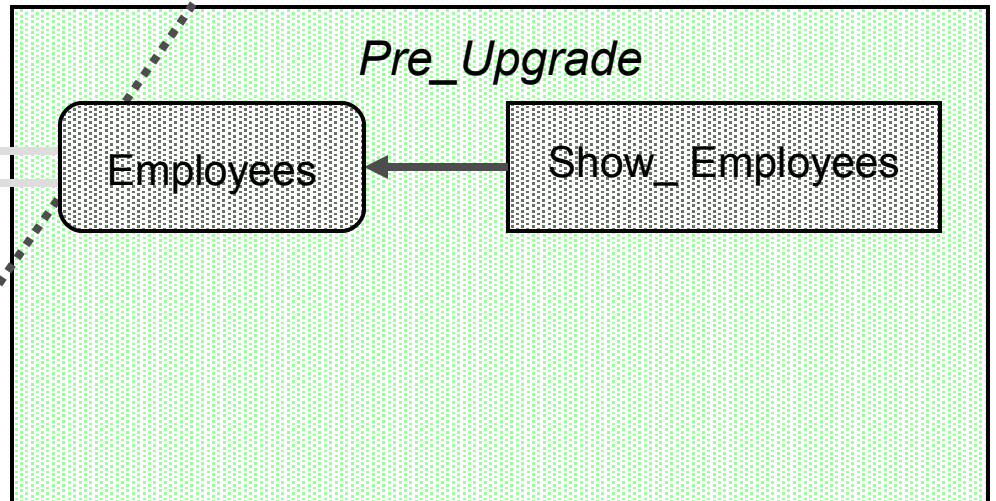
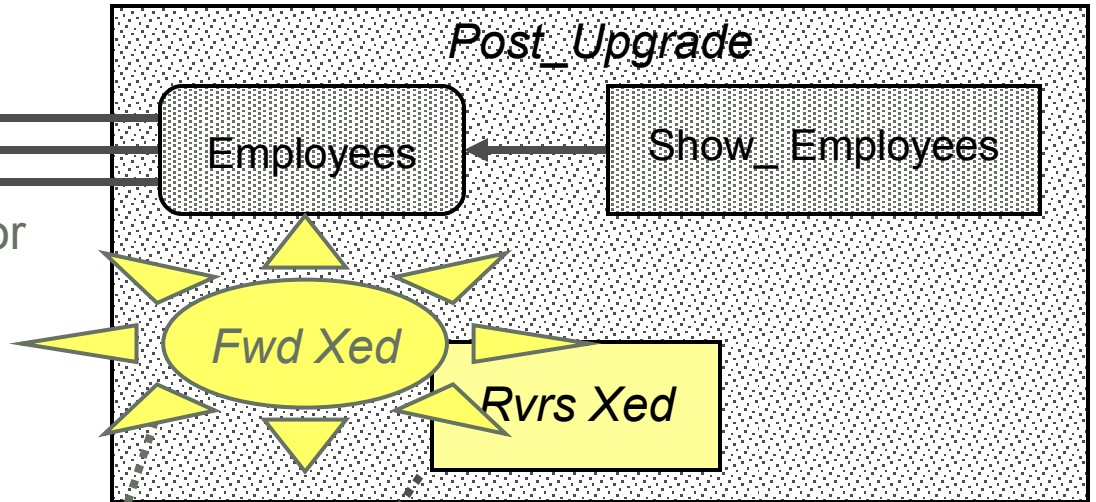
Create the reverse crossedition trigger.

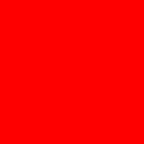


ID	Ph.	...	Cntry	#
Employees_				

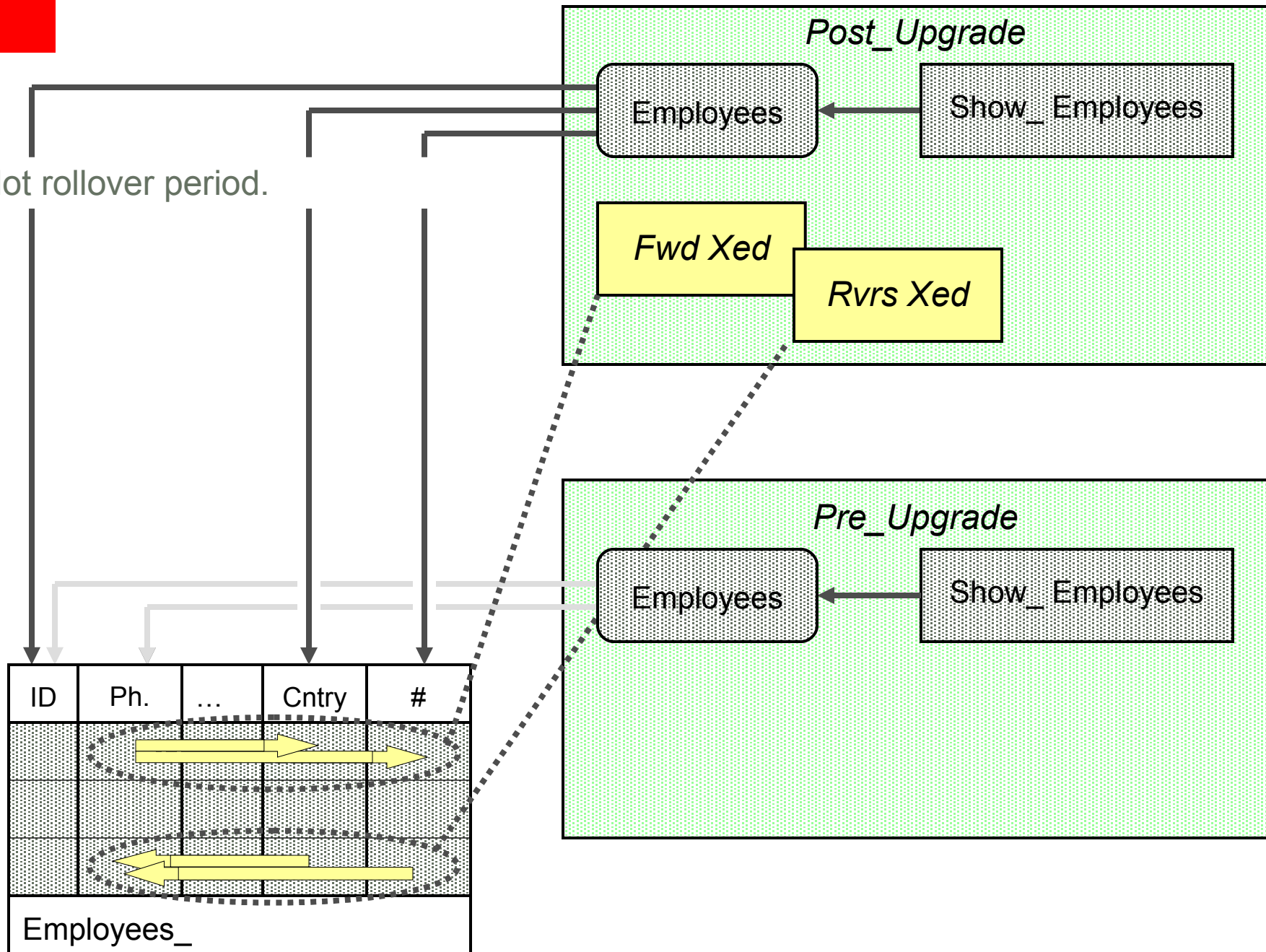
Apply the transform to the data for the new app to use

ID	Ph.	...	Cntry	#
Employees_				





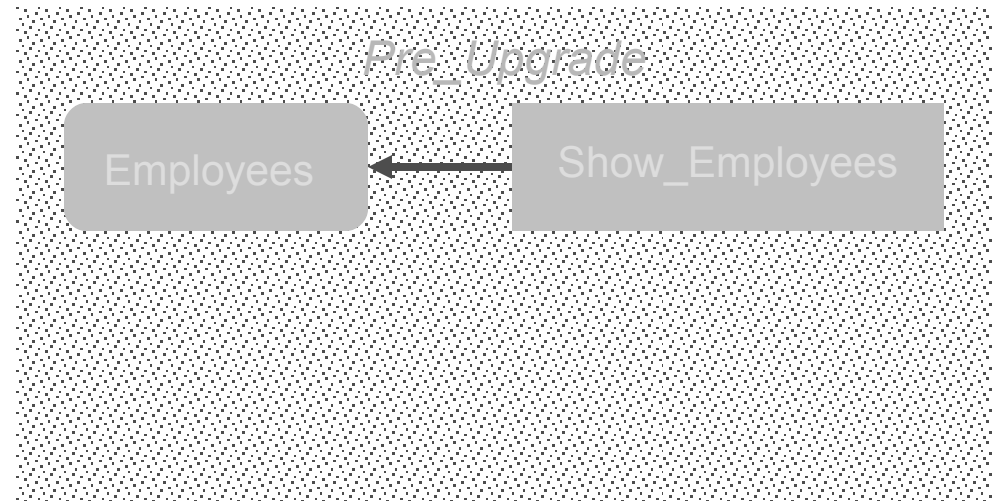
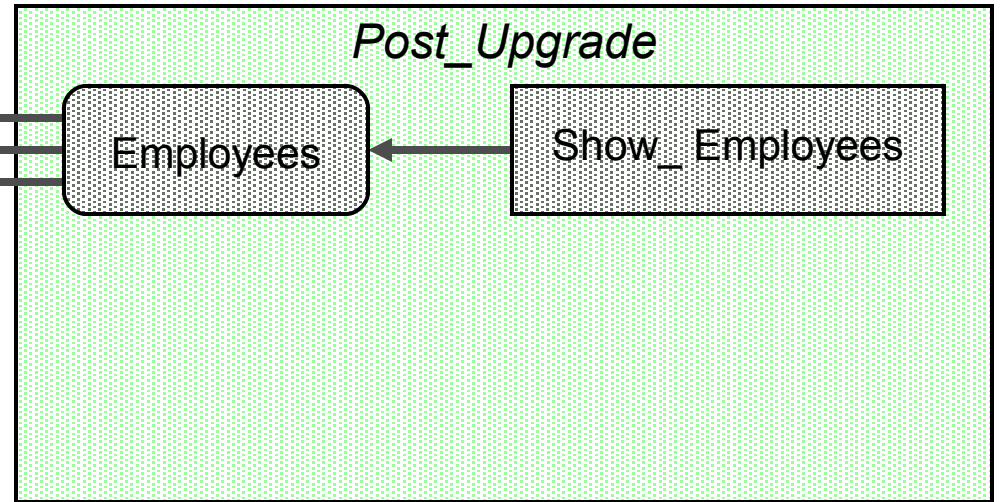
Hot rollover period.



The *Pre\_Upgrade* edition is retired.

The edition-based redefinition exercise is complete.

ID	Ph.	...	Cntry	#
Employees_				



# Edition-based redefinition

- Oracle Database 11g Release 2 brings the edition, the editioning view, and the crossedition trigger
  - Code changes are installed in the privacy of a new edition
  - Data changes are made safely by writing only to new columns or new tables not seen by the old edition
    - An editioning view exposes a different projection of a table into each edition to allow each to see just its own columns
    - A crossedition trigger propagates data changes made by the old edition into the new edition's columns, or (in hot-rollover) vice-versa

# Edition-based Redefinition

## Things to note ....

- Online application upgrade is a high availability subgoal
- Traditionally, HA goals are met by features that the DBA can choose to use at the site of the deployed application
  - independently of the design of the application
  - without the knowledge of the developer
- The features for online application upgrade are used by developers
  - when preparing the application for online upgrade
  - when implementing an upgrade script
- DBAs, of course, will need to understand the features



# Oracle Database 11g Release 2

## Online Application Upgrade

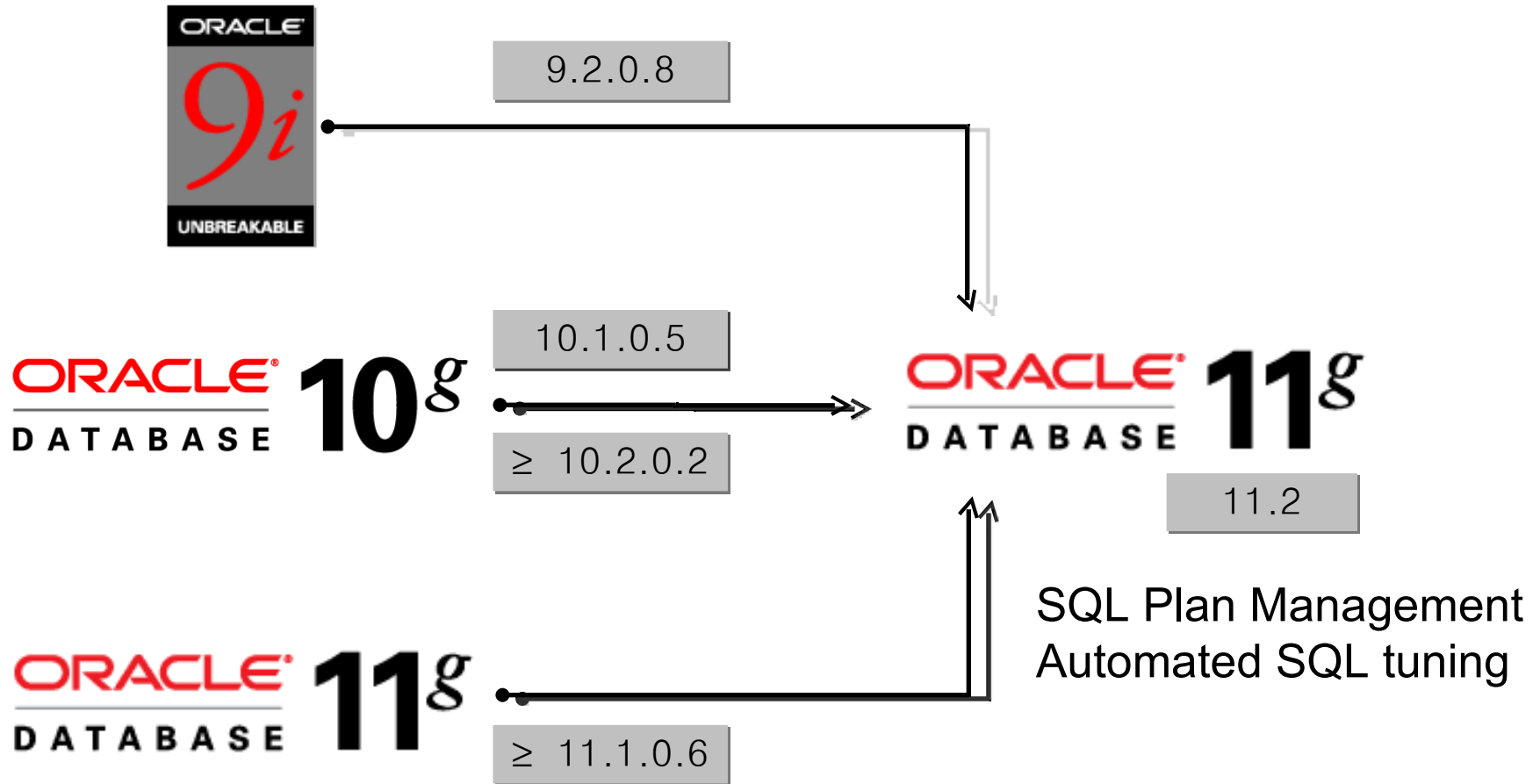
- Large, mission critical applications are often unavailable for tens of hours while a patch or an upgrade is installed
- Oracle Database 11g Release 2 introduces revolutionary new capabilities that allow online application upgrade with uninterrupted availability of the application
- The pre-upgrade application and the post-upgrade application can be used at the same time
- End-user sessions therefore enjoy hot rollover

How to get there



# What are my upgrade paths?

Predictable performance post-upgrade



**For More Information**

search.oracle.com



or

oracle.com

ORACLE®