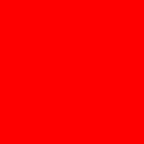


**ORACLE®**

## **Edition-based redefinition: the key to online application upgrade**

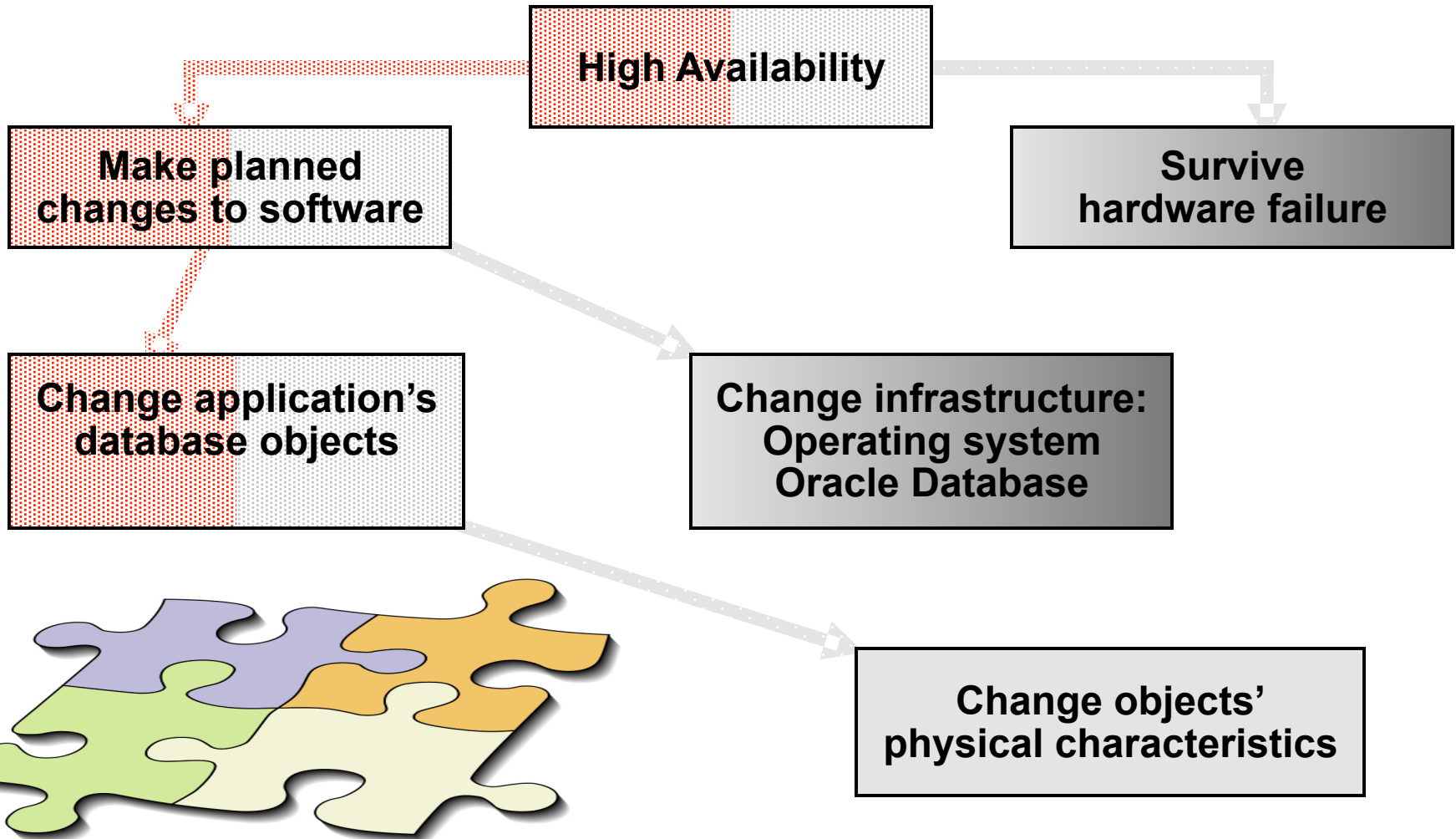
Bryn Llewellyn  
Distinguished Product Manager,  
Database Division, Oracle HQ



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 remain at the sole discretion of Oracle.

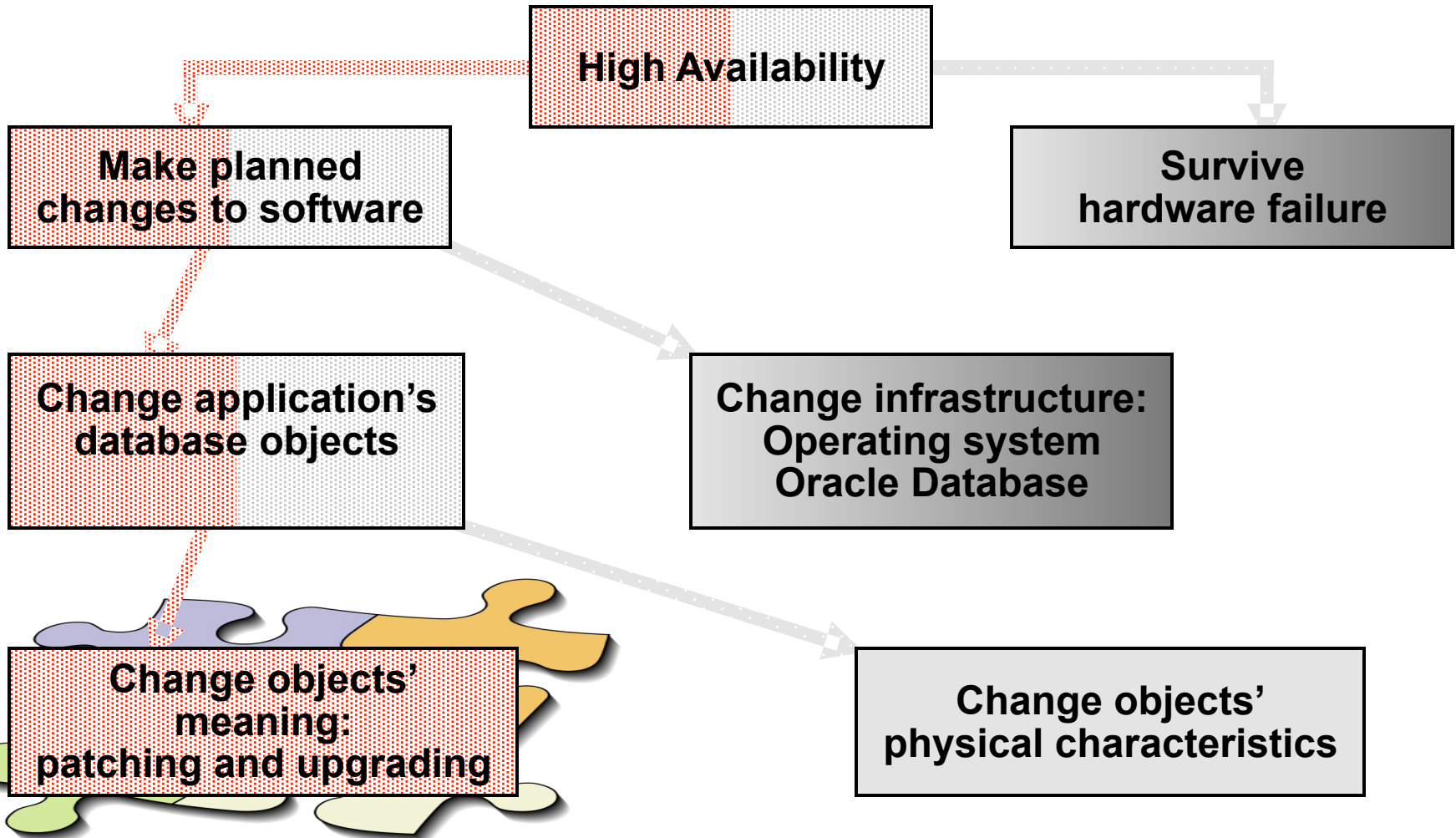
# Online Application Upgrade

– the final piece of the HA jigsaw puzzle



# Online Application Upgrade

– the final piece of the HA jigsaw puzzle



# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- Explanation of the *edition*
- Explanation of the *editioning view*
- Explanation of the *crossedition trigger*
- Readyng an application for EBR
- Case study explained
- Conclusion / Q&A

# Scope

- This presentation explains capabilities, new in Oracle Database 11.2, and enhanced in 12.1, that support online upgrade of the database tier of an application
- The online upgrade of other tiers of the application will need their own specific solutions – not discussed in this presentation
- The take-away from this presentation is that Oracle Database offers *both* an isolation mechanism to allow pre- and post-upgrade schemas to co-exist, *and* a way for client code to choose the particular isolated environment that it wants

# Agenda

- Scope of this presentation
- **The challenge and the solution stated**
- Case study stated
- Explanation of the edition
- Explanation of the editioning view
- Explanation of the crossedition trigger
- Readyng an application for EBR
- Case study explained
- Conclusion / Q&A

# Online Application Upgrade

- Supporting online application upgrade means maintaining *uninterrupted availability* of the application
- But end-user sessions can last tens of minutes or longer
  - Users of the old app don't want to abandon an ongoing session
  - Users wanting to start a session must use the new app, but cannot wait until no-one is using the old app
- This implies that it must be possible to use the pre-upgrade application and the post-upgrade application at the same time – a.k.a. *hot rollover*



# The challenge

- 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

- EBR brings these key features: 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

# Agenda

- Scope of this presentation
- The challenge and the solution stated
- **Case study stated**
- Explanation of the edition
- Explanation of the editioning view
- Explanation of the crossedition trigger
- Readyng an application for EBR
- Case study explained
- Conclusion / Q&A

# Case study

- The HR sample schema, that ships with Oracle Database, represents phone numbers in a single column:
  - Diana Lorentz      590.423.5567
  - John Russell        011.44.1344.429268
- Users now need to ring phone numbers from any country in the world
- So we want a uniform representation with two columns: Country Code; and Number Within Country.

# Case study

ct 1 Show\_20\_Rows\_Repeat\_Pre\_Upgrade

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
Bruce	Ernst	431.800.6569

ct 2 Show\_20\_Rows\_Repeat\_Post\_Upgrade

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

# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- **Explanation of the *edition***
- Explanation of the editioning view
- Explanation of the crossedition trigger
- Readyng an application for EBR
- Case study explained
- Conclusion / Q&A

# Application versioning: the challenge

- Scenario – for now think only about synonyms, views, and PL/SQL units
  - The application has 1,000 mutually dependent code objects
  - In general, there's more than one schema
  - They refer to each other by name – in general, by schema-qualified name
  - The upgrade needs to change 10 of these

# Application versioning: the challenge

**Pre-upgrade app**

1,000 v1 objects



**Post-upgrade app**

990 unchanged v1 objects  
+  
**10 changed v2 objects**



# Application versioning: the challenge

- Of course, you can't change the 10 objects in place because this would change the pre-upgrade app
- How can an old and a new occurrence of the “same” object co-exist?
- Before EBR, the only dimensions that determine which object you mean, when one object refers to another, are its name and its owner
- In short, the *naming mechanisms*, historically, were not rich enough to support online application upgrade

# The solution: editions

- EBR introduces the new nonschema object type, *edition* – each edition can have its own private occurrence of “the same” object
- A database must have at least one edition
- You create a new edition as the child of an existing edition – and an edition can’t have more than one child
- A database session specifies which edition to use  
(of course, the database has a default edition)

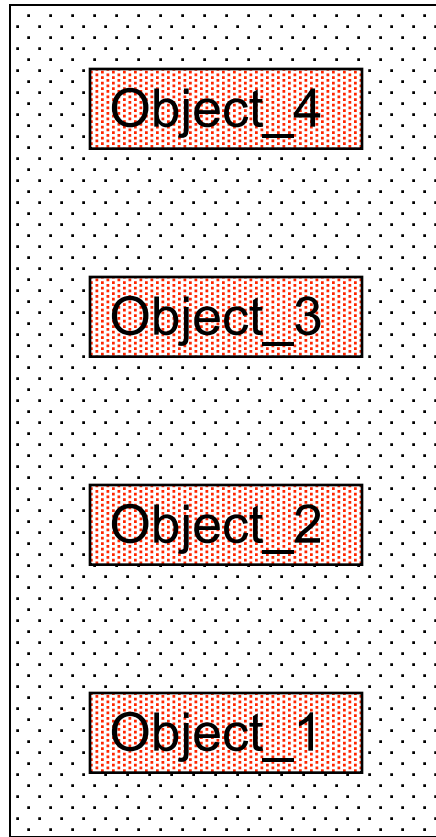
# The solution: editions

- Through 11.1, an object is identified by its name and its owner
- From 11.2, an editioned object is identified by its name, its owner, and the edition where it was created
- However, when you identify it you can mention *only* its name and owner. This reference is interpreted in the context of a current edition
  - in SQL execution
  - in the text of a stored object

# Editions: semantic model

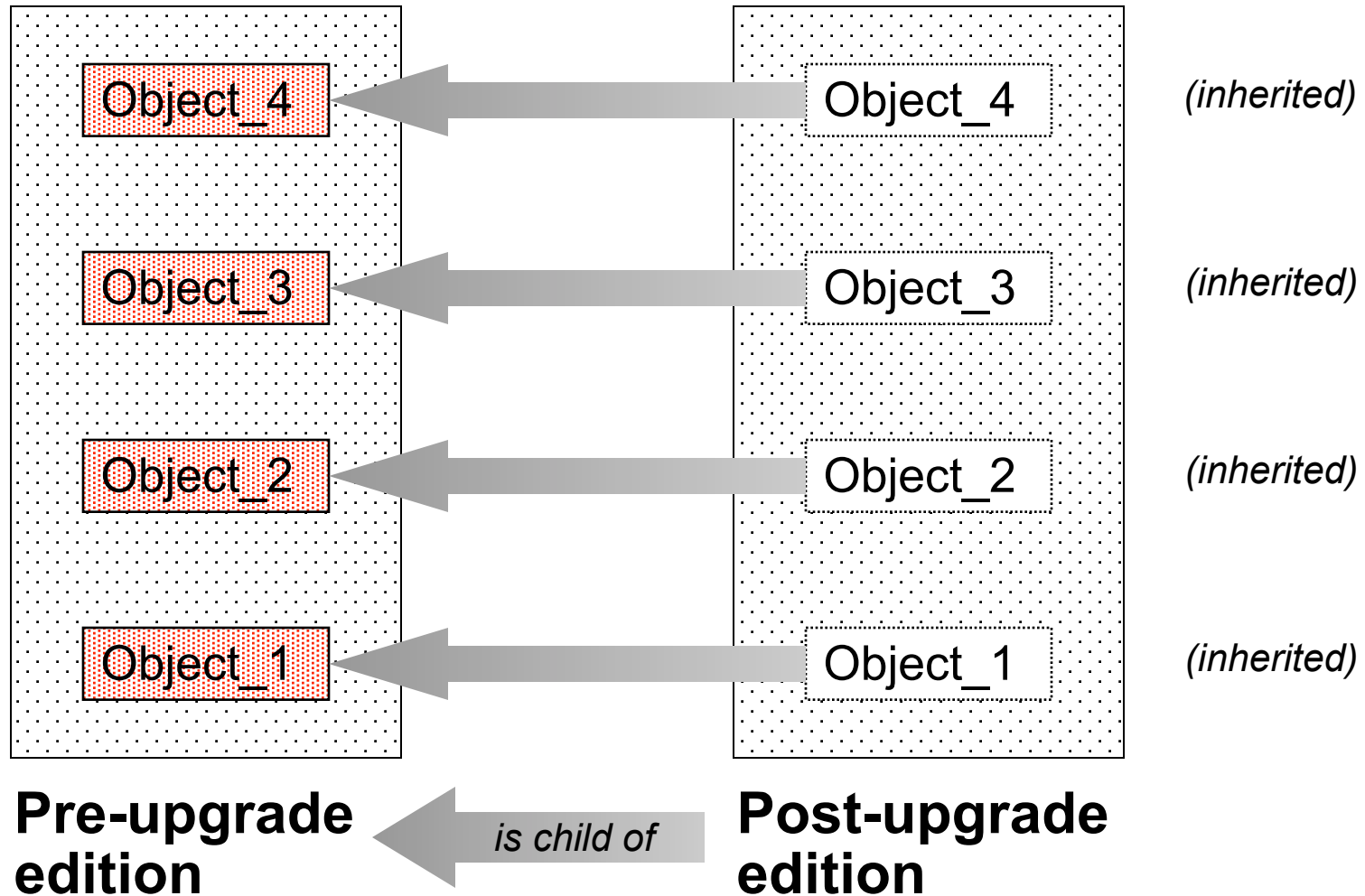
- When you create a new edition, every editioned object in the parent edition is copied into the new edition

# Editions: implementation model

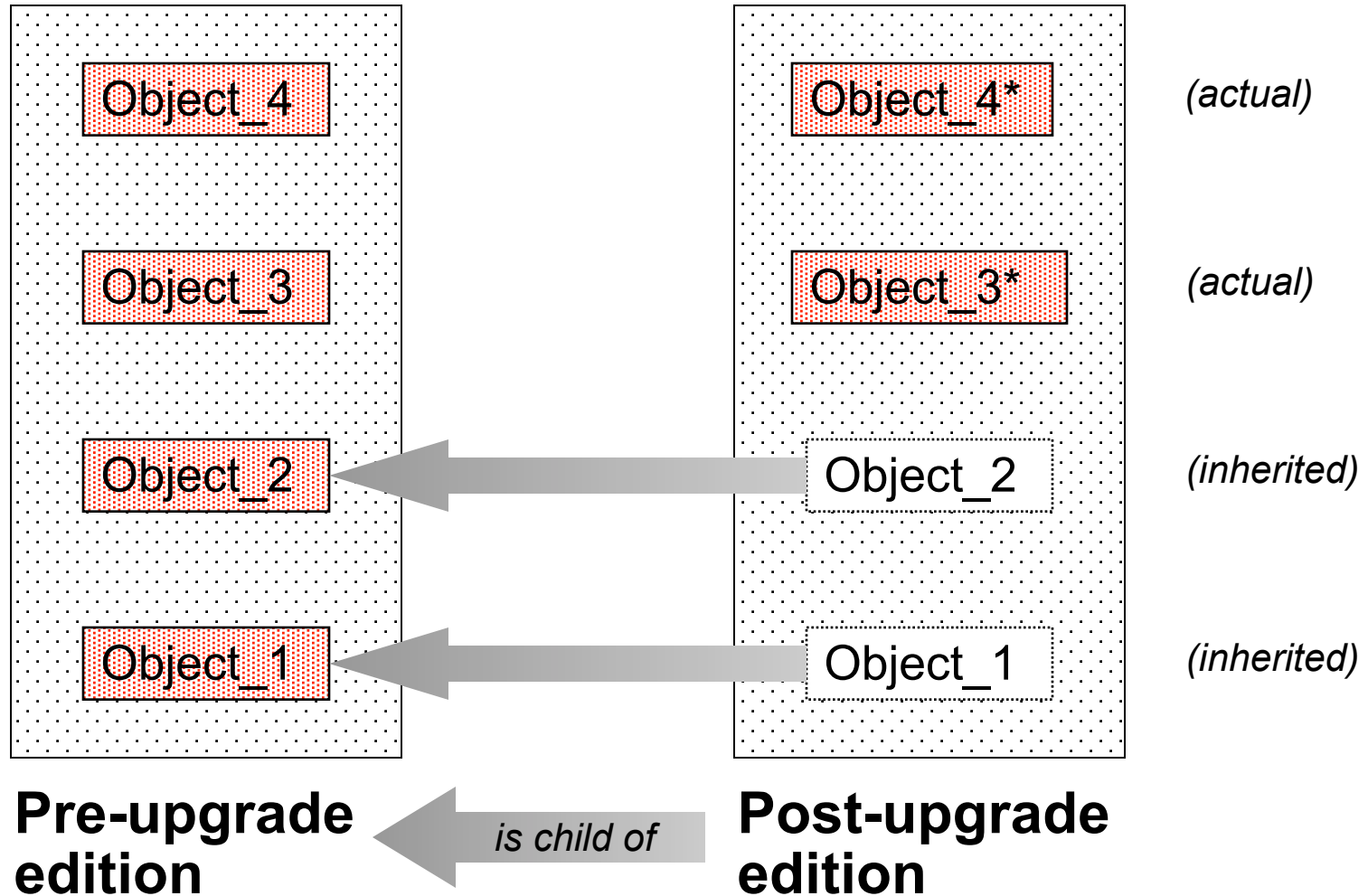


**Pre-upgrade  
edition**

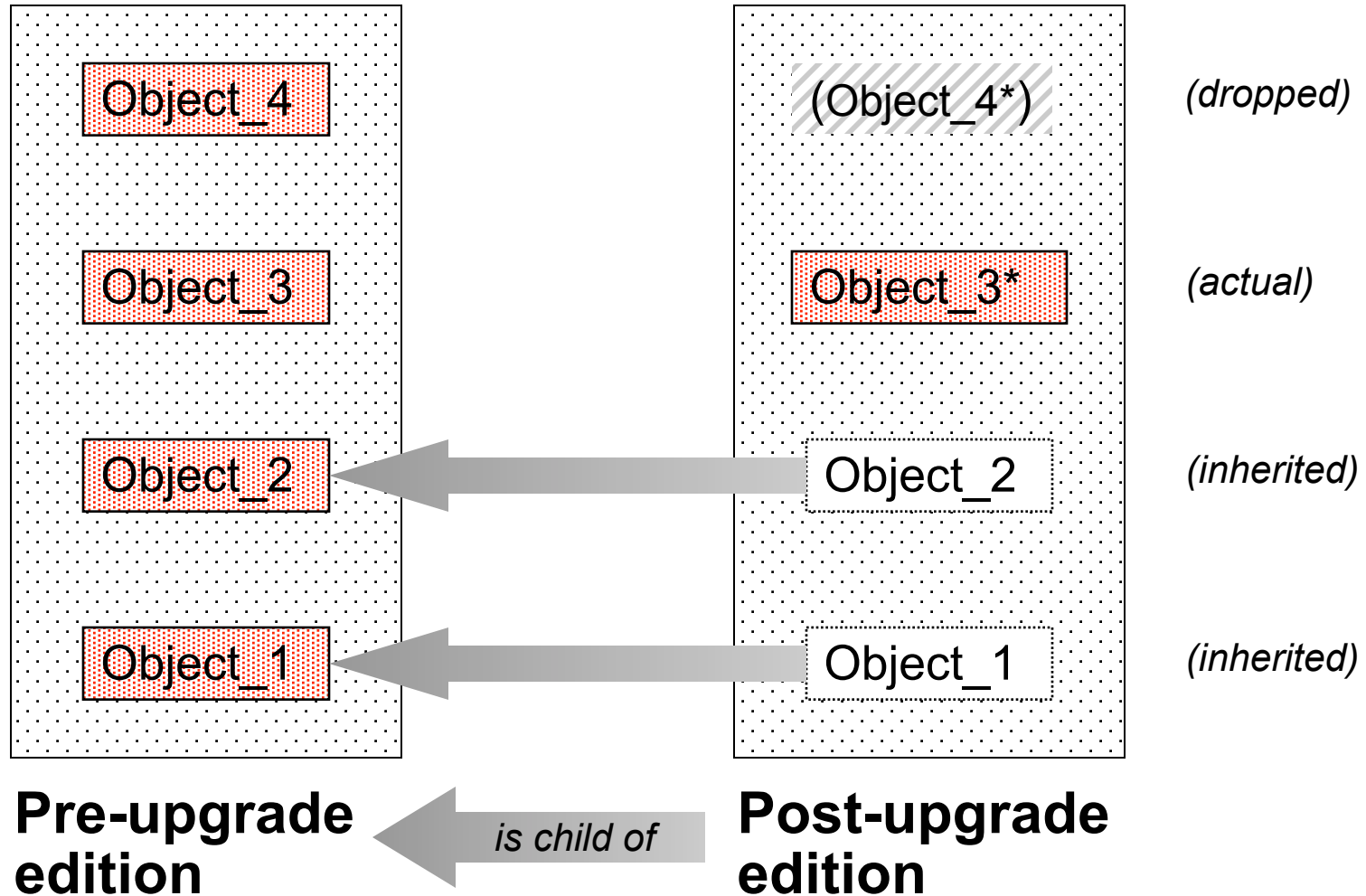
# Editions: implementation model



# Editions: implementation model



# Editions: implementation model





# Editions

- If your upgrade needs only to change synonyms, views, or PL/SQL units, you now have all the tools you need
- Simply run the scripts that you, anyway, have written using a new edition while the application stays online
- Then change the default edition and let new session start in the new edition
- No “package state discarded” errors ever again!

# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- Explanation of the *edition*
- **Explanation of the editioning view**
- Explanation of the *crossedition trigger*
- Readyng an application for EBR
- Case study explained
- Conclusion / Q&A

# 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), and are editable
  - Objects of all other object types – for example tables – are noneditable
- You 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

# The solution: editioning views

- An *editioning view* may only project and rename columns

# Editioning views

- You can't have more than one editioning view for a particular table in a particular edition
- The EV must be owned by the table's owner
- Application code should refer only to the logical world
- You can create table-style triggers (*before* or *after statement* or *each row*) on an editioning view using the "logical" column names
- A SQL optimizer hint can request an index on the physical table by specifying the "logical" column names

# Editing views

- Any SQL statement that refers to one, or several, EVs will get the same execution plan as the statement you'd get if you replaced each of those references, by hand, with a reference to the table that the EV covers
- So using an EV in front of every table brings no performance consequences
- Tests have proved this

# Editing views

- If you can tolerate only read access to the underlying data for an editing view that the upgrade will change<sup>\*</sup>, you now have all the tools you need
- Like all views, an editing view can be read-only
- Ordinary SQL updates can be used safely to install values in the replacement columns – there's no DML that might be missed because of SQL's read-consistency

\* Think of “configuration data” plus... it's probably acceptable to freeze the catalog of wares for a store's online shopping site

# The next couple of slides reflect new-in-12.1 functionality

- The granularity of the editioned state of an object
  - In 11.2, the granularity is the whole schema
  - From 12.1, the granularity is the individual object
- A materialized view or an index on a virtual column is allowed to depend on an editioned PL/SQL function or an editioned view



# Editioned and noneditioned objects

## – slight return

- An object whose type is noneditionable is never editioned
- An object whose type is editionable is editioned only when you request it for that object (requires that the owner is *editions-enabled*)
- ***Theorem: a noneditioned object cannot ordinarily depend on an editioned object***
  - For example, a table cannot depend on an editioned UDT
  - If you want to use a type as the datatype for a column, that UDT must not be editioned

# Materialized views and indexes on virtual columns

- Objects of these kinds have metadata that is explicitly set by the *create* and *alter* statements
  - the *evaluation edition* explicitly specifies the name of the edition in which the resolution of editioned names, within the closure of the object's dependency parents (at compile time), and those objects that are identified during SQL execution (at run time)
  - The *edition range* explicitly specifies the set of adjacent editions in which the optimizer will consider the object when computing the execution plan

# Tables with UDT columns

- An ordinary (as opposed to virtual) column cannot specify the *evaluation edition* or *edition range* metadata
- Therefore, a UDT that defines the datatype for a table column must remain noneditioned.
- In an EBR exercise, if the aim is to redefine the UDT, then the “classic” replacement column paradigm is used
  - A spec doesn't depend on its body. So the body of an ADT, where the code is, can be editioned. (The appropriate one is found at run time.)

# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- Explanation of the *edition*
- Explanation of the *editioning view*
- **Explanation of the *crossedition trigger***
- Readyng an application for EBR
- Case study explained
- Conclusion / Q&A

# What if DML cannot stop during upgrade?

- If the upgrade needs to change the structure that stores transactional data – like the orders customers make using an online shopping site – then the installation of values into the replacement columns must keep pace with these changes
- 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

(even though, for a forward crossedition trigger, the session is using the *Pre\_Upgrade* edition)

# Why such a long name?

- DDL stands for *data definition language*
- “*create or replace*” and “*alter*” re-define an existing object
- These bare commands are *in-place redefinition*
- Online table redefinition (there’s that word again) creates a secret copy, keeps it in step, and then does the twizzle. Similar for online index rebuild
- This is *copy-based redefinition*
- Edition-based redefinition lets you redefine *many* objects in concert



# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- Explanation of the *edition*
- Explanation of the *editioning view*
- Explanation of the *crossedition trigger*
- **Readying an application for EBR**
- Case study explained
- Conclusion / Q&A

# The design before EBR

- Application code accesses tables directly, in the ordinary way

# Readying the application for editions

- Put an editioning view in front of every table
  - The EV and the table it covers can't have the same name
  - 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
- NOTE:
  - If a schema has an object, whose type is noneditionable, that depends on an object whose type *is* editionable, then the adoption plan must accommodate this (using new-in-12.1 functionality) by controlling the granularity of the editioned state of objects, whose type *is* editionable, at the granularity of the individual object
  - Else, the editioned state can be conveniently set for the whole schema

# Readying the application for editions

- Revoke privileges from the tables and grant them to the editioning views
- Move VPD policies to the editioning views
- “Move” triggers to the editioning views
  - Just drop the trigger and re-run the original (or mechanically edited) create trigger statement to recreate it on the editioning view

# 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\_*
- **NOTE:** this readying work must be done by the developers of an application that adopts EBR

# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- Explanation of the *edition*
- Explanation of the *editioning view*
- Explanation of the *crossedition trigger*
- Readyng an application for EBR
- **Case study explained**
- Conclusion / Q&A

# Case study

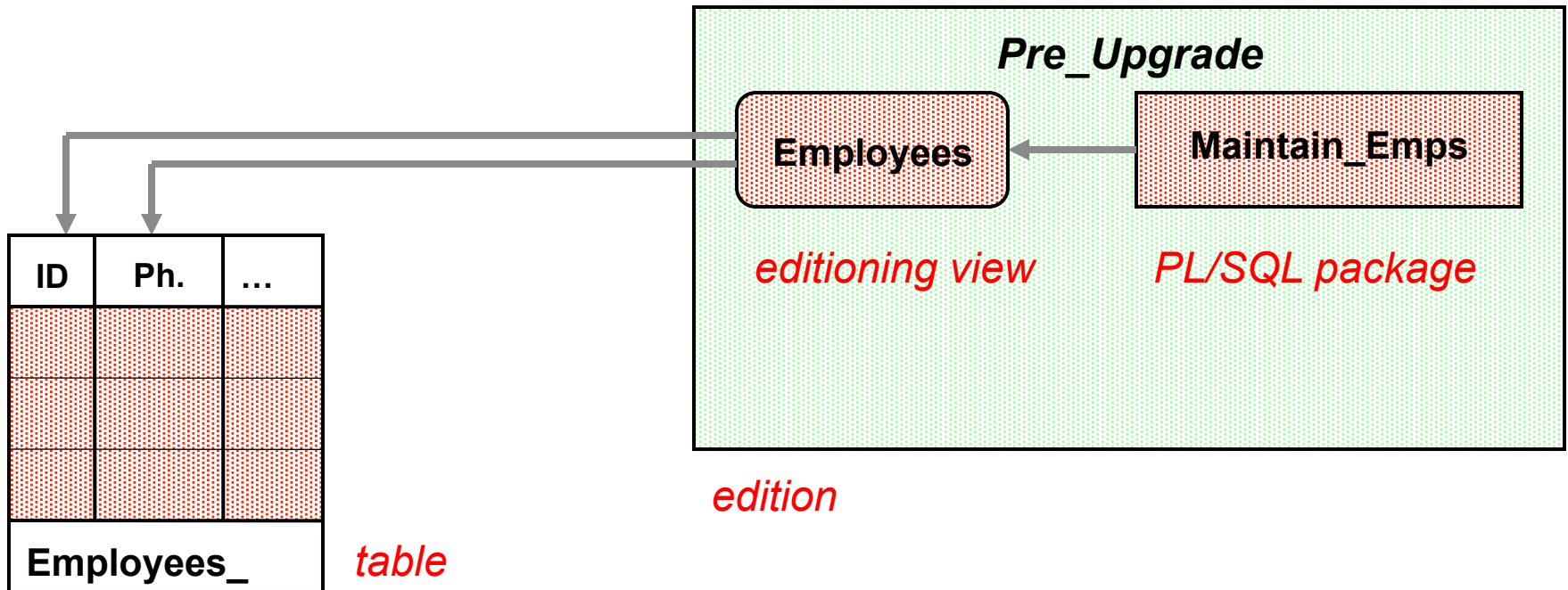
ct 1 Show\_20\_Rows\_Repeat\_Pre\_Upgrade

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
Bruce	Ernst	431.800.6569
David	Austin	
Valli	Pataba	
Diana	Lorent	
Nancy	Greenb	
Daniel	Faviet	
John	Chen	
Ismael	Sciarr	
Jose Manuel	Urman	
Luis	Popp	
Den	Raphae	
Alexander	Khoo	
Shelli	Baida	
Sigal	Tobias	
Guy	Himuro	
Karen	Colmen	

ct 2 Show\_20\_Rows\_Repeat\_Post\_Upgrade

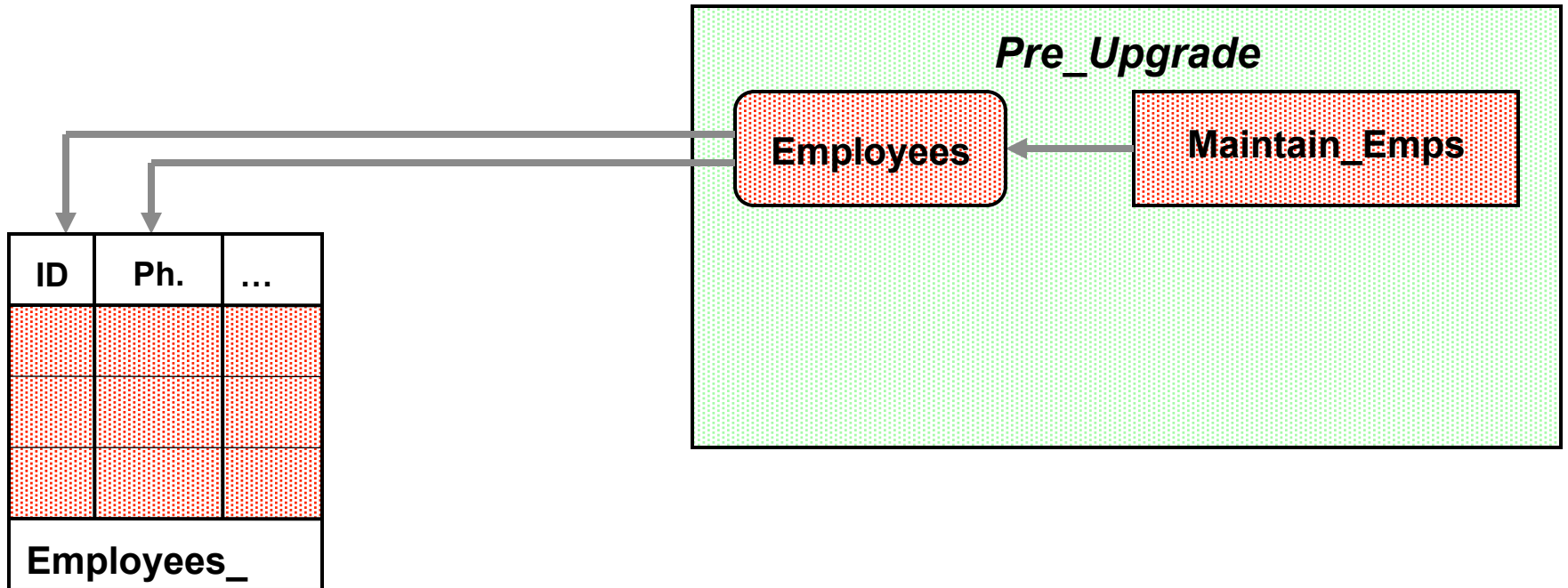
First Name	Last Name	Cntry #
Steven	King	+32 242-647-4719
Neena	Kochhar	+1 708-108-8233
Lex	De Haan	+1 205-621-9819
Alexander	Hunold	+38 209-317-1291
Bruce	Ernst	+1 431-800-6569
David	Austin	+1 207-718-4492
Valli	Pataballa	+45 662-851-6340
Diana	Lorentz	+47 951-260-7204
Nancy	Greenberg	+1 434-531-4024
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

Starting point.  
Pre-upgrade app in normal use.





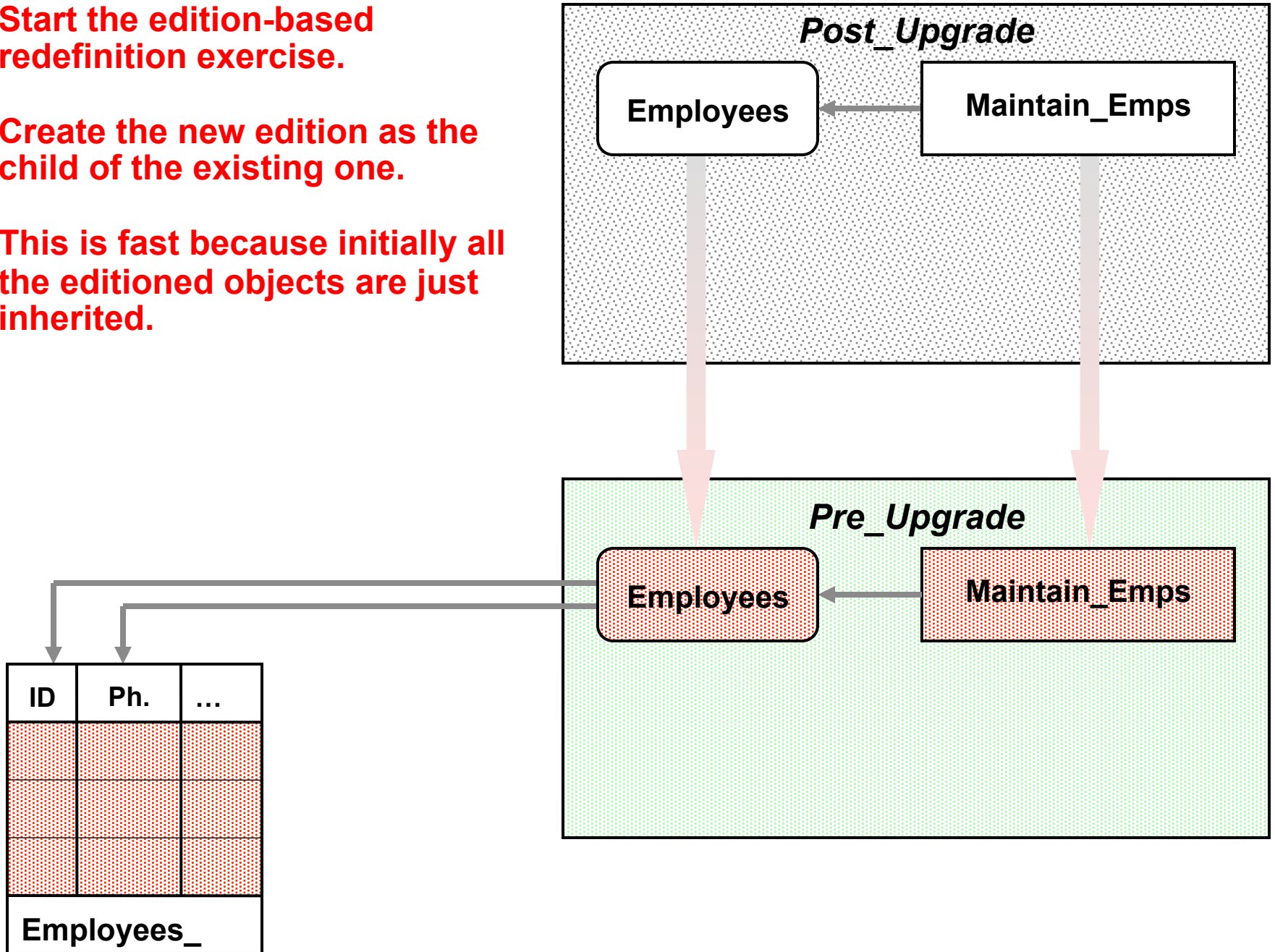
**Starting point.**  
**Pre-upgrade app in normal use.**



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.

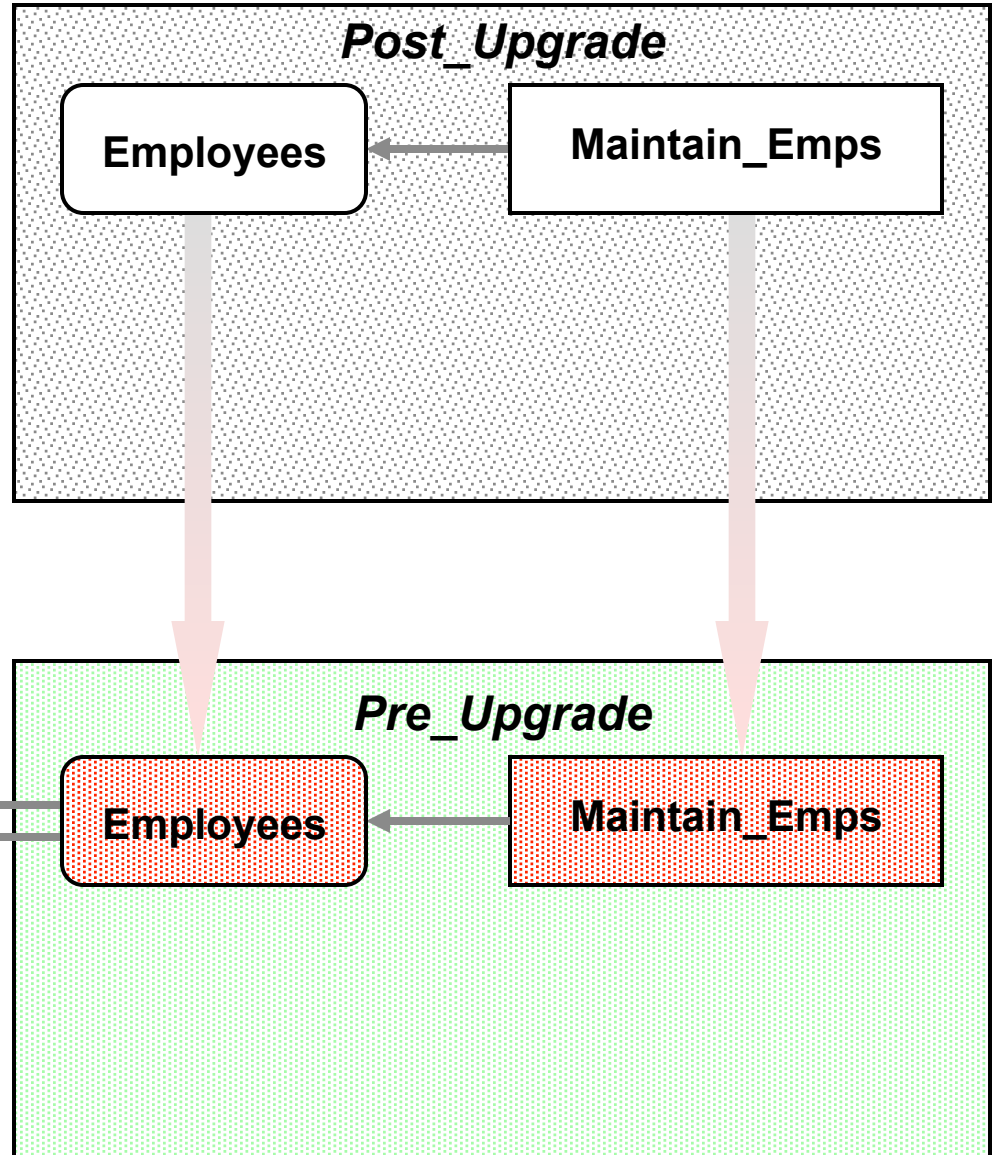


Create the replacement columns in the underlying table.

The editioning view shields the app from this change.

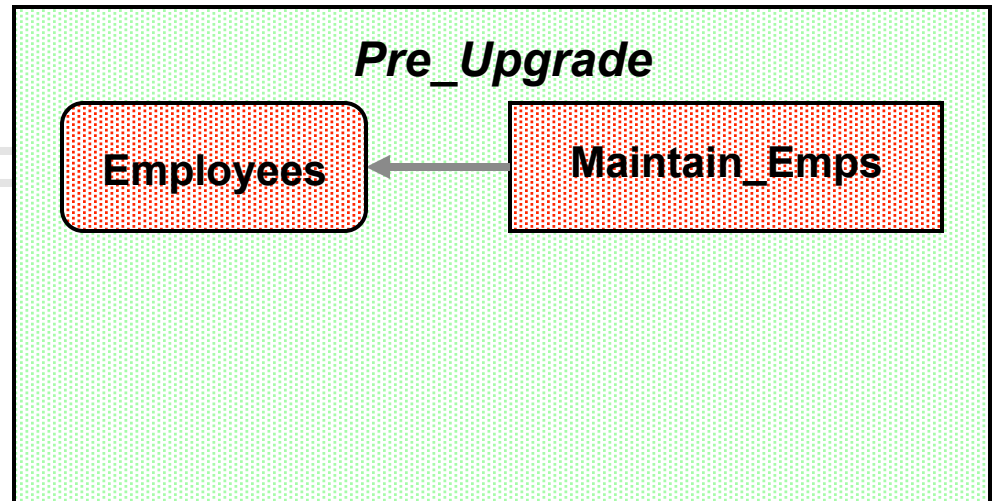
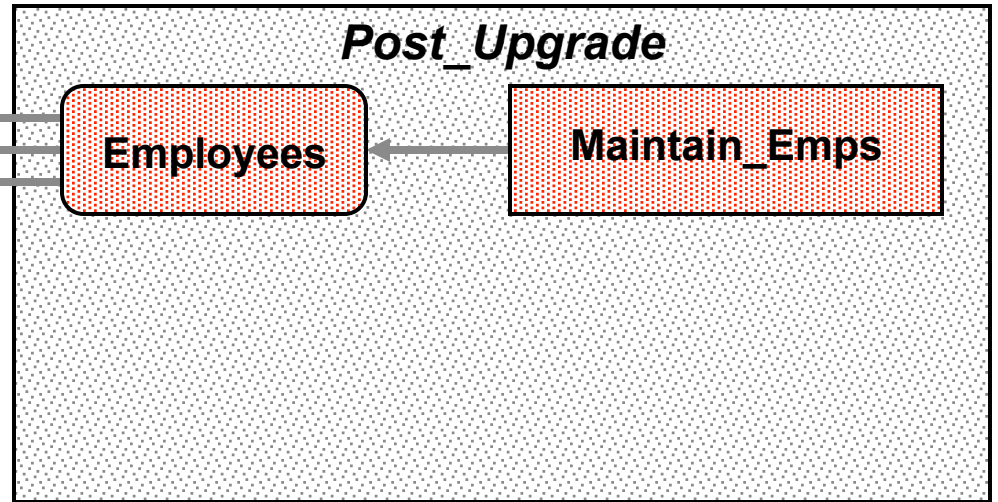
ID	Ph.	...	Cntry	#

Employees\_



Change *Employees* to select the new columns.

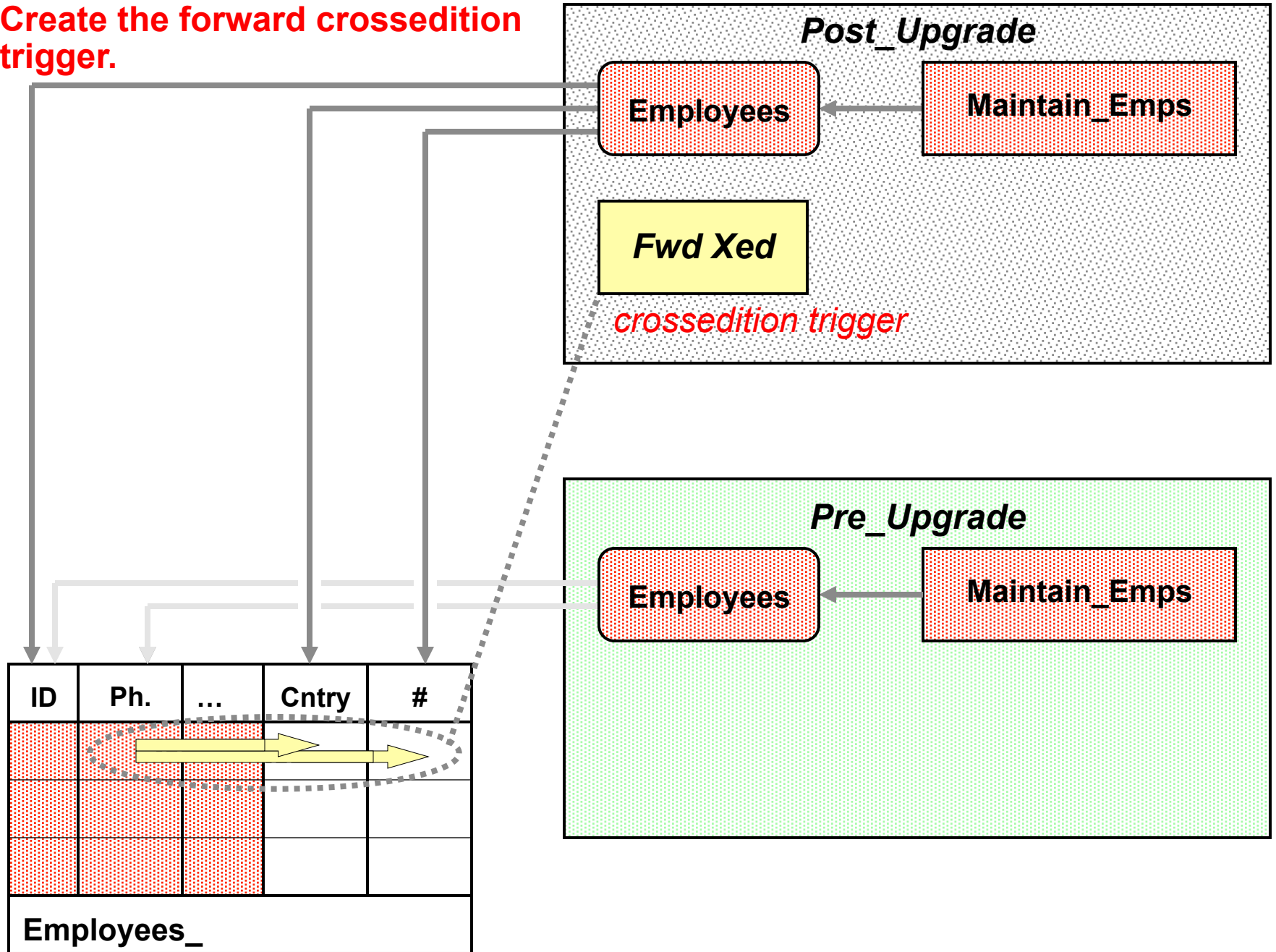
Change *Show\_Employees* to implement the new behavior.



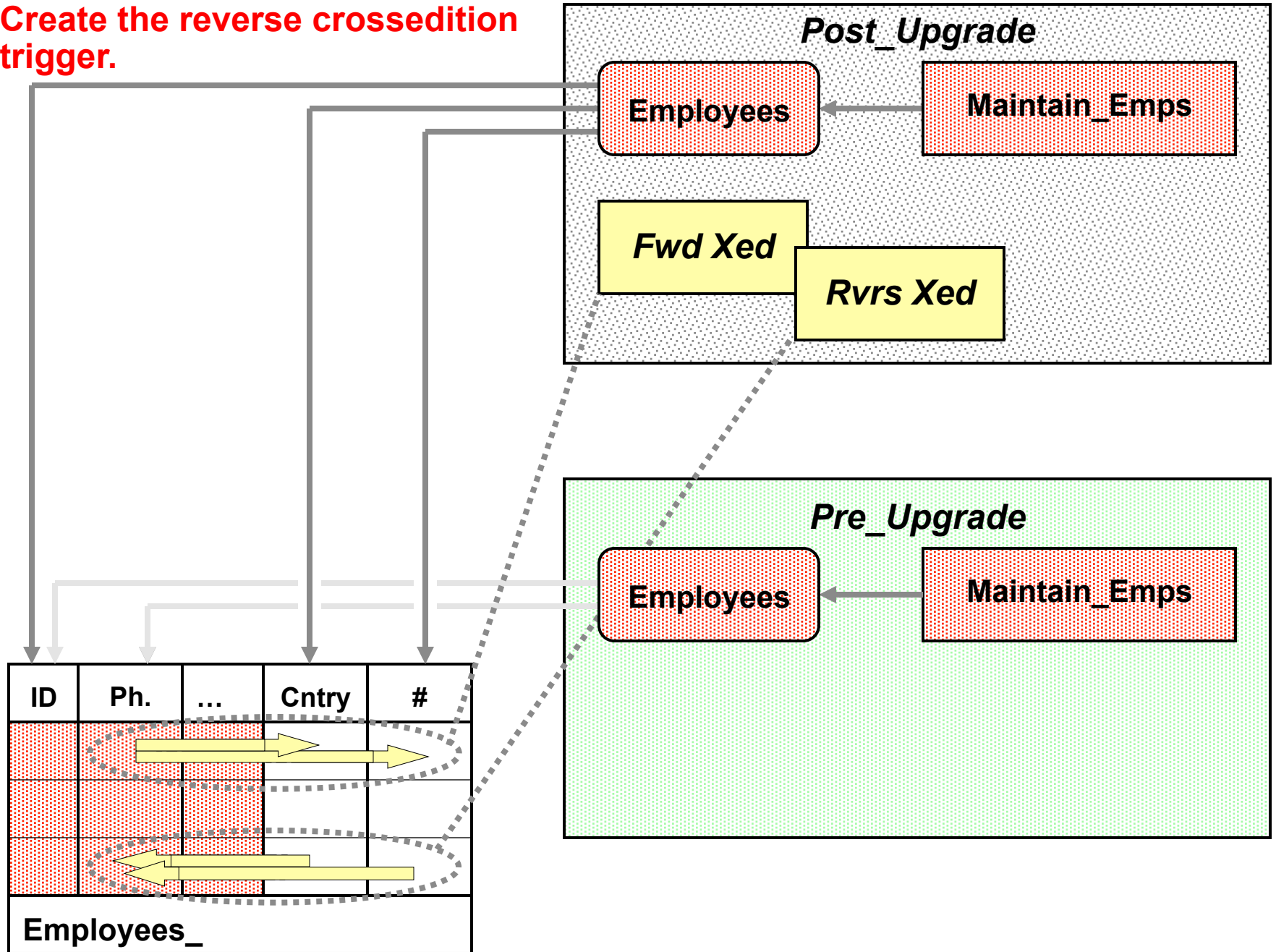
ID	Ph.	...	Cntry	#

Employees\_

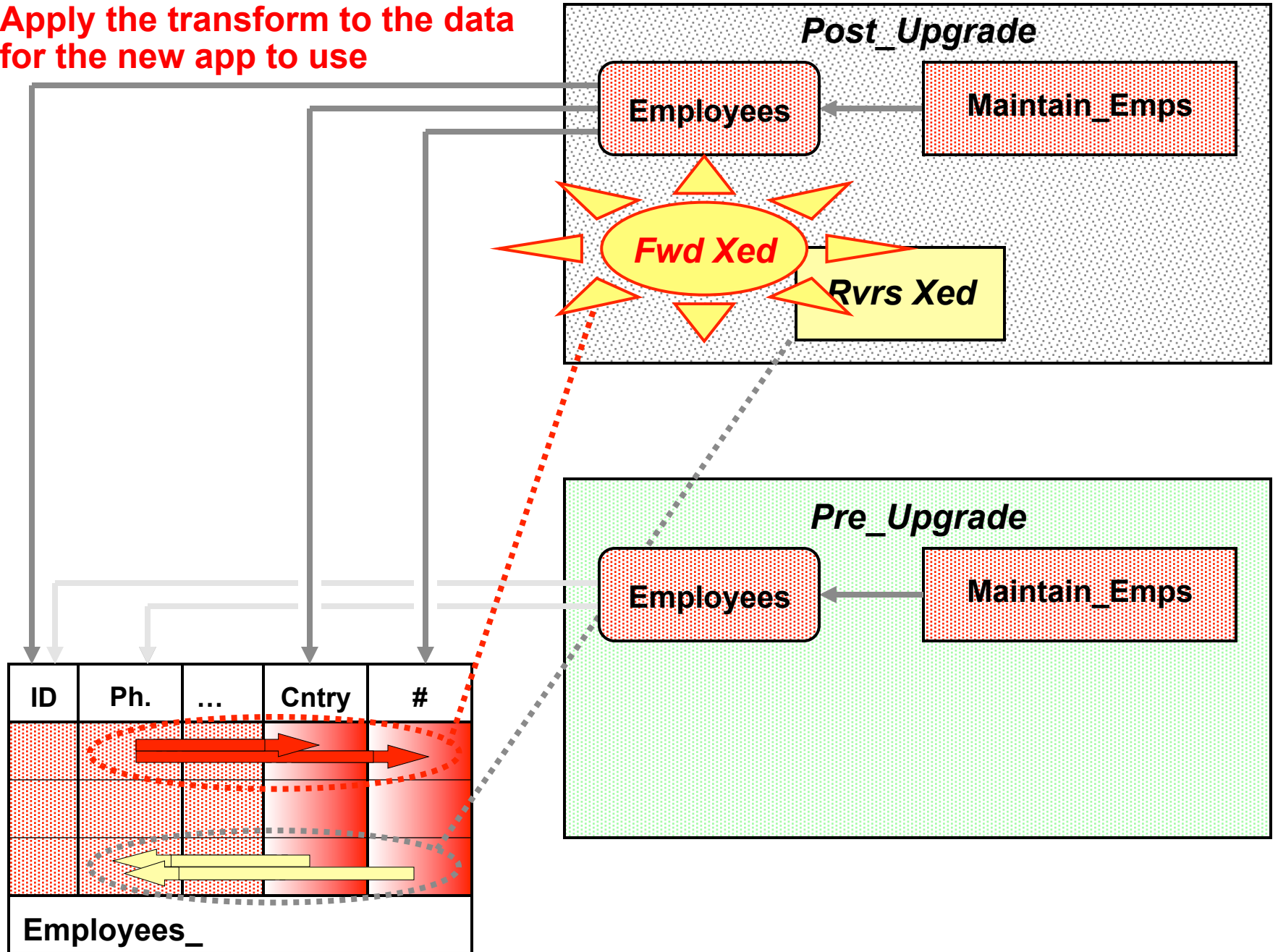
Create the forward crossedition trigger.



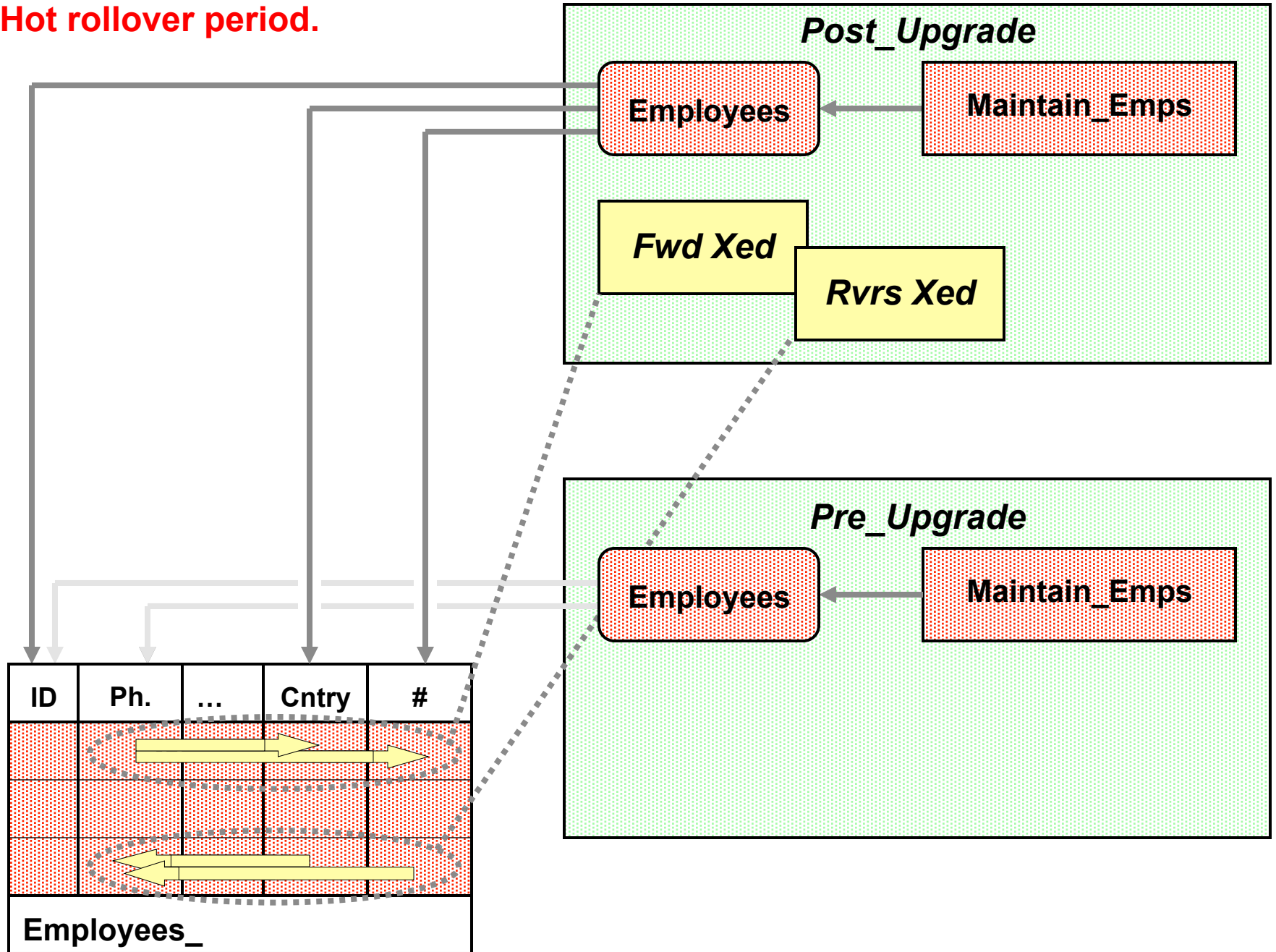
Create the reverse crossedition trigger.



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



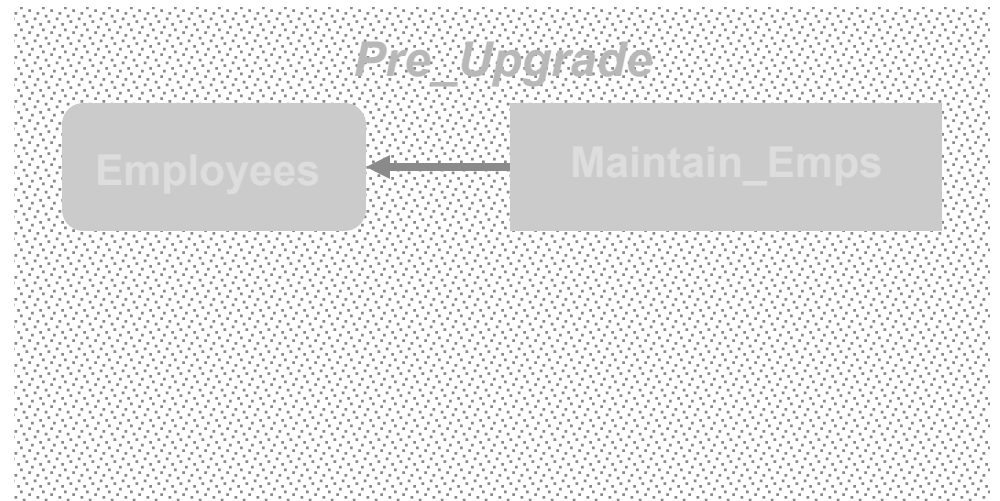
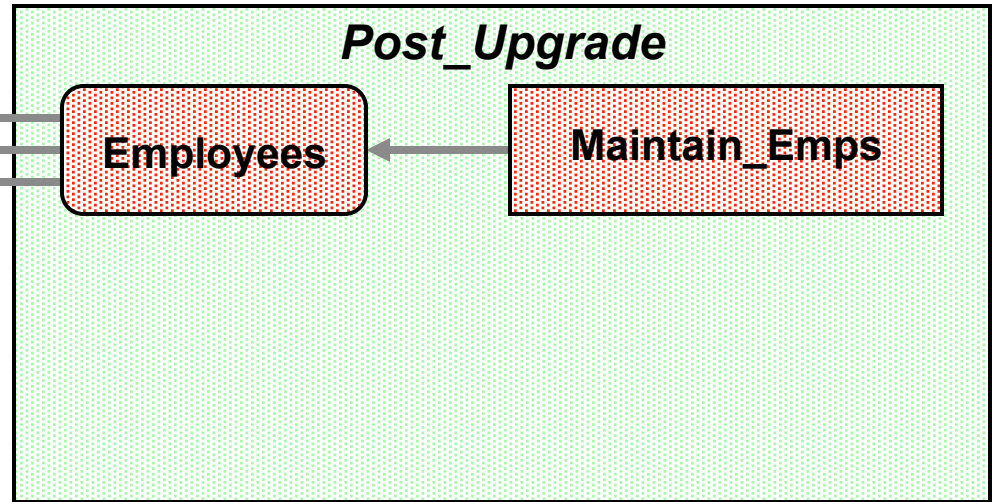
Hot rollover period.





The *Pre\_Upgrade* edition is retired.

The edition-based redefinition exercise is complete.



ID	Ph.	...	Cntry	#

**Employees\_**

**Case study – continued**

**Rolling back the upgrade**



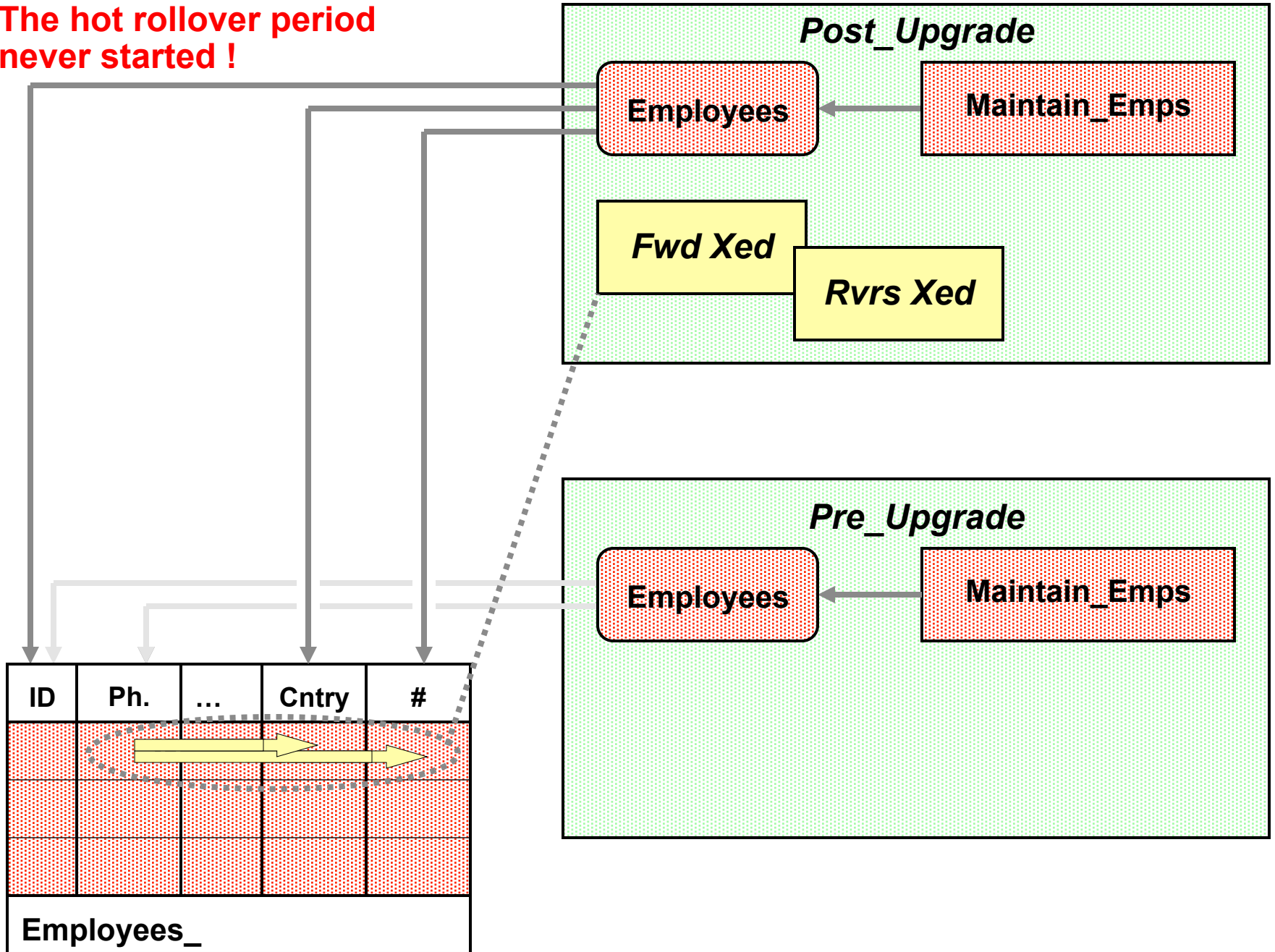
# Rolling back an online app upgrade

- Rolling back an application upgrade that's been installed classically is easy until you go live with the post-upgrade application
  - Presumably you took a backup at the start of the offline period and you just restore to that
- But once you go live with the post-upgrade application, you can't rollback to the pre-upgrade one
  - If you did this, you'd lose transactions made during the live use of the post-upgrade application
- It's just the same with online application upgrade
  - Your grace-period ends when you go live with the post-upgrade application

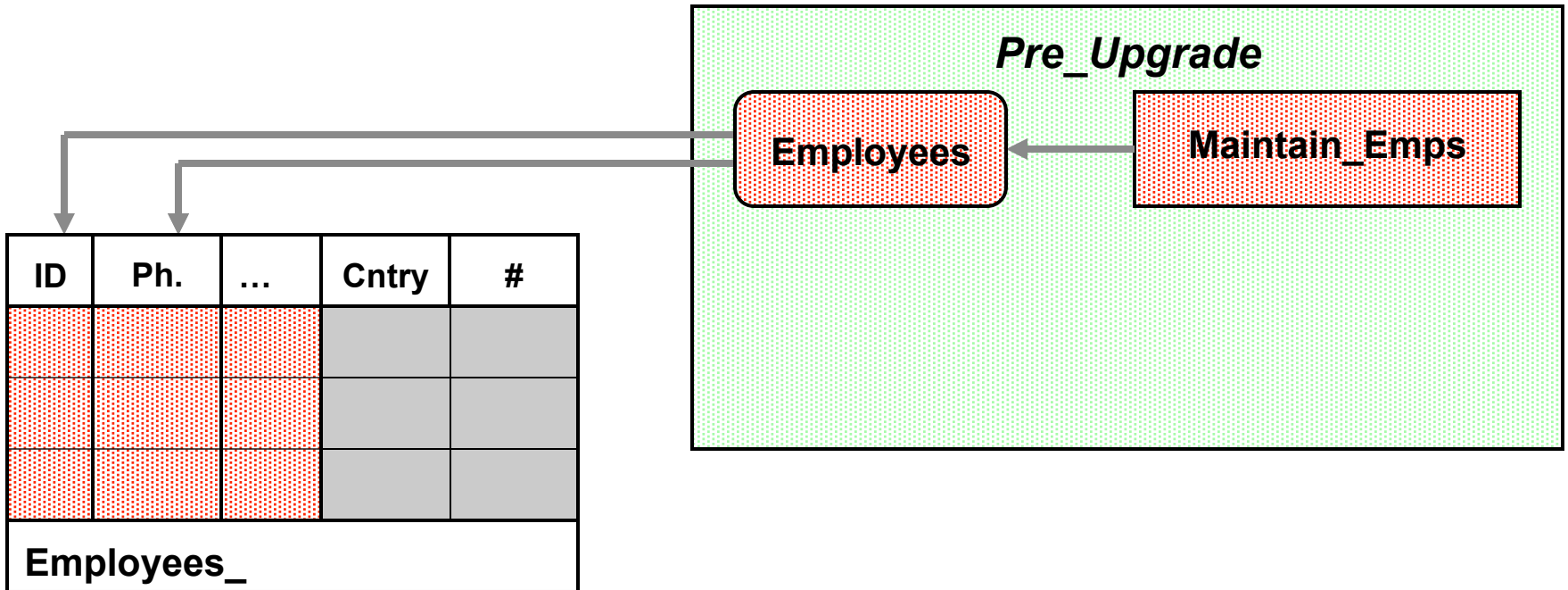
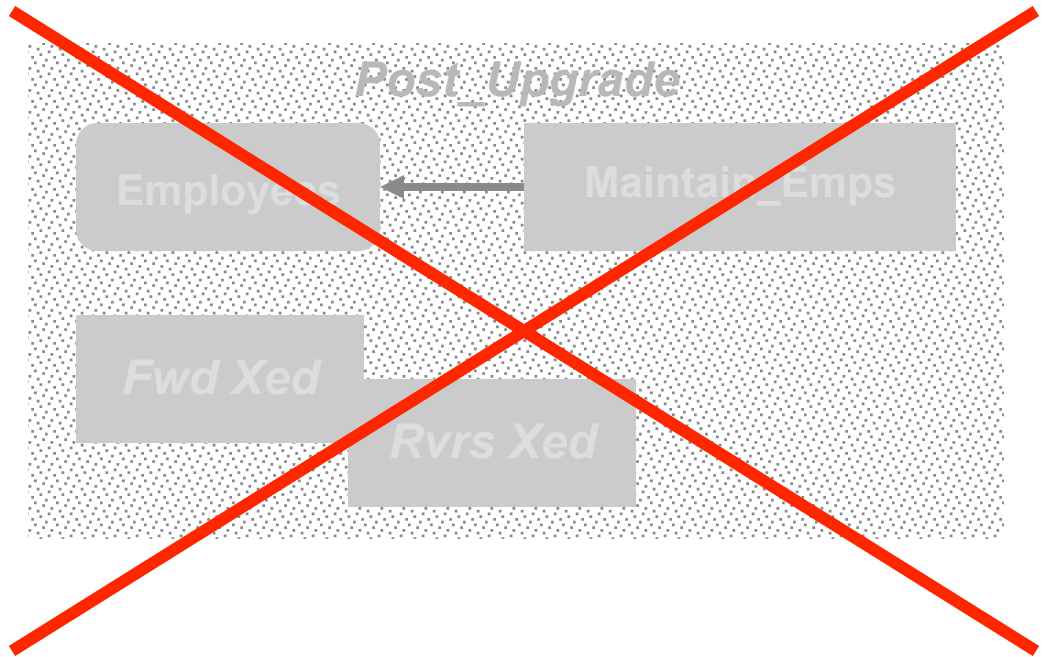
# Rolling back an online app upgrade

- If you haven't gone live with the post-upgrade application
  - Drop the *Post\_Upgrade* edition (cascade)
  - Set any new replacement columns you created *unused*
  - At a convenient later time, recoup the space

The hot rollover period never started !



The pristine Pre\_Upgrade is intact !



# Agenda: optional section

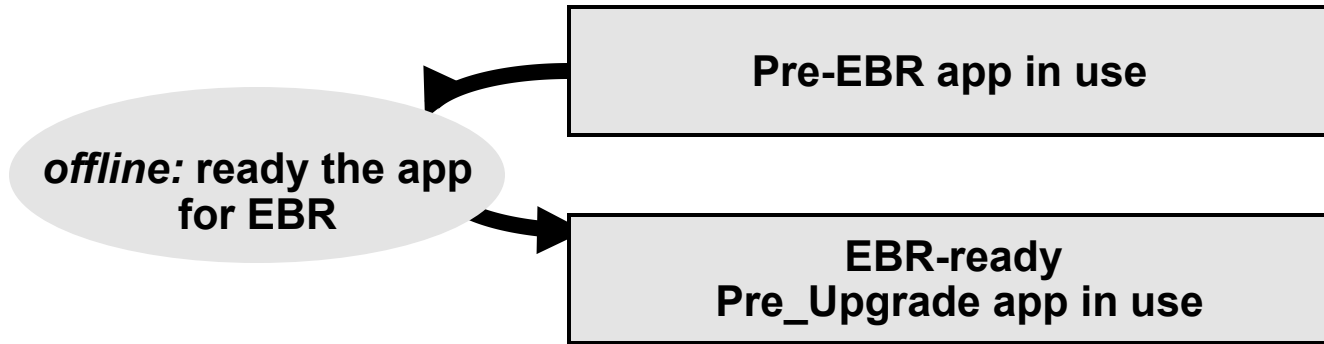
- ...
- Case study stated
- Explanation of the *edition*
- Explanation of the *editioning view*
- Explanation of the *crossedition trigger*
- Readyng an application for EBR
- Case study explained
- **EBR task flow**
- Conclusion / Q&A

# EBR task flow

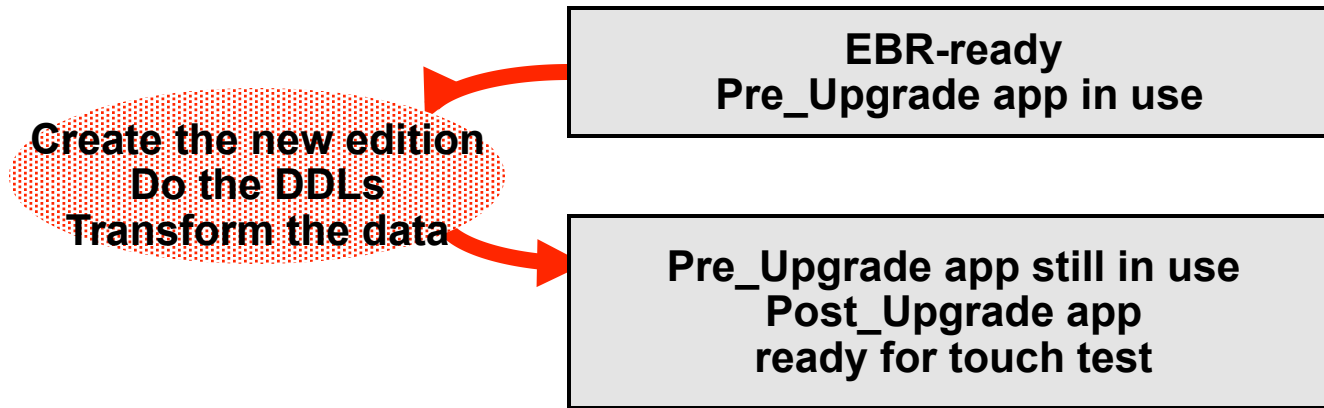
Pre-EBR app in use



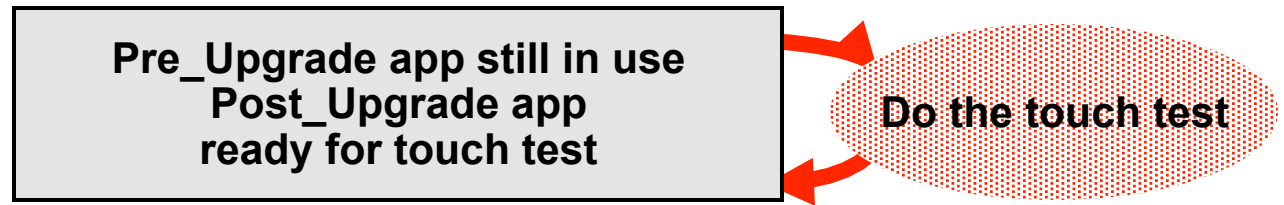
# EBR task flow



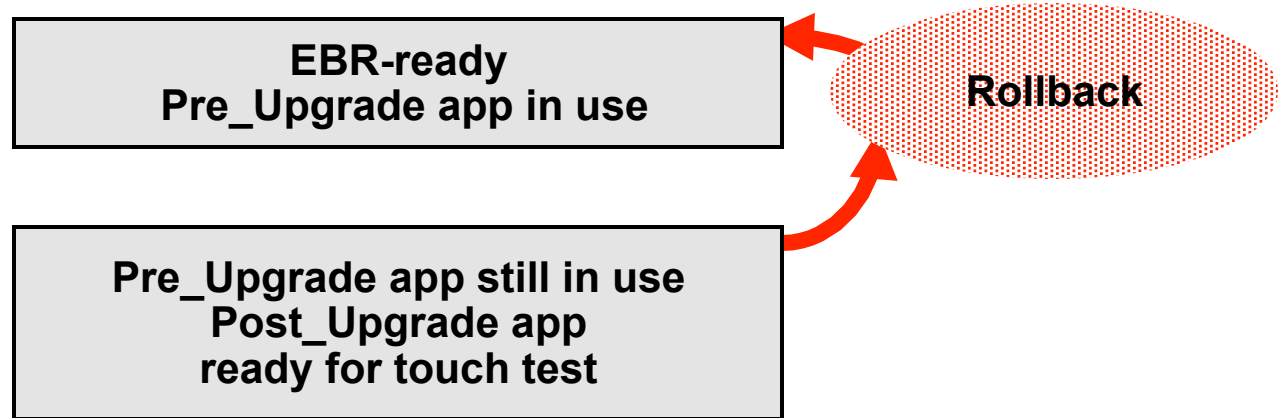
# EBR task flow



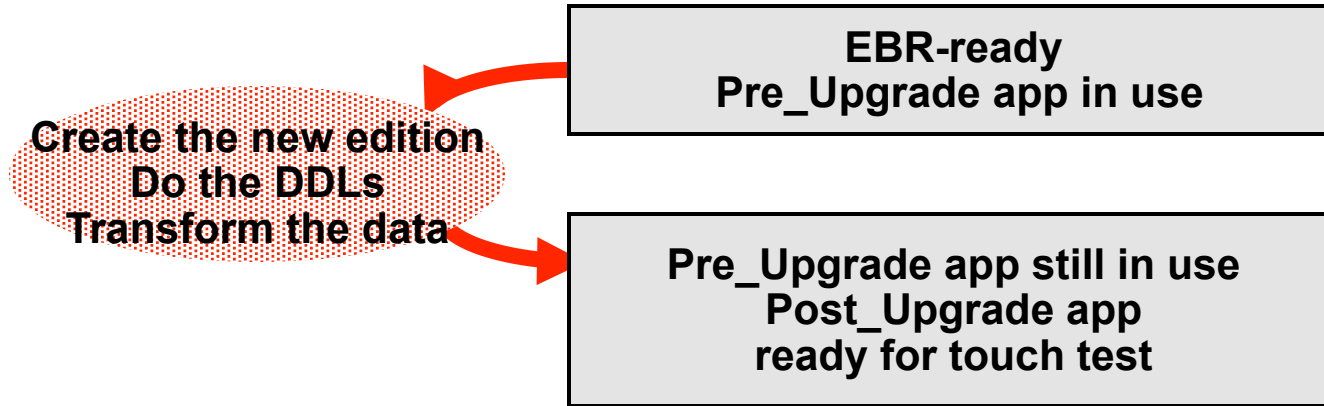
# EBR task flow



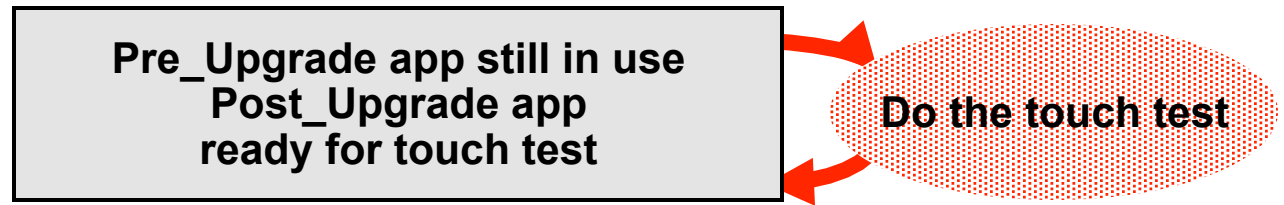
# EBR task flow



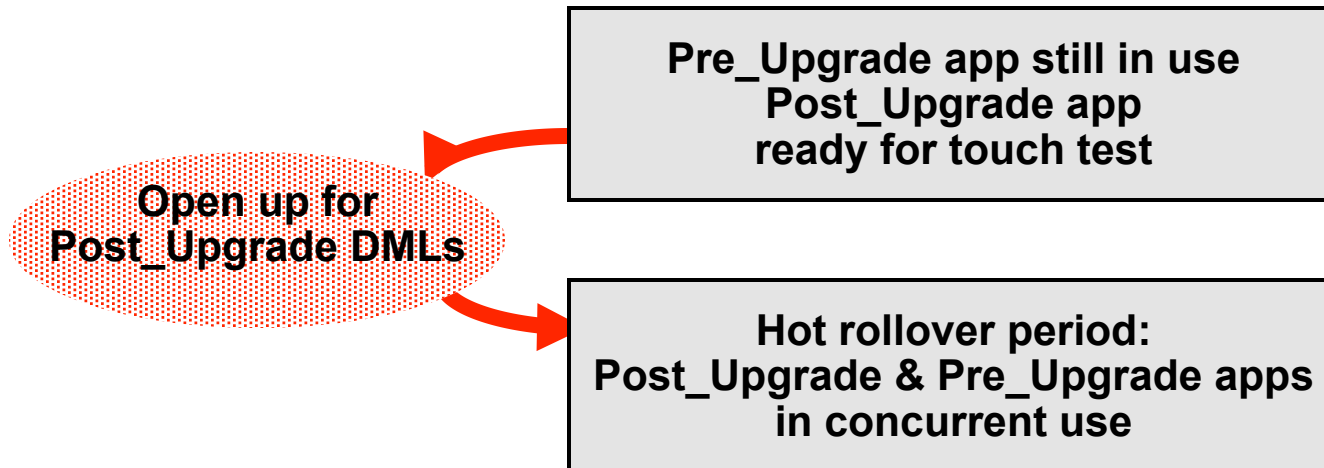
# EBR task flow



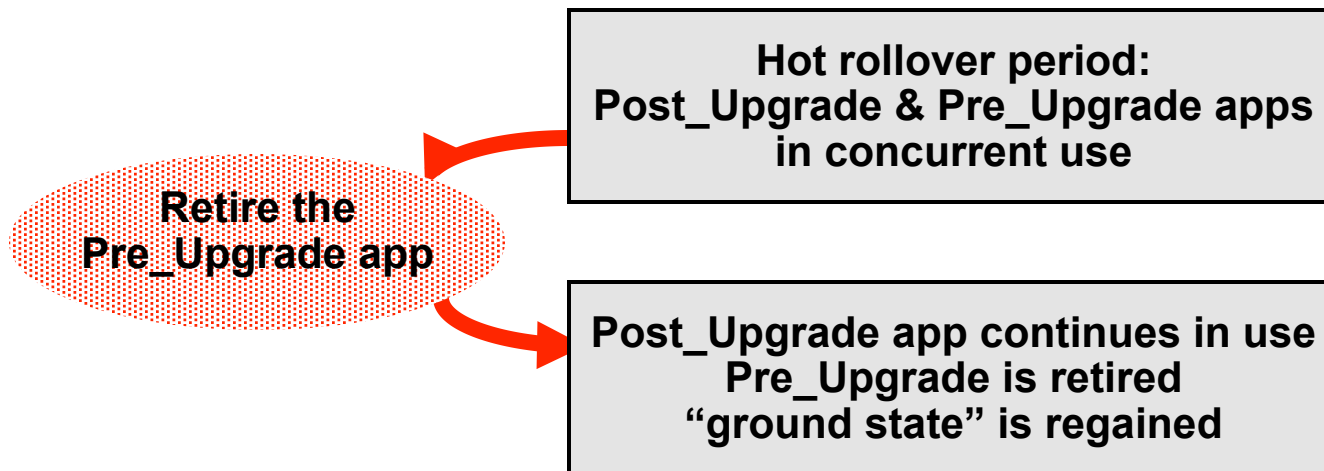
# EBR task flow



# EBR task flow



# EBR task flow

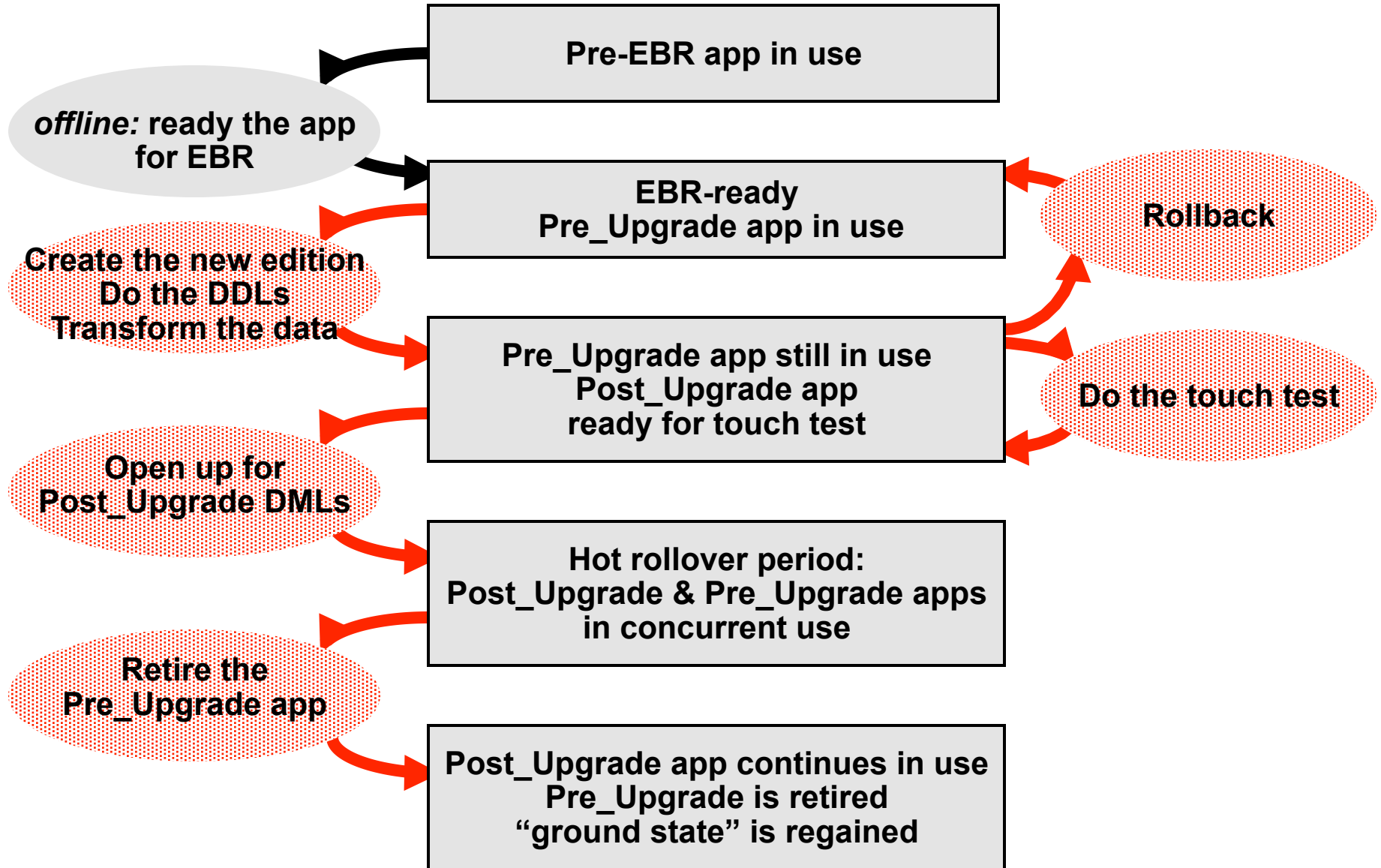




# EBR task flow

**Post\_Upgrade app continues in use  
Pre\_Upgrade is retired  
“ground state” is regained**

# EBR task flow – summary



# Agenda

- Scope of this presentation
- The challenge and the solution stated
- Case study stated
- Explanation of the *edition*
- Explanation of the *editioning view*
- Explanation of the *crossedition trigger*
- Readyng an application for EBR
- Case study explained
- **Conclusion / Q&A**

# Edition-based redefinition

- EBR 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

# Evolutionary capability improvements

- Some table DDLs that used to fail if another session had outstanding DML now always succeed
- Others, that cannot succeed while there's outstanding DML, are now governed by a timeout parameter
- Online index creation and rebuild now never cause other sessions to wait
- The dependency model is now fine-grained:  
*e.g.* adding a new column to a table, or a new subprogram to a package spec, no longer invalidates the dependants

# E-Business Suite 12.2

- The GA of E-Business Suite 12.2 was announced in October 2013
- From now on, all patches are done as EBR exercises
- Critical business operations will not be interrupted
- Revenue generating activities will stay on line
- The short downtime required by patching will be predictable
- The units change:  
**downtime will be measured  
not in days, not in hours  
but in minutes!**

## *Nota bene*

- Online application upgrade is a high availability subgoal
- Traditionally, HA goals are met by features that the administrator can choose to use at the site of the deployed application
  - independently of the design of the application
  - without the knowledge of the application “vendor”
- The features for online application upgrade are used by the application “vendor”
  - when preparing the application for EBR
  - when implementing an EBR exercise
- Site administrators, of course, will need to understand the features

## Next steps...

- Read the edition-based redefinition chapter in the Oracle Database Development Guide (12.1)
- Read my whitepaper:  
published on the High Availability subpage under the Database page on OTN
- Internet search for *edition-based redefinition*



# Q & A

