ORACLE

Edition-Based Redefinition

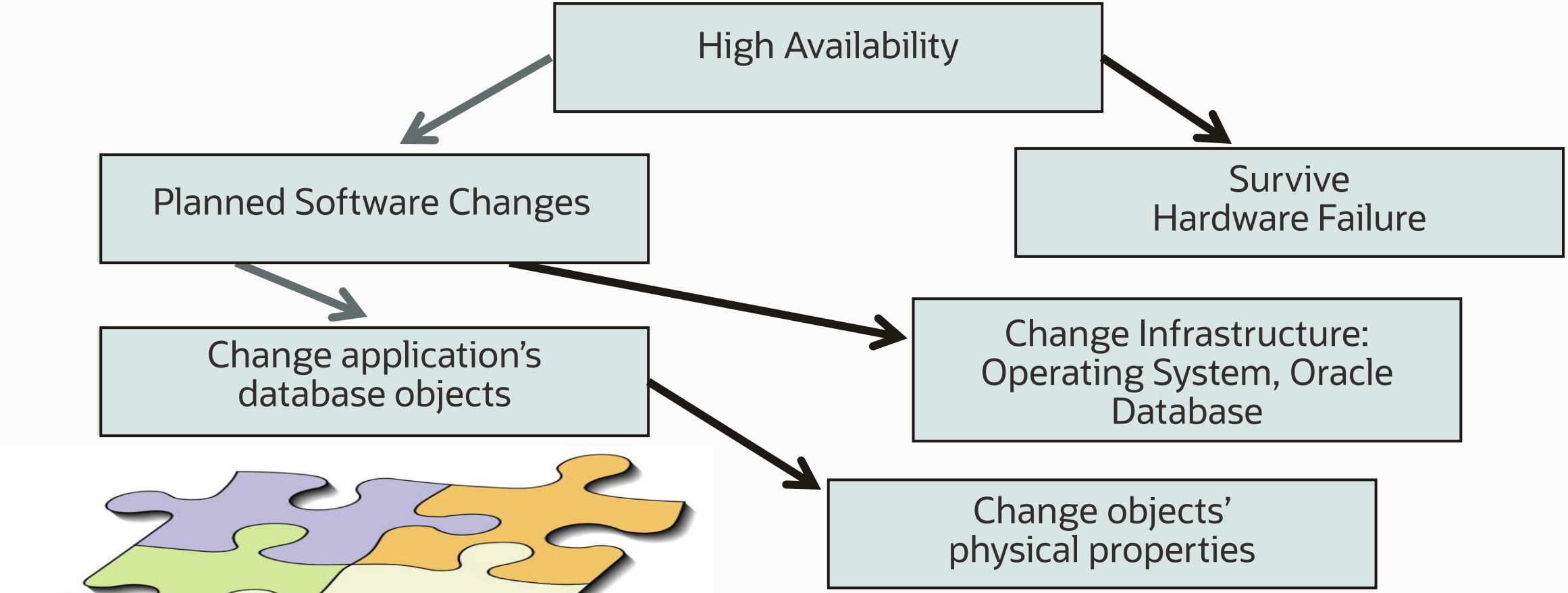
A Key to Online Application Upgrade

Goals of Edition-Based Redefinition

- **Allow** arbitrary changes to the set of artifacts implementing application's database of record
- **Use** both pre-upgrade and post-upgrade application at the same time **(hot rollover)**
- **Maintain** uninterrupted availability of the application
- **Ensure** no noticeable negative impact on performance

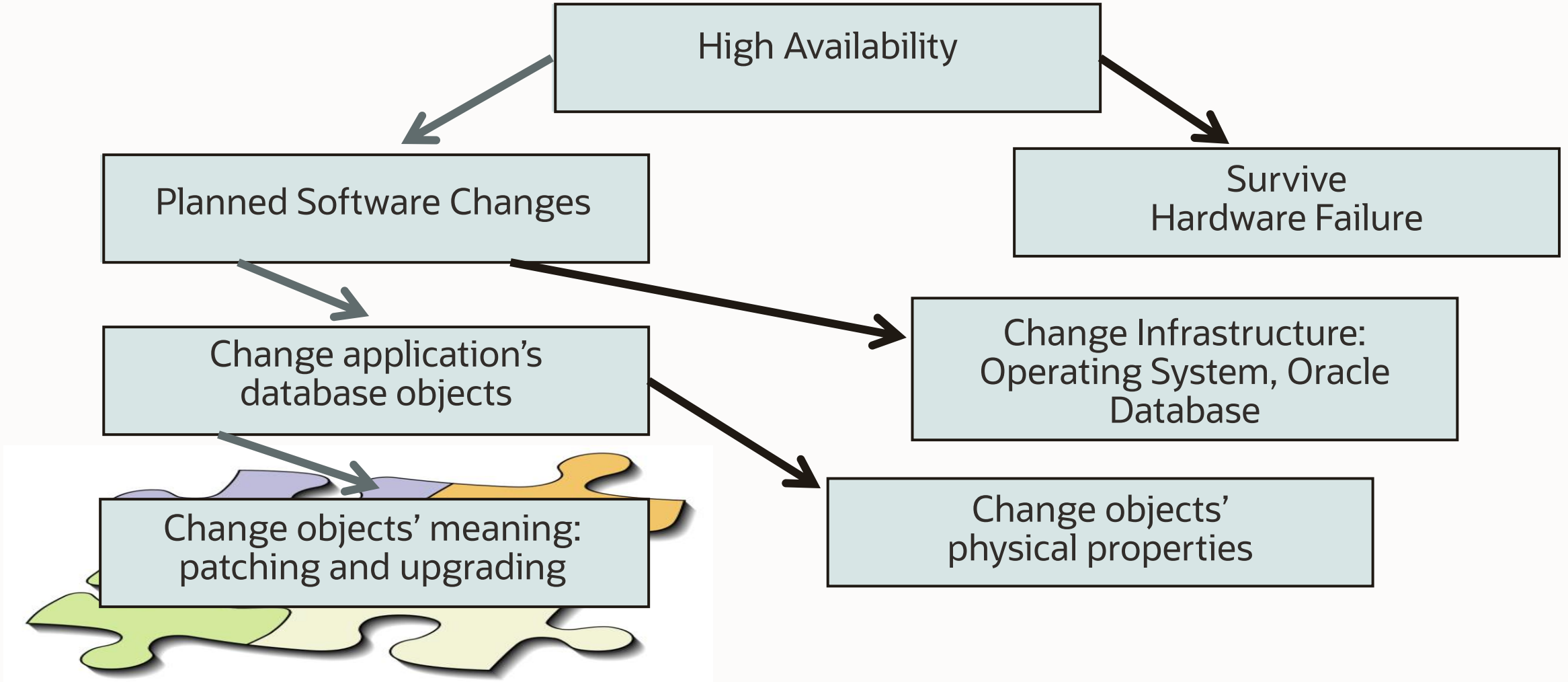
Online Application Upgrade

The final piece of the HA jigsaw puzzle



Online Application Upgrade

The final piece of the HA jigsaw puzzle



Program agenda

- Implementation of case study as an EBR exercise
- The challenge – and the statement of the solution
- Description of case study
- Explanation of edition, editioning view, and cross-edition trigger
- Preparing an application for EBR

Online Application Upgrade

- Supporting online application upgrade requires maintaining uninterrupted availability of the application
- End-user sessions can last tens of minutes or longer because –
  - 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

- Thus, there is requirement for using both the pre-upgrade application and the post-upgrade application at the same time – a.k.a. hot rollover

The Challenge

- Upgrading production database without disturbing live users of the pre-upgrade application
- Making changes to database objects in privacy
- Ensuring transactions done by the users of the pre-upgrade application are reflected in the post-upgrade application
- Achieving hot rollover i.e., transactions done by the users of the post-upgrade application are reflected in the pre-upgrade application.

Functional goals of any ZDT solution

- Cloning V[n] database into V[n+1] database
- Making arbitrary changes in V[n+1] database without them showing up in V[n] database
- Syncing data between V[n] and V[n+1] databases when all intended changes are done in the V[n+1] database.
- Ensuring V[n] database is in sync with V[n+1] database, when V[n+1] is opened for use
- Honoring data and business rules in a transactional fashion—in the presence of this mutual bi-directional synchronization

Non-functional goals of any ZDT solution

- Creating V[n+1] as a clone of V[n] instantaneously without taking additional space.
- It must have no noticeable effect on the performance experience of the online users
- This implies that the solution must work its magic by making one single database seem, to connecting clients, as if it's two separate databases
- This, of course, is how EBR works

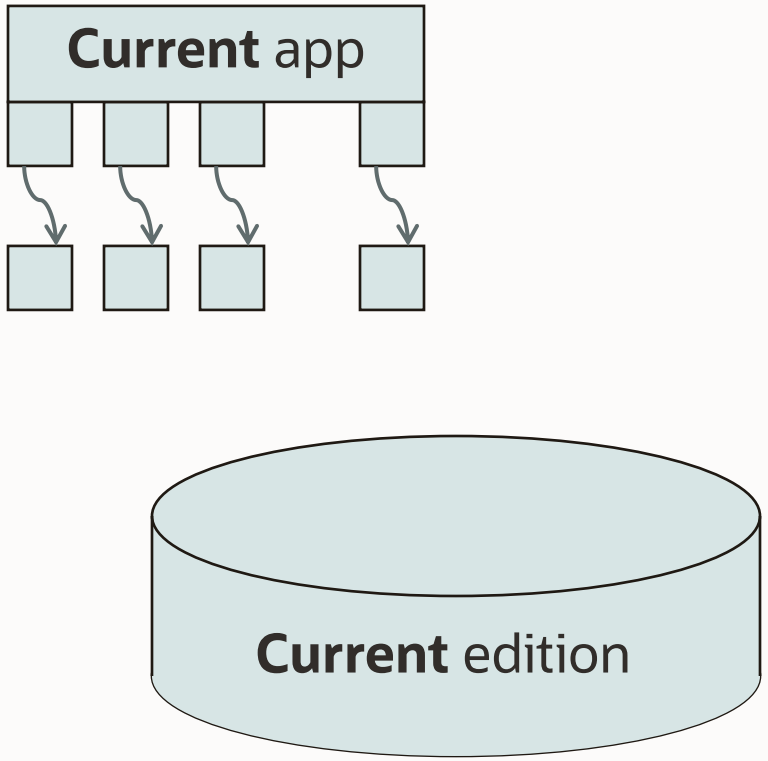
Key Features of Edition-Based Redefinition

- **Edition**
  - Code changes are installed in the privacy of a new edition
  - Data changes are made safely by writing to new columns/tables
  - These changes are not seen by the old edition
- **Editioning View**
  - Exposes a different projection of a table into each edition
  - Allows each user to see just its own columns
- **Cross-edition Trigger**
  - Propagates data changes made by the old edition into the new edition's columns
  - Also, propagates changes made by new edition into old edition's columns in the case of hot- rollover

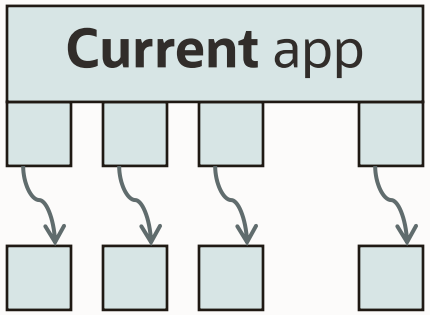
New In 23ai

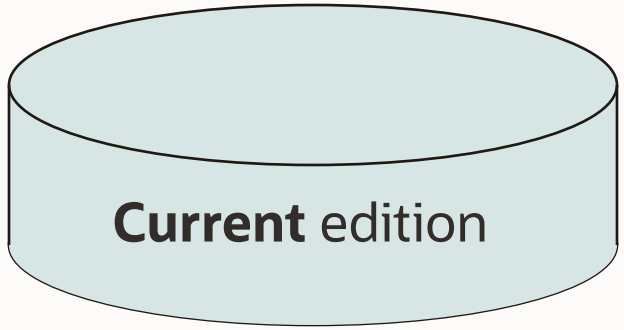
With Oracle Database 23ai, EBR is now compatible with Oracle GoldenGate and other data integration solutions that require supplemental logging.

Hot rollover across stack - General Upgrade

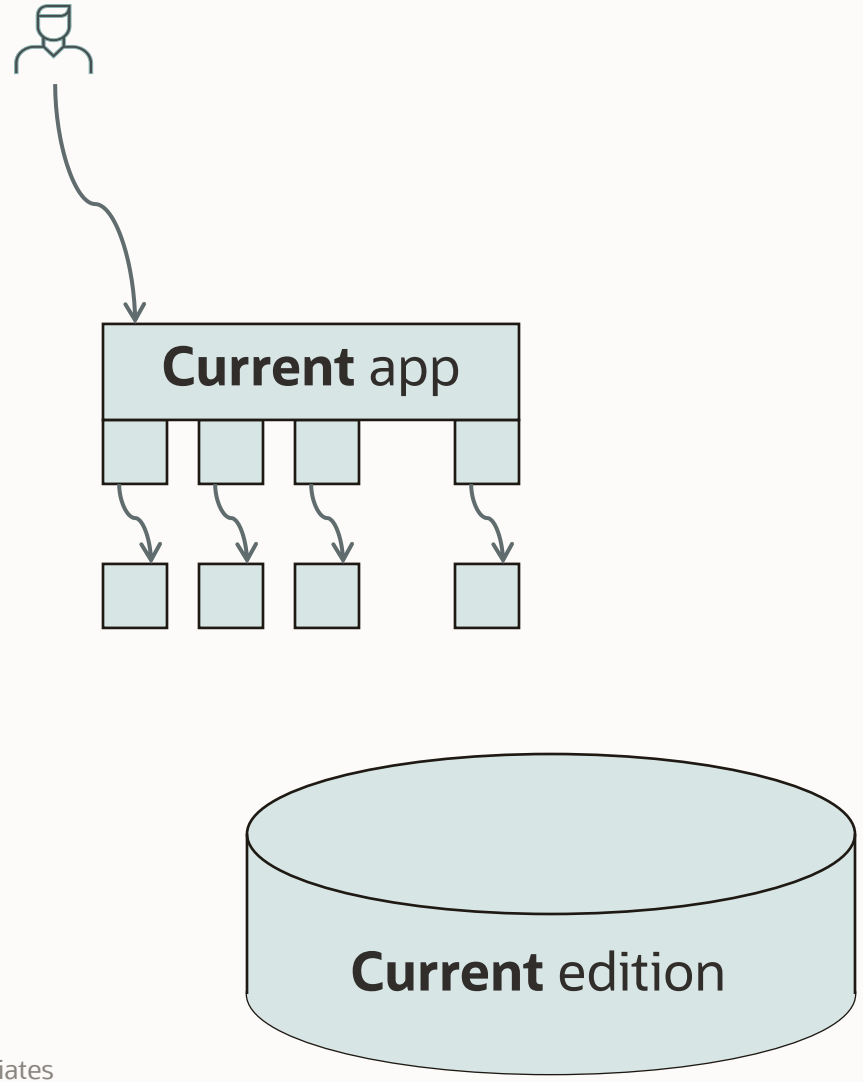


Hot rollover across the stack

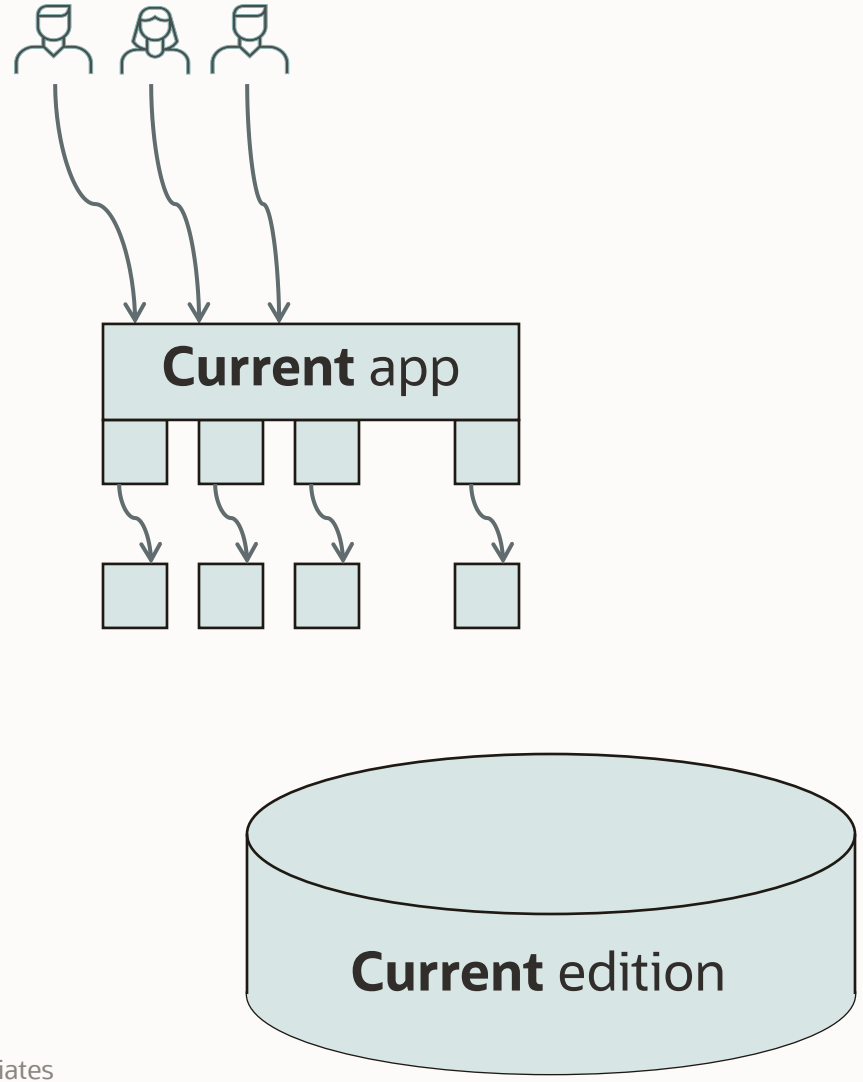




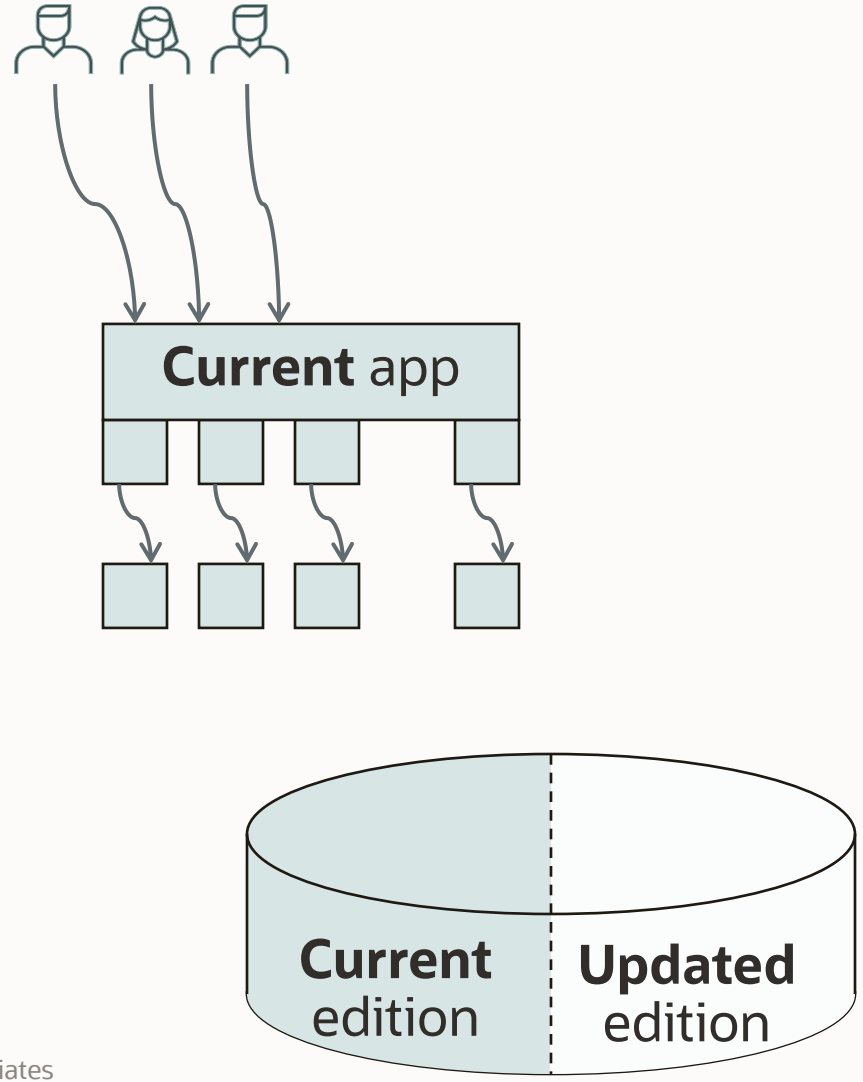
Hot rollover across the stack



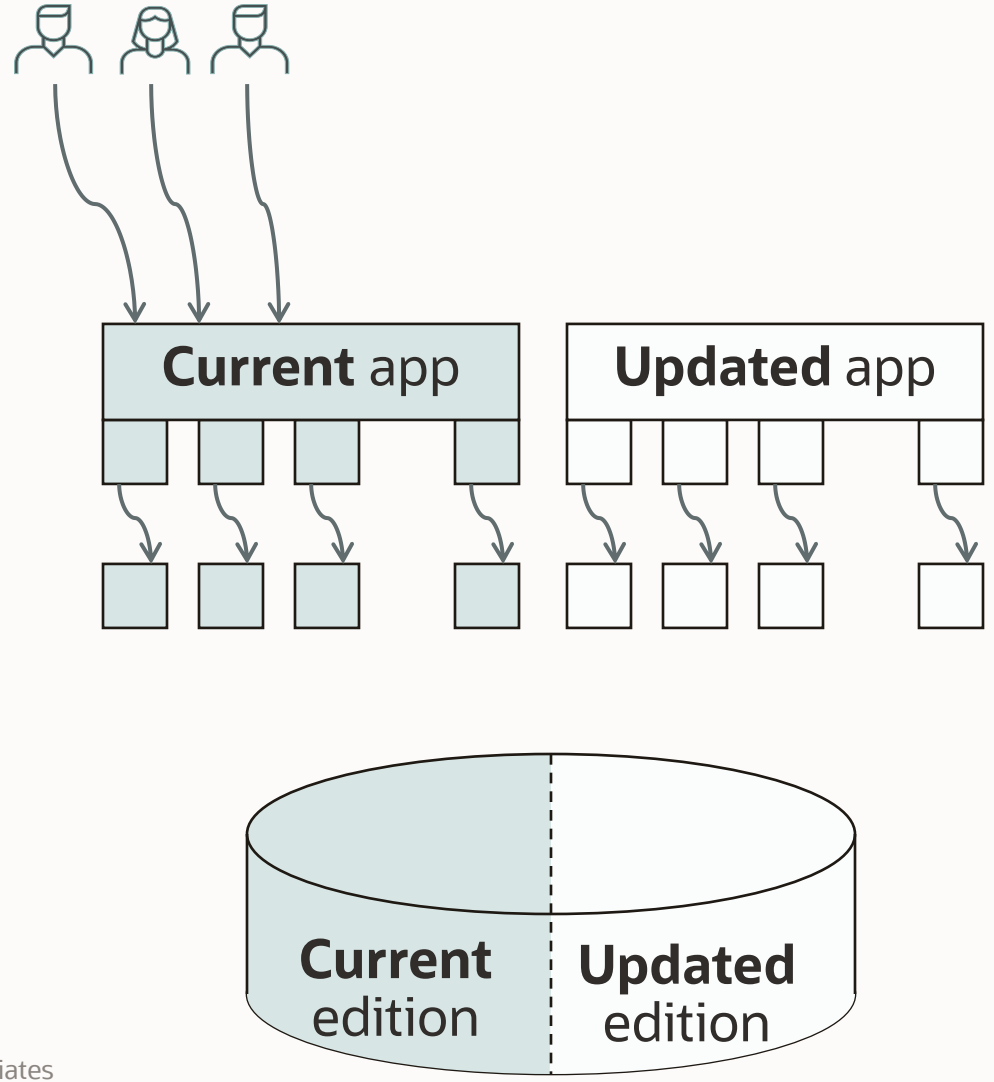
Hot rollover across the stack – Normal use (pre-upgrade)



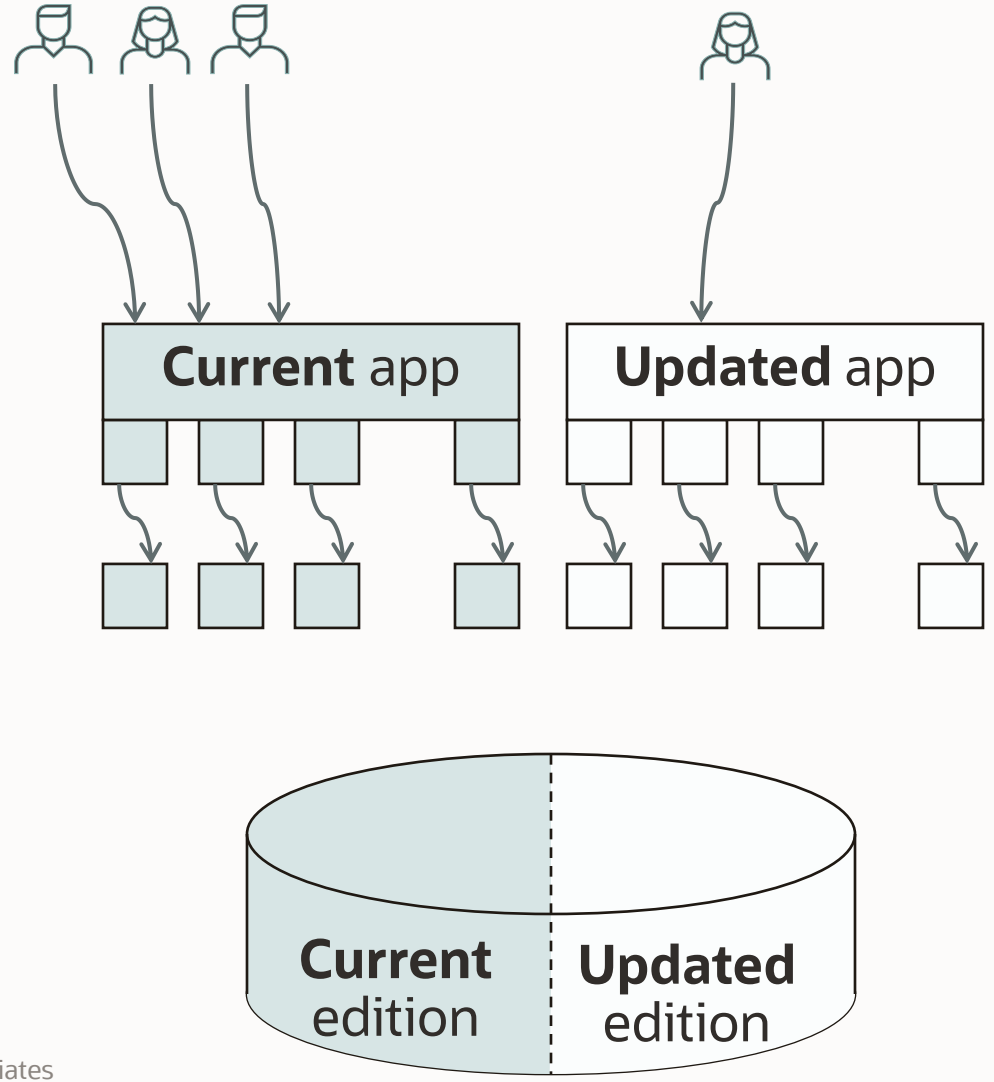
Hot rollover across the stack – Upgrade begins



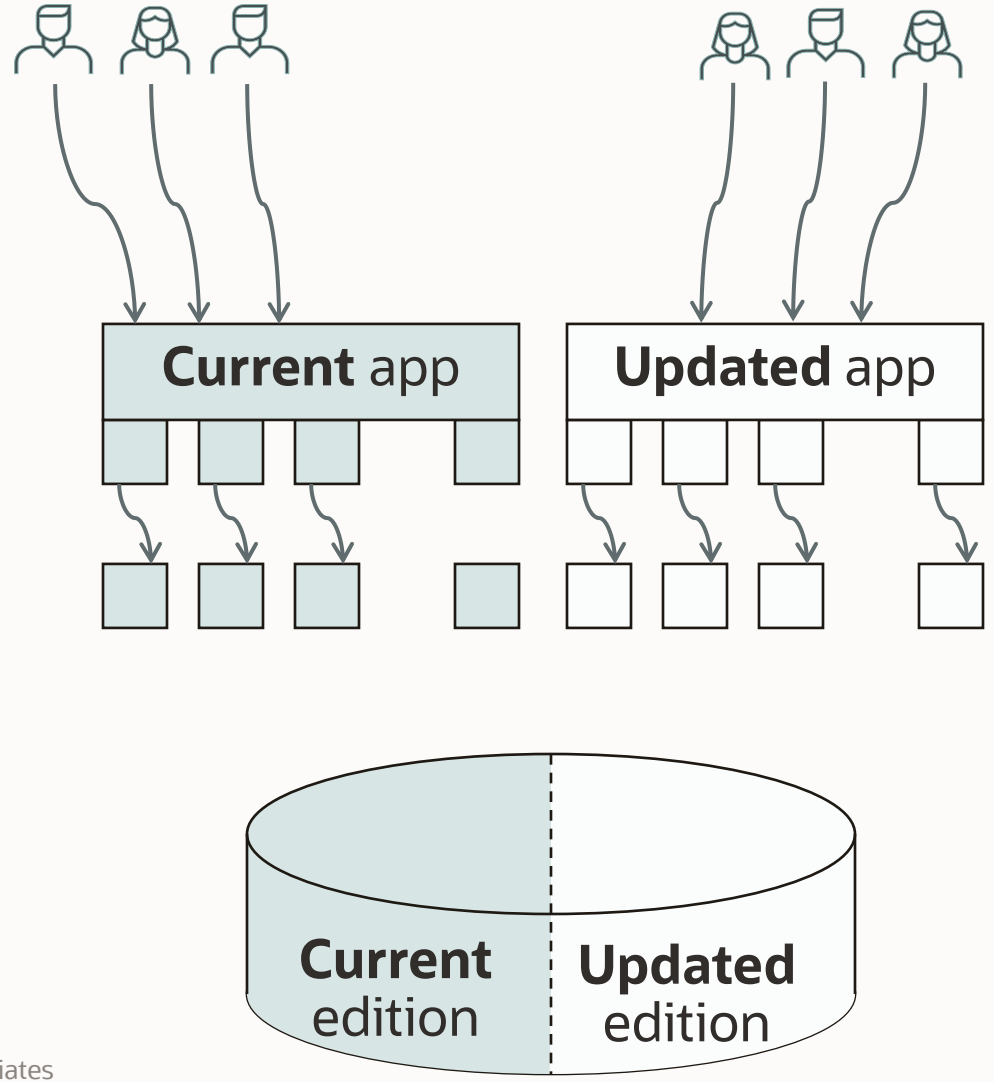
Hot rollover across the stack – Upgrade complete



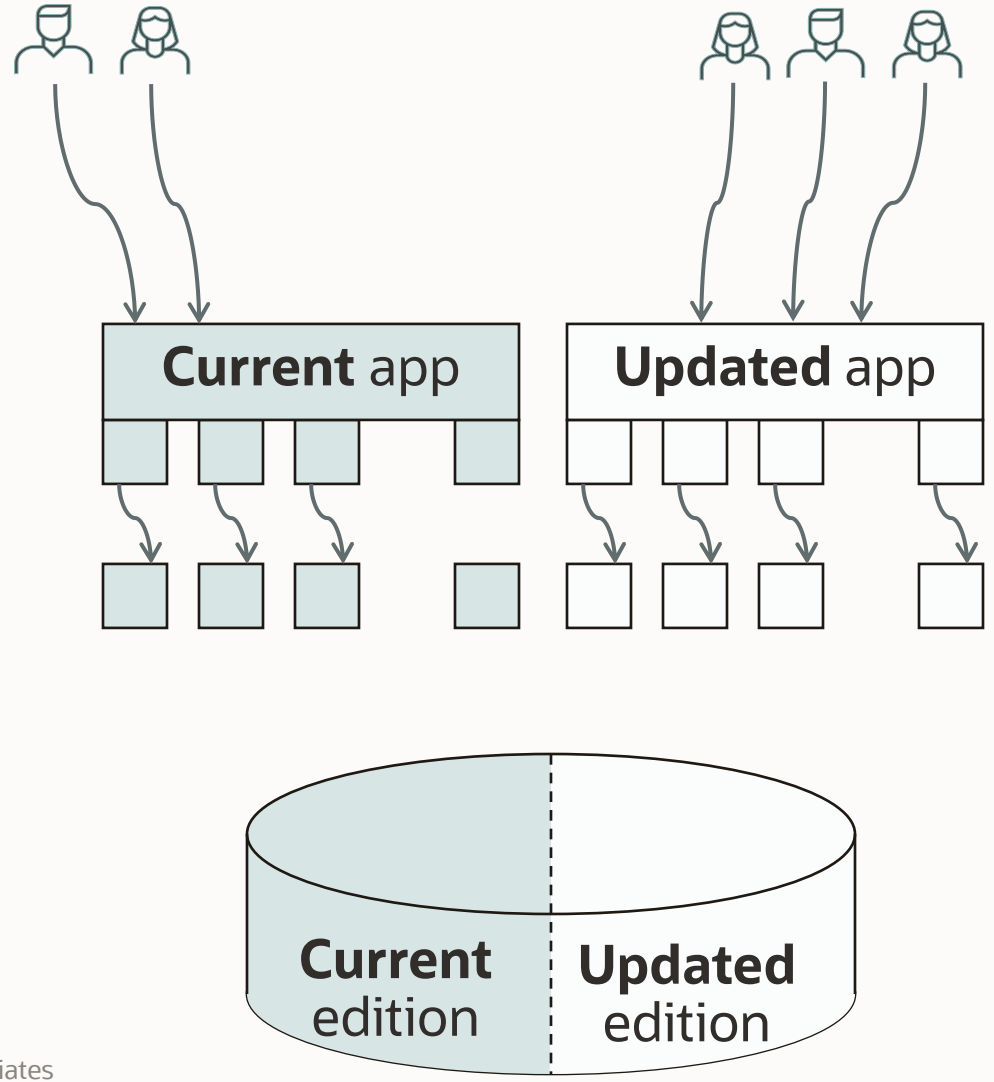
Hot rollover across the stack – Hot rollover begins!



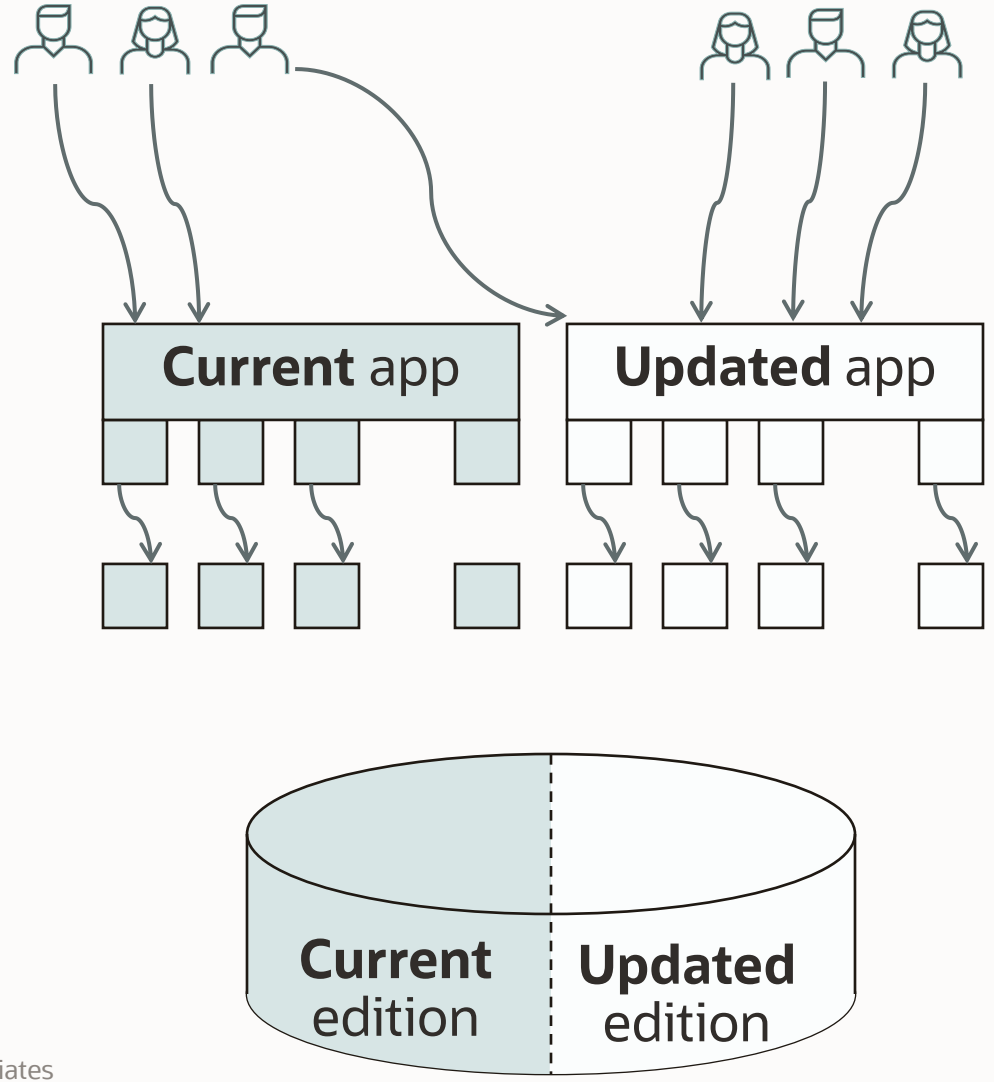
Hot rollover across the stack



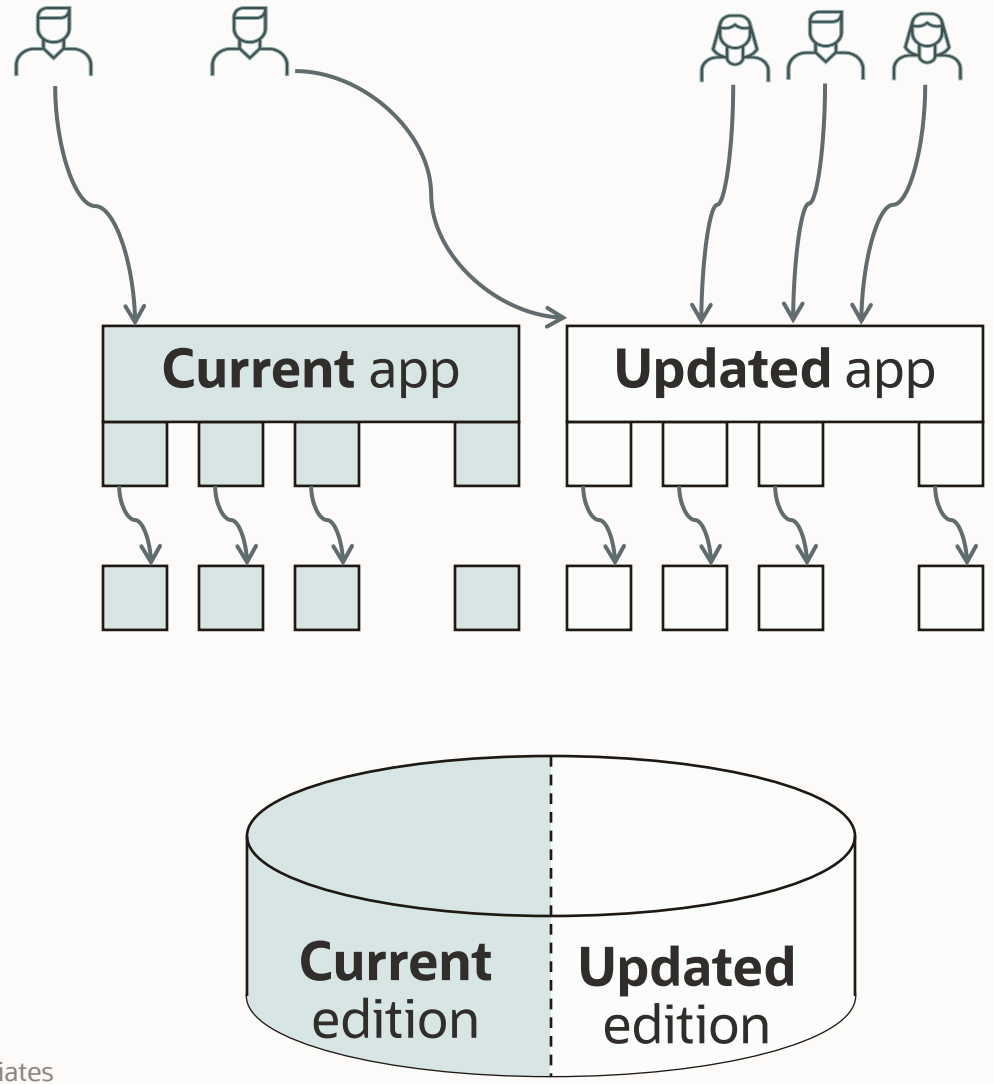
Hot rollover across the stack



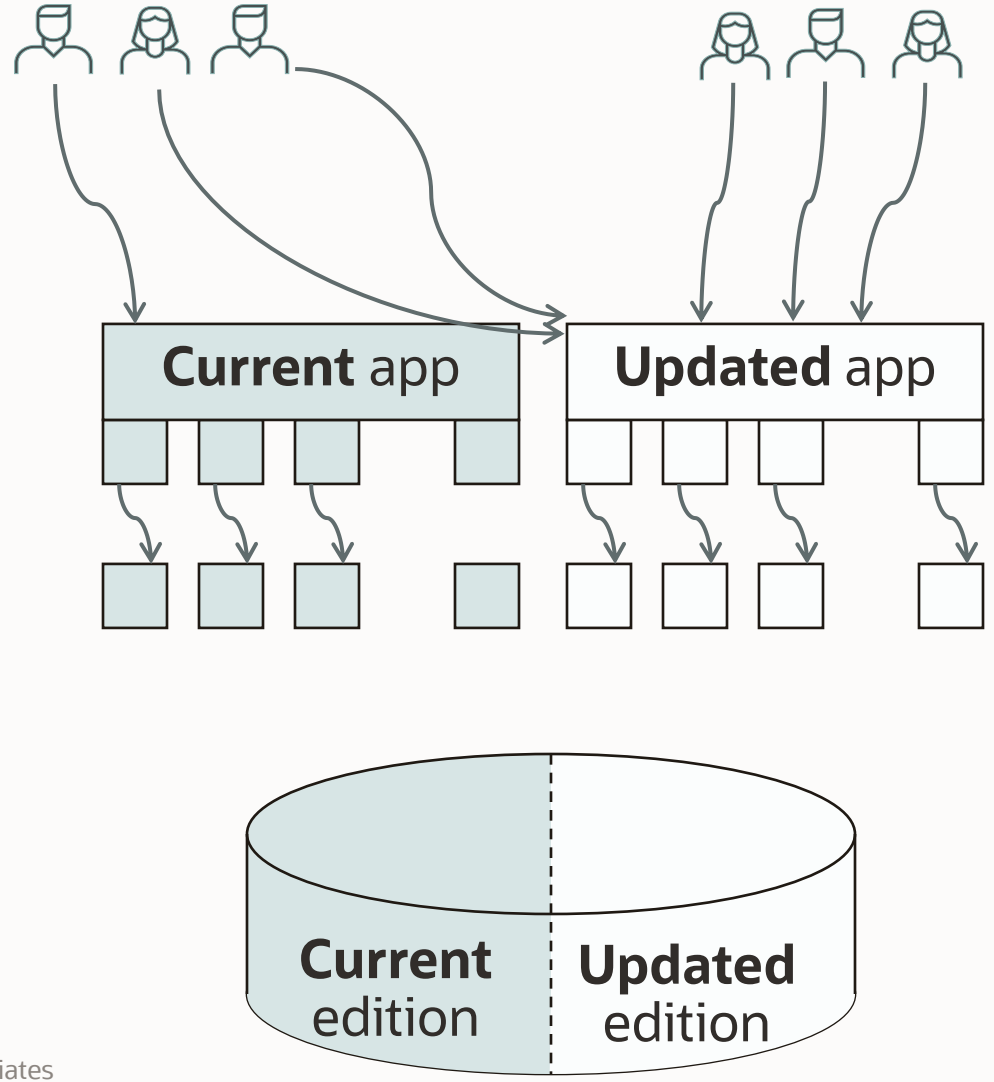
Hot rollover across the stack



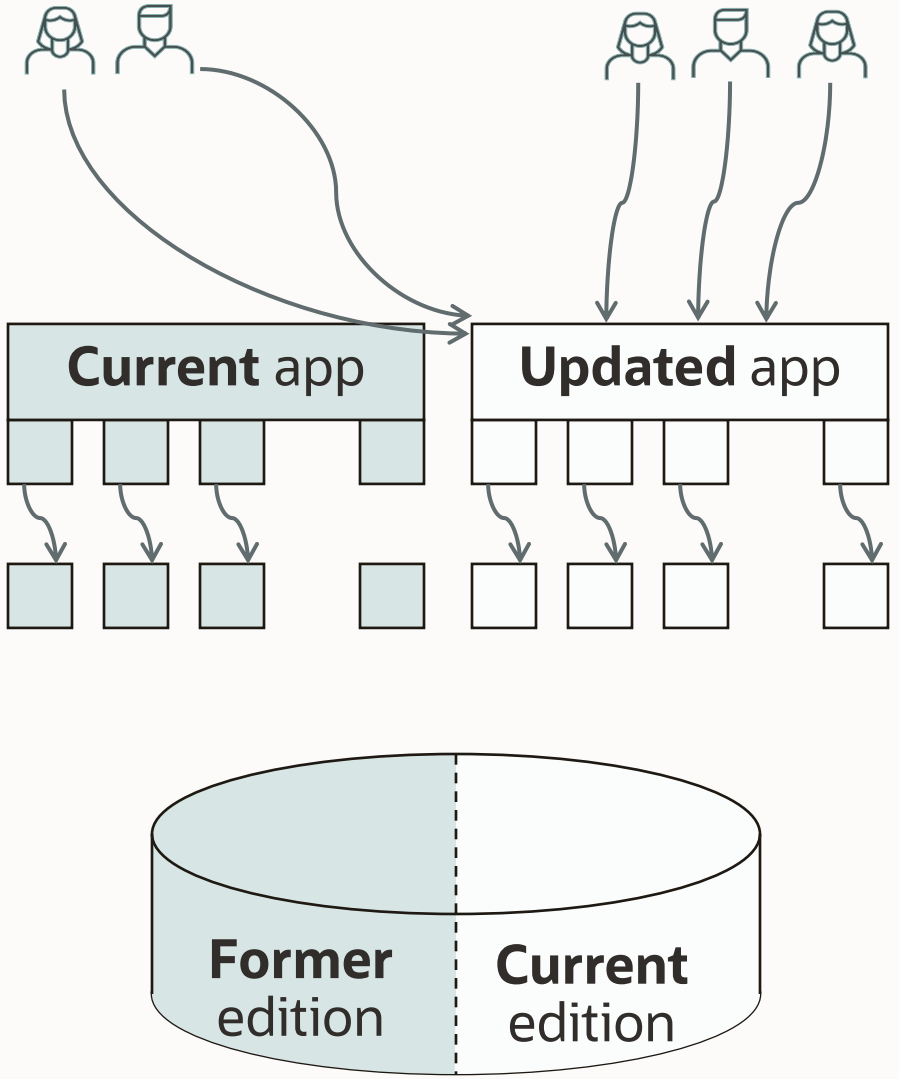
Hot rollover across the stack



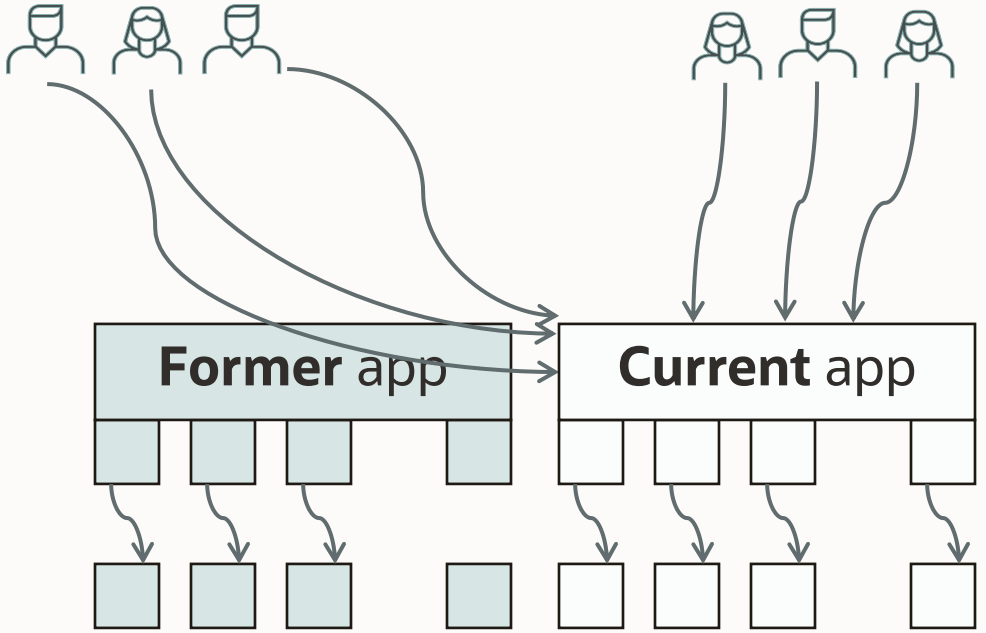
Hot rollover across the stack

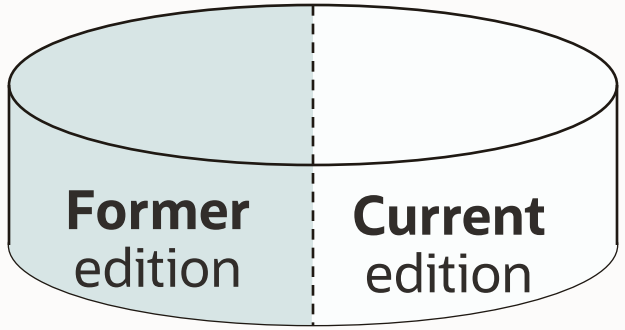


Hot rollover across the stack – hot rollover ends!

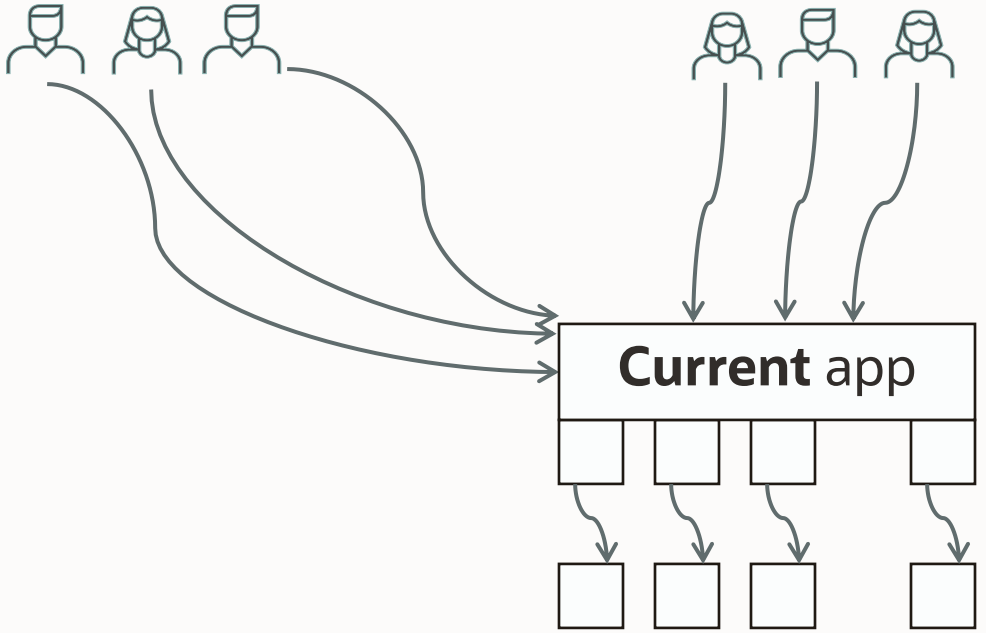


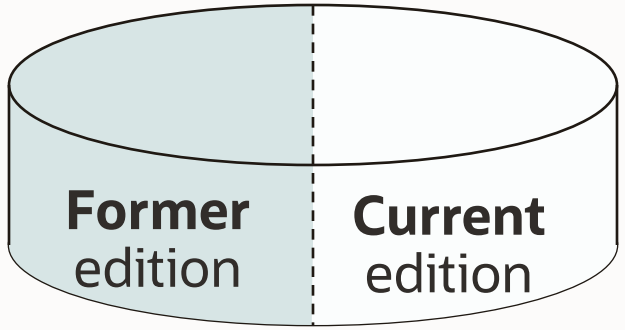
Hot rollover across the stack – normal use (post-upgrade)



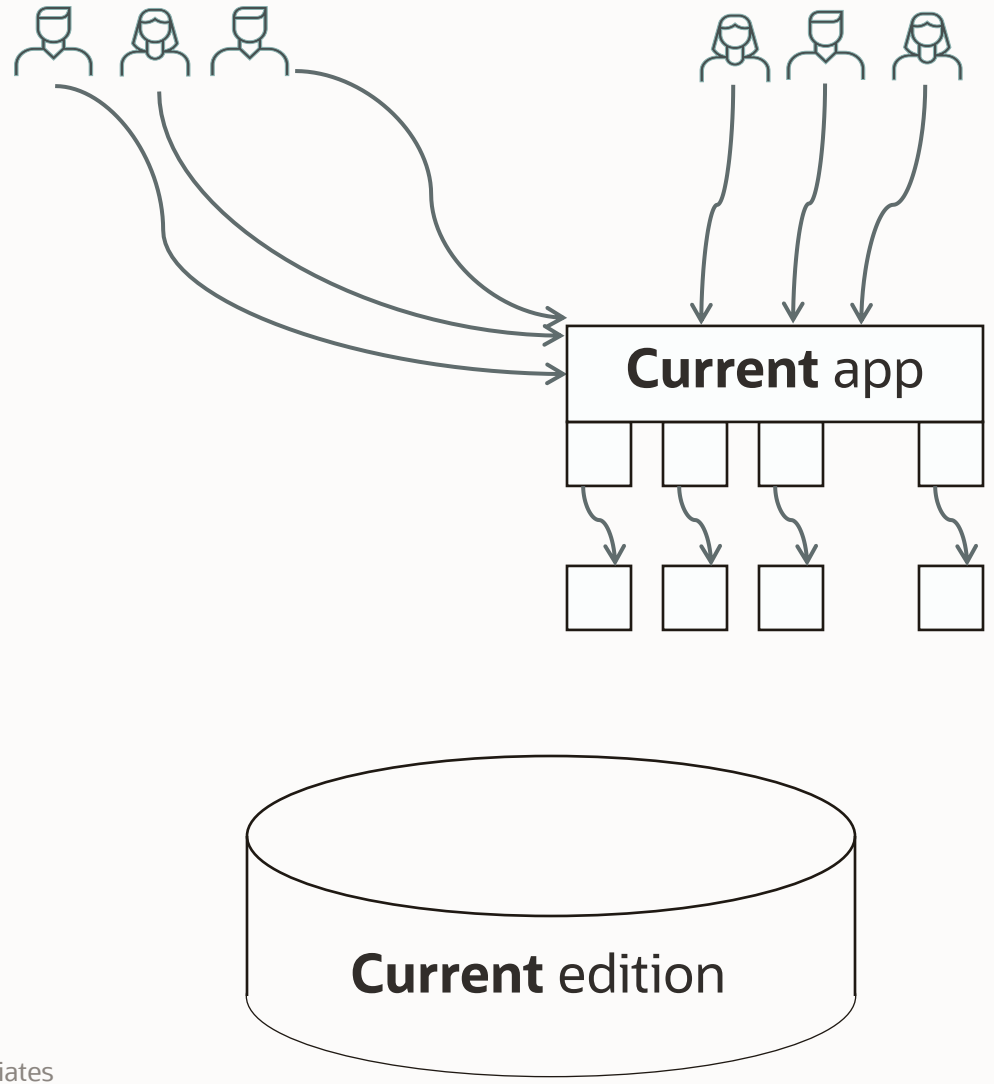


Hot rollover across the stack – cleanup





Hot rollover across the stack



Case Study

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           20
Valli            Pataballa        01
Diana            Lorentz          01
Nancy            Greenberg        43
Daniel           Faviet           66
John             Chen             18
Ismael           Sciarra          23
```

2_Show_20_Rows_Repeat_Post_Upgrade

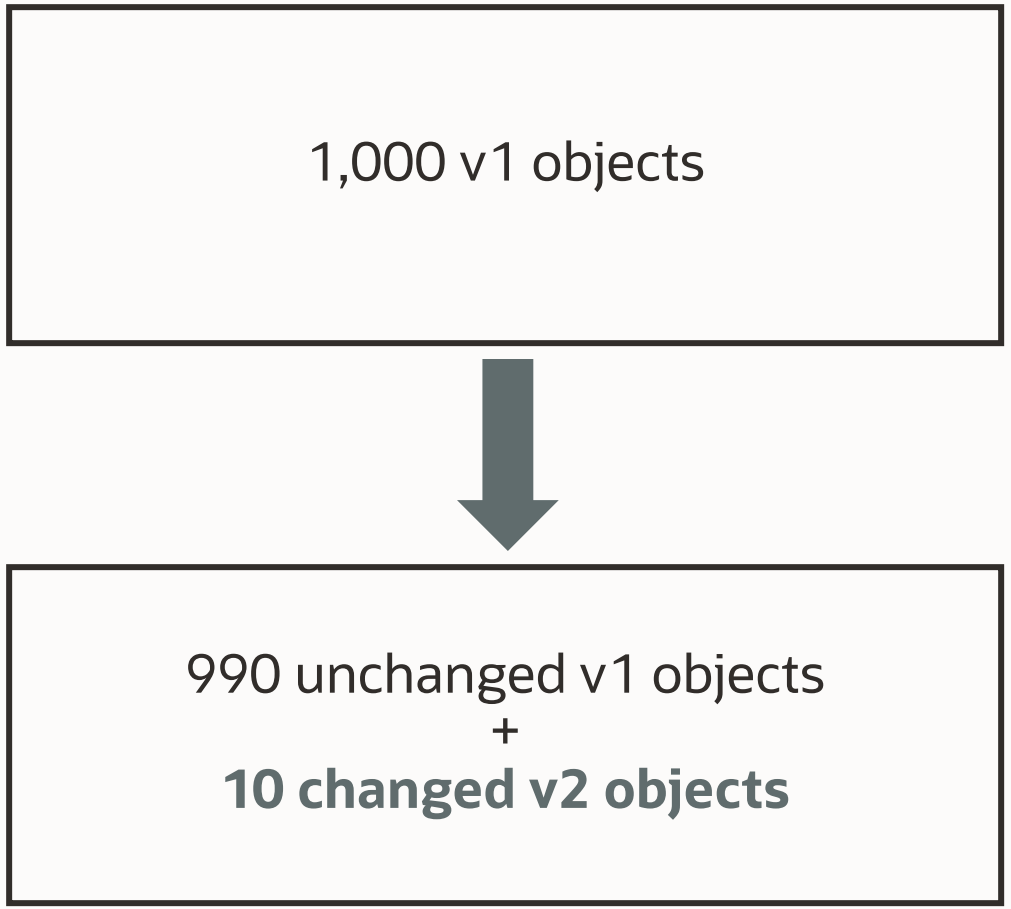
```
First Name       Last Name        Cntry # 
---------------- ---------------- ----- ------------
Steven           King               +32 242-647-4719
Neena            Kochhar             +1 708-108-8233
Lex              De Haan             +1 205-621-9819
Alexander        Hunold             +38 209-317-1291
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
```

The edition – setting the stage

Scenario – for now think only about synonyms, views, and PL/SQL

- 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

The edition – *setting the stage*



The edition – setting the stage

- You can’t change the 10 objects in place as it would change the pre-upgrade app
- An old and a new occurrence of the “same” object cannot 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 edition extends the naming mechanism

- 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)

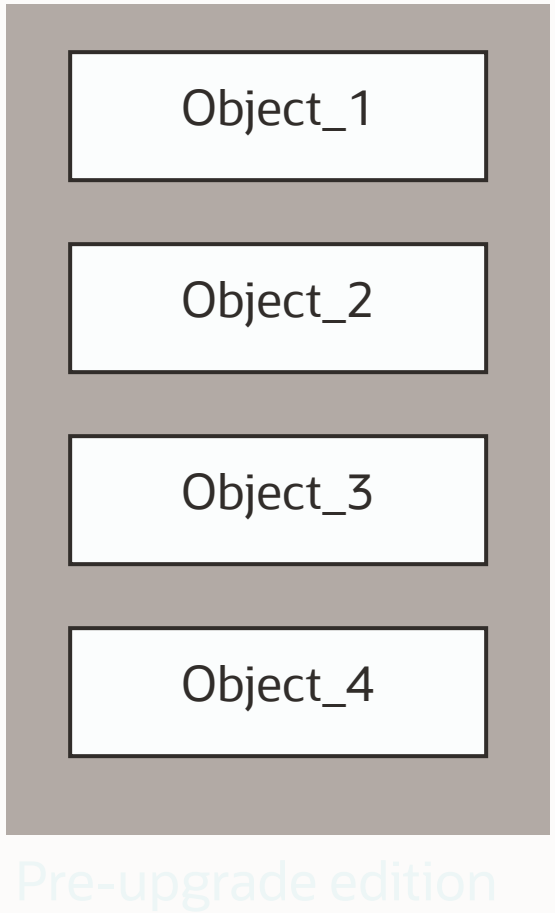
The edition extends the naming mechanism

- An object is identified by its name and its owner
- 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

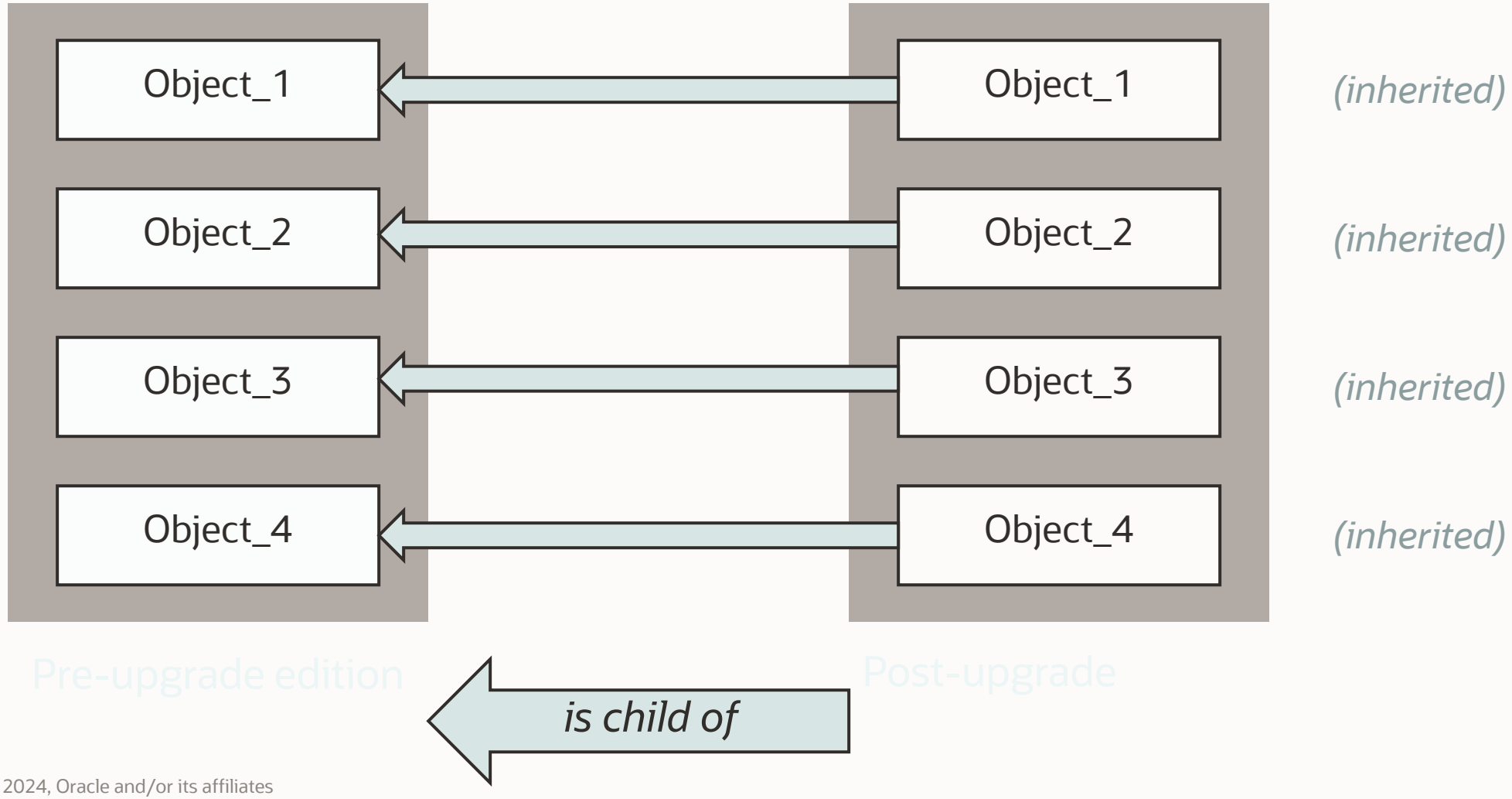
The semantic model for “create edition”

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

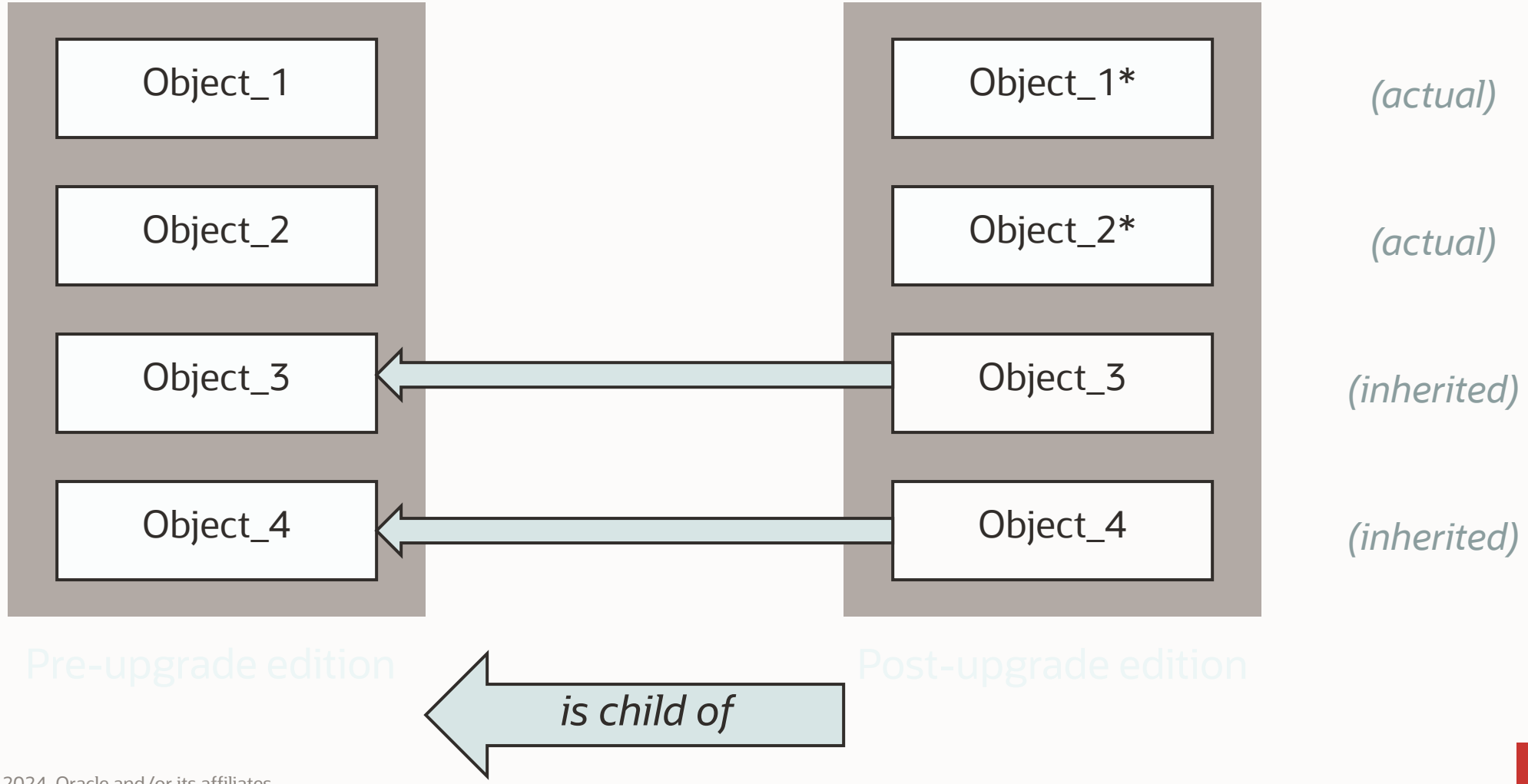
The mental model of the implementation



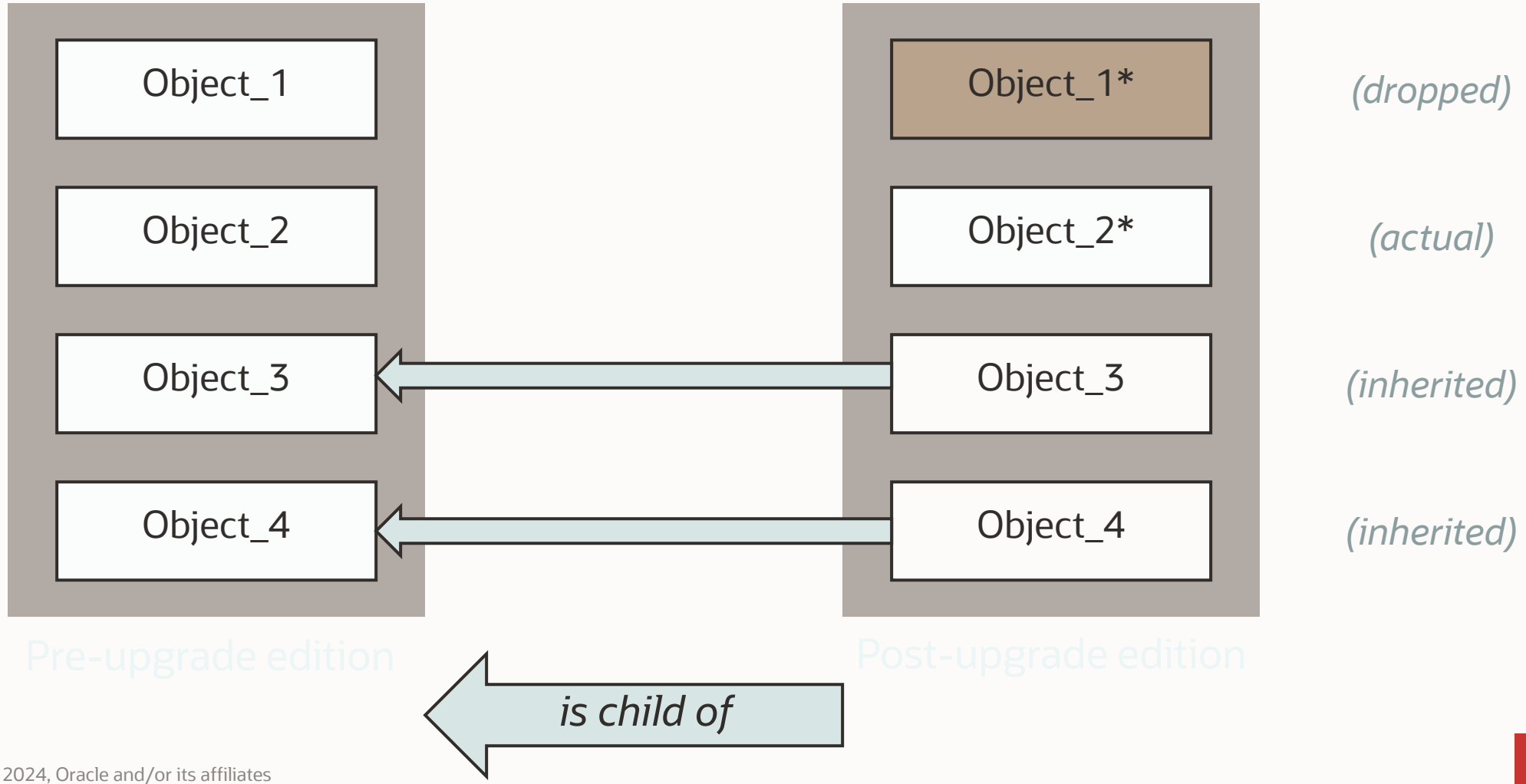
The mental model of the implementation



The mental model of the implementation



The mental model of the implementation



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!

Editionable and noneditionable object types

1. Not all object types are editionable
   - Synonyms, views, and PL/SQL units of all kinds (including, therefore, triggers and libraries), and are editionable
   - Objects of all other object types – for example tables – are noneditionable

2. You need to 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 editionable

Editioning views

- An editioning view (EV) may only project and rename columns
- You can't have more than one editioning view for a 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
- Like all views, an editioning view can be read-only

Editioning views – performance

- 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

Editioned and noneditioned objects – slight return

- An object whose type is non-editionable 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)
- You control the editioned state at the granularity of the single name
- Theorem (the NE-on-E prohibition): a non-editioned 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

- These objects 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 will be done
    - within the closure of the object's static dependency parents (at compile time)
    - and for those objects that are identified dynamically during SQL execution (at run time)
  - The usable edition range explicitly specifies the set of adjacent editions within which the optimizer will consider the object, for query re-write when computing the execution plan

Public synonyms

- A public synonym is just a synonym that happens to be owned by the Oracle-maintained user called “public”
- “public” user is editions enabled, but all existing public synonyms are non-editioned at the per-object level
- As of version 12.1+, you can make your own public synonyms editioned at the per-object level

Tables with UDT columns

- An ordinary column (as opposed to virtual) cannot specify the evaluation edition or the usable 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

The crossedition trigger

- Of course, DML does not stop during online application 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 made to pre-upgrade columns into the post- upgrade columns – and vice versa

The crossedition trigger

- Crossedition triggers directly access the table
- They have special firing rules
- You create crossedition triggers in the Post_Upgrade edition
  - The paradigm is: don't do any DDLs in the Pre_Upgrade edition
- The firing rules rules assume that
  - Pre-upgrade columns are changed only by sessions using the Pre_Upgrade edition
  - Post-upgrade columns are changed only by sessions using the Post_Upgrade edition

The crossedition trigger

- 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

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. append an underscore, or lowercase the name)
  - 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 by controlling 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
- Of course
  - All indexes on the original Employees table 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_
- This readying work must be done by the developers of the application that adopts EBR

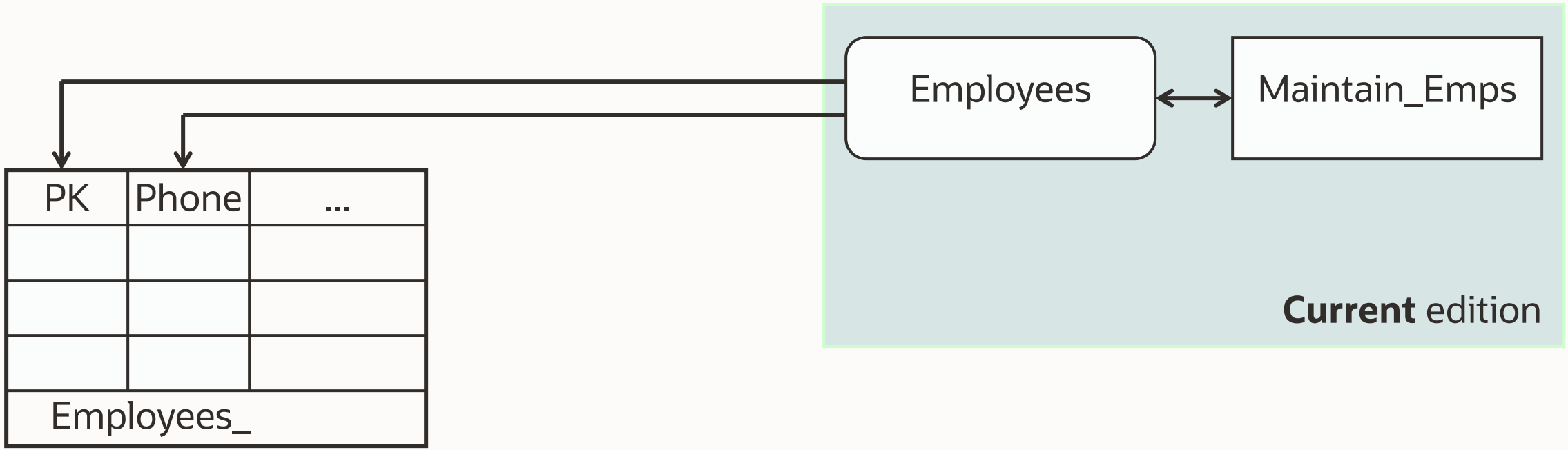
Case study

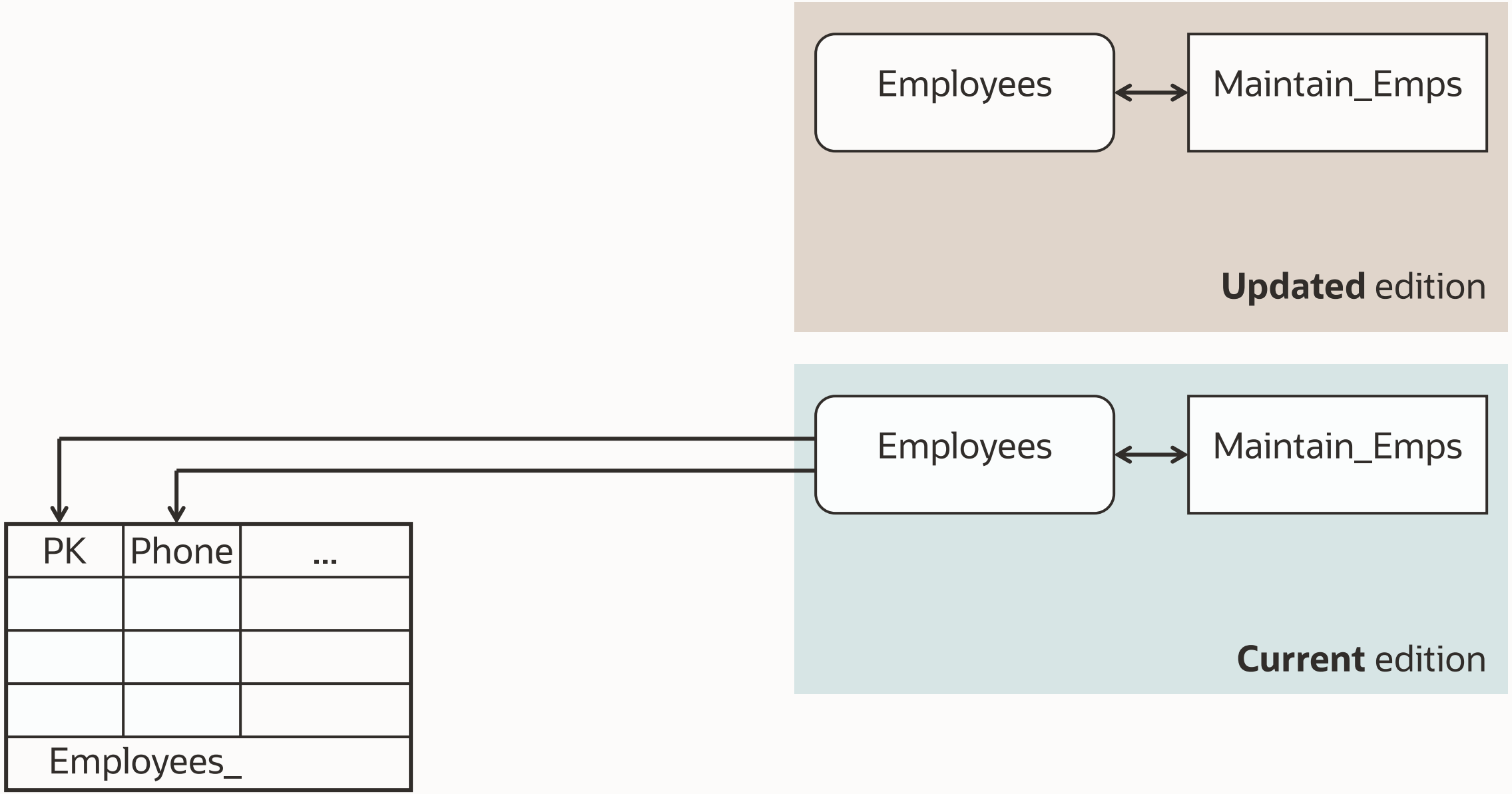
1_Show_20_Rows_Repeat_Pre_Upgrade

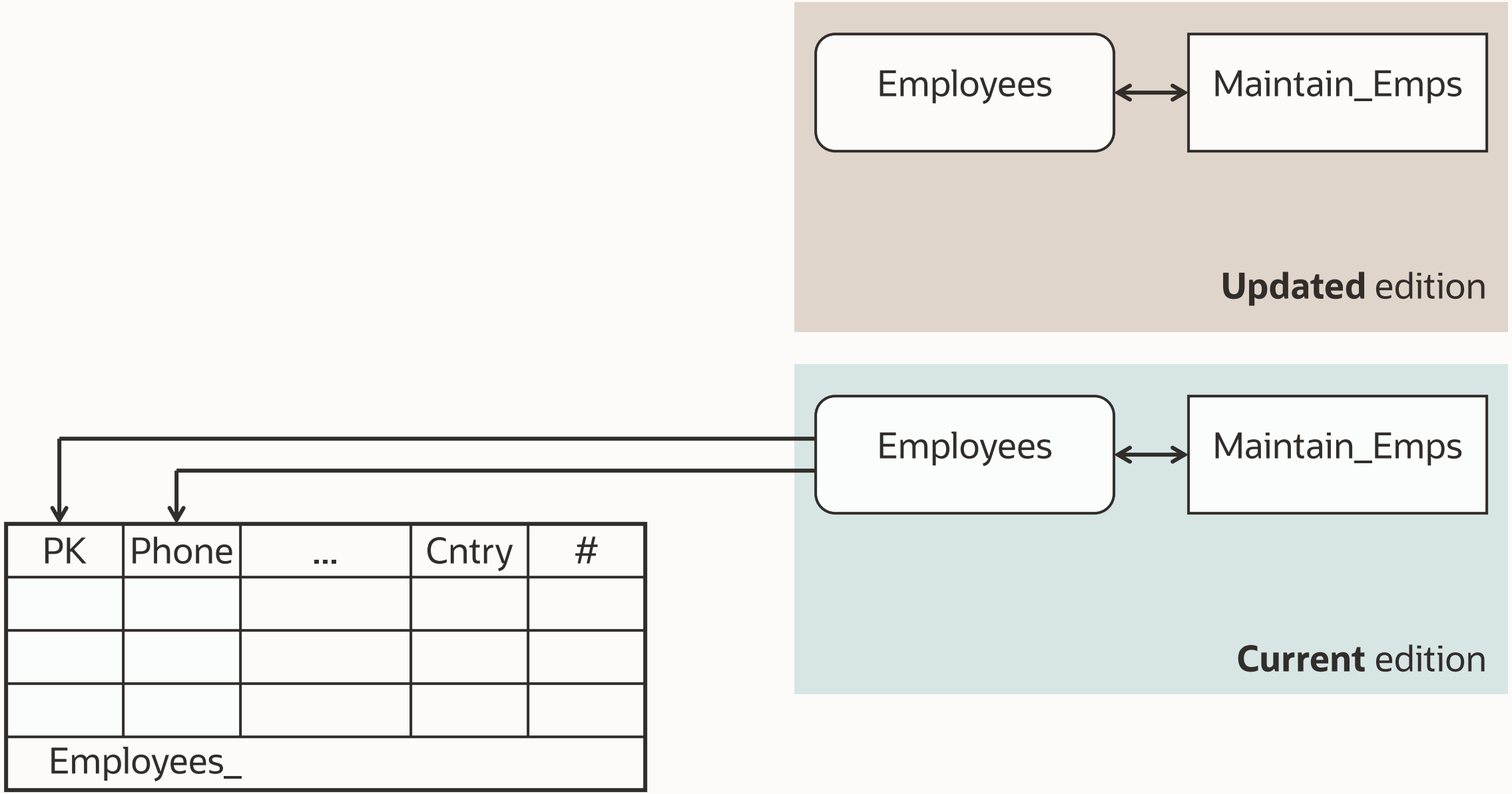
```
First Name           Last Name            Phone
-------------------- -------------------- --------------------
Steven               King                 011.32.242.647.4719
Neena                Kochhar              708.108.8233
Lex                  De Haan              205.621.9819
Alexander            Hunold
Bruce                Ernst
David                Austin
Valli                Pataballa
Diana                Lorentz
Nancy                Greenberg
Daniel               Faviet
John                 Chen
Ismael               Sciarra
```

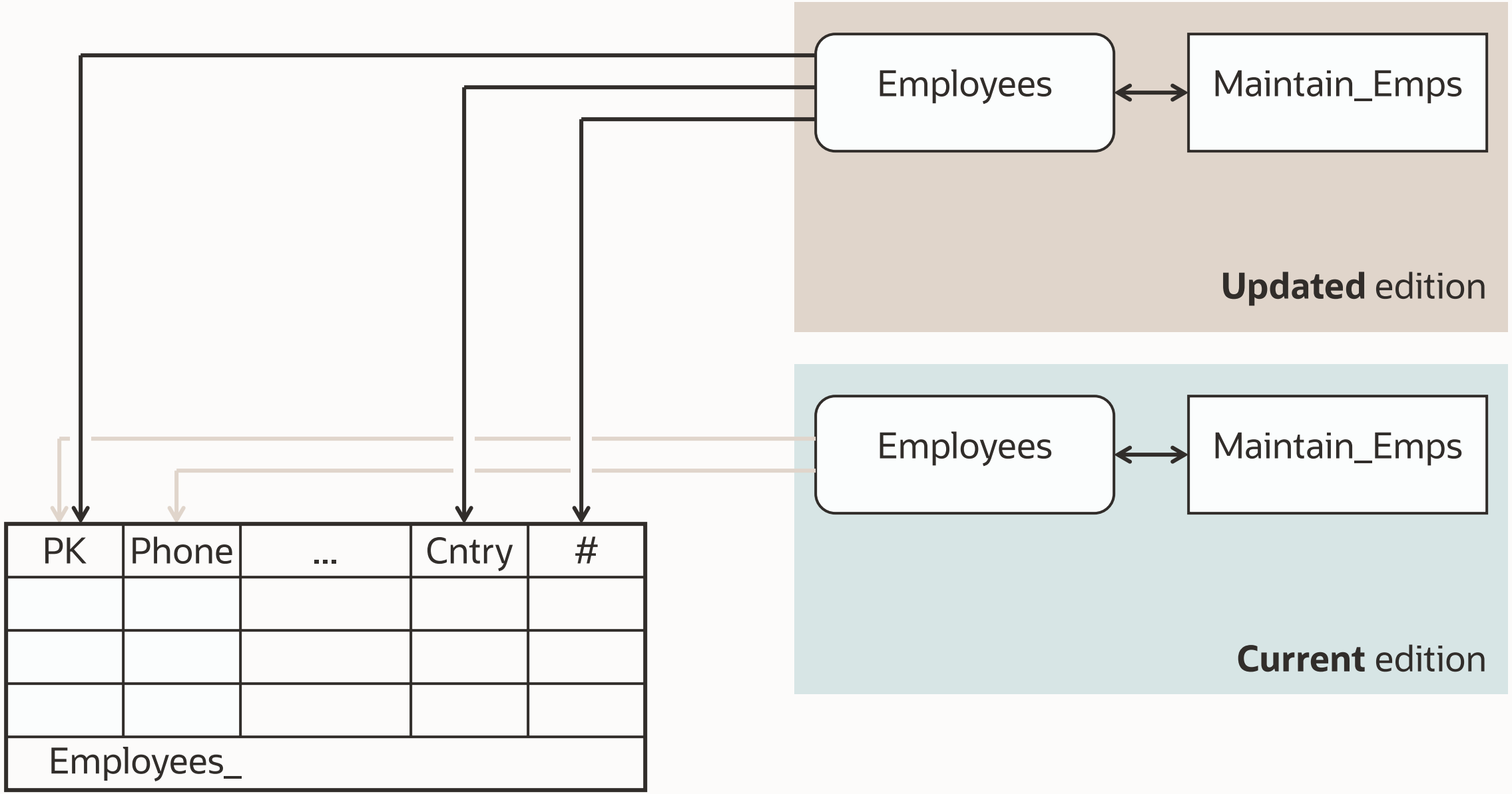
2_Show_20_Rows_Repeat_Post_Upgrade

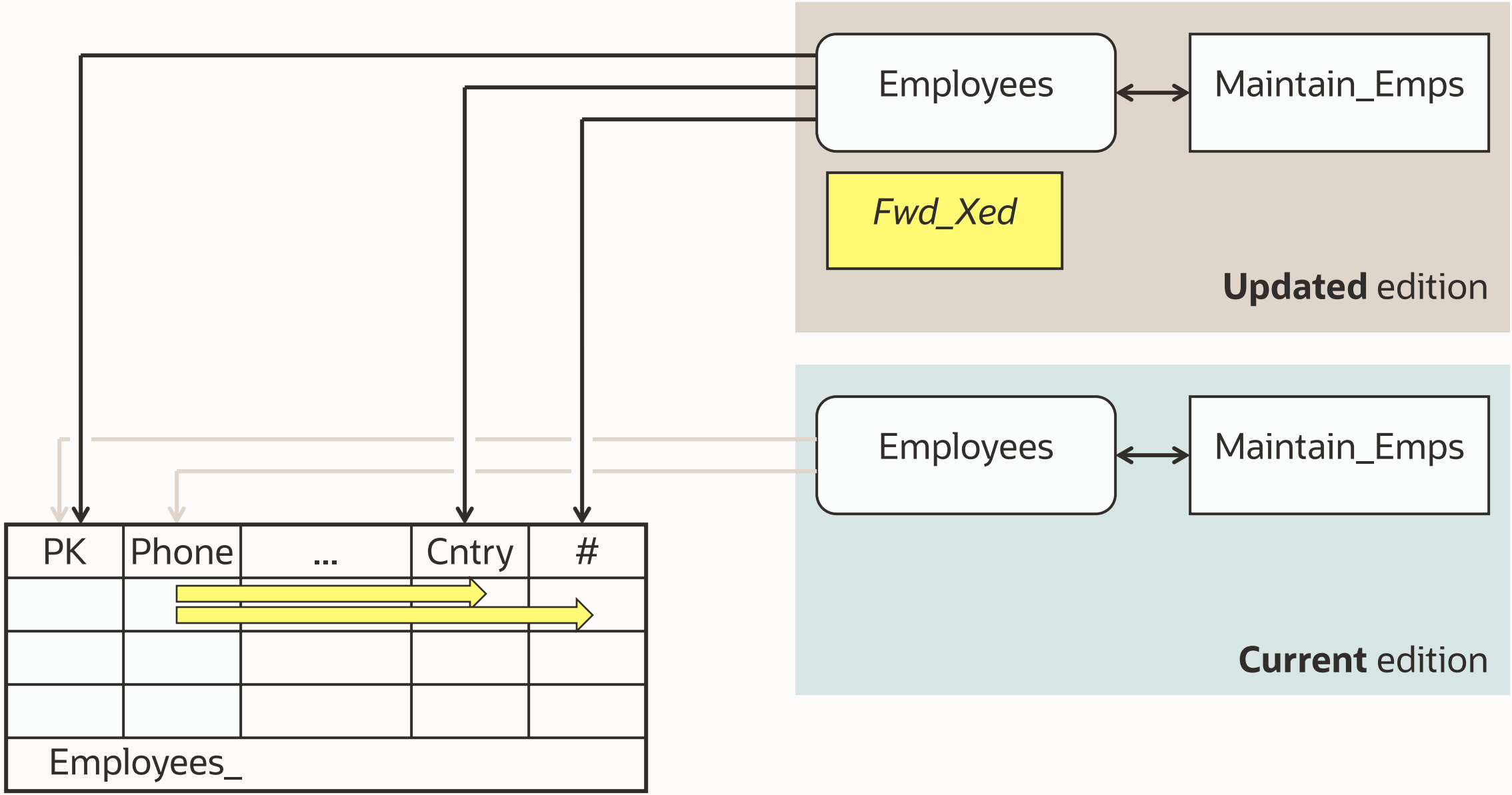
```
First Name           Last Name            Cntry # 
-------------------- -------------------- ----- ------------
Steven               King                   +32 242-647-4719
Neena                Kochhar                 +1 708-108-8233
Lex                  De Haan                 +1 205-621-9819
Alexander            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
```

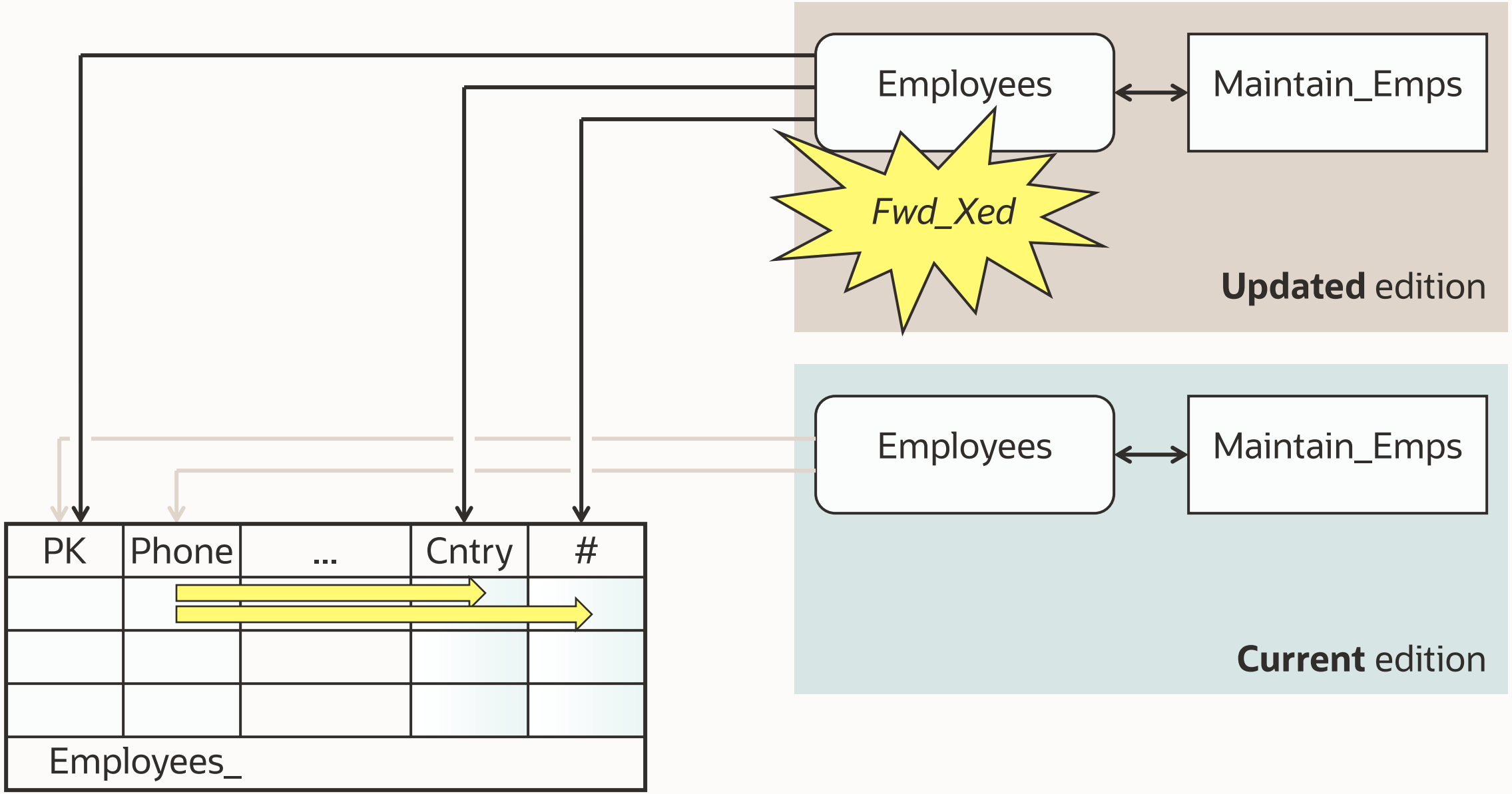


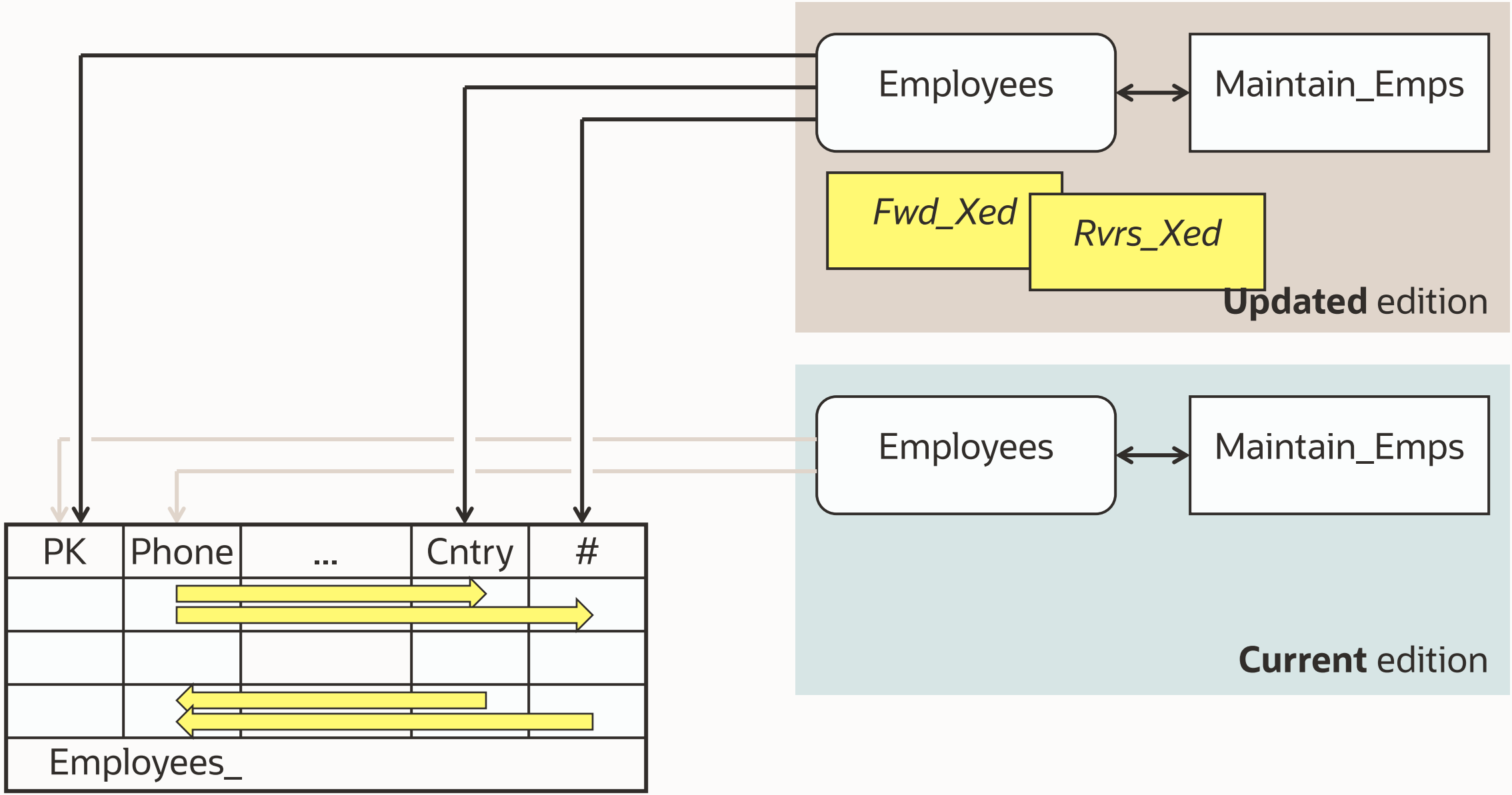


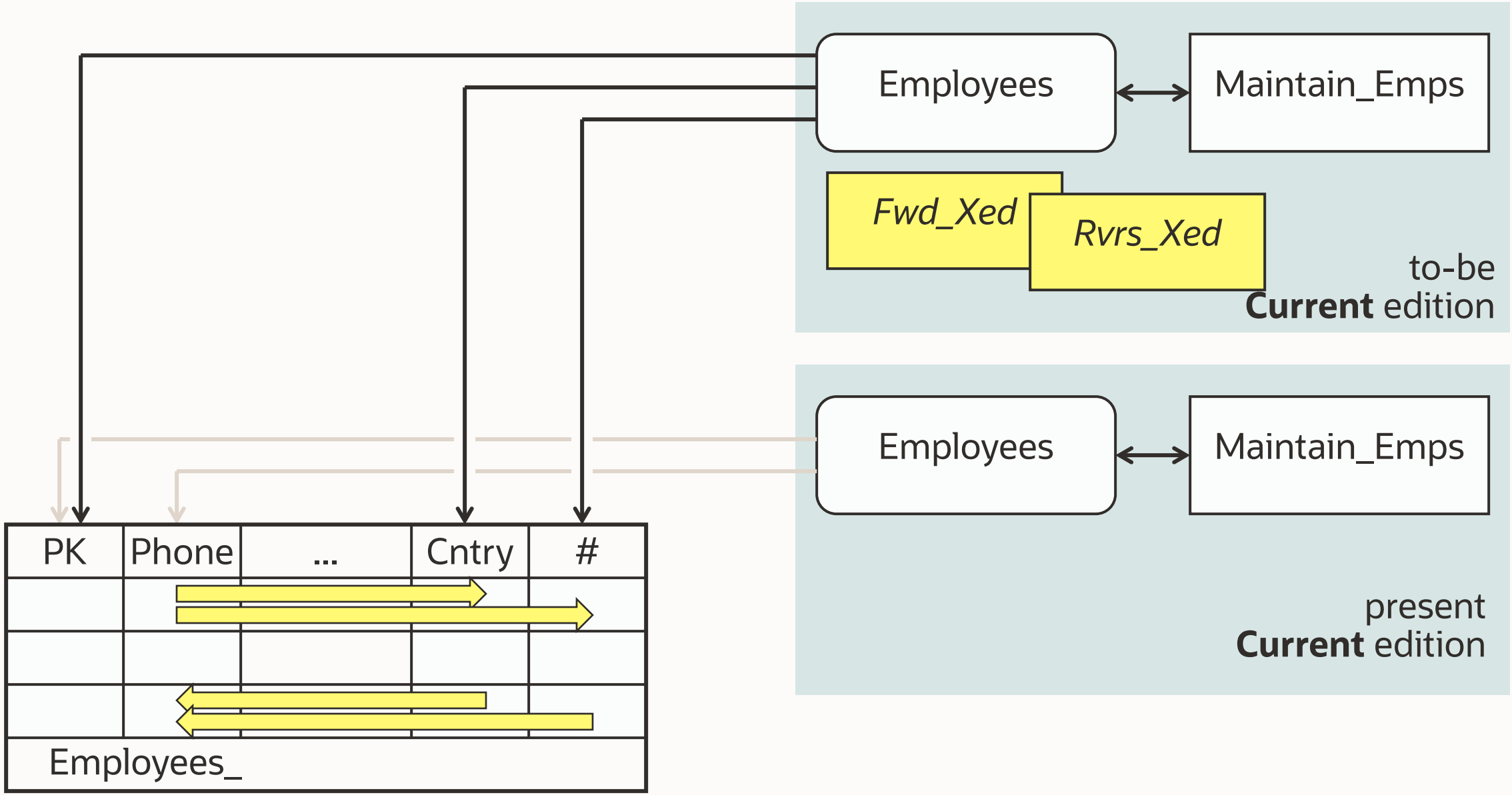


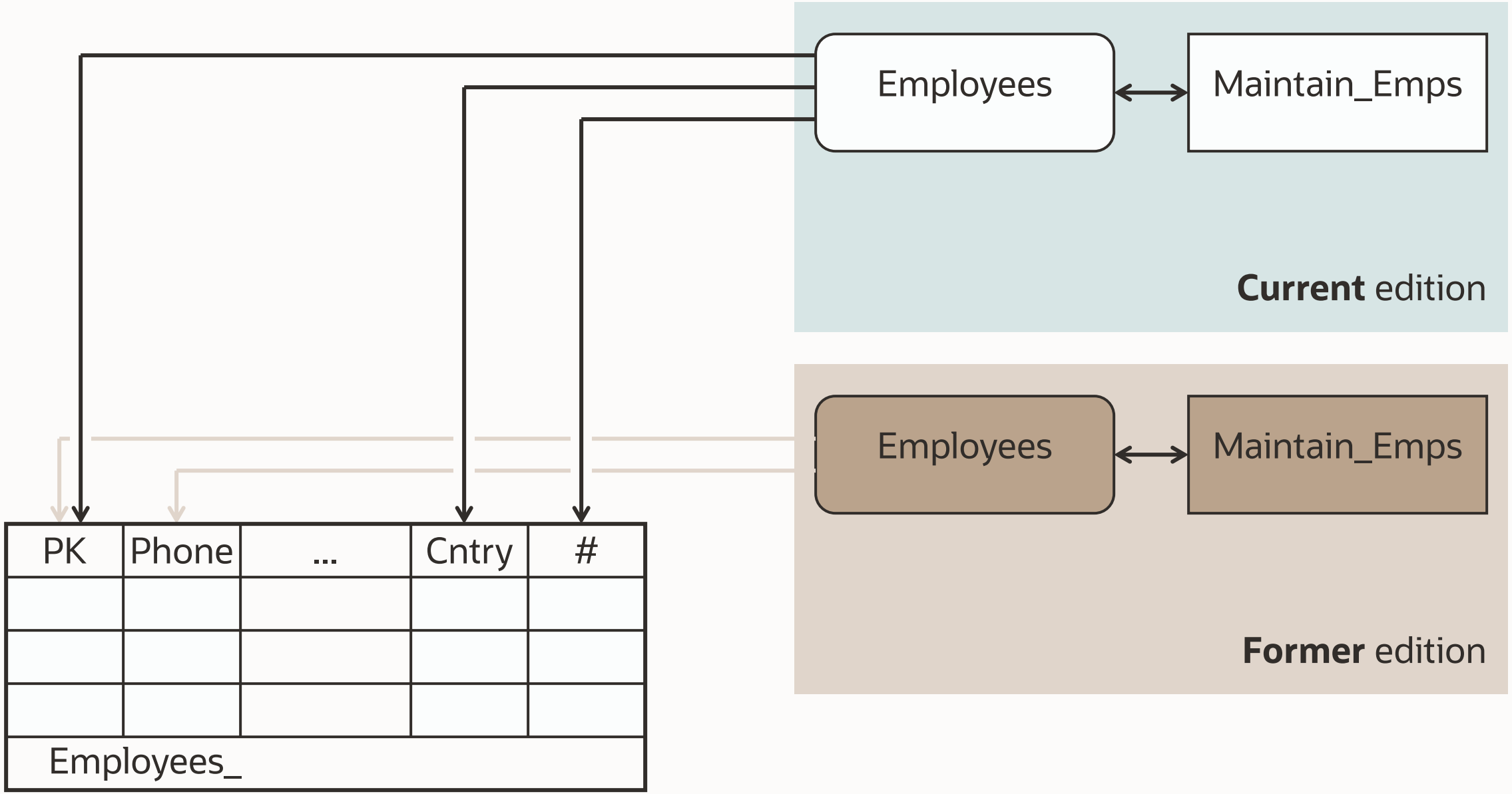


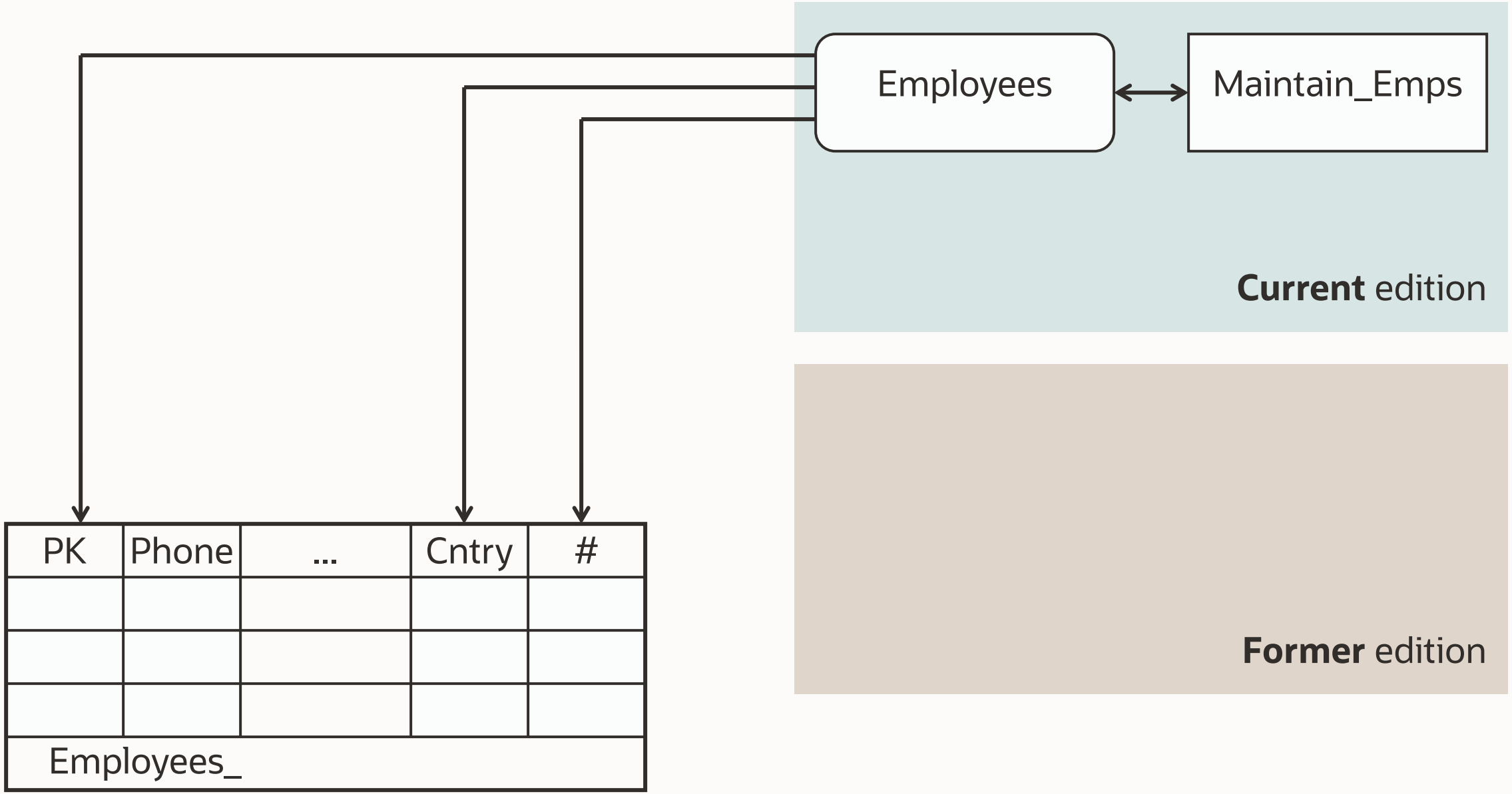


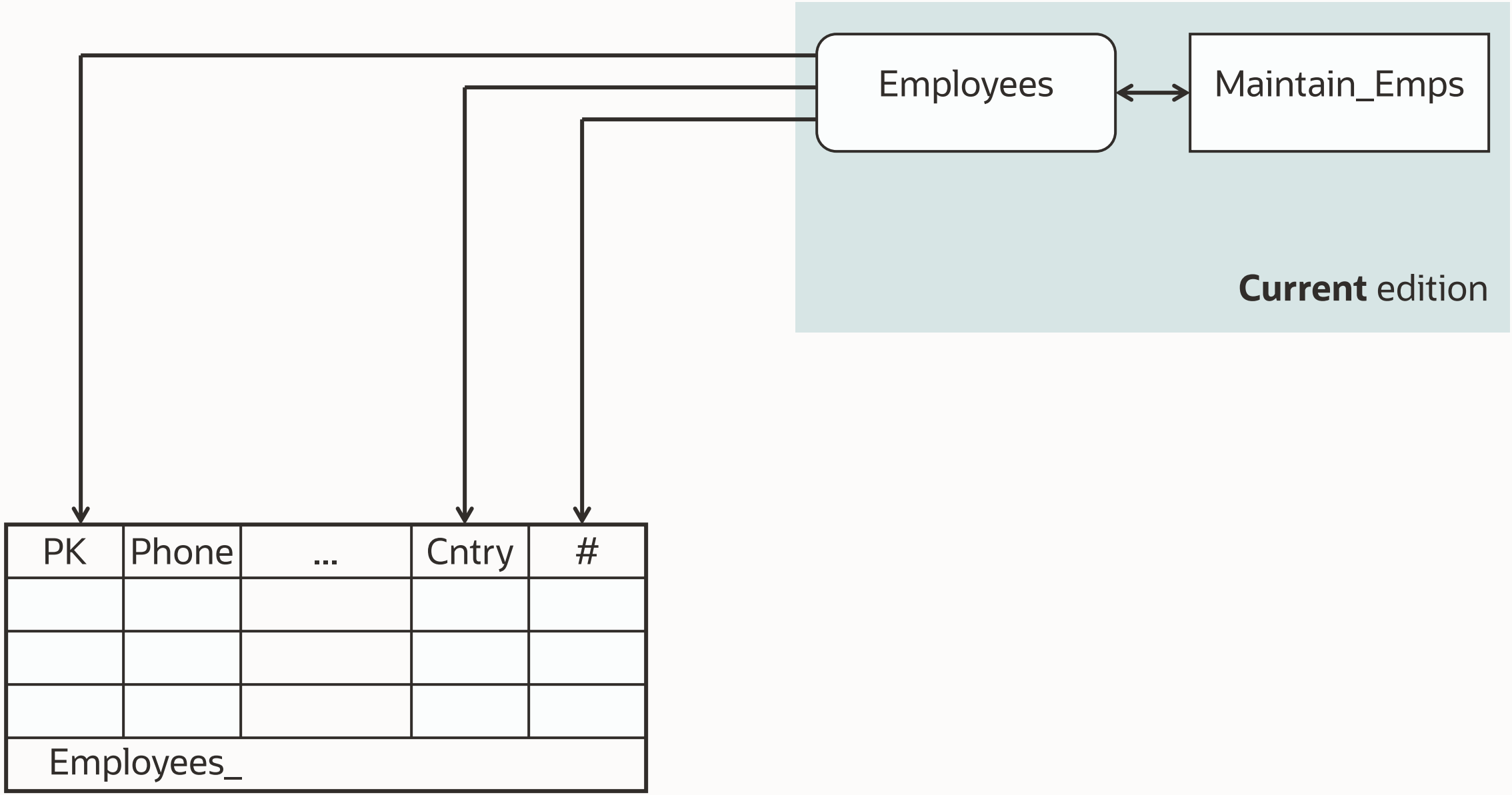


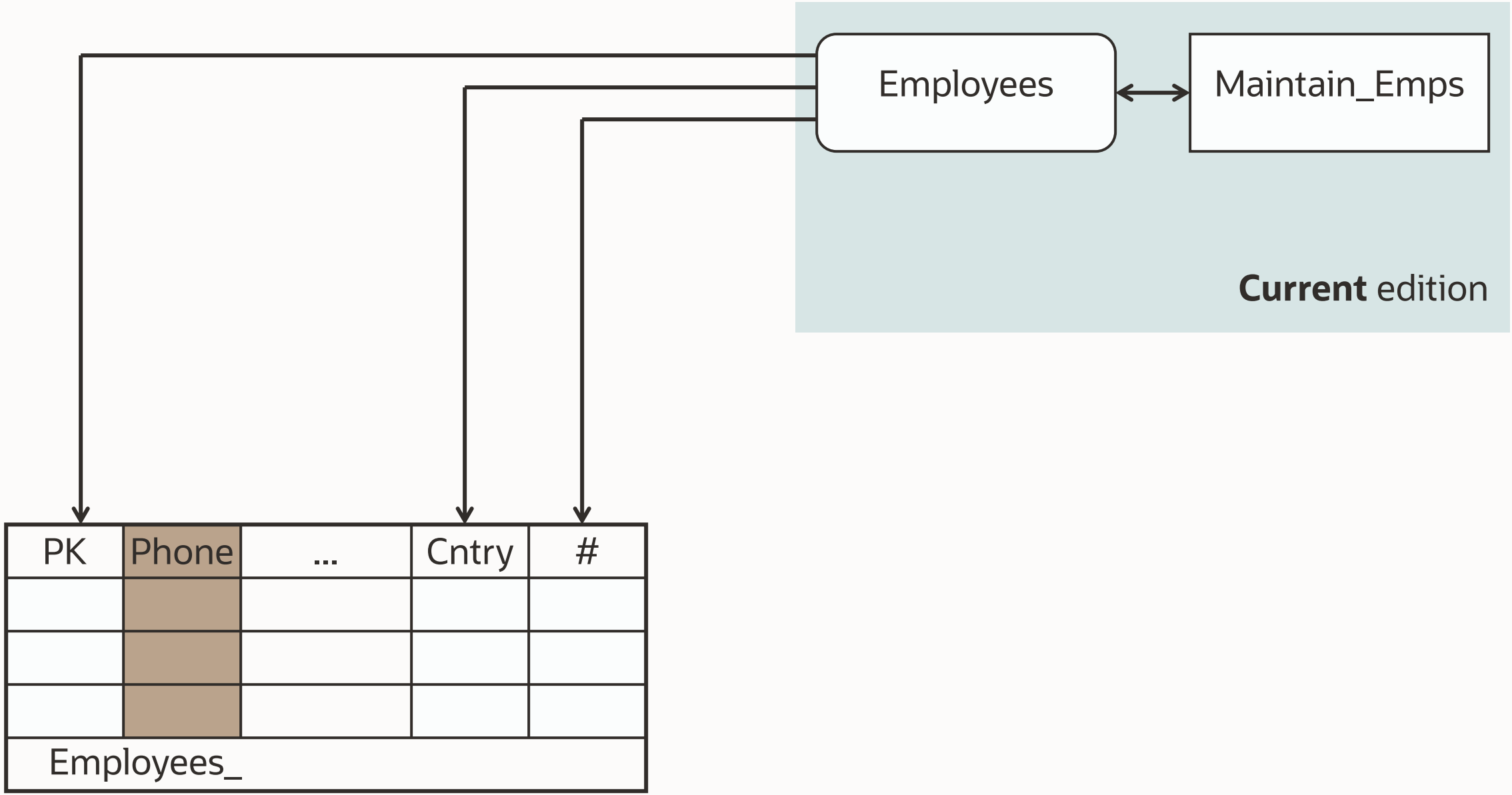








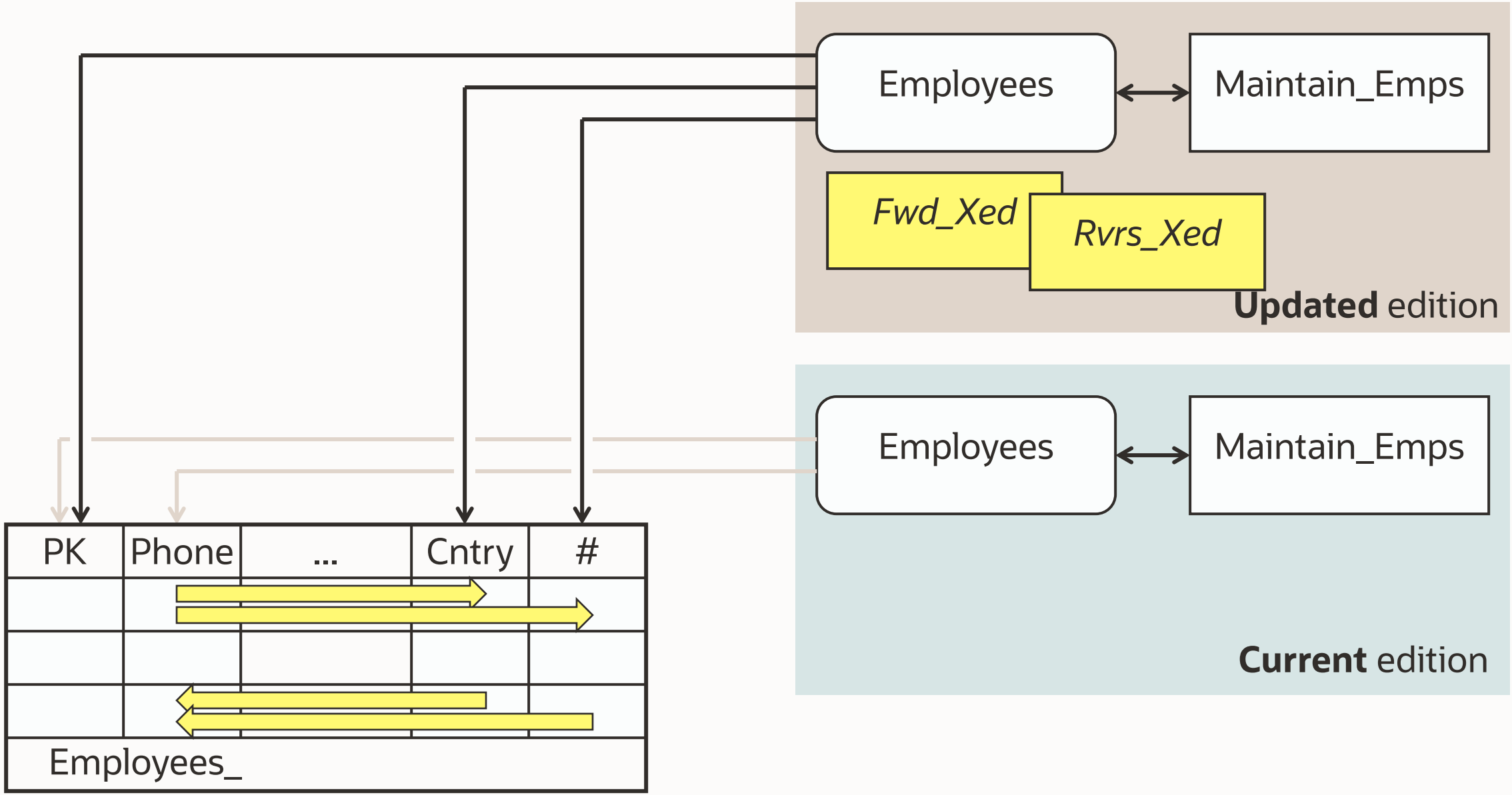


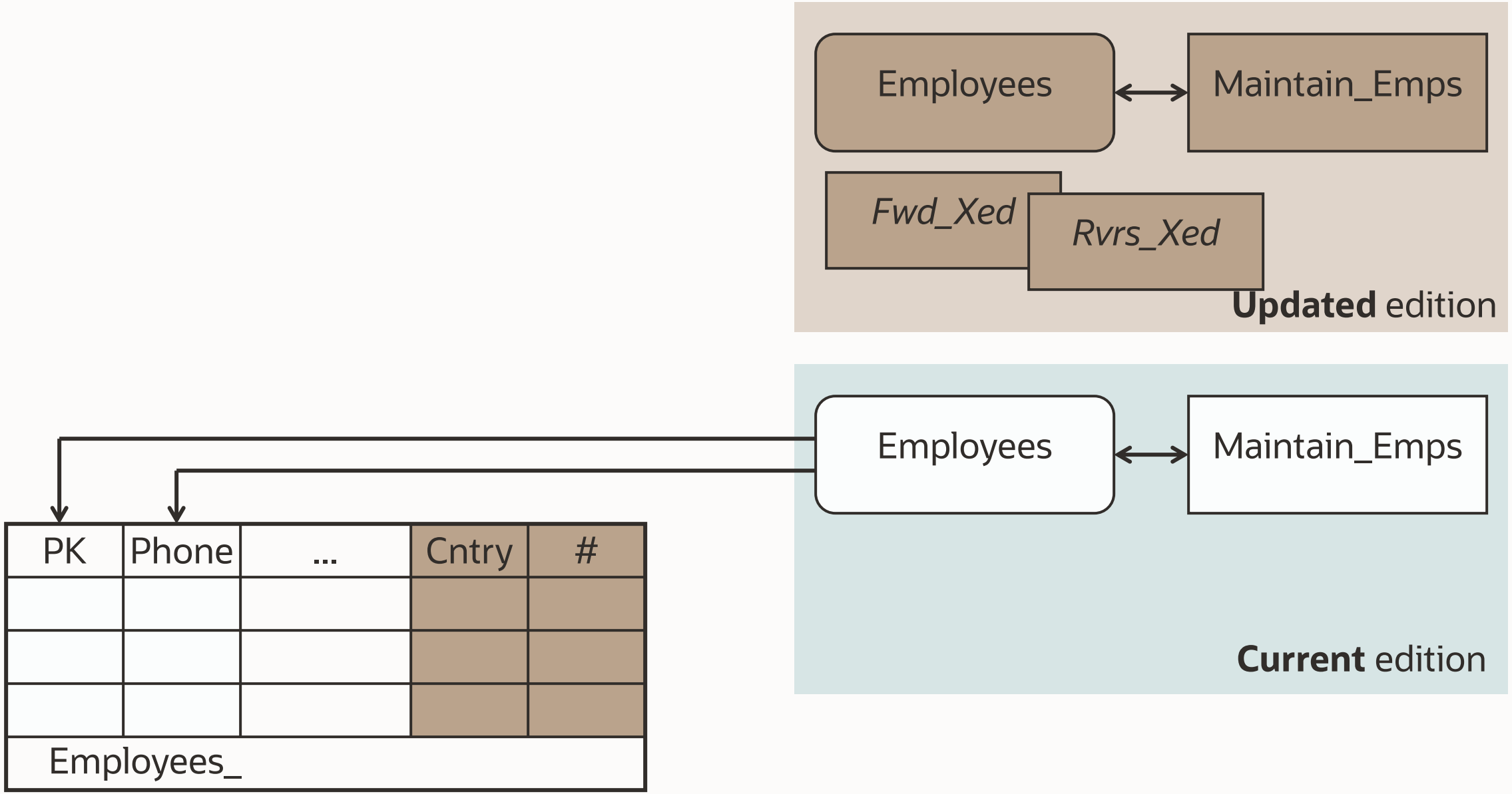


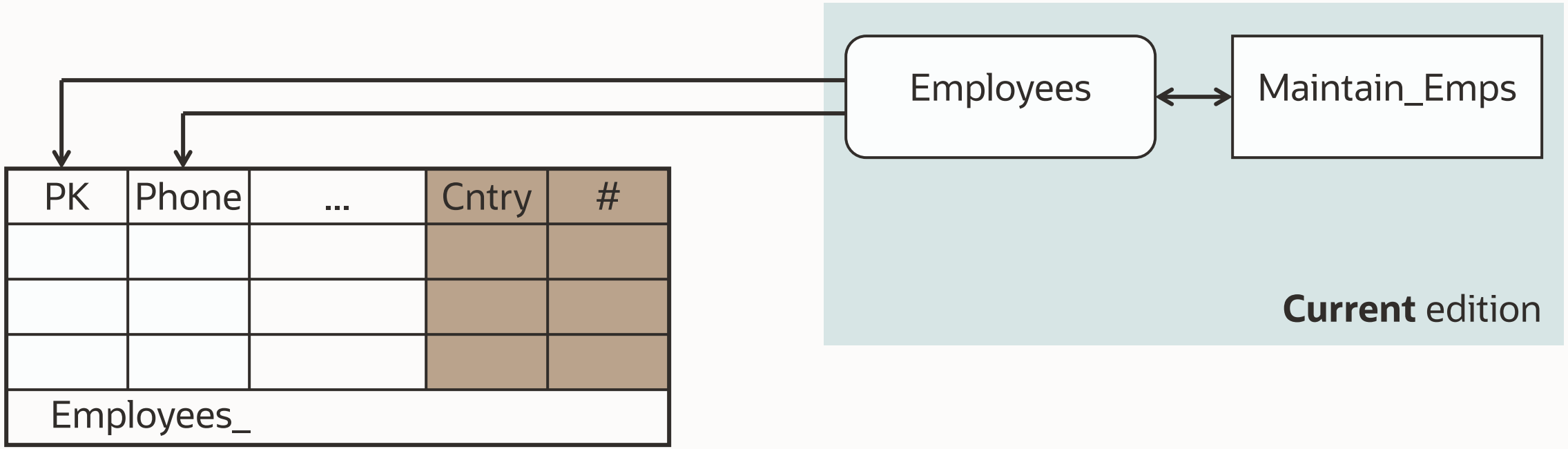
Automated retiring of unused editions as of Oracle Database 12.2

- “Drop edition” is re-implemented to formalize retiring an edition
- You can always drop the root edition, even when it contains actual editioned objects that are inherited by a descendent edition
- It remains in place, but it is marked “unusable” so that you can never use it again
- An editioned object in an unusable edition that is not visible in a usable edition is dropped safely and automatically by a background process
- When an unusable edition contains no actual editioned objects, it is dropped safely and automatically by a background process

Rolling back a failed EBR exercise







Demonstration

Column widening using the replacement column paradigm

Insensitive to pending DML

```
create edition e1 as child of Ora$Base
/

alter session set edition = e1
/
```


Waits on pending DML without blocking new DML

```
alter table t_ add c_Wide varchar2(50)
/
```


Insensitive to pending DML

```
create trigger Fwd
  before update or insert on t_ for each row
  forward crossedition
  disable
begin
  :New.c_Wide := :new.c;
end Fwd;
/

alter trigger Fwd enable
/
```


Deliberately wait on pending DML. Doesn't block new DML.

```
declare
  -- Supplying "null" for the SCN formal to Wait_On_Pending_DML()
  -- asks it to get the most current SCN across all instances.
  SCN number := null;

  -- A null or negative value for Timeout will cause a very long wait.
  Timeout constant integer := null;
begin
  if not Sys.DBMS_Utility.Wait_On_Pending_DML(
    Tables  => 't_',
    Timeout => Timeout,
    SCN     => SCN)
  then
    Raise_Application_Error(-20000,
     'Wait_On_Pending_DML() timed out. '||
     'CET was enabled before SCN: '||SCN);
  end if;
end Wait_On_Pending_DML;
/
```

Apply the crossedition trigger's transform to every row

```
declare
  Cur integer :=
    Sys.DBMS_Sql.Open_Cursor(Security_Level => 2);
  No_Of_Updated_Rows integer not null := -1;
begin
  Sys.DBMS_Sql.Parse(
    c => Cur,
    Language_Flag => Sys.DBMS_Sql.Native,
    Statement => 'update t_ set c = c',
    Apply_Crossedition_Trigger => 'Fwd',
    Fire_Apply_Trigger => true);

  No_Of_Updated_Rows := Sys.DBMS_Sql.Execute(Cur);

  Sys.DBMS_Sql.Close_cursor(Cur);
end;
/
```

Waits on pending DML without blocking new DML

```
alter table t_
modify c_Wide constraint t_c_Wide_NN not null
enable novalidate
/
```


Waits on pending DML without blocking new DML

```
create unique index t_c_Wide_Unq
on t_(c_Wide)
online
/
```


Waits on pending DML without blocking new DML

```
alter table t_
add constraint t_c_Wide_Unq unique (c_Wide)
using index t_c_Wide_Unq
enable novalidate
/
```


Insensitive to pending DML

```
alter table t_
enable validate
constraint t_c_Wide_NN
/

alter table t_
enable validate
constraint t_c_Wide_Unq
/

create or replace editioning view t as
select PK, c_Wide c from t_
/
```


Insensitive to pending DML

```
create trigger Rvrs
  before update or insert on t_ for each row
  reverse crossedition
  disable
begin
  :New.c := :new.c_Wide;

end Rvrs;
/

alter trigger Rvrs enable
/
```

Hot rollover can start now!

Wait until no sessions are using the old edition.

Then retire it.

Insensitive to pending DML

```
alter database default edition = e1
/
```


Waits on pending DML without blocking new DML

```
-- Want to drop the crossedition triggers ASAP
-- to avoid now-unnecessary cost of firing them.
-- Must first drop the constraint that Revrs satisfies.

alter table t_
drop constraint t_c_NN
online
/
```


Insensitive to pending DML

```
alter trigger Rvrs disable
/
drop trigger Rvrs
/

alter trigger Fwd disable
/
drop trigger Fwd
/
```


Waits on pending DML without blocking new DML

```
alter table t_
drop constraint t_c_Unq
online
/
```


Insensitive to pending DML

```
-- Drop all covered objects.
-- Normally this is done entirely mechanically by discovering the
-- covered objects and generating the appropriate "drop" for each.

declare
  Stmt constant varchar2(32767) not null := '
    drop view t';
  Cur integer :=
    Sys.DBMS_Sql.Open_Cursor(Security_level=>2);
begin
  -- Parse for a DDL implies Execute.
  Sys.DBMS_Sql.Parse(
    c             => Cur,
    Language_Flag => Sys.DBMS_Sql.Native,
    Statement     => Stmt,
    Edition       => 'ORA$BASE');
  Sys.DBMS_Sql.Close_Cursor(Cur);
end;
/
```

Waits on pending DML without blocking new DML

```
alter table t_
set unused column c
online
/
```

The EBR exercise is now complete!

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

Next steps

- Read the edition-based redefinition chapter in the Oracle Database Development Guide
- Read EBR whitepaper, published on oracle.com/ebr