

# Rolling Upgrades for Continuous Services

**Antoni Wolski**  
Solid Information Technology

**International Service Availability  
Symposium (ISAS 2004)**

**May 13-14, 2004, Munich, Germany.**

# Intro

**Software upgrades may be  
"traumatic", in an HA system.**

**There are "cures"**

# High availability: dealing with failures and more

- The primary goal of highly available systems is to mask failures
- The target availability has to be maintained

$$Availability \equiv \frac{MTBF_{sys}}{(MTBF_{sys} + MTTR_{sys})} \%$$

- MTBF (mean time between failures)
- MTTR (mean time to repair)

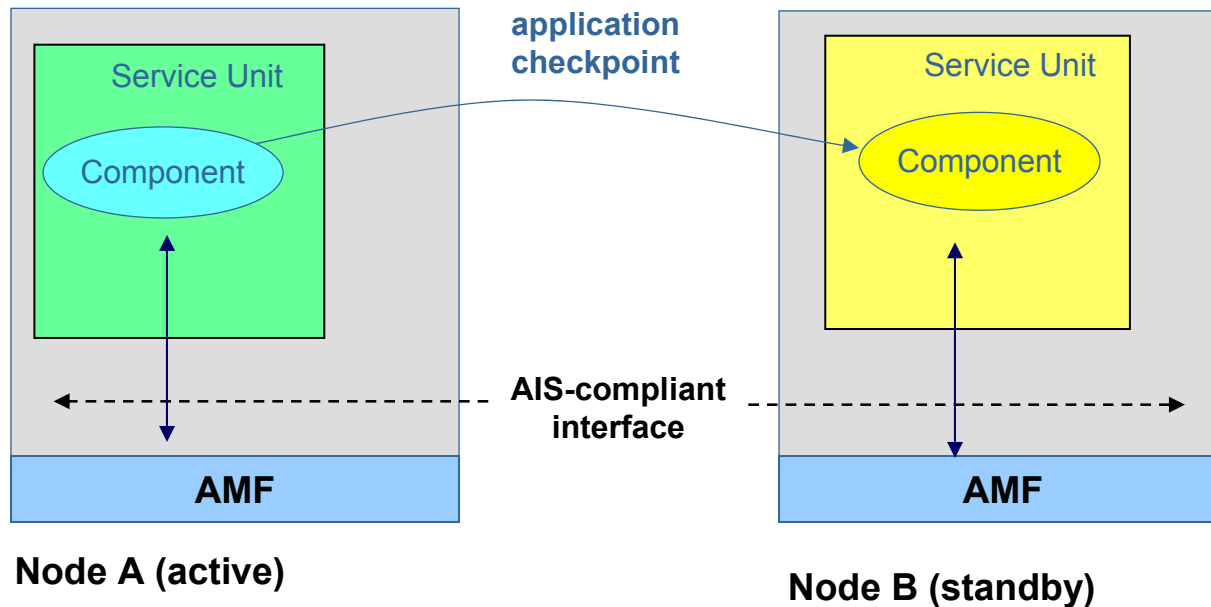
Software upgrade have to be taken care of in accordance with the above formula (MTTR=length of the service breaks resutling from the upgrade).

Rolling upgrade: dynamic upgrade meeting the availability requirement

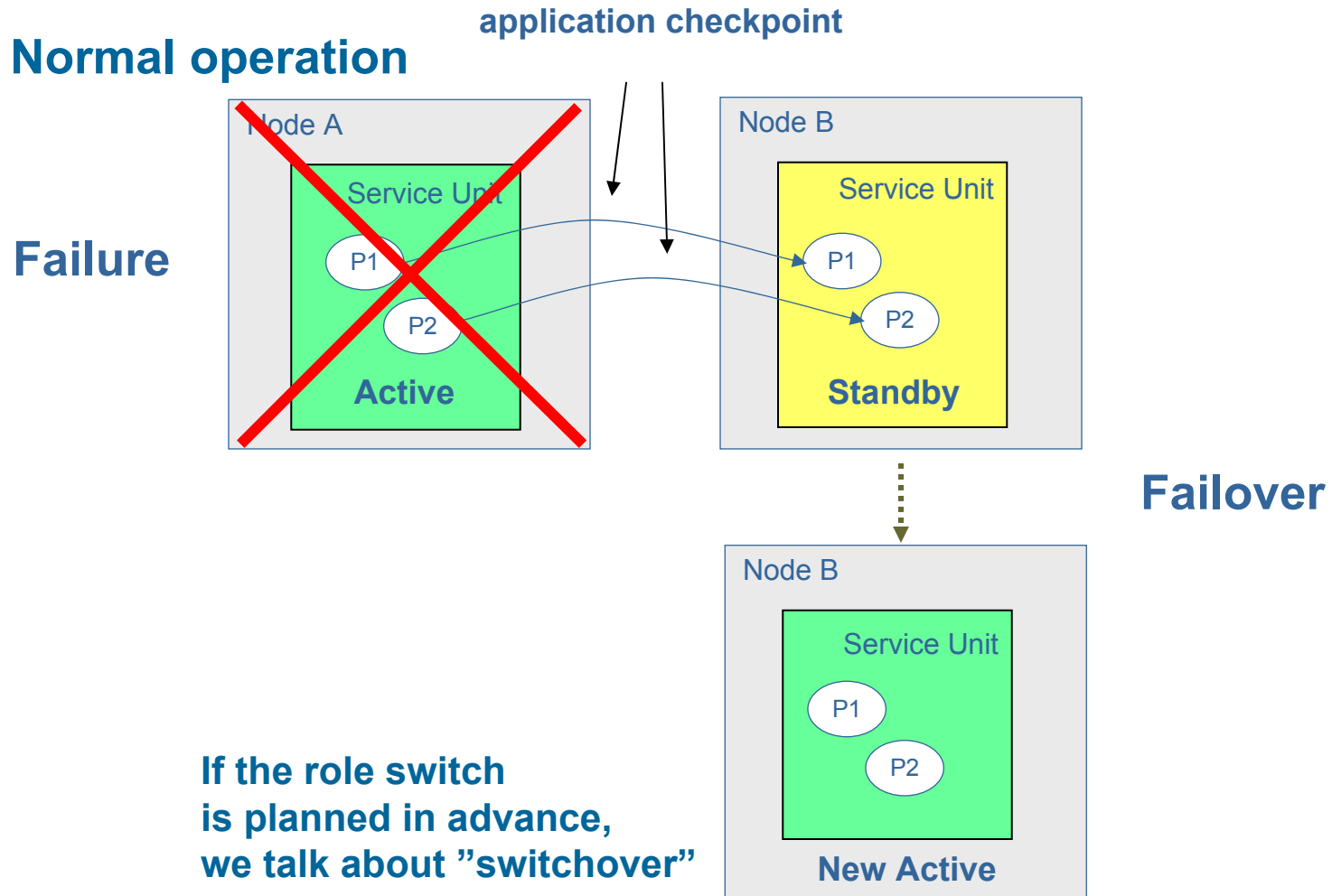
# Basic concepts of HA Framework Architecture

Based on the SA Forum IAS AMF (Availability Management Framework)

Example: the Active/Standby redundancy mode (2N)

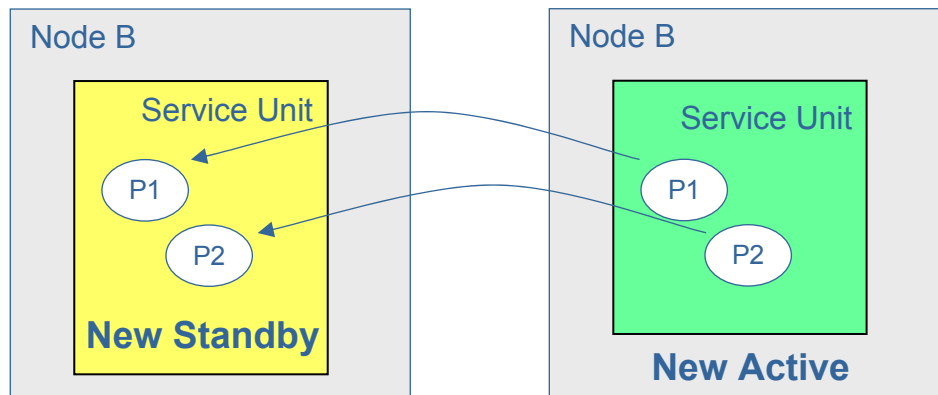


# Failover with active/standby



# Software upgrade using switchover

1. Terminate Standby
2. Upgrade unit
3. Restart Standby
4. Switchover
5. Repeat 1-4



# Challenges of using switchover for SW upgrades

- **A single, atomic switchover operation is beneficial**  
(In SAF AIS, two operations are required at Active (with going to the Quiesced state intermittently) and one at Standby).
- **If the active unit is left running alone for a longer time, the availability level may suffer**  
Use spare nodes or units
- **Only “function-invariant” upgrades (e.g. bug fixes) may be performed easily**
- **“Fully automatic dynamic updating cannot work properly if semantic information is needed to perform any aspect of the updating” (Segal & Frieder 1993)**
- **However, we wish the upgrades would be *unattended* (i.e. automatic) and *safe* (non-disruptive)**

# Related work

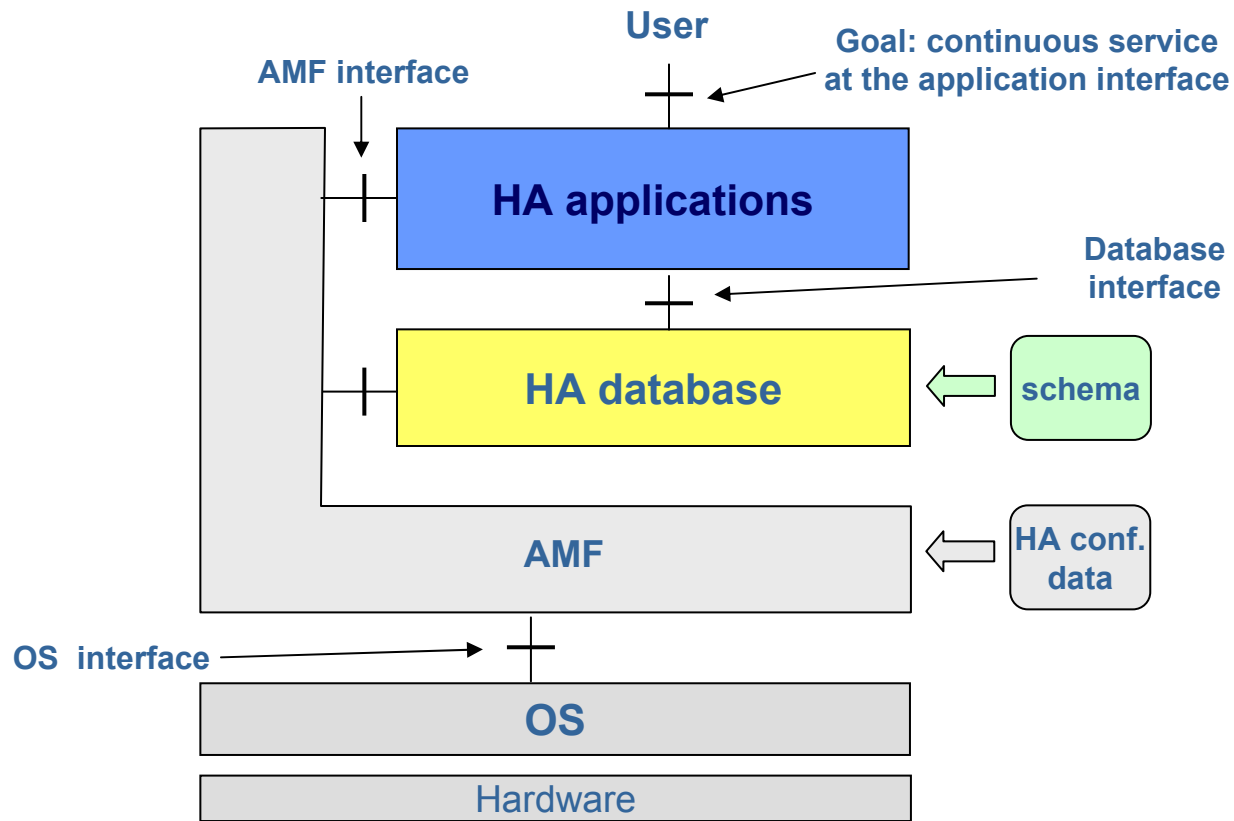
- **Most of the work unrelated to HA systems**
- **Stress on automatic upgrading**
- **Upgrade granularity is low**
  - Argus (Bloom & Day 93) – abstract data types
  - PODUS (Segal & Frieder 93) – procedures
  - Conic (Kramer & Magee 90) – tasks (transactions)
- **Component frameworks help**
  - CORBA (work on upgrade specs started)
  - Java RMI servers (SolarSKI & Meling 02)
  - J2EE EJB (Brada 02)
- **Capturing upgrade metadata**
  - Inter-component dependencies (Kramer & Magee 90)
  - ENT (Exports, Needs, Tags) model (Brada 02) –annotated
  - Why not to use module contracts (Eiffel)?

**Reality: DLLs?**

# The goals of this work

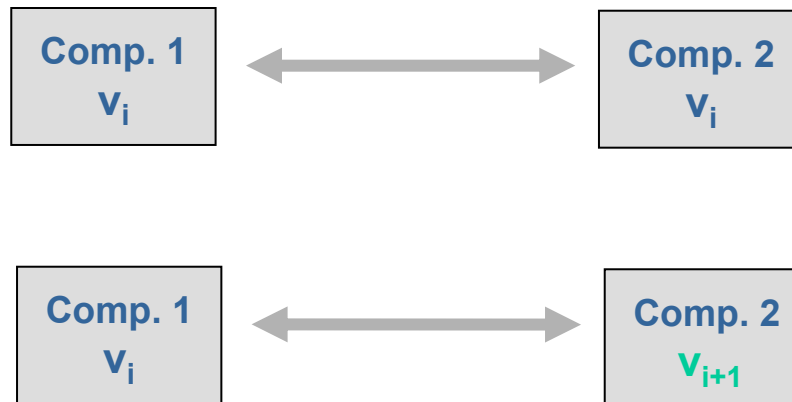
- Find a way to capture the necessary information required to perform dynamic upgrades in HA systems
- Analyze typical upgrade scenarios

# Layers of software in a HA system



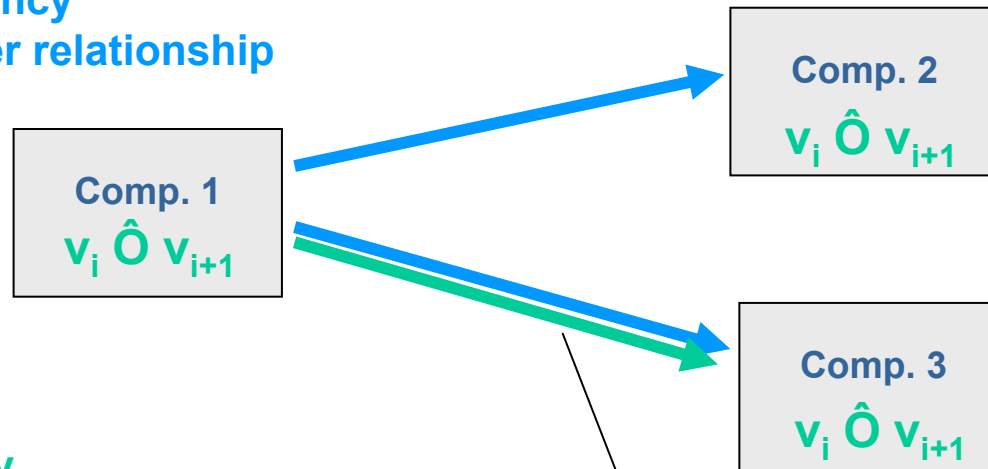
# Assumptions

- **Update granularity: (AIS) component or service unit**
- **Downward compatibility: A new version of the component may be used in the way the previous version was used**
- **Specifically: components can communicate across version boundaries if this is necessary**



# Functional dependency and upgrade dependency

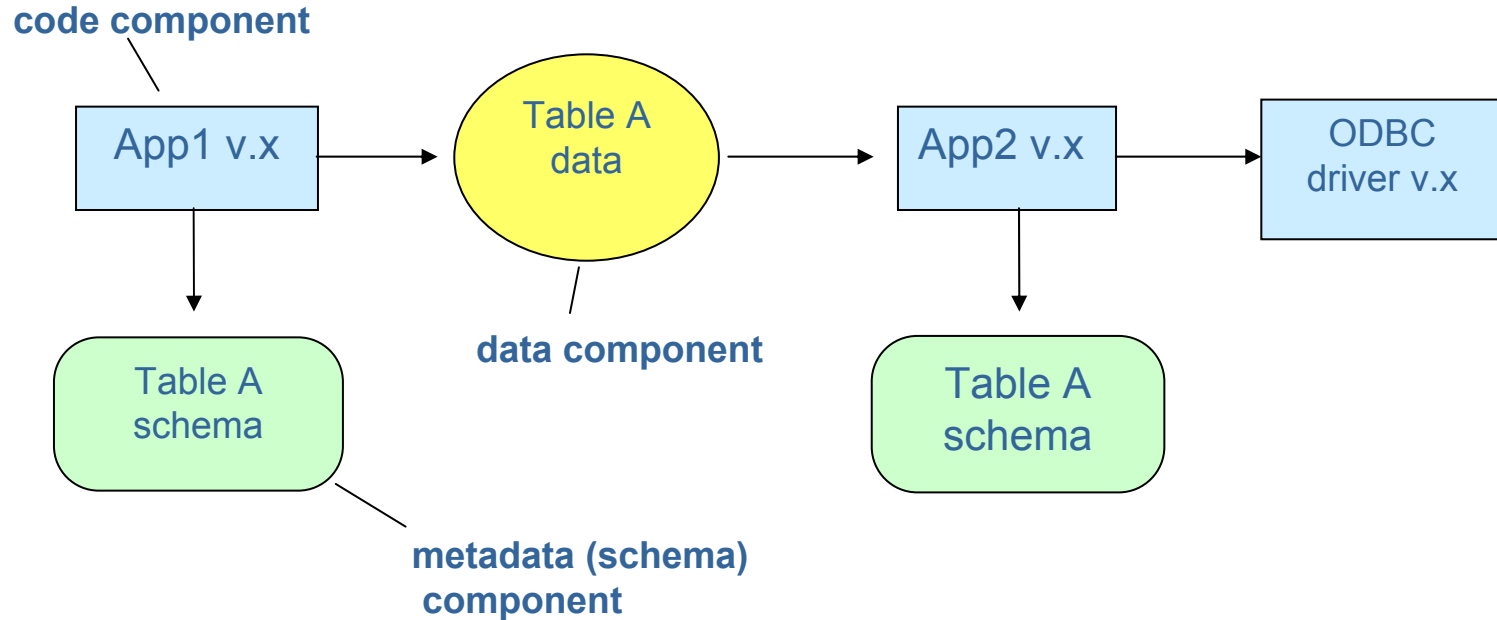
Functional dependency  
= provider/consumer relationship



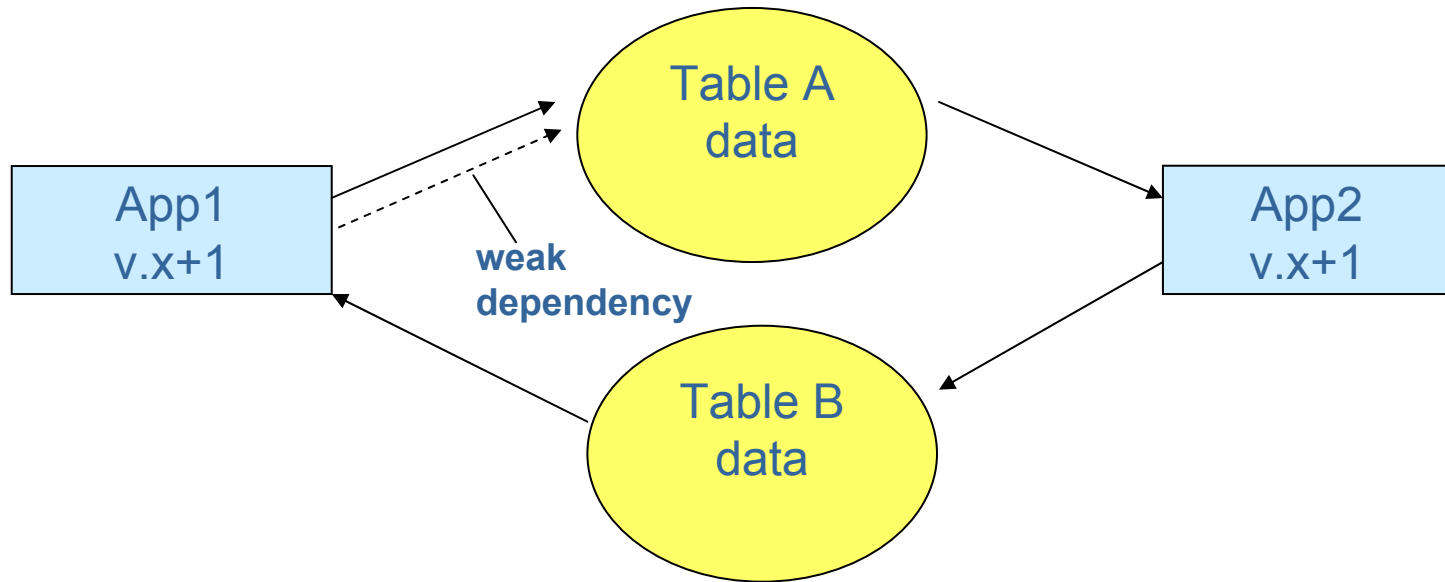
Upgrade dependency  
= dependency on the functionality increment  
between two versions

Comp. 1 upgrade-depends on  
Comp. 3 with respect to upgrade  
 $v_i \hat{O} v_{i+1}$

# Upgrade Food Chain (UFC) diagram



# Cycles in UFC

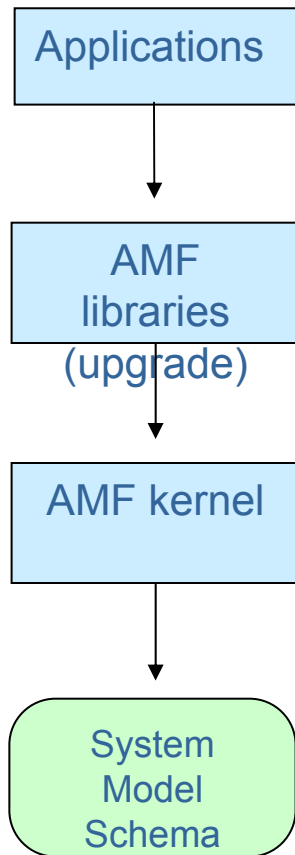


- **Strong (regular) dependency:** the component cannot operate without a service or data provided by the component it is dependent on
- **Weak dependency:** the operation is possible in a degraded way.

# How to acquire information for UFCs?

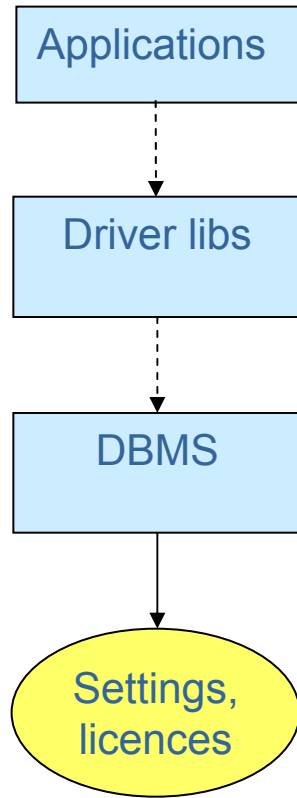
- A. Interview the developers and designers**
- B. Prepare mandatory documentation templates**
- C. Force the information into the code and/or component metadata (like in contract specifications), for automatic processing.**

# Upgrade scenario: HA Framework



1. Perform the schema upgrade in the system model database to support the HA framework upgrade (if applicable)
2. Upgrade the HA framework at the spare node.
3. Re-link other SA-aware subsystems and applications with the upgraded framework libraries, at the spare node.
4. Disconnect current standby node (i.e. the node running standby units) from the active node (resulting in a temporary standalone operation).
5. Transfer the database of the standby node to the upgraded spare node (if applicable).
6. Assign the spare node to be a new standby node. The old standby node becomes a spare node.
7. Perform a controlled switchover.
8. Repeat steps 2-7 starting with the new spare node and new standby node.

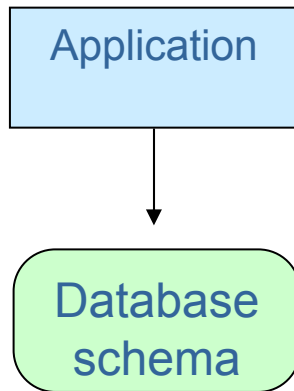
# Upgrade scenario: HA DBMS



1. Stop the standby DBMS server.
2. Upgrade the DBMS software at the standby node.
3. Start the upgraded server in the standby mode, with an optional conversion mode enabled to convert the database to the format supported by the upgraded DBMS (if applicable). Note: if there are applications that are directly linked to the DBMS, they should be re-linked and restarted, too.
4. Reconnect the servers so they resume the active/standby operation. The necessary database catchup (state resynchronization) is performed automatically.
5. Perform the controlled switchover. The active node runs now the new version.
6. Repeat steps 1-5

# Schema upgrade

How to ensure that the schema upgrade can be done safely?



## Monotonic schema upgrade

- i. None of the first-class objects is removed or renamed.
- ii. None of the existing columns is removed or renamed
- iii. None of the existing integrity constraints is changed
- iv. None of the existing active objects (stored procedures, triggers and events) is redefined

## Schema-Upgrade-Safe Application Development

- i. It does not take advantage of any implicit column ordering.
- ii. It does not take advantage of table dimensionality (number of columns).
- iii. Its error processing (especially of DELETE statements) anticipate possible referential enhancements.

# Upgrade scenario: Schema

---

1. Apply the schema upgrade, dynamically, to the active database. The schema changes are automatically propagated to the standby database.

# Conclusions

**The prerequisites for a successful rolling upgrade at any level of the system are:**

- Finding out upgrade dependencies and capturing them with, for example, Upgrade Food Chain (UFC) diagrams.
- Organizing the upgrades into a partial order of upgrade steps by introducing weak dependencies and reorganizing the elements of the UFC diagram.
- Programming the upgrades in a way that allows for existence of weak dependencies and satisfies the rules of schema-upgrade-safe application development.
- Assuring monotonic schema upgrades.
- Using an HA DBMS that supports dynamic and uninterruptible schema update.
- Using an HA DBMS capable of rolling upgrade of the DBMS software.
- Using a HA framework software capable of doing a rolling upgrade of its own.

**Can be done!**

**“5 nines” costs at least 100% more than “4 nines”.  
For this price, you expect to get rolling upgradeability too.**

---