Gdevops

全球敏捷运维峰会

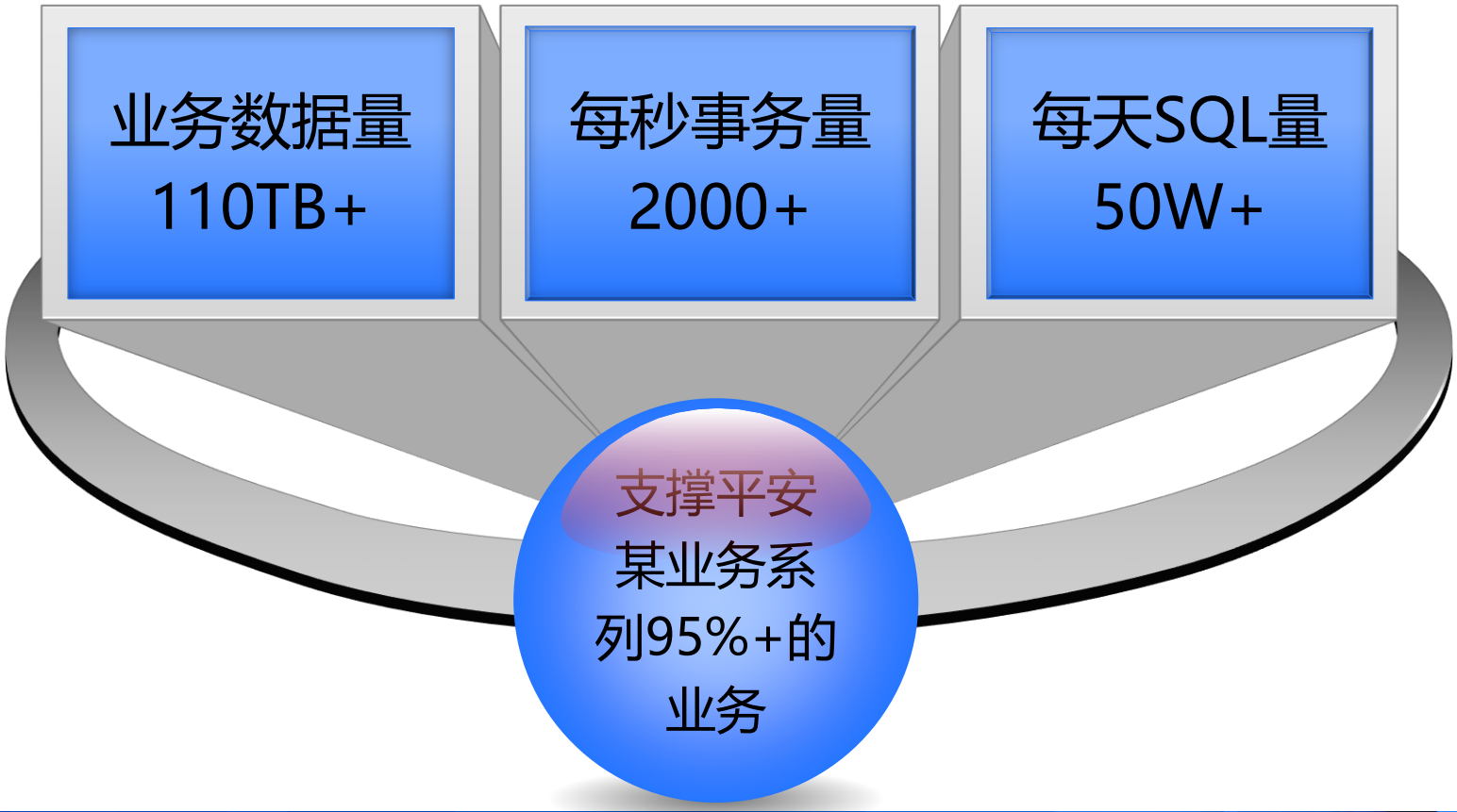
平安某核心DB升级记

演讲人：汪洋

- 我们的困境和挑战
- 选择比努力更重要
- 十月磨一剑
- 最终的决战时刻

我们的困境和挑战——业务系统特点

全球最大的ORACLE 9i OLTP数据库



我们的困境和挑战——当前面临问题

获得Oracle产品的Premier Support 支持

Oracle 9i 版本的延展服务已于2010-07结束，此DB已知有**7**个会导致DB宕机的Bug无法Fix；通过升级，可以获得产品的Premier Support 支持策略，100%解决目前已知的产品问题。

解决硬件扩展的瓶颈

此核心数据库所在的小型机即将过保，新购机型已经不再支持当前的DB版本；必须通过升级版本，才能在硬件上提供更好的支撑能力，支持业务的健康发展。

缓解DB性能不稳定的风险

2013~2014年此DB出现**3**次紧急事件，均与9i版本的CBC Latch相关；2009~2014年此DB出现12次重大事件，其中**5**次与执行计划突变相关；通过升级，可以在92.65%程度上缓解上述风险。

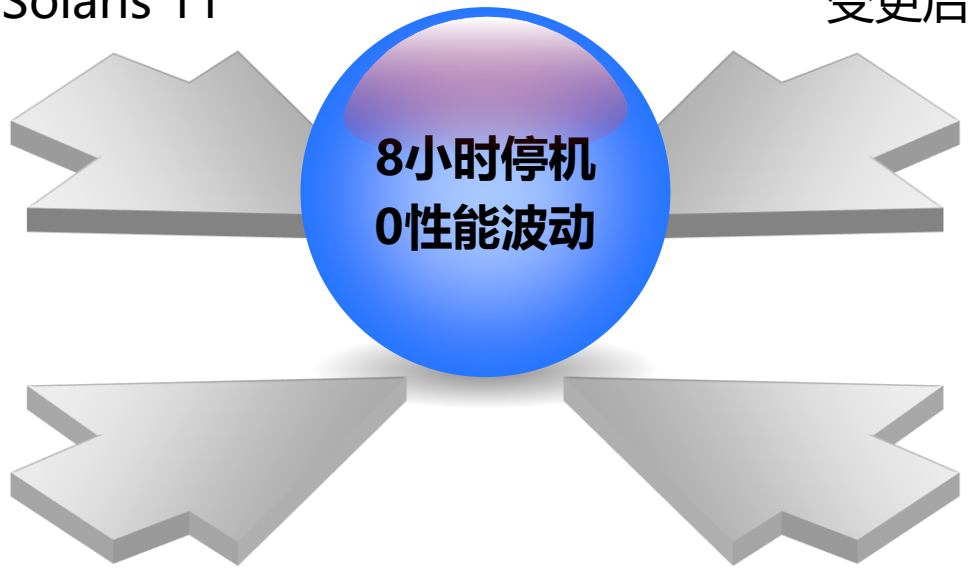
提升技术人力的价值投入

2013~2014针对此DB进行性能影响分析**457**单，投入不低于**65.7**人月的人力成本；通过升级，可缩短版本开发的路径，提高效率。

我们的困境和挑战——系统变更要求

OS 版本
变更前　Solaris 10
变更后　Solaris 11

DB 版本
变更前　9.2.0.8
变更后　11.2.0.4

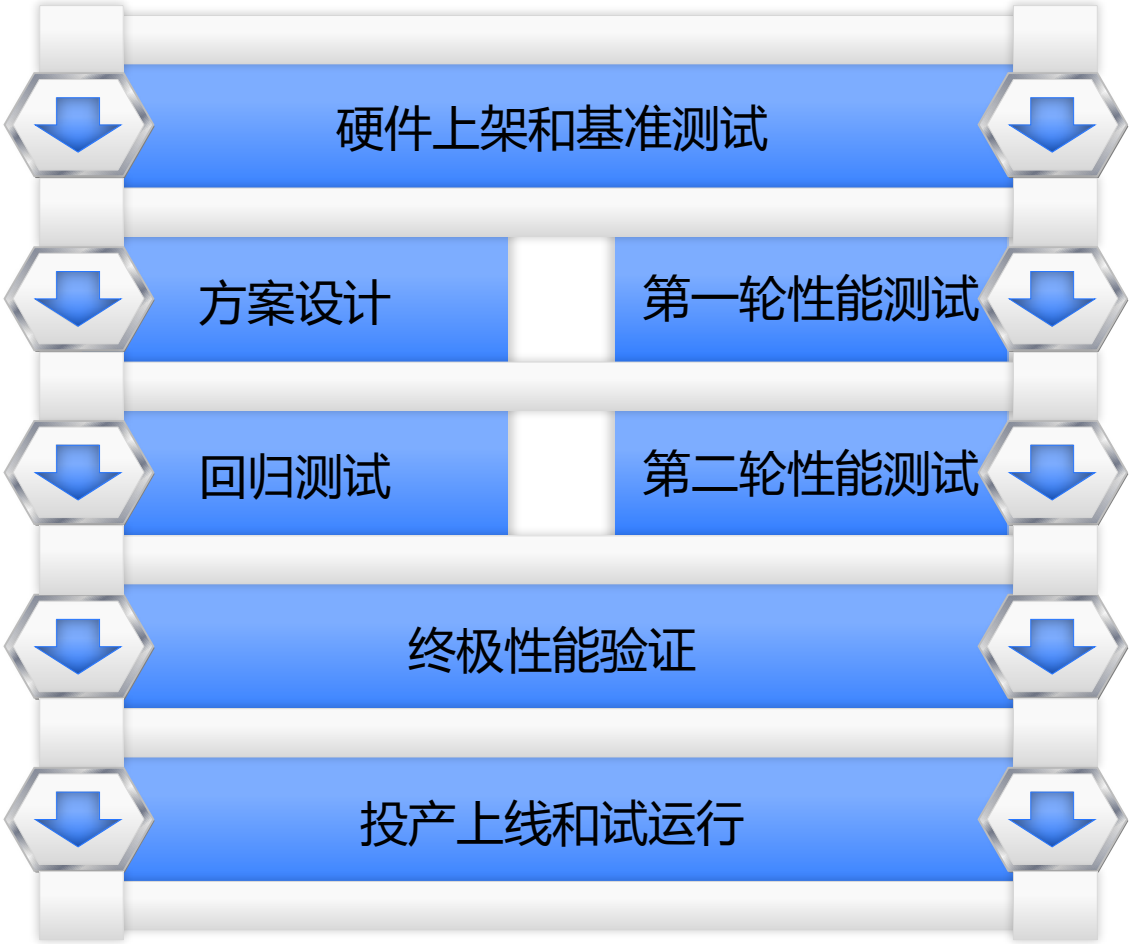


主机硬件
变更前　M9000
变更后　T5-4

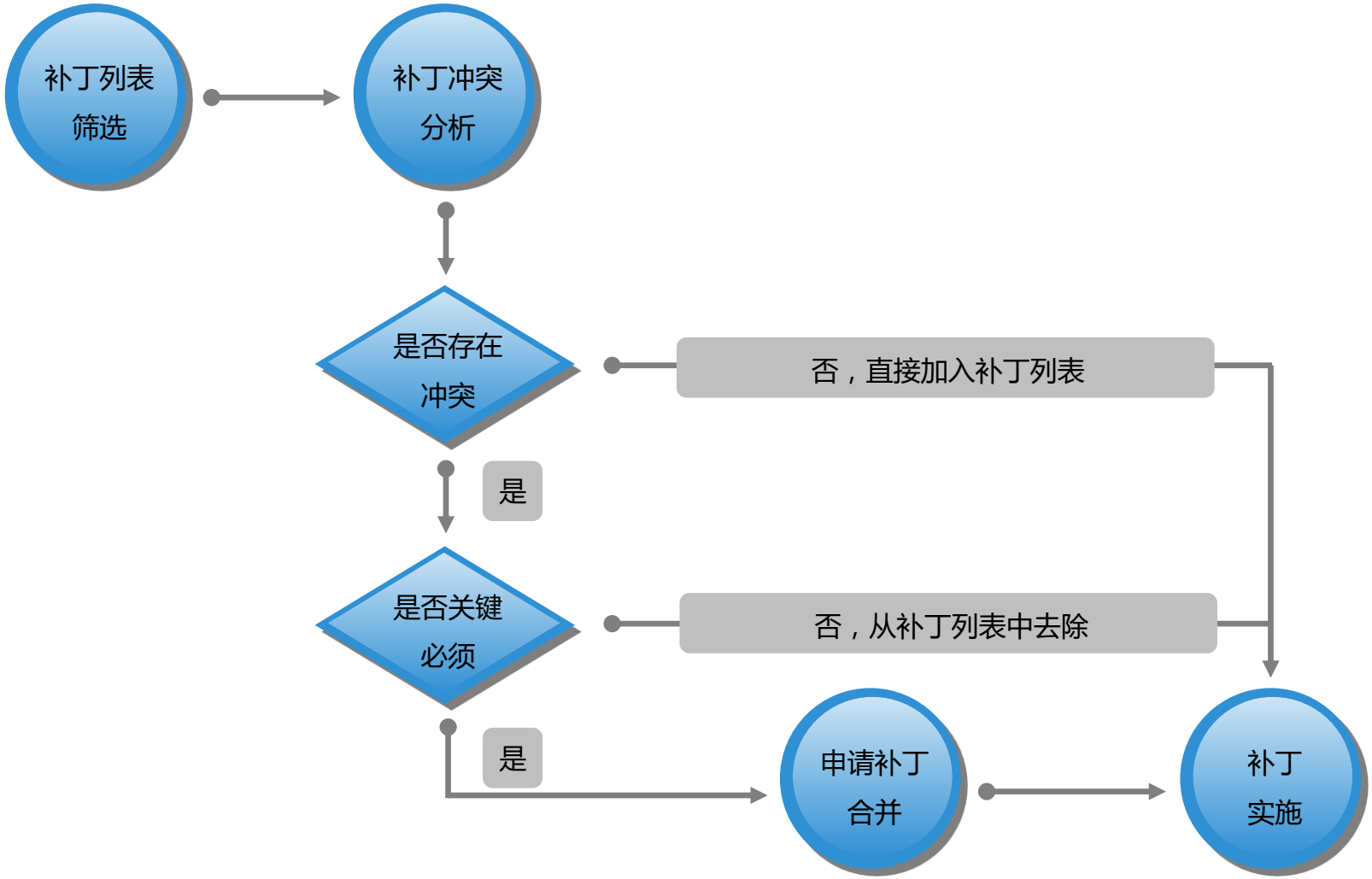
存储硬件
变更前　高端SAN
变更后　闪存

- 我们的困境和挑战
- 选择比努力更重要
- 十月磨一剑
- 最终的决战时刻

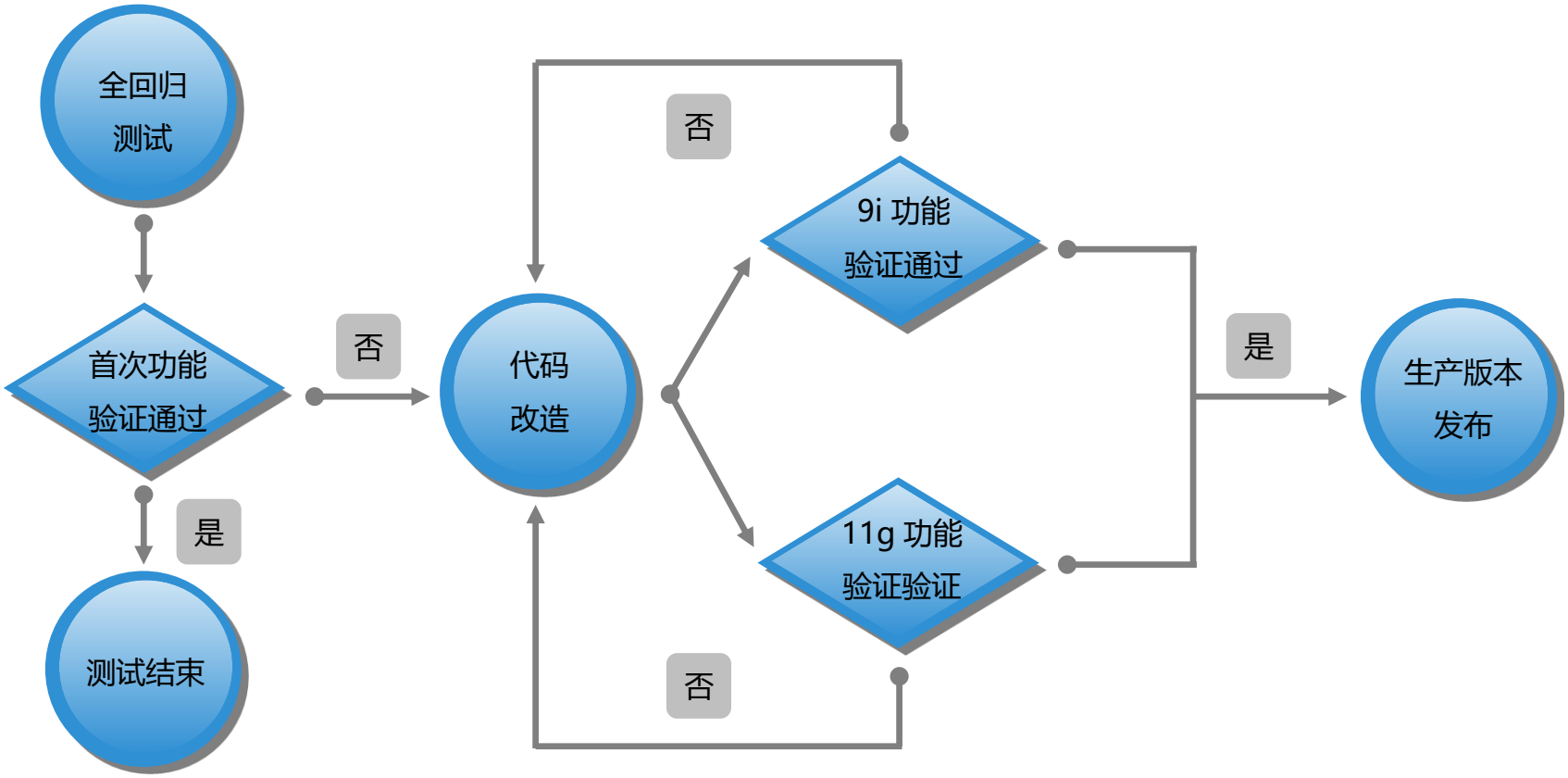
选择比努力更重要——实施过程设计



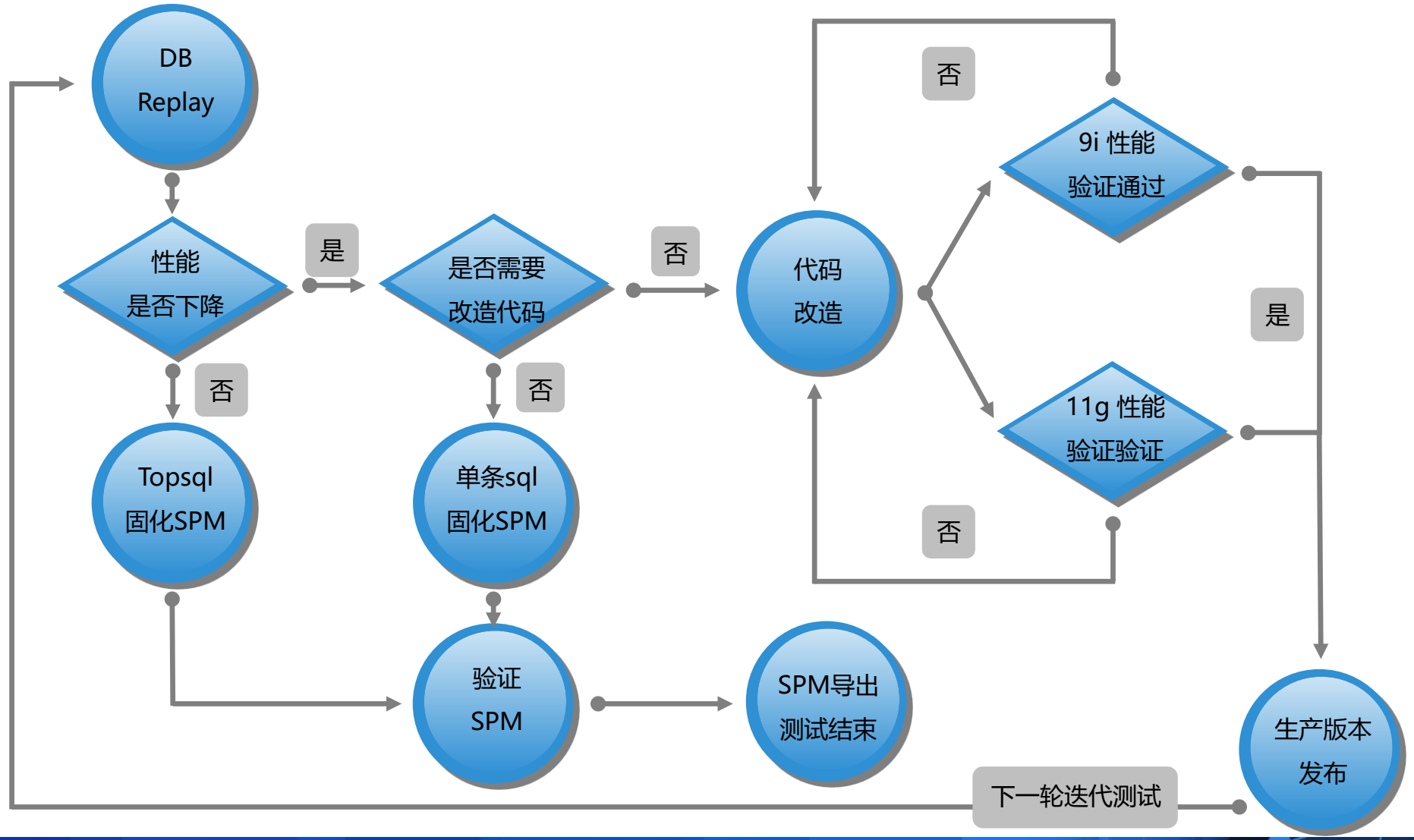
选择比努力更重要——软件补丁方案设计



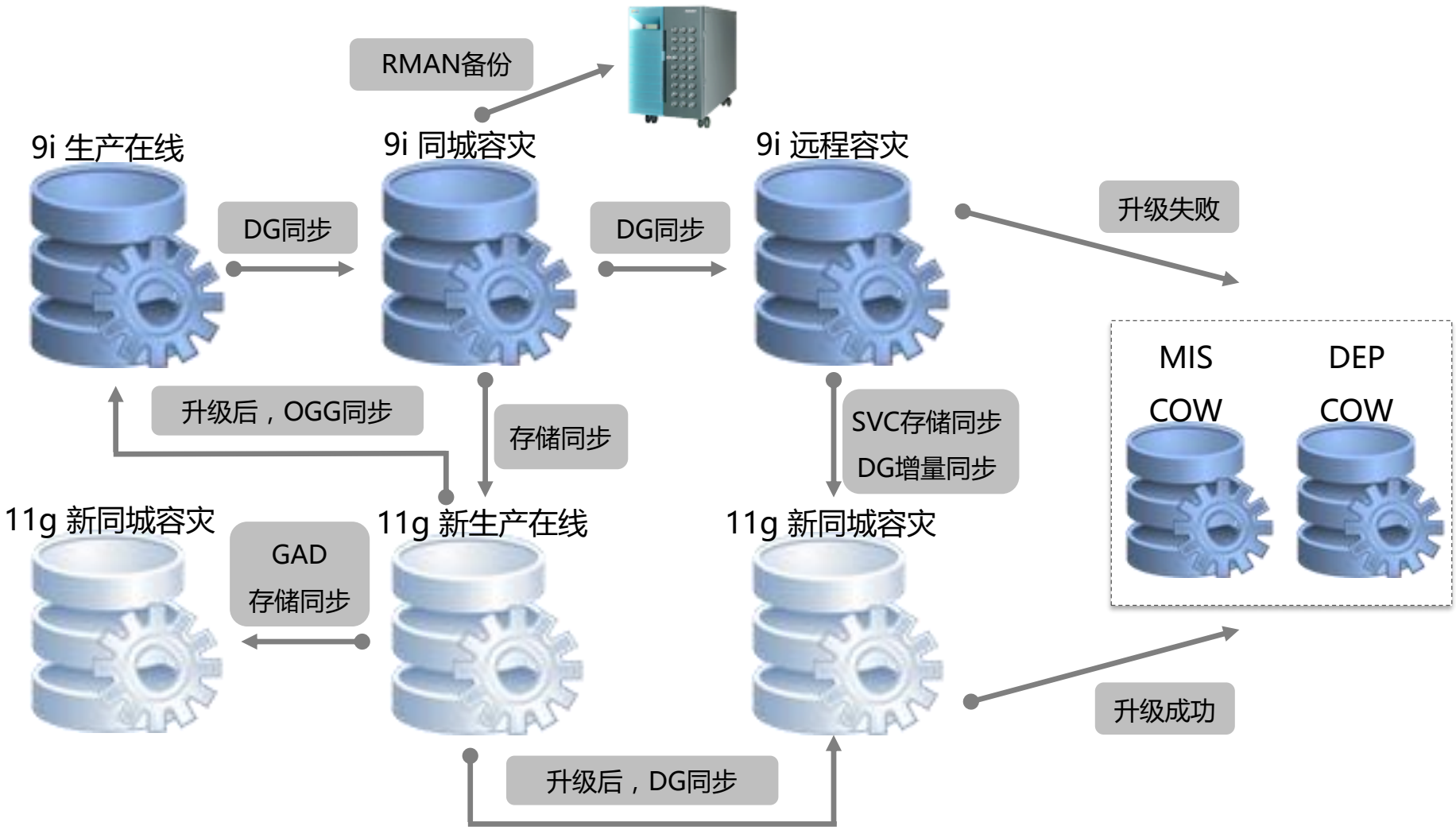
选择比努力更重要——功能测试方案设计



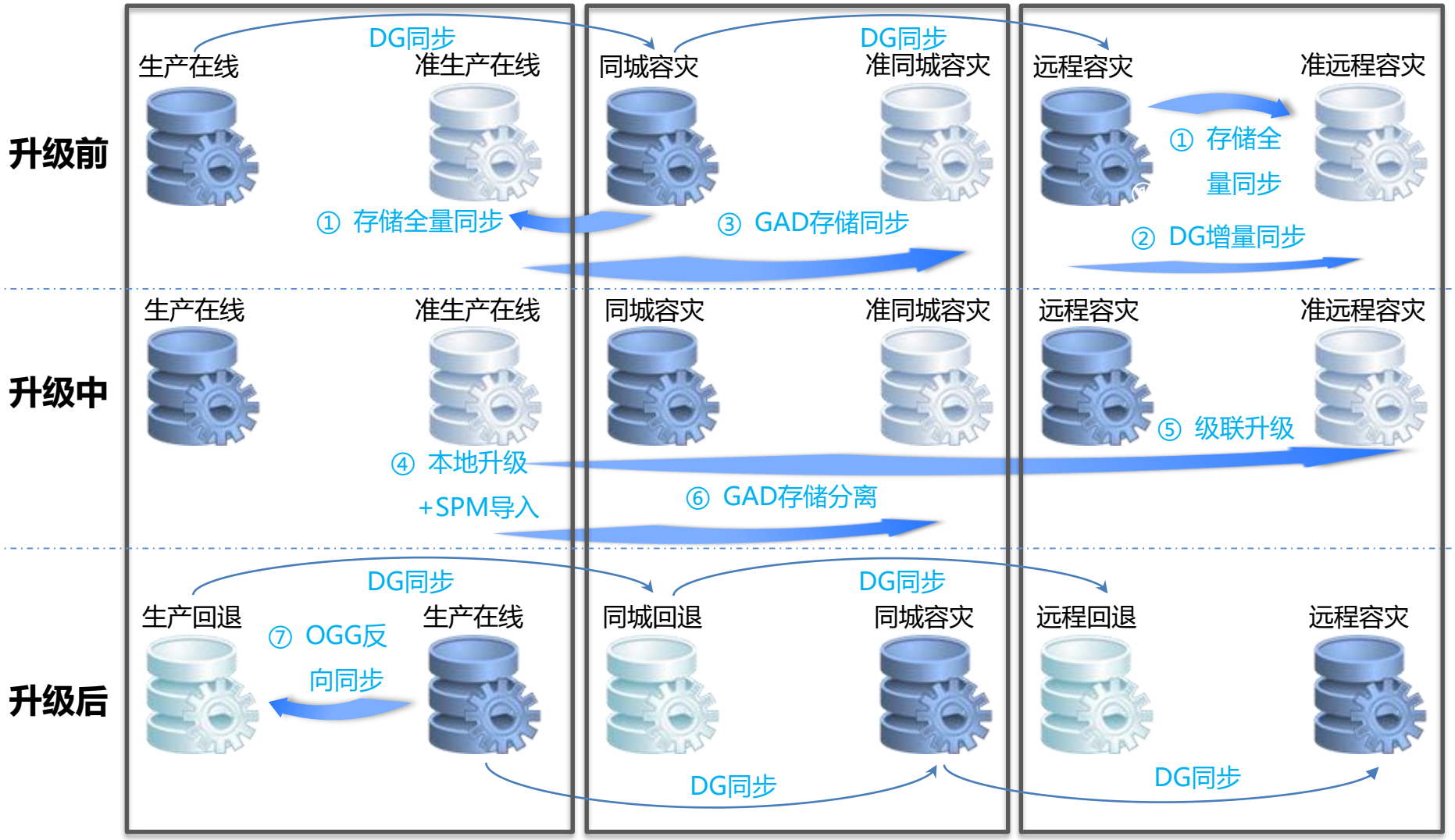
选择比努力更重要——性能测试方案设计



选择比努力更重要——投产上线架构设计



选择比努力更重要——投产实现方案设计



- 我们的困境和挑战
- 选择比努力更重要
- 十月磨一剑
- 最终的决战时刻

十月磨一剑——性能分析迭代

2、构造分析基线

- 保持生产的优化器参数和统计信息
- 通过DB Replay回放生产压力
- 生成Topsql的SPM
- 保留v$sql性能数据

3、分析性能指标

- 启用升级后的优化器参数和最新统计信息
- 导入基线上生成的SPM
- 通过DB Replay回放生产压力
- 比对Topsql和异常等待事件
- 创建优化后的SPM集合

1、抓取生产负载

- 在业务高峰抓取Replay Trace
- 抓取过程中监控生产性能

4、优化性能问题

- 未使用绑定变量
- Date类型上的隐式转换
- 复杂视图查询执的执行计划改变
- 下发版本进行代码改造

十月磨一剑——绑定变量改造

```
IF CONDITION = 1
SELECT col1, col2 FROM table t1 WHERE t1.col3 IN ('')
......
ELSE CONDITION = n
SELECT col1, col2 FROM table t1 WHERE t1.col3 IN ('','',.....,'')
END IF;
```

```
v_sql  : = 'SELECT null FROM dual';
FOR i IN 1..n LOOP
v_sql_part := 'SELECT col1, col2
                FROM table t1 WHERE t1.col3 = :B'|| i;
v_sql := v_sql || ' union all ' || v_sql_part;
END LOOP;
  CASE
    WHEN i = 1 THEN OPEN v_cur for v_sql using c(1);
                ......
    WHEN i = n THEN OPEN v_cur for v_sql using c(1),.....,c(n);
  END;
```

现象

在某个系统中，发现大量类似的sql语句，没有使用绑定变量，无法进行SPM固化

原因

此业务的判断逻辑非常复杂，大量的Package中采用分支条件进行判断；不同分支的IN条件中，带入不同个数的变量值，无法简单地改造成绑定变量

优化方案

- 在v$sql中匹配”EXACT_MATCHING_SIGNATURE”和”FORCE_MATCHING_SIGNATURE” ，按照count(FORCE_MATCHING_SIGNATURE) 进行改造优先级排序
- 采用动态sql的方式进行拼接，合并不同分支的sql语句，将原来5000+的sql量归并到1500-

十月磨一剑——隐式转换改造

```
SELECT COUNT(*) FROM TABLE T1 WHERE
<dynamic>
  <isNotEmpty prepend="and" property="Date1">
   <![CDATA[T1.DATE >= #Date1#]]>
  </isNotEmpty>
  <isNotEmpty prepend="and" property="Date2">
   <![CDATA[T1.DATE <= #Date2# ]]>
  </isNotEmpty>
</dynamic>
```

```
SELECT COUNT(*) FROM TABLE T1 WHERE
<dynamic>
  <isNotEmpty prepend="and" property="Date1">
   <![CDATA[T1.DATE >= cast(#Date1# as date) ]]>
  </isNotEmpty>
  <isNotEmpty prepend="and" property="Date2">
   <![CDATA[T1.DATE <= cast(#Date2# as date) ]]>
  </isNotEmpty>
</dynamic>
```

现象

在某个系统中，发现大量的全表扫描，生成的执行计划无法使用DATE列上的时间索引

原因

ibatis处理入参时，java.util.Date会被自动转换为java.sql.Timestamp；而数据库中的字段类型为date，发生隐式转换，从而使用不了索引

优化方案

- 在v$sql_plan中查询”INTERNAL_”或” TO_NUMBER”关键字，按照 count(FORCE_MATCHING_SIGNATURE)进行改造优先级排序
- 在应用代码中增加cast函数降低数据精度
- 长期方案，针对date类型，应用必须传入string格式，并在XML中使用to_date()函数进行转换

十月磨一剑——复杂视图改造

```
-----------------------------------------------
| Id  | Operation                     | Name  |
-----------------------------------------------
|   3 |    MERGE JOIN                 |       |
|   4 |     TABLE ACCESS BY INDEX ROWID| T4    |
|   5 |      INDEX FULL SCAN          | PK T4 |
|*  6 |     SORT JOIN                 |       |
|*  7 |      TABLE ACCESS FULL        | T1    |
|   8 |    TABLE ACCESS FULL          | T2    |
|   9 |   TABLE ACCESS FULL           | T3    |
-----------------------------------------------
```

```
-----------------------------------------------
| Id  | Operation                     | Name  |
-----------------------------------------------
|   4 |     MERGE JOIN                |       |
|   5 |      TABLE ACCESS BY INDEX ROWID| T4    |
|*  6 |       INDEX FULL SCAN         | PK_T4 |
|*  7 |      SORT JOIN                |       |
|*  8 |       TABLE ACCESS FULL       | T1    |
|   9 |     TABLE ACCESS BY INDEX ROWID | T2    |
|* 10 |      INDEX UNIQUE SCAN        | PK_T2 |
|* 11 |    INDEX RANGE SCAN           | PK_T3 |
|  12 |   TABLE ACCESS BY INDEX ROWID | T3    |
-----------------------------------------------
```

现象

在某个系统中，发现核心大表上出现很多全表扫描，sql性能大幅下降

原因

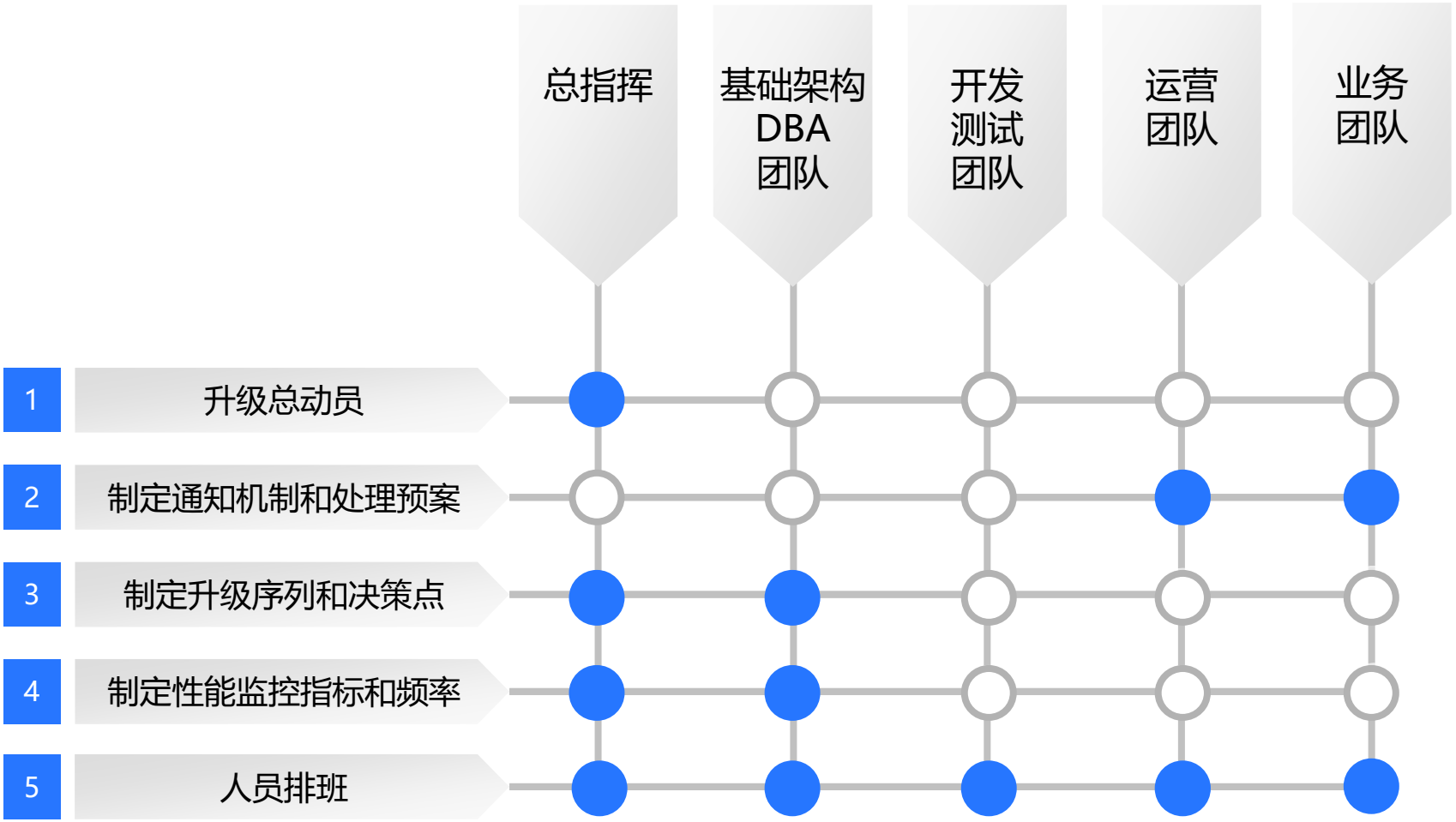
这些sql的构造非常类似，均是由T1、T2、T3三张大表创建的VIEW，再和其他的表进行关联；在9i中filter条件可以先在VIEW上过滤再和其他表关联，但11g必须先实例化VIEW，再进行filter条件过滤

优化方案

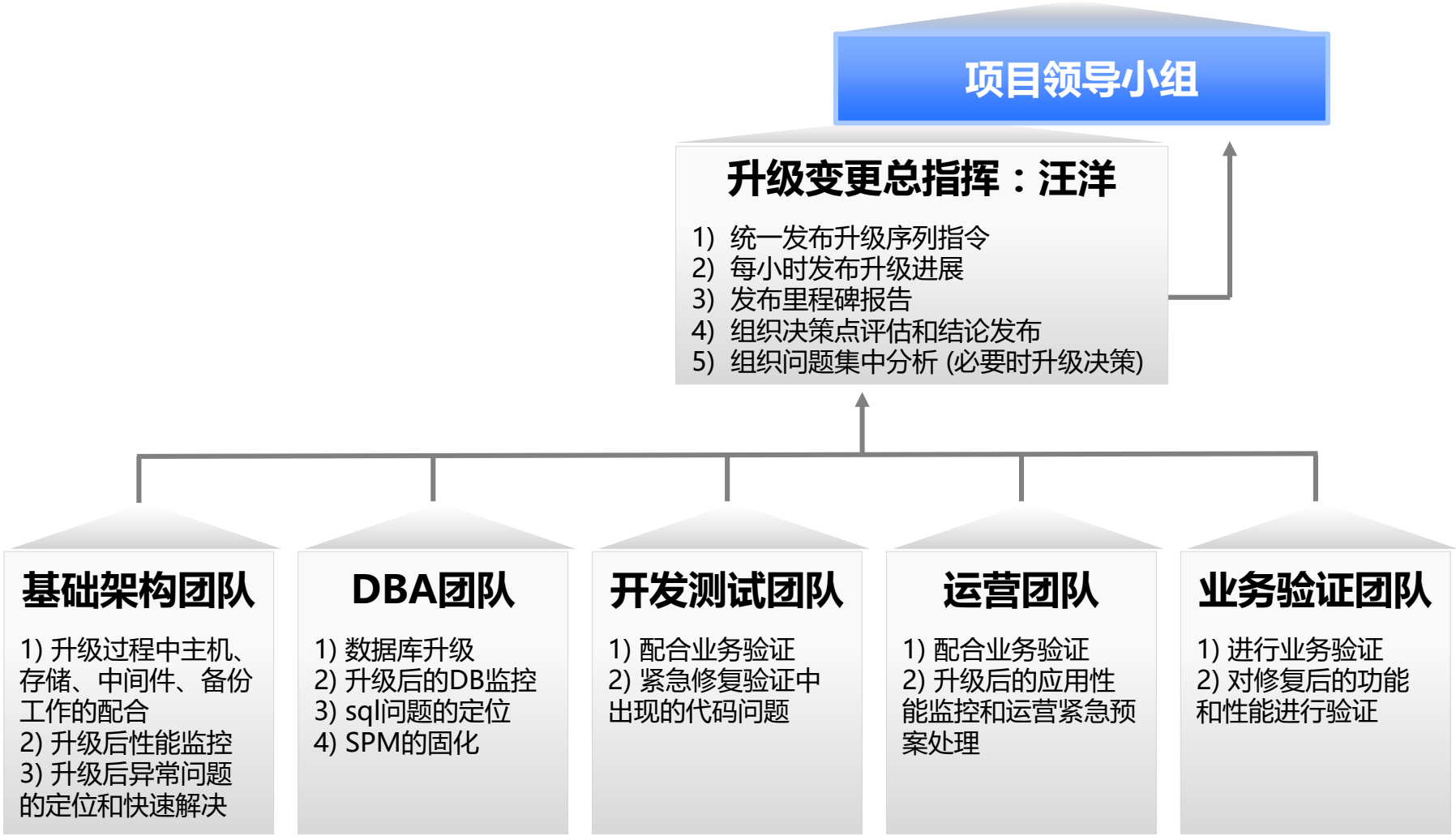
- 在v$sql中查询VIEW相关的代码
- 将VIEW拆开，分别和VIEW之外的表进行关联查询，最后再合并结果集

- 我们的困境和挑战
- 选择比努力更重要
- 十月磨一剑
- 最终的决战时刻

最终的决战时刻——投产组织工作



最终的决战时刻——投产人员架构



最终的决战时刻——投产前小插曲

突变	倒计时	紧急救援
存储全量同步准生产环境，影响在线系统性能急剧下降	72小时	● 业务核心流程应用线程堵塞，数据库中出现大量IO相关等待 ● 停止存储全量同步后系统恢复，但无法按计划准备准生产数据 ● 紧急调整升级方案，改为从同城容灾进行存储全量数据同步 ● 紧急测试，能否通过复制主库control file和redo file的方案进行升级，以及确保同城容灾和远程容灾同步正常，测试成功 ● 调整升级序列
物化视图刷新失败，事务回滚预计98小时	32小时	● 754281个block需要回滚，预估大约98个小时，主要等待事件是db file sequenctial read ● 启动并发回滚，增加回滚条目，发现没有正向作用 ● 将物化视图所在的文件卷替换成Flash闪存，加快db file sequential read的效率，发现回滚速率提升了5倍 ● 启用存储Cache预热功能，将物化视图和物化视图log缓存到内存中，发现回滚速率又提升了3倍 ● 在变更当天中午，完成事务回滚，实际耗时6.5小时

最终的决战时刻——投产运行分析

高峰期业务吞吐量
平均提升23.26%

系统前端响应时间
平均提升30%以上

投产后
未出现性能毛刺
系统性能显著提升

批量作业效率
提升5~25倍不等

单个sql效率
提升5~9000倍不等

Gdevops

全球敏捷运维峰会

THANK YOU！