菜鸟供应链实时数据技术架构的演进

缘桥

菜鸟-数据&规划部-数据技术专家

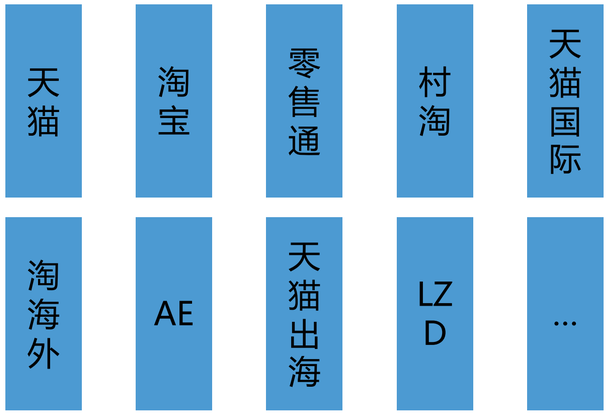
FLINK FORWARD # ASIA
实时即未来 # Real-time Is The Future

FLINK FORWARD

以前的实时数据技术架构

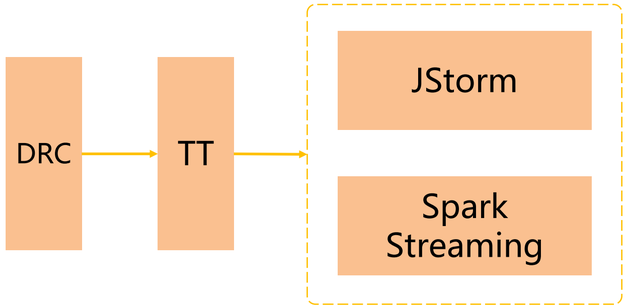
Real-time data warehouse and technology architecture for 2016

数据模型
Data Model



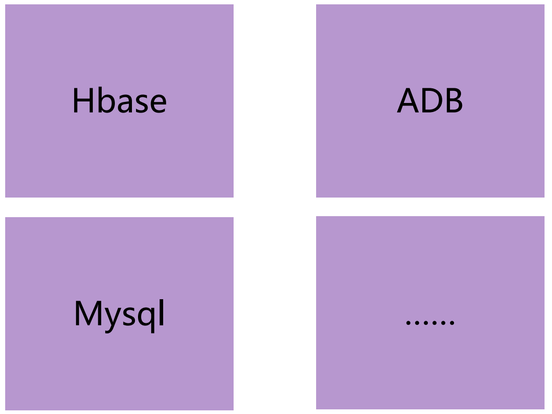
- 业务线内部模型层次混乱，数据使用成本特别高
  - The hierarchy of data model is disordered, and the cost of using data is very high.
- 需求驱动的烟囱式开发，完全没有复用的可能性，计算成本居高不下
  - The chimney development model leads to excessive cost.

实时计算
Computing Engine



- 有些复杂的功能，实现起来不够简单，开发成本较高
  - Some requirements are complex and cost-effective.
- 很难同时兼顾功能、性能、稳定性以及快速的故障恢复能力
  - It is difficult to simultaneously balance function, performance, stability.

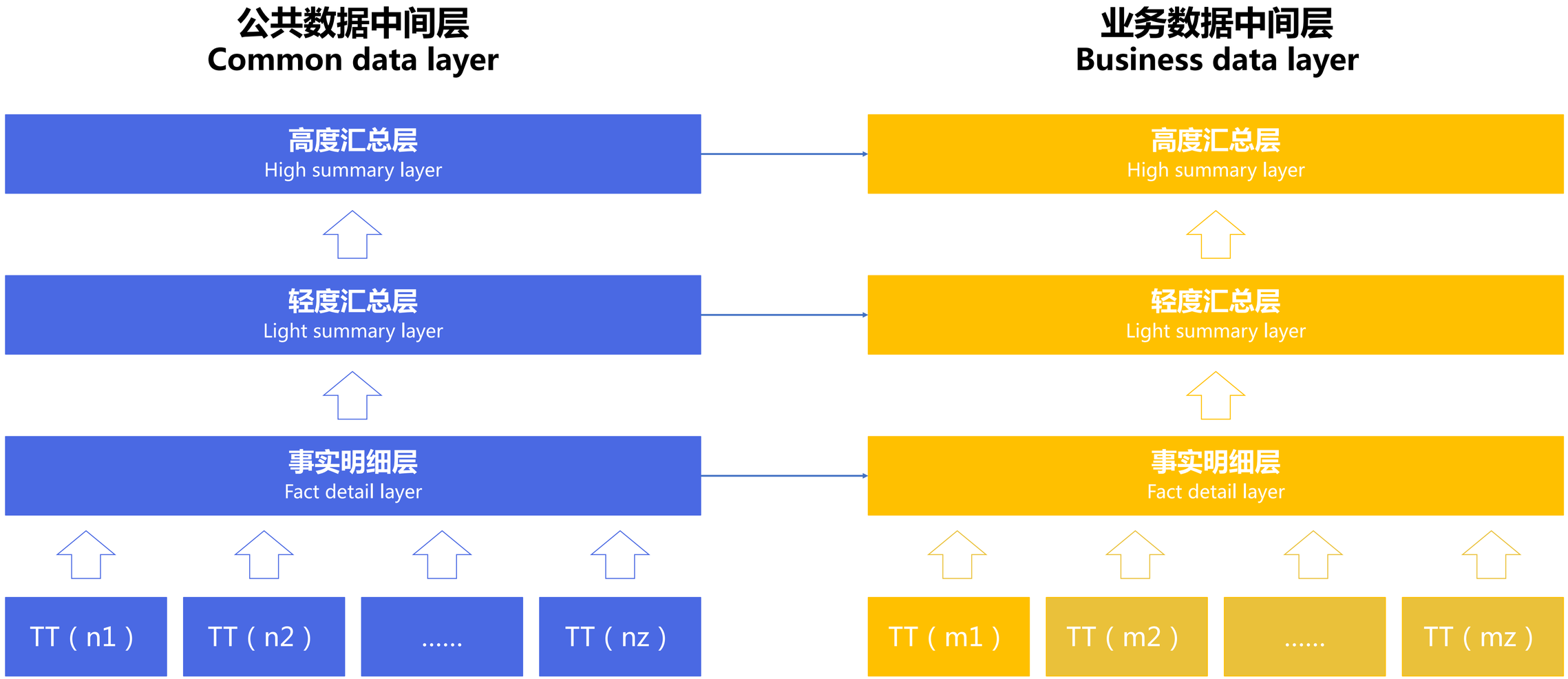
数据服务
Data Service



- 使用成本比较高，尤其是针对一些NoSQL的数据库
  - High cost of use, especially for some NoSQL-based database.
- 数据使用不可控，如热点阻断、权限控制、全链路监控等
  - Data security is not controllable. For example, the lack of full-link monitoring.

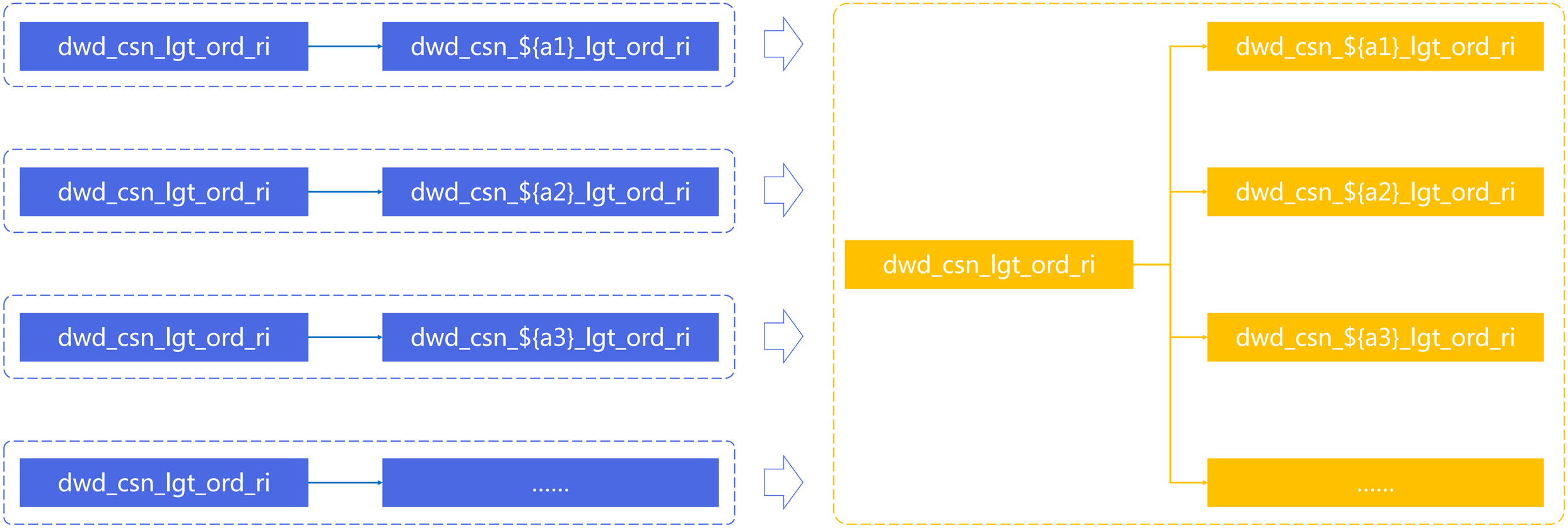
数据模型的升级 – 模型分层，充分复用公共中间层模型

Upgrade of data model - Layering data model to improve the reusability of common data layer model



数据模型的升级 – 预置分流，充分复用公共预置分流模型

Upgrade of data model - Preset dataflow is separated to improve the reusability of common data layer model

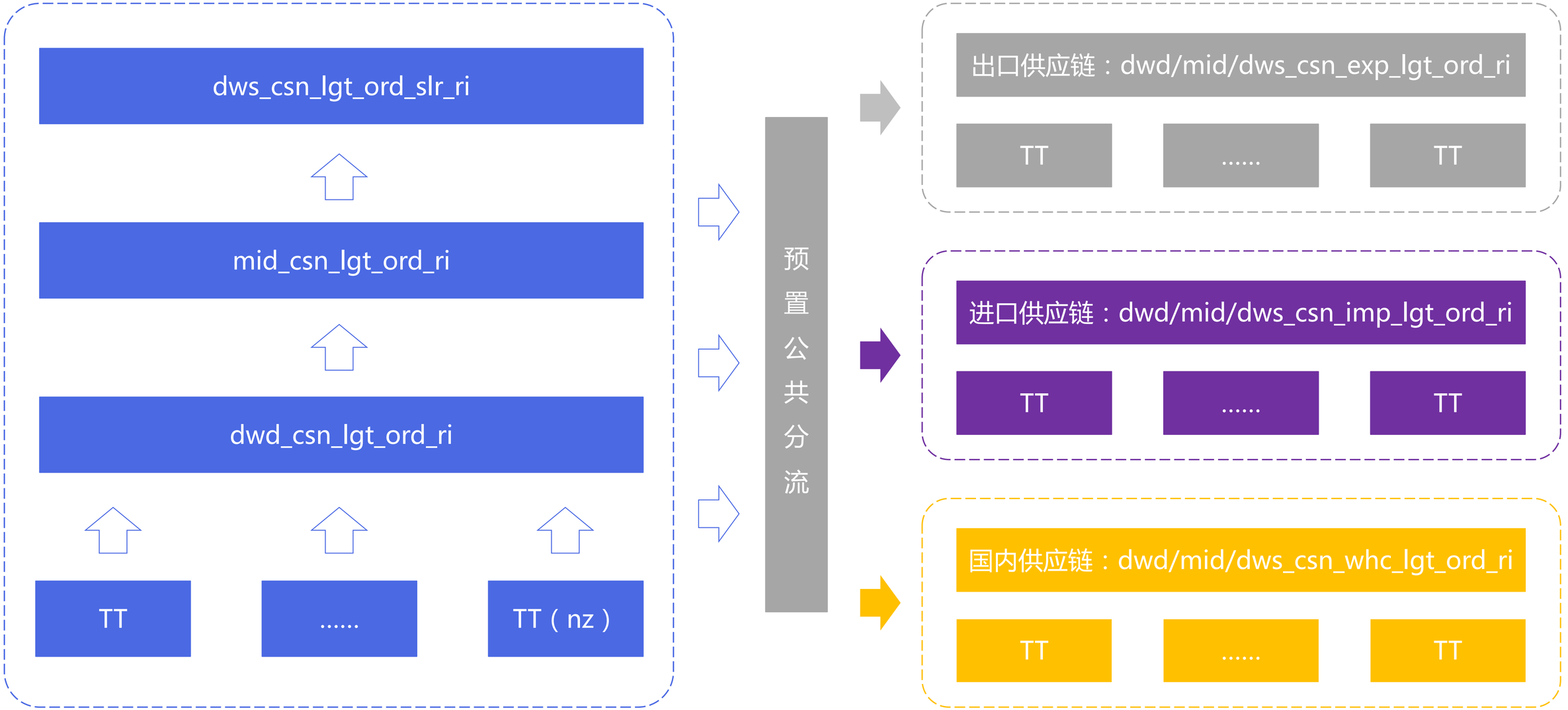


将原来下游做的分流作业，全部转移到上游一个公共分流作业来完成，可以大大节省计算资源。

Transfering from many sub-jobs to a common job can save many of computing resources.

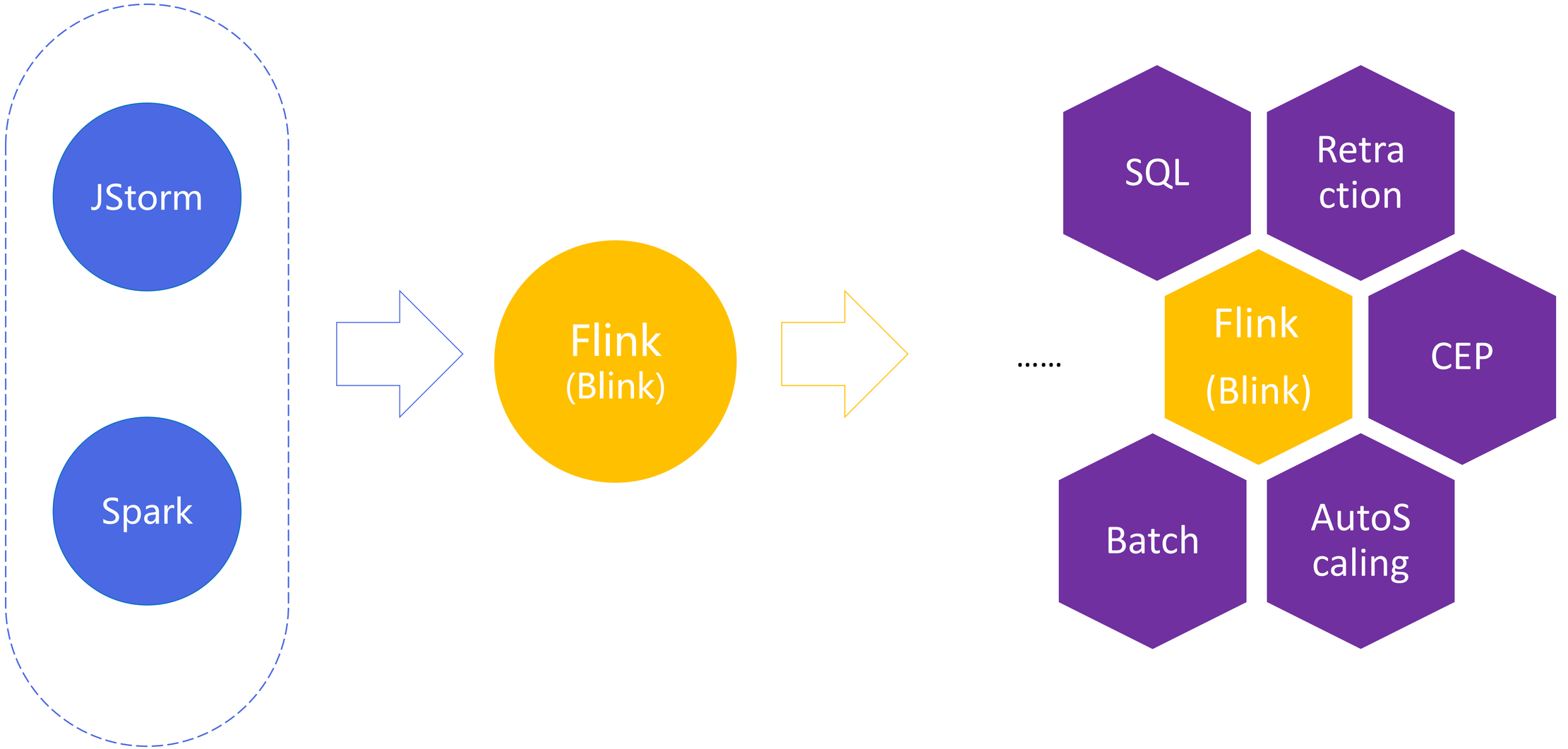
数据模型的升级 – 案例：菜鸟供应链实时数据模型

Upgrade of data model – Cainiao SCM data model



计算引擎的升级 – 基于Flink的实时计算引擎

Upgrade of computing engine - Real-time computing engine based on Flink



计算引擎的升级 – 案例1：神奇的Retraction

Upgrade of computing engine - Amazing retraction

物流订单号 lg_order_code	创建时间 gmt_create	是否取消 is_cancel	计划配送公司 plan_tms
LP1	2019-10-01 00:01:00	Y	tmsA
LP2	2019-10-01 00:05:00	Y	tmsA
LP1	2019-10-01 00:01:00	Y	tmsA
LP1	2019-10-01 00:01:00	Y	tmsB
LP2	2019-10-01 00:05:00	Y	tmsA
LP3	2019-10-01 00:18:00	Y	tmsA
LP2	2019-10-01 00:05:00	Y	tmsA
LP1	2019-10-01 00:01:00	Y	tmsC
LP3	2019-10-01 00:18:00	Y	tmsA
LP2	2019-10-01 00:05:00	Y	tmsA
LP3	2019-10-01 00:18:00	Y	tmsA
LP3	2019-10-01 00:18:00	N	tmsA

如何统计每个配送公司计划履行多少有效单量？

For each tms, how to count the number of plan orders?

计划发货仓 plan_store	创建物流订单量 create_order_cnt
tmsA	3（LP1+LP2+LP3）
tmsB	1（LP1）
tmsC	1（LP1）

计划发货仓 plan_store	创建物流订单量 create_order_cnt
tmsA	1（LP2）
tmsC	1（LP1）

利用Flink内置的Retraction机制
可以轻松实现流式消息的回撤统计

It's very easy to use the flink's build-in retraction mechanism.

计算引擎的升级 – 案例1：神奇的Retraction

Upgrade of computing engine - Amazing retraction

```
--临时视图
create view dws_csn_whc_lgt_ord_tms_ri_v1 as
select   lg_order_code
        ,last_value(gmt_create) as gmt_create
        ,last_value(plan_tms  ) as plan_tms
        ,last_value(is_cancel ) as is_cancel
from     dwd_csn_whc_lgt_ord_ri_v1
group by lg_order_code
;


--最终结果
insert into dws_csn_whc_lgt_ord_tms_ri
select   substr(gmt_create, 1, 10)
        ,plan_tms
        ,count(lg_order_code) as
plan_lgtord_cnt
from     dws_csn_whc_lgt_ord_tms_ri_v1
where    coalesce(is_cancel, 'N') = 'N'
group by substr(gmt_create, 1, 10)
        ,plan_tms
;
```

利用Flink SQL内置函数last_value，获取聚合key的最后一条非空的数值

一旦gmt_create、plan_tms、is_cancel中的任何一个字段发生变化，都会触发Flink的retraction机制

计算引擎的升级 – 案例2：实时超时统计的福音

Upgrade of computing engine - Real-time timeout statistics

日志时间 gmt_modified	物流订单号 lg_order_code	出库时间 storeout_time	揽收时间 tmsgot_time
2019-10-01 00:01:00	LP1		
2019-10-01 00:05:00	LP1	2019-10-01 00:05:00	
2019-10-01 00:10:00	LP2		

业务需求：仓出库超6小时配未揽收的单量？

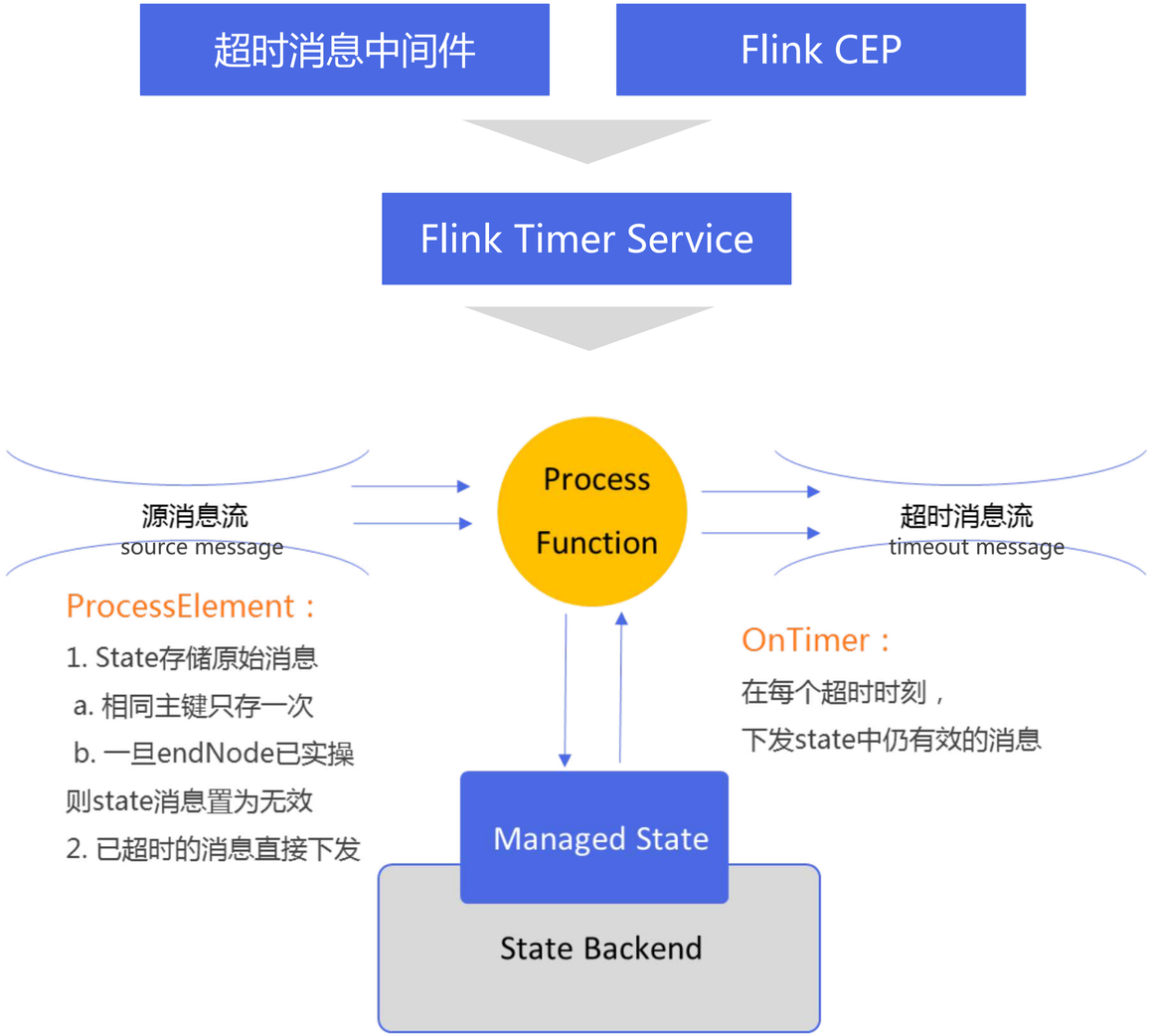
How to count the number of orders which is consigned but not collected more than 6 hours.

难点：如果仓出库后配未揽收，意味着没有新的消息流入，没有消息，如何进行超时消息的计算呢？

If the order is consigned but not collected, it means that there is no new message inflow. In this case, how to count the num of timeout message?

方案：没有消息，通过Flink制造消息！

Making timeout messages through flink!



计算引擎的升级 – 案例2：实时超时统计的福音

Upgrade of computing engine - Real-time timeout statistics

```
--创建执行环境
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
DataStream<Row> pds = env.addSource(tt4Source).....process(new TimeOutEmit())....
--------------------------------------------------------------------------------
-- TimeOutEmit函数
public void processElement(Tuple2<String, TimingMsg> value, ProcessFunction<Tuple2<String,
TimingMsg>, TimingMsg>.Context context, Collector<TimingMsg> collector) throws Exception {
  ...
  for (String timingHour : timingHours) {
     long registerTime = startNodeTimeStamp + Long.valueOf(timingHour) * 3600000L;
     if (registerTime > context.timerService().currentProcessingTime()) {
        context.timerService().registerProcessingTimeTimer(registerTime);
      } ...
   }
   this.state.update(currentMsg);
}
public void onTimer(long timestamp, ProcessFunction<Tuple2<String, TimingMsg>,
TimingMsg>.OnTimerContext ctx, Collector<TimingMsg> out) throws Exception {
  TimingMsg result =  this.state.value();
  ...
   out.collect(result);
  ...
}
```

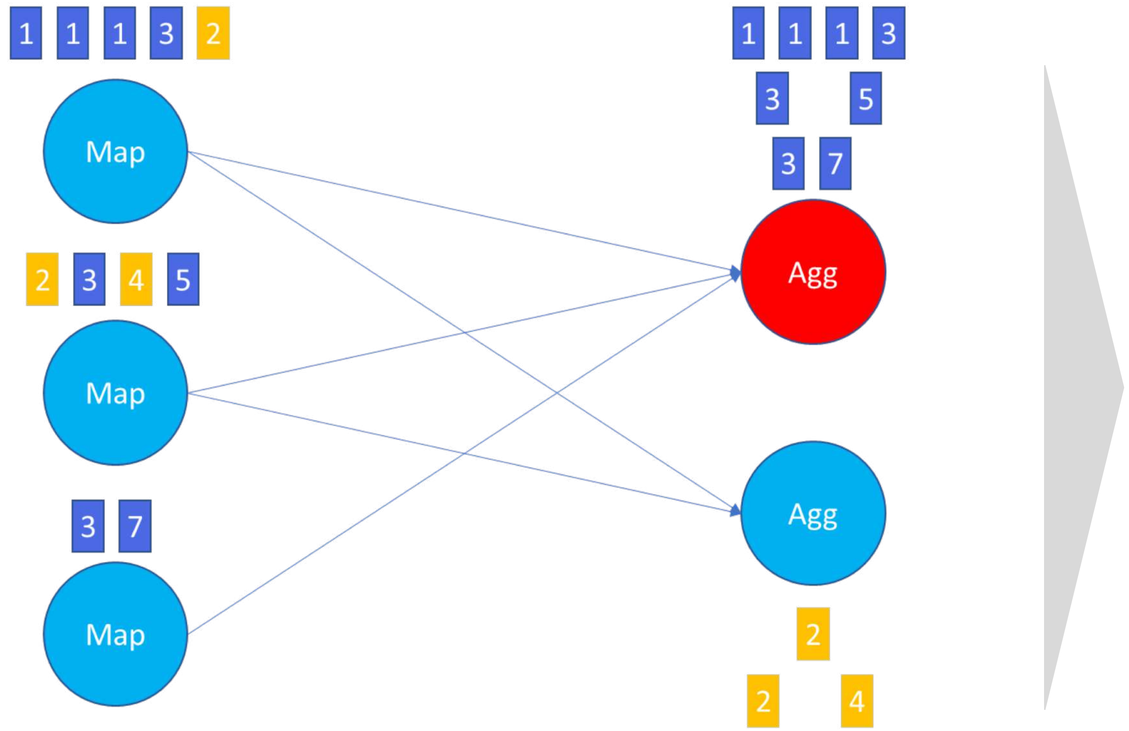
构造Process Function（访问keyed state 和 timers）

processElement，告诉state存储什么样的数据，并为每个超时事件注册一个timerService

onTimer，在超时的时刻去state读取数据，并将超时消息下发

计算引擎的升级 – 案例3：从手动优化到智能优化

Upgrade of computing engine - From manual optimization to intelligent optimization



数据热点

Hotspot

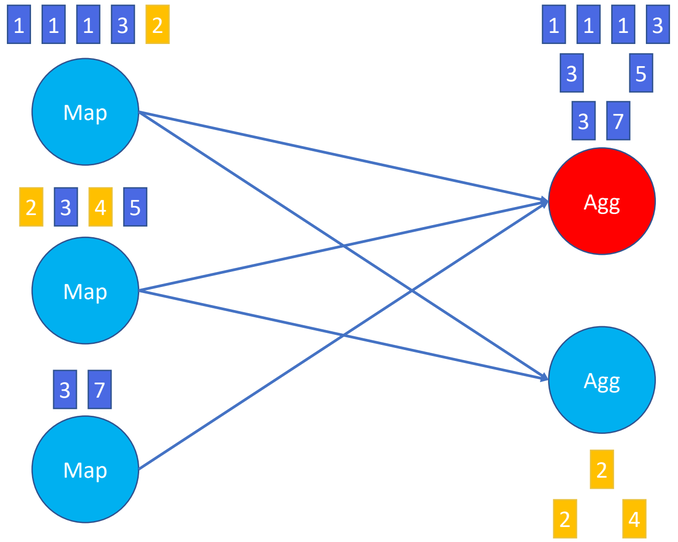
```
--hash散列
create view dws_csn_whc_lgt_ord_si_ri_v1 as
select   mod(hash_code(lg_order_code),256)
        ,substr(gmt_create, 1, 10)
        ,service_item_id
        ,count(distinct lg_order_code) as
mid_crt_lgtord_cnt
from     source_dwd_csn_whc_lgt_fl_ord_ri
group by mod(hash_code(lg_order_code),256)
        ,substr(gmt_create, 1, 10)
        ,service_item_id
;

--汇总结果
create view dws_csn_whc_lgt_ord_si_ri as
select   substr(gmt_create, 1, 10) as stat_date
        ,service_item_id
        ,sum(mid_crt_lgtord_cnt) as
crt_lgtord_cnt
from     dws_csn_whc_lgt_ord_si_ri_v1
group by substr(gmt_create, 1, 10)
        ,service_item_id
;
```

计算引擎的升级 – 案例3：从手动优化到智能优化

Upgrade of computing engine - From manual optimization to intelligent optimization

数据热点

Hotspot



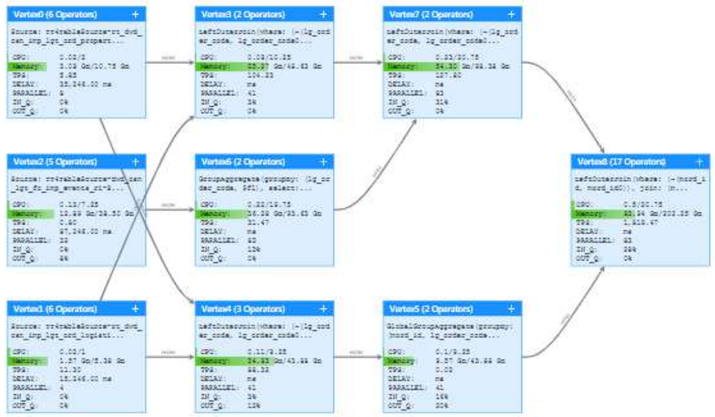
MiniBatch — 减轻对state的查询压力
Relief the pressure of state

LocalGlobal — 轻松应对count热点
Easy to deal with hot issues from count

PartialFinal — 轻松应对count_distinct热点
Easy to deal with hot issues from count distinct

资源配置

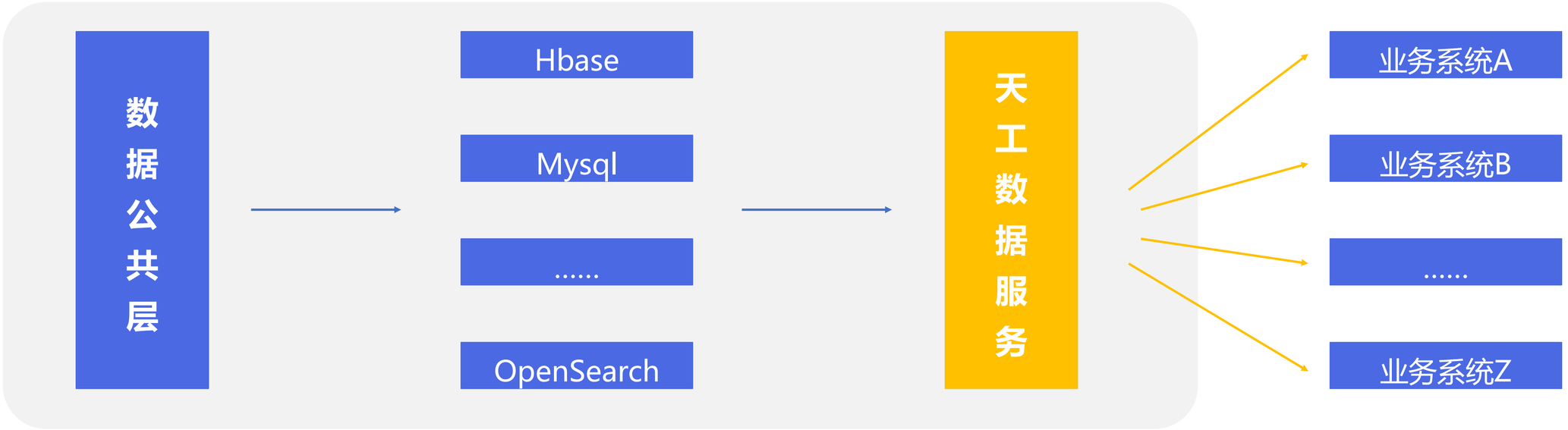
Resource config



AutoScaling智能调优
AutoScaling intelligent optimization

数据服务的升级 – 统一数据服务中间件

Upgrade of data service - Unified data service middleware



中心化

- 统一数据库接入
  Unified database access
- 统一的权限管理
  Unified authority management
- 统一的数据保障
  Unified data guarantee
- 统一全链路监控
  Full link monitoring

标准

- 将 SQL 做为一等公民，作为数据服务的 DSL
  Using SQL as the DSL of data service
- 提供标准化的服务接入方式（HSF）
  Provide standardized service access mode — HSF

数据服务的升级 – 案例1：NoSQL To TgSQL

Upgrade of data service – NoSQL to TgSQL

Employee（物理表）

rowkwy	CF：base	CF：extends
666666	name:xiaoming;age:31;	team:菜鸟数据部;bu:菜鸟
……	……	……

mapping

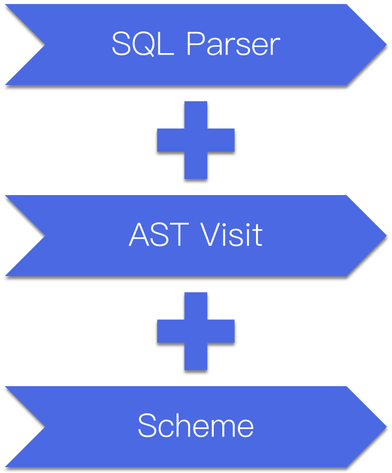
Employee_hb（逻辑表）

rowkwy	name	age	team	bu
666666	xiaoming	32	菜鸟数据部	菜鸟
……	……	……	……	……

```
SELECT
    team
    ,avg(age) AS avg_age
FROM
    employee
WHERE
    rowkey BETWEEN '10000' AND '80000'
    AND bu = '菜鸟'
GROUP BY
    team
ORDER BY
    avg_age
```

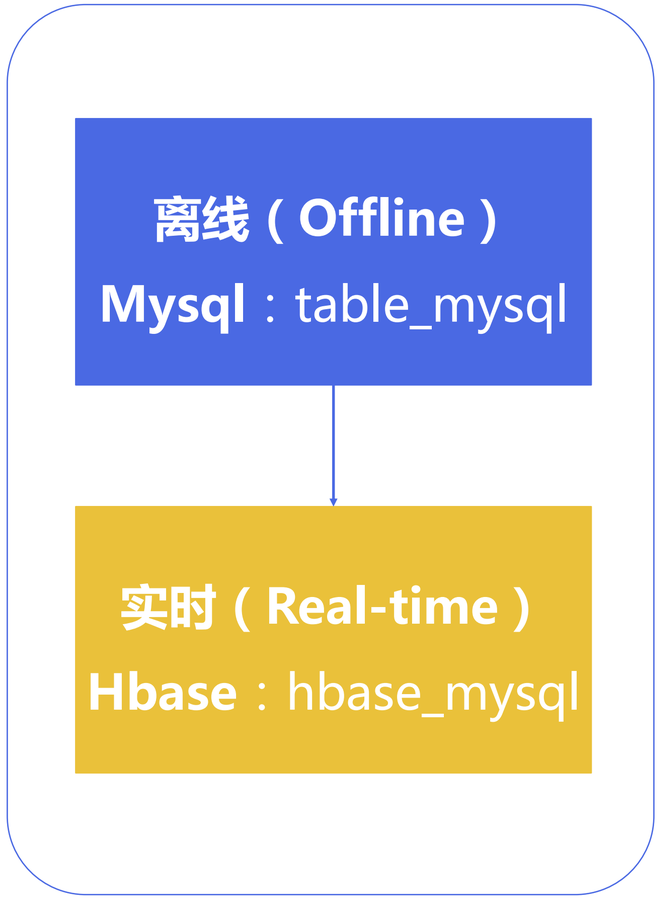
```
Scan scan = new Scan('10000'.byte(), '80000'.byte());

scan.addColumn('extends'.byte(), 'team'.byte());
scan.addColumn('base'.byte(), 'age'.byte());
scan.addColumn('extends'.byte(), 'bu'.byte());

HTableInterface tableInterface = tablePool.getTable('employee');

SingleColumnValueFilter filter = new SingleColumnValueFilter(
    Bytes.toBytes('extends'),
    Bytes.toBytes('bu'),
    CompareOp.EQUAL,
    Bytes.toBytes('菜鸟')
);

scan.setFilter(filter);

ResultScanner scanner = tableInterface.getScanner(scan);
```

```
SelectFunction.execute(
    result,
    selectVistor.getAggregate(),
    selectVistor.getGroup(),
    selectVistor.getOrder(),
    selectVistor.limit
);
```

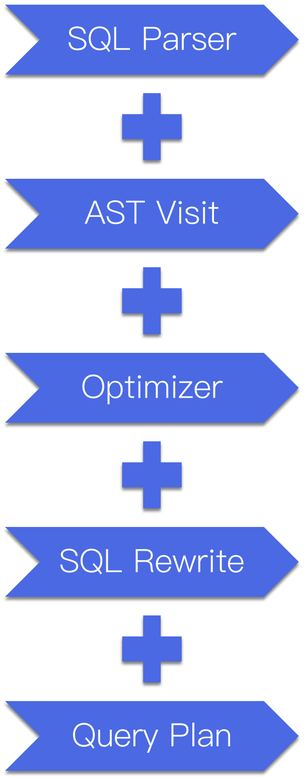


数据服务的升级 – 案例2：跨源数据查询

Upgrade of data service - Cross-source data query




```
select   a.store_code
        ,a.plan_lgtord_cnt
        ,b.real_lgtord_cnt
from
        (select   *
         from     mysql_table
         where    cate_id = 101
         ) a

left outer join

        (select   *
         from     hbase_table
         where    cate_id = 101
         ) b

on      a.store_code = b.store_code
```



数据服务的升级 – 案例3：服务保障升级

Upgrade of data service – Upgrade of Service guarantee

FLINK FORWARD

数据服务保障

- 主备切换 Rapid switchover
- 主备双活 Dual active
- 动态负载 Dynamic load balancing
- 热点服务阻断 Blocking-up hotspot
- 白名单限流 Whitelist current limit
- ……

其他技术工具的探索和创新

Exploration and innovation of other technical tools

FLINK FORWARD

实时压测

Pressure Measurement

过程监控

Process Monitor

资源管理

Resource Management

数据质量

Data Quality

菜鸟实时数仓 – 未来发展与思考

Upgrade of data service - Future development and thinking

联系方式

Contact information

使用任意APP扫码，收下我的名片

钉钉号：缘桥

备注：Flink Forward Asia 2019


THANKS