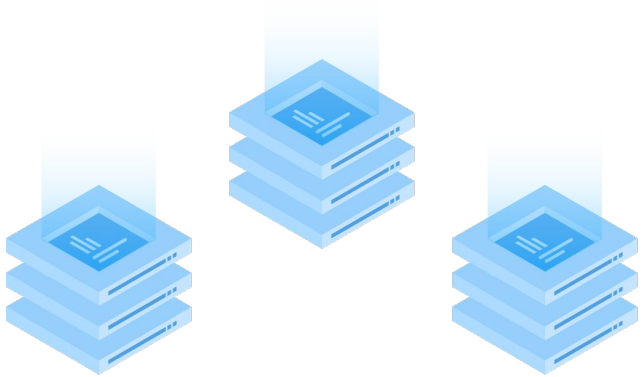
Coprocessor 下推计算实现分析

Presented by Wenxuan

PingCAP TiDB



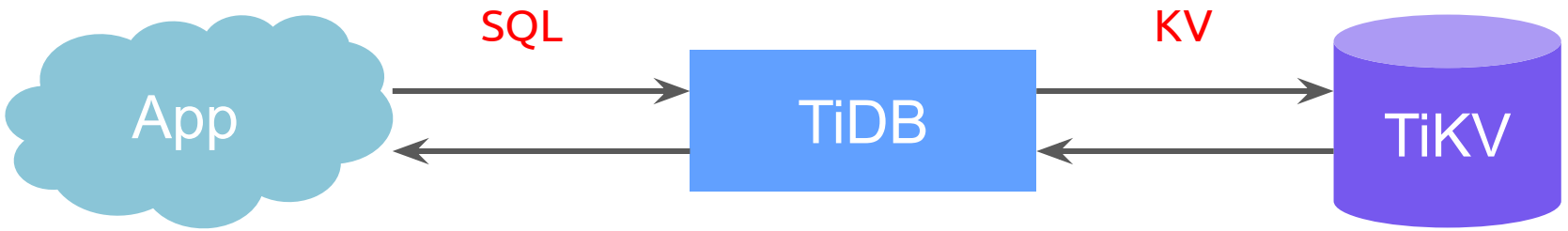
PingCAP

为什么要有 Coprocessor

朴素工作方式(无 Coprocessor)

```
SELECT COUNT(*)
FROM ...
WHERE ...
```

KvScan
```
StartKey: tXXXX_rXXXX
Limit: 1000
```



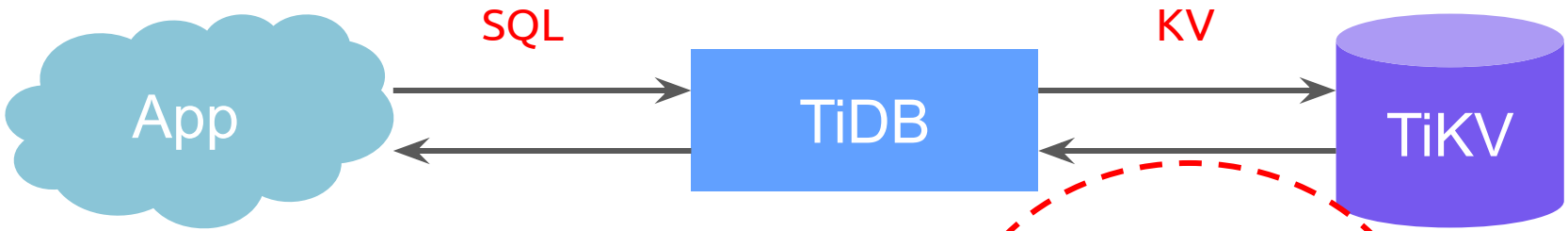
COUNT(*)
4

k1	v1
k2	v2
k3	v3
k4	v4

问题1:大量传输开销

```
SELECT COUNT(*)
FROM ...
WHERE ...
```

KvScan
```
StartKey: tXXXX_rXXXX
Limit: 1000
```



COUNT(*)
4

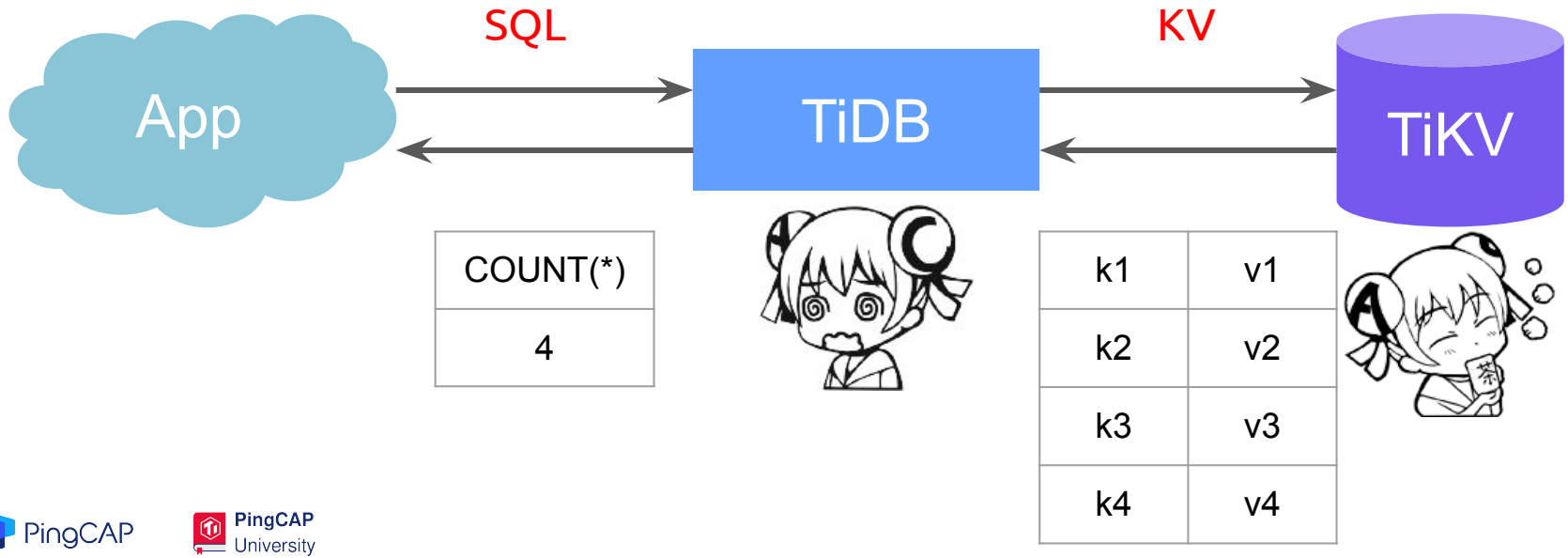
k1	v1
k2	v2
k3	v3
k4	v4

- 序列化
- 反序列化
- 网络传输

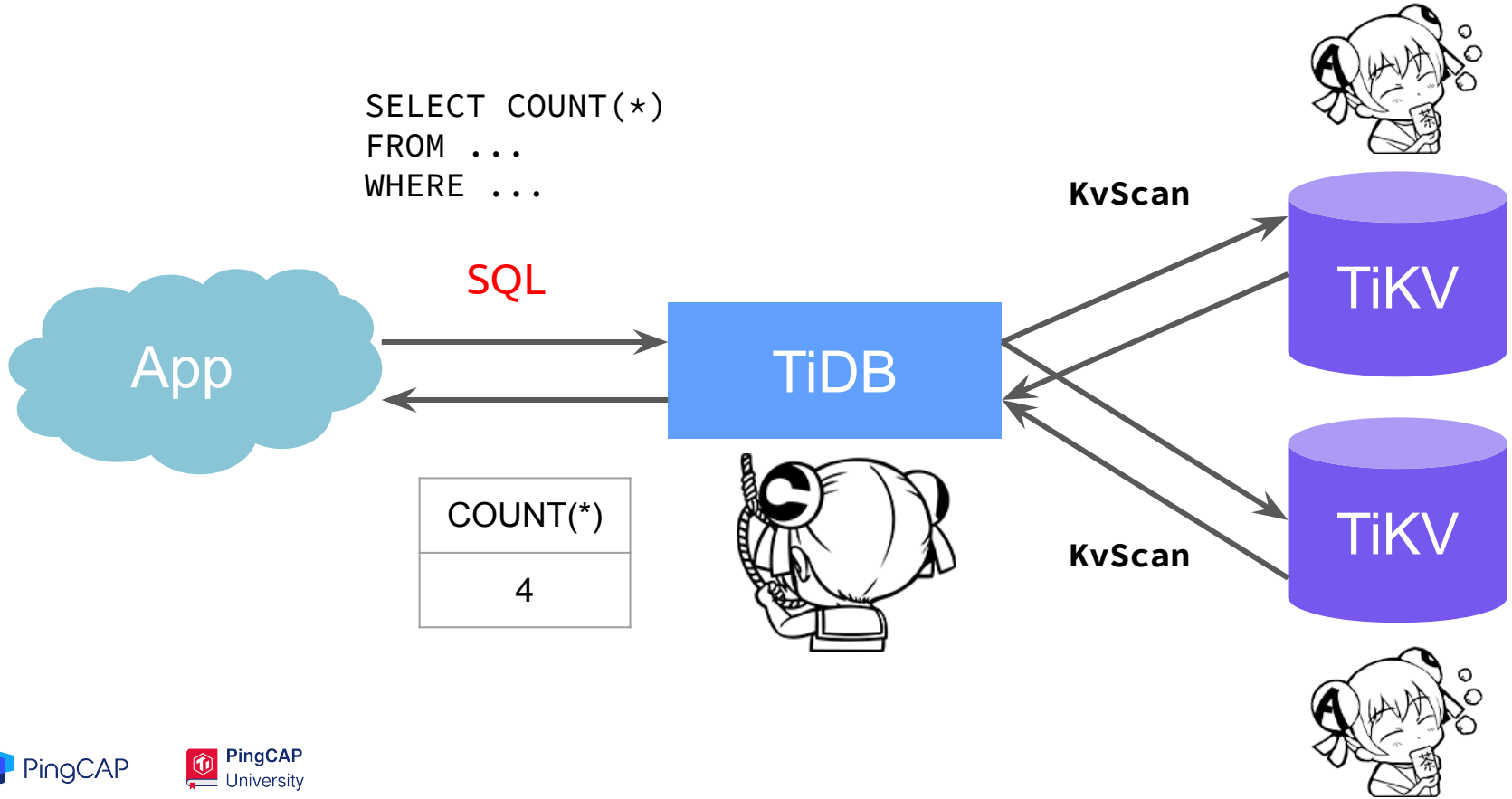
问题2：TiKV CPU 很闲，TiDB CPU 很忙

```
SELECT COUNT(*)
FROM ...
WHERE ...
```

KvScan
```
StartKey: tXXXX_rXXXX
Limit: 1000
```



另外……记住我们是分布式的



解决方法：将一部分计算任务交给 TiKV

- TiKV 中负责这部分计算任务的模块叫做 Coprocessor
- 这个过程也叫做（算子）下推

Coprocessor 可处理的请求

1. **DAG**
   - 执行物理算子，为 SQL 计算出中间结果，从而减少 TiDB 工作量和网络传输开销
   - 这是绝大多数场景下 Coprocessor 执行的任务
   - 虽然被称为 DAG，但实际上是各个算子 Pipeline 排列依次执行
2. **Analyze**
   - 分析表的数据，计算表数据的统计信息，持久化后可被 TiDB 优化器（CBO）采用
3. **Checksum**
   - 对表的全部数据生成校验和，可用于导入数据后进行一致性校验
4. 未来脑洞：DDL 数据更新？查询并更新(Update)？查询并删除(Delete)？

已实现的 DAG 物理算子

底层算子：

- **Table Scan**：根据指定范围扫表，并过滤出一部分列返回
  - 例：`SELECT col FROM t`
- **Index Scan**：根据指定范围扫索引，并过滤出一部分列返回
  - 例：`SELECT PK FROM t WHERE index > 5`

已实现的 DAG 物理算子

上层算子：

- **Selection**：根据指定表达式，对下层算子产生的行进行过滤
  - 例：`SELECT col FROM t WHERE a+b=10`

扫出 col、a、b 列　　按照 a+b=10 为条件过滤行

已实现的 DAG 物理算子

上层算子：

- **`Limit`**：限定只返回前若干行
  - 例：`SELECT col FROM t LIMIT 10`

扫出 col 列　　只返回前 10 行

已实现的 DAG 物理算子

上层算子：

- **TopN**：按照表达式进行排序后，再返回前若干行
  - 例：`SELECT col FROM t ORDER BY a+1 LIMIT 10`

扫出 col、a 列　　按照 a+1 排序，取前 10 行

已实现的 DAG 物理算子

上层算子：

- **Hash/Stream Aggregation**：按照表达式分组（如果有）并按照表达式聚合
  - 例：`SELECT count(1) FROM t GROUP BY score+1`

扫出 score 列

按照 score+1 分组，每组按照 count(1) 聚合

已实现的 DAG 物理算子

算子之间可以进行复杂组合，例：`SELECT count(1) FROM t WHERE age > 10`

扫出 age 列

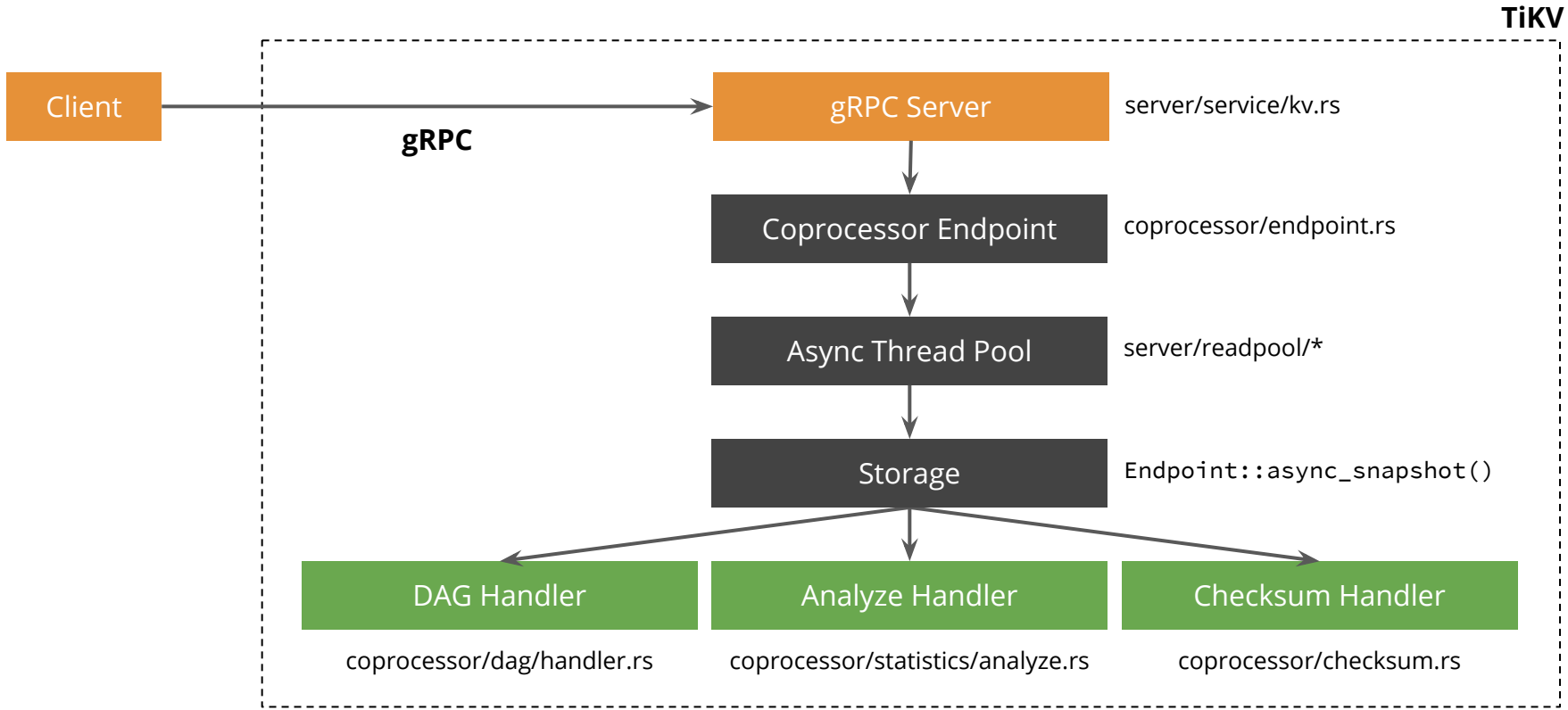
以 age > 10 为条件过滤

对所有行聚合 count(1)

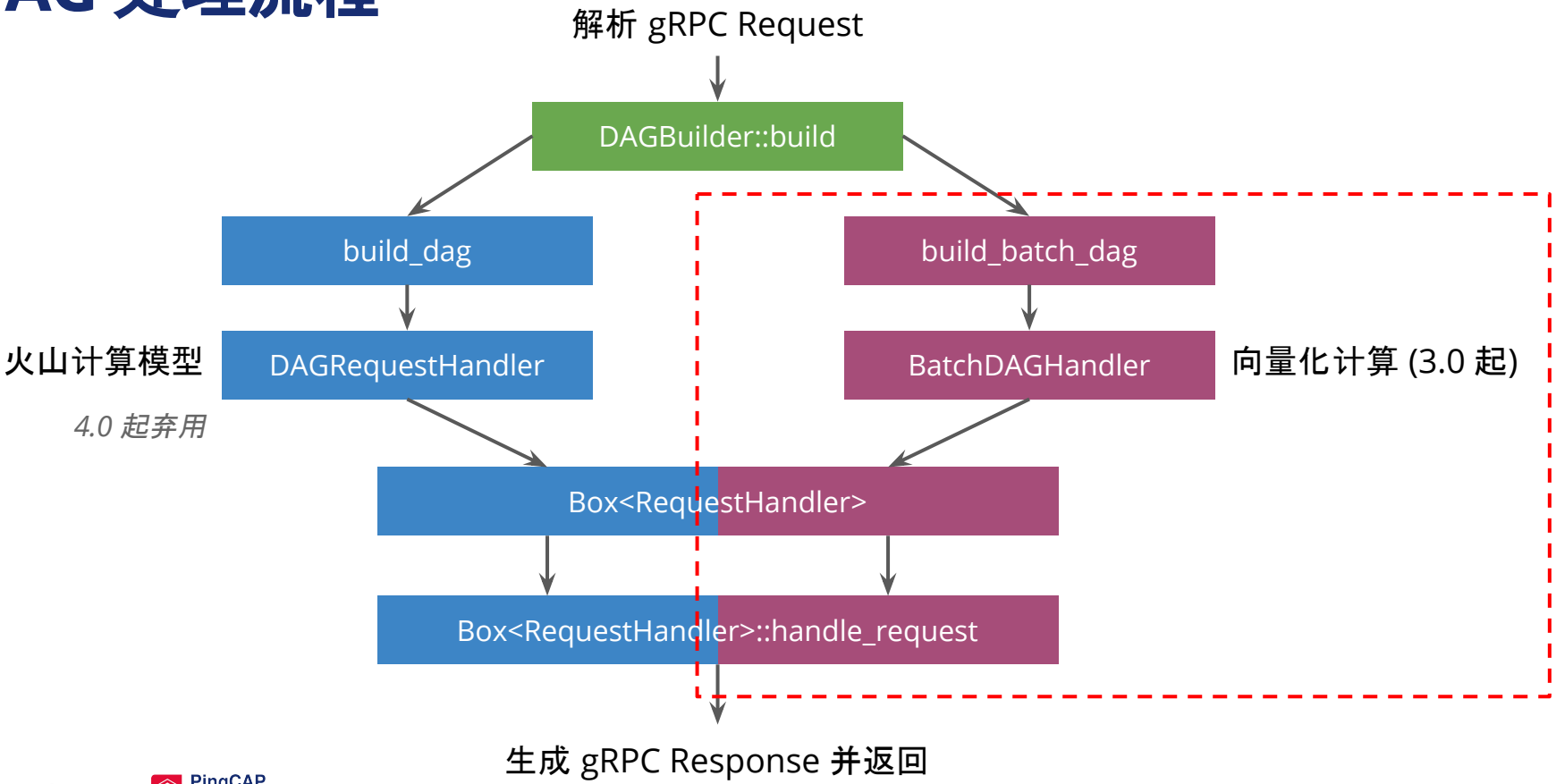
PingCAP

实现概览

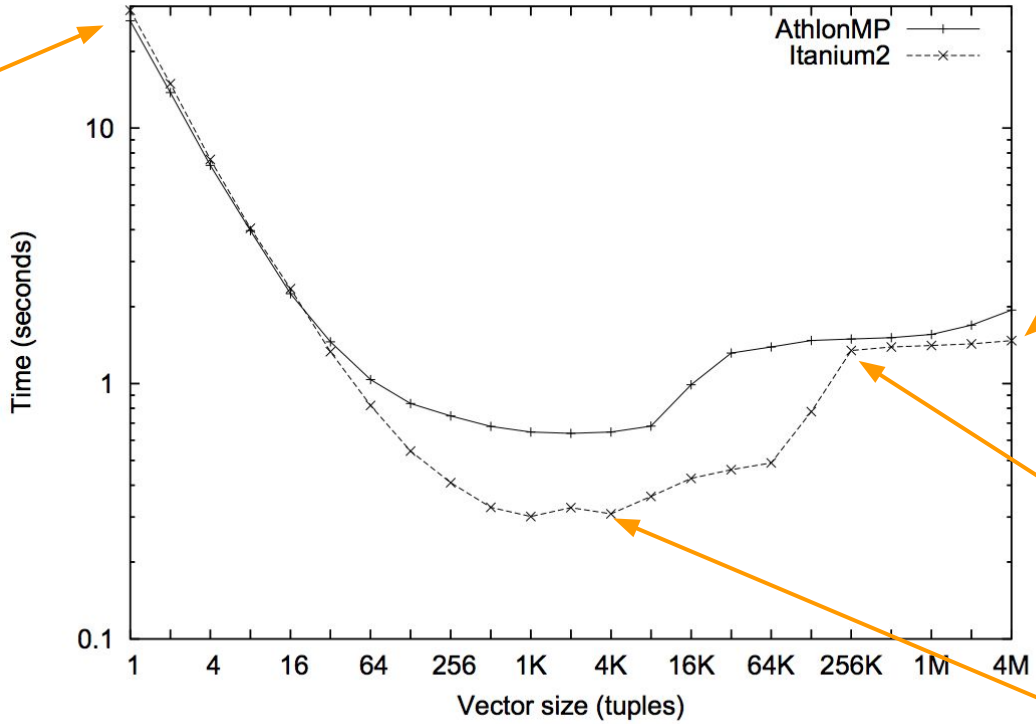
Coprocessor 处理流程



DAG 处理流程


解析 gRPC Request
DAGBuilder::build
build_dag
build_batch_dag
火山计算模型
DAGRequestHandler
BatchDAGHandler
向量化计算 (3.0 起)
4.0 起弃用
Box<RequestHandler>
Box<RequestHandler>::handle_request
生成 gRPC Response 并返回

火山模型
一次处理一行



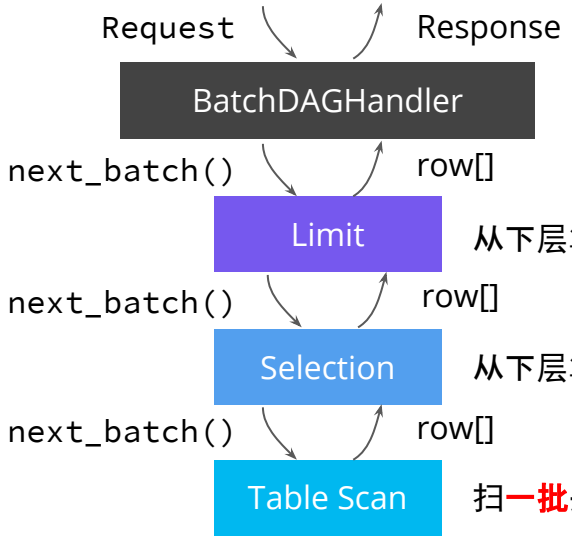
MonetDB
一次处理一整列

L3 exceeded

L1 + L2 exceeded

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



一批 = [Column, Column, Column, ...] 每列至多 1024 项

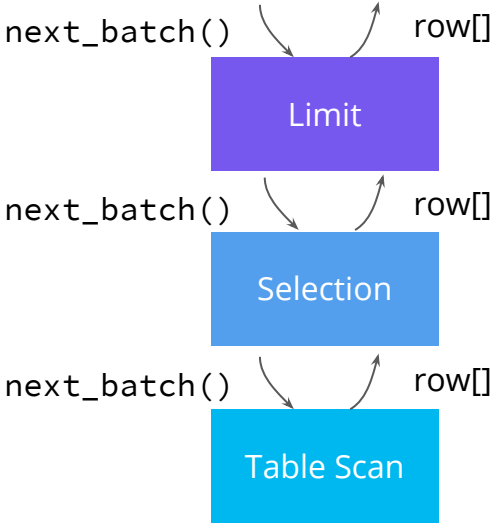
从下层算子取**一批**，若累计已返回了 3 行则告知上层停止

从下层算子取**一批**，按照 age > 10 为条件过滤，返回过滤后的**一批**给上层

扫**一批**并返回给上层，若没有更多数据了则告知上层停止

向量化计算

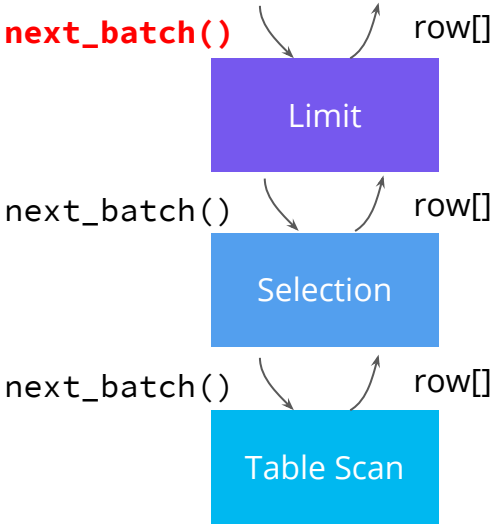
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

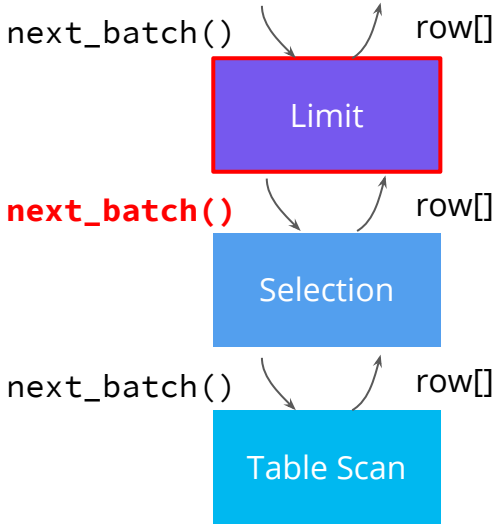
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

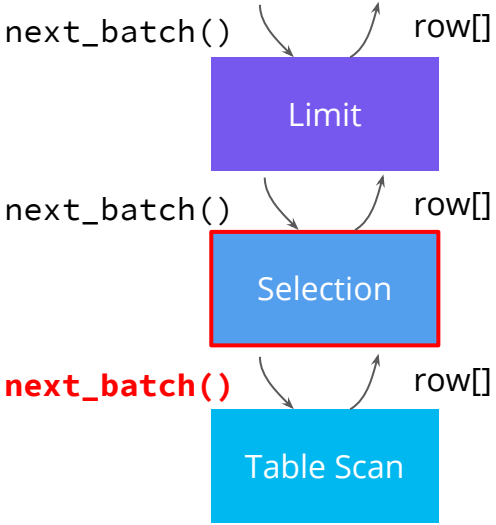
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

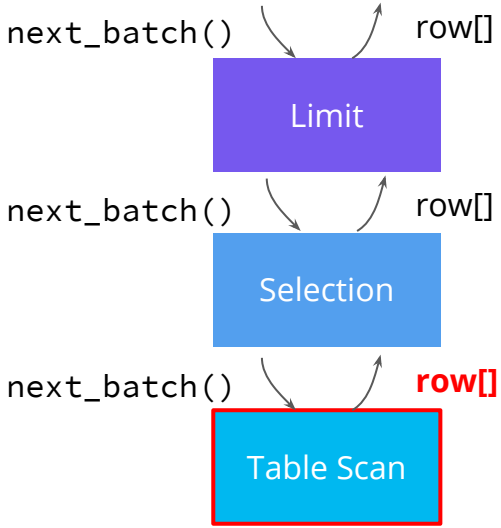
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



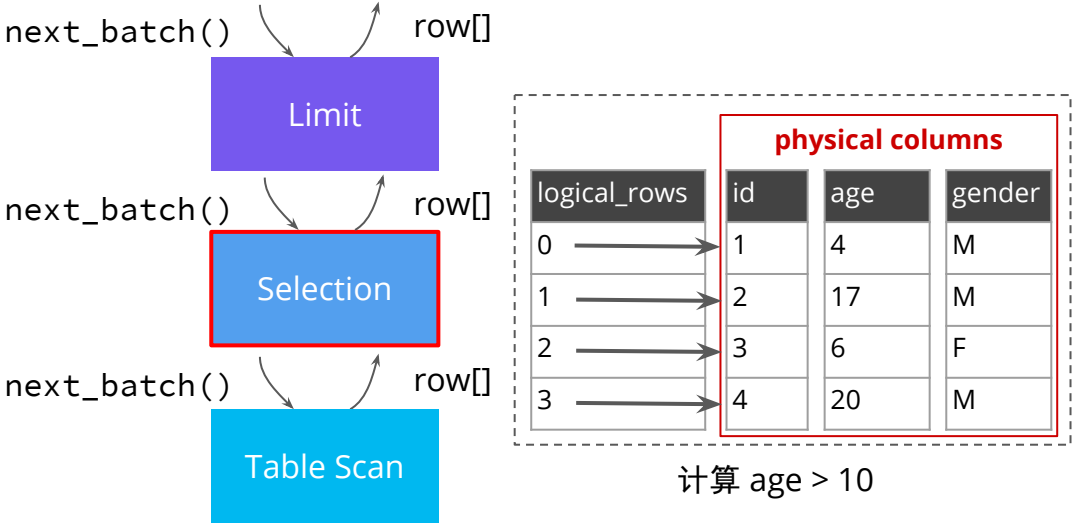
logical_rows	id	age	gender
0	1	4	M
1	2	17	M
2	3	6	F
3	4	20	M

physical columns

id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

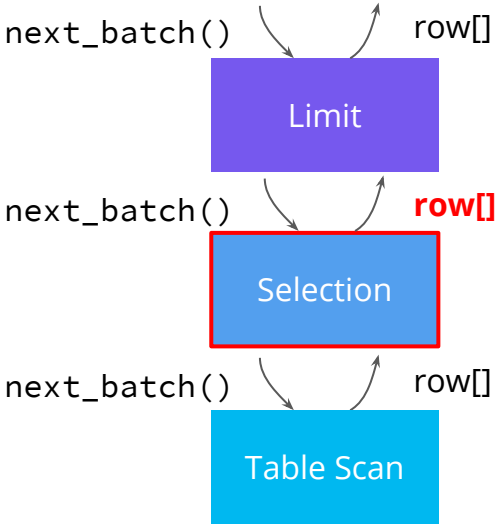


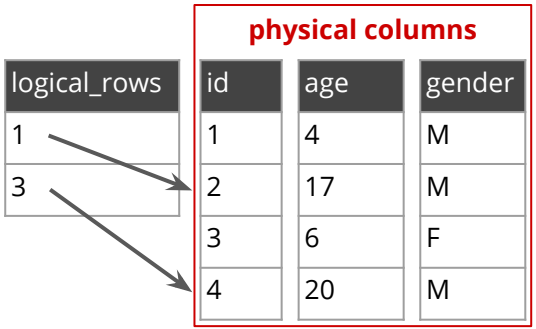
计算 age > 10

id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

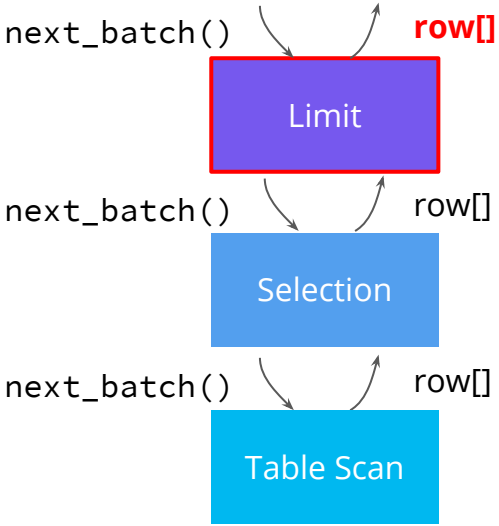




id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



logical_rows
1
3

physical columns

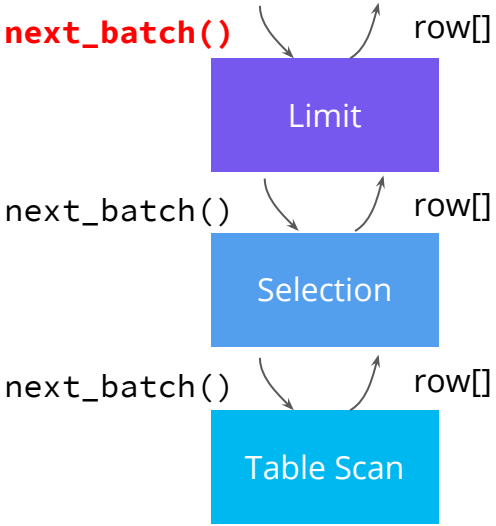
id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M

已输出 0 行，不到 3 行，继续

id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

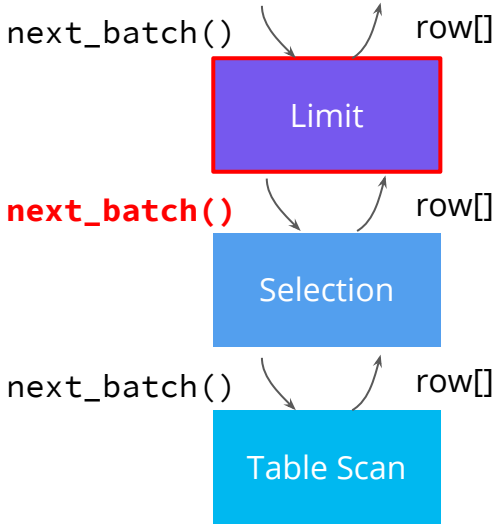
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

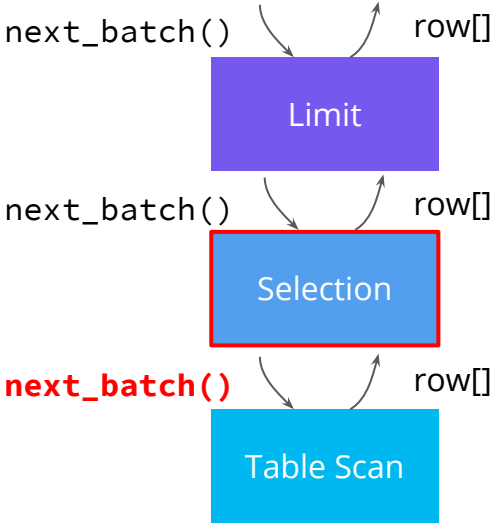
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

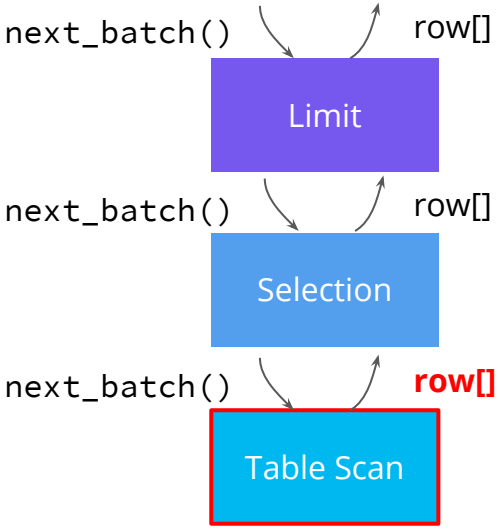
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

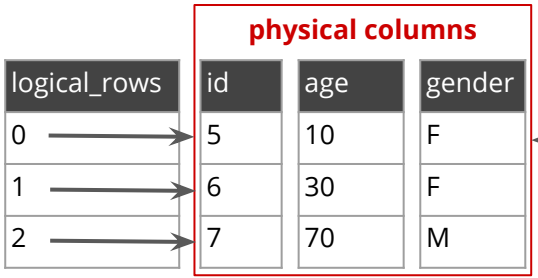


id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

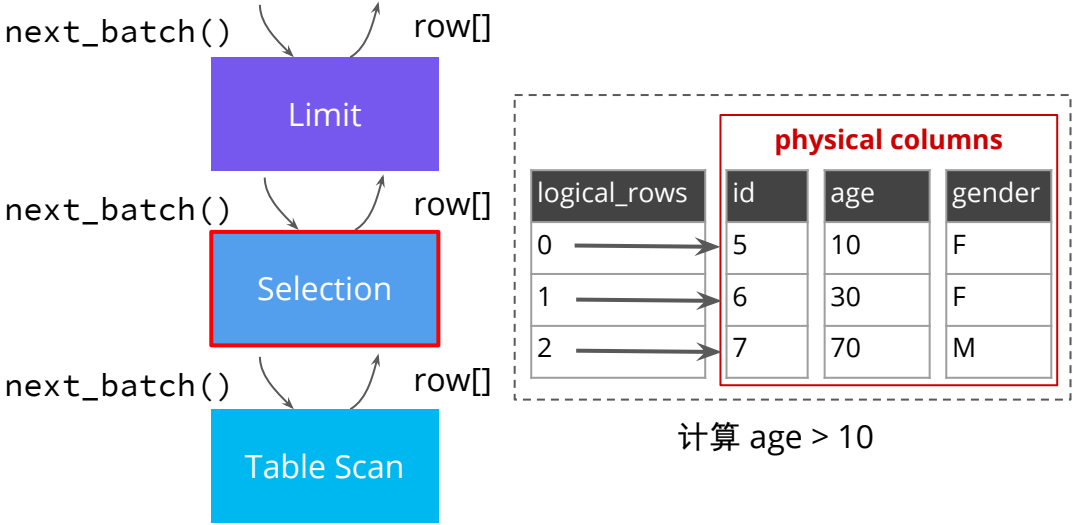




id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

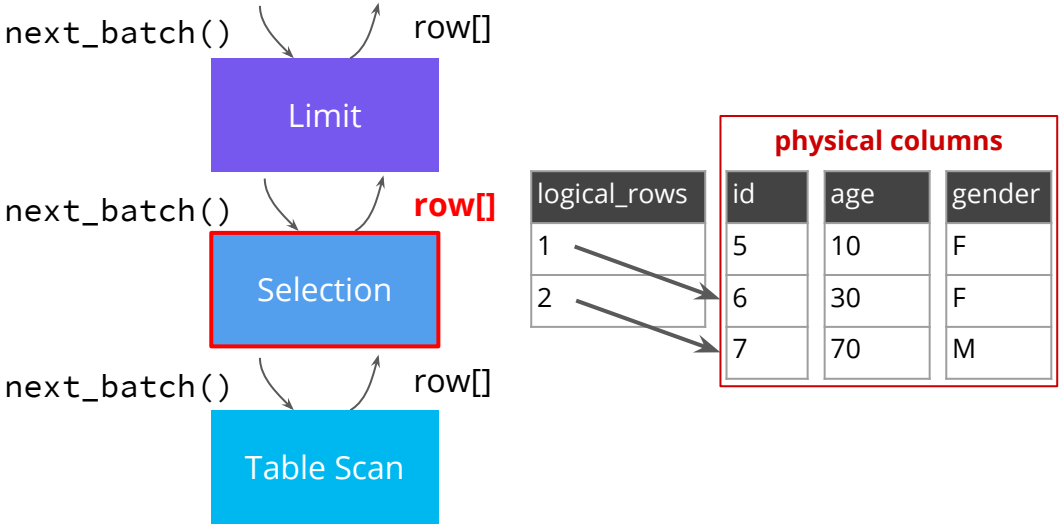


计算 age > 10

id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

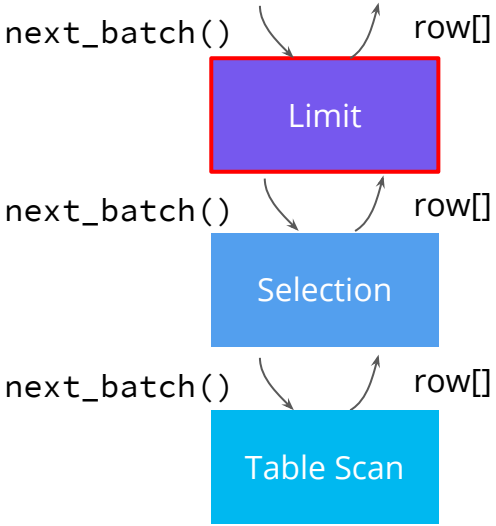
```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

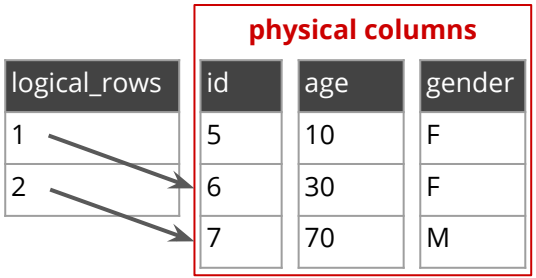


id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```



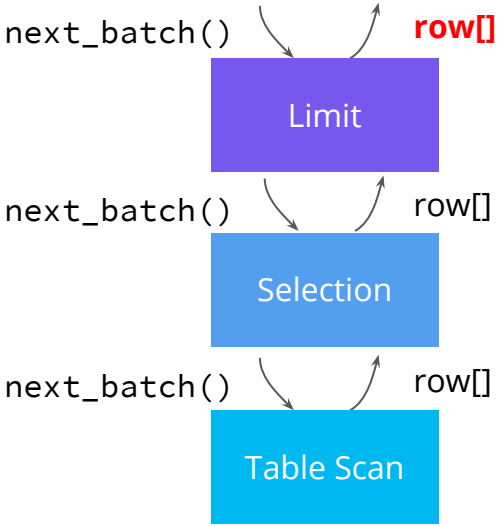


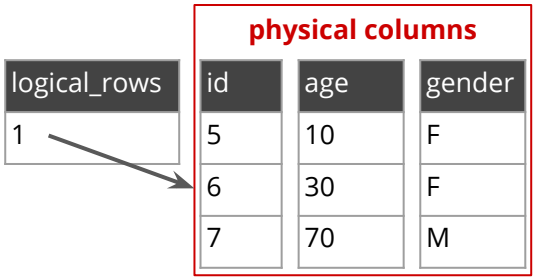
已输出 2 行，还剩 1 行可以输出，这批截断到 1 行再输出，然后停止

id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

向量化计算

```
SELECT * FROM t WHERE age > 10 LIMIT 3
```

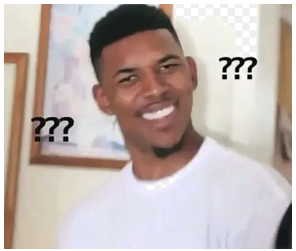




id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

表达式计算

给定一个表达式，输入一行源数据，如何得出表达式的 结果？

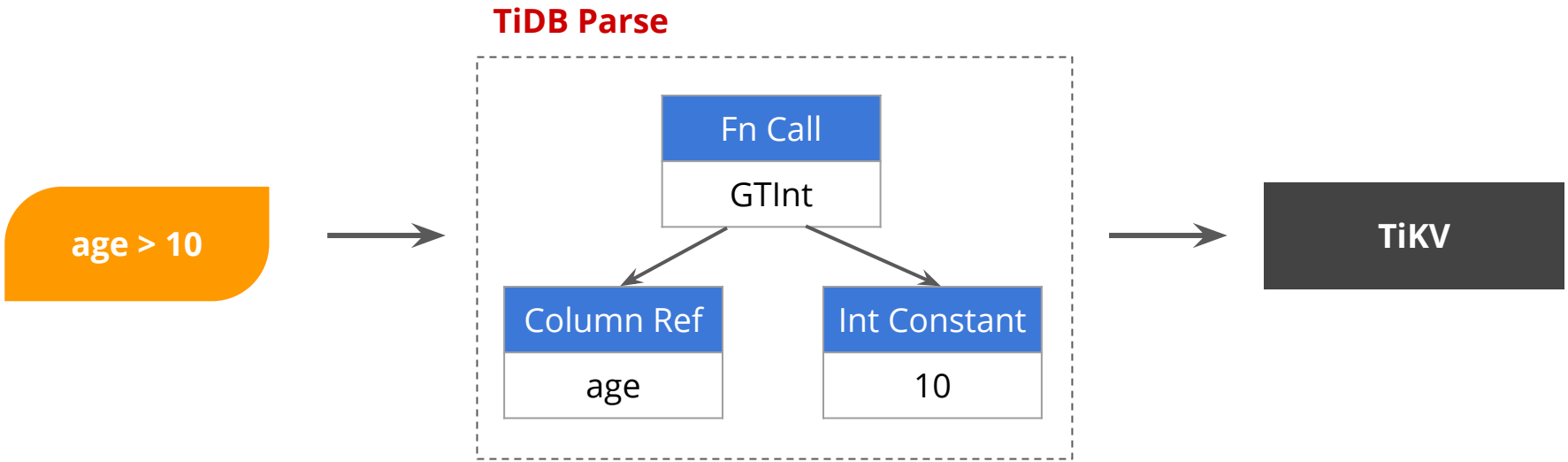


id	age	gender
1	4	M

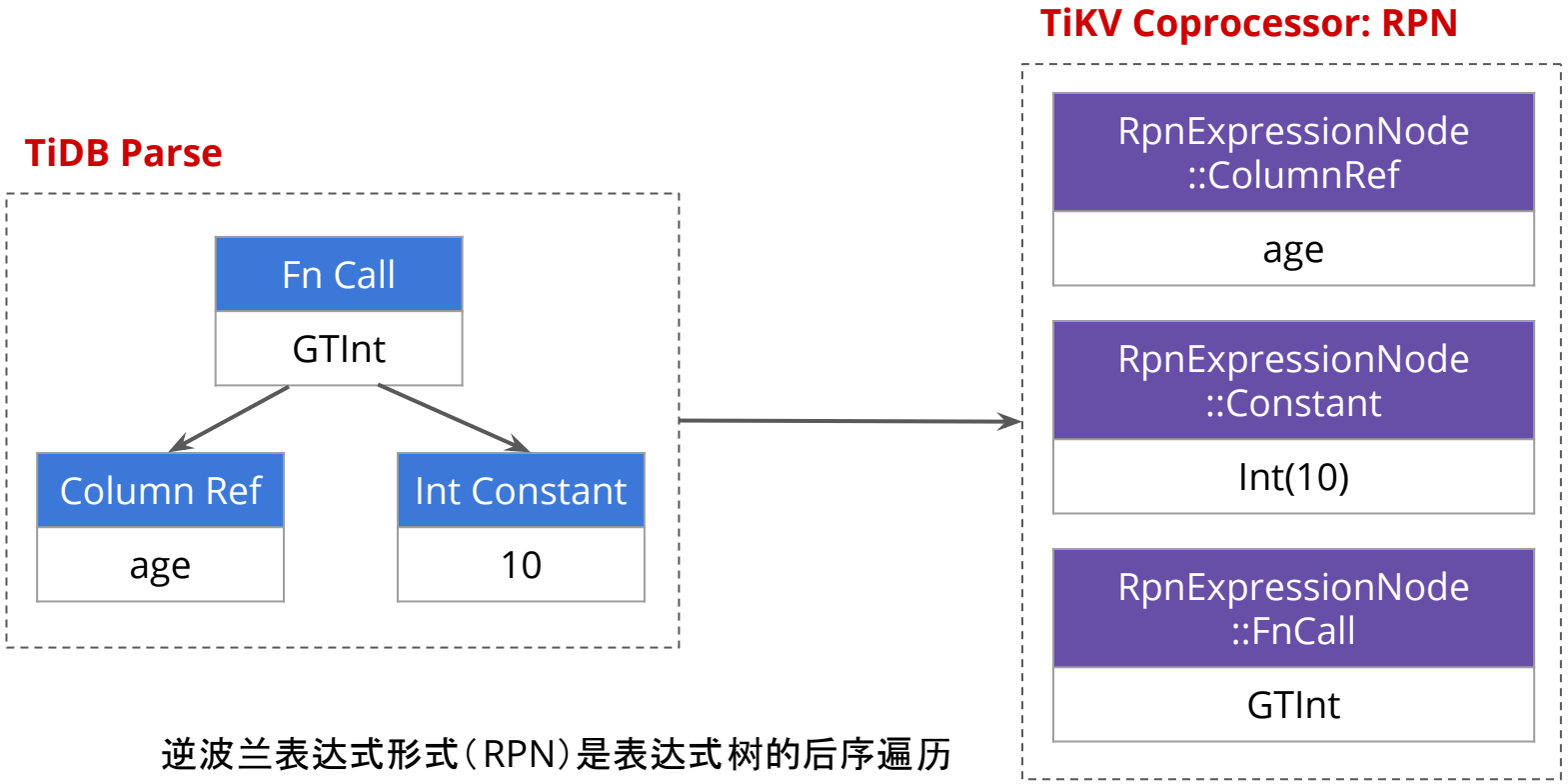
age > 10
FALSE

表达式向量化计算

给定一个表达式，输入**一批**源数据，计算**一批或一个**表达式结果。



表达式向量化计算



逆波兰表达式形式（RPN）是表达式树的后序遍历

表达式向量化计算

表达式

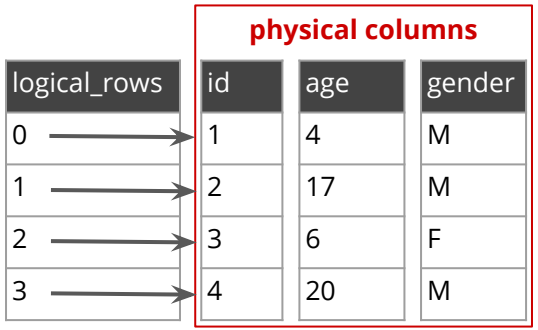
RpnExpressionNode ::ColumnRef
age

RpnExpressionNode ::Constant
Int(10)

RpnExpressionNode ::FnCall
GTInt

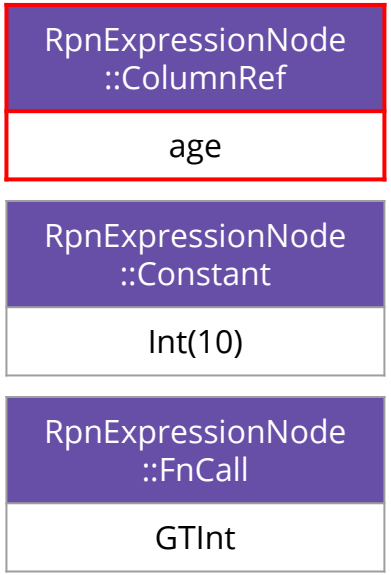
计算数据栈

输入源数据



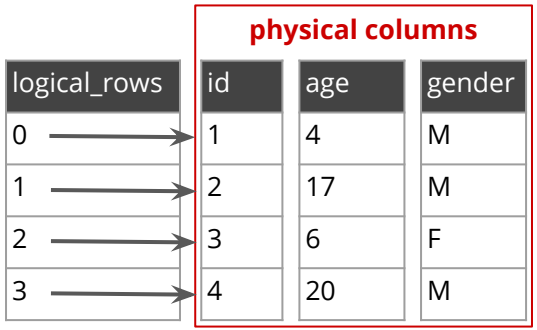
表达式向量化计算

表达式



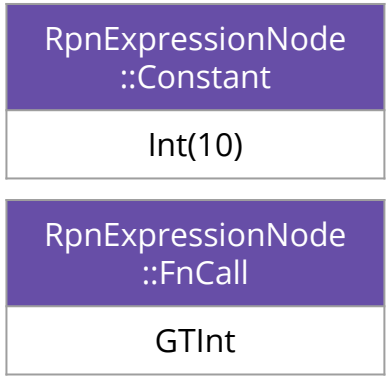
计算数据栈

输入源数据

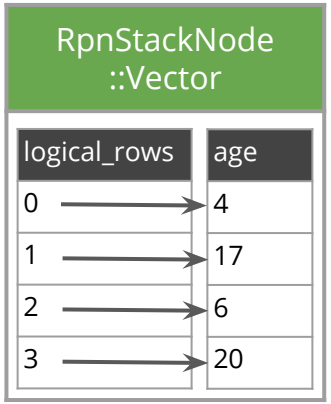


表达式向量化计算

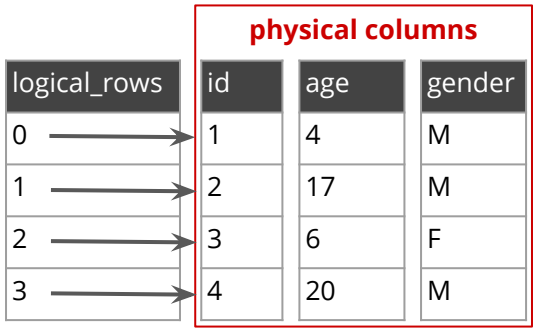
表达式



计算数据栈

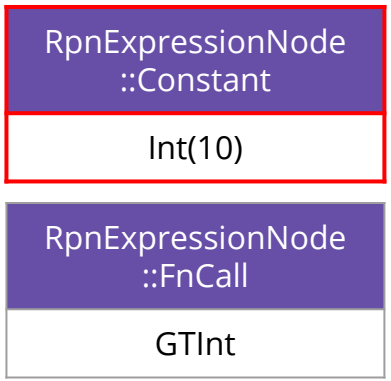


输入源数据

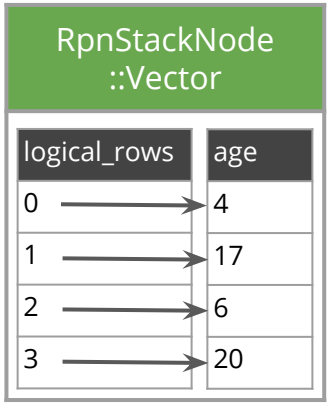


表达式向量化计算

表达式

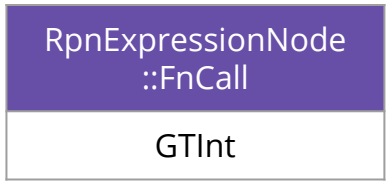


计算数据栈

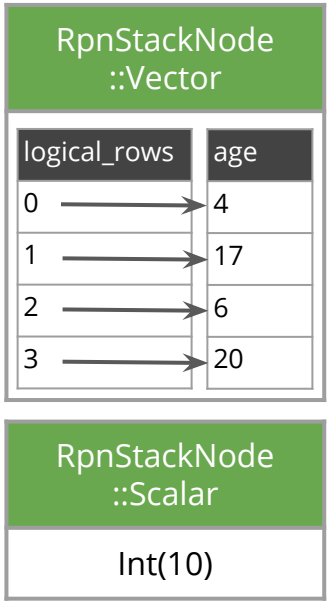


表达式向量化计算

表达式



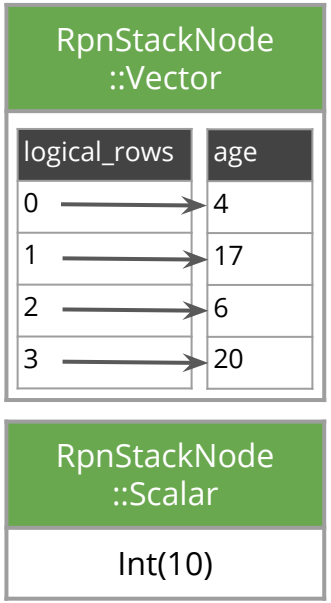
计算数据栈



表达式向量化计算

表达式

计算数据栈



函数调用 GTInt(Int, Int) -> Int 需要两个参数，

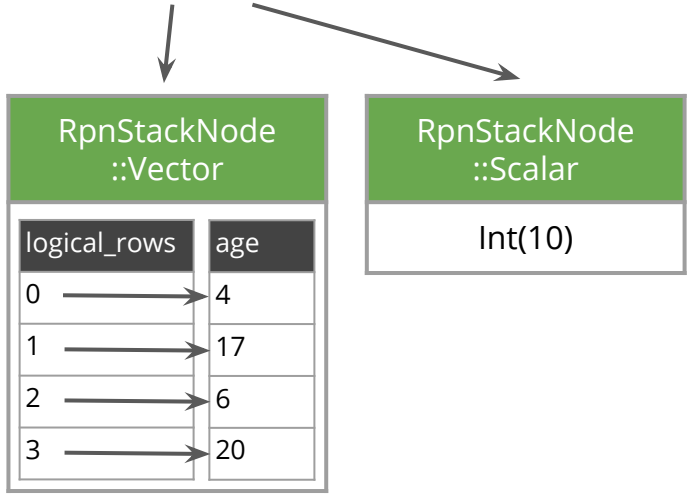
故从数据栈上弹出两个值，分别作为调用的两个参数

表达式向量化计算

表达式

计算数据栈

gt_int_fn(lhs, rhs: RpnStackNode)



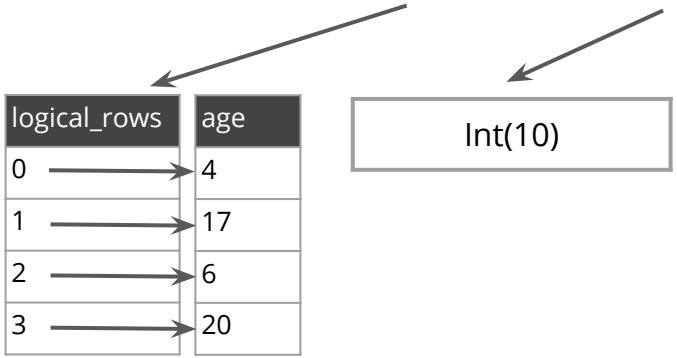
表达式向量化计算

PS：此处流程仅供便于理解，实际代码中使用泛型展开

表达式

计算数据栈

gt_int_vector_scalar(lhs: Vec<Int>, rhs: Int)



实际代码中所有类型的所有比较都使用泛型展开，没有手工实现：

```
#[rpn_fn]
#[inline]
pub fn compare<C>(
  lhs: &Option<C::T>,
  rhs: &Option<C::T>,
) -> Result<Option<i64>>
where
   C: Comparer,
{
   C::compare(lhs, rhs)
}
```

```
let ret = Vec::with_capacity(..);
for idx in logical_rows {
    ret.push(gt_int(lhs[idx], rhs));
}
```

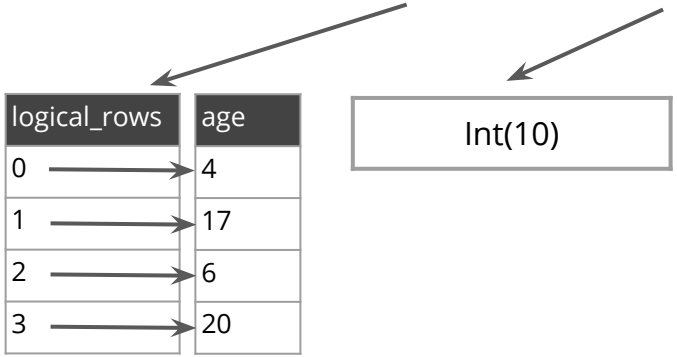
表达式向量化计算

PS：此处流程仅供便于理解，实际代码中使用泛型展开

表达式

计算数据栈

gt_int_vector_scalar(lhs: Vec<Int>, rhs: Int)




```
let ret = Vec::with_capacity(..);
for idx in logical_rows {
    ret.push(lhs[idx] > rhs);
}
```

表达式向量化计算

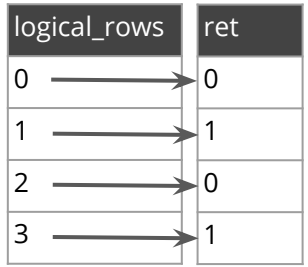
PS：此处流程仅供便于理解，实际代码中使用泛型展开

表达式

计算数据栈

gt_int_vector_scalar(lhs: Vec<Int>, rhs: Int)

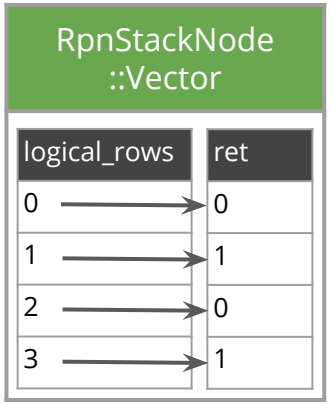
`ret =`



表达式向量化计算

表达式

计算数据栈

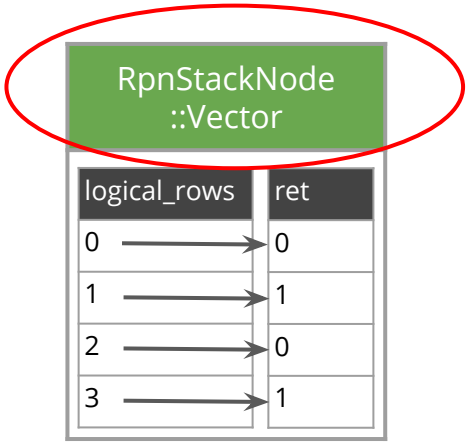


没有更多表达式节点，栈中数据即为表达式计算结果

表达式向量化计算

表达式

计算数据栈



PS: 计算结果也可能是 `RpnStackNode::Scalar`

表达式向量化计算

整个表达式的输出可能是单值(Scalar)而不是列值(Vector)：

- 计算为 Vector：ColumnRef(date)
- 计算为 Vector：ColumnRef(date), Constant("%Y"), FnCall(DateFormat)
- 计算为 Scalar：Constant("%Y")

若表达式是常量表达式，则输出的是单值，否则输出为列值。

更复杂的表达式

```
WHERE DATE_FORMAT(date, "%Y") = "2019"
```

RPN 表达式为：

```
ColumnRef(date), Constant("%Y"), FnCall(DateFormat), Constant("2019"), FnCall(EqInt)
```

代码结构 (4.0 / master branch)

```
+  components/
    +  tidb_query_vec_expr    向量化表达式计算
    +  tidb_query_vec_executors  向量化算子和 DAG Request Handler
    +  tidb_query_vec_aggr    向量化聚合
```

课程作业

分值：1000 + 500 + 500

题目描述：

TiDB: 整合输出 TiKV Coprocessor 提供的 ScanDetailV2 信息 tidb#20255

课程作业

分值：1500 + 1500

题目描述：

TiDB: 优化 CLUSTER_SLOW_QUERY 表逆序扫表性能 tidb#20236

课程作业

分值：1200 + 1000

题目描述：

TiKV: 为 KvGet, KvBatchGet 支持返回 ScanDetailV2 (1200 分) tikv#8756

TiDB: 整合输出 KvGet, KvBatchGet 中提供的 ScanDetailV2 (1000 分) tidb#20256

课程作业

分值：3000

题目描述：

TiKV (RocksDB): 优化 PerfContext 性能 rust-rocksdb#543

课程作业

分值：3000

题目描述：

TiKV: 实现 Coprocessor 流水线化算子优化 tikv#8752

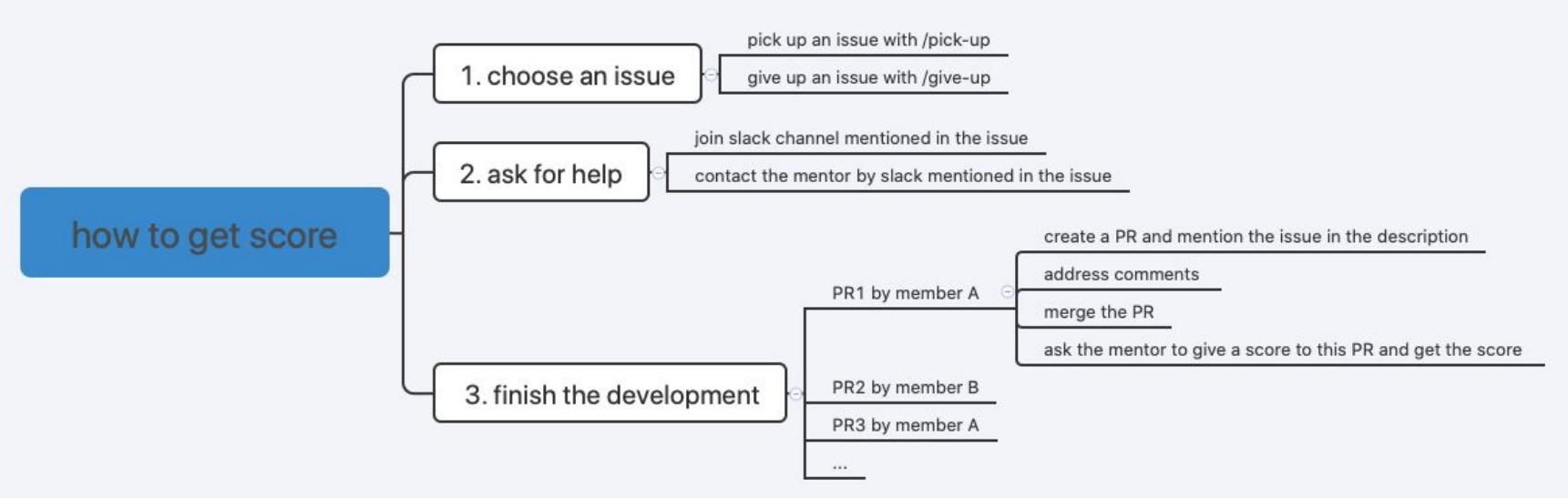
课程作业

关于更多题目请查阅 HPTC 或 High-Performance 标签下的 issue：

TiDB
https://github.com/pingcap/tidb/labels/hptc
https://github.com/pingcap/tidb/labels/high-performance

TiKV
https://github.com/tikv/tikv/labels/hptc
https://github.com/tikv/tikv/labels/high-performance

作业认领方式


how to get score
1. choose an issue
pick up an issue with /pick-up
give up an issue with /give-up
2. ask for help
join slack channel mentioned in the issue
contact the mentor by slack mentioned in the issue
3. finish the development
PR1 by member A
create a PR and mention the issue in the description
address comments
merge the PR
ask the mentor to give a score to this PR and get the score
PR2 by member B
PR3 by member A
...

作业认领相关命令

/pick-up

- 作用：issue 评论中回复认领 issue，如果是多人协作完成，派一个代表 pick 即可，对外只是标记这个任务已经有人在处理了。pick-up 完毕后，该 issue 会自动打上 picked 标签
- 权限：anyone
- 认领后：七天无动态认为该同学无法完成该任务，将自动 give-up

/give-up

- 作用：issue 评论中回复放弃当前认领的任务，give up 完毕后，该 issue 的 picked 标签会被移除
- 权限：当前挑战者

关联 PR 和 issue，PR 描述中按照以下方式之一关联 issue

- Issue Number: close #xxx
- Issue Number: #xxx

课程答疑与学习反馈

扫描左侧二维码填写报名信息，加入课程学习交流群，课程讲师在线答疑，学习效果 up up！

更多课程

想要了解更多关于 TiDB 运维、部署以及 TiDB 内核原理相关课程，可以扫描左侧二维码，或直接进入 http: //university.pingcap.com 查看

Thank you!