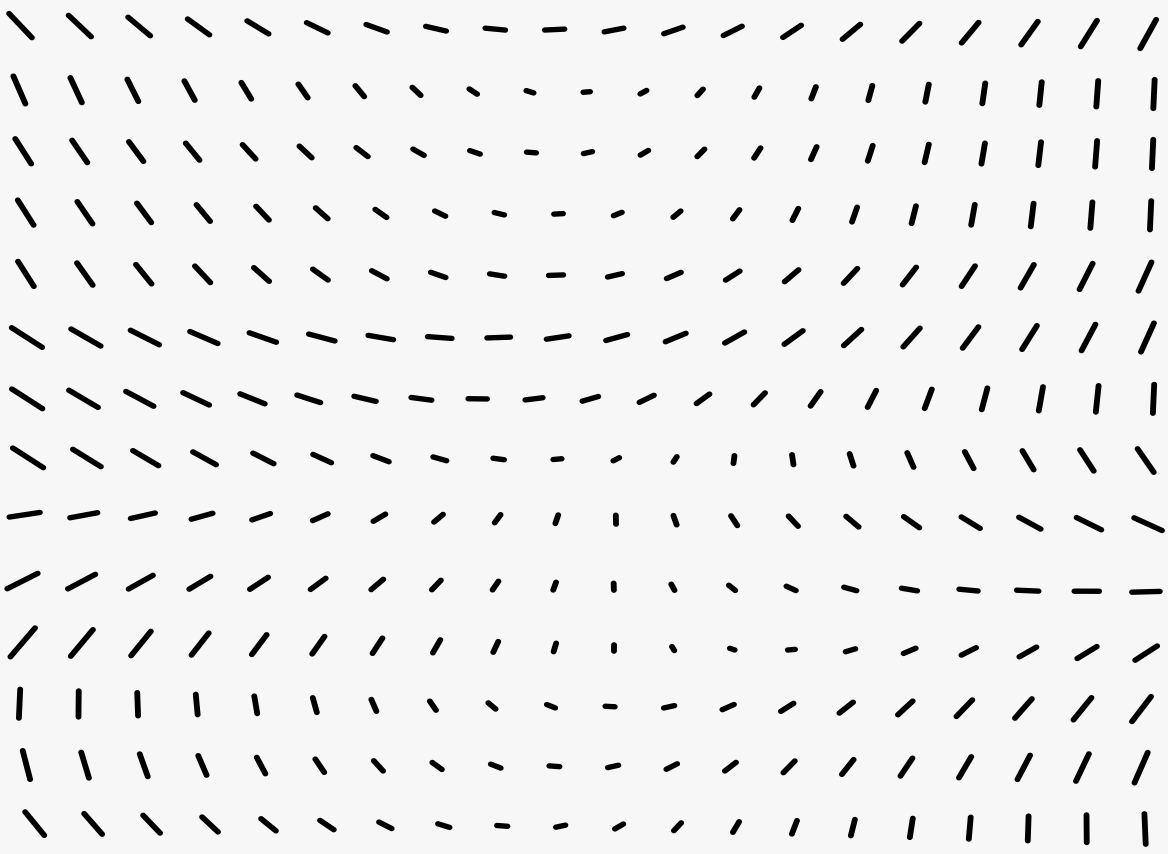
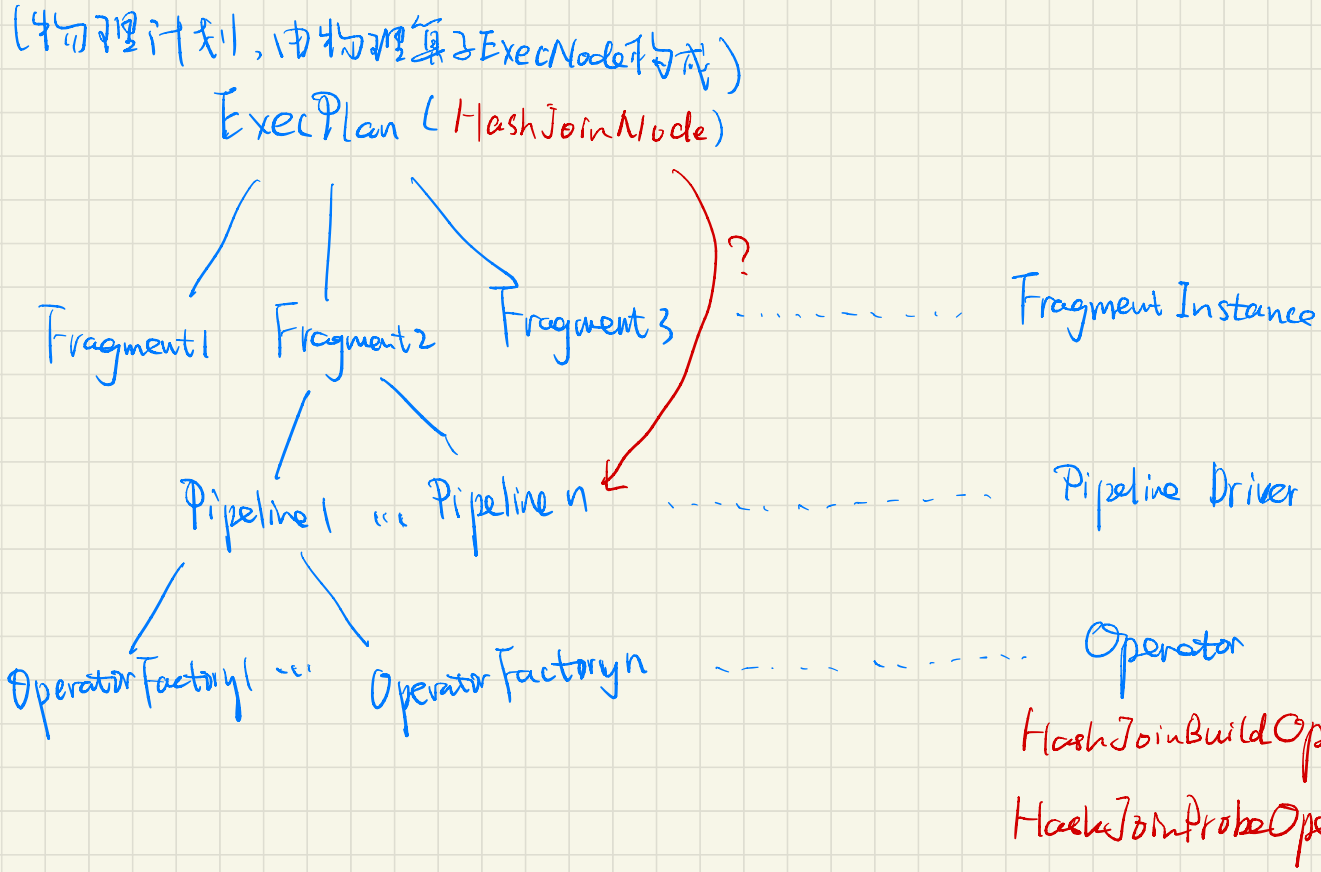
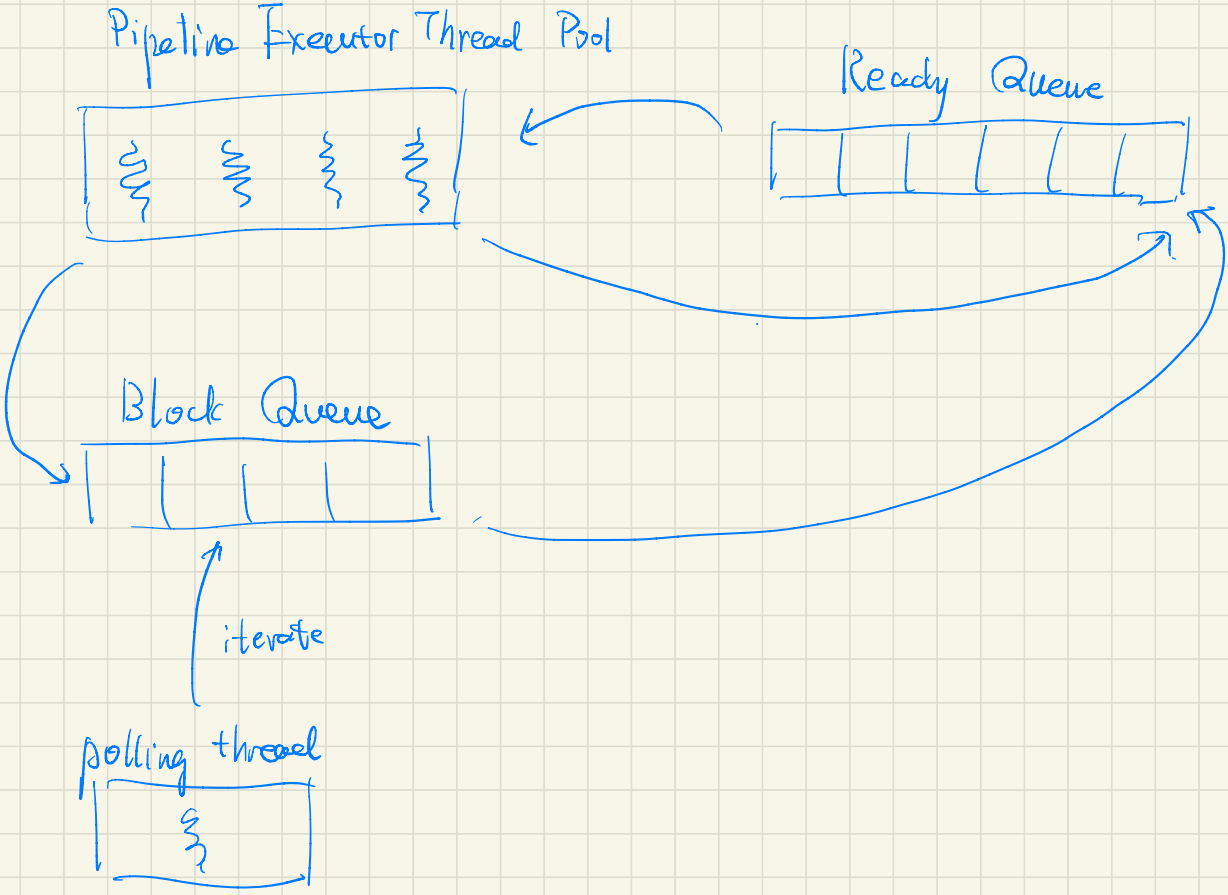



(物理计划，由物理算子ExecNode构成)
ExecPlan (HashJoinNode)
?
Fragment1
Fragment2
Fragment3
Fragment Instance
Pipeline1 ... Pipeline n
Pipeline Driver
OperatorFactory1 ... OperatorFactoryn
Operator
HashJoinBuildOperator
HashJoinProbeOperator
?


Pipeline Executor Thread Pool
Ready Queue
Block Queue
iterate
polling thread

BE ← Fragment — FE

↓

创建 Fragment Executor

↓

初始化 Fragment Instance

↓

将 Fragment 拆分成 Pipeline ⇒ HashJoinNode::decompose_to_pipeline()

↓

根据 dop 创建 Pipeline Driver

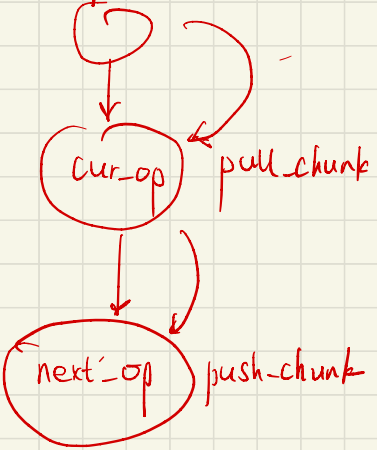
通过 Operator Factory 创建出 Operator

↓

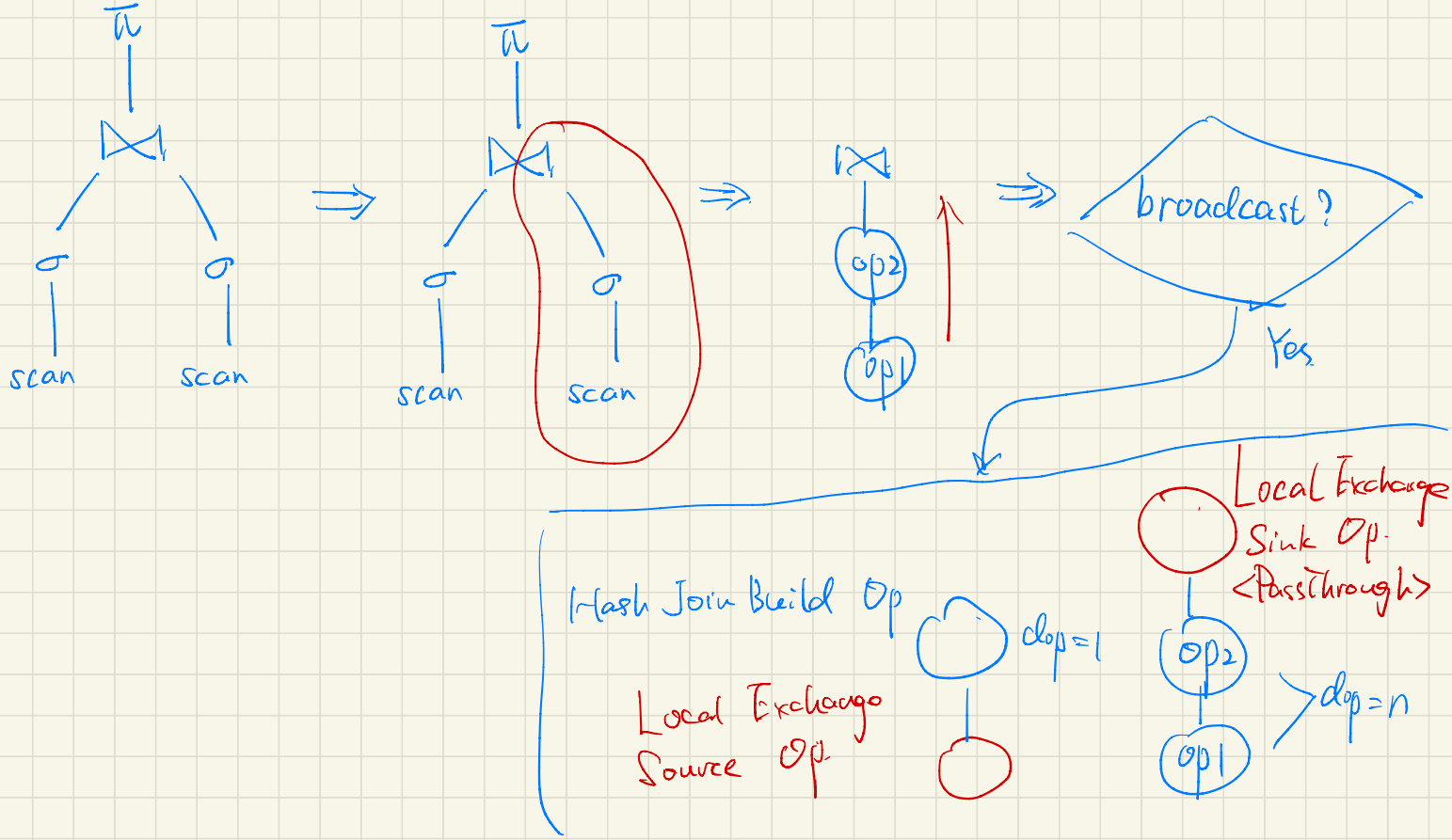
向 Pipeline Executor Thread Pool 提交 Pipeline Driver 运行.

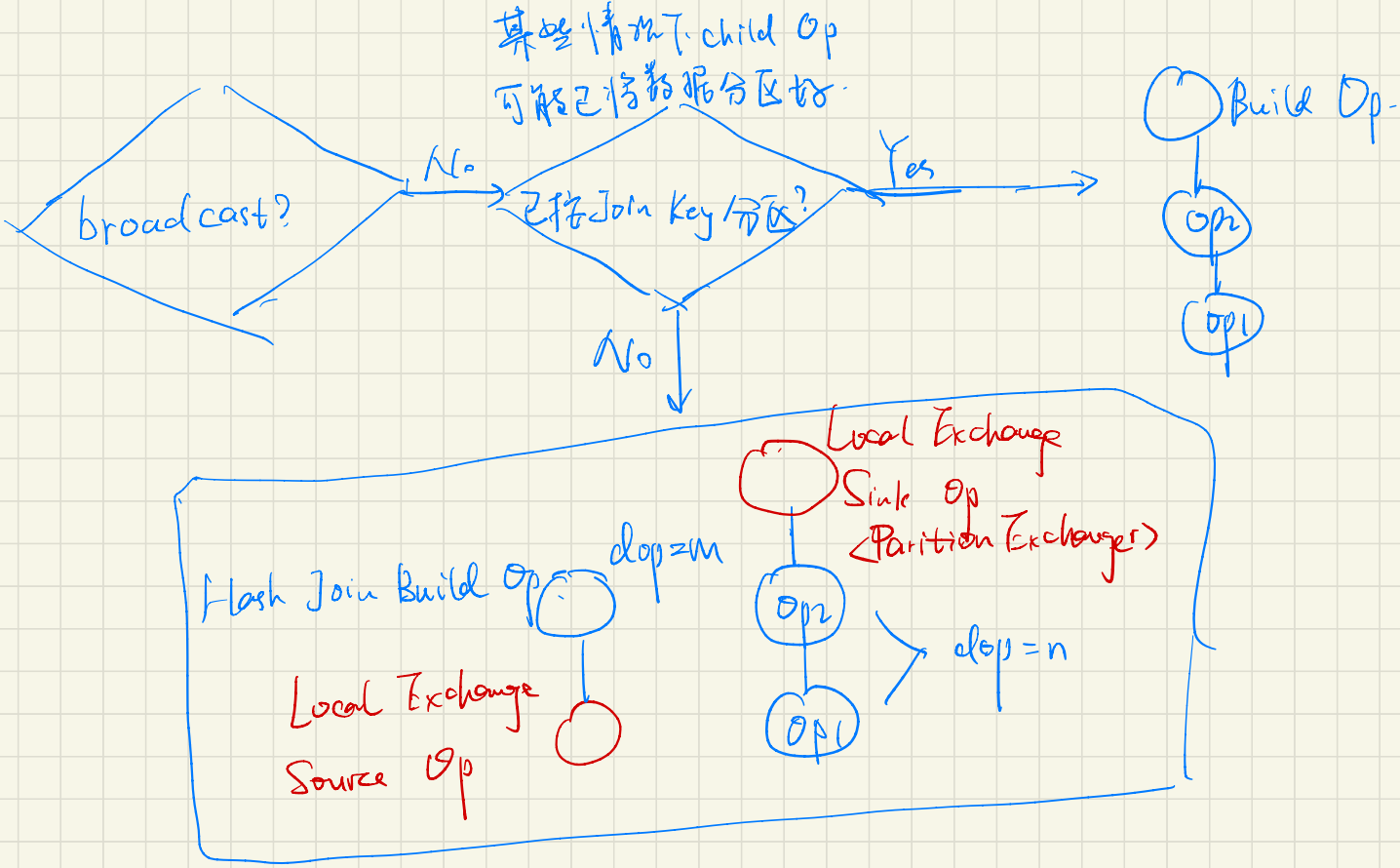
⇒ current_op:: pull_chunk

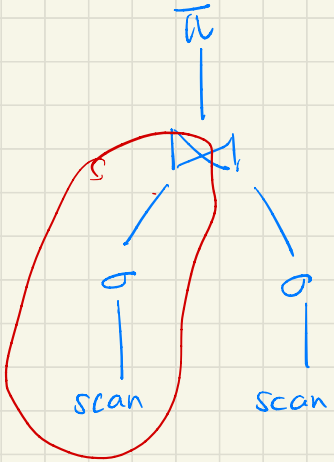
next_op :: push_chunk



HashJoinNode :: decompose_to_pipeline




某些情况下 child Op
可能已经按照分区好
broadcast?
No
已按Join Key分区?
Yes
No
Build Op
Op2
Op1
Local Exchange
Sink Op
<Partition Exchanger>
Hash Join Build Op
dop=m
Local Exchange
Source Op
Op2
Op1
dop=n

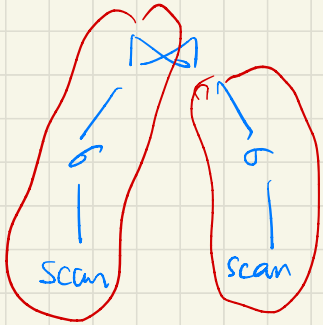


Probe 侧的 Pipeline 扫描方式与 Build 侧类似，也会插入一层 Local Exchange Operator

HashJoinBuildOperator 一定是一条 Pipeline 的最后一个 Operator，因此没有实现 pull_chunk

而 HashJoinProbeOperator 可以处于一条 Pipeline 的任意位置，因此同时实现了 pull_chunk 和 push_chunk

HashJoinProbeOperatorFactory → HashJoinerFactory ← HashJoinBuildOperatorFactory



build 阶段 dop = m

⇒ 创建 m 个 PipelineDriver → 创建 m 个 BuildOperator → 创建 m 个 HashJoiner

HashJoiner
- HashJoinBuilder
- HashJoinProber
- HashTable

x m

if (m == 1) broadcast

probe 阶段 dop = n

创建 n 个 PipelineDriver → 创建 n 个 ProbeOperator → if (m == n), 直接复用

else 创建 n 个 HashJoiner, 同时获取到 build 时创建的 m 个 HashJoiner

m = 5: [0, 1, 2, 3, 4]

n = 10: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

HashJoinBuildOperator

push_chunk(chunk)

↓

HashJoiner::append_chunk(chunk)

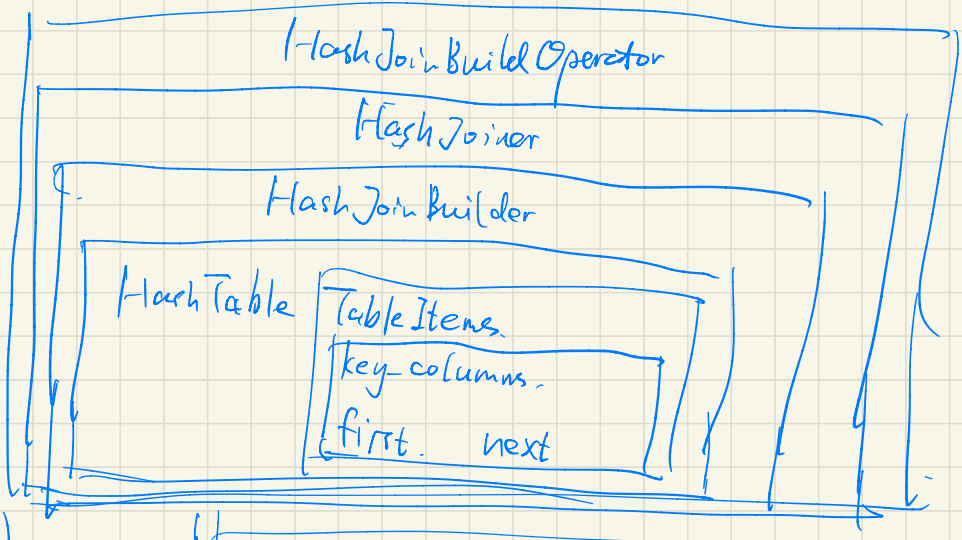
↓

HashJoinBuilder::append_chunk(chunk)

↓

prepare_build_key_columns(&key_columns)

ht.append_chunk(chunk, key_columns)



set_finishing()

↓

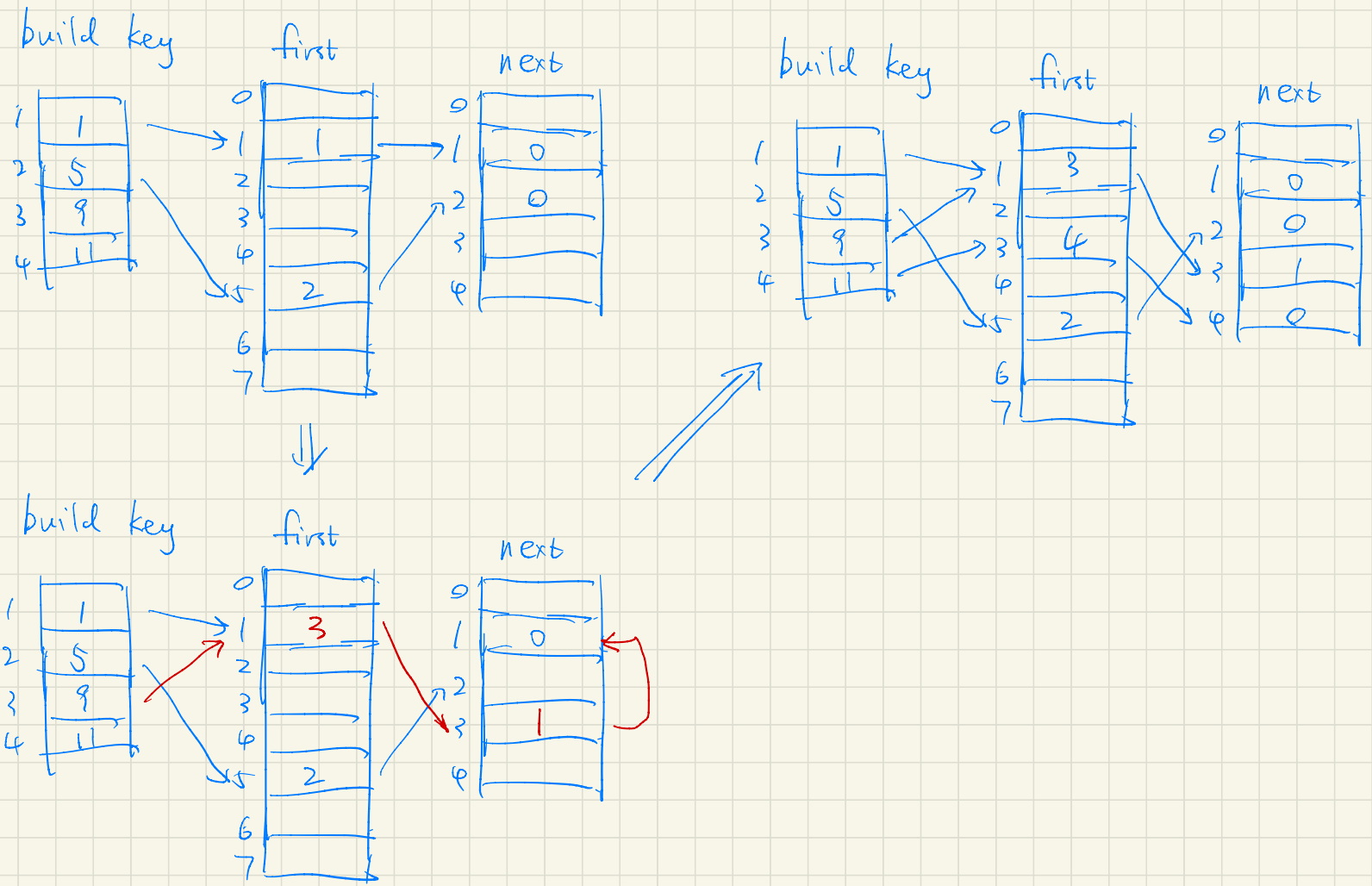
HashJoiner::build_ht()

↓

hashJoinBuilder::build()

↓

HashTable::build()


build key
first
next
1
5
9
11
0
1
2
3
4
5
6
7
build key
first
next
1
5
9
11
3
4
2
build key
first
next
1
5
9
11
3
2
0
1

HashJoinProbeOperator

push_chunk(chunk)

↓

HashJoiner::push_chunk(chunk)

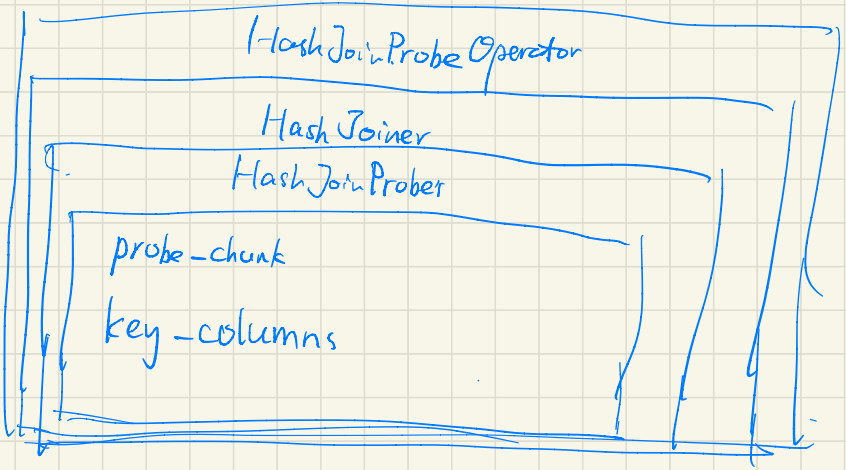
↓

HashJoinProber::push_probe_chunk(chunk)

↓

probe_chunk = chunk

prepare_key_columns



pull_chunk()

↓

HashJoiner::pull_chunk()

↓

HashTable ht

HashJoinProber::probe_chunk(ht)

↓

ht→probe()

不同类型的 hash map 实现

JoinHashMap：单列，定长类型：crc

DirectMappingHashMap：单列，小整数类型：bucket_id = data[1] - MIN_VALUE

FixedSizeHashMap：多列，定长，总长度和为 size 小于 128

先序列化，把多列 key 组合成一个固定长度的 key

再使用 crc 函数

SerializedHashMap：其他没有覆盖到的组合

先计算最大序列化长度，再序列化，最后使用 crc 函数。

针对于不同类型Join的probe实现