2018PostgreSQL中国技术大会

PostgreSQL Parallel Query

Liu Dongming

lingce.ldm@alibaba-inc.com

Alibaba Cloud Database Technology

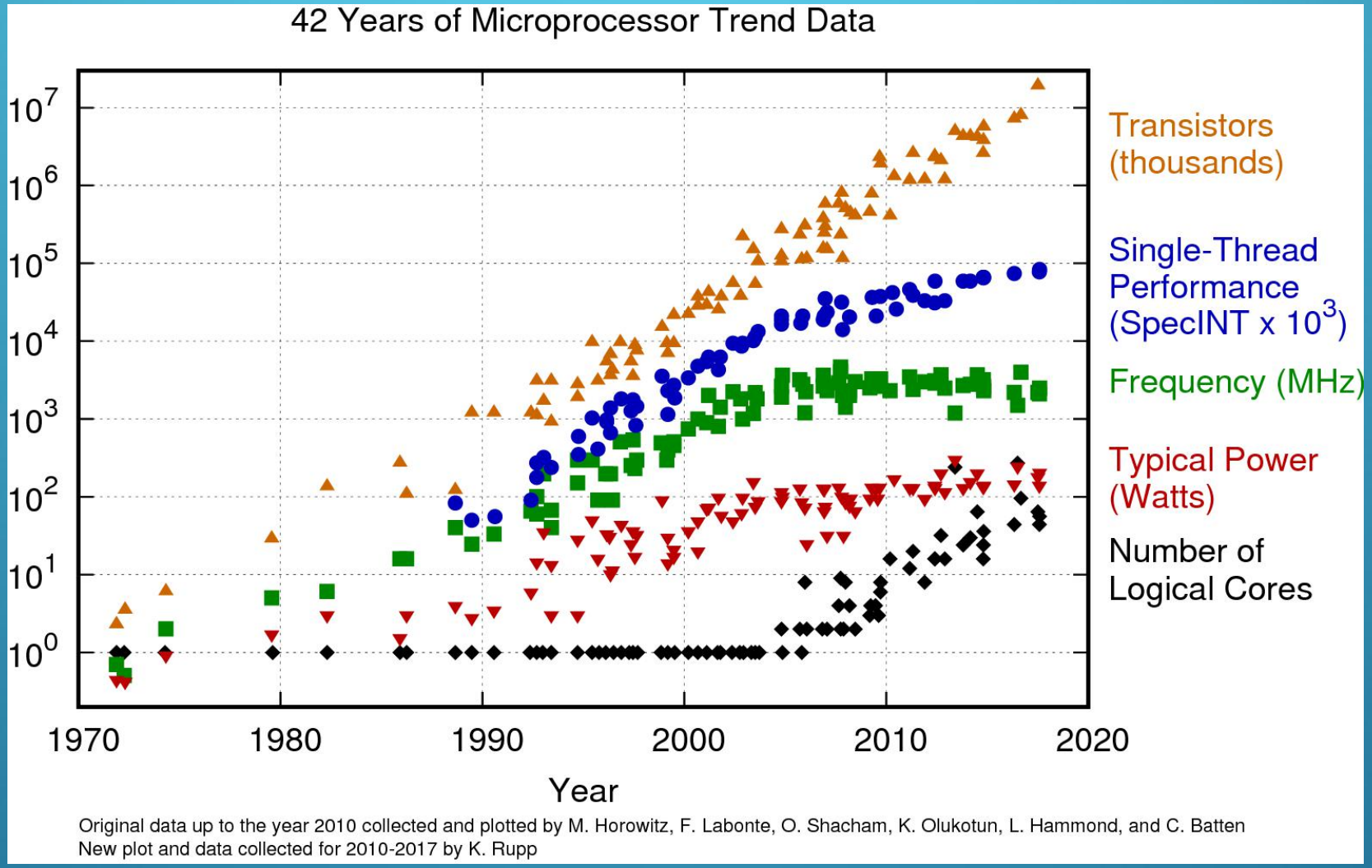
About Me

- Liu Dongming (刘东明)
- DRDS / PostgreSQL
- Alibaba Cloud PostgreSQL Group

Parallel Features

- PostgreSQL 9.4, 9.5 [2014, 2015]
  - Backgound workers
  - Dynamic shared memory(DSM)
  - Shared memory queues
- PostgreSQL 9.6 [2016]
  - Executor nodes：Gather，Parallel Seq Scan，Partial Aggregate，Finalize Aggregate
- PostgreSQL 10 [2017]
  - Partitions
  - Executor nodes：Gather Merge，Parallel Index Scan，Parallel Bitmap Heap Scan
- PostgreSQL 11 [2018]
  - Executor nodes：Parallel Append，Parallel Hash Join
  - Planner：Partition-wise joins
  - Parallel Create Index

The Free Lunch Is Over



https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/
https://en.wikipedia.org/wiki/Herb_Sutter#The_Free_Lunch_Is_Over

Parallel Database System

Parallel Database Systems: The Future of High Performance Database Systems 1992

Authors: David Dewitt and Jim Gray

Why Parallel Databases?

- Relational Data Model – Relational queries are ideal candidates for parallelization
- Multiprocessor systems using inexpensive microprocessors provide more power and scalability than expensive mainframe counterparts

- Shared-memory – All processors have equal access to a global memory and all disks
- Shared-disk – Each processor has its own private memory, but has equal access to all disks
- Shared-nothing – Each processor has its own private memory and disk(s)

For Example

```
SELECT  COUNT(*)
FROM    people
WHERE   inpgconn2018 = 'Y';
```

EXPLAIN ANALYZE SELECT COUNT(*) FROM people WHERE atpgconn2018 = 'Y';

```
Aggregate  (cost=169324.73..169324.74 rows=1 width=8) (actual time=983.729..983.730 rows=1 loops=1)
  ->  Seq Scan on people  (cost=0.00..169307.23 rows=7001 width=0) (actual time=981.723..983.051 rows=9999 loops=1)
        Filter: (atpgconn2018 = 'Y'::bpchar)
        Rows Removed by Filter: 9990001
 Planning Time: 0.066 ms
 Execution Time: 983.760 ms
```

max_parallel_workers_per_gather = 0

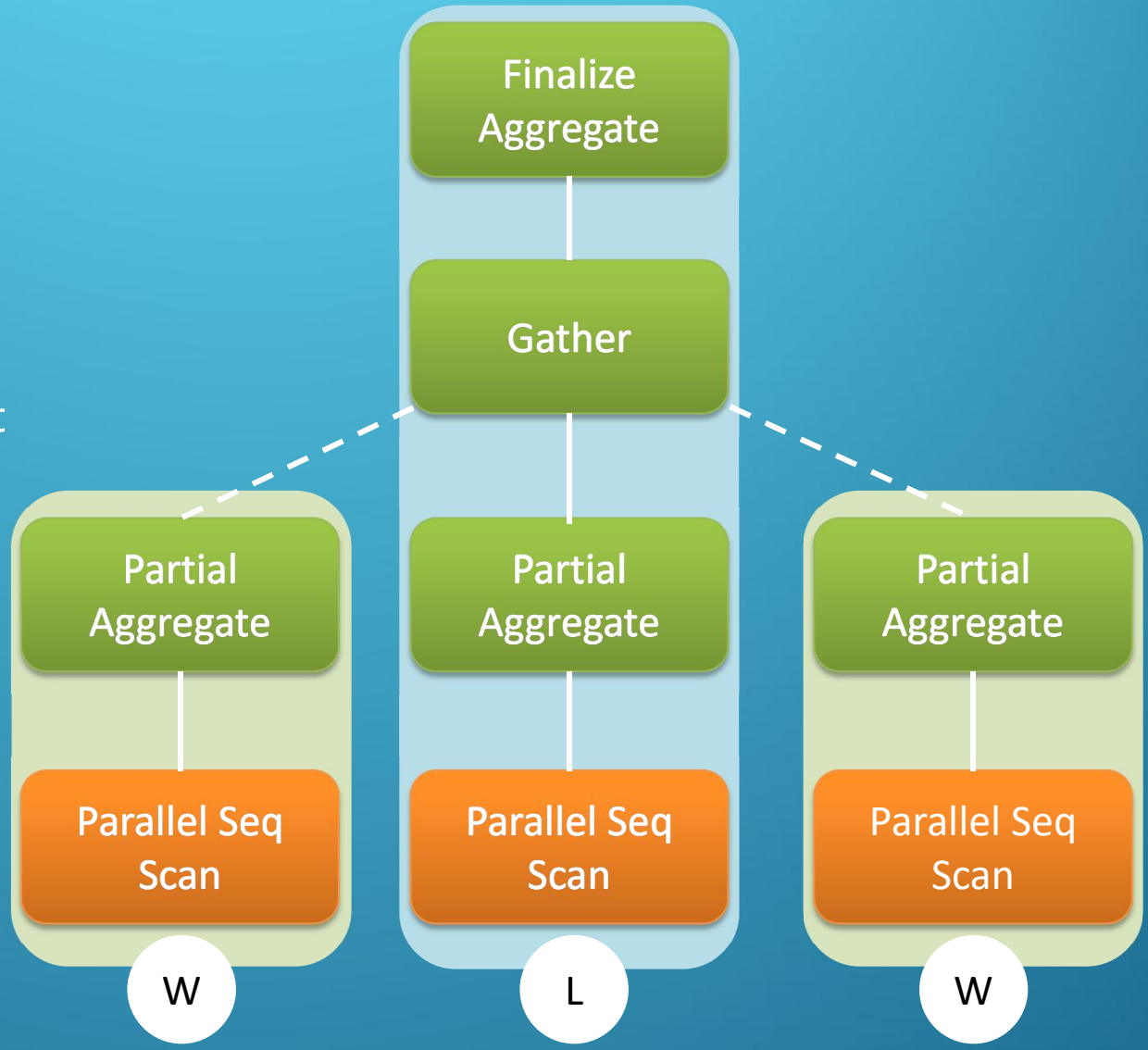
```
Finalize Aggregate  (cost=97389.77..97389.78 rows=1 width=8) (actual time=384.848..384.848 rows=1 loops=1)
  ->  Gather  (cost=97389.55..97389.76 rows=2 width=8) (actual time=384.708..386.486 rows=3 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Partial Aggregate  (cost=96389.55..96389.56 rows=1 width=8) (actual time=379.597..379.597 rows=1 loops=3)
              ->  Parallel Seq Scan on people  (cost=0.00..96382.26 rows=2917 width=0)
                                               (actual time=378.831..379.341 rows=3333 loops=3)
                    Filter: (atpgconn2018 = 'Y'::bpchar)
                    Rows Removed by Filter: 3330000
 Planning Time: 0.063 ms
 Execution Time: 386.532 ms
```

max_parallel_workers_per_gather = 2

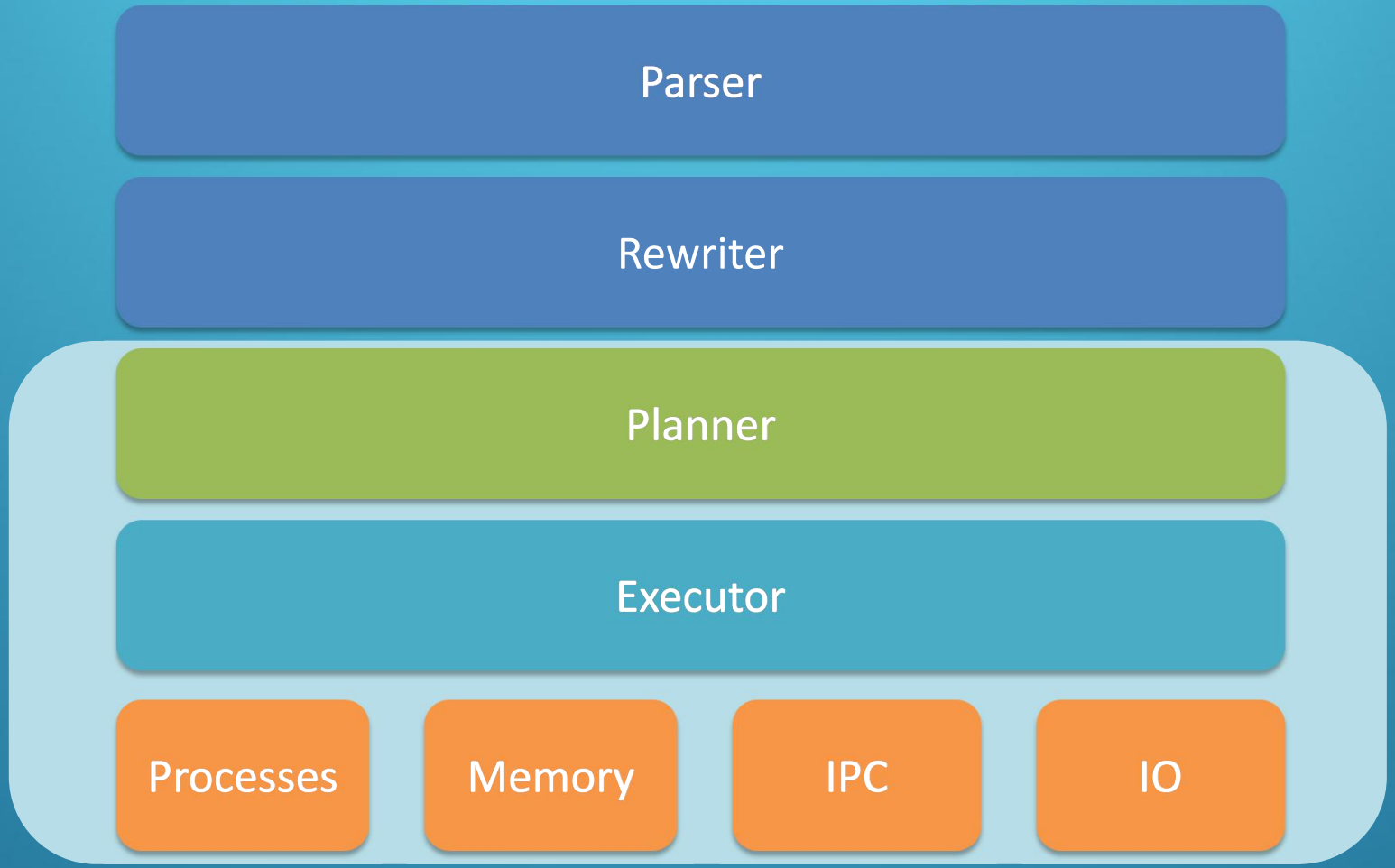
Parallel Plan

Worker(W)： **each** worker runs a copy of the plan fragment beneath of the Gather node.

Leader(L)： leader runs the Gather node and the plan fragment on top of the Gather， may also run the plan fragment beneath of the Gather node.

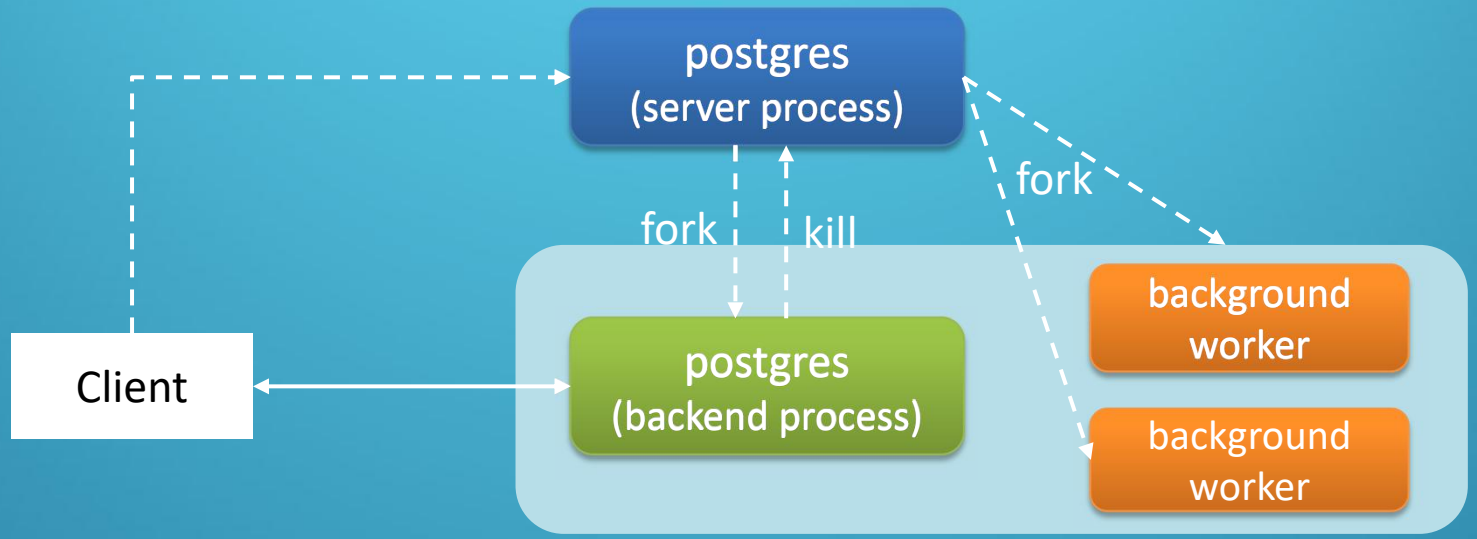


PostgreSQL Query Architecture



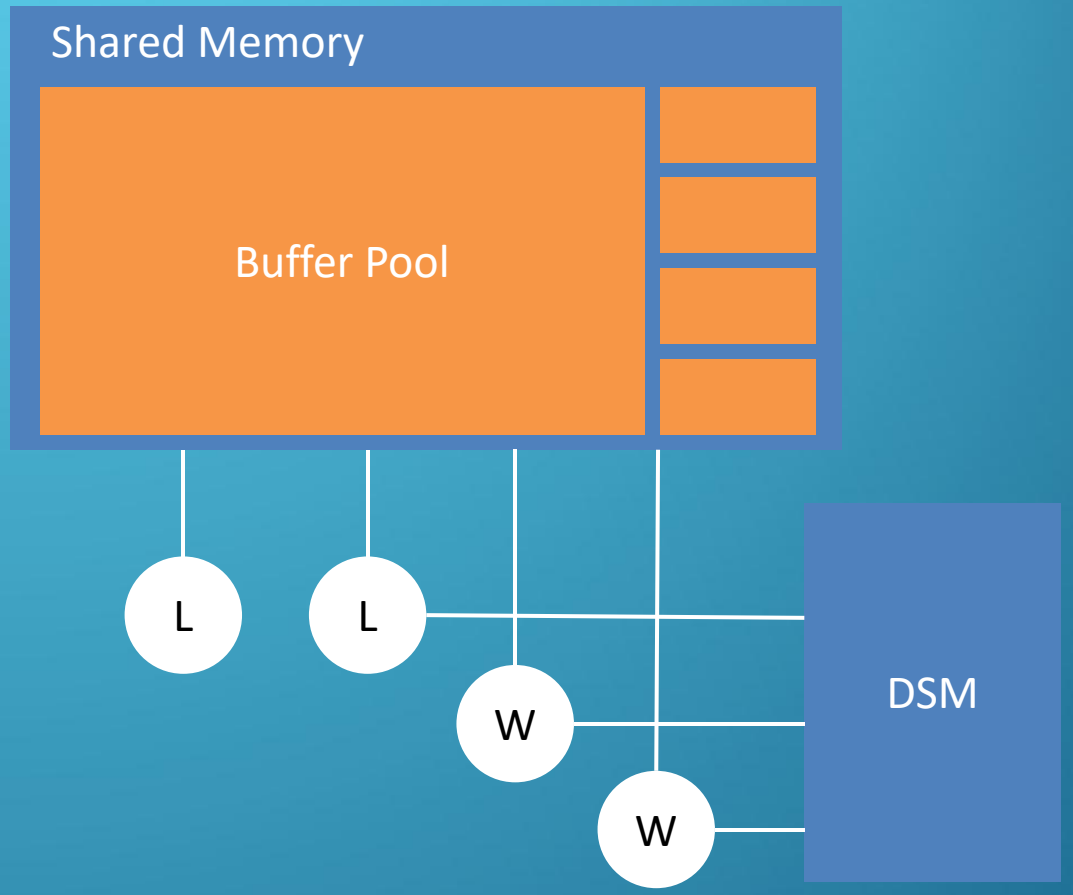
Infrastructure for parallelism

Background worker processes



Dynamic Shared Memory

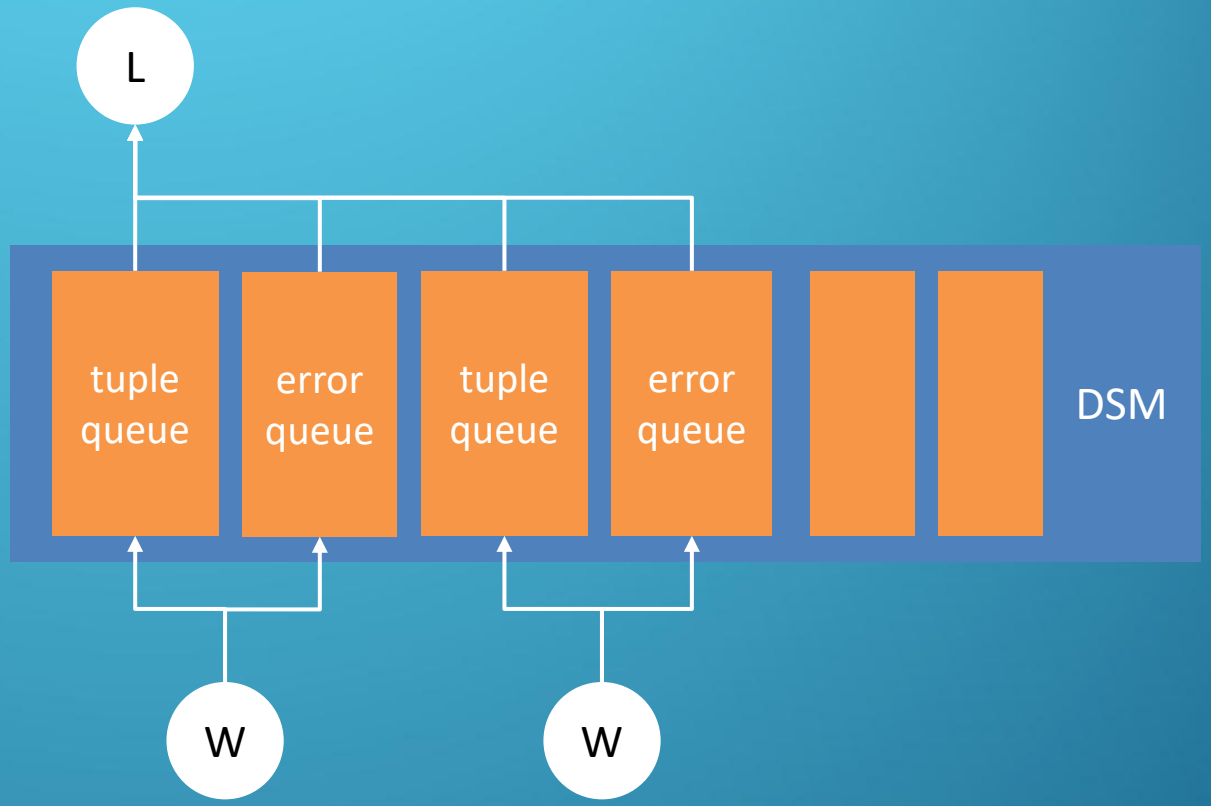
- **Traditionaly,** PostgreSQL has a fixed-size shared memory mapped at the same address in all processes, inherited from the postmaster process.
- For parallel query execution, **dynamic shared memory** segments are used; they are extra shared memory, mapped at an arbitrary address in each backend, and unmapped at the end of the query.



IPC and Message Propagation

Shared memory queues (shm_mq) for control messages and tuples .

If the background worker generates an ERROR, WARNING, or other message, it can send that message to the master, and the master can receive it.



How parallel queries are executed?

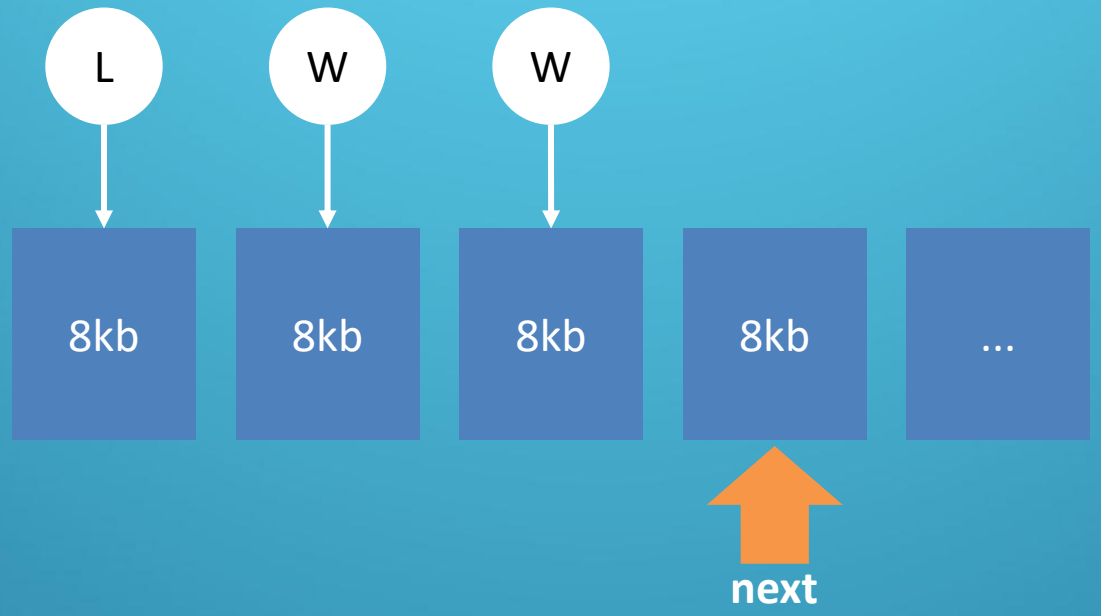
parallel-aware node

Node with **Parallel** prefix can be called **parallel-aware** operators.

Parallel-oblivious node is one where the node is unaware that it is part of a parallel plan.

Parallel Seq Scan	Parallel Index Scan	Parallel Bitmap Heap Scan	Parallel Hash Join
Seq Scan	Index Scan	Bitmap Heap Scan	Hash Join

Parallel Seq Scan

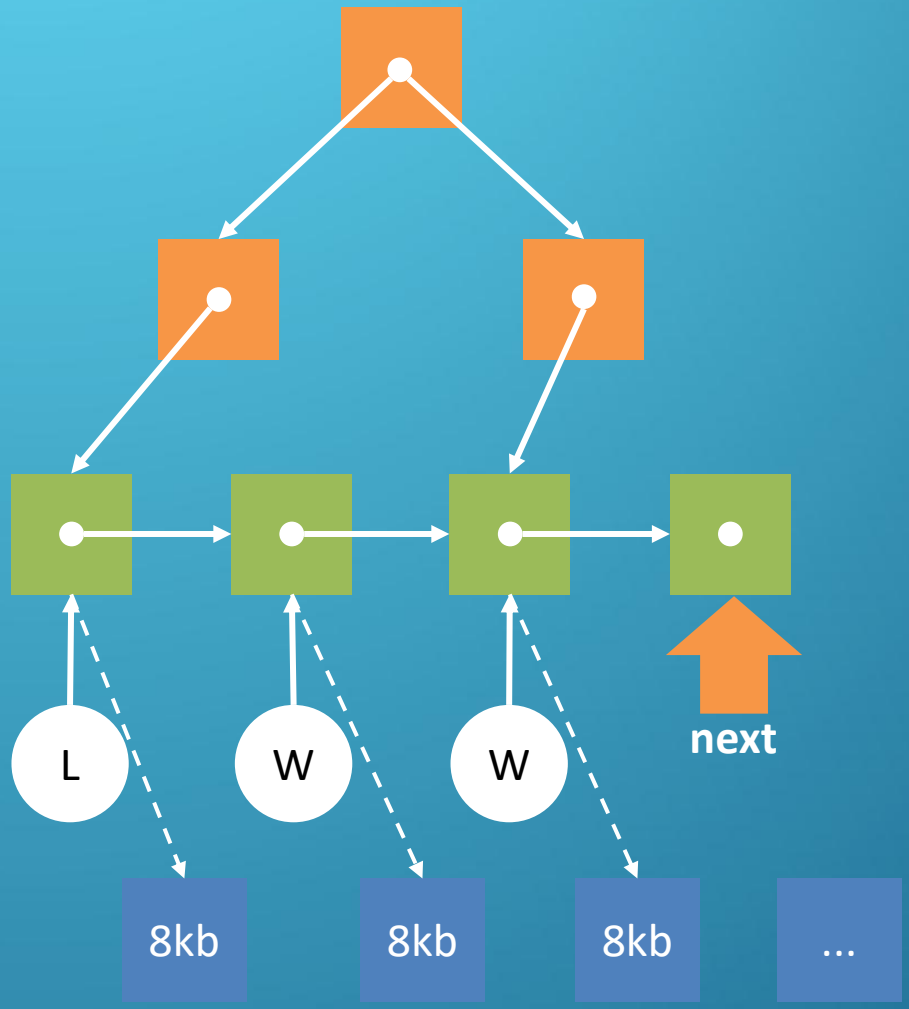


How to allocate work for workers and leader?

Block-By-Block， each process advances a shared **next block** pointer to choose a block to scan.

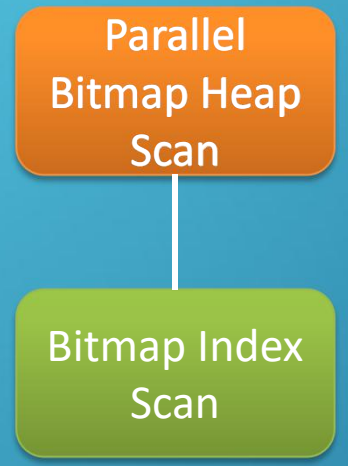
Parallel Index Scan

- Parallel index scans are supported only for **btree** indexes
- Each process advances a shared **next block** pointer to choose an index block and will scan and return all tuples referenced by that block



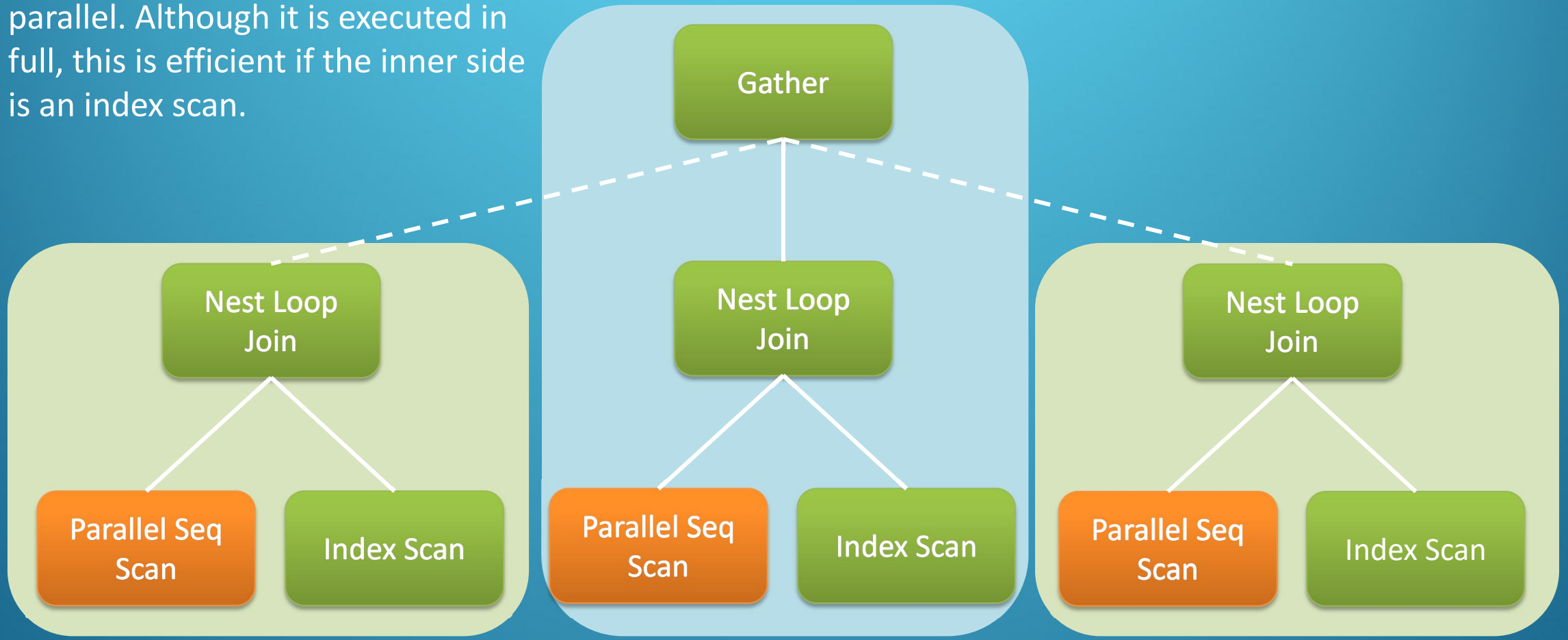
Parallel Bitmap Heap Scan

- Similar to Parallel Seq Scan, but scan only pages that were found to potentially contain interesting tuples
- The bitmap is currently built by a single processes; only the actual Parallel Bitmap Heap Scan is parallel-aware



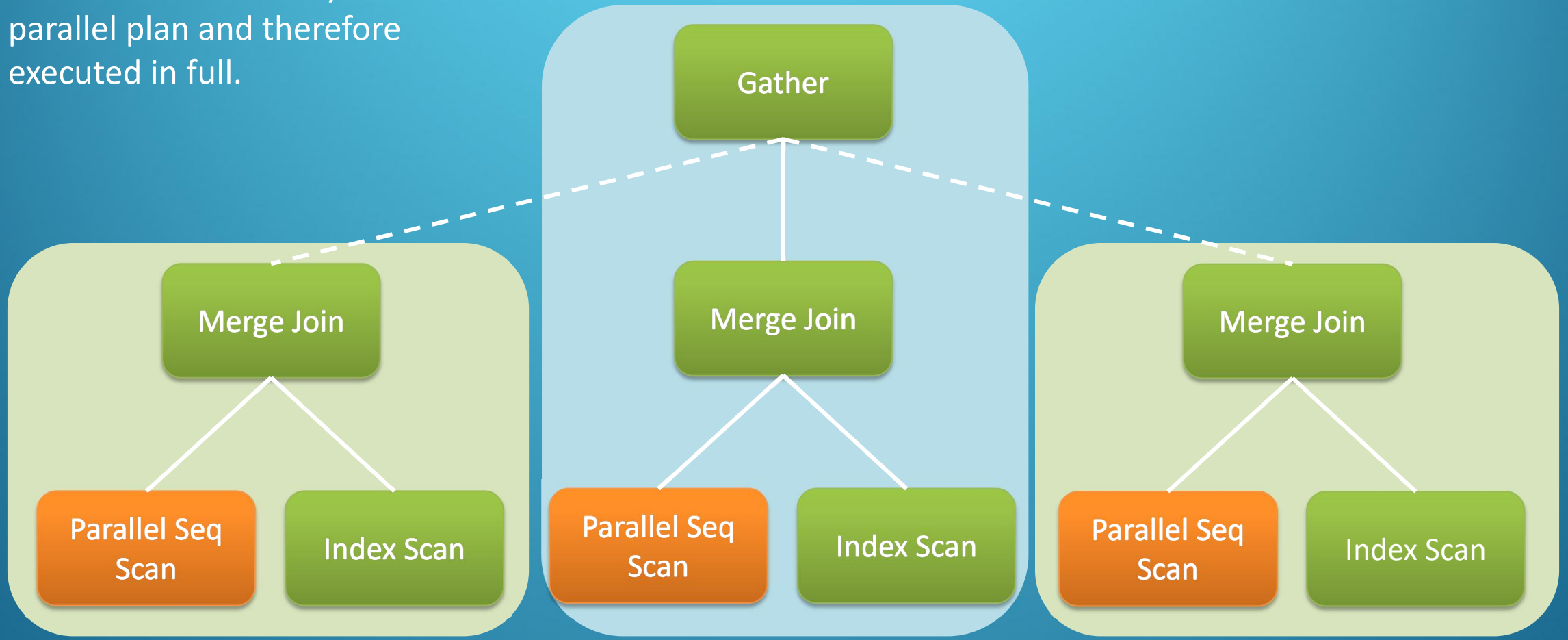
Nest Loop Join

The inner side is always non-parallel. Although it is executed in full, this is efficient if the inner side is an index scan.



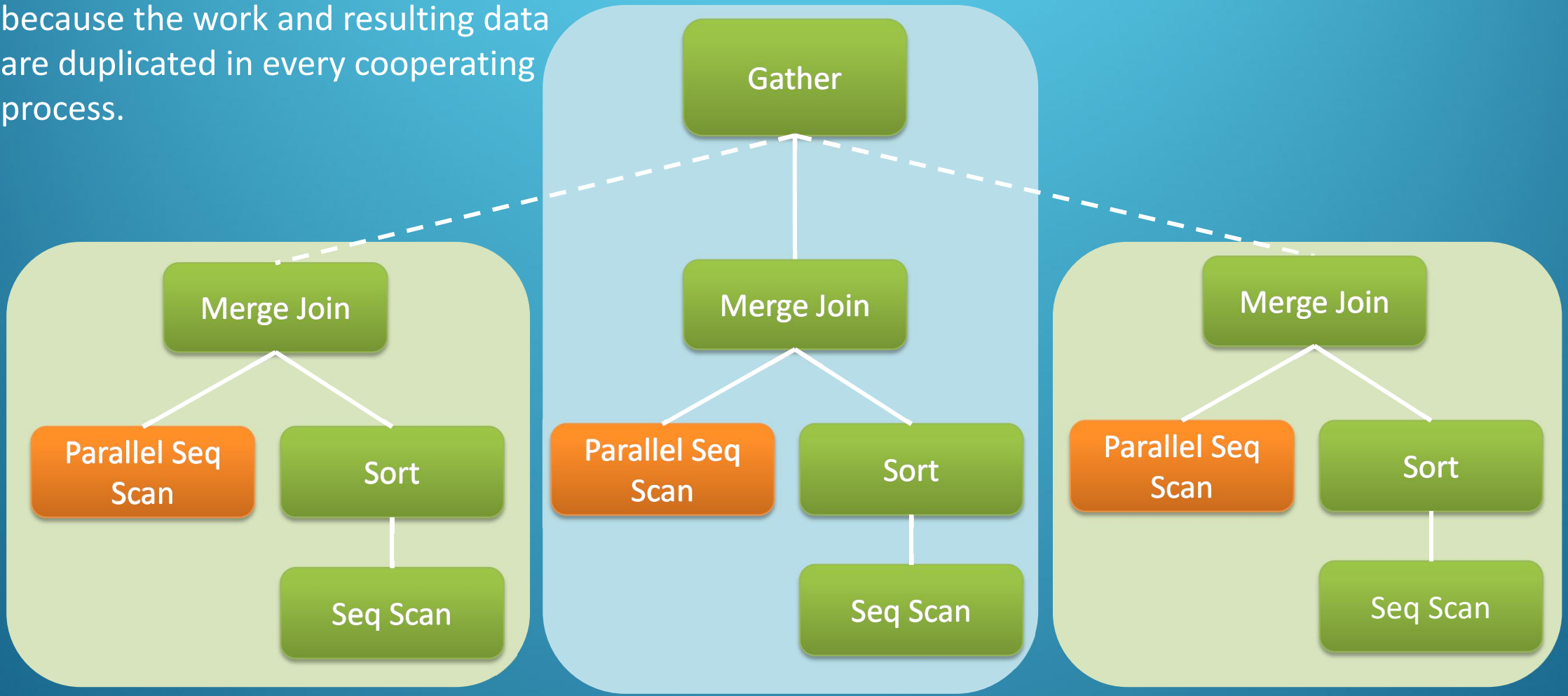
Merge Join

The inner side is always a non-parallel plan and therefore executed in full.



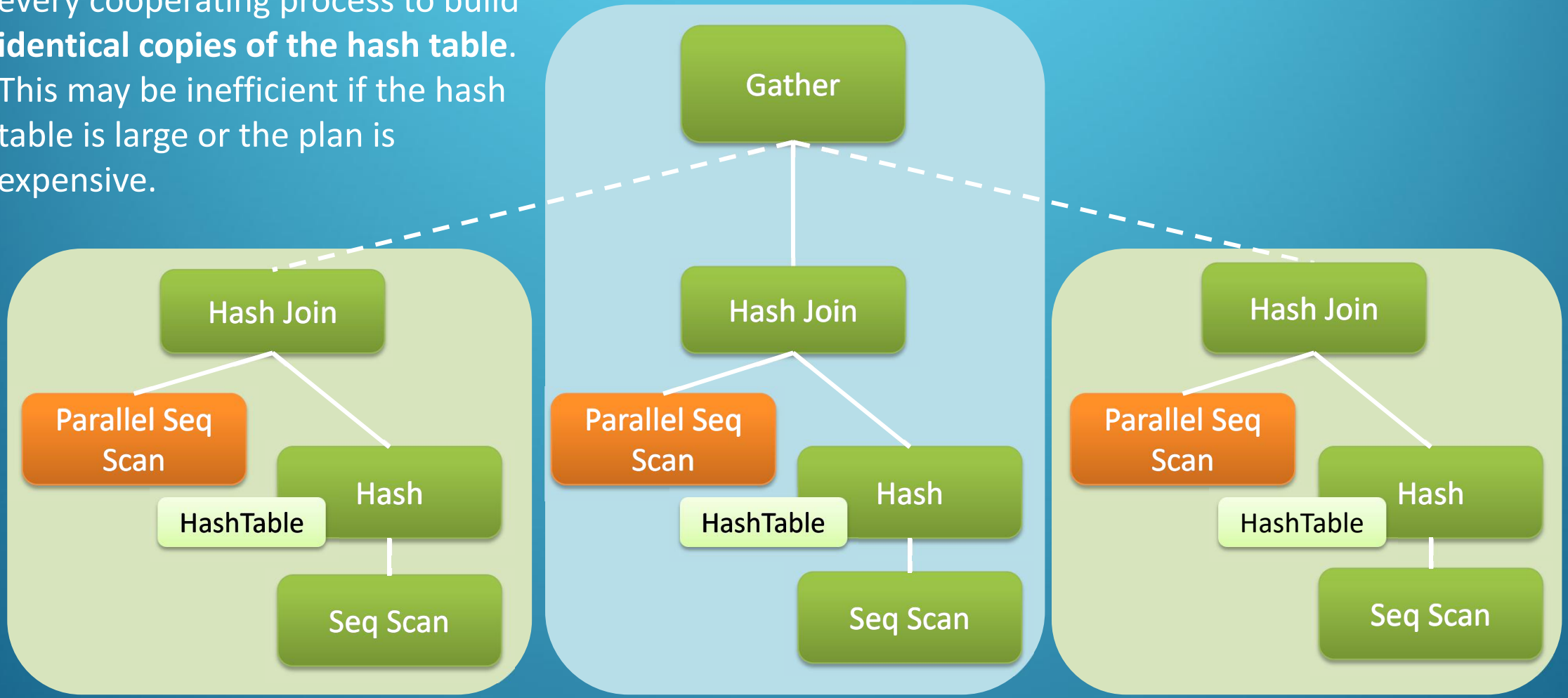
Merge Join

The **merge join** may be inefficient, especially if a sort must be performed, because the work and resulting data are duplicated in every cooperating process.



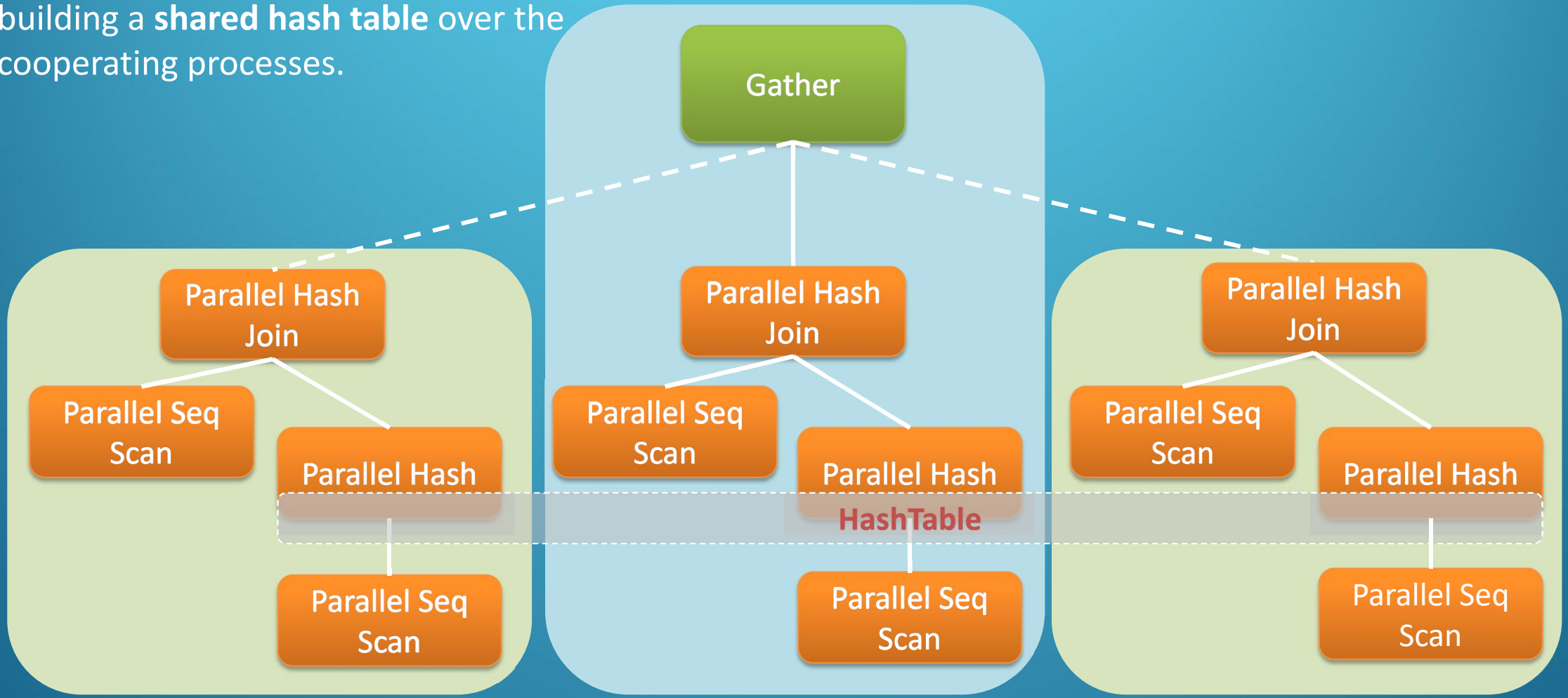
Hash Join

The inner side is executed in full by every cooperating process to build **identical copies of the hash table**. This may be inefficient if the hash table is large or the plan is expensive.

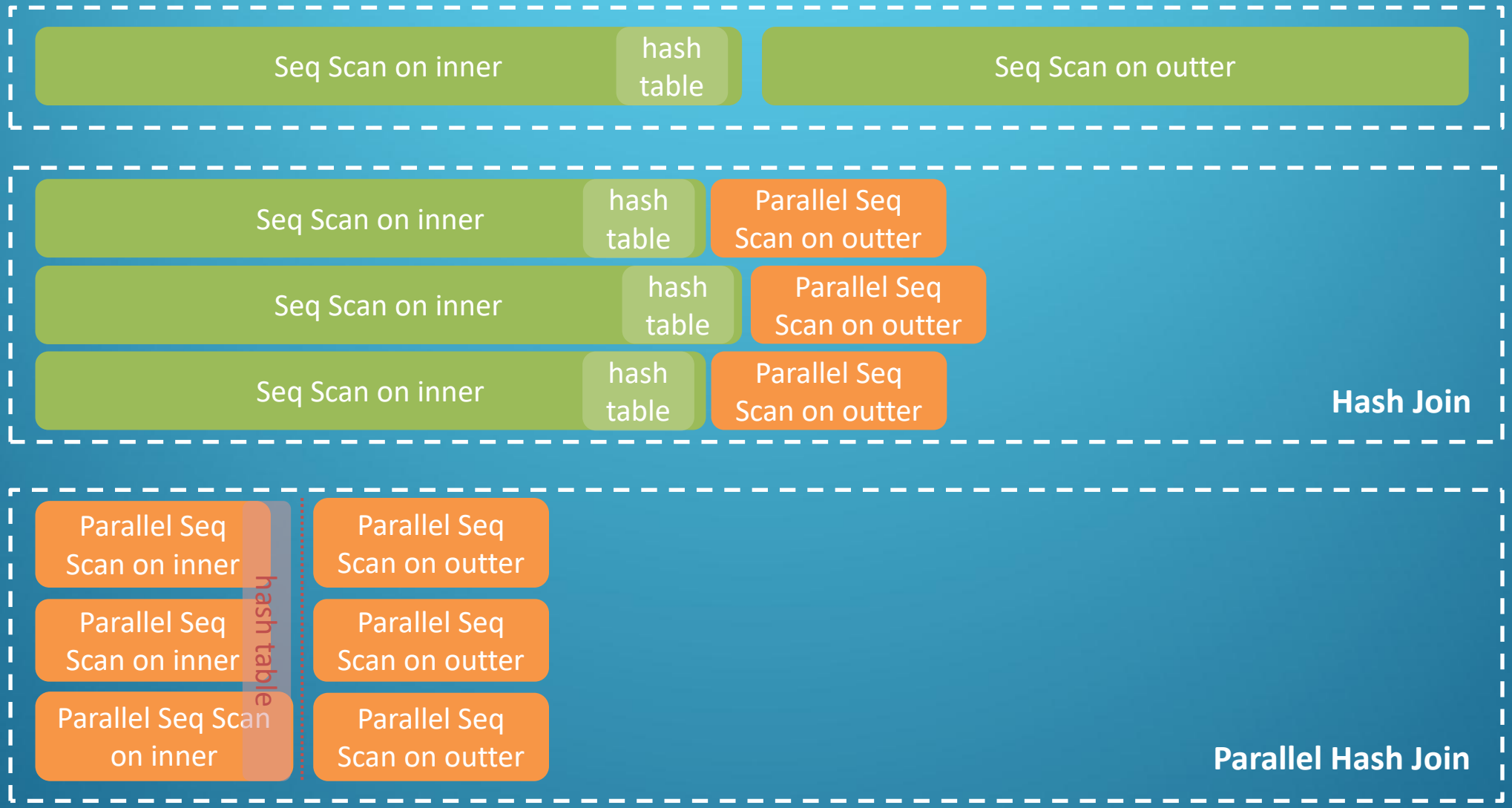


Parallel Hash Join

The ***parallel hash*** divides the work of building a **shared hash table** over the cooperating processes.

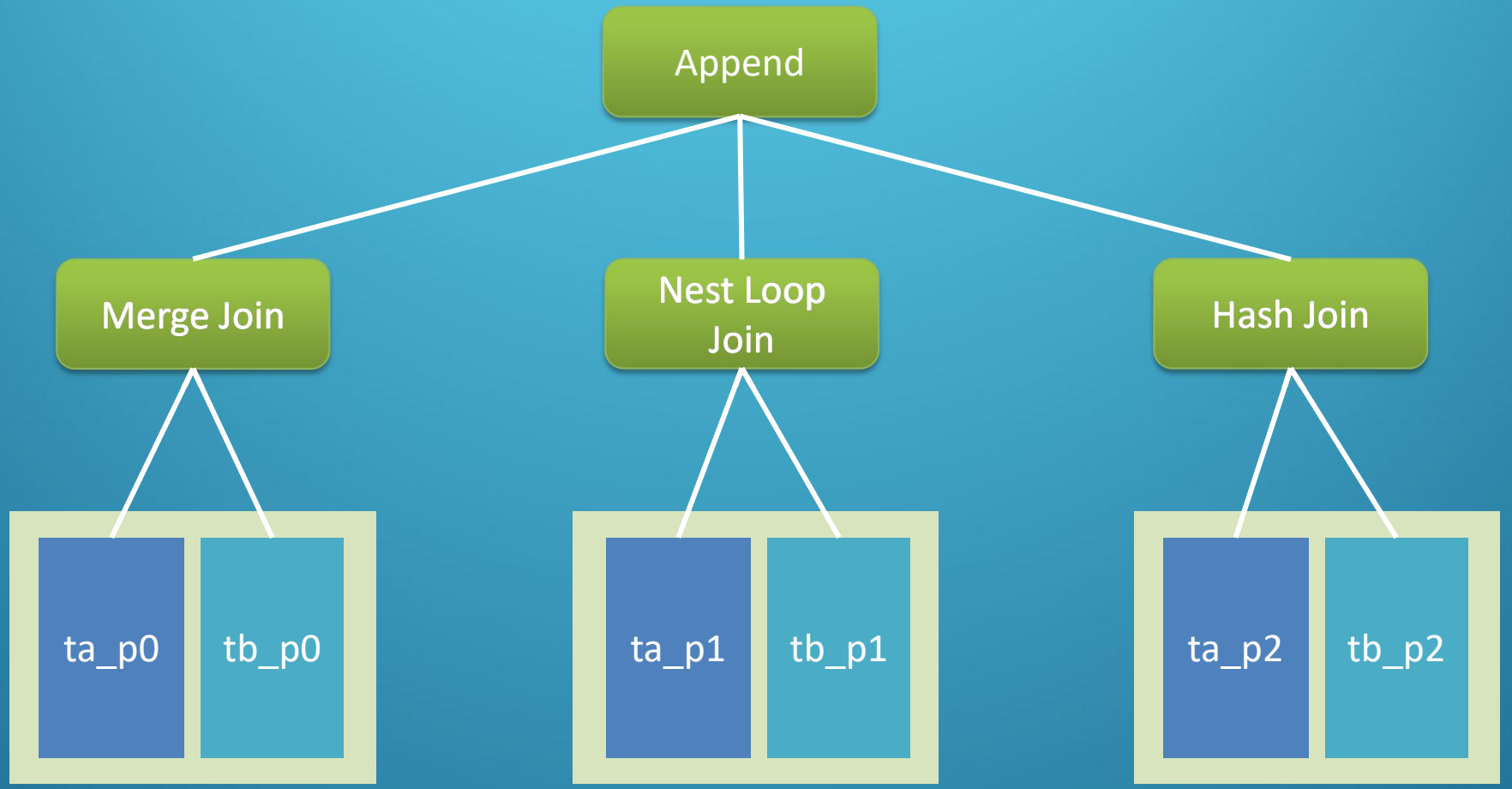


Execution time of different hash join

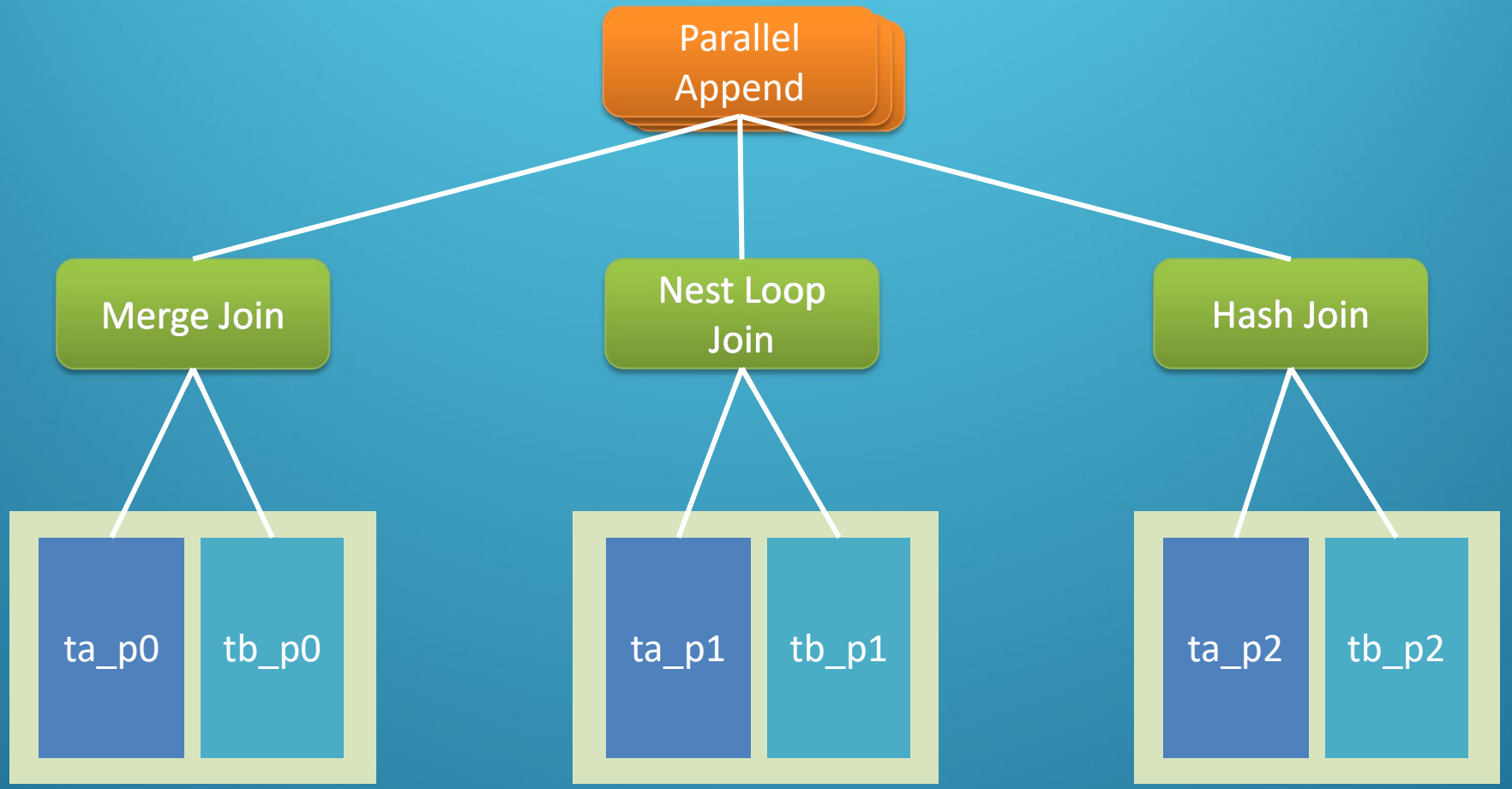


Partition-wise join

Divide and conquer for joins between partitioned table.



Parallel Append

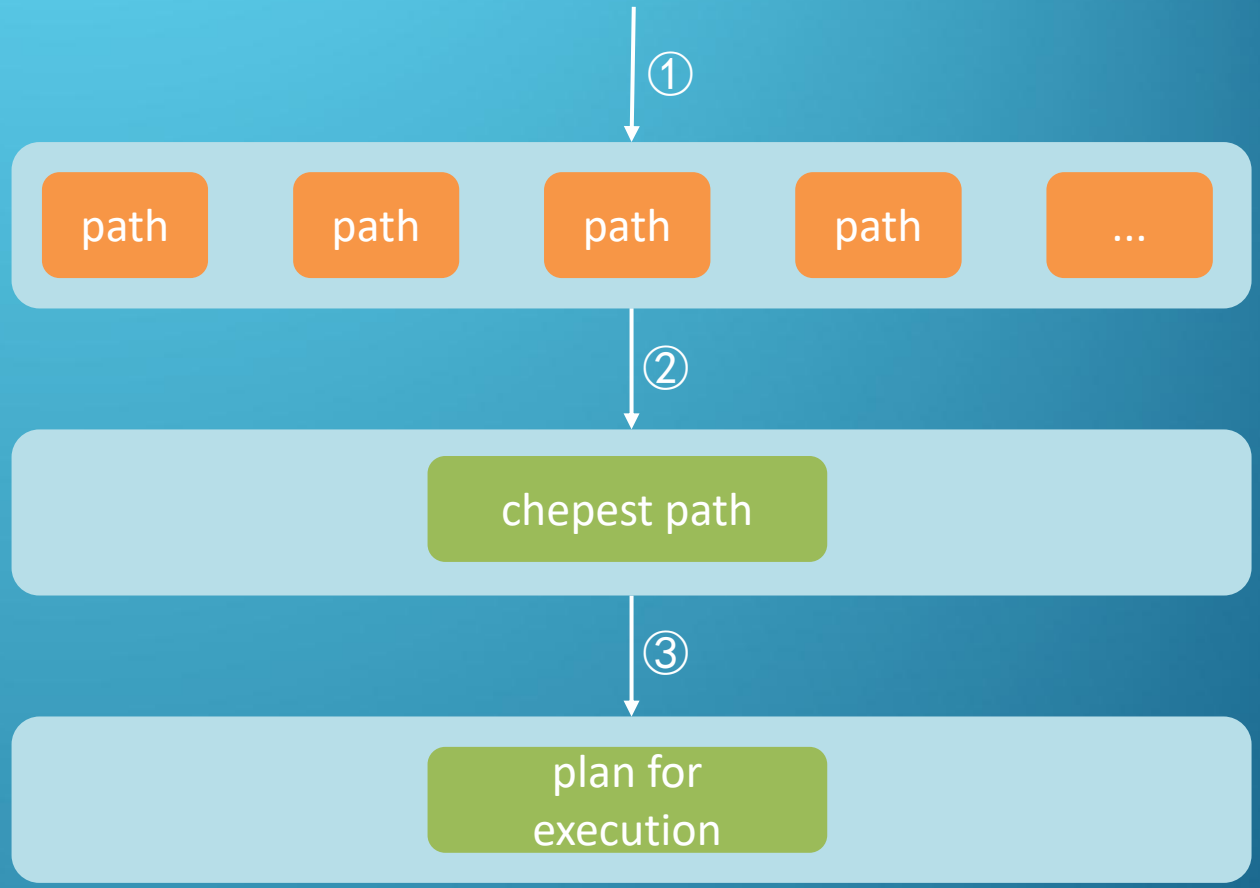


How parallel queries are planned?

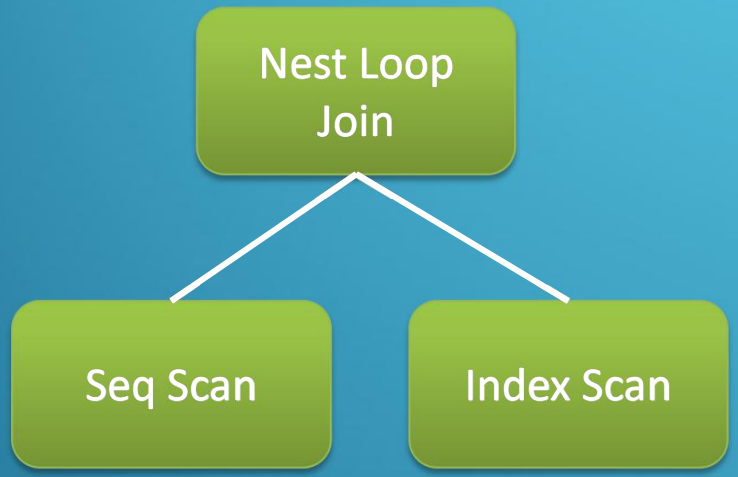
Cost-base Planner

① Think of all ways we could execute a query
② Estimate the runtime of each path, than choose the cheapest path
③ Convert path into a plan ready for execution

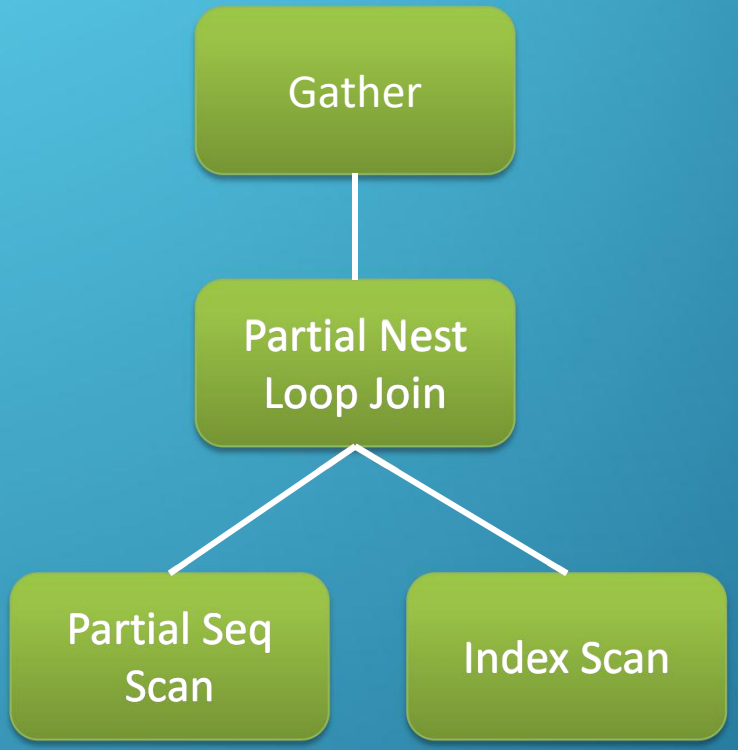
- For parallel query, introduce **parallel-aware** node and **partial paths**
- For partial paths, generate **Gather/GatherMerge** on top of them



Parallel path



VS



Rule-based parallel degree

- For join node, numbfer of workers to consider is based on the **driving** table
- ALTER TABLE ... SET (parallel_workers = N)
- SET min_parallel_table_scan_size = ‘8MB’
- $\sqrt[3]{\text{table\_size} / \text{min\_parallel\_table\_scan\_size}} + 1$
- SET min_parallel_index_scan_size = ‘512kB’
- SET max_parallel_workers_per_gather = 2

Costs

- SET parallel_setup_cost = 1000
  - Cost of setting up shared memory for parallelism, and launching workers.
  - Discourage parallel query for short queries
- SET parallel_tuple_cost = 0.1
  - Cost of CPU time to pass a tuple from worker to leader process
  - Discourage parallel query if large amouts of results have to be sent back

Parallelism cannot be used in the following cases

- Query writes any data or locks any database rows
- CTE(with...)
- FULL OUTER JOINs
- SERIALIZABLE transaction isolation
- Use functions marked PARALLEL UNSAFE
- DECLARE CURSOR

Future work

- More operators support parallelism, such as sort
- Dynamic repartitioning
- Cost-based planning of parallel degree?

References

- https://speakerdeck.com/macdice/parallelism-in-postgresql-11
- https://www.postgresql.org/docs/11/parallel-plans.html#PARALLEL-JOINS
- http://rhaas.blogspot.com/2013/10/parallelism-progress.html
- http://ashutoshpg.blogspot.com/2017/12/partition-wise-joins-divide-and-conquer.html
- http://www.gotw.ca/publications/concurrency-ddj.htm
- https://www.enterprisedb.com/blog/parallel-hash-postgresql
- https://write-skew.blogspot.com/2018/01/parallel-hash-for-postgresql.html
- http://amitkapila16.blogspot.com/2015/11/parallel-sequential-scans-in-play.html
- https://blog.2ndquadrant.com/parallel-aggregate/

Welcome to Alibaba Cloud Database Technology Group.

lingce.ldm@alibaba-inc.com

Thanks