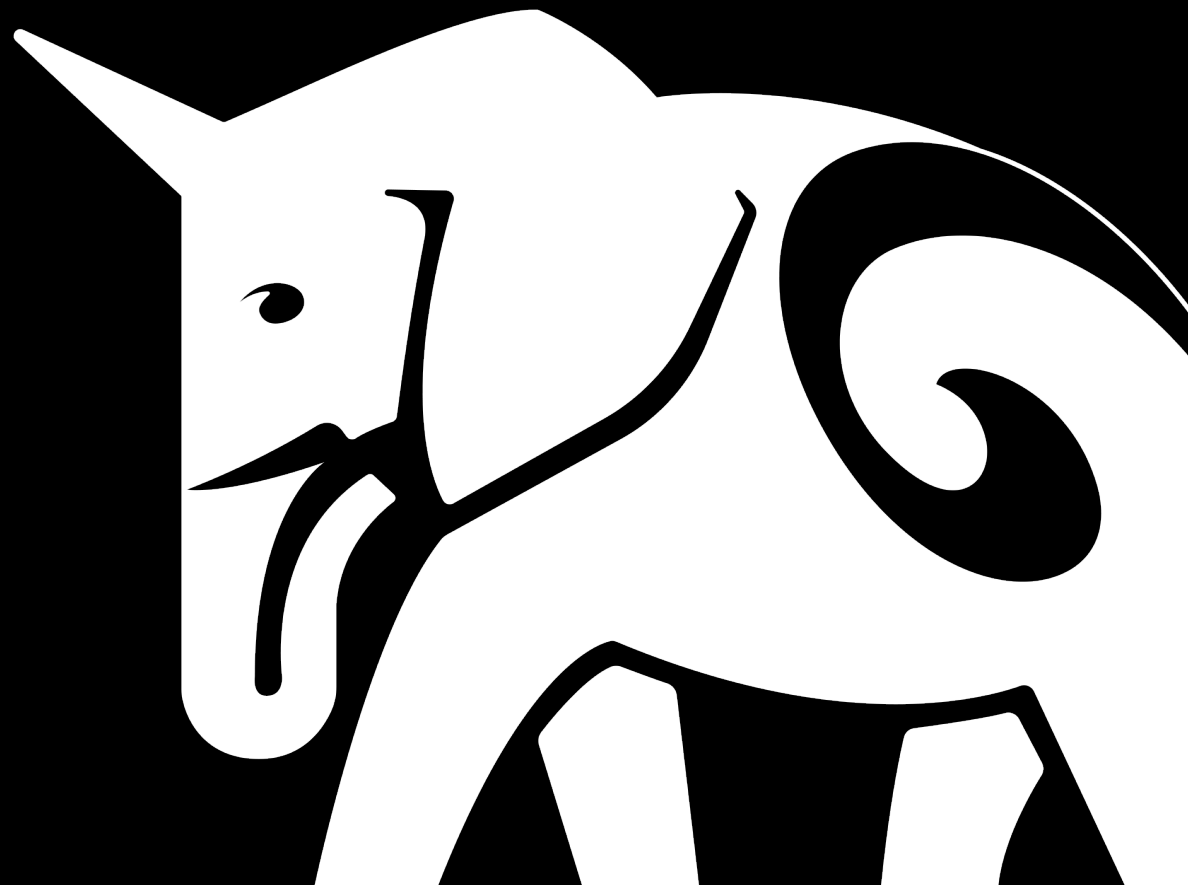
Microsoft

Citus

Distributed PostgreSQL as an Extension

Marco Slot
marco.slot@microsoft.com



What is Citus?

Citus is an extension (plug-in) to PostgreSQL which adds:

- Distributed tables
- Reference tables

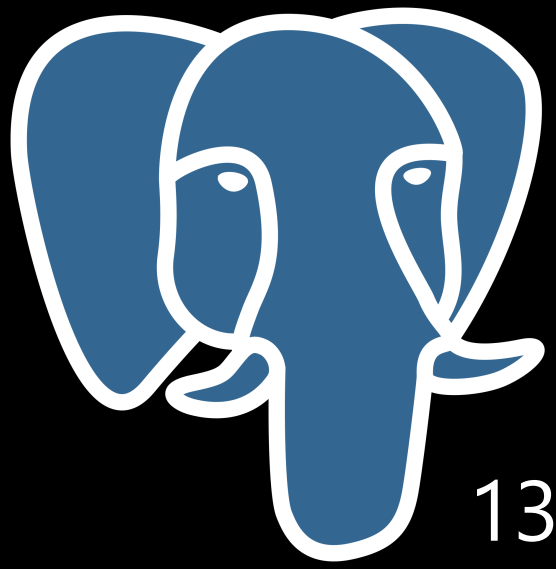
Simplicity and flexibility of using PostgreSQL, at scale.

Multi-purpose:

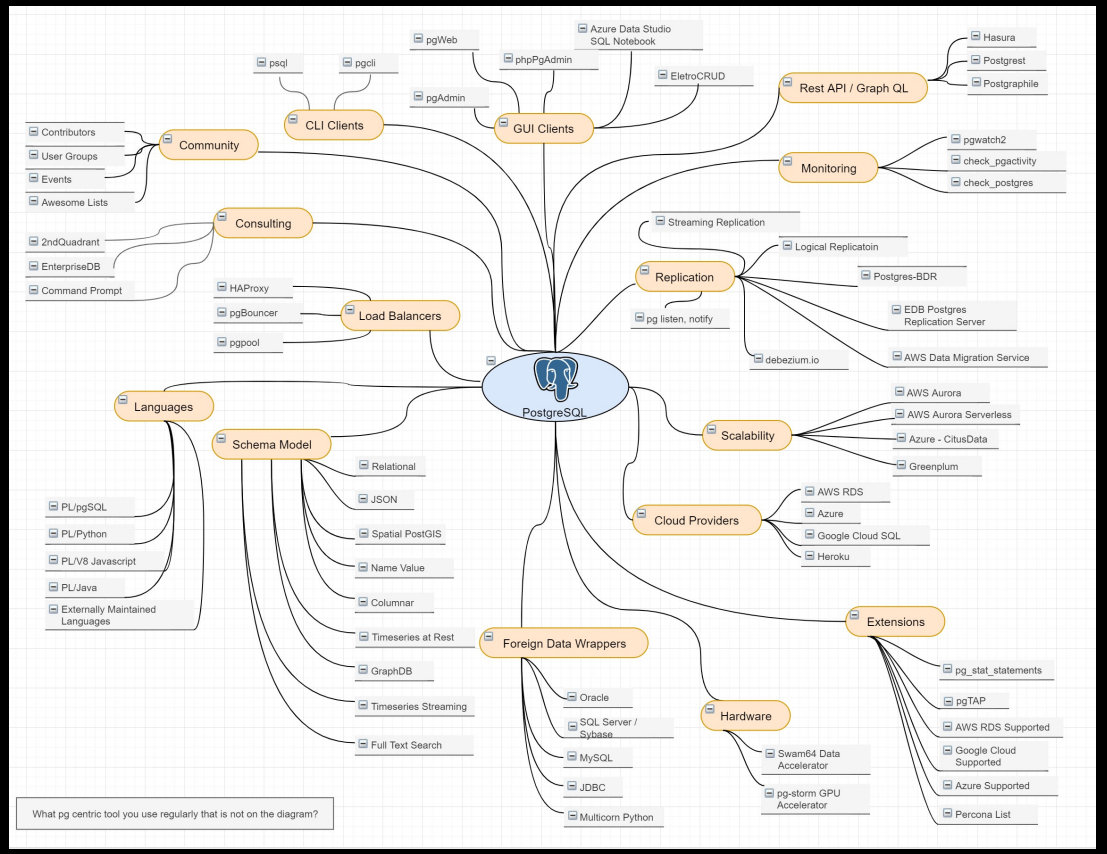
- Scale transactional workloads through routing / delegation
- Scale analytical workloads through parallelism and columnar storage

Why be an extension / not a fork?

PostgreSQL is a core project



and a vast ecosystem



https://github.com/EfficiencyGeek/postgresql-ecosystem

Why Citus?

1) PostgreSQL is limited to a single server

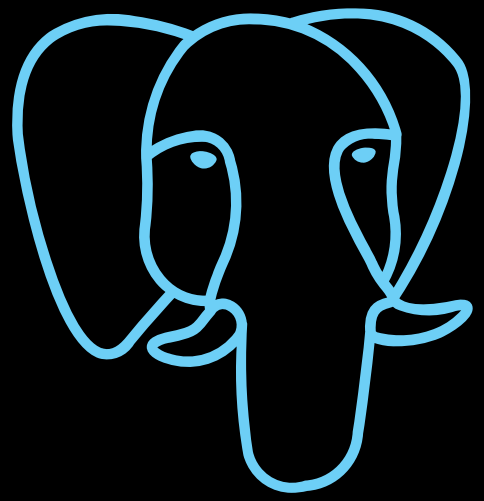
   Capacity / execution time issues:

   - Working set does not fit in memory
   - Reaching limits of network-attached storage (IOPS) / CPU
   - Analytical query takes too long
   - Data transformations are single-threaded (e.g. insert..select)
   - Autovacuum cannot keep up with transactional workload
   - …

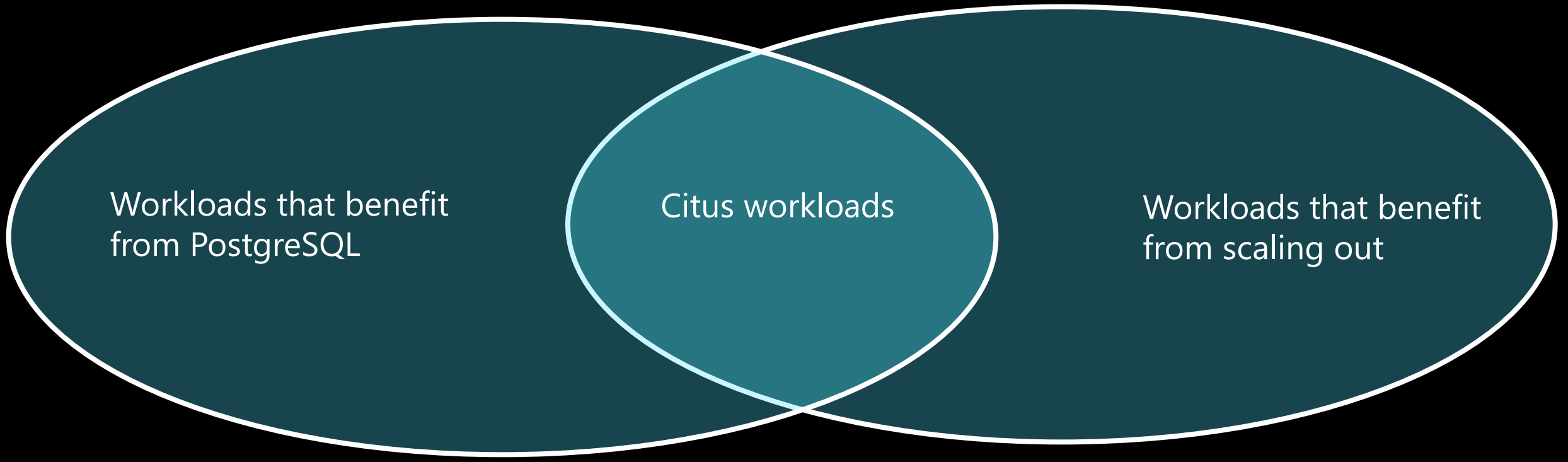
Why Citus?

2) Systems that are not PostgreSQL may be lacking one of:

- Joins
- Functions
- Constraints
- Indexes: B-tree, GIN, BRIN, & GiST
- Partial Indexes
- Other extensions
- PostGIS
- Rich datatypes
- JSONB
- Window functions
- CTEs
- Atomic update / delete
- Partitioning
- Interactive transactions
- Open source
- …



Citus workload patterns



Citus workload patterns

Multi-tenant

Software-as-a-service

Real-time Analytics

Customer-facing dashboards

High performance CRUD

Microservices

Data warehouse

Analytical reports

Citus workload requirements

Capability	Multi-tenant	Real-time Analytics	High perf. CRUD	Data warehouse
Distributed tables	Yes	Yes	Yes	Yes
Reference tables	Yes	Yes	Yes	Yes
Co-location	Yes	Yes	Yes	Yes
Parallel query		Yes		Yes
Parallel DML		Yes		
Query routing	Yes	Yes	Yes	
Fast full table scans				Yes
Fast bulk loading		Yes		Yes
Connection scaling			Yes	
Foreign keys	Yes			

• • •

How to transform PostgreSQL into a distributed database as an extension?

THE DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A. Rowe

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system. The main design goals of the new system are to:

1) provide better support for complex objects,

2) provide user extendibility for data types, operators and access methods,

3) provide facilities for active databases (i.e., alerters and triggers) and inferencing including forward- and backward-chaining,

THE DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A. Rowe

Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper presents the preliminary design of a new database management system, called POSTGRES, that is the successor to the INGRES relational database system. The main design goals of the new system are to:

1) provide better support for complex objects,

2) provide user extendibility for data types, operators and access methods,

3) provide facilities for active databases (i.e., alerters and triggers) and inferencing including forward- and backward-chaining,

transactions
planner, executor,
foreign data wrappers
Background workers

What is an extension?

Extensions consist of:

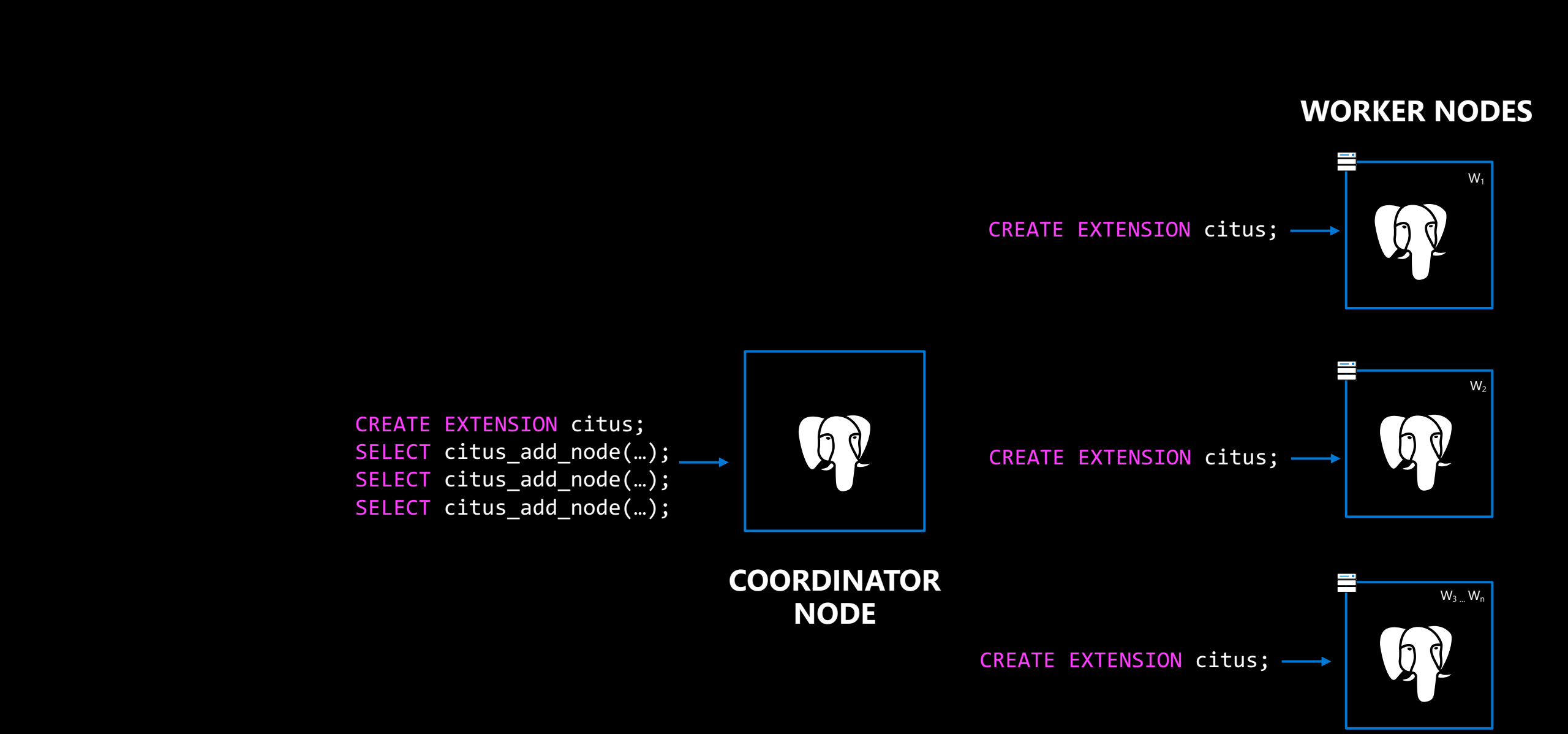
1. SQL objects (tables, functions, types, …)
2. Shared library

citus.sql

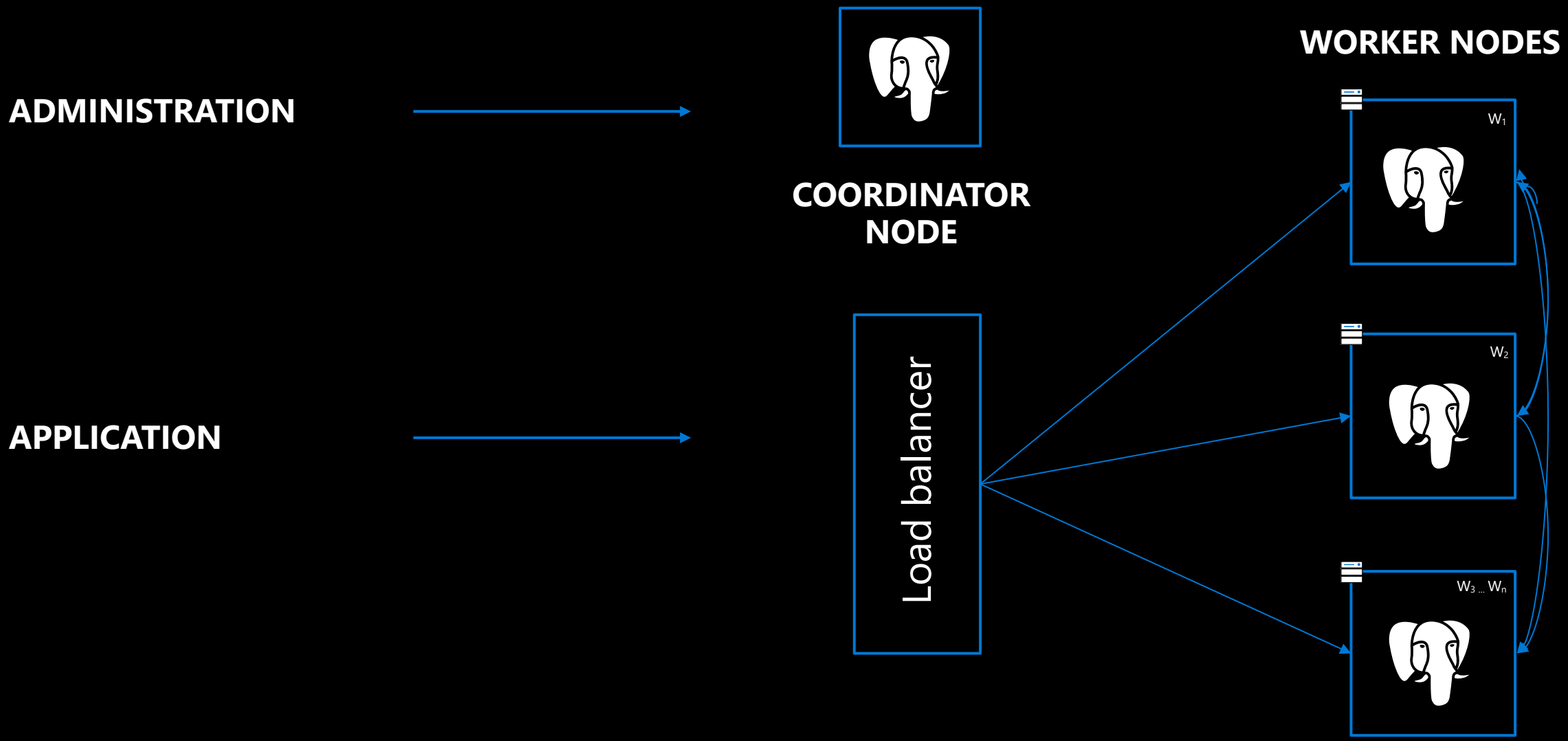
```
CREATE TABLE pg_dist_node (…);
CREATE TABLE pg_dist_partition (…);

CREATE FUNCTION citus_add_node(…)
RETURNS void LANGUAGE c
AS '$libdir/citus',
$function$citus_add_node$function$;

CREATE FUNCTION create_distributed_table(…)
RETURNS void LANGUAGE c
AS '$libdir/citus',
$function$create_distributed_table$function$;
```

citus.c

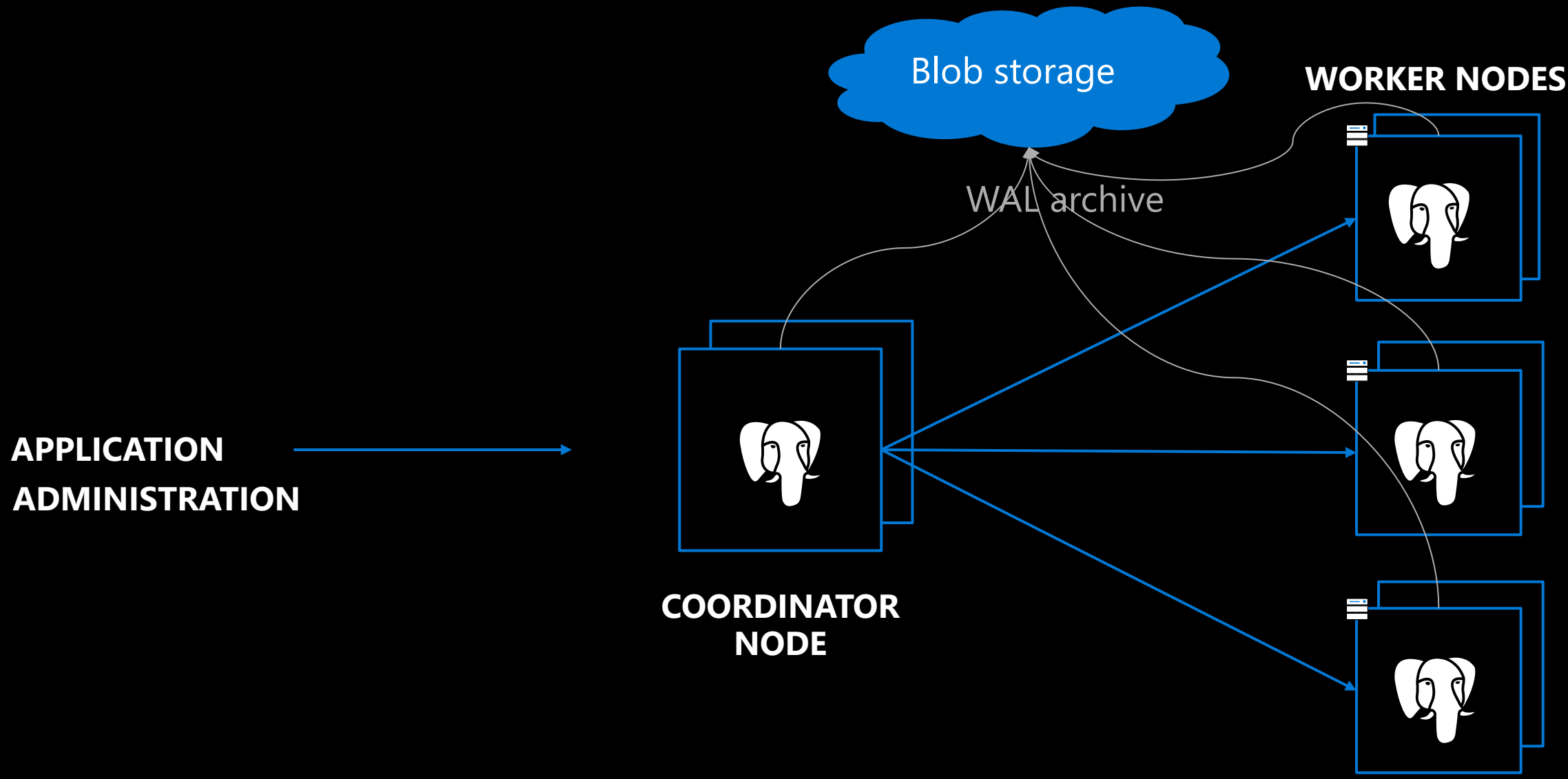
```
#include "postgres.h"


Datum citus_add_node(…)
{
    …
}

Datum create_distributed_table(…)
{
    …
}
```



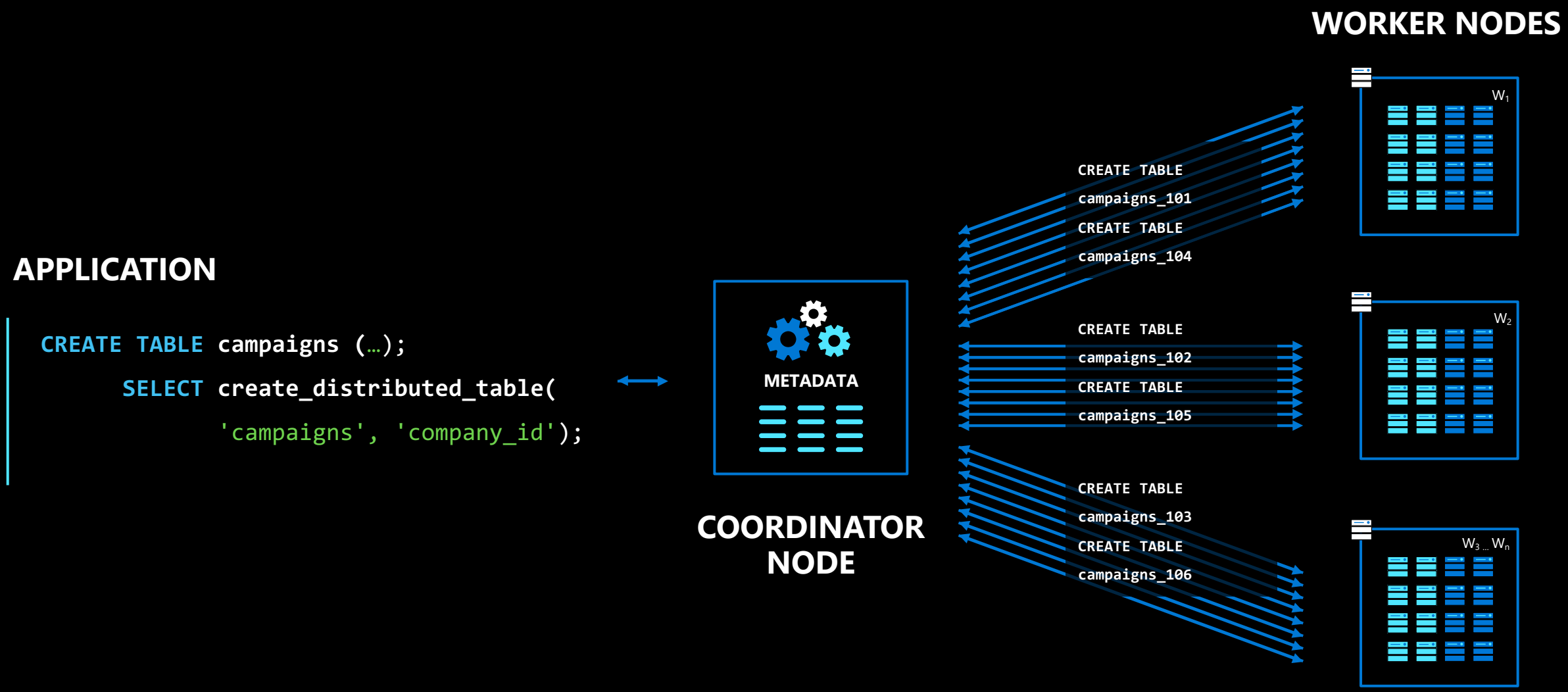
A Citus cluster consists of multiple PostgreSQL servers with the Citus extension.



Worker nodes can also act as coordinator



Typical production deployment



How Citus distributes tables across the database cluster

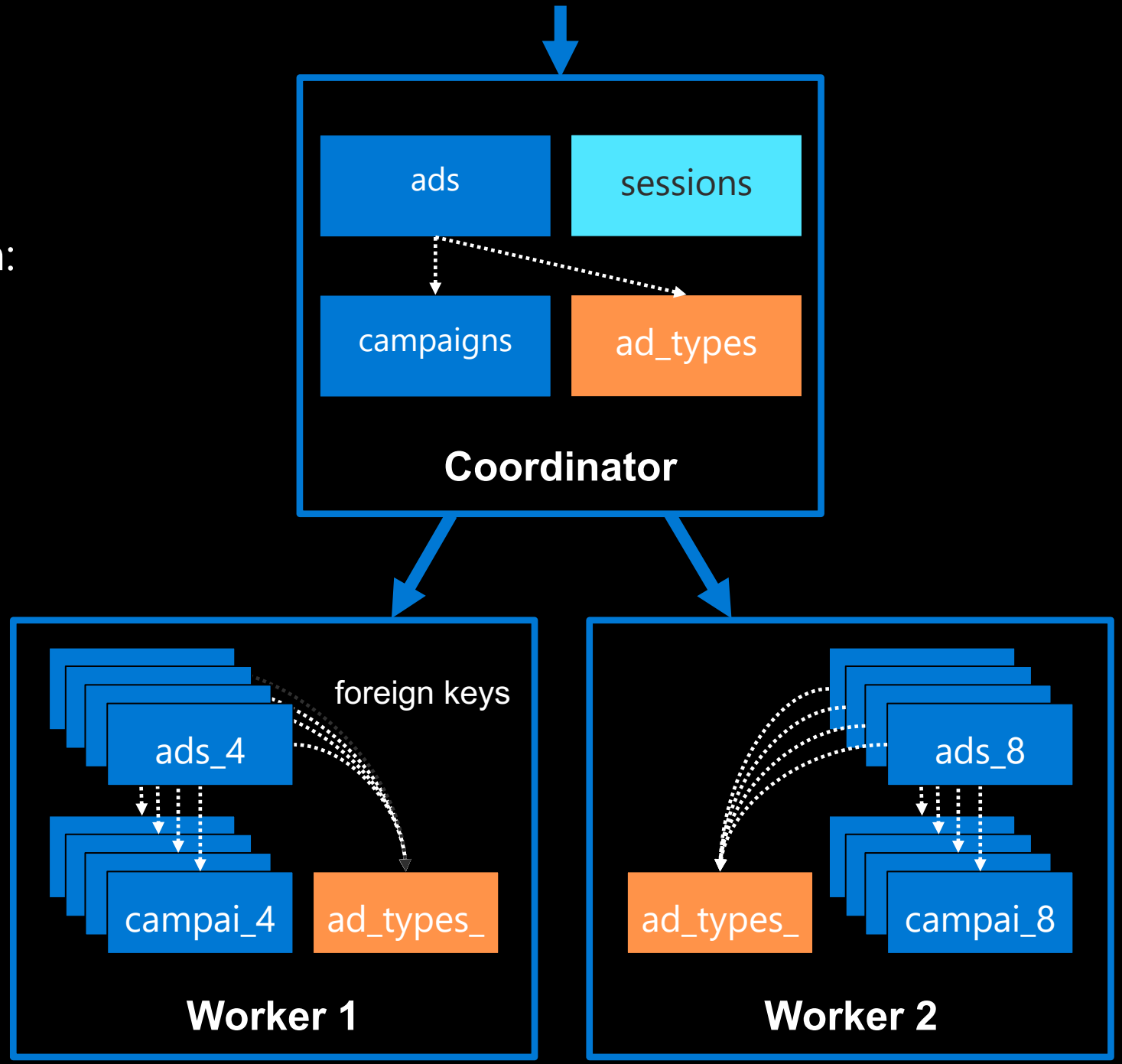
Citus Table Types

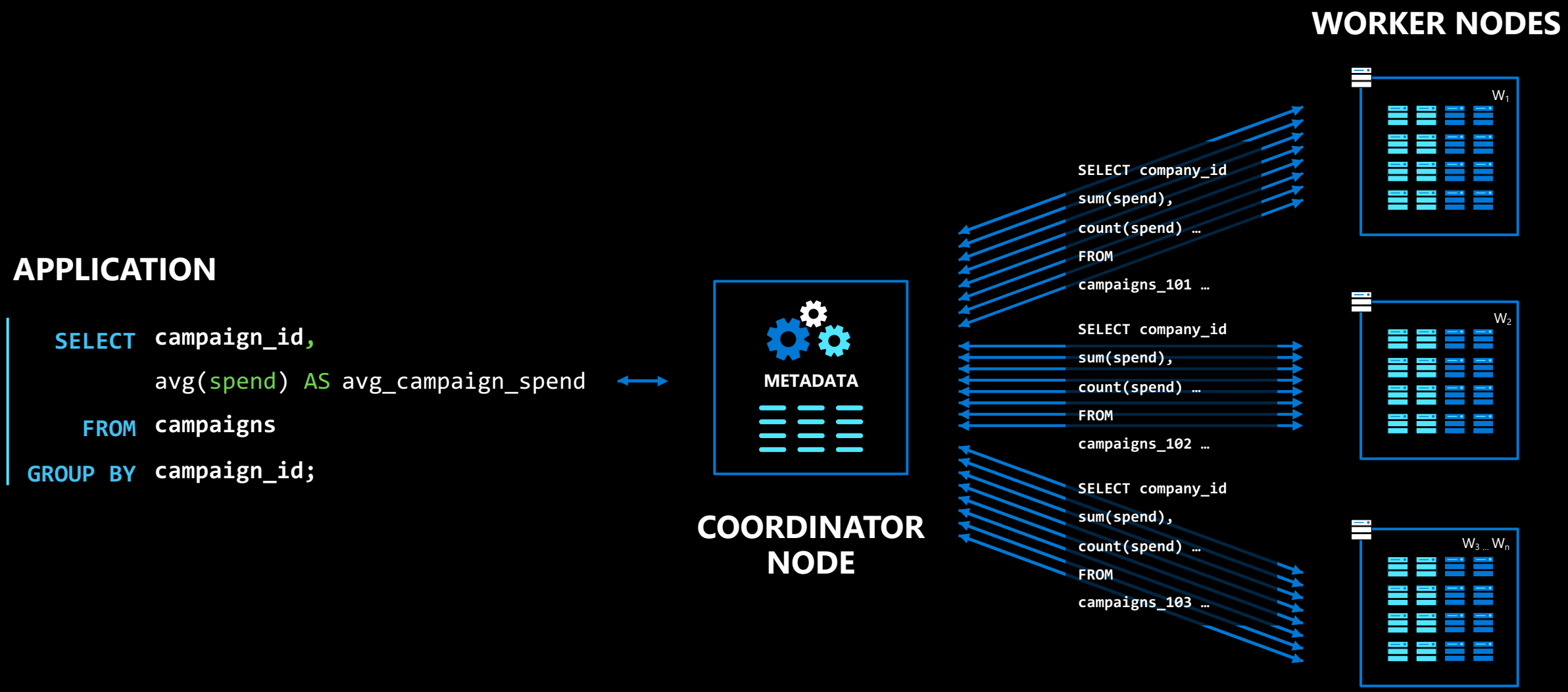
Distributed tables with co-location:

```
SELECT create_distributed_table(
    'campaigns', 'company_id');

SELECT create_distributed_table(
    'ads', 'company_id');
```

Reference table:

```
SELECT create_reference_table(
    'ad_types');
```





How Citus distributes queries across the database cluster

Query planner hook

PostgreSQL defines function pointers that are called at critical points.

postgres.c

```
planner_hook_type planner_hook = NULL;

PlannedStmt *
planner(Query *parse, …)
{
    PlannedStmt *result;

    if (planner_hook)
        result = (*planner_hook) (parse,  …);
    else
        result = standard_planner(parse, …);
    return result;
}
```

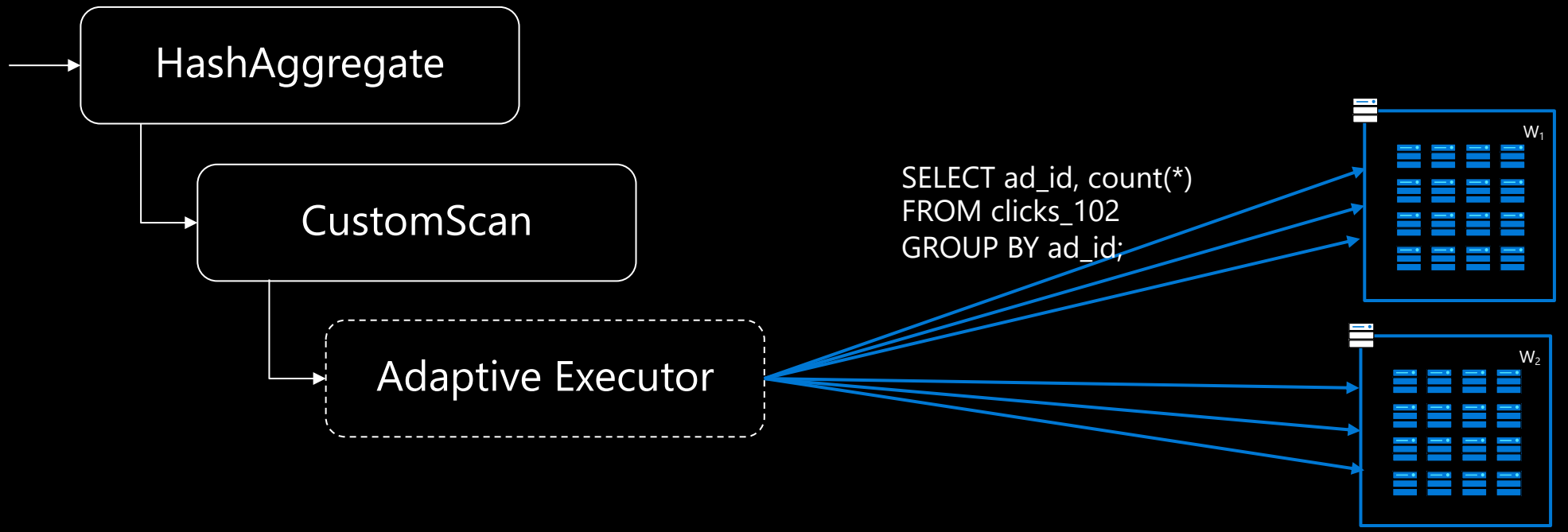
citus.c

```
#include "postgres.h"

void _PG_init(void)
{
    …
    planner_hook = distributed_planner;
    …
}

PlannedStmt *
distributed_planner(Query *parse, …)
{
    …
}
```

Executor hook: CustomScan

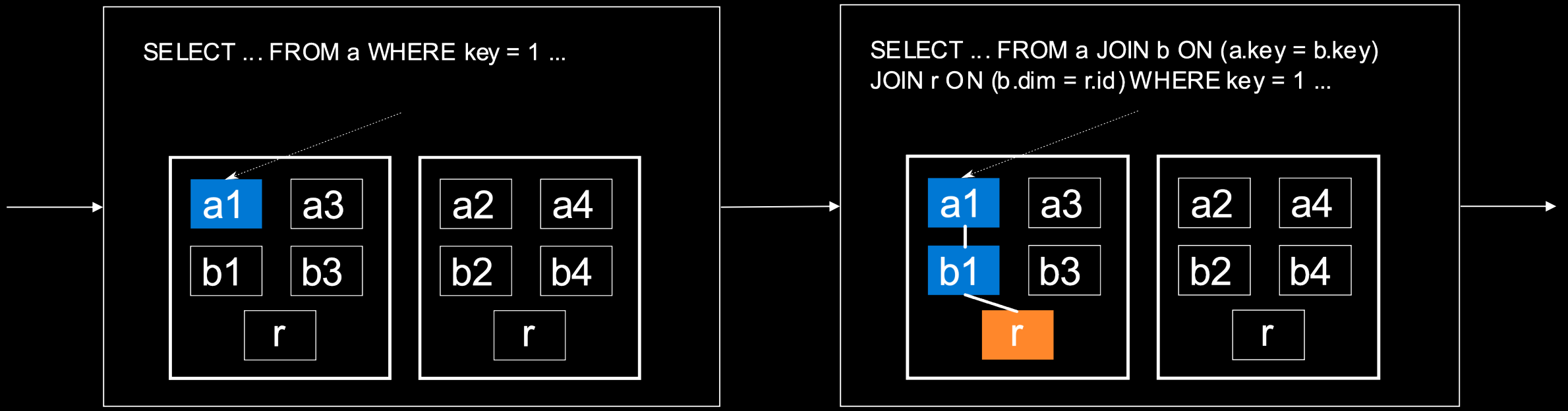
Citus integrates into the PostgreSQL planner via a CustomScan

SELECT ad_id, count(*) FROM clicks GROUP BY ad_id;



Citus query planners

Queries go through several planning layers



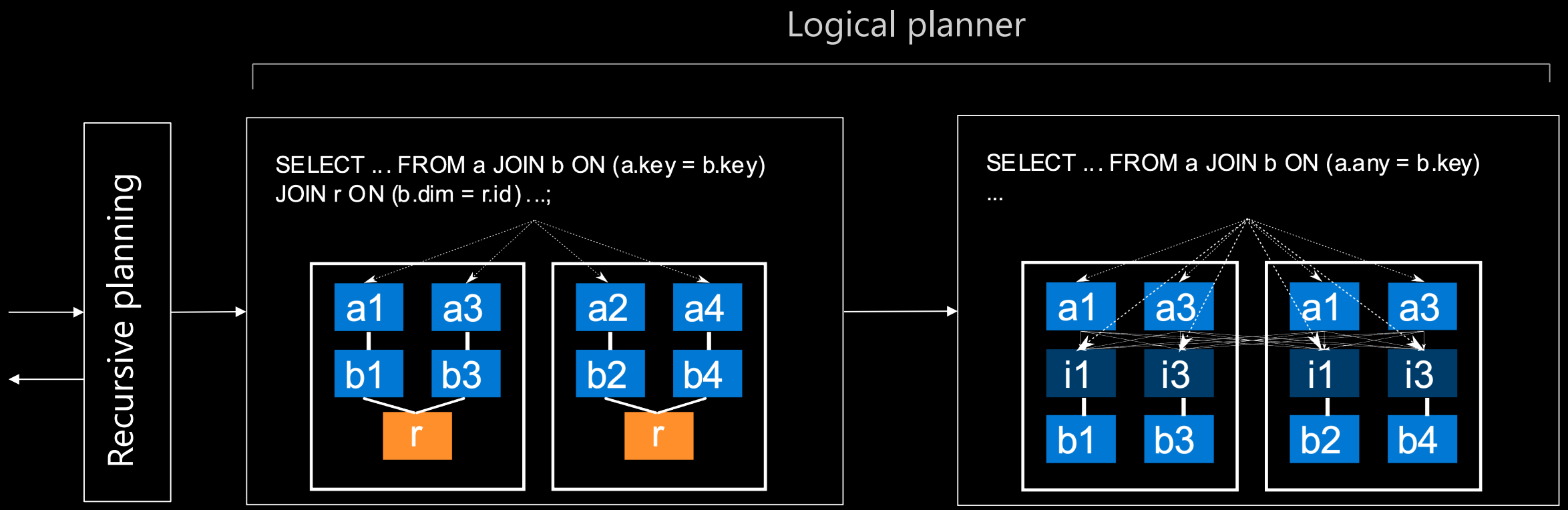
Fast path

(CRUD operations)

Router

(Single tenant operations)

Citus query planners



Query pushdown

(Real-time analytics)

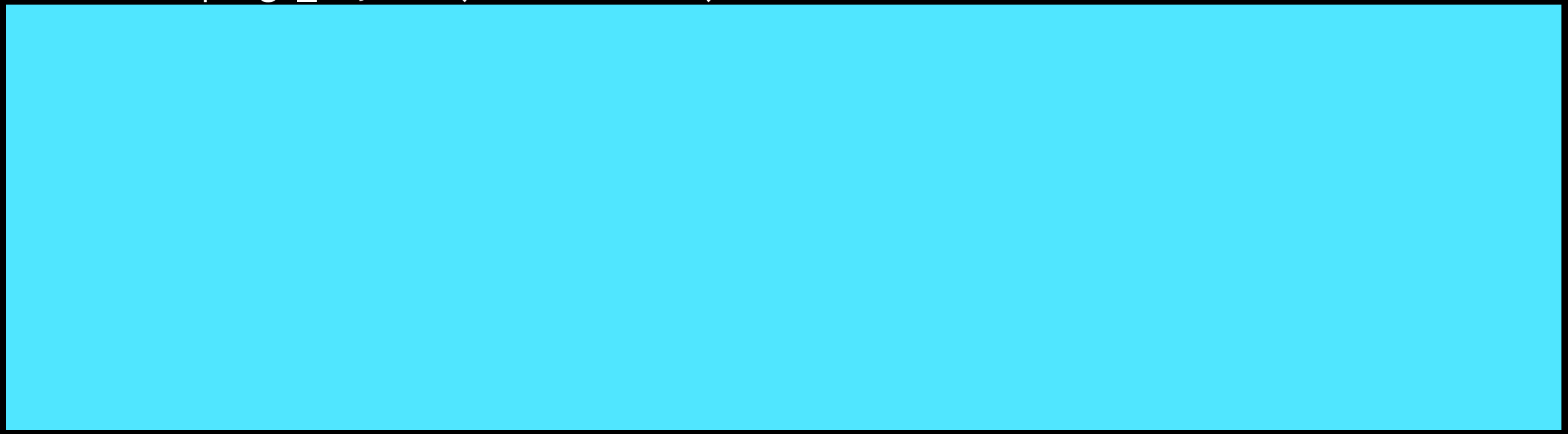
Join order

(Data warehouse)

Query pushdown planning

Complex queries can be embarrassingly parallel

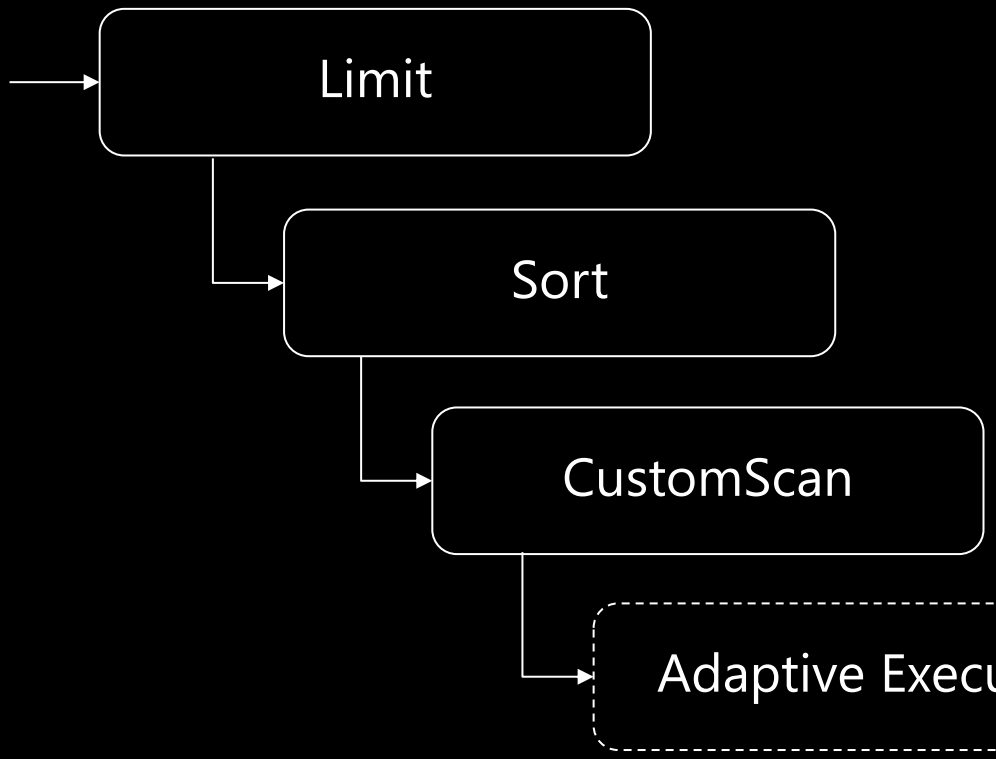
```
SELECT campaign_id, sum(clicks.count) FROM
```



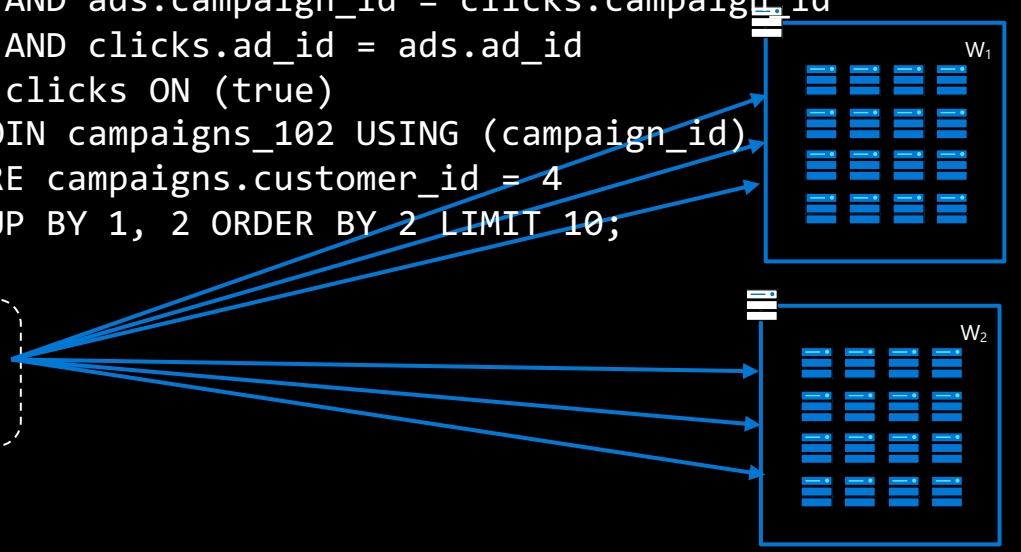
```
GROUP BY 1 ORDER BY 2 DESC LIMIT 10;
```

Query pushdown planning

Complex queries can be embarrassingly parallel



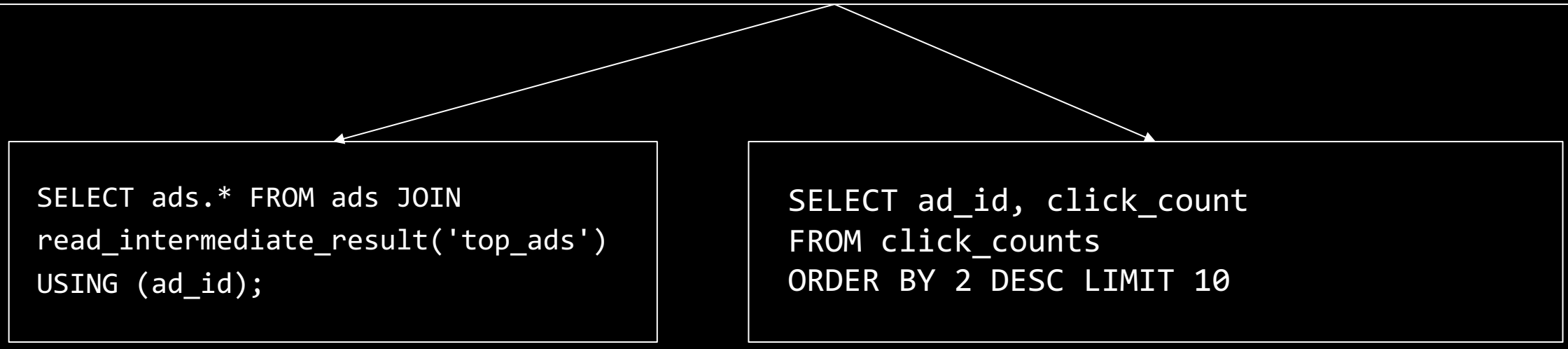

```
SELECT campaign_id, ad_id, sum(clicks.count) FROM
  ads_303 LEFT JOIN LATERAL (
    SELECT campaign_id, count(*)
    FROM clicks_402 JOIN sites_1 USING (site_id)
    WHERE sites.language = 'nl'
    AND clicks.click_time >= now() - interval '1 hour'
    AND ads.campaign_id = clicks.campaign_id
    AND clicks.ad_id = ads.ad_id
  ) clicks ON (true)
  JOIN campaigns_102 USING (campaign_id)
WHERE campaigns.customer_id = 4
GROUP BY 1, 2 ORDER BY 2 LIMIT 10;
```



Recursive planning

Plan non-pushdownable subqueries separately

```
SELECT ads.* FROM ads JOIN (
    SELECT ad_id, click_count FROM click_counts ORDER BY 2 DESC LIMIT 10
) top_ads USING (ad_id);
```



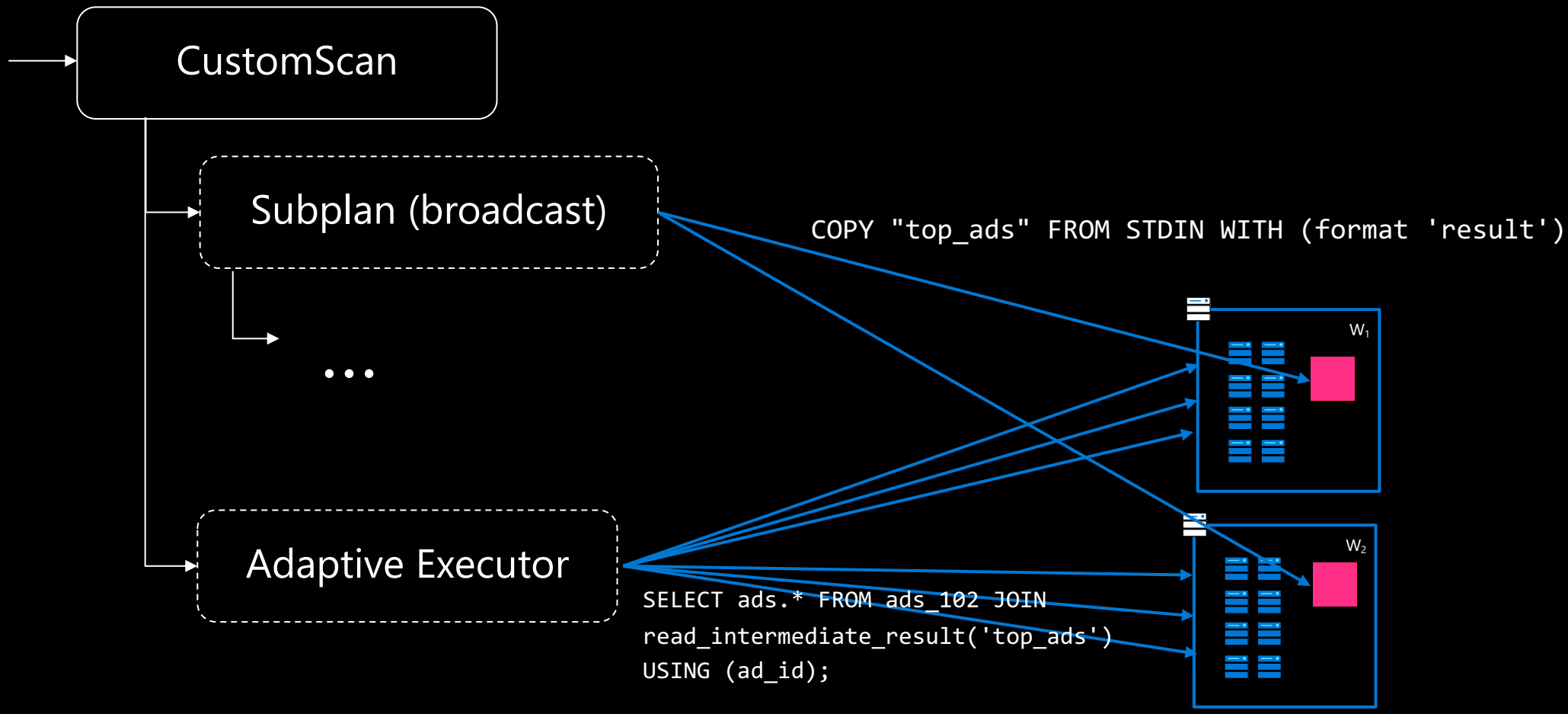
```
SELECT ads.* FROM ads JOIN
read_intermediate_result('top_ads')
USING (ad_id);
```

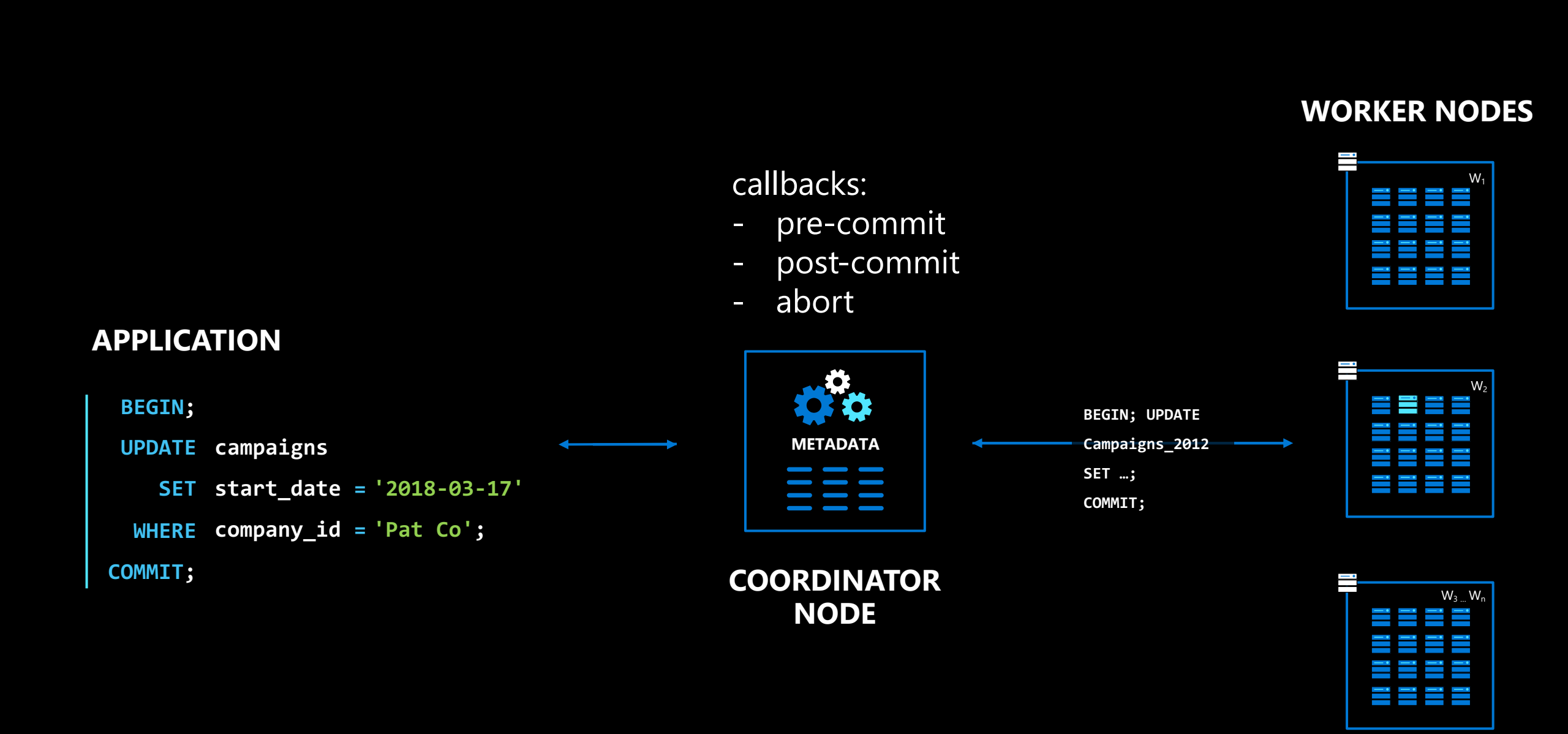
Main query

```
SELECT ad_id, click_count
FROM click_counts
ORDER BY 2 DESC LIMIT 10
```

Subplan query

Subplan execution (broadcast)

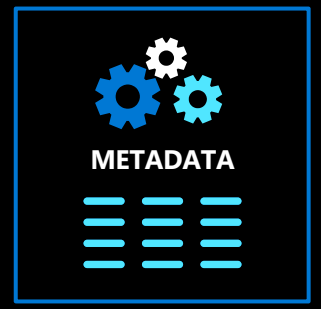




How Citus handles transactions in a multi-node cluster

APPLICATION

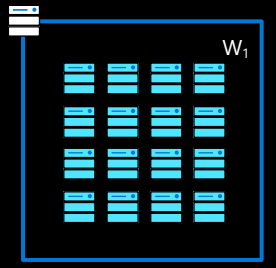
```
BEGIN;
UPDATE  campaigns
   SET  started = true
 WHERE  campaign_id = 2;
UPDATE  ads
   SET  finished = true
 WHERE  campaign_id = 1;
COMMIT;
```

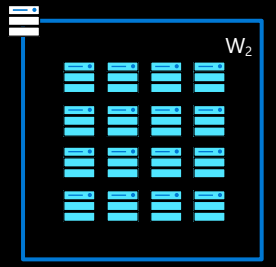


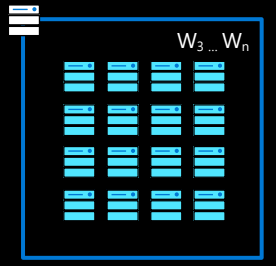
COORDINATOR NODE

callbacks:
- pre-commit
- post-commit
- abort

WORKER NODES



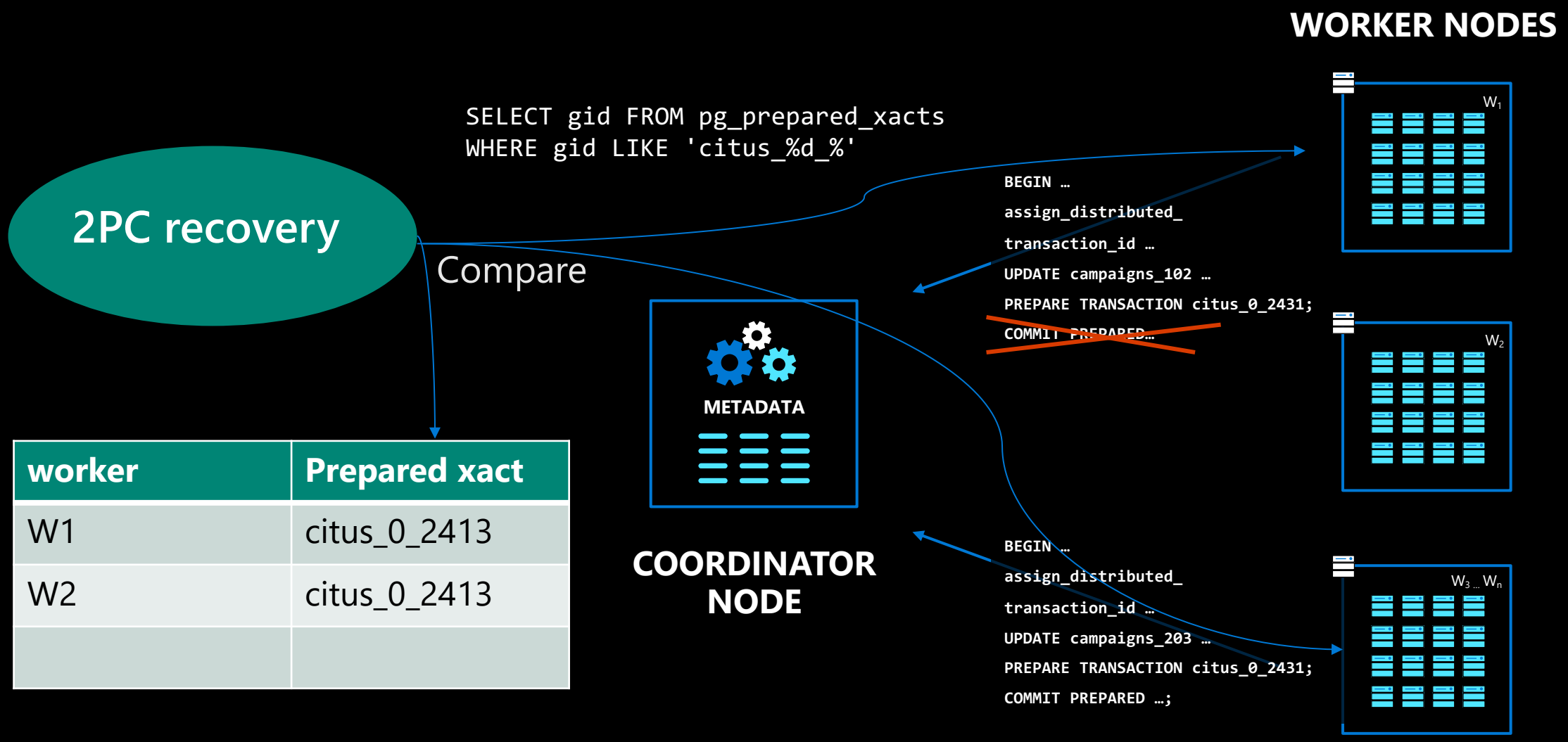




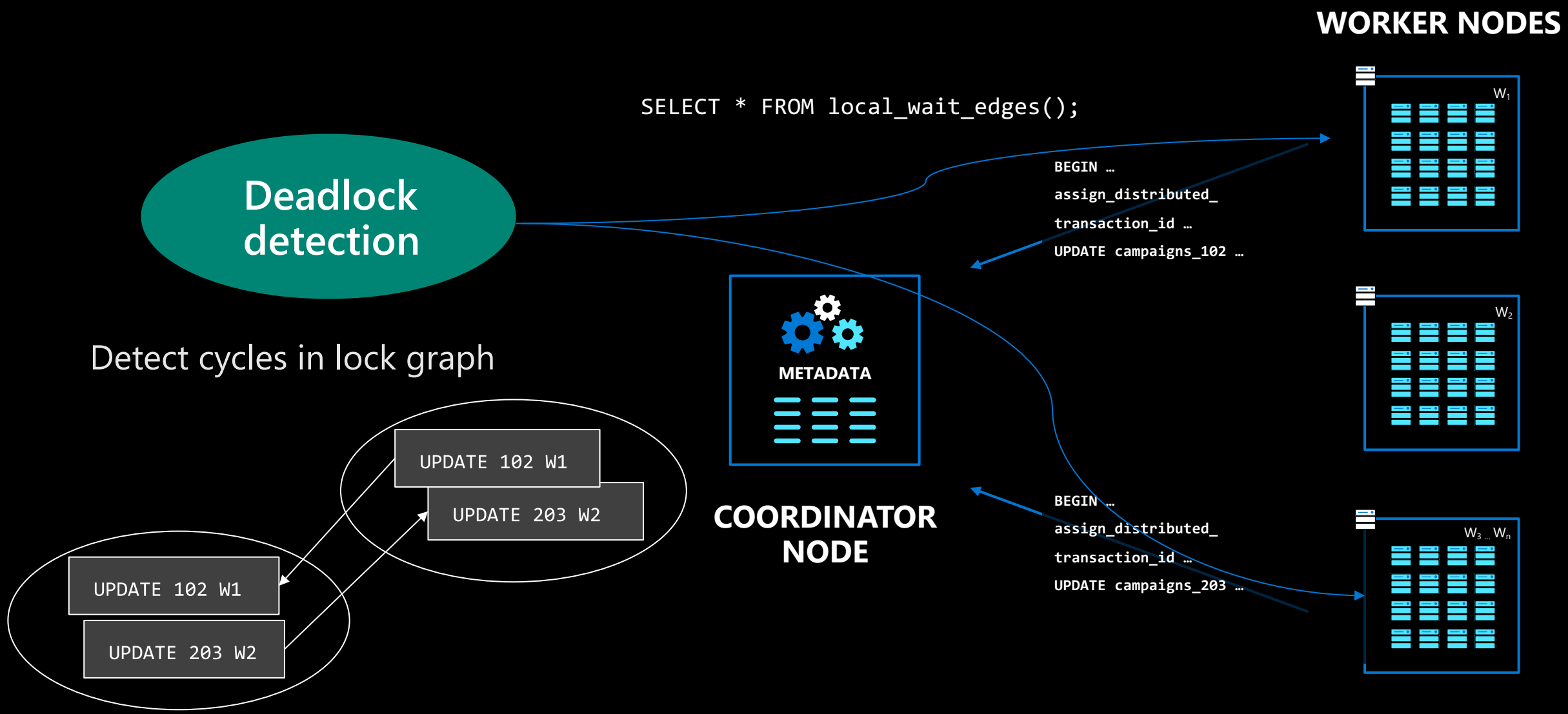

```
BEGIN …
assign_distributed_
transaction_id …
UPDATE campaigns_102 …
PREPARE TRANSACTION…
COMMIT PREPARED…
```

```
BEGIN …
assign_distributed_
transaction_id …
UPDATE campaigns_203 …
PREPARE TRANSACTION…
COMMIT PREPARED…
```

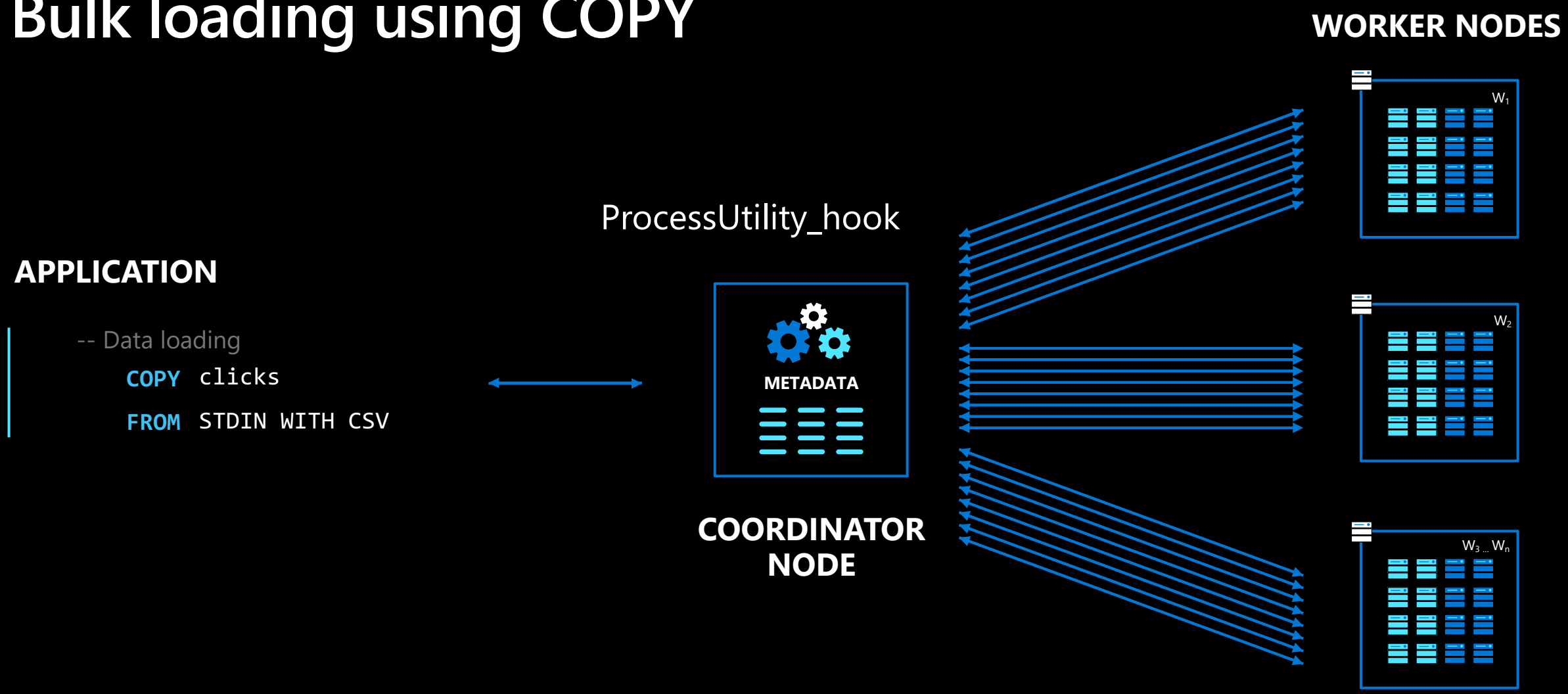
How Citus distributes transactions in a multi-node cluster



worker	Prepared xact
W1	citus_0_2413
W2	citus_0_2413


WORKER NODES
SELECT * FROM local_wait_edges();
Deadlock detection
BEGIN …
assign_distributed_
transaction_id …
UPDATE campaigns_102 …
W1
W2
Detect cycles in lock graph
METADATA
UPDATE 102 W1
UPDATE 203 W2
UPDATE 102 W1
UPDATE 203 W2
COORDINATOR NODE
BEGIN …
assign_distributed_
transaction_id …
UPDATE campaigns_203 …
W3 … Wn

Bulk loading using COPY

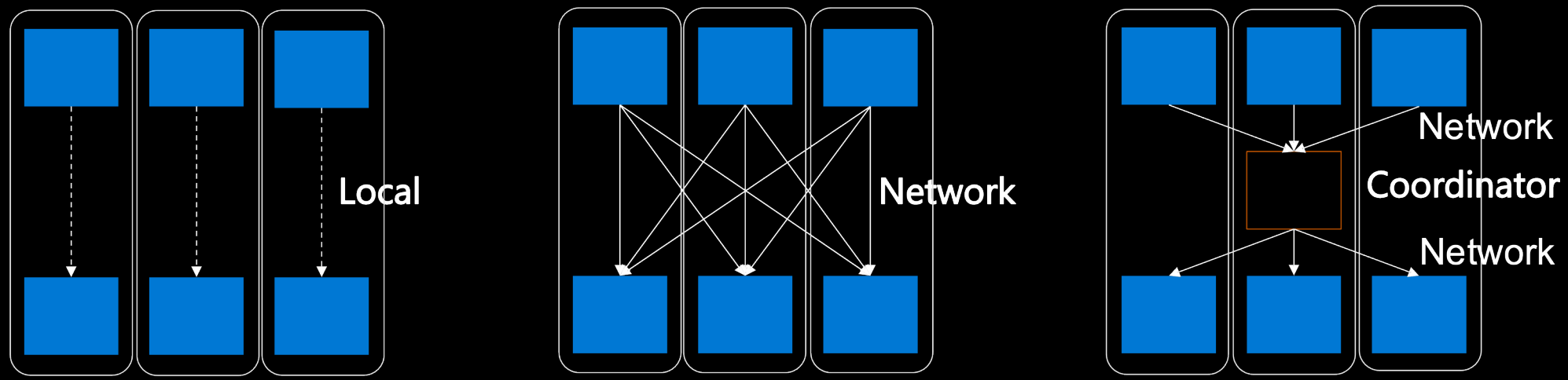


How Citus parallelizes bulk loading

Creating rollup tables with INSERT..SELECT

Distributed, transactional, parallel data transformation:

```
INSERT INTO click_counts
SELECT campaign_id, date_trunc('hour', click_time), count(*)
FROM clicks
WHERE ingest_time BETWEEN '2021-02-22 00:00' AND '2021-02-22 00:05’
GROUP BY 1, 2
ON CONFLICT (campaign_id, hour) DO UPDATE count = rollup.count + EXCLUDED.count;
```



Demo

Lessons learned

- PostgreSQL is extensible enough to build a comprehensive distributed database system.
- Distribution choices are essential to scale, but difficult for users.
- Help customers adopt your database, get paid when they scale out.
- If you’re going for compatibility, look at what ORMs are doing.
- Solve adoption blockers.
- Be prescriptive about the problem/workload you’re solving.

Open challenges

- Hybrid local-distributed databases
- Picking good distribution columns automatically
- Creating good rollups automatically
- Distributed geospatial (PostGIS) join optimization
- Nested distributed transactions (e.g. in triggers)
- Arbitrary foreign keys
- Linear connection scaling

Questions?

marco.slot@microsoft.com

Citus GitHub
github.com/citusdata/citus

Citus Slack
slack.citusdata.com

Citus Docs
docs.citusdata.com

Azure - Hyperscale (Citus) Docs
docs.microsoft.com/azure/postgresql/hyperscale/

PostgreSQL hooks documentation
github.com/AmatanHead/psql-hooks