

# Postgres Scaling Opportunities

BRUCE MOMJIAN



Configuring Postgres for heavy workloads can take many forms.  
This talk explores available Postgres scaling options.

*Creative Commons Attribution License*

*<http://momjian.us/presentations>*

*Last updated: July, 2018*

# Scaling

Database scaling is the ability to increase database throughput by utilizing additional resources such as I/O, memory, CPU, or additional computers.

However, the high concurrency and write requirements of database servers make scaling a challenge. Sometimes scaling is only possible with multiple sessions, while other options require data model adjustments or server configuration changes.

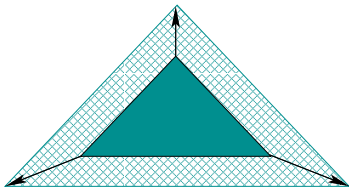
# Outline

Postgres scaling opportunities:

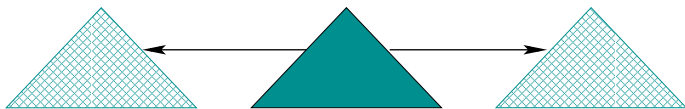
1. Multi-session
2. Single-session
3. Multi-host

# Vertical/Horizontal Scaling

**Vertical**



**Horizontal**



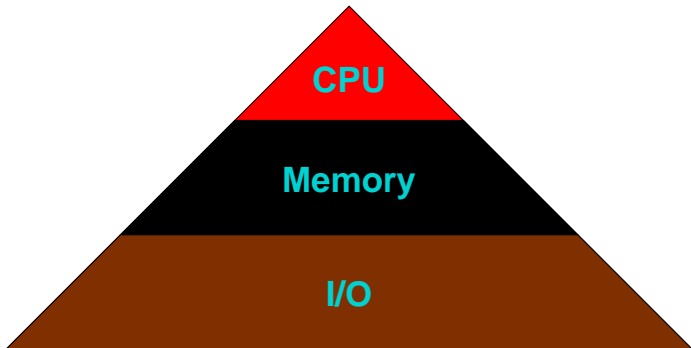
# Examples

Vertical scaling examples:

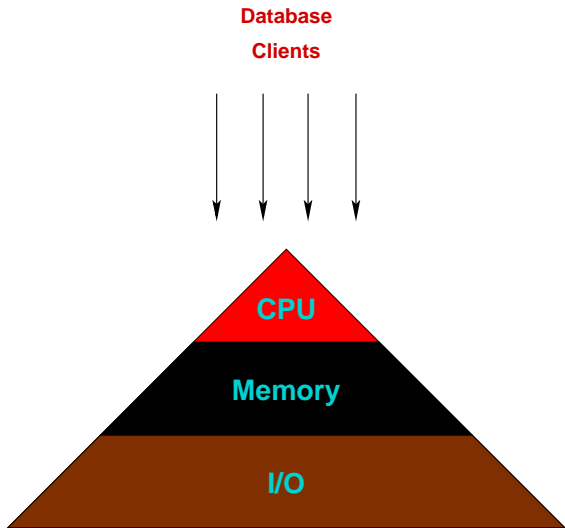
- ▶ More and faster CPUs
- ▶ More memory
- ▶ More and faster storage
- ▶ Battery-backed cache (BBU)

Horizontal scaling involves adding servers.

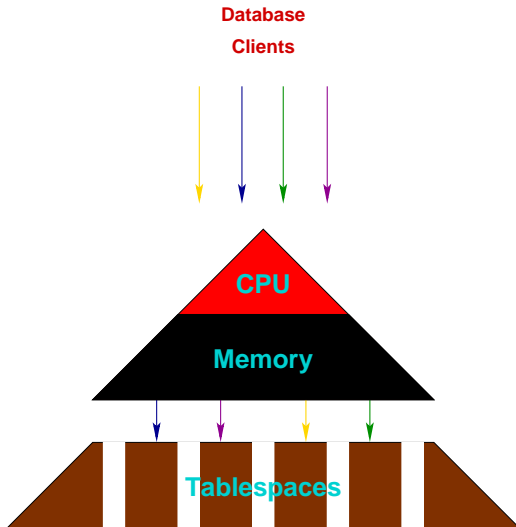
# Hardware Components



# 1. Multi-Session



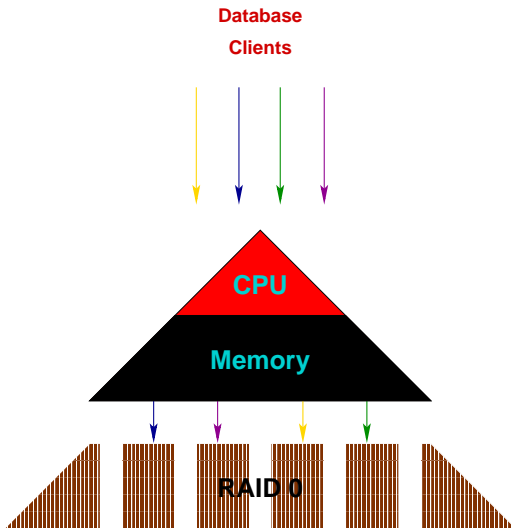
# I/O Spreading Using Tablespaces



Requires tables & indexes to be spread across tablespaces  
Tablespaces should be on different storage devices

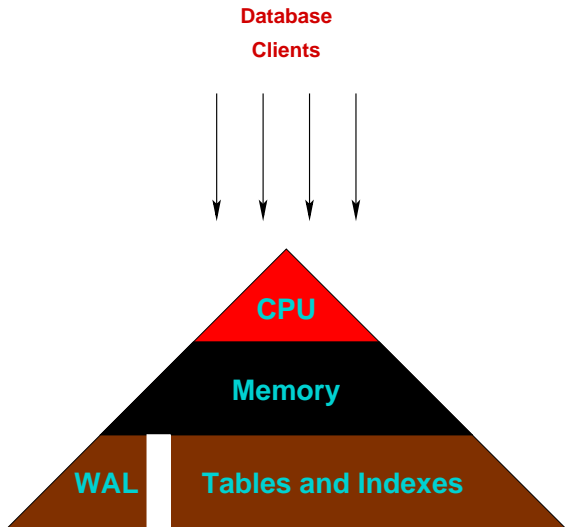


# I/O Spreading Using RAID 0



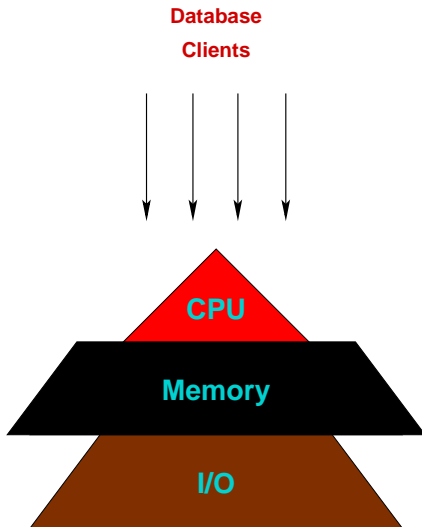
Auto-hashed by storage block number

# Write Spreading Using WAL Relocation



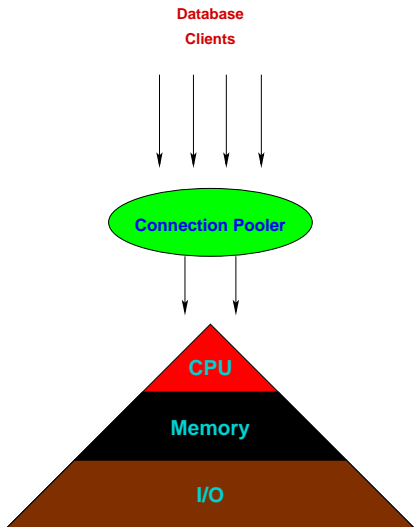
Separates WAL writes from table & index I/O

# Read Reduction via Increased Memory



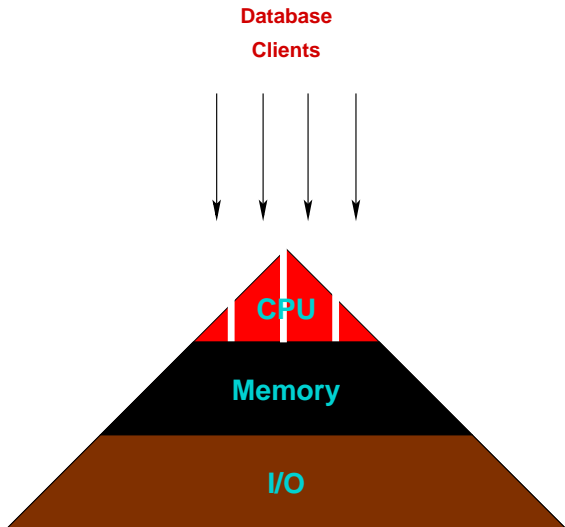
Additional memory caching reduces read requirements

# Scaling Connections Using a Pooler



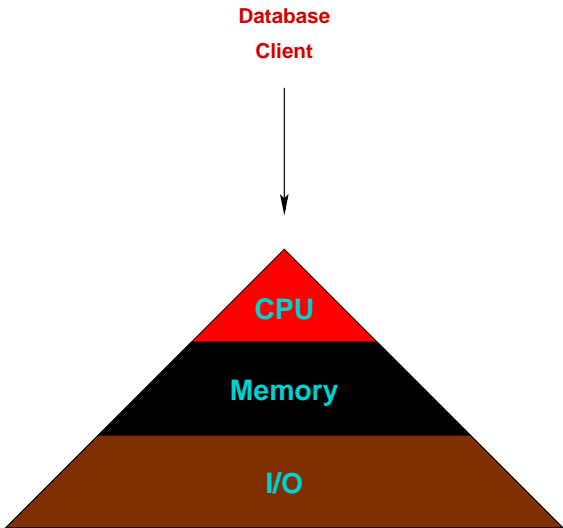
Fewer idle connections reduces resource usage

# Multi-Session CPU Scaling

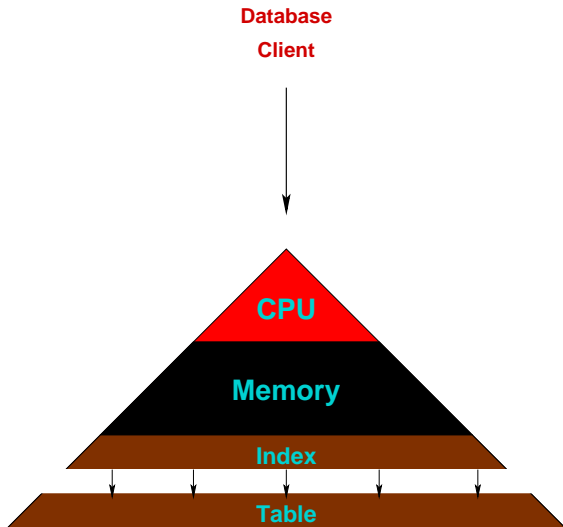


Multiple sessions spread across available CPUs

## 2. Single-Session

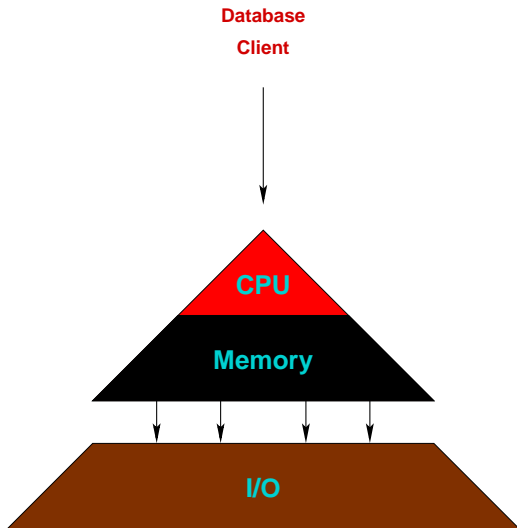


# Read Parallelism Using `effective_io_concurrency`



Used during bitmap heap scans  
Assumes table is hashed across multiple devices

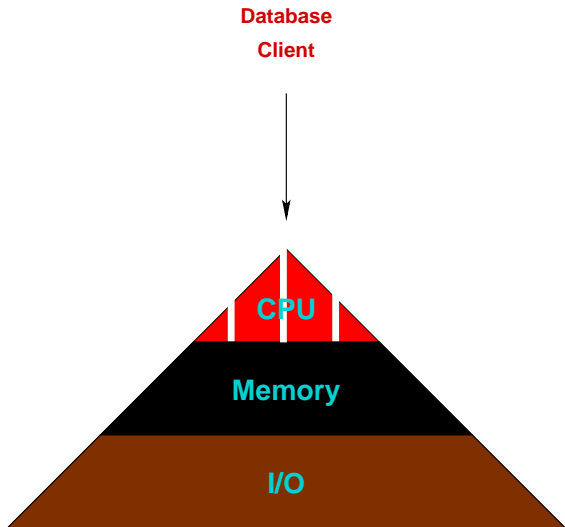
# I/O Scaling via Parallelism (not implemented)



Involves parallel index, heap, partition, and tablespace access

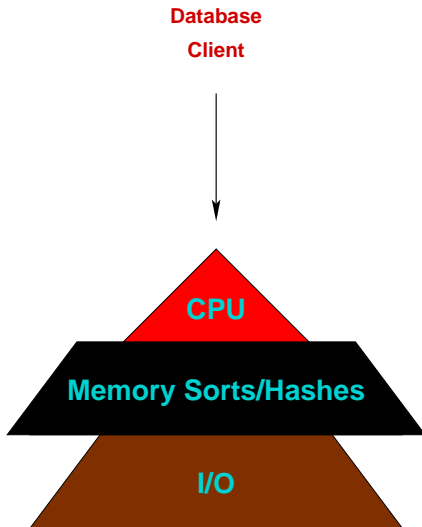


# CPU Scaling via Parallelism (not implemented)



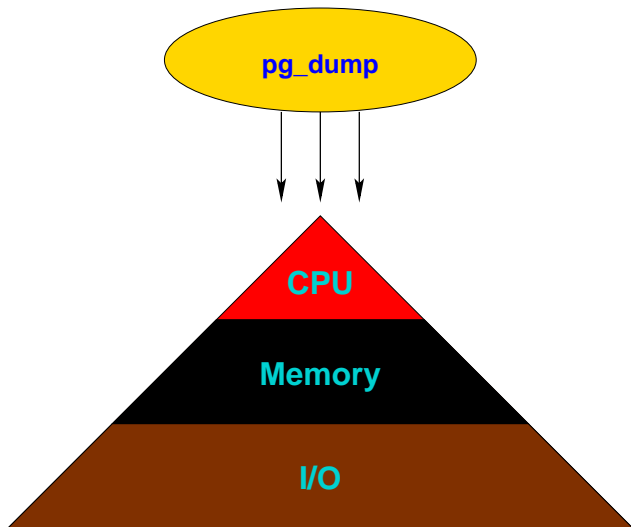
Involves parallel sorts, joins, and function execution

# Sort I/O Reduction Using work\_mem



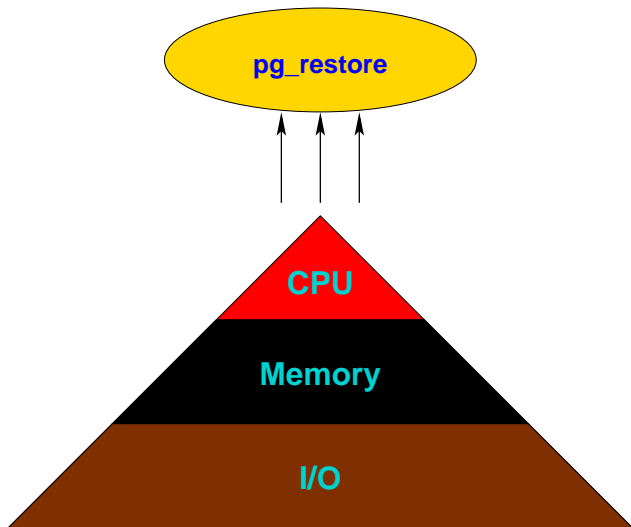
Reduces temporary result reads & writes

# Logical Dump Parallelism



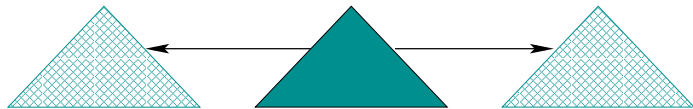
Dumps tables using concurrent database connections

# Logical Restore Parallelism

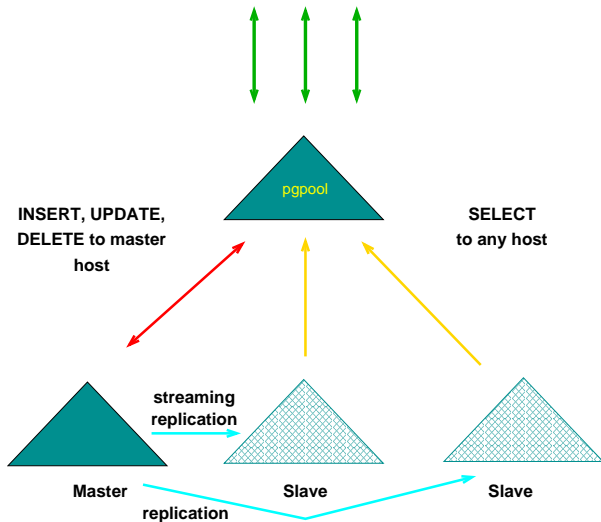


Loads tables and creates indexes using concurrent database connections

### 3. Multi-Host

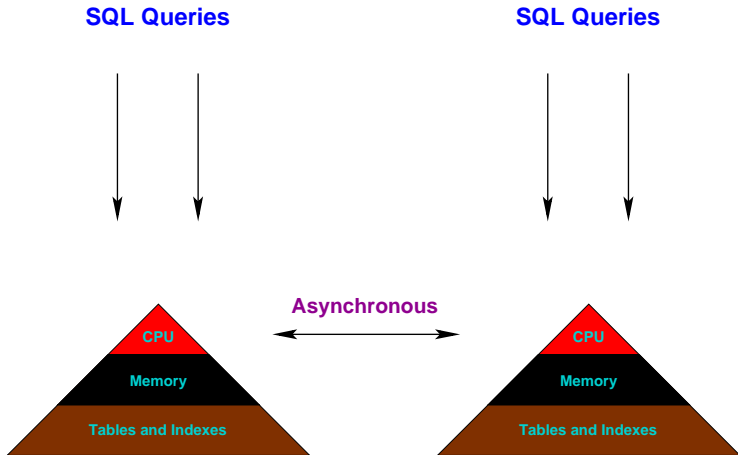


# Read Scaling Using Pgpool & Streaming Replication



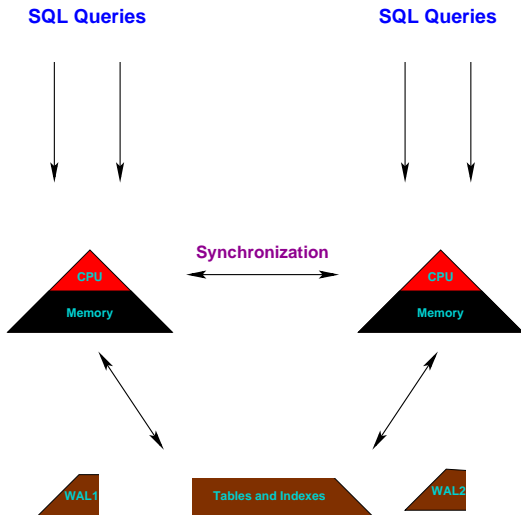
A full copy of the data exists on every node.

# CPU/Memory Scaling With Asynchronous Multi-Master



A full copy of the data exists on every node; requires conflict resolution. The asynchronous delay allows write-load buffering.

# Oracle Real Application Clusters (RAC)



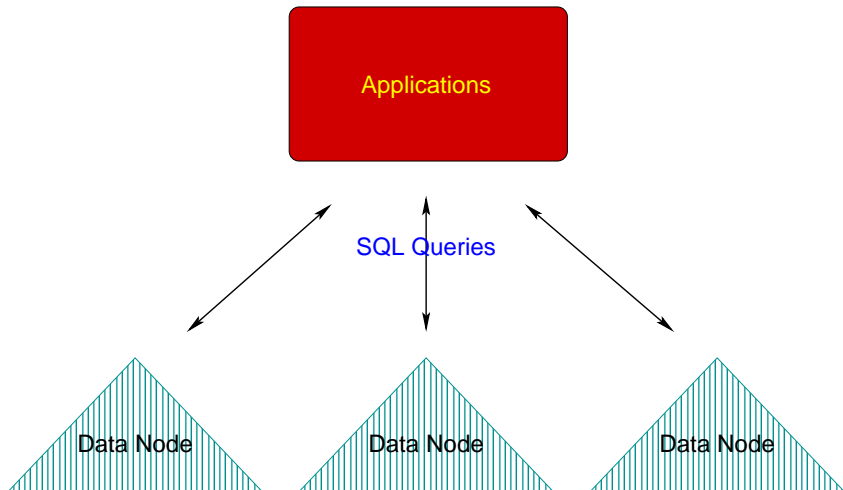
Tables and indexes on shared storage; inter-node synchronization required for cache consistency



# I/O Scaling with Sharding: Challenges

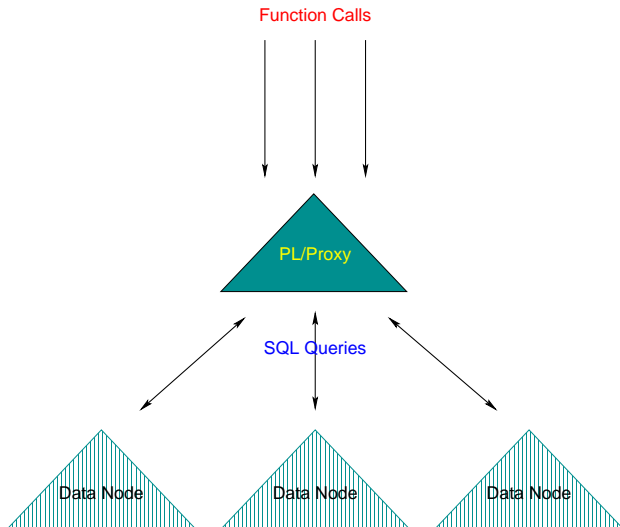
- ▶ Multi-host write queries require two-phase commit (except XC)
- ▶ Multi-host visibility snapshots are not supported (except XC)
- ▶ Sharding benefits are only possible with a shardable workload
- ▶ Changing the sharding layout can cause downtime
- ▶ Additional hosts reduce reliability; additional standby servers might be required

# Application-Based Sharding



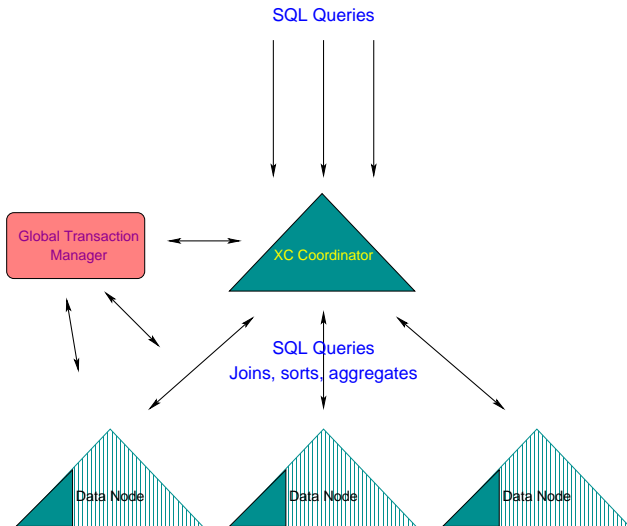
Applications send queries based on the sharding layout.

# Sharding Using PL/Proxy



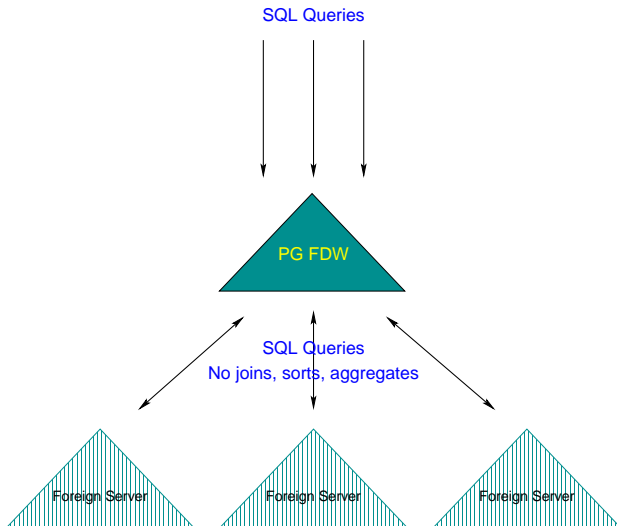
Requires rows to be hashed by key, supports parallel-node query execution

# Sharding Using Postgres-XC



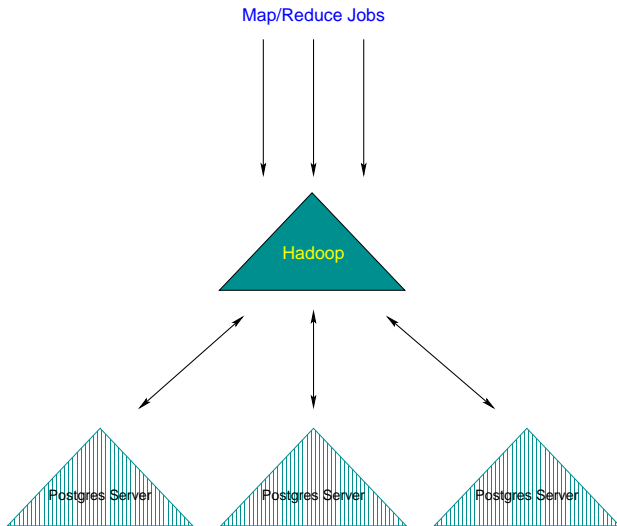
Enables hashing of large tables, replication of others; supports parallel-node consistent transactions and DDL

# Scaling Using Foreign Data Wrappers



Requires rows to be hashed by key

# Bulk Data Scaling Using Hadoop



# Conclusion



*<http://momjian.us/presentations>*

*<https://www.flickr.com/photos/87179607@N06/>*