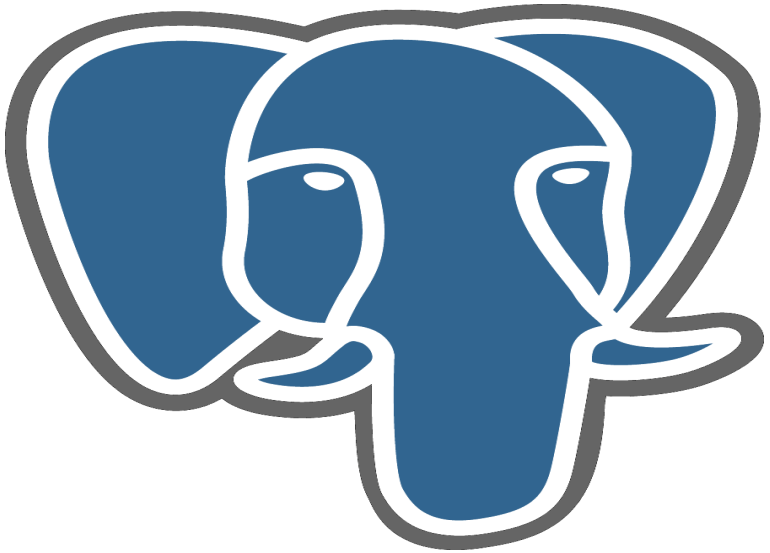
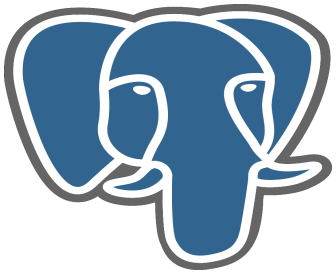
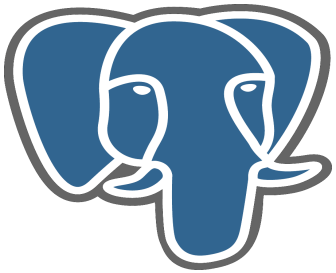
Future In-Core Replication for PostgreSQL





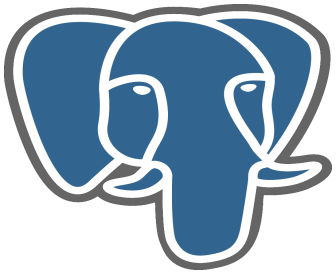
Agenda

- Replication Theory
- Architectures



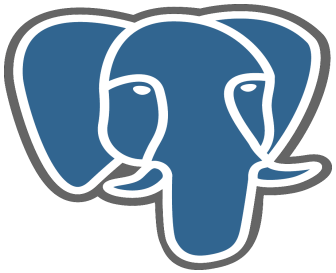
Aspects of Theory

- CAP Theorem
- Apply Efficiency
- Conflict Rate
- Replication Lag

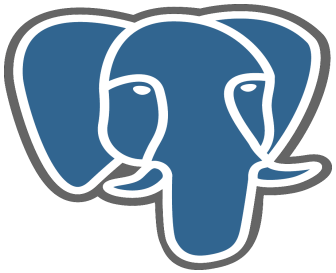


CAP Theorem

- Consistency
- Availability
- Partition Tolerance

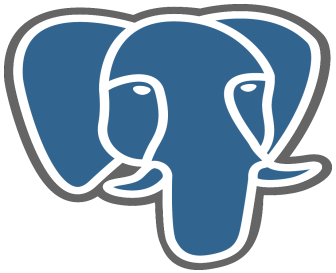


Multi-Master Efficiency Analysis



Efficiency Analysis of Apply

- For simplicity, lets assume that the cost of apply is always proportional to the cost of change...
- If we make changes on one node with cost W, and then apply those changes on another node the cost of those changes is **k**W
- If we have N nodes, then the cost to replay the changes from all other nodes will be (N-1)kW
- Total workload on any node is W + (N-1)kW
- If we define resource limit as 1, then total work from N nodes is N / (1 + k(N-1))

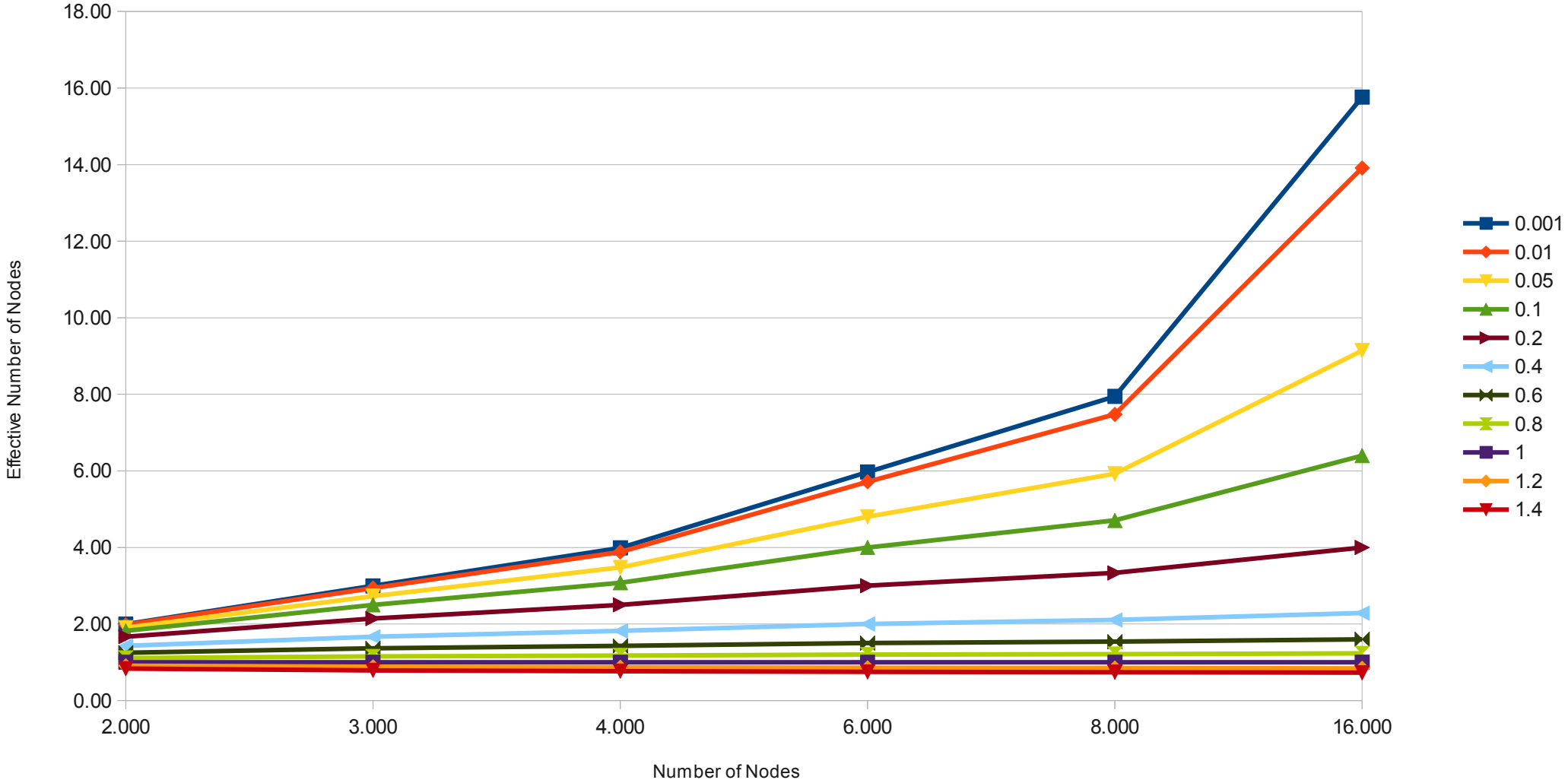


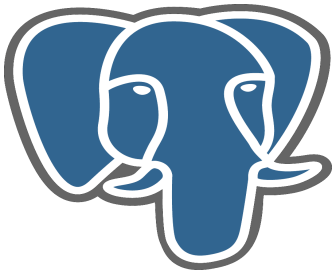
Apply Efficiency

- With 1 node we do work 1.0
- With 2 nodes we do work 2/(1+k)
  - k=1 means it takes exactly same effort to replay changes as it took to make original change
  - k=0.5 means it takes 50% effort to apply changes

k	2 nodes	4 nodes	16 nodes
0.10	1.67	2.86	6.15
0.30	1.25	1.82	2.76
0.50	1.00	1.33	1.78
0.70	0.83	1.00	1.31
1.00	0.67	0.80	0.94

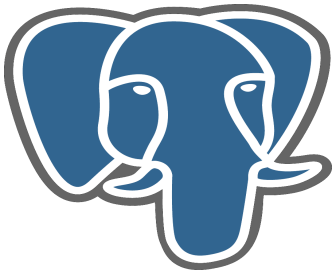
Graphs of Apply Efficiency





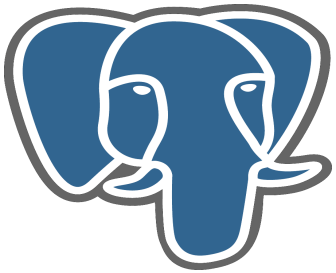
What is the Apply Constant, k?

- Physical replication allows changes directly to data blocks, so we avoid the need to search for keys via index searches, hence k < 1
- Read Scalability is possible with physical replication because k is typically about 0.5 and with only a single master, each node has spare capacity to do additional read workload
- Physical replication forces us to use just one node: multi-master required for write scalability
- Physical replication provides best read scalability



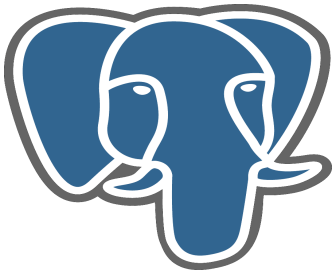
What is the Apply Constant, k?

- Logical replication uses PK values, so we must repeat the key search when we apply
  - Hard to see how k < 1 is possible in Executor
  - Can save effort in parser and planner, but likely that k is in the range 0.3 to 0.7
- As the number of nodes increases the cost of apply must also consider the additional costs of conflict resolution/avoidance
- => ***Full multi-master replication can't deliver write scalability on its own***

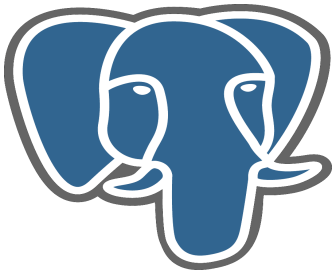


Filtered Replication

- If less than 100% of data needs to be replicated to other nodes, the equation changes
- N / (1 + kf(N-1)) where f is the filter factor
- If f = 0 this means no data changes need to be sent to other nodes (e.g. full sharding)
  - In that case total work from N nodes = N
  - If k >= 0.7 sharding becomes essential if we want write scalability from replication

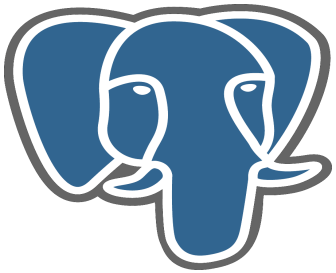


Other Theory



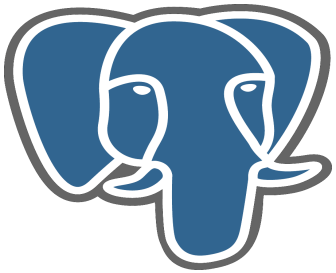
Conflict Rate

- Multi-master conflict rate increases as the cube of the row-level contention [Grey]
- Message delays/offline nodes increases the conflict rate
- High rates of conflict resolution dramatically increase the cost of apply
  - => conflicts reduce scalability



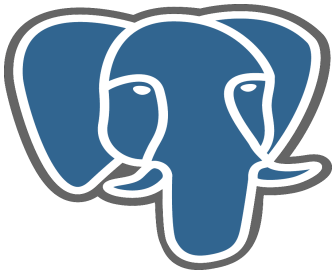
Replication Delay

- If Apply is faster than Master, then replication can always keep up
- If Master is faster than Apply then replication will fall behind and replication is useless
- Whenever we measure performance we must also measure replication delay – if delay begins to increase then we have reached the limit of steady state performance

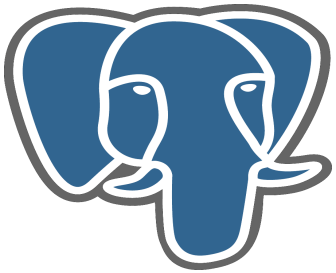


Technical Requirements

- Filtered Replication/Sharding
  - Filtering must take place ***on source***
  - Low values of f required for high scalability
- Highly efficient LCRs/Apply
  - Low values of k required
  - InCore approaches to apply gain importance

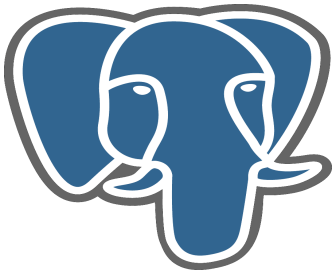


Existing Approaches



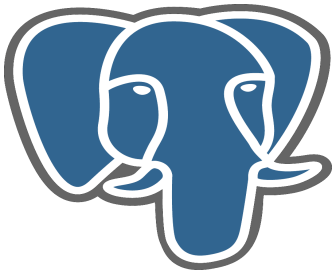
Streaming Replication

- Currently based on binary WAL
  - Efficient because WAL is available “for free”
  - Efficient apply using direct physical addresses
- Uses libpq so provides full security model
- Efficient use of protocol, with 2-way messaging
- Connection parameters, timeouts
- Modular code, becoming battle hardened



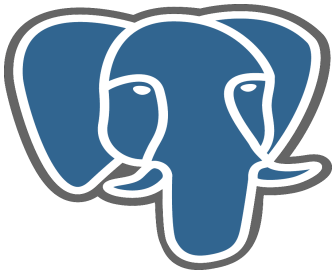
Downsides of Physical Rep

- Using WAL means we reuse all the recovery code, which puts various restrictions
- No xids, No oids, No new WAL
- Master/Standby connected or causes cancels
- Futures Issues
  - Harder to replicate just part of a database
  - Harder to make cross-version replication work for online upgrade



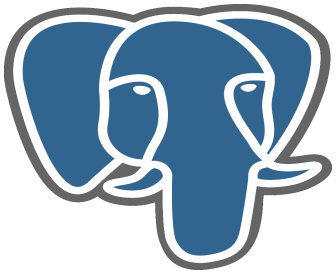
Slony/Londiste/Bucardo

- Been around a while
- We know they work
- Online upgrades
- Writes allowed on target systems
- Schemas can be slightly different
- Indexes/admin can differ on each node



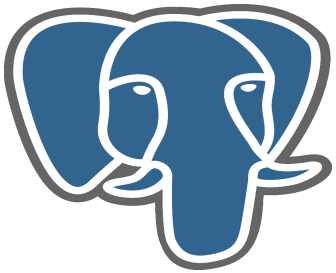
Downsides

- No filtering within a table
- Not part of core
- Many moving parts
- Low performance
- 3+ forks means effort is dissipated
- DDL support

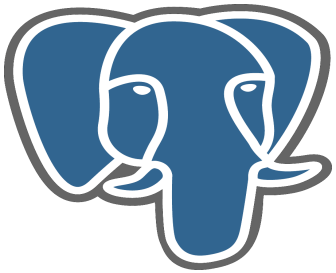


Postgres-XC

- Sharded cluster provides write scalability
- Massive changes to codebase
  - Likely to remain a fork for some time
  - Customers perceive support issues
- Solves single issue only
- No support for widely/geo distributed database



Requirements

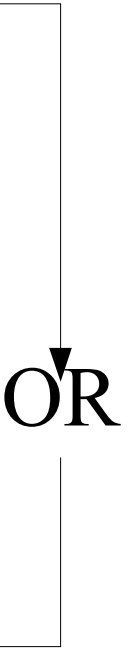


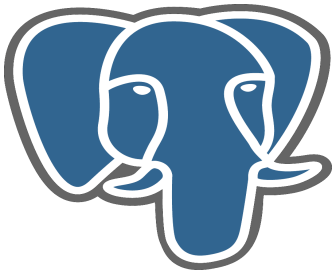
Future Requirements

- In-Core
- Write Scalable
- Geographically Distributed
- Multimaster
- Coherent
- Online Upgrade
- DDL

Generic Replication Model

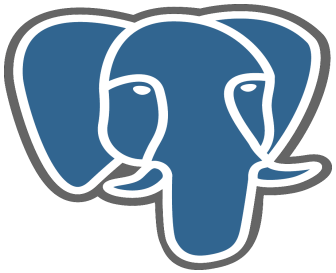
- 1. Establish who is the master
- 2. Generate change messages
- 3. Apply local change
- 4. Transport replication messages
- 5. Apply replicated changes
- 6. Handle conflicts





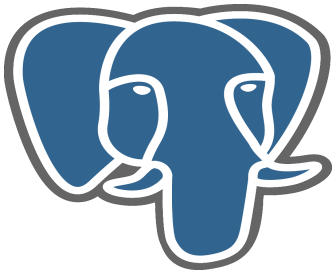
Single Master Replication

- 1. By definition, one master
- 2. Reuse WAL, so only minor additional msgs
- 3. Apply local change
- 4. Streaming Replication
- 5. Server in recovery
- 6. Query conflicts/connected to master



Replication Transport

- What can we achieve if *we assume that the physical transport of messages stay same*, only changes are above that layer?
- => Step4 is already complete
- Each target has one source only
- Allows 2 servers in a pair
- More generically, allows multiple servers arranged in a circle

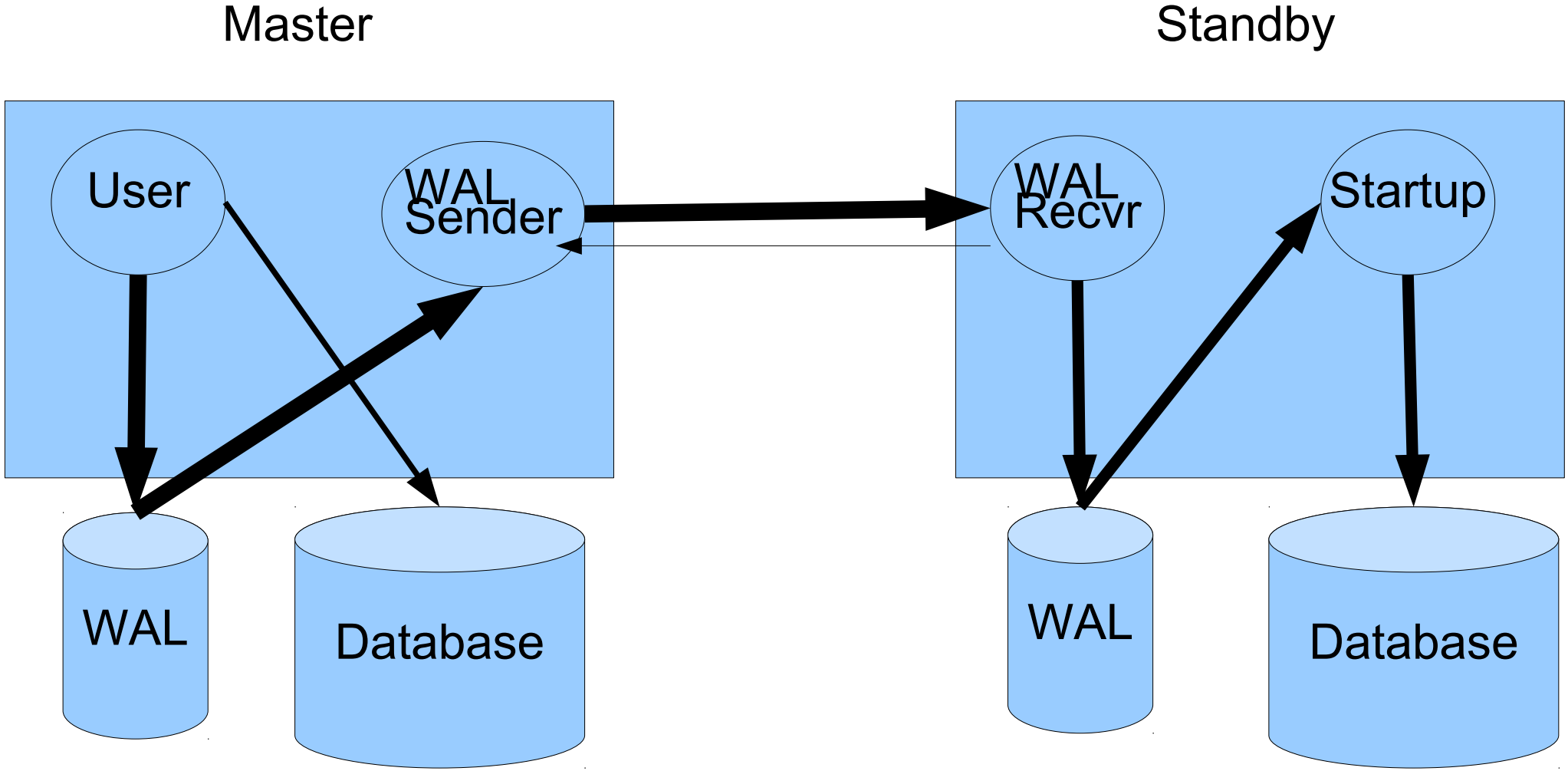


Generic Logical Replication

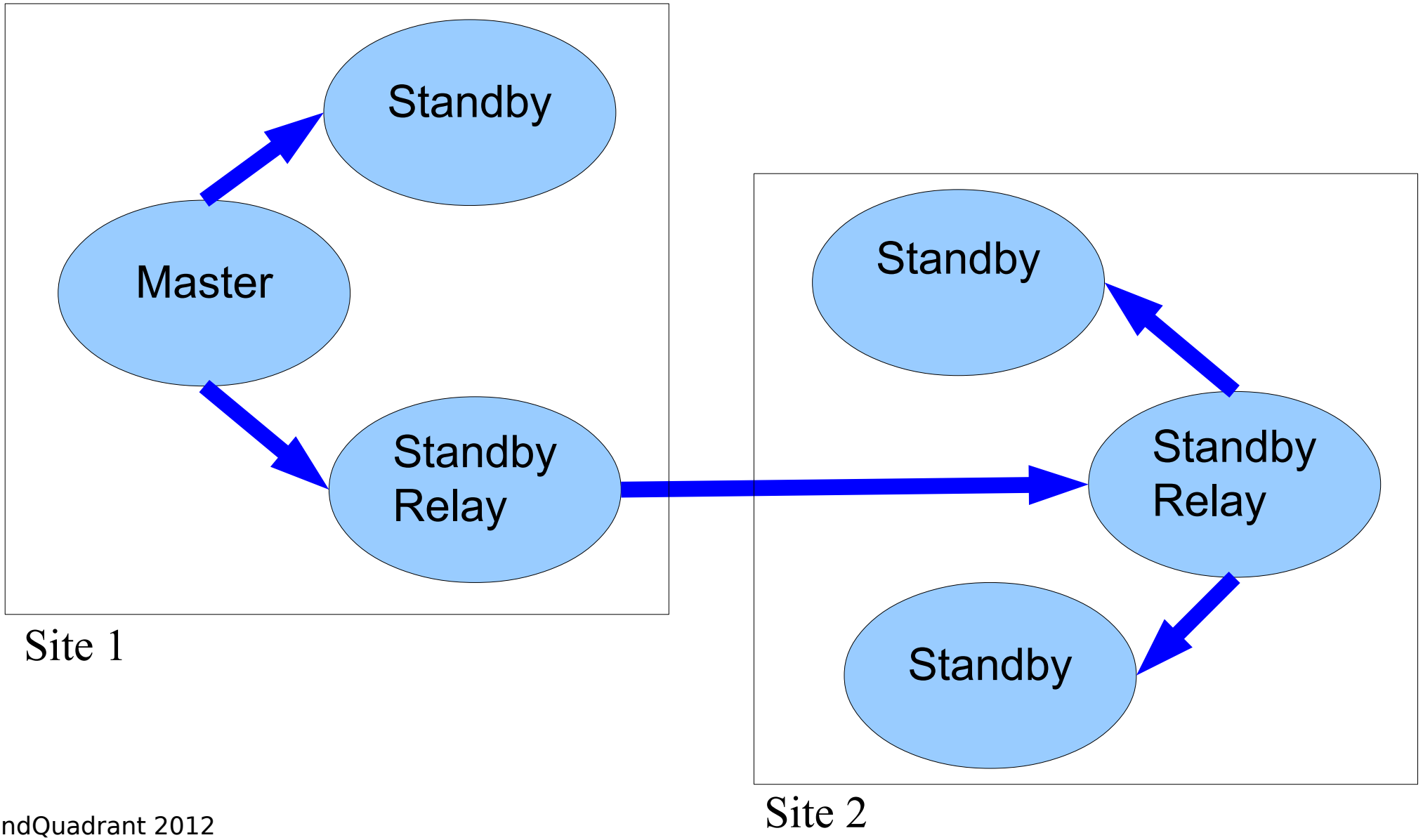
- 1. EstablishMaster
- 2. Generate Logical Change Records (LCRs)
- 3. [Apply local change]
- 4. [Transport replication messages]
- 5. Apply
- 6. Conflicts

- [already have this code]

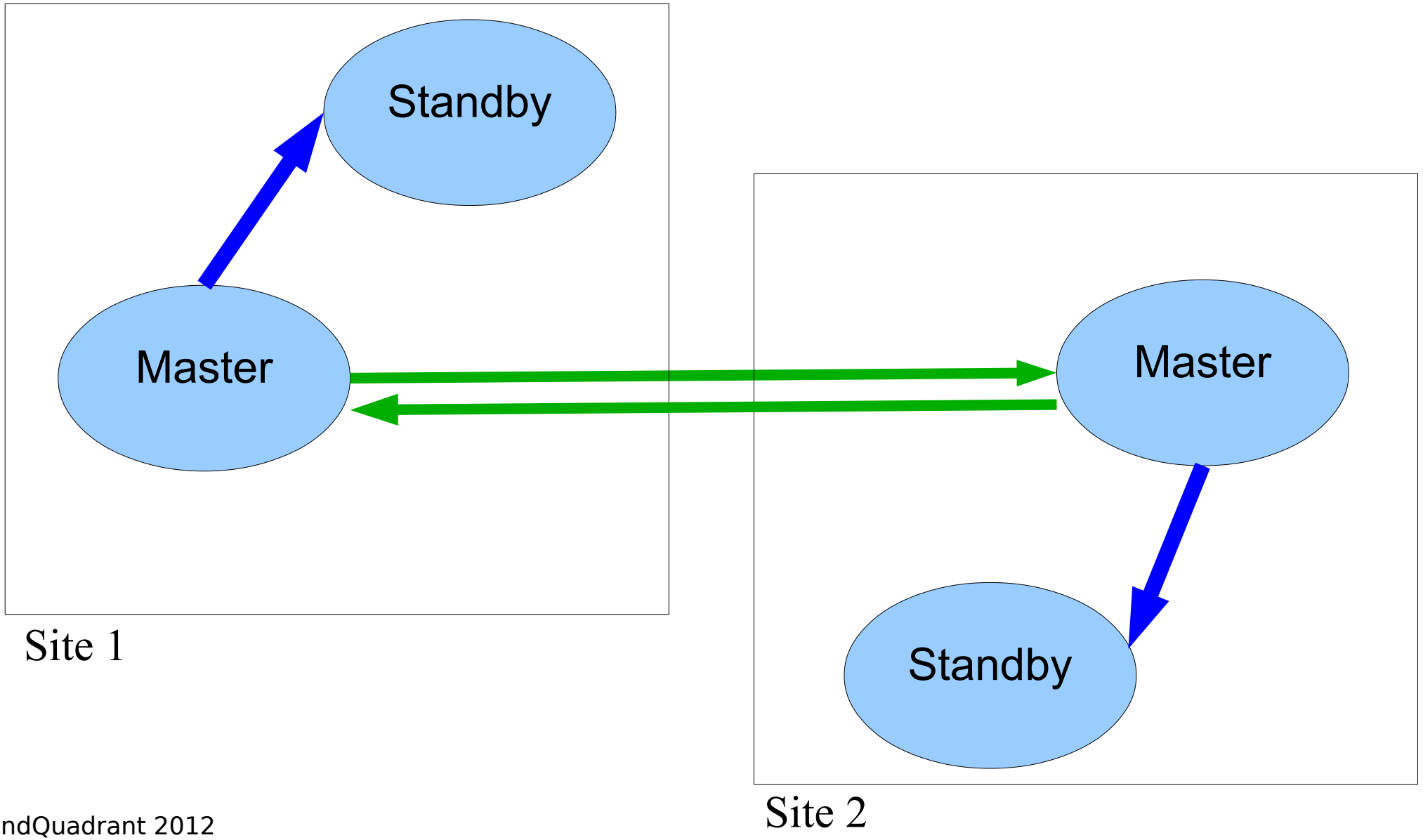
Physical Streaming Replication



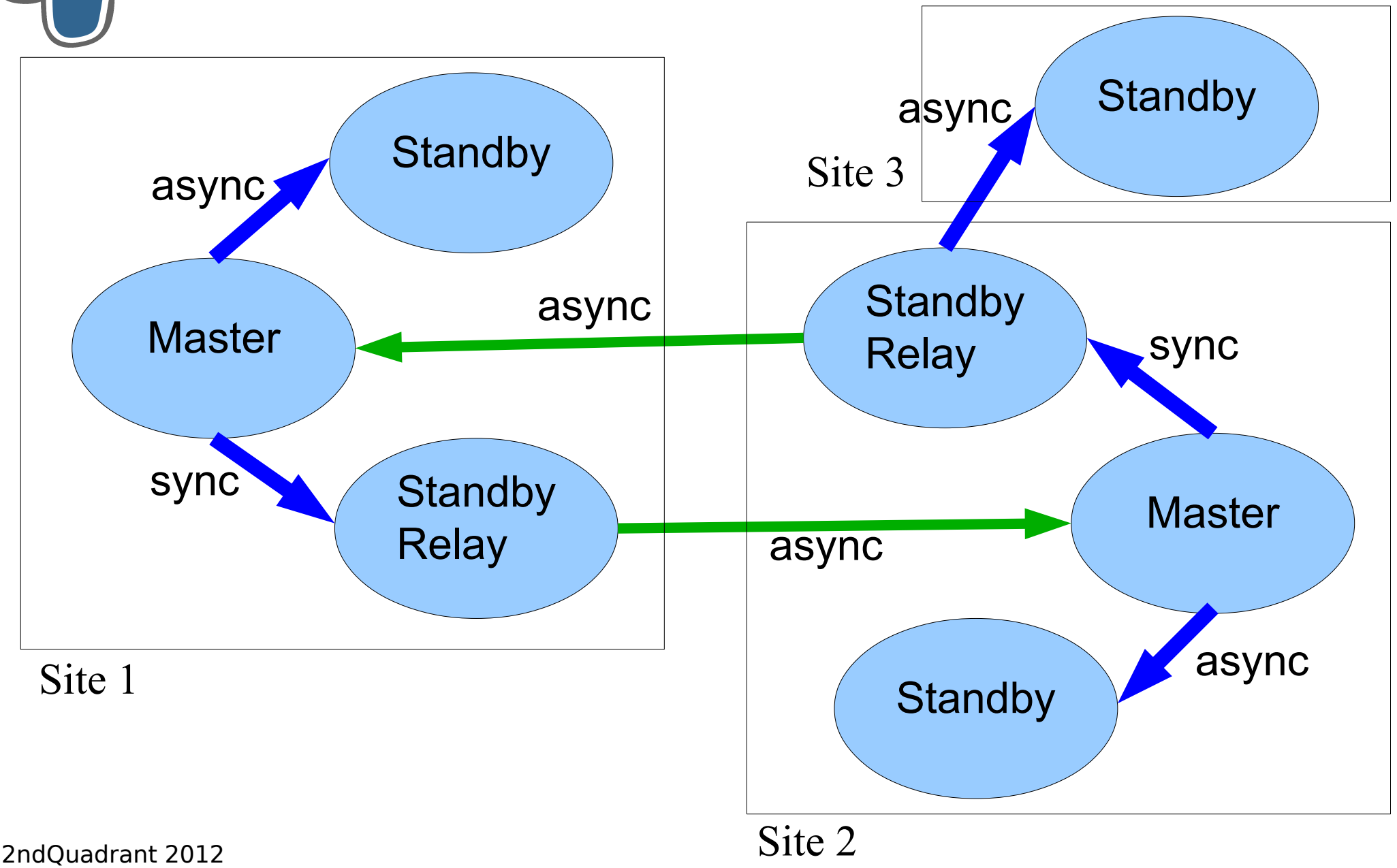
Physical Streaming Topologies



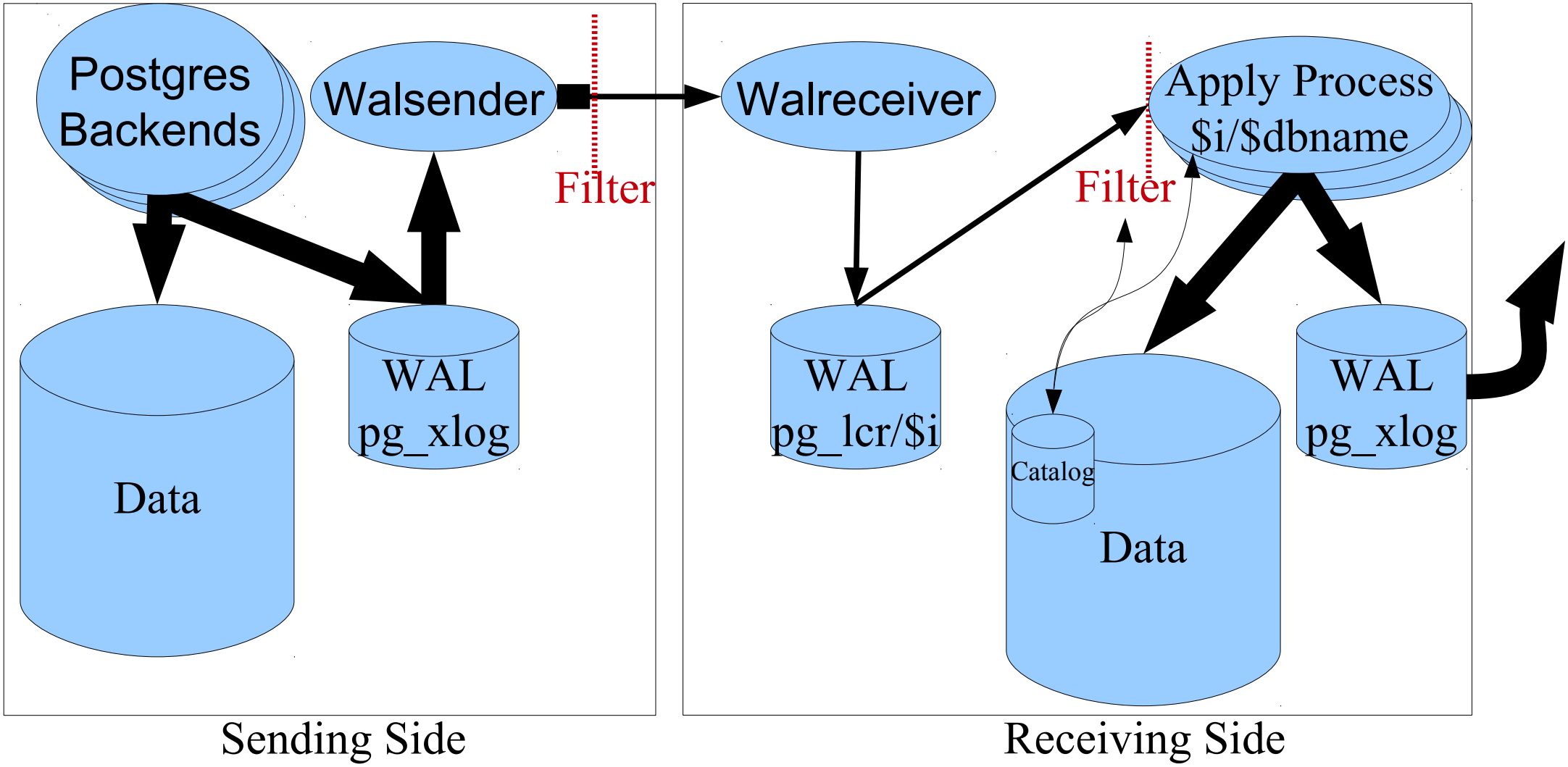
Logical Streaming Topologies



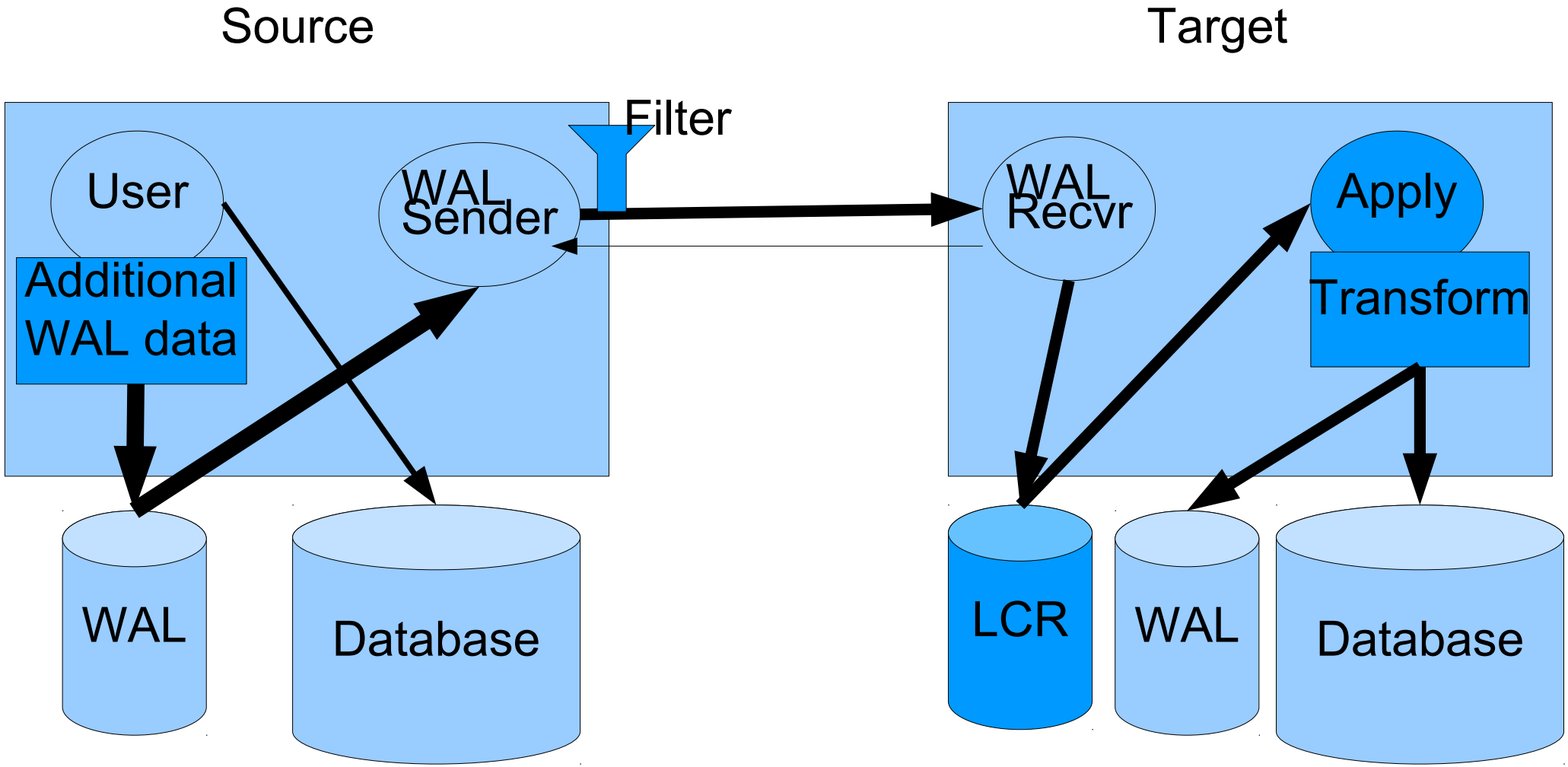
Mixed Streaming Topologies

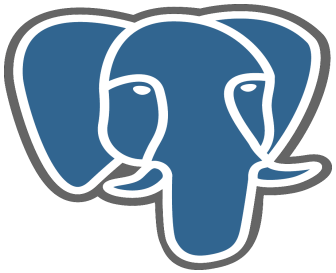


Logical Streaming Replication



Logical Streaming Replication

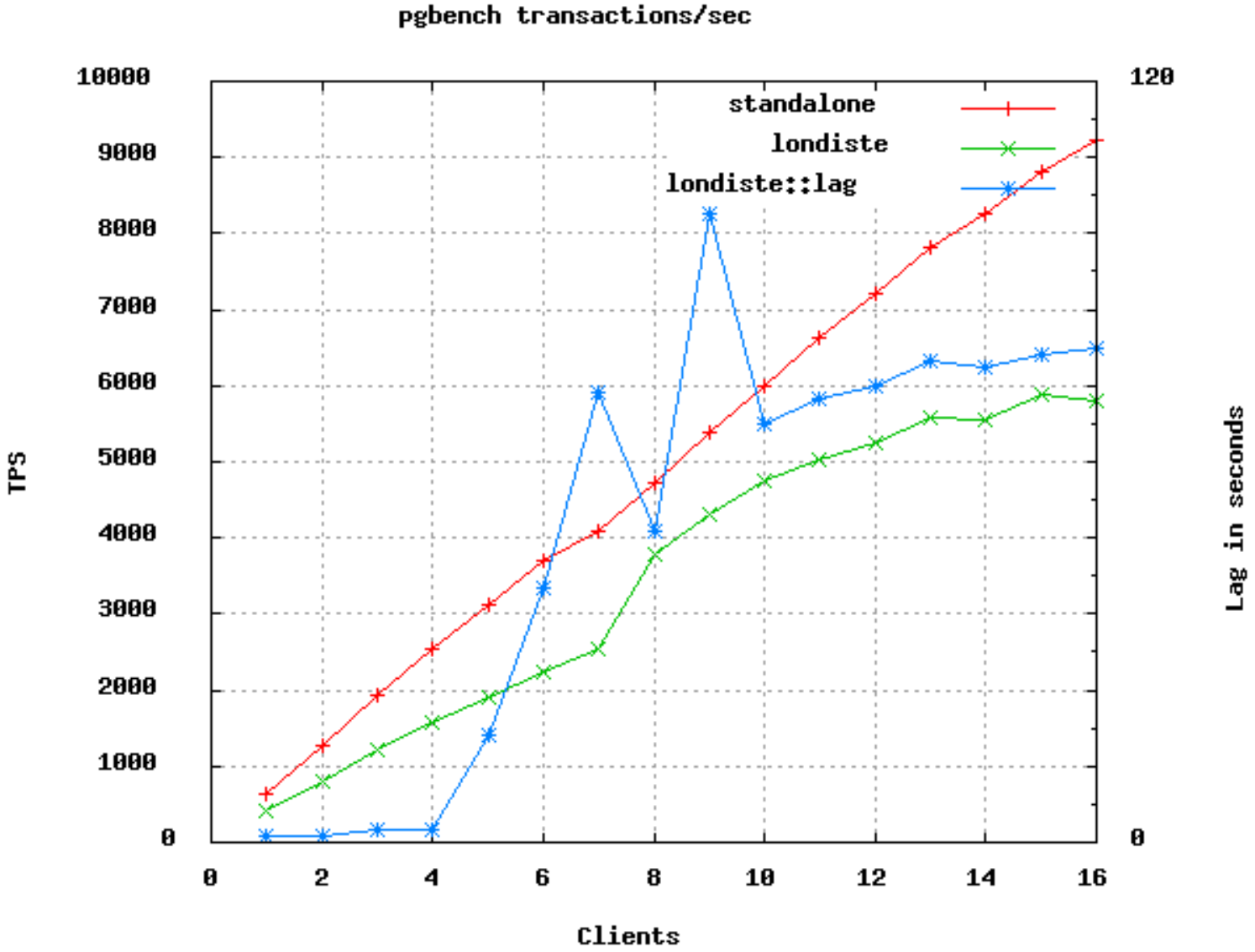


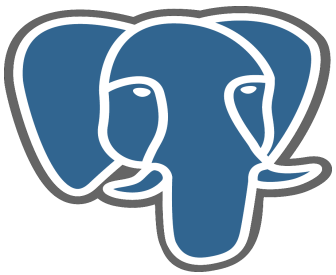


Benchmark

- Modified pgbench tpc-b (+\sleep 1 ms)
- checkpoint_completion_target = 0.9
- checkpoint_segments = 300
- shared_buffers = 4GB
- 2 Servers * Xeon E3-1275 (4 cores), 16GB Ram, 4xSAS (15k disk), Gigabit Ethernet
- 9.2 git @ c2cc5c347 (6 Weeks)

Logical Replication Today





Logical Replication Tomorrow