Slides available at *http://sequoia.continuent.org/Resources*

PostgreSQL replication strategies

Understanding High Availability and choosing the right solution

emmanuel.cecchet@continuent.com

emmanuel.cecchet@epfl.ch

continuent
Open. Always Available.

What Drives Database Replication?

- **Availability – Ensure applications remain up and running when there are hardware/software failures as well as during scheduled maintenance on database hosts**
- **Read Scaling – Distribute queries, reports, and I/O-intensive operations like backup, e.g., on media or forum web sites**
- **Write Scaling – Distribute updates across multiple databases, for example to support telco message processing or document/web indexing**
- **Super Durable Commit – Ensure that valuable transactions such as financial or medical data commit to multiple databases to avoid loss**
- **Disaster Recovery – Maintain data and processing resources in a remote location to ensure business continuity**
- **Geo-cluster – Allow users in different geographic locations to use a local database for processing with automatic synchronization to other hosts**

High availability

/ **The magic nines**

Percent uptime	Downtime/month	Downtime/year
99.0%	**7.2 hours**	**3.65 days**
99.9%	**43.2 minutes**	**8.76 hours**
99.99%	**4.32 minutes**	**52.56 minutes**
99.999%	0.43 minutes	5.26 minutes
99.9999%	**2.6 seconds**	**31 seconds**

Few definitions

/ MTBF

- Mean Time Between Failure
- Total MTBF of a cluster must combine MTBF of its individual components
- Consider mean-time-between-system-abort (MTBSA) or mean-time-between-critical-failure (MTBCF)

/ MTTR

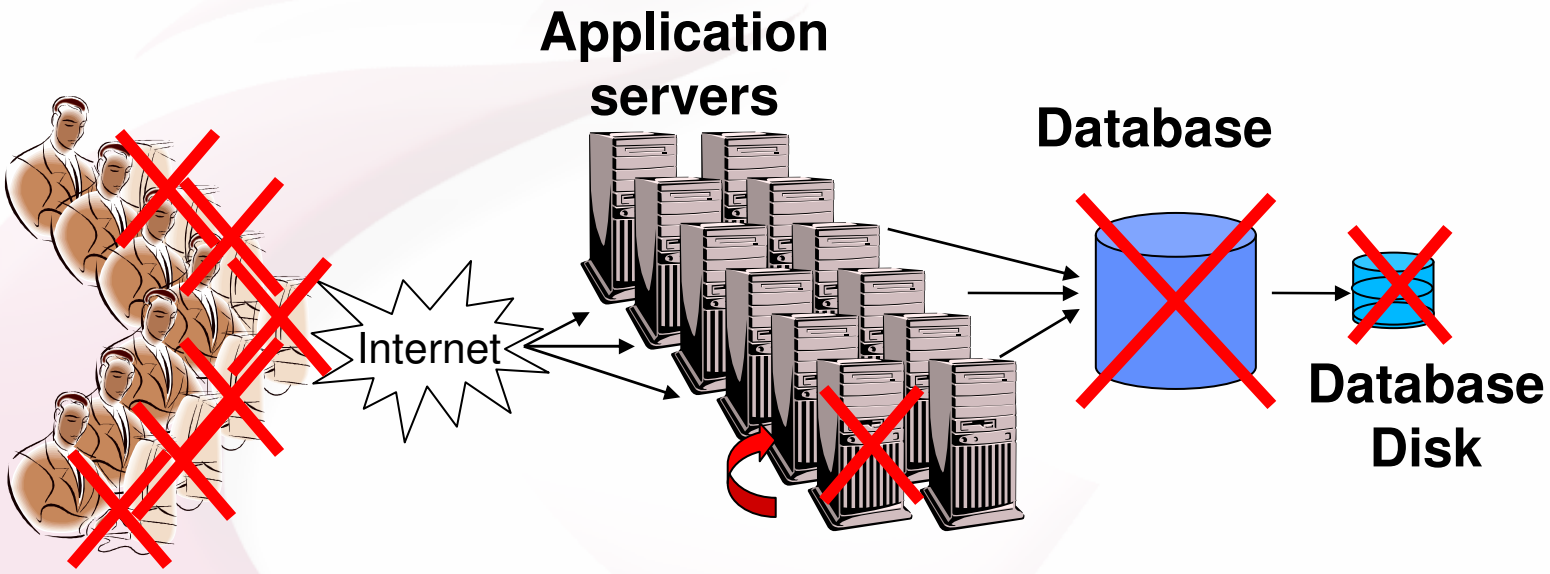
- Mean Time To Repair
- How is the failure detected?
- How is it notified?
- Where are the spare parts for hardware?
- What does your support contract say?

Outline

- Database replication strategies
- PostgreSQL replication solutions
- Building HA solutions
- Management issues in production

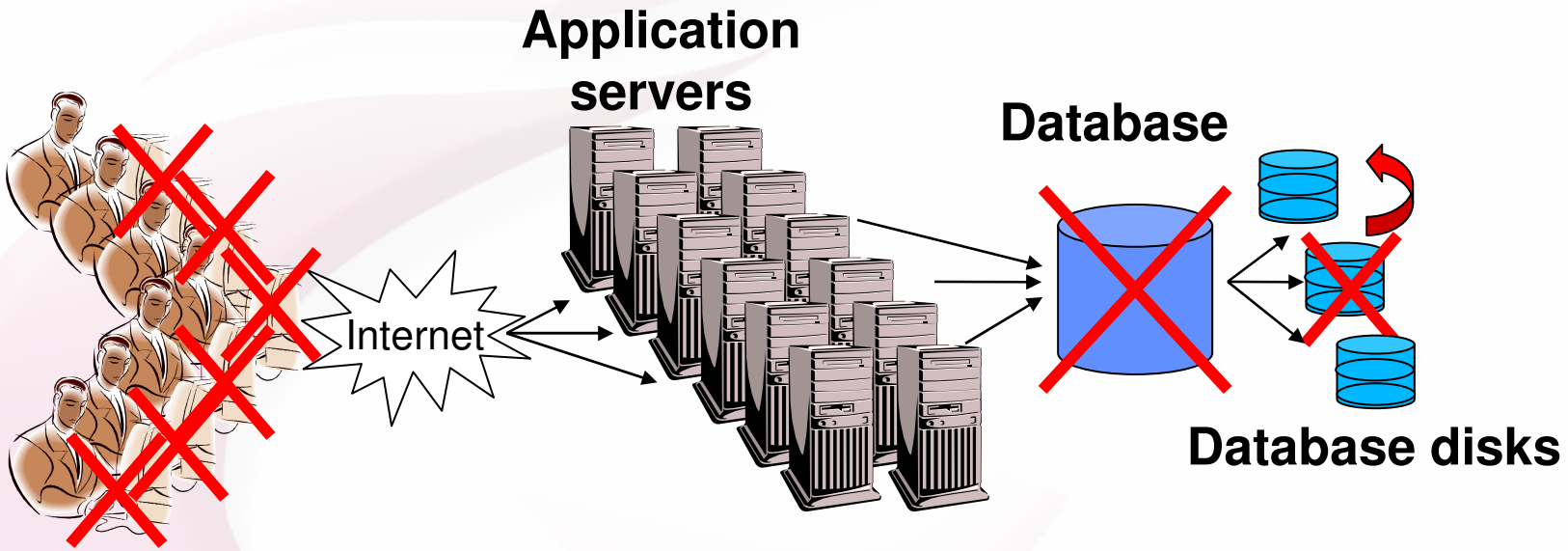
Problem: Database is the weakest link

- Clients connect to the application server
- Application server builds web pages with data coming from the database
- Application server clustering solves application server failure
- Database outage causes overall system outage



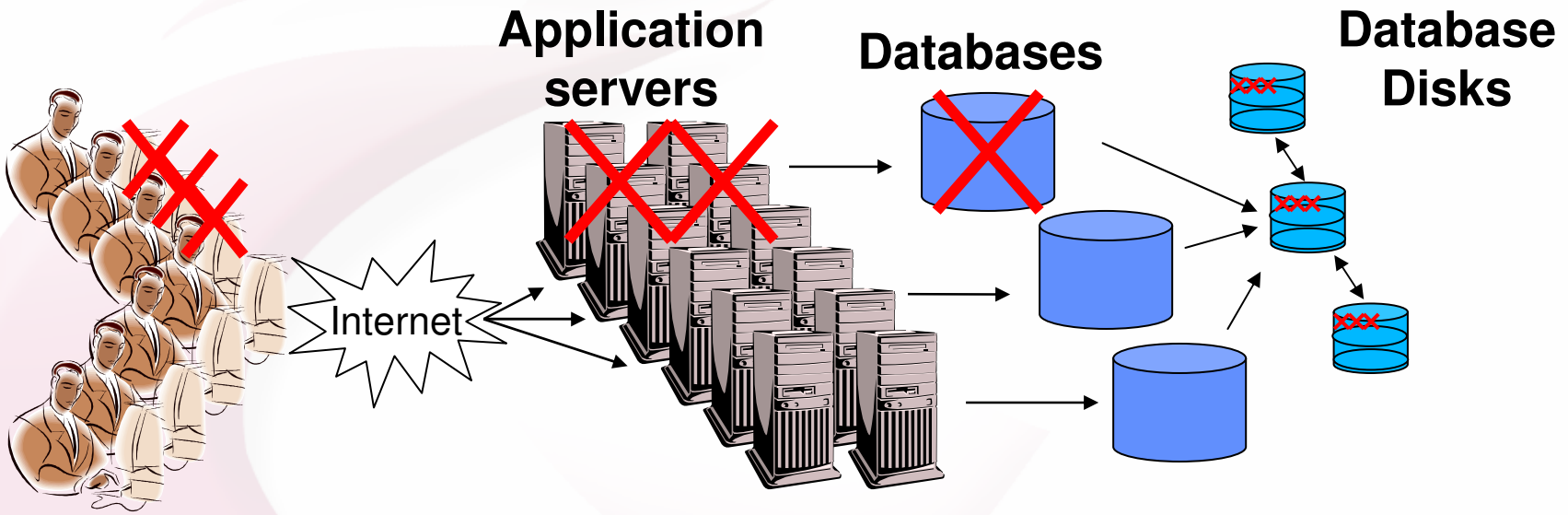
Disk replication/clustering

- Eliminates the single point of failure (SPOF) on the disk
- Disk failure does not cause database outage
- Database outage problem still not solved



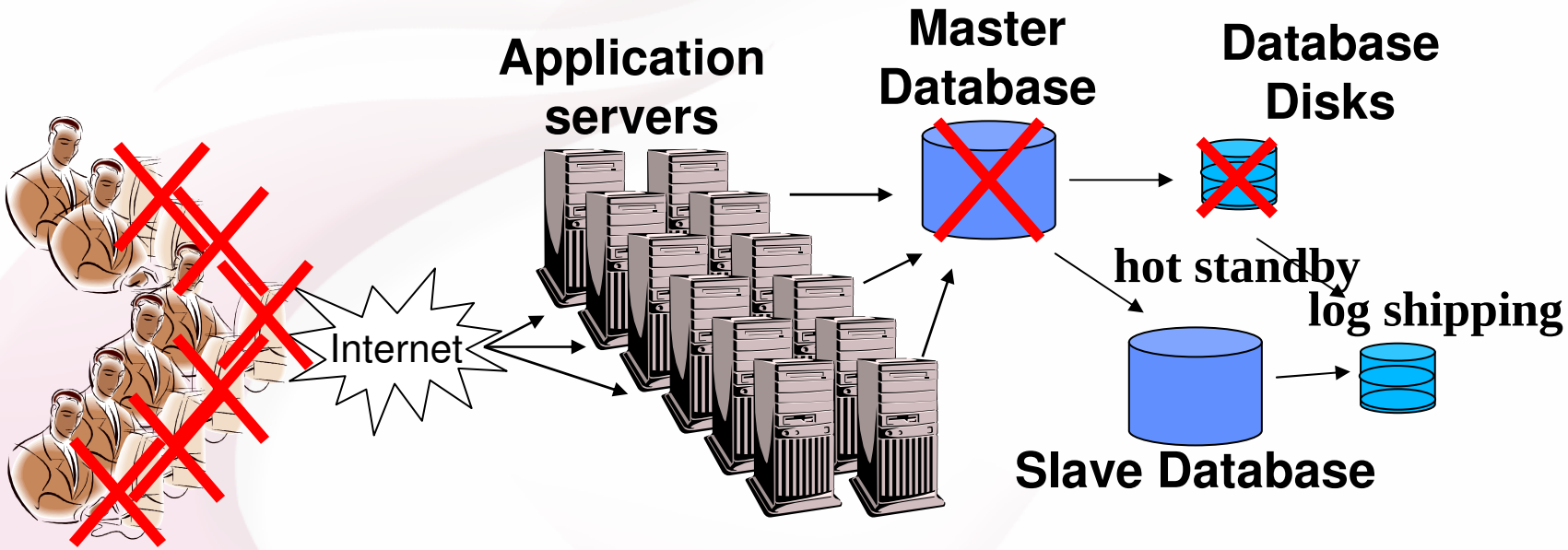
Database clustering with shared disk

- Multiple database instances share the same disk
- Disk can be replicated to prevent SPOF on disk
- No dynamic load balancing
- Database failure not transparent to users (partial outage)
- Manual failover + manual cleanup needed



Master/slave replication

- Lazy replication at the disk or database level
- No scalability
- Data lost at failure time
- System outage during failover to slave
- Failover requires client reconfiguration



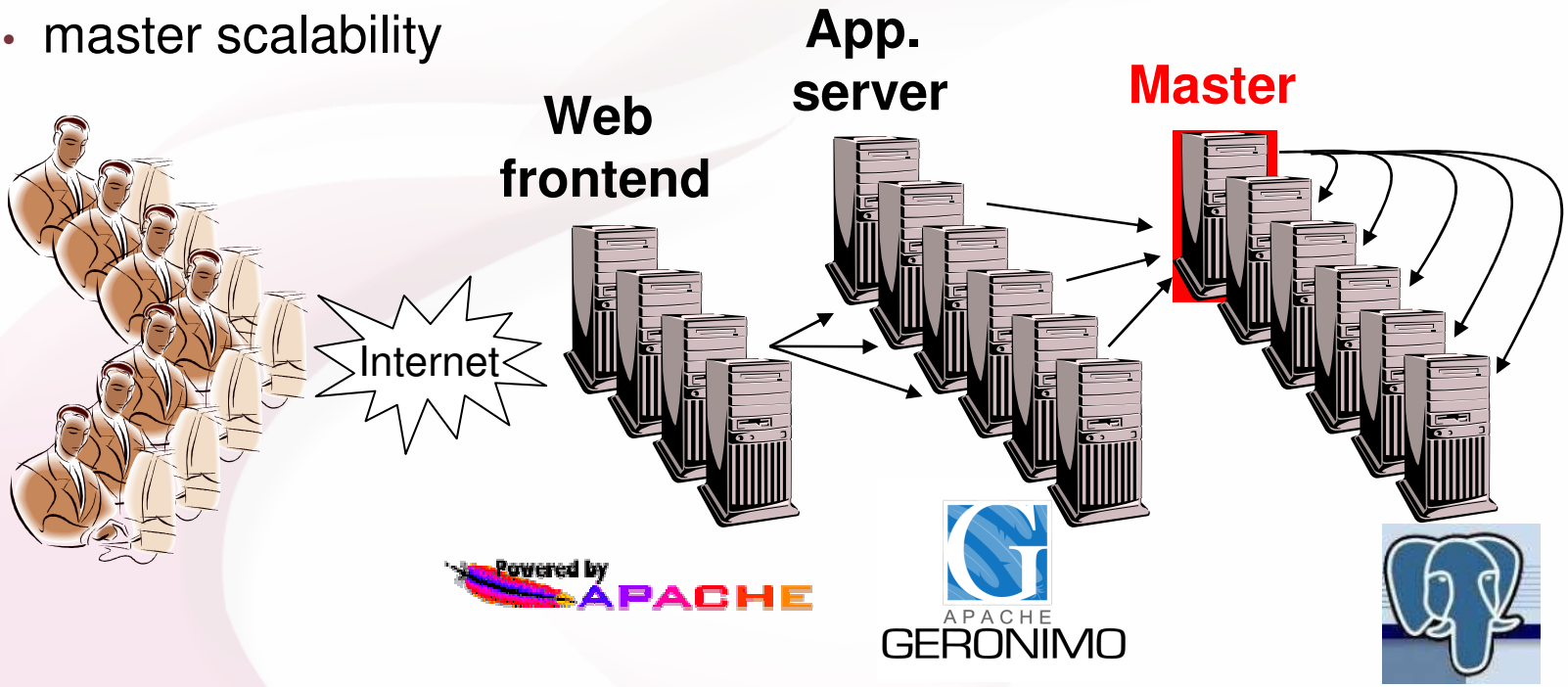
Scaling the database tier
Master-slave replication

/ Pros

- Good solution for disaster recovery with remote slaves

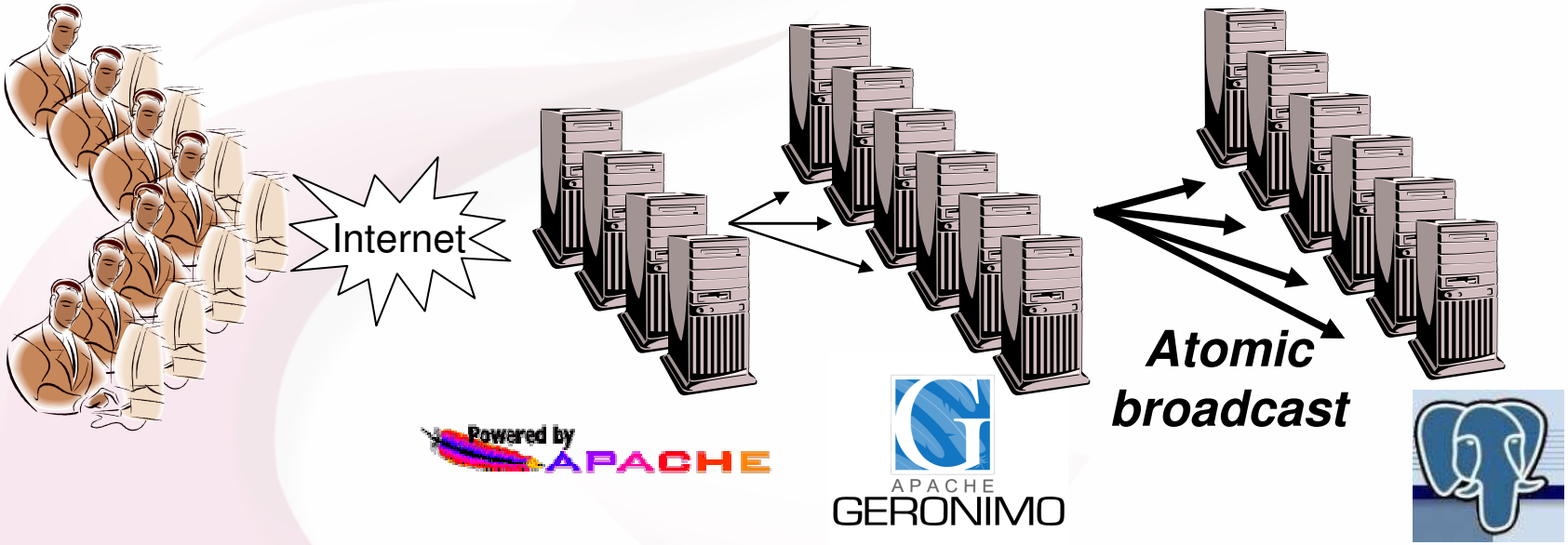
/ Cons

- failover time/data loss on master failure
- read inconsistencies
- master scalability



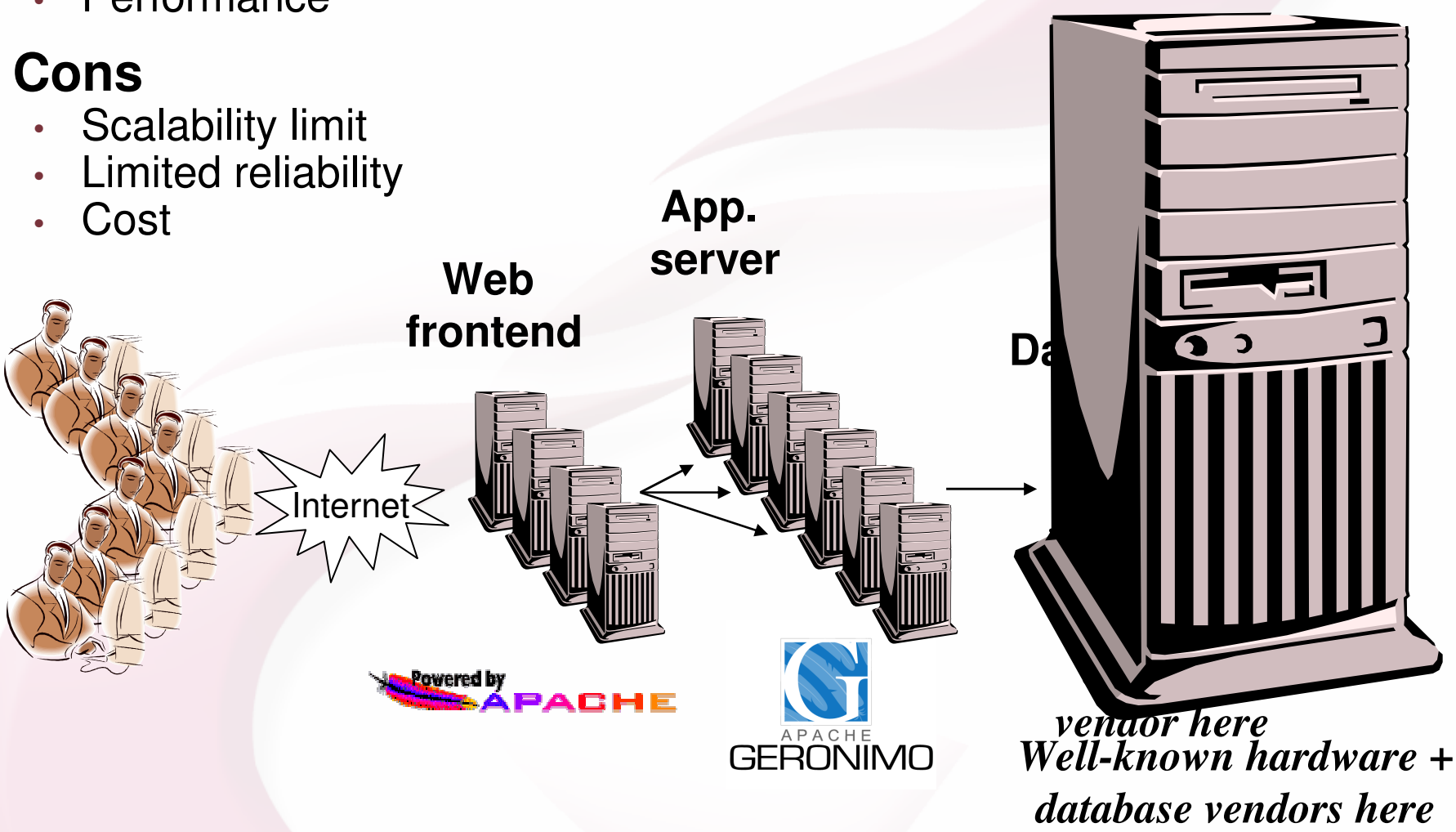
Scaling the database tier
Atomic broadcast

- **Pros**
  - consistency provided by multi-master replication
- **Cons**
  - atomic broadcast scalability
  - no client side load balancing
  - heavy modifications of the database engine



Scaling the database tier – SMP

- **Pros**
  - Performance
- **Cons**
  - Scalability limit
  - Limited reliability
  - Cost



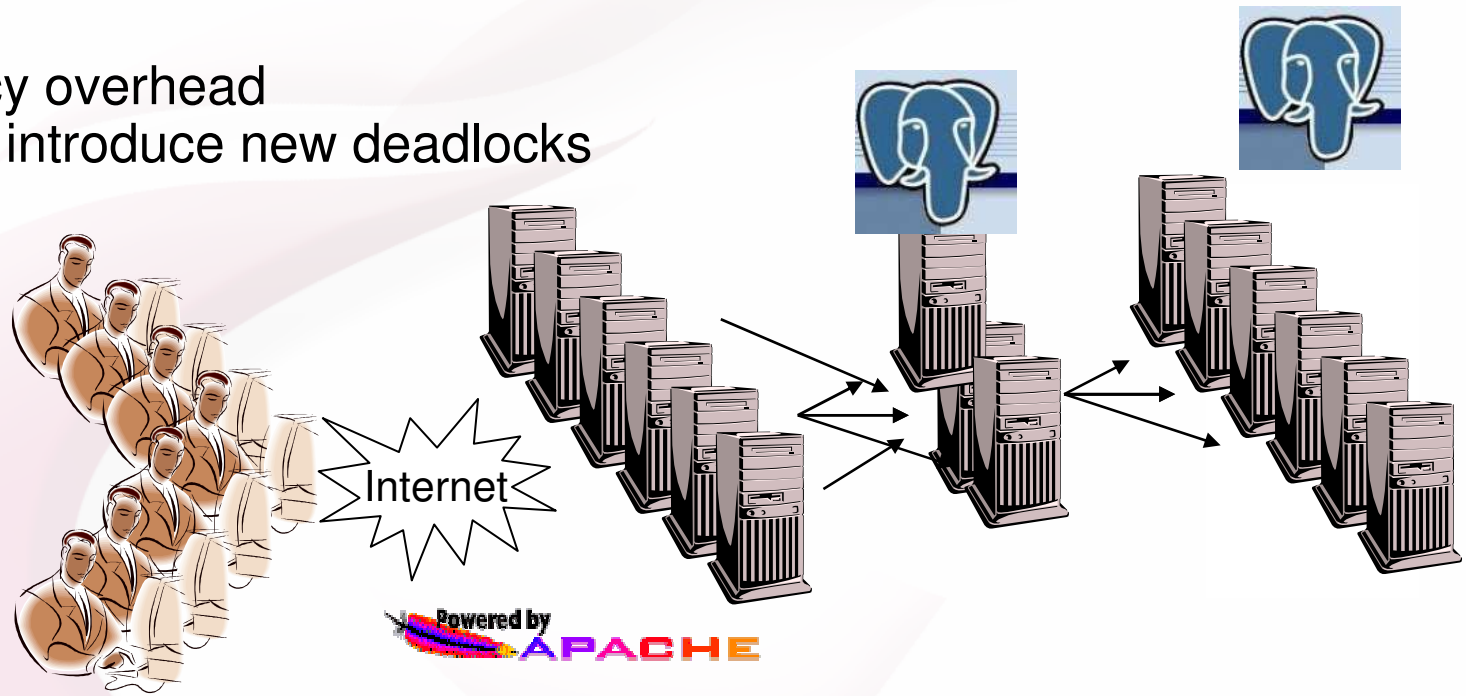
Middleware-based replication

/ Pros

- no client application modification
- database vendor independent
- heterogeneity support
- pluggable replication algorithm
- possible caching

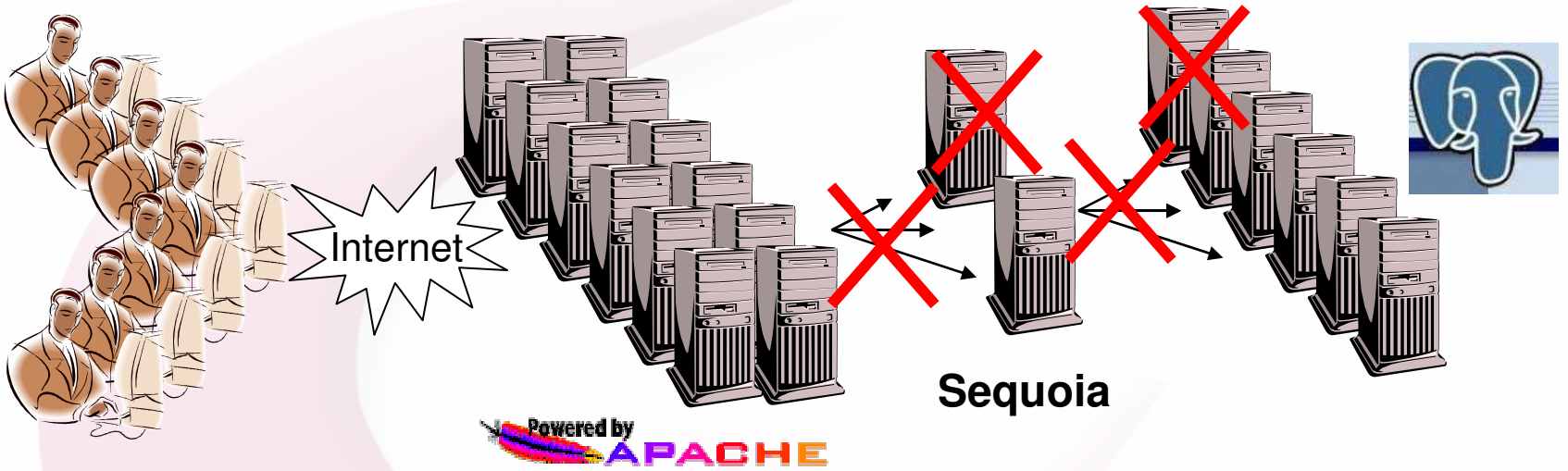
/ Cons

- latency overhead
- might introduce new deadlocks



Transparent failover

- **Failures can happen**
  - in any component
  - at any time of a request execution
  - in any context (transactional, autocommit)
- **Transparent failover**
  - masks all failures at any time to the client
  - perform automatic retry and preserves consistency



Outline

- Database replication strategies
- PostgreSQL replication solutions
- Building HA solutions
- Management issues in production

PostgreSQL replication solutions compared

Feature	pgpool-I	pgpool-II	PGcluster-I	PGcluster-II	Slony-I	Sequoia
Replication type	**Hot standby**	**Multi-master**	**Multi-master**	**Shared disk**	**Master/Slave**	**Multi-master**
Commodity hardware	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
Application modifications	**No**	**No**	**Yes if reading from slaves**	**No**	**Yes if reading from slaves**	**Client driver update**
Database modifications	**No**	**No**	**Yes**	**Yes**	**No**	**No**
PG support	**>=7.4 Unix**	**>=7.4 Unix**	**7.3.9, 7.4.6, 8.0.1 Unix**	**8.? Unix only?**	**>= 7.3.3**	**All versions**
Data loss on failure	**Yes**	**Yes?**	**No?**	**No**	**Yes**	**No**
Failover on DB failure	**Yes**	**Yes**	**Yes**	**No if due to disk**	**Yes**	**Yes**
Transparent failover	**No**	**No**	**No**	**No**	**No**	**Yes**
Disaster recovery	**Yes**	**Yes**	**Yes**	**No if disk**	**Yes**	**Yes**
Queries load balancing	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

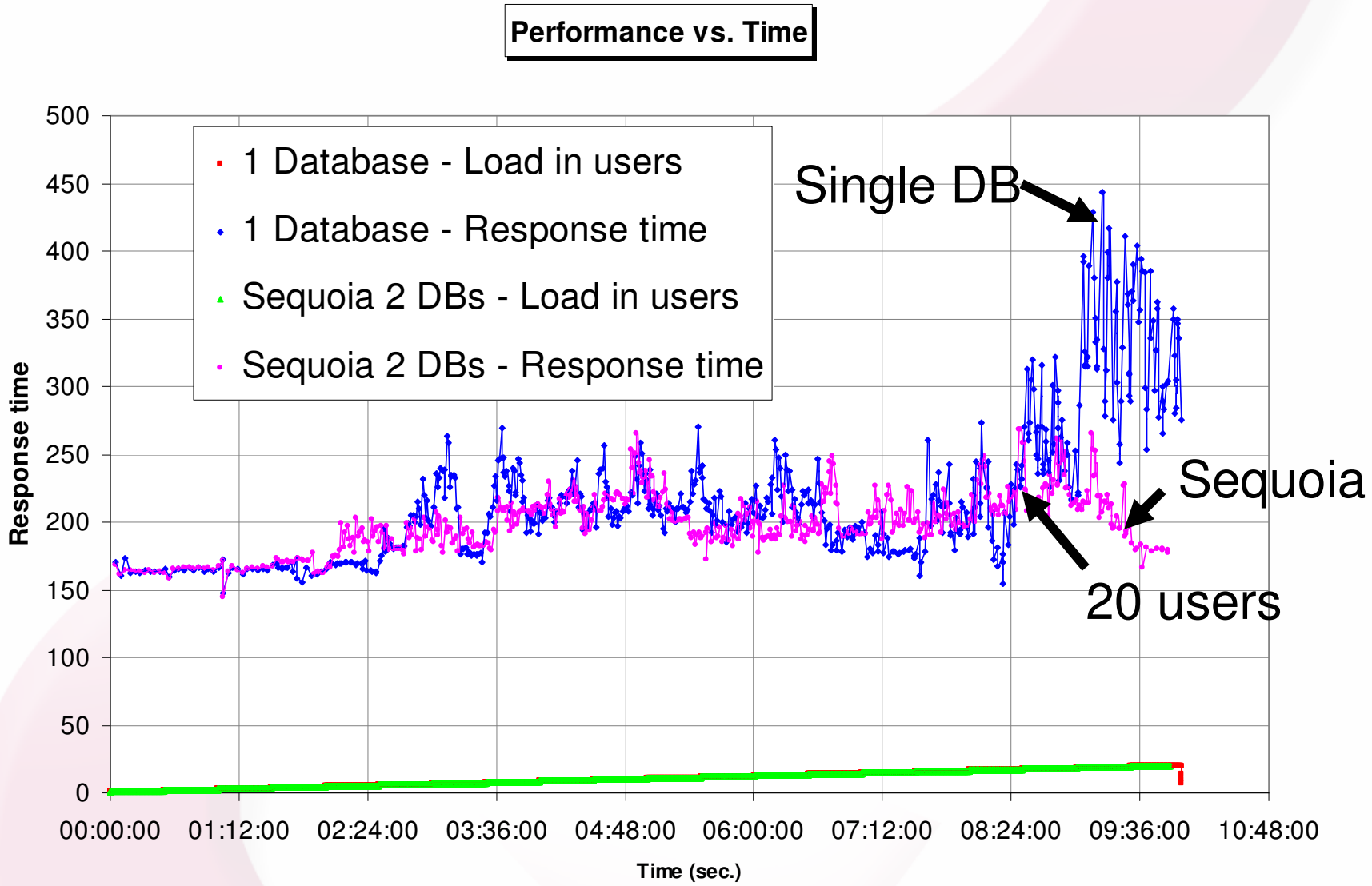
PostgreSQL replication solutions compared

Feature	pgpool-I	pgpool-II	PGcluster-I	PGcluster-II	Slony-I	Sequoia
Read scalability	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Write scalability	**No**	**No**	**No**	**Yes**	**No**	**No**
Query parallelization	**No**	**Yes**	**No**	**No?**	**No**	**No**
Replicas	**2**	**up to 128**	**LB or replicator limit**	**SAN limit**	**unlimited**	**unlimited**
Super durable commit	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**
Add node on the fly	**No**	**Yes**	**Yes**	**Yes**	**Yes (slave)**	**Yes**
Online upgrades	**No**	**No**	**Yes**	**No**	**Yes (small downtime)**	**Yes**
Heterogeneous clusters	**PG >=7.4 Unix only**	**PG >=7.4 Unix only**	**PG**	**PG**	**PG>=7.3.3**	**Yes**
Geo-cluster support	**No**	**No**	**Possible but don't use**	**No**	**Yes**	**Yes**

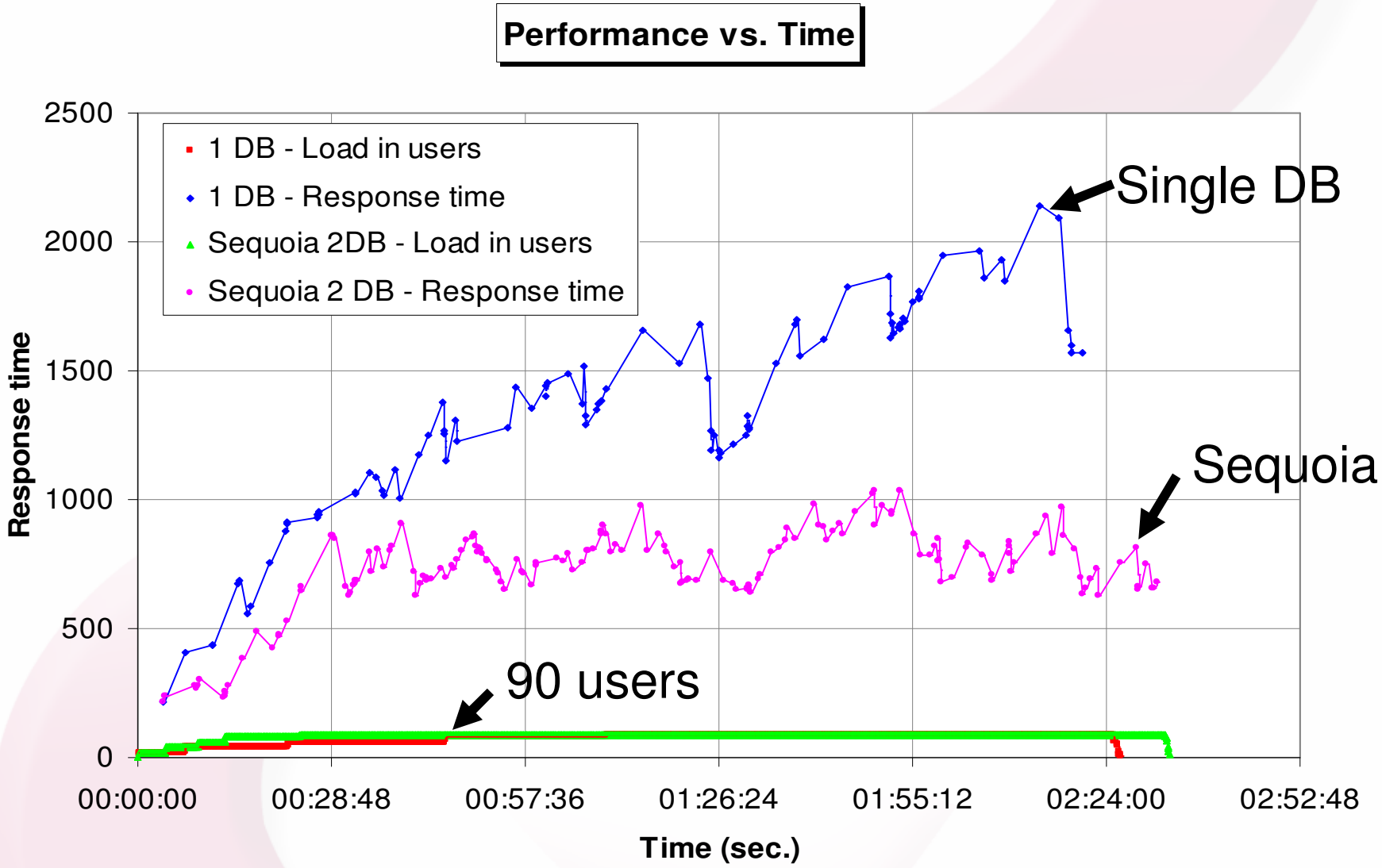
Performance vs Scalability

- **Performance**
  - latency different from throughput
- **Most solutions don't provide parallel query execution**
  - No parallelization of query execution plan
  - Query do not go faster when database is not loaded
- **What a perfect load distribution buys you**
  - Constant response time when load increases
  - Better throughput *when* load surpasses capacity of a single database

Understanding scalability (1/2)

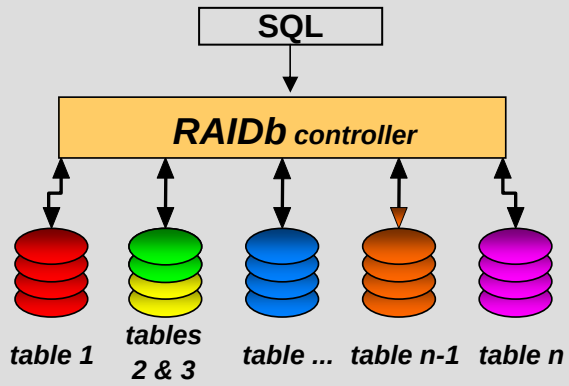


Understanding scalability (2/2)



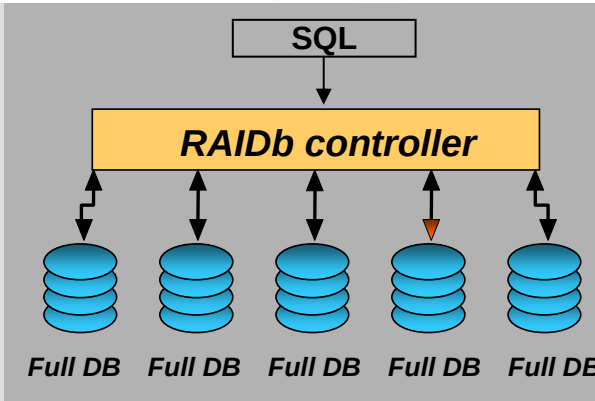
RAIDb Concept: Redundant Array of Inexpensive Databases

- RAIDb controller – creates single virtual db, balances load
- RAIDb 0,1,2: various performance/fault tolerance tradeoffs
- New combinations easy to implement



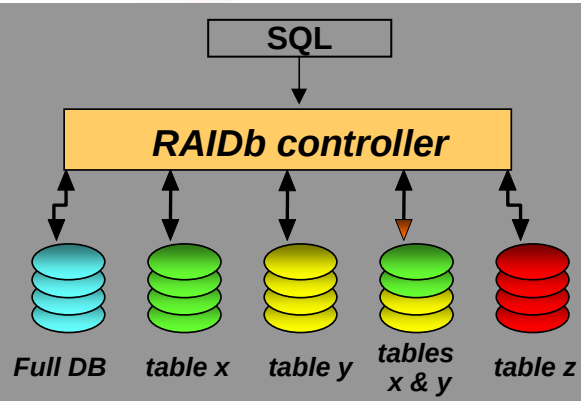
RAIDb-0

- partitioning (whole tables)
- no duplication
- no fault tolerance
- at least 2 nodes



RAIDb-1

- mirroring
- performance bounded by write broadcast
- at least 2 nodes
- uni/cluster certifies only RAIDb-1

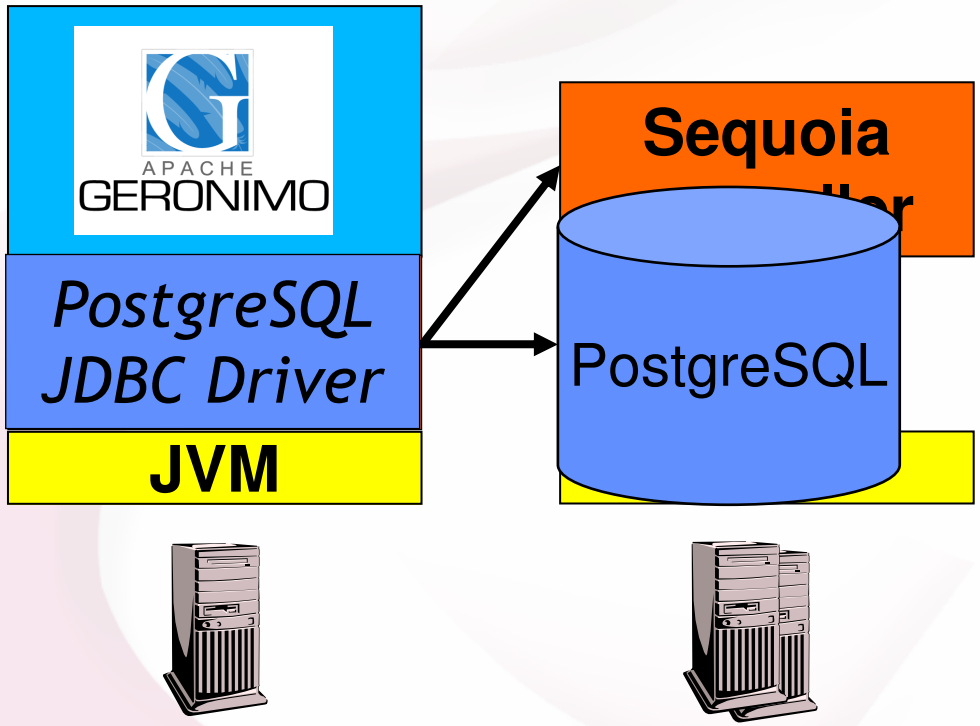


RAIDb-2

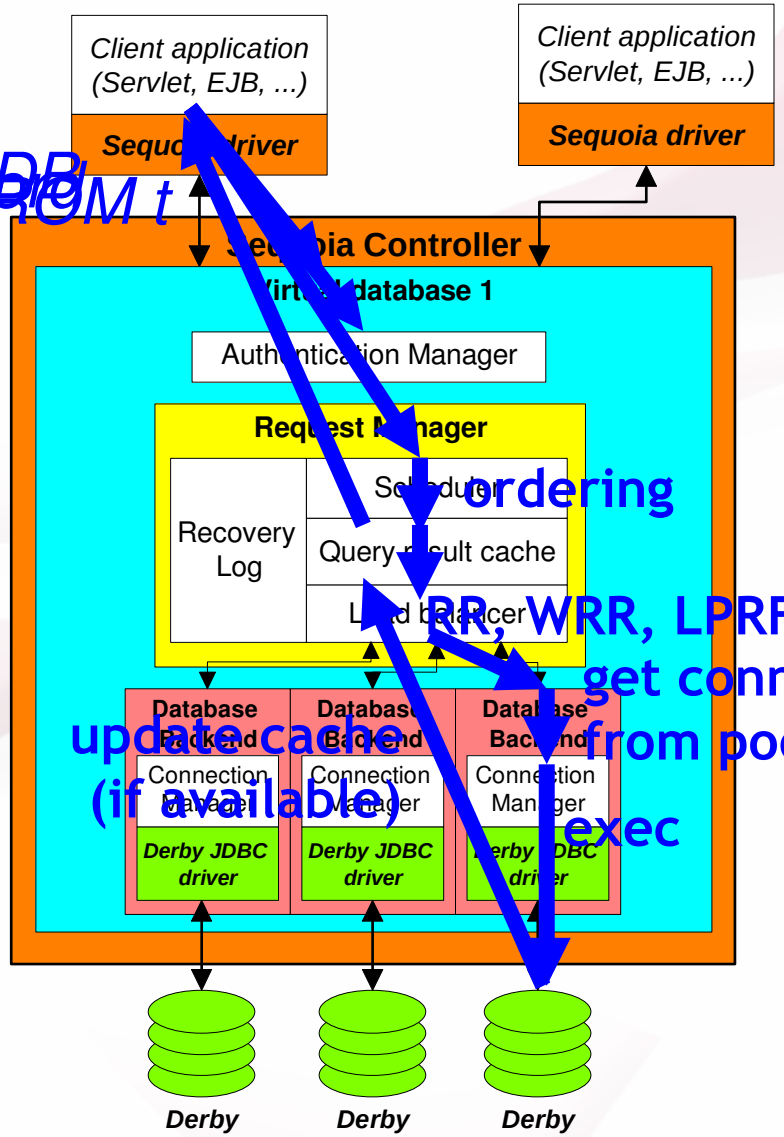
- partial replication
- at least 2 copies of each table for fault tolerance
- at least 3 nodes

Sequoia architectural overview

- **Middleware implementing RAIDb**
  - 100% Java implementation
  - open source (Apache v2 License)
- **Two components**
  - Sequoia driver (JDBC, ODBC, native lib)
  - Sequoia Controller
- **Database neutral**



Sequoia read request



connect connect myDB

execute SELECT * FROM t

ordering

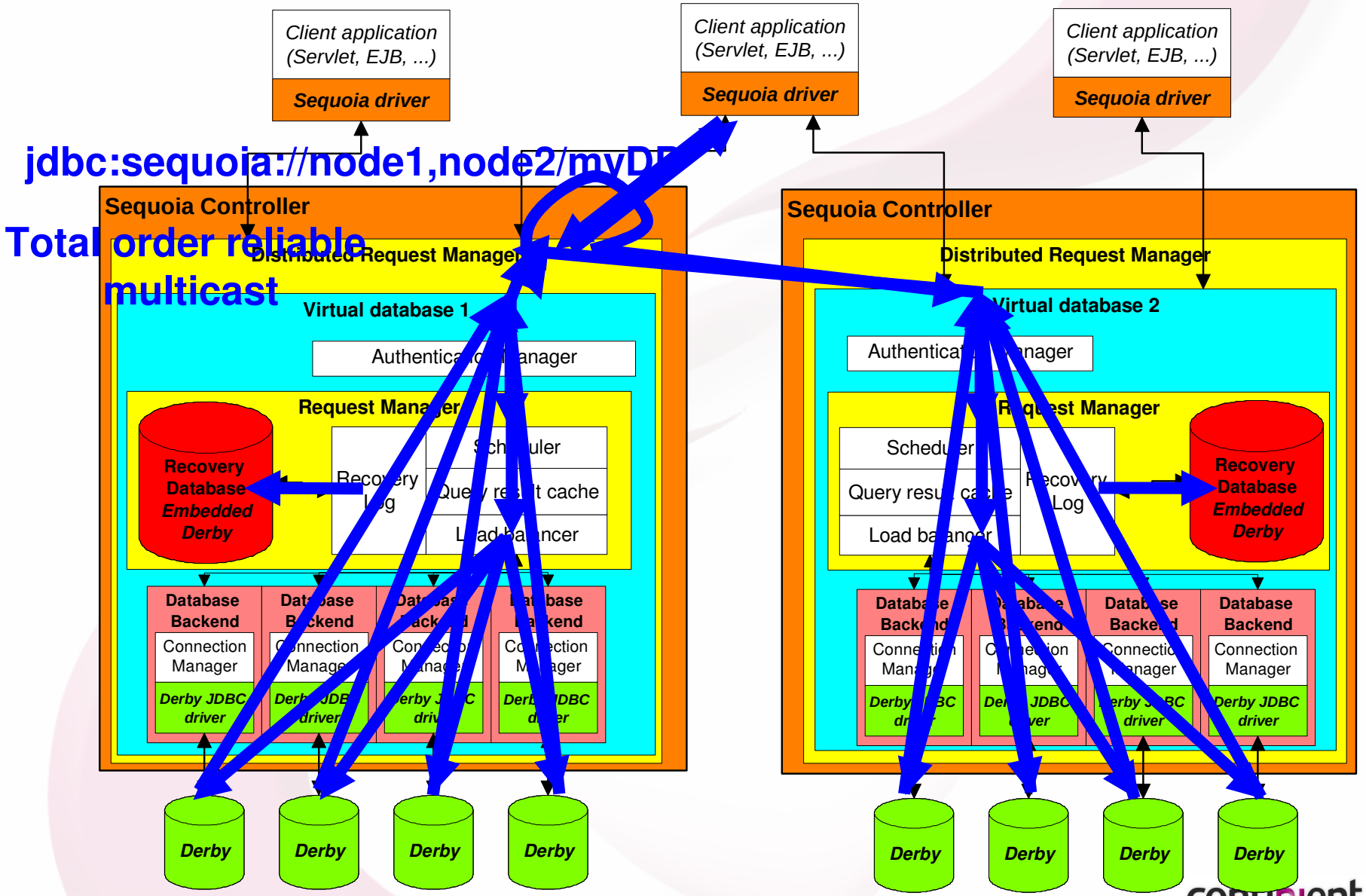
RR, WRR, LPRF, ...

get connection from pool

update cache (if available)

exec

Sequoia write request



Alternative replication algorithms

- **GORDA API**
  - European consortium defining API for pluggable replication algorithms
- **Sequoia 3.0 GORDA compliant prototype for PostgreSQL**
  - Uses triggers to compute write-sets
  - Certifies transaction at commit time
  - Propagate write-sets to other nodes
- **Tashkent/Tashkent+**
  - Research prototype developed at EPFL
  - Uses workload information for improved load balancing
- **More information**
  - http://sequoia.continuent.org
  - http://gorda.di.uminho.pt/

PostgreSQL specific issues

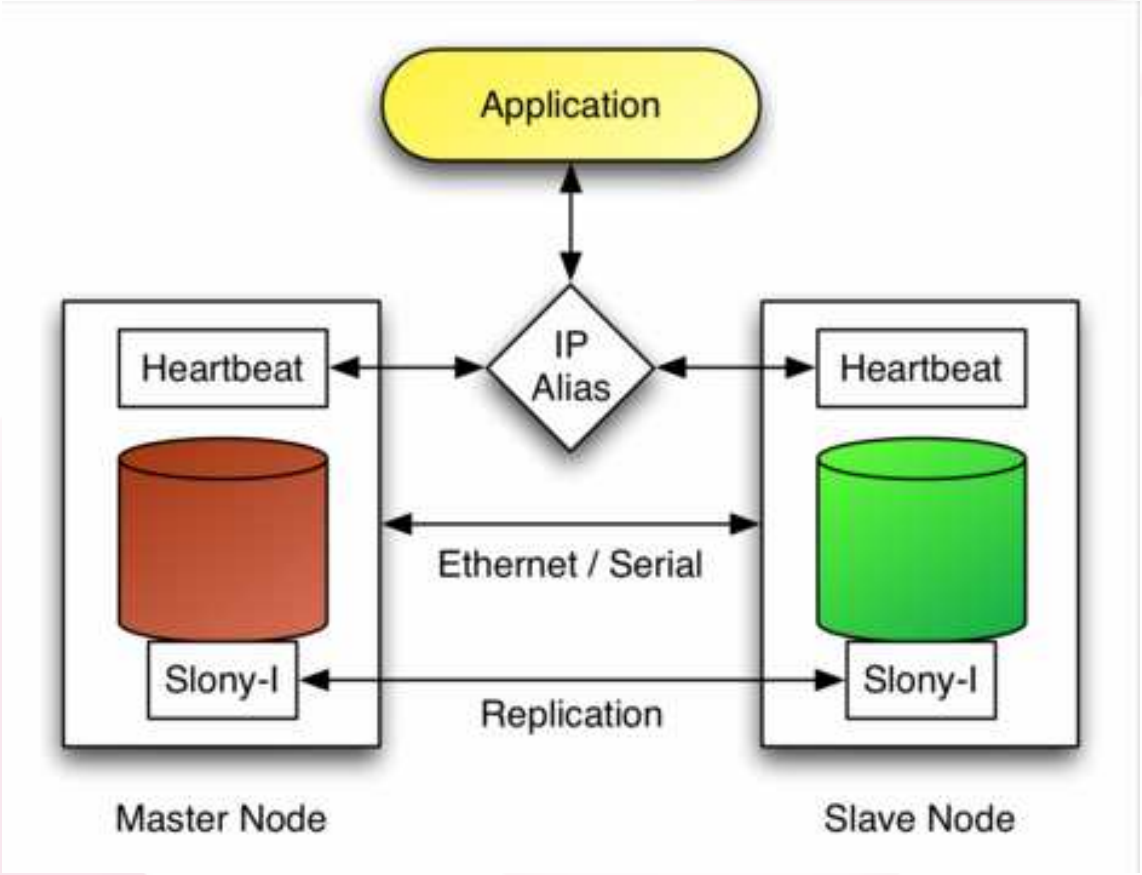
- **Indeterminist queries**
  - Macros in queries (now(), current_timestamp, rand(), …)
  - Stored procedures, triggers, …
  - SELECT … LIMIT can create non-deterministic results in UPDATE statements if the SELECT does not have an ORDER BY with a unique index:
    UPDATE FOO SET KEYVALUE='x' WHERE ID IN
    (SELECT ID FROM FOO WHERE KEYVALUE IS NULL LIMIT 10)
- **Sequences**
  - setval() and nextval() are not rollback
  - nextval() can also be called within SELECT
- **Serial type**
- **Large objects and OIDs**
- **Schema changes**
- **User access control**
  - not stored in database (pg_hba.conf)
  - host-based control might be fooled by proxy
  - backup/restore with respect to user rights
- **VACUUM**

Outline

- Database replication strategies
- PostgreSQL replication solutions
- Building HA solutions
- Management issues in production

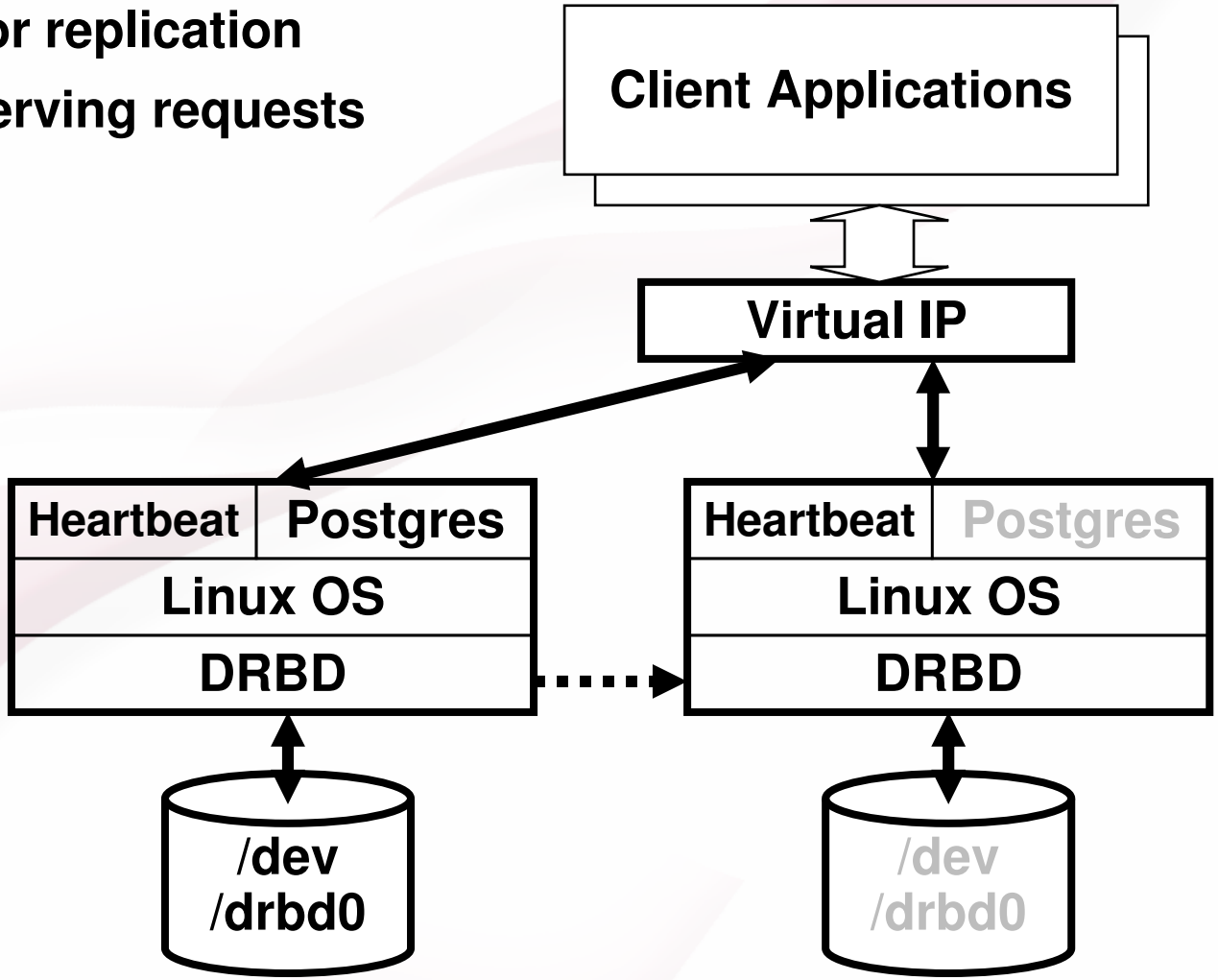
Simple hot-standby solution (1/3)

- Virtual IP address + Heartbeat for failover
- Slony-I for replication



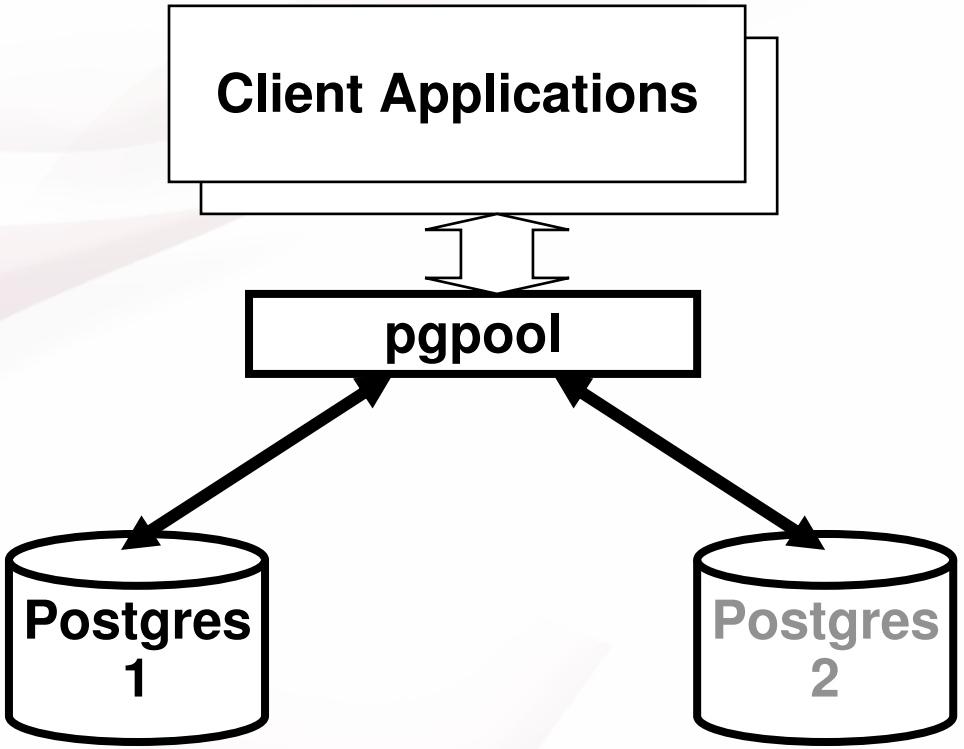
Simple hot-standby solution (2/3)

- Virtual IP address + Heartbeat for failover
- Linux DRDB for replication
- Only 1 node serving requests



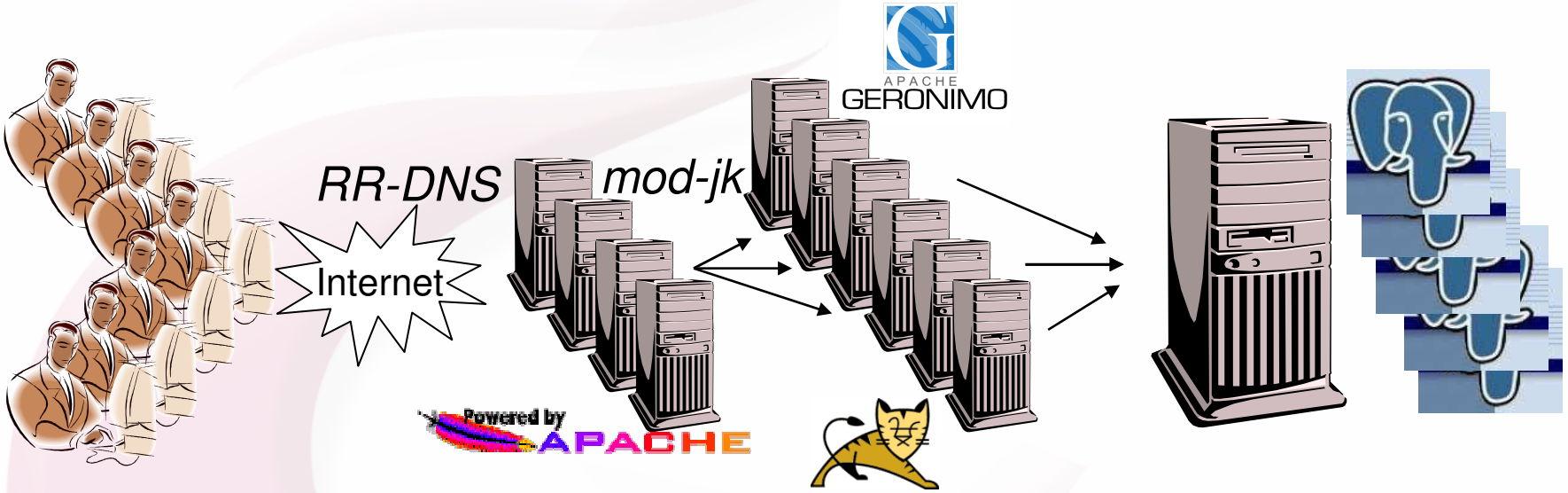
Simple hot-standby solution (3/3)

- **pgpool for failover**
- **proxy might become bottleneck**
  - requires 3 sockets per client connection
  - increased latency
- **Only 1 node serving requests**



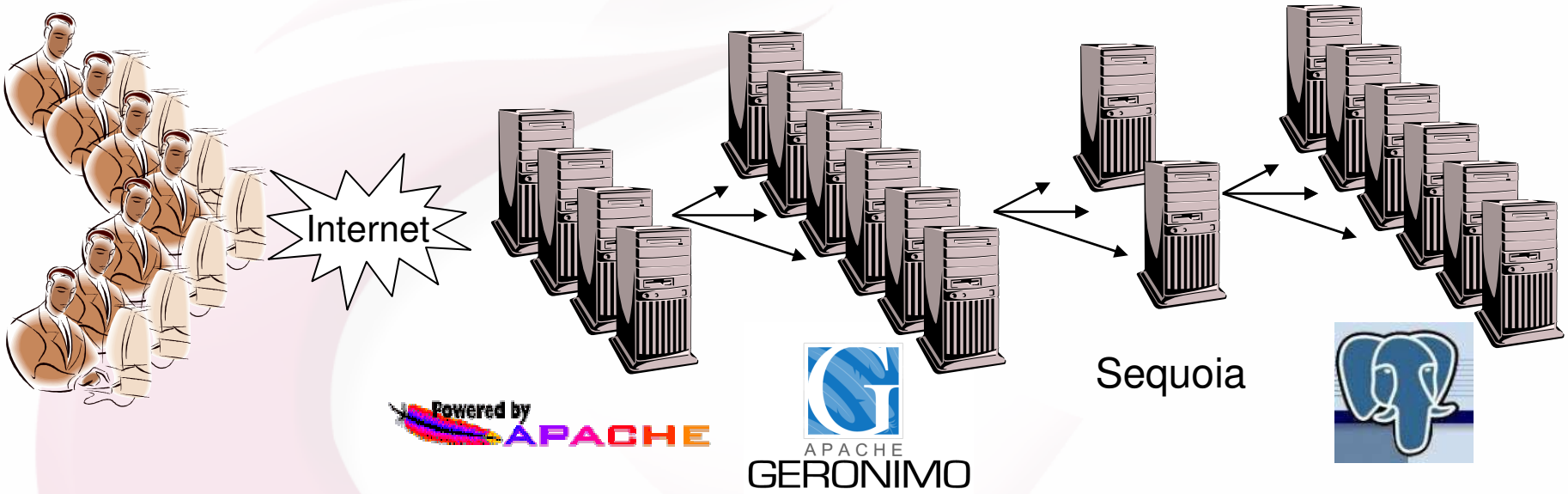
Highly available web site

- **Apache clustering**
  - L4 switch, RR-DNS, One-IP techniques, LVS, Linux-HA, …
- **Web tier clustering**
  - mod_jk (T4), mod_proxy/mod_rewrite (T5), session replication
- **PostgreSQL multi-master clustering solution**

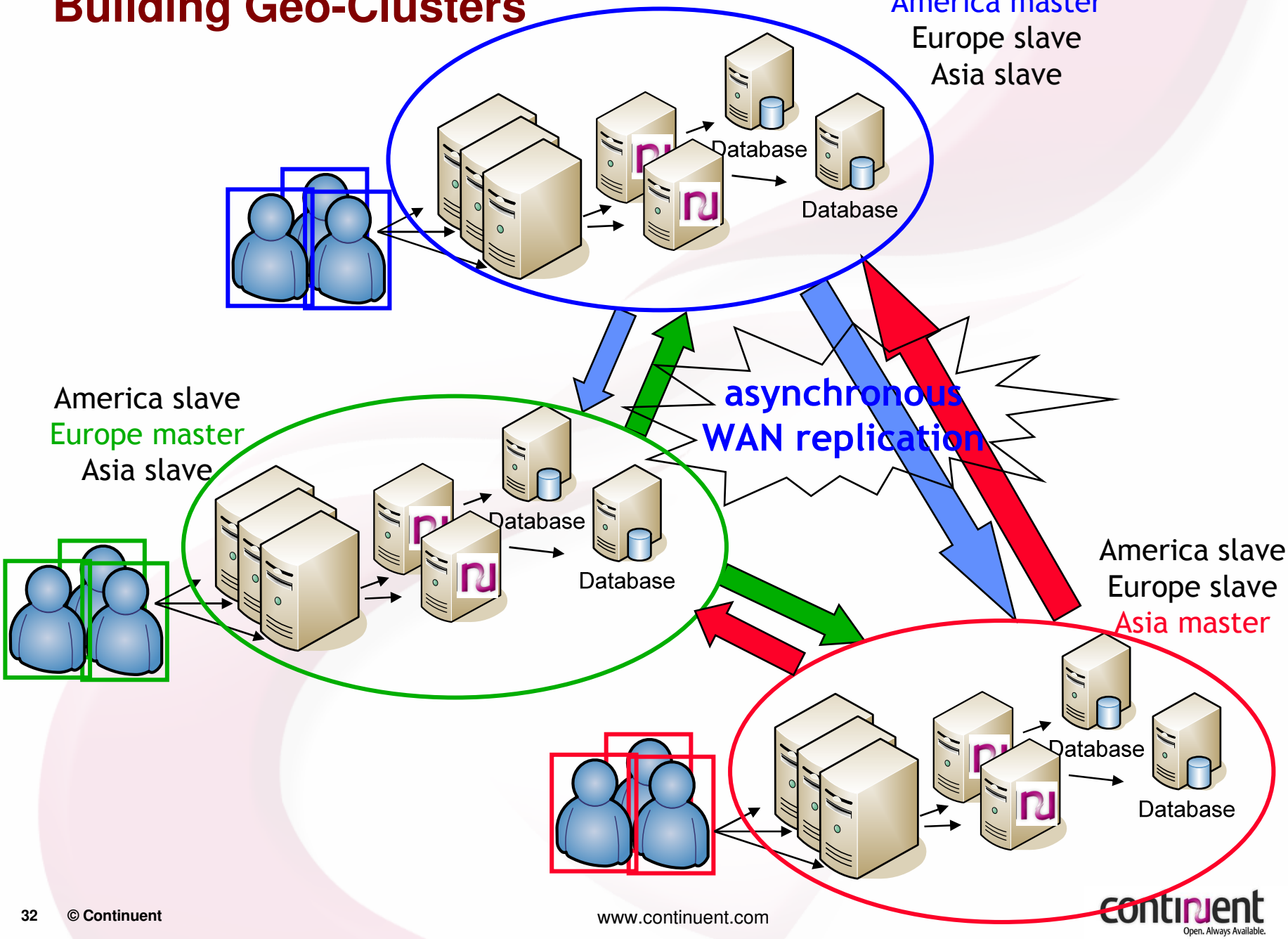


Highly available web applications

- Consider MTBF (Mean time between failure) of every hardware and software component
- Take MTTR (Mean Time To Repair) into account to prevent long outages
- Tune accordingly to prevent trashing

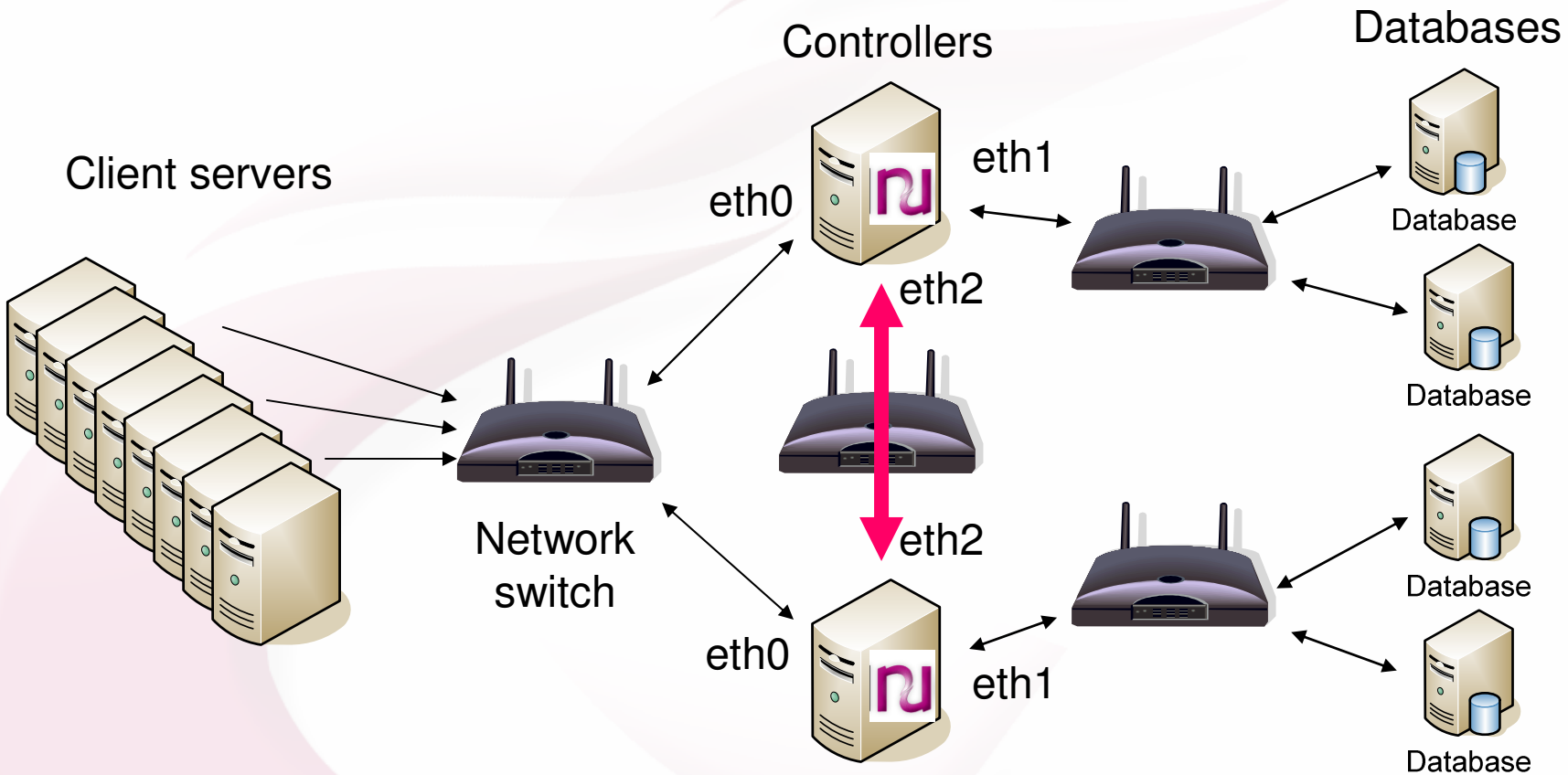


Building Geo-Clusters



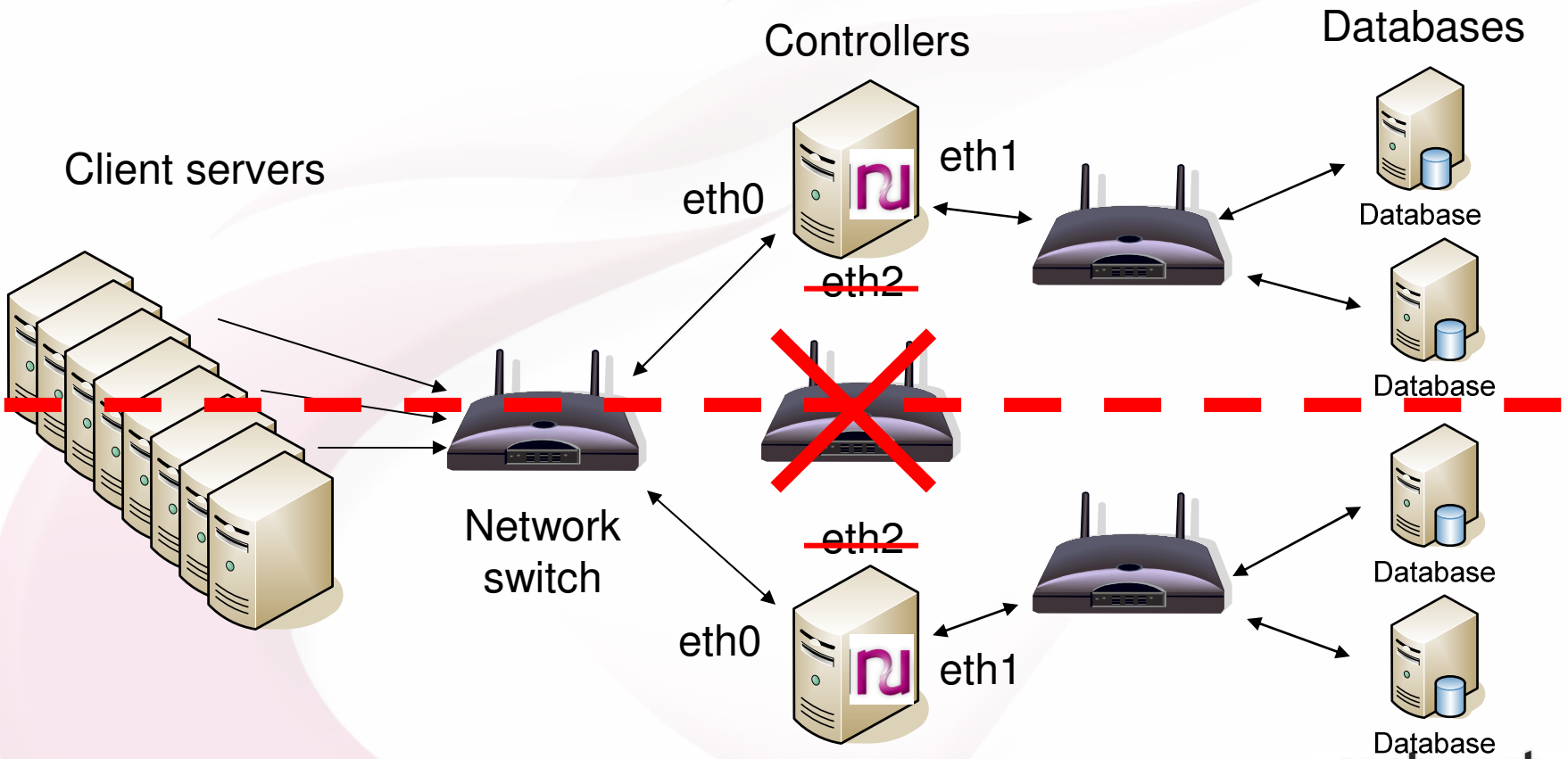
Split brain problem (1/2)

- **This is what you should NOT do:**
  - At least 2 network adapters in controller
  - Use a dedicated network for controller communication



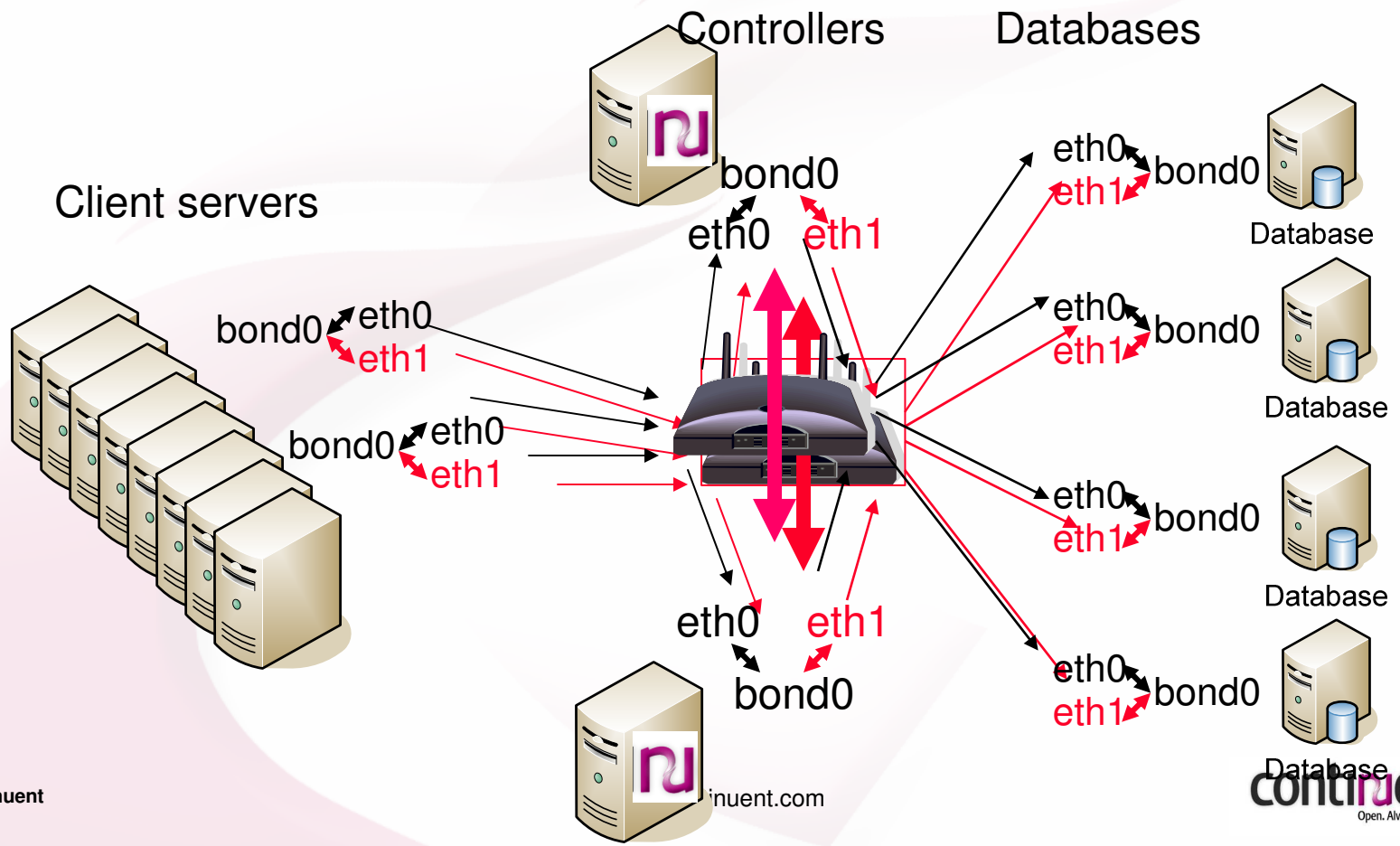
Split brain problem (2/2)

- When controllers lose connectivity clients may update inconsistently each half of the cluster
- No way to detect this scenario (each half thinks that the other half has simply failed)



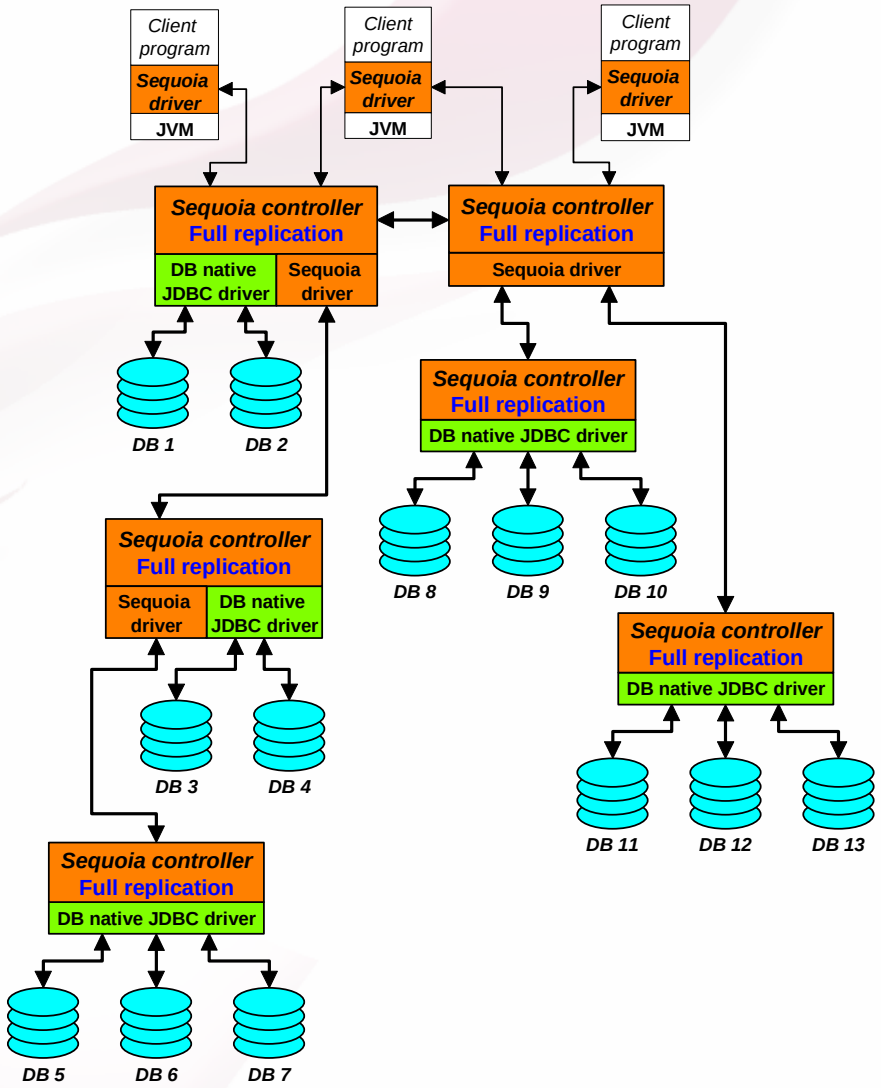
Avoiding network failure and split-brain

- **Collocate all network traffic using Linux Bonding**
- **Replicate all network components (mirror the network configuration)**
- **Various configuration options available for bonding (active-backup or trunking)**



Synchronous GeoClusters

- Multi-master replication requires group communication optimized for WAN environments
- Split-brain issues will happen unless expensive reliable dedicated links are used
- Reconciliation procedures are application dependent



Outline

- Database replication strategies
- PostgreSQL replication solutions
- Building HA solutions
- Management issues in production

Managing a cluster in production

- **Diagnosing reliably cluster status**
- **Getting proper notifications/alarms when something goes wrong**
  - Standard email or SNMP traps
  - Logging is key for diagnostic
- **Minimizing downtime**
  - Migrating from single database to cluster
  - Expanding cluster
  - Staging environment is key to test
- **Planned maintenance operations**
  - Vacuum
  - Backup
  - Software maintenance (DB, replication software, …)
  - Node maintenance (reboot, power cycle, …)
  - Site maintenance (in GeoCluster case)

Dealing with failures

- **Sotfware vs Hardware failures**
  - client application, database, replication software, OS, VM, …
  - power outage, node, disk, network, Byzantine failure, …
  - Admission control to prevent trashing
- **Detecting failures require proper timeout settings**
- **Automated failover procedures**
  - client and cluster reconfiguration
  - dealing with multiple simultaneous failures
  - coordination required between different tiers or admin scripts
- **Automatic database resynchronization / node repair**
- **Operator errors**
  - automation to prevent manual intervention
  - always keep backups and try procedures on staging environment first
- **Disaster recovery**
  - minimize data loss but preserve consistency
  - provisioning and planning are key
- **Split brain or GeoCluster failover**
  - requires organization wide coordination
  - manual diagnostic/reconfiguration often required

Summary

- Different replication strategies for different needs
- Performance ≠ Scalability
- Manageability becomes THE major issue in production

Links

- *pgpool*: http://pgpool.projects.postgresql.org/
- *PGcluster*: http://pgcluster.projects.postgresql.org/
- *Slony*: http://slony.info/
- *Sequoia*: http://sequoia.continuent.org
- *GORDA*: http://gorda.di.uminho.pt/

- Slides: http://sequoia.continuent.org/Resources

continuent
Open. Always Available.

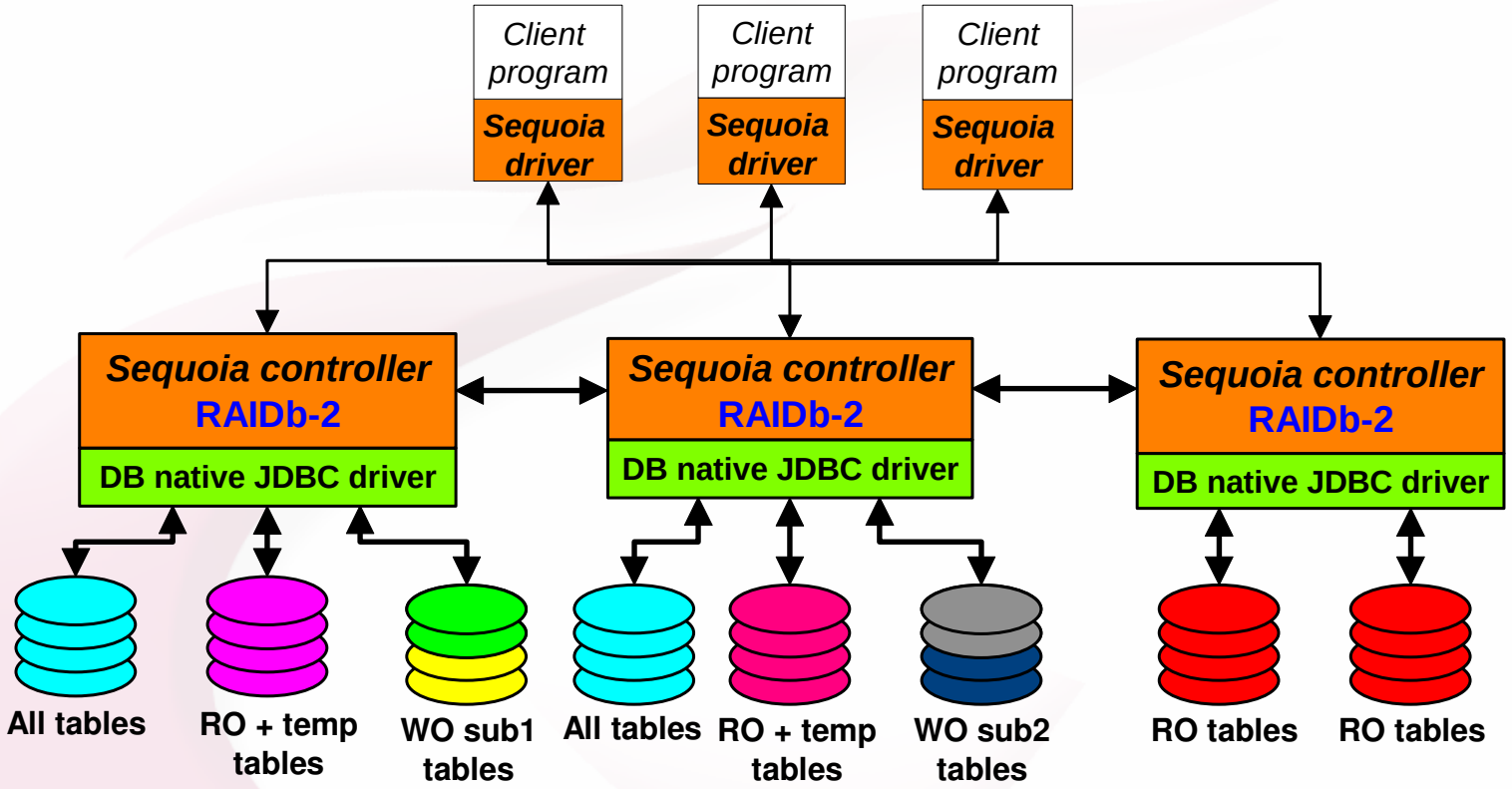
http://www.continuent.org

Bonus slides

continuent

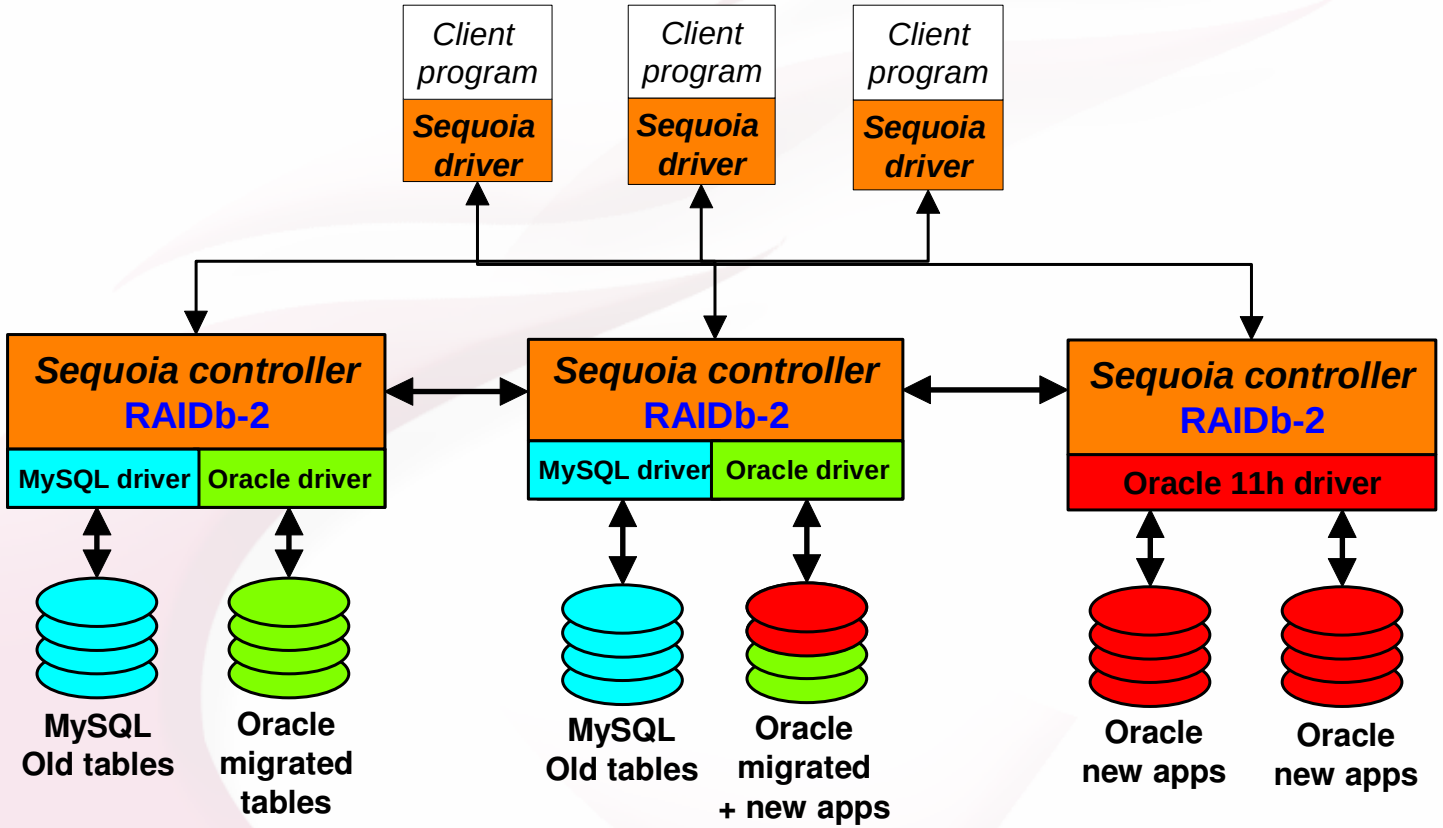
RAIDb-2 for scalability

- limit replication of heavily written tables to subset of nodes
- dynamic replication of temp tables
- reduces disk space requirements



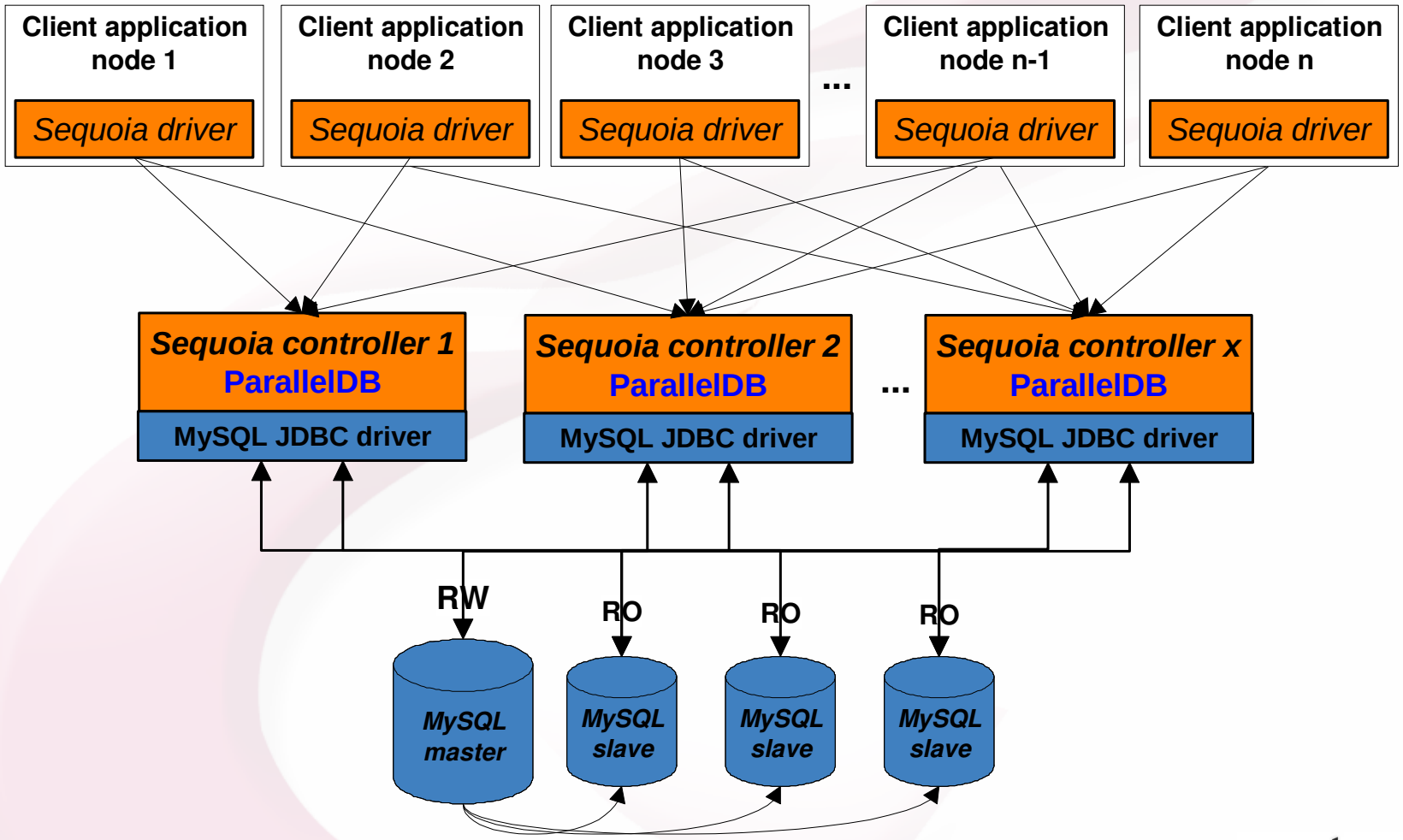
RAIDb-2 for heterogeneous clustering

- Migrating from MySQL to Oracle
- Migrating from Oracle x to Oracle x+1



Server farms with master/slave db replication

- No need for group communication between controller
- Admin. operations broadcast to all controllers



Composing Sequoia controllers

- Sequoia controller viewed as single database by client (app. or other Sequoia controller)
- No technical limit on composition deepness
- Backends/controller cannot be shared by multiple controllers
- Can be expanded dynamically