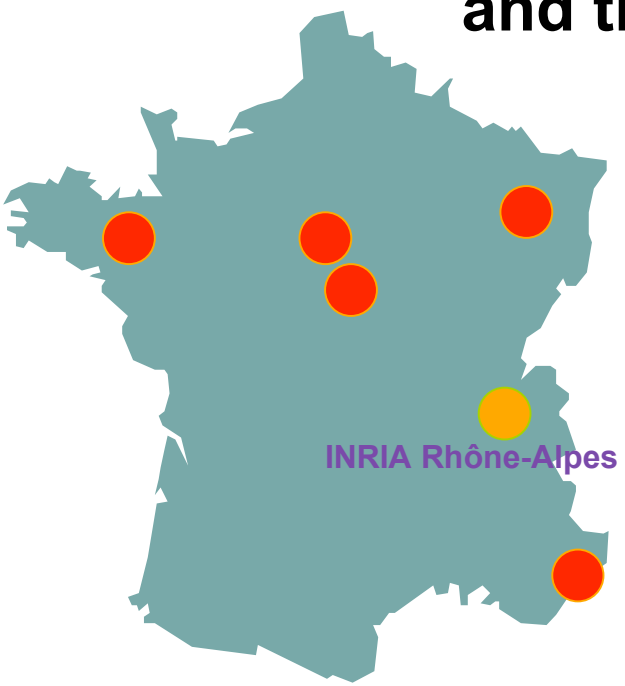
An open source middleware to build Redundant Array of Inexpensive Databases

Emmanuel Cecchet

INRIA key figures

A public scientific and technological research institute in computer science and control under the dual authority of the Ministry of Research and the Ministry of Industry



6 Research Units

A scientific force of 3,000

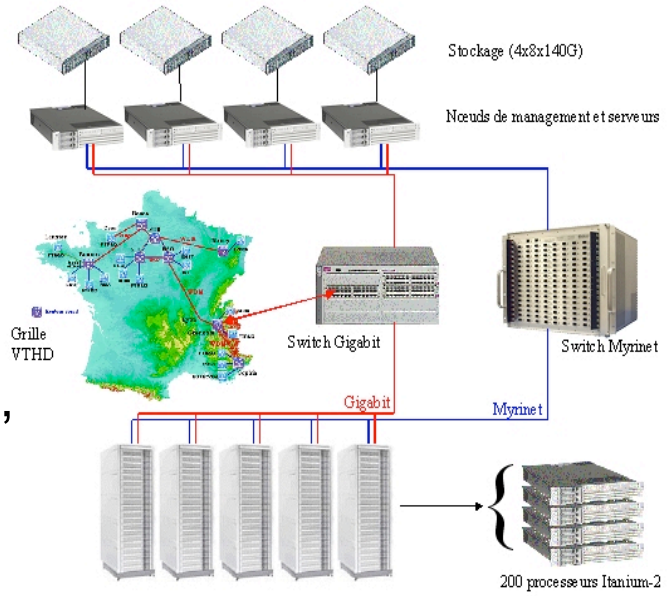
900 permanent staff
- 400 researchers
- 500 engineers, technical and administrative

450 researchers from other organizations
700 Ph.D students
200 external collaborators
750 trainees, post-doctoral students, visiting researchers from abroad (universities or industry)

Budget: 120 M€ (tax not incl.)

iCluster 2

- Itanium-2 processors
- 104 nodes (Dual 64 bits 900 MHz processors, 3 GB memory, 72 GB local disk) connected through a Myrinet network
- 208 processors, 312 GB memory, 7.5 TB disk
- Connected to the GRID network
- Linux OS (RedHat Advanced Server)
- First Linpack experiments at INRIA (Aug. 2003) have reached a 560 GFlop/s performance



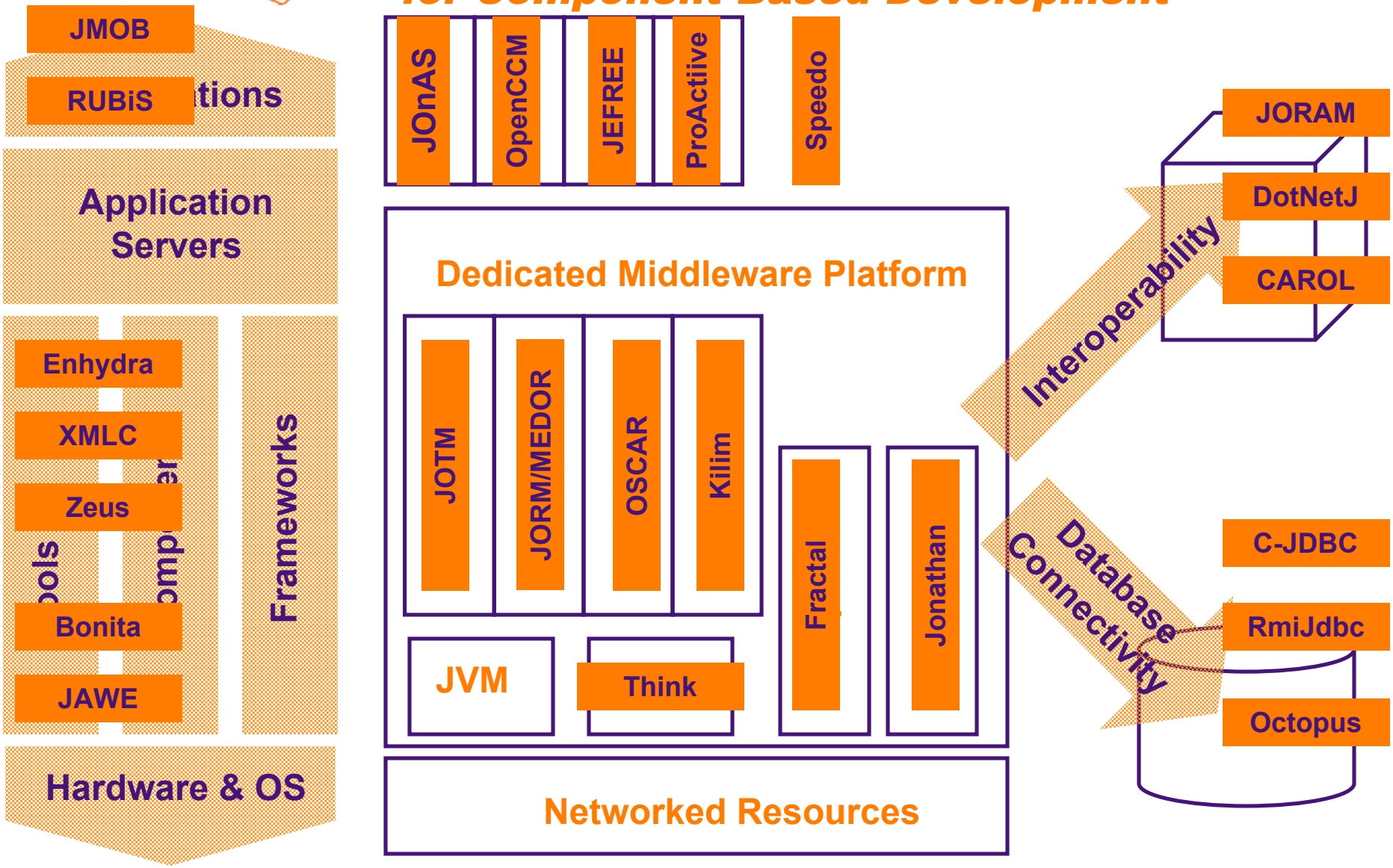
- Applications : Grid computing experiments, classical scientific computing, high performance Internet servers, …

ObjecWeb key figures

- **Open source middleware development**
- **Based on open standard**
  - J2EE, CORBA, OSGi…
- **International consortium**
- **Academic partners**
  - European universities and research centers
- **Industrial partners**
  - RedHat, Suse, …
  - France Telecom,
  - NEC, Bull, …

Common Software & Architecture

for Component Based Development



Outline

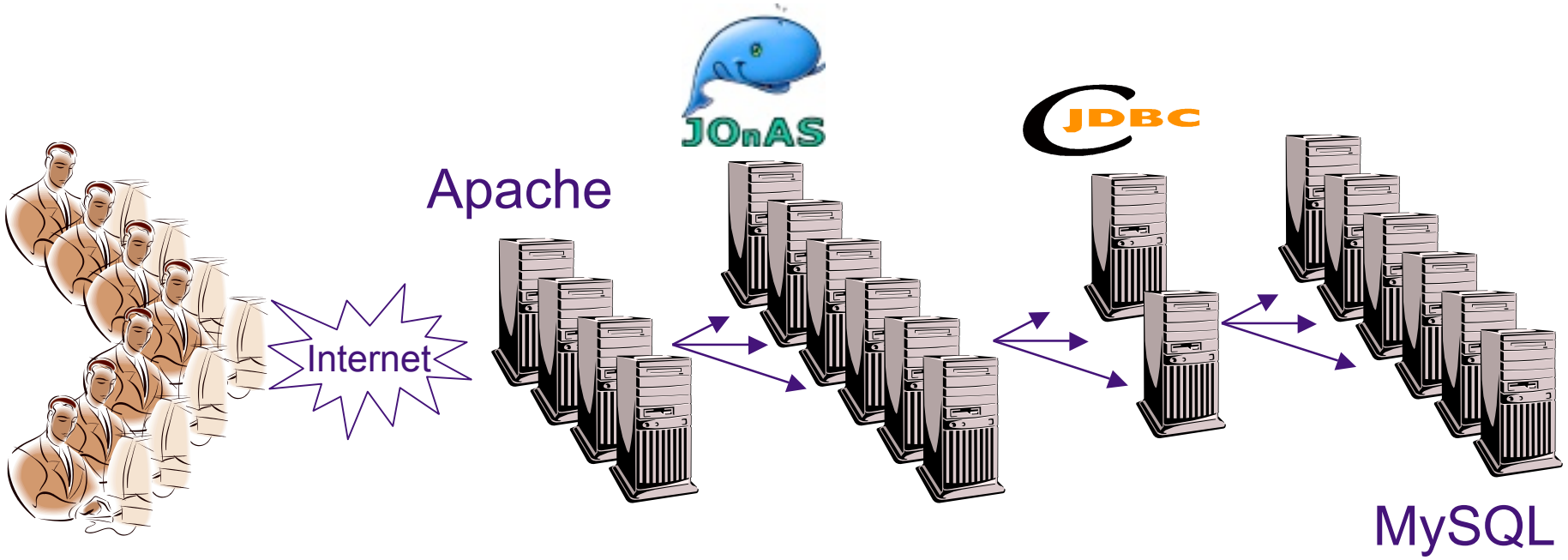
- Motivations
- RAIDb
- C-JDBC
- Performance
- Lessons learned
- Conclusion

Why did we design C-JDBC ?

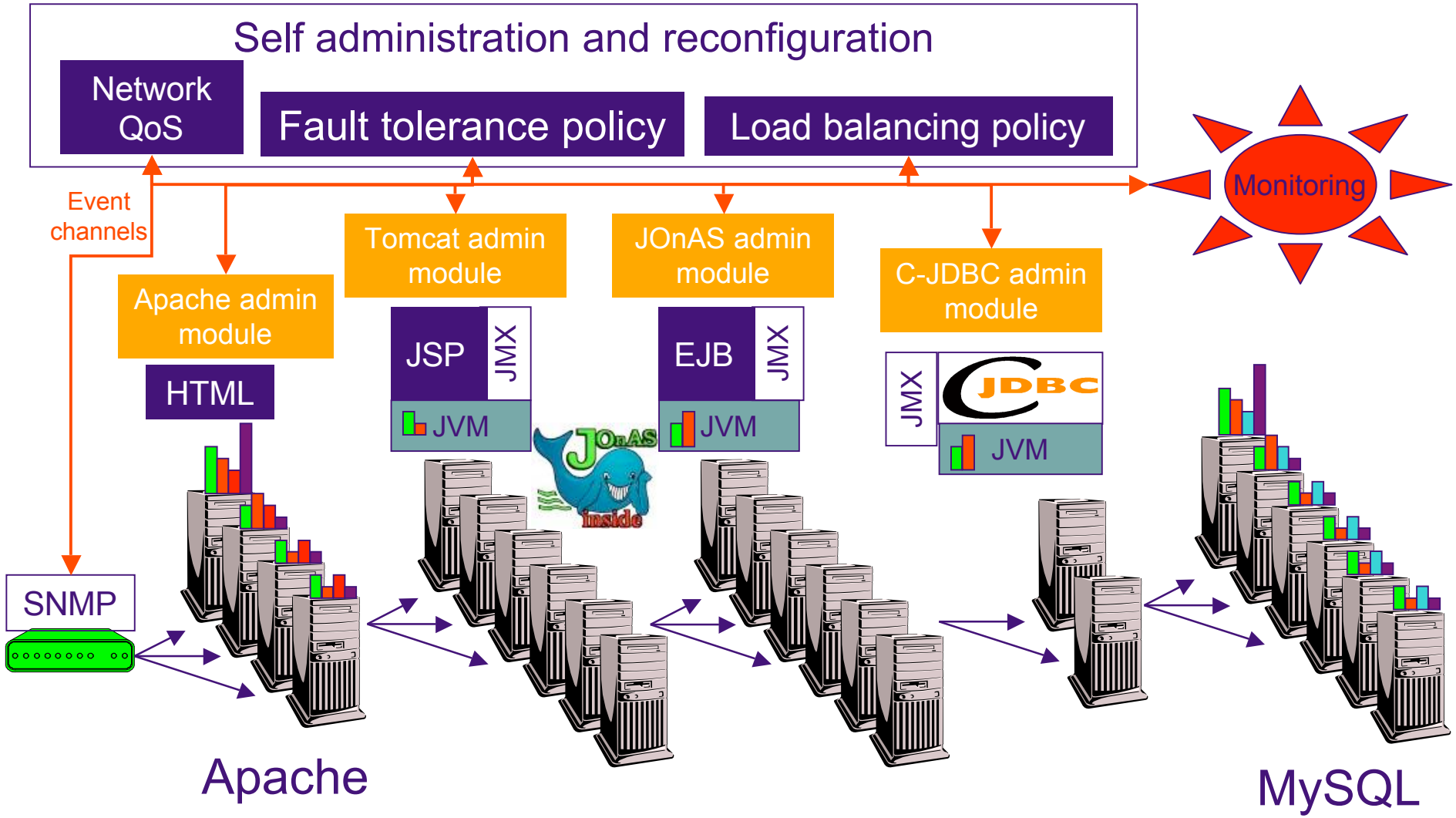
- **Scalability evaluation of J2EE servers**
  - performance bounded by database even with a single server
  - how to compare middleware performance ?
  - how to evaluate clustering features in J2EE servers ?
- **Solutions**
  - Large SMP machine: too expensive
  - Open source solution: do it yourself!
- **Features we wanted ordered by priority**
  - scalability
  - on commodity hardware
  - using open source databases
  - fault tolerance (high availability + failover)
  - without modifying the client application

How do we want to use C-JDBC ?

- ***From*** **small dynamic content web sites using a centralized open source database**
- ***To*** **an end-to-end open source solution for large scale J2EE clustered application servers**



Sardes project objectives: Autonomic J2EE clusters



Outline

- Motivations
- RAIDb
- C-JDBC
- Performance
- Lessons learned
- Conclusion

RAIDb - Definition

- Redundant Array of Inexpensive Databases
- better performance and fault tolerance than a single database, at a low cost, by combining multiple database instances into an array of databases
- RAIDb levels offers various tradeoff of performance and fault tolerance

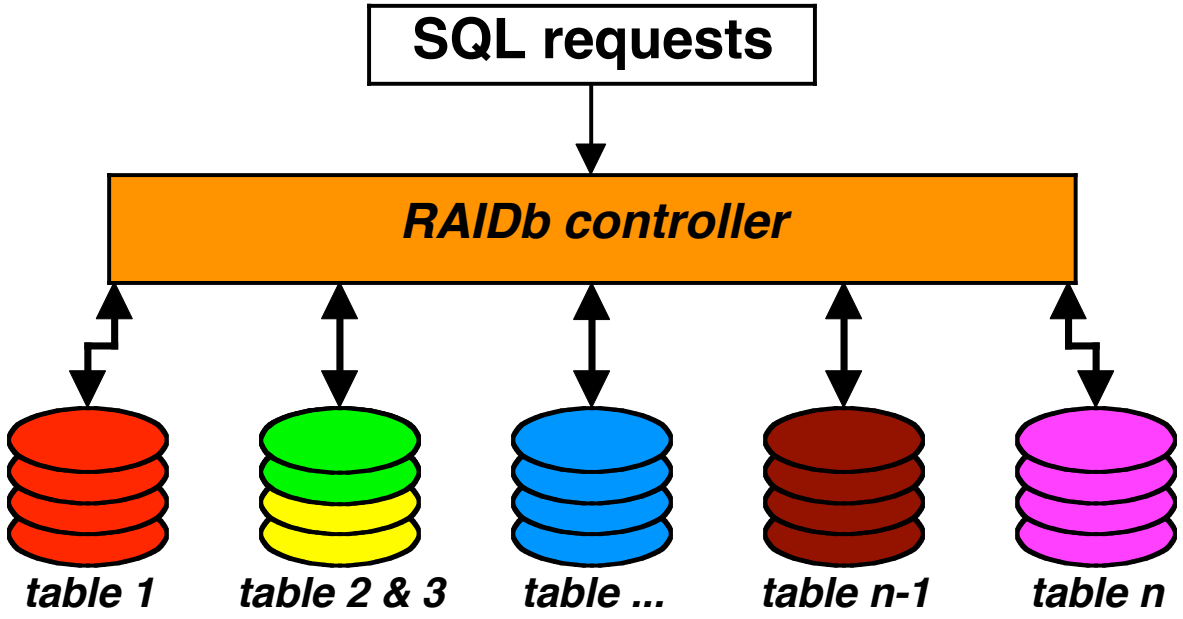
Key ideas

- **RAIDb controller**
  - gives the view of a single database to the client
  - balance the load on the database backends
- **RAIDb levels**
  - RAIDb-0: full partitioning
  - RAIDb-1: full mirroring
  - RAIDb-2: partial replication
  - composition possible

RAIDb levels

RAIDb-0

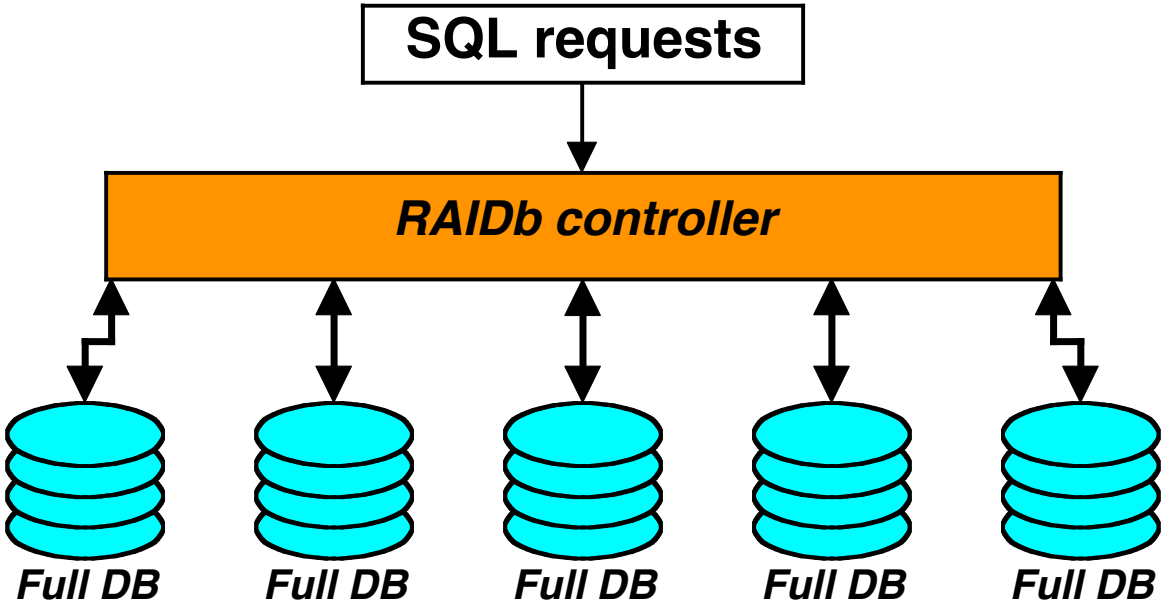
- partitioning
- no duplication and no fault tolerance
- at least 2 nodes



RAIDb levels

RAIDb-1

- mirroring
- performance bounded by write broadcast
- at least 2 nodes



RAIDb levels

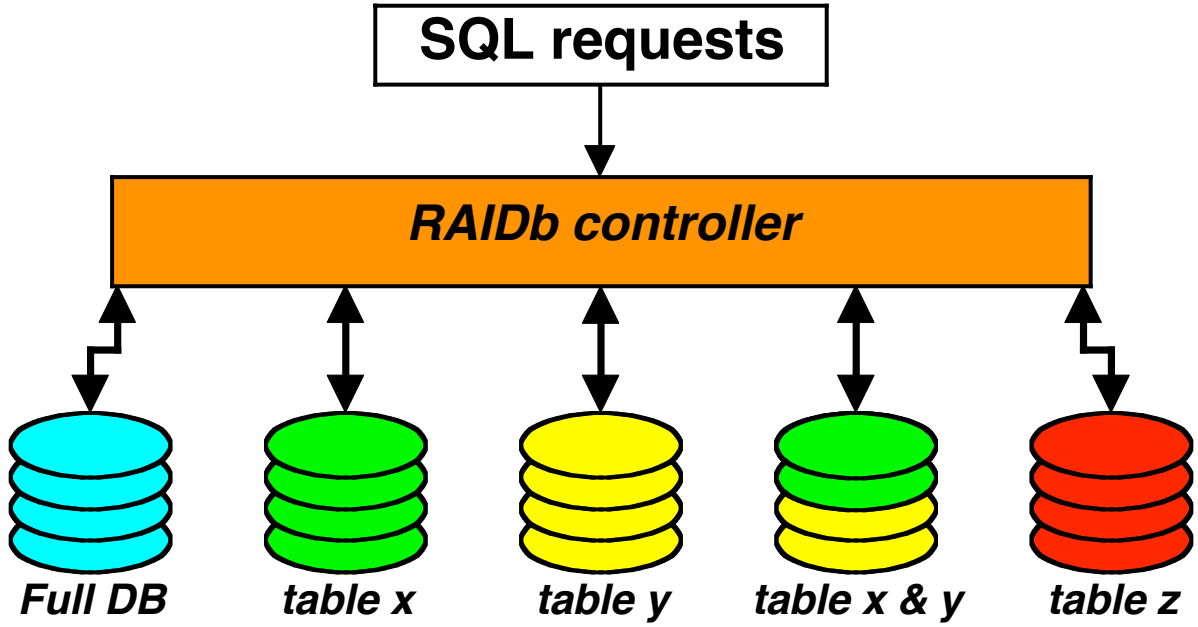
RAIDb-1ec

- mirroring + error checking
- error checking
  - read request sent to multiple databases
  - replies compared
  - result returned only if a quorum is reached
- at least 3 nodes

RAIDb levels

RAIDb-2

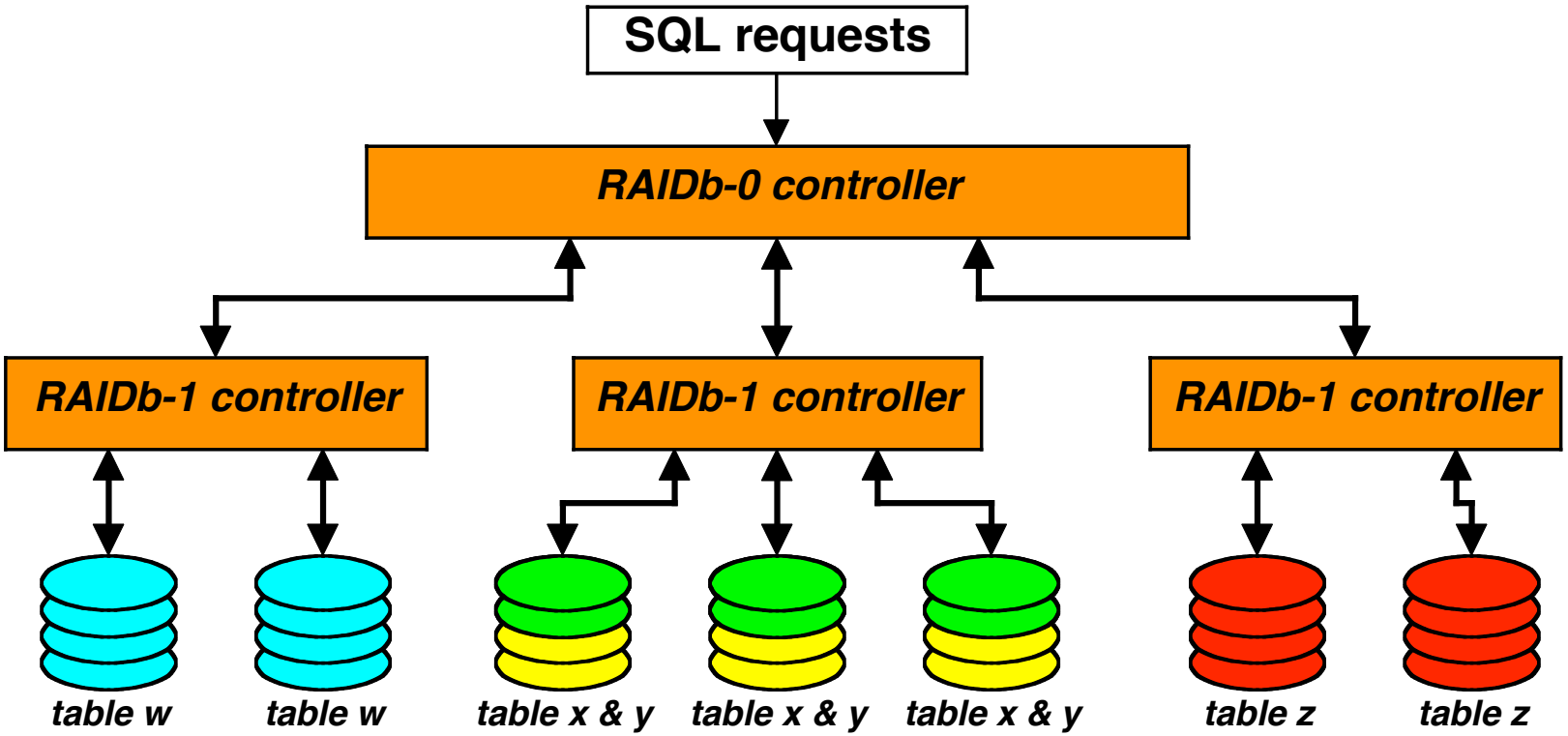
- partial replication
- at least 2 copies of each table
- at least 3 nodes



RAIDb levels composition

RAIDb-0-1

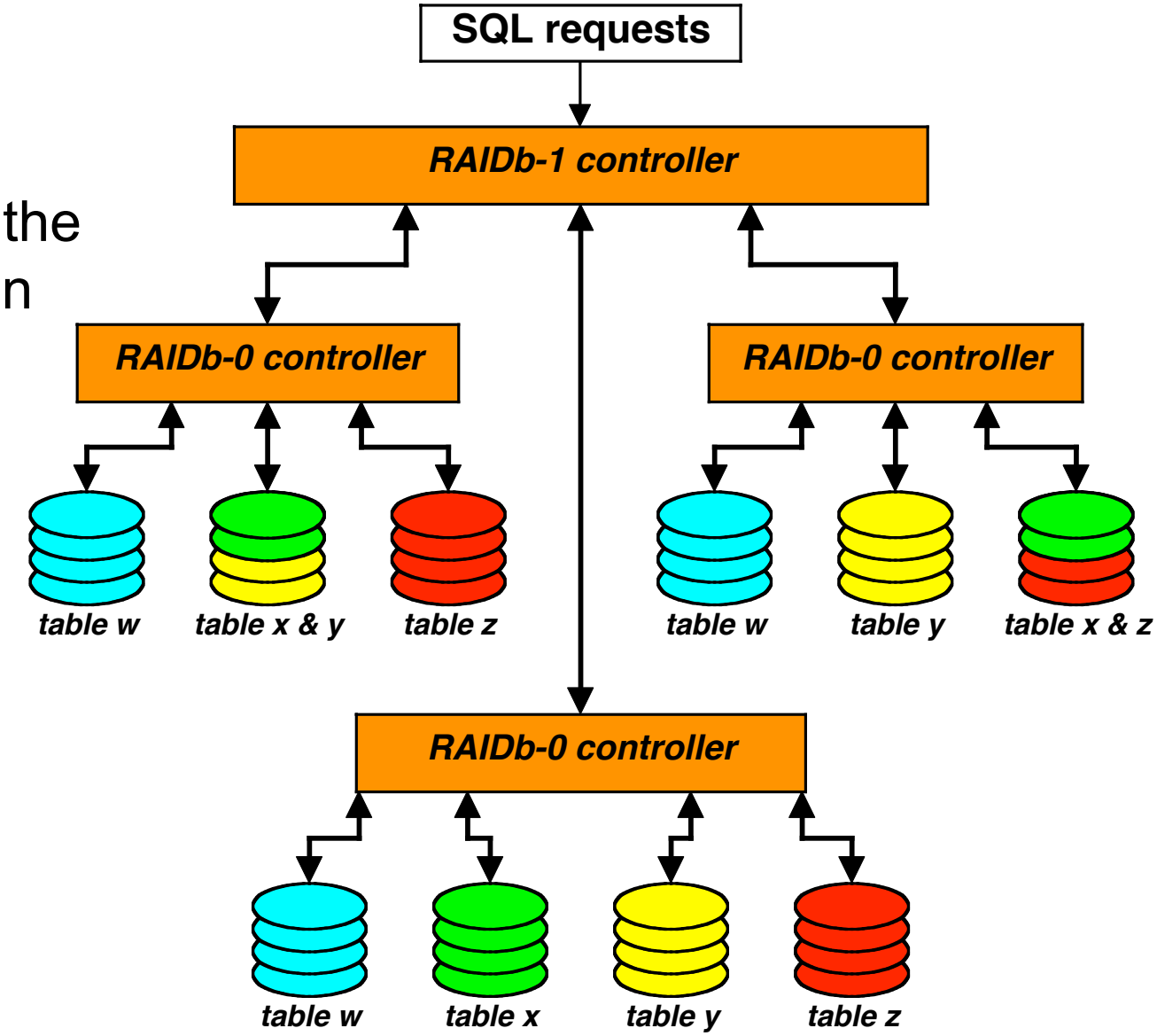
- RAIDb-0 at the top level
- RAIDb-1 underneath



RAIDb levels composition

RAIDb-1-0

- no limit to the composition deepness



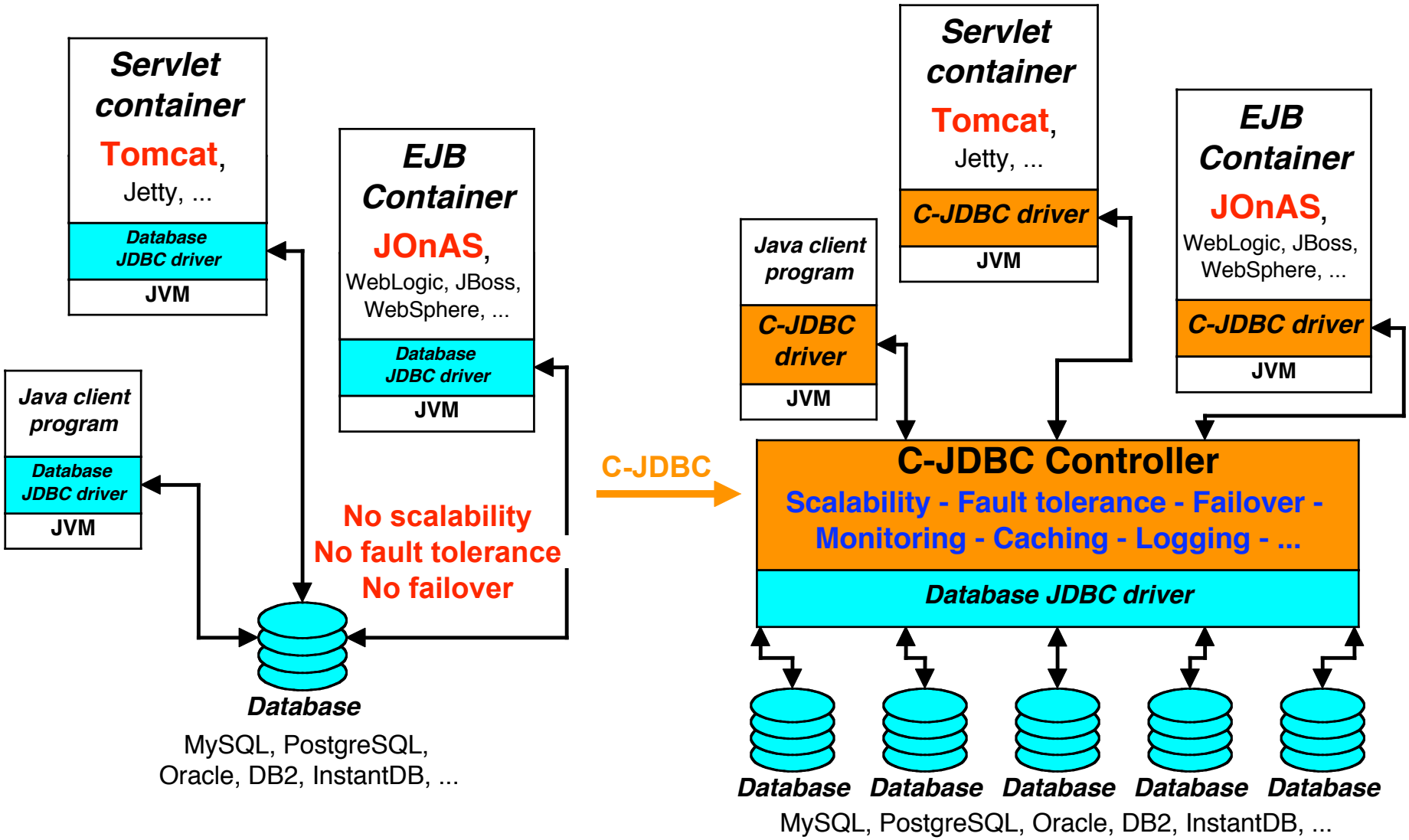
Outline

- **Motivations**
- **RAIDb**
- **C-JDBC**
  - Overview
  - Internals
  - Scalability
  - Checkpointing and Recovery
- **Performance**
- **Lessons learned**
- **Conclusion**

C-JDBC – Key ideas

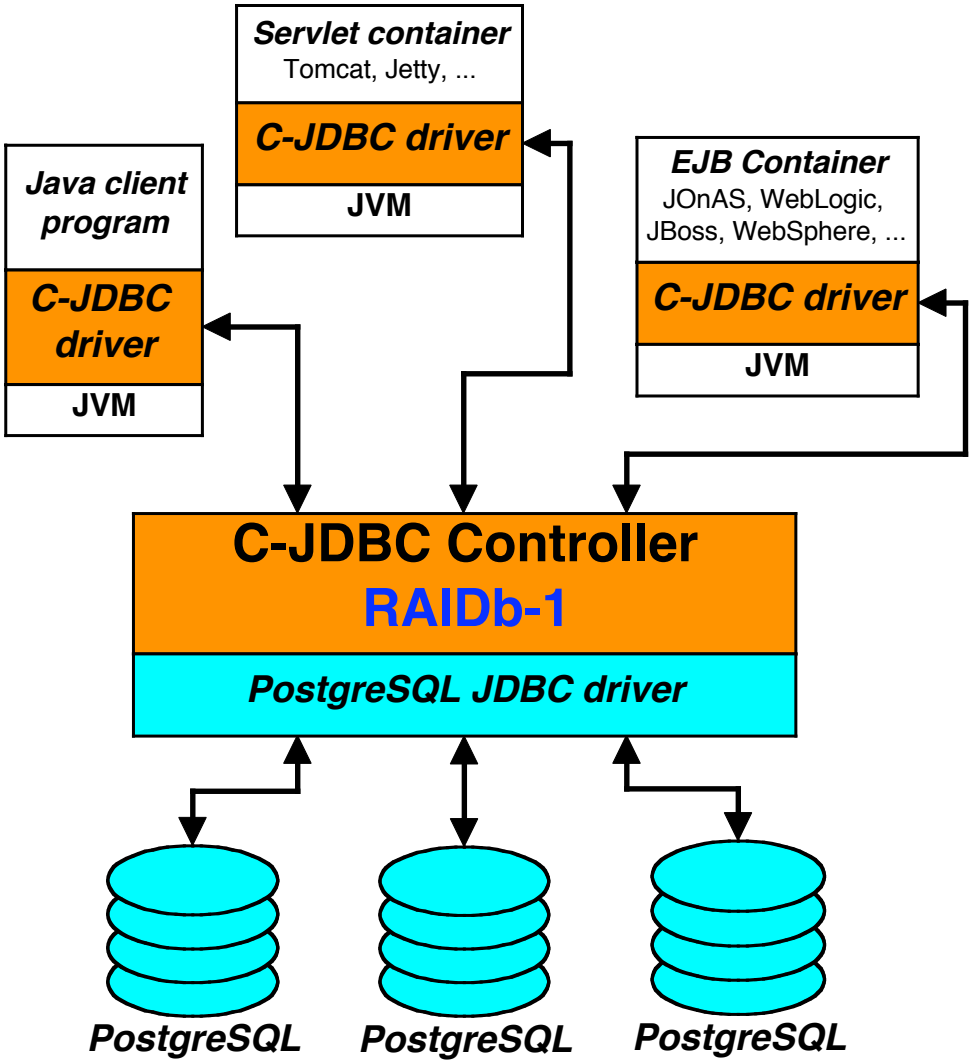
- **Middleware implementing RAIDb**
- **Two components**
  - generic JDBC 2.0 driver (C-JDBC driver)
  - C-JDBC Controller
- **C-JDBC Controller provides**
  - performance scalability
  - high availability
  - failover
  - caching, logging, monitoring, …
- **Supports heterogeneous databases**

C-JDBC – Overview



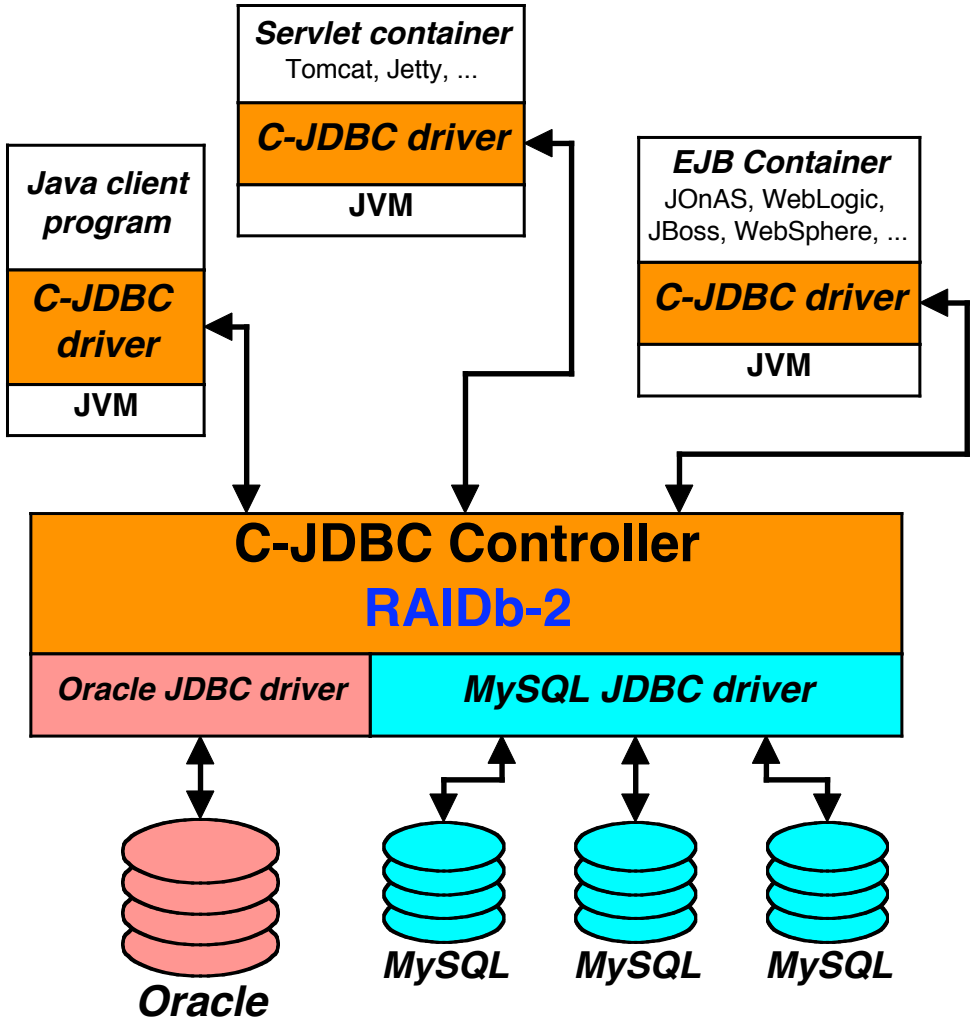
C-JDBC RAIDb-1 example

- no client code modification
- original PostgreSQL driver and RDBMS engine



C-JDBC RAIDb-2 example

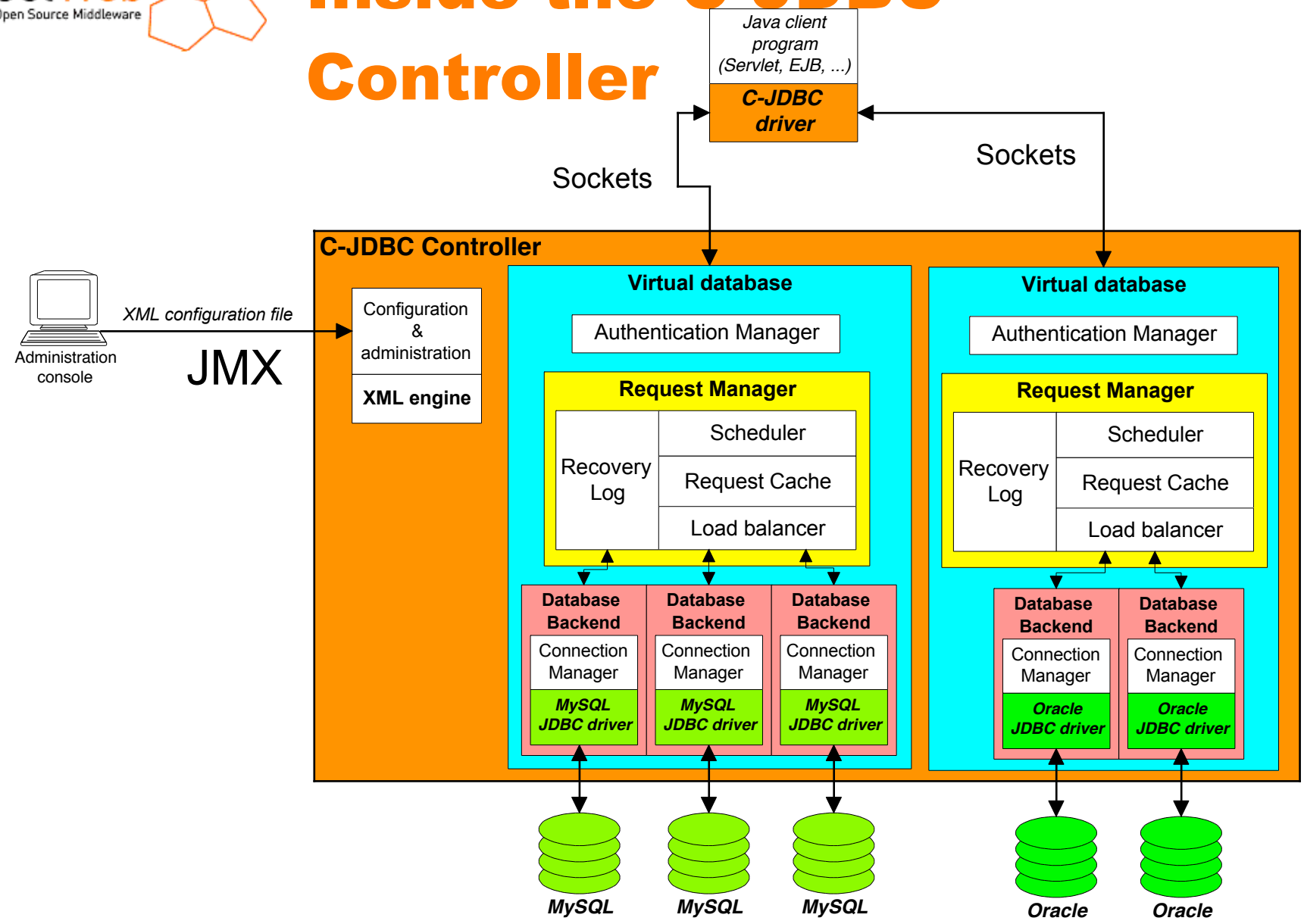
- supports cluster of heterogeneous RDBMS
- unload a single Oracle DB with several MySQL



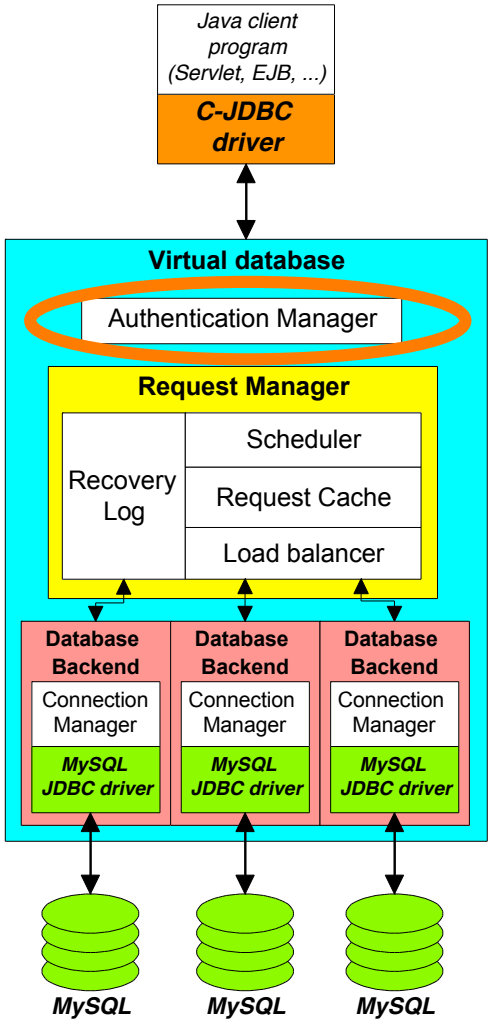
Outline

- **Motivations**
- **RAIDb**
- **C-JDBC**
  - Overview
  - Internals
  - Scalability
  - Checkpointing and Recovery
- **Performance**
- **Lessons learned**
- **Conclusion**

Inside the C-JDBC Controller

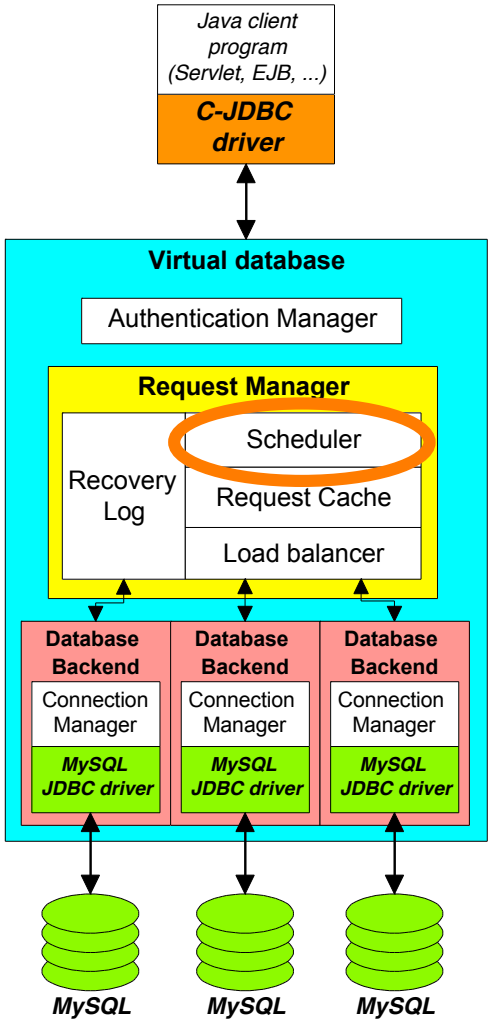


Authentication Manager



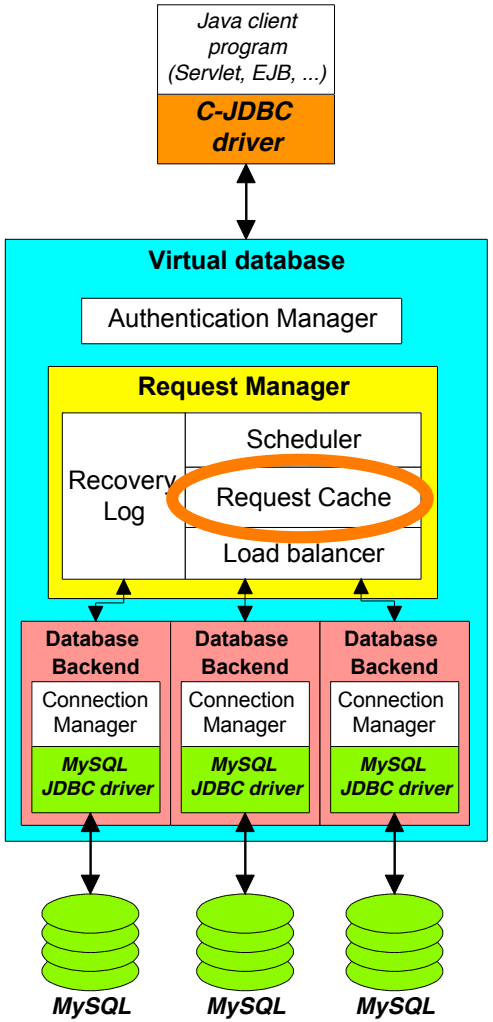
- Matches real login/password used by the application with backend specific login/ password
- Administrator login to manage the virtual database

Scheduler



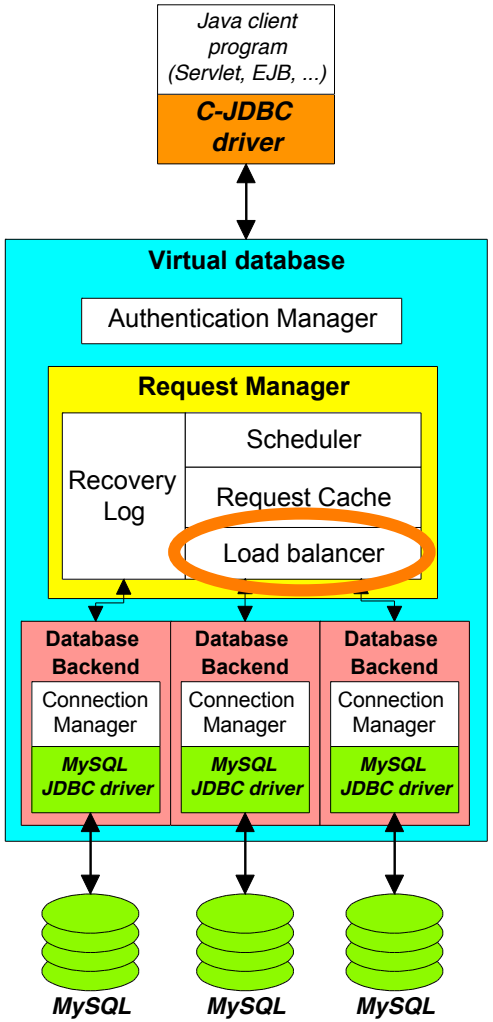
- **Manages concurrency control**
- **Specific implementations for Single DB, RAIDb 0, 1 and 2**
- **Query-level**
- **Optimistic and pessimistic transaction level**
  - uses the database schema that is automatically fetched from backends

Request cache



- **caches results from SQL requests**
- **improved SQL statement analysis to limit cache invalidations**
  - table based invalidations
  - column based invalidations
  - single-row SELECT optimization
- **request parsing possible in the C-JDBC driver**
  - offload the controller
  - parsing caching in the driver

Load balancer 1/2



RAIDb-0

- query directed to the backend having the needed tables

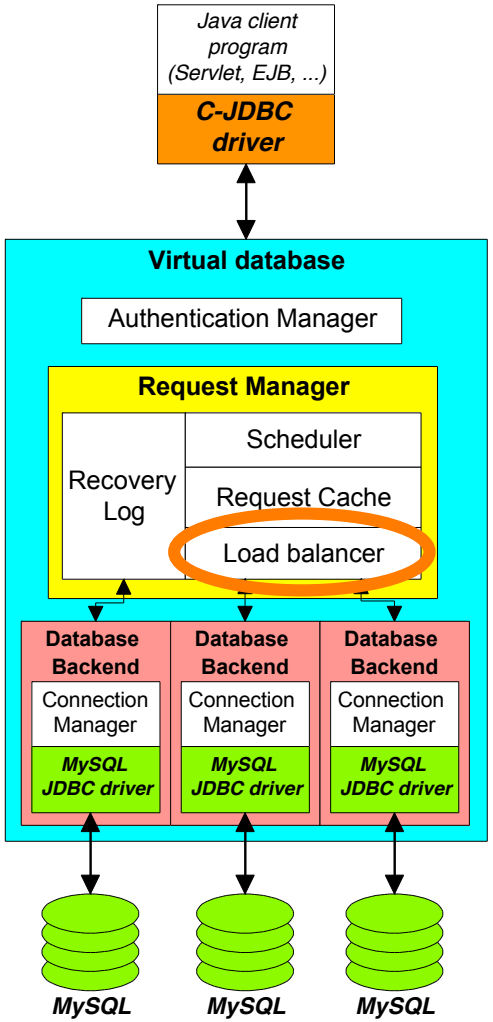
RAIDb-1

- read executed by current thread
- write executed in parallel by a dedicated thread per backend
- result returned if one, majority or all commit
- if one node fails but others succeed, failing node is disabled

RAIDb-2

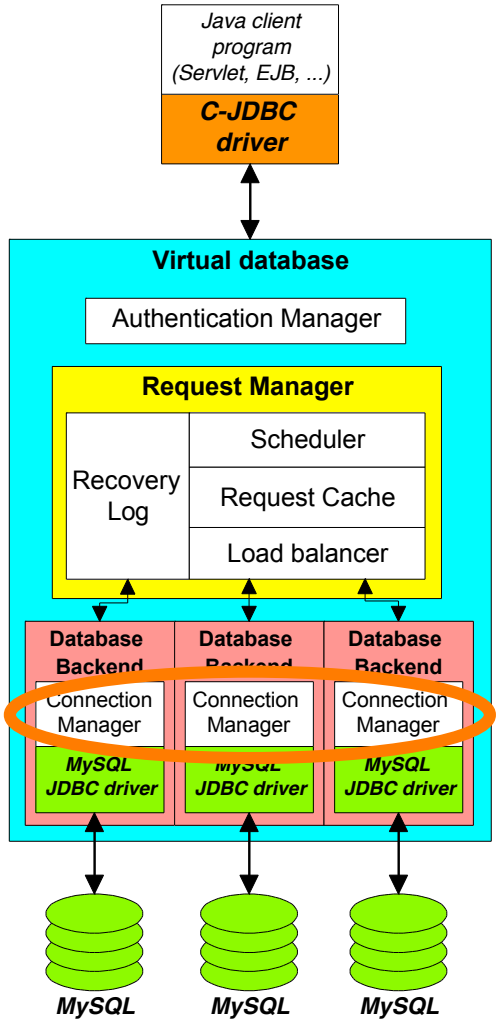
- same as RAIDb-1 except that writes are sent only to nodes owning the written table

Load balancer 2/2



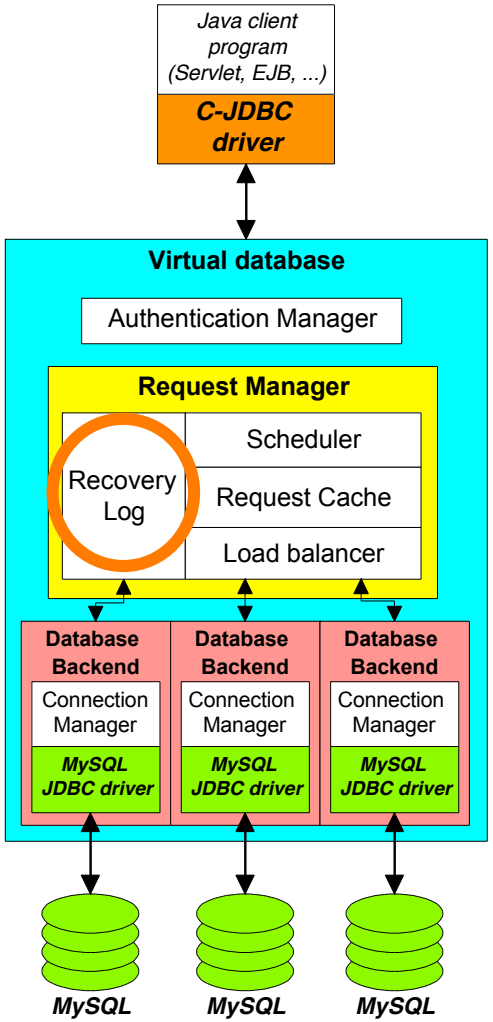
- **Static load balancing policies**
  - Round-Robin (RR)
  - Weighted Round-Robin (WRR)
- **Least Pending Requests First (LPRF)**
  - request sent to the node that has the shortest pending request queue
  - efficient if backends are homogeneous in terms of performance

Connection Manager



- **Connection pooling for a backend**
  - Simple : no pooling
  - RandomWait : blocking pool
  - FailFast : non-blocking pool
  - VariablePool : dynamic pool
- **Connection pools defined on a per login basis**
  - resource management per login
  - dedicated connections for admin

Recovery Log



- **Checkpoints are associated with database dumps**
- **Record all updates and transaction markers since a checkpoint**
- **Used to resynchronize a database from a checkpoint**
- **JDBCRecoveryLog**
  - store information in a database
  - can be re-injected in a C-JDBC cluster for fault tolerance

Outline

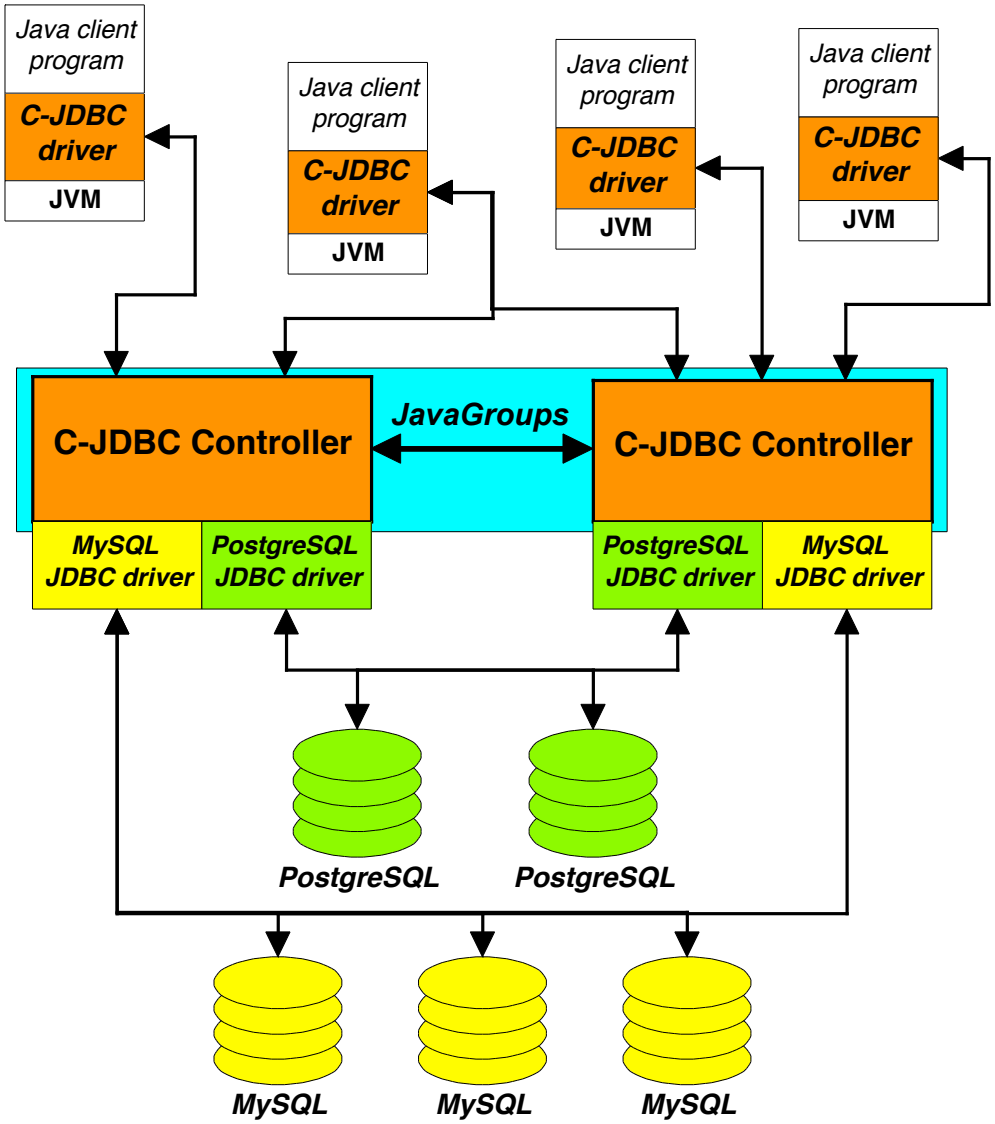
- **Motivations**
- **RAIDb**
- **C-JDBC**
  - Overview
  - Internals
  - Scalability
  - Checkpointing and Recovery
- **Performance**
- **Lessons learned**
- **Conclusion**

C-JDBC scalability

- **Horizontal scalability**
  - prevents the controller to be a Single Point Of Failure (SPOF)
  - distributes the load among several controllers
  - coherency ensured by group communications
- **C-JDBC Driver**
  - multiple controllers automatic failover
    - jdbc:c-jdbc://node1:25322,node2:12345/myDB
  - connection caching
  - URL parsing/controller lookup caching

C-JDBC – Horizontal scalability

- Writes broadcasted by JGroups
- Each backend is accessed in write by only one controller but possibly shared by all for reads
- Group commit only for write transactions

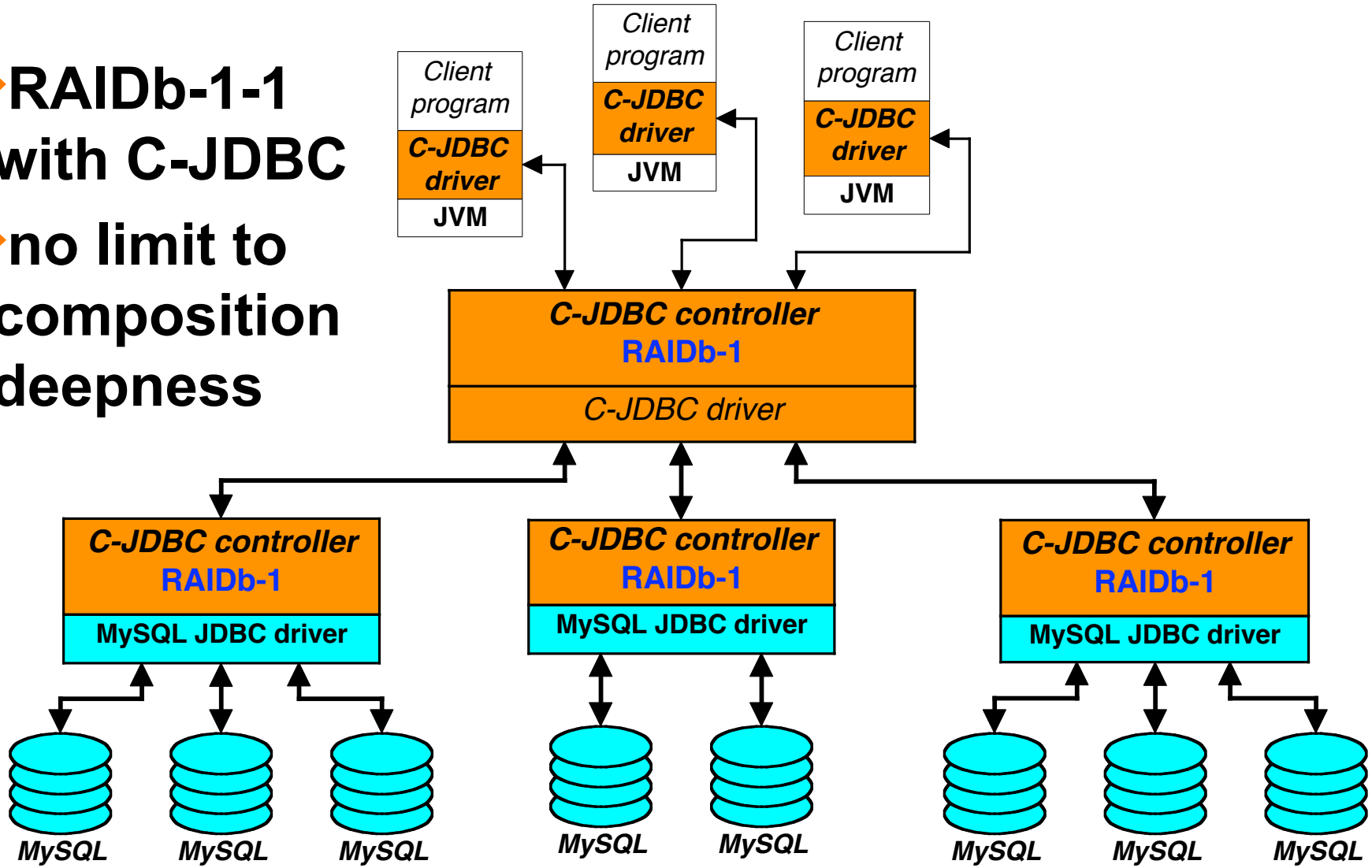


C-JDBC scalability

- **Vertical scalability**
  - allows nested RAIDb levels
  - allows tree architecture for scalable write broadcast
  - necessary with large number of backends
  - C-JDBC driver re-injected in C-JDBC controller

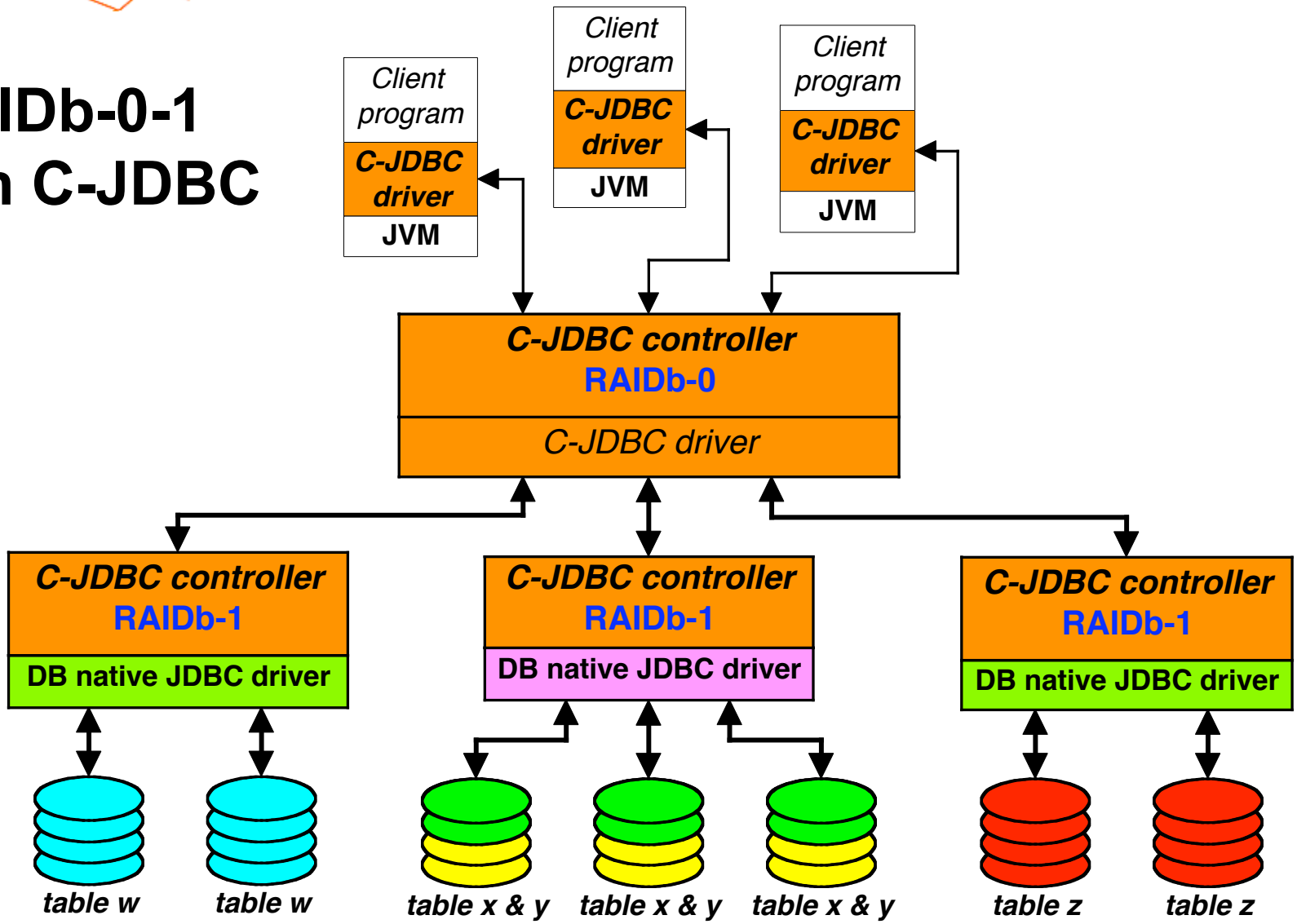
C-JDBC vertical scalability

- RAIDb-1-1 with C-JDBC
- no limit to composition deepness



C-JDBC vertical scalability

RAIDb-0-1 with C-JDBC

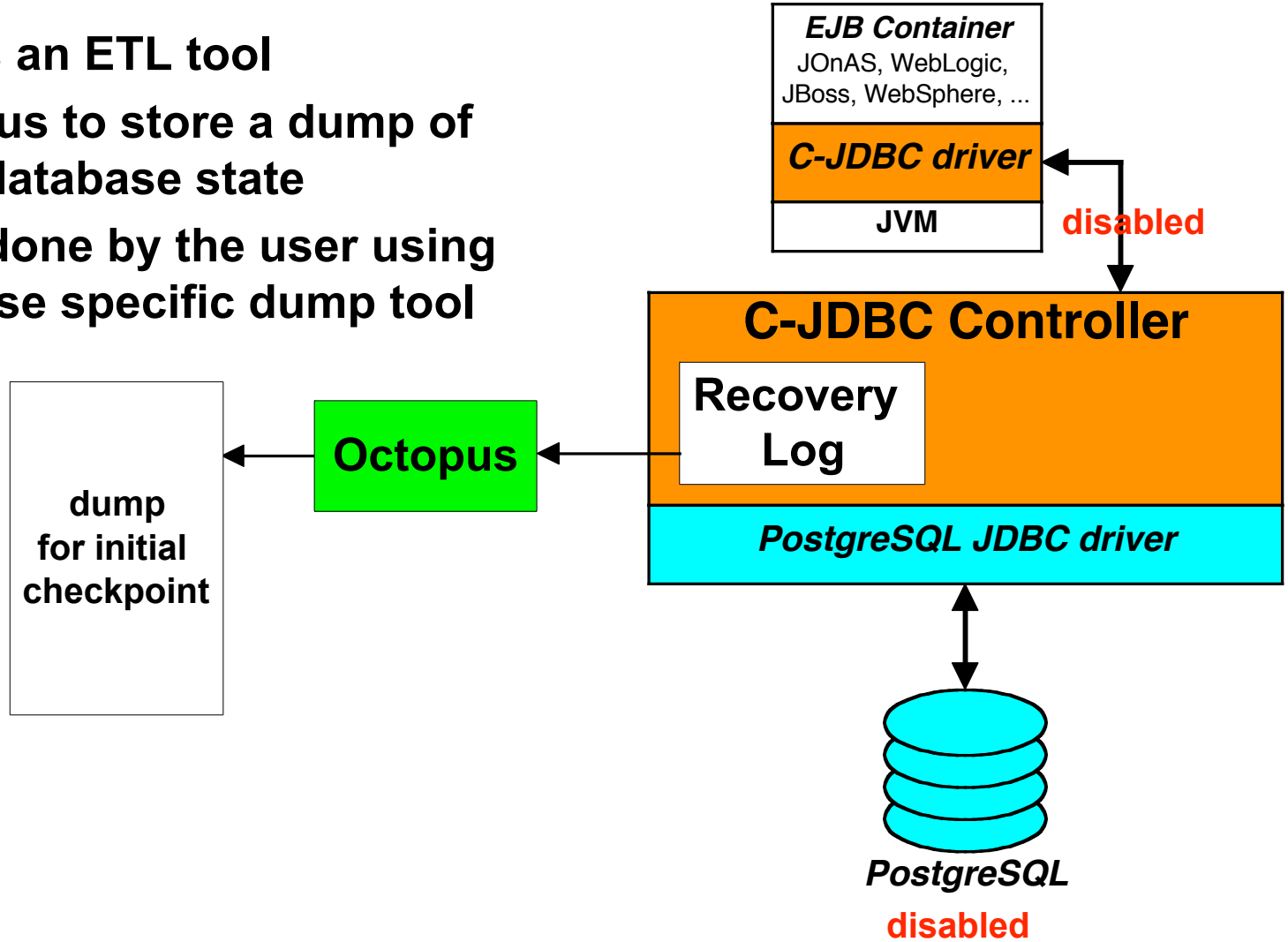


Outline

- Motivations
- RAIDb
- C-JDBC
  - Overview
  - Internals
  - Scalability
  - Checkpointing and Recovery
- Performance
- Lessons learned
- Conclusion

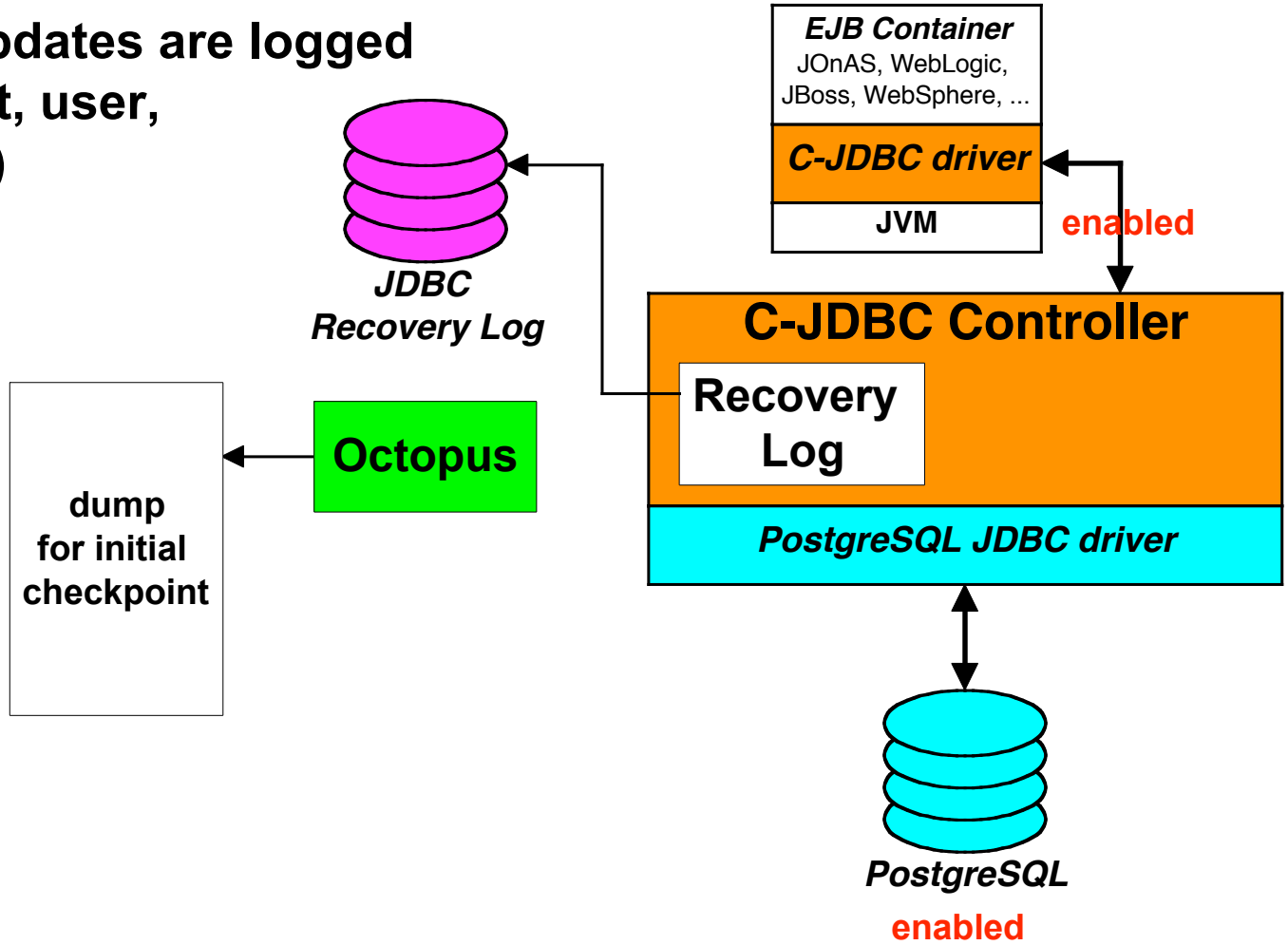
Checkpointing

- Octopus is an ETL tool
- Use Octopus to store a dump of the initial database state
- Currently done by the user using the database specific dump tool



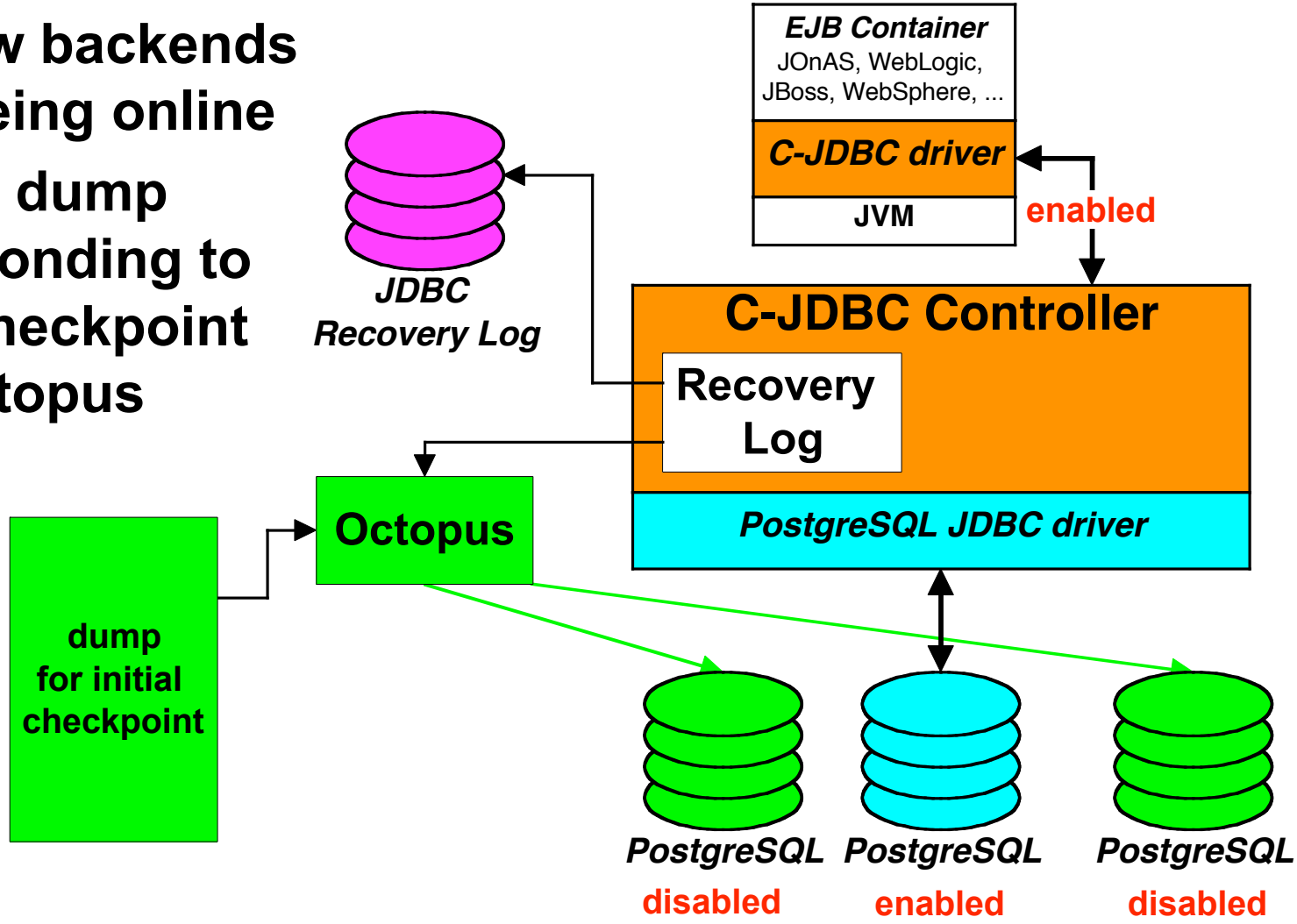
Checkpointing

- Backend is enabled
- All database updates are logged (SQL statement, user, transaction, …)



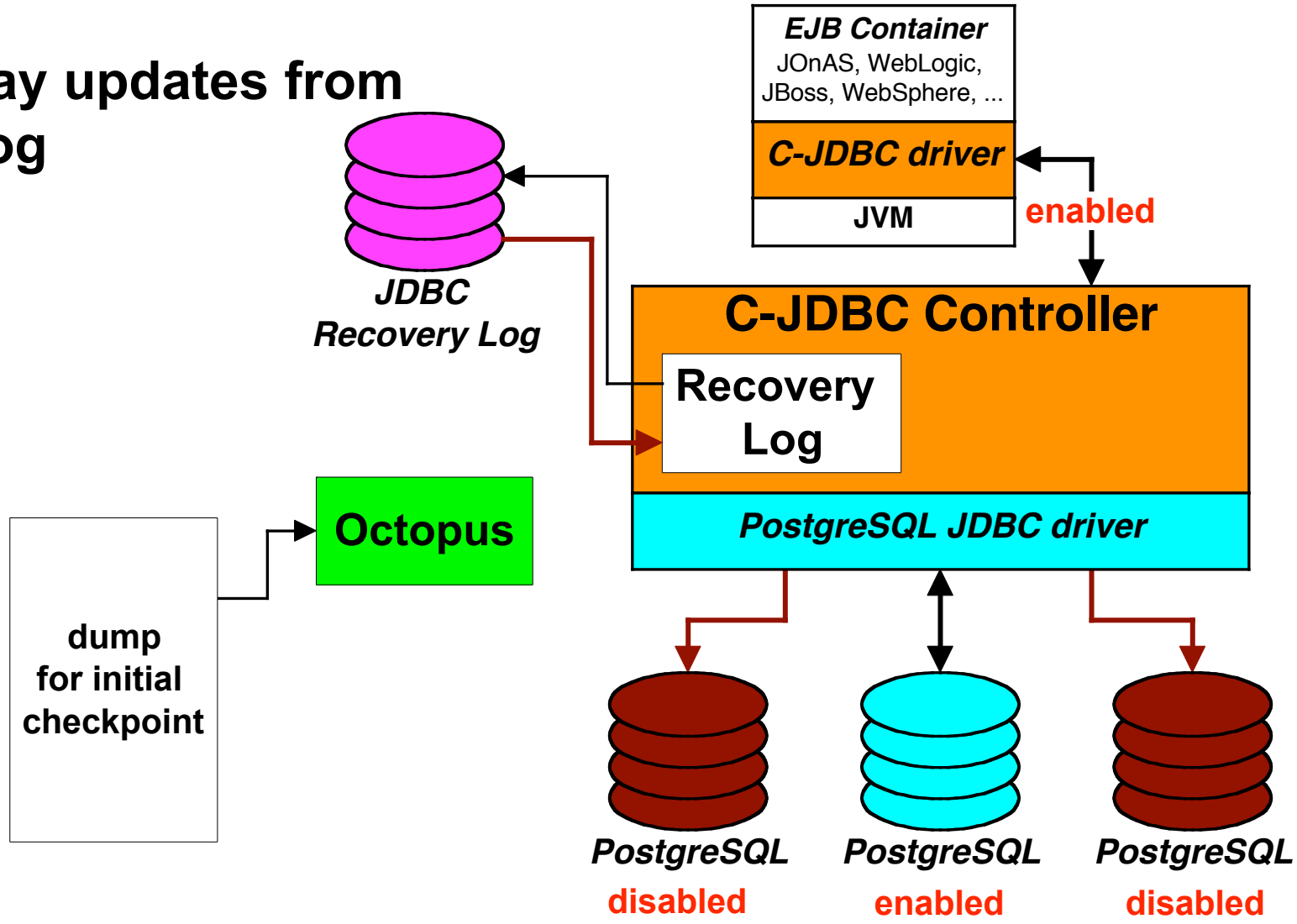
Checkpointing

- Add new backends while being online
- Restore dump corresponding to initial checkpoint with Octopus



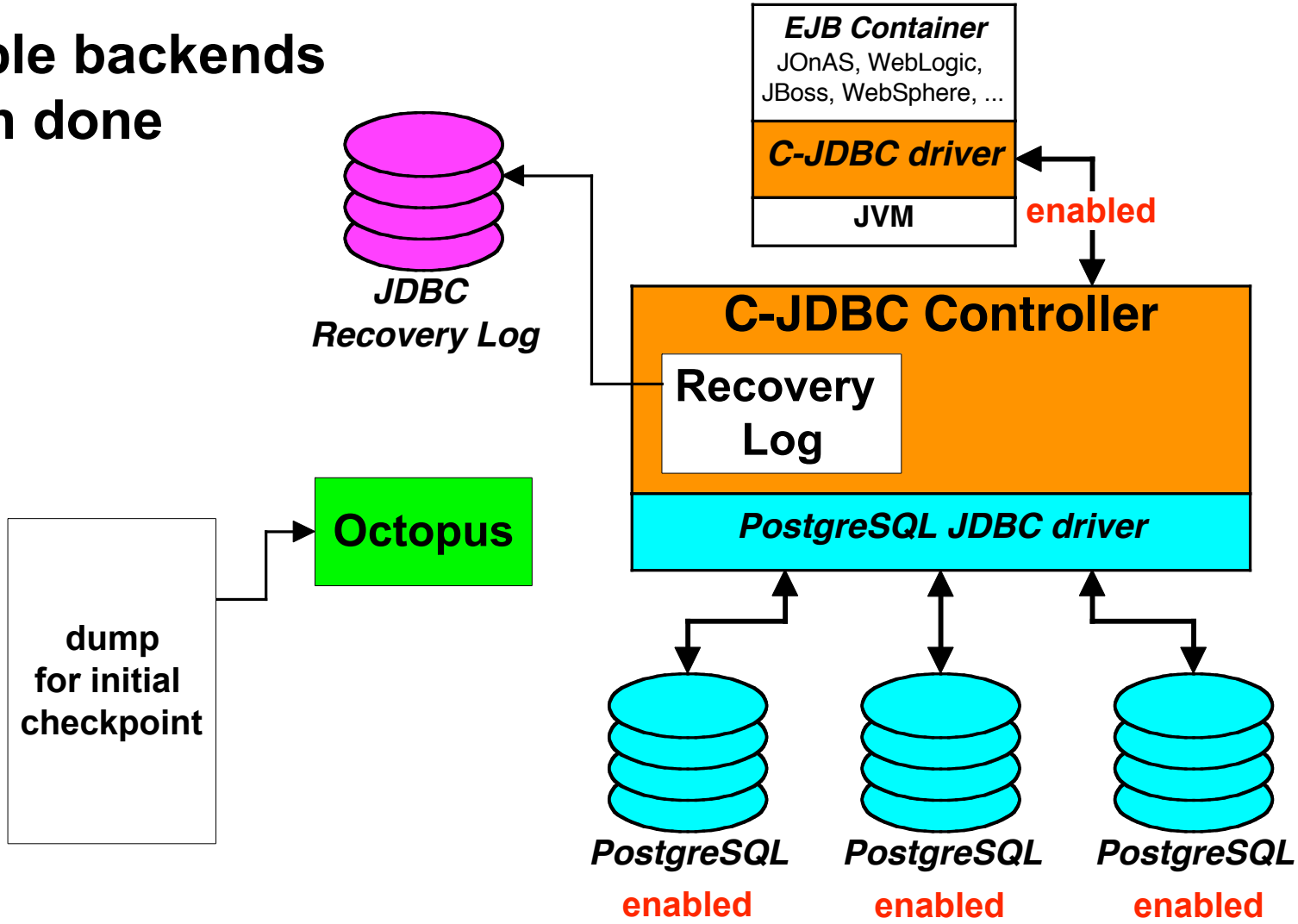
Checkpointing

- Replay updates from the log



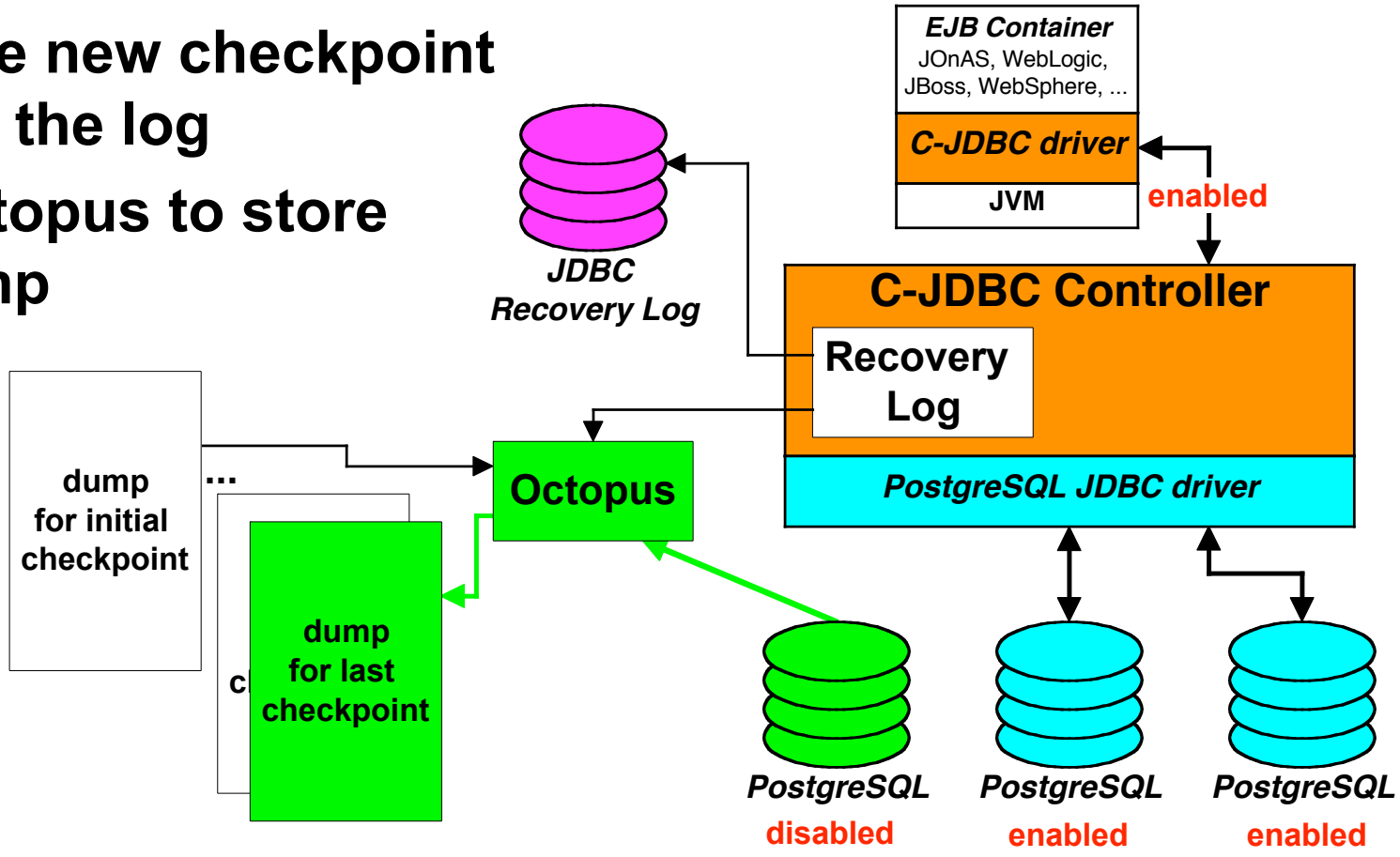
Checkpointing

- Enable backends when done



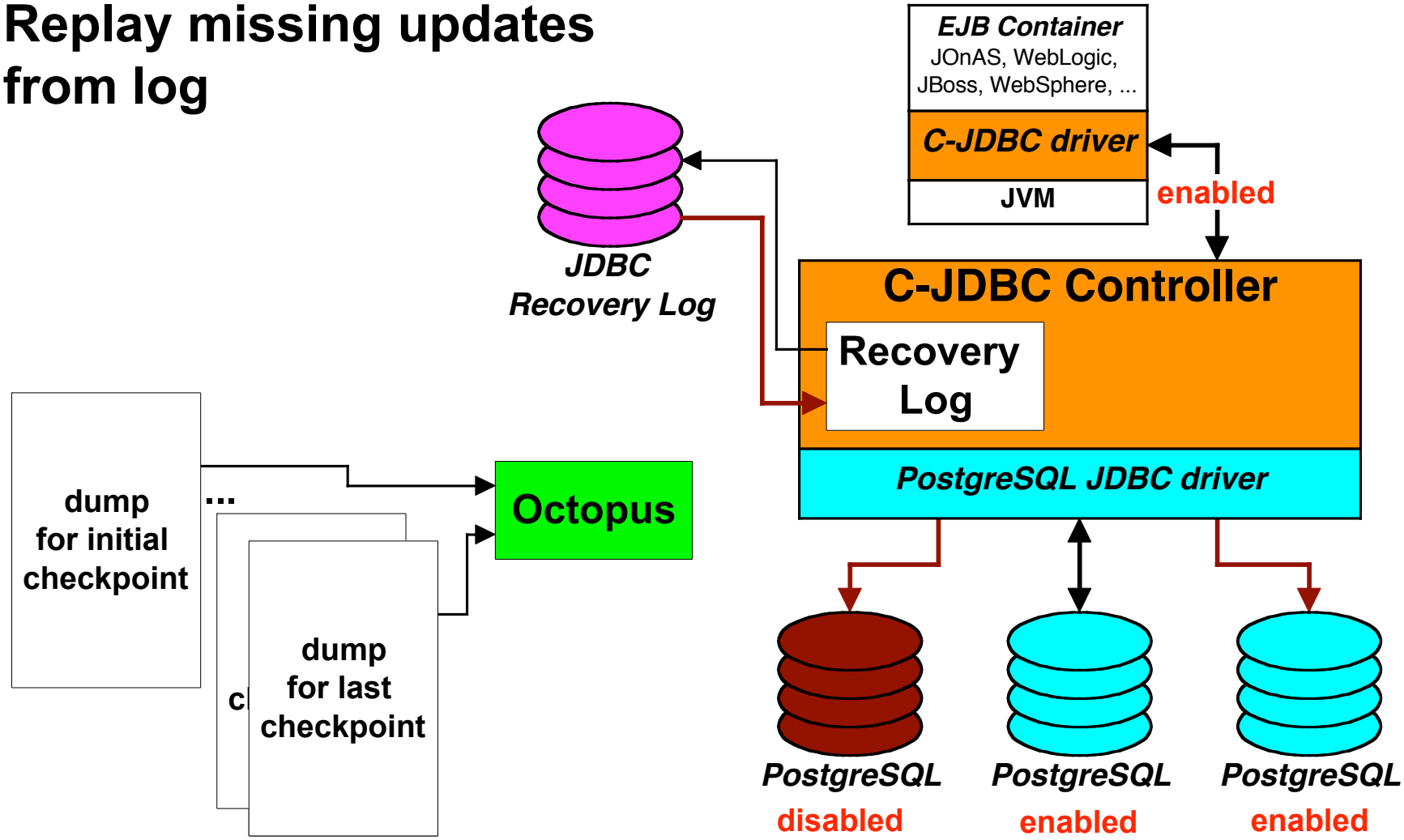
Making new checkpoints

- Disable one backend to have a coherent snapshot
- Mark the new checkpoint entry in the log
- Use Octopus to store the dump



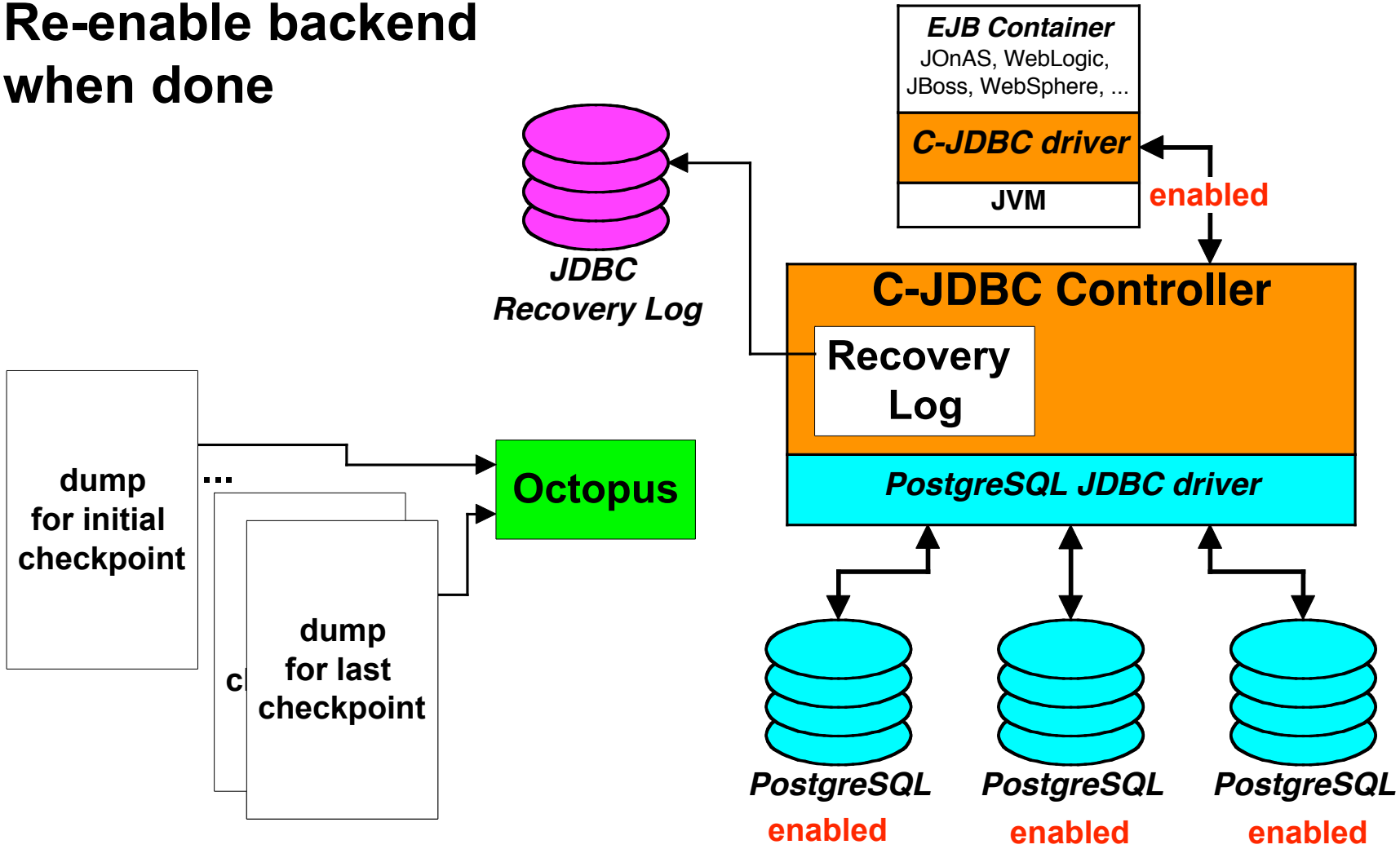
Making new checkpoints

- Replay missing updates from log



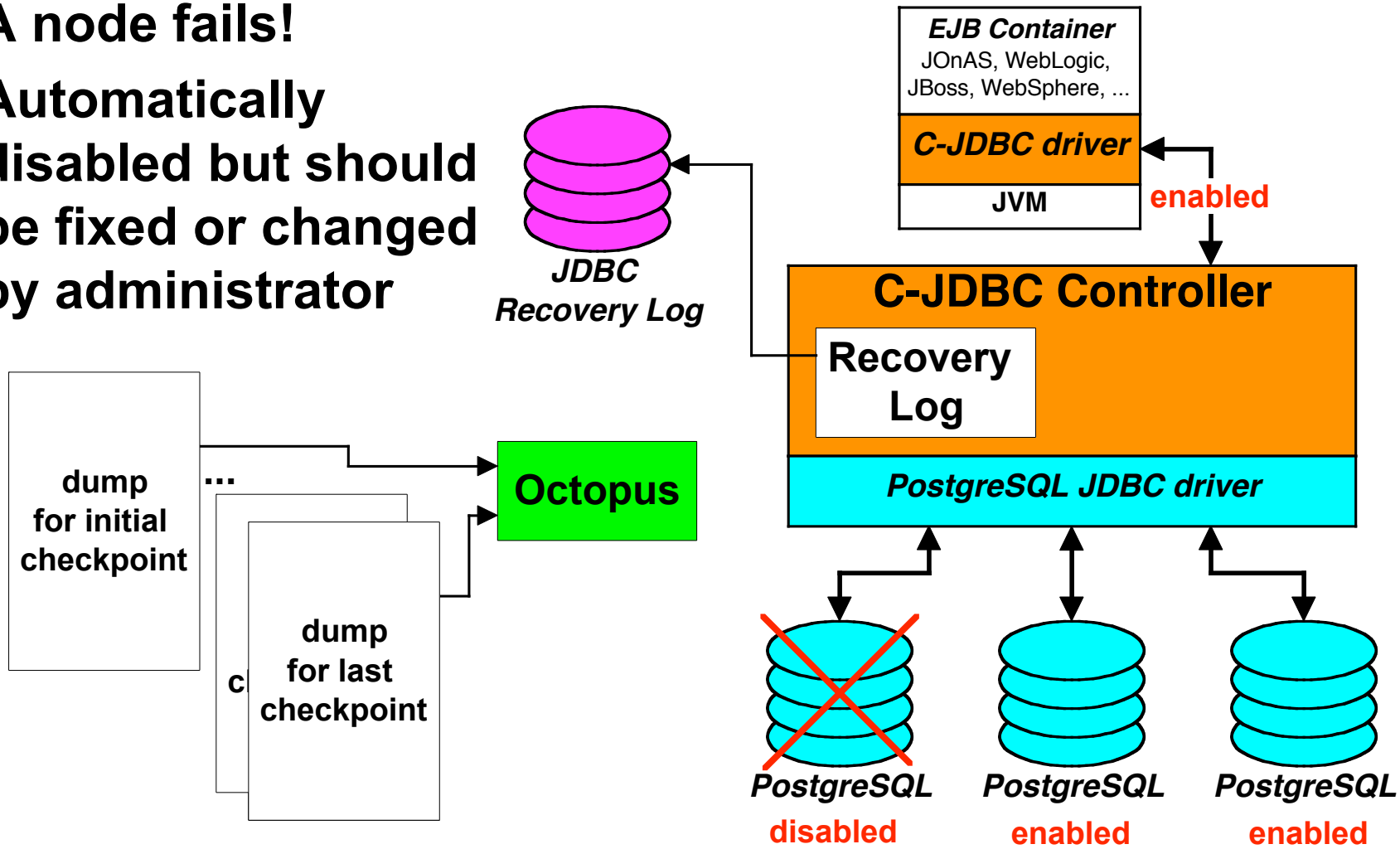
Making new checkpoints

- Re-enable backend when done



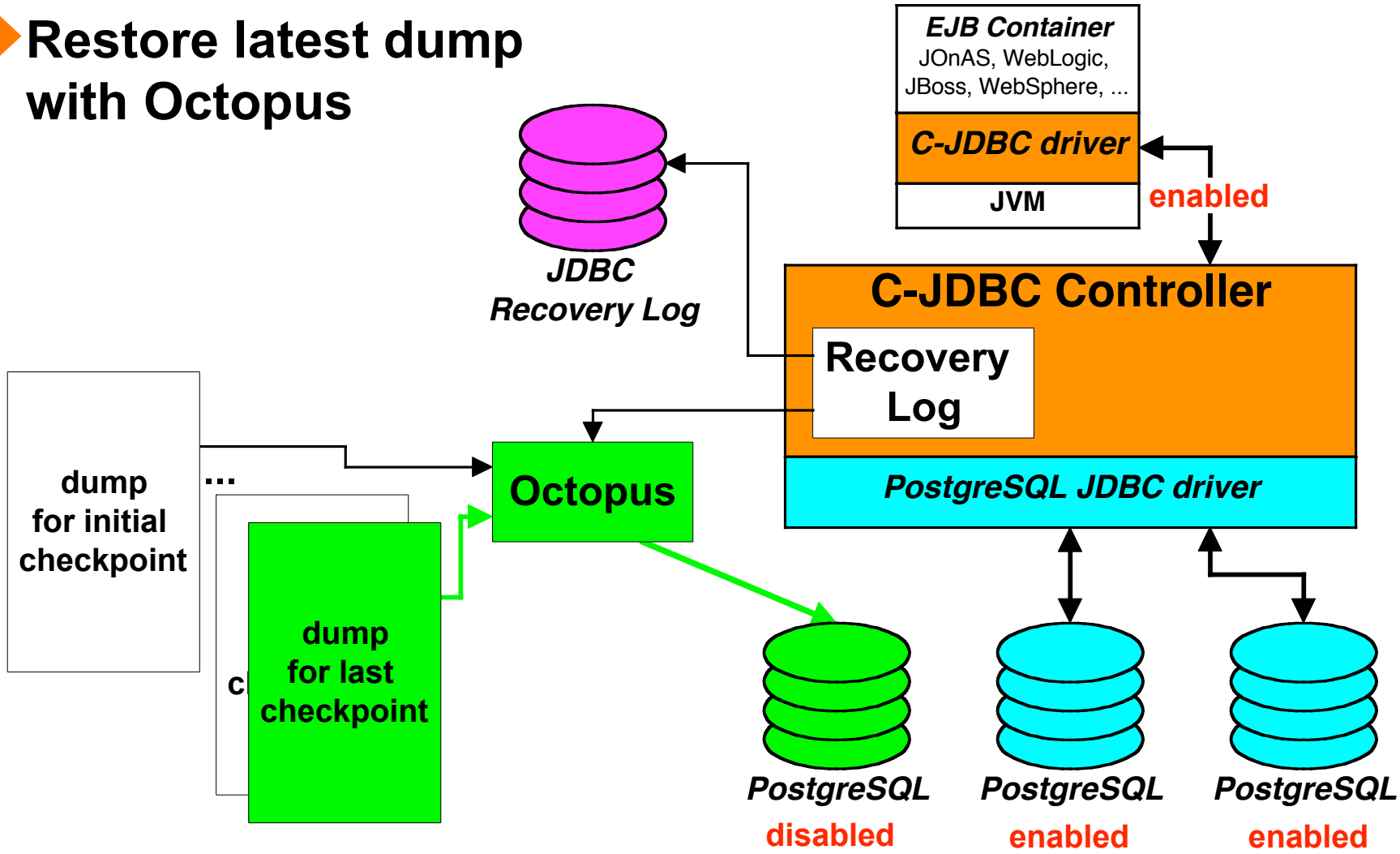
Recovery

- A node fails!
- Automatically disabled but should be fixed or changed by administrator



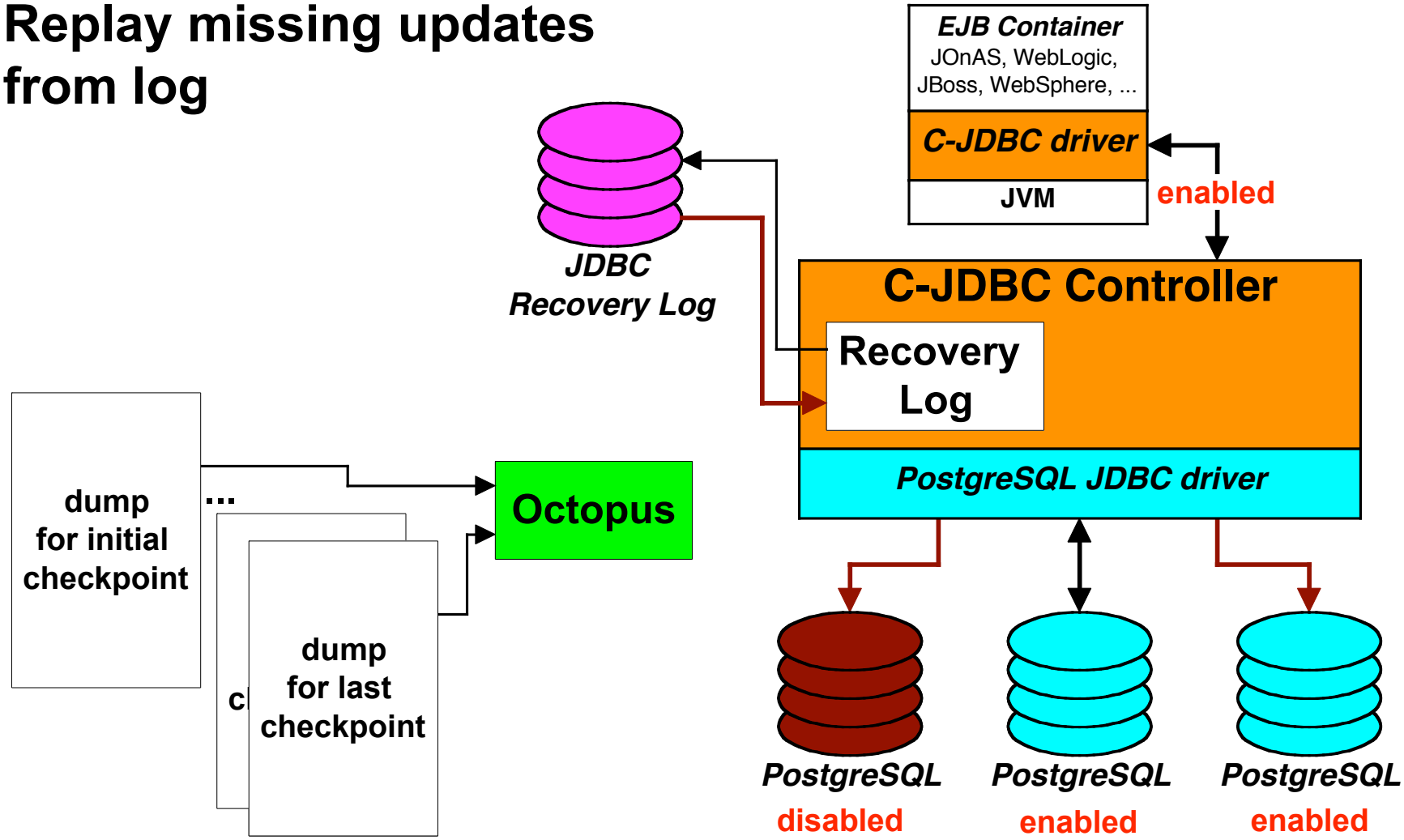
Recovery

- Restore latest dump with Octopus



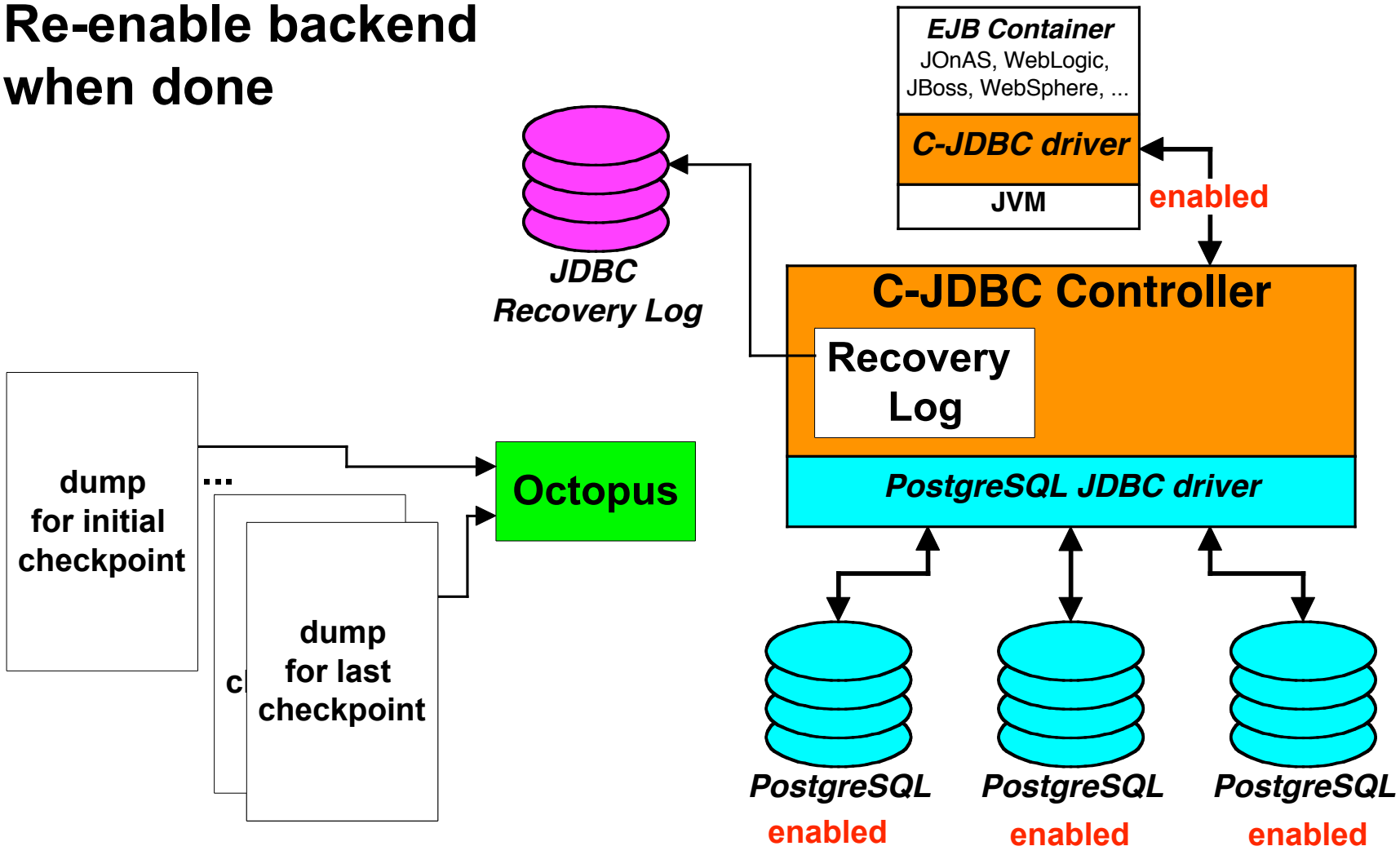
Recovery

- Replay missing updates from log



Recovery

- Re-enable backend when done

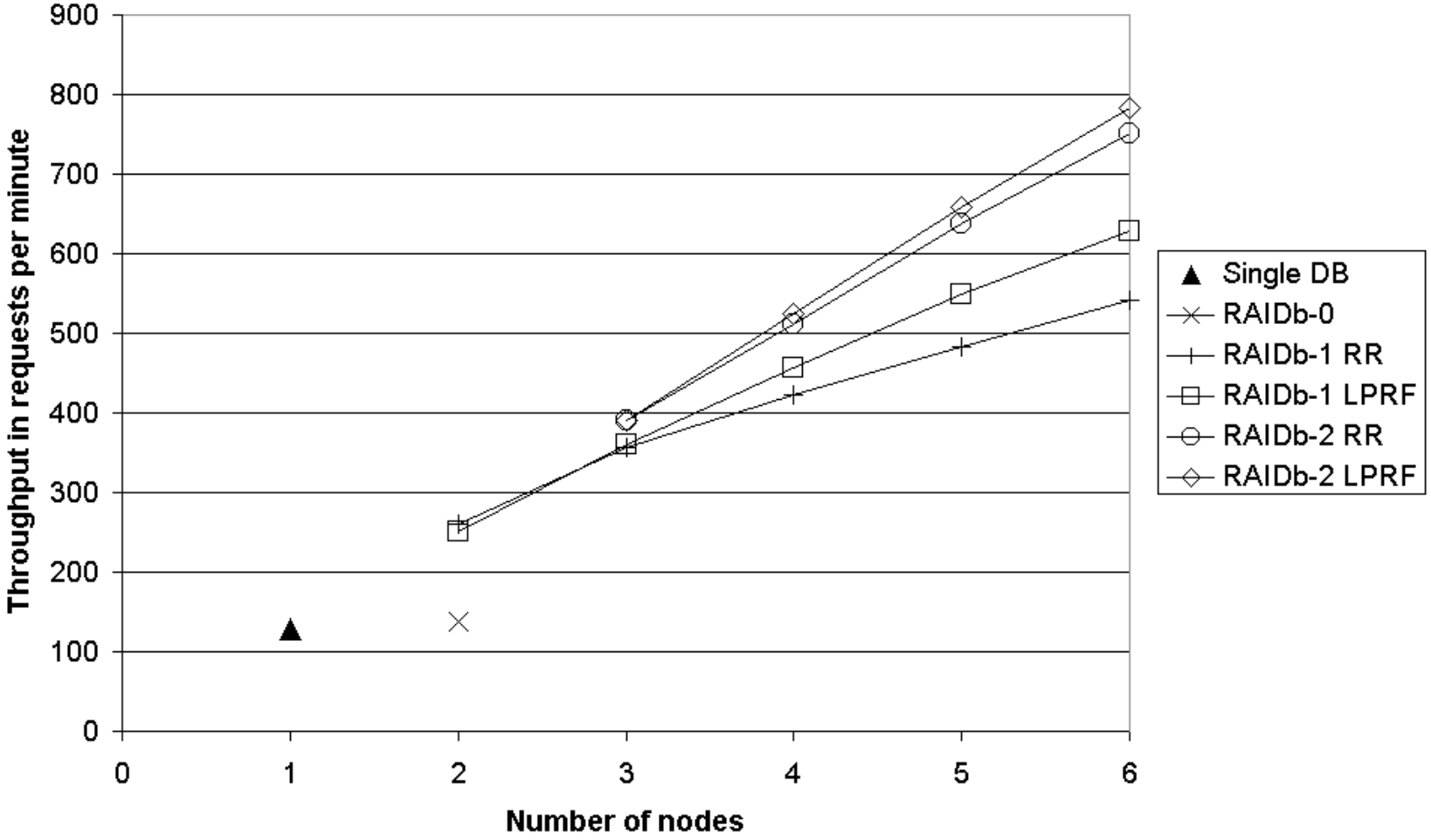


Outline

- Motivations
- RAIDb
- C-JDBC
- Performance
- Lessons learned
- Conclusion

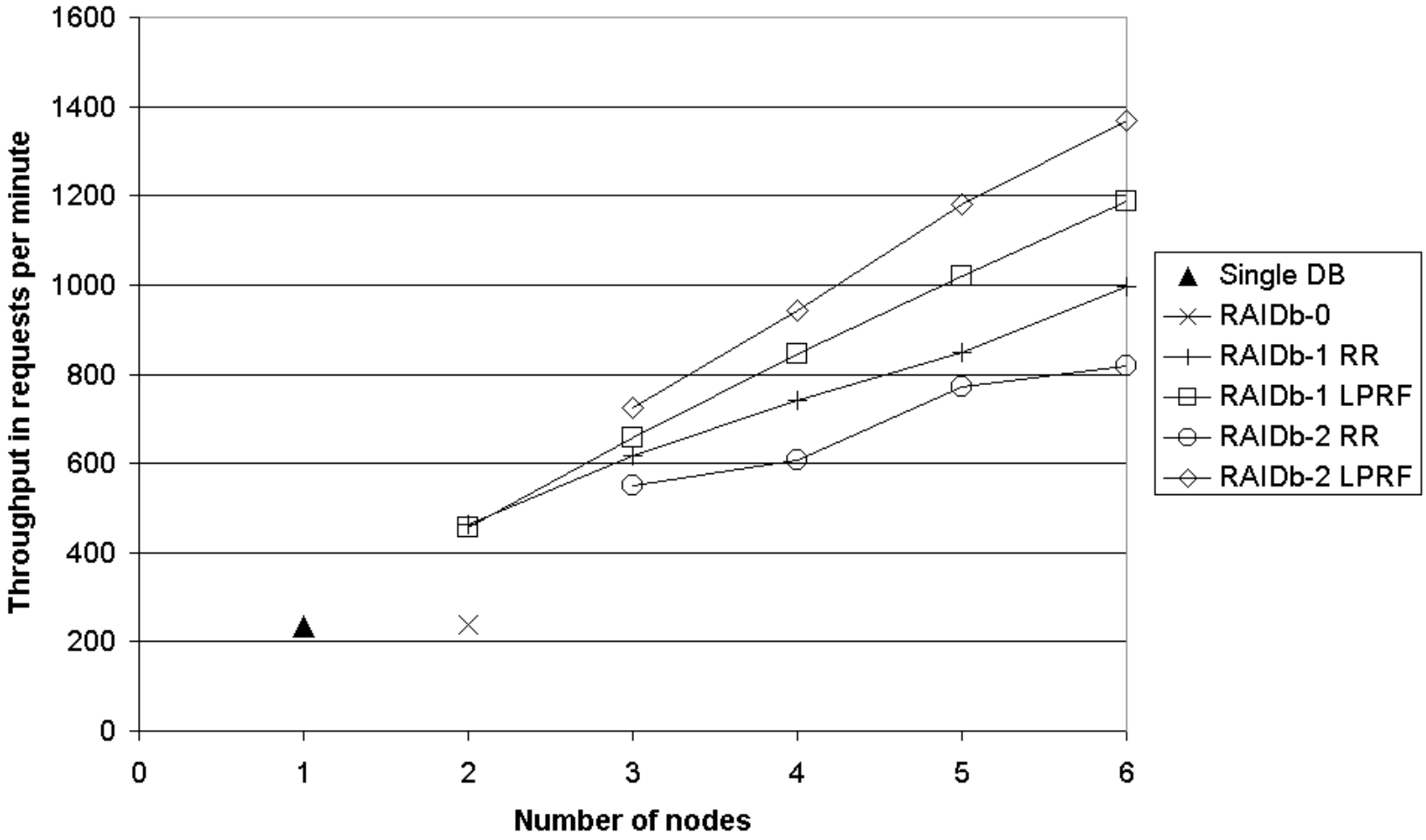
TPC-W

Browsing mix performance



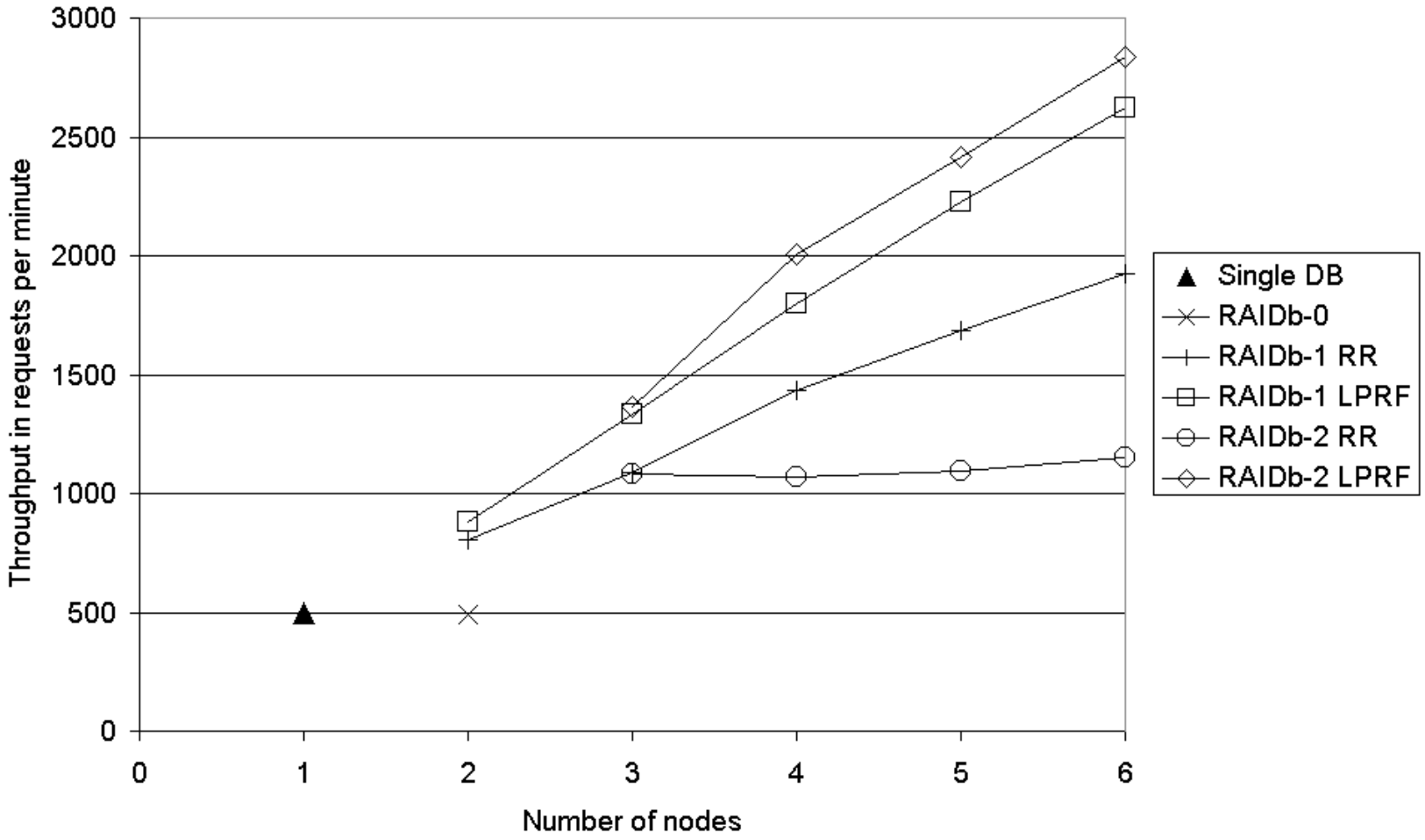
TPC-W

Shopping mix performance



TPC-W

Ordering mix performance



Fine-grain caching

- **Cache hit rate with TPC-W**
  - **browsing mix**
  - only one database backend

	Throughput	Response time	Hit rate
No cache	9.1 req/s	3.30s	
Table	12.9 req/s	1.96s	12.6%
Column	16 req/s	1.36s	48.8%
Column+ single-row	16 req/s	1.35s	49.2%

Fine-grain caching

- **Cache hit rate with TPC-W**
  - **shopping mix**
  - only one database backend

	Throughput	Response time	Hit rate
No cache	12.8 req/s	3.11s	
Table	13.5 req/s	2.58s	3.5%
Column	19.0 req/s	0.93s	30.0%
Column+ single-row	20.2 req/s	0.84s	30.4%

Outline

- Motivations
- RAIDb
- C-JDBC
- Performance
- Lessons learned
- Conclusion

Why did we design C-JDBC ? ... Reloaded

- **Features we wanted ordered by priority**
  - scalability
  - on commodity hardware
  - using open source databases
  - fault tolerance (high availability + failover)
  - without modifying the client application

Why users are using C-JDBC ?

- **JDBC standard**
- **Open source solution**
- **Features they want ordered by priority**
  - fault tolerance (high availability + failover) [was 4/5]
  - using ~~open source~~ existing databases [was 3/5]
  - on commodity hardware [was 2/5]
  - administration tools [was not]
  - security [was not]
  - scalability [was 1/5]
  - without modifying the client application [was 5/5]

How users are using C-JDBC?

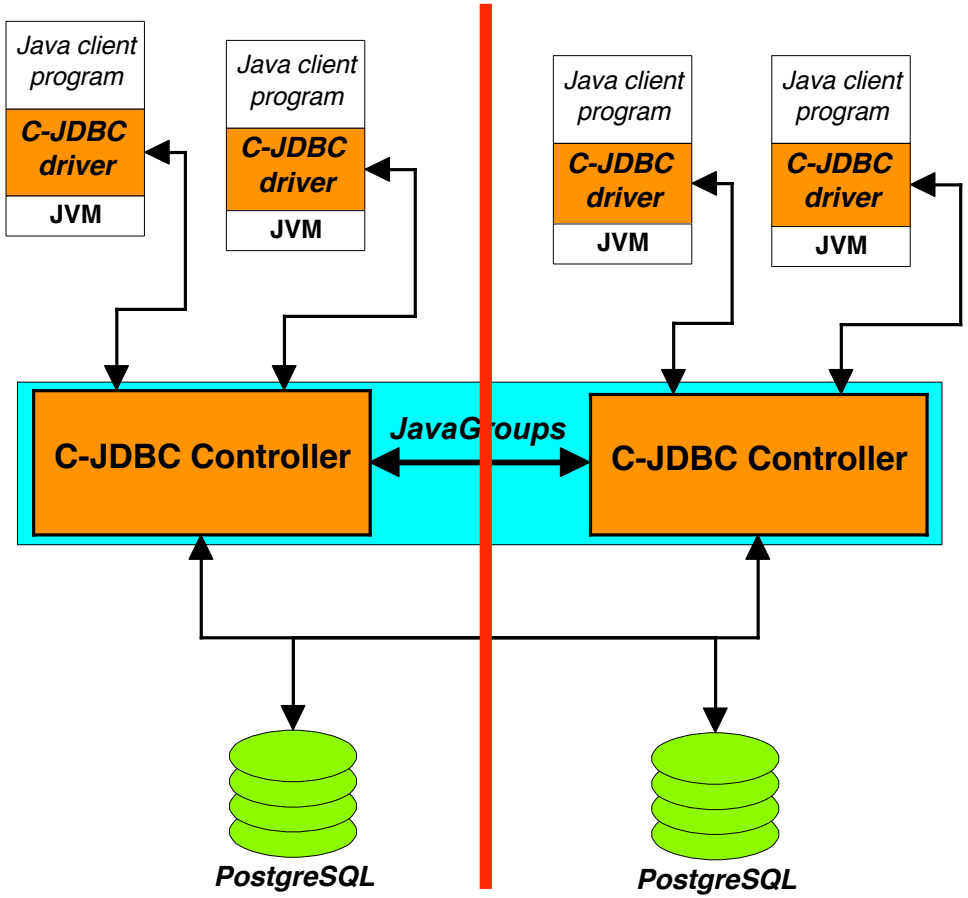
- **Hard to really know**
- **Just default settings!**
- **Most common usage**
  - existing applications (Tomcat/JBoss/JOnAS) with one MySQL/Postgres backend
  - add a second backend for fault tolerance and scalability
- **For things it was not designed for**
  - write mostly workloads
  - distributed databases
  - hosting centers (administration tools missing)

Lessons learned

- **Users do not use it for what it was first designed for**
  - advanced features are never used
  - concerned about ease of use and TCO
- **Default settings are important**
- **Good technology is necessary but not sufficient**
  - [administration] tools are needed
  - minor bugs are ok for open source users

Open problems

- Partition of clusters
- Users want control on failure policy
- Reconciliation must also be user controlled



Open problems

- **Opening the architecture to the users**
  - user defined strategies when a fault or exception occurs
  - which interfaces/callbacks to provide ?
- **Monitoring**
  - needed for more accurate load balancing algorithms
- **Benchmarking**
  - need automatic evaluation of clustered servers
  - platform available: new INRIA 208 itanium-2 cluster
- **Sun Test Suite**
  - should help strengthening C-JDBC code
  - interoperability with J2EE servers

Outline

- Motivations
- RAIDb
- C-JDBC
- Performance
- Lessons learned
- Conclusion

Current status

- **C-JDBC 1.0b15 release**
  - Generic JDBC 2.0 driver
  - Schedulers and load balancers for RAIDb 0, 1 and 2
  - Fine grain query caching
  - JDBC recovery log
  - Logger/request player
  - Java installer
  - User documentation
- **Currently missing**
  - Octopus integration
  - Recovery for horizontal scalability
  - Monitoring/full JMX administration
  - Dynamic reconfiguration

Stats as of Nov 6, 03

Downloads

- total : > 8300 downloads since may 2003
- last 30 days : > 2800 downloads
- > 430.000 hits since first release
- 2nd most downloaded ObjectWeb project

Mailing lists

- c-jdbc@objectweb.org: 101 subscribers
- c-jdbc-commits@objectweb.org: 18 subscribers

Team

- 9 committers
- 1 full-time INRIA engineer

Conclusion

- **RAIDb**
  - classification of replication techniques
  - difficult to publish
- **C-JDBC**
  - open platform for database replication at the middleware level
  - RDBMS independent
  - no application modification required
- **Lot of features missing … join us !**

Questions ?

Visit http://c-jdbc.objectweb.org

c-jdbc@objectweb.org

Bonus slides

ObjectWeb
Open Source Middleware

Fine-grain caching

- increased throughput
- better response time
- even with a single database backend

How do we build a community?

- **Necessary features (but not sufficient)**
  - open source
  - standard API
  - responsiveness on the mailing list
- **Visibility**
  - Web: slashdot, TheServerSide, freshmeat, …
  - Conferences: JAX, Middleware, LinuxWorld, ICAR, …
- **Our weak points**
  - no detailed design documentation
  - beta phase

How do we interact with the user community?

- **only one mailing list**
- **being very responsive on the mailing list**
  - reply even if we don't have a response yet
  - no direct communication with team but share everything on the mailing list
- **benefit from engineers who work 1 week full-time to evaluate C-JDBC for their corporation**
- **plan every feature request in the task list**

How do we interact with the developer community?

- **single user/developer mailing list**
- **post all design questions/choices on the mailing list**
  - most users use default settings
  - hard to get feedback about usage
- **very permissive to accept new commiters**
  - 8 committers (3 outside ObjectWeb – 2 full time)
  - 2 contributors who didn’t want to become committers
  - no problem so far
- **involve people in testing**

Lessons learned

- **Visibility**
  - perpetual involvement
  - time consuming but necessary
- **Responsiveness to user queries**
  - always on the mailing list
  - makes first impression for many users
- **Involve users in all decisions**
- **Be open**
  - source, CVS, contributions, patches, …

Freshmeat.net

- Links to projects
- Users can subscribe to be notified of new releases
- Need to register project in all possible categories to have good visibility
- One release per week is a good timing
- 3 new subscribers with every release