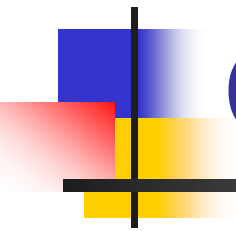
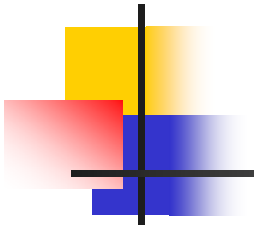


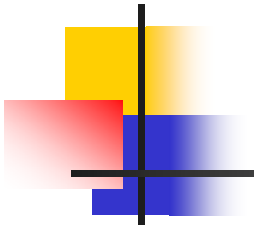
C-JDBC

Emmanuel Cecchet
INRIA, Projet Sardes
http://sardes.inrialpes.fr



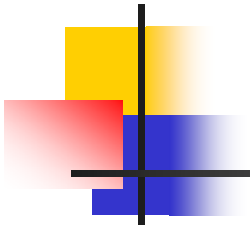
Plan

- Motivations
- Idées principales
- Concepts
- Caching
- Perspectives



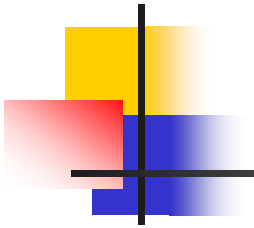
C-JDBC - Motivations

- clustering J2EE
  - serveur Web (switch L4, RR-DNS, …)
  - serveur de servlets (mod_jk, …)
  - serveur d'EJB (travaux de Simon)
  - pas de solution pour les BDs
- perf des applis J2EE limitée par la BD
  - coût des transactions
  - pas d'implantations open-source pour cluster
  - limitation matérielles (petits SMP)
  - solutions commerciales très chères
- serveur d'EJB pas capable d'optimiser les accès ou de cacher les résultats si accès à la BD par un autre tiers
- JDBC est une API bien établie



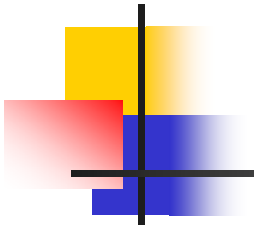
C-JDBC – Etat de l'art

- clustering BD utilisé pour le partitionnement mais pas pour la réplication
  - pb de perf sur les joins
  - utilisation de views pour améliorer les perfs
  - décision de load balancing centralisée
  - pas d'offre open-source
  - les offres commerciales (Oracle, DB2) ont des solutions propriétaires
- support dans le driver JDBC
  - en général pas plus que du pooling de connexions
  - clustering ad-hoc dans WebLogic pour Oracle (limité au round-robin)



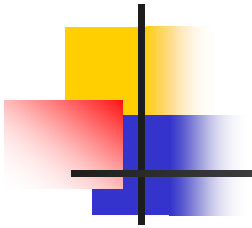
C-JDBC – Idées

- C-JDBC = RAIDb
- fournir à des clients JDBC un accès transparent à un cluster de bases de données
- performance/disponibilité/tolérance aux fautes
- gestion des BD hétérogènes
- caching, logging, monitoring, …
- Hypothèses
  - tous les accès à la BD se font à travers un driver JDBC
  - Jim Gray a tort : eager consistency peut marcher avec des BD répliquées



C-JDBC – Idées

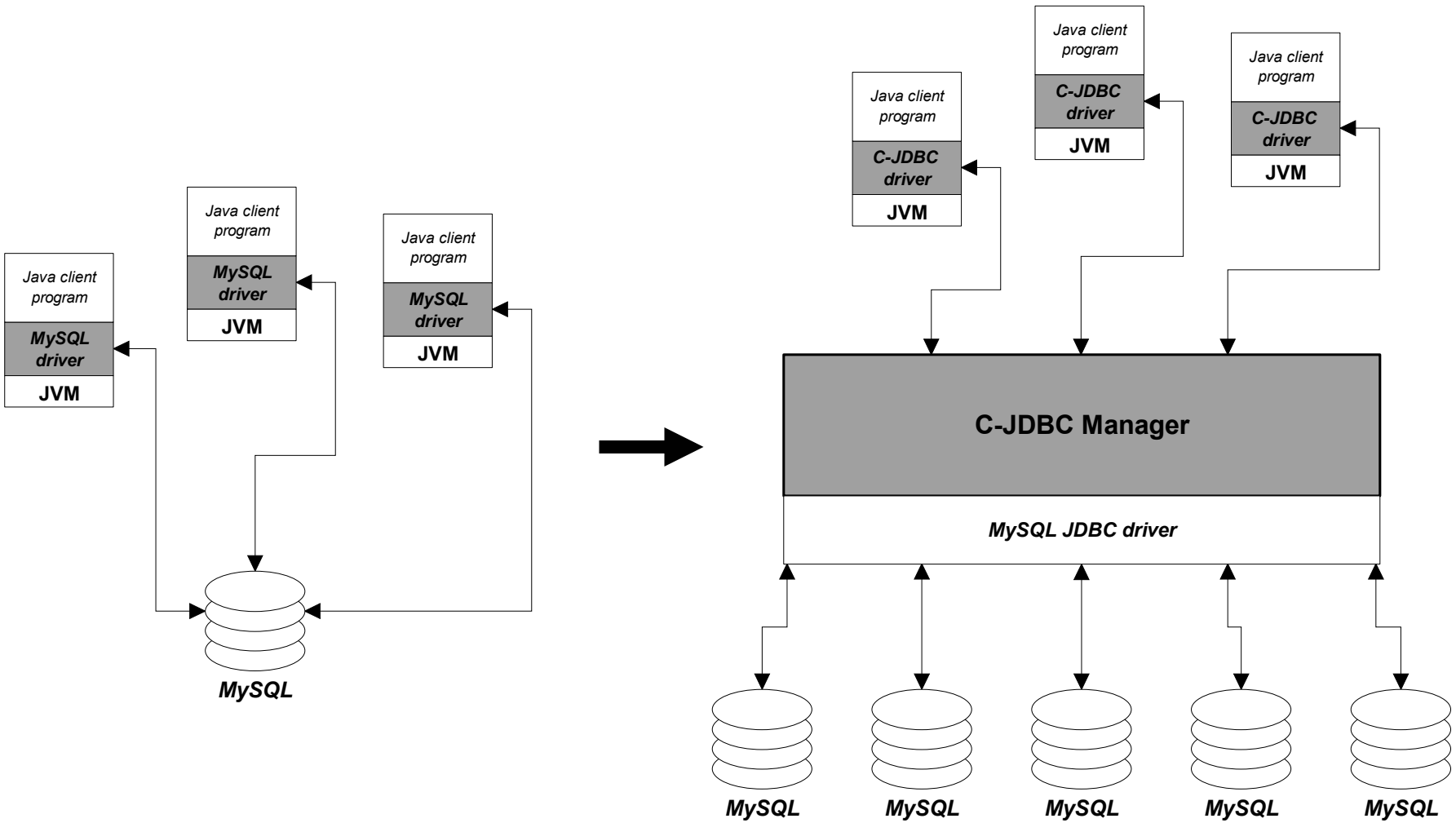
- duplication de la BD sur toutes les machines
- load balancing des reads et broadcast des write
- Propriétés
  - architecture modulaire,
  - reconfigurable,
  - adaptable aux ressources matérielles et aux besoins des applis,
  - portable (écrit en Java)



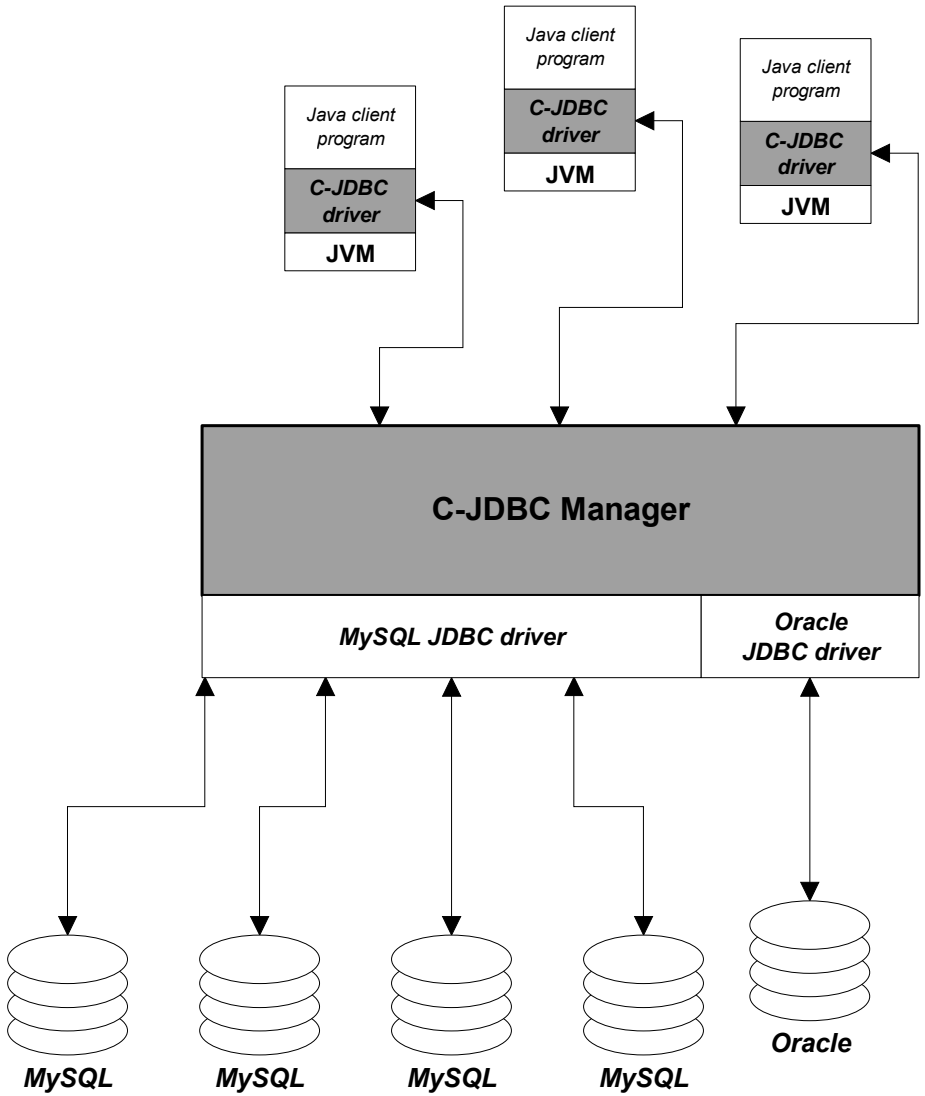
Plan

- Motivations
- Idées principales
- Concepts
- Caching
- Perspectives

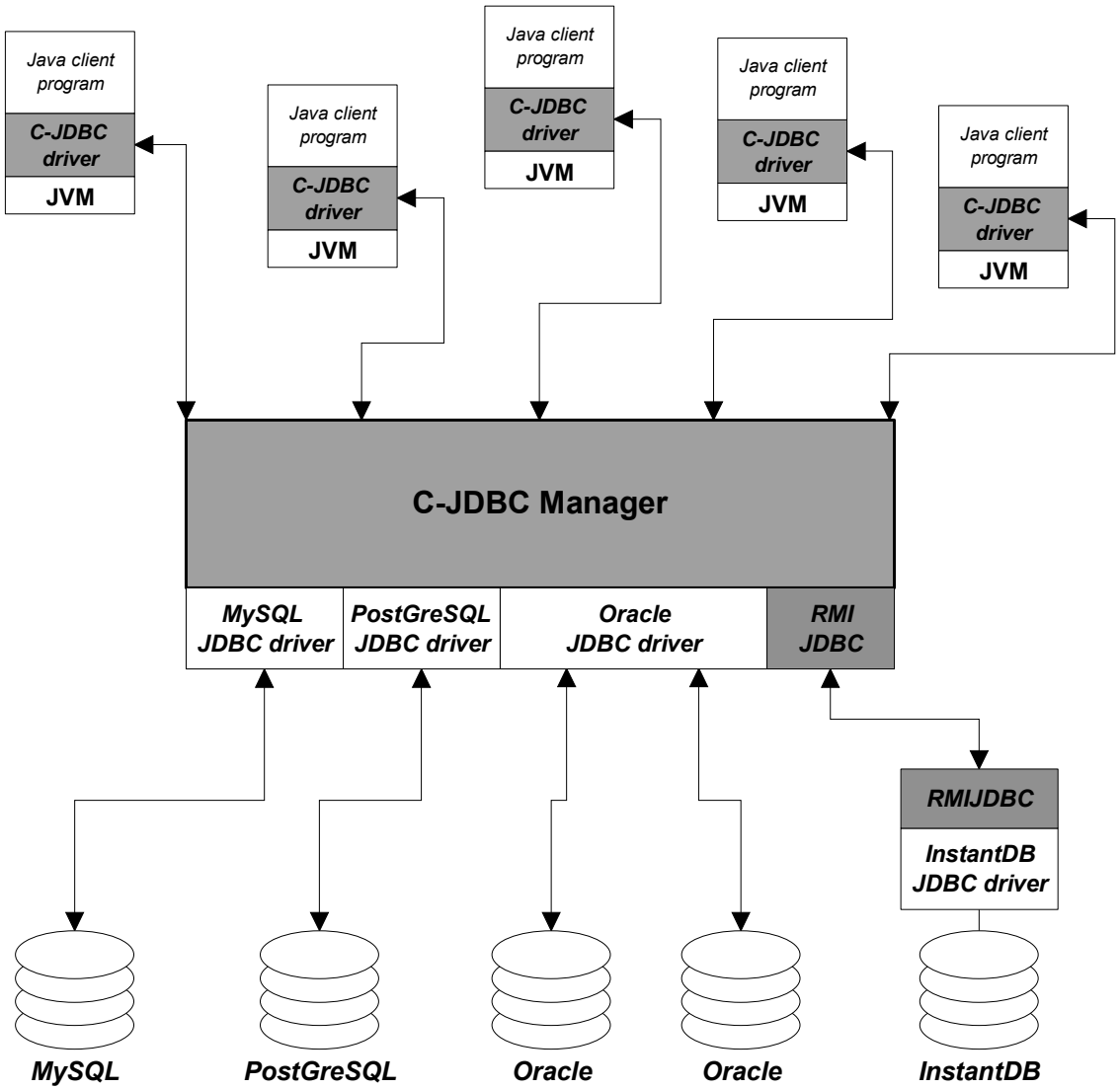
C-JDBC – Vue d'ensemble



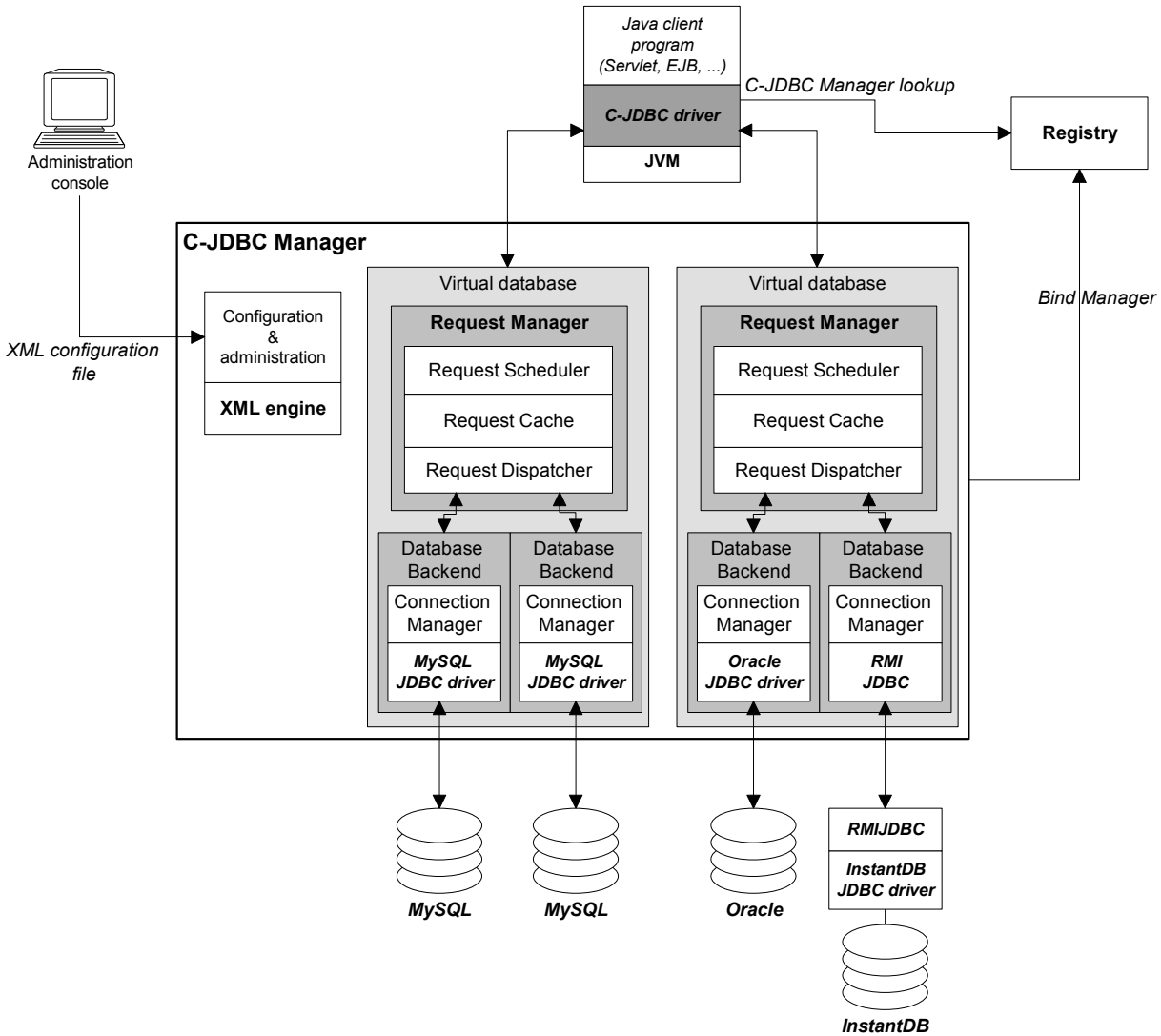
Oracle + C-JDBC



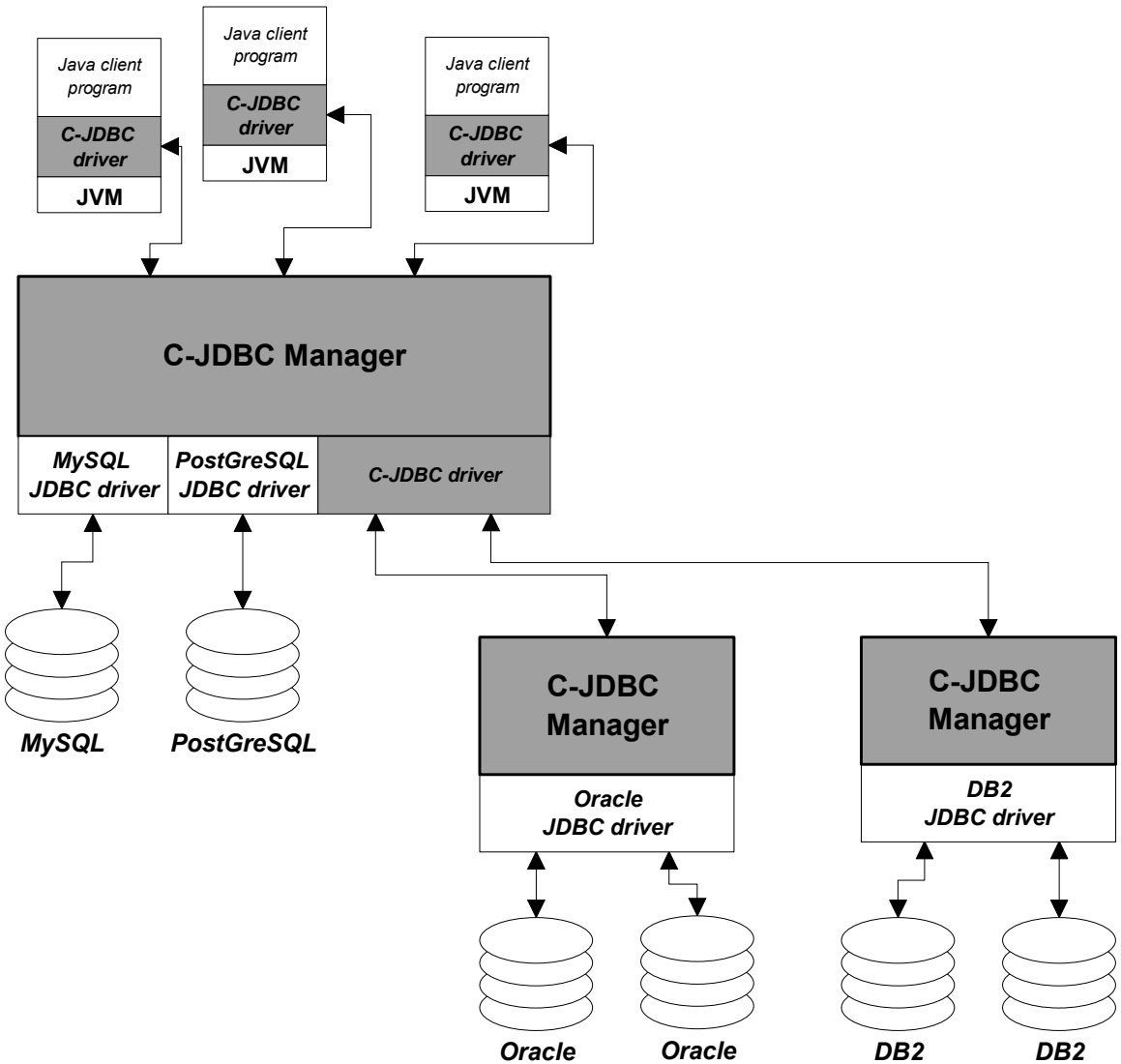
C-JDBC – Hétérogénéité



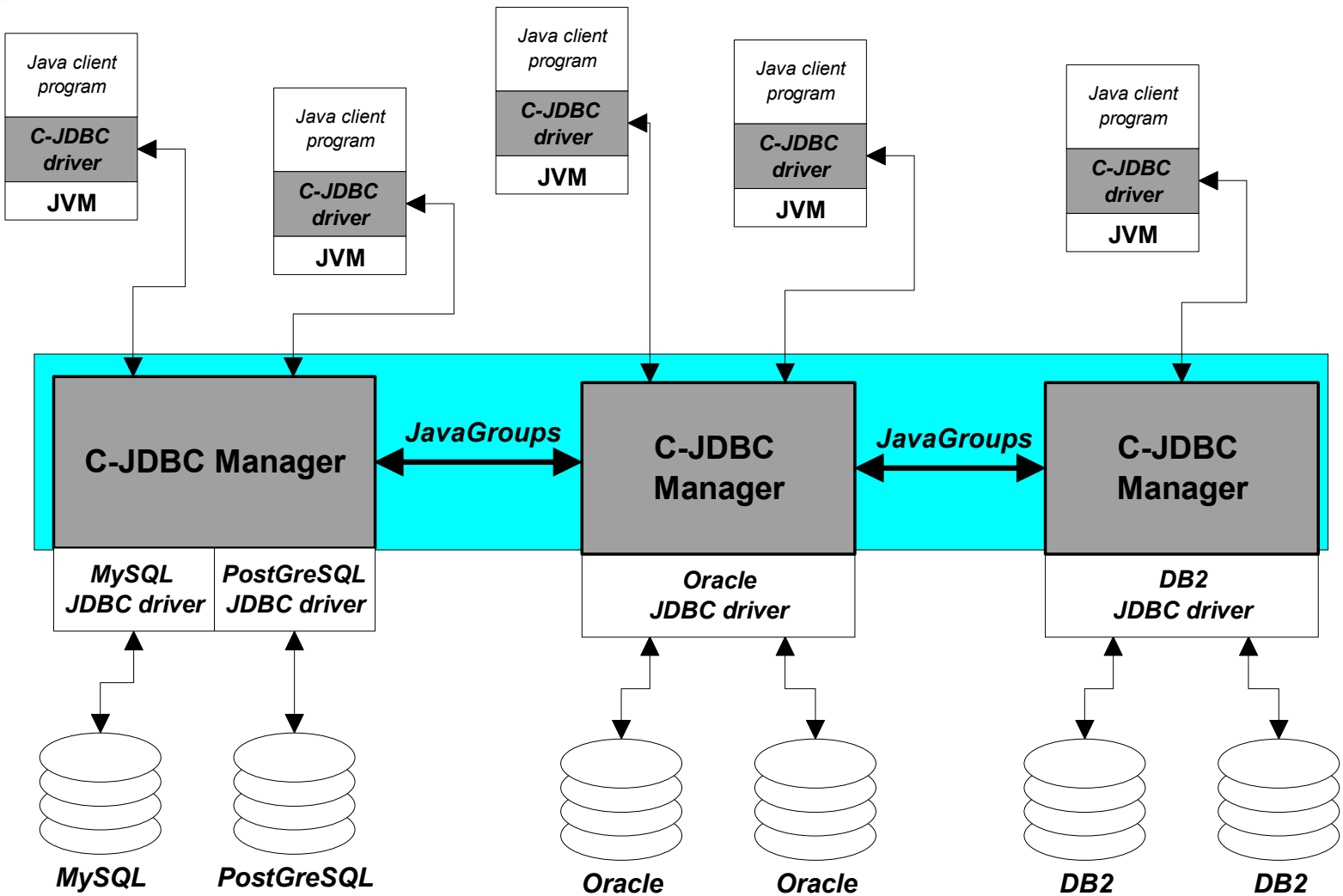
C-JDBC Manager

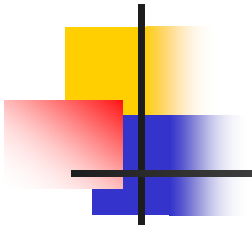


C-JDBC – Scalabilité verticale



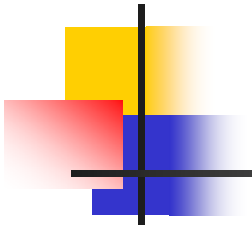
C-JDBC – Scalabilité horizontale





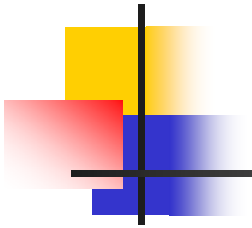
Pistes de recherche - 1

- équilibrage de charge,
  - weighted round-robin,
  - spécialisation selon les requêtes (possibilité de partitionnement de la BD sur la grappe),
  - monitoring/évaluation de la charge des BD.
- scheduling de requêtes
  - étude des possibilités de réordonnancement des requêtes pour améliorer le taux du hit du cache ou les temps de réponse
  - favorisation des batch-updates
  - retard volontaire des écritures, out-of-order write, …
- passage à l'échelle
  - étude architecturale : cluster horizontal à la JavaGroup, diffusion en arbre, …
  - limitation de la diffusion des écritures



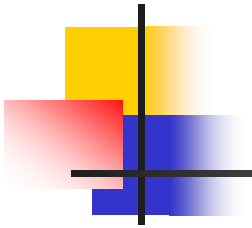
Pistes de recherche - 2

- caching
  - caching de requêtes SQL dans le driver
  - granularités multiples avec BD cache pour requêtes complexes
  - caching de document XML, précalcul de transformations avec feuilles de styles (XSLT)
  - edge-side caching (sujet de recherche à part entière qui aurait plus d'intérêt au niveau Web Services)
- (re)configuration dynamique
  - ajout dynamique d'un nœud (maintien de la cohérence)
  - changement de politique au niveau scheduler, cache ou load balancing
  - détection automatique des schémas des BDs



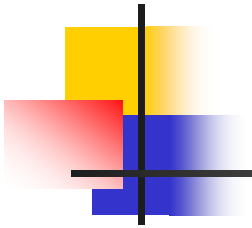
Pistes de recherche - 3

- exploitation de l'hétérogénéité
  - spécialisation de traitement en fonction des spécificités des BDs,
  - réécriture à la volée de requête pour support de l'hétérogénéité
  - synthèse des propriétés de la grappe hétérogène dans le driver
- tolérance aux fautes / reprise sur pannes
  - logs
  - transactions
  - sécurité
- monitoring



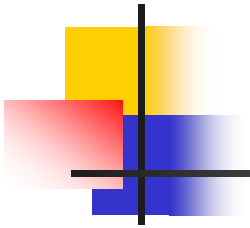
Plan

- Motivations
- Idées principales
- Concepts
- Caching
- Perspectives



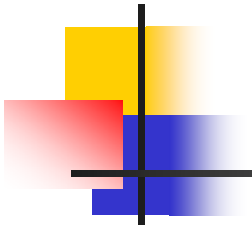
Fine-grain caching

- cacher les réponses des requêtes SQL
- limiter les invalidations en analysant les requêtes SQL
  - Query Cache dans les BD utilise invalidations sur les tables
  - invalidation sur les colonnes
  - optimisation des SELECT uniques
  - peut être possible d'exécuter les UPDATE/DELETE sur le cache dans certains cas
- possibilité de parser les requêtes dans le driver JDBC (et de cacher le parsing)



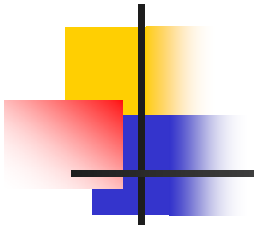
Fine-grain caching

- Taux de hits du cache avec RUBiS
  - pas d'invalidation (cache non cohérent) : 82%
  - invalidation par table : 15%
  - invalidation par colonne : 53%
  - + unique : 54%
  - ++ BD cache : 57.5%
  - +++ unique update : 71%



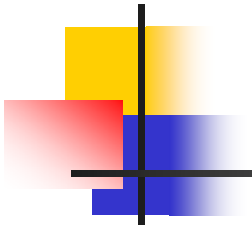
Fine-grain caching

- TPC-W (papier OSDI)
  - débit amélioré jusqu'à x2
  - temps réponse diminué jusqu'à x3
  - obligé d'avoir invalidations à grain fin
  - position du cache (serveur web, bd ou entre les 2) a peu d'impact sur les perfs
  - stratégie pour caches multiples :
    - workload fortement read-only : cache à 2 niveaux (serveur web + machine dédiée)
    - workload ayant plus d'écritures : cache centralisé partagé suffisant



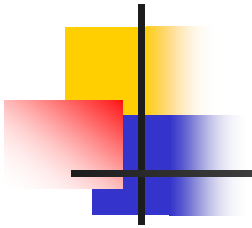
Plan

- Motivations
- Idées principales
- Concepts
- Caching
- Perspectives



Etat actuel

- Code disponible
  - driver JDBC générique utilisant RMI
  - conflict-aware scheduling
  - load-balancing: simple round-robin
  - scalabilité horizontale et verticale opérationnelle mais non évaluée
  - reconfiguration dynamique embryonnaire
  - caching à granularité variable
- Manque
  - tolérance aux fautes/reprise sur panne
  - logging
  - monitoring



Perspectives

- Lancer C-JDBC comme projet ObjectWeb
- Peut s'interfacer avec n'importe quel container Web ou EJB
- Gros potentiel de diffusion
- Nécessaire pour le benchmark de JOnAS cluster et/ou transactions réelles avec RUBiS
- Intégration dans l'offre clustering de JOnAS
  - pas de modification de JOnAS
  - possibilité d'intégrer l'admin/monitoring dans les outils de JOnAS ?