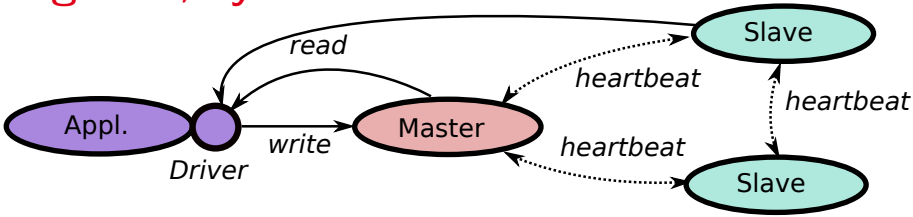
# Bases de données documentaires et distribuées, http://b3d.bdpedia.fr

## Distribution et réplication dans MongoDB

# MongoDB, système distribué *maître-esclave*




**Un replica set, MongoDB**

- MongoDB gère la **replication asynchrone** au sein d'un *replica set* (RS)
- Le RS comprend un *maître* (*primary*) et des *esclaves* (*secondary*)
- Si nombre pair : ajout d'un processus supplémentaire (*arbiter*)
- Les écritures ont **toujours** lieu sur le *maître*
- Le *driver* associé à l'application peut lire sur le *maître* (cohérence forte) ou sur un *secondary* (cohérence faible, mais répartition)

# Election d'un *maître* dans MongoDB

Une élection est déclenchée dans les cas suivants :

- Initialisation d'un nouveau RS
- Un *esclave* perd le contact avec le *maître*
- Le *maître* perd son statut car il ne voit plus qu'une minorité de participants.

- Tous les clients (*drivers*) connectés au RS sont réinitialisés pour dialoguer avec le nouveau *maître*.
- Beaucoup d'options pour paramétrer le vote (p.e., une **priorité** donnée à chaque serveur).

# Cohérence et répartition des lectures

Par défaut, les clients lisent sur le *maître* : **cohérence forte**.

Une option de connexion permet à l'application d'indiquer qu'elle accepte de lire sur un *esclave* : **cohérence faible**

Autre option : l'application peut attendre que $n$ écritures soient faites avant de reprendre la main.

**Remarque**

La réplication n'est pas, dans MongoDB, le moyen privilégié de passer à l'échelle. Le **partitionnement** est un mécanisme plus puissant (à suivre !)

# A l'action ! Test de la réplication

On crée trois conteneurs avec des serveurs MongoDB

```
docker run --name mongo1 --net host mongo mongod --replSet mon-rs --port 300
docker run --name mongo2 --net host mongo mongod --replSet mon-rs --port 300
docker run --name mongo3 --net host mongo mongod --replSet mon-rs --port 300
```

Connectons un client au premier serveur (ou Studio3D si vous voulez).

```
mongo --host 192.168.99.100 --port 30001
```

Et initialisons le RS

```
rs.initiate()
```

On peut alors ajouter le second nœud (remplacer l'IP de la machine).

```
rs.add ("192.168.99.100:30002")
```

# Continuons

La fonction suivante indique le statut du nœud auquel on est connecté.

```
db.isMaster()
```

Information plus complète sur le RS avec :

```
rs.status()
```

Tout va bien ? Alors insérons.

```
use nfe204;
db.test.insert ({"produit": "Grulband", prix: 230, enStock: true})
```

On devrait trouver le document sur chaque *esclave* ?

```
mongo --host 192.168.99.100 --port 30002
use nfe204;
db.test.find ()
```

Hmmm que se passe-t-il... ? Et après `rs.slaveOk()` ? Pourquoi donc ?

# Et la reprise sur panne ?

Tuons (gentiment) notre *maître*.

```
db.shutdownServer()
```

Question : le second va-t-il s'élire *maître* ?

Relancer le premier serveur. Que se passe-t-il ?

**Remarque**

Un bon *driver* devrait enregistrer la liste des serveurs du RS, et se reconnecter automatiquement au *maître* quand celui-ci change.