

A satellite night view of North America, showing the United States and parts of Canada and Mexico. The landmasses are outlined by a dense network of yellow and white lights, representing city lights and urban areas. The background is a deep blue, representing the oceans. A semi-transparent white rectangular box is centered over the image, containing the text "Systèmes distribués".

Systèmes distribués

Objectifs

- Systèmes distribués
- Échanges synchrones / asynchrones
- Théorème de CAP
- Échange de données par fichiers



Système distribué

- Un **système distribué** est constitué d'un **ensemble** de (composants) **logiciels** localisés sur différentes ressources de calcul qui **communiquent** et se **coordonnent** en s'envoyant des **messages**
- On peut distribuer les systèmes de trois façons :
 - Un composant logiciel par serveur
 - Plusieurs instances sur des serveurs différents de chaque composants
 - Un mélange des deux derniers
- Les serveurs peuvent être répartis localement ou géographiquement dans un ou des centres de données à travers le monde

Exemple d'un système de type ERP



ERP : Enterprise Ressources Planning / PGI : Progiciel de Gestion Intégré

CRM : Customer Relationship Management

KPI : Key Performance Indicator /

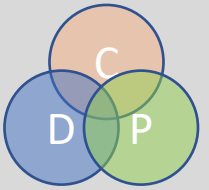
Avantages / inconvénients

- + Tolérance aux pannes : si un composant ne fonctionne plus ou est inaccessible, l'utilisateur a toujours du service
- + Mise à l'échelle horizontale infinie
- + Peut diminuer la latence (géographique) les utilisateurs des centres de données : répartition des données sur plusieurs continents
- Intégration des données et consistance difficiles
- Les messages peuvent ne pas être livrés : problèmes de réseaux ou de perte de messages
- Il est plus difficile de gérer des systèmes répartis plutôt qu'un système centralisé

Synchrone / Asynchrone

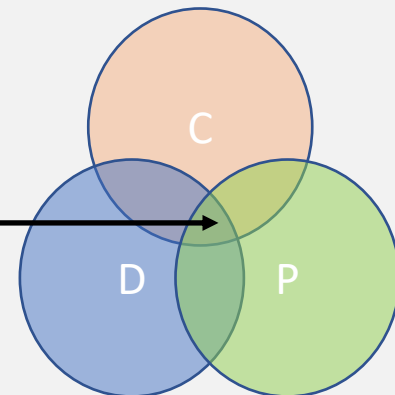
- Dans le mode **synchrone**, un traitement envoie un message et **attend qu'il soit traité** (avec ou sans réponse)
 - Appel d'une méthode
 - Appel d'un service web
 - Dans le mode **asynchrone**, un traitement envoie un message mais **n'attend pas qu'il soit traité** : le message sera traité à un moment donné
 - Envoi d'une lettre
 - Envoi d'un fichier de données
- => Dans le mode **asynchrone** le message peut être **avec ou sans assurance de prise en charge**

Théorème de CAP

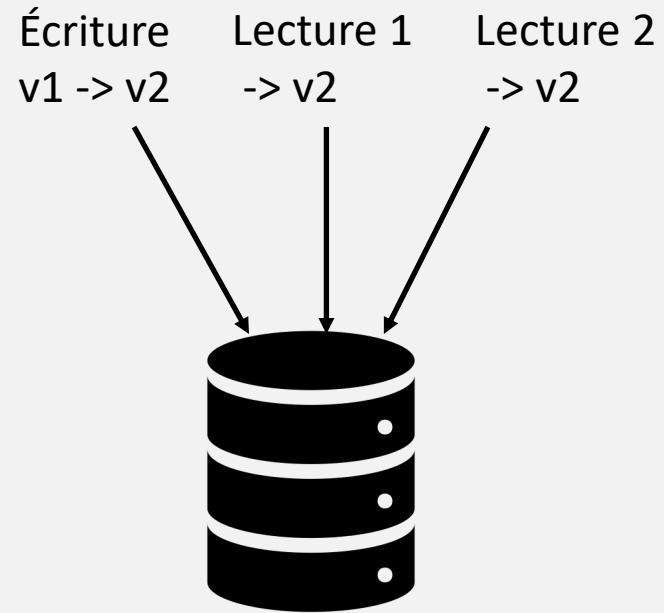


- Le théorème CAP ou CDP dit qu'il est impossible sur un système informatique de calcul distribué de garantir en même temps (c'est-à-dire de manière synchrone) les trois contraintes suivantes mais seulement deux :
 - **Cohérence** (Consistency) : tous les nœuds / systèmes ont les mêmes données au même moment
 - **Disponibilité** (Availability) : toutes les requêtes reçoivent une réponse
 - **Tolérance au partitionnement** (Partition Tolerance) : aucune panne ne doit empêcher le système de répondre correctement

CAP : impossible

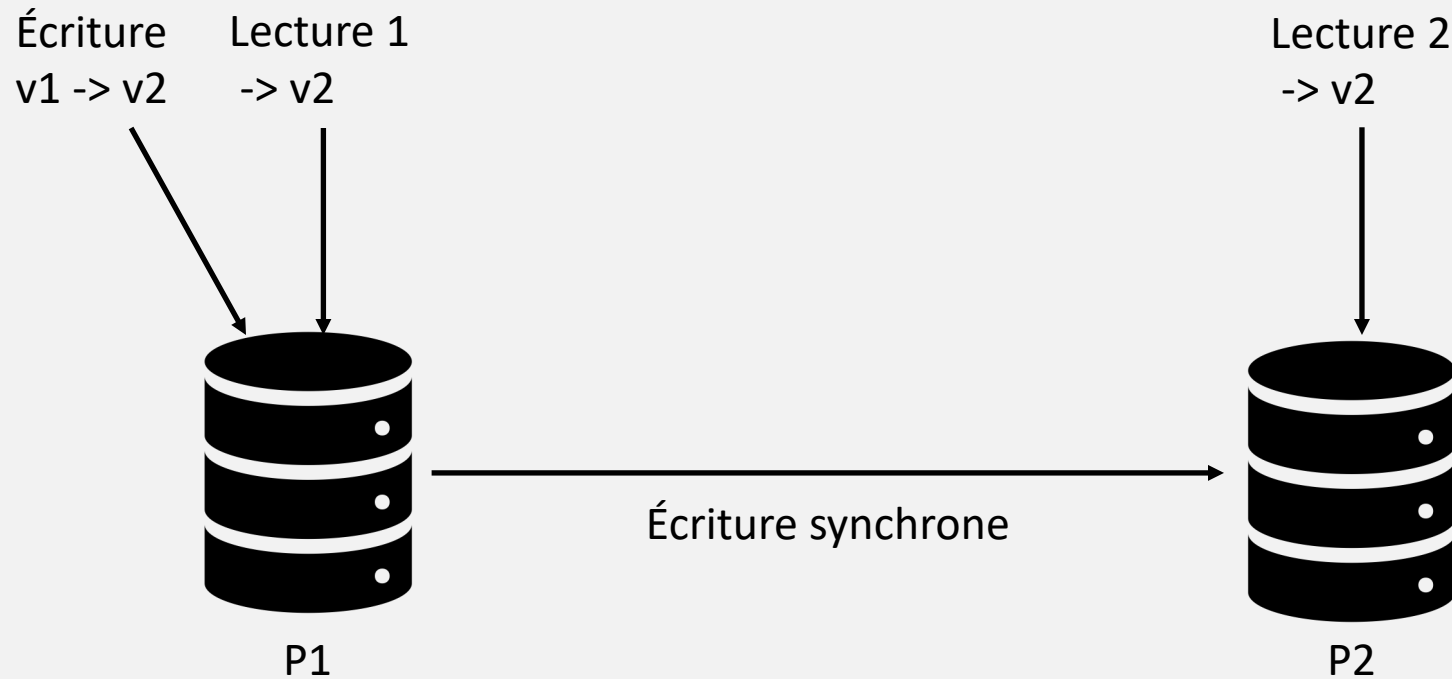


Cohérence et Disponibilité (CA)



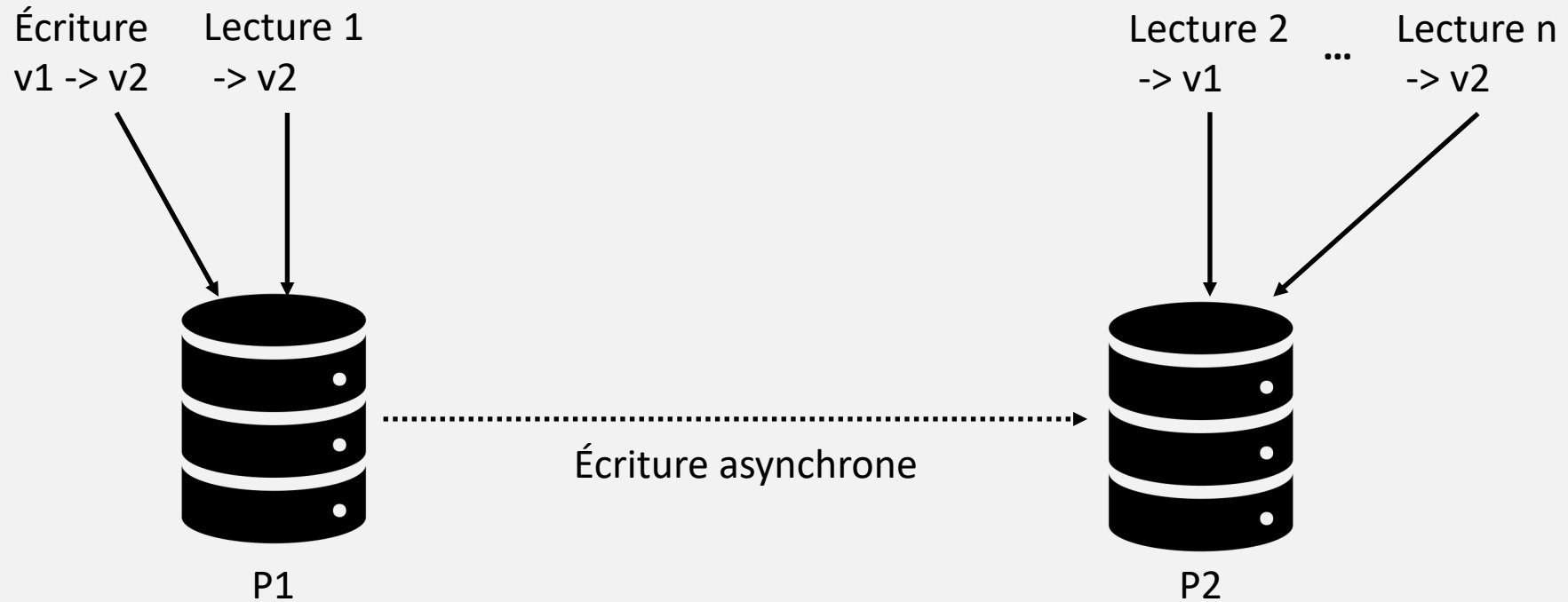
SGBDR : MySQL, Oracle, SQL Server, etc.

Cohérence et Tolérance au partitionnement (CP)



MongoDB, Hbase, etc.

Disponibilité et Tolérance au partitionnement (DP)



Cassandra, CouchDB, etc.

=> On parle de cohérence à terme

Échanges par fichiers

- EDI : Electronic Data Interchange - 1970
 - Échanges basés sur des fichiers
 - Transport : clef USB, (S)FTP(S), HTTP(S), etc.
 - Souvent des fichiers texte :
 - Champs déterminés par la position des caractères (ex. ACP-005)
 - Séparateur de champs : CSV
 - XML
 - JSON
 - ...
 - Si fichiers texte :
 - Attention au(x) caractères de retour de chariot (« \n », « \r », « \r\n »)
 - Encodage des caractères : EBCDIC, ASCII, AINSI, UTF-8, UTF-16, etc.

JSON : JavaScript Object Notation

- JSON = une valeur
- Une valeur :
 - Objet : {...}
 - Ensemble de clef / valeur
 - La clef est une chaîne
 - Collection : [...] de valeurs
 - Nombre
 - Chaîne de caractères
 - true / false / null
- Pour désérialiser les données en C# :
 - Une classe par type d'objet
 - Une propriété par couple clef/valeur

```
{  
  "nom": "Marie",  
  "prenom": "Vièrre",  
  "adresses": [  
    {  
      "numeroCivique": 123,  
      "odonyme": "Nicephore Niepce",  
      "typeVoie": "rue",  
      "ville": "Québec",  
      "province": "Québec",  
      "pays": "Canada"  
    }  
  ],  
  "estClient": true,  
  "magasinPrefere": null  
}
```

```
[  
  {  
    "ISO-3166-1" : "BZ",  
    "nom" : "Belize"  
  },  
  {  
    "ISO-3166-1" : "CA",  
    "nom" : "Canada"  
  },  
  {  
    "ISO-3166-1" : "CC",  
    "nom" : "Îles Cocos"  
  }  
]
```

Sérialisation / désérialisation

- Nous allons principalement utiliser deux méthodes statiques de la classe JsonConvert :
 - string SerializeObjet(object) : renvoie la représentation texte de l'objet « object » passé en paramètre
 - Type DeserializeObject<Type>(string) : interprète le texte passé en paramètre et renvoie l'objet désérialisé de type « Type »
- Newtonsoft.Json propose des attributs afin de modifier le nom des champs : <https://www.newtonsoft.com/json/help/html/SerializationAttributes.htm>

Références

- Exemple ERP :
 - SAP : [https://fr.wikipedia.org/wiki/SAP_\(progiciel\)](https://fr.wikipedia.org/wiki/SAP_(progiciel))
 - Sage X3 : https://fr.wikipedia.org/wiki/Sage_X3
 - Microsoft Dynamics 365 :
https://en.wikipedia.org/wiki/Microsoft_Dynamics_365
- ACP005 format : <https://www.rbcroyalbank.com/ach-fr/file-460199.pdf>