Utilisation de librairies pour l'implémentation TAP de VizieR

G.Landais (CDS) – G.Mantelet (Gavo)

- **TAPLib**
- **ADQLLib**
- **H3C** (PostgreSQL HEALPix indexation)

Besoins VizieR

- Volumétrie des grands catalogues nécessitant un bon index
- La traduction ADQL → SQL nécessite des méta-données supplémentaires :
  - la précision des données
  - la différenciation type → dbtype
  - Le système de coordonnées
- Validation du système de coordonnées
  - Alertes si le syst. de coord. ADQL est différent de celui de la table
  - (éventuellement) effectuer un changement de syst. de coord. (dans le cas d'un crossmatch)
- Utilisation des doubles quottes dans les noms de tables/colonnes
- Ajout de fonctions: round, cast, healpix
- Ajout de service : validation, search, query_plan, MOC

La librairie TAP

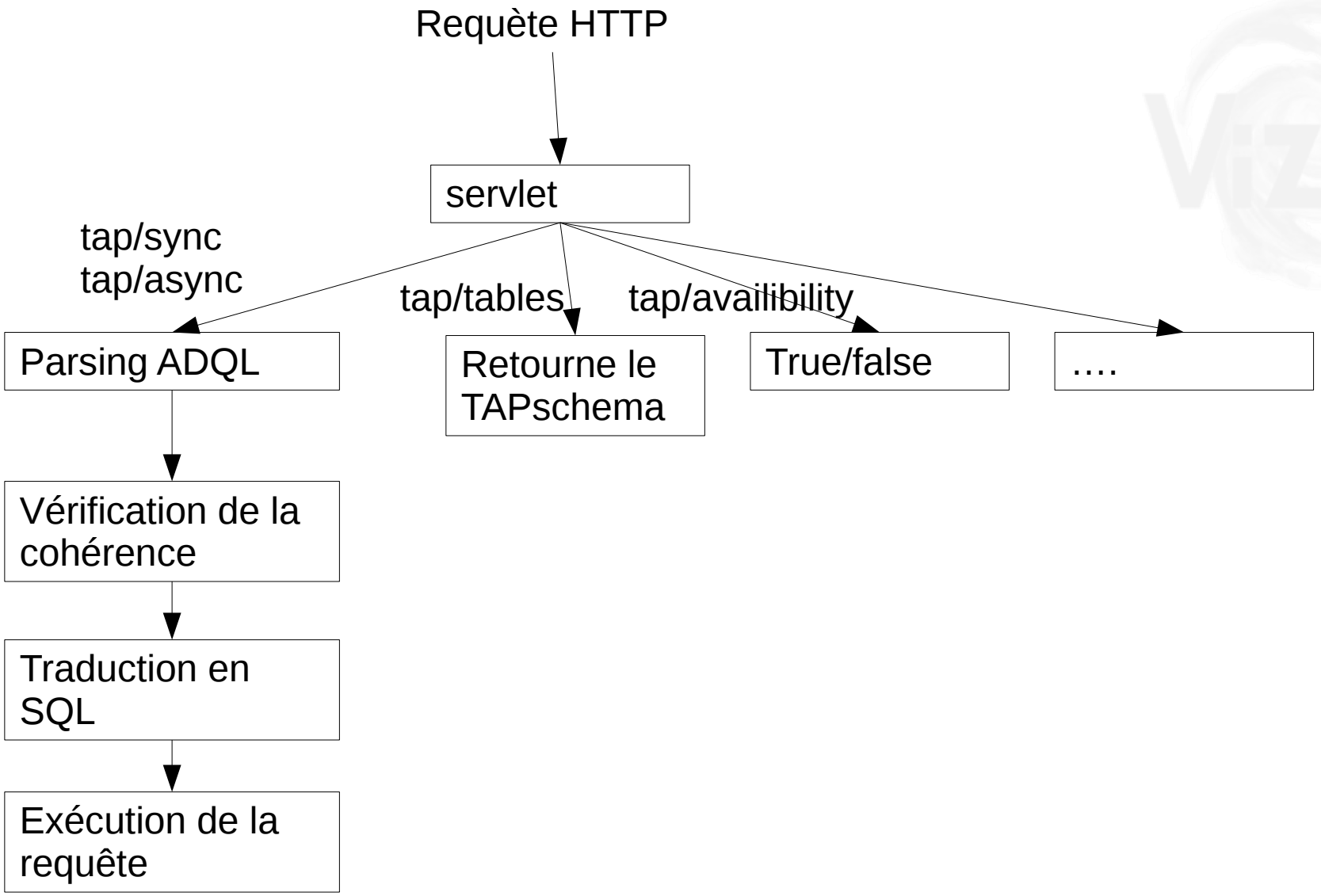
- Développées par Gregory Mantelet au CDS puis maintenue à Gavo (Heidelberg)
- La librairie **TAPlib** est composée des librairies
  - **UWSLib**: pour l'exécution de Jobs selon le protocole VO UWS
  - **ADQLLib**: la librairie de parsing ADQL comprenant la génération de script ADQL en SQL (ou autre langage assimilée)

Note : La librairie TAP contient les éléments ADQL+UWS.

Utilisation de la librairie TAP/ADQL

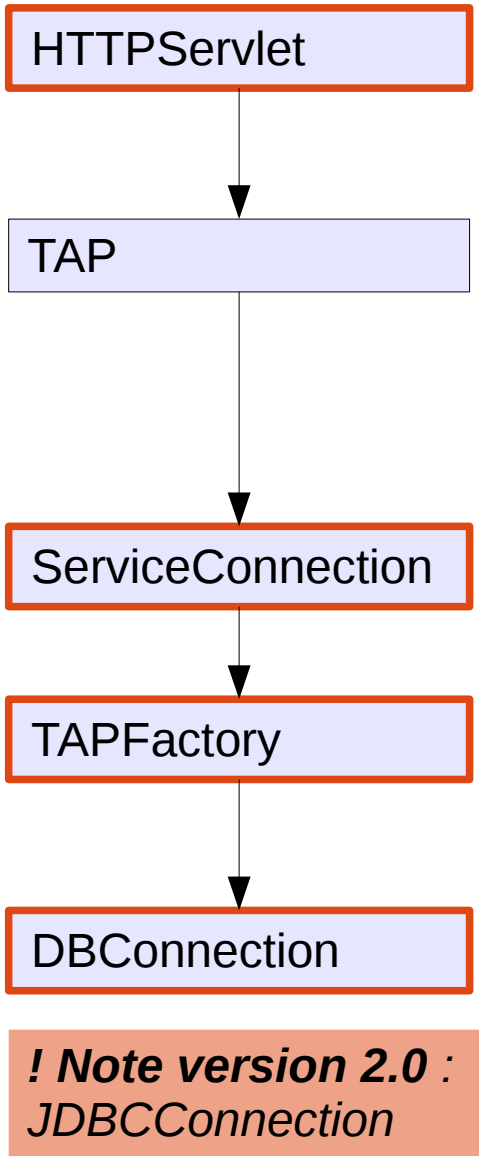
- CDS : VizieR, Simbad
- Saada
- TOPcat
- ???

URL: sources et documentations: http://cdsportal.u-strasbg.fr/taptuto/


Requète HTTP
servlet
tap/sync
tap/async
tap/tables
tap/availibility
Parsing ADQL
Retourne le TAPschema
True/false
….
Vérification de la cohérence
Traduction en SQL
Exécution de la requête

Implémentation de la librairie TAP

Contrôleur web du service
- méthode **init** initialise l'objet TAP.
- méthode **service** exécute le service TAP initialisé.



Initalisation du service TAP
- *ajout de ressources*
  (ex: un service search correspondant à l'URL /tap/search)
- Initialisation du “ServiceConnection”

Gestion des ressources:
- Configuration du service (temps d'exécution, taille des résultats, temps de rétention …)
- TAP_SCHEMA
- Types de sorties (VOTable, TSV, JSON)
- FileMananger (Irods, local)
- Identification
- Initialise la Factory , etc.

Factory TAP
- Choix du traducteur ADQL → SQL
- Initialise l'accès aux données
- Initialisation du backup manager (sauvegarde des jobs asynchrones)
- *(optionnel) Choix du QueryChecker pour vérifier la consistance des données*
- *(optionnel) Modification de la factory ADQL (ex: ajout de fonction utilisateur)*

Accès aux données (SGBD ou autre?)
- Open/close, Connect/disconnect
- Upload: transactions, delete/create/insert

... Classes à implémenter

La librairie ADQL

http://cds.u-strasbg.fr/resources/doku.php?id=adqllib

La librairie ADQL parse et traduit de l'ADQL en langage de requêtes.
Elle est composée par:

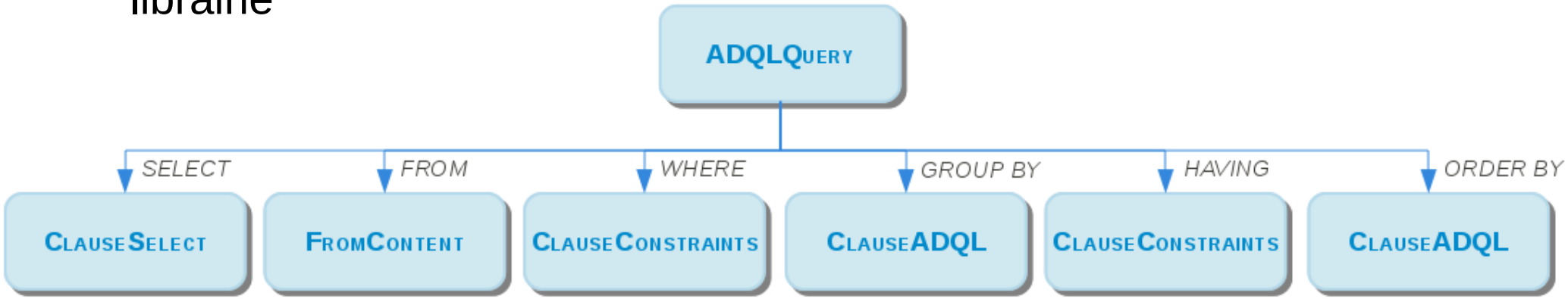
1) Vérification syntaxique avec le parser ADQL
   - L'arbre ADQL
   - *Possibilité d'ajouter de nouvelles fonctions*

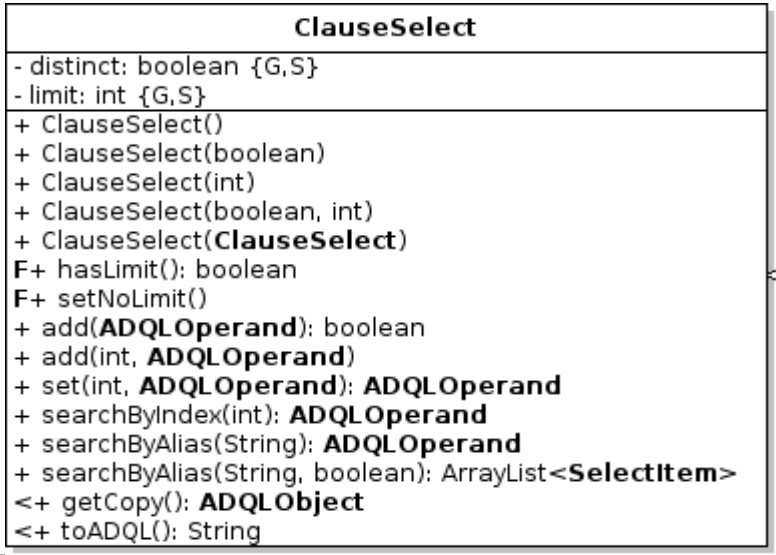
2) Vérification sémantique avec DBChecker
   (résolution des tables et colonnes de la requête ADQL + liens entre objets ADQL et méta-données)
   - Vérification de la consistance ADQL
   - (Ajout de méta-données)
   - *(modification de l'arbre avant la traduction SQL)*

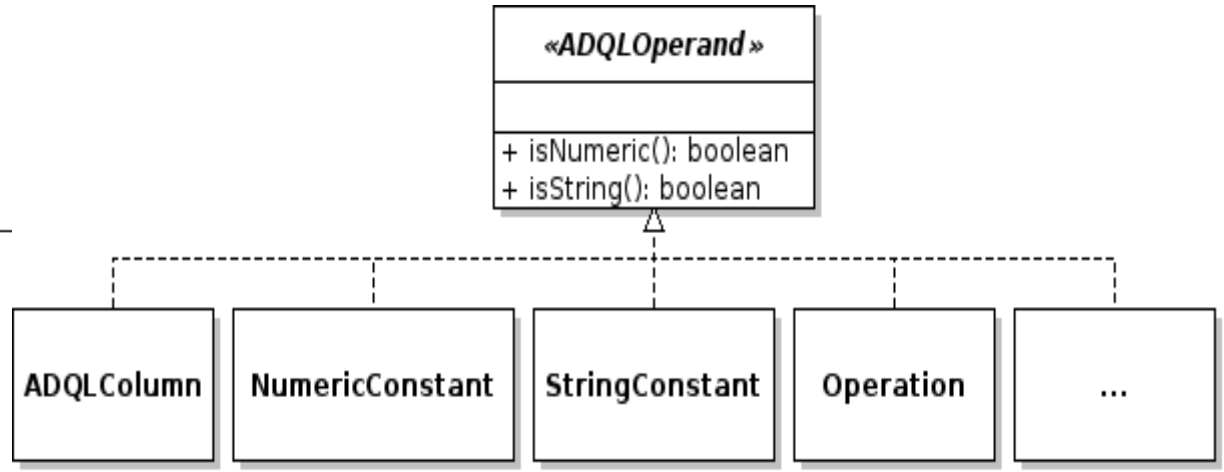
3) Traduction ADQL → SQL

Parsing ADQL (la classe “ADQLParser”)

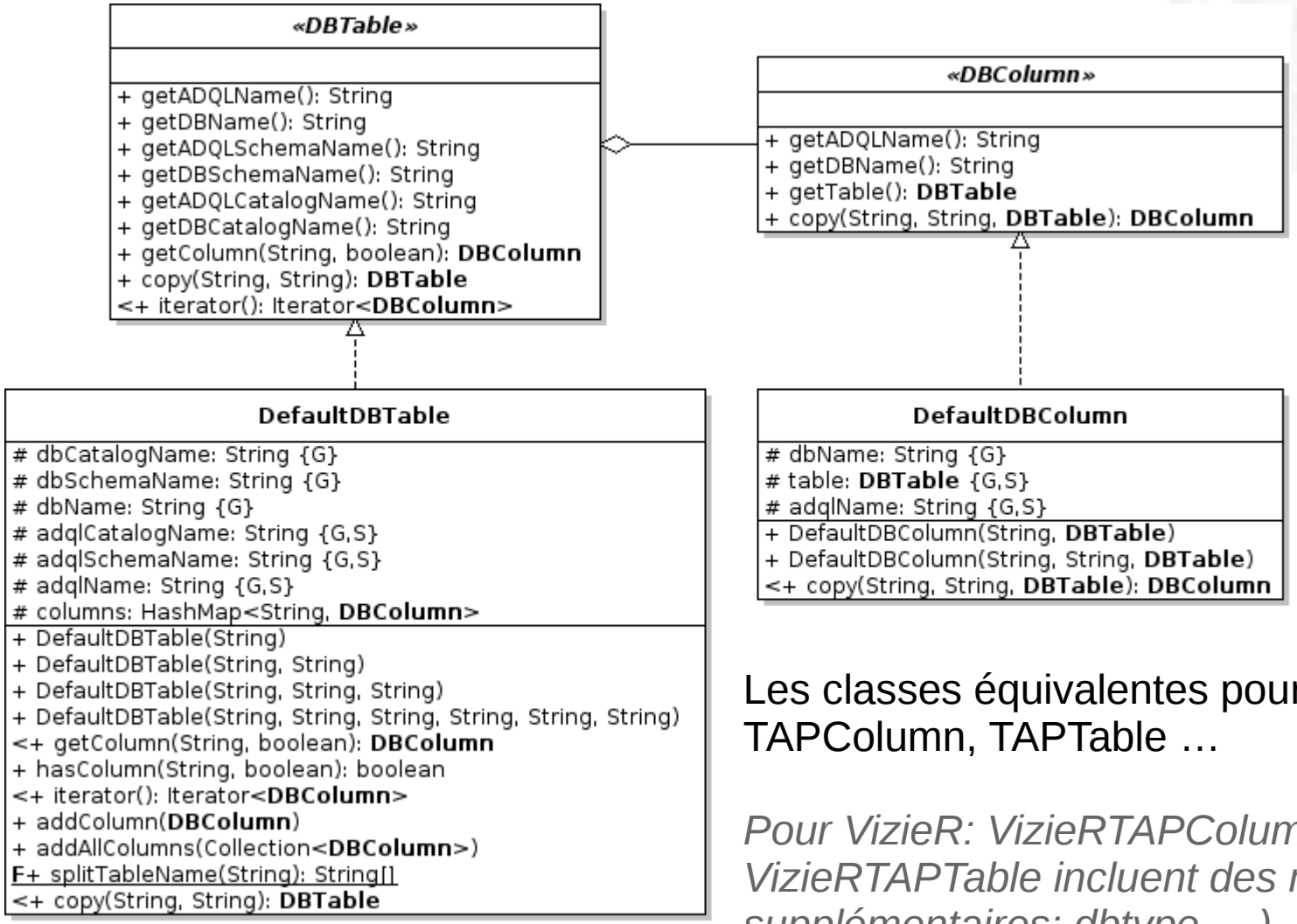
- La requête ADQL est stockée sous forme d'arbre pour pouvoir être manipulé ensuite.
- La librairie utilise JAVACC avec une grammaire ADQL fournie dans la librairie







DB consistency



Les classes équivalentes pour TAP: TAPColumn, TAPTable …

Pour VizieR: VizieRTAPColumn, VizieRTAPTable incluent des méta-données supplémentaires: dbtype, ...)

Traduction ADQL en langage de requêtes (interface “ADQLTranslator”)

- La librairie dispose d'une classe permettant de traduire du ADQL en SQL PgSphere
- Pour VizieR, la classe P3CTranslator pour pour H3C/Q3C

```
public class NewTranslator implements ADQLTranslator {
        ….
        public String translate(ADQLTable table) throws TranslationException{
                …
        }
        ….
        public String translate(CircleFunction circle) throws TranslationException{
                ….
        }
}
```

Parsing avec la classe « DBChecker » (interface QueryChecker)

La classe « DBChecker » vérifie la consistance des objets ADQL: tables, colonnes, système de coordonnées.
La classe à accès à l'arbre ADQL incluant les méta-données.

Il n'est pas nécessaire de l'étendre, cependant en la personnalisant, elle permet de manipuler l'arbre ADQL et d'y faire des modifications qui seront pris en compte lors de la traduction en SQL.

Personnalisation de « QueryChecker » pour modifier l'arbre ADQL

Les classes « SimpleSearchHandler » et « SimpleReplaceHandler » permettent de rechercher des nœuds et de modifier ceux-ci.

Exemple d'utilisation:

- Vérifier la cohérence des systèmes de coordonnées (notamment dans le cas d'un crossmatch)
- Modifier la précision ou gérer la correspondance des types des colonnes (type ↔ dbtype)

Une version 2.0 est en préparation (d'ici 2 mois).
Cette version ne sera pas rétro-compatible !

- Modification DBConnection
- Utilisation de STILTS pour les formats de sorties (VOTable en particulier)
- Configuration TAP dans un fichier de configuration
- Passage des tests TAP par STIL-taplint

H3C : librairie HEALPix pour PostgreSQL

http://cds.u-strasbg.fr/resources/doku.php?id=h3c

H3C : **H**EALPix **T**ree **C**

- Index fonctionnel btree PostgreSQL :

  CREATE index indexname **ON** table(healpix(ra,dec)) ;

  SELECT * **FROM** table **WHERE** healpix(ra,dec)=...

- H3C est basée sur Q3C (Koposov)
- Contient une librairie de fonctions qui utilisent l'index HEALPix.

  Conesearch, polygon, jointures....

Quelques fonctions utiles :

h3c_ang2ipix(ra, de [, nside]) RETURN the HELPix number from a position
h3c_ipix2ang(ipix [,nside]) RETURN the center of the HEALPix cell

h3c_radial_query(ra1, de1, ra0, de0, radius [,nside])
make a cone search arround position (ra0, de0)
h3c_join(ra1, de1, ra2, de2, radius [,nside])
join 2 tables according to their positions

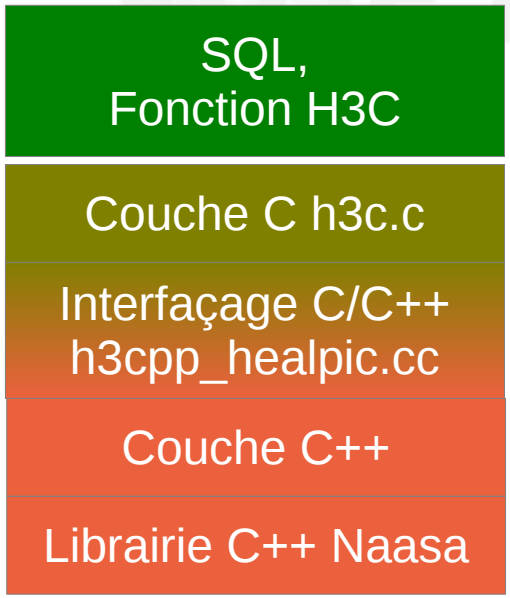
h3c_in_poly(ra, de, Array()) RETURN True if (ra,de) is in the polygon
h3c_poly_query(ra, de, Array() [, nside]) RETURN True if (ra,de) is in a polygon
(available for convex only)

h3c_in_ellipse(ra, de, ra0, de0, may_ax, axis_ratio, position_angle)
RETURN if (ra, de) is in the ellipse
the function doesn't use index

h3c_dist(ra1, de1, ra2, de2) RETURN the distance between 2 points

H3C utilise la librairie 64bits C++ de la nasa (Martin Reineke)

- Librairie C trop pauvre, librairie C++ moins riche que Java
- A nécéssité des modifications pour optimiser les recherches sur de petits cônes dans le cas de crossmatch



Contrainte PostgreSQL et tuning (décrit dans la doc)

1) l'ordre des tables dans la fonction h3c_join est important
2) Forcer l'utilisation de l'index
   - dans la session: set enable_seqscan=false
     set enable_nestloop=false
   - forcer l'utilisation de l'index pour la base de donnée "seq_page_cost" et "random_page_cost"
   - augmenter la mémoire cache
     - augmenter la mémoire cache utilisé par le plan de requete: effective_cache_size = 2048MB
     - augmenter la mémoire cache du dique : shared_buffers = 2048Mb

Postgresql.conf