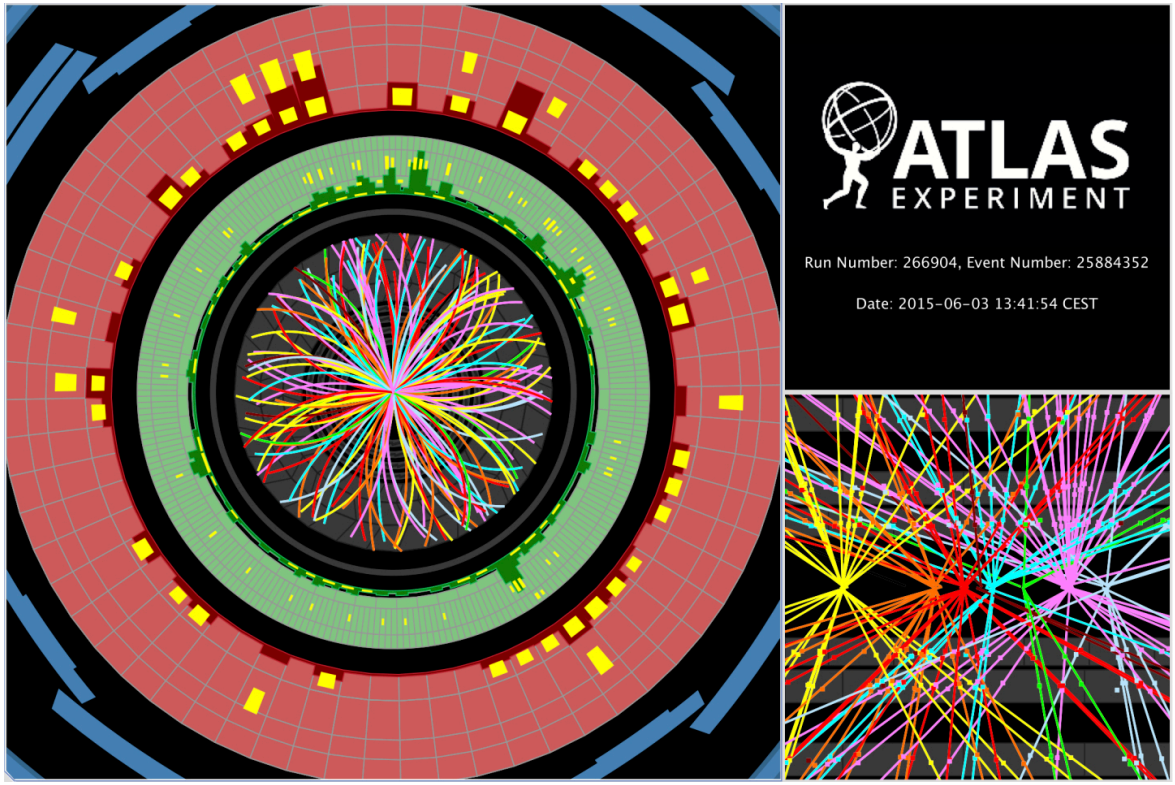
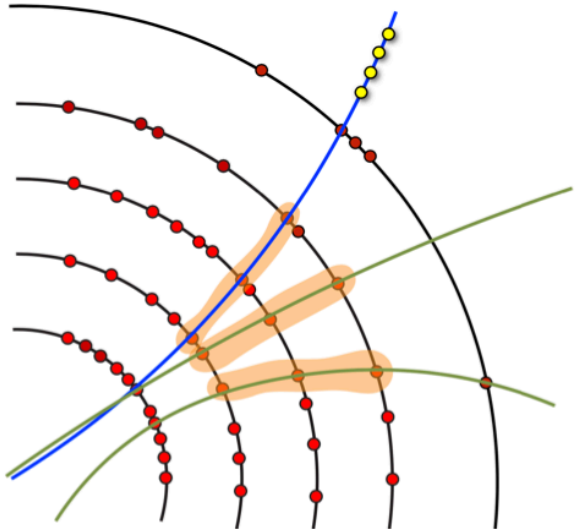
Optimisation automatique de la cartographie des matériaux pour Acts

Corentin Allaire

La trajectographie en physique des particules



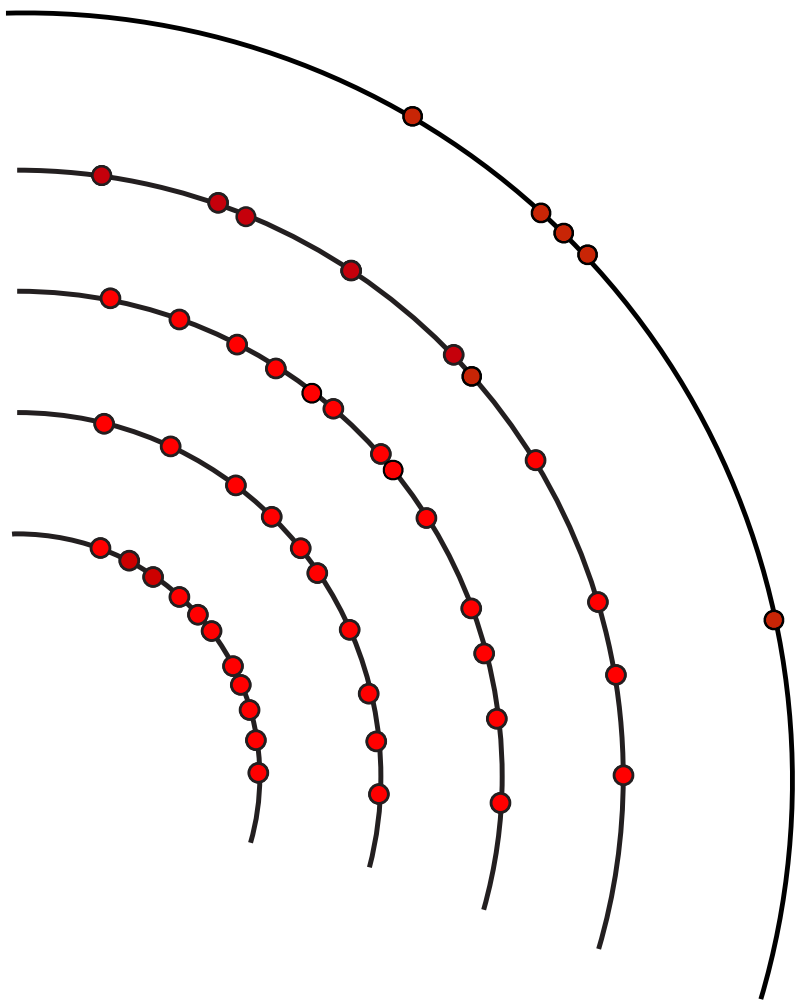
- Reconstruction des particules -> reconstruction de leurs trajectoires
- Quels dépôts d'énergie pour chaque particule
- Particule manquée = mauvaise performances de physique



- Efficacité attendue ~ 99%, faible taux de faux << 1%
- 1k particules par croisement de faisceau

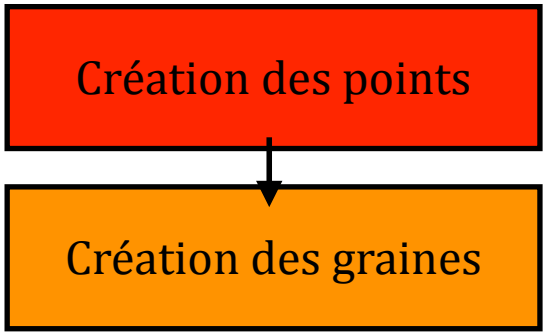
Reconstruction des trajectoires

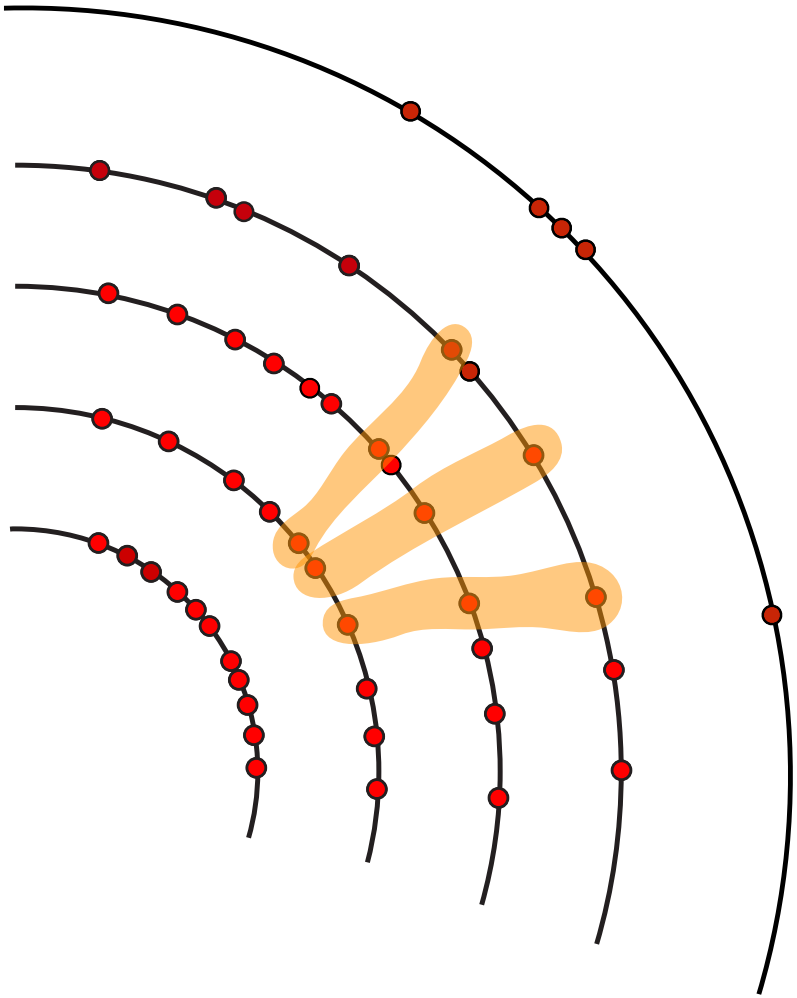
Création des points



- Coups dans le détecteur collectés pour créer des point de mesures
- Utilisés par nos algorithmes

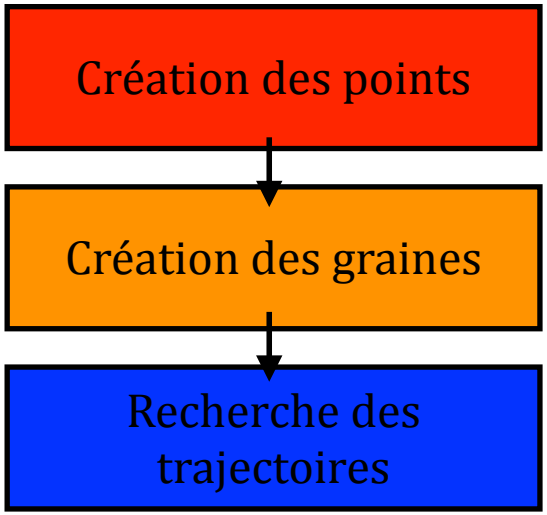
Reconstruction des trajectoires

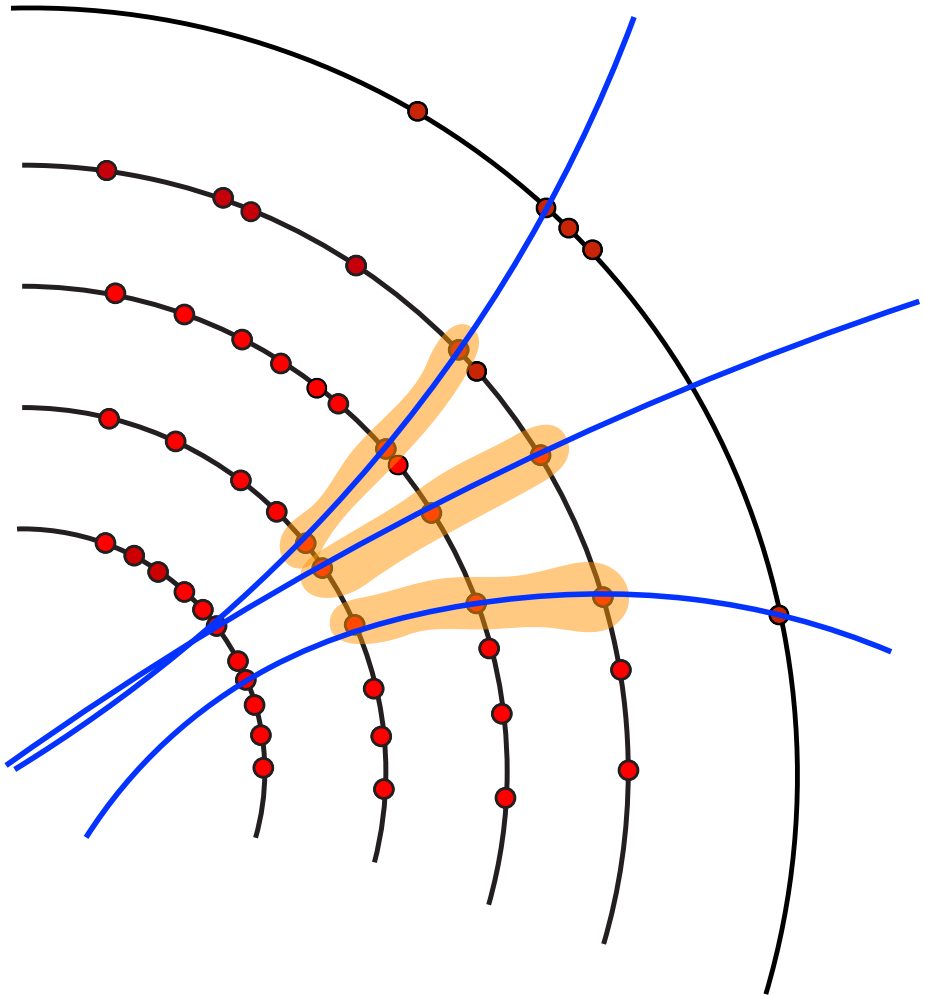




- Les Graines : groupes de 3 points compatibles avec l'hypothèse d'une trajectoire
- Étendues ensuite pour former nos trajectoires

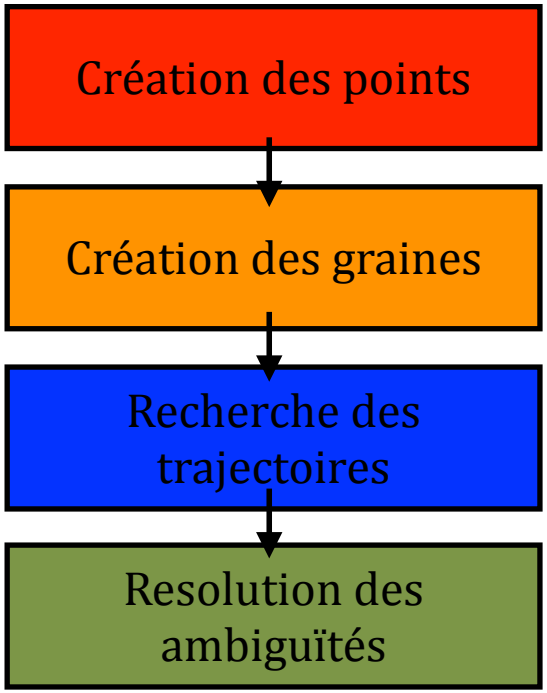
Reconstruction des trajectoires

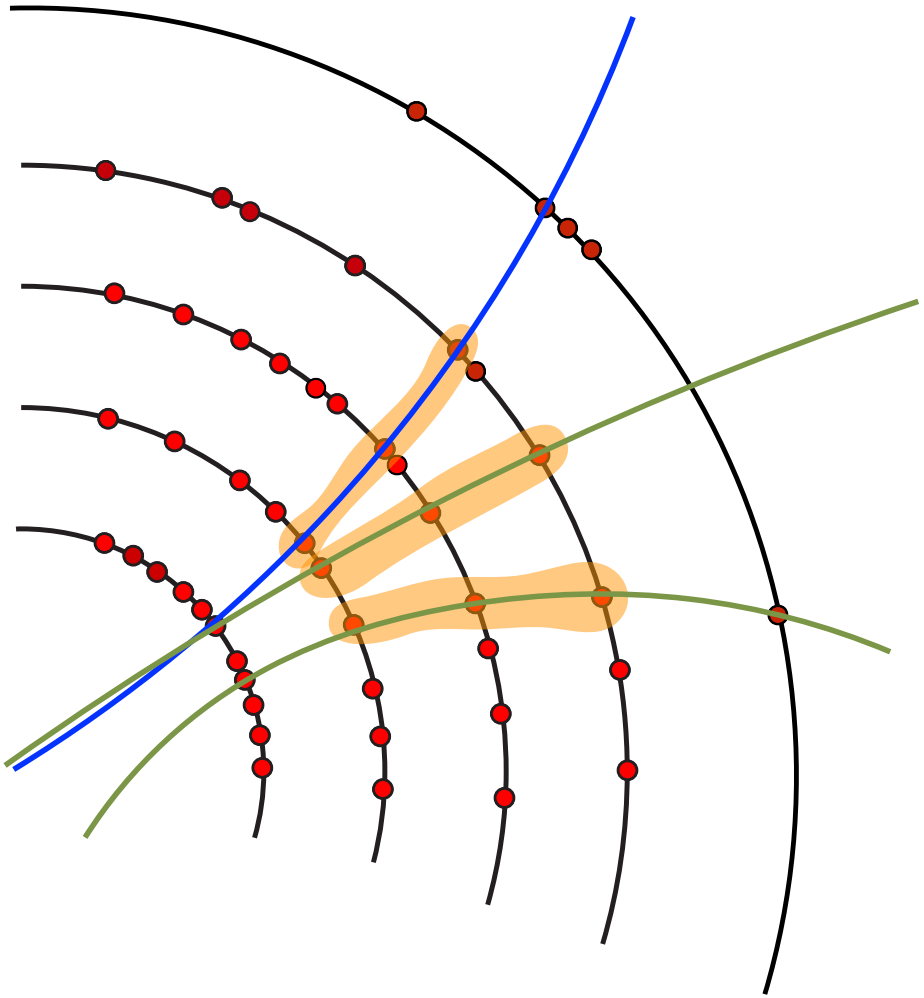




- Filtre de Kalman pour reconstruire les trajectoires
- Points de mesure compatibles ajoutés à notre graine de base
- Plusieurs hypothèses de trajectoires par graine si plusieurs chemins sont possibles

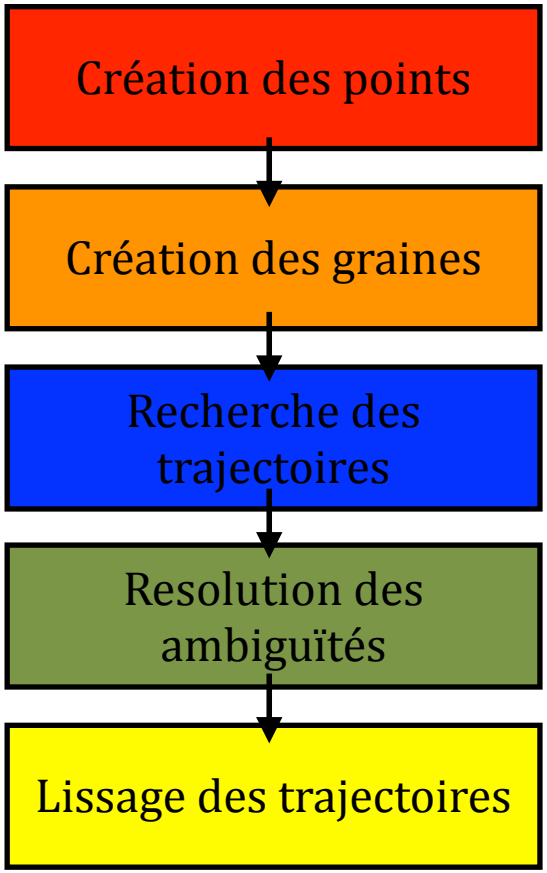
Reconstruction des trajectoires

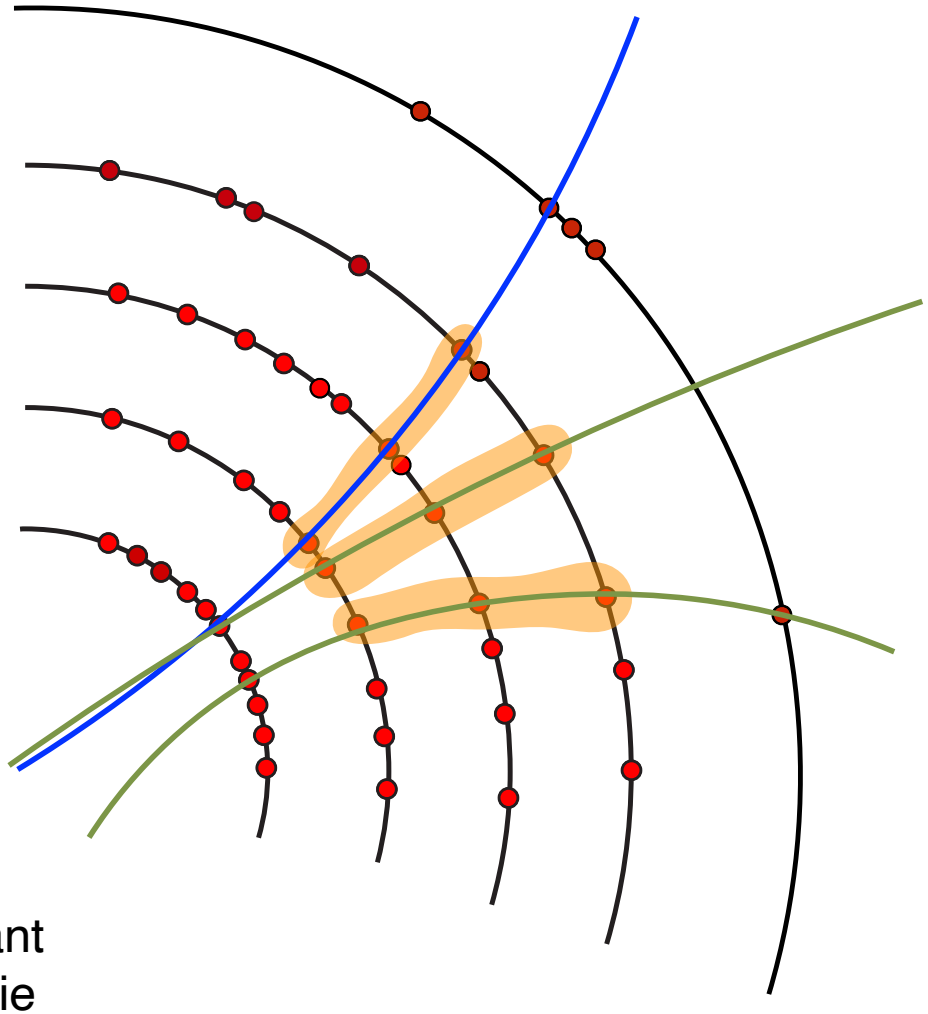




- Associe un score à chaque trajectoire
- Pour résoudre les ambiguïtés et retirer les faux, sélection des traces avec le meilleur score

Reconstruction des trajectoires

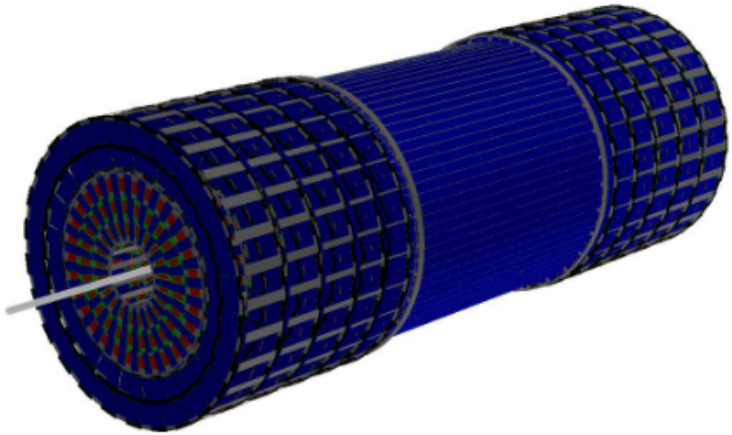




- Les trajectoires sont **lissées** en utilisant toutes les informations sur la géométrie du détecteur et son contenu en matériaux

Acts : A Common Tracking Software

- Acts fait partie de la nouvelle génération de logiciels de trajectographie :
  - C++ moderne (C++17)
  - Thread safe
  - Plus facile à maintenir (git, CI, …)
- Indépendant du détecteur, développé pour ATLAS au HL-LHC, mais d'autres utilisateurs : sPHENIX, EIC, FASER, ALICE, …
- Implémente un détecteur virtuel (Open Data Detector)
- Sert d'environnement de test pour de nouvelles méthodes de reconstruction (ML, …)



Optimisation automatique

- Algorithmes de trajectographie -> Grand nombre de paramètres libres servant à les adapter aux conditions expérimentales
- Optimisation -> plusieurs mois de travail pour les étudiants de la collaboration
- Automatisation de l'optimisation de ces algorithmes :
  - Plus rapide (si on a la puissance de calcul nécessaire)
  - Plus précise (plus grande fraction de l'espace des paramètres)
  - Plus sûre (minimise les erreurs humaines)
- Testé sur différents algorithmes :
  - Reconstruction des graines
  - Reconstruction des points d'interaction
  - Cartographie des matériaux

Oríon

- Oríon : framework python open source pour l'optimisation de fonction de type « boite noire » (https://orion.readthedocs.io/en/stable/index.html)
- Facile à intégrer à n'importe quel code python
- Executable en parallèle grâce à un système de base de données
- Compatible avec un grand nombre d'algorithmes d'optimisation :
  - Random search
  - Grid Search
  - Asynchronous Successive Halving (ASHA)
  - Tree-structured Parzen Estimator (TPE)
  - …
- Optimisation fonctionnant sans dérivation -> fonctionne très bien dans le cas de fonctions de coûts irrégulières et chers à évaluer

Oríon

Oríon implémentation : setup

```python
# Create the database
storage = {
    "database": {
        "name": "database_" + str(key),
        "type": "pickleddb",
        "host": os.path.join(pathDB, "database_" + str(key) + ".pkl"),
        "timeout": 43200,
    },
}
```

- Création d'une base de donnée pour notre optimisation
- Supporte plusieurs DB : pickledDB, MangoDB, …
- Utilisées pour synchroniser les différents essais

- Définition de notre espace des paramètres
- Construit une « expérience »
  - Espace des paramètres
  - Stockage
  - Algorithme d'optimisation (ici recherche aléatoire)

```python
# Create the search space, the range of the binning can be chosen here
# x represent X or phi depending on the type of surface
# y represent Y, R or Z depending on the type of surface
space = {
    "x": "uniform(1, 240, discrete=True)",
    "y": "uniform(1, 240, discrete=True)",
}
# Build the experiment
experiments = build_experiment(
    "s_" + str(key),
    version="1",
    space=space,
    algorithms="random",
    storage=storage,
    max_idle_time=43200,
)
```

Oríon implémentation : optimisation

```
trials = experiments.suggest()
parameters = (trials[job].params, trials.params["y"])
score = Algorithm(parameters)
```

- suggest() : l'expérience nous suggère des paramètres à essayer
- Utilisés comme entrée de l'algorithme à optimiser -> détermination du score

- L'expérience observe ce score -> écriture dans le stockage

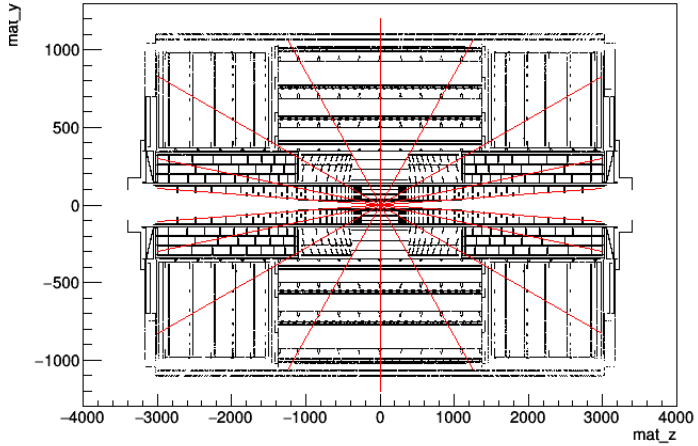
```
experiments.observe(trials, score)
```

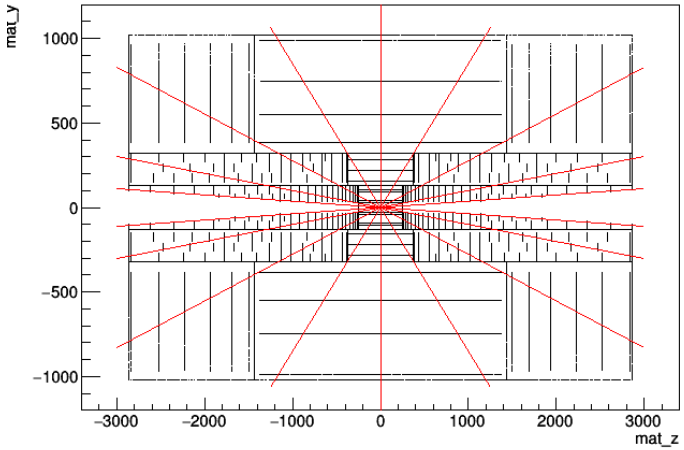
Parallélisme ?

- Grace au système de base de données, suggest() peut être appelée N fois en parallèle
- On exécute ensuite notre algorithme en parallèle N fois et on observe chaque résultat
- Légère perte de prédictibilité de la suggestion, mais grand gain de vitesse

Cartographie des matériaux

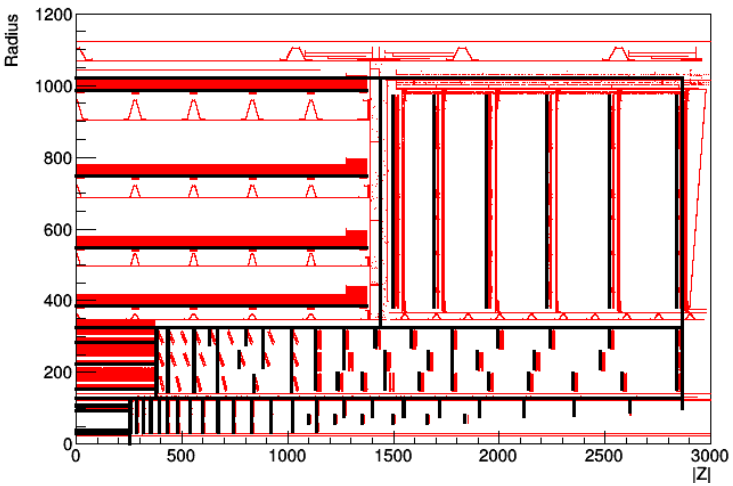
- Simulation Geant4 trop complexe -> inutilisable lors de la reconstruction
- Création de géométrie simplifiée : Tracking Geometry -> sert à naviguer à travers le détecteur
- Les matériaux du détecteur doivent être pris en compte lors du processus de lissage des trajectoires
- Modèle de matériaux simplifié -> Cartes de matériaux

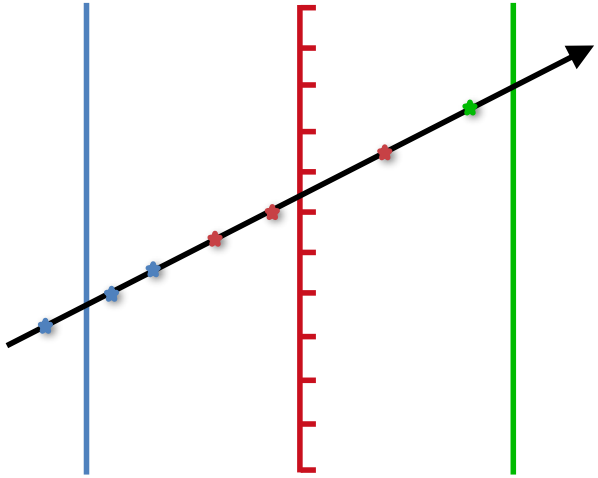




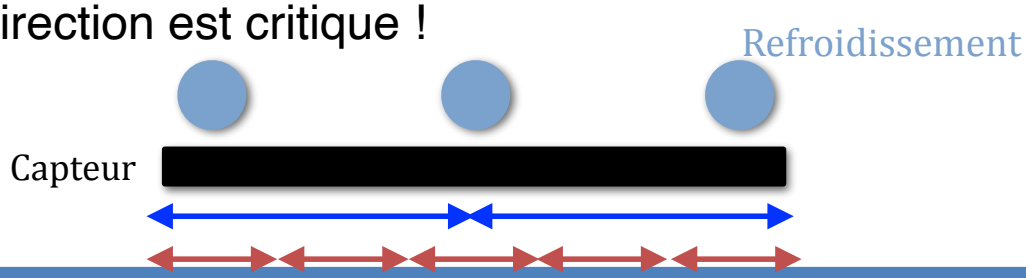
Cartographie des matériaux

- Sélection des surfaces sur lesquels les matériaux seront projetés
- Matériaux collecté en scannant le détecteur avec G4
- Chaque matériau est associé à la surface la plus proche



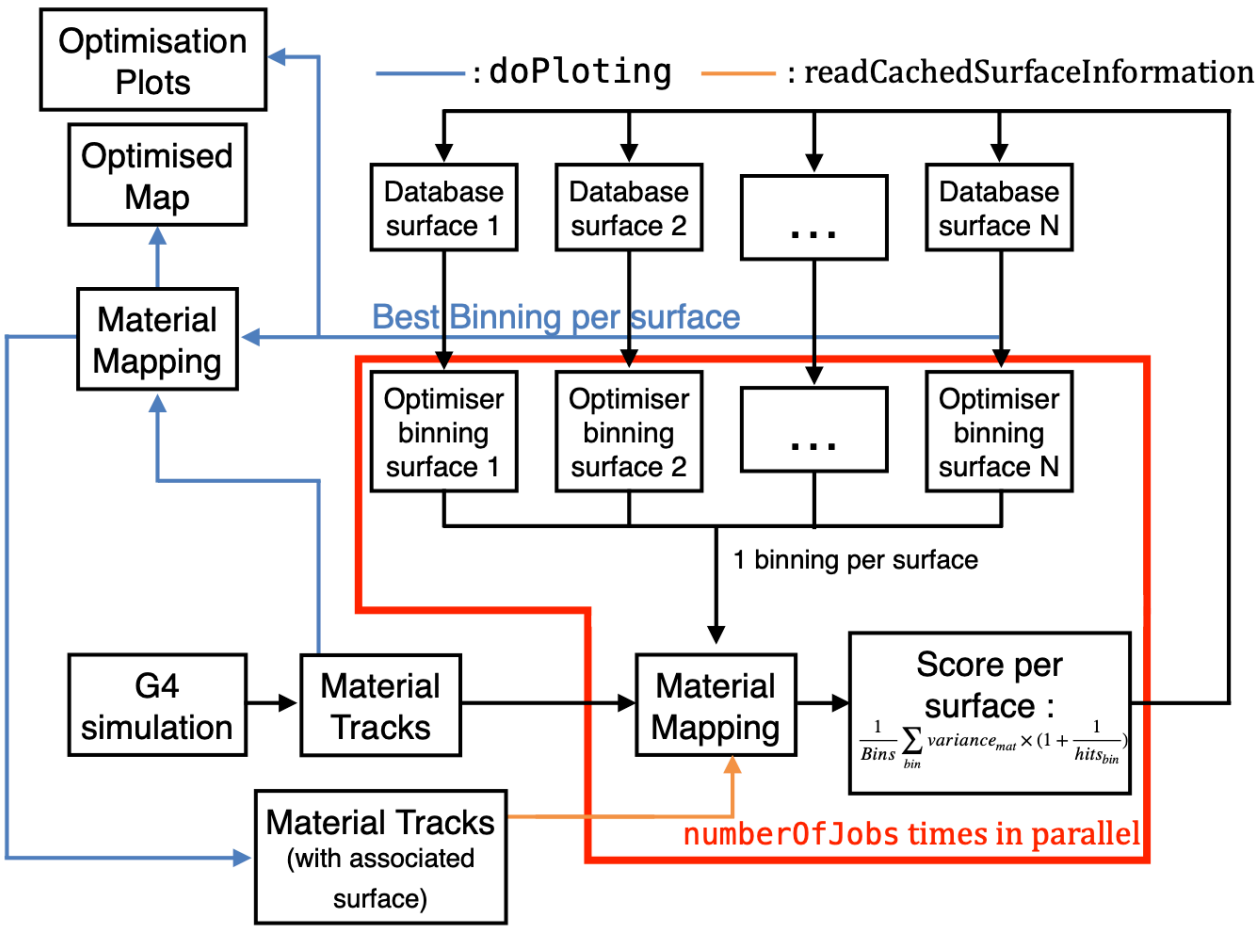


- Chaque surface est divisée selon deux directions (R, Phi), (Z, phi), …
- Les matériaux sont accumulés puis moyennés pour former notre carte
- Déterminer le nombre d'intervalles dans chaque direction est critique !



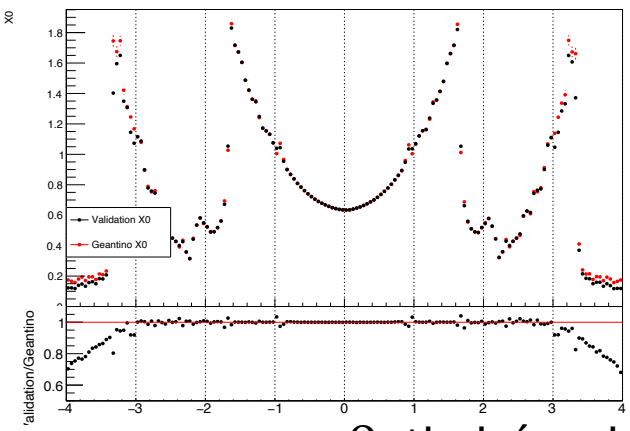
Optimisation automatique de la Cartographie

- Une instance d'Oríon par surface
- Suggestions combinées pour une cartographie
- N cartographies en parallèle
- Après N cartographie -> mise à jour des instances grâce aux scores
- Combinaison des meilleurs scores de chaque surface -> cartographie finale

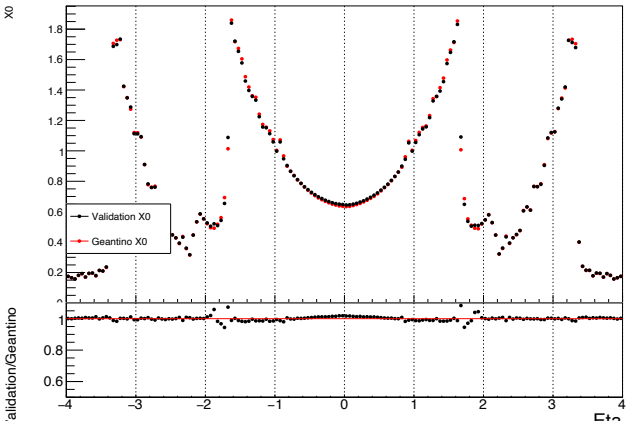


Résultat

- La cartographie testée sur l'ODD manuellement
- Optimisation manuelle : Une semaine de travail
- Utilisation d'Orion pour déterminer le découpage idéal de chaque surface
- Score : $\frac{1}{Bins}\sum_{bin} variance_{mat} \times (1 + \frac{\sqrt{Nbins}}{200})$
- Tests de différents scores en cours
- Recherche aléatoire (lent), 1 journée d'optimisation sur 40 cœurs (~2000 configurations testées)
- Bon accord entre les deux cartes
- Implémentée dans Acts -> Utilisable par n'importe quelle expérience



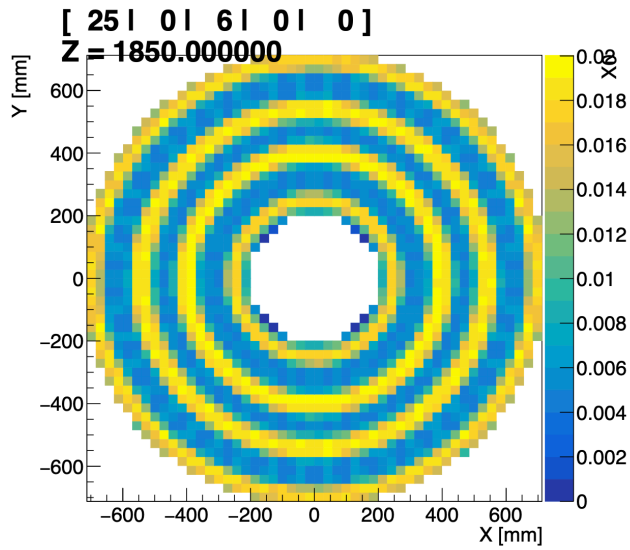
Optimisé main

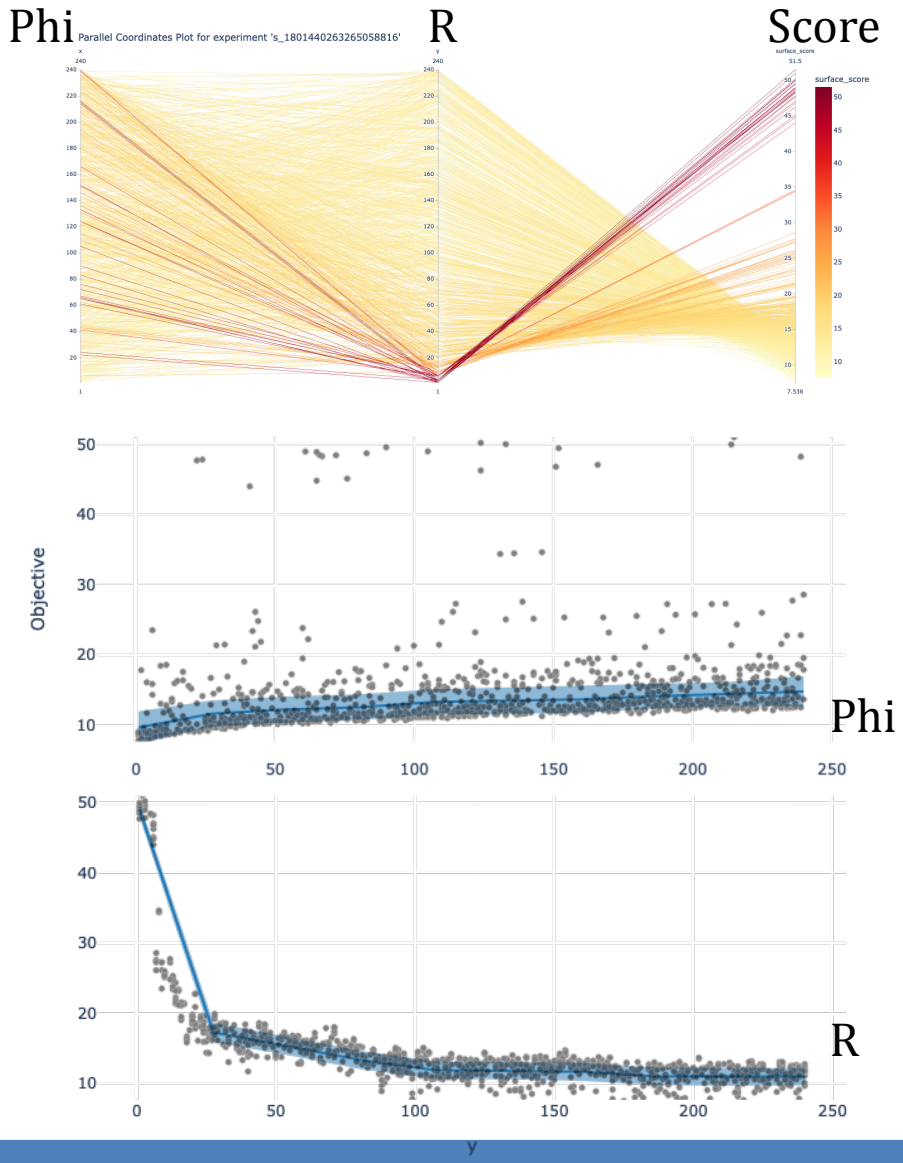


Optimisé automatiquement

Résultat

- Oríon -> grand nombre de figures de validation
- Segmentation pour une des surfaces des bouchons de notre détecteur
- Pas de dependence en Phi -> 1 segmentation
- Forte dépendance en R -> beaucoup de segmentation





Conclusion

- Les algorithmes de trajectographie ont de nombreux paramètres libres qui doivent être optimisés
- Mise en place d'une optimisation automatique -> Quels sont les meilleurs paramètres pour notre expérience / nos conditions expérimentales
- Présenté ici pour la cartographie des matériaux, mais aussi implémenté pour d'autres algorithmes de reconstruction d'ACTS

Et maintenant ?

- Mise en place d'un système général d'optimisation des paramètres pour ACTS
- Jusqu'où peut on pousser cette optimisation ?

BACKUP