REX
Introduire du test dans du code legacy, jusqu'où aller ?

Mettre en oeuvre une démarche de test en développement logiciel.
9-12 décembre Ecully.

Olivier Inizan
oinizan@versailles.inra.fr
@OlivierInizan

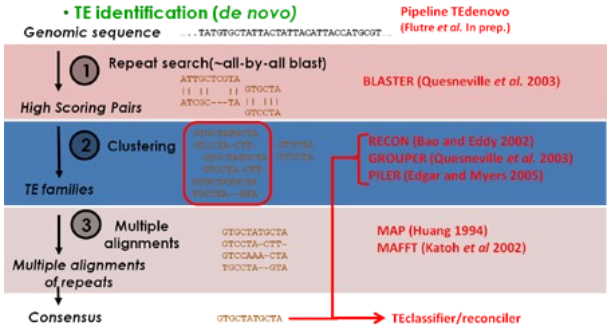
L'unité

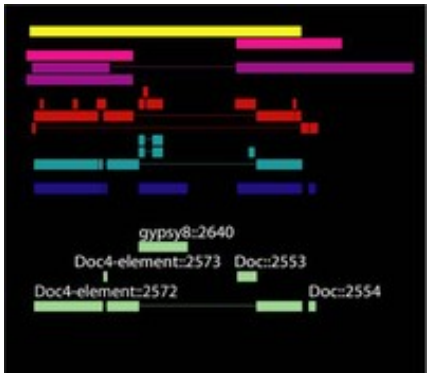
- URGI: Unité de Recherche en Génomique Info
- 2 activités: Plateforme + Recherche
- 2 équipes, =~ 30 ingénieurs
- Equipe Anagen (analyse de génomes)

Equipe dev anagen

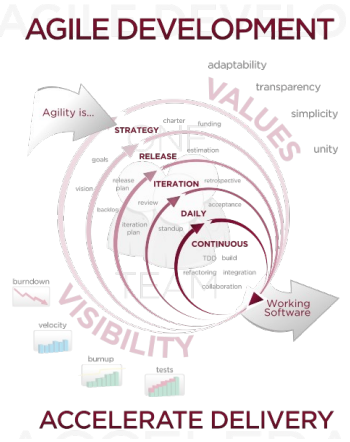
Software developpers team:

(i) working on TEs and mobile elements detection + workflows NGS





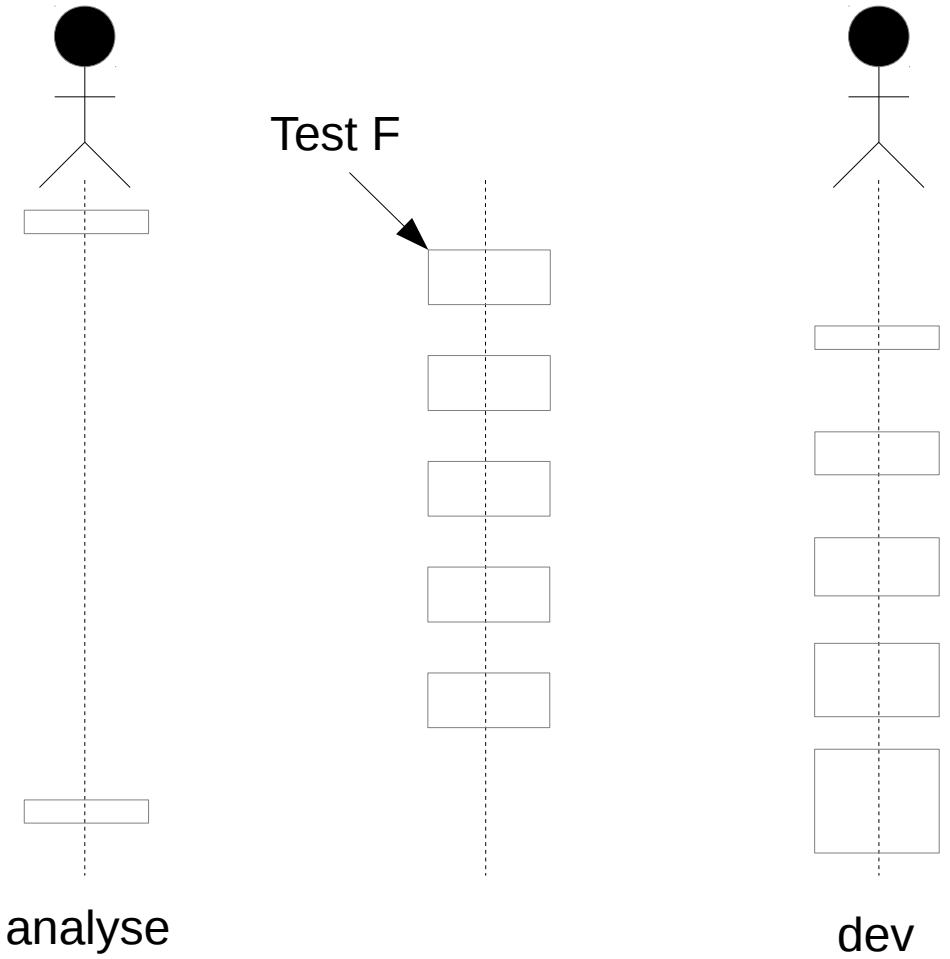
(ii) and organize itself according Agile :



for project managment

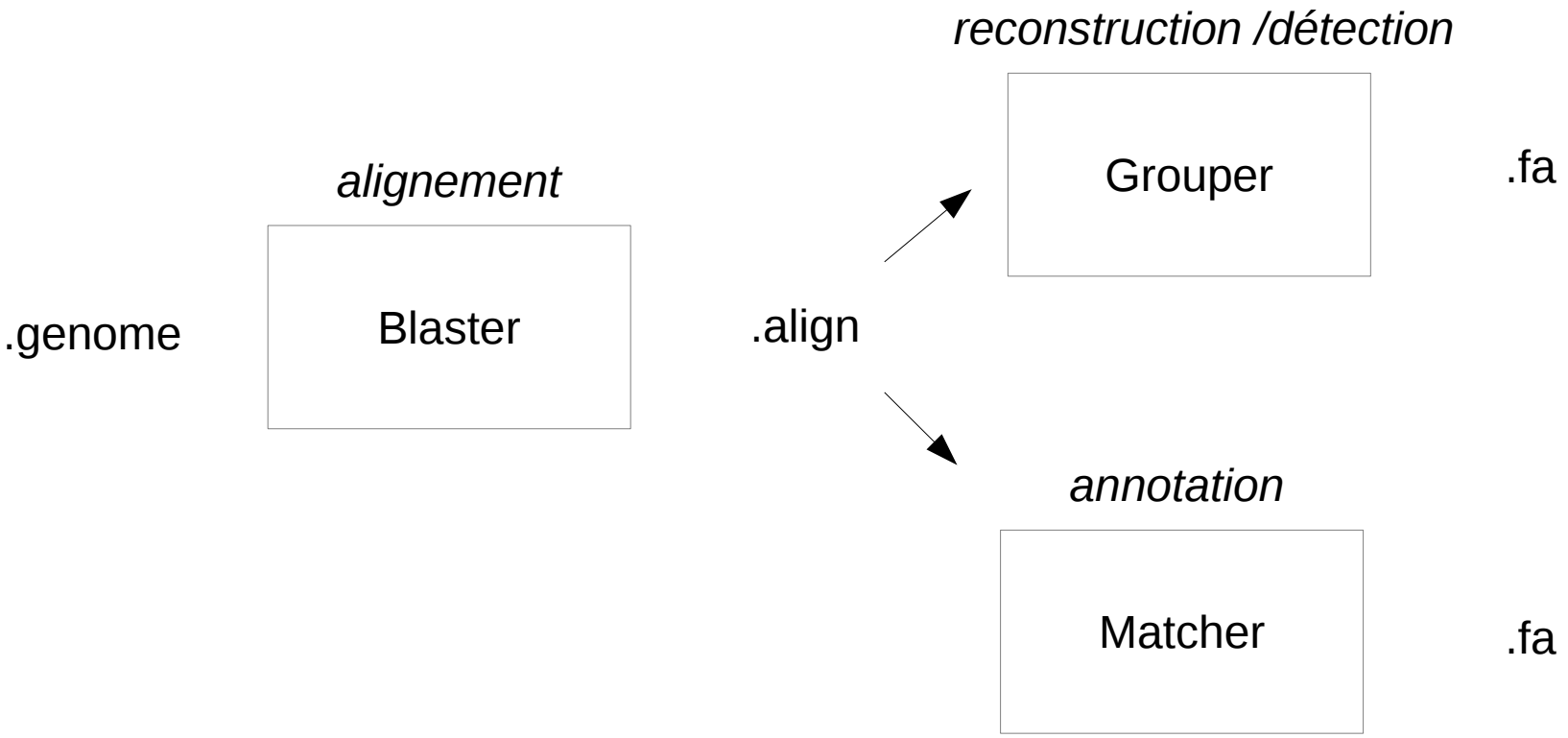
for code production

30 > 15 > 6 > 2

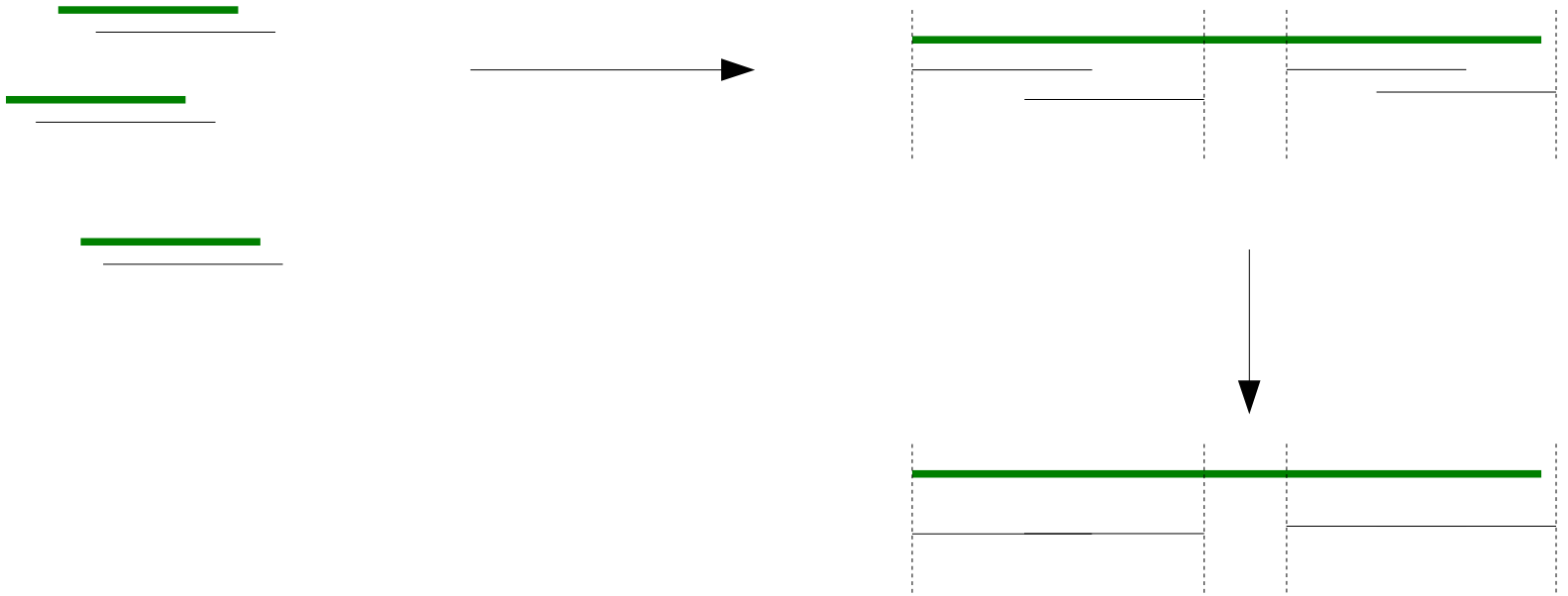


Le code

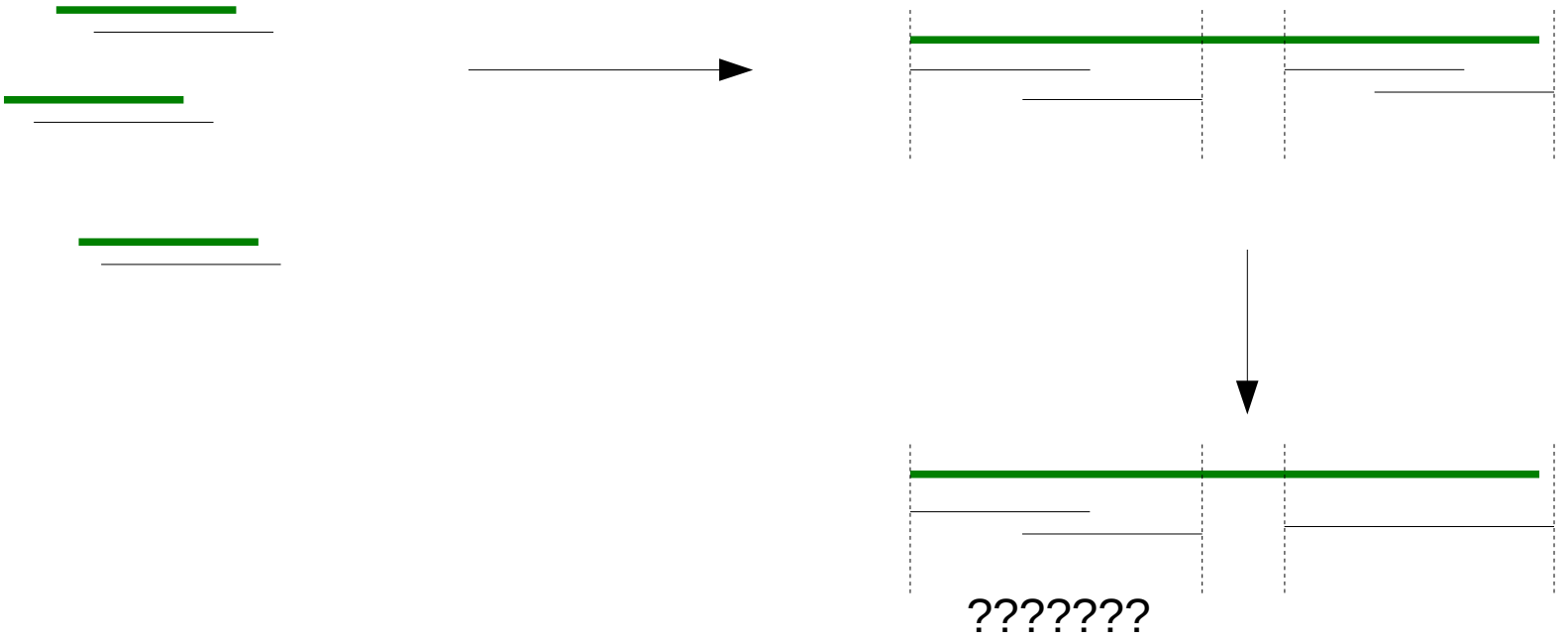
- Détection/Annotation d'éléments répétés



Matcher

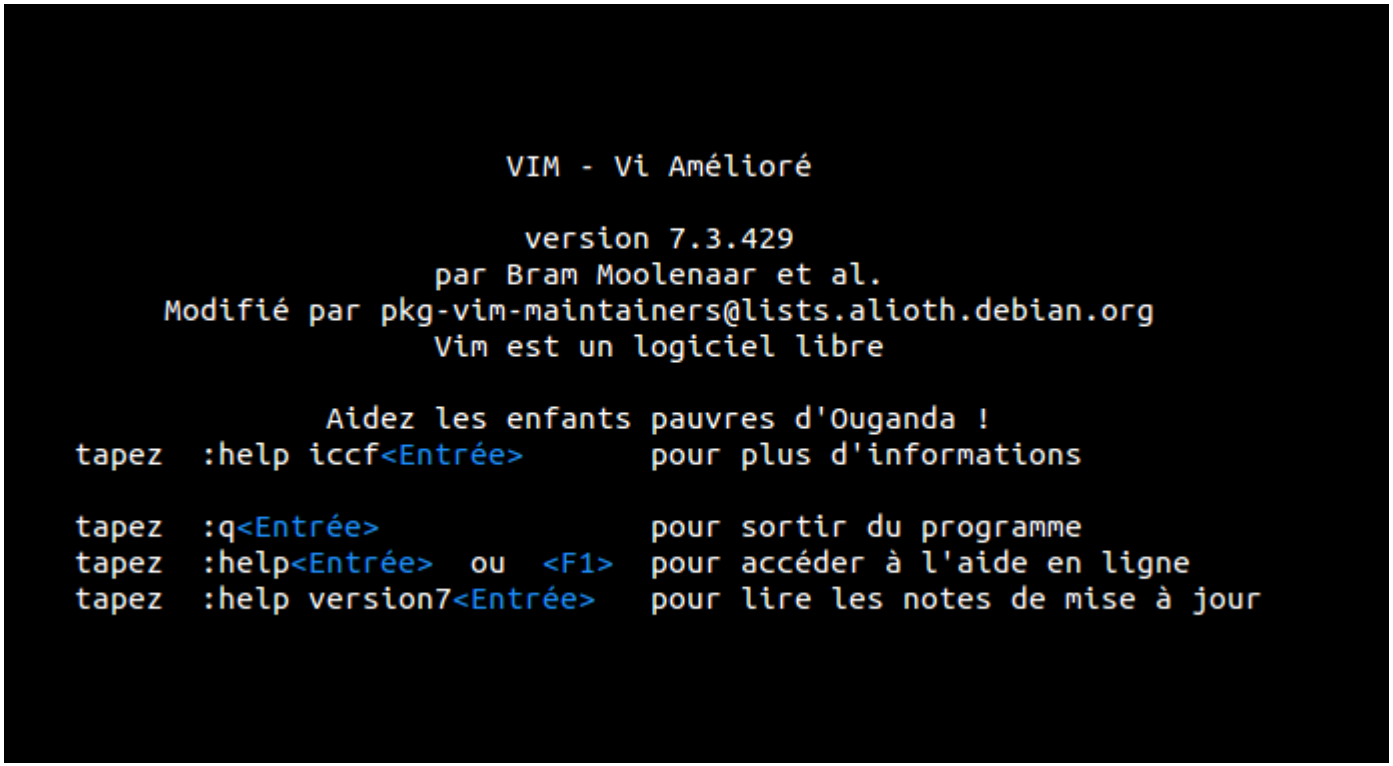


Matcher: le bug



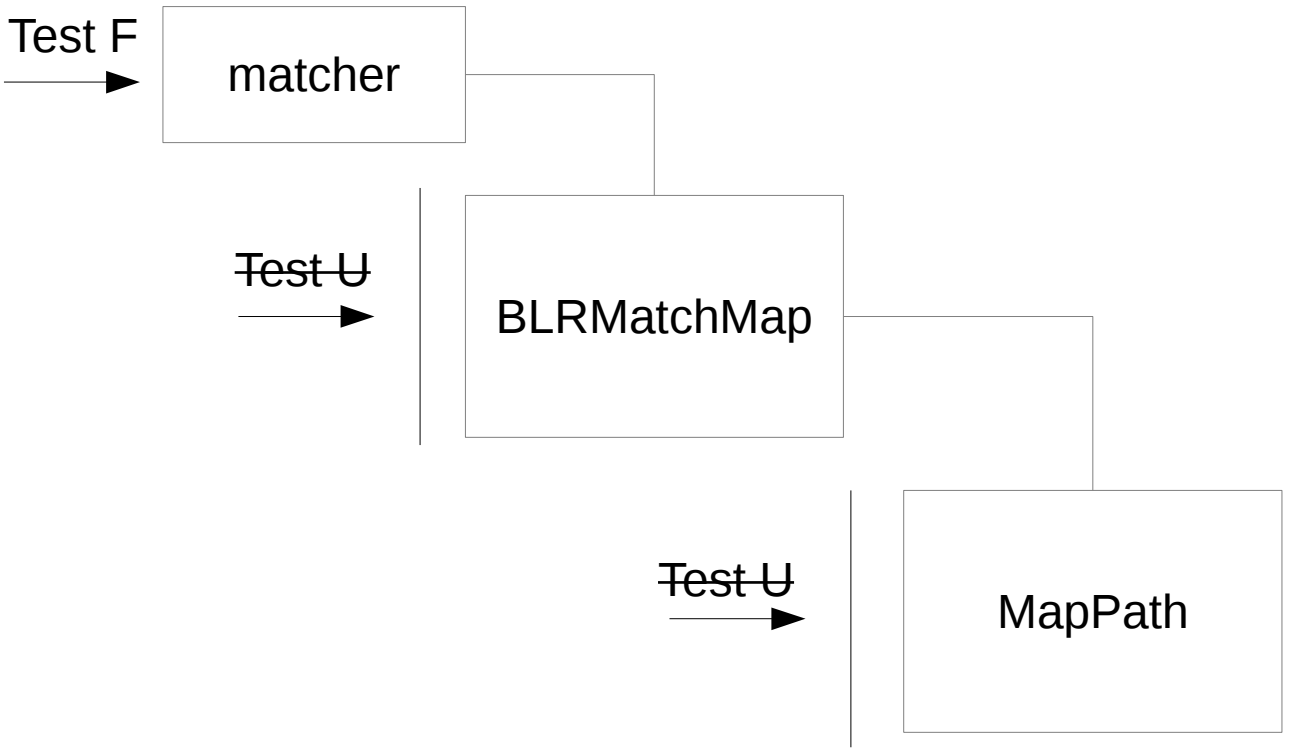
Equipement

- C++ (newbie)
- CPPUNIT
- vi
- make test



La démarche

- Quels tests adopter ?



Typologie: Test Fonctionnel

- Code == Boîte noire

- Obtenir un output de réference
- Vérifier que le code le reproduit

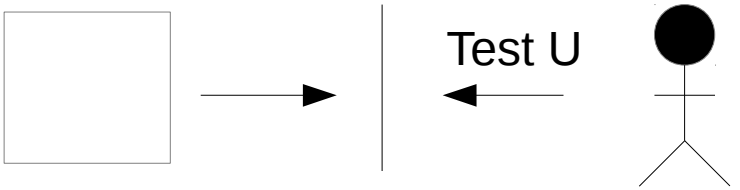
Tests F: quel intérêt ?

- Figer dans du code la ligne de cmd
- Execution routinière des tests
- Assure la non regression

```
oinizan@alaska:~/workspace/repet_pipe/TE_finder/matcher$ ./matcher2.25
-i input_test_runAsScript_bigData.align -j -X
```

Typologie: ~~Tests U~~

- La classe présente une interface (public)
- Les méthodes de l'interface sont testées
- Fournir un input
- Tester l'output et/ou l'attribut



~~Tests U~~: quel intérêt ?

- Assure la non regression
- Documentation: pour comprendre et documenter une méthode, écrire son test
- Seuls les tests sont source de doc, pas de source supplémentaire

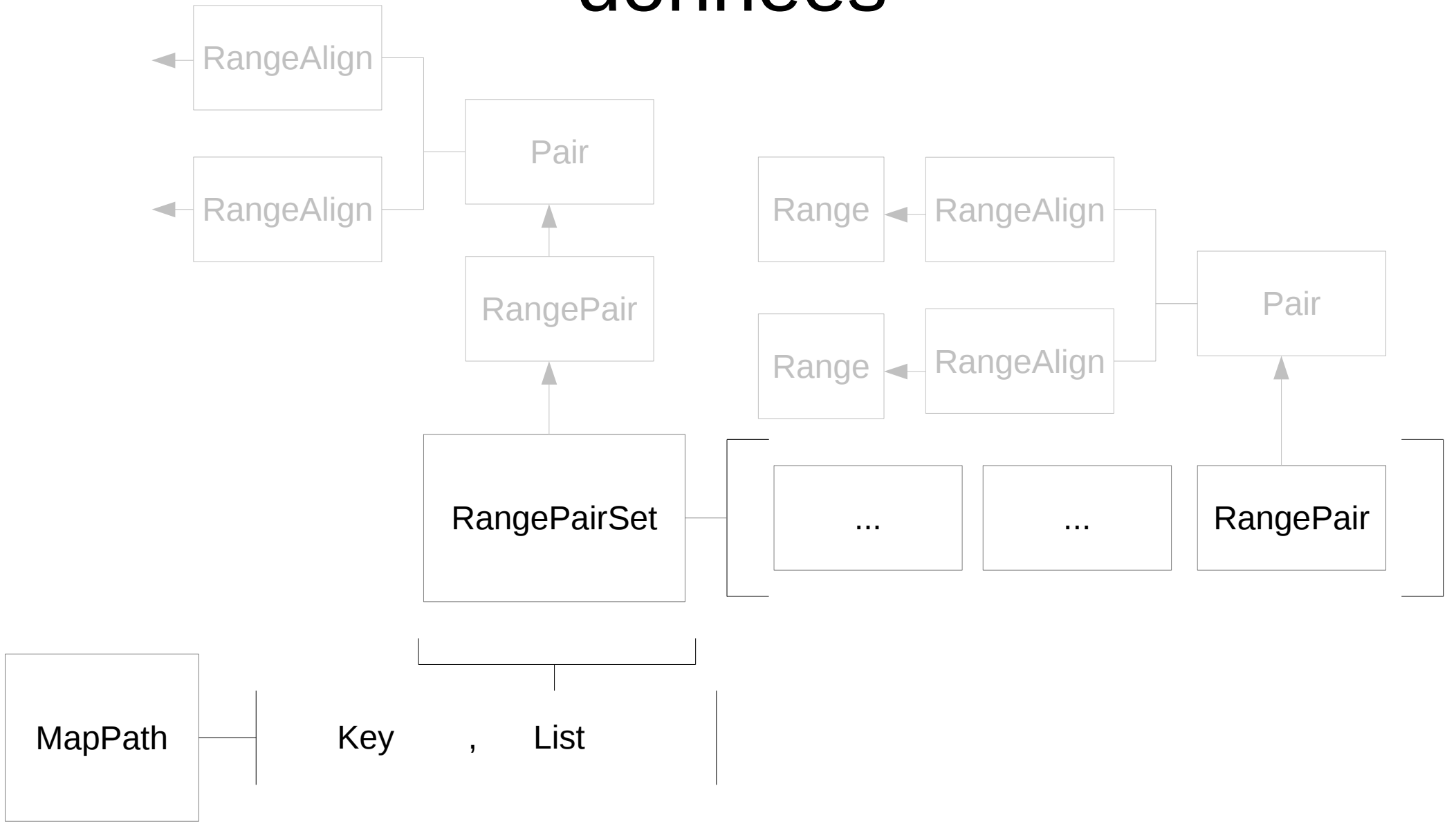
Mise en oeuvre

- Comment écrire les tests ?
- Pour écrire un test F:
  - Isoler le plus petit jeu de données qui provoque le bug
  - => des données en input
  - => des données de réf en output
- Pour écrire un ~~test U~~:
  - La structure avec les données qui provoquent le bug
  - 1 structure de données en input
  - 1 structure de données de réference en output

Pour le TF: des données

- Input:
  - chunk682 4361 4811 refTE_230 3047 2546 0 451 91.571800
  - chunk682 13408 13628 refTE_230 2595 2388 0 221 81.122400
  - chunk682 45755 46076 refTE_230 3051 2742 0 322 88.025900
  - chunk682 46196 46323 refTE_230 2679 2546 0 128 84.920600
  - chunk682 46450 46706 refTE_230 2596 2353 0 257 88.841200
  - ...
- Output:
  - 1 chunk682 104183 104531 refTE_230 2409 2748 8.1e-19 349 76.0518
  - 1 chunk682 104532 104604 refTE_266 344 420 1e-20 73 71.4724
  - 2 chunk682 105085 105353 refTE_251 3202 2921 0 269 84.8361
  - 2 chunk682 109951 109984 refTE_266 149 119 9.4e-19 34 77.768
  - 2 chunk682 109985 110398 refTE_251 2429 2033 0 414 77.8604

Pour le ~~TU~~: une structure de données



1) Ecrire la structure de données

- Ecriture “manuelle”
- Une méthode pour générer une structure de données: 157 lignes

```
BLRMatchMap::MapPath Test_BLRMatchMapUtils::createExpMapPath_for_mapPath(void){
  BLRMatchMap::MapPath mapPath;

  SDGString line1 = " \t105085\t110569\t-1\t0\t0\t0\t888\t0";
  RangePairSet rangePairSet1 = RangePairSet(line1);
  rangePairSet1.getRangeQ().setNumChr(1);
  rangePairSet1.getRangeS().setNumChr(-1);
  rangePairSet1.getRangeQ().setNameSeq("");
  rangePairSet1.getRangeS().setNameSeq("-1");

  rangePairSet1.setLength(888);

  SDGString line11 = " \t105085\t105353\t \t3202\t2921\t0\t269\t84.8361";
  RangePair rangePair11 = RangePair(line11);
  rangePair11.getRangeQ().setNumChr(1);
  rangePair11.getRangeS().setNumChr(1);
  rangePair11.getRangeQ().setNameSeq("");
  rangePair11.getRangeS().setNameSeq("");
  rangePair11.setLength(269);


  //SDGString line12 = " \t109951\t109984\t \t119\t149\t9.4e-19\t34\t77.768";
  SDGString line12 = " \t109951\t109984\t \t149\t119\t9.4e-19\t34\t77.768";
  RangePair rangePair12 = RangePair(line12);
  rangePair12.getRangeQ().setNumChr(1);
  rangePair12.getRangeS().setNumChr(2);
  rangePair12.getRangeQ().setNameSeq("");
  rangePair12.getRangeS().setNameSeq("");
  rangePair12.setLength(34);

  SDGString line13 = " \t109985\t110398\t \t2429\t2033\t0\t414\t77.8604";
  RangePair rangePair13 = RangePair(line13);
  rangePair13.getRangeQ().setNumChr(1);
  rangePair13.getRangeS().setNumChr(1);
  rangePair13.getRangeQ().setNameSeq("");
  rangePair13.getRangeS().setNameSeq("");
  rangePair13.setLength(414);

  //SDGString line14 = " \t110399\t110569\t \t532\t689\t9.4e-19\t171\t77.768";
  SDGString line14 = " \t110399\t110569\t \t689\t532\t9.4e-19\t171\t77.768";
  RangePair rangePair14 = RangePair(line14);
  rangePair14.getRangeQ().setNumChr(1);
  rangePair14.getRangeS().setNumChr(2);
  rangePair14.getRangeQ().setNameSeq("");
  rangePair14.getRangeS().setNameSeq("");
  rangePair14.setLength(171);


  // push range to range pair
  std::list<RangePair> rpList1;
  rpList1.push_back(rangePair11);
  rpList1.push_back(rangePair12);
  rpList1.push_back(rangePair13);
  rpList1.push_back(rangePair14);


  // set match part to match
```

2) Systématiser l'écriture

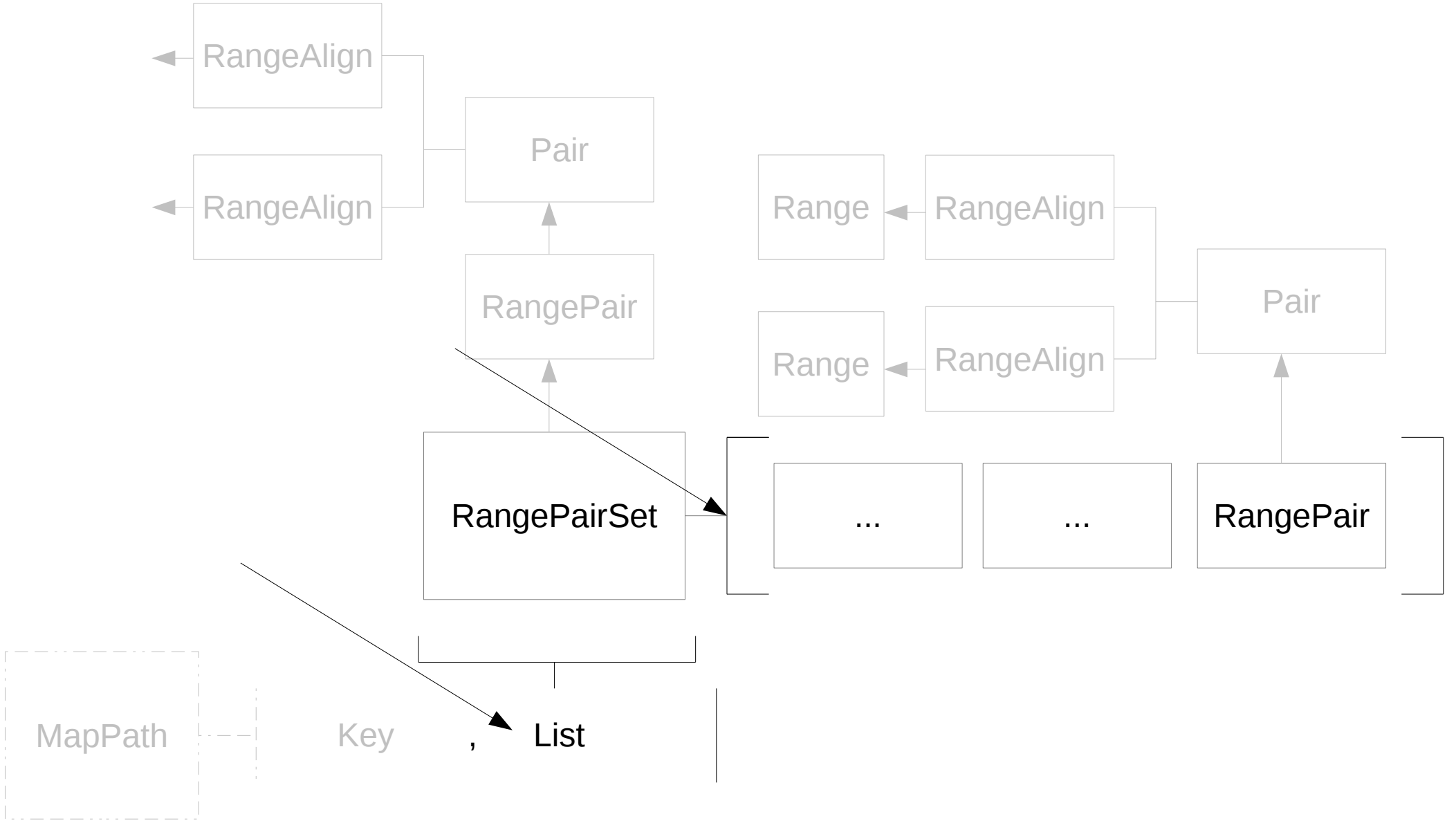
- Ecriture “assitée”
  - Une mérhode générique pour écrire la structure:
    - BLRMatchMap::MapPath Test_BLRMatchMapUtils::**createMapPath**(std::list<SDGString> inputList);
  - L'écriture du test:
    - std::list<SDGString> Test_BLRMatchMapUtils::**generateOutputs_for_test_merge** (void){
  - SDGString str1 = "1\t1\t750\t-1\t0\t0\t0\t481\t0\t0\t481\n";
  - SDGString str11 = "\t1\t100\t1\t1\t3351\t3238\t0\t100\t76.1062\t100\n";
  - std::list<SDGString> lPath;
  - lPath.push_back(str1);
  - lPath.push_back(str11);

    return lPath;
  - }
- La méthode générique sans Test ...

Travailler sur toute la structure ?

- Pour certains cas de test, tous les éléments de la structure ne sont pas pertinents
- class BLRMatchMap
  - {
  - public:
  - **struct Key : std::pair<long,long>**
  - {
  - Key(long i,long j)
  - {
  - first=i; second=j;
  - };
  - };
- Dans les structures comparées les valeurs de clés sont les mêmes ...

3) Court-circuiter la structure

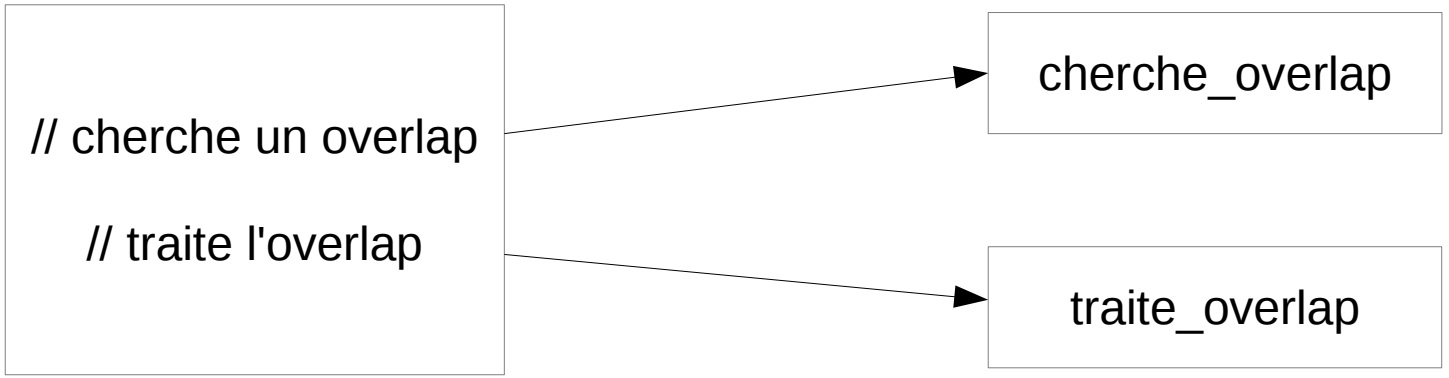


Travailler sur toute la méthode ?

- Une méthode “fait” plus qu'une “chose”
- Exemple:
- void BLRMatchMap::**add_split_path**(std::list<RangePairSet>& rp_list, std::list<RangePairSet>::iterator iter)
  - // cherche un overlap entre 2 frgts
  - // si overlap: maj de la structure
  - // sinon: augmente la structure
- Comment comprendre ? Comment tester ?

4) Court circuiter la méthode

- Réduire la méthode: bool BLRMatchMap::**isOverlapFound_in_add_split_path** …
- Tester la méthode réduite
- Et (mieux) extract method:

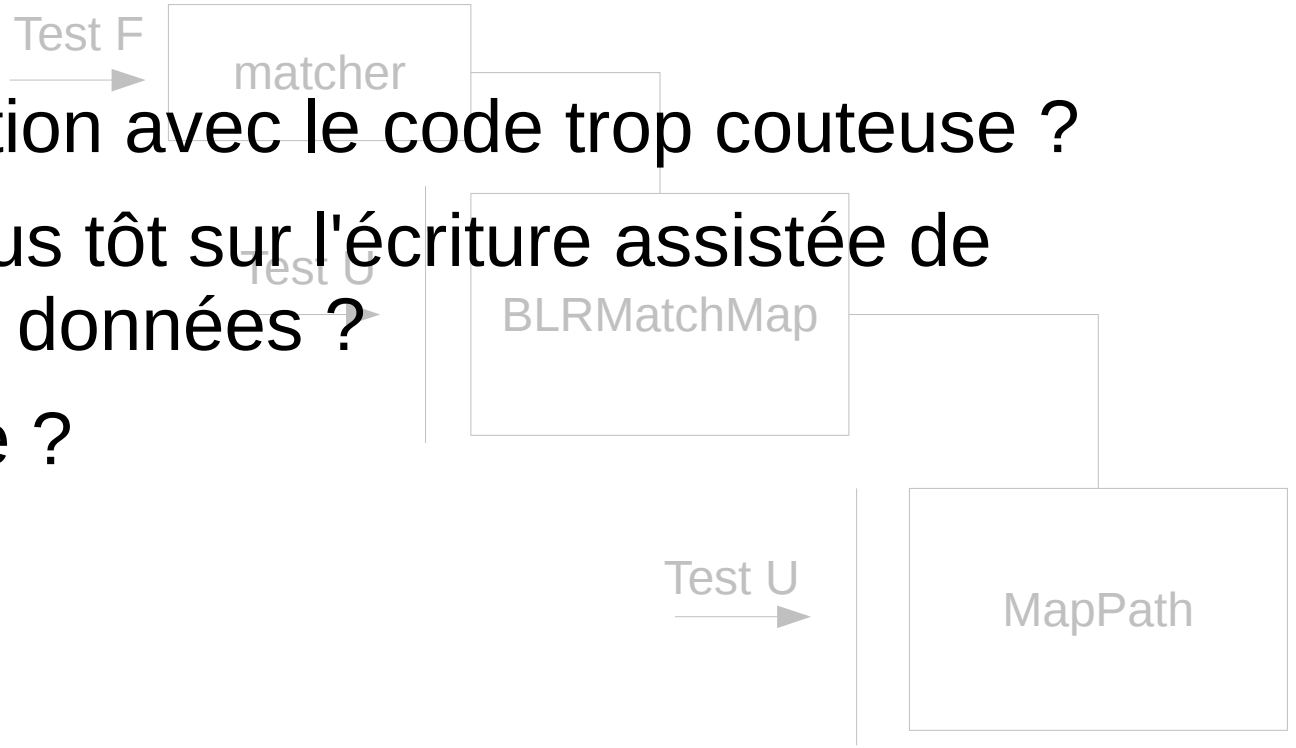


Bilan

- Test F: faible coût de mise en place
- ~~Test U~~: fort coût de mise en place:
  - ~~Génération de structure de données manuelle~~
  - Génération de structure de données assistée
  - ~~Court circuit sur la structure de données~~
  - Court circuit sur la méthode

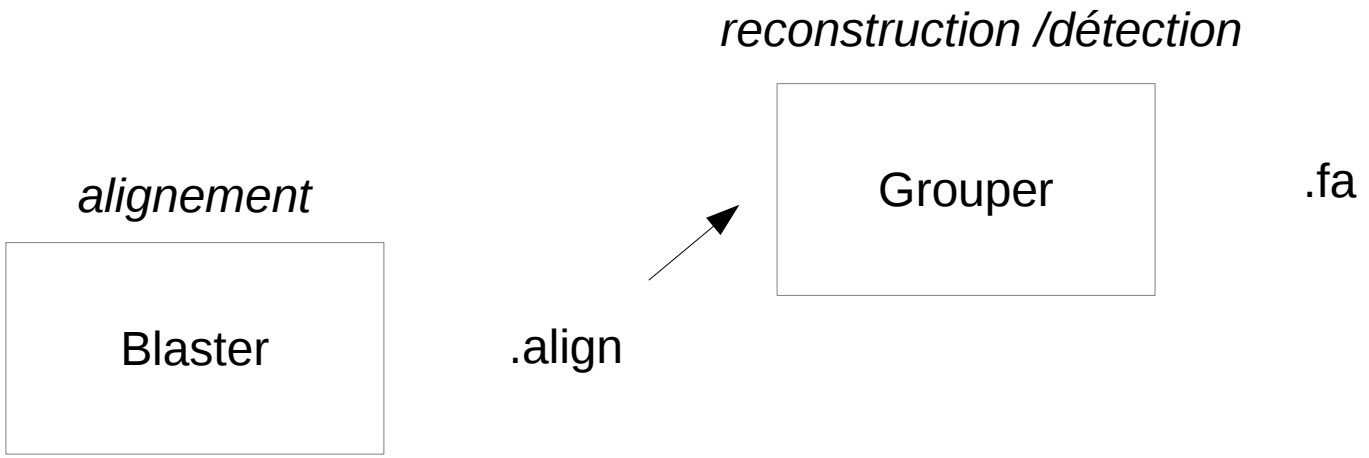
Jusqu'où aller ?

- Tests F => OK
- ~~Tests U~~:
  - Documentation avec le code trop couteuse ?
  - Travailler plus tôt sur l'écriture assistée de structure de données ?
  - Effet newbie ?

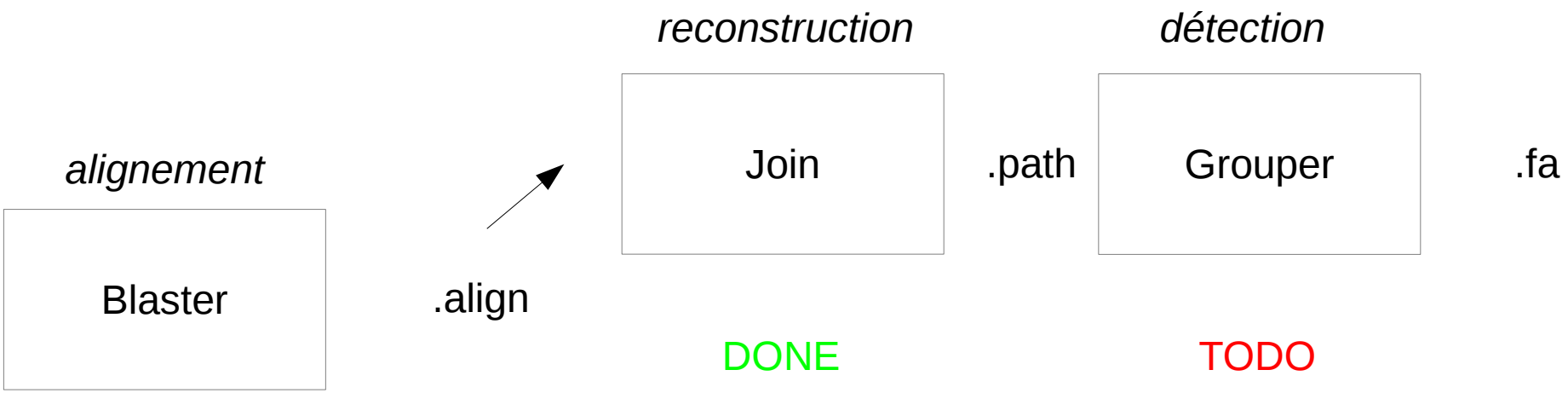


Jusqu'où aller ?

- Au délà des ~~TU~~ et TF
- Pour illustrer: un nouveau cas

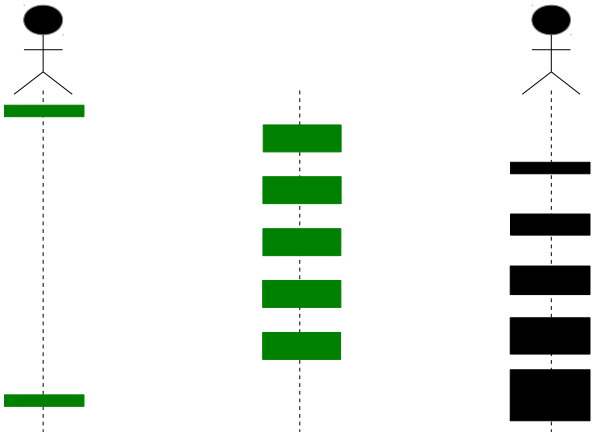


Un nouveau cas



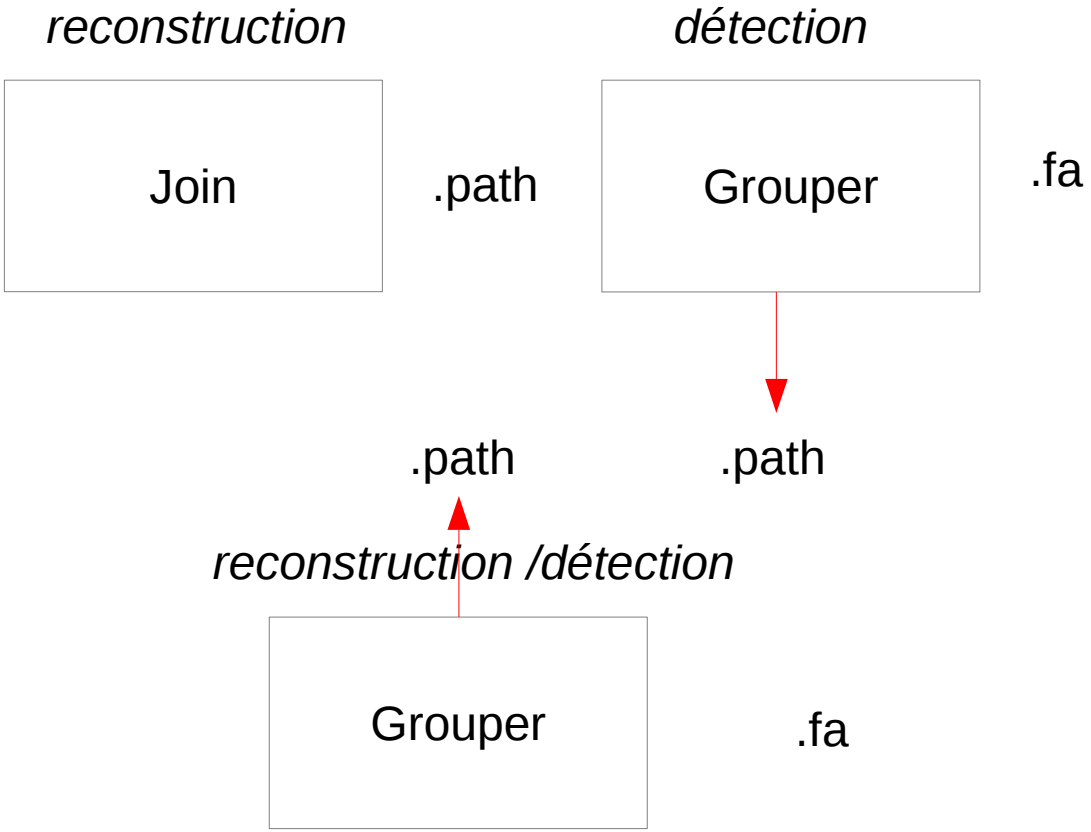
Un nouveau cas

- Le module detection load un fichier path
- Phase 1:
  - Approche ~~TU~~/TF: ok
  - “Release”: run sur données conséquentes
  - Bug !
- Phase 2:
  - Debug en bînome



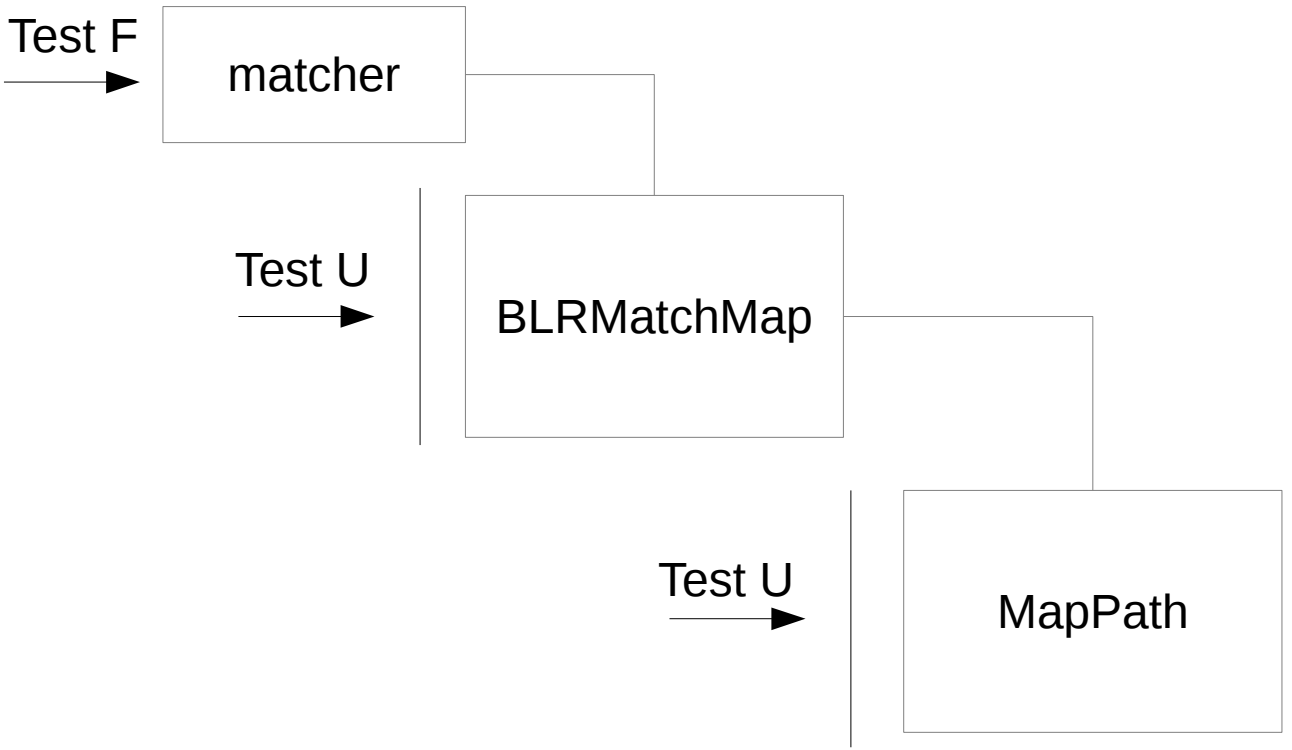
Debug

- Scénario de contrôle
- Structure de données => fichiers
- Manipulation de fichiers: sort, cut, …

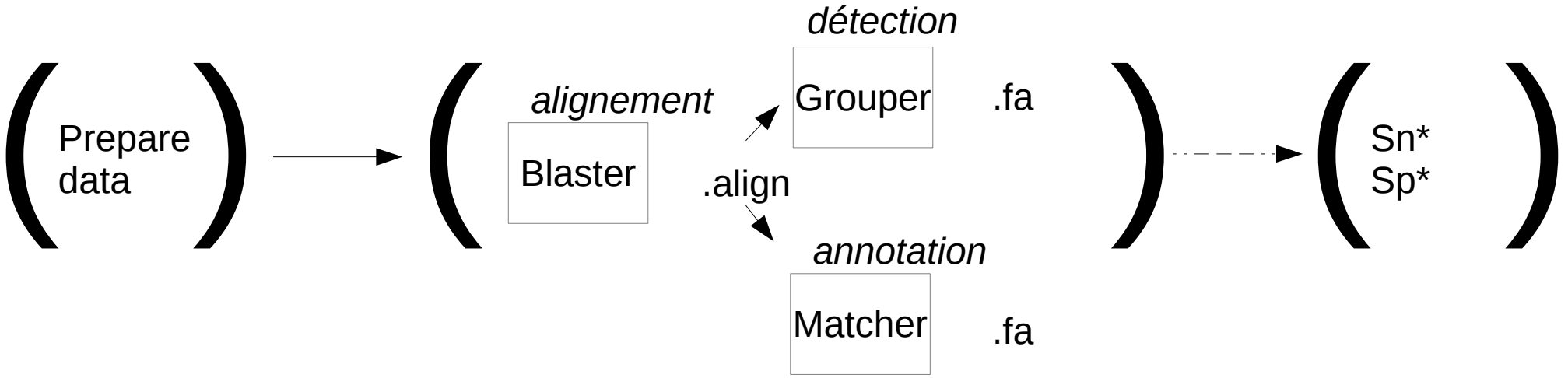


La démarche

- ~~TU~~/TF
- Fichiers ...

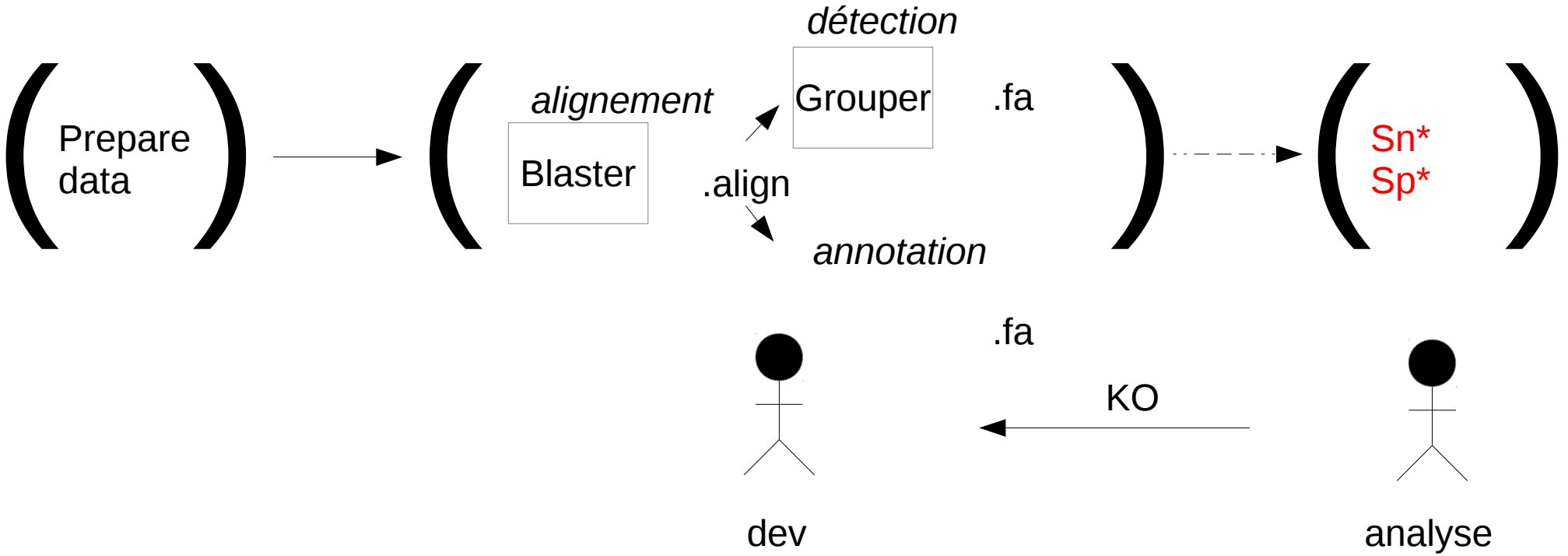


Pipeline


Prepare data
alignement
Blaster
détection
Grouper
.fa
.align
annotation
Matcher
.fa
Sn*
Sp*

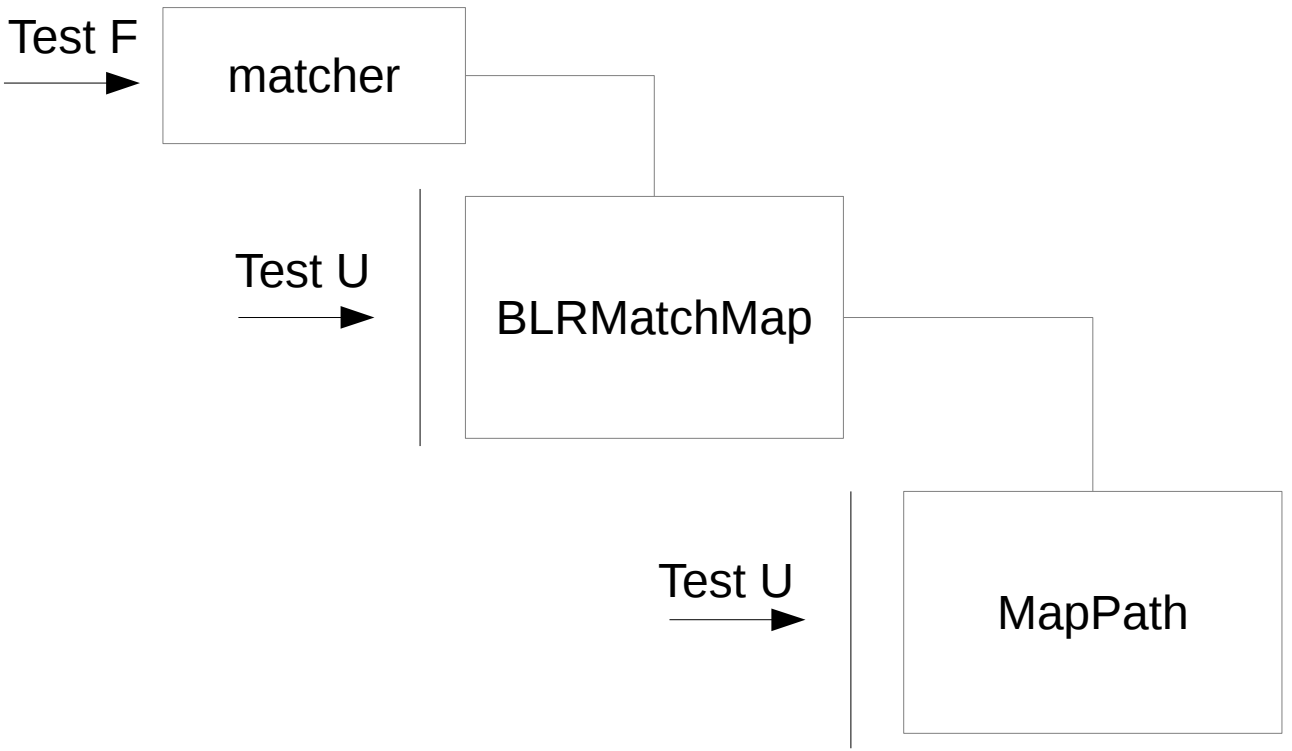
Retour sur ...

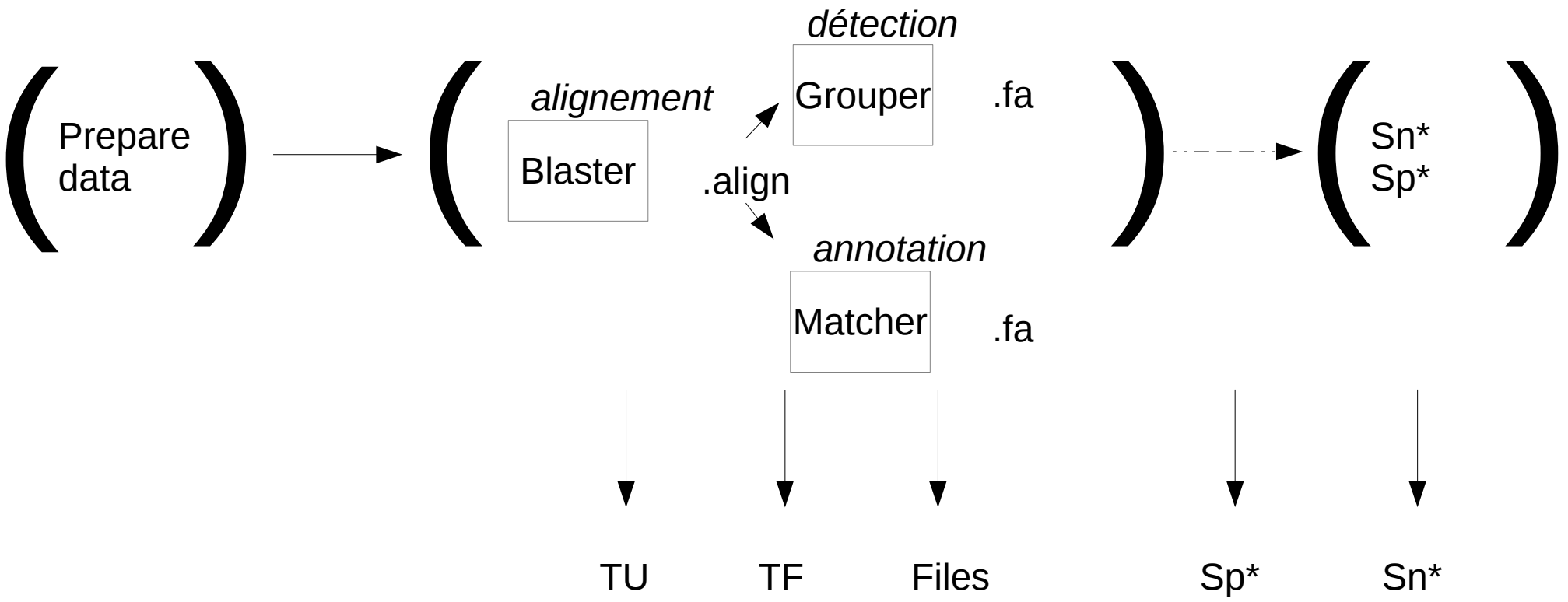
- Release: “run” sur des données conséquentes
- Bug !



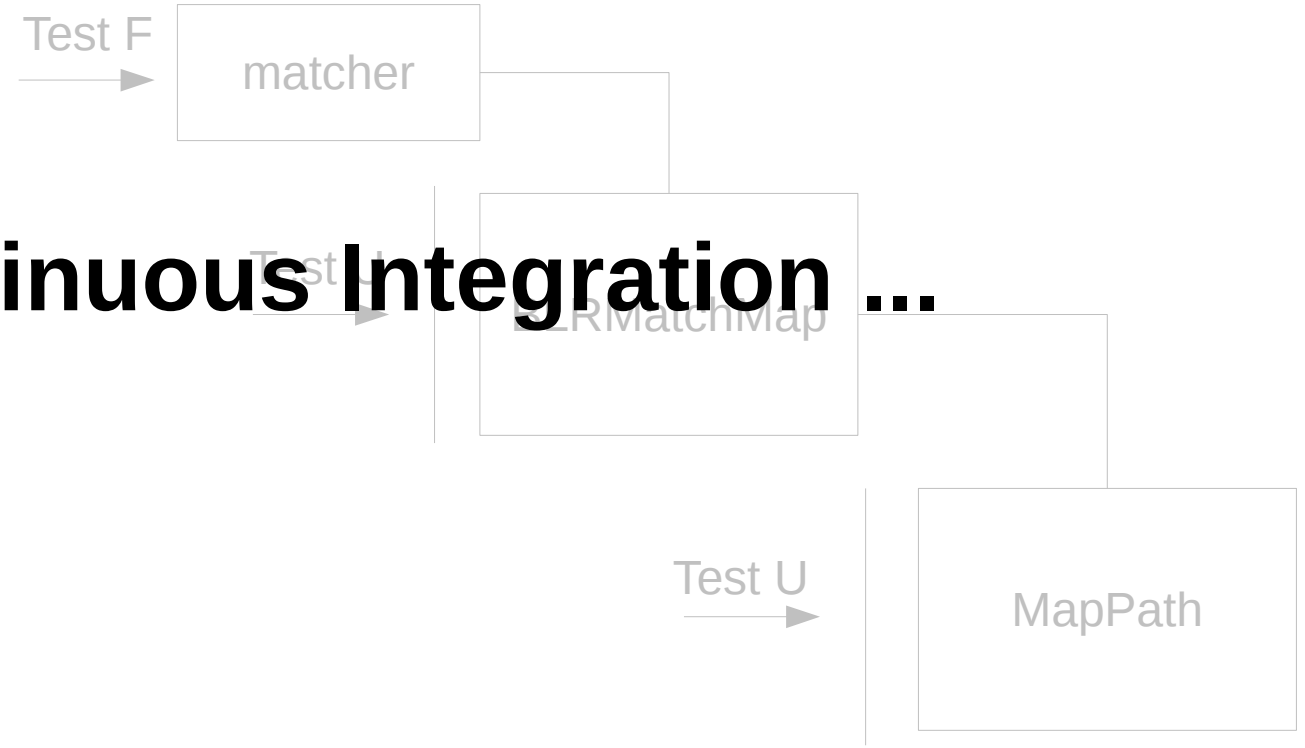
La démarche

- ~~TU~~/TF
- Fichiers
- Sp*, Sn*




Prepare data
alignement
Blaster
détection
Grouper
.fa
.align
annotation
Matcher
.fa
Sn*
Sp*
TU
TF
Files
Sp*
Sn*

Jusqu'où aller ?



Continuous Integration ...

Merci !