CCIN2P3

cnrs

Centre de Calcul
de l'Institut National de Physique Nucléaire
et de Physique des Particules

Optimisation de la pipeline de traitement des images de Rubin-LSST

Quentin Le Boulc'h

Avec J. Bregeon (LPSC), D. Boutigny (LAPP), F. Hernandez (CC-IN2P3),
D. Parello (Univ. Perpignan), C. Parisel (APC)

VERA C. RUBIN OBSERVATORY

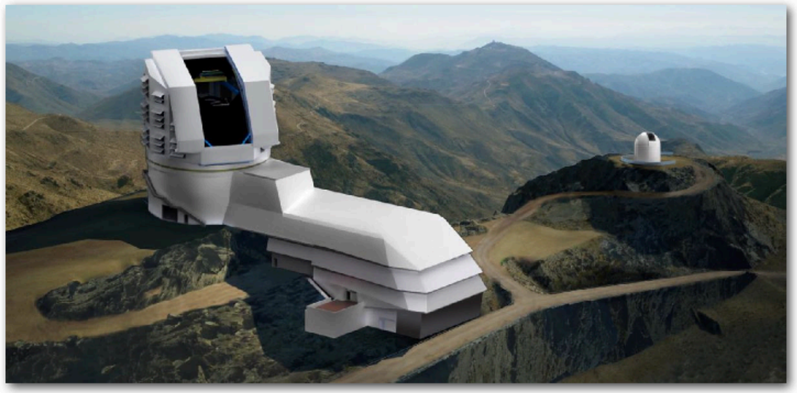
LSST

- L’observatoire Vera C. Rubin et le LSST
- Traitement des données
- Profilage de la pipeline
- Résumé et perspectives

L’observatoire Vera C. Rubin et le LSST

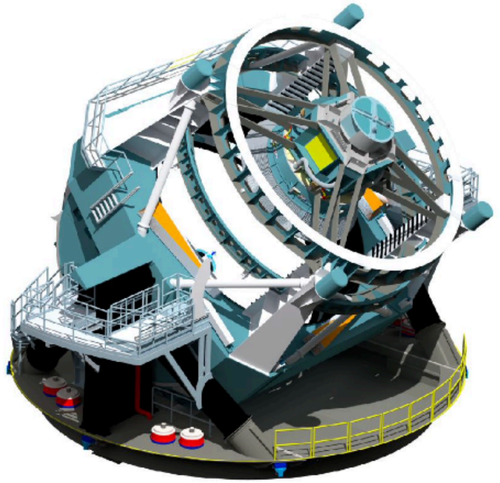
L'observatoire Vera C. Rubin et le LSST

Observatory



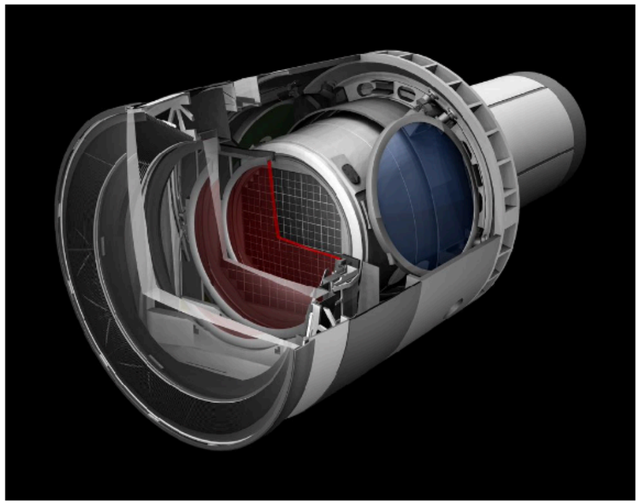
southern hemisphere | 2647m a.s.l. | stable air | clear sky | dark nights | good infrastructure

Telescope



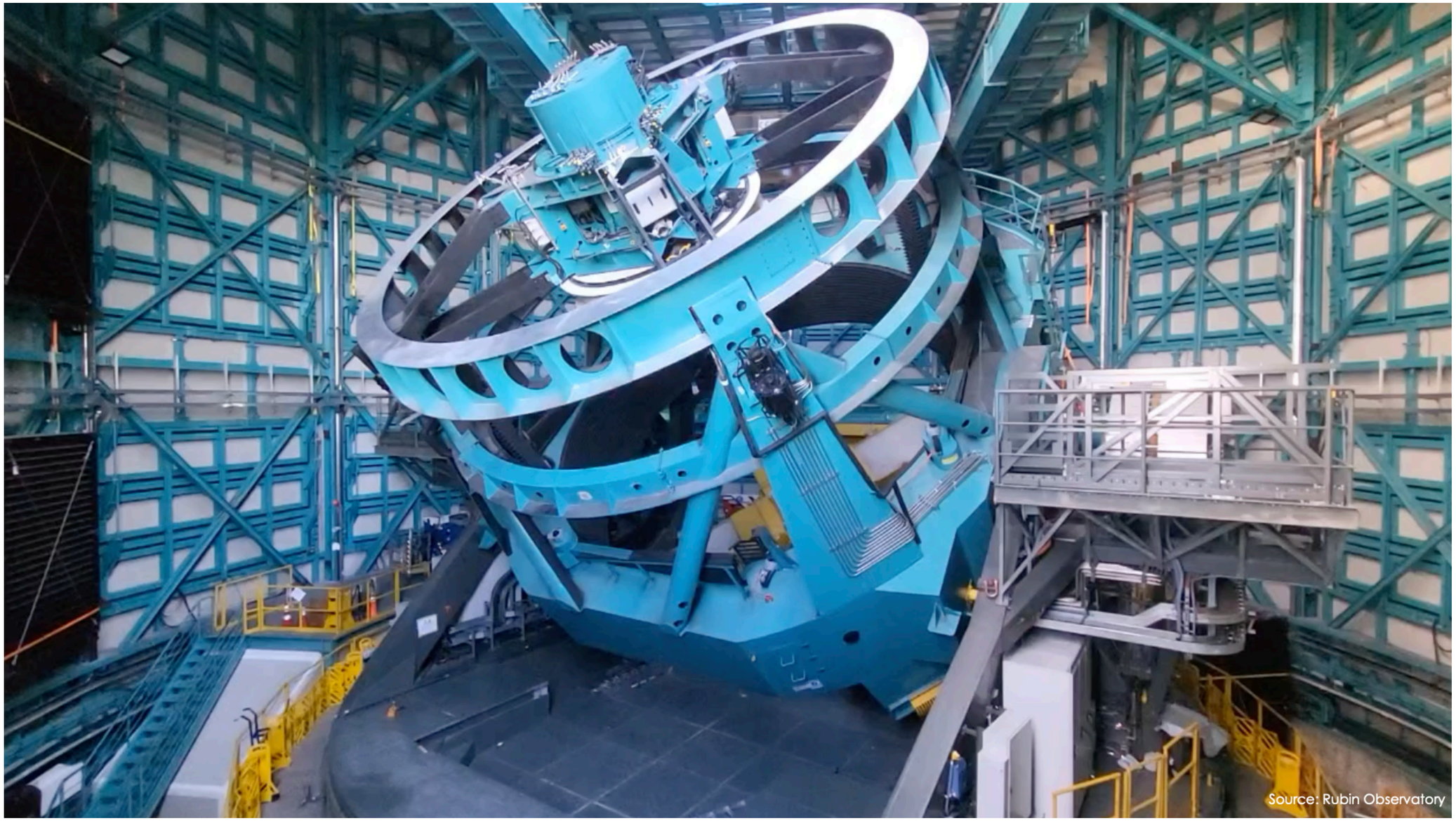
main mirror ⌀ 8.4 m (effective 6.4 m) | large aperture: f/1.234 | wide field of view | 350 ton | compact | to be repositioned about 3M times over 10 years of operations

Camera

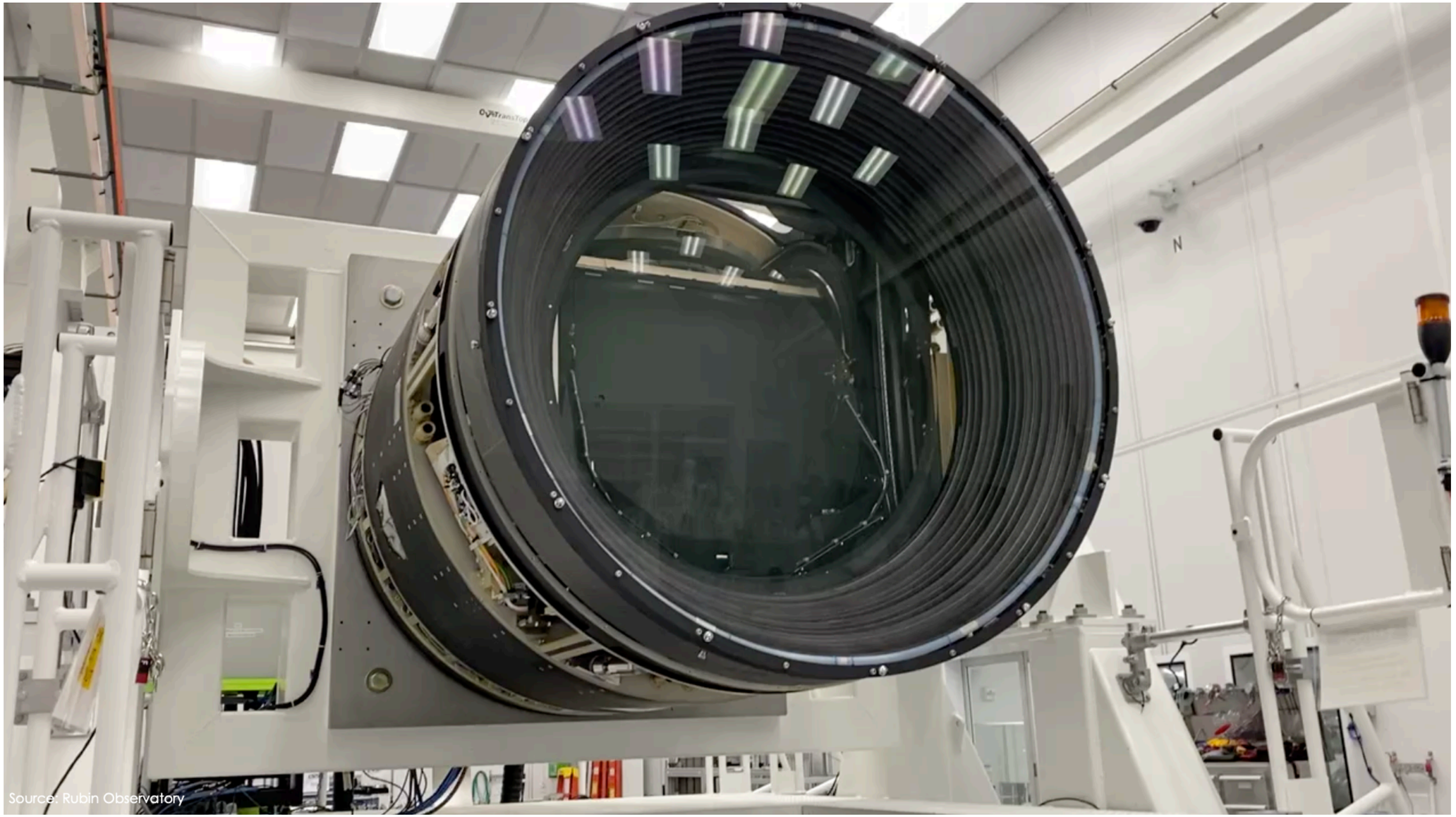


3.2 G pixels | ⌀ 1.65 m | 3.7 m long | 3 ton | 3 lenses | 3.5° field of view | 9.6 deg² | 6 filters *ugrizy* | 320-1050 nm | focal plane and electronics in cryostat at 173K

L’observatoire Vera C. Rubin et le LSST



L’observatoire Vera C. Rubin et le LSST



L’observatoire Vera C. Rubin et le LSST

90% du temps d’observation dédié au Legacy Survey of Space and Time (LSST) :

- 43% du ciel
- Ciel de l’hémisphère sud observé en totalité toutes les 4 nuits
- Chaque zone du ciel observée sera visitée environ 800 fois dans 6 bandes de longueur d’onde différentes
- A partir de 2025 et pendant 10 ans

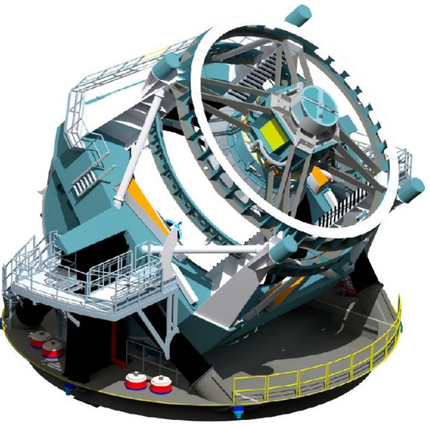
Objectifs scientifiques :

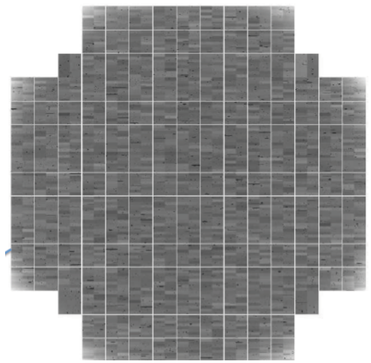
- Explorer la nature de la matière noire et de l’énergie sombre
- Etude du système solaire et de la voie lactée
- Etude des objets transients

Contributions de l’IN2P3 :

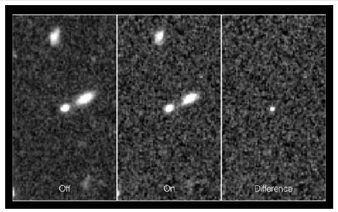
- 10 laboratoires IN2P3 : APC, CC-IN2P3, CPPM, IJCLab, IP2I, LAPP, LPC, LPNHE, LPSC, LUPM
- Plus de 100 personnes (chercheurs, ingénieurs, techniciens)
- Contributions à la caméra, changeur de filtre, traitement des données, analyse scientifique

L'observatoire Vera C. Rubin et le LSST

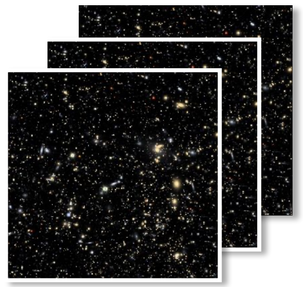




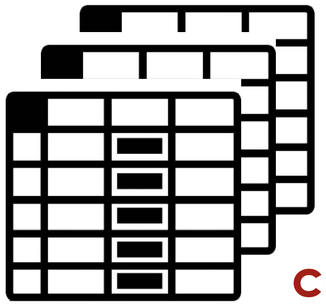
images brutes



alertes



images prêtes pour la science



catalogue astronomique

collaborations scientifiques

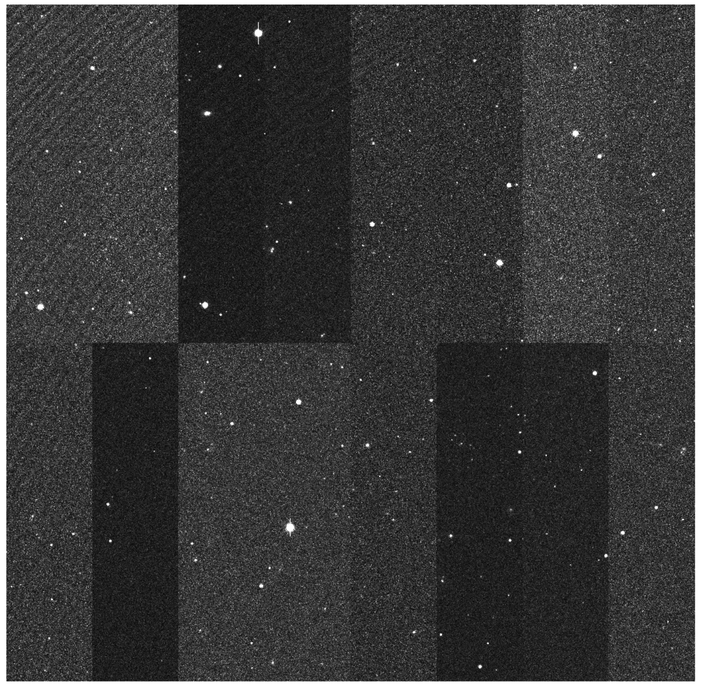
L'objectif de LSST est de cataloguer les propriétés physiques de 20 milliards de galaxies et 17 milliards d'étoiles

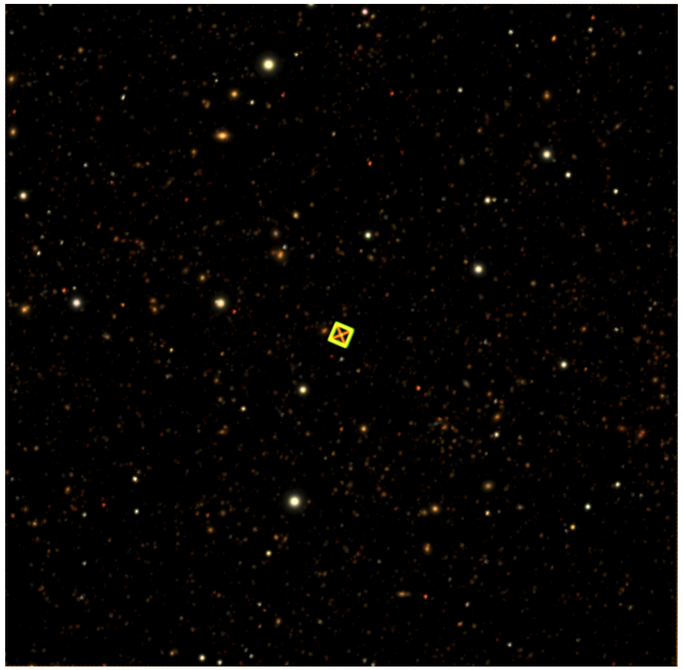
Traitement des données

Traitement des données

Objectifs de la pipeline de traitement des images :

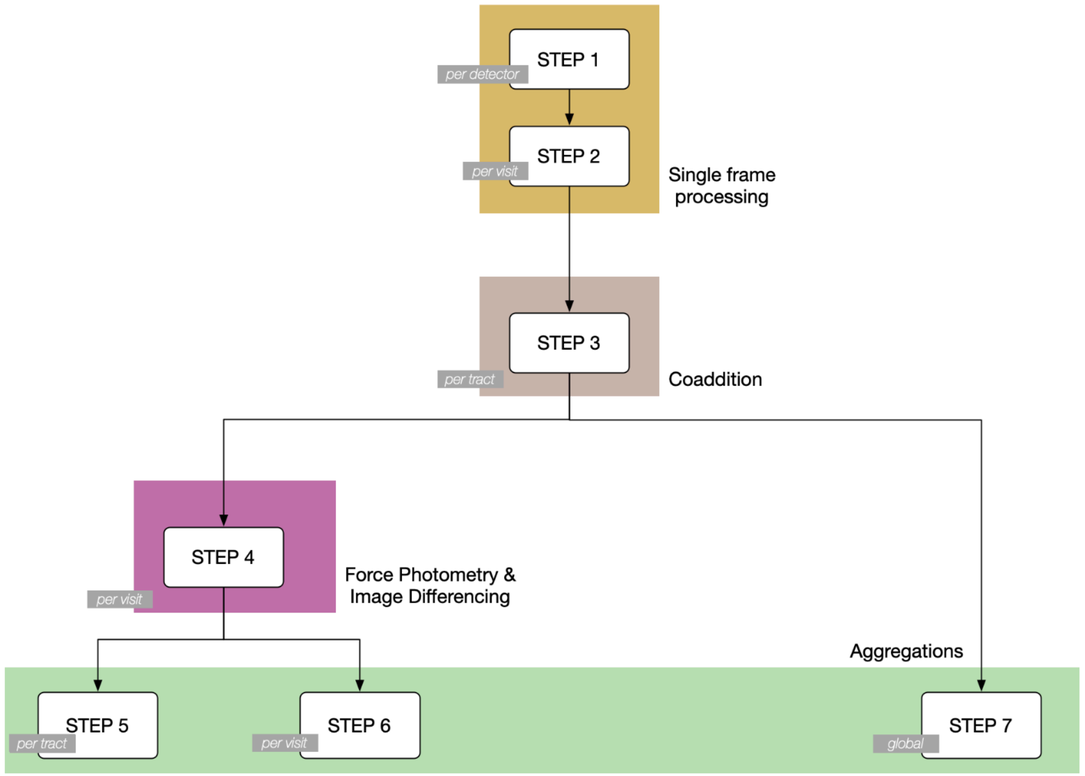
- **Traitement des images brutes** et calibration (suppression des effets instrumentaux, …)
- Empilement d'images (**coaddition**) pour améliorer le rapport signal/bruit
- **Mesure de différences** entre images pour la détection d'objects transients
- Détection d'objets et mesure de leurs propriétés, création de catalogues

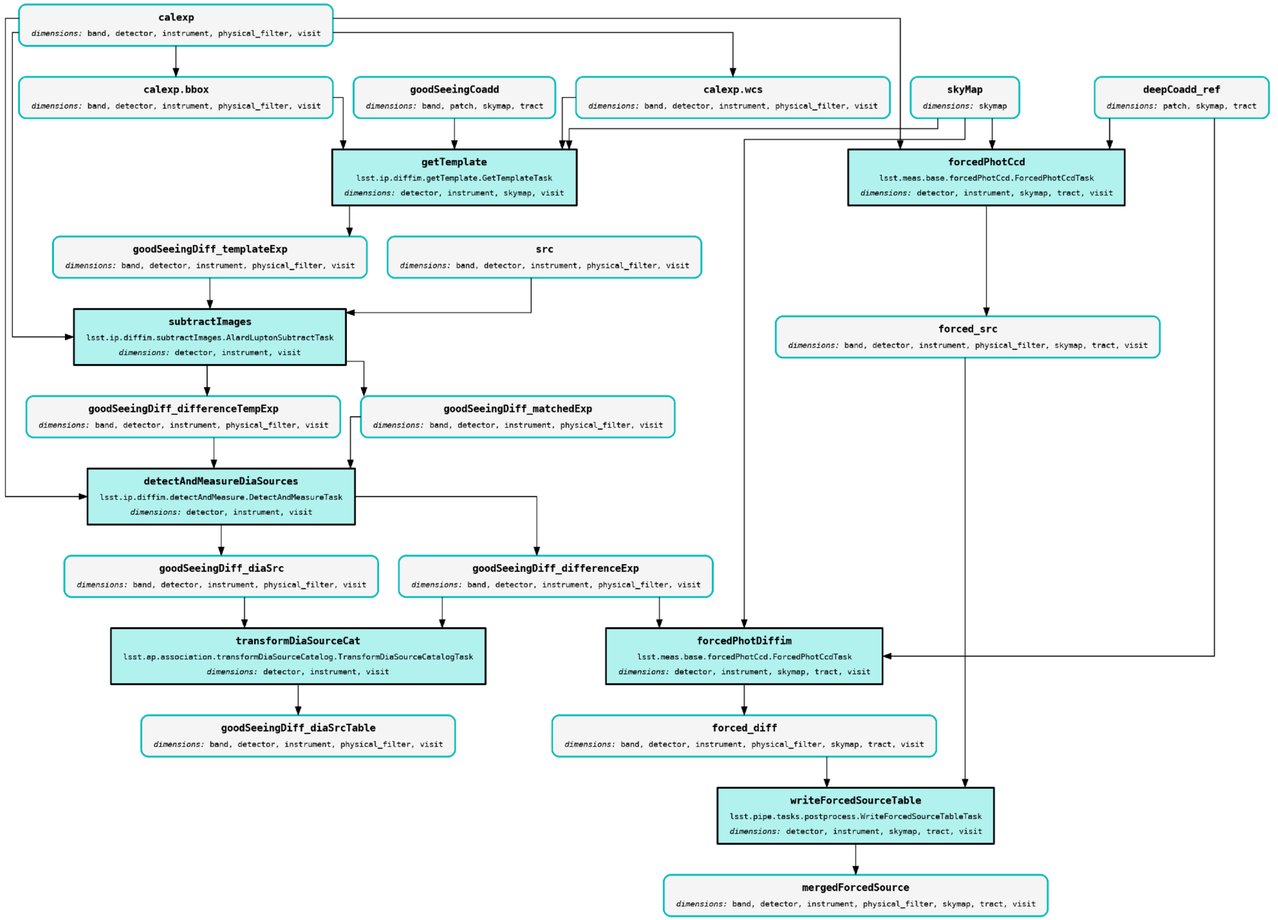


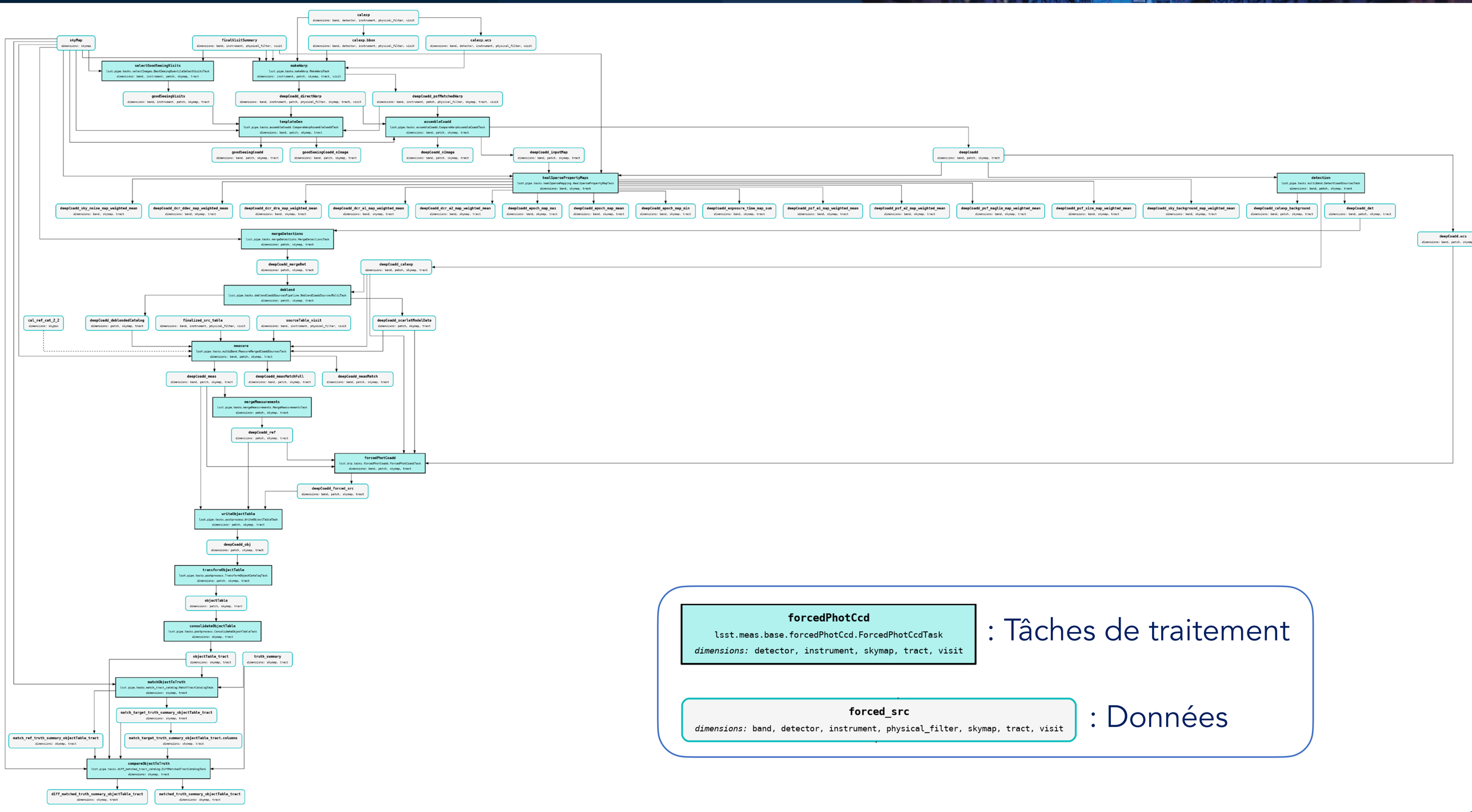


Traitement des données

- Ensemble d'environ 80 tâches de traitement, regroupées en 7 étapes
- Chaque correspond à un workflow (dépendances entre les tâches via les données)







forcedPhotCcd
lsst.meas.base.forcedPhotCcd.ForcedPhotCcdTask
dimensions: detector, instrument, skymap, tract, visit

: Tâches de traitement

forced_src
dimensions: band, detector, instrument, physical_filter, skymap, tract, visit

: Données

Traitement des données

Exigences fortes sur les performances au vu des volumes en jeu :

- 6.4 GB par exposition (images brutes)
- 2500 images (science + calibration) par nuit soit 16 TB (5 PB par an)
- 0.5 EB de données générées au total après 10 ans
- 15 PB de base de données après 10 ans (catalogues)

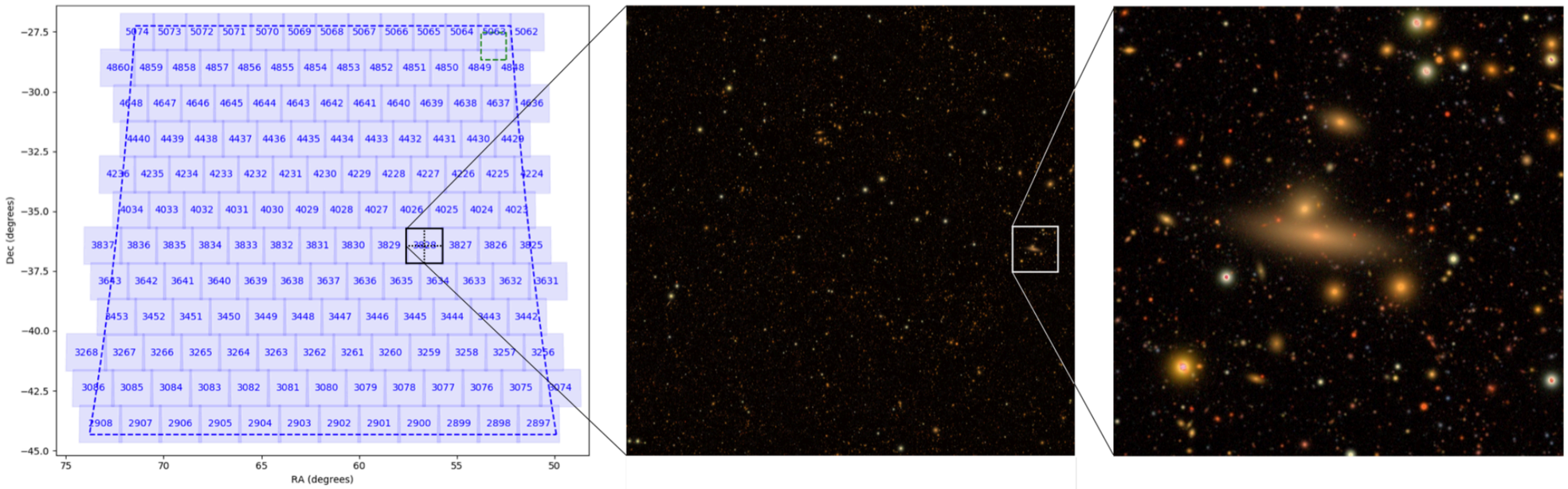
Contributions aux traitements :

- USA : 35%
- UK : 25%
- France (CC-IN2P3) : 40%

Traitement des données

Exemple de la campagne de traitements DP0.2 (2022) :

- Traitement de données simulées produites par la Dark Energy Science Collaboration
- Equivalent à 0.5% du relevé, ou 10 nuits de prise de données
- 58 M de tâches exécutées, pour plus de 2 M d'heures CPU
- 3 PB de données générées



Traitement des données

Fort enjeu sur la réduction de l'utilisation des ressources (temps CPU, mémoire, stockage) :

- Financier
- Environnemental
- Performances

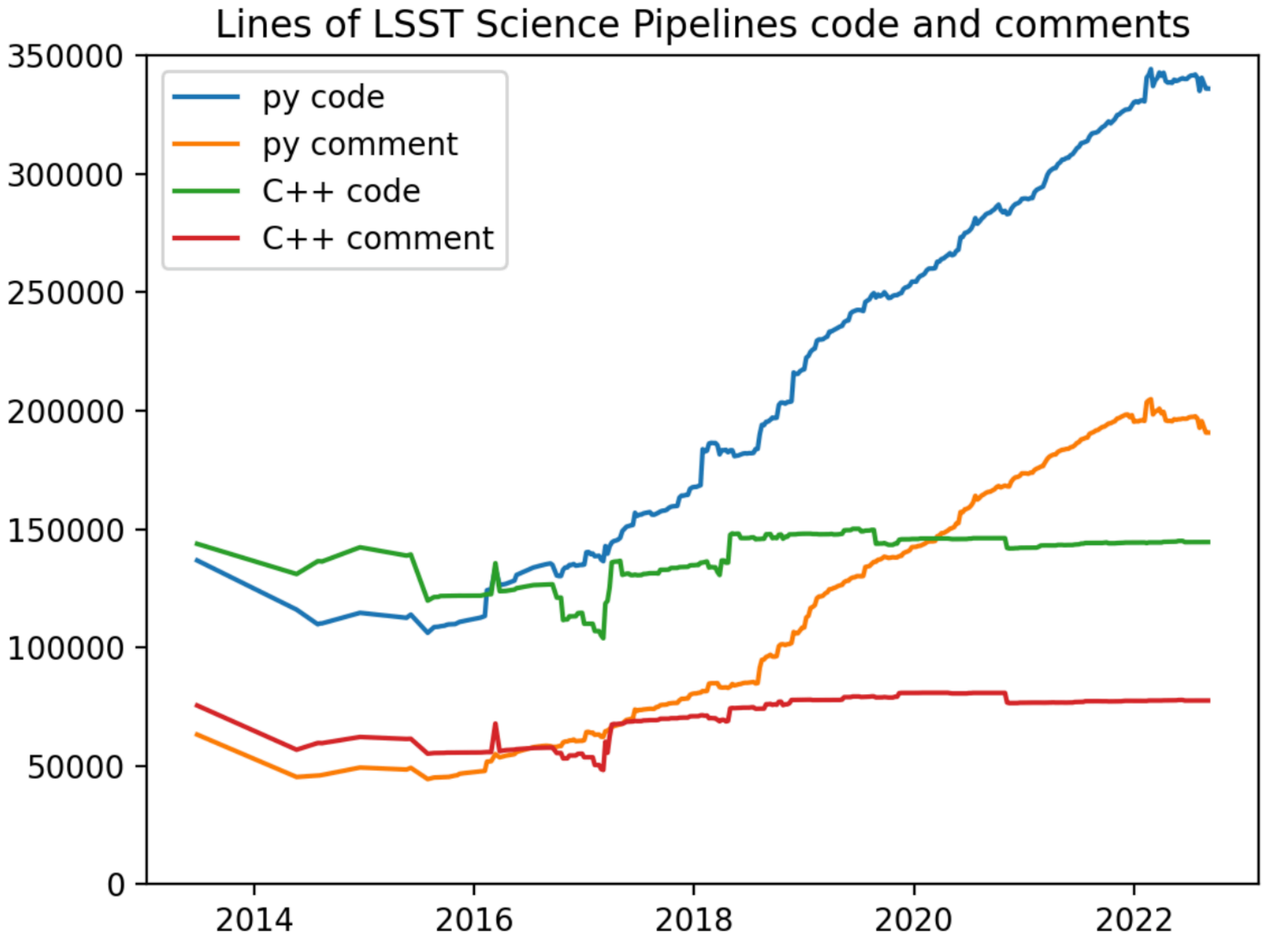
- Code open source développé essentiellement par le projet Rubin (US)
- Intérêt commun sur l'optimisation

=> Projet d'optimisation de la pipeline de traitement des données porté à l'IN2P3 (projet RubinOP de la Mission pour les Initiatives Transverses et Interdisciplinaires)

Avant d'optimiser : besoin de mesures fiables pour comprendre ce qui doit (et peut) être optimisé

Profilage de la pipeline

Profilage de la pipeline

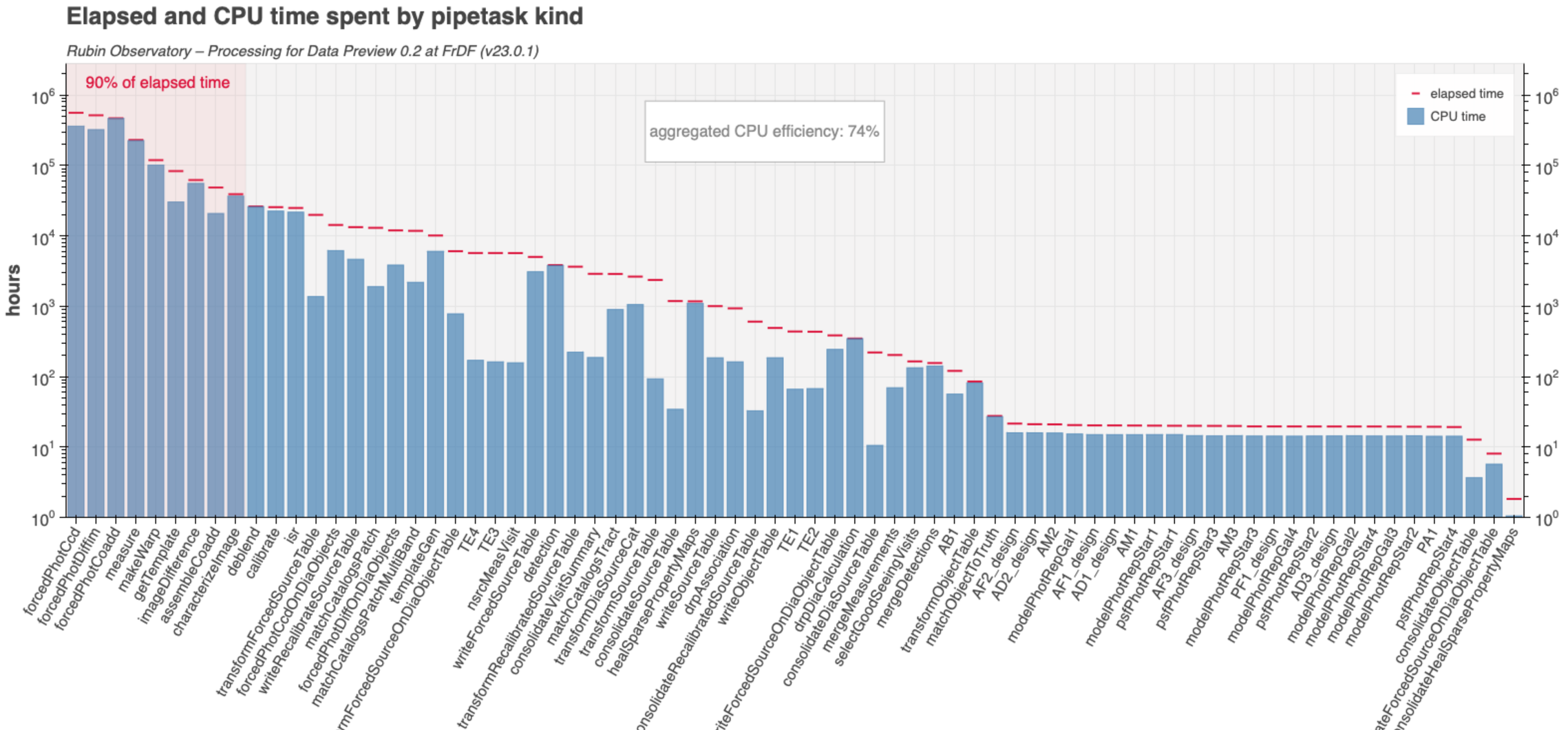


- Coeur du code en C++
- Surcouche Python
- Code disponible sur https://github.com/lsst
- Documentation : https://pipelines.lsst.io

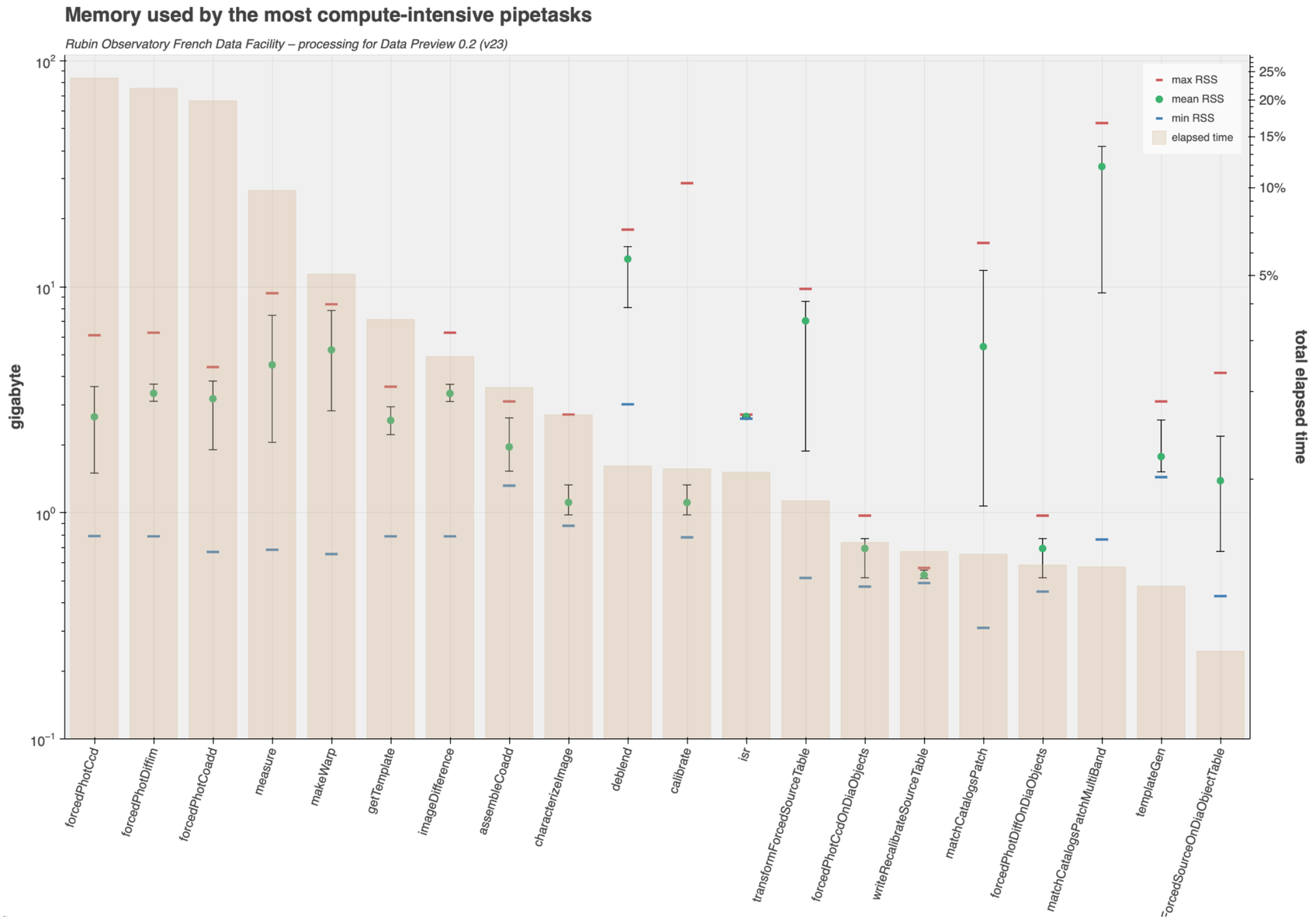
Source: T. Jenness, Rubin Observatory

Profilage de la pipeline

- Le framework d'exécution LSST fournit des mesures de temps de calcul et utilisation mémoire
- Déjà utilisé pour analyser l'utilisation des ressources lors de la campagne DP0.2 :



Profilage de la pipeline



Profilage de la pipeline

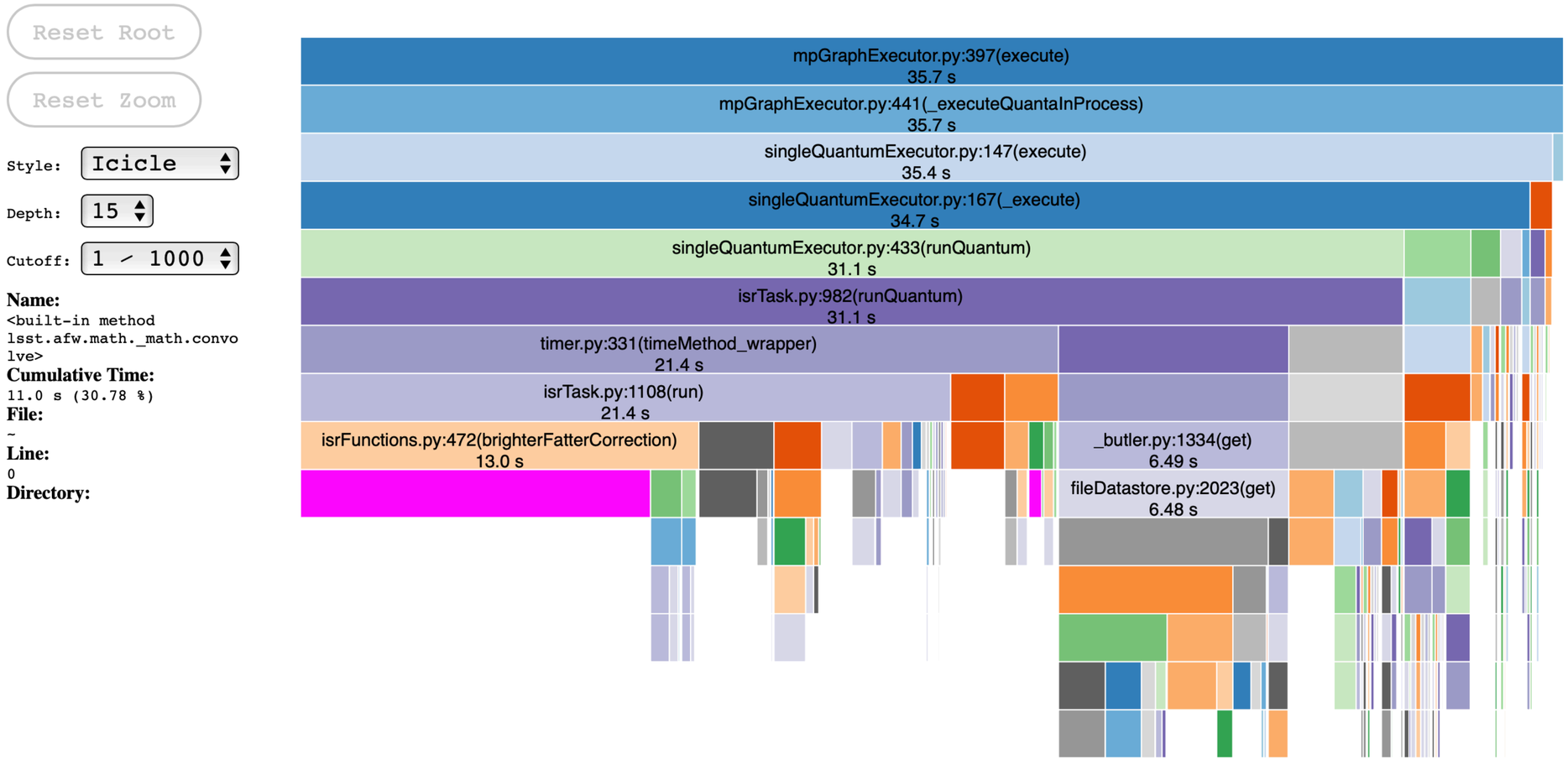
Les limites de cette analyse :

- Uniquement une vue globale, pas de profilage fin sur chaque tâche
    - Besoin de mesurer l'utilisation CPU et mémoire au sein des tâches (par fonction ou ligne de code)

- Exécution à grande échelle, complexe et longue à reproduire
    - Besoin d'une pipeline de référence simple et rapide à exécuter, reproductible et représentative
    - https://github.com/lsst/ci_hsc_gen3 : petit jeu de données de test de la camera HSC (Subaru)
    - Traitement complet avec la pipeline LSST en quelques heures sur un coeur CPU

Profilage de la pipeline

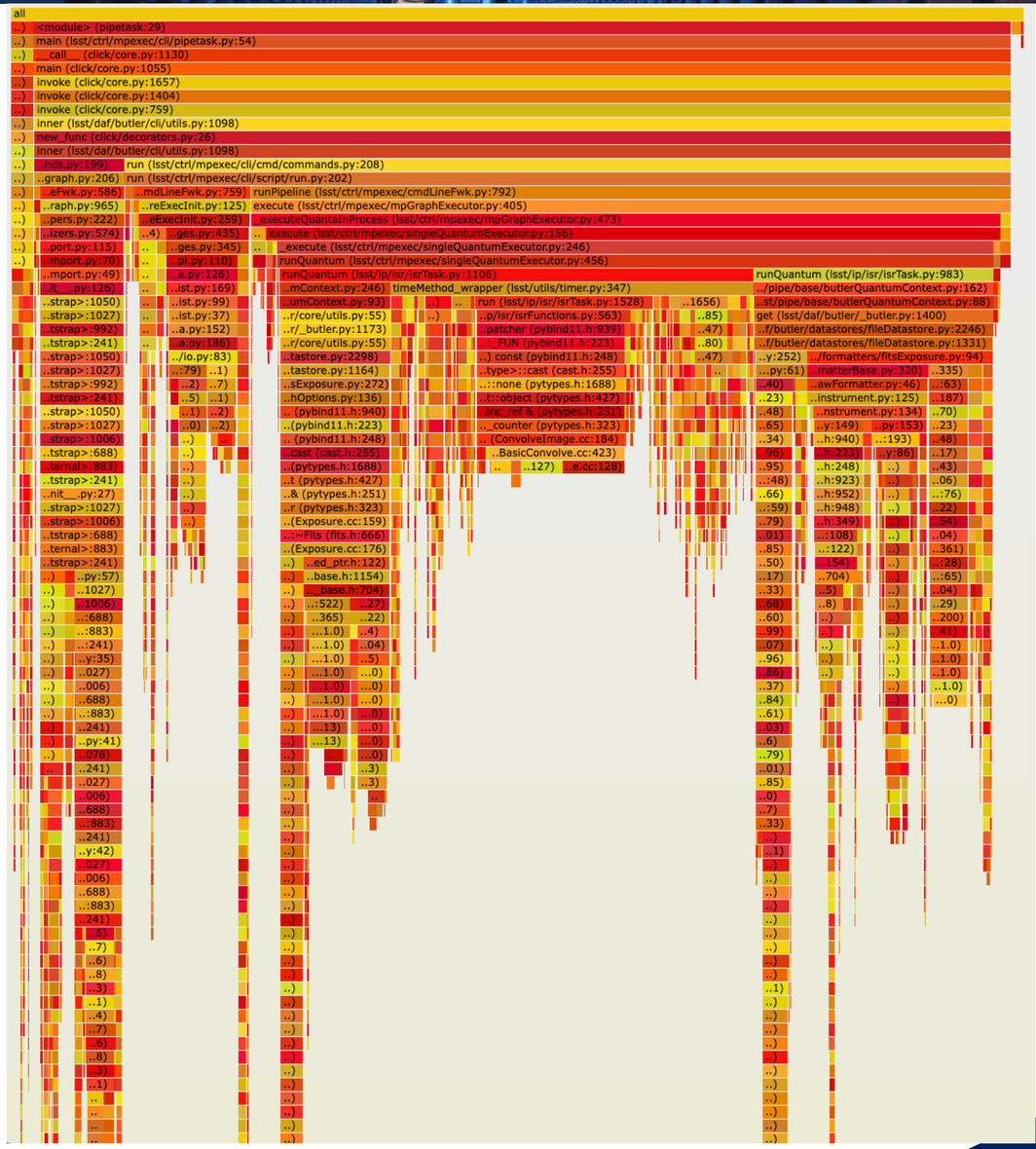
- cProfile : profilage CPU intégré à Python: `python -m cProfile myscript.py`
  - Nombre d'appels de chaque fonction et temps d'exécution
  - La sortie peut être exploitée par SnakeViz:



Profilage de la pipeline

- py-spy : `py-spy record -- python myscript.py`
  - Profilage CPU (fonctions)
  - Profilage des extensions C / C++
  - Génère un SVG de type "flamegraph"

- De nombreux autres outils disponibles :
  - Scalene
  - Austin
  - Pyinstrument
  - …



Profilage de la pipeline

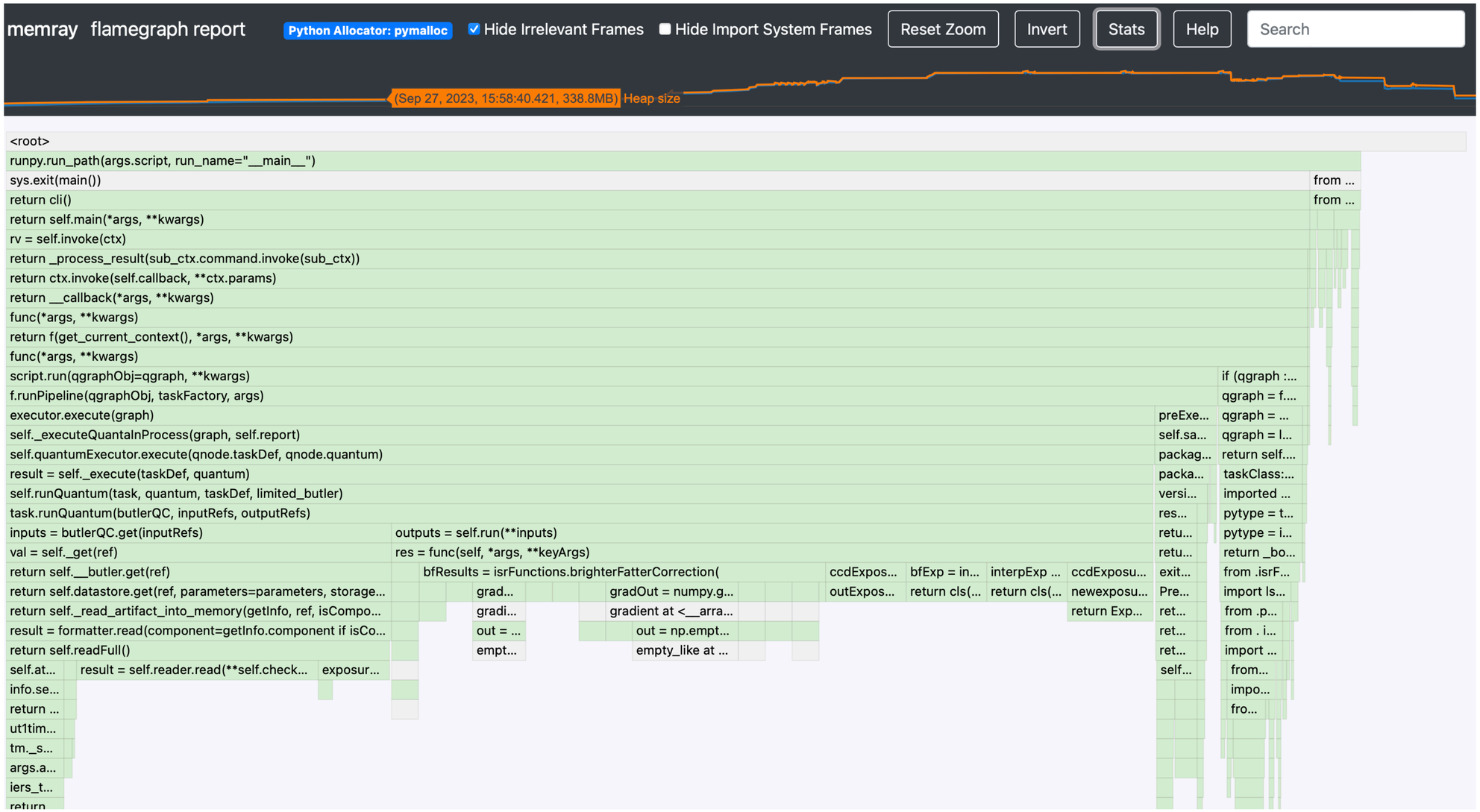
- Memray : profilage mémoire
  - Peut être exécuté directement en ligne de commande : `memray run myscript.py`
  - Ou directement dans le code Python :

```
import memray
with memray.Tracker("output.bin"):
```

  - Nombre d'allocations et quantité de mémoire pour chaque fonction
  - Profilage des extensions C / C++

Profilage de la pipeline

- Sortie "flamegraph" :



Profilage de la pipeline

- Sortie "stats" :

```
→ memray stats caracImage_oct27_full_nonative.bin
📏 Total allocations:
    61840023

📦 Total memory allocated:
    17.855GB

📊 Histogram of allocation size:
    min: 0.000B
    ──────────────────────────────────────────────────
    < 6.000B   :    56584 ▏
    < 36.000B  : 15149095 ▏
    < 224.000B : 41648132 ▏
    < 1.329KB  :  3470307 ▏
    < 8.077KB  :  1289892 ▏
    < 49.068KB :   193616 ▏
    < 298.075KB:    31804 ▏
    < 1.768MB  :      479 ▏
    < 10.741MB :       56 ▏
    ≤ 65.250MB :       58 ▏
    ──────────────────────────────────────────────────
    max: 65.250MB

📂 Allocator type distribution:
    MALLOC: 60200868
    POSIX_MEMALIGN: 726010
    CALLOC: 642234
    REALLOC: 200092
    ALIGNED_ALLOC: 70819

🥇 Top 5 largest allocating locations (by size):
    - measure: ... path ... /DarwinX86/meas_modelfit/g93c4d6e787+0bdc8f6279/python/lsst/meas/modelfit/cmodel/cmodelContinued.py:63 → 11.449GB
    - measure: ... path ... /DarwinX86/meas_base/g67924a670a+6c888dcbf9/python/lsst/meas/base/wrappers.py:43 → 1.369GB
    - _solve: ... path ... /DarwinX86/meas_extensions_gaap/g96f6979c0c+f64f51c0d6/python/lsst/meas/extensions/gaap/_gaussianizePsf.py:314 → 442.305MB
    - getNoiseGenerator: ... path ... /DarwinX86/meas_base/g67924a670a+6c888dcbf9/python/lsst/meas/base/noiseReplacer.py:366 → 384.006MB
    - cosmicRay: ... git ... /pipe_tasks/python/lsst/pipe/tasks/repair.py:183 → 384.006MB

🥇 Top 5 largest allocating locations (by number of allocations):
    - measure: ... path ... /DarwinX86/meas_modelfit/g93c4d6e787+0bdc8f6279/python/lsst/meas/modelfit/cmodel/cmodelContinued.py:63 → 50443948
    - measure: ... path ... /DarwinX86/meas_base/g67924a670a+6c888dcbf9/python/lsst/meas/base/wrappers.py:43 → 1802230
    - _gaussianizeAndMeasure: ... path ... /DarwinX86/meas_extensions_gaap/g96f6979c0c+f64f51c0d6/python/lsst/meas/extensions/gaap/_gaap.py:605 → 1074060
    - convolve: ... path ... /DarwinX86/meas_extensions_convolved/g42fff21dfb+02b235c229/python/lsst/meas/extensions/convolved/convolved.py:459 → 784602
    - selectSources: ... path ... /DarwinX86/meas_algorithms/g8494ff85c9+4cea07f1a2/python/lsst/meas/algorithms/objectSizeStarSelector.py:380 → 752856
```

Profilage de la pipeline

- Profilage ligne par ligne :
  - line_profiler (CPU) et memory_profiler (mémoire)
  - Plus précis mais plus complexe
  - Pour plus tard ?

Profilage de la pipeline

- Perf : outil Linux de mesure de performances CPU
  - Très puissant mais pas d'informations sur la partie Python
  - Exemple : mesure de la distribution des cycles CPU

```
foot
Samples: 6K of event 'cycles:P', Event count (approx.): 307606790201
  Children      Self  Command  Shared Object                                         Symbol
-   98.10%     0.00%  python   [unknown]                                             [.] 0xffffffffffffffff
   - 0xffffffffffffffff
      - 73.03% _PyEval_EvalFrameDefault
         - 71.41% do_call_core (inlined)
            - 70.93% PyObject_Call
                 _PyObject_Call (inlined)
               - _PyVectorcall_Call (inlined)
                  - 70.83% _PyFunction_Vectorcall
                       _PyEval_Vector (inlined)
                       _PyEval_EvalFrame (inlined)
                     - _PyEval_EvalFrameDefault
                        - 70.82% do_call_core (inlined)
                             PyObject_Call
                             _PyObject_Call (inlined)
                             _PyVectorcall_Call (inlined)
                             _PyFunction_Vectorcall
                             _PyEval_Vector (inlined)
                             _PyEval_EvalFrame (inlined)
                             _PyEval_EvalFrameDefault
                           - do_call_core (inlined)
                              - 70.80% PyObject_Call
                                   _PyObject_Call (inlined)
                                   _PyVectorcall_Call (inlined)
                                   _PyFunction_Vectorcall
                                   _PyEval_Vector (inlined)
                                   _PyEval_EvalFrame (inlined)
                                 - _PyEval_EvalFrameDefault
                                    - 68.52% do_call_core (inlined)
                                       - 67.08% PyObject_Call
                                          - _PyObject_Call (inlined)
                                             - 67.07% _PyVectorcall_Call (inlined)
                                                - 67.04% method_vectorcall
                                                     _PyObject_VectorcallTstate (inlined)
                                                     _PyFunction_Vectorcall (inlined)
                                                     _PyEval_Vector (inlined)
                                                     _PyEval_EvalFrame (inlined)
                                                     _PyEval_EvalFrameDefault
                                                     do_call_core (inlined)
                                                     PyObject_Call
                                                     _PyObject_Call (inlined)
                                                     _PyVectorcall_Call (inlined)
                                                     _PyFunction_Vectorcall
                                                     _PyEval_Vector (inlined)
                                                     _PyEval_EvalFrame (inlined)
                                                   - _PyEval_EvalFrameDefault
                                                      - 63.05% _PyObject_MakeTpCall
                                                         - 61.12% cfunction_call
                                                            - 58.56% pybind11::cpp_function::dispatcher(_object*, _object*, _object*)
                                                               - 52.86% void lsst::afw::math::detail::myoptConvolveWithBruteForce<lsst::afw::image::Image<float>, lsst::afw::image::Image<float> >(lsst::afw::image::Image<float>&, lsst::afw::image::Image<float> const&, lsst::afw::math::Kernel con
                                                                    1.22% boost::gil::detail::homogeneous_color_base<double, boost::gil::layout<boost::mp11::mp_list<boost::gil::gray_color_t>, boost::mp11::mp_list<std::integral_constant<int, 0> > >, 1>::operator double() const (inlined)
                                                                  + 0.68% lsst::afw::image::ImageBase<float>::x_at(int, int) const (inlined)
                                                               + 2.85% pybind11::cpp_function::initialize<lsst::afw::math::Statistics (*&)(lsst::afw::image::MaskedImage<float, int, float> const&, int, lsst::afw::math::StatisticsControl const&), lsst::afw::math::Statistics, lsst::afw::image::Ma
                                                               + 2.13% pybind11::cpp_function::initialize<lsst::afw::math::Statistics (*&)(lsst::afw::image::Image<float> const&, int, lsst::afw::math::StatisticsControl const&), lsst::afw::math::Statistics, lsst::afw::image::Image<float> const&,
                                                            + 2.05% array_implement_array_function
                                                         + 1.82% pybind11_meta_call
                                                      + 2.60% do_call_core (inlined)
                                                      + 0.63% array_assign_subscript
                                       + 1.11% _PyFunction_Vectorcall
                                    - 1.27% _PyObject_MakeTpCall
                                       + 1.00% bounded_lru_cache_wrapper
```

Résumé et perspectives

Résumé et perspectives

- CC-IN2P3 : 40% des traitements d'images de l'observatoire Vera C. Rubin
- Des besoins importants en ressources de calcul et stockage
- Une pipeline de traitement complexe
- Début d'un projet d'optimisation :
    - Prise en main d'une pipeline de test
    - Exploration des outils de profilage

- Perspectives :
    - Mise en place d'outils de profilage et d'analyse (sélection de métriques, dashboards)
    - Etude complète CPU + mémoire de quelques tâches
    - Analyse des parties du code consommatrices
    - Travail d'optimisation avec les équipes de développement