A^{2}S: Processing system for the rapid exploitation of satellite data streams on HPC platforms: some examples for solid earth research
Focus on MONITORING Issues

David Michéa - michea[@]unistra.fr
Ing. Scientific Calculation

Bernard Allenbach, Jean-Philippe Malet, Anne Puissant, André Stumpf

University of Strasbourg

A local HPC infrastructure for research applications on change detection techniques using S1/S2

Context (project start March 2016)

- ❑ Construct a dedicated High Performance Computing infrastructure for a **fully automated processing of S1/S2 data to detect changes** on 3 topics related to **science-driven applications**:
  - the quantification of earth surface movements *(e.g. landslides, reservoirs and anthropogenic hazards)*
  - the quantification of continental water surfaces (surface water reservoirs, flooding)
  - the quantification of urban changes

- ❑ Develop and **implement generic tools to classify and interpret changes in time series** using supervised (machine learning) and unsupervised approaches


deformation


water applications


urban applications

Dimensions of the infrastructure

- **Sensors:** Sentinel (S1/S2), Landsat (L7/L8) – (phase 1), *VHRS optical – (phase 2)*
- **Processing capacity in terms of spatial coverage**: 1000 x 1000 km (phase 1), *5000 x 5000 km (phase 2)*
- **Automatic processing in near-real time:** day+1 of data reception, processing finalized in less than 24hrs, on selected regions
- **On-demand processing**
- **Dissemination of the processed data**

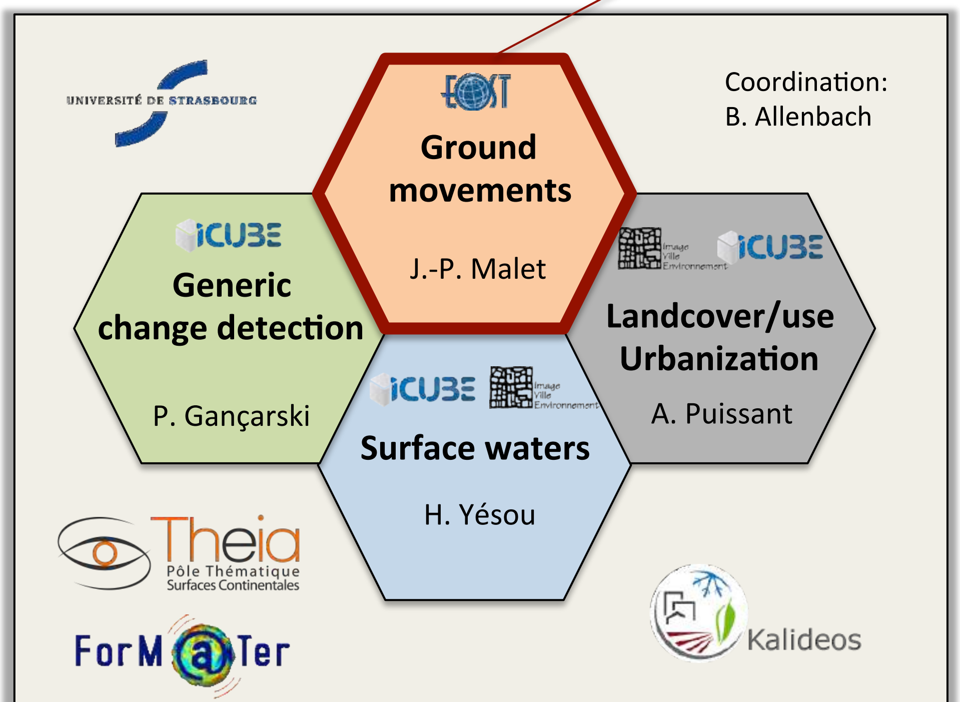
A²S: Environment

Funding

- State + Region: CPER (infrastructure)
- State: FNADT (IT engineer)

- Research projects:
  - CNES TOSCA, CNES R&D, ESA Alcantara: remote sensing algorithm development
  - ANR TIMES (start on 1/12/2017): change detection techniques (e.g. machine learning) developement

Structure



Ground Movements

→ *Optical data*

- Fine co-registation of stacks of S2/L8 time series – *CO-REGIS (EOST)*
- Mutiple Pairwise Image Correlation for landslide, volcano-tectonic and co-seismic slip displacement monitoring – *MPIC-OPT (EOST)*
- Detection of landslides from pre/post)event imagery – *ALADIM (EOST)*
- Generation of VHR DSM from Pléiades / Spot6-7 – *DSM-OPT (EOST)*

→ *SAR data*

- Landslides and subsidence monitoring from times series of wrapped, unwrapped and geocoded interferograms and time serie analysis – *NSBAS-S1 (ISTerre)*

A^{2}S: Specifications of the infrastructure

Processing capacity: Phase 1 - Operational

At Unistra Mesocentre : 20 nodes - 560 cores
1 node = 28 cores
50 nodes = 1400 cores (phase 2 / 2018)

High bandwidth dedicated IO cache system (17 To)

Priority calculation on A^{2}S nodes, and possibility to use additional distributed ressources

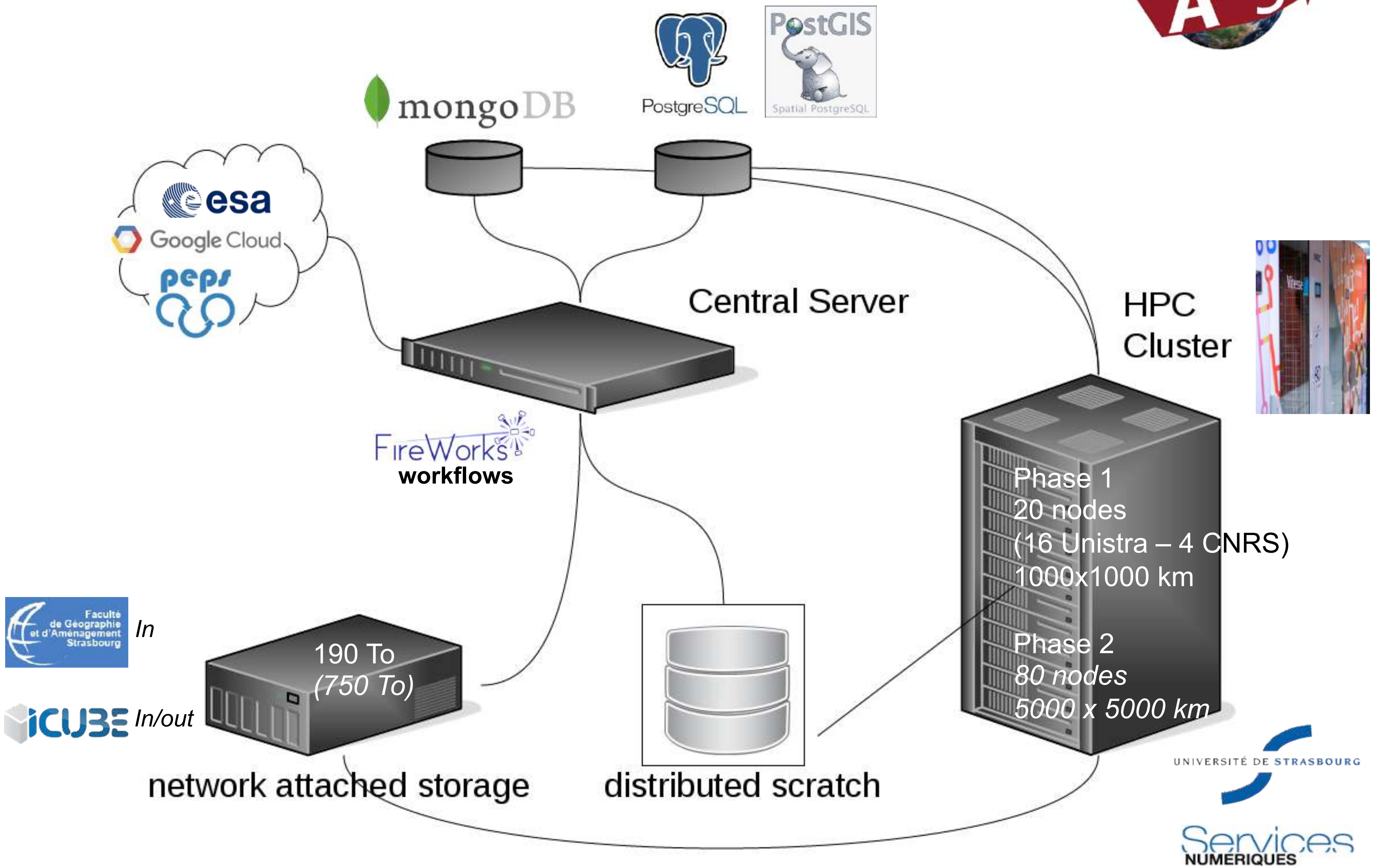
Storage: Phase 1 - Operational

High bandwidth network between HPC and storage: 10 GBits / s
Dedicated iRODS storage server 190 To (750 To – phase 2 / 2018)

Operation

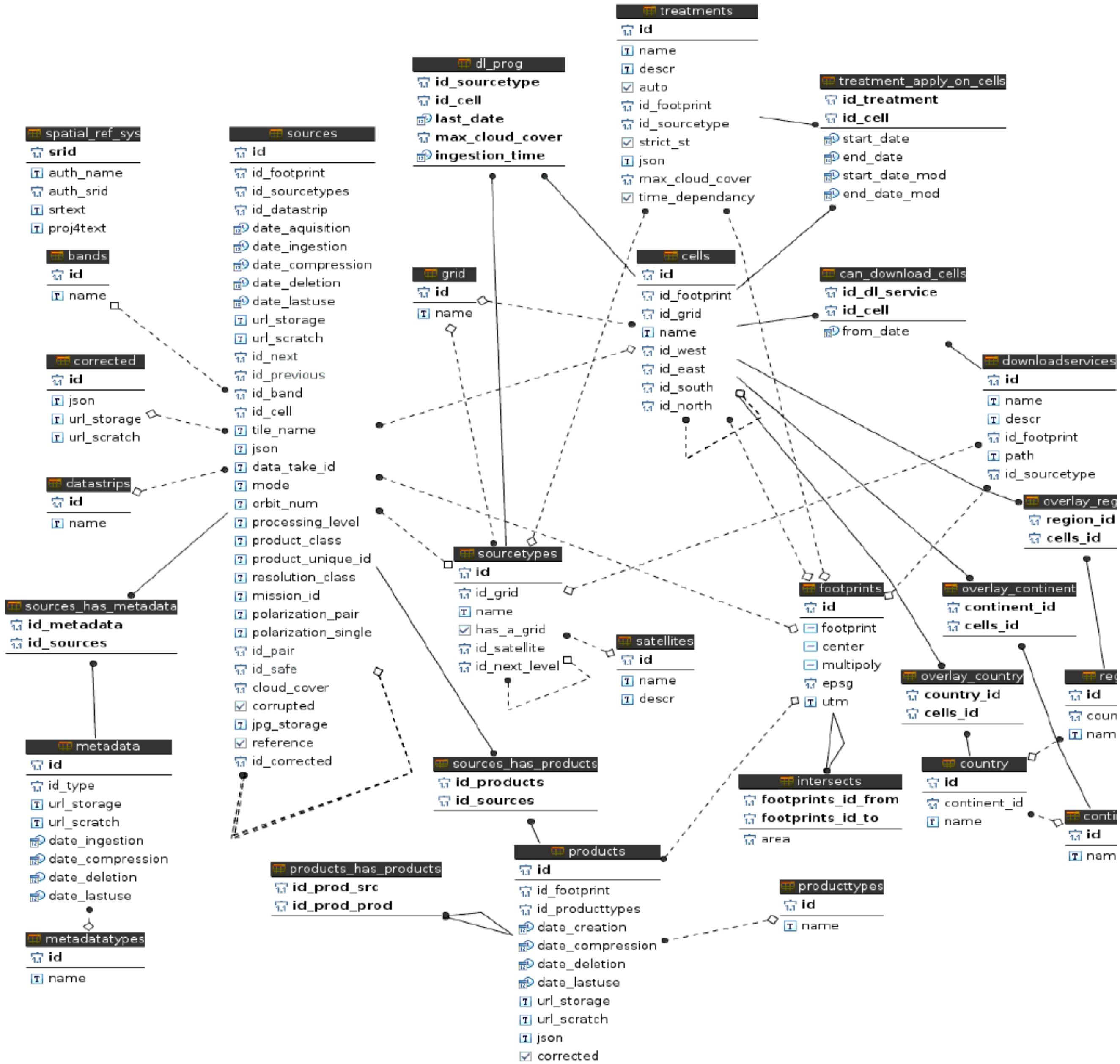
Processing nodes and storage capacity fully operationnal since April 2017
Current work: processing workflows developmment and evaluation of research results from 1st calculations

A²S: Specifications of the infrastructure



Construction of the data model

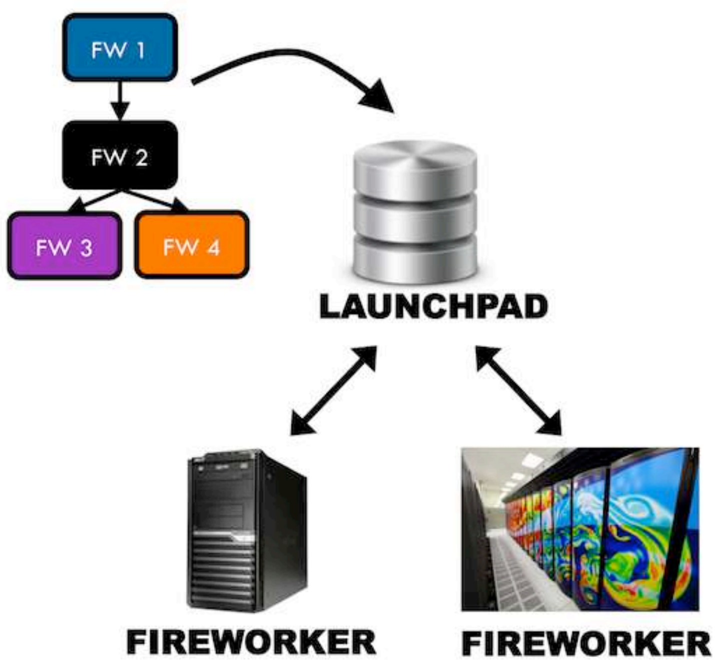
ALSACE AVAL SENTINELLE

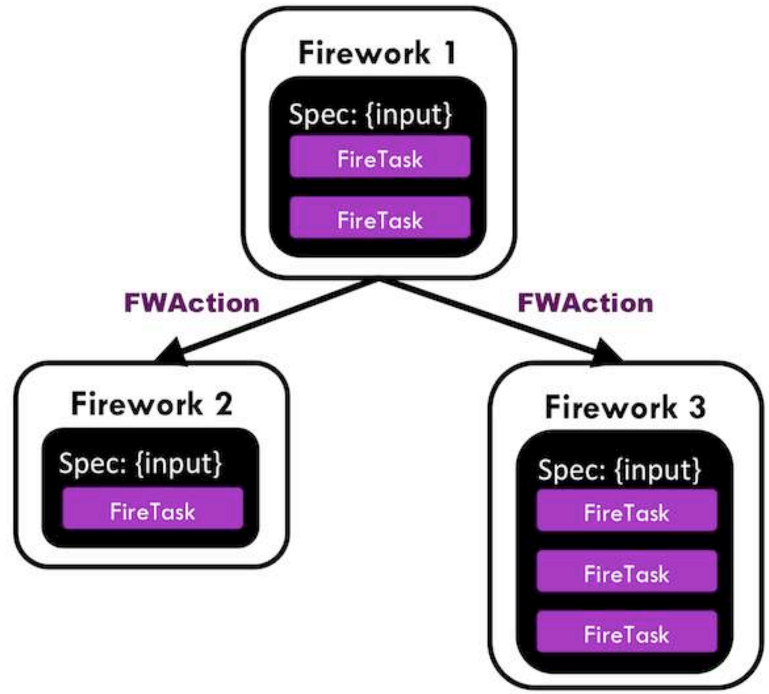


Development of the workflow management system - FireWorks

FireWork: free and open-source code for defining, managing, and executing workflows

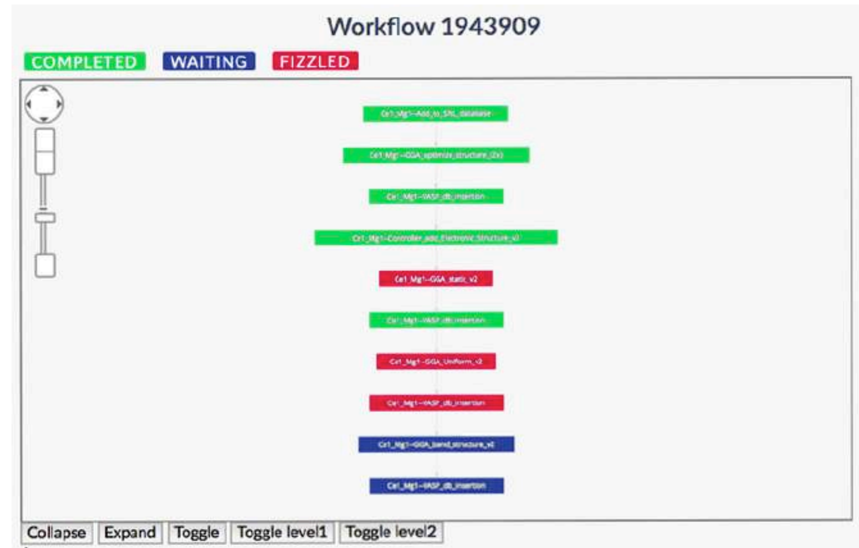
- Complex workflows are defined using Python, stored in a MongoDB instance, and can be monitored through a WEB GUI
- The workflow execution are automated on the computing resources, including those that have a queueing system





Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., and Persson, K. A. (2015). *FireWorks: a dynamic workflow system designed for high-throughput applications.* Concurrency Computat.: Pract. Exper., 27: 5037–5059. doi: 10.1002/cpe.3505

Workflows monitoring through a WEB GUI



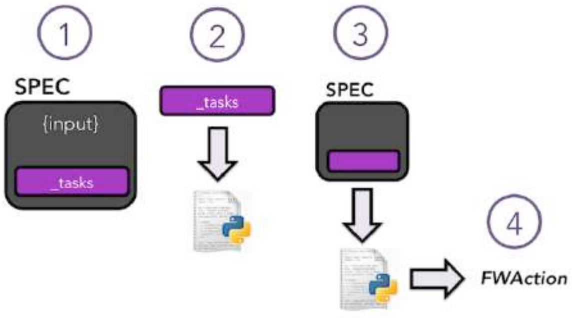
Workflows implementation: main objects of the FireWorks API

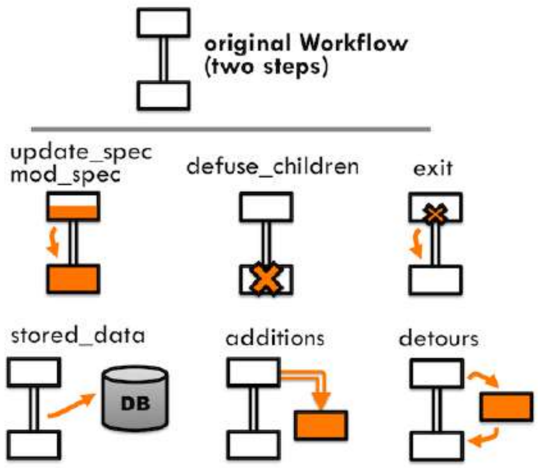
- **LaunchPad**: allow interaction with launchPad → query MongoDB : jobs, states, rocket launch etc.
- **Workflow**: list of Fireworks with dependencies
- **FireWork (FW)**

    - List of FireTasks
    - List of attributes
        - parents: → define the dependencies between FWs
        - spec: → allow defining priority or category ... for the FW

          → Each Firetask in the FW can modify Spec.

          → it provides a way to embed data in the Firework





- **Firetask**: sequential runs in a single FW
    - ScriptTask: run an external command/program
    - FileTransferTask: Helper task allowing file transfer (through ssh)
    - PyTask: run the Python functions passed as arguments
    - DIY : you can write your own Firetask …
- **FW Action**

  Object returned by a Firetask. It allows controlling / modifying the workflow by using a computer programm (additions, defuse_children, detours …) and to act on specific data

How a workflow is generated dynamically?

Start: a unique entry point (e.g. python script)

- 1st step: for each thematic processing → Database query:
  - Where does the thematic processing apply? *(e.g. footprints or tiles list)*
  - What are the source types *(e.g. S1, S2)*
  - What is the period of interest *(e.g. from / to dates)*
  - Are there specifici criteria? *(e.g. max_cloud_cover, geometric_quality_flag, etc)*
  - What is the last imported image *(e.g. date)*
  - Are there already-scheduled-but-not-imported image
  - Who is the data provider *(e.g. ESA Open Hub, GoogleCloud, PEPS …)*

  → For each provider: image/tile list, start date, criteria …

- 2nd step: catalog queries:
  - Tile list
  - Start date
  - Criteria

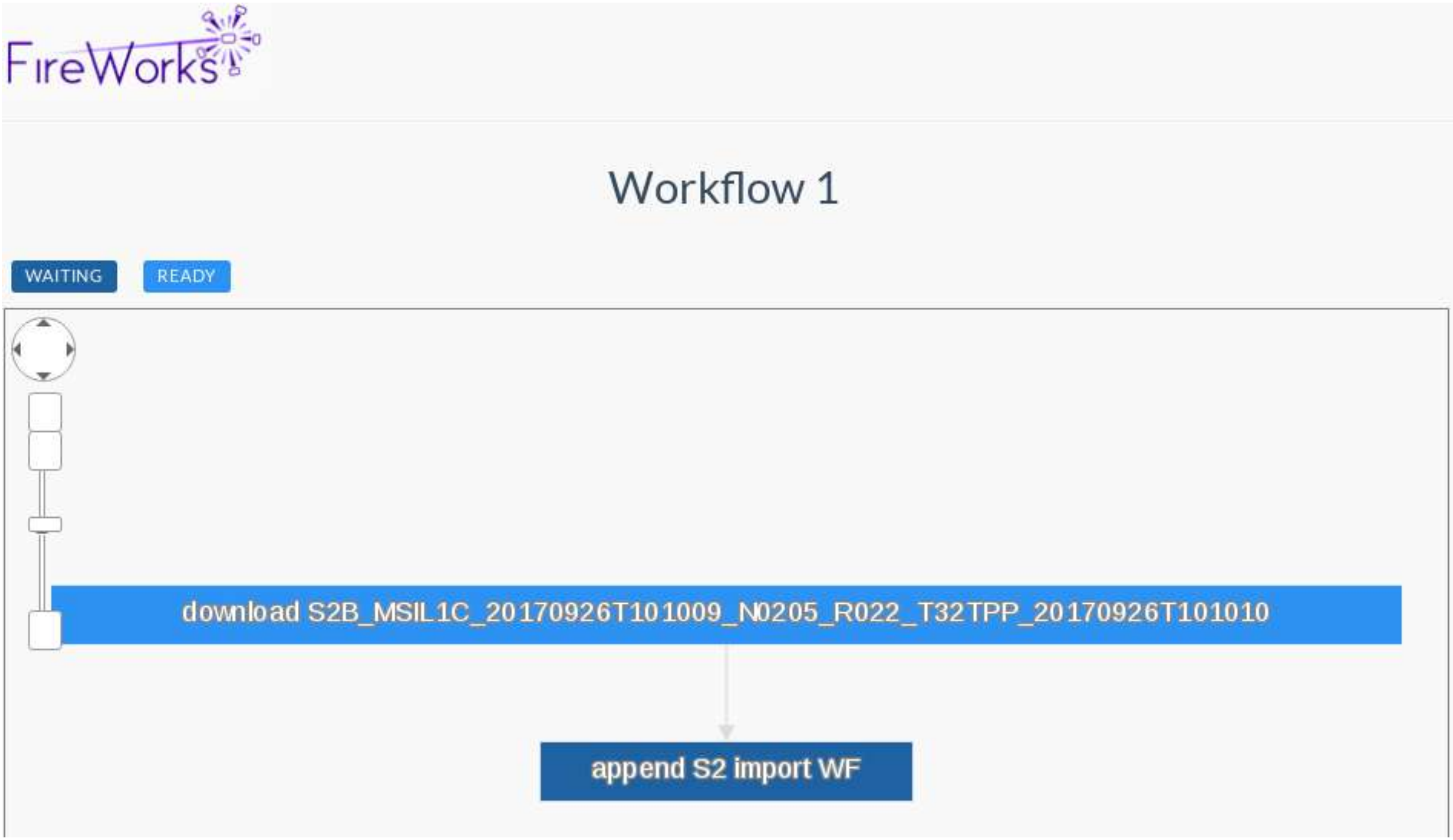
  → List of products to download

- 3rd step: for each image / tile:
  - Sort list of product to download by ascending dates
  - Store images/tiles in DB (*Already-scheduled-but-not-imported*)
  - Build a data structure holding all information for the next tiles/images to download
  - Create the workflow in charge of the first image download embedding information to dynamically create the workflow in charge of the next image download etc.…

How a workflow is generated dynamically?

```
next_dls.append({"provider" : peps",
             "url": url,
             "log_file": None,
             "fname": fname,
             "output_dir": output_dir,
             "dl_dir": dl_dir,
             "priority": priority,
             "SAFE_format": SAFE_format,
             "prod_id": src['filename'][0:-5],
             "corrupted": corrupted,
             "max_cloud_cover": max_cloud_cover
            })

download_fw = Firework(PyTask(func='a2s_entry_point.dl_from_theia',
                        args=[url, fname, output_dir]),
                  name="download %s" % fname,
                  spec={"_category": "seq_long", "_priority" : priority, "next_dls" : next_dls})

append_import_fw = Firework( a2s_tasks.AppendImportS2Task(dl_dir = dl_dir,
                                        SAFE_format = SAFE_format,
                                        priority = priority,
                                        corrupted = corrupted,
                                        max_cloud_cover = max_cloud_cover,
                                        is_ref = is_ref_image),
                        name="append_S2_import_WF",
                        parents=[download_fw],
                        spec={"_category": "seq_short",
                             "_priority" : priority,
                             "next_dls" : next_dls})

dl_workflow_same_tile = Workflow([download_fw, append_import_fw], name="Download of %s" % fname)
launchpad = LaunchPad.auto_load()
launchpad.add_wf(dl_workflow_same_tile)
```

Example of a dynamic workflow



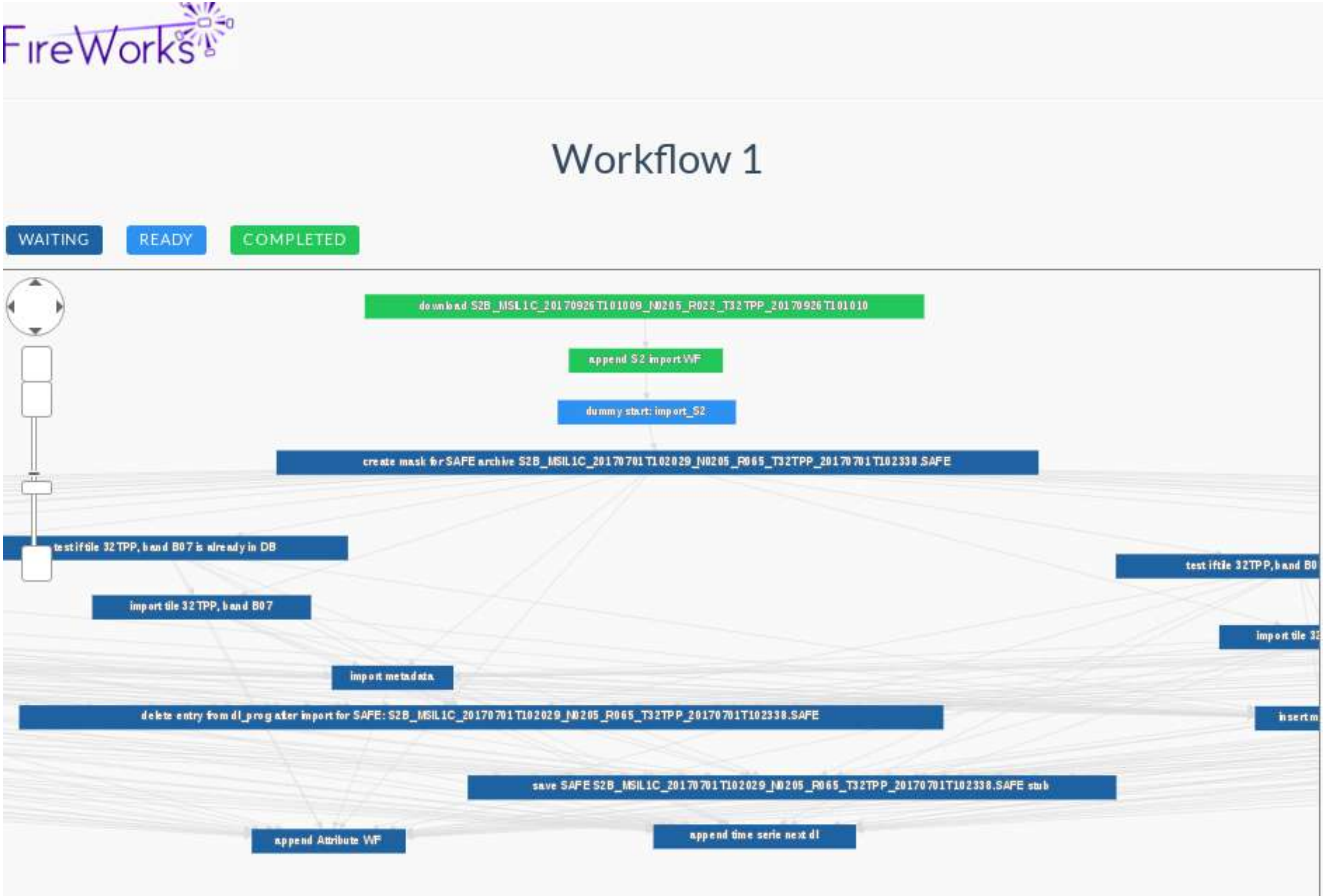
Example of a dynamic workflow

```
class AppendImportS2Task(FireTaskBase):

    _fw_name = 'AppendImportS2Task'
    required_params = ["dl_dir", "SAFE_format", "priority", "corrupted", "is_ref"]

    def run_task(self, fw_spec):
        (A2S_logger, __) = A2S_config.init_loggers()

        dl_dir = self["dl_dir"]
        SAFE_format = self["SAFE_format"]
        priority = self["priority"]
        corrupted = self['corrupted']
        max_cloud_cover = self['max_cloud_cover']
        is_ref = self['is_ref']

        next_dls_params = fw_spec["next_dls_params"]

        import_S2_wf = import_S2.get_wf(dl_dir, SAFE_format, priority, corrupted, max_cloud_cover, is_ref, next_dls_params)
        if import_S2_wf:
            A2S_logger.info("create & append S2 import workflow")
            return FWAction(additions=import_S2_wf)
        else:
            A2S_logger.fatal("import_S2.get_wf failed : no S2 import workflow created for SAFE archive %s" % dl_dir)

        return FWAction()
```

Example of a dynamic workflow

Example of a dynamic workflow

Example of a dynamic workflow



Workflow runs on A^{2}S HPC: qadapters & FW categories

- Interface with the A^{2}S HPC queueing system: queue adapters
- For Fireworks, a job is a job. Parallel, sequential, long, short, GPGPU = a job.
  → use of different categories of queue adapters.
  → a queue adapter only ask the launchpad for a certain category of FW

Example of a queue adapter:

```
_fw_name: CommonAdapter
_fw_q_type: SLURM
rocket_launch: mlaunch -w my_fworker.yaml -l my_launchpad.yaml 28 --nlaunches=infinite --sleep 20
Nodes: 1
Ntasks: 28
ntasks_per_node: 28
Walltime: '08:00:00'
queue: grant2
account: grant2ipgs
job_name: multiseq
logdir: /b/home/ipgs/dmichea/A2S/logs/fw_logs
pre_rocket: null
post_rocket: null
exclude: hpc-n[523-564,585-590]
```

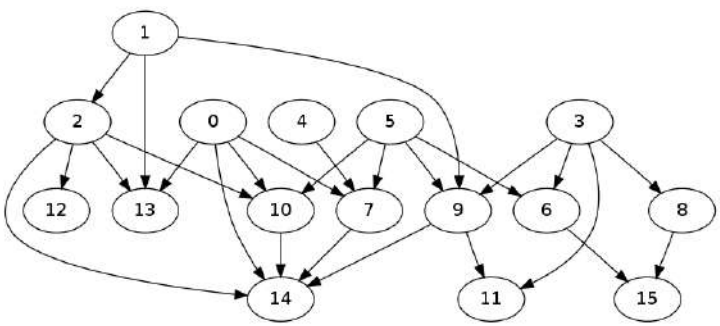
Submission of a queue adapter:

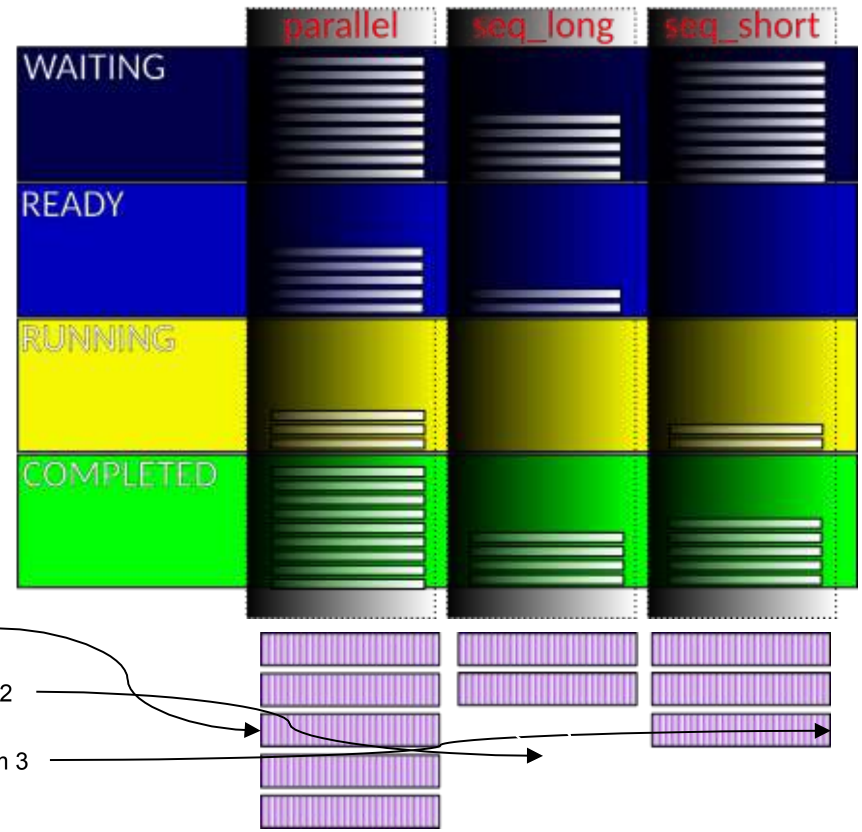
```
qlaunch -fm rapidfire -m 6 --sleep 60 --nlaunches "infinite"
```

Workflow runs on A^2S HPC: qadapters & FW categories

Distributed scheduling by categories of FW

- parallel: parallel / 28 cores (shm)
- seq_long: sequential, long
- seq_short: sequential, short





qlaunch -fm -w my_fworker_parallel.yaml -q my_qadapter_parallel.yaml rapidfire -m 5

qlaunch -fm -w my_fworker_seq_long.yaml -q my_qadapter_seq_long.yaml rapidfire -m 2

qlaunch -fm -w my_fworker_seq_short.yaml -q my_qadapter_seq_short.yaml rapidfire -m 3

→ The process dedicates nodes to the execution of a certain category of jobs

→ The computation needs per FW category can vary a lot in terms of workflow execution

→ There is a delay (sometimes > 1 day) between submission and start of a SLURM job

→ How to dynamically adapt ressources to the needs and avoid nodes starvation?

→ Solution: write a rocket scheduler

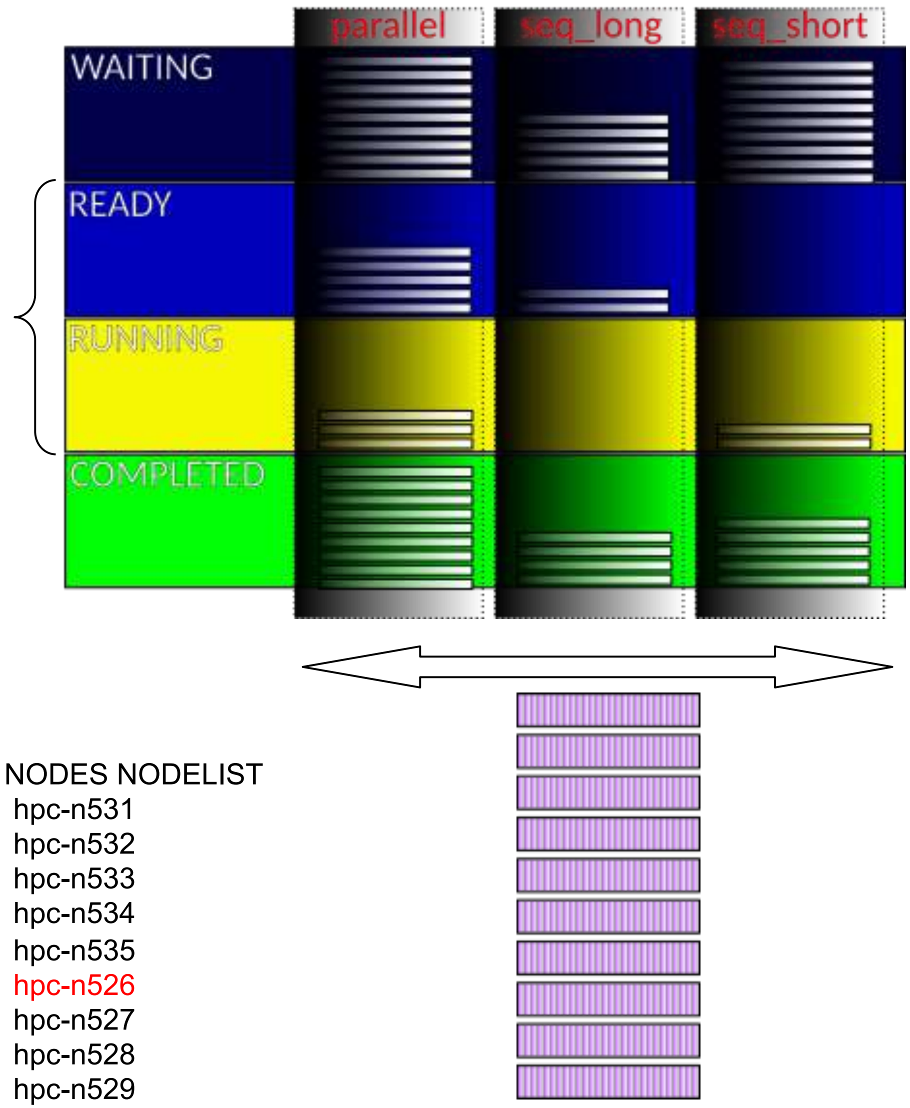
Workflow runs on A^{2}S HPC: rocket scheduler

Categories of FW loads:

- parallel: 28 cores / FW
- seq_long: 1 core / FW
- seq_short: 1 core / FW

ready fws load +
running fws load →
global_capacity_needed

nb_running_jobs *
node_load_capacity
→ global_capacity



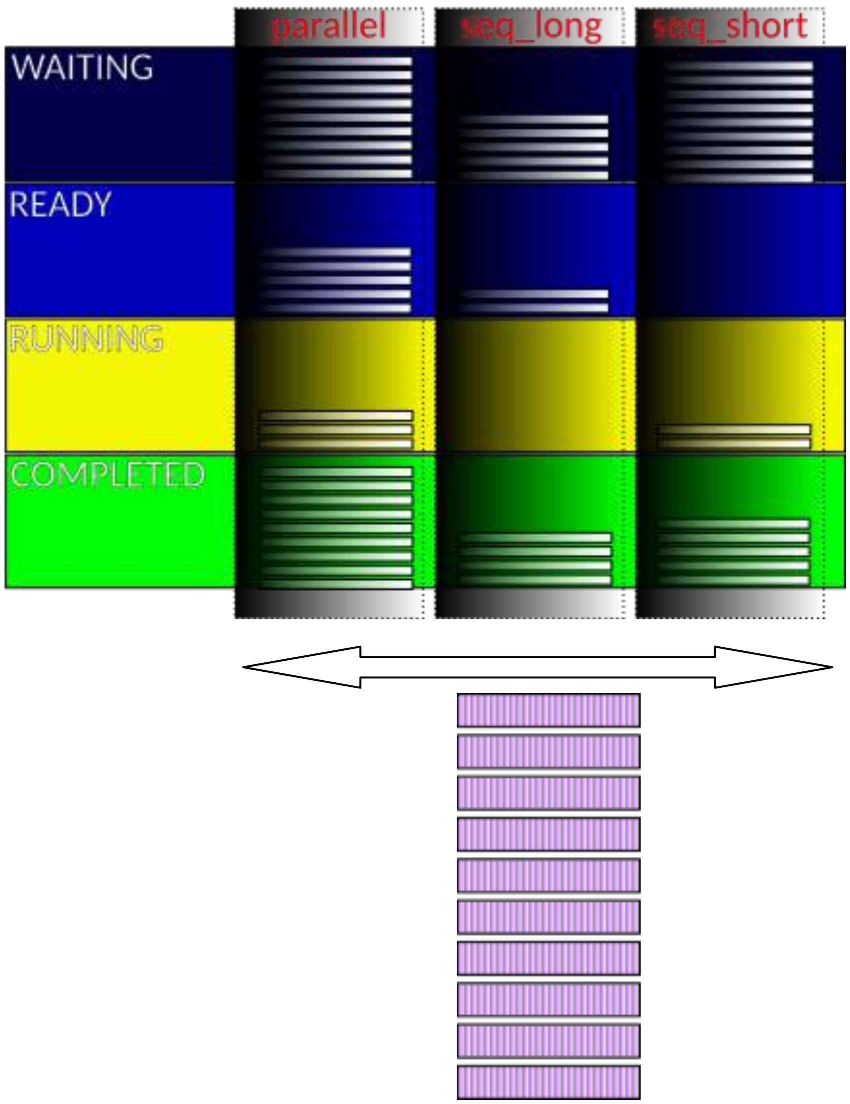
JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST
3213012	grant2	rockets	dmichea	R	6:12:40	1	hpc-n531
3213013	grant2	rockets	dmichea	R	6:12:40	1	hpc-n532
3213014	grant2	rockets	dmichea	R	6:12:40	1	hpc-n533
3213015	grant2	rockets	dmichea	R	6:12:40	1	hpc-n534
3213016	grant2	rockets	dmichea	R	6:12:40	1	hpc-n535
3213006	grant2	rockets	dmichea	R	6:20:09	1	hpc-n526
3213007	grant2	rockets	dmichea	R	6:20:09	1	hpc-n527
3213008	grant2	rockets	dmichea	R	6:20:09	1	hpc-n528
3213009	grant2	rockets	dmichea	R	6:20:09	1	hpc-n529
3213010	grant2	rockets	dmichea	R	6:20:09	1	hpc-n530

Workflow runs on A²S HPC: rocket scheduler

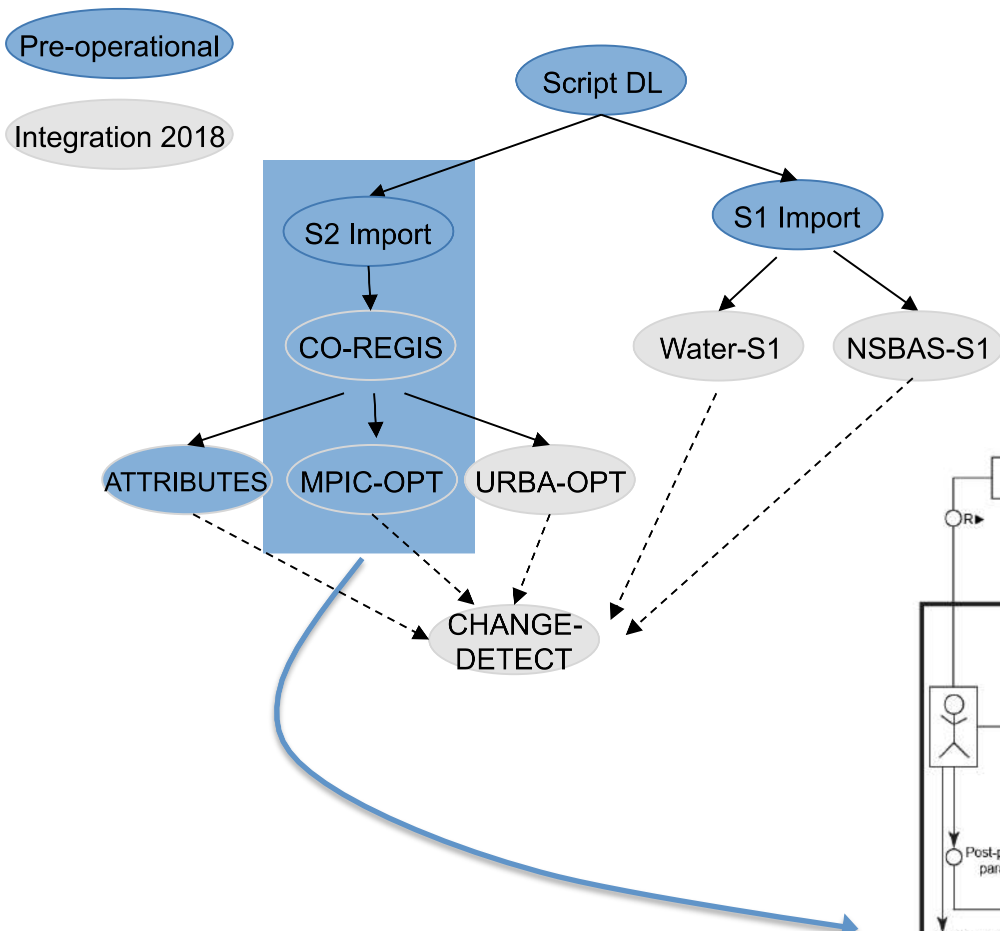
After the workflow creation, one rocket_scheduler instance is submitted to the queue:

sbat ch rocket_scheduler.slurm
→ execute rocket_scheduler.py

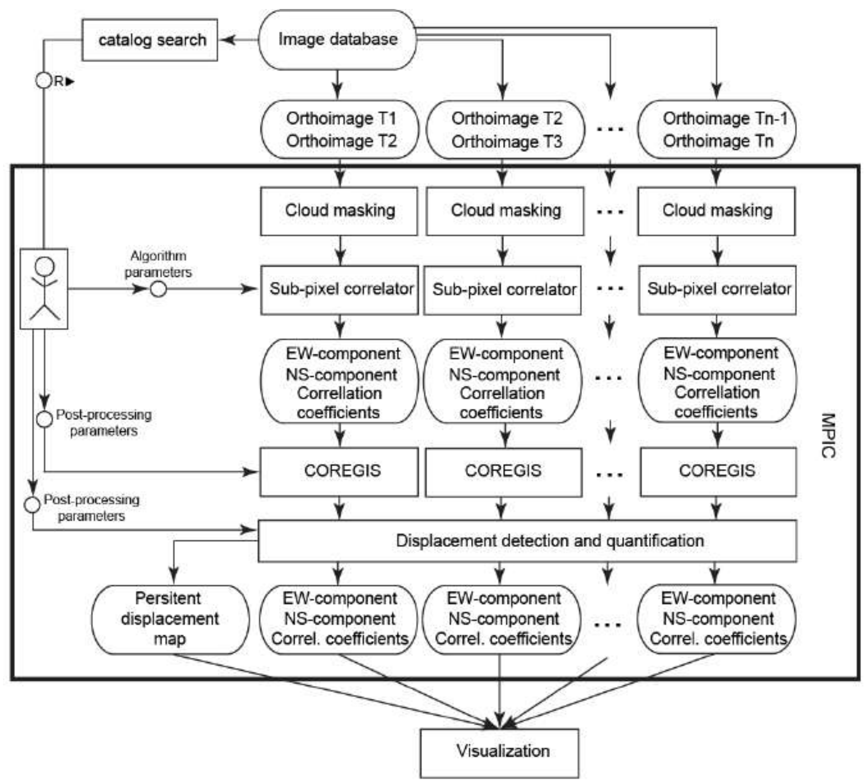
```
Infinite loop:
    get_slurm_queue_infos()
    get_fws_infos()
    while not fully_loaded:
        for each category:
            launch_rocket(category)
            recompute_load()
            if fully_loaded:
                break
    if master:
        if global_capacity < global_needs:

    submit_scheduler_instance()
        if no_more_fws_to_run:
            stop_all_jobs()
            return
    sleep(few seconds)
```



Thematic processing: progress of work

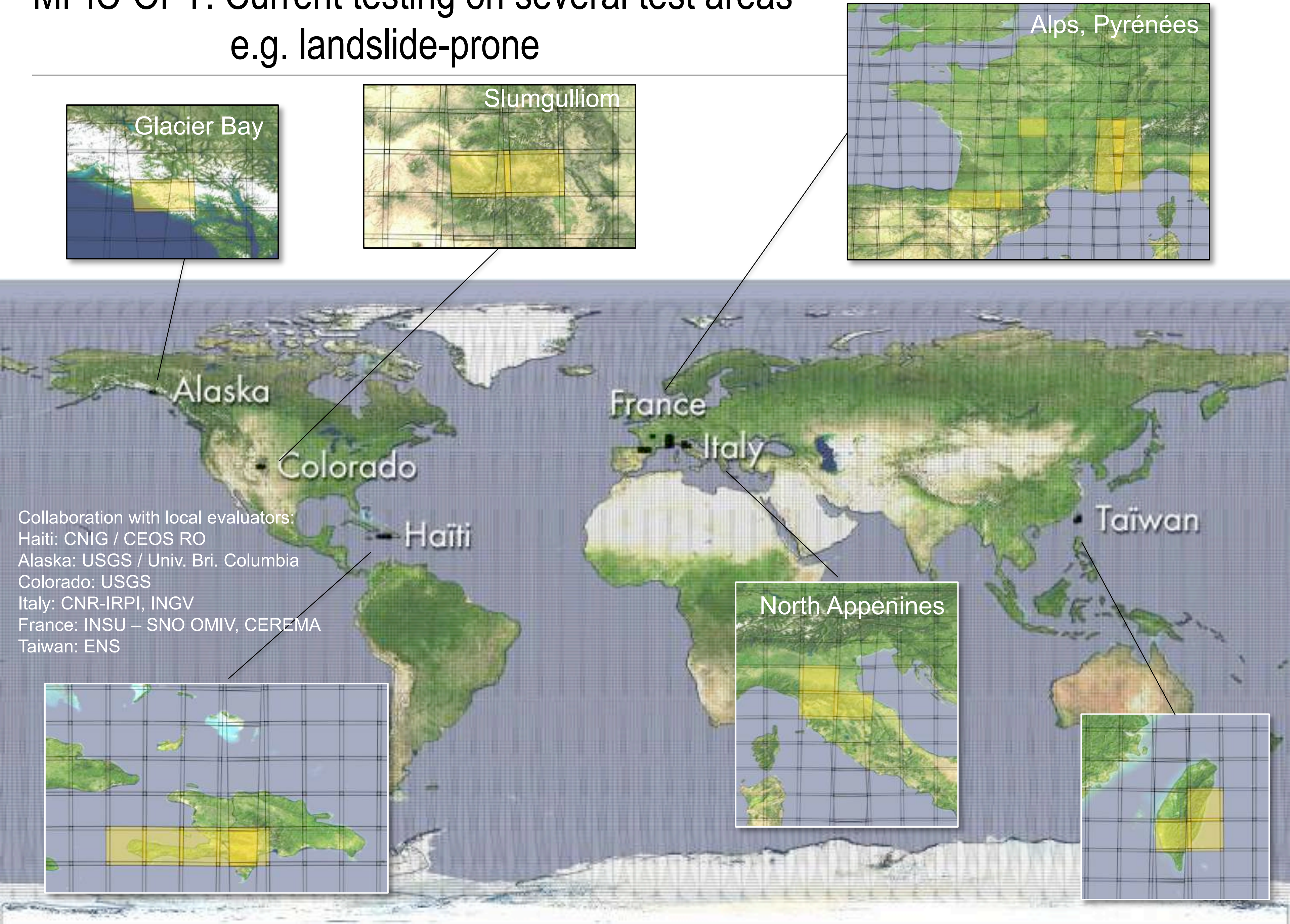


MPIC-OPT / Stumpf et al.



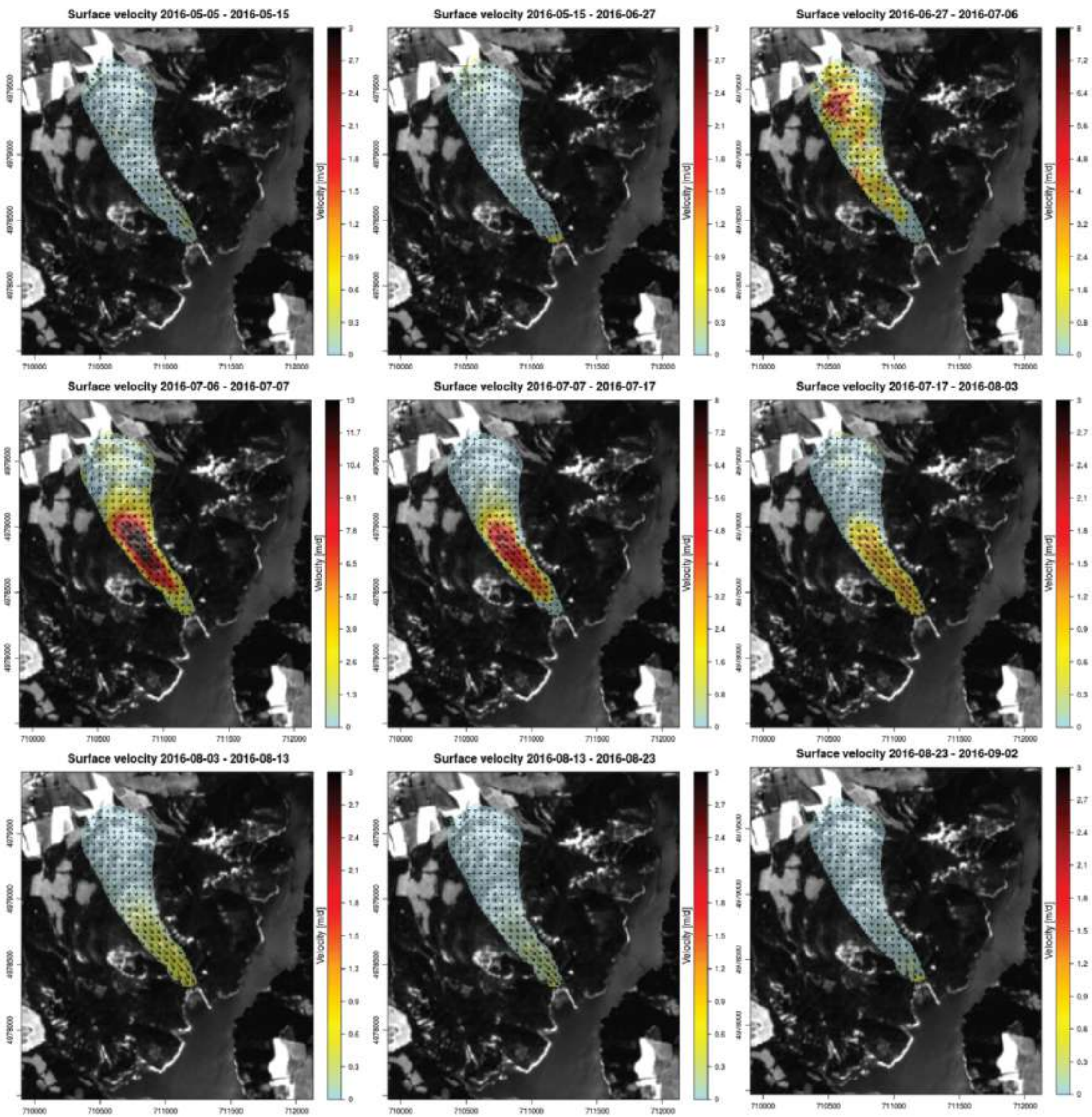
EOST	Script DL, CO-REGIS, MPIC-OPT
Image Ville Environnement	ATTRIBUTES, URBA-OPT, Water-S1
ISTerre	NSBAS

MPIC-OPT: Current testing on several test areas e.g. landslide-prone



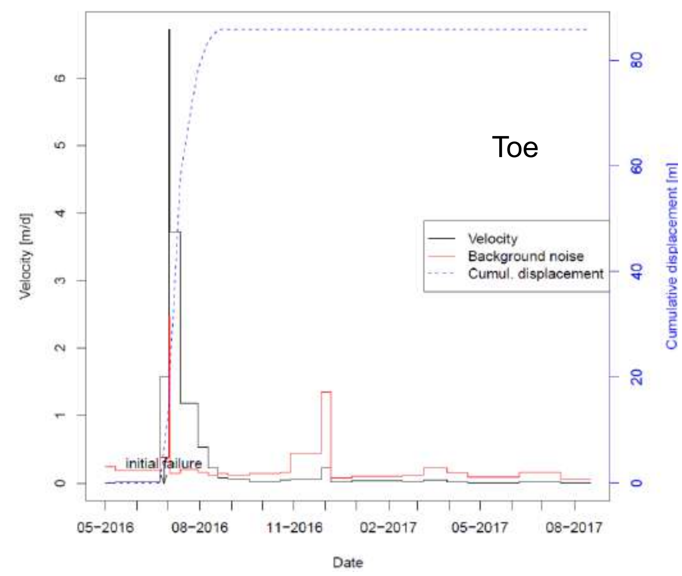
Example of results: MPIC-OPT

Landslide monitoring application



Surface displacement of Harmalière landslide

- 31 S2 images
- 1 L8 image
- 180 correlograms – 22 used for the analysis

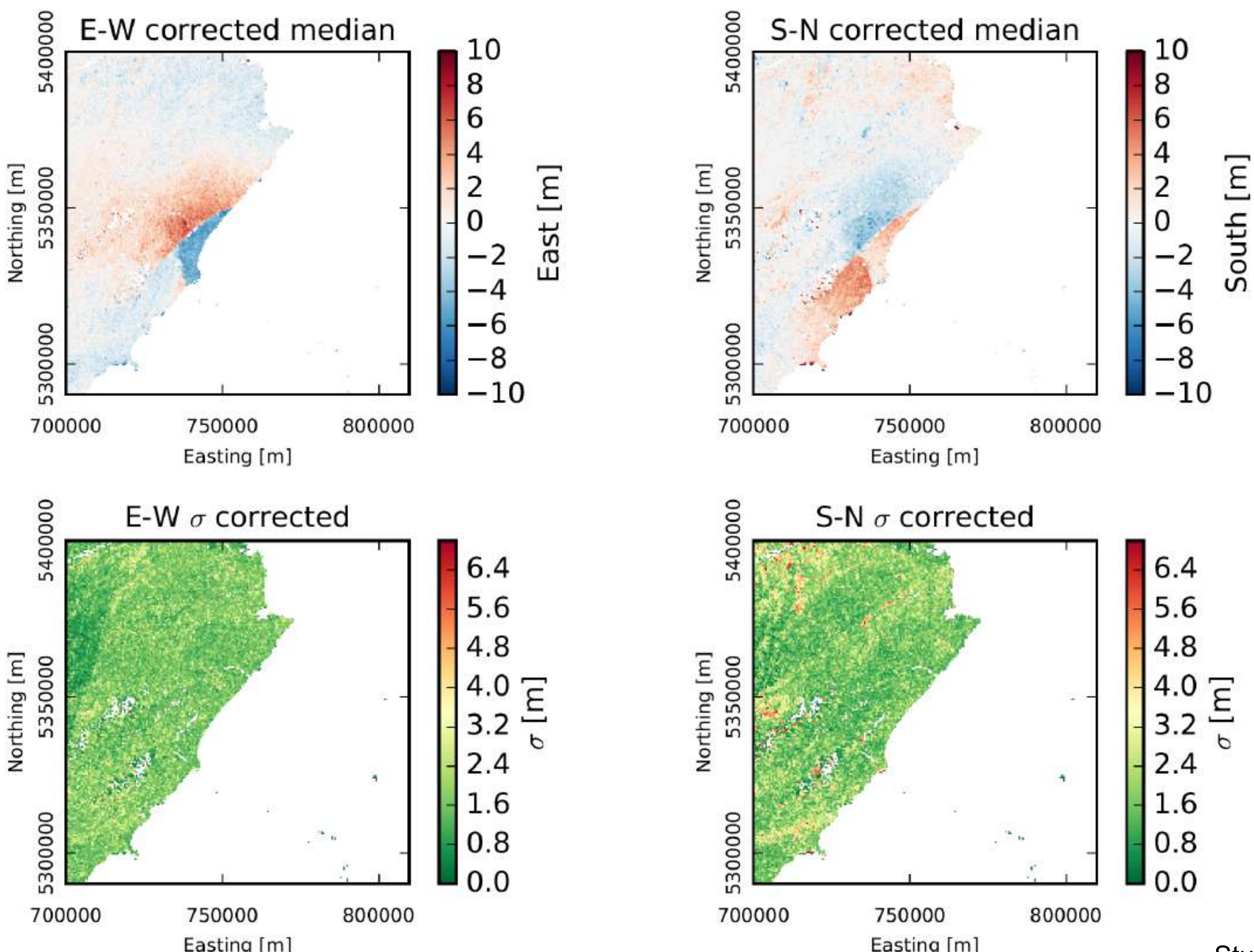


Stumpf et al. (2017)

Example of results: MPIC-OPT

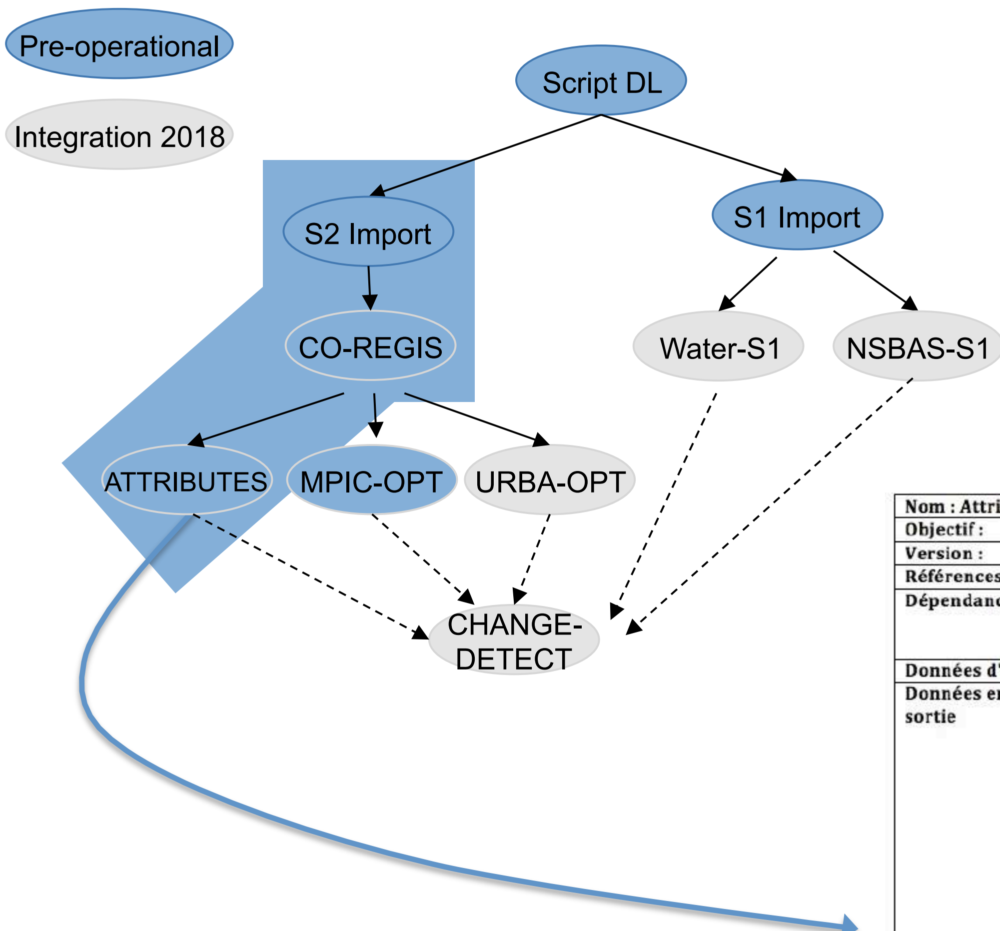
Co-seismic slip / Kaikoura ETQ
Pre / Post event
- 4 x S2 (pre ; 5 x S2 – after)
- 40 correlograms

Tectonic application



Stumpf et al. (2017)

Thematic processing: progress of work

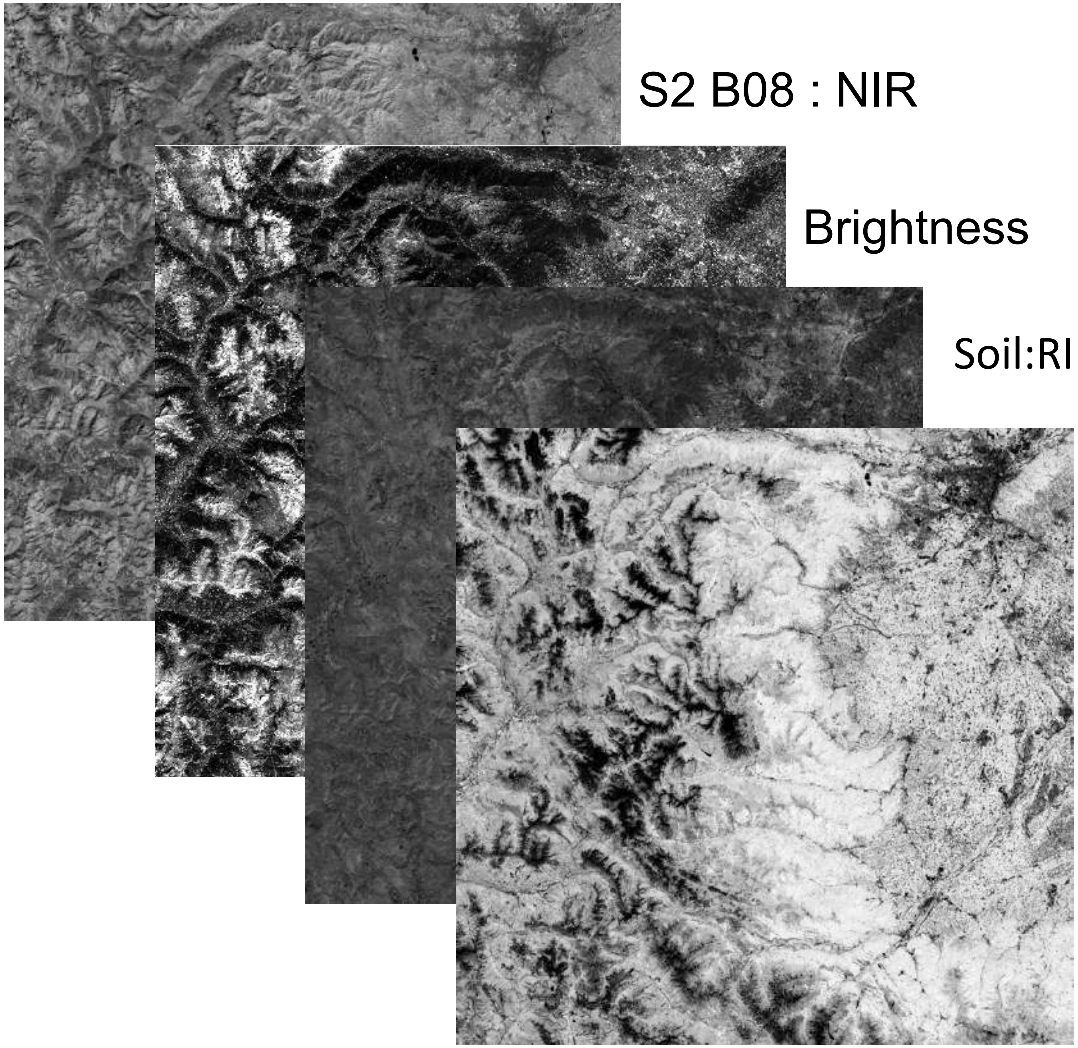


Nom : Attributs-S2				
Objectif :	Calcul d'attributs sur les images S2 (pour l'instant 40 attributs sont calculés)			
Version :	0.1			
Références :	Puissant (CNRS/LIVE)			
Dépendances :	python3 ≥ 3.5	PostgreSQL 9.2.18	FireWorks 1.4.1	OrpheoToolBox (OTB-5.10.1)
	logging (py)	glob (py)	psycopg2 (py)	subprocess (py)
Données d'entrée	img S2 coregistrée	DB connection info	config file	
Données en sortie	spectral_indices	Vegetation:GEMI, Vegetation:MSAVI, Vegetation:NDVI, Vegetation:TSAVI, Water:NDWI2, Soil:BI, Soil:BI2, Soil:CI, Soil:RI		
	Haralick_indices	Simple: Energy, Entropy, Correlation, InverseDifferenceMoment, Inertia, ClusterShade, ClusterProminence, HaralickCorrelation		
		Advanced: Mean, Variance, Dissimilarity, SumAverage, SumVariance, SumEntropy, DifferenceOfEntropies, DifferenceOfVariances, IC1, IC2		
		Complex: ShortRunEmphasis LongRunEmphasis, GreyLevelNonuniformity, RunLengthNonuniformity, RunPercentage, LowGreyLevelRunEmphasis, HighGreyLevelRunEmphasis ShortRunLowGreyLevelEmphasis, ShortRunHighGreyLevelEmphasis, LongRunLowGreyLevelEmphasis, LongRunHighGreyLevelEmphasis		
Scénario d'utilisation	En continu			
Emprise de calcul	Emprise initiale prévue : France Métropolitaine			
Algorithme aval	URBA-OPT, CHANGE-DETECT			
Licence	CNRS/LIVE (Puissant et al.)			

Puissant et al. (2017)

EOST	Script DL, CO-REGIS, MPIC-OPT
Image Ville Environnement	ATTRIBUTES, URBA-OPT, Water-S1
ISTerre	NSBAS

Thematic processing: progress of work



S2 B08 : NIR

Brightness

Soil:RI

Vegetation:NDVI

Current testing on Grand Est Region

Puissant et al. (2017)

Conclusions - A^{2}S short-term roadmap for 2018

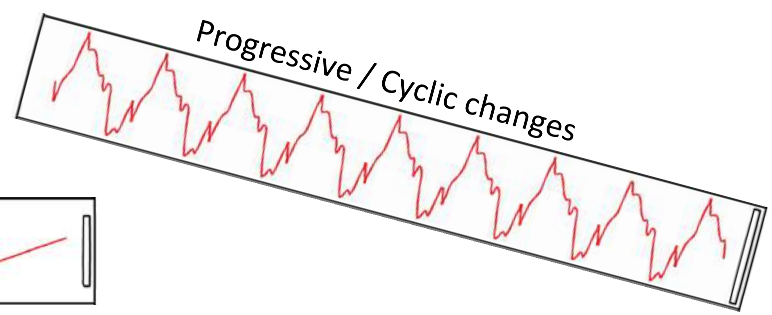
- **Software system**
  - Debugging and evaluation of current processing on the test areas
  - Integration of NSBAS-S1 processing (collab. ISTerre)
  - Implement simple systems for data dissemination

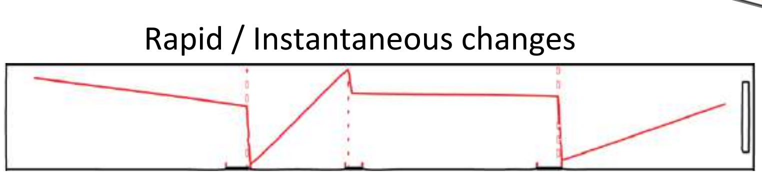
- **Infrastructure**: 2nd phase integration (e.g. 80 nodes, 500 To data storage)

- **Research project**

  ANR TIMES (start date: 1/12/2017): *High-performance processing techniques for mapping and monitoring environmental changes from massive, heterogeneous and high frequency data time series*

  - *Topic: Big Data and Knowledge*
  - *Consortium: LIVE / EOST / LIPADE / MIPS / MONASH / ICUBE*
  - *Objectives: develop generic machine learning techniques to detect and quantify changes in heterogeneous time series*





Financement
CNRS / LIVE
Financement
Unistra / CPER