A²S: Challenges in the automated processing of massive satellite data streams on HPC.

A quick look under the hood

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

Bernard Allenbach, Aline Deprez, Jean-Philippe Malet, Sina Nakhostin, Anne Puissant, André Stumpf

University of Strasbourg

A²S: Presentation

Our Expertise

- Image processing
- Massive computation
- Time Serie Analysis
- Satellite

Our Services

- Design
- Compute
- Disseminate
- Collaborate

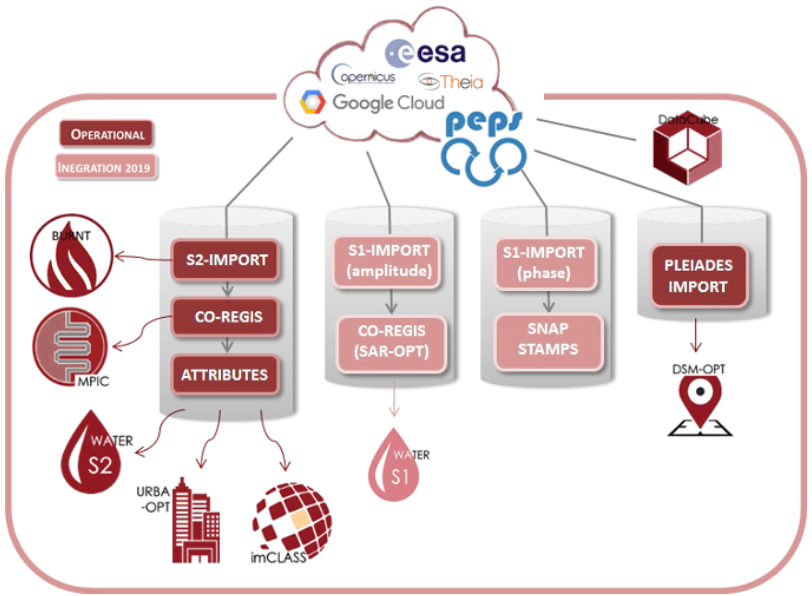
A²S: Automating processing on HPC system

HPC : High Performance Computing

Non extensible ressources
Big nodes (RAM, cores)
Close to metal
Fast buses / low latency networks
Efficient use of computational ressources

A2S : Applications for Satelitte Survey

Run automatically and efficiently a lot of different Services on an HPC architecture.
-> **scheduling problem**
-> **manage complexity** (several thousands of tasks)

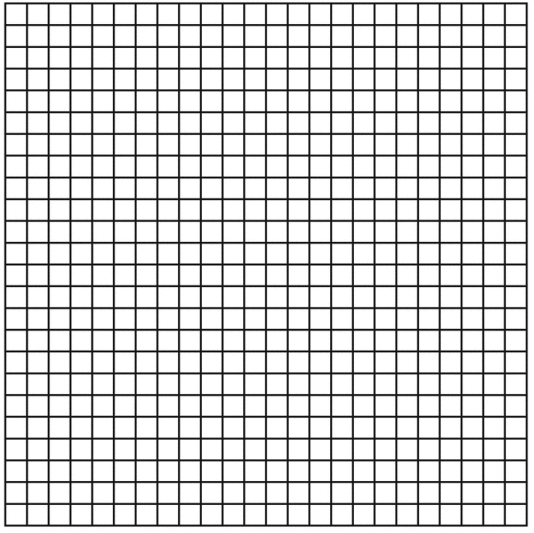


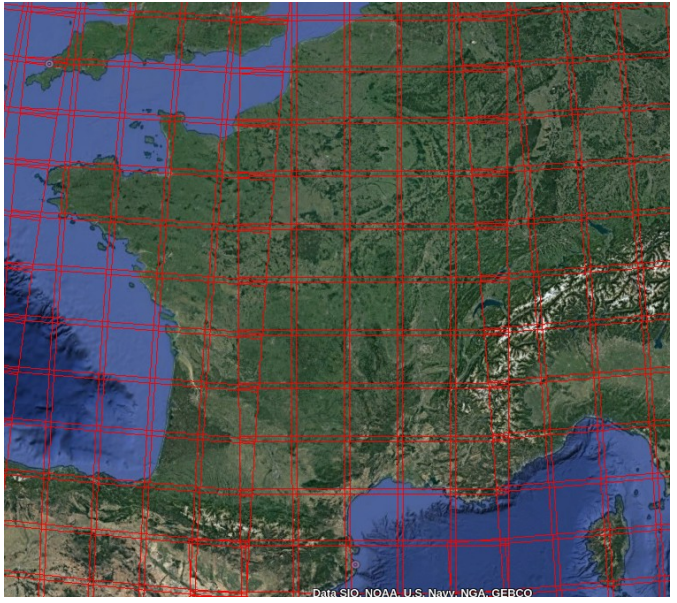
A²S: Automating processing on HPC system

A2S on HPC:

Goal : Efficiently use the ressources

-> **Fill the nodes** (use all their CPUs)
- expose parallelism **intelligently** between:
  . data parallelism (on different levels : pixels, S2 tiles, subtiles etc)
  . task parallelism





A²S: Automating processing on HPC system

A2S on HPC:

Goal : Efficiently use the ressources

-> **Fill the nodes** (use all their CPUs)

- expose parallelism **intelligently** between:
  - . data parallelism
  - . task parallelism

-> **Reduce CPUs idle time**

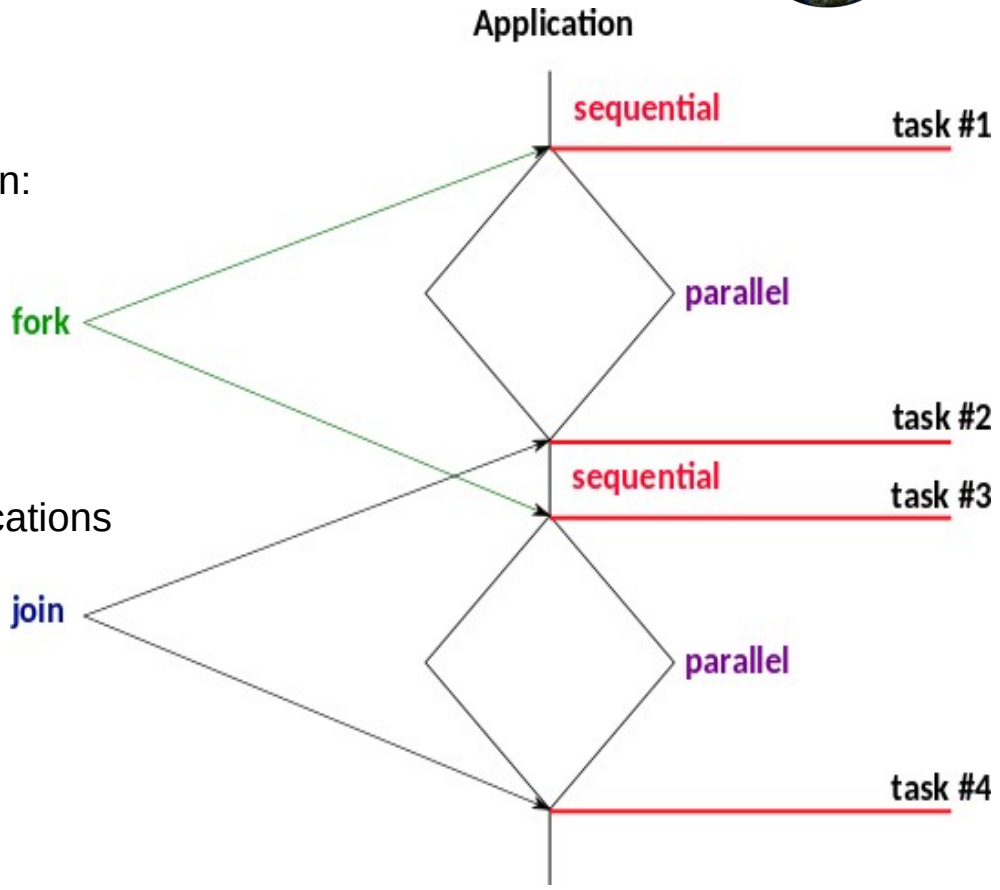
-> Minimize data movement

-> **cache mechanism**

-> beware of **fork / join patterns** in applications

-> **split in tasks**

- sequential
- parallel



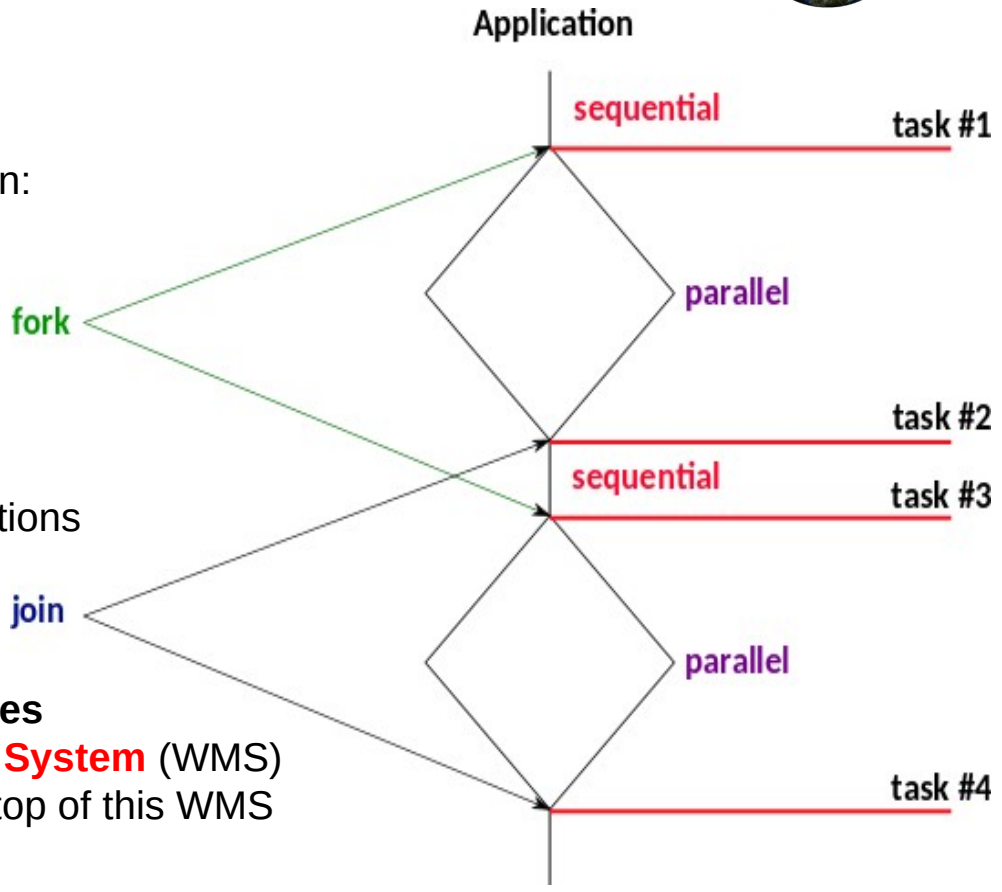
A²S: Automating processing on HPC system

A2S on HPC:

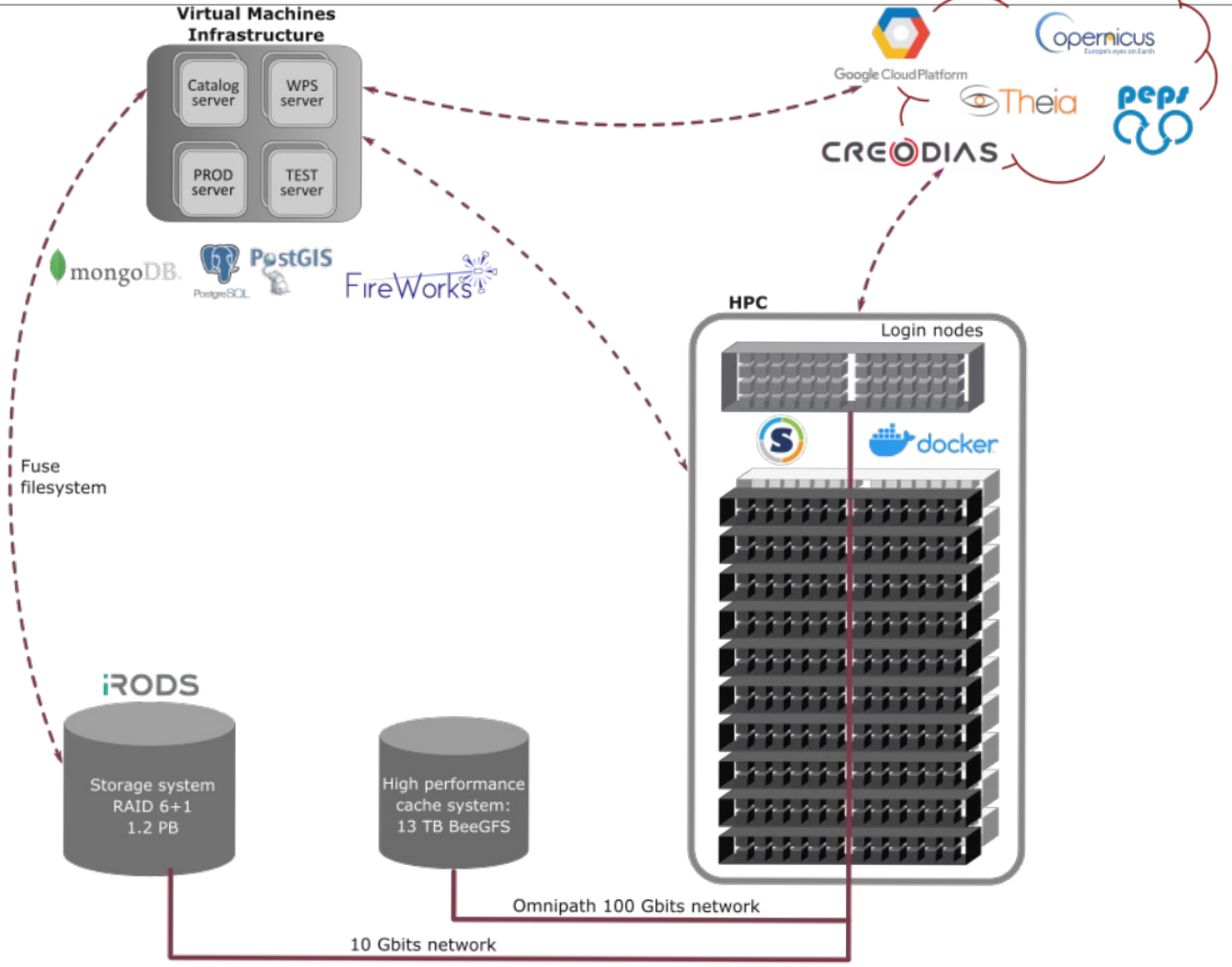
Goal : Efficiently use the ressources

-> **Fill the nodes** (use all their CPUs)
- expose parallelism **intelligently** between:
  . data parallelism
  . task parallelism

-> **Reduce CPUs idle time**
-> Minimize data movement
-> **cache mechanism**
-> beware of fork / join patterns in applications
-> **split in tasks**
- sequential
- parallel
-> **efficiently schedule tasks to the nodes**
-> use of a **Workflow Management System** (WMS)
-> use of a **task scheduler** built on top of this WMS



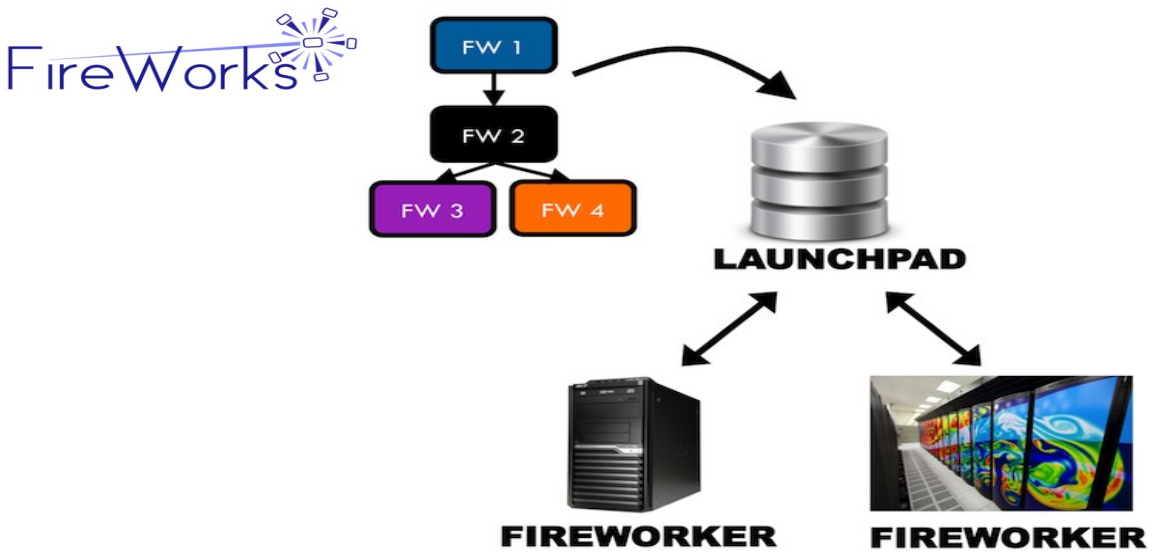
A²S: Infrastructure

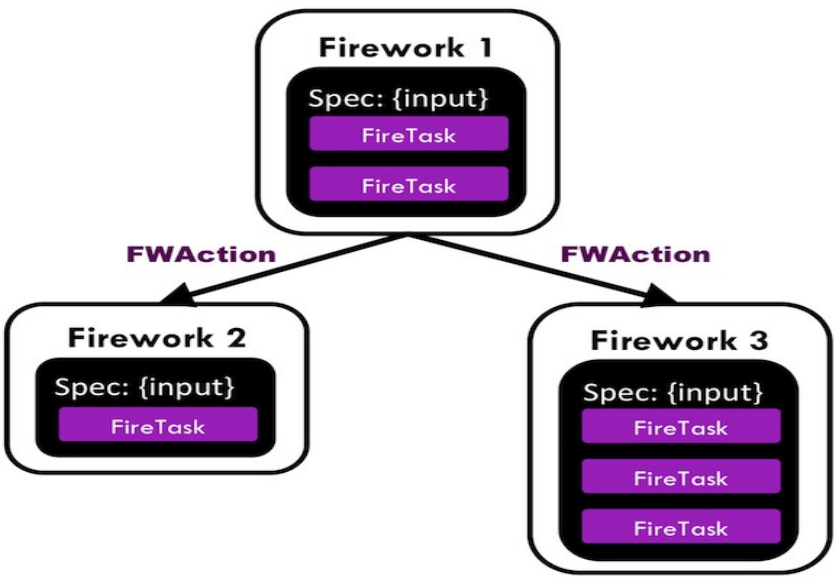


A²S: FireWorks – Workflow Management System

FireWork: free software for defining, managing, and executing workflows

- Complex **dynamic workflows** are defined using Python, stored in a MongoDB instance, can be monitored through a WEB GUI and queried through a python API.
- It allows to expose task parallelism inside a single application WF and task parallelism among different Wfs.
- It allows to **manage great complexity** and **overcome runtime problems**.





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

A²S: FireWorks – web monitoring interface

FireWorks

Newest Workflows

LiSc(PSe3)2 COMPLETED ID: 6832

- LiSc(PSe3)2-structure optimization
- LiSc(PSe3)2-static
- LiSc(PSe3)2-nscf uniform
- LiSc(PSe3)2-nscf line
- LiSc(PSe3)2-hse gap

GdAg(PSe3)2 FIZZLED ID: 6825

- GdAg(PSe3)2-structure optimization
- GdAg(PSe3)2-static
- GdAg(PSe3)2-nscf uniform
- GdAg(PSe3)2-nscf line
- GdAg(PSe3)2-hse gap

YAg(PSe3)2 COMPLETED ID: 6824

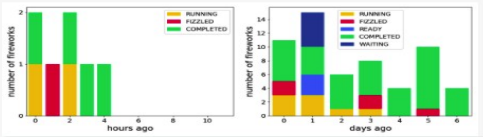
- YAg(PSe3)2-structure optimization
- YAg(PSe3)2-static
- YAg(PSe3)2-nscf uniform
- YAg(PSe3)2-nscf line
- YAg(PSe3)2-hse gap

LuAg(PSe3)2 COMPLETED ID: 6816

- LuAg(PSe3)2-structure optimization
- LuAg(PSe3)2-static
- LuAg(PSe3)2-nscf uniform
- LuAg(PSe3)2-nscf line
- LuAg(PSe3)2-hse gap

ScAg(PSe3)2 COMPLETED ID: 6810

- ScAg(PSe3)2-structure optimization
- ScAg(PSe3)2-static
- ScAg(PSe3)2-nscf uniform
- ScAg(PSe3)2-nscf line
- ScAg(PSe3)2-hse gap



Database snapshot

	Fireworks	Workflows
ARCHIVED	24	12
FIZZLED	344	335
PAUSED	0	0
DEFUSED	339	329
WAITING	297	0
READY	143	5
RESERVED	1	0
RUNNING	43	182
COMPLETED	5,443	2,198
TOTAL	6,634	3,061

Summary Reports

See a visual dashboard.

Get a report of all jobs for the past:

- 30 minutes
- 24 hours
- 7 days
- 30 days
- 6 months
- 24 months
- 10 years

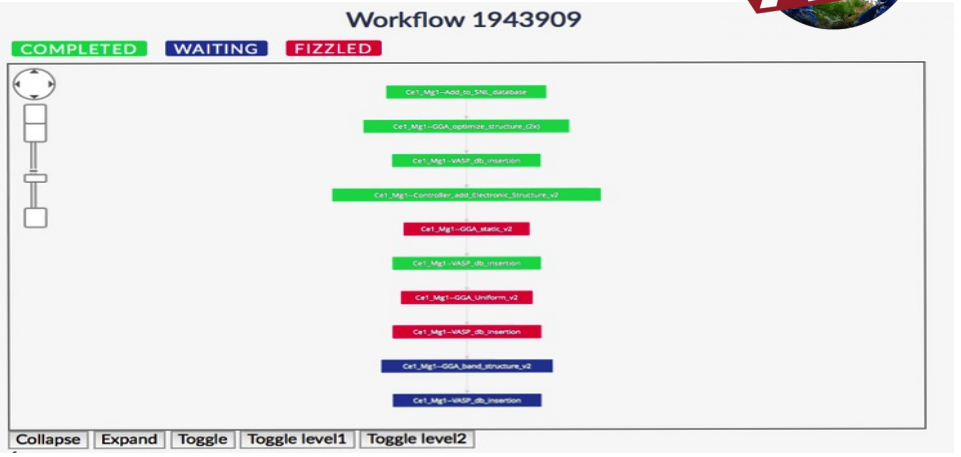
For more reporting options, use the "lpad report --help" command line tool.

MongoDB Query:

FW Query: {"spec.somevar": "someval"}

WF Query: {"metadata.somevar": "someval"}

Submit




```
{
    created_on: "2017-05-29T15:00:25.781000",
  + launch_dirs: { … },
  + links: { … },
  - metadata: {
        anonymized_formula: "AB",
        chemsystem: "Ce-Mg",
      + elements: [ … ],
        formula: "Ce4 Mg4",
        is_ordered: true,
        is_valid: true,
        nelements: 2,
        nsites: 8,
        reduced_cell_formula: "CeMg",
        reduced_cell_formula_abc: "Ce1 Mg1",
        run_version: "May 2013 (1)",
        submission_id: 125605
    },
    name: "Ce1 Mg1",
  + parent_links: { … },
    state: "FIZZLED",
  + states: { … },
    updated_on: "2017-05-31T22:01:19.851000"
}
```

A²S: Launching a Fireworks' rocket

The Workflows DB holds the tasks to be computed. They cycle through different states :

- WAITING : waiting for **all the parent tasks** to be (successfully) achieved
- READY : ready to run
- RUNNING : actually running
- COMPLETED : task succeeded
- FIZZLED : task failed.
- DEFUSED : dynamically defused by a parent task.

Launch a task : run the « **rocket launch** » command :

rlaunch (singleshot | rapidfire) [-q query] [-i task_id]

-> This drag from the WMS a **ready task** and executes it locally.

Database snapshot

	Fireworks	Workflows
ARCHIVED	0	0
FIZZLED	0	0
PAUSED	0	0
DEFUSED	0	0
WAITING	228	0
READY	0	0
RESERVED	0	0
RUNNING	27	2
COMPLETED	9	1
TOTAL	264	3

-> How to manage parallel and sequential tasks ?

A²S: Manage parallelism heterogenity

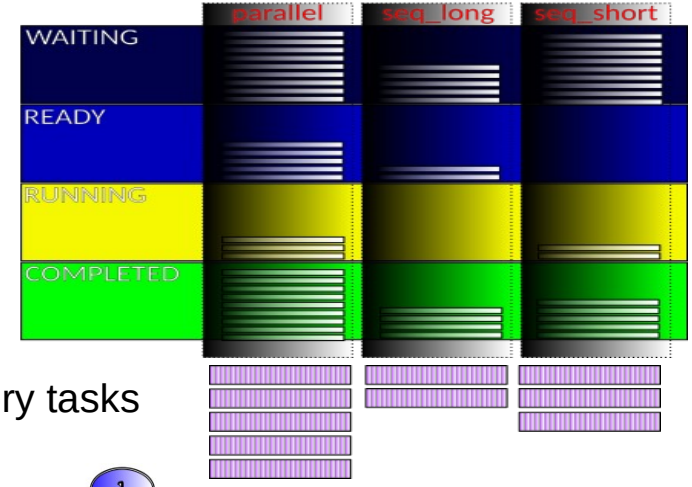
Each task can be tagged with a category.

-> Categories can be used to **discriminate between tasks that need a specific worker**
(ex. GPGPU tasks vs CPU only tasks)

-> We use categories to **discriminate between parallel or sequential tasks**
(full_node, half_node, quarter_node, sequential)

We could write a SLURM submission script for each category running :

rlaunch rapidfire -q <query on category>



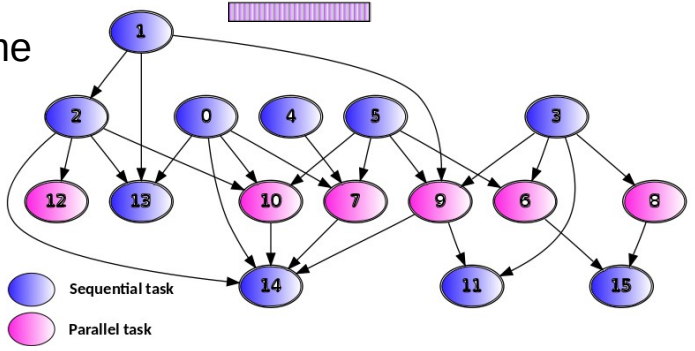
Fireworks provides queue adaptors allowing to directly submit to nodes :

qlaunch -w full_node.yaml -q full_node_qadapt.yaml rapidfire -m 5

-> ensure 5 nodes are always submitted to work on full_node category tasks

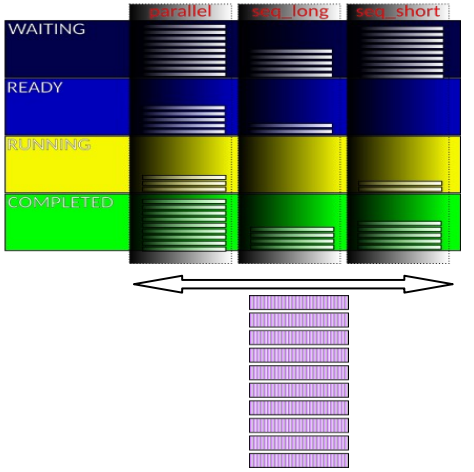
Repartition between parallel and sequential tasks can vary a lot along time

-> **severe load imbalance** (CPU waste)

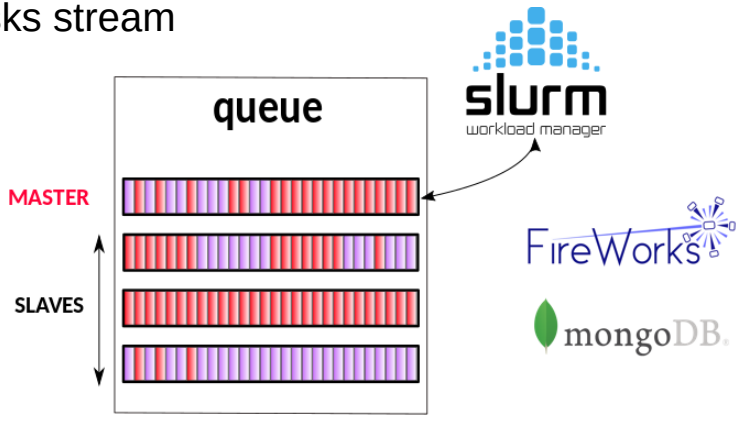


A²S: The Rocket scheduler

- We wrote a rocket scheduler which is the only program actually submitted on nodes.
- It knows :
    -> **How many tasks in each category are available**
    -> **How many ressources are available** on its node
    -> **How many workers are active**
- It launches the READY tasks (starting from higher parallelism to lower)
- it works in a **Master / slaves** model
- Each instance is responsible to **fill its node's cores**



- The master instance :
    -> computes the needed ressources to absorb the current tasks stream
    -> is able to :
        - submit more SLURM jobs if the load increases
        - stop running jobs if the load decreases



A²S: Automated execution and ressource provisionning

- Once a WF is submitted, its **execution is automatic** and start ASAP.

- The Master instance of rocket_scheduler **provisions new SLURM jobs when they are needed**.

- Each instance of rocket_scheduler is responsible to stop itself if the overall capacity of ressources overcomes the global needs.

don’t waste ressources => **get more ressources !**
(SLURM fair share)

File: a2s_scheduler.slurm

```
1  #!/usr/bin/env bash
2
3  #SBATCH --exclusive -N 1 --sockets-per-node=2
4  #SBATCH -t 8-00:00:00
5  #SBATCH -p pri2016
6  #SBATCH -A grant2ipgs
7  #SBATCH -J rockets
8
9  rocket_scheduler $SLURM_JOB_ID $SLURM_CPUS_ON_NODE
```

A²S: Two operating modes : Stream & On demand

- **Stream mode**:
  - **Fully automated**.
  - Fixed set of parameters
  - Workflow creation is triggered by the availability of new sources
  - Intended to produce products based on one basic source as soon as it is available
- **On demand mode**:
  - **Configurable** : Parameter file + entry point
    - Workflow creation & registration.
    - Execution on nodes is managed by the system
  - Can work on :
    - Remote / provided data sources
    - A²S stored products and sources
  - Intended to be triggered through **web-services**
  - **Time series processing** of individual products from the stream platform

A²S: Storage + managed cache

Use of a SRB (Storage Ressource Broker) : **iRODS**

Our configuration:

- Available capacity : **1 PB**
- No data redundancy
- **RAID 6+1** disk failure protection
- Actually 1O Gbits/s HPC <-> storage link
- Next 100Gbits/s inside Data Center

Data Movement is expensive !!!

- We built a 11TB BeeGFS high bandwith managed cache system.
  - -> All products are written in cache, then compressed and pushed on storage
  - -> The cached files live for a fixed duration (with a prolongation each time it used)
  - -> cache miss ? File is silently recreated from storage in background
  - -> **Stream mode should operate quite only on cached files**

A²S: some figures to conclude

We ran extensive tests on Stream platform this summer on year 2017 data.

- over 98 tiles on France & Belgium : Water surface & Urbanized surface
- over 23 tiles distributed world wide : S2 Offset tracking

product	footprint	max % CC	#
water surfaces	98 S2 tiles (Fr+Be)	2	1565
urbanized surfaces	98 S2 tiles (Fr+Be)	10	918
S2 correlograms	23 S2 tiles (worldwide)	30	1776

sources	imported	coregistred
S2	2238	1550

-> **~ 1 month** computation / **year** of data

=> **~ 1/10 global computing capacity**

-> **4 TB storage** used

A²S started operational product realisation phase
for Théia, CNES and ESA-GEP