# Development of a Scout Job Framework for Improving Efficiency of Analysis Jobs in Belle II Distributed Computing System

Hikari Hirata, Nagoya University
for the Belle II computing group
March 24, 2022

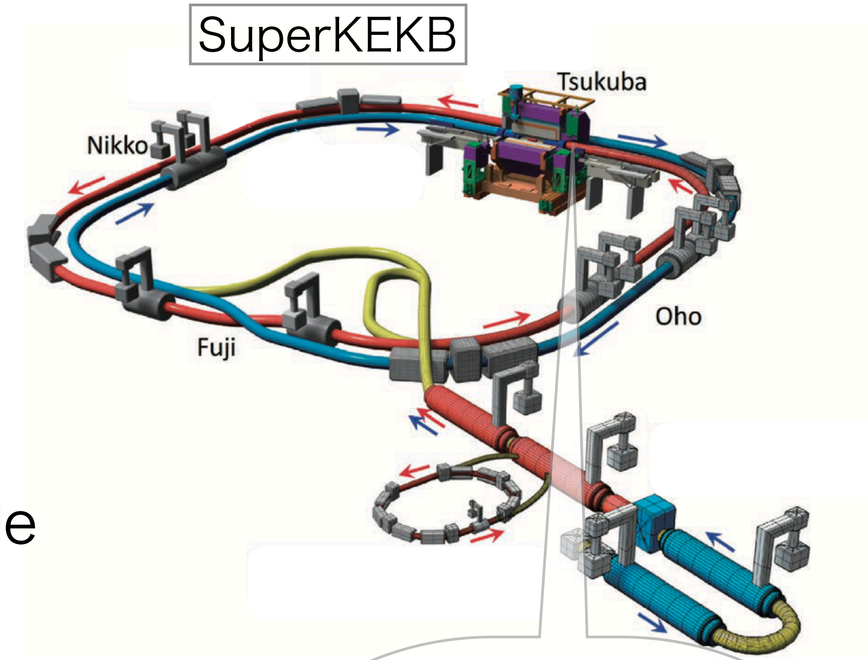
ISGC 2022

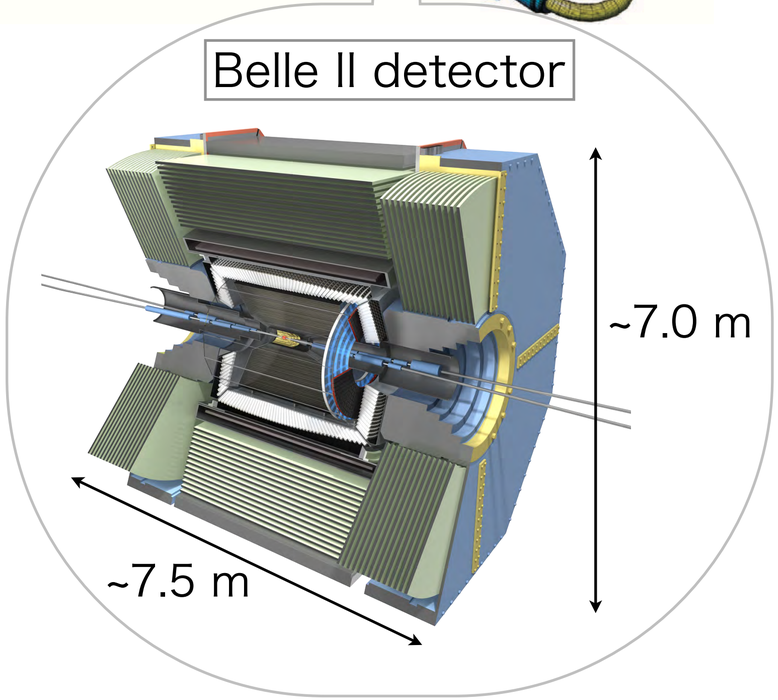
# Belle II Experiment

- Next Generation B-factory experiment using $e^+e^-$ collider at Tsukuba, Japan

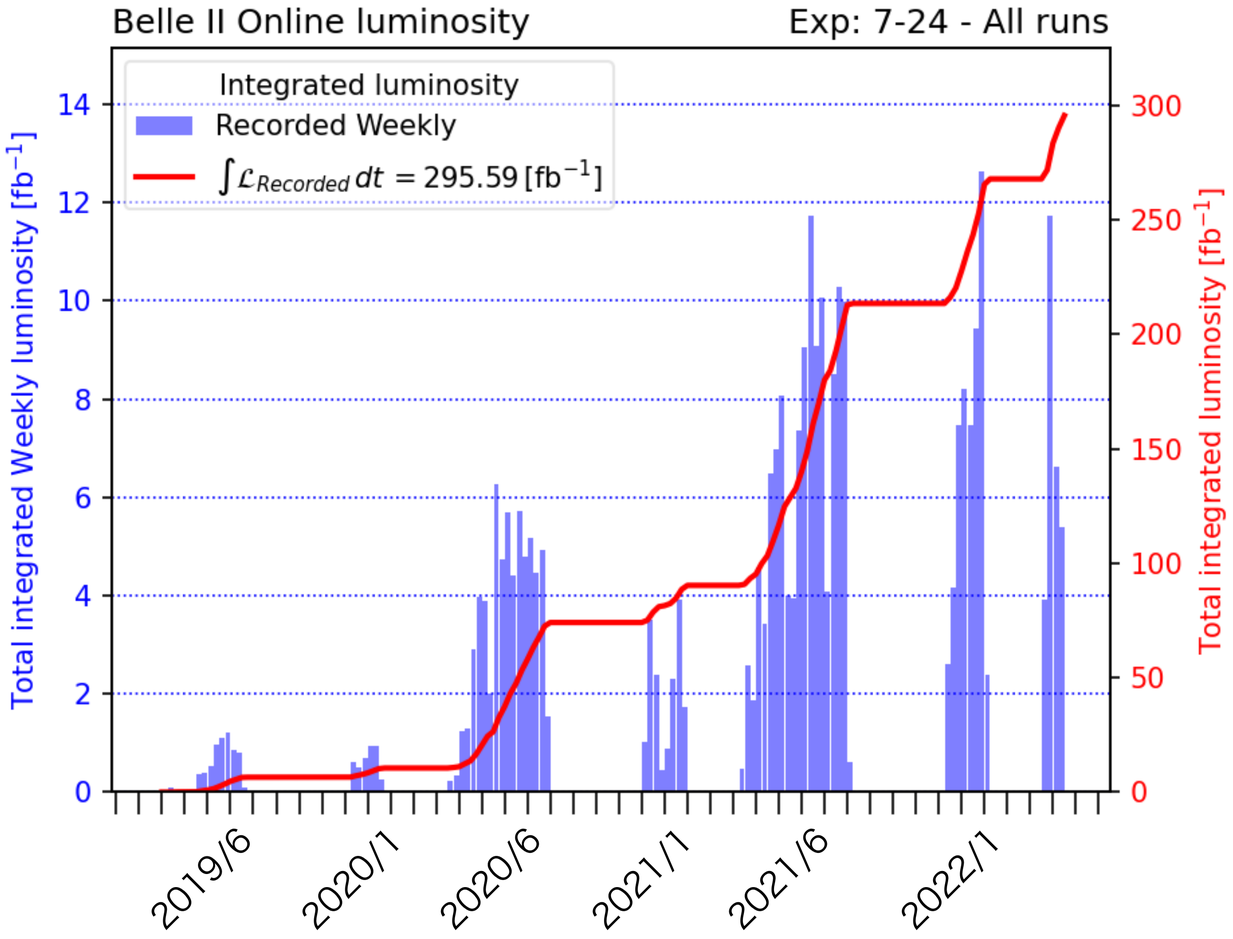
- Aim at advancing broadly understanding of particle physics
  - -- Search for physics beyond Standard Model
  - -- Precise measurement of electroweak interaction
  - -- Exploring properties of the strong interaction

- Started collecting collision data in the spring of 2019



# Status and Requirements for Computing

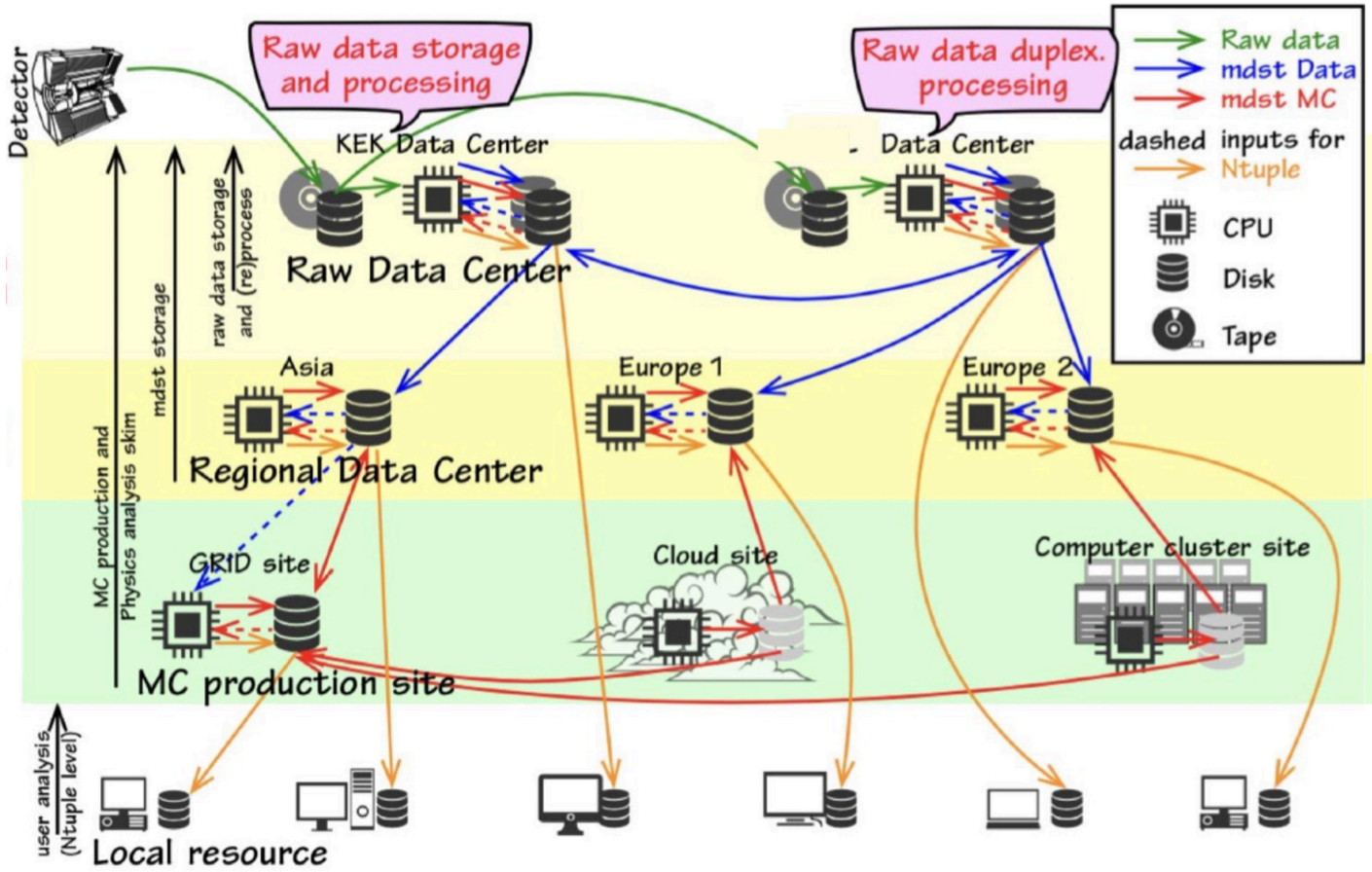



- Current achievements
  - -- A new luminosity world record
    $3.8 \times 10^{34}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$
  - -- 296 $\mathrm{fb}^{-1}$ data was recorded
- Targets
  - -- Instantaneous luminosity : $6.5 \times 10^{35}\ \mathrm{cm}^{-1}\mathrm{s}^{-1}$
  - -- Integrated luminosity : 50 $\mathrm{ab}^{-1}$
    (x 50 of predecessor experiment, Belle)

**At the end of data taking, to process and save data**
**→ need O($10^5$) CPU cores and O(100) PB disk storage**

# Computing Model in Belle II

- Use 60 computing sites around the world as a Distributed Computing (DC) system. (cf. The predecessor used the KEK central computing system mainly)




- Save and process raw data
  - -- A full set of raw data at KEK
  - -- A copy distributed over the other data centers
- Store processed and simulated data for further processing and analyses
- Produce simulation samples
  → Save at sites around the world
- Analyze data for individual physics motivation

# For Realization of Belle II Computing


https://github.com/DIRACGrid/DIRAC

- Interconnection among end-users and heterogeneous computing resources
- Provide a complete DC system

## BelleDIRAC

- Extention to meet our requirement
  - Support of analysis job execution on DC system (gbasf2)
  - Automation of generating simulation samples, processing raw data, etc


http://rucio.cern.ch/

- Advanced scientific Distributed Data Management (DDM) System
- Adopted as the Belle II DDM system snd replaced in early 2021

## Additional services

- FTS, AMGA, VOMS, CVMFS

**More information on our system and client tools using Rucio advantages**

**→ See a presentation of "Distributed Computing at Belle II" by A. Panta**

(24 Mar 2022, 15:50)

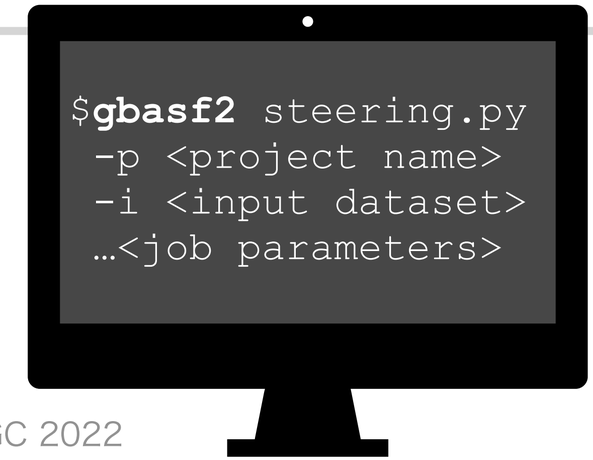
# Physics Analysis at Belle II

## Belle II Analysis Software Framework (basf2)

- Open-source software (https://github.com/belle2/basf2)
- Provide various modules for all aspects of the data-processing chain
  e.g. generating simulated data, reconstruction, etc…
- End-user specify path of modules with scripts based on Python 3
- Distribute it to computing resources by CVMFS

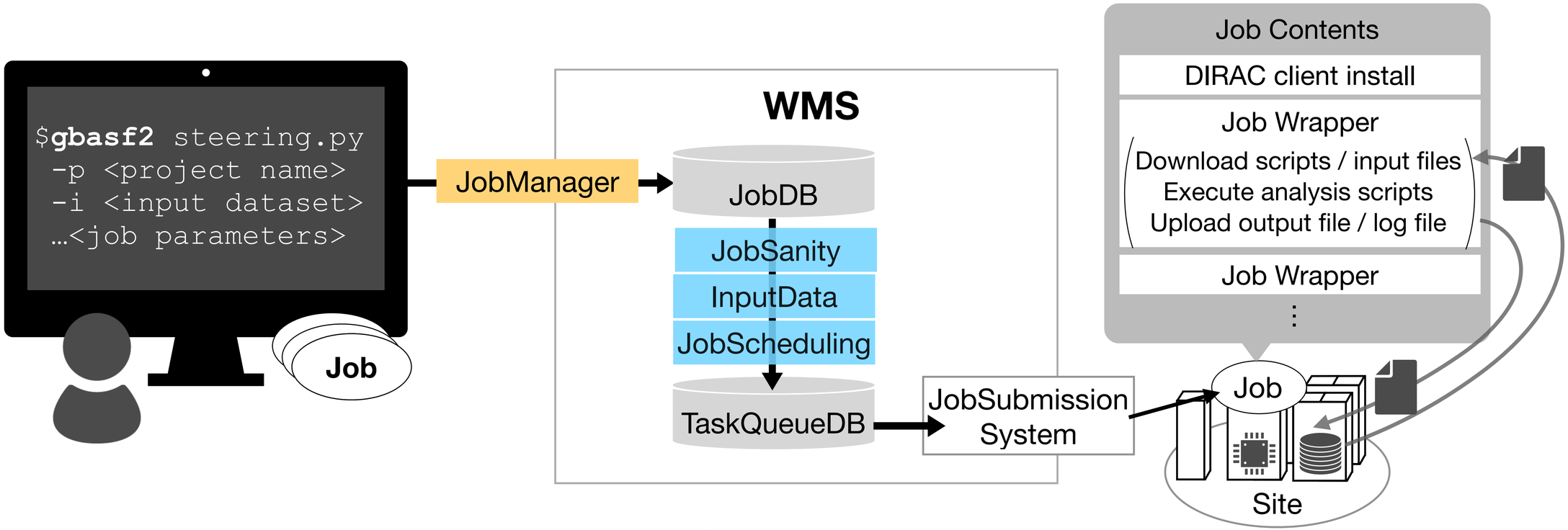

## gbasf2 (grid + basf2)

- To analyze distributed data, provide a tool to support basf2 job execution on DC system
- End-users can submit a cluster of basf2 jobs to DIRAC system by single command including job parameters



# Physics Analysis with Belle II DC system

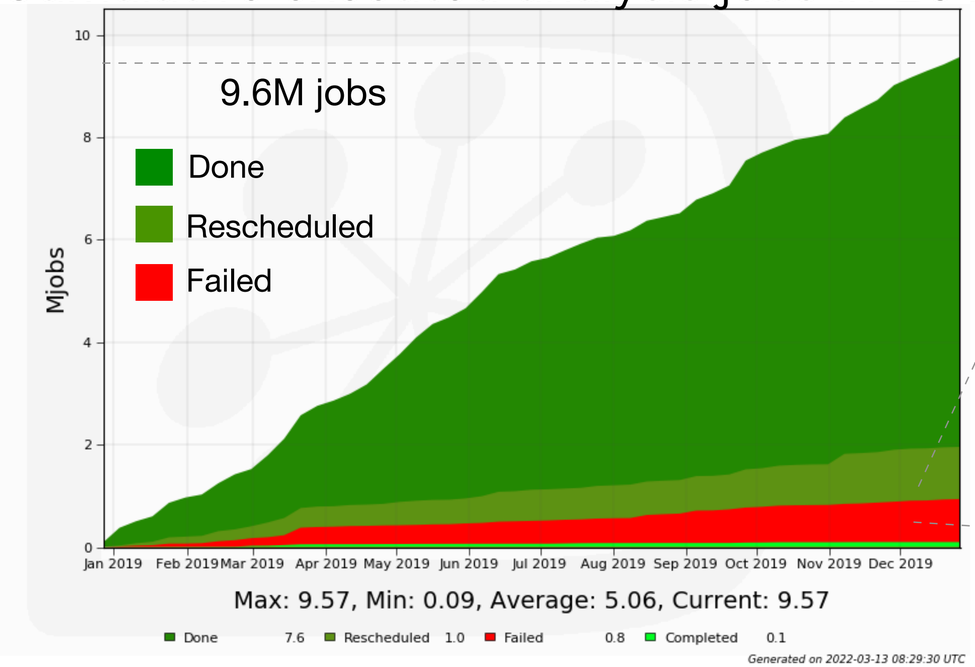
- End-user's jobs are stored and scheduled by Workload Management System (WMS)
- Gradually, WMS submits jobs to sites where input data is hosted
- End-user downloads output files and log after jobs are finished



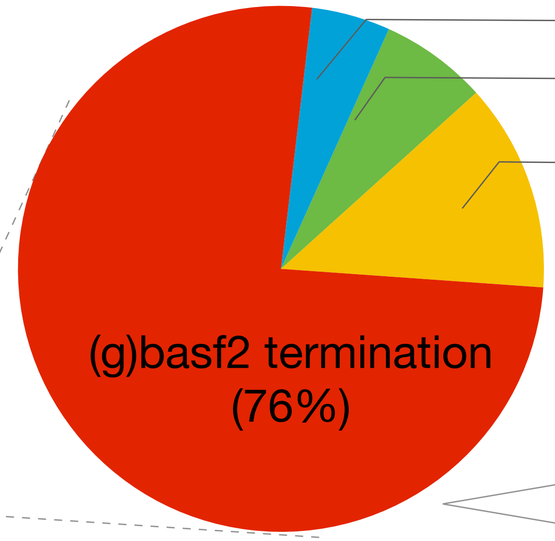
# Analysis Job Efficiency

- Efficient use of computing resources is important
- 9.6M analysis jobs were executed at our DC system, and 8.8% were failed jobs
  → Main causes were problematic analysis script and improper setting of job parameters

Cumulative executed analysis jobs in 2019




Failure reasons




Others (4%)

Killed (7%)

Stalled (13%)

Examples)

- Python syntax error
- basf2 error
- Improper setting of job parameters

# Problems Caused by Failed Jobs

- Failed jobs reduce job execution efficiency in two points
  - -- In our system, any job spends a few minutes on a worker node (download input files, authentication, etc)
    → Worker nodes are unnecessarily occupied
  - -- When many jobs are submitted to sites at once and they quickly fail, access to the central system is concentrated for a short time
    → Often trigger system trouble, and
    solving the trouble becomes a load on the operation side

→ Affect efficiency of analysis job execution

# Countermeasures

For all jobs

**Python Syntax Checker**

Detect a syntax error of Python level
→ Make users aware of careless mistake quickly

×

For huge job submissions

**Scout Job Framework**

Detect a complicated syntax error & improper setting of job parameters
→ Suppress problematic jobs

# Python Syntax Checker

- Compile analysis script at local environment with py_compile module
  → By executing it inside client tool,
  a simple syntax error can be detected quickly (e.g. open (), “”)

- Python environment differs between basf2 and gbasf2 (e.g. version)
  → Use the same environment as basf2 under CVMFS by default

- When an error is found, display a confirmation message to end-user before storing jobs in system

testSteering.py

```
import basf2 as b2
test = ‘Hello World'
print(f‘string value: {test}’)
# new feature incompatible python2
print(‘test) # should detect by this feature
```

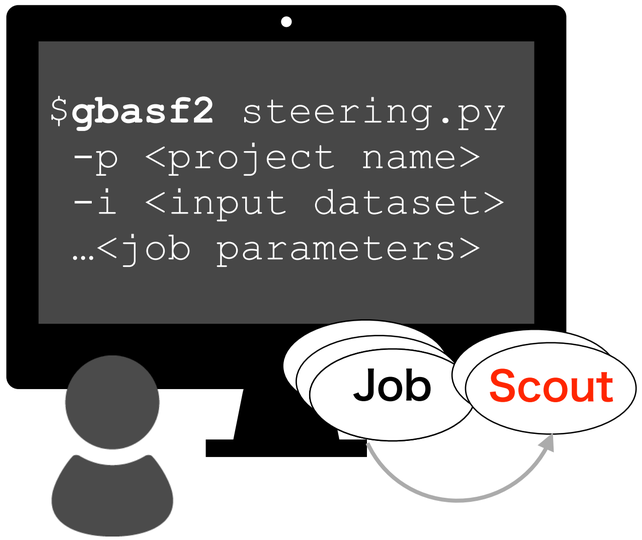
Output

```
[hikari@cw02 gbasf2_dev]$ gbasf2 testSteering.py
************************************************
  File "testSteering.py", line 9
    print("test)
                ^
SyntaxError: EOL while scanning string literal
************************************************
Please fix the SyntaxError.
Are you sure to submit the project?
Please enter Y or N:
```

# Scout Job Framework — Design 1 —

- Test jobs (scout jobs) and main jobs are stored in the system at the same time. While scout jobs are submitted, main jobs are submitted only when scout jobs are successful.

- Scout jobs are generated by client tool (gbasf2)
  - **More realistic test;** Copied from main jobs
    → Has the same analysis script and the same input data
    To distinguish scout jobs from main jobs,
    - set different JobType
    - save scout job information into JobDB
  - **More quick test;** Reduce processing events



# Scout Job Framework — Design 2 —

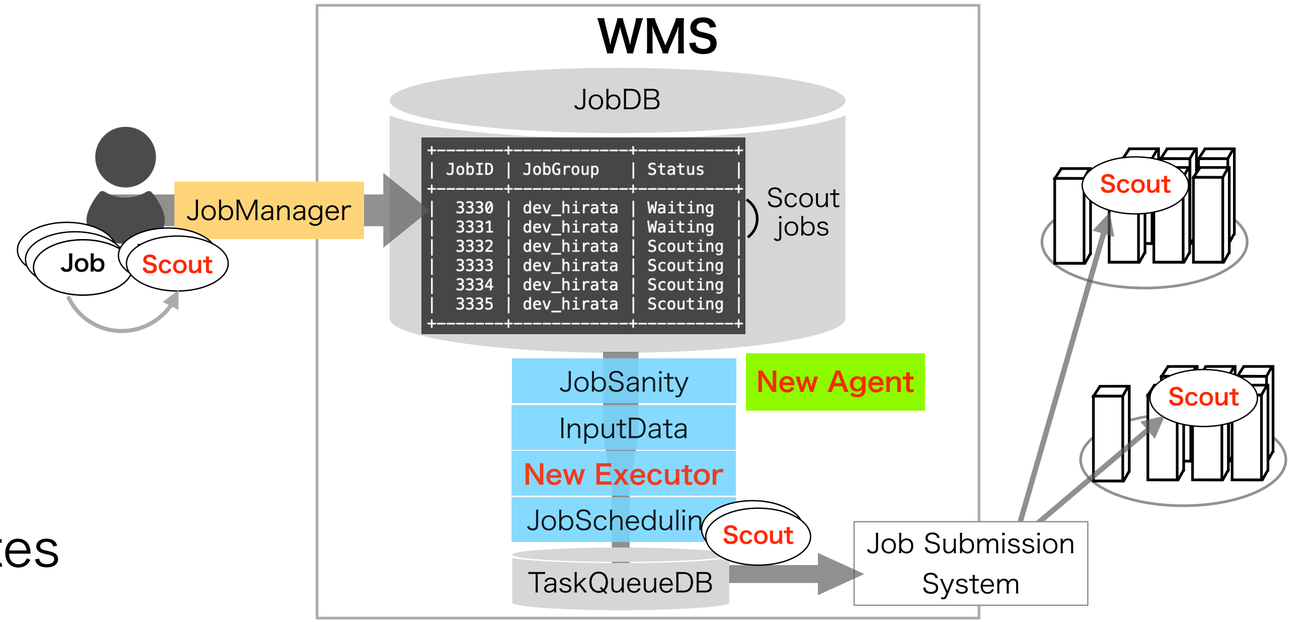
- To control submission of main jobs,
  Introduce new DIRAC modules to WMS

  -- **New Executor** : Task for Job scheduling
    - Until scout job completes, stop registering main jobs to TaskQueueDB

  -- **New Agent** : Component to perform actions periodically
    - Monitor status of scout jobs
    - Change status of main jobs according to the final status of scout jobs

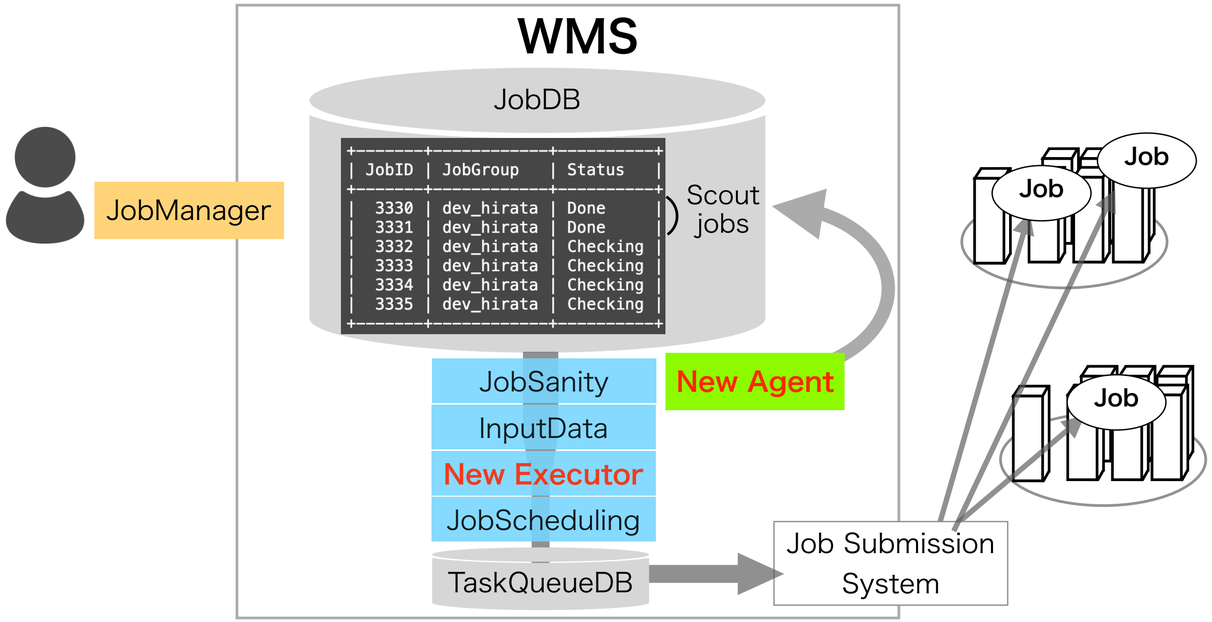
# Scout Job Framework — Workflow 1 —

1. Client tool makes instances of main jobs and scout jobs, and store all the jobs on JobDB
2. **New Executor** filters out main jobs, while only scout jobs go through into TaskQueueDB
3. WMS submits scout jobs to computing sites



# Scout Job Framework — Workflow 2 —

1. Client tool makes instances of main jobs and scout jobs, and store all the jobs on JobDB
2. **New Executor** filters out main jobs, while only scout jobs go through into TaskQueueDB
3. WMS submits scout jobs to computing sites
4. **New Agent** monitors status of scout jobs




- If scout fails, change status of all main jobs by “Failed”, and do nothing anymore
- If scout is successful, change status of all main jobs by “Checking” so that WMS submits them to computing sites

# Scout Job Framework — User Perspective —

- Currently, this framework is automatically used
- End-users can check status of scout jobs on DIRAC Job Monitor like normal jobs.
  - -- Also can check job's log with it

DIRAC Job Monitor when scout is successful

| | | | | | |
|---|---|---|---|---|---|
| ☐ | 171693262 | Done | Execution Complete | Prod_test_scout_1026 | User |
| ☐ | 171693261 | Done | Execution Complete | Prod_test_scout_1026 | User |
| ☐ | 171693260 | Done | Execution Complete | Prod_test_scout_1026 | User |
| ☐ | 171693259 | Done | Execution Complete | Prod_test_scout_1026 | User |
| ☐ | 171693258 | Done | Execution Complete | Prod_test_scout_1026 | UserScout |
| ☐ | 171693257 | Done | Execution Complete | Prod_test_scout_1026 | UserScout |

DIRAC Job Monitor when scout fails

| | | | | | |
|---|---|---|---|---|---|
| ☐ | 171693547 | Failed | Failed in scouting | Prod_test_scoutFail_1026 | User |
| ☐ | 171693546 | Failed | Failed in scouting | Prod_test_scoutFail_1026 | User |
| ☐ | 171693545 | Failed | Failed in scouting | Prod_test_scoutFail_1026 | User |
| ☐ | 171693544 | Failed | Failed in scouting | Prod_test_scoutFail_1026 | User |
| ☐ | 171693543 | Failed | Application Finished With Errors | Prod_test_scoutFail_1026 | UserScout |
| ☐ | 171693542 | Failed | Application Finished With Errors | Prod_test_scoutFail_1026 | UserScout |

# Scout Job Framework — Result —

- Implemented framework to production environment a year ago.
  - Could reduce operational load and waste of computational resources
    - Suppressed 644k failed jobs at least. If this framework was not implemented, there would have been 24% more failed jobs this year
    - Suppressed system troubles due to the failed jobs
  - Also beneficial for end-users because pre-test is streamlined

- Ideas for further improvement
  - Adding a function to automatically correct the problematic job parameters
  - Implement this framework into the system for automatically generating simulation samples and processing data

# Summary

- At Belle II, we are accumulating data to understand particle physics broadly.
- To save and process massive data, DC system with worldwide computing resources is used.
  → Efficient use of computing resources is important
- To suppress analysis jobs failure due to problematic analysis script or improper setting of the job parameters, python syntax checker and scout job framework were introduced.
- System trouble and waste of computing resources could be reduced.
  End-user's pre-test can be streamlined.
  → Improve efficiency of analysis job execution!