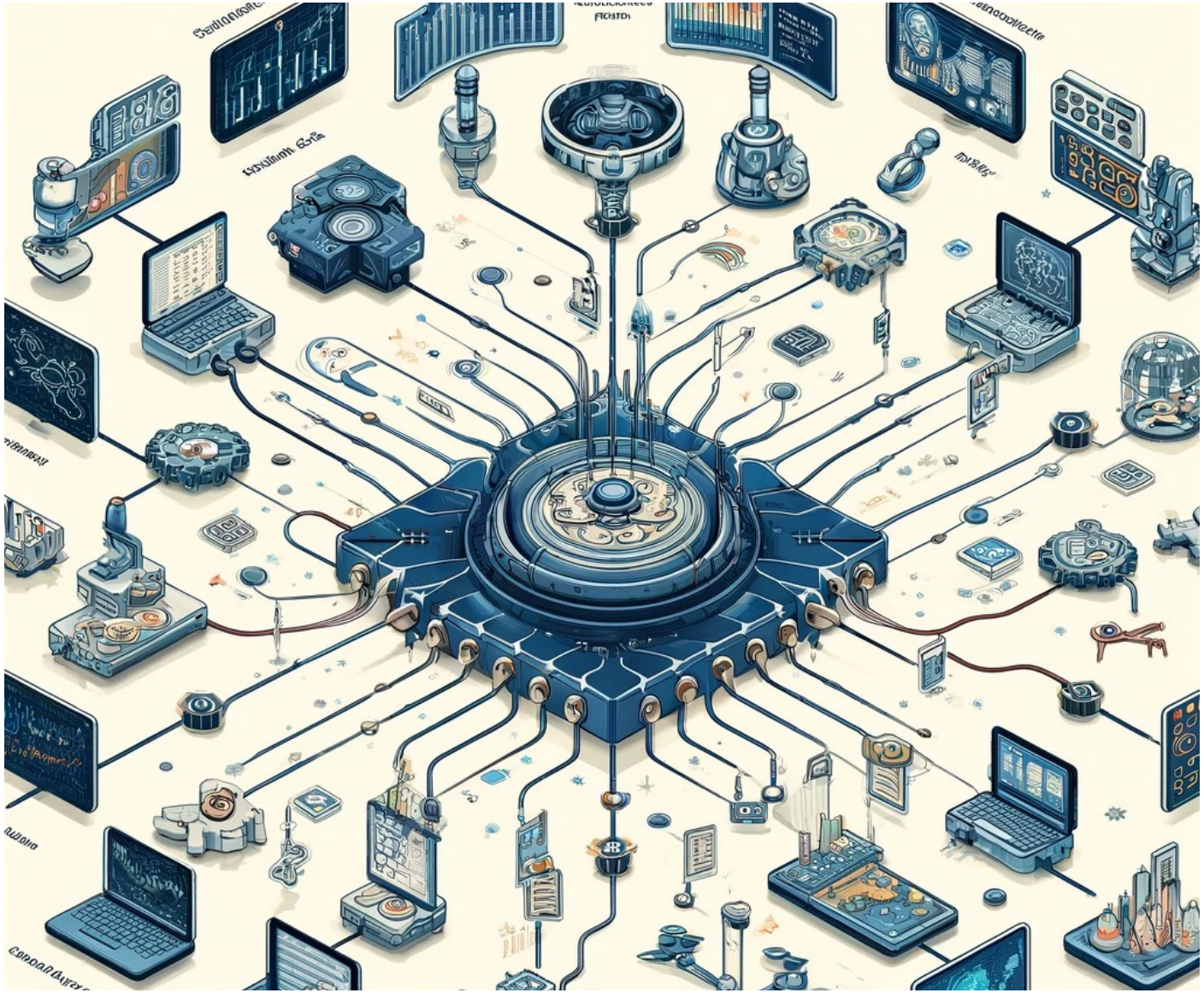
Exploring Scientific Workflows with CWL and dispel4py

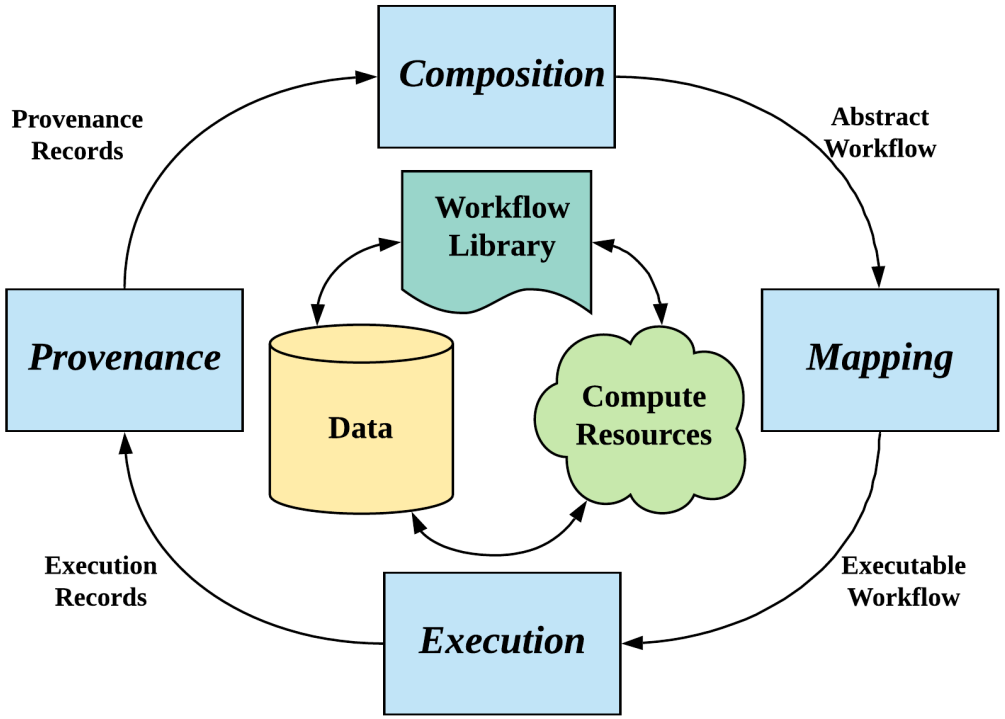
Module 4

- Dr. Rosa Filgueira
- Lecturer at the School of Computer Science
- University of St Andrews
- rf208@st-andrews.ac.uk
- rosa.filgueira.vicente@gmail.com



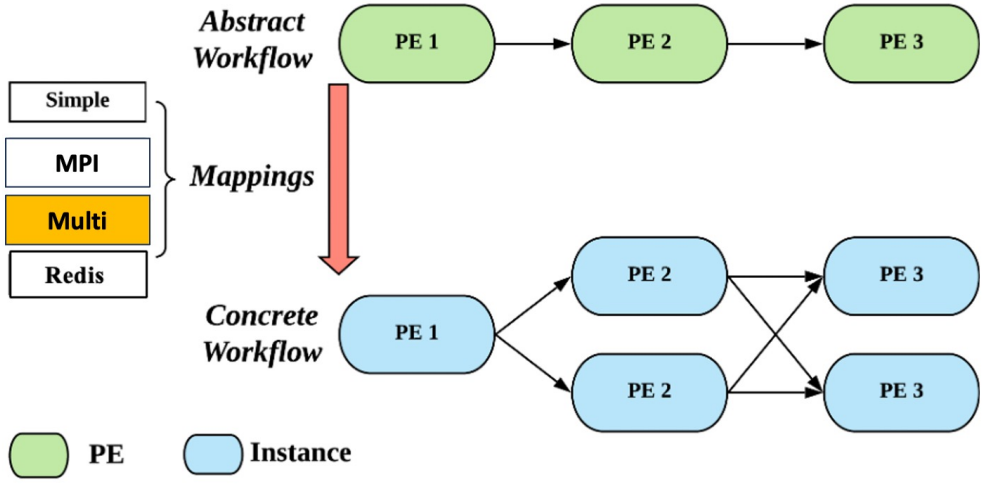
Module 4 – dispel4py latest research updates

1. Recap
2. Laminar
3. dispel4py Scheduling Strategies



Recap

dispel4py is parallel stream-based dataflow library

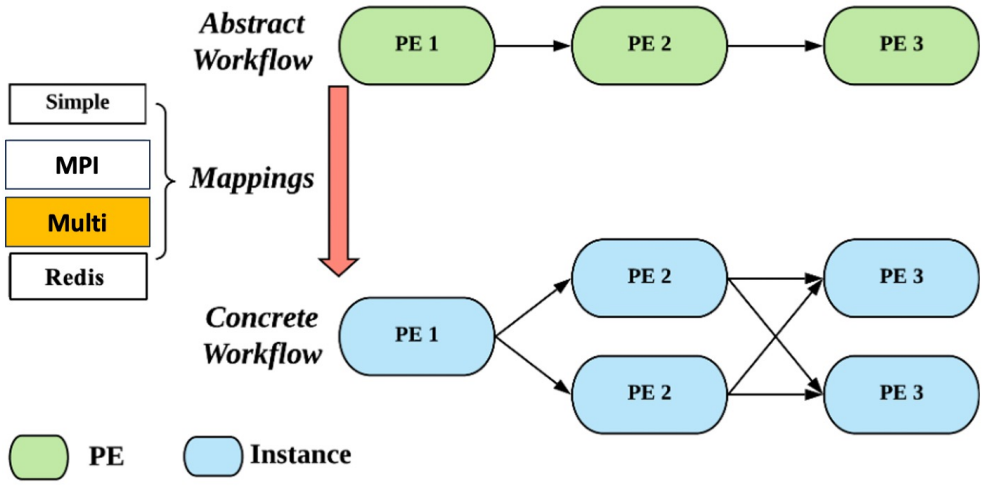



```python
class NumberProducer(ProducerPE):
    def __init__(self):
        ProducerPE.__init__(self)
    def _process(self):
        # Generate a random number
        result= random.randint(1, 1000)
        # Return the number as the output
        return result
```

NumberProducer stateless PE

Recap

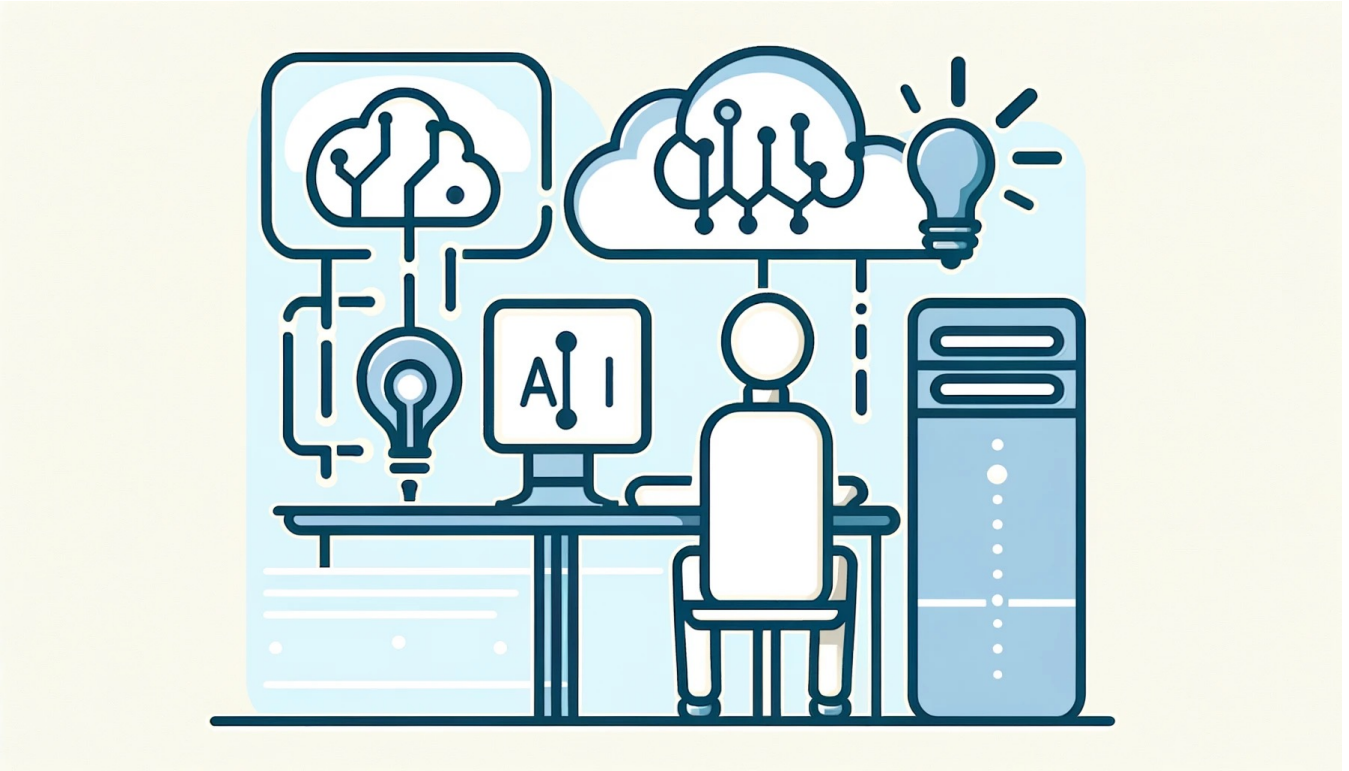
dispel4py is parallel stream-based dataflow library




```python
class CountWords(GenericPE):
    def __init__(self):
        from collections import defaultdict
        GenericPE.__init__(self)
        #Add an input port named "input", from which
        #it will receive tuples with shape (word, 1).
        #Data is group-by (MapReduce)
        #the first element (index 0) of the tuples
        self._add_input("input", grouping=[0])

        #Add an output port named "output"
        self._add_output("output")

        #Initialize a stateful variable
        #to store word counts
        self.count = defaultdict(int)

    def _process(self, inputs):
        import os
        #Extract word and count from the input
        word, count = inputs['input']
        # Update the count for the word
        self.count[word] += count
```

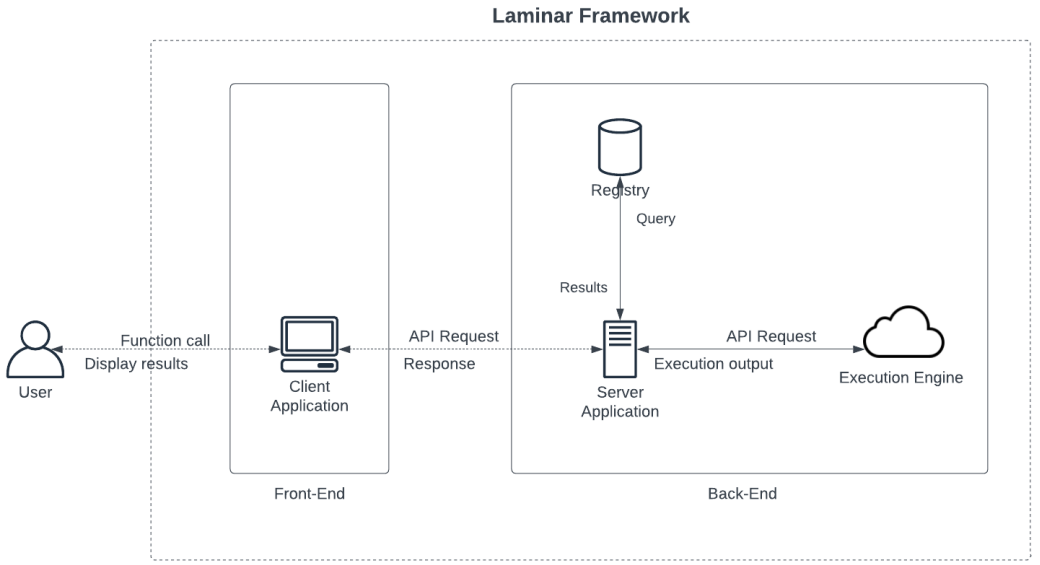
CountWords stateful PE

2. Laminar: Serverless Stream-based Framework with Deep Learning features



Laminar: Serverless Stream-based Framework

- *Laminar* serverless framework for stream-based framework based on *dispel4py*
- It manages streaming data, data-pipelines
- Accommodates stateless and stateful computations
- Deep Learning Features
- Four main components
  - Registry: Stores users, PEs and workflow information
  - Server: Coordinates system functionality
  - Execution Engine: Serverless workflow execution
  - Client: Interacts with server and users requests



[Laminar: a new serverless stream-based framework with semantic code search and code completion]

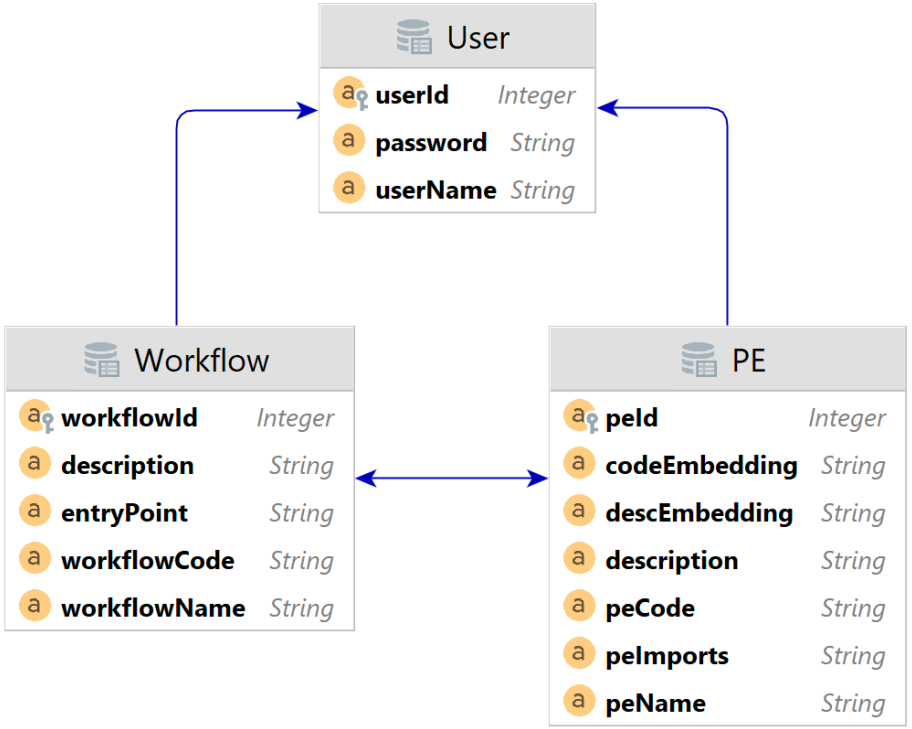
Laminar: Serverless Stream-based Framework

- Registry deep learning models

- *UniXcoder-code-search* description embeddings
- -PE descEmbeddings

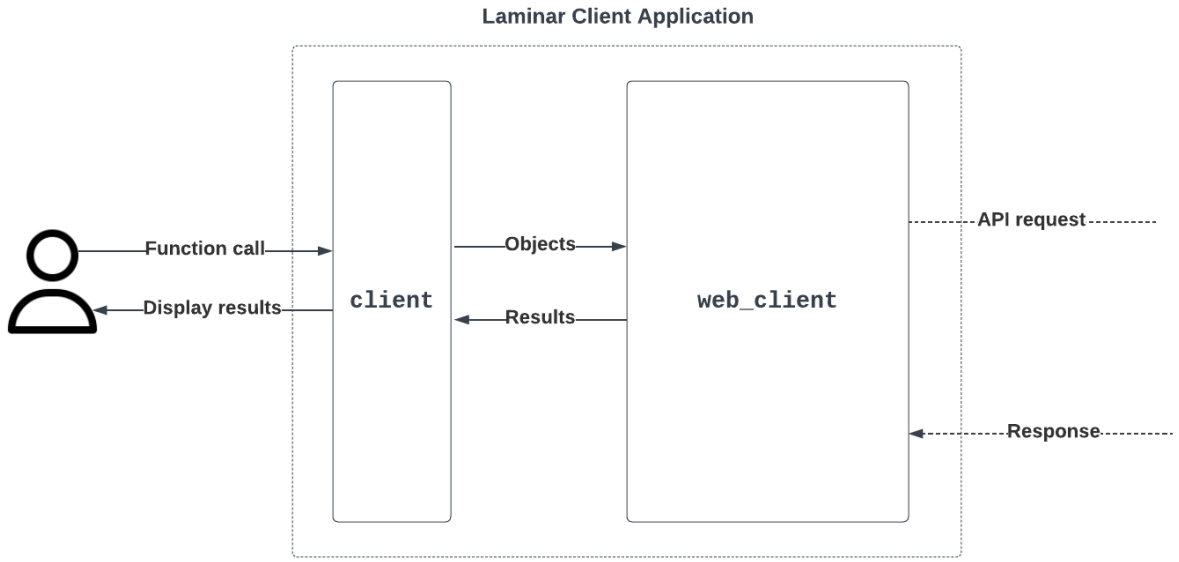
- *ReACC-py-retriever* code embeddings
- -PE codeEmbeddings

- *CodeT5-base-multi-sum*
  - - automatic PE descriptions
  - - only if users do not provide them



Laminar: Serverless Stream-based Framework

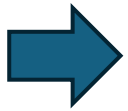
- Client Functions calls:
  - Register Users, PE, workflows (app)
  - Login Users
  - Get PEs and workflows
  - Describe PEs and workflows
  - List PEs of a workflow
  - Remove PEs and workflows
  - Get everything from the registry
  - Execute workflows or PEs
  - Search PEs and Workflows:
    - Text-based Search
    - Semantic Code Search for PEs
    - Automatic Code Completion



Laminar: Serverless Stream-based Framework

Text-based Search

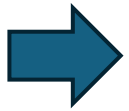
```
client.search_Registry("prime", "workflow")
```



```
Searched for "prime"

REGISTRY

Result 1: ID: 2
Workflow Name: isPrime
Description: Workflow that prints random prime numbers
```

Semantic PE Code Search

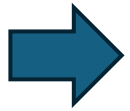
```
client.search_Registry("A PE that checks if
                       a number is prime",
                       "pe","text")
```



```
Searched for "A PE that check if a number is prime"
   peId               peName                                    description  similarity
      3              IsPrime              check if the input is a prime or not    0.796563
     17         TestProducer               This PE produces a range of numbers    0.502303
     10   InternalExtinction  function to calculate internal extinction from...    0.216504
     19  TestDelayOneInOneOut  This method is called by the PE base class to ...    0.213435
     18       TestOneInOneOut              This PE copies the input to an output    0.186635
```

Automatic Code Completion

```
client.search_Registry("random.randint(1, 1000)",
                       "pe","code")
```



```
Searched for "random.randint(1, 1000)"
   peId            peName                                    description  similarity
      4     NumberProducer  This function is called to generate a random s...    0.752691
      5        PrintPrime  Process the sequence of words in the sequence ...    0.671739
     22  TestMultiProducer  Process the sequence of sequence sequence sequ...    0.635483
      3            IsPrime              check if the input is a prime or not    0.619670
     17       TestProducer               This PE produces a range of numbers    0.552182
```

Laminar: Serverless Stream-based Framework

Semantic Code Search Evaluation

Model	Zero-shot Code Search	
	CosQA	CSN
	MRR	
unixcoder-base	43.1	44.7
unixcoder-code-search	**58.8**	**72.2**

Text-2-code evaluations

Mean Reciprocal Rank (MRR) metric: how quickly the system finds the most relevant result on average

Code Completion Evaluation

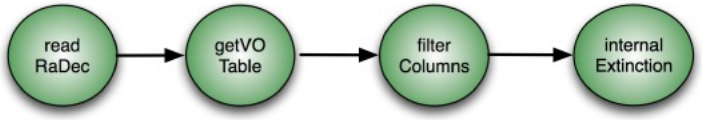
Model	MAP@100	Precision at 1
CodeBERT	1.47	4.75
GraphCodeBERT	5.31	15.68
ReACC-retriever-py	9.60	**27.04**
thenlper/gte-large	1.9	7
BAAI/bge-large-en	8.17	20
unixcoder-clone-detection	**10.4**	17
unixcoder-code-search	8.53	22.84

Zero-shot clone detection evaluation

Assessing a models' ability to retrieve similar code segments from a dataset using partial queries:

- MAP@100 Mean Average Precision at 100):
  ** Average precision of the top 100 retrieved items for each query
- Precision at 1:
  ** Accuracy in retrieving the most relevant item as the top recommendation

Laminar: Serverless Stream-based Framework



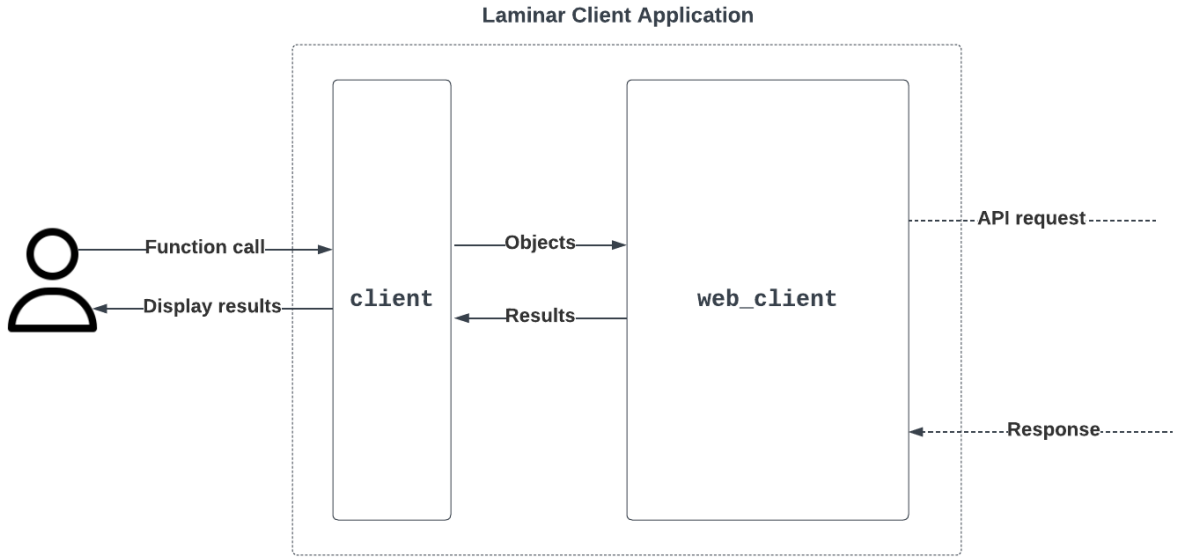
Streaming workflow for calculating the internal extinction of galaxies

```
graph.connect(read, 'output', votab, 'input')
graph.connect(votab, 'output', filt, 'input')
graph.connect(filt, 'output', intext, 'input')
```

```
client.register_Workflow(
    graph,
    "Astrophysics",
    "A workflow to compute the
    internal extinction of galaxies")
```

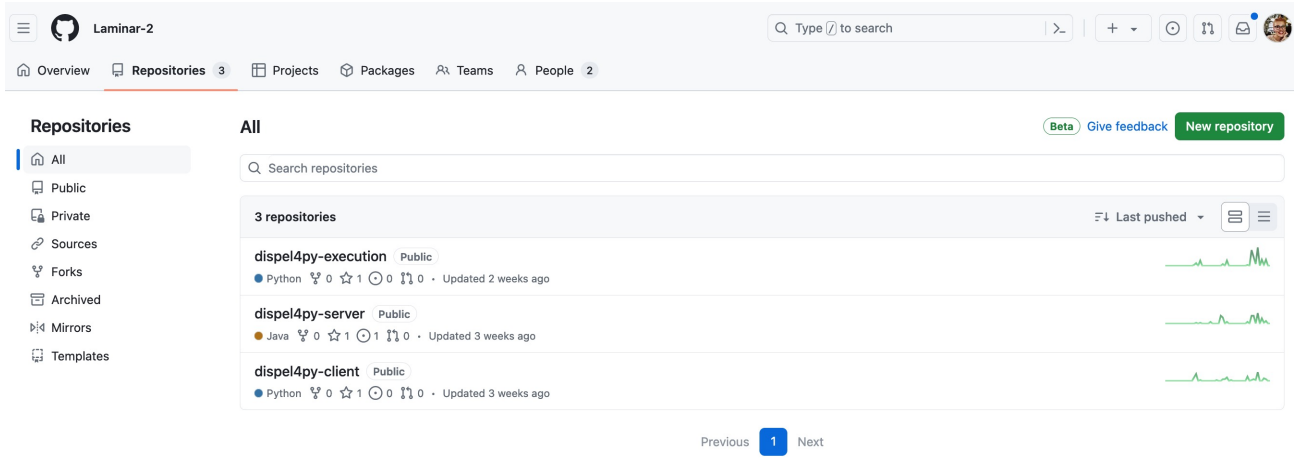
```
workflow = client.get_Workflow("Astrophysics")
```

```
client.run(workflow,
  input=[{"input":"resources/coordinates.txt"}],
  process=REDIS,
  args={'num':10}
  resources=True)
```

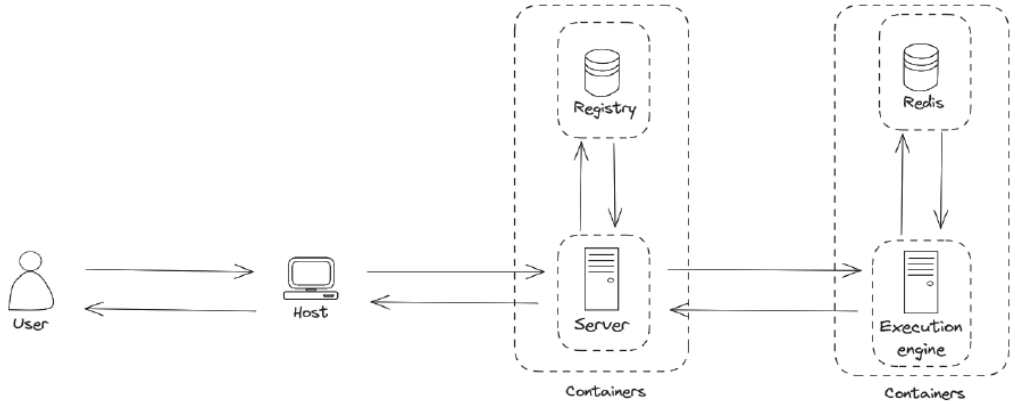


New: Currently – applying semantic facilities to workflows and improving Laminar end-points

Laminar: Serverless Stream-based Framework



Working on a new version – Laminar 2.0 !!



Laminar 2.0's architecture with containerisation

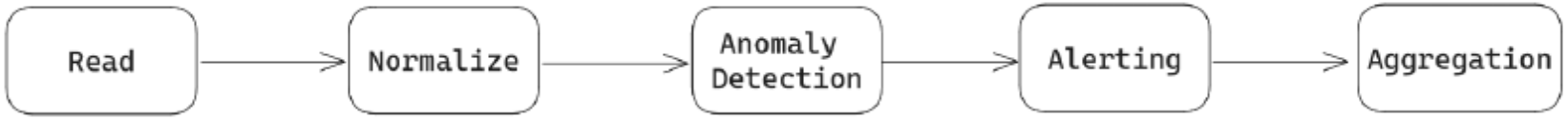
Laminar: Serverless Stream-based Framework

```
from dispel4py.base import ProducerPE, IterativePE, ConsumerPE
from dispel4py.workflow_graph import WorkflowGraph
import random
from easydict import EasyDict as edict
from client import d4pClient,Process
from dispel4py.new.dynamic_redis import process as dyn_process
from dispel4py.new.simple_process import process as simple_process
from dispel4py.new.multi_process import process as multi_process

class NumberProducer(ProducerPE):
    def __init__(self):
        ProducerPE.__init__(self)

    def _process(self, inputs):
        # this PE produces one input
        result= random.randint(1, 1000)
        return result

class IsPrime(IterativePE):
    def __init__(self):
        IterativePE.__init__(self)
    def _process(self, num):
        # this PE consumes one input and produces one output
        print("before checking data - %s - is prime or not\n" % num, end="")
        if all(num % i != 0 for i in range(2, num)):
            return num

class PrintPrime(ConsumerPE):
    def __init__(self):
        ConsumerPE.__init__(self)
    def _process(self, num):
        # this PE consumes one input
        print("the num %s is prime\n" % num, end="")


producer = NumberProducer()
isprime = IsPrime()
printprime = PrintPrime()

graph = WorkflowGraph()
graph.connect(producer, 'output', isprime, 'input')
graph.connect(isprime, 'output', printprime, 'input')
```

IsPrime workflow with Laminar

```
client = d4pClient()
client.login("username", "password") # Provide login details here

#SIMPLE

client.run(graph,input=100)

#MULTI

client.run_multiprocess(graph,input=100)

#REDIS

client.run_dynamic(graph,input=100)
```

Laminar: Serverless Stream-based Framework

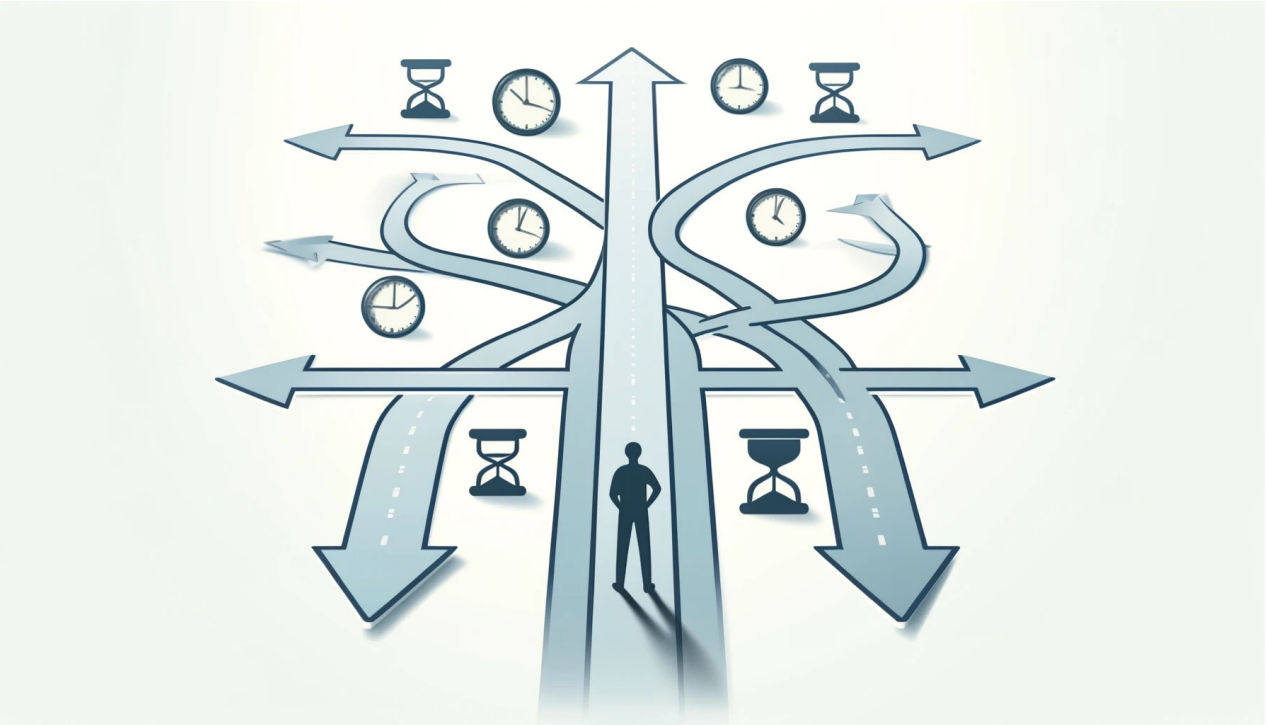


sensor_workflow provides a streamlined process for handling sensor data, from ingestion to analysis and summarization

```
(laminar) register SensorWorkflow.py
Found PEs
• aggregate_data - AggregateDataPE (ID 6)
• alerting - AlertingPE (ID 7)
• anomaly_detection - AnomalyDetectionPE (ID 8)
• normalize_data - NormalizeDataPE (ID 9)
• read - ReadSensorDataPE (ID 10)
Found workflows
• sensorWorkflow - WorkflowGraph (ID 11)
```

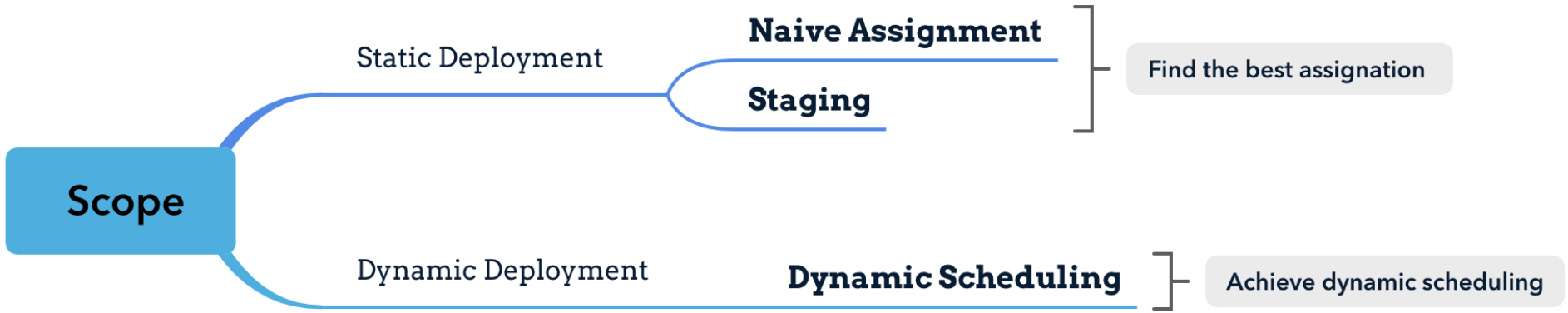
```
run sensorWorkflow -i '[{"input" : "sensor_data_1000.json"}]' \
    --resource sensor_data_1000.json
run sensorWorkflow -i '[{"input" : "sensor_data_1000.json"}]' \
    --resource sensor_data_1000.json --multi
run sensorWorkflow -i '[{"input" : "sensor_data_1000.json"}]' \
    --resource sensor_data_1000.json --dynamic
```

3. dispel4py Scheduling Strategies



Maximize the performance of dispel4py

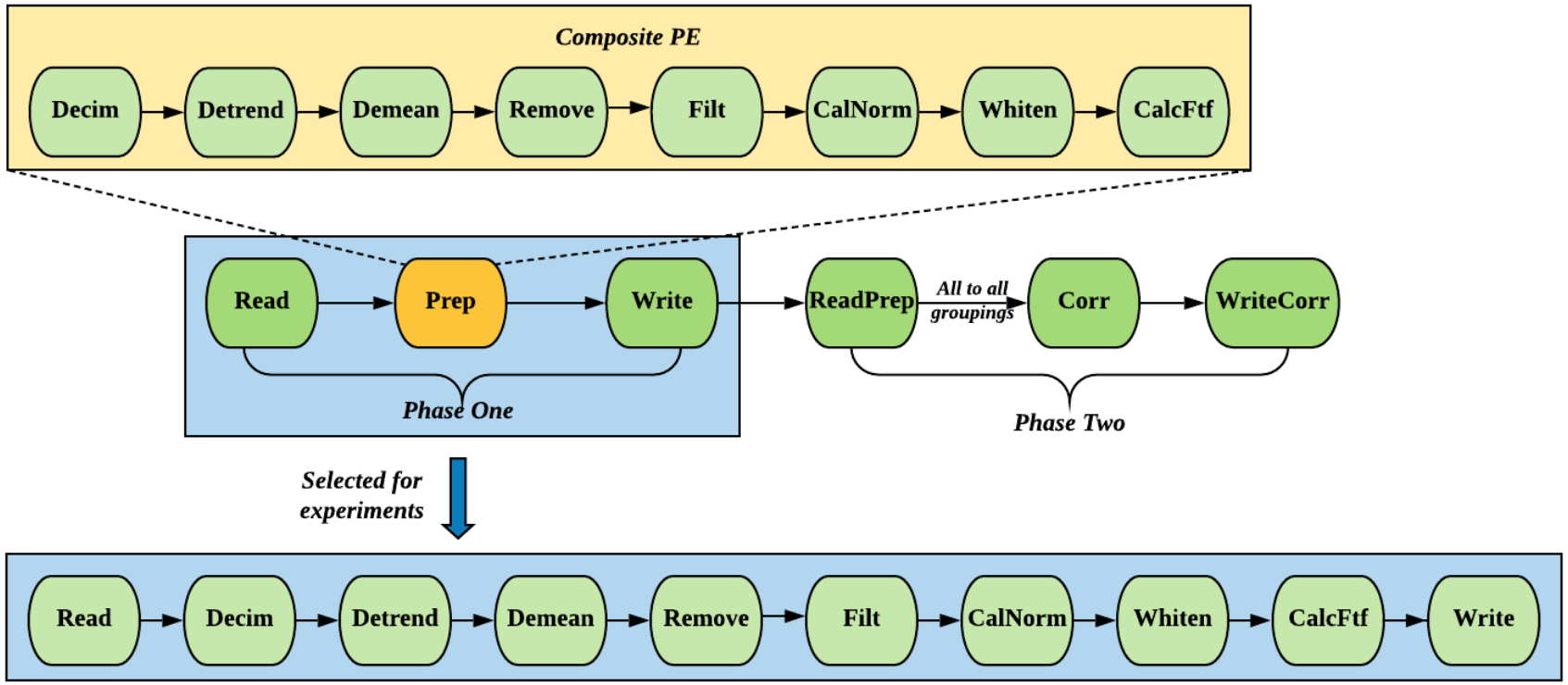
Scheduling strategies that automatically adapt to workflows features



Scalable adaptive optimizations for stream-based workflows in multi-HPC-clusters and cloud infrastructures

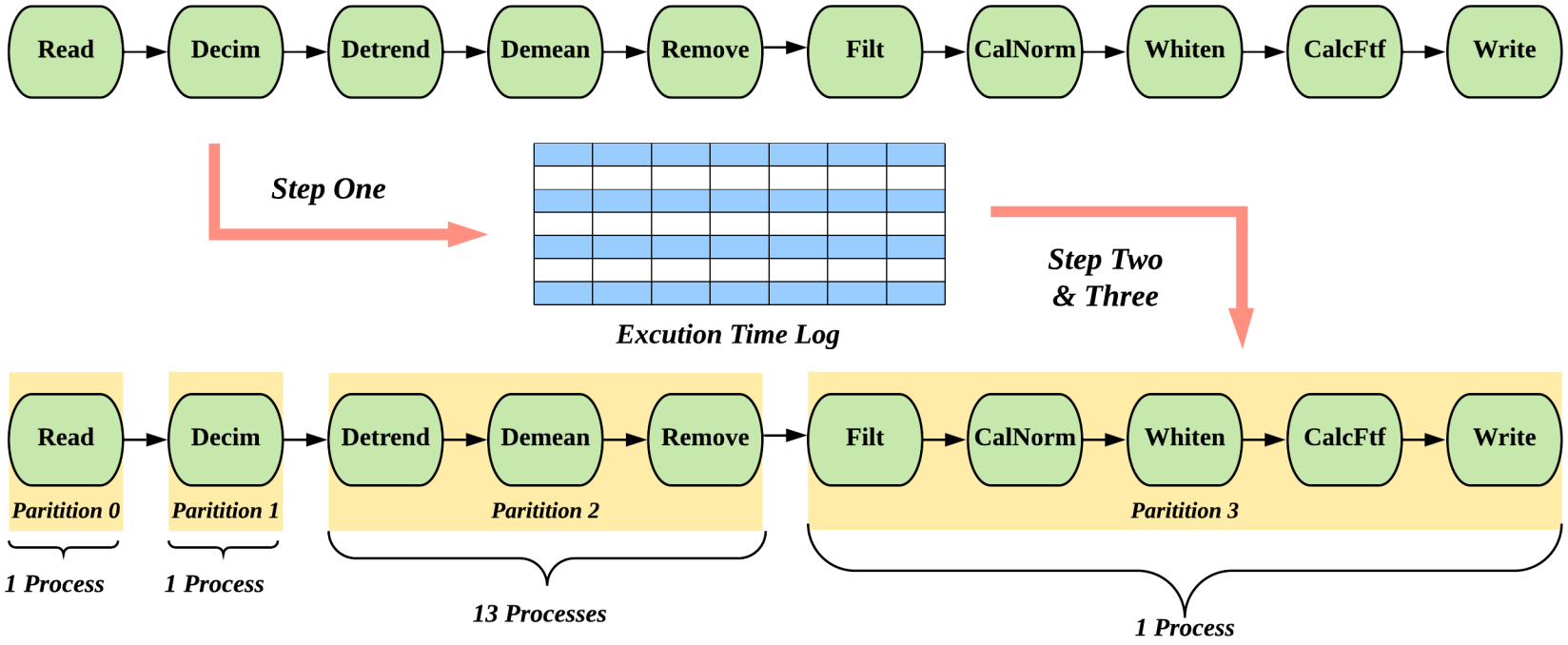
Uses Cases: Seismology Cross-correlation

Assess the risk volcanic eruptions and earthquakes



Naïve Assignment Algorithm

- Identifying the suitable number of partitions
- Appropriate number of processes to assign to each partition
- Based on result of previous executions logs



16 processes

Naïve Assignment Algorithm

Algorithm 1: Assigning Partition

Require: Workflow consisting of N PEs (PE_0, PE_1 ... PE_{N-1})
Require Execution time of each PE as $E(PE_i)$
Require: Communication time between adjacent PEs as $C(PE_i, PE_{i+1})$
for $i = 0$ **to** $i =$ N-2 **do**
  if $i = 0$ **then**
    PE_i is assigned to single partition
  else
    if $C(PE_i, PE_{i+1}) >$ MIN($E(PE_i)$, $E(PE_{i+1})$) **then**
      PE_i and PE_{i+1} are assigned to the same partition or PE_{i+1} is added into the existing partition which PE_i is in
    end if
  end if
end for

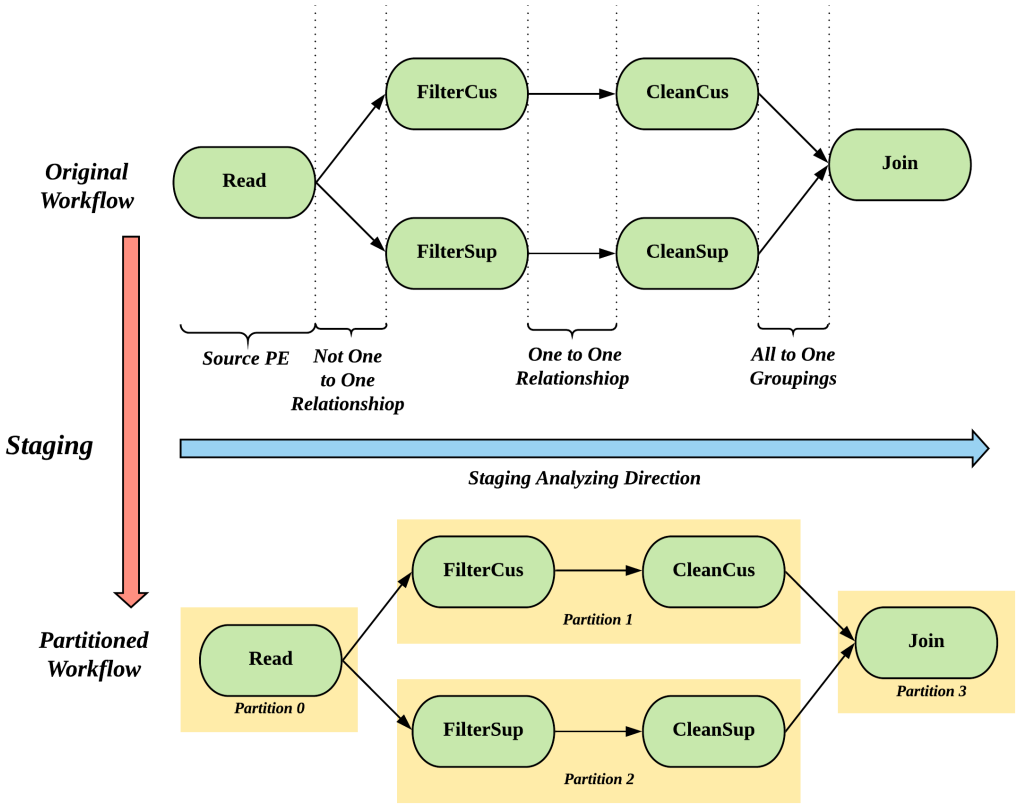
Algorithm 2: Assigning Process

Require: Workflow consisting of M $PARTs$ ($PART_0$, $PART_1$ ... $PART_{M-1}$) or including N PEs (PE_0, PE_1 ... PE_{N-1})
Require: Total number of processes $TotalNumProcess$
Require: Execution time of each PE as $E(PE_i)$
Define: Execution time of each partition as $E(PART_i)$
Define: Number of processes for each partition $NumProcess(PART_i)$
Define: Total execution time $E(TOTAL)$
for $i = 1$ **to** $i =$ N-1 **do**
  $E(TOTAL) = E(TOTAL) + E(PE_i)$
end for
for $i = 0$ **to** $i =$ M-1 **do**
  if $i = 0$ **then**
    $NumProcess(PART_i) = 1$
  else
    for PE in $PART_i$ **do**
      $E(PART_i) = E(PART_i) + E(PE)$
    end for
    $NumProcess(PART_i) = (TotalNumProcess - 1) \times \frac{E(PART_i)}{E(TOTAL)}$
  end if
end for

- Identifying the suitable number of partitions (Naïve 1)
- Calculating number of processes (Naïve 2) to assign to each partition calculated in Naïve 1

Staging Algorithm

- Allocates PEs into the same partition to reduce the communication cost
- Group in the same partition PEs that do not shuffle data



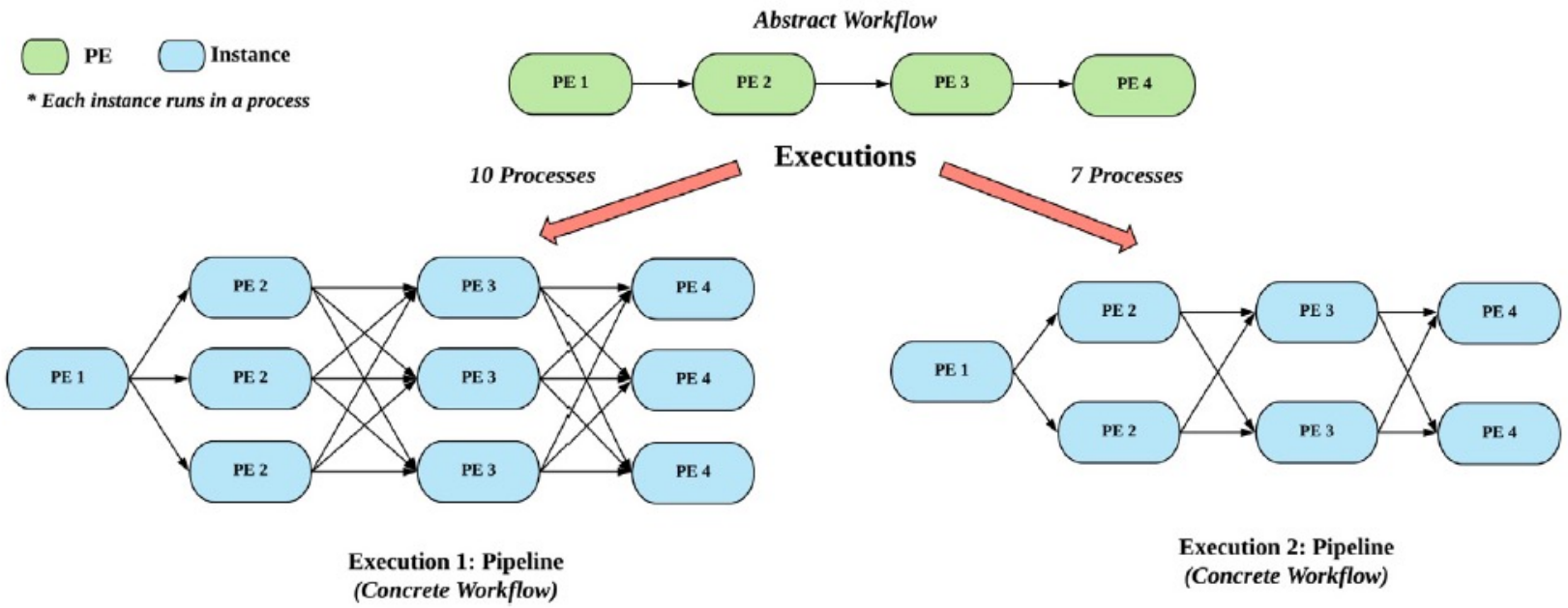
Partitions containing as many neighbouring (in the lineage graph) PEs without shuffles – Inspired by Apache Spark

Dynamic Deployment Techniques

- **Motivation**: Stream-based Scientific workflows are essential
- **Challenge**: Frameworks have limitations – many use static deployment
- **Aim –** Dynamic deployment
- **How -** from static to dynamic deployment, autoscaling techniques
- **Two new heuristic –** on top of dispel4py

Optimization towards efficiency and stateful of dispel4py

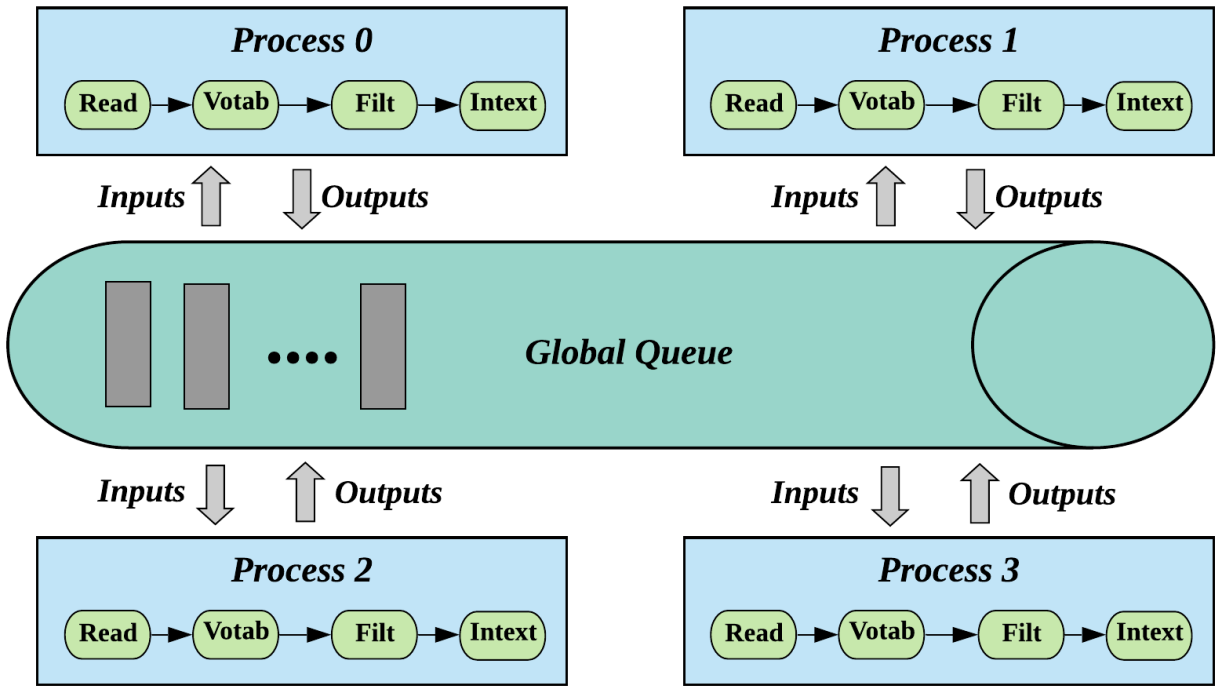
Static Deployment Techniques



dispel4py's static (original) deployment

Dynamic Scheduling Algorithm

- **Dynamic multi mapping** – using multiprocessing library
- Store all PE into a queue
- Available processes take a PE from the queue
- Return results to the queue and wait for next execution

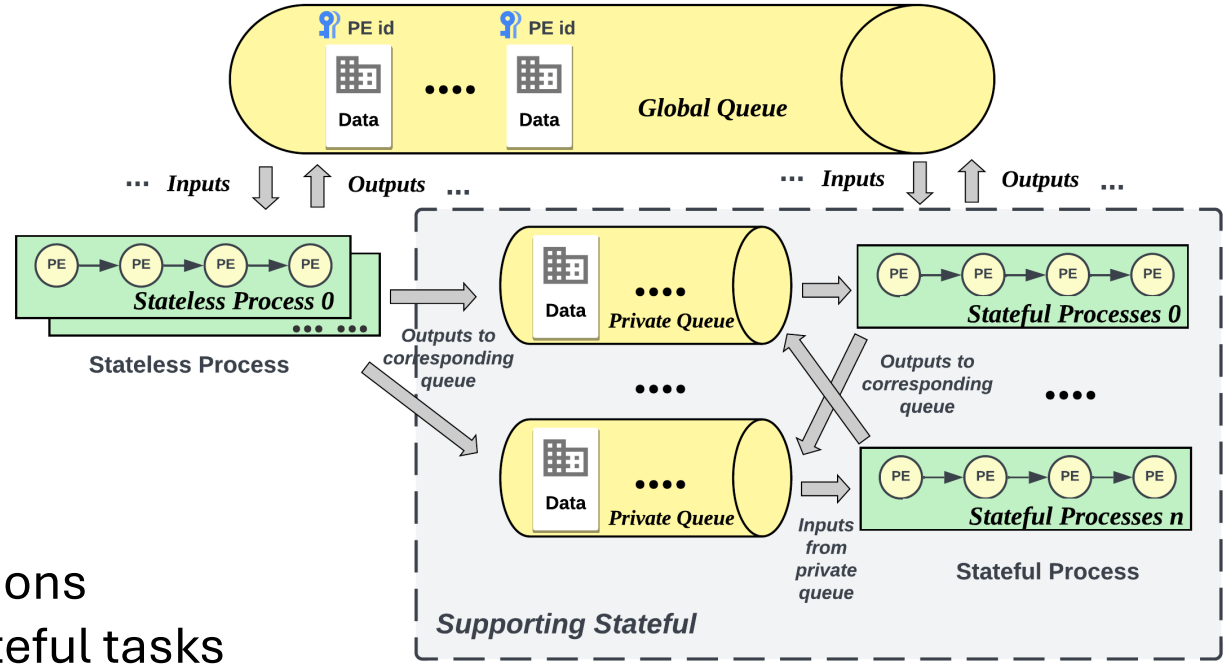


Limitations :

- It does not work with groupings – stateful PEs.
- Only for shared-memory platforms

New Redis Mappings

- Redis
  - Effective data management
  - Redis stream supports real-time sequence
- Dynamic Redis mapping
  - Similar to multi dynamic mapping
  - Replace global queue with Redis Stream
- Hybrid Redis mapping
  - Address the requirements of stateful applications
  - Integrate private queue and processes for stateful tasks



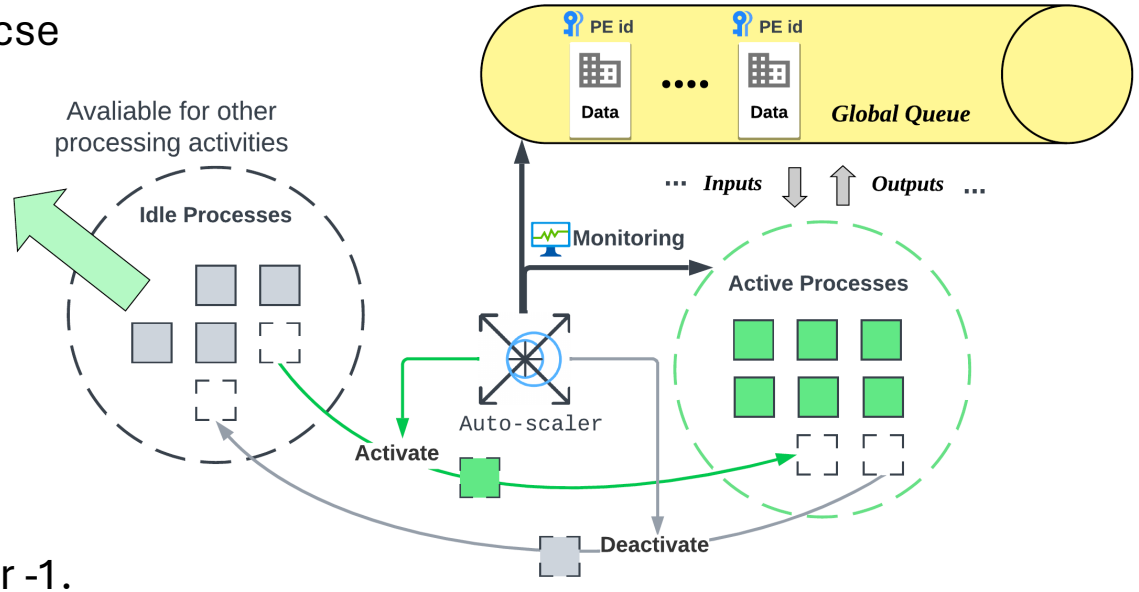
Hybrid Redis Mapping

Dynamic Deployment Techniques

- Auto-scaling optimization address the efficient allocation of resourcse
- Works with both MULTI and REDIS mappings

- When to scale (Monitoring framework)
  - MULTI mapping: Monitoring the queue states and compare with the threshold.
  - REDIS mapping: Monitoring the idle time for active processes and compare with the threshold

- How to scale (scaling strategy)
  - Naïve incremental strategy: incrementing the active size by 1 or -1.



Summary

dyn_multi : dynamic Multiprocessing mapping
dyn_auto_multi : dynamic auto-scaling Multiprocessing mapping
dyn_redis : dynamic Redis mapping
dyn_auto_redis : dynamic auto-scaling Redis mapping
hybrid_redis : hybrid Redis mapping

Dynamic Deployment Techniques

Seismic Preparation Workflow

d4py_workflows / seismic_preparation /

Run the workflow with different mappings

Simple mapping

```
python -m dispel4py.new.processor simple realtime_prep_dict.py -f xcorr_input.jsn
```

OR

```
dispel4py simple realtime_prep_dict.py -f xcorr_input.jsn
```

(Fixed) MPI mapping

```
mpiexec -n 10 dispel4py mpi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

OR

```
mpiexec -n 10 --allow-run-as-root --oversubscribe dispel4py mpi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

OR

```
mpiexec -n 10 python -m dispel4py.new.processor dispel4py.new.mpi_process realtime_prep_dict.py -f xcorr_input.jsn -i
```

(Fixed) Multi mapping

```
python -m dispel4py.new.processor multi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

OR

```
dispel4py multi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

Dynamic Multi mapping

```
python -m dispel4py.new.processor dyn_multi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

OR

```
dispel4py dyn_multi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

Dynamic autoscaling multi mapping

```
python -m dispel4py.new.processor dyn_auto_multi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

OR

```
dispel4py dyn_auto_multi realtime_prep_dict.py -f xcorr_input.jsn -n 10
```

Redis mappings

Remember, you need to have installed both, redis server and redis client.

(Fixed) Redis mapping

> Go to another terminal for following command line

```
redis-server
```

> Go back to previous terminal

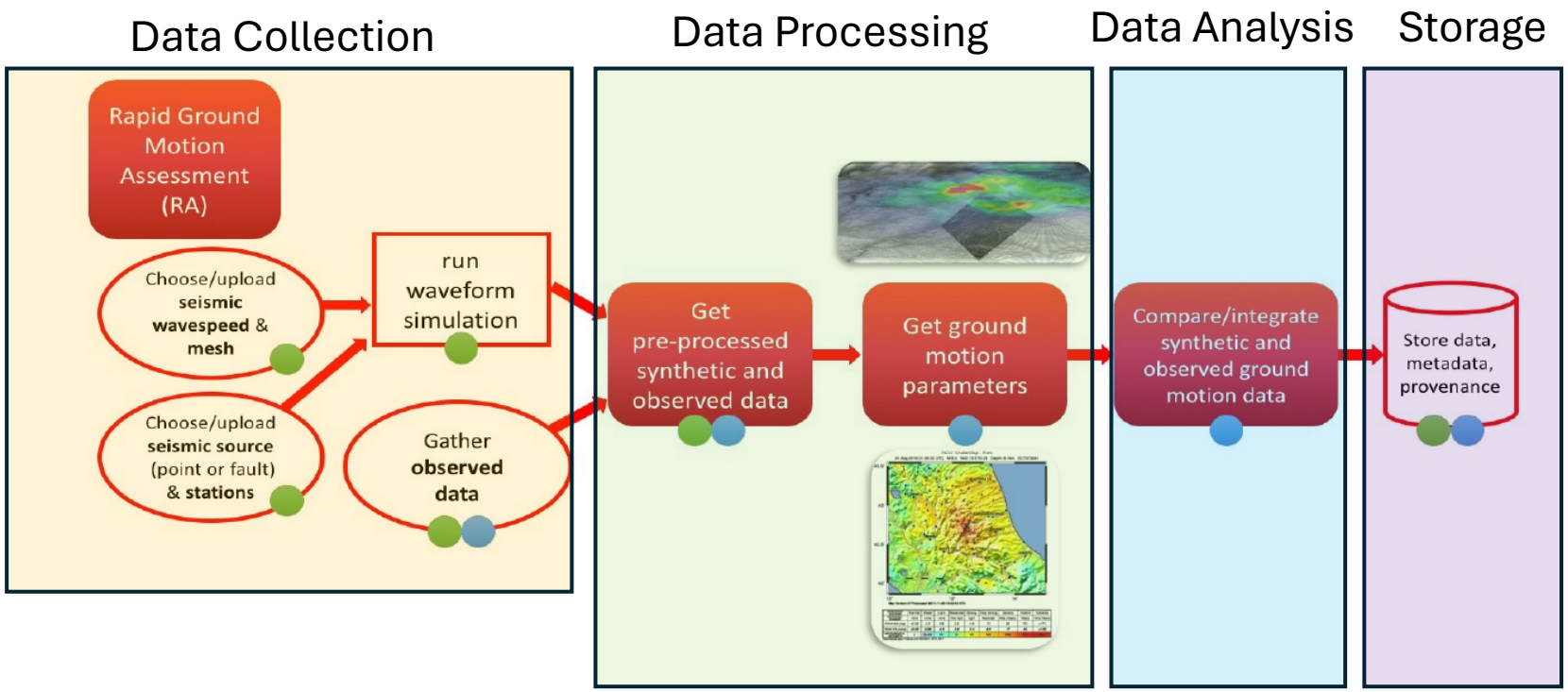
```
python -m dispel4py.new.processor redis realtime_prep_dict.py -f xcorr_input.jsn  -ri localhost -n 10
```

OR

```
dispel4py redis realtime_prep_dict.py -f xcorr_input.jsn -ri localhost -n 10
```

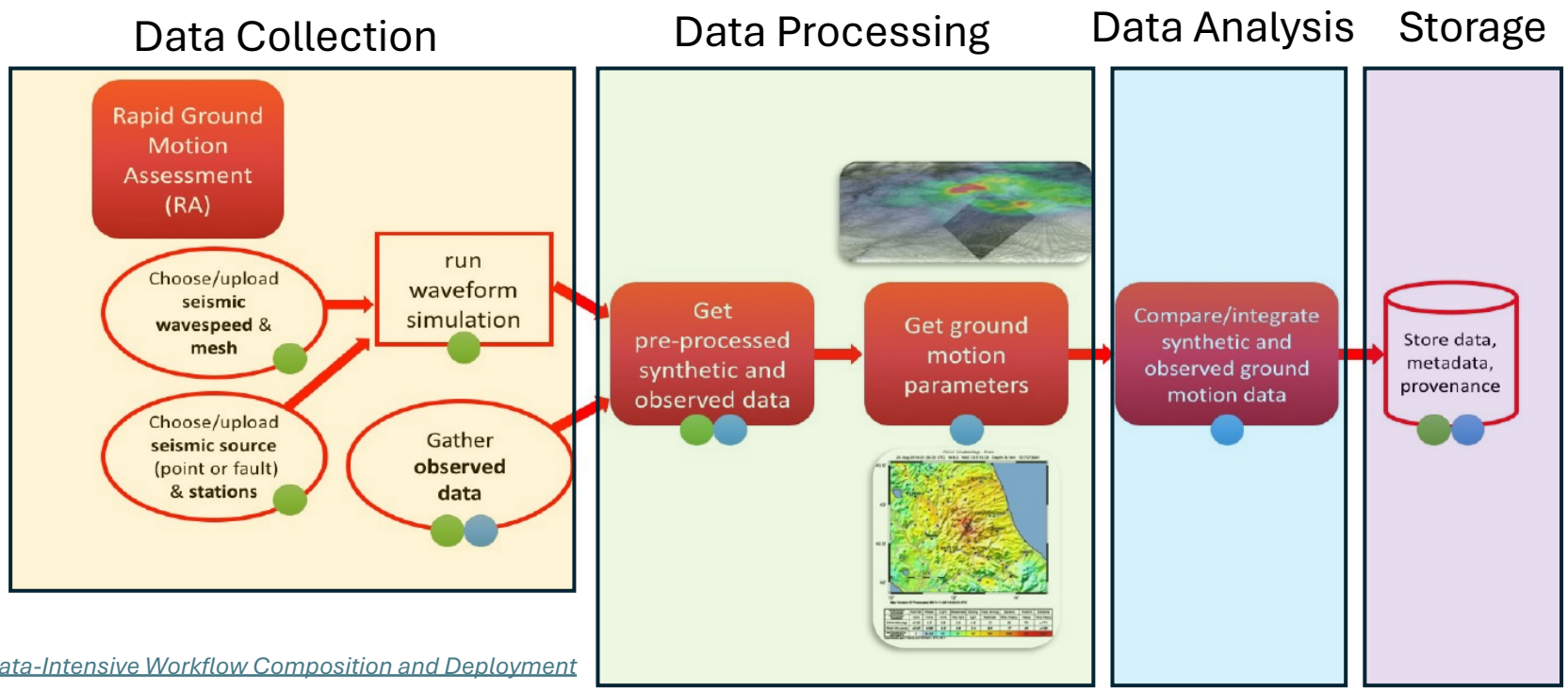
dispel4py & CWL

- Seismological Example:
  - Quickly analyze earthquakes
  - Model the ground motion after earthquakes
  - Rapid assessment of earthquakes' impact, and emergency response



dispel4py & CWL

- Using dispel4py for implementing the RA subcomponents of the application
- Using cwl for connecting the different dispel4y RA subcomponents



dispel4py & CWL

- Using dispel4py for implementing the RA subcomponents of the application
- Using cwl for connecting the different dispel4y RA subcomponents

ra.cwl

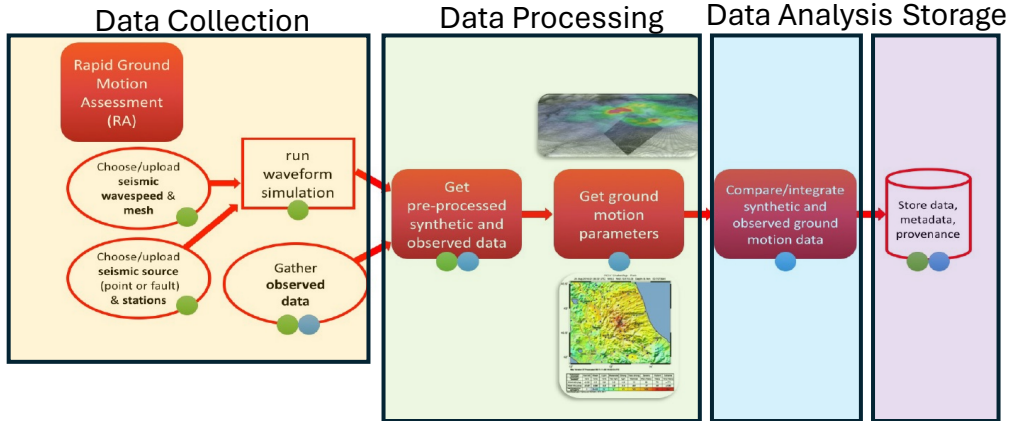
```
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: Workflow

inputs:
   create_env_script: File
   download_workflow: File
   download_argument_f: File
   preprocess_workflow: File
   ra_workflow: File
   ra_argument_d: string
outputs:
    misfit_data:
        type: Directory
        outputSource: preprocess_data/output
    pgm_data:
        type: Directory
        outputSource: rapid_assessment/output
steps:
  create_env:
    run: create_env.cwl
    in:
      script: create_env_script
    out: [output]
  download_data:
    run: dispel4py_download.cwl
    in:
      workflow:  download_workflow
      argument_f: download_argument_f
      misfit_data: create_env/output
    out: [output]
  preprocess_data:
    run: dispel4py_preprocess.cwl
    in:
      workflow: preprocess_workflow
      misfit_data: download_data/output
    out: [output]
  rapid_assessment:
    run: dispel4py-RA-pgm_story.cwl
    in:
      workflow: ra_workflow
      argument_d: ra_argument_d
      misfit_data: preprocess_data/output
    out: [output]
```

ra.yml

```
create_env_script:
    class: File
    path: create_env.sh
    secondaryFiles:
      - class: File
        location: processing.json
      - class: Directory
        location: misfit_data

download_workflow:
    class: File
    path: download_FDSN.py
    secondaryFiles:
    - class: File
      location: download_helpers.py
    - class: File
      location: domain.py
    - class: File
      location: utils.py
download_argument_f:
    class: File
    path:  download_chile.json
preprocess_workflow:
  class: File
  path: create_misfit_prep.py
  secondaryFiles:
    - class: File
      location: preprocessing_functions.py
preprocess_argument_f:
    class: File
    path: misfit_input.jsn
ra_workflow:
  class: File
  path: dispel4py_RA_pgm_story.py
ra_argument_d: '{"streamProducerReal": [ {"input": "output/IV.ARRO.EHR.data"} ], "streamProducerSynth":
[ {"input": "output/IV.ARRO.HXR.synth"} ]}'
```

```
$ cwltool ra.cwl ra.yml
```

Available here



Acknowledgements

- *Optimization towards Efficiency and Stateful of dispel4py*
- *Scalable adaptive optimizations for stream-based workflows in multi-HPC-clusters and cloud infrastructures*
- Laminar: A New Serverless Stream-based Framework with Semantic Code Search and Code Completion
- DARE Platform: Enabling Easy Data-Intensive Workflow Composition and Deployment