International Centre for Radio Astronomy Research

Making 100 Million Parallel Tasks User Friendly

The DALiuGE System

流

Andreas Wicenec

& The ICRAR team

Curtin University

THE UNIVERSITY OF WESTERN AUSTRALIA

Government of Western Australia
Department of the Premier and Cabinet
Office of Science

The *DALiuGE System*

A scalable, graph oriented wor
development, scheduling and e
system.

DALiuGE: Motivation

流

Lower boundary for maximum number of tasks

$$2^{16} * 300 * 4 = 78{,}643{,}200$$

Channels Facets Polarisations

DALiuGE

流

‘Scalabale’ for DALiuGE design means t
about whether a feature could scale and
asking again whether it really scales.

Result is a share nothing and distributed design and implementation *almost* everywhere!

DALiuGE: Key Concepts 流

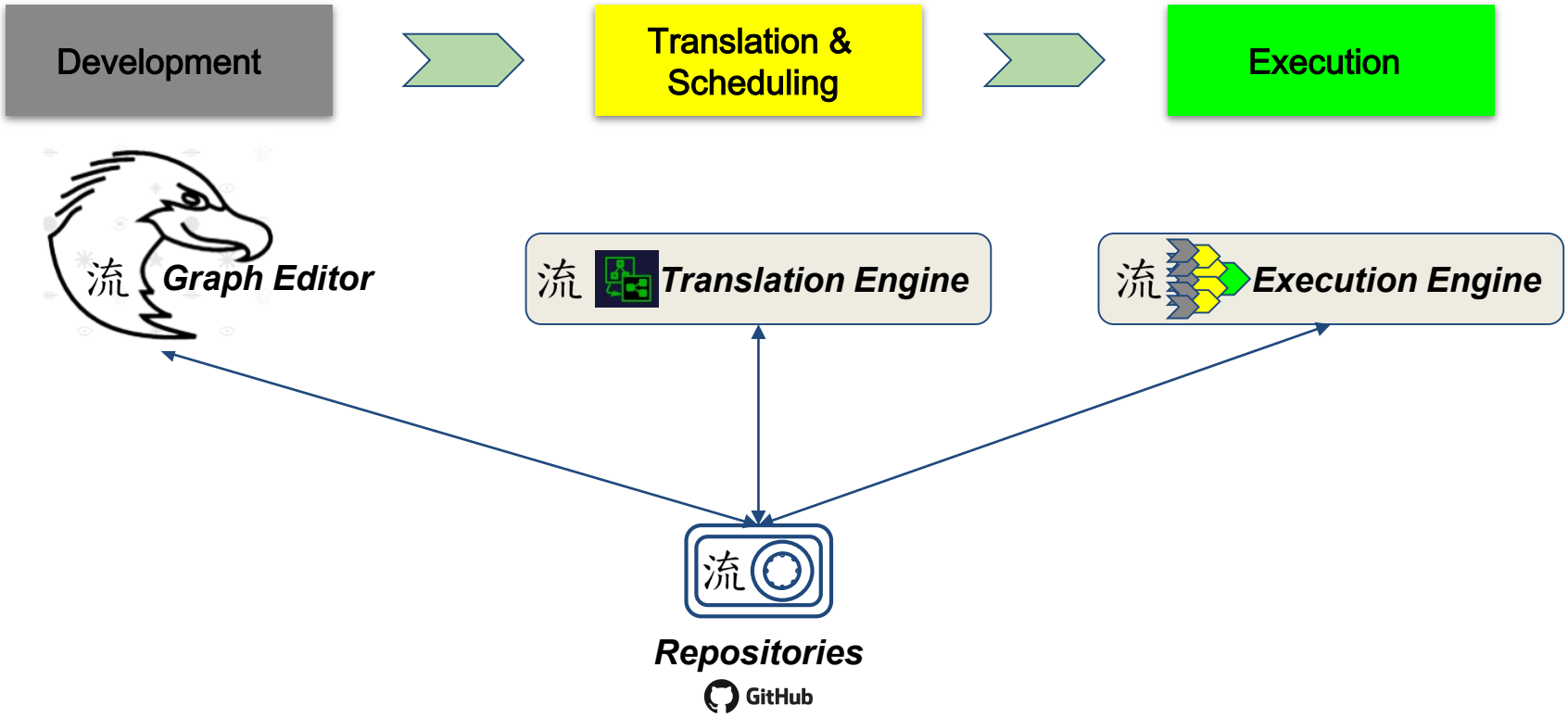
- **Traditional Dataflow**
  - Executability and execution of instructions is solely determined based on the availability of input arguments to the instructions, so that the order of instruction execution is unpredictable: i. e. behavior is nondeterministic
  - Opens up new parallelism opportunities previously masked by application flow control
  - An example: *Makefile*

- **Dataflow Graph**
  - **nodes** (stateless functions, aka operators/actors), **data** and **tokens** travelling across directed edges to be transformed at nodes into other data items
  - need control flow operators and data storage to make it practical and useful e.g. the MIT dataflow architecture [1]

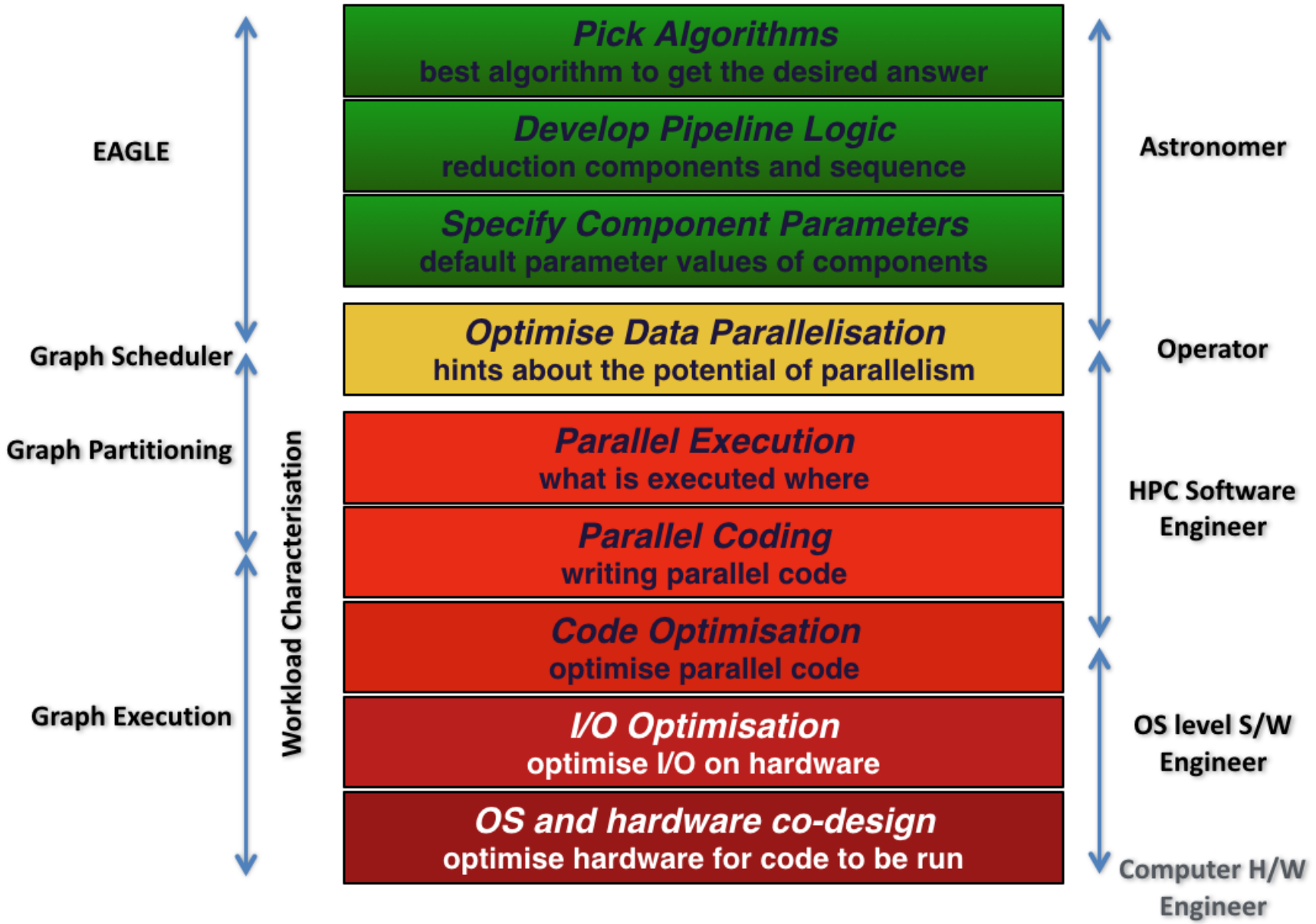
- **Data-triggered Graph**
  - elevate data as graph **Nodes**
  - instead of applications (instructions) polling availability of input, data objects trigger applications based on their own internal (persistable) states
  - what is really “moving along the directed edges” are “**Drop events**” rather than data items.

The DALiuGE System

Stratospheric View



Separation of Concerns



Concepts

Development

流

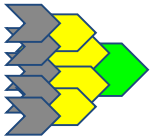
Domain Specific Language ⇋ ***Palette***

Workflow or Pipeline ⇋ ***Logical Graph***

Directed Acyclic Graph ⇋ ***Physical Graph***

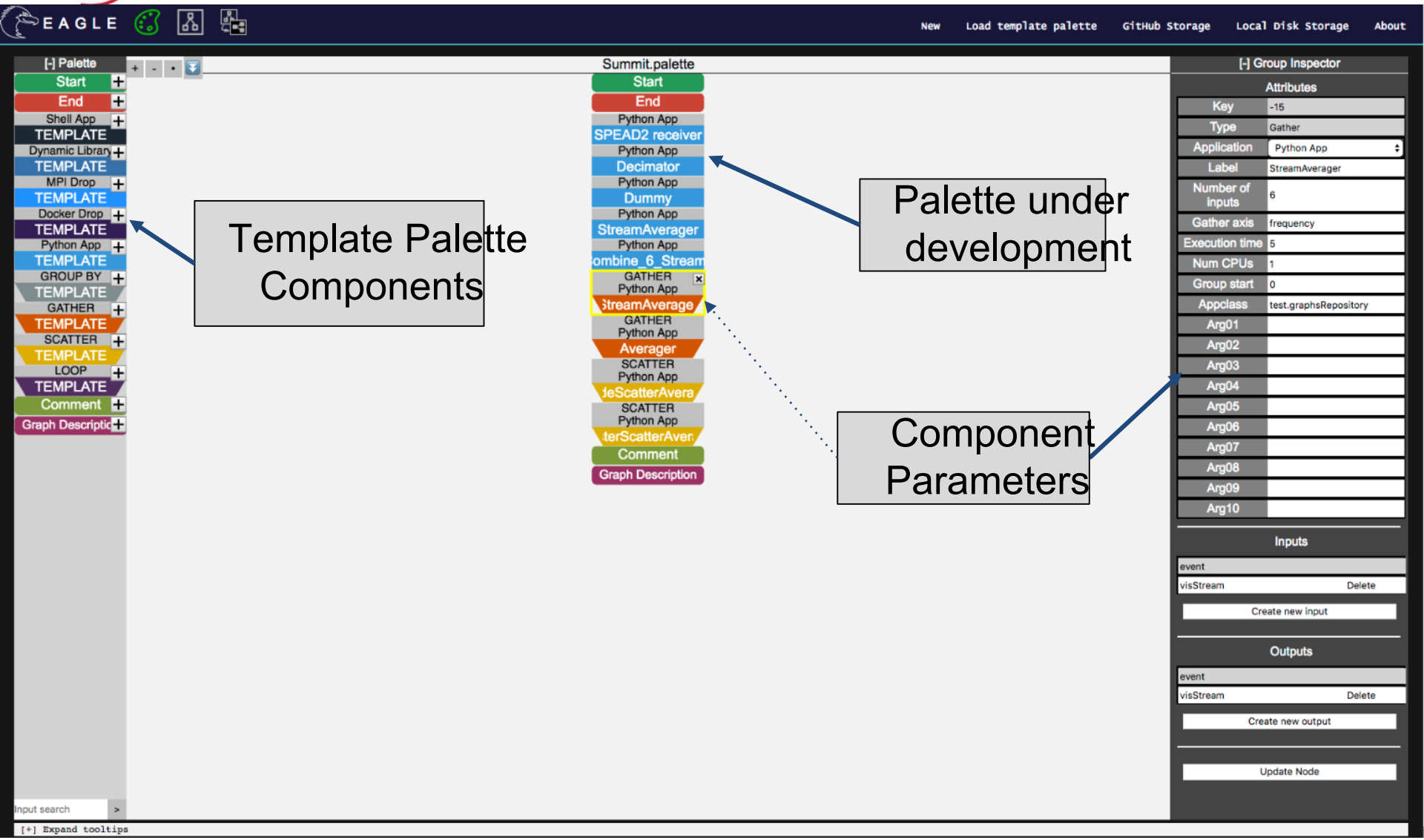
Steps

Development 流

Workflow Components	Workflow Logic	Parameterisation	Translation	Execution
				
Wrap 'your' algorithms into DALiuGE components using an API and define palette components, including their variable parameters. Save the new ***palette*** to gitHub.	Using the palette defined in the previous step construct the logic of the workflow. Save the new ***logical graph template*** to gitHub.	Populate the variable parameters of the components. This will transition the logical graph template to a ***logical graph***.	Submit logical graph to the translator. The translator will generate a ***physical graph template***.	Submit the physical graph template to the execution engine on a cluster. The execution engine will populate the addresses of the compute nodes and generate the ***physical graph***, which will then be executed.
Infrequent	Once per workflow (version)	Once per workflow use case definition	Once per workflow use case platform	Once per workflow use case execution

Palette

Development



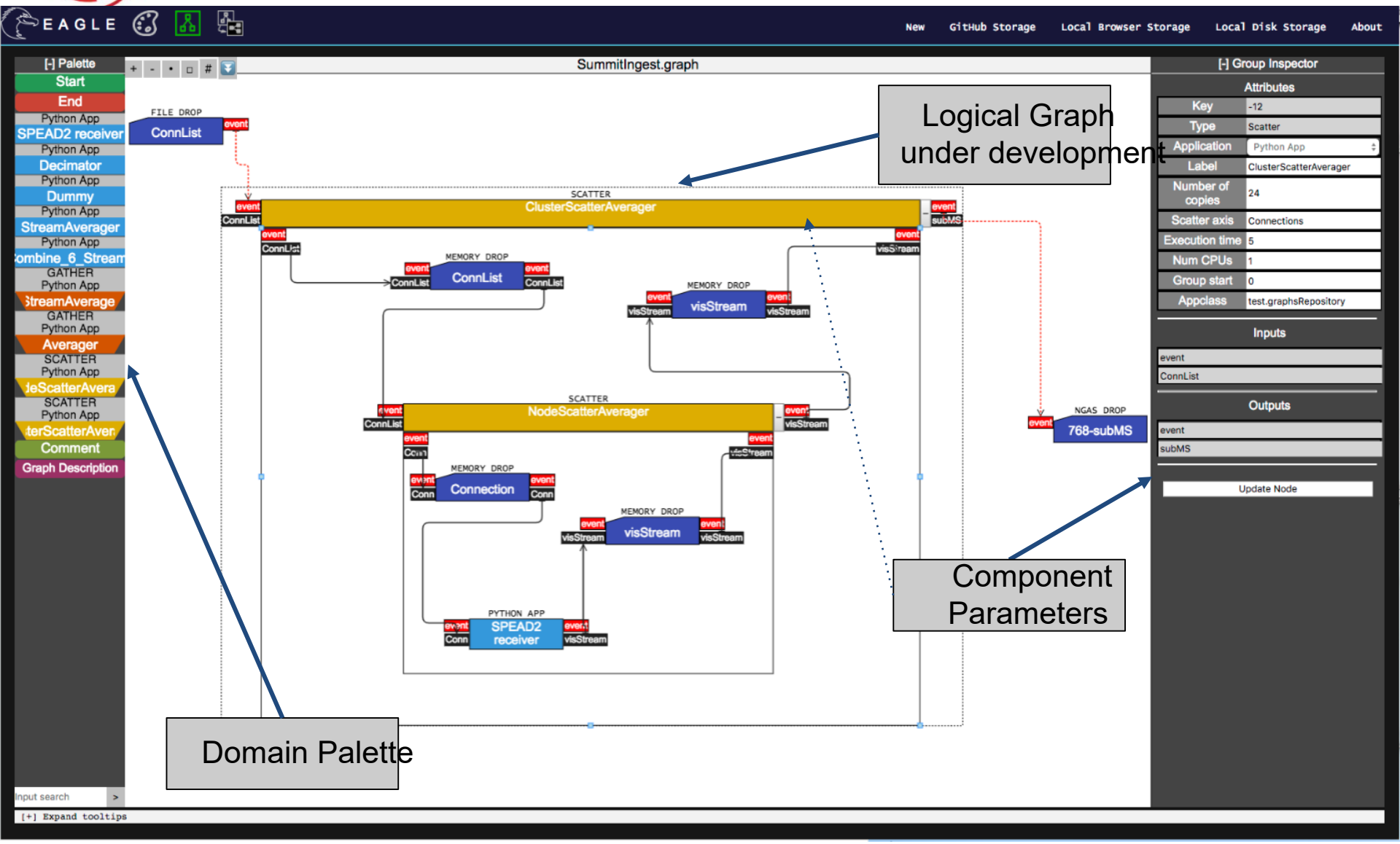
Template Palette Components

Palette under development

Component Parameters

Workflow

Development



Parameterisation

Translation & Scheduling

- ➔ Why??
- ➔ Logical Graph Template represents processing *mode*, e.g. spectral line, continuum, fast imaging, pulsar timing.
- ➔ Actual processing run requires to specify *configuration* or *parameters* for such a *mode*, e.g. number of channels, averaging algorithm, weighting.
- ➔ Can be done manually, but in actual operations these parameters will be filled by processing blocks.
- ➔ Manual filling really just for testing.
- ➔ The available mutable parameters (as well as their defaults) are defined during the creation of palette components.

Parameterisation

Translation & Scheduling

- Why??
- Logical Graph Template represents processing *mode*, e.g. spectral line, continuum, fast imaging, pulsar timing.
- Actual processing run requires to specify *configuration* or *parameters* for such a *mode*, e.g. number of channels, averaging algorithm, weighting.
- Can be done manually, but in actual operations these parameters will be filled by processing blocks.
- Manual filling really just for testing.
- The available mutable parameters (as well as their defaults) are defined during the creation of palette components.

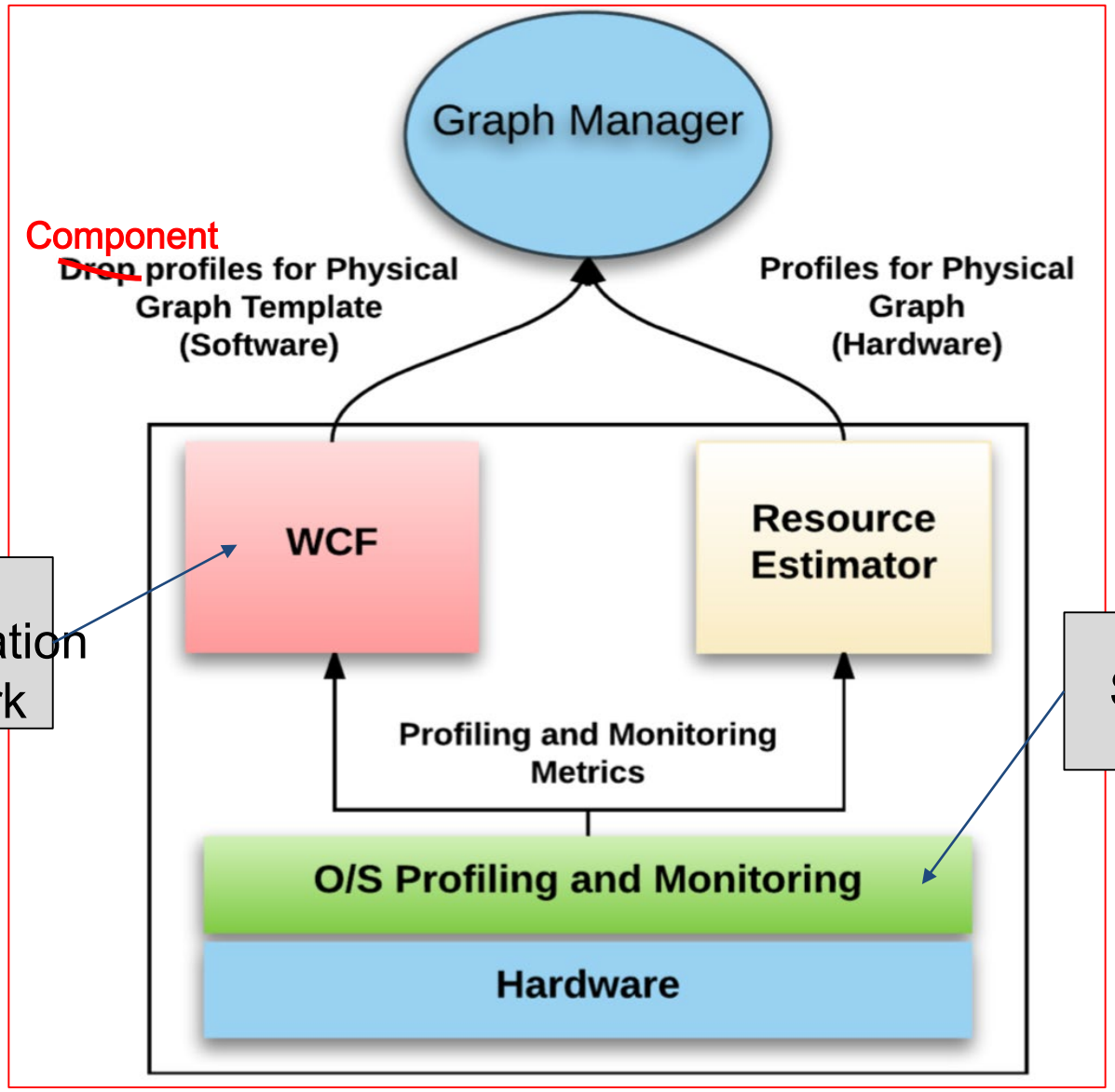
Graph Scheduling

Translation & Scheduling

- Multi-step processing pipeline can be represented as Directed Acyclic Graph (DAG): Physical Graph.
- Hardware capabilities in a distributed compute resource can also be represented as DAG (includes network and I/O costs).
- Requirements of actual tasks (memory, FLOPs) are constraints.
- The description of a pipeline must be independent of hardware.
  - In other words: If you could, you would probably use an infinitely fast computer with infinite amounts of memory and I/O capabilities.
- Usage of very large clusters with very many cores is a necessity, not a choice.
- Hardware will change faster and pretty much out-of-sync with software.
- *Approach: Decouple logical pipeline from physical deployment and let computers perform the optimisation.*
- *This is commonly referred to as workflow or graph scheduling and subject to very active research.*

Graph Scheduling

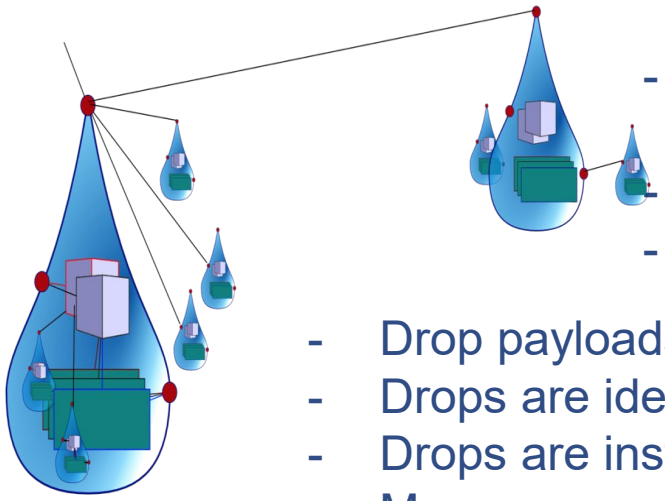
Translation & Scheduling



Workload Characterisation Framework

Separate modu

DALiuGE: Drop Concept 流



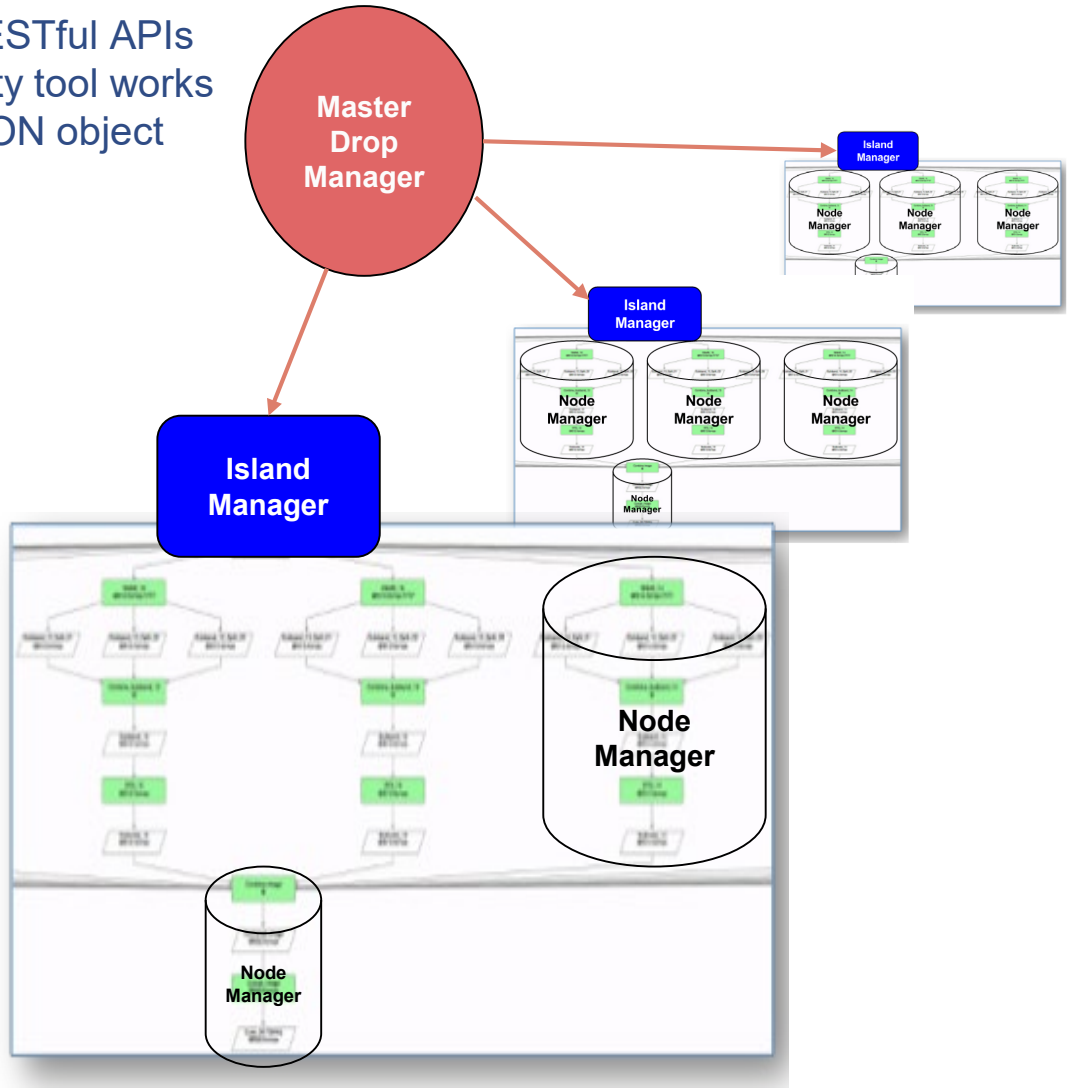
- Drops are ‘software objects’.
- Drops define standard interfaces and wrap a payload.
- Drop payloads can be data, applications and other Drops.
- There are multiple realisations of Drops.
- Drops can raise and consume Drop-Events.

- Drop payloads can be referenced by URIs.
- Drops are identified by globally unique IDs.
- Drops are instantiated, monitored and destroyed by Drop Managers.
- Drops ‘know’ their managers, *producers* and *consumers*.
- Drops and their payload are checksummed.
- Drops follow a life-cycle.
- Drops change state and can be persisted into a sleep state.
- Data Drops are write once, read multiple times.
- Producer and Consumer Application Drops subscribe to Data Drop Events.
- Managers subscribe to Drop-Events.

Graph Execution

Execution 流

- Drop Managers
  - Hierarchical structure, the same RESTful APIs
    - Clients provided, any third-party tool works
  - Receives the physical graph as JSON object
  - Simple Web UI
- Node Drop Manager (`NDM`)
  - Manages individual *Sessions*
  - One per node, long-lived
- Data Island Drop Manager (`DIDM`)
  - Distributes physical graph to NDMs
  - Aggregates info from NDMs
  - not required for small deployments
- Master Drop Manager
  - Routes physical graphs to DIDMs
  - Aggregates info from DIDMs

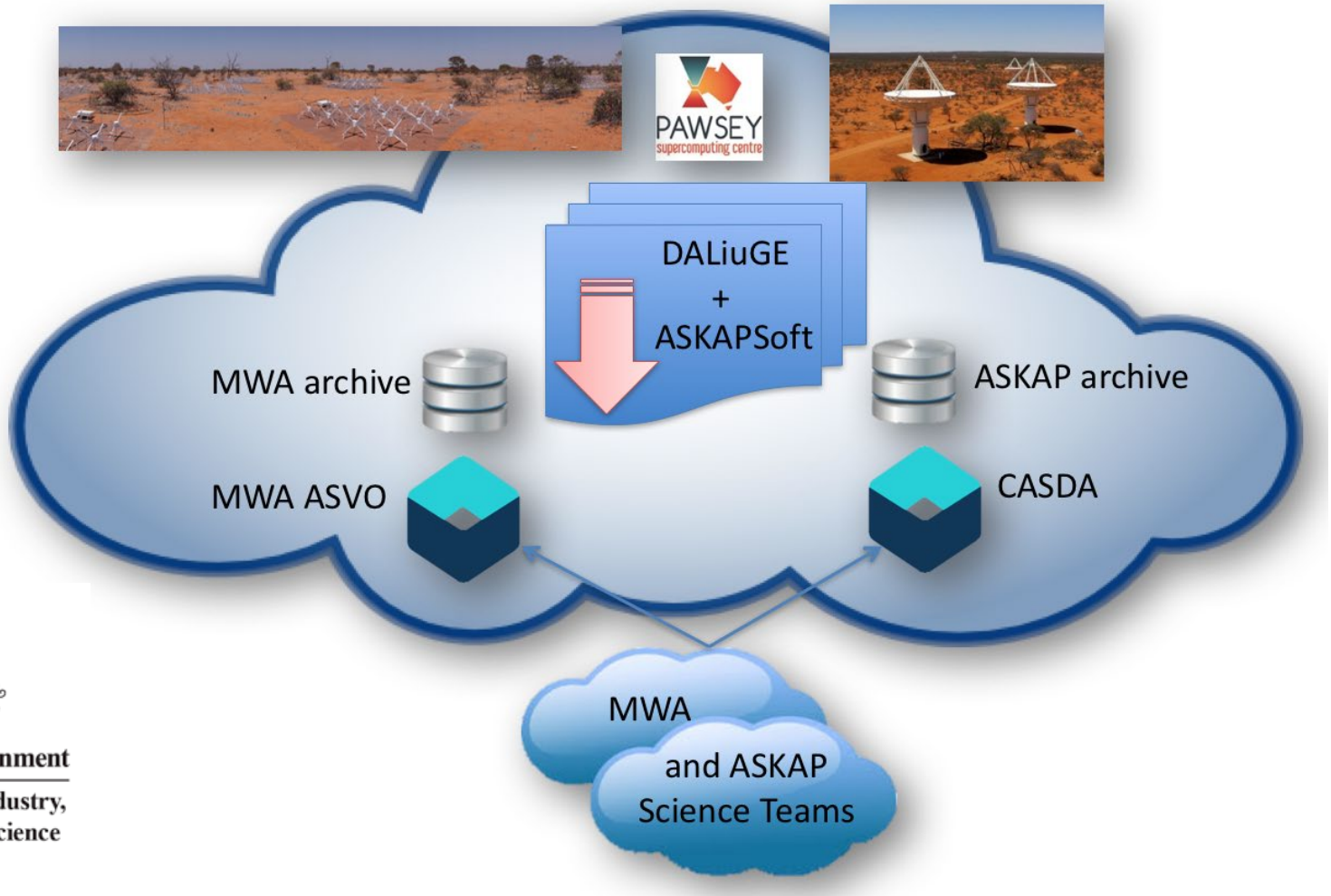


DALiuGE: Advanced Stuff 流

- The Drop abstract class is used for both applications and data ⇒ Data becomes first-class citizen and driving execution.
- Transitioned from compound components like Scatter, Gather and Loop to very generic MKN-components.
- In future we are thinking about an introspection mechanism to populate palette components for EAGLE.
- EAGLE implements ‘type’ safety for component connections (can’t connect, if not matching).
- DALiuGE supports streaming for all I/O data components.
- We only very recently managed to terminate and kill a running graph!
- Trying to implement a double recursive sort algorithm as challenge for non-graph friendly problem.

Status & Outlook

JACal SKA SDP = ASKAPSoft + MWA pipeline + DALiuGE

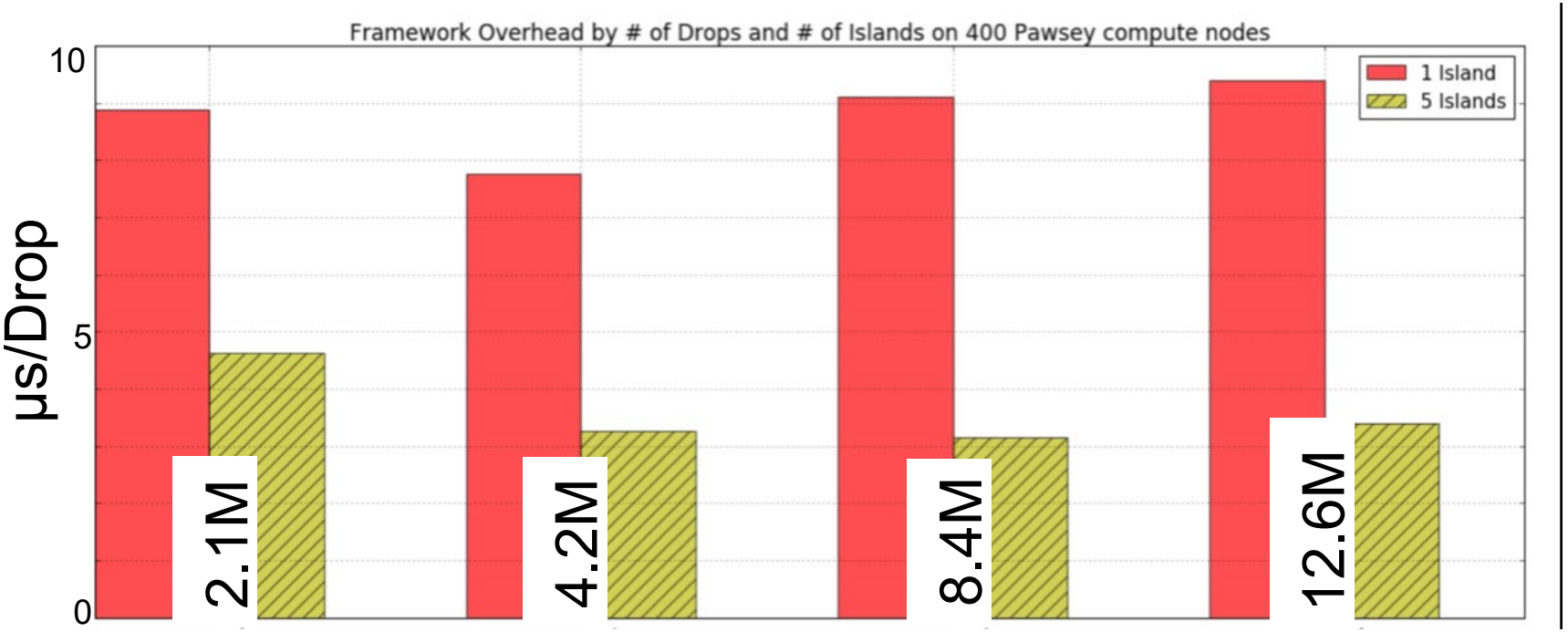


Funded by:

Australian Government
Department of Industry, Innovation and Science

Does it Scale?

- We have executed a whole series of scalability tests on various platforms including Tianhe-2.
- Measured plain overhead imposed by DALiuGE during execution.



Does it Scale?

流

- Yes!

- Test run: 12.6 Million tasks on 400 compute nodes means 31,500 task/node.
- Current expectation for SDP 2,500 nodes running 78M tasks. With actual numbers: 31450 tasks/node.
- Test execution time 420 seconds.
- SDP: several hours.

Problem solved?

流

- Not quite yet!

- It seems to scale, but that's just the framework without real algorithms.
- What's happening when things fail?
- Not production software.
- Scheduling single workflows in a N-P hard problem, scheduling multiple workflows under constraints is even harder (multi N-P hard).
- Without good scheduling overall efficiency will be very low.

DALiuGE: Future work

流

- Reference prototype of SDP execution framework architecture.
- Transitioning from SDP prototype to operational system. Mid-term for ASKAP and MWA. Longer term for SKA1-LOW?
- Development to run a very large scale simulation and reduction deployment (SKA1-LOW scale) on SUMMIT.
- In the process of hiring web application developer to upgrade EAGLE.
- Visualisation of very large running physical graphs under investigation.
- Some of the internal detailed design concepts still not implemented.

DALiuGE: Future work

流

- Drop
  - Drop I/O framework optimisation (RDMA, Island consolidation, etc.)
  - Pipeline component interface expressing ***local dataflow*** capabilities through dynamic GPU / FPGA schedulers
- Graph
  - Graph alteration for dynamic scheduling
    - e.g. Randomisation of *Scatter* partitions
  - Full support for branch condition
  - Support for user defined graph altering components
  - Allow more expressive locality constraints
    - e.g. this Drop must run on GPUs, those two Drops must run together (Graph union-find)
- Deployment
  - Very large scale deployment
  - Multiple platforms
- Monitoring and Optimisation
  - Low latency graph visualisation
  - Integrate Workload Characterisation Framework

The *Data Activated Flow Graph Engine*

Open source, under GitHub and PyPi

https://github.com/ICRAR/daliuge

https://github.com/ICRAR/EAGLE