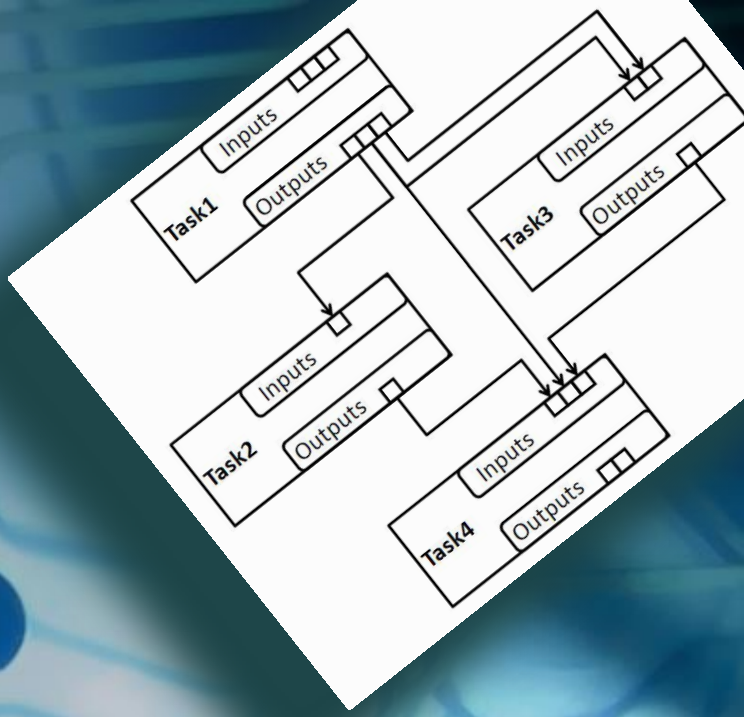
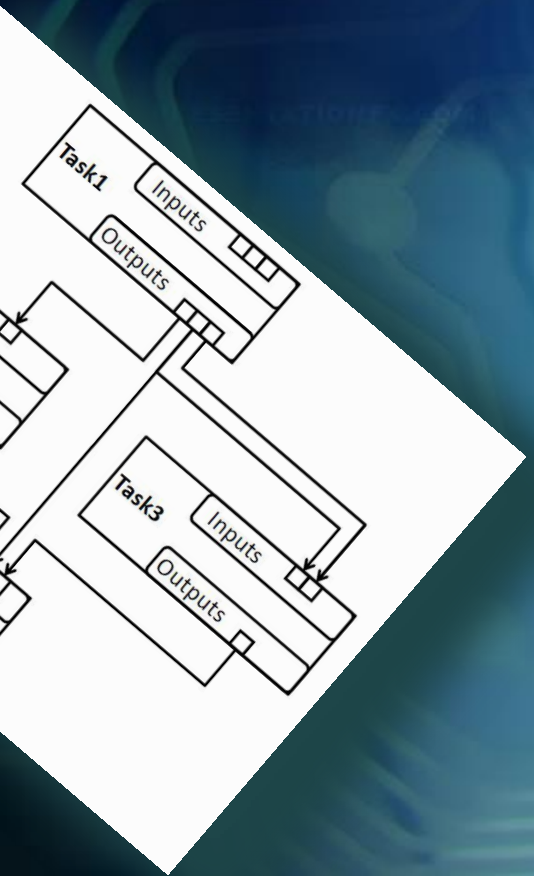
DEFT: status, near term and 2014 plans





Maxim Potekhin
BNL

ATLAS Distributed Computing
Technical Interchange Meeting
May 16, 2013

About this presentation

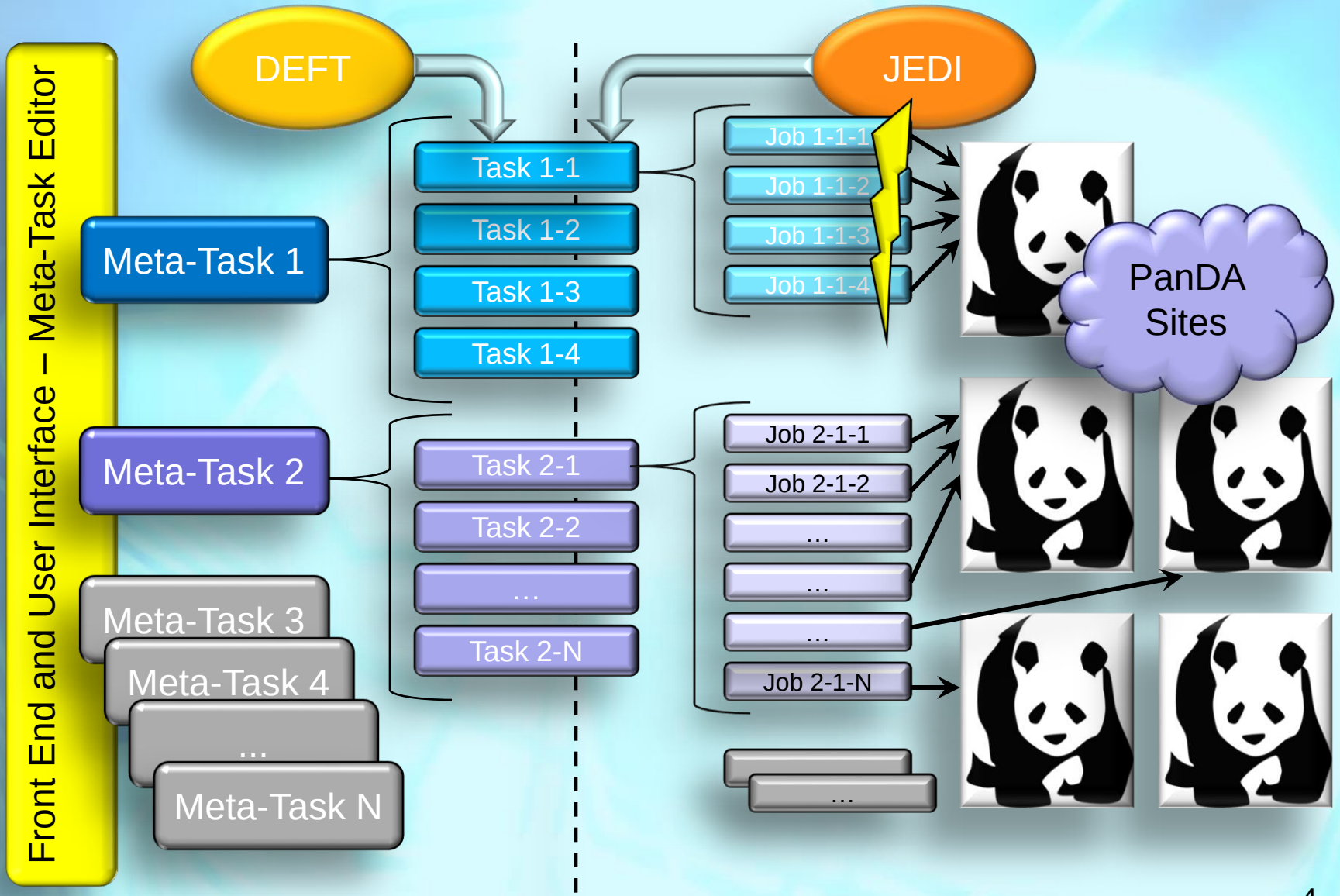
- General plans for the ATLAS Production System evolution and upgrade (ProdSys2) have been worked out about a year ago. Design of its components (DEFT+JEDI) took place in late 2012 – early 2013.
- ”JEDI” will be covered in a separate presentation. The focus of this talk is DEFT, its current status and plans for further development.
- Initial DEFT prototype was ready before the S&C week in March, progress made since then. The crucial DEFT functionality has been tested.
- Important but a subject of a separate presentation:
  - A test version of JEDI (codename JEDI-α) has been tested with appropriate interfaces built to the existing Production System database.
- Substantial effort has been invested in documenting this project
  - https://twiki.cern.ch/twiki/bin/viewauth/Atlas/ProdSys (and its many links!)
  - https://twiki.cern.ch/twiki/bin/viewauth/Atlas/PandaJEDI
  - http://prodsys.blogspot.com
  - Also, ATLAS S&C Workshop: https://indico.cern.ch/conferenceDisplay.py?confId=210656
- Important: The ProdSys2 Technical Design Report (**TDR**) has been created, and will be presented in this meeting – see that document for more detail.

Overview of the context: ProdSys2 = DEFT + JEDI

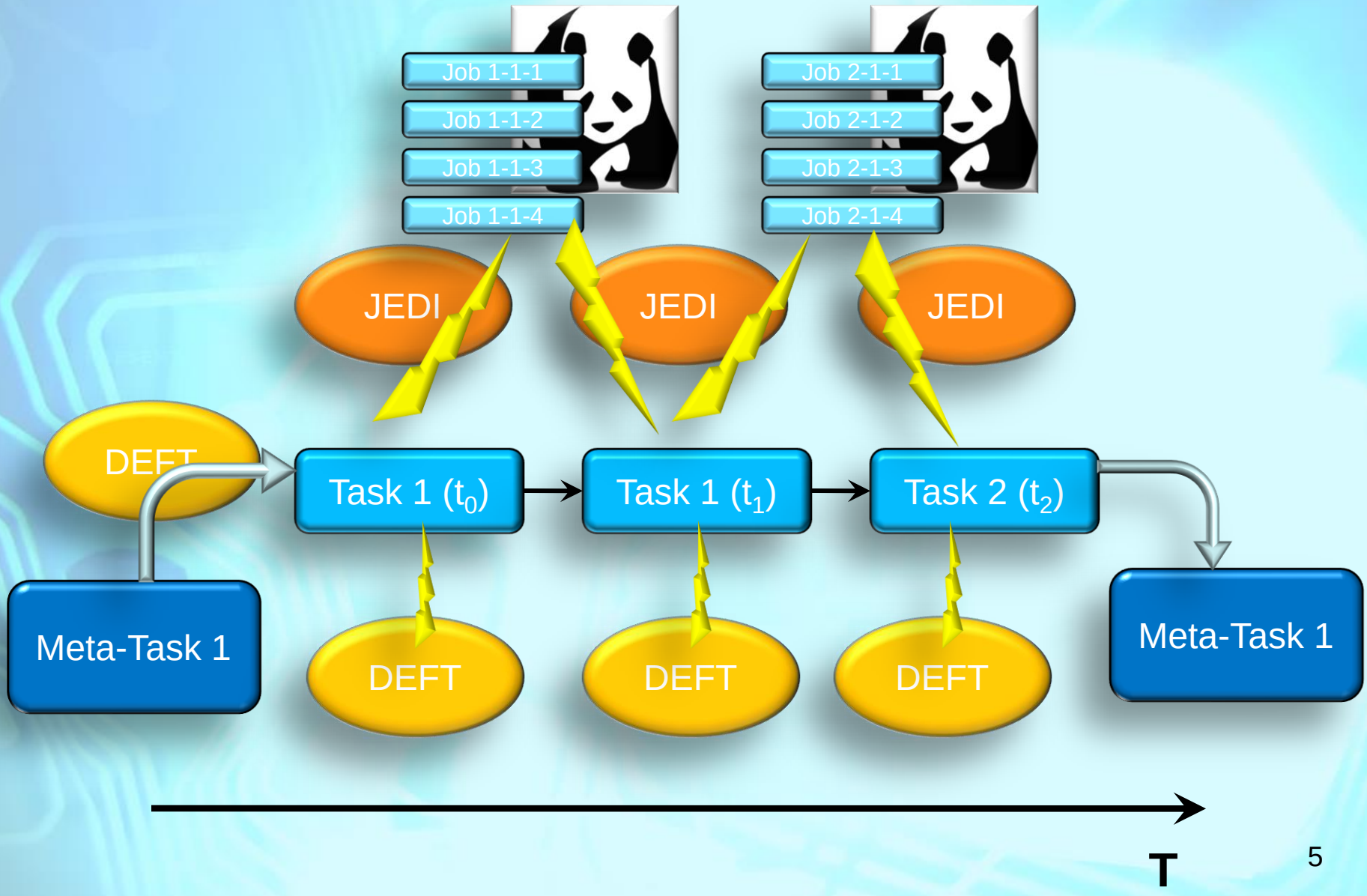
The new Production System (ProdSys2) is a tandem of two subsystems, which play complementary roles and represent two different levels in managing the overall workflow in PanDA :

- DEFT: Database Engine For Tasks:
  - DEFT is responsible for formulating the Meta-Tasks **and** the underlying tasks (collections of jobs). Meta-Tasks can include chains of tasks, bags of tasks and other types of task groupings, complete with all necessary parameters. DEFT keeps track of the state of the Meta-Tasks under its management, and their constituent tasks, by using a database as its persistence mechanism. It accounts for data dependencies among the tasks by maintaining records of datasets and their states. It provides the interface for each Meta-Task definition, management and monitoring throughout its lifecycle.
- JEDI: Job Execution and Definition Interface
  - JEDI is using **the task definitions** formulated in DEFT to define and submit individual jobs to PanDA, keep track of their progress and handle re-tries of failed jobs, job redirection etc. In addition, JEDI interfaces data management services in order to properly aggregate and account for data generated by individual jobs (i.e. general dataset management)

DEFT and JEDI working in tandem



DEFT and JEDI as an assembly line



ProdSys2: Motivations from the DEFT perspective

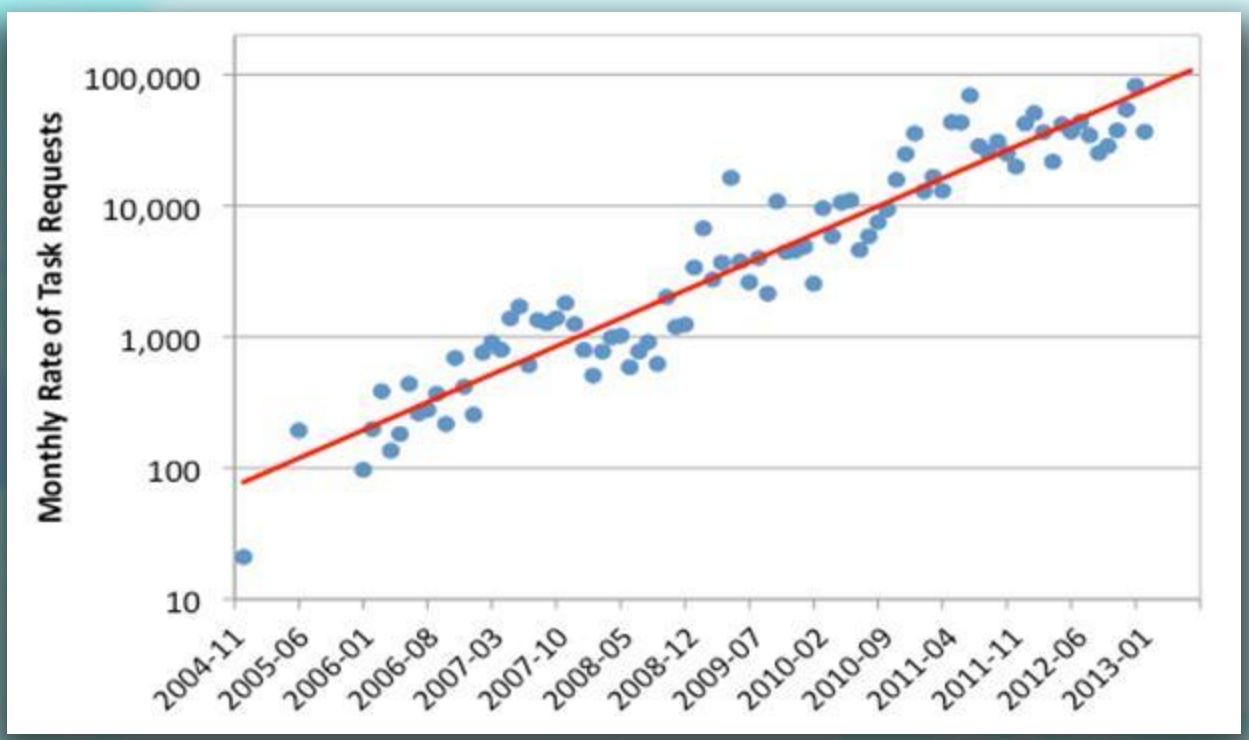
We need to address the following:

- The concept of Meta-Task as a group of logically related tasks, required to complete a specific physics project or sub-project. Absent in the original product (ProdSys), it emerged based on operational experience with PanDA and its workflow. It effectively became the central object in the high-level workflow management, but…
- …it is currently modeled in spreadsheets, which act as surrogate database and GUI, require active maintenance and limit the scalability of the overall Production System.
- Meta-Tasks must be properly modeled, introduced and integrated into the system to guarantee that it delivers adequate scalability and performance going forward, as well as a productive environment for the Production Managers and users involved in the physics analysis.
- Analysis tasks are currently handled outside the Production System, however they also have workflow characteristics. Having a universal system supporting the management of workflows across production and analysis will bring benefits to the ATLAS community.

The Meta-Task

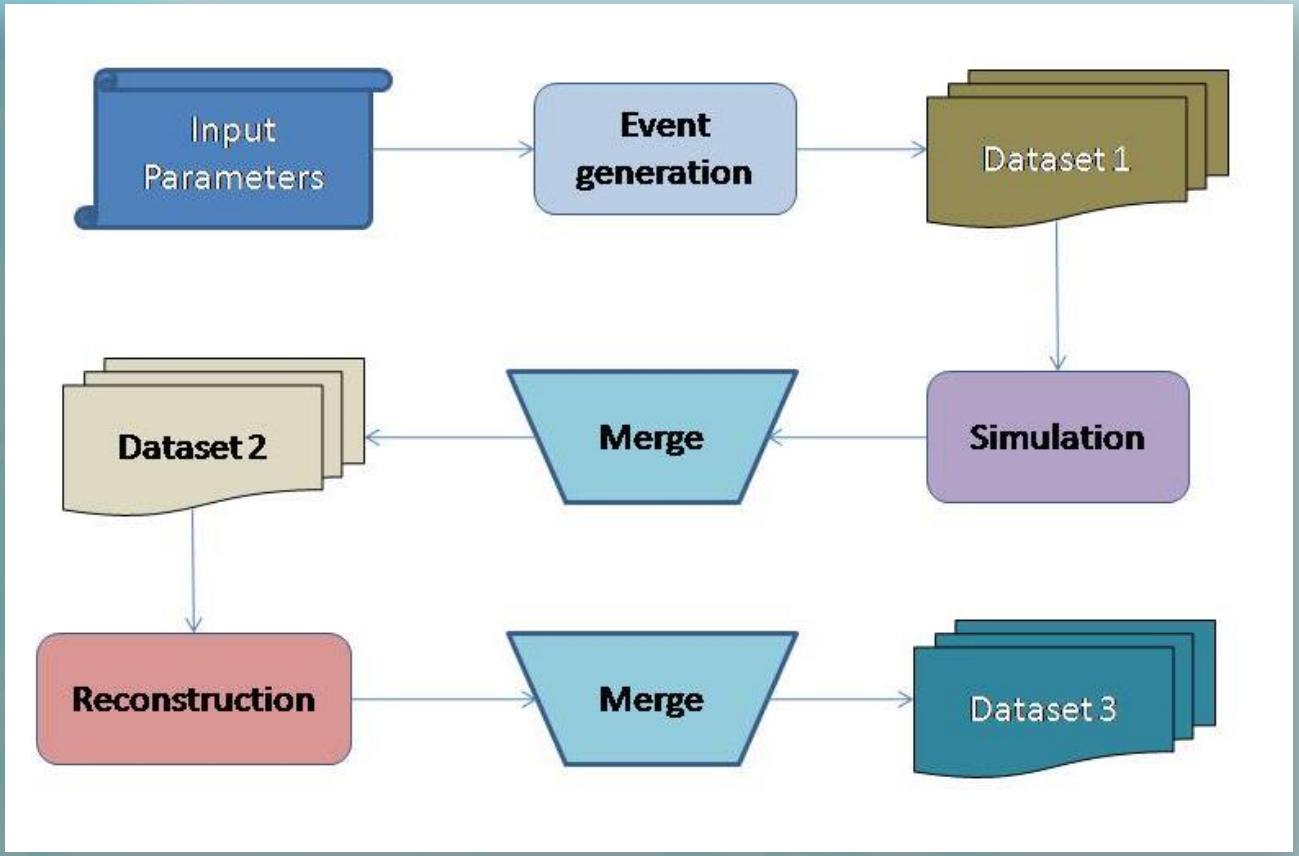
Why Meta-Task?

- The initial PanDA design parameters contained an assumption that a relatively small number of tasks will be maintained in the system, while each task would be translated into a large number of job definitions, to be submitted for execution.
- As we gained more operational experience with PanDA, it became obvious that physics groups and production managers have learned to use the system in *more sophisticated ways*, formulating the workload in a fine-grained fashion, i.e. multiple tasks of smaller size, often logically connected with each other. This resulted in the exponential growth of the rate of PanDA Task Request submissions:



Meta-Task as a way to manage complexity of the workflow

- As shown in the previous page, there is a real need to manage the scale and complexity of the PanDA workflow in order to accommodate the explosive growth of the task requests and complexity of their relationship. For example, the following highly simplified diagram illustrates a chain of PanDA tasks and the data that defines their relationship:



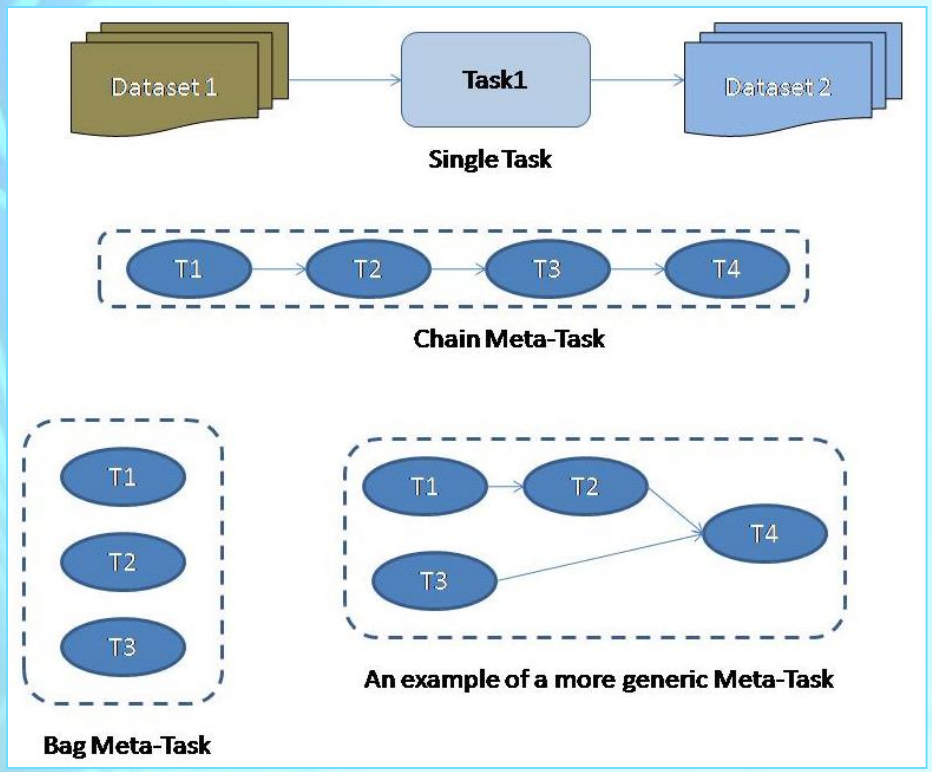
ProdSys2: Requirements from the DEFT perspective

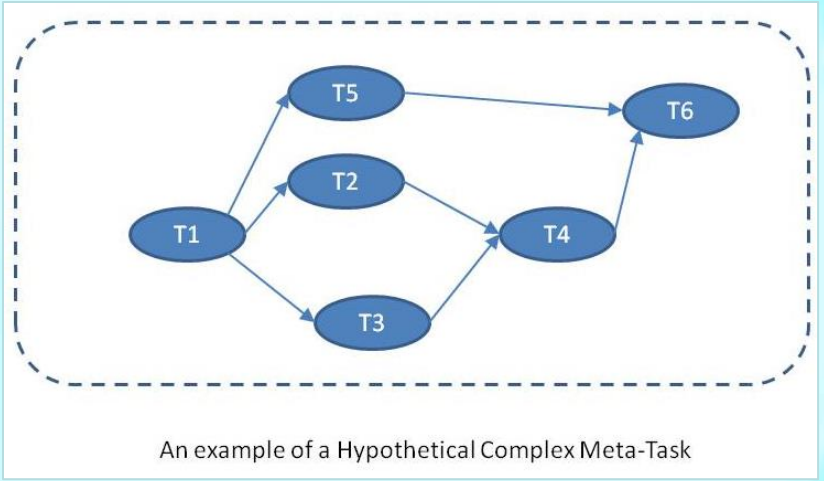
- **Automation**: we need the capability to process Meta-Tasks with minimal human intervention beyond task definition. Right now it is a labor intensive semi-manual process.
- **Automatic recovery** from simple failure modes
- But we also need the capability to have **operator intervention** and Meta-Task recovery: there must be adequate tools for the operators and managers to direct the Meta-Task processing
- **Dynamic** job definition (e.g. making it dynamic as opposed to static once the task is created) – functionality to be implemented in JEDI
- **Maintainability**: the code of the existing Production System was written "organically", in order to actively support emerging requests from the users, and it starts showing its age
- **Scalability**: in general, given the dramatic rise in the number of tasks defined and executed, we must ensure a lot of headroom going forward.
- **Ease of use**: there is currently a great amount of detail that the end user (Physics Coordination) must define in order to achieve a valid task request. We must automate and facilitate the task creation process, whereby cumbersome logic is handled within the application, and the user interface is more transparent and friendly.

DEFT: Meta-Task as a Graph Model

Graph model is commonly used to describe workflow:

- It allows implementation of the “chain”, “bag” and “bag of chains” topologies
- Liberates the designers of the workflow from limitations of the current “spreadsheet” model
- In the following, we will generally use the concept of tasks as a group of jobs using one or more datasets as input, and one or more datasets is output:

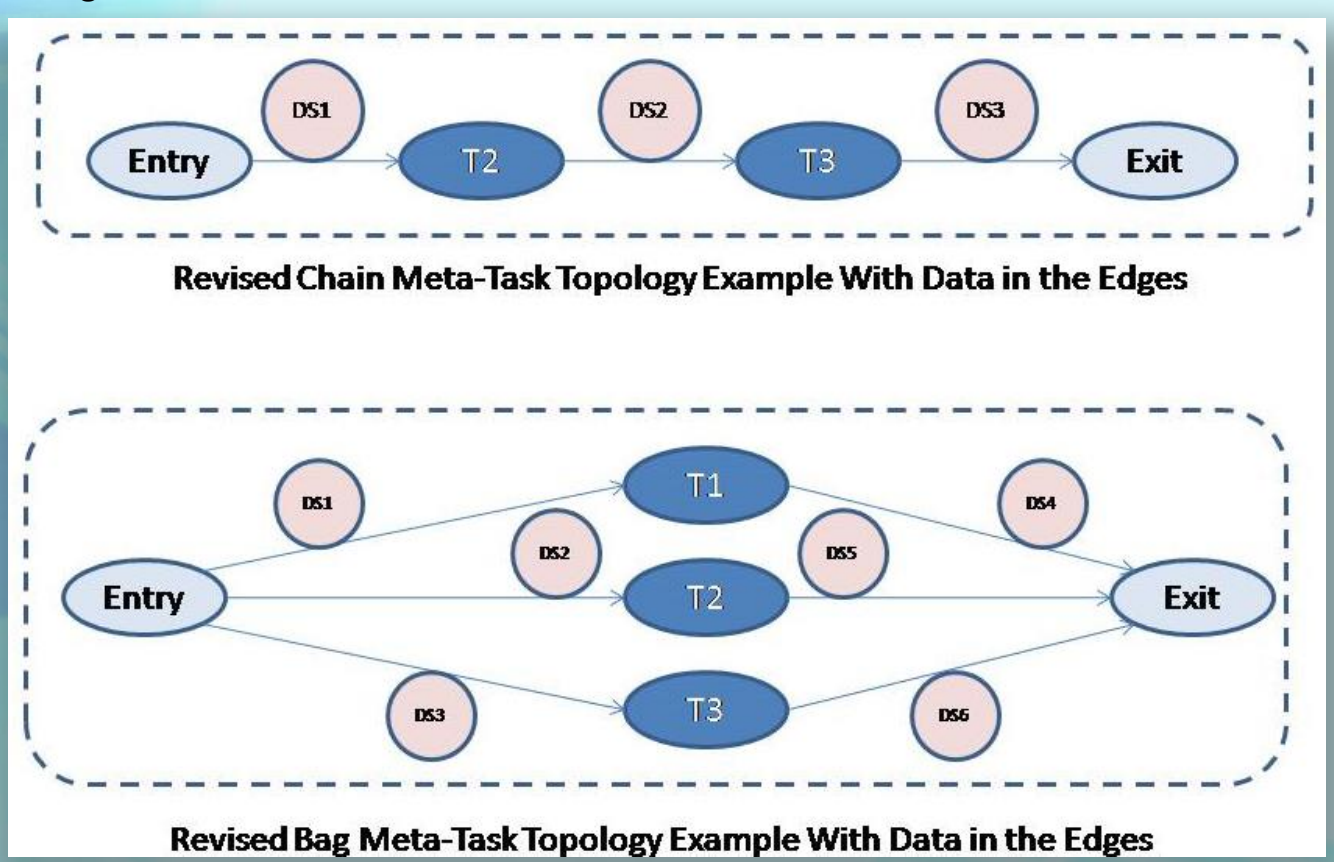




The Graph Model

Crucial features of the DEFT Meta-Task Model

- In accordance with operational practice in ATLAS, the dependencies between two adjacent tasks, which we model as nodes of a graph, are best conceptualized as datasets, which then become the edges of the graph
- In order to handle “bag” and other complex topologies, we introduce “pseudo-tasks” representing the entry and exit points. It is actually an established technique in handling workflows.



The DEFT Equation

We model tasks as workflow elements acting on an array of input datasets, while producing a number of output datasets. The tasks have *states*, and undergo state transitions based on their inputs and applicable rules. Each task in a Meta-Task can be represented as a vector of variables describing its internal state and state of each input and output. The role of DEFT is to apply rules to transform this vector. Importantly, some of the parameters will be modified by JEDI.

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_N \end{pmatrix} = D \otimes \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_N \end{pmatrix}$$

Meta-Task: the Language

How do we represent and document the objects created according to the Graph Model, in human-readable format?

- The need to represent Meta-Tasks and their components in a way amenable to reading, editing and archival by humans was realized early on in the project
- Do we need to build this format from scratch? Probably not,

It's the model!

- Since we consider the Graph Model as the optimal way to represent the workflow in its various states, it is a reasonable approach to try and identify a natural way to represent the graph
- This leads to realization that there are already standard languages and schemas that do exactly that. Some are XML-based, some are not.

Language	Format/Language	Comment
GraphML	XML	Supported for I/O by NetworkX, with some caveats
DGML	XML	Miscrosoft Product, used in Visual Studio
GXL	XML	Top-heavy. Somewhat complex syntax
GML	Custom	Supported for I/O by NetworkX. No clear advantage to GraphML, and would require a custom parser
DOT	Custom	Too graphics-oriented, unwieldy syntax

DEFT: example of a workflow template in GraphML

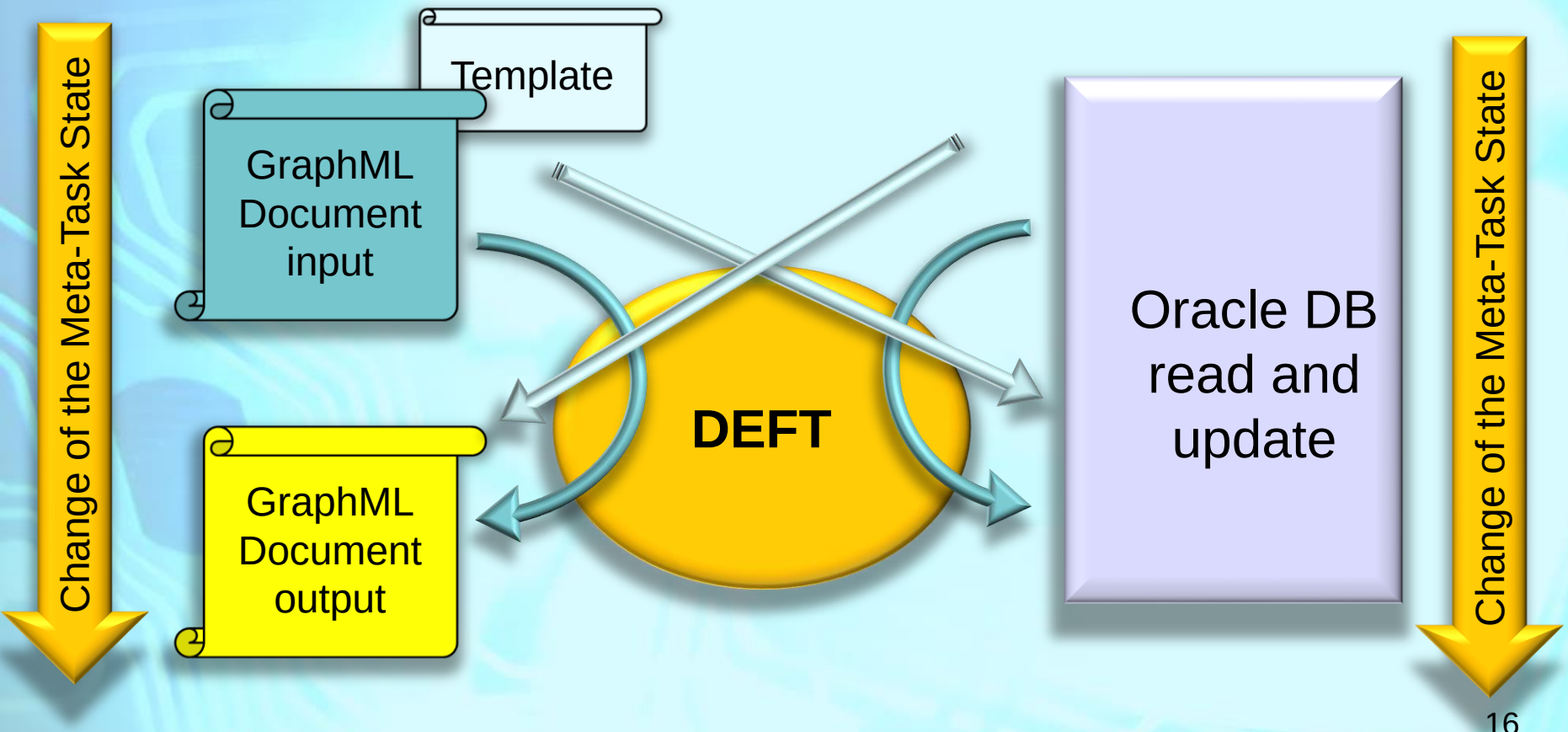
```
<key attr.name="comment"      attr.type="string"      for="edge"      id="comment"     />
<key attr.name="meta"         attr.type="string"      for="edge"      id="meta"        />

<key attr.name="state"        attr.type="string"      for="node"      id="state"       />
<key attr.name="tag"          attr.type="string"      for="node"      id="tag"         />
<key attr.name="comment"      attr.type="string"      for="node"      id="comment"     />
<key attr.name="meta"         attr.type="string"      for="node"      id="meta"        />


<key attr.name="last_update"  attr.type="string"      for="node"      id="last_update"/>

<graph id="simulation template 1" edgedefault="directed">
  <node id="entry">
      <data key="state">disarmed</data>
      <data key="tag">entry</data>
      <data key="comment">Entry point/null task</data>
      <data key="meta">meta-task id</data>
  </node>
  <node id="evgen">
      <data key="state">armed</data>
      <data key="tag">e1119</data>
      <data key="comment">Event Generation</data>
      <data key="meta">meta-task id</data>
  </node>

  <node id="Simulation 1">
      <data key="state">disarmed</data>
      <data key="tag">s3201</data>
      <data key="comment">Simulation 1</data>
      <data key="meta">meta-task id</data>
  </node>

  <node id="Simulation 2">
      <data key="state">disarmed</data>
      <data key="tag">s2_1111</data>
      <data key="comment">Simulation 2</data>
      <data key="meta">meta-task id</data>
  </node>
```

DEFT: Status and Summary of Progress in the past 6 months

- Analyzed initial project requirements (with Wolfgang Ehrenfeld et al)
- Developed an object model for the DEFT component of ProdSys2
- Created (and updated as development progressed) the database schemas for object and graph persistence in RDBMS
- Investigated multiple existing solutions for workflow management and identified software components to be used in the project
- Applied an appropriate (*industry-standard*) format for describing, modeling and version control of the PanDA workflow
- Developed a prototype DEFT application –
  - CLI driver with an extensive set of command-line options to enable realistic testing and bookkeeping of the task database
  - A suite of reusable Python classes and modules
  - Efficient logging mechanism based on a standard Python package
- Code actively maintained in SVN
- Maintained documentation on every aspect of the development effort, from design to JEDI interface, in TWiki and a specially created ProdSys2 blog.
- Documented important use cases
- Started addressing the issues of DEFT performance and scalability by running simulated sweeps over a large group of test Meta-Tasks.

The DEFT Prototype

- DEFT exists as a **functioning**, proof-of-integration prototype (CLI utility)
- Integration of NetworkX, GraphML, PyUtilib and DB Oracle schemas
- Capability **to import and export workflows in GraphML format, as well as to persist data in RDBMS**, and access and modify data transparently across these containers



The DEFT Prototype

- Capability to support workflows described by DAG of **any complexity**, not just “chains” and “bags”
- Straightforward **template capability, cloning and copying of tasks**
- Implementation of rules for **state transition** of tasks

```
[acas0001] ~/pw_test $ deft_v1.1.py -h
usage: deft_v1.1.py [-h] [-d DEBUG] [-t TEMPLATE] [-i IN_META] [-o OUT_META]
                    [-p PURGE] [-s] [-r RETRIEVE] [-f FETCH] [-T UPDATE_TASK]
                    [-D UPDATE_DATASET] [-x]

optional arguments:
  -h, --help            show this help message and exit
  -d DEBUG, --debug DEBUG
                        DEBUG level
  -t TEMPLATE, --template TEMPLATE
                        Meta-Task Template input (GraphXML File)
  -i IN_META, --in_meta IN_META
                        Meta-Task to be processed (GraphXML File)
  -o OUT_META, --out_meta OUT_META
                        Processed Meta-Task (GraphXML File)
  -p PURGE, --purge PURGE
                        Purge Meta-task from RDBMS, '0' means all
  -s, --store           Store Meta-Task in RDBMS
  -r RETRIEVE, --retrieve RETRIEVE
                        Retrieve Meta-Task from RDBMS
  -f FETCH, --fetch FETCH
                        Single Task to be fetched
  -T UPDATE_TASK, --update_task UPDATE_TASK
                        Update attributes of task(s)
  -D UPDATE_DATASET, --update_dataset UPDATE_DATASET
                        Update attributes of dataset(s)
  -x, --execute         Execute Meta-Task
```

Near-term Plans (0-3 months)

Initial stages of DEFT/JEDI Integration

- In a parallel development, not covered in this presentation (credits: T.Maeno, D.Golubkov et al) an interface was developed between the “old” task definition interface and the new JEDI modules. This part of the project has been given the codename **“JEDI-alpha”.**
- With **JEDI-alpha** operational and providing valuable testing platform for JEDI, the next step is to inject tasks from DEFT into JEDI and thus create an instance of ProdSys2.
- Test of functionality while emulating various operational and failure conditions
- Basic “vanilla” use cases as the starting point.

Data Management Integration

- Handling transient datasets needs to be addressed
- Some of the basic functionality such as dataset naming module is yet to be implemented (will follow shortly)

Initial design of the graphic user interface

- The user interface is an important feature of the overall system and must be properly implemented
- Need to evaluate existing packages for graph and other data manipulation

Medium-term Plans (3-6 months)

DEFT/JEDI Integration: covering more use cases

- Will follow the list of use cases beyond the basic one, and implement necessary logic for handling these (incremental task execution, lost files etc)
- Further iteration of the database schemas as necessary

User Interface prototype

- Will have the initial working prototype of the DEFT UI.

Finalizing DEFT/JEDI integration

- To test the complete DEFT/JEDI functionality throughout the lifecycle of a Meta-Task

Developing use cases for ProdSys2 application to analysis workflow

- This hasn’t been done yet, remains an important objective.

Long-term Plans (6-12 months)

End of 2013

- ProdSys32 in pre-production (a limited number of production Meta-Tasks executed under close supervision and QA)
- Functioning UI

Early 2014 – rollout!

- Introduction of ProdSys2 into the ATLAS production cycle

Mid-2014 – UI and Monitoring Integration

- The objective is to provide capable monitoring and management tools to the groups production managers and operations staff. This implies a level of integration of the DEFT UI and various monitoring components in ATLAS.
- While not the core part of the project, this component will play a major role in its success.

Late 2014 – analysis workflow

- Using ProdSys2 to handle the analysis workflow.

Backup slides

Meta-Task: GraphML, NetworkX and RDBMS

Choice of Schema

- There is an obvious advantage in choosing the schema that's is standardized, enjoys support and has parsers already written to handle its specifics.
- **GraphML** appears to be very simple, human-readable and enjoys parser support in many existing visualization and analysis software products
- It allows us to standardize on the workflow description, visualization, editing, documentation and versioning with essentially zero effort.
- Capable graphic editors, such as GePhi, already exist and can be used immediately to create, visualize and edit Meta-Task templates (see backup slides).

NetworkX

- *"NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks."*
- While its functionality is quite rich, and allows all sorts of graph analysis and exploration, the minimal subset of methods is quite easy to learn and use immediately
- Reads GraphML, JSON, and other documents and creates an in-memory model of the graph automatically. Likewise, serializes graphs into a variety of formats, like GraphML, JSON etc
- Visualization can be implemented by a few supported Python packages which need to be installed separately, such as matplotlib etc.

Persistence in RDBMS

- Persisting graphs in RDBMS had been addressed before; we revisited existing approached and chose the "Adjacency Table" approach as the most scalable and easy to implement. Two tables are created, one for nodes and another for edges. See Twiki for details: https://twiki.cern.ch/twiki/bin/viewauth/Atlas/TaskModel#Graphs_in_RDBMS

DEFT/JEDI Communication

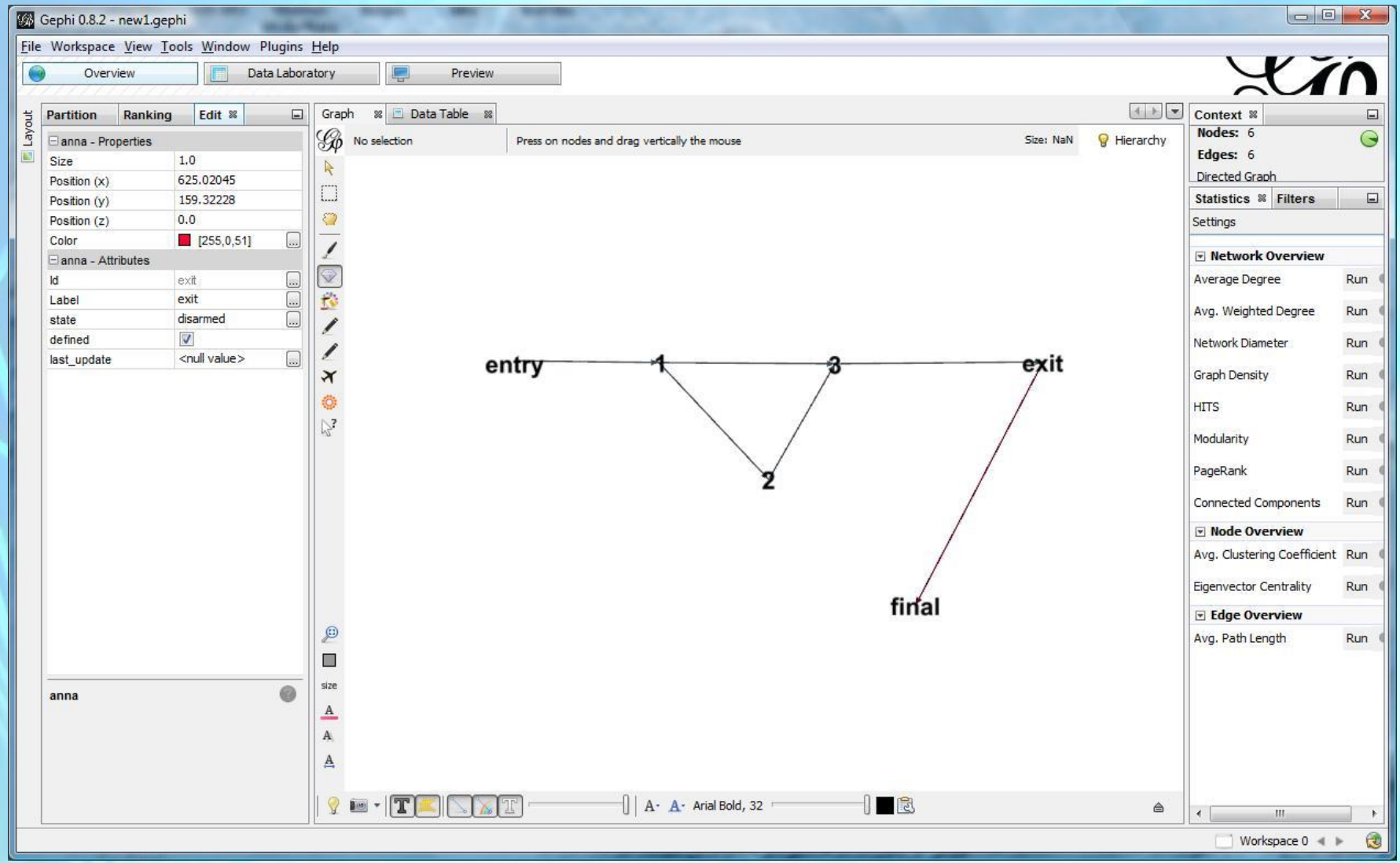
- A few features of the DEfT/JEDI tandem design: in managing the workflow in DEFT, it's unnecessary (and in fact undesirable) to implement subprocesses and/or multi-threaded execution such as one in products like PaPy etc. That is because DEFT is NOT a processing engine, or even a resource provisioning engine, as is the case in some Python-based workflow engines. **It's a state machine** that does not necessarily need to work in real time (or even near time). Near-time state transitions and resource provisioning is done in PanDA
- There are a few ways to establish communication between Deft and Jedi, which are not mutually exclusive. For example, there may be **callbacks**, and database token updates, which may in fact co-exist. If a message is missed for some reason, the information can be picked up during a periodic sweep of the database. In summary, DEFT and JEDI will work **asynchronously**.
- Essentially, both components perform a database sweep
- Since there are no in-memory processes keeping track of all Panda tasks and jobs at any given time, this provides us with better scalability going forward, compared to some other workflow management solutions

Backup slides: notes on visualization

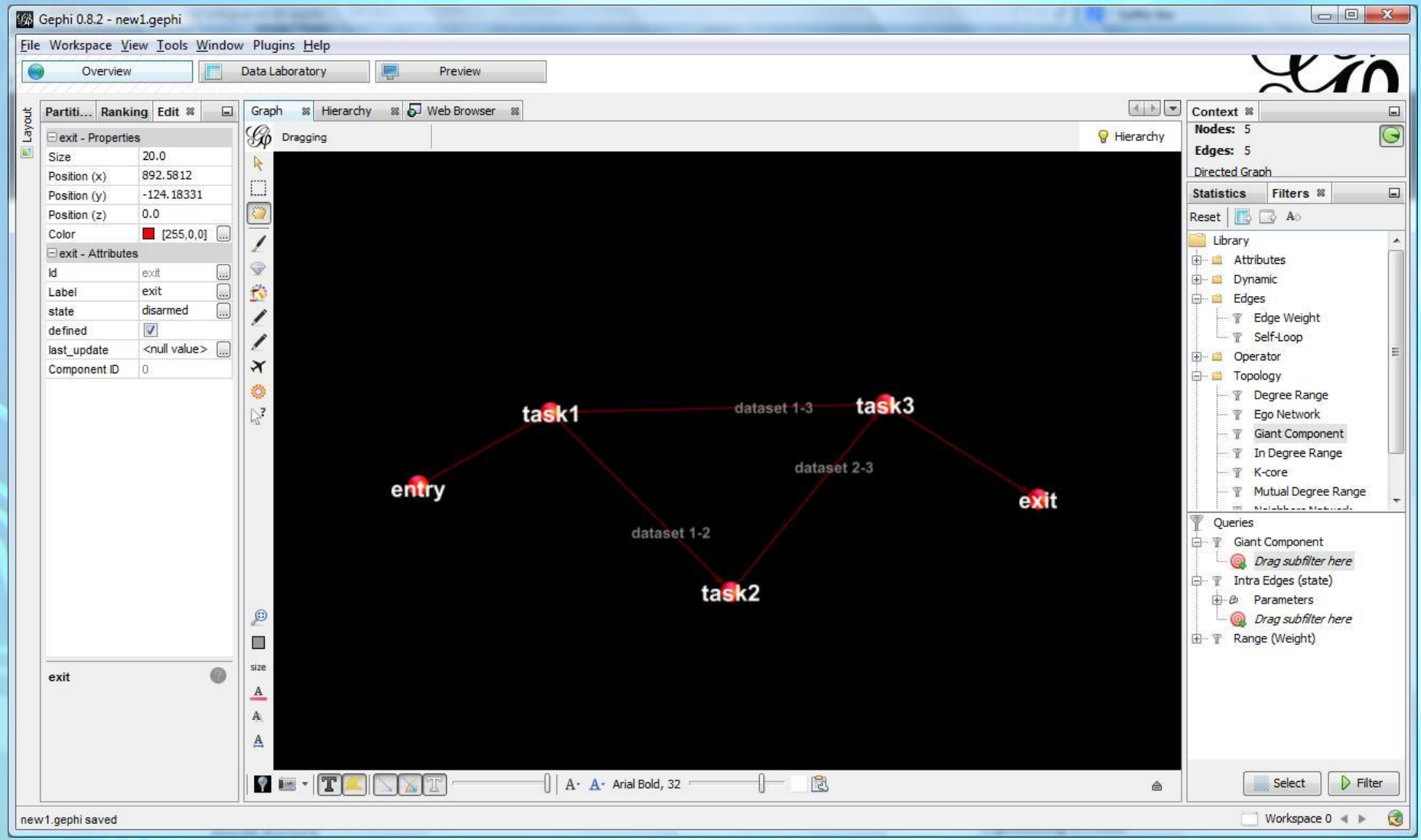
Task visualization, editing and monitoring

- Basic visualization tools are already available, such as GePhi and matplotlib add-on to NetworkX (cumbersome installation though). Editing is available in GePhi complete with a GUI interface, and of course GraphML files can also be edited using any text editor.
- For more polished look and more dynamic and better user experience, we can develop a browser-based frontend utilizing jsPlumb, WireIt, Raphael etc – but we need to budget manpower for that, since the considerable power of these graphics systems comes with significant complexity of logic and API

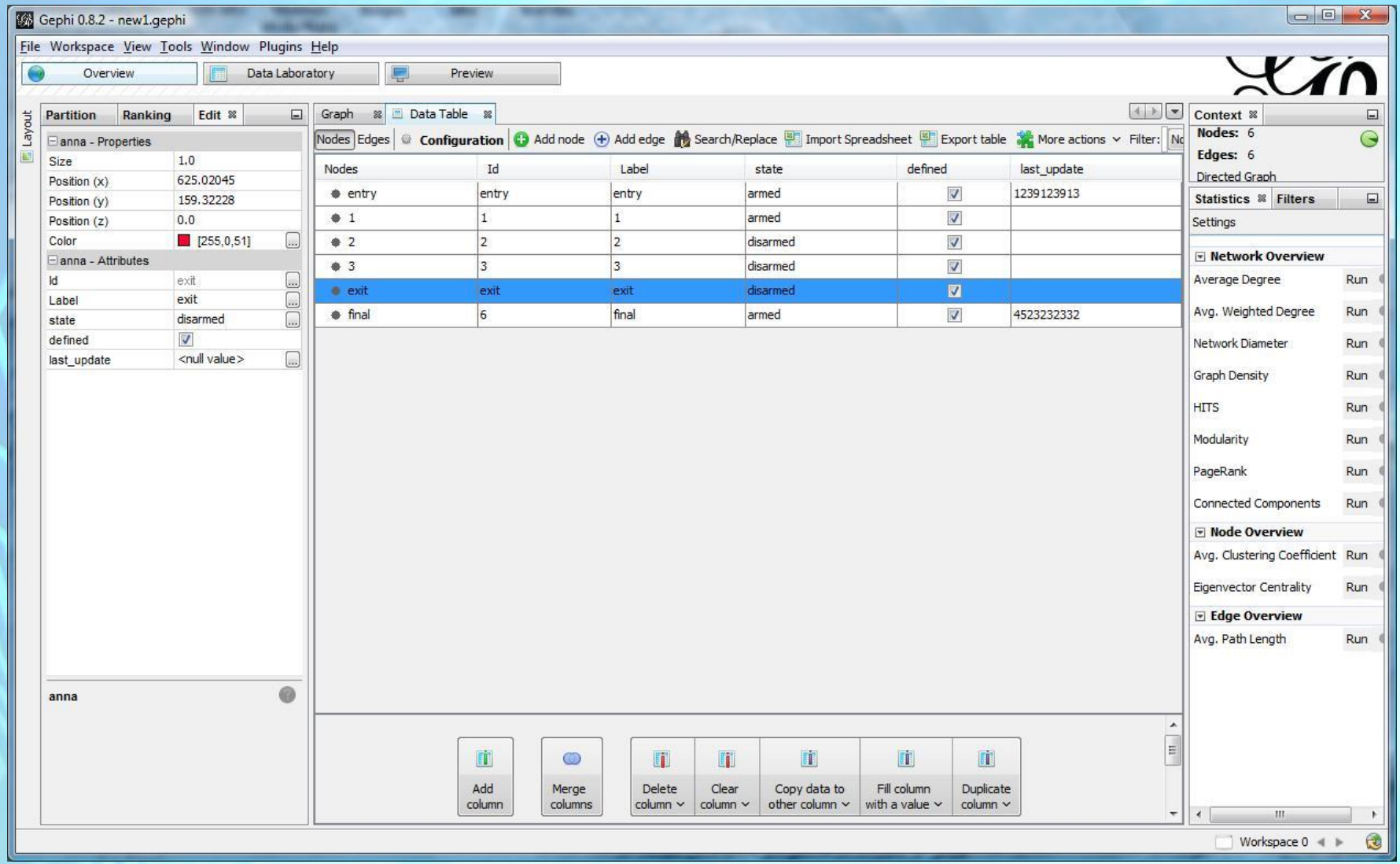
Backup slides: examples of workflow visualization and editing in GePhi



Backup slides: examples of workflow visualization and editing in GePhi



Backup slides: examples of workflow visualization and editing in GePhi



Backup slides: examples of Javascript tools to aid in building Meta-Task GUI in DEFT