High-Volume Pipeline Processing with the LSST Data Management System

Raymond Plante
NCSA
Lead Scientist for Middleware & Infrastructure

http://dev.lsstcorp.org

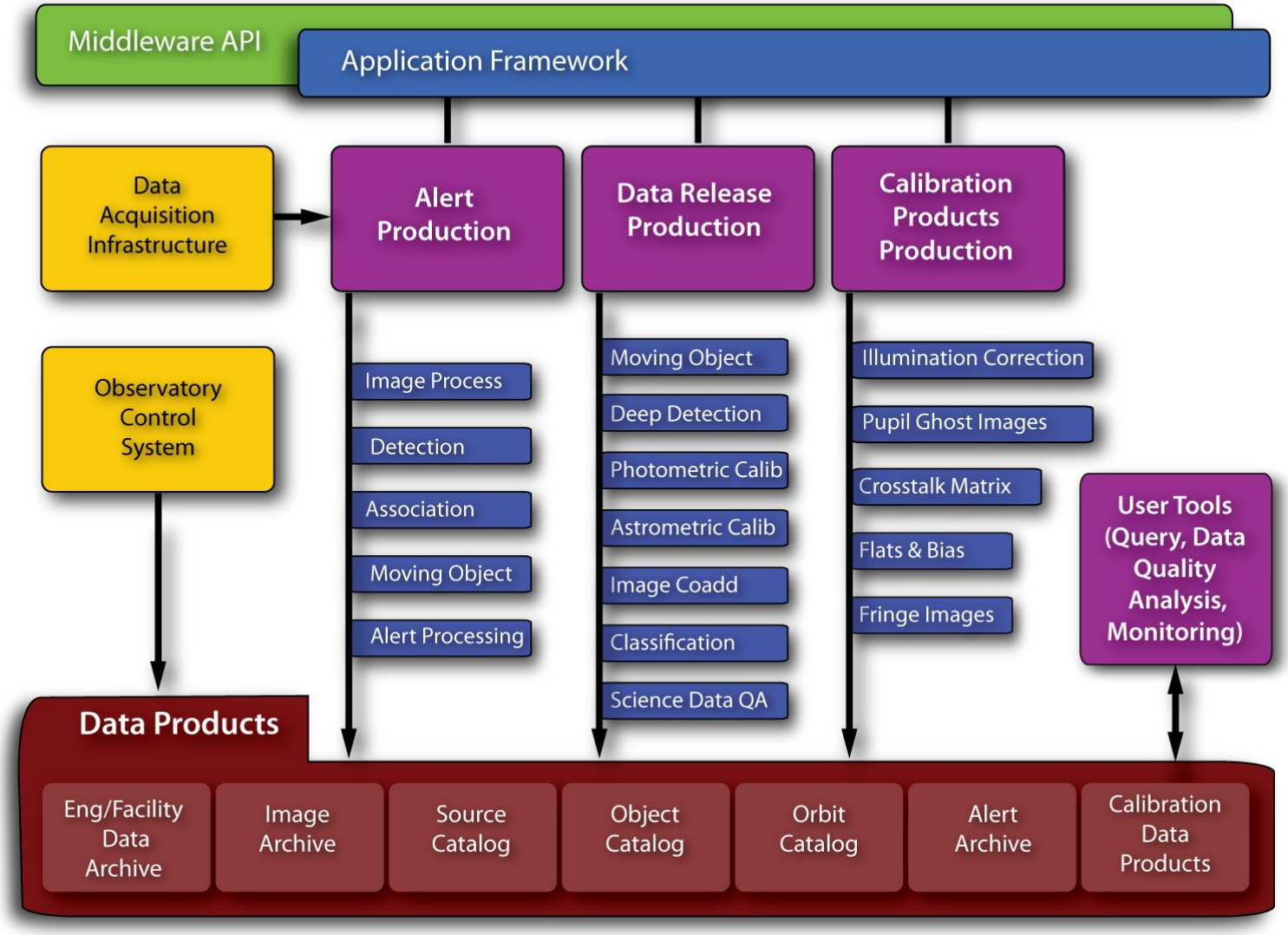
"Layers" of the LSST Data Management System (DMS)

- **Applications:** libraries representing astronomically-relevant classes and algorithms implemented using those classes
- **Middleware:** libraries and tools that handle work-flow, I/O, information management (e.g.inputs provenance,) that are used to stitch algorithms together into processing pipelines.
- **Infrastructure:** Hardware architecture that runs the pipelines
- **Software Environment:** tools for building, installing, and using software

Software Environment

- **DMS software stack: integrated set of 3rd-party and custom packages**
  - **22 3rd-party packages, 25 custom package (distributed) + ~10 (in dev.)**
- **Custom packages** (repos: http://dev.lsstcorp.org/trac/browser/DMS)
  - **C++ for core reusable classes**
  - **SWIG used to create Python bindings**
  - **Python layer for connecting classes into applications and pipelines**
  - Some Java and Jython
- **Stack is highly modular**
  - **Allow one to minimize what gets installed**
  - **Requires that we track dependencies carefully**
  - **EUPS: a tool for managing the environments of a collection of interdependent packages**
    - `setup utils 3.0` **– updates the values of environment variables (PATH, LD_LIBRARY_PATH, PYTHONPATH, etc.) for package utils, version 3.0, *and all its dependencies***
    - **Multiple versions can be installed at the same time**
    - **Provides rpm-like distribution of packages (built from source)**

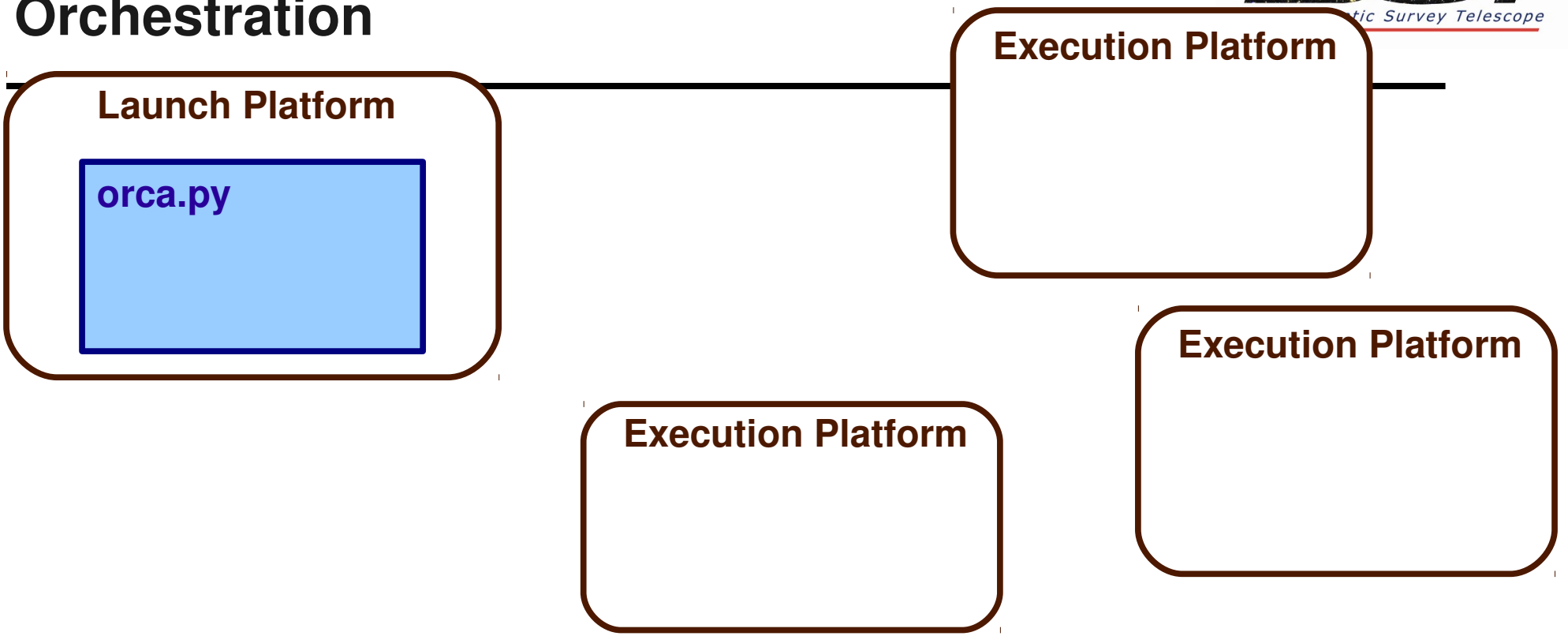
The Science View of LSST Processing



Middleware View of LSST processing

- Two categories of processing => two strategies
  - Alert Production: Real-time Processing
    - Executed nightly
    - Minimize I/O by keeping data in memory
      - Stress data-parallelism; requires consistent routing of data
      - Isolate and minimize parallel process cross-talk
    - Parallelism implemented using MPI
  - Data Release Production: High-volume Processing
    - Executed yearly but continuously
    - We can trade performance for robustness
    - Processing is more complex
      - with changing axes of parallelism
    - Parallelism implemented using Condor
- Categories include some common needs
  - Provenance tracking
  - Logging under high levels of parallelism
  - Encapsulated data access via logical identifiers

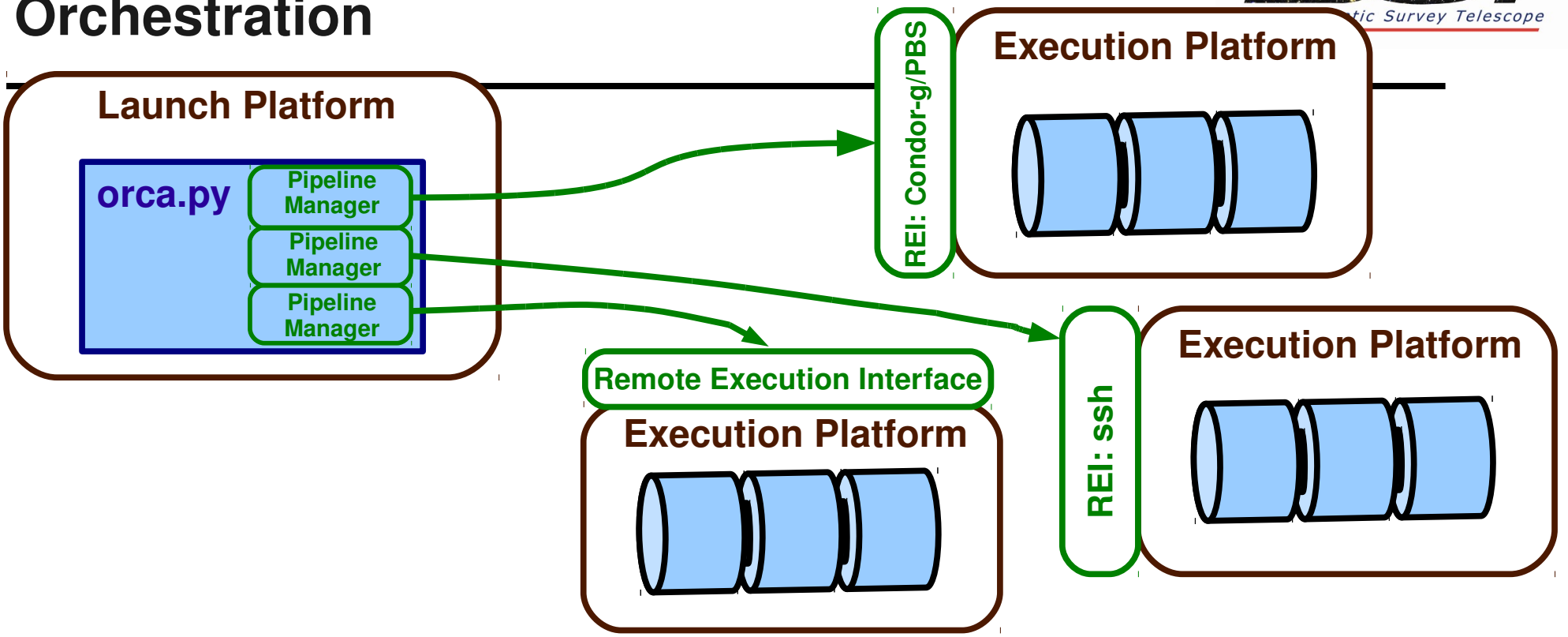
Orchestration



Some LSST vocabulary

- **Production =** a set of pipelines that collaborate to produce data products.
- **Production run =** an execution of the pipelines in a production
- **Orchestration**
  - **The launching and monitoring of the pipelines for a production run**
  - **Applying the planning / strategies for adapting to a particular hardware platform and data being processed.**

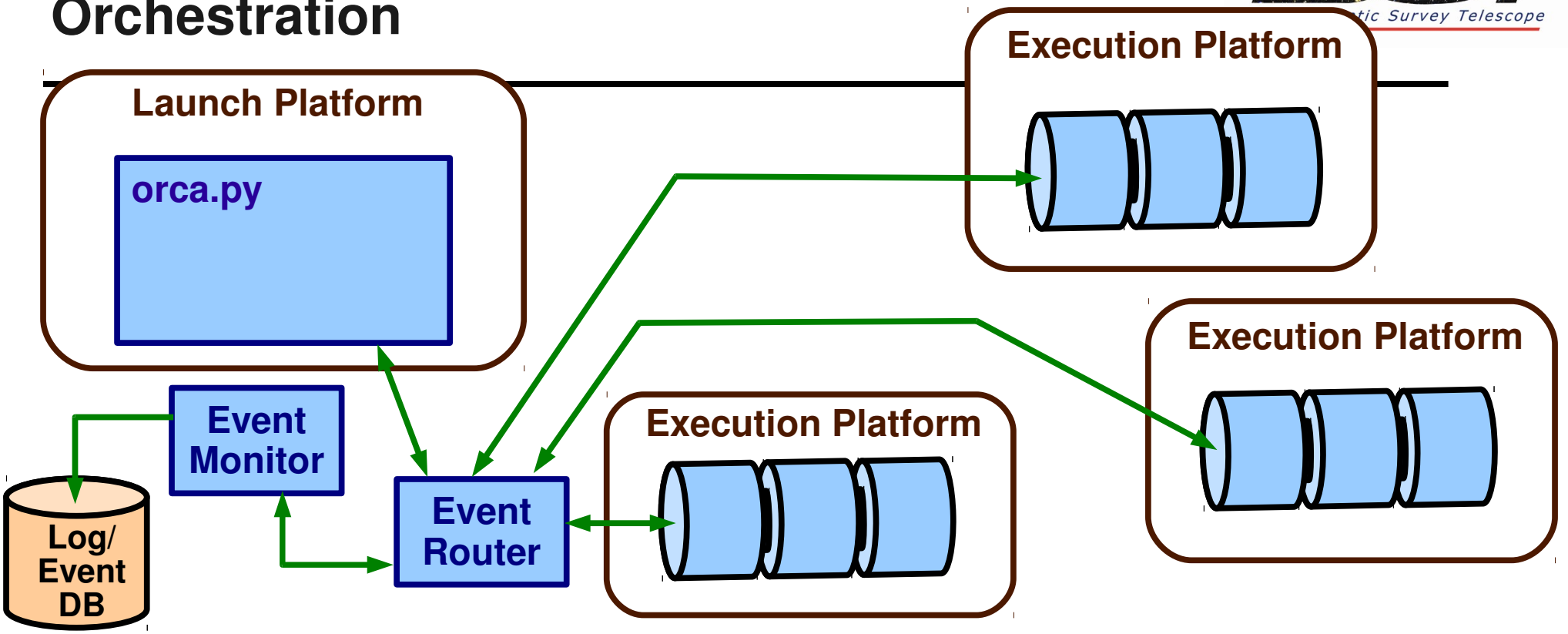
Orchestration



Orchestration layer launches pipelines on remote execution platforms

- adapts to different types of platforms and the way they run applications
- LSST designed to run on own dedicated platforms or community platforms (e.g. NCSA public resources)
- Launch mechanisms currently supported:
  - ssh
  - Condor-g: generic interface to local batch system (e.g. PBS)
  - Condor/DAGman

Orchestration



Pipelines communicate with each other and with the Orchestration layer via Events

- Event system based on the Java Messaging Framework (JMF)
- Pipeline log messages sent out as events for remote recording
- An Event Monitor analyses progress to detect possible problems
  - Node failure, Runaway process, ...
  - Can signal orchestration layer to relaunch failed processing
- Inter-pipeline communication via Events
  - One pipeline may "wait" until an expected event with needed information arrives
  - Event payloads are light: one pipeline may tell another where to look for data.

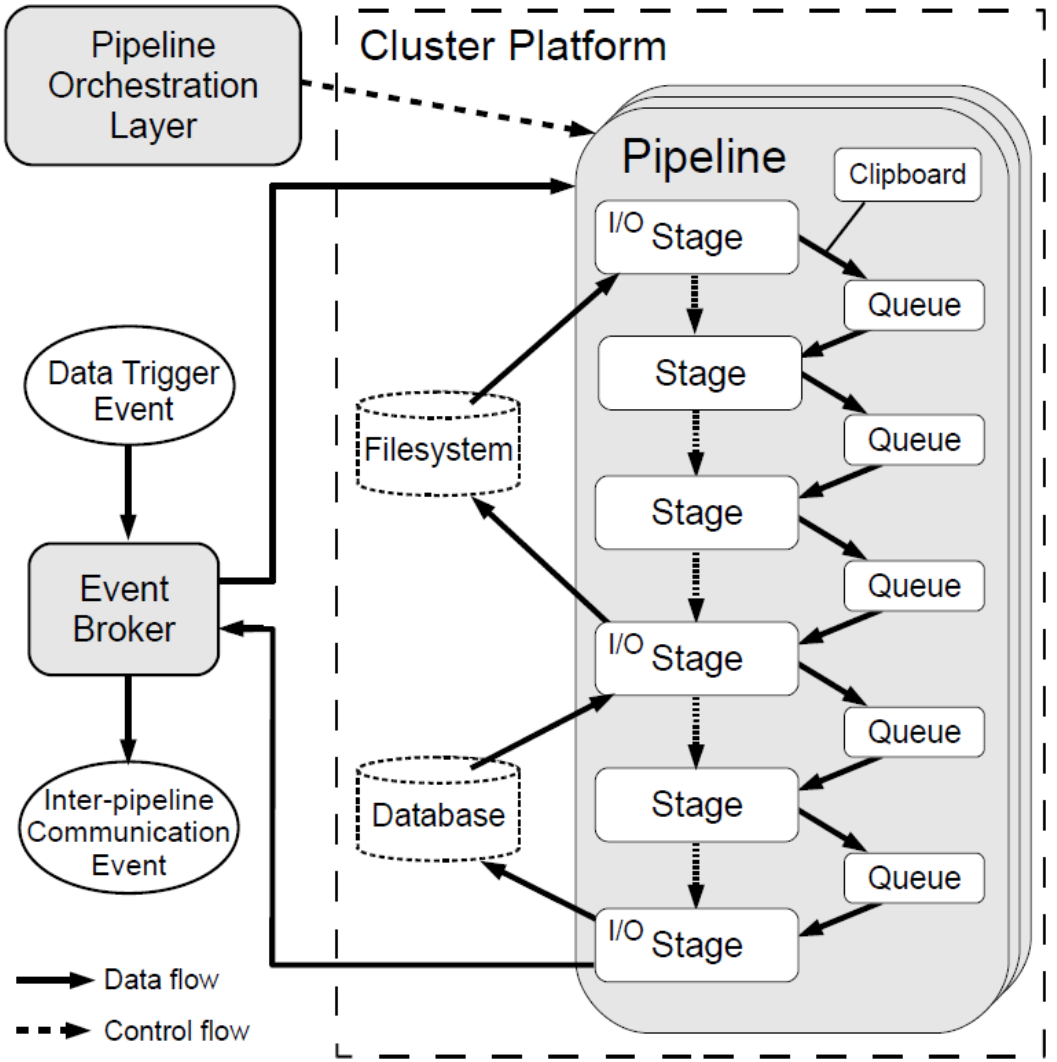
Alert Production (Real-time)

Three MPI-based pipelines

- **Image Processing and Source Detection (IPSD)**
  - Receives an event indicating which exposure to process
  - Removes instrumental signature, subtracts from template, detects source changes
  - Each MPI worker ("slice") operates on a different CCD segment
  - Tells Association Pipeline (via an event) where to find (in database) sources detected in difference image
  - Some "inter-slice" communication for cross-talk correction (not implemented) and WCS solution
- **Moving Objects Prediction (MOPS)**
  - Receives an event indicating a field of view on sky and a time
  - predicts positions of known moving objects
  - Each slice operates on a subset of objects
  - Tells Association Pipeline (via an event) where to find (in database) list of known moving objects in FOV
- **Association Pipeline**
  - Receives list of moving objects in FOV, list of difference sources via events
  - Matches difference sources with known sources

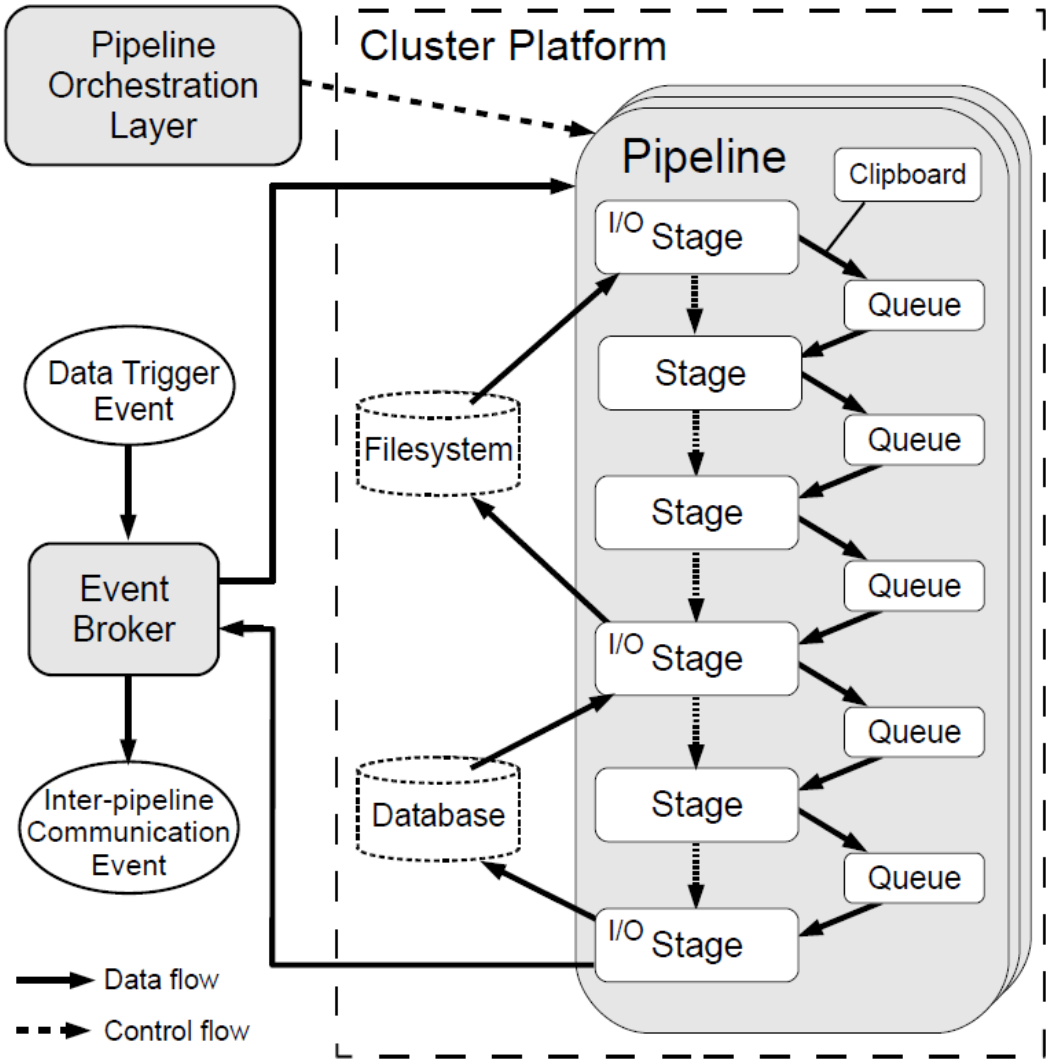
The Pipeline Harness

- Pipeline = a sequence of Stages
  - "Harness" = container for stitching together stages
- Pipeline has N+1 threads
  - Pipeline thread is the master controller
  - Slice threads process a data-parallel unit of data
    - e.g. CCD segment
  - All threads have same basic structure
- Sequence of Stages
  - Data queues sit b/w each stage
  - Data is passed from one stage to another through queues via a "Clipboard"
    - Clipboard = hierarchical dictionary
    - Stages look for input data on clipboard and place new data items on it.



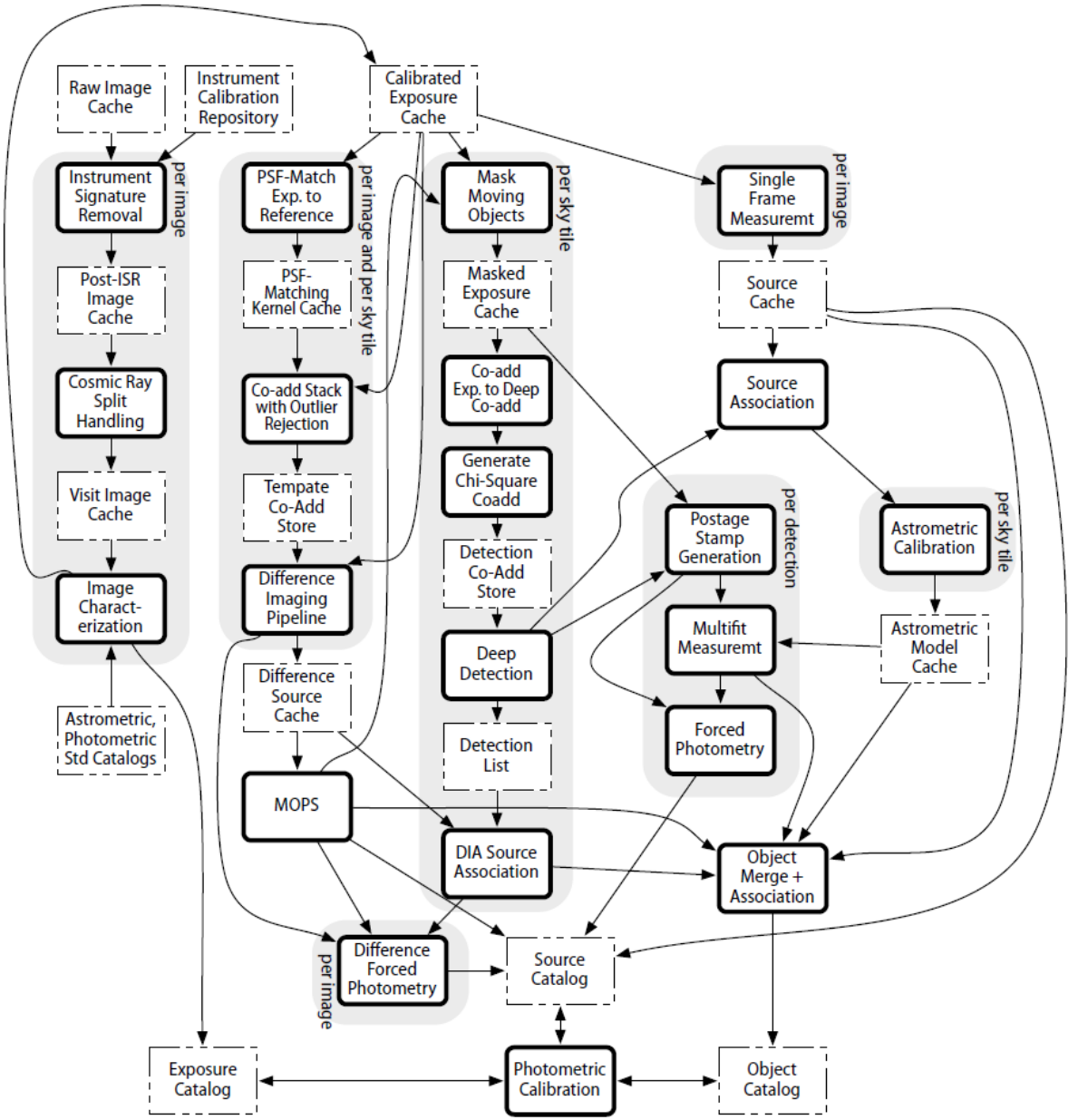
The Pipeline Harness

- I/O is done via dedicated I/O Stages
  - Algorithm stages are isolated from I/O
  - Output Stages can write products to disk at any time
- Processing is Event driven
  - A data trigger event signals the first stage which exposure to process next
  - Harness puts event data on clipboard
  - Each stage is run in sequence
  - After last stage is executed, pipeline returns to first stage, waiting for next event.



Data Release Pipeline (High-Volume)

- Sequencing is more complicated
  - Many interdependencies
  - Shifting "axes of parallelism"
    - By CCD segment
    - By sky-tile
    - By object
- Real-time is not required
  - Trade performance for robustness
  - Do more caching of data to disk
  - Leverage existing technology: Condor/DAGMan

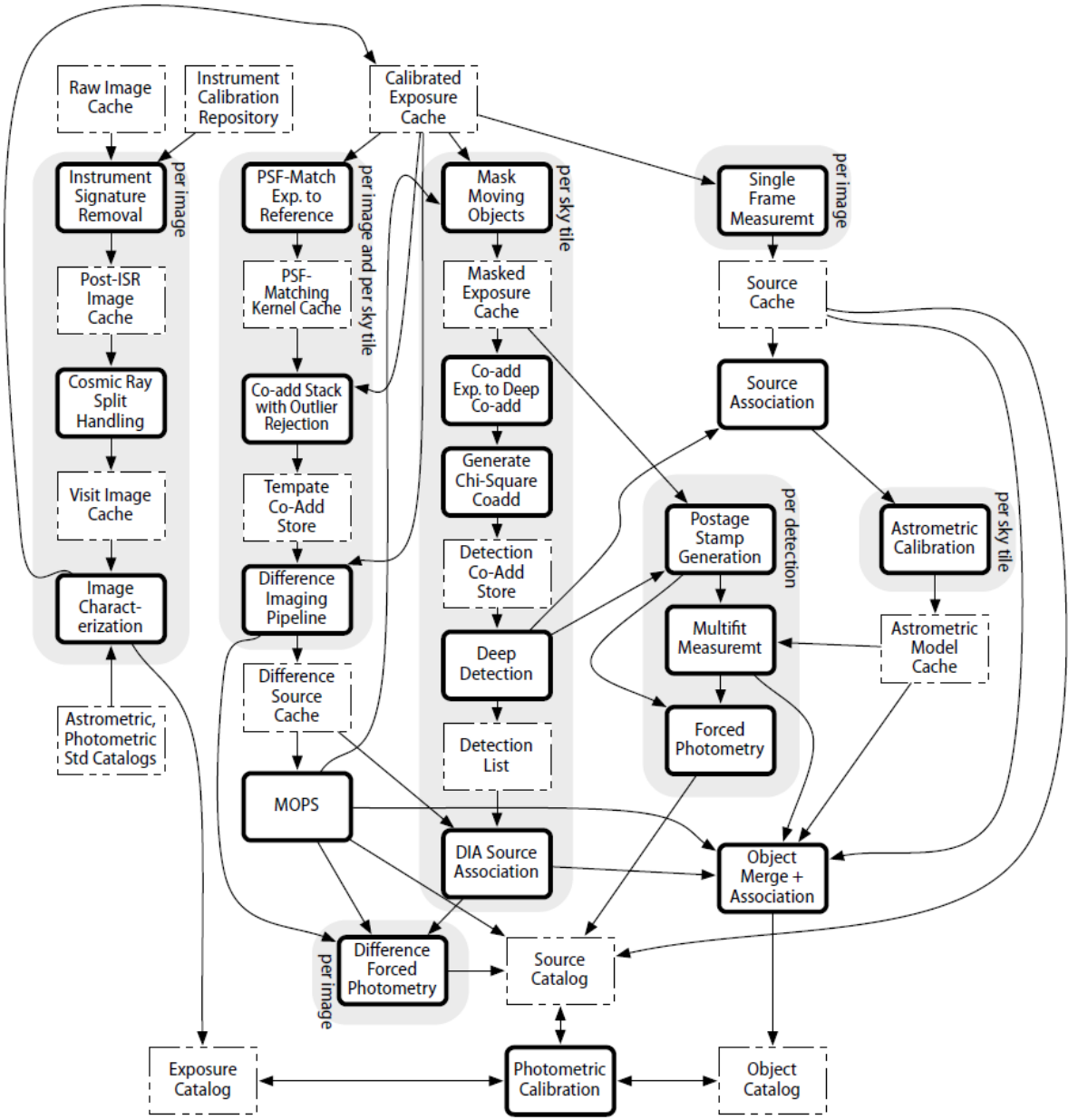


Parallelism via Condor

- Condor: parallel job execution system optimized for highly embarrassingly (data-) parallel applications
- DAGMan: a Make-like description of an application that can be processed by Condor.
  - Model app as jobs with interdependencies in form of a “directed acyclical graph” (DAG)
    - No inter-job communication
    - A job can be an MPI application to run on a cluster
  - Condor marches through DAG executing jobs while honoring dependencies
    - Typically, # of jobs >> # of available cores
    - Condor will maximize core utilization
- Can take advantage of Condor robustness features
  - Automatic rescheduling of failed jobs
  - Can detect loss of contact with jobs

Data Release Production via Condor/DAGMan

- **Short and Narrow pipelines**
  - **Single slice, operating or single data-parallel unit of data**
  - **Only one unit will pass through pipeline per instantiation**
- **External process for staging input data**
  - **Volume to process >> capacity of disk**
  - **Will listen for events indicating progress of pipelines**
  - **Will stage data from mass storage before needed**



Creating a Stage

- Harness provides a Stage API for turning a general algorithm into a pluggable pipeline component
  - Developer concentrates on applying algorithm to a data-parallel unit of data
    - No I/O included
    - No managing of parallel processing
      - Allows stage to be put into either Alert Production or Data Release
  - Stage has up to three steps:
    - Serial pre-processing (via Master Pipeline)
    - Parallel processing (via a Slice)
    - Serial post-processing (via Master Pipeline)
    - Note: Serial and Parallel processing on different nodes!
      - API allows passing data between parallel Slices and between master and slices (scatter/gather)
- Stages are configured via text configuration files call “policy files”
  - Hierarchical property dictionary
  - Simple, hand-editable file format
  - A special “Dictionary” format can be used to validate policy data
    - Much like XML Schema for XML documents

Building Community Software

- LSST Users will create advanced science data products
  - Will want to create new processing and analysis algorithms
  - Will want to tweak the application of standard processing
- LSST Users will want to leverage LSST DMS
  - Use framework to build advanced pipelines
  - LSST Data Access Centers will provide access to computing resources to create advance products
- An open, distributable software framework is, thus, an important requirement
  - Leverage other computing platforms
  - Allow application to other data and observatories