Scientific Applications of Data Distribution Service

Svetlana Shasharina#, Nanbor Wang,
Rooparani Pundaleeka, James Matykiewicz and
Stephen Goldhaber

sveta@txcorp.com

Outline

- Introduction
  - Tech-X Corporation
  - Driving scientific applications
  - Common requirements
- QUIDS project
  - Workflow management
  - Security investigation
  - Python API implementation
- Summary

Tech-X Corporation

- Founded in 1994, located in Boulder Colorado
- 65 people (mostly computational physics, app math, applied computer science)
- Funded by DOE, NASA, DOD and sales
  - Plasma modeling (accelerators, lasers, fusion devices, semiconductors) and beam physics
  - Nanotechnology
  - Data analysis
  - Scientific data visualization
  - GPU programming for science
  - Data management and distribution
  - Distributed middleware (CORBA, DDS, Globus, GRID)

Large Synoptic Survey Telescope (LSST)

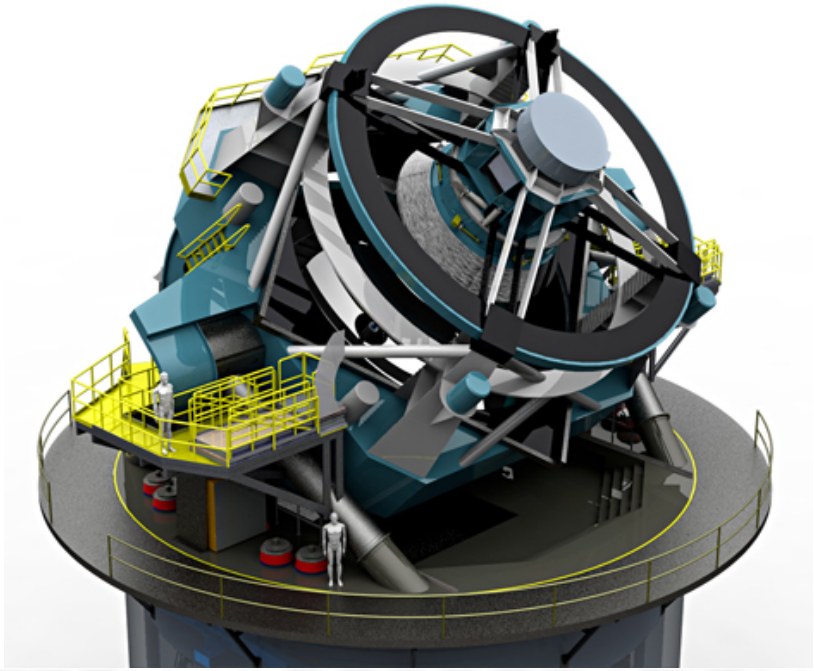
Largest digital camera to be built (Chile) to track moving objects, find remote objects, research on dark energy

Operation ~2020?

Data:

- 1000-2000 3.2GPxl per night + calibration data -> 30TB/day
- Data is shared using dedicated 10 Gbits/sec lines
- Base: 3000 nodes with 16 cores to do initial data processing: within 60 sec to find moving processing to reorient the telescope to follow objects
- Once a day, data and metadata will be sent to NCSA (National Center for Supercomputing Applications, Illinois at Urbana-Champaign) for reprocessing and archiving (needs at least 100 teraflops of processing power and the capacity for 15 petabytes of storage)

- Funding: NSF, DOE and labs, universities, private investors
- Use DDS for control software already (OpenSplice)



NoVA

- NoVA: NuMI Off-axis ve (electron neutrino) Appearance experiment
- Will generate neutrino beams at FNAL and send it to a detector in Ash River, Minnesota (500 mile in < 2 ms)
- DOE funded (many labs and universities)
- RMS (Responsive Messaging System) is DDS-based system to pass control and status messages in the NoVA data acquisition system (two types of messages but has many actual topics to implement point-to-point communications)
- Very timid QoS requirements so far but will eventually need to go over WAN, and provide 10 Hz status transmissions between ~100 applications
- Simplifies OpenSplice using traits (like simd-cxx) to minimize the amount of data types and mapping topics to strings

SciDAC-II LQCD

- LQCD: Lattice Quantum Chromodynamics (computational version of QCD: a theory of strong interaction involving quarks and gluons making up hadrons like protons and neutrons)
- DDS is used to perform monitoring of clusters doing LQCD calculations (detect job failures, evaluate nodes loads and performance, start/kill etc)
- Topics for monitoring and controls of jobs and resources
- Use OpenSplice
- Noticed a problem as run out of memory with the durability on: need to use databases instead of stretching DDS

Common questions in bridging scientific apps and DDS

- RT is not that stringent (or underestimated)
- Common usability issues
  - Support for scientific data formats and data products (from and out of topics): domain data models and data transformation tools
  - Control and monitoring topics (can we come up with reusable data model?)
  - Simple APIs corresponding to expectations of scientists
  - Ease of modification (evolving systems not just production systems)
  - QoS cookbook (how to get correct combinations)
  - General education
    - Is DDS good for point-to-point (Bill talks only to Pete)
    - How one uses DDS without killing the system (memory etc)
- Common requirements
  - Site and community specific security
  - WAN operation (Chicago and Berkeley)

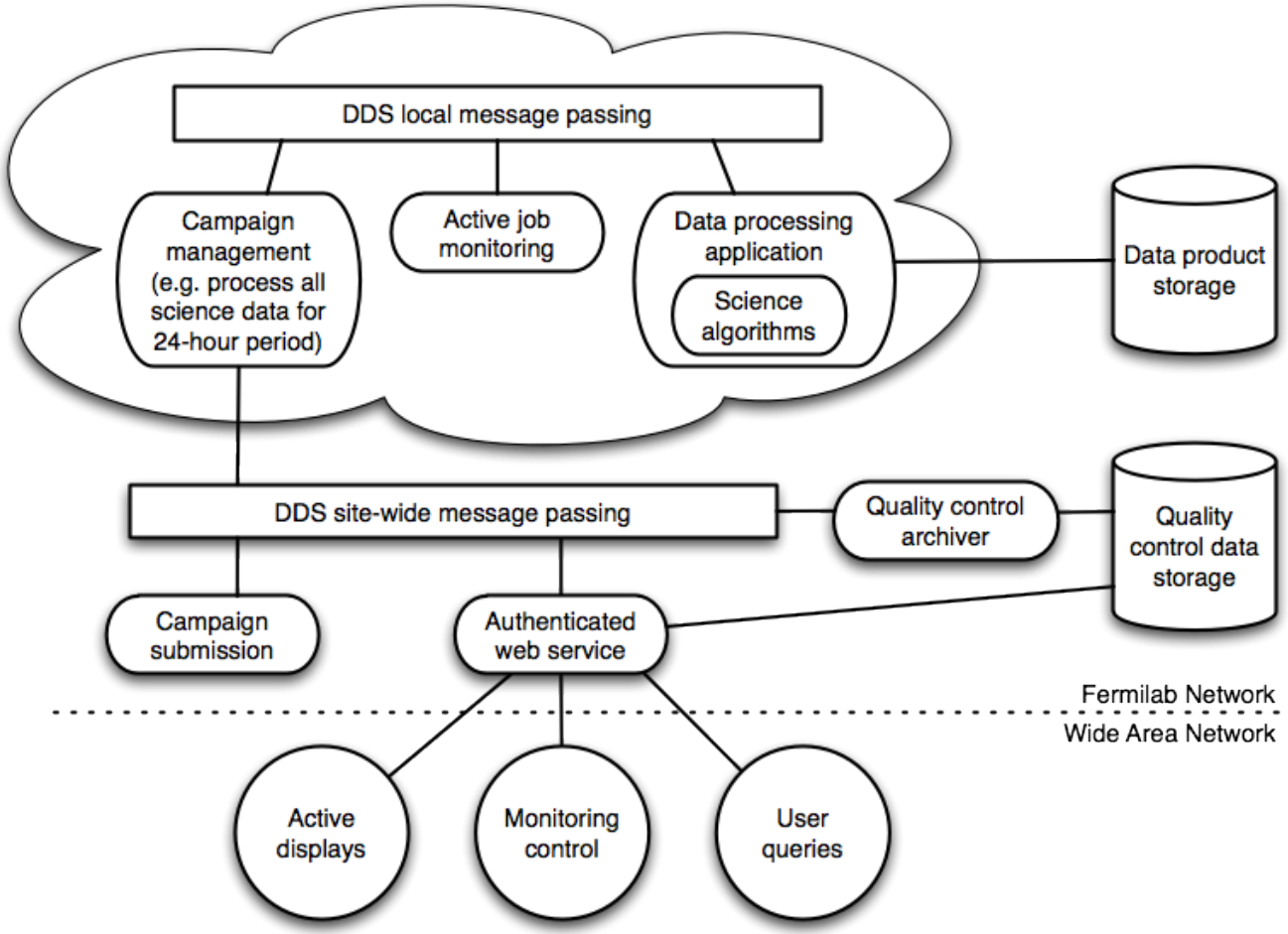
Common extra expectations

- Automated test harness:
  - How one tests for correct behavior and QoS
  - Avoid regression in rapidly evolving system modified by a team
- Interacting with databases (all data should be archived and allow queries)
- Can we do everything using DDS to minimize external dependencies?
  - Efficient bulk data transfer
  - Workflow engine (workflow: loosely coupled applications often through files and can be distributed, while simulations is typically tightly coupled on a HPC resource)
- Interacting with Web interfaces and Web Services

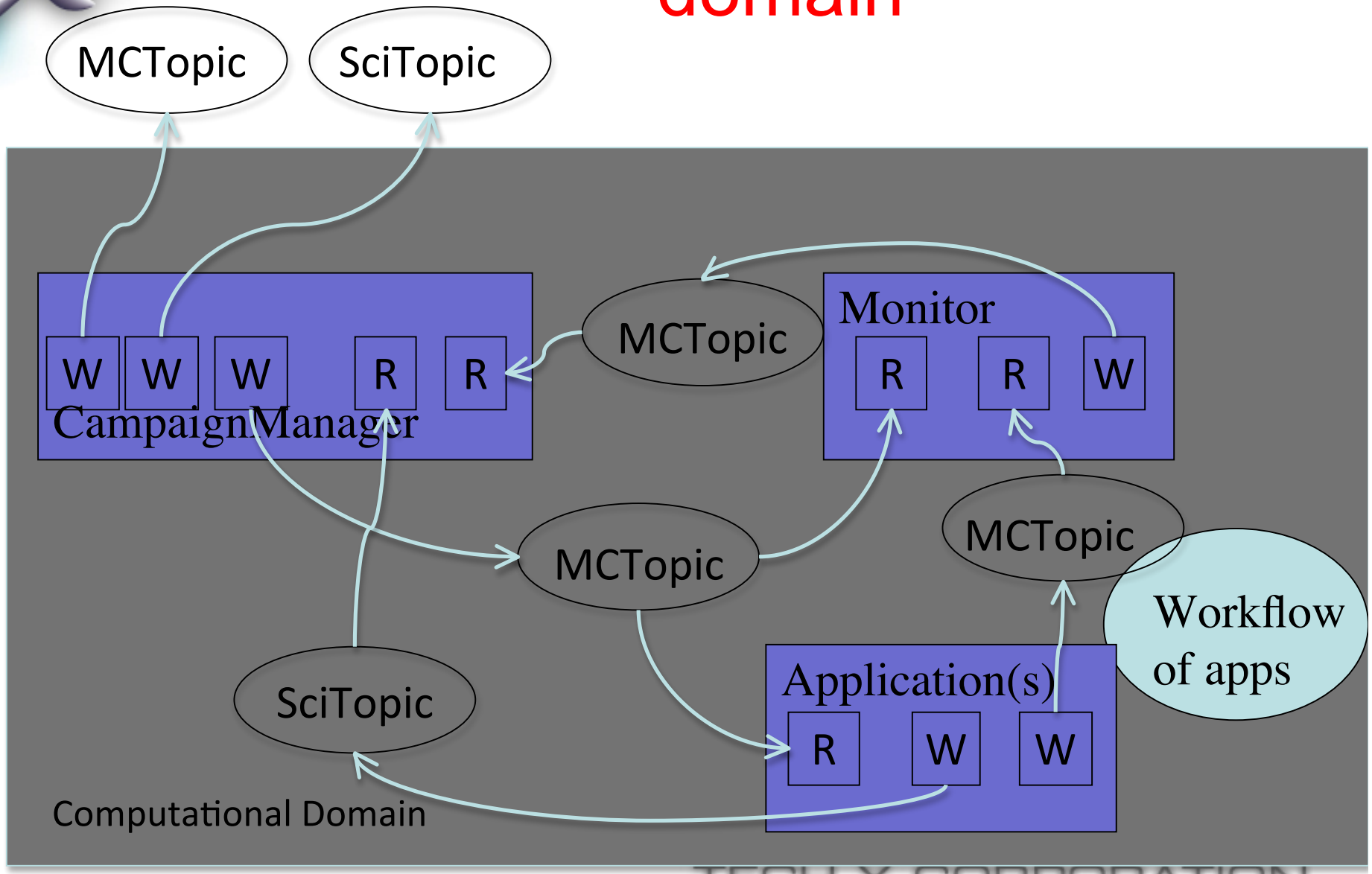
QUIDS tries to address some issues

- Phase II SBIR from DOE (HEP office)
- Collaboration of Tech-X and Fermilab (connection to LSST, NoVA and LQCD SciDAC)
- Goals (we will talk about the ones in red in what follows):
  - Implement a DDS-based system to monitor distributed processing of astronomical data
    - Simplifying C++(done with simdcxx) and Python APIs
    - Support for FITS and monitoring and control data, images, histograms, spectra
    - Security
    - WAN
    - Testing harness
  - Investigate of of DDS for workflow management

QUIDS, bird eye view (courtesy of FNAL)



QUIDS at FNAL computational domain



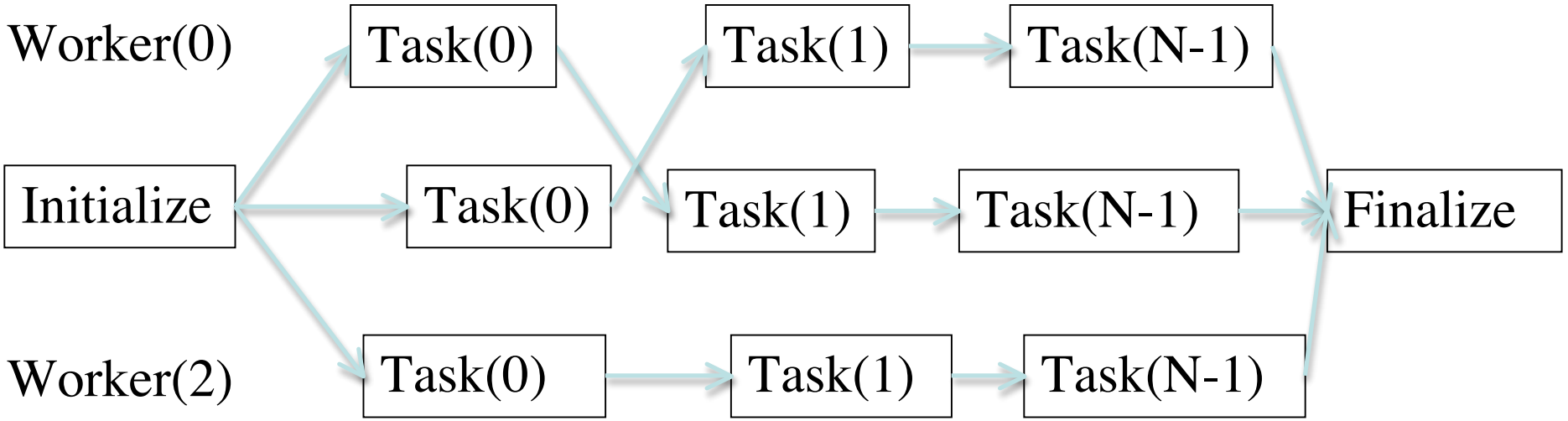
Generic workflows require heavy and hard to use tools

- Kepler (de-facto workflow engine expected for DOE applications):
  - Support for complex workflows
  - Java based
  - Heavy and hard to learn
  - Not portable to future platforms (DOE supercomputers might not have Java at all)
- Real workflows in astronomy are Kahn Network Processes
  - Pipelines
  - DAGs
- How one implements such workflows using DDS?

Parallel Pipelines Cover 90% of Astronomy Workflows

- Parallal pipeline job consist of
  - Initialization phase (splitting data into manageable pieces) running on one node
  - Parallel worker processes doing data processing tasks (possible not sharing the same address space)
  - Finalization step (merging data into a final image or movie)
- There is an expected order in tasks, so that tasks can be numbered and output data of a previous step as input to next
- Design decisions for now:
  - Workers do not communicate to each other
  - Workers are given work by a mediating entity: tuple space manager (no self-organization)
  - No scheduling for now (round-robin: tasks and workers are queued in the server)
  - Workers can up data coming from a task completed by a different worker (do not address the problem of data transfer now)
- All communication is via DDS topics
- Data to process is available through local files to all workers

Job

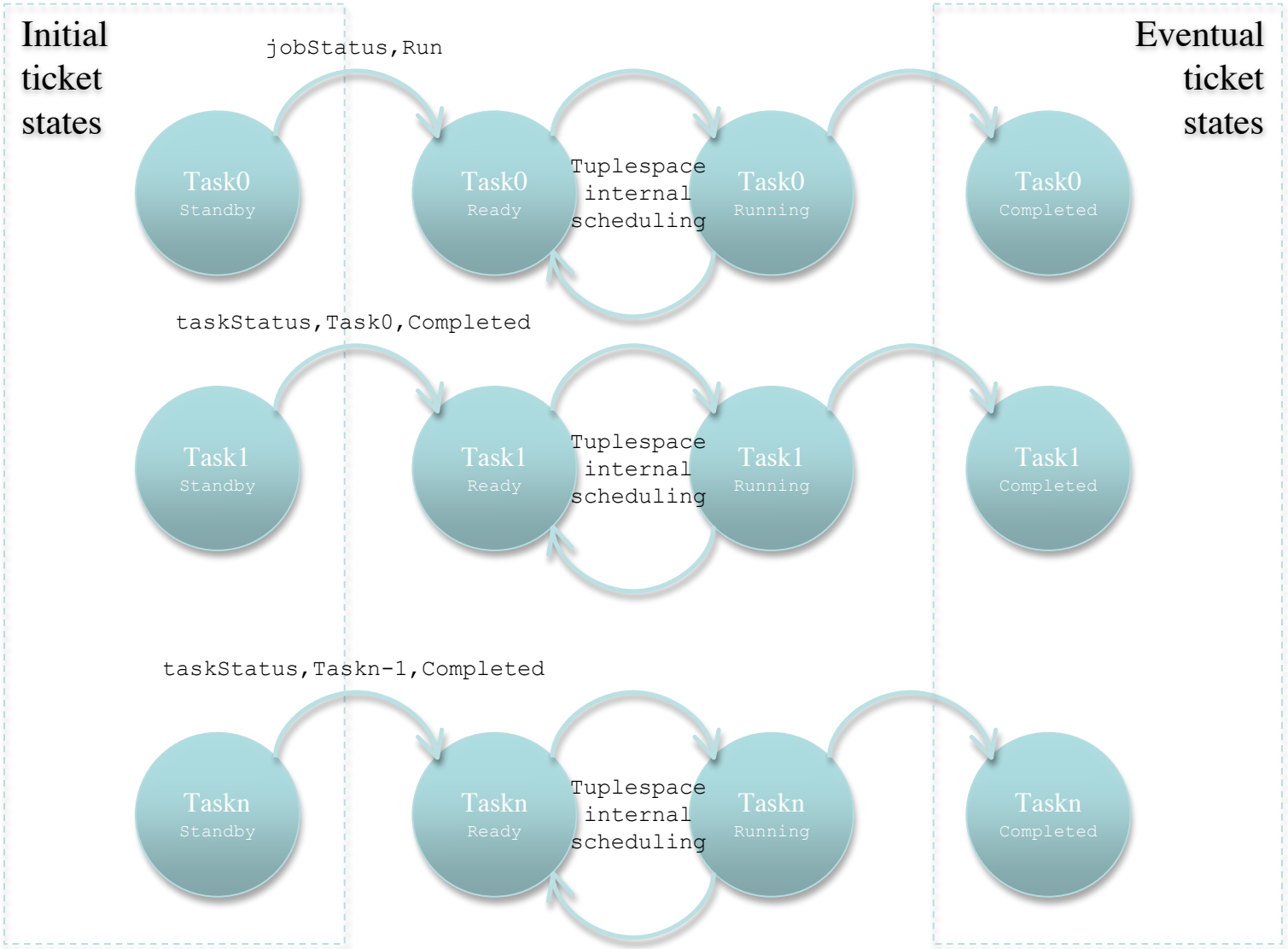


Tasks can be continued by different working processes: data can be passed between them (the Worker(1) performs Task(1) using data from Worker (0))

Interaction entities uses topics (no direct exchange between workers associated with processes

- Initializer
  - Create and publish all initial tickets
  - Initiate jobStatus change to running
- Tuple Space Server
  - Reflects the state of the system (what tasks should be done in which order and who is available to work) and manages the state changes
  - Schedules and issues tasks to available workers and maintains tasks and workers queues
- Workers
  - Publish their status (running, idle)
  - Listen to task assignment (workTicket)
  - Update tickets status
- Finalizer
  - Listen to jobStatus to initiate finalization
  - Publish jobStatus to indicate disposibility of job tickets
- Ticket
  - ids, in and out data, state of tasks and workers (standby, scheduable, running, done)

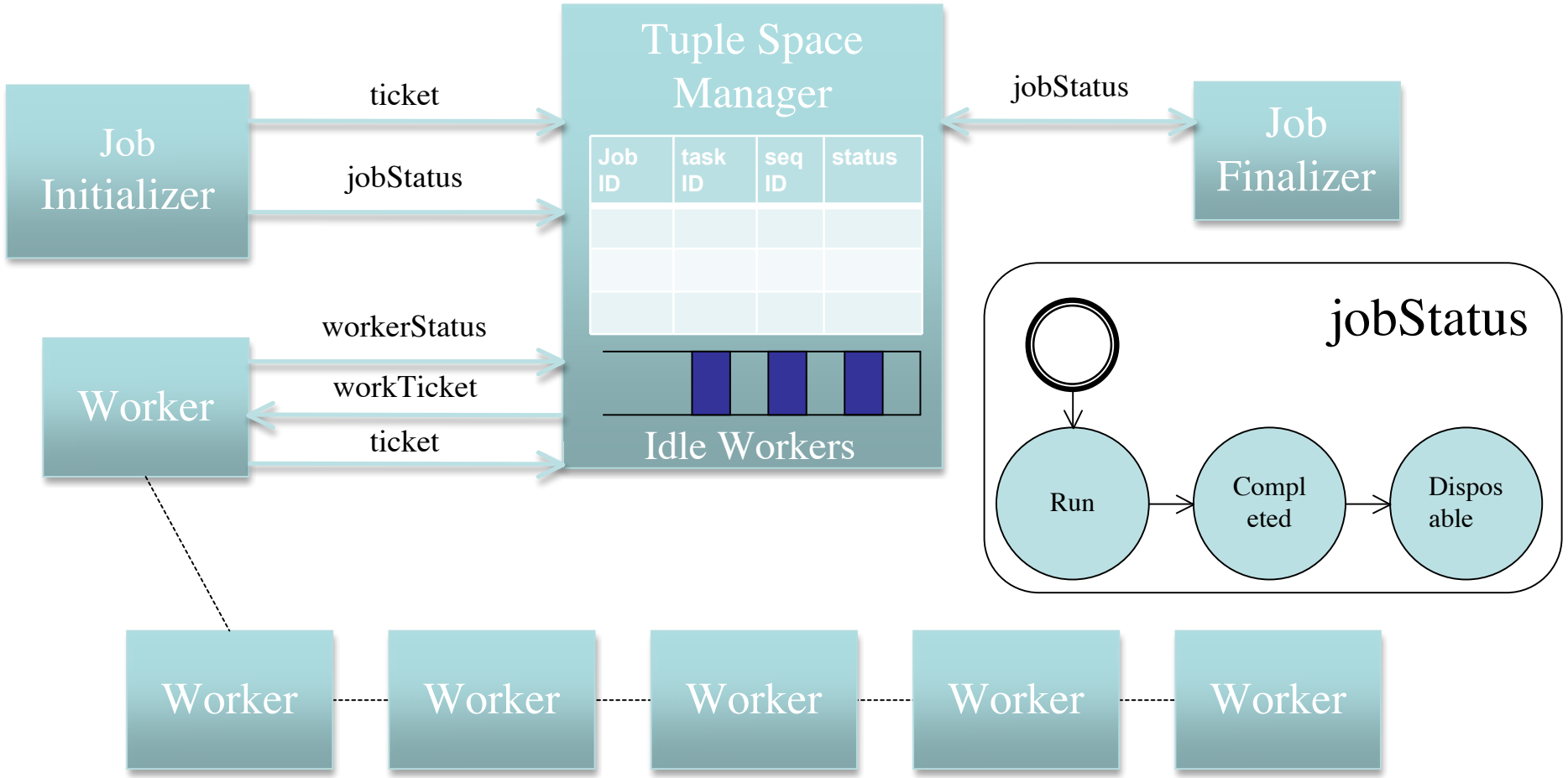
States of Tasks in Tuple Space



Taks executed sequentially

Sequences proceed independently in parallel

Tuple Space Manager Maintains Tasks Tickets and Schedules Tasks



Status of workflow engine

- Prototype working
  - Although we do have some issues with memory
- Next steps
  - User definition of workflow
  - Separation of worker manager from task manager?
  - Implementing workers doing slit-spectroscopy based on running R
  - DAG support
  - Some scheduling (balance between data transfer and mixing data between slow and fast workers?)
  - Data transfer implementation and in-memory data exchange

Security: “the better the fur is, the more expensive it is!”

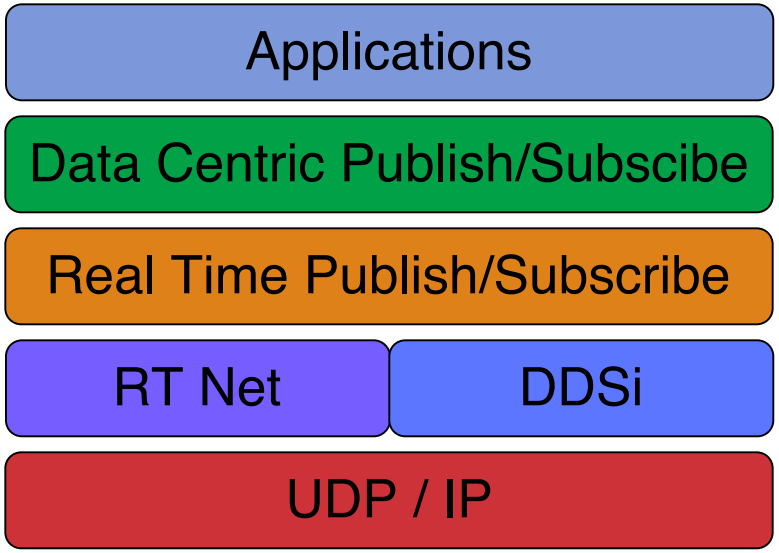
- Enterprise edition provides Secure Networking Service which is adequate for sci apps but is not free:
  - Security parameters (i.e., data encryption and authentication) are defined in Security Profiles in the OpenSplice configuration file.
  - Node authentication and access control are also specified in the configuration profile.
  - Security Profiles are attached to partitions which set the security parameters in use for that partition.
  - Secure networking is activated as a domain Service
- Scientists are not used to pay ☺

Providing security in community edition of OpenSplice

- Tried to replace the the lower networking layer of OpenSplice with OpenSSL and see how one can provide authentication, authorization and encryption
- OpenSSL is an open source toolkit:
  - Secure Sockets Layer
  - Transport Layer Security (new standard, replacing SSL)
  - General purpose cryptography library
  - Public Key Infrastructure (PKI): e.g., certificate creation, signing and checking, CA management
  - Datagram Transport Layer Security (new): UDP version of TLS which only runs over TCP (as does SSL)

OpenSplice Architecture

OpenSplice DDS Community Edition

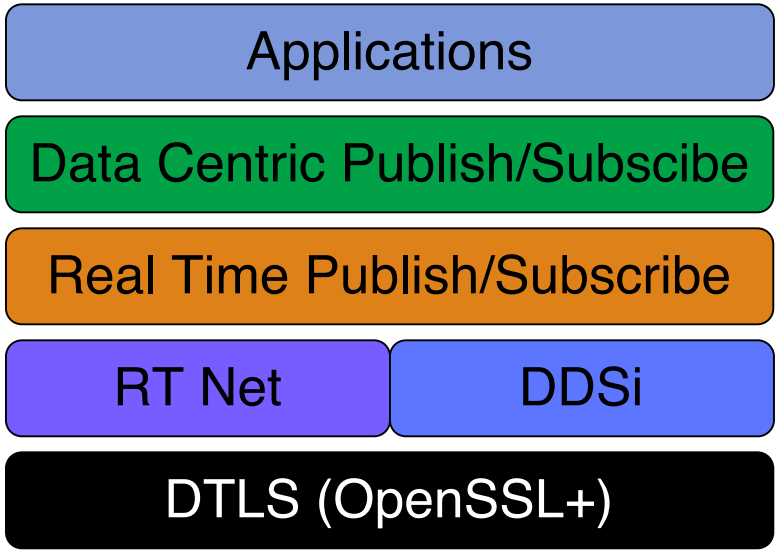


- All networking code is contained in the bottom two layers.
- The Real-time networking or DDSi layer is used to create or read each packet's payload.

- The UDP/IP layer handles the interface to the operating-system's socket interface.

OpenSplice with OpenSSL: authentication using certificates

Secure Version with implementation at UDP / IP layer



- Implemented a single tunnel for a two-site test
- Configuration could be read from ospl.xml

- Advantage: captures all DDS traffic
- Disadvantage:
  - Security parameters limited to one set per node pair
  - Point-to-point configurations (the show stopper)

Future Directions in Security Work

- Concluded that security profile coming with OpenSplice should suffice
- Intend to implement QuIDS security using OpenSplice security profile
- Explore user_data etc fields to address applications specifics if this is not addressed by the security profile

PyDDS: Python bindings for DDS communications

- Tried SWIG for wrapping generated bindings: works fine but needed manual wrapping of multiple generated classes
- Tried Boost.Python for wrapping of communication layer (set of classes that are used in the IDLPP generated code to call into OpenSplice for communication) so that there will be no need to wrap generated bindings
- Problem: need to take care of inheritance manually and deal with several handlers that are unexposed C structs used in forward declarations

Status and next steps for PyDDS

- Hand-wrapping of C++ bindings using SWIG works:

```
#!/usr/bin/python
import time
import TxddsPy
qos = TxddsPy.Qos()
qos.setTopicReliable()
qos.setWriterReliable()
writer = TxddsPy.RawImageWriter("rawImage")
writer.setTopicQos(qos)
writer.setWriterQos(qos)
data = TxddsPy.rawImage()
writer.writeData(data)
```

- Next:
  - Investigate wrapping communication classes so that we expose minimum for Boost.Python
  - Or develop tools for generating glue code needed to Boost.Python using a string list of topics

QUIDS summary and future directions

- We have prototyped DDS-bases solutions for astronomy data processing applications
  - Tools for bringing FITS data into DDS
  - Simple QoS scenarios (getting prepublished data)
  - Parallel pipeline workflows
  - Security studies
  - Python APIs
- Possible next steps
  - Language to describe workflows for user input into the system
  - More complex workflows and concrete implementations
  - Implementing FNAL security requirements
  - WAN communication with LBNL
  - Streamlining glue code generation for Python
  - Testing harness
  - Bulk data transfers
  - Archiving data into databases
  - Web interfaces

Acknowledgements

- JDEM Ground Data System (GDS) team from Fermi National Accelerator Laboratory
- PrismTech (help with licenses)
- OpenSplice mailing list and its contributors – very helpful
- US Department of Energy, Office of High Energy Physics