DECam Community Pipeline Review

NCSA

August 30-31, 2010

DTS – The NOAO Data Transport System

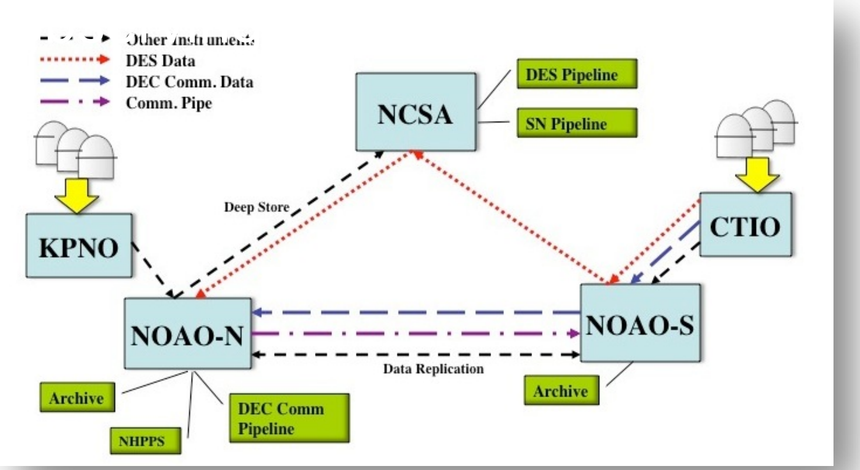
NOAO SDM

Mike Fitzpatrick

DARK ENERGY
SURVEY

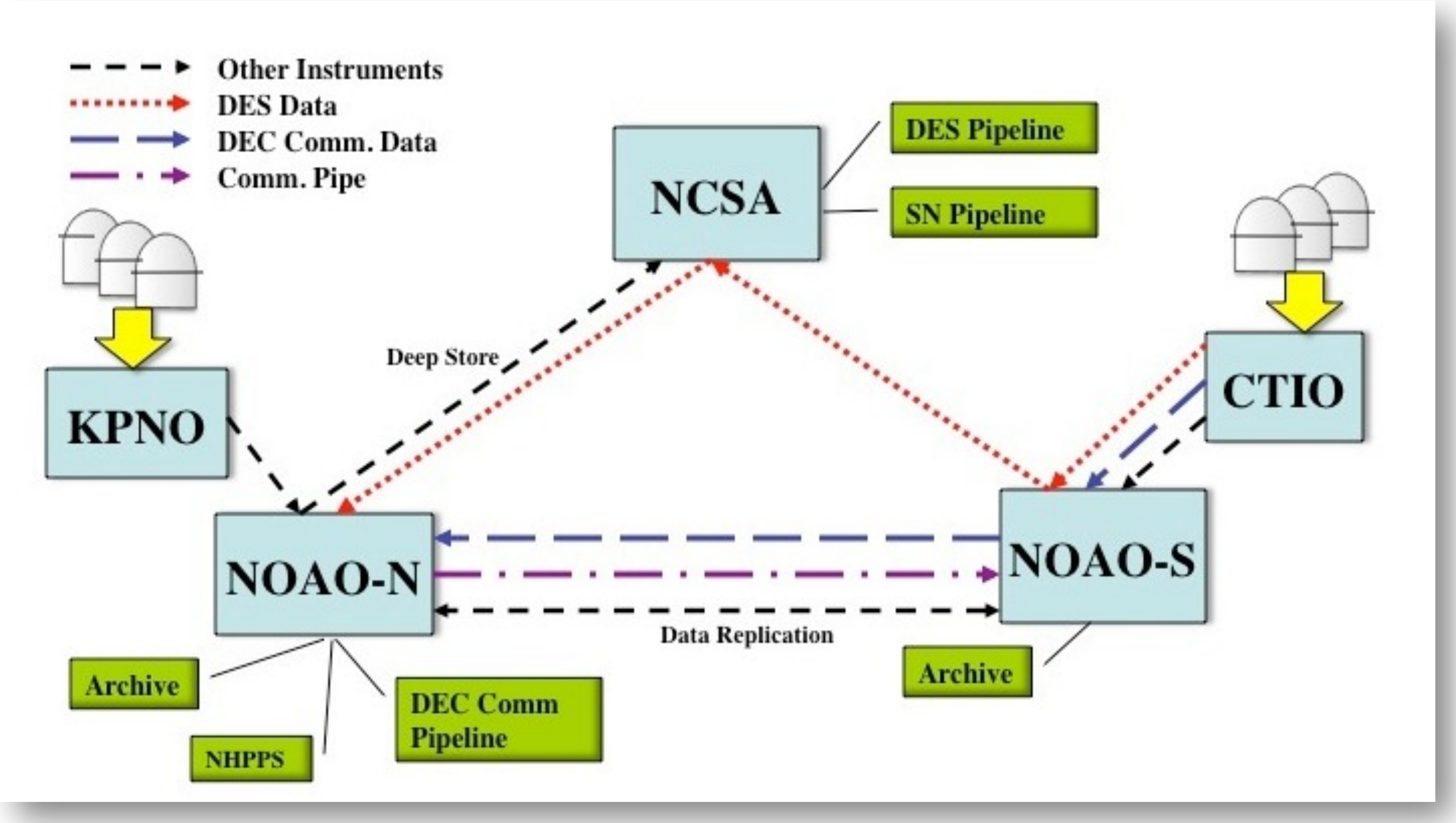
DTS Project Background

DTS is designed for DECam, but it will eventually be used throughout the NOAO E2E system:



- Must make better use of network
- Must allow data to be routed based on DES/Community use
- Must interface with both NOAO and DECam systems
- Must be configurable for data rates and sizes other than DES
- Must operate in variety of network environments

DTS Dataflow



Other Instruments

DES Data

DEC Comm. Data

Comm. Pipe

NCSA

DES Pipeline

SN Pipeline

KPNO

CTIO

Deep Store

NOAO-N

NOAO-S

Data Replication

Archive

NHPPS

DEC Comm Pipeline

Archive

DTS Requirements

- Must transfer a *typical* DES night in 18 hrs. So, what is *typical* ?
  - 350 images (60 are bias/flat calibrations, remainder are science frames in various filters)
  - Differing sizes
    - ~389 MB for a Bias thru ~701 MB for *z*-band object
    - Average file size: 589 MB
    - Total for the night: ~206 GB
- Must play nice with the rest of the network
  - To meet transfer requirement, we need a sustained 27 Mbps assuming we use only 75% of current bandwidth
- Must interface with external entities
  - Not all of which are NOAO-controlled

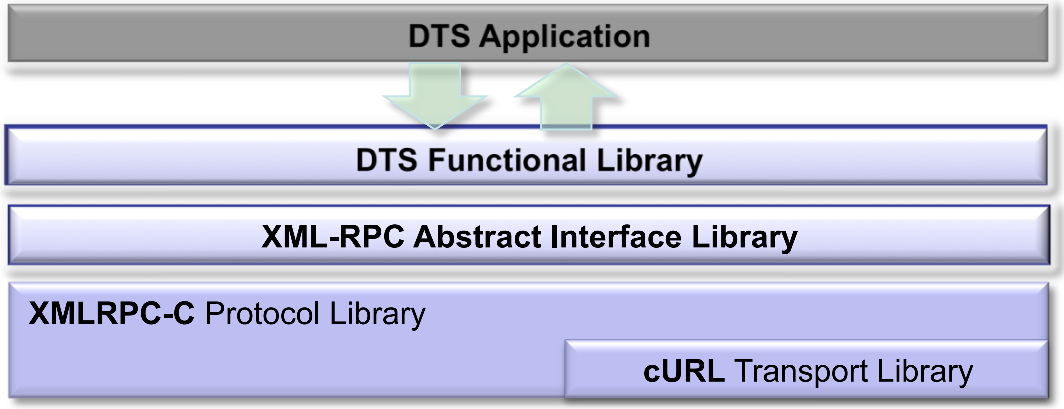
DARK ENERGY SURVEY

External Entities

- At the telescope
  - SISPI must queue the data for transport
- On the mountain
  - iSTB must modify the headers to add archiving keywords
- NOAO-S (La Serena)
  - Save copy of raw data saved to local data store
- NCSA / DES-DM
  - Delivery to DM for (DES) pipeline processing
  - Separate Delivery for SN pipeline (?)
- NOAO-N (Tucson)
  - Ingest into NSA
- Ad-hoc Transport
  - SLAC-Fermi for eng, DB sync, PreCam, etc

DTS Implementation

- SOA based on XML-RPC
  - Provide and consume basic services without requiring persistent connections
  - Can be configured as *Unix* service for automatic recovery in case of crash
  - Allows for remote monitoring and management
  - Client applications are language independent
  - Common DTS process providing services at each site
  - Highly multi-threaded and configurable
- Written in ANSI-C
  - Self-contained
  - Currently ~30 KLOC



DTS Application

DTS Functional Library

XML-RPC Abstract Interface Library

XMLRPC-C Protocol Library

cURL Transport Library

DTS Features

- XML-RPC Based
  - Designed for remote monitoring/control, automated operation
  - Fully configurable at each node in the network
- Multiple, independent transport queues
  - Continuous FIFO, scheduled or priority
  - Queues define the data routing, transfer can be *tee*’d
- Multiple transport methods permitted
  - Parallel TCP/IP sockets (current), UDP packets/other TBD
  - Configurable to optimize for few large files, many small ones or particular network characteristics (e.g. more threads for slow links)
- Allows processing at each stage in transfer
  - Interface with external entities
- Guarantees file integrity and graceful error recovery

DARK ENERGY SURVEY

Bulk Transport Models

The *transport model* is independent of *transport method.* One describes the direction of transport and the RPC activity, the other refers to the actual network protocol used.

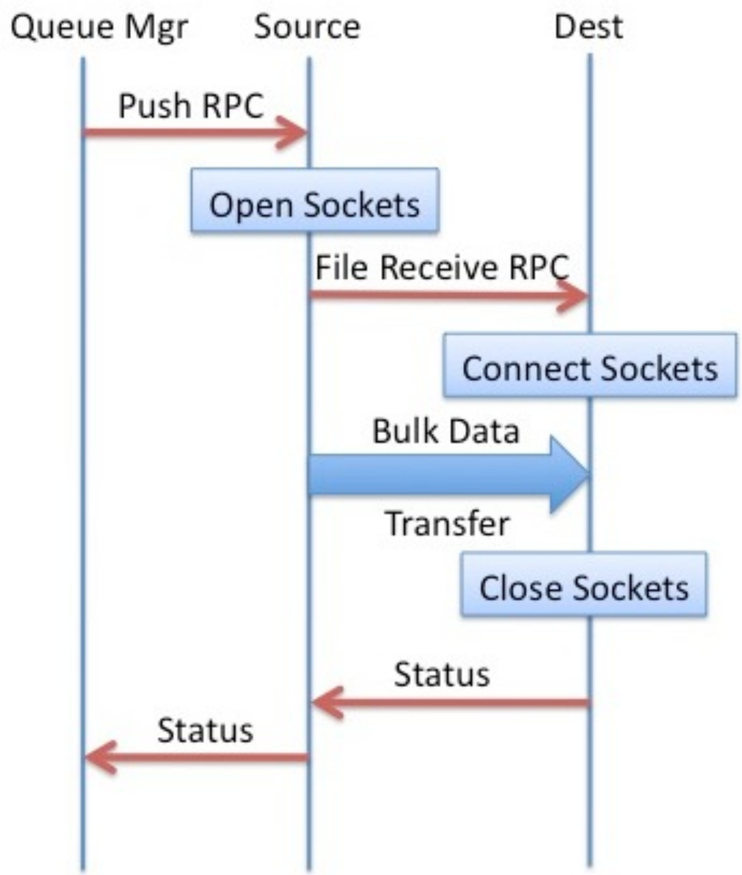
PUSH / PULL

- A *push-to* or *pull-from* a node assumes that node is offering a transport service and is willing to supply the transport sockets

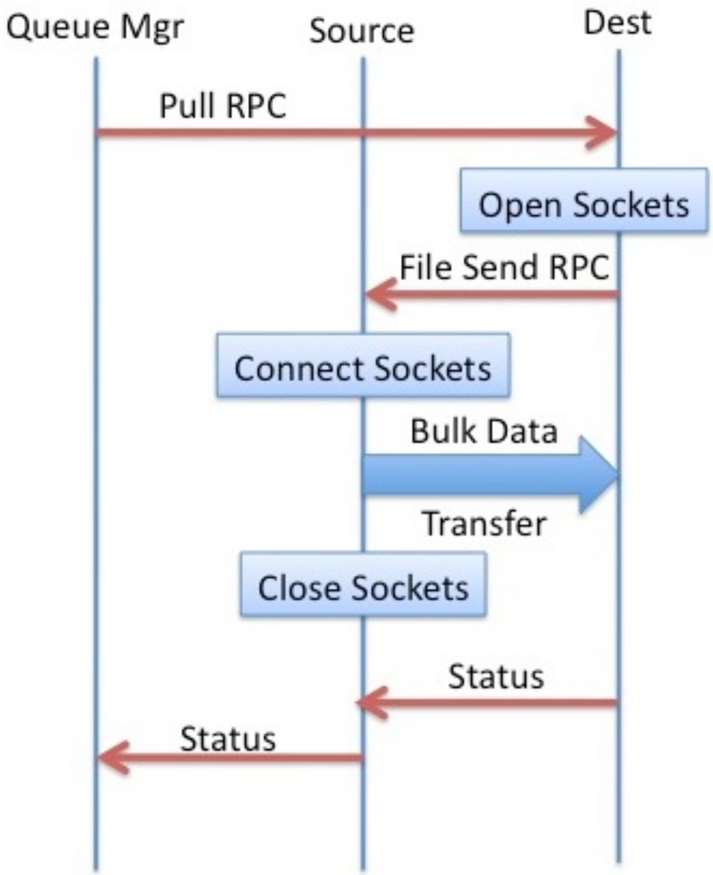
GIVE / TAKE

- A *give-to* or *take-from* a node is used when we're willing to provide the service and sockets to ensure successful transport

Bulk Transport Models



Push Transfer Model



Pull Transfer Model

Bulk Transport Methods

- Currently only implements a Parallel TCP/IP Socket transfer method
  - Number of threads allocated used to regulate bandwidth usage
  - Multiple levels of checksum validation allowed (not all used):
    - Chunk – data transferred in one request, e.g. 64Kb
    - Stripe – data transferred on each thread
    - File – whole-file checksum value
  - Multiple checksum values available (BSD/SYSV, MD5, etc)
  - No assumptions made about contents of files being transferred
- Other methods can (theoretically) be added for use in all transport modes

Queue Types & Modes

Queue Types:

- Ingest -- Entry point to DTS
  - Can modify data for before transport using Delivery App
- Transfer -- Midpoint node in the data path
  - Normal delivery mechanism
- Endpoint -- Terminal point of data path
  - Data can be removed from queue on completion

Queue Modes:

- Normal – FIFO processing as long as data available
- Scheduled – Transfers at specific times / intervals
- Priority – Blocks all other queues until complete

Configurability

Sample DTS (daemon) configuration file.

The configuration file is made up of one or more entries with the following structure:

```
 <global values>            # debug, verbose, etc
dts                         # define DTS node
  <dts parameters>          #
    queue                   # define a queue
      <queue parameters>    #
```

Configurability

Sample Queue Configuration

```
# Queue configuration
queue
    name            test              # queue name
    node            transfer          # ingest, transfer, or endpoint
    type            normal            # normal, scheduled, priority
    mode            push              # push or give
    method          dts               # dts or [TBD]
    nthreads        4                 # No. of outbound threads
    port            3001              # base transfer port
    src             denali            # source machine
    dest            tucana            # destination machine
    deliveryDir     /tmp/foo.crux     # spool or /path/to/copy
    deliveryCmd     /home/fitz/imtest -i $inst $F
```

DTS Components

① **DTSD** – The DTS Daemon

② **DTSQ** – The DTS Queuing Agent

③ **DTSH** – A DTS shell and command-line tool

④ **DTSMON** – A DTS monitoring application

DTS Components

DTSD – The DTS Daemon

- Provides all DTS services on the machine
  - assumed there is one per '*site*'
- Responsible for managing transport queues
  - separate threads manage each queue
- Requires only command port be open to firewall
  - Transport sockets can be managed in configuration
- Sandboxed filesystem view for security
- Can be run as *xinetd* service or run entirely in user-space
- *Set/Get* methods permit remote management, *Status* methods permit monitoring

DTS Components

DTSQ – DTS Queuing Agent

- Queues data for ingestion into the DTS system
- Provides quick response so it won't block caller
- Acts as a ***dtsd*** to provide its own transport methods to the DTS,
    - i.e. SISPI requires no other DTS components to be installed
- Logs all requests
- Verifies DTS status before allowing transfer
- Permits recovery of failed requests for later re-queuing
- Leaves a “token” file with details of transfer on success

DTS Components

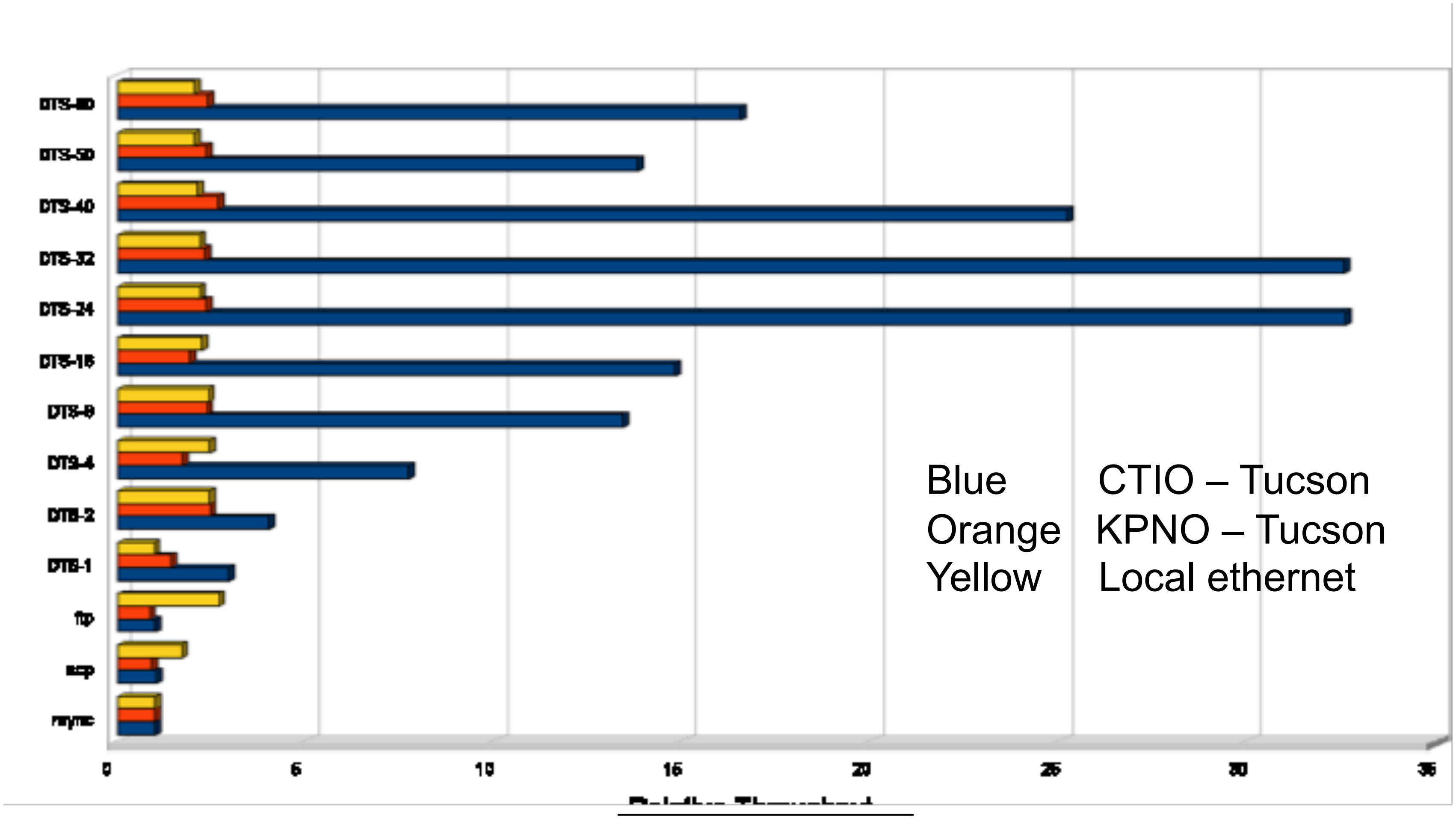
DTSH – A DTS shell and command-line tool

- Allows direct communication with a DTSD site
- Provides scripting capability for DTS commands
  - E.g. cron jobs to collect transport logs, monitor/restart services
  - Full access to all DTSD functionality

DTSMON – A DTS monitoring application

- Simple monitoring application
  - echoes messages generated by remote DTS components
  - Advanced tool / GUI planned for later development
- Runs on a reserved port
- Primarily meant for operations and development use
  - not required for routine use of DTS

DTS Relative Throughput



Blue	CTIO – Tucson
Orange	KPNO – Tucson
Yellow	Local ethernet

SISPI-DTS Interface

At the telescope: IB system invokes *dtsq* as e.g.

```
dtsq -q des -f inst=decam /path/image.fits
```

- The '*-q*' names the transport queue (i.e. *data route*) to be used
- The '-f' forks the command after verifying DTS is ready
  - transfer happens in child process
- The '*inst=decam*' shows use of a *pass-thru parameter*
- IB gets (near) immediate **OK/ERR** response
- On successful completion of transfer, details of transfer (time of request, time of completion, comments, etc) left on calling system logfile
- On error of transfer, request is logged for later recovery

DESDM-DTS Interface

At the DES (or SN) Pipeline:

- DTS invokes an external '*delivery application*'.
- This application:
  - *Requires* a single argument: the local path to the file
    - Additional parameters may be passed through from DTSQ
  - Is free to modify the file, the delivery app runs on a copy of the file in the configured 'delivery directory'
  - Must be provided by DES
    - DES knows best how to interface to their system

SDM-DTS Interface

On the mountain:

- iSTB invoked as '*delivery application*’ for DTS ingestion
  - Modifies image to add necessary archiving keywords
  - This modified image is what is transferred through the system

At NOAO Archive Centers:

- A TBD delivery application is used to trigger an NSA ingest of the file
  - File will be moved from delivery directory to final mass-store location

DTS Status

- Functional development complete
- Final documentation still pending
  - DESDM-DTS ICD document awaiting sign-off
- System currently undergoing stress-testing, e.g.
  - 10,000+ file transfers to eliminate memory leaks
  - Artificial failures of network connectivity, checksum failures, etc
  - Validate there is a proper recovery
- First real-world test expected with PreCam
- Likely requires small additional development for use in final E2E deployment
- Final throughput test waiting for CTIO network upgrade