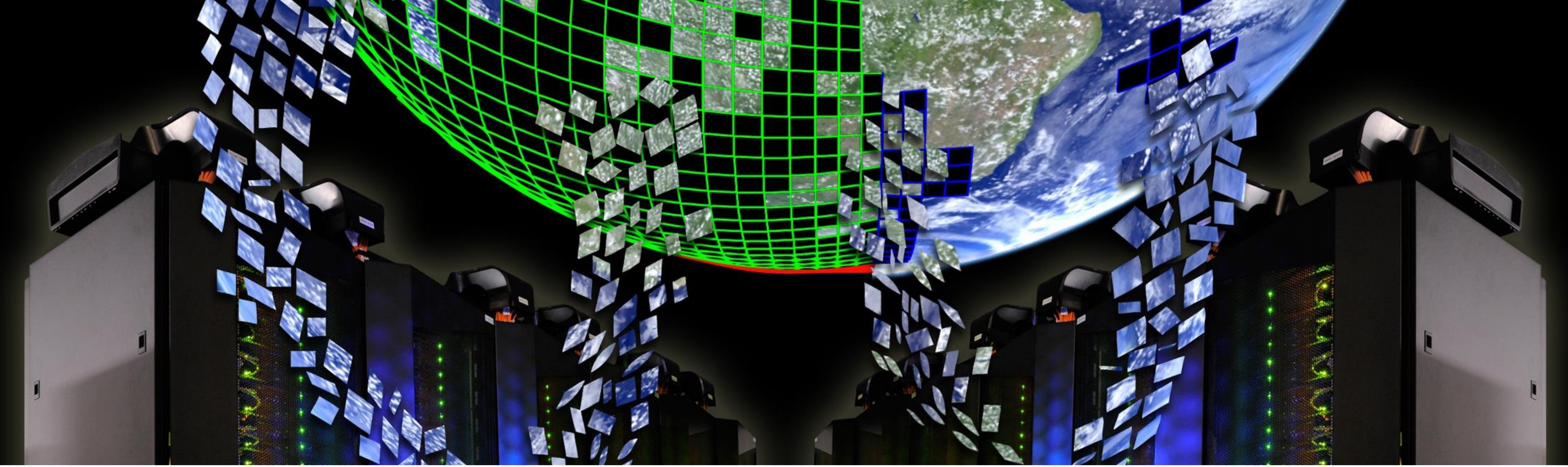


NASA Center for Climate Simulation (NCCS)

Leveraging Python Multiprocessing to Manage a Multi-Petabyte Raster Archive

June 8, 2023

Jim Shute, Ryan Forbes, Erin Goodnough

Agenda

- NCCS Mission
- NCCS Systems
- Functional Area Overviews
- GIS Platform Architecture
- NGA/Maxar/DigitalGlobe Data
- Ingest Process
- Challenges
- Development Workflow
- Standard Script Structure
- Demo
- Parallel Process Applicability and Results
- Multiprocessing Best Practices

NCCS Mission

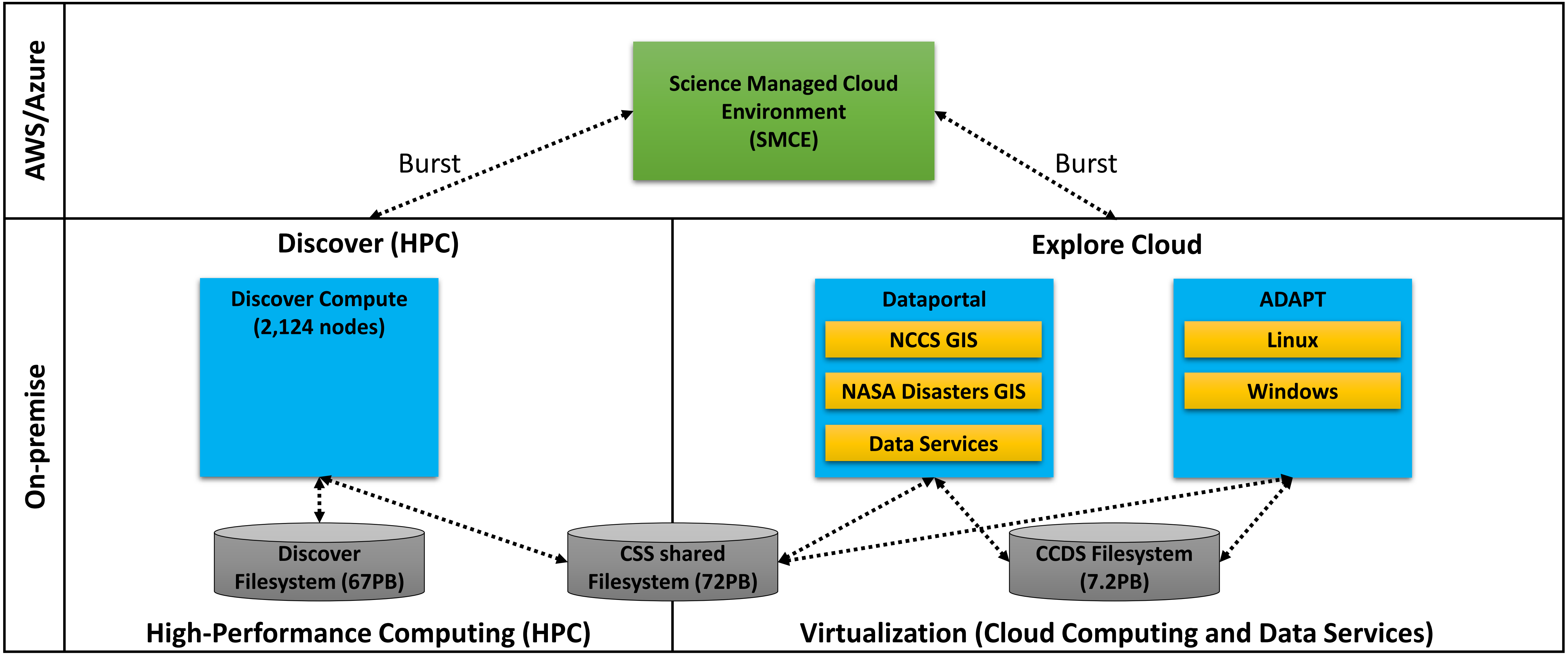
The NCCS provides high performance computing for NASA-sponsored scientists and engineers. Our integrated set of computational capabilities includes High Performance Computing, Cloud Computing, Analytics, Data Sharing and Tools, Visualization, and Climate Data Services. The purpose of the NCCS is to enhance NASA capabilities in Earth science, with an emphasis on weather and climate prediction, and to enable future scientific discoveries that will benefit humankind.

Building 28, Goddard Space Flight Center

Piers J. Sellers Data Visualization Theater

ABoVE Science Team Meeting (ASTM5), 2019

NCCS Systems

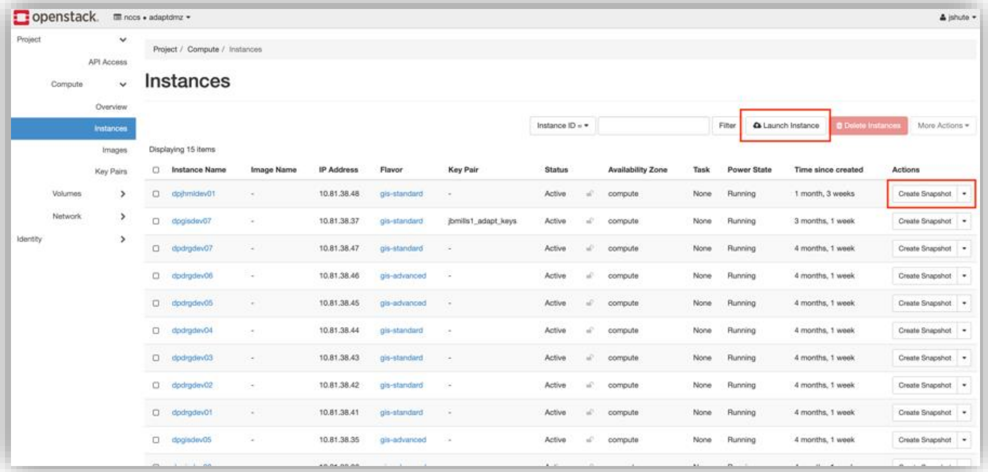


Functional Area Overview – Explore

- Next generation on-premise cloud
- Multiple availability zones (B28, B28D, B32)
- Platform specifications
  - 275 hypervisors
  - 8,184 CPU cores
  - 68.8TB RAM
  - 7.2PB storage

Explore Cloud Control Plane – Building 28

Explore Cloud OpenStack Web Interface



Functional Area Overview – Centralized Storage System

- Centralized data store, accessible by all NCCS platforms/subsystems
- 72PB total storage

CSS1 and CSS2 (30PB total usable storage)

- ABoVE: 80 TB
  - Oak Ridge National Laboratory Distributed Active Archive Center (ORNL DAAC) datasets, including over 130 archived as part of ABoVE.
  - National Snow and Ice Data Center (NSIDC) DAAC datasets, including LVIS products archived in support of ABoVE.
  - Alaska Satellite Facility (ASF) datasets related to L-band SAR.
- AMSR-2: 5 TB
- AVHRR/Polar: 40 TB on ADAPT and 10 GB on CSS
- CFHA: 250 TB
- CMIP5: 105 TB
- CREATE-IP: 79 TB
- CSDA-Spire: 30 TB

  Note: Access permission required, please contact NCCS support

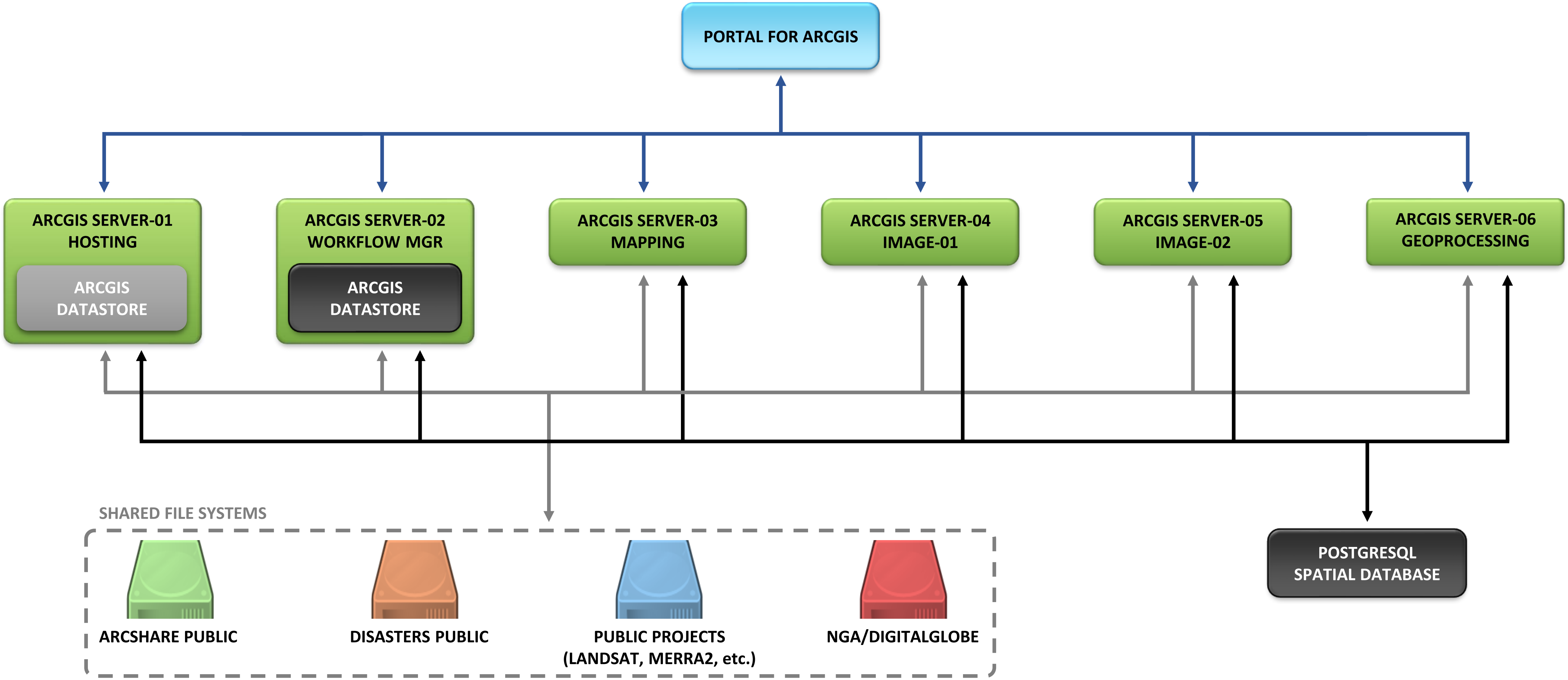
- DSCOVR: 72 TB (L1B)
- DSCOVR: 72 TB (L2_CLOUD_03)
- FLDAS: 40 TB
- GeoMIP: 14 TB
- Geostationary (GOES): Ingest starting now, planning for 1 PB
- GEOS-IT: 420 TB

  Note: Public access coming soon.

- GEOS-5 Nature Runs (g5nr): 5 PB
- HIMAT Snow Reanalysis: 5 TB
- ICEBridge: 2 TB
- ICESat: 8 TB
- ICESat-2: 161 TB
- IMERG: 15 TB
- Landsat: 186 TB
- MAIAC: 107 TB

Subset of datasets stored on CSS

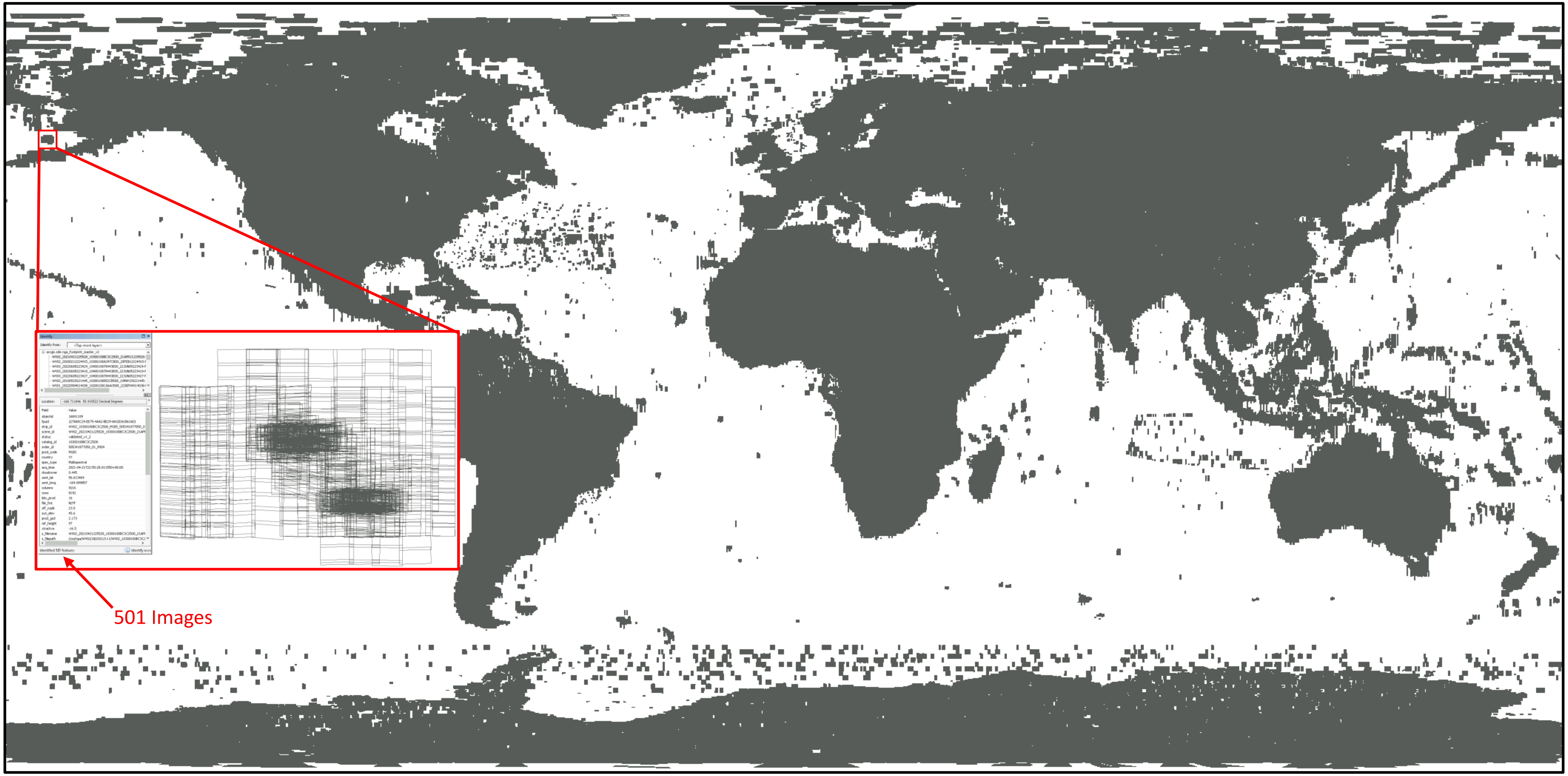
GIS Platform Logical Architecture



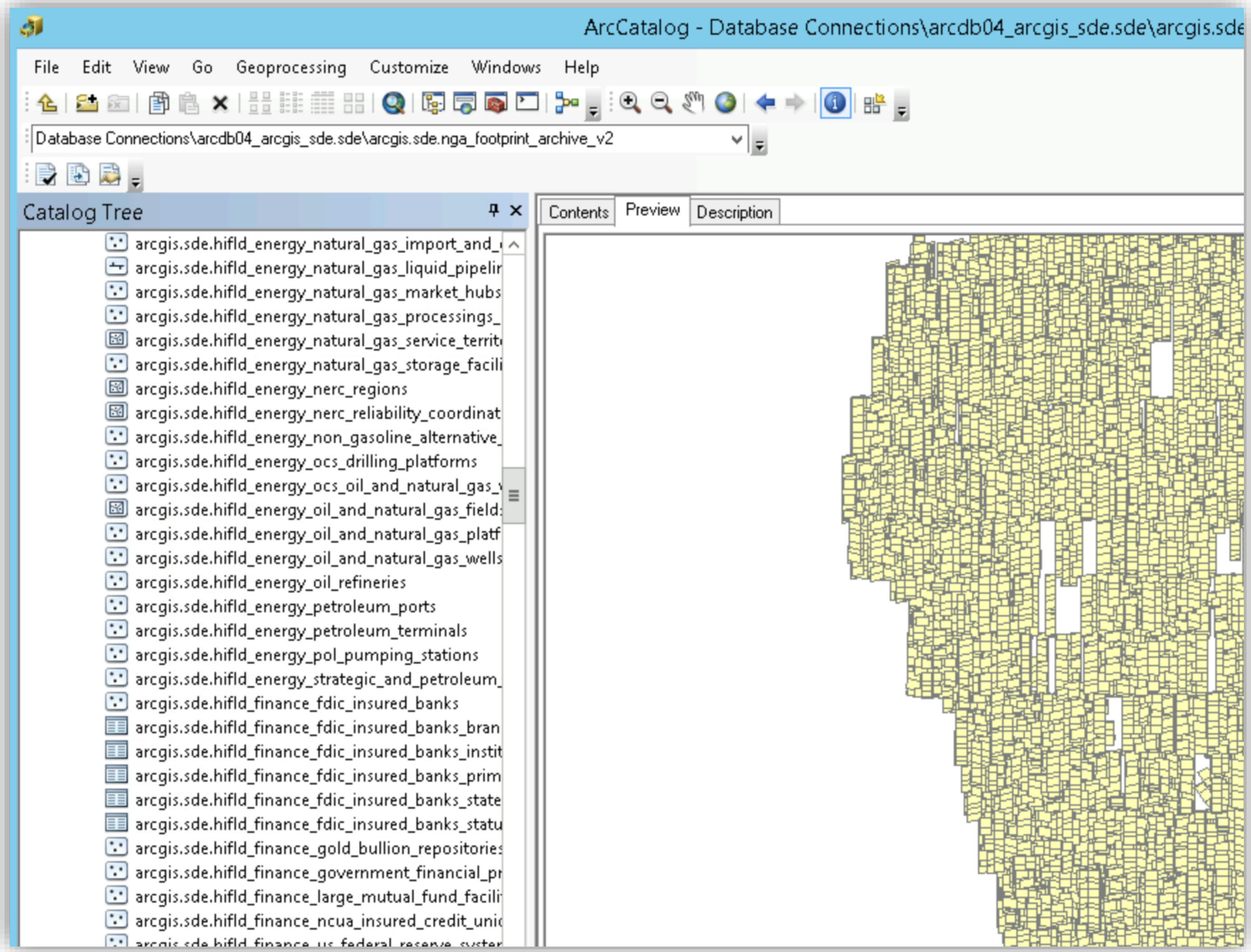
NGA/Maxar/DigitalGlobe Data Statistics

- 24.8 million images
- 14.5 petabytes
- Sensors
  - GE01 (2006859)
  - IK01 (320056)
  - OV03 (227)
  - QB02 (2304775)
  - WV01 (5353377)
  - WV02 (10392289)
  - WV03 (4416135)
  - WV04 (12455)
- Years
  - 1999 (10), 2000 (25,470), 2001 (34,011), 2002 (108,325), 2003 (172,904), 2004 (172,543), 2005 (177,619), 2006 (200,988), 2007 (270,816), 2008 (504,240), 2009 (600,408), 2010 (1,005,111), 2011 (1,477,337), 2012 (1,703,910), 2013 (1,592,369), 2014 (1,454,682), 2015 (1,612,178), 2016 (1,780,965), 2017 (1,837,000), 2018 (1,591,966), 2019 (1,382,450), 2020 (2,124,463), 2021 (2,337,951), 2022 (2,190,762), 2023 (447,695)

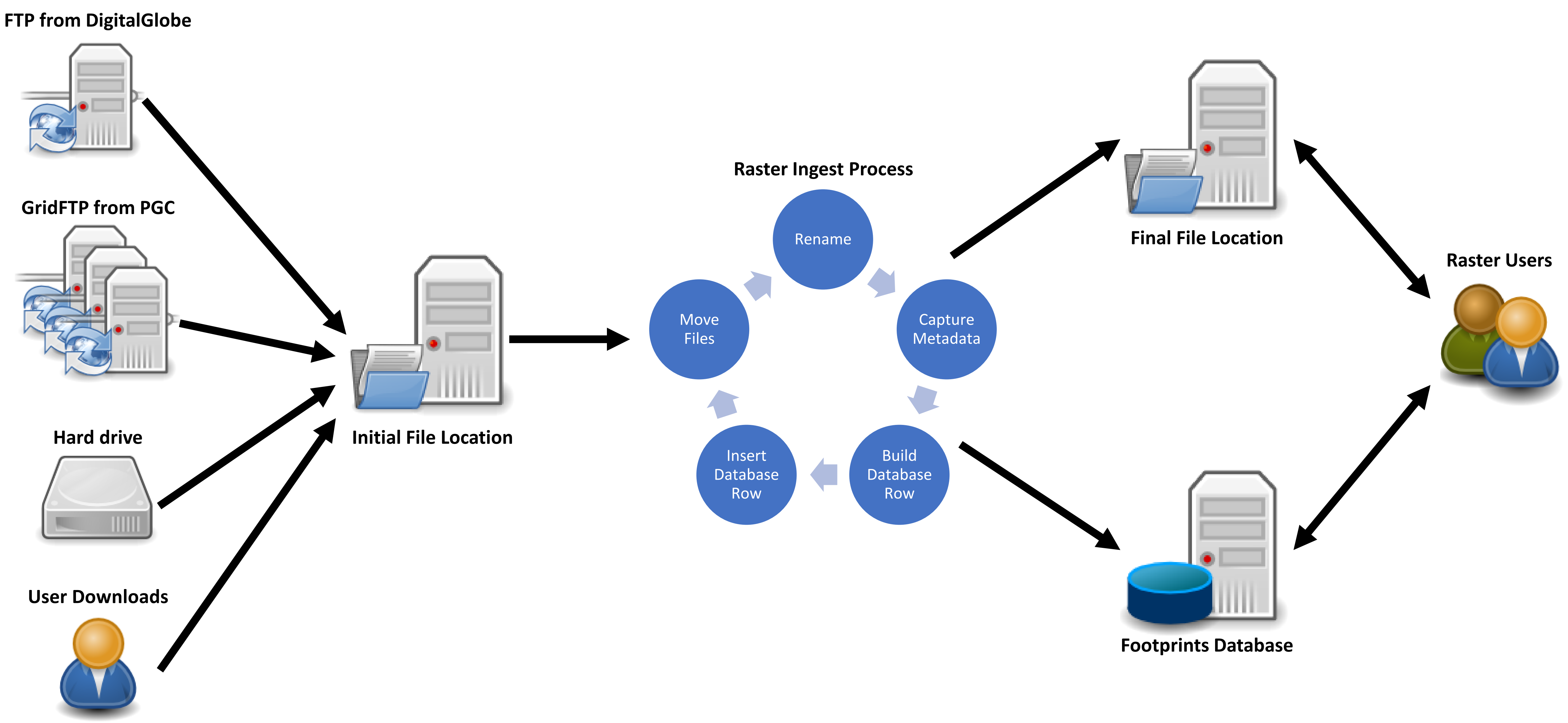
NGA/Maxar/DigitalGlobe Data Coverage



NGA/Maxar/DigitalGlobe Data Coverage

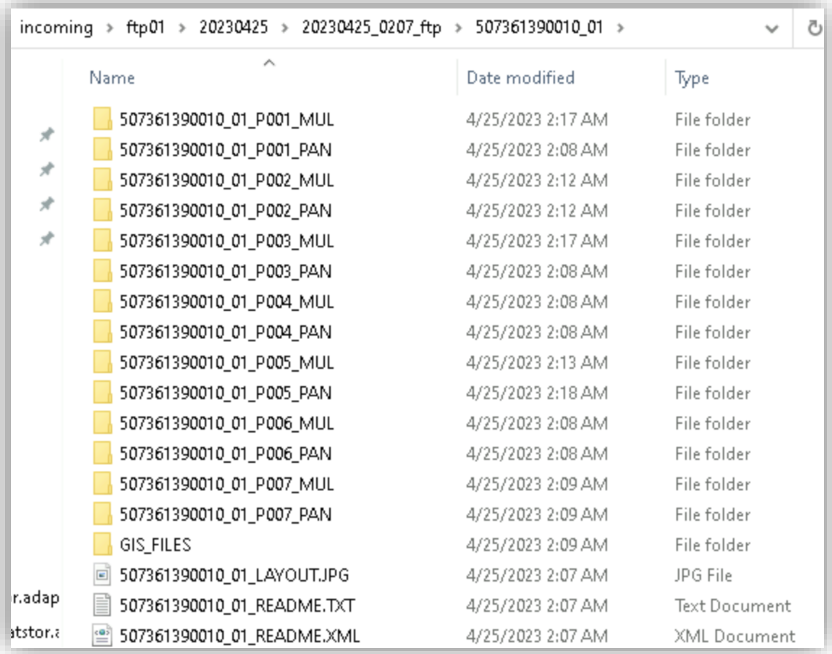


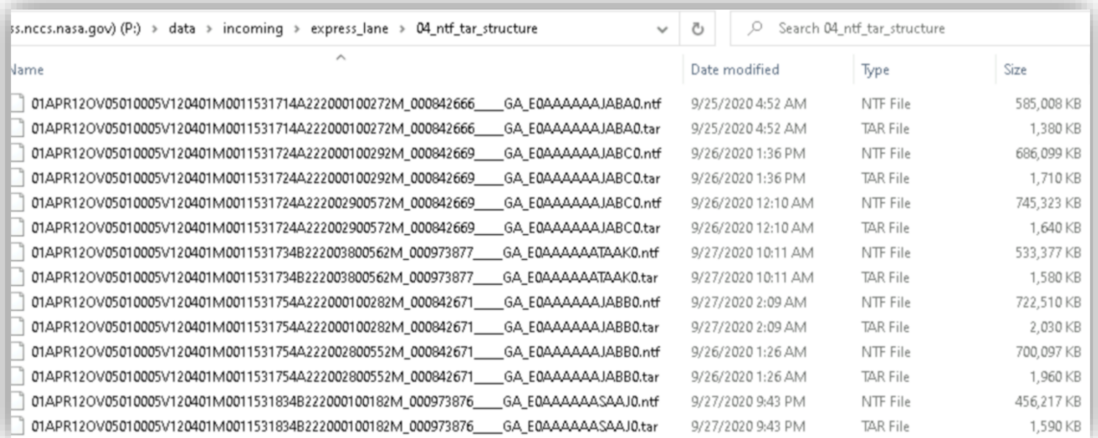
Ingest Process



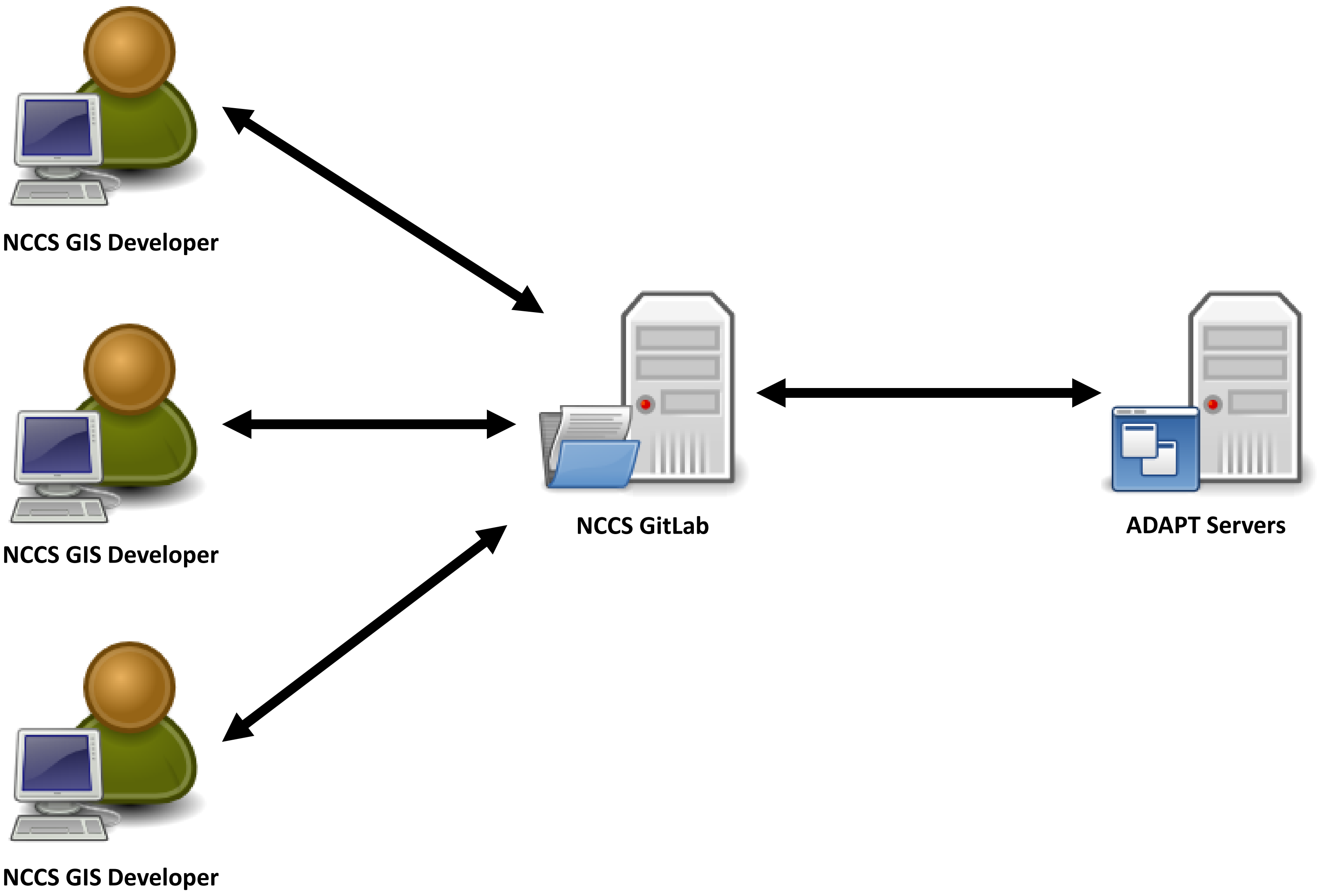
Challenges

- Multiple data streams and formats (e.g., "renamed" data vs. multiple raw file structures)
- At the beginning of the redesign process, image backlog was 1.1PB
- Slow ingest rates of 2,000 rows/hour
  - Backlog would have taken over 1 year
- Process and resulting database needed to meet the needs of multiple user groups
- Bad/duplicate data was complicating users' raster analytics workflows





Development Workflow

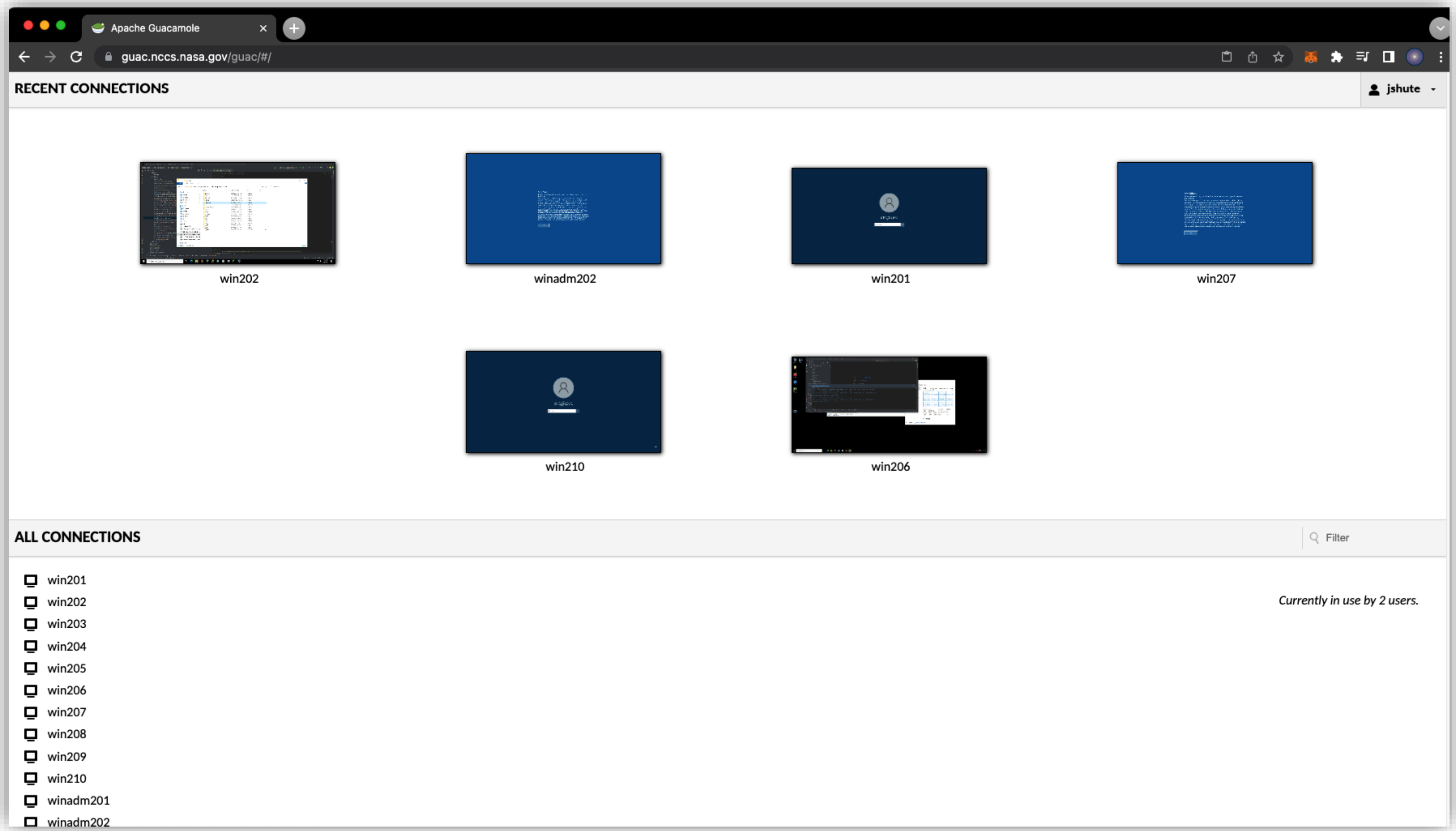


Standard Script Structure

1. Notes/purpose
2. Import modules
3. Configure global variables (if needed)
4. Supporting functions
5. Main function
    a. Capture script parameters (if needed)
    b. Import special script/class files (e.g., UtilitiesGeneral)
    c. Global logger settings
    d. Set up the main logger
    e. Check script parameters
    f. Configure pointers to subdirectories (e.g., output)
    g. Import email recipients into a list
    h. Set up counters
    i. Set up list variables
    j. Execute primary tasks (e.g., key parallel function)
    k. Process results
    l. Calculate statistics
    m. Prepare email content
        1. Attachments
        2. Email notification message body
    n. Send email
    o. Script complete message
6. Script start (executes the main function)

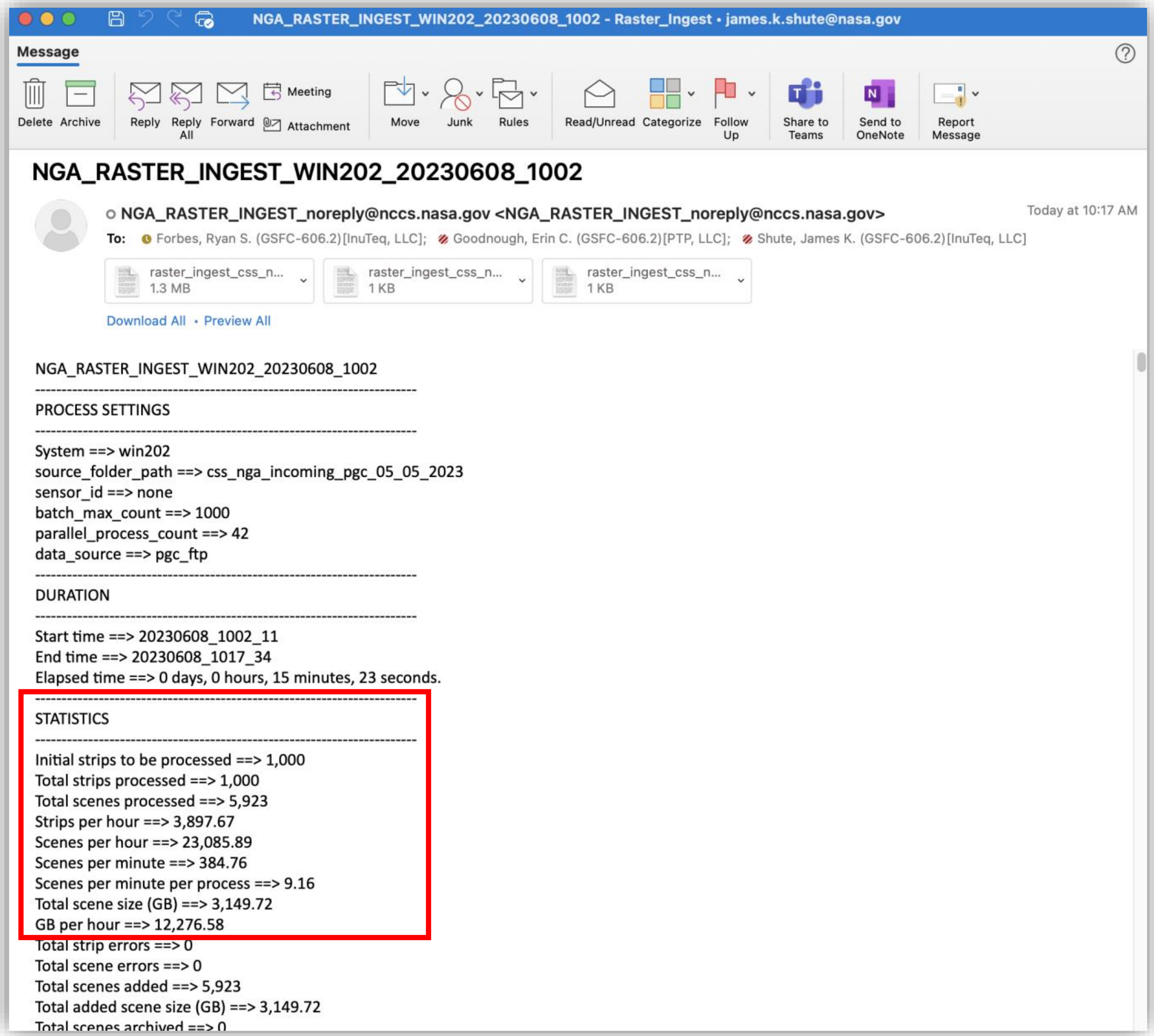
```
# Spatial Join and Calculate Attribute Function
# ---------------------------------------------
def spatial_join_and_calculate_attribute(input_feature_class, join_feature_class, spatial_match_option,
                                         join_attribute_field, source_calculate_field, target_attribute_field,
                                         search_radius):...


# Split Path Function
# -------------------
def split_path(path):...


# Process Raster Strip Function
# -----------------------------
def process_raster_strip(strip_path, linux_path, windows_path, master_feature_class, master_feature_class_full_path,
                         archive_feature_class, archive_feature_class_full_path, template_feature_class_full_path,
                         data_zone_feature_class_full_path, country_feature_class_full_path, operating_system,
                         data_source, database_host, database_port, database_name, database_user, database_password,
                         horizontal_distance_threshold):...


# Process Result Function
# -----------------------
def process_result(result):...


# Main body of the script
# ------------------------------------------------------------------------------
# Defines the entry point into the script
def main():

    # Capture script parameters
    # -------------------------
    root_folder = sys.argv[1]  # e.g., C:/att/project/arcshare/public/raster-ingest/ | /att/arcshare/raster-ingest/
    server_environment = sys.argv[2]  # e.g., alpha | dev-nccs | prod-nccs | dev-drg | prod-drg
    linux_path = sys.argv[3]  # e.g., /css/nga/
    windows_path = sys.argv[4]  # e.g., P:/data/
    master_feature_class = sys.argv[5]  # e.g., nga_footprint_master_v2
    archive_feature_class = sys.argv[6]  # e.g., nga_footprint_archive_v2
```

Demo


Apache Guacamole
guac.nccs.nasa.gov/guac/#/
RECENT CONNECTIONS
jshute
win202
winadm202
win201
win207
win210
win206
ALL CONNECTIONS
Filter
win201
win202
win203
win204
win205
win206
win207
win208
win209
win210
winadm201
winadm202
Currently in use by 2 users.

Parallel Process Applicability and Example Results

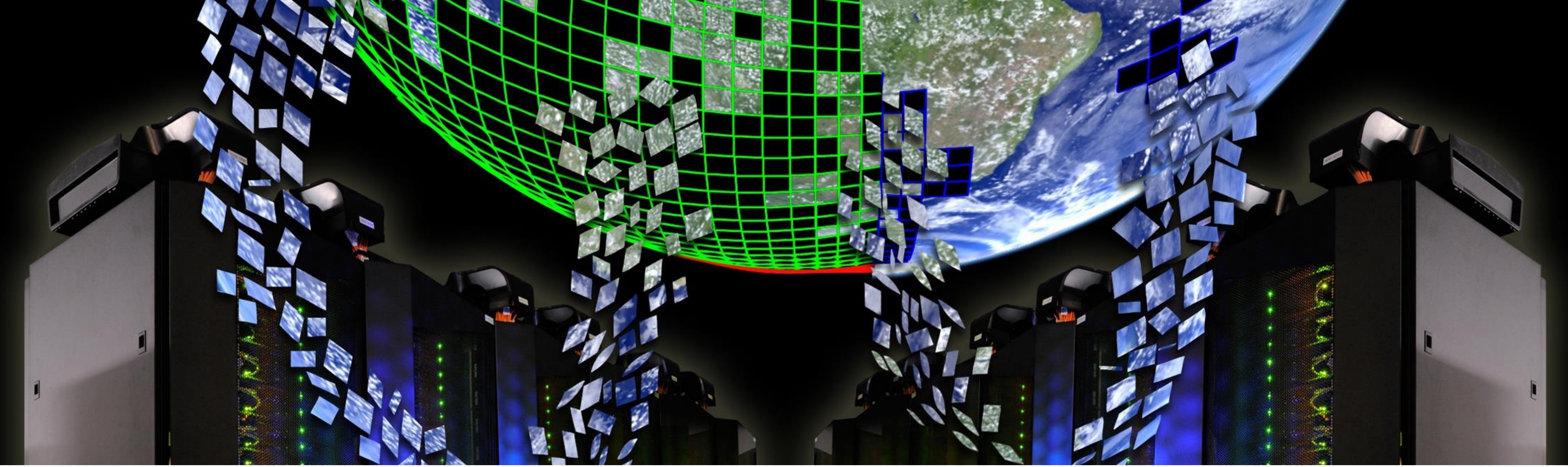
- Ingest rates are now 25k rows/hour/VM (as many as 10 VMs at the same time)
- Other processes utilizing this approach
  - Deleting rows or files
  - Duplicate record removal
  - Exporting rows from development to production (500-600k rows/hour)
  - Master Footprint (MFP) evaluation
  - Raster validation (100-200k rows/hour)



Multiprocessing Best Practices

- Using an IDE (e.g., PyCharm) was instrumental in helping with the debugging process
- Build repeatable tasks into their own functions and leverage across the environment (e.g., email_notification function)
- Python multiprocessing pool.apply_async was the easiest approach to get working
  - Requires a callback function
- Use script parameters to segment the workload (e.g., sensor_id)
- Write the script to work in serial or parallel mode
  - Debugging does not work inside parallel functions/processes
- Issues may arise when attempting to run too many processes in parallel (e.g., bad/incorrect results or counts)
- Build in logging, statistics calculations, and email notifications
- Functions called from external/other files, global counts, and lists cannot be accessed by a parallel process
- A complex object (e.g., database connection) cannot be sent to a parallel function.

```python
if process_method == "serial":
    # Use this function to test one record at a time
    # ---------------------------------------------
    strip_result = process_raster_strip(strip_full_path, linux_path, windows_path,
                                        master_feature_class, master_feature_class_full_path,
                                        archive_feature_class, archive_feature_class_full_path,
                                        template_feature_class_full_path, data_zone_feature_class_full_path,
                                        country_feature_class_full_path, operating_system, data_source,
                                        database_host, database_port, database_name, database_user,
                                        database_password, horizontal_distance_threshold)

    # Append the strip_result from the individual strip to the result list
    # ----------------------------------------------------------------
    if strip_result:
        logger.info("Append the strip_result from the individual strip to the result list")
        result_list.append(strip_result)

else:
    # Use this function when parallel processing
    # ----------------------------------------
    pool.apply_async(process_raster_strip, args=(strip_full_path, linux_path, windows_path,
                                                 master_feature_class, master_feature_class_full_path,
                                                 archive_feature_class, archive_feature_class_full_path,
                                                 template_feature_class_full_path,
                                                 data_zone_feature_class_full_path,
                                                 country_feature_class_full_path, operating_system,
                                                 data_source, database_host, database_port, database_name,
                                                 database_user, database_password,
                                                 horizontal_distance_threshold), callback=process_result)
```



Questions?

Contact Information	
NCCS Cloud Computing and Data Sciences (CCDS) Lead	Jim Shute (james.k.shute@nasa.gov)
NCCS GIS Manager	Ryan Forbes (ryan.s.forbes@nasa.gov)
NGA Data Manager	Erin Goodnough (erin.c.goodnough@nasa.gov)
NCCS Website	https://www.nccs.nasa.gov
NCCS Spatial Analytics Platform	https://maps.nccs.nasa.gov