Leveraging CSPP: Building a cloud based direct broadcast processing system

Thad Chee (SSAI), Louis Nguyen (NASA), Andrei Vakhnin (SSAI), A. Jason Barnett (BAH)

Leveraging CSPP: Building a cloud based direct broadcast processing system

Why build a cloud-based system for direct broadcast reception and processing?

- Up front costs are lower than building your own antenna network
- Reception of Direct Broadcast signals as needed

Ground Station Network Providers

- Leaf Space
- Kongsberg Satellite Services
- The Swedish Space Corp (SSC)
- CONTEC
- Sfera Technologies
- ATLAS Space Operations
- BridgeComm, Inc.
- Libre Space Foundation*

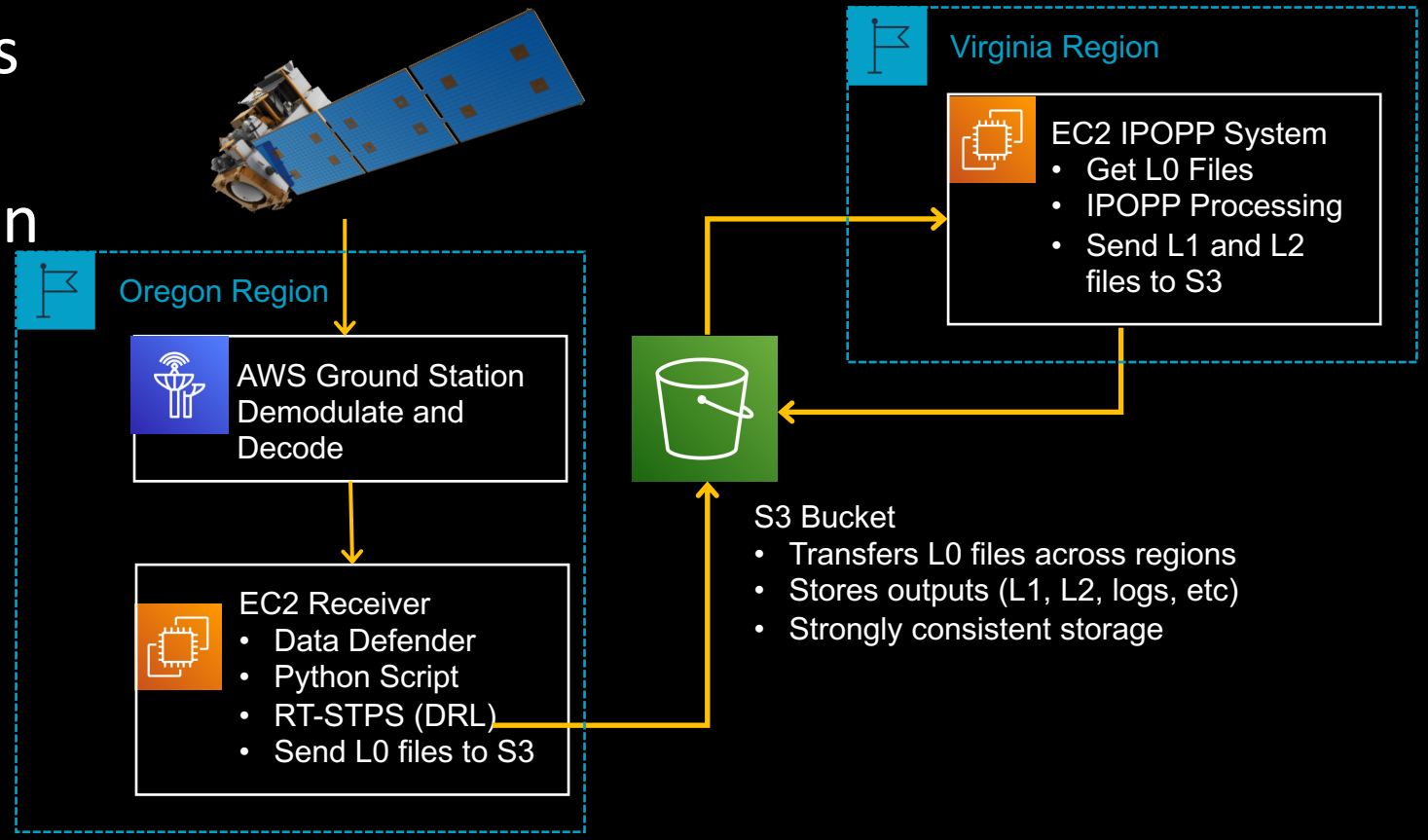
Ground Station Capacity Aggregators

- StellarStation
- Global Ground Station Network
- AWS Ground Station
- Microsoft Azure Orbital

** Open source*

How does AWS GS work? The first implementation:

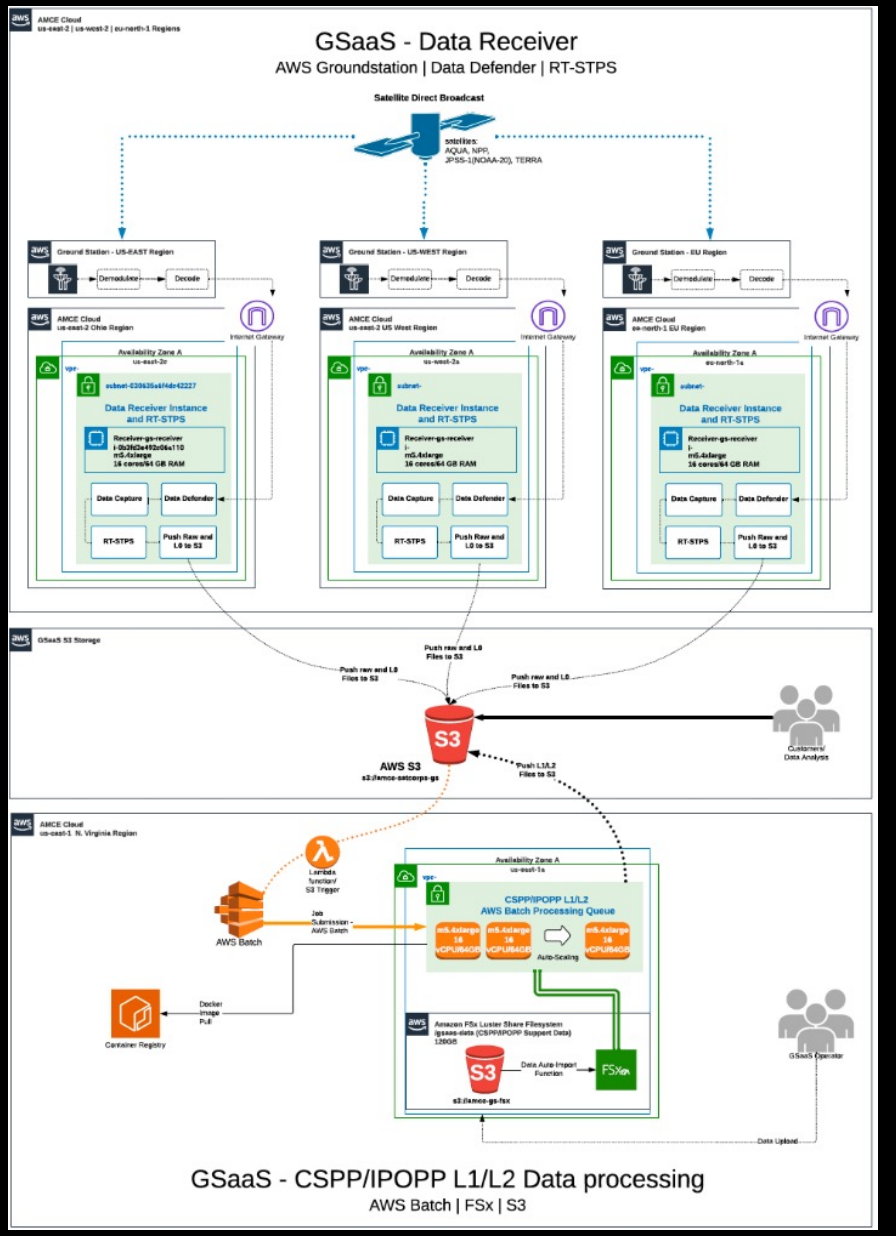
- The version 1 implementation of a direct broadcast system worked well with some scaling issues discovered later.
  - Managing the receiver ends
  - IPOPP's operation is for a different model of operation
- Note the use of the AWS S3 bucket, it's pivotal from a regional standpoint



Leveraging CSPP: Building a cloud based direct broadcast processing system

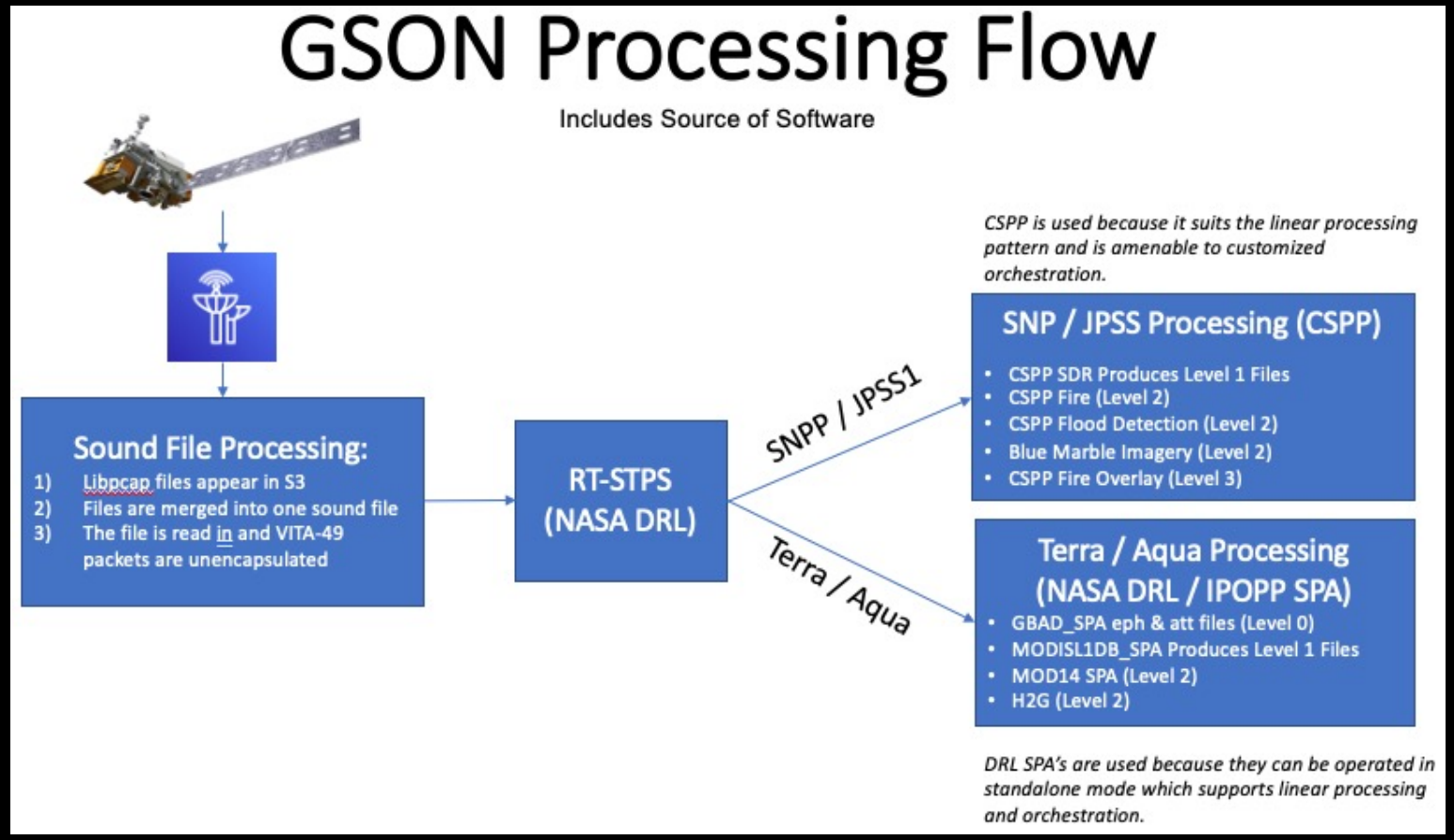
A management problem:

- This is an example using the version 1 set up if you are using only three ground stations.
- Each ground station operates its own receiver instance with a data defender, python scripts and RT-STPS installed in each geographical location.



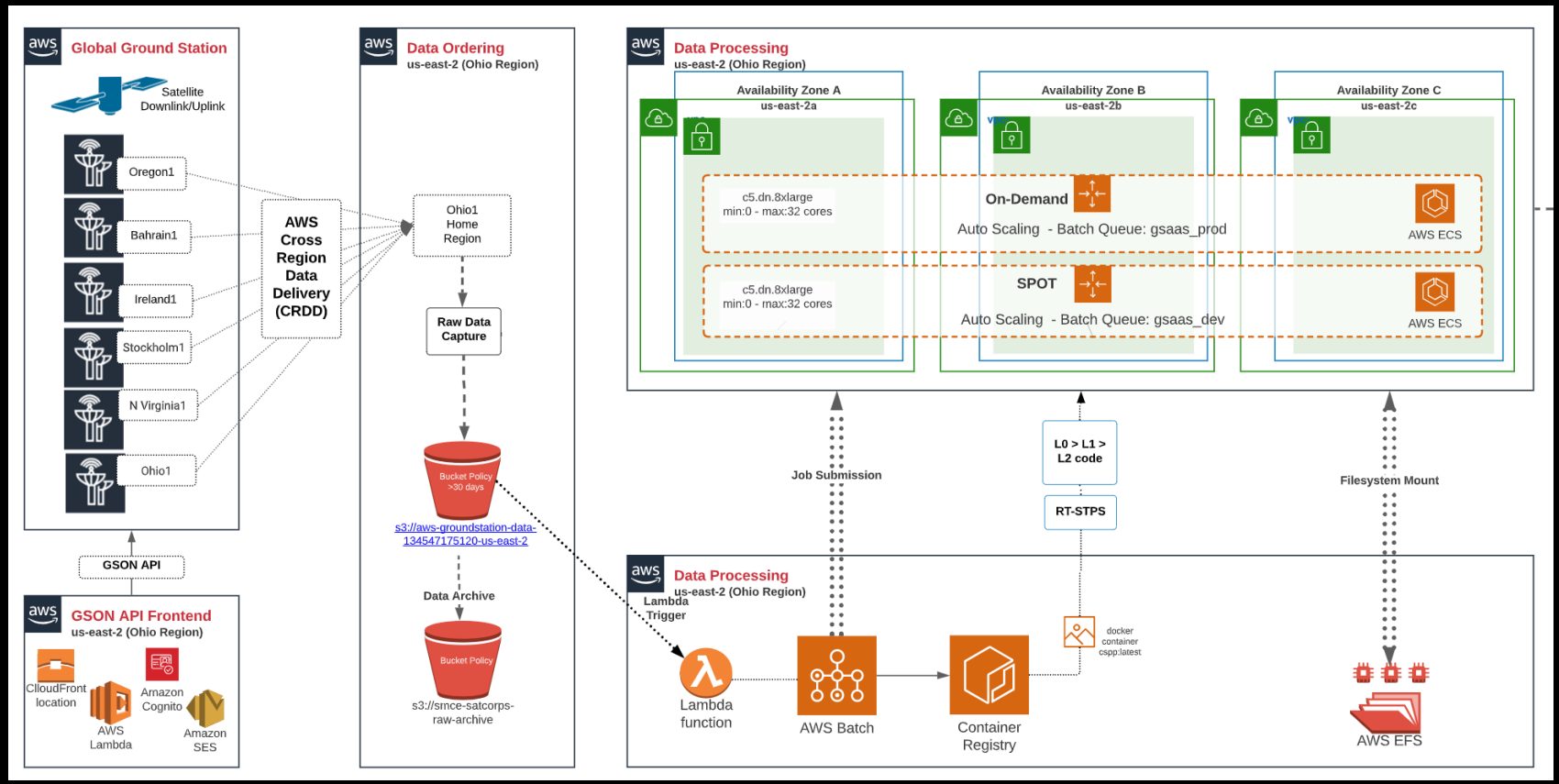
Leveraging CSPP:

- Initially started with a bespoke linear script
- We realized that after running CSPP SDR, that there was no need to run any of the other packages serially



Enter CRDD:

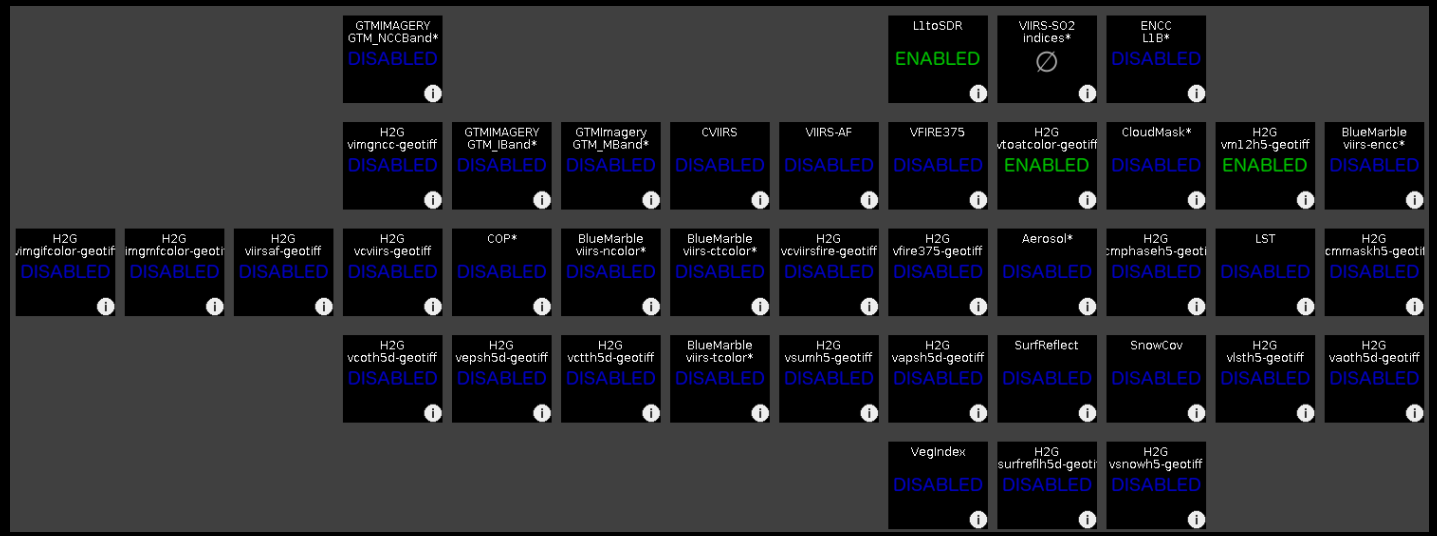
- Cross Region Data Delivery (CRDD) – an architectural change presented a new opportunity for a new processing model
- No more receivers!
  - management headaches
  - No more “Must work” pieces
  - Easier to debug and develop



Leveraging CSPP: Building a cloud based direct broadcast processing system

Leveraging CSPP:

- With the realization that we could mix and match different CSPP software, we also realized they could be run in parallel as long as prerequisites are met (L1 exists etc)



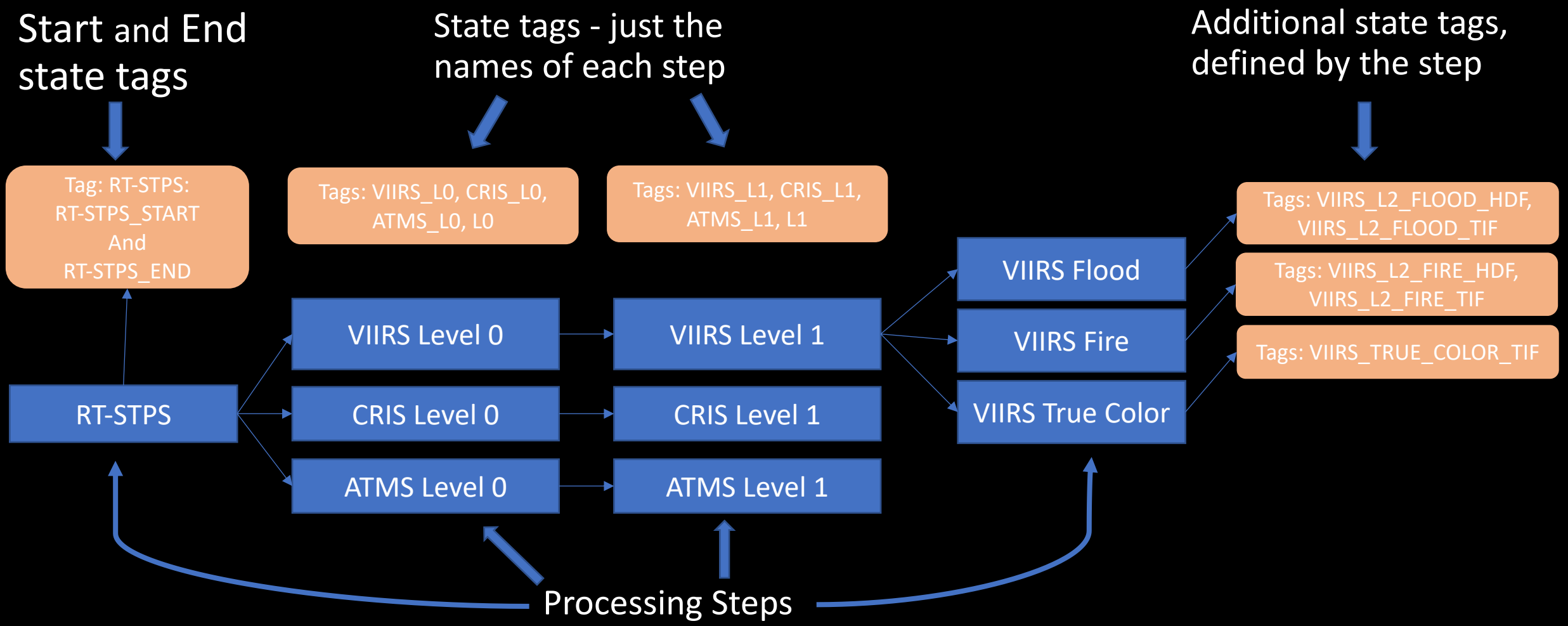
IPOPP Status Screen over VNC

- We also realized that the bespoke linear design was repeating the same steps over and over, just with different CSPP commands and that the CSPP software all had the same general requirements.
- After reviewing our existing code and identifying these patterns, we developed an abstraction that captures how we use CSPP as JSON objects.
- This “step” model encapsulates each work task and its requirements and insulates the system from the details. This is very similar to the IPOPP processing tree.

Leveraging CSPP: Building a cloud based direct broadcast processing system

- The idea of “steps”
  - Each step is given a “name”
  - A step is a unit of work of any size:
    - RT-STPS
    - CSPP SDR
    - Fire / Flood
    - Copying results to S3
  - How to coordinate the steps in sequence and in parallel – state tags
- The engine keeps a “bag” of state tags
  - Start state tag – the step has started
  - End state tag– the step has completed
  - End_OK state tag – the step has completed successfully
  - End_Failed state tag – the step has completed but in a failed state
- The engine is really a coordinator of the steps
  - Initializing, loading steps
  - Picking steps to run
  - Determining when there are no more steps that can run
    - No running steps
    - No steps that can run - there are normally steps that don’t run (error handling steps)

GSON System – The State Driven Engine



Benefits of a state driven engine:

- Parallelism: Speed
  - Notifications happen immediately, L0, L1, L2 as soon as files are available on s3
  - All instruments and Levels are processed simultaneously
- Modularity: Flexibility
  - Processing steps can be easily swapped in or out by modifying the JSON meta-data
  - Ease of debugging or prototyping – Changing JSON meta-data to linear script
- Complexity: With error handling, the linear execution script was very large with multiple scripts for each satellite or sensor. The state engine reduced the code complexity by 70%
- Simple debugging: Change the start tag allows one to test specific parts of the processing individually
- Allows us to run CSPP science algorithm on multiple satellites/instruments by only modifying meta-data, not code. You can run serial or parallel depending on your situation.

Going Forward:

- Additional input sources: Azure and other Direct Broadcast Receivers
- Increases in capability:
  - More options for user steps
  - Adding capability to the step processing
- Increases in speed:
  - Starting earlier
  - Processing the CRDD granules as they arrive
  - Parallelizing: Processing 1 granule per CSPP run instead of all
    - Minimal benefits anticipated from Fire algorithm – it is already internally parallelized
    - Expect to see flood algorithm speed gains but need to merge netcdf's afterwards (currently about 20 minutes to run)
- User defined monitoring in addition to MQTT

Thank you!

Questions?

http://52.200.226.137/cgi-bin/site/showdoc?mnemonic=AWS-NOTIFICATION-ENDPOINT&c=gsaas-pickup