The Self-describing Portable Data Container

Maohai Huang, NAOC

September 15-18, 2019, Nanning

spdc?

- SPDC is a ``container'' package written in Python for packing different types of data together,
- letting the container take care of inter-platform compatibility, serialization, persistence, and data object referencing that enables lazy-loading.
- The word ``container'' in the name is more similar to that in ``a shipping container'' (emphasizing association) instead of ``a Docker container'' (emphasizing isolation).

SPDC packages

- The base data model is defined in package `dataset`.
- Persistent data access, referencing, and Universal Resource Names are defined in package `pal`.
- A reference REST API server designed to communicate with a data processing docker using the data model is in package `pns`.

- All classes are individually versioned.

CSC Standard Product Generation (SPG)

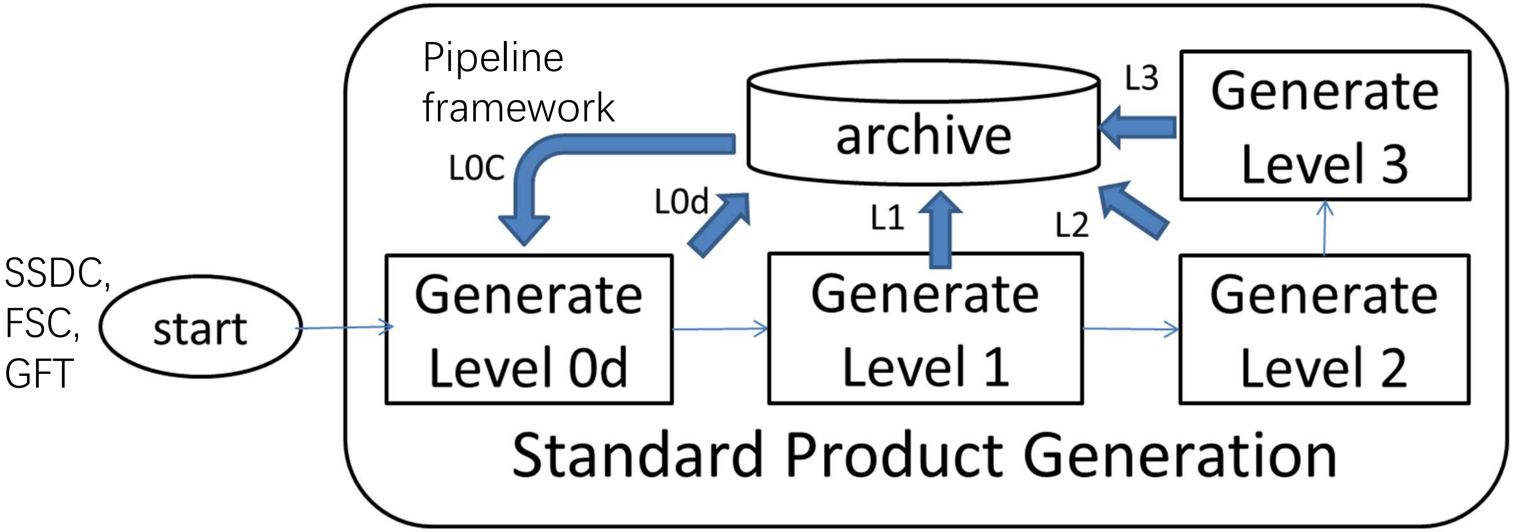
- Specifically 'Data Products' means deliverable legacy data of the mission with controlled quality.
- CSC produces L0d – L2.5 Data Products are generated from L0c data, calibration data, and external auxiliary data. Short Description of "L" levels:
    1. L0d: telemetry reorganized for L1 generation.
    2. L1: uncalibrated instrument data. Organized according to astronomical convention.
    3. L2: calibrated instrument data with physical units that can be used by the general community.
        - Calibration models and algorithms are **published**, **standardized**, and **configuration controlled**.
    4. L2.5: conceptually simple merging of L2 data to: e.g. **stitched maps**, **connected spectra** and **SEDs**, **light curves**.
        - May be derived from multiple instruments.
        - Calibration models and algorithms are **published**, **standardized**, and **configuration controlled**
    - "L" levels are not always the same as "SP" levels.
- L3: Calculation is not standard model-dependent and/or **external data sets**.
- SPG controls input, processing, validating, output, pipeline delivery time. Has production timing requirements
    - L3 is not in SPG. Best-effort-basis for resource spent on validation, production timing

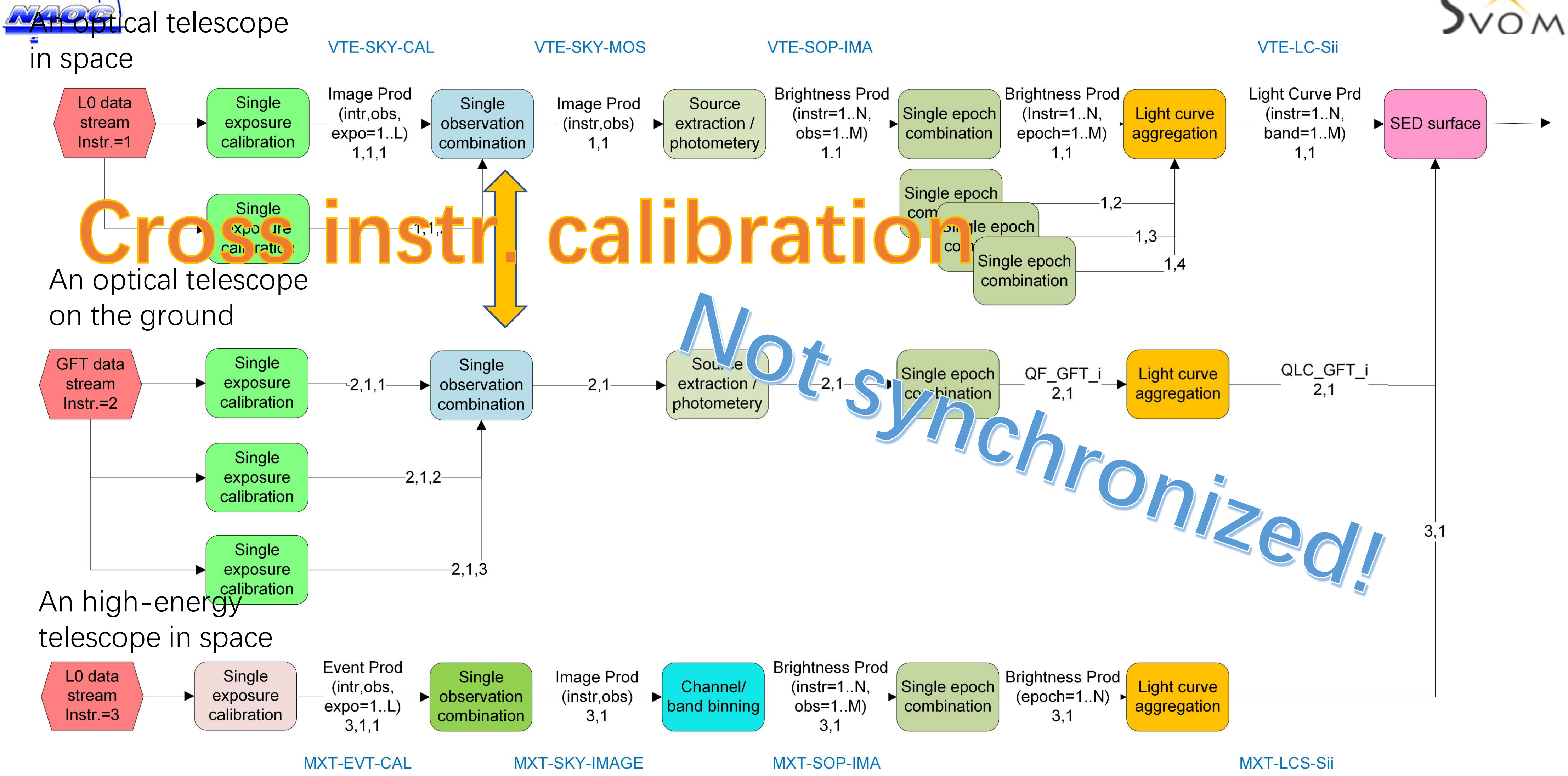
Types of Product Generation

L0c

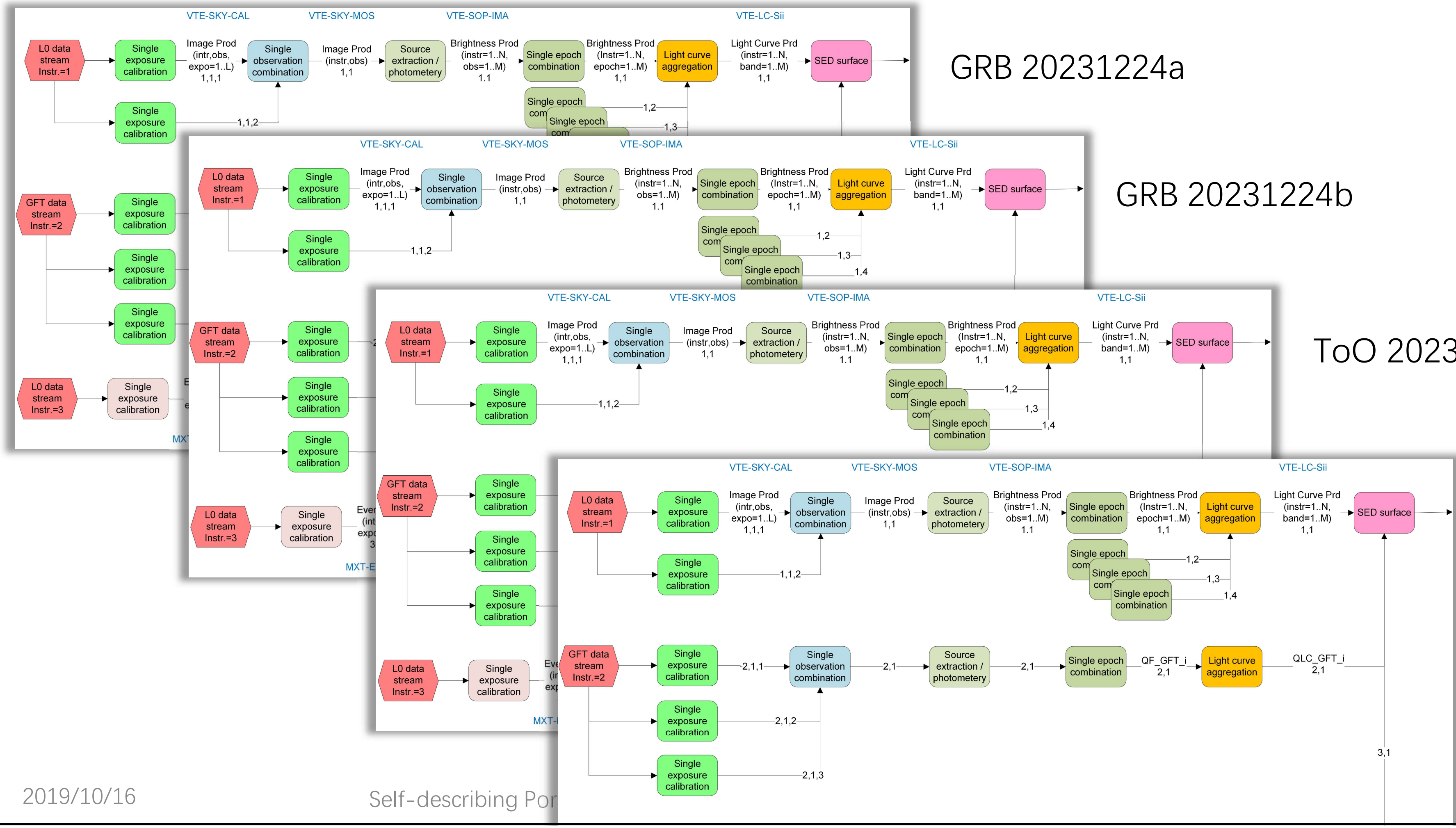
1. **Automatic SPG** started by the arrival of new data from VHF, S-band, X-band (L0c), and ground instruments.
    1. Runs only once.
    2. Uses the latest production release pipeline and calibration products.
2. **Updating SPG** started by hand or scheduler.
    1. Runs after every new production release of pipeline and calibration products on all data as needed.
    2. Uses the latest production release pipeline and calibration products.
3. **Customized Product Generation**: a possible subset of all pipelines are run with customized configuration by a creator.
4. In Automatic and Updating SPG, pipelines are run by a central scheduler. sequentially to traverse a SPG dependency Tree. (in parallel TBD)
5. In all cases the starter should provide with a ‘Creator’ and a ‘RootCause’ parameters to the piplelines, which will produce Products that remember these parameters.

A naiive view of the pipelines




An optical telescope in space
VTE-SKY-CAL
VTE-SKY-MOS
VTE-SOP-IMA
VTE-LC-Sii
L0 data stream Instr.=1
Single exposure calibration
Image Prod (intr,obs, expo=1..L) 1,1,1
Single observation combination
Image Prod (instr,obs) 1,1
Source extraction / photometery
Brightness Prod (instr=1..N, obs=1..M) 1.1
Single epoch combination
Brightness Prod (Instr=1..N, epoch=1..M) 1,1
Light curve aggregation
Light Curve Prd (instr=1..N, band=1..M) 1,1
SED surface
1,2
1,3
1,4
Cross instr. calibration
An optical telescope on the ground
Not synchronized!
GFT data stream Instr.=2
2,1,1
2,1
QF_GFT_i 2,1
QLC_GFT_i 2,1
2,1,2
2,1,3
3,1
An high-energy telescope in space
L0 data stream Instr.=3
Event Prod (intr,obs, expo=1..L) 3,1,1
Image Prod (instr,obs) 3,1
Channel/ band binning
Brightness Prod (instr=1..N, obs=1..M) 3,1
Brightness Prod (epoch=1..N) 3,1
MXT-EVT-CAL
MXT-SKY-IMAGE
MXT-SOP-IMA
MXT-LCS-Sii

And there are can multiple objects to be observed simultaneously…



GRB 20231224a

GRB 20231224b

ToO 20231225a

GW 20231225

Data processing pipeline requirements summary

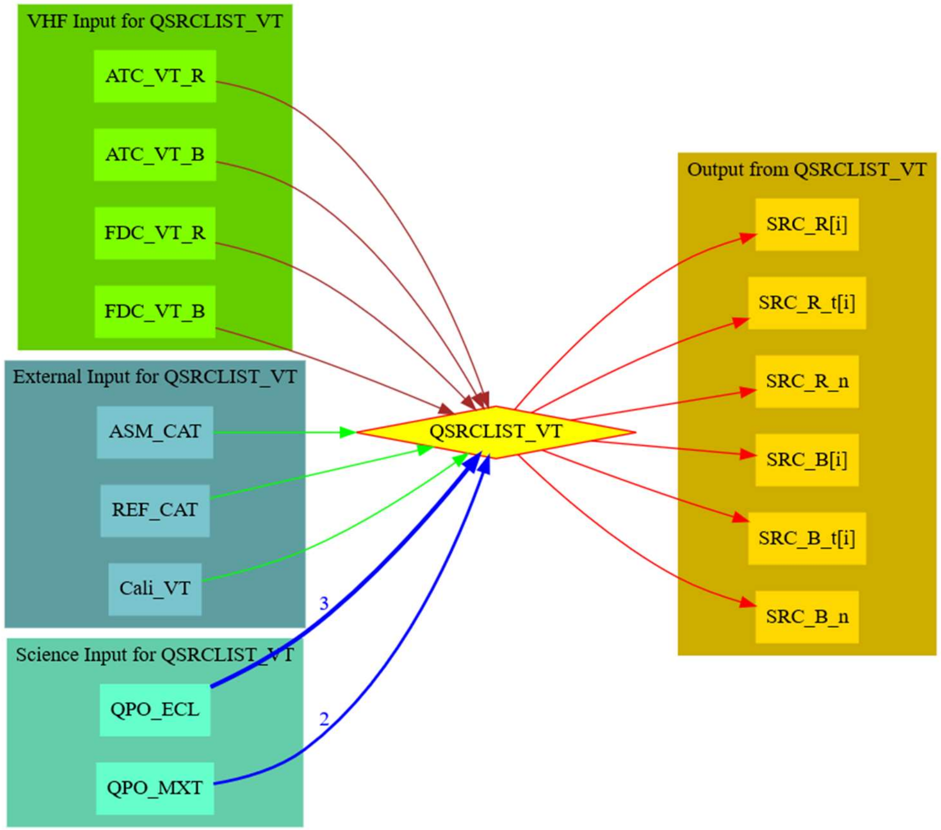
- **A pipeline**
  - the pipeline software (with their configuration) should be versioned.
  - The pipeline is driven asynchronously or synchronously.
  - Be modularized, made of “nodes” that generate products.
  - Deployment should allow robust scaling
- **A Data Product**
  - Should be versioned
  - results are to be saved in a persistent store.
  - A Product needs a URI.
  - Has uniform format or APIs.
- **Node**:
  - The function of a node may need to be used at the same time, asynchronously.
  - With one interface for I/O and control
  - Accommodates domain experts’ need to use their favorite tools (hardware, OS, software, libs…) to make the Processing Task Software.
  - Easy and quick access for developers.

Example:

Use SPDC for developing a simplified VVPP node

VVPP input / output and deployment

- Input
  - 4 VHF datasets
  - 3 external datasets
  - 2 science datasets
- Output
  - 6 datasets
- Being developed at NAOC
- to be integrated as a processing node in the FSC VHF pipeline.
- To communicate with the FSC pipeline with LAN.
- To be deployed in a cluster environment managed by FSC.
- http://svom.iap.fr/fiches/view_map.php?list_id[]=217



INPUT

3	Cali_VT	Calibration database -VT	M	text	database	FSC, CSC

Input from VHF messages

Order	Field	Name	Status	Format	Expected values/units	Origin
1	ATC_VT_R	Attitude Chart - VT - R filter	M/O	text	x y mag	VHF message VT Attitude Chart R
2	ATC_VT_B	Attitude Chart - VT - B filter	O/M	text	x y mag	VHF message VT Attitude Chart B
3	FDC_VT_R	Finding Chart - VT - R filter	M/O	text	x y mag snr	VHF message VT finding chart R
4	FDC_VT_B	Finding Chart - VT - B filter	O/M	text	x y mag snr	VHF message VT finding chart B

Input science products

Order	Field	Required output fields	Status
1	QPO_ECL	RA	M
2	QPO_ECL	DEC	M
3	QPO_ECL	ERR	M
4	QPO_MXT	RA	M
5	QPO_MXT	DEC	M

data model of input products

Comments

-- Attitude Charts (ATC_VT_R,B) and Finding Charts (FDC_VT_R,B) are generated by Payload Data Processing Unit (PDPU) on board. -- ASM_CATs are the astrometric catalogues for VT astrometric calibrations, such as USNO, Gaia, UCAC, etc. -- REF_CATs are the deep all-sky survey catalogues, i.e. USNO, Sloan, Gaia, etc. -- Cali_VT is the VT calibration database. It is to be used to correct the vignetting (illumination uniformity) and do photometric calibrations. -- QPO_MXT (Quick source position of MXT detection) should include the detection errors, which determine accurately cross-matching radius in VT Finding Charts

Draft

Check name, status and origin for VHF inputFDaigne: Please check if QPO_ECLAIRs is really needed.FDaigne: Please check if QPO_MXT is really needed: is the on-board processing not already limiting sources in the finding chart based on MXT position?

OUTPUT

Output science products

Order	Field	Name	Status	Format	Expected values/units
1	SRC_R[i]	List of bright source positions with mag. (R filter).	M/O1	m col.; n rows; float	Coord. in J2000 (RA, Dec). Mag. in AB
2	SRC_R_t[i]	Observation time of the list n (R filter)	M/O2	m col.; n rows; float	Coord. in J2000 (RA, Dec). Mag. in AB
3	SRC_R_n	Number of available source lists (R filter)	M	integer	1 to 6
4	SRC_B[i]	List of bright source positions with mag. (B filter).	O/M1	m col.; n rows; float	Coord. in J2000 (RA, Dec). Mag. in AB
5	SRC_B_t[i]	Observation time of the list n (B filter)	M/O2	m col.; n rows; float	Coord. in J2000 (RA, Dec). Mag. in AB
6	SRC_B_n	Number of available source lists (B filter)	M	integer	1 to 6

Draft

SRC_R,B: number of lines : $n \leq 200$.
SRC_R,B: number of columns and content of columns
check range for SRC_R_n and SRC_B_n

Define a data model for this example:

- Input product:
  - dataset
    - ATC_R, ATC_B as (type, unit):
      - [(float, None), (float, None), (float, mag)]
  - Metadata(?)
    - Pointing as (type, unit):
      - (float, None), (float, None)
    - This is not in the table. Is it a product or a paremeter?

```
00001_R_1_00_atti.cat
1473.710    1467.940     9.020
 538.960    1214.460    12.230
1921.380     919.370    12.290
 801.550    1919.200    12.310
 986.500     741.020    12.350
1350.460     661.700    12.360
 715.140     920.660    12.390
 671.520    1499.250    12.510
 465.150    1898.750    12.550
…
…
00001_R_1_00_atti.pn
142.93425555 42.6292460757
```

Define the ATC product SPDC

```python
from dataset.dataset import TableDataset
from pal.context import MapContext

import logging
import logging.config
# create logger
logger = logging.getLogger(__name__)

#logger.debug('level %d' % (logger.getEffectiveLevel()))


class Chart(MapContext):
    """ Chart has a TableDataset named 'table' as its data    """

    def __init__(self, **kwds):
        # must be the first line to initiate meta and get description
        super().__init__(**kwds)
        # implemented with TableDataset for easier row operation
        self['table'] = TableDataset()


class ChartXY(Chart):
    """ ATC is a Chart that has 'x', 'y', 'xerr', 'yerr' columns
    for coordinates and 'm', 'merr' for measurement. they are accessed with f
oo.x and foo.y etc.
    """

    def __init__(self, **kwds):
        # must be the first line to initiate meta and get description
        super().__init__(**kwds)
        self['table'].data = [
            ('x', [], ''),
            ('y', [], ''),
            ('m', [], ''),
            ('xerr', [], ''),
            ('yerr', [], ''),
            ('merr', [], '')
        ]
```

```python
class ATC_VT_R(ChartXY, Monochrome):
    """
    """

    def __init__(self, **kwds):
        # must be the first line to initiate meta and get description
        super().__init__(**kwds)
        self.band = 'R'


class ATC_VT_B(ChartXY, Monochrome):
    """
    """

    def __init__(self, **kwds):
        # must be the first line to initiate meta and get description
        super().__init__(**kwds)
        self.band = 'B'
```

- Taken from `svom/products/chart.py`
- This python class is hand-written. It is planned to generate Python product classes automatically from data model XML files.

Read the ascii file and make an ATC product

```python
def makeProdfrom(self, filep):
    """ Read attitute chart csv file and make products that a
osed to be the output of the upstream VHF module.
    """
    #logger.debug('reading csv file ' + str(filep))
    d = self.loadCSV(filep)
    now = FineTime1(datetime.datetime.now(tz=datetime.timezon
    atc = ATC_VT_R(description=filep.name,
                   creator=__name__,
                   creationDate=now,
                   instrument='VT',
                   startDate=now,
                   endDate=now,
                   rootCause='VVPP Simulation',
                   modelName='prototest',
                   type='ATC_VT_R')
    # atc has no error ?
    d.append([0] * len(d[0]))
    d.append([0] * len(d[0]))
    d.append([0] * len(d[0]))
    atc['table'].data = d

    filea = filep.with_suffix('.pn')
    d = self.loadCSV(filea)
    atc.meta['pointing'] = Parameter([d[0][0], d[1][0]])

    # assemble
    logger.debug(atc.toString())
    return atc
```

- From `svom/vvpp.engsimulator.py`

```
# description = "00001_R_1_00_atti.cat"
# meta = MetaData{[description = 00001_R_1_00_atti.cat, crea
tor = vvpp.engsimulator, creationDate = 2019-10-15T17:53:34.
761498 TAI(624477214761498), instrument = VT, startDate = 20
19-10-15T17:53:34.761498 TAI(624477214761498), endDate = 201
9-10-15T17:53:34.761498 TAI(624477214761498), rootCause = VV
PP Simulation, modelName = prototest, type = ATC_VT_R, missi
on = SVOM, pointing = Parameter{ description = "UNKNOWN", va
lue = [142.93425555, 42.6292460757], type = }, ], listeners
= [ATC_VT_R 7696557388016 "00001_R_1_00_atti.cat", ]}
# History
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =

# data =


# [ refs ]
# MapRefsDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =


# [ table ]
# TableDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =

# x y m xerr yerr merr
#
1473.71 1467.94 9.02 0 0 0
538.96 1214.46 12.23 0 0 0
1921.38 919.37 12.29 0 0 0
801.55 1919.2 12.31 0 0 0
986.5 741.02 12.35 0 0 0
```

SPDC Properties

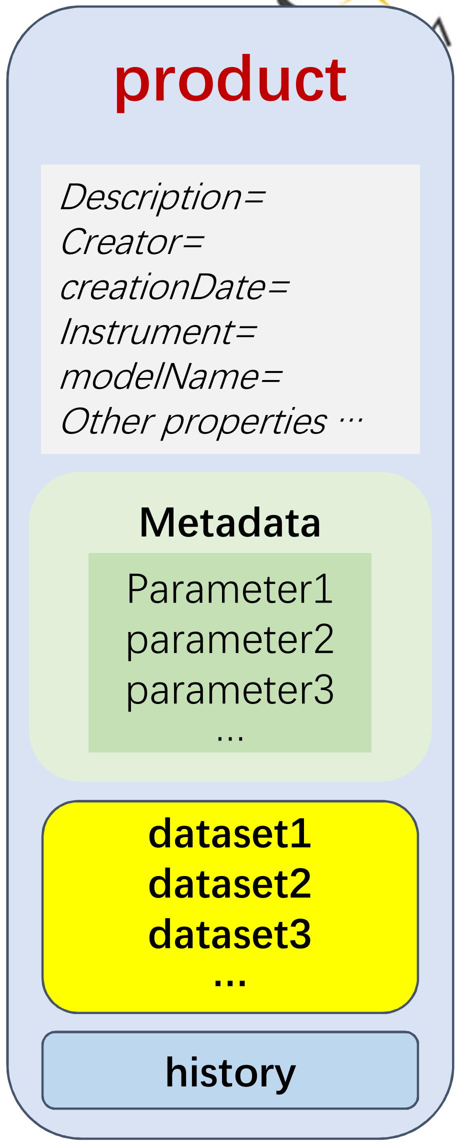
- **Annotatable**: one can use textual description to annotate the contained data;
  - `myData.description = 'My Precious'`
- **Attributable**: one can add attributes (or called properties, meta data) to the contained data;
  - `myData['creator'] = 'The Dark Lord Sauron'`
- **Copyable**: one can ask for a copy of the data;
  - `anotherOne = myData.copy()`
- **Comparable**; one can compare two containers to see if they are equal;
  - `anotherOne == myData`

- **Queryable**: Can be queried to discover its contents, and obtain references of the components
- **Serializable**: one can transmit the data across the network and re-construct (deserialize) them on the receiving side
- Accepts **Change Listeners**
- Easy to handle with **REST API**
- **Free to add properties** which are accessible by users.

SPDC allows one to organize data into:

- A **Product** -- an arbitrary combination of datasets with some mandatory meta data.
- A **dataset** -- an arbitrary combination of
  - N dimensional arrays with an optional unit,
  - Tables
  - An arbitrary combination of the above.
- **Metadata** – a list of named parameters
- A **Parameter** – a string or quantity.

- Besides using the above base class directly, one can also
  - associate groups, arrays, or tables of Products using basic data structures such as sets, sequences (Python `list`), mappings (Python `dict`),
  - Define custom-made classes that inherit functionalities from base classes provided by the package.
- SPDC accommodates highly complex associated and nested structures.



Product Access Layer (PAL)

- Provides classes for the **storing, retrieving, tagging,** and **context creating of data product** modeled in the dataset package.
- Lets one store data in logical ``pools'', and makes the data accessible with light weight product references. A ProductStorage interface is provided to handle saving/retrieving/querying data in registered ProductPools.
- In a data processing pipeline or network of processing nodes, data products are generated within a context. Data processers, data storages, and data consumers often need to have relevant context data recorded with a product. However the context could have a large size so including them as metadata of the product is often impractical.
- Once a data product is saved by ProductStorage it can **have a reference generated for the saved Product**. Through its reference the product can be accessed. The size of such references are typically less than a hundred bytes, like a URL. **References enable SPDCs to encapsulate rich, deep, sophisticated, and accessible contextual data, yet remain light weight.**

PAL example

Create a product and a productStorage with a pool registered

```
# a pool for demonstration will be create here
demopoolpath = '/tmp/demopool'
demopool = 'file://' + demopoolpath
# clean possible data left from previous runs
os.system('rm -rf ' + demopoolpath)
# create a prooduct
x = Product(description='in store')
print(x)
{meta = "MetaData['description', 'creator', 'creationDate', 'instrument', 'startDate', 'endDate', 'rootCause', 'modelName', 'type', 'mission']", _sets = [], history = {meta = "MetaData[]", _sets = []}}
# create a product store
pstore = ProductStorage(pool=demopool)
```

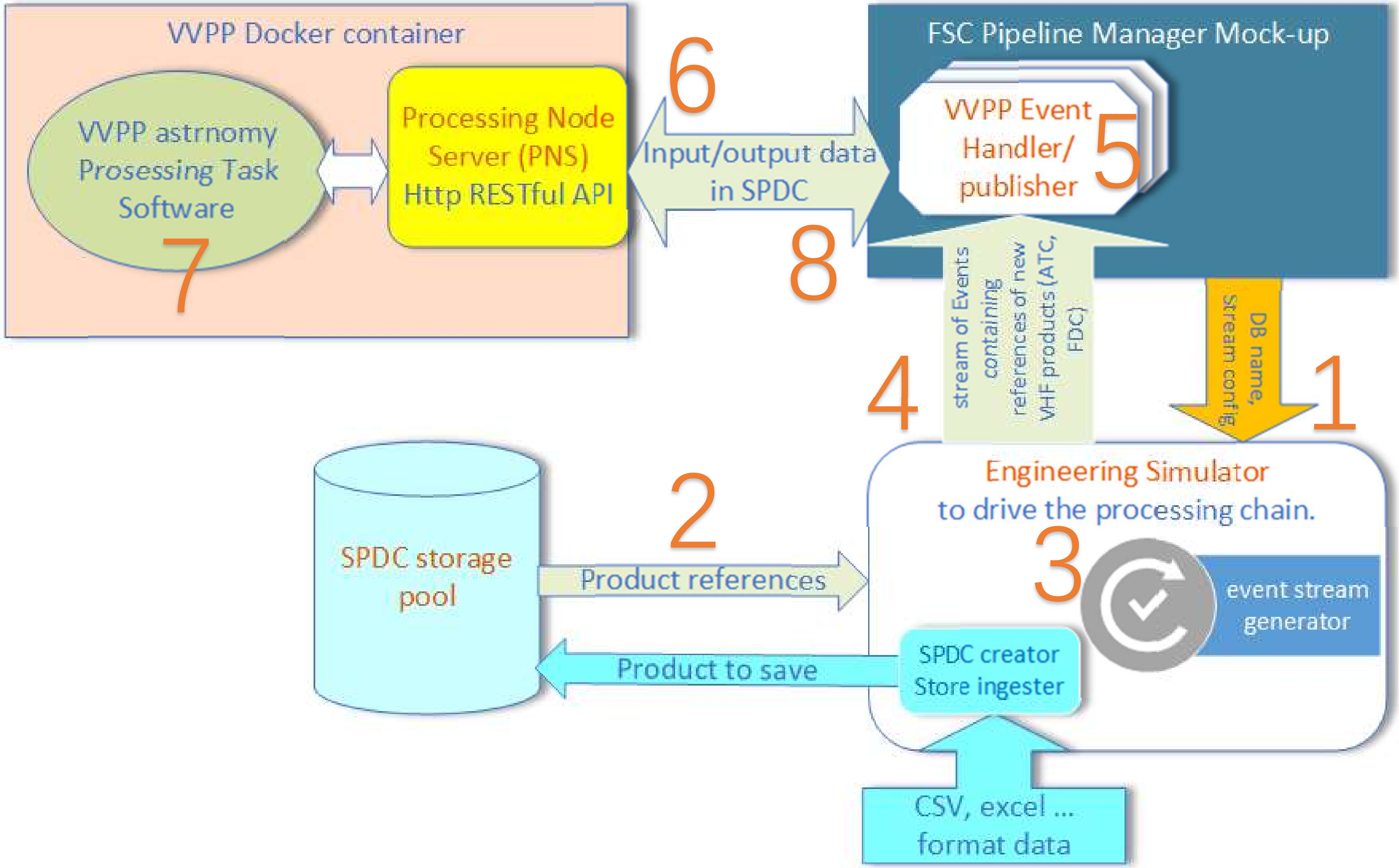
Save the product and get a reference

```
prodref = pstore.save(x)
# create an empty mapcontext
mc = MapContext()
# put the ref in the context.
mc['refs']['very-useful'] = prodref
# get the urn string
urn = prodref.urn
print(urn)
urn:file:///tmp/demopool:Product:0
```

re-create a product only using the urn

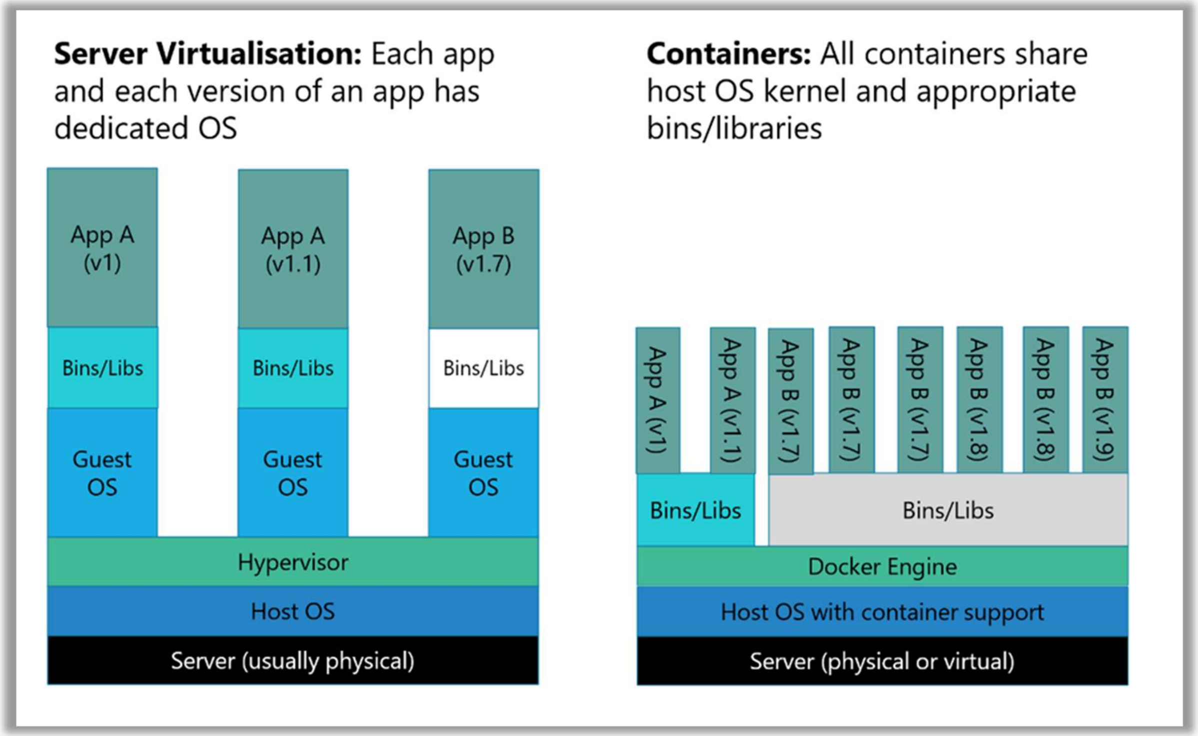
```
newp = getProductObject(urn)
# the new and the old one are equal
print(newp == x)
True
```

VVPP external environment setup



Docker Container for pipelines

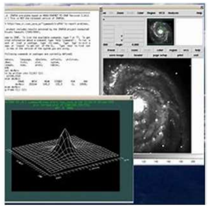
- Many data processing pipelines need to run software that only runs on a specific combination of OS type, version, language, and library.
- These software could be impractical to replace or modify but need to be run side-by-side with software of incompatible environments/formats to form an integral data processing pipeline, each software being a ``node'' to perform a processing task.
- Docker containers are often the perfect solution to run software with incompatible dependencies.



Microsoft

Develop and Deploy a Node for a pipeline

- Astronomers develop the Processing Task Software (**PTS**) in his/her favorite environment:
  - Concentrate in solving astronomy and algorithmic problems
  - Define input and output data models.
  - Prepare test data so that the PTS can be tested.



Xspec

- When the PTS is ready to deploy, with supporting tools and libraries, it is cloned to a in a Docker container.
  - All docker images are managed by a repository

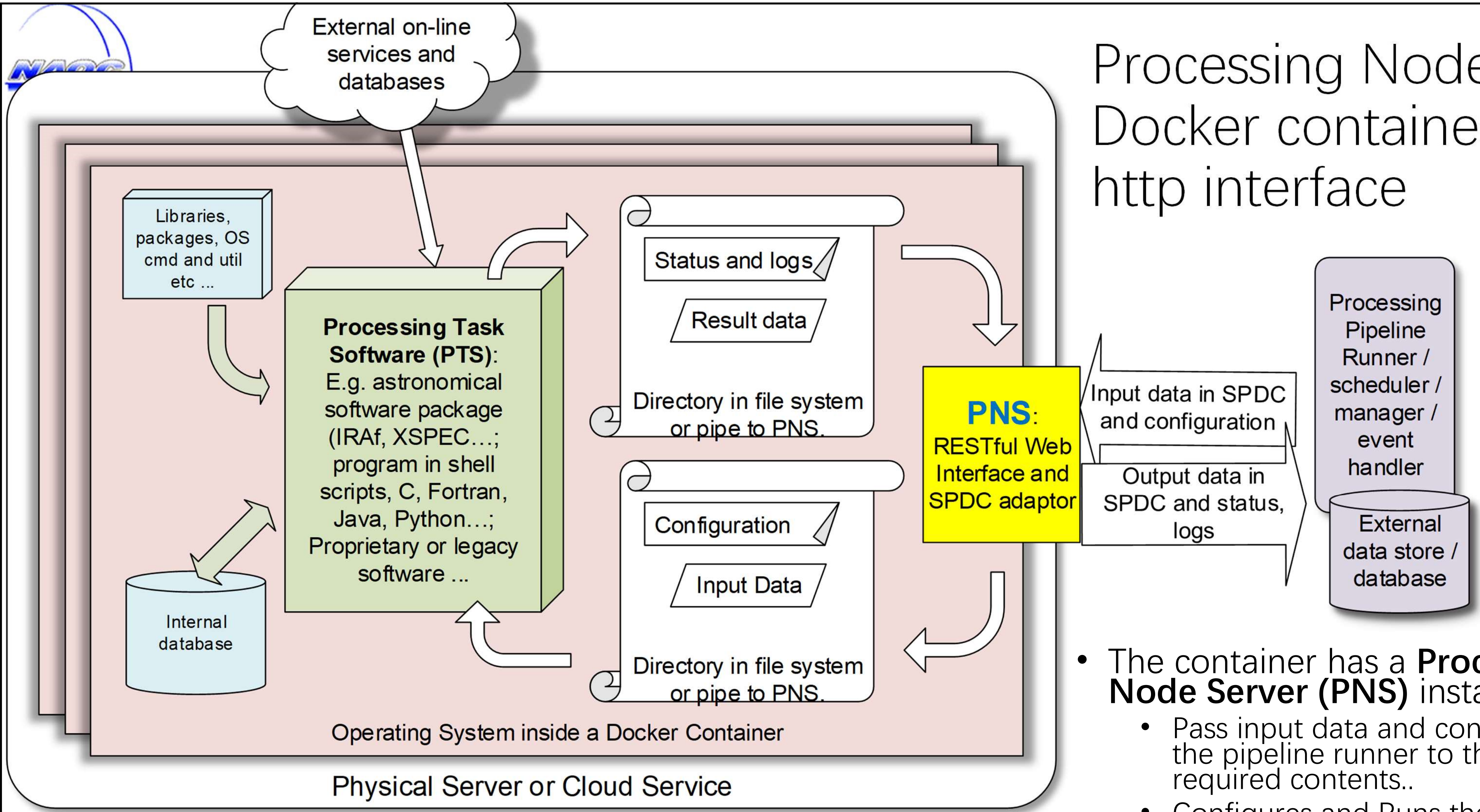
- A Processing Node Server is (PNS) installed in the container OS.
- PNS is a Web RESTful API Server for a data processing pipeline/network node that provides interfaces to
  - configure the data processing task software (PTS) in a processing node,
  - make a run request,
  - deliver necessary input data, and to read results.
- PNS uses a ‘Delivery Man’ Protocol to talk to PTS.

APACHE HTTP SERVER

Flask web development, one drop at a time

python

Processing Node in a Docker container with an http interface



- **PTS is stateless** -- the result can be calculated and reproduced with given input and configuration. PTS does not remember previous runs.

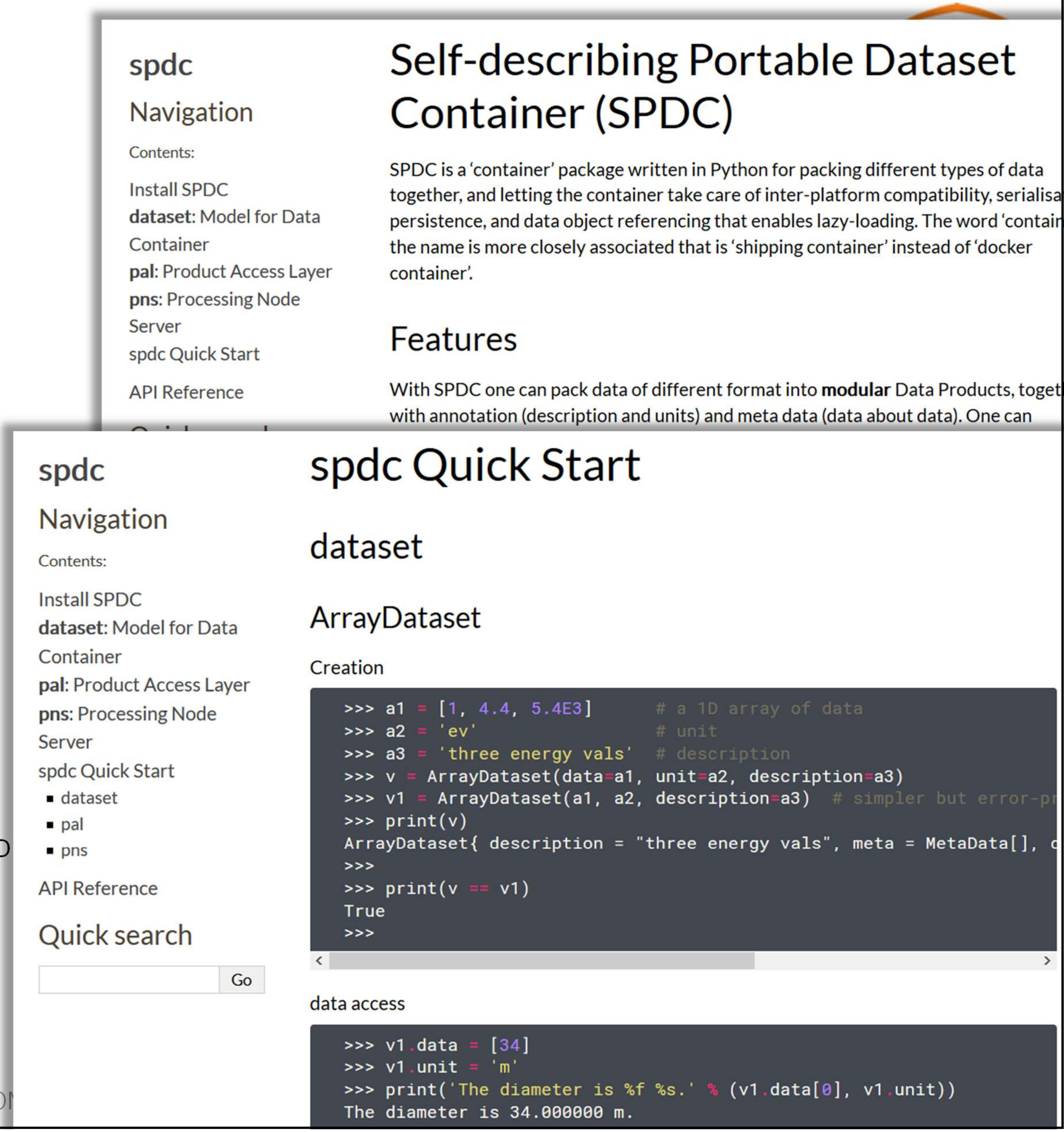
- The container has a **Processing Node Server (PNS)** installed to
  - Pass input data and configuration from the pipeline runner to the PTS with required contents..
  - Configures and Runs the PTS and produce results (successful or not)
  - Collect the results and return them to the pipeline runner.

PNS is Working

- PNS installed on a Docker container or a normal server allows a processing tasks to
  - run in the PNS memory space, in a daemon process, or as an OS process
  - receiving input and delivering output through a ``delivery man'' protocol.
- SPDC v0.8 test suite has been run on CentOS, Ubuntu, and Cygwin with Apache and Flask servers.
- The client-server pipeline architecture is shown to work with VVPP proof-of-concept server running processing software written in C, FORTRAN, Python using Pyraf and Anaconda.

Repo and docs

- Gitlab repo
  - http://mercury.bao.ac.cn:9006/mh/spdc
  - welcome to register and try out
  - Products and vvpp:
    - http://mercury.bao.ac.cn:9006/svom-csc/svom
- Document by Sphinx
  - Will move to readthedoc.io
- To do:
  - Validate with GRM, GFT prototype pipelines
  - Generate product classes from data model descriptio
  - history
  - Beta release
  - Define messaging architecture
  - Serialization and toString based on STATE.



spdc

Navigation

Contents:

Install SPDC
dataset: Model for Data Container
pal: Product Access Layer
pns: Processing Node Server
spdc Quick Start

API Reference

Self-describing Portable Dataset Container (SPDC)

SPDC is a 'container' package written in Python for packing different types of data together, and letting the container take care of inter-platform compatibility, serialisa persistence, and data object referencing that enables lazy-loading. The word 'contai the name is more closely associated that is 'shipping container' instead of 'docker container'.

Features

With SPDC one can pack data of different format into **modular** Data Products, toget with annotation (description and units) and meta data (data about data). One can

spdc

Navigation

Contents:

Install SPDC
dataset: Model for Data Container
pal: Product Access Layer
pns: Processing Node Server
spdc Quick Start
- dataset
- pal
- pns

API Reference

Quick search

Go

spdc Quick Start

dataset

ArrayDataset

Creation

```
>>> a1 = [1, 4.4, 5.4E3]      # a 1D array of data
>>> a2 = 'ev'                 # unit
>>> a3 = 'three energy vals'  # description
>>> v = ArrayDataset(data=a1, unit=a2, description=a3)
>>> v1 = ArrayDataset(a1, a2, description=a3)  # simpler but error-p
>>> print(v)
ArrayDataset{ description = "three energy vals", meta = MetaData[], d
>>>
>>> print(v == v1)
True
>>>
```

data access

```
>>> v1.data = [34]
>>> v1.unit = 'm'
>>> print('The diameter is %f %s.' % (v1.data[0], v1.unit))
The diameter is 34.000000 m.
```

thanks