Massively Parallel Analysis System for Seismology (MsPASS): a Framework for New Frontiers in Seismology Research

Gary L. Pavlis

Department of Earth and Atmospheric Sciences

Indiana University, Bloomington, IN

Outline

- Overview of MsPASS
- Q&A
- MsPASS and Antelope
- Discussion: symbiosis of Antelope and MsPASS

MsPASS

1. Massively Parallel
   - Single CPU thing of the past
   - Archaic software infrastructure

2. Analysis System for Seismology
   - Research analysis – flexibility
   - For Seismology – tool for us not computer scientists

https://www.mspass.org/

What is in the box?

- Database to manage large data sets
- Scheduler to implement parallel processing
- Flexible but simple to use IO abstractions (read-write almost any format and from URL)
- Python job control language
- Algorithms – low level preprocessing (solved problem) focus
  - All obspy signal processing algorithms
  - Low-level windowing, bundling, and similar grungy stuff
  - Three-component primitives (rotation and transformation)
  - Trace editng
  - Header math
  - SNR calculations
  - All common P receiver function deconvolution+novel new method
- Runs on all linux, MacOS, and Windows via docker/singularity

One-liner perspectives on MsPASS

- Only open-source, generic system today for large-scale parallel processing of seismology data
- Only open-source, generic solution today suitable for cloud computing
- Obspy on steroids
- Runs on almost any system
- Written by a pair of geeky seismologists for seismologists

Design Goals

- Framework for processing seismology data
  - Scalable to exploit parallelism and massive storage
  - Open source
  - Enable reproducible science results – publish your notebook and someone can recreate our data set
- Research system
  - Flexible but powerful
  - Scalable from desktop to large clusters
  - Minimize initial startup energy barrier (essential for students) yet be extensible to any known data processing algorithm
  - Generic as possible

Production versus Research IT systems

Production

- Solves Specific Problem
- Performance is critical
  - Time is money
  - Mission critical role
- Operable with minimum skill set necessary for job
- Data model well known and fixed

Research

- Handle range of problems
- Performance is secondary
  - Feasible sufficient
  - Many one-up solutions
- Can assume users are specialists and life-long learners
- Data highly variable (one person's signal is another's noise)

MsPASS is NOT

- Ideal solution for problems we viewed as solved:
  - Real time data handling
  - Seismic event catalog processing
  - Seismic reflection processing
  - Archival data management
- Fully optimized
- Complete
  - This is a framework, not a turnkey solution to all problems
  - Full success requires extensions from people like you

Developers

- Ian (Yinzhi) Wang
  - Leader
  - His brainchild
  - Python guru
- Gary Pavlis
  - Design builds on my exper
  - C++ code base
  - Documentation
- UT Graduate Students
  - Weiming Yang
  - Jinxin Ma
  - Zhengtang Yang
  - Chenxiao Wang

Kent Lindquist: MsPASS is an example of “Software Craftsmanship” not “Software Engineering”

Sustainability (NSF jargon)

- A big issue in most open-source packages – software rusts
- Wang has NSF funding for now through SCOPED project
- I expect to work on MsPASS until I'm mentally incompetent or dead
- Most pressing long-term need is to build the user base

End of Nontechnical Overview

Put on your geek hats

MsPASS Major Components

Docker container

Scheduler (Dask or Spark)

MongoDB

python

jupyter notebook

Python code base ↔ pybind11 ↔ C++ code base

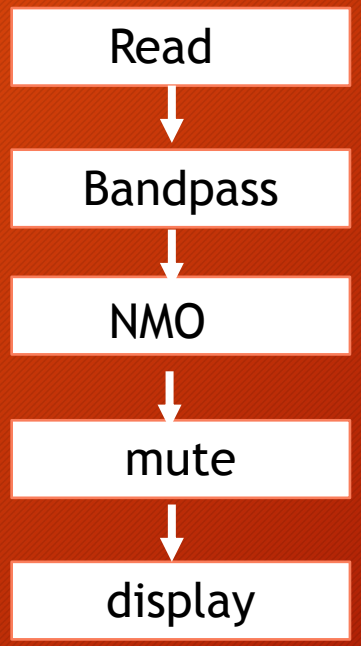
Element 1: parallel schedulers

- Map-reduce model
  - Modern jargon term
  - Online sources obscure some simple concepts
  - Main paradigm for MsPASS parallel processing
- A modern paradigm likely to have a long lifetime
- I will introduce by analogy to unix pipeline concepts I assume all of you know

Seismic Reflection Workflow model

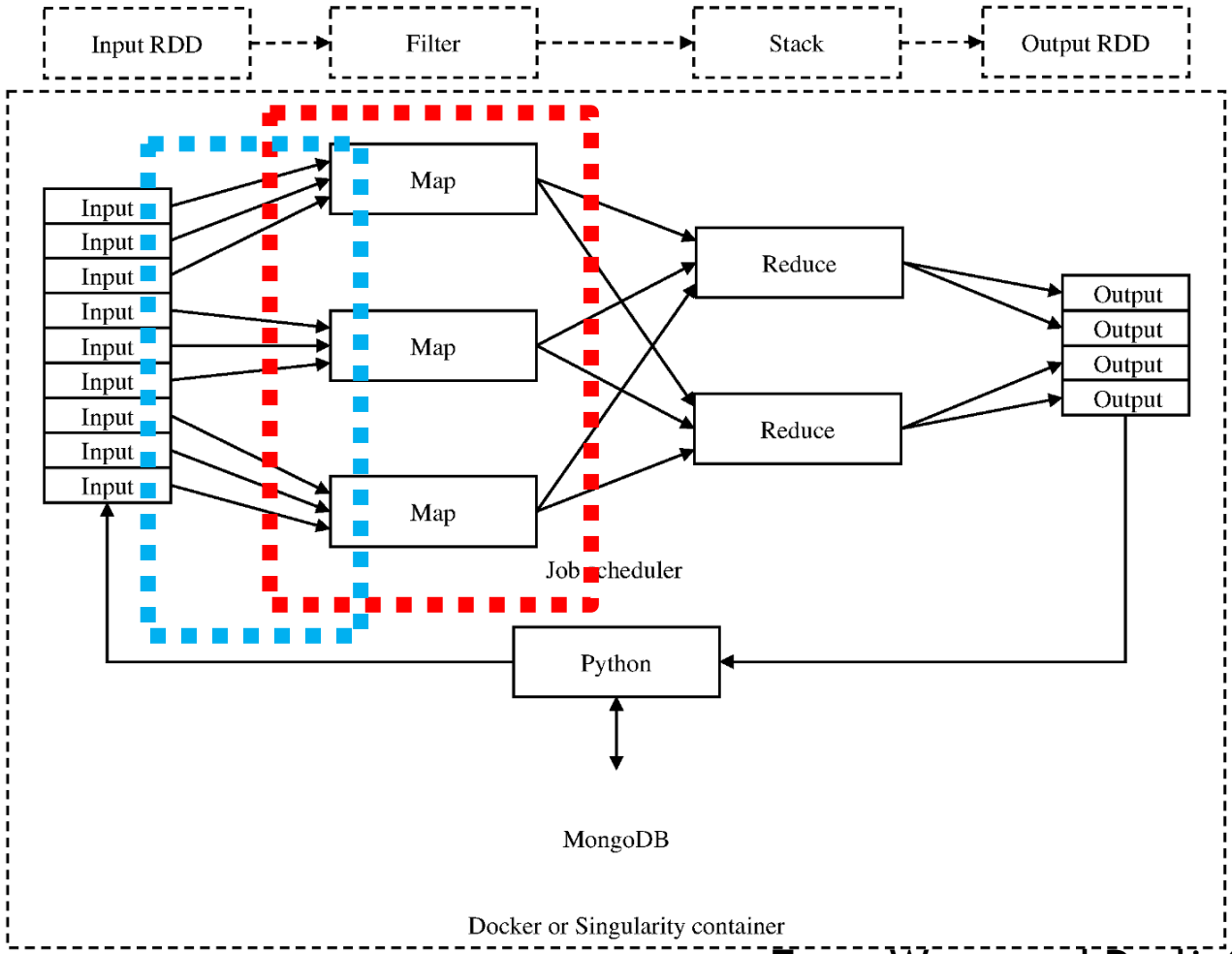
Example: seismic unix

- Illustrates traditional model
- Key point is data flow through processors
- Processors read input, modify it, and emit output




```
# real examples would add arguments for parameters
subfilt < mydata | sunmo | sumute | suxwig
```

MsPASS Parallelization: map-reduce model

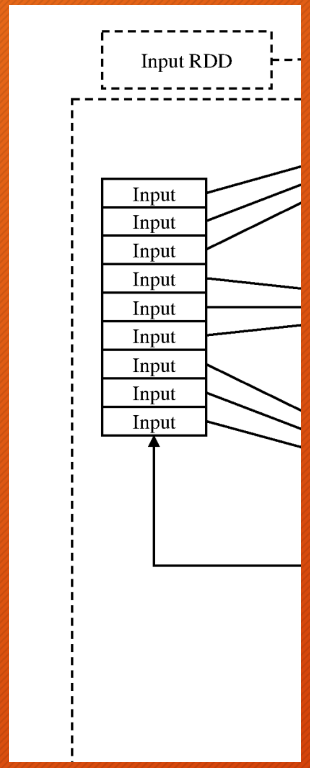


From Wang and Pavlis(2021)

Key points

- Map operator
  - Behaves like a unix filter
  - Scheduler assigns each datum to processes
- Reduce operator
- Data flow
  - Parallel pipeline
  - Conceptually similar to unix shell | symbol
  - Scheduler moves data
  - Faster for threads than nodes

Spark RDD == Dask bag



- Documentation for both Spark and Dask obscure this topic
- RDD/bag concepts
  - Algorithmically identical to a large array of things (objects)
  - Workers can pull any component in equal time
  - The collection of things (data set) may not fit in memory
- Other parallel containers
  - Dataframe (table)
  - Array (matrix bigger than memory)

Comparison of serial and parallel workflows

Serial

```
cursor = db.wf_Seismogram.find({})
for doc in cursor:
  d = db.read_data(doc,collection="wf_Seismogram")
  d = signals.detrend(d,'demean')
  d = signals.filter(d,"bandbass",
    freqmin=0.01,freqmax=2.0)
  d = WindowData(d,200.0,500.0,t0shift=d.t0)
  db.save_data(d,collection="wf_Seismogram",
     data_tag="results")
```

Parallel (Dask)

```
cursor = db.wf_Seismogram.find({})
data = read_distributed_data(db, cursor)
data = data.map(signals.detrend,'demean')
data = data.map(signals.filter,"bandpass",
freqmin=0.01, freqmax=2.0)
# windowing is relative to start time.  300 s window
starting at d.t0+200
data = data.map(lambda d : WindowData(d, 200.0,
500.0, t0shift=d.t0))
res = data.map(db.save_data,
     collection="wf_Seismogram",data_tag="results")
data_out = data.compute()
```

Key point: loop processing easily translated to series of map operators

Result acts like: demean < datafile | filter > outfile

Database Overview

- MsPASS uses MongoDB
  - What it is?
  - How it differs from Datascope and other relational dbms?
- Why we use MongoDB in MsPASS?

Header or Database: a 40+ yr long debate

Headers (e.g. SAC)

Strengths

- Simple conceptual model==Simple API
- Lightning fast metadata attribute access
- Fixed namespace reduces complexity

Weaknesses

- Repairing headers requires reading entire data set
- Fixed, limited attribute namespace

Relational DBMS

- Strengths
  - Easier to maintain metadata attributes
  - Extensible metadata namespace
- Weaknesses
  - Conceptual model much more complex
  - Today all transactions are VERY slow compared to computational speeds

MongoDB – what is it?

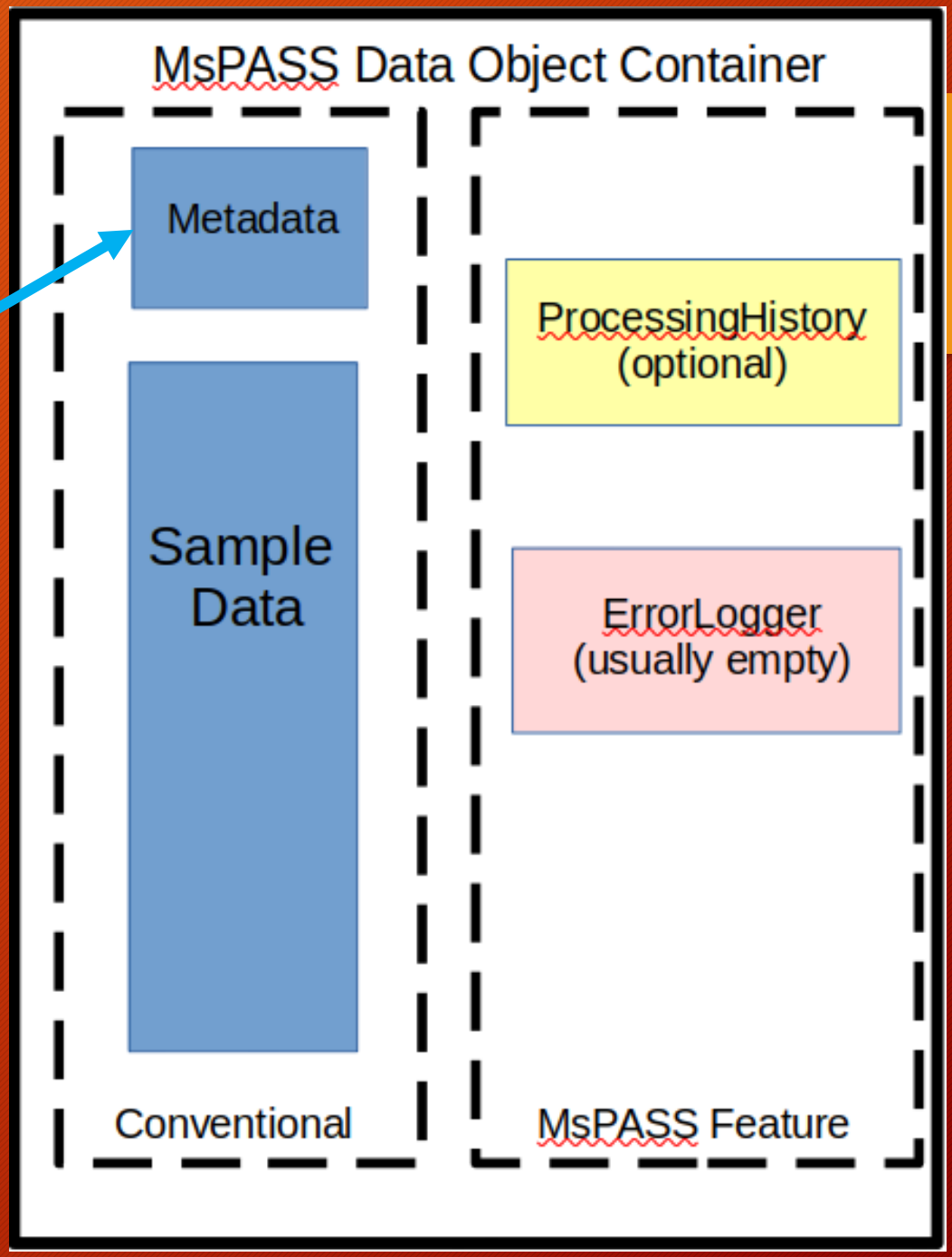
- Part of a family of “NoSQL DBMS” == Not Relational
- MongoDB a “Document Database” - misleading name
- Critical concepts:
  - All about key-value pairs
  - MongoDB’s “document” maps exactly into a python dictionary container
- Let’s look at a simple demonstration with a jupyter notebook

Key Points

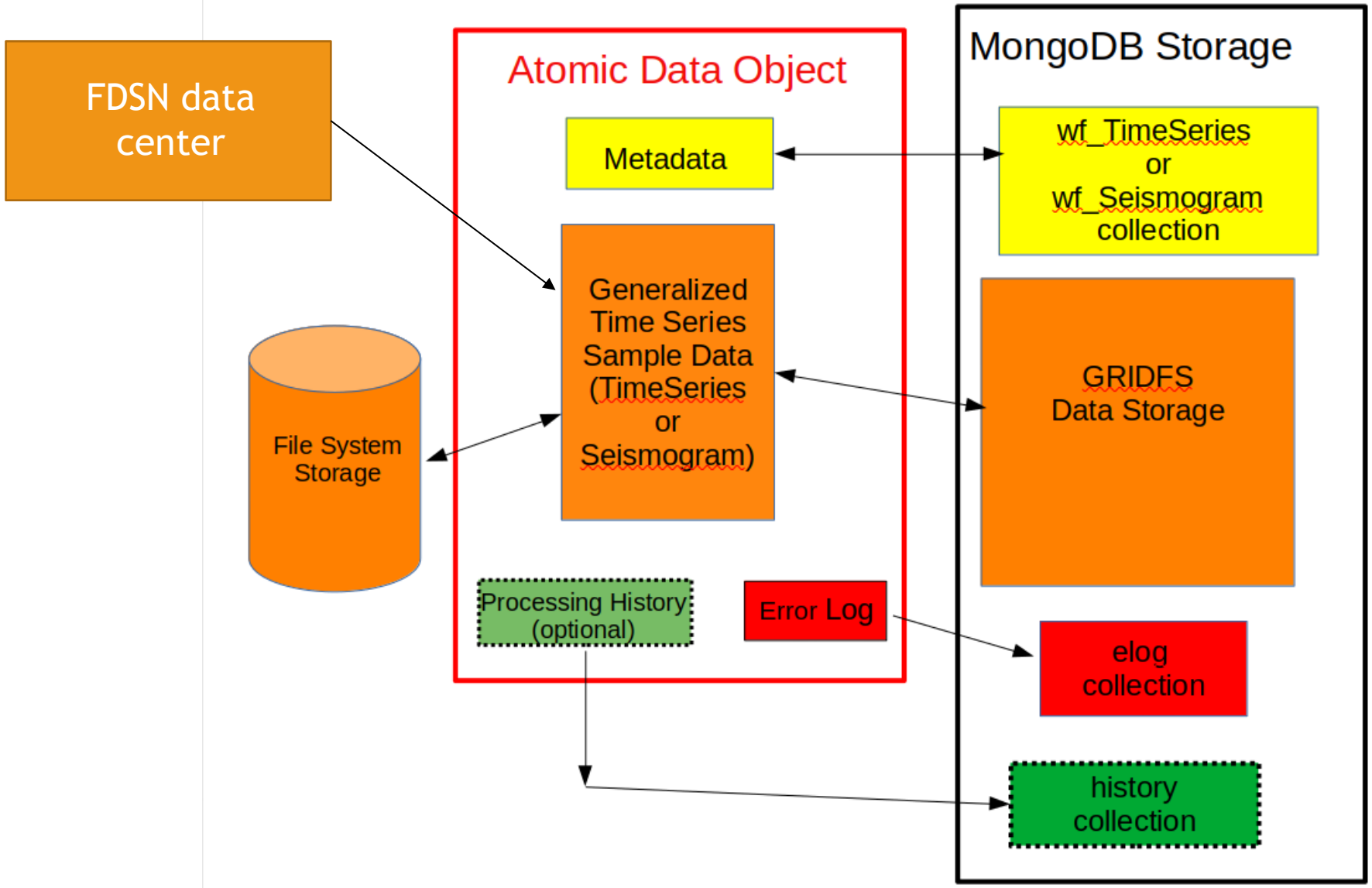
- MongoDB a perfect match for a “generalized header”
- By default MongoDB is completely promiscuous about what it saves (anything you can put in a python dictionary can be saved in db)

MongoDB collection

```
{ “npts” : 1024,
   “t0” : 601489897876697.89,
   “sta” : “AAK”
   ”chan” : “BHZ”
   …
}
{ “npts” : 1024,
   “t0” : 601489897876684.24,
   “sta” : “BZN”
   ”chan” : “BHZ”
   …
}
…
```



For geeky details see:
https://www.mspass.org/user_manual/data_object_design_concepts.html#


FDSN data center
Atomic Data Object
Metadata
Generalized Time Series Sample Data (TimeSeries or Seismogram)
File System Storage
Processing History (optional)
Error Log
MongoDB Storage
wf_TimeSeries or wf_Seismogram collection
GRIDFS Data Storage
elog collection
history collection

Docker

Practical

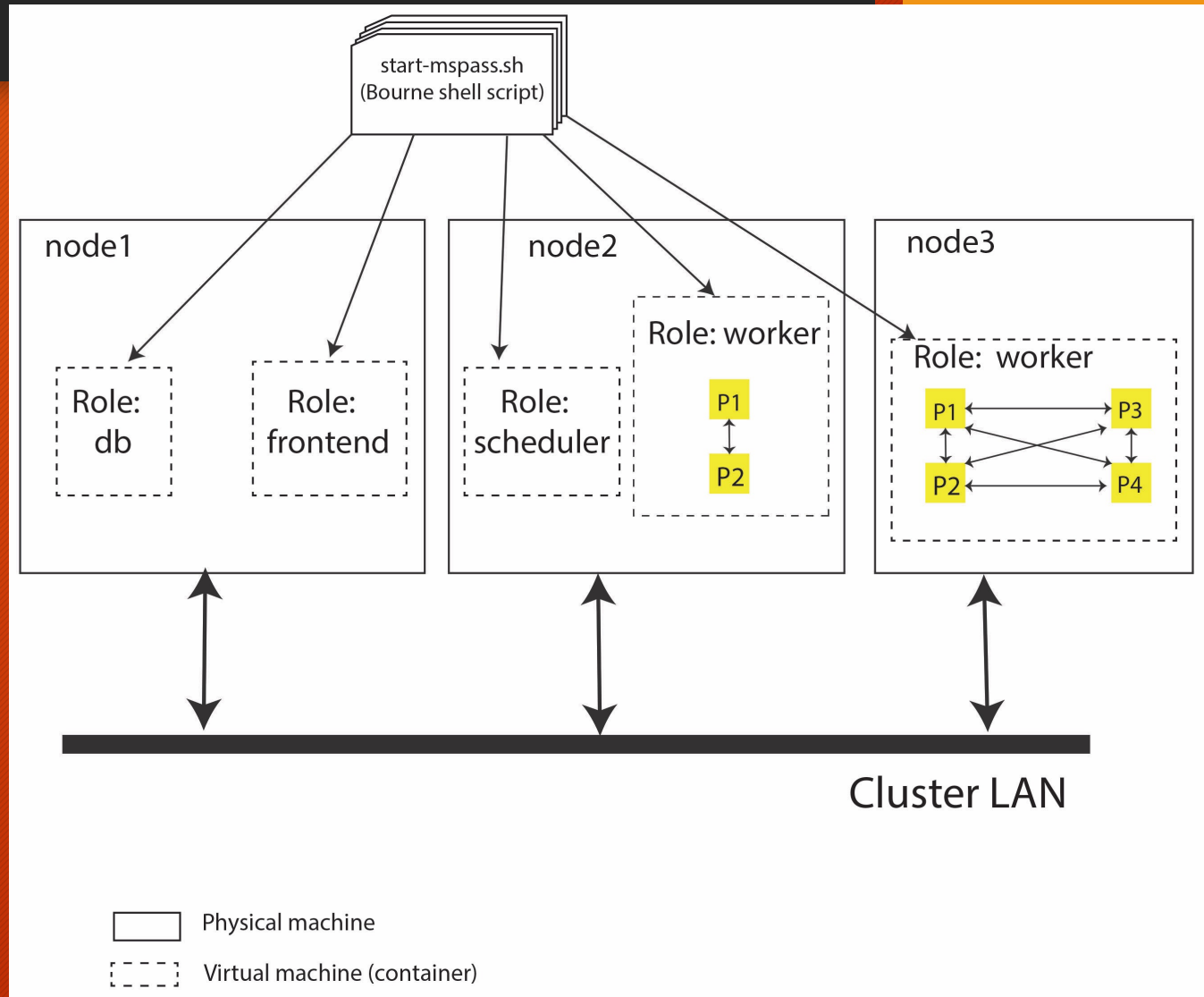
- Solution for python package collisions
- Allows MsPASS to run on any platform supporting docker
- Avoids open source complexities to build from source code (binary distribution)
- Easy entry point for cloud computing – you can just run on AWS

Details

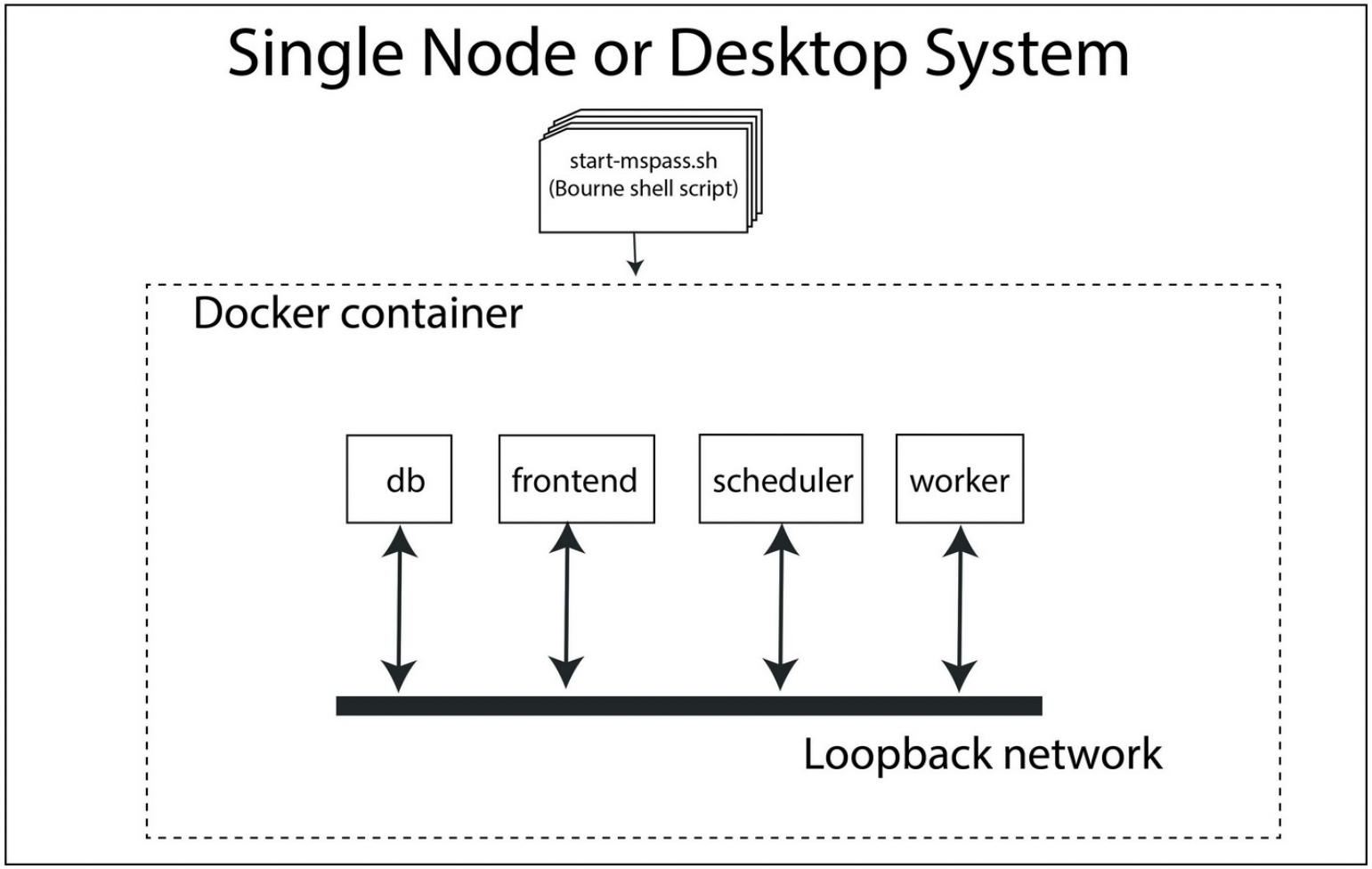
- Our container uses MongoDB set up with Ubuntu as base
- We extend the base with Dask, Spark, and Jupyter
- We extend the base container with MsPASS code base (including obspy) and a few other smaller packages
- Support intel and new mac hardward (arm64)

Abstraction: Virtual Cluster

- Example:
  - 6 workers
  - 6 processor ”cluster”
  - 3 physical “nodes”
- Expandable to as many nodes/cores as available
- Docker is the enabling technology for our approach



Desktop: abstracted as one worker cluster



- Abstracted as one worker cluster
- Worker can use multiple cores
- The 4 “roles” are background processes in the same container

References:

- User manual
- github site
  - Source code for entire package
  - Discussion pages
  - Issues pages
- Tutorial github repository of Jupyter notebooks
- Publication in SRL
- SCOPED docker repository

Cite this article as Wang, Y., G. L. Pavlis, W. Yang, and J. Ma (2021). MsPASS: A Data Management and Processing Framework for Seismology, *Seismol. Res. Lett.* **93**, 426–434, doi: 10.1785/0220210182.

Part 2: MsPASS and Antelope

Questions about MsPASS?

Starting Points

- We thought hard about lessons from Antelope's Datascope we should adopt in MsPASS
- We thought hard about weaknesses in Datascope we needed to overcome
- Our aim was to complement Antelope and other software – MsPASS is a framework

Datascope Strengths and Weaknesses

Strengths

- Performance
- Relatively easy to maintain
- Multiple ways to access the db: shell, C/C++, perl, and python
- Join is fast and efficient
- It just works

Weaknesses

- Designed before parallel computing became mainstream
- Schema can be changed but is inflexible
- The API is horribly complex (e.g, how many 0's for dblookup?)
- A simple "find" is hard
- Platform portability limits
- Documentation

MsPASS design leans on Antelope and other seismology software development history

- MongoDB, like Datascope, can be easily maintained without a DB admin
- Worked hard to simplify API as much as possible
- Leading edge but not bleeding edge components (The lesson of CORBA)
- Make it work before you make it fast – not Enterprise software
- Militantly object-oriented design
  - Essential to separate API from implementation details
  - Well matched to python
  - More maintainable
- Documentation viewed as critical
  - User manual
  - Python API – sphynx generated docstrings (similar to obspy)
  - C++ code base – doxygen generated pages

Antelope-MsPASS interaction: what is needed?

- Datascope table translation to MsPASS MongoDB
  - Main topic we focused on
  - Multiple prototypes (next slide)
- Simplified ways to import/export data between systems
  - Needs work on both ends
  - May allow evolution away from aging Datascope
- Interaction with orb
  - ORB+MsPASS as data transfer agent (slide later)
  - MsPASS alternative to wfprocess for event driven RT applications

Current MsPASS and Antelope interface

- Caveat: first 2 should be considered prototypes
- Version 1: found here on github
  - Early prototype we may deprecate
  - Not integrated with automatic test suite (may be broken)
- Version 2: Datascope handle python object found here
  - In development site for new plane wave migration
  - An OOP interface to Datascope
  - Driven by a pf partial css3.0 defintion found here
  - Loads tables in pandas dataframe
- Framework component of importance: normalize module
  - Generic algorithms solving more than Antelope imports
  - Likely framework for additional development

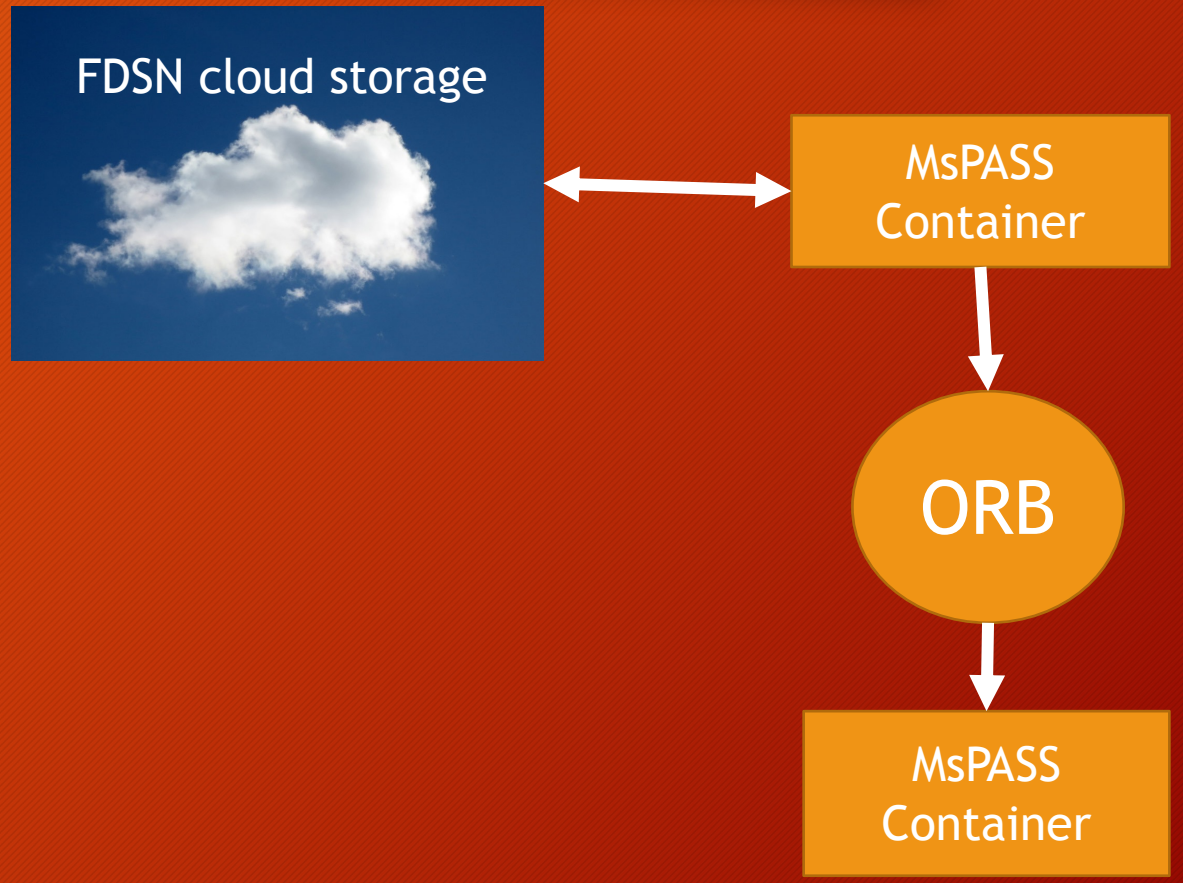
Normalization

- Reference in user manual here
- Generic definition of “Matcher” object
  - Concept is a generic api to match a row of a table to one or more Metadata key-value pairs
  - Generic version of what dbjoin does
- Have multiple working examples using Datascope tables loaded as Panda Dataframe
- Biggest future use is matching arbitrary tables of data to waveforms to load Metadata

Idea: Antelope as transfer middleware

Concepts:

- ORB is efficient long-haul internet transfers (orb2orb)
- Cloud storage
  - Future for FDSN data delivery
  - Web service NOT error free transmission and slow
  - MsPASS prototype with SCEC data center
- MsPASS could be effective as a reader and writer through ORB
- ”All” that is needed is an orb reader and writer for MsPASS



Discussion:

- How might MsPASS help your personal research program?
- How might MsPASS help others in your institution?
- What could we do to improve MsPASS to help you?
- How can we expand MsPASS functionality?
- I showed what we have for import tools from Datascope tables. What else might be needed?
- Are there ways MsPASS could be an aid in network operations?
  - Event-driven real time processes?
  - Bulletin preparation? (e.g. framework for machine learning or large scale cross-correlation processing?)