JavaSeis Cloud: A micro-service framework for seismic processing

Chuck Mosher[1], Sanjay Sood[1], Robert Ferguson[2]

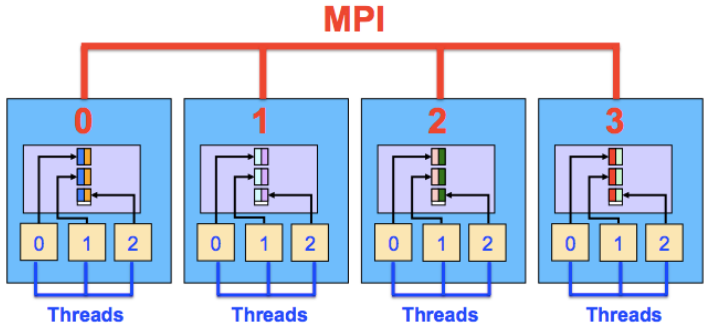
1- MoMacMo Limited, 2- University of Calgary

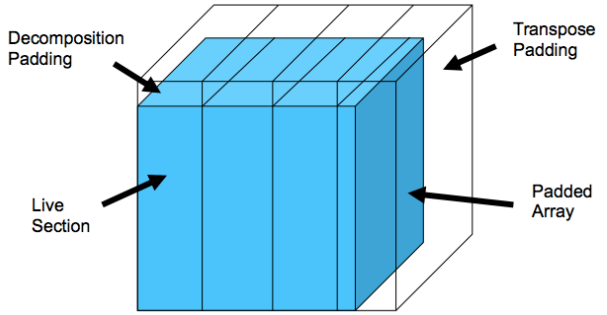
Outline

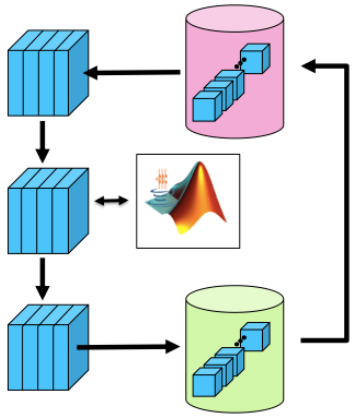
- How cloud-based microservices add value for processing
- Cloud components for microtask services
- Language and Format vs API and Object Protocols
- Protocols for cloud based seismic processing
- The JavaSeis model for seismic data
- Compact state for seismic processing tasks
- Object structure for a seismic processing task
- Managing task submission and completion
- Examples and conclusions

JavaSeis: Synchronous Parallel Computing

- Public domain (JavaSeis.org)
- Hosted on SourceForge and GIT
- Object oriented synchronous parallel design
- High level abstractions for geophysics
- Parallel input, Parallel computation, Parallel output
- Scalable to 1000's of cores
- 20 years of development and production applications



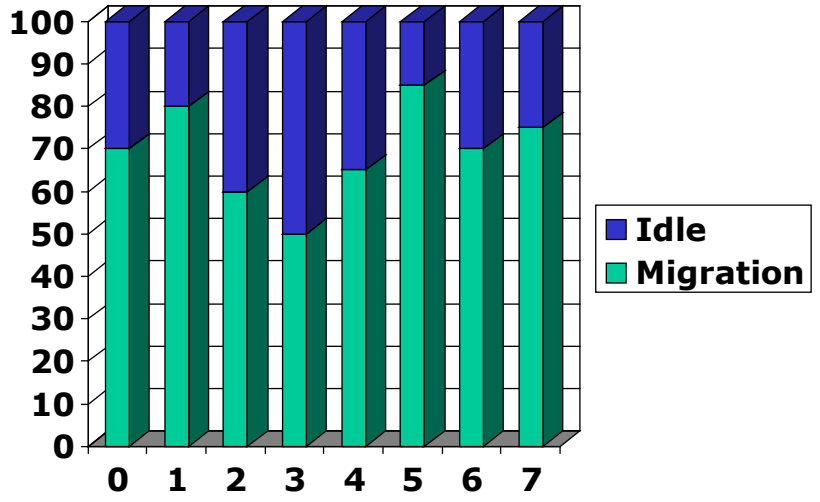


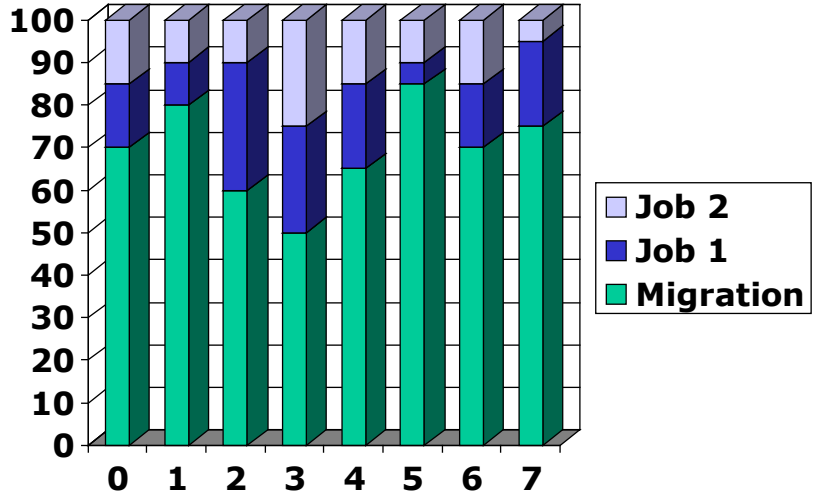


JavaSeis

Microtasks can “scavange” unused cycles

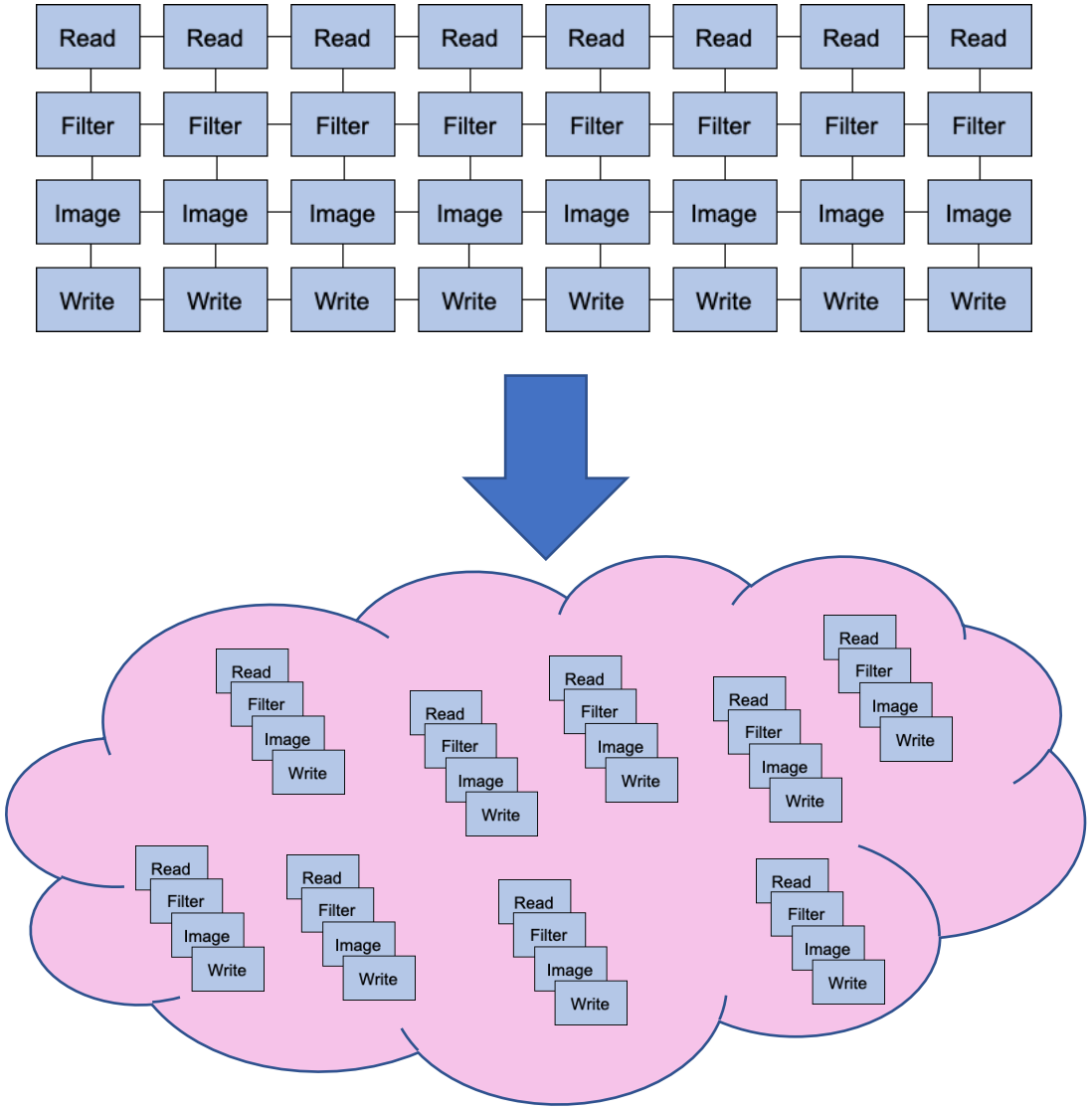
- Large MPI jobs occupy ALL cores but rarely run at 100%
- Microtasks can be submitted to utilize unused cycles
- Microtask submission can be throttled back if MPI utilization increases
- The number of “concurrent” microtasks grows and shrinks with available capacity





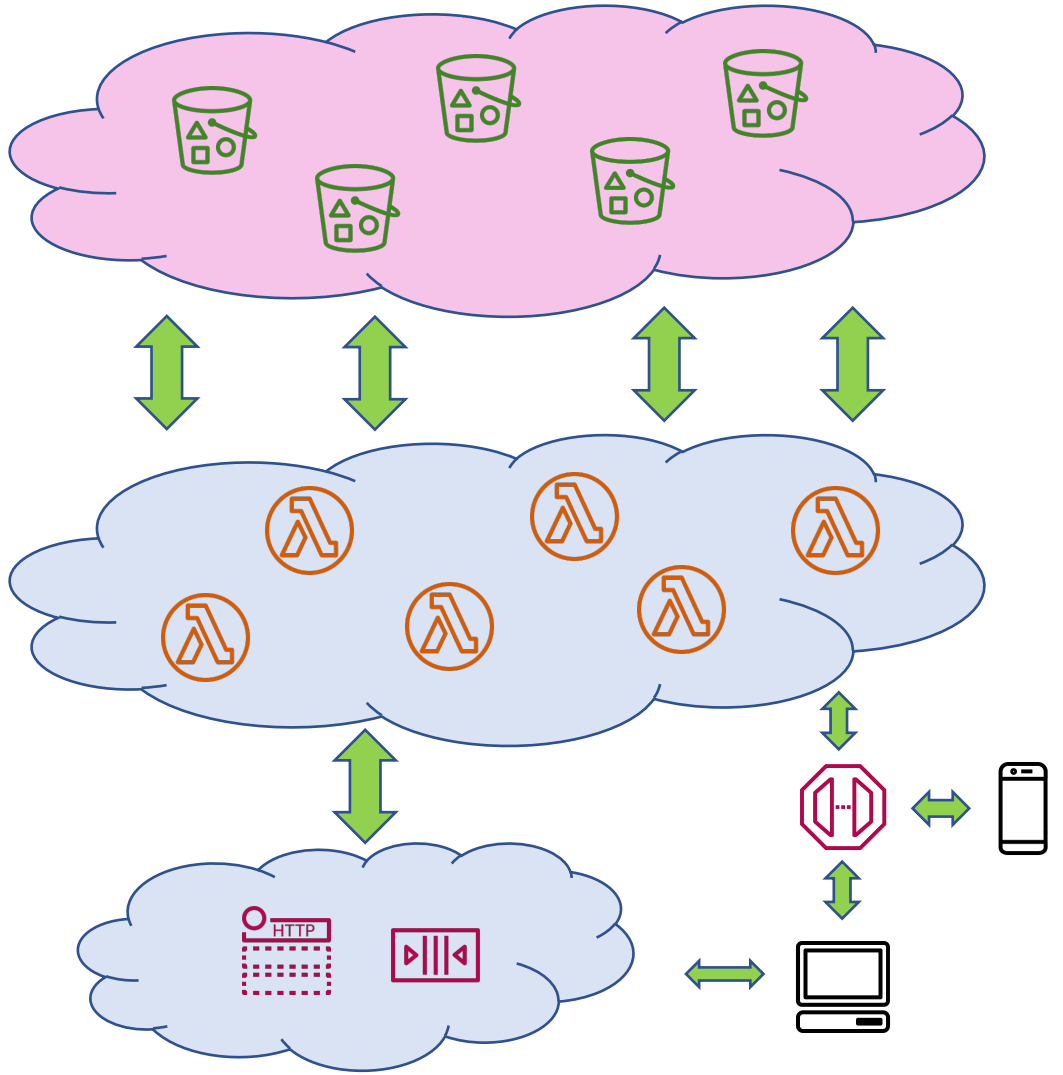
Asynchronous Task Parallel Computing

- Augment synchronous MPI style with a model for “stateless” independent tasks
- Supplement high-performance parallel file systems with distributed key-object storage
- Provide infrastructure for starting, monitoring, and completing a “cloud” of tasks
- Requires re-architecting applications for the new model



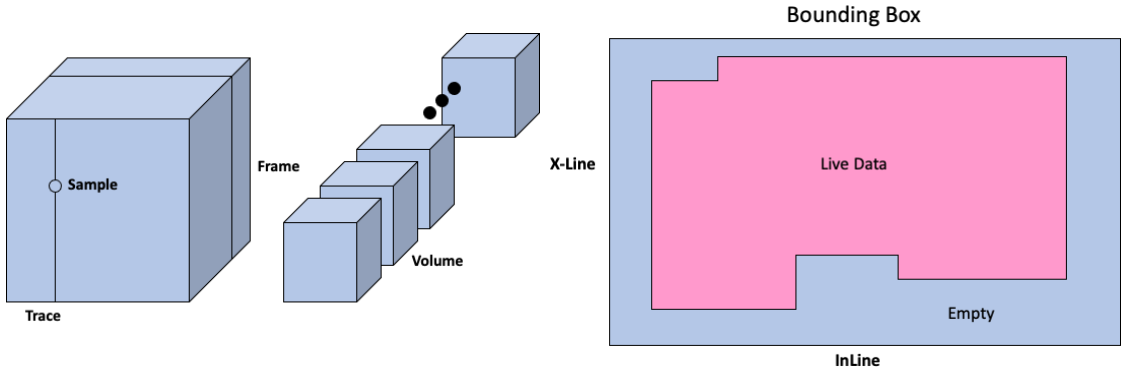
Cloud Components for Microservices

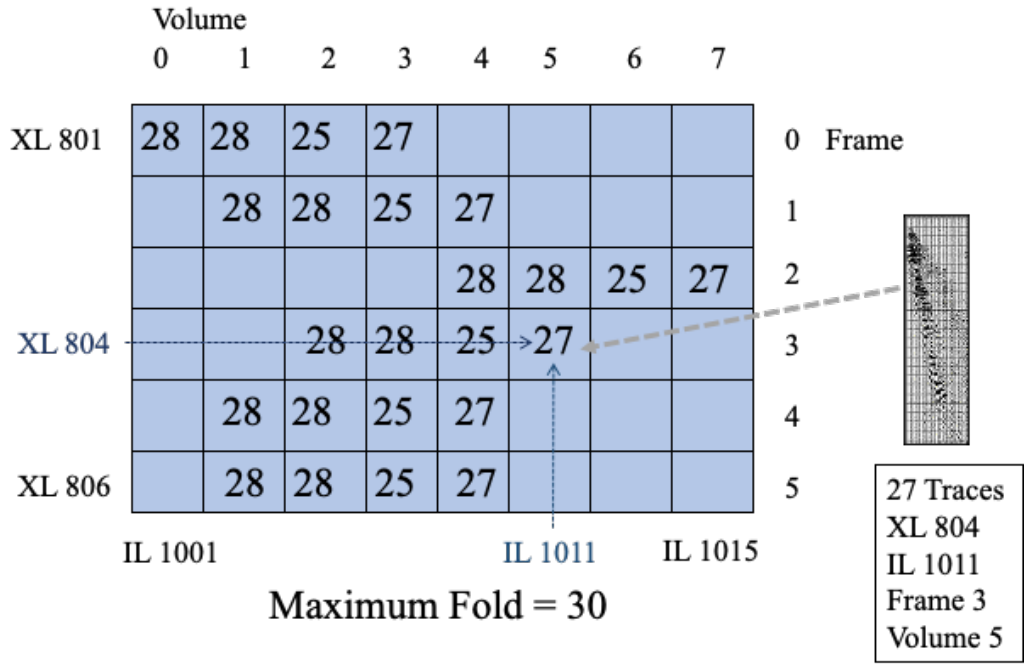
- Key-Value Object storage
  - Scalable object storage
  - Amazon S3, Azure Blob Storage
- Serverless Compute Functions
  - Stateless compute service
  - AWS Lambda, Azure Functions
- Application Programming Interfaces
  - Allows computer programs to interact with compute and storage over the internet
  - AWS API Gateway, Azure API Management
- Messaging and Notification
  - Services that send and receive messages
  - AWS Simple Notification Service, Azure Notification Hubs



JavaSeis Object Protocol for Pre-Stack Seismic Data

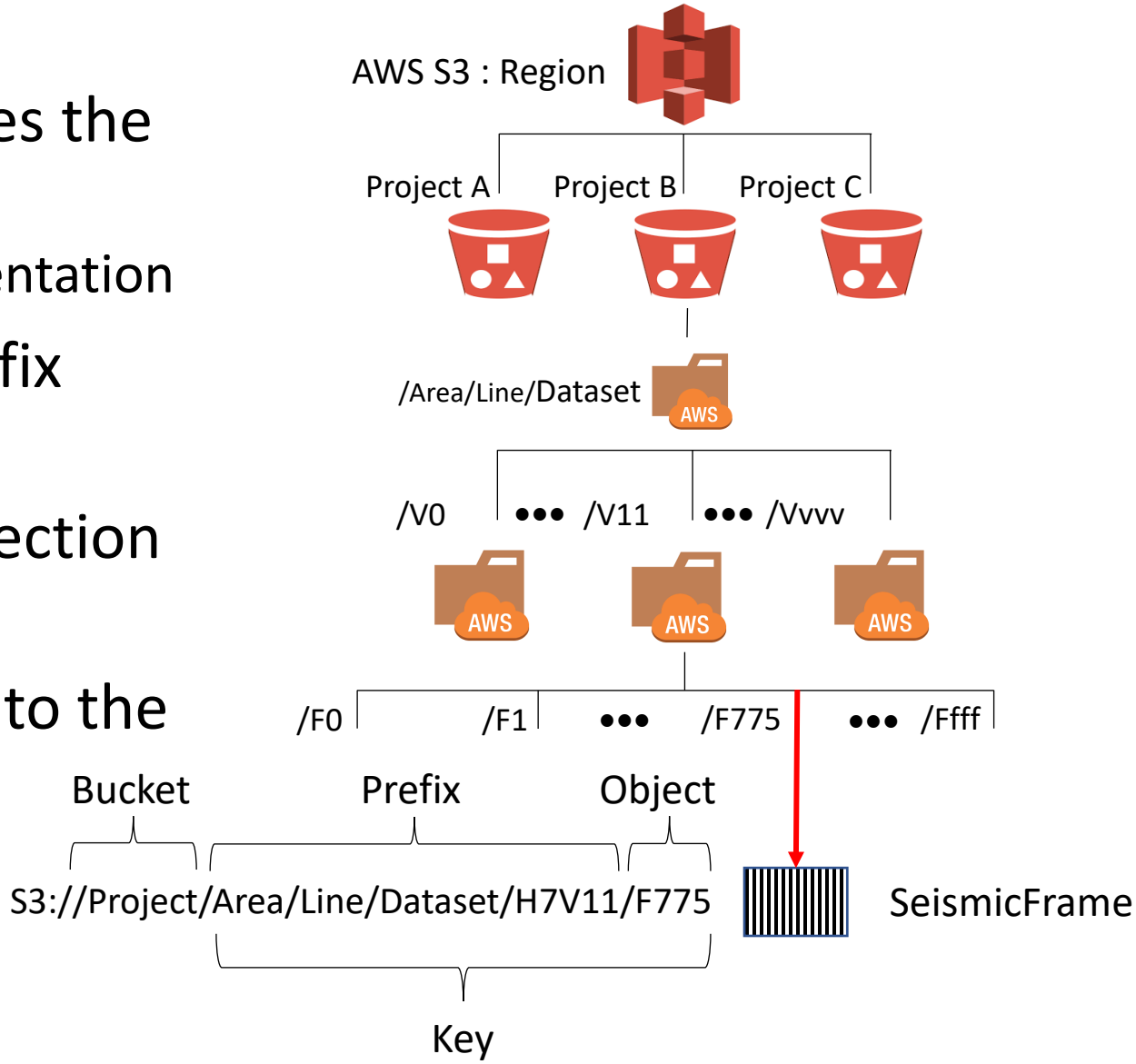
- Represent multi-dimensional seismic data as a collection of 3D volumes
- A bounding box defines the limits of the dataset
- A “Fold Map” records the number of traces in each cell of the bounding box
- JavaSeis provides an interface that supports this model
- Build a key-object interface





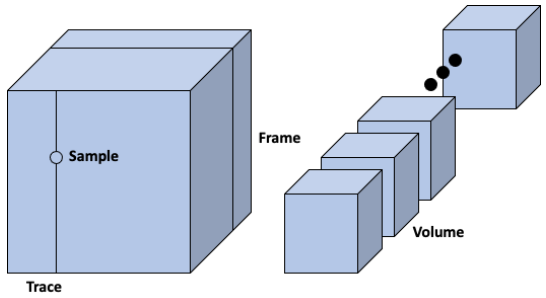
JavaSeis Cloud Interface for Key-Object Storage

- JavaSeis Cloud (JSC) Interface uses the same objects as JavaSeis
  - This example shows AWS implementation
- A JSC dataset is mapped to a prefix (folder) in an AWS bucket
- N-D data is represented as a collection of volumes
- Higher dimensions are flattened to the volume dimension
- Frame keys are defined as: DataSet/Vvvv/Ffff



Compact State for Seismic Tasks

- We define a JavaSeis Context that contains:
  - The N-Dimensional definition of the axes of the dataset framework
  - The range of indices in the dataset that this task will process
  - Description of available properties for the dataset
  - A coordinate transformation grid to convert from local to world coordinates
- And a task state the contains:
  - A list of processing tasks and their parameters
  - Objects and protocols for seismic data
  - Logging functions

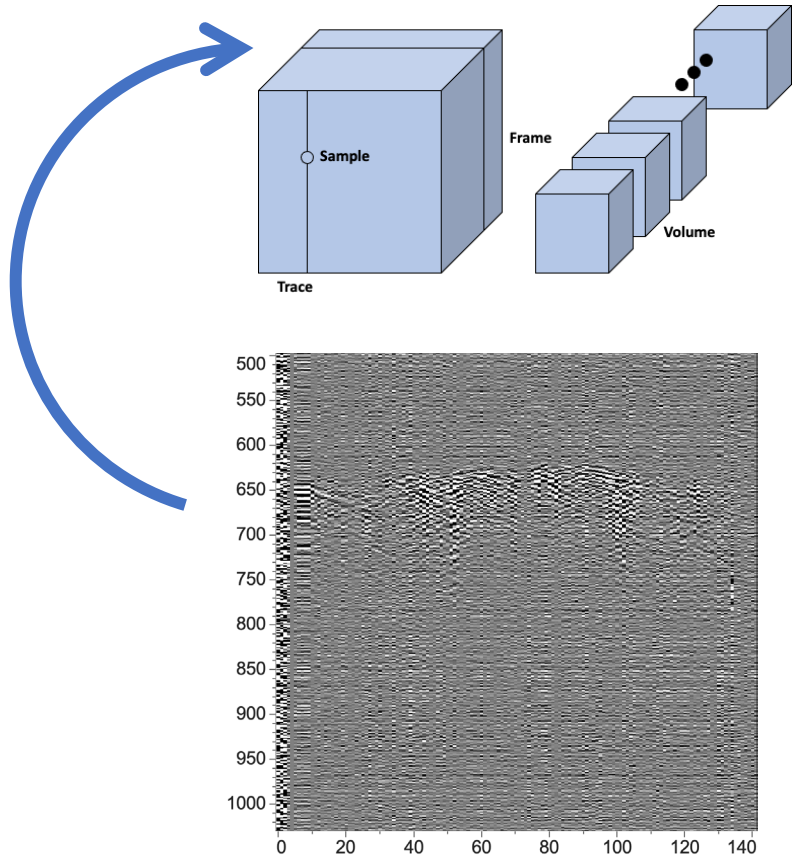



```
<parset name="JavaSeis Metadata">
  <parset name="FileProperties">
    <par name="Comments" type="string"> "www.javaseis.org - JavaSeis
    <par name="JavaSeisVersion" type="string"> 2006.3 </par>
    <par name="DataType" type="string"> UNKNOWN </par>
    <par name="TraceFormat" type="string"> COMPRESSED_INT16 </par>
    <par name="ByteOrder" type="string"> LITTLE_ENDIAN </par>
    <par name="Mapped" type="boolean"> true </par>
    <par name="DataDimensions" type="int"> 4 </par>
    <par name="AxisLabels" type="string">
      TIME
      CHANNEL
      TIME_WIN
      TIME_GRP
    </par>
    <par name="AxisUnits" type="string">
      milliseconds
      meters
      meters
      meters
    </par>
    <par name="AxisDomains" type="string">
      time
      space
      space
      space
    </par>
    <par name="AxisLengths" type="long">
      1250
      380
      8640
```

AWS S3 Seismic Input
    Input Path Name
Band Pass Filter
    Filter corner points
AWS S3 Seismic Output
    Output path

JSC SeismicFrame Object

- The JSC SeismicFrame object contains:
  - A reference to the JavaSeis Context for the data being processed
  - A 2D frame of seismic trace samples
  - A Frame of trace properties (trace headers)
  - The Position of the frame in the context
- And methods that provide:
  - Access to trace samples
  - Getting and setting trace properties
  - Iterators that traverse traces and properties

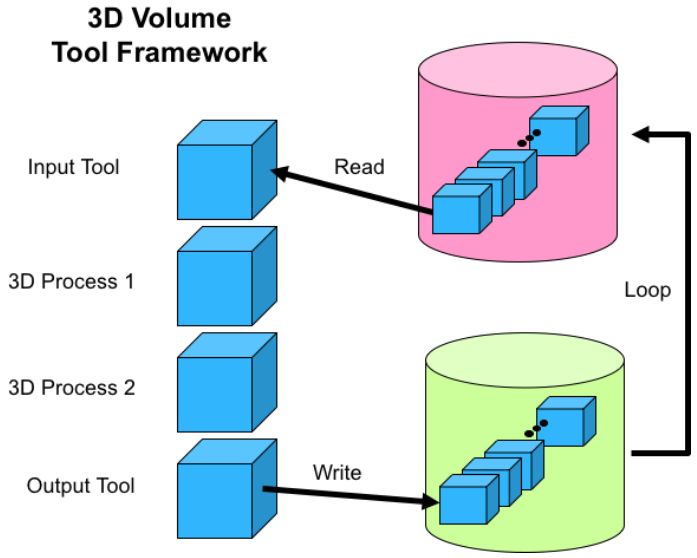



```
seismicFrame.getProperty("SOU_X")

for ( float[] trace : seismicFrame)
    applyFilter(trace);
```

JavaSeis Cloud Processing Task

- Based on the JavaSeis Parallel Computing engine:
  - Processes 2D frames rather than 3D volumes
  - Processing modules are executed sequentially
  - Input and output modules at head and tail
  - JavaSeis Tiled Transpose for sorting
- Task state that contains:
  - A list of processing modules and their parameters
  - The location in the framework of the frame(s) to be processed
- Simple processing executive that:
  - Allocates resources based on context/task state
  - Calls processing modules in order
  - Manages input and output state for each module



AWS S3 Seismic Input
s3://bucket/prefix/input
Volume 47 Frame 13
Band Pass Filter
{ 5, 15, 45, 60 }
AWS S3 Seismic Output
s3://bucket/prefix/output

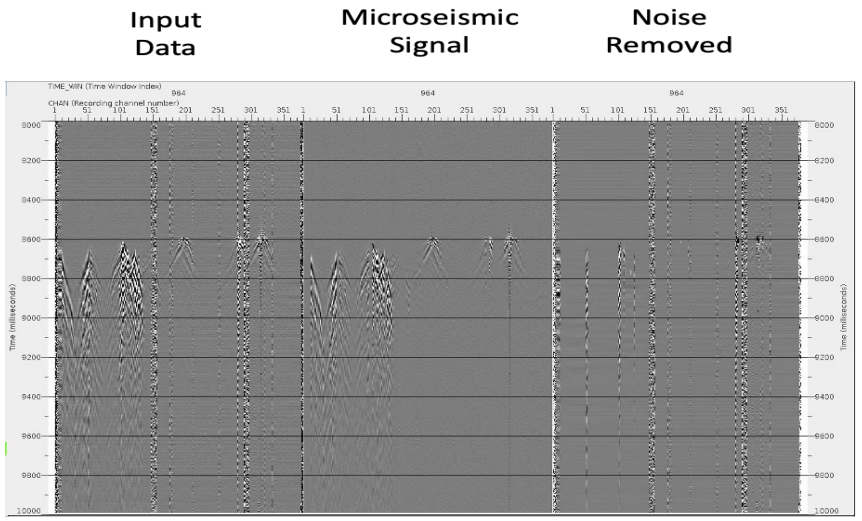
JavaSeis Cloud Processing Module

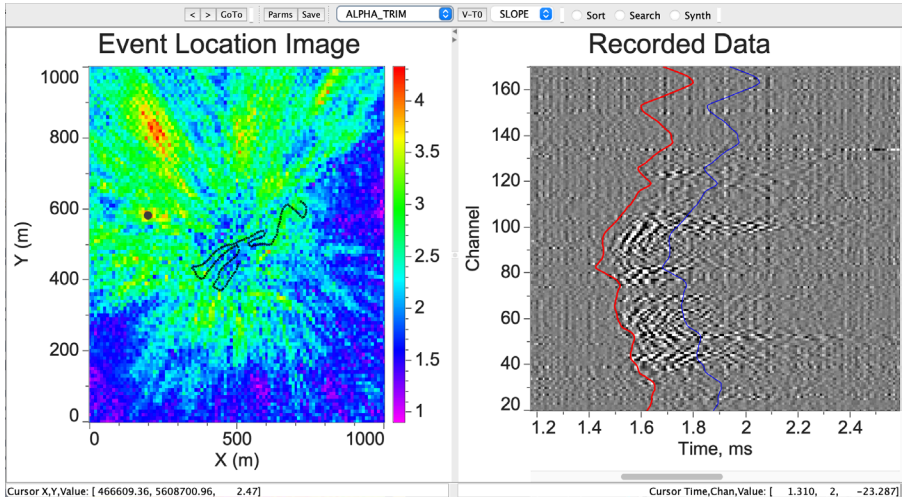
- Familiar JavaSeis processing model:
  - Serial Init phase to check parameters and declare resources needed
  - Parallel init phase when for initialization of local and shared resources
  - Parallel exec phase for processing frames
  - Parallel finish phase to release local resources
  - Serial finish phase to release shared resources
- Modules can:
  - Communicate and collaborate
  - Change the state and framework at runtime
  - Read and write from available local disk
  - Read and write from object and database storage

```
 8  public class ExampleProcessingModule implements IFrameModule {
 9
10    private static final long serialVersionUID = 1L;
11
12    @Override
13    public void checkPhaseInit(ModuleState moduleState) throws SeisException {
14       // TODO Auto-generated method stub
15    }
16
17    @Override
18    public void checkPhaseFinish(ModuleState moduleState) throws SeisException {
19      // TODO Auto-generated method stub
20    }
21
22    @Override
23    public void runPhaseInit(ModuleState moduleState) throws SeisException {
24      // TODO Auto-generated method stub
25    }
26
27    @Override
28    public boolean processFrame(ModuleState moduleState) throws SeisException {
29      // TODO Auto-generated method stub
30      return false;
31    }
32
33    @Override
34    public boolean outputFrame(ModuleState moduleState) throws SeisException {
35      // TODO Auto-generated method stub
36      return false;
37    }
38
39    @Override
40    public void runPhaseFinish(ModuleState moduleState) throws SeisException {
41      // TODO Auto-generated method stub
42
43    }
44
```

Example JavaSeis Cloud Processing Workflow

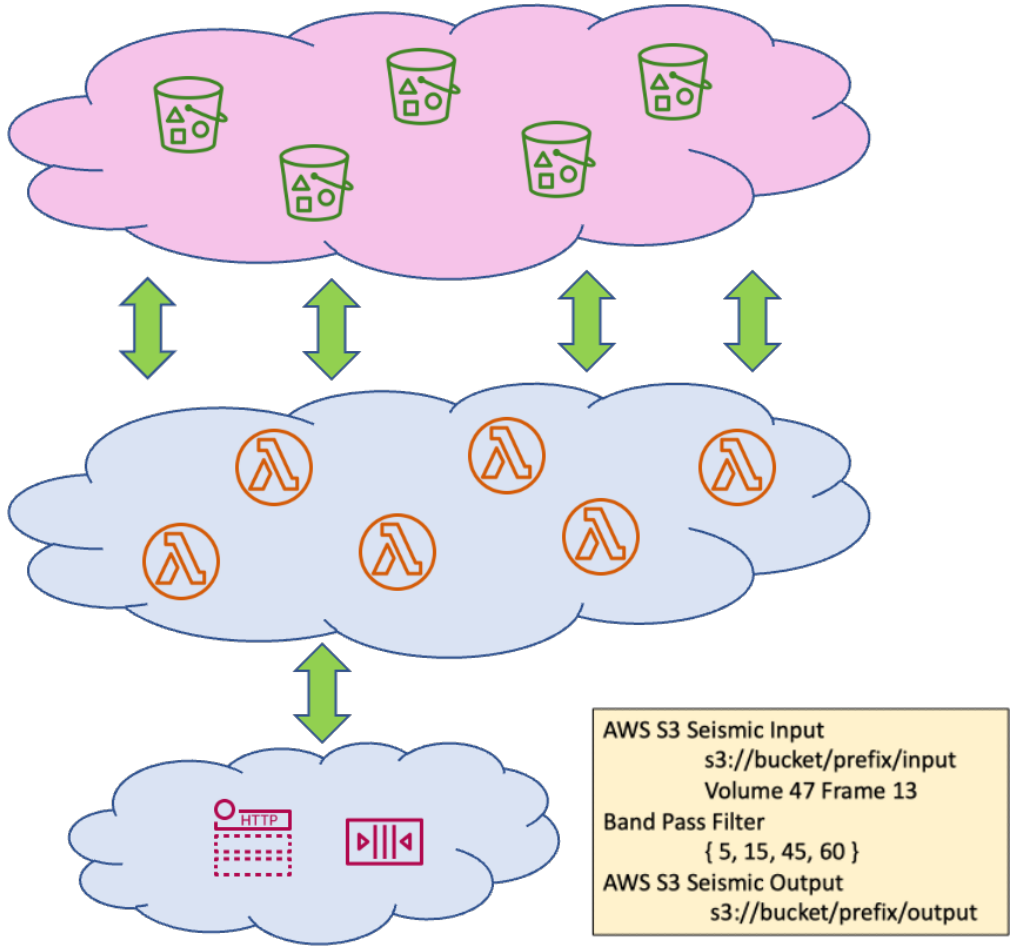
- Sort from continuous time windowed time:
  - Input data as TIME,CHAN,TIME_GRP
  - Create windowed time gathers TIME,TIME_WIN,CHAN,TIME_GRP
  - Sort to TIME_WIN domain TIME,CHAN,TIME_WIN,TIME_GRP
  - Write to object storage
- Extract features and apply machine learning
  - Despike in TIME_WIN domain
  - Apply Bandpass filter
  - Apply Spatial median filter
  - Write to object storage in TIME_WIN domain
  - Machine learning event detection and location
  - Write event table to object storage





Example submission and monitoring workflow

- Construct object containing input parameters
  - Same object used by all tasks
  - Serialize to JSON
- Construct and send API request
  - Request for task submission
  - Triggers serverless function that issues message notifications for all tasks
  - Each message triggers serverless task that performs computation
- Monitor task completion
  - Through console or API requests from an application



Observations so far …

- This is a work in progress
  - We use JavaSeis Cloud for the majority of our data storage and I/O
  - All of the processing we have applied to large datasets uses JavaSeisCloud
  - Tools and processes are in their infancy
- We use a ”host” processing system to manage projects and datasets
  - We build tools that read and write from the cloud
  - Job submission to the cloud as “standalone” modules
  - Plug-ins that run JavaSeisCloud Modules in the host system as well
- JavaSeis Cloud Costs are lower than “lift and shift”
  - Serverless compute is cheaper when “bursting” to the cloud
  - There is a breakover point in utilization where spot instances are cheaper

JavaSeis Cloud: Conclusions

- Scalable computation based on microservices is a strength of cloud
- JSC provides an additional model for computation complementary to traditional seismic HPC synchronous parallel applications
- A significant portion of seismic data processing tasks can be mapped to cloud microservices
- Costs are lower than dedicated resources
- Fault tolerance is not free, but can be engineered appropriately
- Requires modest investment in refactoring existing code
- Cloud hosting of data and applications enables collaboration and access to a rich set of machine learning and data science tools

Thanks!