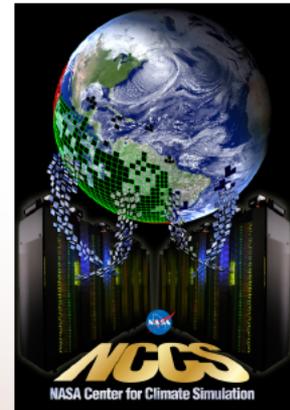


National Aeronautics and Space Administration



SCIENCE

The Climate Data Analytic Services (CDAS) Framework

Serving ESGF CWT and NASA CDS

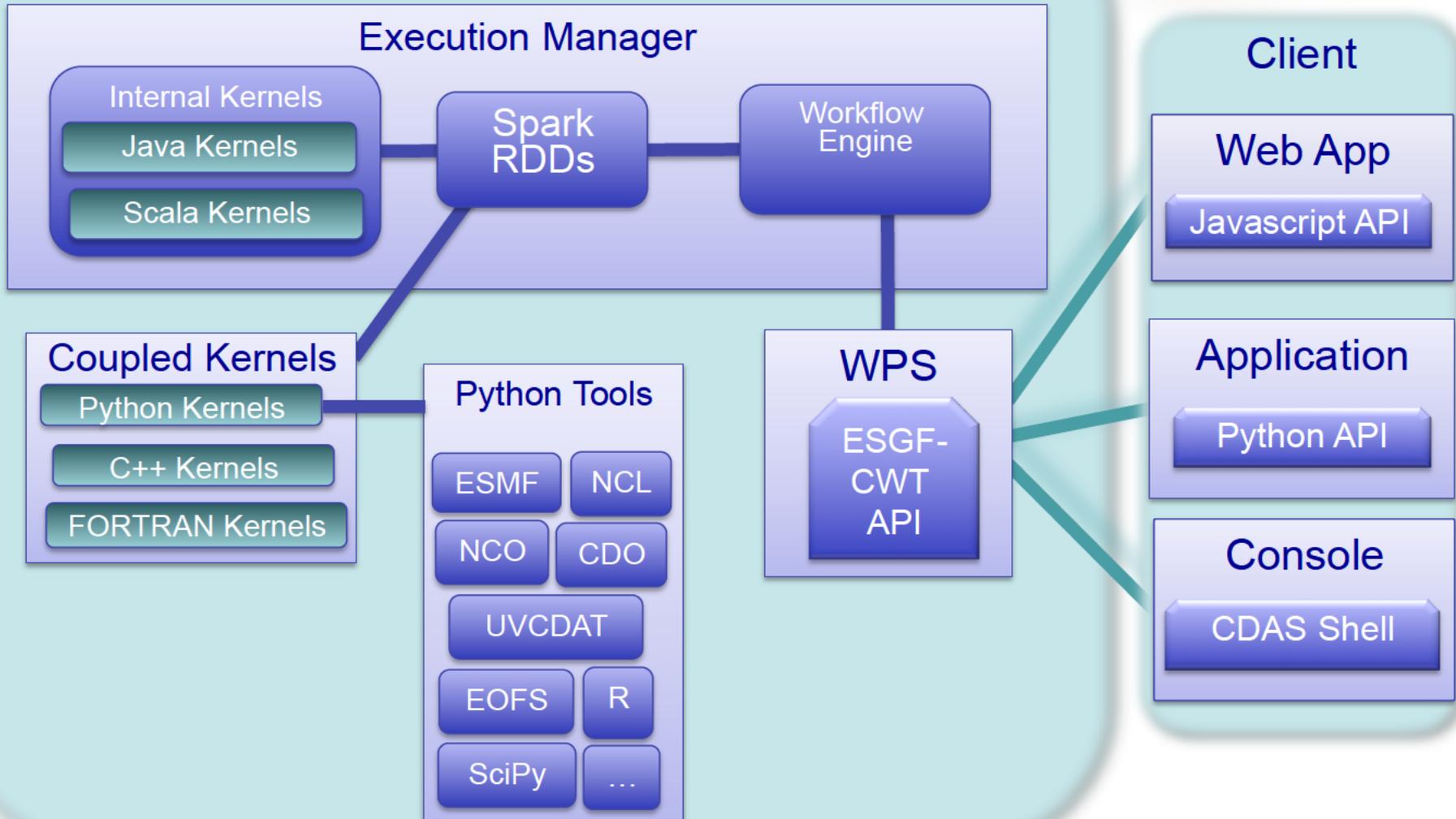
Thomas Maxwell and Dan Duffy

CDAS Design Drivers

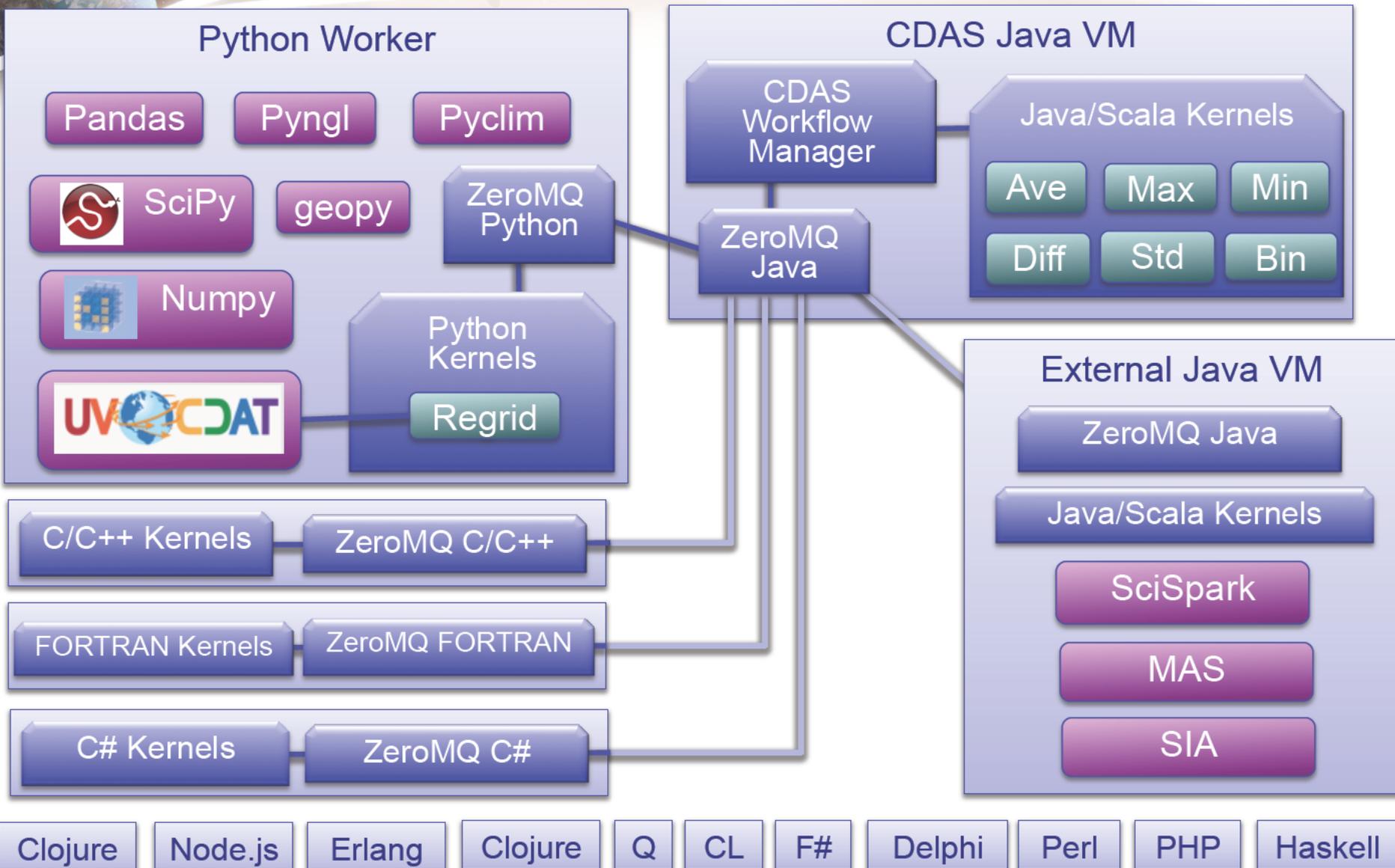
- Access data in raw (NetCDF, HDF) format in POSIX filesystem.
 - Avoid supporting additional copies of entire data holdings.
 - Cache variables of interest in domain (xyzt) of interest.
- Enable interactive performance on simple operations.
 - Light weight WPS implementation using the scala Play framework.
 - Most operations highly IO bound: high performance data cache.
- Utilize modular, composable compute operations (kernels).
 - Link kernels to compose workflows.
- Support existing climate data analysis packages.
 - Enable kernel development in a wide range of programming languages.
 - Parallelize data, not analysis packages.
- Deploy the ESGF CWT WPS API.
- Leverage existing big data technologies.
 - CDAS core developed in java/scala using Apache Spark engine.

CDAS: Climate Data Services Framework

CDAS Service



CDAS Workflow Kernels



ZeroMQ Worker Interface

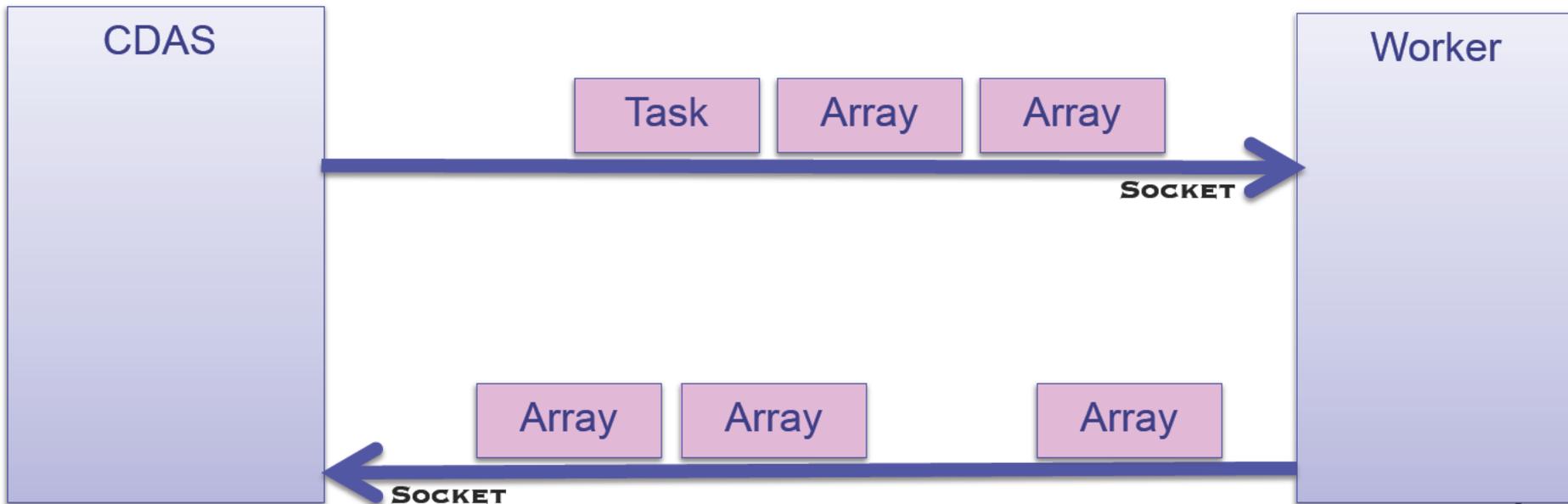
Array:

Header: String
ID, origin, shape, metadata

Data: byte[]

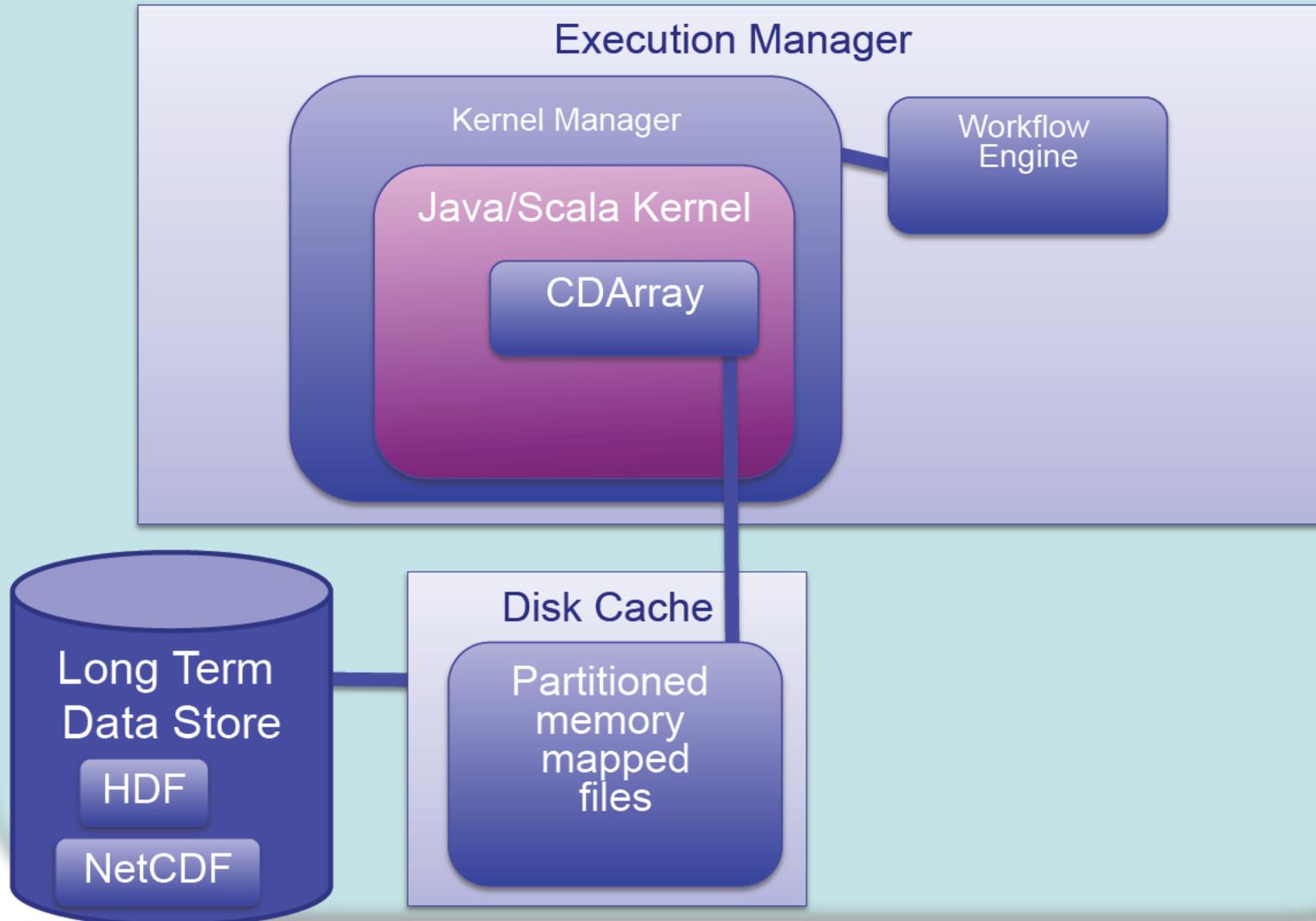
Task:

Task: String
Operation, RID, inputs, metadata



CDAS Data Input and Caching

CDAS



CD Array

Java Heap

Analytics API: Map, Reduce, Broadcast

Array API: get(i), put(i), slice, iterate

Array Core

Indexer

1D Float Buffer

Java Float Array

Disk Cache

Memory Mapped File

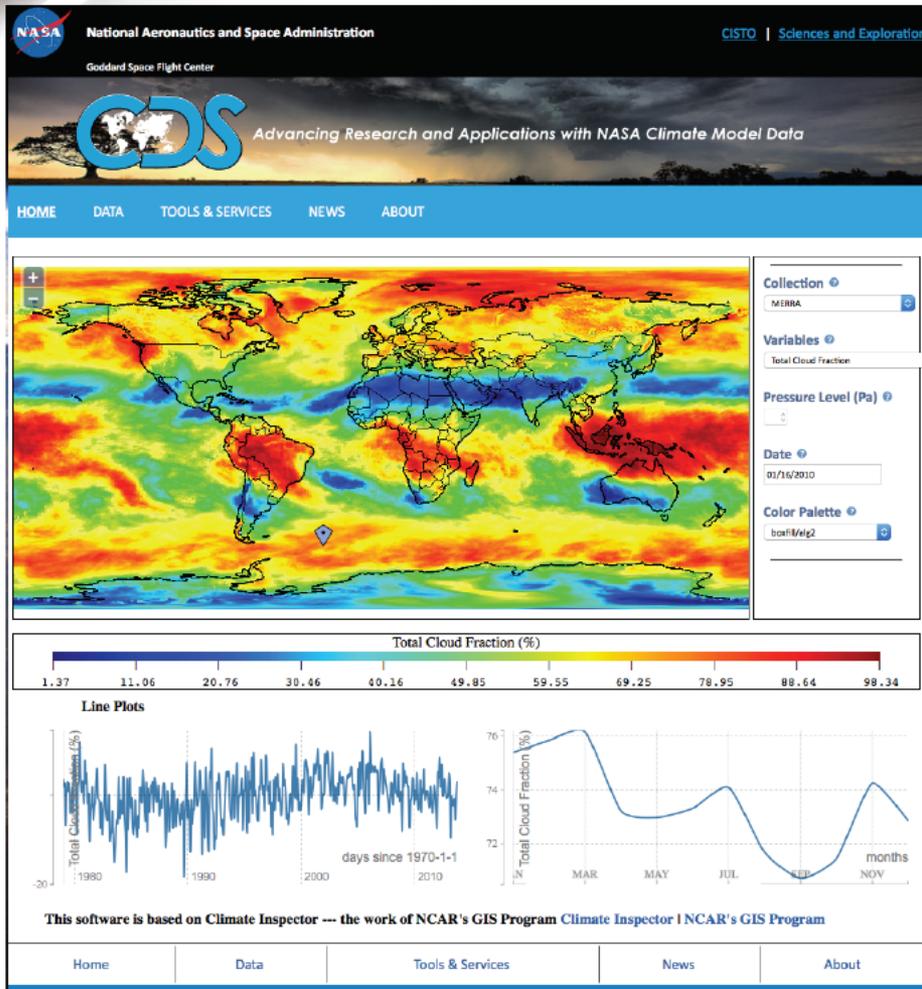
Dynamic Data Cache

- Data requested by cached fragment ID:
 - Use that fragment directly
- Data requested by collection ID, varName, and ROI:
 - Search cache for fragment that satisfies request:
 - Matches collection ID and varName
 - Overlaps requested ROI
 - If cached fragment is found then subset and return
 - If no fragment is found:
 - Cache requested ROI for the requested variable
 - Can be different from ROI of operation
 - “Precache” with empty operation
 - Return new fragment when done.

Partitioning and Parallelism

- Data Initially Partitioned over Time Axis:
 - Matches data file partitioning.
 - MERRA: ~ 10K files partitioned by time.
 - Other partition schemes require a reshuffle operation.
 - Each partition represented by a CDArray
- Streaming parallelism implemented using Spark.
 - In-memory workflow pipelines.
 - Extends Sparks' lazy execution model.
 - Kernel computations utilize Map-Reduce style operations.

CREATE-V CDAS Client

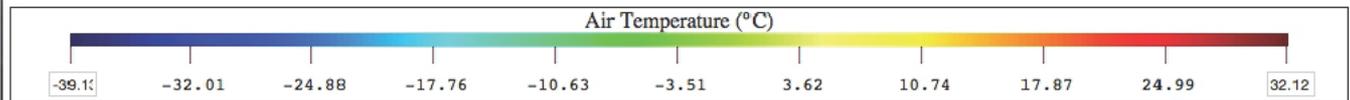
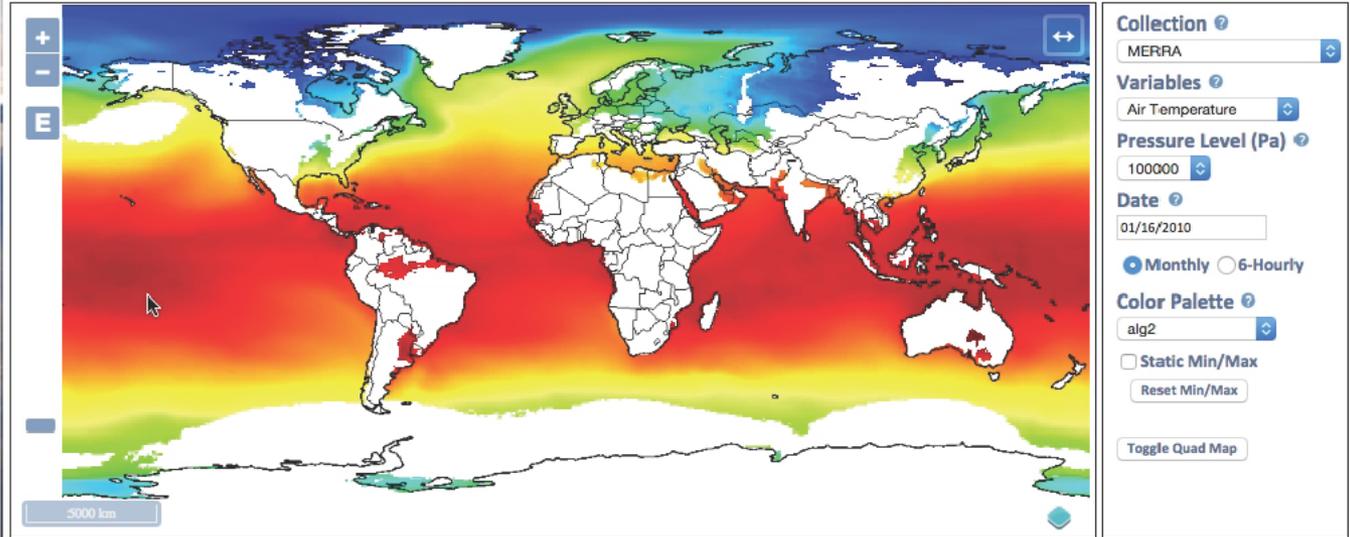


An interactive web application that expands GIS mapping capabilities for visual analysis of reanalysis data

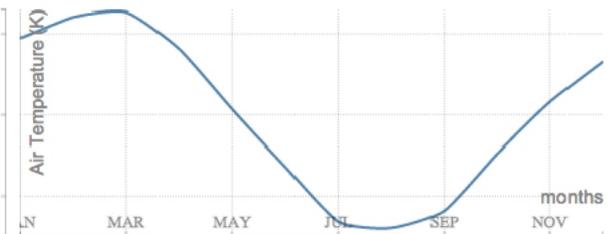
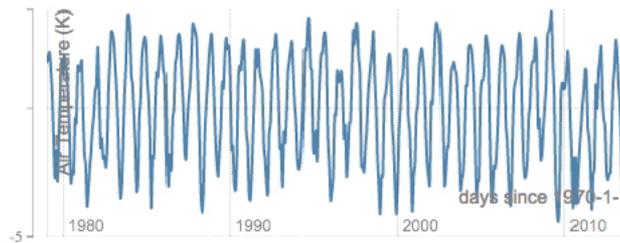
Permits decision makers to investigate climate changes around the globe, through time, inspect model trends, compare multiple reanalysis datasets, and variability



CDAS Analytic Services for Create-V



Analysis Plots



CDAS Python API

```
import esgf, os
```

```
class DemoWorkflow:
```

```
    def run( self ):
```

```
        d0 = esgf.Domain([], name='d0')
```

```
        op1 = esgf.Operation.from_dict( { 'name': 'CDSpark.multiAverage' } )
```

```
        for i in range(1,3): op1.add_input( esgf.Variable('collection:/GISS_r%di1p1'%(i), 'tas' ) )
```

```
        op3 = esgf.Operation.from_dict( { 'name': 'CDSpark.regrid', 'crs':'gaussian~128' } )
```

```
        op3.add_input( op1 )
```

```
        op2 = esgf.Operation.from_dict( { 'name': 'CDSpark.multiAverage' } )
```

```
        for i in range(1,3): op2.add_input( esgf.Variable('collection:/GISS-E2-R_r%di1p1'%(i), 'tas' ) )
```

```
        op4 = esgf.Operation.from_dict( { 'name': 'CDSpark.regrid', 'crs':'gaussian~128' } )
```

```
        op4.add_input( op2 )
```

```
        op5 = esgf.Operation.from_dict( { 'name': 'CDSpark.multiAverage' } )
```

```
        op5.add_input( op3 )
```

```
        op5.add_input( op4 )
```

```
        wps = esgf.WPS( 'http://localhost:9001/wps', log=True, log_file=os.path.expanduser("~/esgf_api.log") )
```

```
        wps.init()
```

```
        process = esgf.Process( wps, op5 )
```

```
        process.execute( None, d0, [], True, True, "GET")
```

```
    executor = DemoWorkflow()
```

```
    executor.run()
```

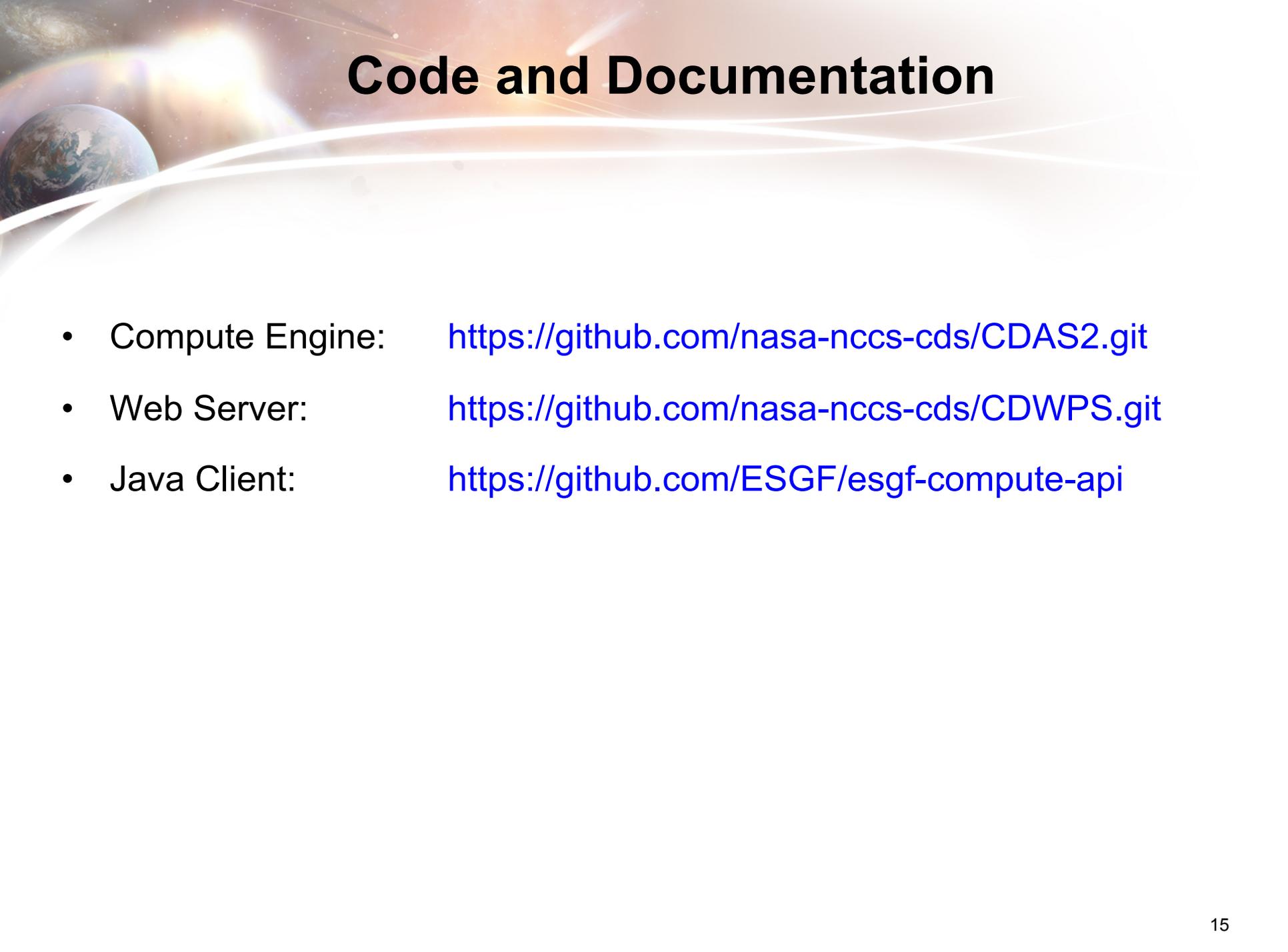
WPS Request

```
http://localhost:9001/wps?status=True&version=1.0.0&datainputs=[
  variable=[
    {"domain":"d0","uri":"collection:/GISS-E2-R_r3i1p1","id":"tas|vR3"},
    {"domain":"d0","uri":"collection:/GISS-E2-R_r2i1p1","id":"tas|vR2"},
    {"domain":"d0","uri":"collection:/GISS-E2-R_r1i1p1","id":"tas|vR1"},
    {"domain":"d0","uri":"collection:/GISS_r5i1p1","id":"tas|vH5"},
    {"domain":"d0","uri":"collection:/GISS_r4i1p1","id":"tas|vH4"},
    {"domain":"d0","uri":"collection:/GISS_r1i1p1","id":"tas|vH1"},
  domain=[{"id":"d0"}];
  operation=[
    {"input":["vR1","vR2","vR3"], "name":"CDSpark.multiAverage", "result":"b2761"},
    {"input":["vH1","vH2","vH3"], "name":"CDSpark.multiAverage", "result":"665c"},
    {"input":["665c"], "crs":"gaussian~128", "name":"CDSpark.regrid", "result":"32235"},
    {"input":["b2761"], "crs":"gaussian~128", "name":"CDSpark.regrid", "result":"12d9f"},
    {"input":["32235","12d9f"], "domain":"d0", "name":"CDSpark.multiAverage", "result":"323a5f"} ]
]
&service=WPS&Identifier=CDSpark.multiAverage&request=Execute&store=True
```

Get Capabilities

- List available resources
 - Available Data Collections.
 - Analytical Operations,
 - Cached Data Fragments,
 - Executing Jobs,
 - Cached Execution Results

```
http://localhost:9000/wps?  
    request=getCapabilities&  
    service=cds2&  
    identifier=collections
```



Code and Documentation

- Compute Engine: <https://github.com/nasa-nccs-cds/CDAS2.git>
- Web Server: <https://github.com/nasa-nccs-cds/CDWPS.git>
- Java Client: <https://github.com/ESGF/esgf-compute-api>