Coasters:
uniform resource provisioning and access for clouds and grids

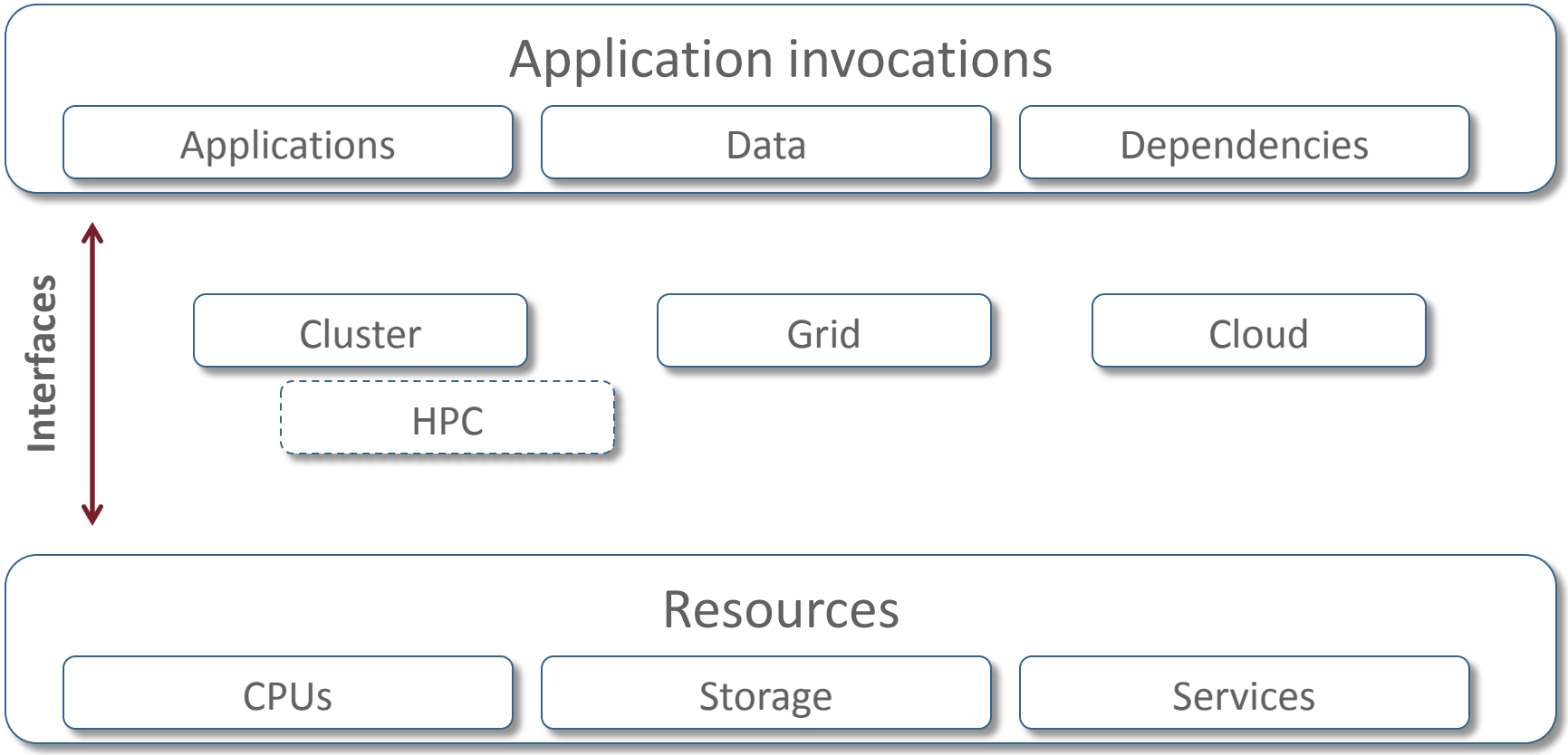
Mihael Hategan, **Justin M Wozniak**, Ketan Maheshwari

Argonne National Laboratory

Presented at:

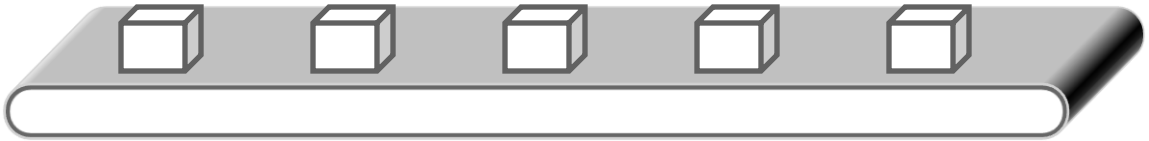
Conference on Utility and Cloud Computing (UCC 2011)
Melbourne, Australia– December 6, 2011

Big Picture: Deploy applications on resources

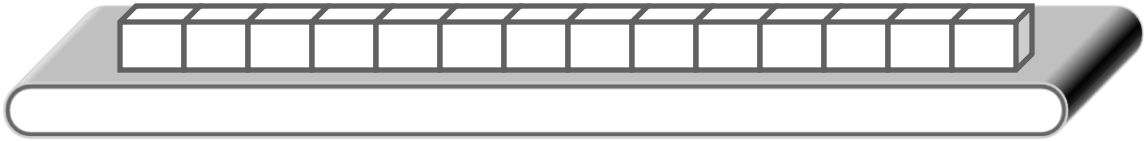


Motivation: Queuing systems

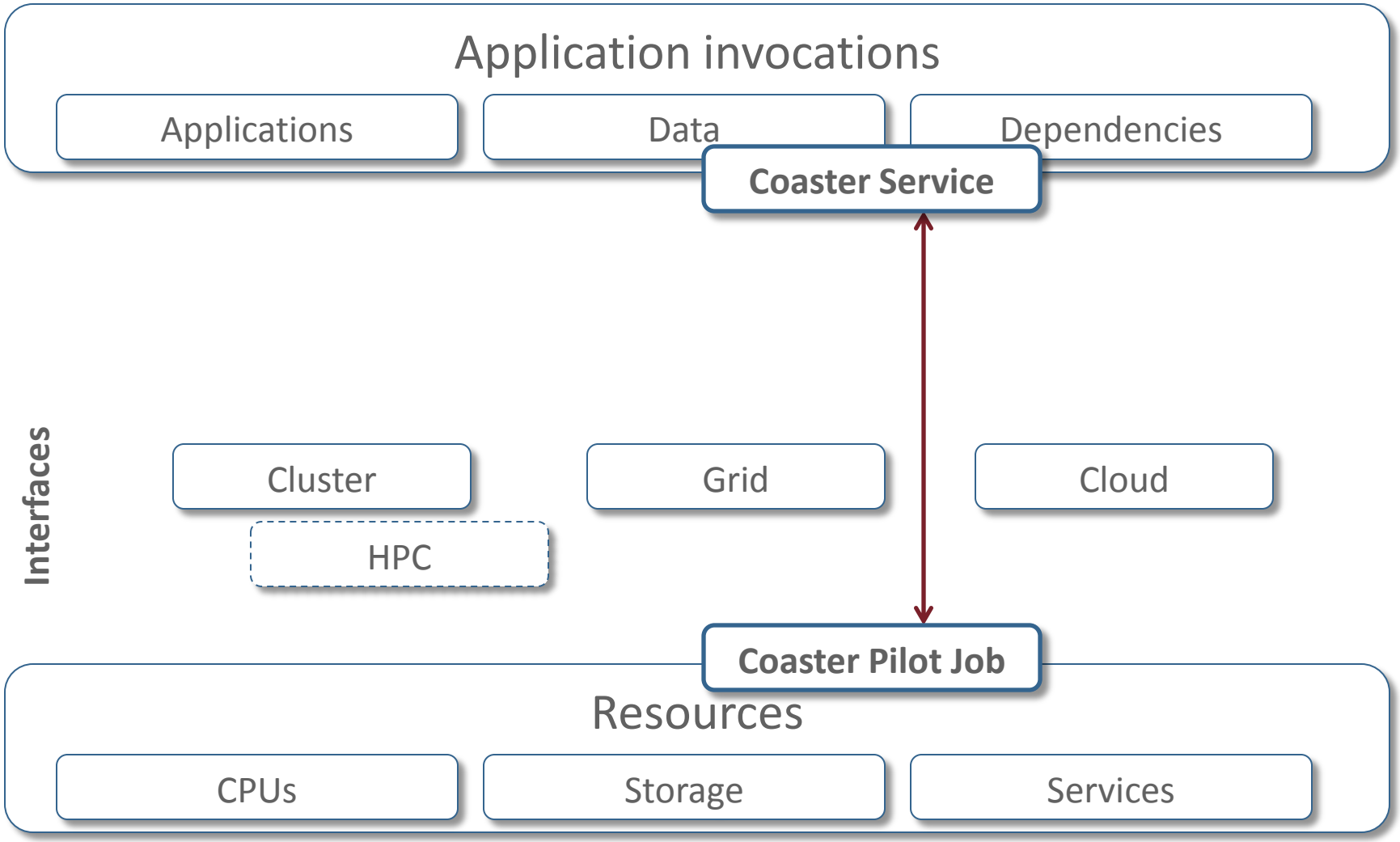
- What we have (PBS, SGE)



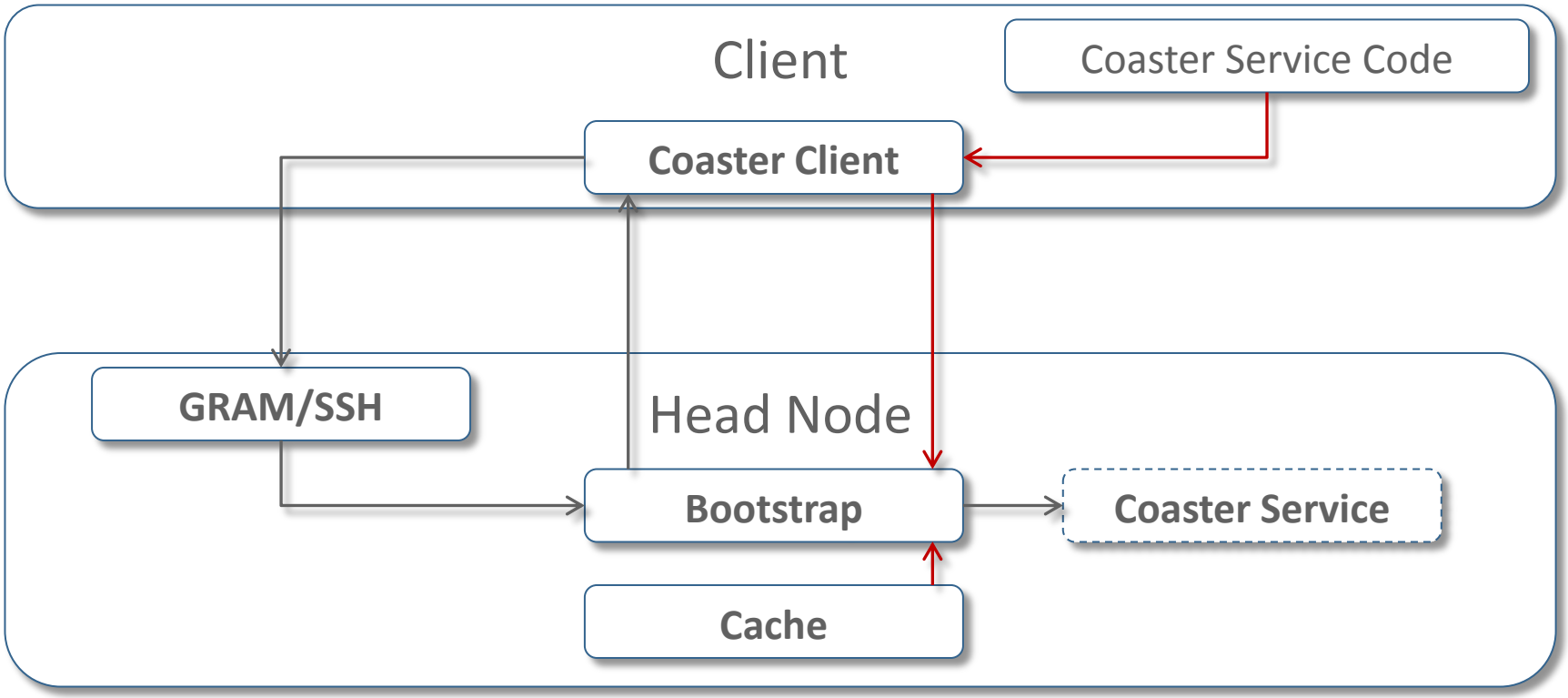
- What we would like



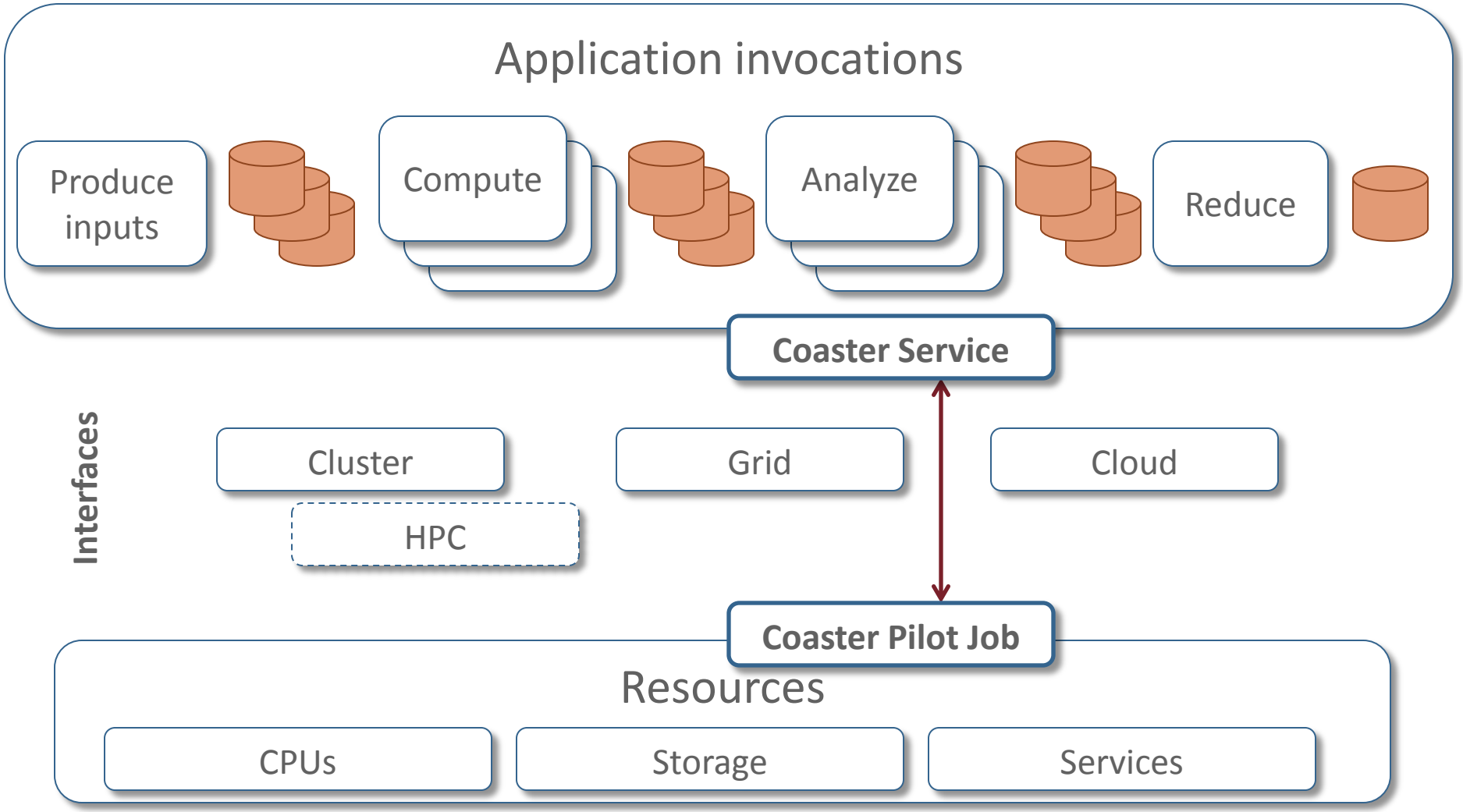
Big Picture: Reusable service components



Big Picture: No hands operation

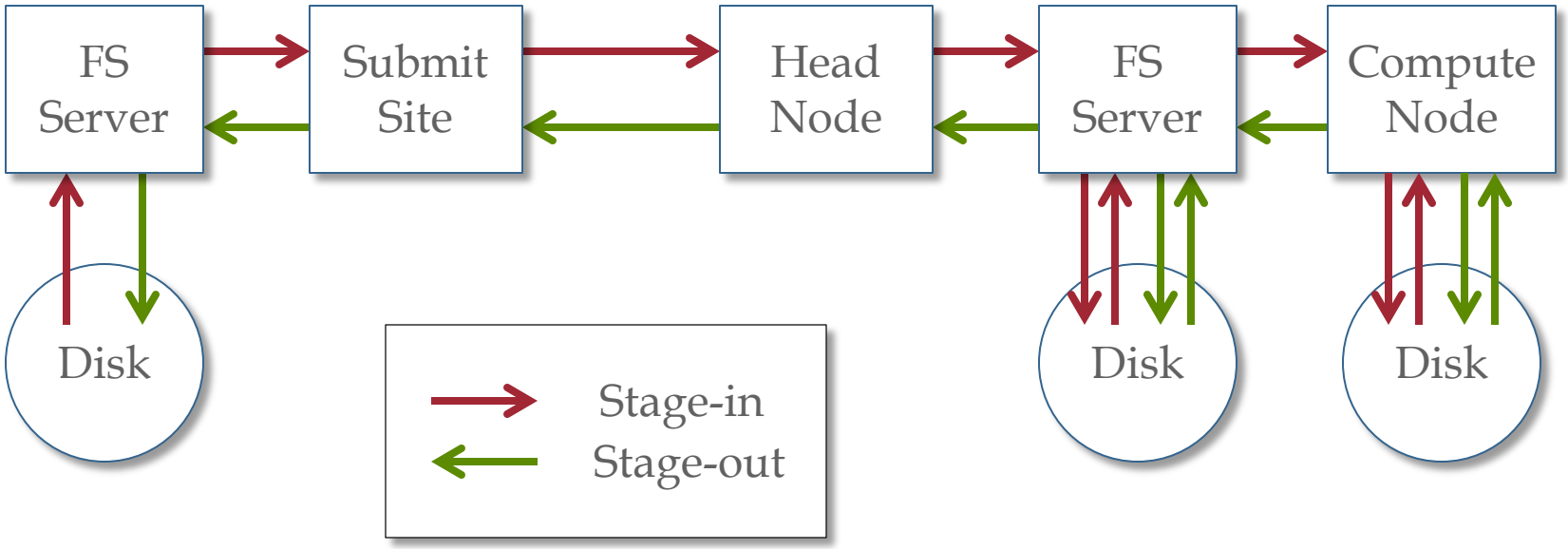


Big Picture: Manage scientific applications



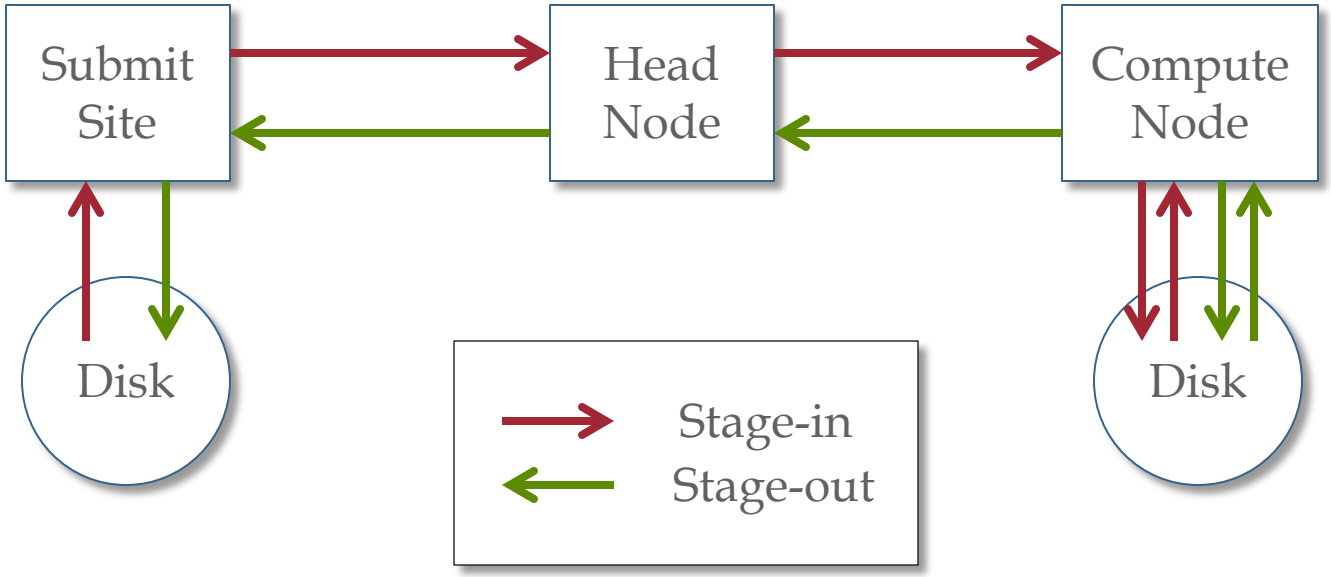
Motivation: File system use on Grids

- Typical file path when using GRAM
- For a stage-in, a file is read 3 times from disk, written 2 times to disk, and goes 4 times through some network
- Assumption: it's more efficient to copy data to compute node local storage than running a job directly on a shared FS.



Motivation: … can be made more efficient

- 2 disk reads, one write, and 2 times through network
- Assumption: compute node has no outside world access, otherwise the head node can be bypassed

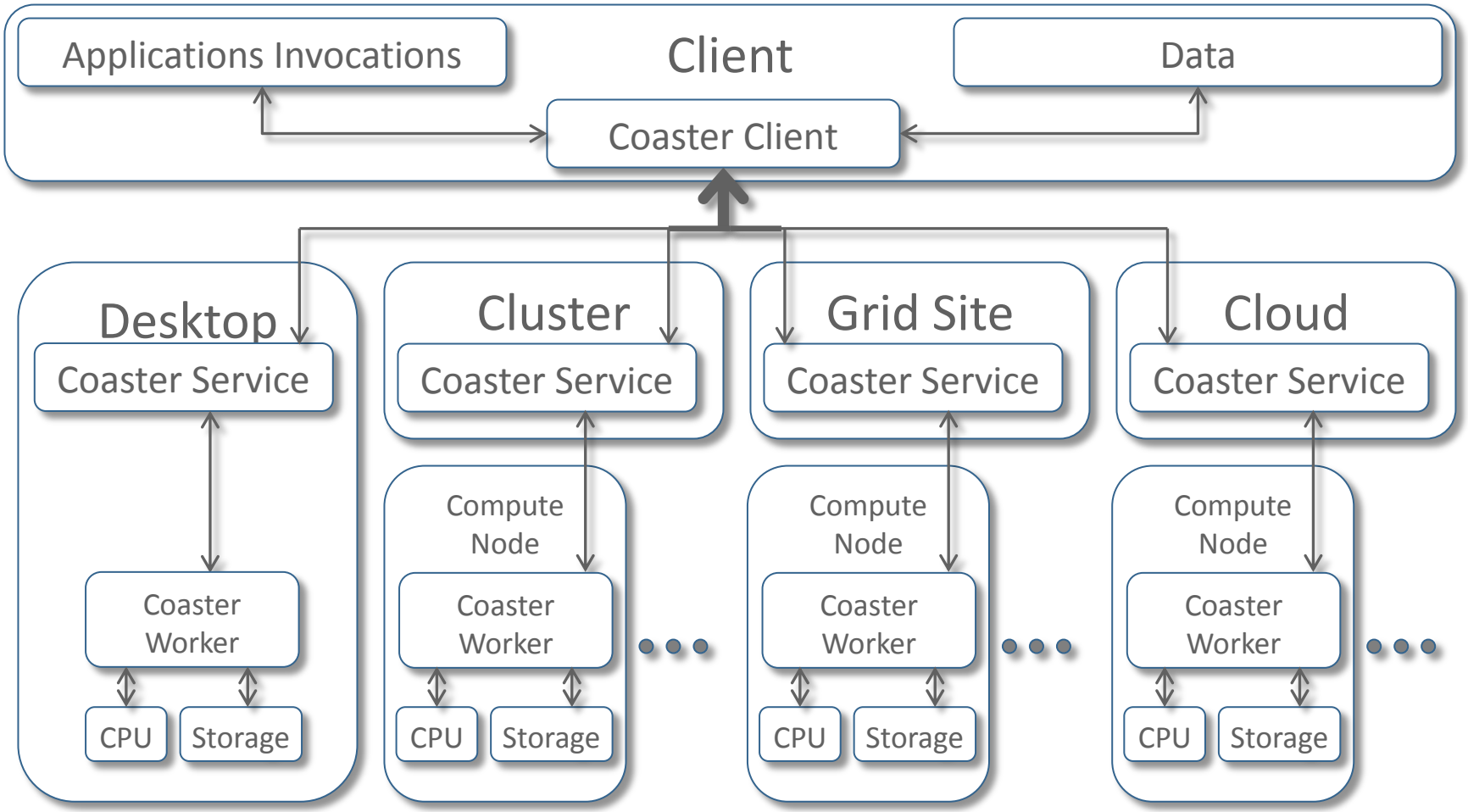


Big Picture: Make these resources *uniform*

Enable high portability

- Application-level
  - Task execution
  - Data access and transfer
- Infrastructure-level
  - Pilot job management
  - Configuration management
- Enable automation of configuration

Big Picture: Create a live pathway between client side and compute nodes



Outline

- Overview of scientific scripting with Swift
- Description of Coasters features
- Application use case: protein docking
- Data transfer methods in Coasters
- Deployment on Amazon EC2 with Globus Provisioning

Big Picture: Enable complex application logic

Not just MapReduce!

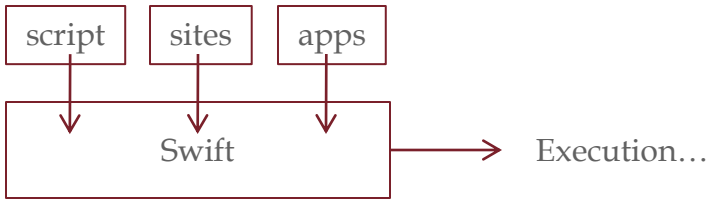
- Iteration
- Typed data
- Structures, arrays
- Automatic dataflow concurrency
- Functions, external applications
- External file data
- Site-specific configuration

Scientific scripting - SwiftScript

- Support file/task model directly in the language

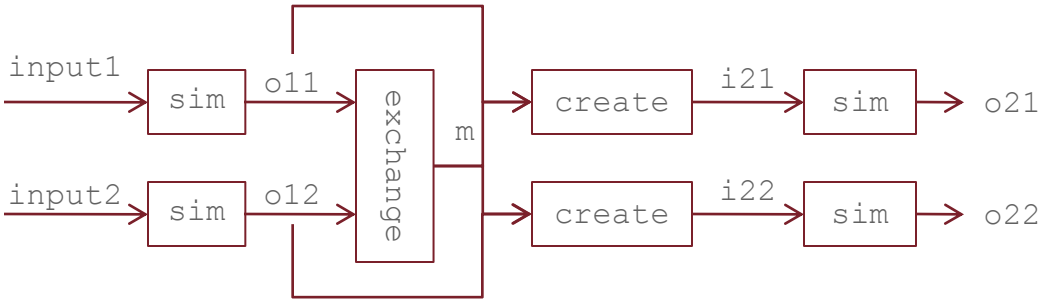
```
app (file output) sim(file input) {
   namd2 @input @output
}
```

- Separate application script from site configuration details



- Provide natural concurrency through automatic data flow analysis and task scheduling

```
file o11 = sim(input1);
file o12 = sim(input2);
file m   = exchange(o11, o12);
file i21 = create(o11, m);
file o21 = sim(i21);
...
```



- Support scientific data sets in the language through language constructs such as structs, arrays, mappers, etc.

Swift features

- Data types

```
string s = "hello world";
int i = 4;
int A[];
```

- Mapped data types

```
type image;
image file1<"snapshot.jpg">;
```

- Conventional expressions

```
if (x == 3) {
    y = x+2;
    s = @strcat("y: ", y);
}
```

- Structured data

```
image  A[]<array_mapper…>;
```

- Loops

```
foreach f,i in A {
    B[i] = convert(A[i]);
}
```

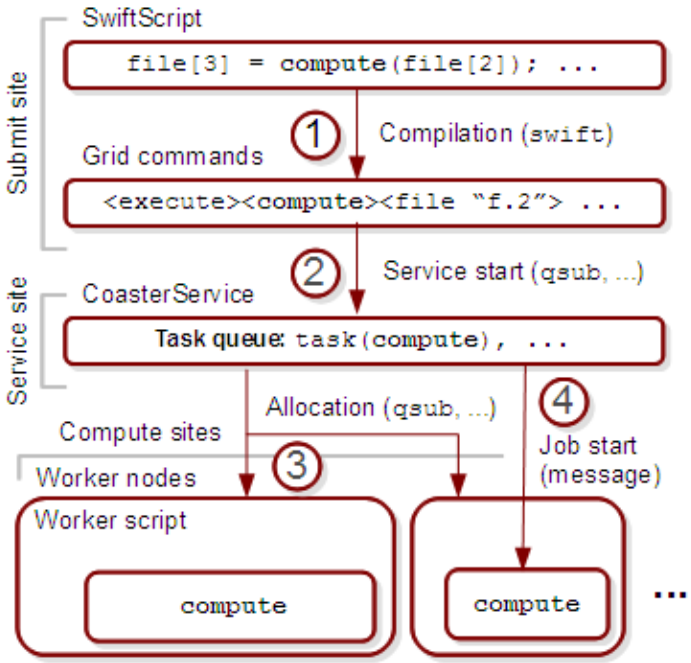
- Data flow

```
analyze(B[0], B[1]);
analyze(B[2], B[3]);
```

Swift: A language for distributed parallel scripting, J. Parallel Computing, 2011

Execution infrastructure - Coasters

- Coasters: a high task rate execution provider
  - Automatically deploys worker agents to resources with respect to user task queues and available resources
  - Implements the Java CoG provider interfaces for compatibility with Swift and other software
  - Currently runs on clusters, grids, and HPC systems
  - Can move data along with task submission
  - Contains a "block" abstraction to manage allocations containing large numbers of CPUs



Infrastructure: Execution/data abstractions

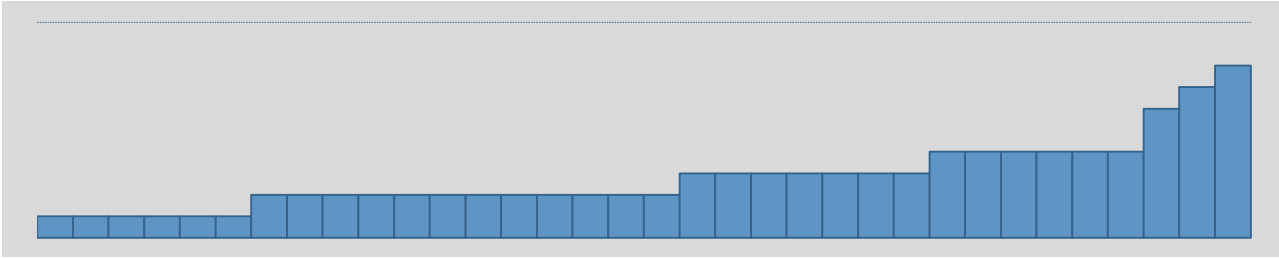
- We need to submit jobs and move data to/from various resources
- Coasters is implemented to use the Java CoG Kit providers
- Coasters implements the Java CoG abstractions
- **Java CoG supports:** Local execution, PBS, SGE, SSH, Condor, GT, Cobalt
- **Thus, it runs on:** Cray, Blue Gene, OSG, TeraGrid, clusters, Bionimbus, FutureGrid, EC2, etc.
- Coasters can automatically allocate blocks of computation time for use in response to user load: "block queue processing"
- Runs as a service or bundled with Swift execution

Implementation: The job packing problem (I)

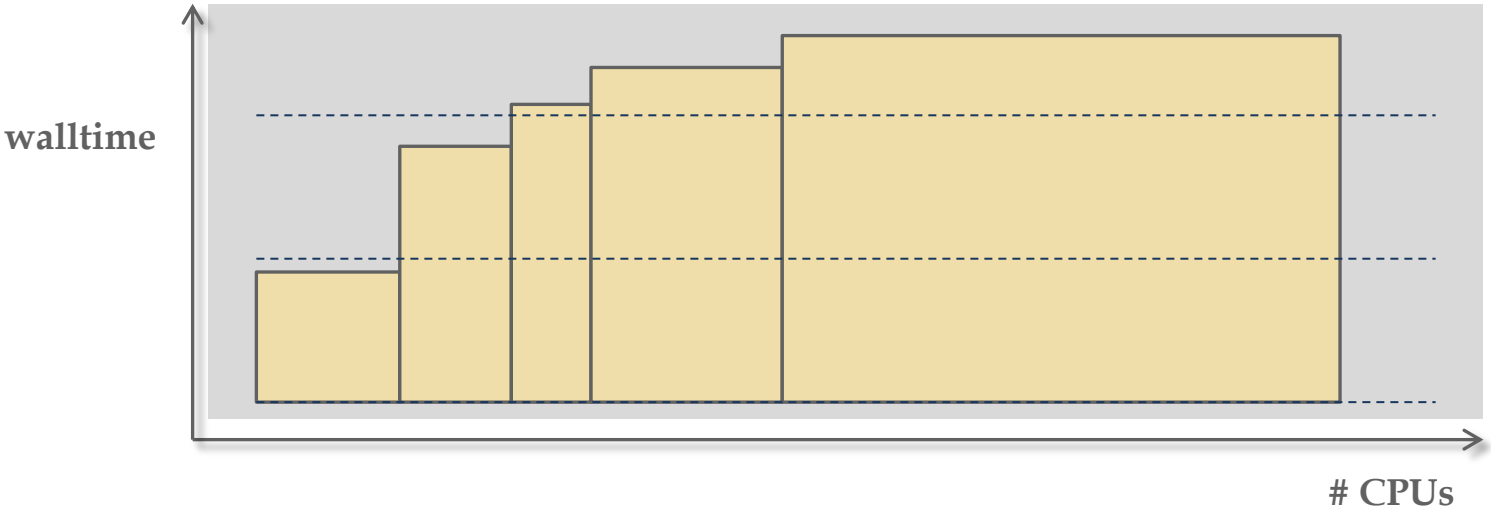
- We need to pack small jobs into larger jobs (blocks)
- Resources have restrictions:
  - Limit on total number of LRM jobs
  - Limit on job times
  - Non-unity granularity (e.g. CPUs can only be allocated in multiples of 16)
- We don't generally know where empty spots in the scheduling are
- We want to fit Coasters pilot jobs into the queue however we can

Implementation: The job packing problem (II) (not to scale)

- Sort incoming jobs based on walltime

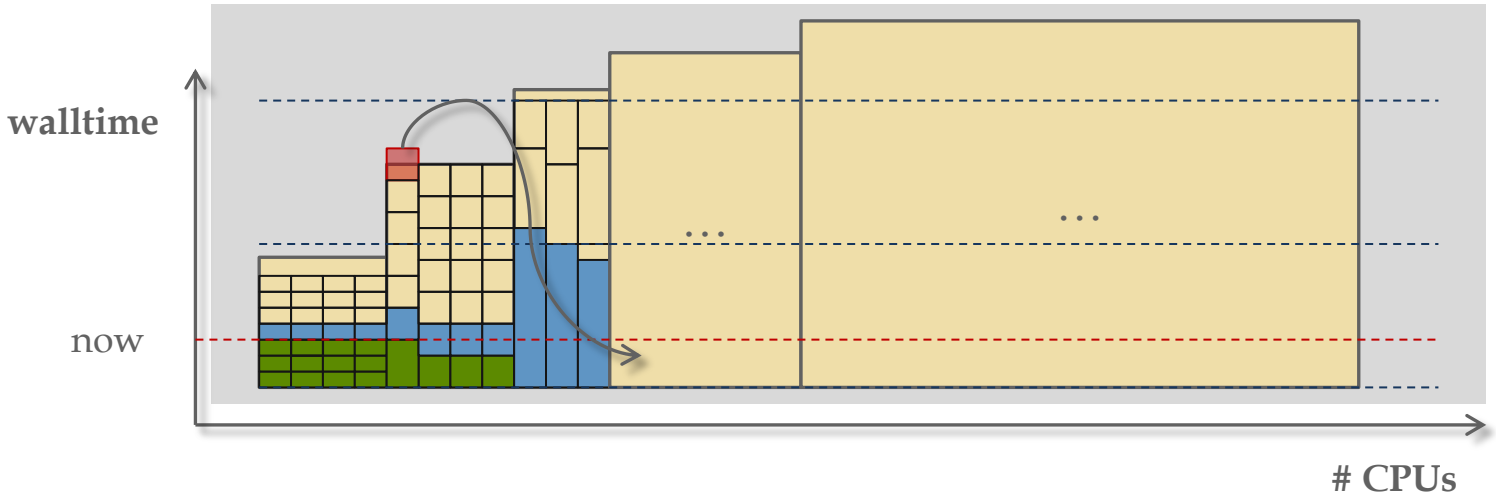


- Partition required space into blocks



Implementation: The job packing problem (II) (also not to scale)

- Commit jobs to blocks and adjust as necessary based on actual walltime



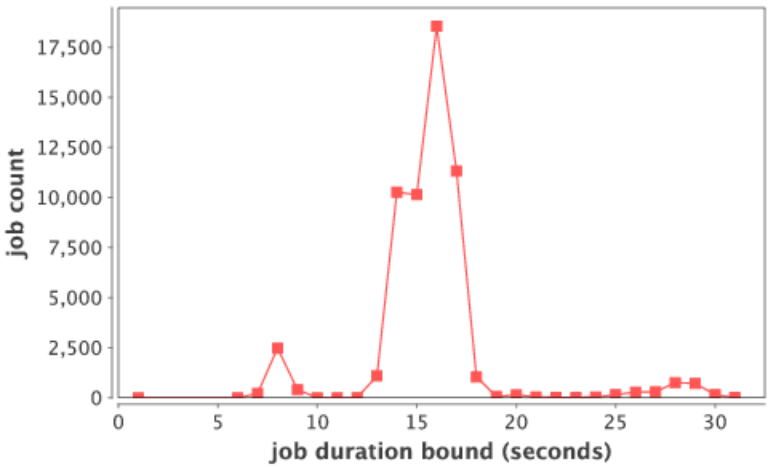
- The actual packing problem is NP-complete
- Solved using a greedy algorithm: always pick the largest job that will fit in a block first

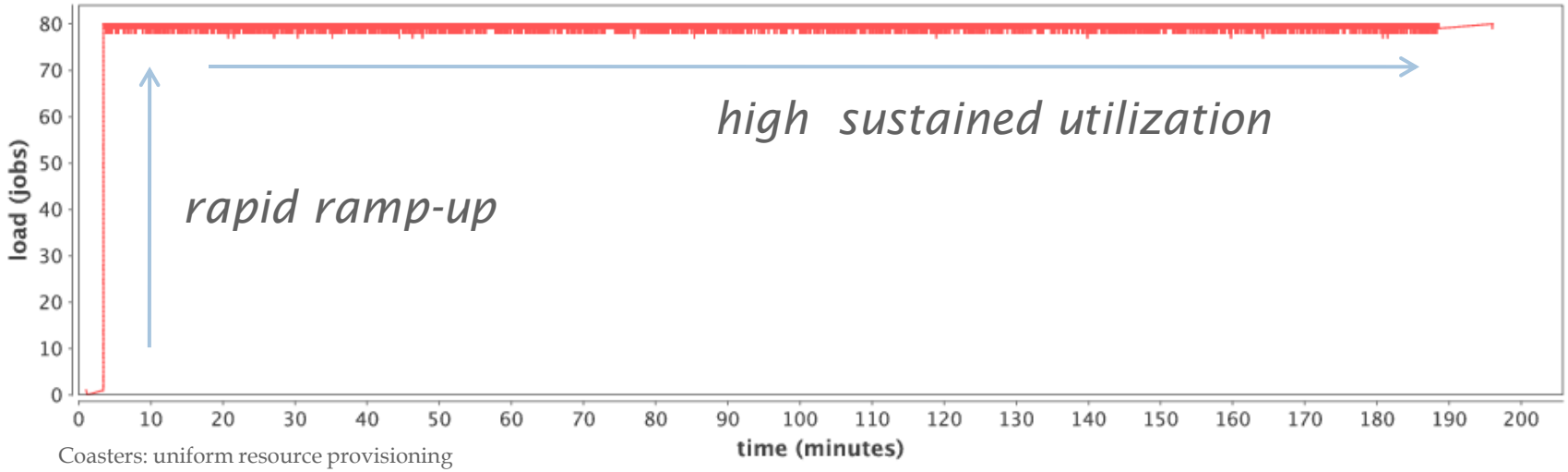
Coasters settings

- "Block" or "Passive" queue processor
- Wall time inputs and customization (overallocation)
- General queue settings
  - Node count specification
  - Spread
- Scheduler-specific queue settings
  - PBS, SGE, Globus settings, etc.
- Swift settings
  - Throttles on job submission, data transfer
  - Allows user to conform to site rules

Application case - ModFTDock

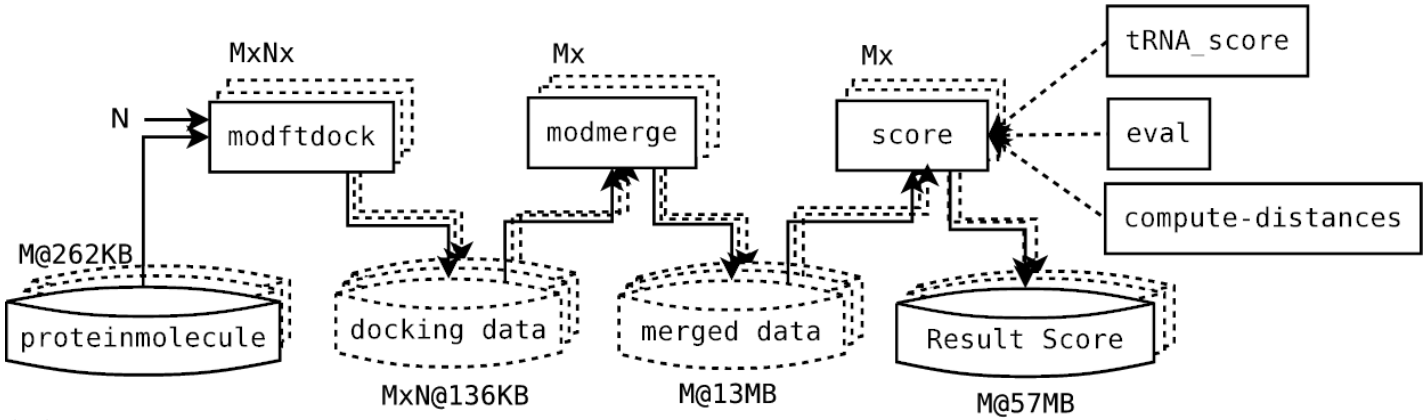
- modFTDock: Novel application to perform protein docking using large batches of sequential tasks
- SwiftScript was rapidly built and deployed on Beagle, the new Cray XE6, and the Bionimbus cloud system
- Bionimbus: easily scaled to 100,000 tasks on 20 nodes, 80 cores in preliminary testing
- Production runs will require similar task quantities, longer tasks, more cores



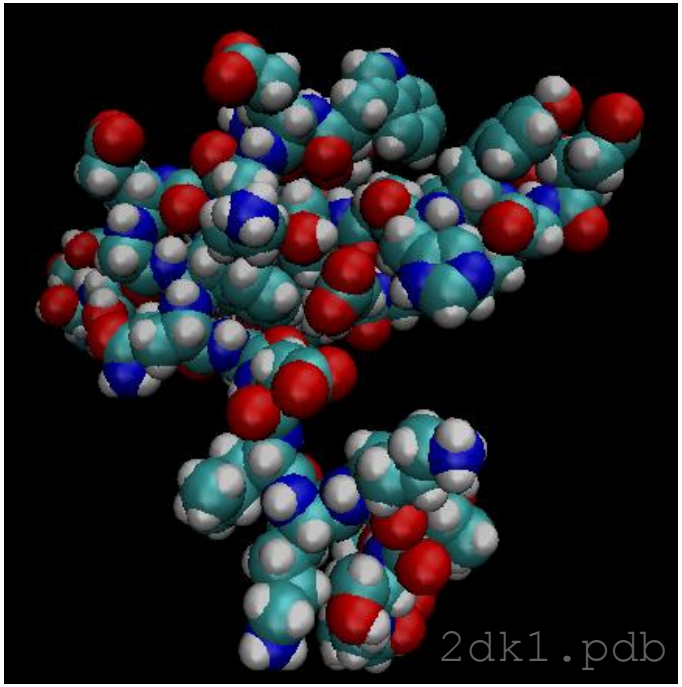


Coasters: uniform resource provisioning

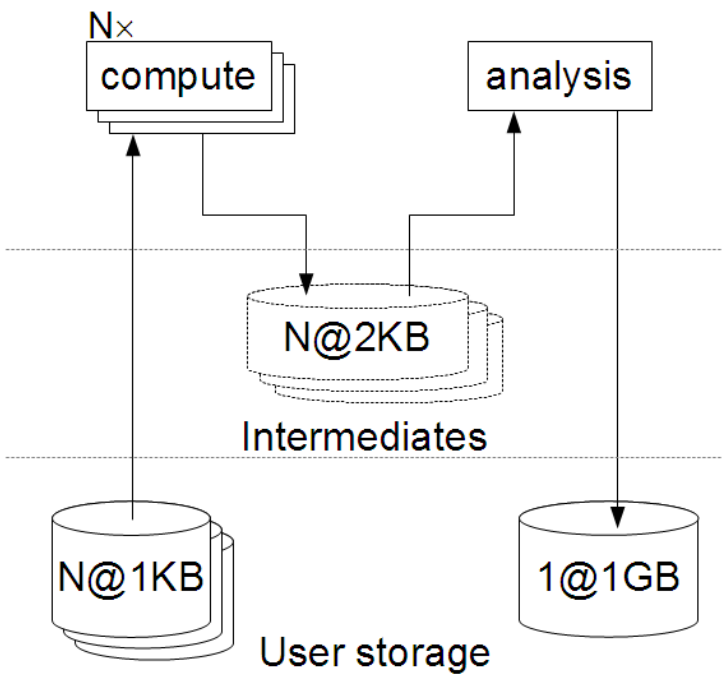
ModFTDock




```
string str_roots[] =
   readData( @arg( "list" ) );
int n = @toint(@arg("n","1"));

foreach str_root in str_roots
{
  string str_file_static =
    @strcat( @arg("in", "input/"), str_root, ".pdb");
  string str_file_mobile = "input/4TRA.pdb";

  file_pdb file_static<single_file_mapper;file=str_file_static >;
  file_pdb file_mobile<single_file_mapper;file=str_file_mobile >;
  file_dat dat_files[]<simple_mapper;
                        padding = 3,
                        location=@arg("out", "output"),
                        prefix=@strcat(str_root, "_"),
                        suffix=".dat">;

  foreach mod_index in [0:n-1]
  {
    string str_modulo = @strcat(mod_index, ":", modulus);
    dat_files[mod_index] =
        do_one_dock(str_root, str_modulo, file_static, file_mobile);
  }
}
```

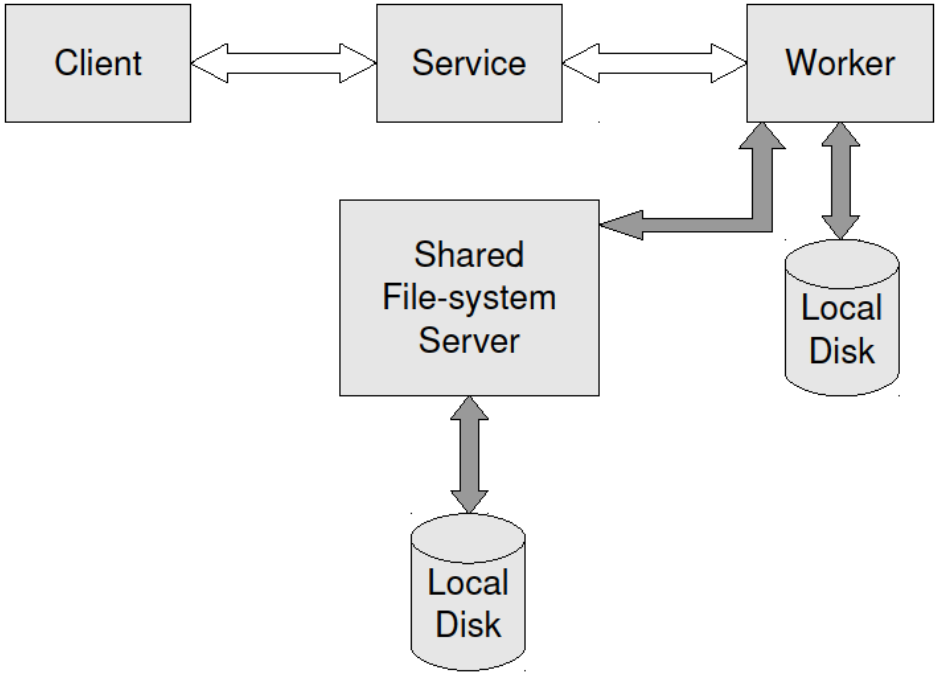


Many tasks: many small files



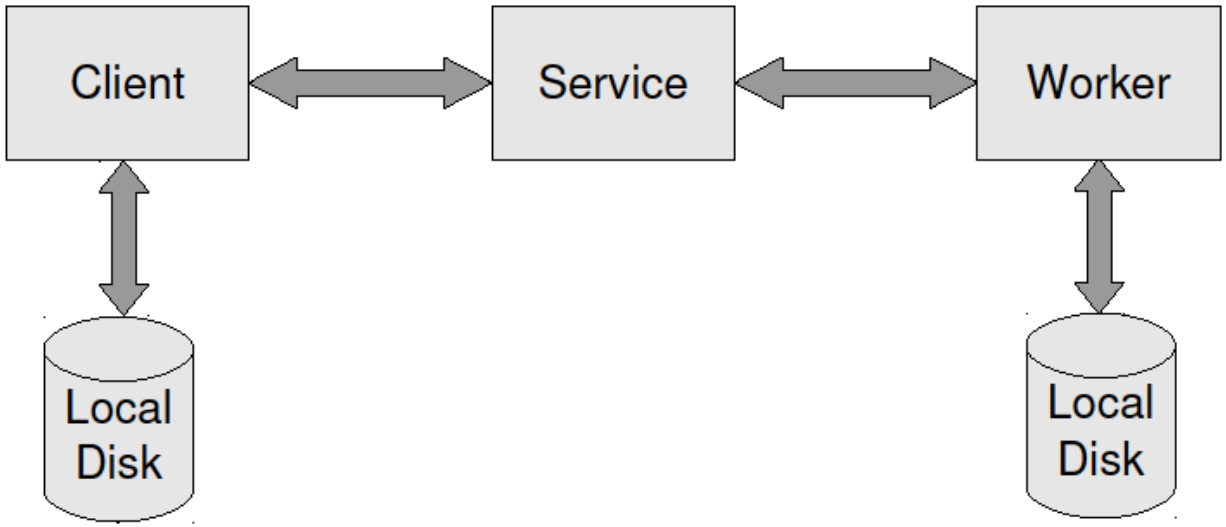
fMRI image analysis

Coasters provider staging



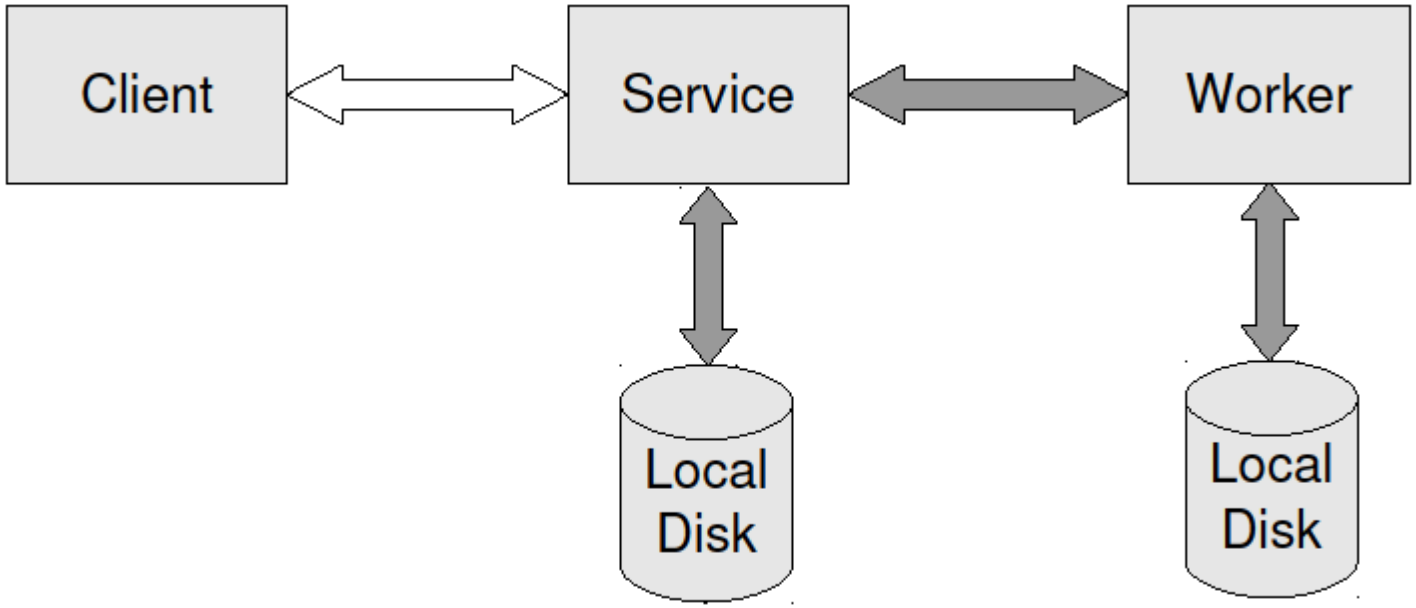
Shared file system

Coasters provider staging



Proxy file transfer

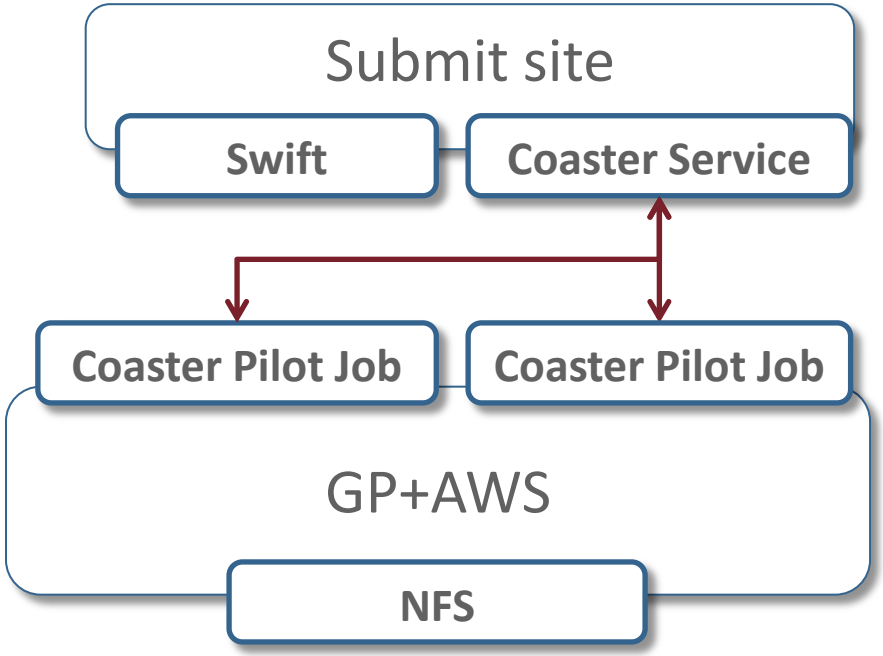
Coasters provider staging



Service file transfer

Use on commercial clouds

- Swift/Coasters is easy to deploy on Globus Provisioning (GP)
- GP provides simple start-up scripts and several other features we may use in the future (NFS, VM image, CHEF)
- Usage:
  0. Authenticate to AWS
  1. `start-coaster-service`
     `gp-instance-create`
     `gp-instance-start`
  2. **`swift`** `myscript.swift …`
     Repeat as necessary
  3. `stop-coaster-service`
     `gp-instance-terminate`



Related Swift/Coasters work…

- Collective Data Management (CDM)
  - Improve support for shared filesystems on distributed resources
  - Make use of specialized, site-specific data movement features
  - Employ caching through the deployment of distributed storage resources on the computation sites
  - Aggregate small file operations into operations on archives, etc.
    Case studies in storage access by loosely coupled petascale applications, Proc. Workshop PDSW at SC, 2009

- Many Parallel-Task Computing (MPTC)
  - Support large batches of small MPI or Global Arrays jobs
  - Multiple scheduler modes - JETS project
    JETS: Language and system support for many-parallel-task computing , Proc. Workshop P2S2 at ICPP, 2011

- ExM: Many-task computing on extreme-scale systems
  - Deploy SwiftScript applications on exascale-generation systems
  - Developing new compiler, new distributed progress management infrastructure, and new global data store

Thanks

- Thanks to the organizers
- Grants:

This research is supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy under Contracts DE-AC02-06CH11357. Work is also supported by DOE with agreement number DE-FC02-06ER25777.

Questions