

# Controlling Ad Hoc Storage Grids with GEMS



Justin M. Wozniak

2007



# Motivation



- User needs:
  - Large scale file space
  - File organization
  - Storage site management
  - Access for existing codes, grids
  - Collaboration across networks
- Available resources:
  - Large, uncontrolled storage network
  - Various access methods

# Solution

---

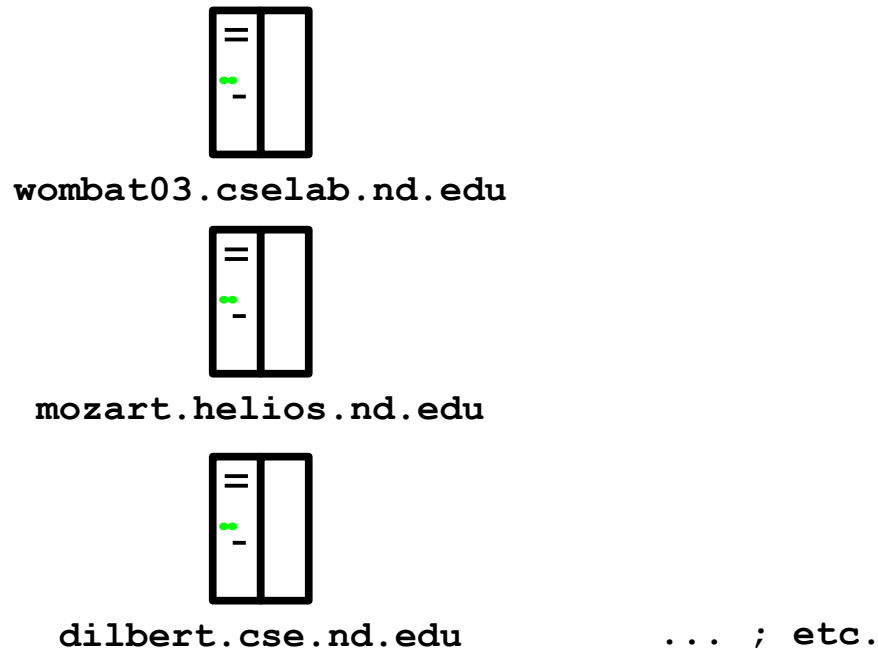
---

- Opportunistic storage:
  - Gain file space along with CPU by running Chirp
  - Build a central file manager (GEMS repository)
  - Automatically manage replicas and disks
  - Seamlessly employ Chirp authentication
  - Integrate networks with abstract authentication
- New resources for users:
  - A central parameter sweep database
  - A parallel, orchestrated Chirp file server network

# Discovery

---

---



- Start with flat index of compute/storage sites
- Each site runs Chirp and is in the Catalog
- Not all sites need Chirp: at least 1 per cluster is ideal

# Repository Installation

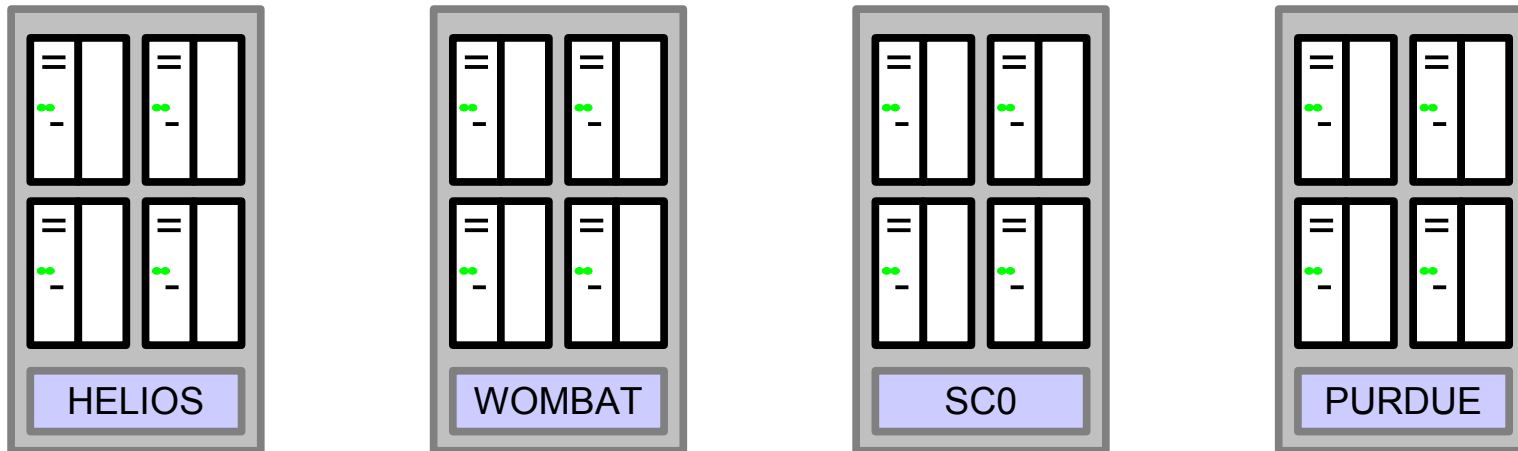


- Drop in the GEMS repository
- GEMS: Grid-Enabled Molecular Simulation
- GEMS is not application-specific anymore

# Organization

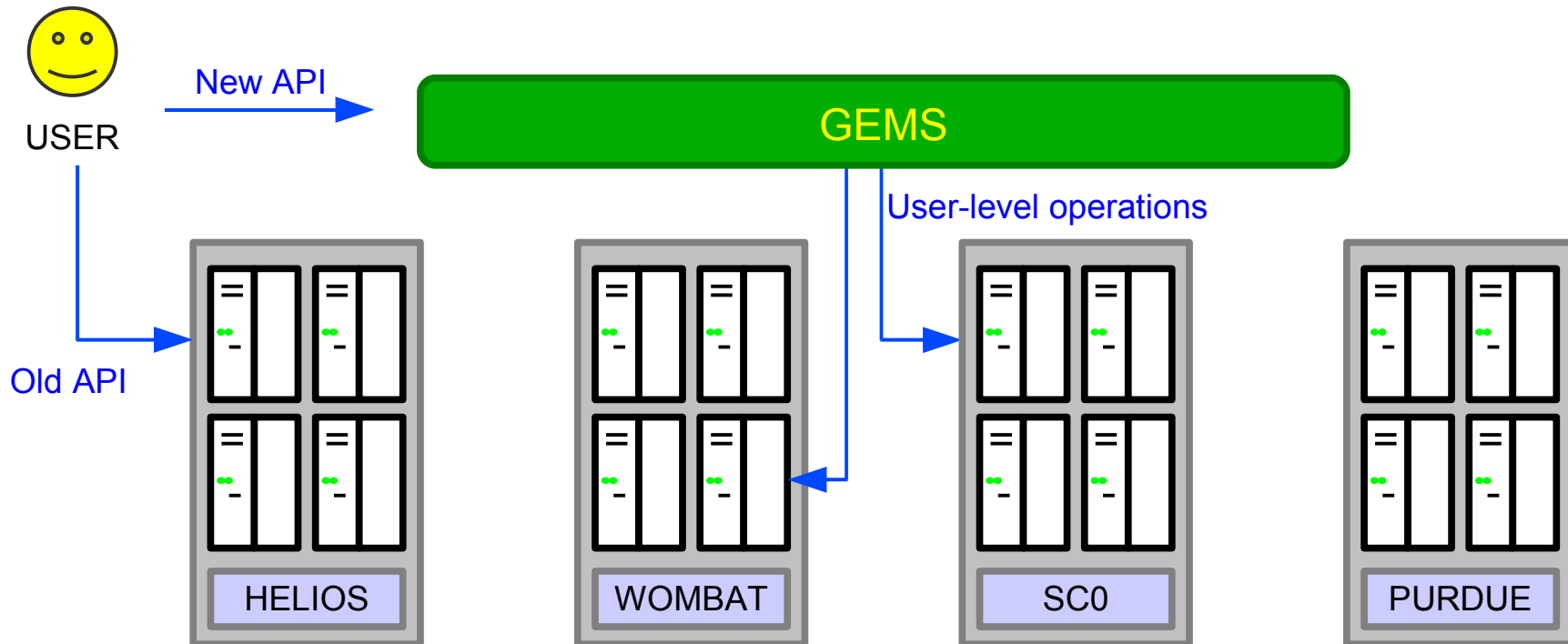
---

---



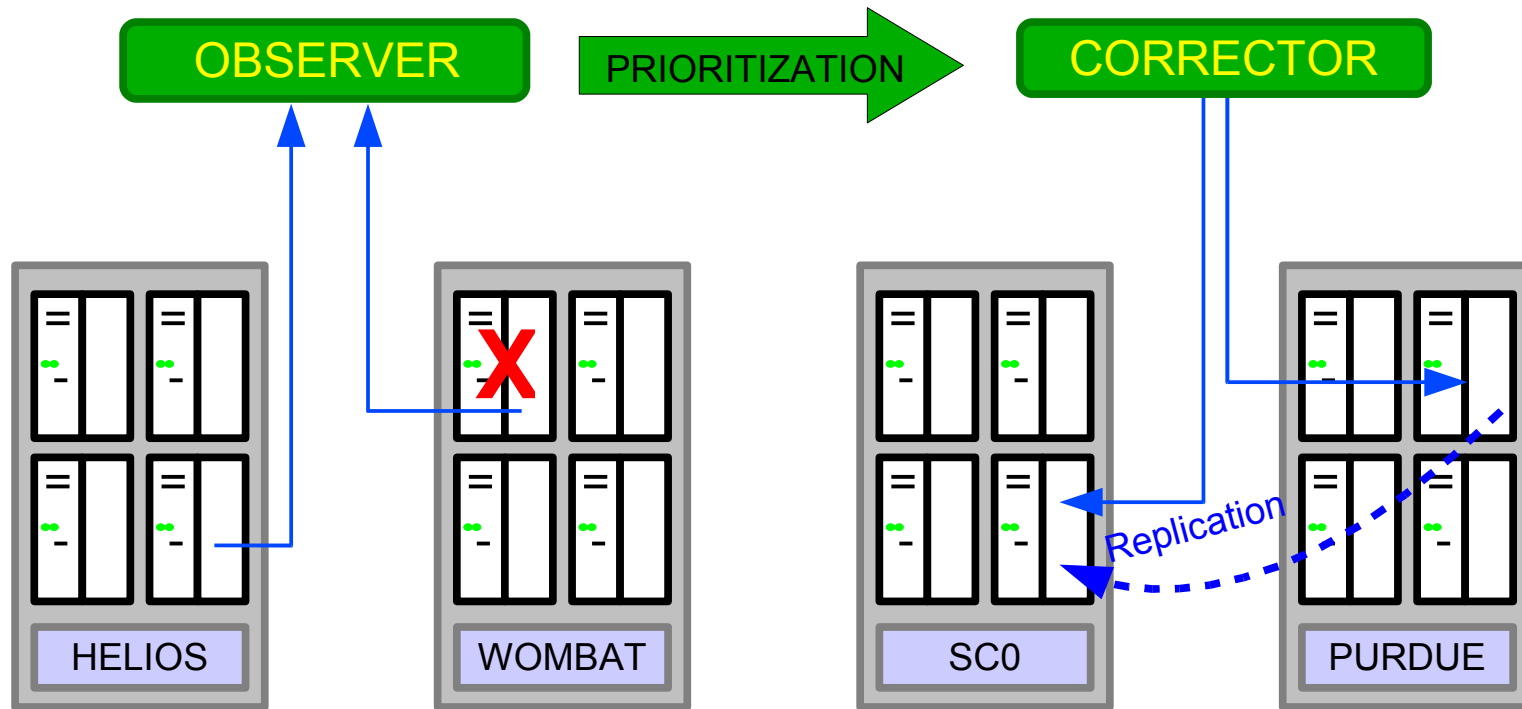
- Build (logical) clusters from index
- Allow users to do the same
- Managed through GEMView (Storage Map)

# Storage Management



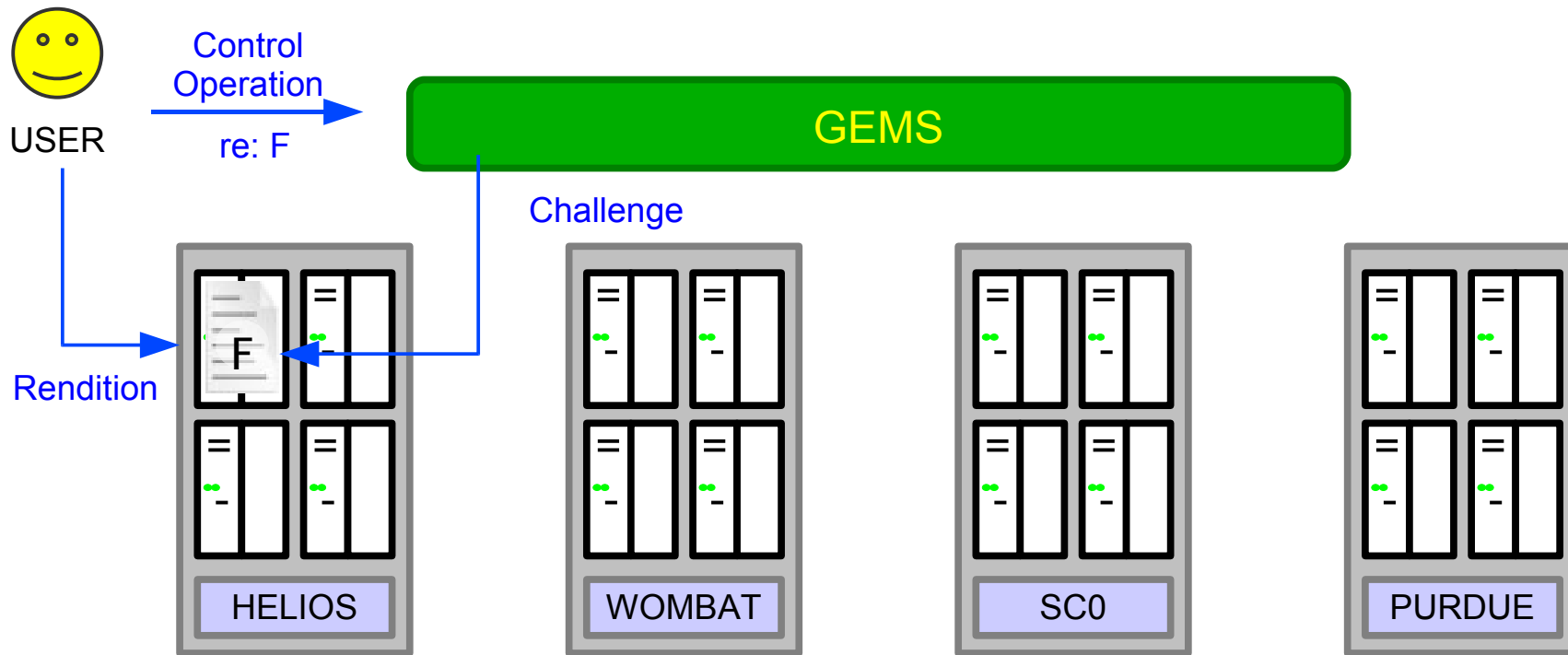
- Allow for insertion, replication, and automatic monitoring
- See GEMSpout, GEMSget, etc.

# Storage Control



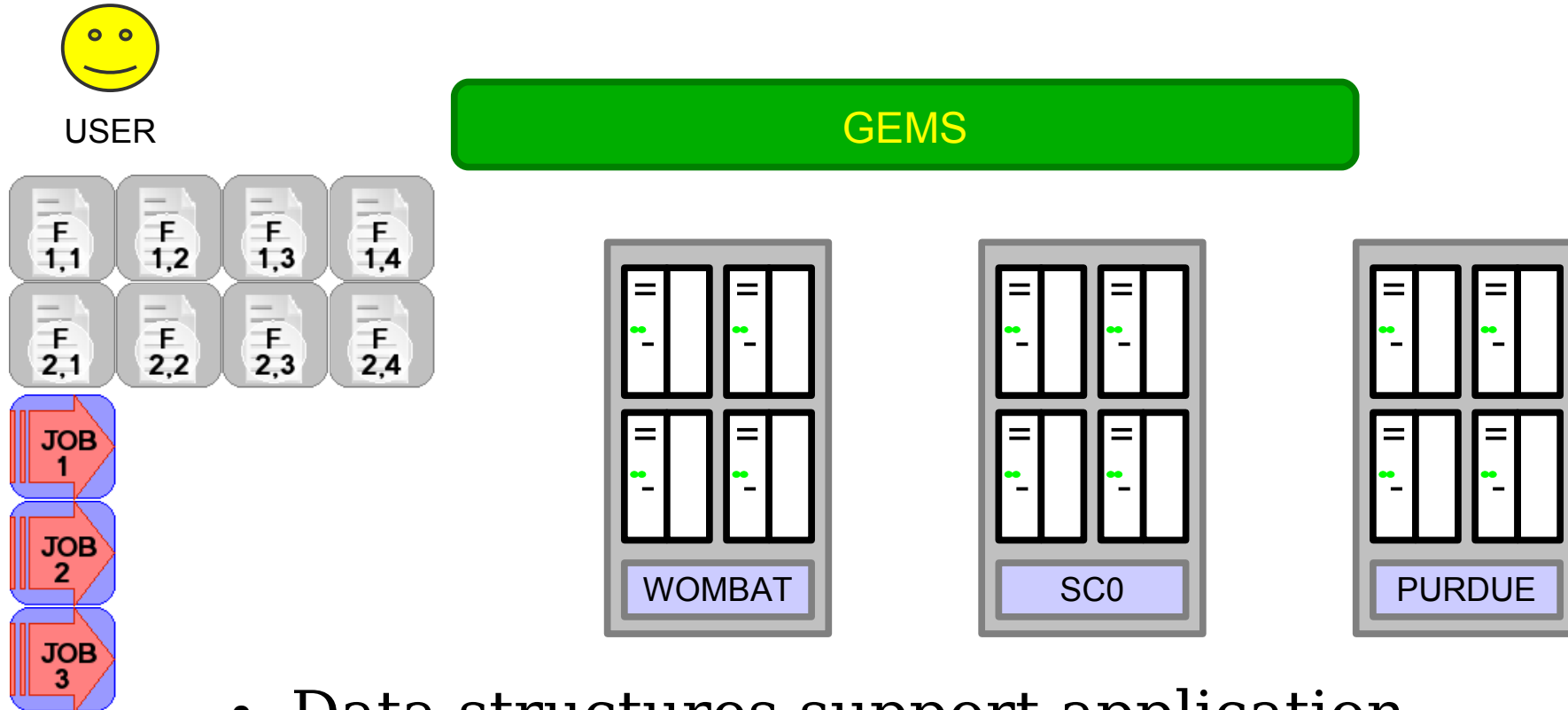
- Automatically detect/correct faults
- Managed by GEMSD

# Access Control



- Existing Chirp access controls enforce “local” operations
- Control operations authenticated by rendition
- No new passwords or global security software!

# Parameterized Storage Organization



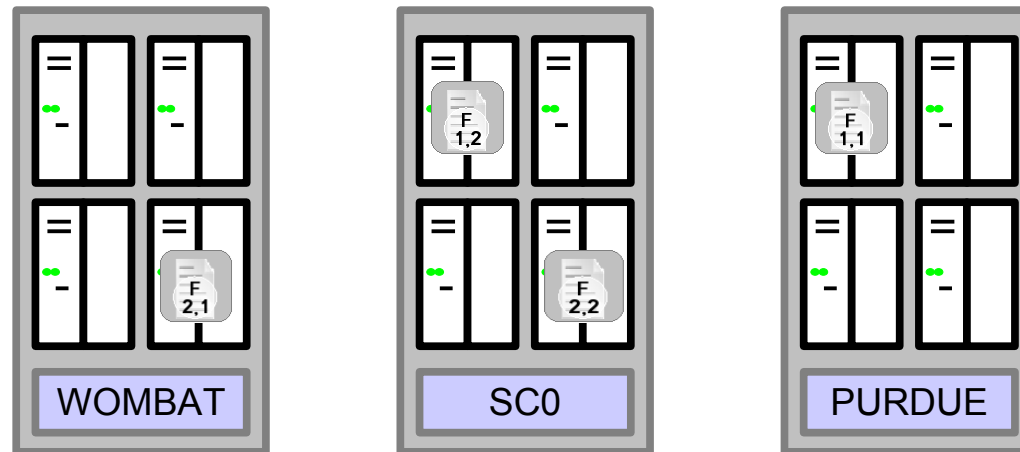
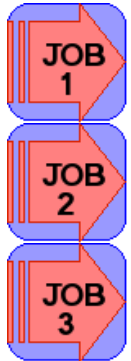
- Data structures support application-specific tagging and searches

# Parameterized Storage Organization



USER

GEMS

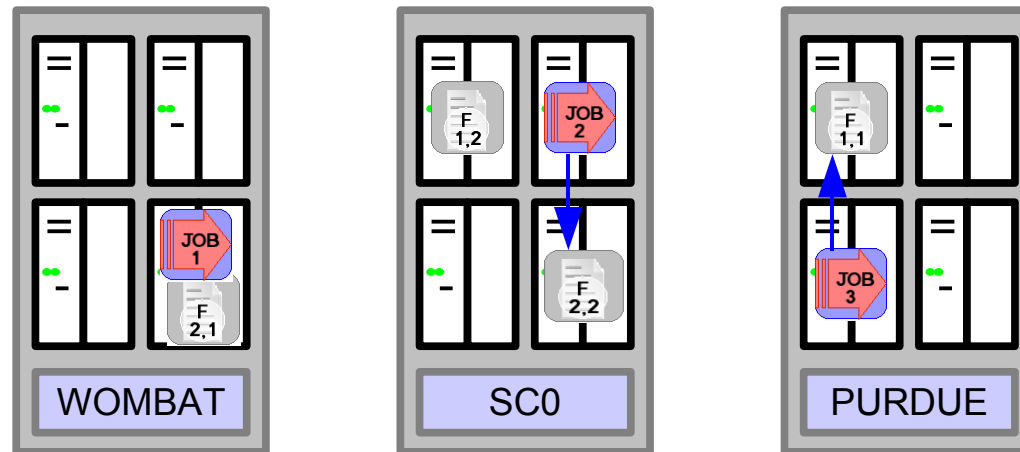


- File placement managed by user-supplied topology information

# Parameterized Storage Organization



USER



- Topology information may be tapped when placing jobs or accessing replicas

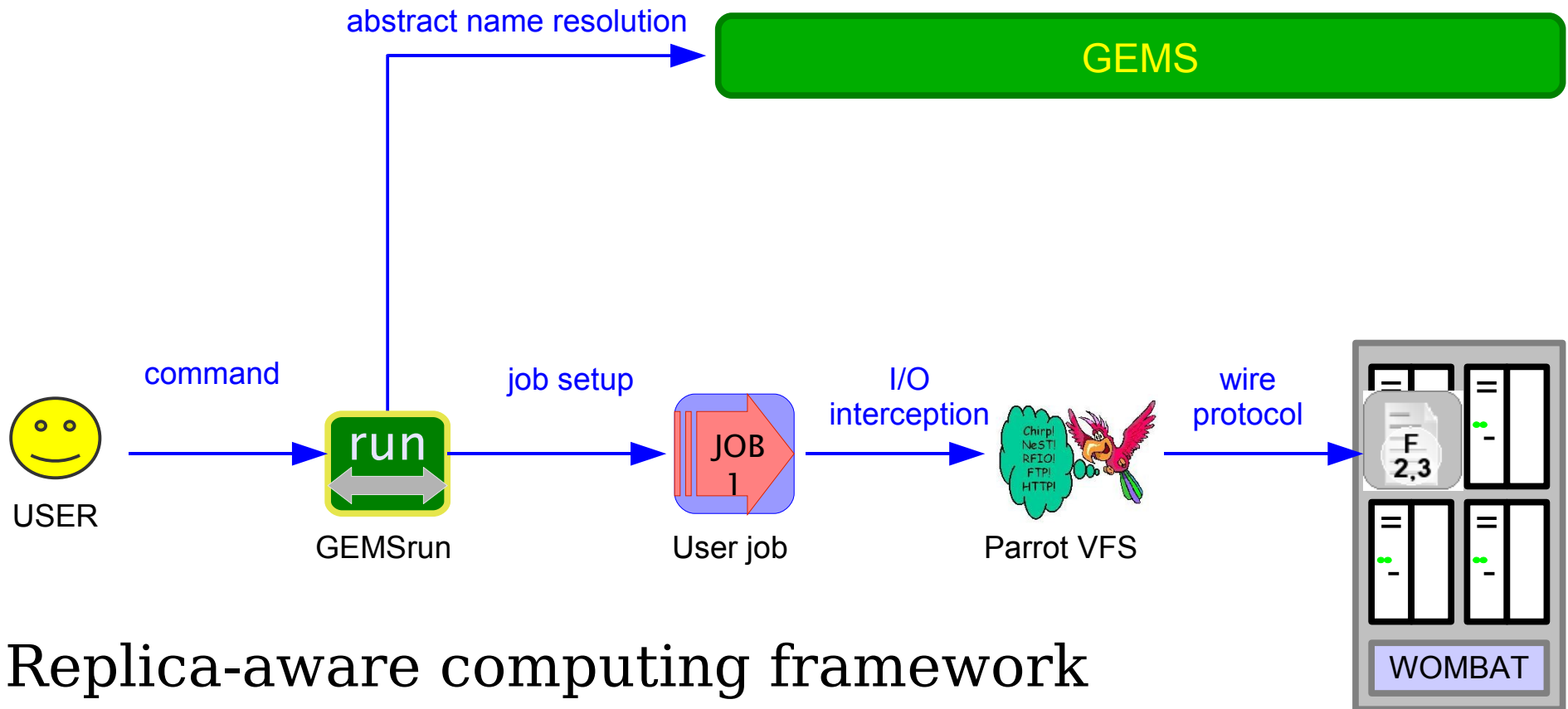
# Graphical User Interface

The screenshot displays the GEMView application window. The title bar reads 'GEMView'. The main interface is divided into several sections:

- Server/Account/View/Settings:** Server: gems.cse.nd.edu, Account: unix:jwozniak, View: Params, Settings: Storage, Access Control.
- Search Mode:** Simple (selected), Config, Full.
- GEMS Search:** A search form with fields for Name (Mr. Demo), Date (2/28/2007), App (Protomol), and Description (Protomol Projects). Below the form are 'Match' and 'Put' buttons.
- Results:** A list of search results under the 'Preview' tab. The results are numbered 1-3 of 3. Each result includes a 'Download' link, a 'Details' link, and a list of files with their sizes. The first result is 'TPS Scripts' with a config key of 1604247299. The second is 'TPS Segment Data' with a config key of 908313744. The third is 'Protomol Source Code' with a config key of 945616186.

• Allows for quick browsing

# Computation *Among* Replicas



- Replica-aware computing framework
- See GEMSSrun

# Replica System Methods

---

---

- Basic methods:
  - Simple puts & gets
  - Replica location
  - Replica access site evaluation
- Streaming methods:
  - Advanced disk space reservation
  - I/O setup
  - I/O translation
- All can be placed in Condor scripts

# Script snippets: Explicit puts & gets

---

---

```
> alias GEMSget='java -cp GEMS.jar GEMS.client.GEMSget'  
> alias GEMSpout='java -cp GEMS.jar GEMS.client.GEMSpout'  
> alias GEMSmatch='java -cp GEMS.jar GEMS.client.GEMSmatch'  
  
> KEY=$( GEMSmatch name=${USER} metadata=400 )  
> GEMSget --config ${KEY}  
  
> perform_user_computation on downloaded files creating data.out  
  
> GEMSpout name=${USER} metadata=10 --file data.out
```

- Send the `GEMS.jar` file along with your job to find, retrieve, and insert data
- Metadata (shown with '=') is completely user-defined
- Replica location is used to speed up and distribute the data movement cost

# Script snippets: Virtual filesystem

---

---

```
> alias GEMSmatch='java -cp GEMS.jar GEMS.client.GEMSmatch'  
> alias GEMStrun='java -cp GEMS.jar GEMS.client.GEMStrun'  
  
> KEY=$( GEMSmatch name=${USER} metadata=400 )  
  
> GEMStrun --input DATA_IN /${KEY}/data.in  
          name=${USER} metadata=10 --output DATA_OUT data.out  
          --local --exec computation DATA_IN DATA_OUT
```

- GEMS integrates with the Parrot VFS
- Transformations result in new GEMS records

# Script Snippets: Streaming output

---

---

- Archive creation

*obtain sink:*

```
> SINK=$( GEMSreserve -size 100000000 )
```

*pipe command output:*

```
> alias stream='java -cp GEMS.jar Chirp.io.ChirpOutputStream'
```

```
> tar c dir | stream ${SINK}/archive.tar
```

```
> GEMScommit --config ${SINK} --file archive.tar
```

- Simple shell setup for streaming replica system I/O
- Java solution is portable

# Script Snippets: Notification

---

---

- Workflow element creation

*wait until dependency is satisfied:*

```
> KEY=$( GEMSnotify name=${USER} fileset=setup )
```

*GEMSnotify waits until match is ready- then returns key:*

```
> GEMSget --config ${KEY}
```

*Now you can operate on the downloaded files*

- Simple shell setup for dependent execution
- Inter-job communication can thus be synchronized with respect to the tuple space state

# Summary

---

---

- Analogies for GEMS:
  - A drop-in tuple space for Condor
  - A replica selection system for the grid
  - A resource integration toolkit
  - A data-driven computation launcher
- Implements a runtime repository for distributed computation and parallel data movement
- ***With*** automatic file auditing and replication
- See the manual for more information

# Acknowledgments



- Collaborators:
  - Paul Brenner
  - Santanu Chatterjee
  - Douglas Thain
  - Aaron Striegel
  - Jesus Izaguirre
- NSF DBI-0450067