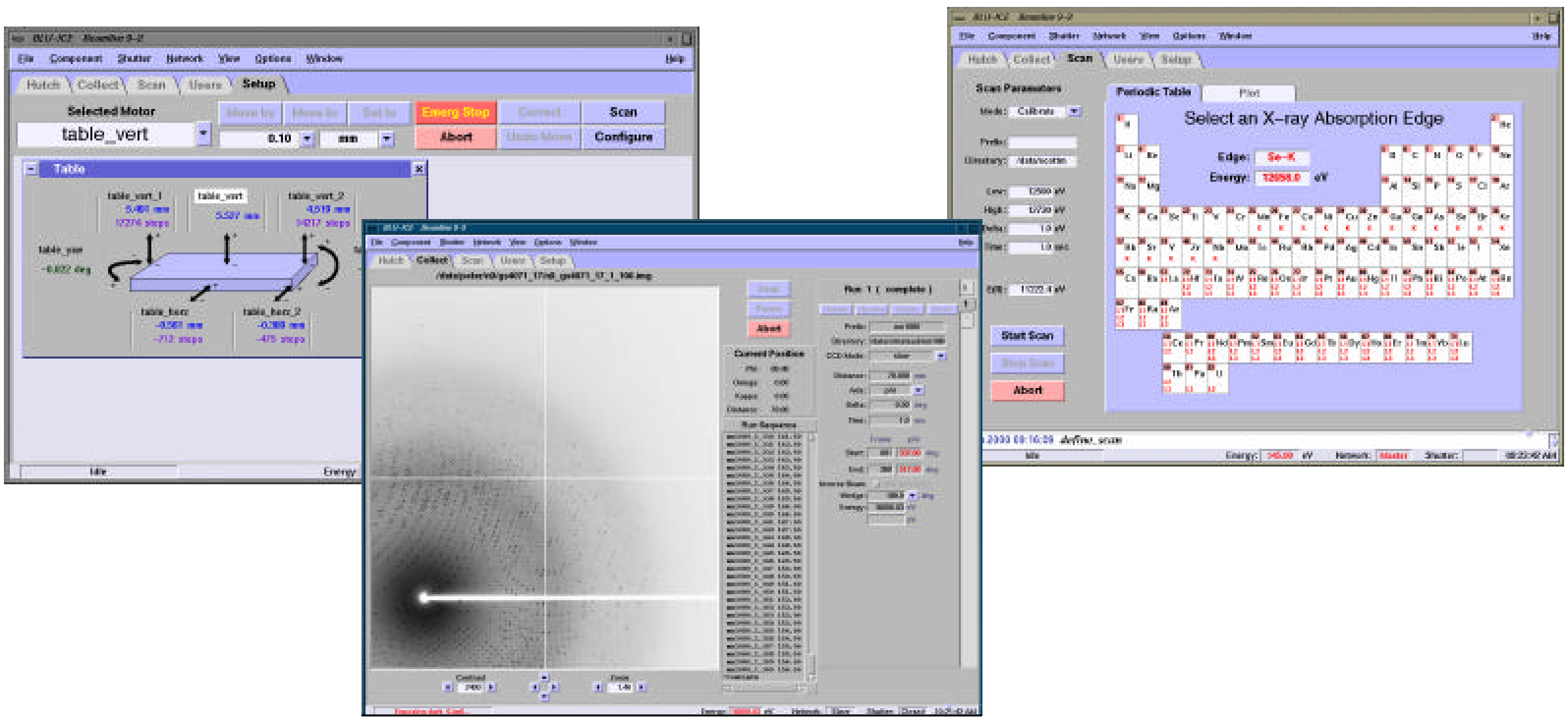
Structural Molecular Biology

Software Development Workshop

September 18th, 2000

Collaborative Data Collection with BLU-ICE



Timothy McPhillips

Stanford Synchrotron Radiation Laboratory

Overview

Architecture of the Distibuted Control System

- Life cycle of a DCS message.
- Collaboration features.

Hybrid Software Development Strategy

- XOS library for platform-independent C/C++ code.
- Tcl/Tk for platform independent GUI development.

Users’ Favorite Features in BLU-ICE

- GUI focuses on the experiment.
- Simple data collection set up for advanced MAD experiments.

Scripted Devices

- All operations may be scripted in BLU-ICE.
- Same scripts may be run within the DCS server.
- Supports virtual motors with parent-child relationships.

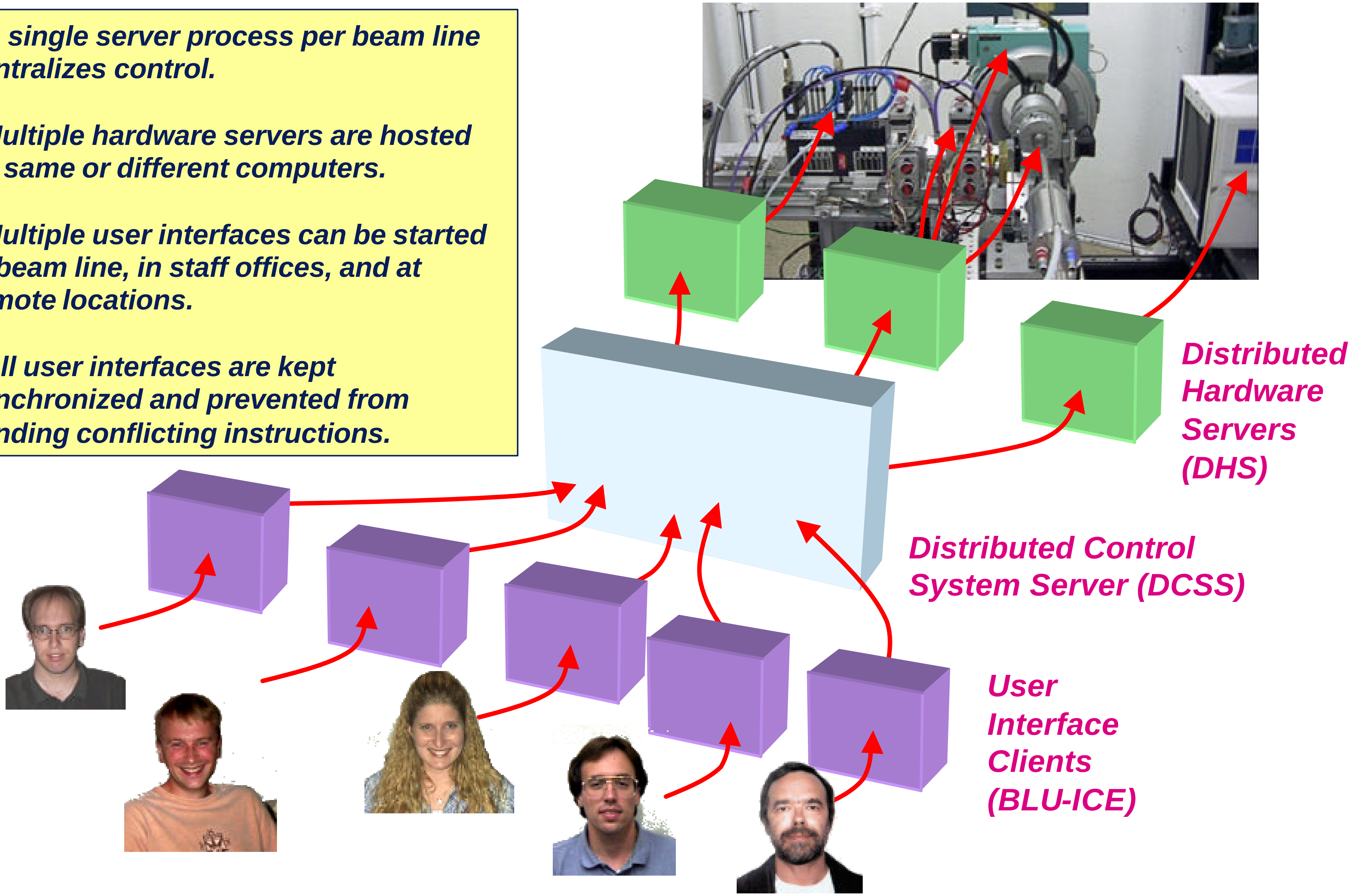
Architecture of the Distributed Control System

- *A single server process per beam line centralizes control.*

- *Multiple hardware servers are hosted on same or different computers.*

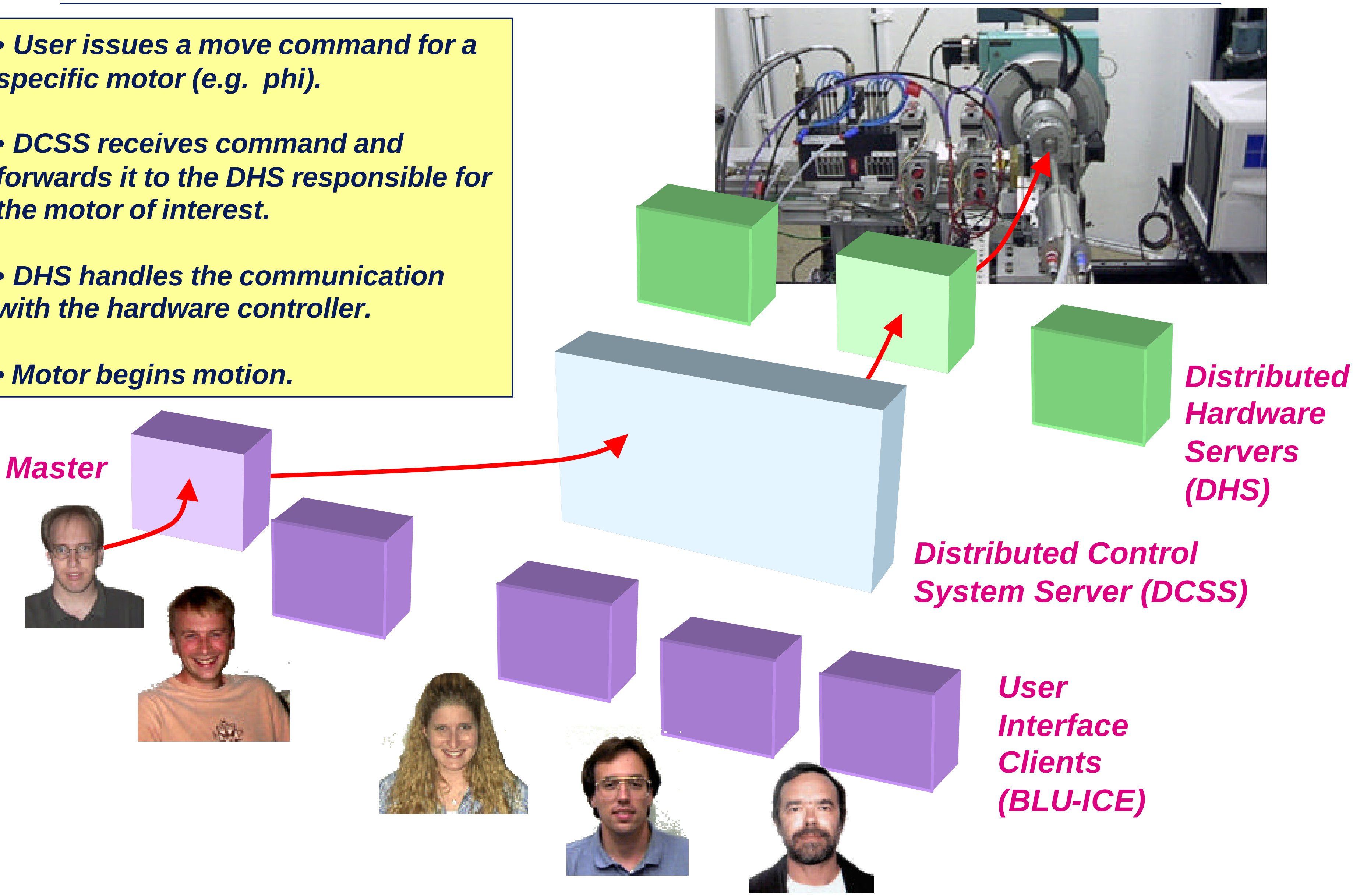
- *Multiple user interfaces can be started at beam line, in staff offices, and at remote locations.*

- *All user interfaces are kept synchronized and prevented from sending conflicting instructions.*



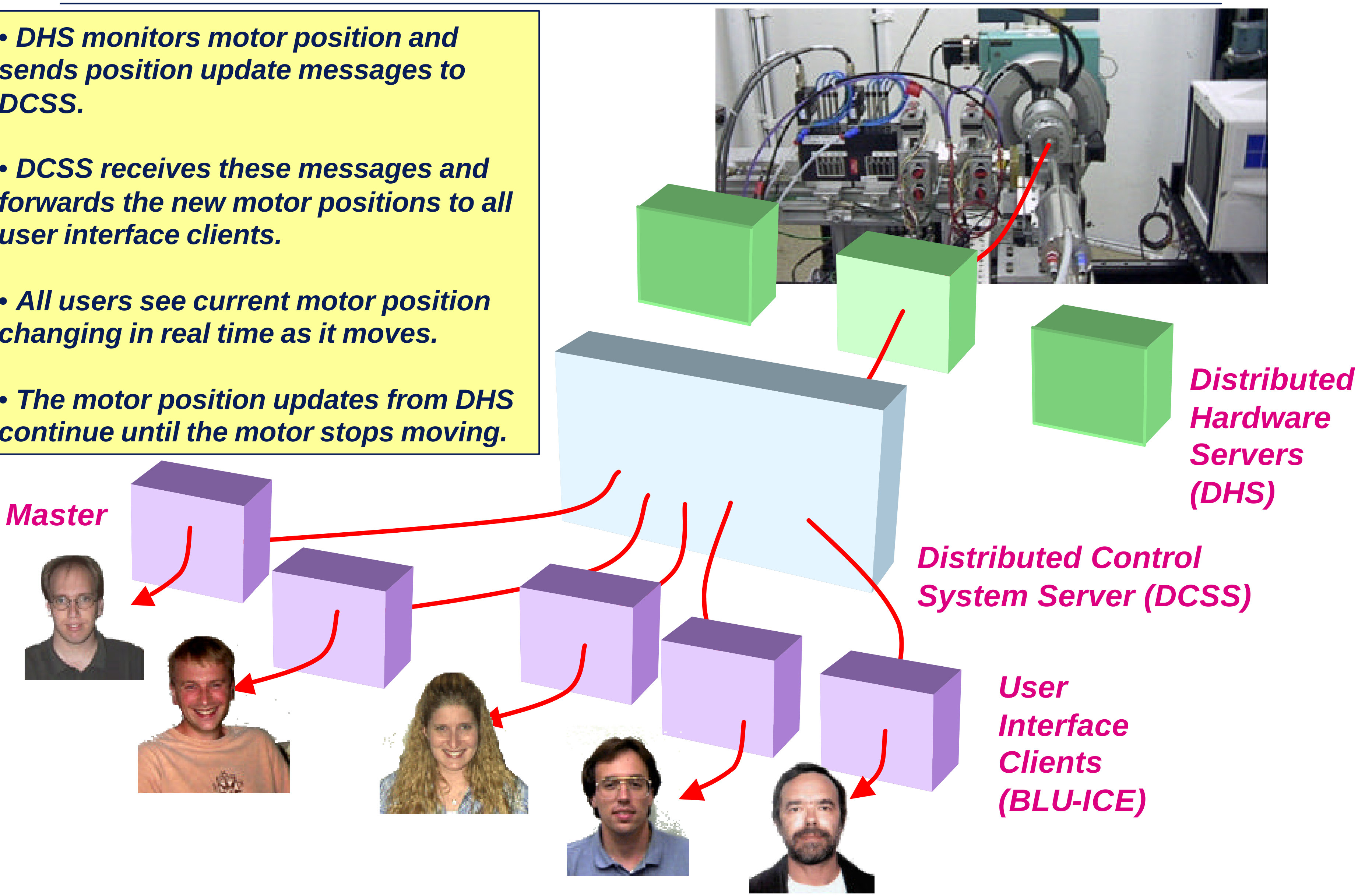
Life Cycle of a Motor Control Command

- *User issues a move command for a specific motor (e.g. phi).*
- *DCSS receives command and forwards it to the DHS responsible for the motor of interest.*
- *DHS handles the communication with the hardware controller.*
- *Motor begins motion.*

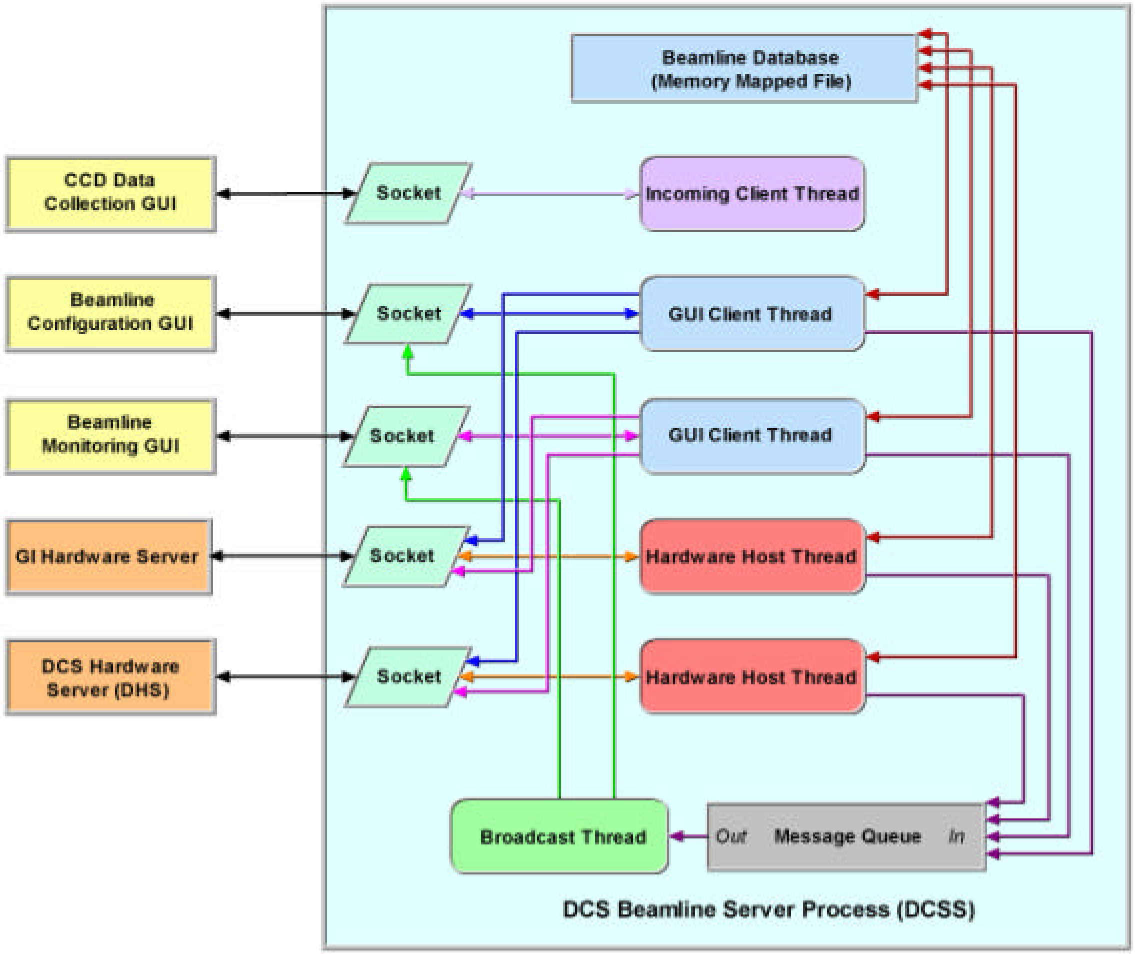


Response to Control Command

- *DHS monitors motor position and sends position update messages to DCSS.*
- *DCSS receives these messages and forwards the new motor positions to all user interface clients.*
- *All users see current motor position changing in real time as it moves.*
- *The motor position updates from DHS continue until the motor stops moving.*



Portability of High Performance C/C++ Server Processes Based on Cross-Operating System (XOS) Library



Supports multithreaded, distributed programs

- Much simplified TCP socket object for rapid network application development.
- Thread creation and synchronization with mutexes and semaphores.
- Memory mapped files.
- Interthread communication.

Portability

- Compile code on Digital Unix, IRIX, OpenVMS, 32-bit Windows.
- Easy to port to new platforms similar to any of the above.

Compile-time approach

- Header file xos.h loads appropriate, system-dependent include files
- Native system calls on each platform for maximum performance
- No runtime overhead for platform independence

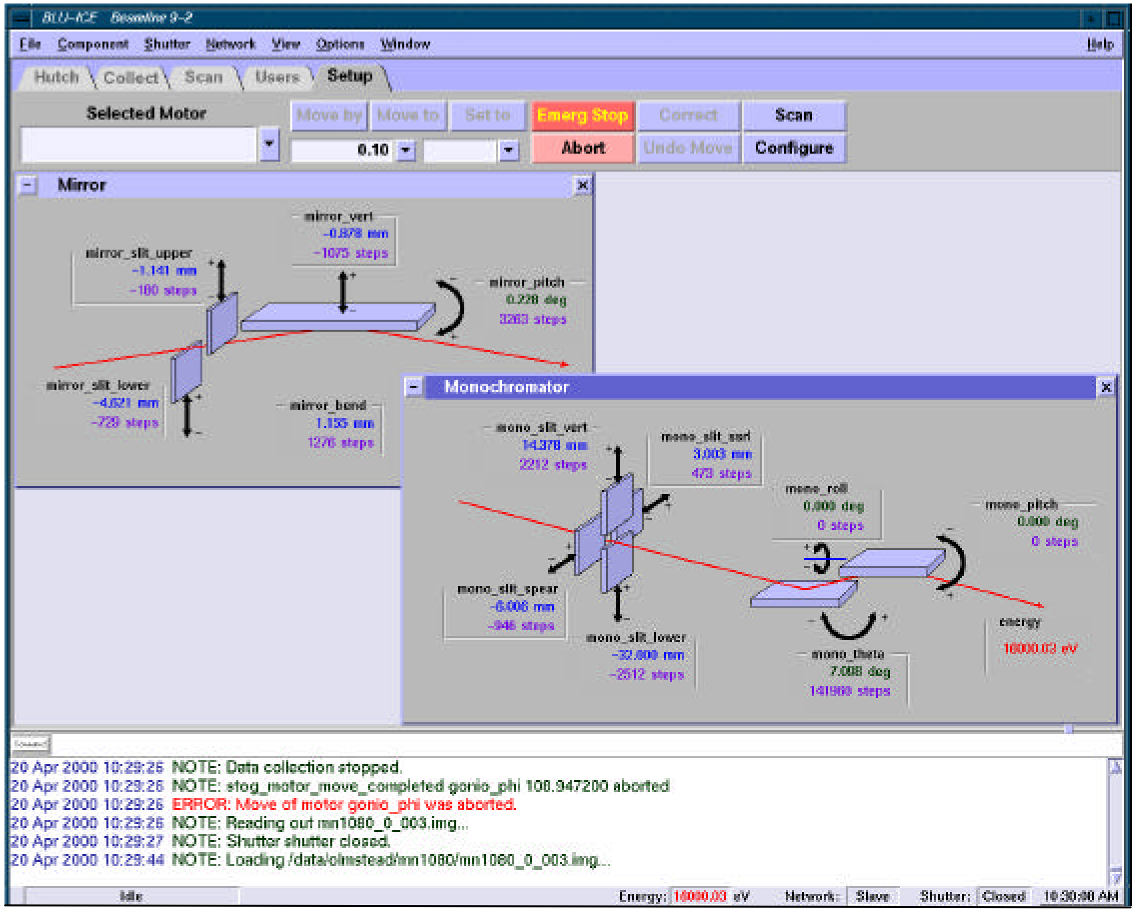
Reliability

- Simpler APIs for more reliable code.
- Less need to study different platforms.

New C++ version under development

- Object oriented, STL based, with error handling via C++ exceptions.
- Cross-platform user authentication and permissions-based file I/O.
- Generalized communication streams for uniform network communication, interthread messaging and file I/O with transparent encryption, compression, etc.
- Cross-platform, cross-vendor, database interface.

Tcl/Tk Features Are Key to the Development of BLU-ICE



Platform Independent

- Tcl/Tk runs on any Unix, VMS, Mac, and 32-bit Windows computer.
- Scripts can also be bundled with Tcl/Tk binaries.

Rapid Development

- Required only a fraction of the code necessary if written in C, C++, or Java.
- Easy maintenance critical at the beam lines.

Object Orientation

- The [Incr Tcl] extension to Tcl provides object-oriention.
- Migration to [Incr Tcl] is simplifying the BLU-ICE code.

Active Canvas

- Binding of events to canvas objects facilitates highly graphical, very interactive user interfaces.

Event based concurrency

- Much simpler than multithreading models for GUI development.

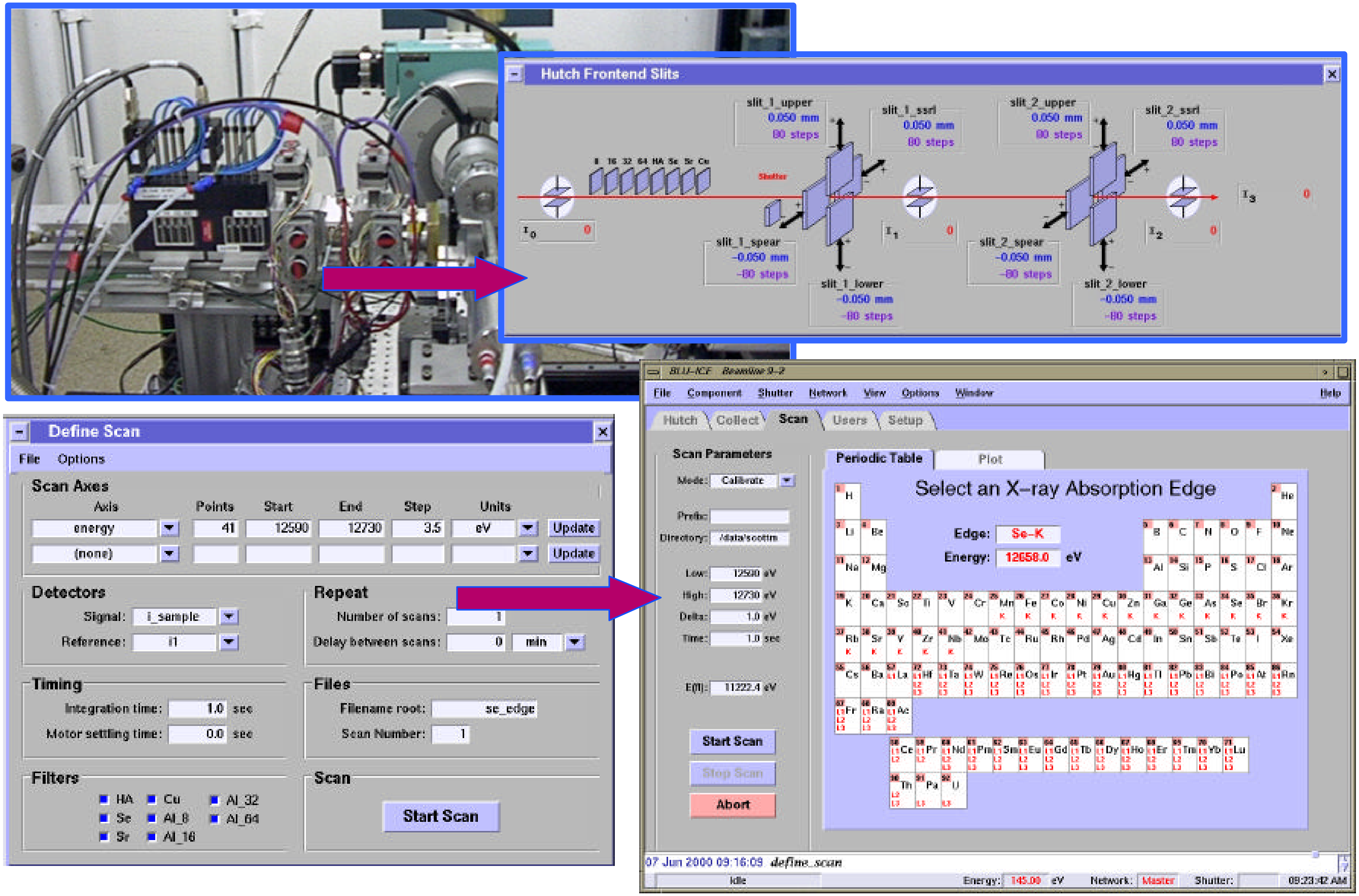
Command Prompt with Scripting

- Command prompt and a full featured programming language for scripting.
- User can script any operation in BLU-ICE using control structures, variables, procedures, file I/O, even classes.

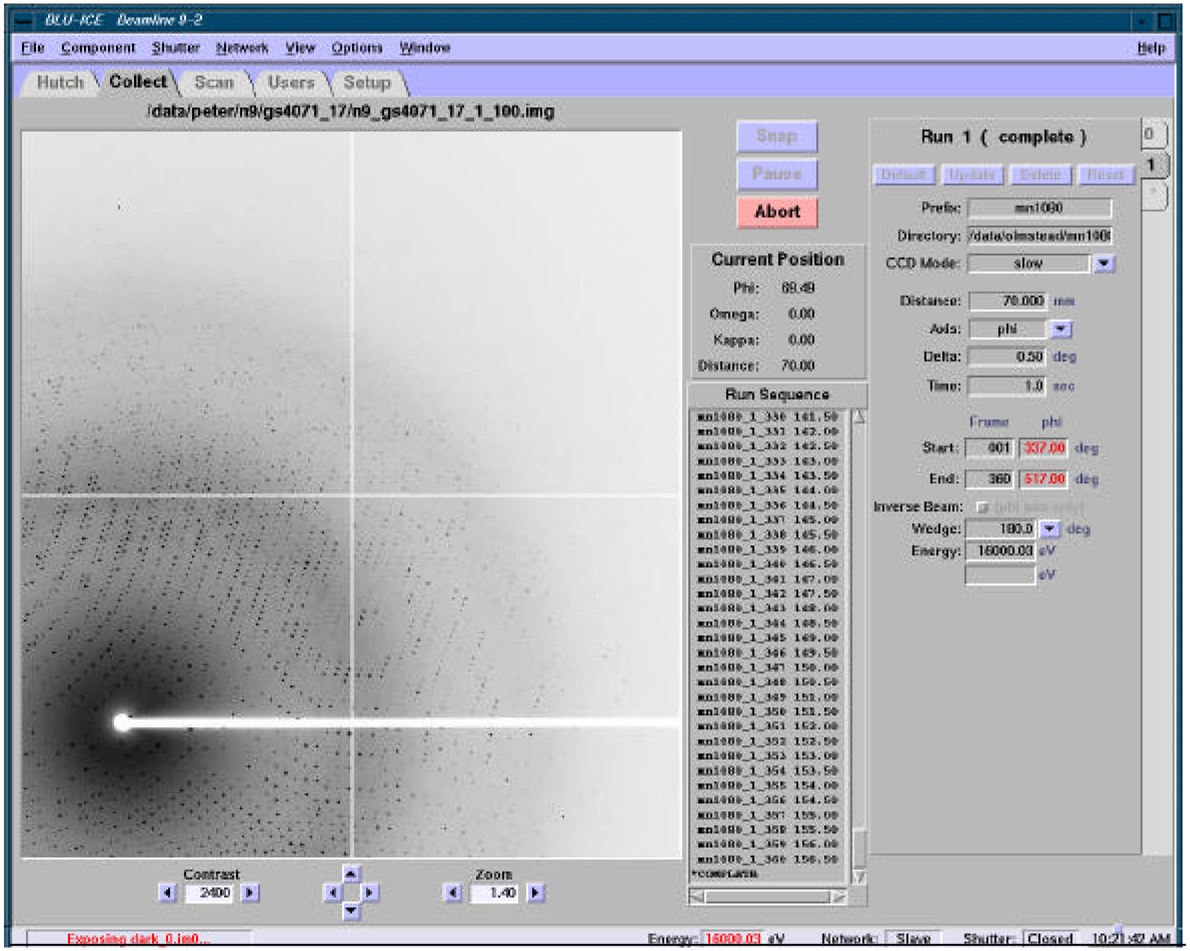
Extensible in C/C++

- Tcl was designed to be extended readily in C. Extensions can be loaded dynamically.
- High performance code, multiple threads and so on are best implemented in extensions.

Extending the GUI Building Philosophy to Support Users



Collaborative Data Collection Control with BLU-ICE



Intuitive MAD Data Collection

- Multiple runs can be defined and started together.
- Supports multiple energies, inverse beam, and wedges.

Run Sequence Preview

- Lists details of data frames to be collected for each run.
- Changes as data collection parameters are edited.

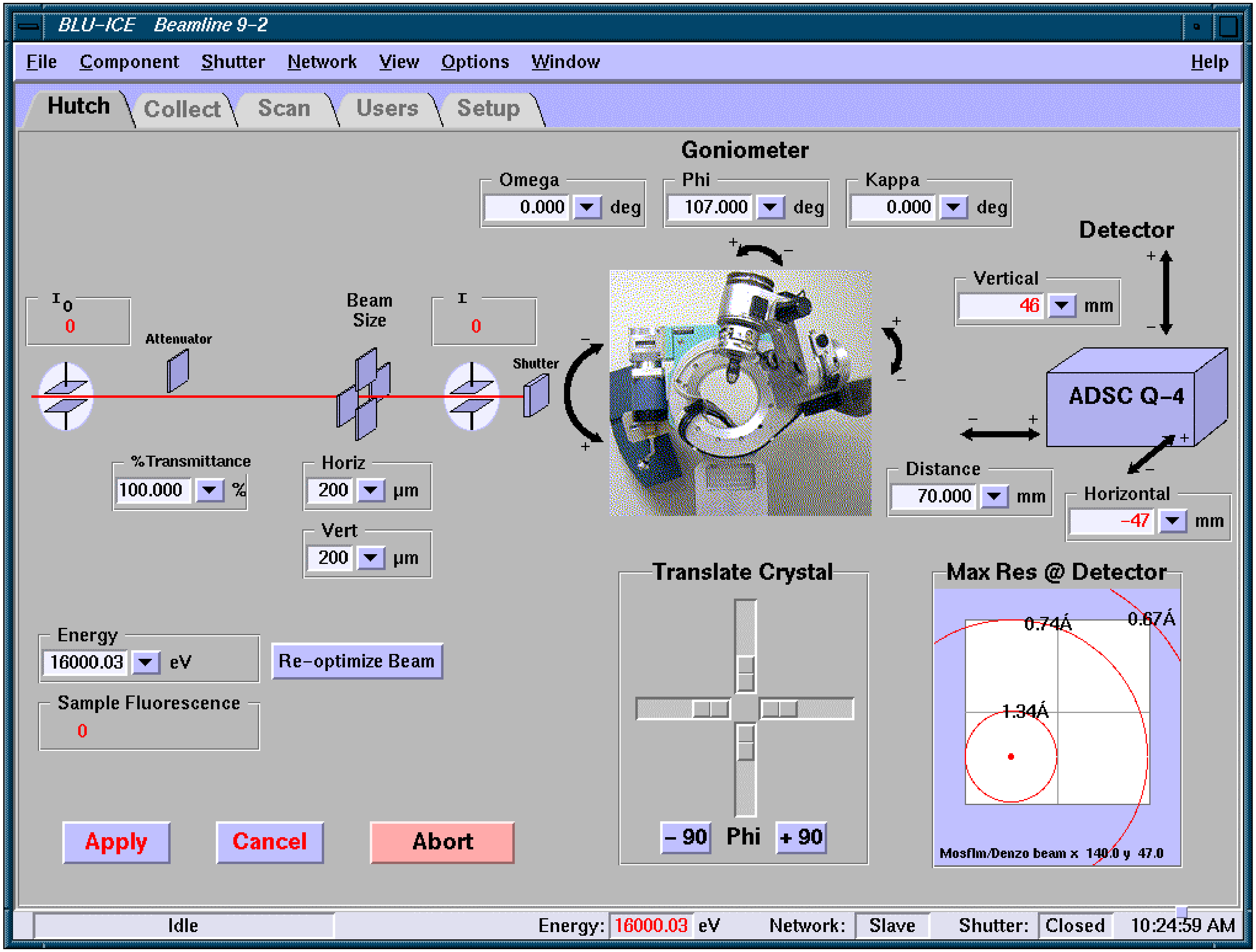
Robust Pause/Restart Capability

- Data collection may be paused at any time.
- Next frame to collect may be selected in sequence window.

Supports Multiple Instances

- Multiple instances of BLU-ICE may monitor data collection.
- All instances show run definitions being edited.
- New diffraction images are displayed on all instances.
- Data collection continues even if all BLU-ICE instances exit.

BLU-ICE Experimental Setup Window



Example Scripted Device: TABLE_VERT

```
proc table_vert_move { new_table_vert } {

   # import device variables
   variable table_pitch

   # move the two motors
   move  table_vert_1  to  [ table_vert_1_calculate $new_table_vert $table_pitch ]
   move  table_vert_2  to  [ table_vert_2_calculate $new_table_vert $table_pitch ]

   # wait for the moves to complete
   wait_for_devices  table_vert_1  table_vert_2
}


proc table_vert_1_calculate { tv tp } {

   # import device variables
   variable table_pivot
   variable table_v1_z
   variable table_pivot_z

   # calculate position of table_vert_1 given vertical height and pitch
   return [ expr  $tv + ( $table_v1_z - $table_pivot_z ) * tan ( [ rad $tp ] )  ]
}
```

DCSS Scripting Engine: Server-side Motion Control Scripting

Scripting capability of BLU-ICE embedded in DCSS

- [Incr Tcl] interpreter executes as one thread within DCSS.
- Interpreter runs the non-GUI portions of BLU-ICE.
- Scripting Engine can run any scripts developed within BLU-ICE.

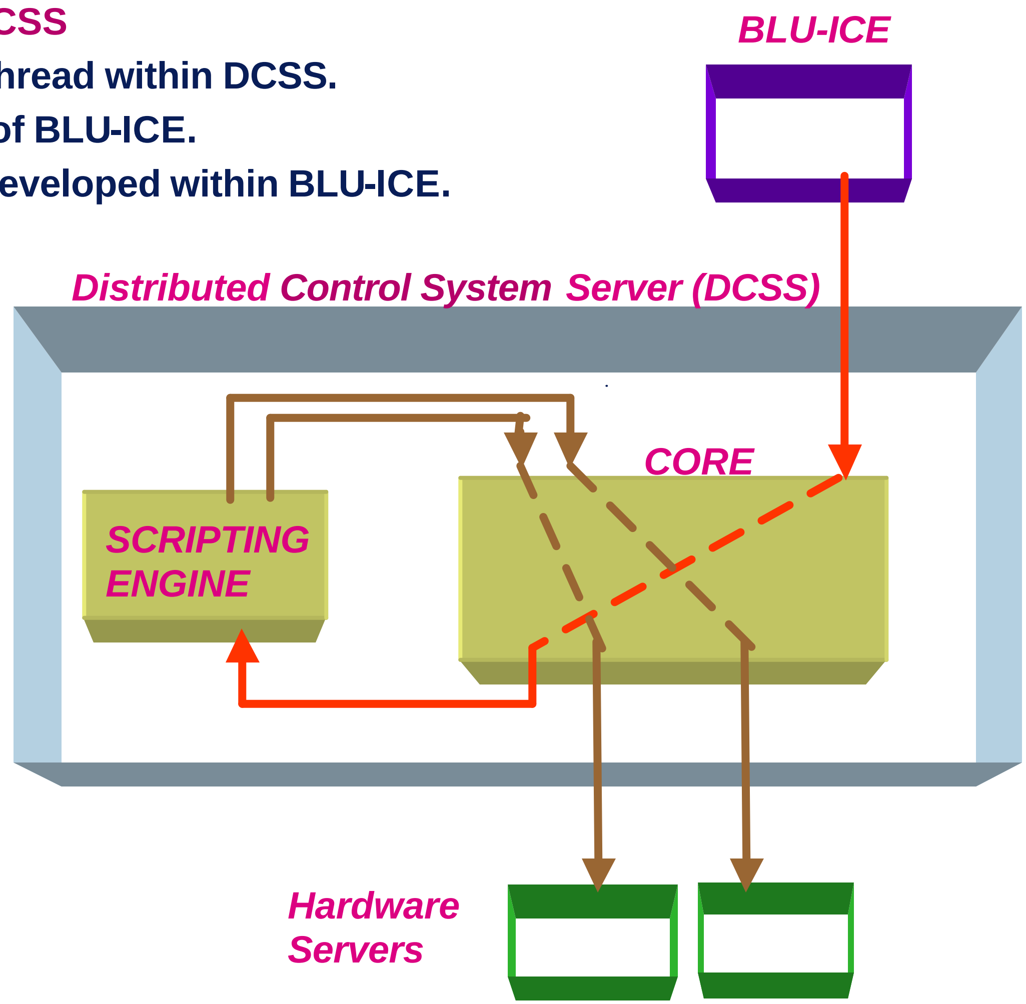
Scripting Engine connects to DCSS core twice

- Once as a Hardware Server serving "Scripted Devices."
- Once as a "User Interface Client."

Scripted device command routing

- Request for a move of a scripted device is routed to the Scripting Engine hardware server interface.
- Scripting Engine runs a script that issues move requests for the child motors.

Children motors may also be scripted devices with their own children!



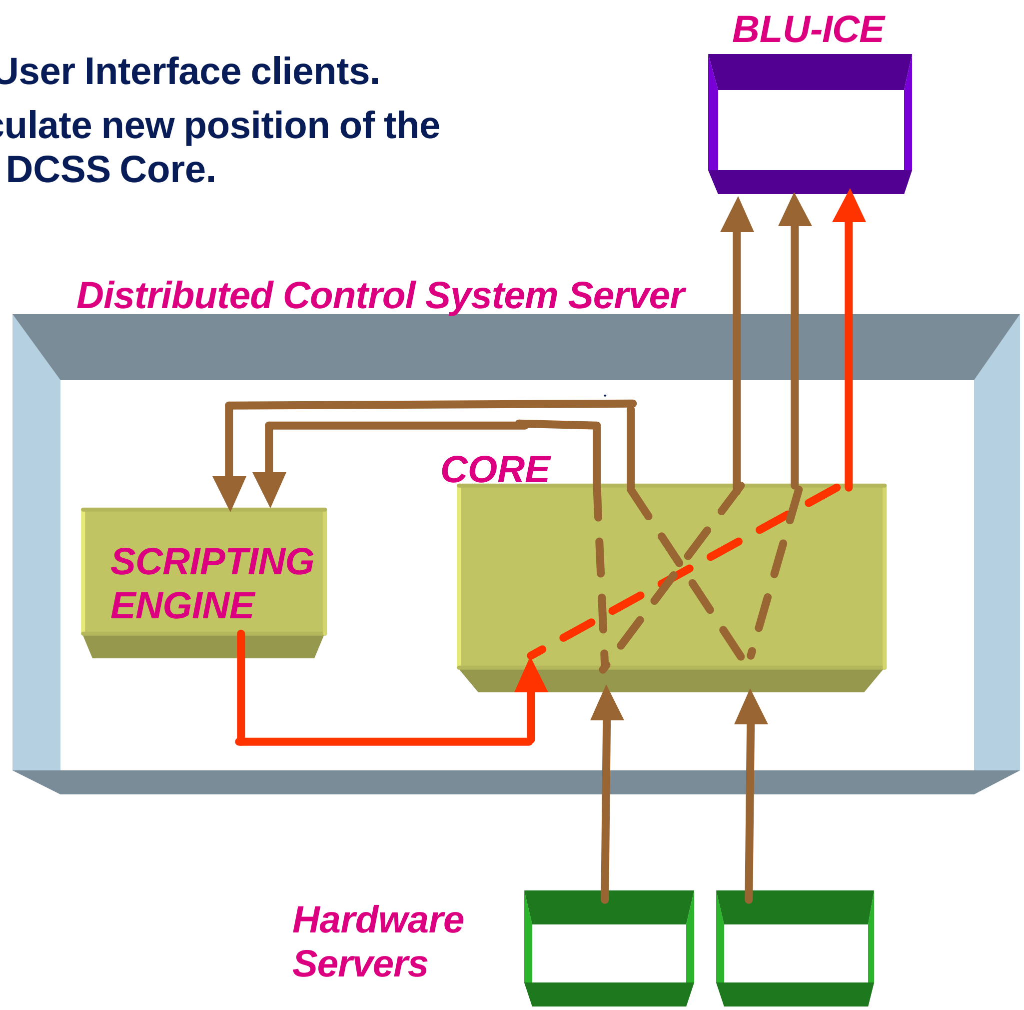
Scripted Devices: Path of a Response Message

Response messages flow backwards.

- Real-motor position updates go to all User Interface clients.
- Scripting Engine runs scripts that calculate new position of the Scripted Device and sends updates to DCSS Core.
- DCSS Core forwards the updated Scripted Device positions to all User Interface client.
- BLU-ICE gets a continuous stream of updated positions for both real motors and scripted devices.
- System works recursively, handling nested Scripted Devices.

Possible Uses for Scripted Devices Unlimited

- Sophisticated device control with many intermediate states.
- Complex limit checking for collision avoidance.
- New data collection schemes.



Acknowledgements

The Macromolecular Crystallography Group

Mike Soltis
Peter Kuhn

Henry Bellamy
Aina Cohen
Ashley Deacon
Paul Ellis
Thomas Eriksson
Ana Gonzales
Mike Hollenbeck
Scott McPhillips
Tim McPhillips
Fred Bertsch
Pavel Petrashen
Paul Phizackerley
Amanda Prado