



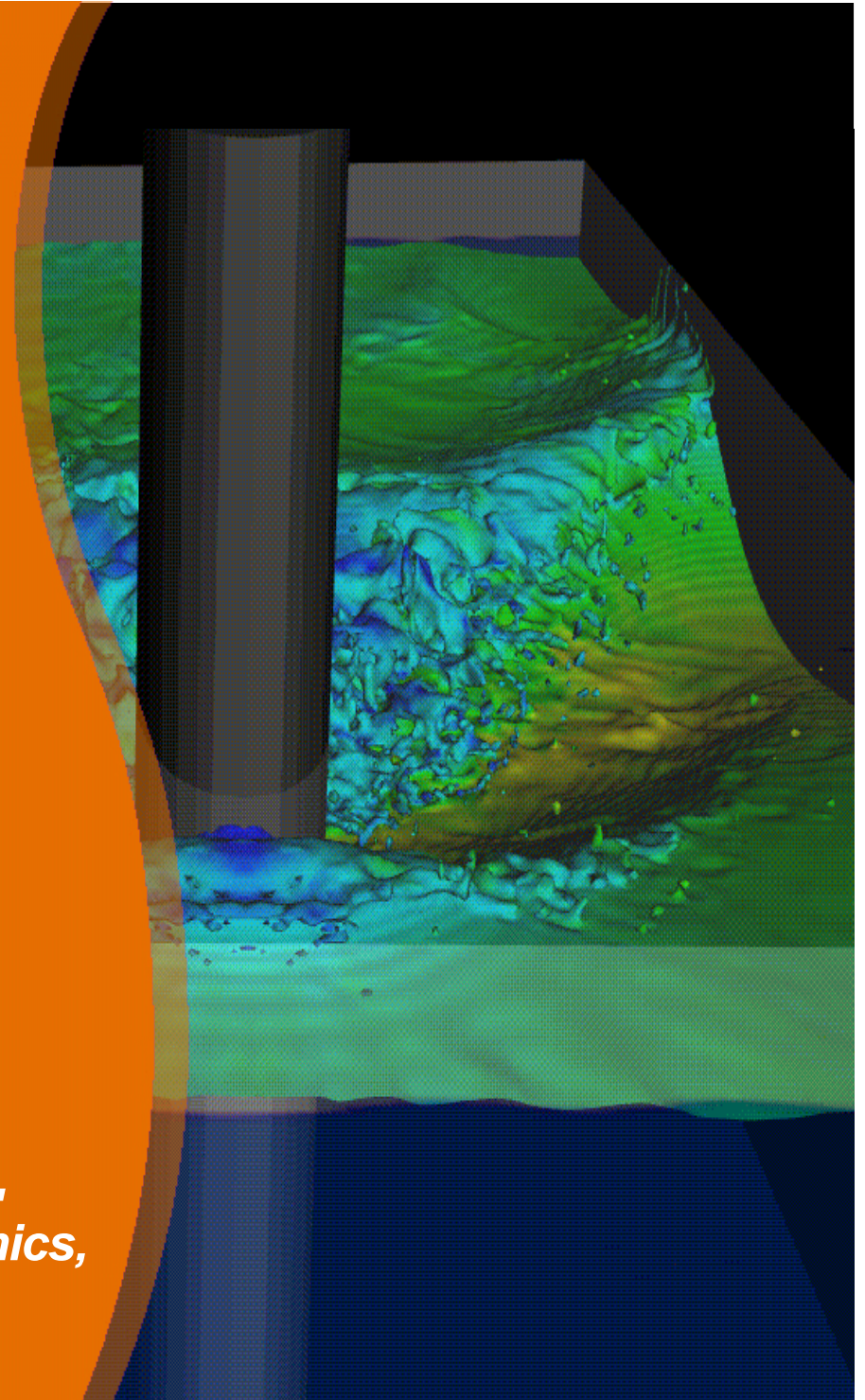
HPC Visualization On the Grid

W. Dean Stanton

*Sr. Staff Engineer, Advanced Visualization,
Sun Microsystems*

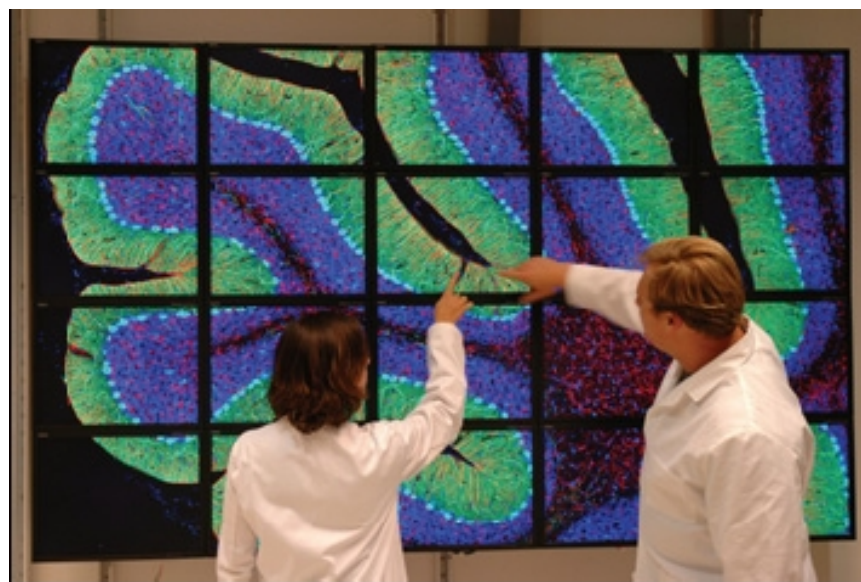
Linda Fellingham, Ph. D.

*Manager, Advanced Visualization and Graphics,
Sun Microsystems*

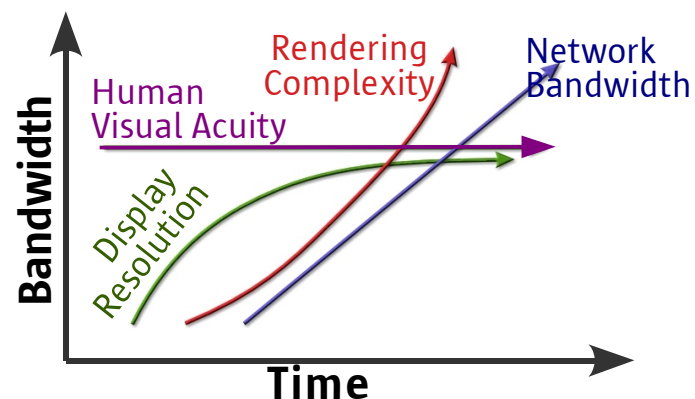


Understanding Huge Data Sets Requires Interactive Visualization But visualization is hardware intensive

- Big data clogs networks
- Requires lots of memory
- Requires lots of CPU power
- Graphics accelerators need lots of power and cooling
- Workstations inadequate?
- And who wants to work near the heat and noise?

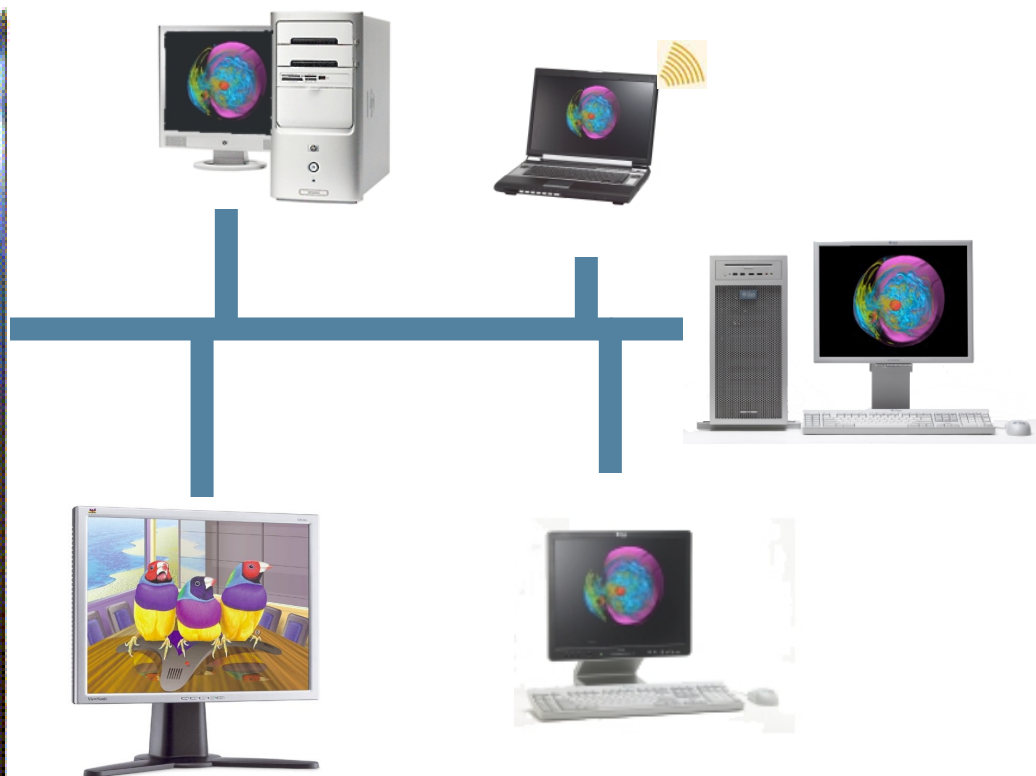


Change the Model



Share Visualization on the Grid over the Network

- Big Data
- Big Memory
- Secure
- Graphics Power
- CPU Power
- Keep heat and noise in the server room
- Send images over network
- Share the cost





Agenda:

- **Scalable Visualization Solutions**
- **Shared Visualization Software**
- **Integration with Sun Grid Engine**

Sun Scalable Visualization Solutions

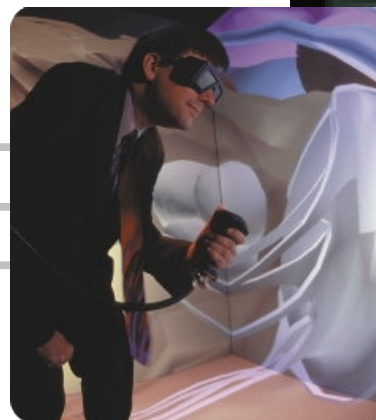
Graphics servers support multiple graphics devices to drive:

- Higher performance
- Higher image quality
- Higher resolution



Sun Fire Servers

HPC Visualization on the Grid



CAVEs



**Power Walls,
Immersive Projections**

Scalability Problem 1: Servers

- Servers have lots of processors and memory, but do not have adequate space, power, or cooling for a high-end 3D graphics accelerator
- Solution 1:
 - > Get the graphics card(s) out of the system
- Hardware Technology
 - > NVidia's Quadro® Plex Visual Computing Systems

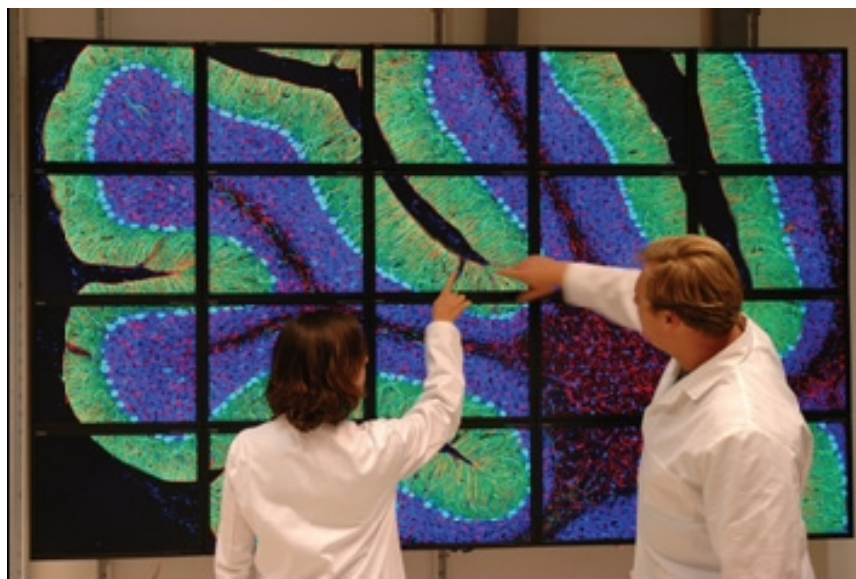


Scalability Problem 2: Lots of Screens

- Need to view applications across many screens, in order to view adequate detail
- Solution 2:
 - > Distribute the rendering across many systems (sort first)
 - > High-bandwidth, low-latency interconnects (InfiniBand or 10gigE)
- Open Source Software: Chromium or OpenSceneGraph



HPC Visualization on the Grid

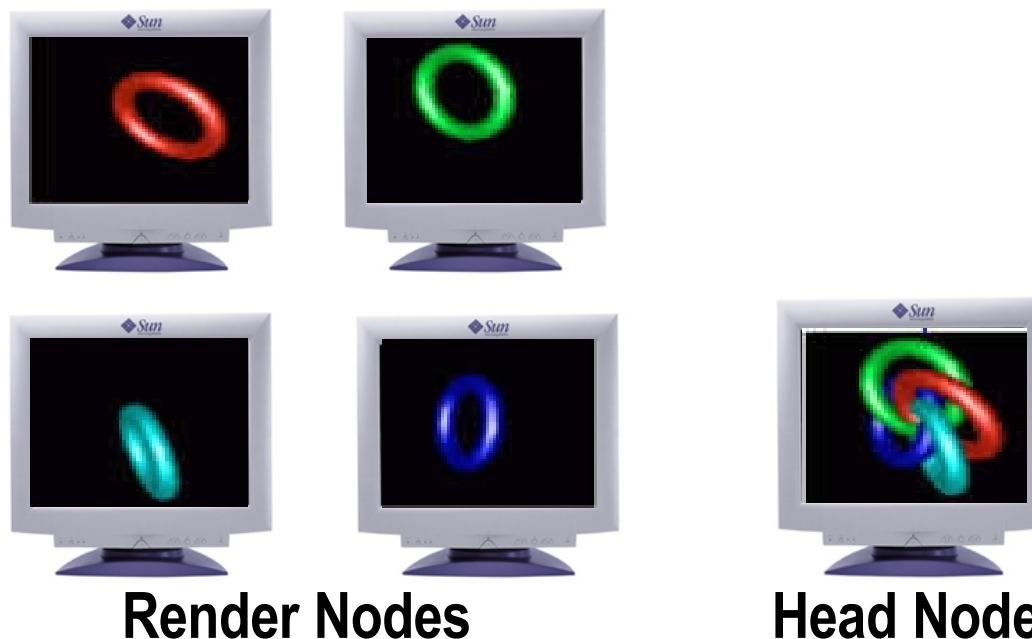


Open Source Grid and Cluster Conference

May 14, 2008

Scalability Problem 3: Performance

- Performance can be too slow for very large data sets
- Solution 3
 - > Break the problem up and distribute the rendering to multiple render nodes, and reassemble on the head node (sort last)
- Open Source Software Technologies:
 - > Paraview (Parallel Visualization Application) or Chromium (with work)



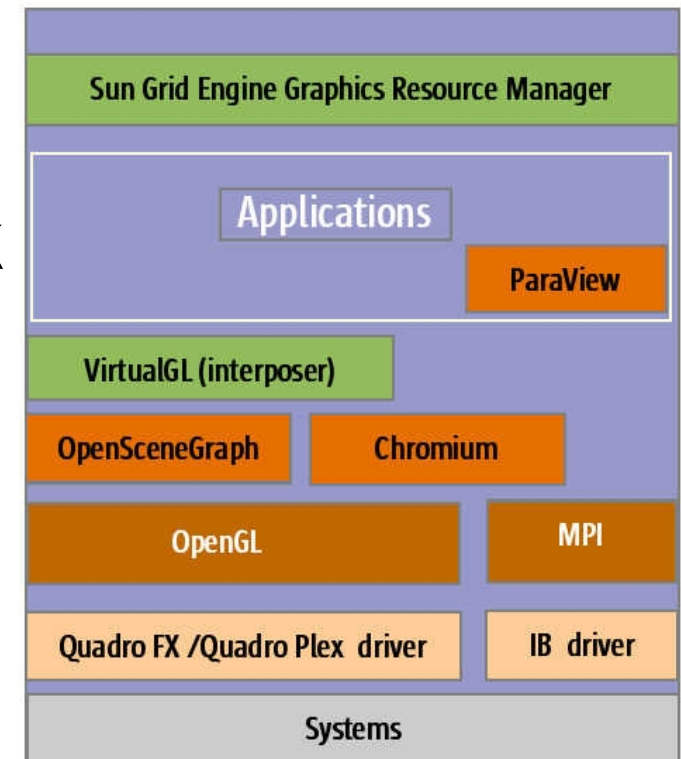
Software Details

- Scalable Visualization 1.1 software supports:

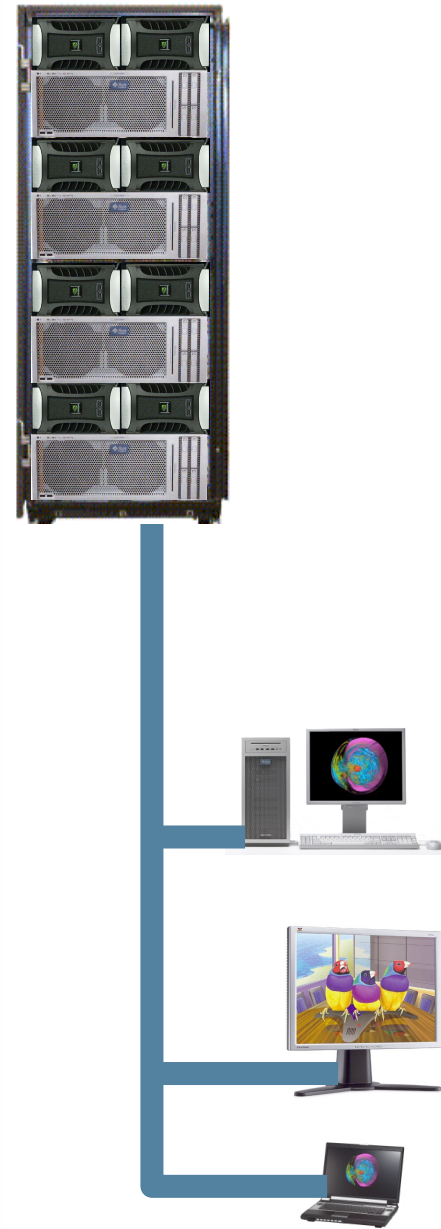
- > Red Hat Linux (RHEL 4U2-5 and RHEL 5/5.1)
- > SLES 10
- > Solaris 10 Update 3 or 4

- Complete Open Source software stack

- > Chromium, MVAPICH2, OFED, OpenSceneGraph, Paraview
- > Sun added value:
 - > Pre-built binaries, tested for interoperability
 - > Installation scripts and configuration files
 - > Wrappers for greater ease of use
 - > MPI protocol added as a Chromium interconnect
 - > Supported on Sun hardware
 - > Free download for Solaris at OpenSolaris.org/os/project/visualization-hpc/



Sun Shared Visualization Software



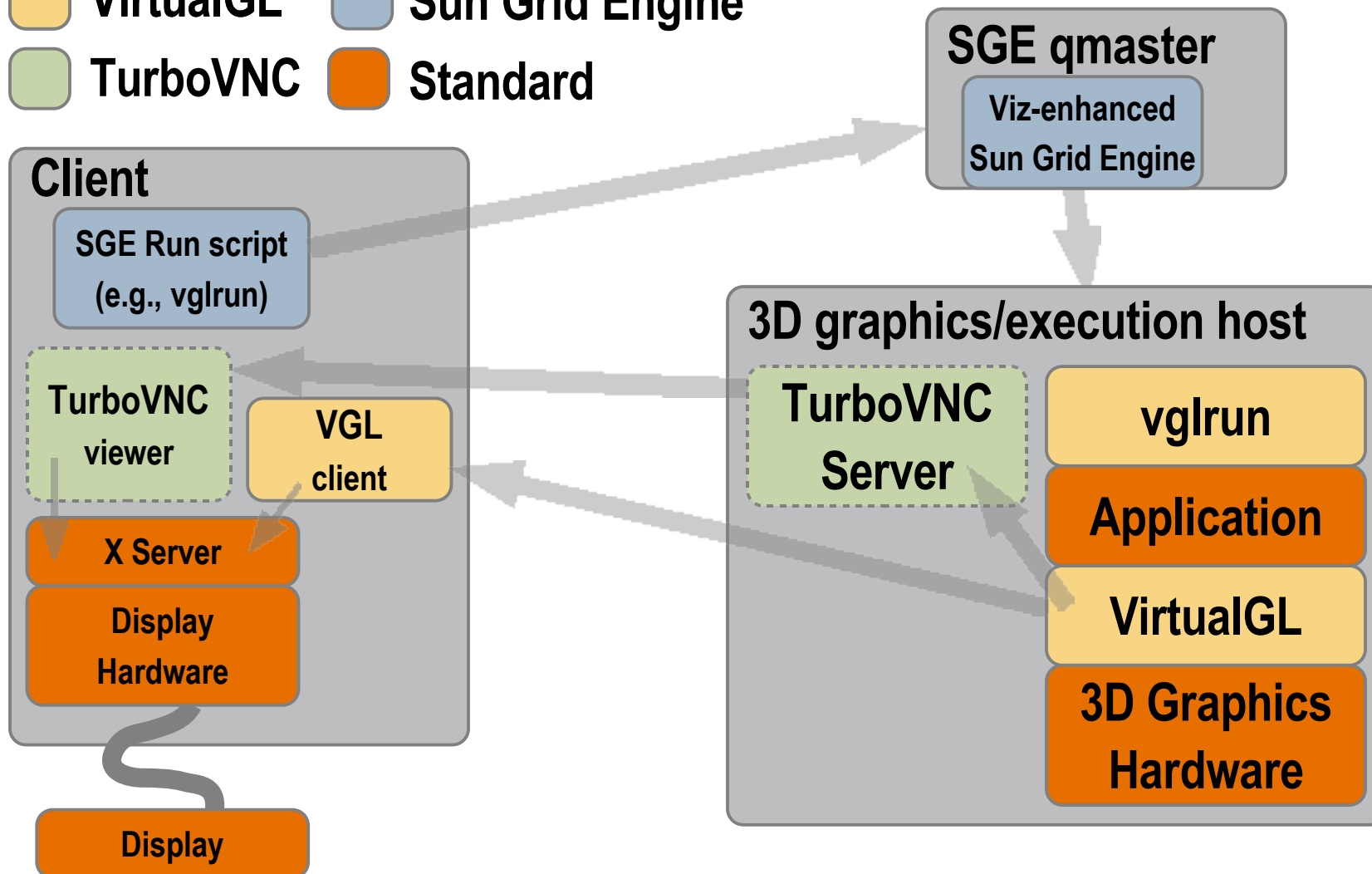
The Shared Visualization Model

Feature	Benefit
Secure	Data stays on the server. Control data access, even among 3D application users.
Performant	Optimized compression and decompression transfers visually lossless images at interactive speeds (more than 20 frames/sec).
Interoperable	Client only needs enough network performance and a display. Interoperable with a variety of devices
Sharable	Average CPU, memory, and graphics needs over many users. Reduce total cost of ownership
Scalable	A single user can access lots of CPU, memory, and attached graphics. Get more resources than possible in ANY workstation
Flexible	Graphics computation and display technology are separated. Display on what you already have, upgrade graphics separately
Load Balancing	Better utilization of compute and graphics resources. Grid software helps to find and manage resources.

Sun Shared Visualization Software

Transparent Remote Access to 3D Applications

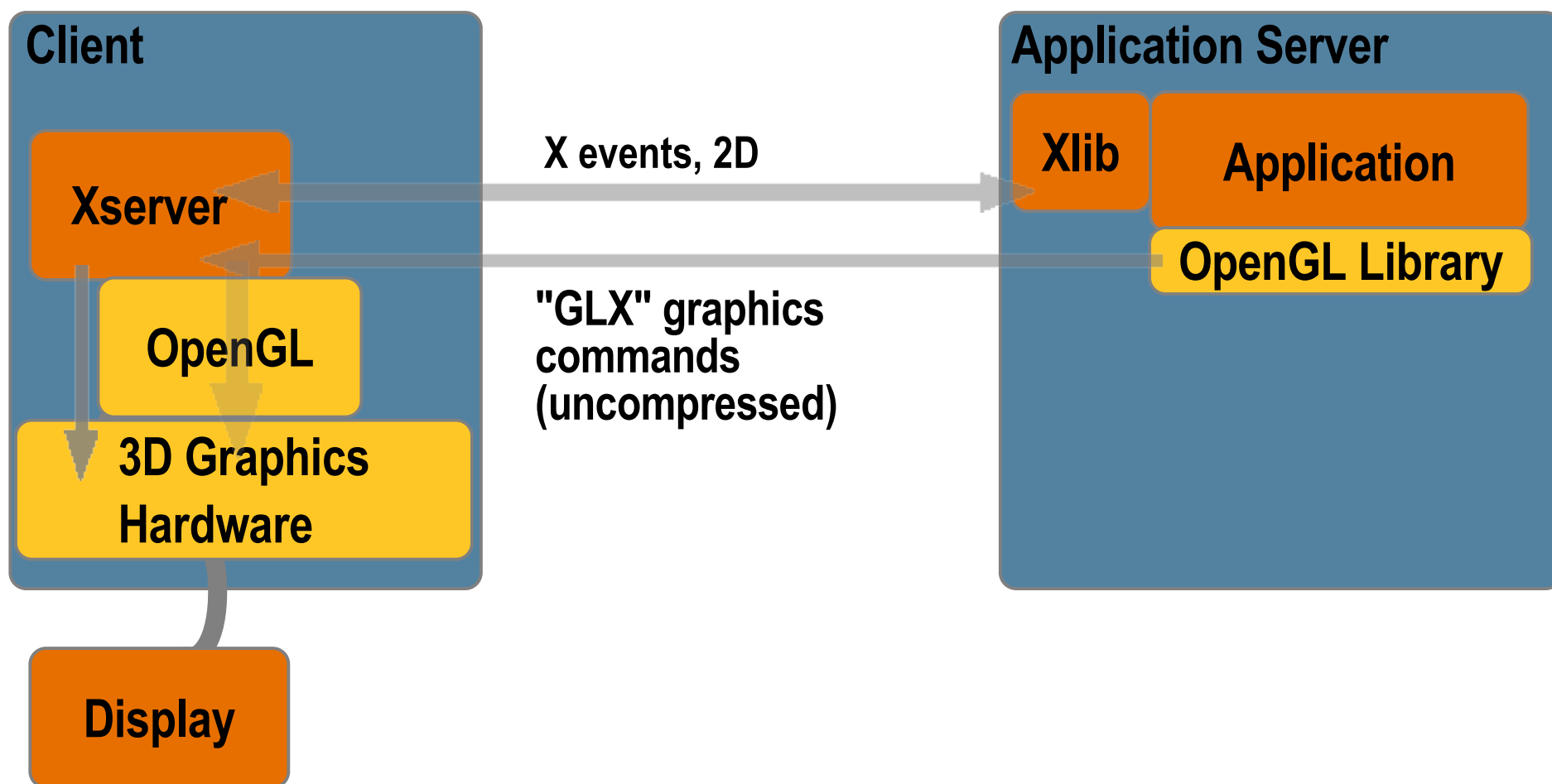
- VirtualGL**
- Sun Grid Engine**
- TurboVNC**
- Standard**



VirtualGL

- Purpose
 - > Allows OpenGL applications which were designed to run and display on the same system to efficiently and transparently run on one system and display on another.
- Components
 - > "Middleware" software for Linux and Solaris servers
 - > Platform-optimized Image compression technology
 - > Various data transport methods
 - > vglclient program decompresses and displays the images
 - > Clients for Solaris, Linux, Windows, Mac OS X
 - > Sun Ray ultra thin clients
- Open source software project sponsored by Sun

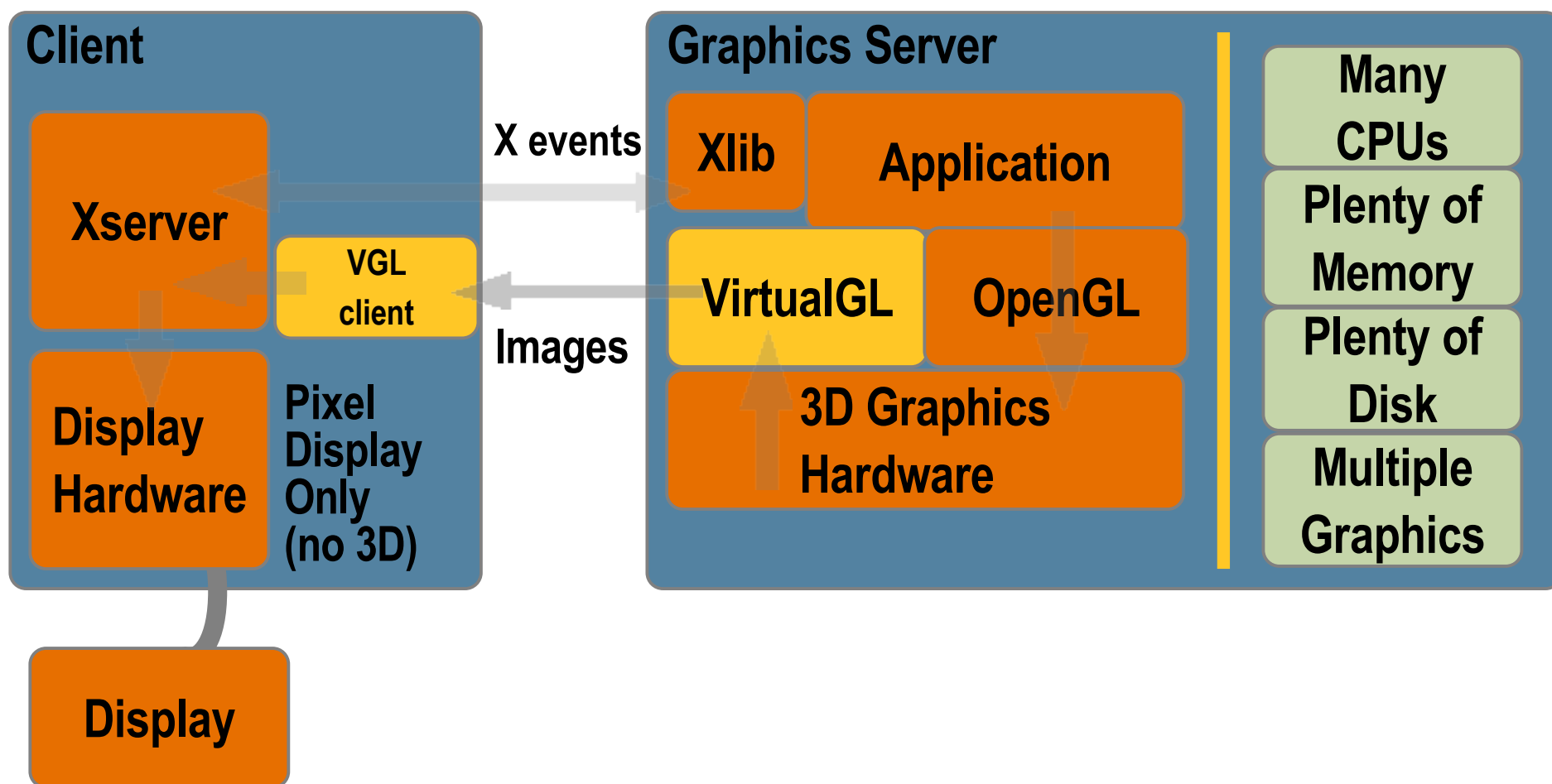
Remote Graphics In the Past



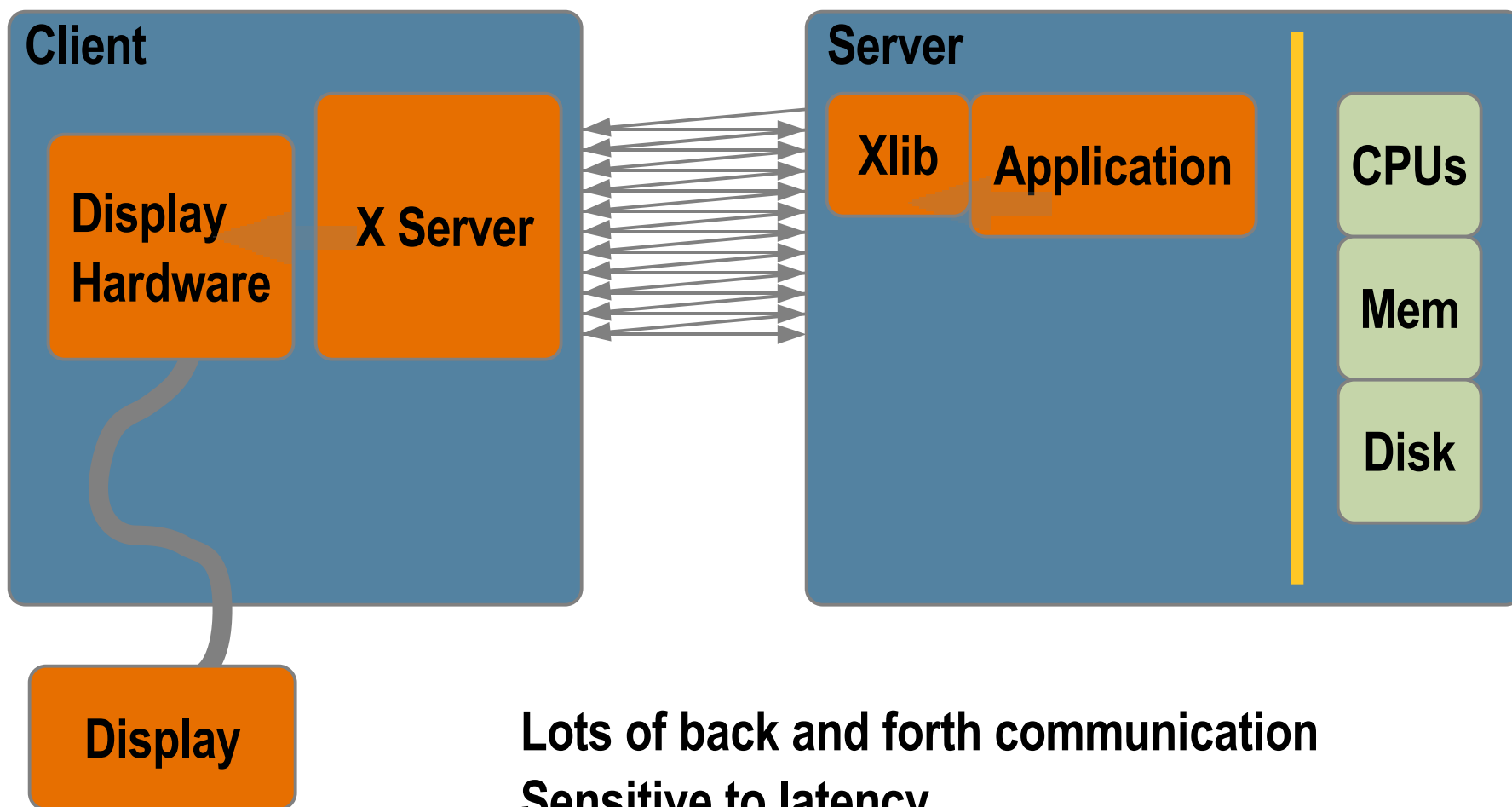
Remote Graphics Using VirtualGL

client% `vglconnect my_server`

server% `vglrun my_application`



Standard Remote X

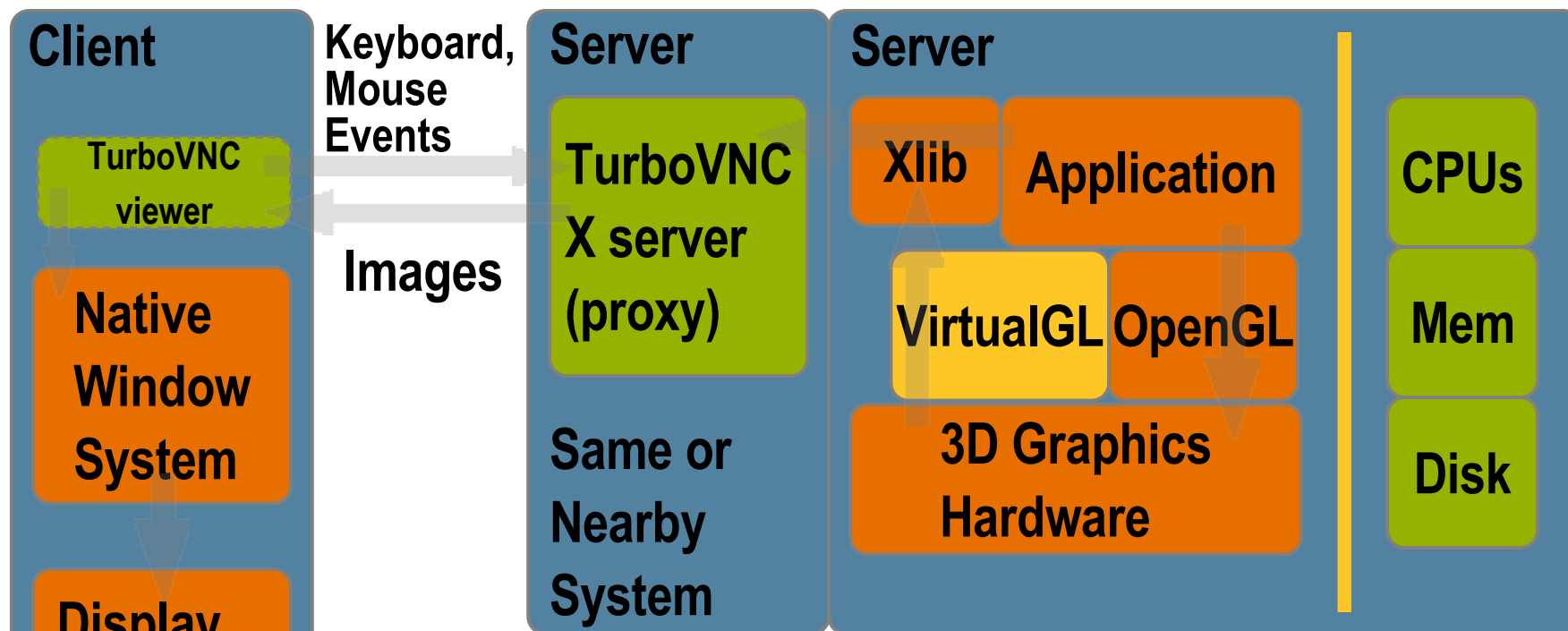


TurboVNC

Latency Tolerant Remote Visualization with Collaboration

- Purpose
 - > Allows X applications which were designed to run and display on the same system (or on low latency networks) to transparently run on one system and display on another.
 - > Also enables collaboration by allowing more than one client system to display the X session.
- Open source project sponsored by Sun
 - > Derived from TightVNC but uses same optimized image compression technology as VirtualGL.
 - > Interoperable with other VNC viewers, including Java-based WebVNC

VirtualGL With TurboVNC



```

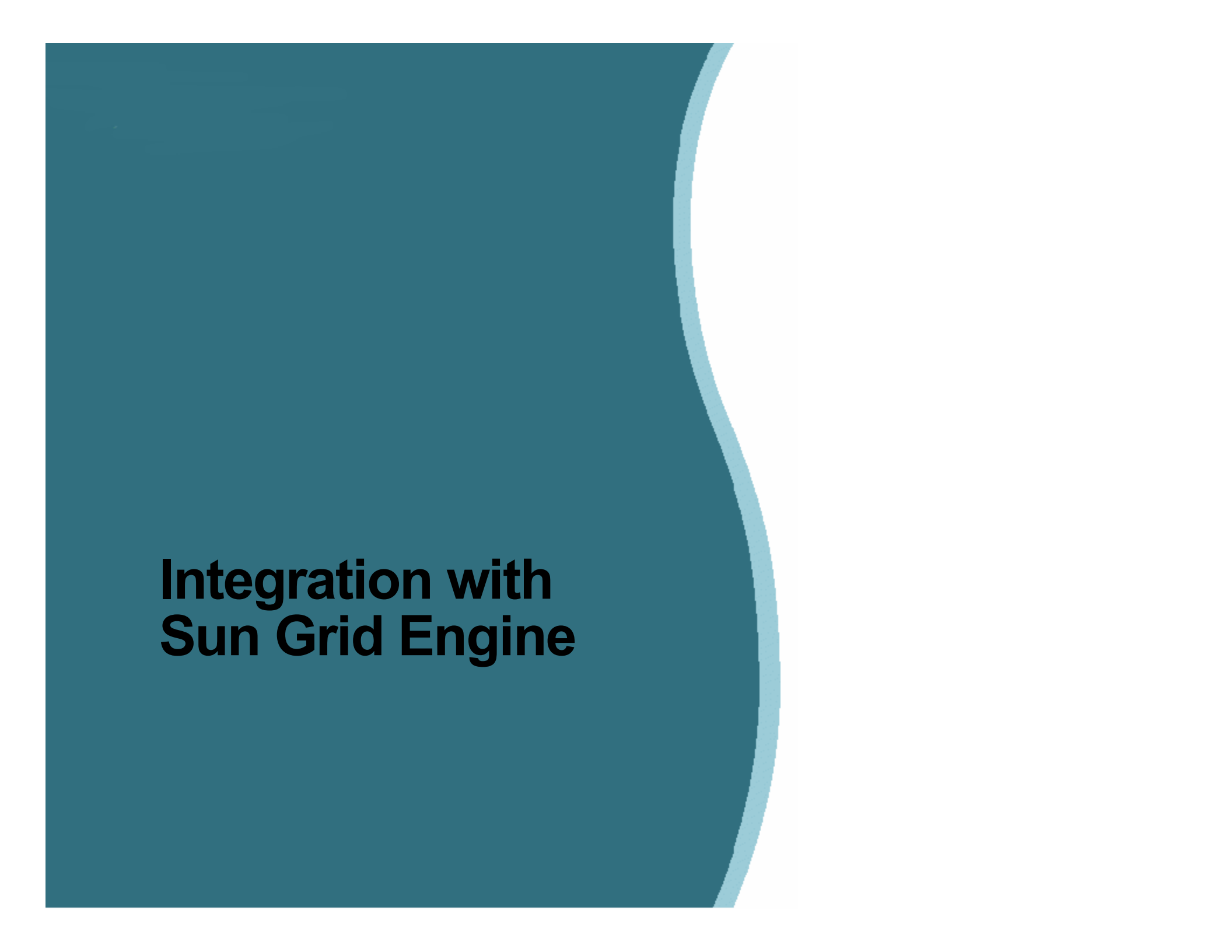
client% ssh my_server
server% vncserver
New 'X' desktop is my_server:1
client% vncviewer my_server:1

```

```

server% vglrun my_application

```

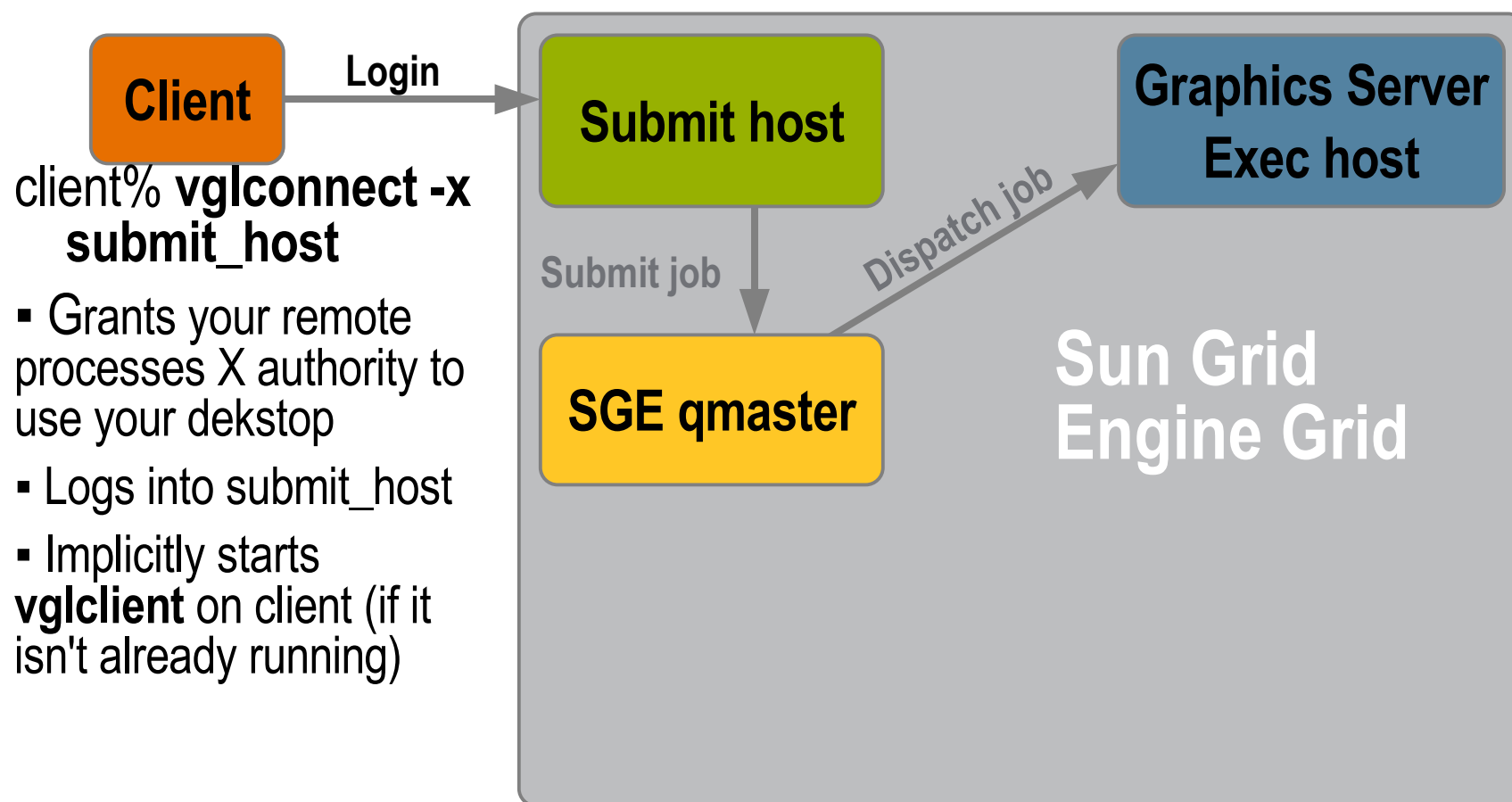


Integration with Sun Grid Engine

Sun Grid Engine (With Shared Viz Enhancements)

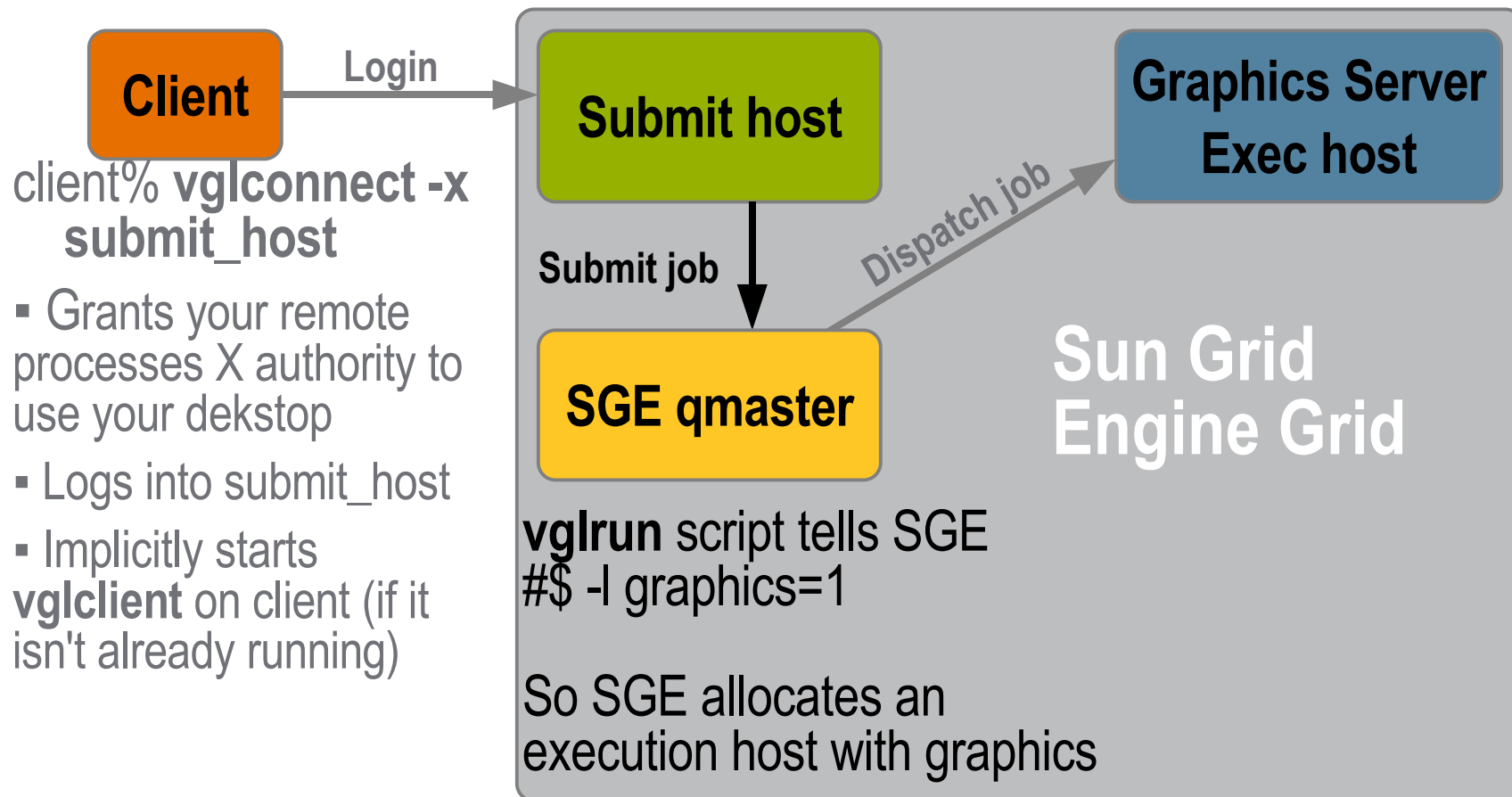
- Purpose
 - > Lets users share graphics servers
 - > SGE assures available CPU, memory, and graphics resources
- Components
 - > Standard SGE provides management and allocation of regular compute resources (CPUs, memory, OS, software licenses).
 - > Enhancements allow SGE to manage graphics resources
 - > provides “user-transparent” connection between the allocated graphics device and the user's display on the remote client.
 - > Advance Reservation system allows resources to be reserved for a specific time in the future.
- Open source software developed by Sun

Sun Grid Engine With Shared Visualization



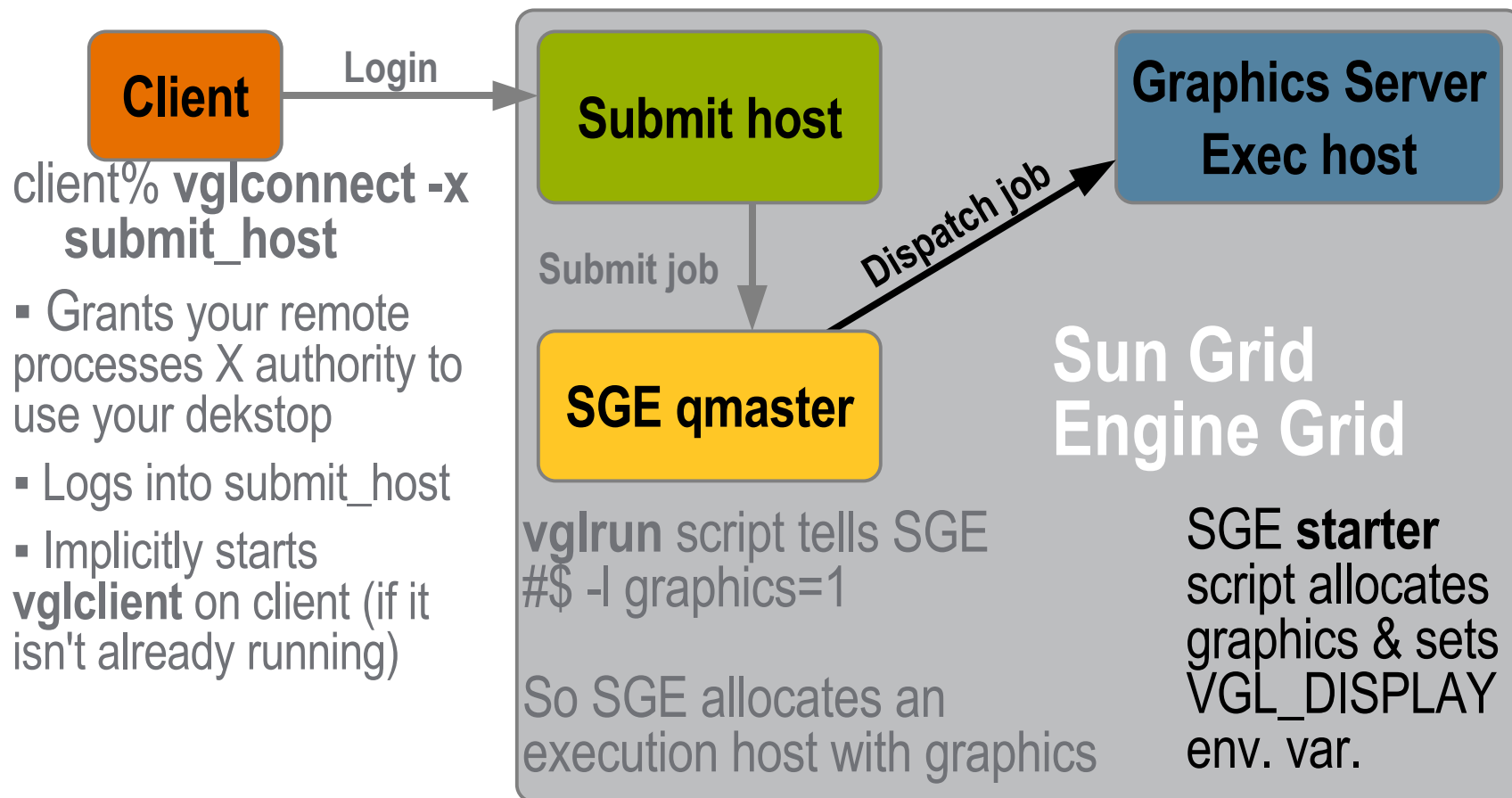
Sun Grid Engine With Shared Visualization

```
submit_host% qrush -b no /opt/VirtualGL/bin/vglrun my_app
```

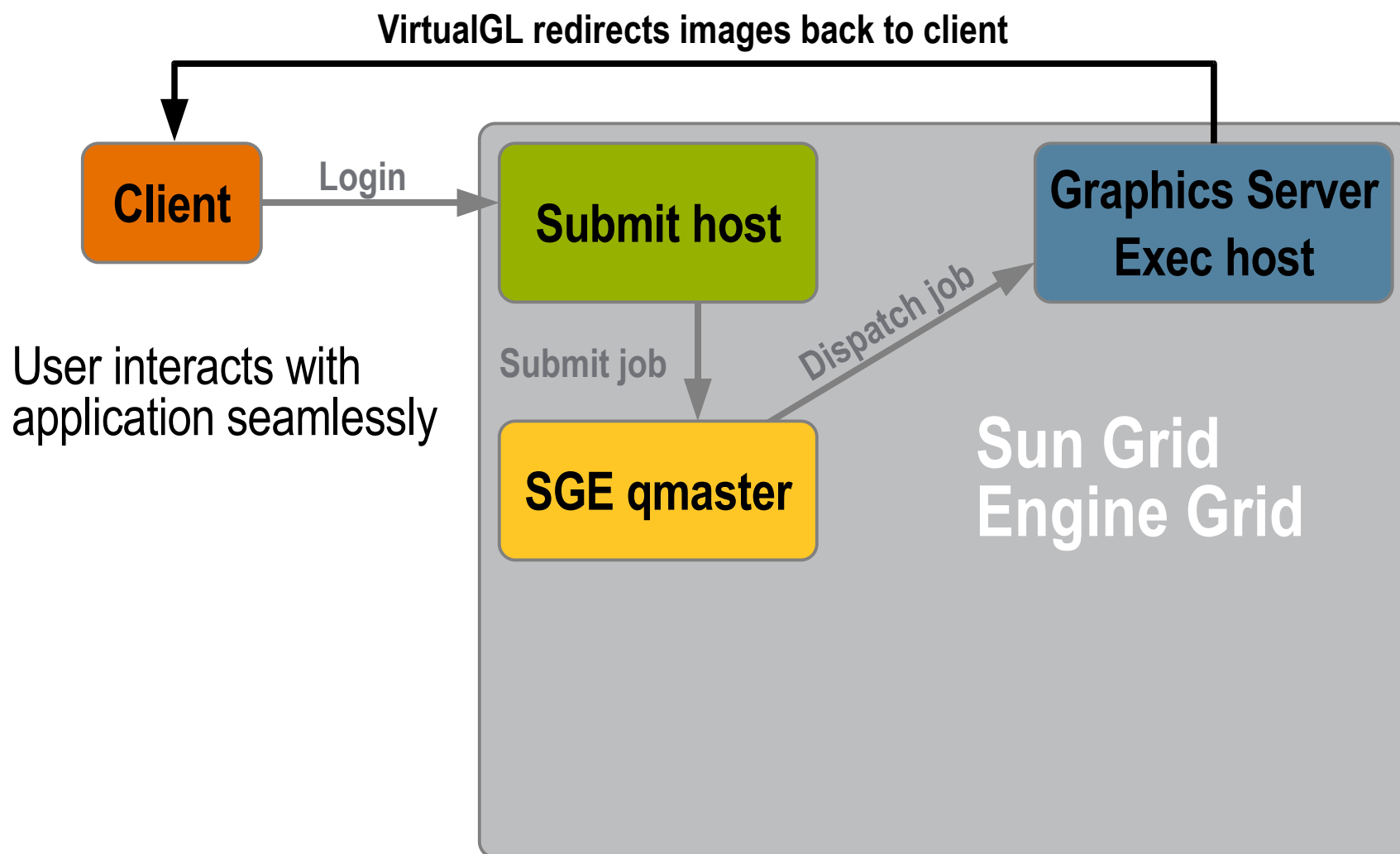


Sun Grid Engine With Shared Visualization

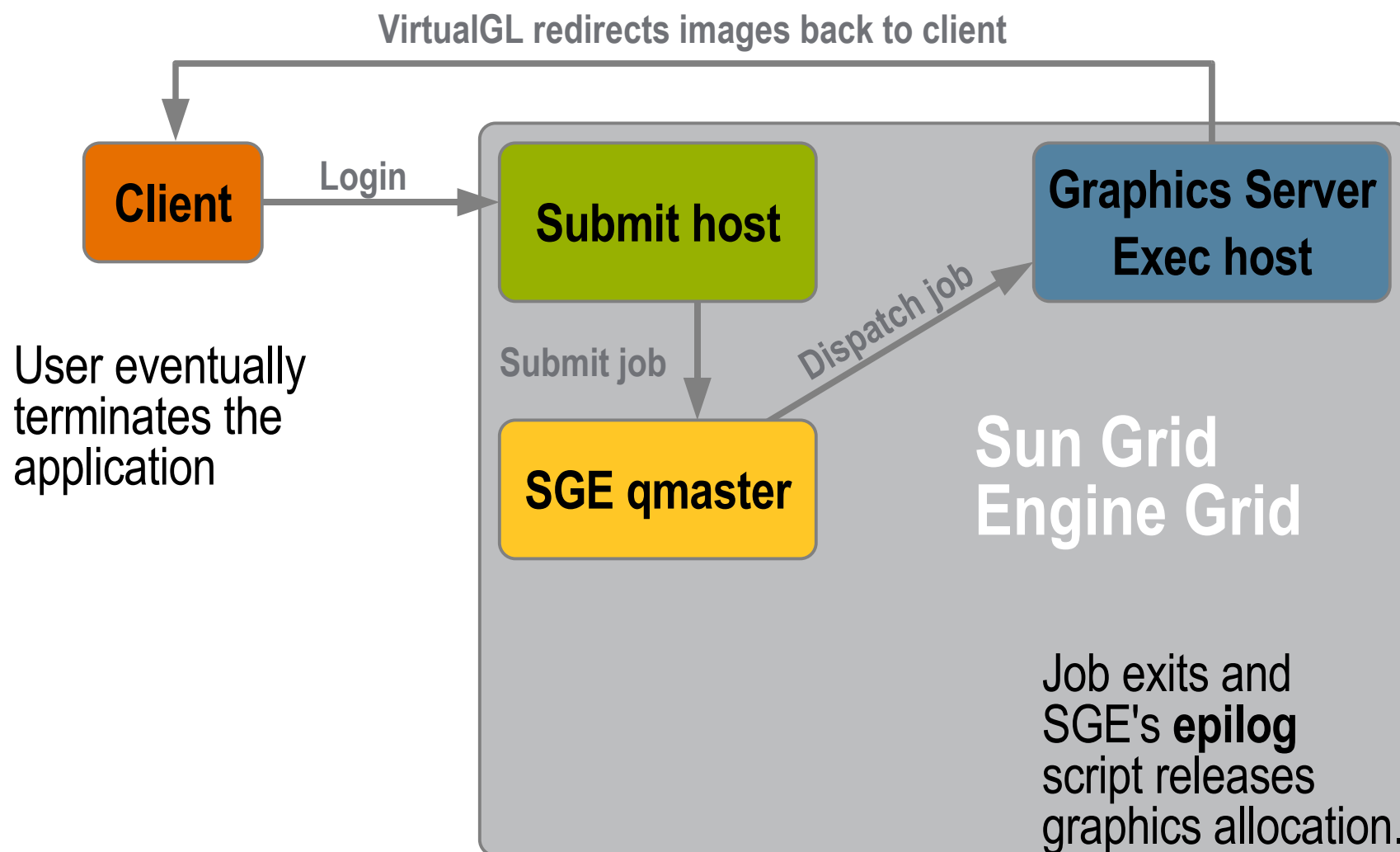
```
submit_host% qrush -b no /opt/VirtualGL/bin/vglrun my_app
```



Sun Grid Engine With Shared Visualization



Sun Grid Engine With Shared Visualization



Grid Engine Application Script

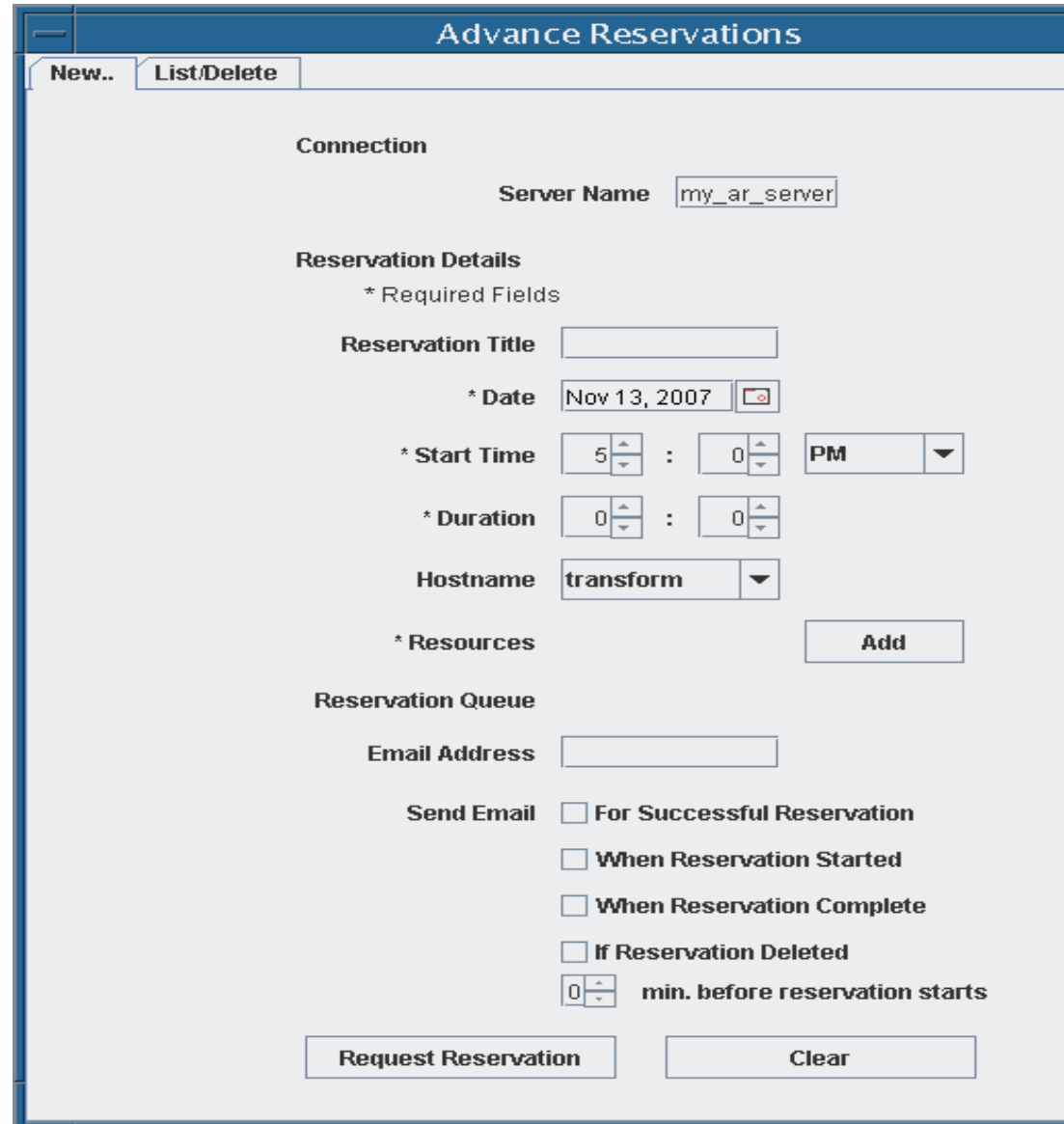
Script Uses `#$` Comments to Describe its Needs to Grid Engine

- Grid Engine will allocate acceptable host and graphics
 - > `qrsh -b no my_script` > `qsub -now y my_script`

```
#!/bin/sh
#$ -N my_app      ← Name of Grid Engine job
#$ -l graphics=1      ← Job requires a graphics card
#$ -l arch=lx24-x86 | lx24-amd64      ← Required Architecture(s)
# Here, required architecture is "Linux (with 2.4 or 2.6 kernel) on x86 or x64"
#$ -v DISPLAY,SSH_CLIENT      ← Save environment vars. with job
# These variables enable VirtualGL to find the client's X display (desktop).
# Invoke application using VirtualGL to route 3D to allocated graphics card
vglrun my_application "$@"      ← Invoke app with any args
# Script invokes VirtualGL implicitly, so submitter need not mention vglrun
```

Making an Advance Reservation

- GUI invoked with **runar** script (that invokes Java)
 - > `$SGE_ROOT/ar/bin/runar ReserveGUI`
- Command line
 - > also uses **runar**
 - > **-help** shows options.
 - > `$SGE_ROOT/ar/bin/runar \`
`Reserve -a 12250730 \`
`-duration 1:30:0 \`
`-l graphics=1`



The screenshot shows the 'Advance Reservations' web interface. It has a title bar with 'Advance Reservations' and two tabs: 'New..' and 'List/Delete'. The main content area is divided into several sections:

- Connection:** A text field for 'Server Name' containing 'my_ar_server'.
- Reservation Details:** A section with a sub-header '* Required Fields'.
 - 'Reservation Title': An empty text input field.
 - '* Date': A date picker showing 'Nov 13, 2007'.
 - '* Start Time': A time picker showing '5 : 0' and a dropdown menu set to 'PM'.
 - '* Duration': A time picker showing '0 : 0'.
 - 'Hostname': A dropdown menu showing 'transform'.
 - '* Resources': An empty text input field with an 'Add' button to its right.
- Reservation Queue:** A text input field for 'Email Address'.
- Send Email:** A group of four checkboxes:
 - For Successful Reservation
 - When Reservation Started
 - When Reservation Complete
 - If Reservation Deleted
- Below the checkboxes is a time picker showing '0' and the text 'min. before reservation starts'.
- At the bottom are two buttons: 'Request Reservation' and 'Clear'.

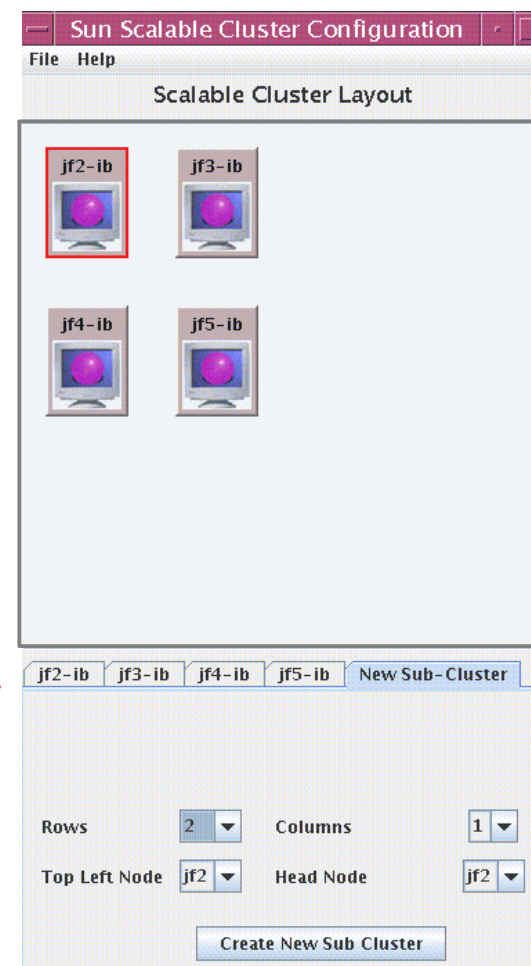
Using an Advance Reservation

- AR server confirms request and creates a queue.
 - > AR queue name such as **deans123456** is shown, E-mailed
 - > Specific to the reserved execution host and to the reserving user
 - > User can submit jobs to the reservation before or during its period
 - > `qssh -b no -q deans123456 /opt/VirtualGL/bin/vglrun my_app`
 - > Before the reservation's start time, resources are removed from the generic queue so others won't start using them.
 - > During the reservation period, jobs can run on the reservation queue, using reserved resources.
 - > After the reservation period, the resources are returned to the host's generic queue.
- Grid Engine 6.2 plans to provide its own AR facility.

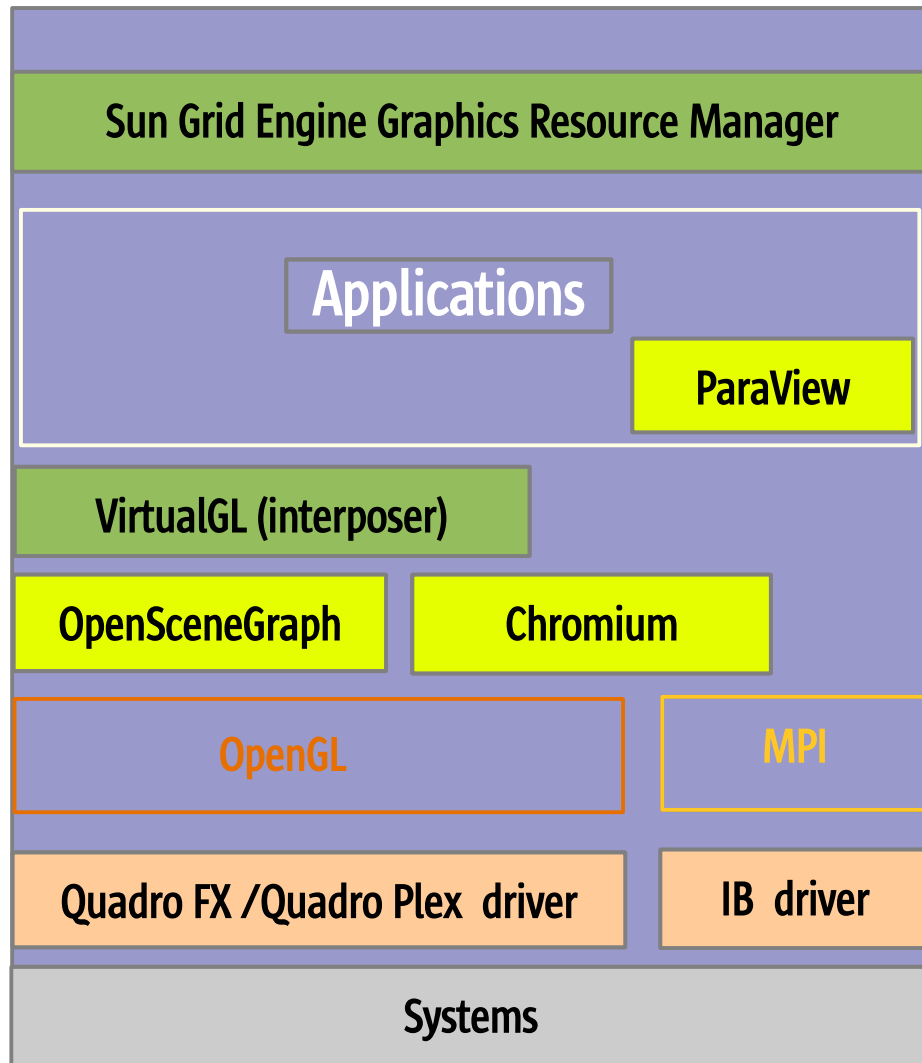
Integration with Scalable Visualization 1.1

Each graphics cluster is exposed as an **SGE queue**

- New GUI for admin to define a **subcluster**
 - > Scalable Viz configuration files for the subcluster are created and stored for use with Shared Viz
- Shared Visualization adds to SGE a subcluster "parallel environment" **sc** and its master job script.
 - > Submission for a 2x1 host (2x2 display) power wall:
 - > `qcrsh -b n -q head_2x1 $SGE_ROOT/graphics/sc/master cr_start.sh ...`
 - > The **master** script starts the Scalable Viz script (here, `cr_start.sh`) on the subcluster's head node.



Sun Visualization Stack



- **Shared Visualization software stack** - visualization services to a variety of remote clients
 - **SGE** - graphics resource management
 - **VirtualGL** - remote access via any client over standard IP networks
- **Scalable Visualization software**
 - **ParaView** - open-source parallel rendering application optimized for SMPs with multiple graphics.
 - **OpenSceneGraph** - open-source parallel rendering toolkit for building parallel applications.
 - **Chromium** - virtualized graphics devices for Solaris or Linux. Provides transparent parallelization for fill-rate limited applications; api for parallelizing applications by splitting up the data.
- **Quadro Plex** - connects graphics devices to Linux or Solaris servers over a PCI-E cable
- **Systems** - Sun Fire x86 & SPARC systems provide the most scalable platform

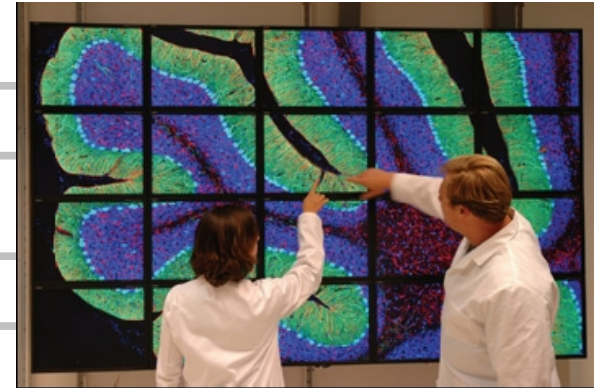


Dean Stanton, Advanced Visualization, Sun Microsystems

dean.stanton@sun.com

www.sun.com/visualization/

OpenSolaris.org/os/project/visualization-hpc/





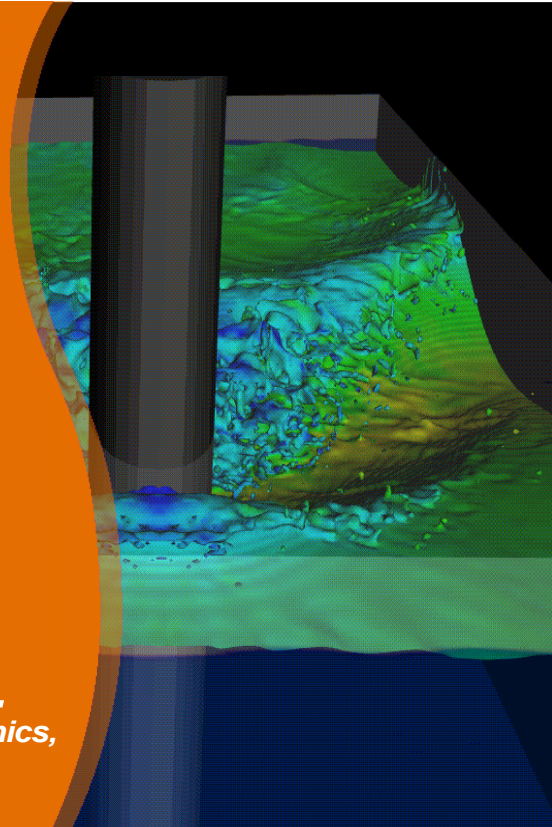
HPC Visualization On the Grid

W. Dean Stanton

*Sr. Staff Engineer, Advanced Visualization,
Sun Microsystems*

Linda Fellingham, Ph. D.

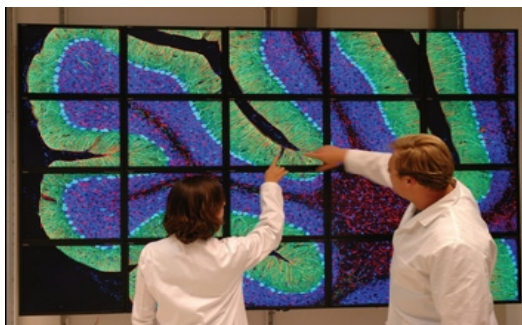
*Manager, Advanced Visualization and Graphics,
Sun Microsystems*



Understanding Huge Data Sets Requires Interactive Visualization

But visualization is hardware intensive

- Big data clogs networks
- Requires lots of memory
- Requires lots of CPU power
- Graphics accelerators need lots of power and cooling
- Workstations inadequate?
- And who wants to work near the heat and noise?

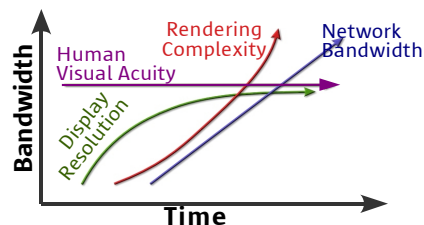


•Visualization is necessary . . .

•Collaboration is usually necessary . . .

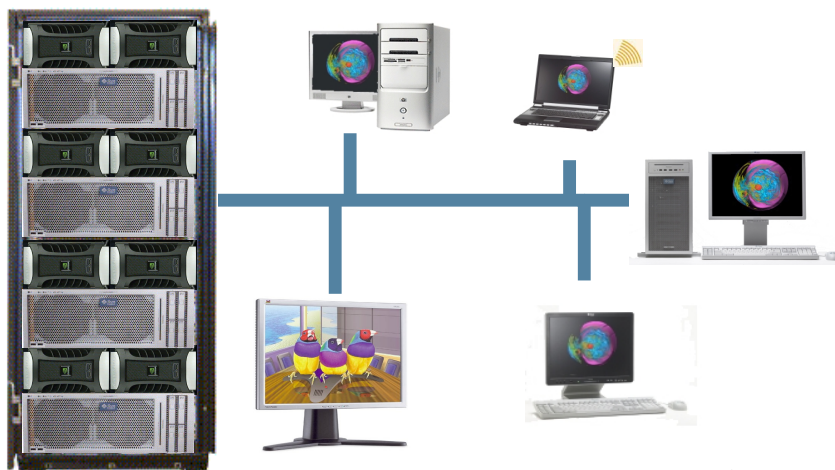
•Most of us have notebook computers, rather than bigger, faster workstations?

Change the Model



Share Visualization on the Grid over the Network

- Big Data
- Big Memory
- Secure
- Graphics Power
- CPU Power
- Keep heat and noise in the server room
- Send images over network
- Share the cost



HPC Visualization on the Grid

Open Source Grid and Cluster Conference

May 14, 2008

3

•Display resolution has increased, but now is increasing only slowly. Some of us even reduced our resolution when we switched from CRTs to flat panel displays.

•Human eye sight isn't getting any better. In fact, as the average worker ages, it may decrease a bit.

•But the data to be rendered (and the fidelity of the rendering) keep going up, up, up.

•Here's the good news: network bandwidth is still increasing rapidly.

•Therefore, it is now practical to transmit image sequences from where the data is and the compute power is, to where the user's eyes are.

•On the left is a grid of "graphics servers" – servers with adequate CPU and memory, connected to external graphics accelerators.

•On the right are clients sharing the power of the systems on the left. They are using the grid resources to run their applications, but are viewing and interacting with their applications from their desks or notebooks. (Here, they all have just one screen, but multiple screens are also feasible.)

•We'll talk first about scaling the systems and forming a graphics grid, and then about how clients use the grid for visualization – the software used for remote visualization and for grid management.



Agenda:

- **Scalable Visualization Solutions**
- **Shared Visualization Software**
- **Integration with Sun Grid Engine**

4

- **Sun builds "visualization solutions" comprised of these components, which we'll discuss in turn:**
- **Graphics servers and clusters, which run visualization applications.**
- **Our "Scalable Visualization Software" that allows graphics clusters to work as a single system on a visualization problem.**
- **Our "Shared Visualization Software" that remotes the visualization applications over the network, and provides collaboration support.**
- **And we'll talk about how we use Grid Engine to allocate and manage graphics resources.**

Sun Scalable Visualization Solutions

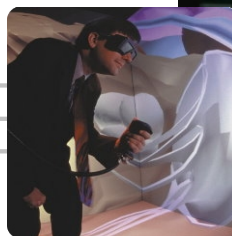
Graphics servers support multiple graphics devices to drive:

- Higher performance
- Higher image quality
- Higher resolution



Sun Fire Servers

HPC Visualization on the Grid



CAVEs

Open Source Grid and Cluster Conference



Power Walls,
Immersive Projections

May 14, 2008

5

•If you have a grid of graphics-capable servers in your machine room, you may also want to gang them together to drive a high-resolution power wall or immersive environment.

•Or multiple graphics accelerators can work together to improve application performance, or image quality.

Scalability Problem 1: Servers

- Servers have lots of processors and memory, but do not have adequate space, power, or cooling for a high-end 3D graphics accelerator
- Solution 1:
 - > Get the graphics card(s) out of the system
- Hardware Technology
 - > NVidia's Quadro® Plex Visual Computing Systems



HPC Visualization on the Grid

Open Source Grid and Cluster Conference

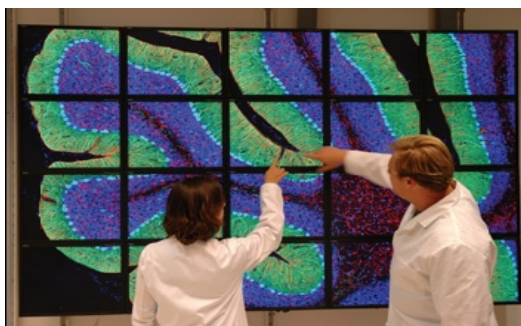
May 14, 2008

Scalability Problem 2: Lots of Screens

- Need to view applications across many screens, in order to view adequate detail
- Solution 2:
 - > Distribute the rendering across many systems (sort first)
 - > High-bandwidth, low-latency interconnects (InfiniBand or 10gigE)
- Open Source Software: Chromium or OpenSceneGraph



HPC Visualization on the Grid



Open Source Grid and Cluster Conference

May 14, 2008

•More screens than a single-system can drive.

Scalability Problem 3: Performance

- Performance can be too slow for very large data sets
- Solution 3
 - > Break the problem up and distribute the rendering to multiple render nodes, and reassemble on the head node (sort last)
- Open Source Software Technologies:
 - > Paraview (Parallel Visualization Application) or Chromium (with work)



Render Nodes

Head Node

HPC Visualization on the Grid

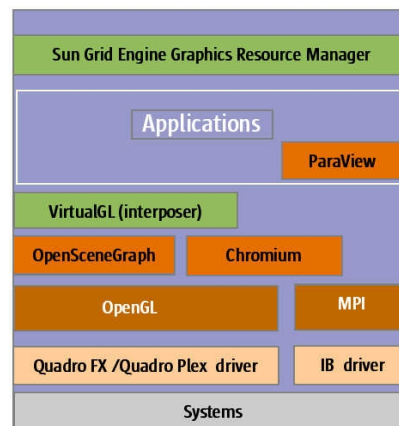
Open Source Grid and Cluster Conference

May 14, 2008

8

Software Details

- Scalable Visualization 1.1 software supports:
 - > Red Hat Linux (RHEL 4U2-5 and RHEL 5/5.1)
 - > SLES 10
 - > Solaris 10 Update 3 or 4
- Complete Open Source software stack
 - > Chromium, MVAPICH2, OFED, OpenSceneGraph, Paraview
 - > Sun added value:
 - > Pre-built binaries, tested for interoperability
 - > Installation scripts and configuration files
 - > Wrappers for greater ease of use
 - > MPI protocol added as a Chromium interconnect
 - > Supported on Sun hardware
 - > Free download for Solaris at OpenSolaris.org/os/project/visualization-hpc/

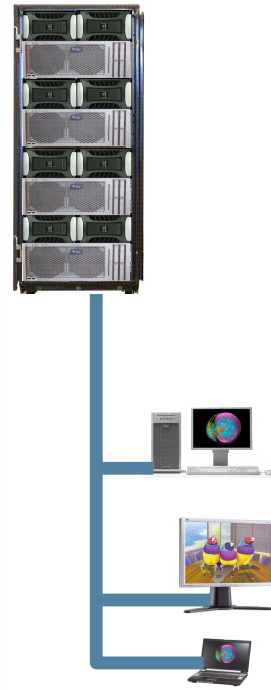


- Suse Linux Enterprise System
- Open Fabric Enterprise Distribution (OFED) 1.2
- MVA-PITCH-2 Message Passing implementation
- Open Scene Graph
- Paraview

•You could collect all this open source, compile it, and see if the versions you chose will work together, but we've built and tested them for you.

•Often easy to get this open source software to work for some simple demo, but then it fails for real world applications or data sets.

Sun Shared Visualization Software



10

The Shared Visualization Model

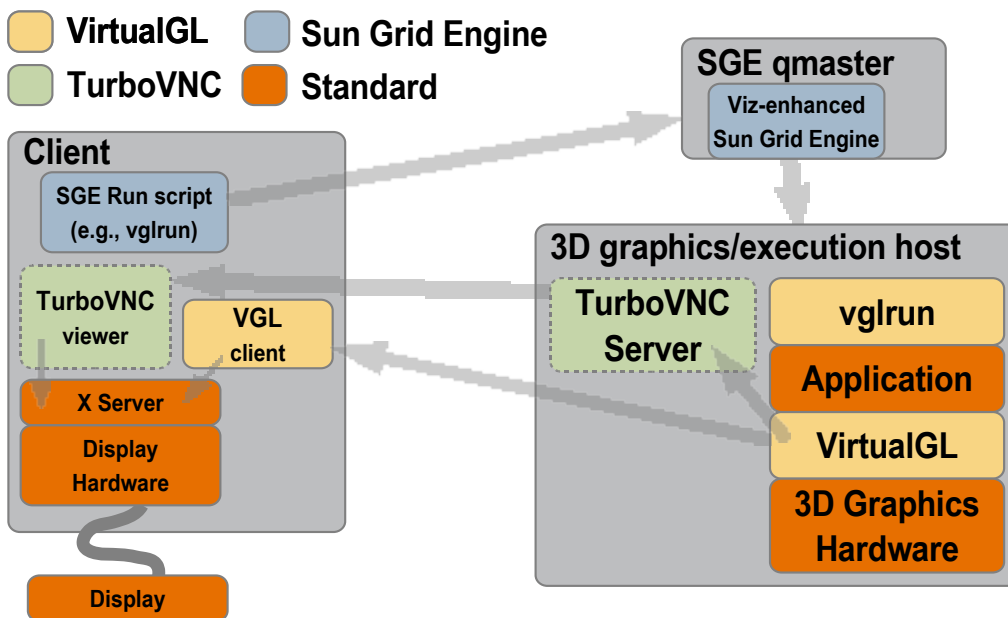
Feature	Benefit
Secure	Data stays on the server. Control data access, even among 3D application users.
Performant	Optimized compression and decompression transfers visually lossless images at interactive speeds (more than 20 frames/sec).
Interoperable	Client only needs enough network performance and a display. Interoperable with a variety of devices
Sharable	Average CPU, memory, and graphics needs over many users. Reduce total cost of ownership
Scalable	A single user can access lots of CPU, memory, and attached graphics. Get more resources than possible in ANY workstation
Flexible	Graphics computation and display technology are separated. Display on what you already have, upgrade graphics separately
Load Balancing	Better utilization of compute and graphics resources. Grid software helps to find and manage resources.

•Why would anyone do such a thing? Here are the reasons . . .

• Can also run legacy SPARC Solaris 3D applications on your x64 or SunRay client

Sun Shared Visualization Software

Transparent Remote Access to 3D Applications



HPC Visualization on the Grid

Open Source Grid and Cluster Conference

May 14, 2008

12

- We're going to talk about these components and what they do, but here's the big picture.

- An application running on a shared central server thinks it is running on incredibly well provisioned workstation.

- Transparent middleware moves images from the graphics server to the client, typically with optimized, visually-lossless compression.

- The client decompresses the images and displays them.

- And the user interacts with windows holding remote 3D content just as if the application were running locally.

- We have two technologies to move the images to the client: VirtualGL and TurboVNC, which we'll discuss next.

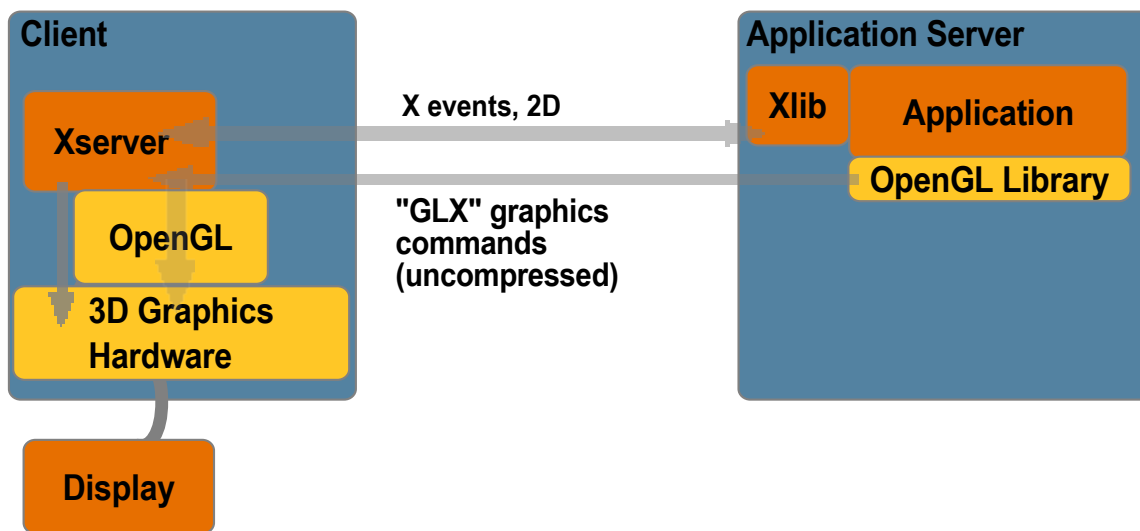
- Grid Engine allocates to the user a lightly-loaded server and a lightly-loaded graphics accelerator on that server.

- This allows the graphics servers and their graphics boards to be shared among simultaneous users.

VirtualGL

- Purpose
 - > Allows OpenGL applications which were designed to run and display on the same system to efficiently and transparently run on one system and display on another.
- Components
 - > "Middleware" software for Linux and Solaris servers
 - > Platform-optimized Image compression technology
 - > Various data transport methods
 - > vglclient program decompresses and displays the images
 - > Clients for Solaris, Linux, Windows, Mac OS X
 - > Sun Ray ultra thin clients
- Open source software project sponsored by Sun

Remote Graphics In the Past



HPC Visualization on the Grid

Open Source Grid and Cluster Conference

May 14, 2008

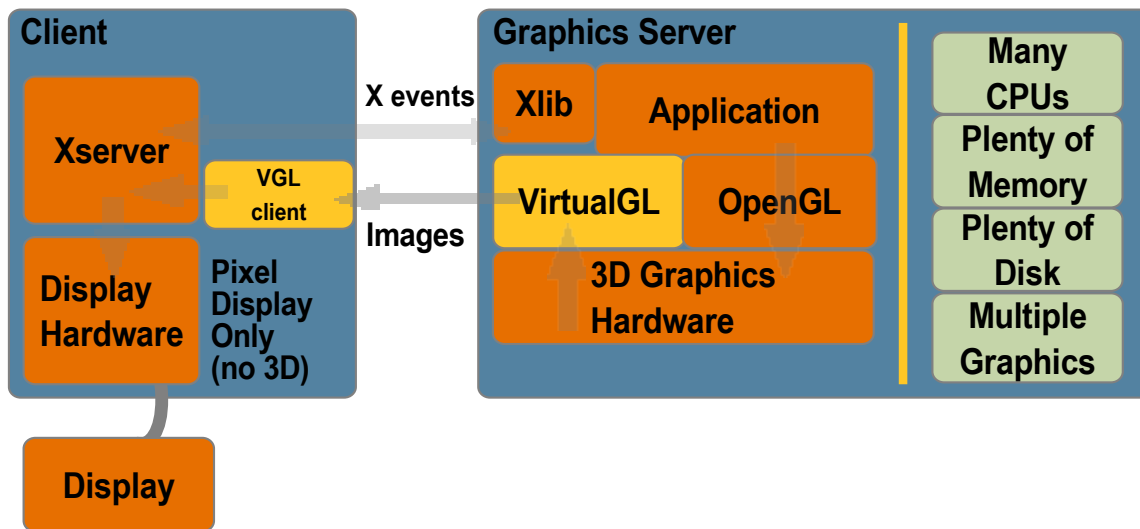
14

- Using PEX and OpenGL, you've been able to remote your graphics from an application server to your client, but . . .
- The server needed the graphics library to know how to produce the "GLX" (or PEX) protocol.
- The client needs to have the graphics accelerator, as it does all the rendering (drawing).
- All the graphics go over the network, and they're all uncompressed. This works okay if your application uses display lists "retained graphics", but the latency and bandwidth is quite a performance problem for "immediate mode" graphics programs.

Remote Graphics Using VirtualGL

client% `vglconnect my_server`

server% `vglrun my_application`



HPC Visualization on the Grid

Open Source Grid and Cluster Conference

May 14, 2008

15

• Using VirtualGL solves the problems described above:

- The Graphics Server uses its local graphics accelerator for optimum rendering performance.

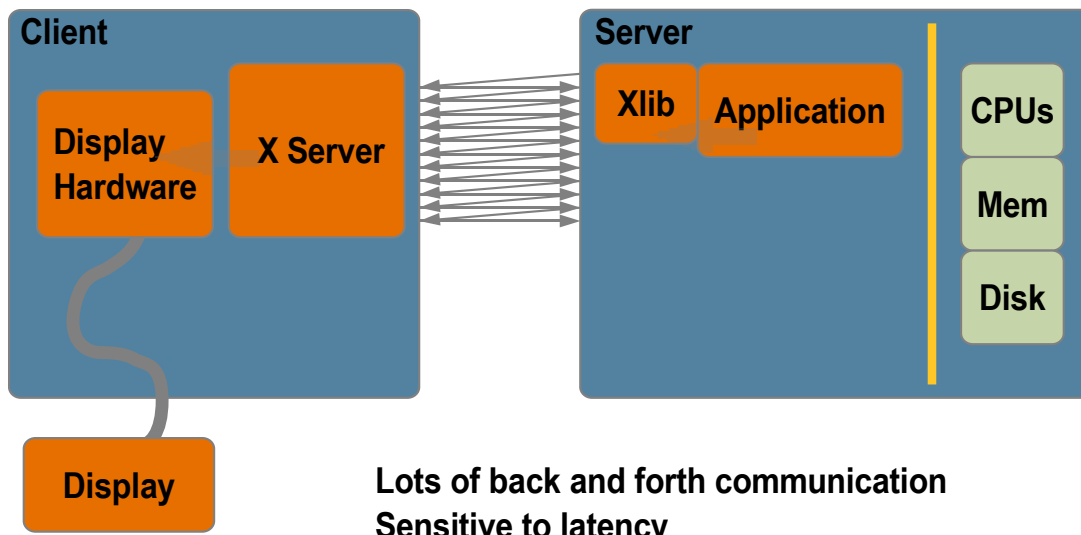
- VirtualGL reads from the graphics hardware the resultant images, compresses them, and sends them over the network.

- The VirtualGL client, `vglclient`, receives the images, decompresses and displays them.

The client needs to display Red-Green-Blue pixels, but doesn't need any 3D hardware at all.

- The Graphics Server can be shared among many clients, both serially and simultaneously.

Standard Remote X



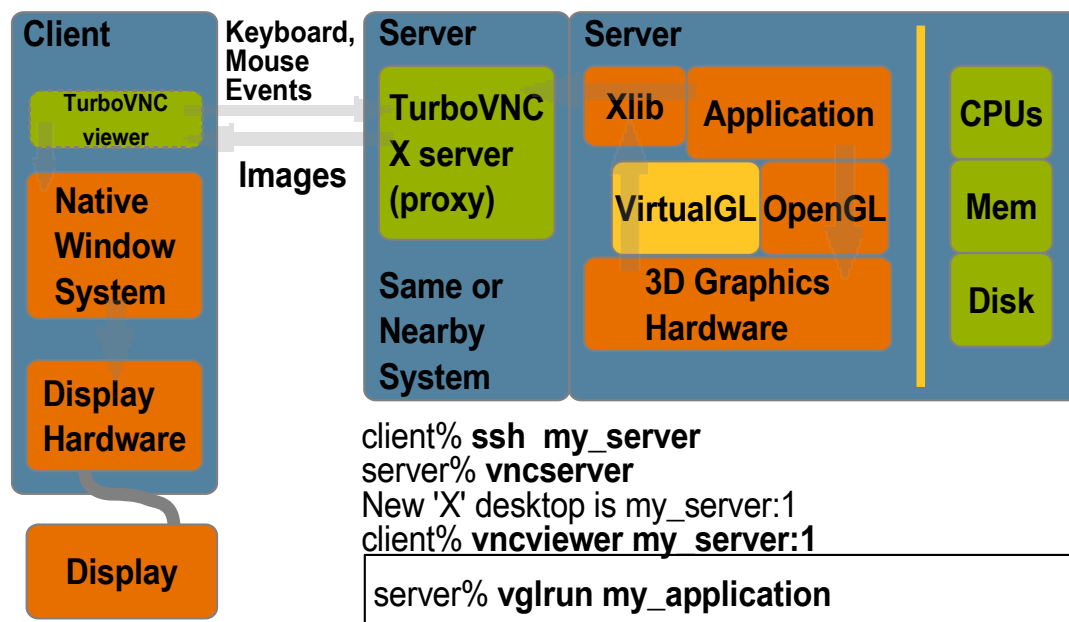
- "Remote X has lots of back-and-forth communication between application and X server, so latency is a big problem"?

TurboVNC

Latency Tolerant Remote Visualization with Collaboration

- Purpose
 - > Allows X applications which were designed to run and display on the same system (or on low latency networks) to transparently run on one system and display on another.
 - > Also enables collaboration by allowing more than one client system to display the X session.
- Open source project sponsored by Sun
 - > Derived from TightVNC but uses same optimized image compression technology as VirtualGL.
 - > Interoperable with other VNC viewers, including Java-based WebVNC

VirtualGL With TurboVNC



HPC Visualization on the Grid

Open Source Grid and Cluster Conference

May 14, 2008

18

- **VirtualGL reads back images from the graphics accelerator, but doesn't compress them. It gives them to the X proxy (TurboVNC server), probably on the same server, and probably using shared memory.**
- **TurboVNC compresses the images using the same optimized "codex" as VirtualGL used above.**
- **TurboVNC transmits the compressed images to the client (or clients).**
- **Here are the commands used. The client connects to the graphics server, starts the VNC server, connects at least one client to the VNC server, and within the TurboVNC session, invokes graphics applications using vglrun (so that VirtualGL will read back the images).**



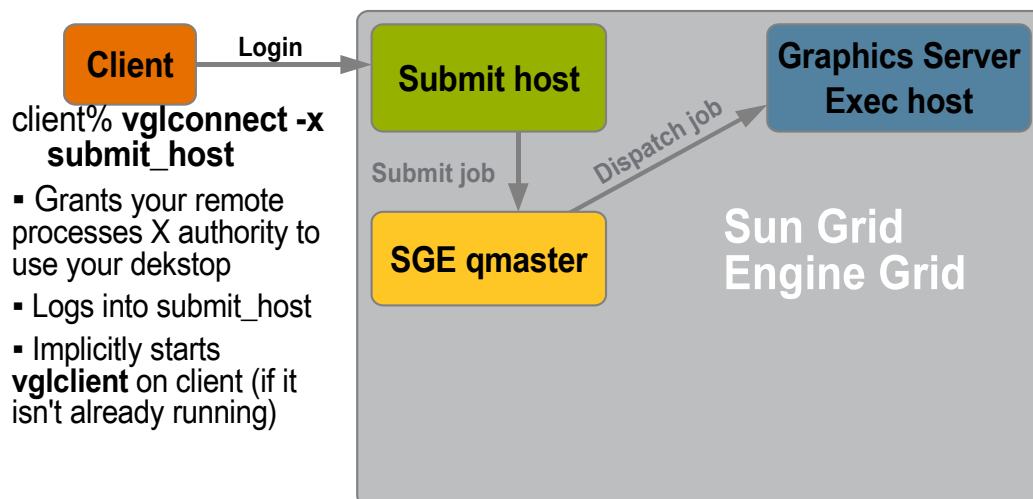
Integration with Sun Grid Engine

19

Sun Grid Engine (With Shared Viz Enhancements)

- Purpose
 - > Lets users share graphics servers
 - > SGE assures available CPU, memory, and graphics resources
- Components
 - > Standard SGE provides management and allocation of regular compute resources (CPUs, memory, OS, software licenses).
 - > Enhancements allow SGE to manage graphics resources
 - > provides “user-transparent” connection between the allocated graphics device and the user’s display on the remote client.
 - > Advance Reservation system allows resources to be reserved for a specific time in the future.
- Open source software developed by Sun

Sun Grid Engine With Shared Visualization



```
client% vglconnect -x
submit_host
```

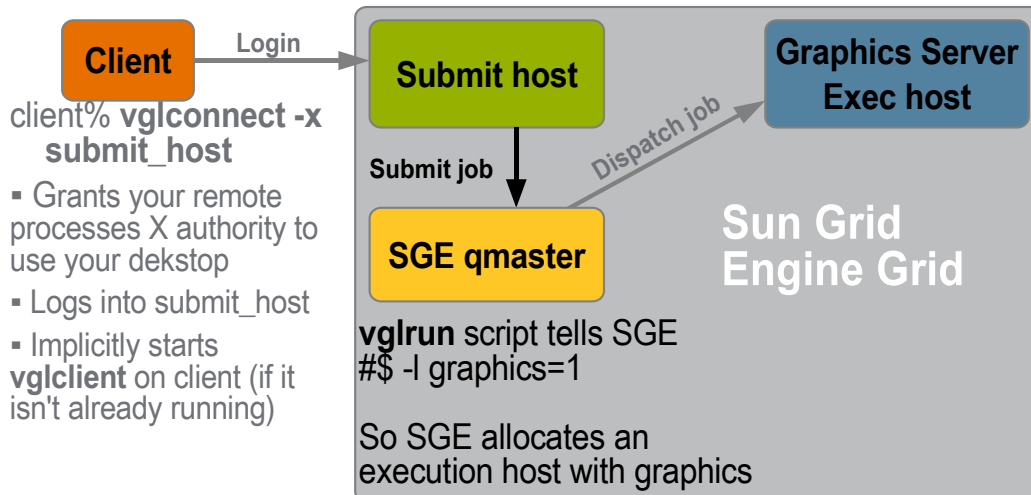
- Grants your remote processes X authority to use your dekstop
- Logs into submit_host
- Implicitly starts **vglclient** on client (if it isn't already running)

•If the client host is a Grid Engine submit host, it is even easier.

Client Tasks	Sun Grid Engine & VirtualGL (Server) Tasks												
1. User permits her remote processes access to her client X display using <code>vglconnect -k</code> (or <code>vglconnect -x submit-host</code>). <code>vglconnect</code> implicitly starts <code>vglclient</code> on client													
2. Use AR client to make reservation (<i>optional</i>)	If AR Server confirms reservation, it makes queue susieq123												
3. User submits graphics job (<code>qsub</code> or <code>qrsh</code>)	4. SGE saves options, environment variables, script with job.												
	5. Queue master schedules job & dispatches it to execution host												
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #E6E6FA;"><i>queue</i></th> <th style="background-color: #E6E6FA;"><i>host1</i></th> <th style="background-color: #E6E6FA;"><i>host2</i></th> <th style="background-color: #E6E6FA;"><i>host3</i></th> </tr> </thead> <tbody> <tr> <td style="background-color: #E6E6FA;">all.q</td> <td style="background-color: #E6E6FA;">all.q@host1</td> <td style="background-color: #E6E6FA;">all.q@host2</td> <td style="background-color: #E6E6FA;">all.q@host3</td> </tr> <tr> <td style="background-color: #E6E6FA;">susieq123</td> <td colspan="3" style="background-color: #E6E6FA;">susieq123@host1</td> </tr> </tbody> </table>	<i>queue</i>	<i>host1</i>	<i>host2</i>	<i>host3</i>	all.q	all.q@host1	all.q@host2	all.q@host3	susieq123	susieq123@host1		
<i>queue</i>	<i>host1</i>	<i>host2</i>	<i>host3</i>										
all.q	all.q@host1	all.q@host2	all.q@host3										
susieq123	susieq123@host1												
	6. Execution host starts SGE job												
	7. Graphics <code>starter</code> script allocates graphics device to job												
	8. <code>vglrun</code> runs graphics application under VirtualGL control												
9. Application window appears on client's X desktop	9. Graphics application opens window on client \$DISPLAY												
	10. VirtualGL connects <code>vglclient</code> to application on server												
11. <code>vglclient</code> receives pixels & displays on Client	11. VirtualGL transmits completed images to client												
12. User interacts with application													
13. User terminates application													
	14. Graphics <code>epilog</code> script releases graphics device												

Sun Grid Engine With Shared Visualization

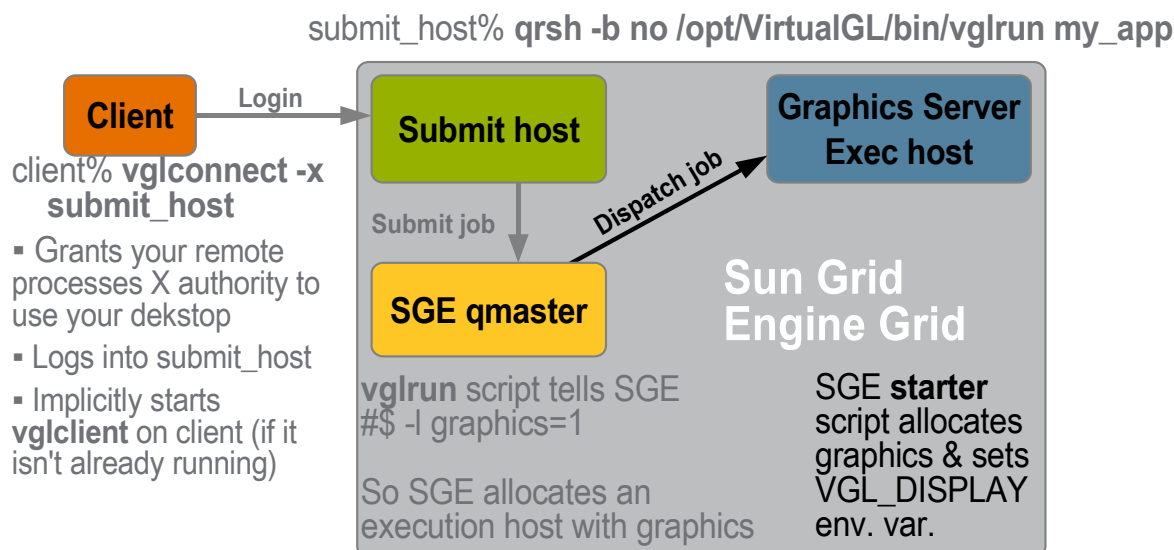
submit_host% `qrsh -b no /opt/VirtualGL/bin/vglrun my_app`



• **Make an Advance Reservation, if desired (that's optional)**

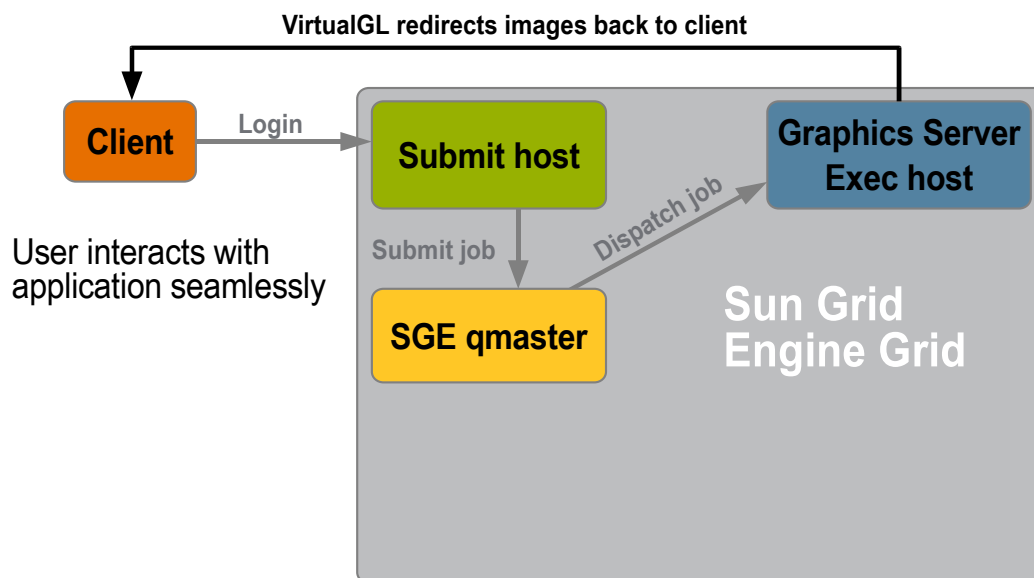
Client Tasks	Sun Grid Engine & VirtualGL (Server) Tasks												
1. User permits her remote processes access to her client X display using <code>vglconnect -k</code> (or <code>vglconnect -x submit-host</code>). <code>vglconnect</code> implicitly starts <code>vglclient</code> on client													
2. Use AR client to make reservation (<i>optional</i>)	If AR Server confirms reservation, it makes queue susieq123												
3. User submits graphics job (<code>qsub</code> or <code>qrsh</code>)	4. SGE saves options, environment variables, script with job.												
	5. Queue master schedules job & dispatches it to execution host												
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #E6E6FA;">queue</th> <th style="background-color: #E6E6FA;">host1</th> <th style="background-color: #E6E6FA;">host2</th> <th style="background-color: #E6E6FA;">host3</th> </tr> </thead> <tbody> <tr> <td style="background-color: #E6E6FA;">all.q</td> <td style="background-color: #E6E6FA;">all.q@host1</td> <td style="background-color: #E6E6FA;">all.q@host2</td> <td style="background-color: #E6E6FA;">all.q@host3</td> </tr> <tr> <td style="background-color: #E6E6FA;">susieq123</td> <td colspan="3" style="background-color: #E6E6FA;">susieq123@host1</td> </tr> </tbody> </table>	queue	host1	host2	host3	all.q	all.q@host1	all.q@host2	all.q@host3	susieq123	susieq123@host1		
queue	host1	host2	host3										
all.q	all.q@host1	all.q@host2	all.q@host3										
susieq123	susieq123@host1												
	6. Execution host starts SGE job												
	7. Graphics <code>starter</code> script allocates graphics device to job												
	8. <code>vglrun</code> runs graphics application under VirtualGL control												
9. Application window appears on client's X desktop	9. Graphics application opens window on client \$DISPLAY												
	10. VirtualGL connects <code>vglclient</code> to application on server												
11. <code>vglclient</code> receives pixels & displays on Client	11. VirtualGL transmits completed images to client												
12. User interacts with application													
13. User terminates application													
	14. Graphics <code>epilog</code> script releases graphics device												

Sun Grid Engine With Shared Visualization



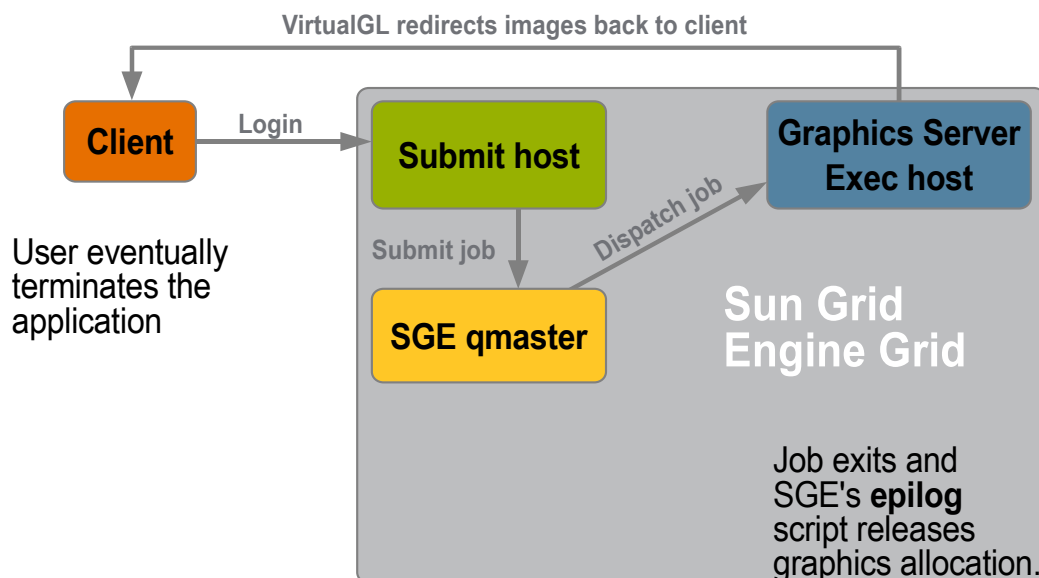
Client Tasks	Sun Grid Engine & VirtualGL (Server) Tasks												
1. User permits her remote processes access to her client X display using <code>vglconnect -k</code> (or <code>vglconnect -x submit-host</code>). <code>vglconnect</code> implicitly starts <code>vglclient</code> on client													
2. Use AR client to make reservation (<i>optional</i>)	If AR Server confirms reservation, it makes queue susieq123												
3. User submits graphics job (<code>qsub</code> or <code>qrsh</code>)	4. SGE saves options, environment variables, script with job.												
	5. Queue master schedules job & dispatches it to execution host												
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #ADD8E6;"><i>queue</i></th> <th style="background-color: #ADD8E6;"><i>host1</i></th> <th style="background-color: #ADD8E6;"><i>host2</i></th> <th style="background-color: #ADD8E6;"><i>host3</i></th> </tr> </thead> <tbody> <tr> <td style="background-color: #ADD8E6;">all.q</td> <td style="background-color: #ADD8E6;">all.q@host1</td> <td style="background-color: #ADD8E6;">all.q@host2</td> <td style="background-color: #ADD8E6;">all.q@host3</td> </tr> <tr> <td style="background-color: #ADD8E6;">susieq123</td> <td colspan="3" style="background-color: #ADD8E6;">susieq123@host1</td> </tr> </tbody> </table>	<i>queue</i>	<i>host1</i>	<i>host2</i>	<i>host3</i>	all.q	all.q@host1	all.q@host2	all.q@host3	susieq123	susieq123@host1		
<i>queue</i>	<i>host1</i>	<i>host2</i>	<i>host3</i>										
all.q	all.q@host1	all.q@host2	all.q@host3										
susieq123	susieq123@host1												
	6. Execution host starts SGE job												
	7. Graphics starter script allocates graphics device to job												
	8. <code>vglrun</code> runs graphics application under VirtualGL control												
9. Application window appears on client's X desktop	9. Graphics application opens window on client <code>\$DISPLAY</code>												
	10. VirtualGL connects <code>vglclient</code> to application on server												
11. <code>vglclient</code> receives pixels & displays on Client	11. VirtualGL transmits completed images to client												
12. User interacts with application													
13. User terminates application													
	14. Graphics epilog script releases graphics device												

Sun Grid Engine With Shared Visualization



Client Tasks	Sun Grid Engine & VirtualGL (Server) Tasks												
1. User permits her remote processes access to her client X display using <code>vglconnect -k</code> (or <code>vglconnect -x submit-host</code>). <code>vglconnect</code> implicitly starts <code>vglclient</code> on client													
2. Use AR client to make reservation (optional)	If AR Server confirms reservation, it makes queue susieq123												
3. User submits graphics job (<code>qsub</code> or <code>qrsh</code>)	4. SGE saves options, environment variables, script with job.												
	5. Queue master schedules job & dispatches it to execution host												
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #E6E6FA;">queue</th> <th style="background-color: #E6E6FA;">host1</th> <th style="background-color: #E6E6FA;">host2</th> <th style="background-color: #E6E6FA;">host3</th> </tr> </thead> <tbody> <tr> <td style="background-color: #E6E6FA;">all.q</td> <td style="background-color: #E6E6FA;">all.q@host1</td> <td style="background-color: #E6E6FA;">all.q@host2</td> <td style="background-color: #E6E6FA;">all.q@host3</td> </tr> <tr> <td style="background-color: #E6E6FA;">susieq123</td> <td colspan="3" style="background-color: #E6E6FA;">susieq123@host1</td> </tr> </tbody> </table>	queue	host1	host2	host3	all.q	all.q@host1	all.q@host2	all.q@host3	susieq123	susieq123@host1		
queue	host1	host2	host3										
all.q	all.q@host1	all.q@host2	all.q@host3										
susieq123	susieq123@host1												
	6. Execution host starts SGE job												
	7. Graphics <code>starter</code> script allocates graphics device to job												
	8. <code>vglrun</code> runs graphics application under VirtualGL control												
9. Application window appears on client's X desktop	9. Graphics application opens window on client \$DISPLAY												
	10. VirtualGL connects <code>vglclient</code> to application on server												
11. <code>vglclient</code> receives pixels & displays on Client	11. VirtualGL transmits completed images to client												
12. User interacts with application													
13. User terminates application													
	14. Graphics <code>epilog</code> script releases graphics device												

Sun Grid Engine With Shared Visualization



Client Tasks	Sun Grid Engine & VirtualGL (Server) Tasks												
1. User permits her remote processes access to her client X display using <code>vglconnect -k</code> (or <code>vglconnect -x submit-host</code>). <code>vglconnect</code> implicitly starts <code>vglclient</code> on client													
2. Use AR client to make reservation (<i>optional</i>)	If AR Server confirms reservation, it makes queue susieq123												
3. User submits graphics job (<code>qsub</code> or <code>qrsh</code>)	4. SGE saves options, environment variables, script with job.												
	5. Queue master schedules job & dispatches it to execution host												
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #D9E1F2;"><i>queue</i></th> <th style="background-color: #D9E1F2;"><i>host1</i></th> <th style="background-color: #D9E1F2;"><i>host2</i></th> <th style="background-color: #D9E1F2;"><i>host3</i></th> </tr> </thead> <tbody> <tr> <td style="background-color: #D9E1F2;"><code>all.q</code></td> <td style="background-color: #D9E1F2;"><code>all.q@host1</code></td> <td style="background-color: #D9E1F2;"><code>all.q@host2</code></td> <td style="background-color: #D9E1F2;"><code>all.q@host3</code></td> </tr> <tr> <td style="background-color: #D9E1F2;"><code>susieq123</code></td> <td colspan="3" style="background-color: #D9E1F2;"><code>susieq123@host1</code></td> </tr> </tbody> </table>	<i>queue</i>	<i>host1</i>	<i>host2</i>	<i>host3</i>	<code>all.q</code>	<code>all.q@host1</code>	<code>all.q@host2</code>	<code>all.q@host3</code>	<code>susieq123</code>	<code>susieq123@host1</code>		
<i>queue</i>	<i>host1</i>	<i>host2</i>	<i>host3</i>										
<code>all.q</code>	<code>all.q@host1</code>	<code>all.q@host2</code>	<code>all.q@host3</code>										
<code>susieq123</code>	<code>susieq123@host1</code>												
	6. Execution host starts SGE job												
	7. Graphics <code>starter</code> script allocates graphics device to job												
	8. <code>vglrun</code> runs graphics application under VirtualGL control												
9. Application window appears on client's X desktop	9. Graphics application opens window on client <code>\$DISPLAY</code>												
	10. VirtualGL connects <code>vglclient</code> to application on server												
11. <code>vglclient</code> receives pixels & displays on Client	11. VirtualGL transmits completed images to client												
12. User interacts with application													
13. User terminates application													
	14. Graphics <code>epilog</code> script releases graphics device												

Grid Engine Application Script

Script Uses `#$` Comments to Describe its Needs to Grid Engine

- Grid Engine will allocate acceptable host and graphics
 > `qcrsh -b no my_script` > `qsub -now y my_script`

```
#!/bin/sh
#$ -N my_app          ← Name of Grid Engine job
#$ -l graphics=1     ← Job requires a graphics card
#$ -l arch=lx24-x86 | lx24-amd64 ← Required Architecture(s)
# Here, required architecture is "Linux (with 2.4 or 2.6 kernel) on x86 or x64"
#$ -v DISPLAY, SSH_CLIENT ← Save environment vars. with job
# These variables enable VirtualGL to find the client's X display (desktop).
# Invoke application using VirtualGL to route 3D to allocated graphics card
vglrun my_application "$@" ← Invoke app with any args
# Script invokes VirtualGL implicitly, so submitter need not mention vglrun
```

- Grid Engine saves as part of the job the **DISPLAY** or **SSH_CLIENT** environment variable values of the submitter, and will give those values to the job when it starts.

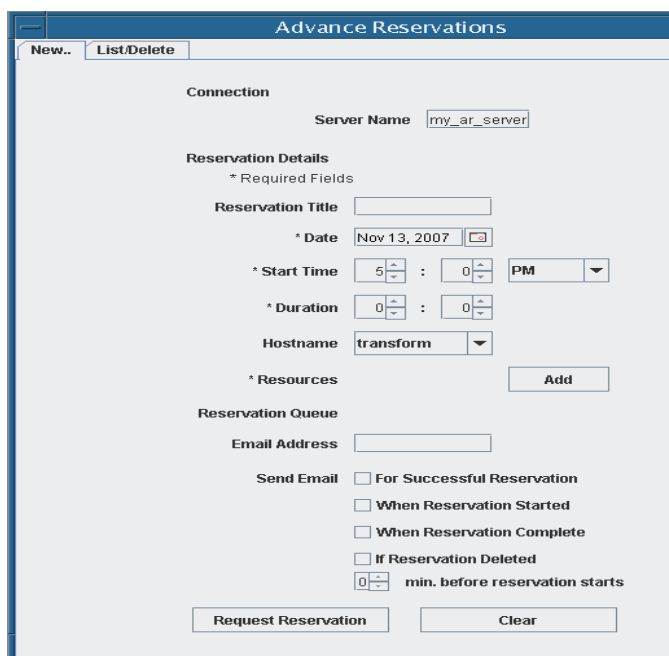
- VirtualGL will use the **DISPLAY** or **SSH_CLIENT** environment variable values to route the 3D images back to the user.

- This script always invokes `my_application` using VirtualGL, which uses the graphics accelerator allocated by SGE in response to `graphics=1` and it will use **DISPLAY** or **SSH_CLIENT** to route the 3D images back to the user.

- (We also provide a sample run script, which is slightly more sophisticated.)

Making an Advance Reservation

- GUI invoked with `runar` script (that invokes Java)
 - > `$SGE_ROOT/ar/bin/runar ReserveGUI`
- Command line
 - > also uses `runar`
 - > `-help` shows options.
 - > `$SGE_ROOT/ar/bin/runar \ Reserve -a 12250730 \ -duration 1:30:0 \ -l graphics=1`



•The Shared Visualization 1.1 release adds control over when E-mail is sent to you by the AR server. The E-mail includes the queue name, to which you can submit jobs.

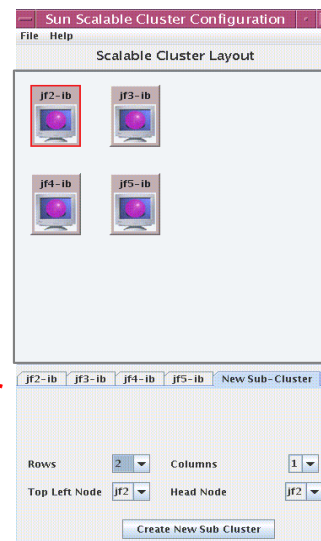
Using an Advance Reservation

- AR server confirms request and creates a queue.
 - > AR queue name such as **deans123456** is shown, E-mailed
 - > Specific to the reserved execution host and to the reserving user
 - > User can submit jobs to the reservation before or during its period
 - > **qrsh -b no -q deans123456 /opt/VirtualGL/bin/vglrun my_app**
 - > Before the reservation's start time, resources are removed from the generic queue so others won't start using them.
 - > During the reservation period, jobs can run on the reservation queue, using reserved resources.
 - > After the reservation period, the resources are returned to the host's generic queue.
- Grid Engine 6.2 plans to provide its own AR facility.

Integration with Scalable Visualization 1.1

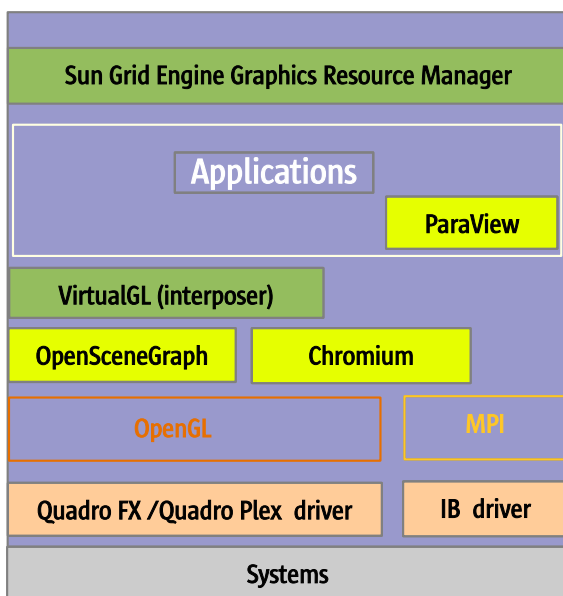
Each graphics cluster is exposed as an **SGE queue**

- New GUI for admin to define a **subcluster**
 - > Scalable Viz configuration files for the subcluster are created and stored for use with Shared Viz
- Shared Visualization adds to SGE a subcluster "parallel environment" **sc** and its master job script.
 - > Submission for a 2x1 host (2x2 display) power wall:
 - > `qsh -b n -q head_2x1 $SGE_ROOT/graphics/sc/master cr_start.sh ...`
 - > The **master** script starts the Scalable Viz script (here, `cr_start.sh`) on the subcluster's head node.



- **Grid engine knows which hosts are busy and therefore which clusters can be allocated.**

Sun Visualization Stack



- **Shared Visualization software stack** - visualization services to a variety of remote clients
 - **SGE** - graphics resource management
 - **VirtualGL** - remote access via any client over standard IP networks
- **Scalable Visualization software**
 - **ParaView** - open-source parallel rendering application optimized for SMPs with multiple graphics.
 - **OpenSceneGraph** - open-source parallel rendering toolkit for building parallel applications.
 - **Chromium** - virtualized graphics devices for Solaris or Linux. Provides transparent parallelization for fill-rate limited applications; api for parallelizing applications by splitting up the data.
- **Quadro Plex** - connects graphics devices to Linux or Solaris servers over a PCI-E cable
- **Systems** - Sun Fire x86 & SPARC systems provide the most scalable platform

Summary of all you've learned:

Sun's "Scalable Visualization Systems" are (from the bottom up) . . .

- **Systems** (workstations or servers), which can run graphics applications
- **With graphics accelerator boards inside the workstation or attached to the server using NVidia's QuadroPlex VCS systems (bus extender boxes)**
- **Open Source "Scalable Visualization Software" (shown in yellow), which Sun builds, tests, and makes easier to use**
- **Sun's "Shared Visualization Software" which uses**
 - **Open Source VirtualGL to transmit efficiently the images from accelerated 3D rendering to remoteLAN users.**
 - and
 - **Open Source TurboVNC to do the same for Wide Area Network users and for collaborative sessions.**
 - and
 - **Sun Grid Engine extensions to manage graphics resources (as well)**
- **And Sun will sell, build, install, and test the whole thing.**

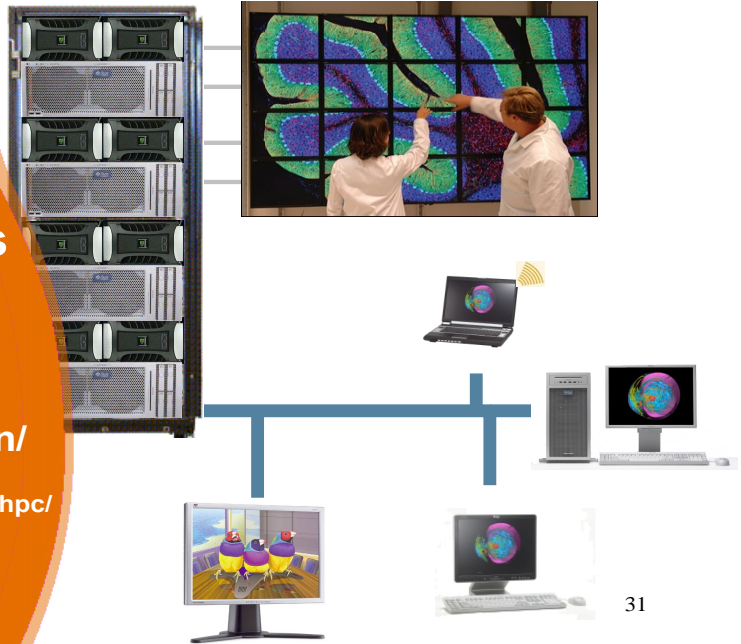


**Dean Stanton,
Advanced
Visualization,
Sun Microsystems**

dean.stanton@sun.com

www.sun.com/visualization/

OpenSolaris.org/os/project/visualization-hpc/



- You can use these Visualization Solutions to drive a power wall, immersive environment (CAVE), and/or run applications for remote users over your LAN or across a higher-latency WAN.