



KATHOLIEKE UNIVERSITEIT  
**LEUVEN**

universiteit  
**hasselt**  
KNOWLEDGE IN ACTION



# Visualization @HPC KU Leuven

Mag Selwa

ICTS, Leuven



27 October 2015

# Scientific Visualization

“The purpose of computing is insight not numbers.”

R. W. Hamming (1961)

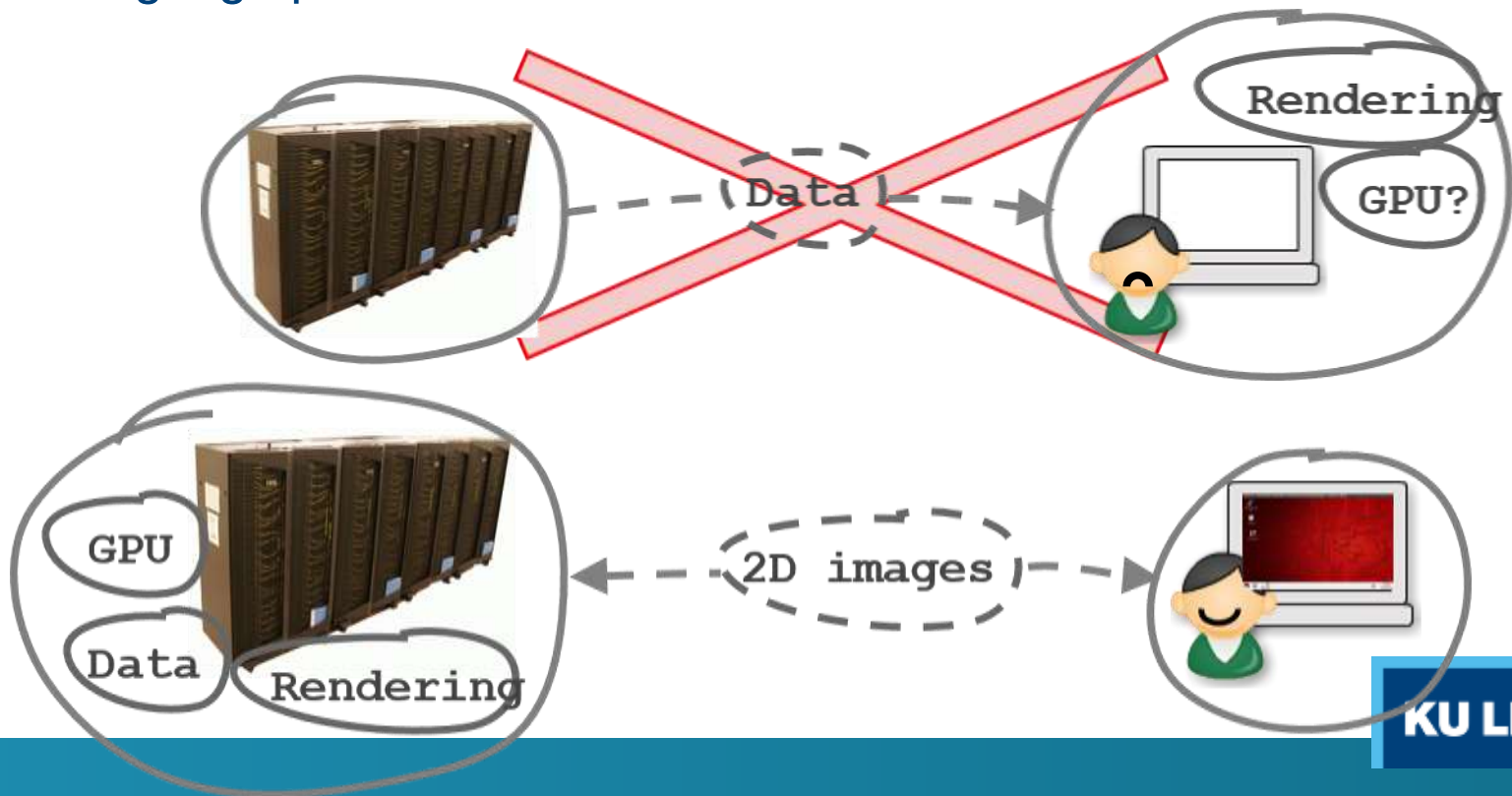
# Remote Visualization concepts

- Remote Visualization techniques address the problem of providing system and applications enabling remote users to interactively visualize large data sets residing on centralized data infrastructure.
  - Data could be collected and shared centrally or produced on HPC systems by numerical simulations codes.
  - Remote users could possibly connect from home (low speed networks).
- The traditional approach to visualization has been to transfer the remote data to a local workstation and then run locally a visualization application. This approach has several drawbacks:
  - The local system has to be powerful enough to handle data
  - The latency is concentrated into the initial data transfer step
  - The need of explicit complete data transfer prevent tight coupling between simulation and visualization applications

# HPC Visualization concepts

Perform scientific visualization on large amounts of data produced on HPC systems

- without moving data
- using high performance machine



# Overview

- VSC infrastructure
- Visualization possibilities @ HPC KU Leuven
  - X11
  - NX
  - Paraview remote visualization
- Visualization node
  - Why the best choice
  - How to start
- Demo

# VSC infrastructure



# VSC Tier-2

- UAntwerpen
  - 2 clusters, 336 nodes 4492 cores -> 90 Tflops
  - + Fat nodes
- VUB
  - 3 clusters, 274 nodes -> 14 TFlops
  - + Grid specialization
- UGent
  - 8 clusters, 627 nodes, 10184 cores -> 228 TFlops
  - + BigData specialization
- KU Leuven
  - 4 clusters, 256 nodes, 5312 cores -> 140 TFlops
  - + shared memory (640 cores, additional 12 TFlops)
  - + accelerator specialization (additional 25 nodes)
  - + visualization nodes

# KU Leuven/UHasselt Tier 2

*ThinKing*

*Cerebro*

4

8

8

5

176+32

140 TF

48

12 TF

2x10-core  
Ivy Bridge  
64/128  
GB

2x12-core  
Haswell  
64 GB

64x10-core  
Ivy Bridge  
14 TB

SAS  
21.8 TB

2x448  
CUDA cores  
2x5 GB

2x2688  
CUDA cores  
2x6 GB

2x60  
Xeon Phi cores

2x2880  
CUDA cores  
2x12 GB

2x6-core  
Westmere  
24 GB

2x6-core  
Sandy Bridge  
64 GB

2x6-core  
Sandy Bridge  
64 GB

2x10-core  
Haswell  
64 GB

IB QDR

IB FDR

NUMalink6 / FDR IB

IB QDR

*Thin node cluster*

*SMP*

*Accelerators*



**NX Server**

2x10-core  
Ivy Bridge

64 GB

2 service nodes

2x2304  
CUDA cores  
2x8 GB

2x10-core  
Haswell  
2x64 GB

Home

NetApp  
30 TB

Scratch

DDN1  
92 TB

DDN2  
192 TB

Long  
Term

600 TB

Login nodes

2 visualization  
nodes

**KU LEUVEN**

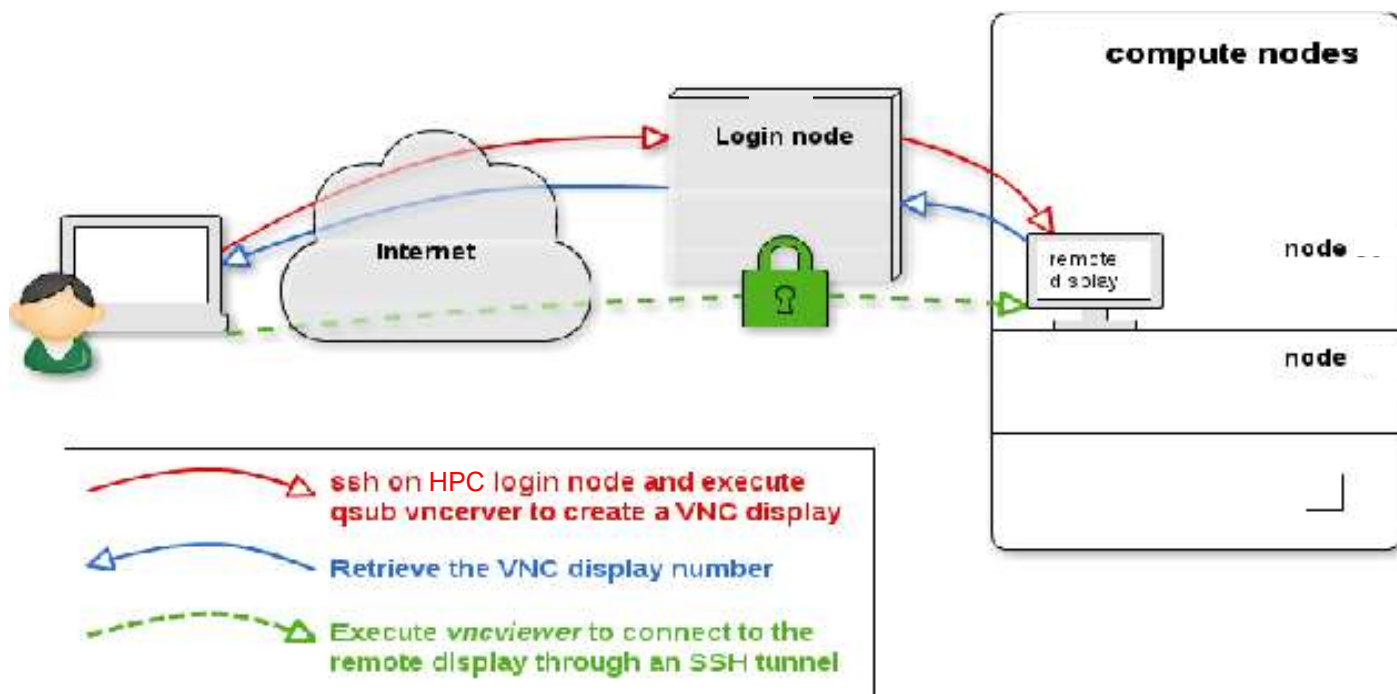


# Remote Desktop Manager: VNC

- A VNC system consists of a client, a server, and a communication protocol
- The VNC server is the program on the machine that shares its screen. The server passively allows the client to take control of it.
- The VNC client (or viewer) is the program that watches, controls, and interacts with the server. The client controls the server.
- The VNC protocol (RFB) is very simple, based on one graphic primitive from server to client ("Put a rectangle of pixel data at the specified X,Y position") and event messages from client to server.
- The server receives from client input events and dispatches them to running applications, it also controls which parts of the screen have been updated by the running apps, grab them from the Framebuffer, compresses them and provides them to the client(s).
- More than one client can connect to the same server sharing the framebuffer and controlling applications (desktop sharing)

# Remote Desktop Manager: VNC

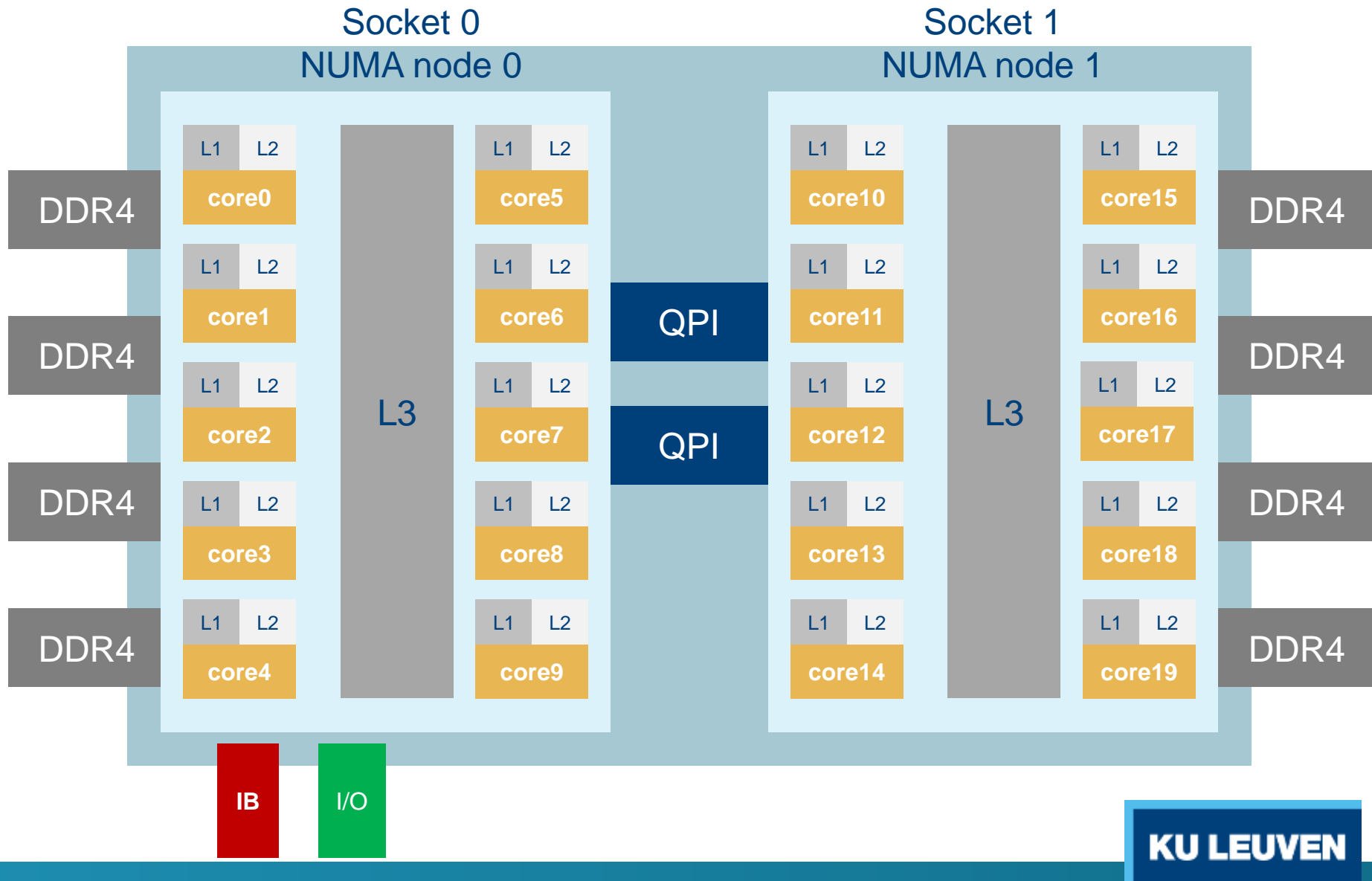
- The setup of VNC connection from user desktop / laptop to the compute node running the VNC server job results to be quite cumbersome (Job submission + ssh tunneling on cluster login node)



# KU Leuven Remote Visualization Service

- 2 visualization nodes (accessible through login node)
  - 2x Haswell (Intel Xeon CPU E5-2650 v3)
  - 2x 64 GB RAM
  - 2x NVIDIA Quadro K5200
  - 250 GB scratch/node
  - Red Hat Enterprise Linux ComputeNode release 6.4 (Santiago), 64 bit, Kernel 2.6.x
  - Direct connection with GPFS file system
- Software
  - ParaView, VisIt,
  - VMD,
  - ImageMagick , VTK
  - VirtualGL

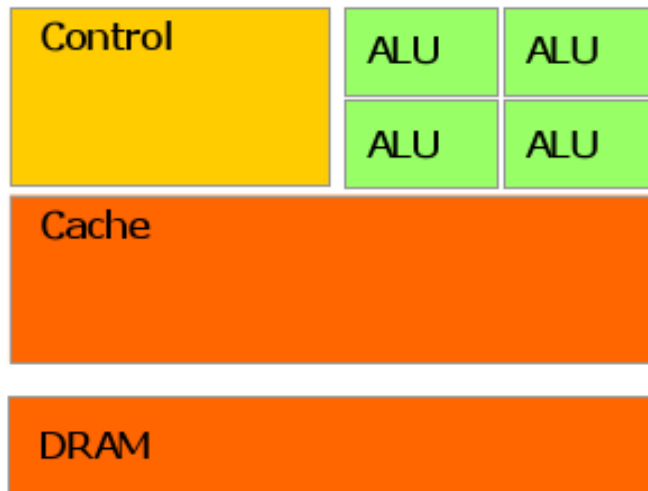
# Hasswell visualization node



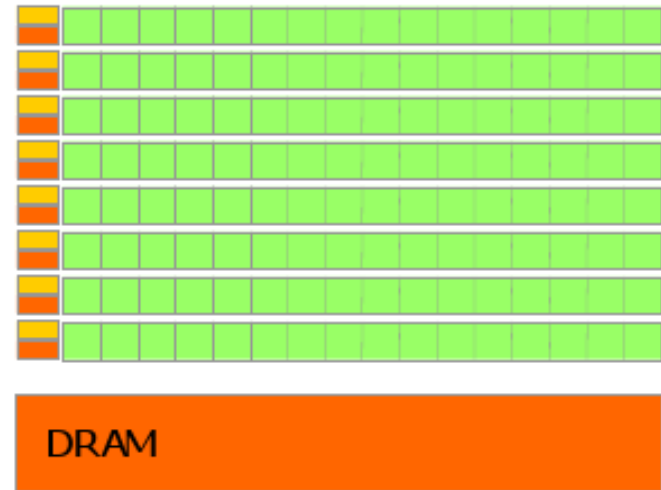
# System comparison

	Ivybridge	Haswell	Haswell visualization
Total nodes	176/32	48	2
Processor type	Ivybridge	Haswell	Haswell (Intel Xeon CPU E5-2650 v3)
Base Clock Speed	2.8 GHz	2.5 GHz	2.3 GHz
Cores per node	20	24	20
Total cores	4160	1152	40
Memory per node (GB)	112x64/32x128	48x64	2x128
Memory per core (GB)	3.2/6.4	2.667	6.4
Peak performance (Flops/cycle)	8 DP FLOPs/cycle: 4-wide AVX addition + 4-wide AVX multiplication	16 DP FLOPs/cycle: two 4-wide FMA (fused multiply-add) instructions	16 DP FLOPs/cycle: two 4-wide FMA (fused multiply-add) instructions
Network	Infiniband QDR 2:1	Infiniband FDR	Infiniband FDR
Cache (L1 KB/L2 KB/L3 MB)	10x(32i+32d)/10x256/25	12x(32i+32d)/12x256/30MB	10x(32i+32d)/10x256/25MB
Total L3 cache per core	2.5 MB	2.5MB	2.5MB

# CPU vs GPU



**CPU**



**GPU**

**GPU Devotes More Transistors to Data Processing** (is designed such that about 80% of transistors are devoted to data processing rather than data caching and flow control - the same function is executed on each element of data with high arithmetic intensity).

# GPU comparison

	K20Xm	K40c	NVIDIA Quadro K5200 (visualization)
Total CUDA cores	2688	2880	2304
SMX	14	15	12
Memory	6GB	12GB	8GB
Base Clock Speed cores	732MHz	745MHz	667MHz
Max clock speed cores	784MHz	874MHz	771MHz
Memory Bandwidth	249,6GB/s	288GB/s	192GB/s
Memory Clock	2,6GHz	3.0GHz	1502MHz
Peak double precision floating point performance	1,31Tflops	1,43Tflops	N/A
Peak single precision floating point performance	3,95Tflops	4,29Tflops	3,074Tflops
Features	SMX, Dynamic Parallelism, Hyper-Q, GPUBoost	SMX, Dynamic Parallelism, Hyper-Q, GPUboost	SMX, Dynamic Parallelism, Hyper-Q, GPUboost

# GPU NVIDIA Quadro K5200 (visualization)

## Advanced Display Features

- 30-bit color (10-bit per each red, green and blue channel)
- Support for any combination of four connected displays
- DisplayPort 1.2 (up to 4096 x 2160 at 60Hz and 2560 x 1600 at 120Hz)
- DP 1.2 Supports 4K Displays and MST Hubs
- DVI-I Dual-Link output (up to 2560 x 1600 at 60Hz and 1920 x 1200 at 120Hz)
- DVI-D Dual-Link output (up to 2560 x 1600 at 60Hz and 1920 x 1200 at 120Hz)
- Internal 400MHz DAC DVI-I output (analog display up to 2048 x 1536 at 85Hz)
- DisplayPort 1.2, HDMI 1.4, and HDCP support (HDMI requires 3rd party adapter)
- 10-bit internal display processing
- NVIDIA 3D Vision technology, 3D DLP, interleaved, and other 3D stereo format support
- Full OpenGL quad buffered stereo support
- Underscan/overscan compensation and hardware scaling
- Support for NVIDIA Quadro Mosaic, NVIDIA NVIEW multi-display technology, NVIDIA Enterprise Management Tools
- Support for large-scale, ultra-high resolution visualization using the NVIDIA SVS platform which includes NVIDIA Mosaic, NVIDIA Sync and NVIDIA Warp/Blend technologies



# GPU NVIDIA Quadro K5200 (visualization)

## Render Config

	NVIDIA Quadro K5200 (visualization)
Shading Units:	2304
TMUs:	192
ROPs:	48
SMX Count:	12
Pixel Rate:	32.0 GPixel/s
Texture Rate:	128 GTexel/s
Floating-point performance:	3,074 GFLOPS
Graphics Features	DirectX, OpenGL, OpenCL, Shader model





# Visualization possibilities @ HPC KU Leuven

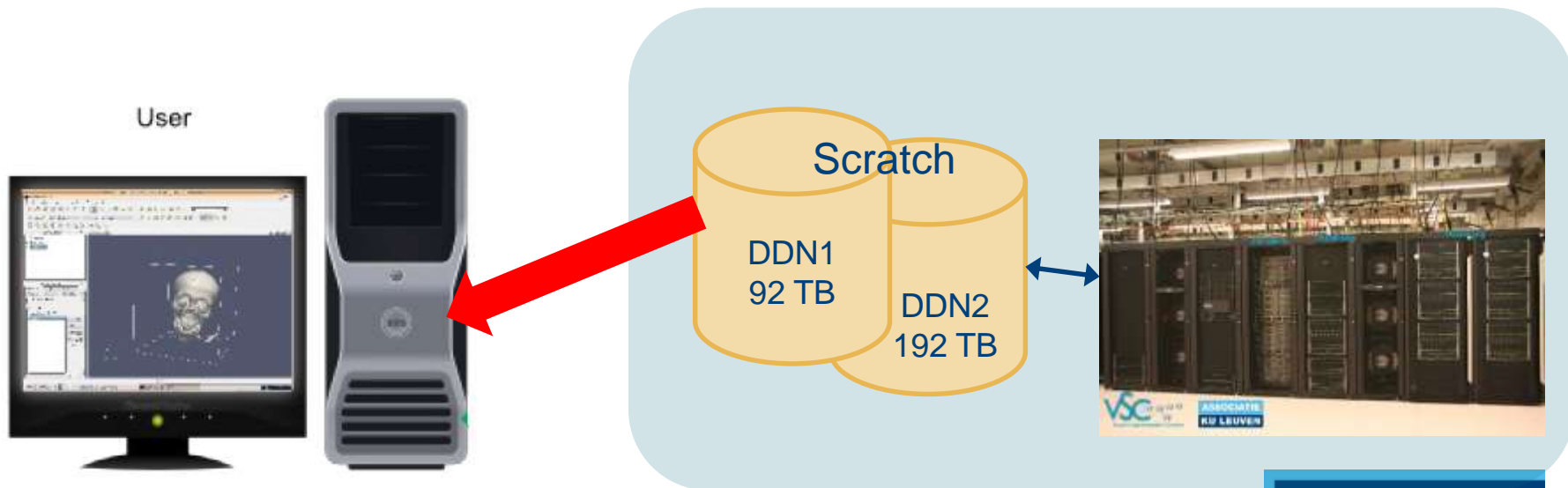
X11, NX, Paraview remote visualization

# Large-scale visualization

- Large-scale parallel computations usually produce large-scale output data
  - Gigabytes to terabytes (or more) data
  - → In general also need parallel processing and rendering in order to visualize
  - → Fast access to data storage necessary
- **Approach A:** visualize on HPC system where data was produced
  - Problem #1 - HPC systems usually batch-oriented
    - Creating visualizations has important interactive component
    - Queue: need to wait for job to start, unpredictable when it runs
  - Problem #2 - GPUs often not available in HPC systems
    - Software-only 2D/3D rendering possible, but on the order of 5x - 50x slower than using a GPU

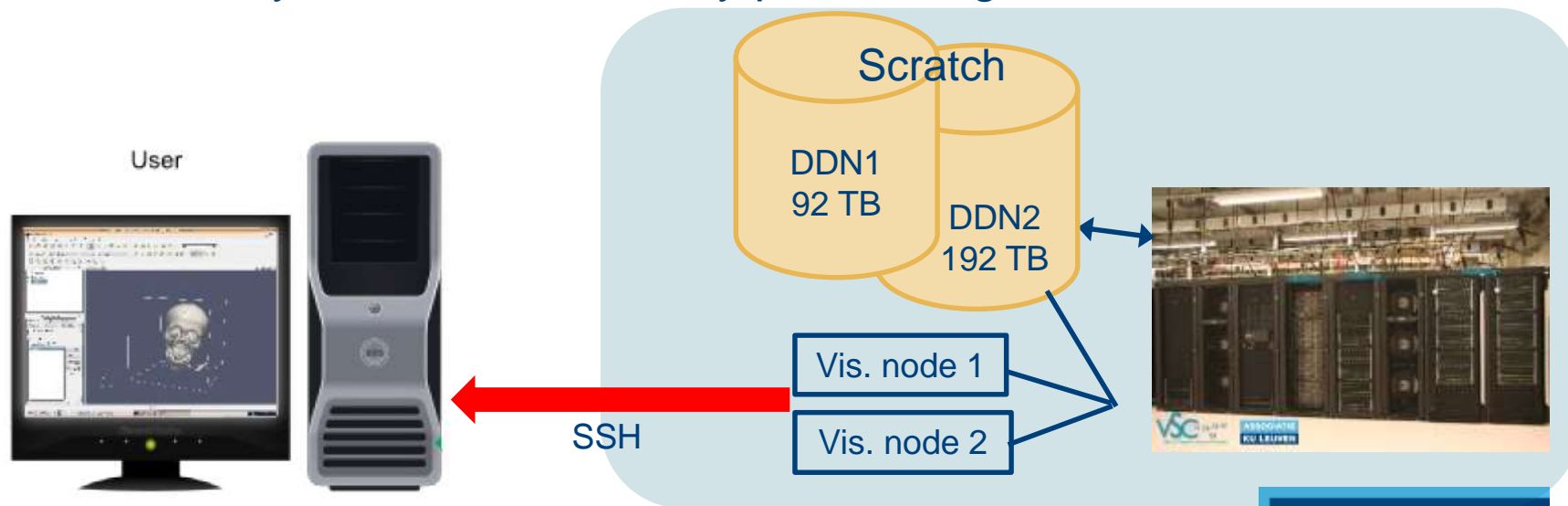
# Large-scale visualization (2)

- **Approach B:** visualize locally
  - I.e. on user's own PC
  - Problem #1 - Need to transfer data and store it locally
    - Network bandwidth? Storage?
  - Problem #2 - Enough CPU/GPU resources available?



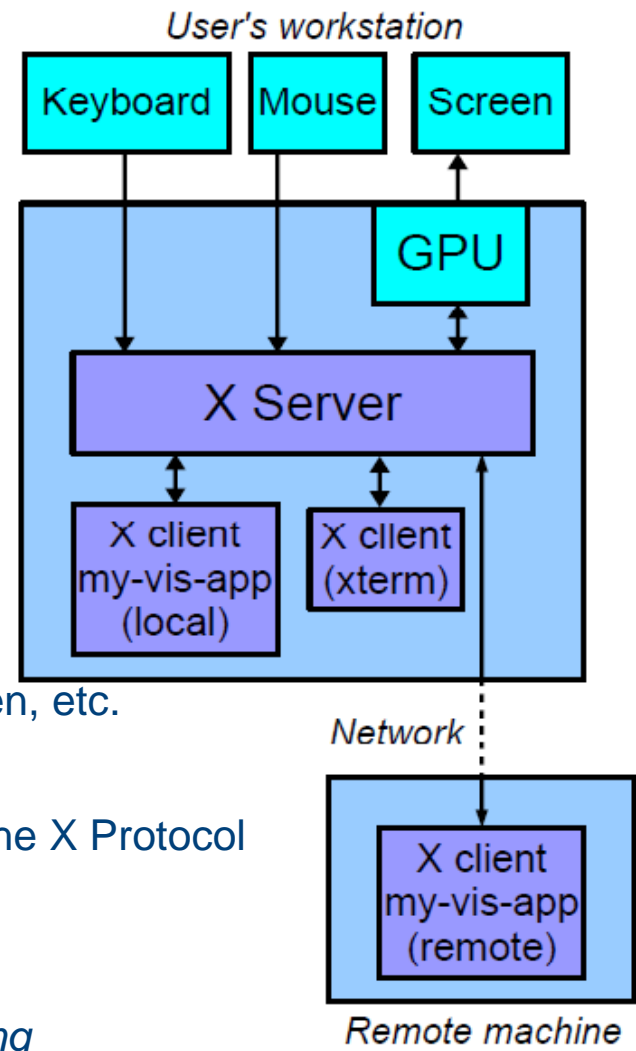
# Large-scale visualization (3)

- **Approach C:** remote visualization
  - Provide dedicated visualization resources in data center
  - Visualization resource has direct fast connection to central storage
  - Avoid large data transfers over external network by running visualization application remotely
  - Data stays in data center, only pixels/images sent to external user



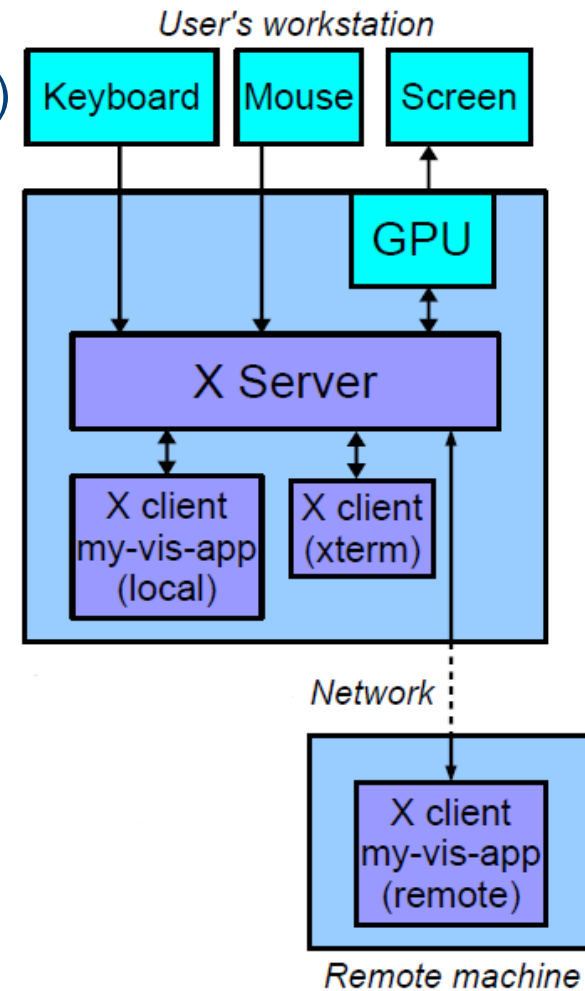
# Remote visualization – X11

- X Windows (X11)
  - Used in all modern Linux desktop installations
    - X.org server
  - Client-server model
- X Server
  - Client applications connect to server
  - Receives input from user (keyboard, mouse, tablet, etc)
  - Forwards input to correct application
  - Applications tell X server what to draw on the screen, etc.
  - X server manages screen contents
  - Communication between clients and server using the X Protocol
- Powerful feature:
  - Run X11 applications remotely
  - I.e. X Protocol over network connection *X forwarding*



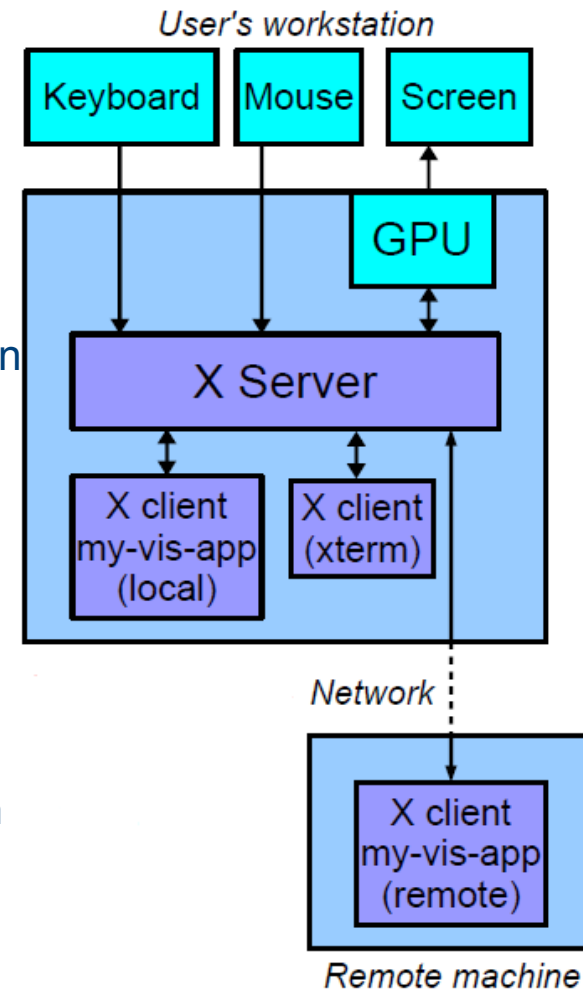
# X11 & OpenGL: GLX

- GLX (OpenGL eXtension for the X Windows System) provides the interface connecting OpenGL and the X Window System: it enables programs wishing to use OpenGL to do so within a window provided by the X Window System.
- GLX consists of three parts:
  - An API that provides OpenGL functions to an X Window System application.
  - An extension of the X protocol, which allows the client (the OpenGL application) to send 3D rendering commands to the X server (the software responsible for the display). The client and server software may run on different computers.
  - An extension of the X server that receives the rendering commands from the client and passes them on to the installed OpenGL library



# X11 & OpenGL: GLX

- Application that wants to draw with OpenGL:
  - App uses GLX to get OpenGL context
  - App draws 2D/3D primitives with OpenGL commands
  - X Server takes care that commands get to OpenGL library
  - App → GLX → X Server → OpenGL library → GPU → Screen
- Direct Rendering Interface (DRI):
  - App → OpenGL library → GPU → Screen
  - Only for applications running on the same system as the X server (app needs direct access to GPU)
- Remote applications can only use GLX over network (“indirect rendering”)
  - Lots of 2D/3D geometry or textures might be transferred from server to client with GLX
  - Possibly new data each frame
    - e.g. animated isosurface, 2D textured slice being moved through a volume, etc.





# General remote X11 disadvantages

- X11 protocol is quite verbose. On high-latency and/or low-bandwidth connections it doesn't really work nicely
  - e.g. home ↔ office connection
  - on gigabit LANs usually quite usable
- Need X11 server on client-side
- GLX Problems:
  - Subset of OpenGL features available when using indirect rendering
  - Certain OpenGL extensions need direct GPU access

# Using X11 for GUI interaction

- Run an X Window display server (X server) on your local computer. You start up such a program and leave it running in the background.
- Connect to the cluster through your normal ssh terminal program, with X11 forwarding enabled. This establishes the X11 connection between the VSC cluster and your local computer.
- Start the GUI application (e.g. gnuplot) on the cluster. The graphical output of the program will display on your desktop.
- Interact with the GUI application using your mouse and keyboard. Your keyboard and mouse commands will be relayed in the other direction, allowing to interact with the running application on the VSC system.
- **Problems:**
  - slow; not recommended for complex graphics
  - X server running locally -> enough resources???

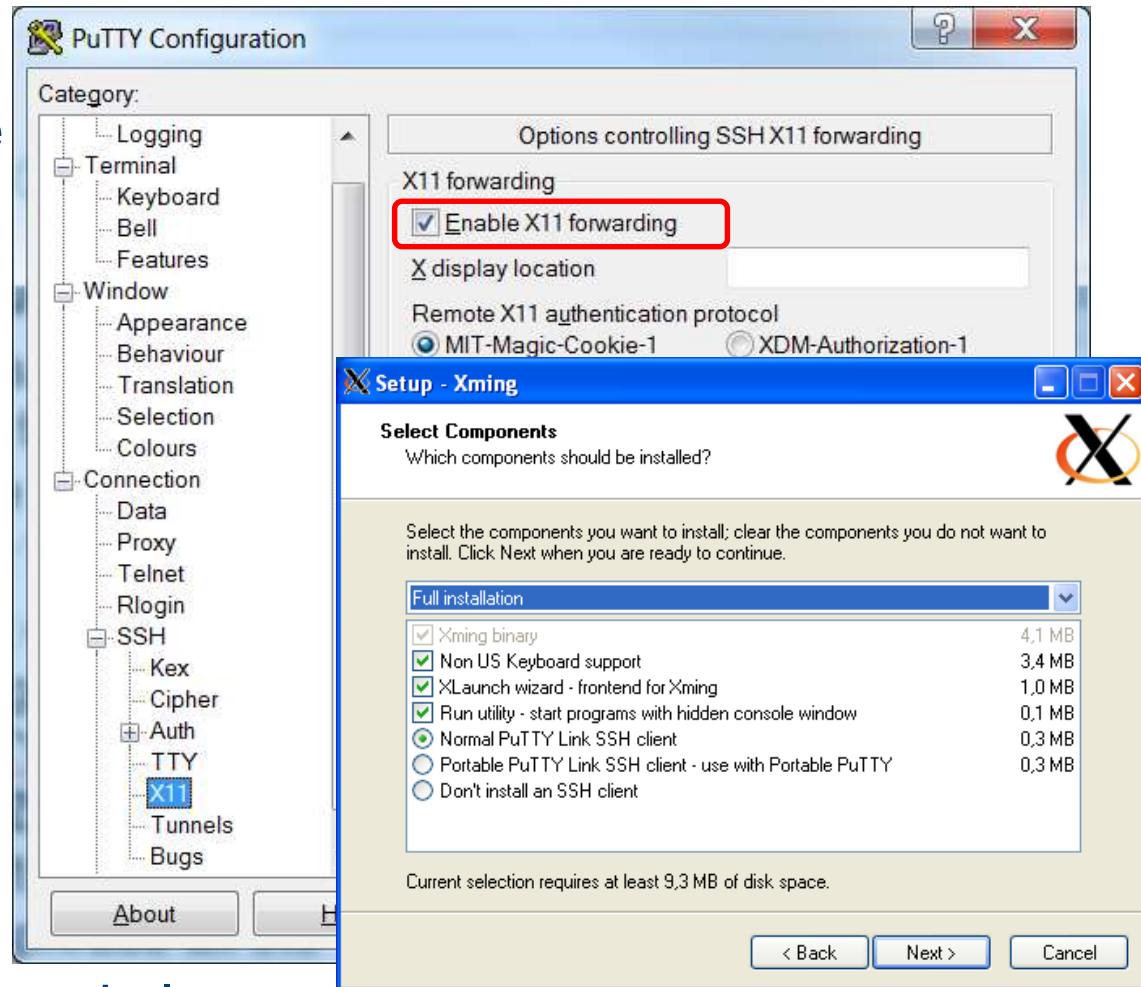
# Using X11 for GUI interaction

## Windows users:

- PuTTY is a simple-to-use and freely available GUI SSH client for Windows.
- Pageant can be used to manage active keys for PuTTY
- Xming: using X-windows to display graphical programs

## Linux users:

- `ssh -X vsc30000@login.hpc.kuleuven.be`



# Using X11 for GUI interaction

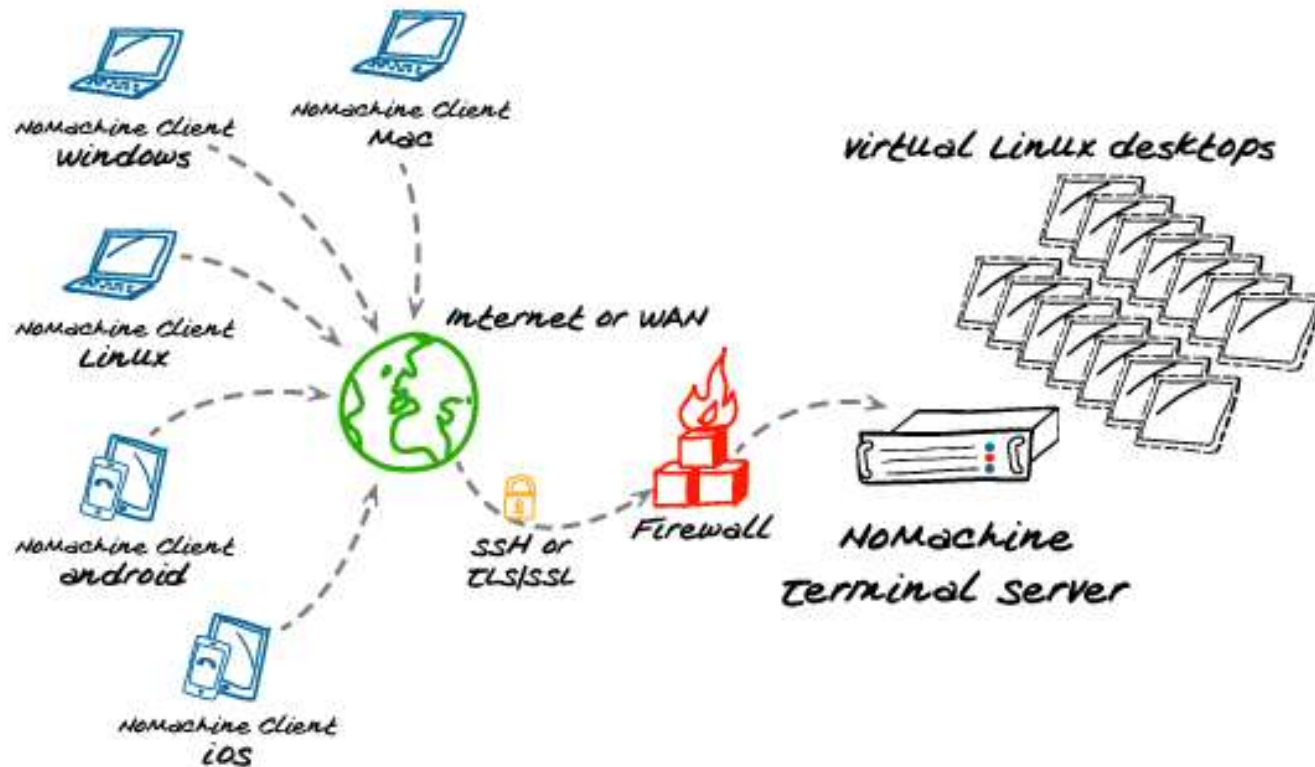
- Too slow?-> NX
  - NX accelerates remote X11 applications by using various strategies to compress the X protocol, reducing the amount of round-trip network interactions
- Problems:
  - NX cannot take advantage of the 3D acceleration provided by modern video cards, i.e. the 3D applications either do not work at all, are forced to use a slow software 3D renderer, or (worse) are forced to send every 3D command and piece of 3D data over the network to be rendered on the client machine.

# What is NX?

## NX technology:

- Handles remote X Window System connections, and attempts to greatly improve on the performance of the native X display protocol to the point that it can be usable over a slow link such as a dial-up modem.
- Wraps remote connections in SSH sessions for encryption.
- NX software is currently available for Windows, Mac OS X, Linux, and Solaris.
- The two principal components of NX are **nxproxy** (both on remote and local - server machines simulating an X server on the client) and **nxagent** (started on the remote - client machine, thus avoiding most X11 protocol round trips).

# What is NX?



NX works by creating an nx-user on the server machine whose shell is executed any time a remote NX user connects to SSH using NX Client.

# What is NX?

NX allows the user:

- To log in remotely (over a slow internet) to a server,
- Login is graphical (GUI serves as development environment),
- NX also allows to suspend and resume sessions,
- During suspension, the processes invoked inside the session continue to run,
- Therefore NX can be used as a graphical alternative to **SSH+screen** application.

# Main advantages of NX

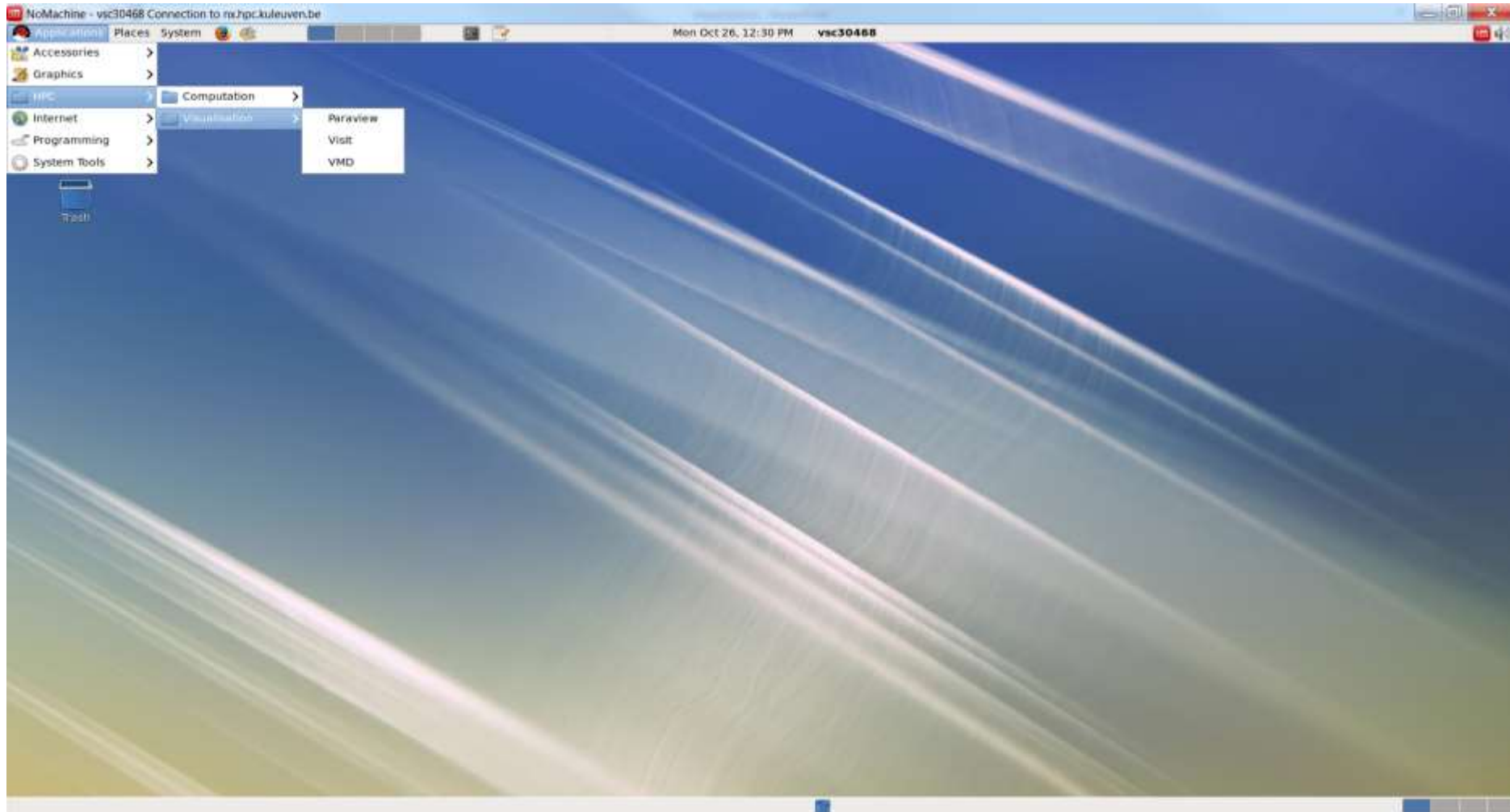
- Keeps session open,
- Alternative for people using screen,
- More interactive jobs,
- Easy in use for editing, file management, developing software,
- Different limits of CPU (regular login node 36 min, NX node extended to 2 hrs)
- **It is a shared resource!**



# Who should use it?

- HPC users that prefer GUI,
- HPC users that have no/little experience with command line,
- HPC Linux users that are used to certain settings and shortcuts,
- HPC researchers that would like to display results easily,
- HPC users that are in a process of interactive job and do not want to loose the session when taking laptop home,
- HPC users that often use screen,
- HPC users that need to work from mobile devices.

# NX virtual desktop



# NX: available software

- **Accessories:** Gedit, Vi IMproved, Emacs (dummy version), Calculator,
- **Graphics:** gThumb (picture viewer), Xpdf Viewer,
- **Internet:** Firefox,
- **HPC: Computation:** Matlab (2014a), RStudio, SAS;  
**Visualisation:** Paraview, VisIt, VMD
- **Programming:** Meld Diff Viewer (visual diff and merge tool),
- **System tools:** File Browser, Terminal,
- **Additionally:** Gnuplot (graphing utility), Filezilla (file transfer tool), Evince (PDF, PostScript, TIFF, XPS, DVI Viewer),
- Software launched through modules from Terminal.

# NX: How to get started

- <https://www.vscentrum.be/client/multiplatform/nx-start-guide>
- **Configuration guide:**  
<https://www.vscentrum.be/assets/195>  
**needs conversion of key (Windows)**

# Paraview remote visualization

- Paraview is a powerful, general purpose, open source visualization tool.
- It is possible to connect a ParaView client, running on your local desktop, to a ParaView server running on a compute node (CPU or GPU).
- The server streams the geometry data to the client and the client handles all of the other interactions locally. This is quite efficient on slower network connections.
- Using ParaView in this manner is similar to using ParaView in a remote visualization form but in this mode of operation Paraview (rather than VNC) takes care of communicating the user interaction and visualization data between the client (on your local computer) and the server (running on a compute node).

# Paraview remote visualization

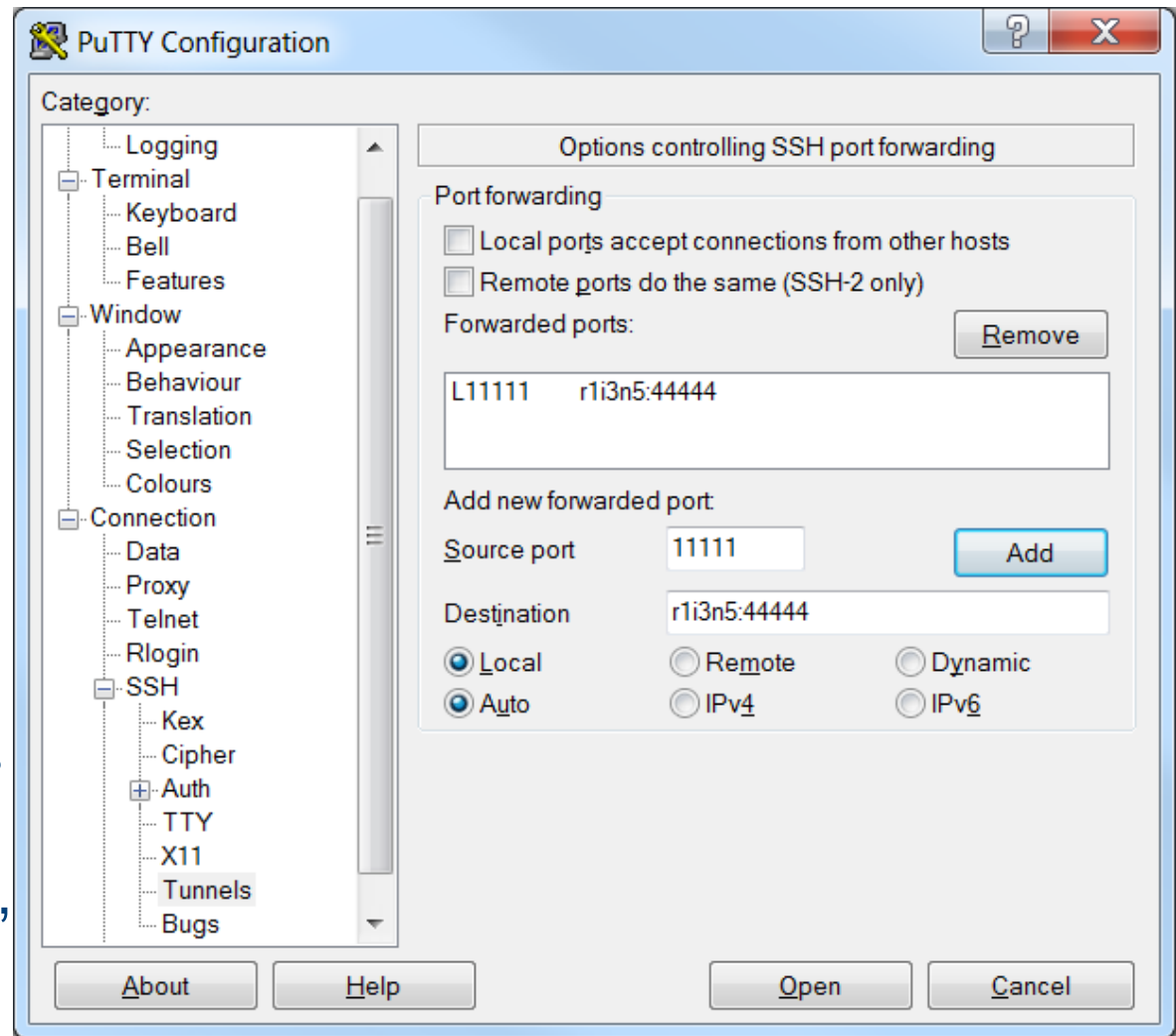
- You should have ParaView installed on your desktop. The client and server version should match to avoid problems!
- Working with ParaView to remotely visualize data requires several steps:

## 1. start ParaView on the cluster

- `$ qsub -I -l nodes=1:ppn=20`
- `$ module load Paraview/4.1.0-foss-2014a`
- `$ n_proc=$(cat $PBS_NODEFILE | wc -l)`
- `$ mpirun -np $n_proc pvserver --use-offscreen-rendering --server-port=11111`
- In this mode, all data processing and rendering are handled in the same parallel job

# Paraview remote visualization

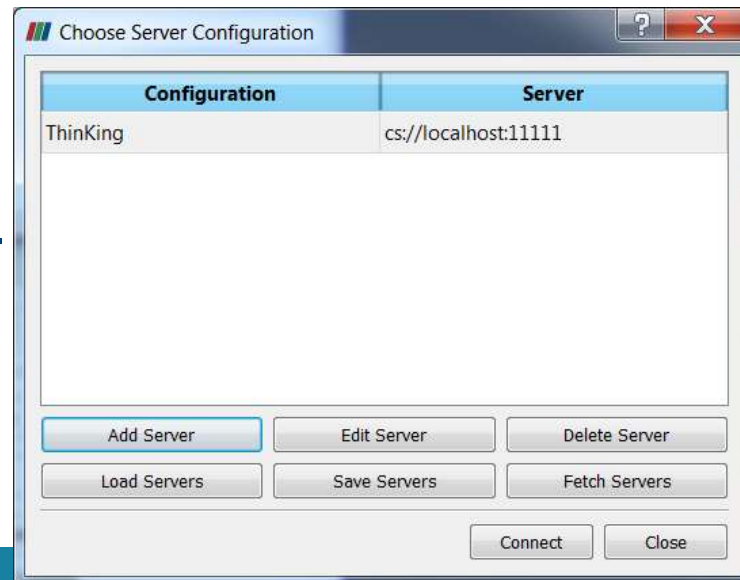
2. establish an SSH tunnel;
  - Log in on the login node
  - Start the server job, note the compute node's name the job is running on (e.g., 'r1i3n5'), as well as the port the server is listening on (e.g., '44444').



```
ssh -L11111:r1i3n5:44444 -N vsc30000@login.hpc.kuleuven.be
```

# Paraview remote visualization

3. connect to the remote server using ParaView on your desktop:
  - start ParaView on your Desktop machine
  - from the 'File' menu, choose 'Connect,
  - click the 'Add Server' button
  - enter a name in the 'Name' field, e.g., 'Thinking'.
  - set the 'Startup Type' from 'Command' to 'Manual' in the drop-down menu, and click 'Save'.
  - in the 'Choose Server' dialog, select the server, i.e., 'Thinking' and click the 'Connect' button.
  - you can now work with ParaView as you would when visualizing local files.
4. terminate the server session on the compute node
  - `$ logout`

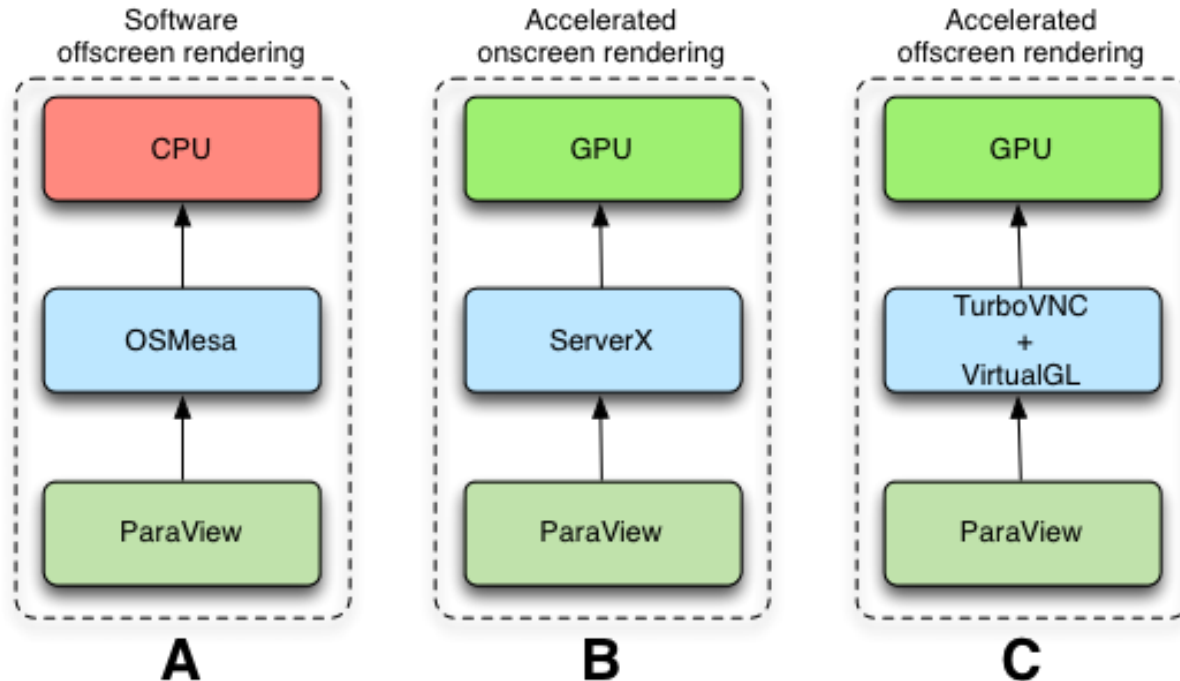




# Visualization node – why the best choice



# Remote visualization



- VirtualGL
  - Popular package for remote visualization
- TurboVNC
  - Good way to provide access to remote visualization applications

# Remote visualization

- Meant for interactive visualization applications
  - ParaView already remote (client-server)
  - What about other applications?
- Remote rendering using Linux/Unix application and visualization nodes
- 2D/3D rendering with OpenGL
  - Low-level API for drawing 2D and 3D content
  - All self-respecting graphics cards support it
  - De-facto standard for rendering on Linux/Unix

# Remote visualization - Concept

- Dedicated visualization resource
  - Usually cluster of render nodes (one or more GPUs per node)
  - (Fast) interconnect between nodes
  - Fast connection to central storage
  - Reasonable connection to outside world
  - User connects to resource and only receives visualization output
- User advantage: facility maintained by HPC center
- Open questions
  - How to work with applications remotely?
  - License issues
  - What software is supported/what isn't?

# Better remote rendering idea?

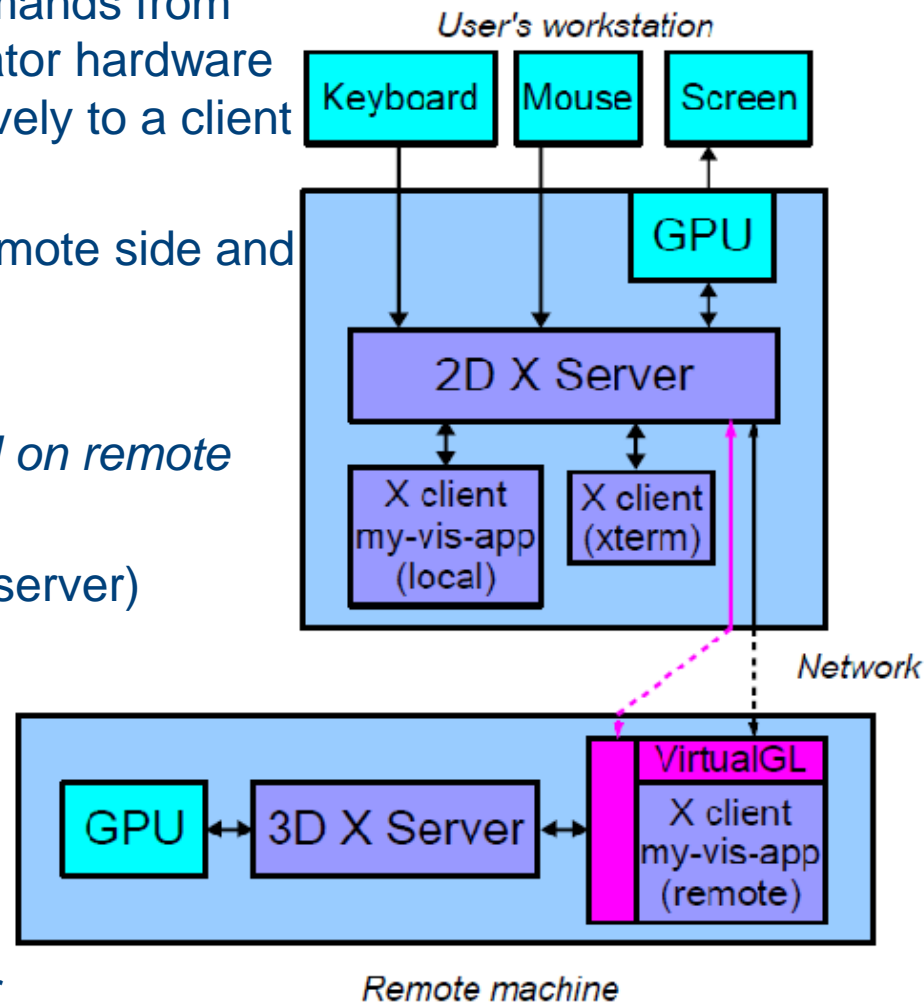
- We want to run applications remotely, as if they're running on our desktop
- But want to use a server-side GPU
- How? This is where VirtualGL comes in...

# VirtualGL

- History
  - Started by Sun as part of the Sun Visualization System
  - VirtualGL open-sourced when Sun stopped supporting SVS
- Open-source
- Supported platforms
  - Server: Unix (Linux, Solaris)
  - Client: Unix, Windows, MacOS
- Note: server only available on Unix-like systems, but in general those are the systems used for servers anyway

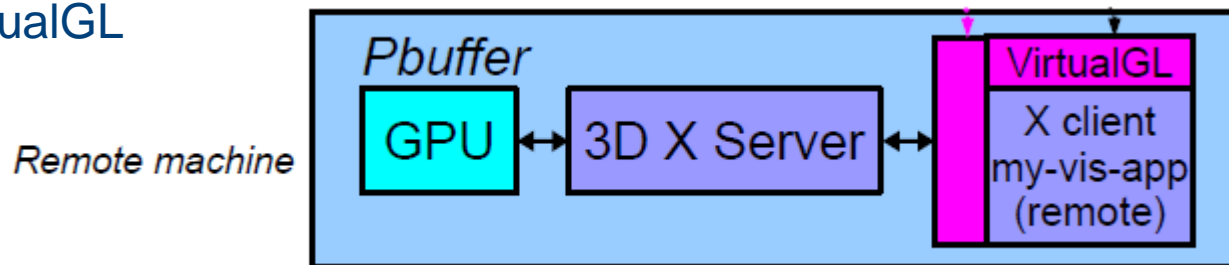
# VirtualGL – Split Rendering

- VirtualGL redirects the 3D rendering commands from Linux OpenGL applications to 3D accelerator hardware and displays the rendered output interactively to a client located elsewhere on the network.
- 3D rendering commands intercepted at remote side and forwarded to remote GPU
  - 3D OpenGL rendering commands
  - *Need X server (3D X server) and GPU on remote side*
- 2D commands go to local X server (2D X server) untouched
  - Menus, buttons, etc.
  - Basically unaltered remote X11
- Resulting image from 3D rendering read back
  - Extra image stream needed to transfer 3D image result to user's side



# VirtualGL – Behind the scenes

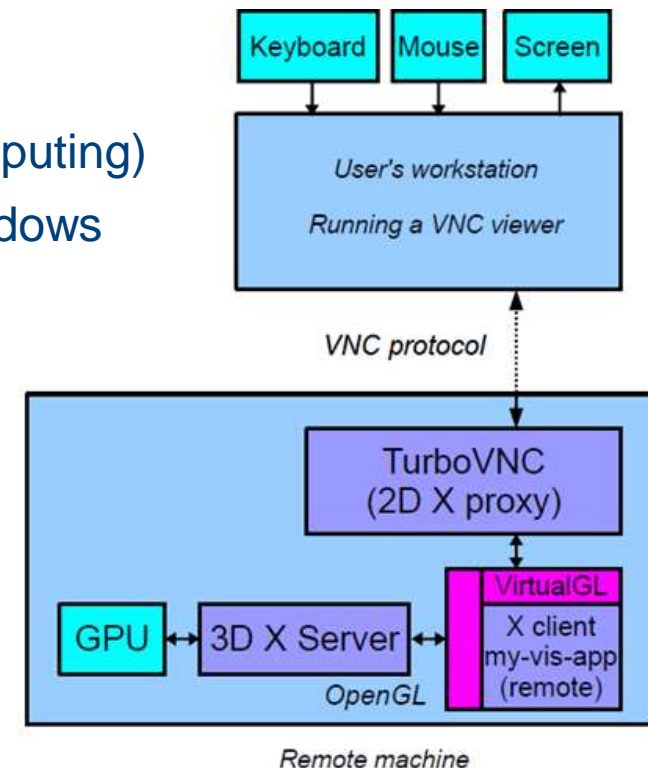
- VirtualGL preloads a shared library that intercepts some GLX, X11 and OpenGL calls
  - New window created by application → VirtualGL creates corresponding offscreen rendering buffer (PBuffer) on remote GPU
    - Window resize → offscreen buffer adjusted
  - OpenGL draw commands executed → Drawing is done on Pbuffer
  - Application finishes rendering frame → Pbuffer contents read back and 3D image sent to client
  - All other calls left untouched
- All OpenGL features available, because Pbuffer on local GPU is used
  - No missing extensions, etc.
  - If application runs locally on a server, it will run remotely on that server under VirtualGL





# Image delivery method: VNC (X Proxy)

- X Proxy (Virtual X server)
  - Acts like an X server, but provides only minimal functionality
    - E.g. no keyboard/mouse input
    - No access to GPU
- Most useful proxy is VNC server (Virtual Network Computing)
  - Similar to Remote Desktop Protocol (RDP) on Windows
  - VNC is a widely used remote desktop technology
    - User connects VNC viewer to VNC server
    - Server sends back updated desktop image, shown in VNC client
    - User's keyboard and mouse input are transferred to application on VNC server
  - Efficient transfer of changes in the desktop image
    - Desktop image divided in tiles
    - Only tiles that have changed since last frame sent to client
    - Different compression method per tile possible



# TurboVNC

- Preferred X proxy for VirtualGL
  - Good integration with VirtualGL
  - VirtualGL and TurboVNC maintained by same software developers
  - TurboVNC was specifically created to handle interactive 3D and video workloads
    - Uses optimized JPEG compression routines
- Stable, good documentation
- Supported platforms
  - Server: Unix (Linux, Solaris)
  - Client: Unix, Windows, MacOS
- See <http://www.virtualgl.org>

# TurboVNC - Usage

## Server node

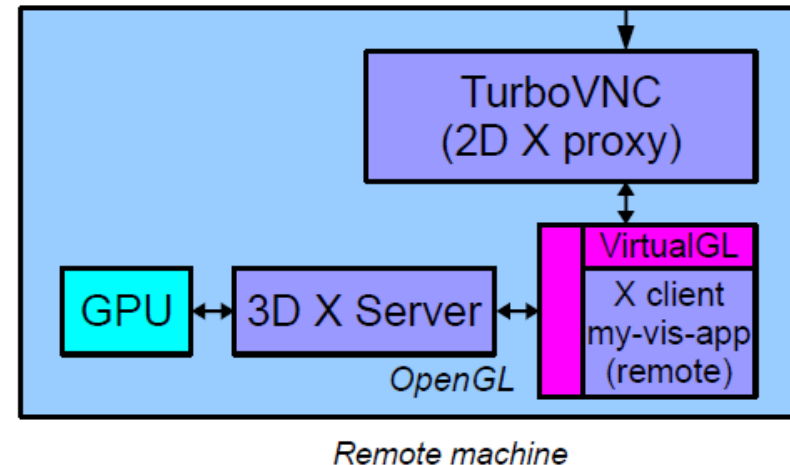
- Set password (one-time action)
  - `$ vncpasswd`
- Start VNC server
  - `$ vncserver [-geometry <width>x<height>]`
- Stopping
  - `$ vncserver -kill :1`
- Listing active VNC servers
  - `$ vncserver -list`
- Server log files in `$VSC_HOME/.vnc`
  - Contains info on connections, compression settings, etc.
  - Useful for debugging connection problems
- Can put extra environment options in startup script  
`$VSC_HOME/.vnc/xstartup.turbovnc`

## Client node

- Connect with VNC viewer from client machine
  - `$ vncviewer <server>:1`

# TurboVNC + VirtualGL

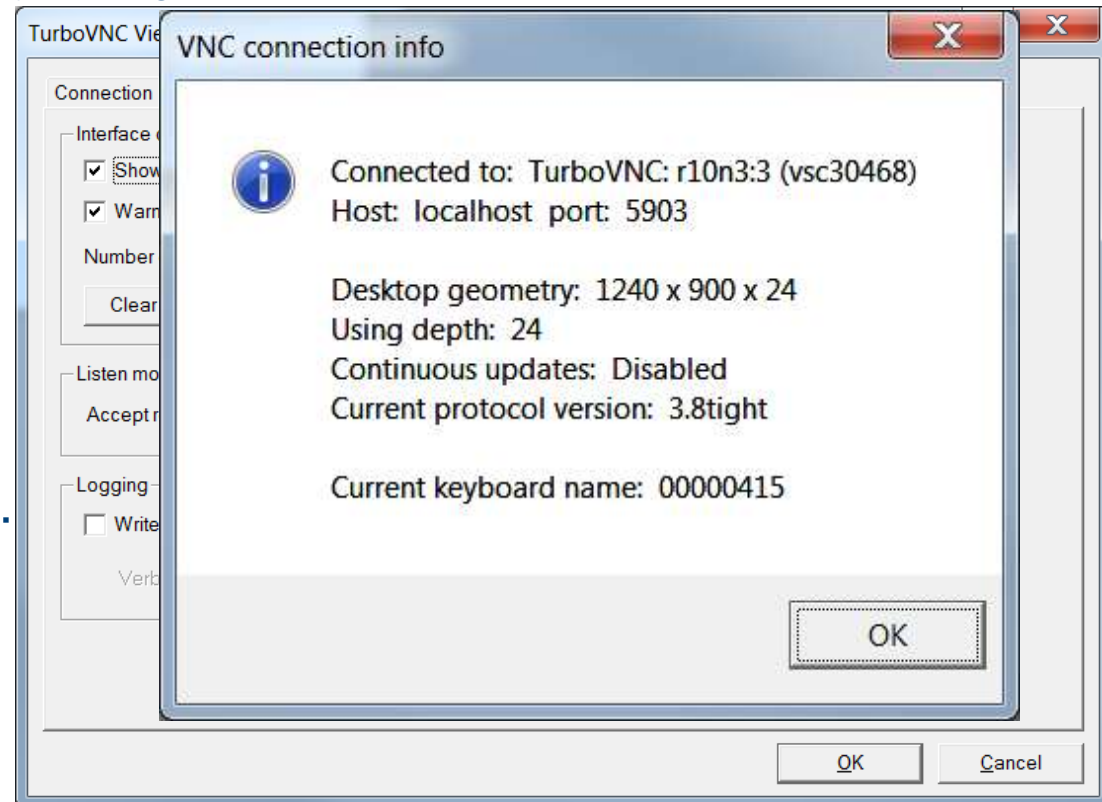
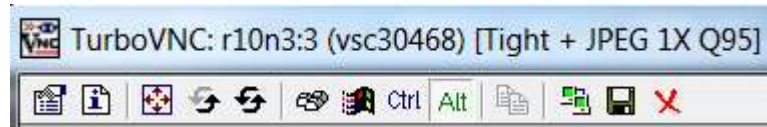
- Visualization applications running inside a VNC session need access to GPU
  - The VNC server itself has no access to GPU, as it is only a virtual X server, not a real one
  - VirtualGL takes care of 3D interception
- How to use:
  - Login in on server machine,
  - start TurboVNC server
  - Connect with VNC client to server
  - Use vglrun command when starting application in VNC session, e.g. `$ vglrun -d :0 paraview`



# TurboVNC - Options

In viewer:

- Options
  - Set compression and JPEG quality settings
  - Mouse/cursor settings
  - Logfile settings
- Connection info
- Fullscreen on/off
- Request screenrefresh
- Request lossless refresh
- Send Ctrl+Alt+Del, Ctrl+Esc...
- New connection
- Save connection
- Disconnect



# (Turbo)VNC - Security?

- Basically none provided in the RFB protocol!
  - Although passwords aren't sent in cleartext when authenticating...
  - *...keystrokes and desktop images in a VNC session are unprotected!*
  - E.g. passwords you enter *within* a VNC session to login to other machine are going over the network in the clear
- How paranoid should you be about this?
- We use SSH tunneling
  - Will cost a bit of performance, but still very usable
  - `$ ssh -L 590<d>:localhost:590<d> <remote-server>`
  - `$ vncviewer localhost:<d>`
  - On Linux with TurboVNC:
    - `$ vncviewer -via <remote-server> localhost:<d>`

# (Turbo)VNC - Summary

- Advantages

- Better interactivity compared to remote X11, efficient network usage
  - High-speed JPEG image compression
  - Only updated parts of desktop image get transferred
- If connection from client to server is lost, remote desktop will stay alive as long as VNC server keeps running. Simply reconnect with client.

- Disadvantages

- Need to open up extra TCP port(s) on the remote machine for incoming VNC client connections
  - VNC :<d> will listen to TCP port 590<d>
  - Use SSH tunneling
- Need to use SSH tunneling when security matters
- VNC client window shows remote desktop in a window on your normal desktop, can be confusing with keys, focus, mouse, etc

# Settings





# How to connect

- `$ qsub -I -X -l partition=visualization`
- `vsc30468@r10n3 ~ $ module load TurboVNC/1.2.3-foss-2014a`
- `$ vncpasswd`
  - Password:
  - Verify:
  - Would you like to enter a view-only password (y/n)? n
  - Cannot write password file  
/user/leuven/304/vsc30468/.vnc/passwd
- `$ mkdir .vnc`
- `$ touch .vnc/passwd`
- `$ vncpasswd`
  - Password:
  - Verify:
  - Would you like to enter a view-only password (y/n)? n

# How to connect

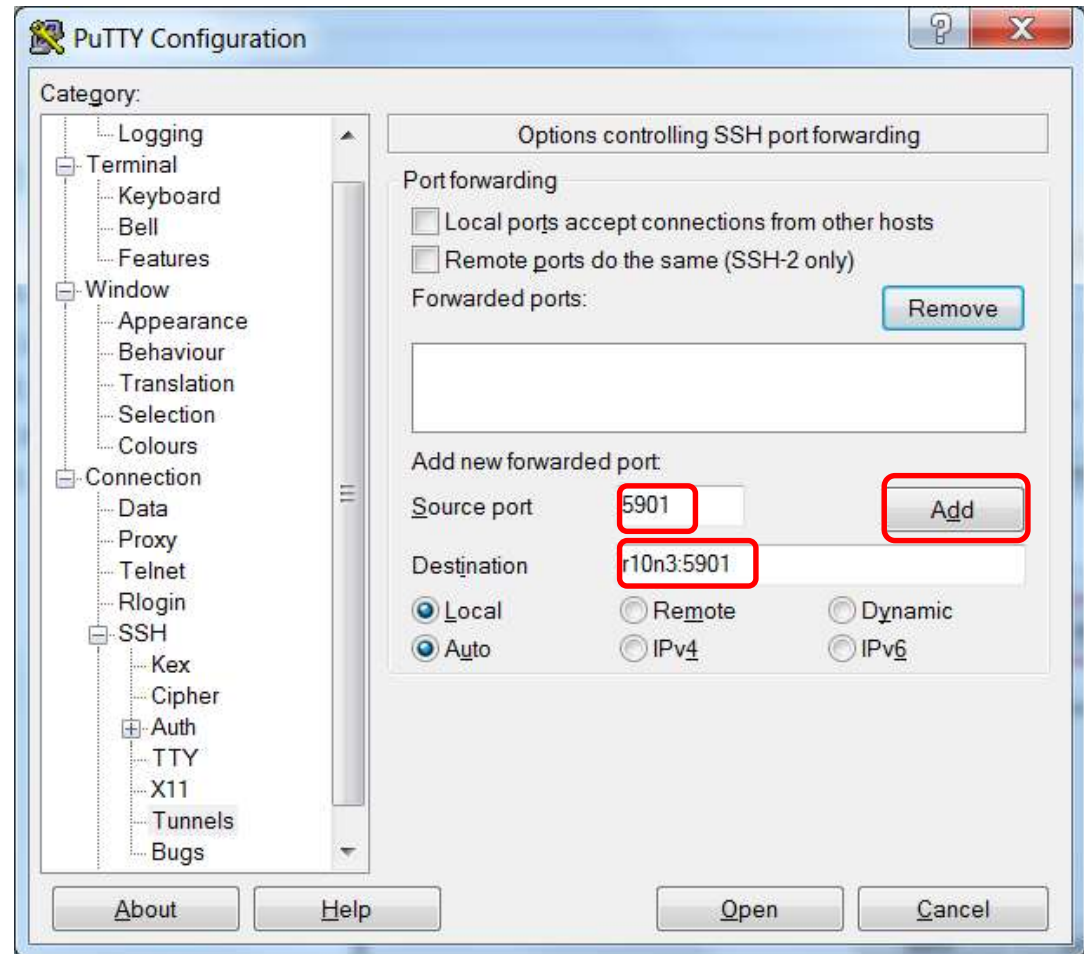
- ```
$ vncserver (-depth 32 -geometry 1600x1000)  
Desktop 'TurboVNC: r10n3:1 (vsc30468)' started  
on display r10n3:1  
Creating default startup script  
/user/leuven/304/vsc30468/.vnc/xstartup.turbovnc  
Starting applications specified in  
/user/leuven/304/vsc30468/.vnc/xstartup.turbovnc  
Log file is  
/user/leuven/304/vsc30468/.vnc/r10n3:1.log
```

host

display

# How to connect

- `$ vncserver`
  - Desktop 'TurboVNC:  
`r10n3:1` (vsc30468) '  
started on `display`  
`r10n3:1`
  - Source port=590<d>,  
where <d> is display  
number
  - Destination=  
host:5901<d>

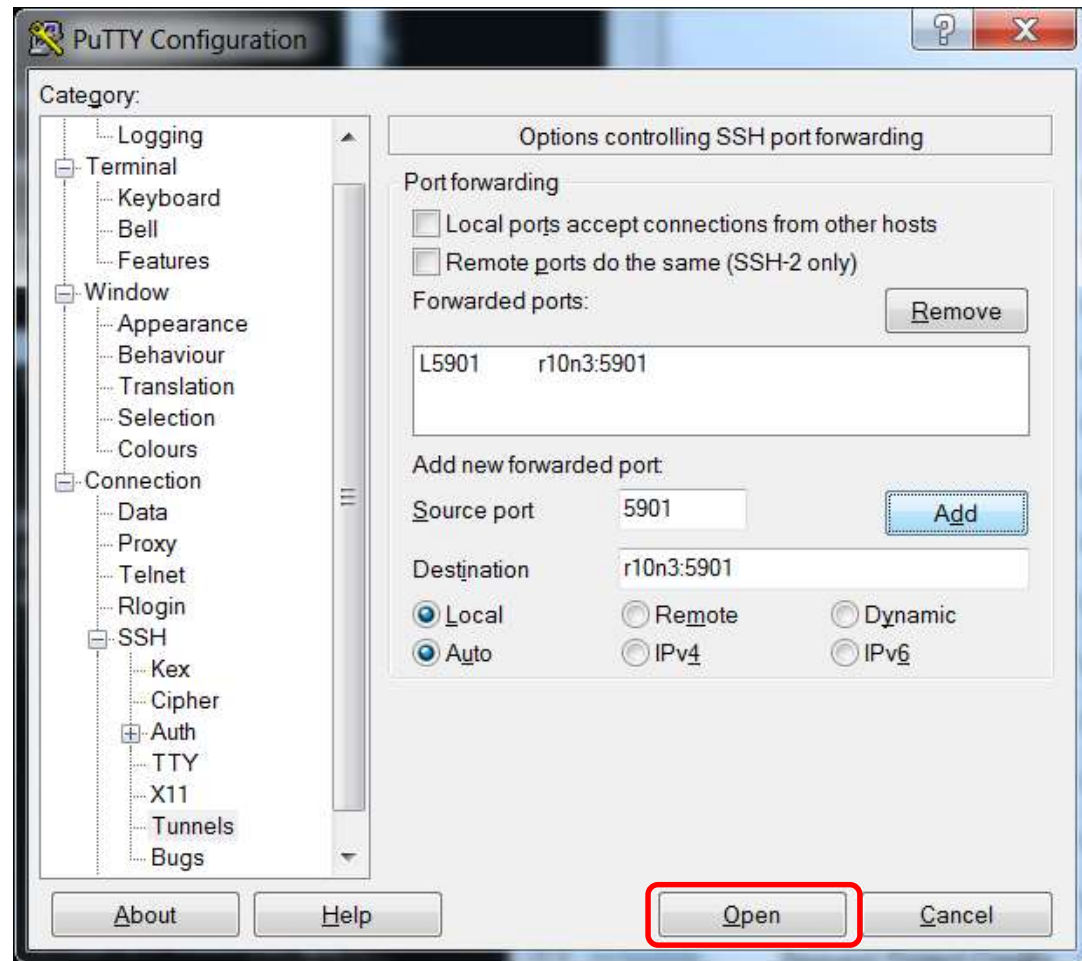


## In Linux:

- `$ ssh -L 590<d>:host:590<d> -N vsc30000@login.hpc.kuleuven.be`
- e.g. `$ ssh -L 5901:r10n3:5901 -N vsc30000@login.hpc.kuleuven.be`

# How to connect

- `$ vncserver`
  - Desktop 'TurboVNC:  
r10n3:1 (vsc30468) '  
started on display  
r10n3:1
  - Source port=590<d>,  
where <d> is display  
number
  - Destination=  
host:5901<d>

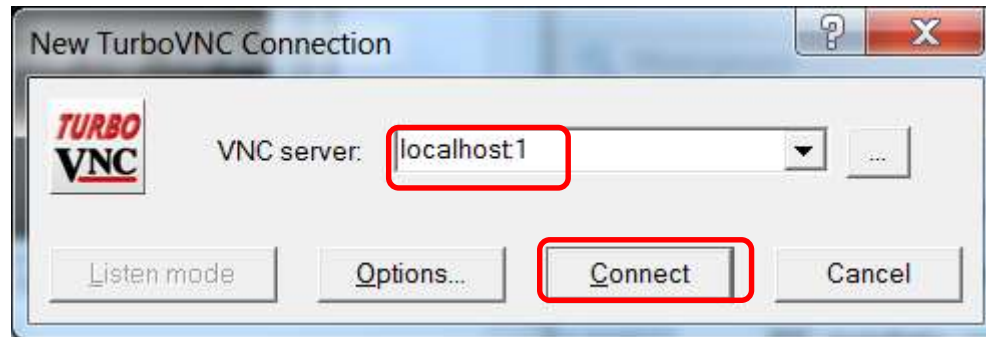


In Linux:

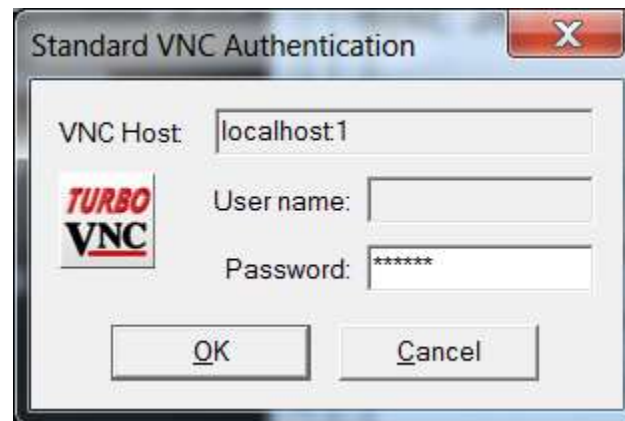
- `$ ssh -L 590<d>:host:590<d> -N vsc30000@login.hpc.kuleuven.be`
- e.g. `$ ssh -L 5901:r10n3:5901 -N vsc30000@login.hpc.kuleuven.be`

# How to connect

- Start the client: VSC server as `localhost:<d>` (where `<d>` is display number)



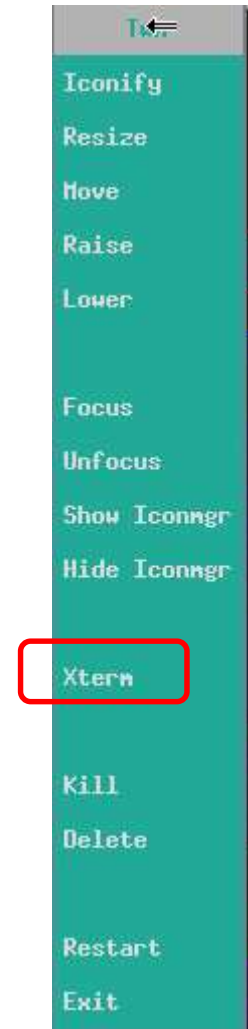
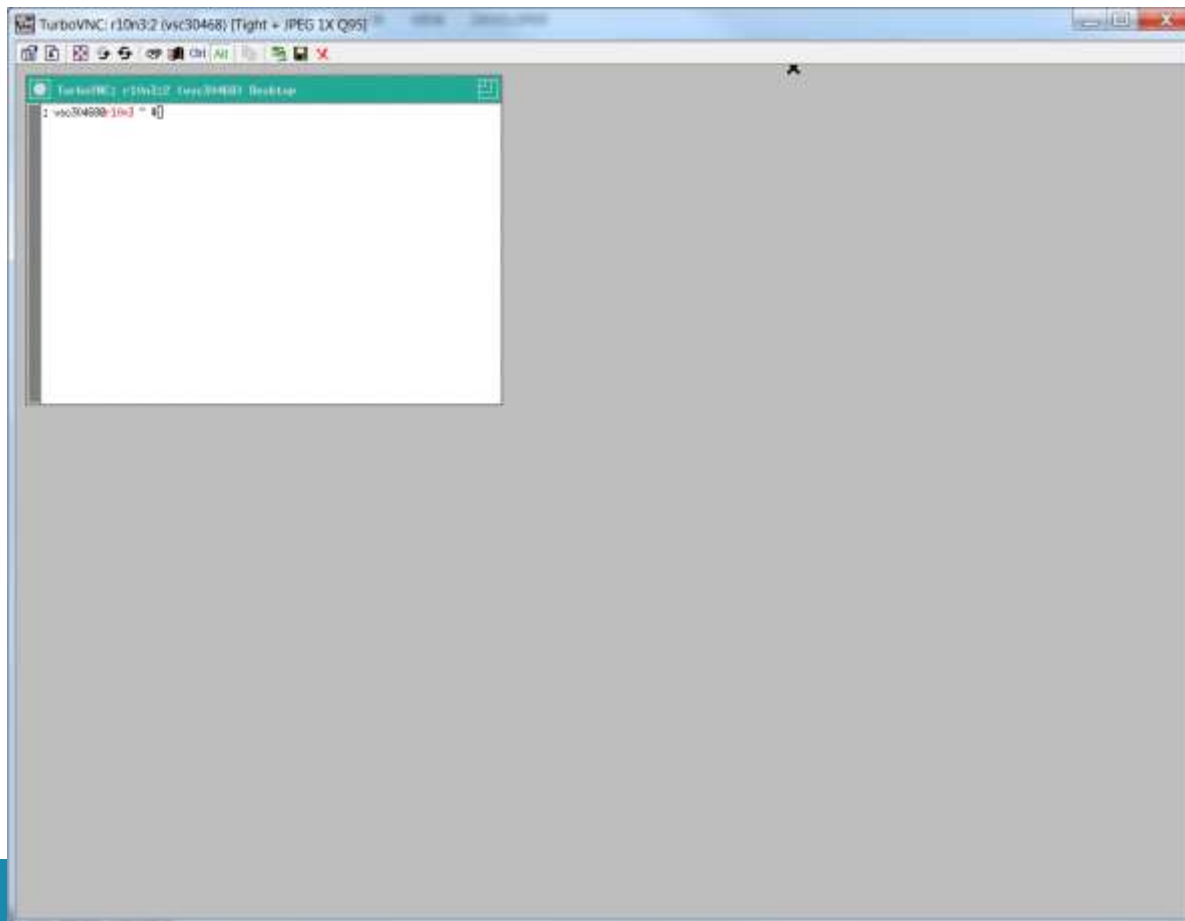
- Authenticate with your password



# How to use

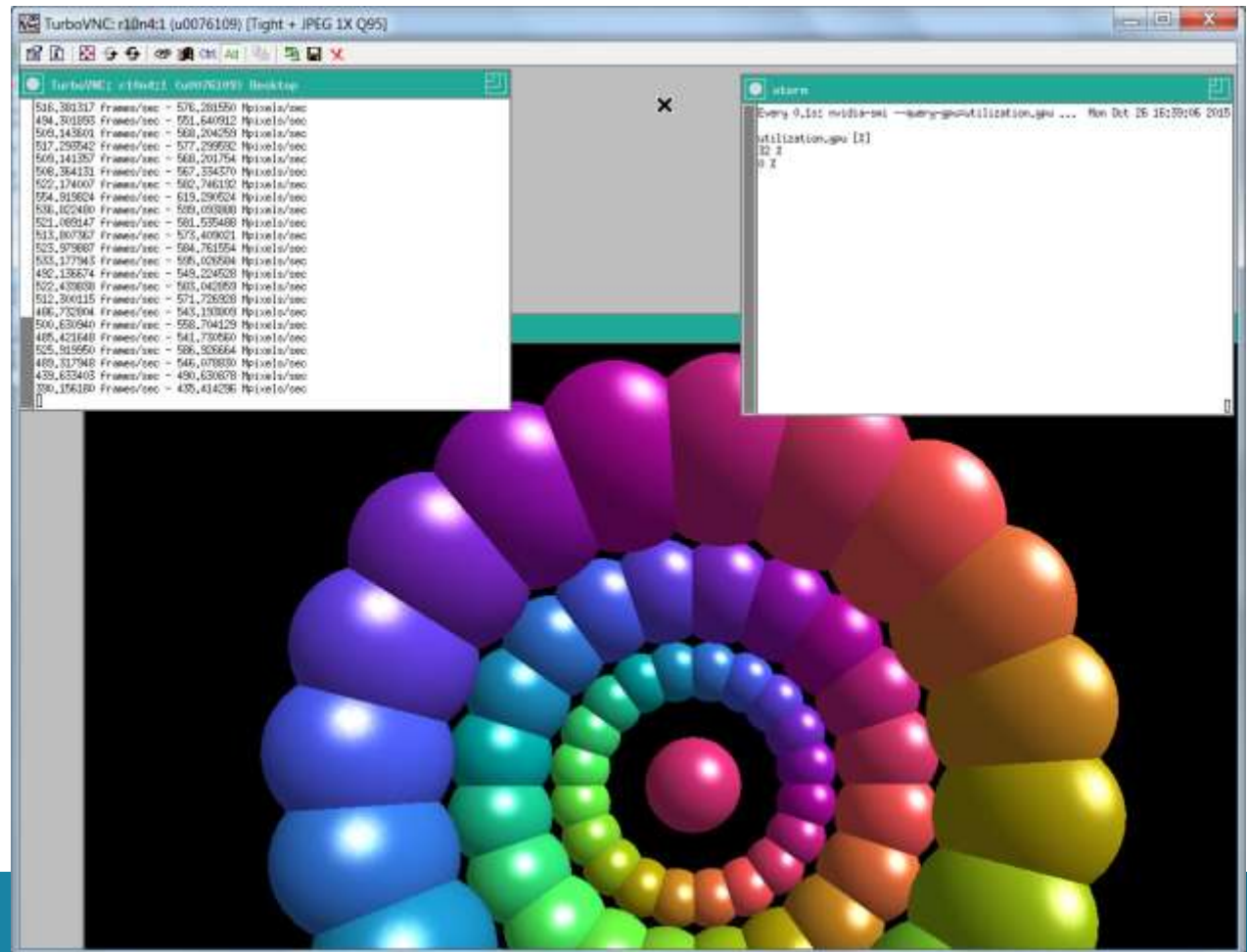
twm (Tab Window Manager) -> left click=menu

<https://en.wikipedia.org/wiki/Twm>



# How to use

- `$ module load VirtualGL/2.4-foss-2014a`
- `$ vglrun -d :0 glxspheres64`
- Some applications already have shortcuts
- `$ gpuwatch`



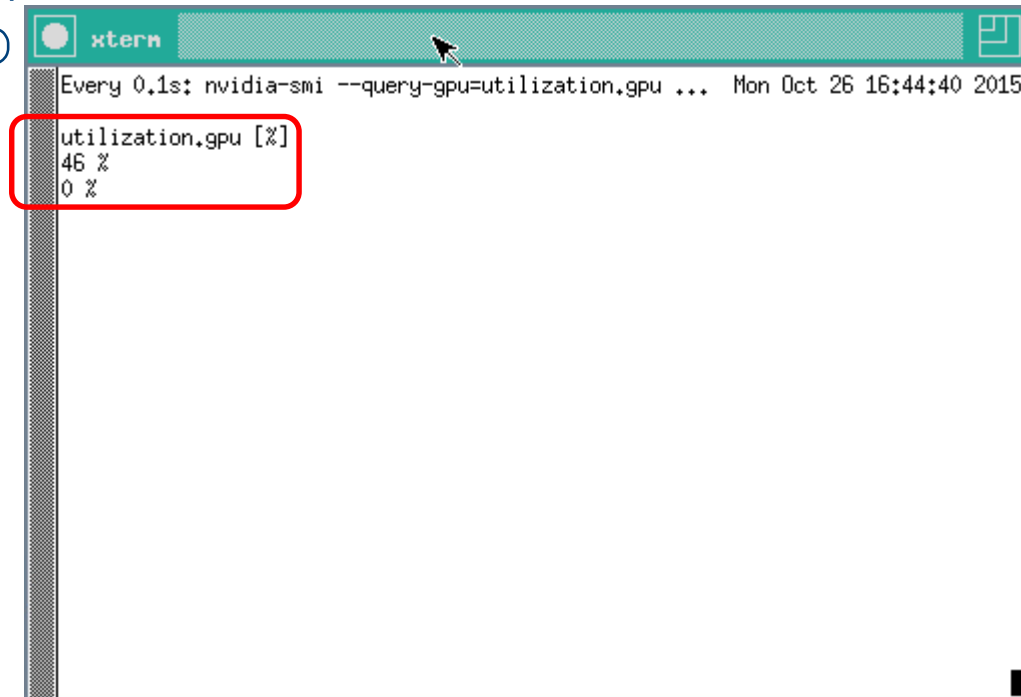
# How to use

- `$ module load VirtualGL/2.4-foss-2014a`
- `$ vglrun -d :0 glxspheres64`
- **All adjusted executables start with capital letter:**
  - `$ module load Paraview; Paraview` (instead of `vglrun -d :0 paraview`),
  - `$ module load VisIt; Visit` (instead of `vglrun -d :0 visit`)
  - `$ module load VMD; VMD` (instead of `vglrun -d :0 vmd`)



# How to use

- `$ module load VirtualGL/2.4-foss-2014a`
- `$ vglrun -d :0 glxspheres64`
- **All adjusted executables start with capital letter:**
  - Paraview (instead of `vglrun -d :0 paraview`),  
Visit, VMD
- `$ gpuwatch`



```
xtern  
Every 0.1s: nvidia-smi --query-gpu=utilization.gpu ... Mon Oct 26 16:44:40 2015  
utilization.gpu [%]  
46 %  
0 %
```

# How to use: summary

## 1. Request a visualization job

- `$ qsub -I -X -l partition=visualization`

## 2. On the visualization node (r10n3 or r10n4) load TurboVNC

- `$ module load TurboVNC/1.2.3-foss-2014a`

- **Create your password**

- `$ vncpasswd (mkdir .vnc; touch .vnc/passwd; vncpasswd if necessary)`

- **Start the server**

- `$ vncserver (-depth 32 -geometry 1600x1000)`

## 3. Establish the tunnel and VNC viewer connection

## 4. Load the module and start the application

- `$ module load Paraview; Paraview (or $ vglrun -d :0 paraview),`

## 5. Close the connection (on visualization node)

- `$ vncserver -kill :1; exit`

# Settings

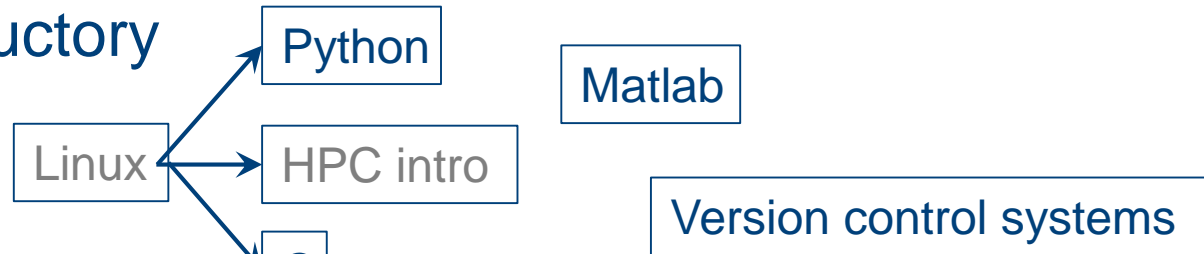
- 2 nodes
- 1 user on each node
- So far gratis
- Interactive jobs only (no batch)
- Max walltime 8 hrs (remember with qsub!)

# Visualization software

- Installed
  - Paraview - <http://www.paraview.org/>
  - VisIt - <https://wci.llnl.gov/simulation/computer-codes/visit/>
  - VMD - <http://www.ks.uiuc.edu/Research/vmd/>
  - Avizo (under trial license) - <http://www.fei.com/software/avizo3d/>
- If requested?
  - Blender - <http://www.blender.org/>
  - Vaa3D - <http://www.vaa3d.org/>
  - Tecplot (under license) - <http://www.tecplot.org/>
  - StarCCM+ (under license) - [http://www.cd-adapco.com/products/star\\_ccm\\_plus/](http://www.cd-adapco.com/products/star_ccm_plus/)
  - Ansys (under license) - <http://www.ansys.com>

# VSC training 2015/2016

- Introductory



- Intermediate

- Advanced

Lunchbox sessions

- visualization nodes
- ...

Debugging techniques

Demo

