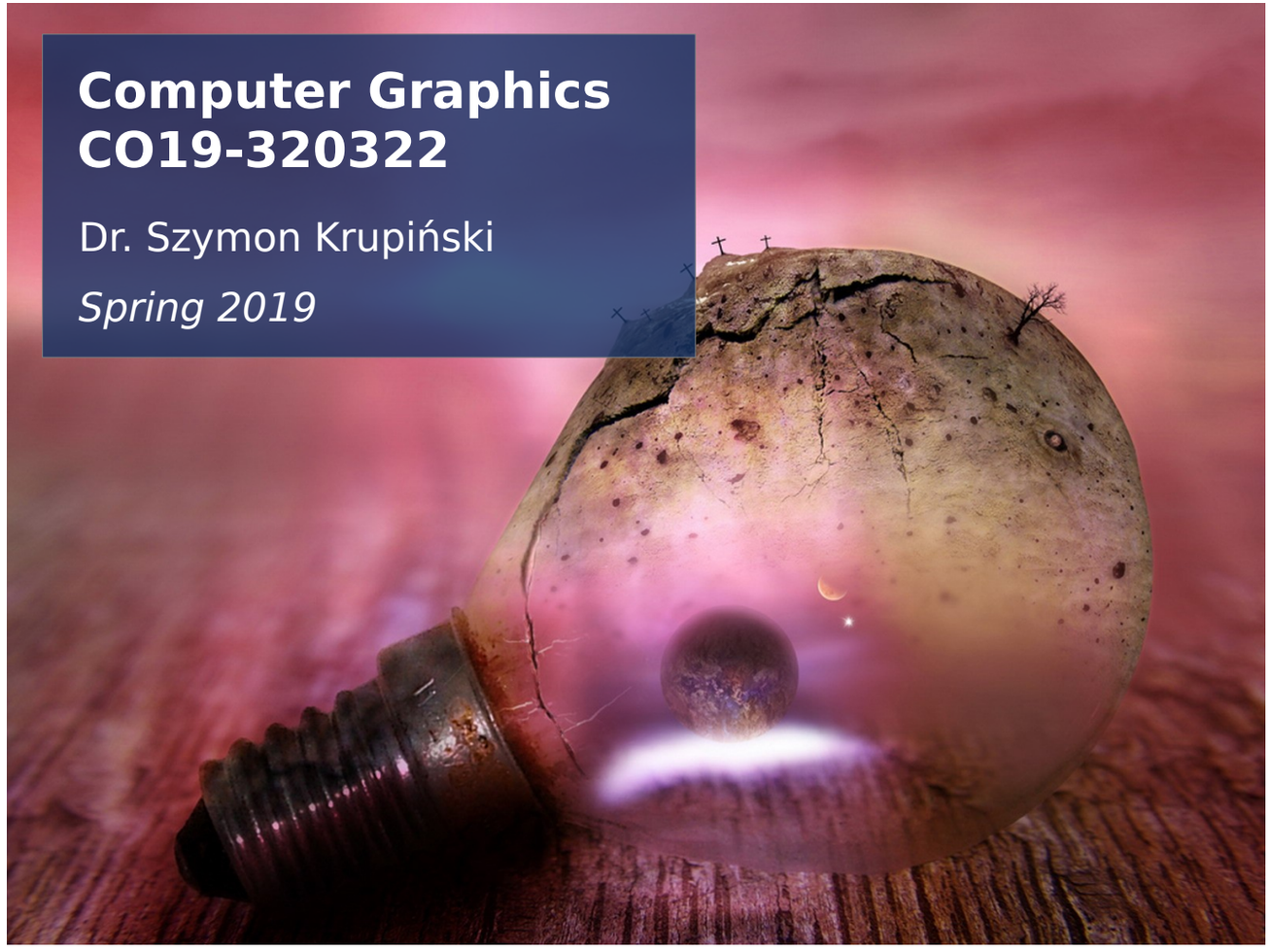
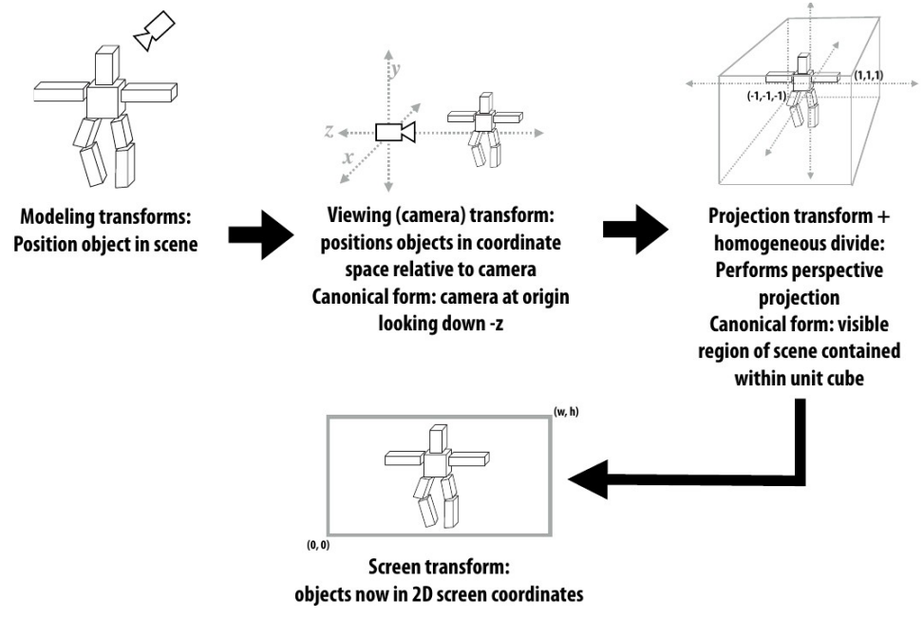
Lecture 5: OpenGL Basics

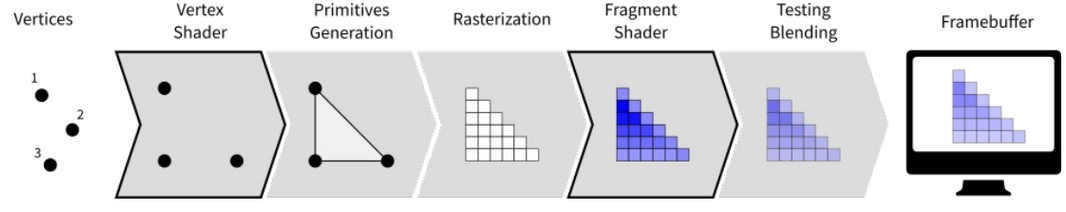
JACOBS UNIVERSITY



Up until now...

- How to model in 3-D
  - coordinates, transformations, triangular meshes...
- How to transform this into 2-D
  - projection, rastering, sampling...
- How it is done in real life?
  - CG pipeline
- Let’s take a break and dive into programming some OpenGL!



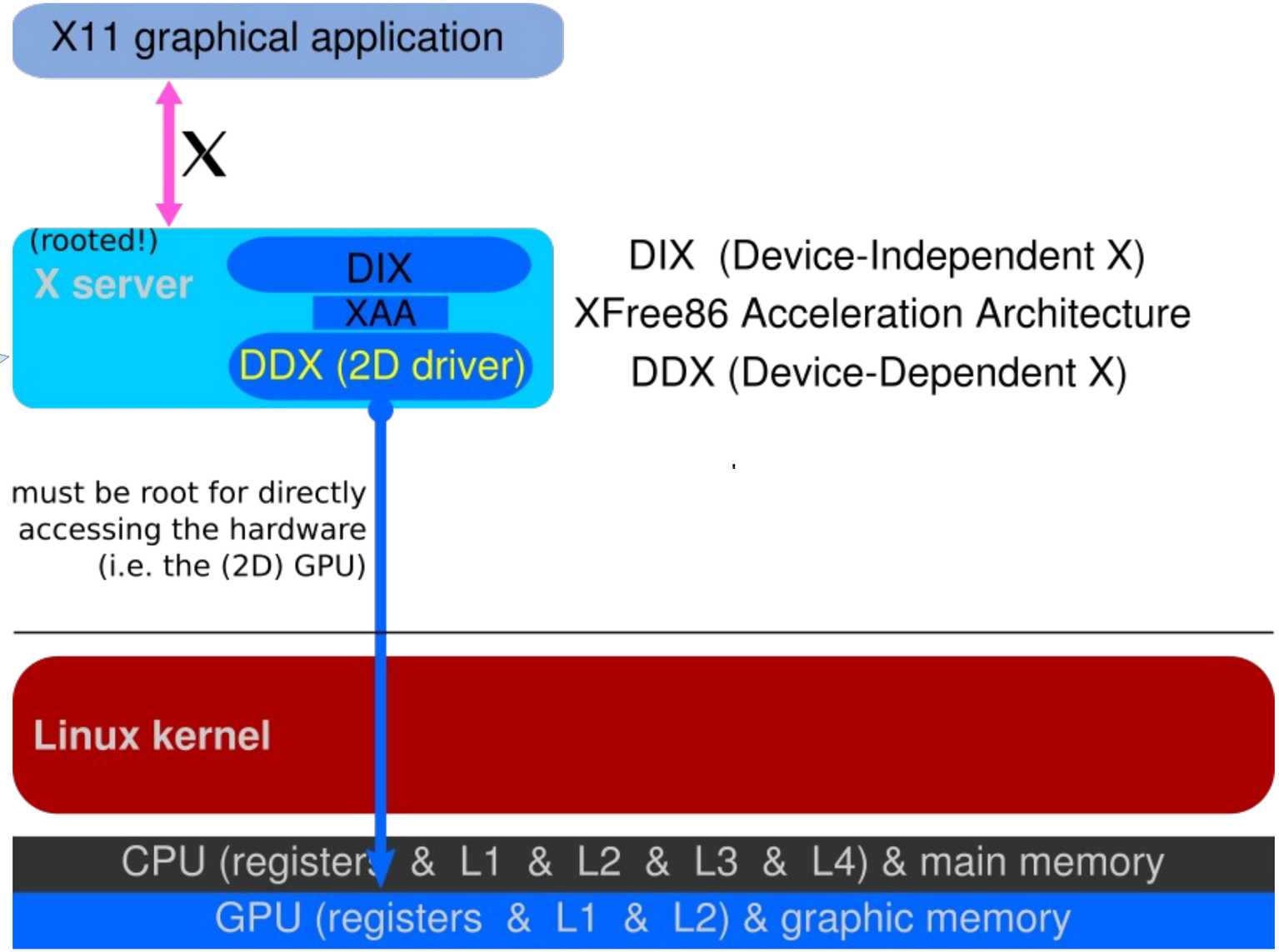


Graphic display architecture

- Question: how do we create a program to go from some points in space to displaying graphics on the screen?
- Sub-question: what is the software architecture for graphics?

CG, OS and hardware

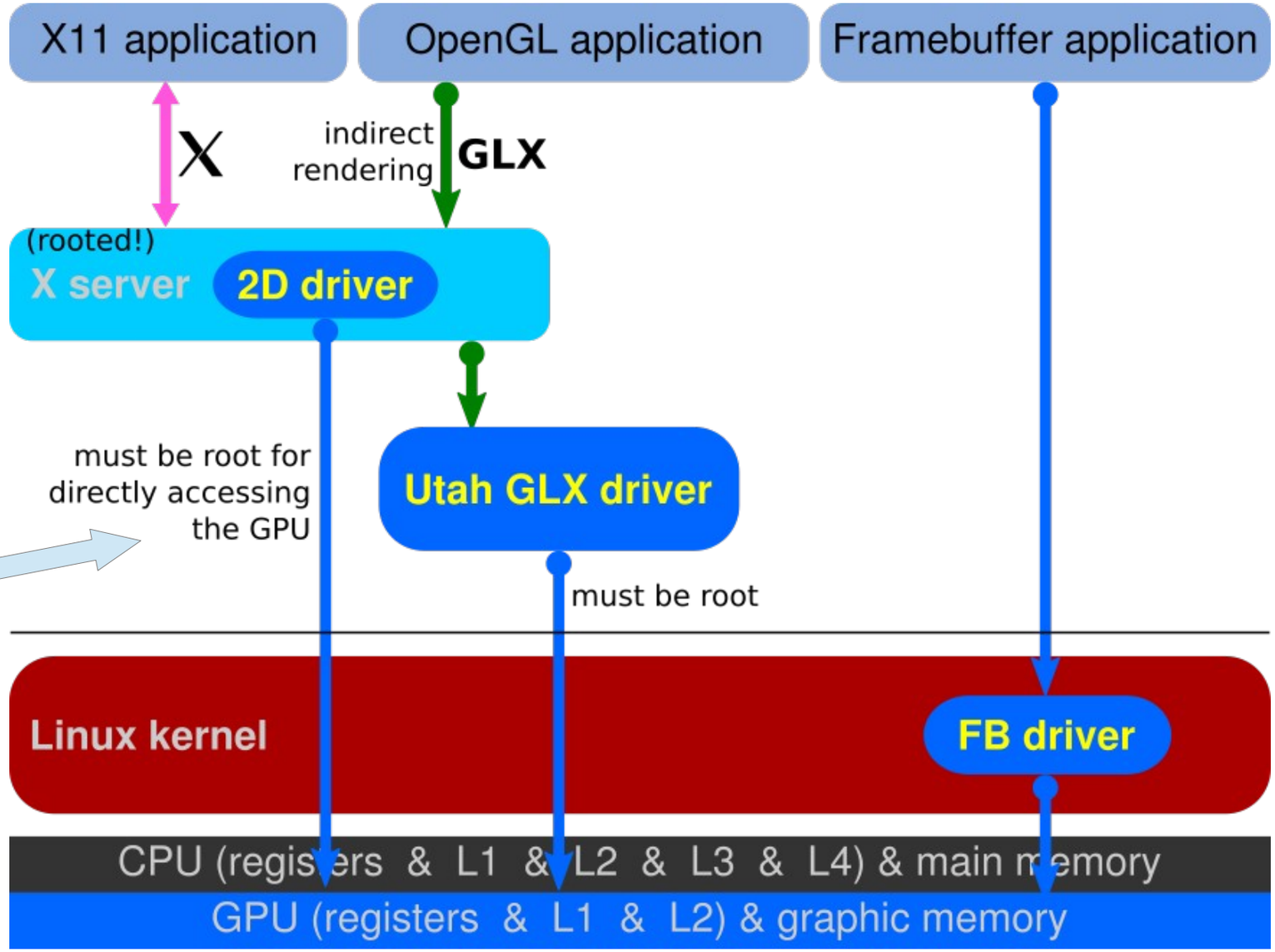
- Early stage



Managing programs' display zones and input devices (mouse, keyboard). Actual window management is done by a… window manager.

CG, OS and hardware

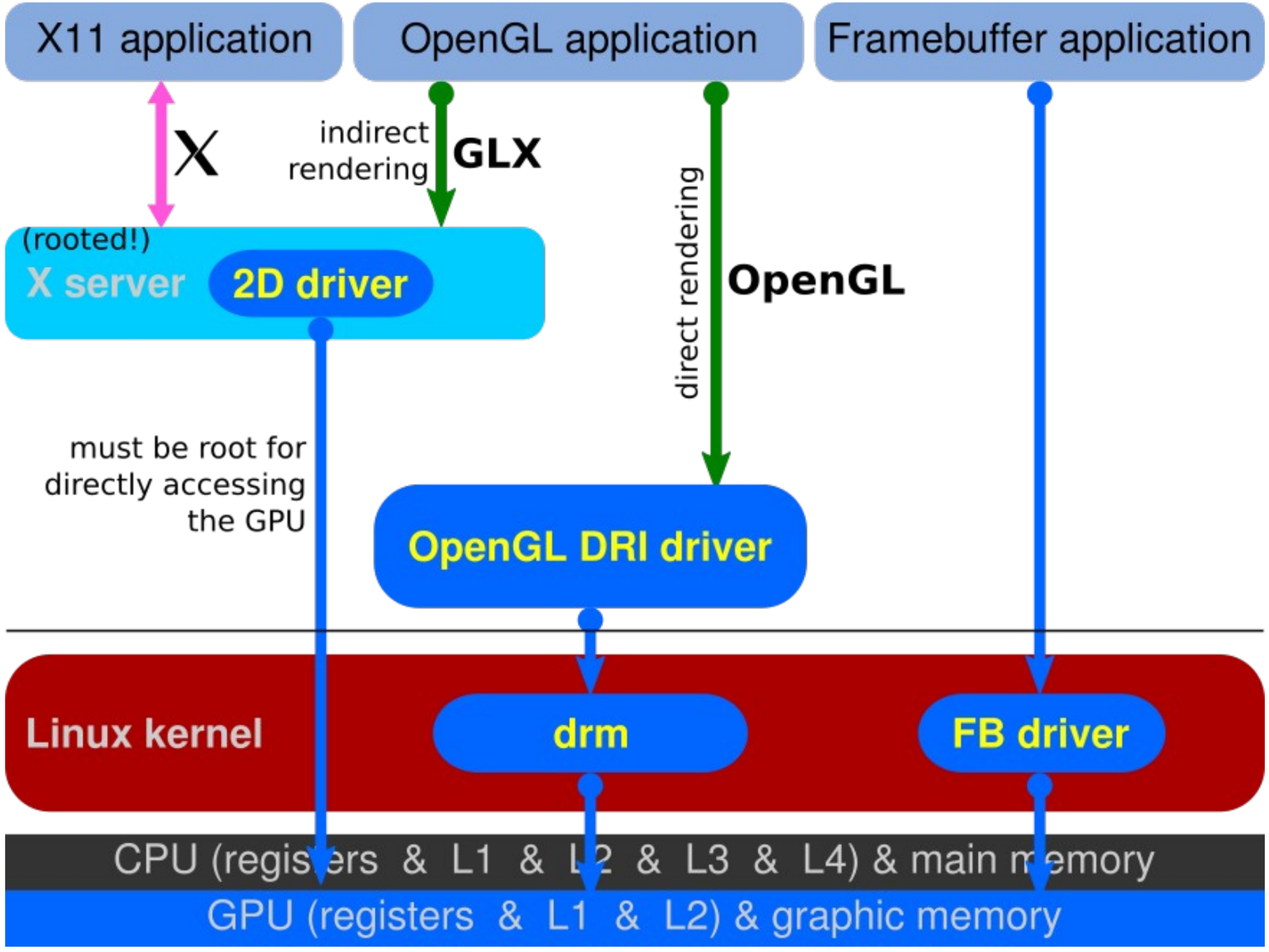
- “UTAH GLX”



Comes from UNIX way of enforcing security. On Windows every user could do it.

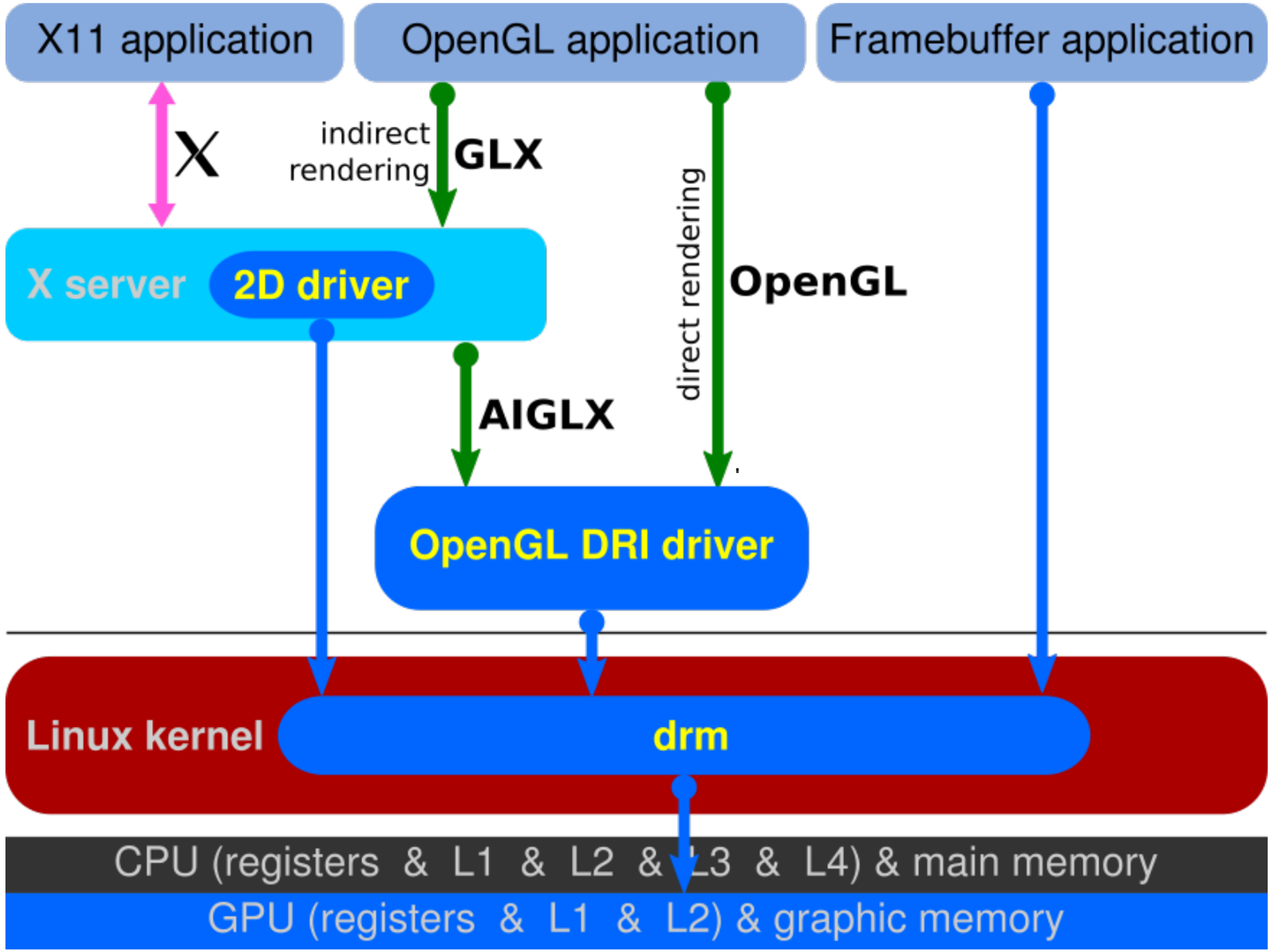
CG, OS and hardware

- Direct Rendering Interface



CG, OS and hardware

- Proper Access Right Model

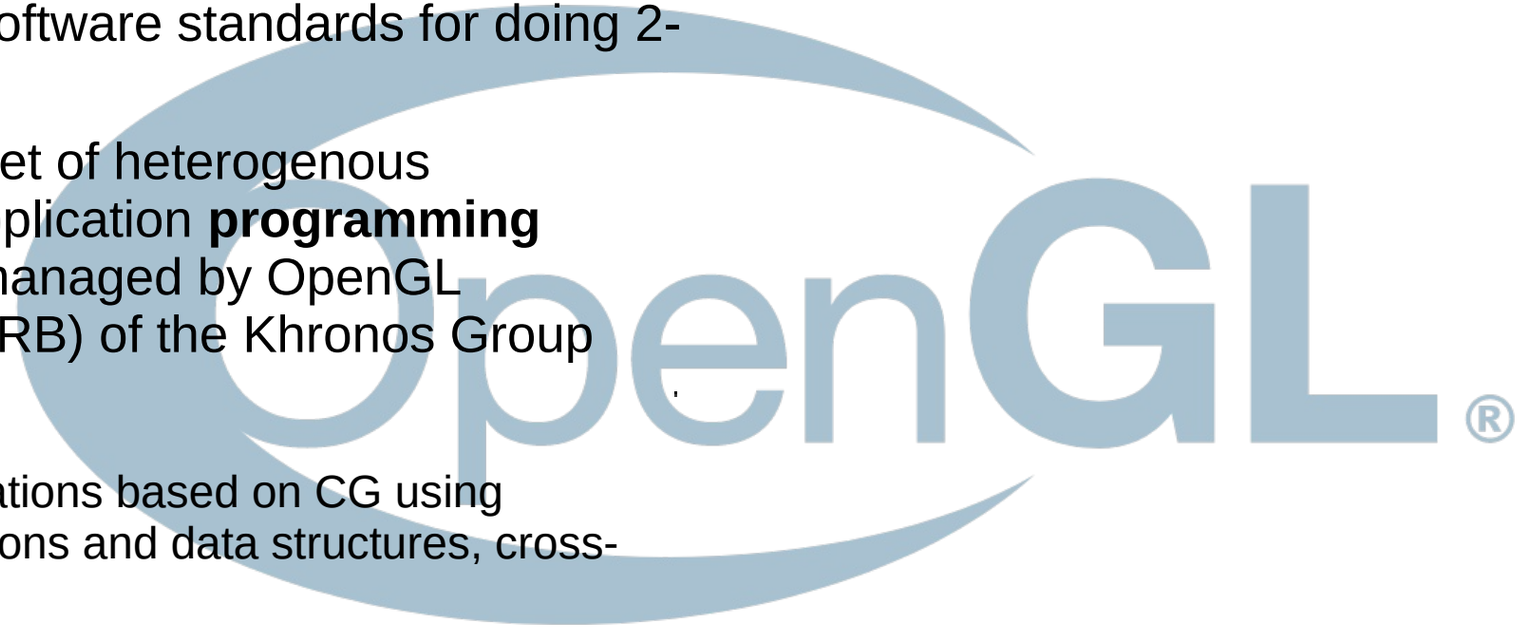


Other CG sinks

- Of course, immediate and real-time display of computer generated graphics is not the only destination
- Animation, CG special effects in movies and high fidelity synthetic image rendering are all “off-line” applications
  - Can be executed by large, massively parallel render server farms
  - Factoids of the day:
    - CGI in Cars 2 (Pixar, 2011) required a render farm containing 12,500 CPU cores and on average, it took 11.5 hours to render a single frame
    - Transformers: Dark of the Moon (DreamWorks, 2011) - It took 288 CPU hours per frame to render the Driller (consists of 70,051 polygons) along with the photoreal CG building that includes all those reflections in its glass

OpenGL

- Created by Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released the **standard** one year later
- One of the most widespread software standards for doing 2-D and 3-D vector graphics CG
- Not just a library but rather a set of heterogenous implementations under one application **programming interface** (API) specification managed by OpenGL Architectural Review Board (ARB) of the Khronos Group
- Overall goal:
  - allow developers to write applications based on CG using (relatively) standard set of functions and data structures, cross-language, cross-platform
  - Get highest performance possible by optimal algorithms and maximal use of **hardware acceleration** / **GPU**
  - (dominate the market and drive development of CG)



OpenGL implementations

- It has many existing implementations = actual software libraries with (à priori) identical functionality
  - GPU developers (Nvidia, AMD) write their own versions optimised for and containing routines specific for their hardware (typically bundled with drivers)
  - OS developers do, too (Apple)
  - Open source implementation: Mesa
- Most recent version of the API definition: 4.6
- It has sister projects and competitors
  - Direct3D (Microsoft)
  - Metal (Apple)
  - OpenGL ES (embedded systems)
  - GPU programming (CUDA, etc)

OpenGL API family

- **GLU** (OpenGL Utility Library)
  - consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides
  - Example functionality: mapping between screen- and world-coordinates, generation of texture mipmaps, drawing of quadric surfaces, NURBS, tessellation of polygonal primitives, interpretation of OpenGL error code, etc.
  - generally in more human-friendly terms than the routines presented by OpenGL
  - all GLU functions start with the `glu` prefix. Example: function `gluOrtho2D` which defines a two dimensional orthographic projection matrix

OpenGL API family

- **GLUT** (OpenGL Utility Toolkit)
  - library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system
  - Not officially part of OpenGL standard
  - example functionality: window definition, window control, and monitoring of keyboard and mouse input, routines for drawing a number of geometric primitives such as cubes, NURBS or spheres (solid/wireframe mode) etc.
  - objectives: 1) portable code between operating systems (GLUT is cross-platform) and 2) make learning OpenGL easier
  - All GLUT functions start with the `glut` prefix. Example: `glutPostRedisplay` marks the current window as needing to be redrawn

OpenGL API family

- **PyOpenGL** (Python OpenGL module)
  - the most common cross platform Python binding to OpenGL and related APIs
  - Supports OpenGL v1.1 to 4.4
  - GLUT, FreeGLUT
  - GLES, GLU, EGL, WGL, GLX, GLE 3

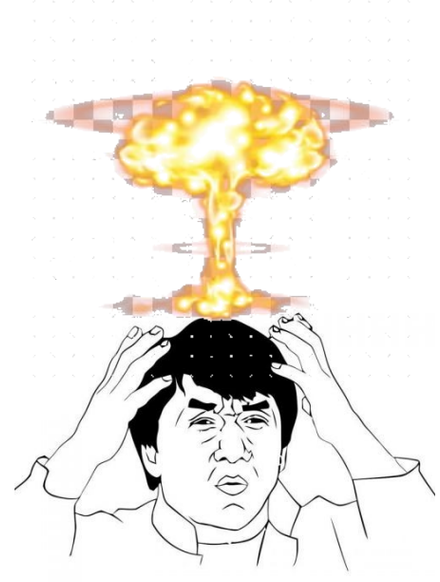
"Glue" libraries between windows systems and CG (AGL, GLX, WGL) and other

OpenGL API family

- **GLEW** (OpenGL Extension Wrangler Library)
  - provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform

OpenGL extensions

- The extensions are new pieces of functionality that hardware vendors or others bring into OpenGL
- Each extension is associated with a short identifier, based on the name of the company which developed it
  - For example, Nvidia's identifier is `NV`, which is part of the extension name `GL_NV_half_float`, the constant `GL_HALF_FLOAT_NV`, and the function `glVertex2hNV()`
- There is about 300 extensions...

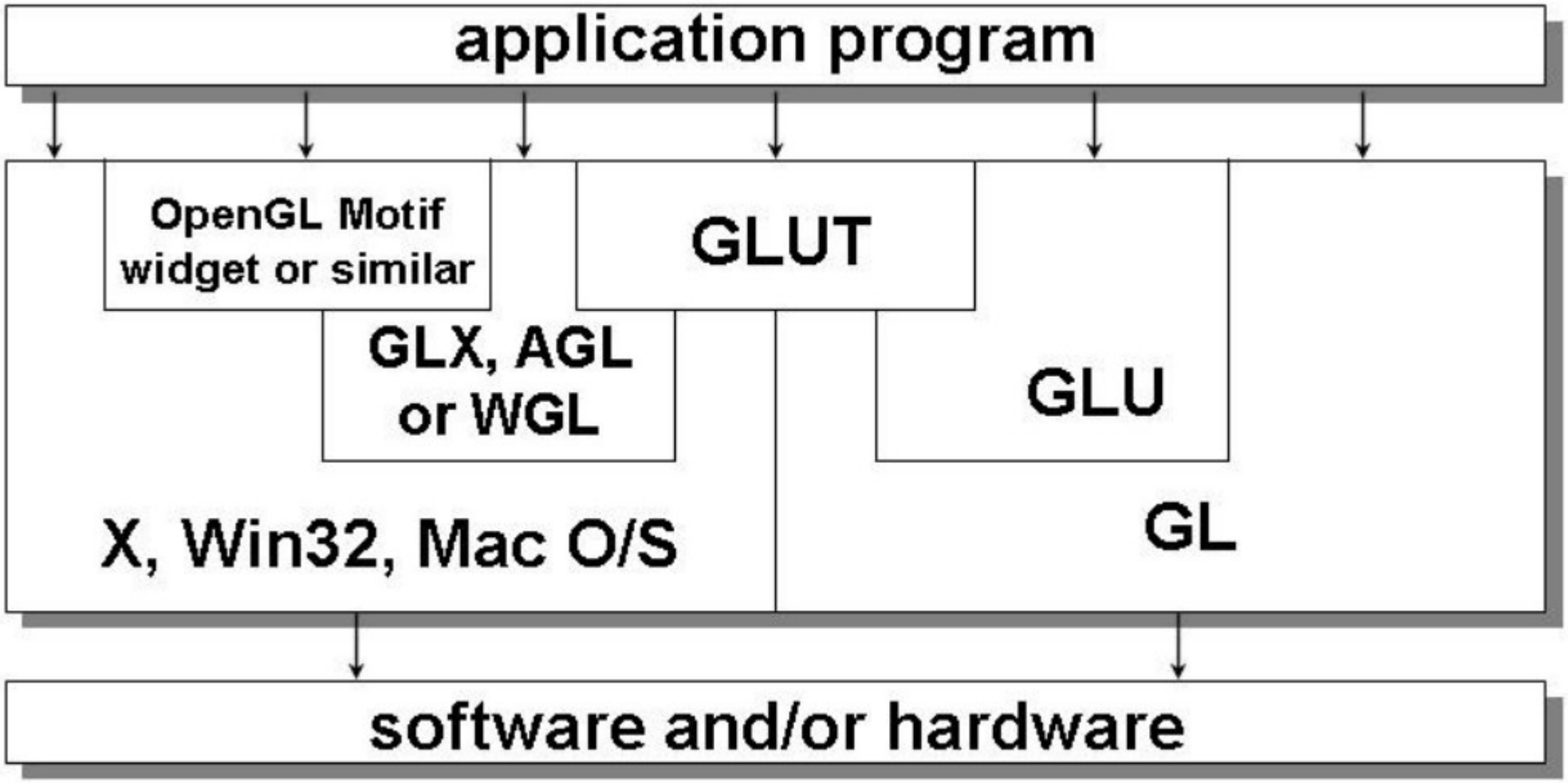


It’s not over!

Useful to know:

- **GLM** (OpenGL Mathmatics Library):
  Used for vector classes, quaternion classes, matrix classes and math operations that are suited for OpenGL applications.
- **SOIL** (Simple OpenGL Image Library):
  A library that provides functions to load textures from disc as easy as possible. This library is ideal for loading images to be used by OpenGL

OpenGL – general picture of the core libraries

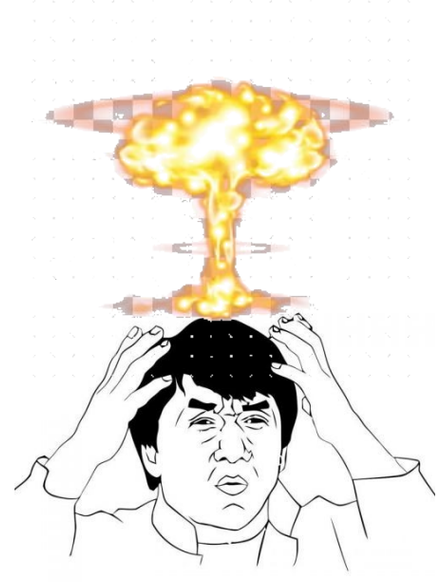


Now, a calming pause

- Don’t be alarmed!
  - the installation will work fine and without complicated setup
  - you will not have to know the details of most of this!
  - there will be cake at the end...

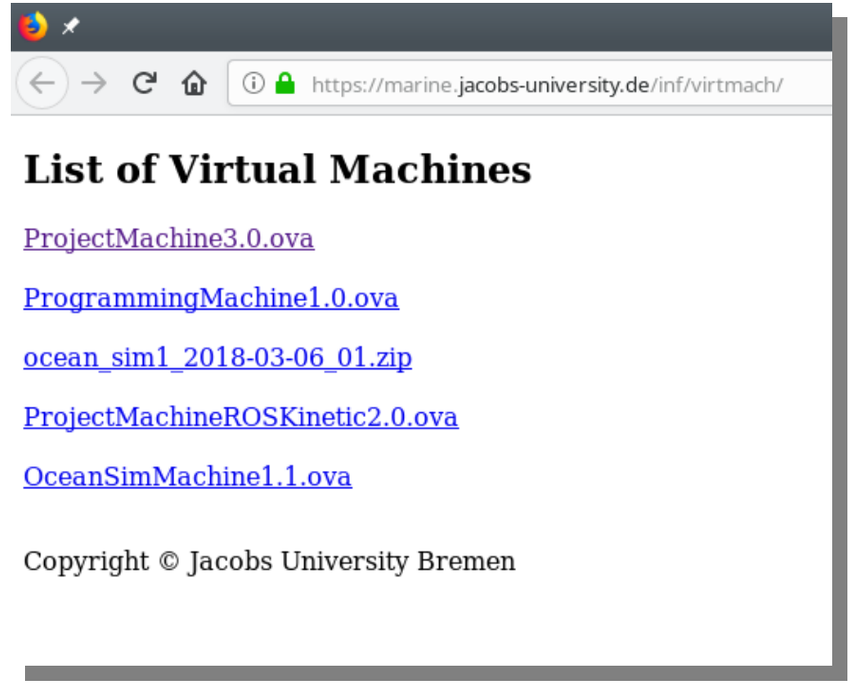
OpenGL Shading Language

- This course will not only teach you how to use the calls to the OpenGL library – you will learn (a little bit of) a new programming language, too!
- A big part of the CG pipeline is little programs running directly on the GPU called **shaders** used for
  - In general/initially: production of appropriate levels of light, darkness, and color within an image
  - In the modern CG pipeline: all of the above plus...
    - Geometry evaluation and modification
    - Fragment processing
    - Even running general calculations
- OpenGL uses **GLSL** for writing these programs in a language which is similar to C and saves you from writing in assembly-like code for GPU

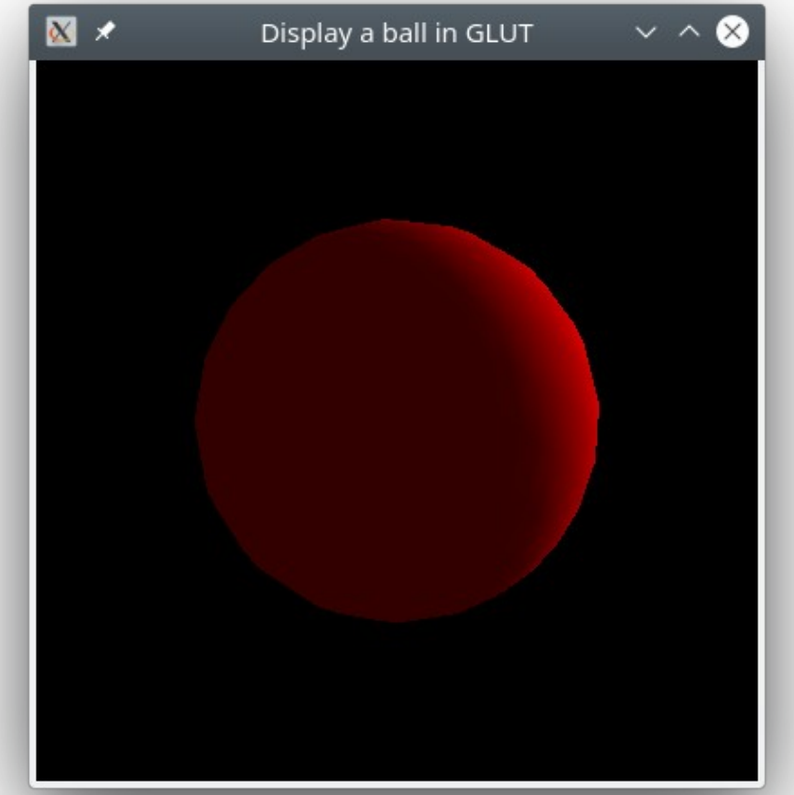


OpenGL setup

- As you please BUT it must work on our computers
- You need a **GLUT** development environment which will allow you to compile programs using GLUT (and thus OpenGL), so just a working OpenGL installation will not suffice
- **FreeGLUT** is the most suitable version
- I have prepared a virtual machine with full setup of dev libraries
  - Download from: https://marine.jacobs-university.de/inf/virtmach/ProgrammingMachine1.0.ova
  - VirtualBox software will be needed (Oracle, open source), available for Window, Linux and MacOS
  - Runs Ubuntu system, user name and password: “user”



Hello world!



- Enough theory!

(all code below available on moodle)

myfirstcode.c – a simple C program using GLUT to display a shaded red ball

Compilation on Unix:

```
user@system:~$ gcc -lglut -lGLU -lGL -o myfirstcode myfirstcode.c
```

Btw: the C++ compiler will work, too:

```
user@system:~$ g++ -lglut -lGLU -lGL -o myfirstcode myfirstcode.c
```

Running:

```
user@system:~$ ./myfirstcode
```

Hello world!

Python equivalent:

myfirstcode.py

Running:

```
user@system:~$ python myfirstcode.py
```

Hello world!

Using OpenGL primitives to set up the object to display

(GLUT call still used to do the initialisation and
GLU call to manage the perspective matrix!)

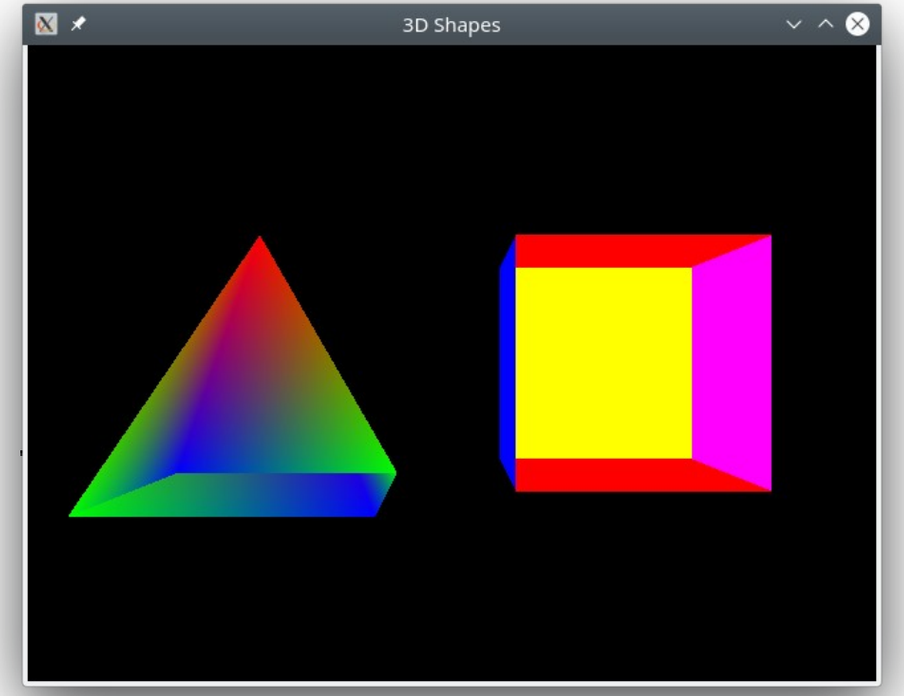
mypurecode.c

Compiling:

```
user@system:~$ gcc -lglut -lGLU -lGL -o
mypurecode mypurecode.c
```

Running:

```
user@system:~$ ./mypurecode
```



Thank you!

- Questions?