NVISION 08
THE WORLD OF VISUAL COMPUTING

OpenGL and the Future

Michael Gold, Mark Kilgard and Barthold Lichtenbelt

Agenda

- **OpenGL 3.0 / GLSL 1.30 (Barthold Lichtenbelt)**
  - blichtenbelt@nvidia.com
- **OpenGL 3.0 and Cg 2.1 (Mark Kilgard)**
  - mjk@nvidia.com
- **CUDA <-> OpenGL interop (Michael Gold)**
  - gold@nvidia.com

OpenGL 3.0

- **Announced two weeks ago**
- **Support for latest generations of Programmable Hardware**
  - Installed base > 60 Million units
- **New deprecation model with profiles**
  - Streamline the API
- **Full interoperability with OpenCL and CUDA**
  - Access to compute
- **Collaboration among hardware vendors and software vendors**
  - Solving real needs
- **Cross platform**
  - Windows XP and Vista, Linux, Mac OS, ...

OpenGL 3.0 new features

- Forward-looking context
- Greater VBO performance
- FBO and related extensions
- Conditional rendering
- Transform feedback
- FP internal formats for textures, renderbuffers
- Half-float (16-bit) vertex and pixel data formats
- Array textures
- One and two-channel (R and RG) internal formats for textures and renderbuffers
- RGTC internal compressed texture formats, packed float and texture shared exponent
- sRGB framebuffer support

GLSL 1.30

- **Native integer support**
  - bitwise operators, texture return values, uniforms, shader IO
- **Expanded texturing support**
  - Size queries, offsets, explicit LOD and derivative control, texture arrays, integer support
- **Switch statements**
- **Several new built-in functions**
  - Hyperbolic trig functions
  - trunc(), round(), roundEven(), isnan(), isinf(), modf()
  - Integer related: sign(), min/max(), abs(), ….
- **Pre-processor token pasting (##)**
- **User-defined fragment outputs**
- **Non-perspective interpolation of varyings**
- **gl_VertexID vertex shader input**

Extensions for OpenGL 3.0

Feature	Extension for OpenGL 3.0
Platform extension support for managing OpenGL 3.0 contexts	`{WGL|GLX}_ARB_create_context`
Geometry shaders to modify vertices and/or generate new vertices and primitives	`ARB_geometry_shader4`
Large 1D table lookups for GLSL	`ARB_texture_buffer_object`
Instanced primitive rendering for OpenGL 3.0 capable hardware	`ARB_draw_instanced`

Extensions for OpenGL 2.x

Feature from OpenGL 3.0	Extension for OpenGL 2.x
All framebuffer object functionality	`ARB_framebuffer_object`
16-bit floating point vertex formats	`ARB_half_float_vertex`
sRGB color space rendering	`ARB_framebuffer_sRGB`
More efficient buffer mapping	`ARB_map_buffer_range`
1 and 2 component texture compression	`ARB_texture_compression_rgtc`
Efficient vertex array state management	`ARB_vertex_array_object`
1 and 2 component render-to-texture	`ARB_texture_rg`
Vertex array instancing for OpenGL 2.x capable hardware	`ARB_instanced_arrays`

Deprecated features

- **OpenGL has never removed features**
  - Commitment to backwards compatibility is one of OpenGL's strengths
  - After 15+ years, defining new features to work with old features becomes increasingly difficult
- **OpenGL 3.0 <u>does not remove</u> any features**
- **OpenGL 3.0 does mark certain features as deprecated**
  - Redundant, Legacy and obsolete features
  - Parts of OpenGL unlikely to be accelerated
- **Future OpenGL revisions will remove these deprecated features**
  - Guidance to developers to prepare for future revisions
  - Plan to remove these features sooner, rather than later.

Deprecated features

- **Fixed-function vertex and fragment processing**
- **Color-index mode**
- **Display lists, and Selection and Feedback modes**
- **GLSL 1.10 and 1.20**
- **Begin/End based rendering**
- **Application-generated object names**
- **Quads and polygon primitives**
- **Polygon and Line Stipple**
- **Pixel transfer modes**
- **Bitmaps, DrawPixels, PixelZoom**
- *and quite a few others*…
  - See **Appendix E** of OpenGL 3.0 specification for the list

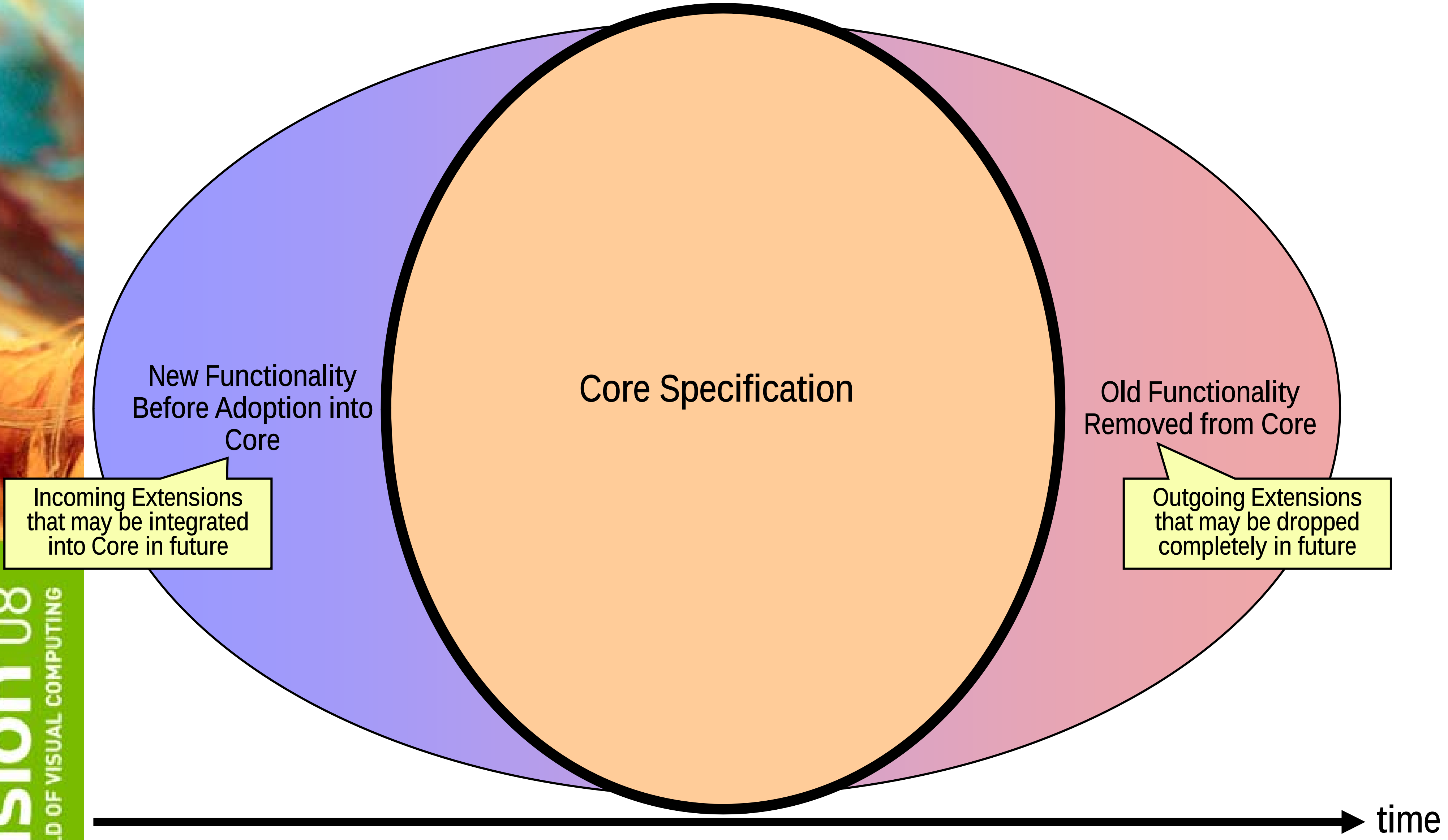
Deprecation mechanism

- **Step 1 Core feature**
  - In core, fully supported. **Will** be in the next API version
- **Step 2 Core (Deprecated feature)**
  - In core, marked as deprecated
  - **May** be fully or partly removed in a later version
  - New features need not define interactions with deprecated ones
- **Step 3 ARB approved Extension**
  - **Removed** from core -> an ARB extension (no suffix)
  - Extension spec identifies the removed functionality
  - Vendors may support the extension if markets require it
- **Step 4 Removed from ARB extension list**
  - Could be an EXT or vendor extension, if vendor markets still require it (still no suffixes required)

Deprecation mechanism

- **Features will be deprecated for at least one spec release (step 2) before being removed**
- **Extension Path: `Vendor/EXT->ARB->Core`**
  - With possible API / functionality changes as we learn from experience
- **Deprecation Path: `Core->ARB->EXT/Vendor`**
  - No API or functionality changes

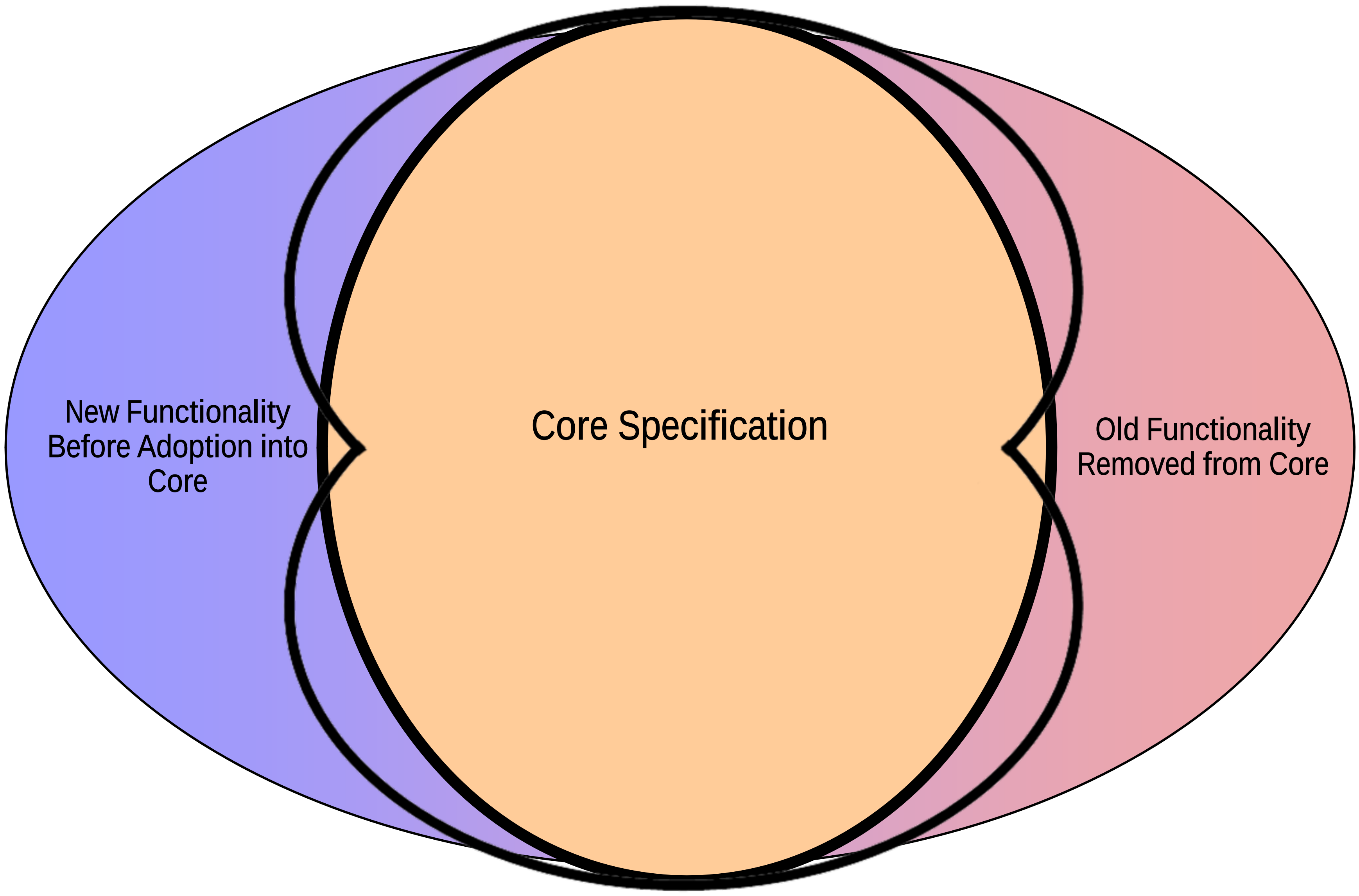
Feature Evolution Model - Deprecation


New Functionality Before Adoption into Core
Incoming Extensions that may be integrated into Core in future
Core Specification
Old Functionality Removed from Core
Outgoing Extensions that may be dropped completely in future
time

Profiles

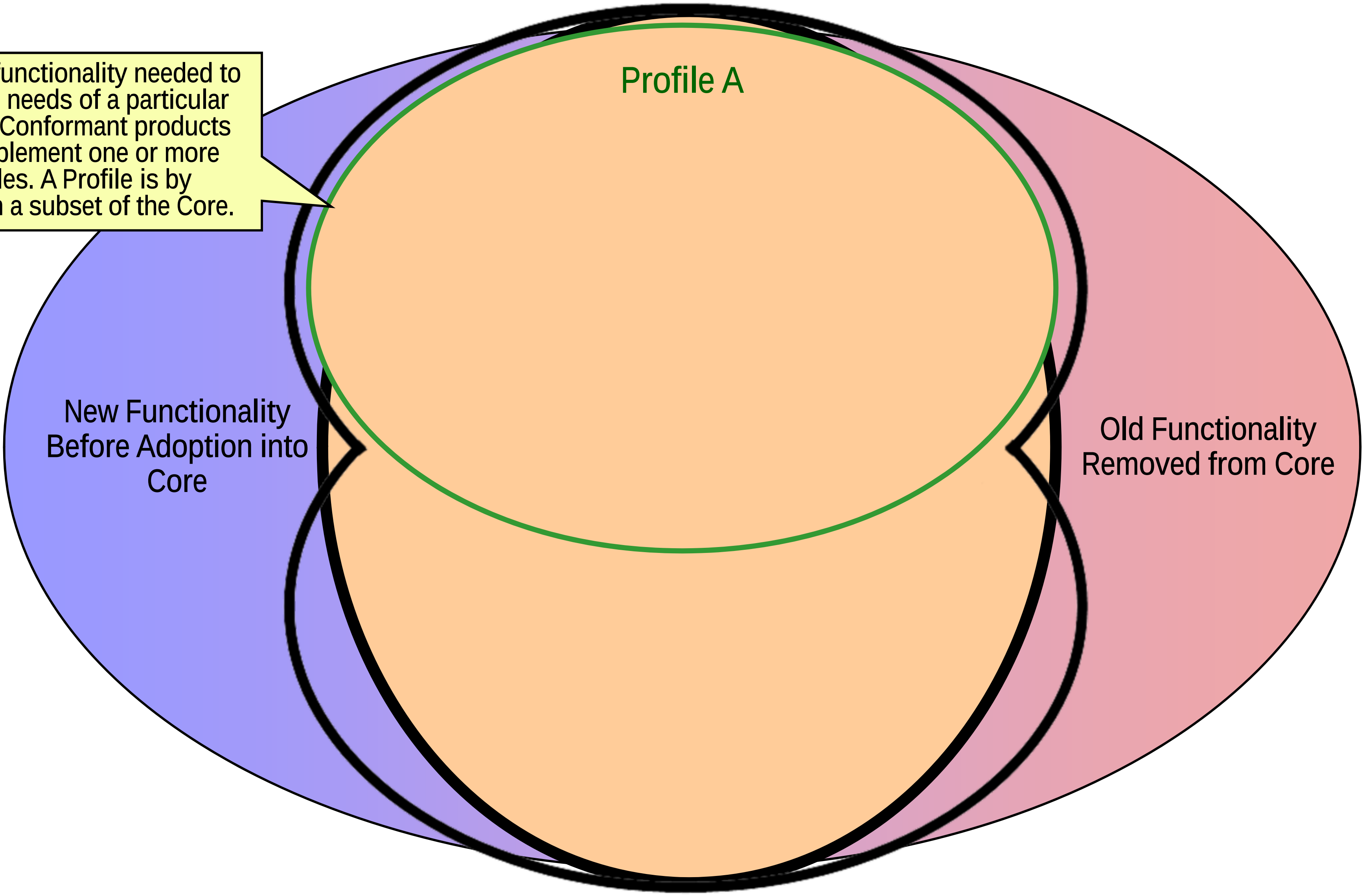
- **Encapsulates a set of functionality**
- **Optional to implement for vendors**
- **Sum of all profiles makes up the Core spec**
  - OpenGL 3.0 is one big profile
- **Deprecation mechanism is applied per profile**
- **Only the OpenGL ARB can define profiles**
- **Currently discussing need for “workstation” profile**
  - Could contain most of the deprecated functionality
  - Need input from you!

Evolution Model - Profiles


New Functionality Before Adoption into Core
Core Specification
Old Functionality Removed from Core

Core Specification
The sum of all Profile functionality

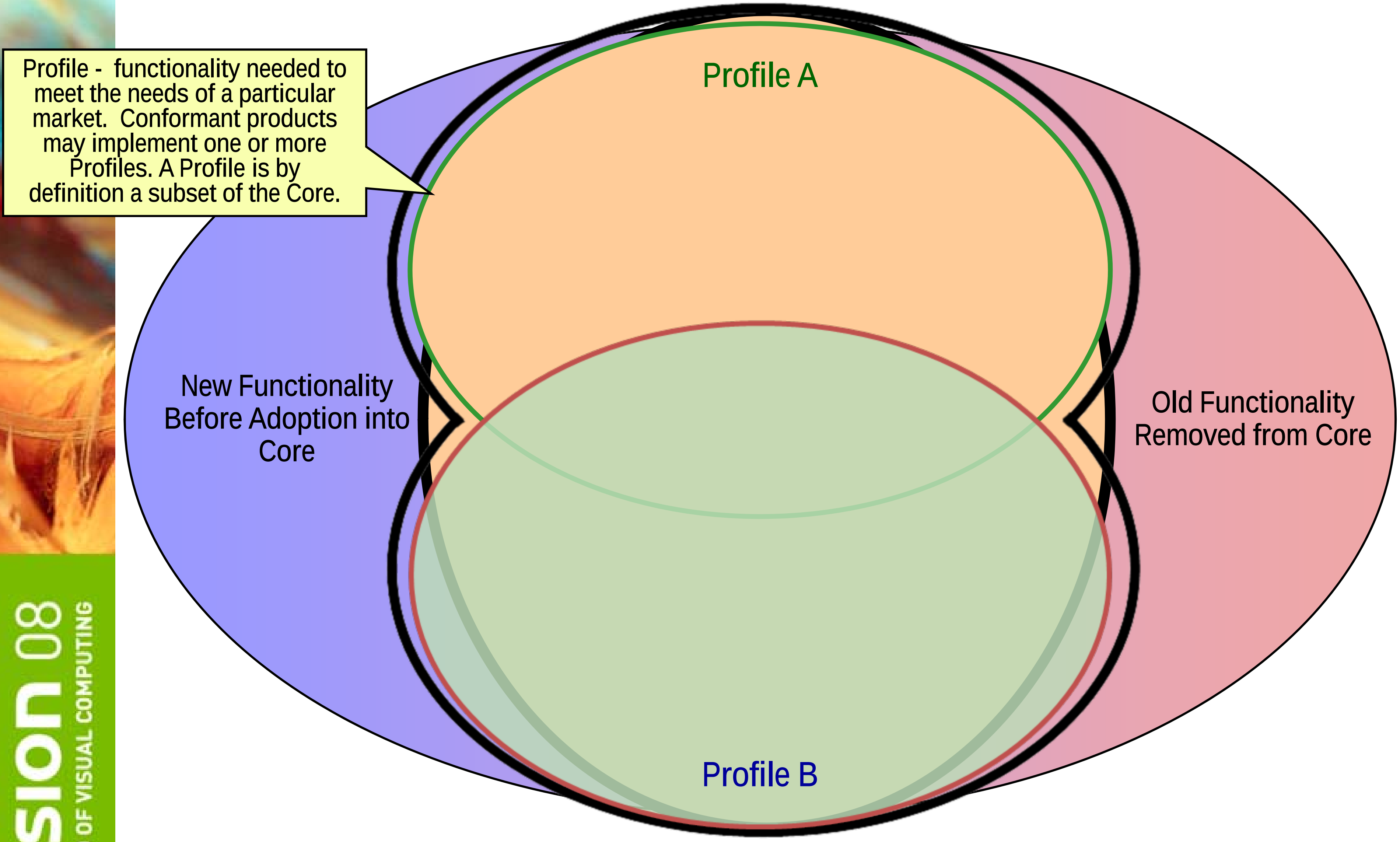
Evolution Model - Profiles


Profile A
New Functionality Before Adoption into Core
Old Functionality Removed from Core

Profile - functionality needed to meet the needs of a particular market. Conformant products may implement one or more Profiles. A Profile is by definition a subset of the Core.

Core Specification
The sum of all Profile functionality

Evolution Model - Profiles

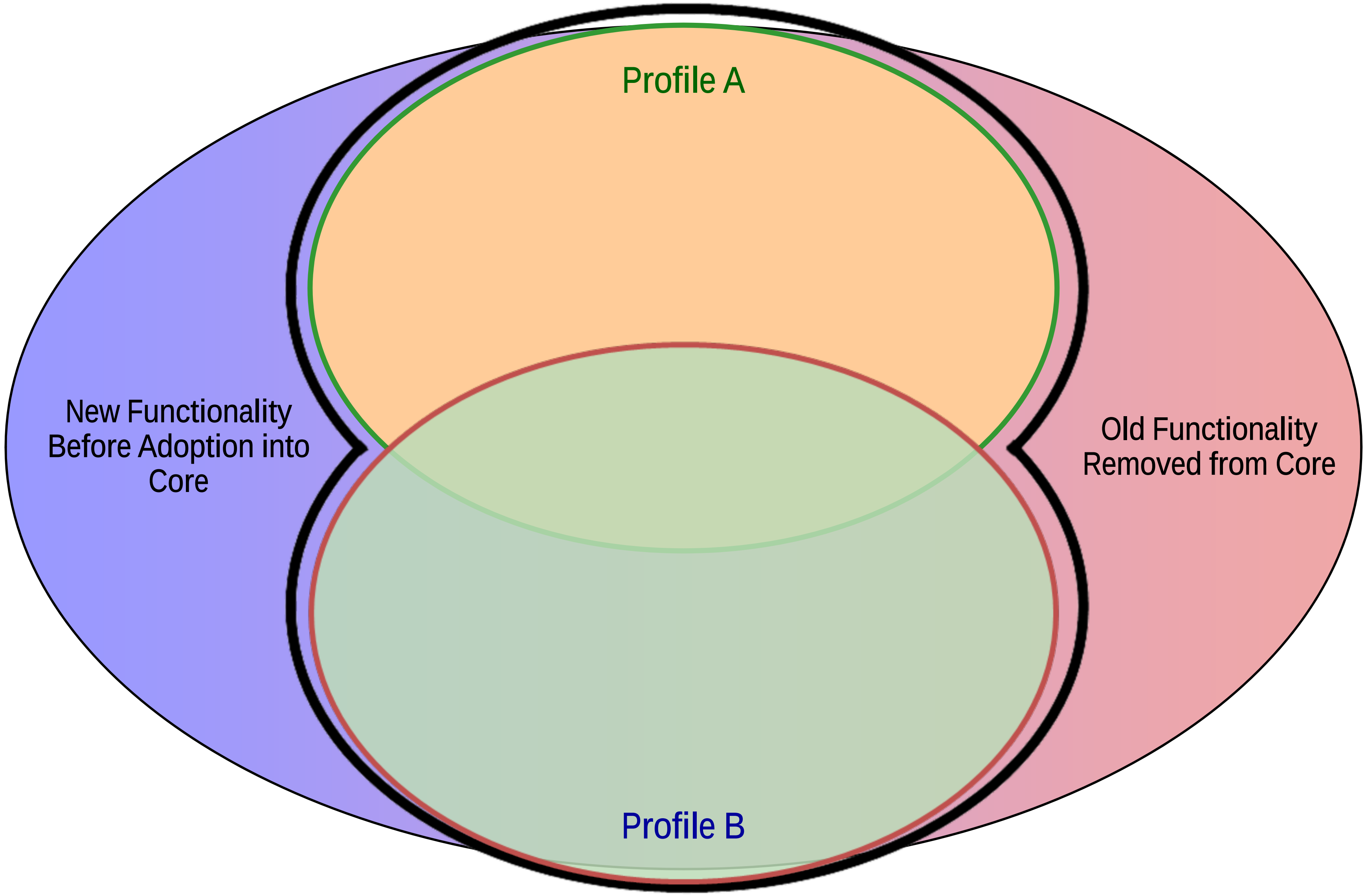

Profile - functionality needed to meet the needs of a particular market. Conformant products may implement one or more Profiles. A Profile is by definition a subset of the Core.
Profile A
New Functionality Before Adoption into Core
Old Functionality Removed from Core
Profile B

Core Specification
The sum of all Profile functionality

Context types

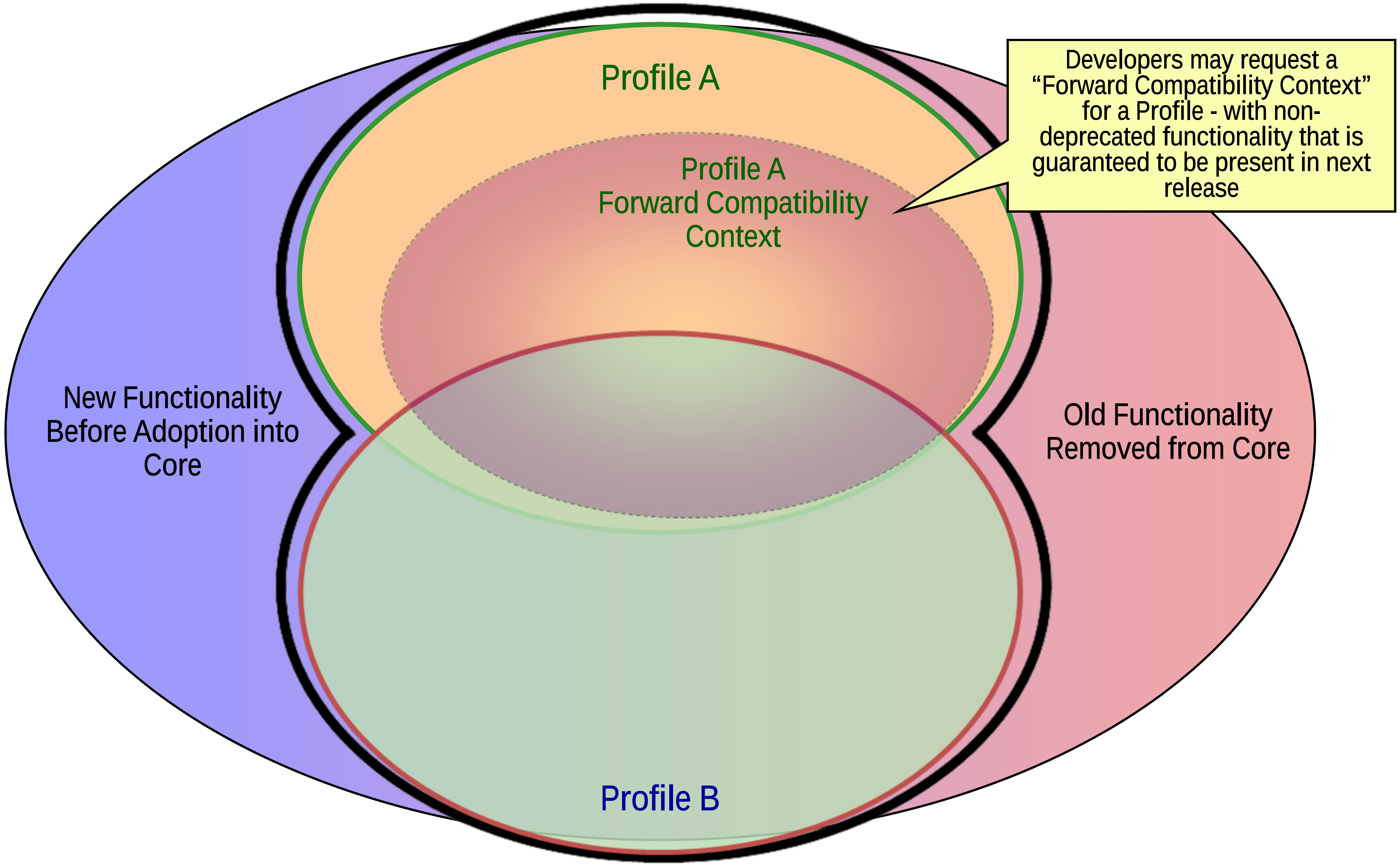
- **Full context**
  - Contains all features in a version of the core specification
- **Forward compatible context**
  - Contains only the non-deprecated functionality in a context and profile

Evolution Model – Forward Compatibility


Profile A
New Functionality Before Adoption into Core
Old Functionality Removed from Core
Profile B

Core Specification
The sum of all Profile functionality

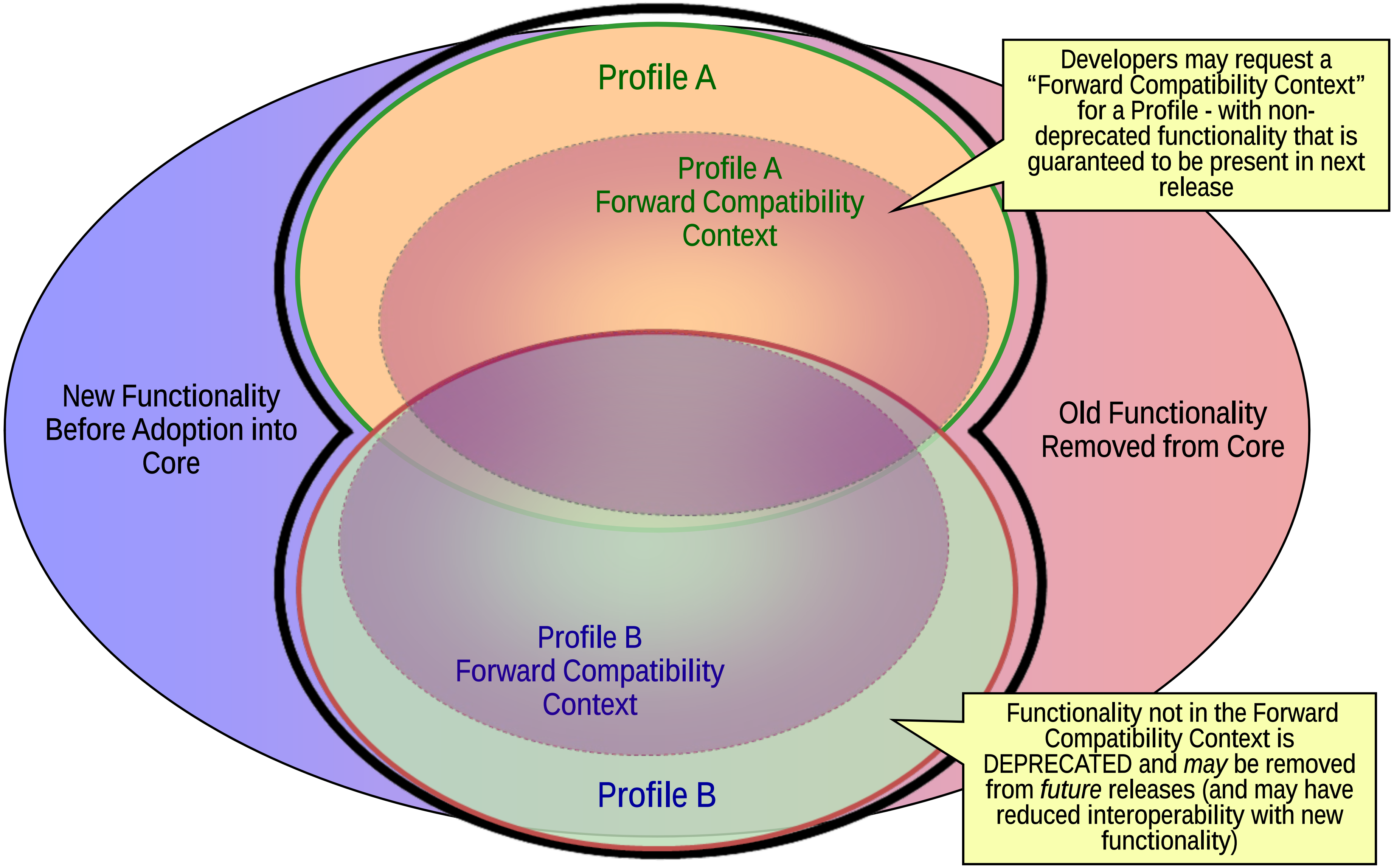
Evolution Model – Forward Compatibility


Profile A
Profile A Forward Compatibility Context
New Functionality Before Adoption into Core
Old Functionality Removed from Core
Profile B

Developers may request a "Forward Compatibility Context" for a Profile - with non-deprecated functionality that is guaranteed to be present in next release

Core Specification
The sum of all Profile functionality

Evolution Model - Forward Compatibility


Profile A
Profile A Forward Compatibility Context
New Functionality Before Adoption into Core
Old Functionality Removed from Core
Profile B Forward Compatibility Context
Profile B

Developers may request a "Forward Compatibility Context" for a Profile - with non-deprecated functionality that is guaranteed to be present in next release

Functionality not in the Forward Compatibility Context is DEPRECATED and *may* be removed from *future* releases (and may have reduced interoperability with new functionality)

Core Specification
The sum of all Profile functionality

Context creation

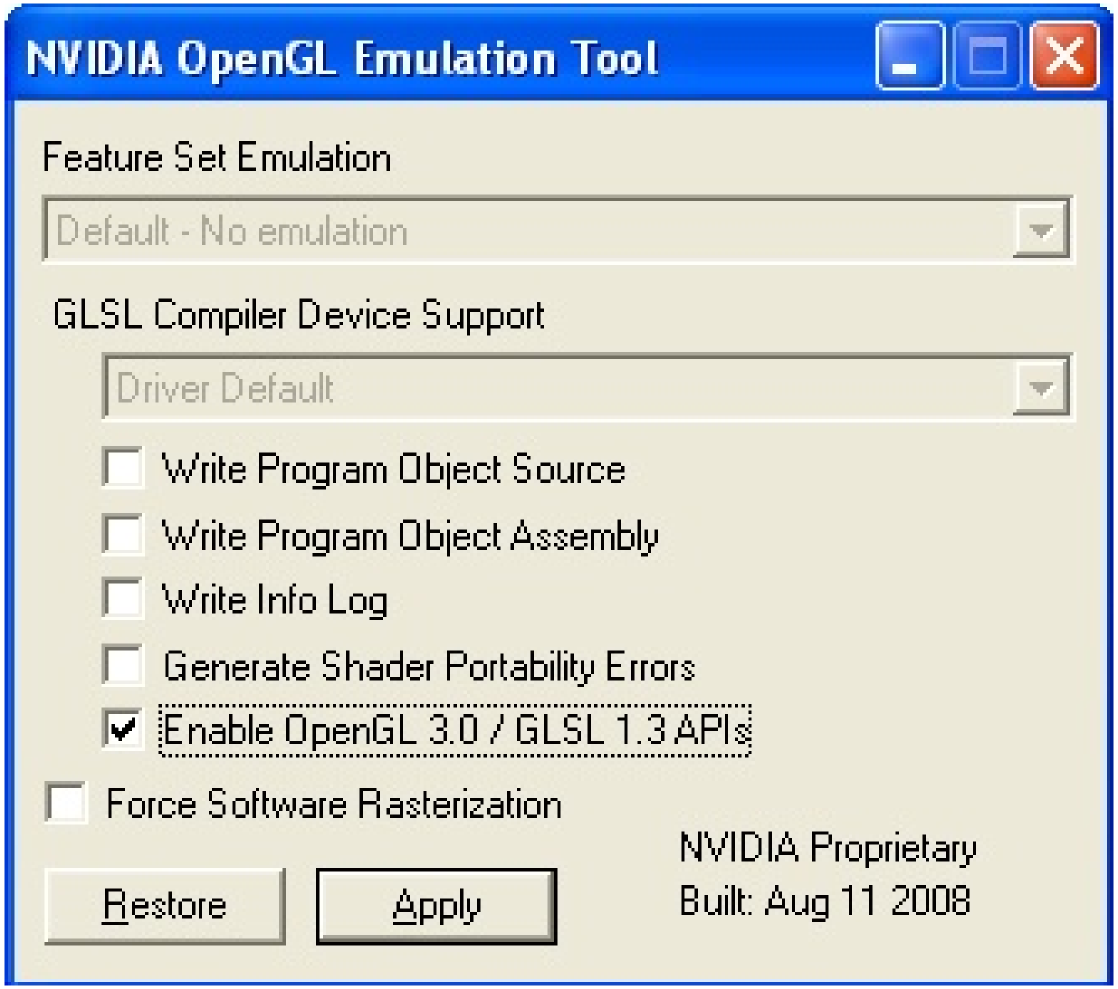
- **In the past creating a context gave you whatever version the driver decided**
  - No issue since the API was always backwards compatible,
- **Starting with OpenGL 3.1, backwards compatibility may no longer exist**
  - due to deprecation
  - Apps need a way to specify which functionality they require when creating a context
- **Existing context creation calls cannot return 3.0 or later contexts**
- **WGL/GLX_ARB_create_context**
  - To request specific context version, profile, forward compatible context or debug context.
  - `wgl/glxCreateContextAttribsARB()`

OpenGL 3.0 beta drivers

- **Beta drivers available for download now**
  - For Windows XP and Vista
  - Linux to follow shortly
  - G80 and higher GPUs supported. Geforce and Quadro
- **Beta drivers, aimed at developers to get started**
- **Supports full OpenGL 3.0 context**
- **Supports GLSL 1.30**
- **Also supporting most of the extensions**
- **See driver release notes for details**

developer.nvidia.com/object/opengl_3_driver.html

NVemulate


NVIDIA OpenGL Emulation Tool

Feature Set Emulation

Default - No emulation

GLSL Compiler Device Support

Driver Default

☐ Write Program Object Source

☐ Write Program Object Assembly

☐ Write Info Log

☐ Generate Shader Portability Errors

☑ Enable OpenGL 3.0 / GLSL 1.3 APIs

☐ Force Software Rasterization

NVIDIA Proprietary
Built: Aug 11 2008

- **developer.nvidia.com/object/nvemulate.html**

Future OpenGL plans

- **Schedule driven**
- **ARB extensions are candidates for folding into a future core**
  - ARB_draw_instanced
  - ARB_geometry_shader
  - ARB_texture_buffer_object
- **Backing uniform variables with buffer objects**
- **#include mechanism for GLSL**
- **Attribute index offsets**
- **Remove deprecated features**
- **Profiles**
- **Object model improvements**
- **Other functionality you need?**

Questions