

Data Storage and Transfer in OpenGL®

Kurt Akeley

NVIDIA Corporation

NVIDIA U Presentation, 25 July 2003

Outline

- **Data movement is the issue**
- **Memory types**
- **Design examples**
 - **Buffer objects**
 - **Superbuffers (in process)**
- **Programmable GPU memory**
- **Q & A**

Data Movement

Semiconductor Scaling Rates

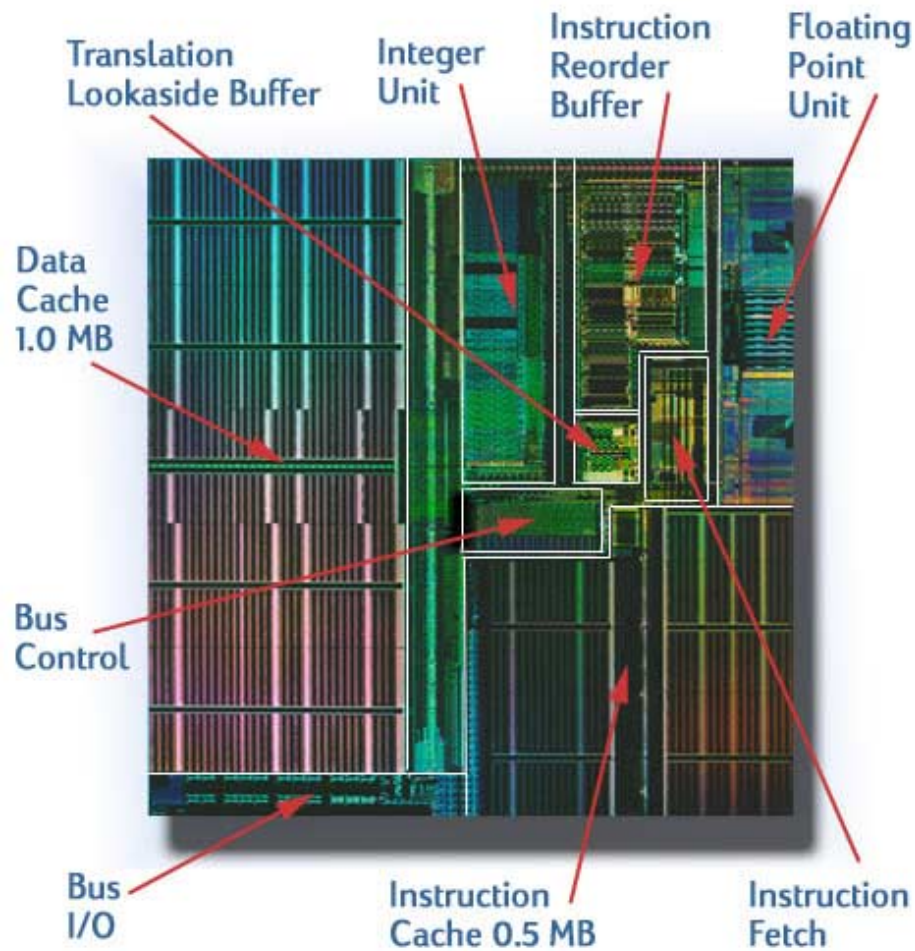
From: *Digital Systems Engineering*, Dally and Poulton

Parameter	Current Value	Yearly Factor	Years to Double (Half)
Moore's Law (grids on a die)**	1 B	1.49	1.75
Gate Delay	150 pS	0.87	(5)
Capability (grids / gate delay)		1.71	1.3
Device-length wire delay		1.00	
Die-length wire delay / gate delay		1.71	1.3
Pins per package	750	1.11	7
Aggregate off-chip bandwidth		1.28	3

Two Approaches

- **Move data faster (optimize speed)**
 - Point-to-point wiring
 - Advanced protocols (e.g. clock in data)
 - Wide interfaces (256-bit GPUs)
- **Move data less (optimize locality)**
 - Algorithm
 - Architecture (e.g. pipeline GPU)
 - Cache data

It's All Cache!



PA-8500 microprocessor

Memory Types

OpenGL's Memory Types

- **Byte memory**
 - Client pixel data
- **Element memory**
 - Framebuffer
 - Texture object's image
- **Others**
 - Display list memory
 - ...

Memory Properties

Property	Byte memory	Element memory
Basic unit	Byte	Component, index
Formatting	Per-transaction	Persistent
CPU mapping	May be allowed	Not allowed
Out-of-order ops	May be allowed	Not allowed
Typical location	Client side *	Server side

Client pixel data

Framebuffer

* We'll see an exception soon

Buffer Objects

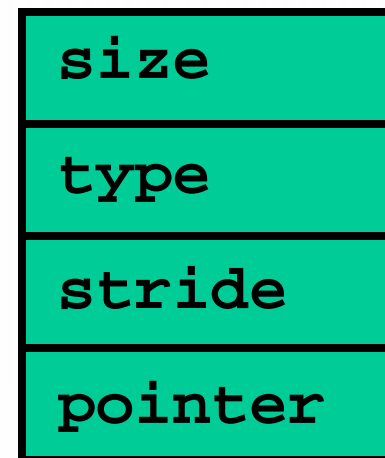
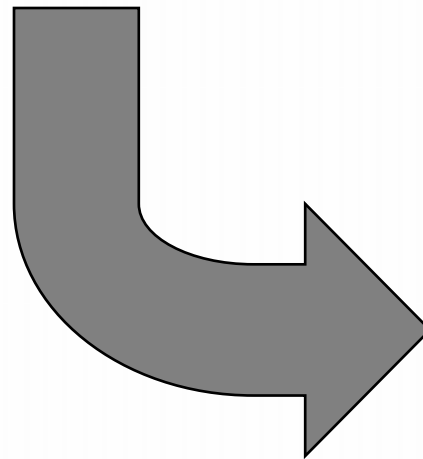
Classic Immediate Mode

- **Convenient**
- **Flexible**
 - Match client data structures
 - Per-vertex / per-primitive
- **Slow**
 - Vector form slightly faster

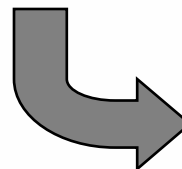
```
float loc[2] = {2, 3};  
glColor3f(1, 1, 1);  
glBegin(GL_LINES);  
glVertex2f(0, 1);  
glVertex2fv(loc);  
glEnd();
```

Vertex Arrays

```
glVertexPointer( size, type, stride, *pointer );
```



```
glEnableClientState( GL_VERTEX_ARRAY );
```



```
glDrawArrays( mode, first, count );
```

Vertex Array Problems

- **Client-side array is far from pipeline**
- **Hard for application to use the *right* memory**
 - **On PC, AGP memory**

Solutions

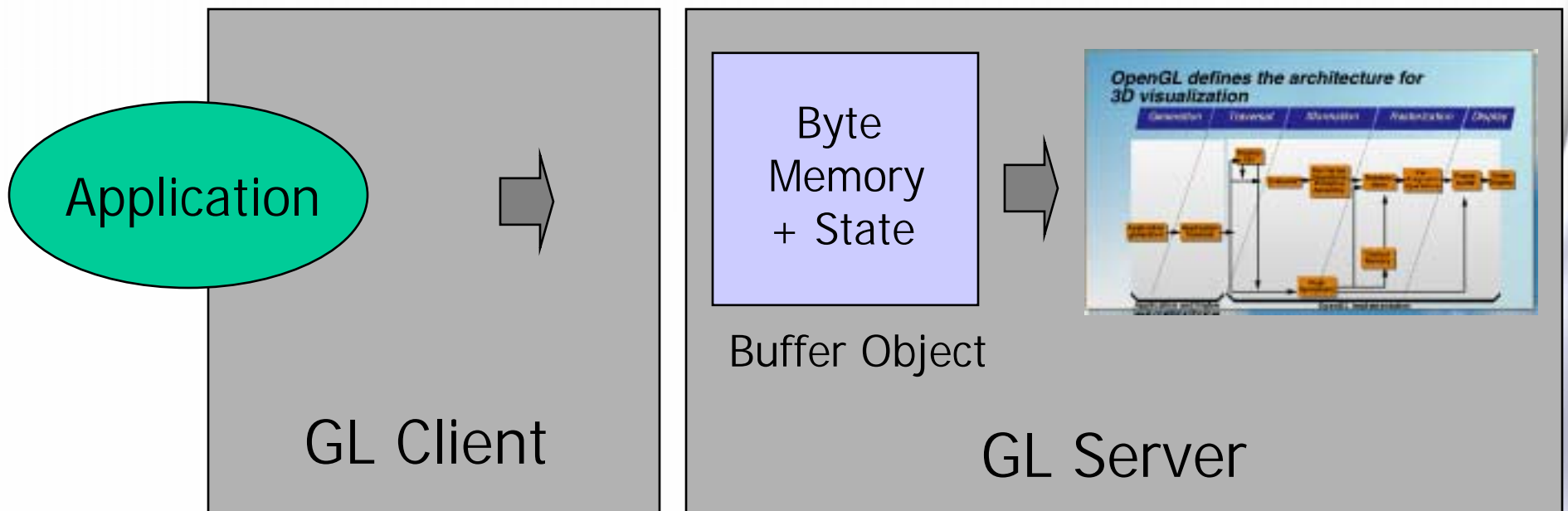
- **NV_vertex_array_range**
 - **Make vertex arrays work better**
 - **Relax coherence rules to enable pulling**
 - **Help client to position arrays in AGP memory**
- **Array objects**
 - **Extend vertex arrays with element memory**

Good Qualities of Vertex Arrays

- **Easy to mix and match array data**
 - Share vertex array with multiple color arrays
 - Disable any array for debugging
- **Memory mapped**
 - Client can cherry-pick updates
 - Extra copies avoided
 - Great for immediate mode rendering
- **Easy to mix usage models**
 - Static array data
 - Dynamic array data

ARB Solution: Server-side Buffer Objects

- Byte memory (mappable)
- Server side (efficient for rendering, sharable)



Modifying Buffer Object Data

- **Functional**

- `glBufferSubDataARB(target, offset, size, *data) ;`

- `glGetBufferSubDataARB(target, offset, size, *data) ;`

- **Always safe**

- **Memory mapped**

- `void *glMapBufferARB(target, access) ;`

- `GLboolean *glUnmapBufferARB(target) ;`

- **May result in data loss**

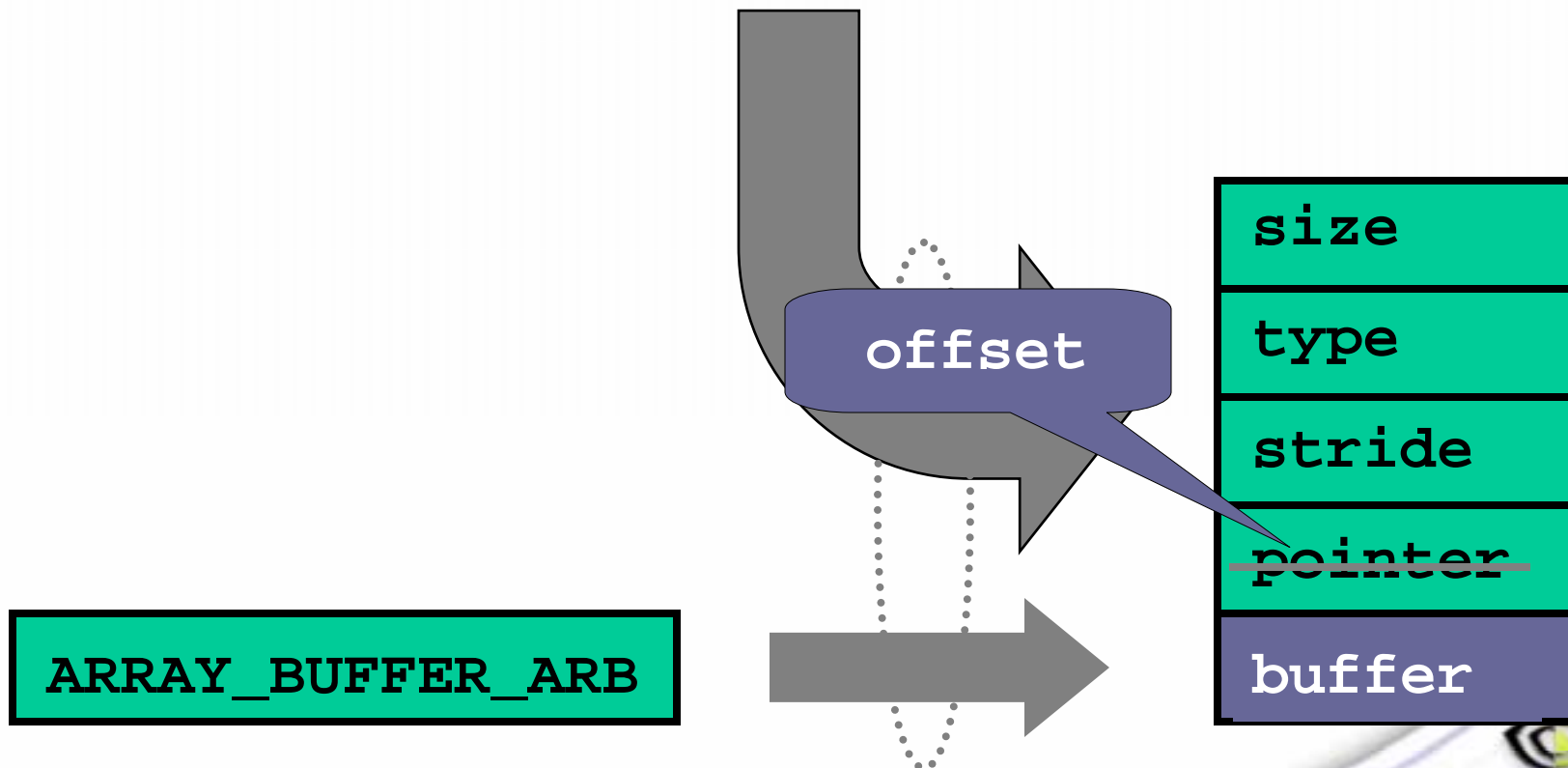
- **May not be faster**

Mapping Rules

- **Be prepared for data loss**
- **Map for brief periods only**
 - **Map it, modify it, then unmap it**
 - **Don't render from a mapped buffer**
- **Don't pass a map pointer to the GL**

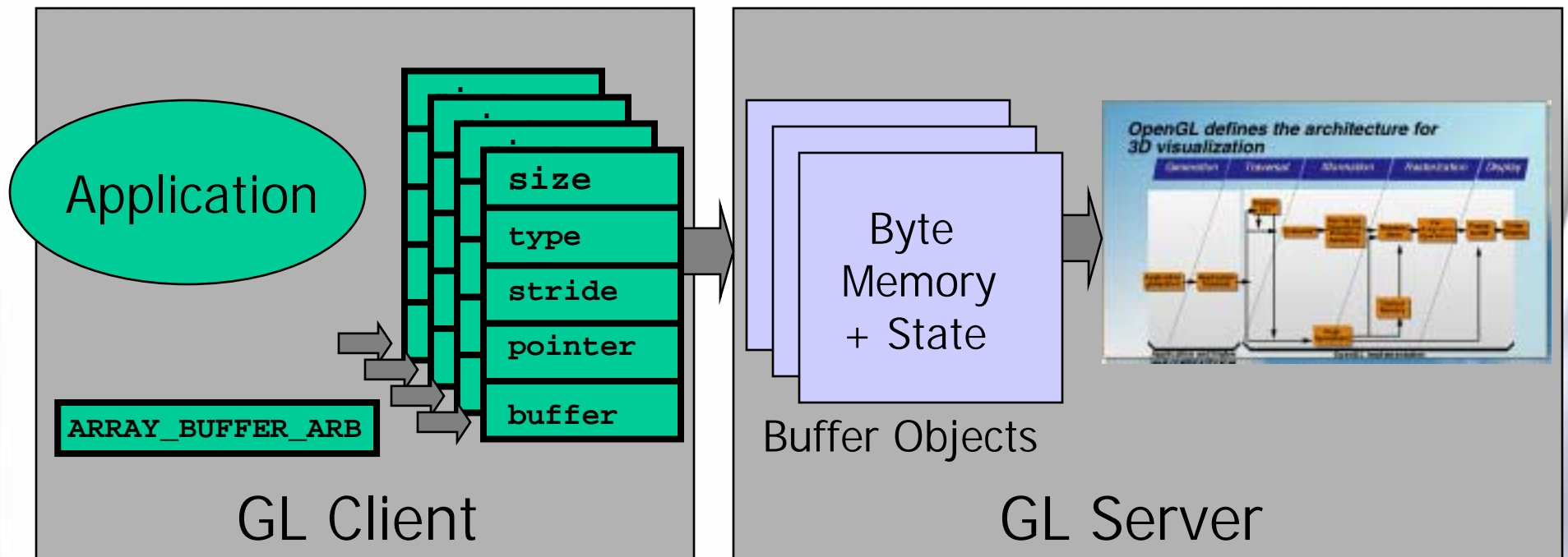
Store Vertex Arrays in Buffers

```
VertexPointer( size, type, stride, *pointer );
```



Client and Server State

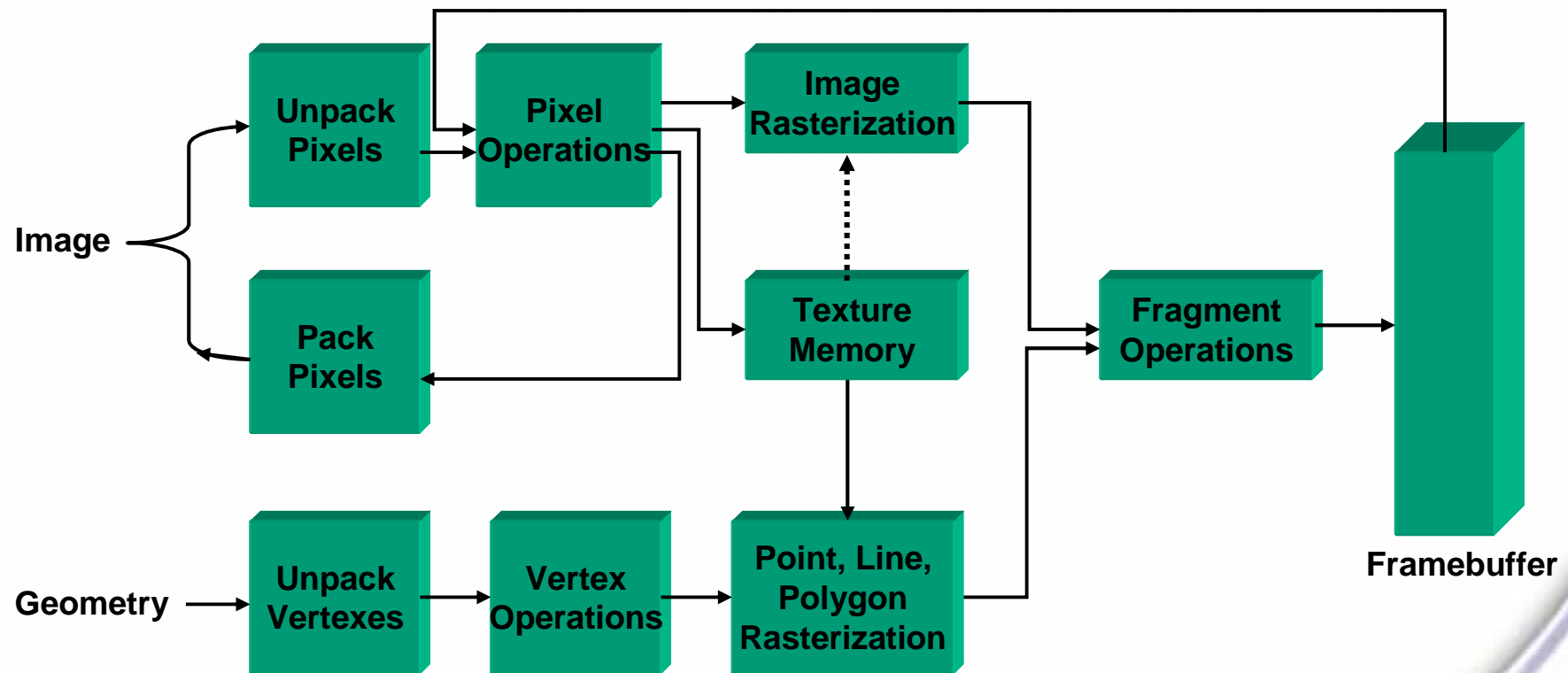
- Buffer objects are server state
- Vertex arrays parameters are client state



Superbuffers

(In Process)

The OpenGL “Machine Tool”



Problems

- **Framebuffer to texture transfer**
 - **Copy is too slow**
 - **Current acceleration methods (pbuffers) are**
 - **Window-system specific**
 - **Awkward to use**
- **Framebuffer to geometry transfer**
 - **Copy is too slow**
 - **There are no acceleration methods**

~~○ **Render to texture**~~

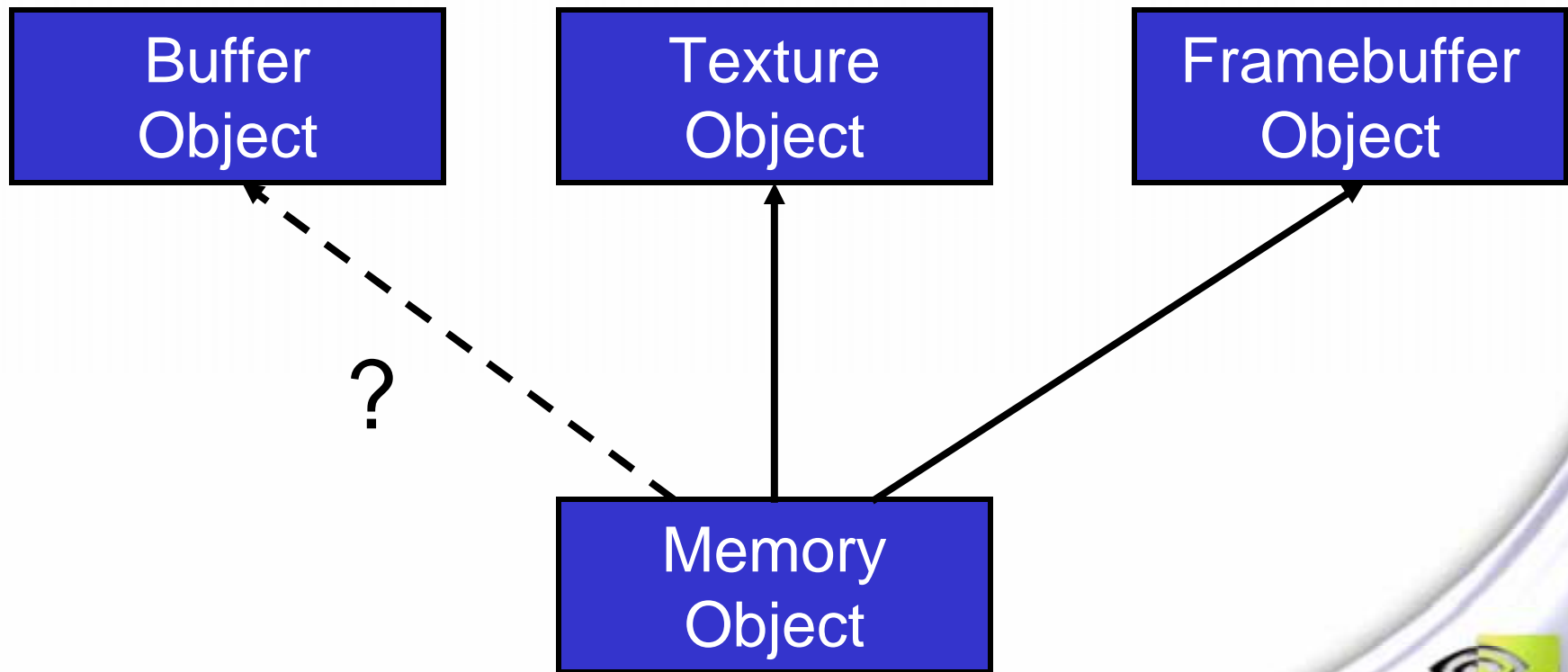
Why Not Add Destructive Copy?

- **Virtue of simplicity**
- **Issues**
 - **Can't be anticipated**
 - **Forces 2x memory when not otherwise needed**

Superbuffer Solution

- **Invent framebuffer objects**
 - Much like texture objects
- **Invent element memory objects**
 - 1D, 2D, or 3D images
- **Attach element memory objects**
 - To texture objects, as image levels
 - To framebuffer objects, as color, depth, and stencil buffers

Superbuffers



**Can element memory
objects be attached to
buffer objects?**

GPU Memory

So Far So Good

- **Byte / element distinction makes sense**
- **Texture is element memory**
 - **Always accessed through functional interface**
- **Framebuffer is element memory**
 - **Rendered through functional interface**
 - **Cannot be read by vertex/fragment processor**
- **Language arrays are “byte” memory**

Summary

- **NVIDIA is committed to OpenGL**
- **OpenGL is evolving**
 - **Buffer objects will be in 1.5, due this summer**
 - **Superbuffers are underway, no target date**
- **Byte / element memory distinction has been useful**

For More Information

- www.opengl.org
- <http://developer.nvidia.com/view.asp?IO=presentations>