GPU Memory Model Overview

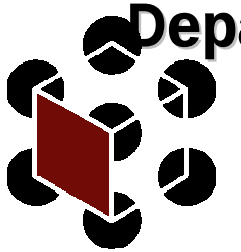
John Owens

University of California, Davis

Department of Electrical and Computer Engineering

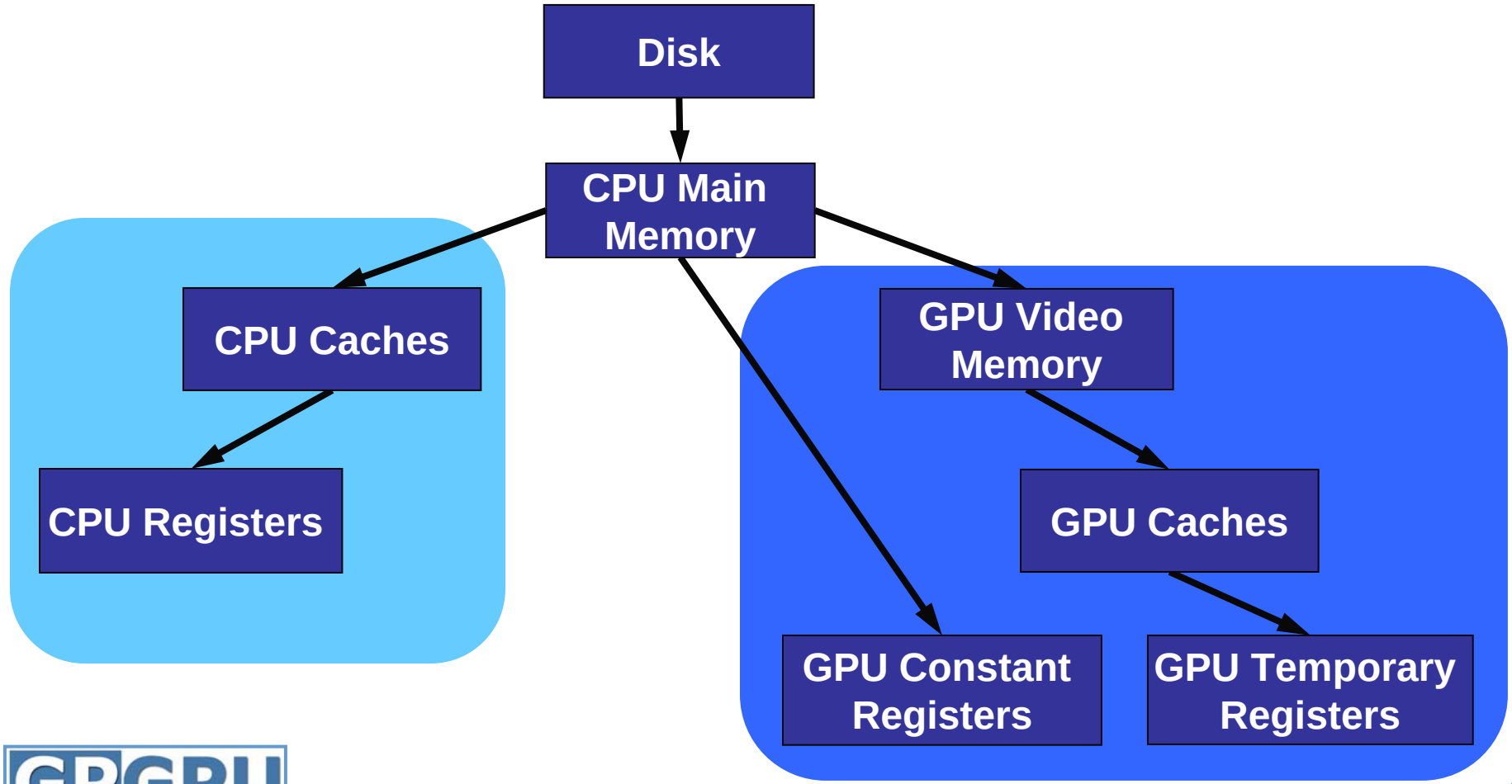
Institute for Data Analysis and Visualization

SciDAC Institute for Ultrascale Visualization



Memory Hierarchy

- CPU and GPU Memory Hierarchy

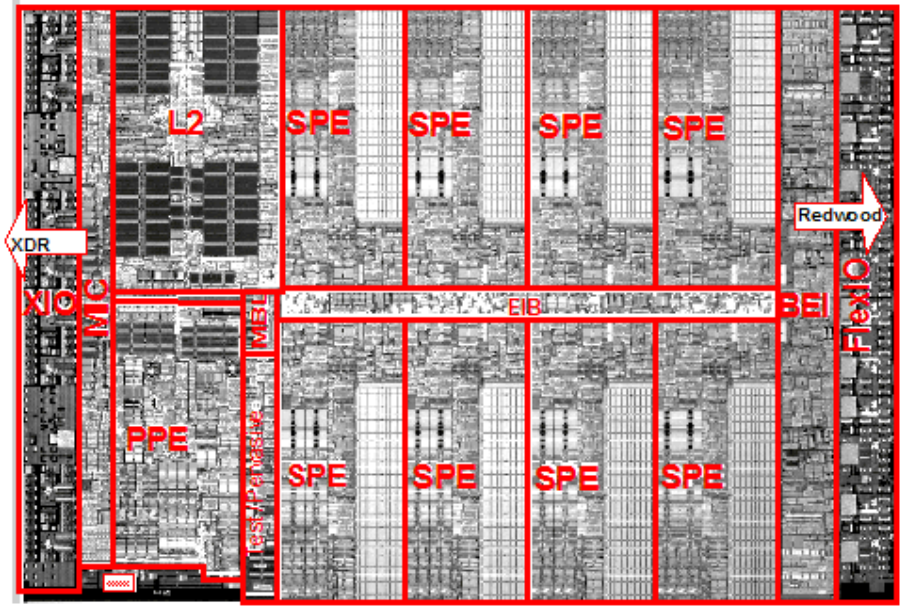


CPU Memory Model

- **At any program point**
  - Allocate/free local or global memory
  - Random memory access
    - Registers
      - Read/write
    - Local memory
      - Read/write to stack
    - Global memory
      - Read/write to heap
    - Disk
      - Read/write to disk

Cell

- **SPU memory model:**
- **128 128b local registers**
- **256 kB local store**
  - 6 cycles access time
- **Explicit, asynchronous DMA access to main memory**
  - Allows comm/comp overlap
- **No explicit I or D cache**
- **No disk access**



http://www.realworldtech.com/includes/images/articles/cell-1.gif

GPU Memory Model

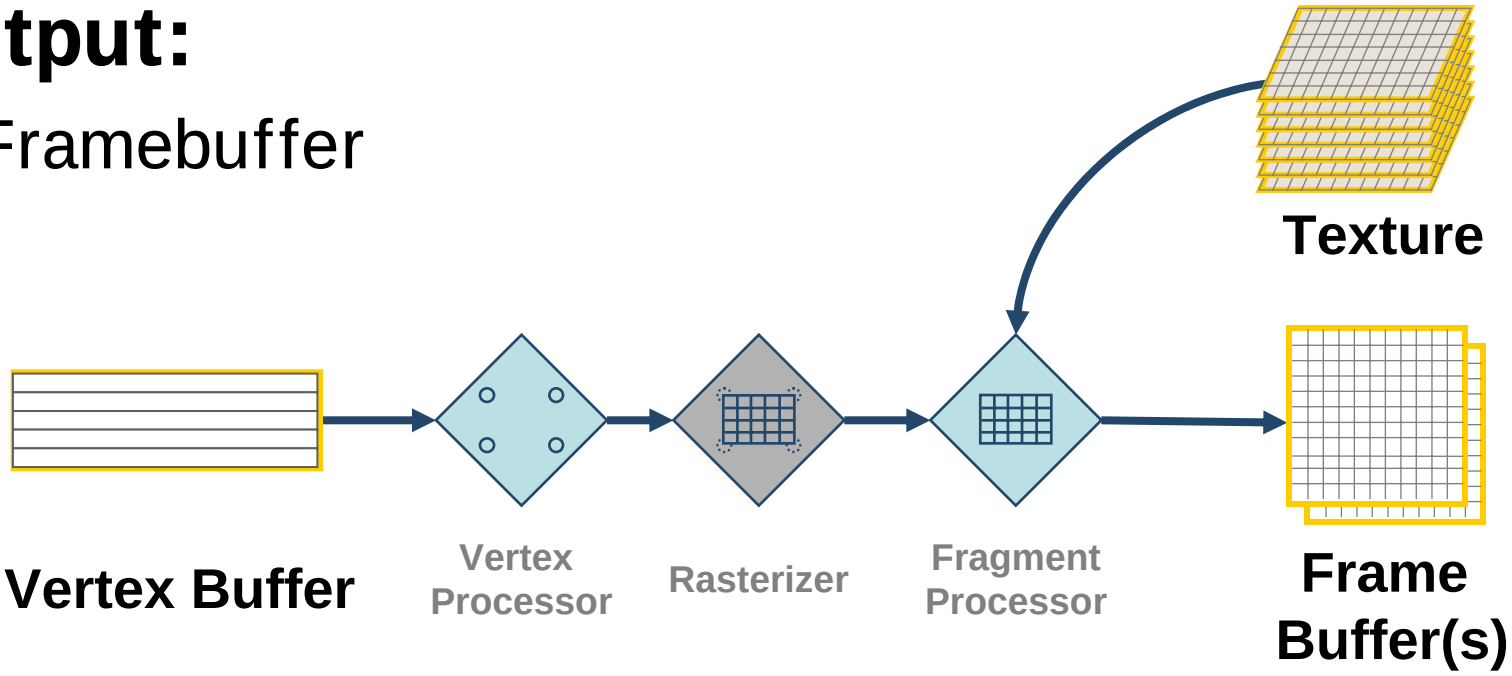
- **Much more restricted memory access**
  - Allocate/free memory only before computation
  - Limited memory access during computation (kernel)
    - Registers
      - Read/write
    - Local memory
      - Does not exist
    - Global memory
      - Read-only during computation
      - Write-only at end of computation (precomputed address)
    - Disk access
      - Does not exist

GPU Memory Model

- **GPUs support many types of memory objects in hardware**
  - 1D, 2D, 3D grids
    - 2D is most common (framebuffer, texture)
  - 2D cube maps (6 faces of a cube)
  - Mipmapped (prefiltered) versions
  - DX10 adds arrayed datatypes
- **Each native datatype has pros and cons from a general-purpose programming perspective**

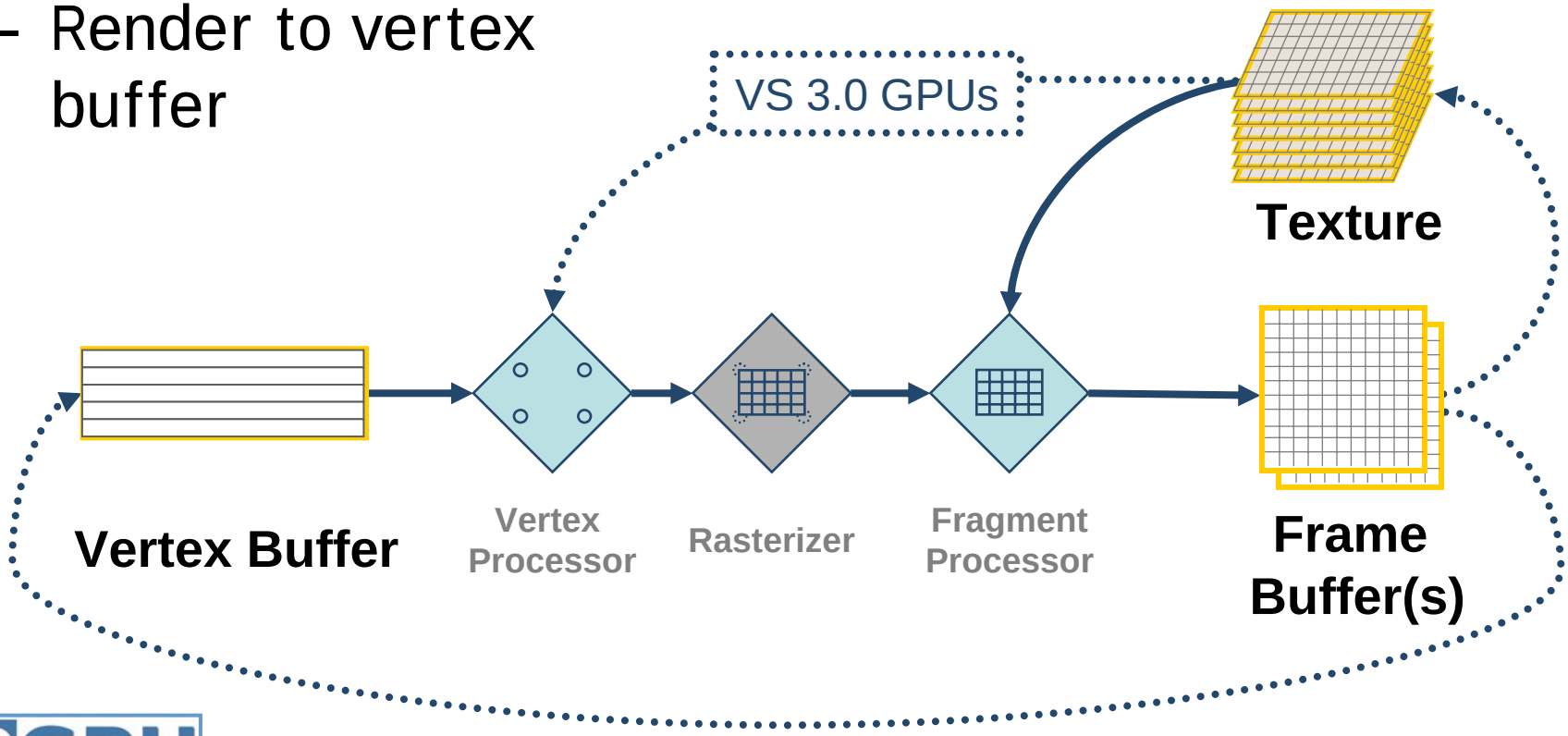
Traditional GPU Pipeline

- **Inputs:**
  - Vertex data
  - Texture data
- **Output:**
  - Framebuffer



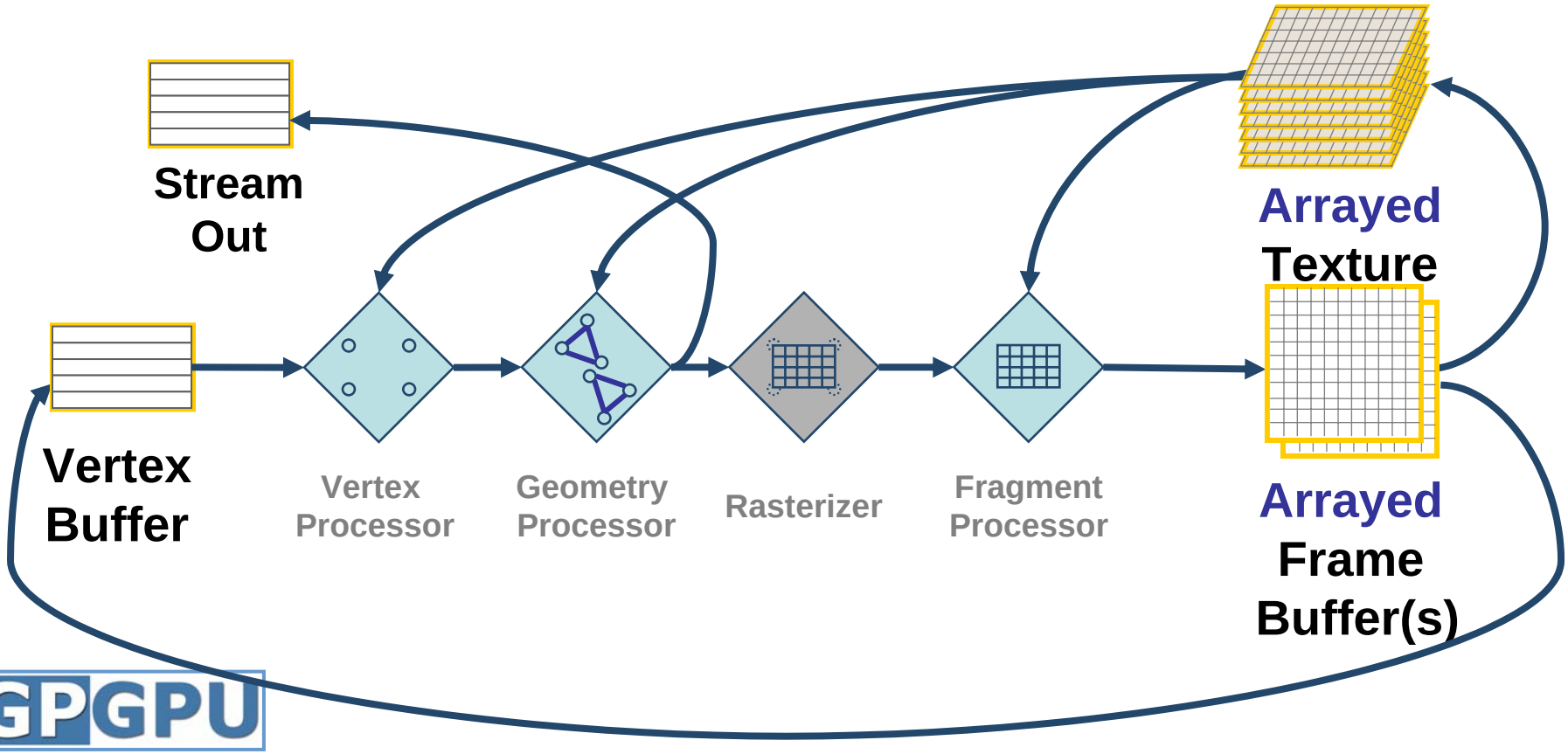
GPU Memory Model (DX9)

- **Extending memory functionality**
  - Copy from framebuffer to texture
  - Texture reads from vertex processor
  - Render to vertex buffer



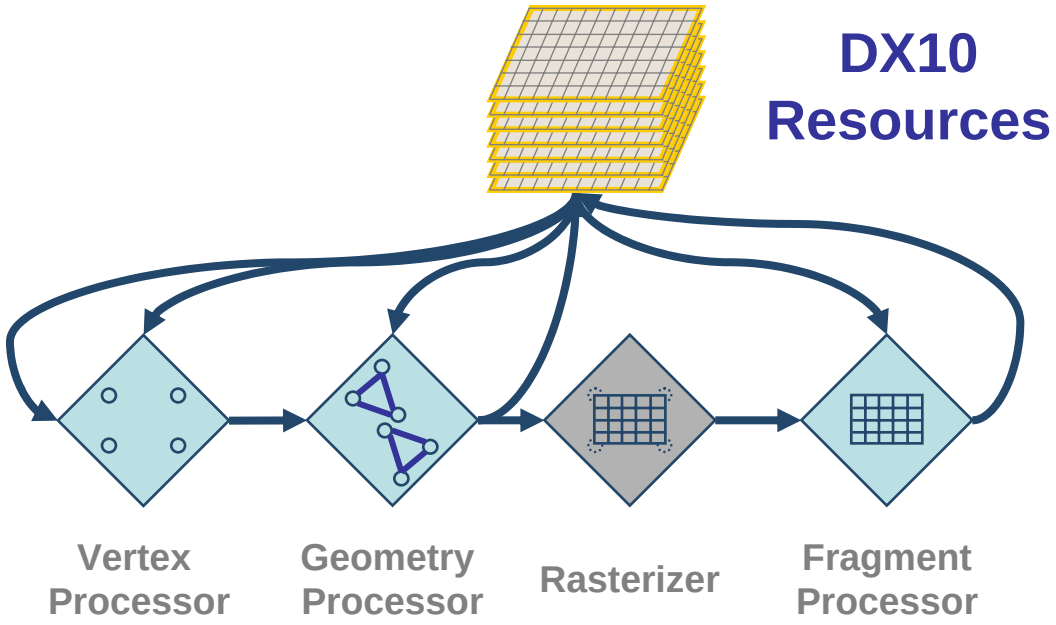
GPU Memory Model (DX10, traditional)

- **More flexible memory handling**
  - All programmable units can read texture
  - "Stream out" after geometry processor



GPU Memory Model (DX10, new)

- DX10 provides “resources”
- Resources are flexible!



GPU Memory API

- **Each GPU memory type supports subset of the following operations**
  - CPU interface
  - GPU interface

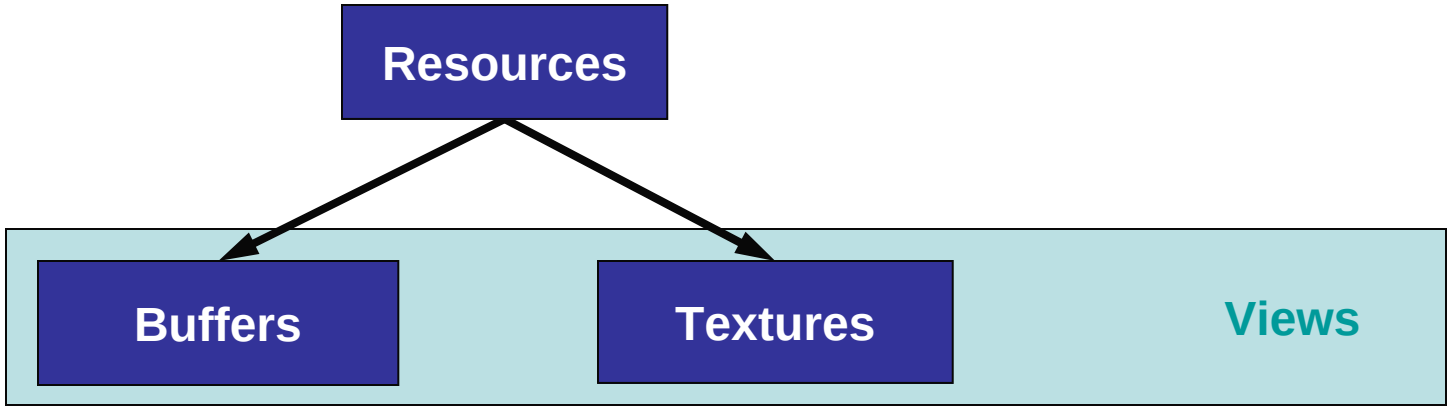
GPU Memory API

- **CPU interface**
  - Allocate
  - Free
  - Copy CPU → GPU
  - Copy GPU → CPU
  - Copy GPU → GPU
  - Bind for read-only vertex stream access
  - Bind for read-only random access
  - Bind for write-only framebuffer access

GPU Memory API

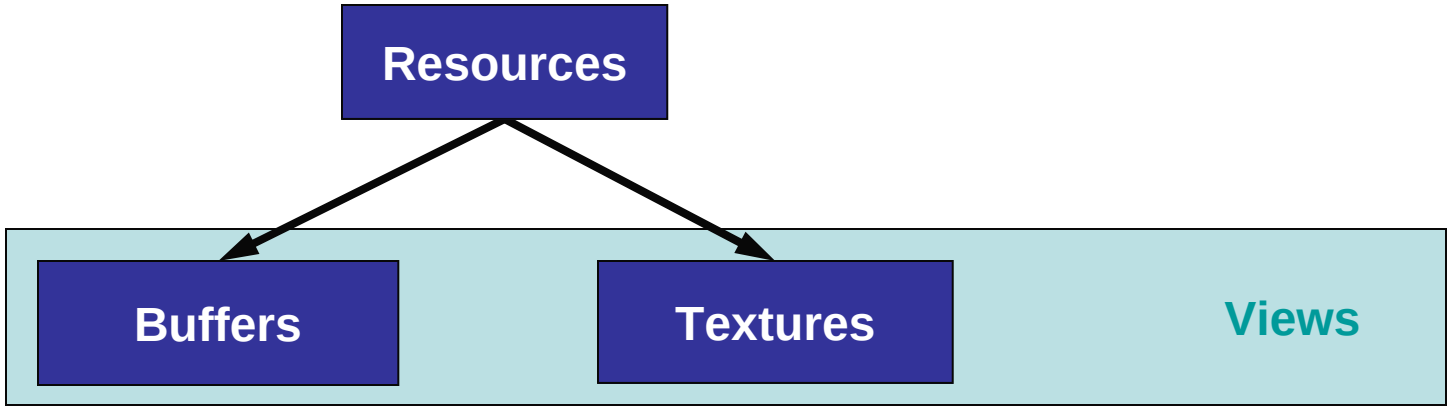
- **GPU (shader/kernel) interface**
  - Random-access read
  - Stream read

DX10 View of Memory



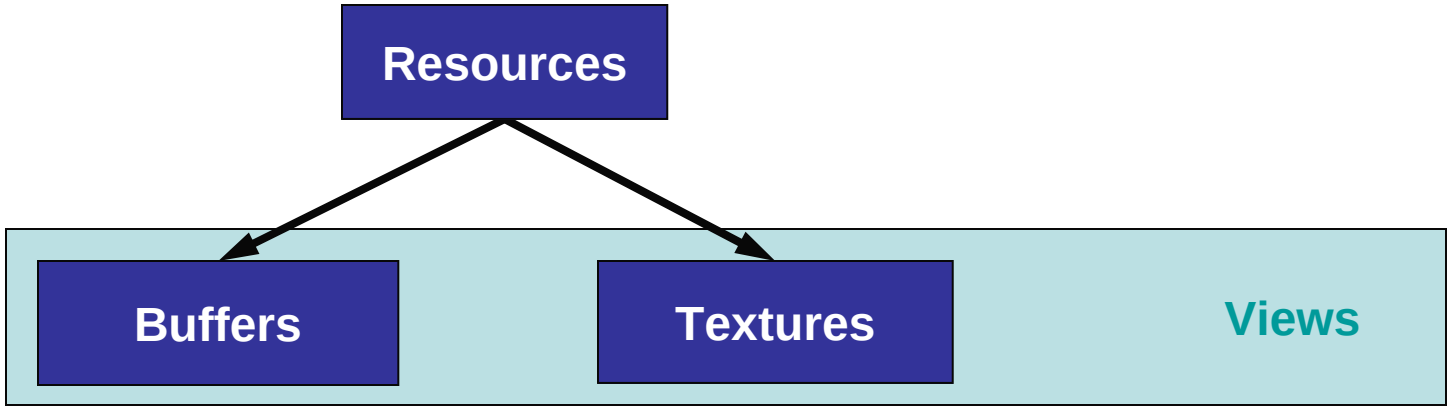
- **Resources**
  - Encompass buffers and textures
  - Retained state is stored in resources
  - Must be bound by API to pipeline stages before called
    - Same subresource cannot be bound for both read and write simultaneously

DX10 View of Memory



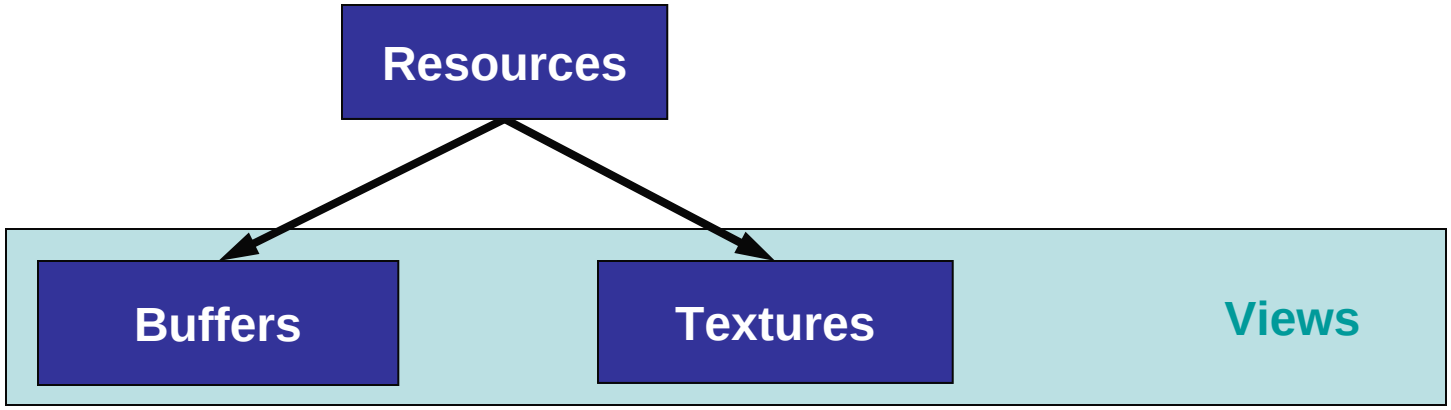
- **Buffers**
  - Collection of *elements*
    - Few requirements on type or format (heterogeneous)
    - Elements are 1-4 components (e.g. R8G8B8A8, 8b int, 4x32b float)
  - No filtering, subresourcing, multisampling
  - Layout effectively linear (“casting” is possible)
  - Examples: vertex buffers, index buffers, ConstantBuffers

DX10 View of Memory



- **Textures**
  - Collection of *texels*
  - Can be filtered, subresourced, arrayed, mipmapped
  - Unlike buffers, must be declared with texel type
    - Type impacts filtering
  - Layout is opaque - enables memory layout optimization
  - Examples: texture{1,2,3}d, mipmapped, cubemap

DX10 View of Memory

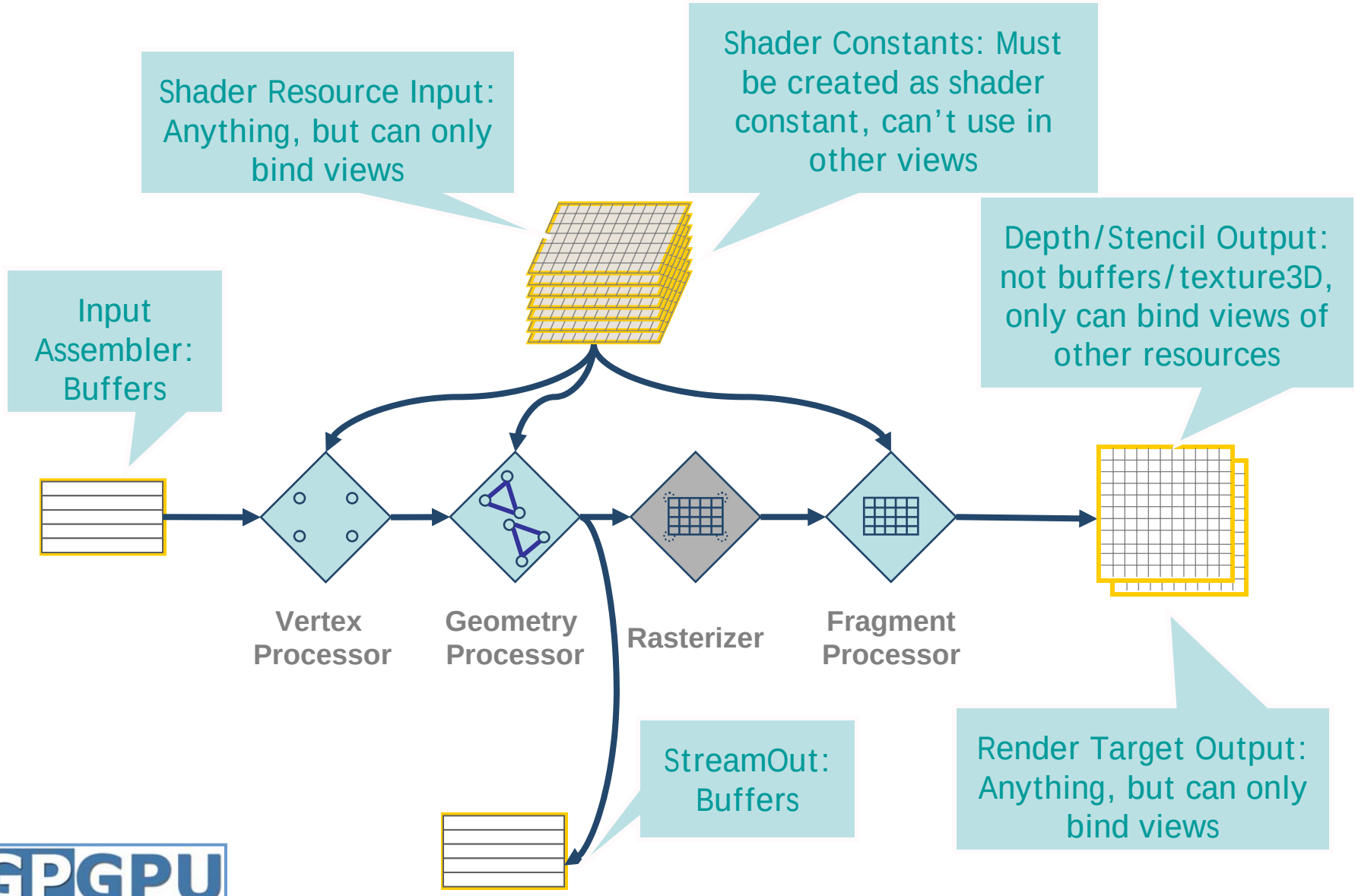


- **Views**
  - "mechanism for hardware interpretation of a resource in memory"
  - Allows structured access of subresources
  - Restricting view may increase efficiency

Big Picture: GPU Memory Model

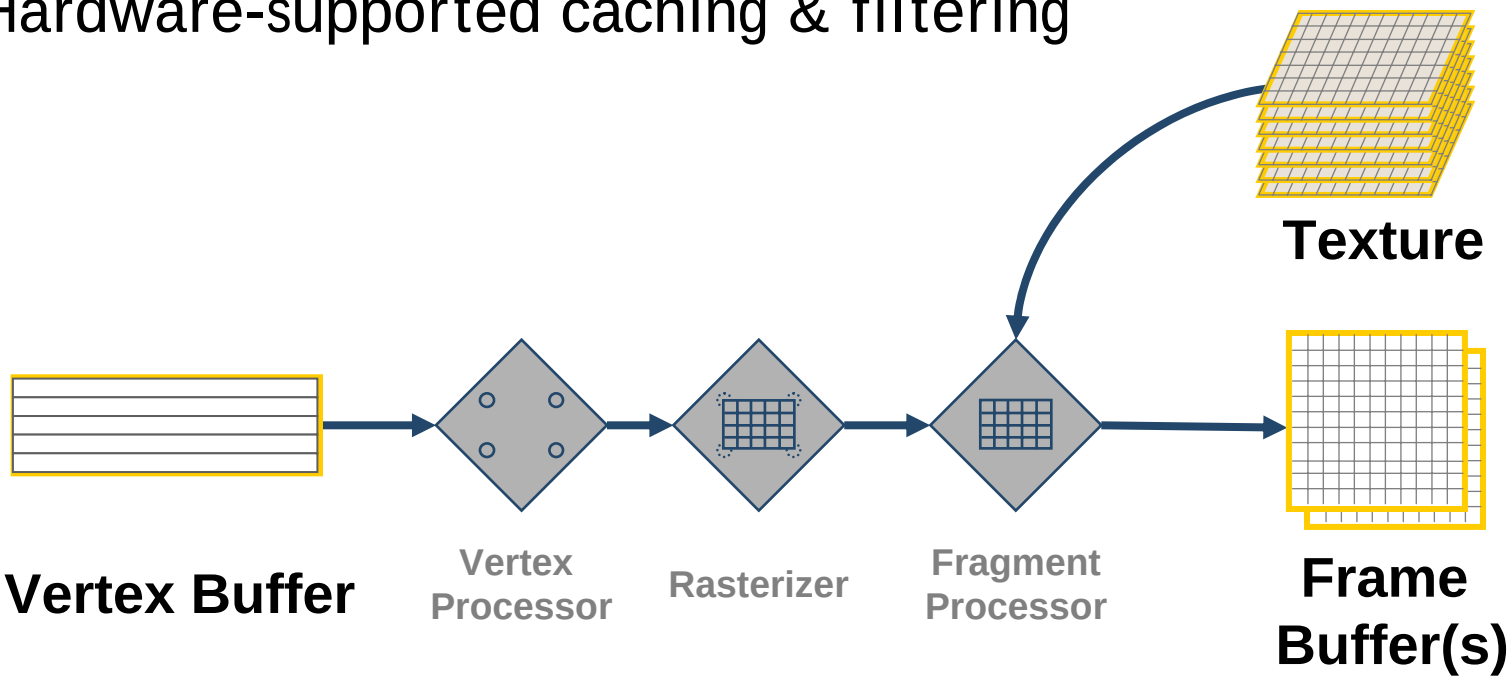
- **GPUs are a mix of:**
  - Historical, fixed-function capabilities
  - Newer, flexible, programmable capabilities
- **Fixed-function:**
  - Known access patterns, behaviors
  - Accelerated by special-purpose hardware
- **Programmable:**
  - Unknown access patterns
  - Generality good
- **Memory model must account for both**
  - Consequence: Ample special-purpose functionality
  - Consequence: Restricting flexibility may improve performance

DX10 Bind Rules



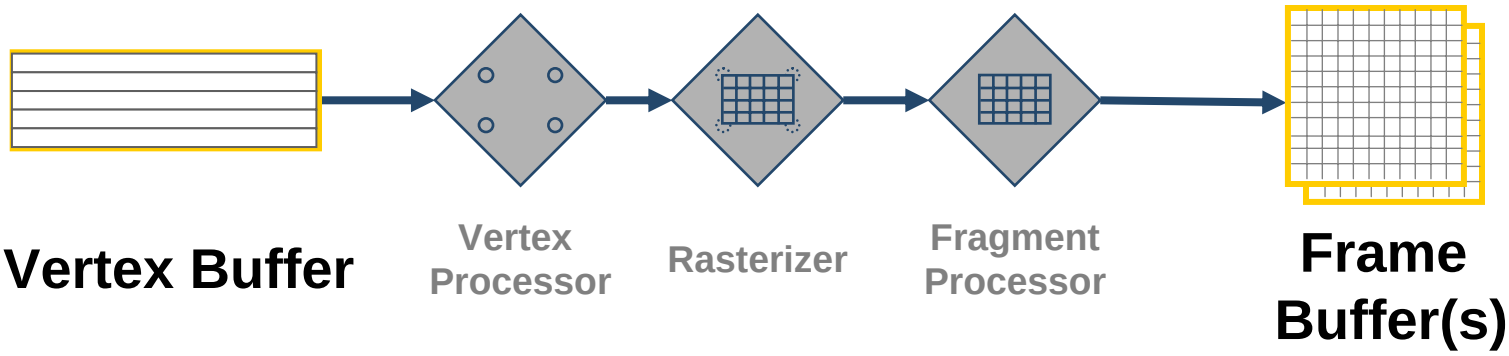
Example: Texture

- **Texture mapping fundamental primitive in GPUs**
- **Most typical use: random access, bound for read only, 2D texture map**
  - Hardware-supported caching & filtering



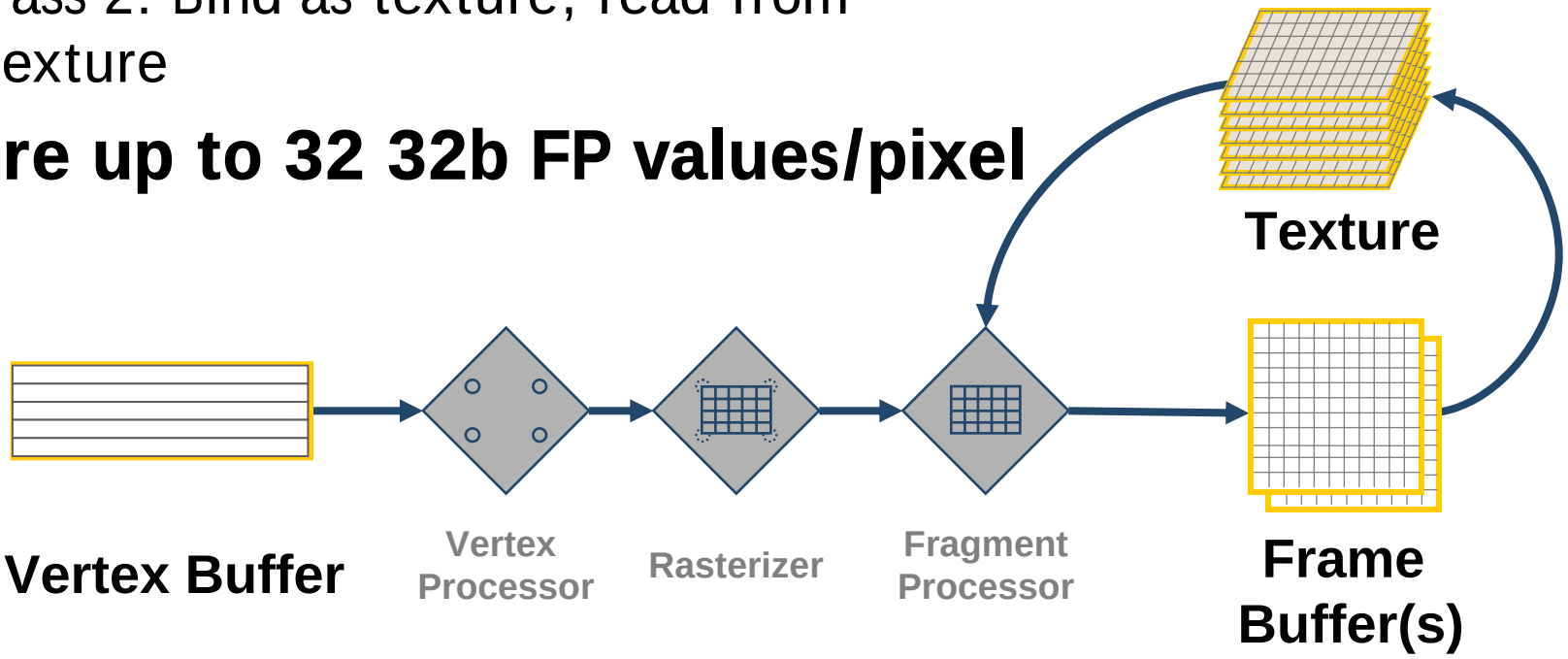
Example: Framebuffer

- **Memory written by fragment processor**
- **Write-only GPU memory (from shader's point of view)**
  - FB is read-modify-write by the pipeline as a whole
- **Displayed to screen**
- **Can also store GPGPU results (not just color)**



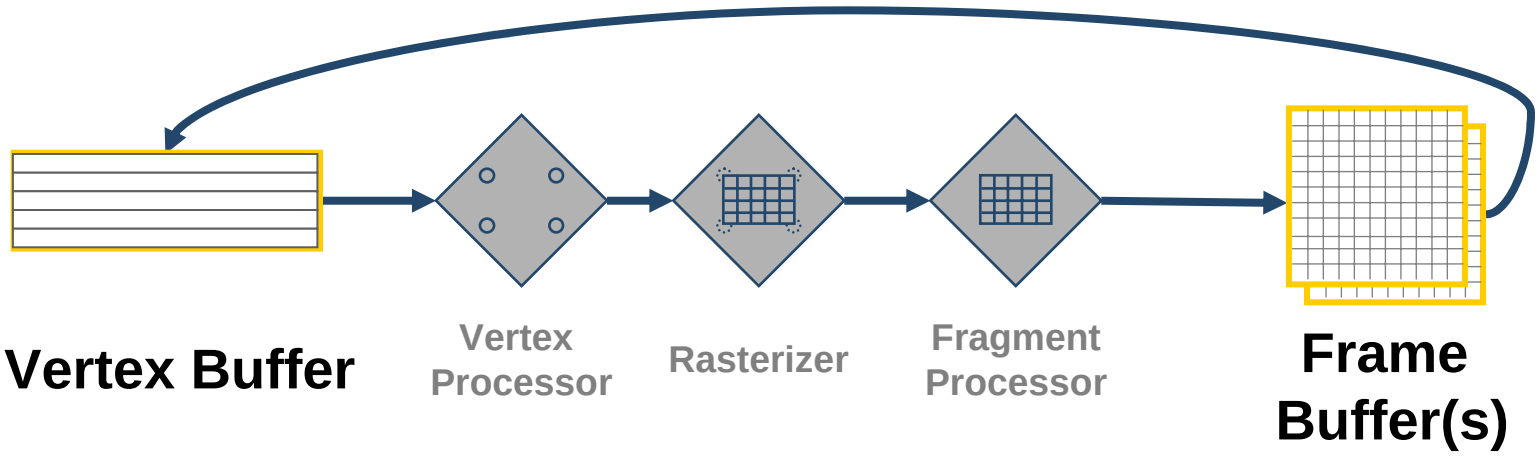
Example: Render to Texture

- **Very common in both graphics & GPGPU**
- **Allows multipass algorithms**
  - Pass 1: Write data into framebuffer
  - Pass 2: Bind as texture, read from texture
- **Store up to 32 32b FP values/pixel**



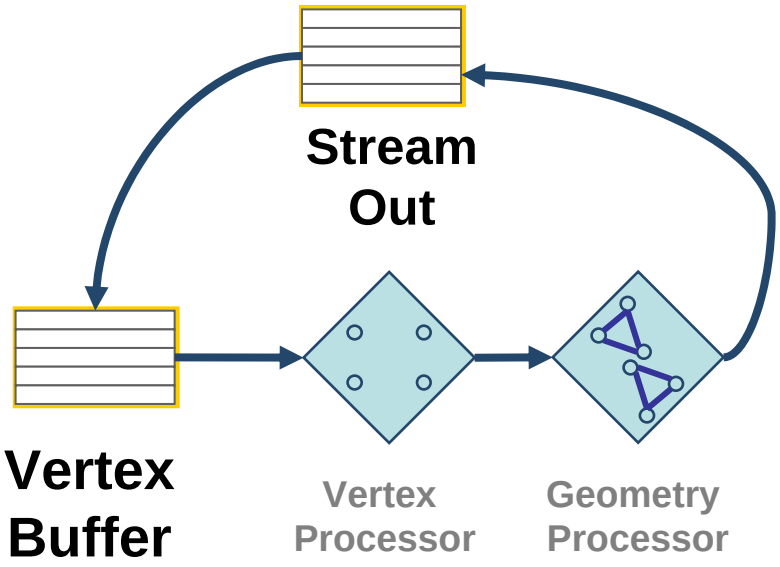
Example: Render to Vertex Array

- Enables top-of-pipe feedback loop
- Enables dynamic creation of geometry on GPU



Example: Stream Out to Vertex Buffer

- **Enabled by DX10 StreamOut capability**
- **Expected to be used for dynamic geometry**
  - Recall geometry processor produces 0-n outputs per input
- **Possible graphics applications:**
  - Expand point sprites
  - Extrude silhouettes
  - Extrude prisms/tets



Summary

- **Rich set of hardware primitives**
  - Designed for special purpose tasks, but often useful for general purpose ones
- **Memory usage generally more restrictive than other processors**
  - Becoming more general-purpose and orthogonal
- **Restricting generality allows hw/sw to cooperate for higher performance**