GPU TECHNOLOGY CONFERENCE

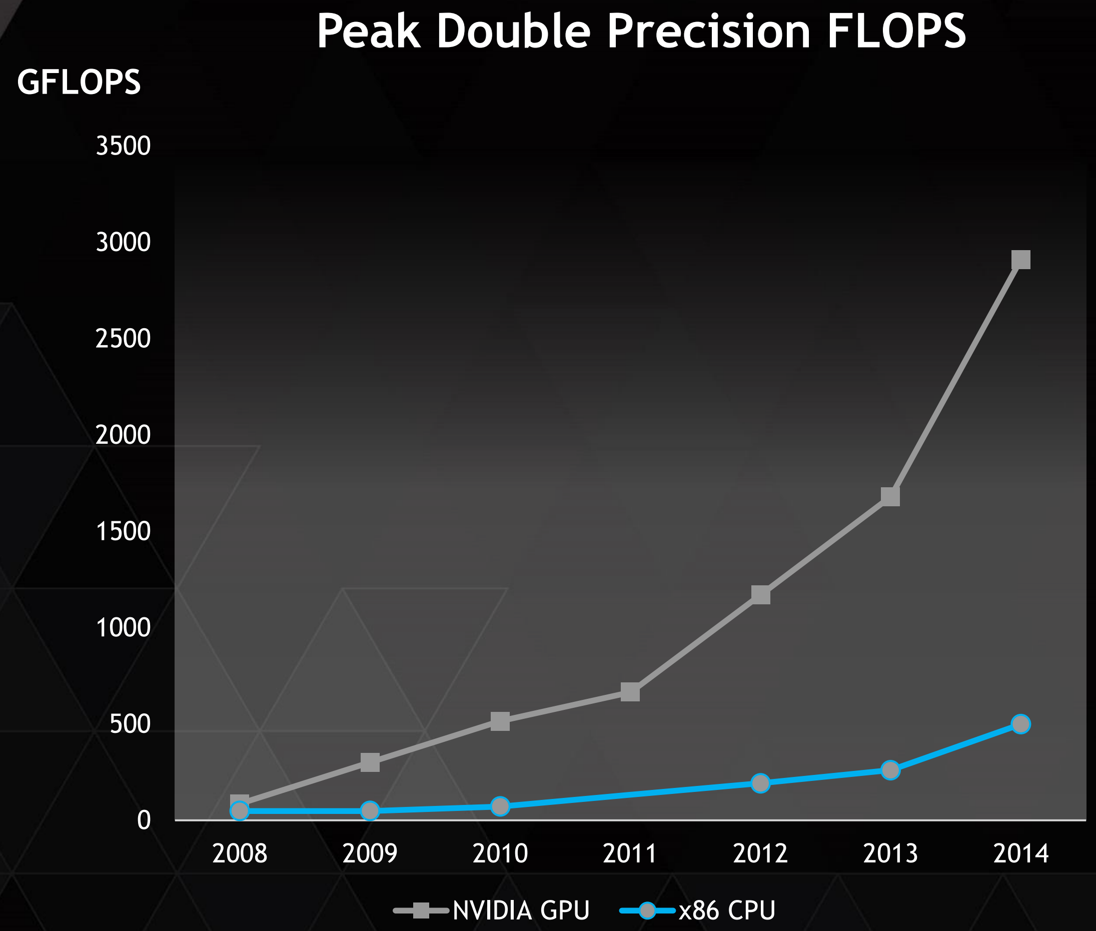
NVPRO-PIPELINE
A RESEARCH RENDERING PIPELINE

MARKUS TAVENRATH – MATAVENRATH@NVIDIA.COM

SENIOR DEVELOPER TECHNOLOGY ENGINEER, NVIDIA

GPU TECHNOLOGY CONFERENCE

NVPRO-PIPELINE

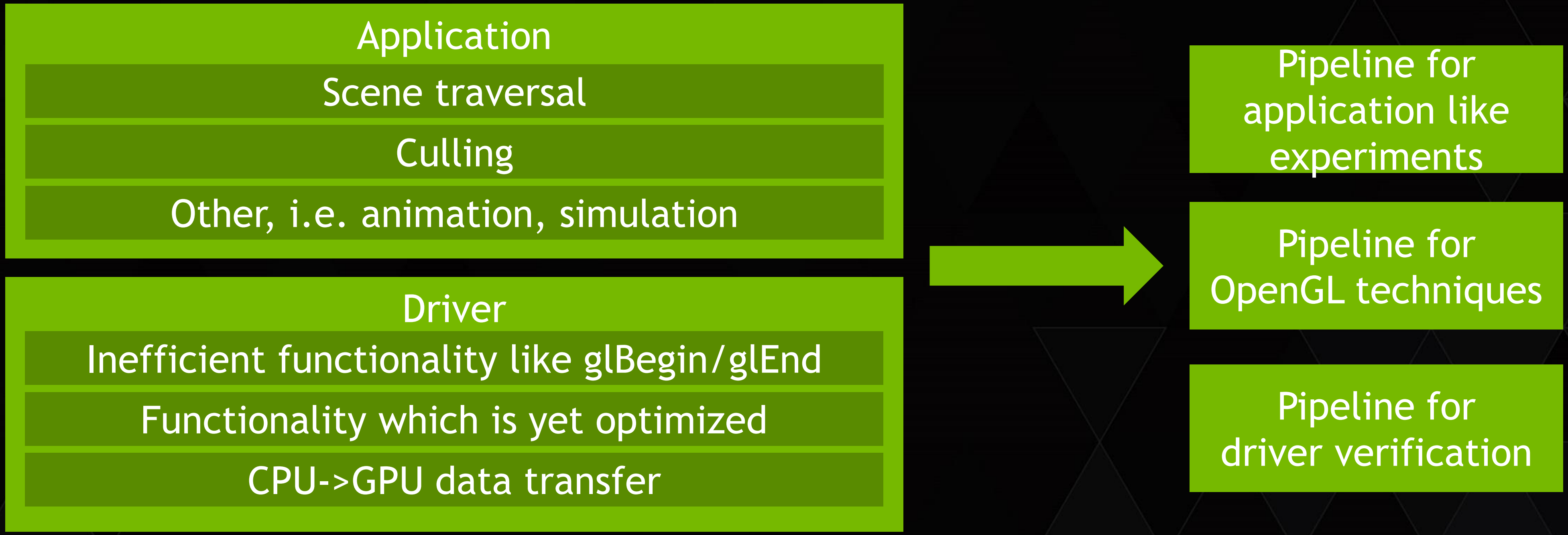


- GPU perf improved better than CPU perf
- In the past apps were GPU bound
- Today apps tend to become CPU bound

- nvpro-pipeline started as research platform to address this issue

- http://github.com/nvpro-pipeline

CPU BOUNDEDNESS REASONS



Application

Scene traversal

Culling

Other, i.e. animation, simulation

Driver

Inefficient functionality like glBegin/glEnd

Functionality which is yet optimized

CPU->GPU data transfer

Pipeline for application like experiments

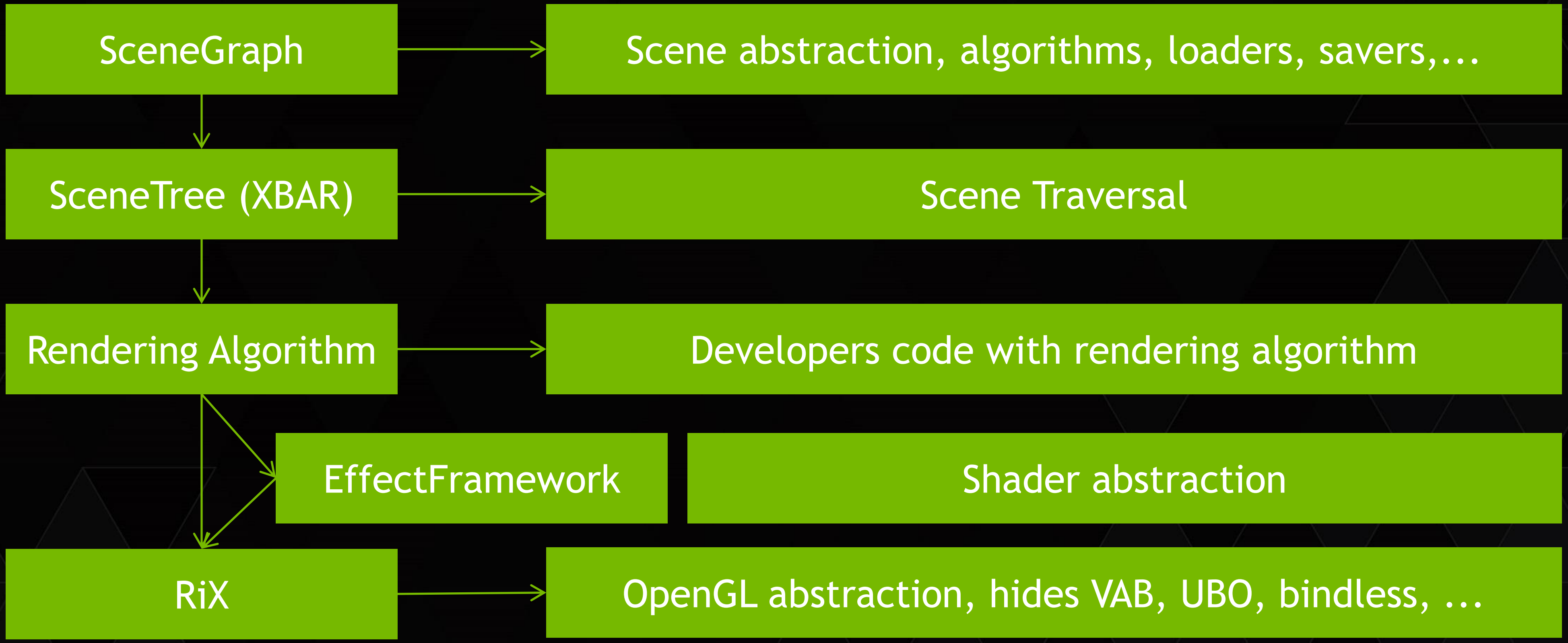
Pipeline for OpenGL techniques

Pipeline for driver verification

NVPRO-PIPELINE MODULES

SceneGraph [dp::sg]	RiX (Renderer) [dp::rix]	Effect System [dp::fx]	Utilities [dp::util]
Algorithms	GL Backend [dp::rix::gl]	XML Based for GLSL [dp::fx::xml]	Math library [dp::math]
SceneTree (XBAR)	Vulkan backend planned		Culling [dp::culling]
Loaders/Savers			Windowing [dp::ui]
Renderer for RiX::GL			Manipulators [dp::ui::manipulator]

RENDERING PIPELINE



SceneGraph → Scene abstraction, algorithms, loaders, savers,…

SceneTree (XBAR) → Scene Traversal

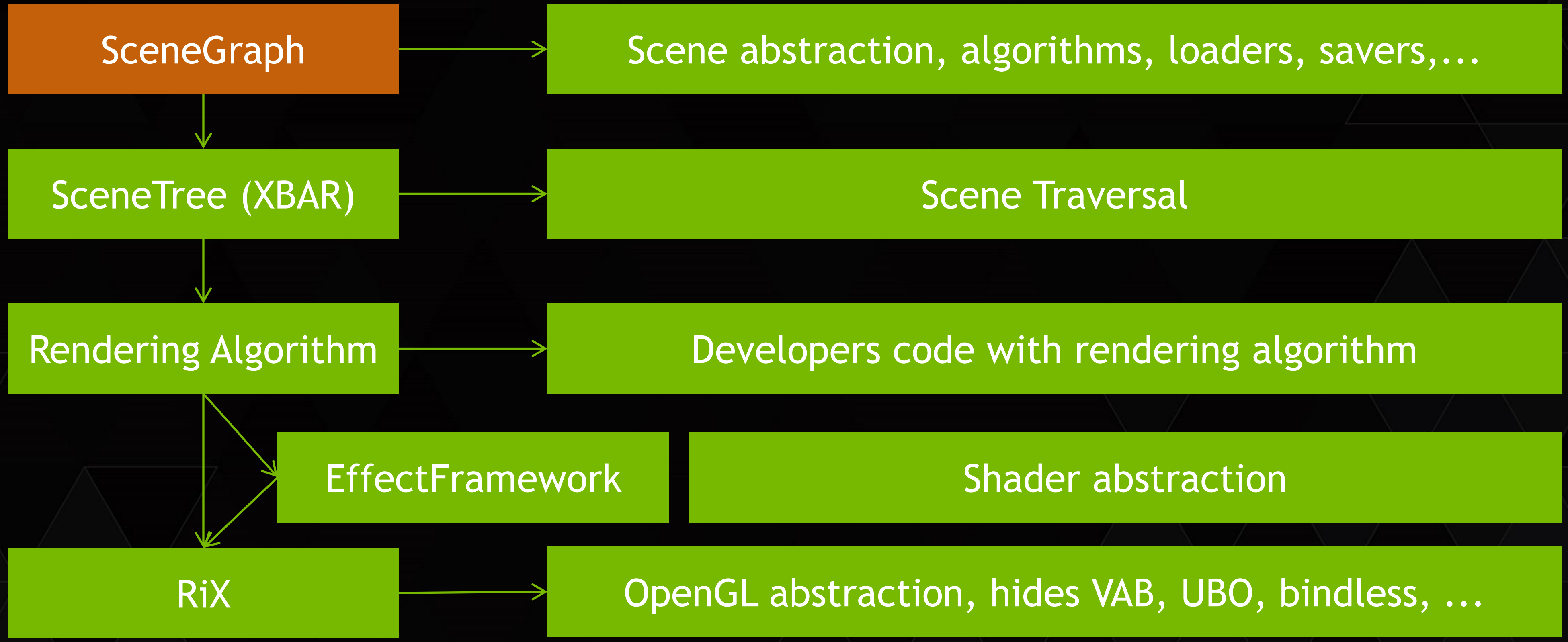
Rendering Algorithm → Developers code with rendering algorithm

EffectFramework

Shader abstraction

RiX → OpenGL abstraction, hides VAB, UBO, bindless, …

RENDERING PIPELINE



SceneGraph

Scene abstraction, algorithms, loaders, savers,…

SceneTree (XBAR)

Scene Traversal

Rendering Algorithm

Developers code with rendering algorithm

EffectFramework

Shader abstraction

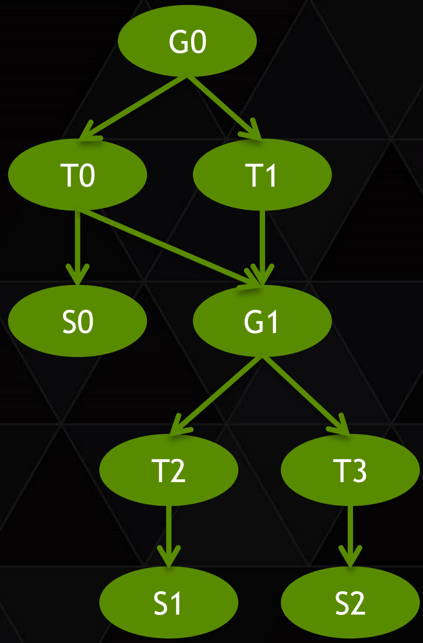
RiX

OpenGL abstraction, hides VAB, UBO, bindless, …

SCENEGRAPH

Traverse & Render

- Simplified version of SceniX SceneGraph
  - GeoNodes, Groups, Transforms, Billboards, Switches still available
  - Animated* objects have been removed to make development easier
  - New property based animation system prepared, but not yet active (LinkManager)

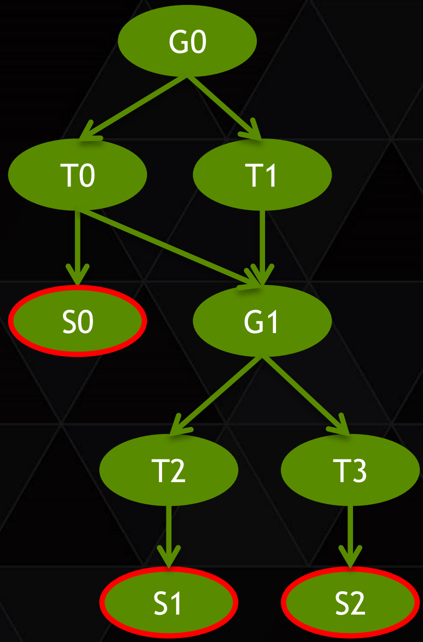


SCENEGRAPH TRAVERSAL COST

- Memory cost
  - Objects scattered in RAM
    - Latency when accessing an object
  - Objects are big
    - Traversing one object might touch multiple cache-lines

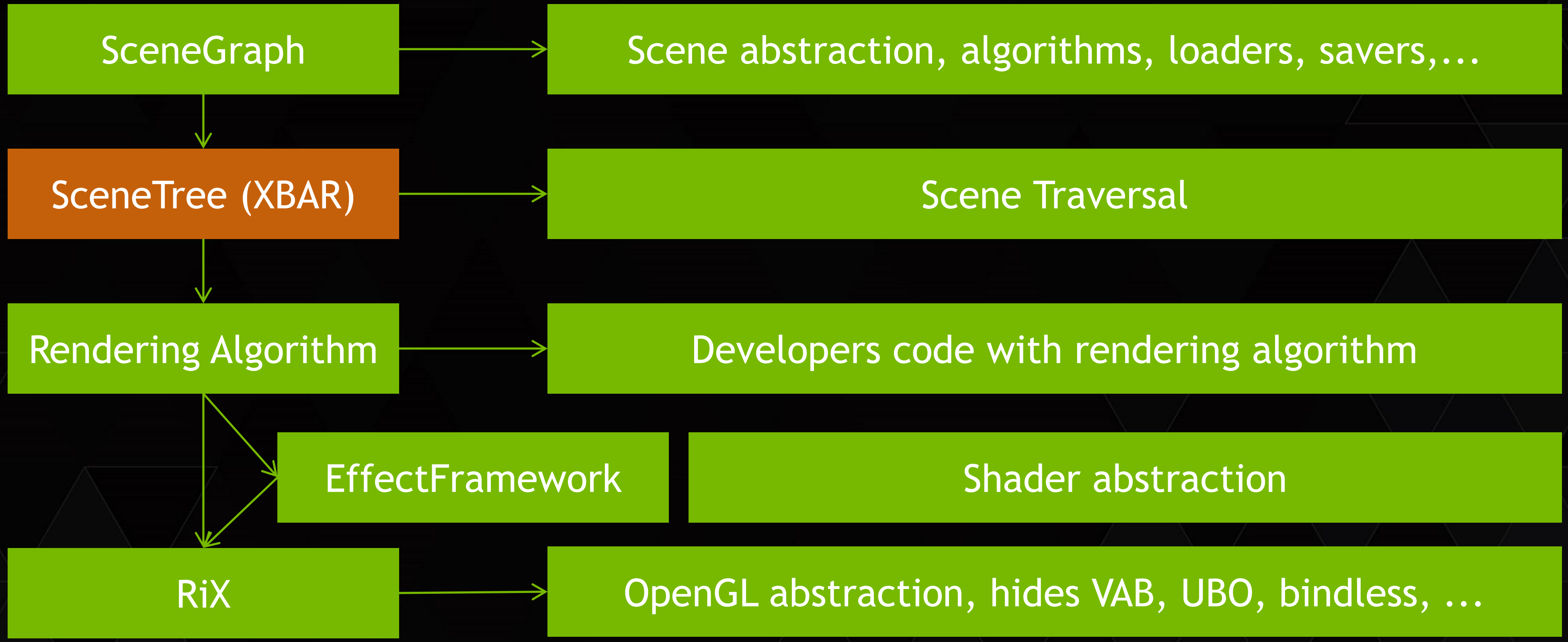
- Instruction calling cost

  -
    ```
    void processNode(Node *node) {      // function call
      switch (node->getType()) {        // branch misprediction
         case Group:
           handleGroup((Group*)node); // virtual function call
           break;
         case Transform:
           handleTransform((Transform*)node);
             break;
         case GeoNode:
           handleGeoNode((GeoNode*)node);
           break;
     }
    ```
- Transformation Cost
  - Compute accumulated transformations during traversal
- Hierarchy Cost
  - Deep hierarchy adds ‚needless‘ traversal cost (5/14 nodes in example of interest)



GPU TECHNOLOGY CONFERENCE

RENDERING PIPELINE



SceneGraph → Scene abstraction, algorithms, loaders, savers,…

SceneTree (XBAR) → Scene Traversal

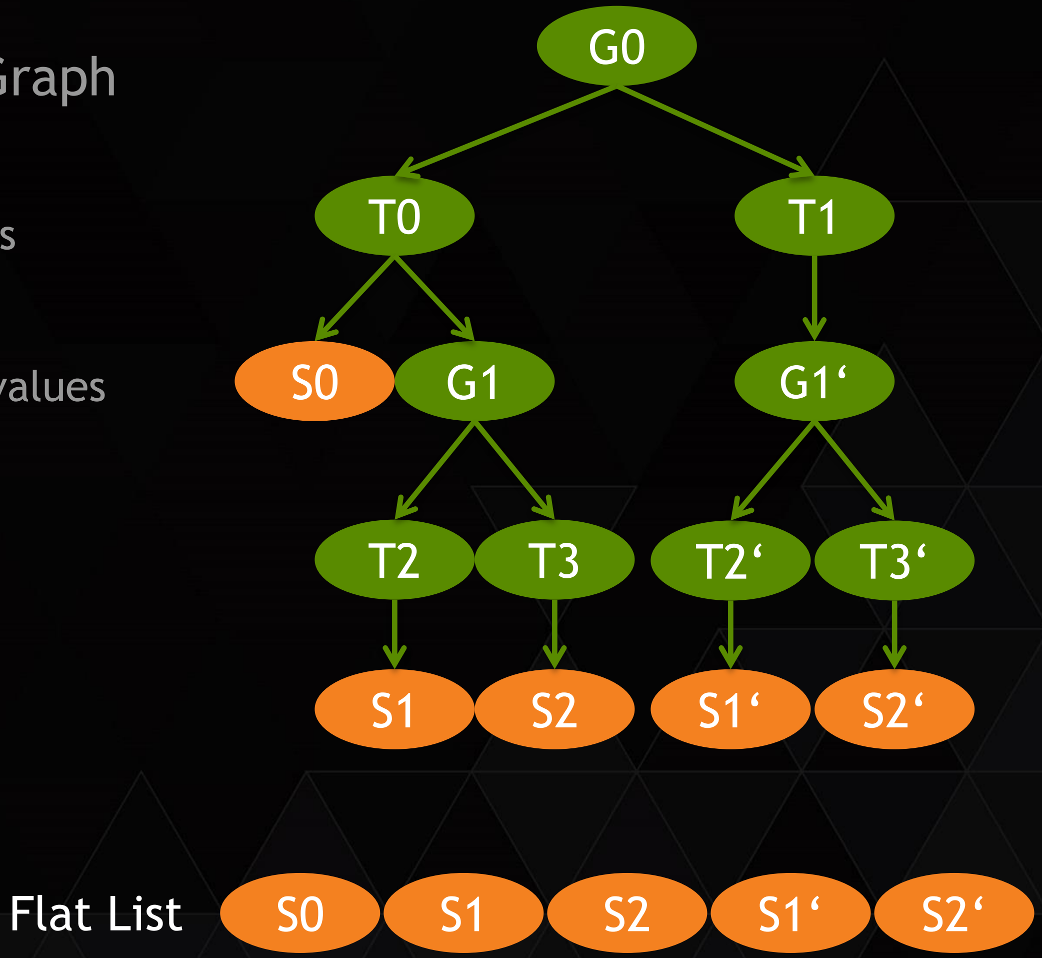
Rendering Algorithm → Developers code with rendering algorithm

EffectFramework — Shader abstraction

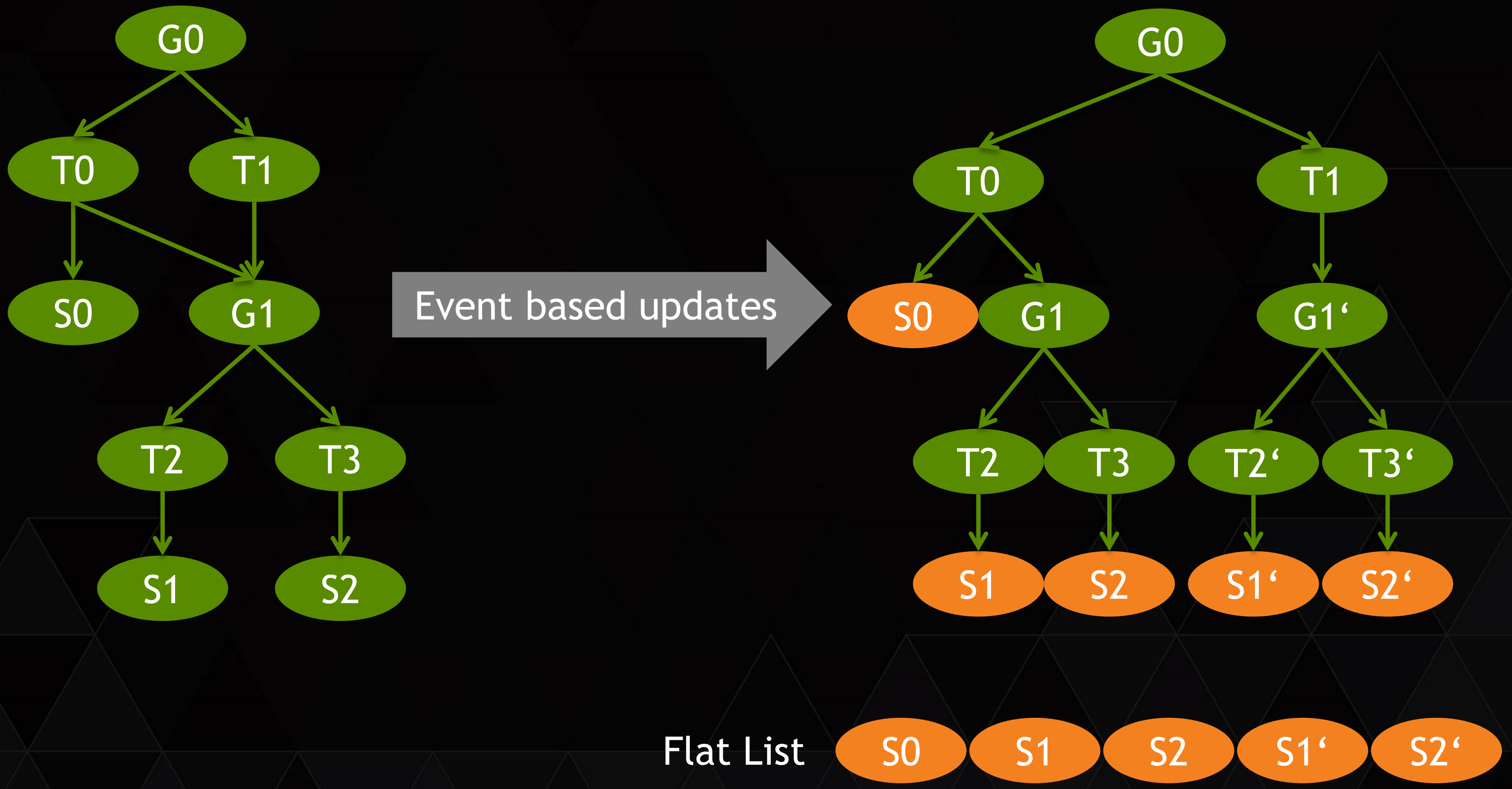
RiX → OpenGL abstraction, hides VAB, UBO, bindless, …

SCENETREE REQUIREMENTS

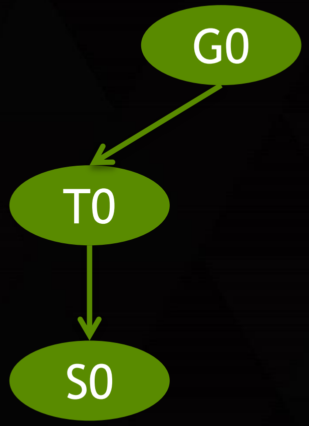
- Generate on the fly from SceneGraph
- Incremental updates
  - Minimal amount of work on changes
- Caching mechanism per path
  - No recomputation of ‚unchanged‘ values
- Flat list of GeoNodes
  - Get rid of traversal
- Memory efficient
  - Don‘t copy data, keep references

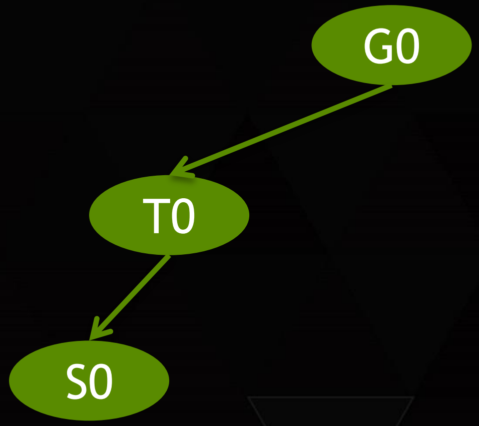


SCENETREE CONSTRUCTION



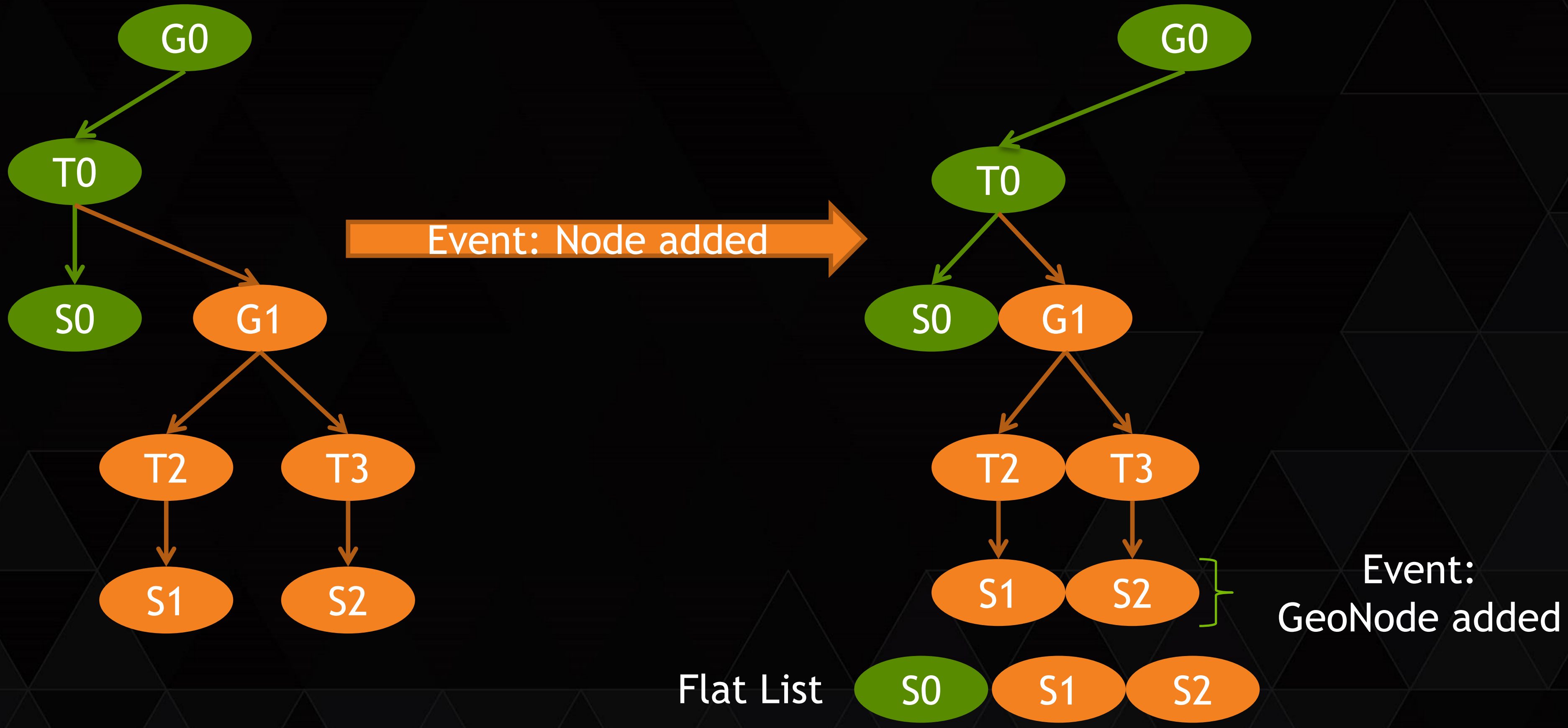
SCENETREE CONSTRUCTION





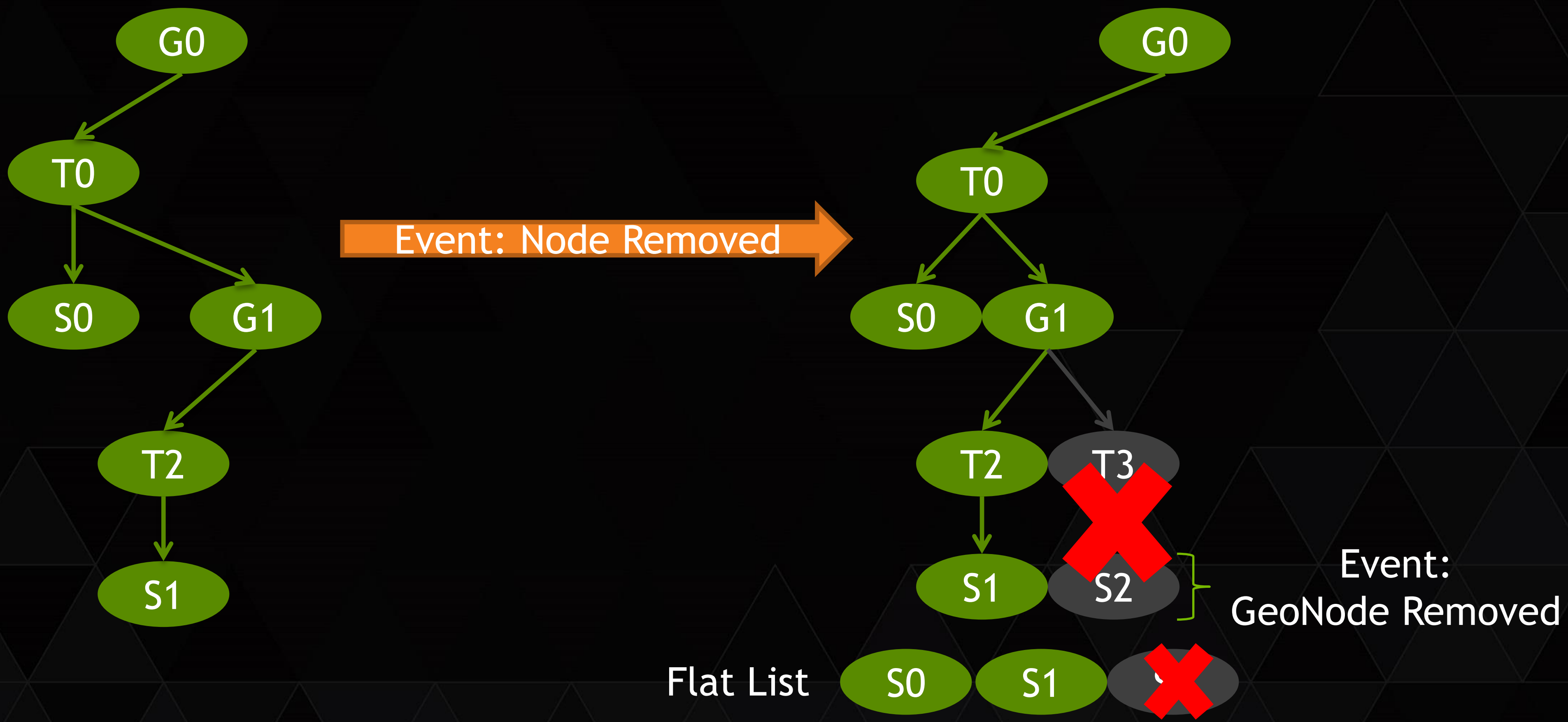
Flat List S0

SCENETREE CONSTRUCTION

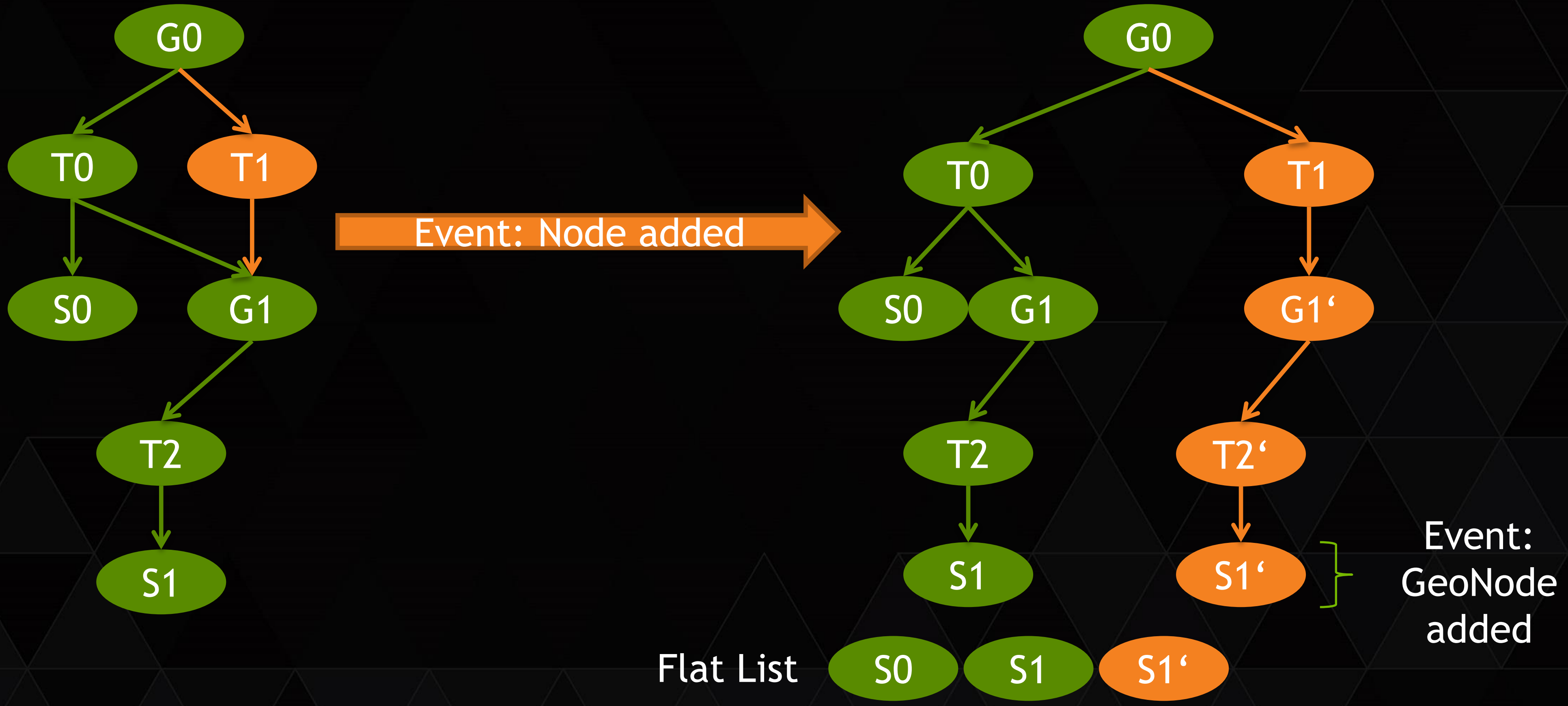


GPU TECHNOLOGY CONFERENCE

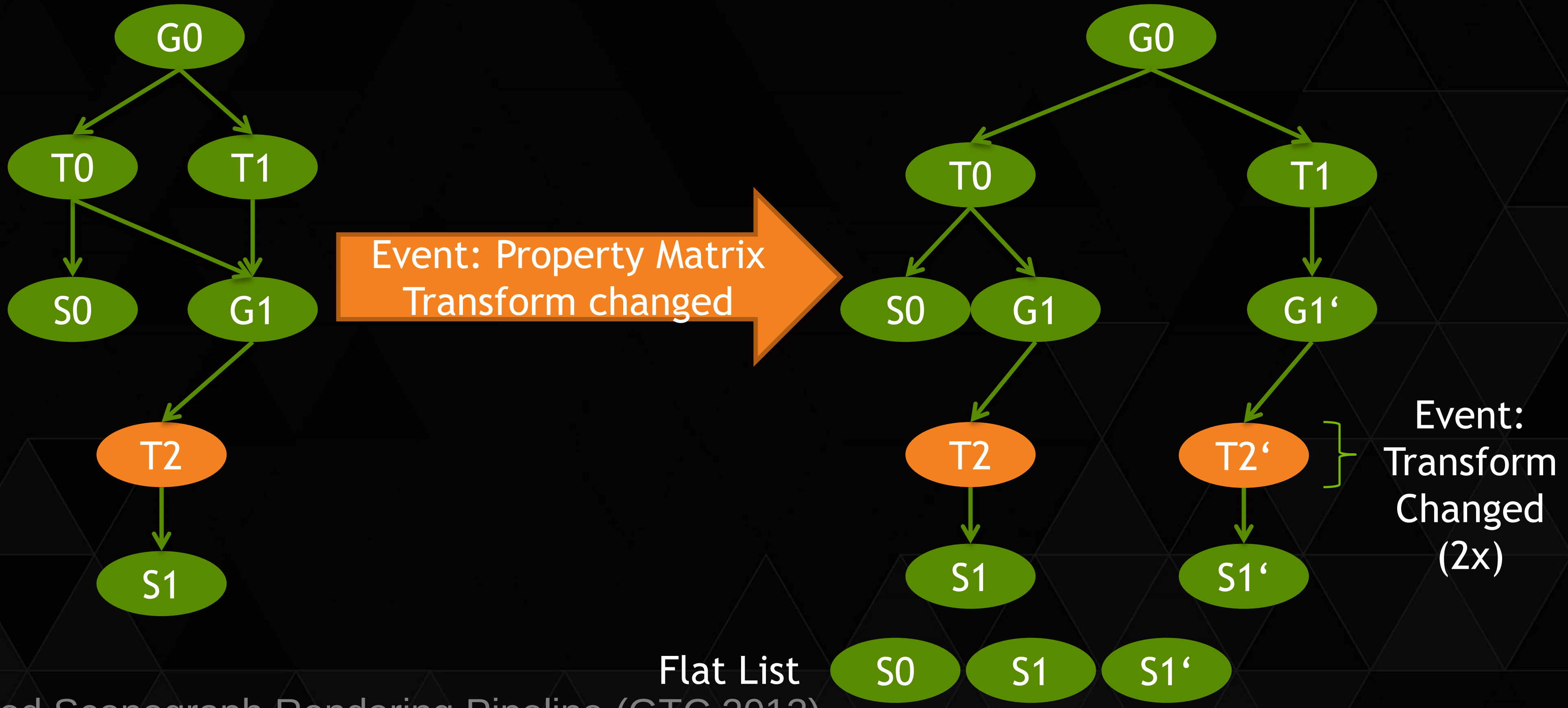
SCENETREE CONSTRUCTION



SCENETREE CONSTRUCTION

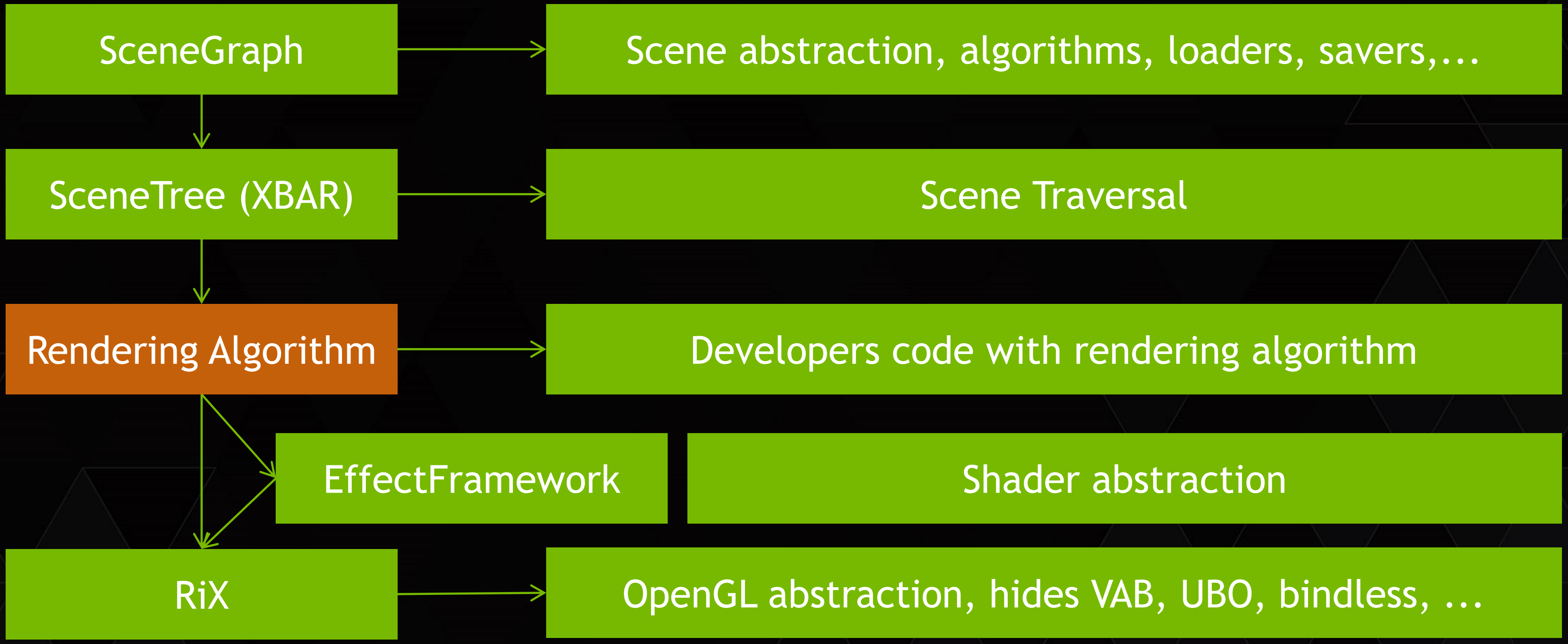


SCENETREE CONSTRUCTION


G0
T0
T1
S0
G1
T2
S1
Event: Property Matrix Transform changed
G0
T0
T1
S0
G1
G1‘
T2
T2‘
S1
S1‘
Event: Transform Changed (2x)
Flat List
S0
S1
S1‘

Construction:
S3032 Advanced Scenegraph Rendering Pipeline (GTC 2013)

RENDERING PIPELINE



SceneGraph → Scene abstraction, algorithms, loaders, savers,...

SceneTree (XBAR) → Scene Traversal

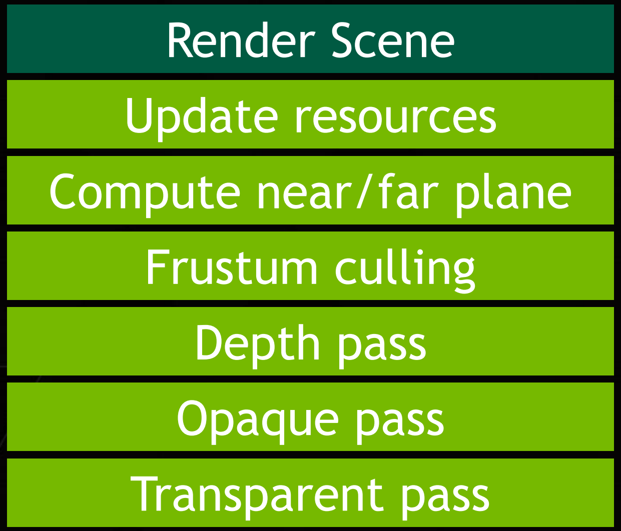
Rendering Algorithm → Developers code with rendering algorithm

EffectFramework – Shader abstraction

RiX → OpenGL abstraction, hides VAB, UBO, bindless, ...

SCENERENDERER

- Observe SceneTree to track GeoNodes in arrays
- dp::sg::renderer::rix::gl is ‚example‘ renderer



Render Scene

Update resources

Compute near/far plane

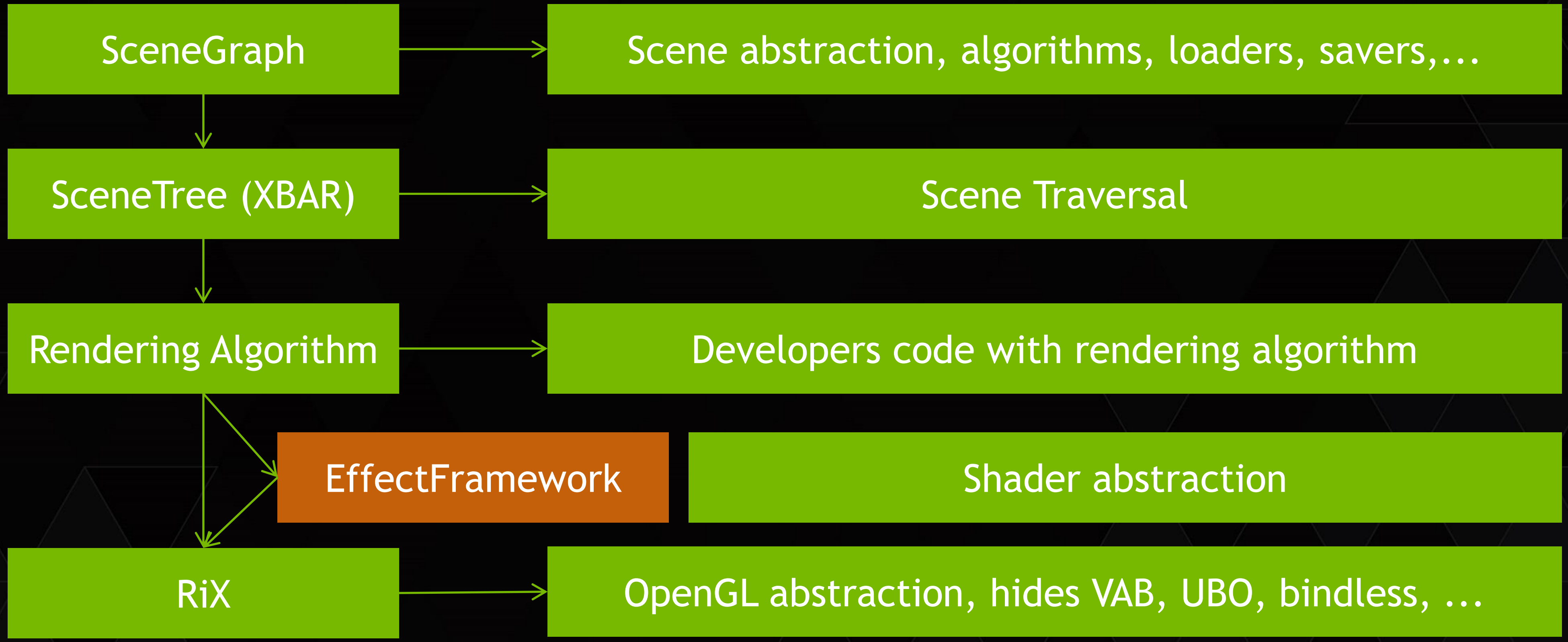
Frustum culling

Depth pass

Opaque pass

Transparent pass

RENDERING PIPELINE



SceneGraph → Scene abstraction, algorithms, loaders, savers,...

SceneTree (XBAR) → Scene Traversal

Rendering Algorithm → Developers code with rendering algorithm

EffectFramework — Shader abstraction

RiX → OpenGL abstraction, hides VAB, UBO, bindless, ...

ANATOMY OF A SHADER

Shader Part	Source Code Example	Pipeline Module
Version Header	// version header & extensions #version 330 #extension GL_NV_shader_buffer_load : enable	Renderer

Uniforms	// Uniforms
	uniform struct Parameters{
	  float parameter;
	};
		Material description

Attributes	// vertex attributes (vertex shader) layout(location = 0) in vec4 attrPosition;	(Material description)
Shader Stage variables	in/out vec3 varPosition;	Hardcoded or generated

Library functions	Bsdf*(params);
	determineMaterialColor();
	determineNormal();
		User provided to generator

User Implementation	void main()
	{
	  // some code
	}
		Material description or rendering system

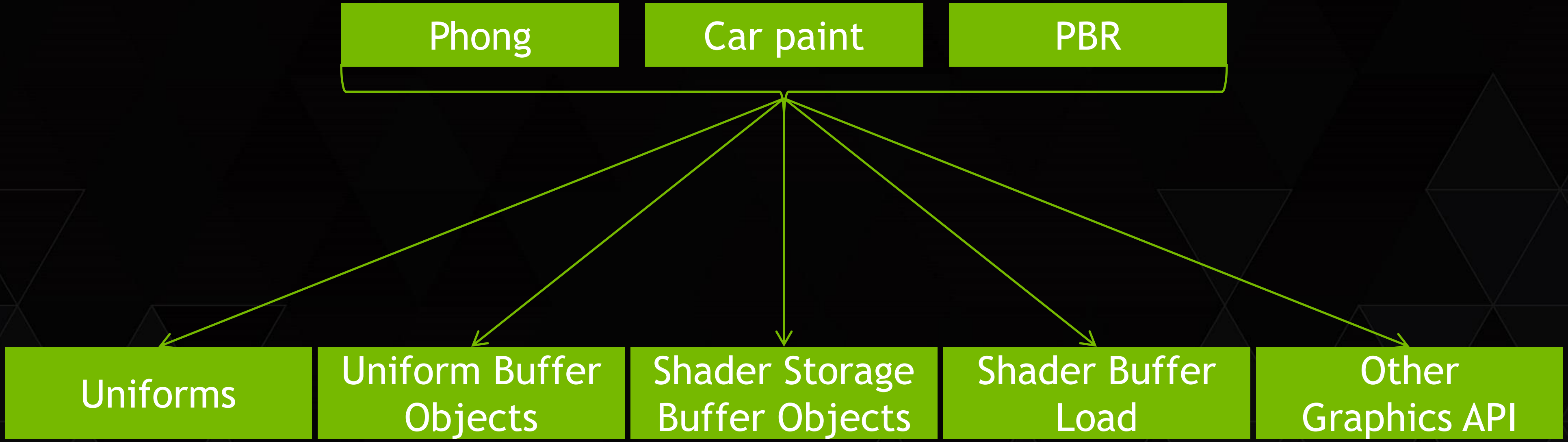
PARAMETER GROUPING

	ParameterGroupSpecs	Binding Frequency
EffectSpec	Shader independent globals, i.e. camera	constant
	Shader dependent globals, i.e. environment map	
	Light, i.e. light sources and shadow maps	rare
	Material parameters without objects, i.e. float, int and bool	frequent
	Material parameters with objects, i.e. textures and buffers	
	Object parameters, i.e. position/rotation/scaling	always

GPU TECHNOLOGY CONFERENCE

EFFECT FRAMEWORK GOALS

- Unique shader interface with support of multiple rendering APIs
- Code generation for different kind of parameter techniques, i.e.



PARAMETER SHADER CODE GENERATION

ParameterGroup phong_fs	
vec3	ambient
vec3	diffuse
vec3	specular
float	specularExp

Uniforms

```
uniform  vec3 ambient;
uniform  vec3 diffuse;
uniform  vec3 specular;
uniform  float specularExp;
```

UBO

```
layout(std140)
uniform ubo_phong_fs {
  uniform  vec3 ambient;
  uniform  vec3 diffuse;
  uniform  vec3 specular;
  uniform  float specularExp;
}
```

shaderbufferload

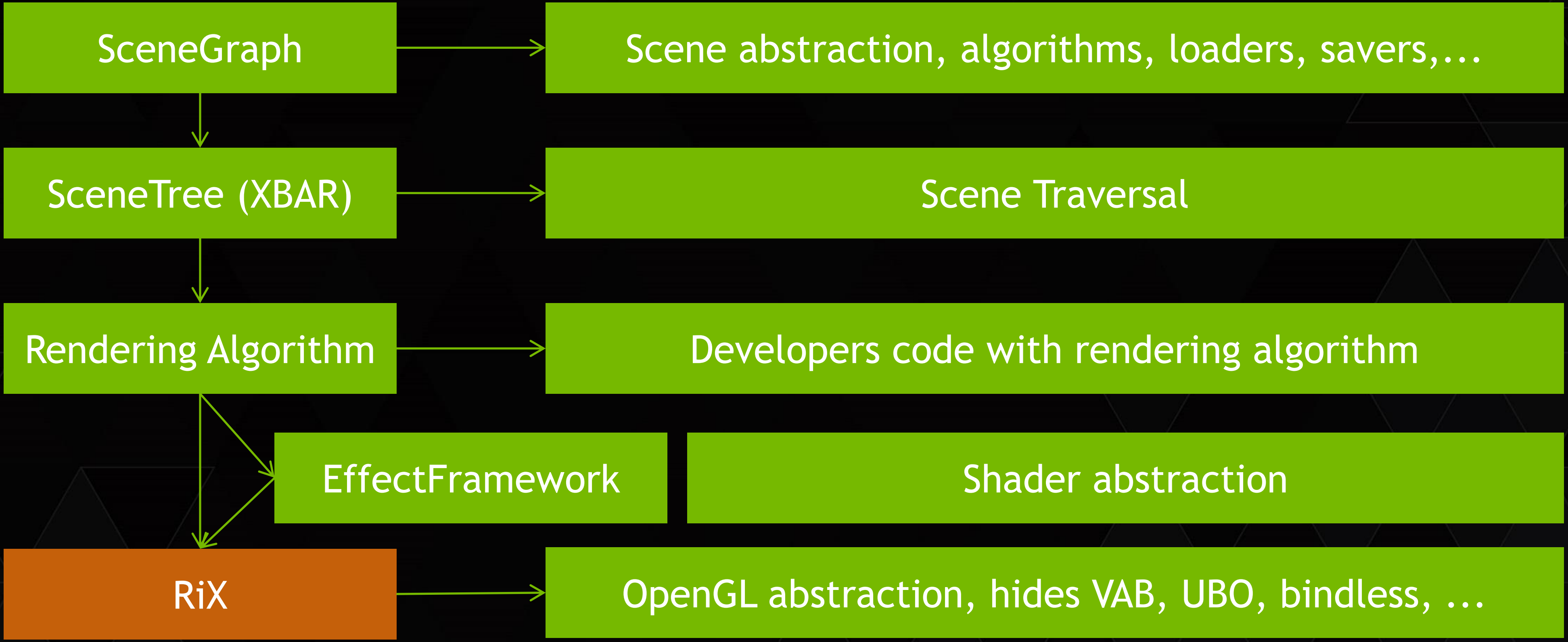
```
struct sbl_phong_fs {
  uniform  vec3 ambient;
  uniform  vec3 diffuse;
  uniform  vec3 specular;
  uniform  float specularExp;
}

uniform sbl_phong_fs *sys_phong_fs;

#define ambient     sys_phong_fs->ambient
#define diffuse     sys_phong_fs->diffuse
#define specular    sys_phong_fs->specular
#define specularExp sys_phong_fs->specularExp
```

Details:
S3032 Advanced Scenegraph Rendering Pipeline (GTC 2013)

RENDERING PIPELINE



SceneGraph → Scene abstraction, algorithms, loaders, savers,...

SceneTree (XBAR) → Scene Traversal

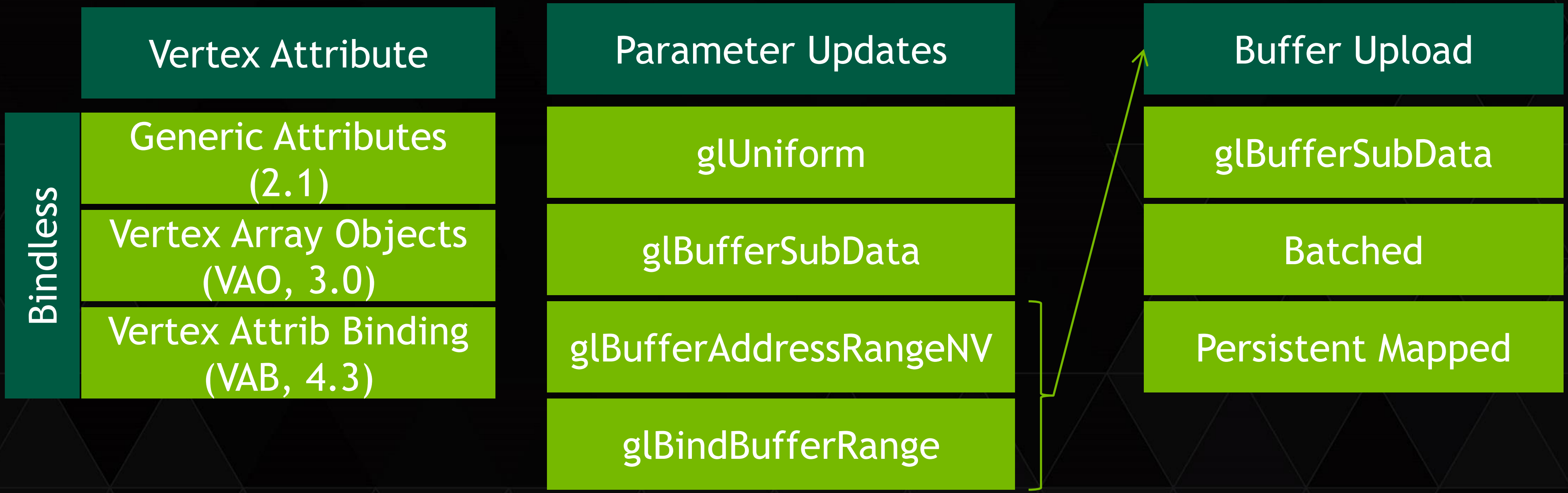
Rendering Algorithm → Developers code with rendering algorithm

EffectFramework – Shader abstraction

RiX → OpenGL abstraction, hides VAB, UBO, bindless, ...

RIX

- Rendering API abstraction with OpenGL backend in place
- Hide implementation details which generate all kind of (OpenGL) streams



Vertex Attribute

Bindless

Generic Attributes (2.1)

Vertex Array Objects (VAO, 3.0)

Vertex Attrib Binding (VAB, 4.3)

Parameter Updates

glUniform

glBufferSubData

glBufferAddressRangeNV

glBindBufferRange

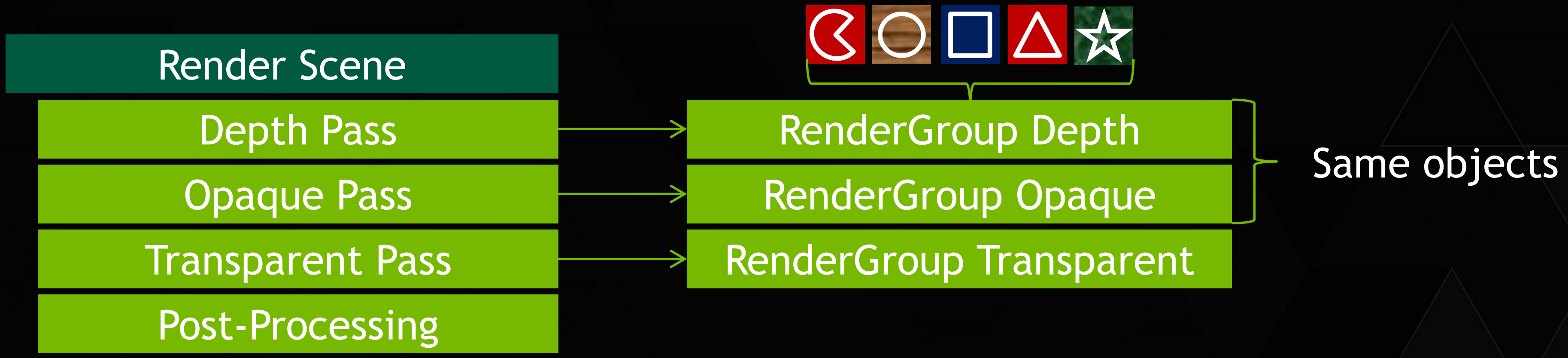
Buffer Upload

glBufferSubData

Batched

Persistent Mapped

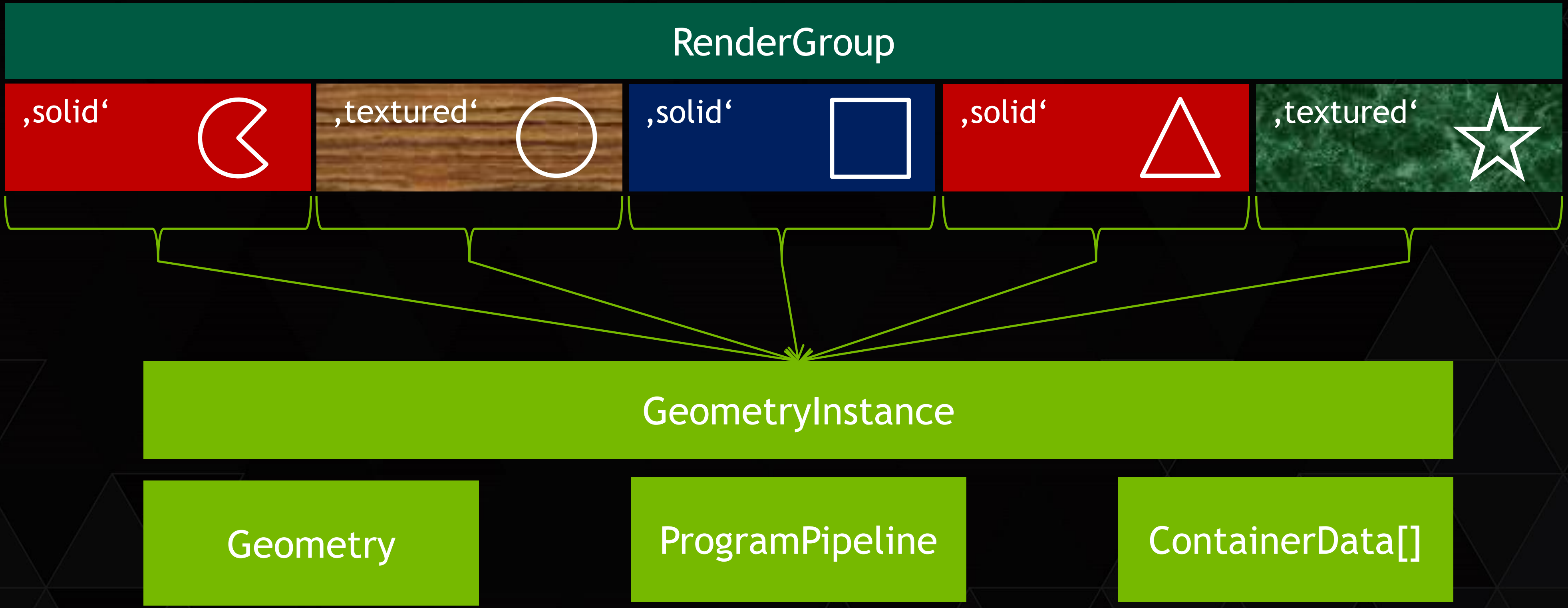
RENDER PIPELINE USING RIX



- RenderGroup per render pass
  - Rendering cache can be optimized for pass
  - Depth-Pass might require only positions, but not normals and texture coordinates -> smaller cache
    - Fewer OpenGL calls than opaque pass with optimized cache
  - Transparent pass might or might not require ordering

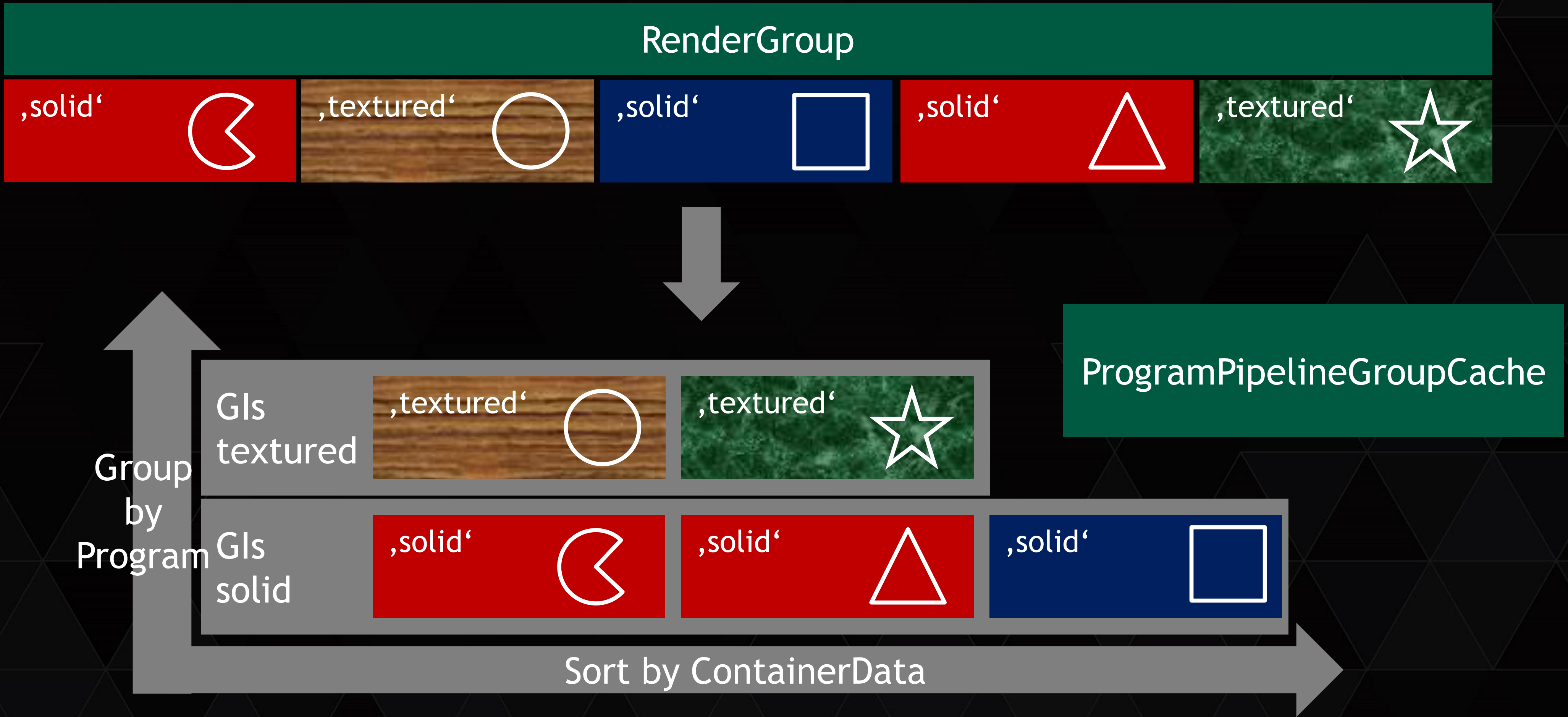
GPU TECHNOLOGY CONFERENCE

RENDER GROUP

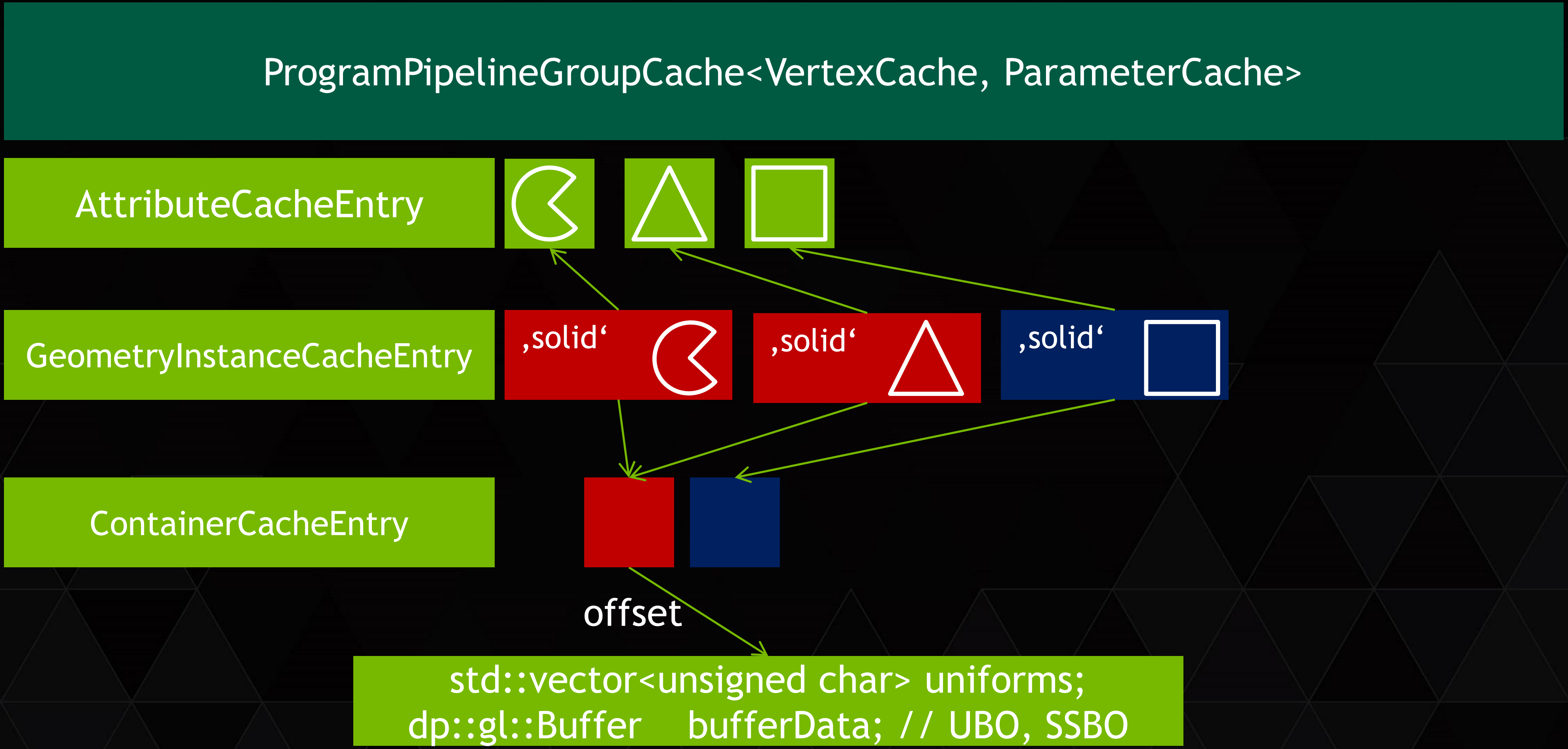


GeometryInstance can only be referenced by single RenderGroup

RENDER GROUP



PROGRAM PIPELINE GROUP CACHE



ProgramPipelineGroupCache<VertexCache, ParameterCache>

AttributeCacheEntry

GeometryInstanceCacheEntry

‚solid‘

‚solid‘

‚solid‘

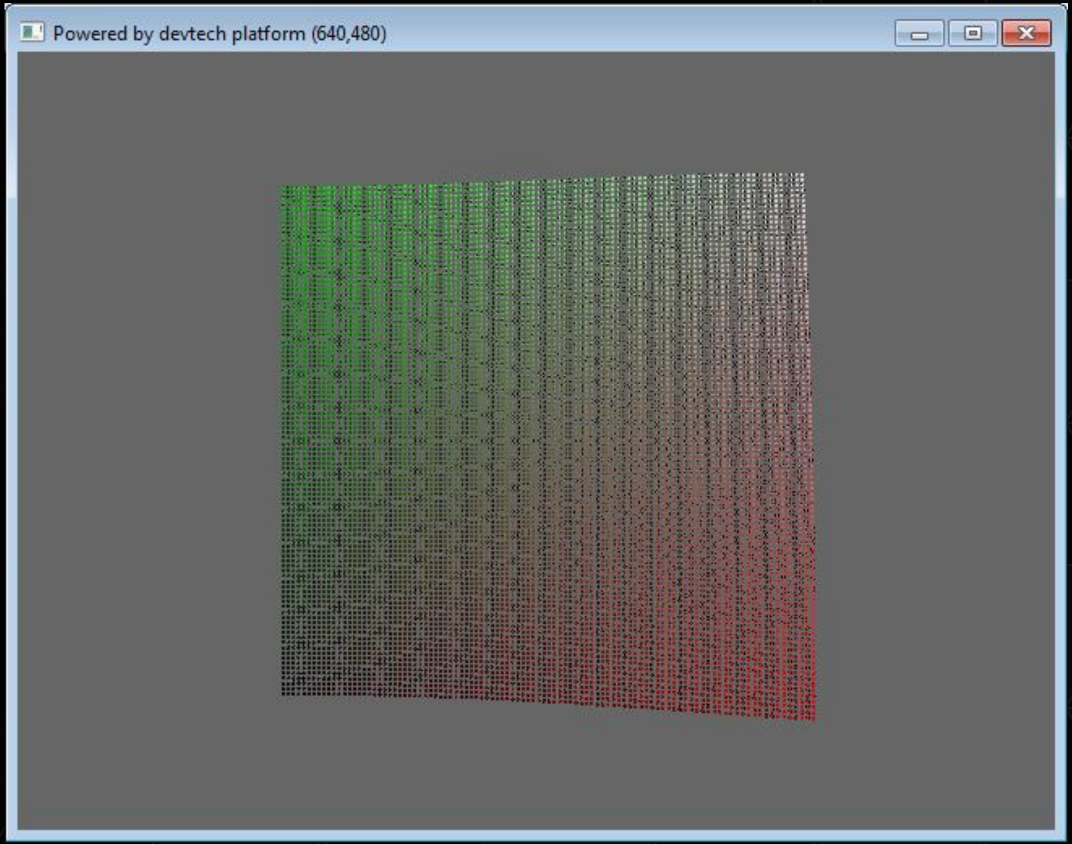
ContainerCacheEntry

offset

```
std::vector<unsigned char> uniforms;
dp::gl::Buffer    bufferData; // UBO, SSBO
```

BENCHMARK

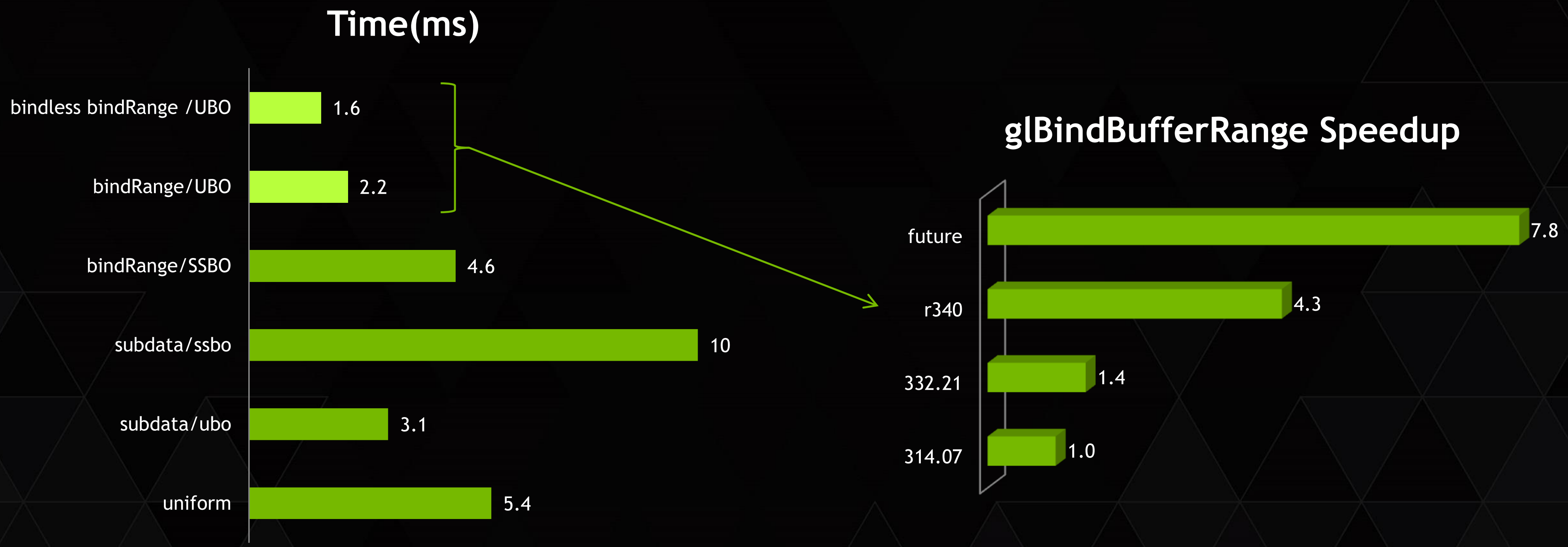
- GLUTAnimation
  - 100x100 Spheres
  - Geometry duplication
  - 5 different materials
  - Each sphere has own ‚color‘



CPU TIME VERTEX TECHNIQUES

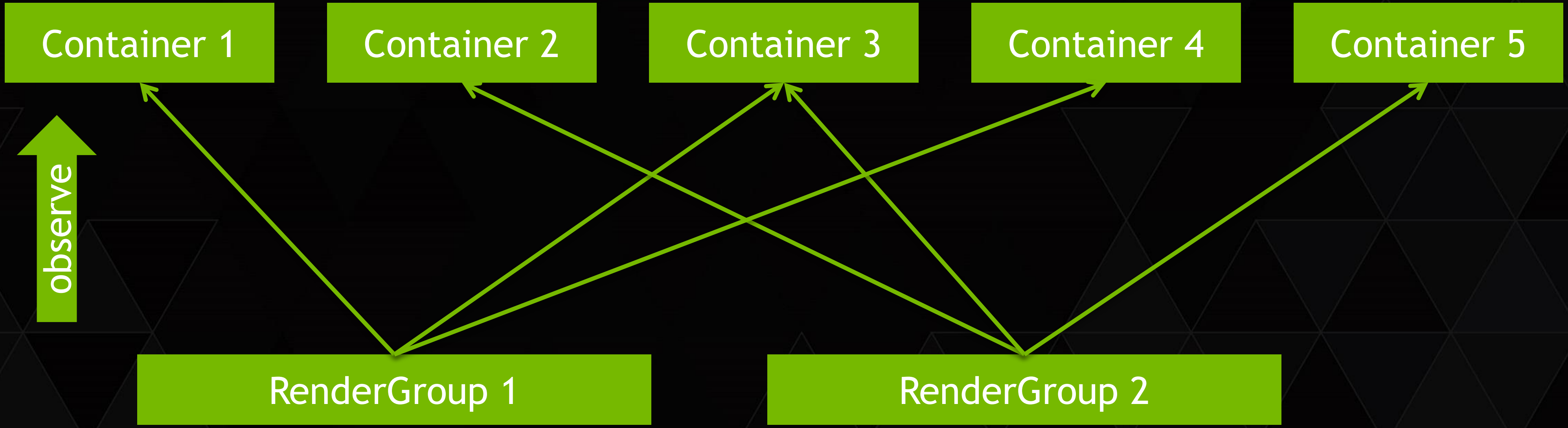
		Bindless	Bindless
Technique	Rendertime (ms)		
VBO	5.7	1.8	2
VAB	4.9	1.6	1.8
VAO	7.5	3.2	3.2
	1 stream		2 stream

CPU TIME PARAMETER TECHNIQUES



PARAMETERS UPDATE HANDLING

- Each RenderGroup has a set of ContainerDatas
  - Map of containerData -> cache position (IMAGE)
- How to manage dirty state per RenderGroup efficient?
  - Set of ContainerData

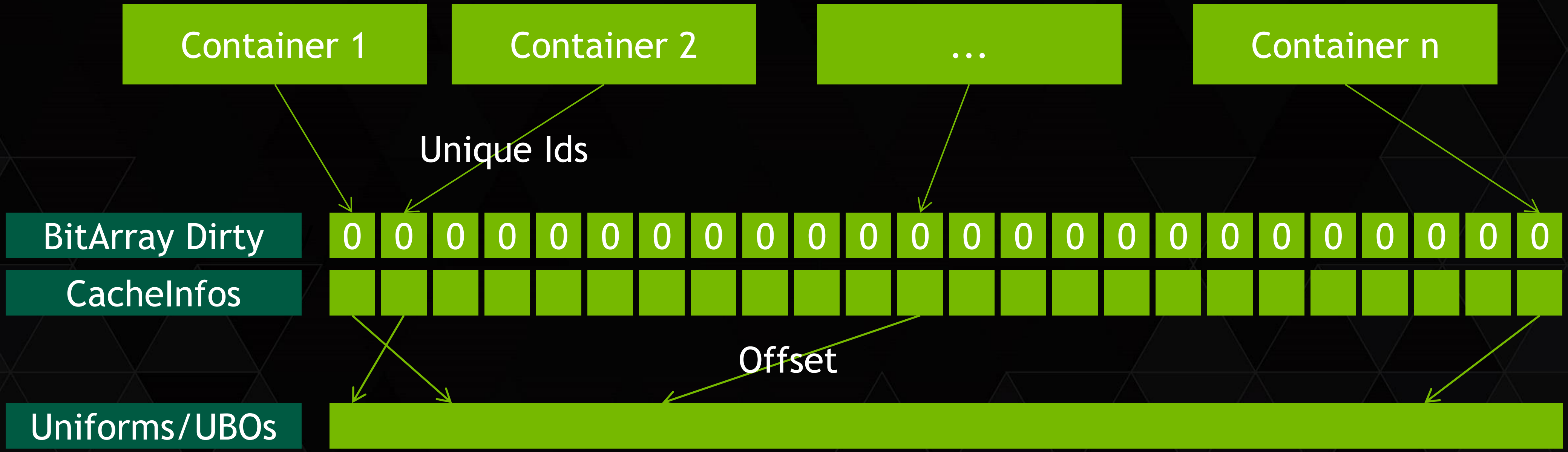


CONTAINERDATA UPDATE HANDLING

- First approach
  - RenderGroup holds std::set<ContainerData> of dirty objects
  - std::map<ContainerData, CacheLocation> for ContainerData->CacheLocation mapping
- Profiling revealed this was a bad idea
  - Dirty phase
    - std::set::insert, top hotspot in GLUTAnimation
    - Binary search, allocation, large amount of ‘random memory access‘ ops
  - Update Phase
    - std::map<ContainerData*,CacheLocation>::find()
    - Binary search, ‘random memory access‘

CONTAINERDATA UPDATE HANDLING

- Second approach
  - Assign each Container a unique id, keep unique ids as dense as possible



Container 1 | Container 2 | ... | Container n

Unique Ids

BitArray Dirty | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

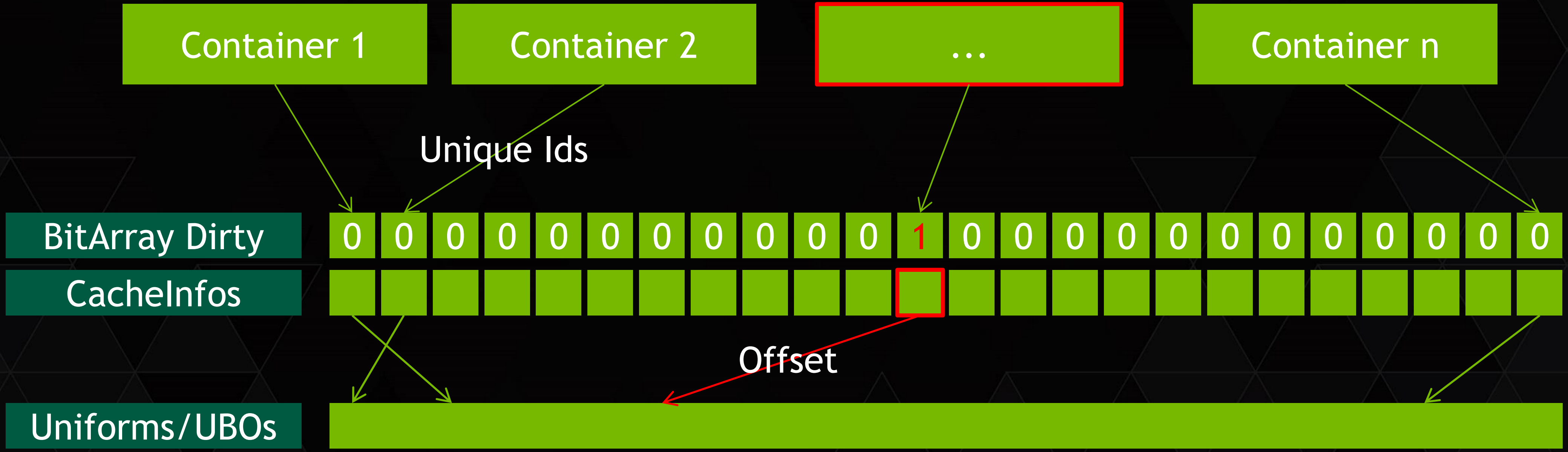
CacheInfos

Offset

Uniforms/UBOs

CONTAINERDATA UPDATE HANDLING

- Update phase: Set bits in dirty array
- Process update phase: Get offset from CacheInfos Array
- Constant time operations



RESULTS

	Time STL (ms)	Time BitArray (ms)		Profiler Hotspot
Do Updates	4.8	2.5	→	Event handling
Process Update	4.0	0.9	→	Cache update
Total Time	8.8	3.6		

BITARRAY::TRAVERSEBITS

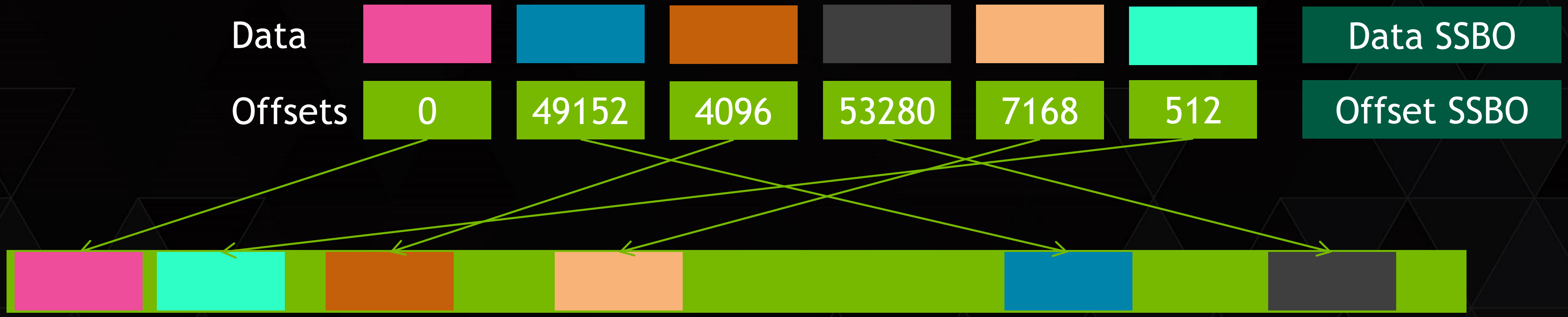
- Linear memory -> cache efficient
- Works on size_t type, skips 32/64 bits if no bit is set in a element
- Uses ctz (count trailing zeroes) intrinsics
    - No branch mispredicion issues on 01001101 pattern
- 1M bits need 122kb, ~0.4us traversal time if no bit set
- As comparison
    - Red-Black treenode has 3 ptrs and a color, at least
        - 64-bytes per node + payload
        - 1953 nodes need more memory than 1m bits
- BitTree would solve linear problem during traversal

SPARSE UBO/SSBO UPDATES

- Efficient algorithm to handle changed containers -> done
- Assuming thousands of Containers referencing UBOs are dirty
    - How to execute an efficient update?
    - One map/unmap call for the UBO?
        - No, too much data transfer between CPU and GPU
    - One mapRange/unmapRange per update?
        - No, mapRange/unmapRange create sync points
    - glBufferSubData?
        - If glBindBufferRange is being used it‘ll be slow too!

SPARSE UBO/SSBO UPDATES

- dp::gl::BufferUpdater
  - Supports updates of any block-size which is a multiple of 16
  - Gathers all updates, uploads them as compact buffer and scatters on the GPU



Data	🟪	🟦	🟧	⬛	🟨	🟩	Data SSBO
Offsets	0	49152	4096	53280	7168	512	Offset SSBO

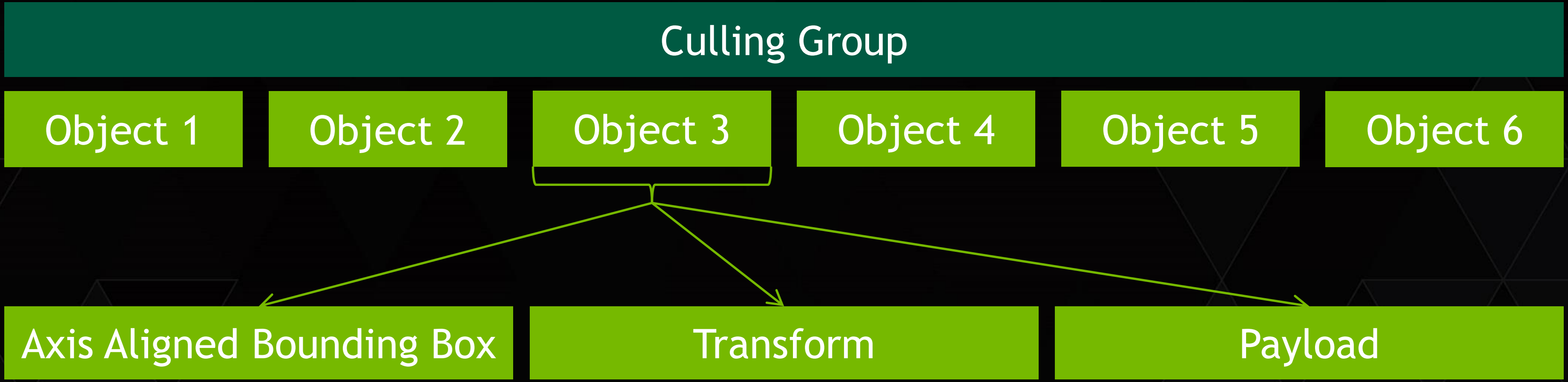
Time glBufferSubData (ms)	Time batched(ms)
7.2	0.07

→ 100x speedup

GPU TECHNOLOGY CONFERENCE

CULLING

- dp::culling abstract API for frustum culling
  - CPU & OpenGL compute backend



CULLING

```
foreach(object : group) {
  isVisible = result->isVisible(object->culling);
  setVisible(object->rix, isVisible);
}
```

- expensive ‚query‘ and update call for each object
- Solution: ResultOject. `Cull(group, result, viewProjection);`

	Culling Group						
	Object 1	Object 2	Object 3	Object 4	Object 5	Object 6	
Old visibility	1	0	0	1	0	1	Result
New visibility	1	0	1	0	0	1	
XOR	0	0	1	1	0	0	

BitArray::TraverseBits on XOR result

RESULTS

- Scene traversal can be avoided for static scene parts
- Rendering time depends a lot on used OpenGL methods
  - VAB + glBindBufferRange UBO good, in combination with bindless best
- BitArrays can be a good tool to avoid maps/sets
- Try to batch small updates to GPU memory

- Still CPU bound?
  - **S5135 - GPU-Driven Large Scene Rendering in OpenGL (Tue 16:00, LL21B)**
- GPU bound?
  - **S5291 - Slicing the Workload: Multi-GPU Rendering Approaches (Web 10:00, LL21B)**

GPU TECHNOLOGY CONFERENCE

THANK YOU

MATAVENRATH@NVIDIA.COM

JOIN THE CONVERSATION

#GTC15

http://github.com/nvpro-pipeline