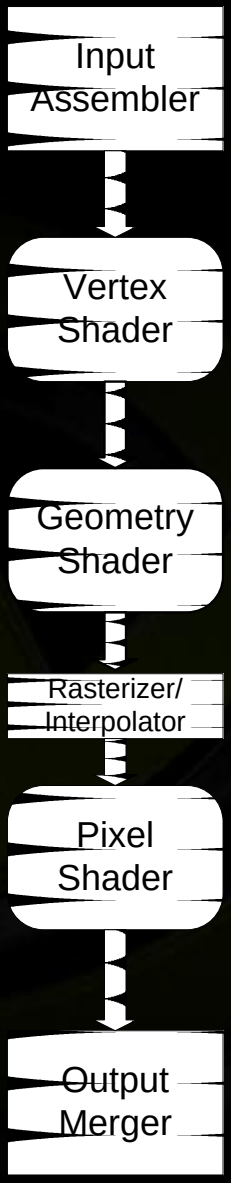

Fast Tessellated Rendering on Fermi GF100
Hot3D @ HPG 2010
Tim Purcell
NVIDIA

Outline

- DX10 state of the art
- DX11 tessellation
- Fermi GF100 architecture
- Results
- Demo

State of the Art: DX10 pipeline



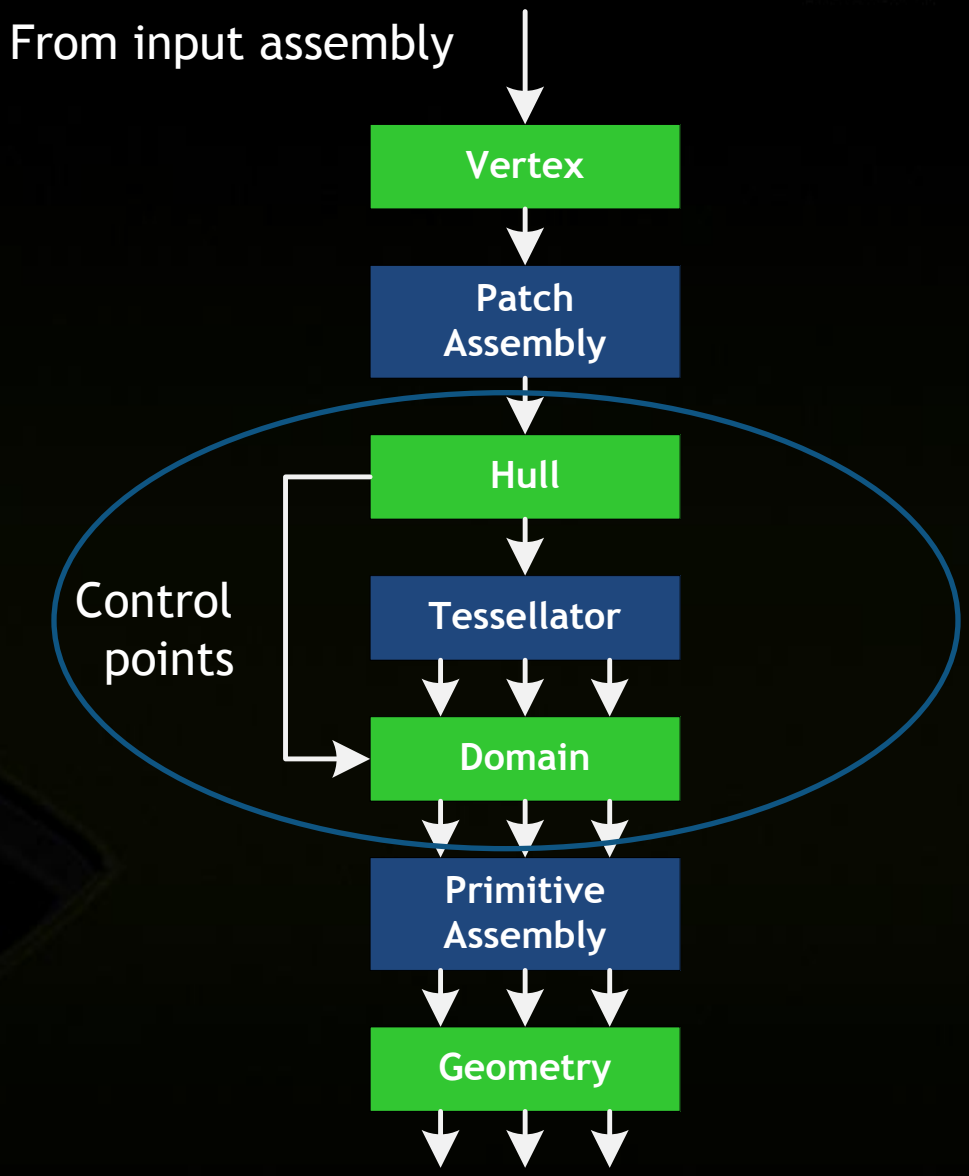
- **Pixels are meticulously shaded**

State of the Art: DX10 pipeline

- Pixels are meticulously shaded, but geometric detail is modest

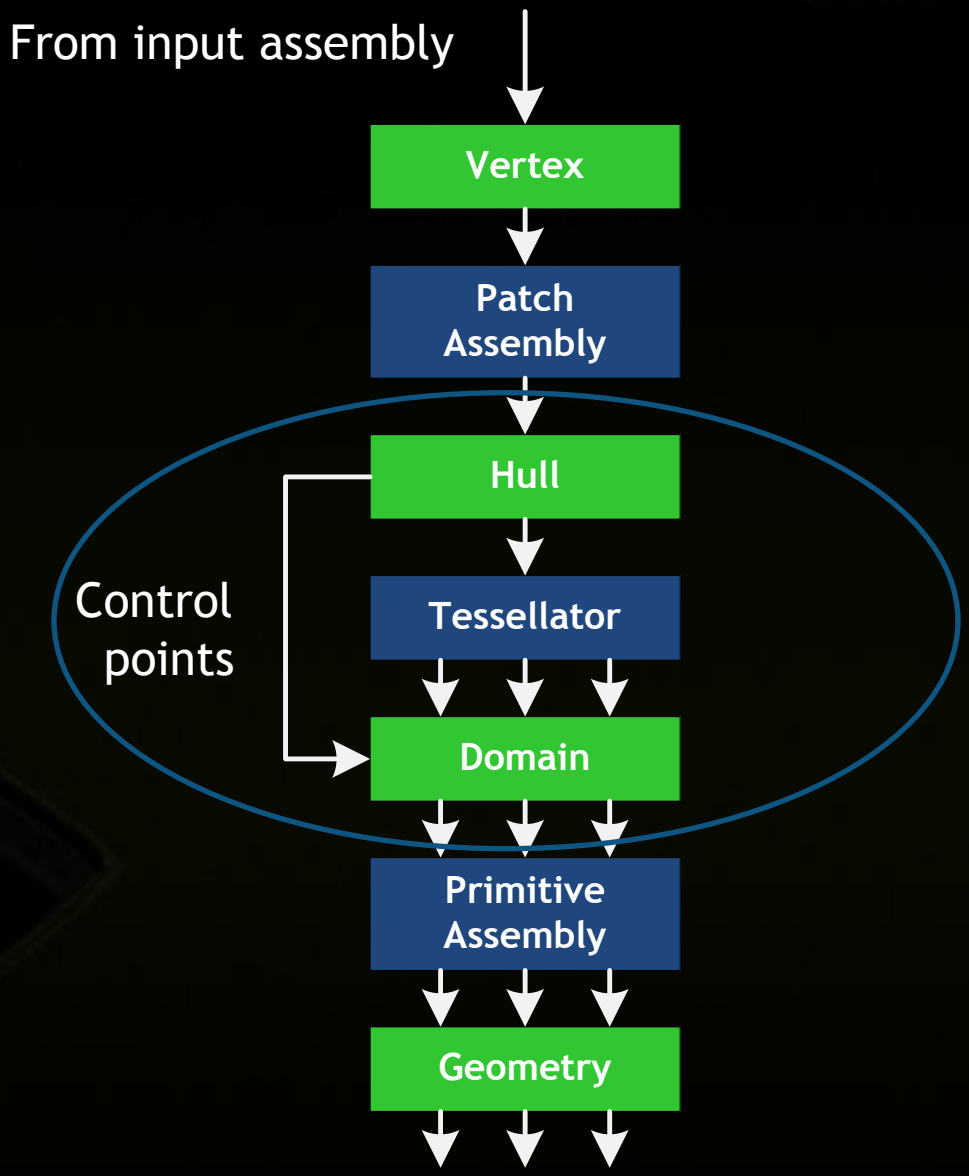
Tessellation in DirectX 11

- **Hull shader**
  - Computes the tessellation factor
  - Runs pre-expansion
  - Explicitly parallel across control points



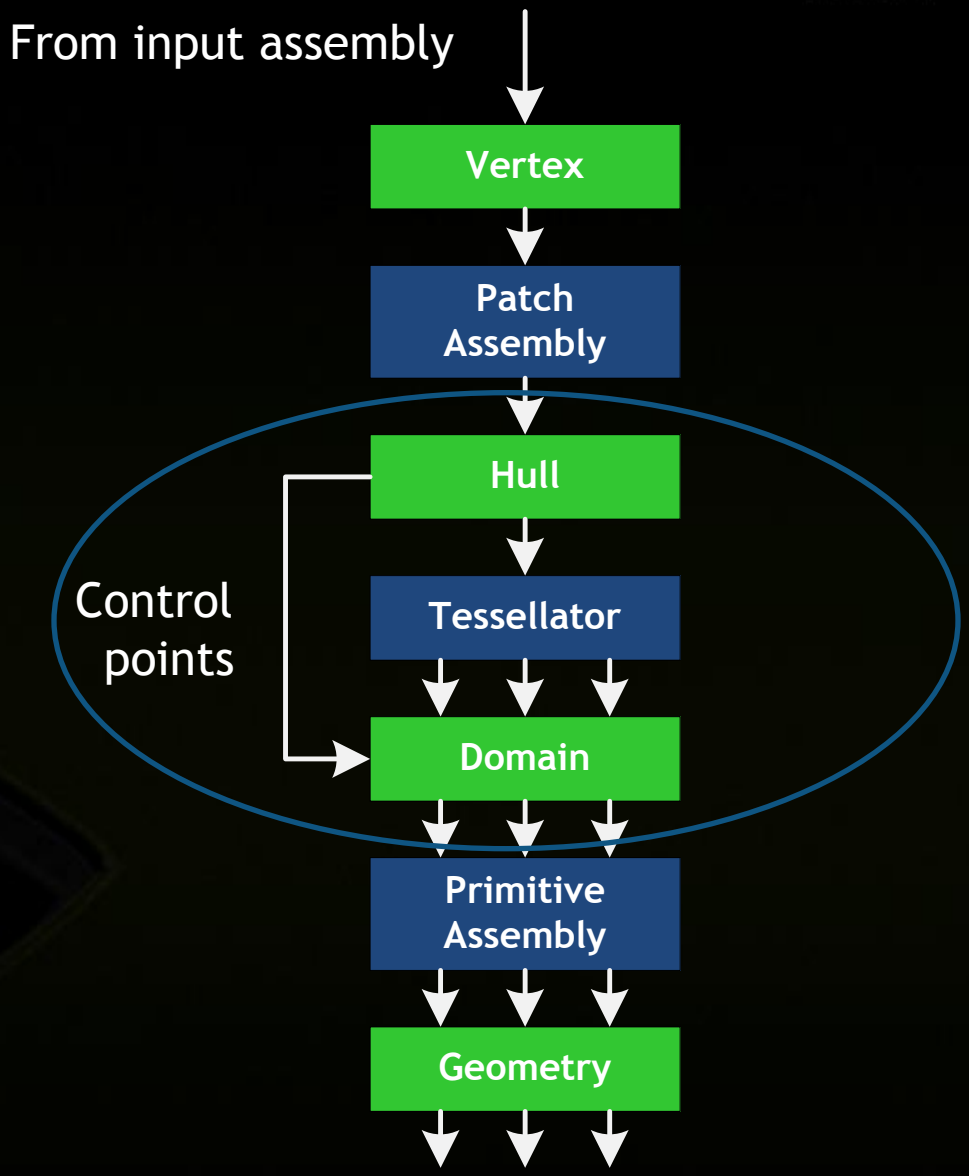
Tessellation in DirectX 11

- Fixed function tessellation stage
  - Configured by LOD output from Hull Shader
  - Produces triangles and lines

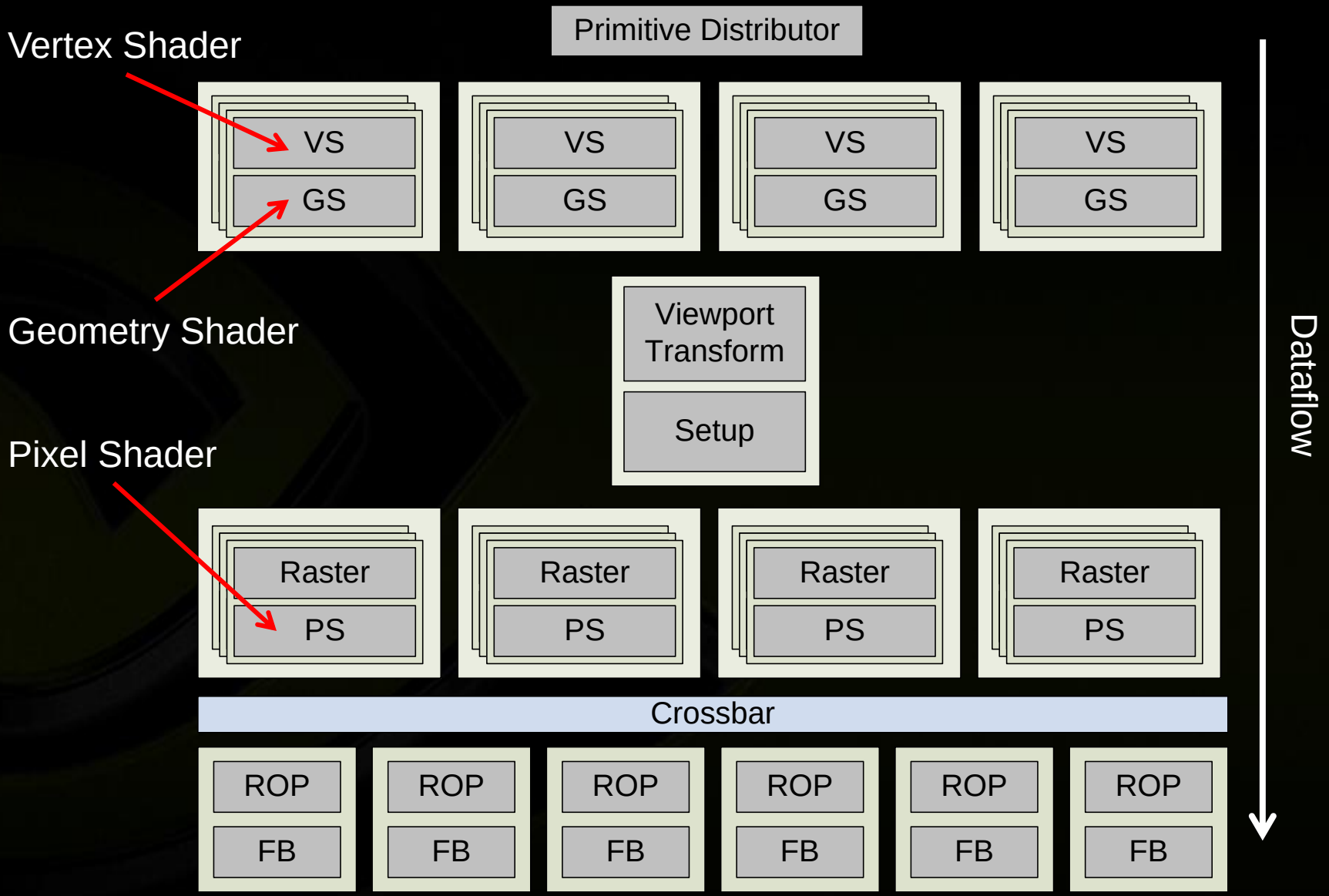


Tessellation in DirectX 11

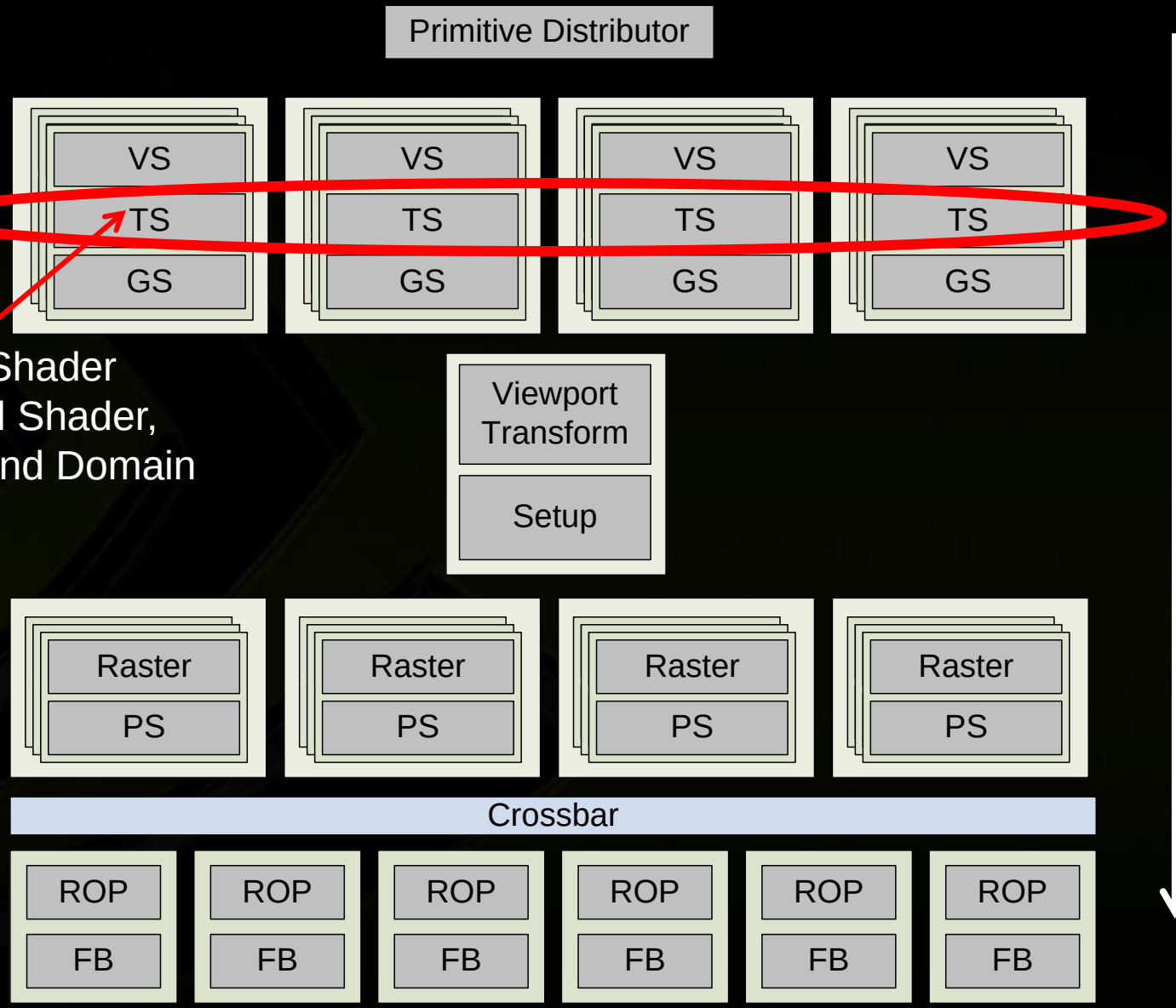
- Domain shader
  - Runs post-expansion
  - Maps (u,v) to (x,y,z,w)
  - Implicitly parallel



NVIDIA DX10 Logical Pipeline

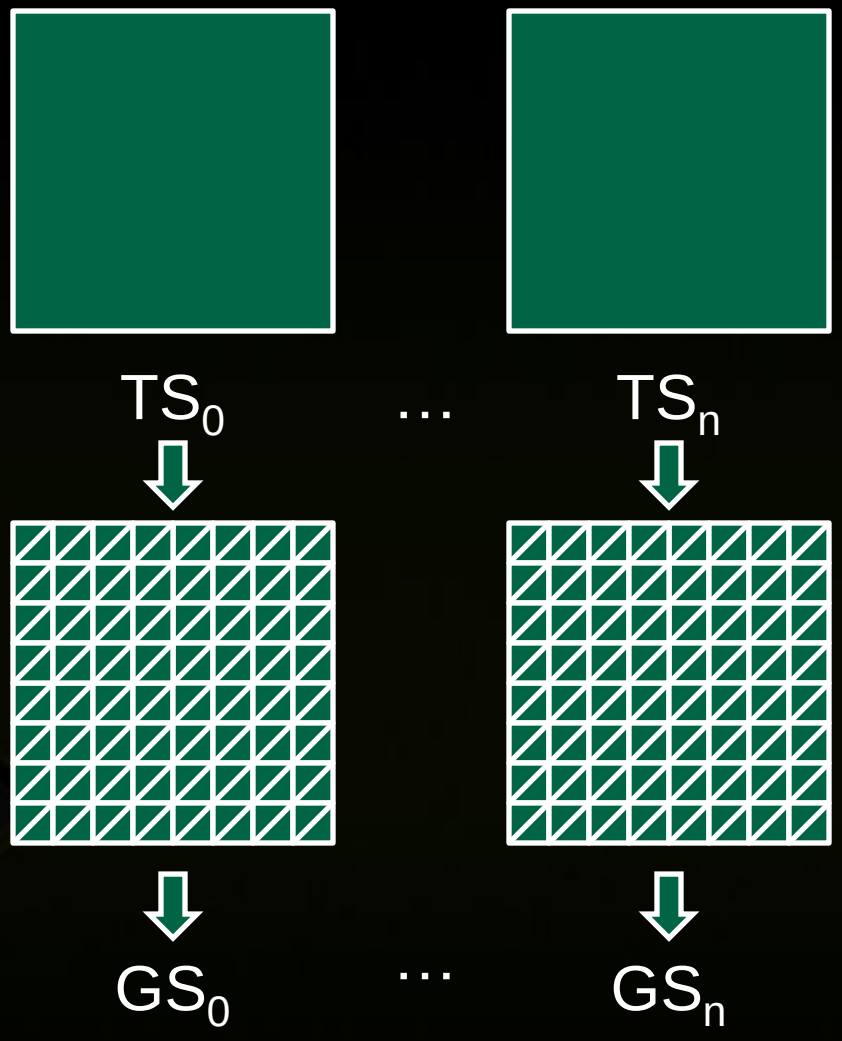


DX10 Logical Pipeline + Tessellation



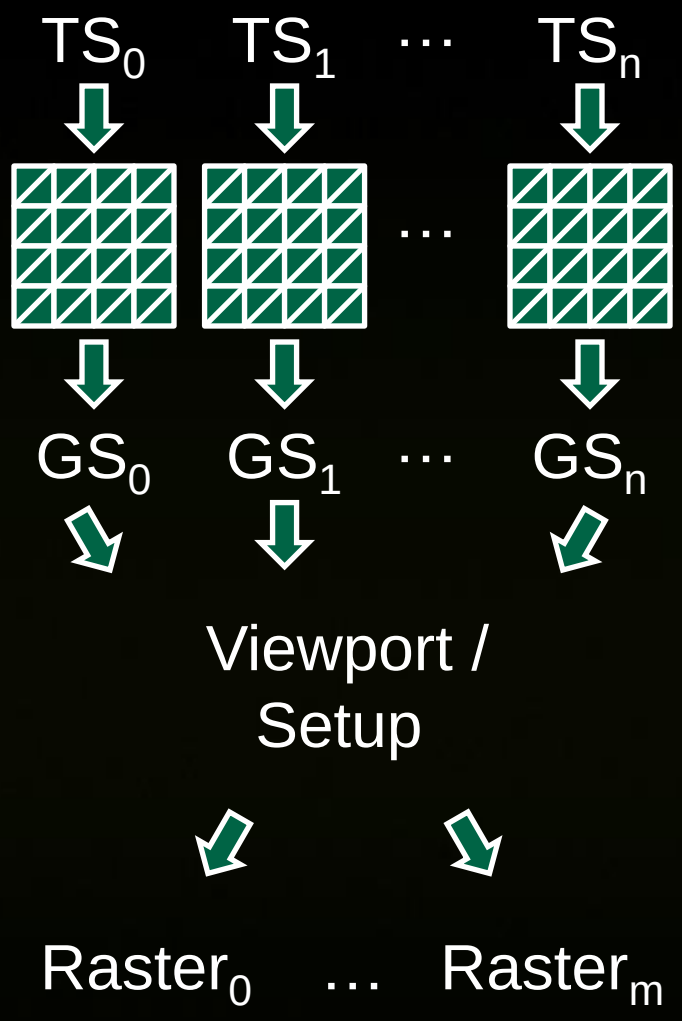
Problems With DX10 + Tess. (1)

- Data expansion in TS can lead to excessive buffering requirements after GS
  - Have to drain GS_0 before GS_1 to maintain API ordering



Problems With DX10 + Tess. (2)

- **Setup limited prim rate**
  - **Already a problem for CPU-generated geometry**
  - **Tessellation generates even more primitives**

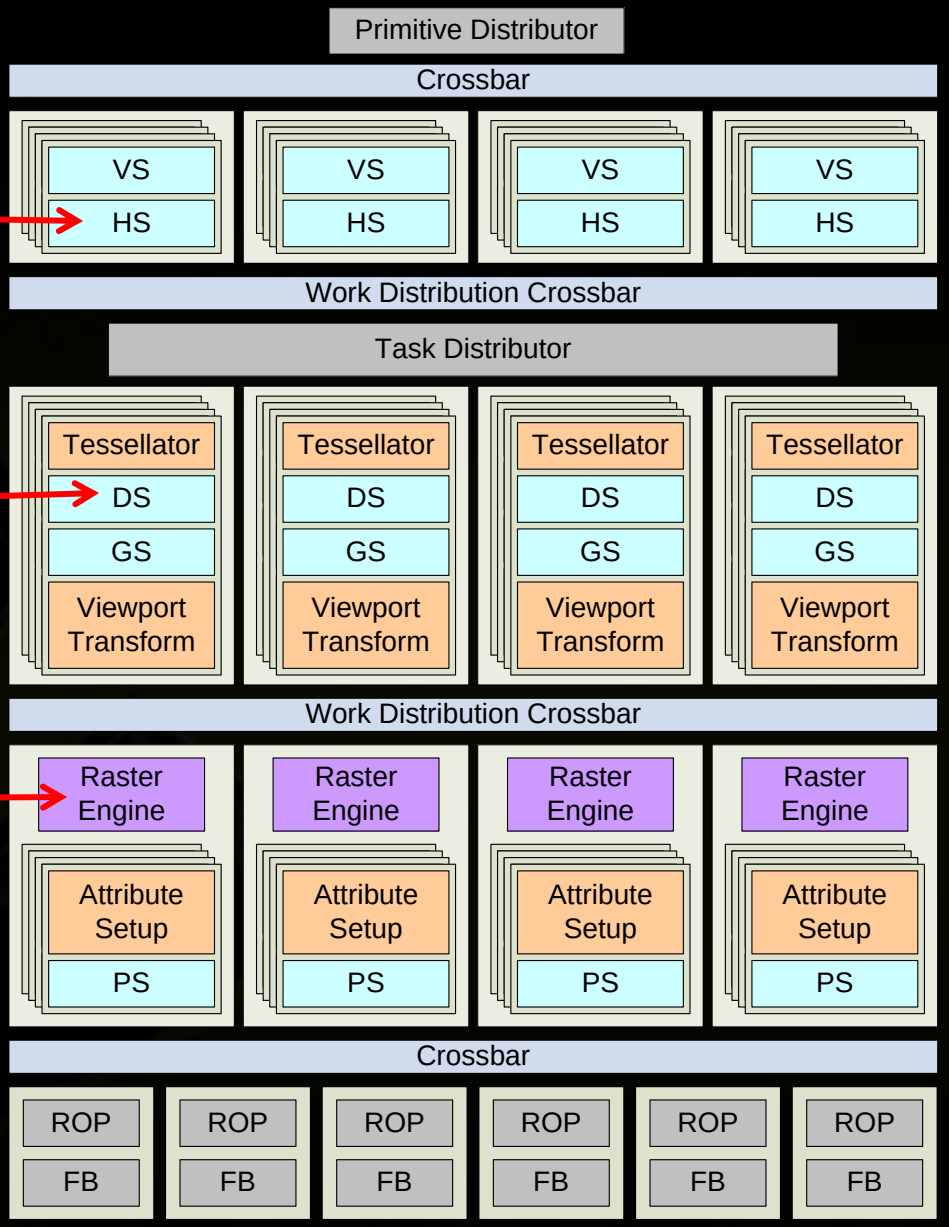


Fermi GF100 Logical Pipeline

Hull Shader

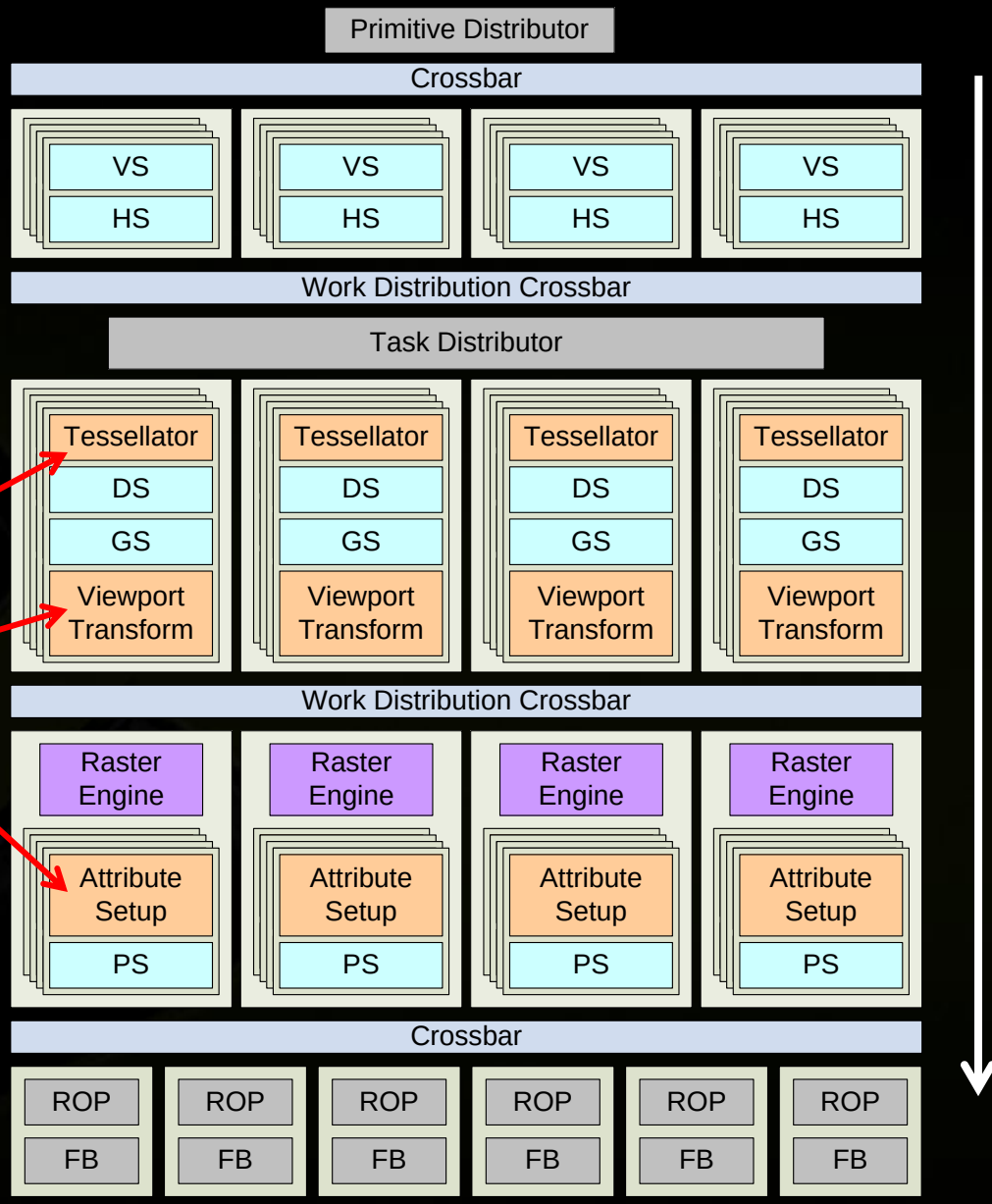
Domain Shader

Raster Engine
(includes Edge Setup,
Rasterizer, Zcull)



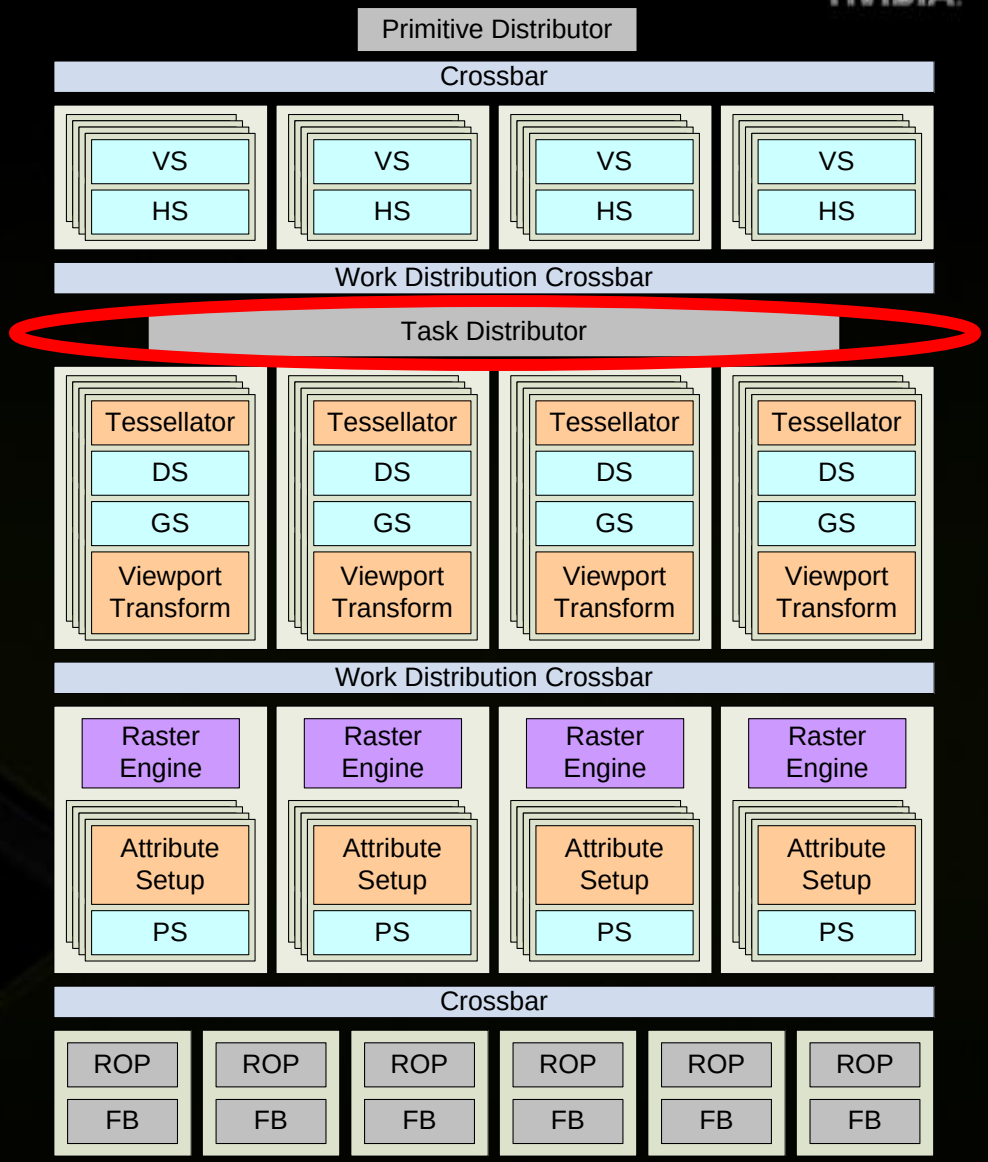
Fermi GF100 Logical Pipeline

Polymorph Engine
(includes Tessellator,
Viewport Transform,
and Attribute Setup)



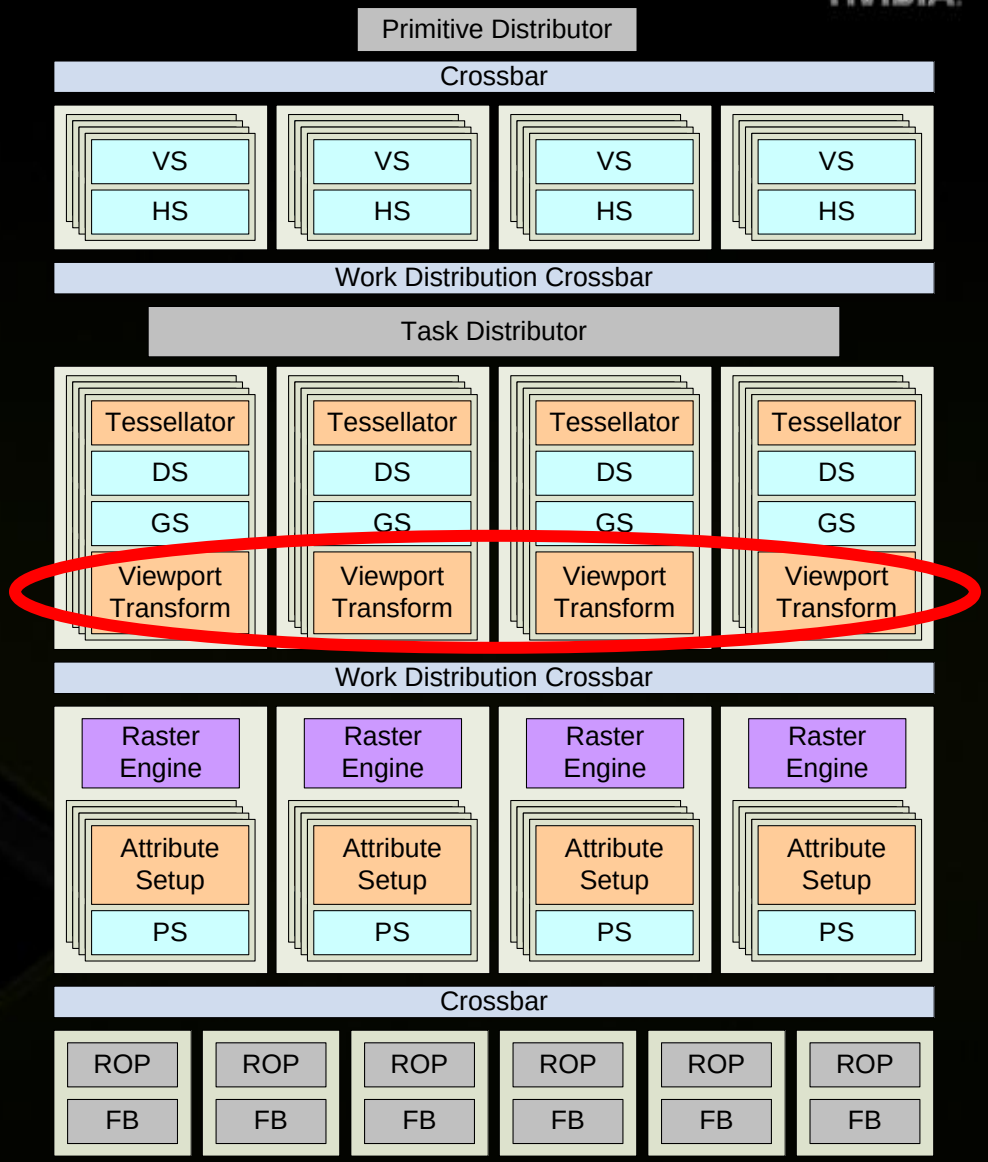
Fermi GF100 Tessellation Features 1

- Task Distributor
  - Task ~= Hull Shader output
    - Control points plus tessellation factor
    - Pre-expansion
  - Distributes tasks for subsequent geometry processing
    - Expand patch into primitives
    - Optional GS
  - Reduces buffering requirements



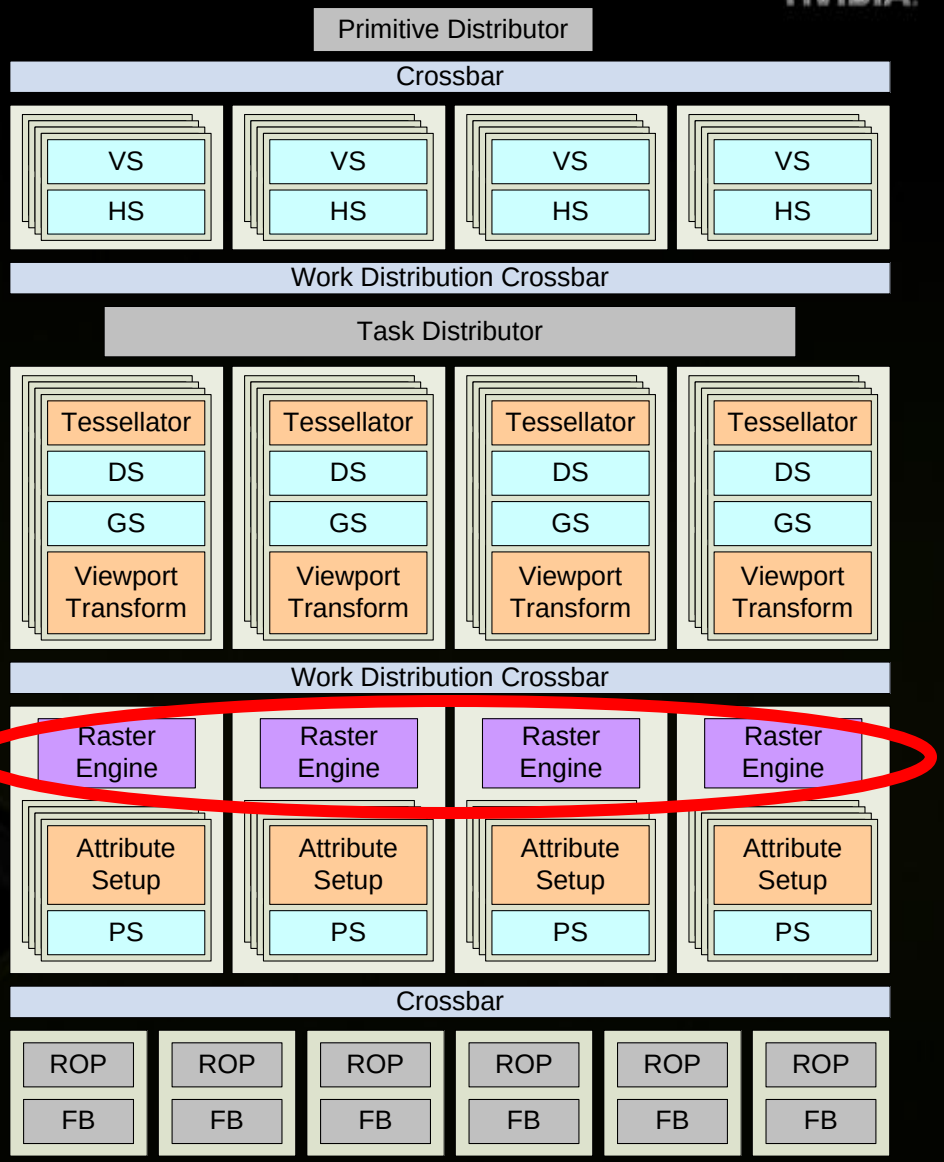
Fermi GF100 Tessellation Features 2

- **Parallel Viewport Transform**
  - **Includes clipping and culling**
  - **Eliminate serialization point**

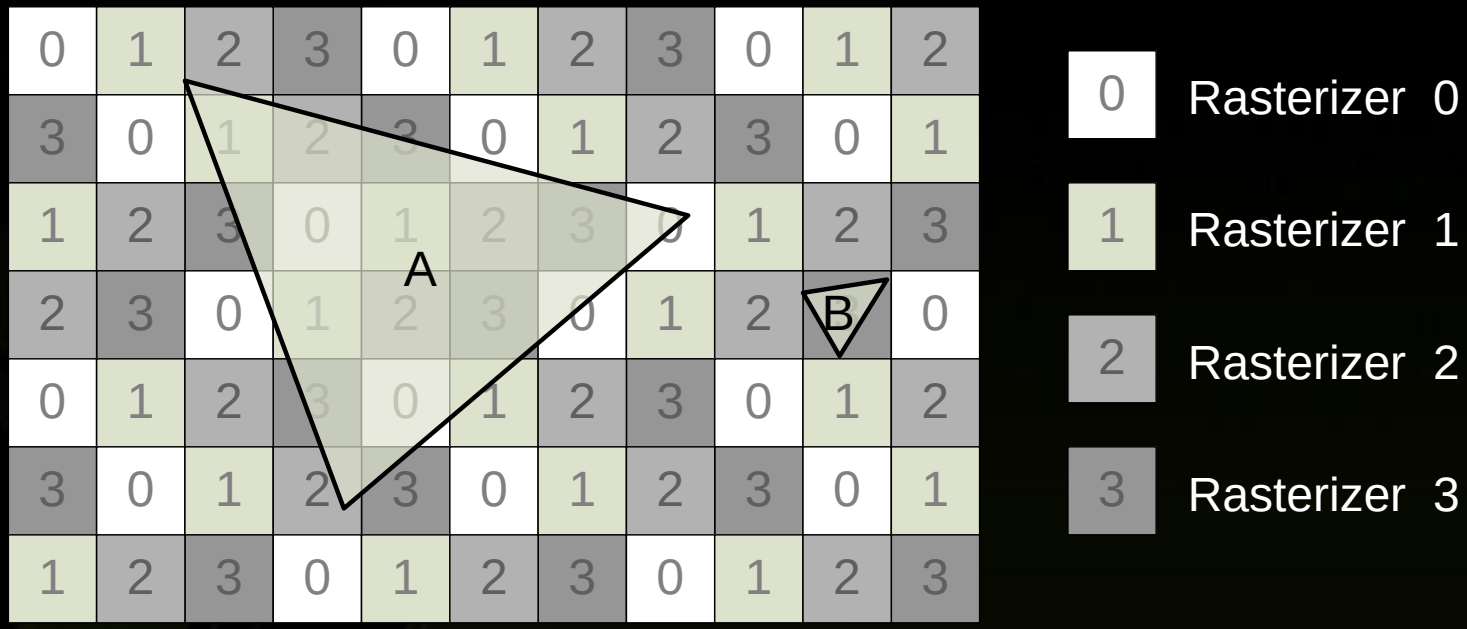


Fermi GF100 Tessellation Features 3

- Parallel Rasterization
  - Including Edge Setup, Rasterization, and Zcull
  - Improved primitive throughput
    - Multiple primitives per clock
  - Screen mapped for load balancing



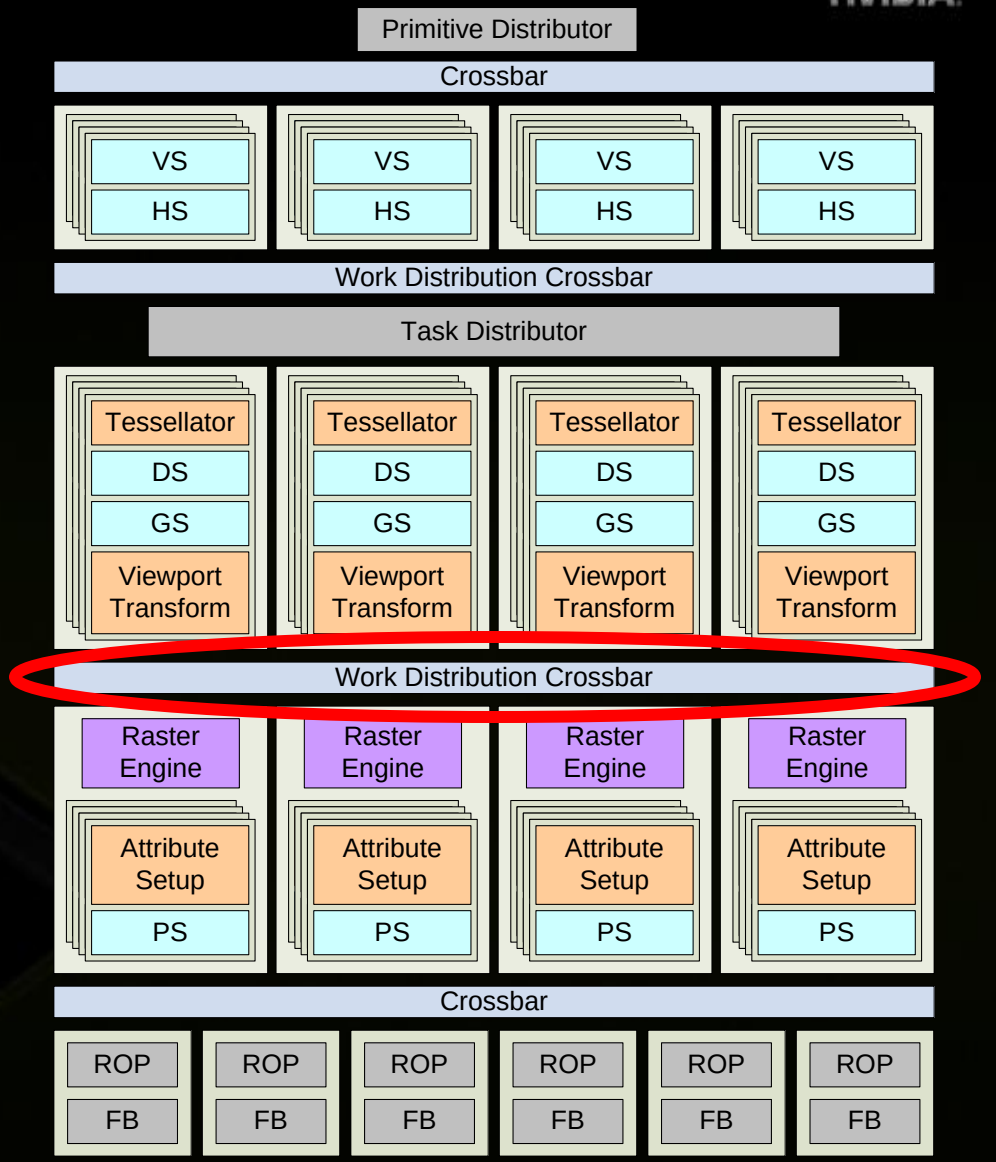
Screen Mapped Rasterization



- Each block is a tile of pixels
- Rasterizers uniquely own pixel tiles
- Note: small primitives can overlap multiple tiles

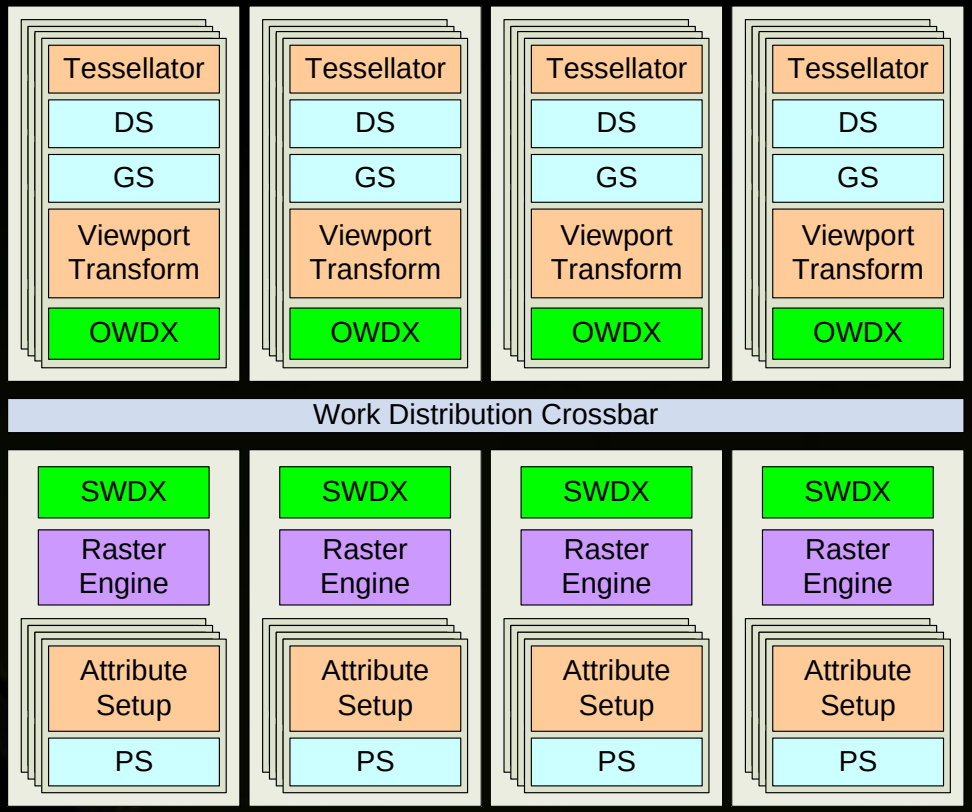
Challenge: Maintaining API Order

- Parallelization is easy
  - Maintaining API ordering is a challenge
- Work Distribution Crossbar (WDX)
  - Similar to Pomegranate [Eldridge et al. 2000]



Work Distribution Crossbar

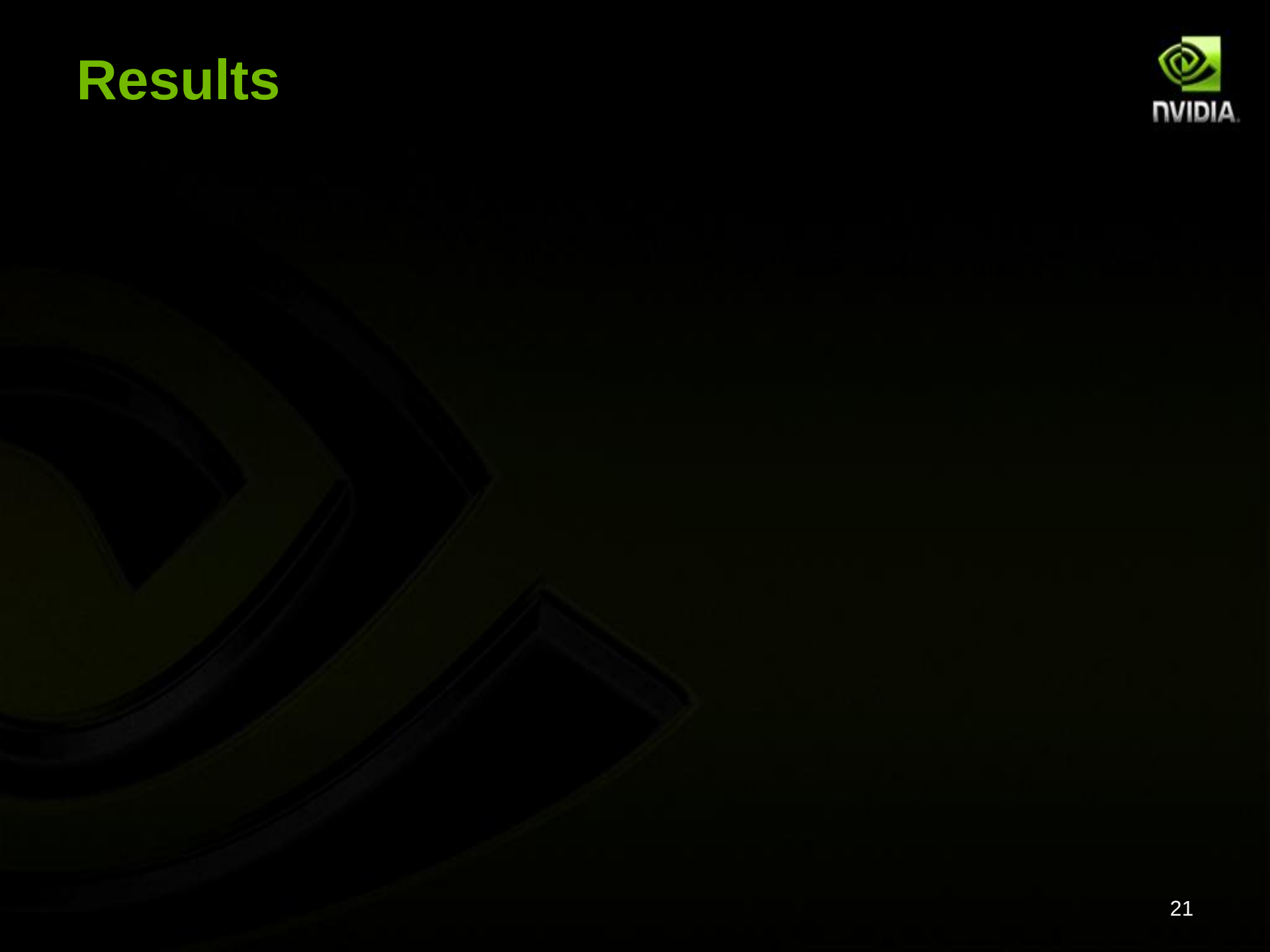
- OWDX
  - Uses primitive bounding box to determine which rasterizers get which primitives
- SWDX
  - reconstructs API order
- Rasterizers uniquely own pixels
  - No subsequent sorting for ordering required



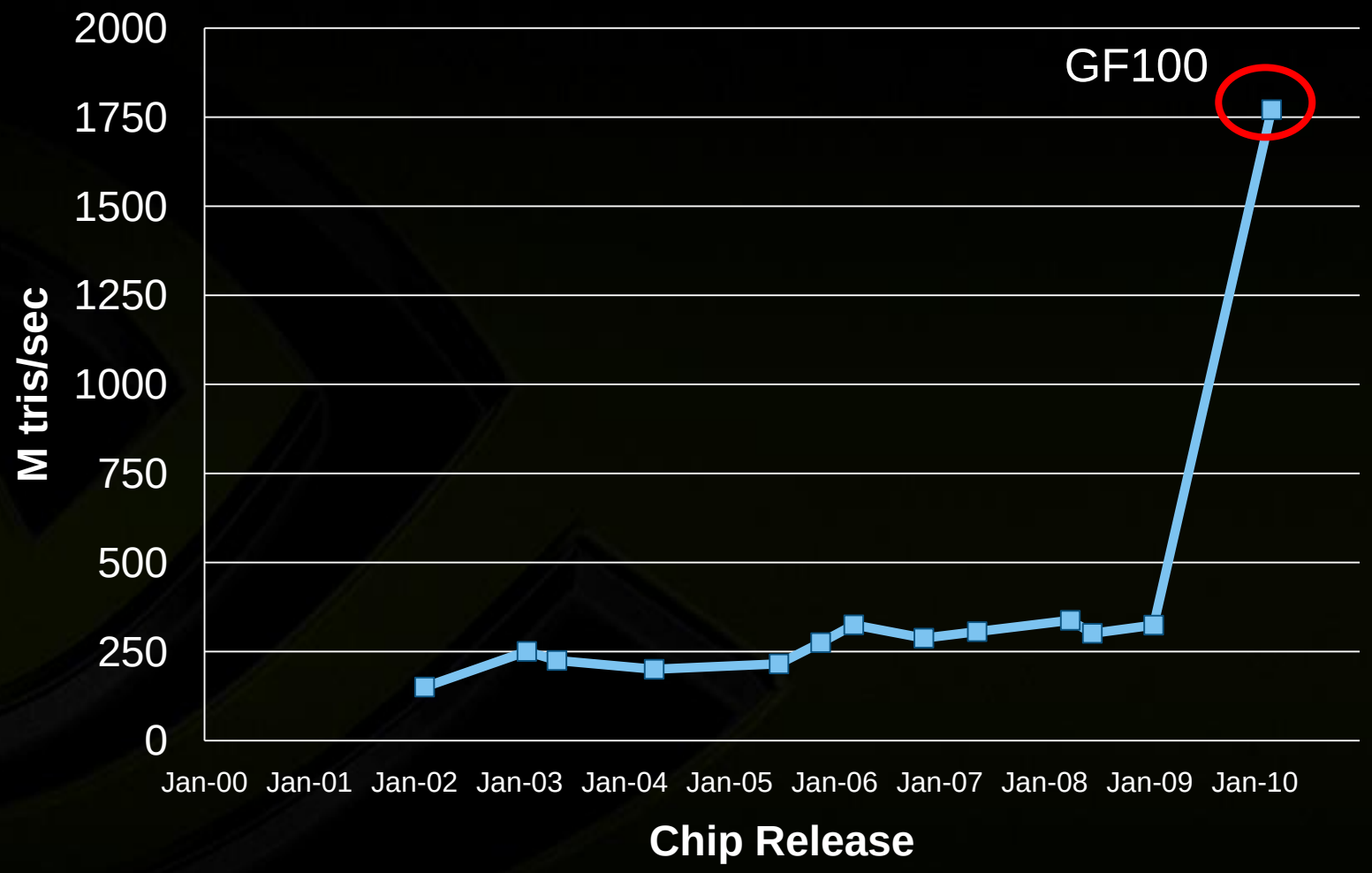
Handling Load Imbalance

- **Load imbalance is a concern for all region-based architectures because they are non adaptive**
  - **Static imbalance**
    - **Different screen regions get differing amounts of work**
    - **Mitigated by relatively small pixel tiles**
  - **Dynamic imbalance**
    - **Lots of tiny primitives in same screen region at the same time**
    - **Mitigated by buffering**
      - We can only buffer so much…

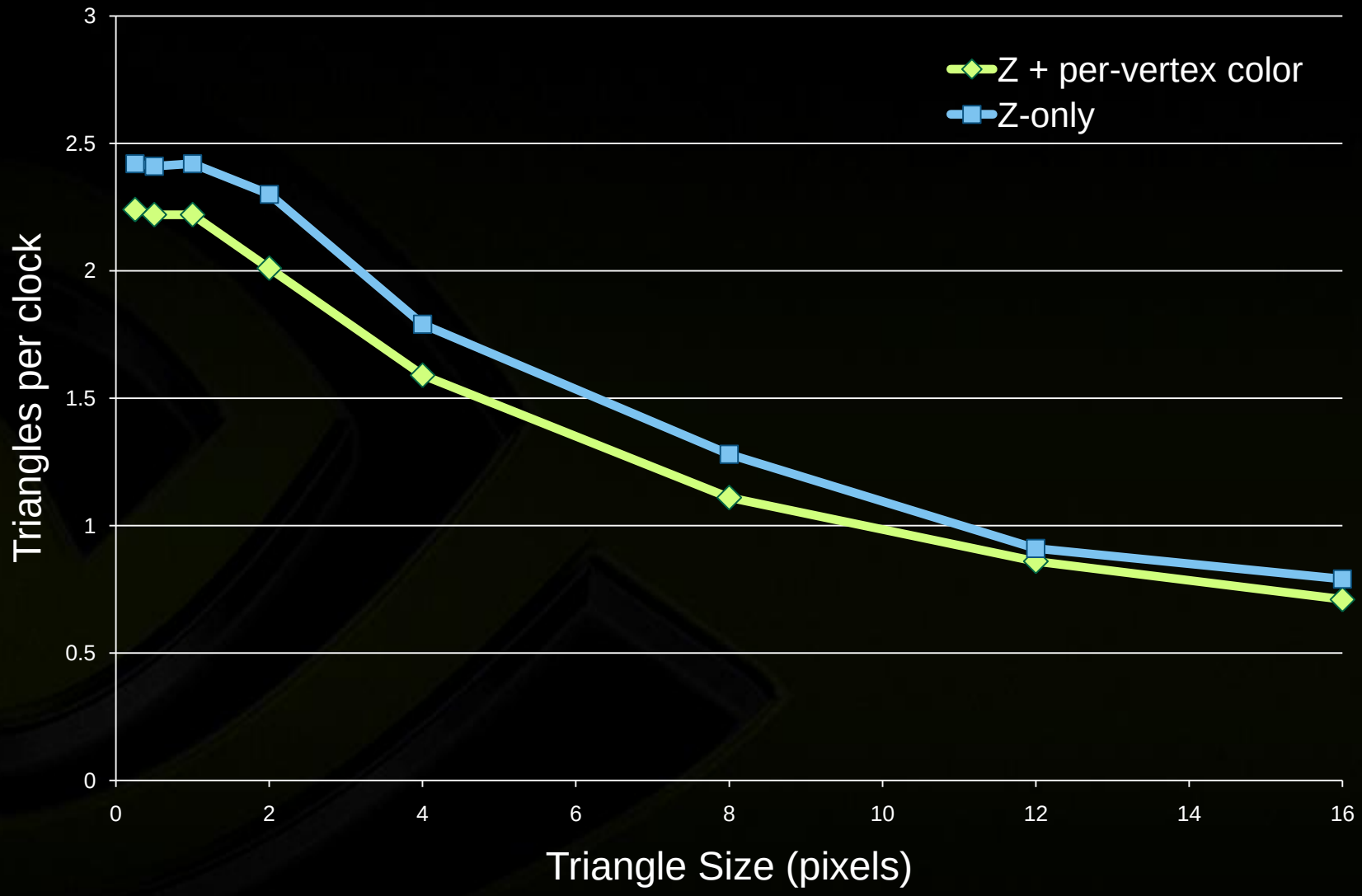
Results



Historical Triangle Rate



Results – Synthetic Performance Test



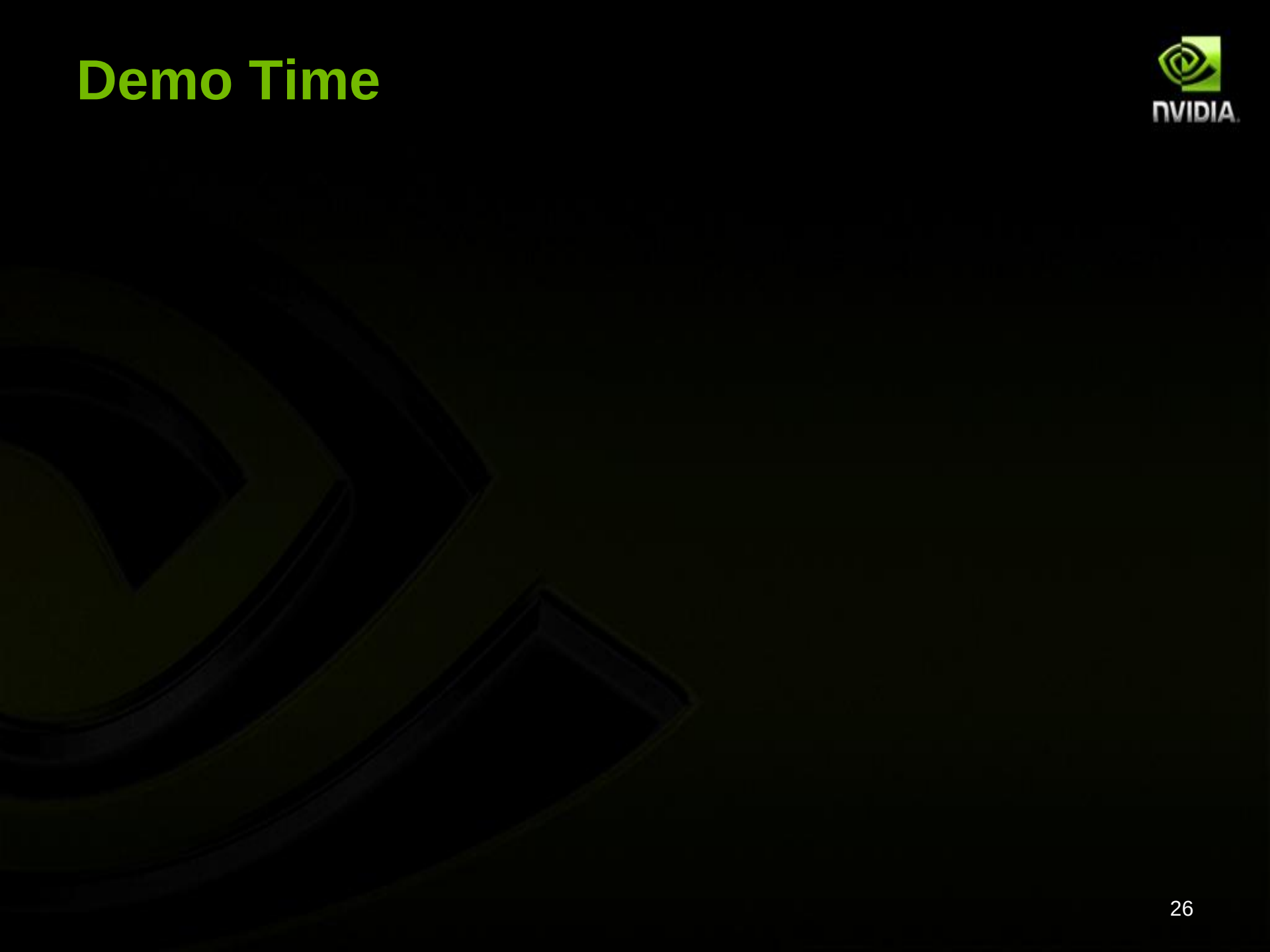
Results – Island Demo

Frame average of 1.94 triangles per clock

Results – Unigine Heaven

2.34 triangles per clock in highly tessellated areas

Demo Time



Summary

- Tessellation
  - Increases geometric realism without burdening the CPU
- Fermi GF100 is designed for tessellation
  - Eliminates serialization points
  - Dramatically improved primitive throughput
  - Next-generation visual fidelity

Acknowledgements

- Steve Molnar
- Ziyad Hakura
- Andreas Dietrich (demo help)
- Entire Fermi GF100 team

Questions?