

# Architektur und Programmierung von Grafik- und Koprozessoren

## Die Grafik Pipeline

Stefan Zellmann

Lehrstuhl für Informatik, Universität zu Köln

SS2019

# GPU Cache Hierarchien

# GPU Cache Hierarchien

- ▶ Unterschiedlich je nach Architektur (Hersteller / Chipset).
- ▶ Nicht für alle Architekturen bekannt.
- ▶ Referenzen:
  - ▶ **AMD:** GDC2016: Right On Queue, Stephan Hodes, Dan Baker, Dave Oldcorn.
  - ▶ **Nvidia:** Life of a triangle - NVIDIA's logical pipeline

# GPU Cache Hierarchien

Beispiel AMD Graphics Core Next Architektur (2012).

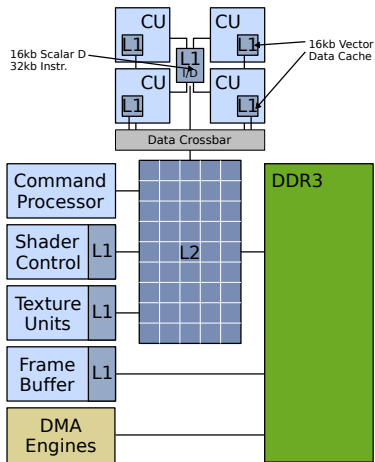
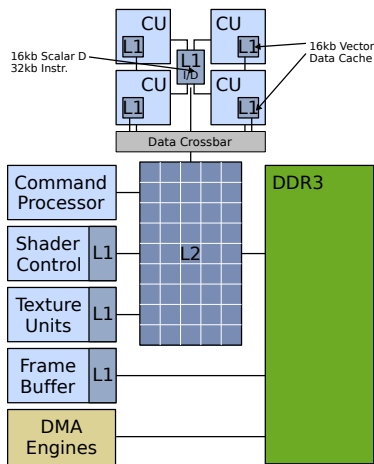


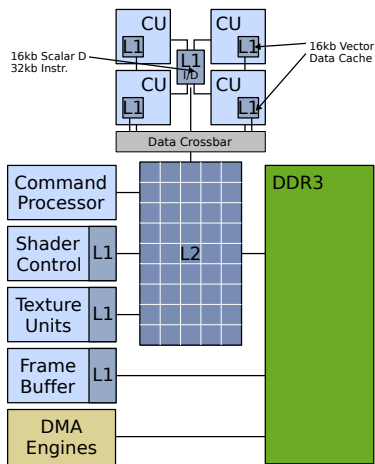
Abbildung: vgl. White Paper AMD Graphics Cores Next (GCN) Architecture & GDC2016: Right on Queue.

# GPU Cache Hierarchien



- ▶ Exemplarisch an GCN. Keine symmetrische Cache Hierarchie wie bei CPU NUMA.
- ▶ Compute Units (CU) (a.k.a. Core a.k.a. SM). Lokaler *und* geteilter L1.
- ▶ Manche funktionale Einheiten haben L1, der direkt an Speicher angebunden.
- ▶ DMA Engines umgehen Cache vollständig.

# GPU Cache Hierarchien



- ▶ In Wirklichkeit noch komplizierter: mehrere Texture Units und ROPs.
- ▶ Nicht ein großer L2 Cache.
- ▶ Cache Kohärenz - L1 Caches der CUs werden durch Flushes nach L2 kohärent gehalten.
- ▶ Synchronisation durch komplexe Speicherhierarchie noch schwieriger ⇒ moderne APIs: Barrier Synchronisation Teil der *Applikation*, nicht Treiber.

# GPU Cache Hierarchien

Pipeline Stages / Raster Engines etc. kommunizieren nur über Caches. Nvidia: *Crossbar* koordiniert die Kommunikation. Framebuffer in VRAM.

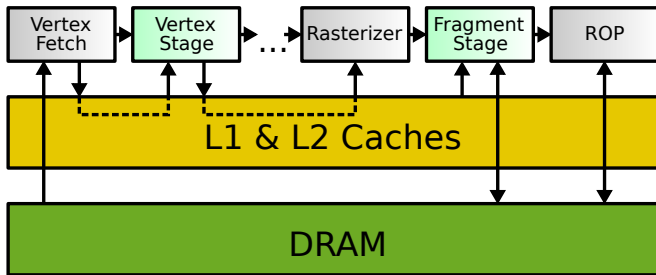


Abbildung: vgl: Life of a triangle - NVIDIA's logical pipeline

# Sort-Last vs. Sort-Middle Architekturen



# Sort-Middle Architectures

- ▶ Auch: *Tile-Based Rendering* (nicht zu verwechseln mit Kacheln bei Raster Engines!)
- ▶ Desktop GPUs implementieren traditionell Immediate-Mode Rendering: Verarbeitung orientiert sich an Reihenfolge, in der Dreiecke submittiert wurden (nicht zu verwechseln mit Immediate-Mode APIs!)
- ▶ Tile-Based Rendering hauptsächlich auf Mobile Architekturen.
  - ▶ Mobile GPUs häufig limitiert durch *externe* Speicherbandbreite
  - ▶ Mobile Architekturen konzentrieren sich auf minimalen Energieverbrauch.
  - ▶ Nvidia setzt Tile-Based Rendering seit Maxwell auch für Desktop GPUs ein (vgl. <https://www.realworldtech.com/tile-based-rasterization-nvidia-gpus/> und [https://www.youtube.com/watch?time\\_continue=324&v=Nc6R1hwXhL8](https://www.youtube.com/watch?time_continue=324&v=Nc6R1hwXhL8))

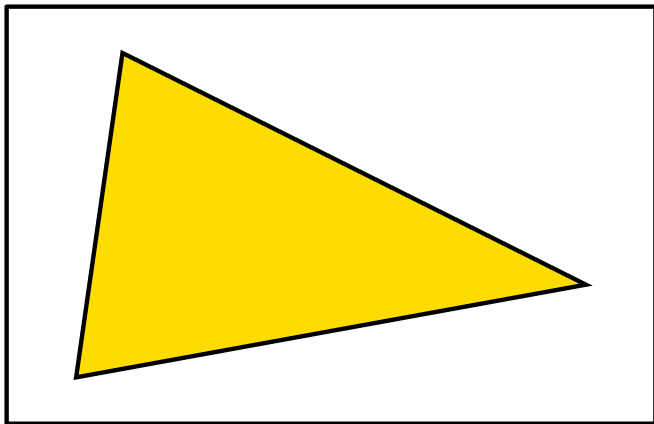
# Sort-Middle Architectures

## Immediate-Mode Rendering



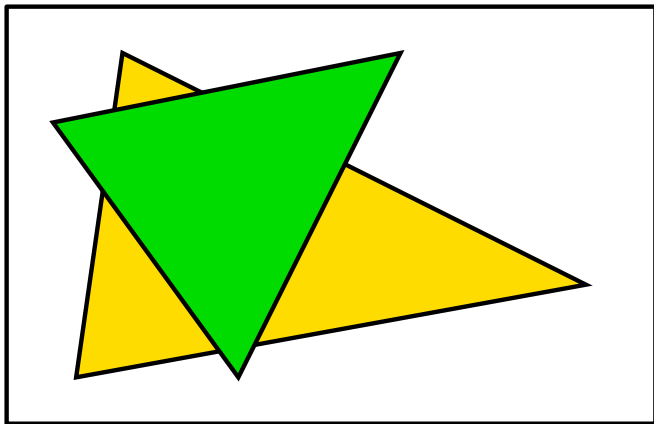
# Sort-Middle Architectures

## Immediate-Mode Rendering



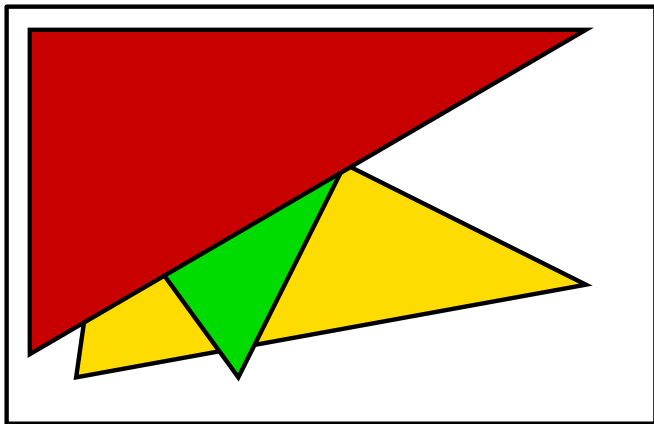
# Sort-Middle Architectures

## Immediate-Mode Rendering



# Sort-Middle Architectures

## Immediate-Mode Rendering



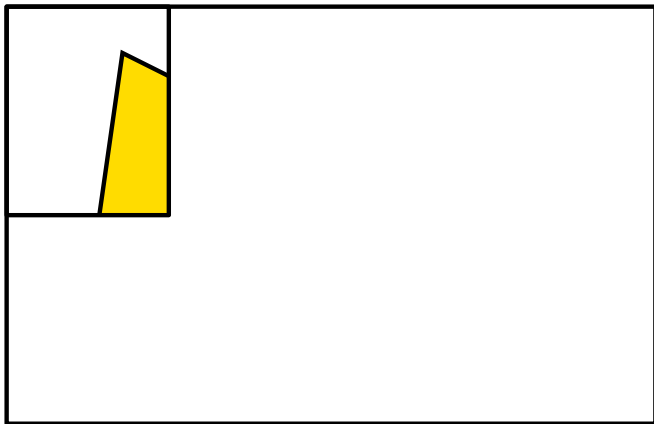
# Sort-Middle Architectures

## Tile-Based Rendering



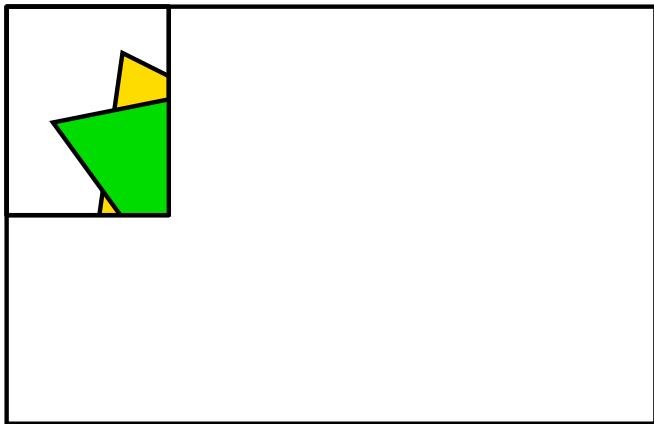
# Sort-Middle Architectures

## Tile-Based Rendering



# Sort-Middle Architectures

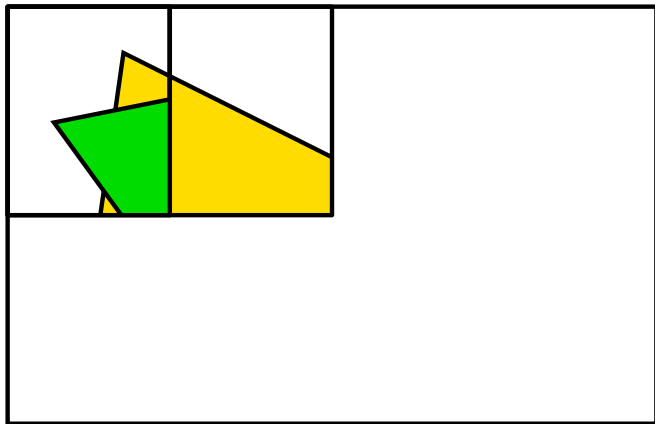
## Tile-Based Rendering





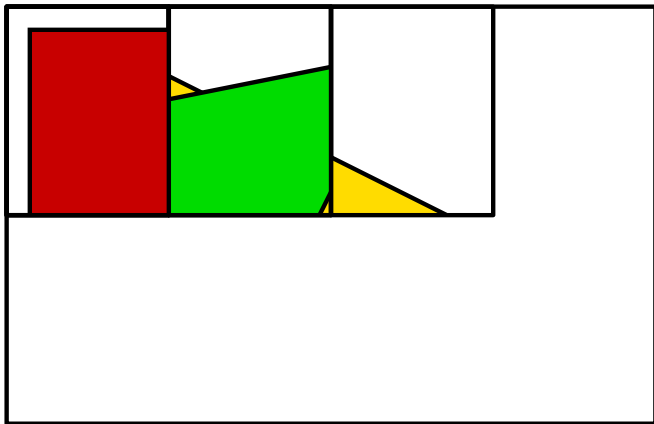
# Sort-Middle Architectures

## Tile-Based Rendering



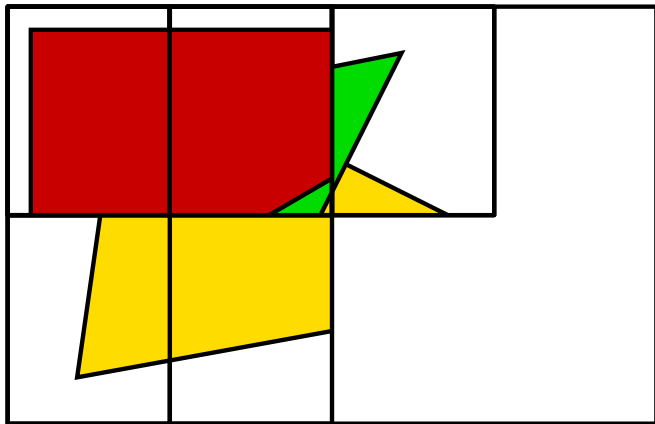
# Sort-Middle Architectures

## Tile-Based Rendering



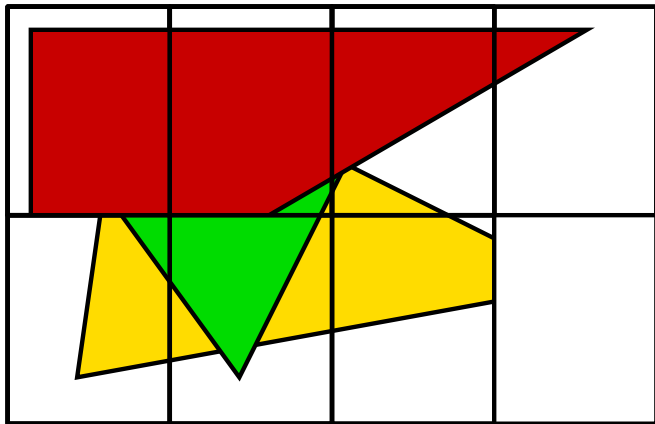
# Sort-Middle Architectures

## Tile-Based Rendering



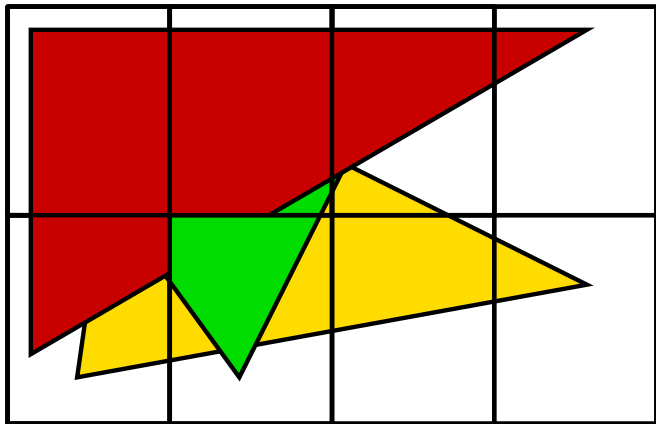
# Sort-Middle Architectures

## Tile-Based Rendering



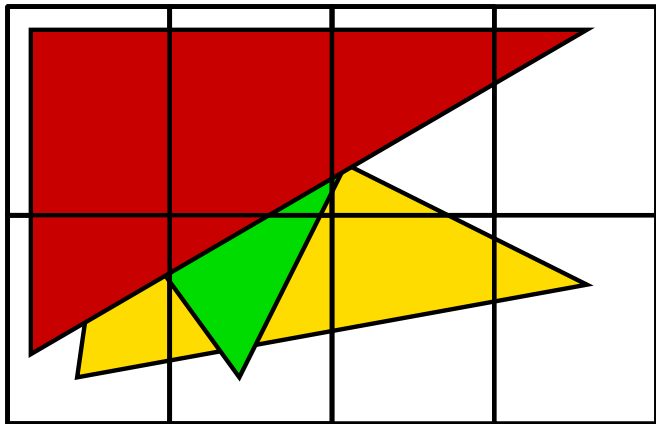
# Sort-Middle Architectures

## Tile-Based Rendering



# Sort-Middle Architectures

## Tile-Based Rendering



# Sort-Middle Architectures

- ▶ Tile-Based Rendering: Sort-Middle mit vergleichsweise großen Kacheln (z. B.  $128 \times 128$  oder größer).
- ▶ Erfordert weiteren Schritt in der Grafik Pipeline: *Binning*.
  - ▶ Vor Scan Konvertierung muss ermittelt werden, welche Dreiecke welche Kacheln überdecken  $\Rightarrow$  Latenz: es muss gewartet werden, bis alle Dreiecke submittiert wurden.

# Sort-Middle Architectures

- ▶ Vorteile:
  - ▶ Reduziert Speicherbandbreitebedarf - wegen räumlicher Lokalität innerhalb Kachel wird Cache besser ausgenutzt.
  - ▶ Weniger Speicherzugriffe  $\Rightarrow$  geringerer Energieverbrauch (vgl. Vorlesung zu dynamischem Speicher).
- ▶ Nachteile:
  - ▶ Latenz - Scan Konvertierung kann erst beginnen, wenn alle Dreiecke submittiert wurden.
  - ▶ Lastimbilanzen im Fall, dass Dreiecke das Anzeigerechteck ungleichmäßig überlappen.



# Recap

- ▶ GPUs exponieren Parallelismus entlang der gesamten Grafik Pipeline:
  - ▶ Asynchrones Host Interface.
  - ▶ Parallele Vertex Phase.
  - ▶ Mehrere Raster Engines.
  - ▶ Parallele Fragment Phase.
- ▶ Wichtiges Charakteristikum von Speicheroperationen auf GPUs:
  - ▶ Hohe Bandbreite, hohe Latenz.
  - ▶ Daher Kompressionsverfahren, Cache Hierarchien etc. integraler Bestandteil von GPUs.
- ▶ Grafik Pipeline in Teilen als Fixed-Function Pipeline (Scan Konvertierung, Render Output) und in Teilen als frei programmierbare Pipeline (Vertices, Fragmente, Geometrie) implementiert.

# Literaturempfehlungen

- ▶ Matthew Eldridge, Homan Igehy, Pat Hanrahan:  
Pomegranate: A Fully Scalable Graphics Architecture,  
Siggraph 2000.
- ▶ Diverse “Grey Literature” (vgl. IHV Whitepapers und sonstige  
Online-Quellen in Folien). Außerdem:
  - ▶ Fabian Giesens Blog “A trip through the graphics pipeline”:  
[https://fgiesen.wordpress.com/2011/07/09/  
a-trip-through-the-graphics-pipeline-2011-index/](https://fgiesen.wordpress.com/2011/07/09/a-trip-through-the-graphics-pipeline-2011-index/)
  - ▶ GPU Framebuffer Memory: Understanding Tiling (Samsung):  
<https://developer.samsung.com/game/gpu-framebuffer>