Harnessing The Power Of Multiple GPUs

Games Developer Conference 2008

Holger Gruen, holger.gruen@amd.com

Material:

Holger Gruen,

Ignacio Llamas, illamas@nvidia.com

Jon Story, jon.story@amd.com

Agenda

- Why MGPU?
- Driver Considerations
- Programming for MGPU
- Common Pitfalls & Solutions

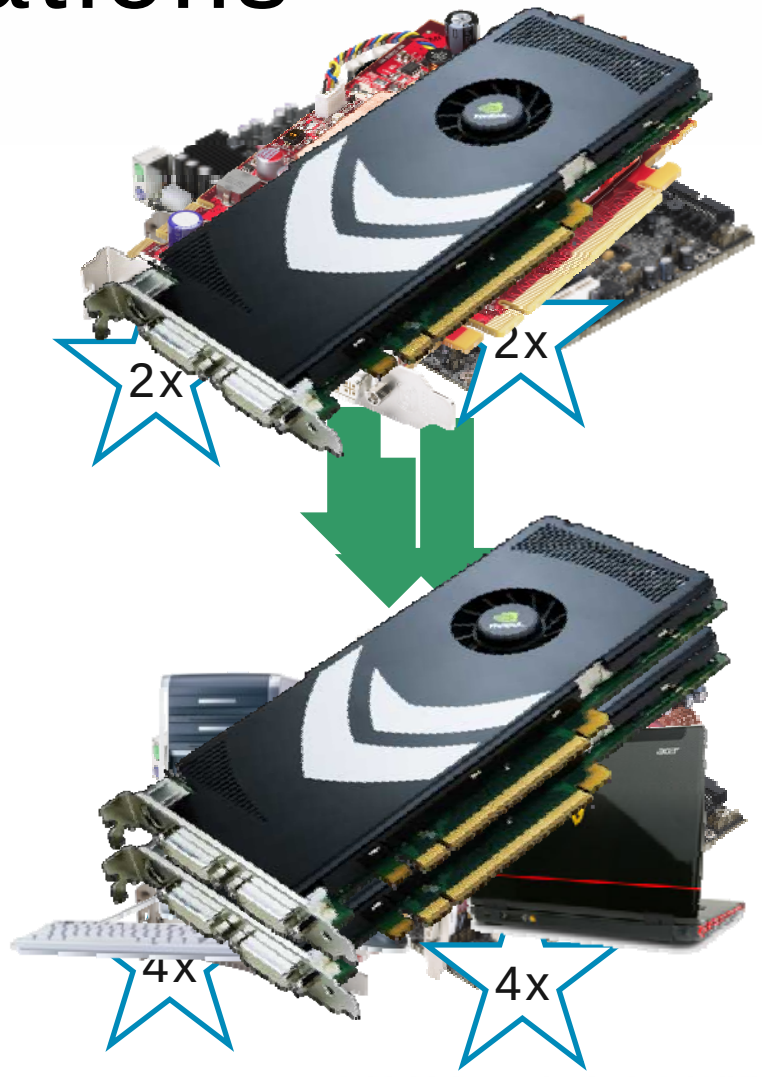
Why MGPU?

Why MGPU?

- Many apps are GPU limited at high resolutions
  - Generally CPU limited at lower resolution
- High res monitors have become affordable
  - Consumer expectations have risen
- MGPUs can dramatically increase performance
  - Especially at higher screen resolutions
- Next gen performance on today‘s HW
  - Prototype your next engine

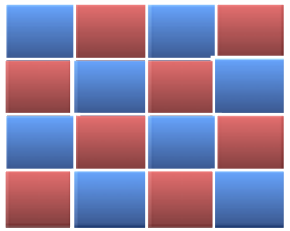
MGPU configurations

- Multiple Boards
- Multiple GPUs per Board
- Hybrid MGPU

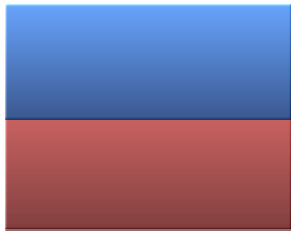


MGPU Rendering Modes

- SuperTiling (AMD)
  - Screen divided in a grid
  - GPUs take alternate tiles



- Split Frame Rendering / Scissor
  - Screen is divided between GPUs
  - Dynamic load balancing



- Alternate Frame Rendering
  - GPUs take alternate frames
  - Highest performing mode

Driver Considerations

Driver modes for MGPU

- AFR compatible mode
  - Default – driver works around problems
- App Profile mode
  - Profile fully defines driver behaviour
- Forced AFR - speed test mode
  - AFR-FriendlyD3D.exe – no work arounds

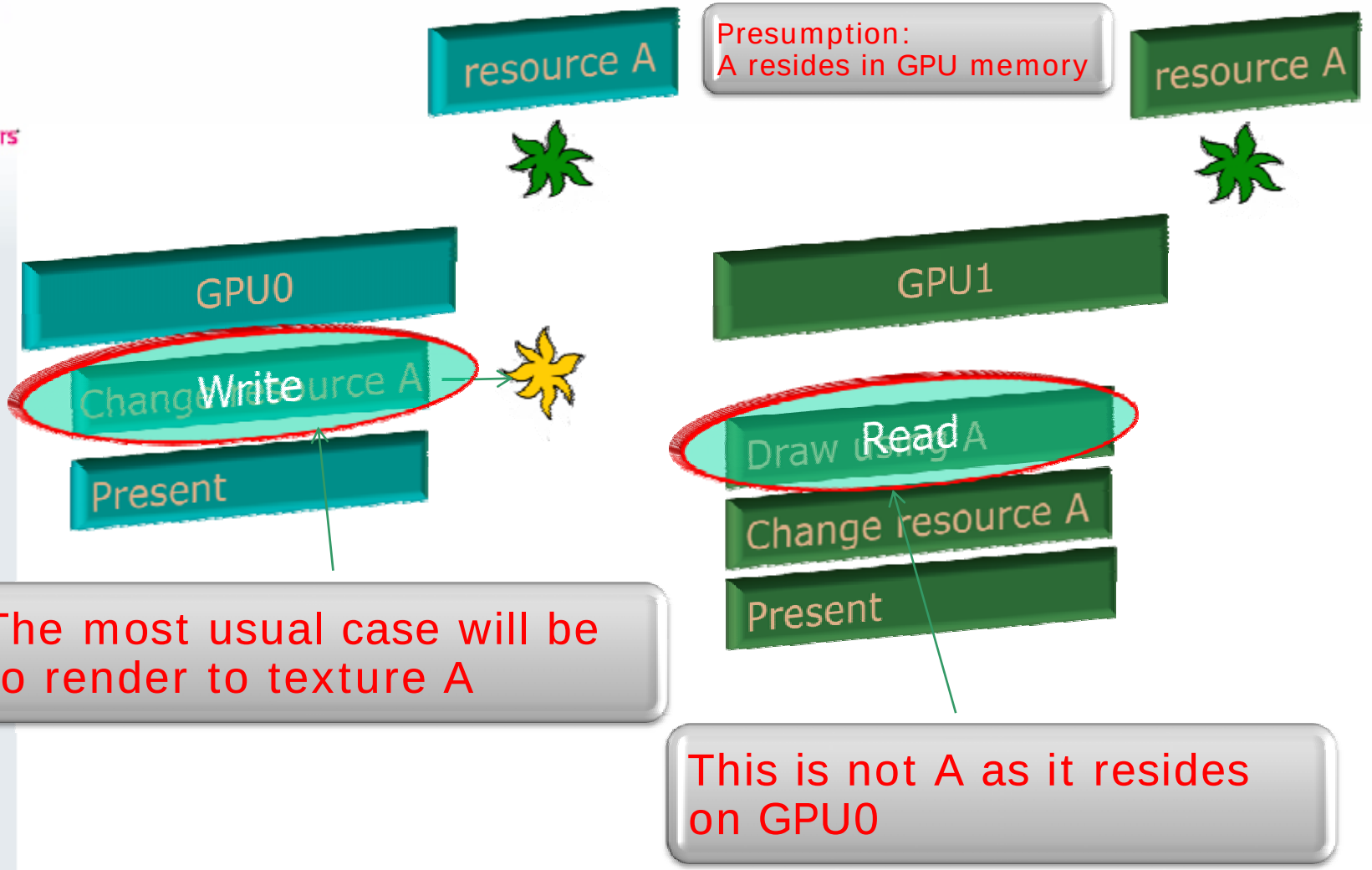
Programming for MGPU

Programming for MGPU

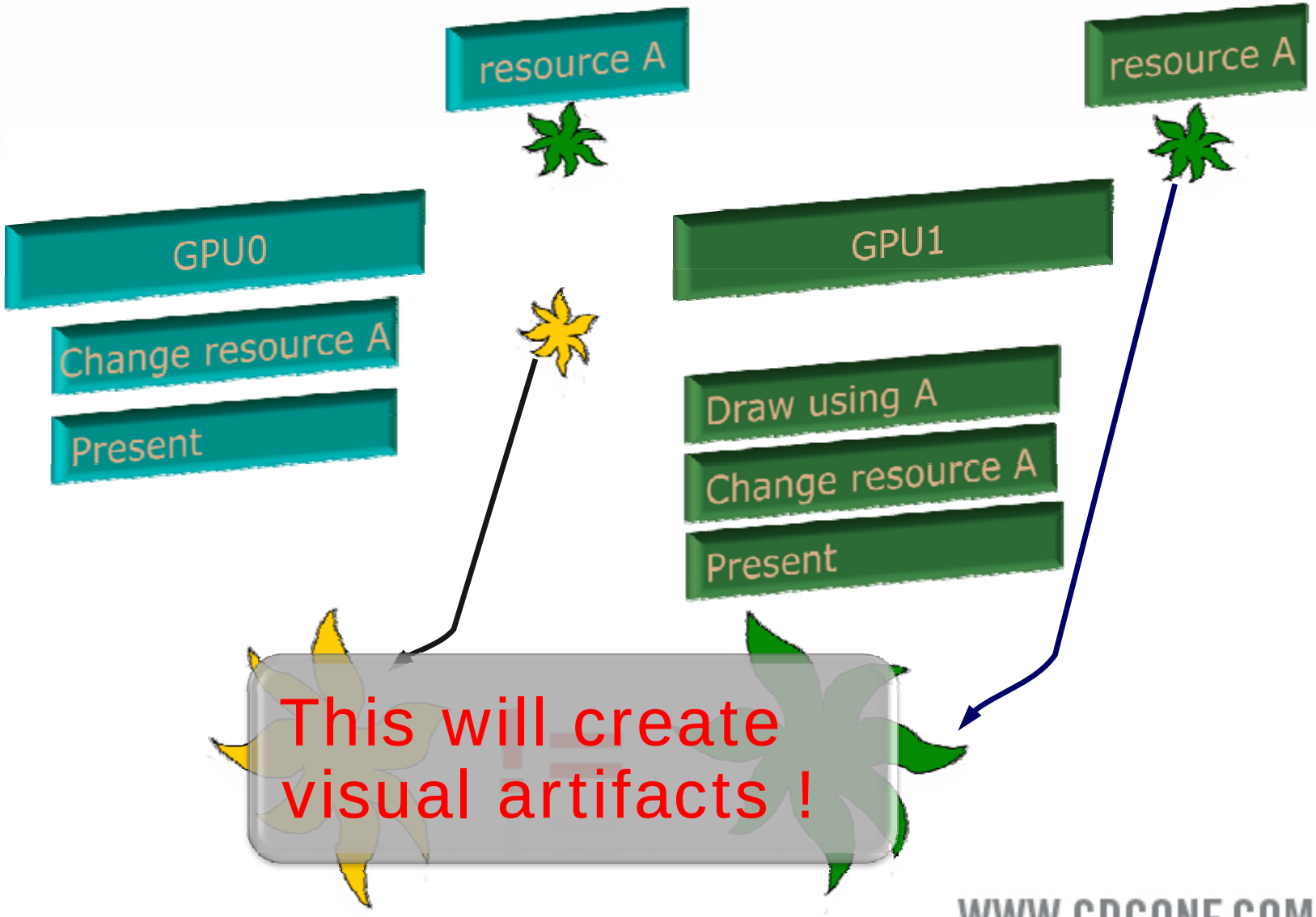
- MGPU no shared memory architecture
  - Apps need to behave well to scale
- Use AMD / NVIDIA libraries
  - Allow to query MGPU topology
- Know what GPU/s are rendering the current frame
  - Critical to adapt aplpication behavior

Common Pitfalls & Solutions

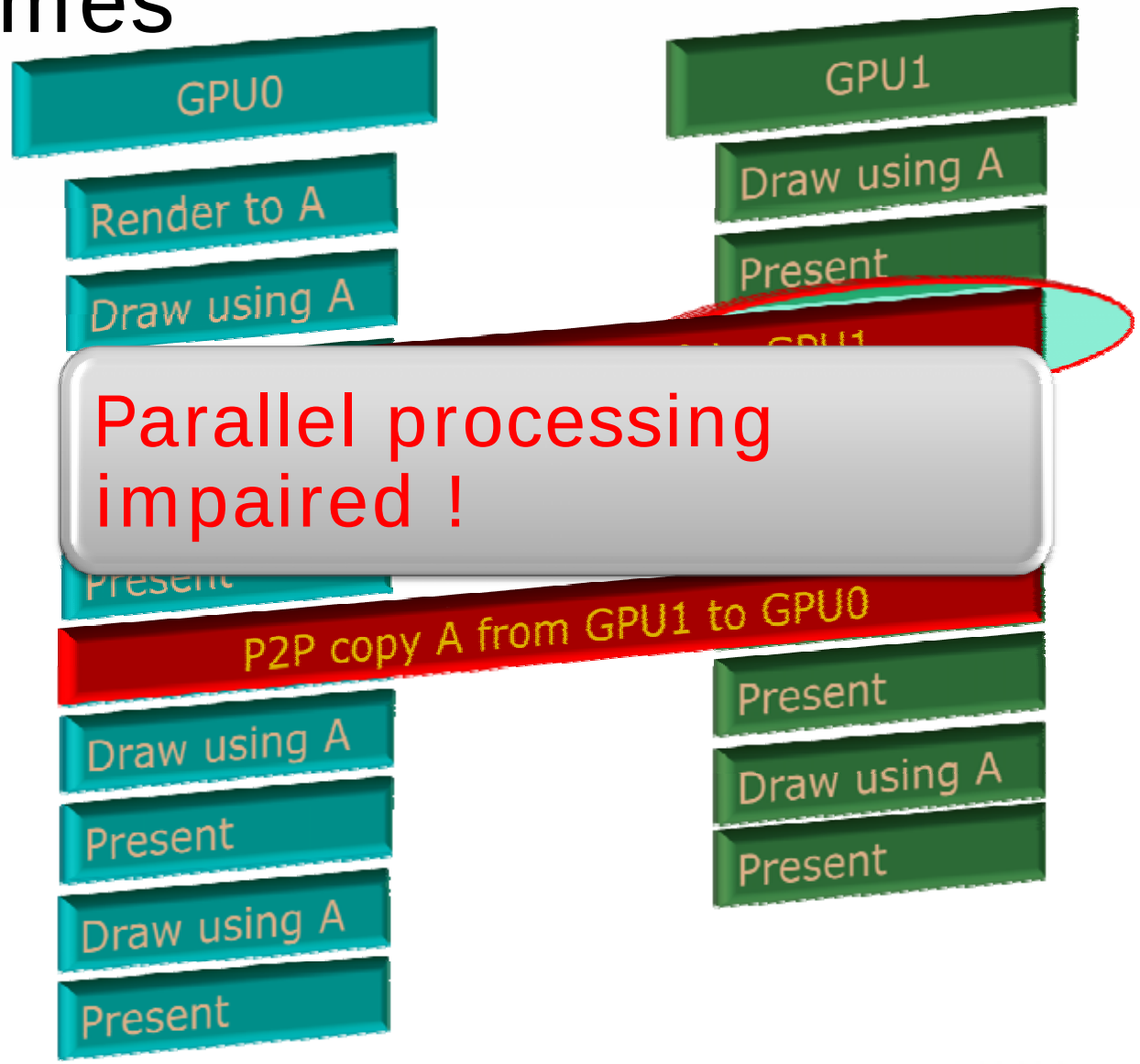
Pitfalls: Dependencies between frames



Pitfalls: Dependencies between frames

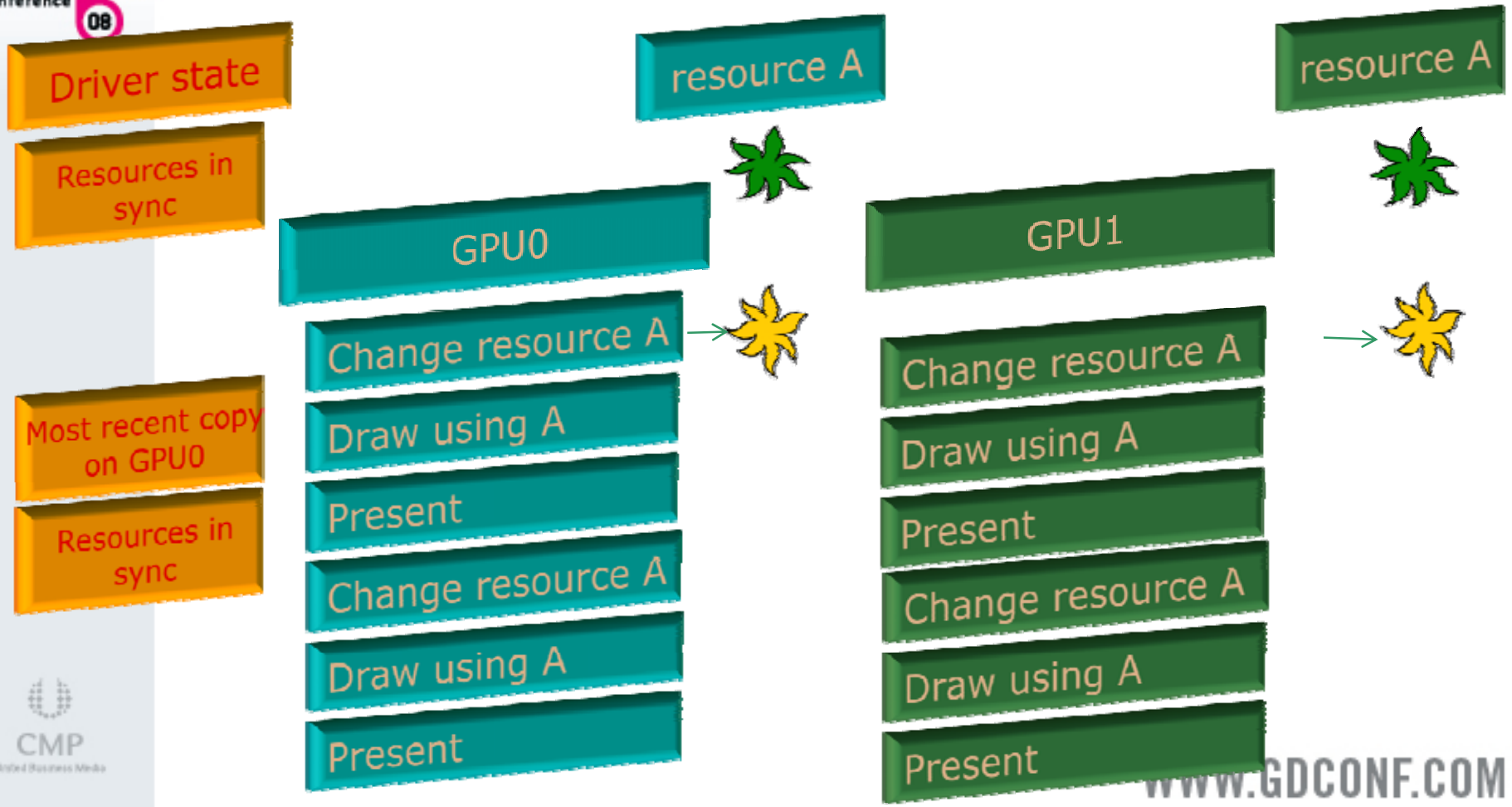


Pitfalls: Dependencies between frames

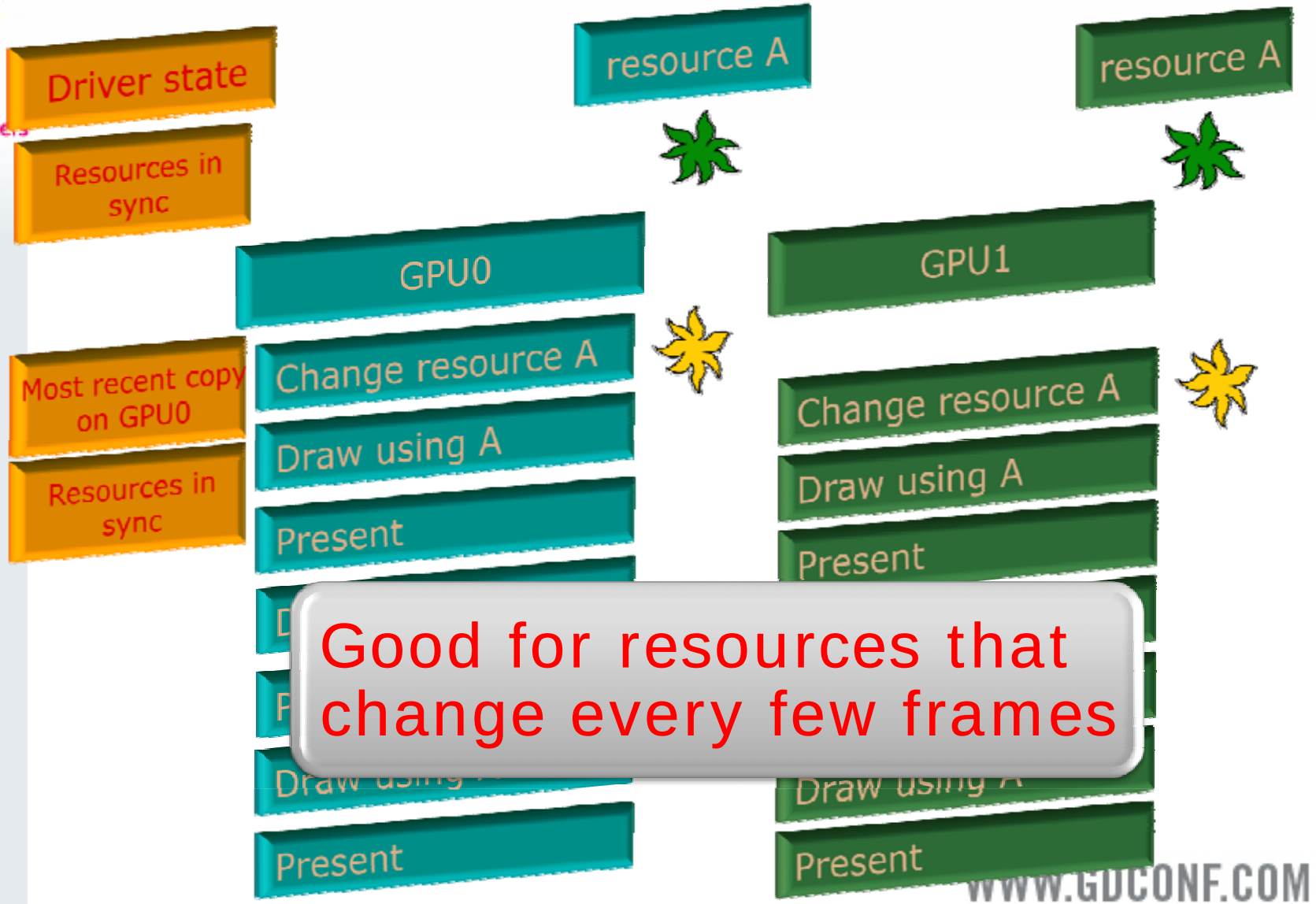


Solutions: Change resource every frame before using it

Good for data that changes every frame



Solutions: Repeat change on each GPU

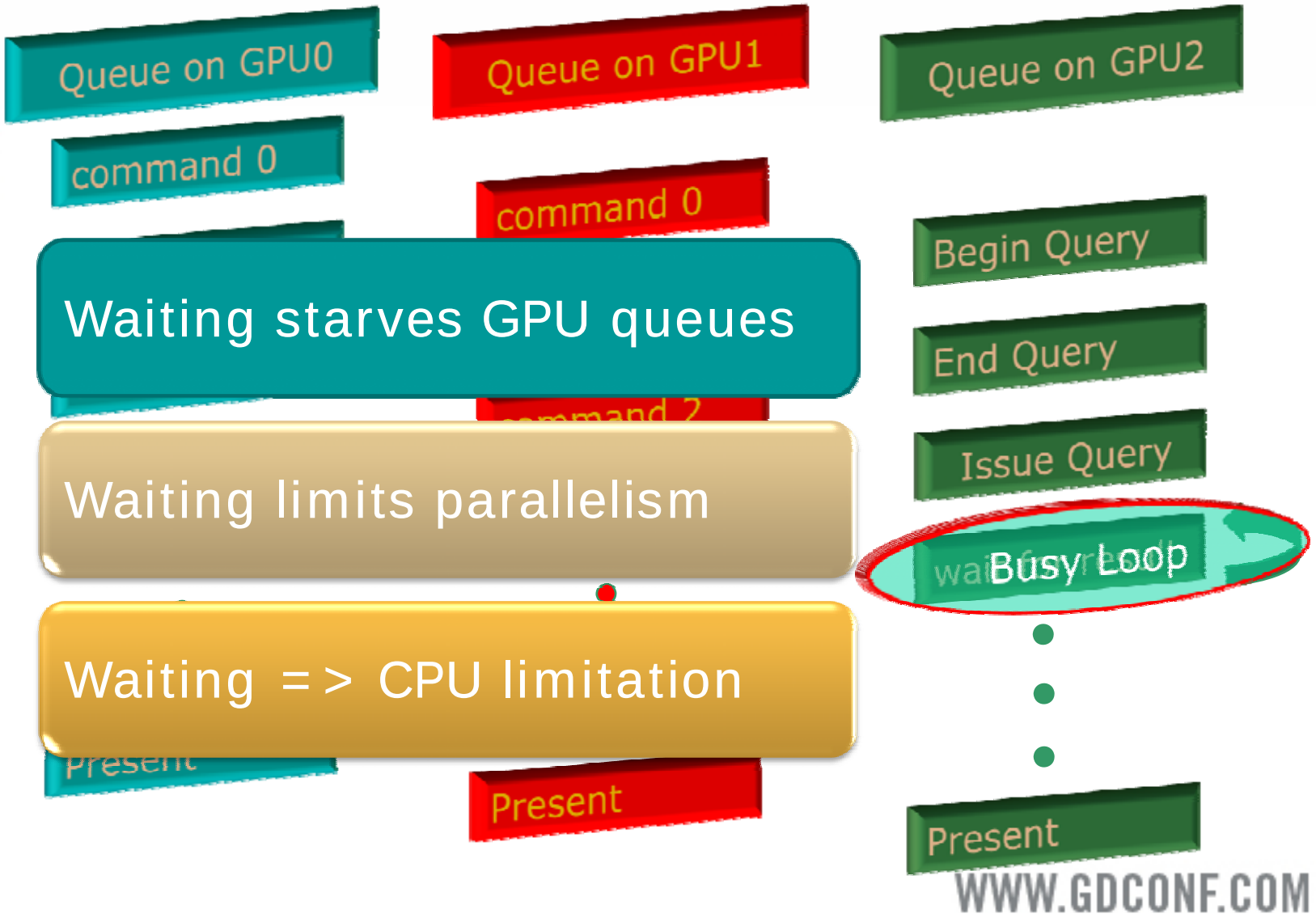


Good for resources that change every few frames

Pitfalls:Things to watch out for under DX10

- Drawing to vertex/index buffers
- Stream output buffers
- CopyResource calls
- CopySubresourceRegion calls
- GenerateMips calls
- ResolveSubresource calls
- Do not use same resource as destination of both Map(WRITE_DISCARD) and CopyResource/CopySubresourceRegion calls

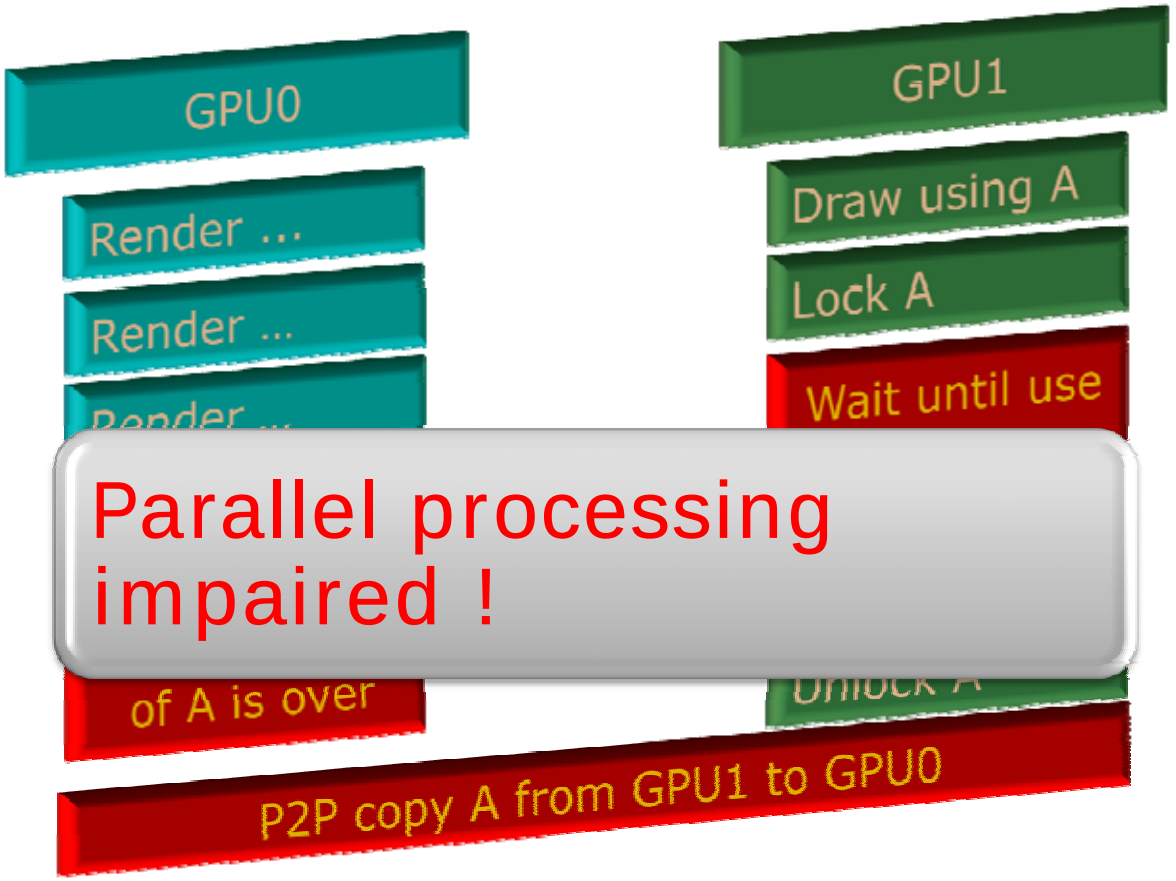
Pitfalls: Busy waits on Queries



Solutions for Queries

- Begin/End queries in same frame
- Use N-GPU queries if used every frame
- Expect results *starting* N-GPU frames after ending the query

Pitfalls: Locks/Maps on renderable resources



Solutions/Pitfalls: Locks/Maps

Lock/Map flip-chain or render-able resources

- On DX10 call UpdateXX() (copy from STAGING resources)
- On DX9 blits are always better than locks
- Use dynamic hint/CPU-flags at creation
- Hint discard/no-overwrite during Lock/Map
  - DX10: don‘t use discard a lot !!

Lock (DX9) static vertex/index buffers

- Change happens only on one card => P2P copies

Concluding Pitfalls & Solutions

- Not all AFR unfriendliness causes artifacts
- One frame old data may be OK
- Compatible AFR mode can't detect this
- Fixing invisible problems sacrifices perf.
- Never use shared resources on DX10
  - No way to detect update by other app

Call to Action

- Use AMD / NVIDIA libraries to detect MGPU topology
- Write AFR friendly rendering code
- Find out about your scaling
- Talk to us if you do encounter problems

Questions?