Parallel Frame Rendering: Trading Responsiveness for Energy on a Mobile GPU

Jose-Maria Arnau[1]
jarnau@ac.upc.edu

Joan-Manuel Parcerisa[1]
jmanel@ac.upc.edu

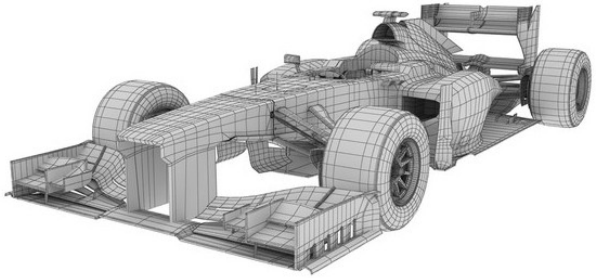
Polychronis Xekalakis[2]
polychronis.xekalakis@intel.com

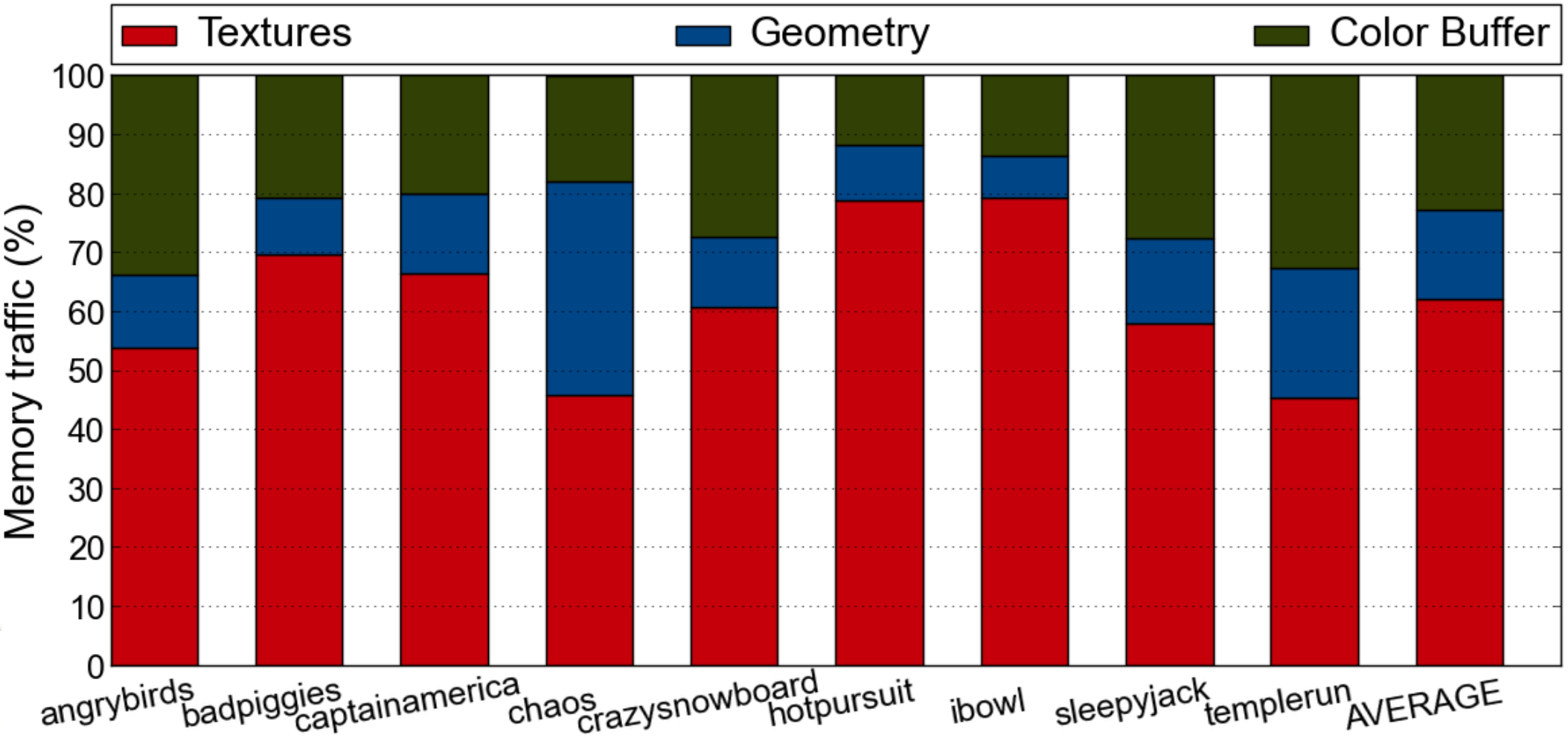
[1]Universitat Politecnica de Catalunya

[2]Intel Labs, Intel Corporation

09 / September / 2013

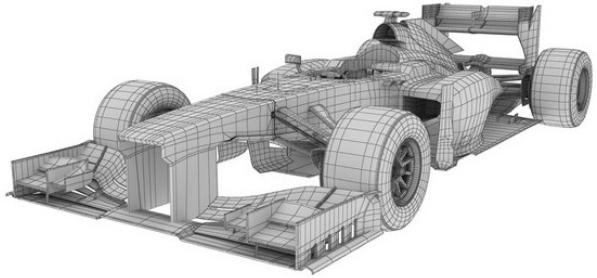
Bandwidth Usage for Graphics

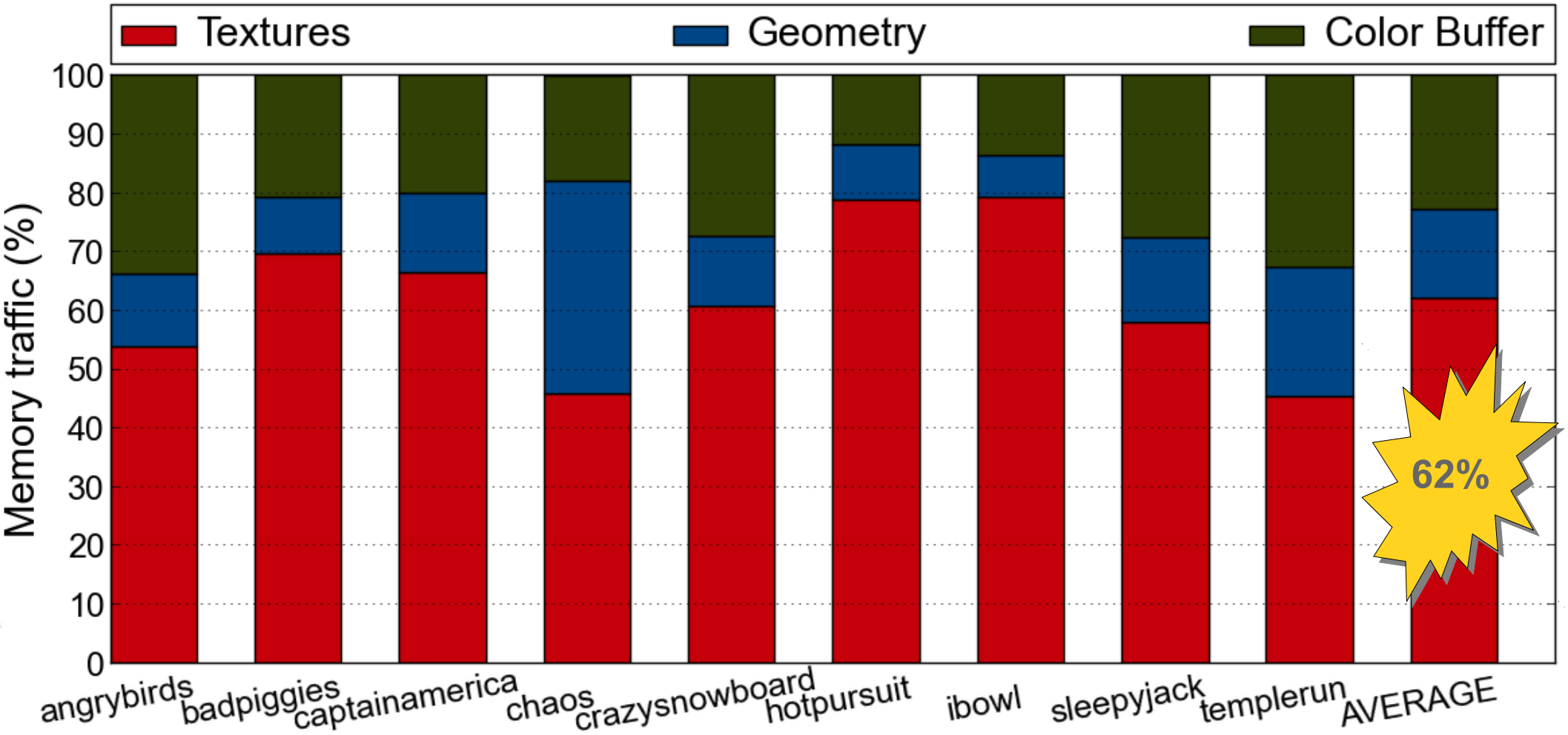




Textures and 3D models from: http://www.turbosquid.com

Bandwidth Usage for Graphics

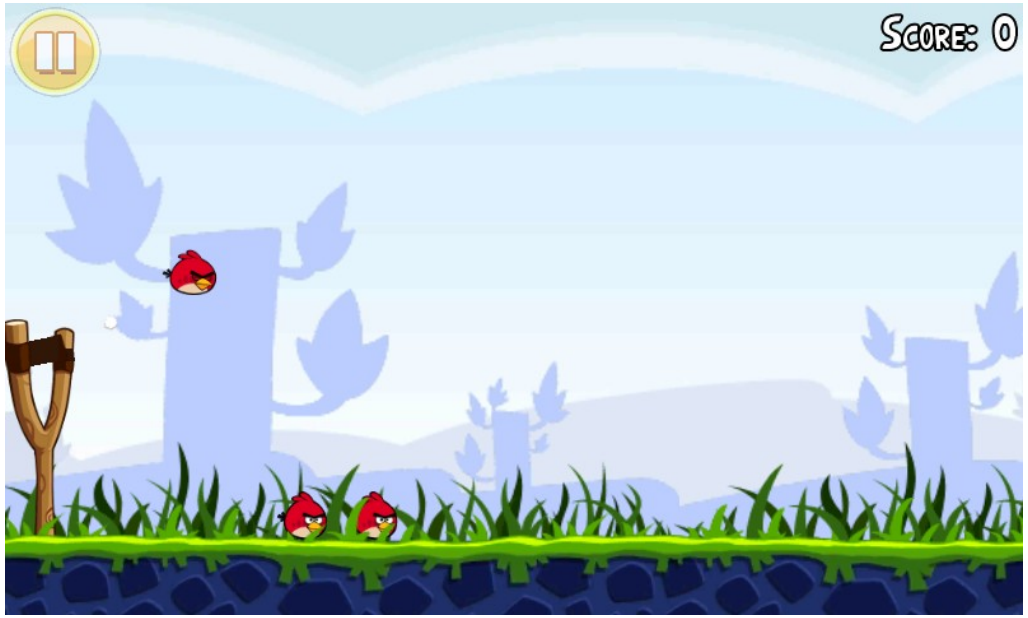




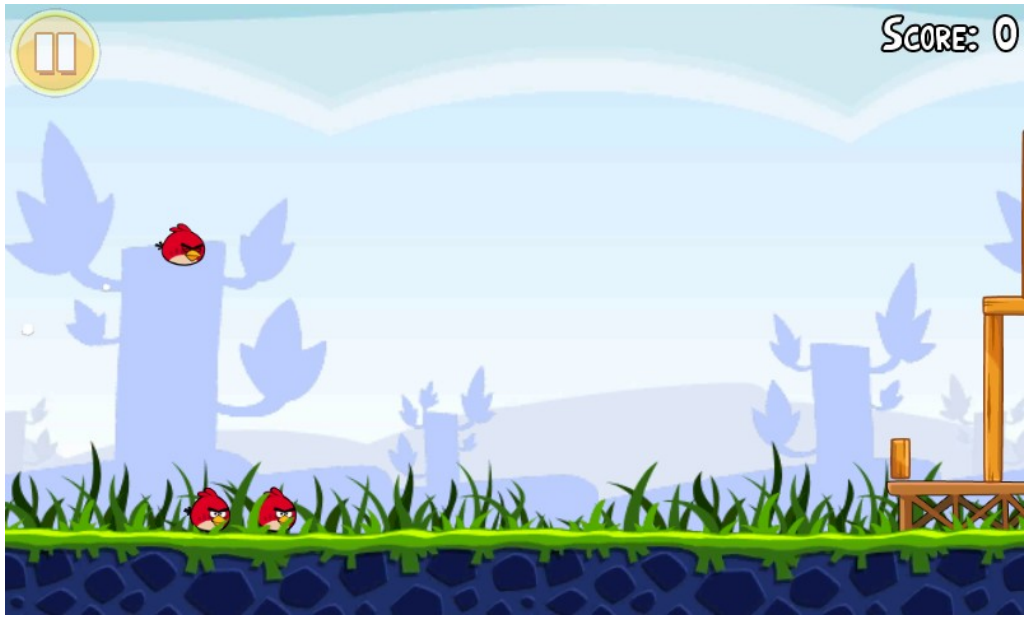
Textures and 3D models from: http://www.turbosquid.com

Texture Reuse

Frame i

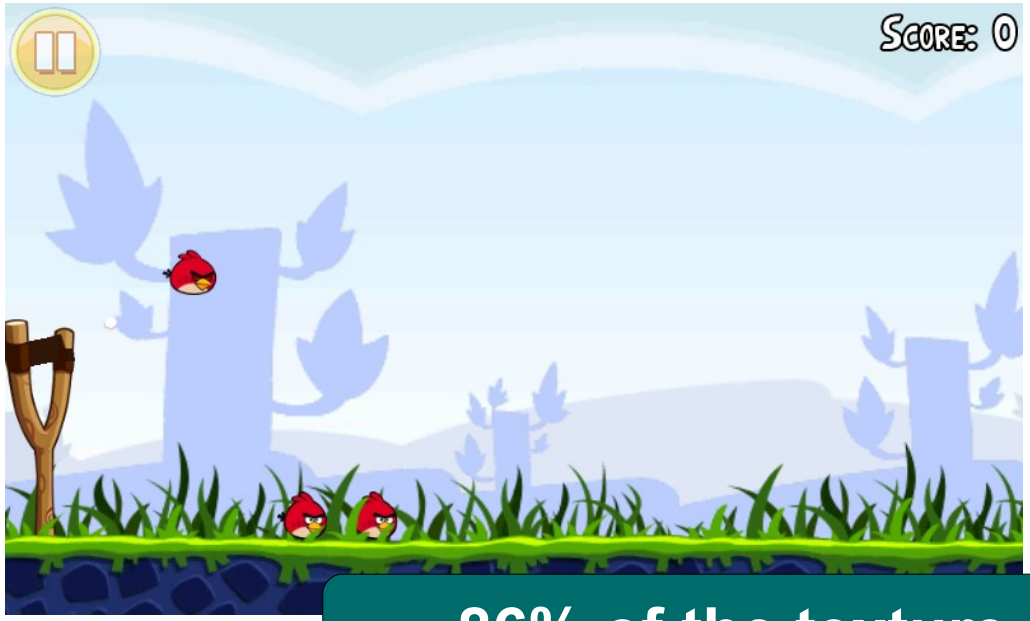


Frame i+1

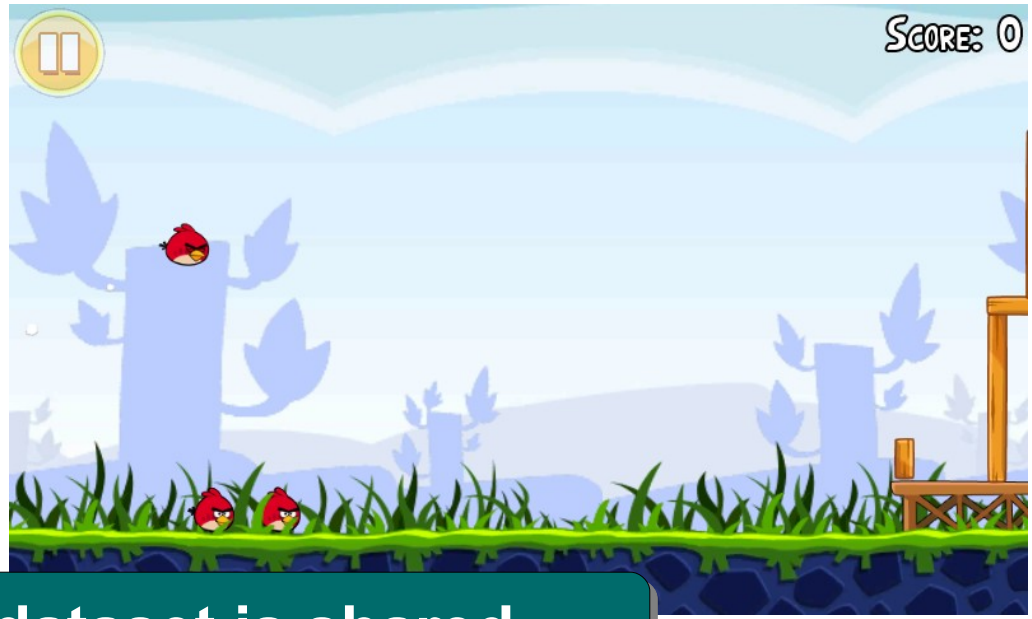


Texture Reuse

Frame i



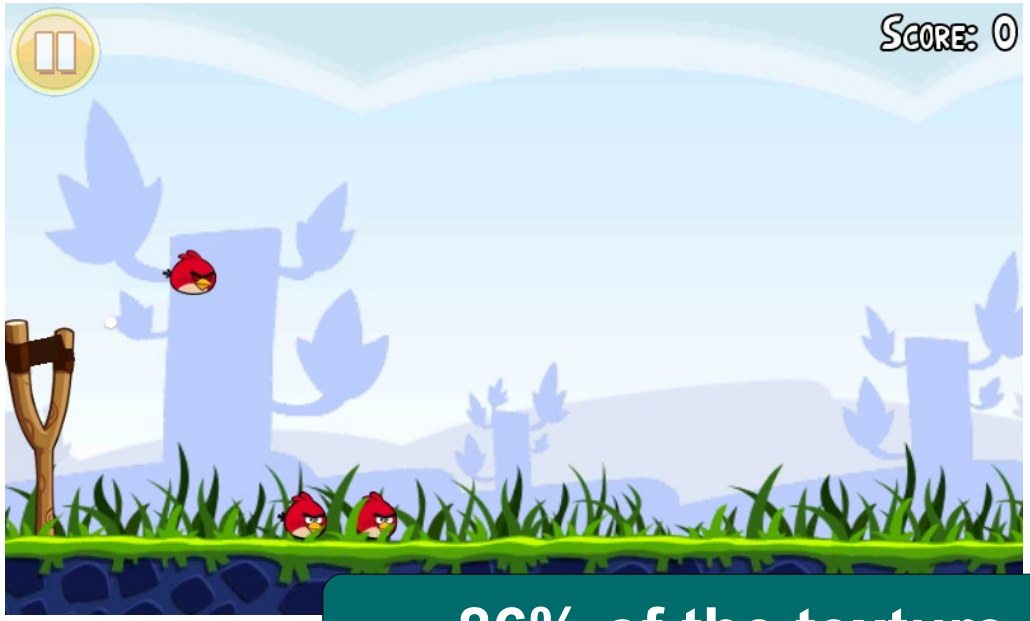
Frame i+1



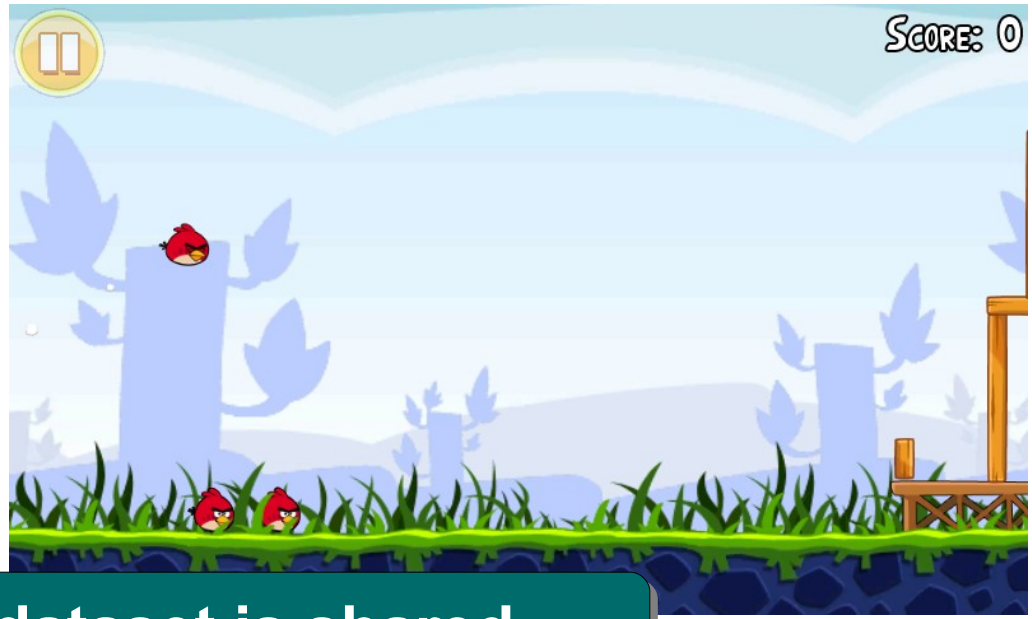
86% of the texture dataset is shared

Texture Reuse

Frame i



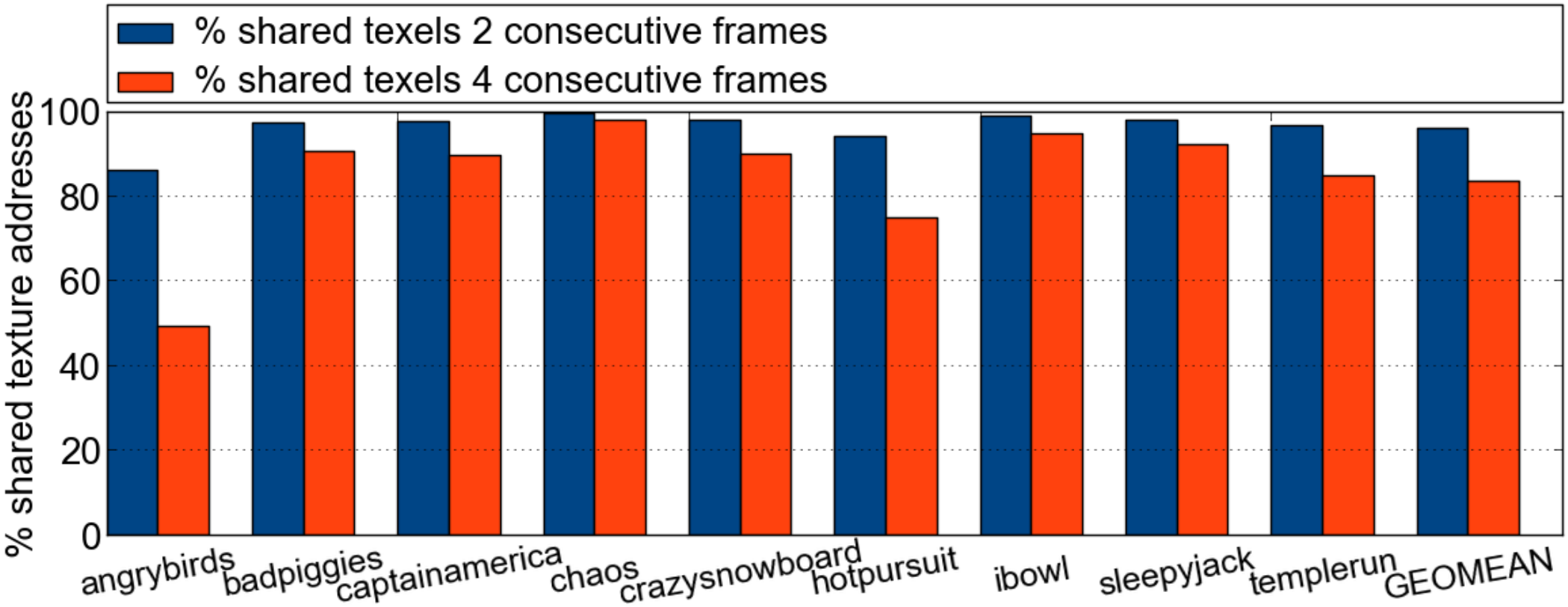
Frame i+1



86% of the texture dataset is shared

Mobile games exhibit a high degree of **texture similarity** between consecutive frames

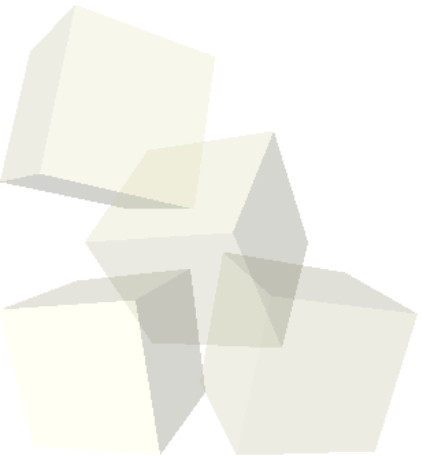
Texture Reuse



Mobile games exhibit a high degree of **texture similarity** between consecutive frames

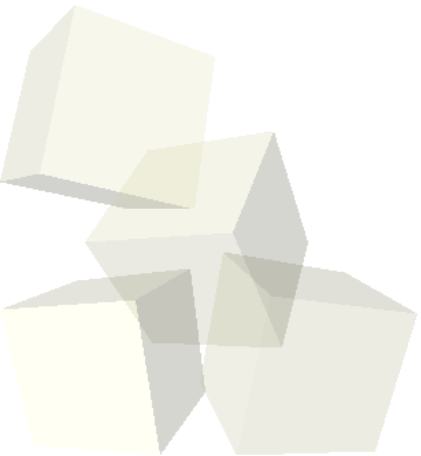
Outline

1. Motivation
2. Conventional Rendering
3. Parallel Frame Rendering
4. Experimental Results
5. Conclusions



Outline

1. Motivation
2. Conventional Rendering
3. Parallel Frame Rendering
4. Experimental Results
5. Conclusions



Conventional Tile-Based Rendering

CPU

Process user inputs
Physical simulation
Dispatch drawing commands

System Memory

Textures

Geometry

GPU

Command Processor

L2 Cache

Geometry Unit

Raster Unit 0

Memory Controller

Tiling Engine

Raster Unit 1

Color Buffer

Conventional Tile-Based Rendering

CPU

Process user inputs
Physical simulation
Dispatch drawing commands

System Memory

Textures

Geometry

GPU

Command Processor

L2 Cache

Geometry Unit

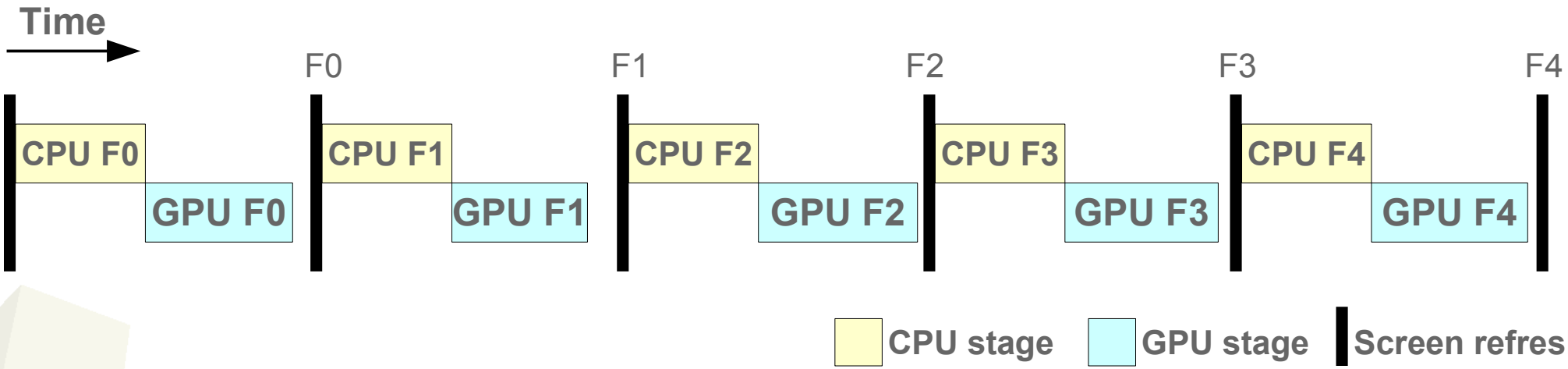
Raster Unit 0

Memory Controller

Tiling Engine

Raster Unit 1

Color Buffer



Conventional Tile-Based Rendering

CPU

Process user inputs
Physical simulation
Dispatch drawing commands

System Memory

Textures

Geometry

GPU

Command Processor

L2 C 

Geometry Unit

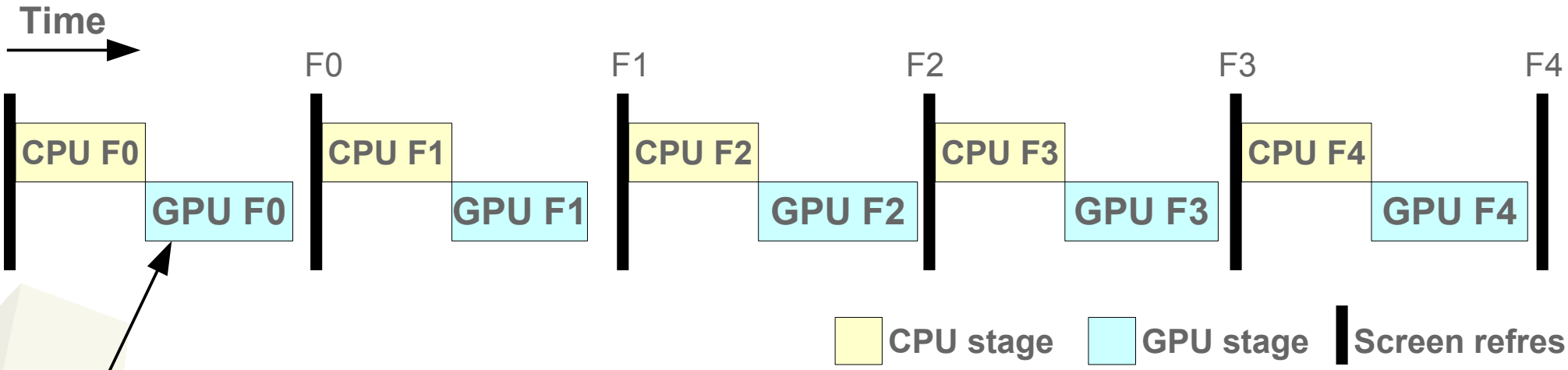
Raster Unit 0

Memory Controller

Tiling Engine

Raster Unit 1

Color Buffer



Conventional Tile-Based Rendering

CPU

Process user inputs
Physical simulation
Dispatch drawing commands

System Memory

Textures

Geometry

GPU

Command Processor

L2 Ca

Geometry Unit

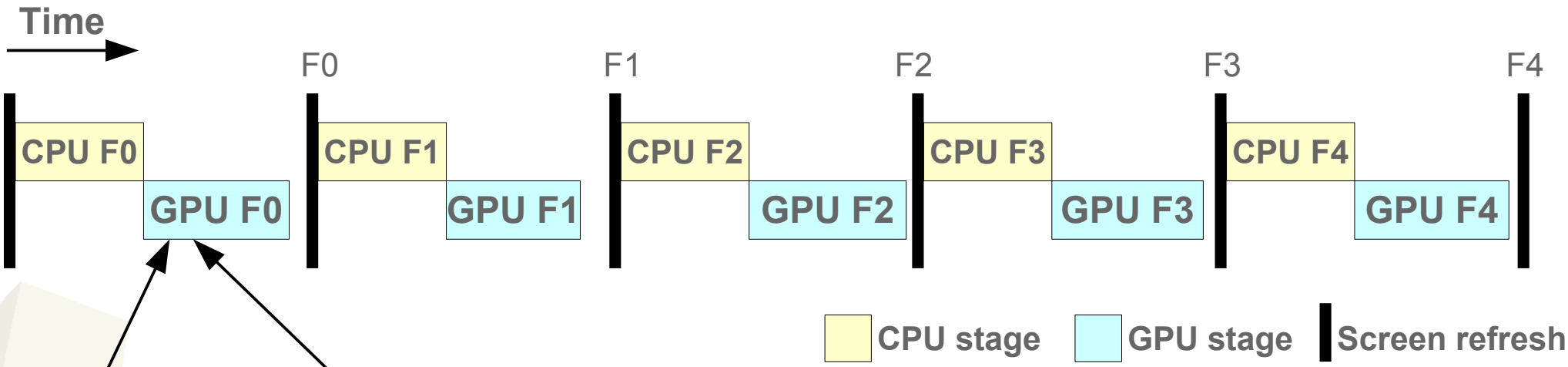
Raster Unit 0

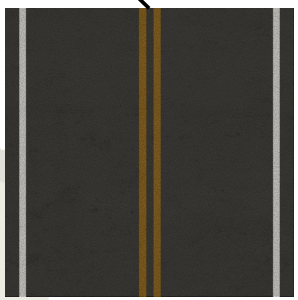
Memory Controller

Tiling Engine

Raster Unit 1

Color Buffer





Conventional Tile-Based Rendering

CPU

Process user inputs
Physical simulation
Dispatch drawing commands

System Memory

Textures

Geometry

GPU

Command Processor

L2 C

Geometry Unit

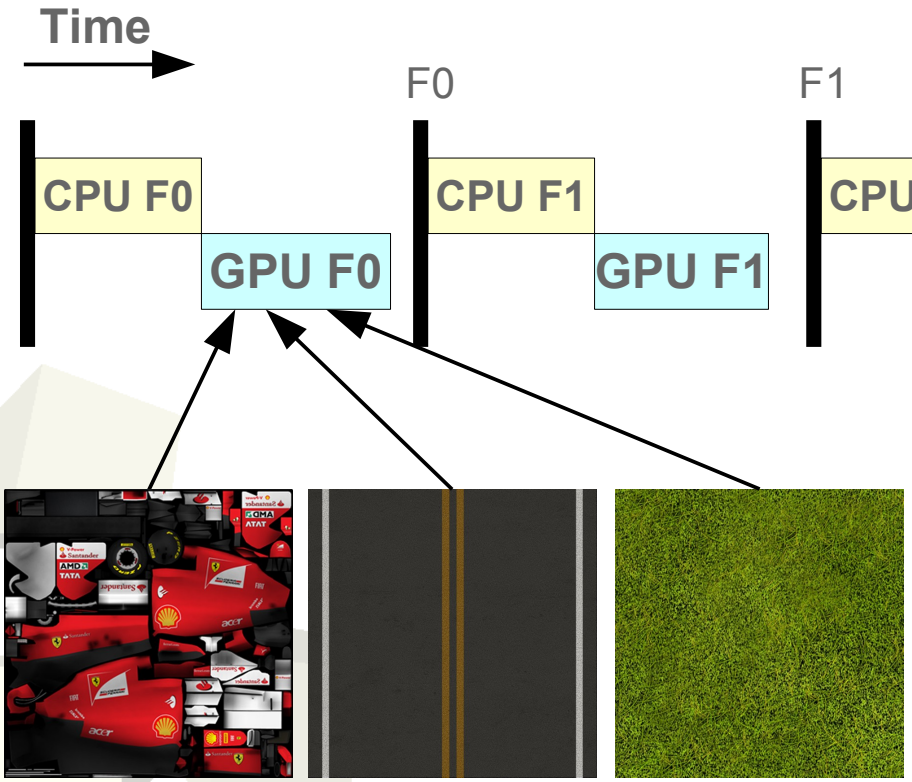
Raster Unit 0

Memory Controller

Tiling Engine

Raster Unit 1

Color Buffer



Conventional Tile-Based Rendering

CPU

Process user inputs
Physical simulation
Dispatch drawing commands

System Memory

Textures

Geometry

GPU

L2 C

Command Processor

Geometry Unit

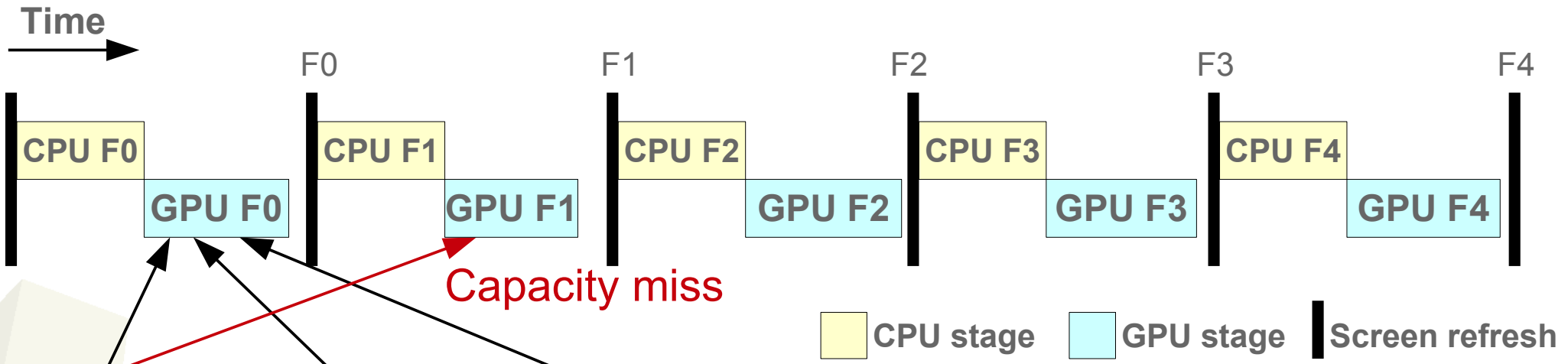
Raster Unit 0

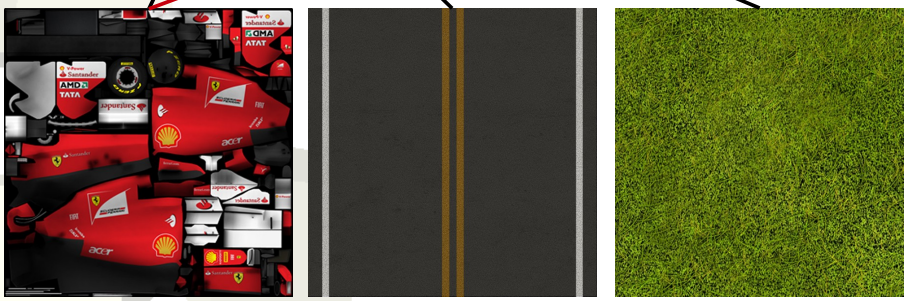
Memory Controller

Tiling Engine

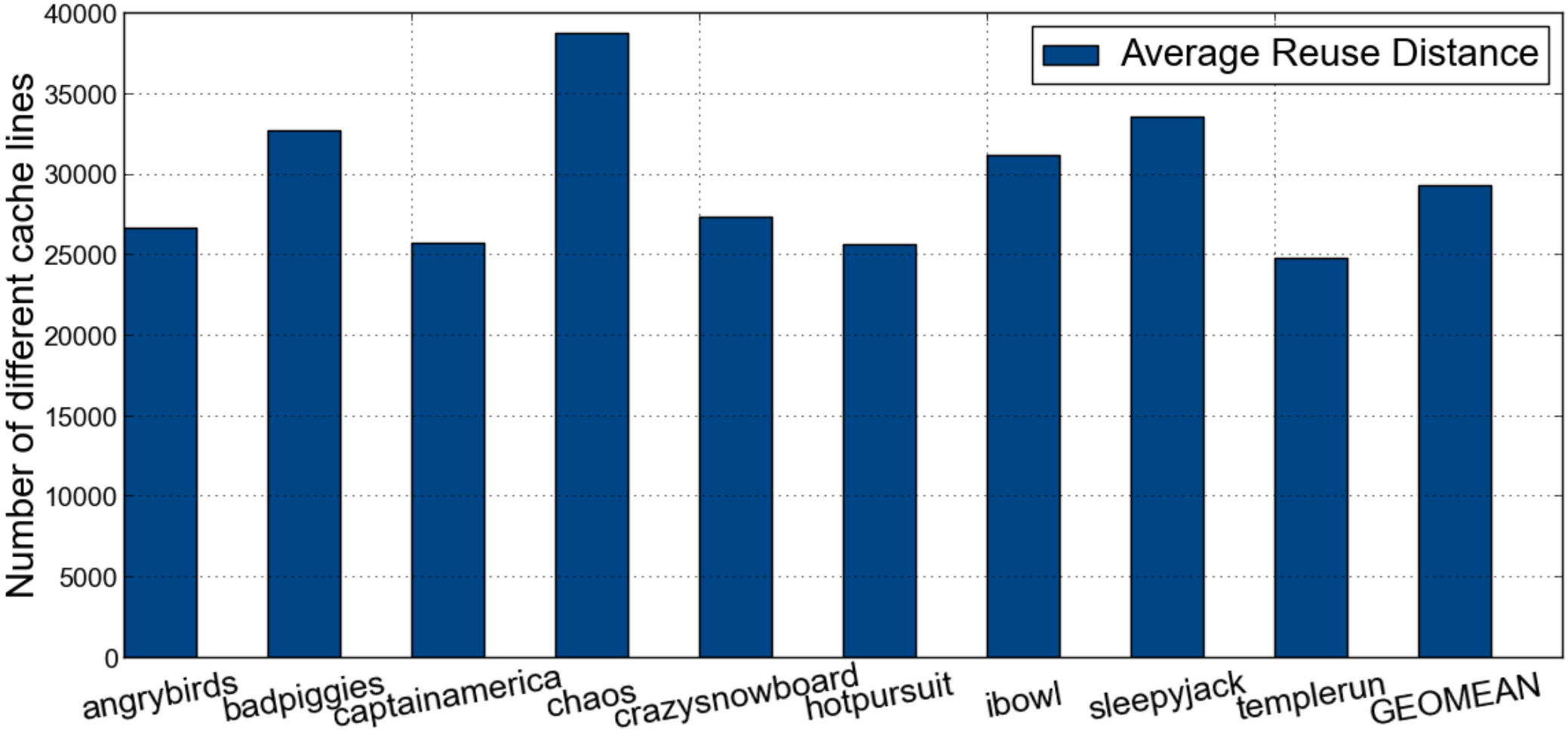
Raster Unit 1

Color Buffer

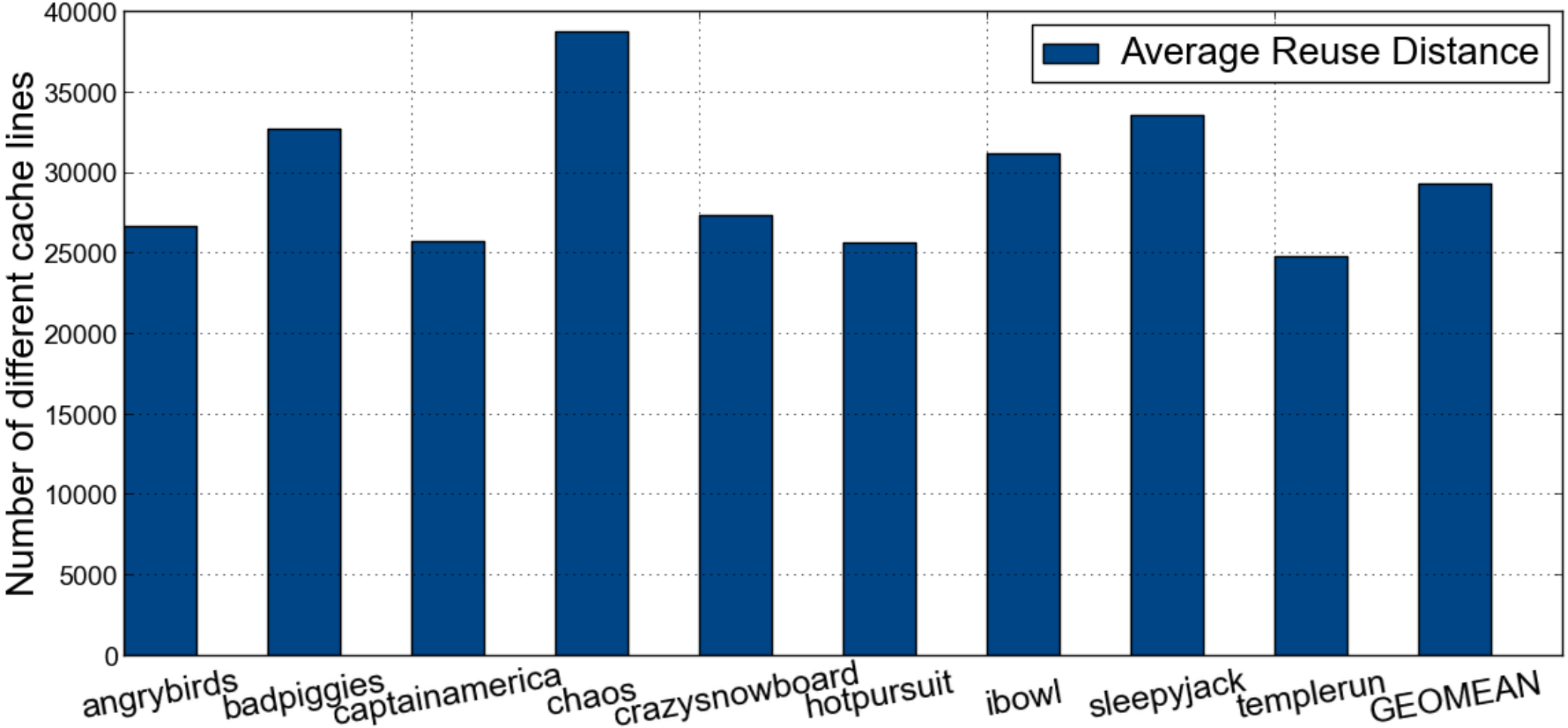




L2 Cache Reuse Distances



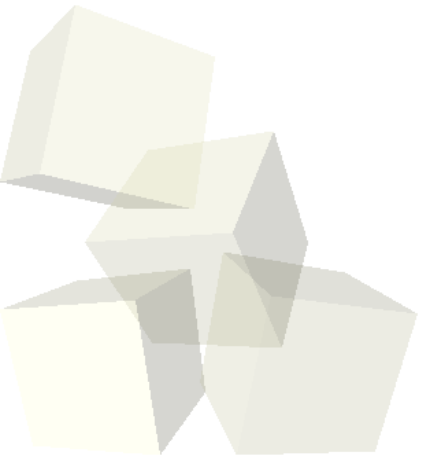
L2 Cache Reuse Distances



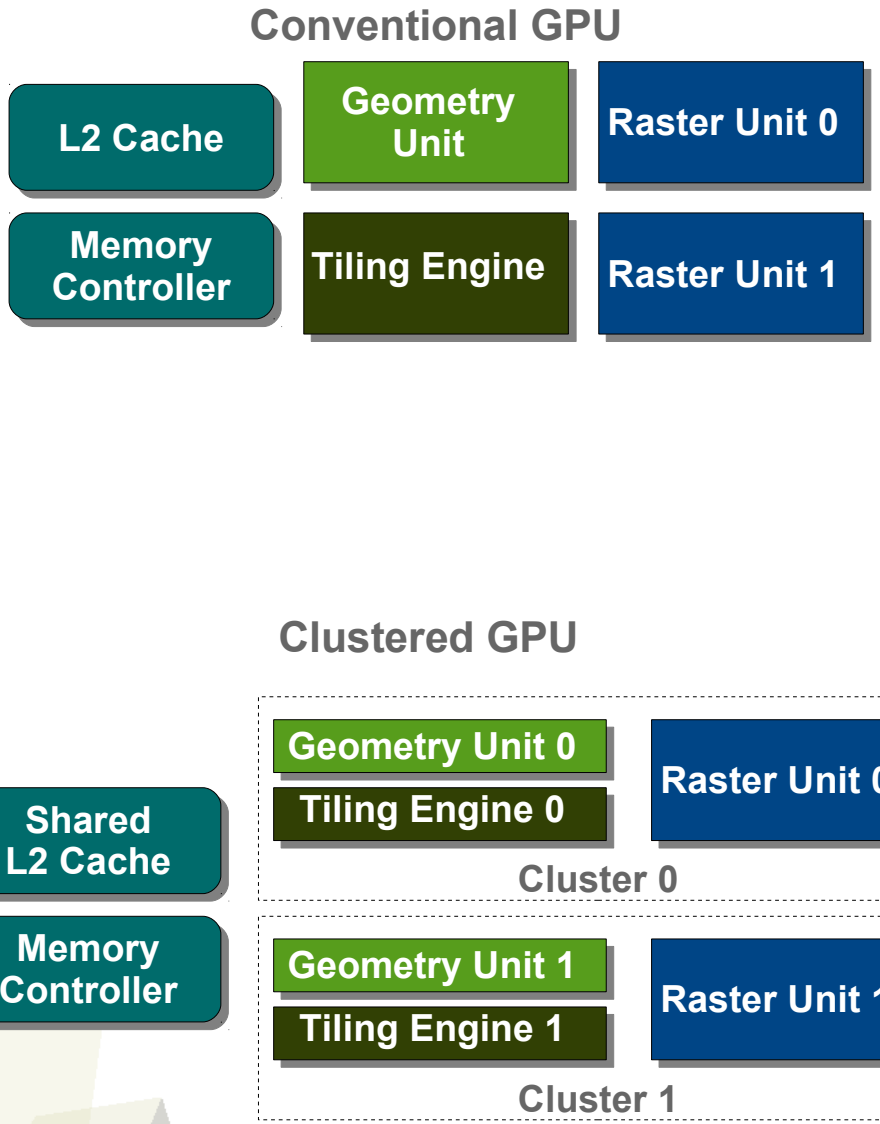
The L2 Cache cannot capture the inter-frame texture reuse due to the huge distances

Outline

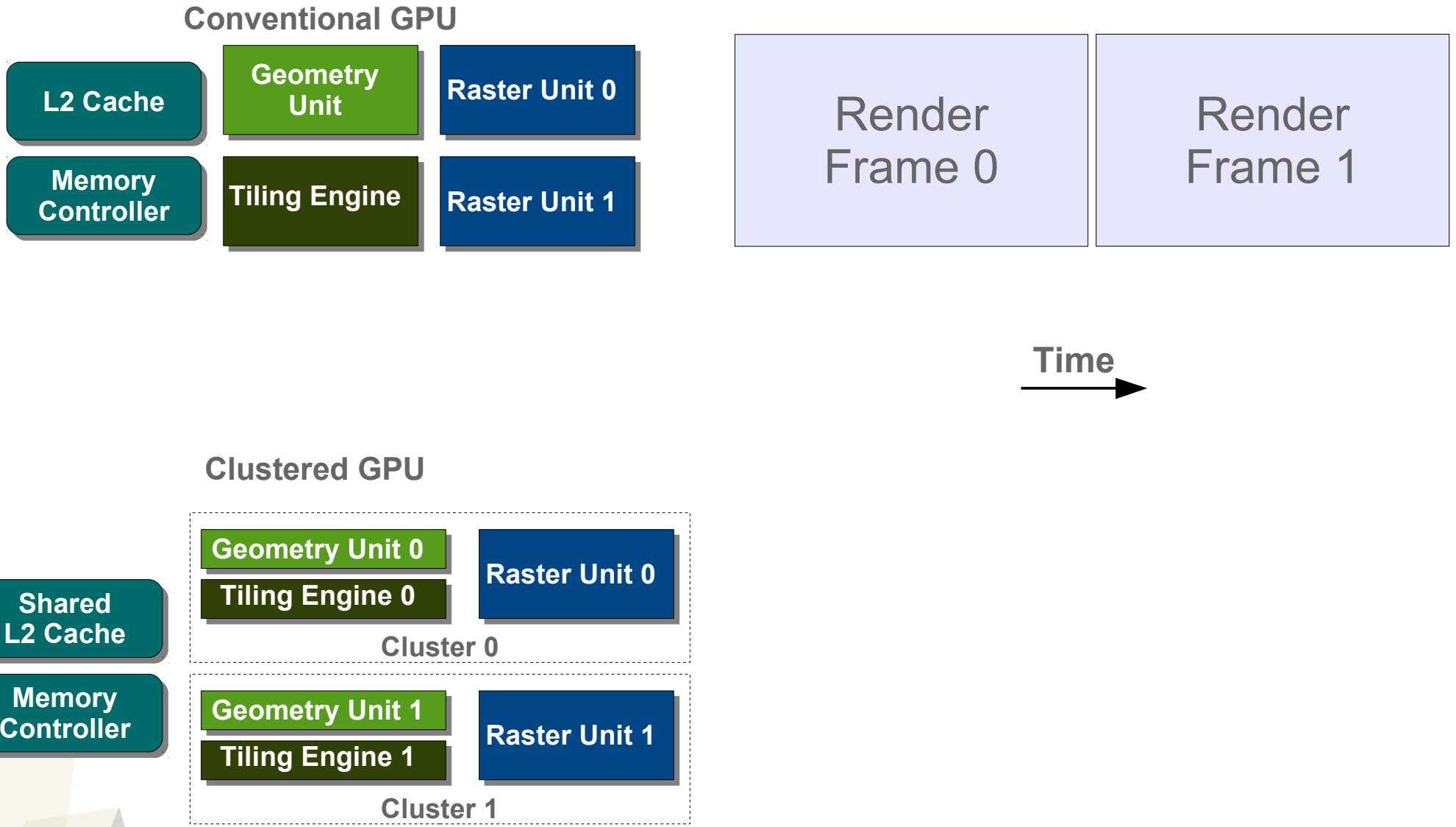
1. Motivation
2. Conventional Rendering
3. Parallel Frame Rendering
4. Experimental Results
5. Conclusions



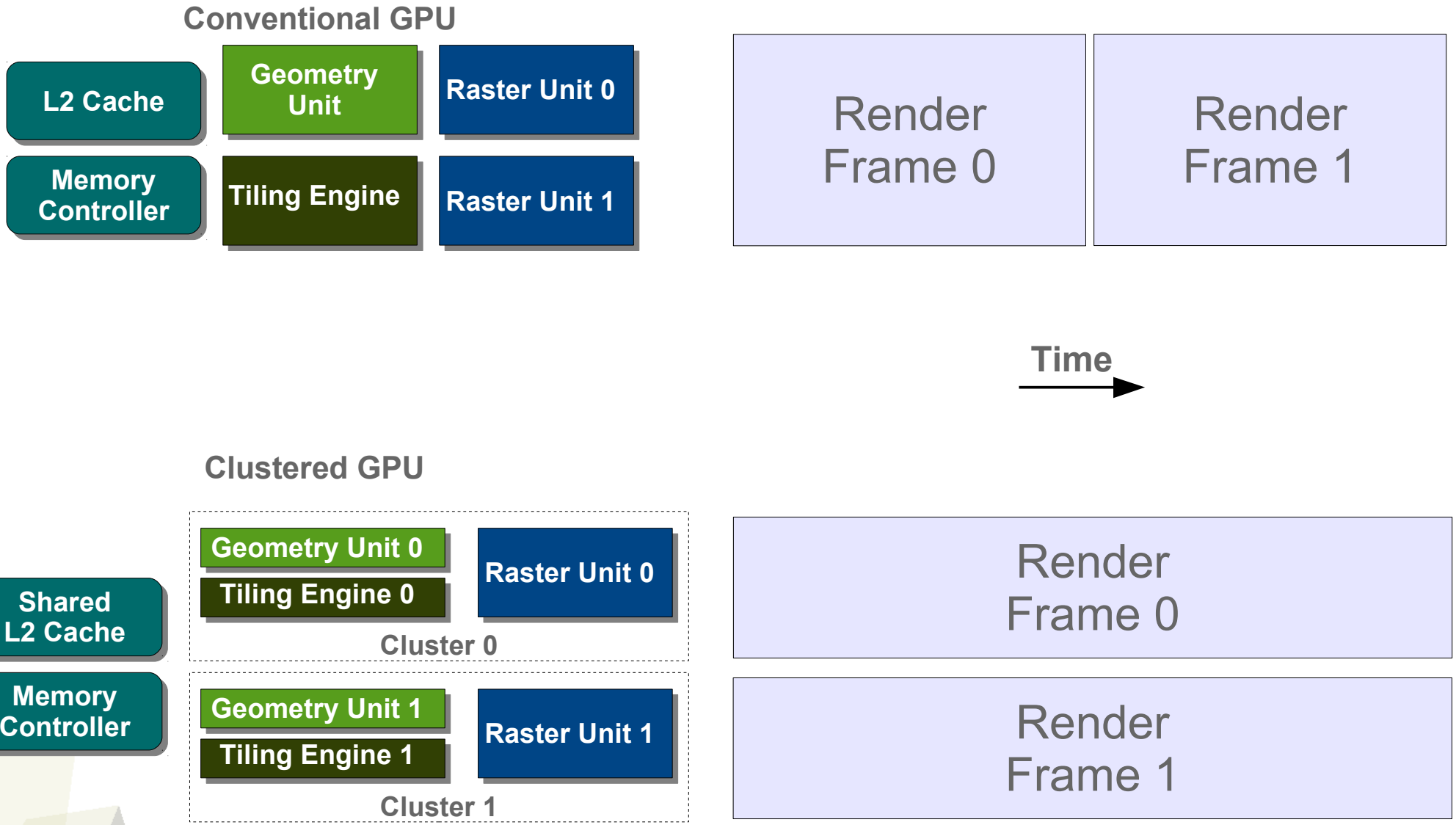
Parallel Frame Rendering



Parallel Frame Rendering



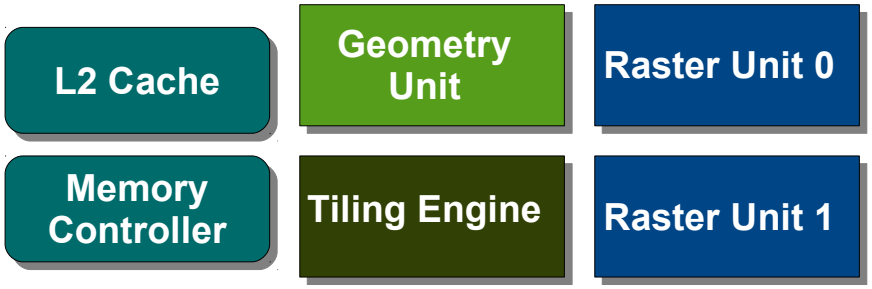
Parallel Frame Rendering

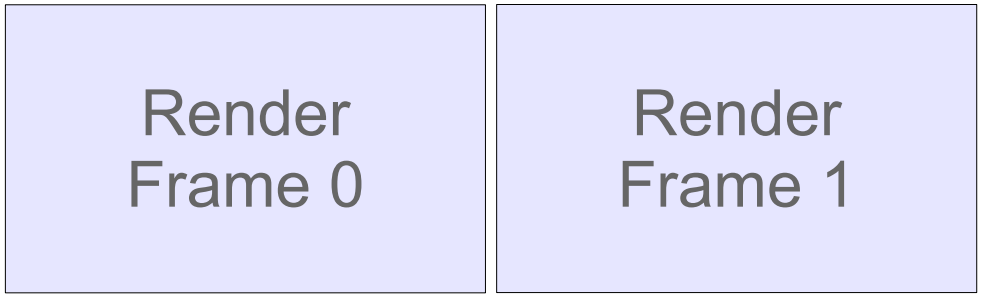


Parallel Frame Rendering

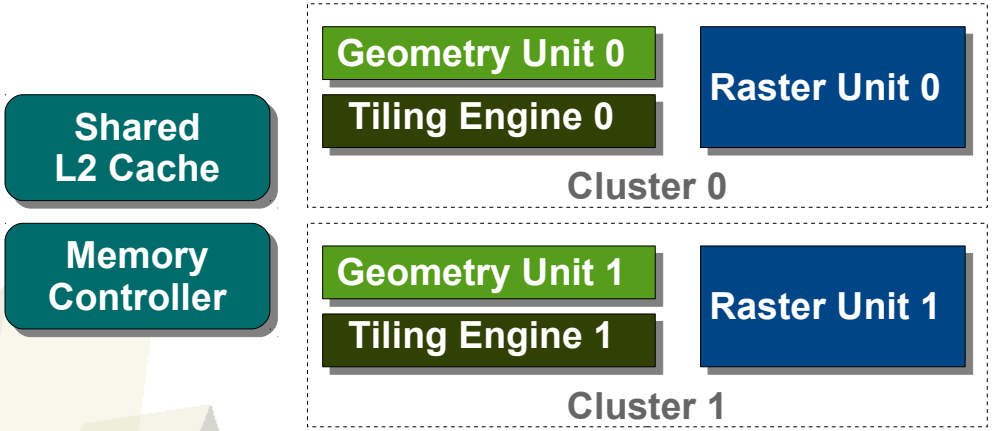
Time

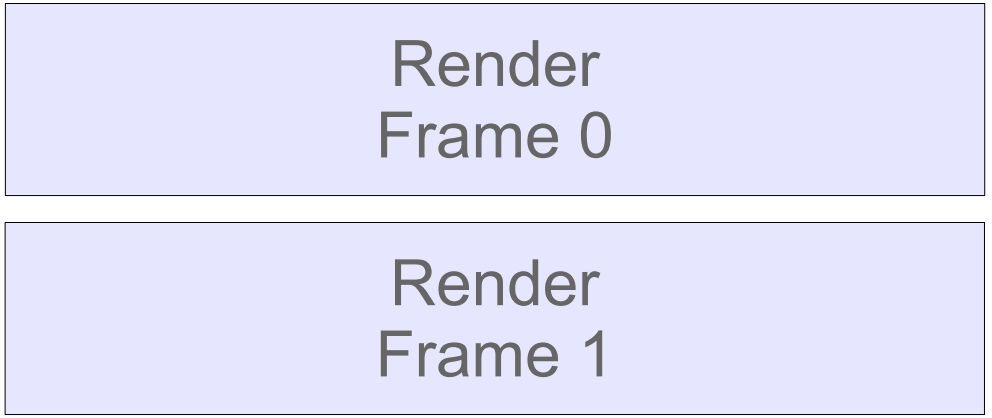
Conventional GPU



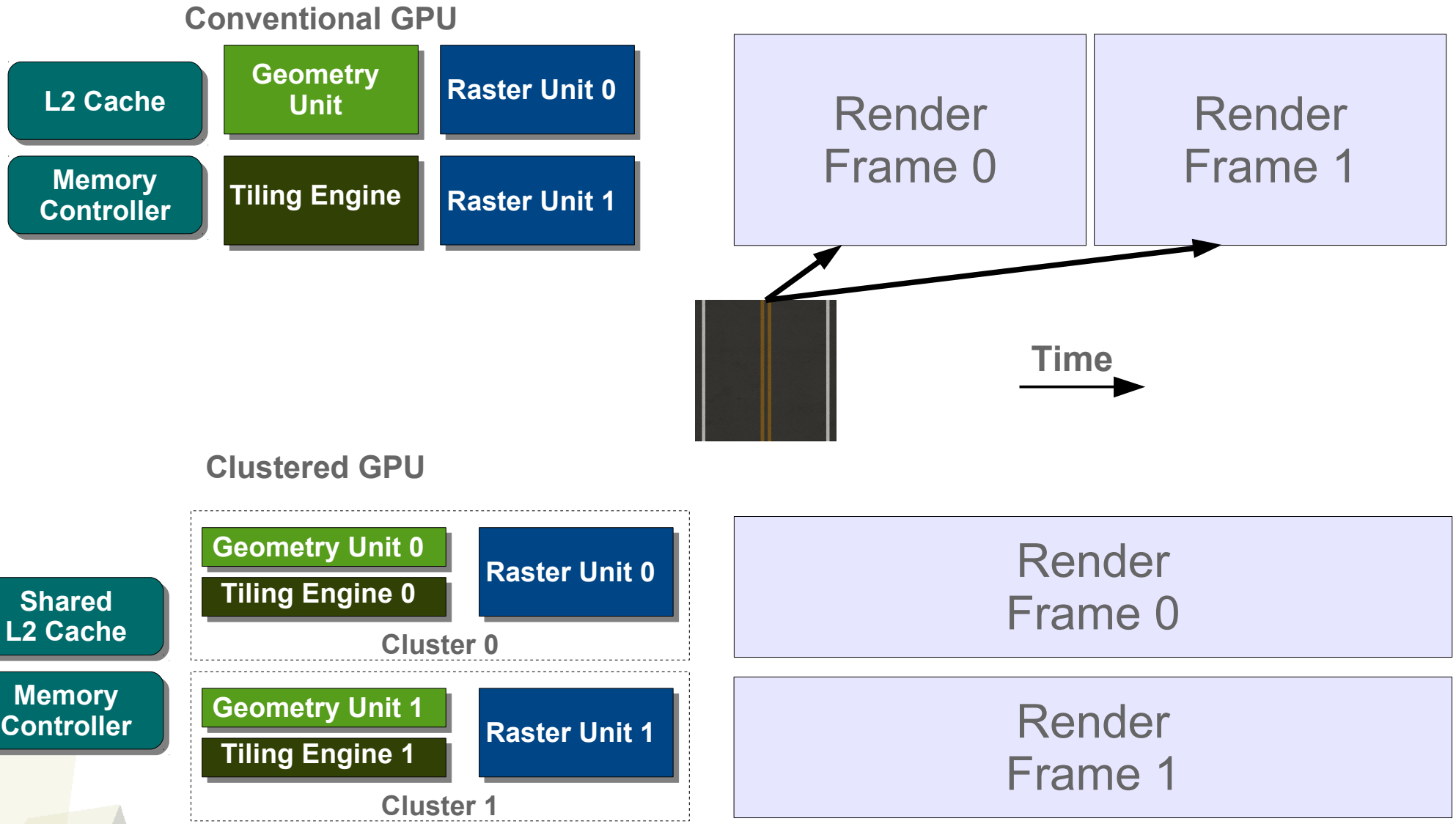


Clustered GPU

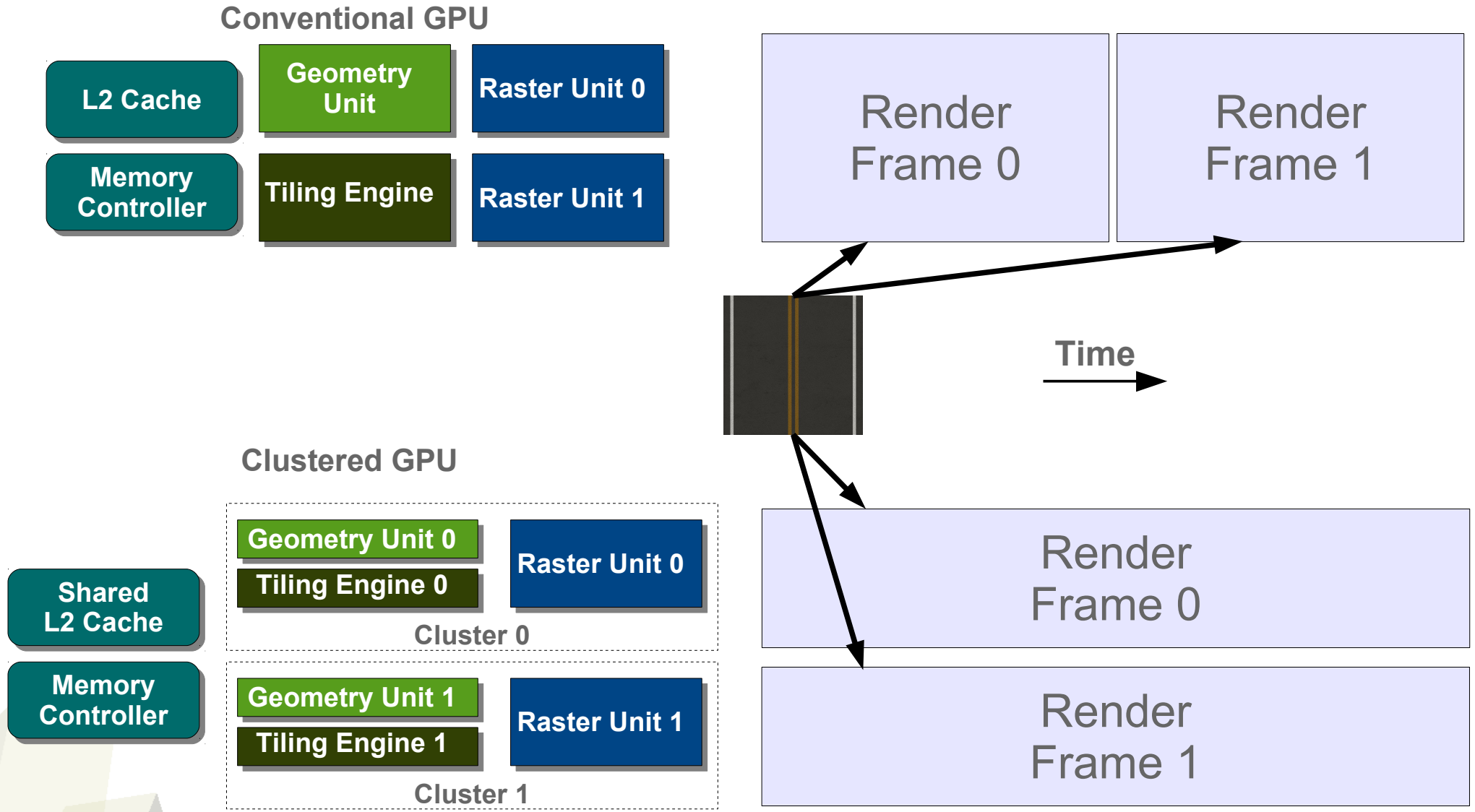




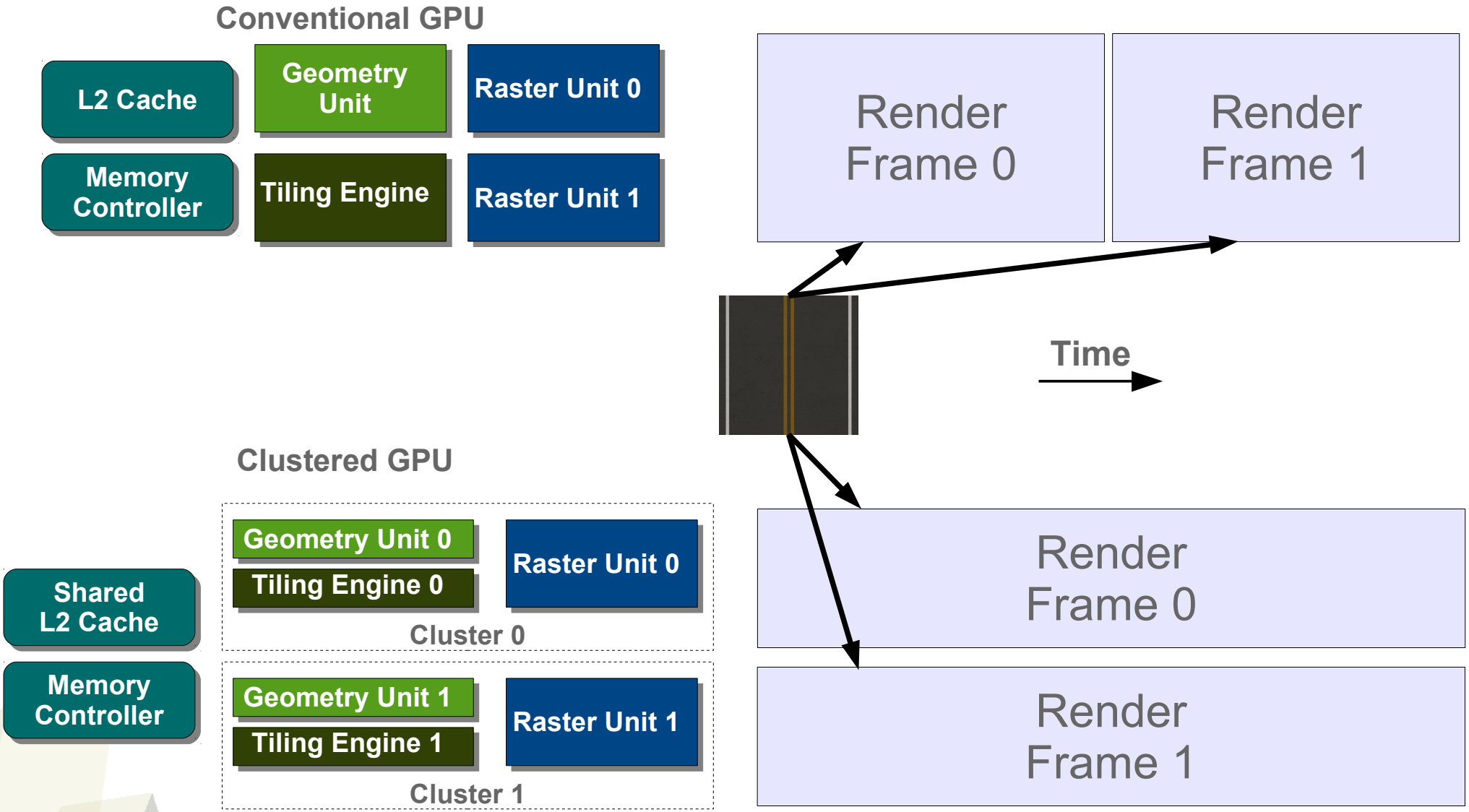
Parallel Frame Rendering



Parallel Frame Rendering

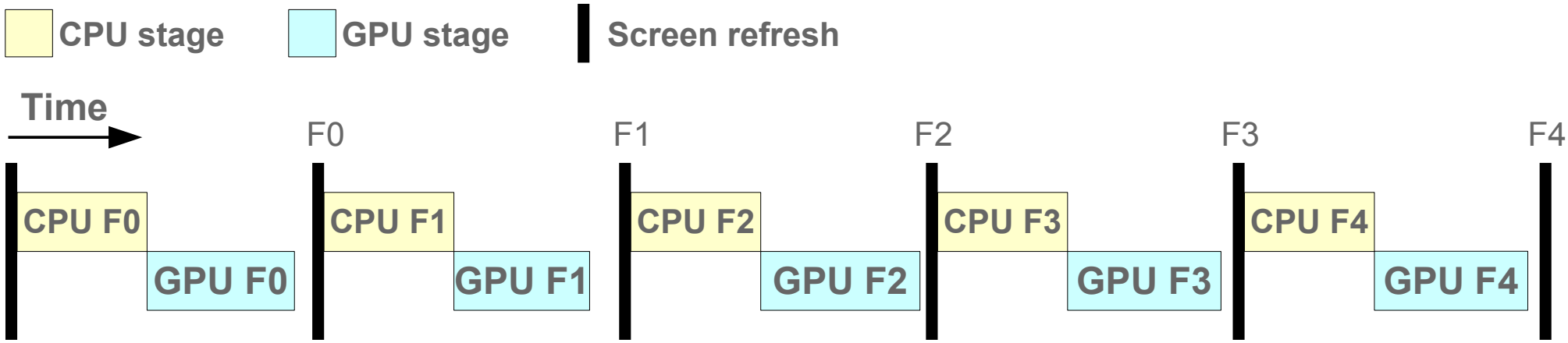


Parallel Frame Rendering

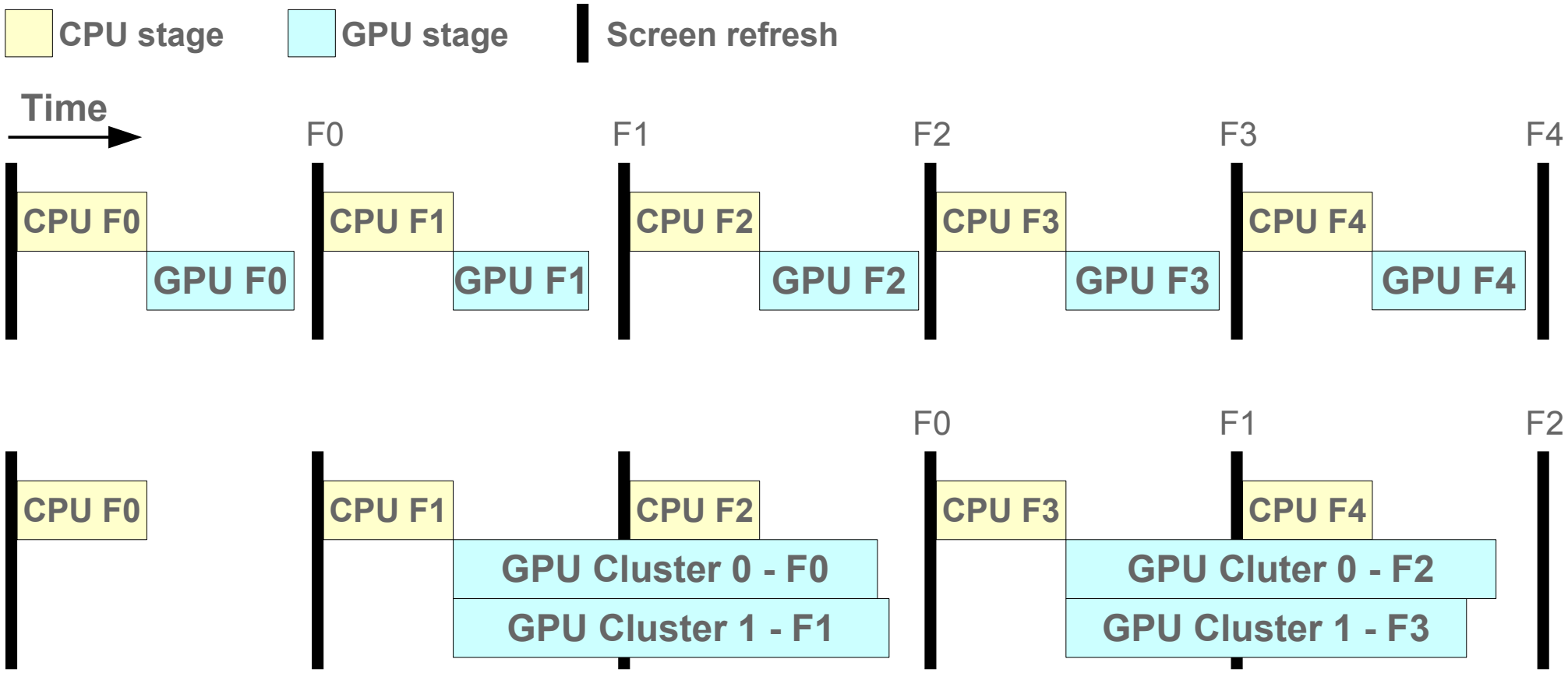


Textures are fetched once in the shared L2 cache and accessed by the 2 clusters in a short timespan

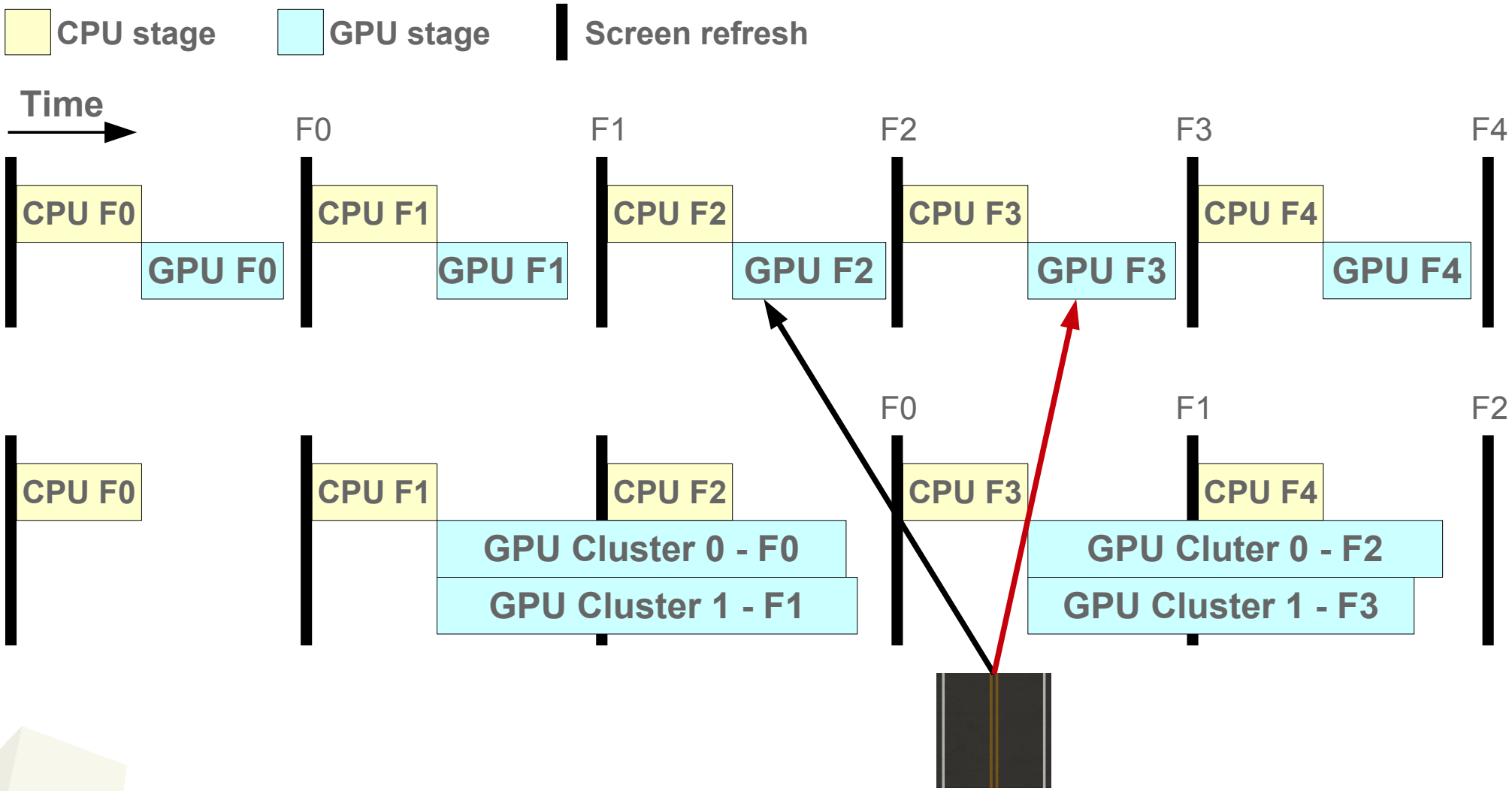
Parallel Frame Rendering



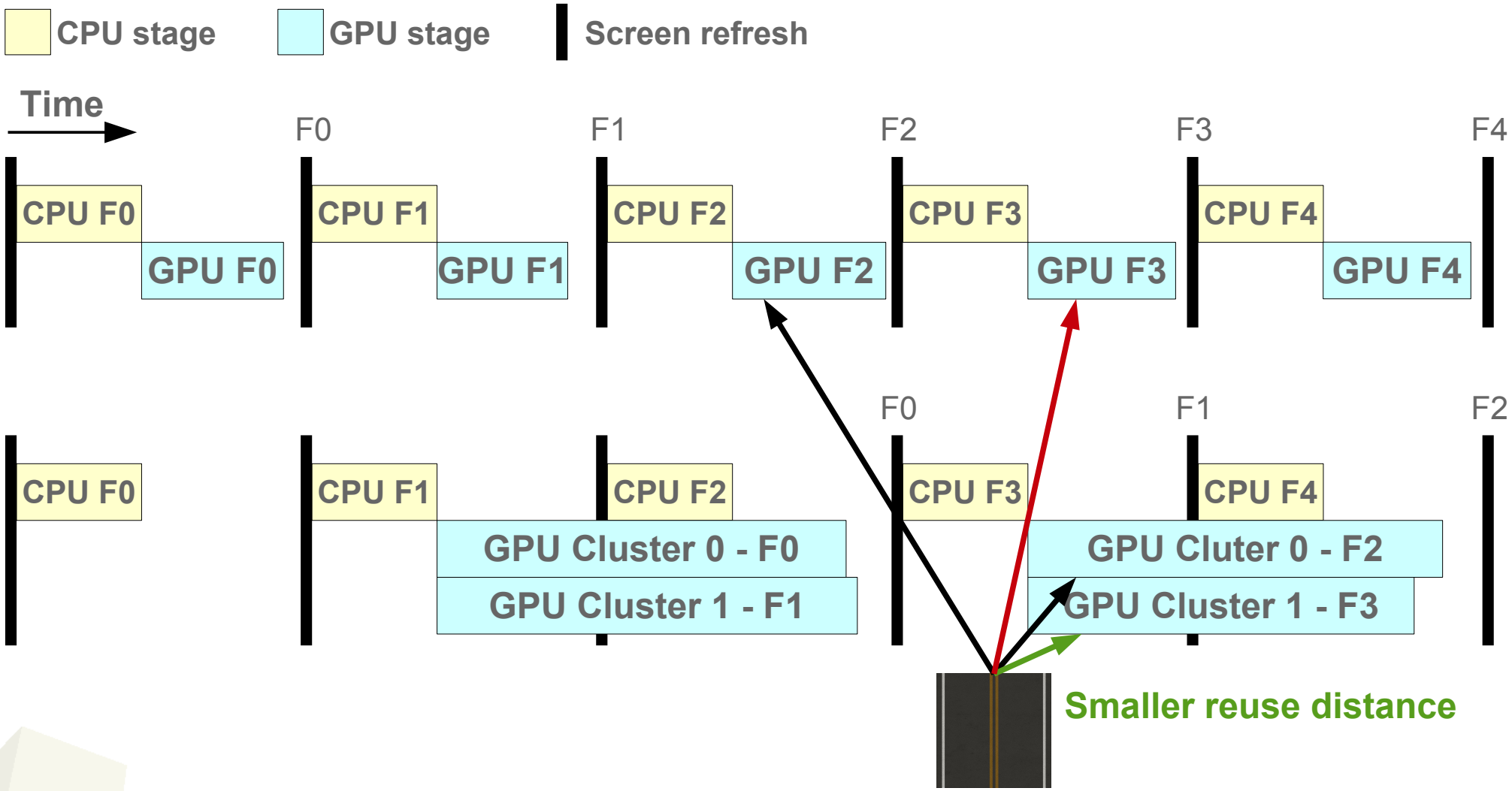
Parallel Frame Rendering



Parallel Frame Rendering

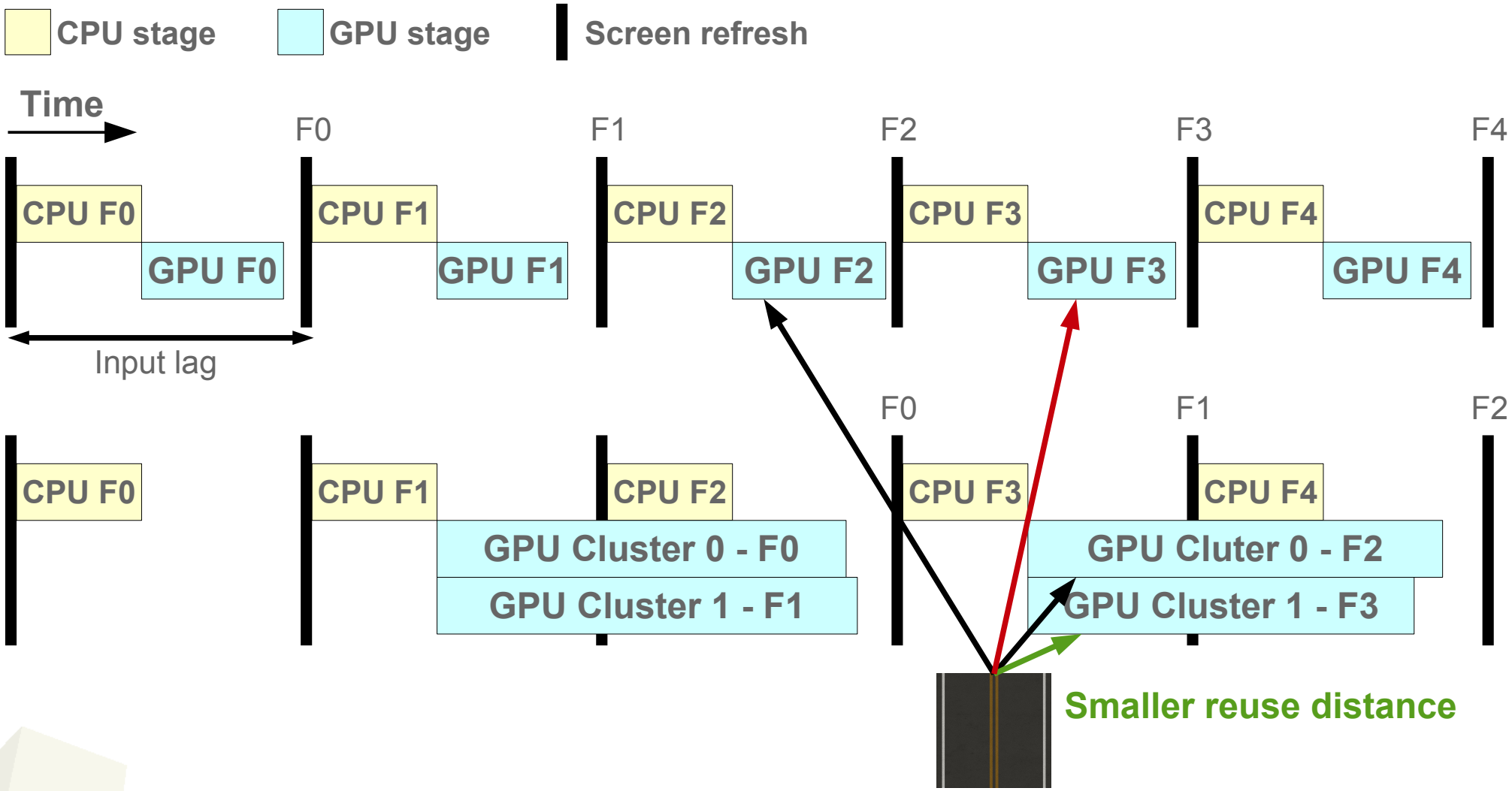


Parallel Frame Rendering



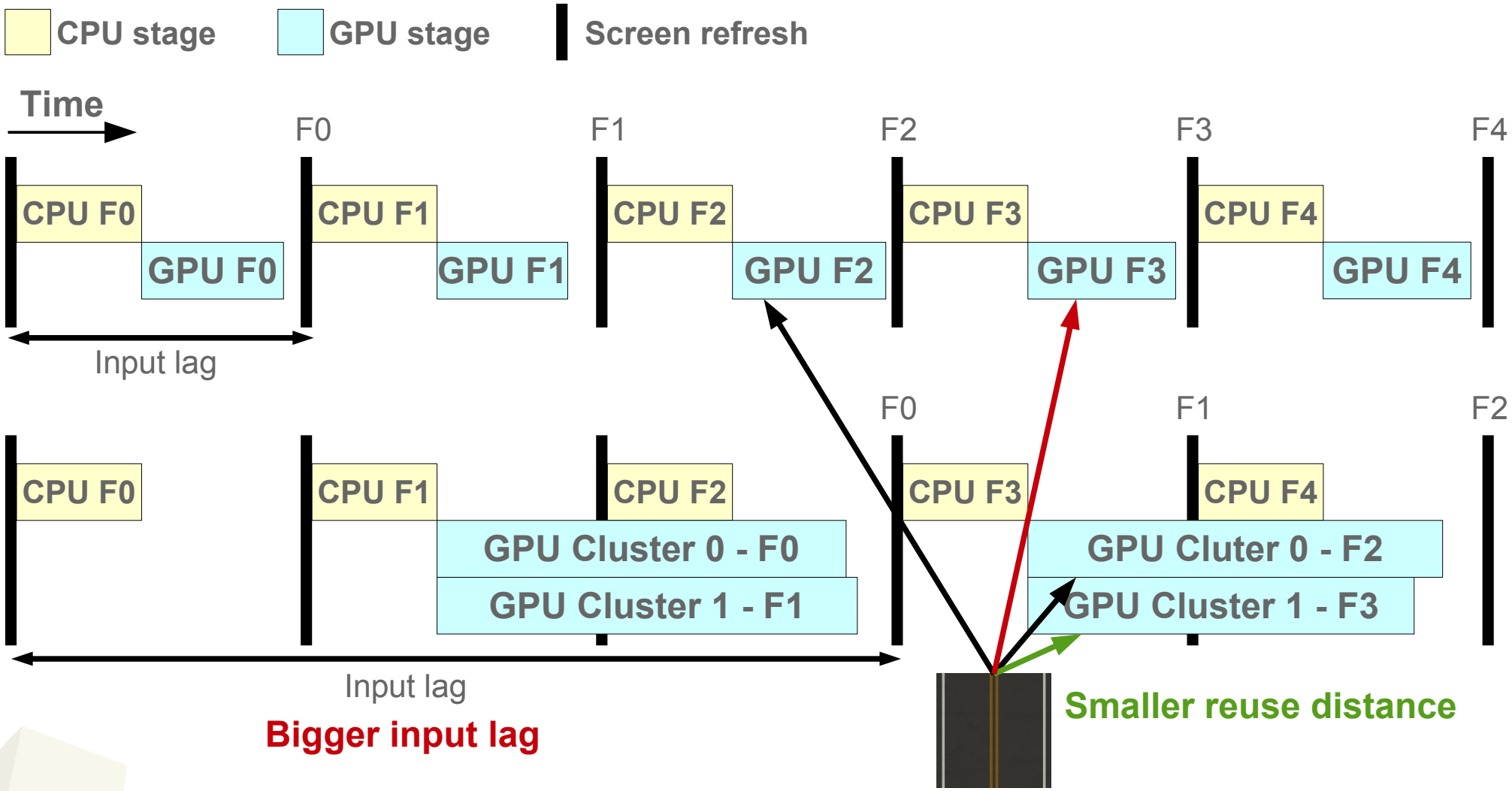
PFR saves bandwidth by overlapping the memory accesses of 2 consecutive frames

Parallel Frame Rendering



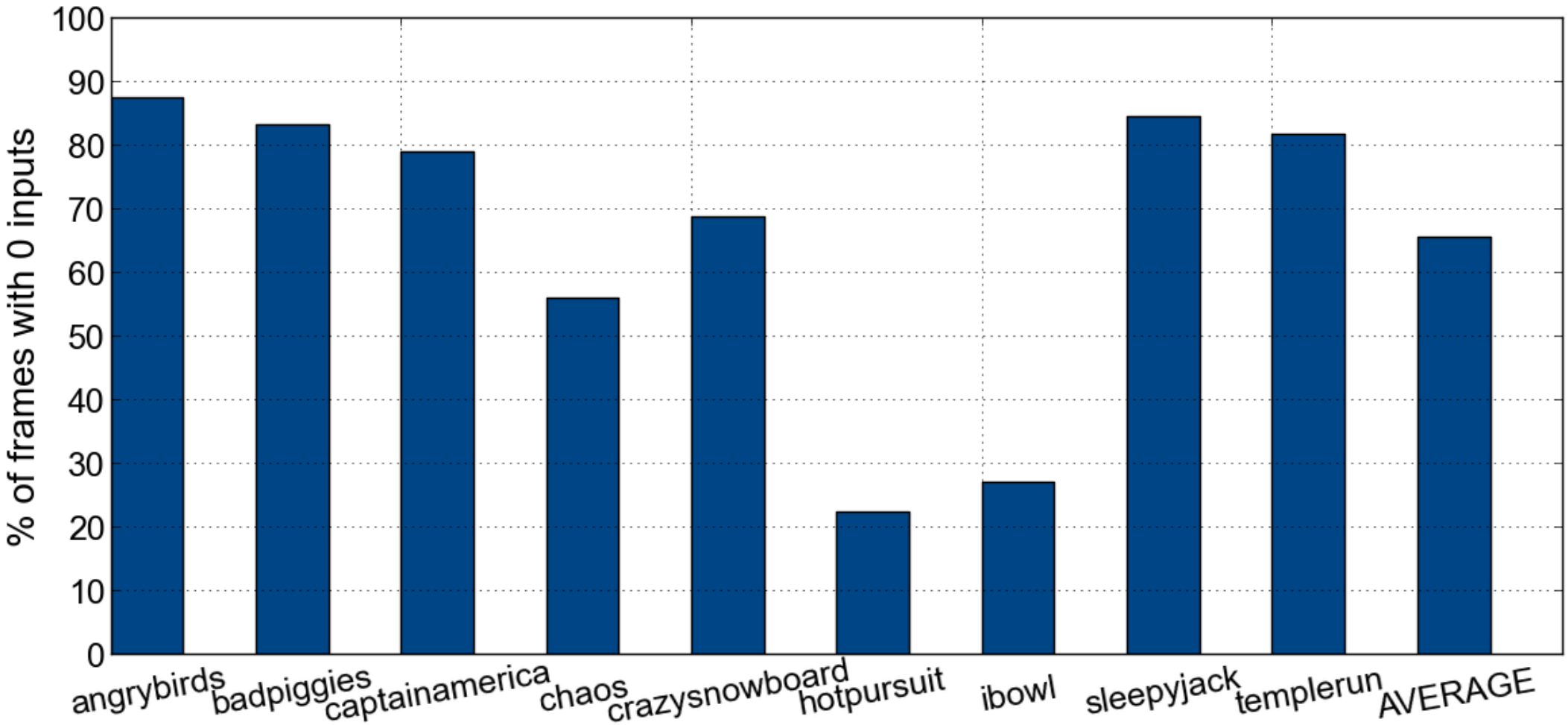
PFR saves bandwidth by overlapping the memory accesses of 2 consecutive frames

Parallel Frame Rendering



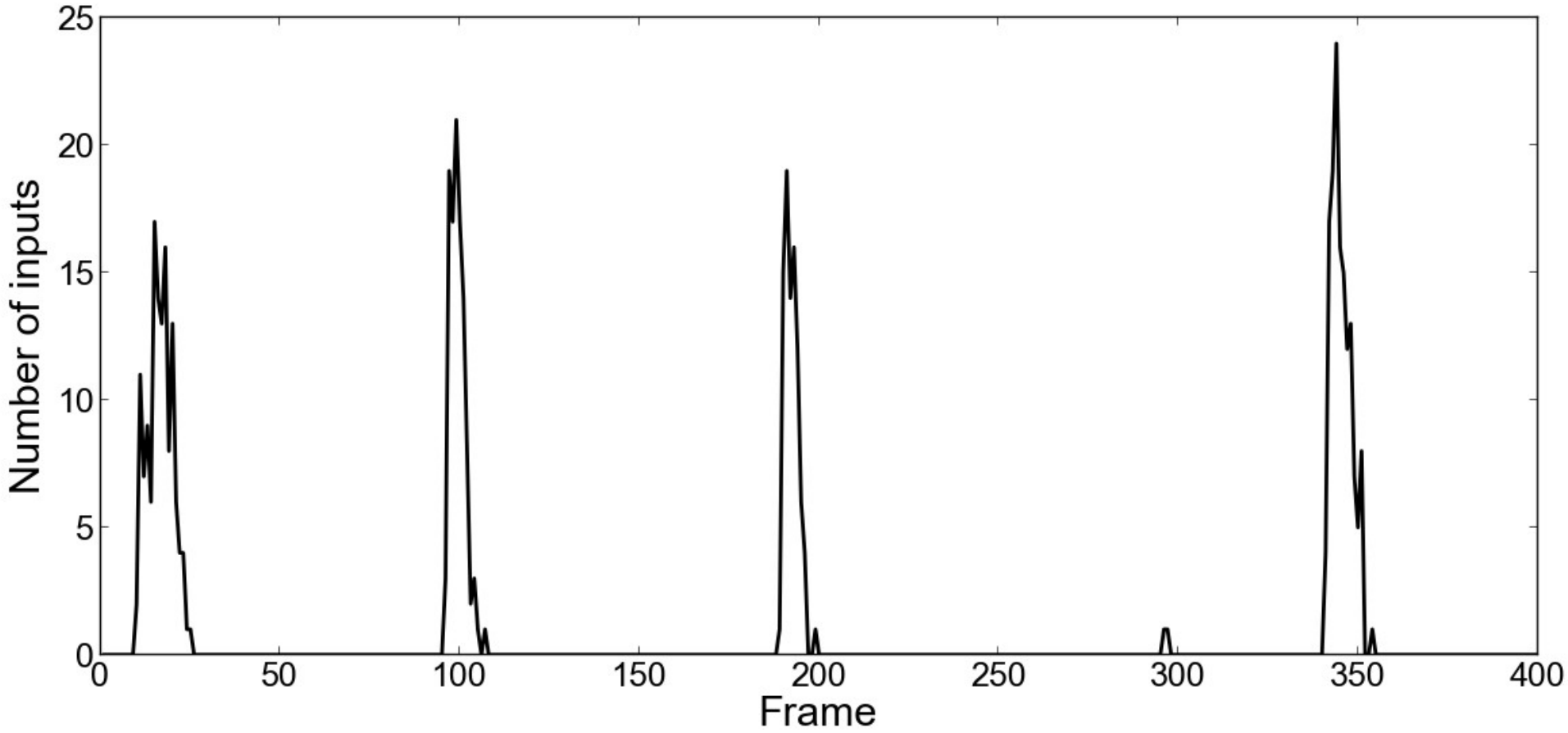
PFR saves bandwidth by overlapping the memory accesses of 2 consecutive frames

User Inputs in Mobile Games

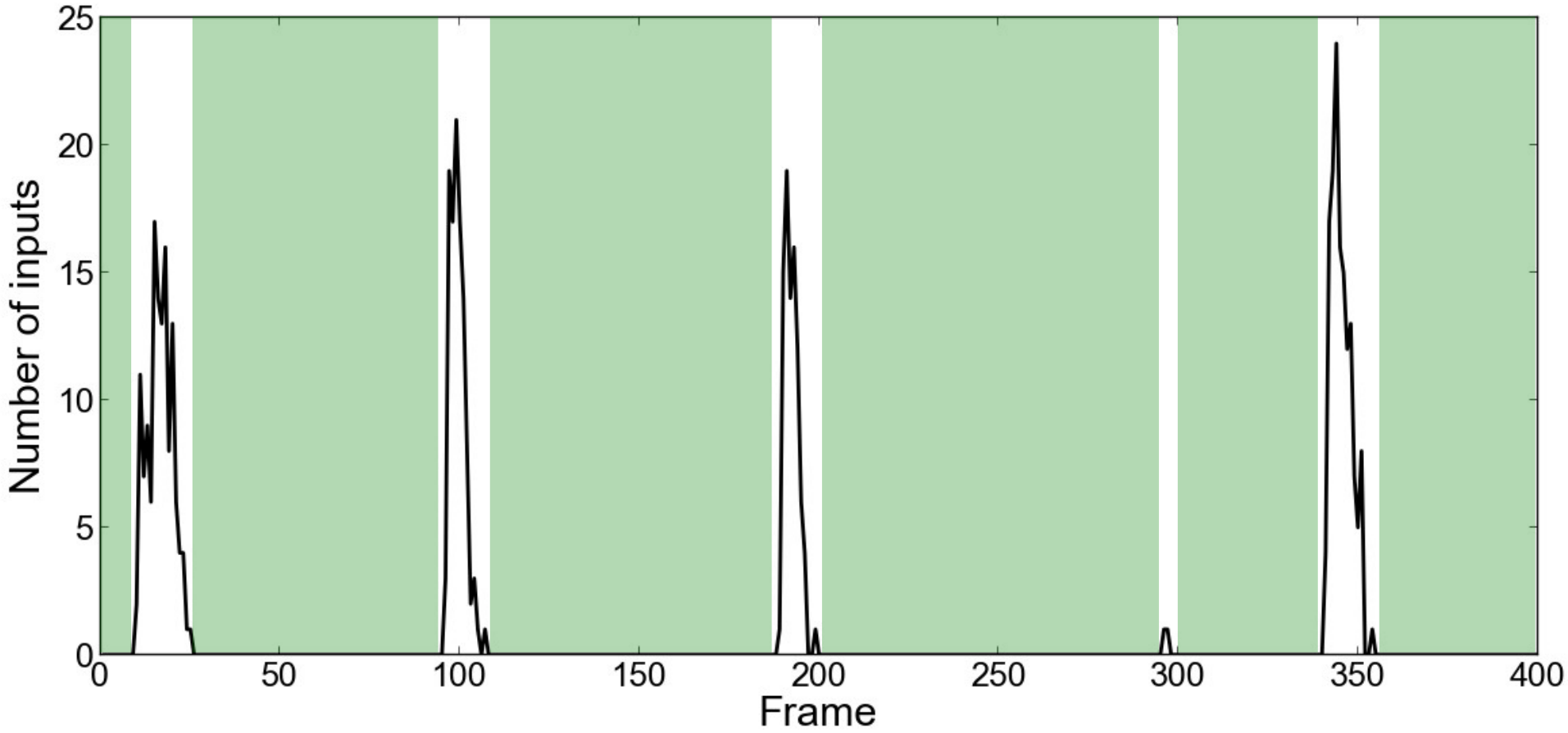


The user does not provide any input most of the time, except in *hotpursuit* and *ibowl*

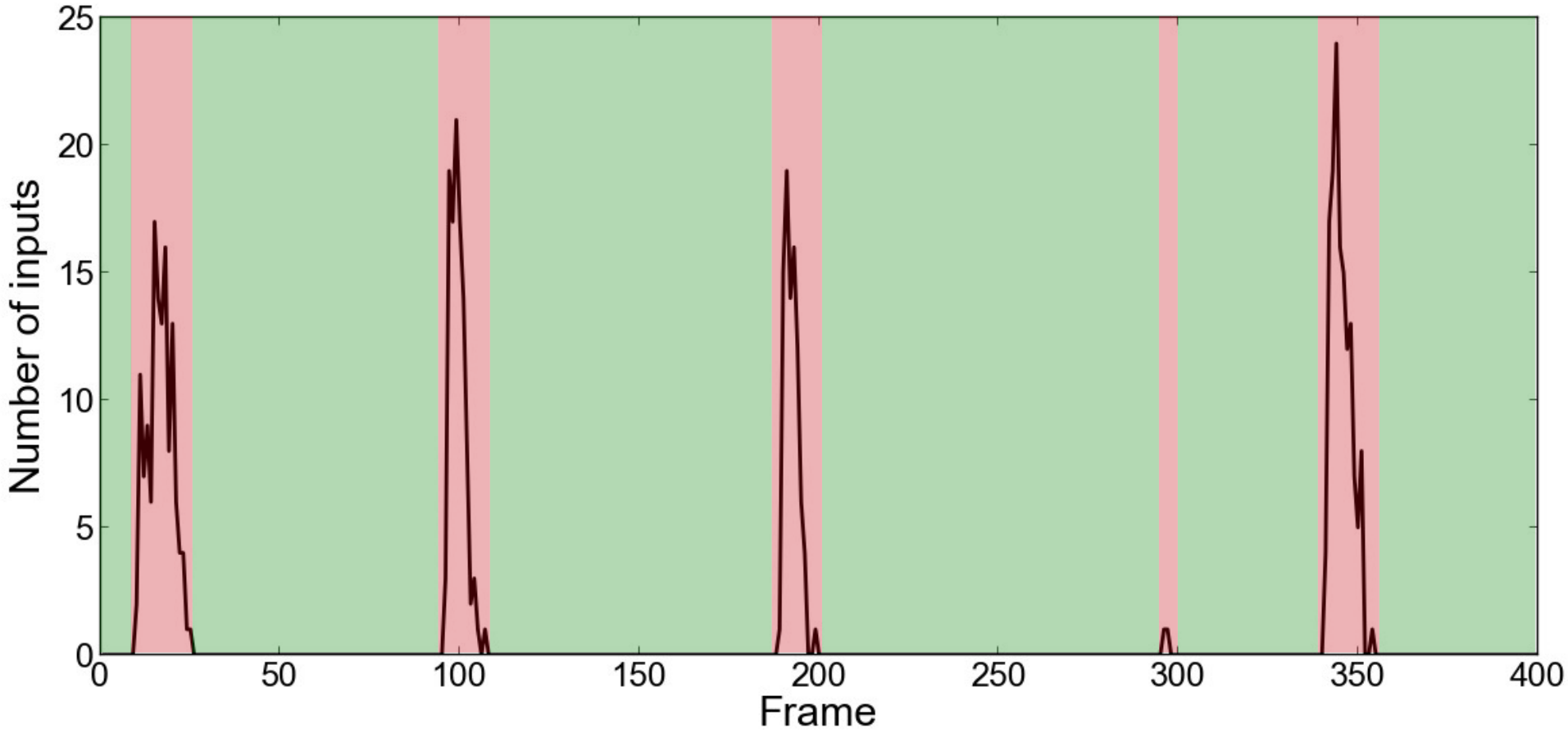
Reactive-PFR



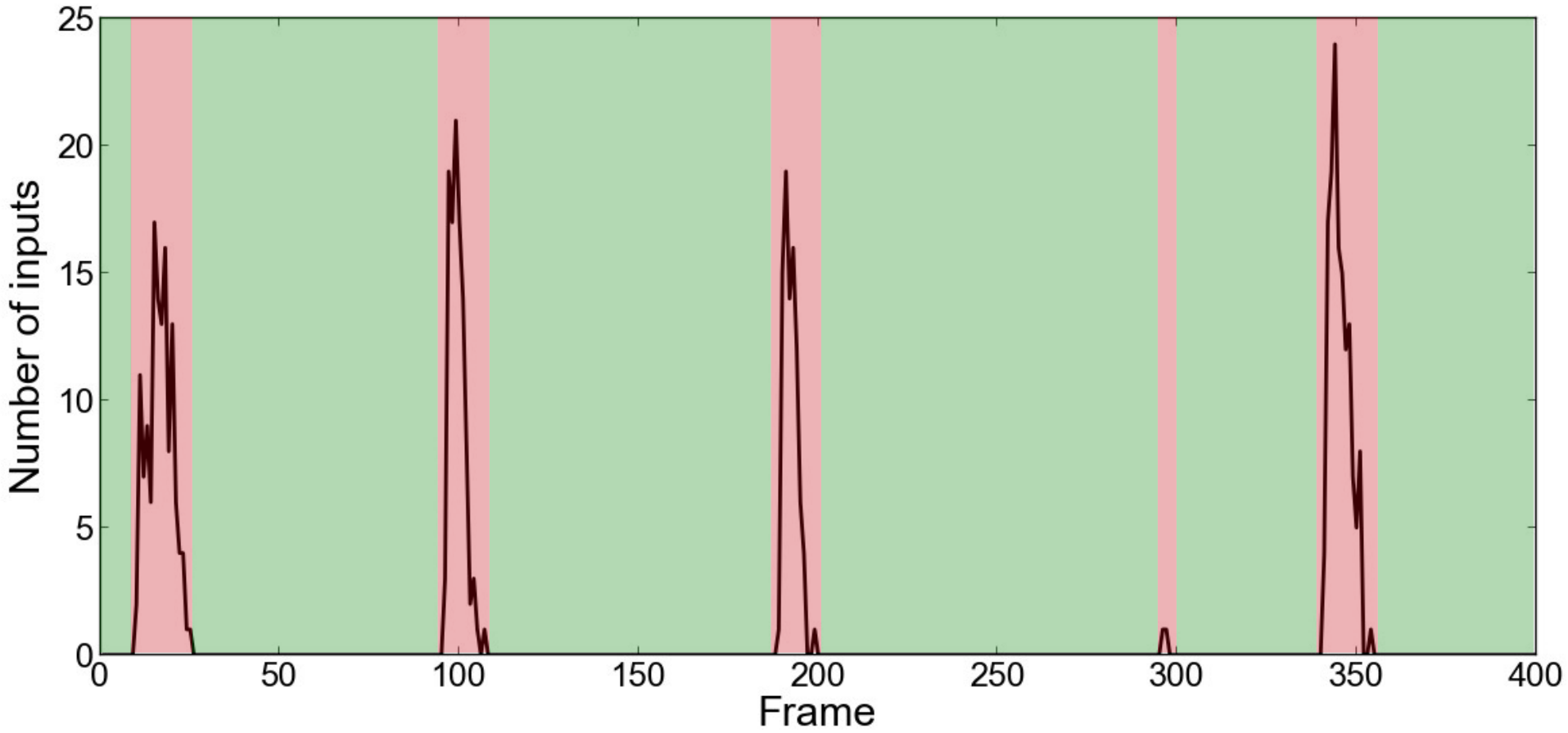
Reactive-PFR

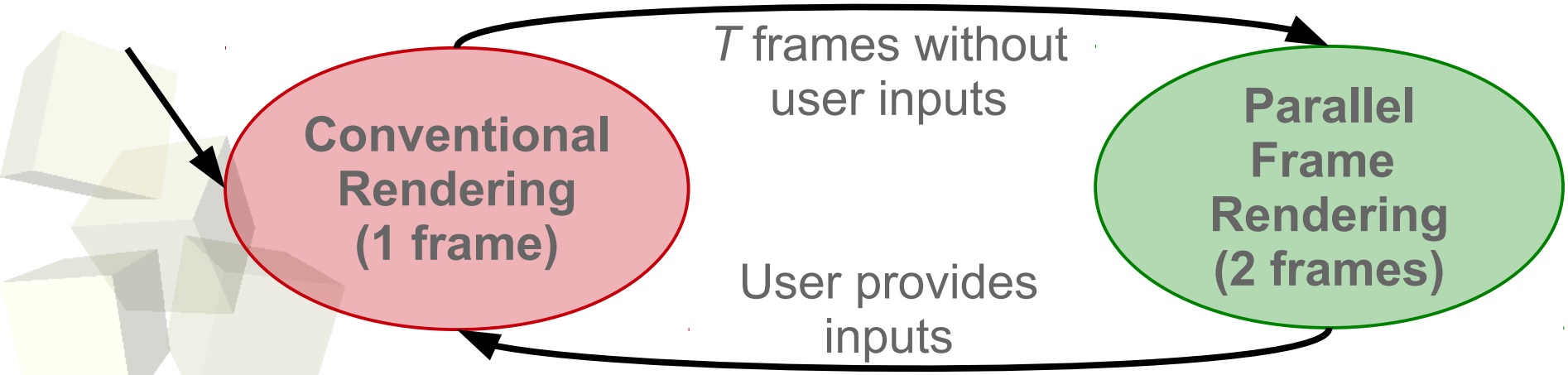


Reactive-PFR

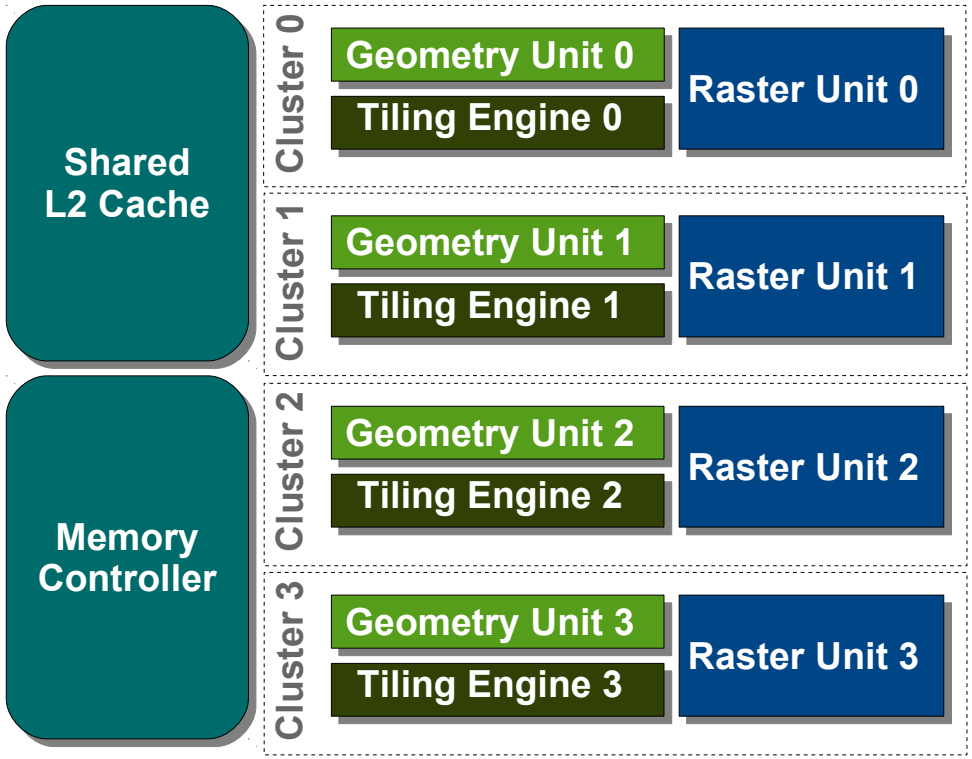


Reactive-PFR

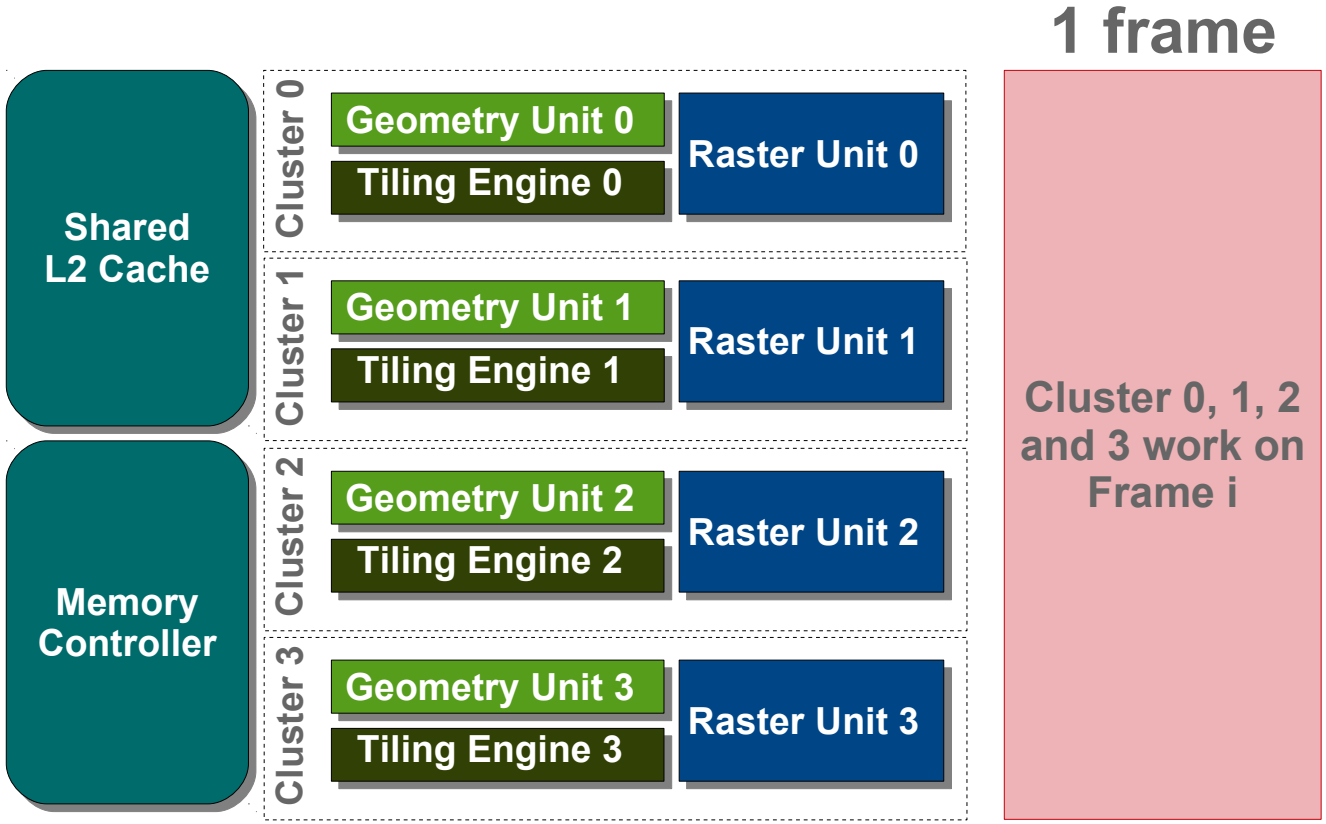




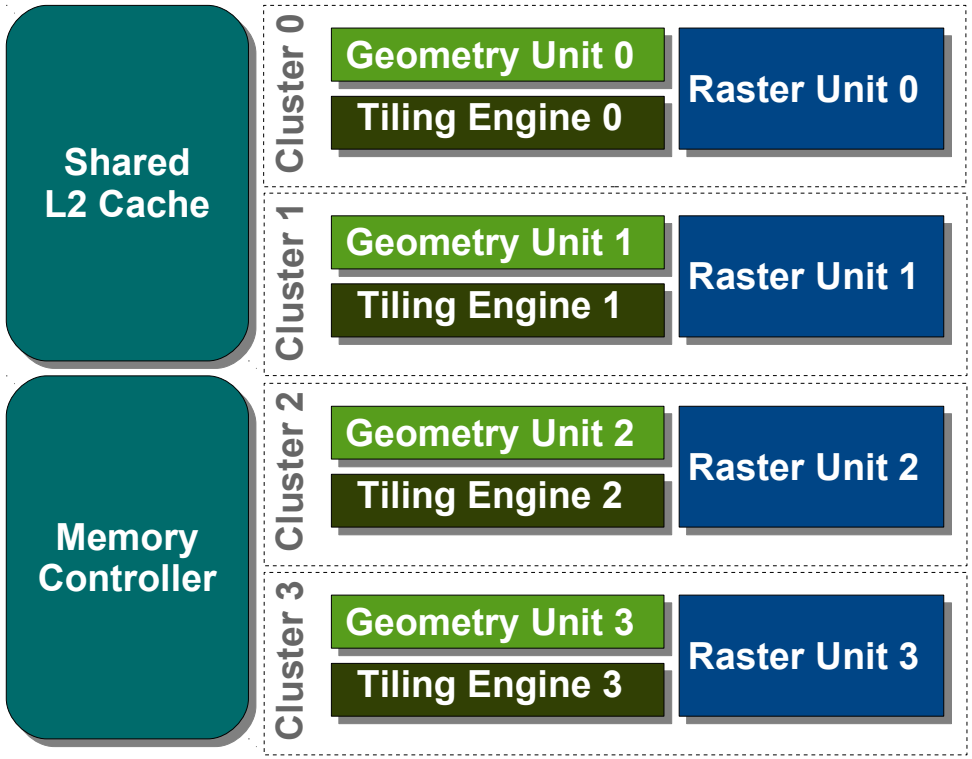
N-Frames Reactive-PFR



N-Frames Reactive-PFR

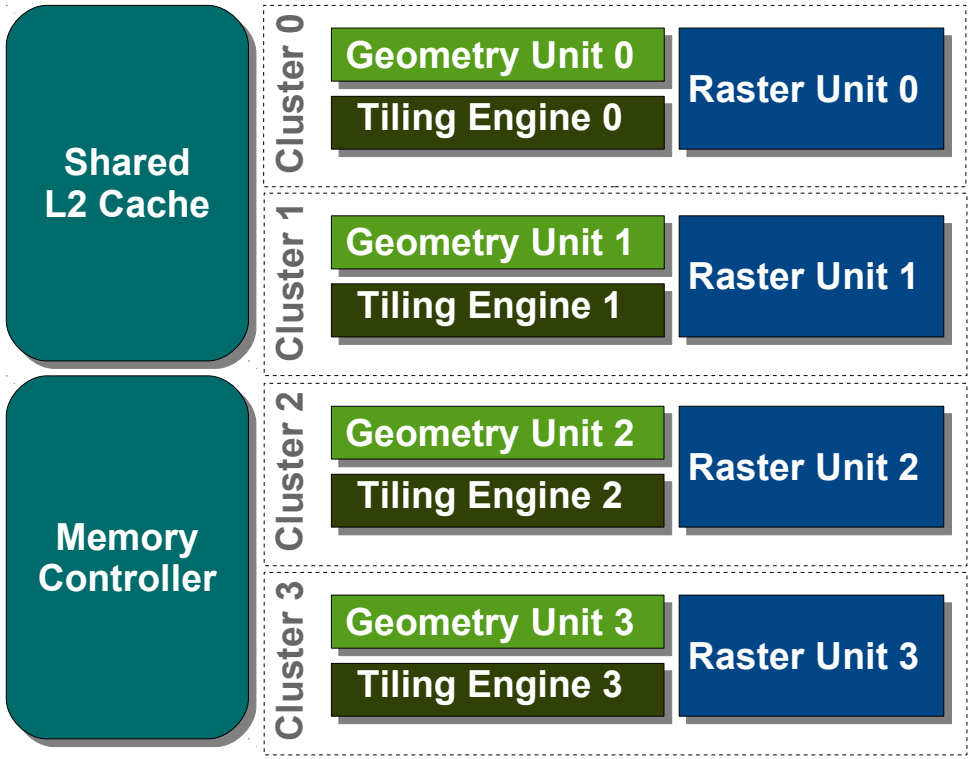


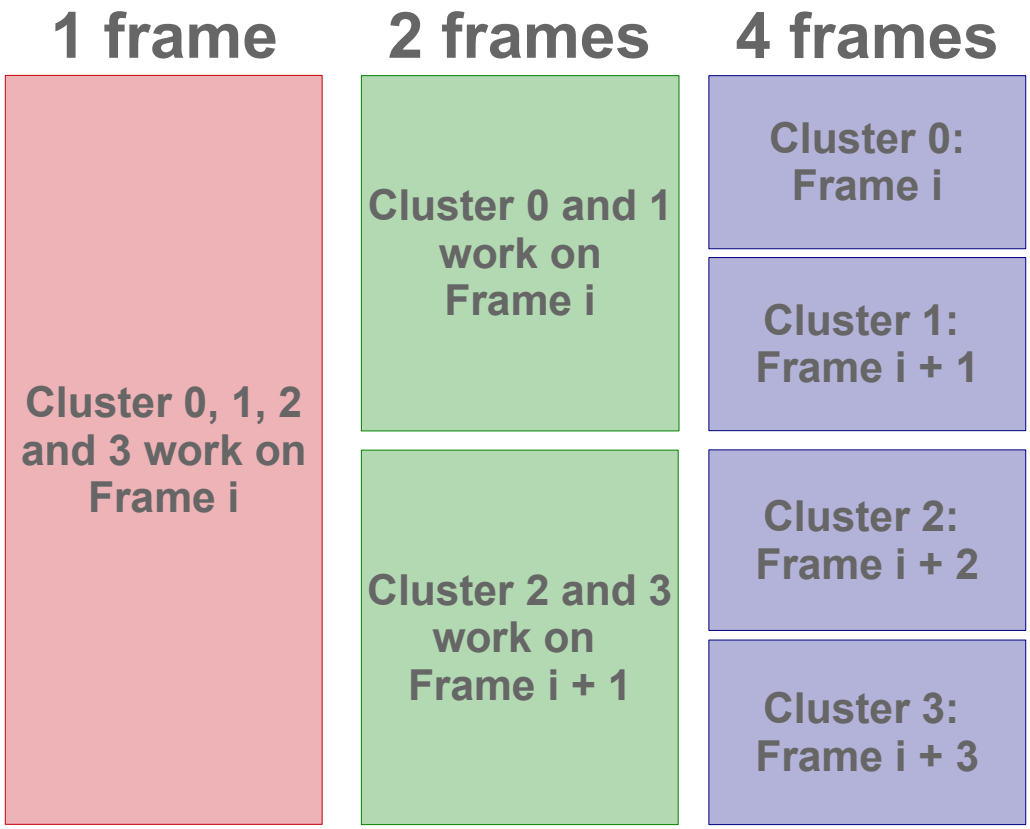
N-Frames Reactive-PFR



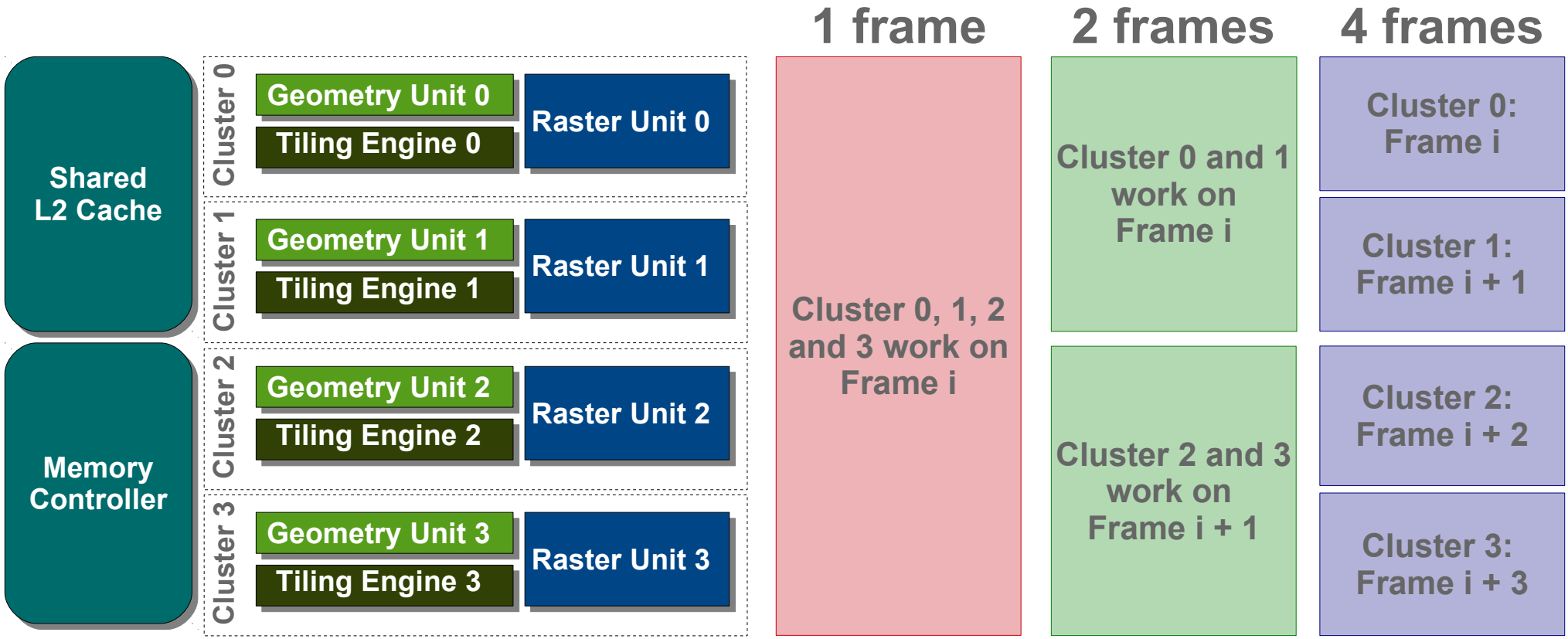
1 frame	2 frames
Cluster 0, 1, 2 and 3 work on Frame i	Cluster 0 and 1 work on Frame i
	Cluster 2 and 3 work on Frame i + 1

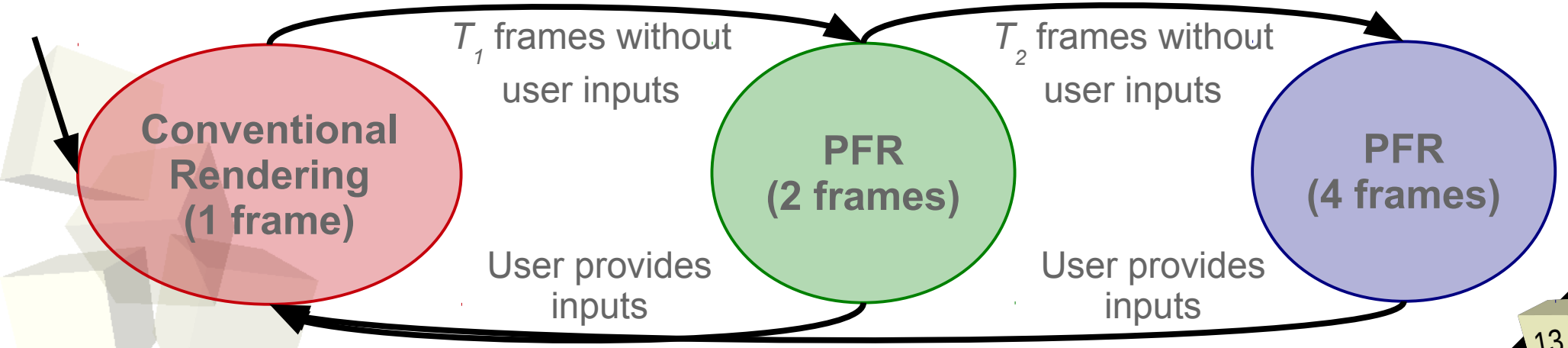
N-Frames Reactive-PFR





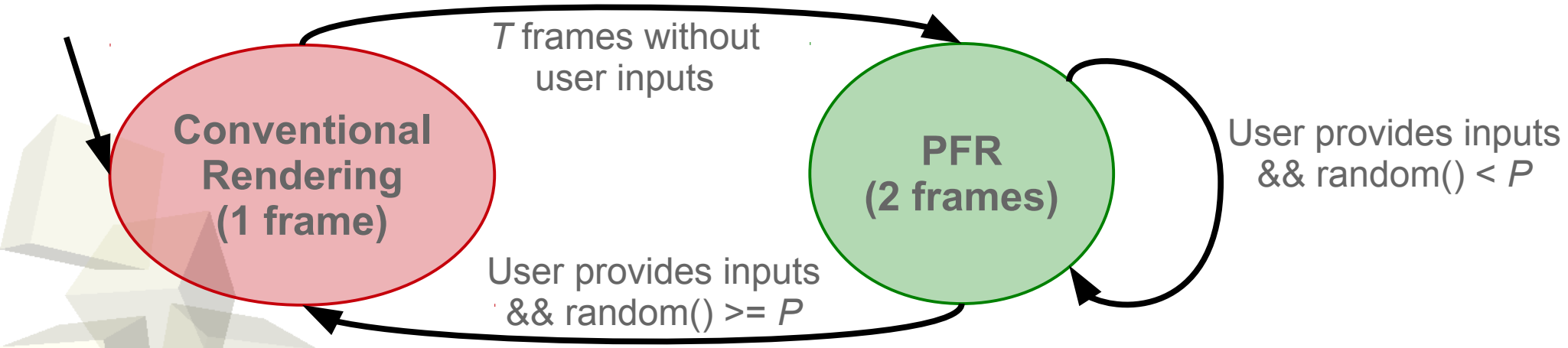
N-Frames Reactive-PFR





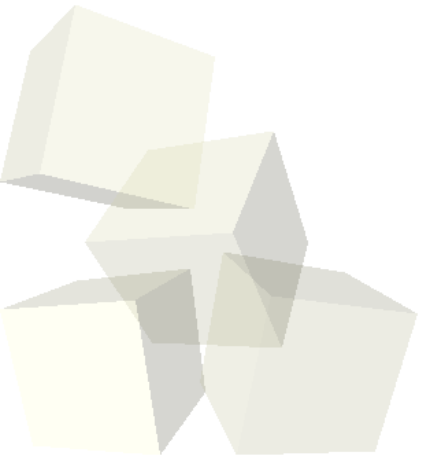
Delay Randomly PFR

- PFR delays all user inputs
  - Big bandwidth savings
  - Worse responsiveness
- R-PFR and NR-PFR do not delay any input
  - Same responsiveness than conventional GPUs
  - Smaller bandwidth savings
- Delay Randomly PFR (DR-PFR)
  - Delay a percentage, *P*, of randomly selected frames with user inputs



Outline

1. Motivation
2. Conventional Rendering
3. Parallel Frame Rendering
4. Experimental Results
5. Conclusions



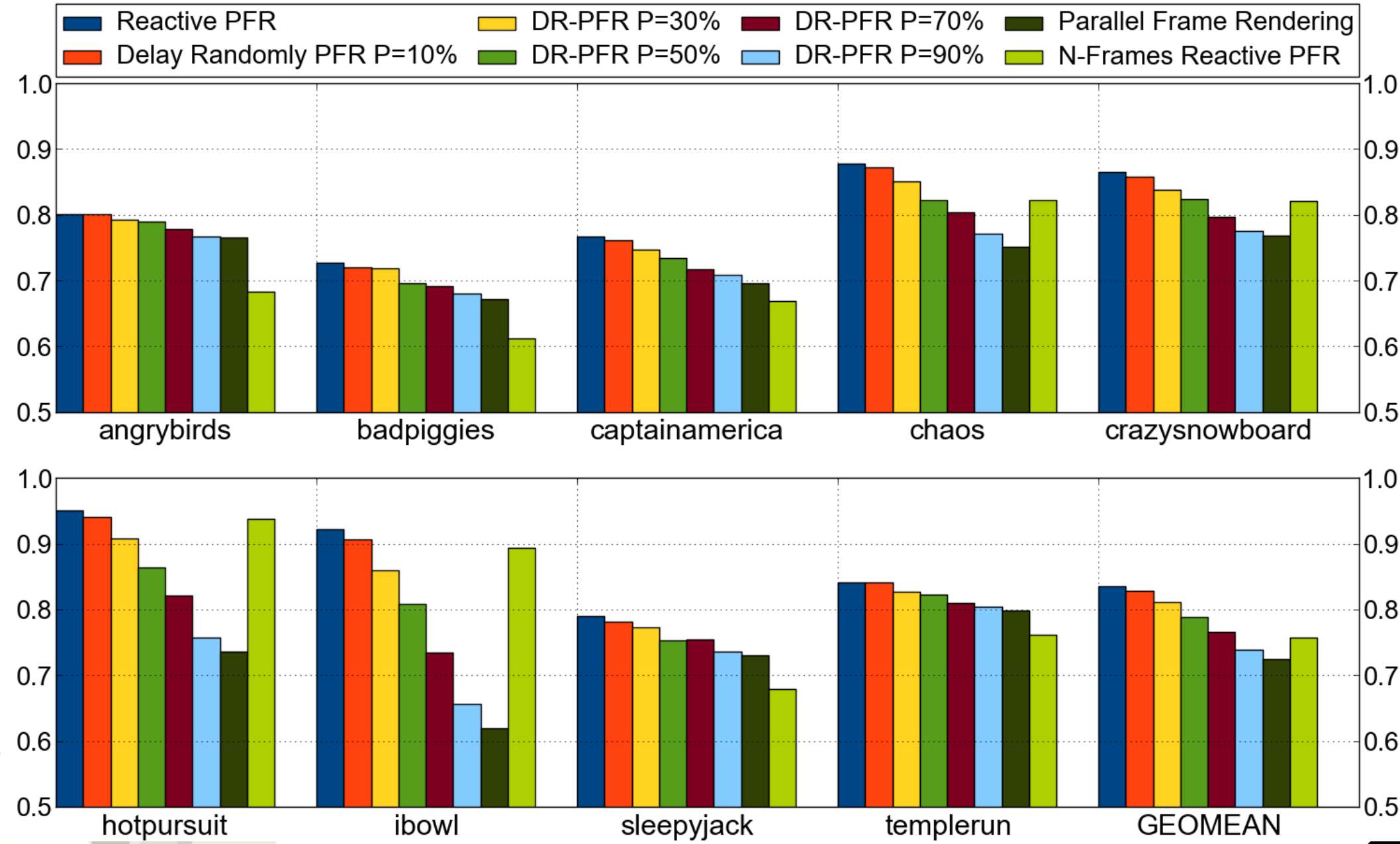
Evaluation Methodology

- TEAPOT simulator
  - Runs unmodified Android applications
  - Mobile GPU timing simulator (TBDR)
  - Power estimations (McPAT)
  - “TEAPOT: A Toolset for Evaluating Performance, Power and Image Quality on Mobile Graphics Systems”. ICS 2013.
- Workloads
  - 2D games: angrybirds, badpiggies
  - Simple 3D games: crazysnowboard, ibowl, templerun
  - Complex 3D games: captainamerica, chaos, hotpursuit, sleepyjack
- Hardware Parameters

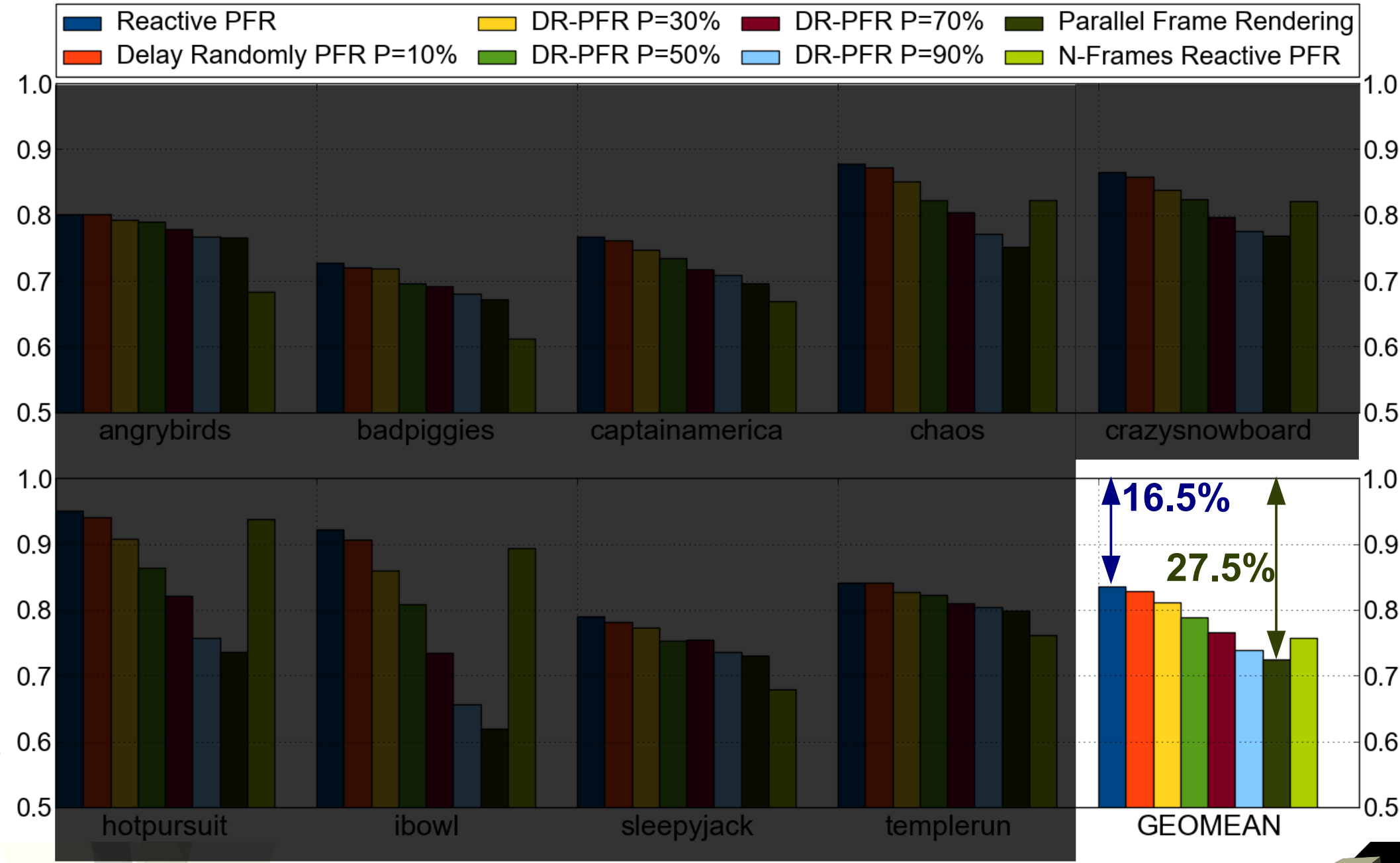
Technology	32 nm	**L2 Cache**	128 KB, 8-way, 12 cycles
Frequency	300 MHz	**Tile Cache**	32 KB, 4-way, 4 cycles
Tile size	16 x 16	**Texture Caches**	8 KB, 2-way, 1 cycle
Screen	800 x 480 (WVGA)	**Main memory**	1 GB, 8 bytes/cycle

	Conv. Rendering	**PFR, R-PFR, DR-PFR**	**NR-PFR**
Num clusters	1	2	4
Raster units/cluster	4	2	1
Vertex Proc./cluster	4	2	1

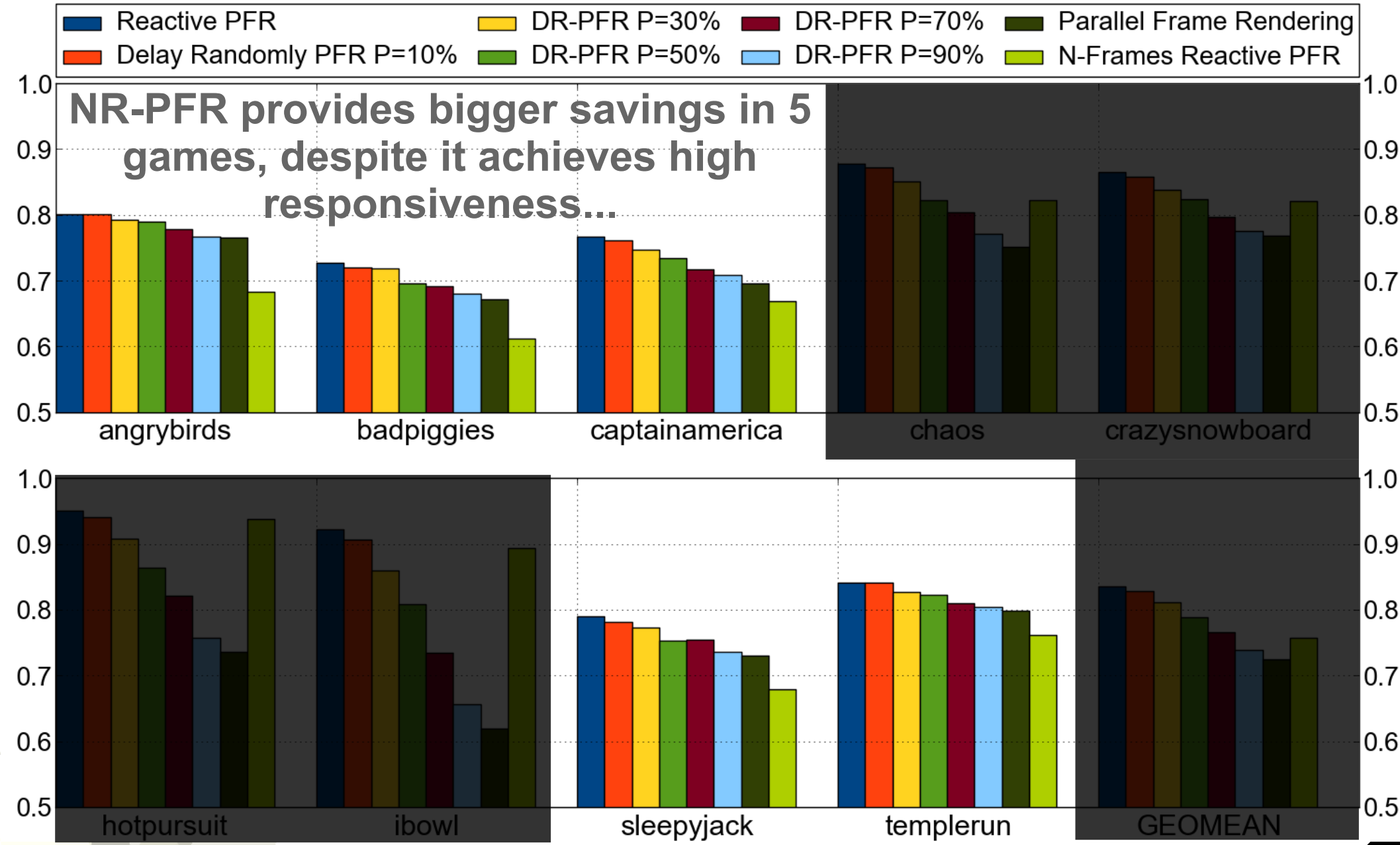
Normalized Memory Traffic



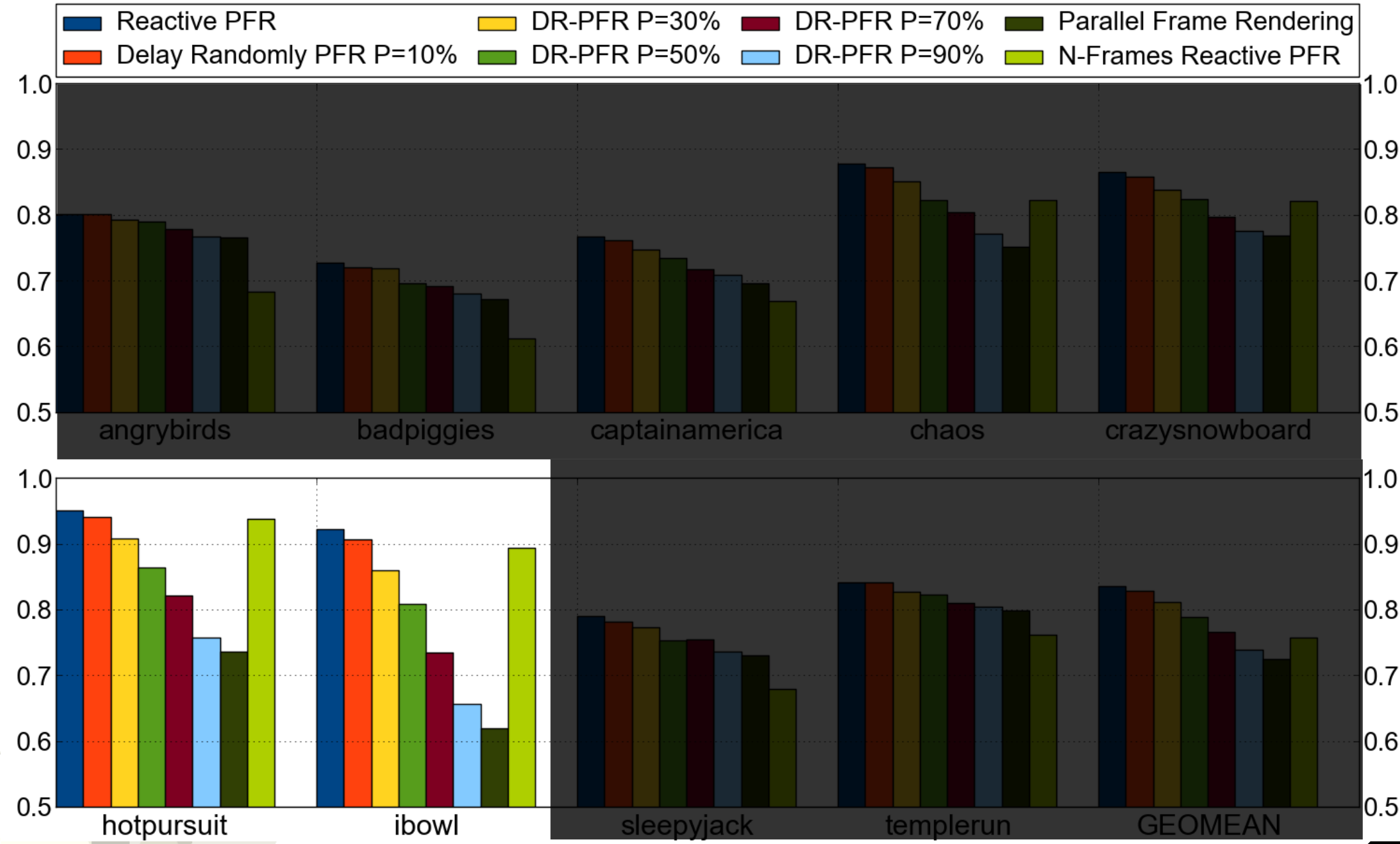
Normalized Memory Traffic



Normalized Memory Traffic

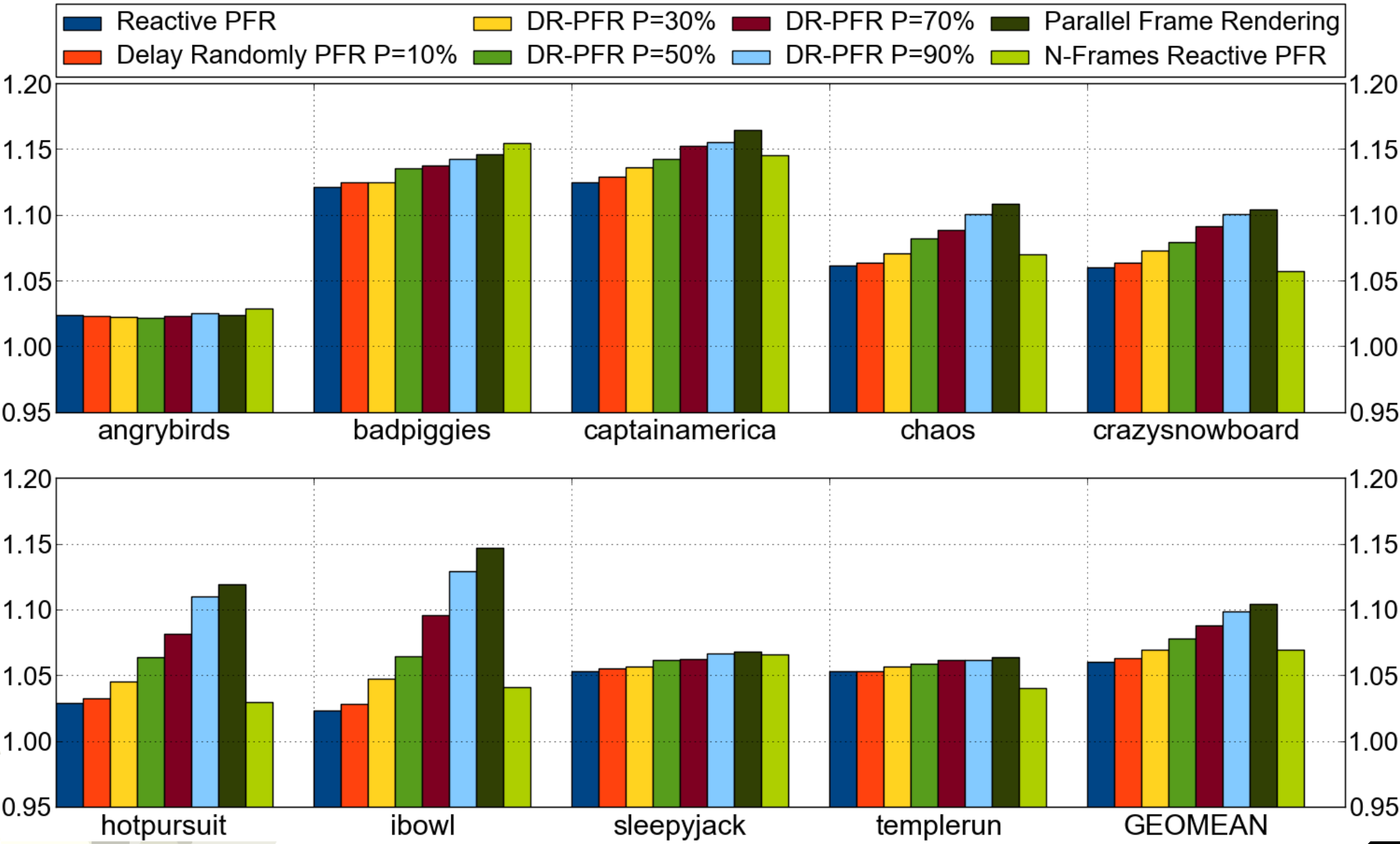


Normalized Memory Traffic



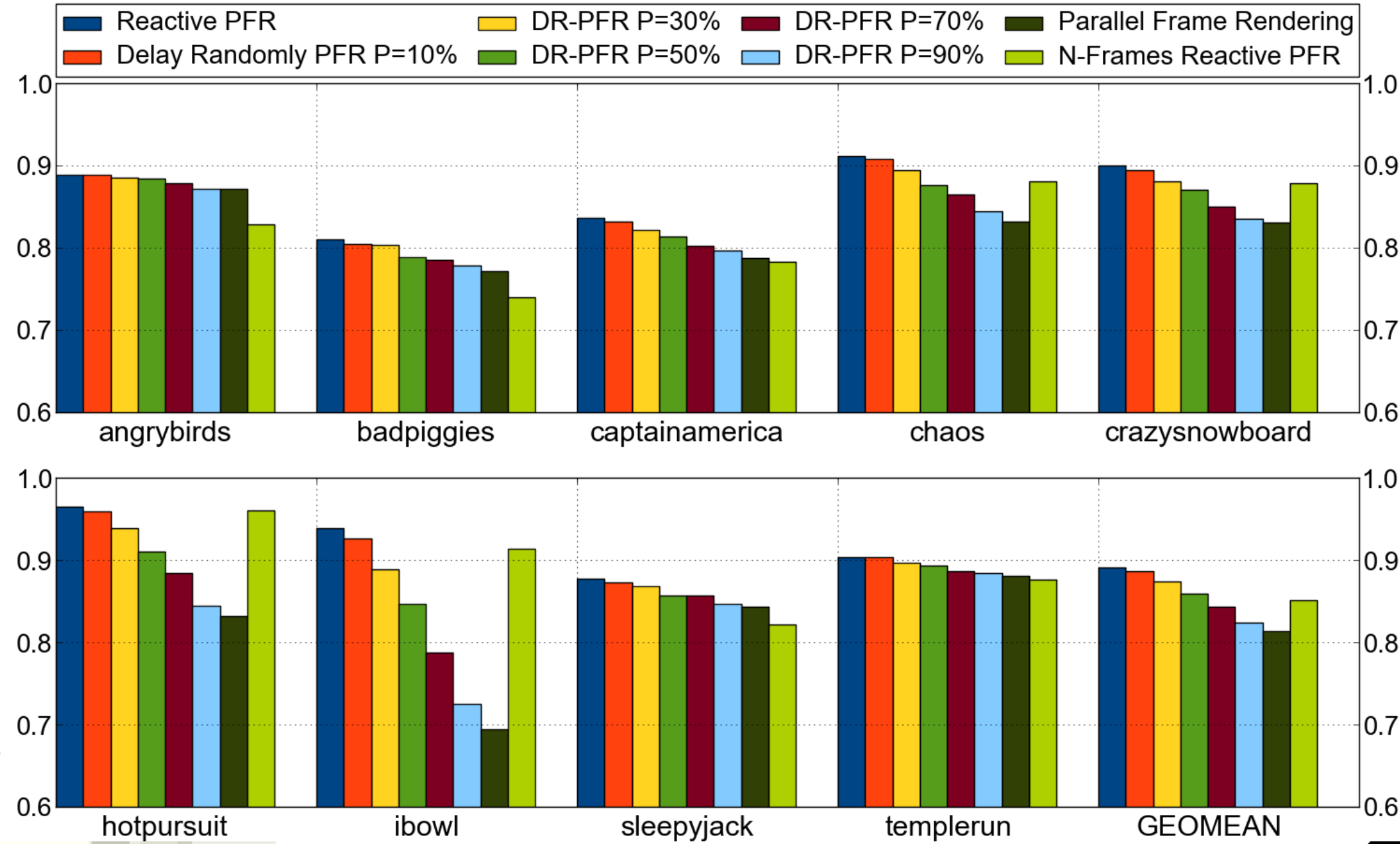
...but the Reactive versions offer modest savings for games with intensive user interaction

Speedup

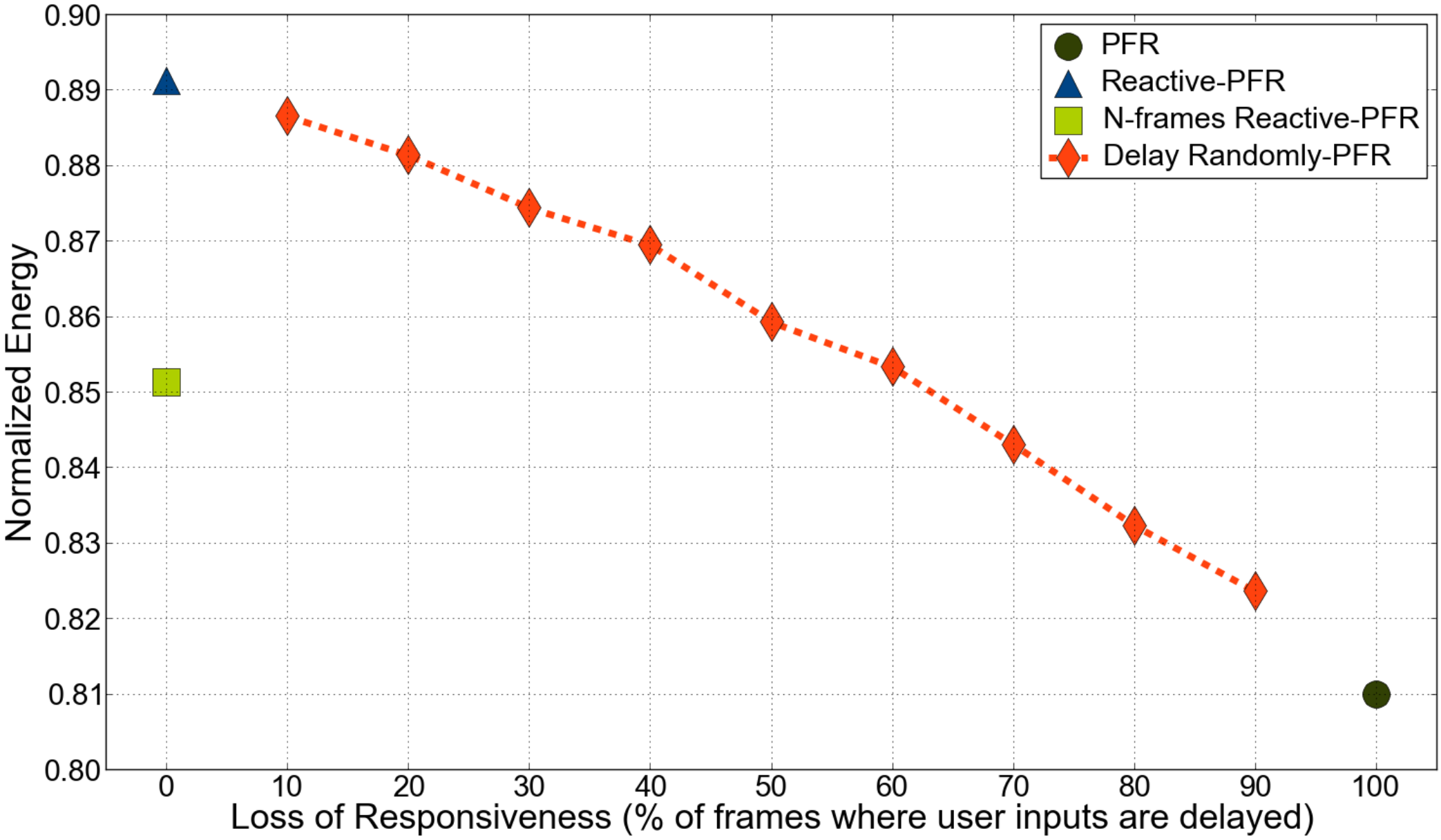


GPUs are bandwidth bound

Normalized Energy

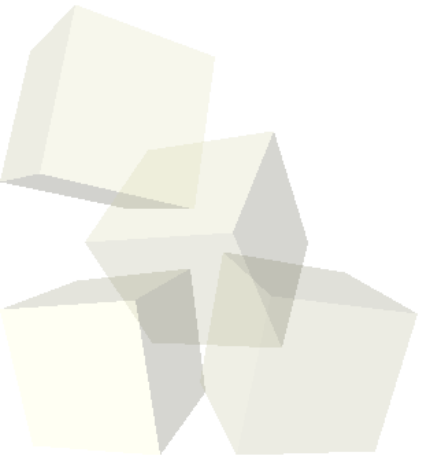


Trading Responsiveness for Energy



Outline

1. Motivation
2. Conventional Rendering
3. Parallel Frame Rendering
4. Experimental Results
5. Conclusions



Conclusions

- Consecutive frames exhibit a high degree of **texture similarity**, since a big percentage of the texture dataset is shared
- Mobile GPU **memory bandwidth** can be significantly reduced by overlapping the memory accesses of 2 consecutive frames, we term this technique as Parallel Frame Rendering (PFR)
- PFR comes at a cost in the **responsiveness** of the system, as the input lag is increased
- **Reactive** versions of PFR are able to adapt to the amount of input provided by the user, achieving high responsiveness when necessary
- The experimental results show that PFR and its different variations provide consistent **performance** and **energy** improvements

Parallel Frame Rendering: Trading Responsiveness for Energy on a Mobile GPU

Jose-Maria Arnau[1]
jarnau@ac.upc.edu

Joan-Manuel Parcerisa[1]
jmanel@ac.upc.edu

Polychronis Xekalakis[2]
polychronis.xekalakis@intel.com

[1]Universitat Politecnica de Catalunya

[2]Intel Labs, Intel Corporation

09 / September / 2013