41st International Symposium on Computer Architecture

Eliminating Redundant Fragment Shader Executions on a Mobile GPU via Hardware Memoization

Jose-Maria Arnau
jarnau@ac.upc.edu

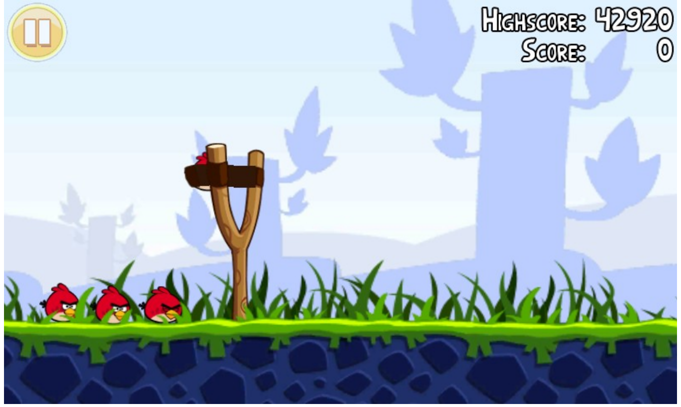
Joan-Manuel Parcerisa
jmanel@ac.upc.edu

Universitat Politecnica de Catalunya

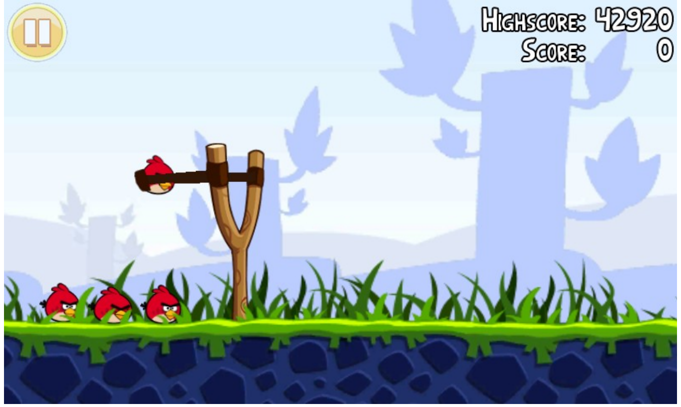
Polychronis Xekalakis
polychronis.xekalakis@intel.com
Intel Corporation

Redundancy in Mobile Games

Frame i

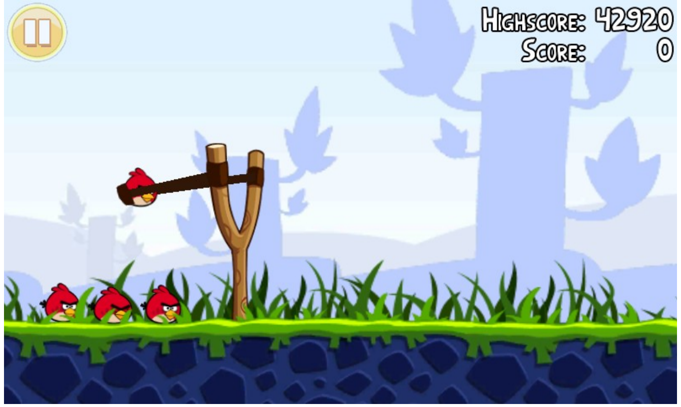


Frame i + 1



98% of fragments already computed in prior frame

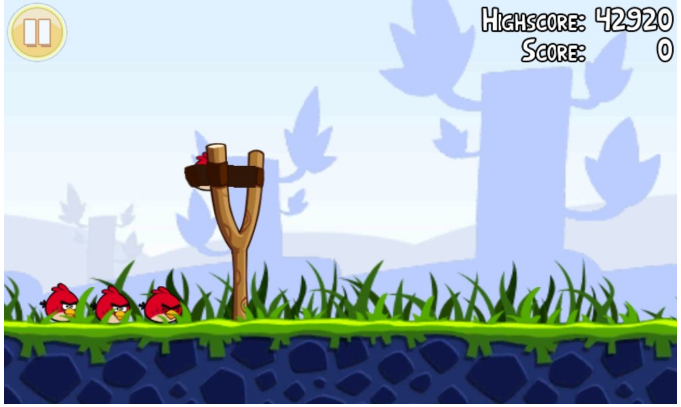
Frame i + 2



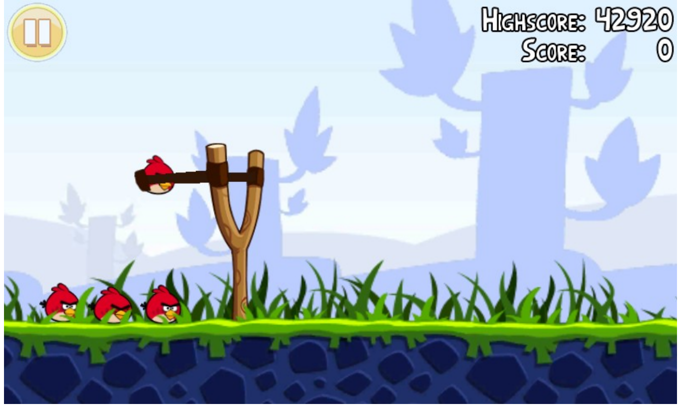
99% of fragments already computed in prior frame

Redundancy in Mobile Games

Frame i

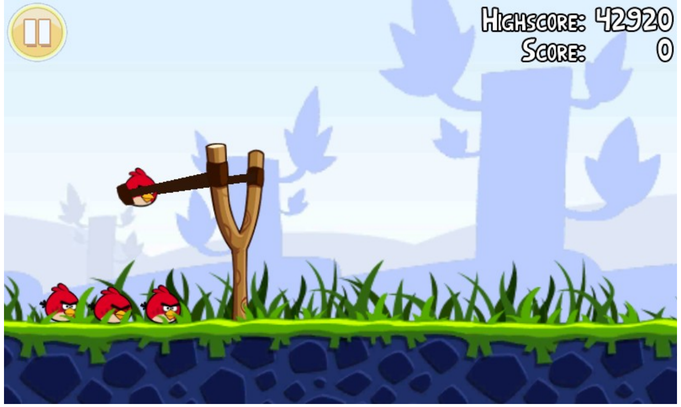


Frame i + 1

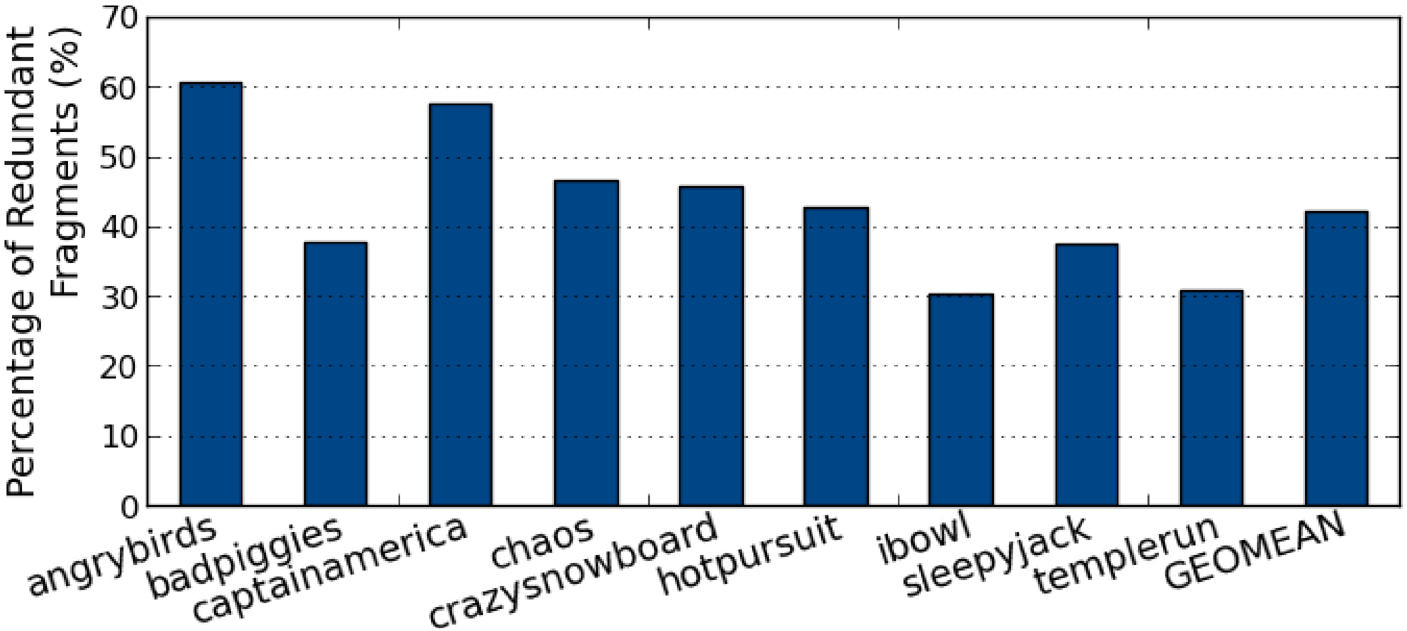


98% of fragments already computed in prior frame

Frame i + 2



99% of fragments already computed in prior frame



42.7% of Fragment Program executions are redundant on average

Redundant fragment has:

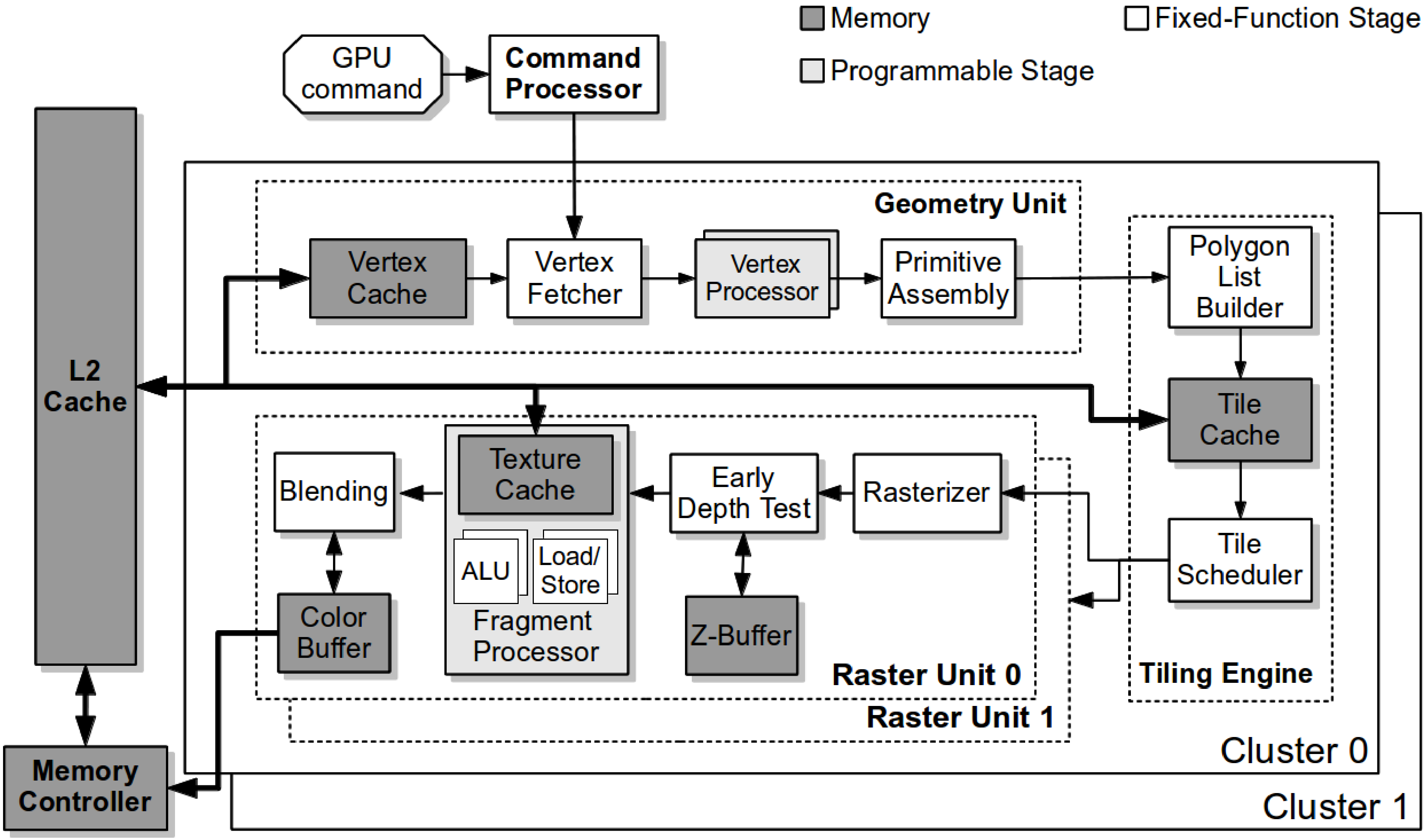
- same inputs
- same fragment program
- same output result

than a previous fragment

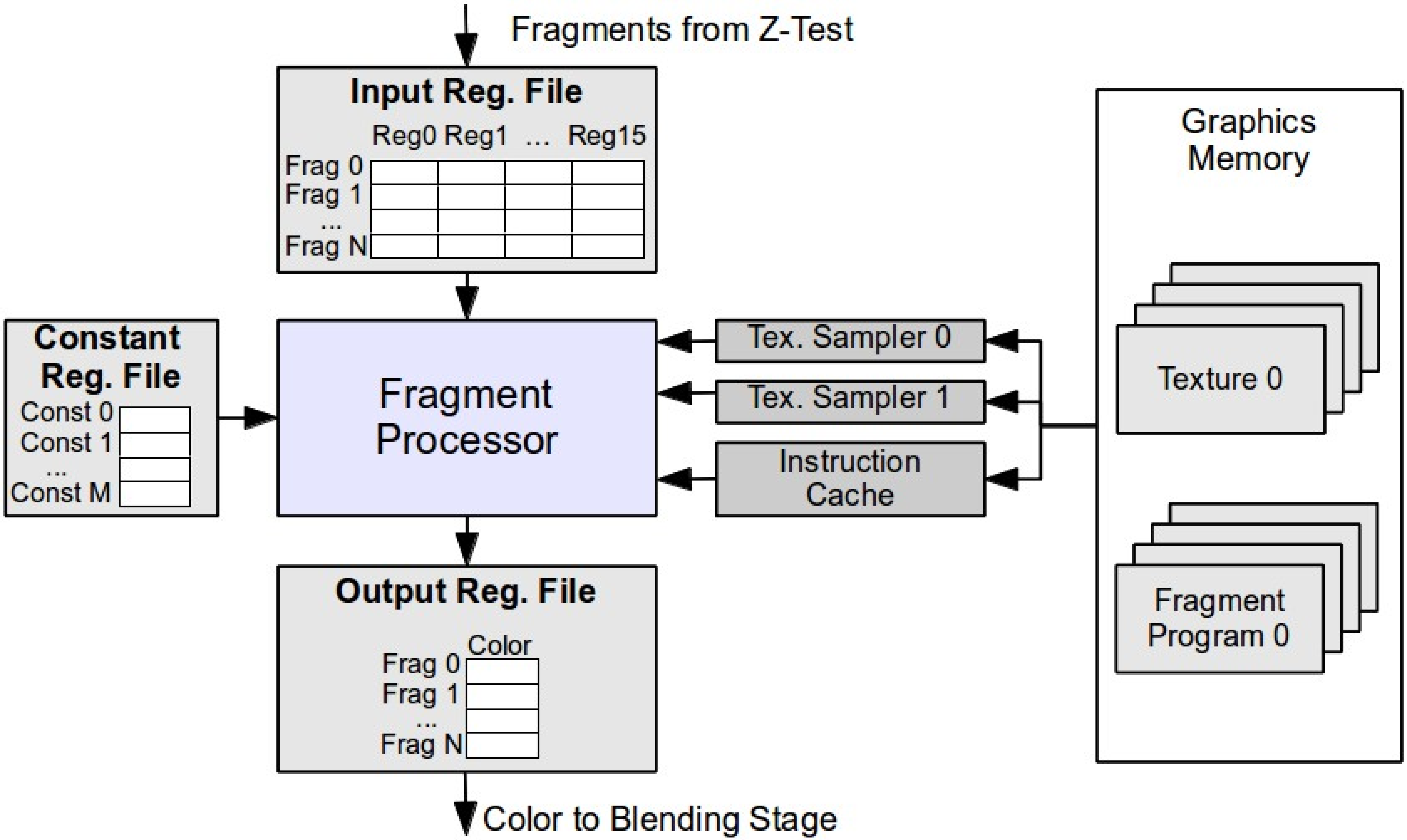
Outline

1. Motivation
2. **Fragment Stage**
3. Redundancy and Memoization
4. Memoization in a Mobile GPU
5. Experimental Results
6. Conclusions

Assumed Baseline Mobile GPU



Fragment Processing Stage



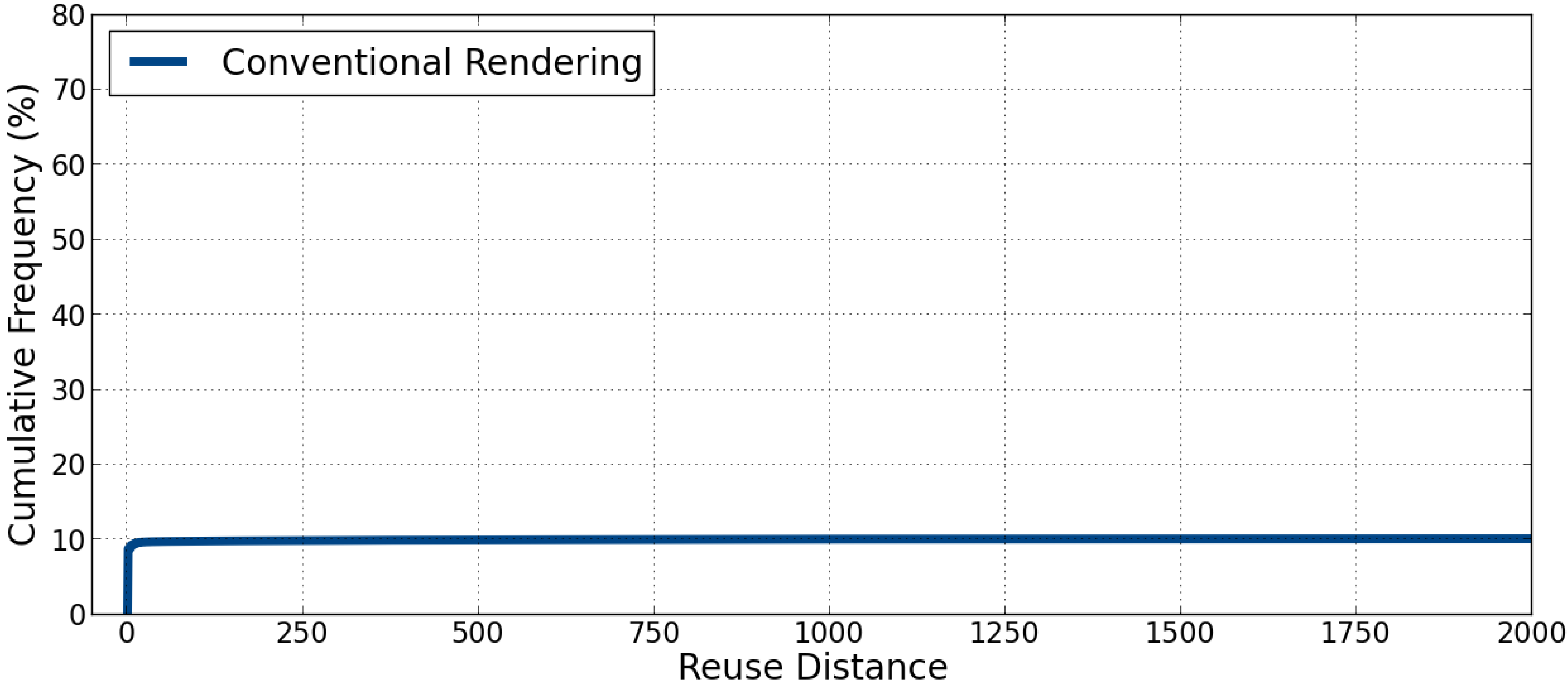
Outline

1. Motivation
2. Fragment Stage
3. **Redundancy and Memoization**
4. Memoization in a Mobile GPU
5. Experimental Results
6. Conclusions

Hardware Memoization Requirements

- **Redundancy**
  - 42.7% of the fragments are redundant on average
- **Locality**
  - Reuse distance:
    - Number of unique fragments processed between two occurrences of the same fragment
- **Complexity**
  - Cost of accessing HW structures for memoization must be smaller than the cost of executing Fragment Program
- **Referential Transparency**
  - Side effects
  - Same input values must always produce same output

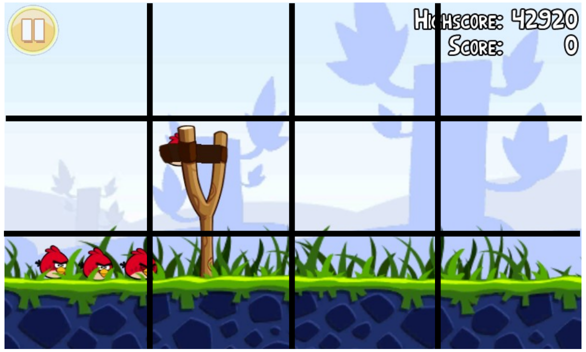
Reuse Distance



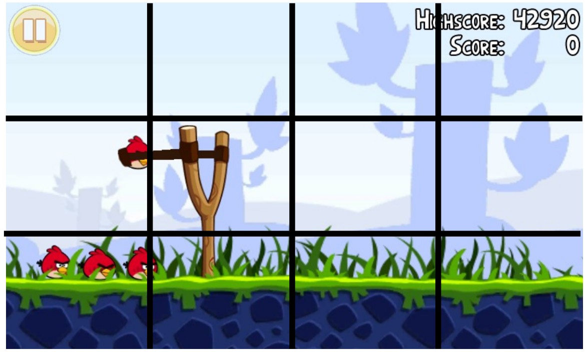
Only 10% of redundant fragments can be captured with realistic HW constraints

Reuse Distance

Frame i



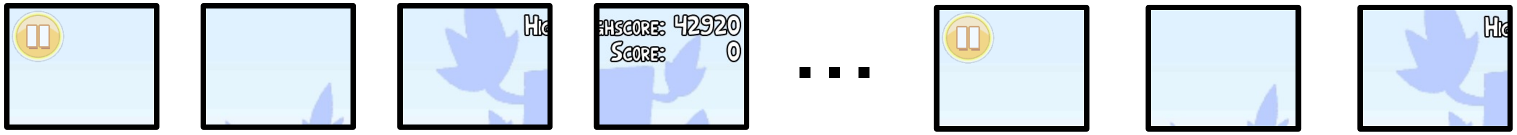
Frame i+1



Conventional Rendering

GPU

Cluster 0 and 1 render same frame



Tile 0 Frame i

Tile 1 Frame i

Tile 2 Frame i

Tile 3 Frame i

Tile 0 Frame i+1

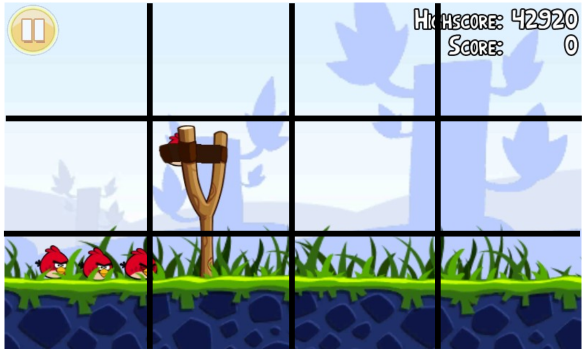
Tile 1 Frame i+1

Tile 2 Frame i+1

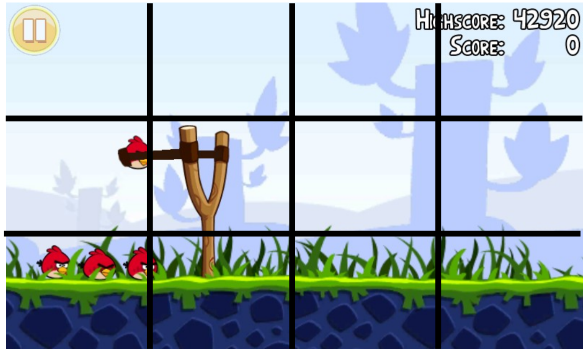
Time →

Reuse Distance

Frame i



Frame i+1

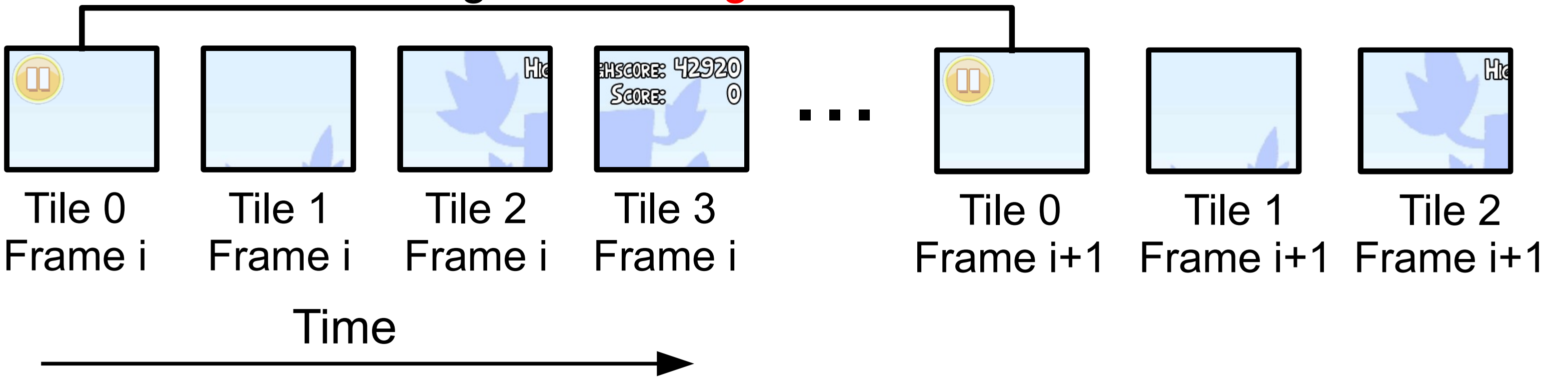


Conventional Rendering

GPU

Cluster 0 and 1 render same frame

Redundant fragments at big distances



Tile 0 Frame i

Tile 1 Frame i

Tile 2 Frame i

Tile 3 Frame i

...

Tile 0 Frame i+1

Tile 1 Frame i+1

Tile 2 Frame i+1

Time

Reuse Distance

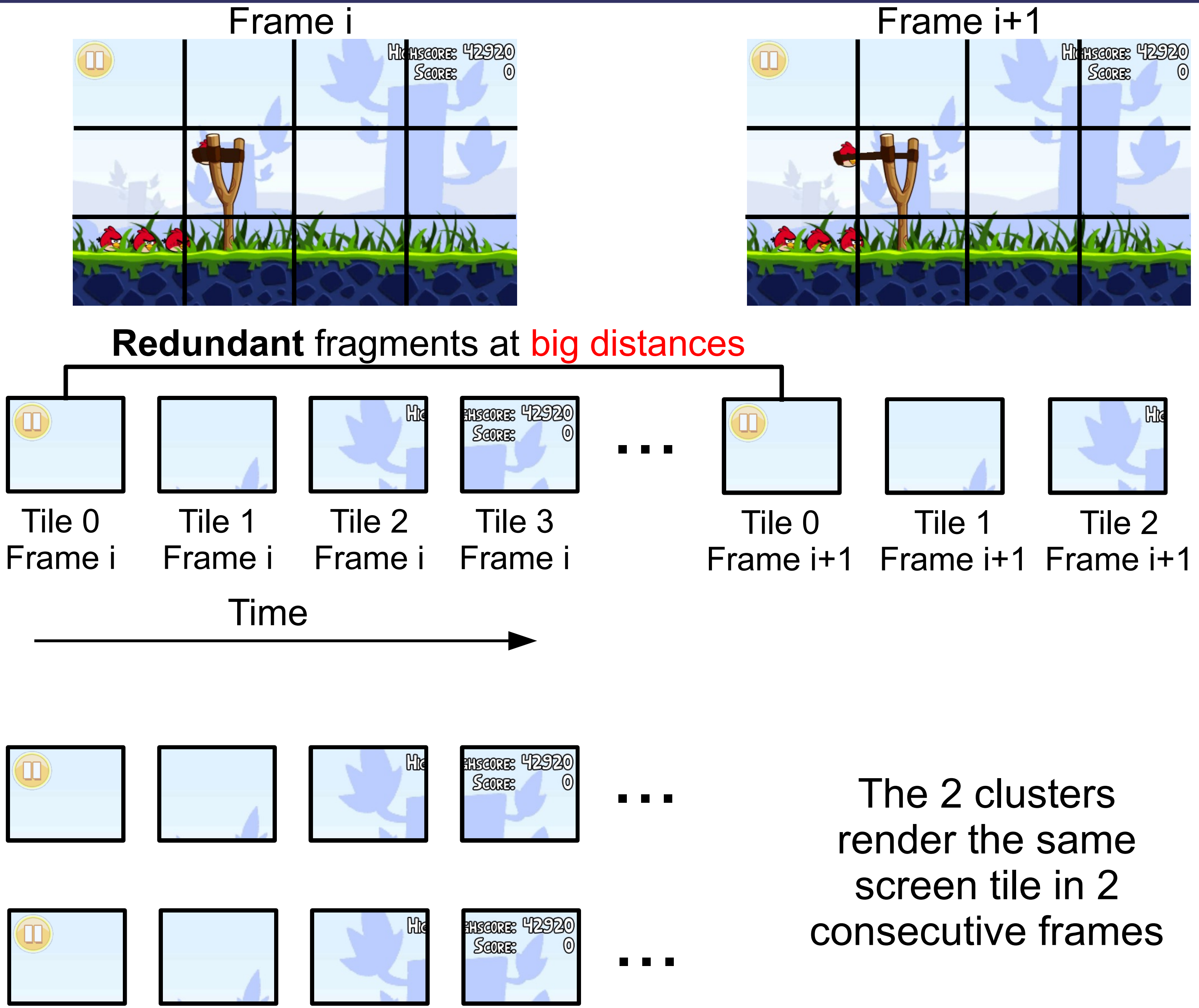
Conventional Rendering

GPU
Cluster 0 and 1 render same frame

Parallel Frame Rendering

GPU Cluster 0 renders frame i

GPU Cluster 1 renders frame i+1



Frame i

Frame i+1

Redundant fragments at big distances

Tile 0 Frame i

Tile 1 Frame i

Tile 2 Frame i

Tile 3 Frame i

...

Tile 0 Frame i+1

Tile 1 Frame i+1

Tile 2 Frame i+1

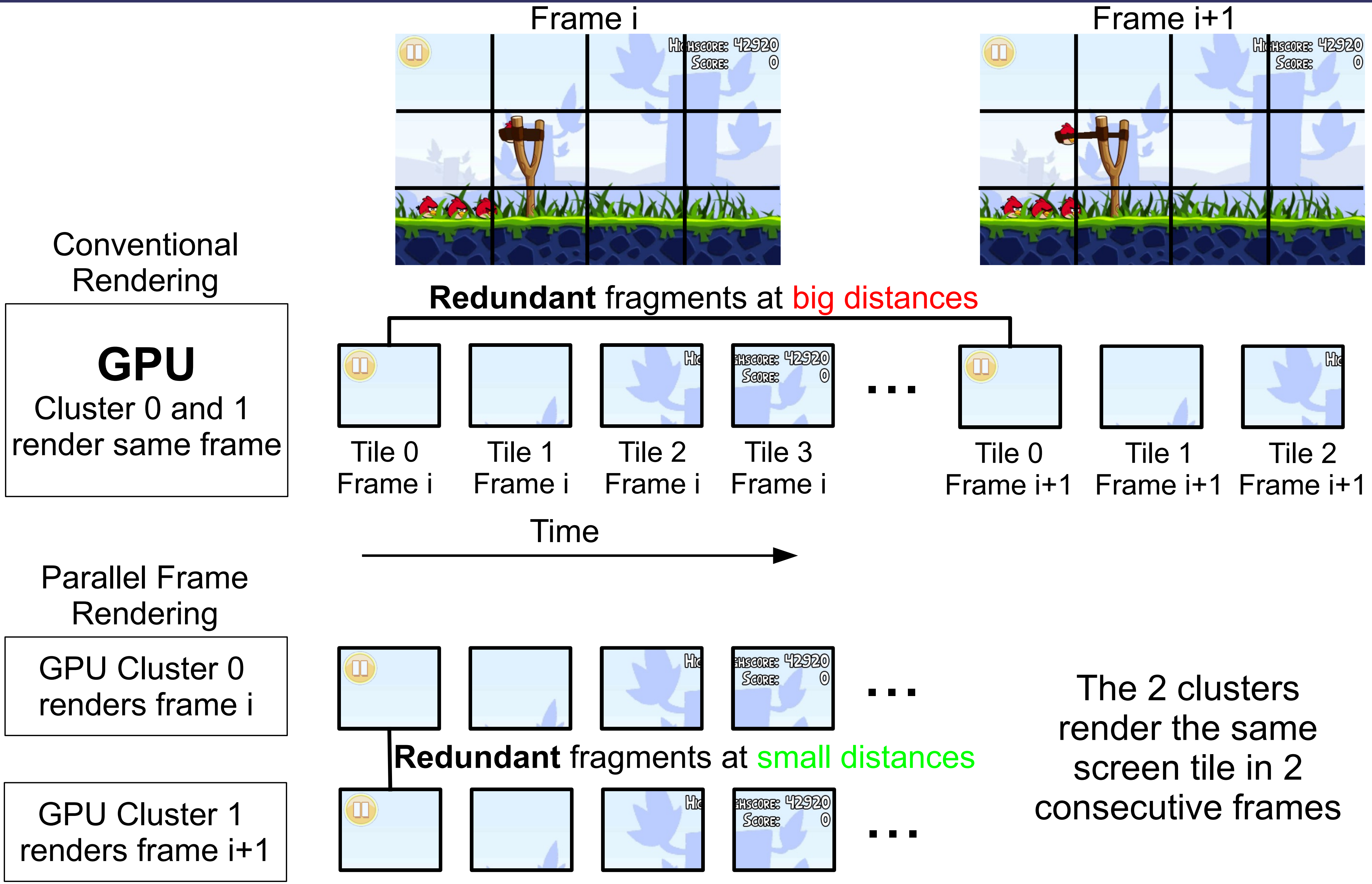
Time

...

...

The 2 clusters render the same screen tile in 2 consecutive frames

Reuse Distance



Frame i

Frame i+1

Conventional Rendering

GPU

Cluster 0 and 1 render same frame

Redundant fragments at big distances

Tile 0 Frame i

Tile 1 Frame i

Tile 2 Frame i

Tile 3 Frame i

...

Tile 0 Frame i+1

Tile 1 Frame i+1

Tile 2 Frame i+1

Time

Parallel Frame Rendering

GPU Cluster 0 renders frame i

GPU Cluster 1 renders frame i+1

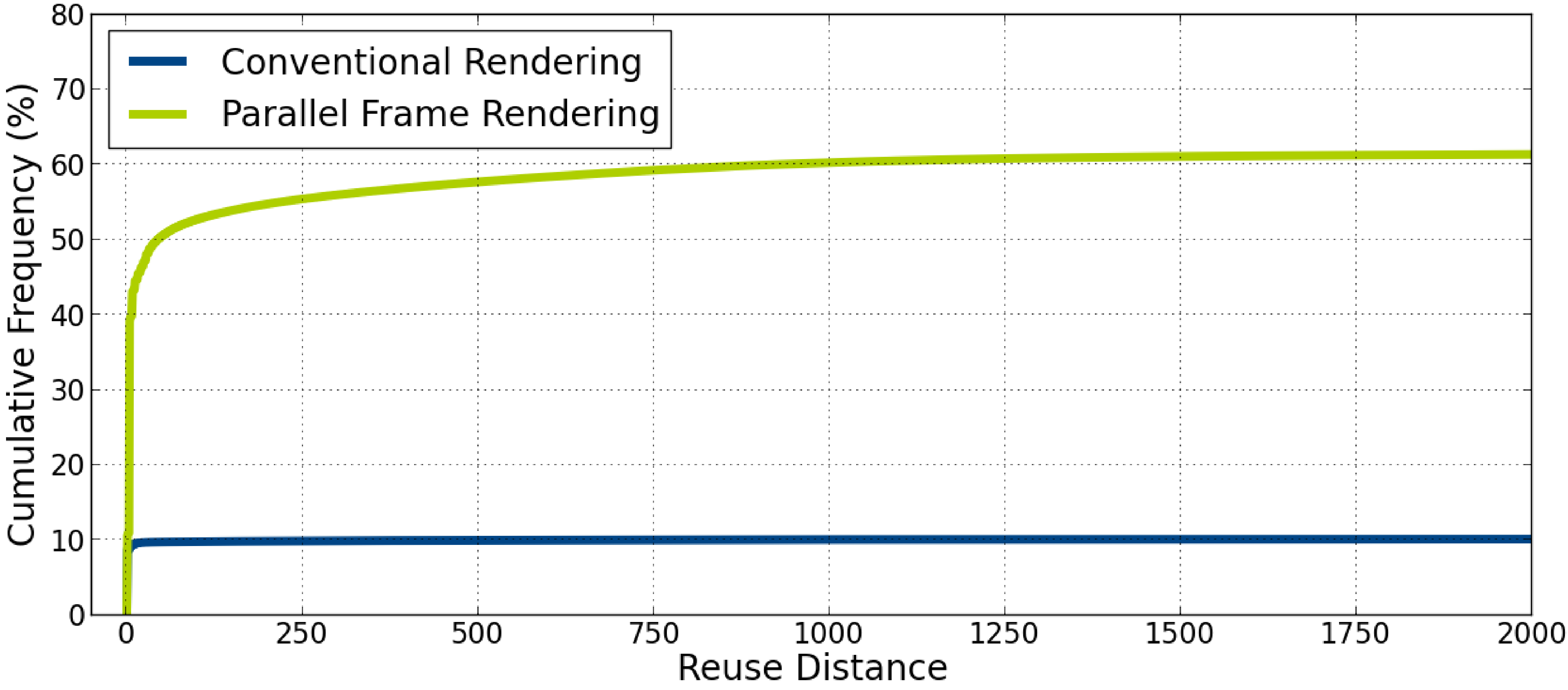
Redundant fragments at small distances

...

...

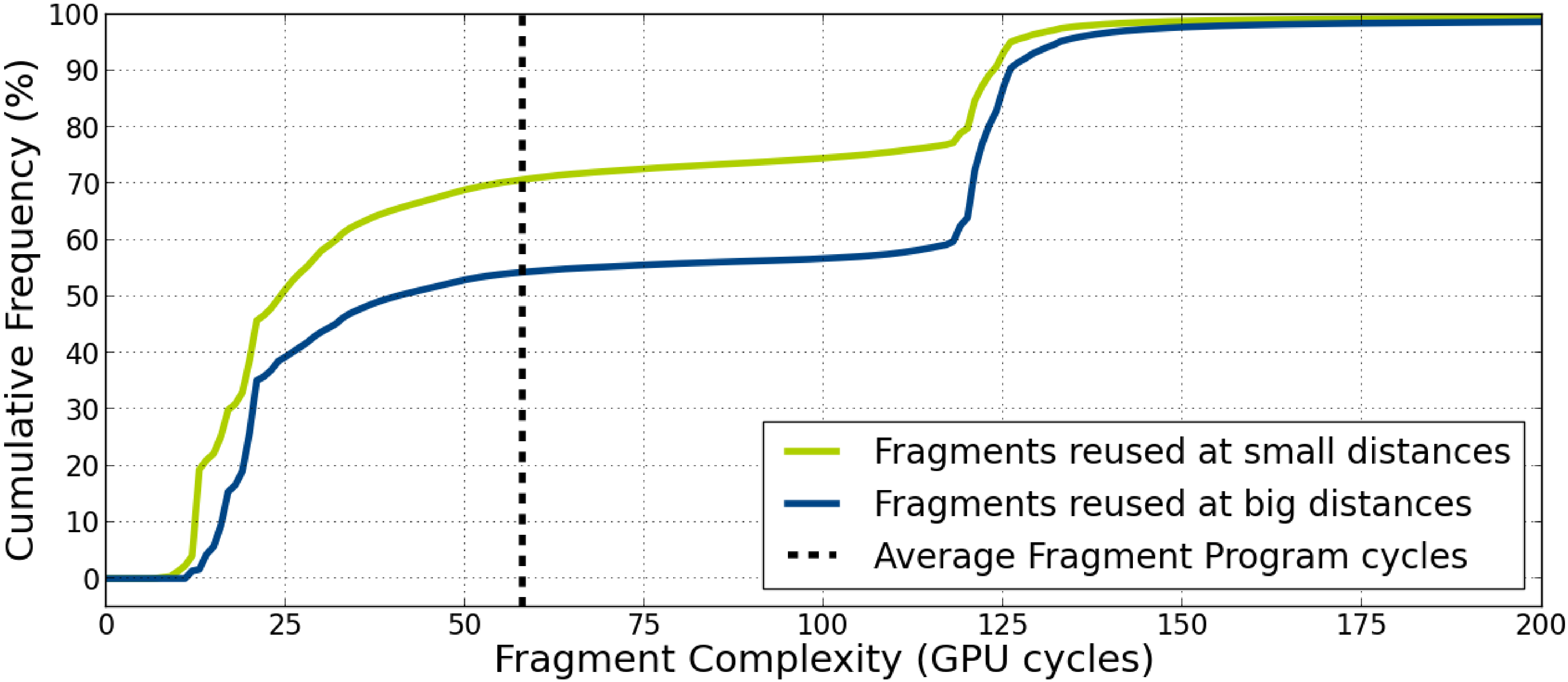
The 2 clusters render the same screen tile in 2 consecutive frames

Reuse Distance



61.3% of redundant fragments can be captured with realistic HW constraints when using Parallel Frame Rendering

Fragment Complexity



- Redundant fragments at small distances tend to be simpler
- All fragments take more than 6 cycles
  - → Enough to amortize the cost of accessing the hardware structures for memoization

Referential Transparency

- No **side-effects** in Fragment Program
  - It just computes the color of the fragment
- Updates to **global data**
  - **Texture** and **fragment program** updates are infrequent and easy to track
  - Typical loop in graphical applications:

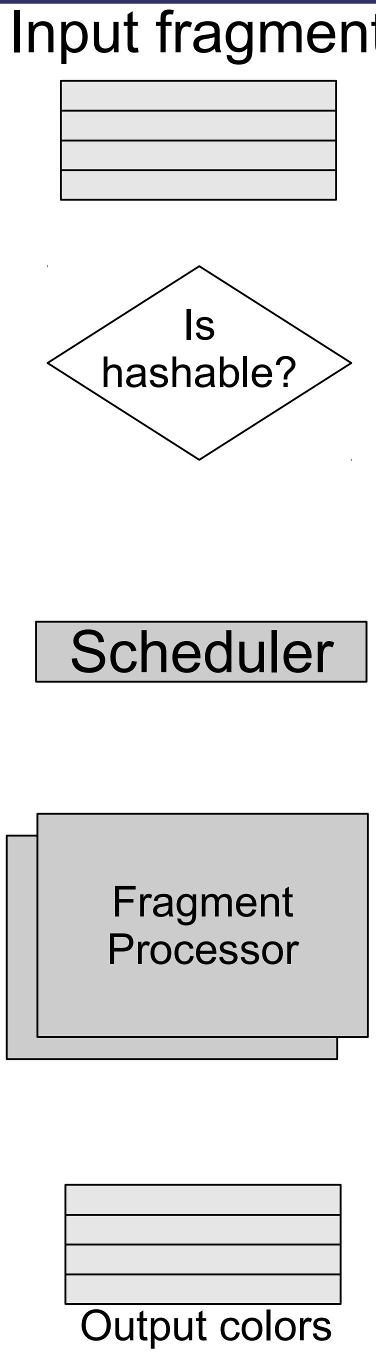
```
Initialize graphics data: textures, fragment programs...
while (true)
{
    Process inputs
    Animate scene
    Render
}
```

  - Discard memoized fragments when application updates global data used by the fragment program

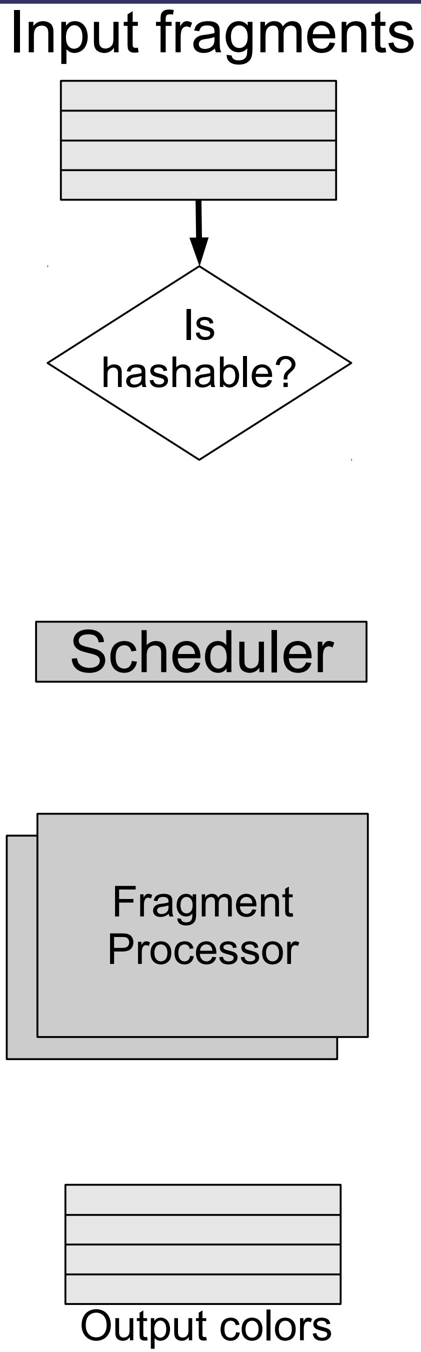
Outline

1. Motivation
2. Fragment Stage
3. Redundancy and Memoization
4. **Memoization in a Mobile GPU**
5. Experimental Results
6. Conclusions

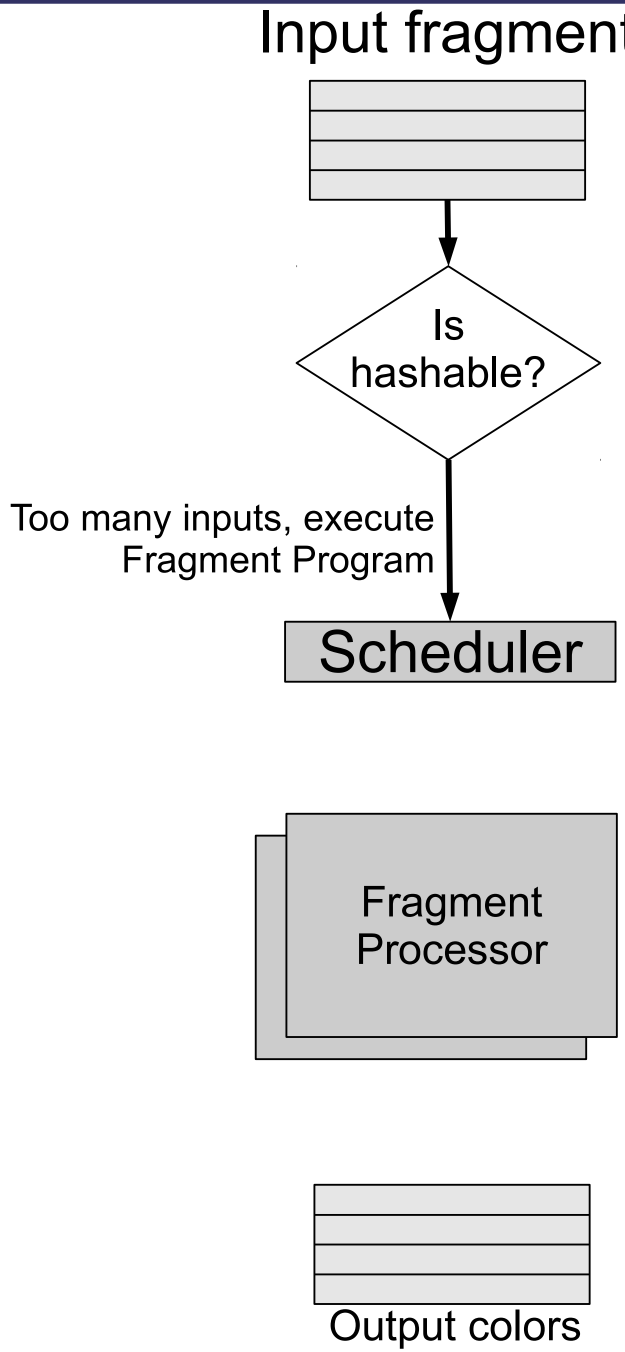
Task-Level Hardware Memoization



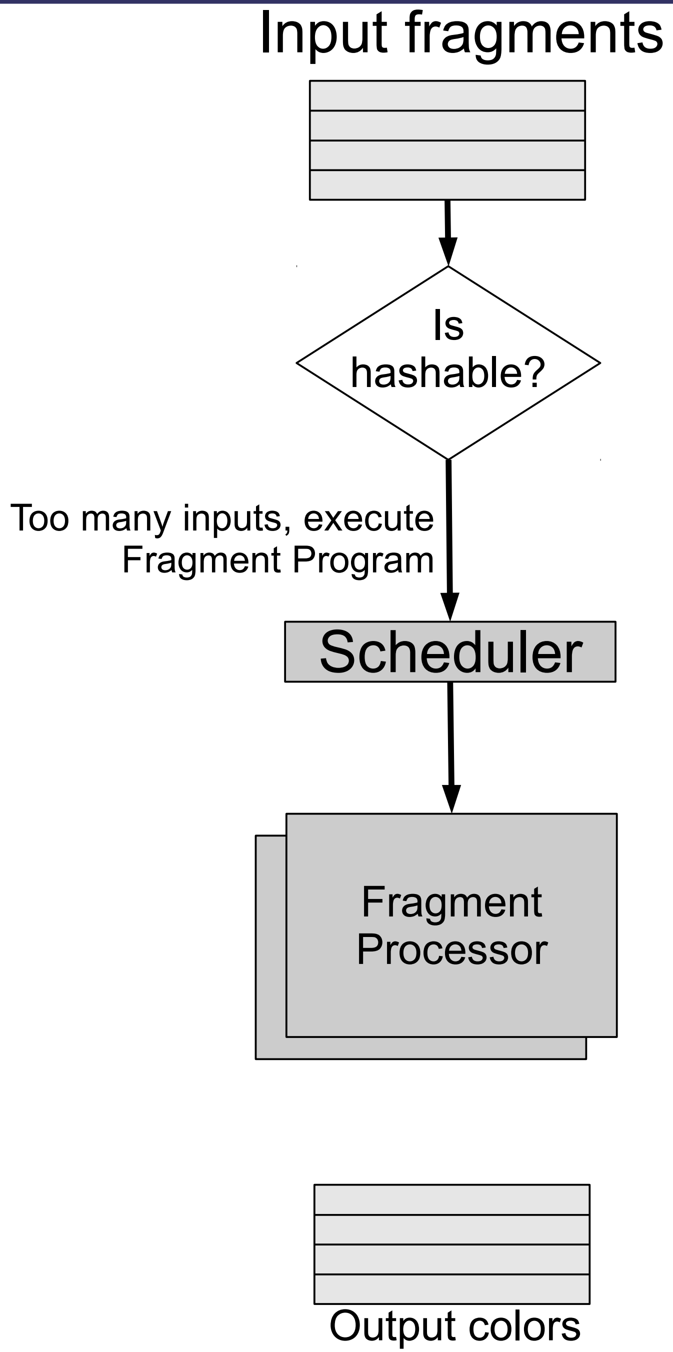
Task-Level Hardware Memoization



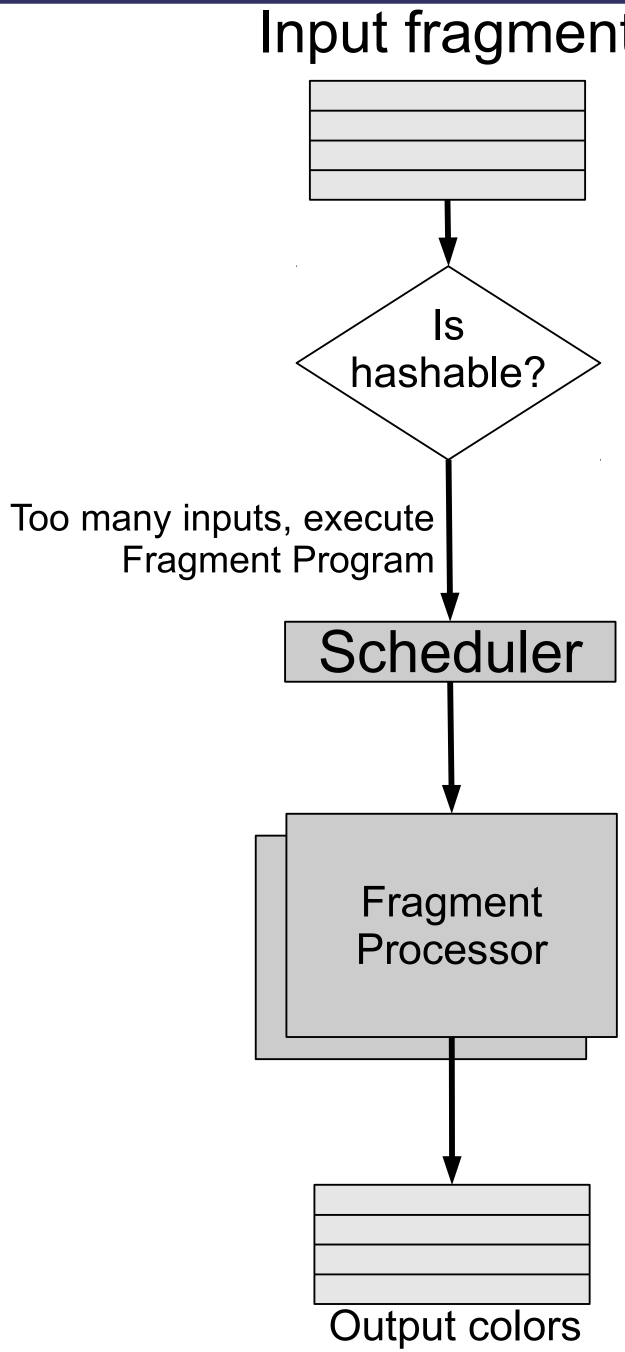
Task-Level Hardware Memoization



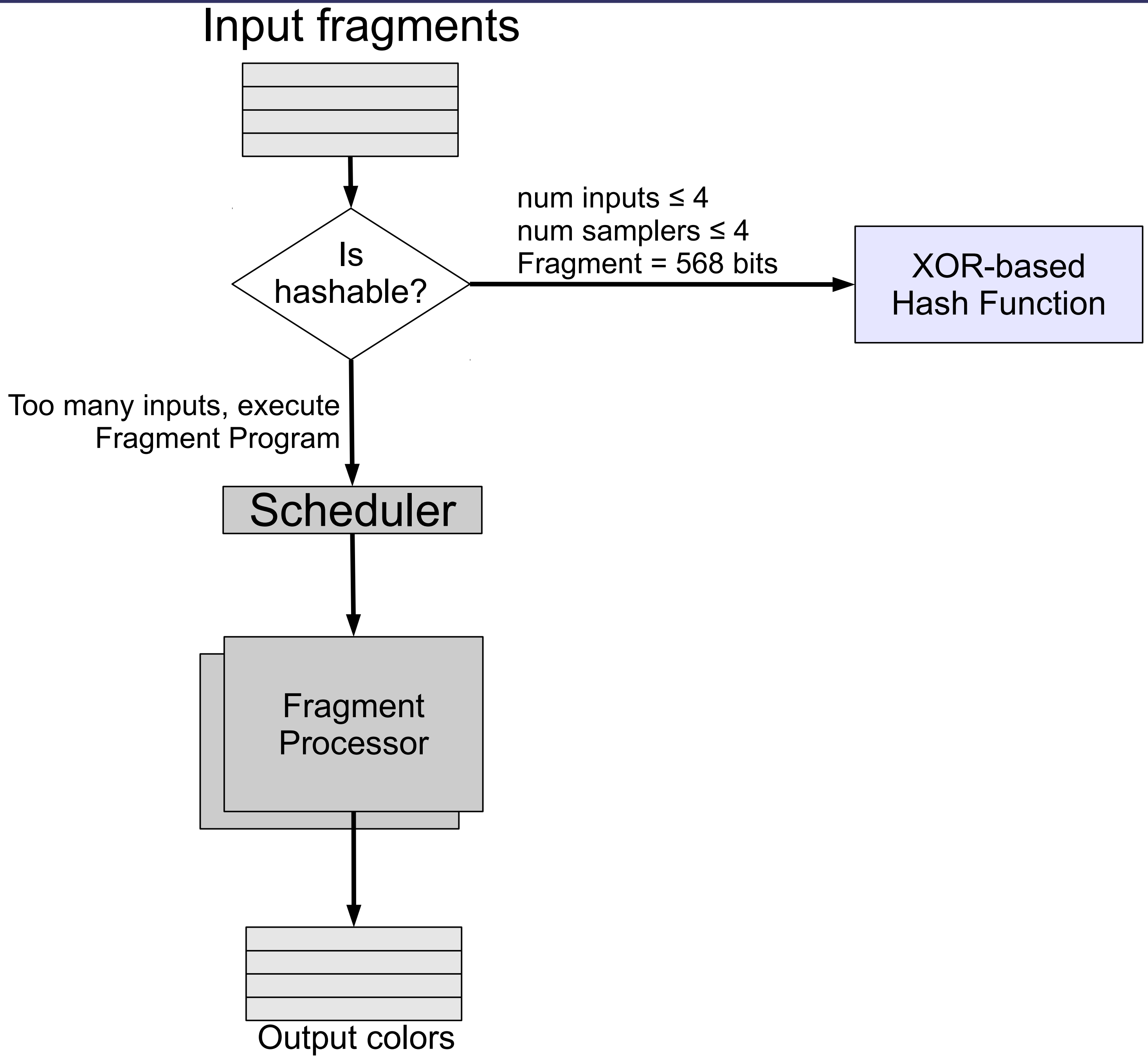
Task-Level Hardware Memoization



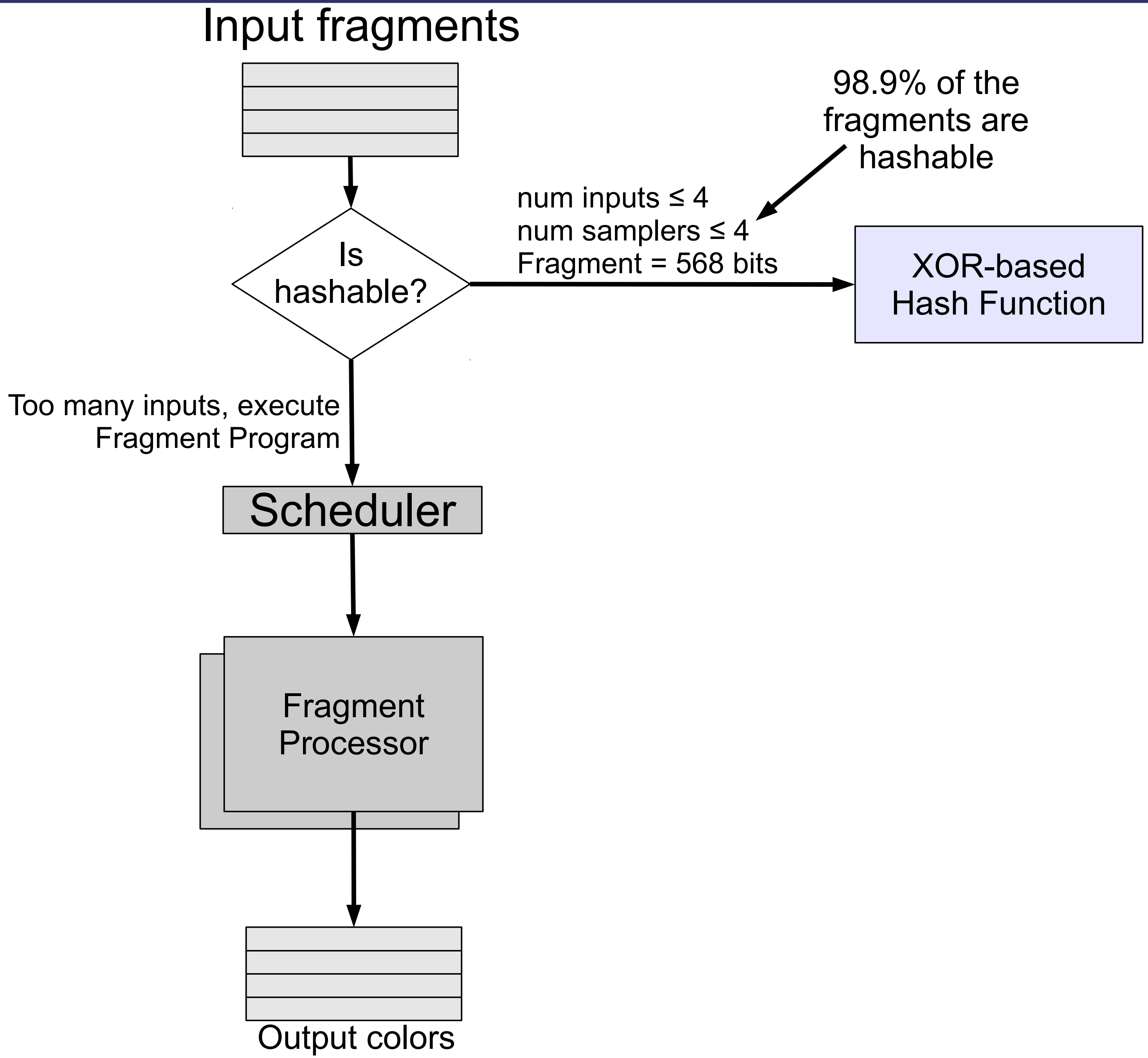
Task-Level Hardware Memoization



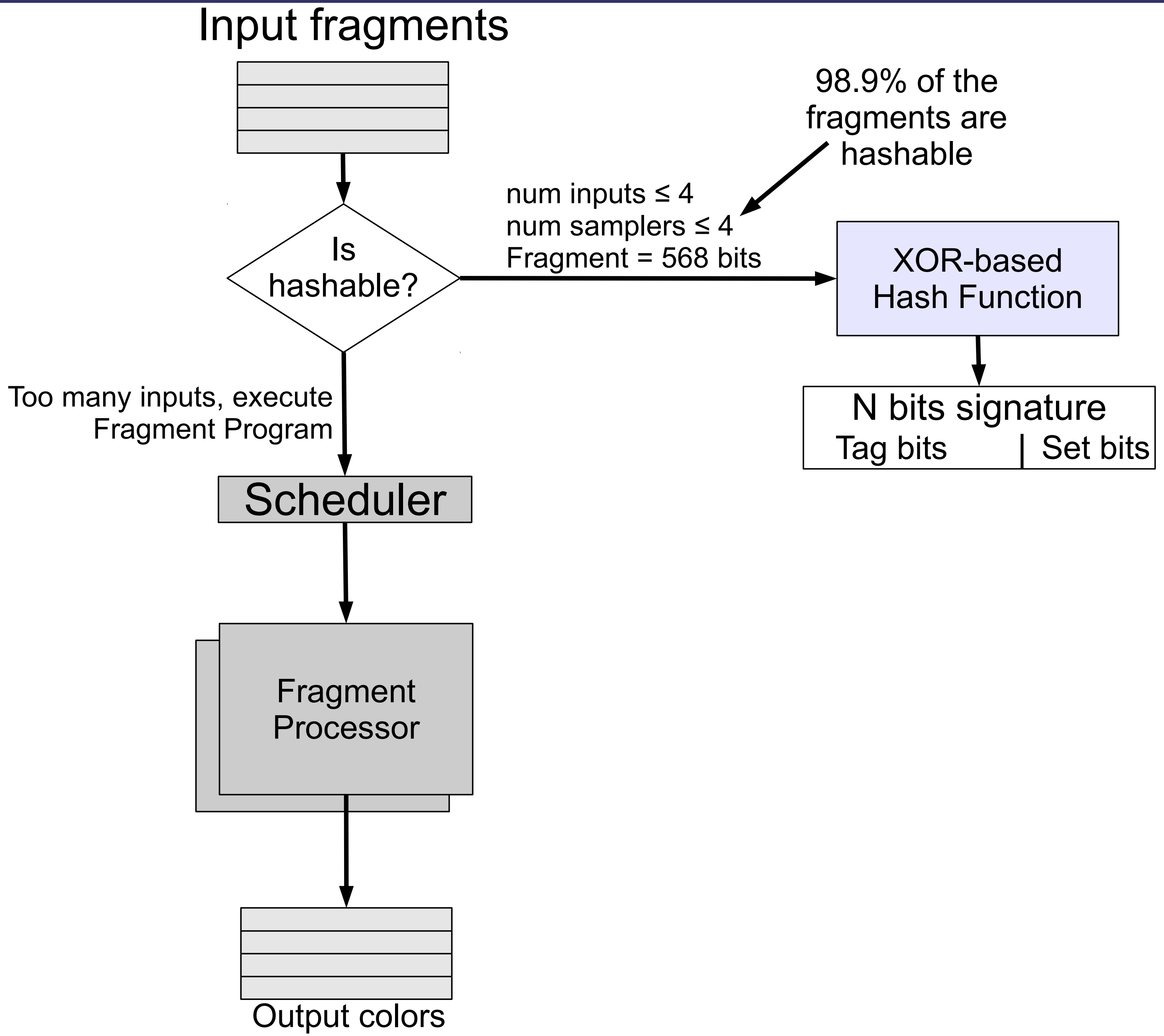
Task-Level Hardware Memoization



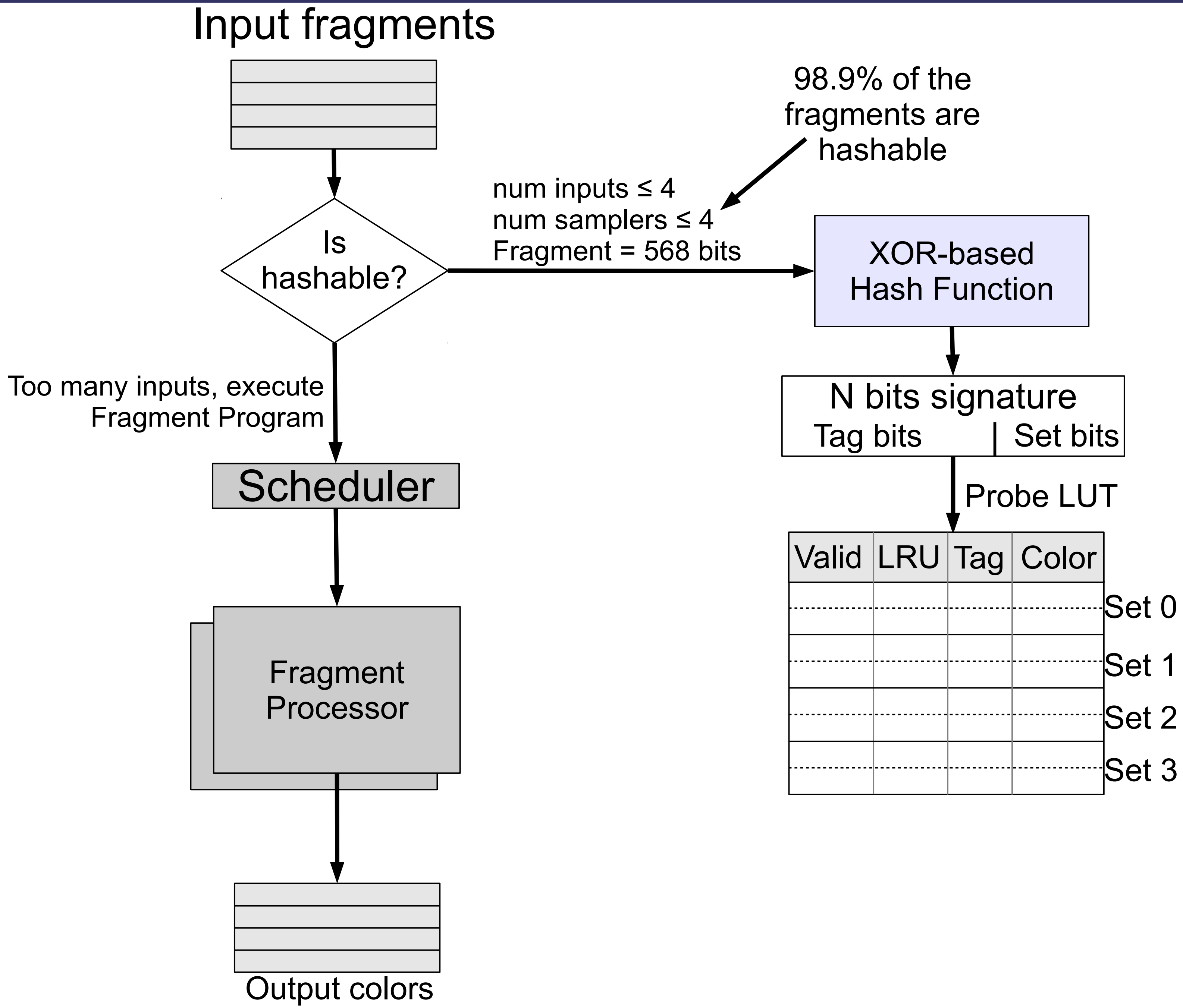
Task-Level Hardware Memoization



Task-Level Hardware Memoization



Task-Level Hardware Memoization



Input fragments

num inputs ≤ 4
num samplers ≤ 4
Fragment = 568 bits

98.9% of the fragments are hashable

N bits signature
Tag bits | Set bits

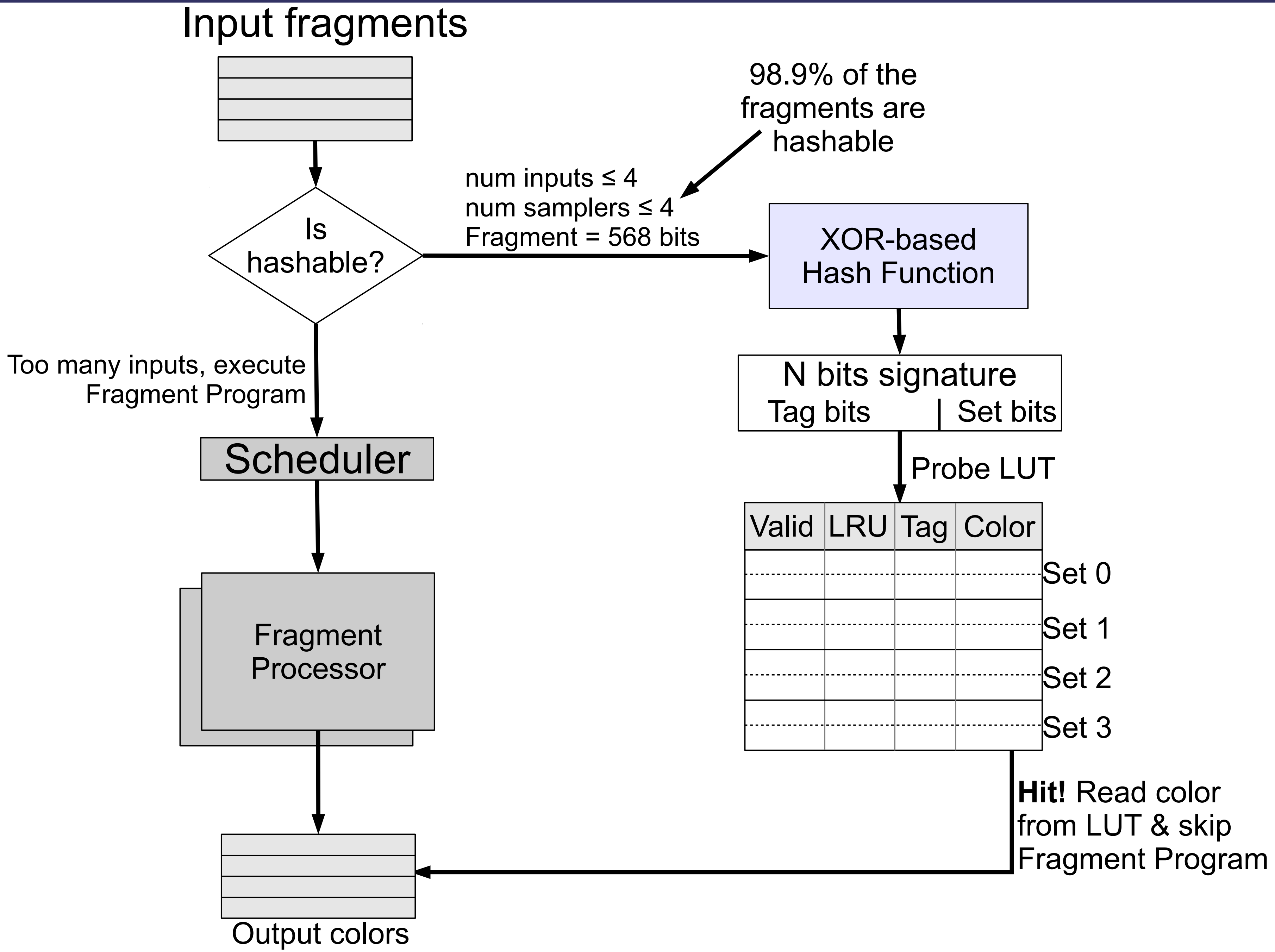
Probe LUT

Valid	LRU	Tag	Color	
				Set 0
				Set 1
				Set 2
				Set 3

Too many inputs, execute Fragment Program

Output colors

Task-Level Hardware Memoization



Input fragments

Is hashable?

num inputs ≤ 4
num samplers ≤ 4
Fragment = 568 bits

98.9% of the fragments are hashable

XOR-based Hash Function

N bits signature
Tag bits | Set bits

Probe LUT

Valid	LRU	Tag	Color	
				Set 0
				Set 1
				Set 2
				Set 3

Hit! Read color from LUT & skip Fragment Program

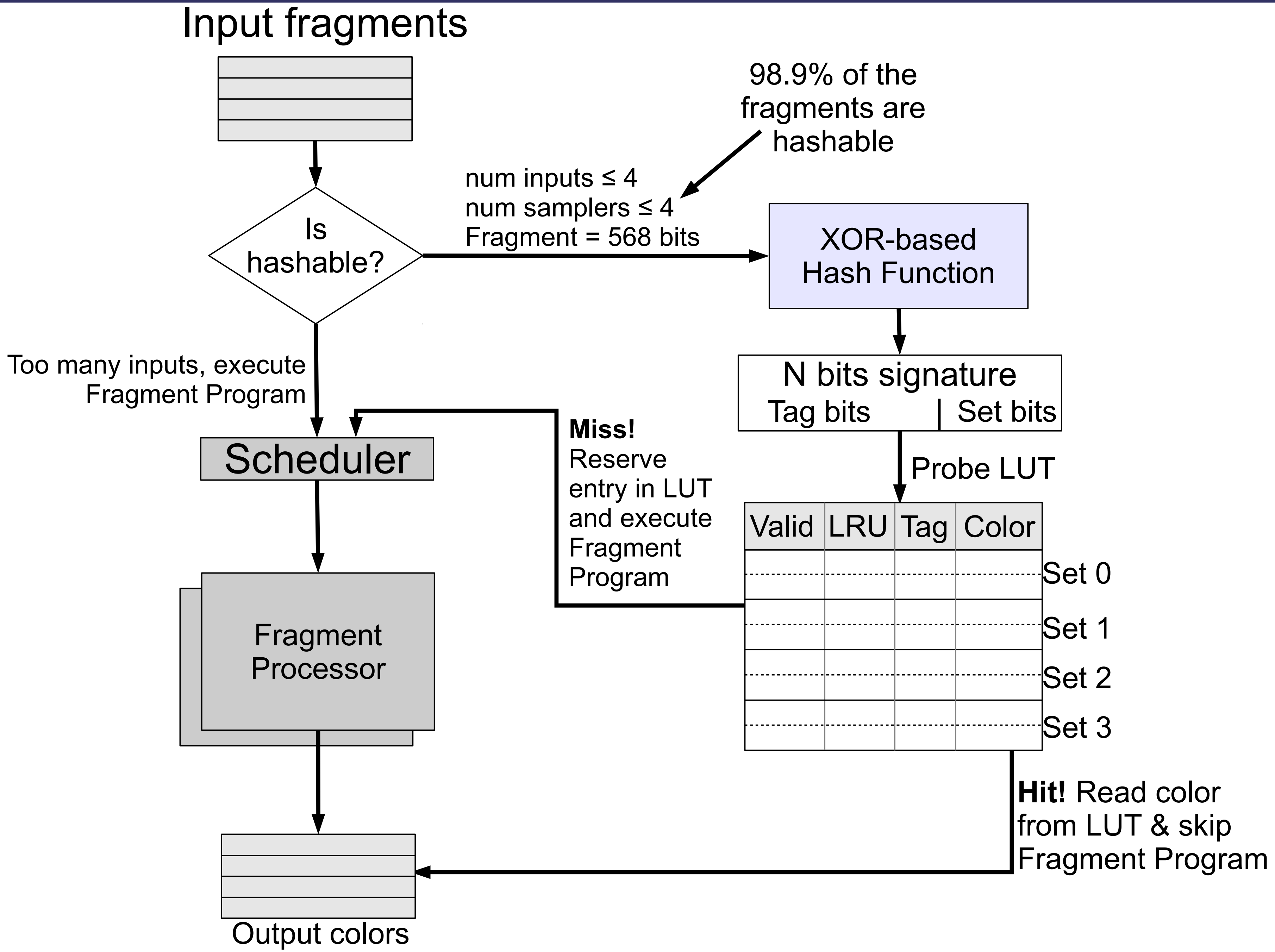
Too many inputs, execute Fragment Program

Scheduler

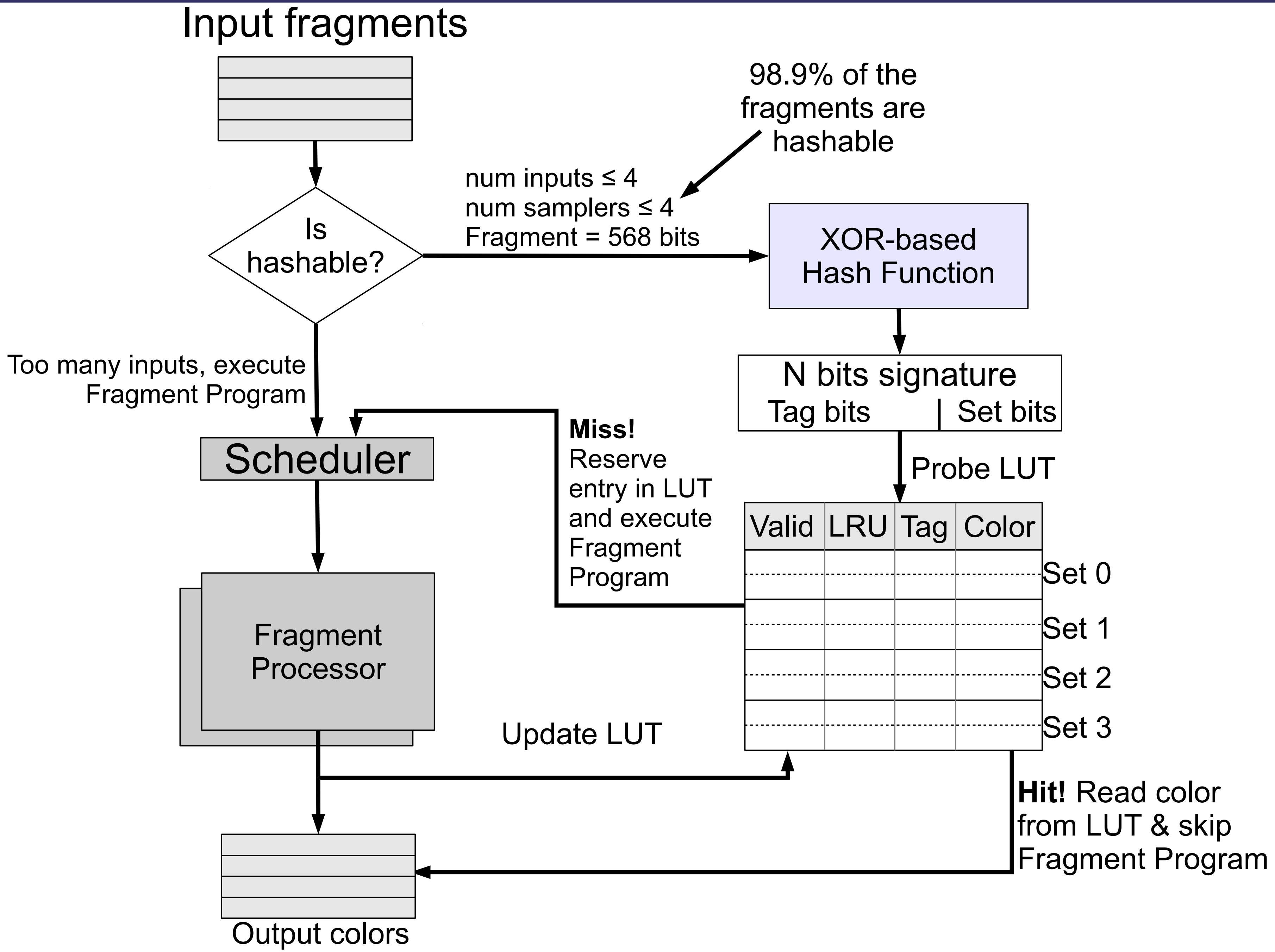
Fragment Processor

Output colors

Task-Level Hardware Memoization



Task-Level Hardware Memoization



Input fragments

Is hashable?

num inputs ≤ 4
num samplers ≤ 4
Fragment = 568 bits

98.9% of the fragments are hashable

XOR-based Hash Function

N bits signature
Tag bits | Set bits

Probe LUT

Valid	LRU	Tag	Color	
				Set 0
				Set 1
				Set 2
				Set 3

Too many inputs, execute Fragment Program

Scheduler

Miss! Reserve entry in LUT and execute Fragment Program

Fragment Processor

Update LUT

Hit! Read color from LUT & skip Fragment Program

Output colors

Outline

1. Motivation
2. Fragment Stage
3. Redundancy and Memoization
4. Memoization in a Mobile GPU
5. **Experimental Results**
6. Conclusions

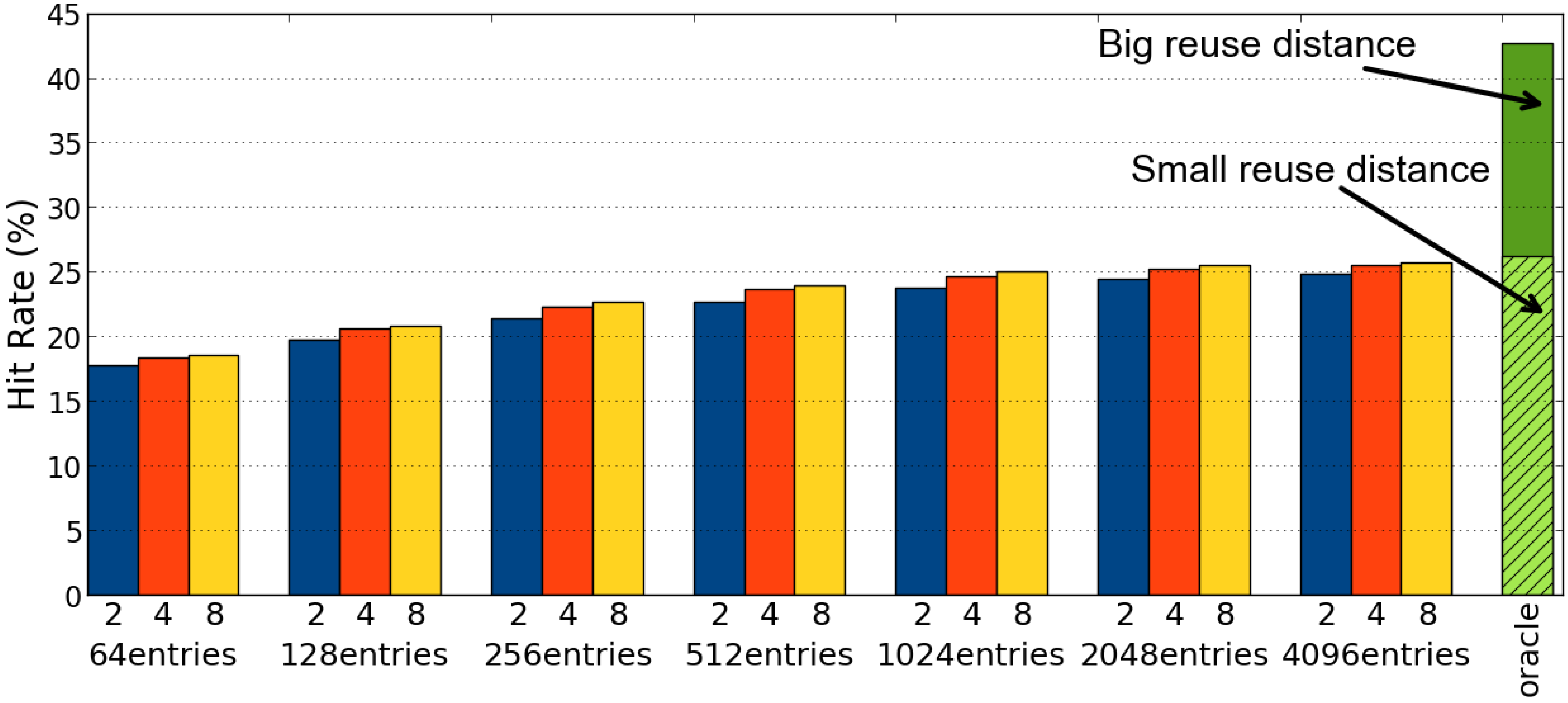
Evaluation Methodology

- TEAPOT simulation infrastructure
  - Android and OpenGL ES
  - GPU timing simulator models:
    - Tile-Based Rendering architecture (ARM Mali 400MP-like)
    - Parallel Frame Rendering (2 frames in parallel)
  - GPU power model based on **McPAT**
- Workloads
  - 9 Android commercial games, 400 frames each

Technology	32nm
Tile size	16x16 pixels
Number of clusters	2 **(PFR)**
Fragment processors per cluster	2
Vertex processors per cluster	2

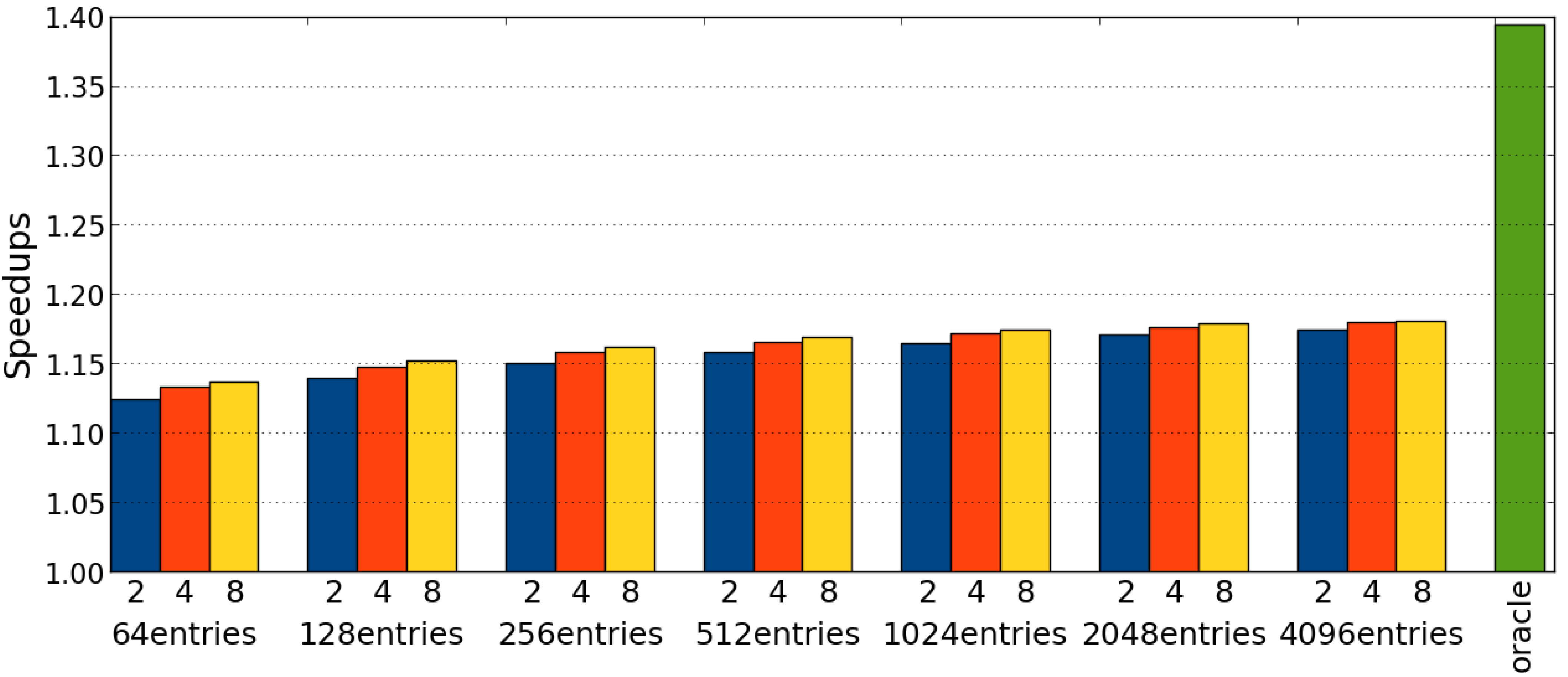
L2 cache	128 KB, 8-way
Main memory	1 GB, 16 bytes/cyle
Look Up Table num sets	8 → 2048 : 2*
Look Up Table num ways	2, 4, 8
Signature Size	32 bits

Hit Rate in the Look Up Table



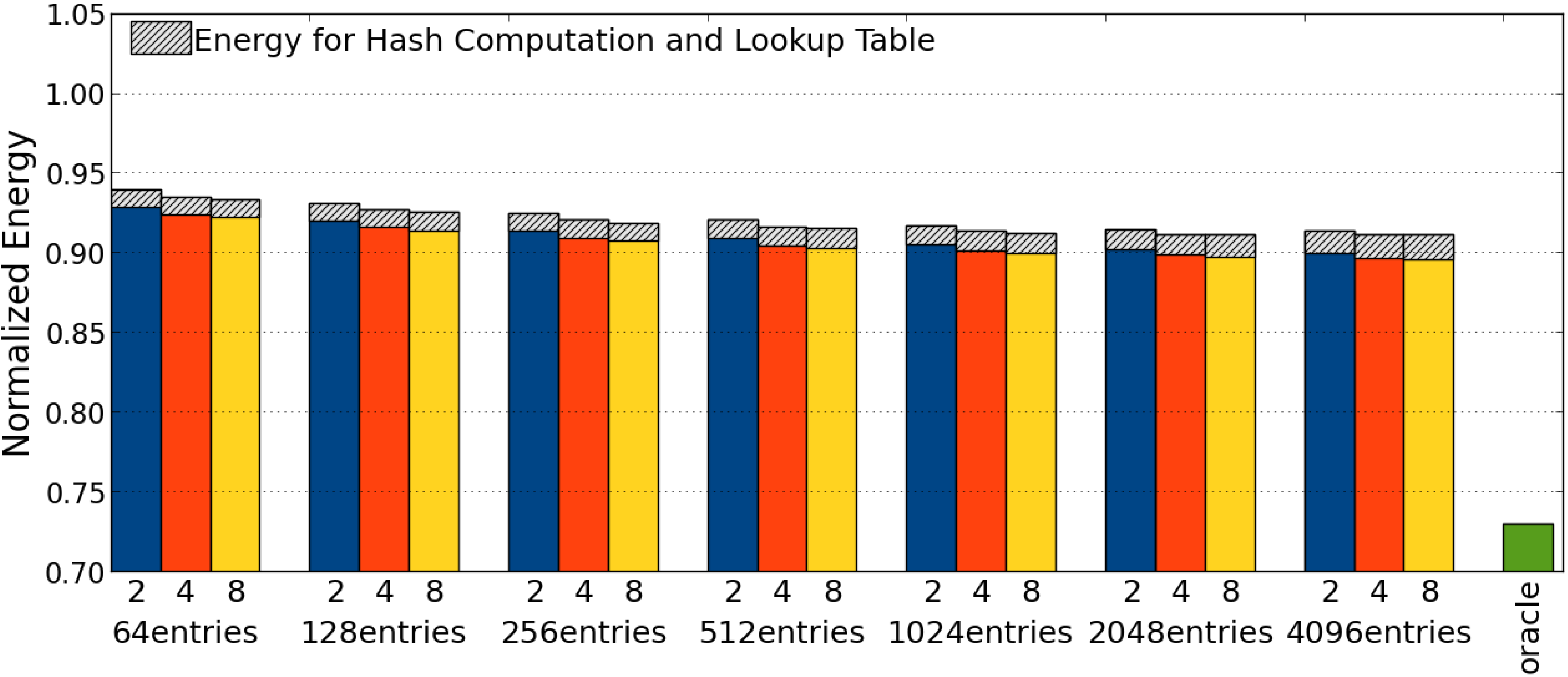
- 42.7% of fragments are redundant on average
- 26.1% of fragments are redundant at small distances (<2048)
- Hardware LUT with 2048 entries and 4-way achieves 25.2% hit rate
- A small LUT captures 96.5% of the redundancy at small distances

Speedup



Hardware LUT with 2048 entries and 4-way achieves **17.6% speedup** on average for a set of commercial Android games

Normalized Energy



- Hardware LUT with 2048 entries and 4-way achieves **9% energy savings** on average for a set of commercial Android games
- LUT energy represents just 1.5% of overall GPU energy consumption on average (2048 entries 4-way configuration)

Outline

1. Motivation
2. Fragment Stage
3. Redundancy and Memoization
4. Memoization in a Mobile GPU
5. Experimental Results
6. **Conclusions**

Conclusions

- Graphical applications exhibit a high degree of redundancy
  - 42.7% of the fragments are redundant on average
- Hardware memoization is no simple task, as most of the redundancy is inter-frame
- Parallel Frame Rendering brings 61.3% of the redundant fragments at distances amenable for hardware memoization
- A simple hardware LUT captures most of the redundancy at small distances, providing 17.6% speedup and 9% energy savings on average

41st International Symposium on Computer Architecture

Eliminating Redundant Fragment Shader Executions on a Mobile GPU via Hardware Memoization

Jose-Maria Arnau
jarnau@ac.upc.edu

Joan-Manuel Parcerisa
jmanel@ac.upc.edu

Universitat Politecnica de Catalunya

Polychronis Xekalakis
polychronis.xekalakis@intel.com

Intel Corporation