A Novel Texture Cache Architecture for Fully Pipelined Graphics Engine

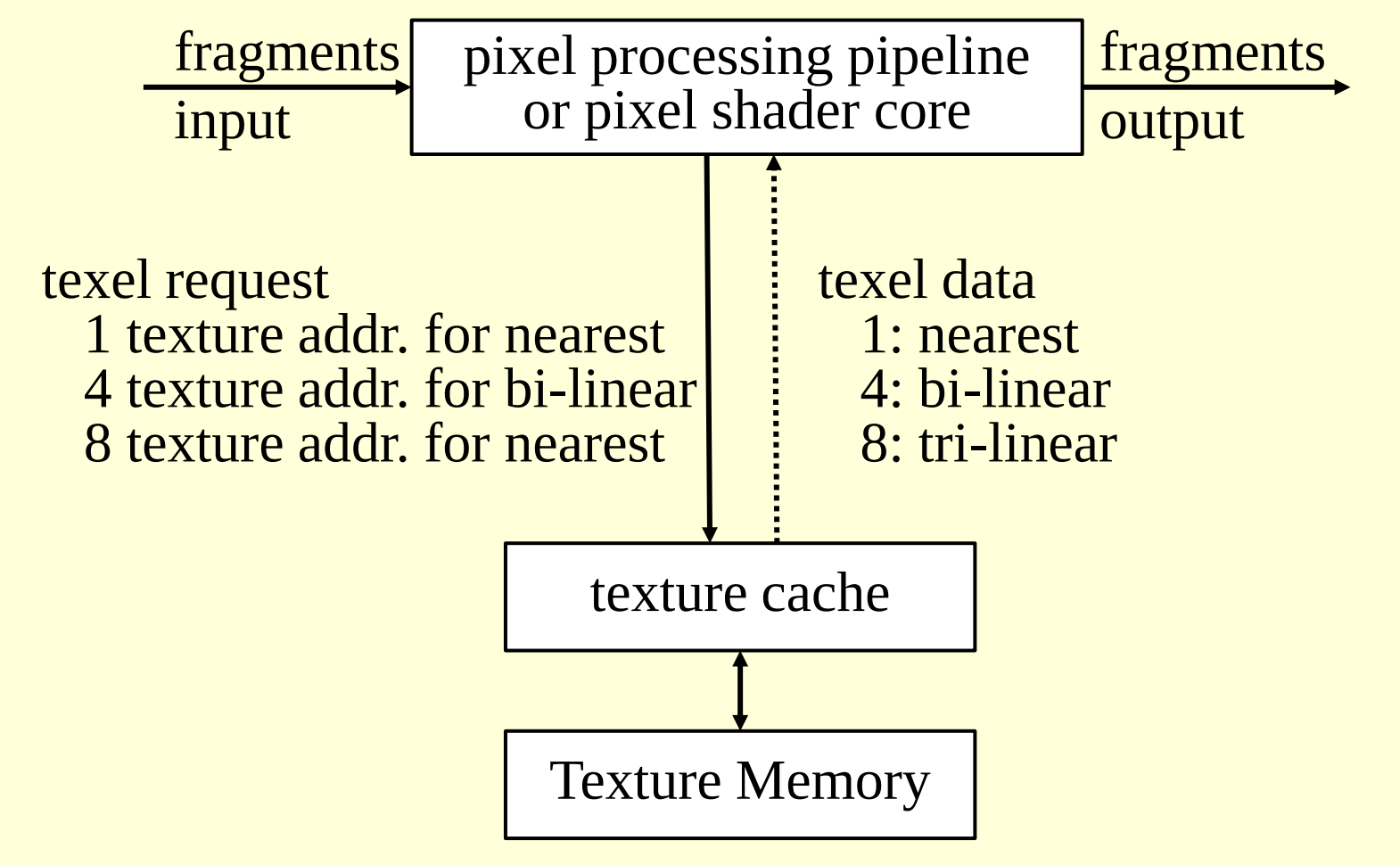
Seokyoon Jung, Kwon-taek Kwon, Sang-oak Woo, Sungjin Son
System Architecture Lab, Samsung AIT, Samsung Electronics Co., Ltd.

HIGH-PERFORMANCE GRAPHICS
PARIS, FRANCE JUNE 25-27, 2012

SAMSUNG ADVANCED INSTITUTE OF TECHNOLOGY

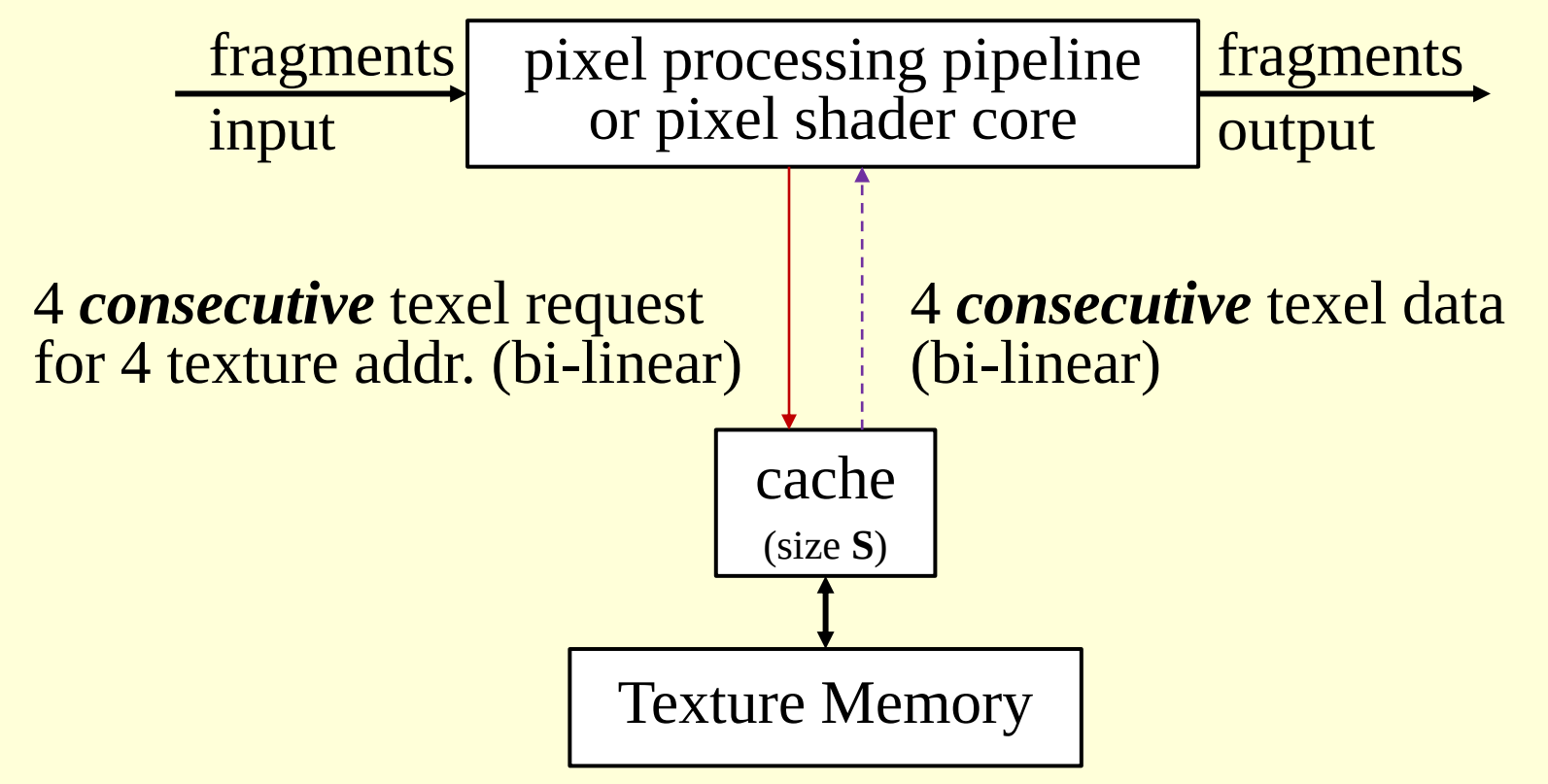
Motivation

- Worldwide sales of mobile devices totaled 440.5M units in the 3rd quarter of 2011, according to Gartner, Inc., Nov. 15, 2011.
- Texture Mapping – the number of fragments to be textured can be large, and each textured fragments requires multiple texture lookup (usually 1~8) [4].
- Texture Cache – the essential component of graphics rendering system, to provide very high texture memory bandwidth and **low-latency access for any texture filtering mode** ! [4]

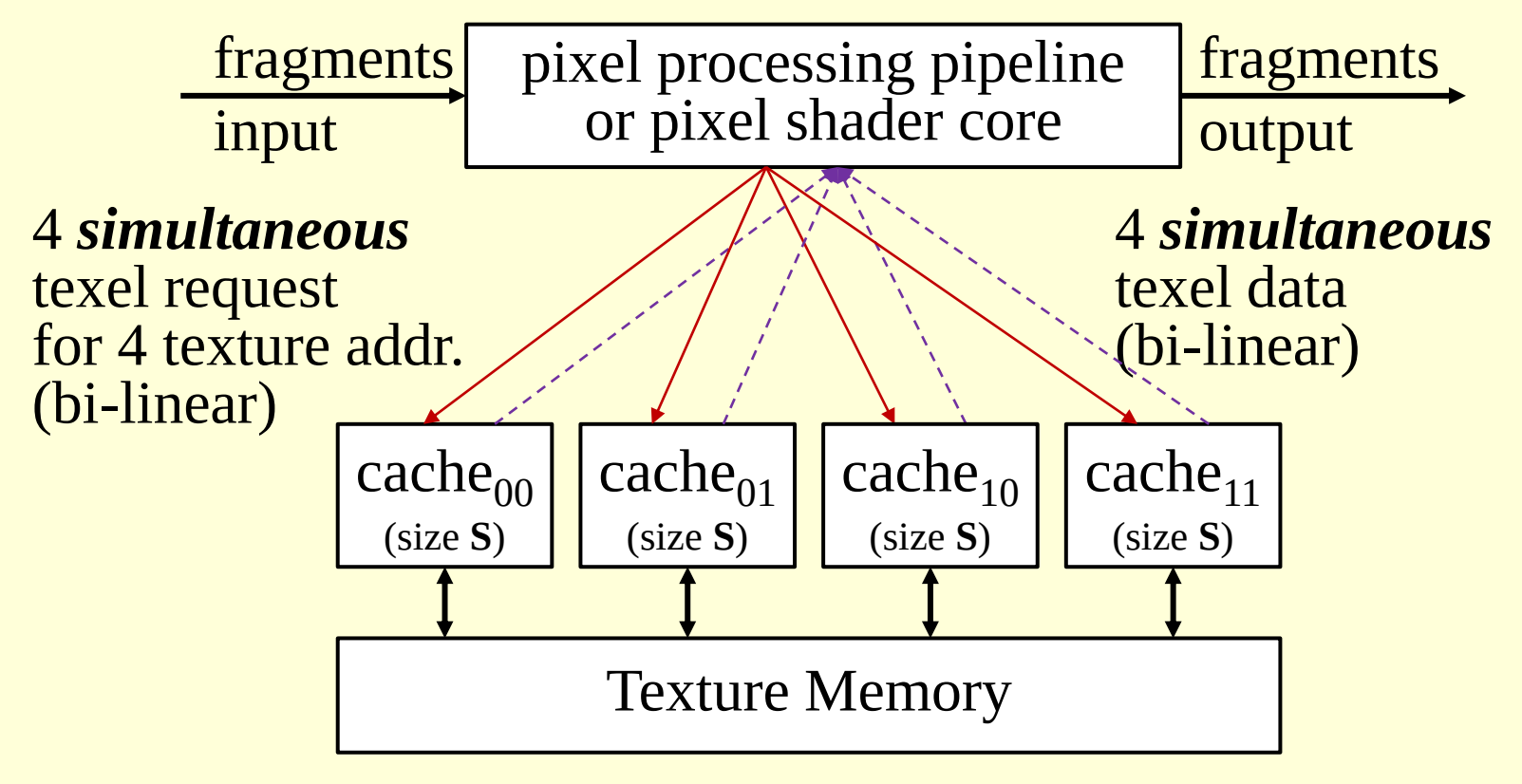

fragments input
pixel processing pipeline or pixel shader core
fragments output
texel request
1 texture addr. for nearest
4 texture addr. for bi-linear
8 texture addr. for nearest
texel data
1: nearest
4: bi-linear
8: tri-linear
texture cache
Texture Memory

Texture Cache

- Memory bandwidth reduction – one major objective of texture cache, depends on the architecture of cache module, *i.e.* cache associativity, cache block size, total cache memory size, etc [2].
- For bi-linear filtering mode, for example, four texel data has to be fetched from the cache for a single fragment texture mapping (or pixel shading)
- Brute force implementation – A cache, consist of conventional cache architecture, can provide one single data for each request → **need to wait for three cycles** to get all four texel for bi-linear
- The lowest-latency but the most expensive – Four identical independent caches can provide four texel data in simultaneously → **very inefficient** scheme in terms of cost *vs.* performance, because all four cache_{ij} has to cover the same memory space


fragments input
pixel processing pipeline or pixel shader core
fragments output
4 consecutive texel request for 4 texture addr. (bi-linear)
4 consecutive texel data (bi-linear)
cache (size S)
Texture Memory

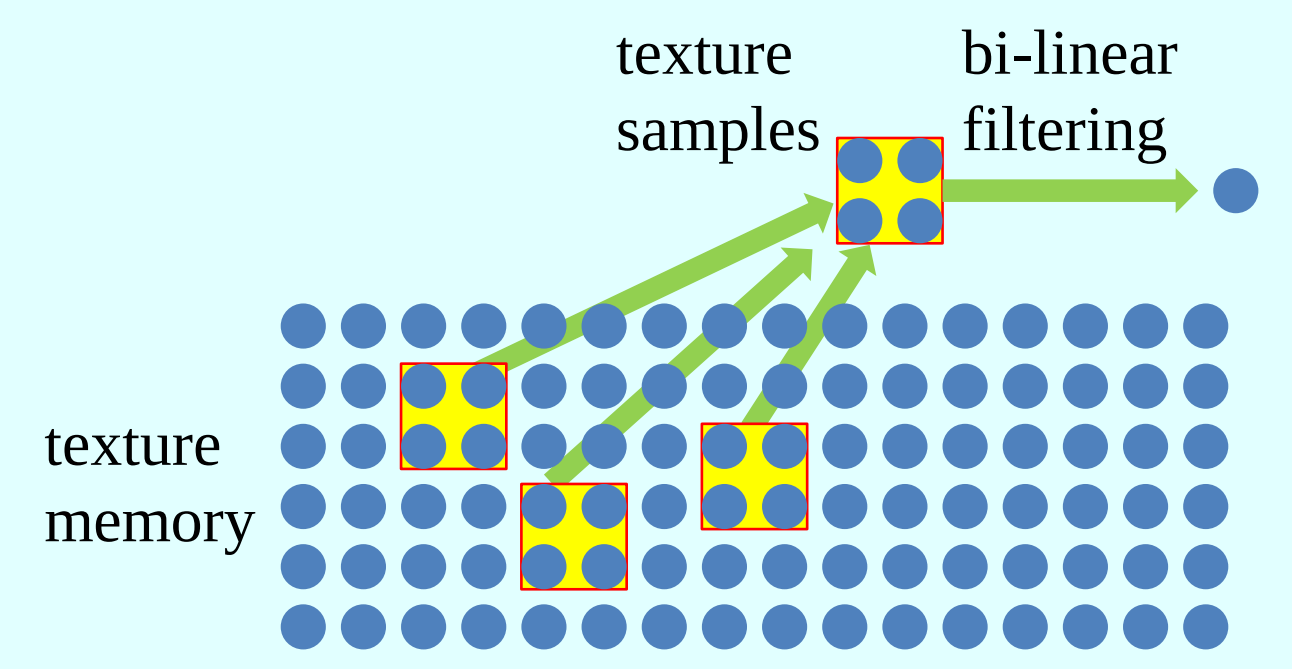
Brute force implementation


fragments input
pixel processing pipeline or pixel shader core
fragments output
4 simultaneous texel request for 4 texture addr. (bi-linear)
4 simultaneous texel data (bi-linear)
cache00 (size S)
cache01 (size S)
cache10 (size S)
cache11 (size S)
Texture Memory

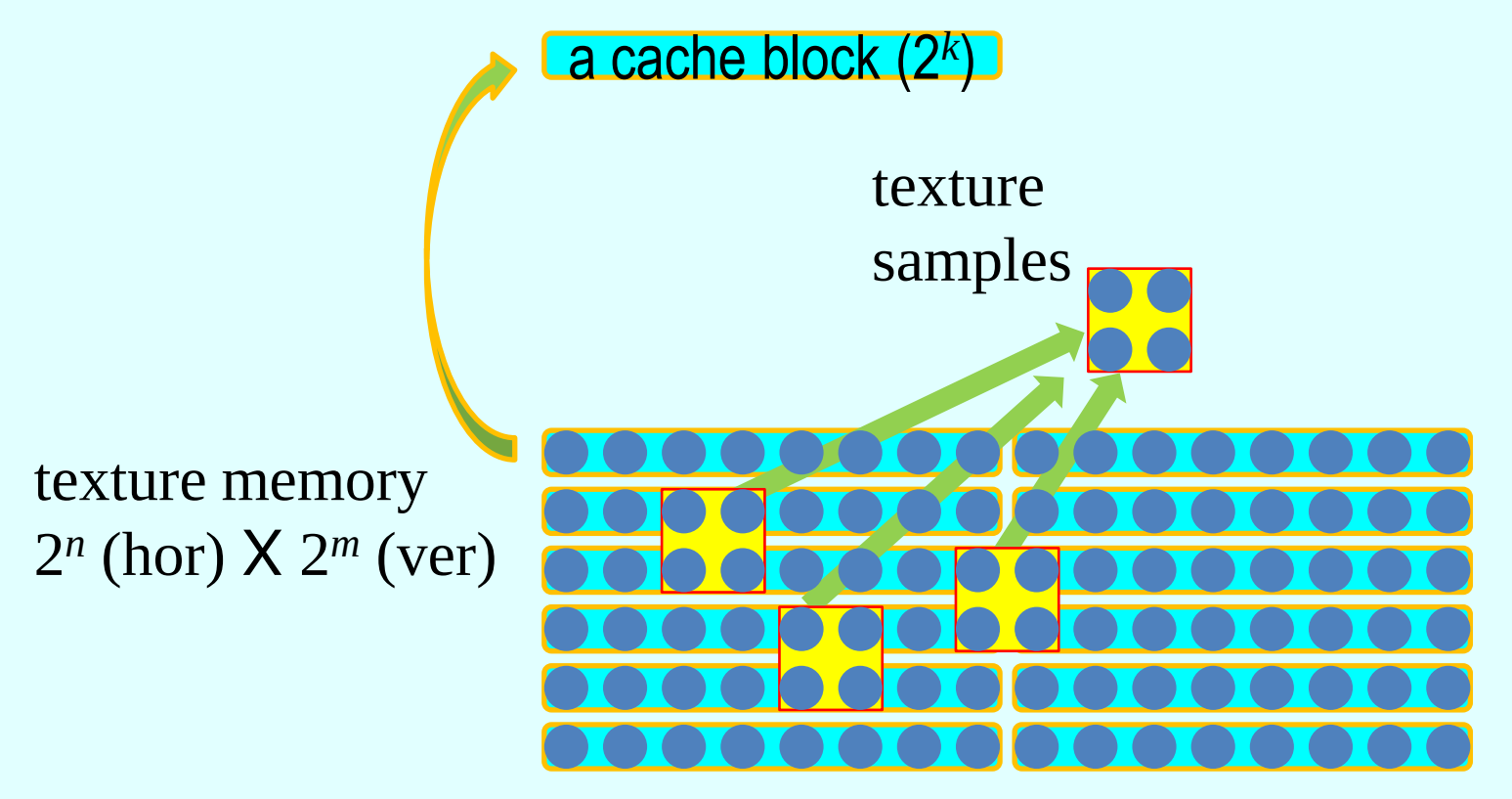
The lowest-latency but the most expensive !

Quad-split Cache

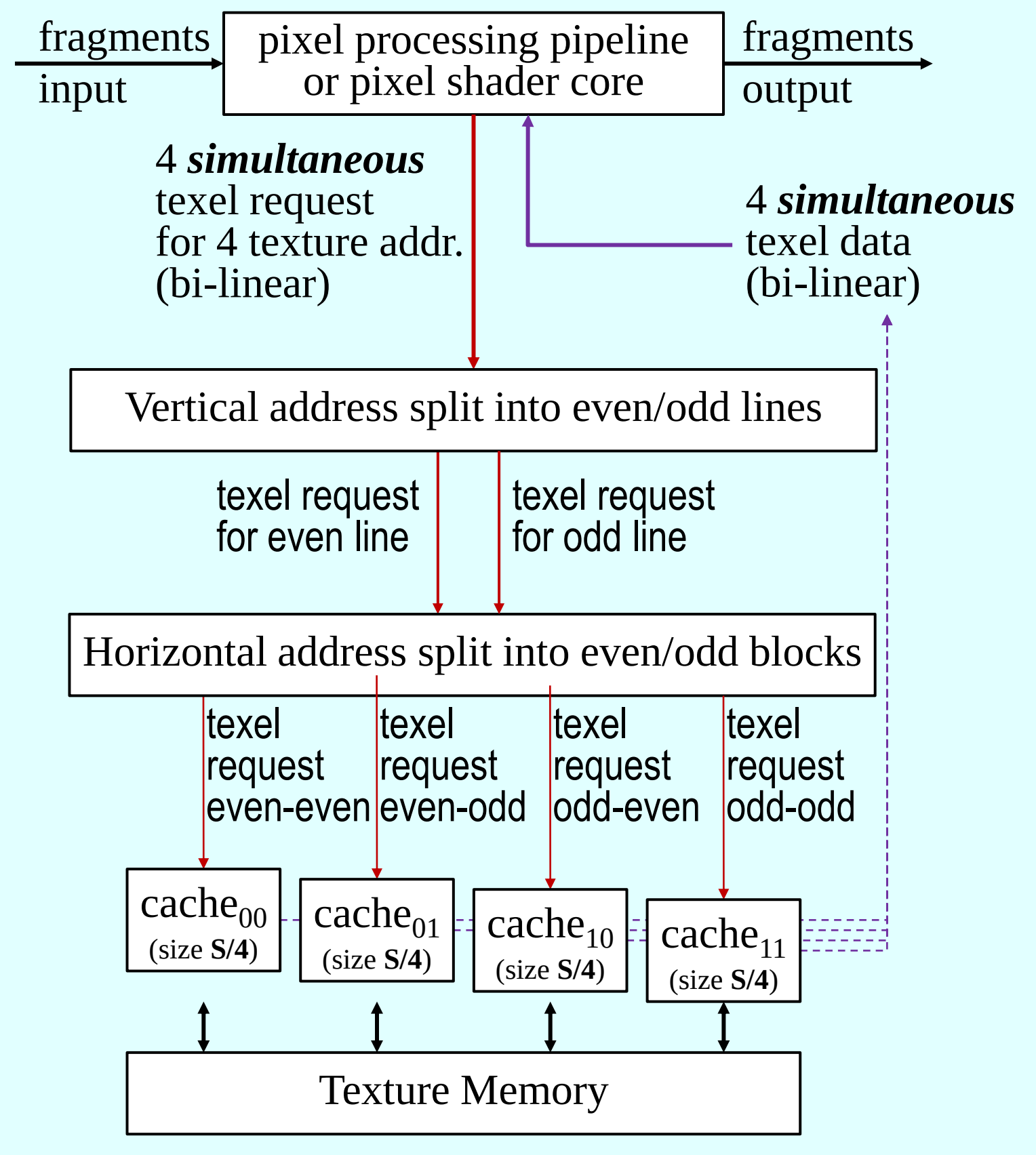
- A typical bi-linear texture mapping – samples four texels, **adjacent each other** in horizontal and vertical direction.
- Proposed cache scheme, assumes texture size is power of two (2^n X 2^m), and also the cache block size (2^k)
- We can assign numbers for each blocks of texture memory area, in accordance with texture cache block to be cached → any texture sample for bi-linear filtering, a pair of horizontal direction must be laid **both in the same block** or **one by one in the adjacent blocks** each other.
- Thus, we can split the texture memory space for the texture data into two sub-spaces. These two sub-spaces can be cached/fetched independently simultaneously → **even blocks and odd blocks.**
- Moreover, 2D texture can be easily **split into odd and even lines** in vertical direction. → It is easy to combine both horizontal and vertical split schemes, we can get **quad-split cache** [1].
- We developed quad-split cache scheme for any integer combinations of (n,m,k), which can support not only bi-linear but also nearest.
- Quad-split scheme can support texture wrapping modes – clamp, repeat and mirrored repeat.
- The limitations – texture size (both horizontal and vertical) and cache block size must be two to the powers of any integer, due to binary split scheme.
  And, the base address for any texture data must be aligned with the first even cache block.



Typical bi-linear texture mapping



Proposed texture sampling with proposed texture cache architecture


fragments input
pixel processing pipeline or pixel shader core
fragments output
4 simultaneous texel request for 4 texture addr. (bi-linear)
4 simultaneous texel data (bi-linear)
Vertical address split into even/odd lines
texel request for even line
texel request for odd line
Horizontal address split into even/odd blocks
texel request even-even
texel request even-odd
texel request odd-even
texel request odd-odd
cache00 (size S/4)
cache01 (size S/4)
cache10 (size S/4)
cache11 (size S/4)
Texture Memory

Proposed quad-split cache scheme

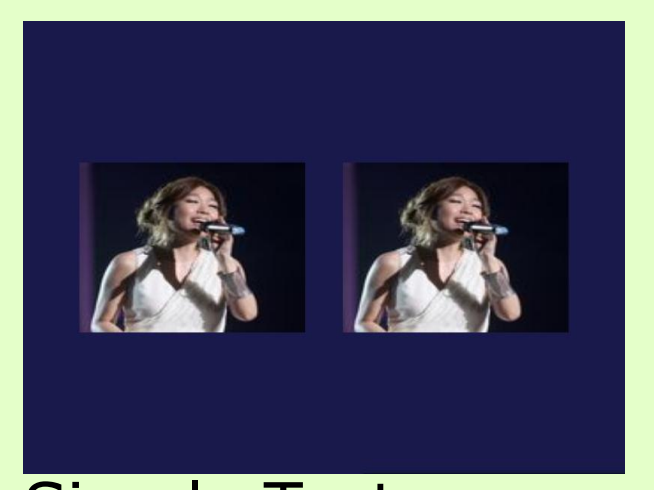
FPGA Proto-typing

- The proposed quad-split cache scheme is adopted into our own GPU system.
- Synopsys HAPS-64 base FPGA proto-typing, running at 25MHz with single port SDRAM modules
- The cache configuration – total 32KB, 8KB for each quad-split cache, direct mapped cache scheme for each quad-split cache, 4 texels per a cache block
- The rendering pipeline – Quad-core SAIT TBR engine [3]
- Some famous OpenGL ES benchmarks are tested in quantitative manner

HAPS64 base FPGA proto-typing board

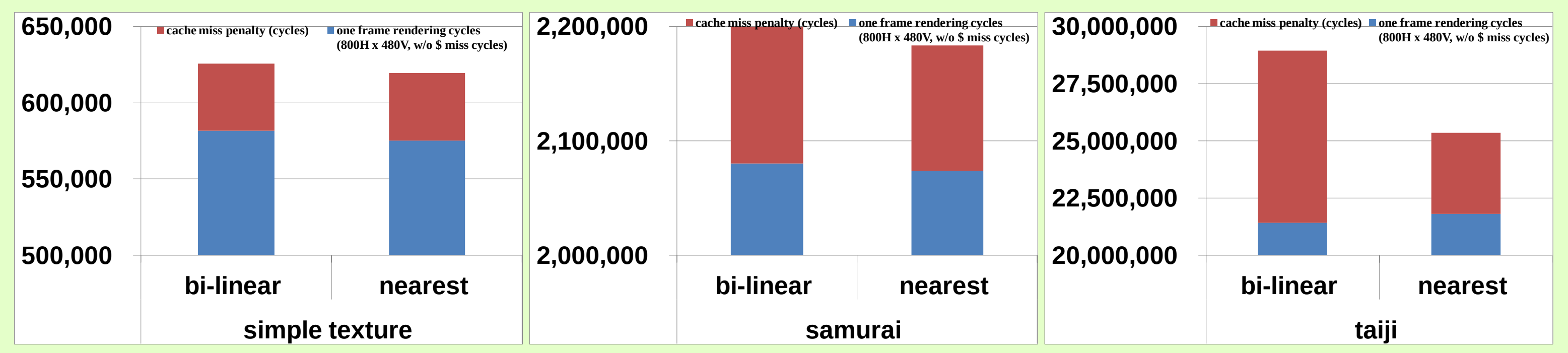
Samurai (OES1.1)

Taiji Girl (OES2.0)


Simple Texture map

Results

- The proposed scheme provides one-cycle latency for cache hit. regardless of number of texels need to be fetched simultaneously. Practically, current implementation supports up to 4 texels.
- Pixel pipeline throughput does not affected due to texture filtering mode, nearest or bi-linear.



- cache configs for this results: total cache memory 32KB (8KB for each sub-caches), 4 texels per cache block, direct mapped cache architecture

References

[1] Seokyoon Jung, *et al*, *Device and Method of Reading Texture Data for Texture Mapping*, Korea IPO 10-2010-0050107, 2010

[2] Tomas Akenine-Möller, Eric Haines, *Real-time Rendering 2nd edition*, A K Peters, 2002

[3] Won-Jong Lee, *et al*, *A Scalable GPU Architecture based on Dynamically Reconfigurable Embedded Processor*, HPG 2011

[4] Ziyad S. Hakura, Anoop Gupta, *The Design and Analysis of a Cache Architecture for Texture Mapping*, ACM SIGGRAPH Computer Architecture News, Volume 25 Issue 2, May 1997