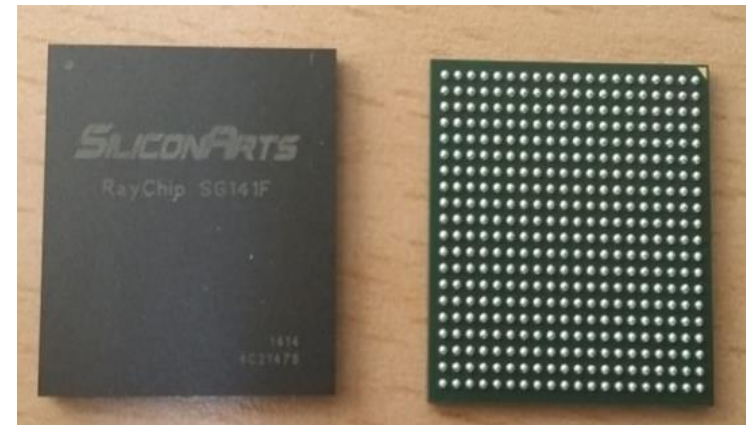
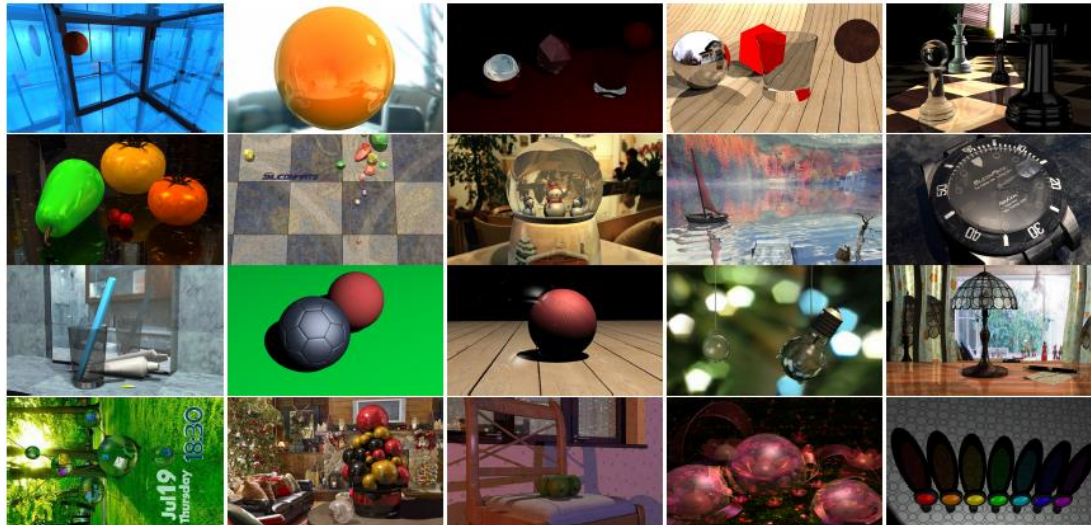


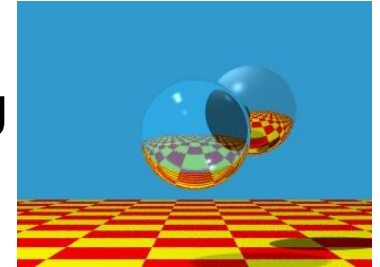
# RayCore: A ray-tracing hardware architecture for mobile devices



Jae-Ho Nah<sup>1,3,4,5</sup>, Hyuck-Joo Kwon<sup>1</sup>, Dong-Seok Kim<sup>1</sup>,  
Cheol-Ho Jeong<sup>2</sup>, Jinhong Park<sup>3</sup>,  
Tack-Don Han<sup>4</sup>, Dinesh Manocha<sup>5</sup>, and Woo-Chan Park<sup>1</sup>

<sup>1</sup>Sejong University, <sup>2</sup>Siliconarts, <sup>3</sup>LG Electronics,  
<sup>4</sup>Yonsei University, <sup>5</sup>University of North Carolina at Chapel Hill

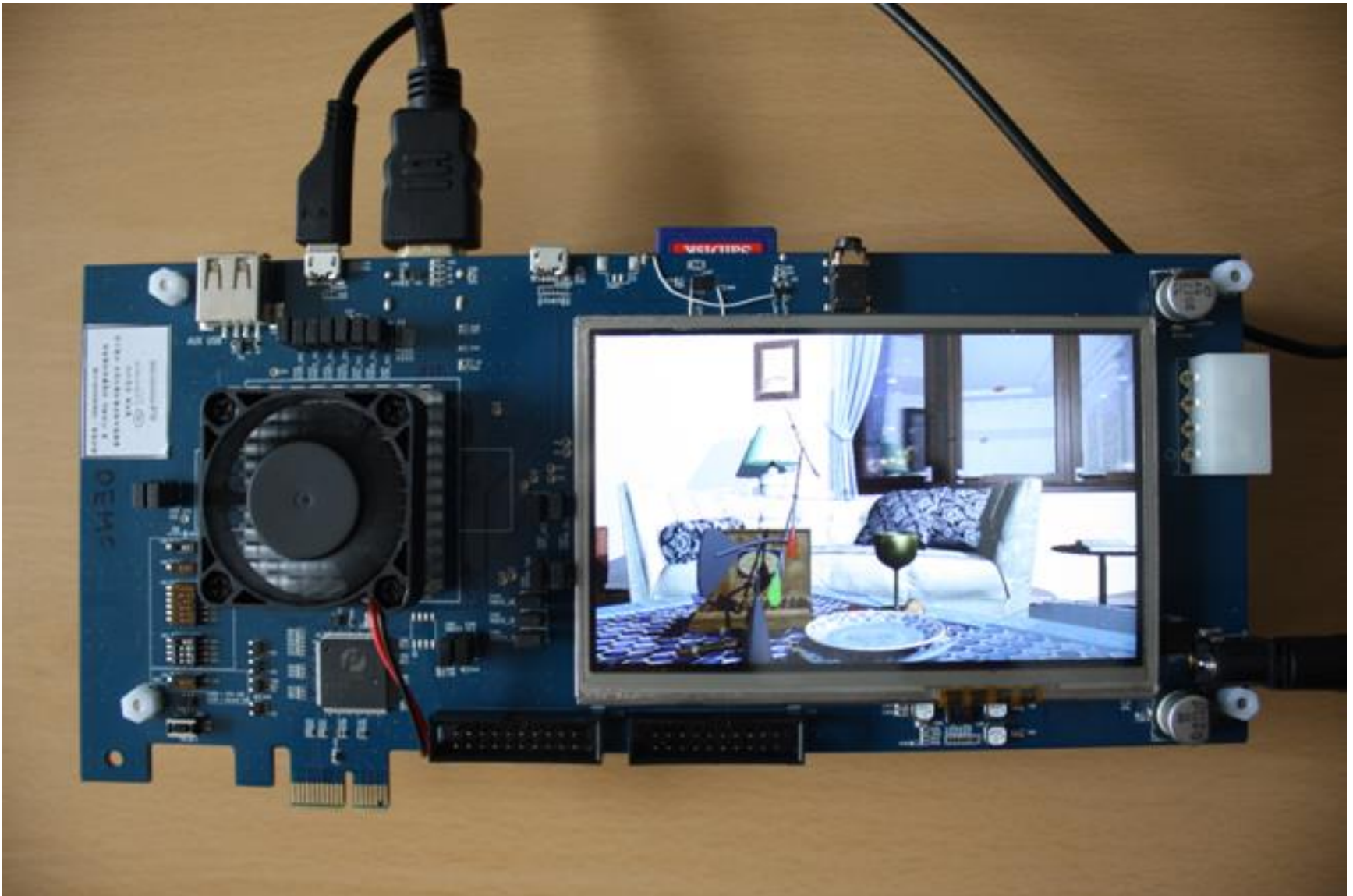
- Ray tracing (RT) [Whitted 1980]
  - Classic rendering algorithm for photo-realistic rendering



- ▶ Our goal
  - “**Real-time**” ray tracing dynamic scenes on “**mobile devices**” for triangulated models



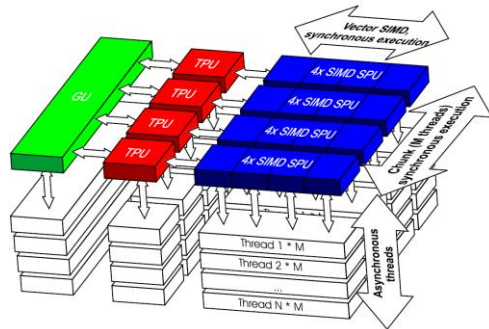
- ▶ Why mobile ray tracing?
  - High interest in generating photo-realistic images at low power cost
  - Ray tracing H/W can be a solution for mobile graphics



# Related Work: Dedicated RT H/W

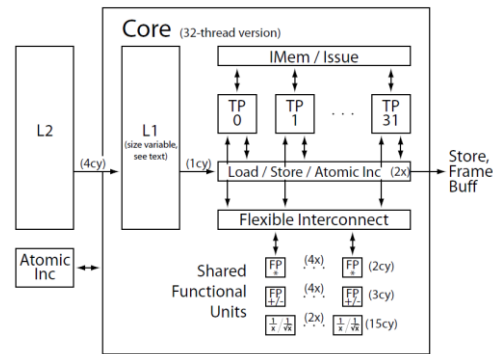
## ▶ RPU/D-RPU

[Woop et al. 2005/2006]



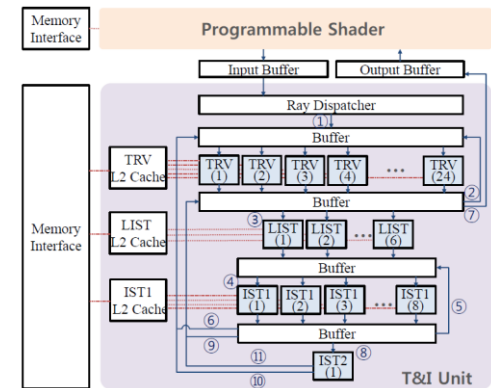
## ▶ TRaX

[Spjut et al. 2009]



## ▶ T&I Engine

[Nah et al. 2011]



## ▶ Mainly focus on high-quality rendering on desktop PCs

- A large chip area & high power consumption for high performance



# Related Work: Mobile RT

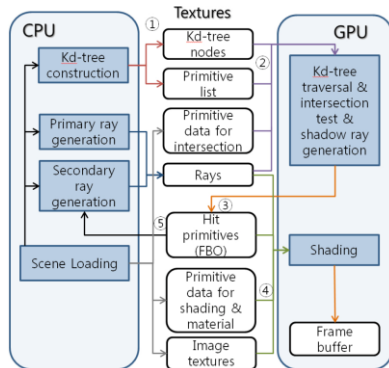
## ▶ Mobile environment

- Very limited area and power budget ( $\sim 20 \text{ mm}^2$ ,  $\sim 2 \text{ W}$  for a GPU)

## ▶ S/W approach

- MobiRT

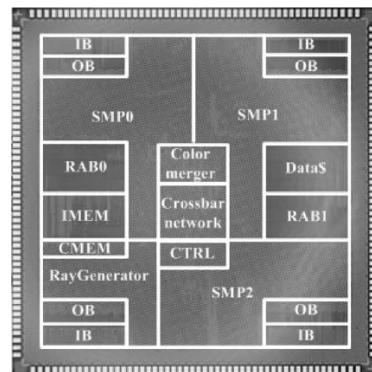
[Nah et al. 2010]



## ▶ H/W approaches

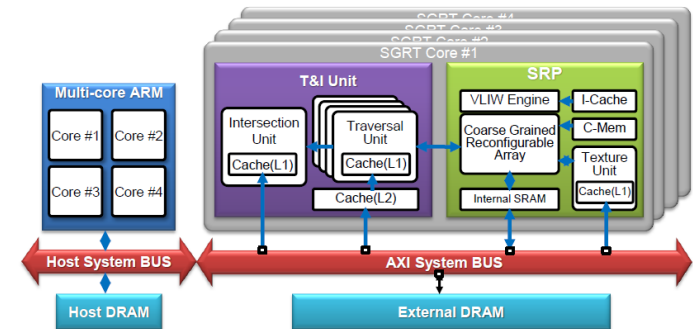
- MRTP

[Kim et al. 2012, 2013]



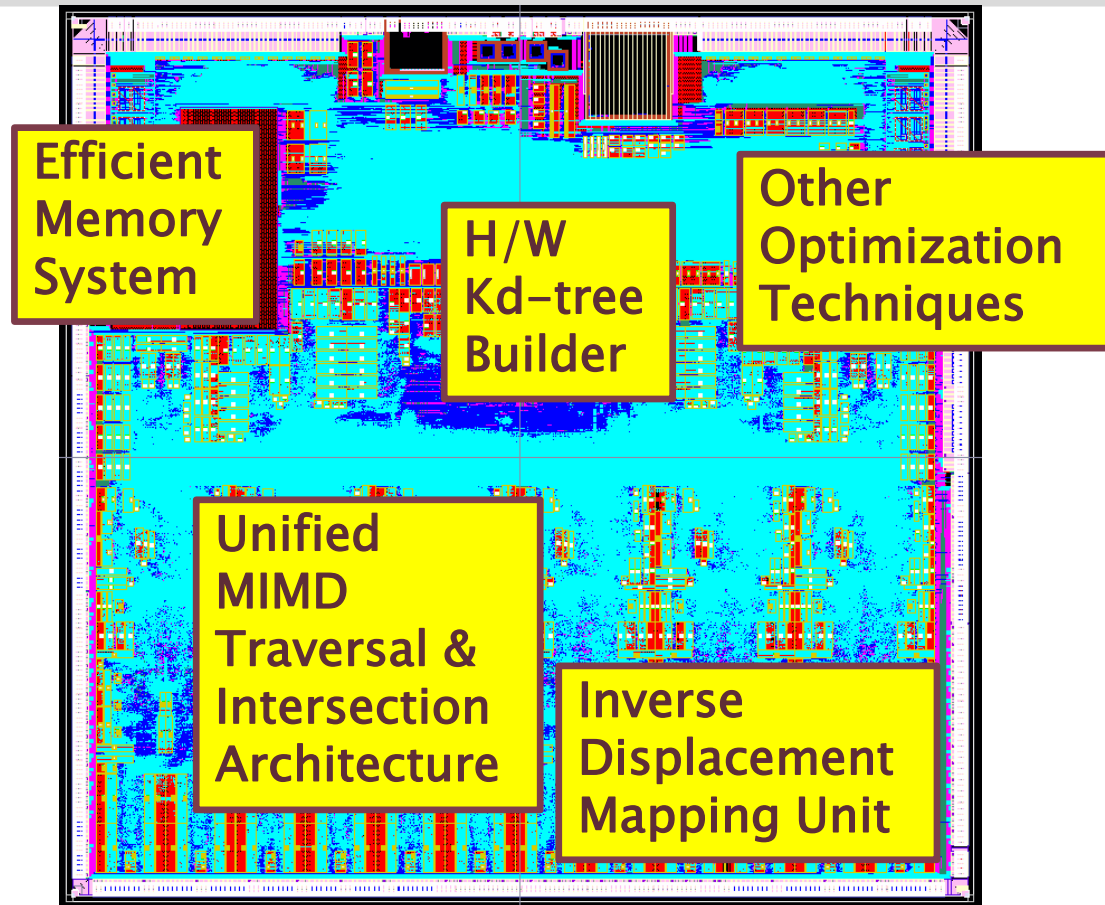
- SGRT

[Lee et al. 2012, 2013]



## ▶ Either no real-time performance or results based on S/W simulations

# Main Results



**Performance:**  
~239M rays/s &  
~6M triangles/s

**Area:**  
20mm<sup>2</sup>  
@28nm HPL

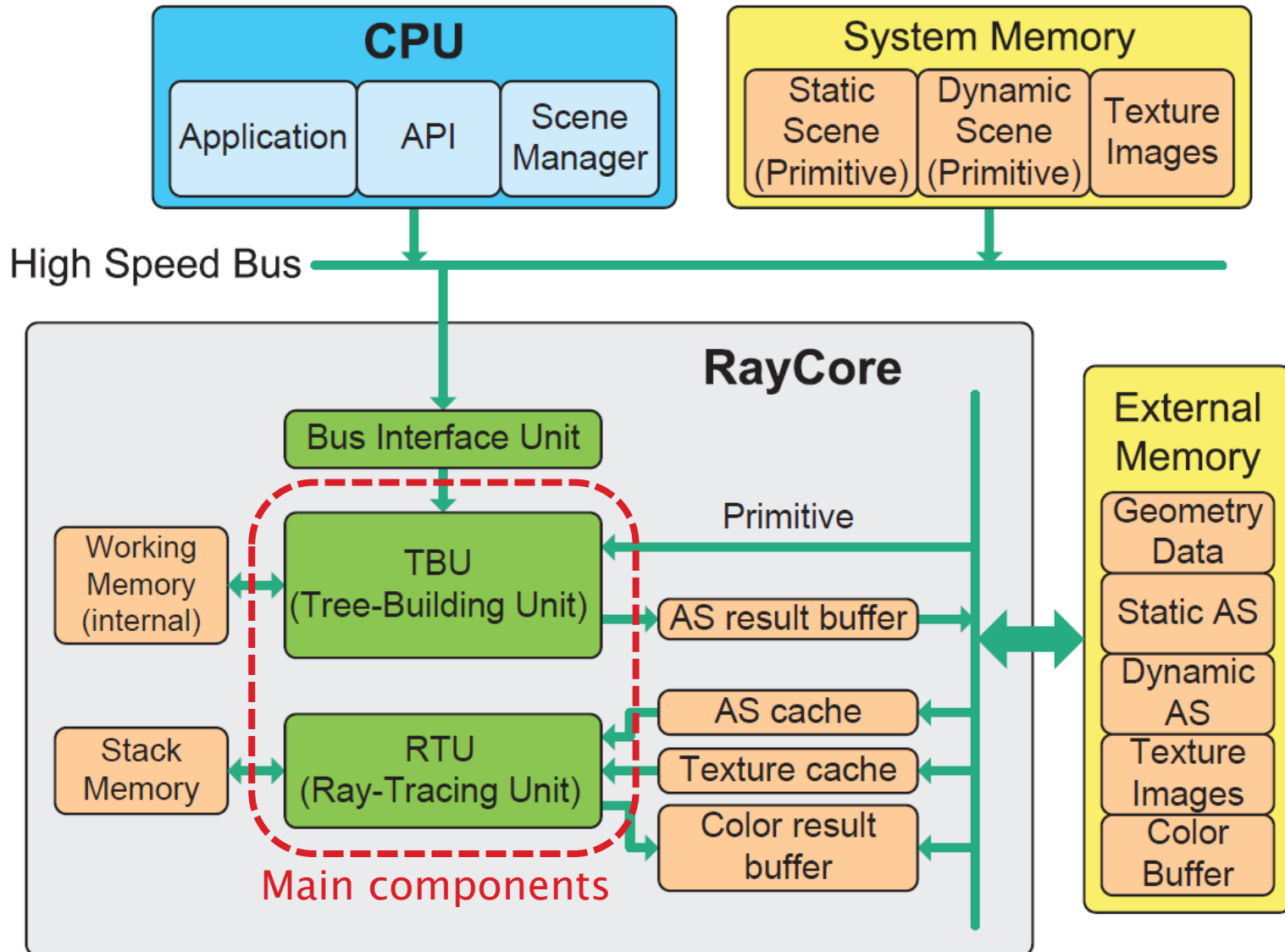
**Power consumption:**  
1W@28nm HPL

~60X faster than  
previous mobile RT H/W

Similar to current mobile GPUs

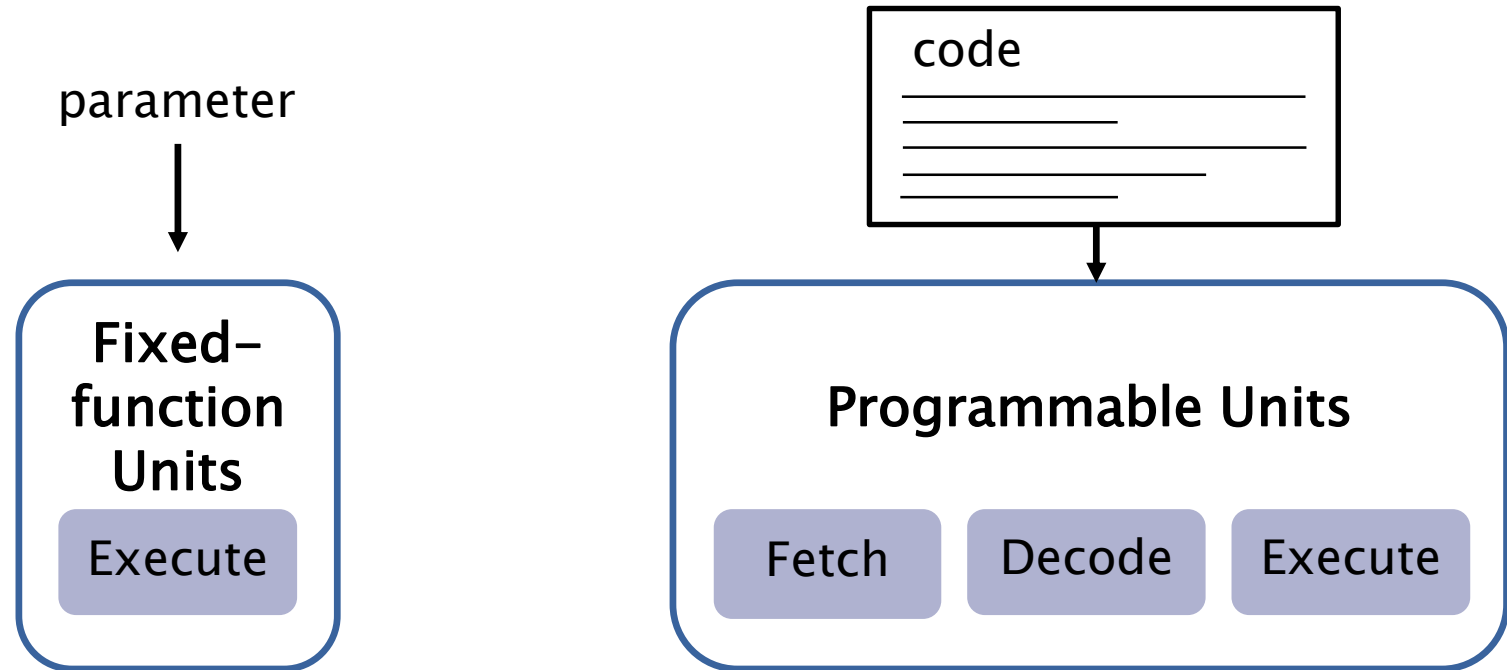
# **Overall H/W Architecture & Design Decisions**

# RayCore H/W Architecture





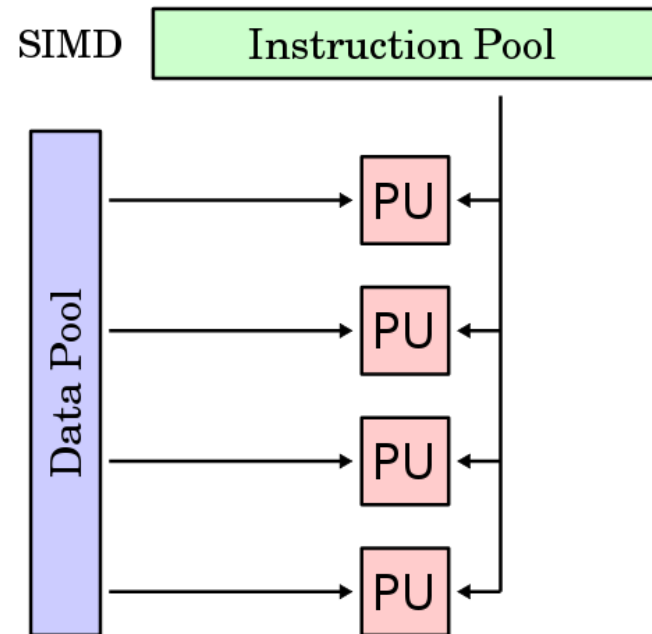
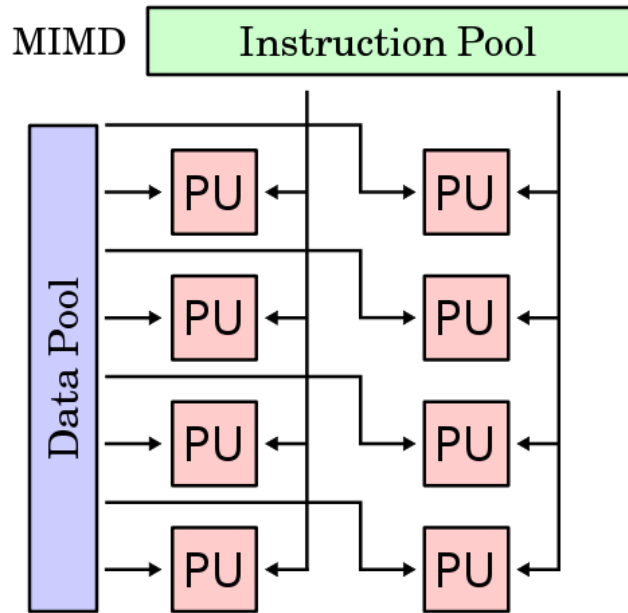
- ▶ Fixed-function units **vs** programmable units



✓ Our choice

- High area and power efficiency

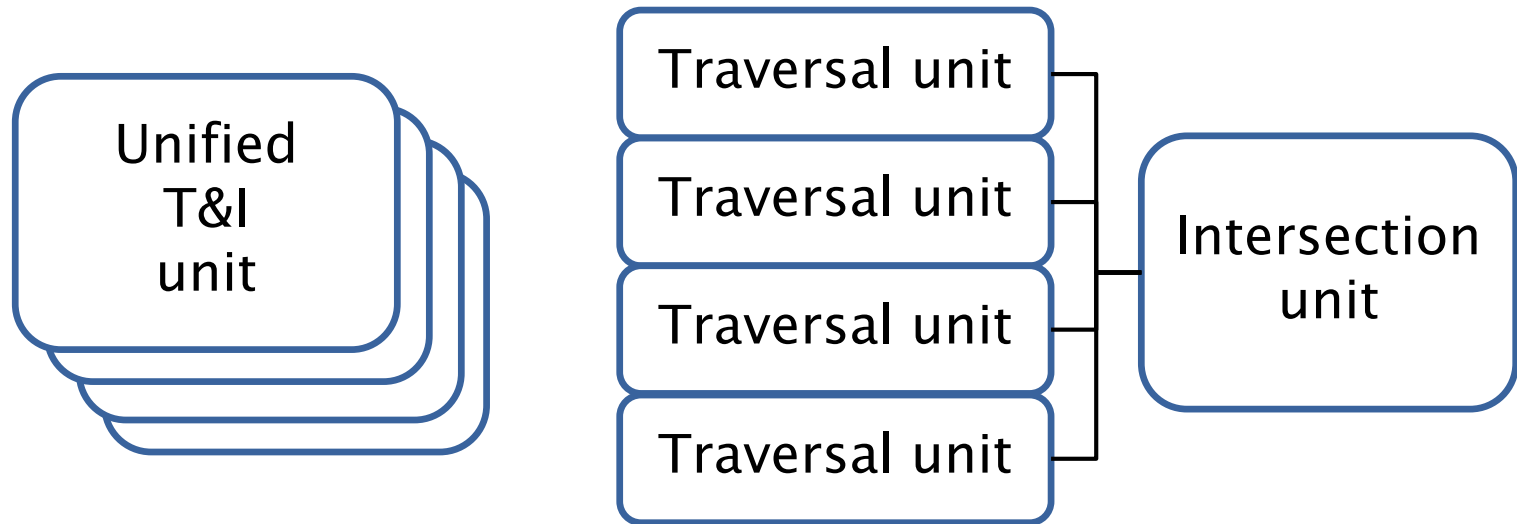
## ► MIMD vs SIMD



✓ Our choice

- High H/W utilization regardless of ray coherence

- ▶ Unified traversal & intersection (T&I) units **vs** separate T&I units

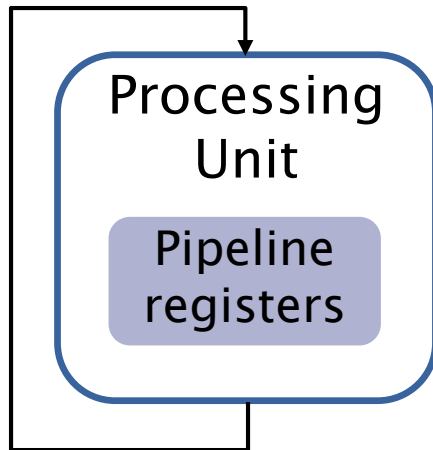


✓ Our choice

- No load imbalance problem in prior separate T&I units

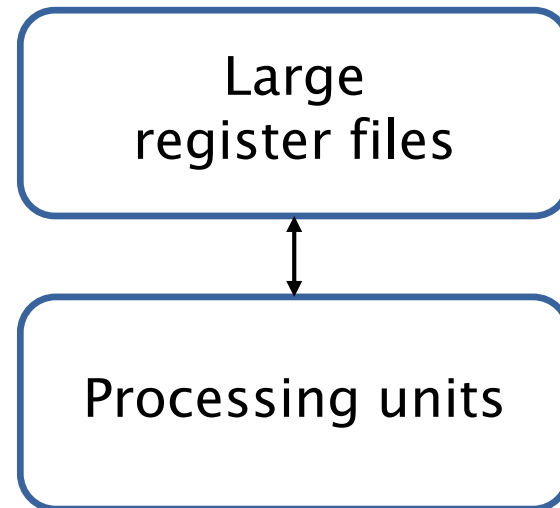
## ▶ Multi-threading method

Looping for  
the next chance



✓ Our choice

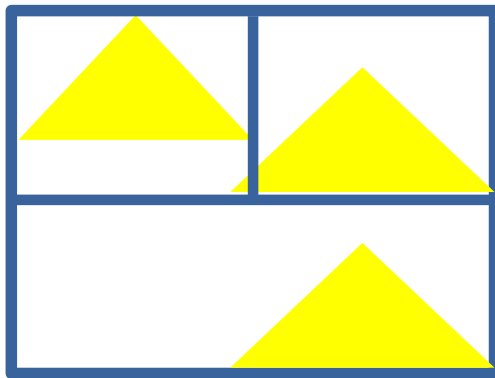
GPU-style  
H/W multi-threading



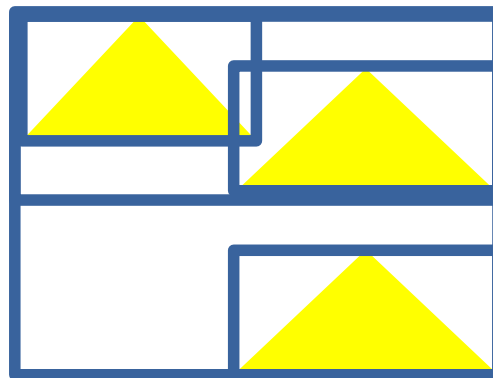
- Reuse existing buffers/registers to minimize additional register files

## ► Acceleration structure (AS)

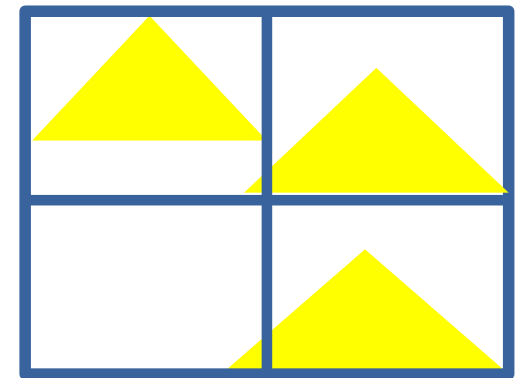
Kd-tree



BVH



Grid



✓ Our choice

- Fast traversal with early termination
- Good cache efficiency with a small node size (8 bytes)
- Our H/W tree builder solves the tree-build time problem

# Rendering effects

- ▶ Full Whitted ray-tracing effects



Specular reflection



Refraction



Shadows



- ▶ Distribution ray-tracing effects



Ambient occlusion (AO)



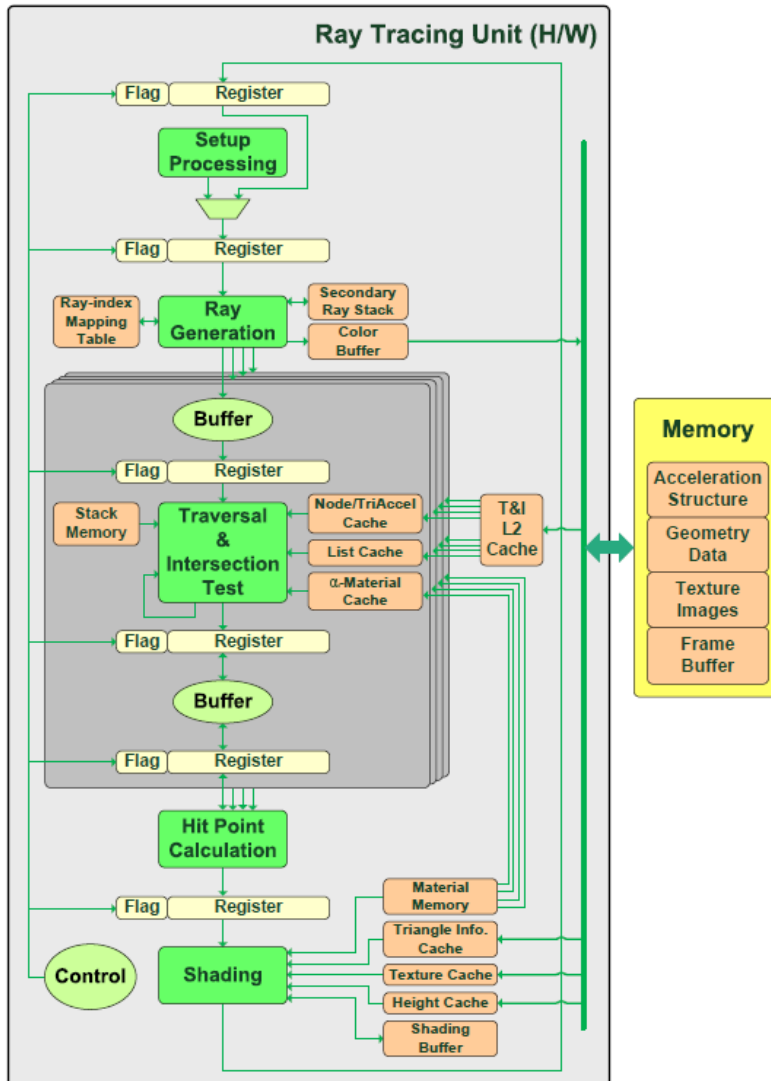
Diffuse inter-reflection

- ▶ Inverse displacement mapping



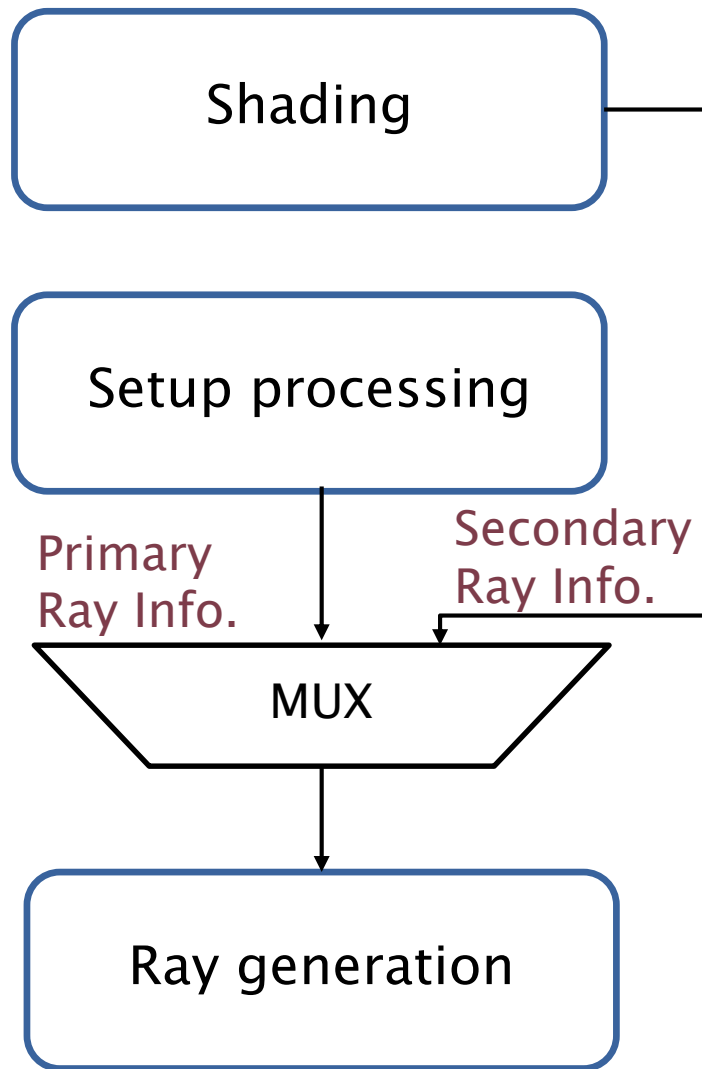
# **Ray-Tracing Unit (RTU) & Tree-Building Unit (TBU)**

# Ray-Tracing Unit (RTU)

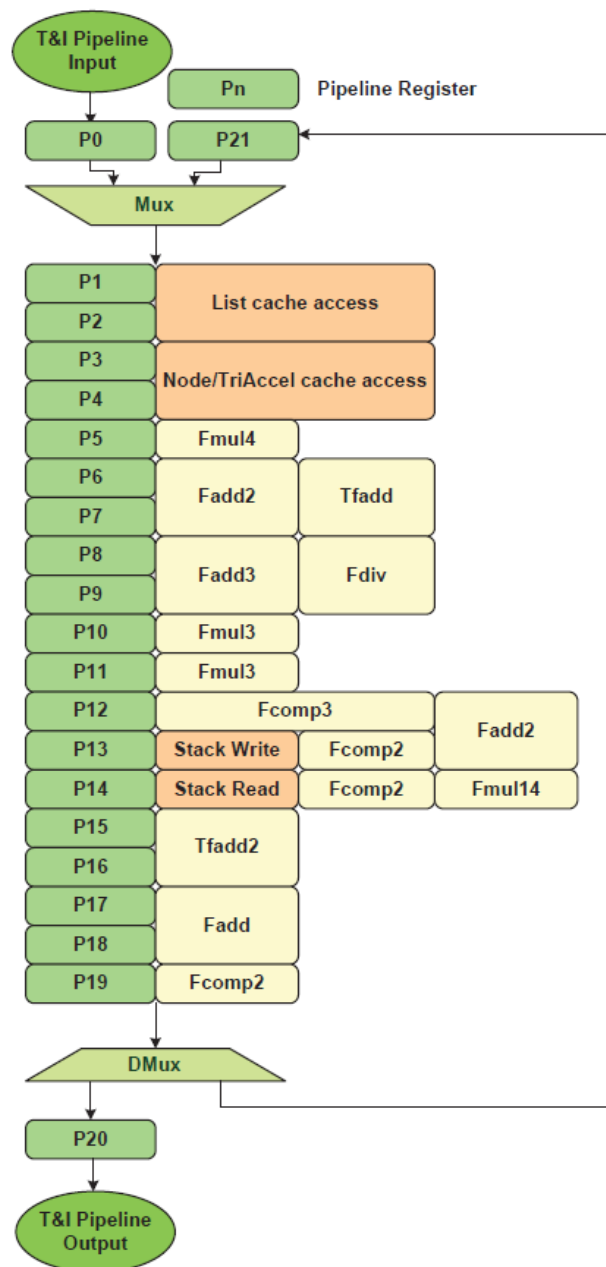


- ▶ Setup-processing unit
- ▶ Ray-generation unit
- ▶ Multiple traversal & intersection (T&I) units
- ▶ Hit-point calculation unit
- ▶ Shading unit

# Setup Processing & Ray Generation



- ▶ Initialize primary-ray information (ray type+ray index)
- ▶ Pass secondary-ray information defined by the shading unit
- ▶ Generate primary/secondary/shadow rays



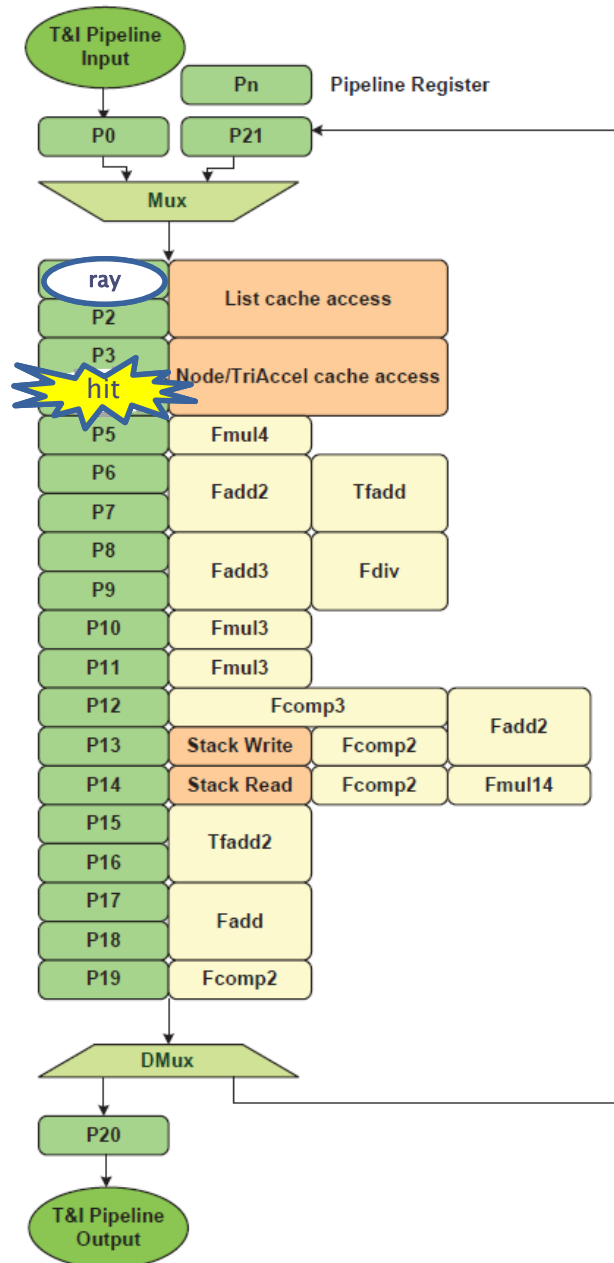
## ► A single pipeline with three modes

	Ray-Box Intersection Test Mode	Traversal Mode	Ray-Triangle Intersection Test Mode
P1			List cache access
P2			
P3			TriAccel cache access
P4		Node cache access	
P5			Fmul (0-3)
P6	Fadd (0-1)		Fadd (0-1)
P7			TFadd (0)
P8	Fadd (0-2)	Fadd(0-1)	Fadd (0-2)
P9			Fdiv (0)
P10	Fmul (0-2)		Fmul (0)
P11	Fmul (0-2)	Fmul(0-1)	Fmul (0-1)
P12	Fcomp (0-2)	Fcomp (0-3)	
P13	Fcomp (0-1)	Stack write	Fadd (0-1)
P14	Fcomp (0-1)	Stack read	Fmul (0-3)
P15	Fcomp (0)		TFadd (0-1)
P16			
P17			Fadd (0)
P18			
P19			Fcomp (0-1)

- No load imbalance problem
- Greatly simplified control logic and interfaces between units

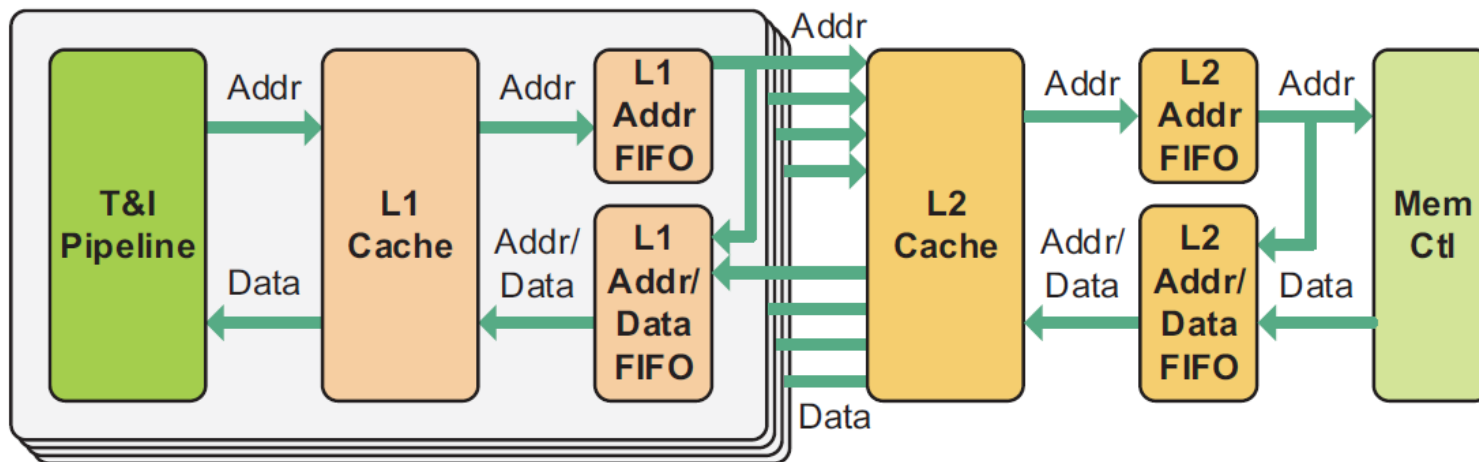


# Looping for the Next Chance



- ▶ Simple multi-threading technique
- ▶ Operation
  - Cache miss → idle
  - Next loop → reactive
  - A cache miss acts as prefetching data
- ▶ Pros
  - Ease of H/W implementation
  - Use of existing internal memory (input/output buffers & pipeline registers)

- ▶ Two-levels of caches (L1/L2)
  - L1/L2 Address FIFO for handling memory requests
  - L1/L2 Address/Data FIFO for delivering address & data to the upper-level cache



- ▶ Hit-point calculation

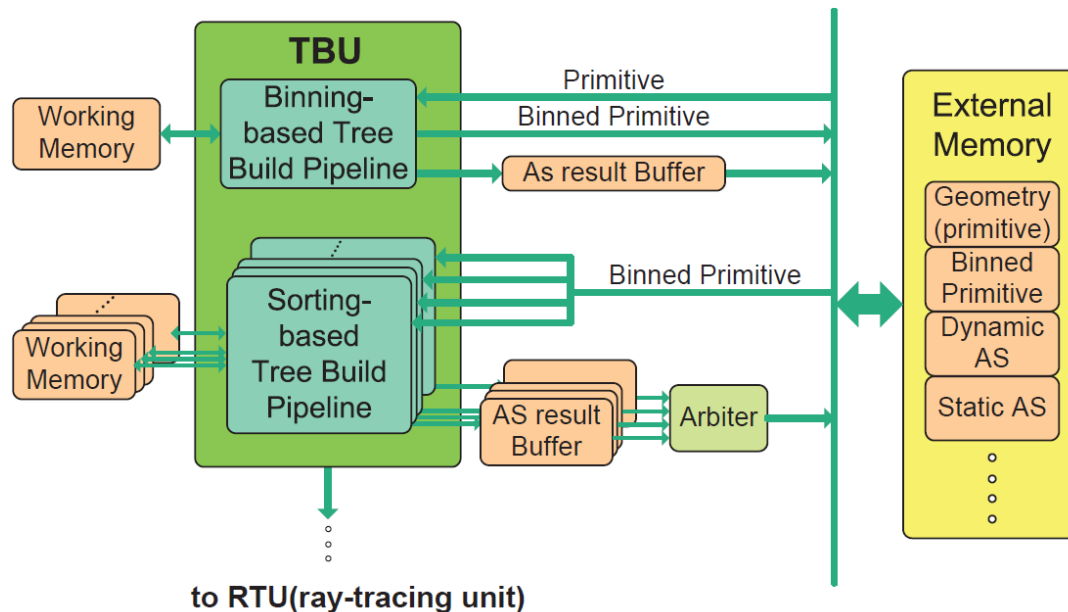
- Calculate the final (x,y,z) position

$$p(t) = o + t \cdot d$$

- ▶ Shading

- Phong illumination
  - Texture mapping with bilinear filtering & mip-mapping
  - Inverse displacement mapping

- ▶ Two pipeline types
  - Binning for upper-level nodes + sorting for lower-level nodes
  - Fast kd-tree construction without significant tree-quality degradation
- ▶ Memory traffic reduction techniques
  - Internal working memory for local tree-building procedure
  - Node scheduling for burst memory accesses



- ▶ OpenGL ES 1.1-like API
  - Provides similar interfaces to OpenGL ES programming
  - Static objects/tree are retained for subsequent frames
  - Dynamic objects/subtrees are updated during each frame
- ▶ Complete specification and its programming guide are provided on the Siliconarts homepage ([www.siliconarts.com](http://www.siliconarts.com))

```
// set a box
rcStaticSceneBegin();

...

rcVertexPointer(3, RC_FLOAT, 0, box);
rcGenMaterials(1, &material_box);
rcBindMaterial(material_box);
rcMaterialf (RC_FRONT_AND_BACK, RC_REFLECTION, 0.0f);
rcMaterialfv(RC_FRONT_AND_BACK, RC_DIFFUSE, &cyan.r);
rcDrawArrays(RC_TRIANGLES, 0, 30);

rcDisableClientState(RC_VERTEX_ARRAY);
rcStaticSceneEnd();
```

# Implementation and Results



## ▶ iNEXT-V6 board

- 2 Xilinx Virtex-6 LX550 FPGA chips
- 2 GB of DDR3 RAM & 8 MB of SRAM
- A TFT LCD board with 800x480 resolution
- PCI Express interface



## ▶ Our implementation

- 4 FPGA chips for 4 RTUs (with 2 iNEXT-V6 boards)
- 1 FPGA chip for 1 TBU
- 84 MHz core and memory clock
- Total required SRAM: 507KB for an RTU and 218KB for a TBU

- ▶ TSMC's 28 nm HPL process and Synopsys design compiler
- ▶ Clock frequency: 500MHz@0.9V
- ▶ Area: 3mm<sup>2</sup> per RTU (18mm<sup>2</sup> for 6 RTUs) + 1.6mm<sup>2</sup> per TBU
- ▶ Internal power consumption : 1W for 6 RTUs and 1 TBU
- ▶ Particularly suitable for mobile devices

## ▶ Whitted ray tracing



Kitchen



Moving light



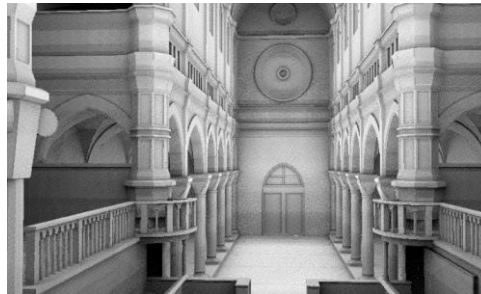
Living room

- 4 RTUs @ 84MHz
- 3-5 rays per pixel
- Performance: 23-26 Mrays/s
- Memory traffic: 46-122 MB/s
- FPS: **13-21** @ 840 X 480 resolution
- **Interactive** frame rates

## ► Distribution ray tracing



Conference



Sibenik

- 4 RTUs @ 84MHz
- 2 (**primary**), 16 (**AO**), 32 (**diffuse**) rays per pixel
- Performance: 21 (**primary**), 23 (**AO**), 18–20 (**diffuse**) Mrays/s
- Memory traffic: 8-62 (**primary / AO**), 420-605 (**diffuse**) MB/s
- Low performance degradation when tracing **incoherent** rays

- ▶ Inverse displacement mapping (IDM)

BART Kitchen



(IDM off)

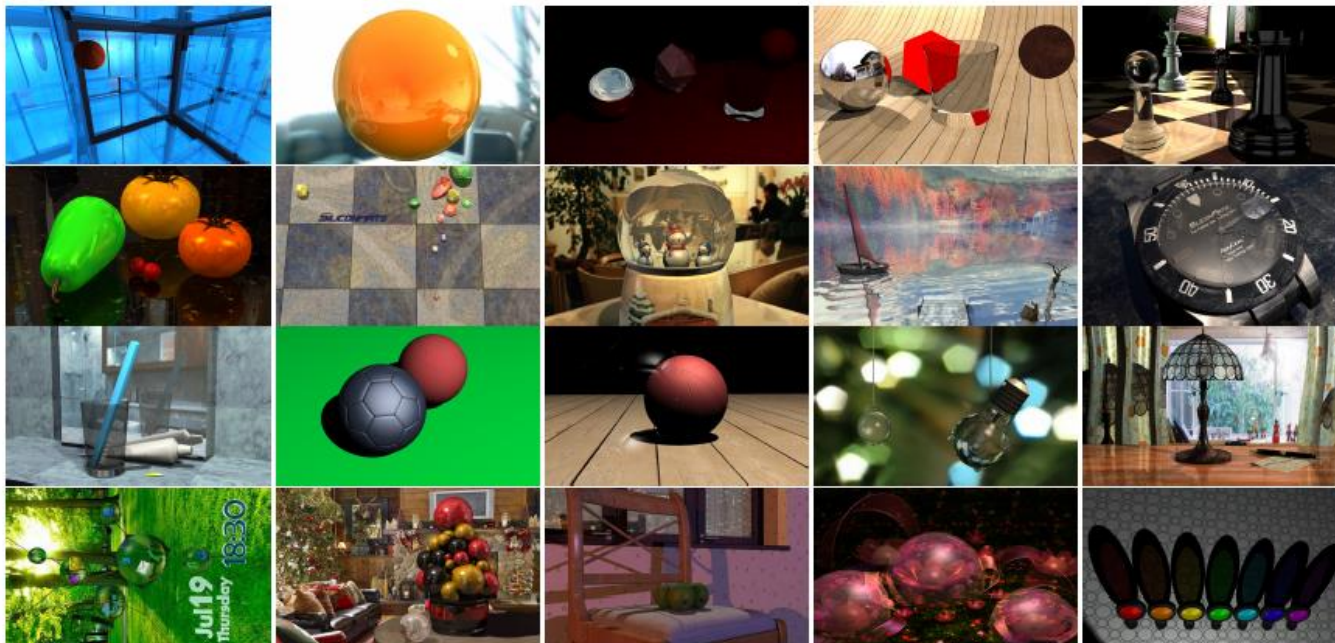


(IDM on)

- 4 RTUs @ 84MHz
- Performance: 18-20 Mrays/s
- 30% increase of memory traffic to access the height map
- Detailed results are included in  
Kwon *et al.*, Effective traversal algorithms and hardware architecture for  
pyramidal inverse displacement mapping, *Computers & Graphics*, 2014

# KD-tree Construction Performance

- ▶ Performance (1 TBU)
  - 1.6~117.9 ms/frame @ 84MHz for 0.6~64K triangles
  - 0.4~1M triangles/s
- ▶ Memory traffic
  - 0.1~36.1 MB/frame





## ▶ RTU performance

- 9X faster than the FPGA ver. (84→500MHz & 4→6 RTUs)
- ~239M rays/s & 56 FPS@720p

## ▶ TBU performance

- 6X faster than the FPGA ver. (84→500MHz & 1 TBU)
- 2~6M triangles/s

## ▶ Required memory bandwidth

- 1.1GB/s on 6 RTUs for Whitted ray tracing
- 1.1GB/s on 1 TBU for 30FPS kd-tree construction
- Much less than the bandwidth of dual LPDDR3 1333MHz (12.8GB/s)

# Preliminary Comparison

- ▶ Ray-tracing performance in the Conference scene



	GPU [Gribble and Naveros 2013]	MIC [Benthin et al. 2012]	Mobile RT H/W [Kim et al. 2013]	RayCore ASIC (ours)
<b>Mrays/s</b>	500	210	4	239
<b>Platform</b>	NVIDIA GTX690	Intel MIC	Reconf. SIMT	RTU
Process(nm)	28	45	90	28
Clock (MHz)	915	1200	50-400	500
<b>Area (mm<sup>2</sup>)</b>	294 X 2	–	16	18 (6 RTUs)
<b>Power consumption(W)</b>	300 (TDP)	–	0.2 @100MHz	1 (RTU+TBU)

- Desktop-level performance
- Mobile-level area and power consumption

## ► KD-tree construction performance

	CPU [Shevtsov et al. 2007]	GPU [Hou et al. 2011]	RayCore ASIC (ours)
Time to build a kd-tree (ms)	27	38	20
Scene (# of tris)	Bunny (69K)	Robots (71K)	Transparent Shadows (64K)
Platform	Intel Core2 Duo X2	NVIDIA GTX280	TBU
Process	65	55	28
Clock (MHz)	3000	1476	500
Area (mm <sup>2</sup> )	143 X 2	576	1.6 (1 TBU)
Power consumption(W)	65 X 2 (TDP)	236 (TDP)	1 (RTU+TBU)

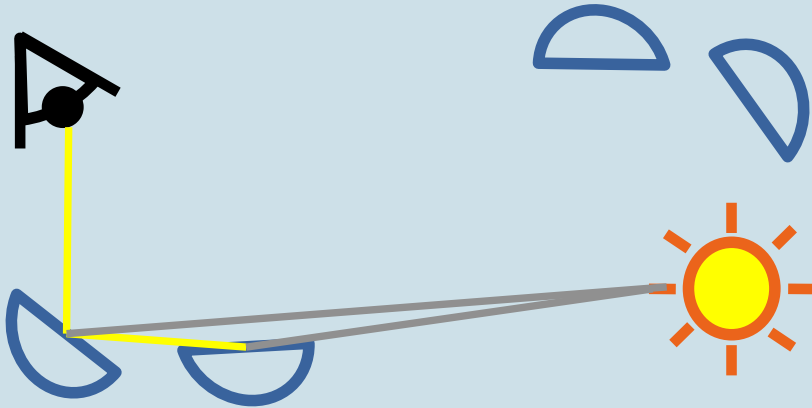
- Comparable performance to CPU/GPU approaches & much less H/W resources/power consumption

# **Conclusions, Limitations, and Future Work**

# Limitations and Future Work

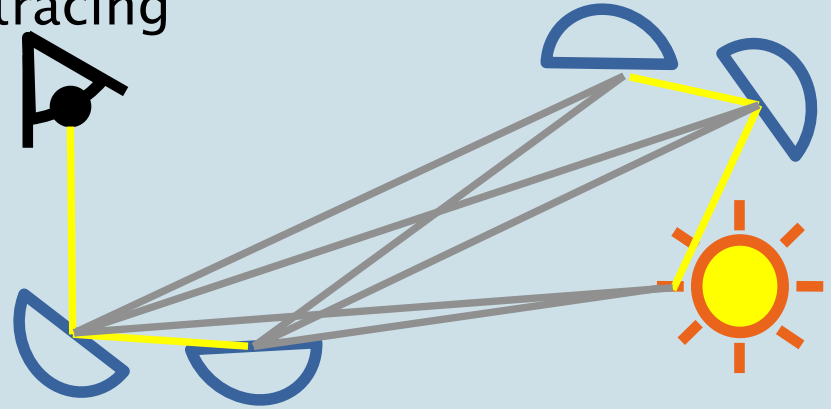
## Limitations

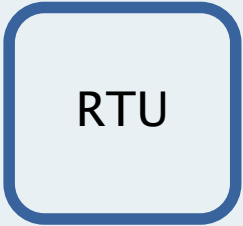
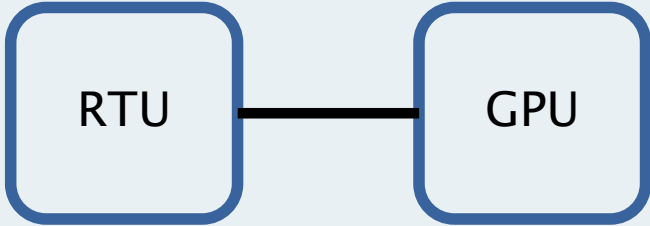
Focus on Whitted ray tracing

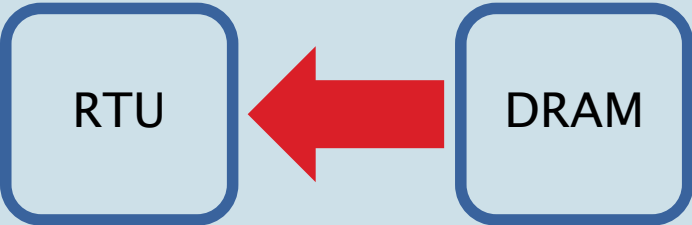
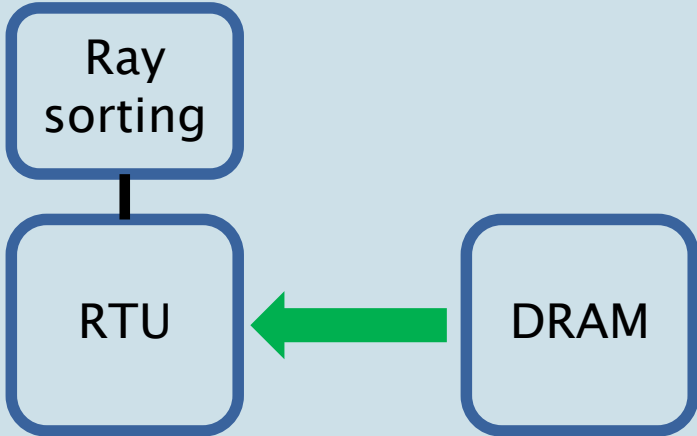


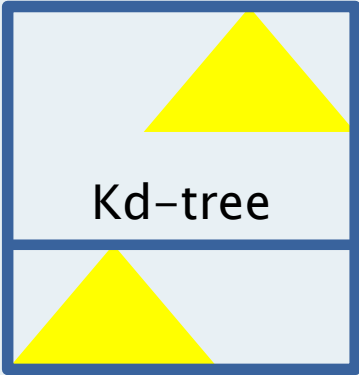
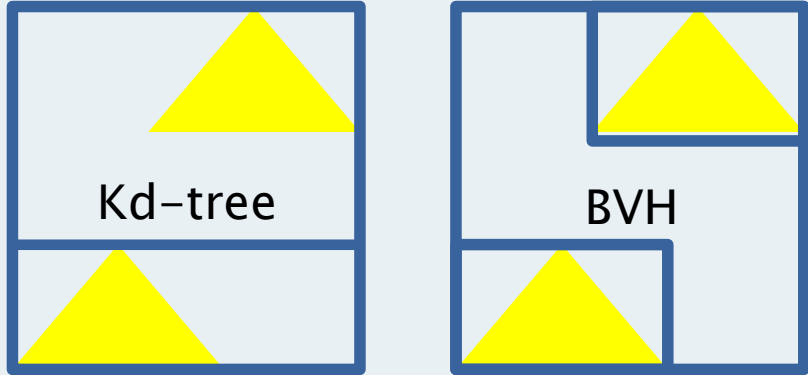
## Future work

To accelerate bidirectional path tracing



Limitations	Future work
Focus on Whitted ray tracing	To accelerate bidirectional path tracing
Fixed pipelines	To combine with programmable shaders
	

Limitations	Future work
Focus on Whitted ray tracing	To accelerate bidirectional path tracing
Fixed pipelines	To combine with programmable shaders
High memory traffic on incoherent ray tracing  	Additional ray-sorting logic  

Limitations	Future work
Focus on Whitted ray tracing	To accelerate bidirectional path tracing
Fixed pipelines	To combine with programmable shaders
High memory traffic on incoherent ray tracing	Additional ray-sorting logic
Only support kd-trees 	To support both kd-trees and BVHs 



- ▶ A new hardware ray tracer for mobile devices including
  - Unified T&I pipelines
  - H/W kd-tree builder
  - Other various novel techniques
- ▶ RayCore can be used for various mobile applications
  - Games, UX, AR, etc.
  - High-quality images & simpler programming



# Acknowledgements

Siliconarts, NRF, ARO & NSF

## Q&A