ACCELERATING RAY-TRACING ON FPGA

BA Semester project - Andrew Dobis
Supervisor: Mikhail Asiatici

PROJECT GOAL

- Design a Hardware Accelerator for Raytracing that's capable of running on an FPGA

- Make use of Mikhail's memory system when fetching the data associated to each ray in memory

- Verify that it indeed speeds-up the the Raytracer's unpredictable access patterns

OUTLINE

- What is raytracing: basic algorithm

- How does it work: details about traversal and geometry intersection

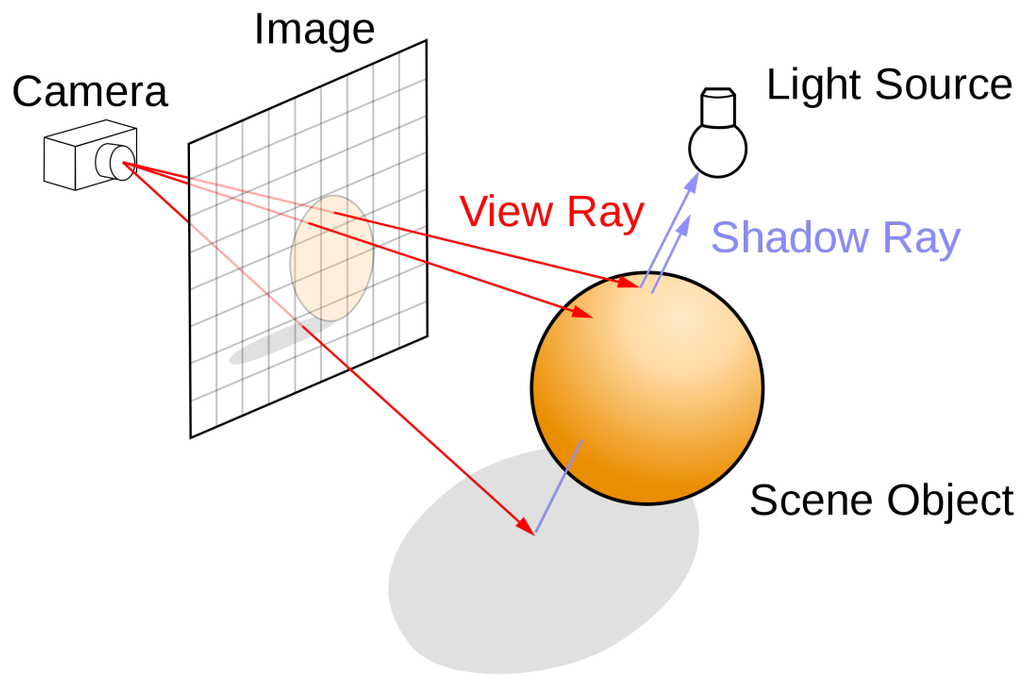
- Overview of the hardware architecture

- Details about the different components: Fetch, Traversal & Intersect

WHAT IS RAYTRACING ?

WHAT IS RAYTRACING ?

- Algorithm for simulating photorealistic light in a CG scene
- Basic algorithm :
    - Project ray from a point (light source or geometry)
    - Ray continues until either it intersects an object or ray fades out
- If we intersect an object: shoot a new ray and repeat



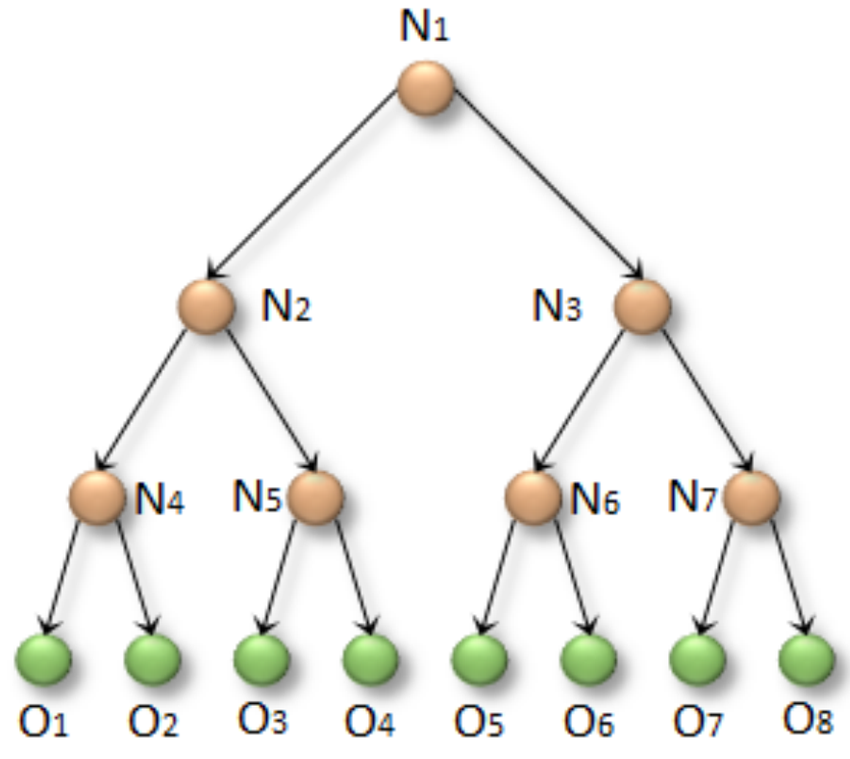
NORI RAYTRACER

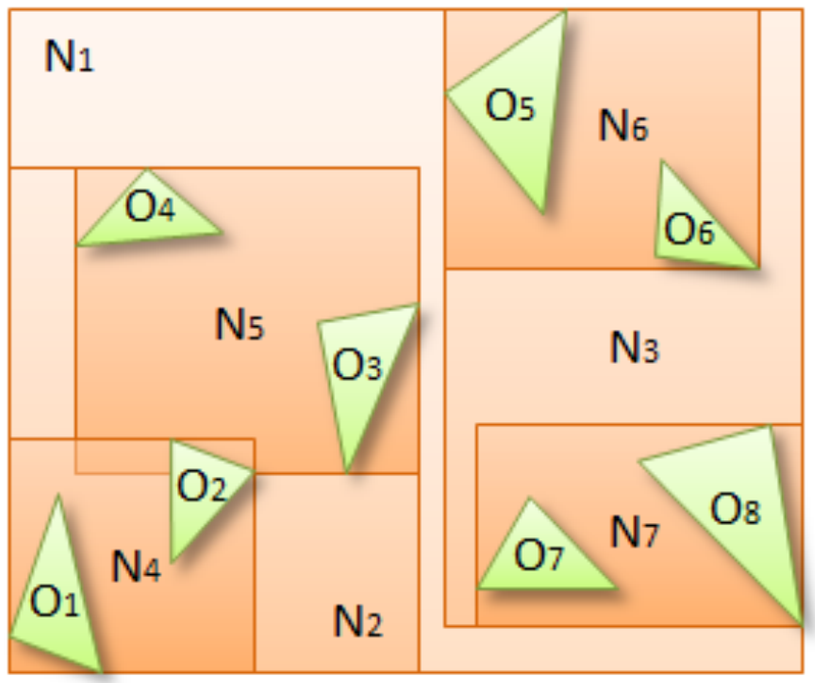
- Raytracer used in the advanced computer graphics course
- Generalized raytracer that works with many materials
  - An accelerator for nori can work for any other raytracer

- Heaviest function: RayIntersect (~80% of the computation time) => This is what we will accelerate
  - Takes care of scene **traversal** & **intersection** detection

TRAVERSAL & INTERSECTION

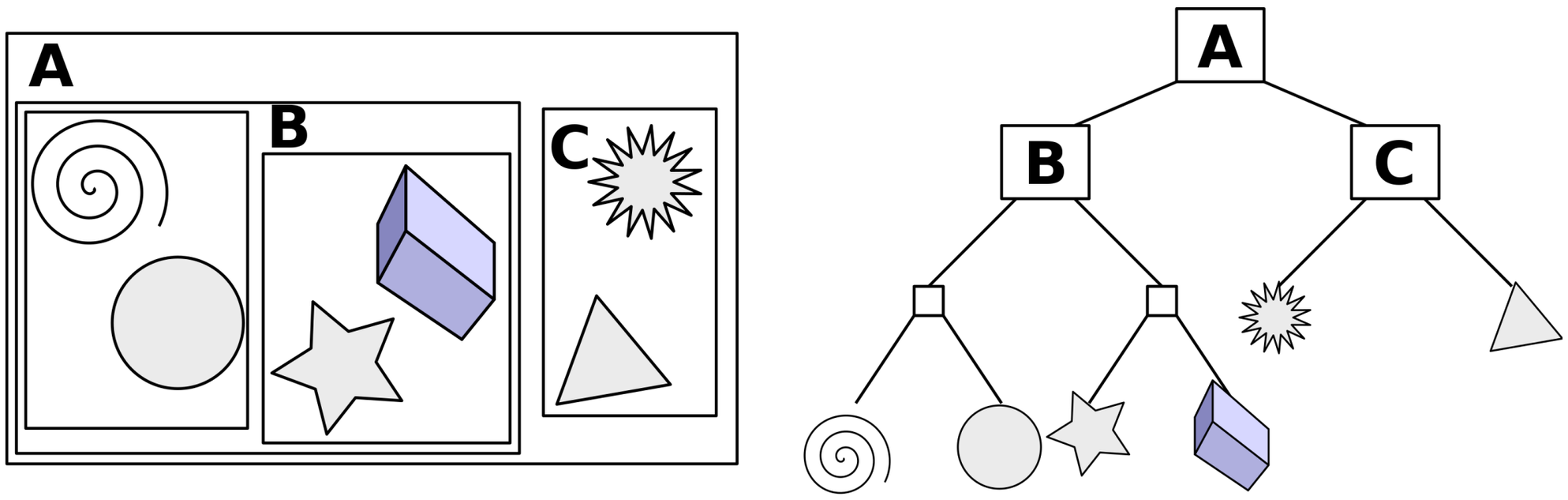
How does it work?





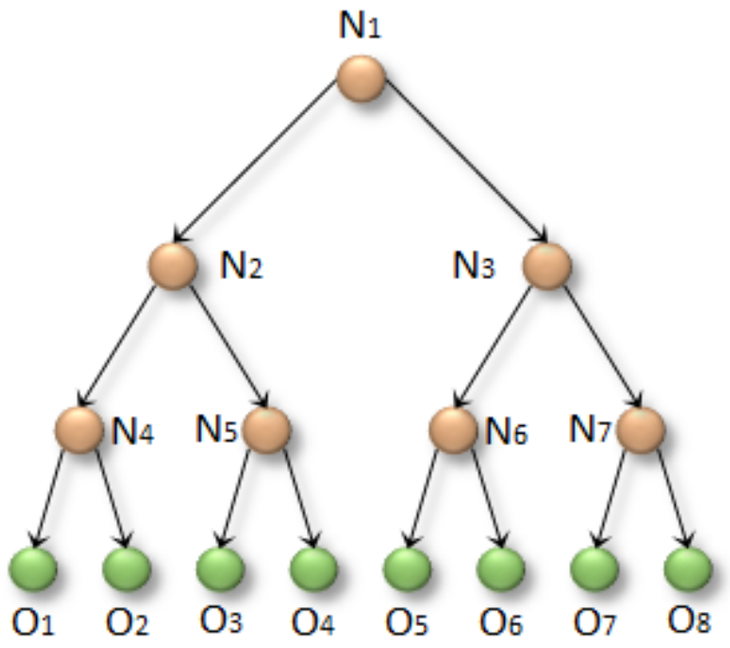
TRAVERSAL: HOW DOES IT WORK ?

- Organize the scene by grouping primitives (triangles) with a conservative bounding box and then create a hierarchy where each box is then grouped with neighbouring boxes:
    - This is called a **Bounding Volumes Hierarchy (BVH)** which contains all of the primitives in the scene



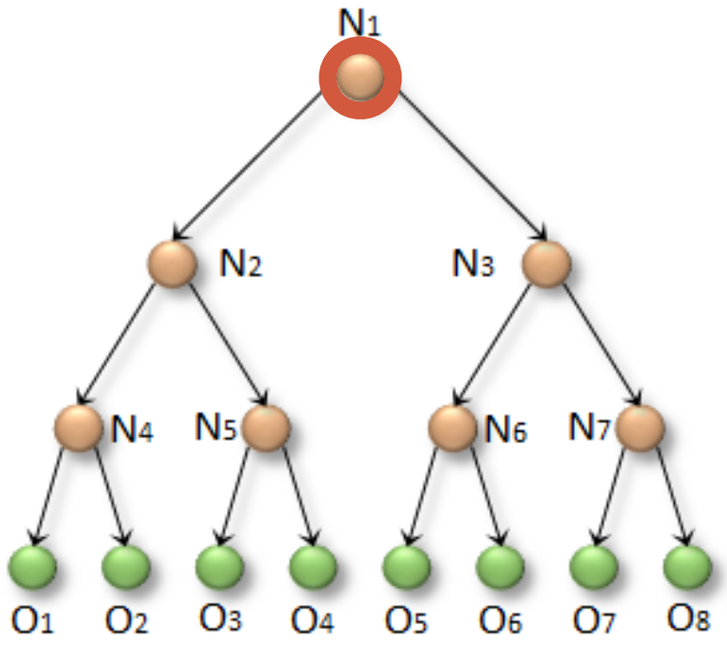
TRAVERSAL: HOW DO WE DO IT?

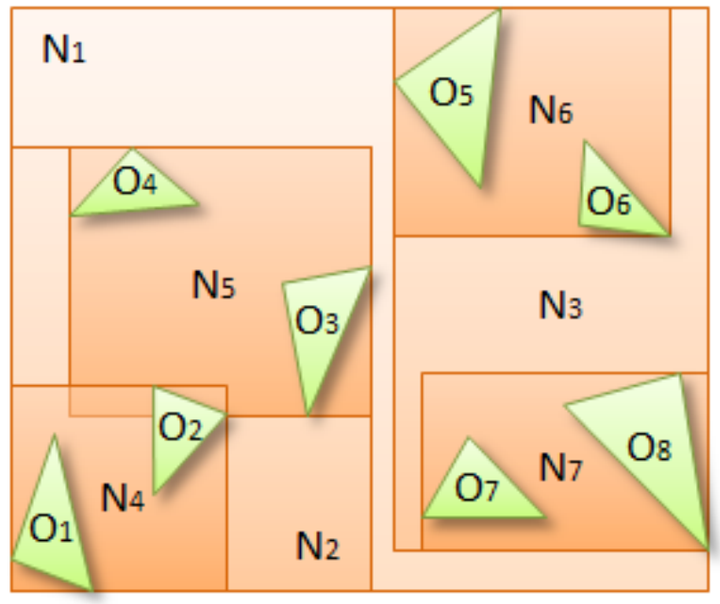
- We start by looking at the first two children of the root.
  - if we intersect with with one of them, then go to the next two children of that node.
  - else try intersecting with the other child's bounding box.
- Continue doing that until you reach a leaf for which we have intersected with its bounding box.



TRAVERSAL: HOW DO WE DO IT?

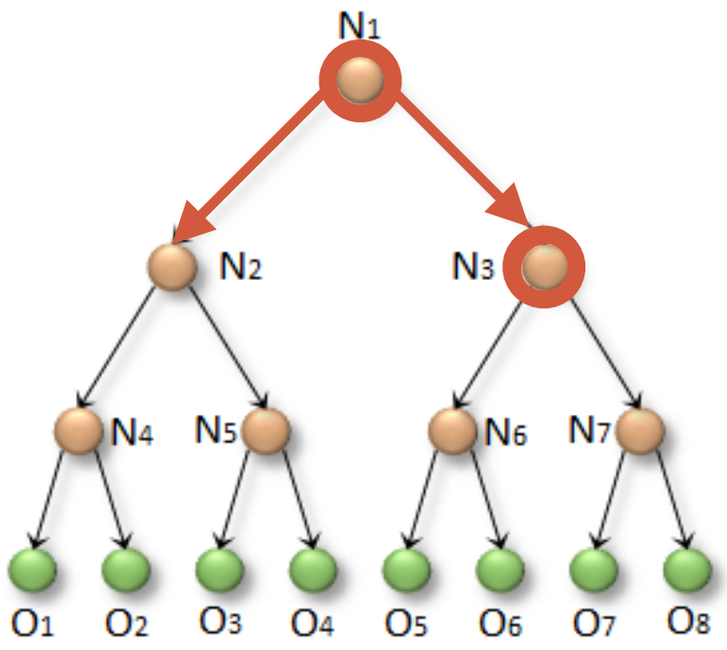
- We start by looking at the first two children of the root.
  - if we intersect with with one of them, then go to the next two children of that node.
  - else try intersecting with the other child's bounding box.
- Continue doing that until you reach a leaf for which we have intersected with its bounding box.

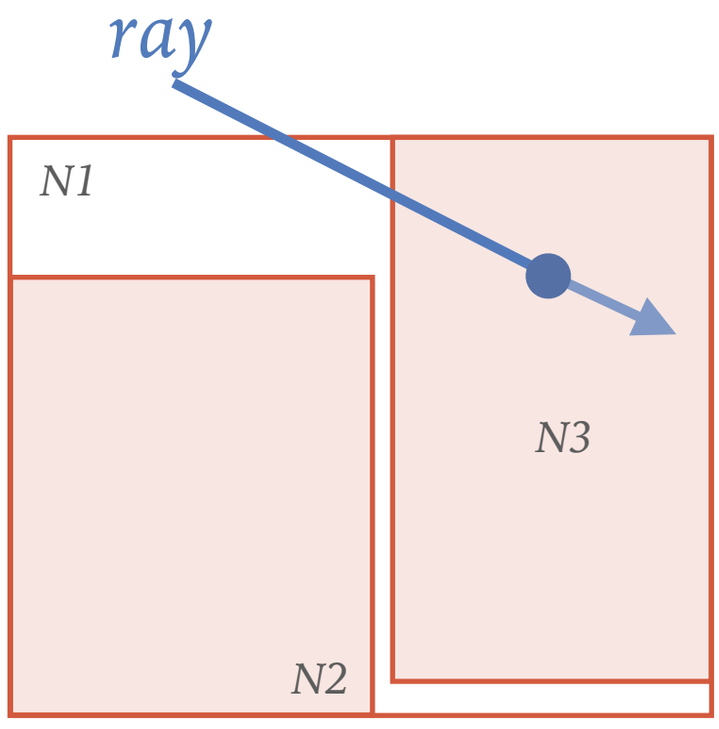




TRAVERSAL: HOW DO WE DO IT?

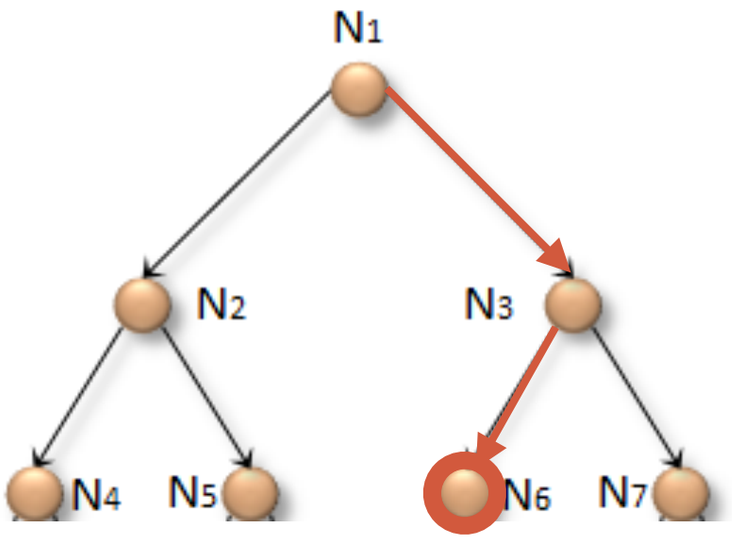
- We start by looking at the first two children of the root.
  - if we intersect with with one of them, then go to the next two children of that node.
  - else try intersecting with the other child's bounding box.
- Continue doing that until you reach a leaf for which we have intersected with its bounding box.

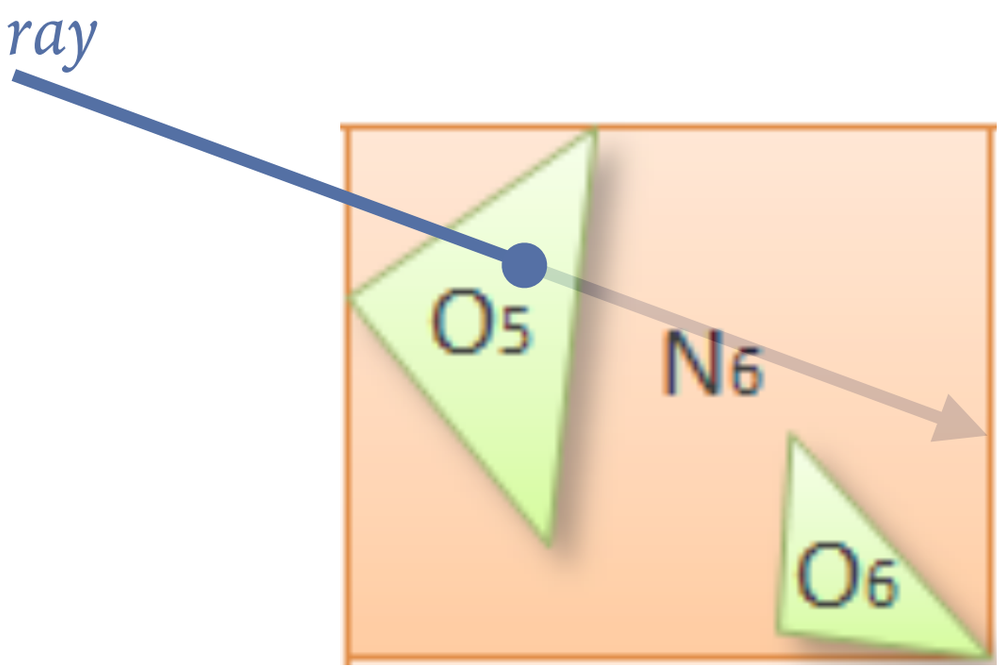




INTERSECTION: HOW DOES IT WORK?

- Once we've traversed the whole tree (current node is thus a leaf) we have to check to see if we intersect with the final bounding box.
- If we do, we now must check if our ray intersects with one of the primitives inside our leaf.
- If it does => return data about the mesh our ray intersected.





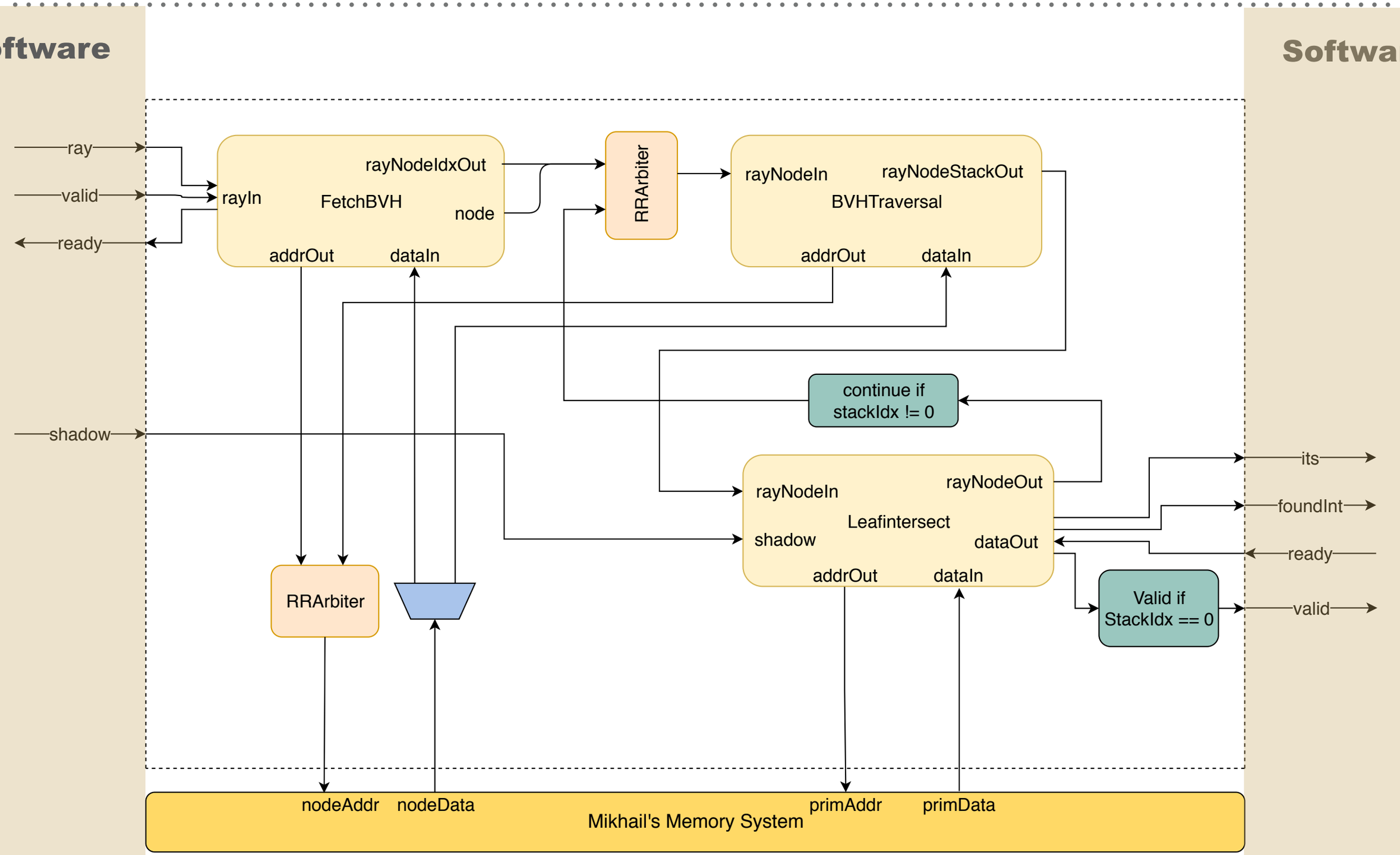
HARDWARE IMPLEMENTATION

HARDWARE: THE BOARD

- Board used for the project: Xilinx ZC706
  - Two Cortex-A9 ARM processors
  - Artix-7 series FPGA

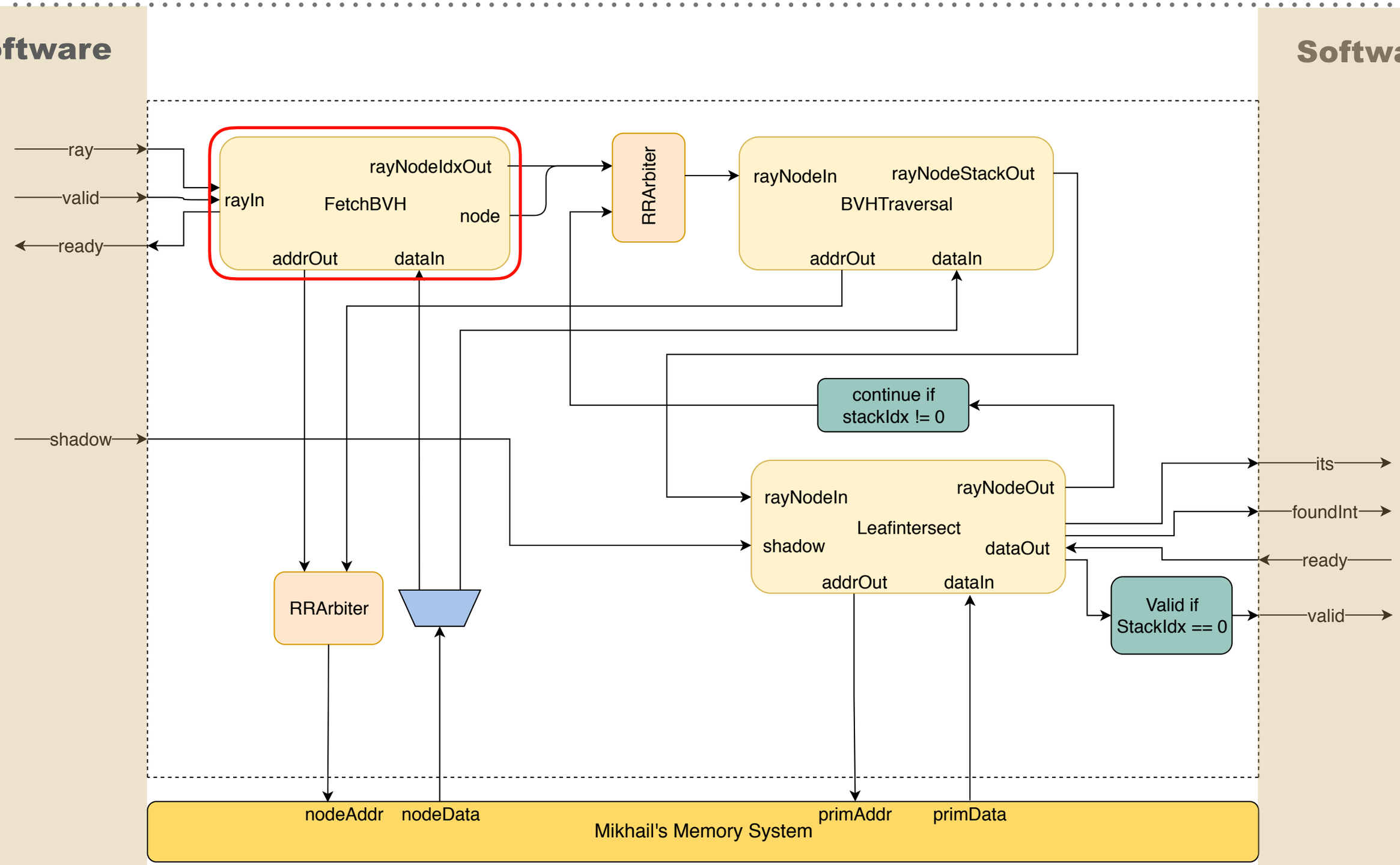
- Allows us to run our entire system (software + hardware) on the board:
  - Software renderer runs on the arm processors
  - Accelerator runs on the FPGA

HARDWARE: ACCELERATOR TOP-LEVEL



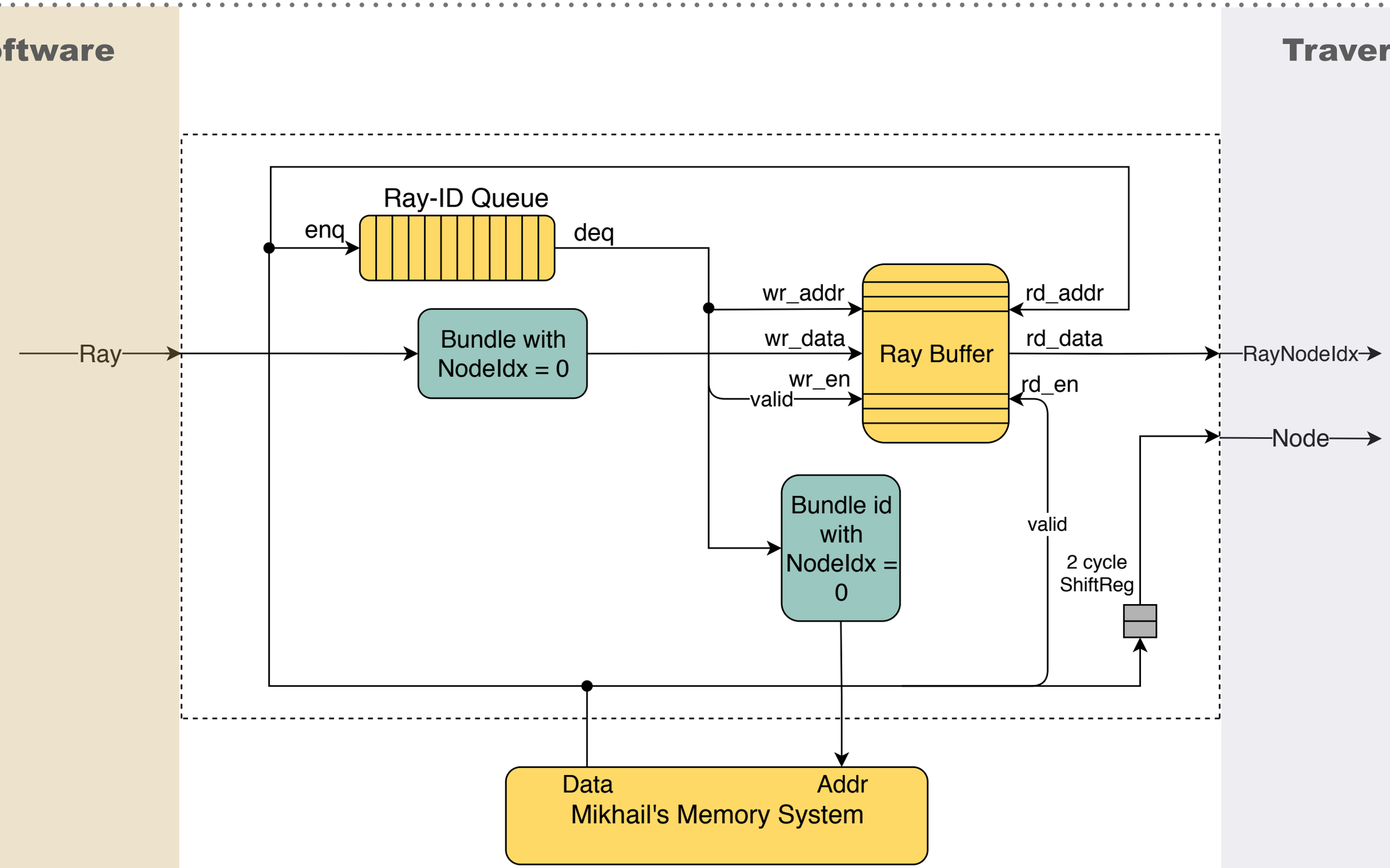
External module

HARDWARE: ACCELERATOR TOP-LEVEL



External module

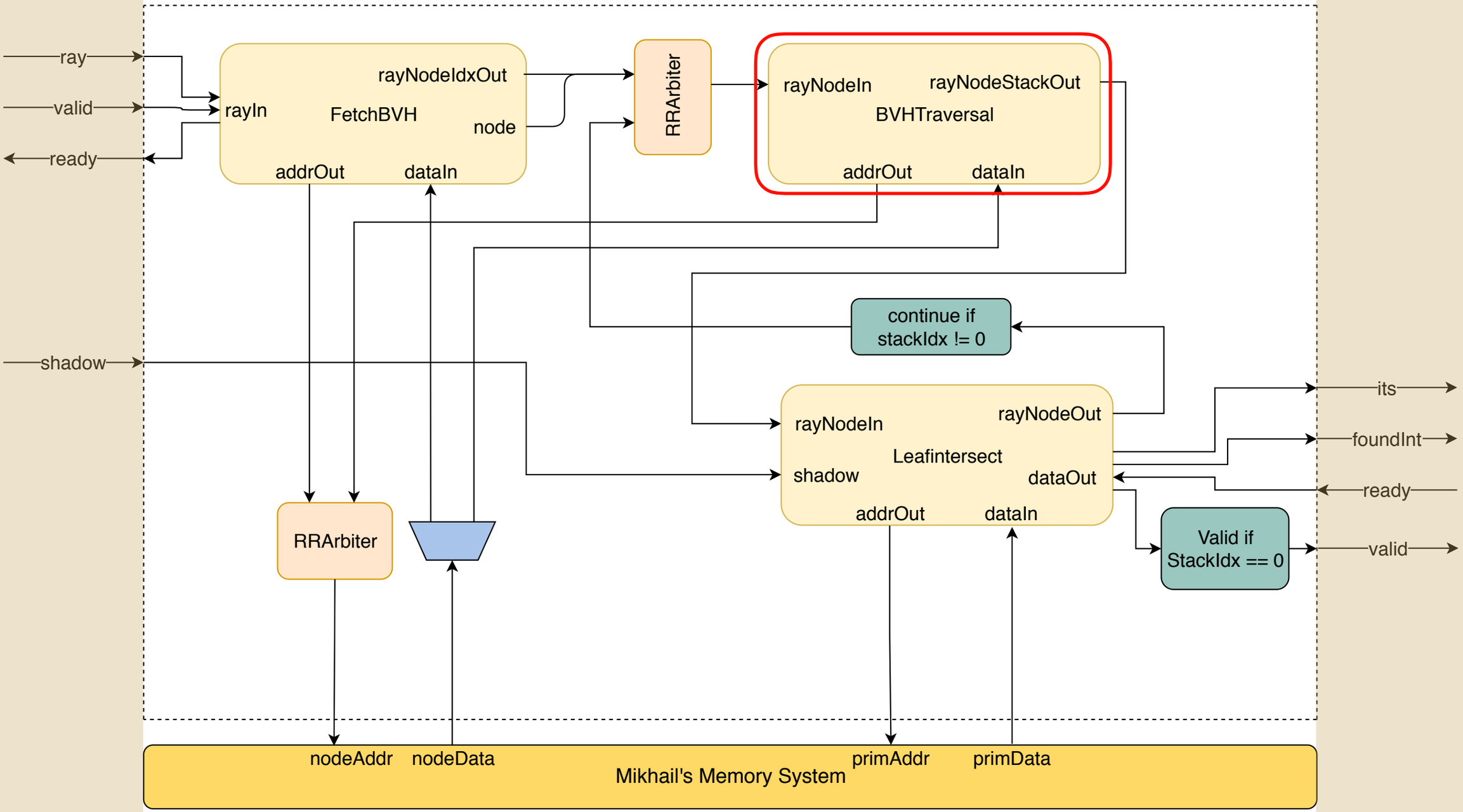
HARDWARE: FETCH BVH



HARDWARE: ACCELERATOR TOP-LEVEL

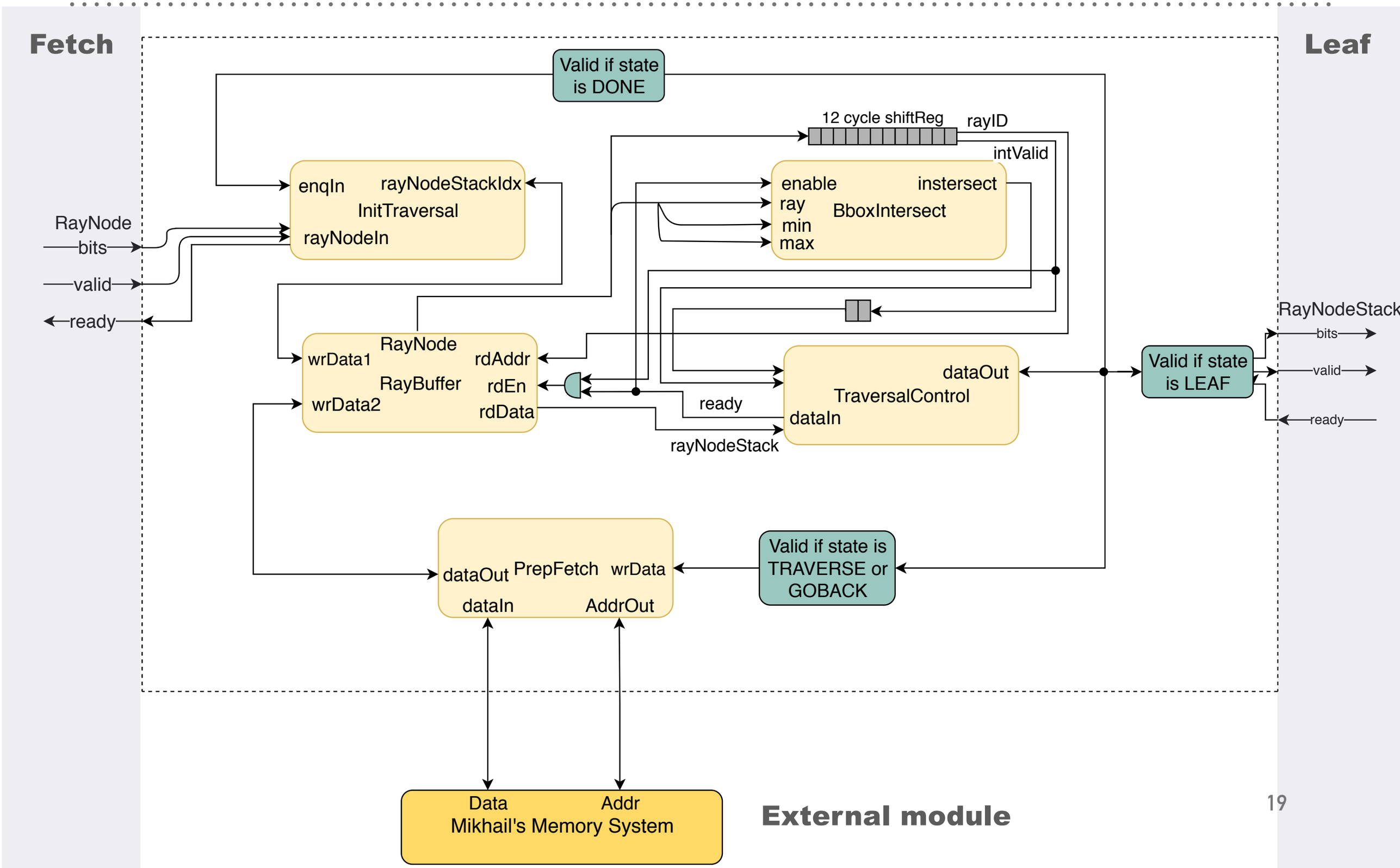
Software

Software

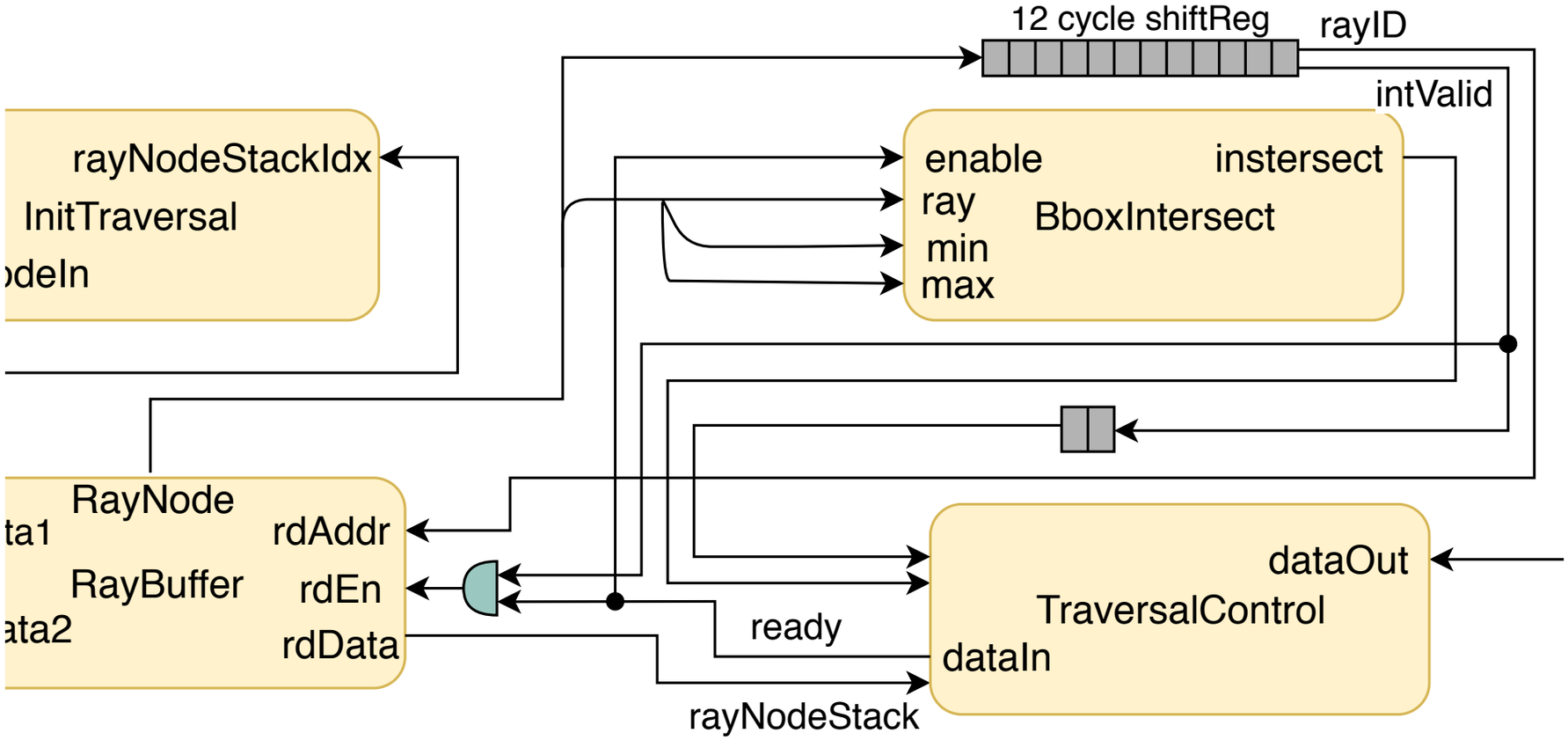


External module

HARDWARE: BVH TRAVERSAL

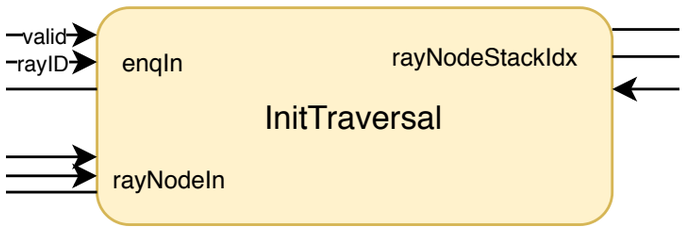


HARDWARE: BVH TRAVERSAL



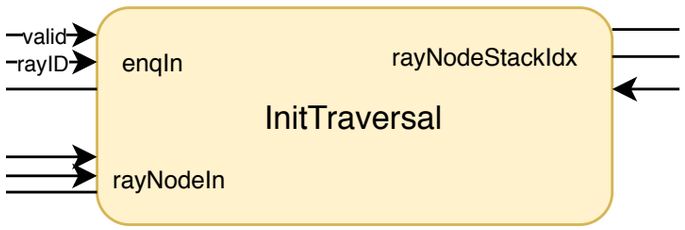
Idea: Prefetch the ray from the buffer in order for it to be ready at the same time as the associated result from the BboxIntersect module

HARDWARE: BVH TRAVERSAL SUB-MODULES

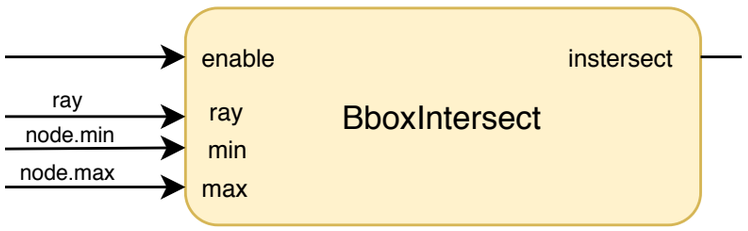


Similar to the first part of FetchBVH: Given a ray and a node, associate said ray to a unique ID. These IDs can be recycled via the enqIn interface.

HARDWARE: BVH TRAVERSAL SUB-MODULES

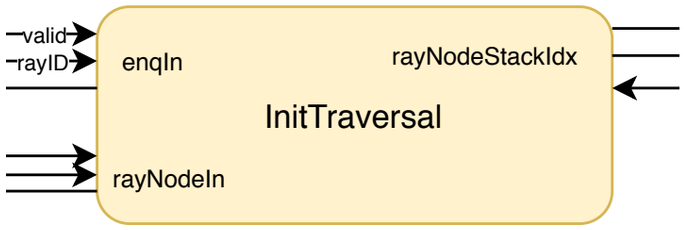


Similar to the first part of FetchBVH: Given a ray and a node, associate said ray to a unique ID. These IDs can be recycled via the enqIn interface.

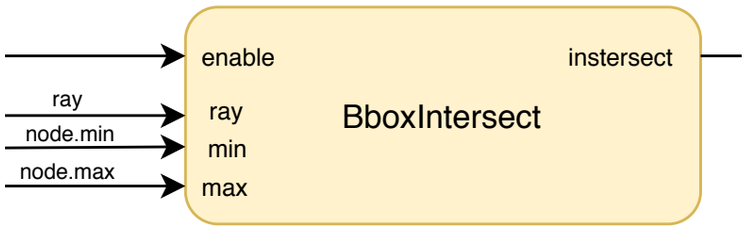


Module synthesised using HLS, from the software function's modified C++ code (see slide 24).

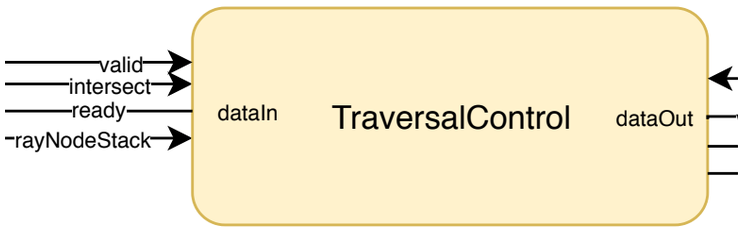
HARDWARE: BVH TRAVERSAL SUB-MODULES



Similar to the first part of FetchBVH: Given a ray and a node, associate said ray to a unique ID. These IDs can be recycled via the enqIn interface.

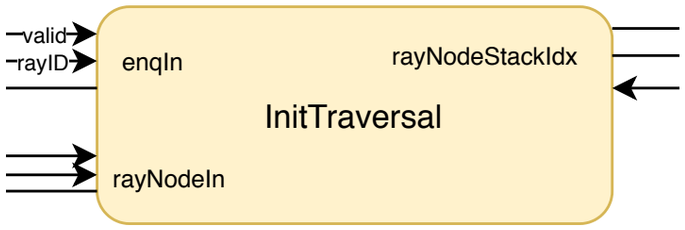


Module synthesised using HLS, from the software function's modified C++ code (see slide 24).

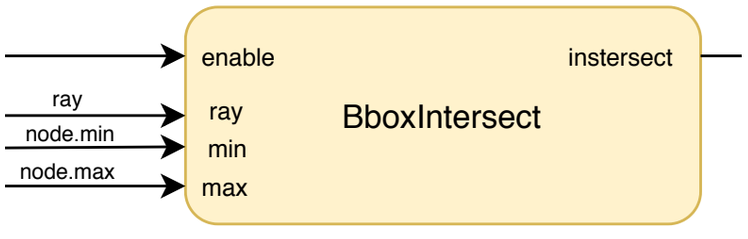


Assigns a state to each ray depending on the given node, node index and stack index. The state is then output and used to decide wether to continue the traversal down the BVH, continue back up the BVH (DFS), end the traversal with no output or end it with a leaf as output

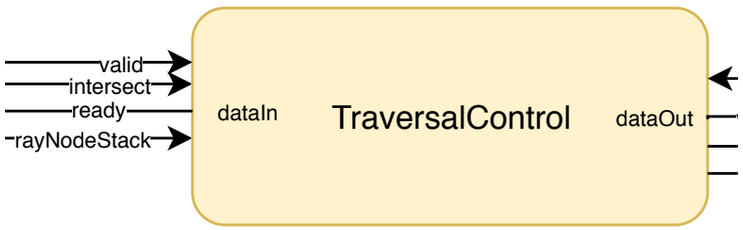
HARDWARE: BVH TRAVERSAL SUB-MODULES



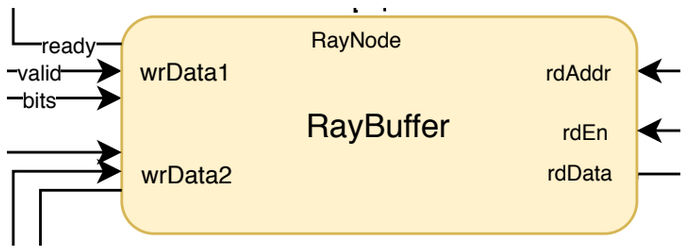
Similar to the first part of FetchBVH: Given a ray and a node, associate said ray to a unique ID. These IDs can be recycled via the enqIn interface.



Module synthesised using HLS, from the software function's modified C++ code (see slide 24).

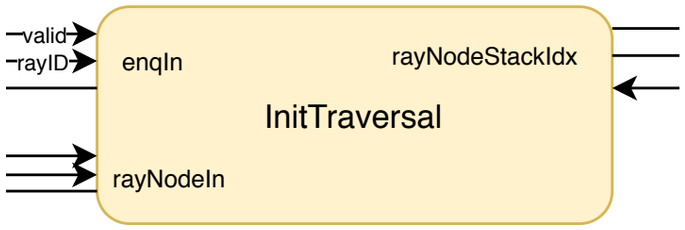


Assigns a state to each ray depending on the given node, node index and stack index. The state is then output and used to decide wether to continue the traversal down the BVH, continue back up the BVH (DFS), end the traversal with no output or end it with a leaf as output

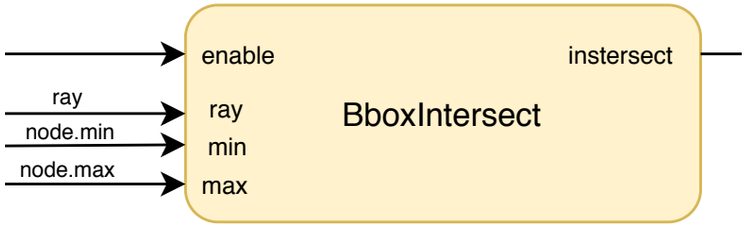


Stores the Ray-Node data while waiting for the results from the BboxIntersect sub-module (which only outputs a boolean value).

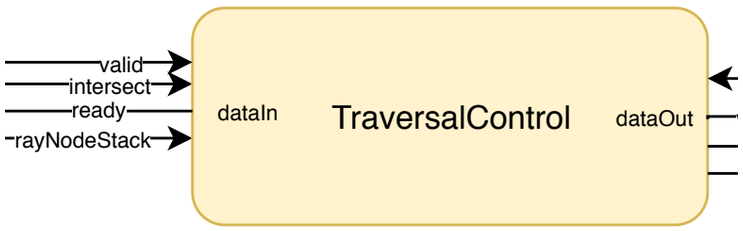
HARDWARE: BVH TRAVERSAL SUB-MODULES



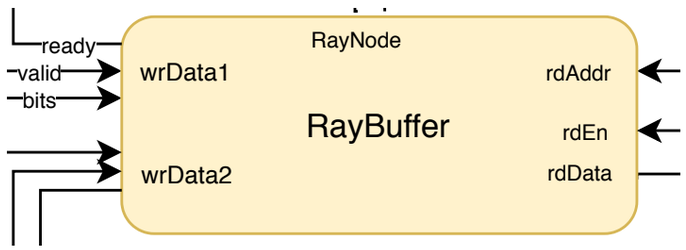
Similar to the first part of FetchBVH: Given a ray and a node, associate said ray to a unique ID. These IDs can be recycled via the enqIn interface.



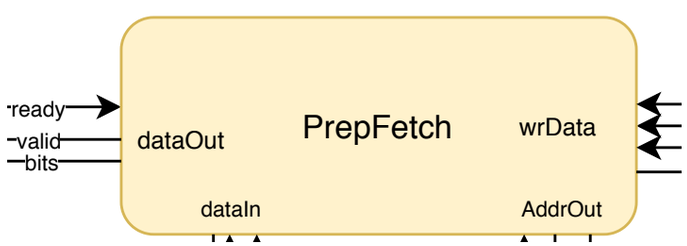
Module synthesised using HLS, from the software function's modified C++ code (see slide 24).



Assigns a state to each ray depending on the given node, node index and stack index. The state is then output and used to decide wether to continue the traversal down the BVH, continue back up the BVH (DFS), end the traversal with no output or end it with a leaf as output



Stores the Ray-Node data while waiting for the results from the BboxIntersect sub-module (which only outputs a boolean value).

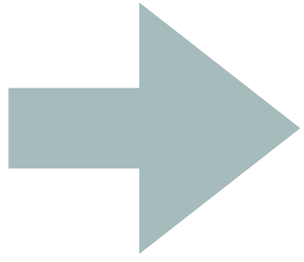


Prepares the interface with Mikhail's Memory system by storing rays and synchronising the output with the buffered data.

HARDWARE: BBOX-INTERSECT SUB-MODULE

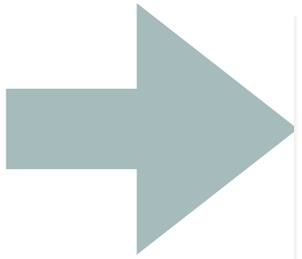
- Main element: Bbox-RayIntersect => done with HLS

```
/// Check if a ray intersects a bounding box
bool rayIntersect(const Ray3f &ray) const {
```



```
typedef struct {
    float o[3];
    float d[3];
    float dRcp[3];
    float mint;
    float maxt;
} Ray3f;
```

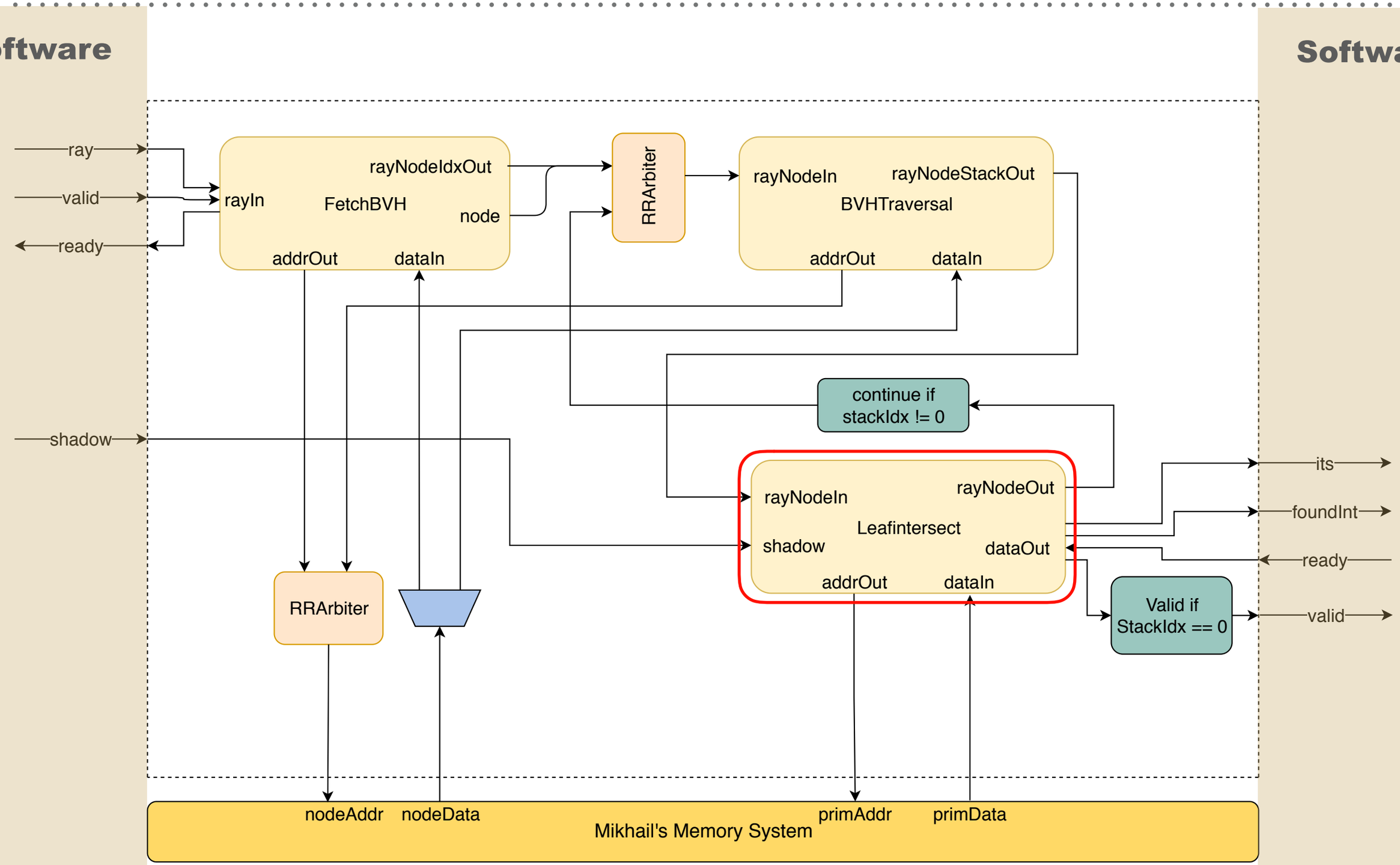
```
for (int i=0; i<3; i++) {
    float origin = ray.o[i];
    float minVal = min[i], maxVal = max[i];
```



```
    for (int i=0; i<3; i++) {
#pragma HLS unroll
        float origin = ray.o[i];
        float minVal = min[i], maxVal = max[i];
```

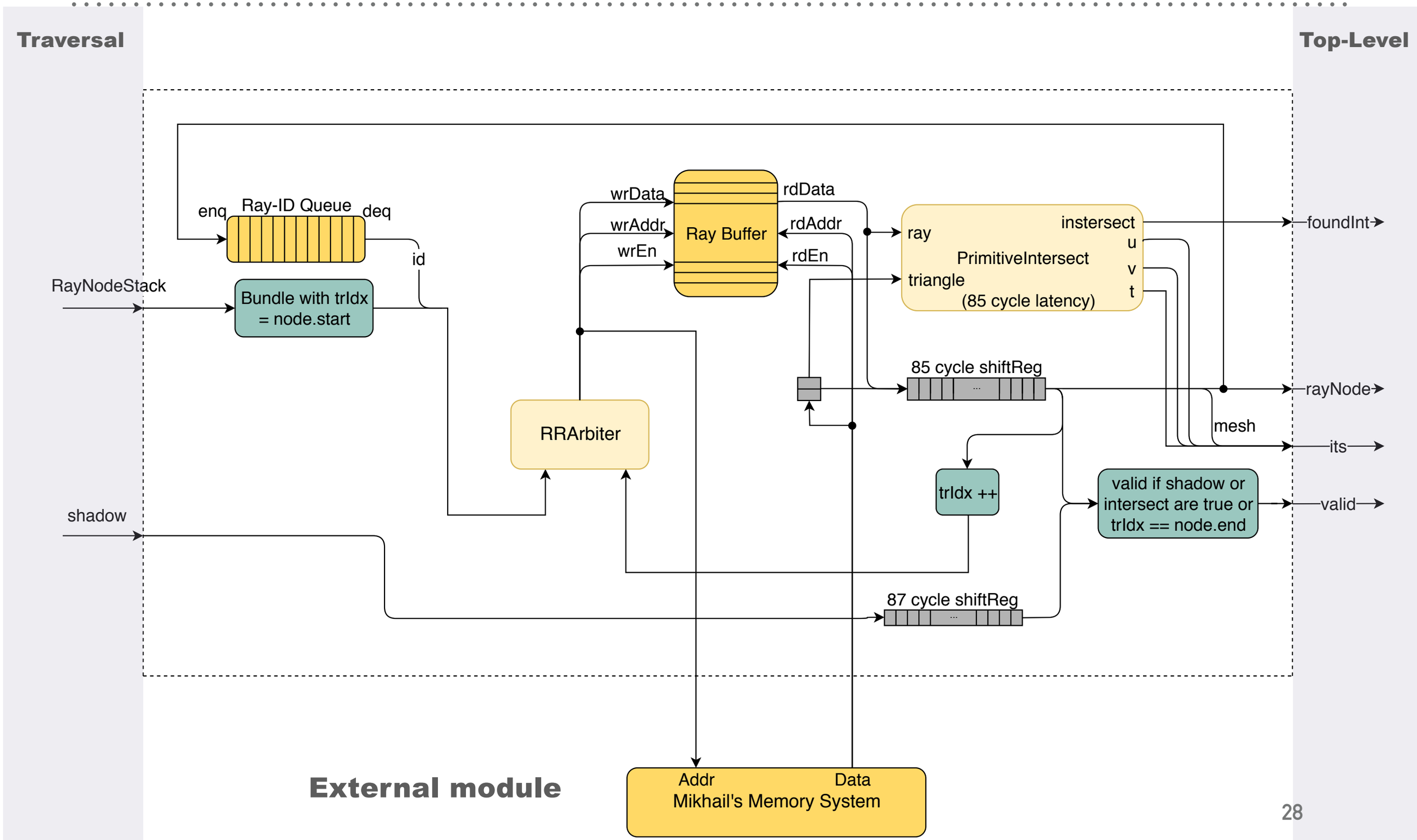
- We also have to flatten all of the arrays and enable pipelining

HARDWARE: ACCELERATOR TOP-LEVEL



External module

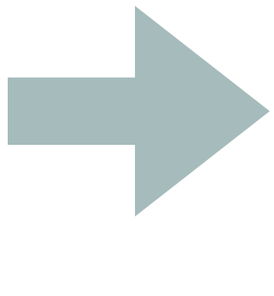
HARDWARE: LEAF INTERSECTION



HARDWARE: PRIMITIVE INTERSECT SUB-MODULE

- Main element: Primitive-RayIntersect => done with HLS.

```
, const Ray3f &ray,
struct myTriangle {
    Point3f p0, p1, p2;
    float idx;
    Mesh *mesh;
};
```



```
typedef struct {
    float o[3];
    float d[3];
    float dRcp[3];
    float mint;
    float maxt;
} Ray3f;
```

```
typedef struct {
    float p0[3];
    float p1[3];
    float p2[3];
} myTriangle;
```

- We have to flatten all of the arrays and enable pipelining.

RESULTS: HLS BLOCK SUMMARY

BoundingBox-RayIntersection

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
14	14	1	1	function

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1812	-
FIFO	-	-	-	-	-
Instance	-	30	3516	3498	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	2519	544	-
Total	0	30	6035	5854	0
Available	2060	2800	607200	303600	0
Utilization (%)	0	1	~0	1	0

Primitive-RayIntersection

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
85	85	1	1	function

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	249	-
FIFO	-	-	-	-	-
Instance	-	129	9885	10191	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	0	-	4790	1794	-
Total	0	129	14675	12270	0
Available	2060	2800	607200	303600	0
Utilization (%)	0	4	2	4	0

PROJECT ACHIEVEMENTS

- All modules have been implemented and are synthesisable

- Traversal and top-level have yet to be verified (mostly due to difficulties finding useful test data), test benches have been written

- Fetch & LeafIntersect have been verified in simulation using data extracted from the nori rayTracer

- FPGA environment has been setup (interface is ready, but verification needs to be finished)

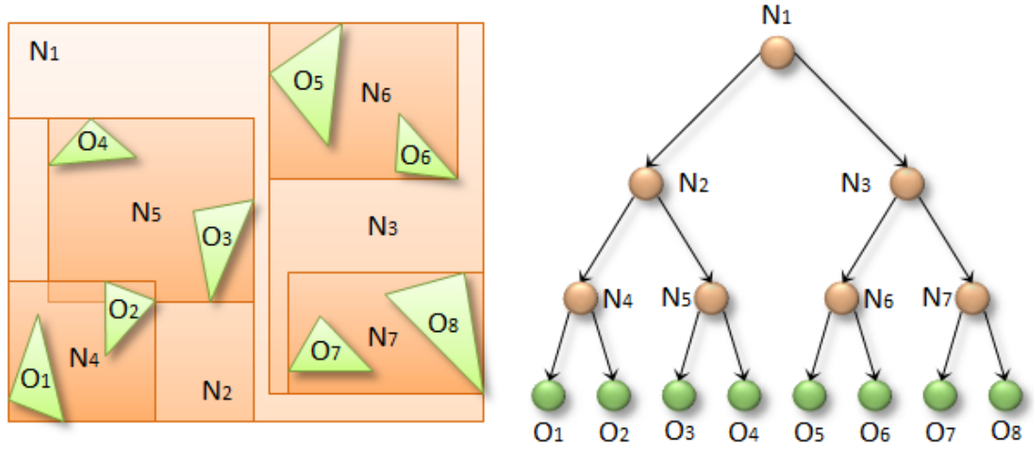
FUTURE WORK

FUTURE WORK

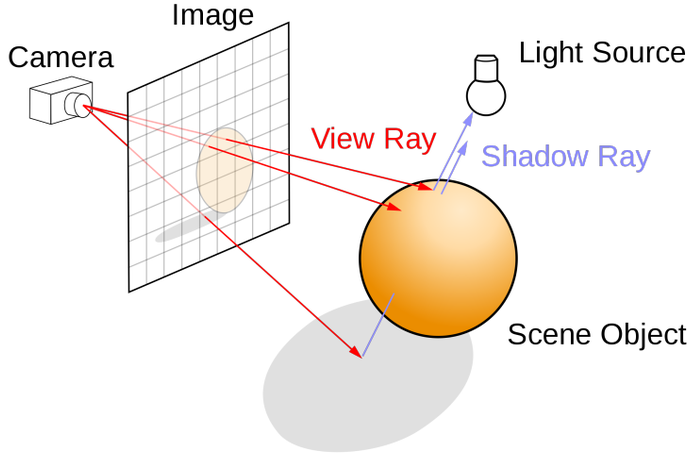
- Fetch BVH is sub-optimal:
  - Since root is the only fetched node, store it in a single line cache for it to be reused, rather than fetching it every time.
- Get the Accelerator running along-side the software on the Xilinx ZC706 FPGA.
- On the FPGA, run the Accelerator with and without using Mikhail's Memory System to check if there is indeed a performance boost.
- Adapt the solution so that it can be run in a more realistic environment, such as an Amazon F1 instance which use much larger FPGAs paired with Xeon processors.

FIGURE REFERENCES

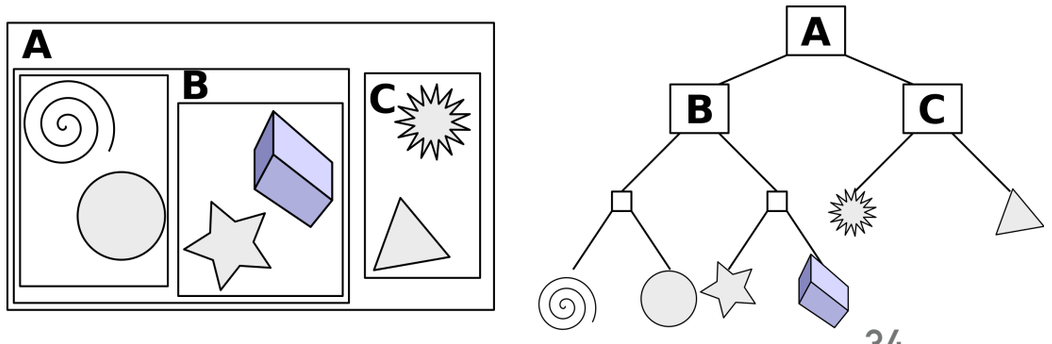
https://devblogs.nvidia.com/wp-content/uploads/2012/11/fig03-bvh.png



https://upload.wikimedia.org/wikipedia/commons/thumb/8/83/Ray_trace_diagram.svg/2560px-Ray_trace_diagram.svg.png



https://upload.wikimedia.org/wikipedia/commons/thumb/2/2a/Example_of_bounding_volume_hierarchy.svg/2880px-Example_of_bounding_volume_hierarchy.svg.png



QUESTIONS?