Writing a Formal Execution and Memory Model for Execution Synchronization Primitives on AMD GPUs

Pierre van Houtryve

AMD

together we advance_

A Simple Single-Threaded Algorithm

```
void reduce_sum(unsigned* data, int N)
{
    for (int stride = N/2; stride > 0; stride /= 2)
    {
        for (int i = 0; i < stride; i++)
        {
            data[i] += data[i + stride];
        }
    }
}
```

Single-thread execution is sequential.

Parallelism on the GPU with HIP

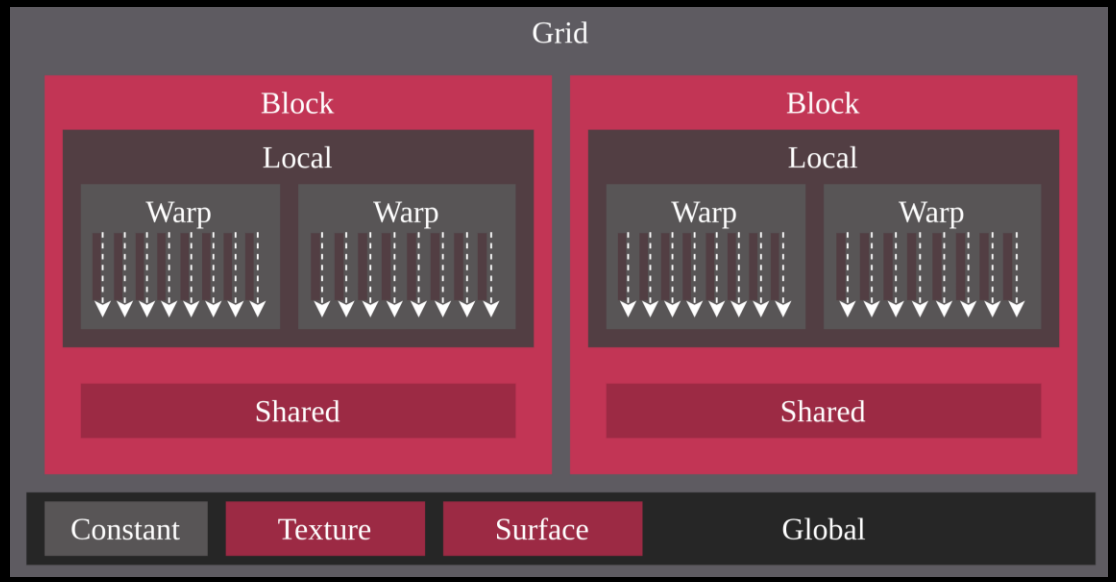
```
__global__ void reduce_sum(unsigned* data, int N)
{
    for (int stride = N/2; stride > 0; stride /= 2)
    {
        int i = threadIdx.x;
        if (i < stride)
            data[i] += data[i + stride];

        __syncthreads();// Barrier to synchronize execution of threads.
    }
}
```

In a parallel program, threads can progress independently.

Execution Synchronization Beyond "syncthreads"

- "syncthreads" synchronizes memory & execution at a fixed scope.
- Other APIs can synchronize:
  - Threads at different scopes.
  - Only specific address spaces.
- Both parameters affect performance.



AMDGPU Memory Hierarchy.

The AMDGPU Target & Execution Synchronization

- AMDGPU barrier primitives must be:
  - Reusable.
  - Portable.
  - Efficient.
- Documentation is hard.
- Imprecise documentation leads to:
  - Ad-hoc changes.
  - Confusing user experience.

Documenting Barriers in AMDGPU

- Fences must pair with an atomic memory operation.
- Barriers are not memory operations, yet this must work:

```
fence syncscope("workgroup") release
call void @llvm.amdgcn.s.barrier()
fence syncscope("workgroup") acquire
```

LLVM IR for "`__syncthreads()`".

The Formal Barrier Execution Model

- Formally document:
  - Barrier operations and how to use them.
  - Guarantees offered by barriers.

Challenge: Compatibility between Targets

- Need backward and forward compatibility.
- (Try to) support alternative implementations.
- **Solution:** Separate model from implementations.

Challenge: Memory Model Integration

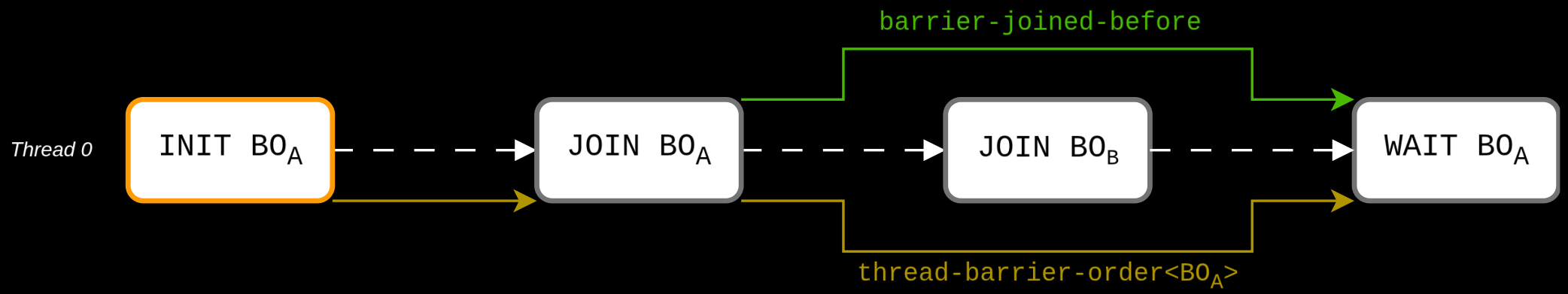
- Memory models are:
  - Precise.
  - Defined over program executions, not source code.
- Memory model does not apply to barriers.
  - Execution model must be defined separately.

Challenge: Mastering the Chaos

- Experimental work.
- Remember the big picture.
- **Solutions:**
  - Iterative development.
  - Frequent feedback.

Modeling the Barrier Flow

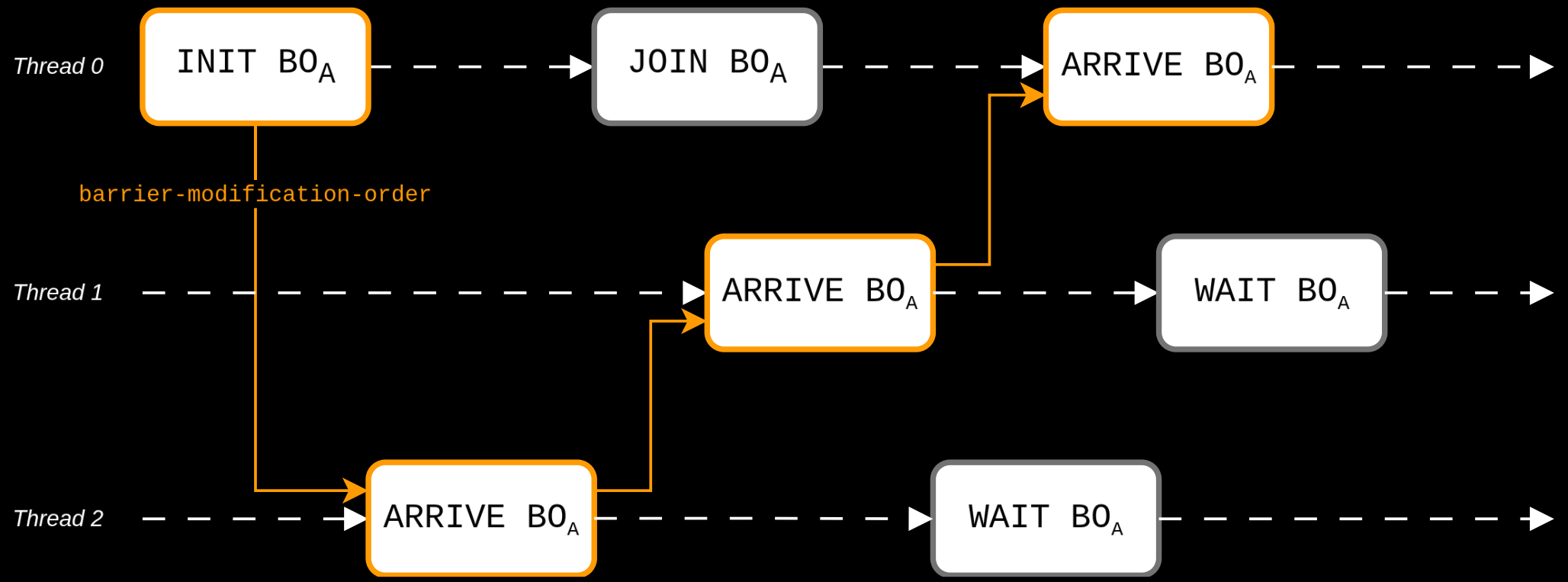
- Barrier *objects* (BOs).
- Barrier *operations* on barrier *objects*:
  - *join, wait, init, arrive, drop*
    modification operations
- Basic single-thread relations:
  - *thread-barrier-order<BO>*
  - *barrier-joined-before*



Note: *thread-barrier-order<BO>* is transitive, but only the transitive reduction is drawn on the diagram.

Modeling the Barrier Flow: Modification Order

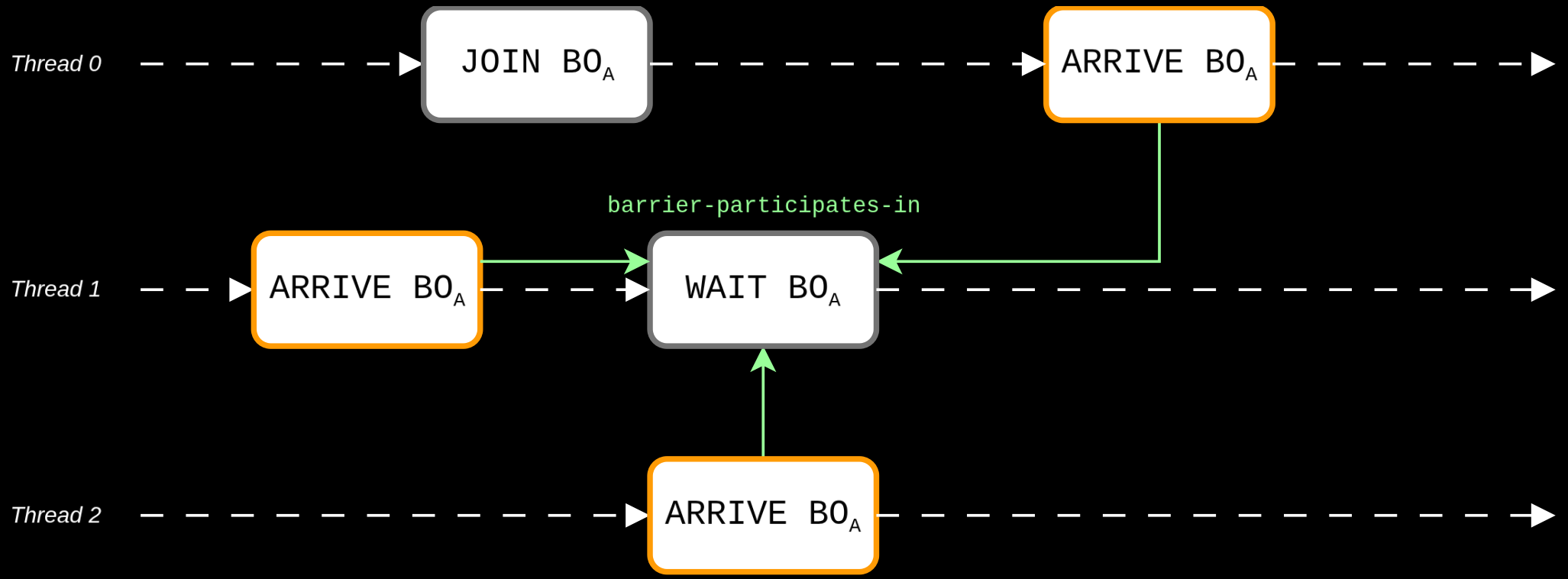
- *barrier-modification-order<BO>*



Note: This relations is transitive, but only the transitive reduction is drawn on the diagram.

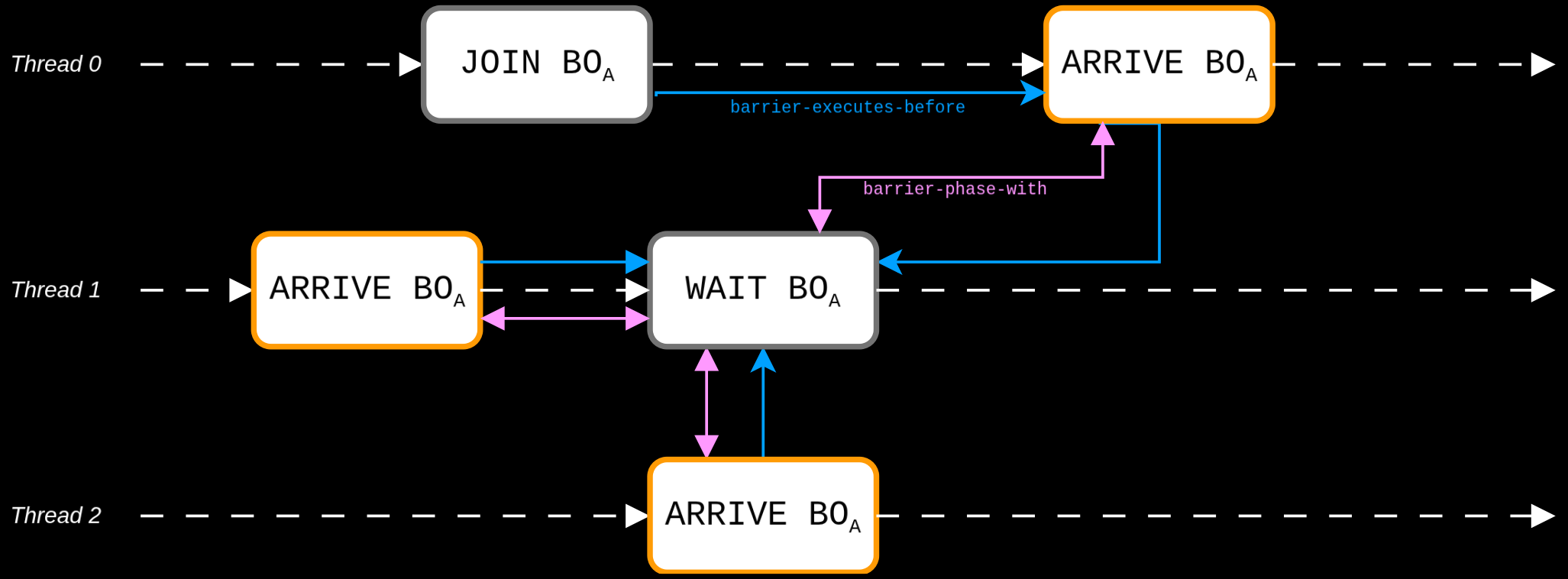
Modeling Execution Dependencies

- Barrier completion represented by *barrier-participates-in*



Modeling Execution Dependencies: Emergent Relations

- *barrier-executes-before* = *barrier-participates-in* + *program-order* of barrier operations
- *barrier-phase-with: b*arrier phases via *barrier-participates-in*



Note: both relations are transitive, but only the transitive reduction is drawn on the diagram.

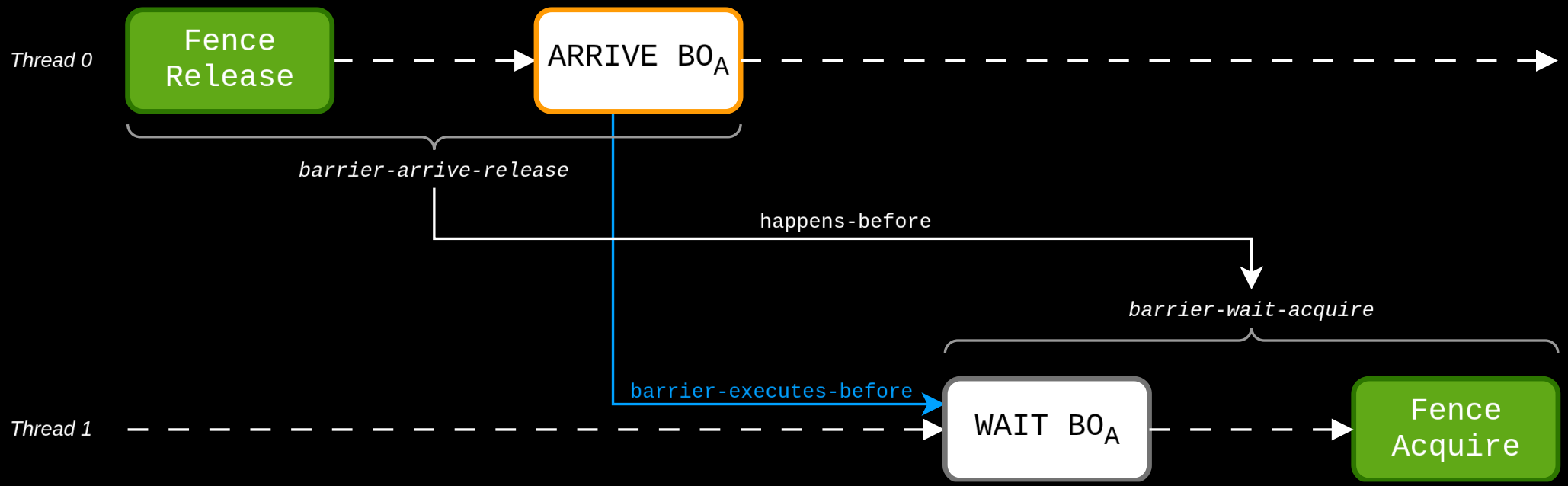
Applying the Execution Model to AMDGPU Targets

- Formal model has common semantics.
- Target-specific semantics specified separately.
- Informational notes guide users.
- Intrinsics rely on barrier operations.
- Barrier operations implementation documented for all targets.

Documentation Table of Content.

Synchronizing Memory

- *barrier-executes-before* alone does not affect memory.
- Use fences with barrier operations to synchronize memory:



Testing

- Barrier execution model implemented in Alloy (https://alloytools.org/).
- Test cases written in a custom format and translated to Alloy.
- Determines whether a given program execution is allowed by the model or not.

```
#push expected_count[A, 2] && only_arrive_participates

#thread 0
  init A
  join A
  arrive A
  drop A

#thread 1
  join A
  arrive A
  wait A
  drop A

// Cannot arrive then drop, it could corrupt the barrier.
#check unsat
```

Test File Example.

Conclusion & What's Next ?

- Find the text at https://llvm.org/docs/AMDGPUUsage.html#execution-barriers
- Keep improving the model.
- Continue supporting new targets as they come.

Copyright and disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION


AMD