



RUTGERS

UNIVERSITY | NEW BRUNSWICK



Generic System Calls for GPUs

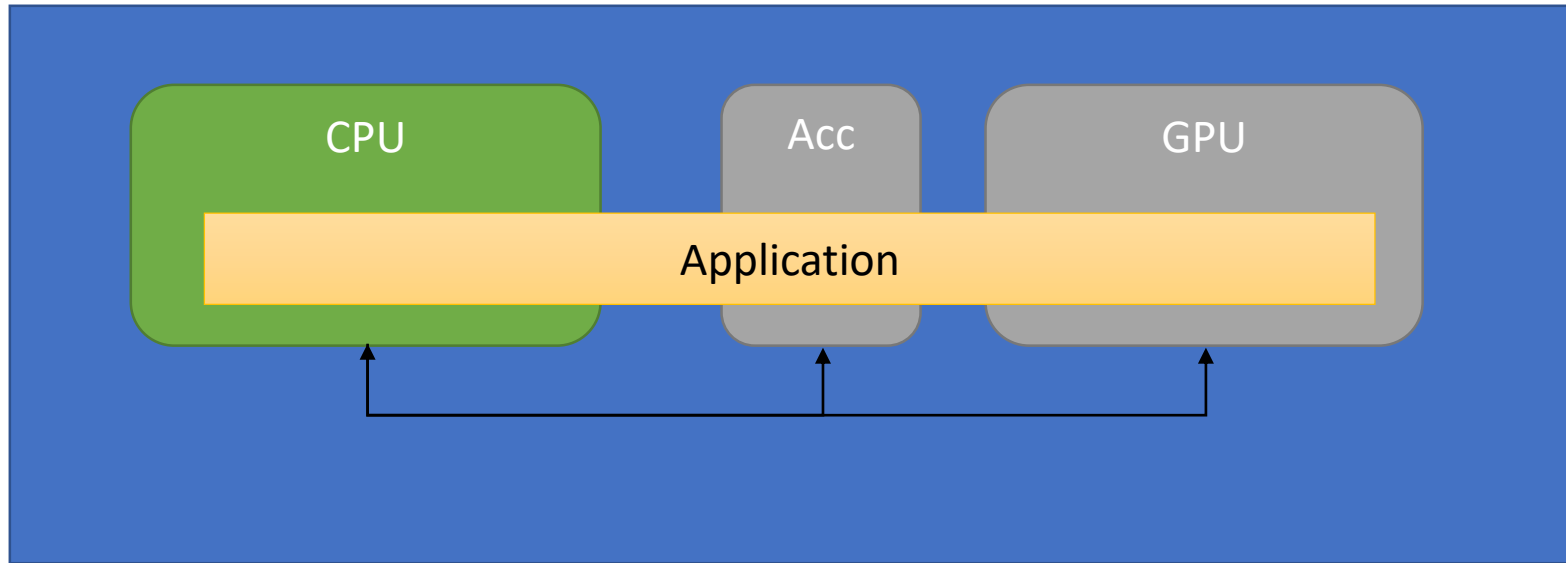
Ján Veselý*, Arkaprava Basu[†], Abhishek Bhattacharjee*,
Gabriel H. Loh[‡], Mark Oskin^{§‡}, Steven K. Reinhardt[¶]

*Rutgers University, [†]Indian Institute of Science,

[‡]Advanced Micro Devices Inc.,

[§]University of Washington, [¶]Microsoft Inc.

Towards heterogeneous computing



Programs require system services

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc();

    data = recvmsg(port);

    idx = get_idx(data);

    response_data = pread(file, idx);

    response = process(response_data, data);

    log("request processed\n");

    sendmsg(port, response);

    free(data, response_data, response);
}
```

Programs require system services

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc(); → Memory allocation

    data = recvmsg(port);

    idx = get_idx(data);

    response_data = pread(file, idx);

    response = process(response_data, data);

    log("request processed\n");

    sendmsg(port, response);

    free(data, response_data, response); → Memory allocation
}
```

Programs require system services

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc(); → Memory allocation
    data = recvmsg(port); → Network
    idx = get_idx(data);
    response_data = pread(file, idx);
    response = process(response_data, data);
    log("request processed\n");
    sendmsg(port, response); → Network
    free(data, response_data, response); → Memory allocation
}
```

Programs require system services

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc(); → Memory allocation
    data = recvmsg(port); → Network
    idx = get_idx(data);
    response_data = pread(file, idx); → Storage
    response = process(response_data, data);
    log("request processed\n");
    sendmsg(port, response); → Network
    free(data, response_data, response); → Memory allocation
}
```

Programs require system services

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc(); → Memory allocation
    data = recvmsg(port); → Network
    idx = get_idx(data);
    response_data = pread(file, idx); → Storage
    response = process(response_data, data);
    log("request processed\n"); → Terminal/Storage
    sendmsg(port, response); → Network
    free(data, response_data, response); → Memory allocation
}
```

Programs require system services

CPU:

```
function process(port, file)
{
```

```
    data, response_data, response = malloc();
```

→ Memory allocation

```
    data = recvmsg(port);
```

→ Network

```
    idx = get_idx(data);
```

```
    response_data = pread(file, idx);
```

→ Storage

```
    response = process(response_data, data);
```

```
    log("request processed\n");
```

→ Terminal/Storage

```
    sendmsg(port, response);
```

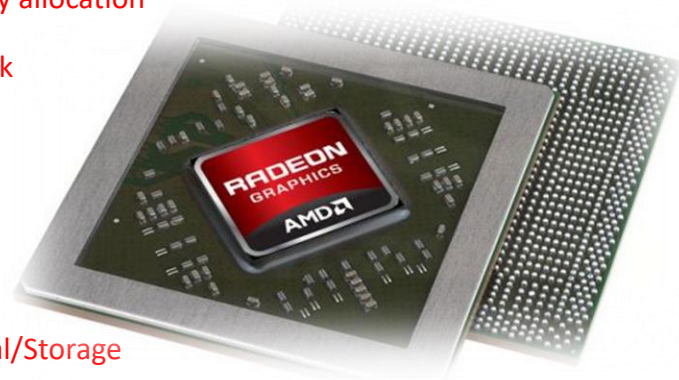
→ Network

```
    free(data, response_data, response);
```

→ Memory allocation

```
}
```

GPU:



Computation can be offloaded

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc();

    data[] = recvmsgs(port);

    copy_to_device(data[]);
    gpu_get_idx(&idx[], data[]);
    copy_from_device(idx[]);
    for (d in data)
        response_data[] = pread(file, idx[i]);
    copy_to_device(response_data[]);
    gpu_process(&response[], response_data[], data[]);
    copy_from_device(response[]);

    log("requests processed\n");

    sendmsgs(port, response[]);
    ...
}
```

GPU:

 GPU MANAGEMENT
 GPU INVOCATION
 GPU KERNEL

Kernel 1

```
idx[i] = get_idx(data[i])
```

Kernel 2

```
response[i] =
    process(response_data[i], data[i])
```

GPUs are tightly integrated

- Unified virtual memory (UVM)
 - HSA, CUDA® UVM, OpenCL™ SVM
- CPU↔GPU cache coherence
 - HSA, CCIX®, Gen-Z

UVM and cache coherence ease programmability

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc();

    data[] = recvmsgs(port);

    copy_to_device(data[]);
    gpu_get_idx(&idx[], data[]);
    copy_from_device(idx[]);
    for (d in data)
        response_data[] = pread(file, idx[i]);
    copy_to_device(response_data[]);
    gpu_process(&response[], response_data[], data[]);
    copy_from_device(response[]);

    log("requests processed\n");

    sendmsgs(port, response[]);
    ...
}
```

GPU:

 GPU MANAGEMENT
 GPU INVOCATION
 GPU KERNEL

Kernel 1

```
idx[i] = get_idx(data[i])
```

Kernel 2

```
response[i] =
    process(response_data[i], data[i])
```

UVM and cache coherence ease programmability

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc();

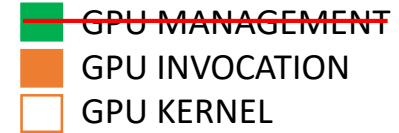
    data[] = recvmsgs(port);

copy_to_device(data[]);
gpu_get_idx(&idx[], data[]);
copy_from_device(idx[]);
    for (d in data)
        response_data[] = pread(file, idx[i]);
copy_to_device(response_data[]);
gpu_process(&response[], response_data[], data[]);
copy_from_device(response[]);

    log("requests processed\n");

    sendmsgs(port, response[]);
    ...
}
```

GPU:



Kernel 1

```
idx[i] = get_idx(data[i])
```

Kernel 2

```
response[i] =
    process(response_data[i], data[i])
```

UVM and cache coherence ease programmability

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc();

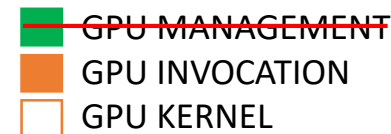
    data[] = recvmsgs(port);

copy_to_device(data[]);
gpu_get_idx(&idx[], data[]);
copy_from_device(idx[]);
    for (d in data)
        response_data[] = pread(file, idx[i]);
copy_to_device(response_data[]);
gpu_process(&response[], response_data[], data[]);
copy_from_device(response[]);

    log("requests processed\n");

    sendmsgs(port, response[]);
    ...
}
```

GPU:



Kernel 1

```
idx[i] = get_idx(data[i])
```

Kernel 2

```
response[i] =
    process(response_data[i], data[i])
```

Next step is system services

Next step is system services

- Memory allocation
 - HSA, CUDA®

Next step is system services

- Memory allocation
 - HSA, CUDA®
- Printf
 - HSA, OpenCL™, CUDA®

Next step is system services

- Memory allocation
 - HSA, CUDA®
- Printf
 - HSA, OpenCL™, CUDA®
- Academic research
 - GPUfs [Silberstein, ASPLOS'13], GPUnet [Kim, OSDI'14],
 - SPIN [Bergman, ATC'17], ...

Some services can be invoked from GPU

CPU:

```
function process(port, file)
{
    data, response_data, response = malloc();
    gpu_process(port, file, response[],
                response_data[], data[]);
    free(data, response_data, response);
}
```

GPU:

```
void gpu_group_process(port, file)
{
    data = Grecv(port);

    idx = gpu_get_idx(&idx, data);

    response_data = Gread(file, idx);

    response = process(response_data,
                       data);

    Gprintf("request processed\n");

    Gsend(port, response);
}
```

■ GPUnet
■ GPUfs
■ CUDA

Previous solutions took the first steps

Previous solutions took the first steps

- Subsystem specific

Previous solutions took the first steps

- Subsystem specific
- Specialized, restricted functionality

Previous solutions took the first steps

- Subsystem specific
- Specialized, restricted functionality
- Custom API/semantics

Our work takes the next step

Our work takes the next step

- **GENE**ric **SY**stem call interface
 - Efficient direct-to-OS communication

Our work takes the next step

- **GENE**ric **SY**stem call interface
 - Efficient direct-to-OS communication
- Allows all system calls implementable for GPUs
 - 79% of all system calls

Our work takes the next step

- **GENERIC SYStem** call interface
 - Efficient direct-to-OS communication
- Allows all system calls implementable for GPUs
 - 79% of all system calls
- Original OS (Linux) semantics
 - POSIX®-like

Our work takes the next step

- **GENERIC SYStem** call interface
 - Efficient direct-to-OS communication
- Allows all system calls implementable for GPUs
 - 79% of all system calls
- Original OS (Linux) semantics
 - POSIX®-like
- Available on github
 - https://github.com/RadeonOpenCompute/{ROCK,ROCT,HCC}_syscall

Genesys subsumes previous work (and more)

CPU:

```
function process(port, file)
{
    gpu_process(port, file, response[],
                response_data[], data[]);
}
```

GPU:

 GENESYS

```
void gpu_process(port, file)
{
    data, response_data, response = malloc();

    data = recvmsg(port);

    idx = get_idx(data);

    response_data = pread(file, idx);

    response = process(response_data, data);

    log("requests processed\n");

    sendmsg(port, response);

    free(data, response_data, response);
}
```

Ideal system services properties

- Familiarity
 - Known semantics



Ideal system services properties

- Familiarity
 - Known semantics
- Flexibility
 - Do not restrict programmers
- Adaptability
 - Adapt to workload needs



Flexibility in application interface

- Invocation granularity

Flexibility in application interface

- Invocation granularity
- Observed ordering

Flexibility in application interface

- Invocation granularity
- Observed ordering
- Blocking vs. Non-blocking

Flexibility: Any thread can invoke system call

- GPU execution hierarchy

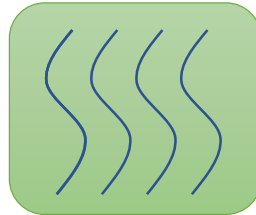
Flexibility: Any thread can invoke system call

- GPU execution hierarchy
 - Workitem (thread)



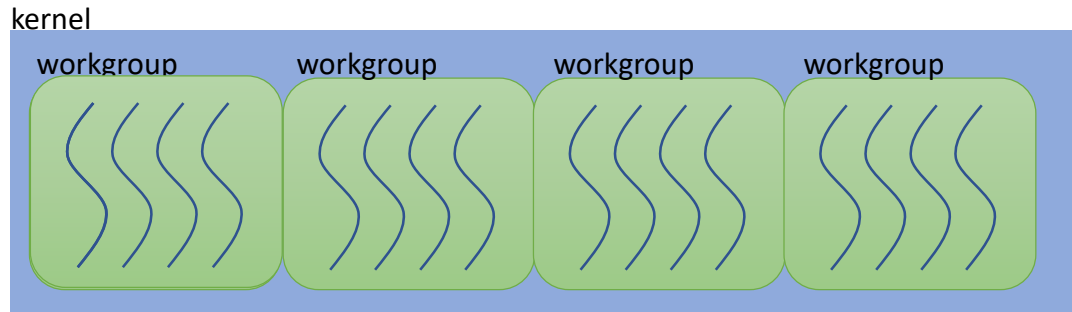
Flexibility: Any thread can invoke system call

- GPU execution hierarchy
 - Workitem (thread)
 - Workgroup (thread group)



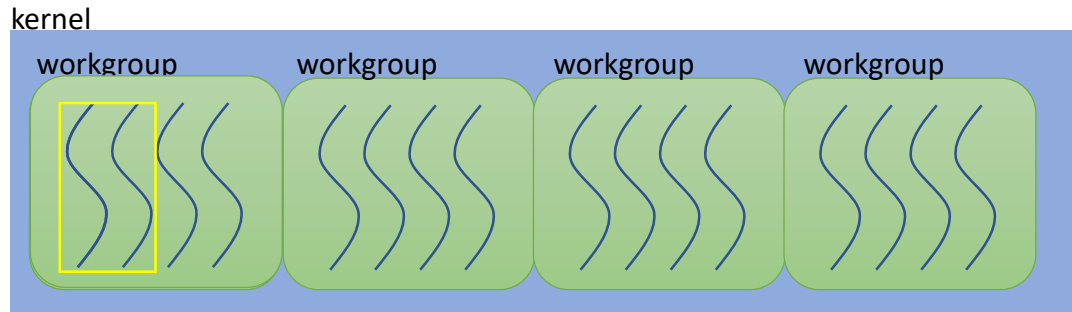
Flexibility: Any thread can invoke system call

- GPU execution hierarchy
 - Workitem (thread)
 - Workgroup (thread group)
 - Kernel



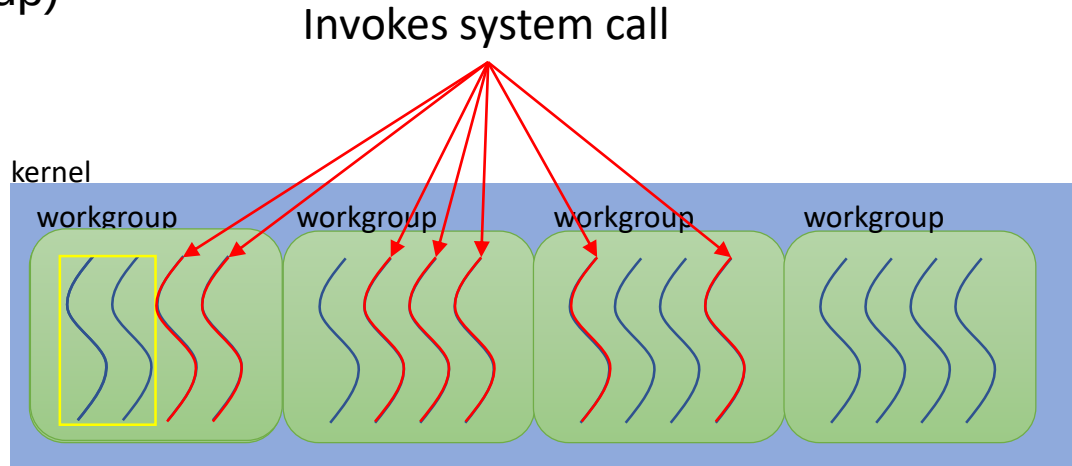
Flexibility: Any thread can invoke system call

- GPU execution hierarchy
 - Workitem (thread)
 - Workgroup (thread group)
 - Kernel
 - Wavefront (warp)
 - HW specific!
 - Do not expose!



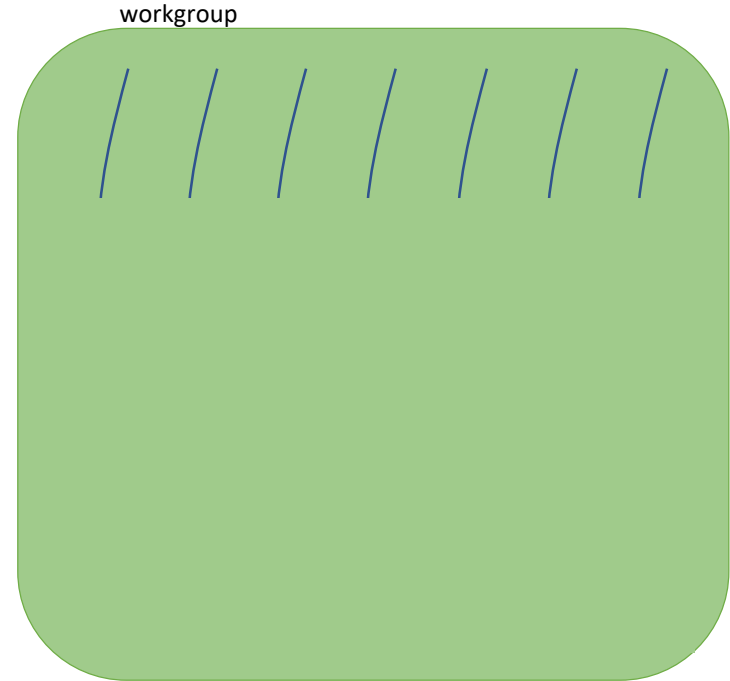
Flexibility: Any thread can invoke system call

- GPU execution hierarchy
 - Workitem (thread)
 - Workgroup (thread group)
 - Kernel
 - Wavefront (warp)
 - HW specific!
 - Do not expose!



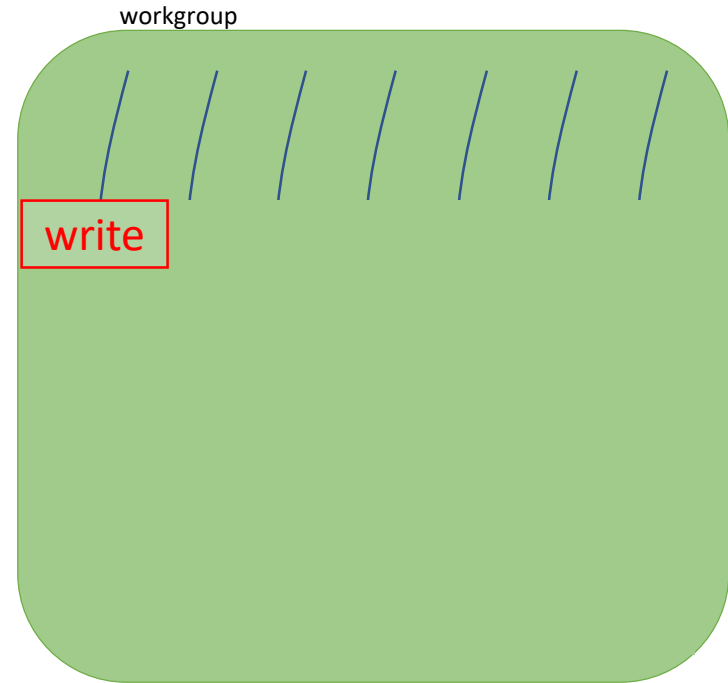
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



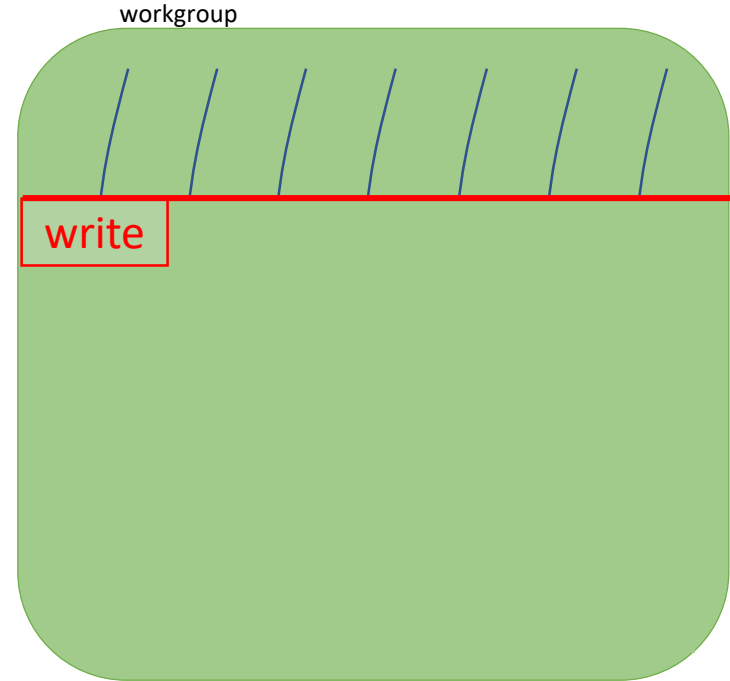
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



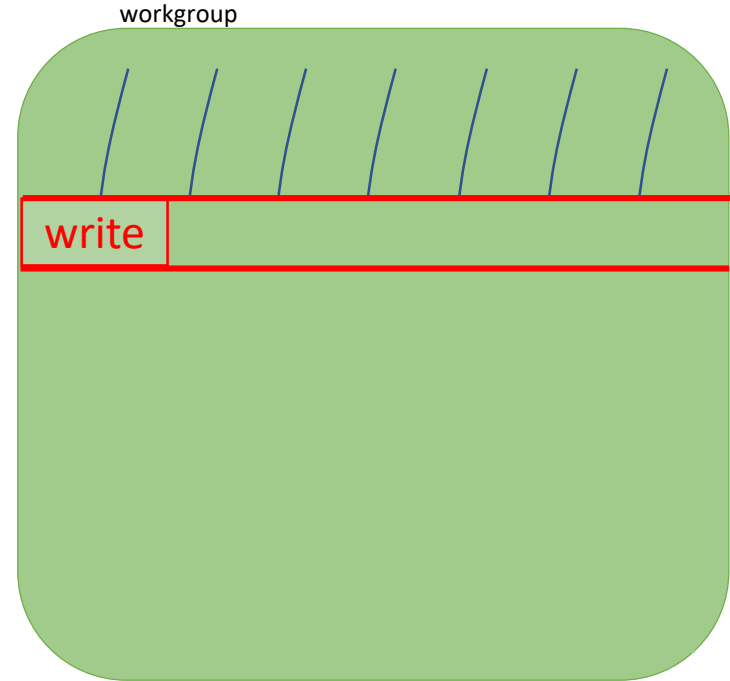
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



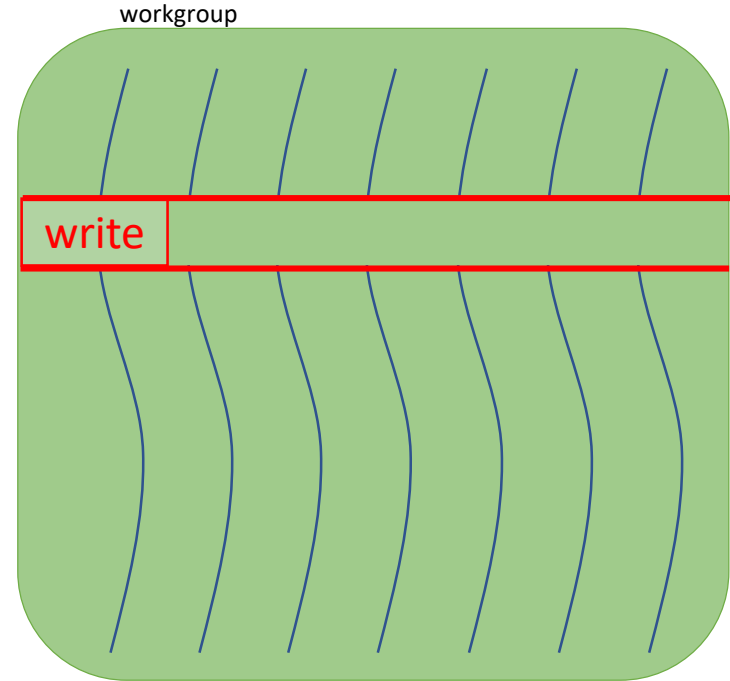
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



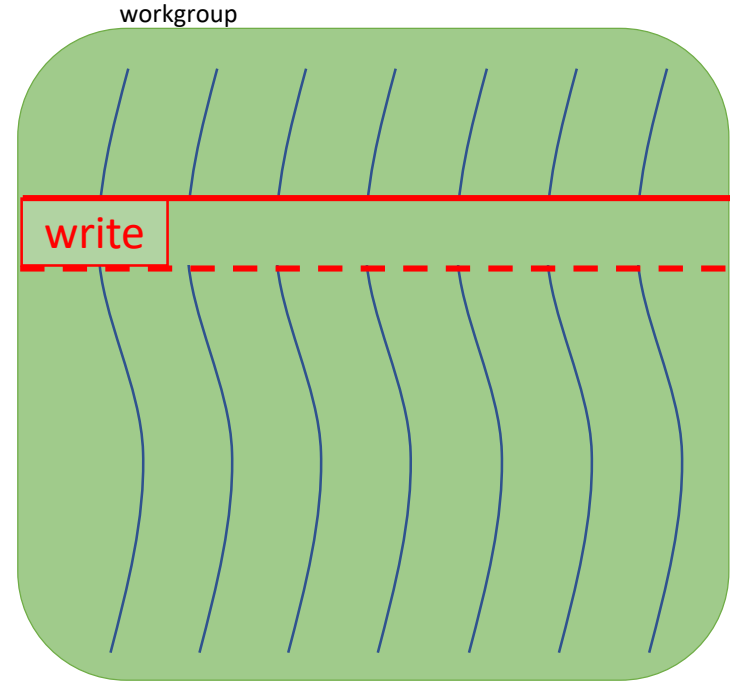
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



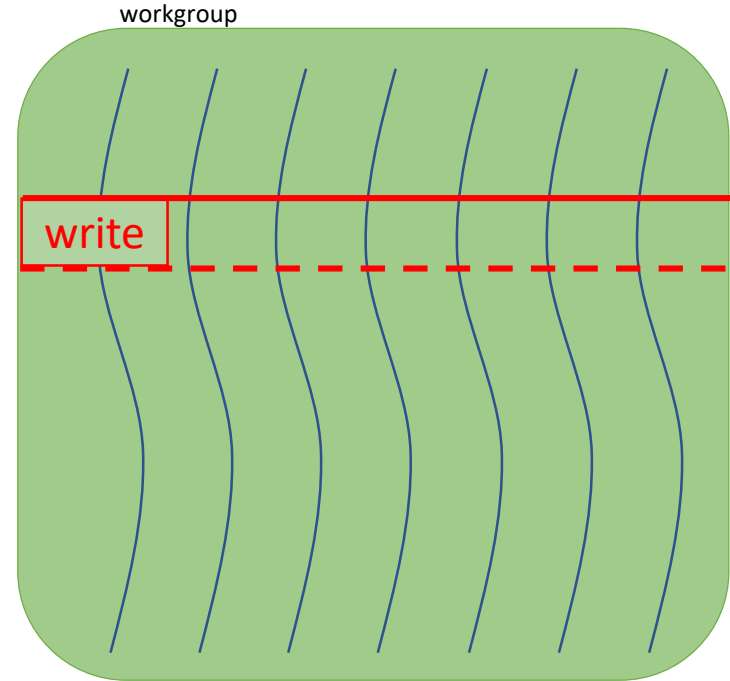
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



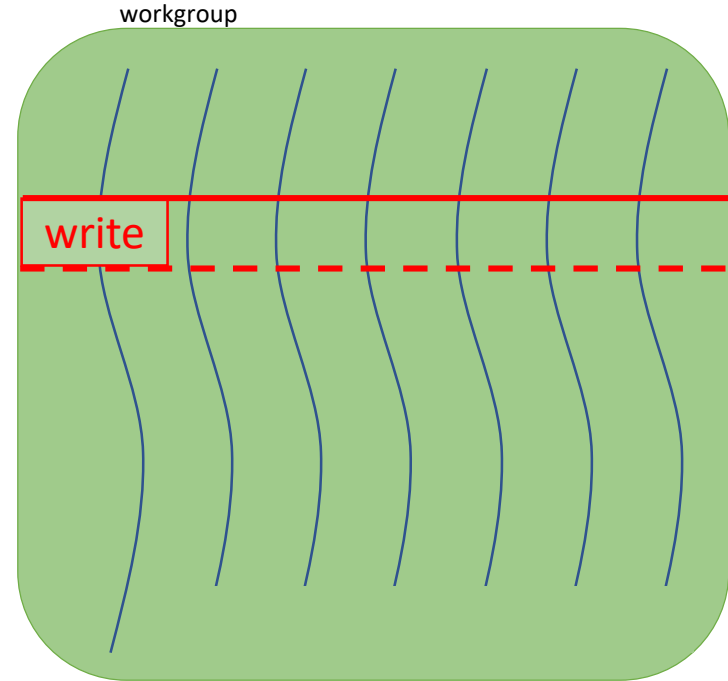
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



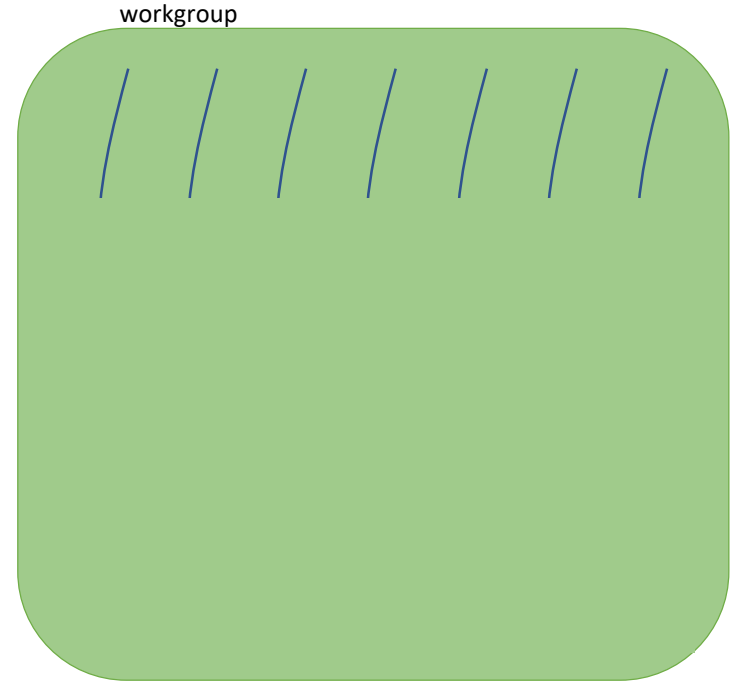
Flexibility: Ordering can be relaxed (group)

- Strict ordering
 - Both barriers
- Relaxed ordering
 - Remove one barrier
 - Before *read*
 - After *write*



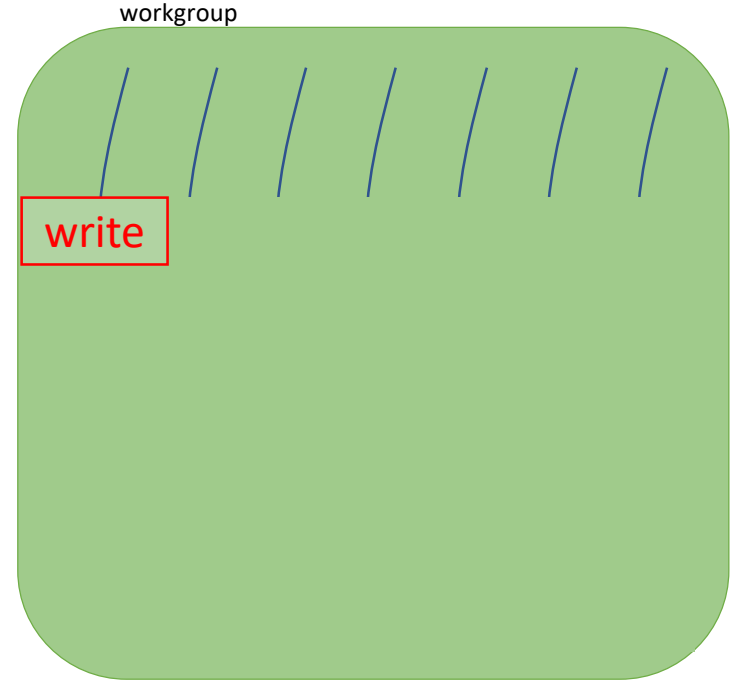
Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



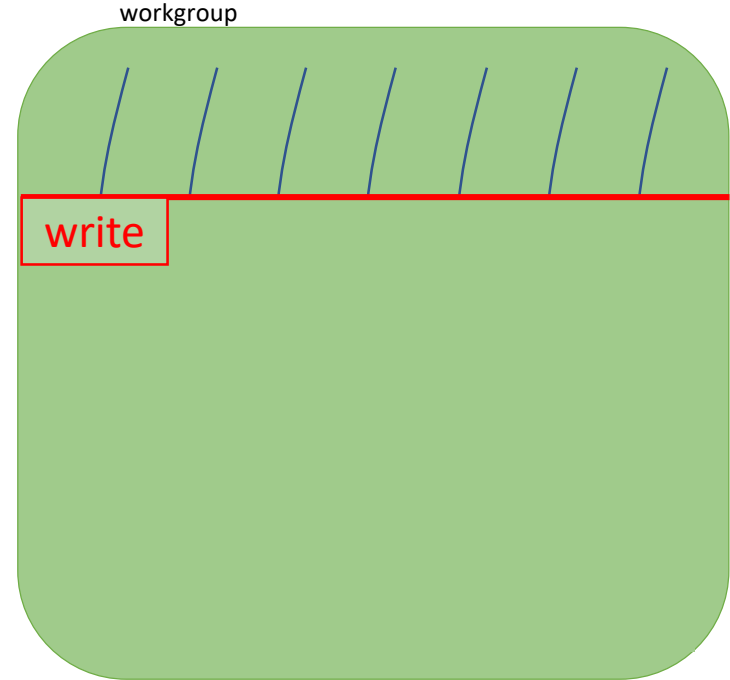
Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



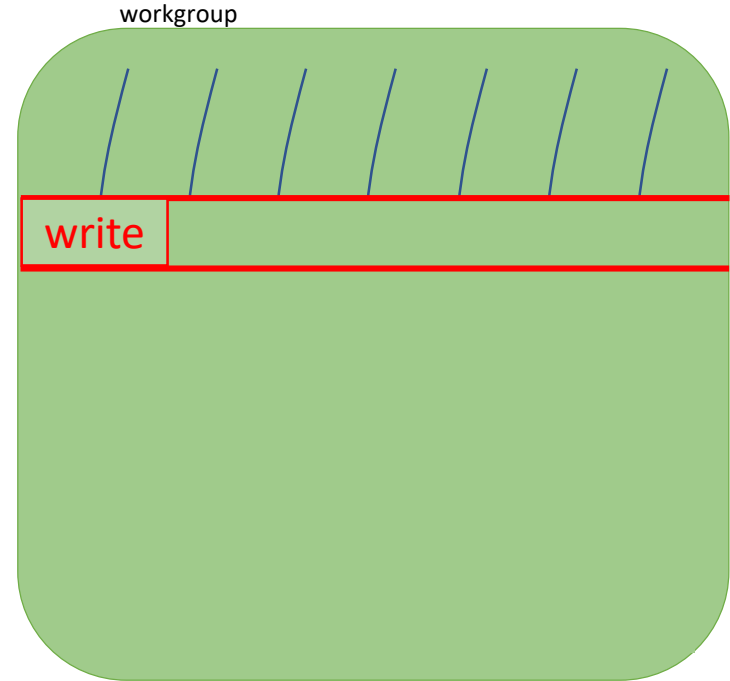
Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



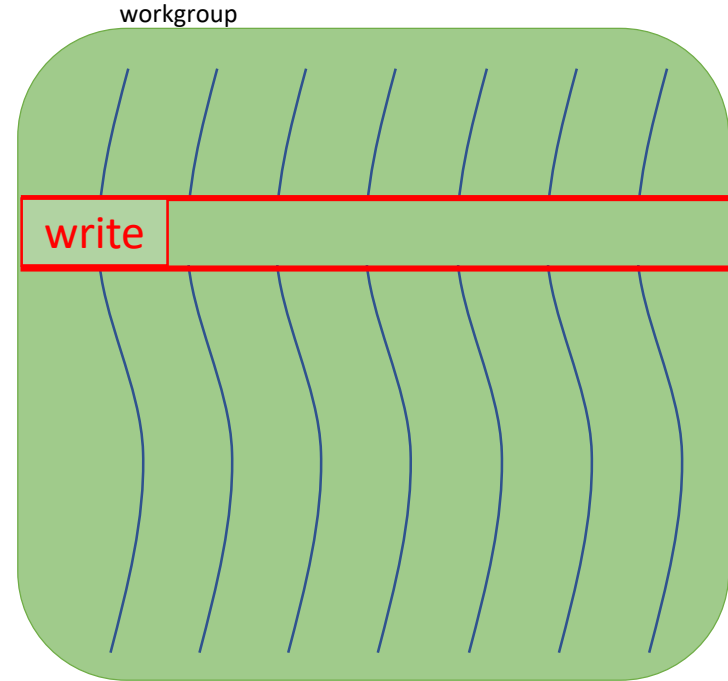
Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



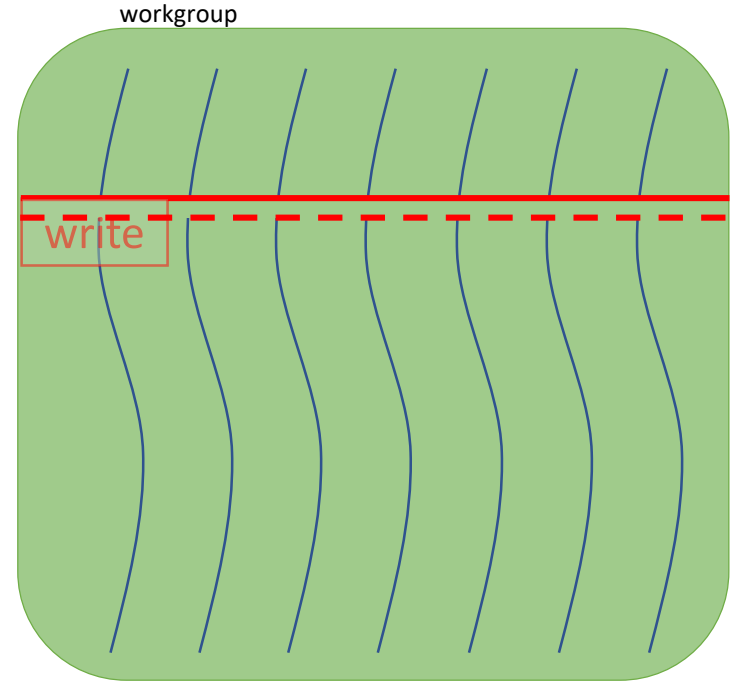
Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



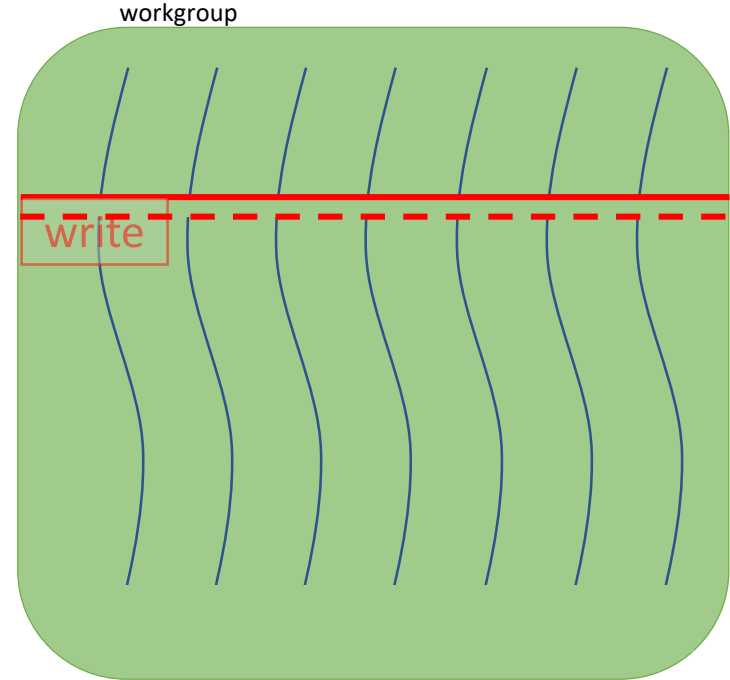
Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



Flexibility: Allow non-blocking invocation

- Blocking invocation
 - Wait for result
- Non-blocking invocation
 - Return value collected later or not at all



Ideal system services properties

- Familiarity
 - Known semantics
- Flexibility
 - Do not restrict programmers
- Adaptability
 - Adapt to workload needs



Adaptability in implementation

Adaptability in implementation

- Don't waste resources
 - Syscall light applications
 - Important for heterogeneous systems
 - Share power and energy budget

Adaptability in implementation

- Don't waste resources
 - Syscall light applications
 - Important for heterogeneous systems
 - Share power and energy budget
- Use as many resources as possible
 - Syscall heavy applications

Implementation

GPU

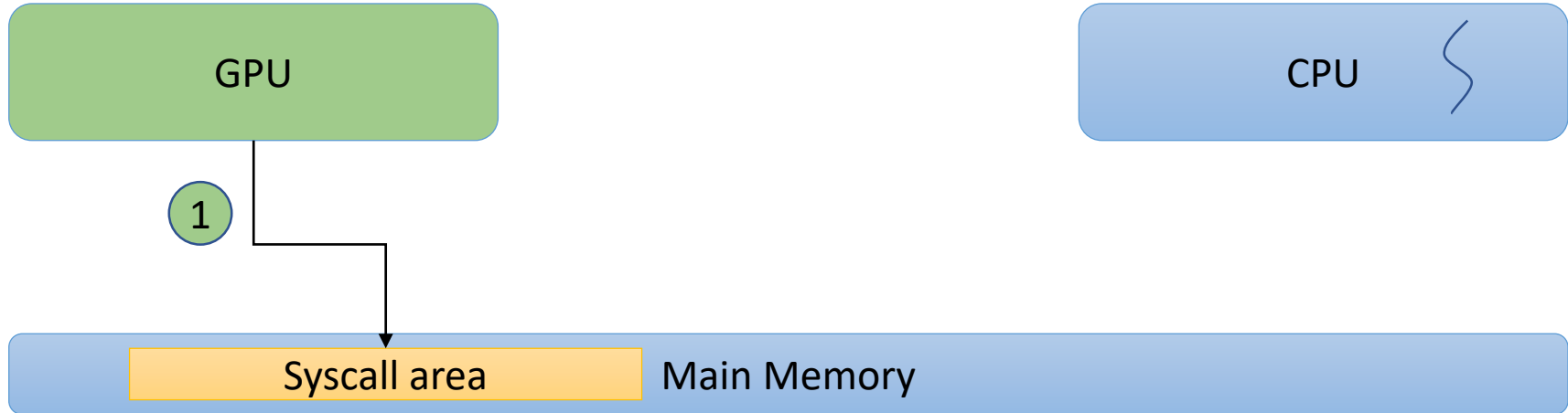
CPU



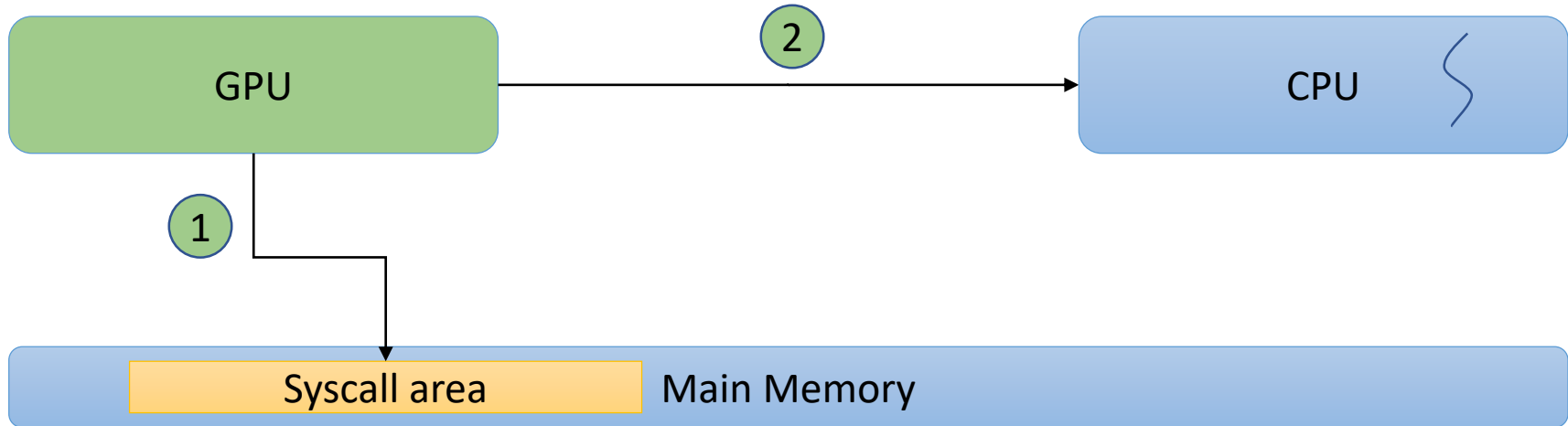
Syscall area

Main Memory

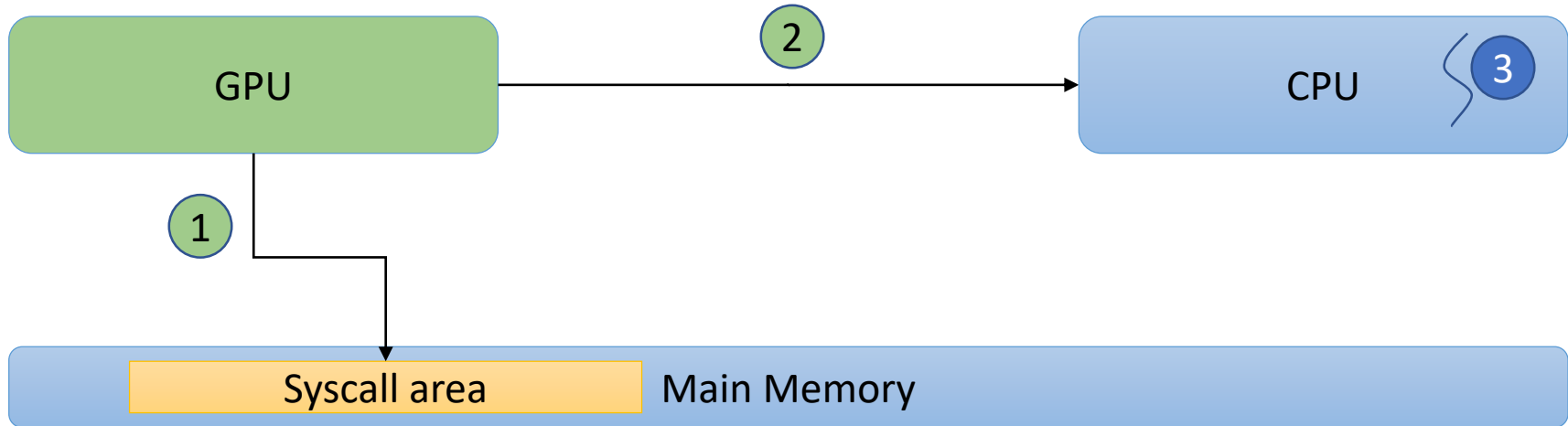
Implementation



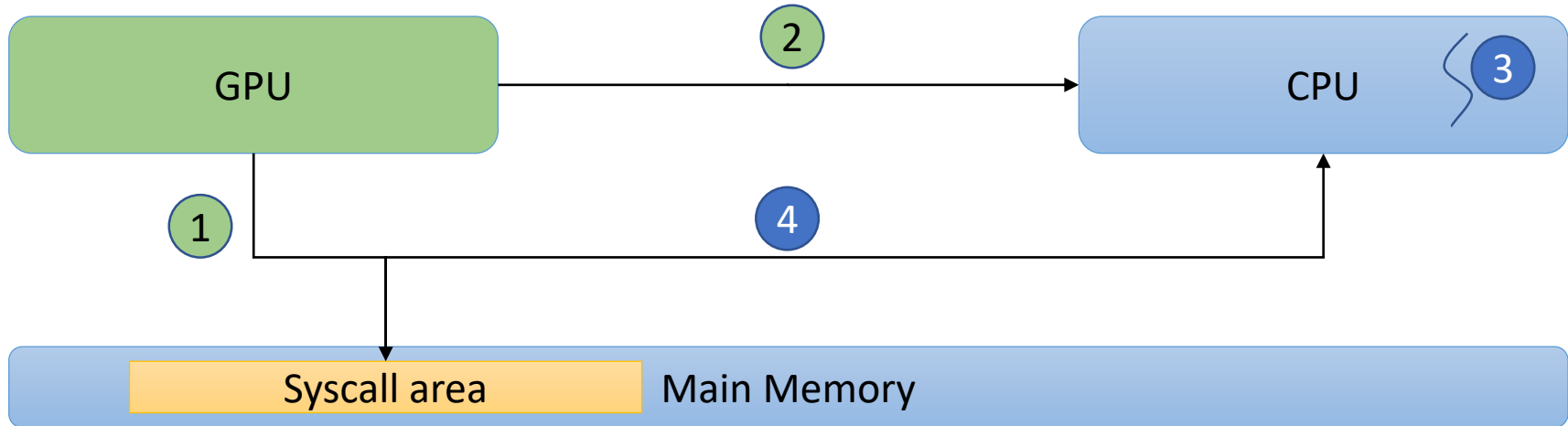
Implementation



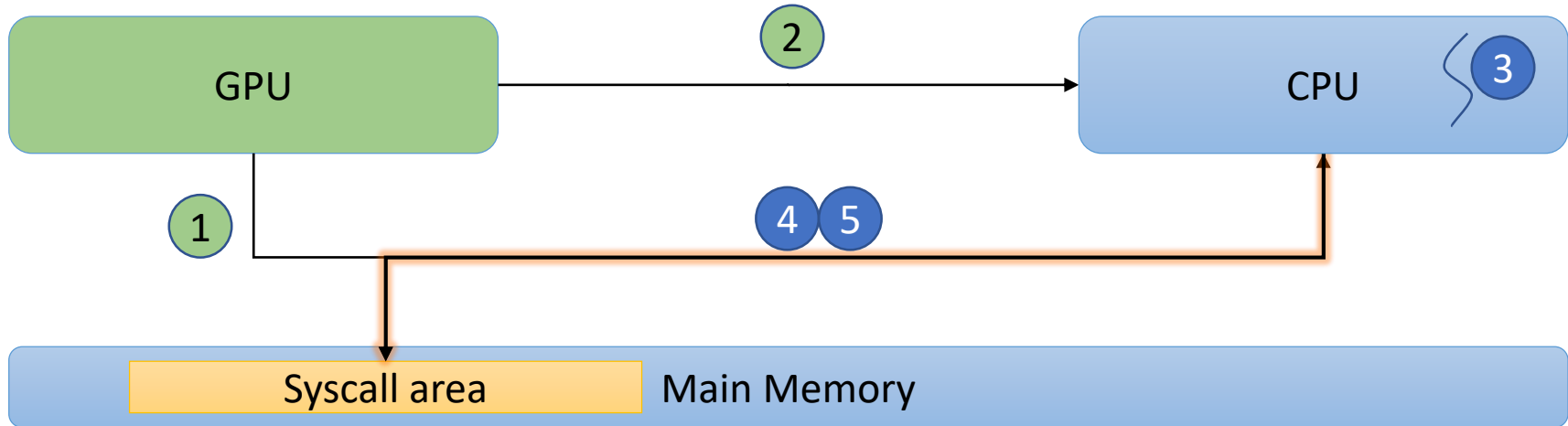
Implementation



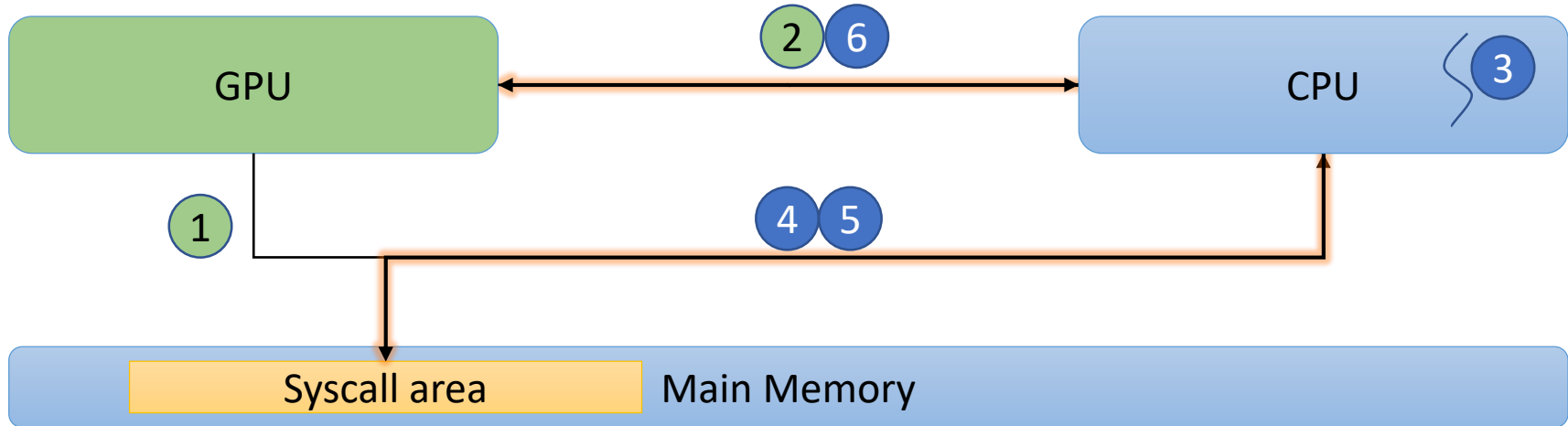
Implementation



Implementation



Implementation



Genesys works on off-the-shelf hardware

Genesys works on off-the-shelf hardware

- AMD FX-9800P
 - 4 CPU cores, 8 CUs (gpu cores)
 - Share 15W of TDP
 - 16GB DDR4 RAM

Genesys works on off-the-shelf hardware

- AMD FX-9800P
 - 4 CPU cores, 8 CUs (gpu cores)
 - Share 15W of TDP
 - 16GB DDR4 RAM
- GPU L2 cache is CPU coherent
 - GPU L1 coherence is handled in software
 - Provides CPU $\Leftrightarrow$ GPU atomic operations

Ideal system services properties

- Familiarity
 - Known semantics
- Flexibility
 - Do not restrict programmers
- Adaptability
 - Adapt to workload needs



Genesys supports wide range of use cases

- Storage

Genesys supports wide range of use cases

- Storage
- Networking

Genesys supports wide range of use cases

- Storage
- Networking
- Memory Management

Genesys supports wide range of use cases

- Storage
- Networking
- Memory Management
- Device Control

Storage workload — grep

Storage workload — grep

- Parallelize across number of files

Storage workload — grep

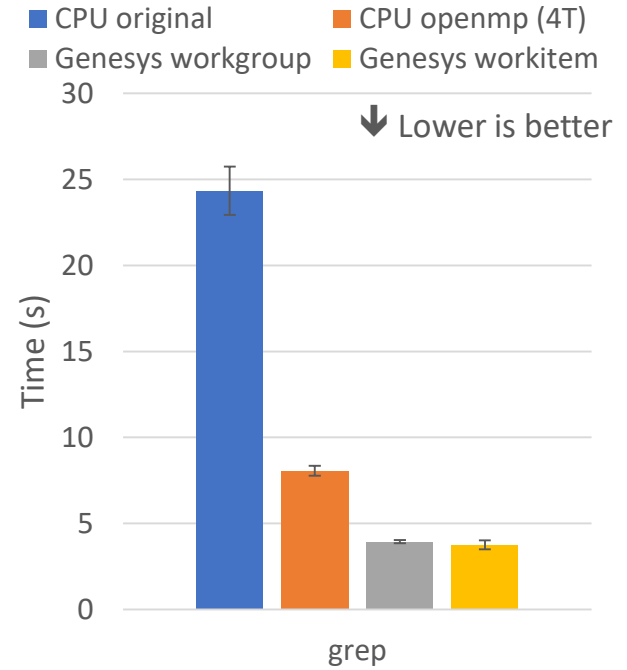
- Parallelize across number of files
- Exploit high throughput storage devices

Storage workload — grep

- Parallelize across number of files
- Exploit high throughput storage devices
- Each workitem (thread):
 - open, read, write(stdout), close

Storage workload — grep

- Parallelize across number of files
- Exploit high throughput storage devices
- Each workitem (thread):
 - open, read, write(stdout), close



Networking workload — memcached

Networking workload — memcached

- Heterogeneous application

Networking workload — memcached

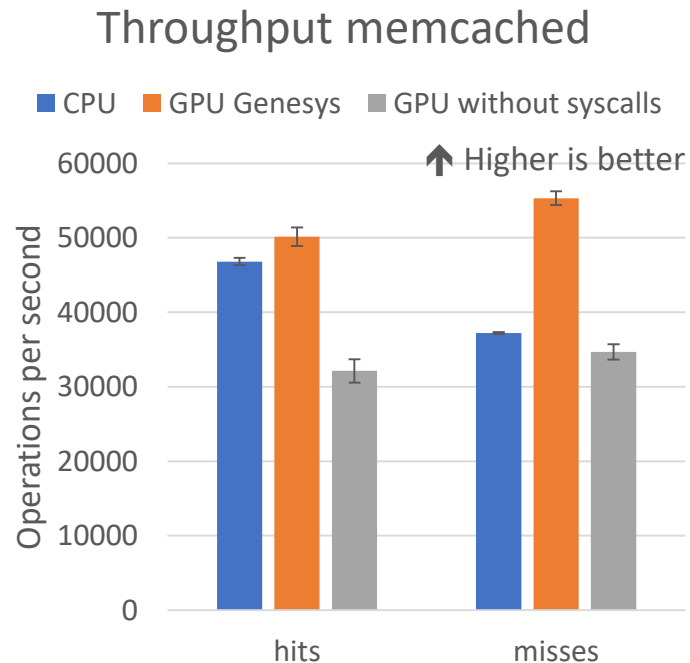
- Heterogeneous application
 - CPU and GPU work on the same data
 - SET $\Rightarrow$ CPU
 - GET $\Rightarrow$ GPU, CPU

Networking workload — memcached

- Heterogeneous application
 - CPU and GPU work on the same data
 - SET $\Rightarrow$ CPU
 - GET $\Rightarrow$ GPU, CPU
- Each workgroup (thread group)
 - `recvmsg`, `write(stderr)`, `sendmsg`
 - Parallelize; hash, lookup, data copy

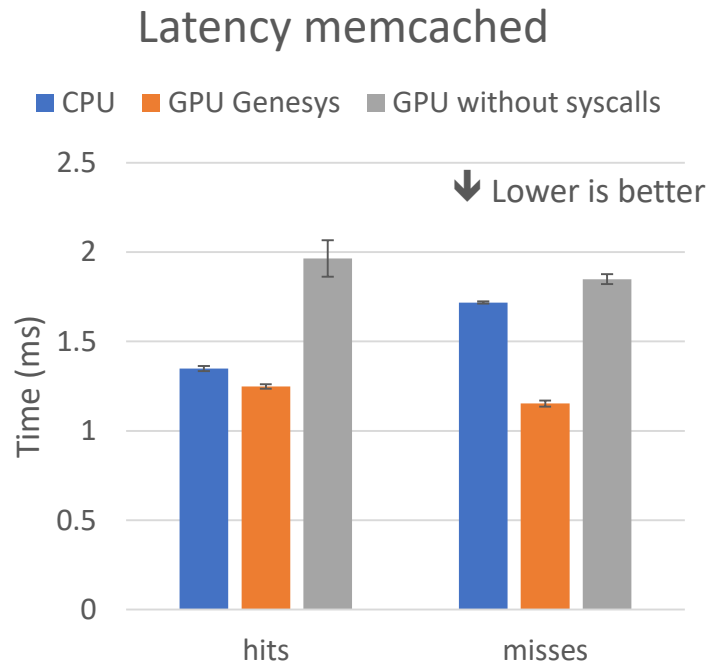
Networking workload — memcached

- Heterogeneous application
 - CPU and GPU work on the same data
 - SET $\Rightarrow$ CPU
 - GET $\Rightarrow$ GPU, CPU
- Each workgroup (thread group)
 - recvmsg, write(stderr), sendmsg
 - Parallelize; hash, lookup, data copy



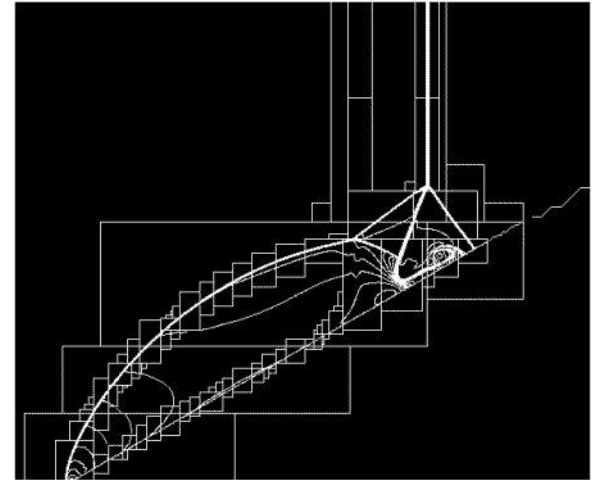
Networking workload — memcached

- Heterogeneous application
 - CPU and GPU work on the same data
 - SET $\Rightarrow$ CPU
 - GET $\Rightarrow$ GPU, CPU
- Each workgroup (thread group)
 - recvmsg, write(stderr), sendmsg
 - Parallelize; hash, lookup, data copy



Memory management — miniAMR

- Algorithm includes memory allocator
 - Adaptive mesh refining
- Enable judicious use of system resources
 - Accelerator multiprocessing
- Coarsening workitems (threads)
 - `madvise(MADV_DONTNEED)`



Device control — ioctl

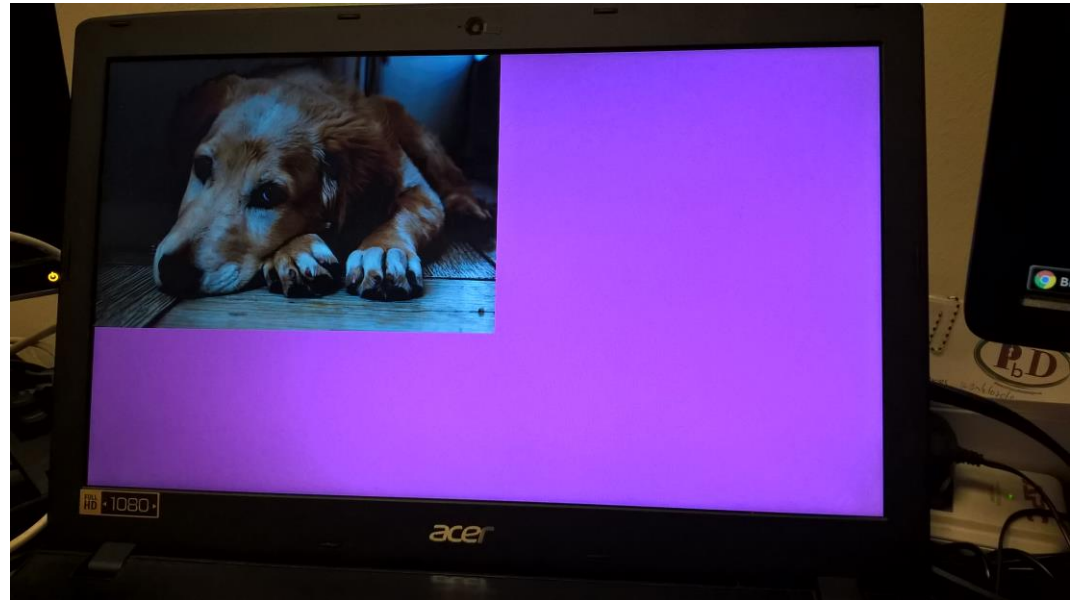
- Audio devices
- USB devices
- Network devices
- GPU!

Device control — ioctl

- Audio devices
 - USB devices
 - Network devices
 - GPU!
-
- Display frame buffer

Device control — ioctl

- Audio devices
 - USB devices
 - Network devices
 - GPU!
-
- Display frame buffer



Conclusion

- Generic POSIX®-like system calls for GPUs are viable
- Improvement in programming environment leads to new applications and improved performance of traditional ones
- All code is available on github, hosted by AMD ROCm project
 - https://github.com/RadeonOpenCompute/{ROCK,ROCT,HCC}_syscall

Disclaimer & Attribution

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 for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. 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.

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 MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, Radeon, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.