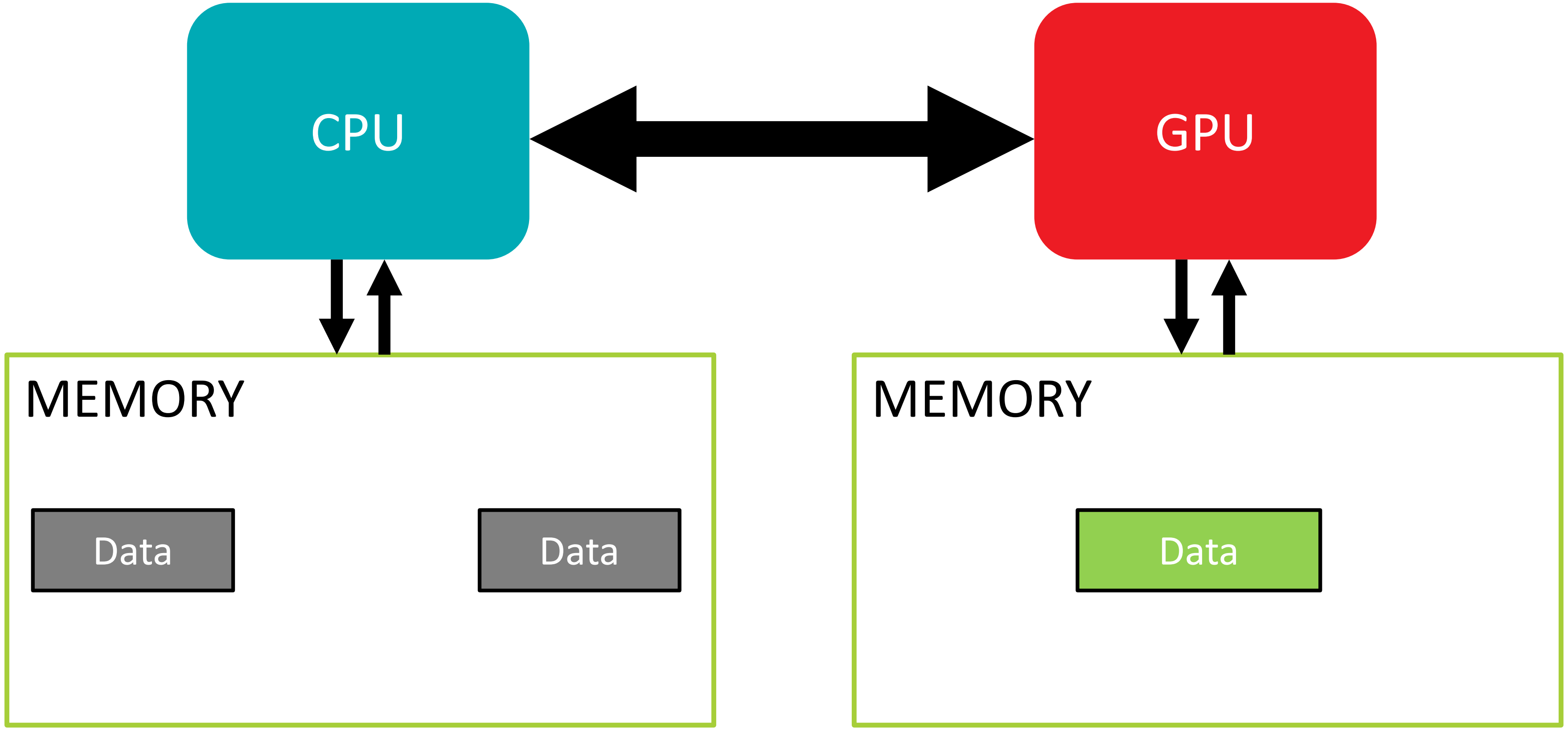
clARMOR: A DYNAMIC BUFFER OVERFLOW DETECTOR FOR OPENCL KERNELS

CHRIS ERB, JOE GREATHOUSE,
MAY 16, 2018

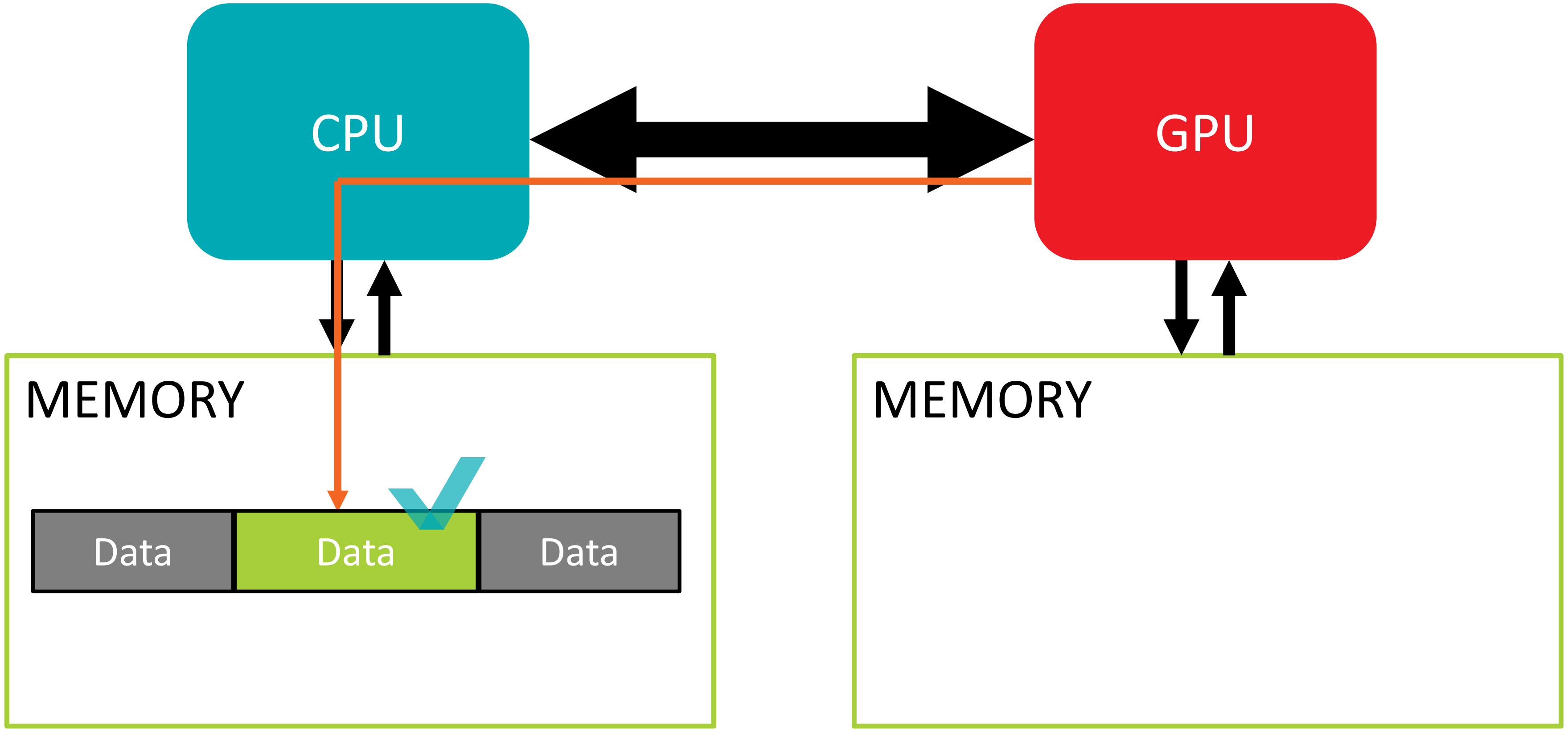
ANECDOTE

DISCOVERING A BUFFER OVERFLOW



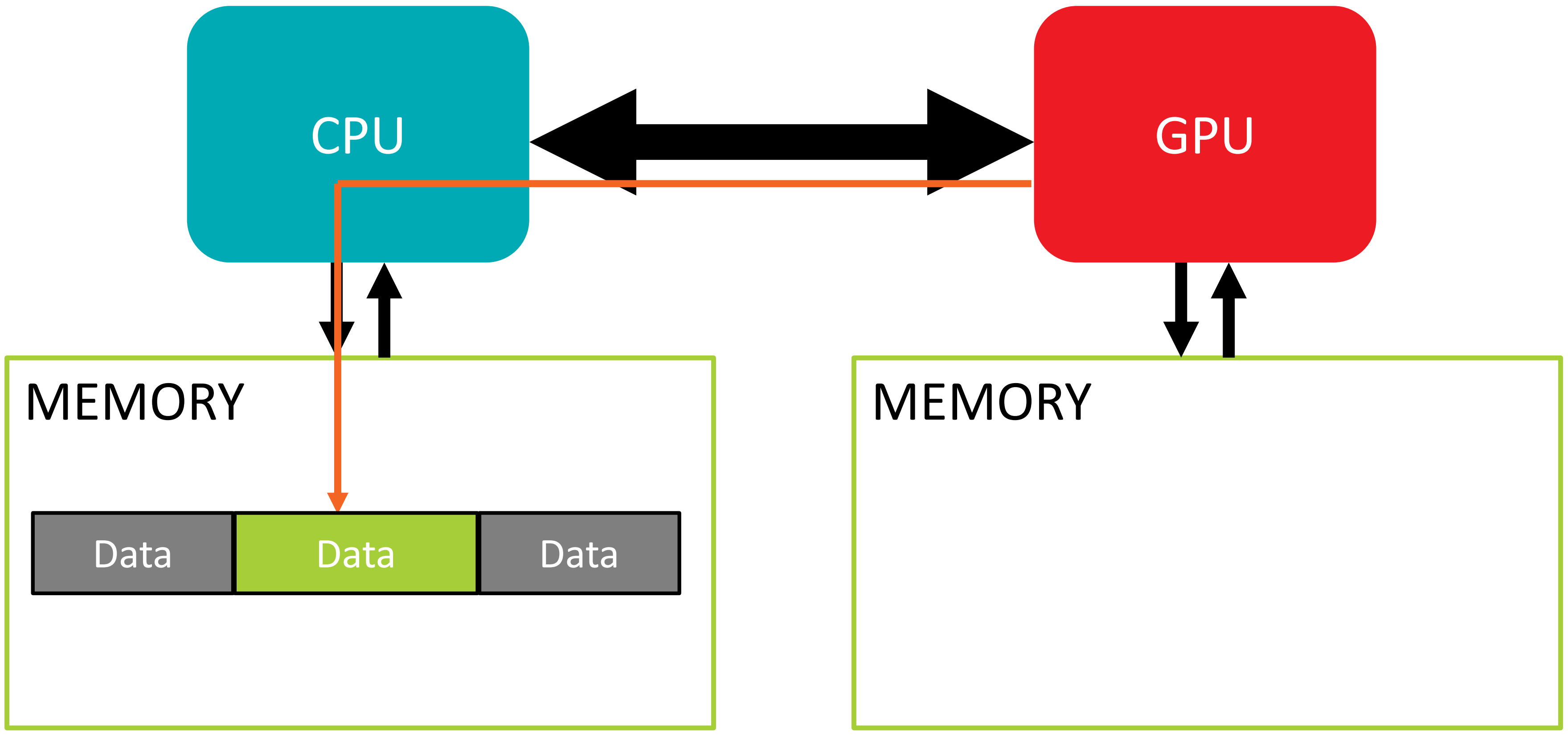
ANECDOTE

DISCOVERING A BUFFER OVERFLOW

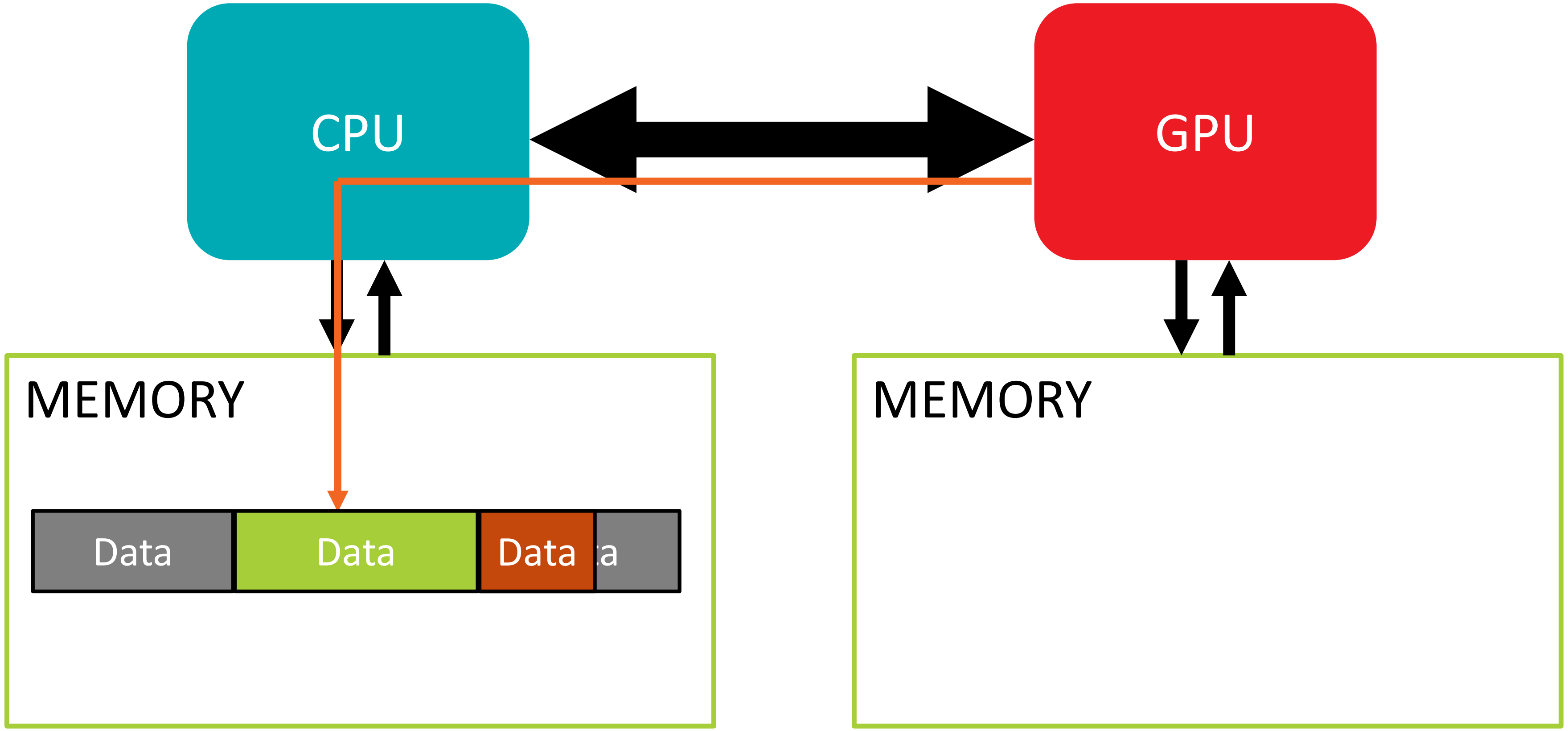


ANECDOTE

DISCOVERING A BUFFER OVERFLOW

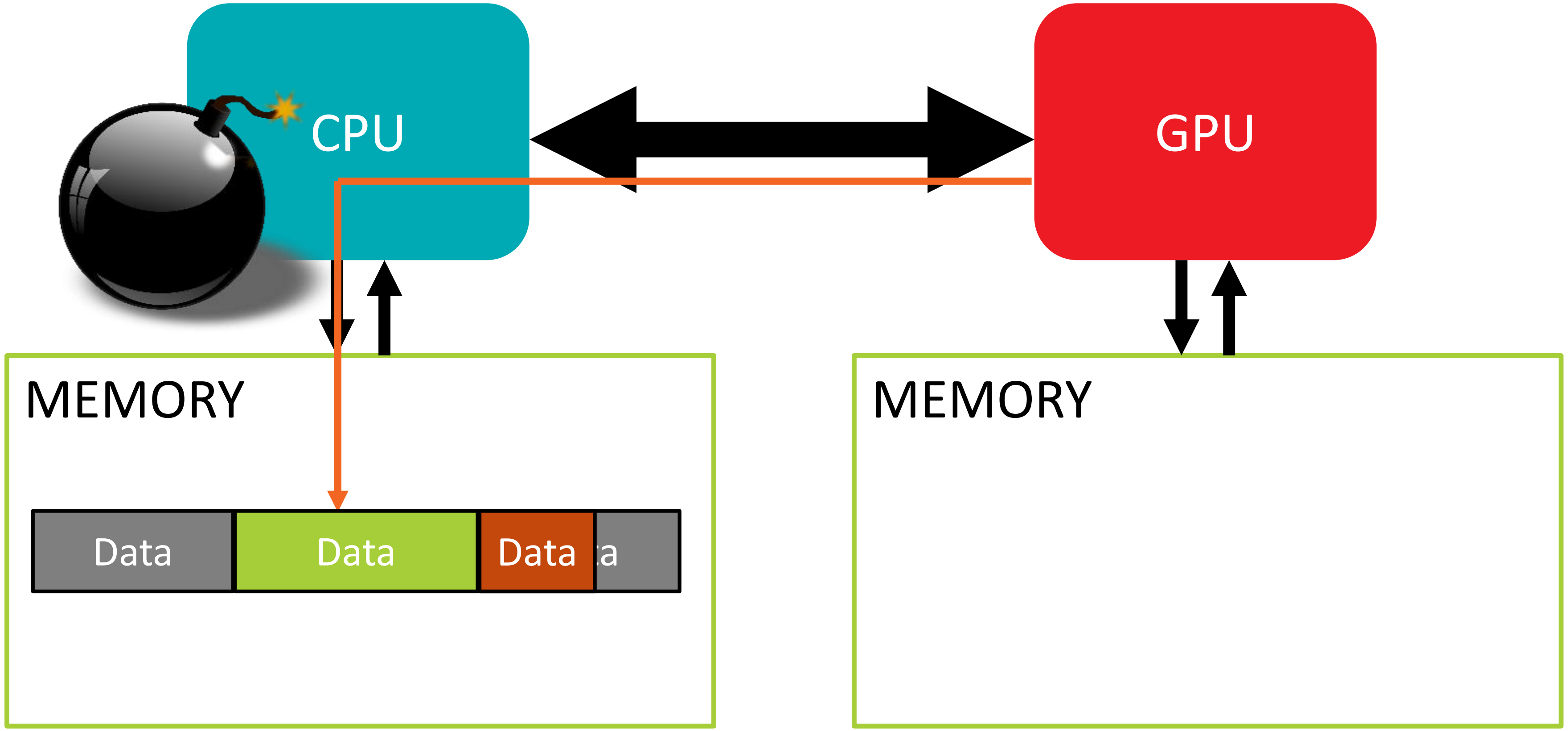


ANECDOTE
DISCOVERING A BUFFER OVERFLOW



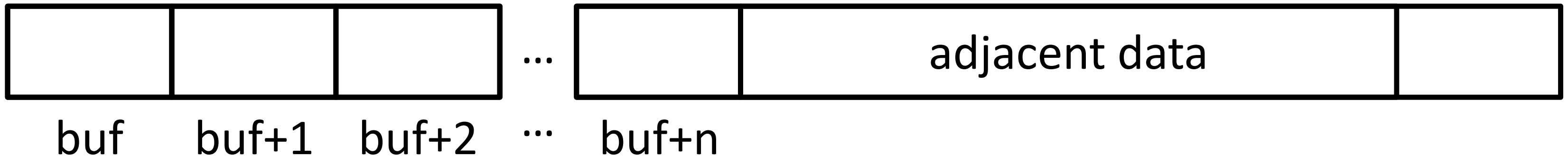
ANECDOTE

DISCOVERING A BUFFER OVERFLOW



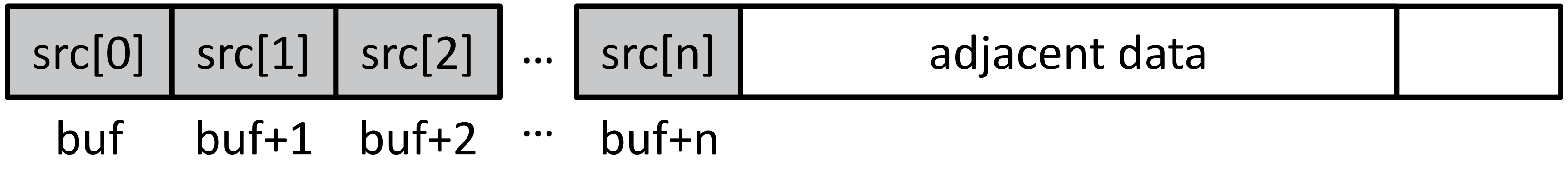
BACKGROUND: NORMAL BUFFER FILL

- buf[n+1]
- memcpy(buf, src, n+1)



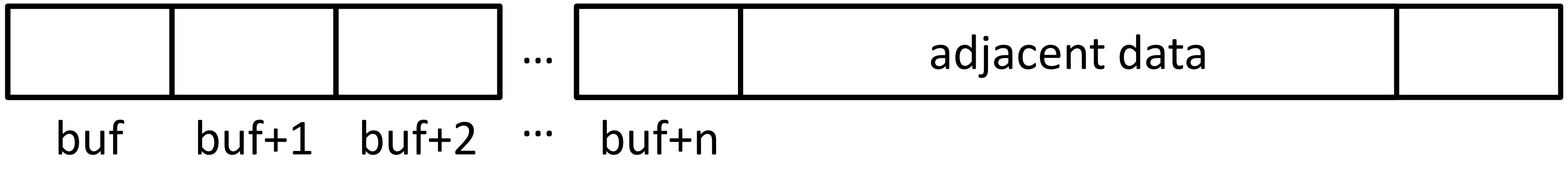
BACKGROUND: NORMAL BUFFER FILL

- buf[n+1]
- memcpy(buf, src, n+1)



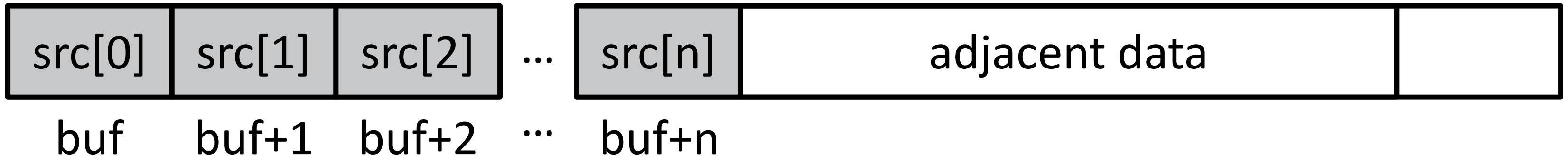
BACKGROUND: BUFFER OVERFLOW

- buf[n+1]
- memcpy(buf, src, n+5)



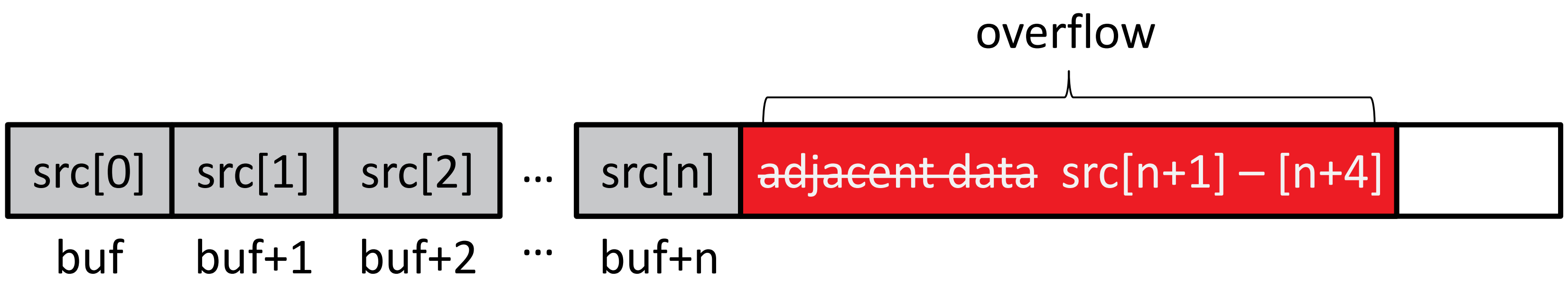
BACKGROUND: BUFFER OVERFLOW

- buf[n+1]
- memcpy(buf, src, n+5)



BACKGROUND: BUFFER OVERFLOW

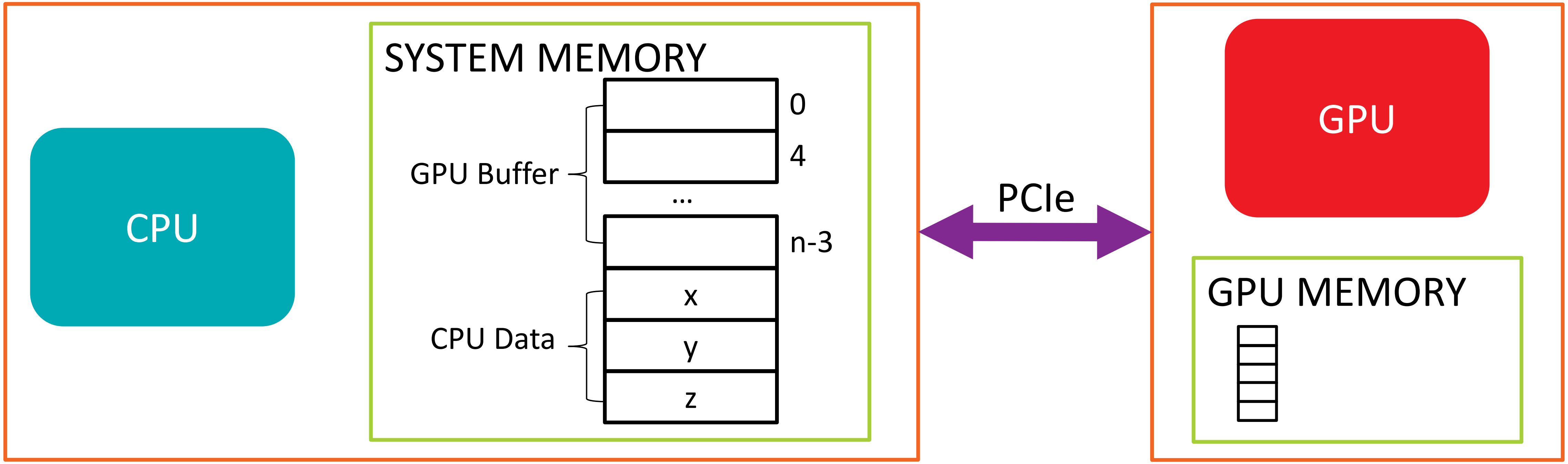
- buf[n+1]
- memcpy(buf, src, n+5)



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

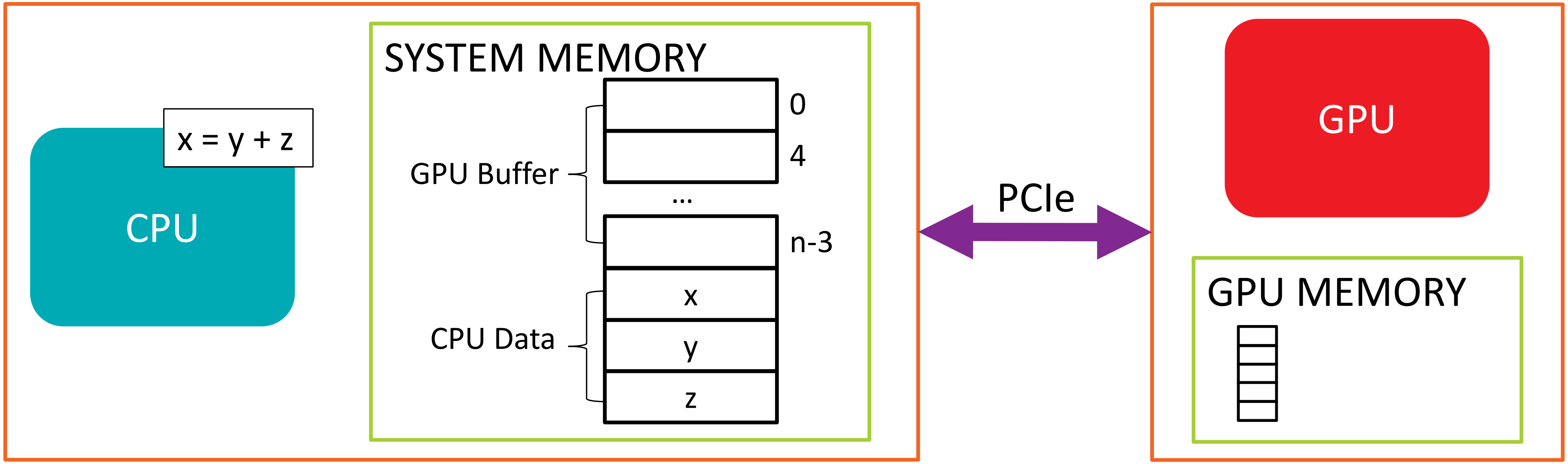
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

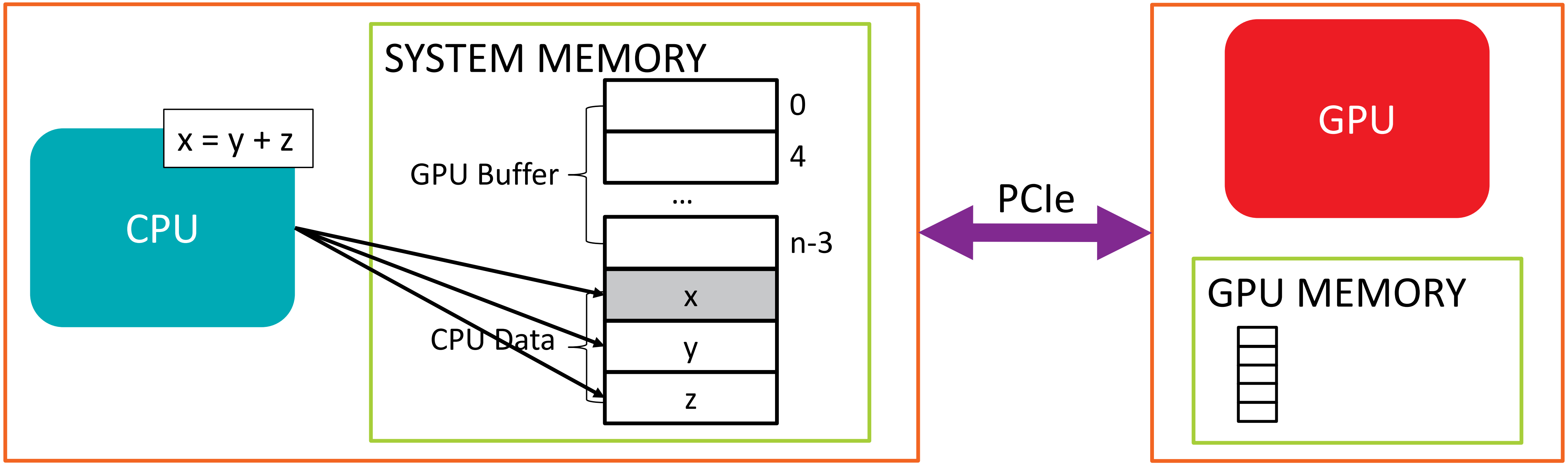
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

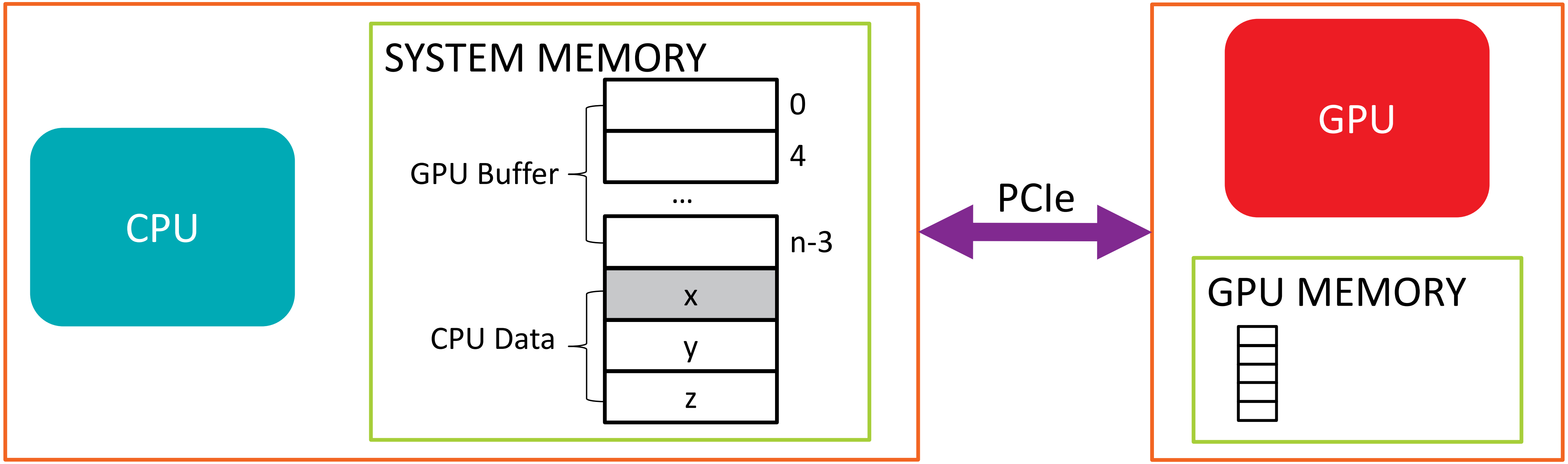
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

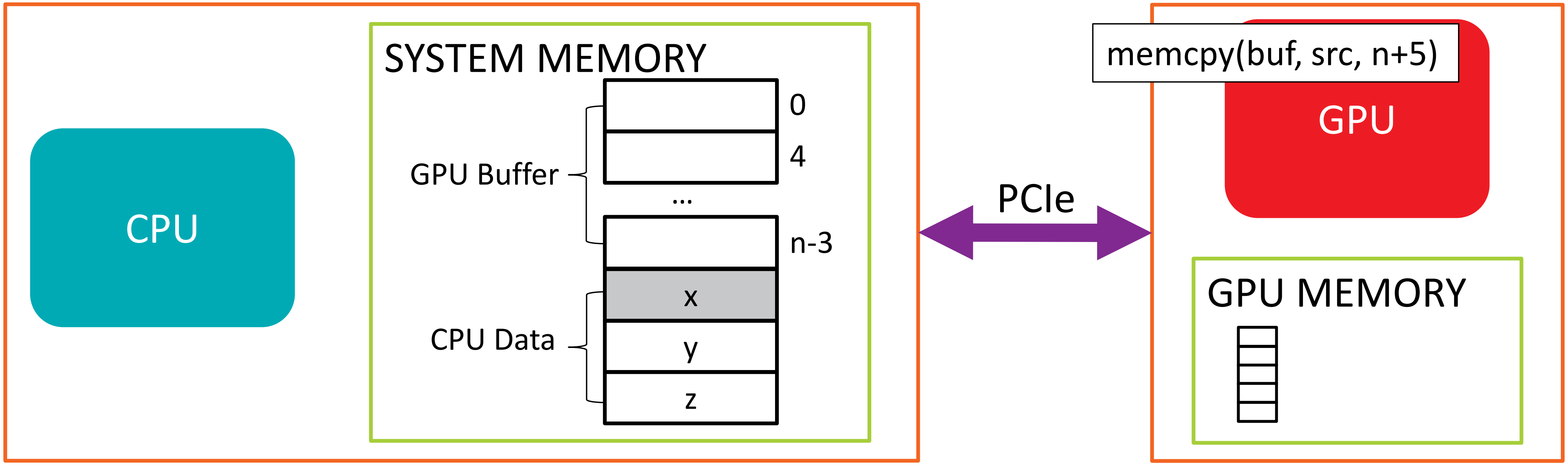
SHARED MEMORY CORRUPTION

- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW
SHARED MEMORY CORRUPTION

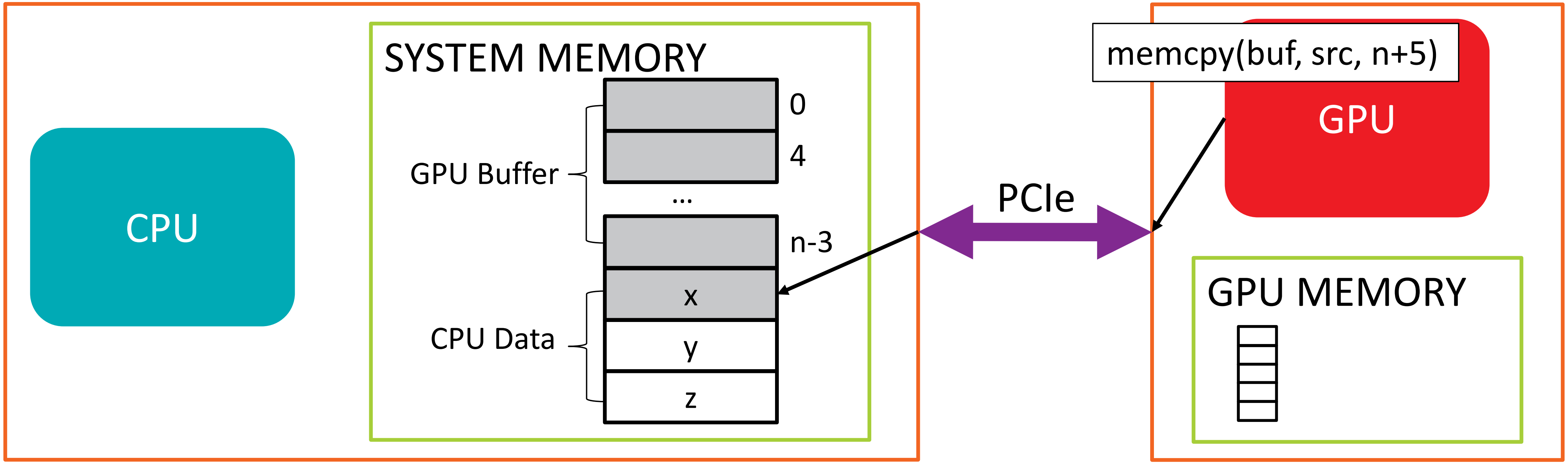
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

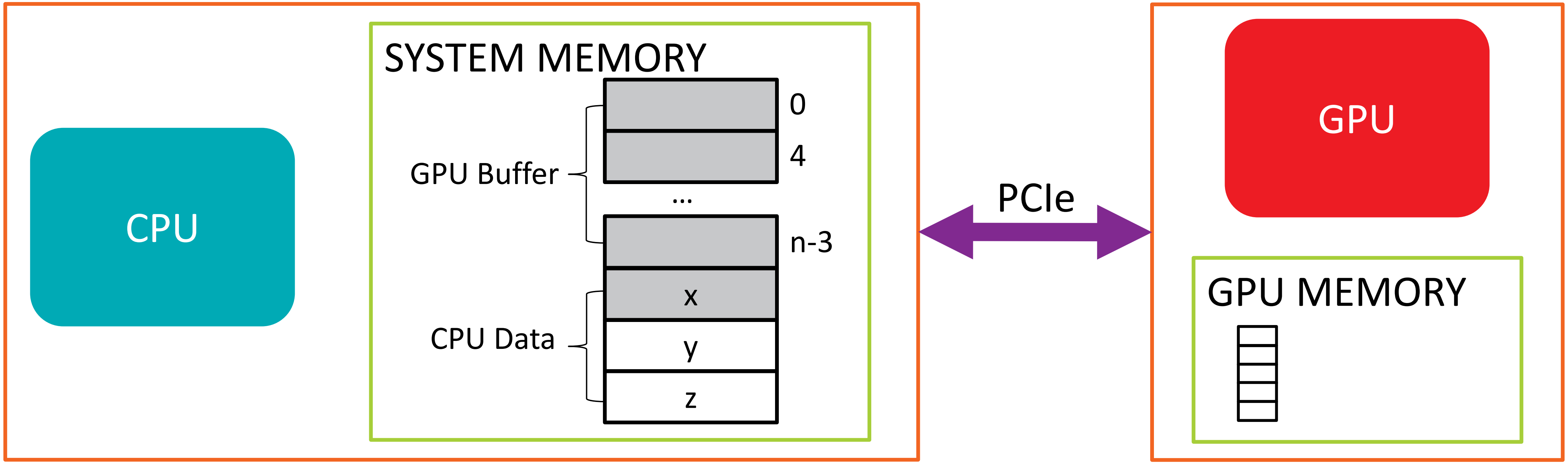
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

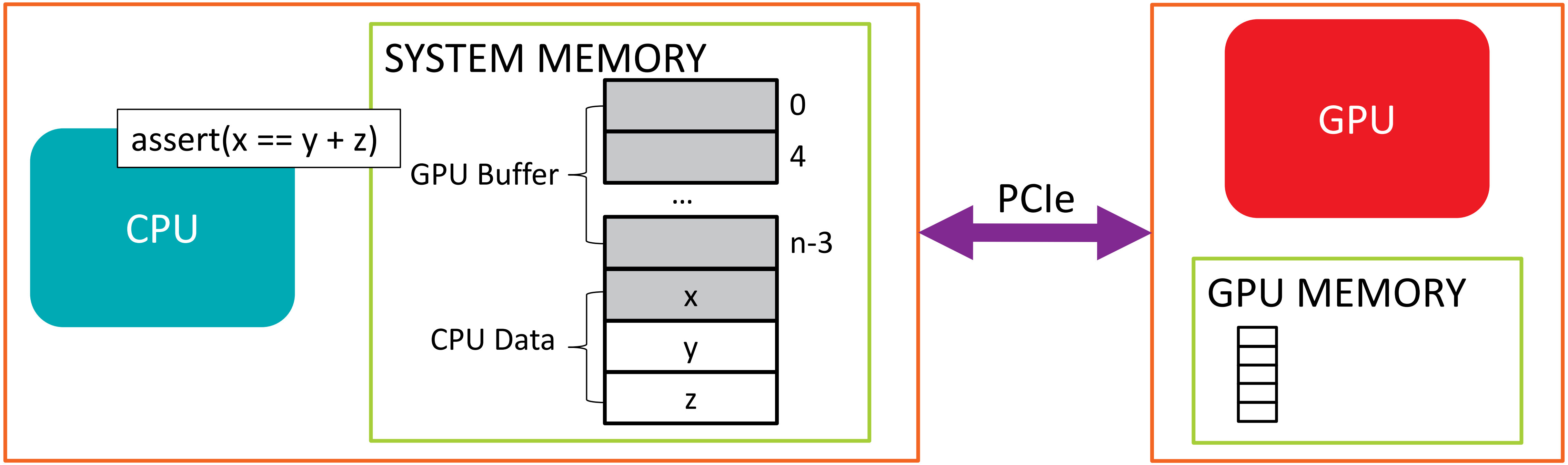
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

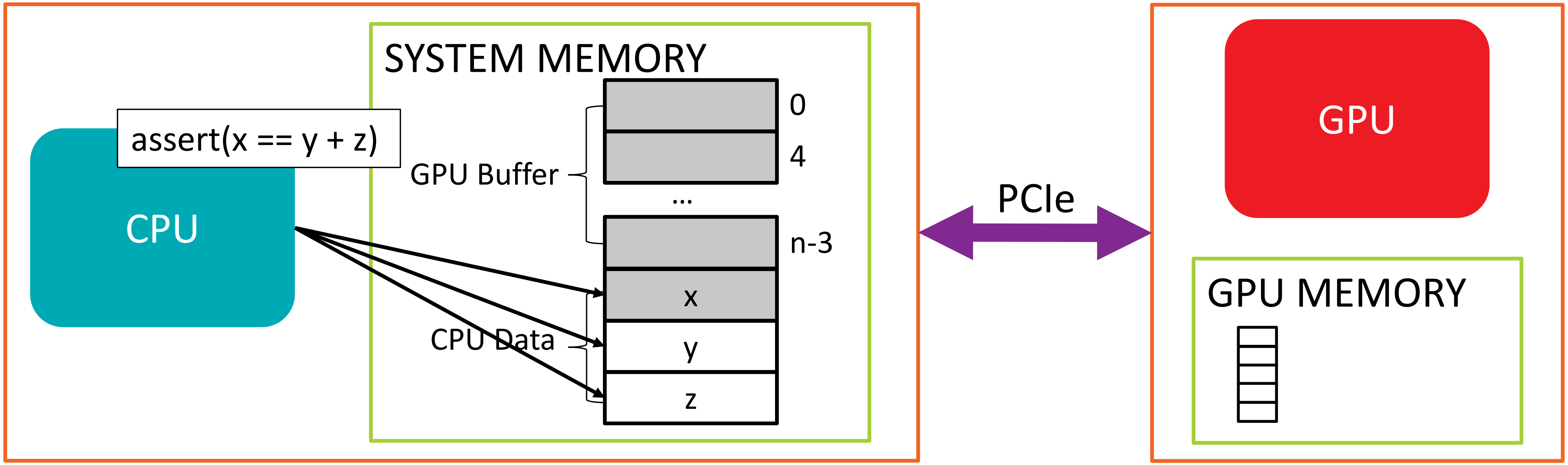
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

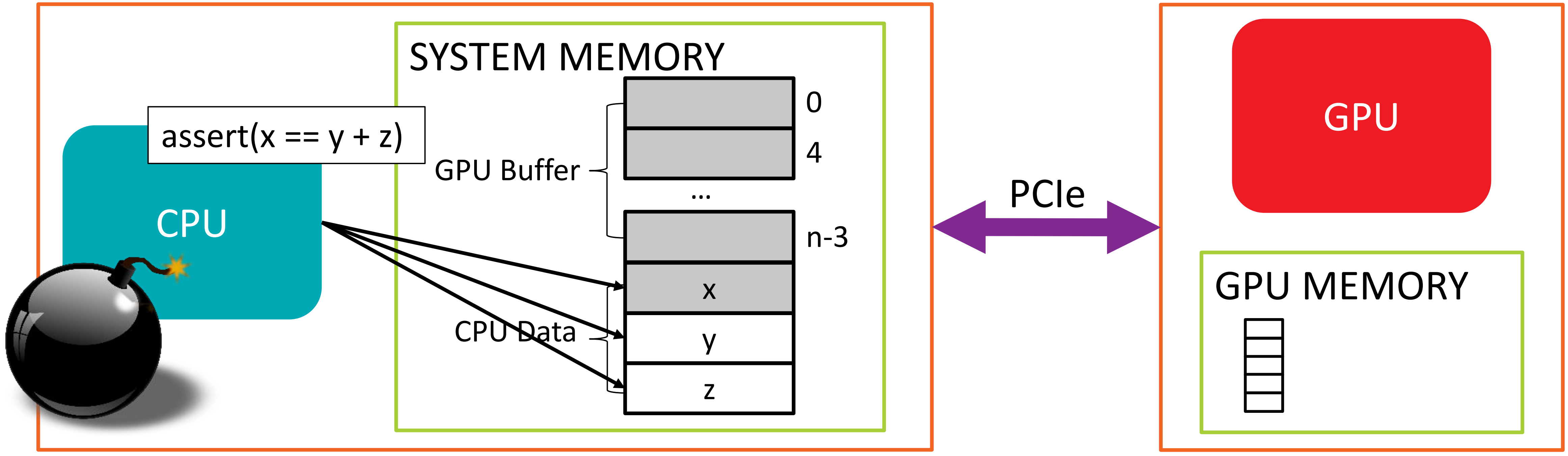
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

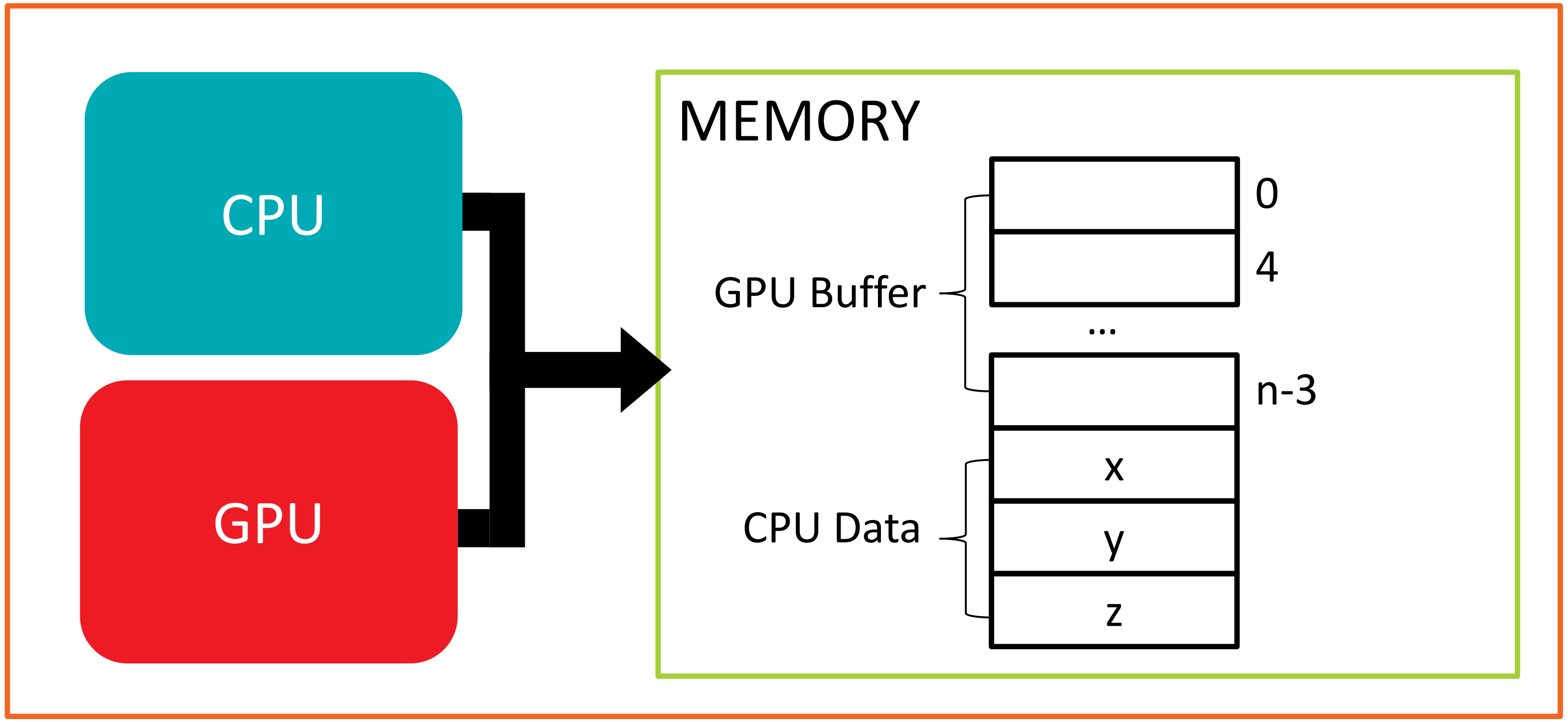
- GPU can overflow buffers in system memory
  - Over Interconnects like PCIe®



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

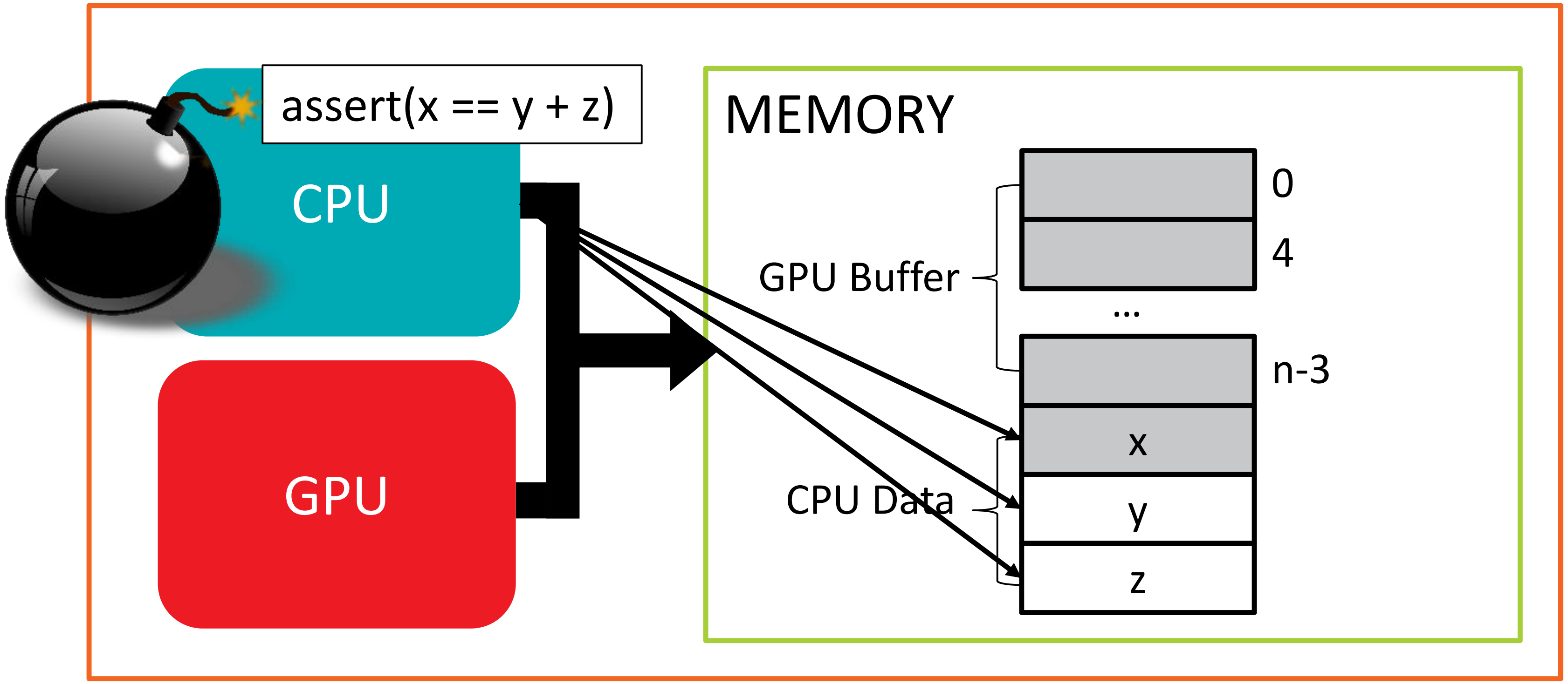
- CPU and GPU as part of the same package



GPU INDUCED OVERFLOW

SHARED MEMORY CORRUPTION

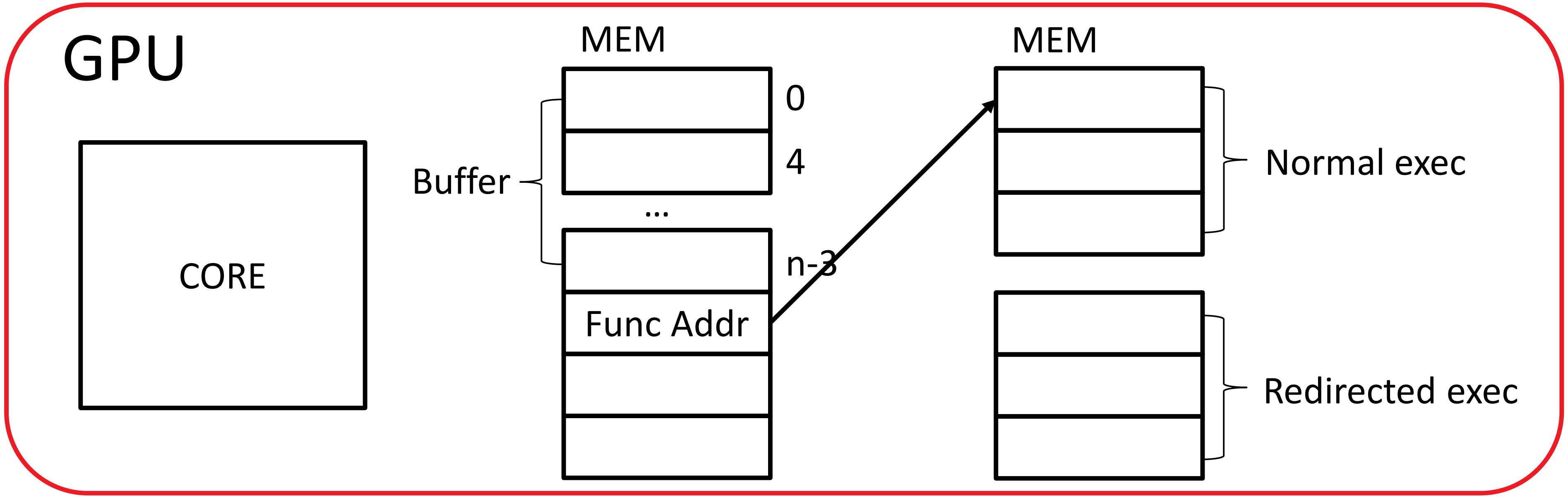
- CPU and GPU as part of the same package
  - Every GPU buffer overflow may affect CPU data



GPU INDUCED OVERFLOW

REMOTE CODE EXECUTION

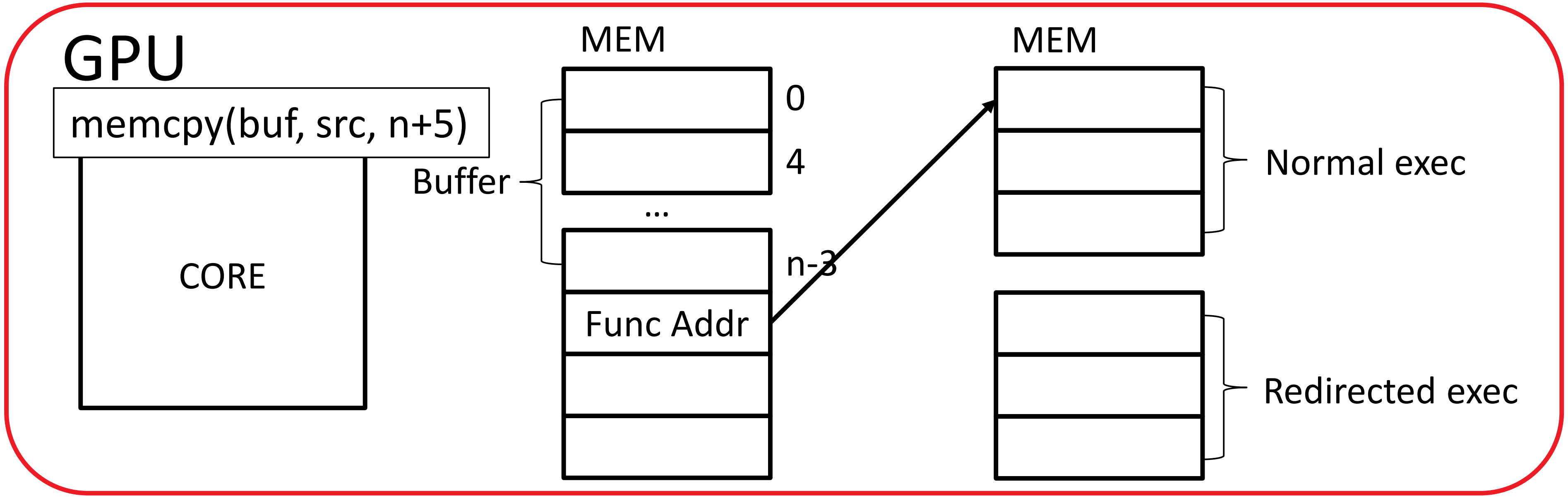
- Overflows on GPU can cause remote GPU code execution
  - A. Miele. *Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis.*
  - B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*



GPU INDUCED OVERFLOW

REMOTE CODE EXECUTION

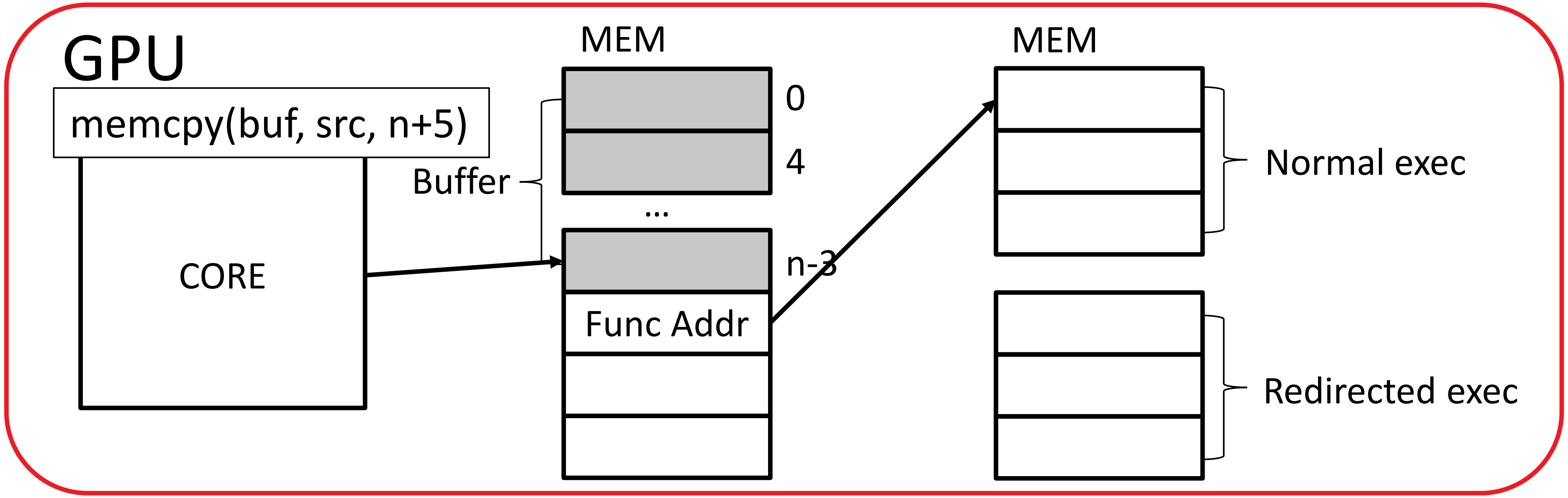
- Overflows on GPU can cause remote GPU code execution
  - A. Miele. *Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis.*
  - B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*



GPU INDUCED OVERFLOW

REMOTE CODE EXECUTION

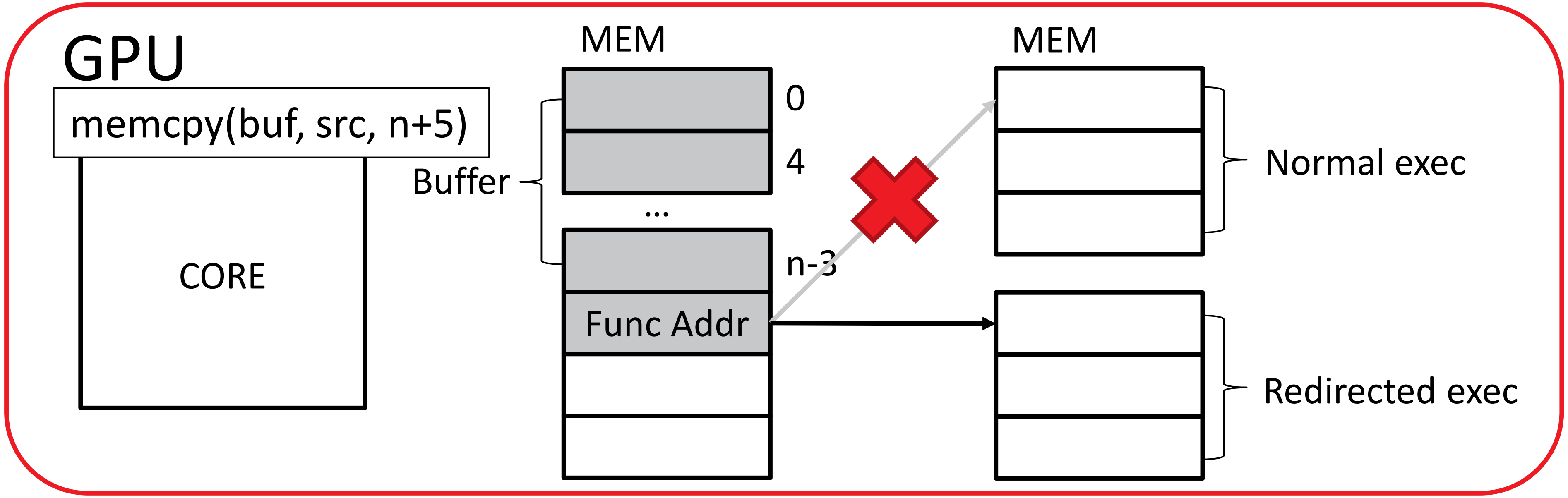
- Overflows on GPU can cause remote GPU code execution
  - A. Miele. *Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis.*
  - B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*



GPU INDUCED OVERFLOW

REMOTE CODE EXECUTION

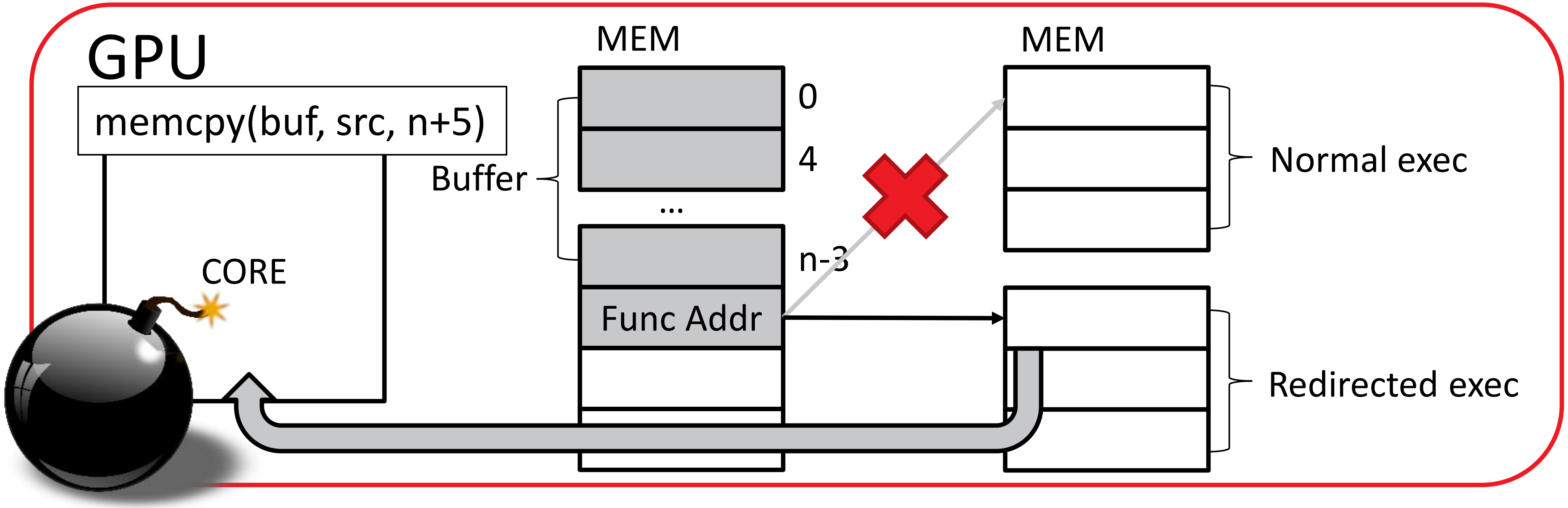
- Overflows on GPU can cause remote GPU code execution
  - A. Miele. *Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis.*
  - B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*



GPU INDUCED OVERFLOW

REMOTE CODE EXECUTION

- Overflows on GPU can cause remote GPU code execution
  - A. Miele. *Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis.*
  - B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*



GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU

- Runnable with most OpenCL™ applications

- Low runtime overhead

GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection
- Runnable with most OpenCL™ applications
- Low runtime overhead

GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection

- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU devices from multiple vendors

- Low runtime overhead

GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection

- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU devices from multiple vendors

- Low runtime overhead
  - 9.7% average overhead

GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

AMD

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection
- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU devices from multiple vendors
- Low runtime overhead
  - 9.7% average overhead

GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection
- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU device types from multiple vendors
- Low runtime overhead
  - 9.7% average overhead

BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

AMD

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+1)

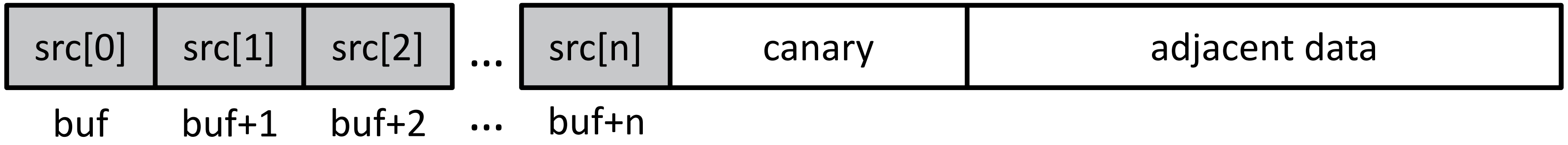
	src[1]	src[2]	…	src[n]	canary	adjacent data
buf	buf+1	buf+2	…	buf+n		

BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+1)

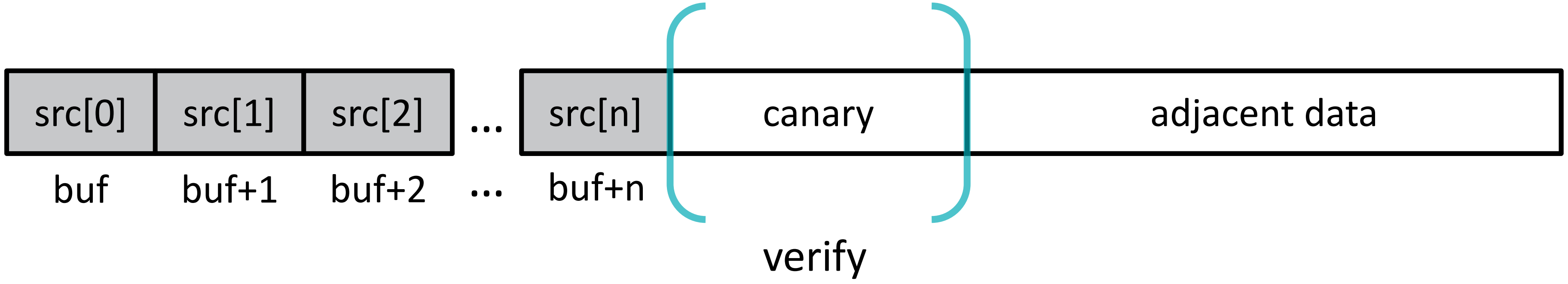


BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+1)

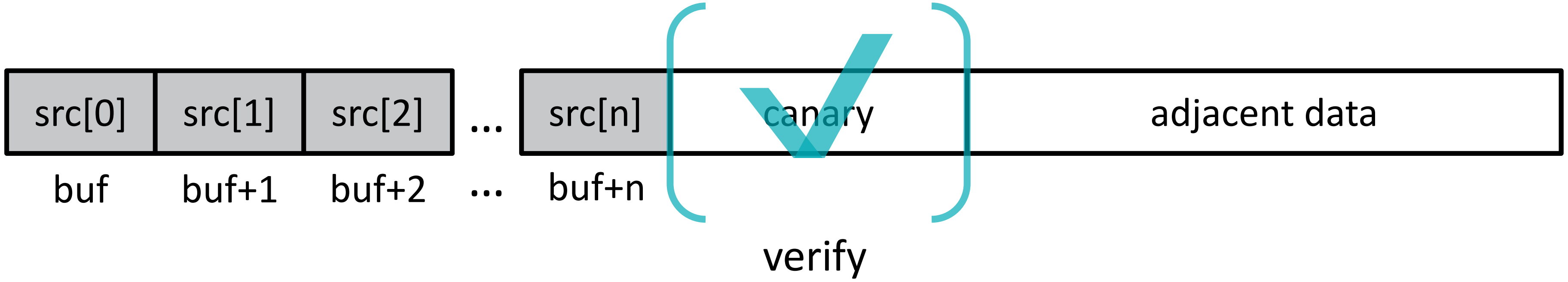


BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+1)

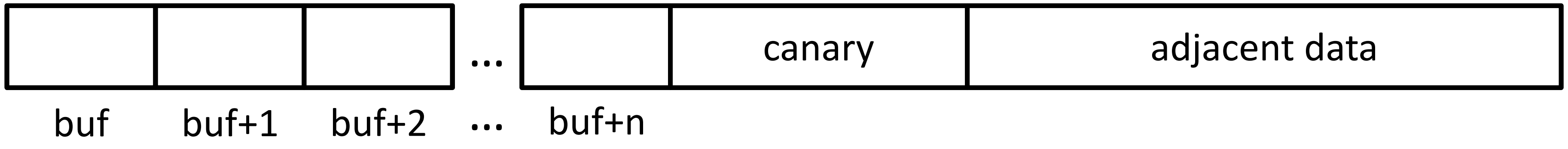


BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

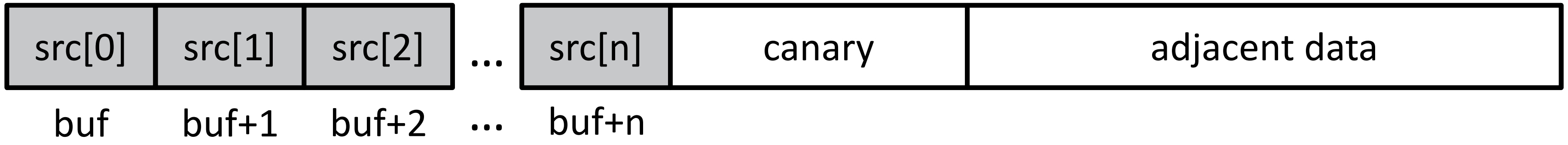
- buf[n+1]
- memcpy(buf, src, n+5)



BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

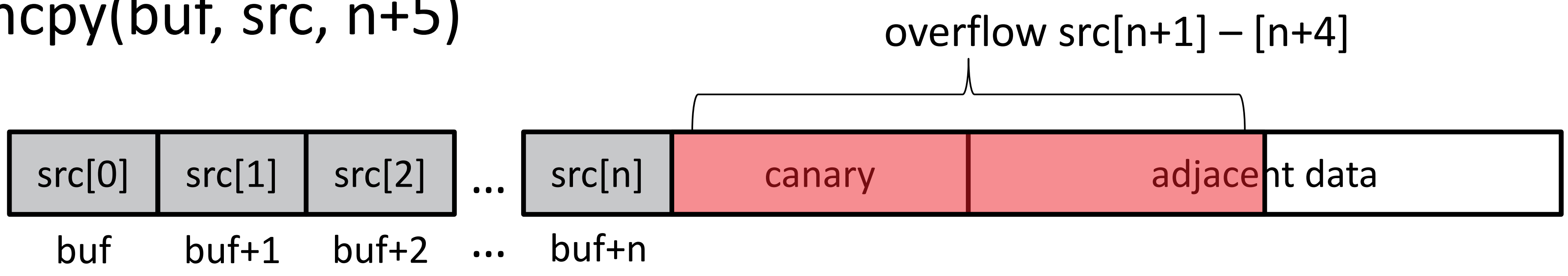


BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

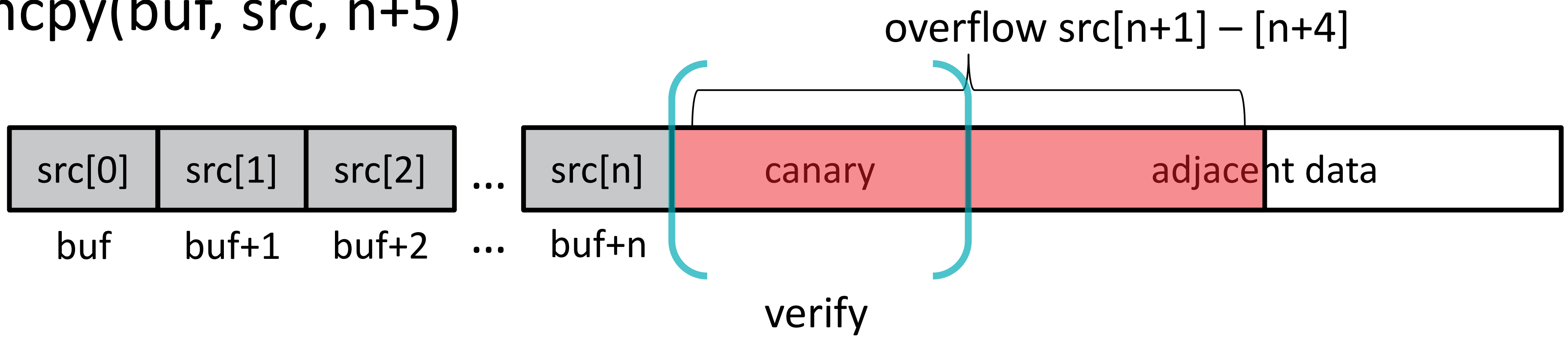


BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

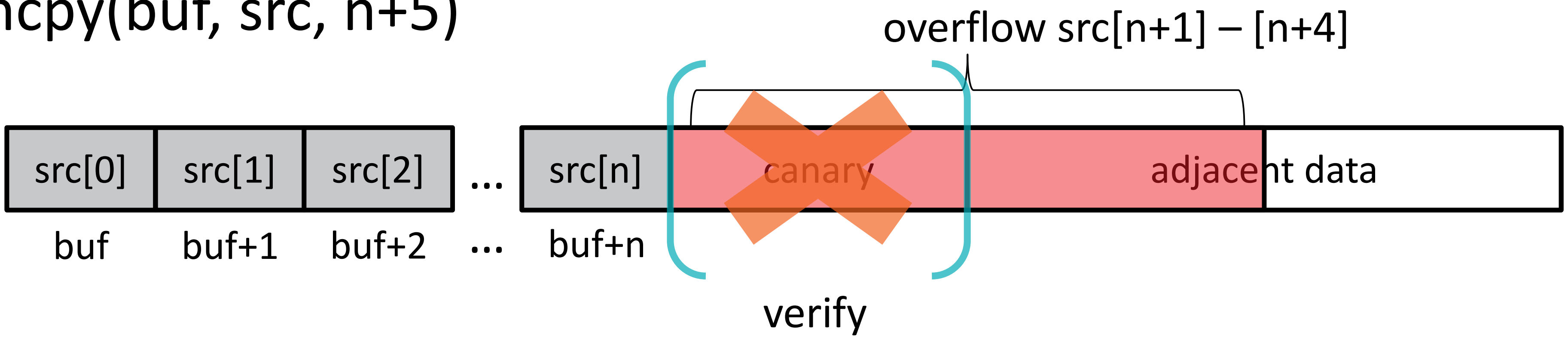


BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

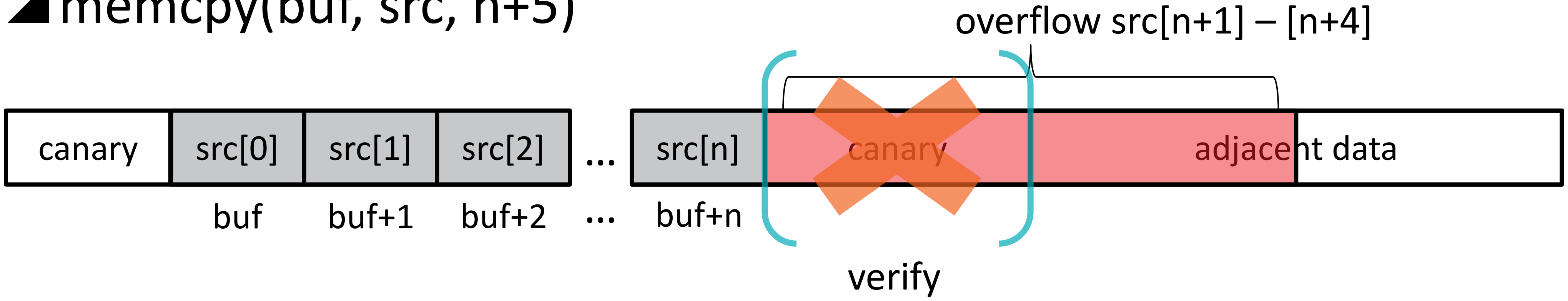


- Absence of known canary values indicates an invalid write.

BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.
- buf[n+1]
- memcpy(buf, src, n+5)



- Absence of known canary values indicates an invalid write.
- Can find underflow as well!

GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

AMD

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection

- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU device types from multiple vendors

- Low runtime overhead
  - 9.7% average overhead

LAUNCHING AN OPENCL™ KERNEL

LAUNCHING AN OPENCL™ KERNEL

Buffer Create

Buffer

LAUNCHING AN OPENCL™ KERNEL

Set Arguments

LAUNCHING AN OPENCL™ KERNEL

Set Arguments

LAUNCHING AN OPENCL™ KERNEL

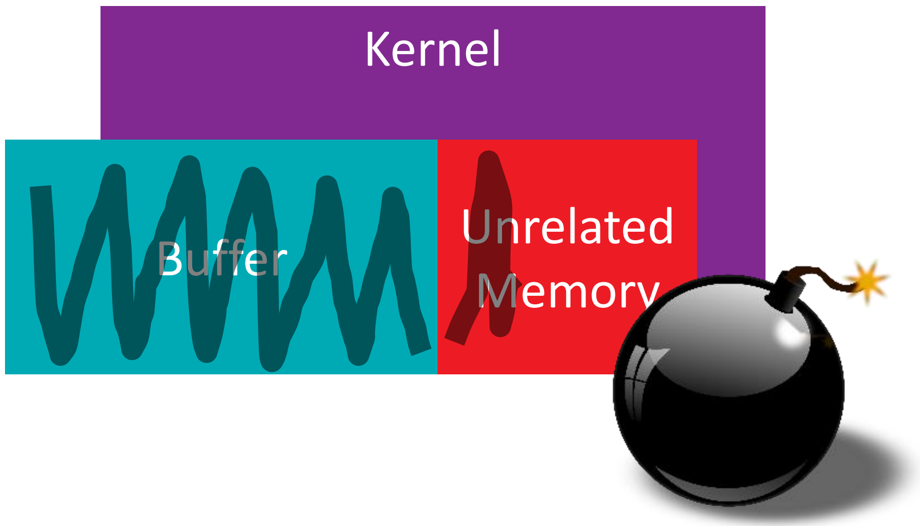
Launch Kernel

LAUNCHING AN OPENCL™ KERNEL

Launch Kernel

LAUNCHING AN OPENCL™ KERNEL

Launch Kernel



LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

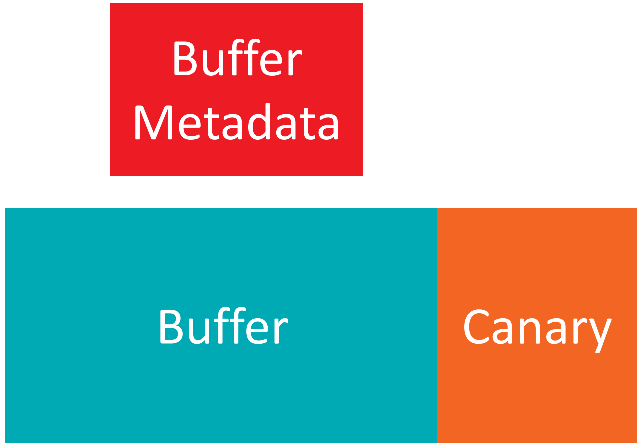
Buffer Create

LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Buffer Create

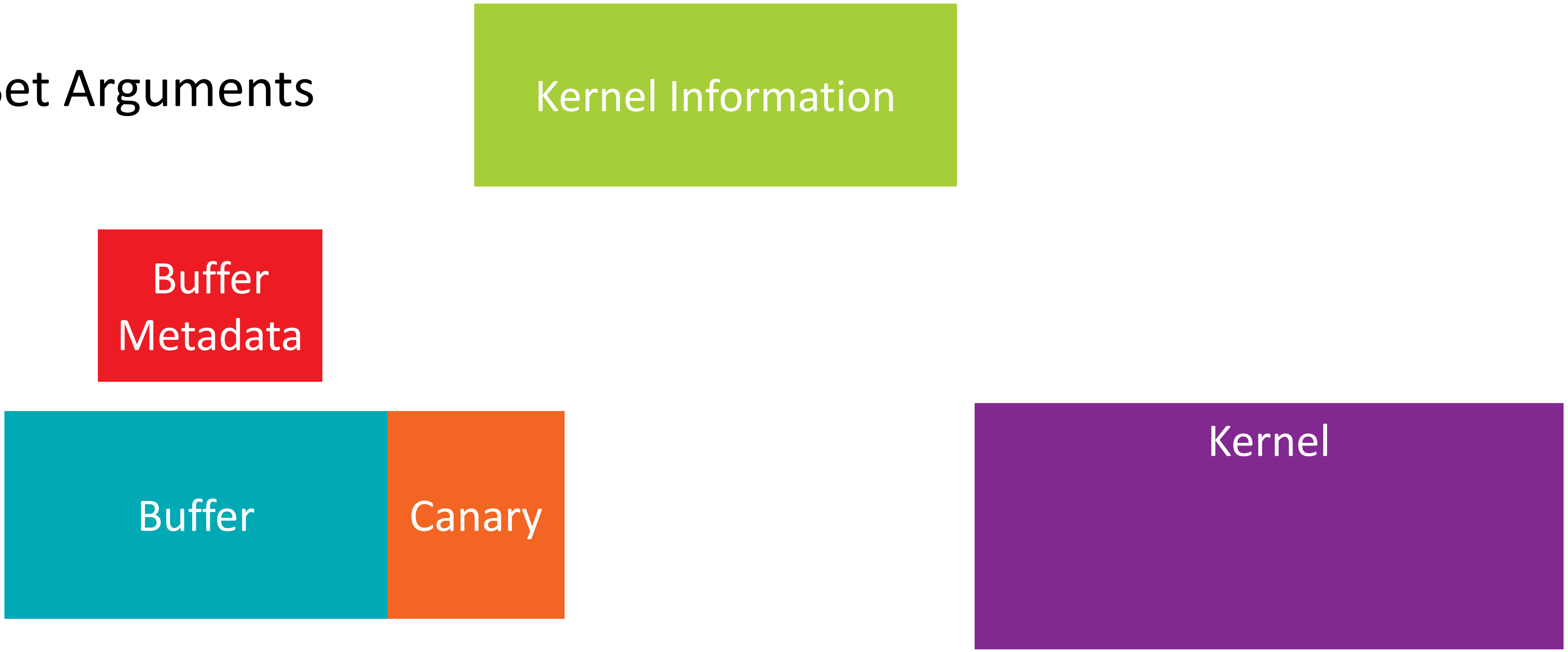
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Buffer Create



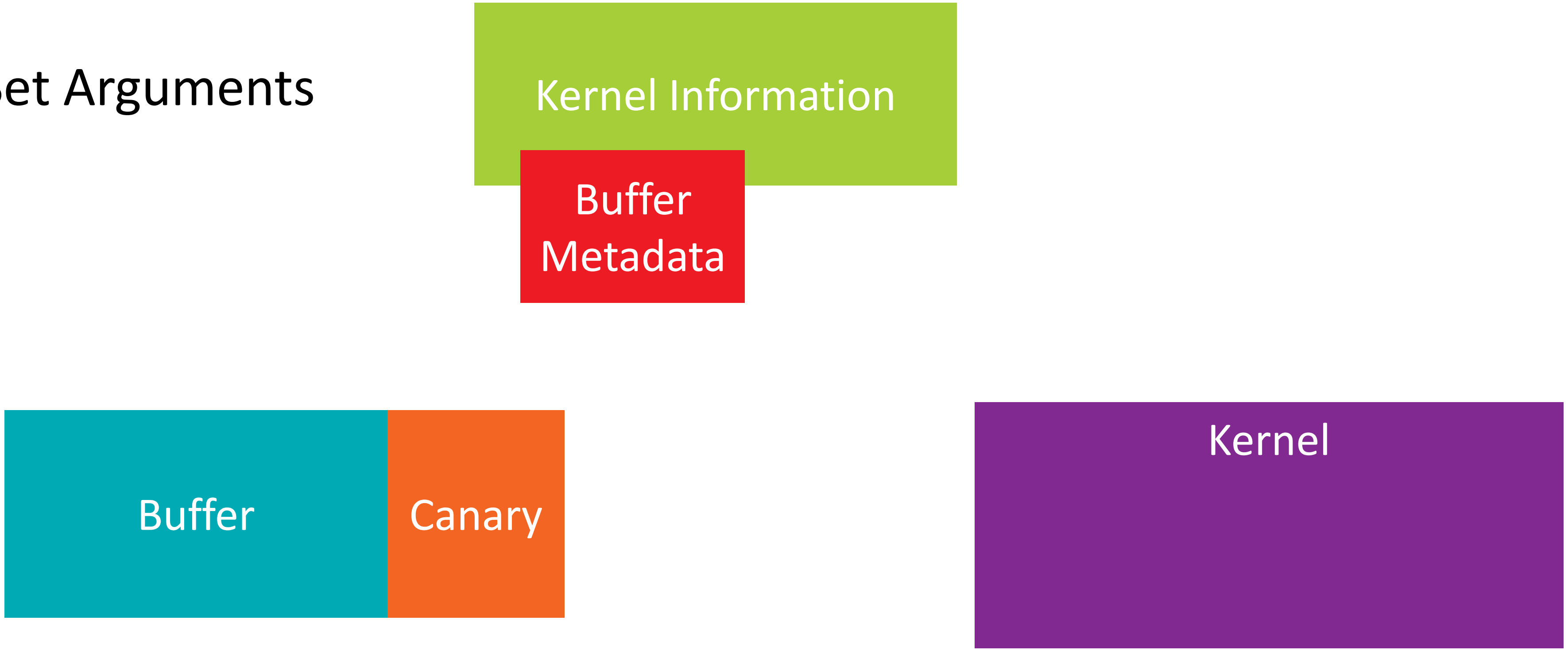
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Set Arguments



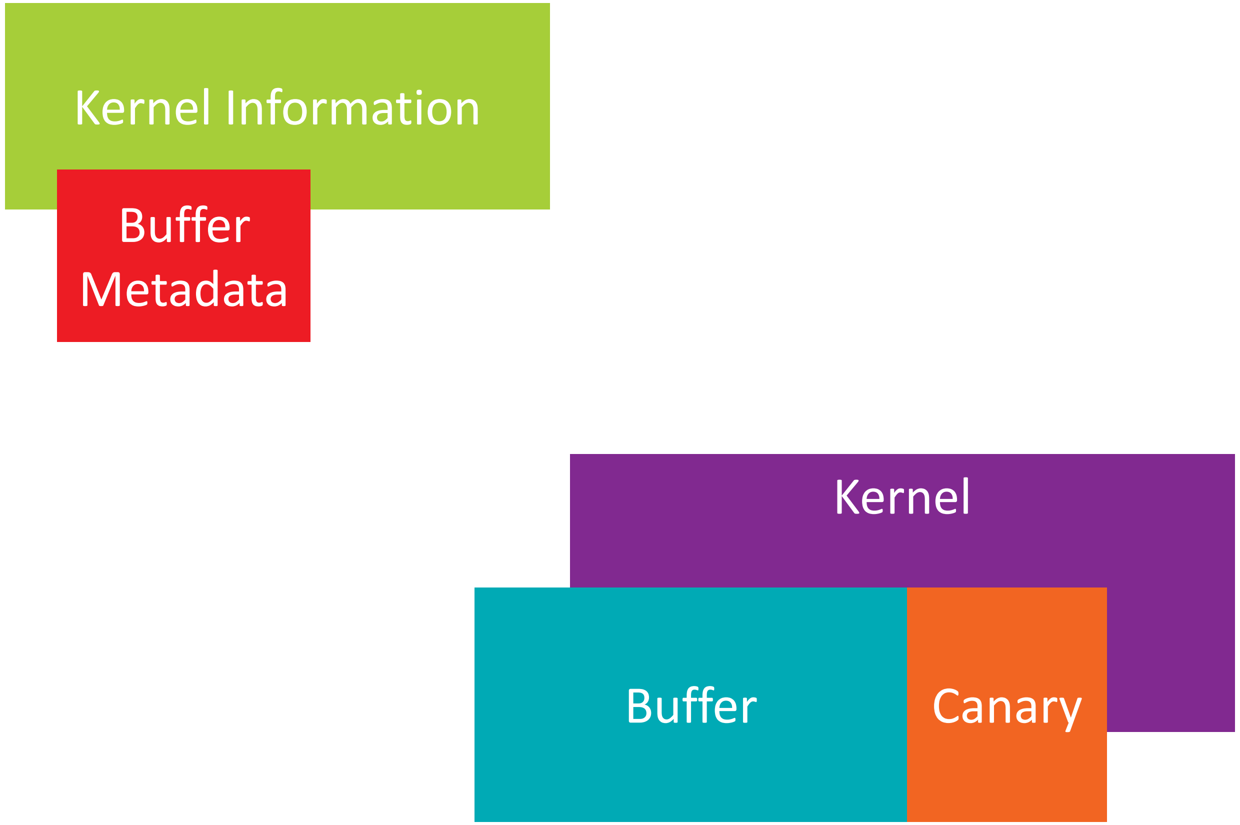
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Set Arguments

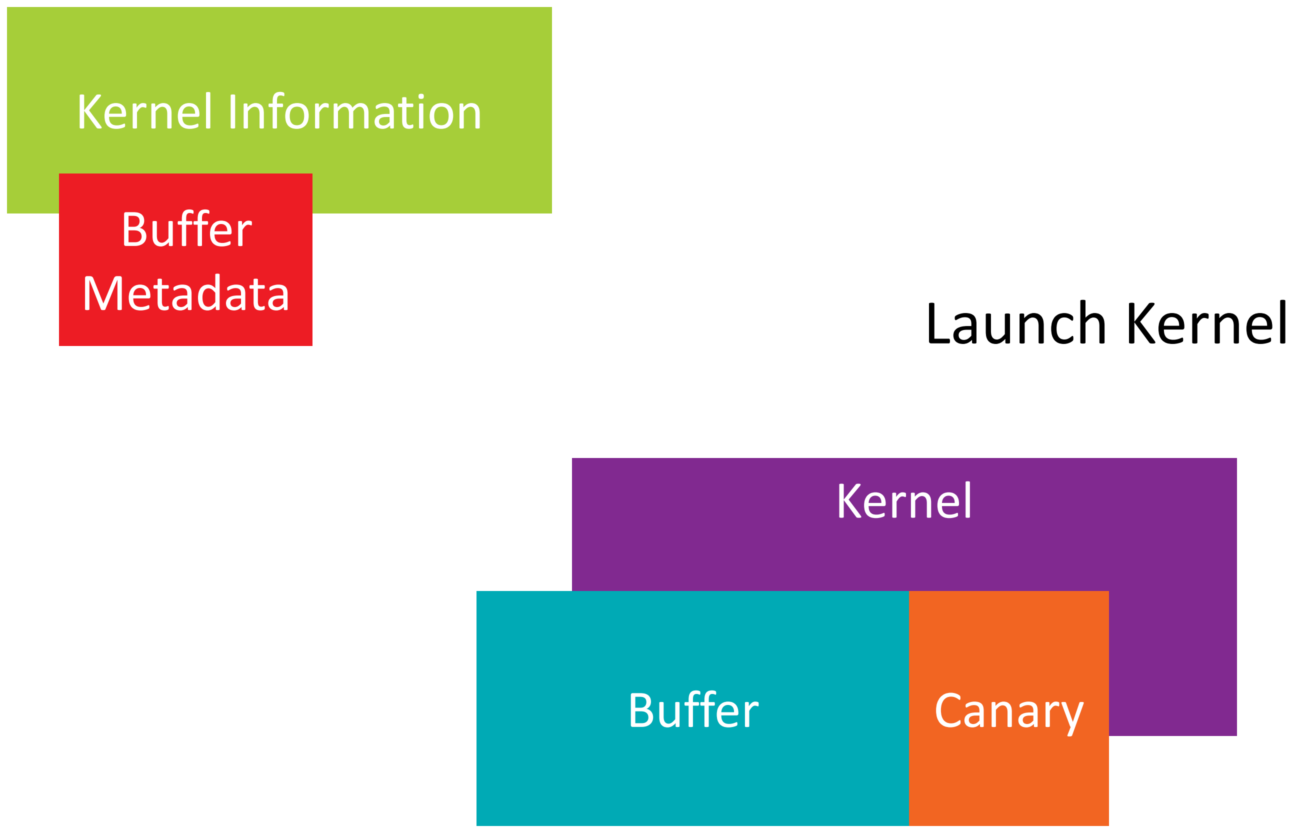


LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

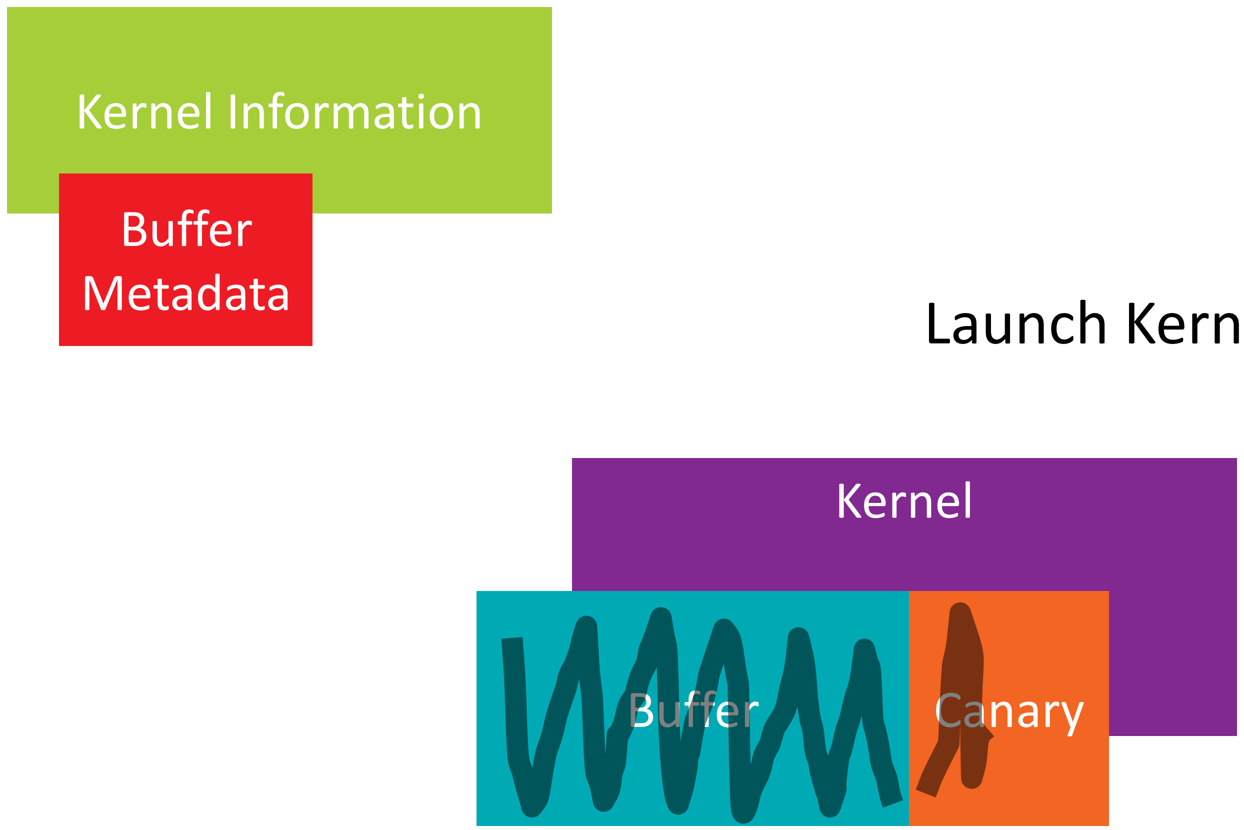
Set Arguments



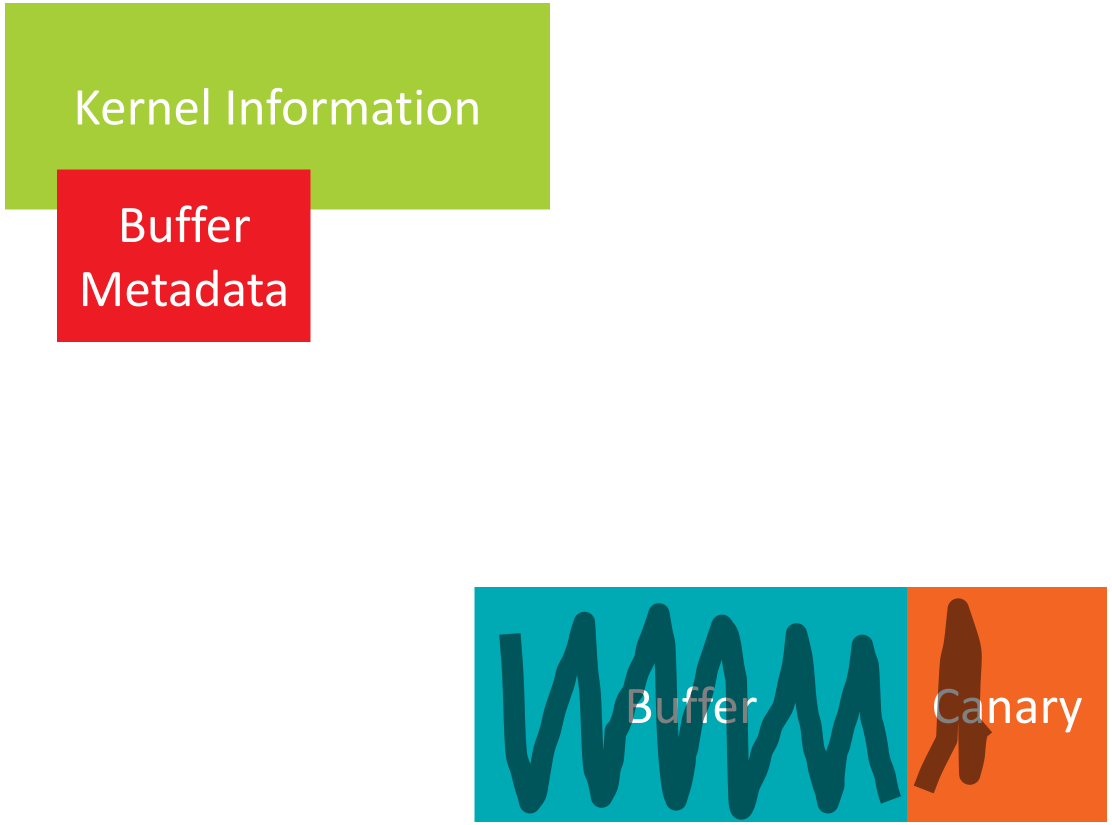
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR



LAUNCHING AN OPENCL™ KERNEL WITH clARMOR



LAUNCHING AN OPENCL™ KERNEL WITH clARMOR



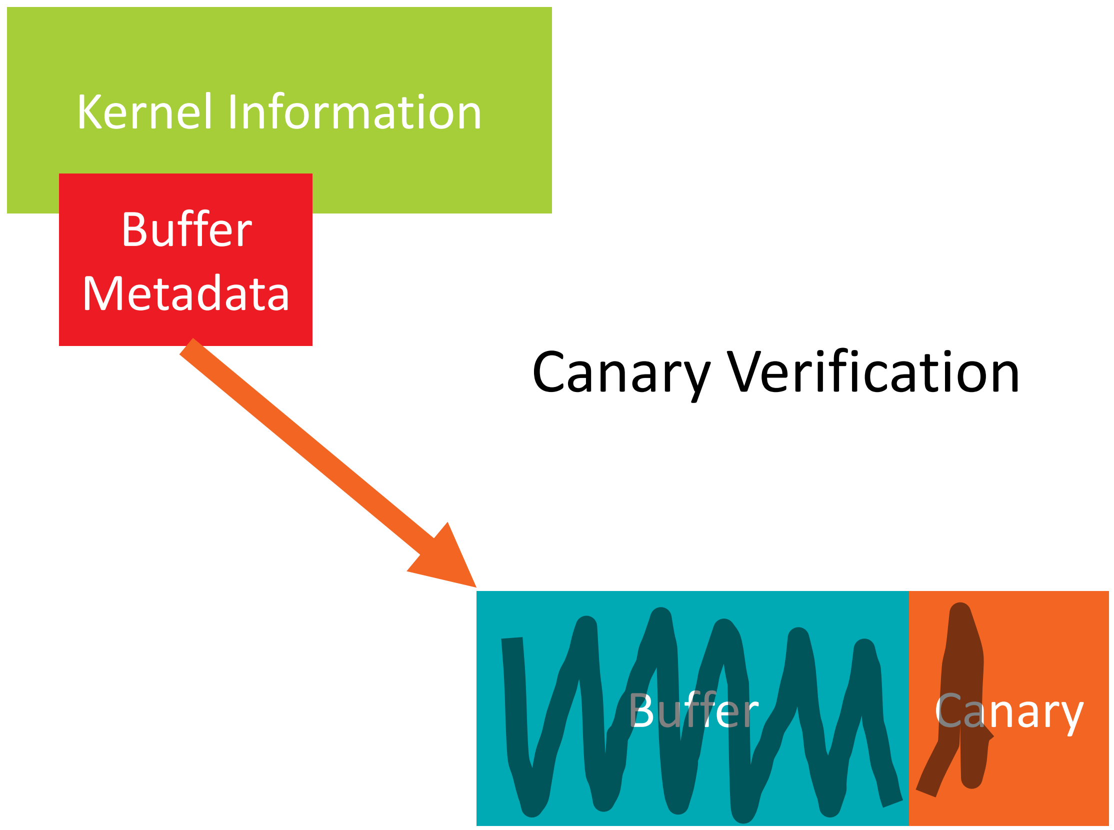
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Kernel Information

Buffer Metadata

Canary Verification

LAUNCHING AN OPENCL™ KERNEL WITH clARMOR



LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Kernel Information

Buffer Metadata

Canary Verification

LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Kernel Information

Buffer Metadata

Canary Verification

LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Kernel Information

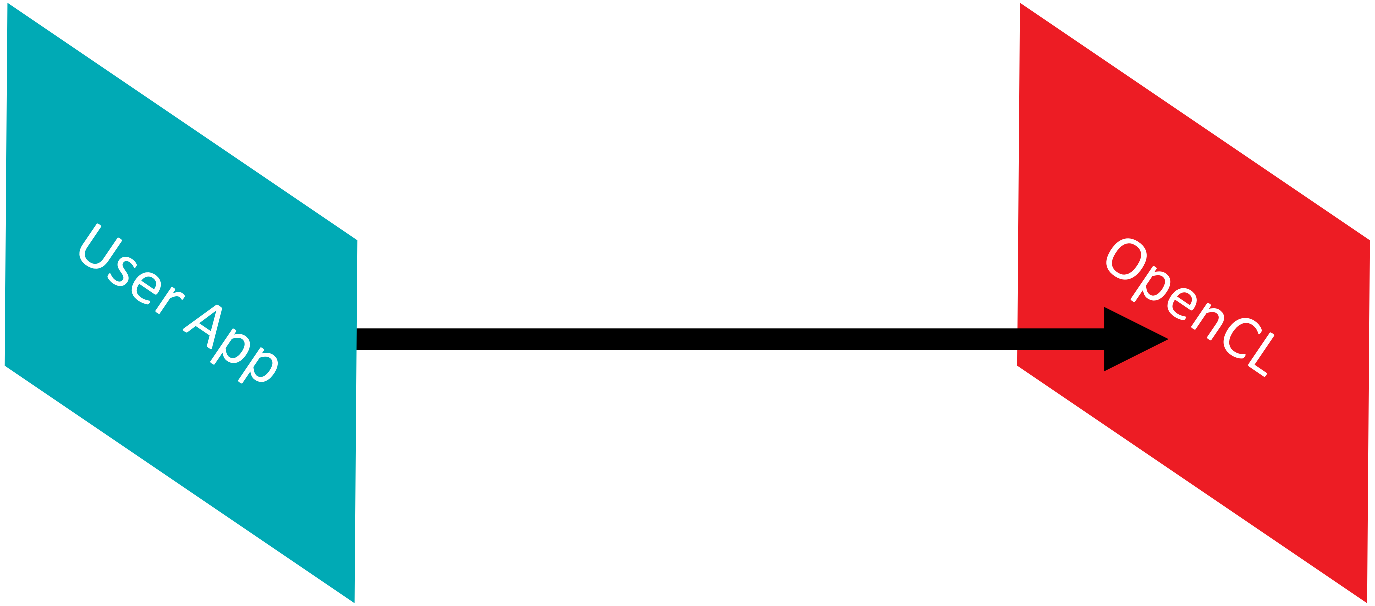
Buffer Metadata

Canary Verification

WRAPPING OPENCL™

clARMOR BETWEEN YOUR APPLICATION AND OPENCL

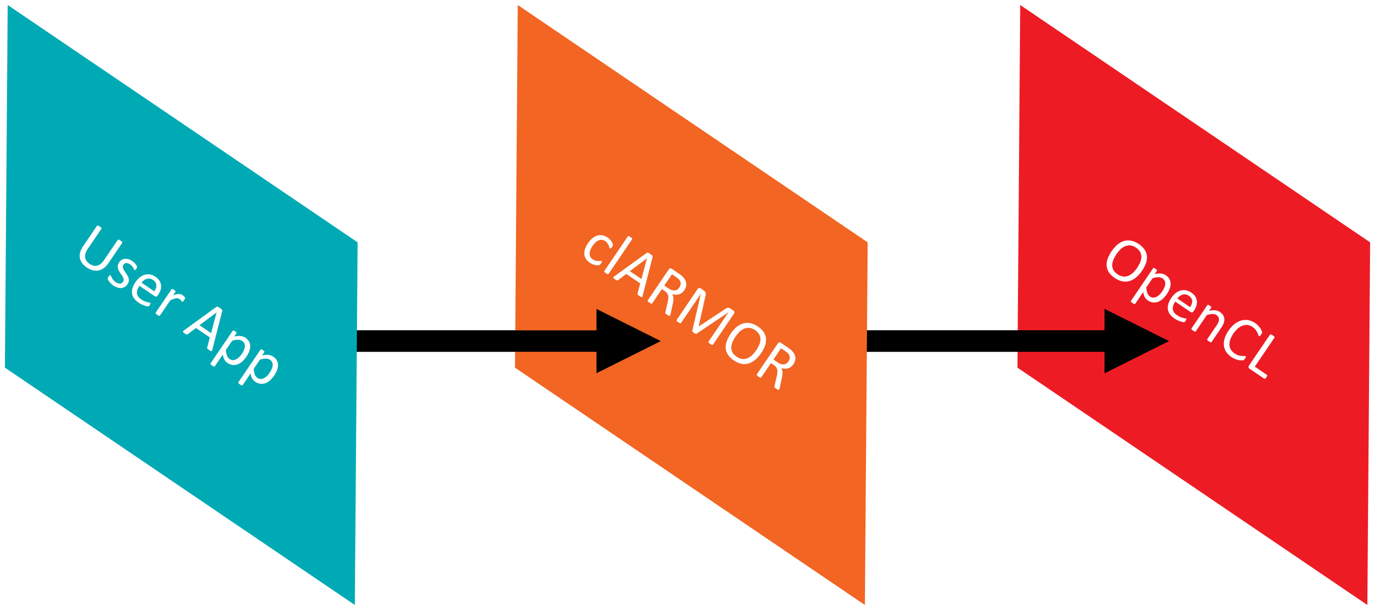
- clARMOR is a Linux® library that uses LD_PRELOAD to wrap OpenCL™ library calls
- Call Wrapping
  - Buffer, SVM, and Image creates
  - Argument setters
  - Kernel launches
  - Information functions



WRAPPING OPENCL™

clARMOR BETWEEN YOUR APPLICATION AND OPENCL

- clARMOR is a Linux® library that uses LD_PRELOAD to wrap OpenCL™ library calls
- Call Wrapping
  - Buffer, SVM, and Image creates
  - Argument setters
  - Kernel launches
  - Information functions



GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection
- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU device types from multiple vendors
- Low runtime overhead
  - 9.7% average overhead

TEST SETUP

HARDWARE SPECIFICATIONS AND BENCHMARKS SUITES

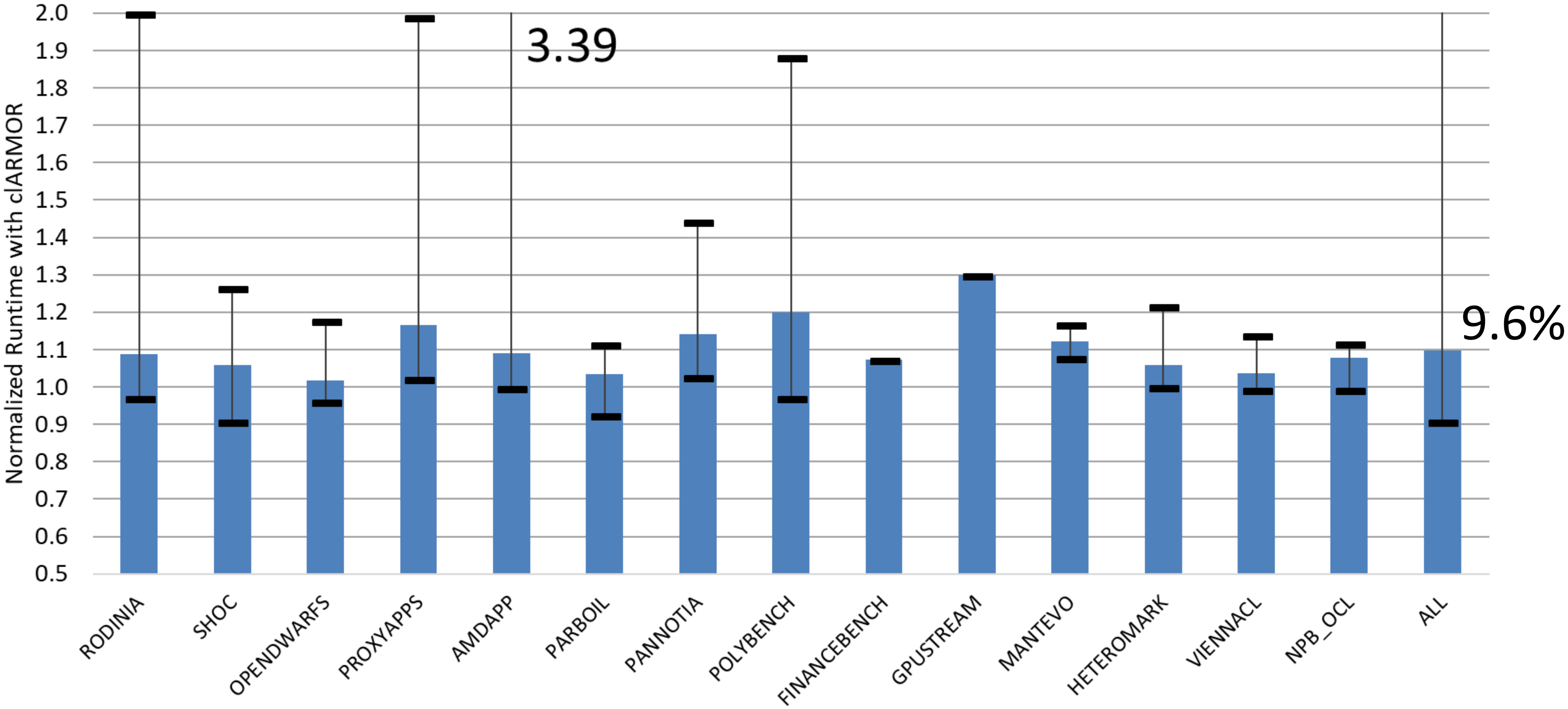
- AMD Ryzen™ 7 1800X CPU
- AMD Radeon™ Vega Frontier Edition discrete GPU
- ROCm 1.7
- 143 benchmarks in 14 benchmark suites

Suite	Num Benchmarks
AMDAPP	33
FINANCEBENCH	1
GPUSTREAM	1
HETEROMARK	9
MANTEVO	2
NPB_OCL	8
OPENDWARFS	7
PANNOTIA	6
PARBOIL	8
POLYBENCH	21
PROXYAPPS	6
RODINIA	19
SHOC	14
VIENNACL	8

PERFORMANCE EVALUATION

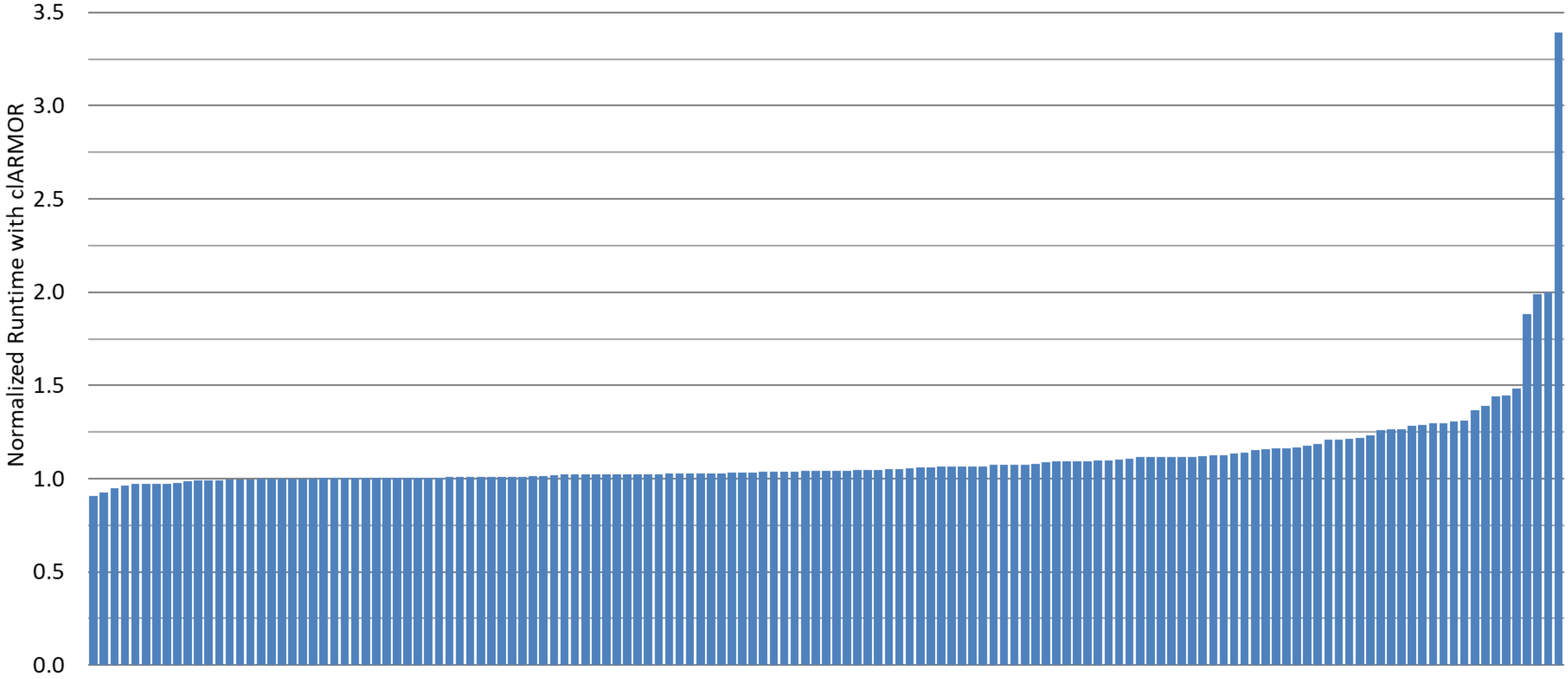
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better



PERFORMANCE EVALUATION

APPLICATION RUNTIME: WITH / WITHOUT TOOL



EXAMPLE USAGE

BAD_CL_MEM TEST

```
bin/clarmor tests/bad_cl_mem/bad_cl_mem.exe
```

```
[redacted]:~/tools/clARMOR$ bin/clarmor tests/bad_cl_mem/bad_cl_mem.exe
clARMOR: Final command line to run: LD_PRELOAD='[redacted]/tools/clARMOR/bin/../lib/libclbuff
erwrapper.so.1.0' PATH='[redacted]/tools/clARMOR:[redacted]/bin:[redacted]/.local/bin:/opt/ro
cm/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/ga
mes:/snap/bin'  CLARMOR_LOG_PREFIX="clARMOR: "  CLARMOR_ERROR_EXITCODE=-1  tests/bad_cl_mem/b
ad_cl_mem.exe

clARMOR: Loaded CL_WRAPPER
Searching for platforms...
    Using platform: AMD Accelerated Parallel Processing
Searching for devices...
    Using device: gfx803


Running Bad cl_mem Test...
    Using buffer size: 1048566
Launching 262144 work items to write up to 262144 entries.
This will write 1048576 out of 1048566 bytes in the buffer.
clARMOR:
clARMOR: ATTENTION:
clARMOR: ************* Buffer overflow detected ***********
clARMOR: Kernel: test, Buffer: cl_mem_buffer
clARMOR:    Write Overflow 1 byte(s) past end.
clARMOR:
Done Running Bad cl_mem Test.
clARMOR: Done!
```

EXAMPLE USAGE

GOOD_CL_MEM TEST

```
[redacted]:~/tools/clARMOR$ bin/clarmor tests/good_cl_mem/good_cl_mem.exe
clARMOR: Final command line to run: LD_PRELOAD='[redacted]/tools/clARMOR/bin/../lib/libclbuf
ferwrapper.so.1.0' PATH='[redacted]/tools/clARMOR:[redacted]/bin:[redacted].local/bin:/opt/
rocm/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local
/games:/snap/bin'  CLARMOR_LOG_PREFIX="clARMOR: "  CLARMOR_ERROR_EXITCODE=-1  tests/good_cl_
mem/good_cl_mem.exe

clARMOR: Loaded CL_WRAPPER
Searching for platforms...
    Using platform: AMD Accelerated Parallel Processing
Searching for devices...
    Using device: gfx803


Running Good cl_mem Test...
    Using buffer size: 1048576
Launching 262144 work items to write up to 262144 entries.
This will write 1048576 out of 1048576 bytes in the buffer.
Done Running Good cl_mem Test.
clARMOR: Done!
```

BONUS DETAILS

- What do the wrapped OpenCL™ library calls have to do?
  - Buffer and Image creates
  - Argument setters
  - Kernel launches
  - Information functions
- What are we doing to make the check faster?

WRAPPING THE OPENCL™ API

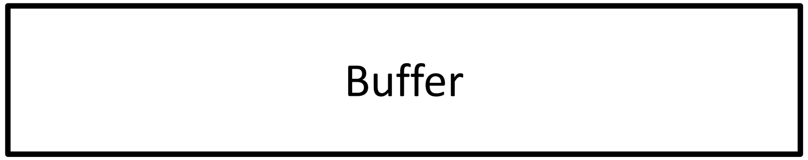
BUFFER AND IMAGE CREATION

- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary

WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

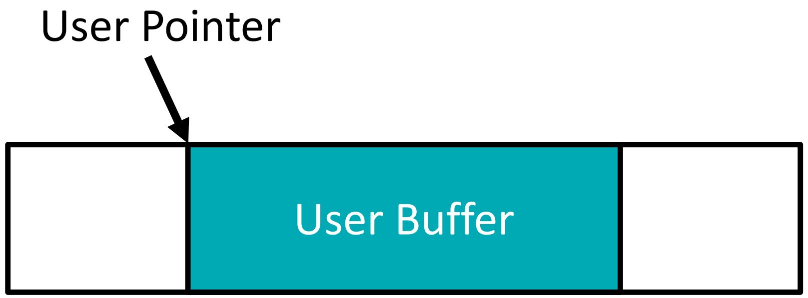
- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary



WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

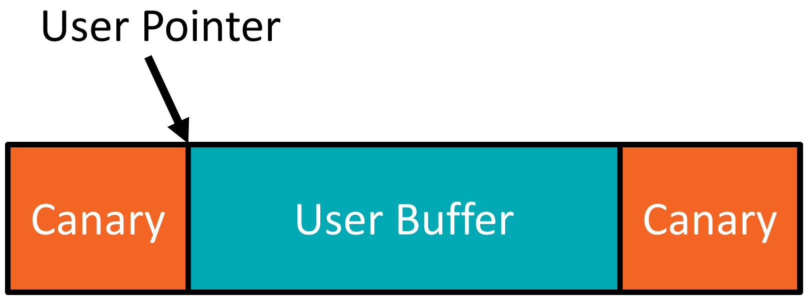
- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary



WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

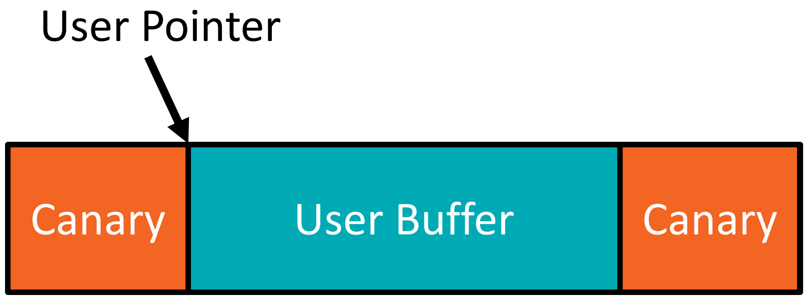
- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary



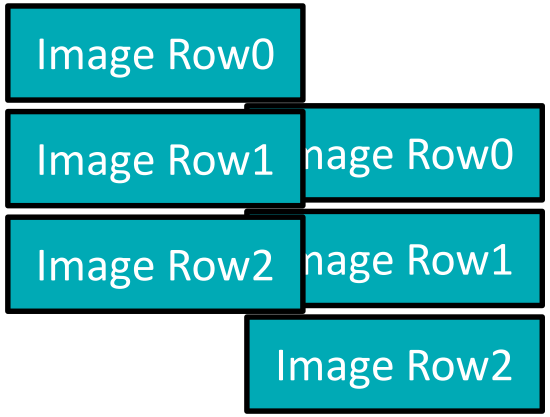
WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary



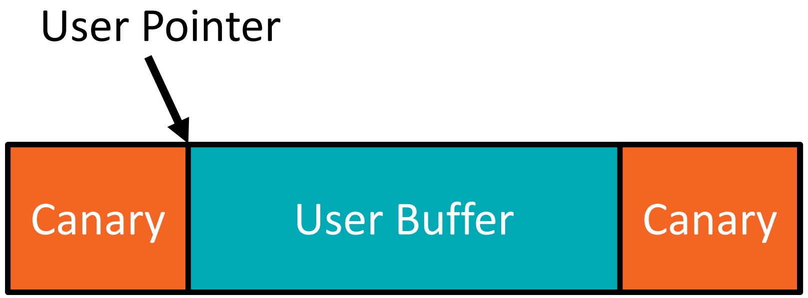
- Image Creation
  - Calls to ***clCreateImage***, ***clCreateImage2D***, or ***clCreateImage3D***
  - Potential for multi dimensional overflow
  - Add canary regions to each dimension



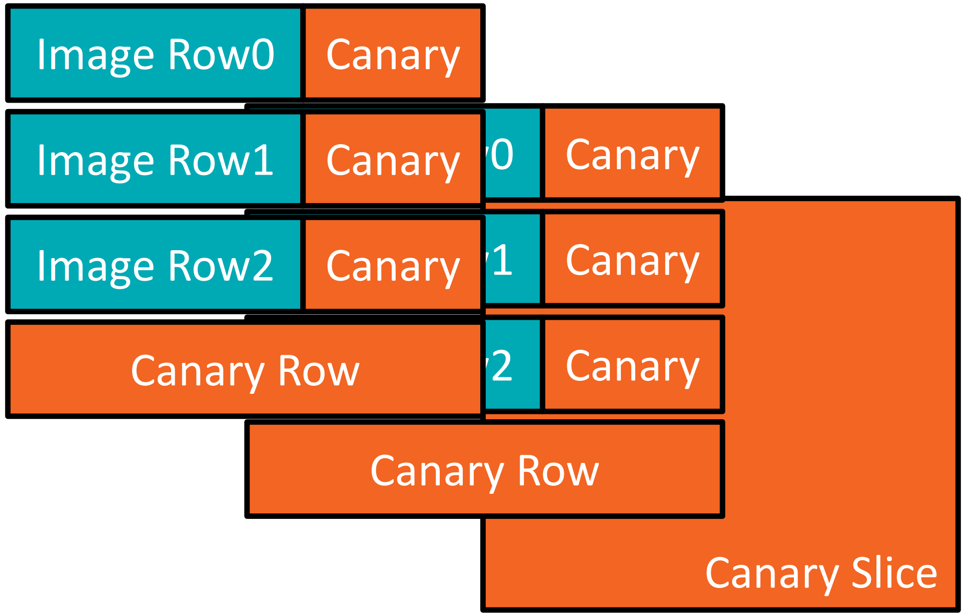
WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary



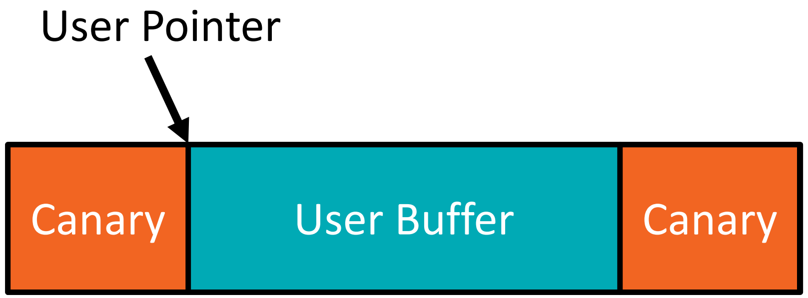
- Image Creation
  - Calls to ***clCreateImage***, ***clCreateImage2D***, or ***clCreateImage3D***
  - Potential for multi dimensional overflow
  - Add canary regions to each dimension



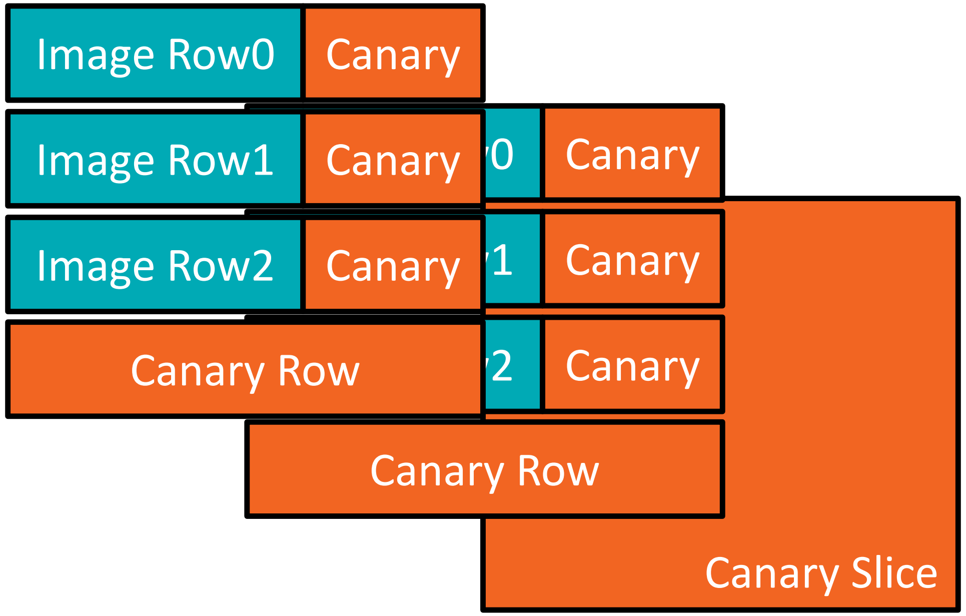
WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

- Buffer Creation
  - Calls to ***clCreateBuffer*** or ***clSVMAlloc***
    - Allocate buffer
    - Create sub buffer for user
    - Surround with canary



- Image Creation
  - Calls to ***clCreateImage***, ***clCreateImage2D***, or ***clCreateImage3D***
  - Potential for multi dimensional overflow
  - Add canary regions to each dimension



- Annotations for location of canaries, etc.

WRAPPING THE OPENCL™ API

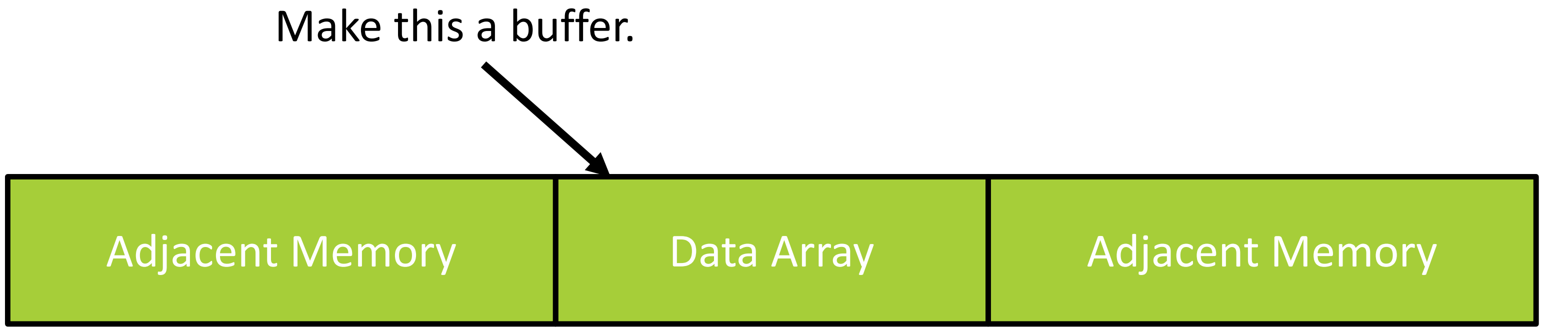
BUFFER CREATION FROM EXISTING ALLOCATIONS

- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)

WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

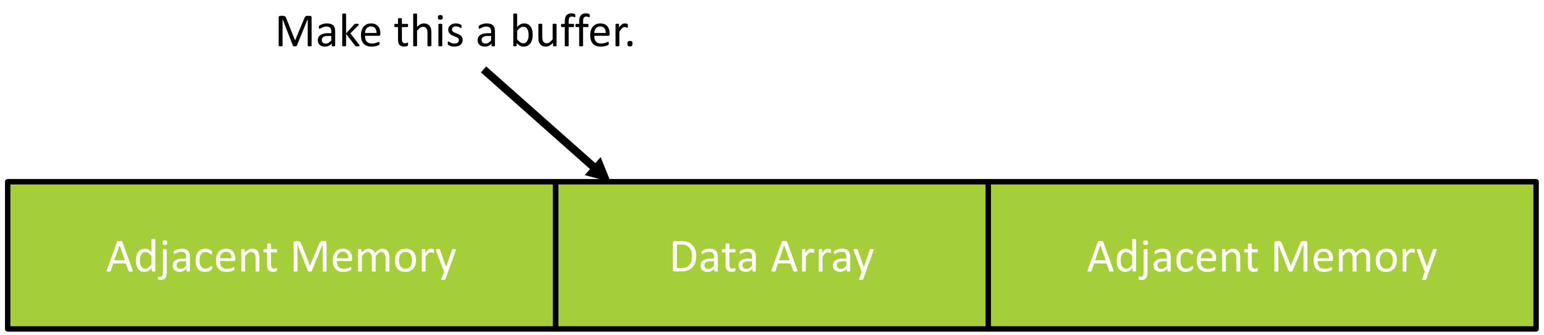
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

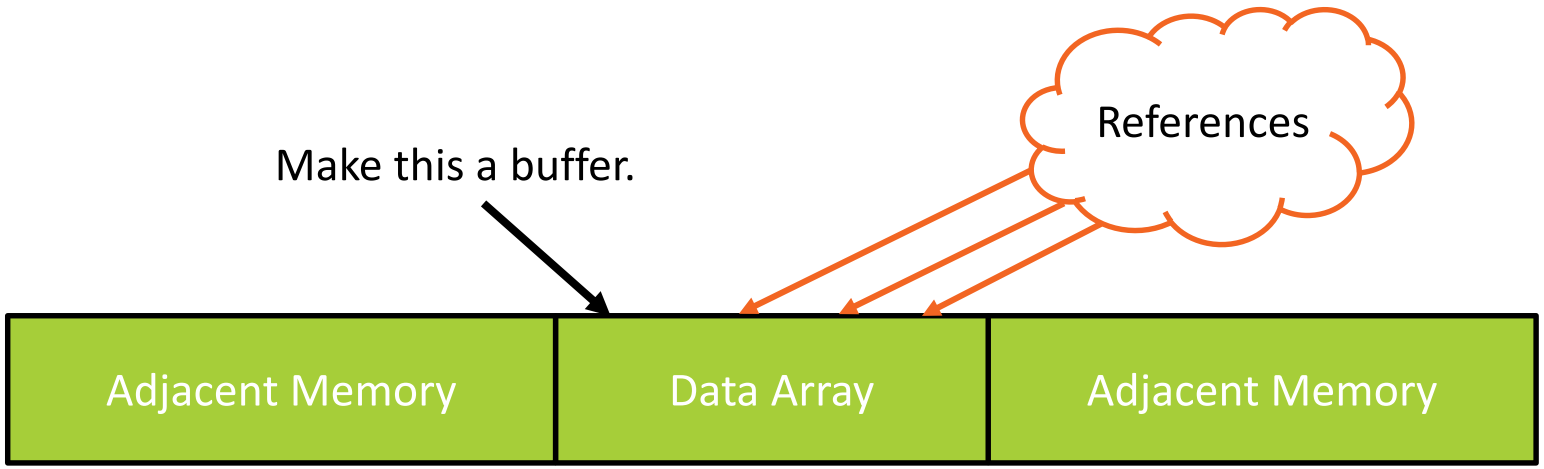
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

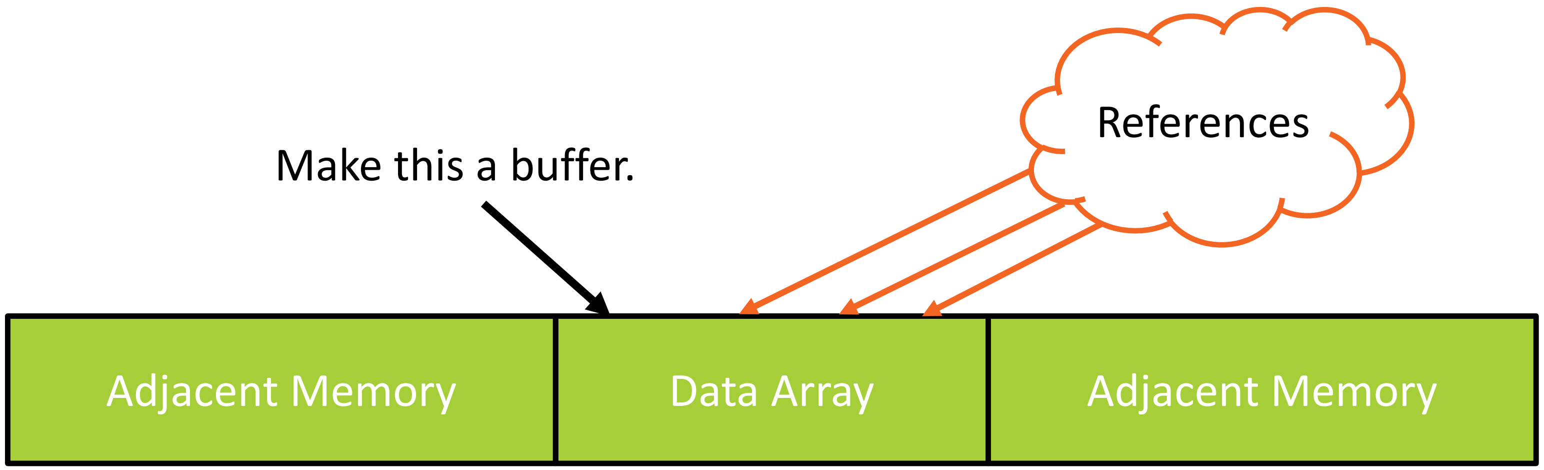
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

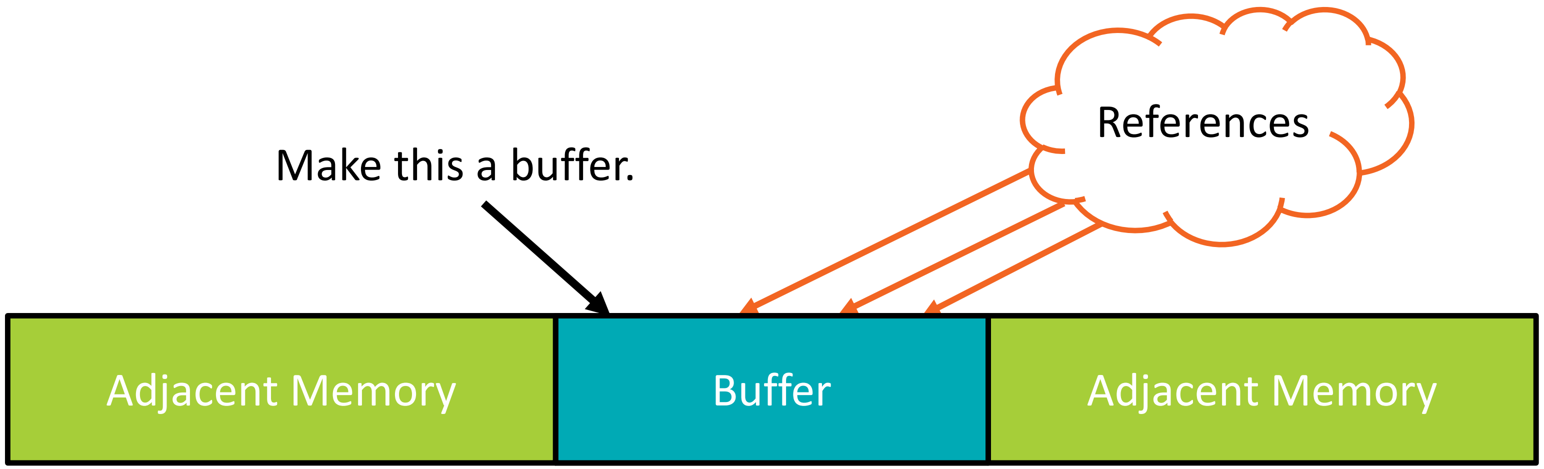
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer
  - Cannot move buffer



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

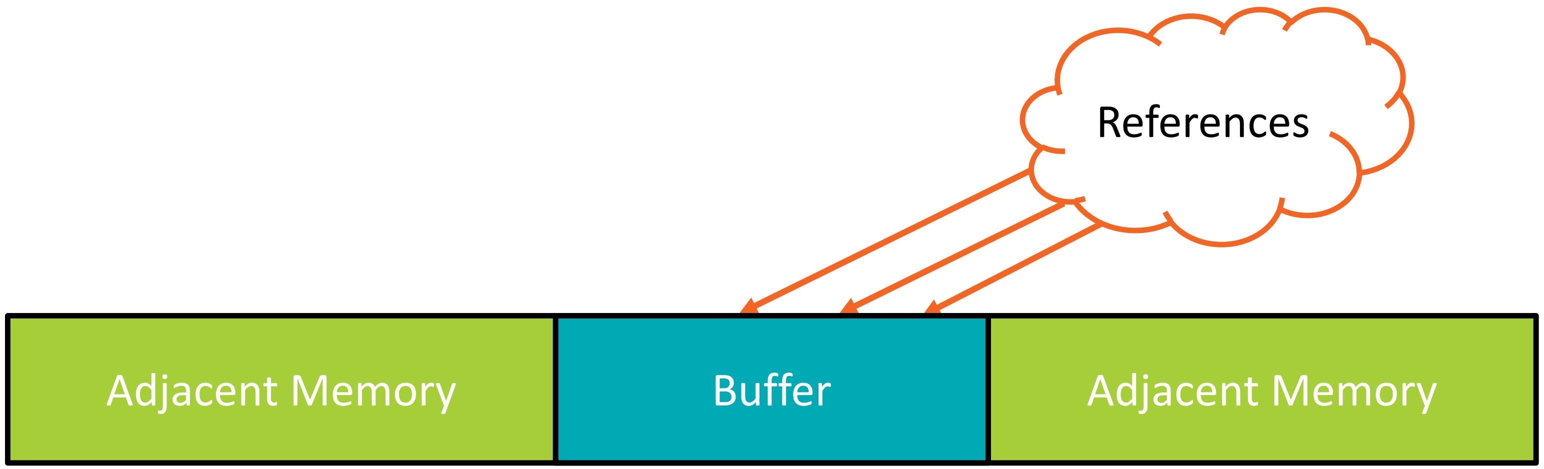
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer
  - Cannot move buffer



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

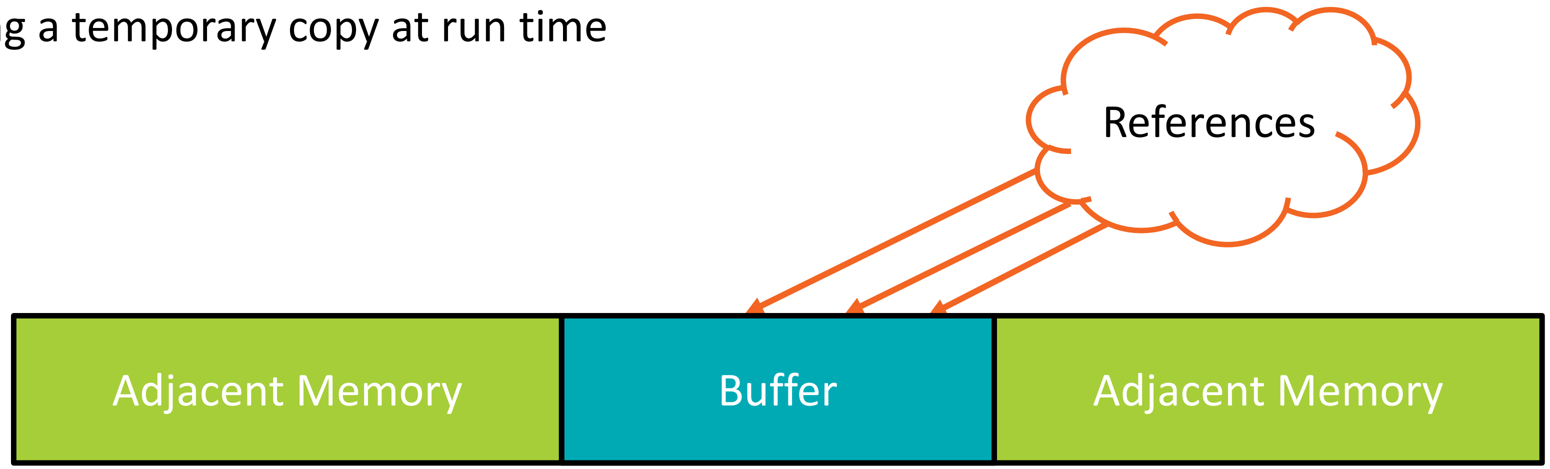
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer
  - Cannot move buffer



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

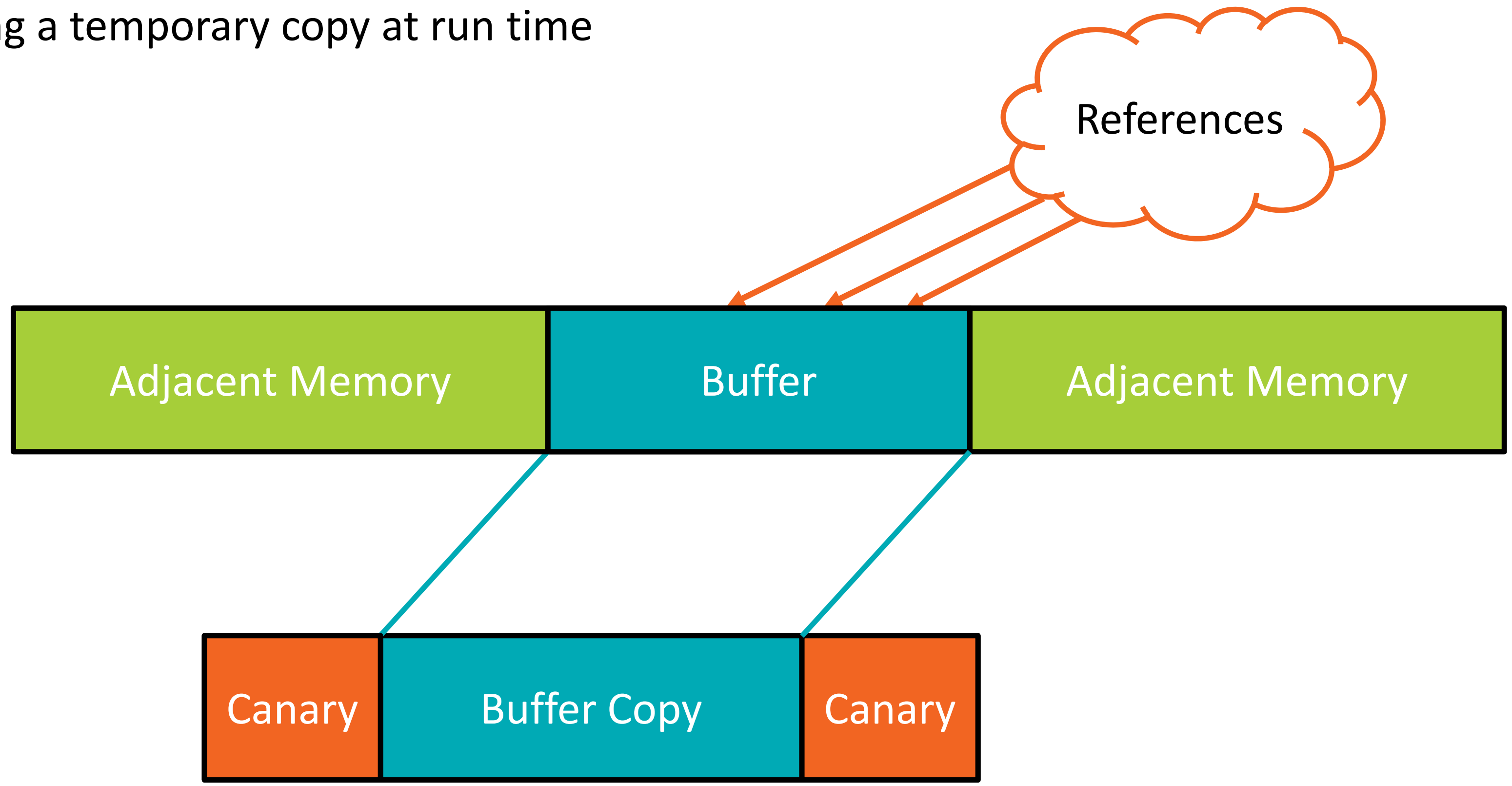
- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer
  - Cannot move buffer
  - Solution using a temporary copy at run time



WRAPPING THE OPENCL™ API

BUFFER CREATION FROM EXISTING ALLOCATIONS

- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer
  - Cannot move buffer
  - Solution using a temporary copy at run time



WRAPPING THE OPENCL™ API

SET ARGUMENTS

- clARMOR needs to know which buffers/images to check for overflows
- Kernel information object
  - map kernel argument number to buffer information
- Update on call to ***clSetKernelArg*** or ***clSetKernelArgSVMPointer***

WRAPPING THE OPENCL™ API

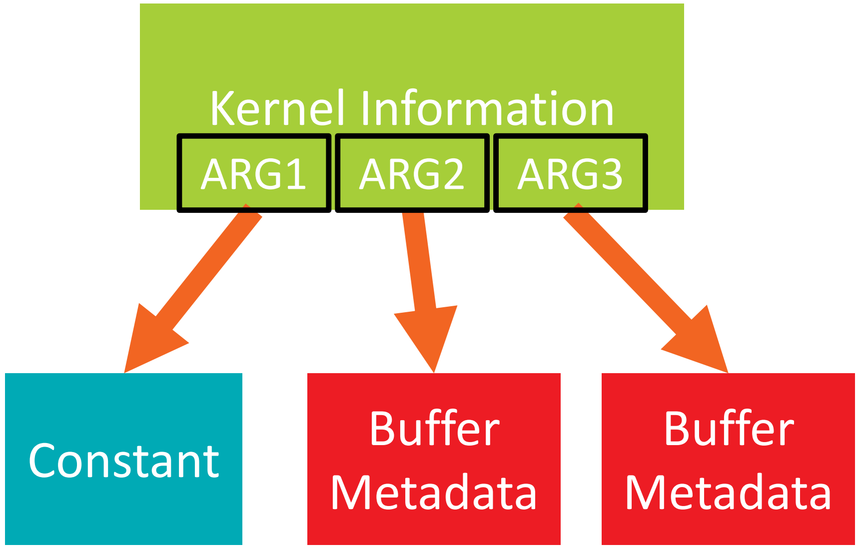
SET ARGUMENTS

- clARMOR needs to know which buffers/images to check for overflows
- Kernel information object
  - map kernel argument number to buffer information
- Update on call to ***clSetKernelArg*** or ***clSetKernelArgSVMPointer***

WRAPPING THE OPENCL™ API

SET ARGUMENTS

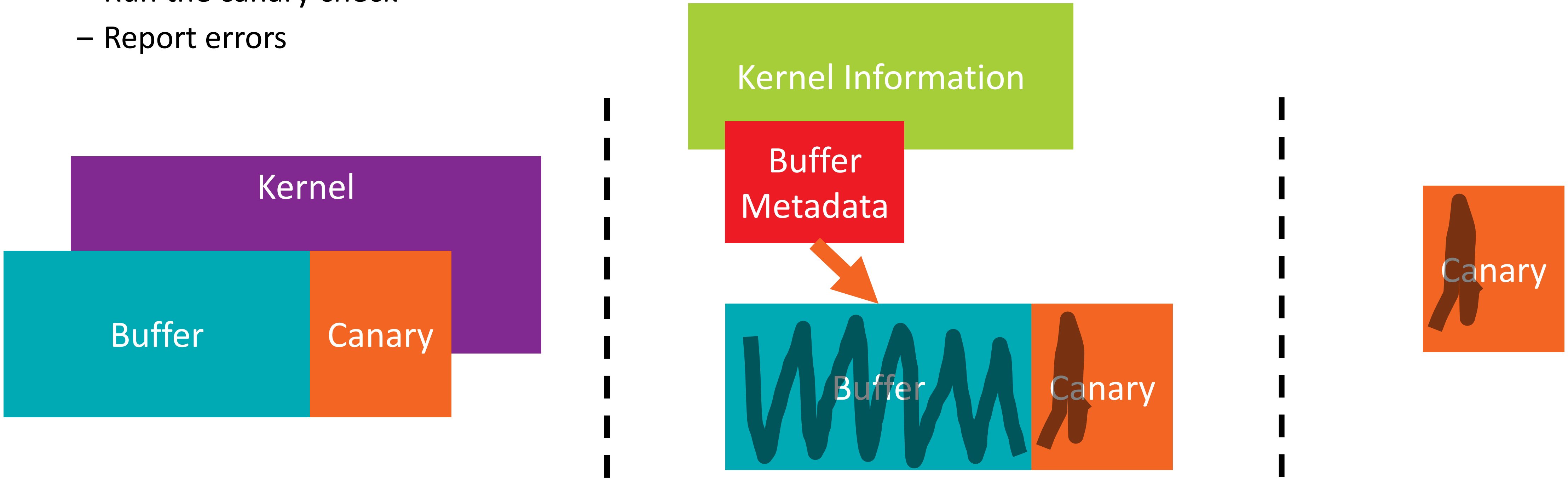
- clARMOR needs to know which buffers/images to check for overflows
- Kernel information object
  - map kernel argument number to buffer information
- Update on call to ***clSetKernelArg*** or ***clSetKernelArgSVMPointer***



WRAPPING THE OPENCL™ API

KERNEL LAUNCH

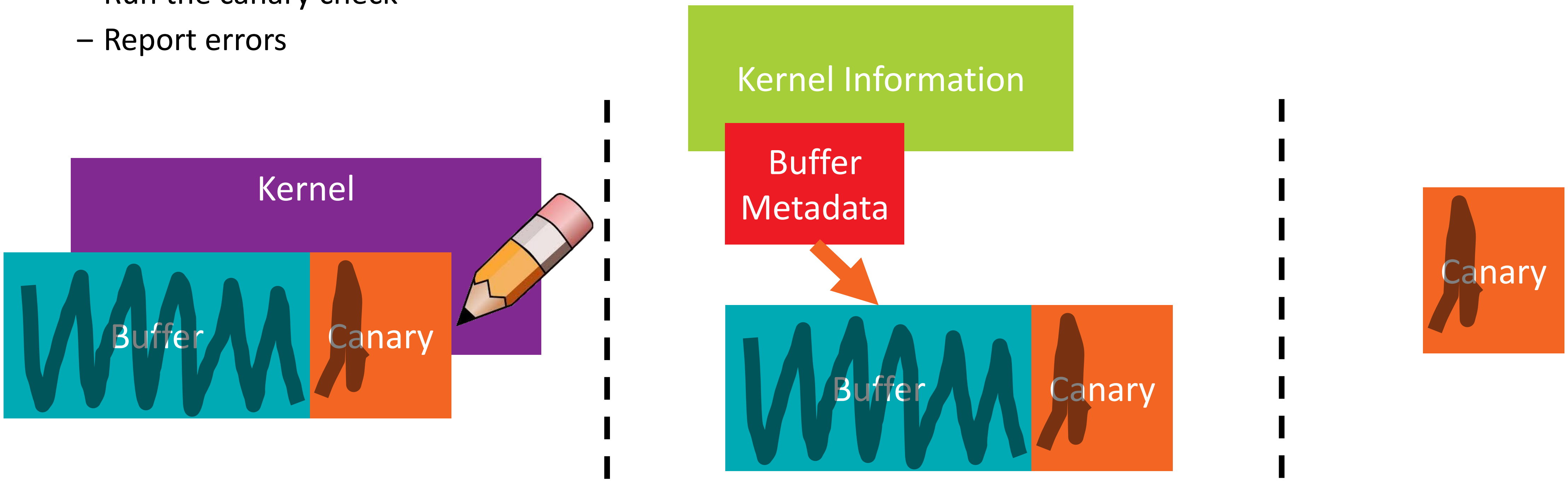
- Do the work of detecting buffer overflows
- On call to ***clEnqueueNDRangeKernel***
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors



WRAPPING THE OPENCL™ API

KERNEL LAUNCH

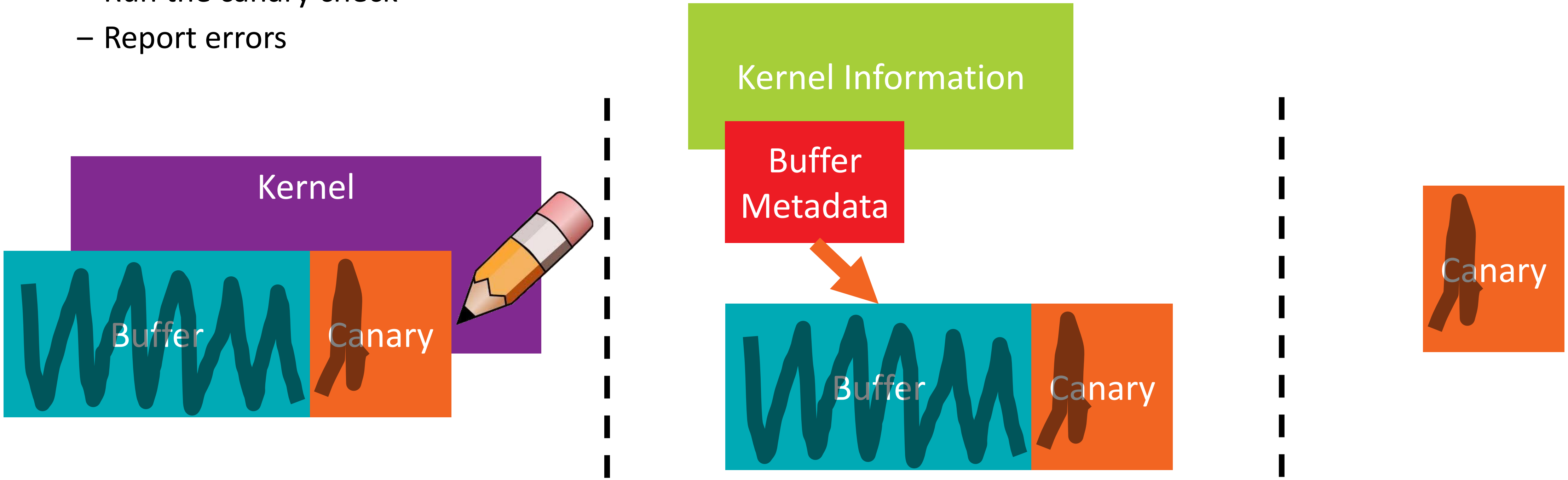
- Do the work of detecting buffer overflows
- On call to ***clEnqueueNDRangeKernel***
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors



WRAPPING THE OPENCL™ API

KERNEL LAUNCH

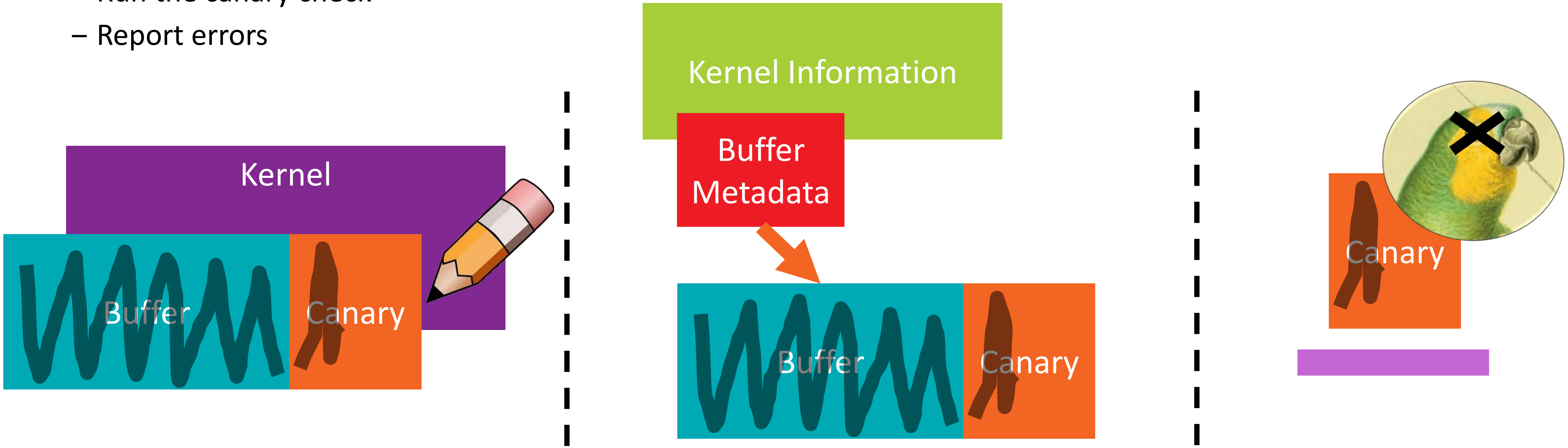
- Do the work of detecting buffer overflows
- On call to ***clEnqueueNDRangeKernel***
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors



WRAPPING THE OPENCL™ API

KERNEL LAUNCH

- Do the work of detecting buffer overflows
- On call to ***clEnqueueNDRangeKernel***
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors



WRAPPING THE OPENCL™ API

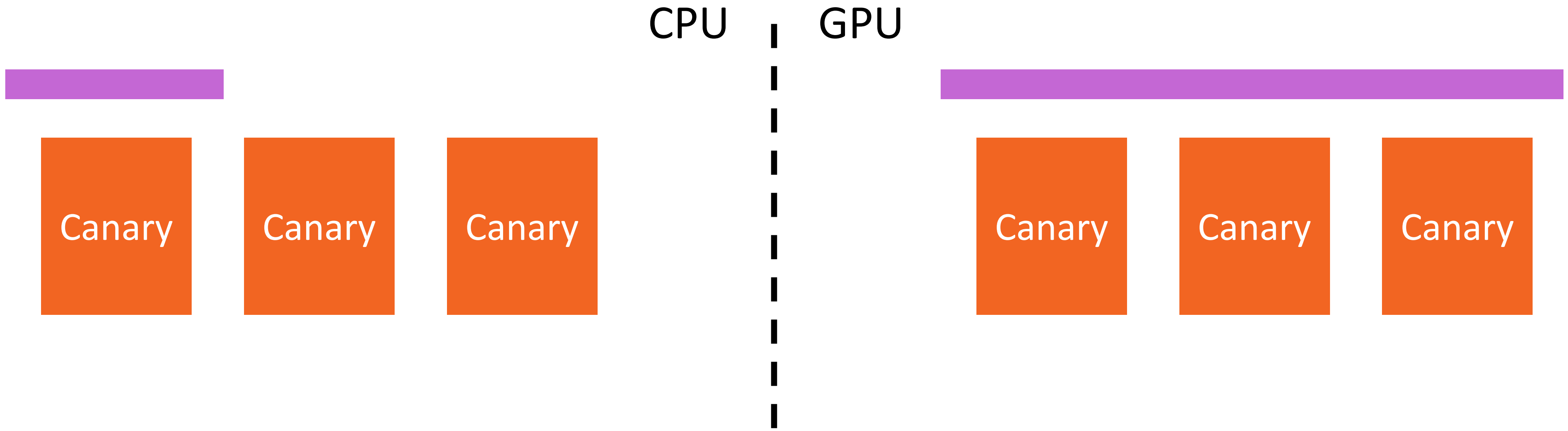
GETTERS AND SETTERS

- GetMemObjectInfo, GetImageInfo
  - Reserve space for canaries
- Enqueue Functions
  - Read / Write / Fill / Copy
  - Buffer / BufferRect / Image

ACCELERATION

SELECTING A DEVICE FOR PERFORMING CANARY VERIFICATION

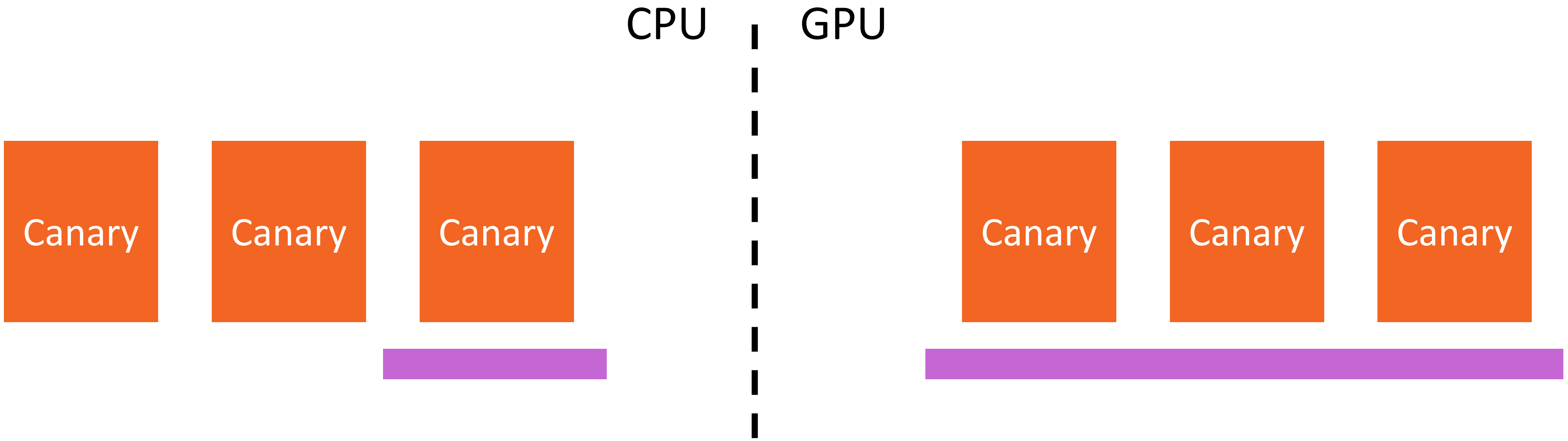
- CPU is faster
  - small / few canary regions (latency advantage)
- GPU is faster
  - large / many canary regions (throughput advantage with embarrassingly parallel workload)
  - reduced transfers over PCIe® by keeping on GPU



ACCELERATION

SELECTING A DEVICE FOR PERFORMING CANARY VERIFICATION

- CPU is faster
  - small / few canary regions (latency advantage)
- GPU is faster
  - large / many canary regions (throughput advantage with embarrassingly parallel workload)
  - reduced transfers over PCIe® by keeping on GPU



ACCELERATION

USING OPENCL™ EVENTS TO INCREASE THROUGHPUT

- Maximizing asynchrony
  - Event-based programming wherever possible
  - GPU check kernels enqueue behind work kernels and wait on completion
  - Evaluation of check kernel results is done with call-backs

synchronous | asynchronous

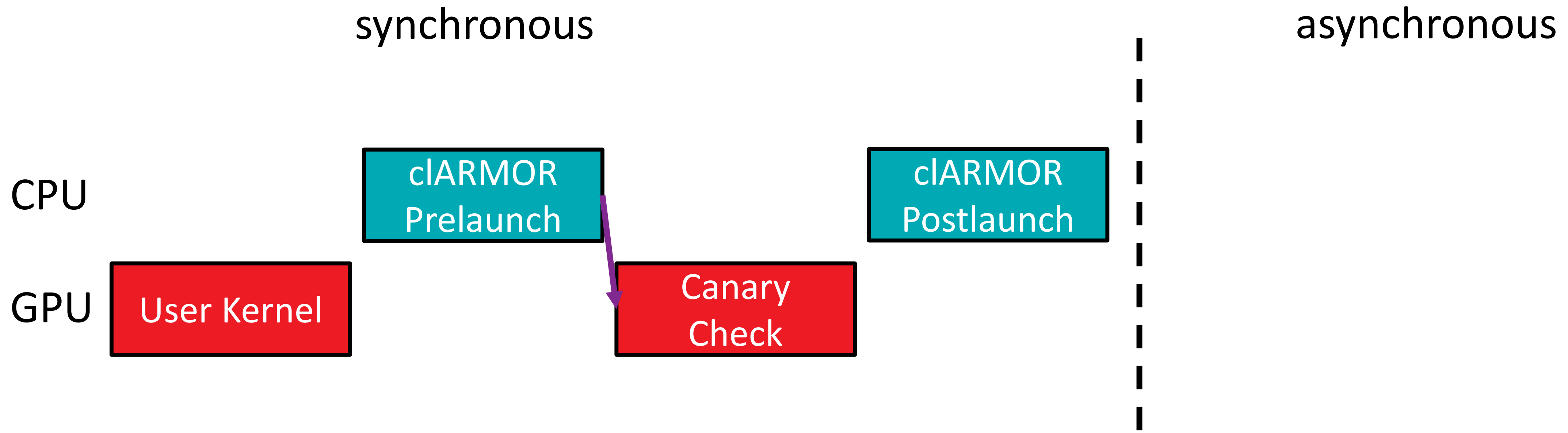
CPU

GPU

ACCELERATION

USING OPENCL™ EVENTS TO INCREASE THROUGHPUT

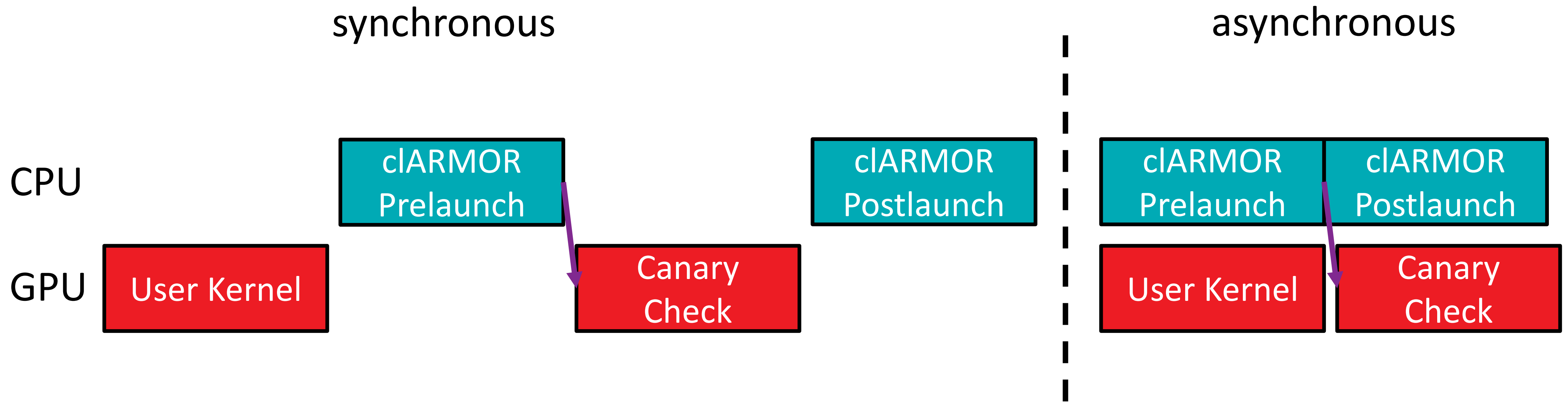
- Maximizing asynchrony
  - Event-based programming wherever possible
  - GPU check kernels enqueue behind work kernels and wait on completion
  - Evaluation of check kernel results is done with call-backs



ACCELERATION

USING OPENCL™ EVENTS TO INCREASE THROUGHPUT

- Maximizing asynchrony
  - Event-based programming wherever possible
  - GPU check kernels enqueue behind work kernels and wait on completion
  - Evaluation of check kernel results is done with call-backs



CONCLUSION

clARMOR IS READY FOR YOU TO USE

- Canary-based detection scheme finds GPU write overflows
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection
- Works for most OpenCL™ applications
  - Running on GPU or CPU, not vendor specific
- Near real-time detection
  - 9.7% average overhead

- Open Sourced
  - https://github.com/ROCm-Developer-Tools/clARMOR - MIT
- Technical Details
  - *Dynamic buffer overflow detection for GPGPUs*, CGO 2017

DISCLAIMER & ATTRIBUTION

AMD

ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

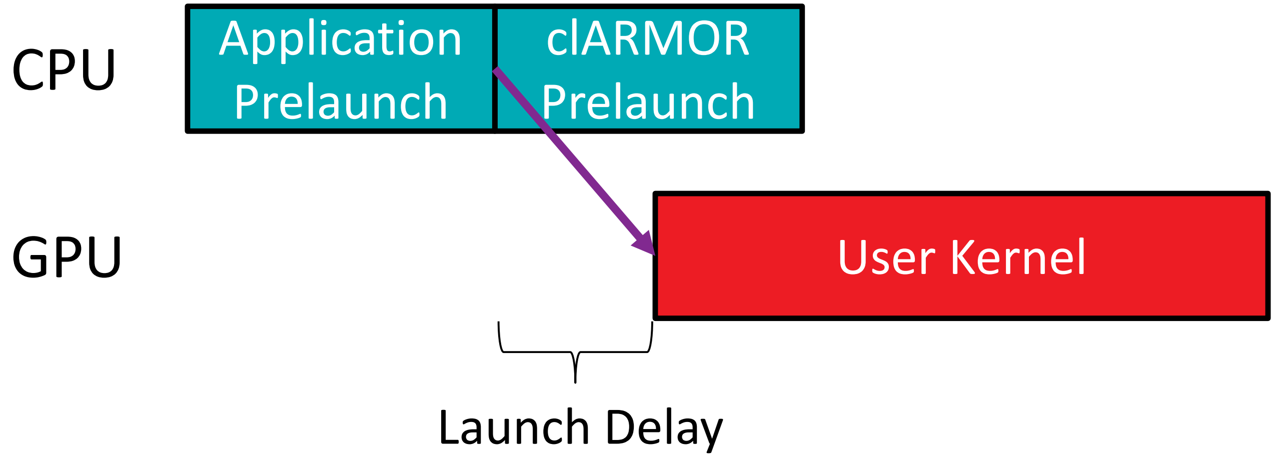
CPU

Application Prelaunch

GPU

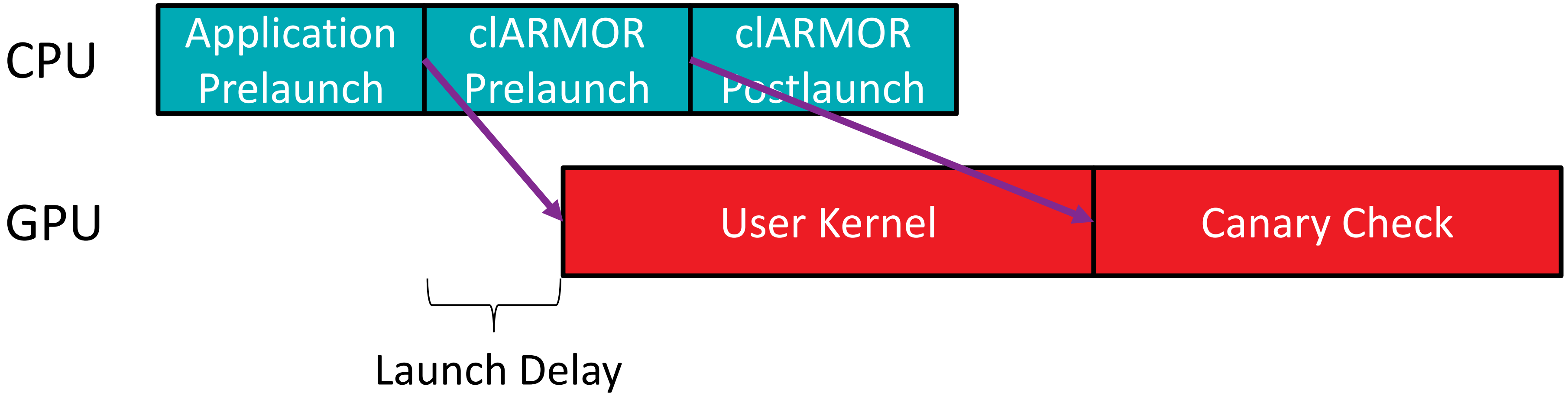
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch



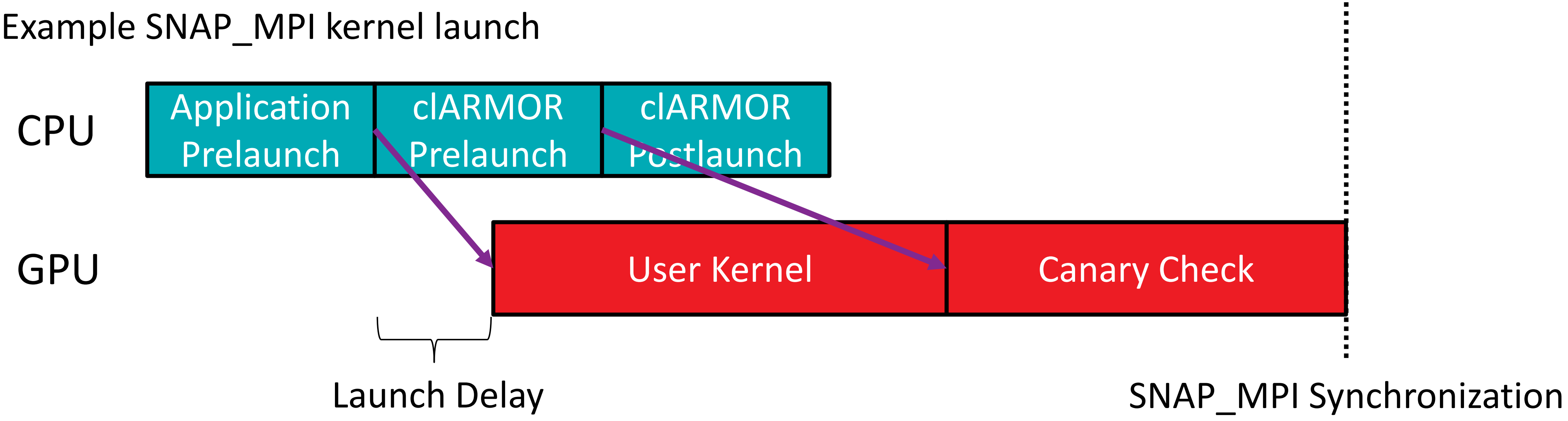
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch



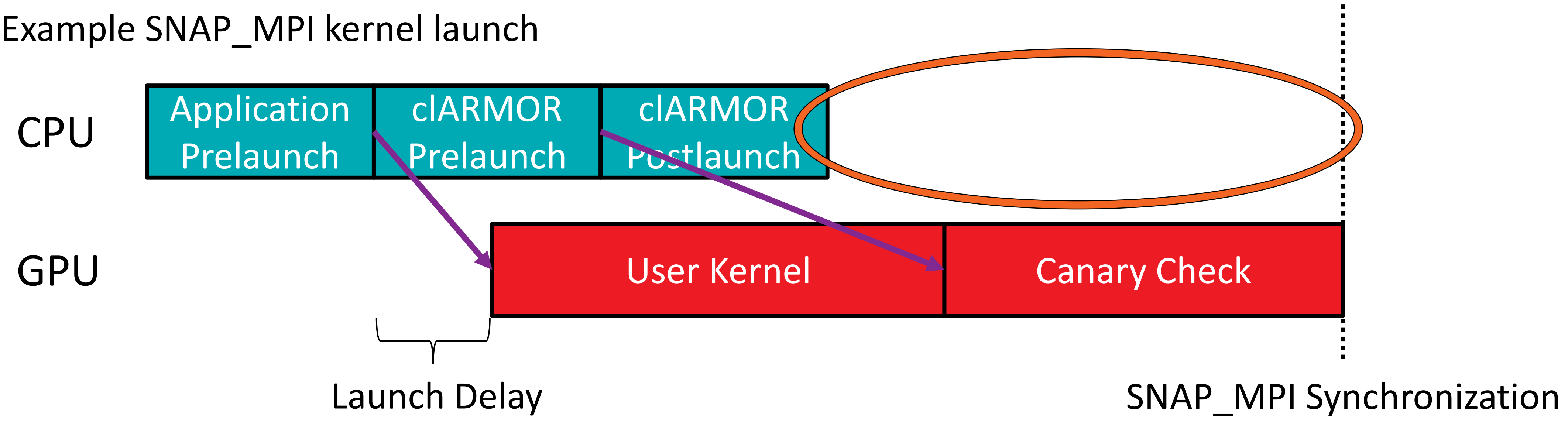
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch



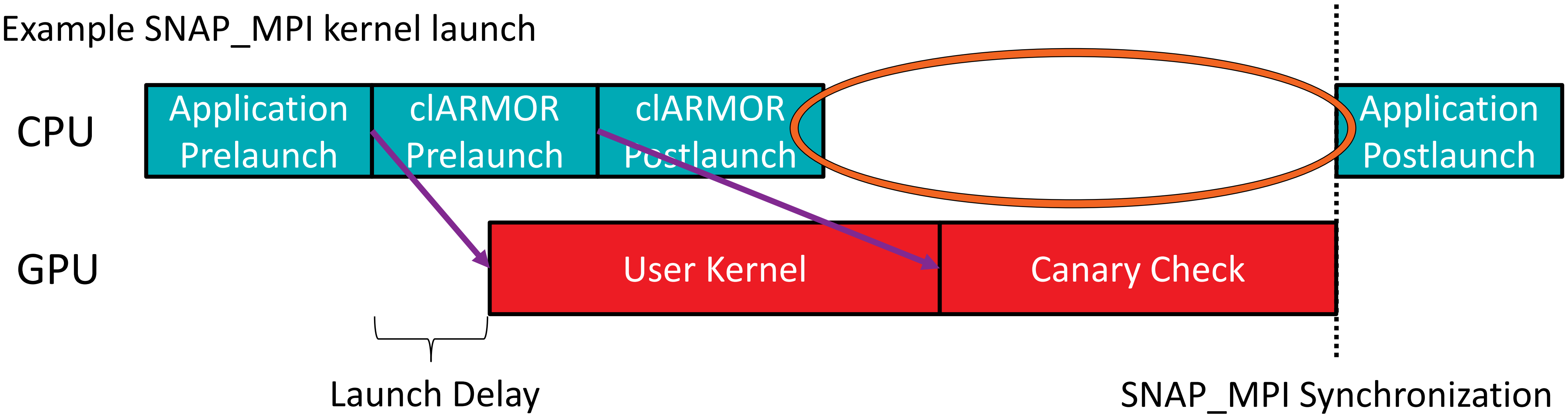
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch



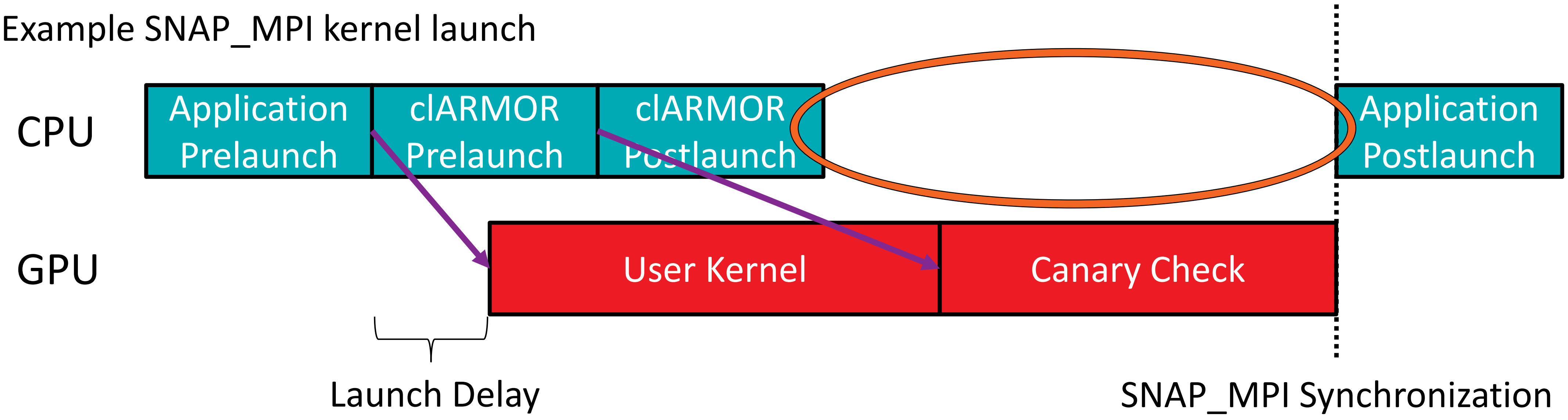
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch



ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

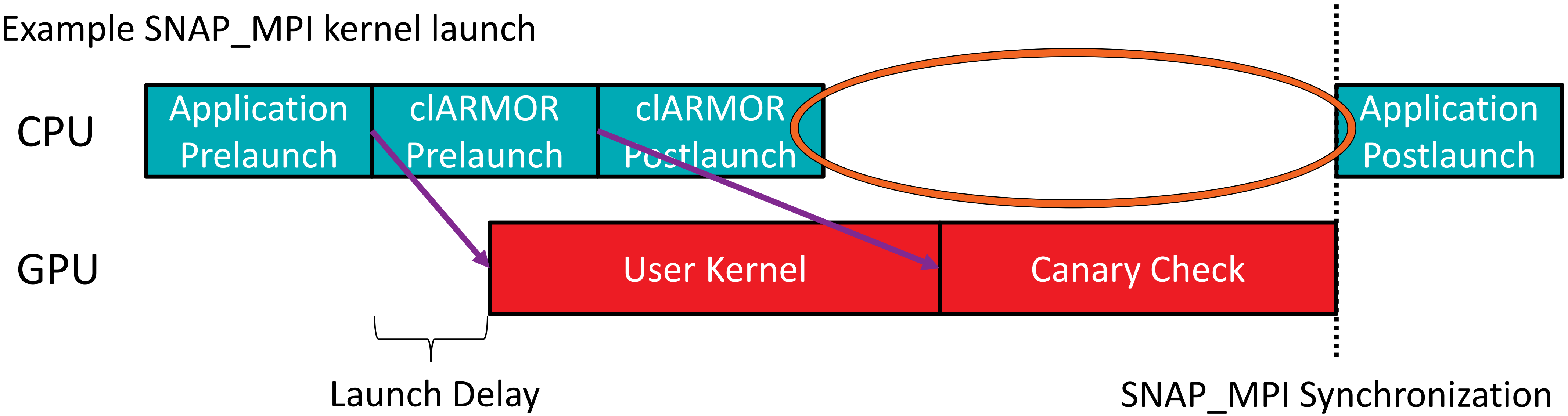
Example SNAP_MPI kernel launch



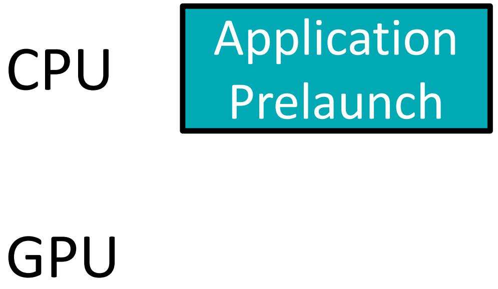
Possible improvement for SNAP_MPI kernel launch

ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

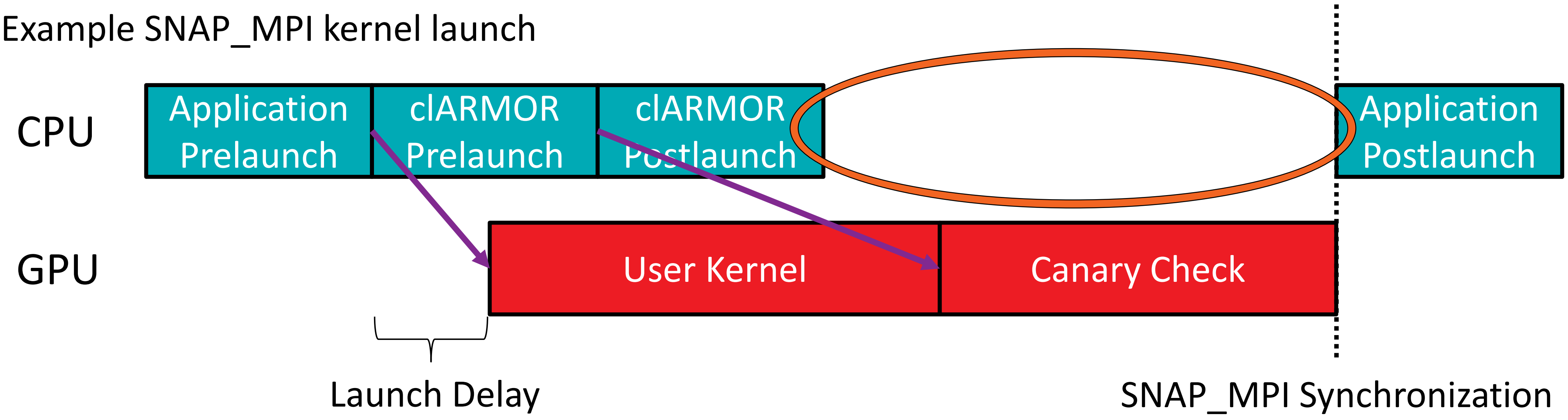


Possible improvement for SNAP_MPI kernel launch

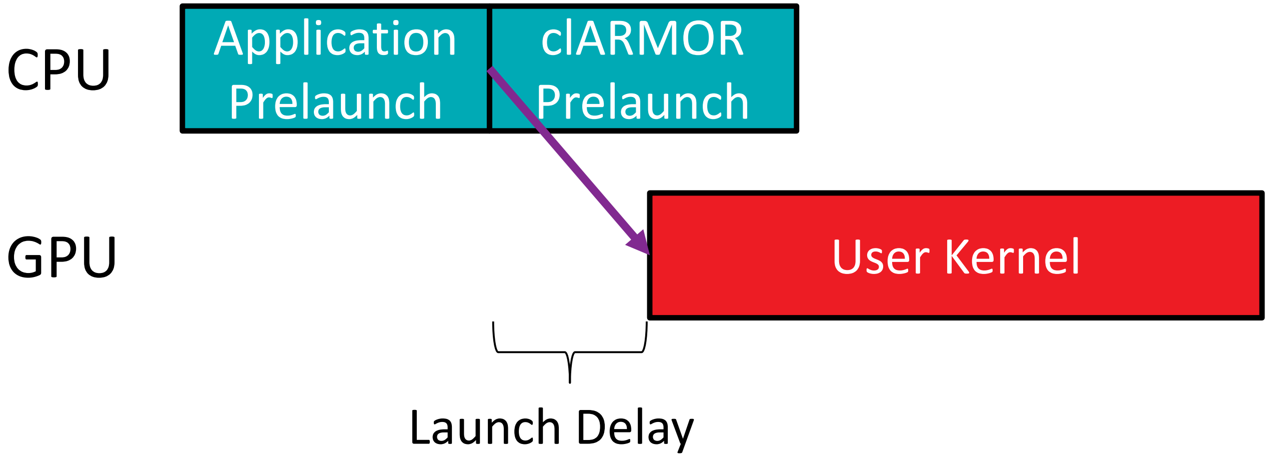


ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

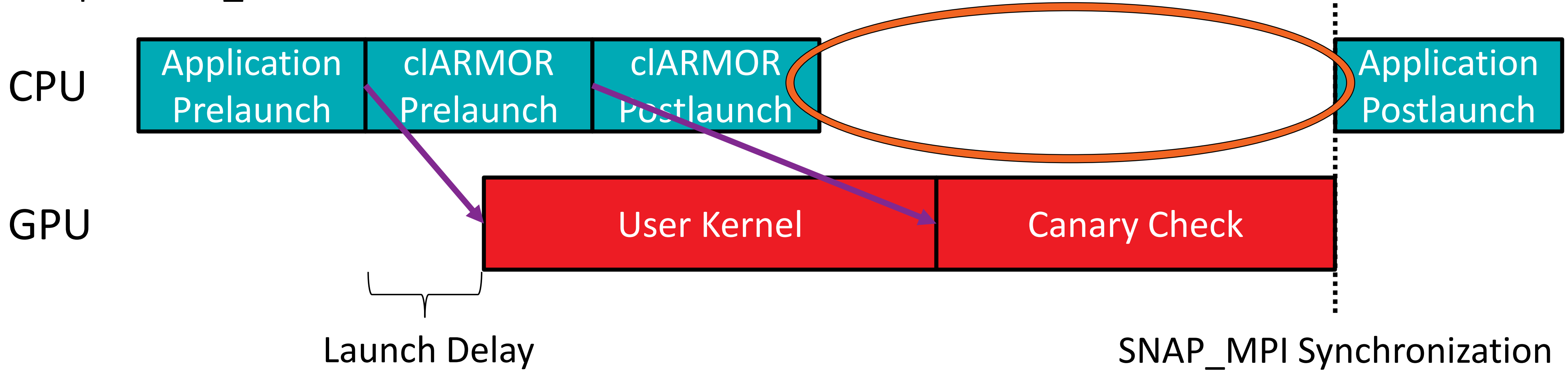


Possible improvement for SNAP_MPI kernel launch

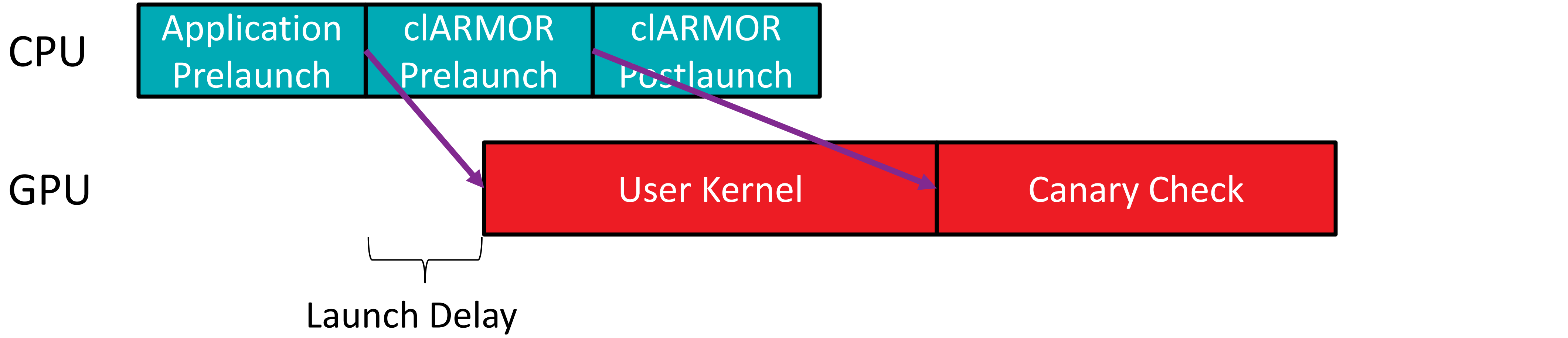


ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

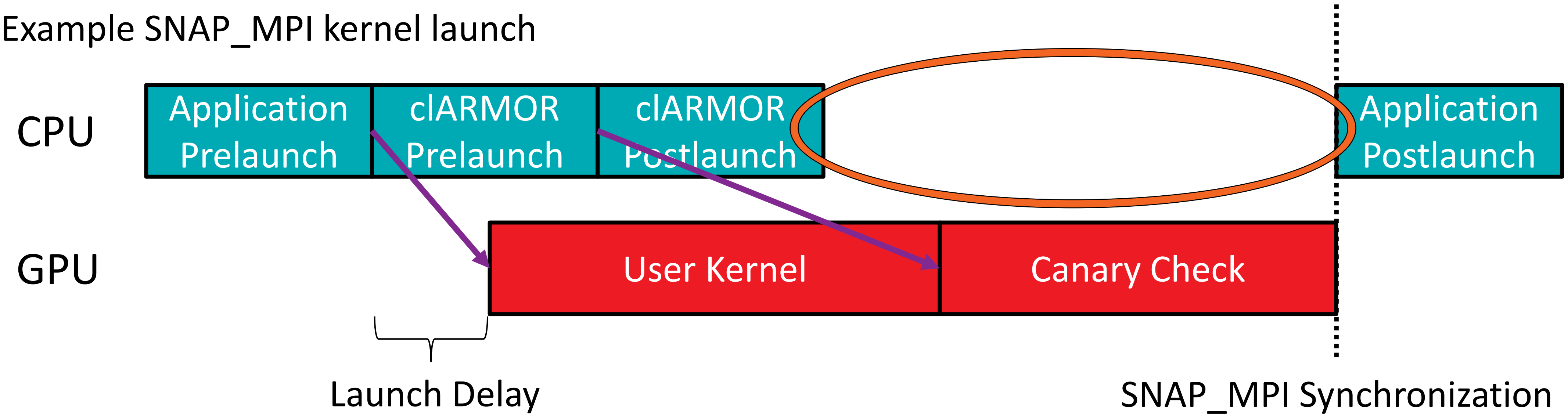


Possible improvement for SNAP_MPI kernel launch

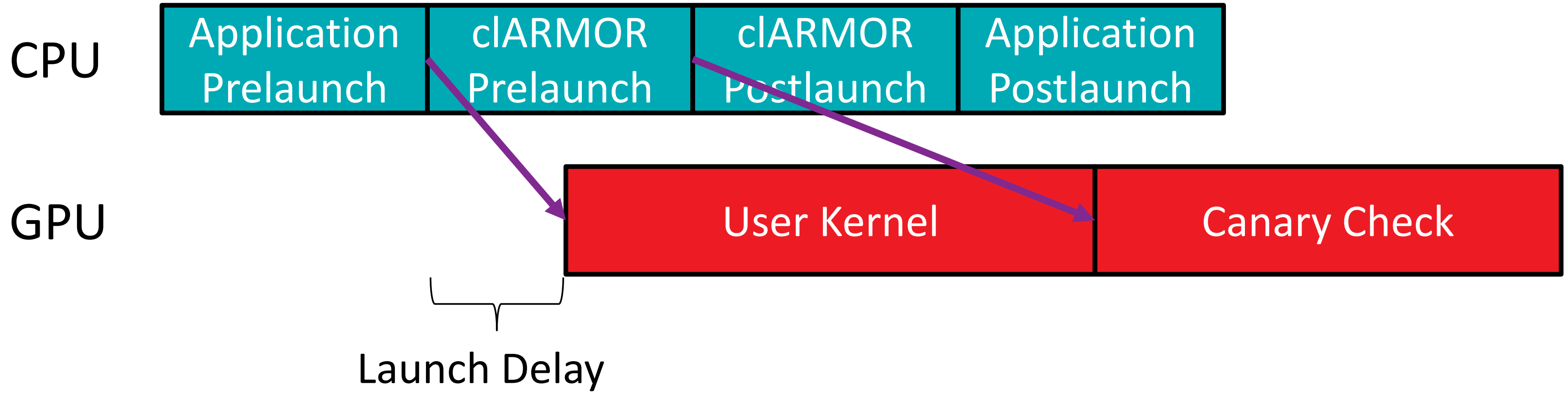


ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

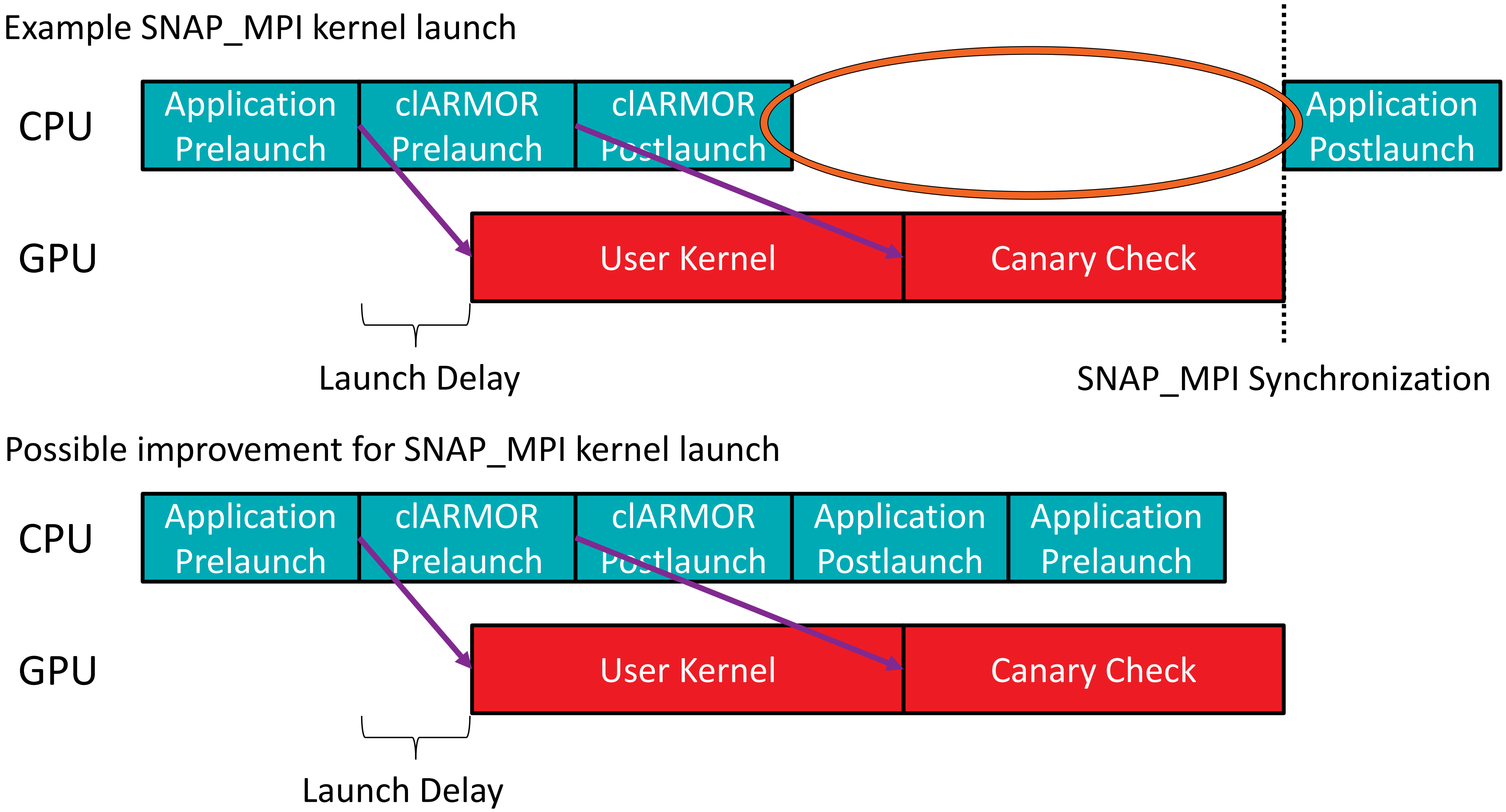
Example SNAP_MPI kernel launch



Possible improvement for SNAP_MPI kernel launch

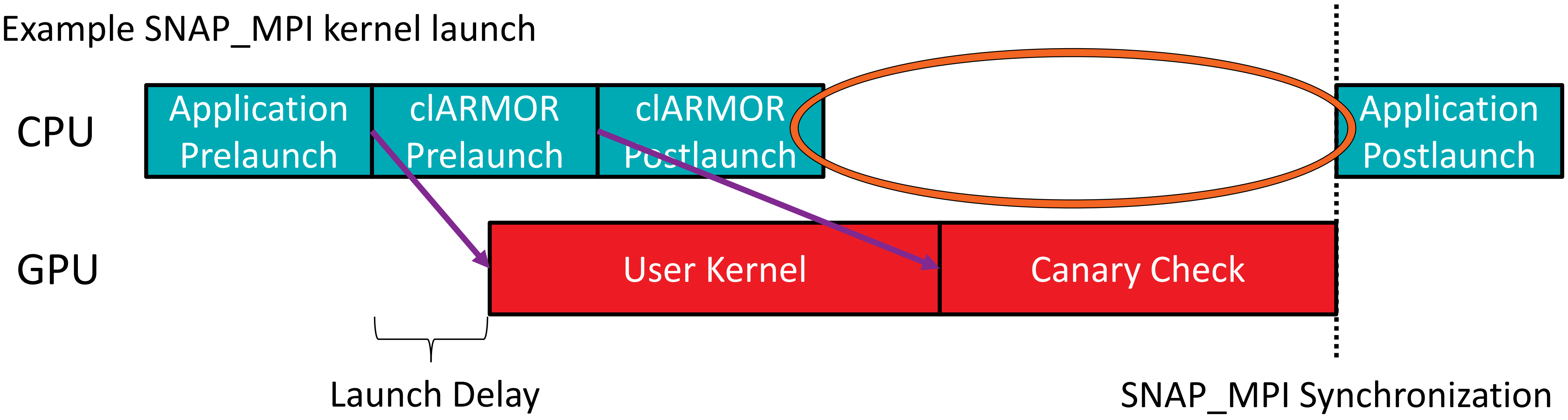


ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

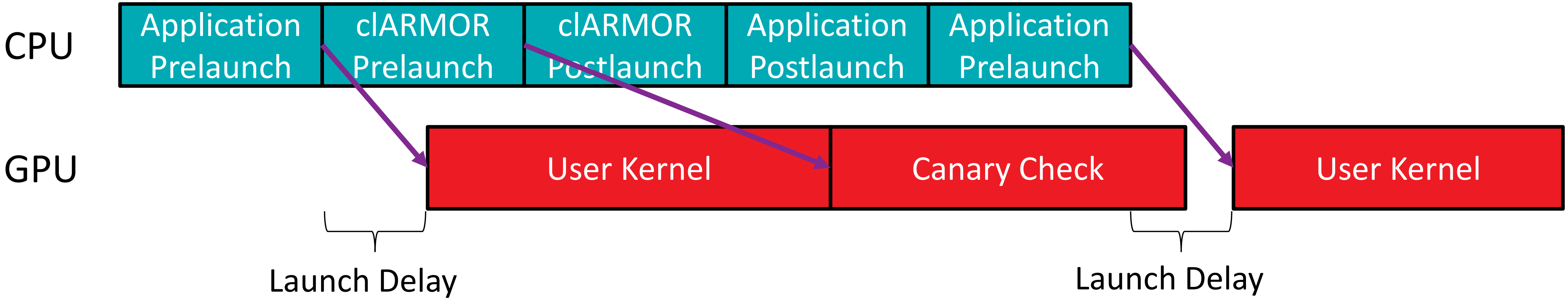


ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch



Possible improvement for SNAP_MPI kernel launch



Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

x = m - 1

y = n - 1

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

x = m - 1

y = n - 1

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
```

$x = m - 1$

$y = n - 1$

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
```

x = m - 1

y = n - 1

➡ m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
```

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
```

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
```

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
```

➡ x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
```

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
```

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
m*n – 1 + m <= m*n - 1
```

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
```

$\cancel{m*n - 1} + m <= \cancel{m*n} - 1$

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

```
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
m*n - 1 + m <= m*n - 1
m <= 0
```

$$\cancel{m*n - 1} + m <= \cancel{m*n -} 1$$

x = m - 1

y = n - 1

m == M_LEN

Host

```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

Hetero-Mark OpenCL™ 1.2 SW Overflow Error

AMD

Kernel

```
__kernel void sw_compute0(…
        const unsigned M_LEN,
        …
        __global double *cu,
        … ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
        …
}
```

(y + 1) * M_LEN + x

(y + 1) * m + x

(n) * m + x

n * m + m - 1

~~m*n - 1~~ + m <= ~~m*n -~~ 1

m <= 0

x = m - 1

y = n - 1

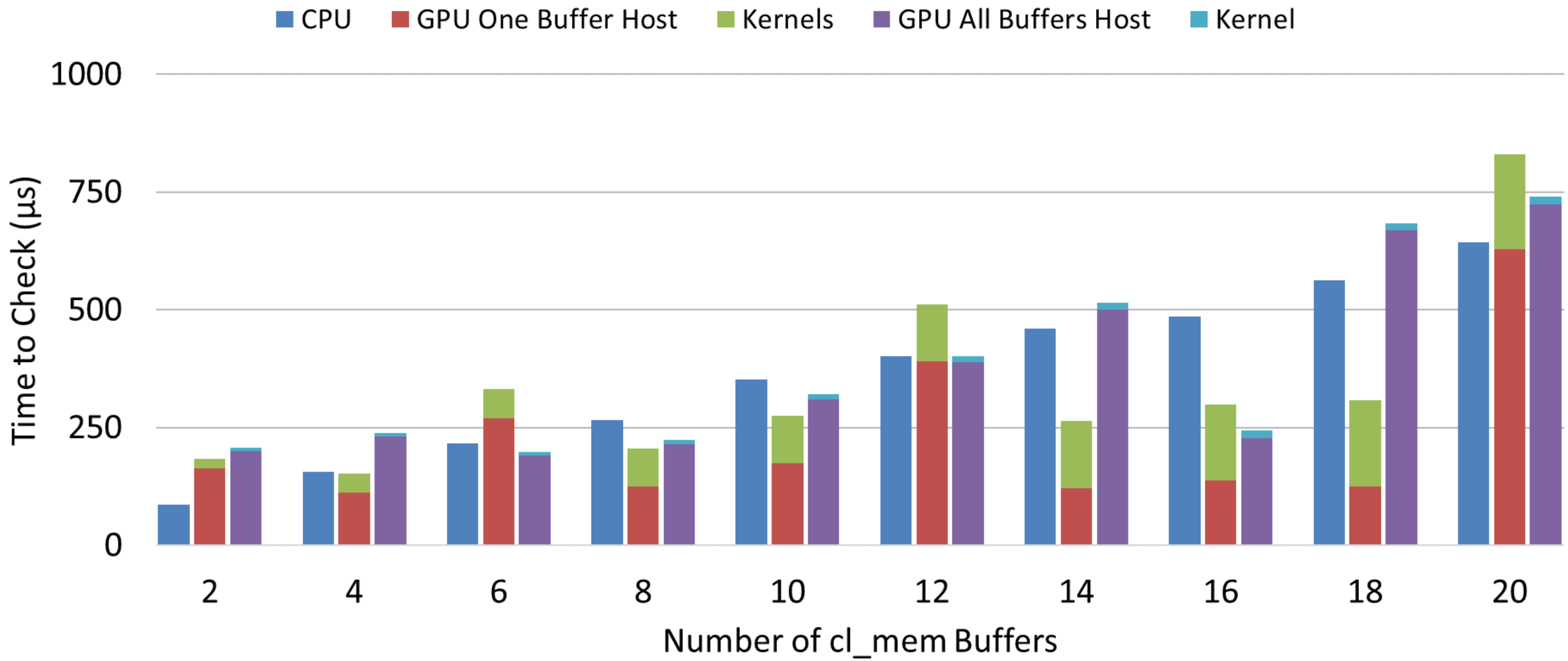
m == M_LEN

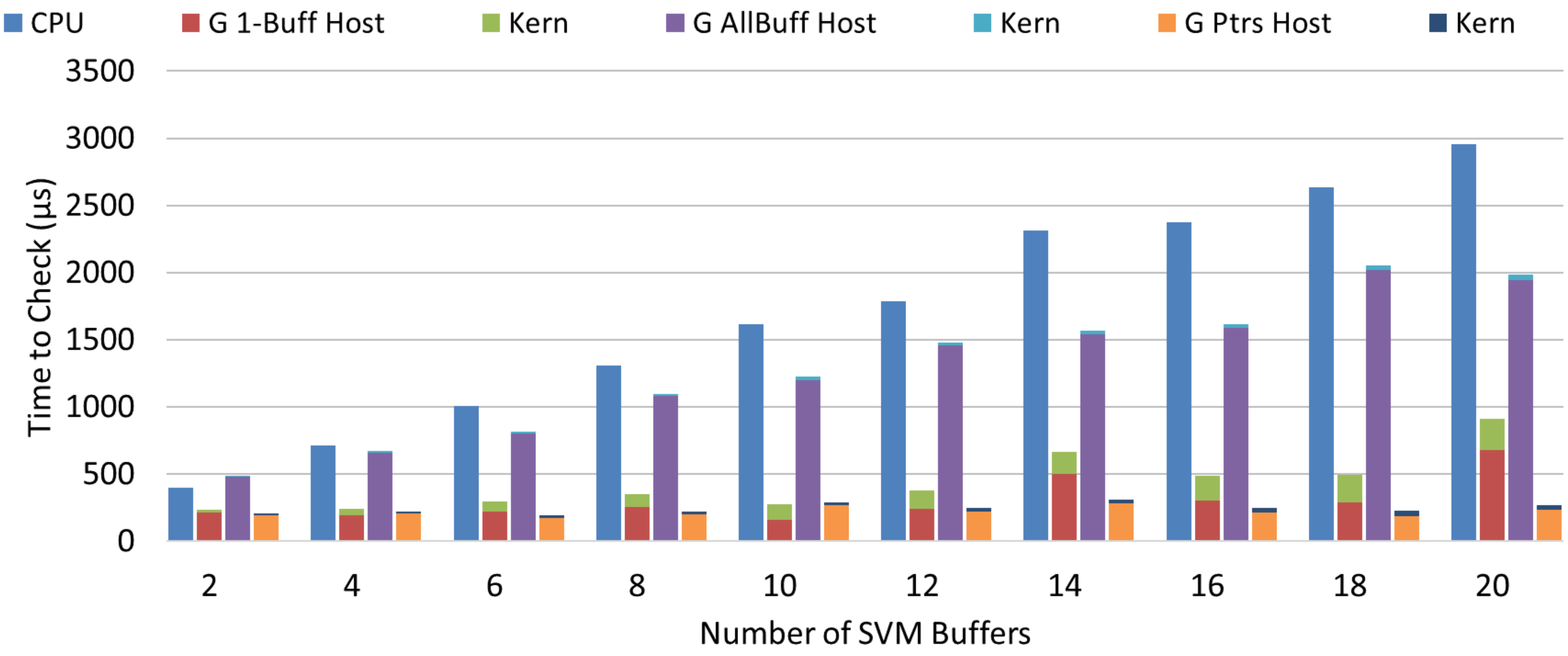
Host

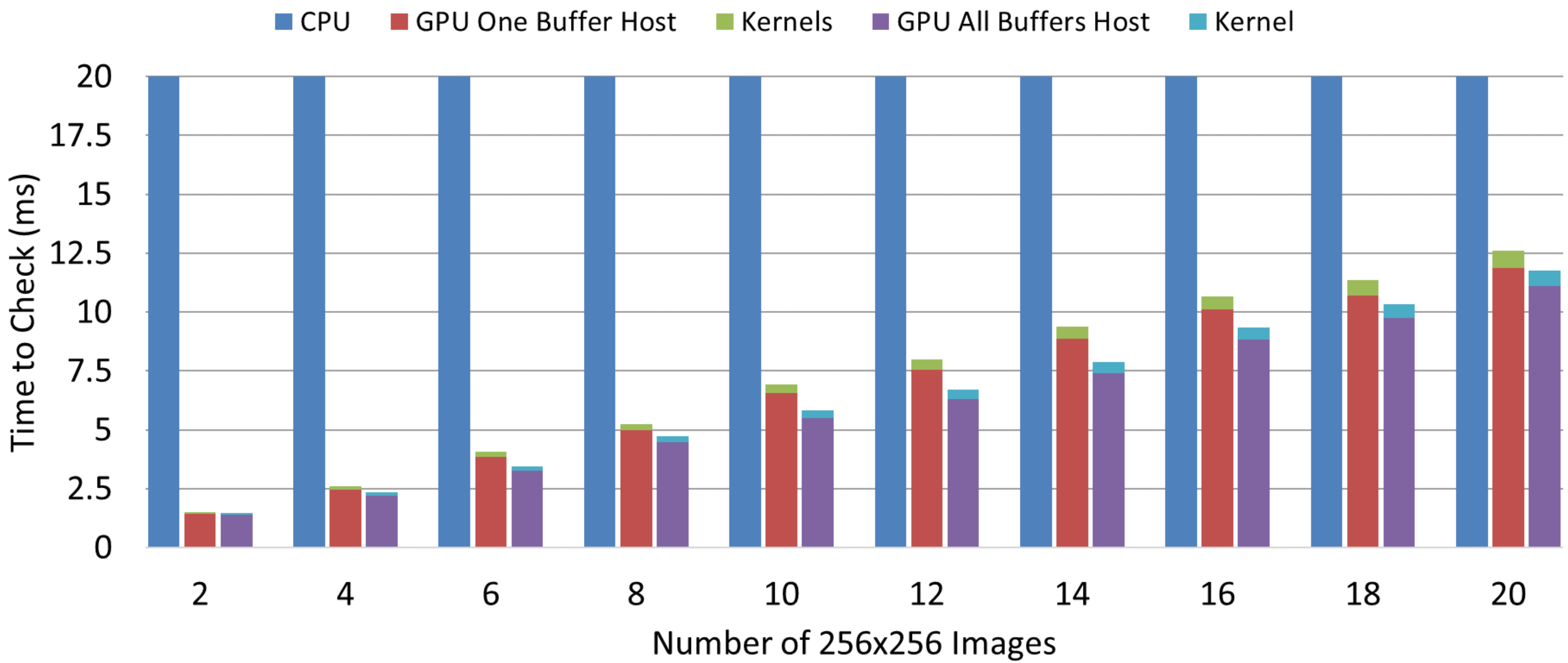
```
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL, globalSize,
        localSize, 0, NULL, NULL);
```

EXAMPLE ERROR

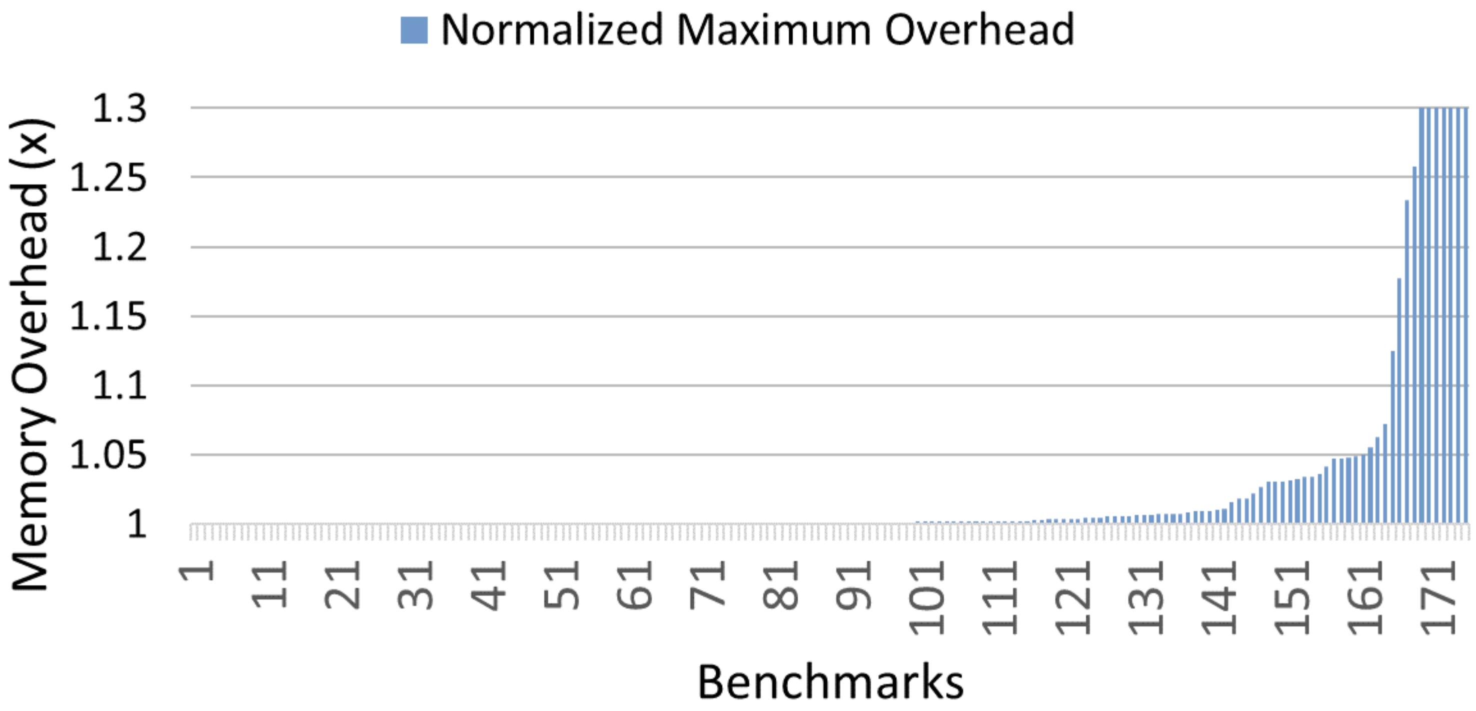
```
clARMOR: Loaded CL_WRAPPER
clARMOR:
clARMOR: ATTENTION:
clARMOR: ************** Buffer overflow detected ************
clARMOR: Kernel: sw_compute0, Buffer: cu
clARMOR:     First observed writing 1 byte(s) past the end.
clARMOR:
clARMOR: Exiting application because of buffer overflow.
```


CPU
GPU One Buffer Host
Kernels
GPU All Buffers Host
Kernel
1000
750
500
250
0
Time to Check (µs)
2
4
6
8
10
12
14
16
18
20
Number of cl_mem Buffers

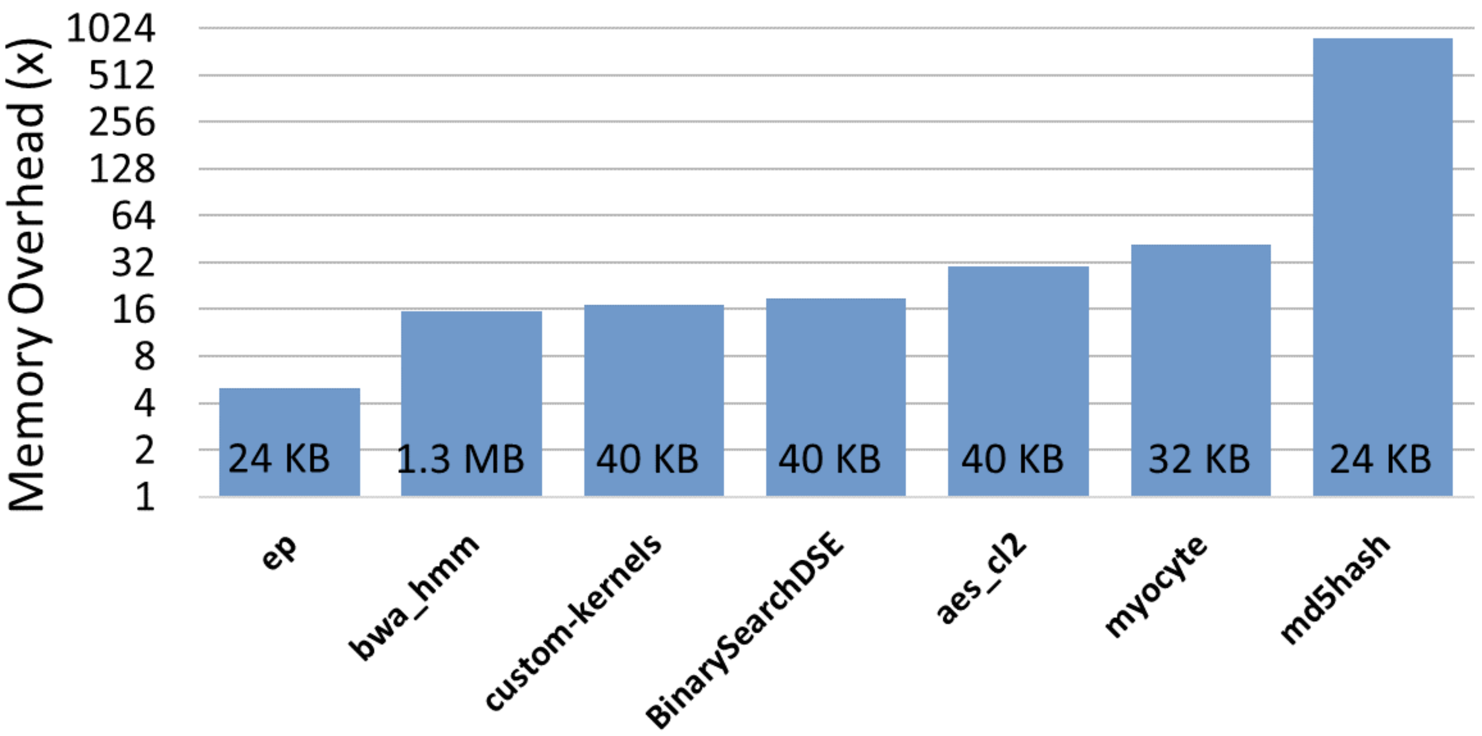

CPU
G 1-Buff Host
Kern
G AllBuff Host
Kern
G Ptrs Host
Kern
Time to Check (µs)
0
500
1000
1500
2000
2500
3000
3500
2
4
6
8
10
12
14
16
18
20
Number of SVM Buffers



CPU
GPU One Buffer Host
Kernels
GPU All Buffers Host
Kernel
Time to Check (ms)
0
2.5
5
7.5
10
12.5
15
17.5
20
2
4
6
8
10
12
14
16
18
20
Number of 256x256 Images



Normalized Maximum Overhead
Memory Overhead (x)
1.3
1.25
1.2
1.15
1.1
1.05
1
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
Benchmarks



Memory Overhead (x)
1024
512
256
128
64
32
16
8
4
2
1
24 KB
1.3 MB
40 KB
40 KB
40 KB
32 KB
24 KB
ep
bwa_hmm
custom-kernels
BinarySearchDSE
aes_cl2
myocyte
md5hash

clARMOR DETECTION RESULTS

LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding
- StreamMR
  - kmeans
  - wordcount
- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors
- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)
- Note: These have been reported, and most fixed.

clARMOR DETECTION RESULTS

LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding
- StreamMR
  - kmeans
  - wordcount
- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors
- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)
- Note: These have been reported, and most fixed.

- [ ] Detect GPU Buffer Overflows
- [x] Compatible With Most OpenCL™
- [x] Low Runtime Overhead

clARMOR DETECTION RESULTS

LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding
- StreamMR
  - kmeans
  - wordcount
- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors
- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)
- Note: These have been reported, and most fixed.

- [x] Detect GPU Buffer Overflows
- [x] Compatible With Most OpenCL™
- [x] Low Runtime Overhead

clARMOR DETECTION RESULTS

LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding
- StreamMR
  - kmeans
  - wordcount
- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors
- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)
- Note: These have been reported, and most fixed.

- [x] Detect GPU Buffer Overflows
- [x] Compatible With Most OpenCL™
- [x] Low Runtime Overhead

CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

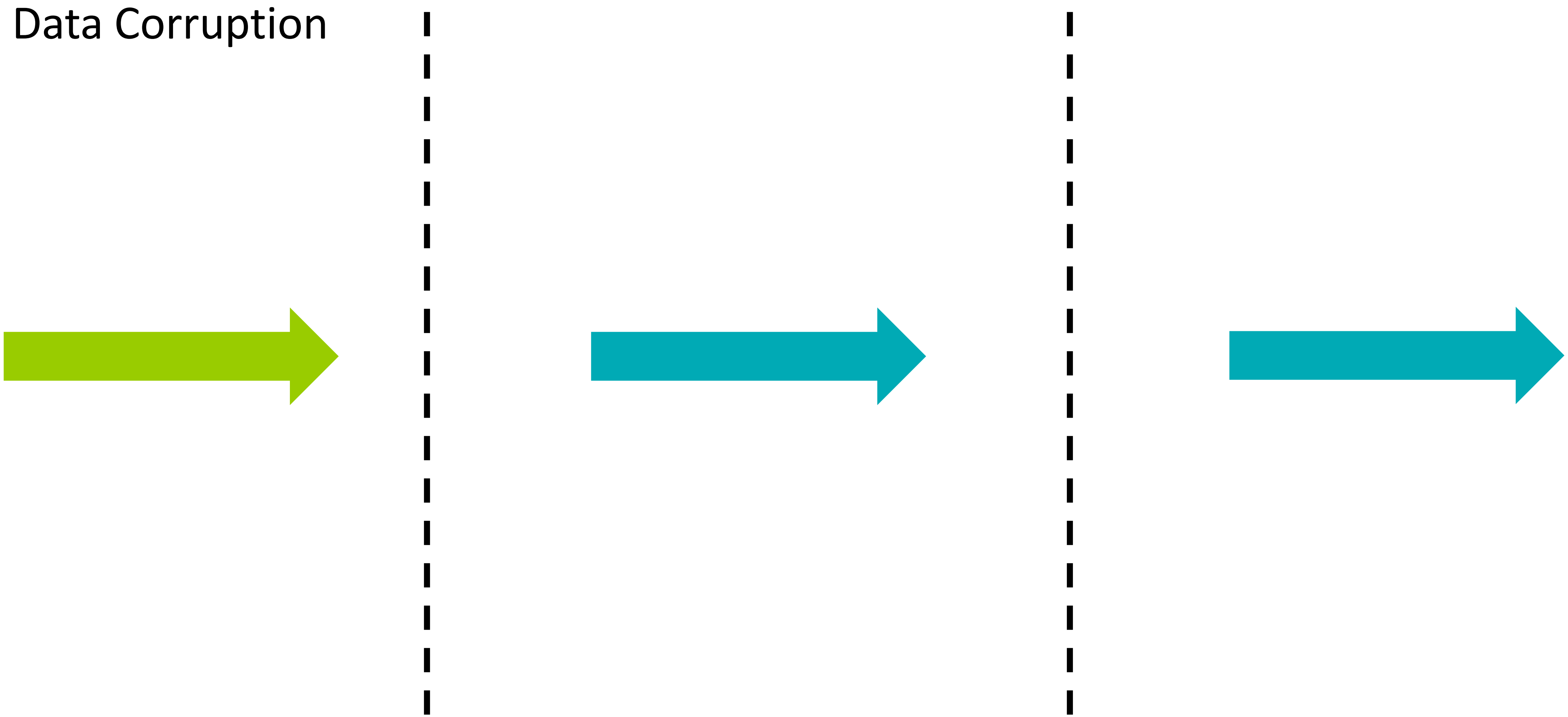
CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

CONSEQUENCES OF BUFFER OVERFLOWS

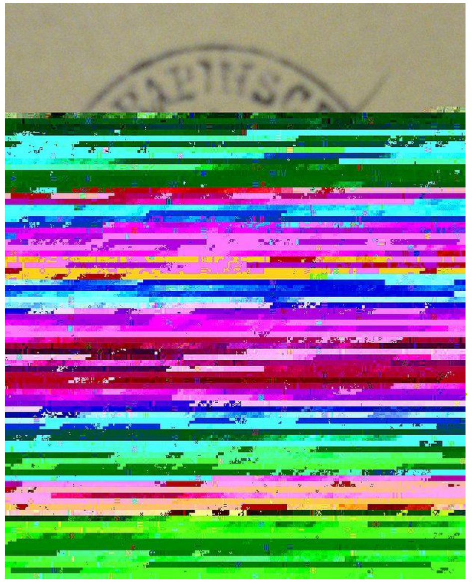
DEGRADING USER EXPERIENCE, AND SECURITY RISKS



CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

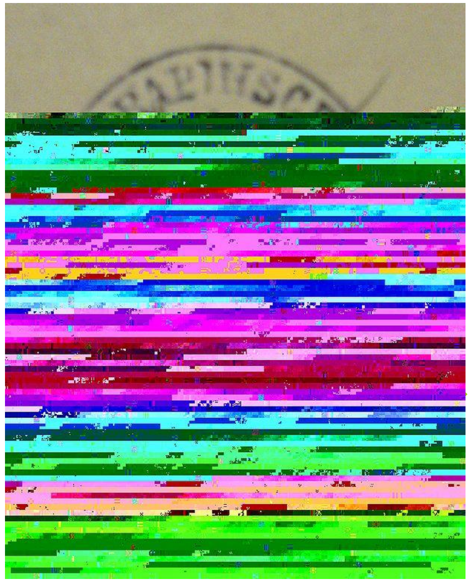
Data Corruption



CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

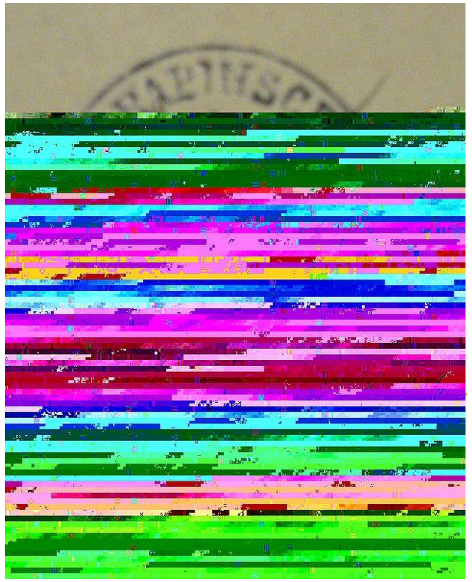


Segmentation Faults

CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

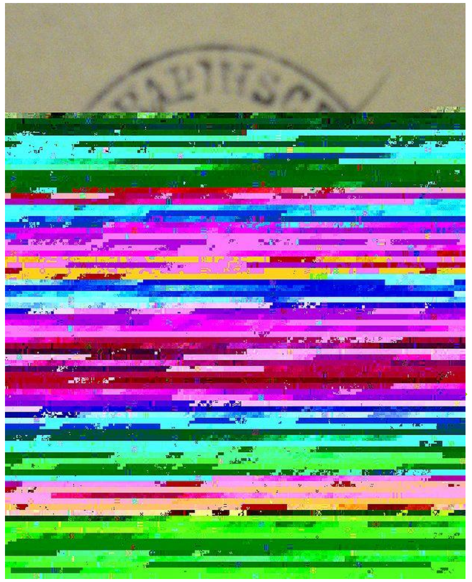


Segmentation Faults

CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

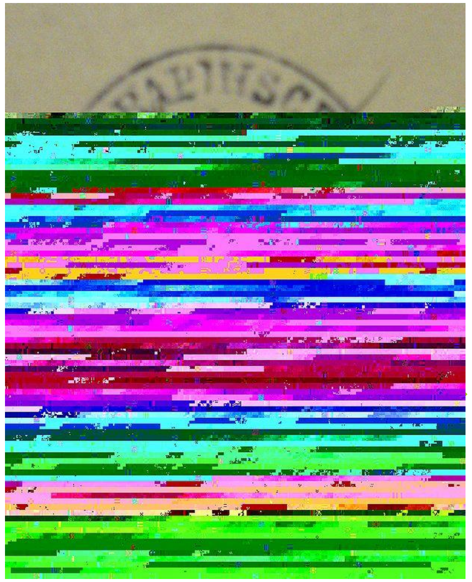


Segmentation Faults

CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption



Segmentation Faults

Altered Control Flow

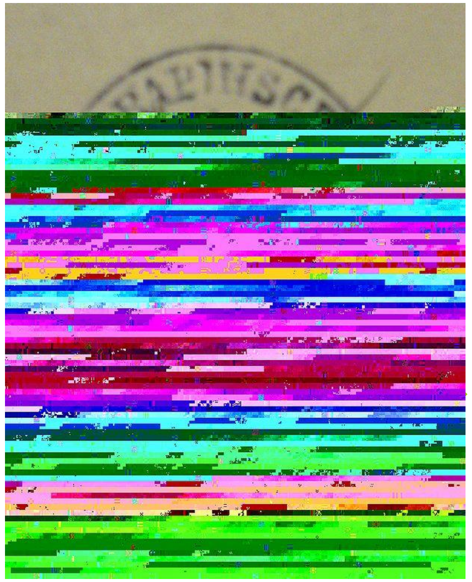
(Security Subversion)

CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS

AMD

Data Corruption



Segmentation Faults

Altered Control Flow

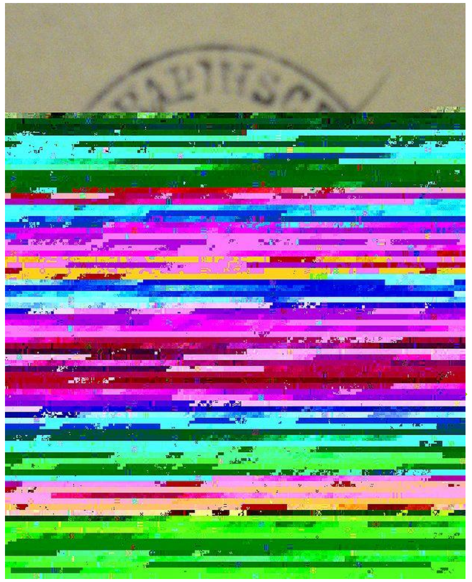
(Security Subversion)

CONSEQUENCES OF BUFFER OVERFLOWS

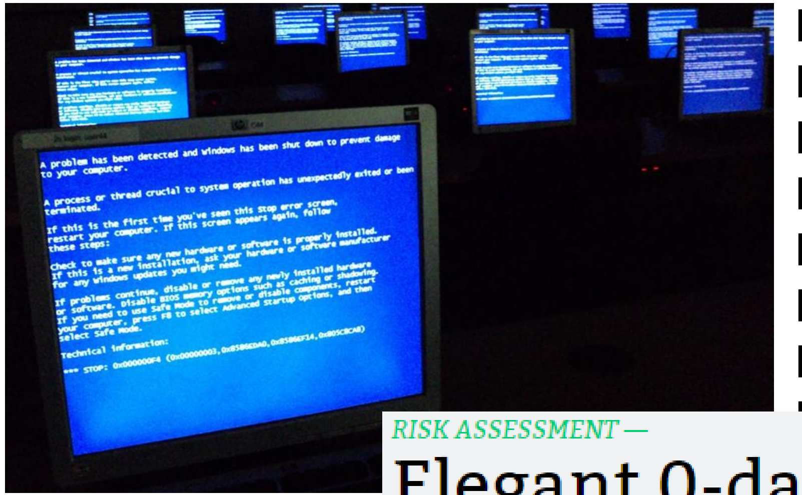
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

AMD

Data Corruption



Segmentation Faults



Altered Control Flow

(Security Subversion)

RISK ASSESSMENT —

Elegant 0-day unicorn underscores "serious concerns" about Linux security

Scriptless exploit bypasses state-of-the-art protections baked into the OS.

DAN GOODIN - 11/22/2016, 3:48 PM