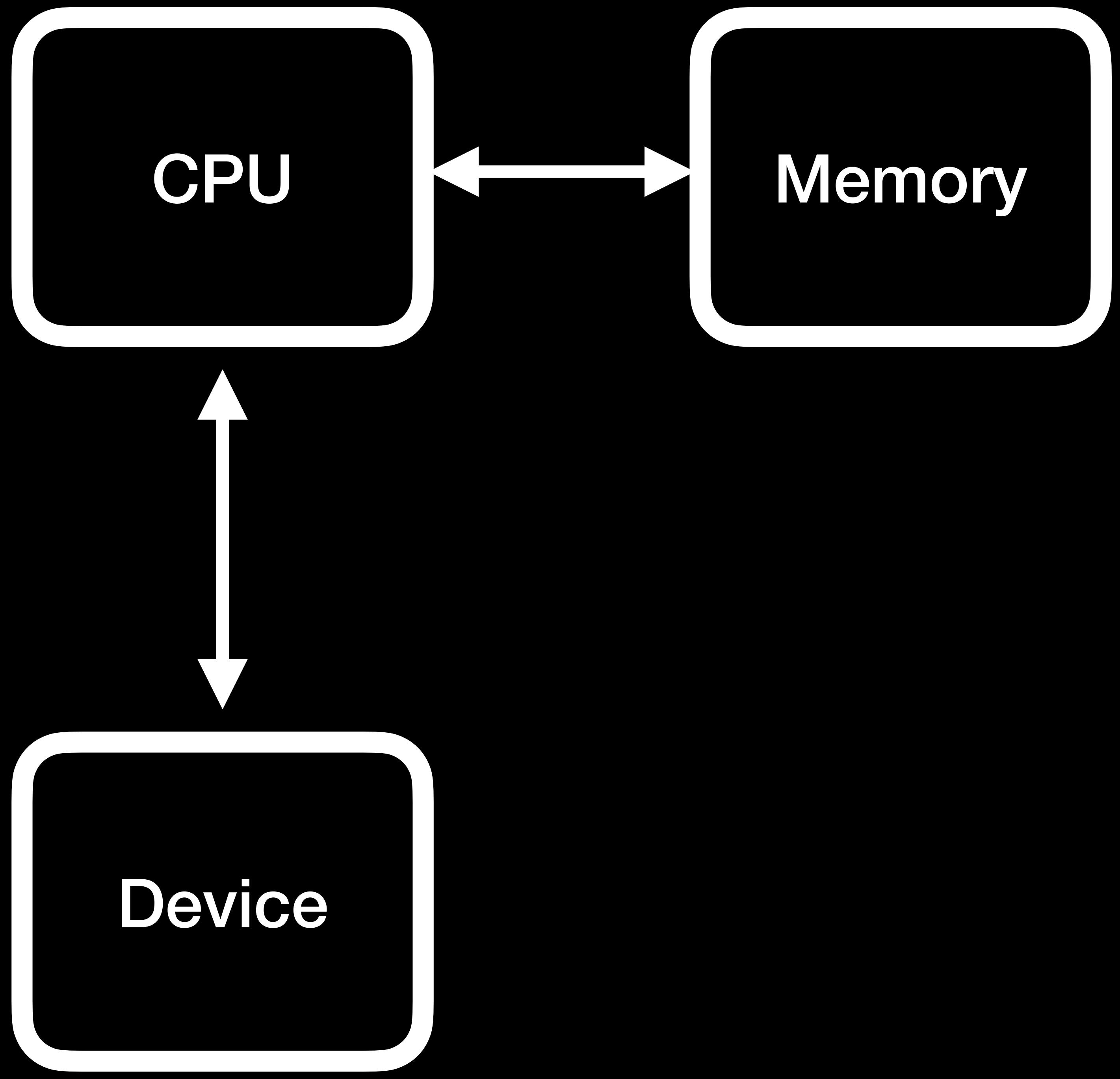
DMARacer

Dynamic Detection of Vulnerable DMA Race Conditions

Brian Johannesmeyer, Raphael Isemann,
Cristiano Giuffrida, Herbert Bos

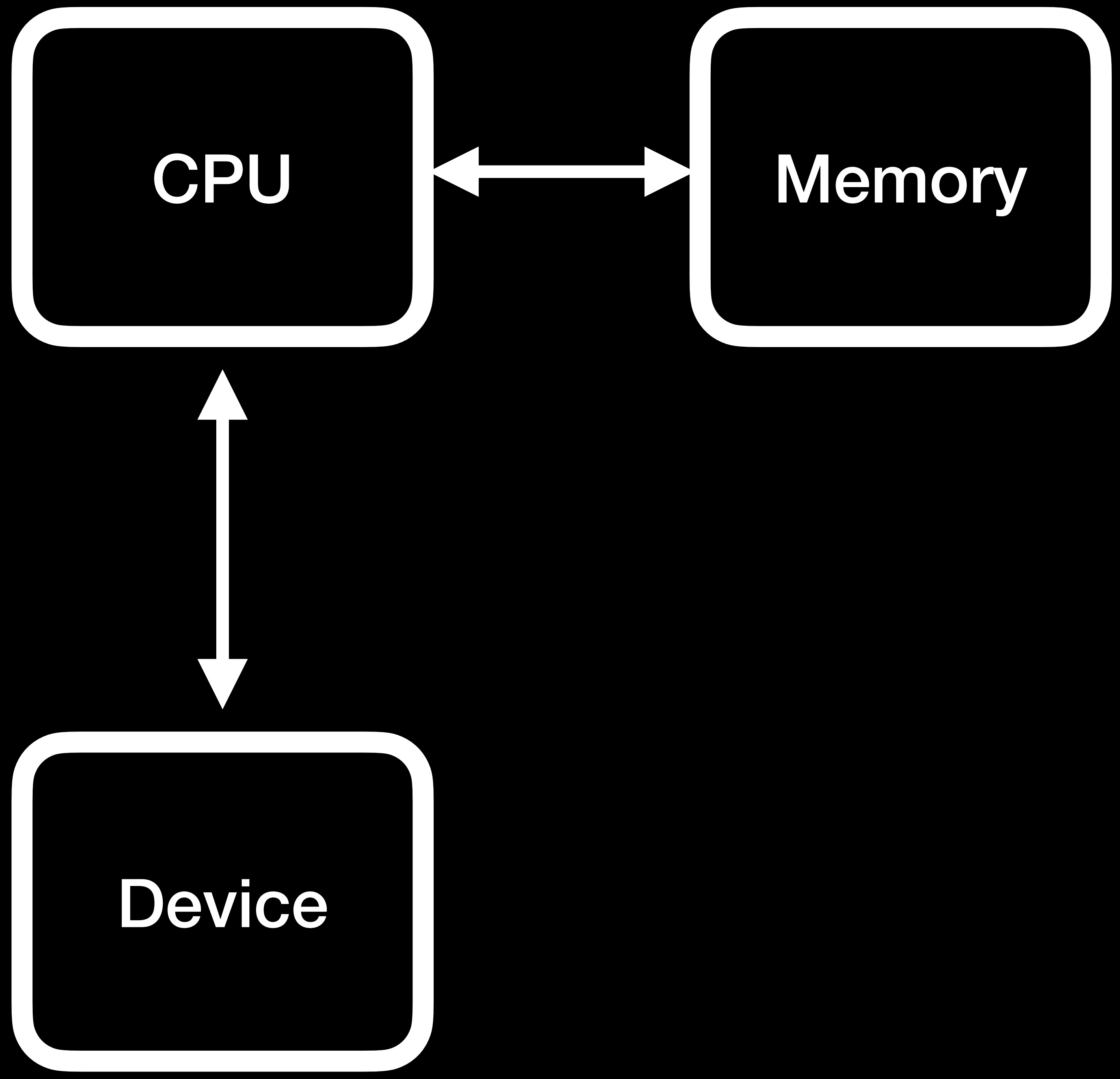
VUSec

Direct Memory Access (DMA)



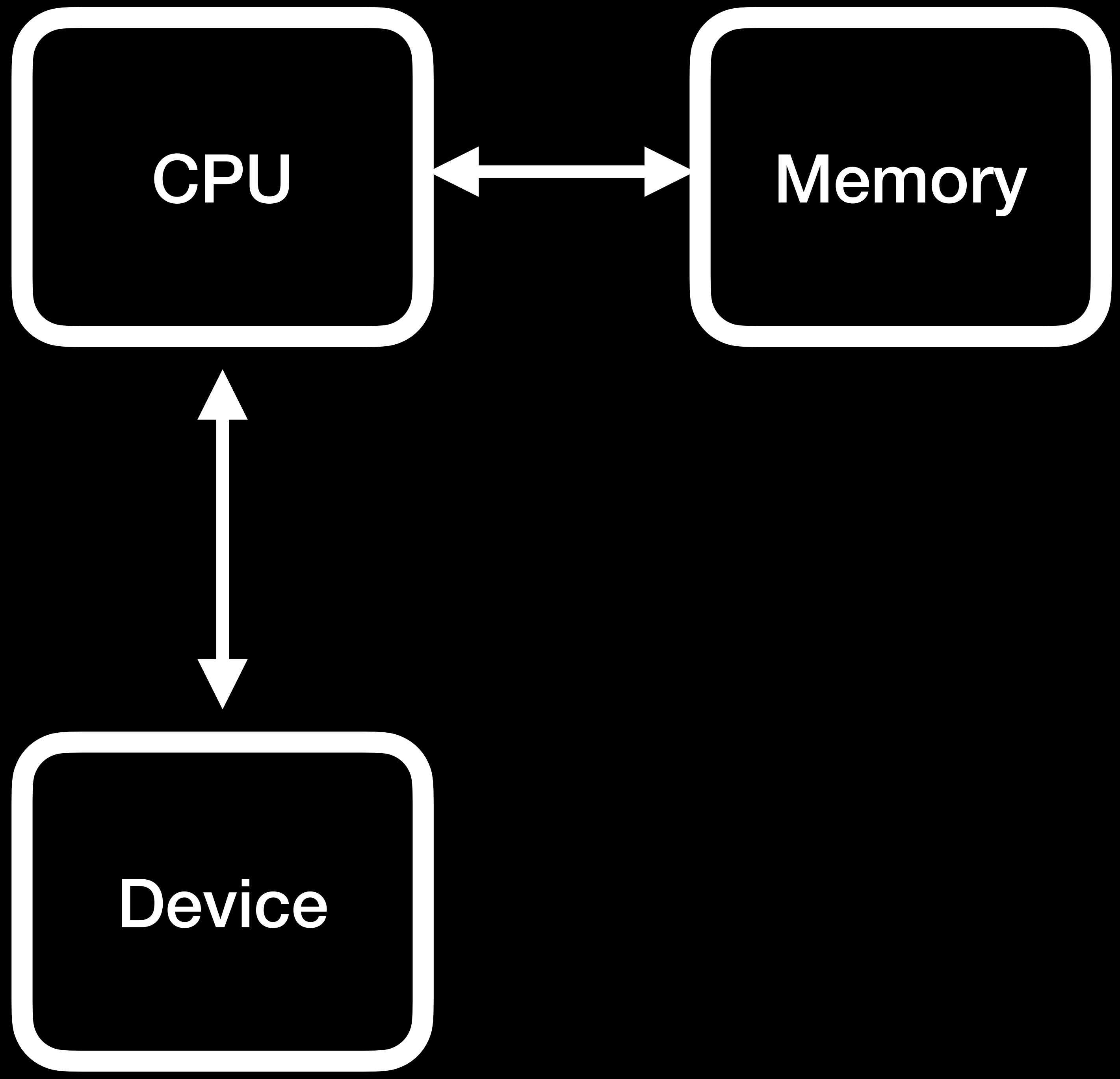
Direct Memory Access (DMA)

- Kernel and devices need to **communicate with each other**.



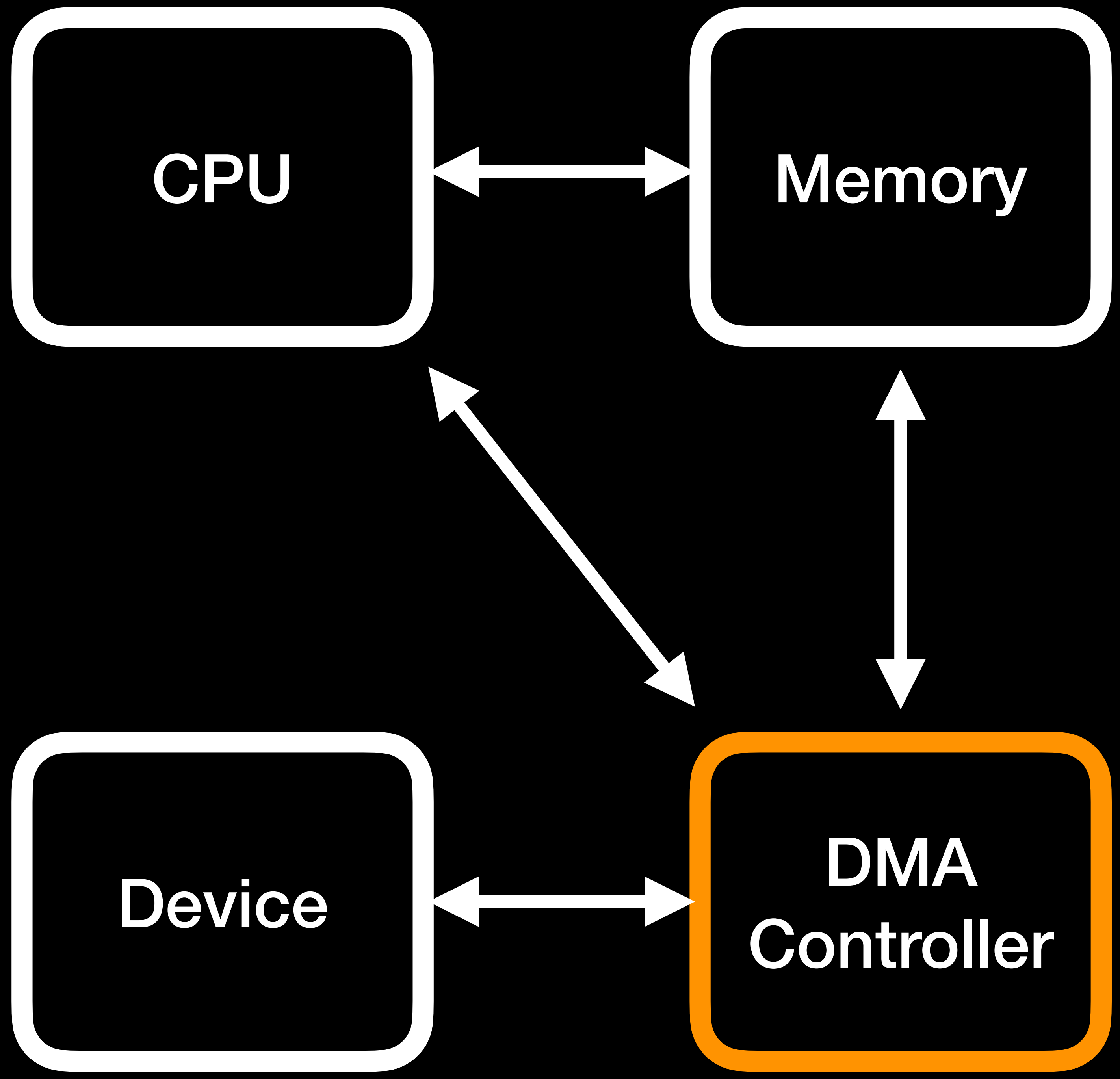
Direct Memory Access (DMA)

- Kernel and devices need to **communicate with each other**.
- How do we do communicate **efficiently**?



Direct Memory Access (DMA)

- Kernel and devices need to **communicate with each other**.
- How do we do communicate **efficiently**?
- → **DMA controller**
  - **Untrusted Peripherals** access parts of main memory
  - CPU not involved in transfer



DMA in the Linux Kernel

```
int *dma = dma_alloc(…);
// Writes shared memory!
*dma = 4;
```

DMA in the Linux Kernel

- DMA buffers are normal buffers

```
int *dma = dma_alloc(…);
// Writes shared memory!
*dma = 4;
```

DMA in the Linux Kernel

- DMA buffers are normal buffers
- Two DMA ‘modes’:
  - **Coherent**: Synchronizes automatically.

```
int *dma = dma_alloc(…);
// Writes shared memory!
*dma = 4;
```

DMA in the Linux Kernel

- DMA buffers are normal buffers
- Two DMA 'modes':
  - **Coherent**: Synchronizes automatically.
  - **Streaming**: Synchronizes when kernel explicitly requests it.

```
int *dma = dma_alloc(…);
// Writes shared memory!
*dma = 4;
```

```
int *dma = map_dma(…);
sync_to_cpu(dma);
*dma = 4;
```

DMA Race Conditions

DMA Race Conditions

Time-of-Check/Time-Of-Use

```
if (*dma < 10)
  array[*dma] = 5;
```

DMA Race Conditions

Time-of-Check/Time-Of-Use

```
if (*dma < 10)
  array[*dma] = 5;
```

Time-Of-**Init**/Time-Of-Use

```
*dma = 3
// …
array[*dma] = 5;
```

DMA Race Conditions

Time-of-Check/Time-Of-Use

```
if (*dma < 10)
  array[*dma] = 5;
```

Time-Of-**Init**/Time-Of-Use

```
*dma = 3
// …
array[*dma] = 5;
```

Access to device-synced Memory

(Streaming DMA Only)

```
sync_to_device(dma);
// …
*dma = 10
```

DMARacer

```
int *dma = dma_alloc(…);



if (*dma < 10) {

  array[dma] = 5;
}
```

DMARacer

- Sanitizer for DMA race conditions

```
int *dma = dma_alloc(…);



if (*dma < 10) {

  array[dma] = 5;
}
```

DMARacer

- Sanitizer for DMA race conditions
- Custom **Runtime** in Kernel
  - Tracks state of DMA regions

```
int *dma = dma_alloc(…);
dmaracer_new(dma, …);

if (*dma < 10) {

  array[dma] = 5;
}
```

(DMARacer logic)

DMARacer

- Sanitizer for DMA race conditions
- Custom **Runtime** in Kernel
  - Tracks state of DMA regions
- **Compiler Instrumentation**
  - Informs runtime about all memory accesses

```
int *dma = dma_alloc(…);
dmaracer_new(dma, …);

dmaracer_load(dma);
if (*dma < 10) {
  dmaracer_load(dma);
  array[dma] = 5;
}
```

(DMARacer logic)

DMARacer

- Sanitizer for DMA race conditions
- Custom **Runtime** in Kernel
  - Tracks state of DMA regions
- **Compiler Instrumentation**
  - Informs runtime about all memory accesses
- Runtime then identifies races

```
int *dma = dma_alloc(…);
dmaracer_new(dma, …);

dmaracer_load(dma);
if (*dma < 10) {
  dmaracer_load(dma);
  array[dma] = 5;
}
```

(DMARacer logic)

Taint Tracking

```
if (*dma < 10)
  array[*dma] = 5;
```

Taint Tracking

- How do we what code is **vulnerable**?

```
if (*dma < 10)
  array[*dma] = 5;
```

Taint Tracking

- How do we what code is **vulnerable**?

```
if (*dma < 10)
  array[*dma] = 5;

if (*dma < 10)
  process_val(*dma);
```

Taint Tracking

- How do we what code is **vulnerable**?
- We use **dynamic taint tracking** (DFT)
  - Part of compiler-instrumentation

```
if (*dma < 10)
  array[*dma] = 5;
```

```
if (*dma < 10)
  process_val(*dma);
```

↓

```
void process_val(val) {
  arr[val] = 5;
}
```

Taint Tracking

- How do we what code is **vulnerable**?
- We use **dynamic taint tracking** (DFT)
  - Part of compiler-instrumentation
- Report tainted **sinks** such as:
  - memory accesses ⟹ buffer overflow
  - conditional jumps ⟹ DoS, etc.

```
if (*dma < 10)
  array[*dma] = 5;
```

```
if (*dma < 10)
  process_val(*dma);
```

↓

```
void process_val(val) {
  arr[val] = 5;
}
```

Overhead

(Means, measured via LMBench)

Overhead

DMARacer **401%**

(Means, measured via LMBench)

Overhead

DMARacer **401%**

156%

Detection

(Means, measured via LMBench)

Overhead

DMARacer **401%**

156% — Detection

245% — Taint-Tracking

(Means, measured via LMBench)

Generating Coverage

Generating Coverage

- Problem 1: We need to **run** drivers
  - Simple if you have physical hardware
  - We can **emulate some** of them **via QEMU**

Generating Coverage

- Problem 1: We need to **run** drivers
  - Simple if you have physical hardware
  - We can **emulate some** of them **via QEMU**
- Problem 2: We need to **exercise** devices
  - Also need to **spread taint** to sinks.
  - We use **device-specific workloads**
  - Ideal: Have a proper **fuzzing** system

Summary

Summary

- DMA-managed memory is a source of race conditions

Summary

- DMA-managed memory is a source of race conditions
- We can detect It using **DMARacer**

Summary

- DMA-managed memory is a source of race conditions
- We can detect It using **DMARacer**
  - **Kernel runtime** and **instrumentation for detection**

Summary

- DMA-managed memory is a source of race conditions
- We can detect It using **DMARacer**
  - **Kernel runtime** and **instrumentation for detection**
  - **Taint tracking** to identify affected **vulnerable code**

Summary

- DMA-managed memory is a source of race conditions
- We can detect It using **DMARacer**
  - **Kernel runtime** and **instrumentation for detection**
  - **Taint tracking** to identify affected **vulnerable code**
- Open questions: How can we **generate driver-specific coverage**?

Questions?

(Link to paper)