Understanding The Security of Discrete GPUs

Zhiting Zhu[1], Sangman Kim[1], Yuri Rozhanski[2], Yige Hu[1], Emmett Witchel[1], Mark Silberstein[2]

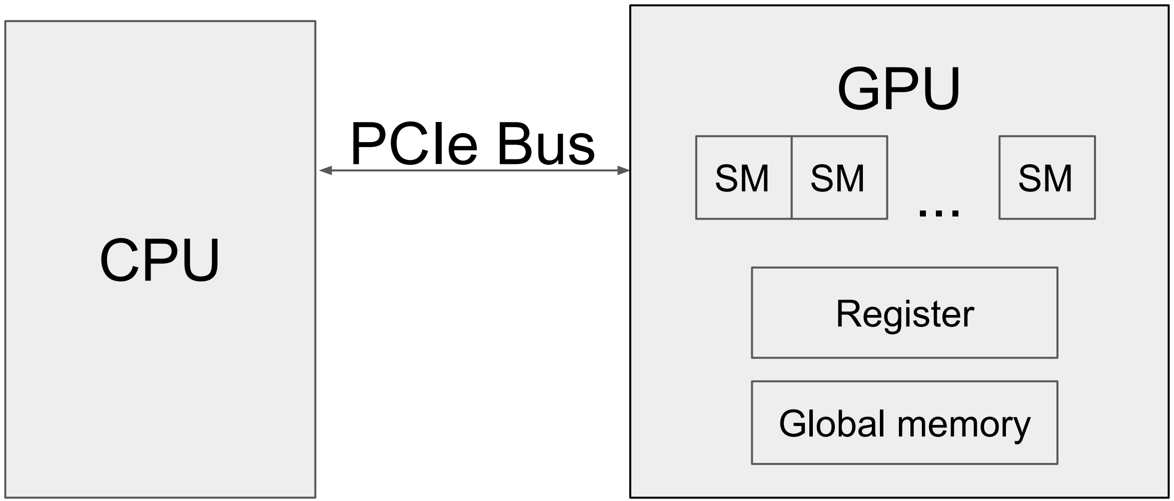
1.The University of Texas at Austin   2.Technion-Israel Institute of Technology

Outline

- Can GPUs improve the security of a computing system?
  - PixelVault
  - Attacking PixelVault
- Can GPUs subvert the security of a computing system?
  - GPU driver attack
  - GPU microcode attack
  - IOMMU mitigation

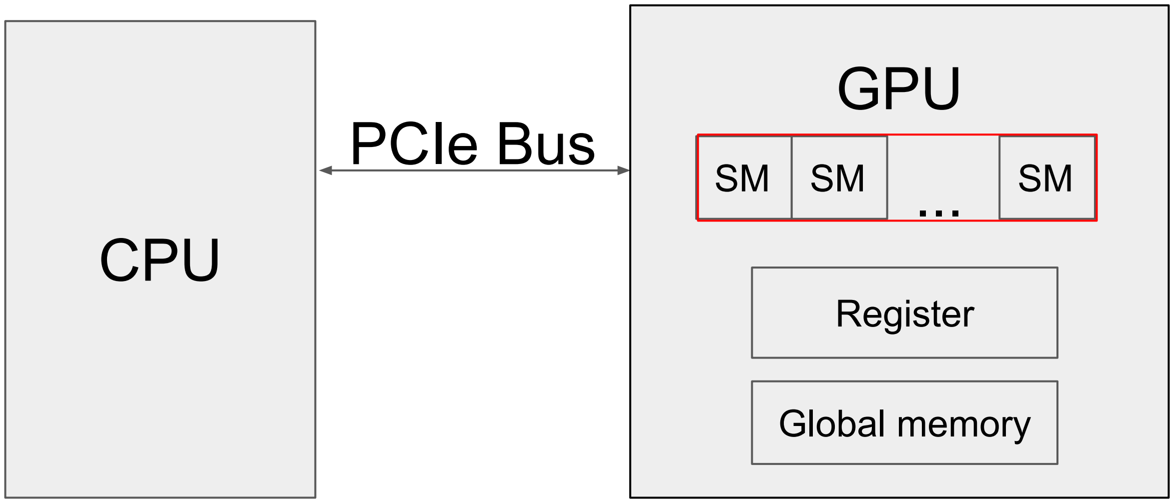
Can GPUs improve the security of a computing system?

Motivation: Dedicated hardware resources



Can GPUs improve the security of a computing system?

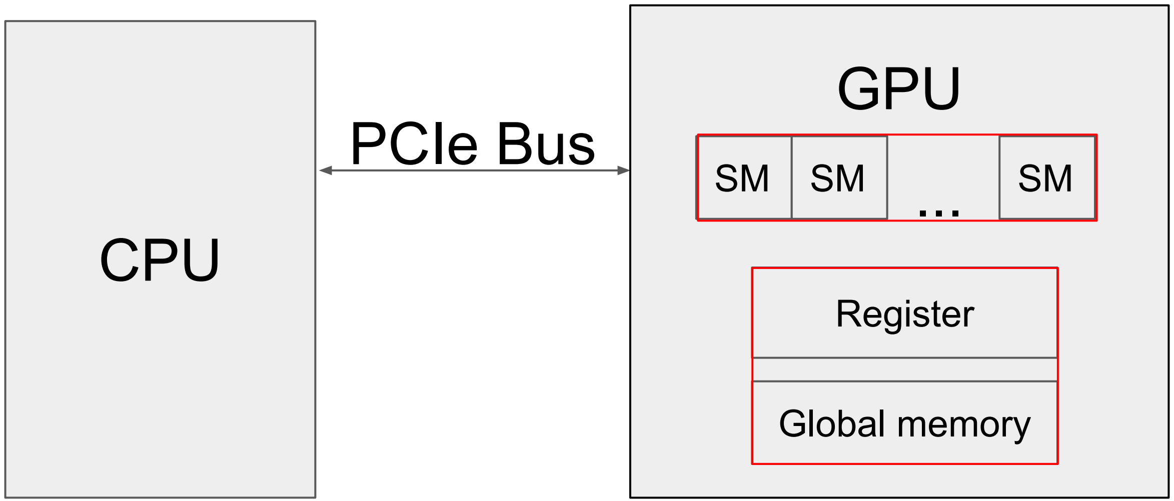
Motivation: Dedicated hardware resources



Independent computational resources

Can GPUs improve the security of a computing system?

Motivation: Dedicated hardware resources

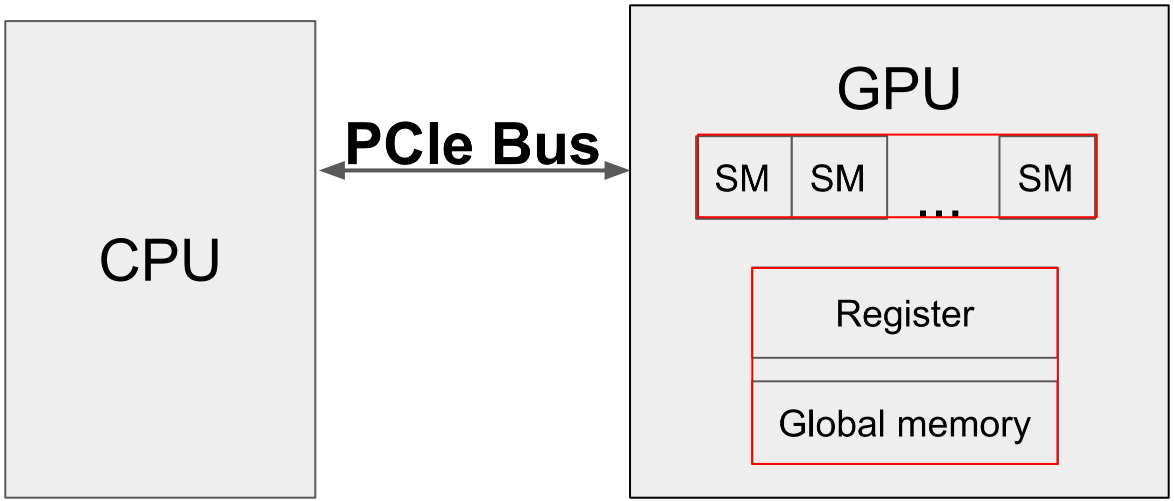


Independent computational resources

Independent memory system

Can GPUs improve the security of a computing system?

Motivation: Dedicated hardware resources

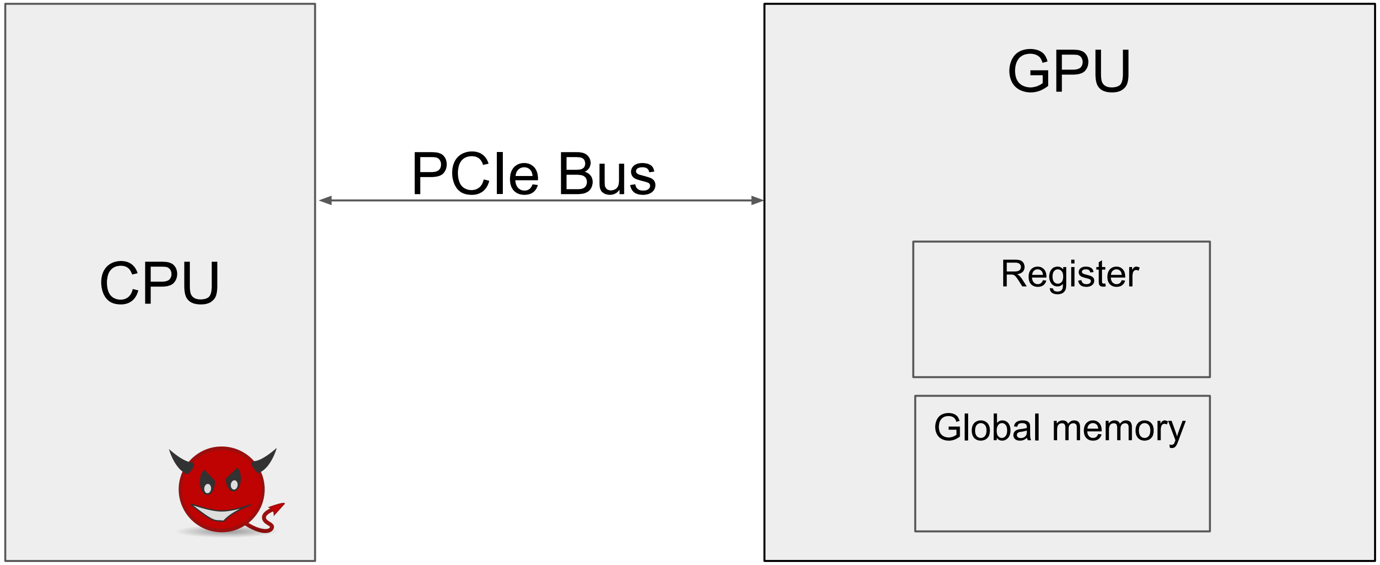


Independent computational resources

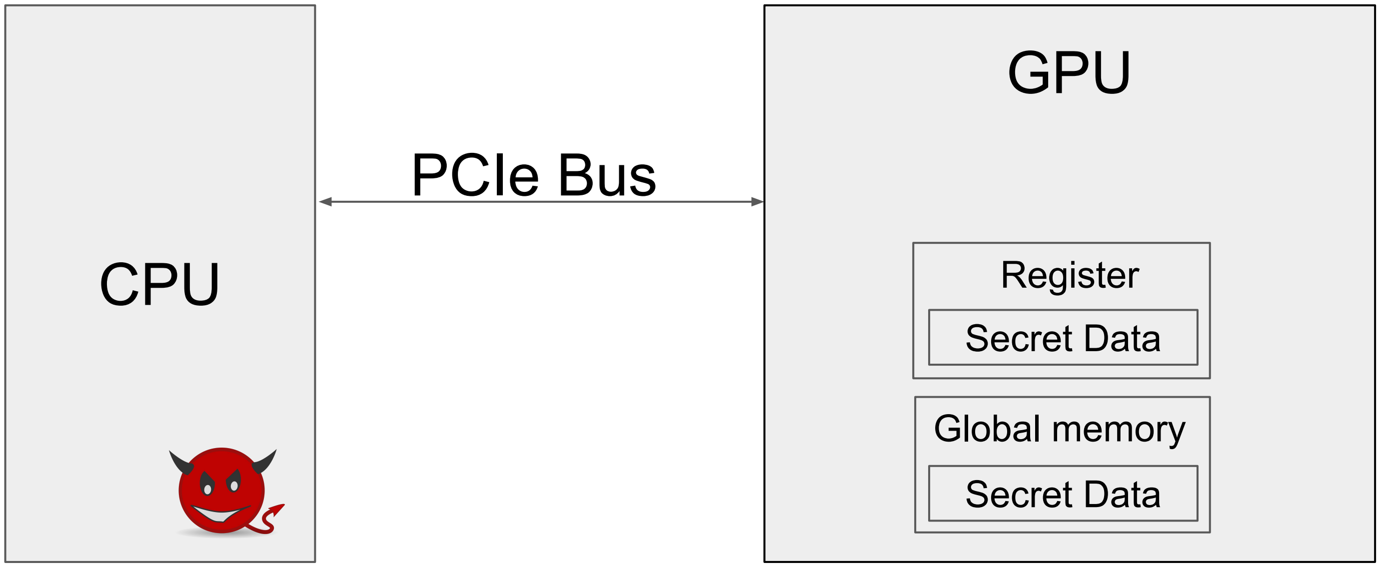
Independent memory system

Physically partitioned from CPU

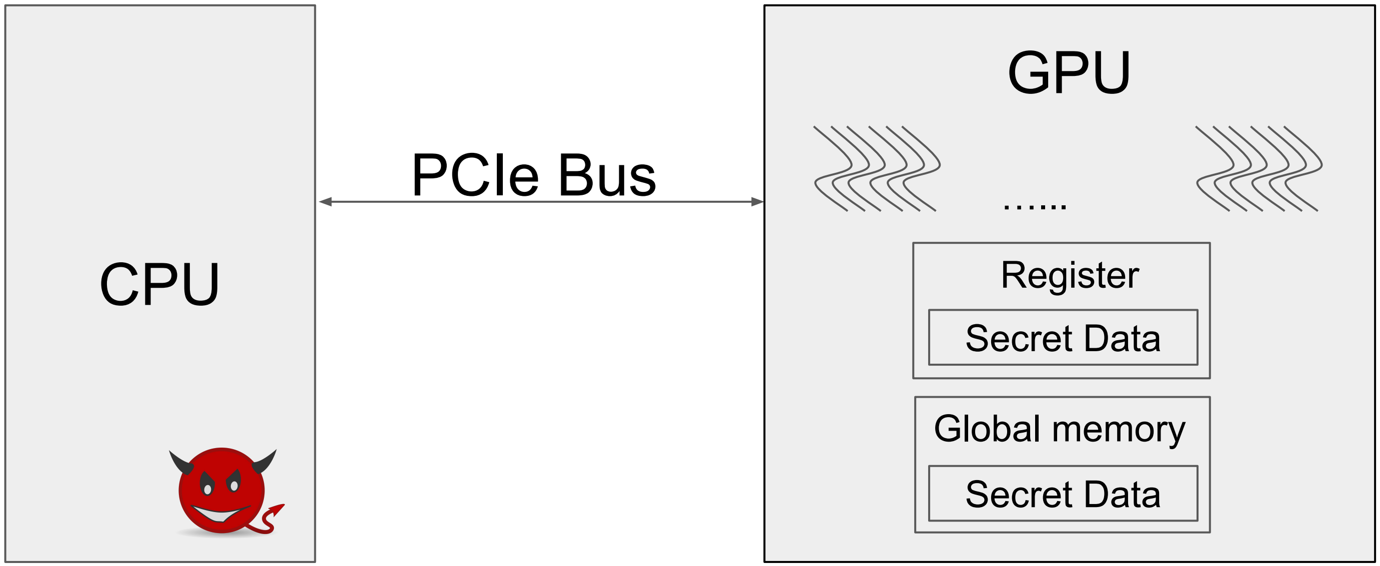
Can discrete GPUs enhance the security of a computing system?



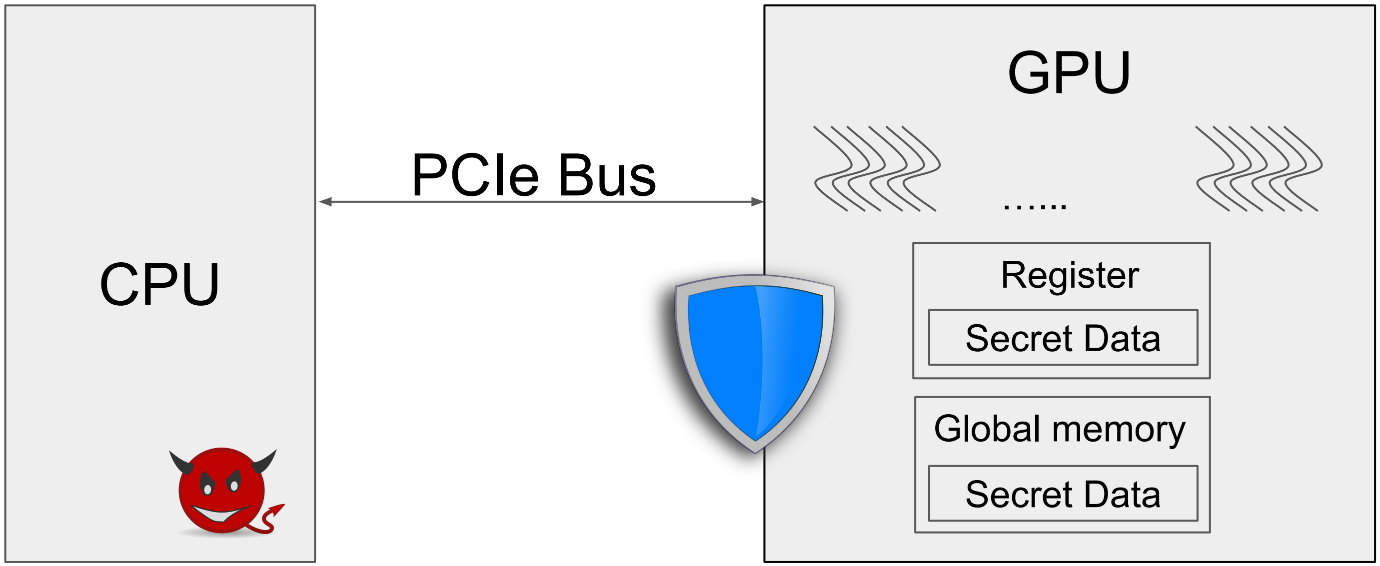
Can discrete GPUs enhance the security of a computing system?



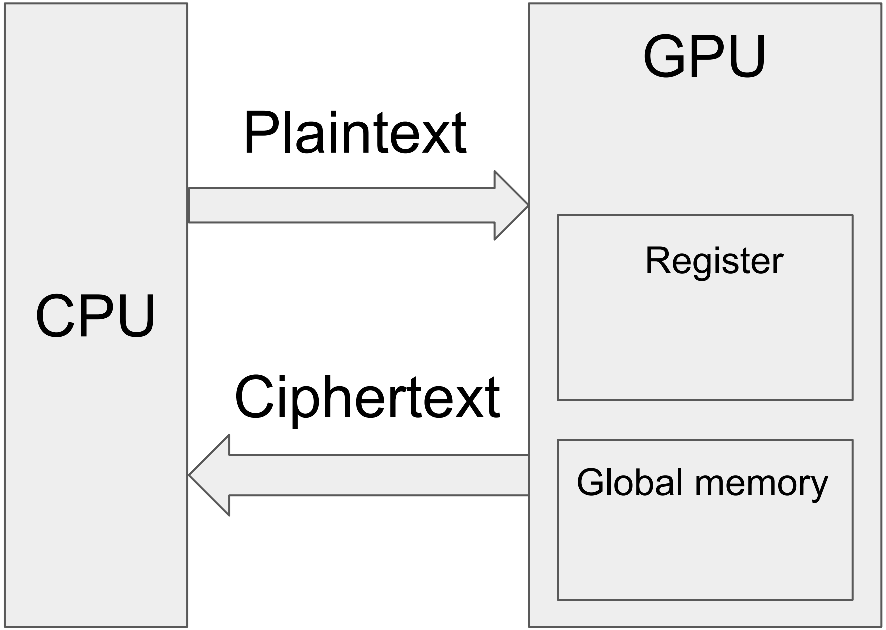
Can discrete GPUs enhance the security of a computing system?



Can discrete GPUs enhance the security of a computing system?

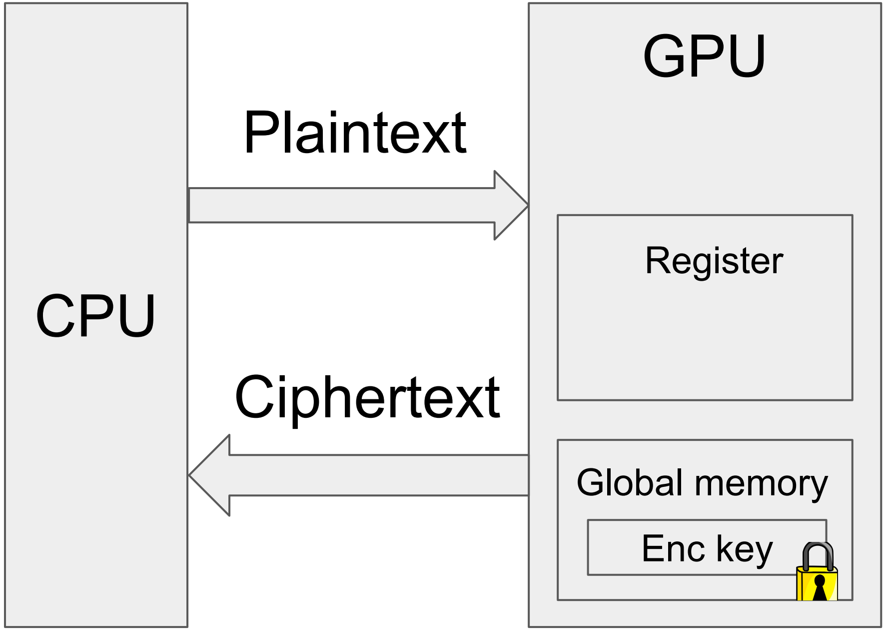


PixelVault (CCS 14)



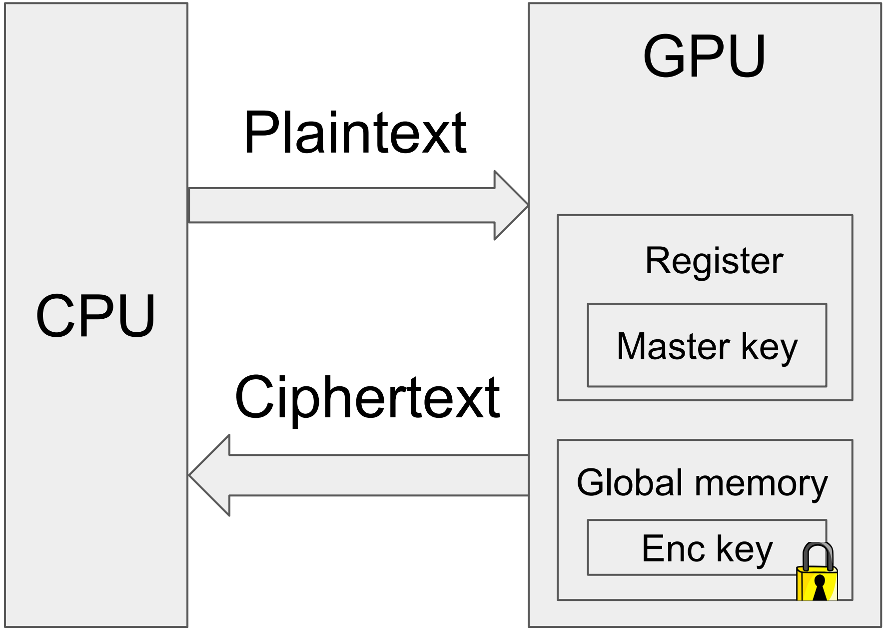
- Runs AES/RSA encryption in GPU.

PixelVault (CCS 14)



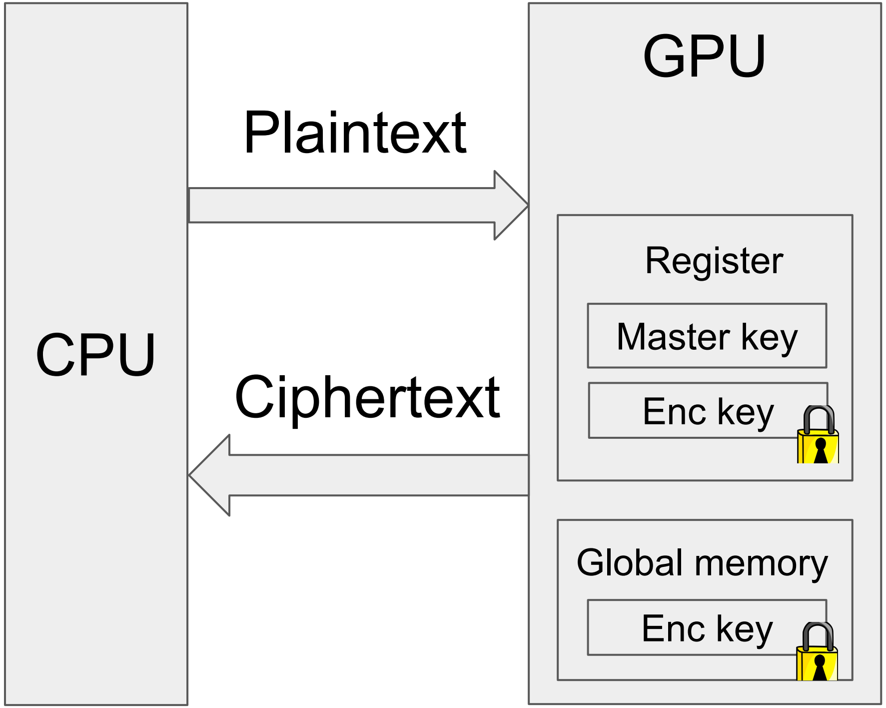
- Runs AES/RSA encryption in GPU.
- Encryption(Enc) keys are encrypted by a master key and are stored in GPU memory.

PixelVault (CCS 14)



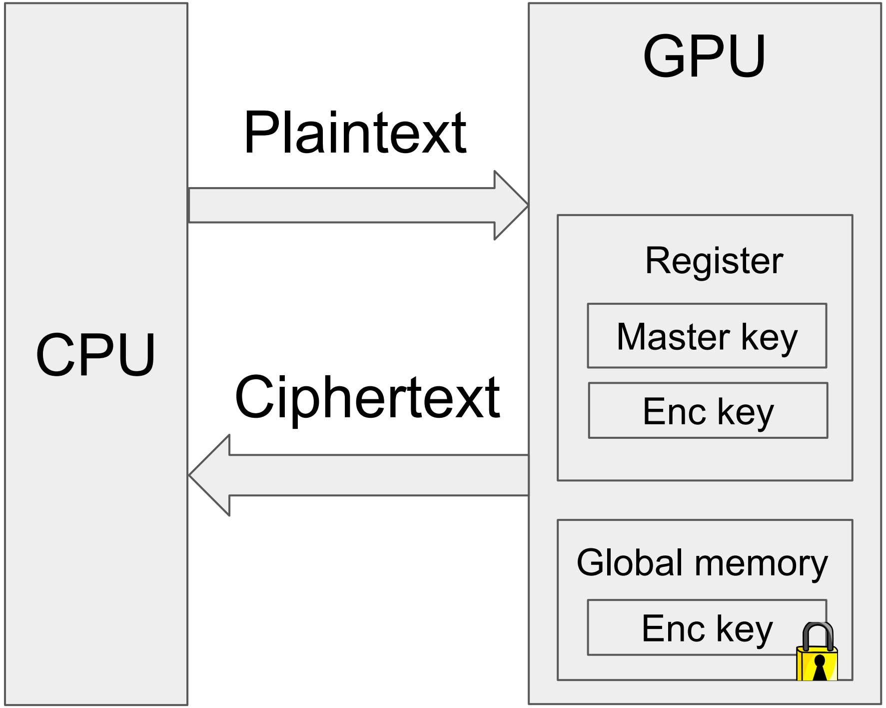
- Runs AES/RSA encryption in GPU.
- Encryption(Enc) keys are encrypted by a master key and are stored in GPU memory.
- Master key is stored in a GPU register.

PixelVault (CCS 14)



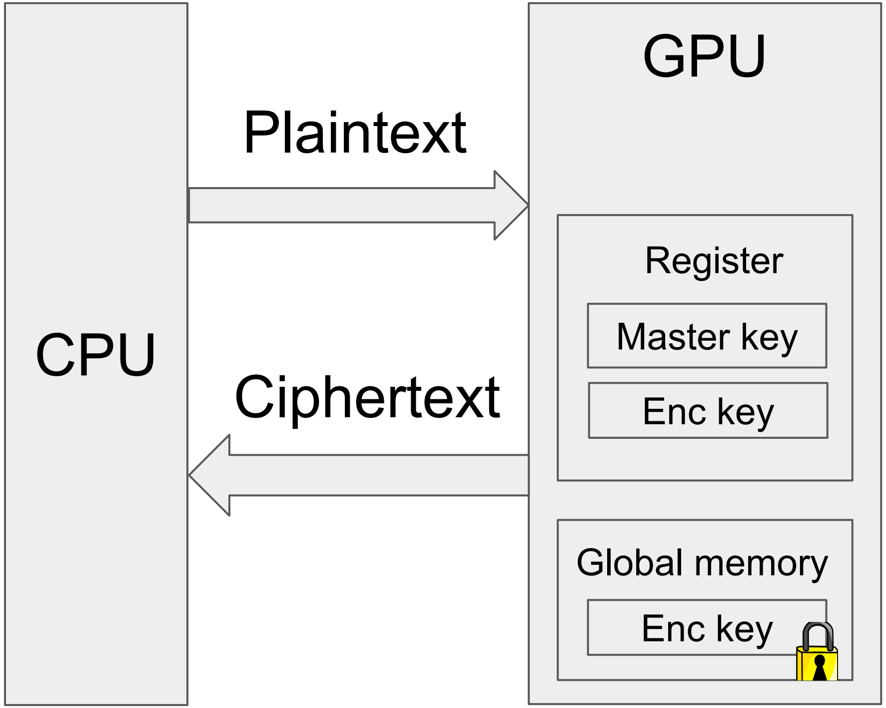
- Runs AES/RSA encryption in GPU.
- Encryption(Enc) keys are encrypted by a master key and are stored in GPU memory.
- Master key is stored in a GPU register.

PixelVault (CCS 14)



- Runs AES/RSA encryption in GPU.
- Encryption(Enc) keys are encrypted by a master key and are stored in GPU memory.
- Master key is stored in a GPU register.

PixelVault (CCS 14)



- Runs AES/RSA encryption in GPU.
- Encryption(Enc) keys are encrypted by a master key and are stored in GPU memory.
- Master key is stored in a GPU register.
- Prevent any adversarial from accessing registers.

Threat model

- System boots from a trusted configuration and sets up PixelVault execution environment on GPU.

Threat model

- System boots from a trusted configuration and sets up PixelVault execution environment on GPU.
- After setup, attacker can have full control over the platform.
  - Execute code at any privilege.
  - Has access to all platform hardware.
- Attack goal: Steal keys from GPU.

Threat model

Security guarantees depend on several NVIDIA GPU characteristics.

- Some of these characteristics are well known and confirmed.
- Some are experimentally validated.
- Others are only assumed to correct.
    - Experimentally verify.

Assumption about NVIDIA GPU

Assumption	PixelVault safety property	Attack
A running GPU kernel cannot be stopped and debugged.	Secure register contents from CPU-based debugger.	Debugger API.
GPU registers can’t be read after kernel termination.	Cannot get the master key after kernel termination.	Concurrent kernel.
Can’t replace code of GPU kernel executing from instruction cache.	Cannot replace PixelVault code without stopping the kernel.	Flush instruction cache using MMIO registers.

Assumption: A running GPU kernel cannot be stopped and debugged.

CUDA 4.2	CUDA 5.0 and newer
● Compiled with explicit debug support. ● Insert breakpoints before kernel is running.	Stop a running kernel and inspect all GPU registers via debugger API.

Assumption: A running GPU kernel cannot be stopped and debugged.

CUDA 4.2	CUDA 5.0 and newer
● Compiled with explicit debug support. ● Insert breakpoints before kernel is running.	Stop a running kernel and inspect all GPU registers via debugger API.

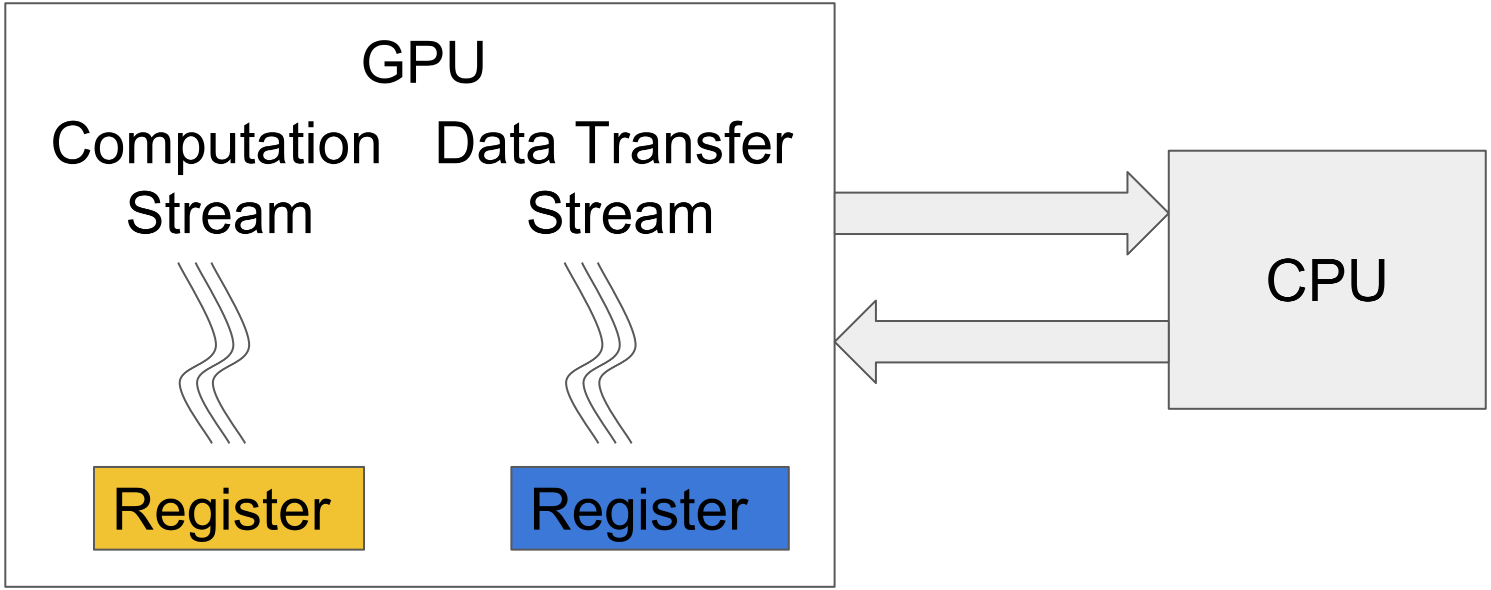
Assumption about NVIDIA GPU

Assumption	PixelVault safety property	Attack
A running GPU kernel cannot be stopped and debugged.	Secure register contents from CPU-based debugger.	Debugger API.
GPU registers can’t be read after kernel termination.	Cannot get the master key after kernel termination.	Concurrent kernel.
Can’t replace code of GPU kernel executing from instruction cache.	Cannot replace PixelVault code without stopping the kernel.	Flush instruction cache using MMIO registers.

CUDA Stream

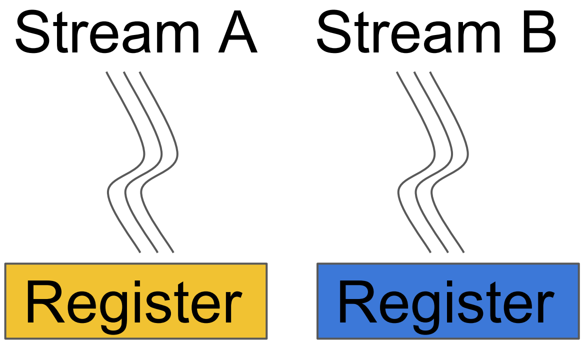
- An operation sequence on a GPU device.
- Every CUDA kernel is invoked on an independent stream.
- Share the same address space.

PixelVault


GPU
Computation Stream
Data Transfer Stream
Register
Register
CPU

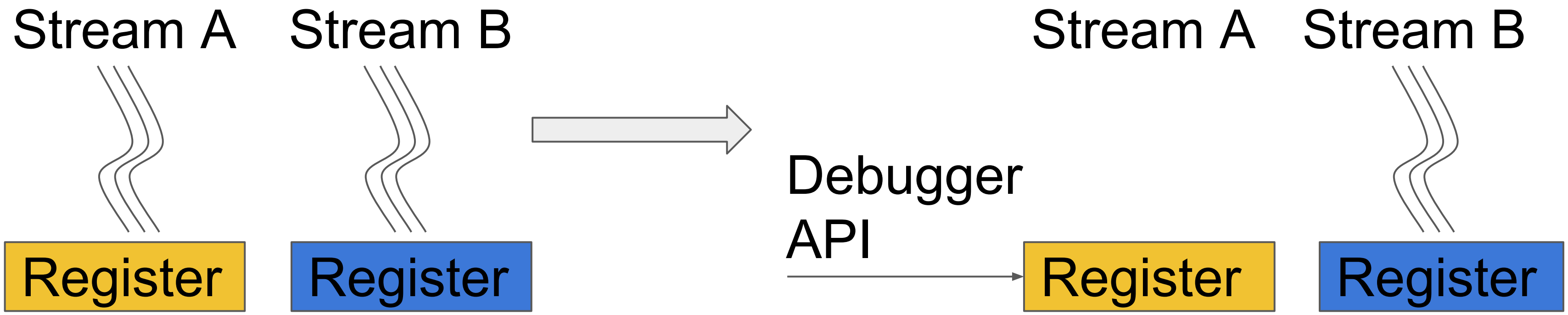
Assumption: GPU registers can’t be read after kernel termination.

Attack:

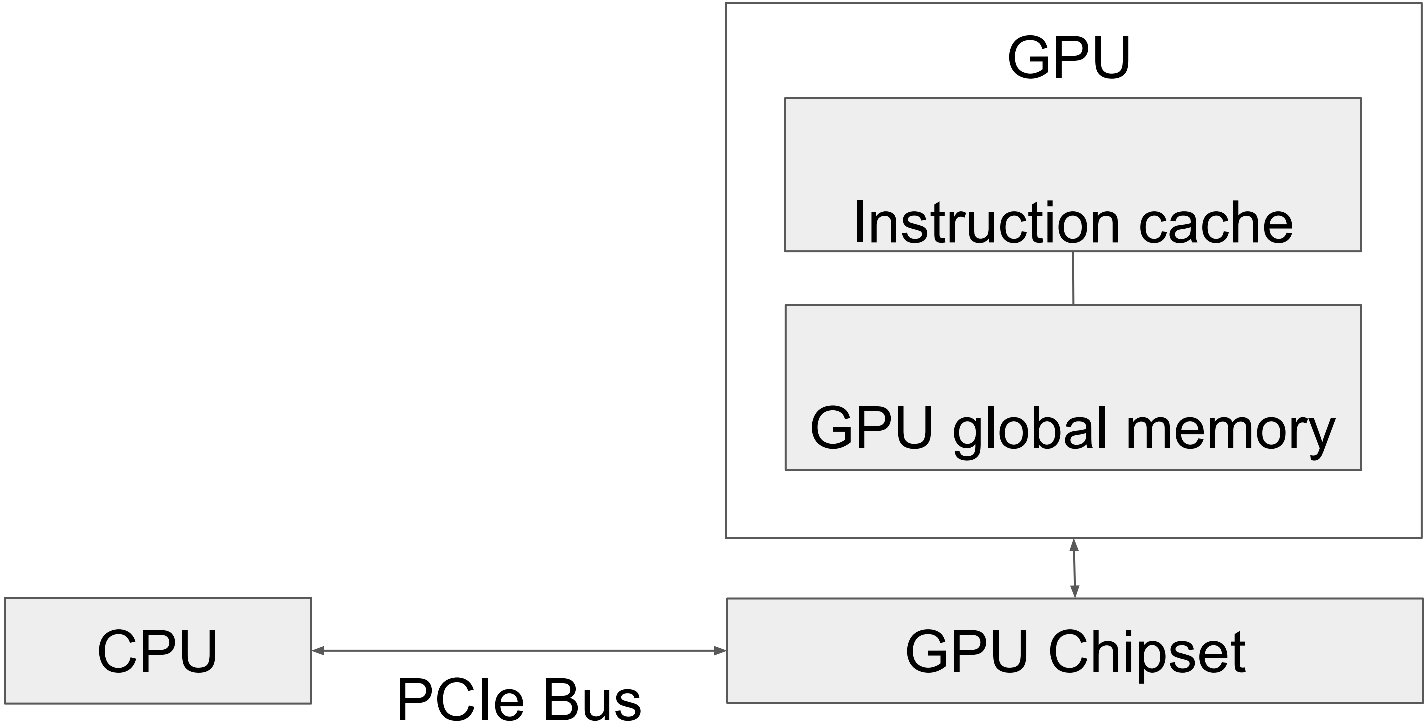


Assumption: GPU registers can’t be read after kernel termination.

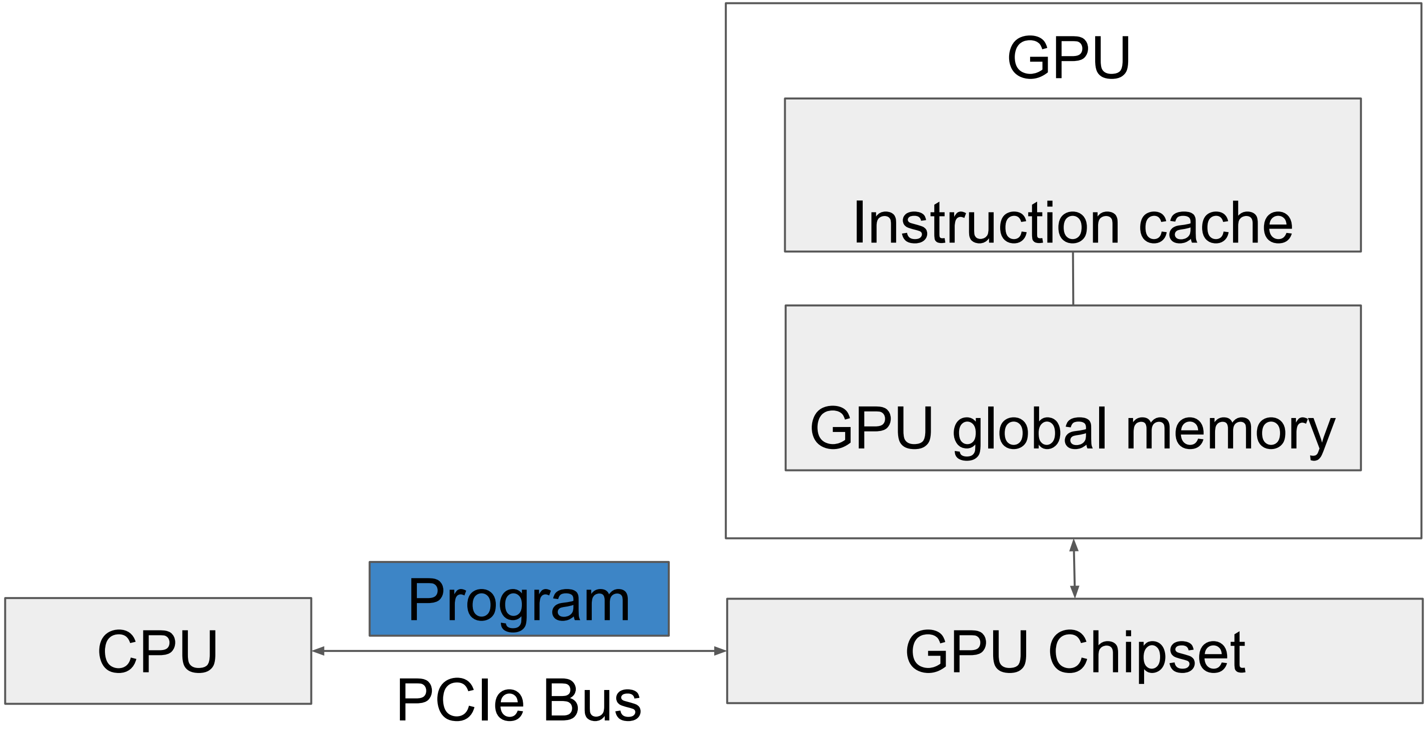
Attack: If GPU kernel B is invoked in parallel with running kernel A, A’s register state can be retrieved using the debugger API even after A terminates, as long as B is still running.



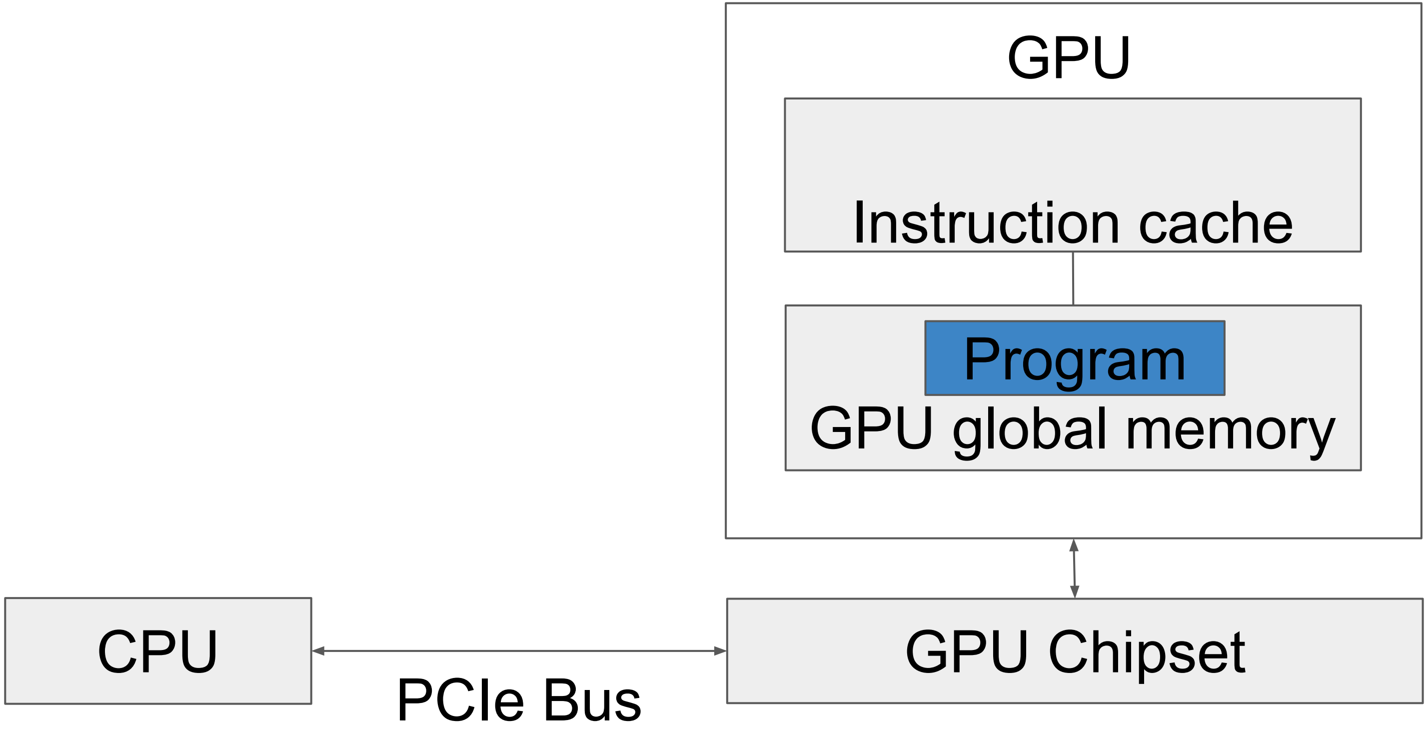
Loading a program into the GPU



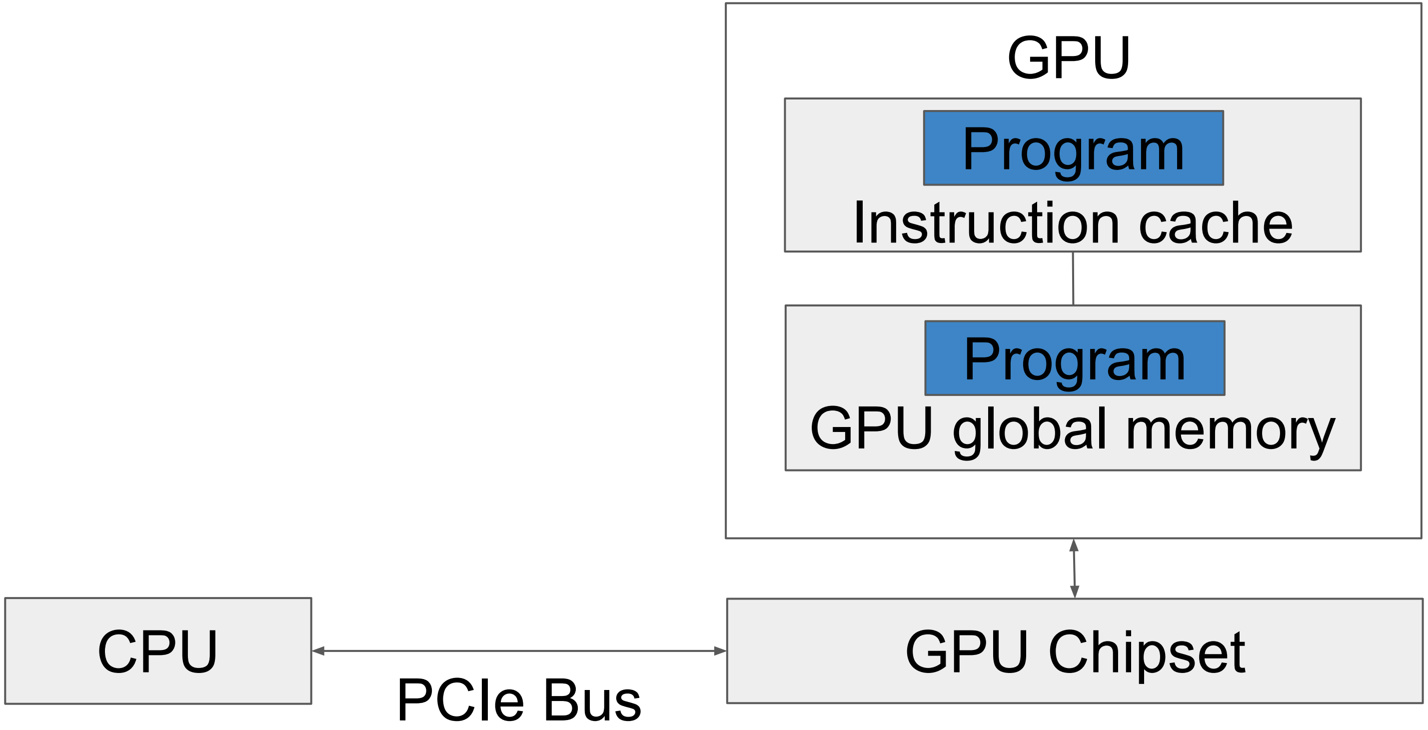
Loading a program into the GPU



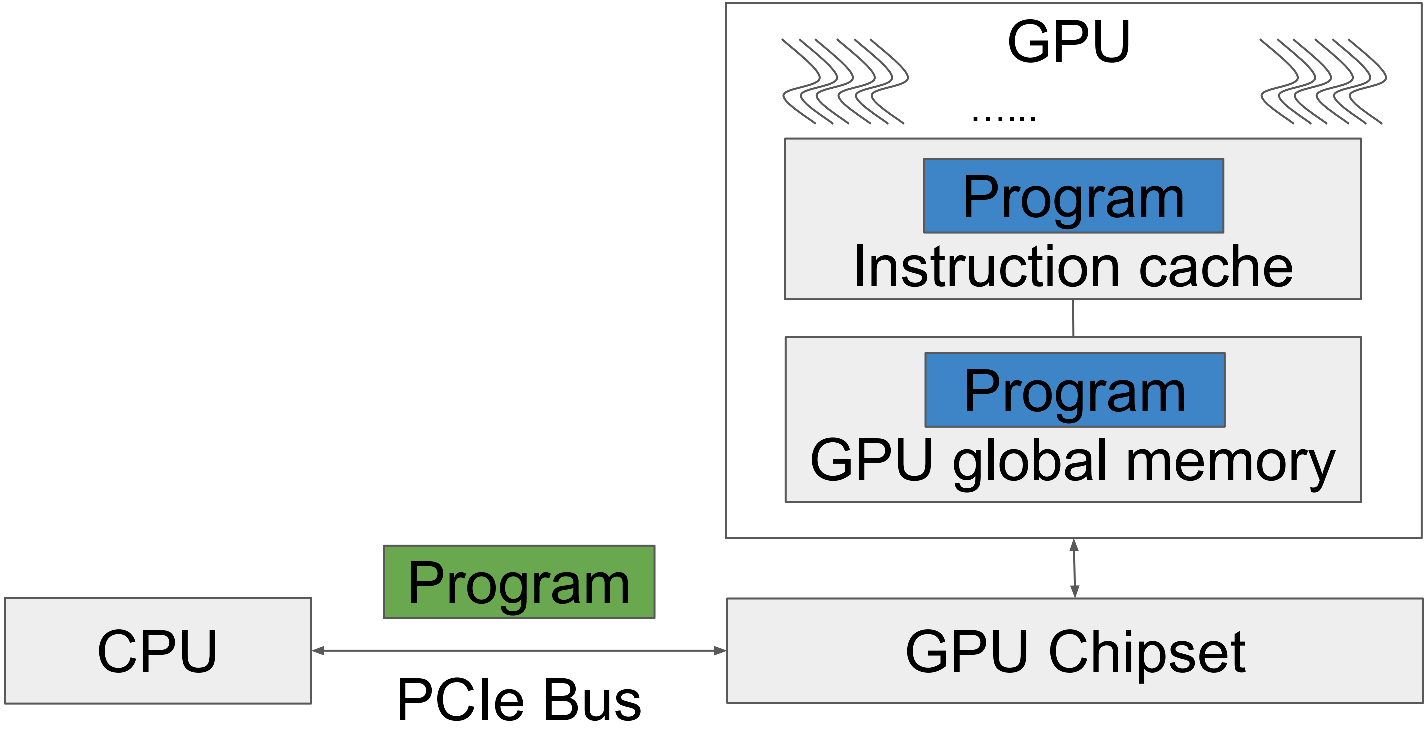
Loading a program into the GPU



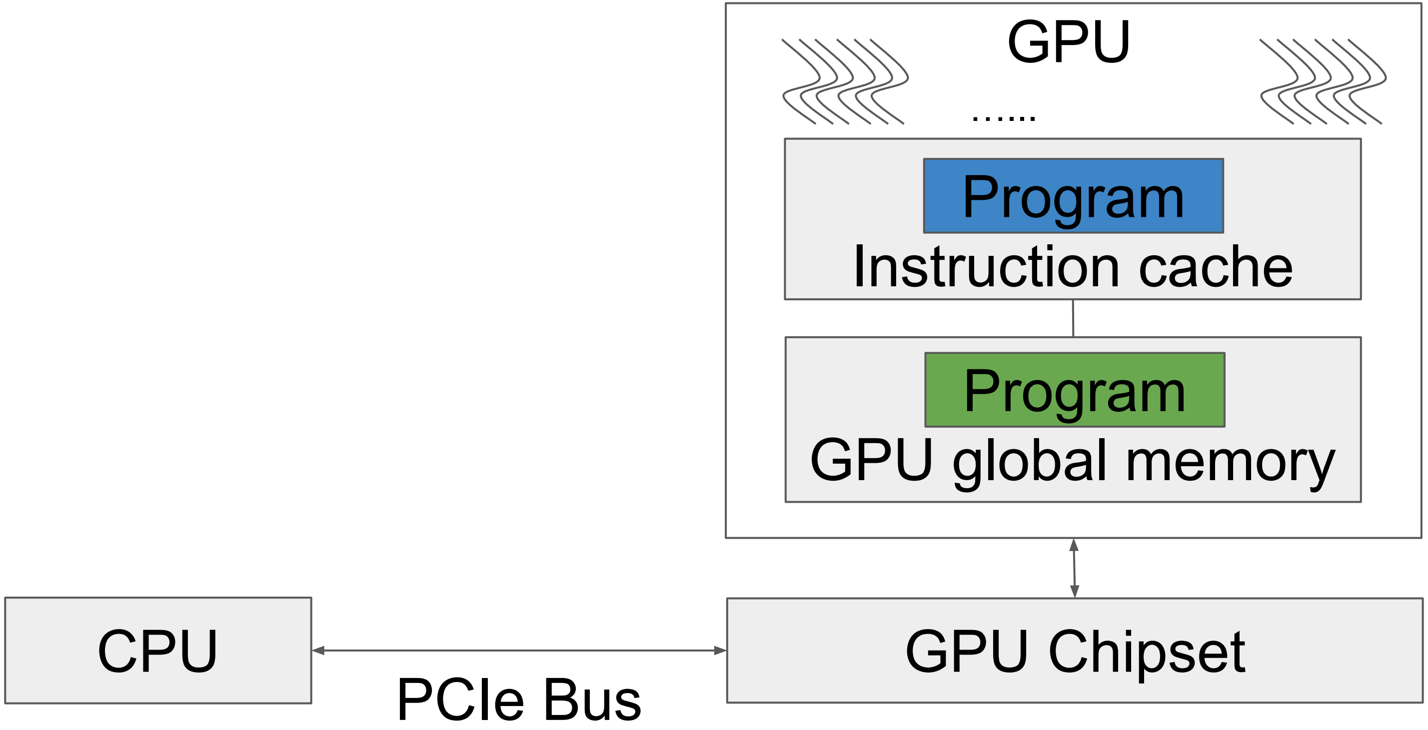
Loading a program into the GPU



If CPU writes to GPU instructions in memory while the GPU is running



If CPU writes to GPU instructions in memory while the GPU is running



No public API for flushing the instruction cache.

Assumption about NVIDIA GPU

Assumption	PixelVault safety property	Attack
A running GPU kernel cannot be stopped and debugged.	Secure register contents from CPU-based debugger.	Debugger API.
GPU registers can’t be read after kernel termination.	Cannot get the master key after kernel termination.	Concurrent kernel.
Can’t replace code of GPU kernel executing from instruction cache.	Cannot replace PixelVault code without stopping the kernel.	Flush instruction cache using MMIO registers.

Discussion

- Security guarantees rely on proprietary hardware and software which is poorly (often purposefully) publicly documented.
  - Some MMIO registers that flush the GPU instruction cache are not documented as flushing the cache.
  - Private debugger API.

Discussion

- Security guarantees rely on proprietary hardware and software which is poorly (often purposefully) publicly documented.
- Manufacturers are free to change what’s implemented in software and what’s implemented in hardware across generations.
  - Debugger API

Discussion

- Security guarantees rely on proprietary hardware and software which is poorly (often purposefully) publicly documented.
- Manufacturers are free to change what’s implemented in software and what’s implemented in hardware across generations.
- Manufacturers can change the architecture that invalidates the security of systems based on GPU.

Discussion

- Security guarantees rely on proprietary hardware and software which is poorly (often purposefully) publicly documented.
- Manufacturers are free to change what’s implemented in software and what’s implemented in hardware across generations.
- Manufacturers can change the architecture that invalidates the security of systems based on GPU.
- Discrete GPUs cannot enhance the security of the computing system.

GPU as a host for stealthy malware

1. Threat Model
2. GPU driver attack
3. GPU microcode attack
4. IOMMU mitigation

Threat model

Attacker:

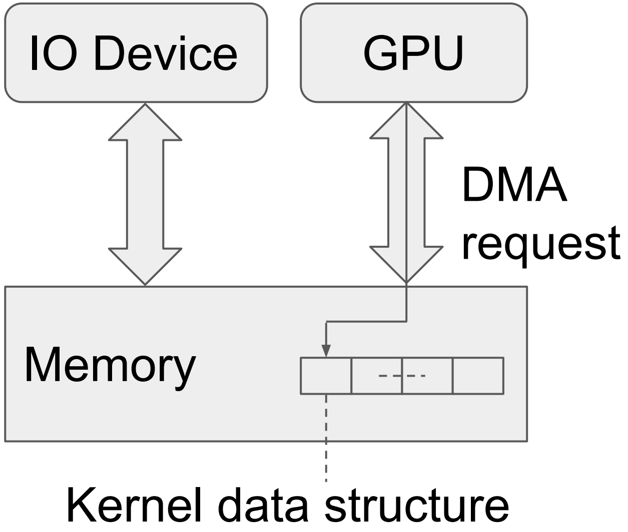
- Load and unload kernel modules via module loading capability.
- Access the GPU control interface i.e., MMIO register regions.
- Loses the module loading capability and is allowed only unprivileged access after the malware is installed.

Stealthiness

- Originate with the GPU reading and writing CPU memory.

DMA attack

- GPU is a programmable device.
- Easier to launch DMA attack compared to other DMA capable devices.
- GPU driver attack.
- GPU microcode attack.

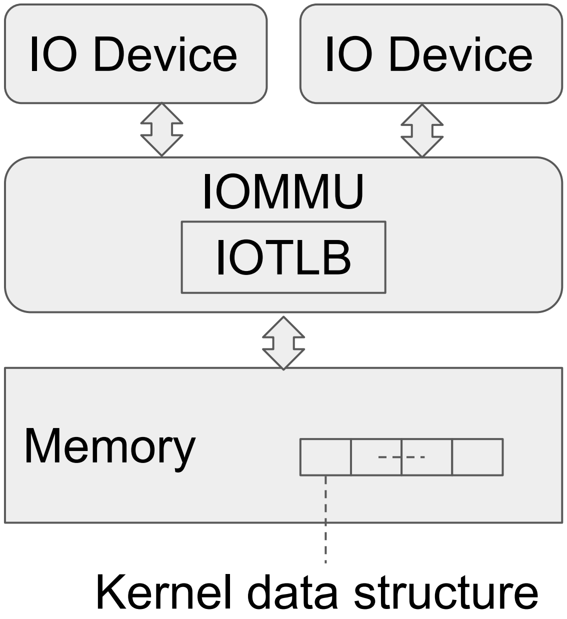


Device address = Physical address

IOMMU

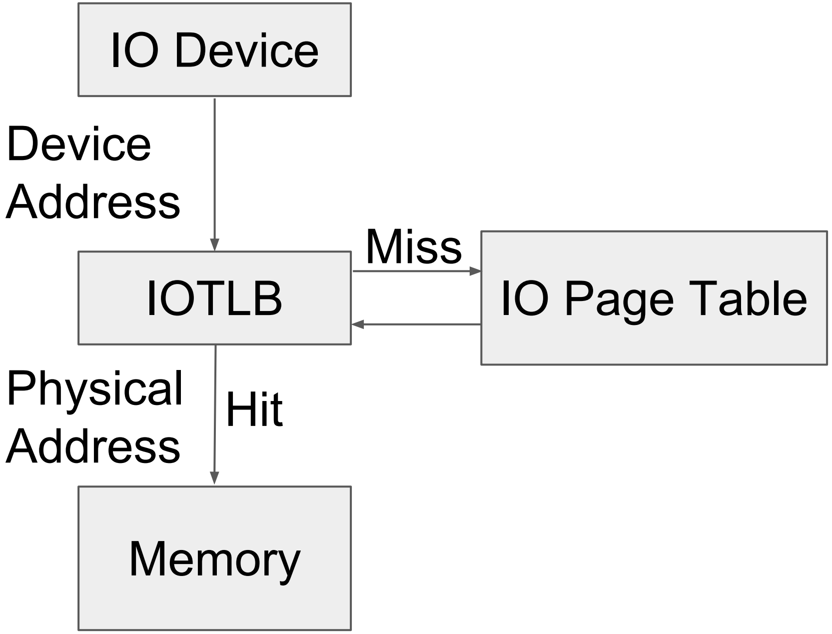
- Hardware
- Software management
- IOMMU attack

IOMMU



- Maps device addresses to CPU physical addresses.
- Check access permission.

IOTLB



- Not kept coherent with the IO page table by hardware.
- Software must explicitly flush the cached mappings when they are removed from the IO page table.

IOMMU configurations

Not secure

Security

Secure

Mode	Characteristics
Disable	• Default configuration for many linux distributions. • Reduce IO performance. • Incompatible with certain devices and features.
Pass through	• Hardware IOMMU is turned off. • Device address is used as CPU physical address.
Deferred	Default mode when IOMMU enabled.
Strict	IOMMU enabled.

Fast

Performance

Slow

IOMMU configurations

Not secure

Fast

Security

Performance

Mode	Characteristics
Disable	● Default configuration for many linux distributions. ● Reduce IO performance. ● Incompatible with certain devices and features.
Pass through	● Hardware IOMMU is turned off. ● Device address is used as CPU physical address.
Deferred	Default mode when IOMMU enabled.
Strict	IOMMU enabled.

Secure

Slow

IOMMU configurations

Not secure

Security

Secure

Fast

Performance

Slow

Mode	Characteristics
Disable	• Default configuration for many linux distributions. • Reduce IO performance. • Incompatible with certain devices and features.
Pass through	• Hardware IOMMU is turned off. • Device address is used as CPU physical address.
Deferred	Default mode when IOMMU enabled.
Strict	IOMMU enabled.

When system memory is unmapped from IO devices:

Clear the entry in IO page table

When system memory is unmapped from IO devices:

Clear the entry in IO page table

	IOTLB Flush	
	Deferred Mode	Strict Mode
Strategy	Flush entire IOTLB.	Flush individual entry in given domain.
Timing	When deferred list is full or 10 ms after the first entry, whichever comes first.	Immediately after unmapping entry from IO page table.

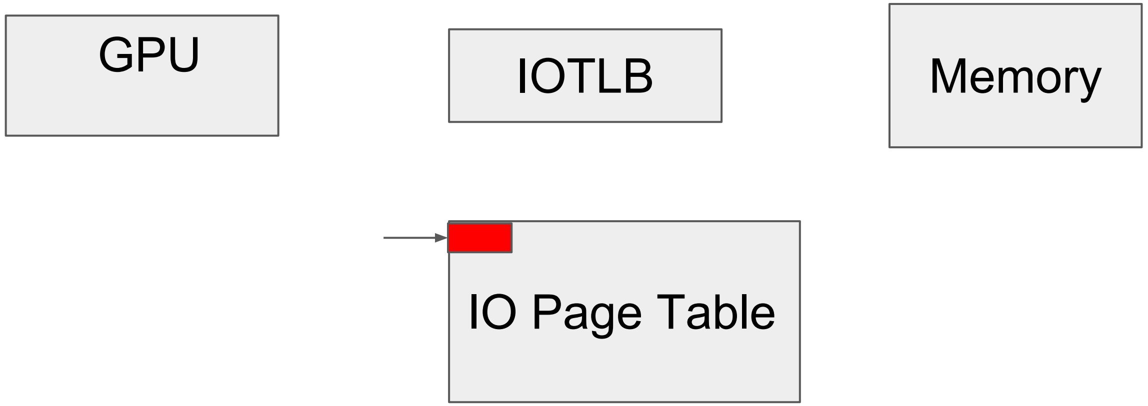
When system memory is unmapped from IO devices:

Clear the entry in IO page table

	IOTLB Flush	
	Deferred Mode	Strict Mode
Strategy	Flush entire IOTLB.	Flush individual entry in given domain.
Timing	When deferred list is full or 10 ms after the first entry, whichever comes first.	Immediately after unmapping entry from IO page table.

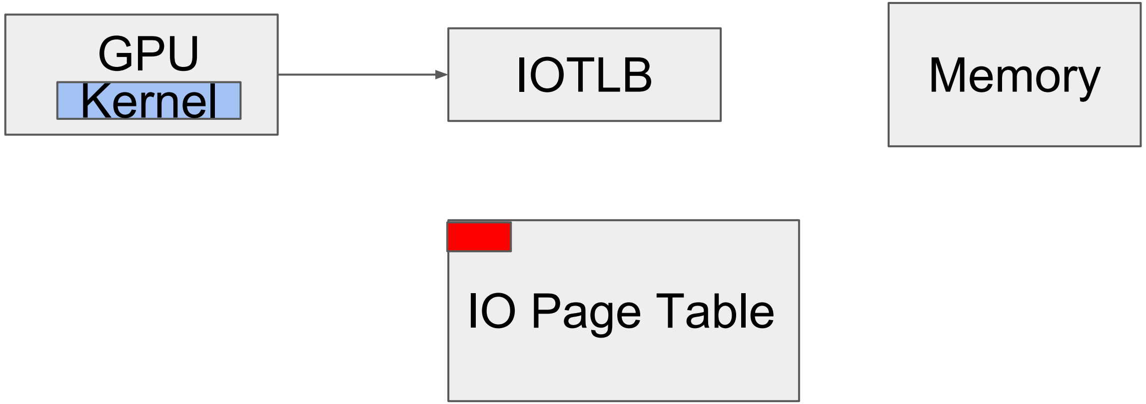
IOMMU attack

1. Writes a malicious IO page table entry.



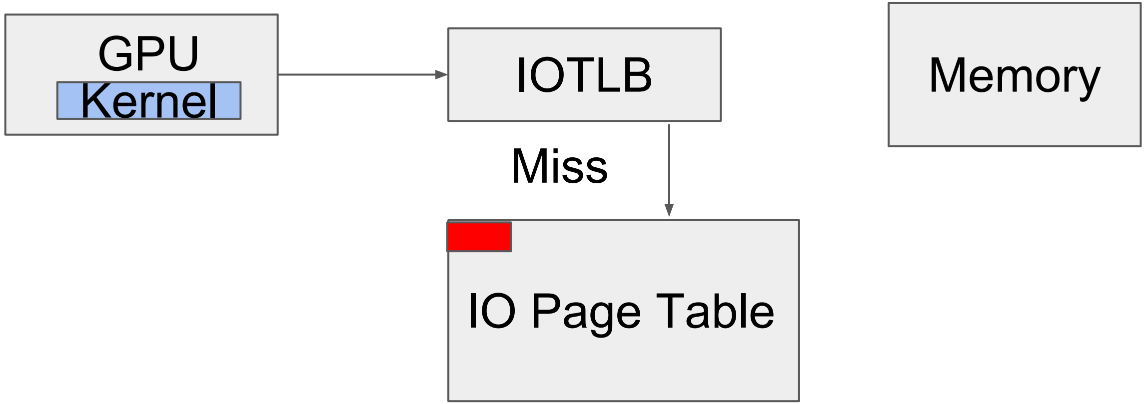
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.



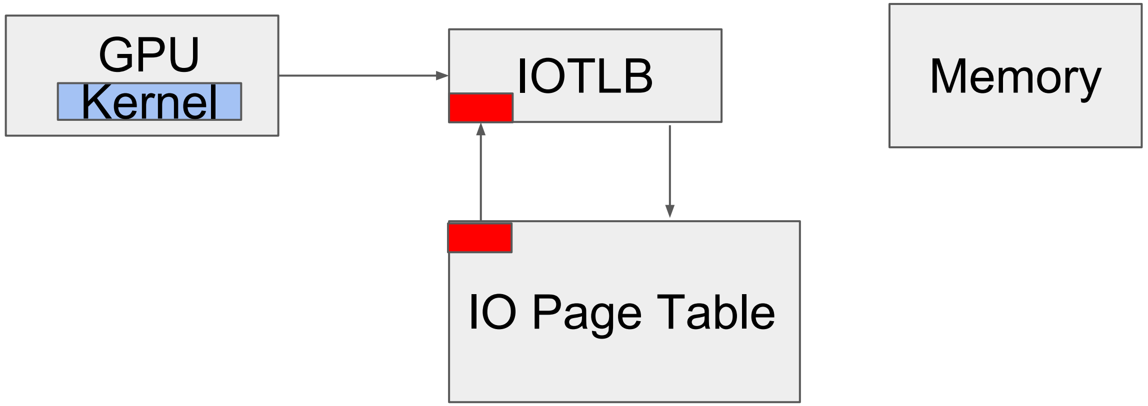
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.



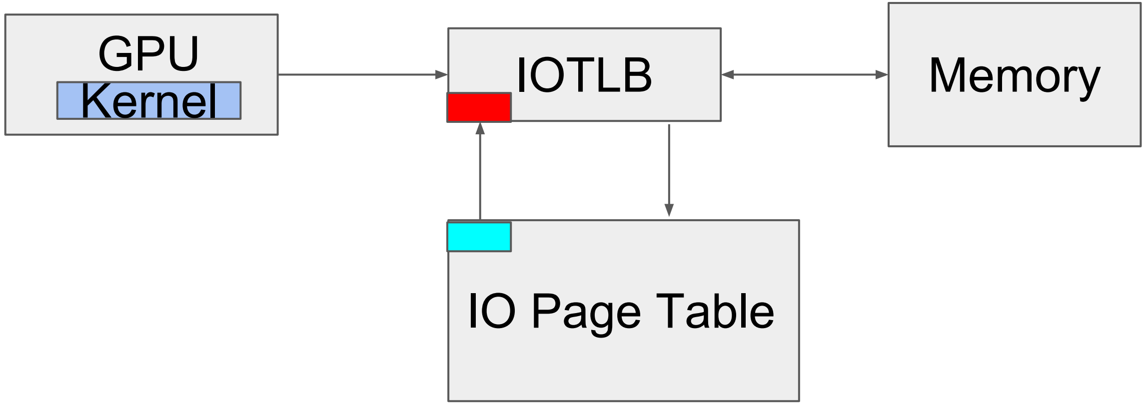
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.



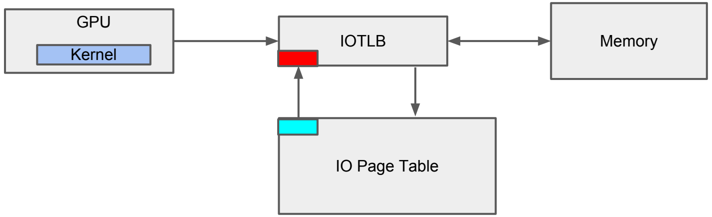
IOMMU attack

1. Writes a malicious IO page table entry.
2. Launch a GPU kernel which accesses the device address of the mapping, causing the entry to be cached in IOTLB.
3. Overwrite the IO page table.



How long can a stale entry last in IOTLB?

Workload	Bit rate	Stale period
Idle ssh connection	10 bps	1 day
Web radio	130 Kbps	1 hour
Web video: Auto (480p)	2 Mbps	1 min



Stealthiness

- IOTLB entry is not accessible by software.
- IO page table can be monitored by security tools.

Conclusion

- Discrete GPUs are not an appropriate choice for a secure coprocessor.
- Discrete GPUs pose a security threat to computing platform.