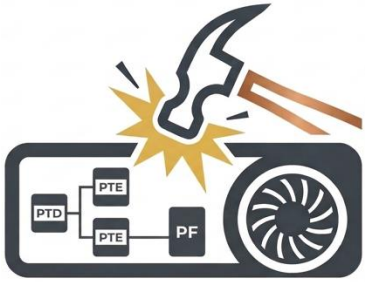


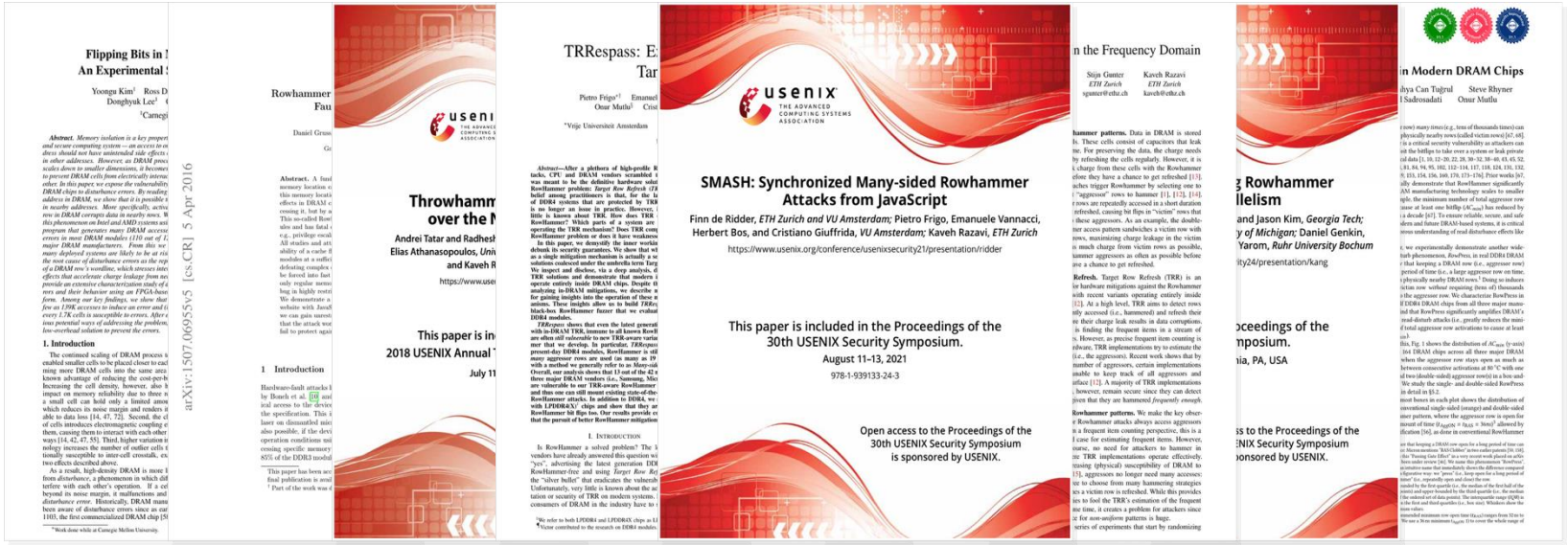
GeForge: Hammering GDDR Memory to Forge GPU Page Tables for Fun and Profit

Junpeng Wan[1,2], Yanan Guo[3], Zhi Zhang[4], Zhuo Li[5], Dave (Jing) Tian[2], Zhenkai Zhang[1]

[1]*Clemson University* [2]*Purdue University* [3]*University of Rochester* [4]*University of Western Australia* [5]*HydroX AI*

Revisiting Rowhammer on the GPU

- Rowhammer
  - Modify memory content without access
  - Extensive studies on CPU DRAM



Rowhammer (ISCA'14) | Rowhammer.j (DIMVA'16) | Throwhamme (ATC'18) | TRRespass (S&P'20) | SMASH (Sec'21 | Blacksmith (S&P'22) | Sledgehamme (Sec'24) | RowPress (ISCA'23)

Revisiting Rowhammer on the GPU

- Rowhammer
  - Modify memory content without access
  - Extensive studies on CPU DRAM

- Increasing interest in hammering GPU
  - GPUHammer (Sec'25)
  - GDDRHammer / GPUBreach (This Session)

RTX A6000 (Source: NVIDIA.com)

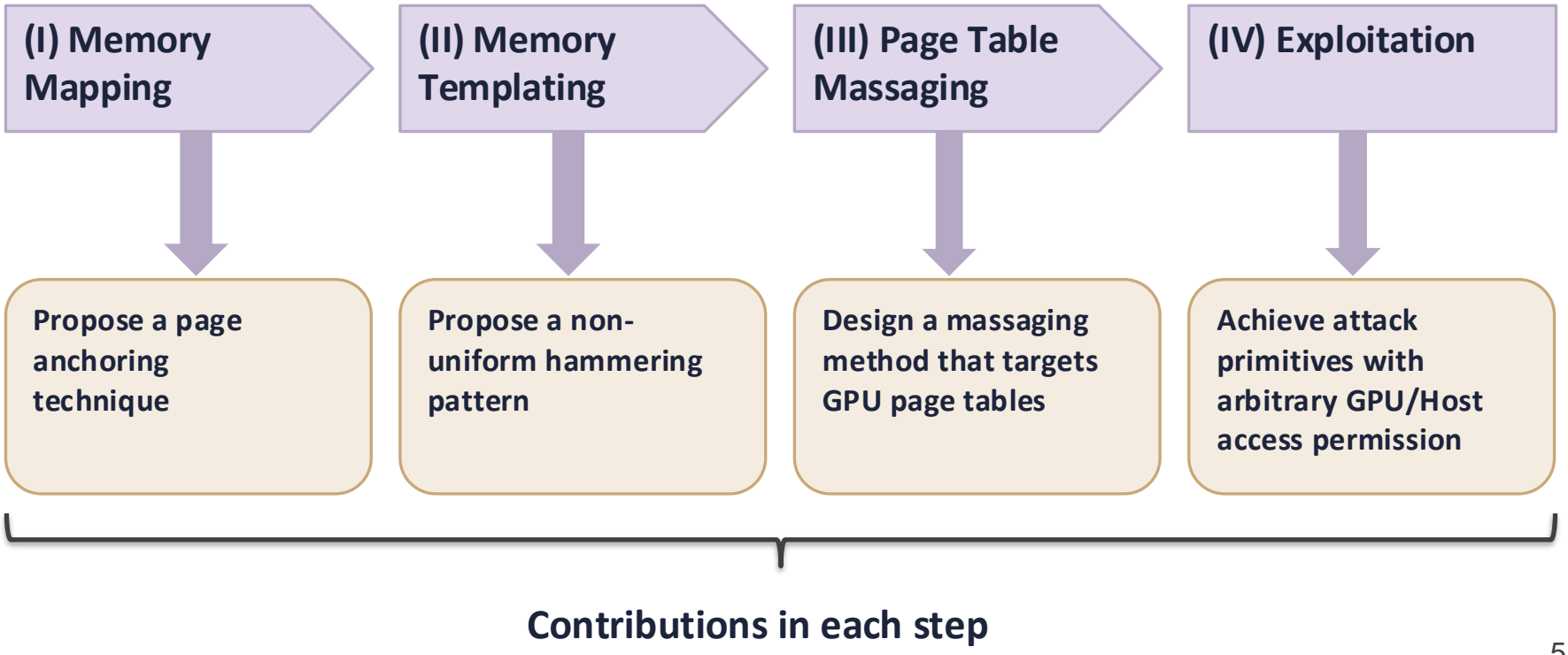
GeForge Highlights

- **High-level idea**
  - Targets GPU page table
  - Achieve arbitrary read/write permission on GPU/host memory
- **Contributions**
  - Better hammering pattern
  - Bit flips on *GeForce RTX 3060* – a widely deployed desktop GPU
  - Page anchoring & page table massaging technique

GeForce RTX 3060 (Source: ebay.com)

GeForge Workflow

- Turning a single bit flip into an end-to-end exploit



(I) Memory Mapping

- **Goal: Map Virtual Address (VA) to DRAM Bank/Row of a huge *cudaMalloc()* buffer**

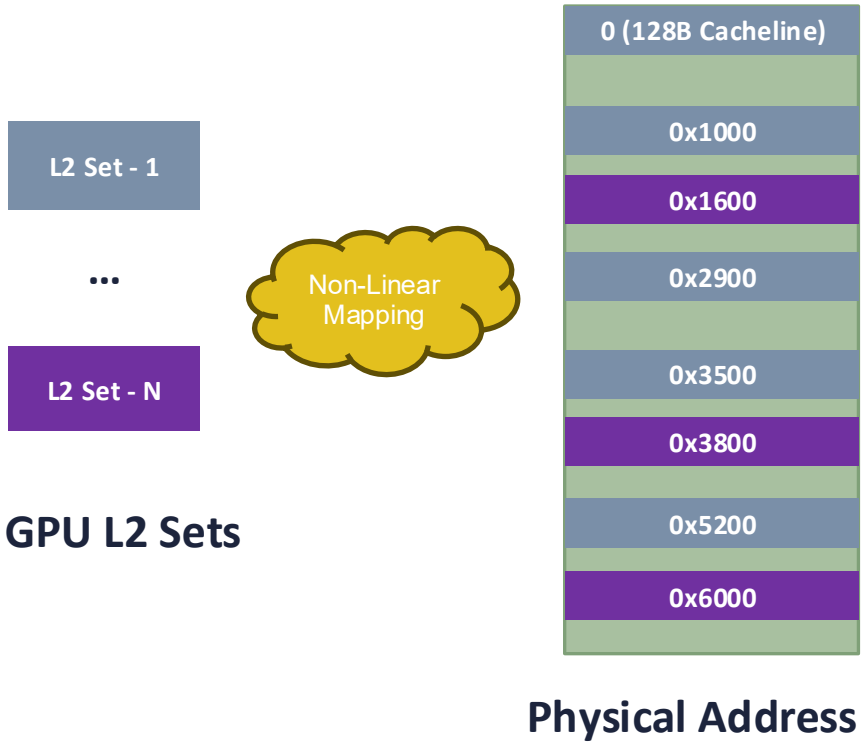
- **Previous approach (Mapping VA <-> DRAM directly)**
  - **Assumes the physical address of the *cudaMalloc()* buffer stays the same**
  - **Returned physical address offset changes with OS/Driver/CPU**
  - **Need to re-profile the whole mapping**

Detect Row Buffer conflicts

(I) Memory Mapping

- ***Page anchoring technique***
  - Resolving virtual-to-physical address mapping at runtime

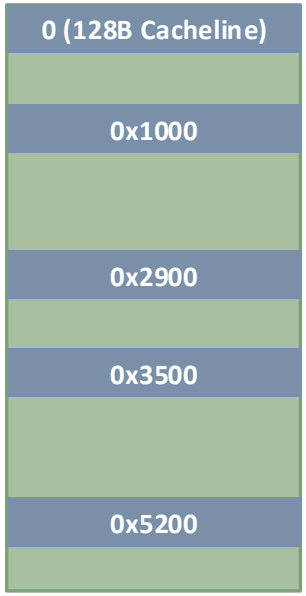
- Key insights
  - Observation: the mapping between GPU L2 set and physical address is non-linear



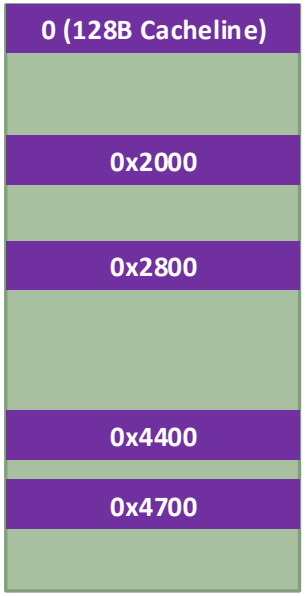
(I) Memory Mapping

- ***Page anchoring technique***
  - Resolving virtual-to-physical address mapping at runtime

- Key insights
  - Observation: the mapping between GPU L2 set and physical address is non-linear
  - Eviction Set (ES) of the first cacheline as a fingerprint for different pages
  - Invalidate+Compare (Sec'24)



Page A - ES

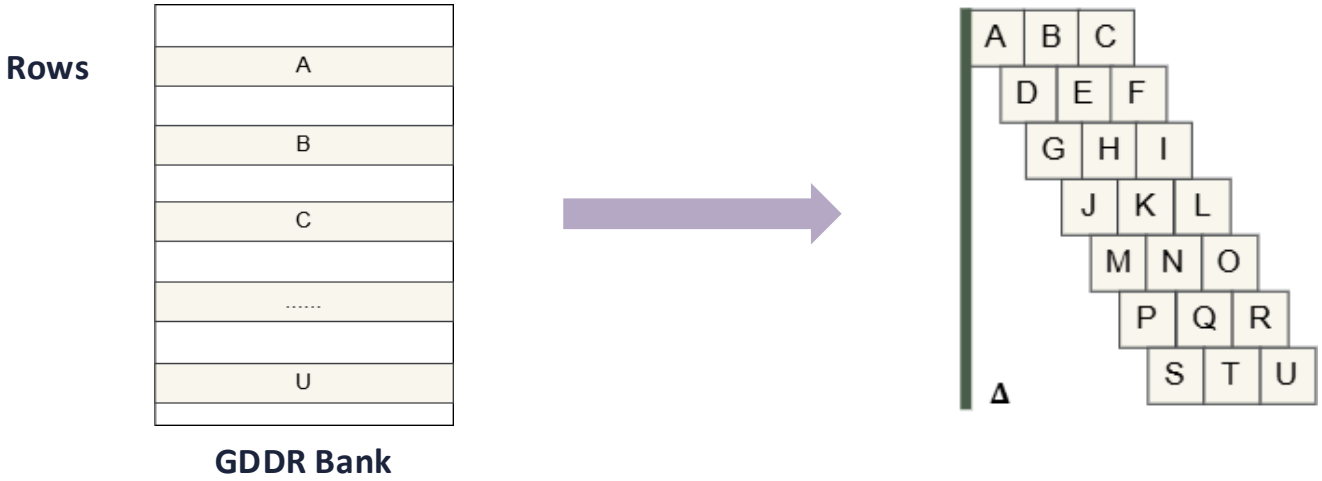


Page B - ES

(II) Memory Templating

- **Goal: Find the bit flip locations**

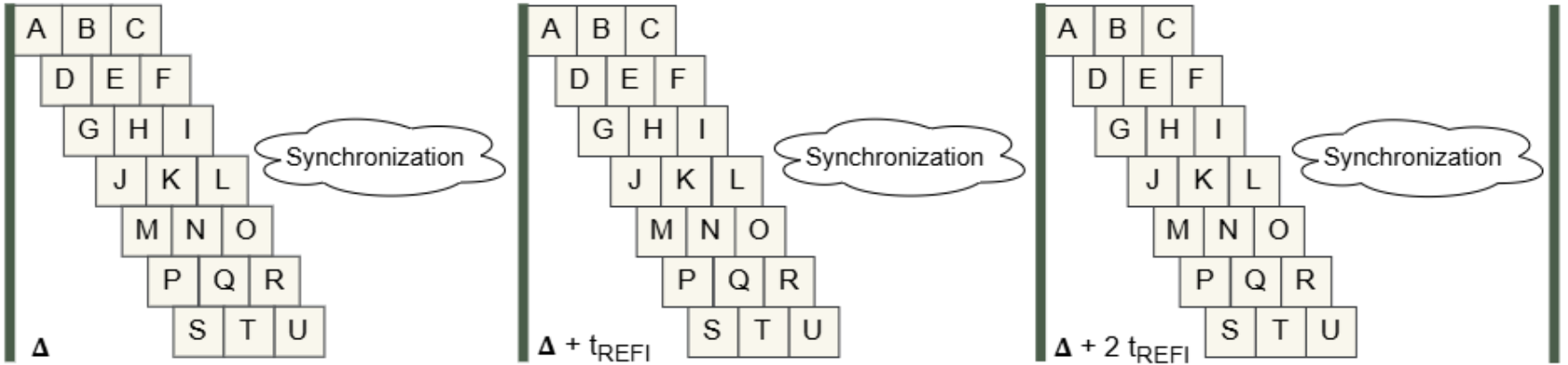
- **(a) Allocate a huge memory buffer with *cudaMalloc()***
- **(b) Activate aggressor rows to trigger bit flips**



(II) Memory Templating

- **Goal: Find the bit flip locations**

- **(a) Allocate a huge memory buffer with *cudaMalloc()***
- **(b) Activate aggressor rows to trigger bit flips**

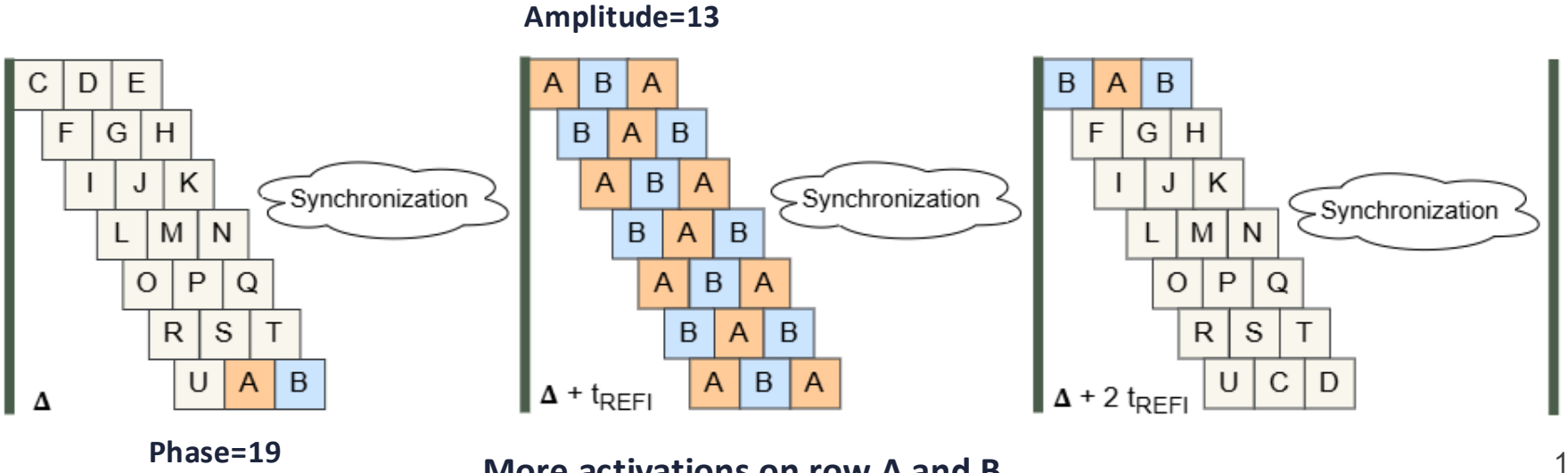


Not efficient in finding bit flips

(II) Memory Templating

- **Goal: Find the bit flip locations**

- **(a) Allocate a huge memory buffer with *cudaMalloc()***
- **(b) Activate aggressor rows to trigger bit flips - *Non-uniform pattern***



(II) Memory Templating

- **Goal: Find the bit flip locations**

- **(a) Allocate a huge memory buffer with *cudaMalloc()***
- **(b) Activate aggressor rows to trigger bit flips - *Non-uniform pattern***
- **(c) Scan and log the bit flip locations**

- ***Found over 1,100 bit flips on RTX 3060!***

(III) Page Table Massaging

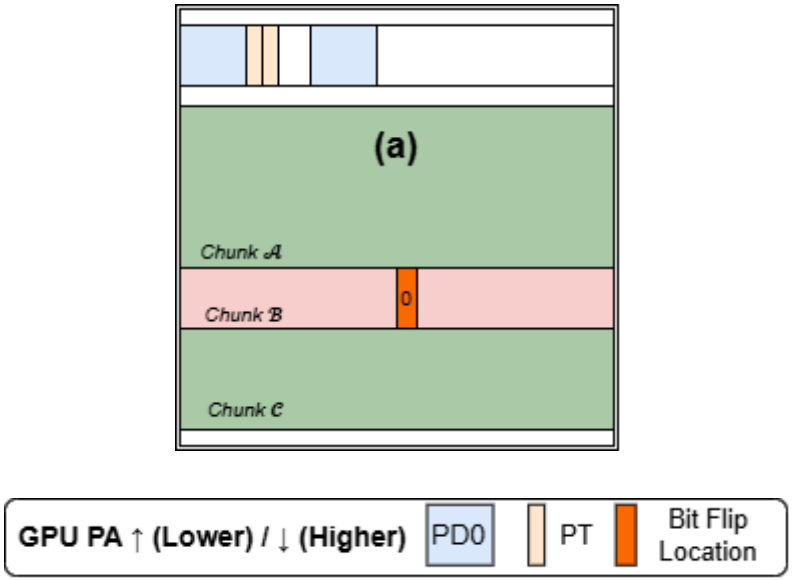
- **Goal: Steer GPU page tables to bit flip locations**

- **Steps**
  - **(a) Allocate huge memory buffers with *cudaMalloc()***
  - **(b) Deplete the default page table region**
  - **(c) Free vulnerable chunk**
  - **(d) Steer page table structures into the bit flip location**
  - **(e) Hammer to trigger the bit flip**

- ***Targets page directory (PD0) not page table (PT)***
  - **Gaining full control over a page table**

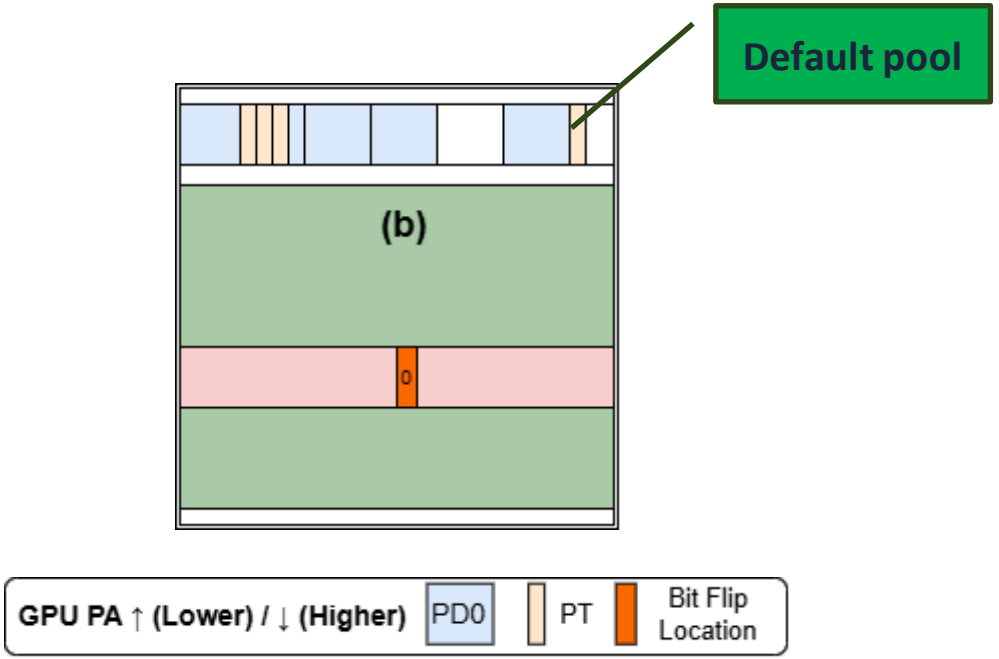
(III) Page Table Massaging

- (a) Allocate huge memory buffers with *cudaMalloc()*



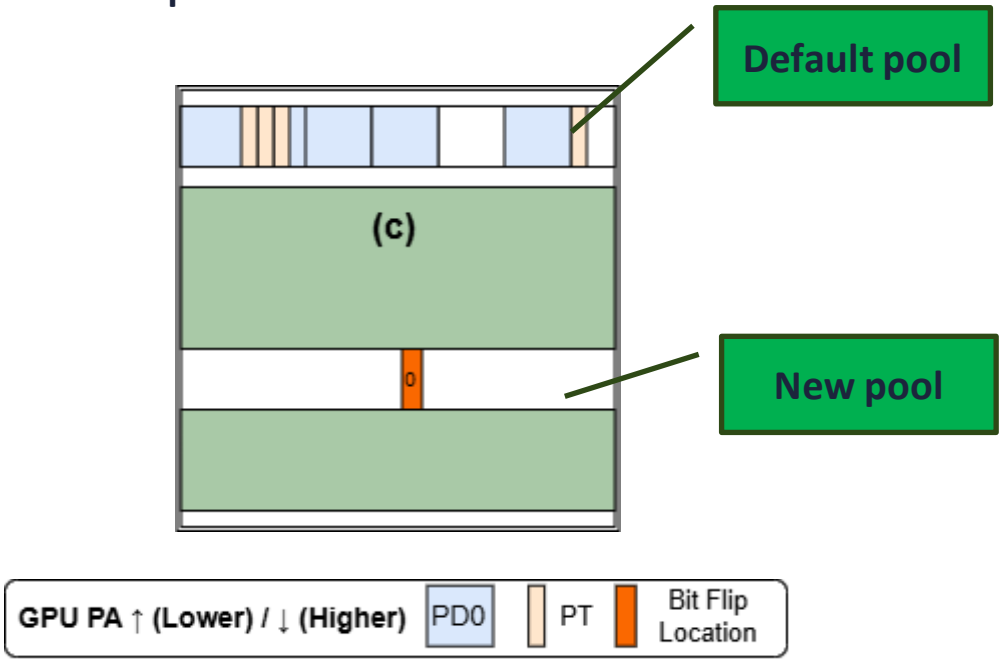
(III) Page Table Massaging

- (b) Deplete the default page table region (pool)



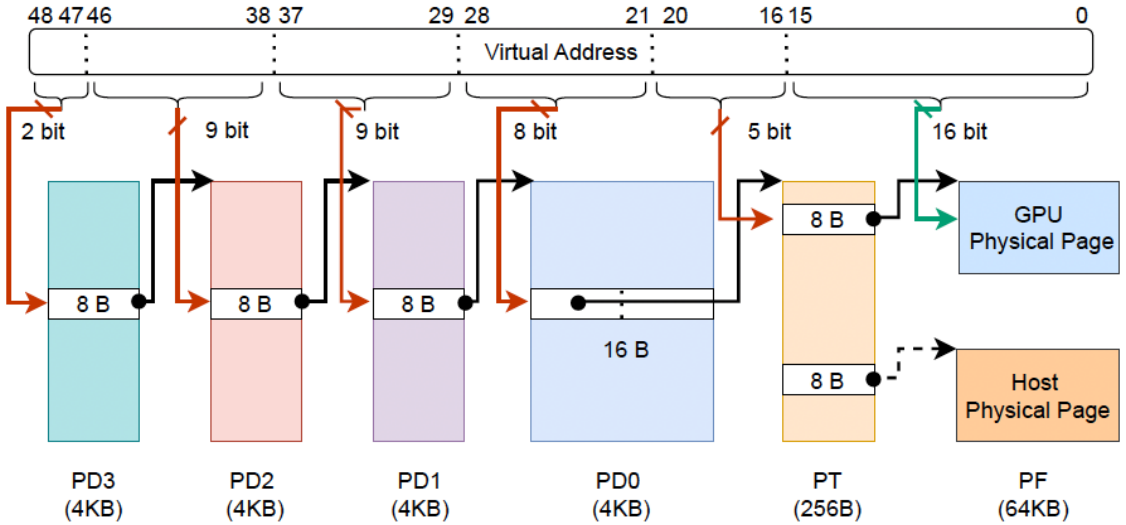
(III) Page Table Massaging

- (c) Free vulnerable chunk - *Chunk B*
  - Right after the default pool is depleted
  - *Chunk B* becomes the new pool



(III) Page Table Massaging

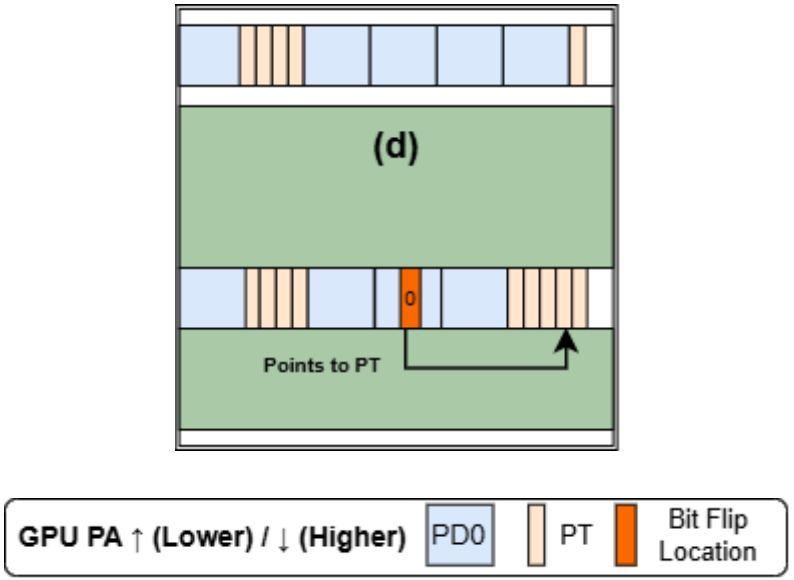
- (d) Steer page table structures into the bit flip location
  - Which part of the page table to exploit?



A PD0 Entry!

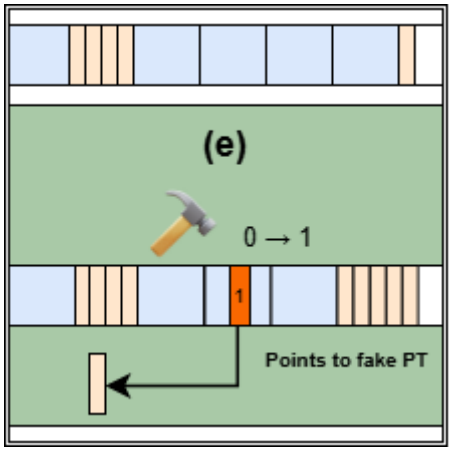
(III) Page Table Massaging

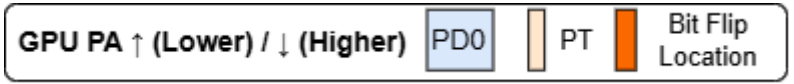
- (d) Steer page table structures into the bit flip location
  - Point the PD0 entry to a real Page Table (PT)



(III) Page Table Massaging

- **(e) Hammer to trigger the bit flip**
  - **Point the PD0 entry to a faked Page Table (PT)**





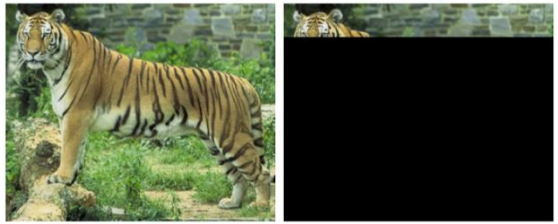
(IV) Exploitation

- **Primitives: arbitrary Read/Write**
  - **GPU memory**
  - **Host memory (Set System Aperture bit/Without IOMMU)**

(IV) Exploitation

- **Primitives: arbitrary Read/Write**

- **GPU Exploitation**
  - **GPU contexts**



(a) Original figure (b) Recovered by GeForge

(c) Sketched figure (d) Sketched after falsifying

(IV) Exploitation

- Primitives: arbitrary Read/Write
- GPU Exploitation
- Root Privilege Escalation

Summary

- **GeForge makes GPU Rowhammer more practical**
  - **Proposes a new hammering pattern**
  - **Finds more bit flips and attacks another GPU model – RTX 3060**
  - **Proposes a new massaging strategy to corrupt GPU page tables**
  - **Achieves arbitrary read/write permission with a single bit flip**

https://gddr.fail

https://stefan1wan.github.io

Junpeng Wan, wan155@purdue.edu