DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips

Fan Yao[1] Adnan Siraj Rakin[2] Deliang Fan[2]

[1]University of Central Florida
Orlando, FL

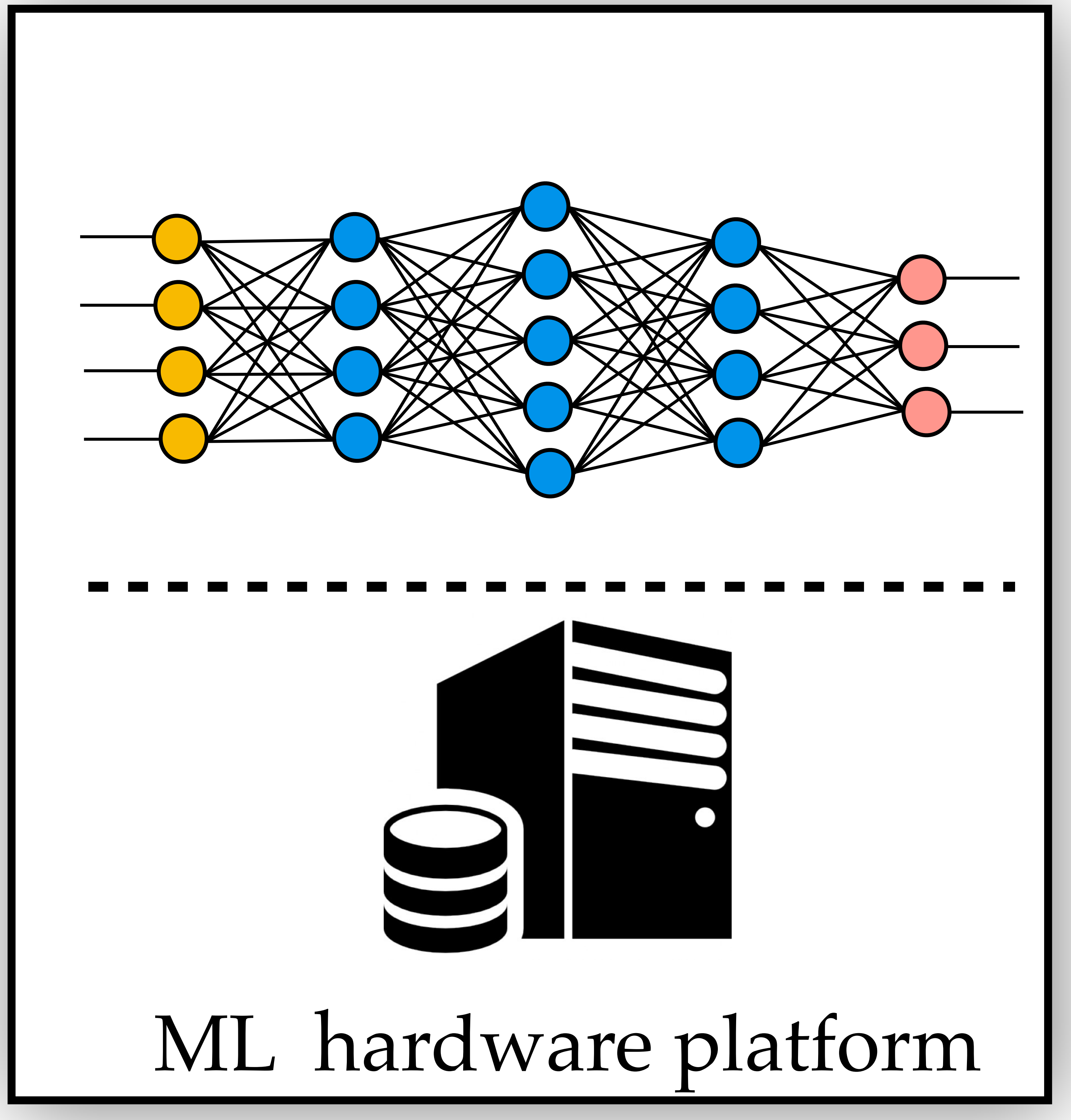
[2]Arizona State University
Tempe, Arizona

29th USENIX Security Symposium, August, 2020

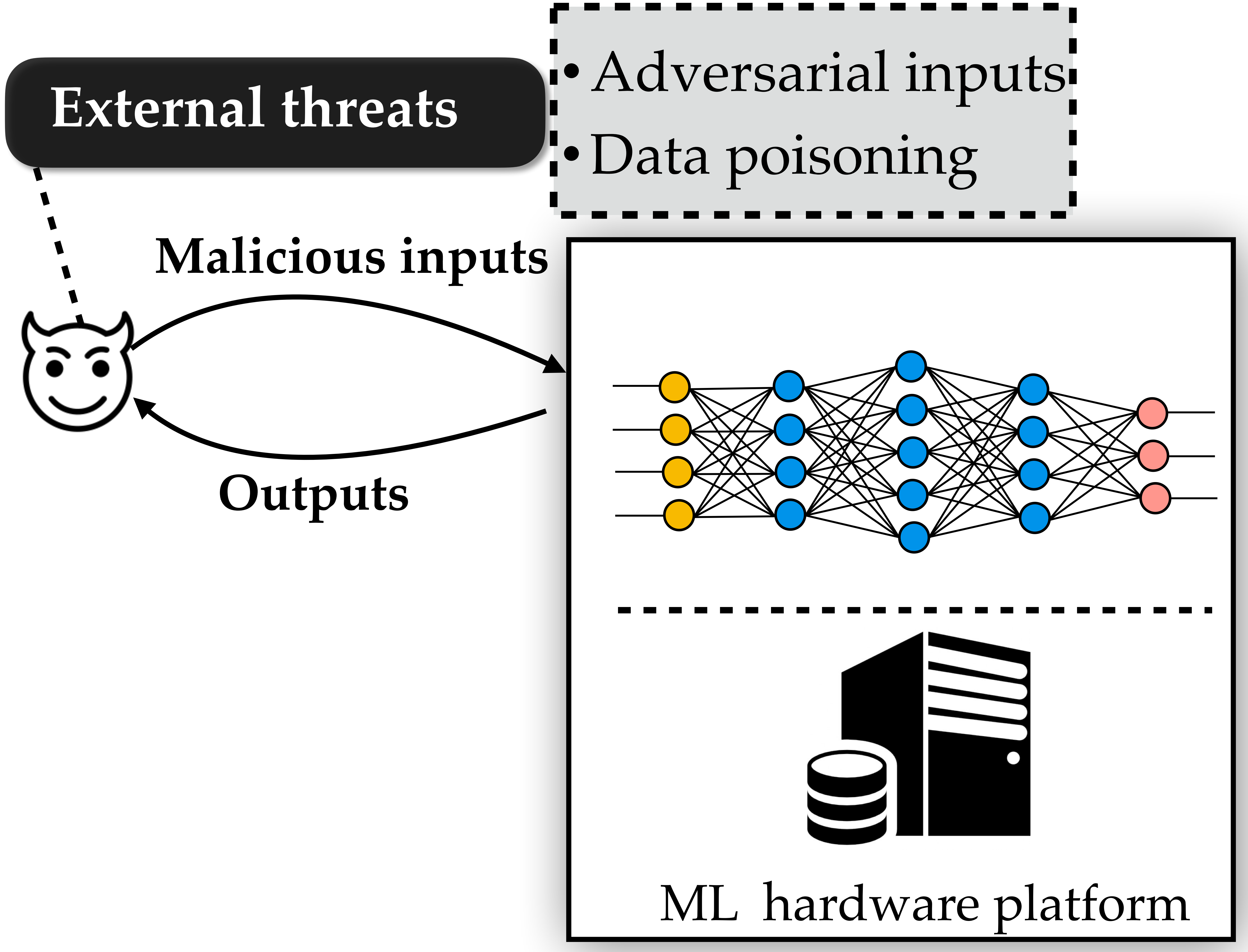
Security of Machine Learning

- Tremendous advances of machine learning (ML)
  - Wide deployment of machine learning platforms (e.g., **MLaaS**)
    - Amazon AWS AI, Google Cloud and Microsoft Azure ML
  - DNN applications increasingly integrated in **critical systems**
    - E.g., Medical diagnostics, access control and malware detection
- **DNN model integrity as a key concern**
  - Model tampering can introduce **severe consequences**
    - E.g., Making wrong decisions during autonomous driving

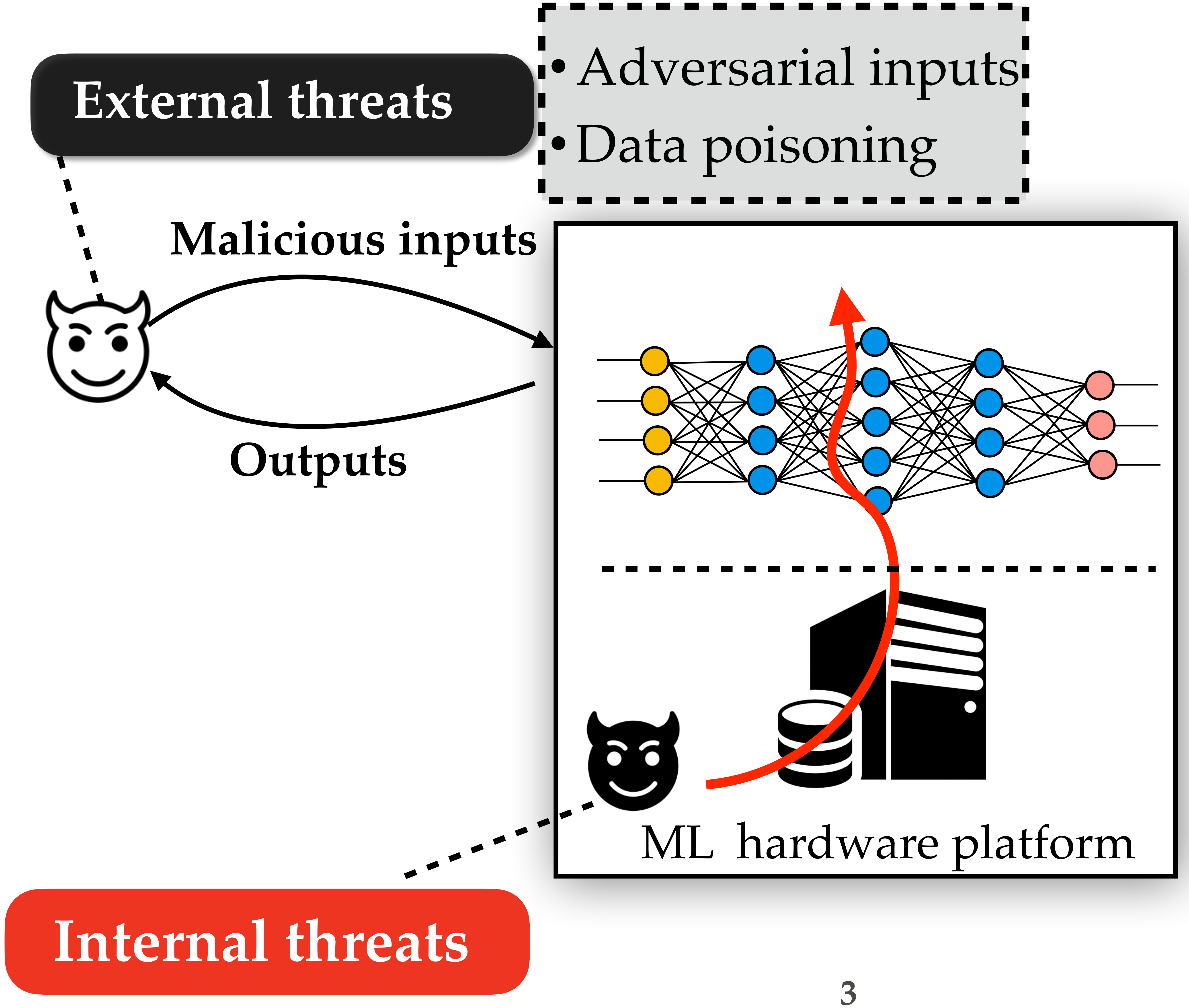
DNN Model Tampering Threats



DNN Model Tampering Threats



DNN Model Tampering Threats



❖ **HW is prone to faults**
- Computing logic
- Caches
- DRAM modules **(i.e., Rowhammer)**

Are Deep Neural Networks vulnerable to **Internal Adversaries** exploiting **Hardware-based Faults**?

Scope of Attack

- Focusing on **Quantized** DNNs
  - Quantized models are more robust to bit flip (**Hong et al. SEC'19**)
  - Quantization is a widely applied technique
- Leveraging **Rowhammer** to inject faults to **DNN model weights**
  - Allow deterministic bit flips in memory by unprivileged software
- We termed the attack: **DeepHammer**

Objective of DeepHammer

Objective of DeepHammer

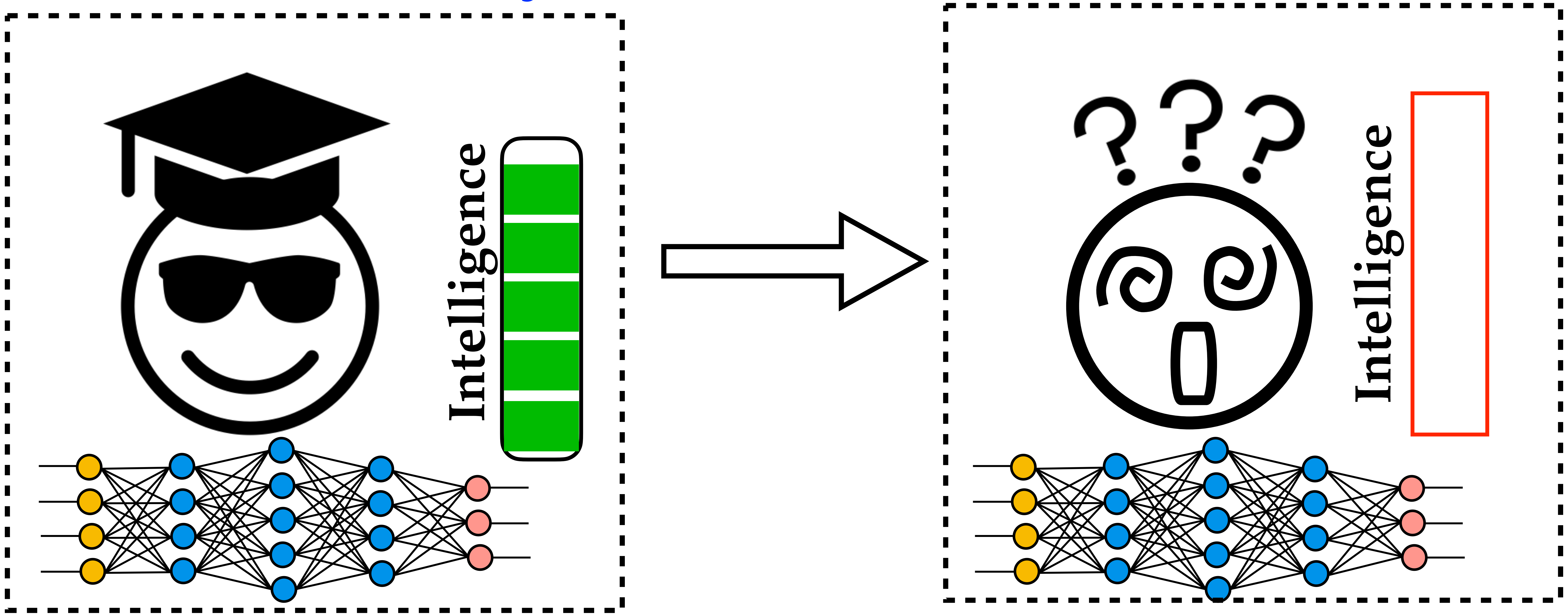
Degrade the **inference accuracy** to the level of **Random Guess**

Objective of DeepHammer

Degrade the inference accuracy to the level of Random Guess

Example: ResNet-20 for CIFAR-10, **10** output classes

Before attack, **Accuracy: 90.2%** After attack, **Accuracy: ~10% (1/10)**

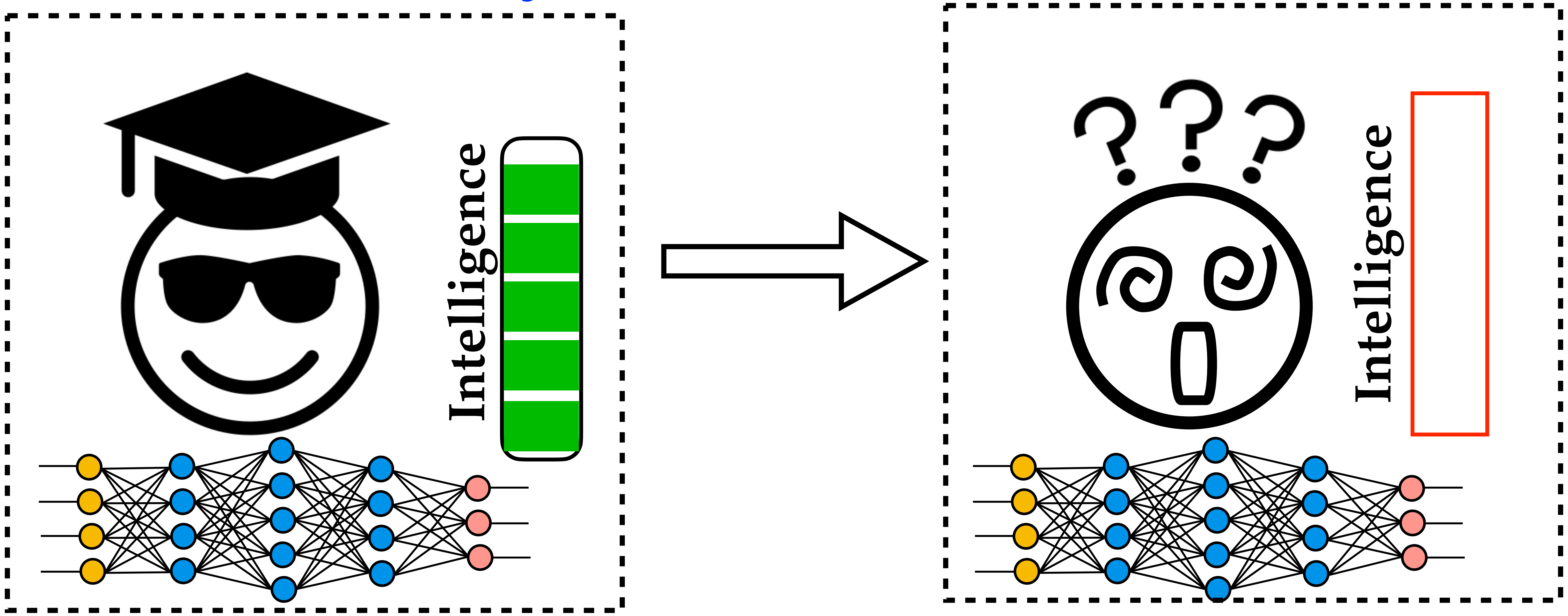


Objective of DeepHammer

Degrade the inference accuracy to the level of Random Guess

Example: ResNet-20 for CIFAR-10, **10** output classes

Before attack, **Accuracy: 90.2%** After attack, **Accuracy: ~10% (1/10)**



Depleting the intelligence of well-trained DNNs

Attack Challenges

- **Challenges** to carry out attacks on quantized DNNs

Attack Challenges

- **Challenges** to carry out attacks on quantized DNNs

C1: How to identify the most vulnerable bits? — Algorithm challenge

C2: How to successfully flip the selected bits? — System challenge

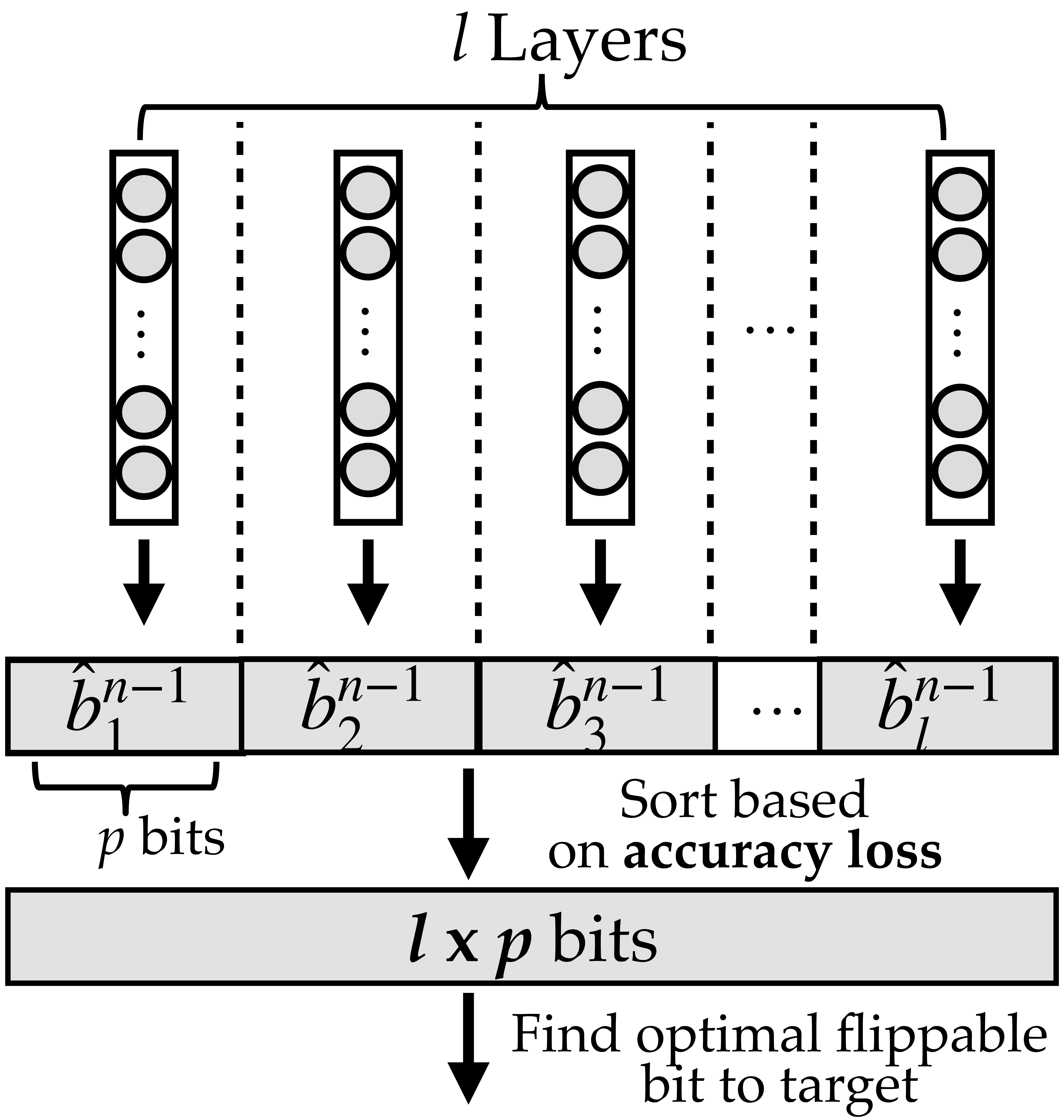
Locating the Most Vulnerable Weight Bits

- An iterative bit search process (one bit at a time)
- For each iteration:
  - Perform Gradient-based Bit Ranking (**GBR**)

$$\hat{\boldsymbol{b}}_m^{n-1} = \operatorname*{Top}_p \left| \nabla_{\hat{\mathbf{B}}_m^{n-1}} \mathcal{L}\Big(f(\boldsymbol{x}; \{\hat{\mathbf{B}}_m^{n-1}\}_{m=1}^{l}), \boldsymbol{t} \Big) \right| \quad (1)$$

$$\mathcal{L}_i^n = \mathcal{L}\Big(f(\boldsymbol{x}; \{\hat{\mathbf{B}}^n\}_{i=1}^{l\times p}, \boldsymbol{t} \Big) \quad (2)$$

  - Flip-aware Bit Search (**FBS**), Select a bit that:
    - **Incurs most accuracy lost**
    - **location flippable (checks bit flip profile)**
- If accuracy target not reached: next iteration



Locating the Most Vulnerable Weight Bits

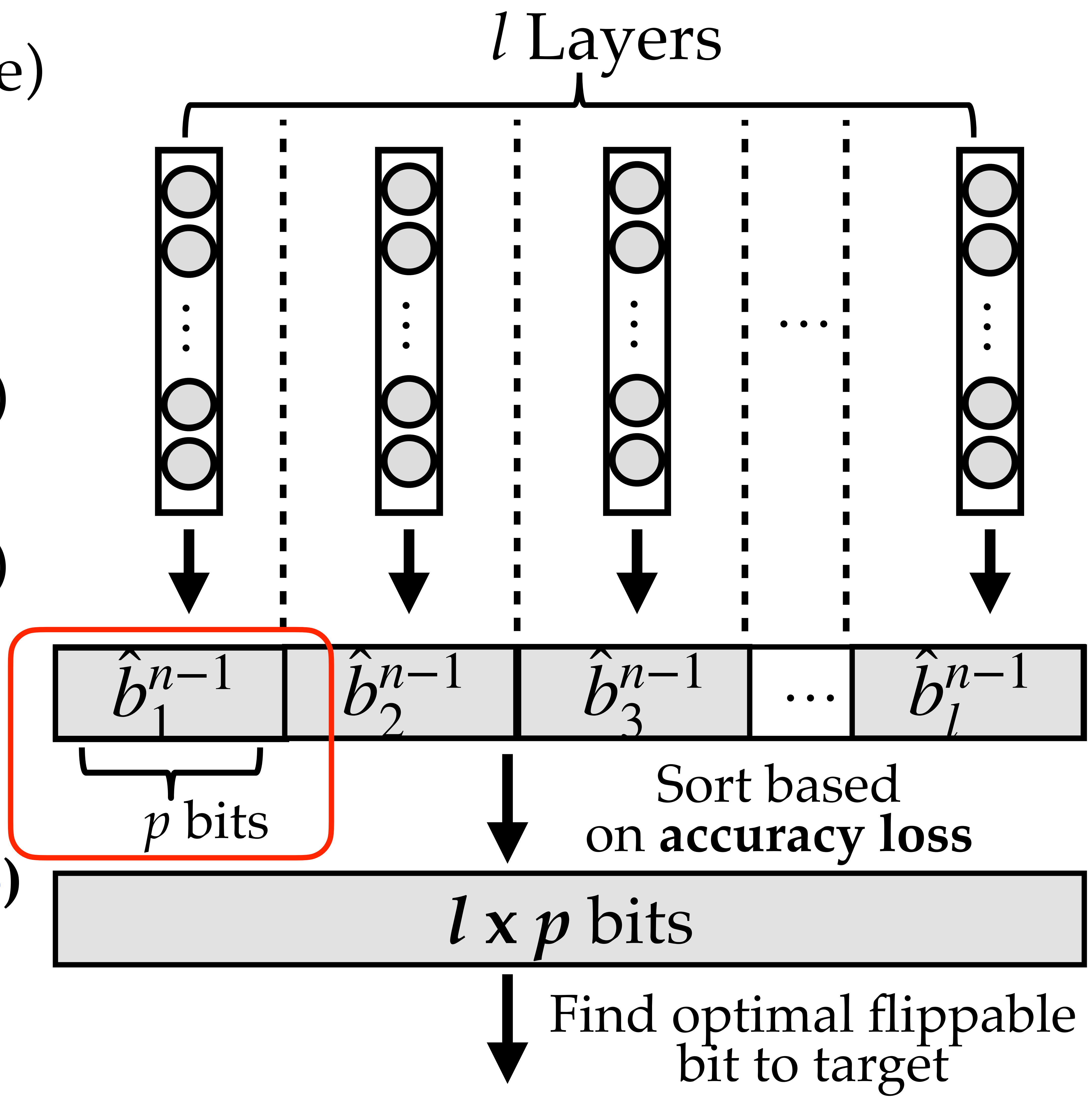
- An iterative bit search process (one bit at a time)
- For each iteration:
  - Perform Gradient-based Bit Ranking (**GBR**)

$$\hat{\boldsymbol{b}}_m^{n-1} = \mathop{\mathrm{Top}}_{p} \left| \nabla_{\hat{\mathbf{B}}_m^{n-1}} \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}_m^{n-1}\}_{m=1}^{l}\big), \boldsymbol{t}\Big) \right| \quad (1)$$

$$\mathcal{L}_i^n = \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}^n\}_{i=1}^{l\times p}, \boldsymbol{t}\Big) \quad (2)$$

  - Flip-aware Bit Search (**FBS**), Select a bit that:
    - **Incurs most accuracy lost**
    - **location flippable (checks bit flip profile)**

- If accuracy target not reached: next iteration



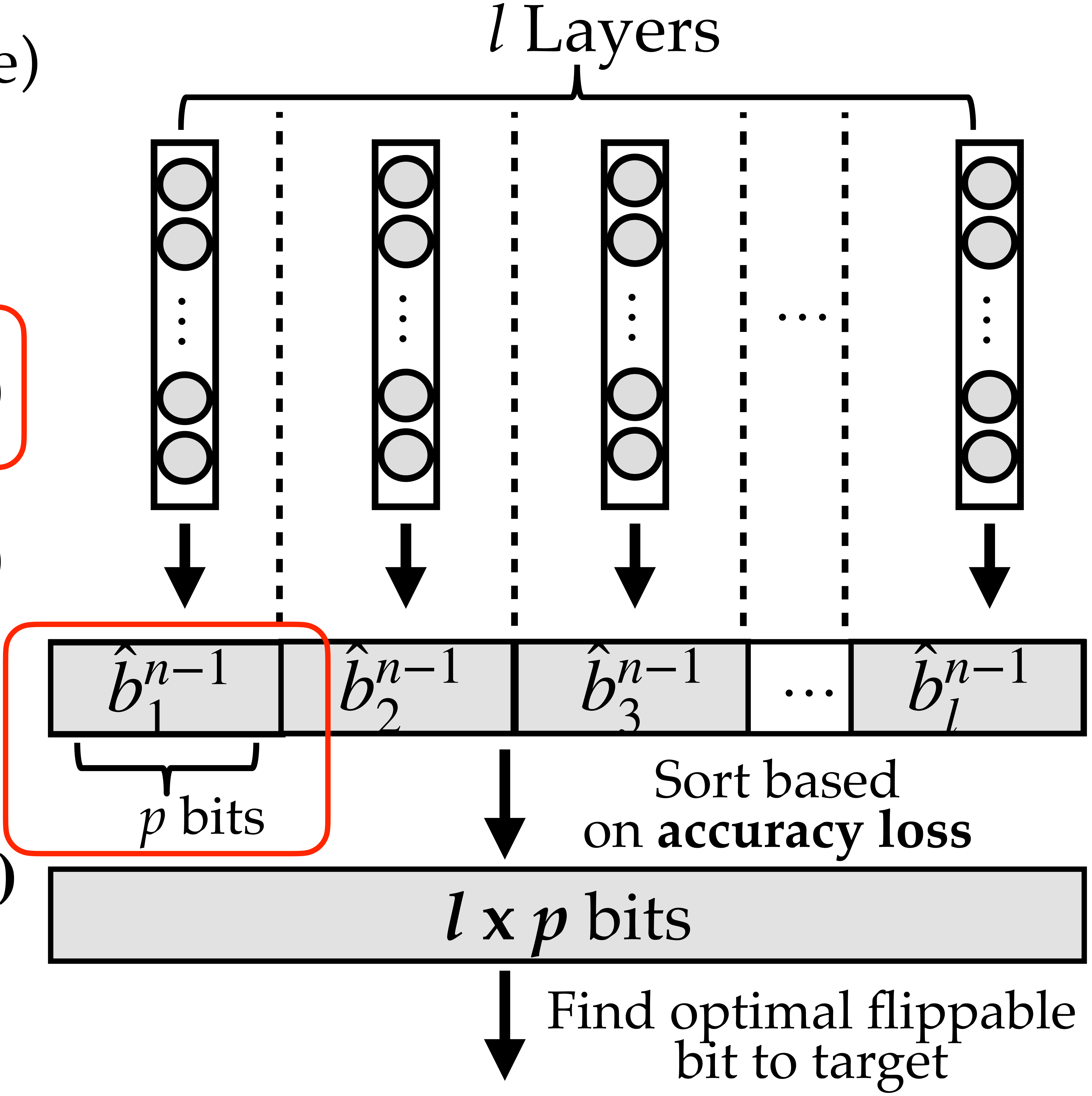
Locating the Most Vulnerable Weight Bits

- An iterative bit search process (one bit at a time)
- For each iteration:
  - Perform Gradient-based Bit Ranking (**GBR**)

$$\boxed{\hat{\boldsymbol{b}}_m^{n-1} = \underset{p}{\mathrm{Top}} \left| \nabla_{\hat{\mathbf{B}}_m^{n-1}} \mathcal{L}\left(f(\boldsymbol{x}; \{\hat{\mathbf{B}}_m^{n-1}\}_{m=1}^{l}), \boldsymbol{t} \right) \right| \quad (1)}$$

$$\mathcal{L}_i^n = \mathcal{L}\left(f(\boldsymbol{x}; \{\hat{\mathbf{B}}^n\}_{i=1}^{l \times p}, \boldsymbol{t} \right) \quad (2)$$

  - Flip-aware Bit Search (**FBS**), Select a bit that:
    - **Incurs most accuracy lost**
    - **location flippable (checks bit flip profile)**
- If accuracy target not reached: next iteration



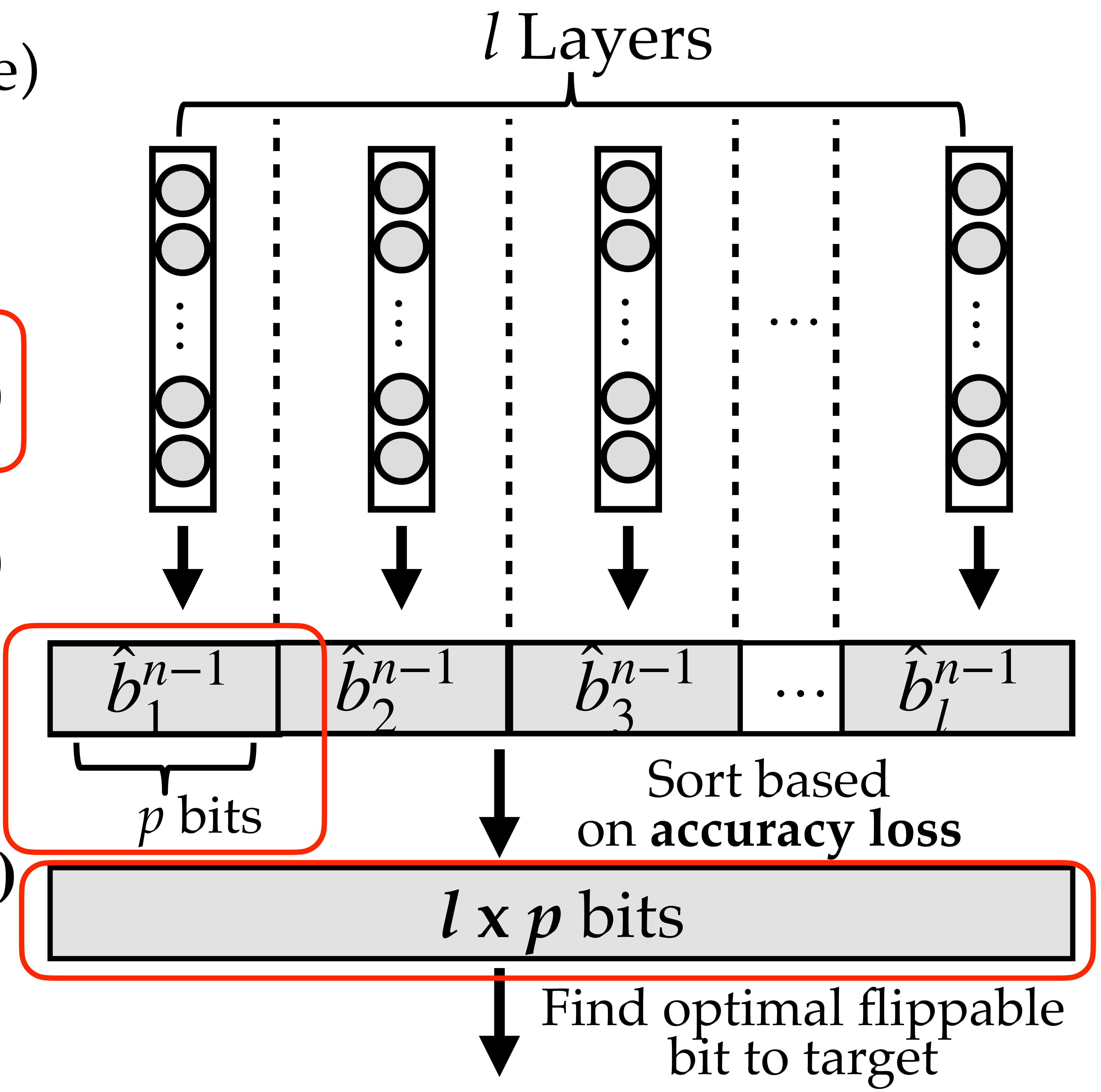
Locating the Most Vulnerable Weight Bits

- An iterative bit search process (one bit at a time)
- For each iteration:
  - Perform Gradient-based Bit Ranking (**GBR**)

$$\boxed{\hat{\boldsymbol{b}}_m^{n-1} = \operatorname*{Top}_p \left| \nabla_{\hat{\mathbf{B}}_m^{n-1}} \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}_m^{n-1}\}_{m=1}^{l}\big), \boldsymbol{t}\Big) \right| \quad (1)}$$

$$\mathcal{L}_i^n = \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}^n\}_{i=1}^{l\times p}, \boldsymbol{t}\big)\Big) \quad (2)$$

  - Flip-aware Bit Search (**FBS**), Select a bit that:
    - **Incurs most accuracy lost**
    - **location flippable (checks bit flip profile)**



- If accuracy target not reached: next iteration

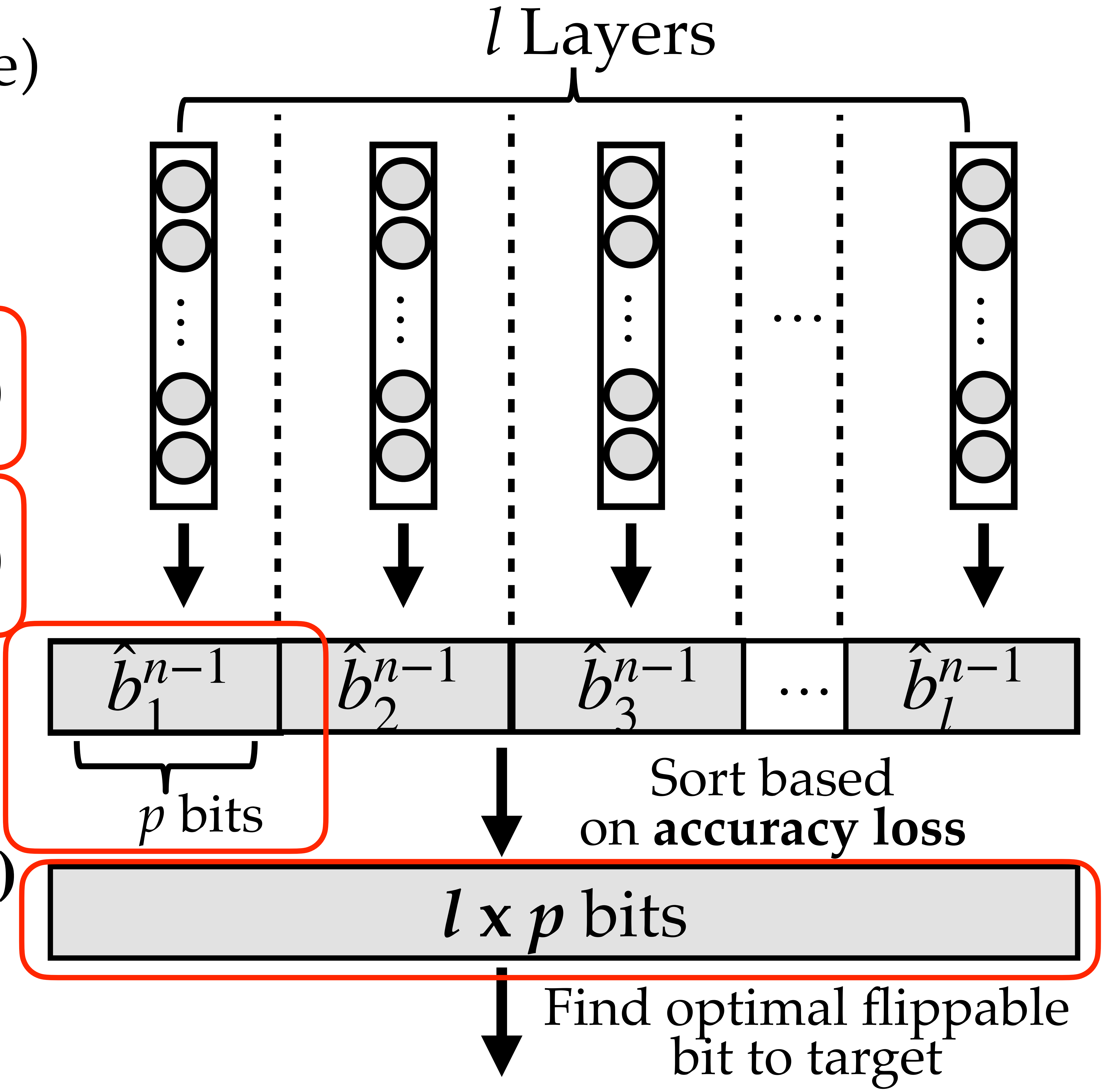
Locating the Most Vulnerable Weight Bits

- An iterative bit search process (one bit at a time)
- For each iteration:
  - Perform Gradient-based Bit Ranking (**GBR**)

$$\hat{\boldsymbol{b}}_m^{n-1} = \operatorname*{Top}_p \left| \nabla_{\hat{\mathbf{B}}_m^{n-1}} \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}_m^{n-1}\}_{m=1}^{l}\big), \boldsymbol{t}\Big) \right| \quad (1)$$

$$\mathcal{L}_i^n = \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}^n\}_{i=1}^{l\times p}, \boldsymbol{t}\Big) \quad (2)$$

  - Flip-aware Bit Search (**FBS**), Select a bit that:
    - **Incurs most accuracy lost**
    - **location flippable (checks bit flip profile)**



- If accuracy target not reached: next iteration

Locating the Most Vulnerable Weight Bits

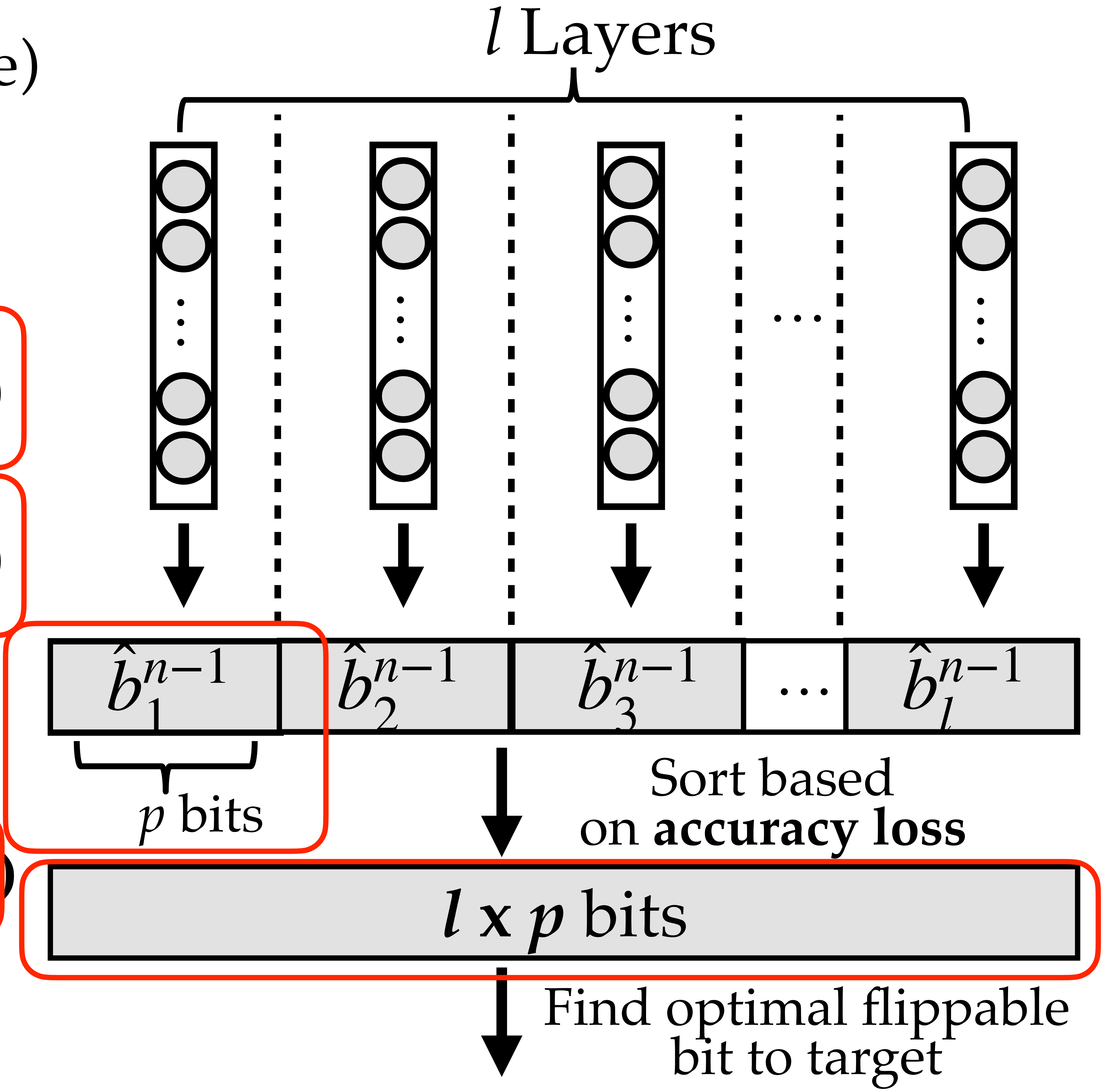
- An iterative bit search process (one bit at a time)
- For each iteration:
  - Perform Gradient-based Bit Ranking (**GBR**)

$$\hat{\boldsymbol{b}}_m^{n-1} = \operatorname{Top}_p \left| \nabla_{\hat{\mathbf{B}}_m^{n-1}} \mathcal{L}\left(f(\boldsymbol{x};\{\hat{\mathbf{B}}_m^{n-1}\}_{m=1}^{l}),\boldsymbol{t}\right)\right| \quad (1)$$

$$\mathcal{L}_i^n = \mathcal{L}\left(f(\boldsymbol{x};\{\hat{\mathbf{B}}^n\}_{i=1}^{l\times p},\boldsymbol{t}\right) \quad (2)$$

  - Flip-aware Bit Search (**FBS**), Select a bit that:
    - **Incurs most accuracy lost**
    - **location flippable (checks bit flip profile)**

- If accuracy target not reached: next iteration



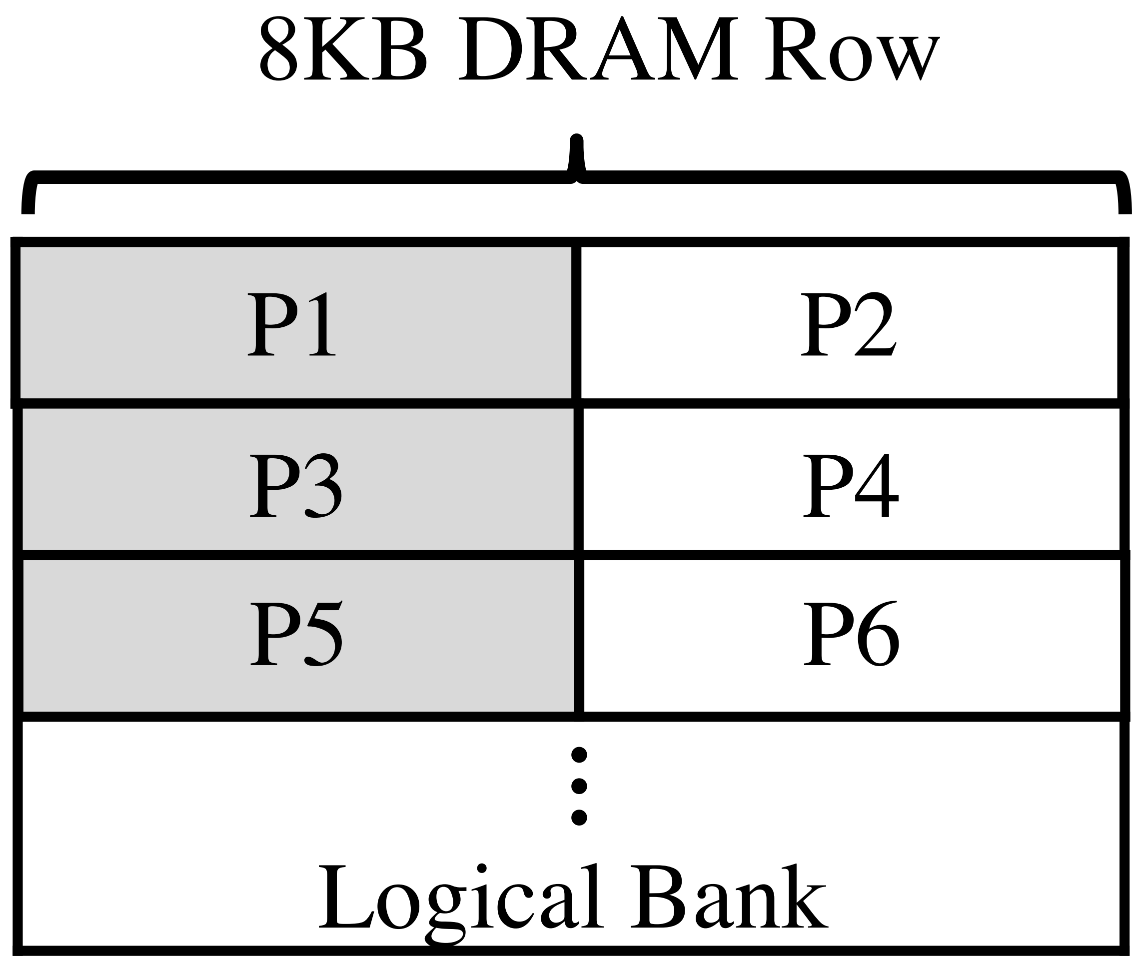
Rowhammer Framework in DeepHammer

Rowhammer Framework in DeepHammer

- Three advanced Rowhammer techniques
  - **Multi-page memory massaging**
    - **Enables fast and efficient victim page relocation**
  - **Precise rowhammering**
    - **Ensures exact bit flips based on the targeted bit chain**
  - **Online memory re-templating**
    - **Allows fast correction of obsolete DRAM bit flip profile**

Multi-page Memory Massaging

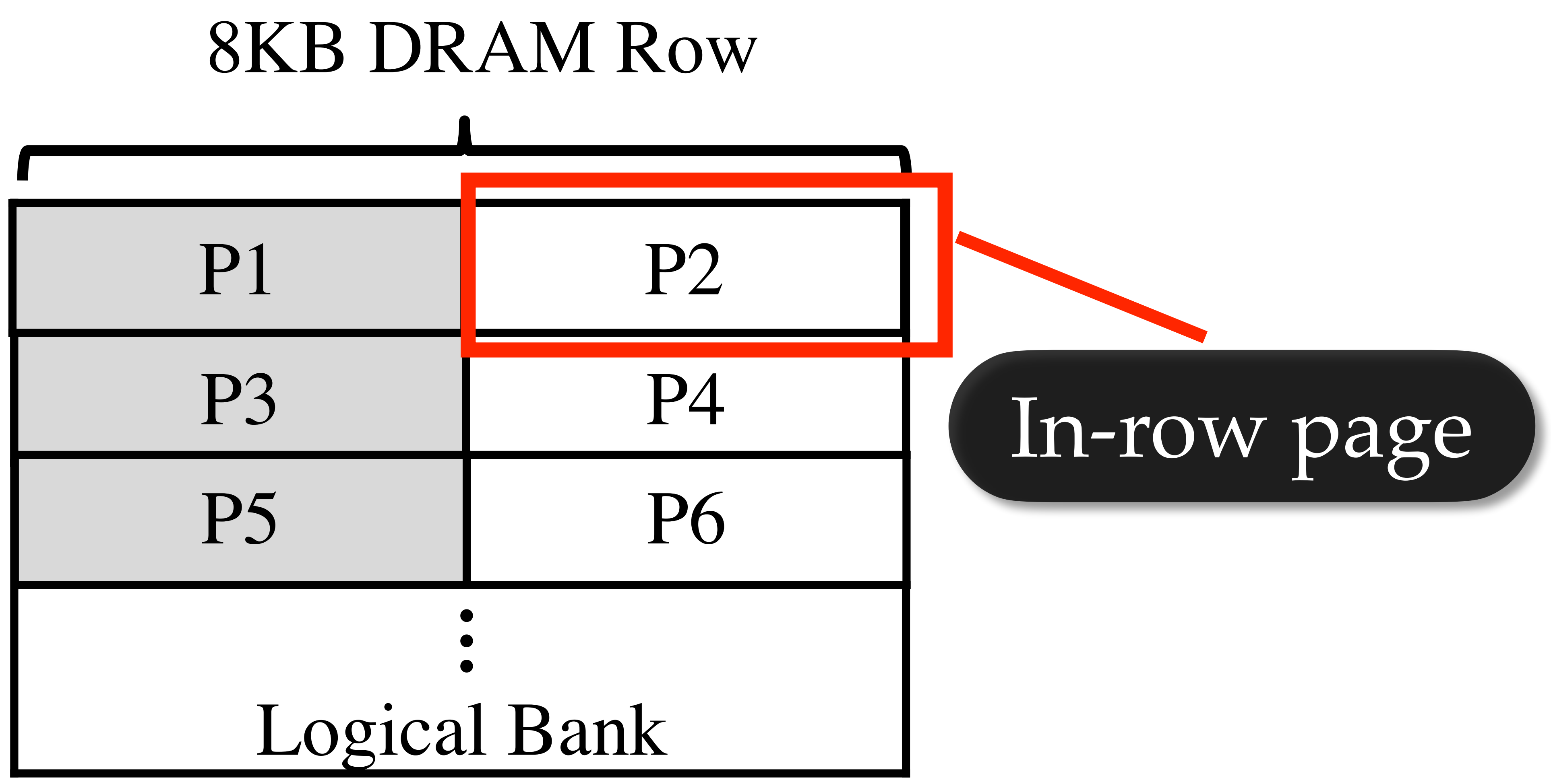
- Goal: **map multiple victim weight pages to exploitable DRAM rows**
  - In-row pages and compact aggressor rows
  - Target page positioning using **per-cpu pageset**
    - Last In First Out (LIFO)



Single channel single DIMM

Multi-page Memory Massaging

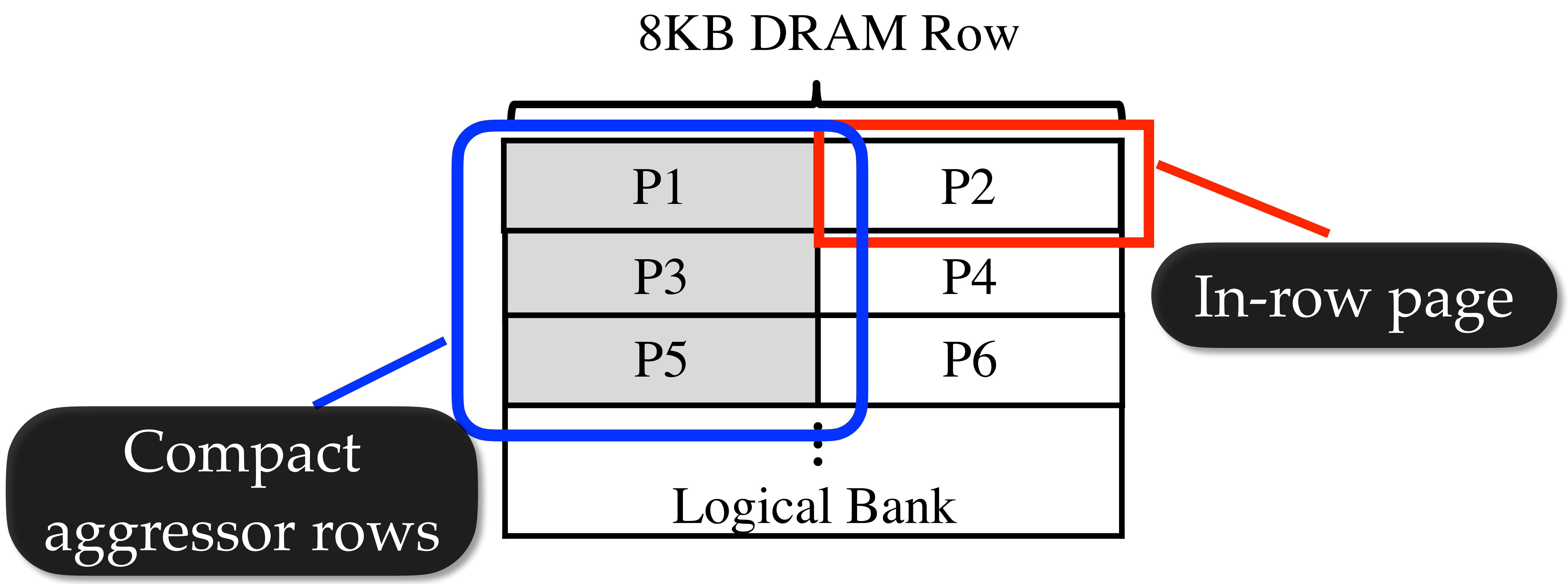
- Goal: **map multiple victim weight pages to exploitable DRAM rows**
  - In-row pages and compact aggressor rows
  - Target page positioning using **per-cpu pageset**
    - Last In First Out (LIFO)



Single channel single DIMM

Multi-page Memory Massaging

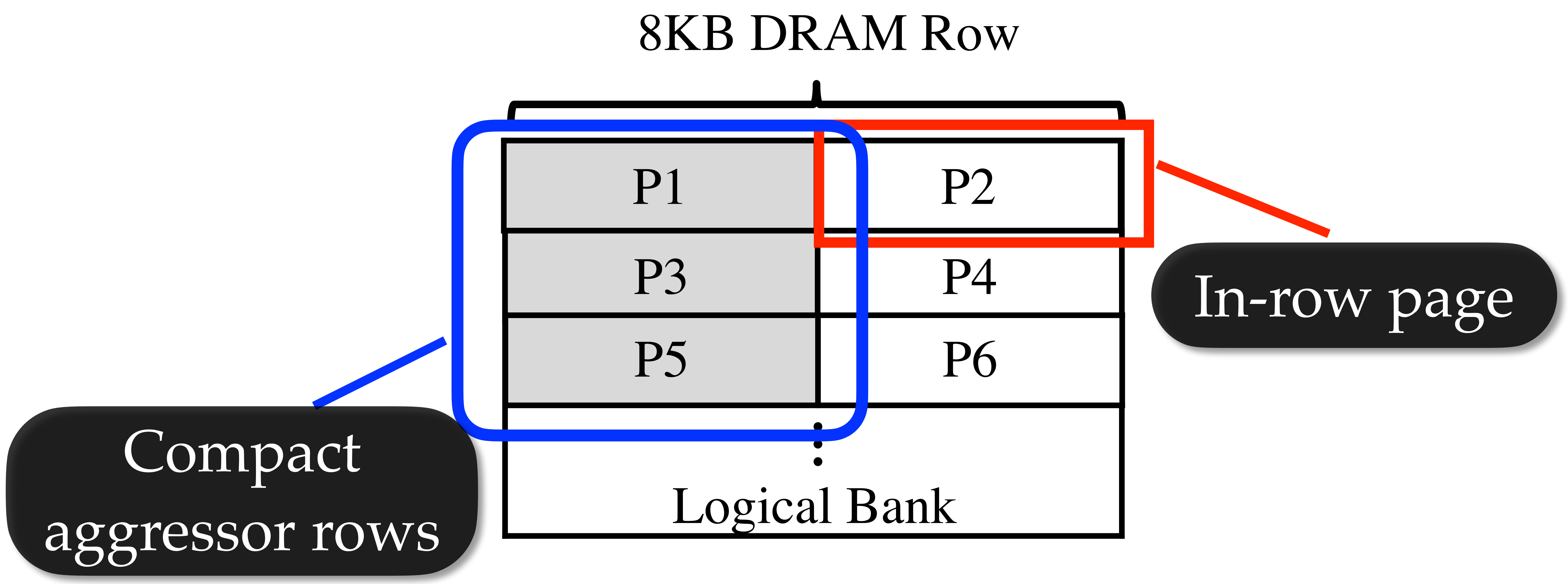
- Goal: **map multiple victim weight pages to exploitable DRAM rows**
  - In-row pages and compact aggressor rows
  - Target page positioning using **per-cpu pageset**
    - Last In First Out (LIFO)



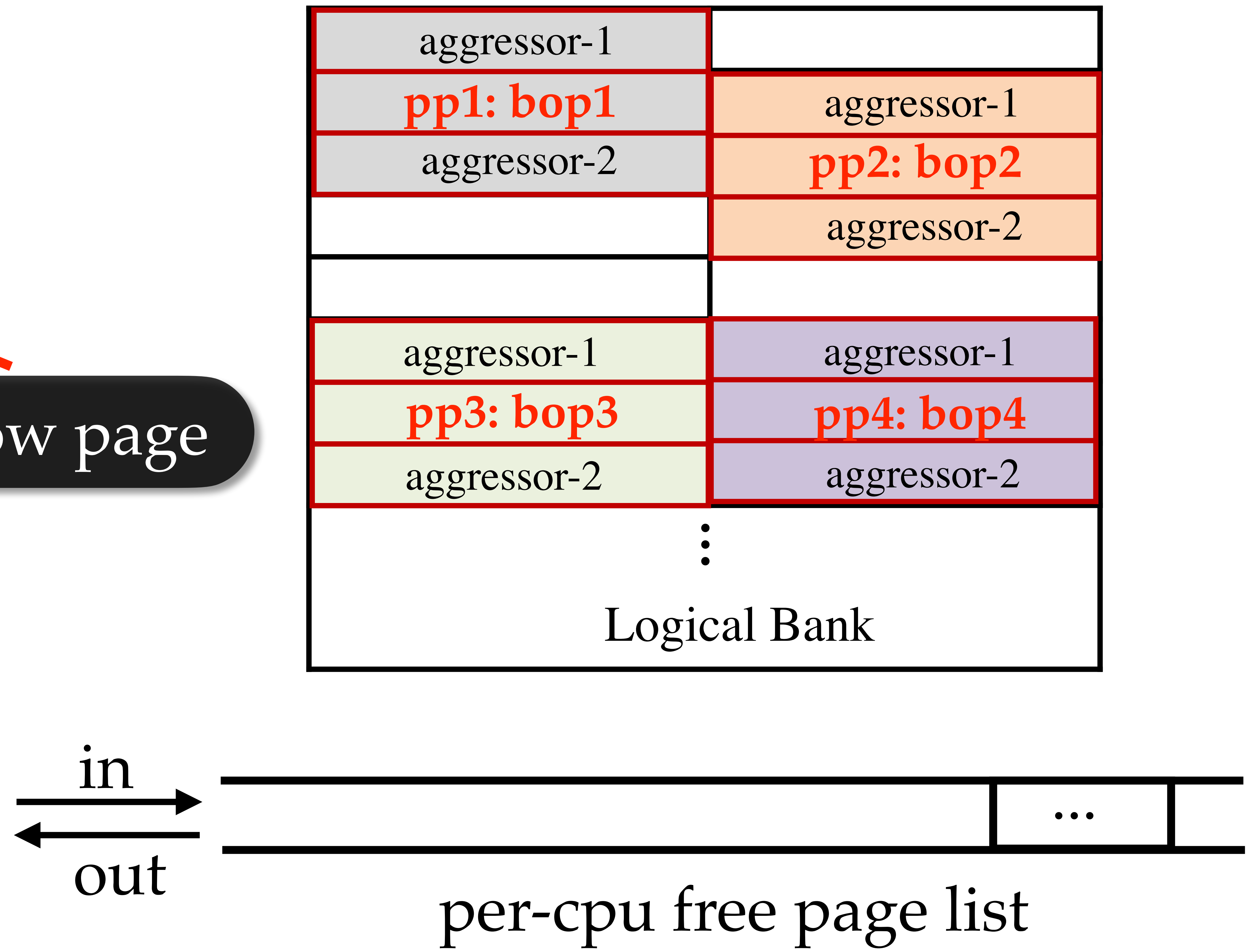
Single channel single DIMM

Multi-page Memory Massaging

- Goal: **map multiple victim weight pages to exploitable DRAM rows**
  - In-row pages and compact aggressor rows
  - Target page positioning using **per-cpu pageset**
    - Last In First Out (LIFO)

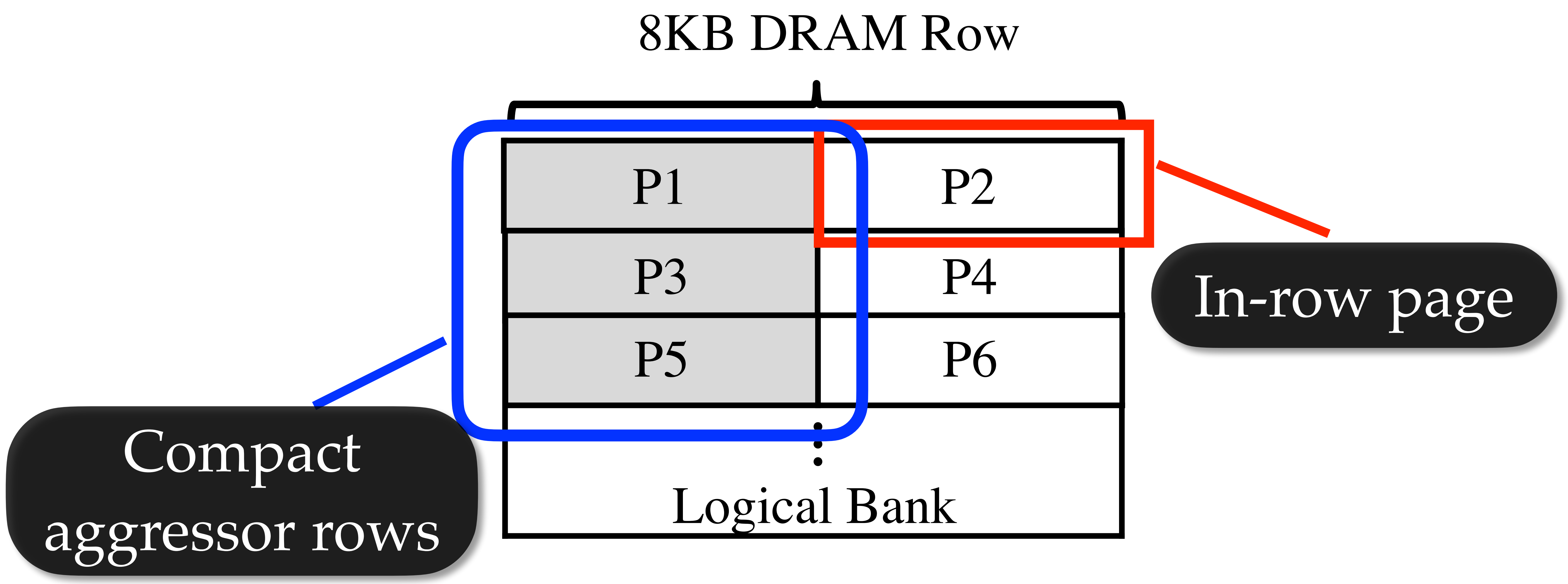


Single channel single DIMM

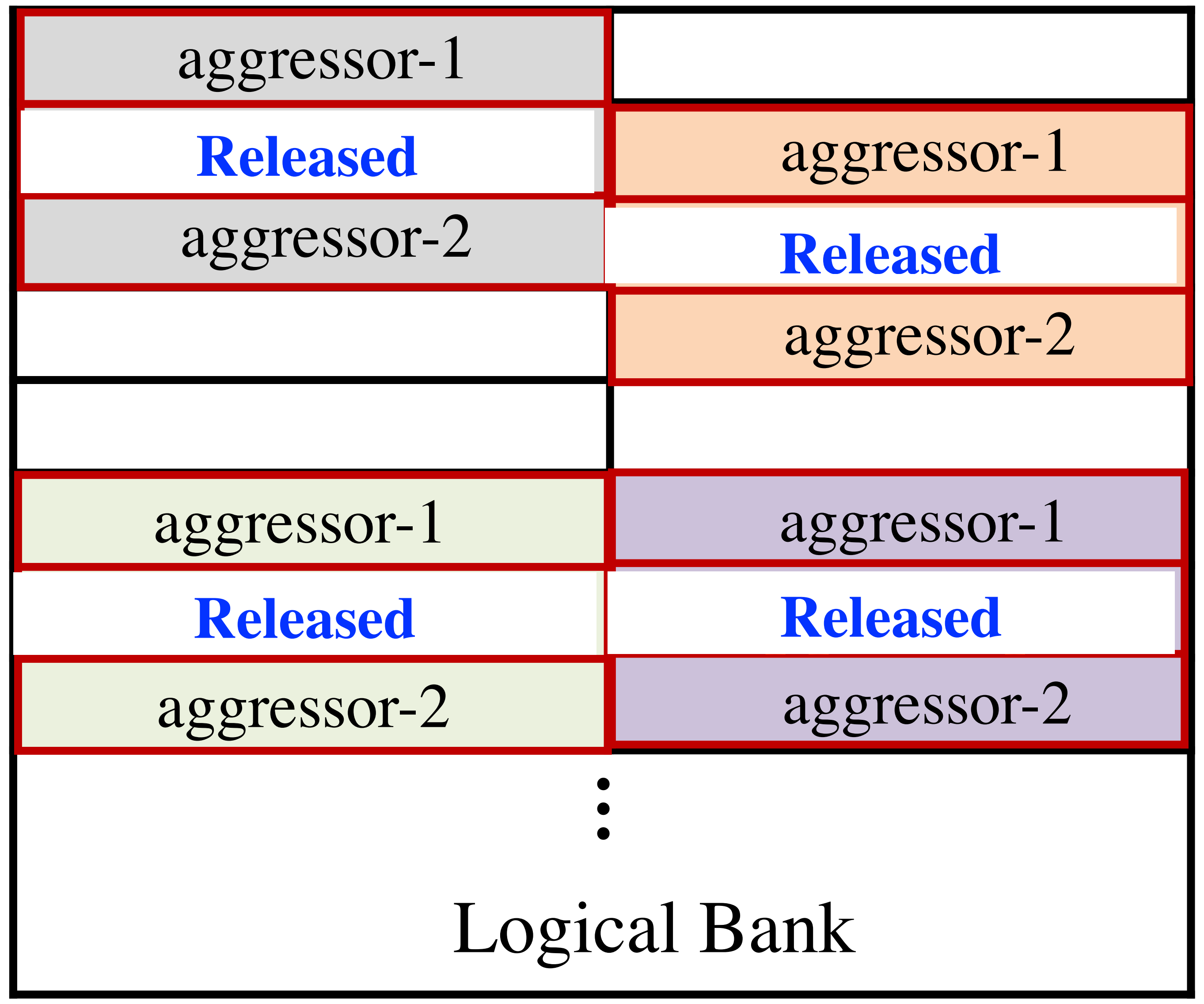


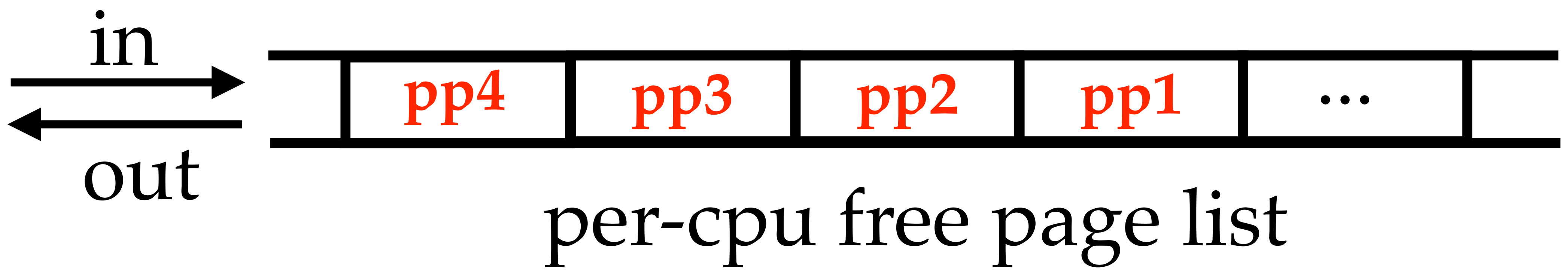
Multi-page Memory Massaging

- Goal: **map multiple victim weight pages to exploitable DRAM rows**
  - In-row pages and compact aggressor rows
  - Target page positioning using **per-cpu pageset**
    - Last In First Out (LIFO)



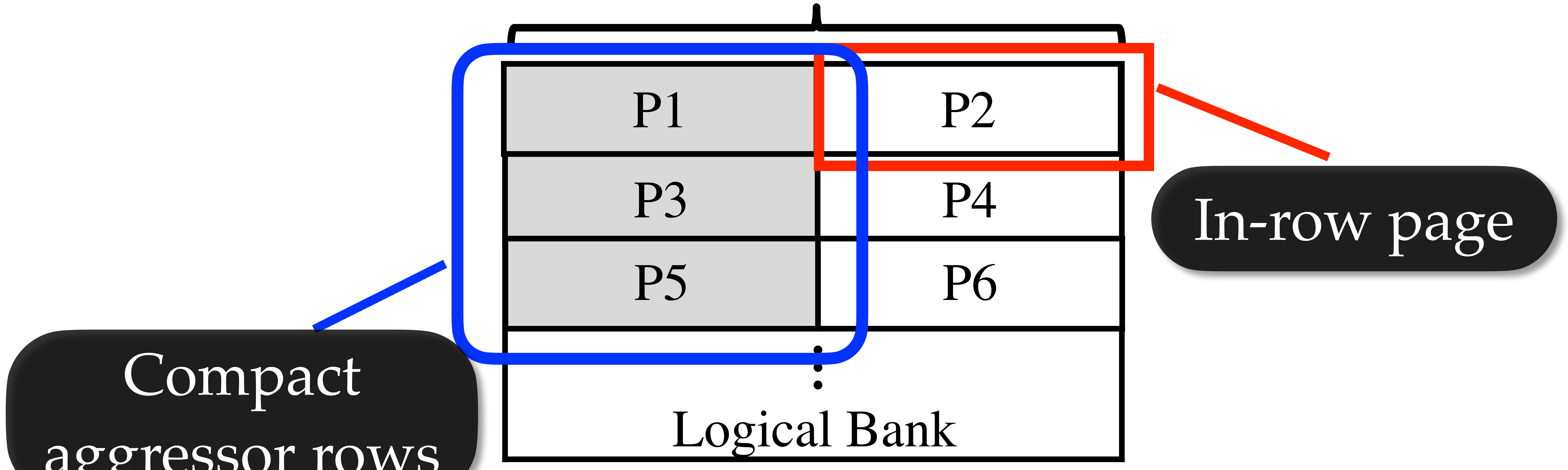
Single channel single DIMM



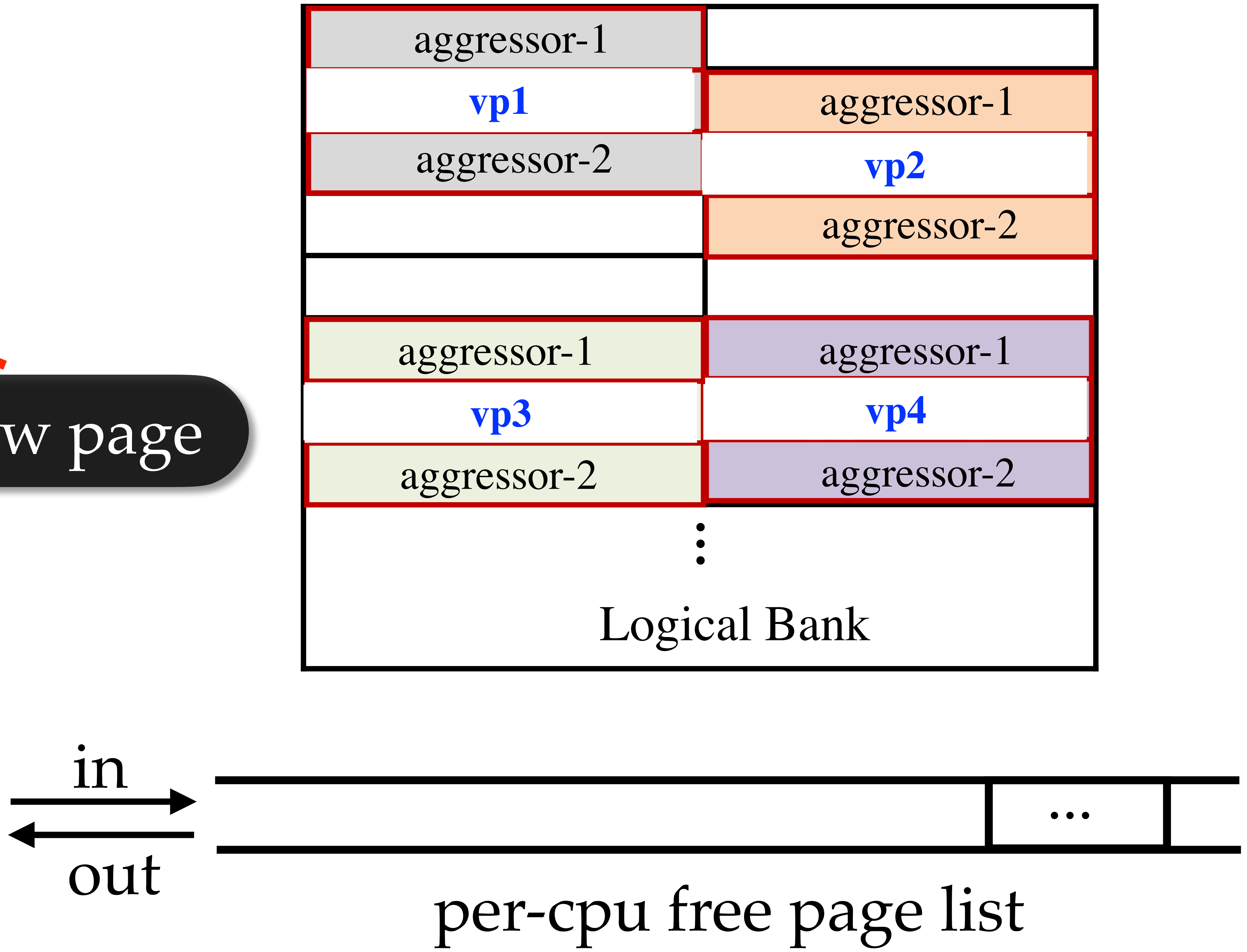


Multi-page Memory Massaging

- Goal: **map multiple victim weight pages to exploitable DRAM rows**
  - In-row pages and compact aggressor rows
  - Target page positioning using **per-cpu pageset**
    - Last In First Out (LIFO)

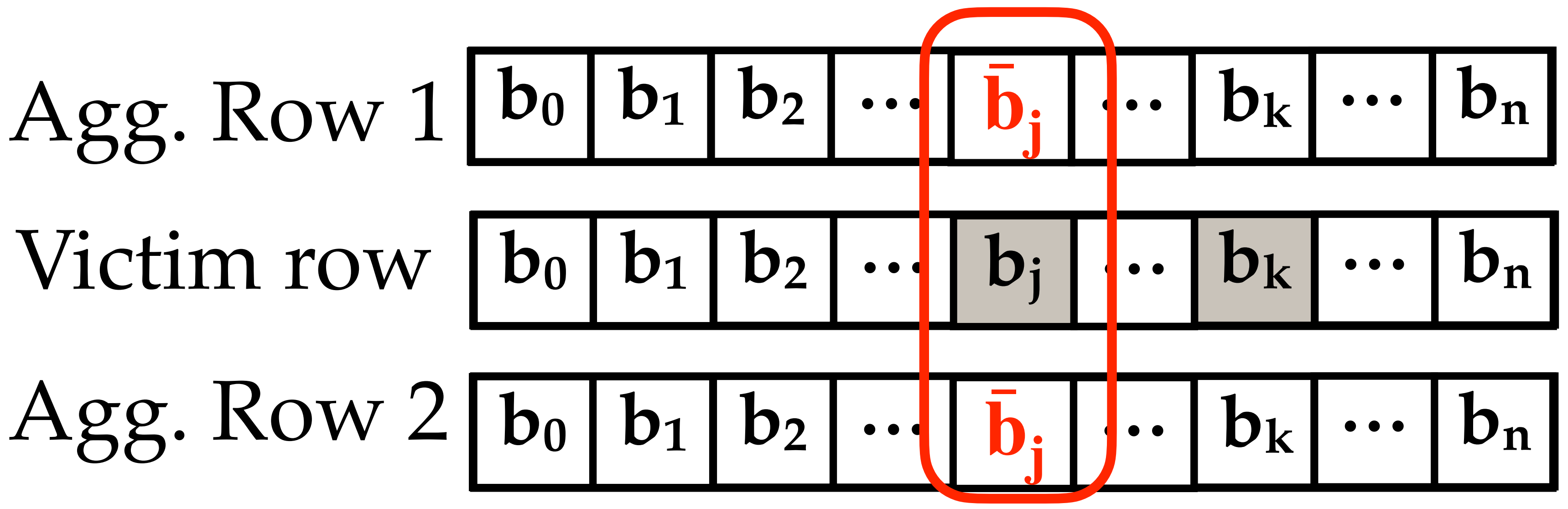


Single channel single DIMM



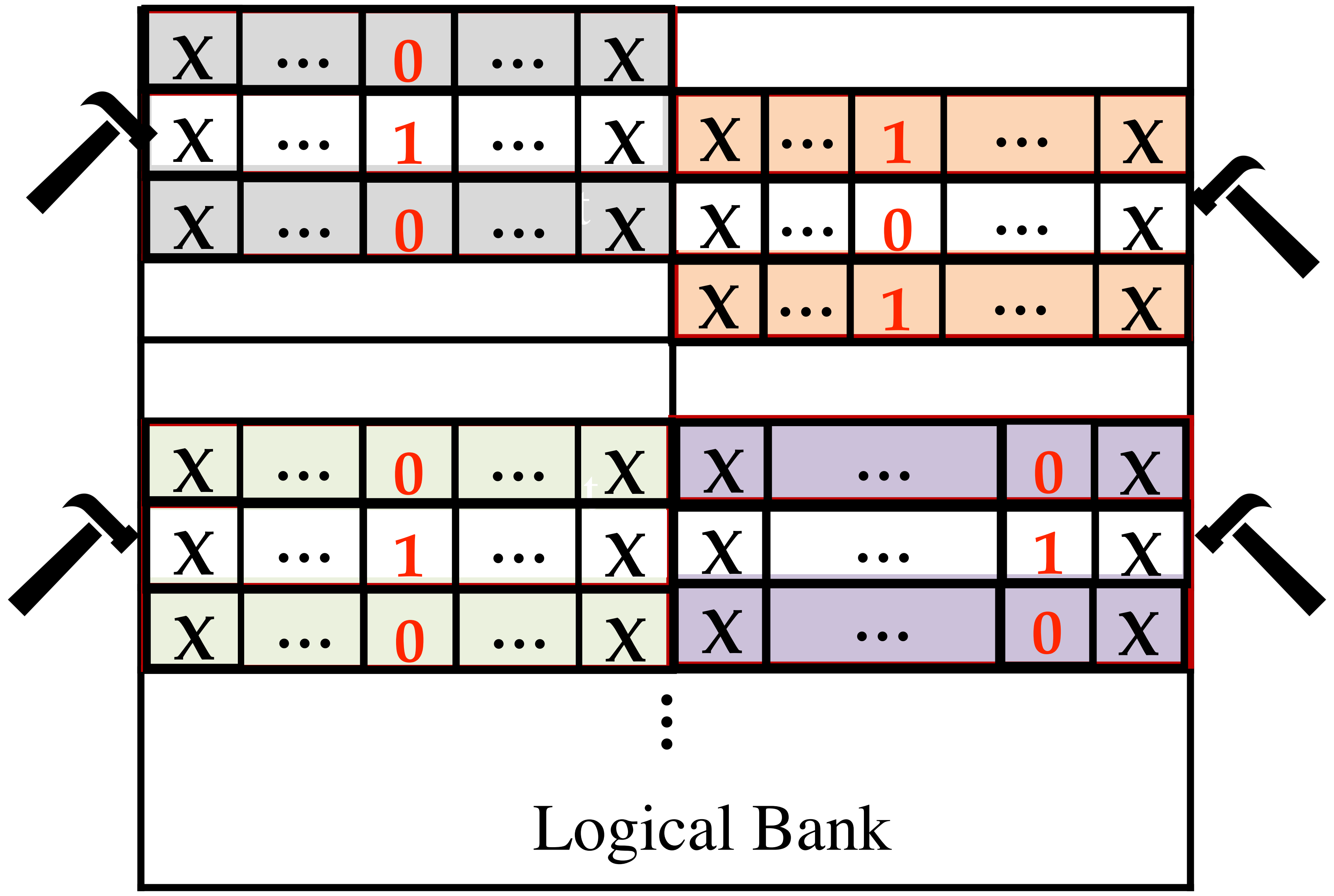
Precise Hammering

- Motivation: need to flip the exact bits
  - Undesired bit flips can fail the attack
- Unexpected bit flips could happen
  - E.g., multiple vulnerable cells in one row



Proposed column-page-stripe

Only b_j will flip



Fast Memory Re-templating

- **New issue: Bit flip profile can be obsolete**
  - After power cycle or reboot
- Observations
  - The **location of vulnerable cells** have not changed (page offset)
  - Potential reason: **data scrambling by memory controllers**
- How to update the bit flip profile at runtime?
  - Only re-template physical pages with desired exploitable offsets
  - Drastically reduce templating time: **days to minutes!**

Experimental Setup

- DNN configurations
  - Image processing dataset: **Fashion MNIST**, **CIFAR-10** and **ImageNet**
  - Speech recognition dataset: **Google Speech Command**
  - DNN models: **11** mainstream architectures, including **2** mobile networks
- Training platform (GPU)
  - GeForce GTX 1080 Ti GPU, 11 GB dedicated memory
- Inference platform (CPU)
  - Intel Ivy-Bridge processors
  - 4GB DDR3 DIMMs with single/dual channel setup

Evaluation: Bit Search Results

Dataset	Architecture	Network Parameters	Acc. Before Attack (%)	Random Guess Acc. (%)	Acc. After Attack (%)	Min. # of Bit-flips
Fashion MNIST	LeNet	0.65M	90.20	10.00	10.00	**3**
Google Speech Command	VGG-11	132M	96.36	8.33	3.43	**5**
	VGG-13	133M	96.38		3.25	**7**
CIFAR-10	ResNet-20	0.27M	90.70	10.00	10.92	**21**
	AlexNet	61M	84.40		10.46	**5**
	VGG-11	132M	89.40		10.27	**3**
	VGG-16	138M	93.24		10.82	**13**
ImageNet	SqueezeNet	1.2M	57.00	0.10	0.16	**18**
	MobileNet-V2	2.1M	72.01		0.19	**2**
	ResNet-18	11M	69.52		0.19	**24**
	ResNet-34	21M	72.78		0.18	**23**
	ResNet-50	23M	75.56		0.17	**23**

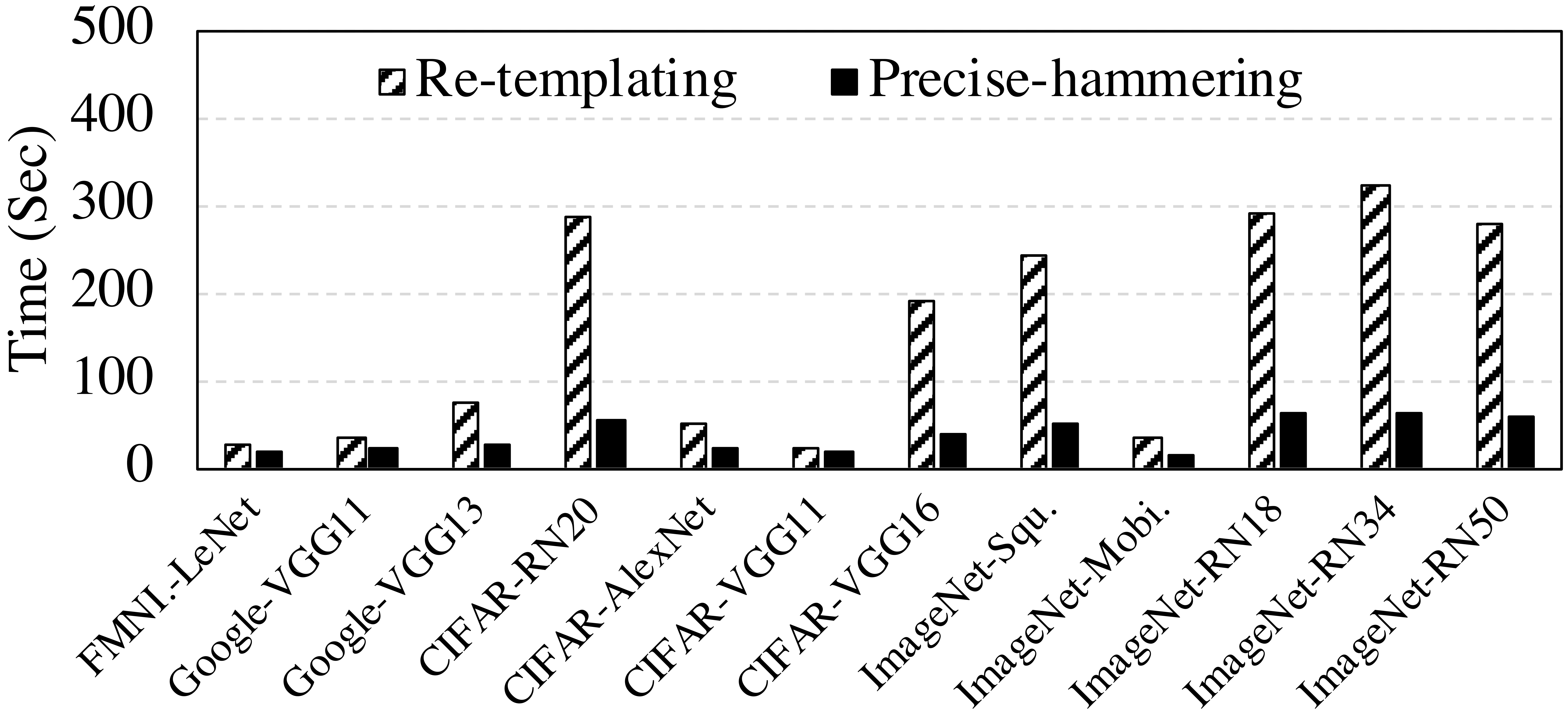
Evaluation: Bit Search Results

Dataset	Architecture	Network Parameters	Acc. Before Attack (%)	Random Guess Acc. (%)	Acc. After Attack (%)	Min. # of Bit-flips
Fashion MNIST	LeNet	0.65M	90.20	10.00	10.00	**3**
Google Speech Command	VGG-11	132M	96.36	8.33	3.43	**5**
	VGG-13	133M	96.38		3.25	**7**
CIFAR-10	ResNet-20	0.27M	90.70	10.00	10.92	**21**
	AlexNet	61M	84.40		10.46	**5**
	VGG-11	132M	89.40		10.27	**3**
	VGG-16	138M	93.24		10.82	**13**
ImageNet	SqueezeNet	1.2M	57.00	0.10	0.16	**18**
	MobileNet-V2	2.1M	72.01		0.19	**2**
	ResNet-18	11M	69.52		0.19	**24**
	ResNet-34	21M	72.78		0.18	**23**
	ResNet-50	23M	75.56		0.17	**23**

Evaluation: Bit Search Results

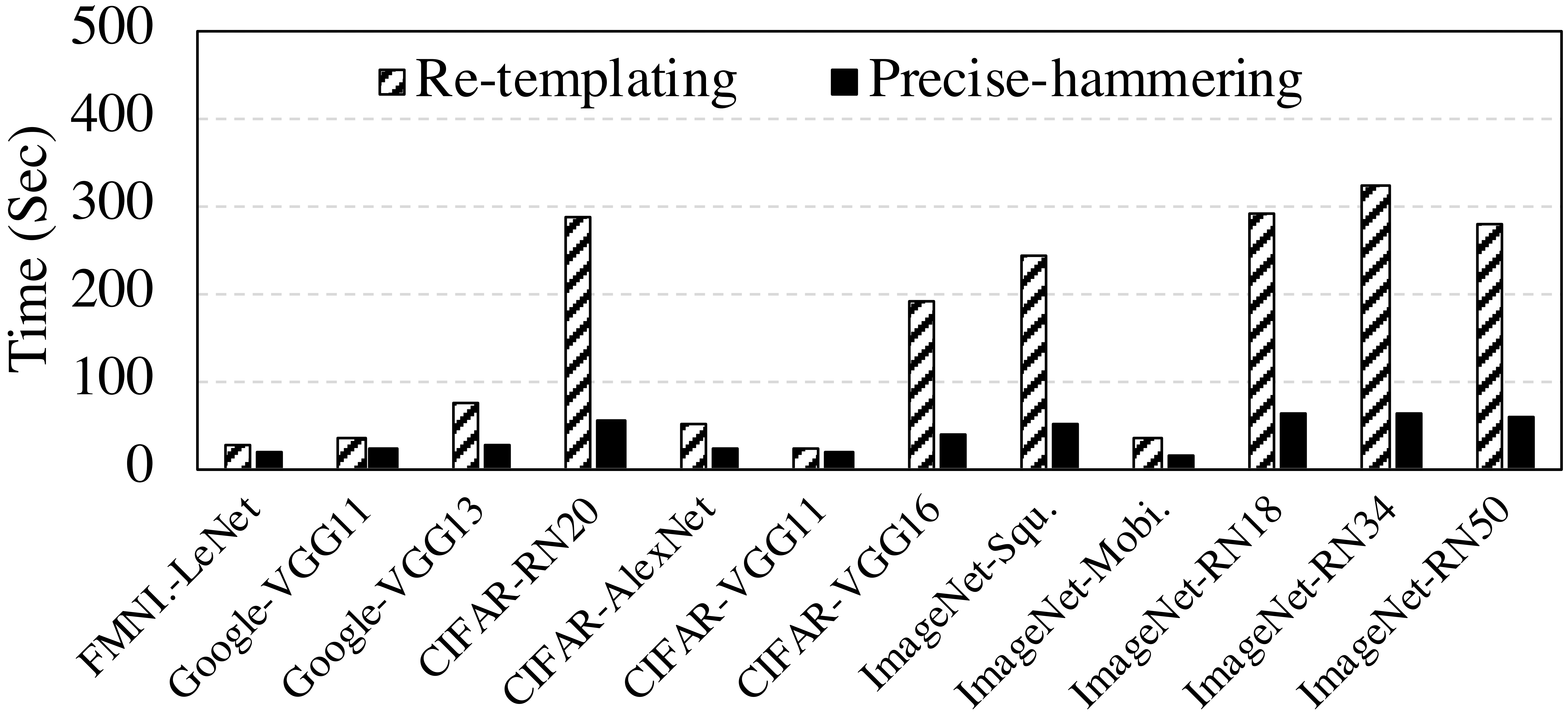
Dataset	Architecture	Network Parameters	Acc. Before Attack (%)	Random Guess Acc. (%)	Acc. After Attack (%)	Min. # of Bit-flips
Fashion MNIST	LeNet	0.65M	90.20	10.00	10.00	**3**
Google Speech Command	VGG-11	132M	96.36	8.33	3.43	**5**
	VGG-13	133M	96.38		3.25	**7**
CIFAR-10	ResNet-20	0.27M	90.70	10.00	10.92	**21**
	AlexNet	61M	84.40		10.46	**5**
	VGG-11	132M	89.40		10.27	**3**
	VGG-16	138M	93.24		10.82	**13**
ImageNet	SqueezeNet	1.2M	57.00	0.10	0.16	**18**
	MobileNet-V2	2.1M	72.01		0.19	**2**
	ResNet-18	11M	69.52		0.19	**24**
	ResNet-34	21M	72.78		0.18	**23**
	ResNet-50	23M	75.56		0.17	**23**

DeepHammer Runtime Exploitations



DeepHammer re-templating time and hammering time

DeepHammer Runtime Exploitations



DeepHammer re-templating time and hammering time

Refer to the paper for more details on the attack and mitigation dicussions

Conclusions

- We highlighted that **multiple deterministic bit flips** are required to tamper **quantized DNN models.**
- We proposed a new attack-**DeepHammer**-that depletes DNN intelligence through DRAM fault injections.
- We designed novel **algorithm-** and **system-level techniques** that enable **internal tampering** of DNNs with DeepHammer.
- Our work motivates the need to **enhance the robustness of DNNs** against **hardware-based fault injections.**

Thanks! Questions?

Email: fan.yao@ucf.edu