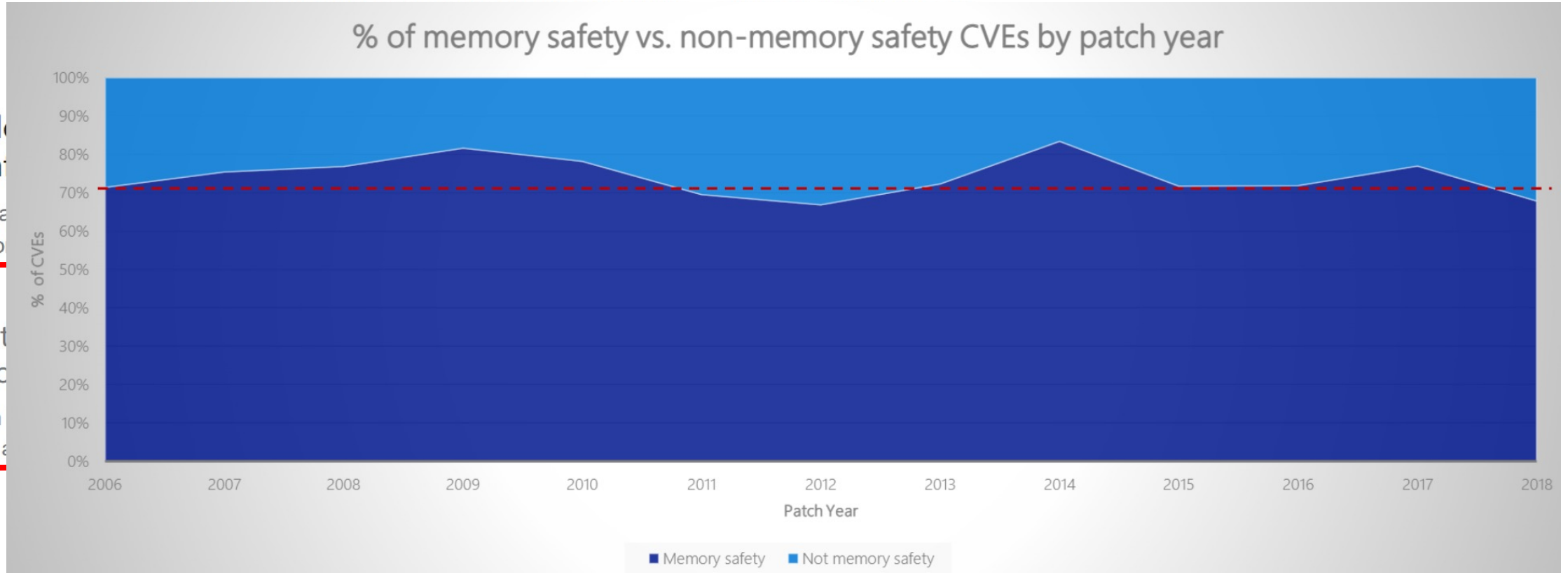
Temporal Exposure Reduction Protection for Persistent Memory

Yuanchao Xu, Chencheng Ye, Yan Solihin, Xipeng Shen

UNIVERSITY OF CENTRAL FLORIDA

Memory Security is Important

70% patches of Microsoft products are fixing memory safety issues!



https://github.com/microsoft/MSRC-Security-Research/tree/master/presentations/2019_02_BlueHatIL

Security is more Important for Persistent Memory (PM)

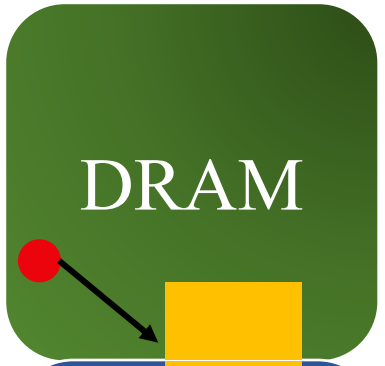
Application 1

Attacker

Application 2

Users

- Leak sensitive information
- Corrupted sensitive data
- Bypass policy enforcement
- …



Pointer (ld/st)

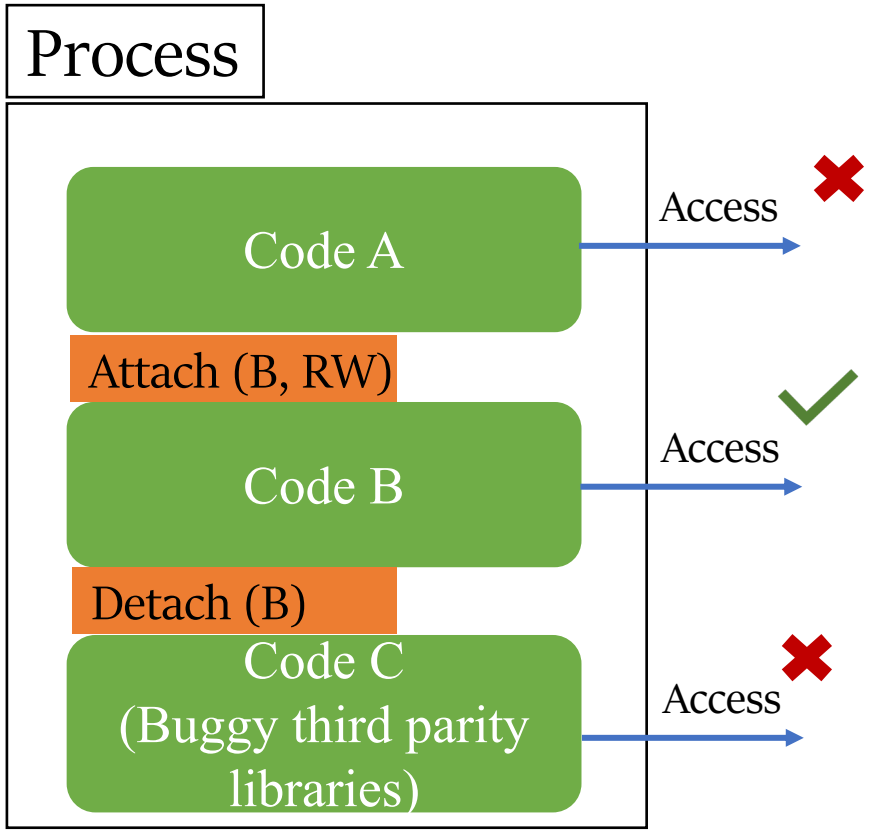
- Wrong results
- Dangling pointers
- Active attack payloads
- …

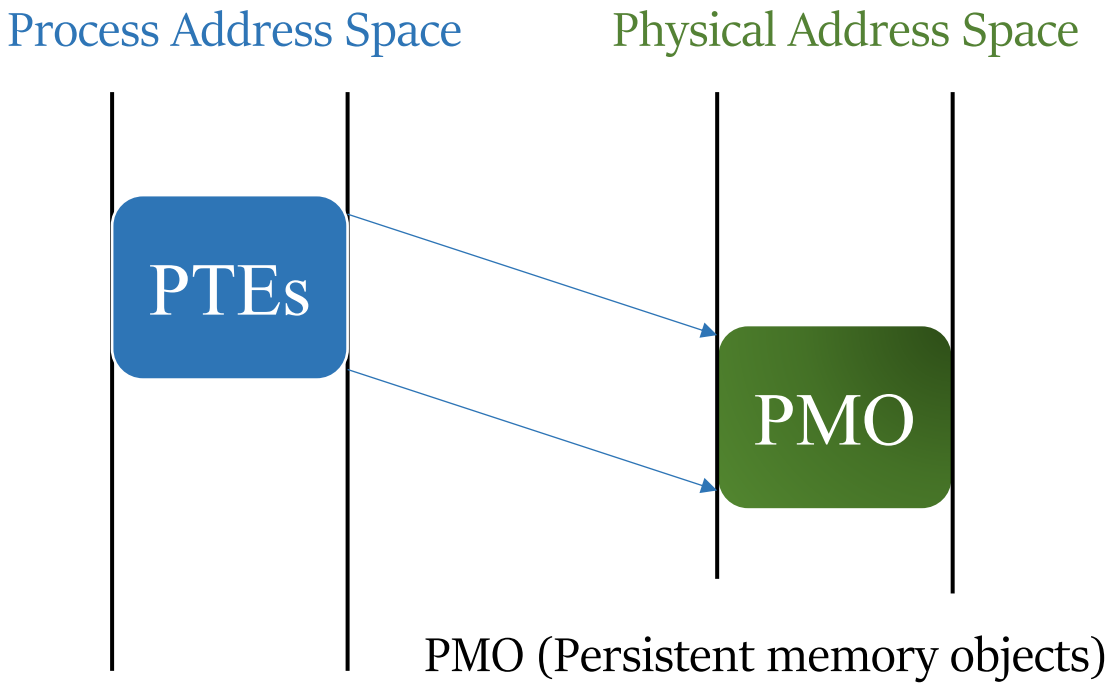
Improving Security with Temporal Exposure

Expose the data to process/thread only when it needs to access the data

(1) MERR [1]: attach and detach

- Process-level map and ummap system calls for PM
- Provide address randomization





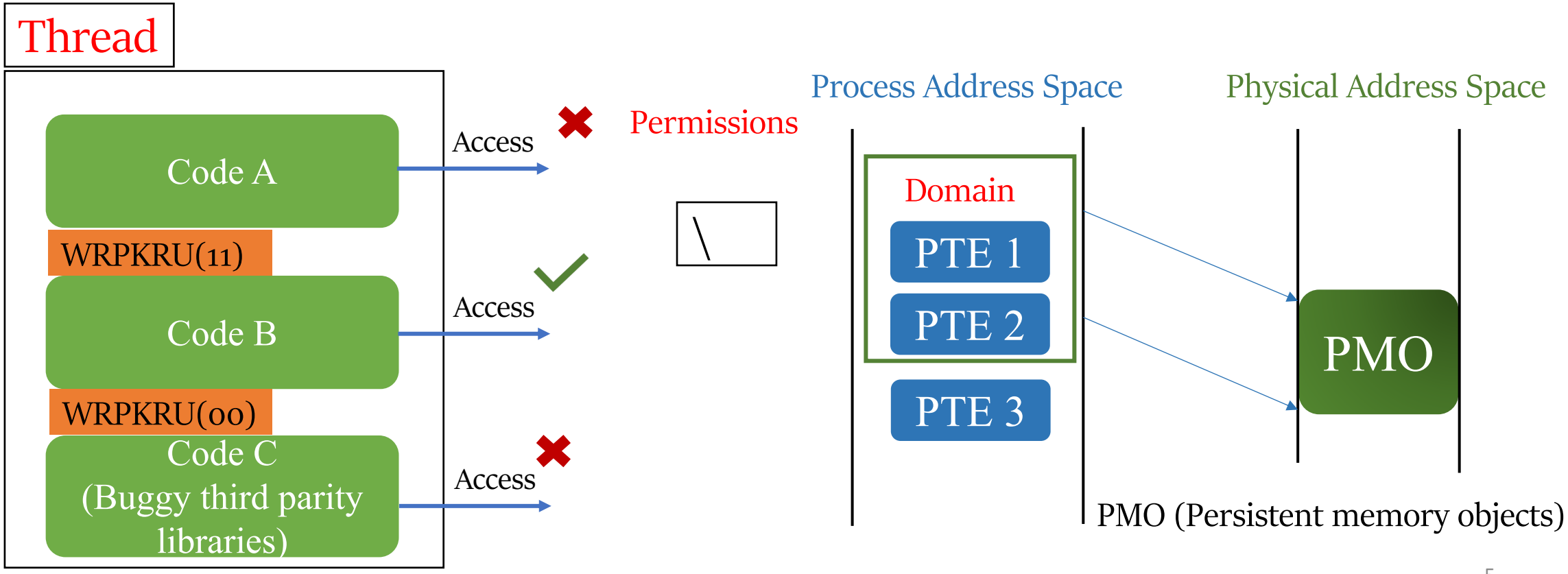
[1] MERR: Improving Security of Persistent Memory Objects via Efficient Memory Exposure Reduction and Randomization, Xu et al. (ASPLOS 2020)

Improving Security with Temporal Exposure

Expose the data to process/thread only when it needs to access the data

(2) Intel Memory Protection Keys (MPK) [1]: WRPKRU

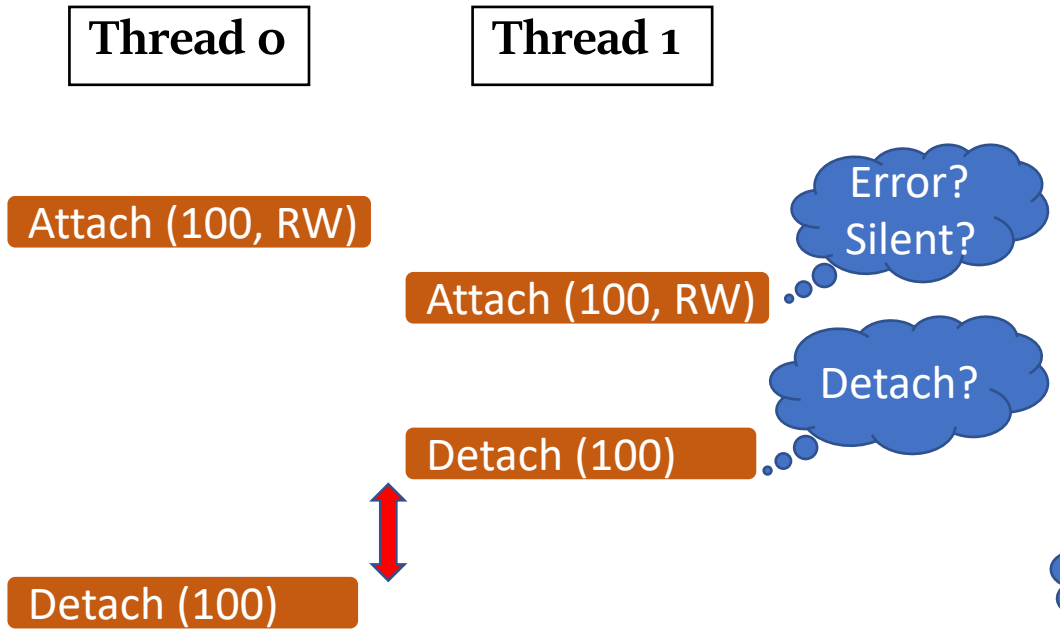
Thread-level permission control for memory domains



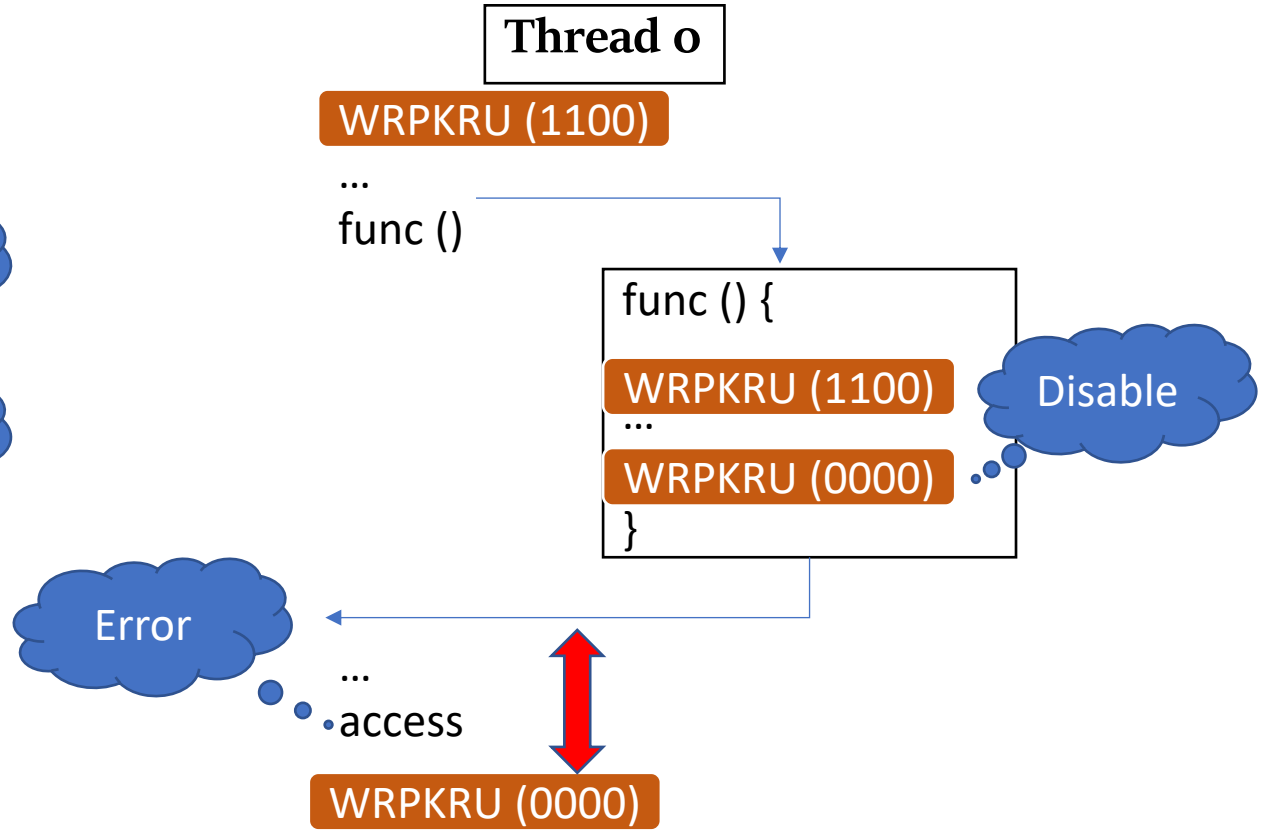
[1] https://www.kernel.org/doc/html/latest/core-api/protection-keys.html

Adoption Challenge I: Lack of Composability

(1) Multithread composability

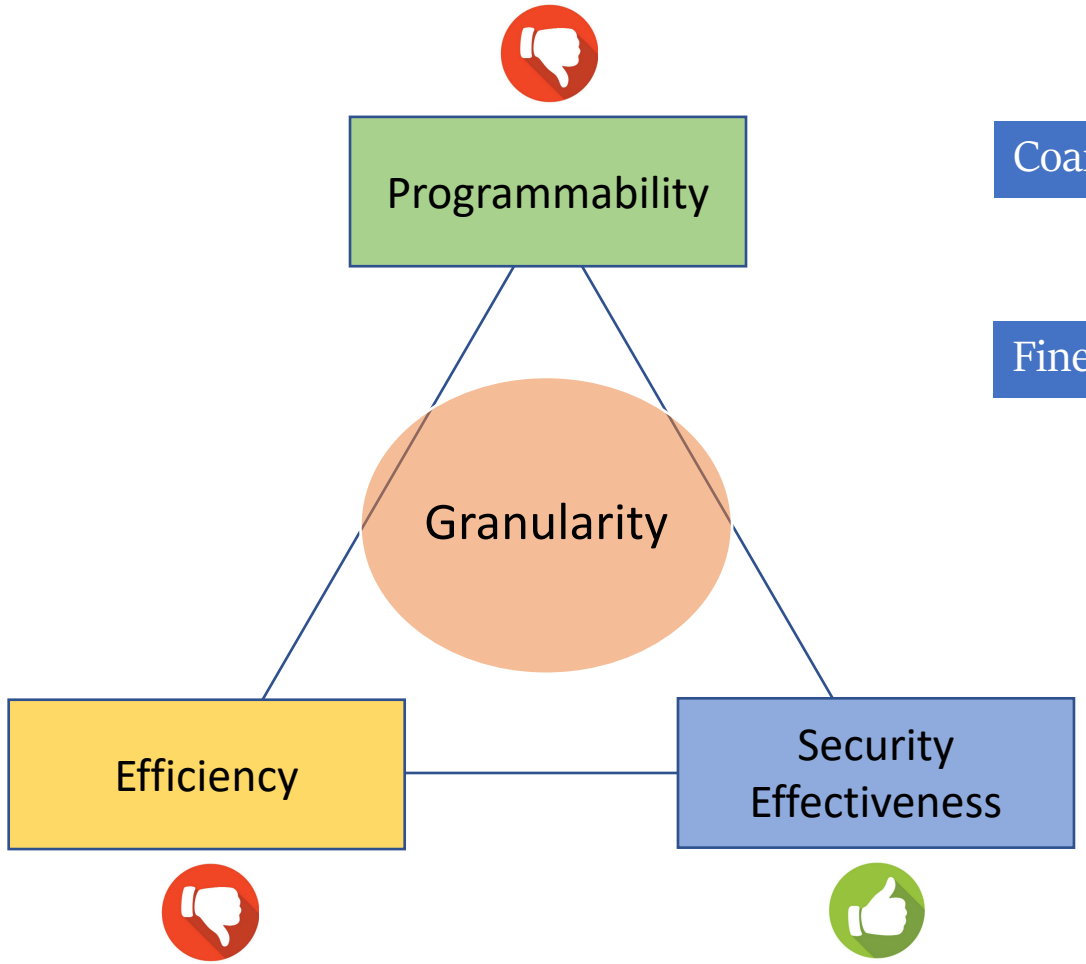


(2) Function composability for both



Adoption Challenge II: Three-way Tension

Security Goal: All Exposure Window (EW) is equal to or smaller than the target.



Coarse-grained Insertion

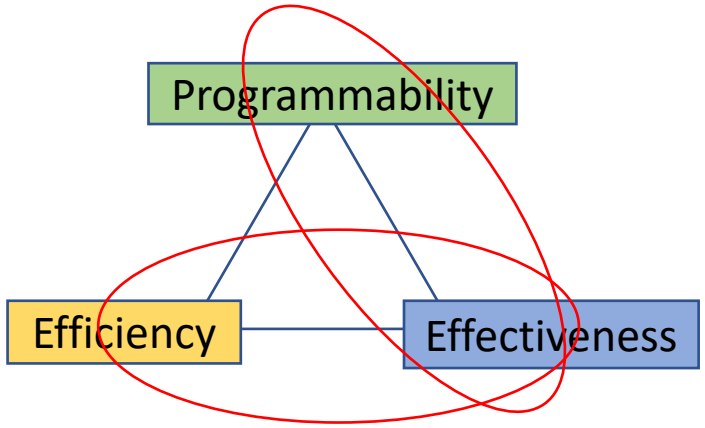
Fine-grained Insertion

```
Attach ()
if () {
  for () {   Attach ()
    ...      Detach ()
  }
  if () {    Attach ()
  ...        Detach ()
  }
  else {
     for ()  Attach ()
        ...  Detach ()
   ...
  }
} Attach ()
...
Detach ()
```

Contributions

Adoption Challenge I: Lack of Composability

Adoption Challenge II: Three-way Tension



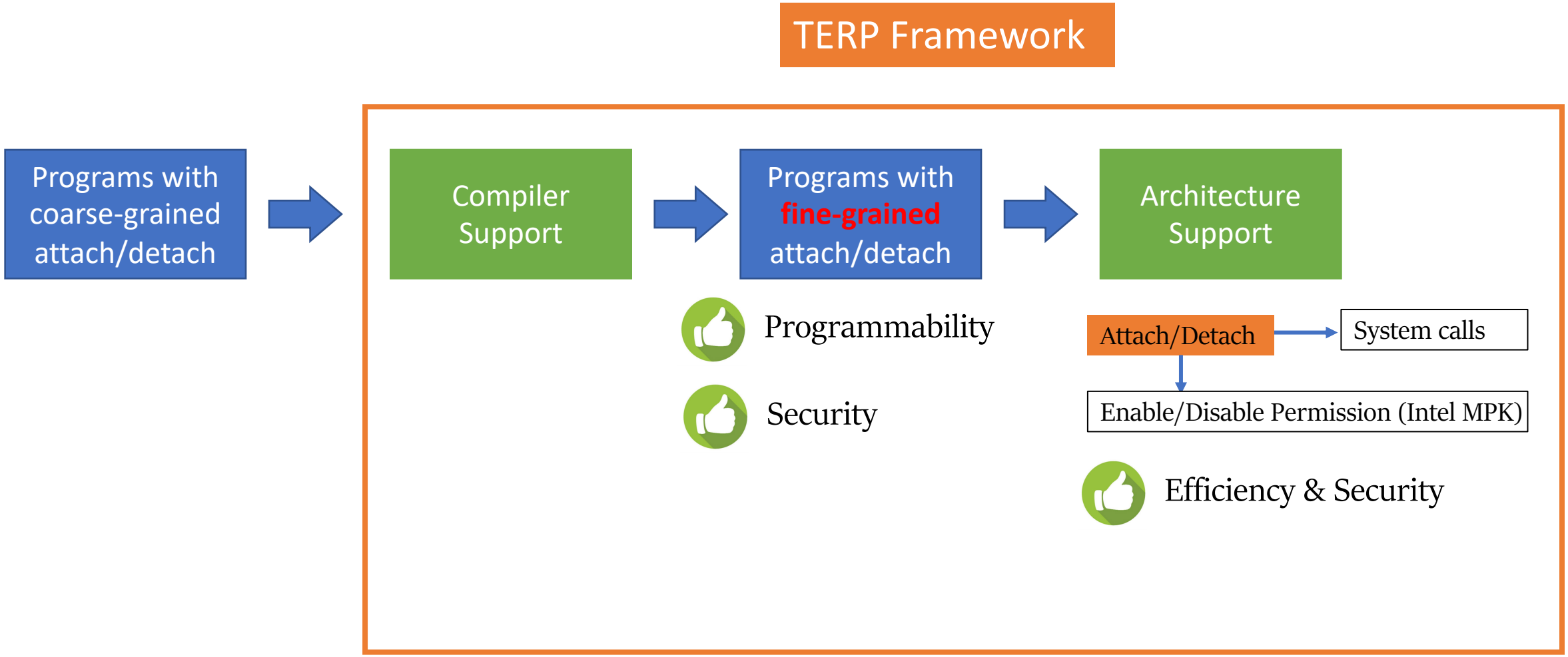
TERP: Temporal Exposure Reduction Protection

EW-conscious Semantics

Compiler Support

Architecture Support to Combine EWs

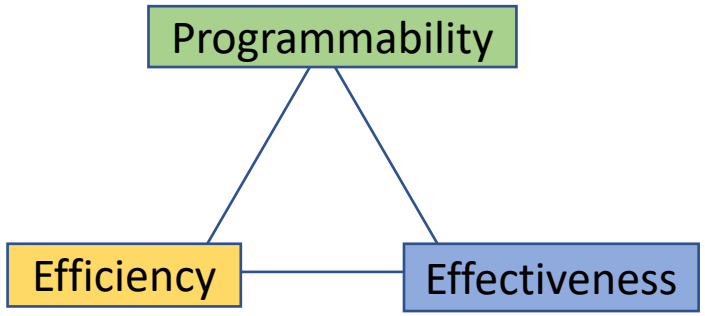
Overview of TERP



Contributions

Adoption Challenge I: Lack of Composability

Adoption Challenge II: Three-way Tension



TERP: Temporal Exposure Reduction Protection

EW-conscious Semantics

Compiler Support

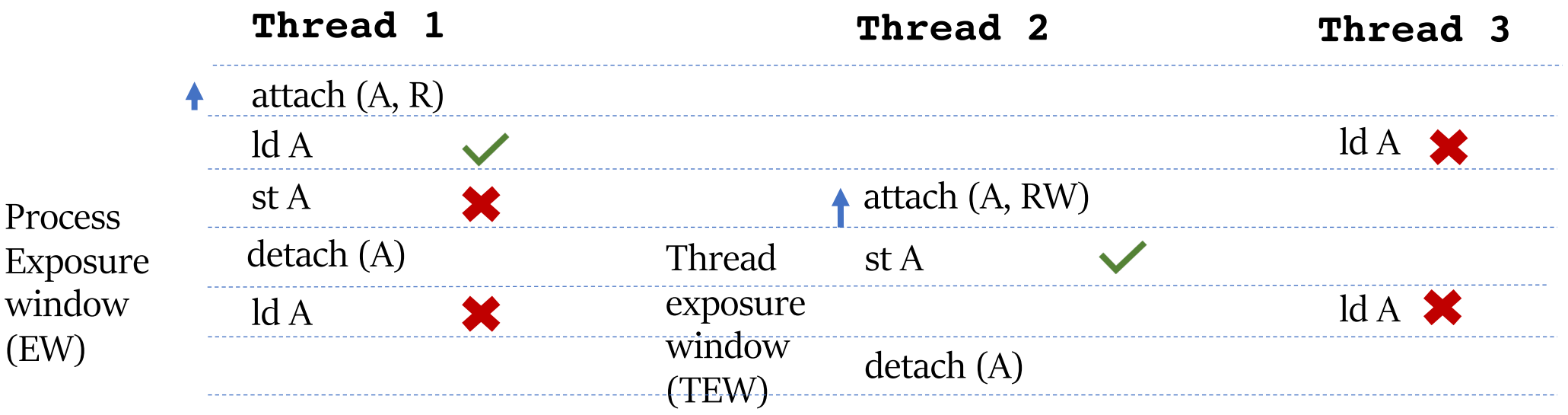
Architecture Support to Combine EWs

EW-conscious semantics

Semantic Definition

- Attach and detach may conditionally executed as thread permission enabling/disabling
- No overlapped attach/detach in one thread
- Allow temporal overlapped

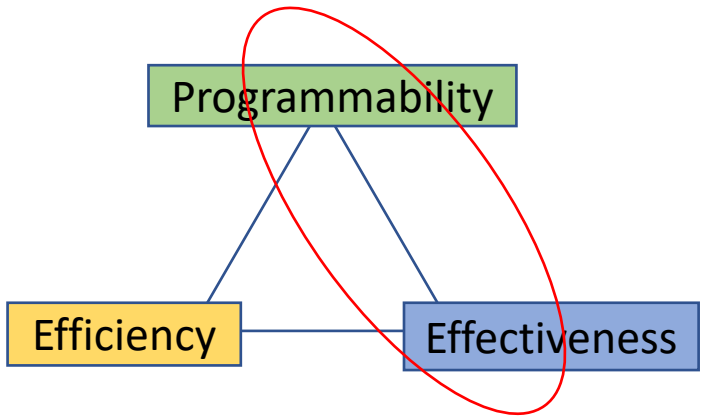
More detail in the paper about other semantic exploration and formulation



Contributions

Adoption Challenge I: Lack of Composability

Adoption Challenge II: Three-way Tension



TERP: Temporal Exposure Reduction Protection

EW-conscious Semantics

Compiler Support

Architecture Support to Combine EWs

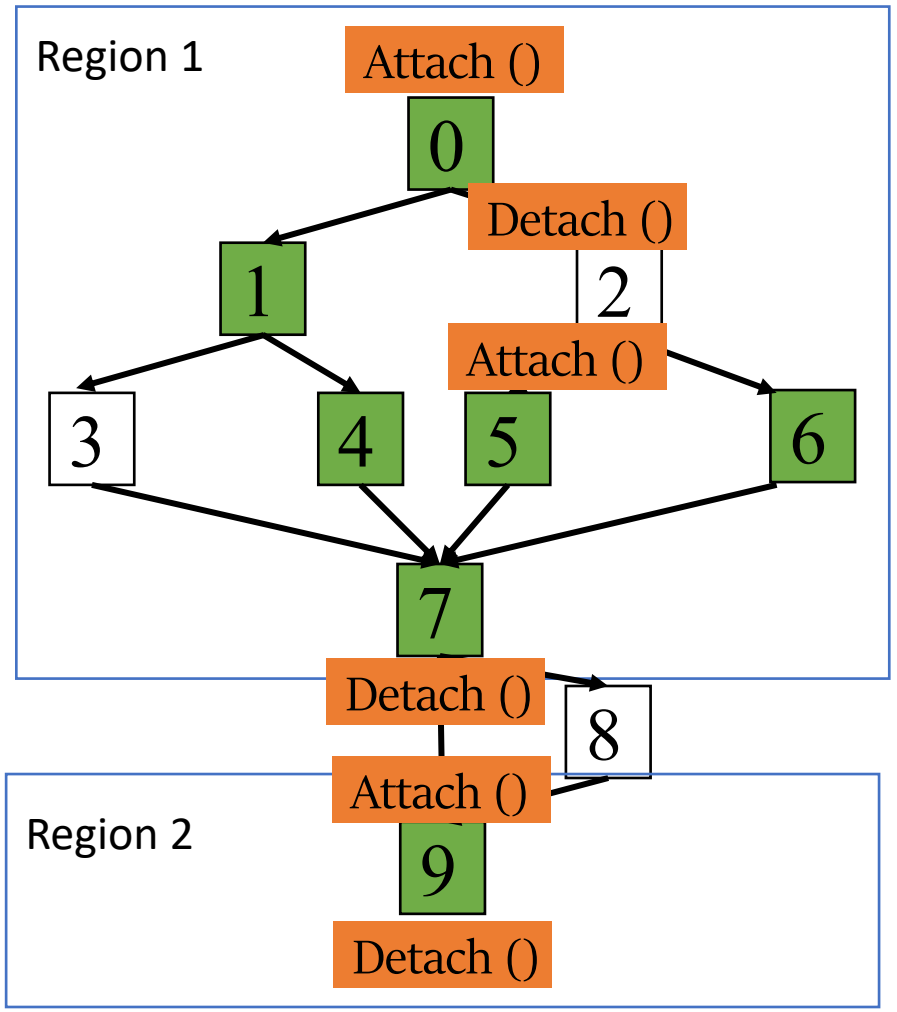
Compiler Support

Control flow graph

Region analysis

Small regions: Direct insertion

Large regions:
Path-sensitive Insertion



Control flow graph

Path-sensitive insertion:

Each EW < Target

Direct insertion:

Longest execution path < Target

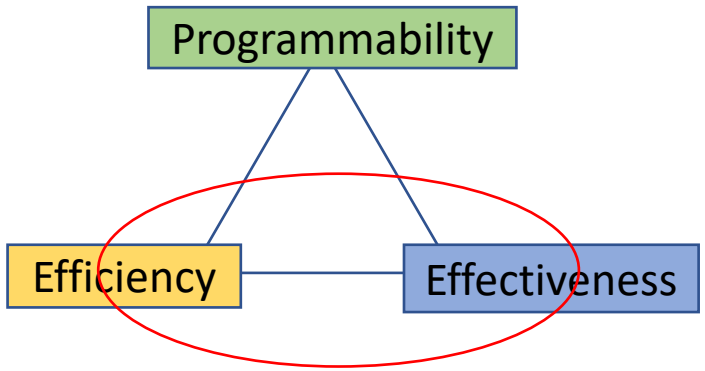
Basic blocks with PMO accesses

Basic blocks without PMO access

Contributions

Adoption Challenge I: Lack of Composability

Adoption Challenge II: Three-way Tension



TERP: Temporal Exposure Reduction Protection

EW-conscious Semantics

Compiler Support

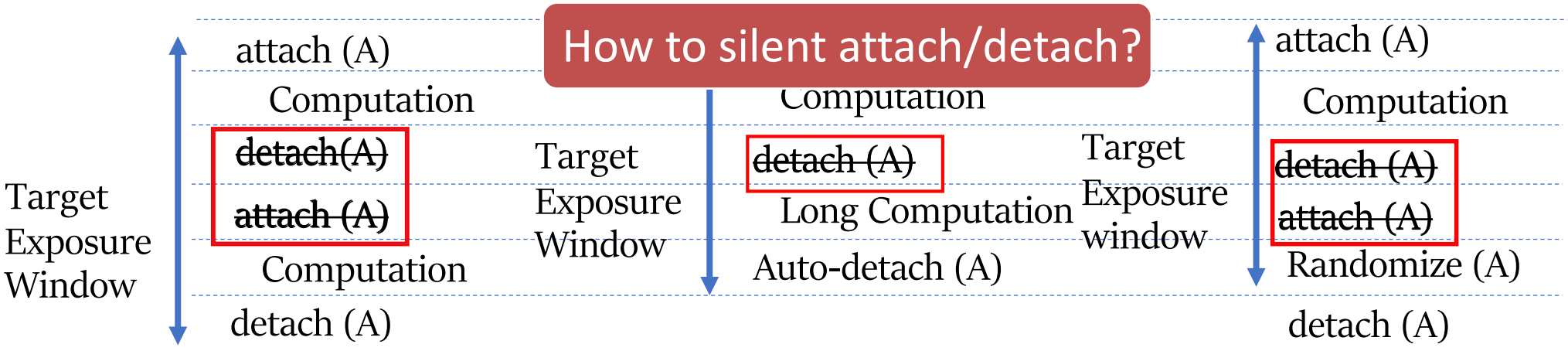
Architecture Support to Combine EWs

Exposure Window Combining

Code is inserted with small EW (2us) security goal.

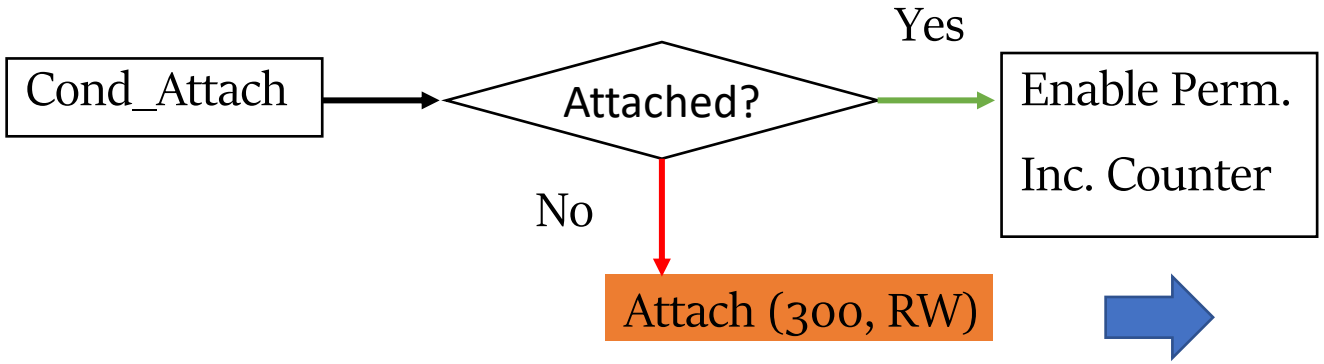
Dynamic combining to achieve the security goals

- EW (randomization) (e.g., 40us)
- TEW (permission) (e.g., 2us)



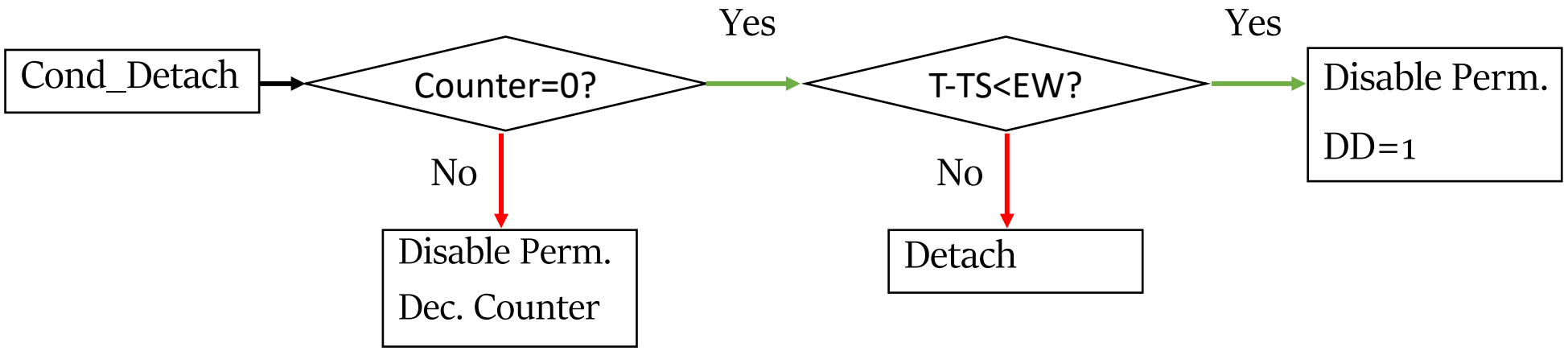
Full Combining **Mis-combining** **Partial Combining**

Architectural Support



Circular Buffer

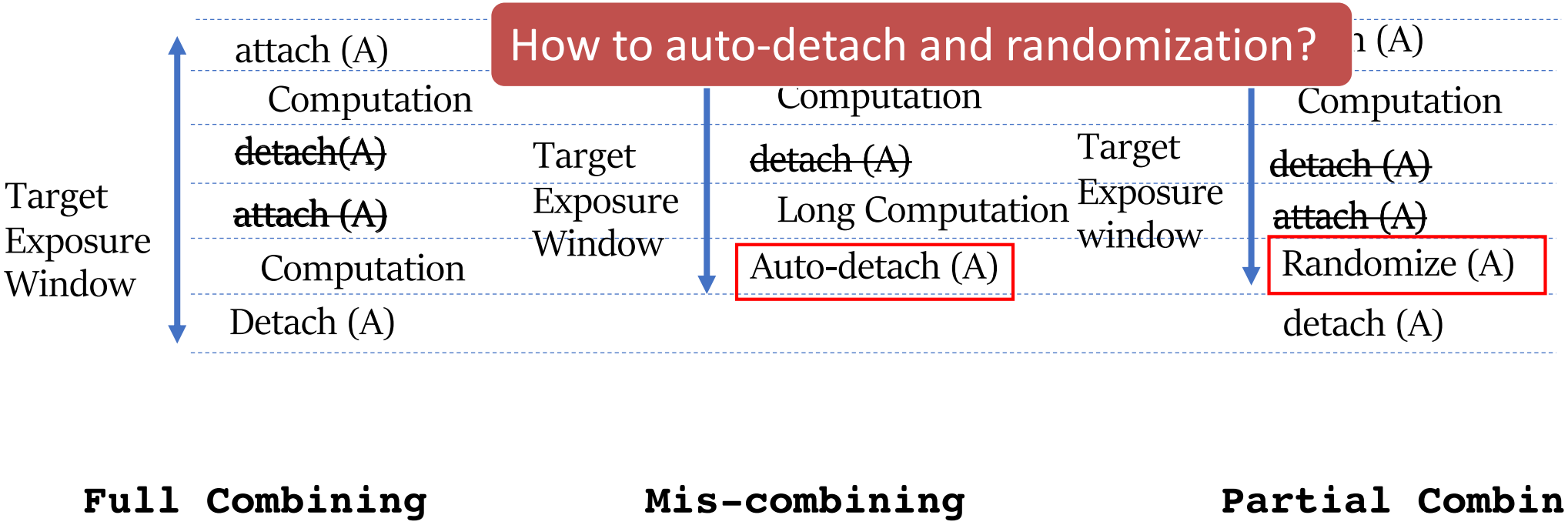
PMOID	Time stamp (TS)	Counter of attached threads	Delayed detach (DD)
100	3	0	1
200	5	2	0



Exposure Window Combining

Dynamic combining to achieve the security goals

- Exposure window (randomization) (e.g., 40us)
- Thread exposure window (permission) (e.g., 2us)



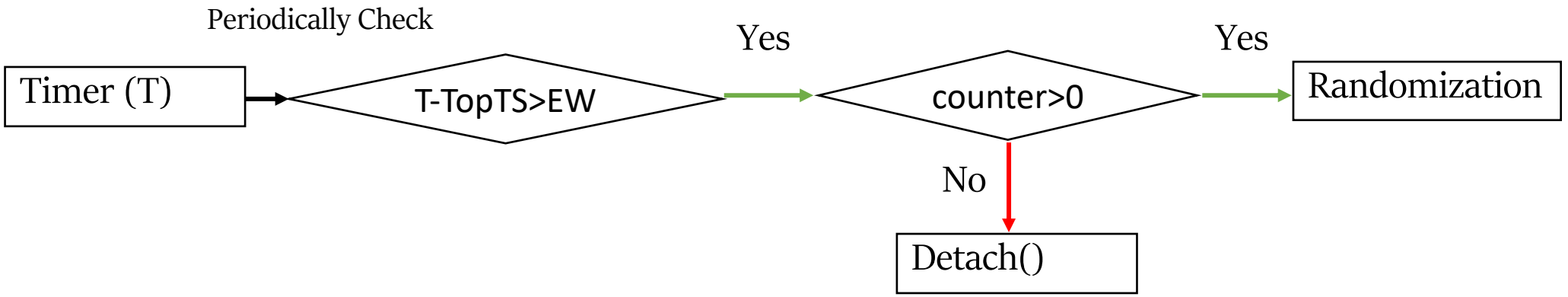
Full Combining **Mis-combining** **Partial Combining**

Architectural Support

Circular Buffer

PMOID	Time stamp (TS)	Counter of attached threads	Delayed detach (DD)
100	3	0	1
200	5	2	0

Attach (300, RW)



Evaluation

Benchmarks:

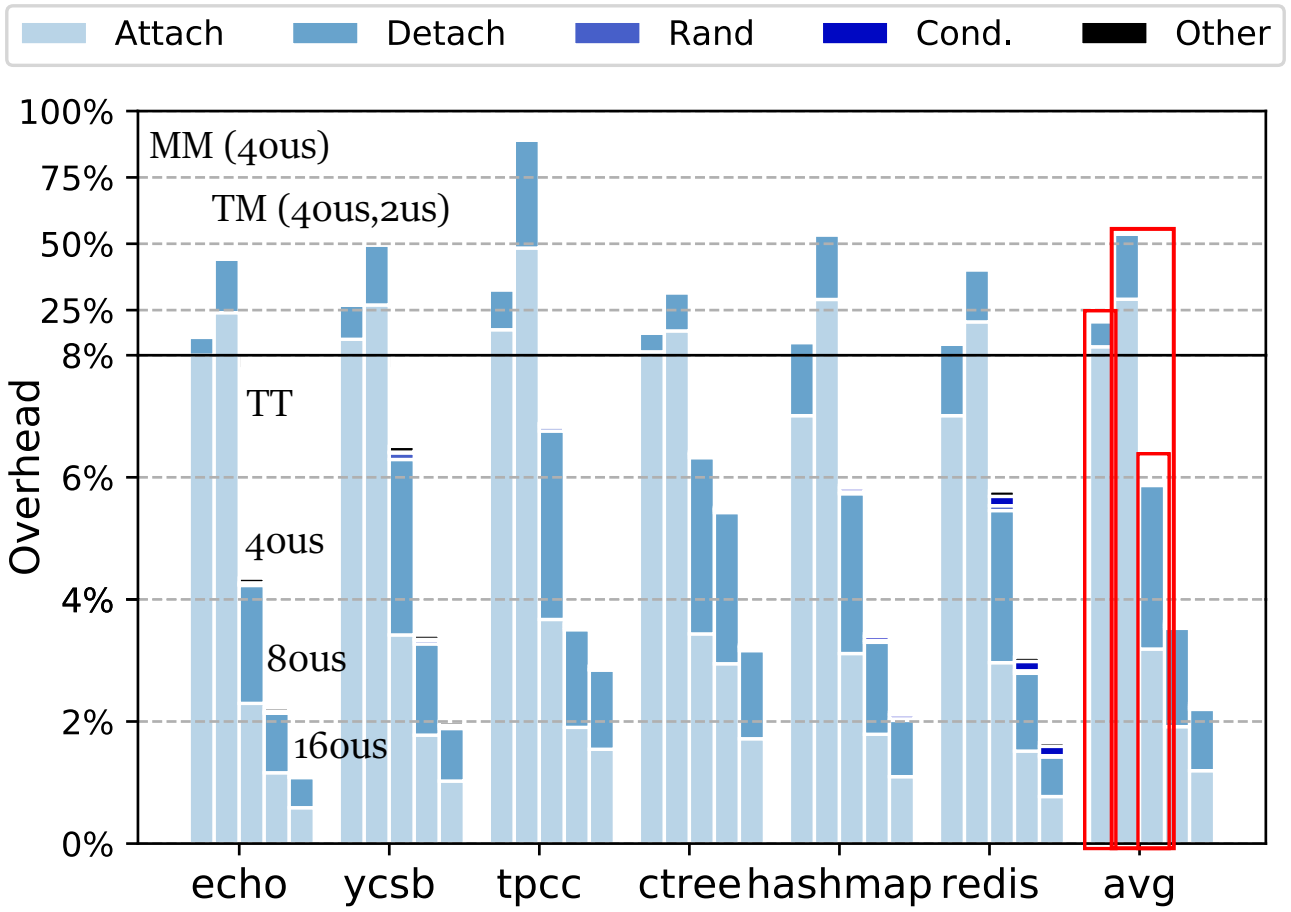
- WHISPER (single thread)
- SPEC2017 (multi-threaded)

Schemes	Insertion	Architecture
MM	Manual (insert & execute)	MERR architecture
TM	Compiler insertion	MERR architecture (no EW combining)
TT	Compiler insertion	TERP architecture

EW and TEW on WHISPER

Program	MM		TT	
	EW (us) avg/max	EW (us) avg/max	Slient (%)	TEW (us)
Echo	17.3/33.5	39.2/40.0	90.4	1.5
YCSB	13.1/38.1	39.4/40.0	87.3	0.9
TPCC	11.2/32.5	39.7/40.0	92.5	0.7
ctree	16.3/39.4	38.9/40.0	80.1	1.8
hash	19.7/37.2	39.5/40.0	91.2	0.9
Redis	8.1/25.1	39.5/40.0	91.1	1.1
Average	14.5/34.3	39.4/40.0	88.8	1.2

Results of WHISPER



Benefit from precise EW

Benefit from window combining

Conclusion

- Define the semantics of attach and detach constructs, which is **exposure window conscious semantics**.
- Design and implement compiler to **address TERP programming burden**.
- Design and implement architecture support to **guarantee security goals** and **improve efficiency.**
- Validate our design in WHISPER and SPEC2017, **reducing 70%-94% overhead and 96% attack success probability**.

Thank you for your attentions!

Q & A

backup

Data-oriented Attack

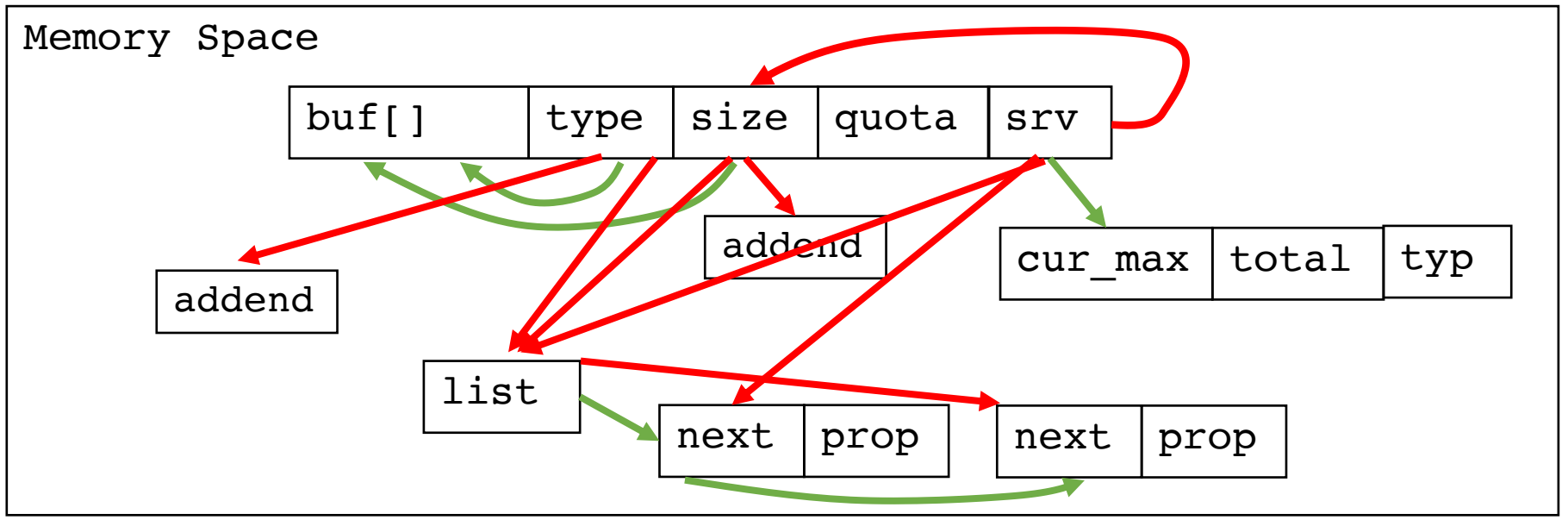
```
1  struct server {int *cur_max, total, typ;} *srv;
2  int quota = MAXCONN; int *size, *type
3  char buf[MAXLEN];
4  size = &buf[8]; type= &buf[12];
5  …
6  while (quota--) {
7  readData(sockfd, buf);
8    if (*type==NONE) break;
9    if (*type==STREAM)
10     *size=*(srv->cur_max);
11   else {
12     srv->typ = *type;
13     srv->total += *size;
14   }
15 }
```

```
1  struct Obj {struct Obj *next; int prop;}
2
3  void updatelist (struct Obj* list, int addend) {
4  for (; list != NULL; List=list->next)
5      list->prop += addend;
6  }
```

Data-oriented Attack

```
6  while (quota--) {
7    readData(sockfd, buf);
8     if (*type==NONE) break;
9     if (*type==STREAM)
10      *size=*(srv->cur_max);
11    else {
12      srv->typ = *type;
13      srv->total += *size;
14    }
15 }
```

```
1  struct Obj {struct Obj *next; int prop;}
2
3  void updatelist (struct Obj* list, int addend) {
4  for (; list != NULL; List=list->next)
5      list->prop += addend;
6  }
```



TERP Security

(1) Reduce data exposure window for PMOs (Intel MPK)

(2) Apply frequent address randomization for PMOs (MERR)