ARM Research Summit 2018

Software Techniques for Hardware Defects

Kyoungwoo Lee
Department of Computer Science
Yonsei University, Seoul, Korea

ARM Research Summit 2018

Software Techniques for Hardware Defects

Many thanks to collaborators:

Moslem Didehban[1], Sai Ram Dheeraj Lokam[1], Reiley Jeyapaul[3], Aviral Shrivastava[1]

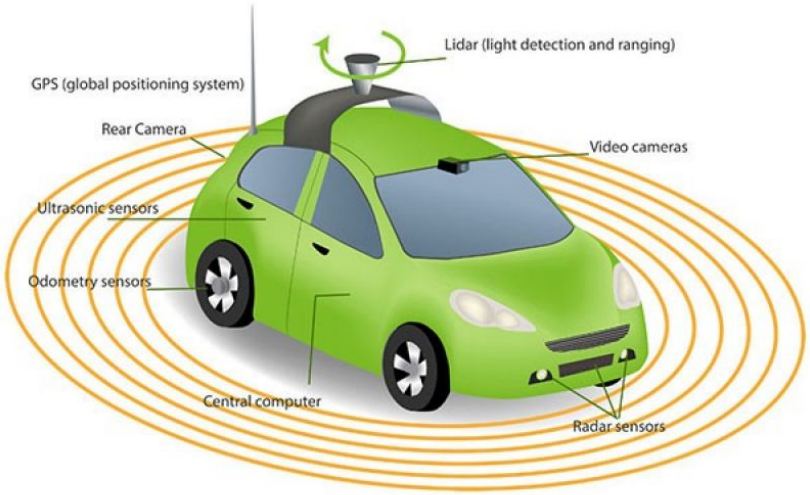
Hwisoo So[2], Yohan Ko[4], Youngbin Kim[2]

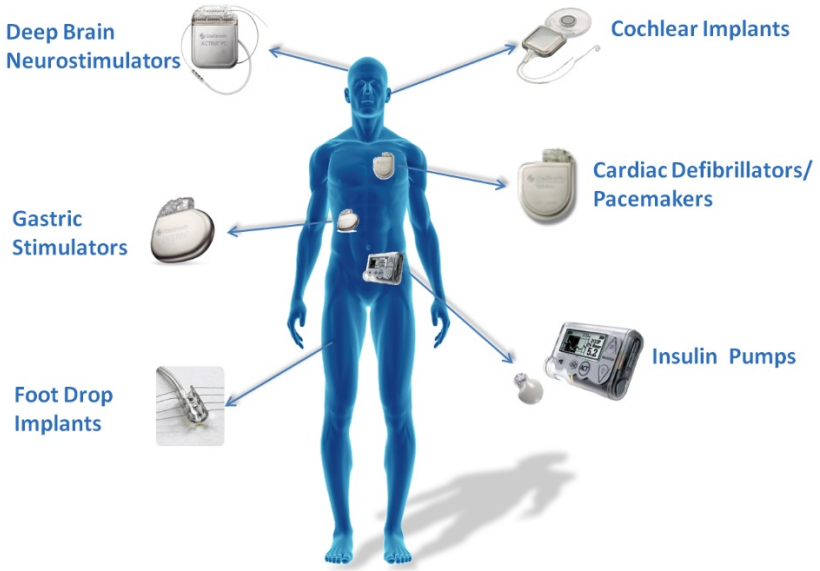
[1]Compiler Microarchitecture Lab, Arizona State University, Tempe, AZ

[2]Department of Computer Science, Yonsei University, Seoul, Korea

[3]ARM, Cambridge, UK, [4]SK Hynix, Icheon, Korea

Reliability is the main design concern

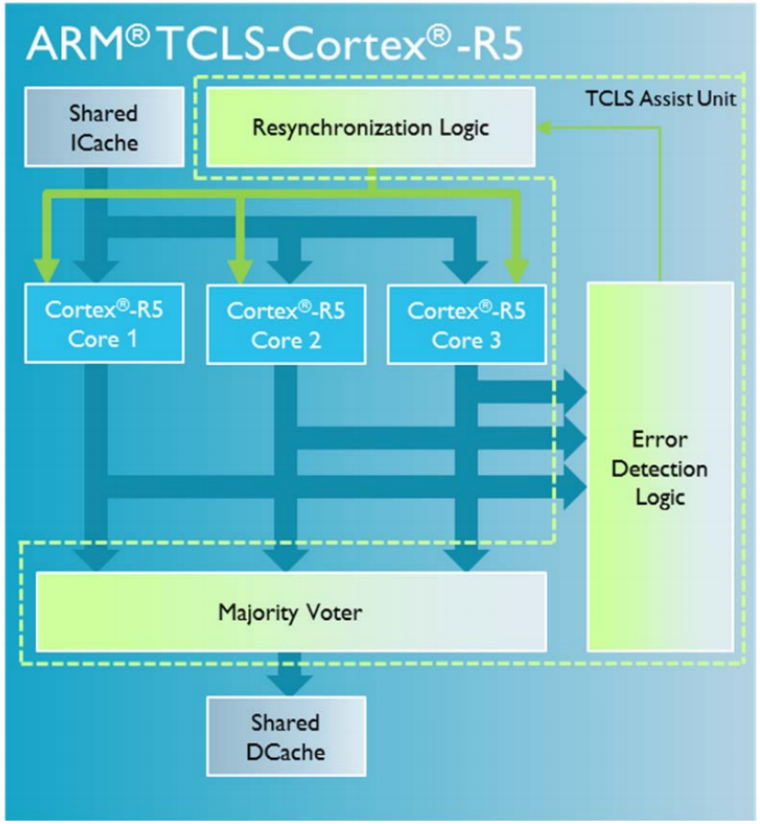




- **Main sources of hardware unreliability**
  - **Soft error, aka <u>transient</u> fault**
  - **Hard error, aka <u>permanent</u> fault**

Soft error protection is required

- **Soft errors: Historically, a problem for high-altitude applications**
- **ITRS 2015 predicts soon even ground-level applications will be at risk.**
- **Failure rate is expected to increase:**
  - **More components → more failures**
- **Solution: Redundancy**
  - **Hardware-level solutions**
    - **ARM Cortex-R Dual lockstep processor**
  - **Software-level solutions**
    - **Time redundancy (Flexible)**



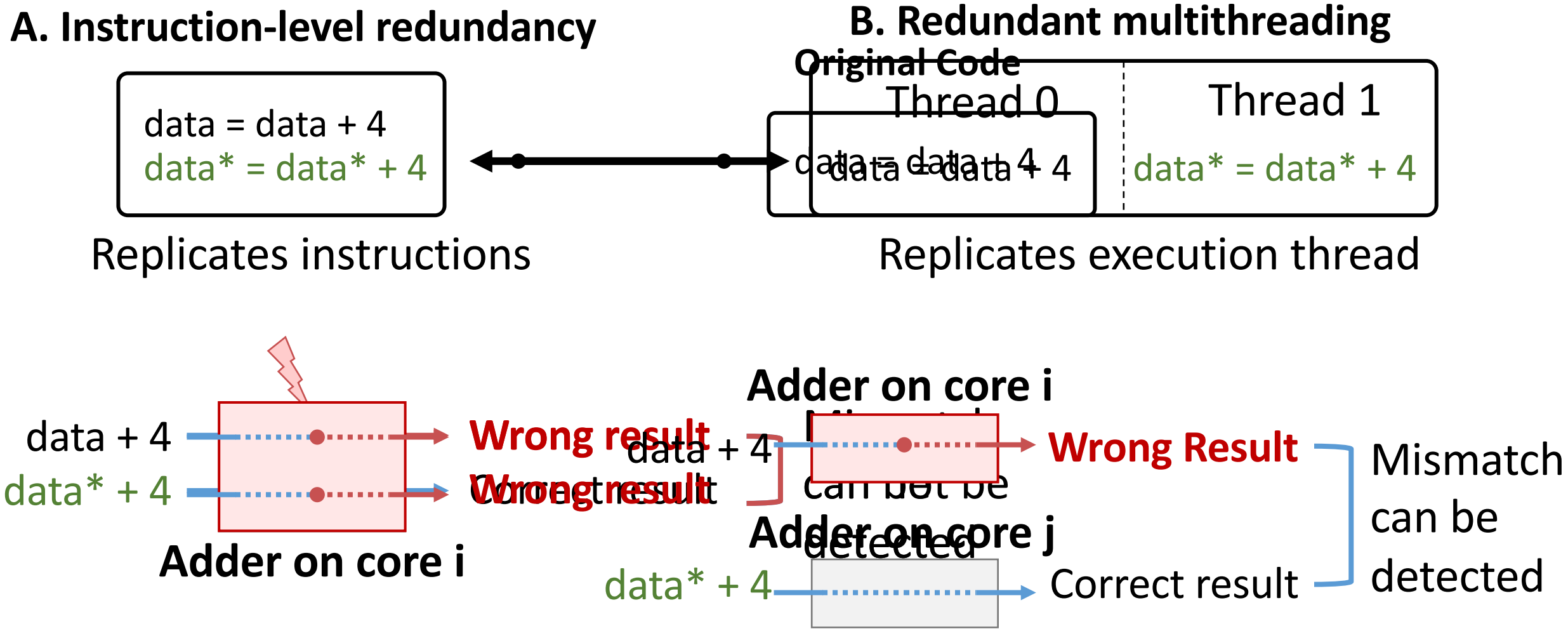
Software redundancy: flexible and effective

- **Software-level redundancy: flexible error detection**
  - **No hardware modification**
  - **Can provide flexibility and selective protections**

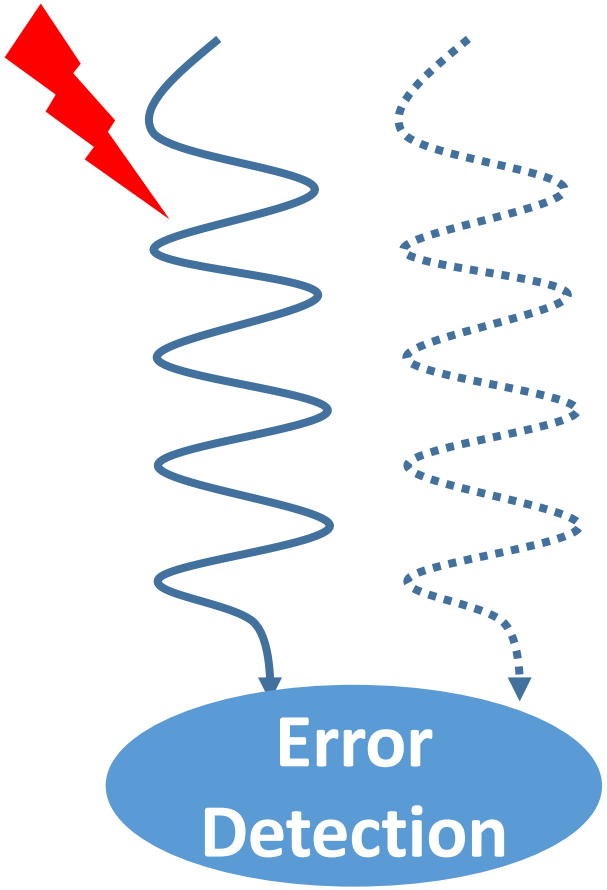
- **Main approaches of software-level redundancy are**

	Instruction-level redundancy	Redundant multithreading
Soft error	Can detect	Can detect
Hard error	**Can not detect**	Can detect

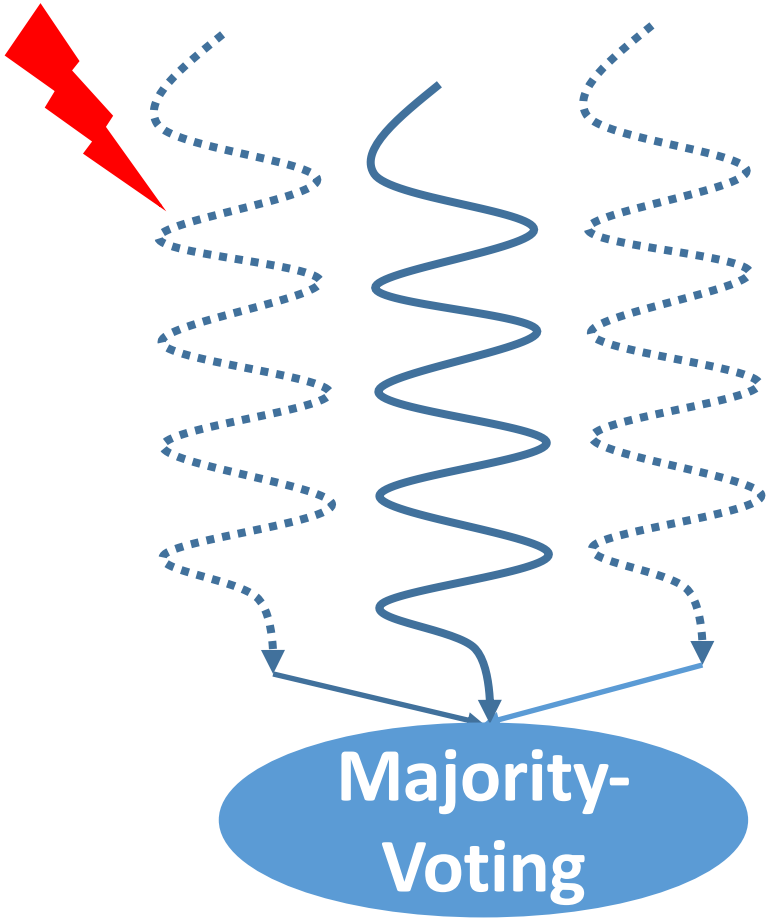
2-ways of software-level error detection



Instruction-level redundancy schemes



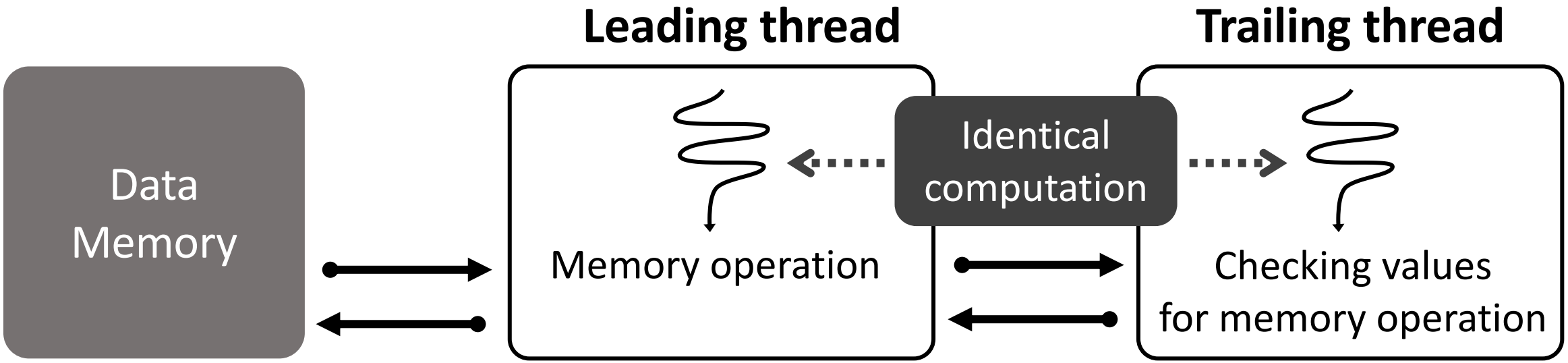
Examples: EDDI[2002], SWIFT[2005], Shoestring[2010], DRIFT[2013], SIMD-Based Soft Error Detection [16], IPAS [2016], nZDC [2016]



Examples: SWIFTR[2007], selective-SWIFTR[2013], ELZAR [2016]

Redundant multithreading schemes

- **SRMT: software-based redundant multithreading** [Wang, CGO '07]



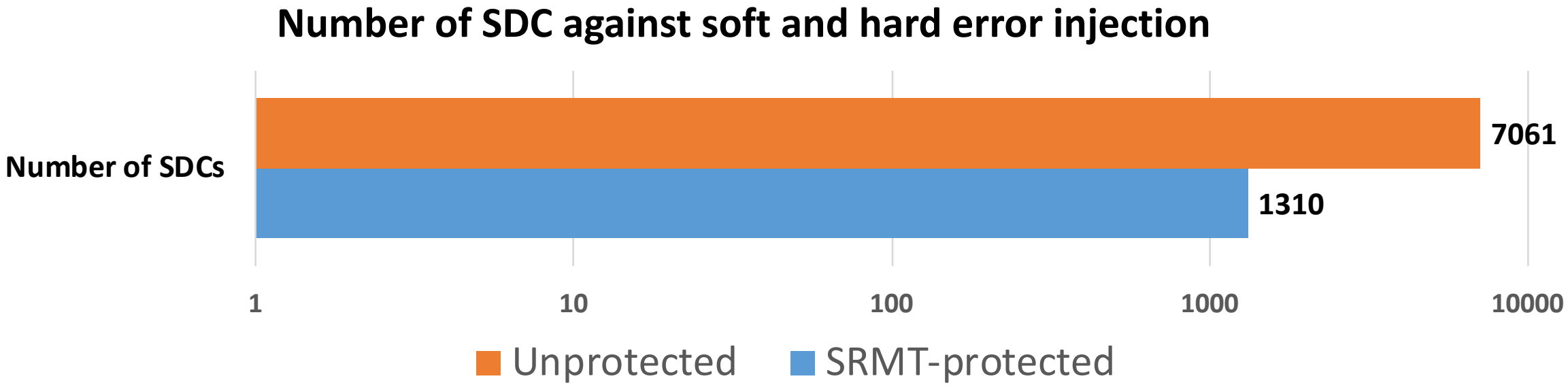
- **COMET**[Mitropoulou, CASES '16], **DAFT**[Zhang, IJPP '12]: **Improves runtime**
- **[Wadden, ISCA '14][Gupta, DAC '17]: Applies SRMT to GPU**
- **RedThreads**[Hukerikar, IJPP '16]: **Programmer-tunable SRMT for HPC**

Experiment setup: SRMT error coverage

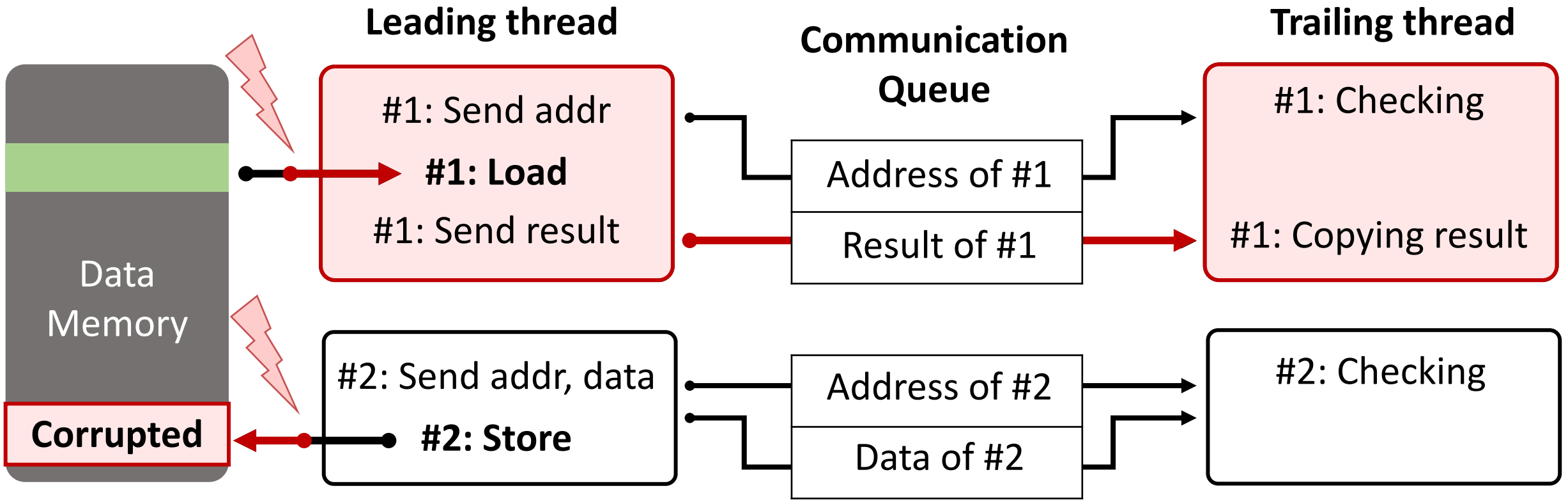
- **Benchmark: 9 applications in MiBench**
  - **Original / SRMT-protected**
    - ✓**Without hardware supports for inter-thread communication**
- **Fault Injection on cycle-accurate gem5 simulator**
  - **6 components for fault injection**
  - **1 error injection per 1 execution**
    - ✓**500 soft errors and 100 hard errors per component / benchmark**
- **Fault coverage validation**
  - **Main target: # of silent data corruption (SDC)**
  - **With correction factor[Schirmeier, DSN '15] (# of SDCs * runtime * # of cores)**

Experimental observation: SRMT error coverage

- **Total: 27,000 soft error and 5,400 hard error injections**
  - **For unprotected application vs. SRMT-protected application**
  - **On average, SRMT requires ~3.9x runtime than unprotected**
  - **2 cores are used for physically separated multithreading**
  - **Still, SRMT suffers from SDC (Silent Data Corruption)**



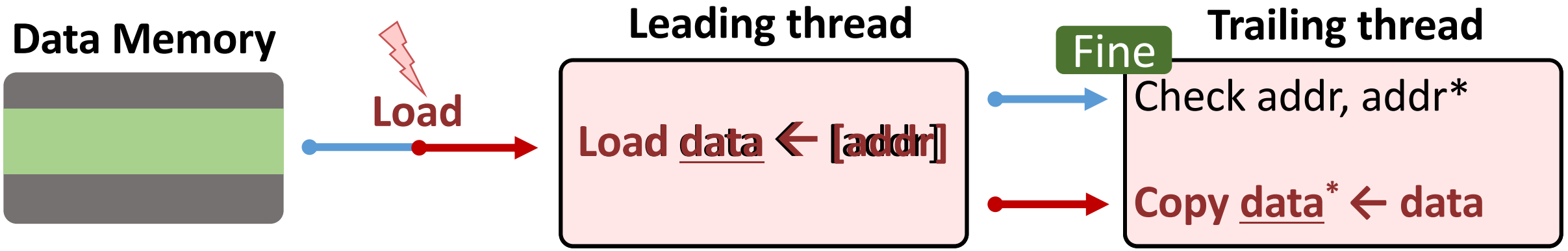
Why SRMT suffers vulnerability?



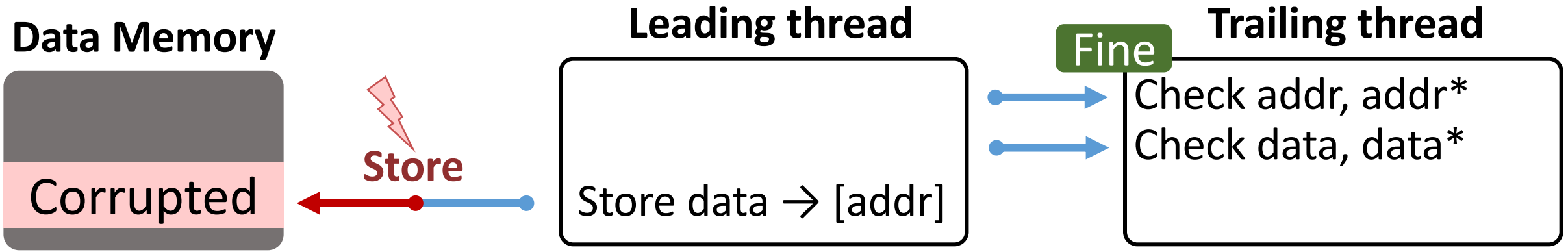
- **SRMT checking only checks old snapshot of registers**
  - **Incorrect execution of memory operation can be undetected**
    - ✓ **Vulnerable input replication & vulnerable output comparison**

SRMT: Error cases

- **Load in SRMT-protection**



- **Store in SRMT-protection**



EXPERT: Removing Vulnerability from LOAD

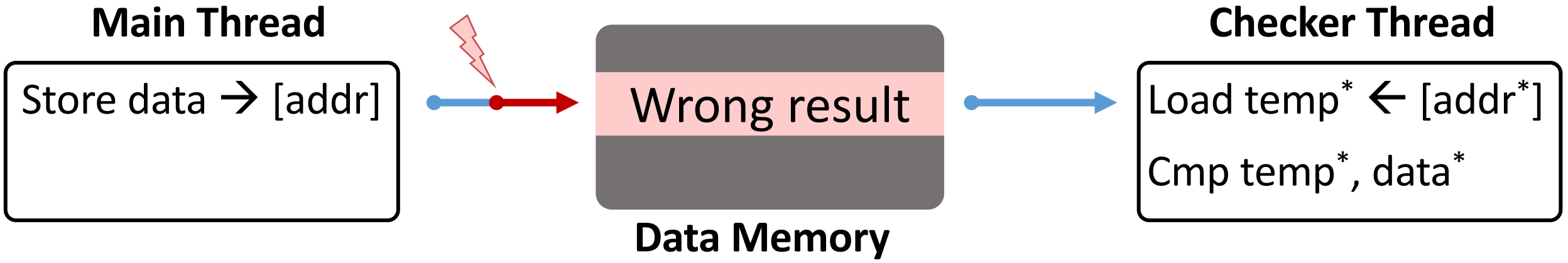
- **Replicating load operation on checker thread**

NOTE: Checker thread access memory with its local register

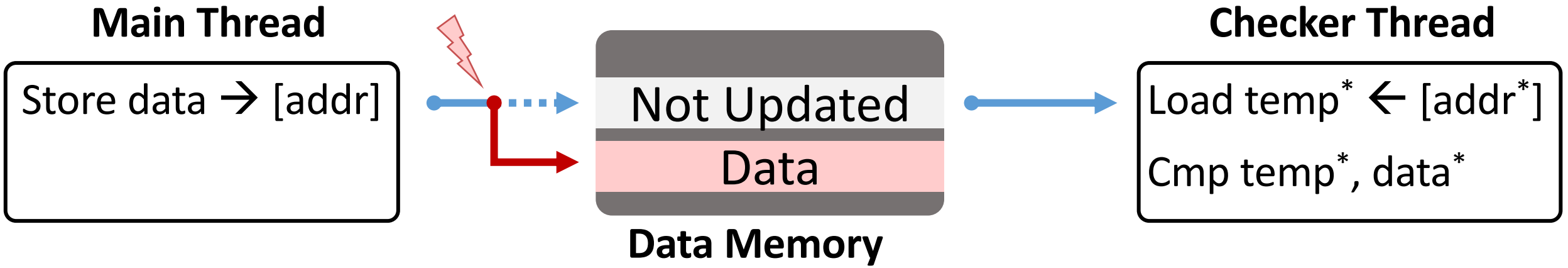
- **Soft error on load operation can only corrupt one thread**
  - **System can detect mismatch, as another thread is clean**
- **Checking for load operation is not necessary**
  - **Only store operation can propagate error effect**
  - **Mismatch will be found on later checking for store operation**

EXPERT: Load-back checking against error

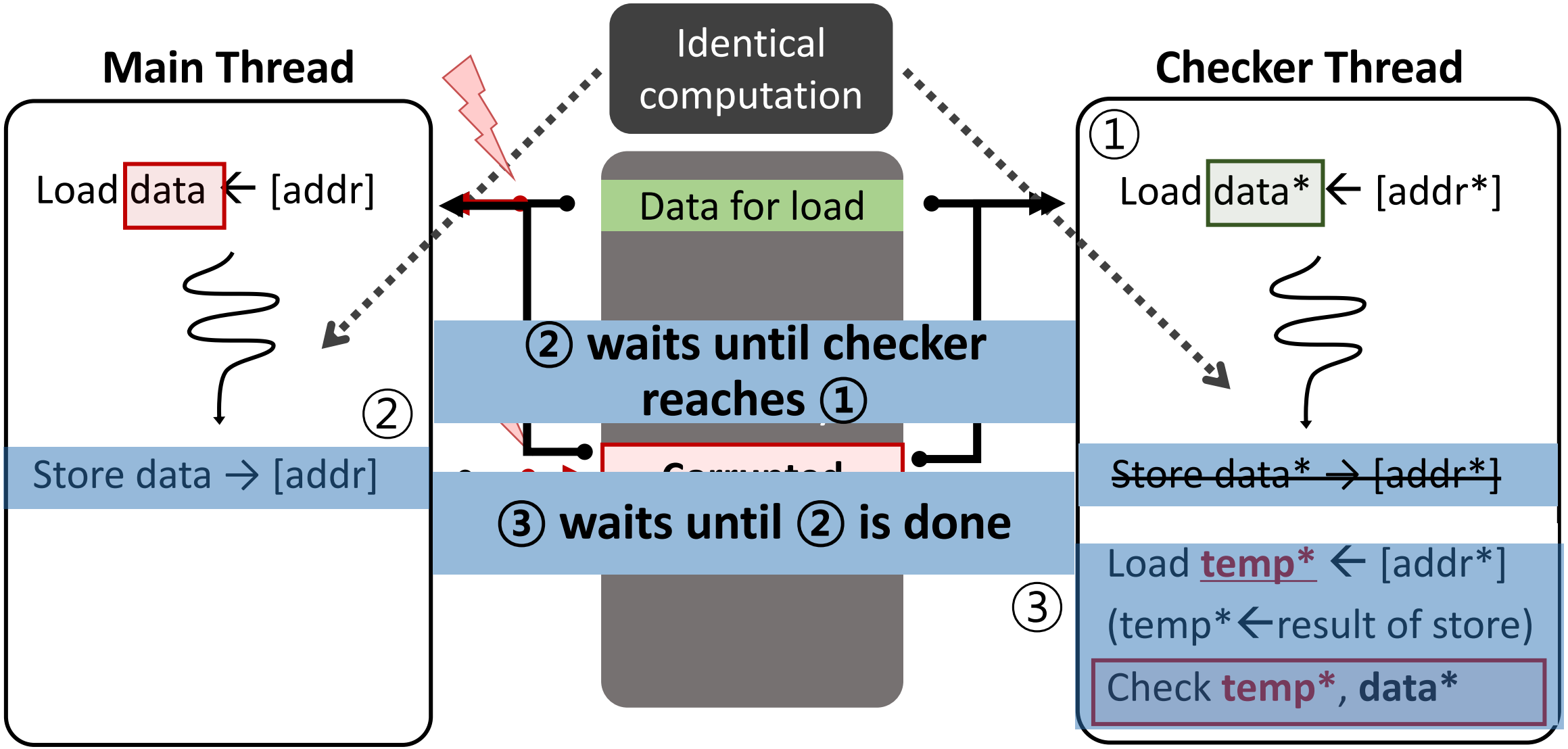
- **If error corrupts data of store operation**



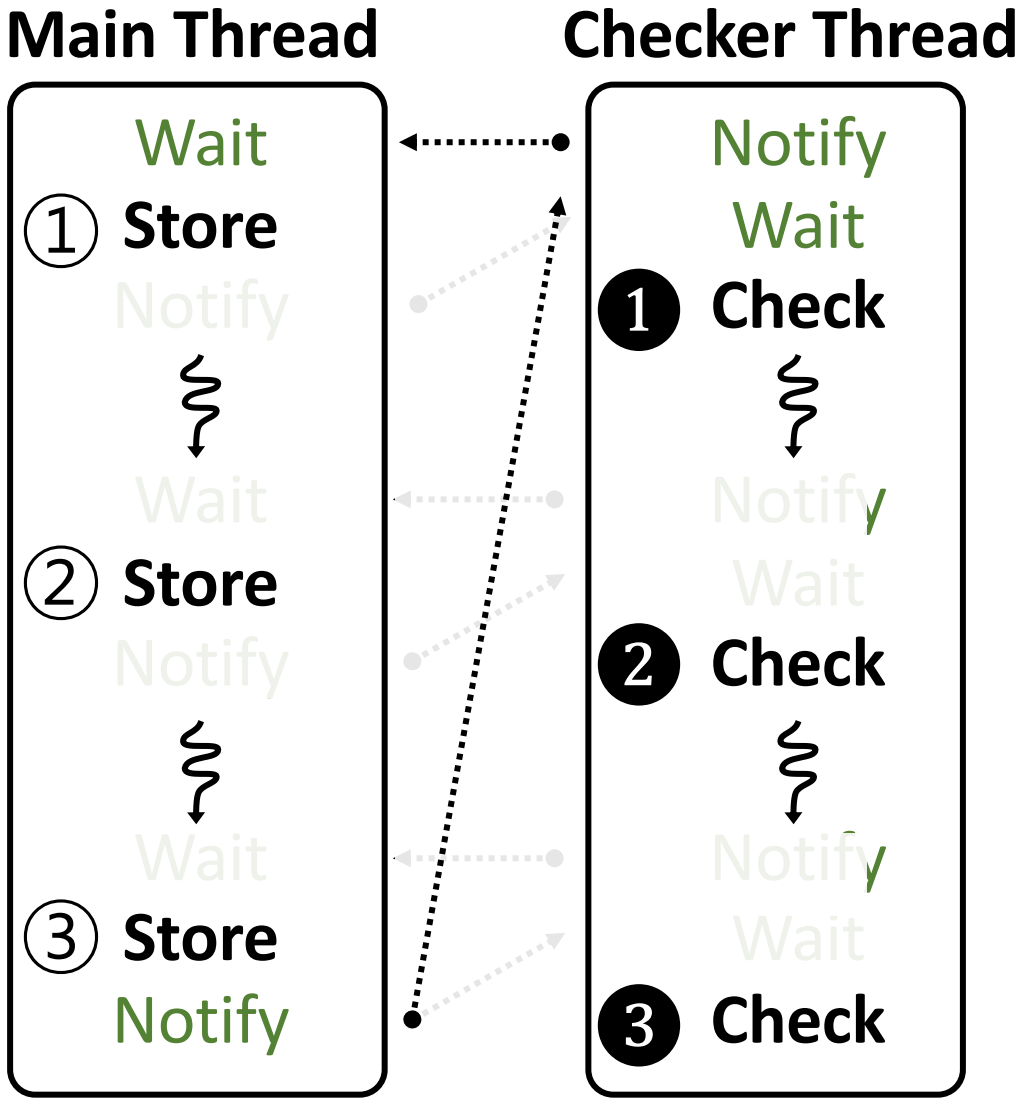
- **If error corrupts address of store operation**



EXPERT: Reliable software-level RMT



EXPERT: Store Packing Optimization



- **2-way sync for every store**
  - **~7.2x runtime on average**
- **If there is no dependency between ①, ②, and ③**
  - **Expert checking needs to keep**

① ➡ ❶

- **"Store Packing" is possible**
  - **If there is no memory dependency for both STORE and LOAD**
  - **~43% performance improvement**

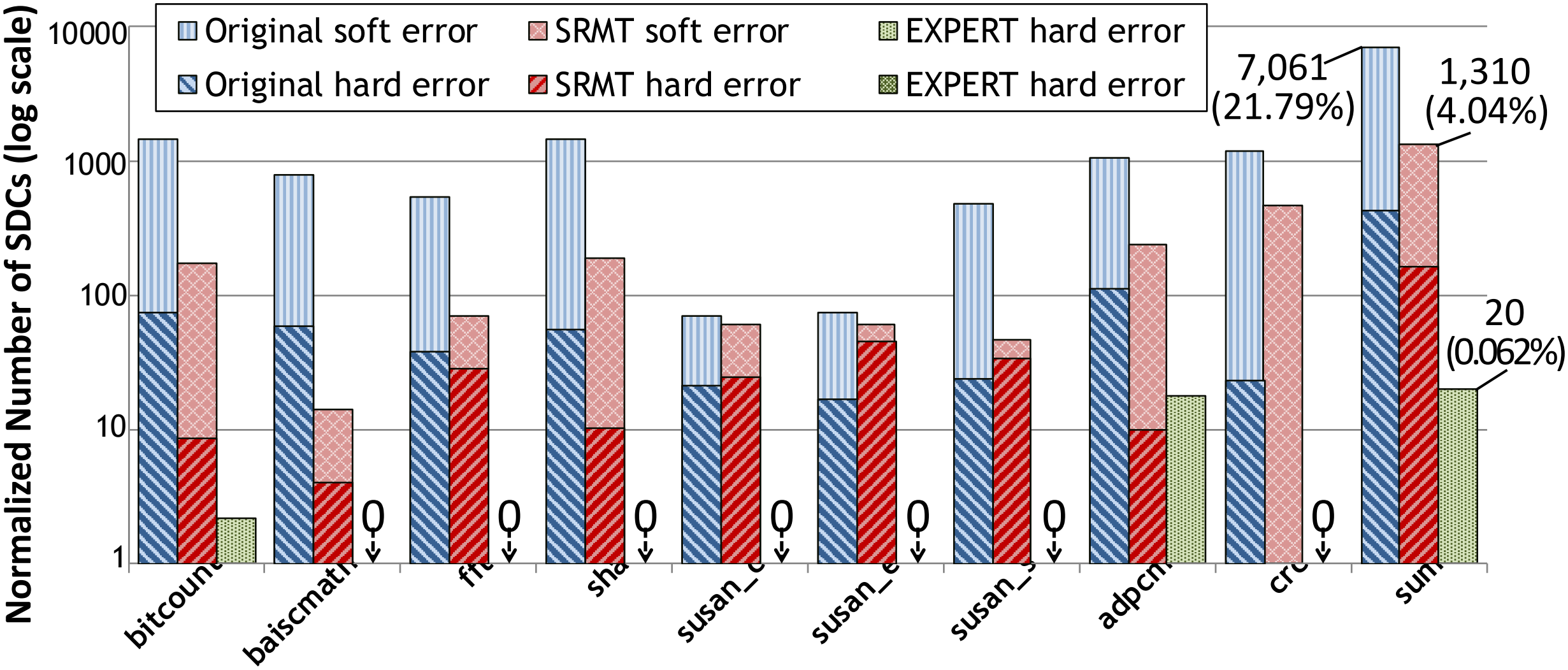
② ➡ ❷

③ ➡ ❸

Experiment: Setup

- **Benchmark: 9 applications in miBench**
  - **Original / SRMT-protected / EXPERT-protected**
- **Fault Injection on cycle-accurate gem5 simulator**
  - **6 components for fault injection**
  - **1 error injection per 1 execution**
    - **✓500 soft errors and 100 hard errors per component / benchmark**
    - **✓Total # of injections : 81,000 soft errors & 16,200 hard errors**
- **Fault coverage validation**
  - **Main target: # of silent data corruption**
  - **With correction factor[Schirmeier, DSN '15] (# of SDCs * runtime * # of cores)**

Experiment: SDC coverage validation



Conclusion

- **Improved soft and hard error detection**
  - **With load-back checking & load replication on redundant multithreading**
    - **Additional sync scheme is needed**
  - **65x better SDC coverage compared to SRMT**

- **Limitations**
  - **Runtime becomes ~5.0x on average, even with sync optimization,**
    - **SRMT: 3.9x on average**
    - **Can be improved with hardware support for communication**
  - **SDC cases on silent store**

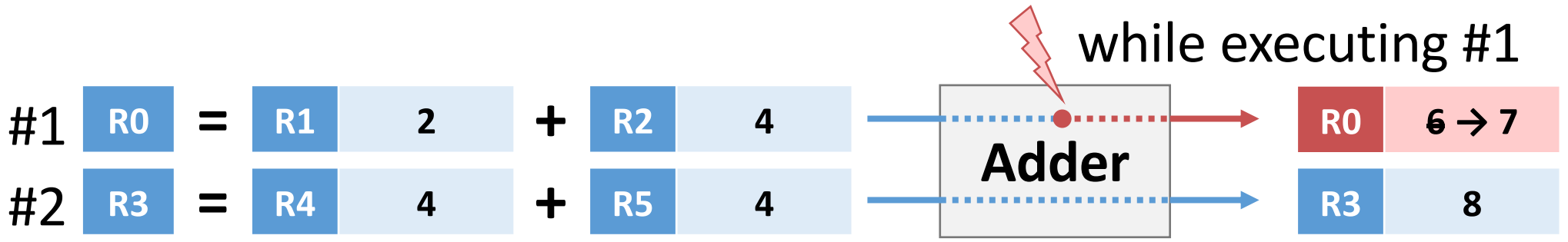
References

- [Wang, CGO '07] C. Wang et al., "Compiler-managed software-based redundant multi-threading for transient fault detection," in CGO, 2007.
- [Mitropoulou, Cases '16] K. Mitropoulou et al., "Comet: communication-optimised multithreaded error-detection technique," in CASES. ACM, 2016.
- [Zhang, IJPP '12] Y. Zhang et al., "DAFT: Decoupled Acyclic Fault Tolerance," International Journal of Parallel Programming, 2012.
- [Wadden, ISCA '14] J.Wadden et al., "Real-world design and evaluation of compilermanaged gpu redundant multithreading," in ISCA. IEEE, 2014.
- [Gupta, DAC '17] M. Gupta et al., "Compiler techniques to reduce the synchronization overhead of gpu redundant multithreading," in DAC, 2017.
- [Hukerikar, IJPP '16] S. Hukerikar et al., "Redthreads: An interface for applicationlevel fault detection/correction through adaptive redundant multithreading," IJPP, 2016.
- [Schirmeier, DSN '15]] H. Schirmeier et al., "Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors," in DSN, 2015.

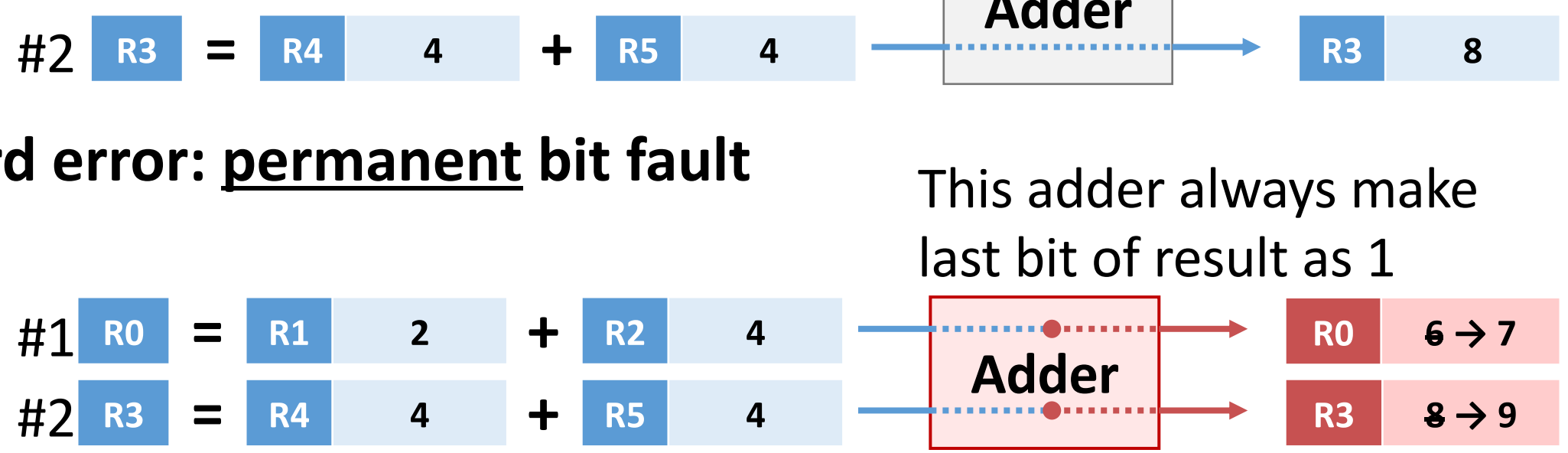
Extra slides

Soft error and hard error

- **Soft error: <u>temporal</u> bit flip**

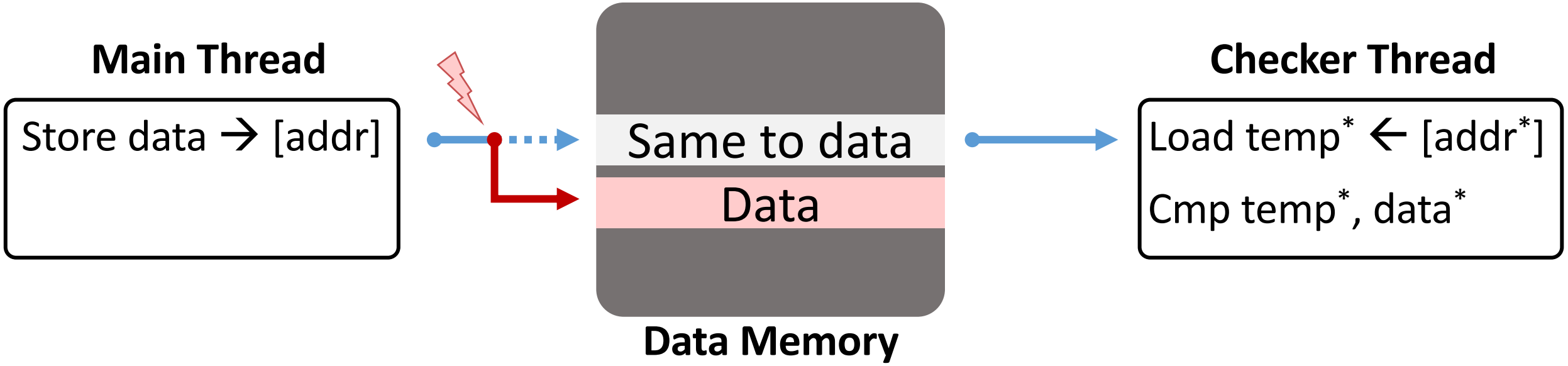


- **Hard error: <u>permanent</u> bit fault**



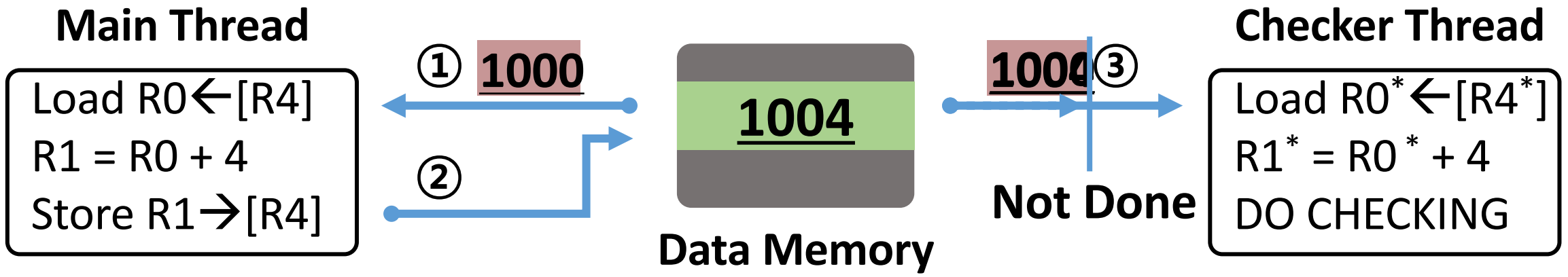
Silent Store Problem

- **Silent store: if the previous value in memory is the same as data of store, store does not change memory**
- **If address of silent store is corrupted, EXPERT can not detect memory corruption**

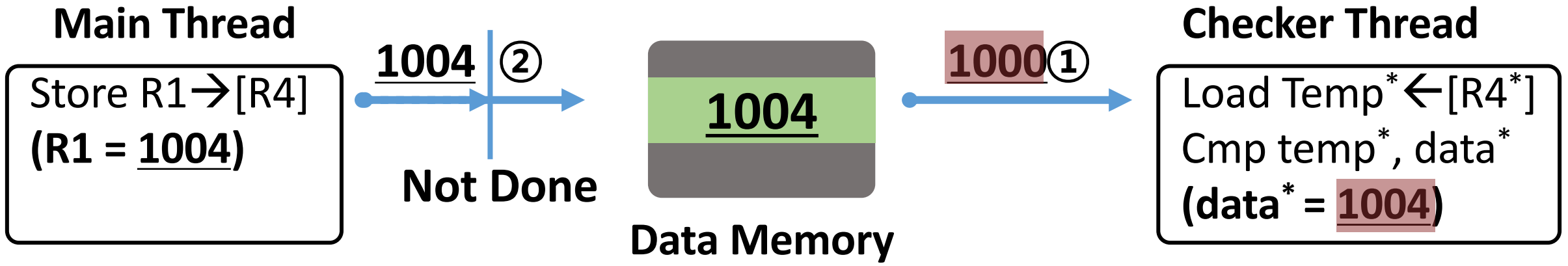


EXPERT: Memory Coherence Problem

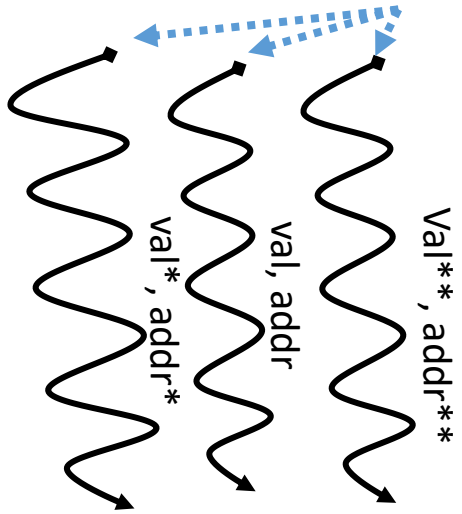
- **In LOAD and STORE with same address**



- **In STORE and relative CHECKING**



A closer look into SWIFT-R



Majority-voter(val, val*, val)**

Majority-voter(adr, adr*, adr)**

store val→[addr]

```
if ((adr != adr*) || (addr != adr **) || (adr * != adr **)){
   if (adr == adr *)              // addr ** is faulty
         adr ** = adr;
   else if (adr * == adr **)      // addr is faulty
         adr = adr *;
   else if (adr == adr **)        // addr * is faulty
         adr * = adr;
}
```

```
   movl   -4(%rbp), %eax
   cmpl   -8(%rbp), %eax
   jne    .L2
   movl   -4(%rbp), %eax
   cmpl   -12(%rbp), %eax
   jne    .L2
   movl   -8(%rbp), %eax
   cmpl   -12(%rbp), %eax
   je     .L6
.L2:
   movl   -4(%rbp), %eax
   cmpl   -8(%rbp), %eax
   jne    .L4
   movl   -4(%rbp), %eax
   movl   %eax, -12(%rbp)
   jmp    .L6
.L4:
   movl   -4(%rbp), %eax
   cmpl   -12(%rbp), %eax
   jne    .L5
   movl   -4(%rbp), %eax
   movl   %eax, -8(%rbp)
   jmp    .L6
.L5:
   movl   -8(%rbp), %eax
   cmpl   -12(%rbp), %eax
   jne    .L6
   movl   -12(%rbp), %eax
   movl   %eax, -4(%rbp)
.L6:
```

PR Tradeoffs from single thread to multi threads