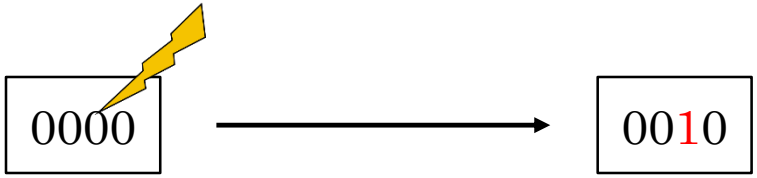
Techniques for Improving Soft error Resilience

Sparsh Mittal

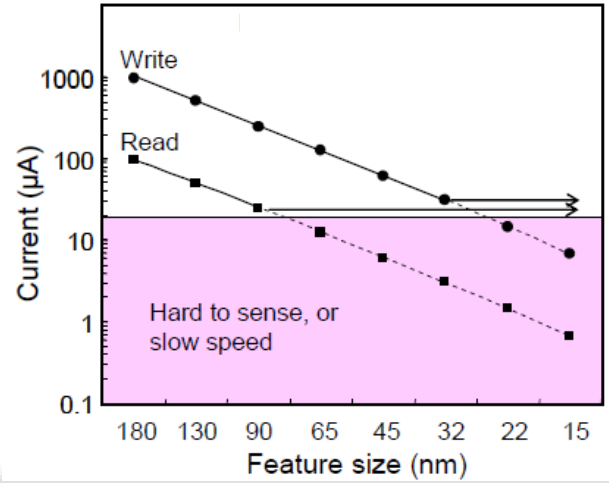
IIT Hyderabad, India

An Overview of Soft (transient) Errors

- Bit-flip in DRAM, SRAM: Particle strike can flip a value



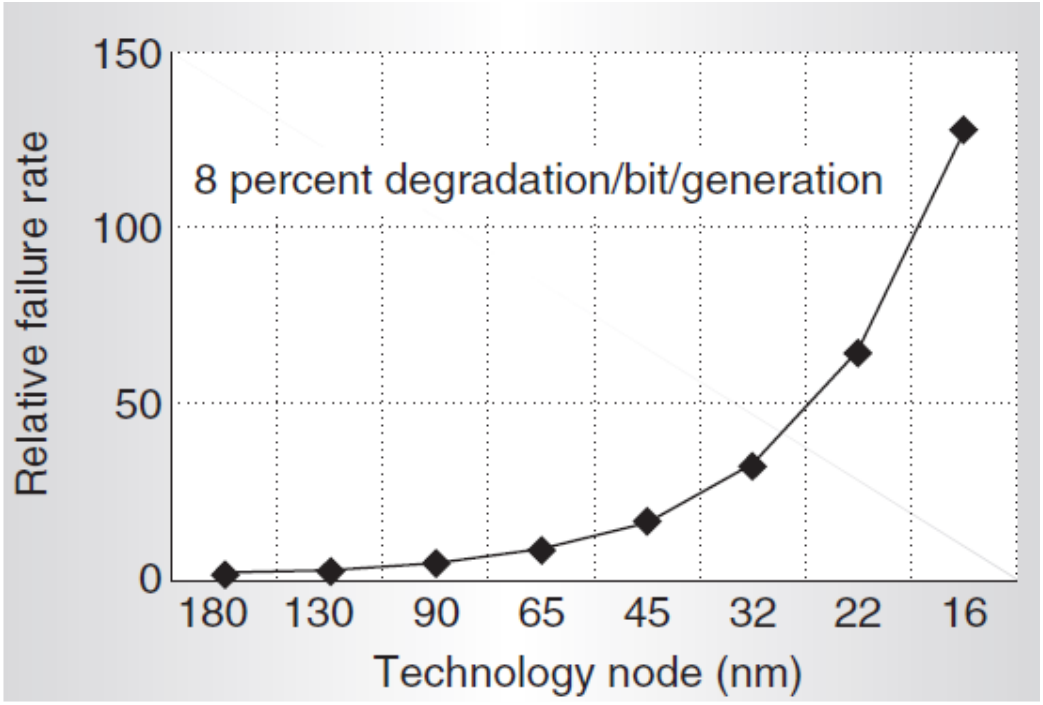
- Read disturbance in STT-RAM: Read and write currents are becoming so close that reading a cell may inadvertently modify it



Takemura et al., IMW 2010.

This PPT mainly focuses on particle-strike induced soft-errors in SRAM/DRAM

Exponentially increasing soft-error (SE) rate



Soft-error failure-in-time of a chip [Borkar Micro'05]

- Feature size scaling trends have exacerbated the occurrence and impact of faults
- Soft-error failure rate at 16nm node is expected to be 100 times that at 180nm node

Increasing impact of faults on modern processors

- With ongoing voltage scaling, the critical charge required for flipping a stored bit has been decreasing.
- Even lower-energy particles, which are more numerous, can also cause soft-errors
- The financial and social impact of faults can be huge

Reliability has now become the primary optimization target in processor design

Importance of addressing soft-errors

- Mission-critical systems require high reliability, e.g., defense, health
- Impact of altitude on soft-errors
  - Compared to sea level, the rate of neutron flux is 3.5X higher at 1.5km and 300X higher at 10-12km (typical altitude where airplanes fly).
- Voltage scaling => even lower energy particles can cause soft-errors

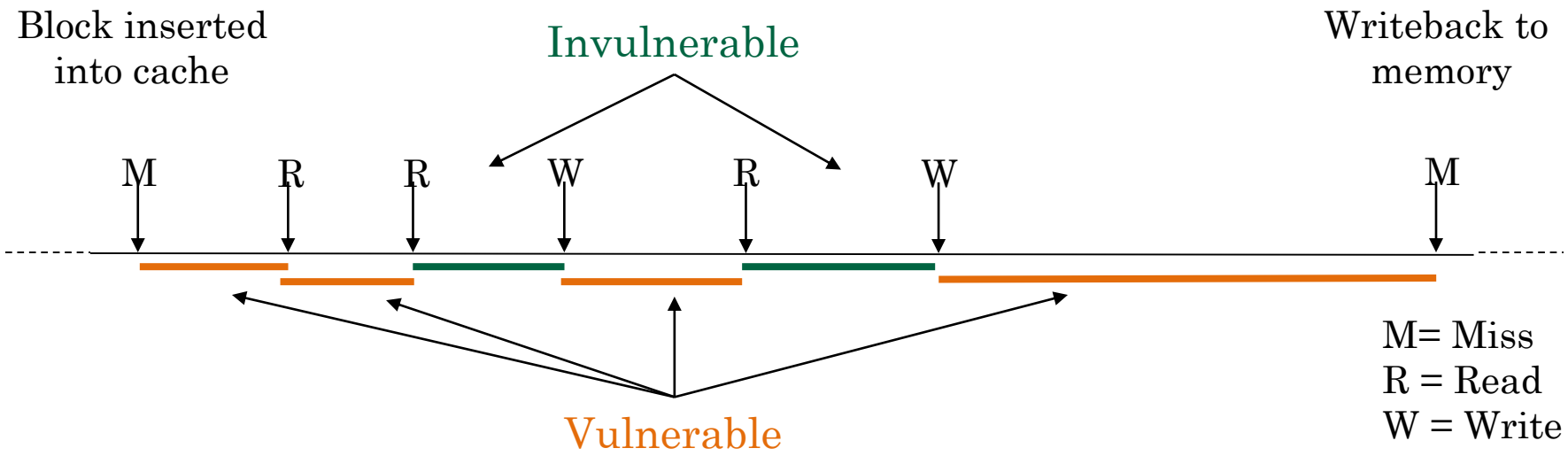
Importance of addressing soft-errors

- Errors necessitate use of ECC => latency overhead which grows quickly with increasing correction capability
- Error => an interrupt is raised => servicing it takes >1000X higher latency than cache/ memory access latency => unpredictable slowdowns.

- Errors exceed correction capability => page is retired => memory capacity degraded

Modeling (Quantifying) Soft-error Resilience

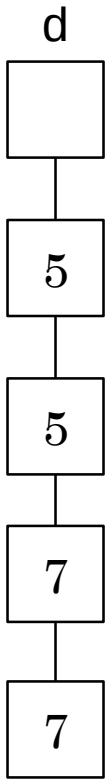
Error-masking in caches



- Key insight: not every *fault* leads to final *error*
- Data are
  - vulnerable from write to read/writeback
  - Not vulnerable from read to write, etc.

Error-masking at program level

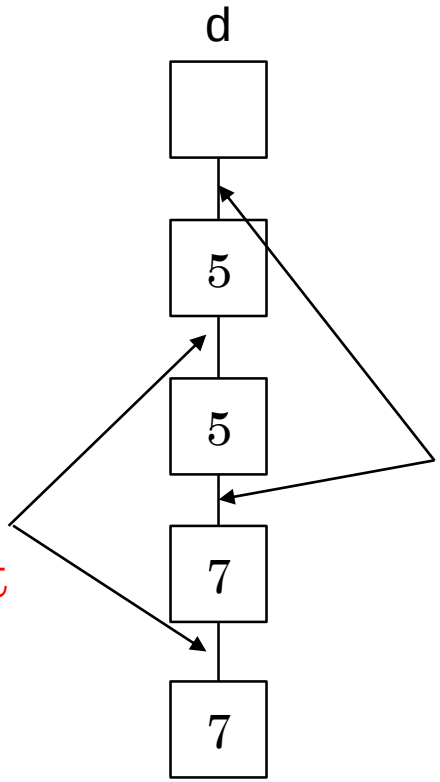
```
float d, c;

d= 5;

c = d * pi;

cin>>d;

c = d* pi;
```



Error-masking at program level

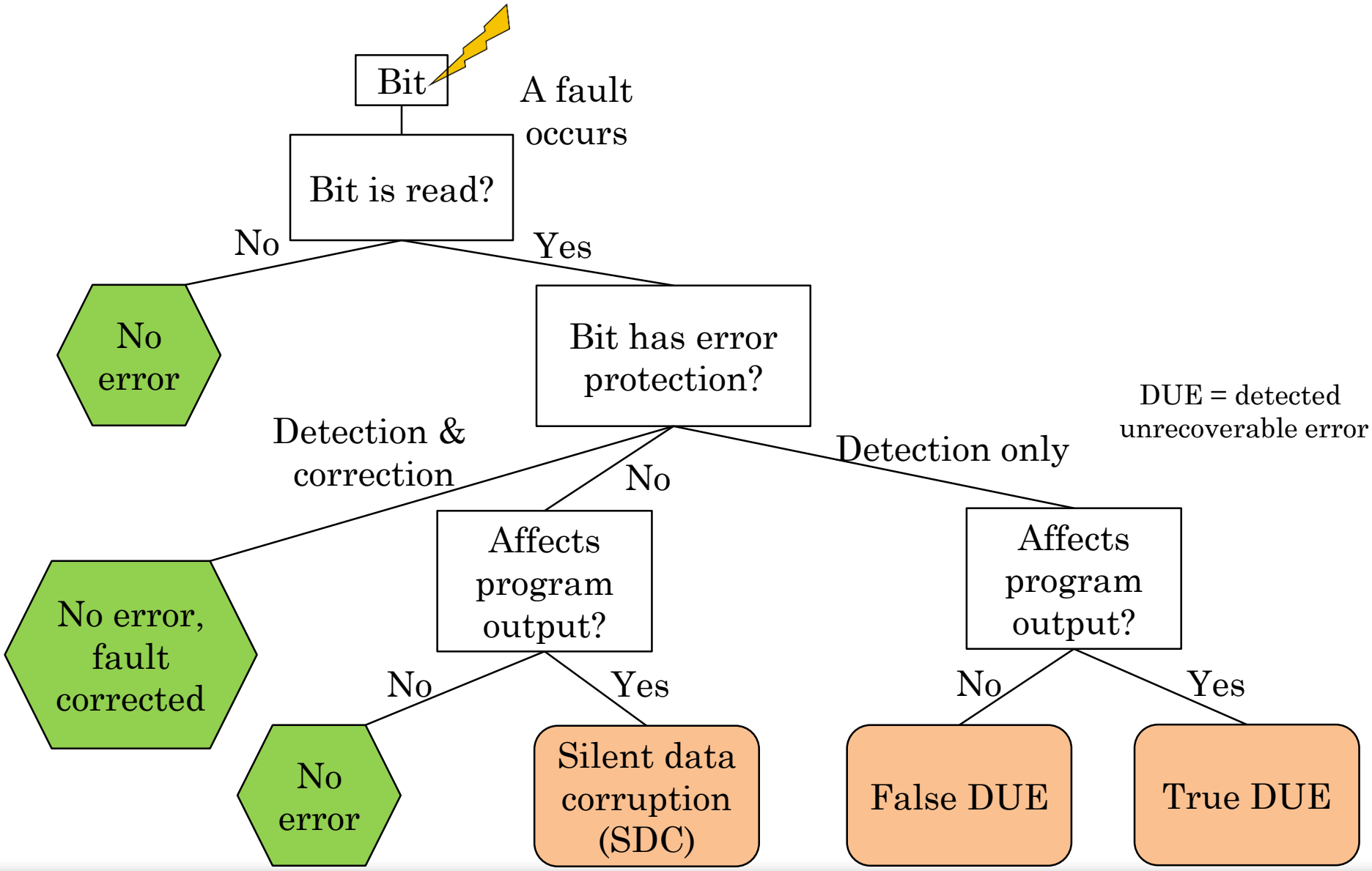
```
float d, c;

d= 5;

c = d * pi;

cin>>d;

c = d* pi;
```

A bit-flip here will affect output



A bit-flip here will not affect output

Illustration of faults leading to errors



"Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor", Weaver et al., ISCA 2004

Measuring soft-error vulnerability (SEV)

- SEV is modeled based on architectural vulnerability factor (AVF) metric [Mukherjee et al. MICRO'03]

- AVF measures the fraction of time RF is vulnerable to soft errors, i.e., when a circuit-level soft-error gets propagated to other components

$$AVF = \left(\sum_{i=1}^{M} CT_i\right) / (TT \cdot R)$$

- Here, TT = total execution time, CT_i = critical time of a critical register i, M = total # of critical registers, R = total # of registers

Measuring soft-error vulnerability (SEV)

$$SEV = R \cdot bpR \cdot AVF = \frac{bpR \cdot \sum_{i=1}^{M} CT_i}{TT}$$

- Here, bpR = # bits per register

- SEV can be defined for any processor component, e.g., cache, main memory etc.
- SEV allows avoiding under/over-protection

We use the terms ‘vulnerable’ and ‘critical’ interchangeably

Strategies for Improving Soft-error Resilience

- Clearly, SE vulnerability can be reduced by:
  1. Designing memory with radiation-hardened cells
  2. Reducing number of critical bits
     A. Exploiting data compression and narrow data
     B. Turning-off unused portion
  3. Reducing critical time
     A. Instruction rescheduling
     B. Early write-back

Data compression: opportunity

Zero Values: initialization, sparse matrices, NULL pointers

0x00000000	0x00000000	0x00000000	0x00000000	…

Repeated Values: common initial values, adjacent pixels

0x*000000***FF**	0x*000000***FF**	0x*000000***FF**	0x*000000***FF**	…

Narrow Values: small values stored in a big data type

0x*00000000*	0x*0000000***B**	0x*00000003*	0x*00000004*	…

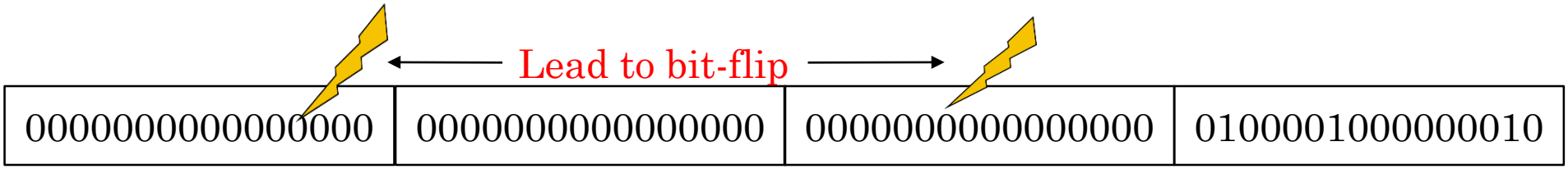
Other Patterns: pointers to the same memory region

0x*C04039***C0**	0x*C04039***C8**	0x*C04039***D0**	0x*C04039***D8**	…

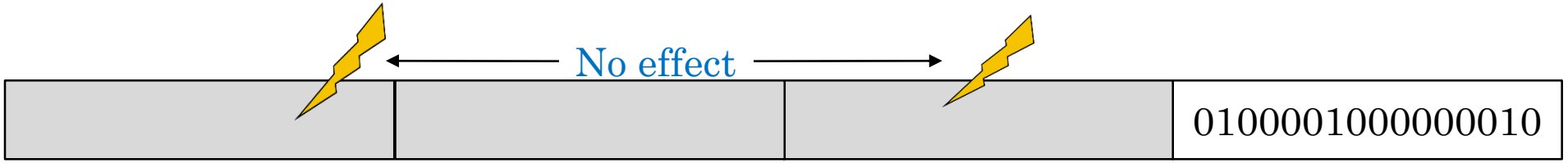
Using compression, data can be stored in smaller number of bits

Reducing number of critical bits

An example of narrow value: 16bit data in 64bit storage

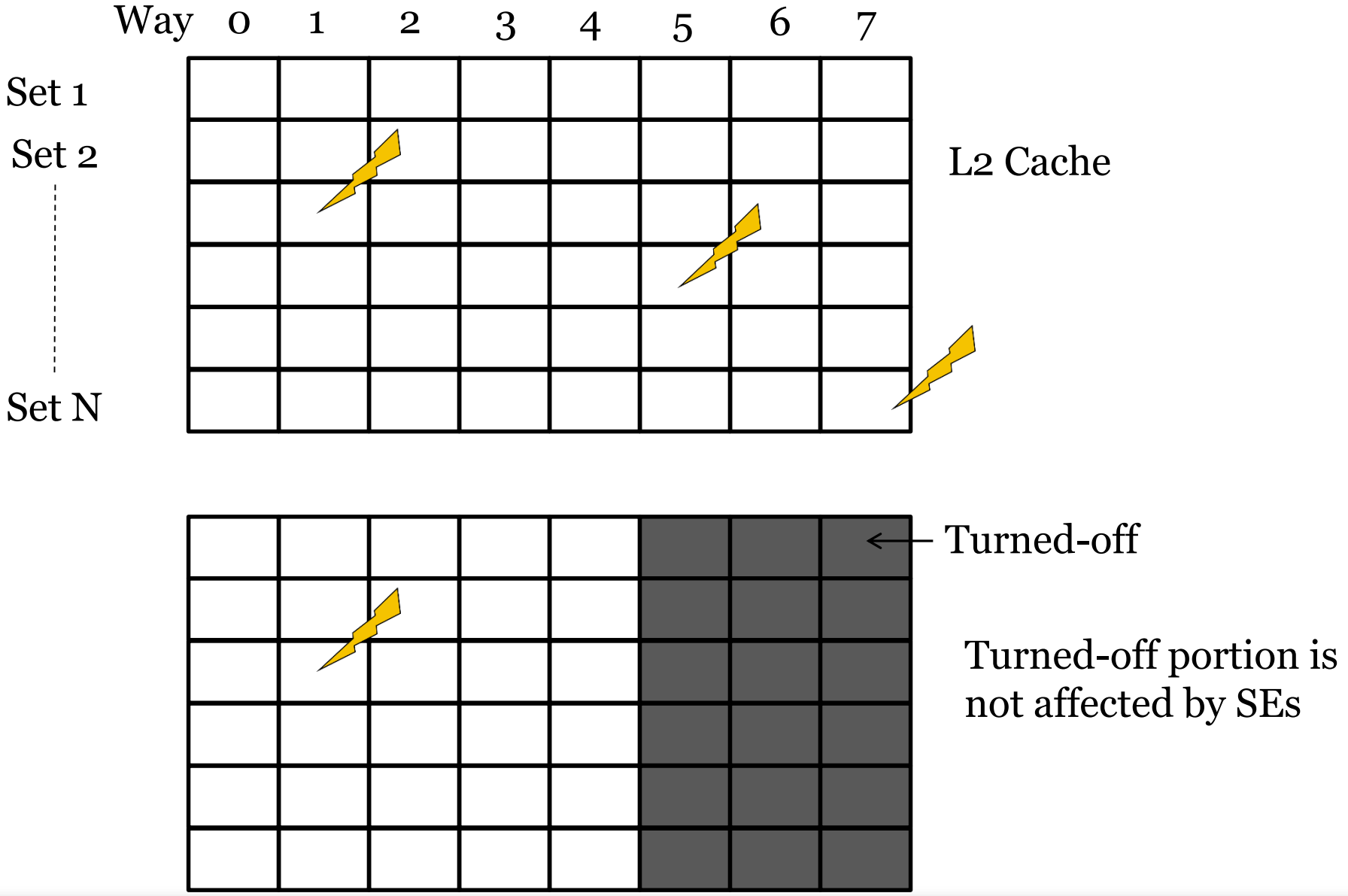


Store only non-zero lower bytes and set isNarrow bit.

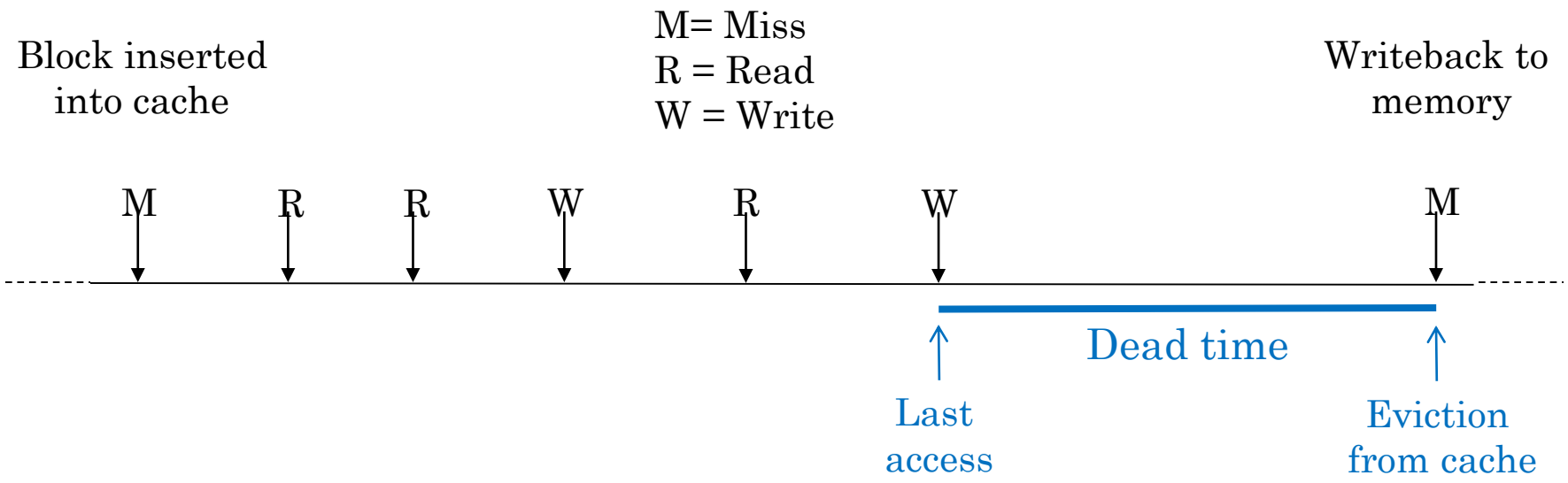


Original data can be reconstructed using isNarrow bit.

Turning-off unused portion

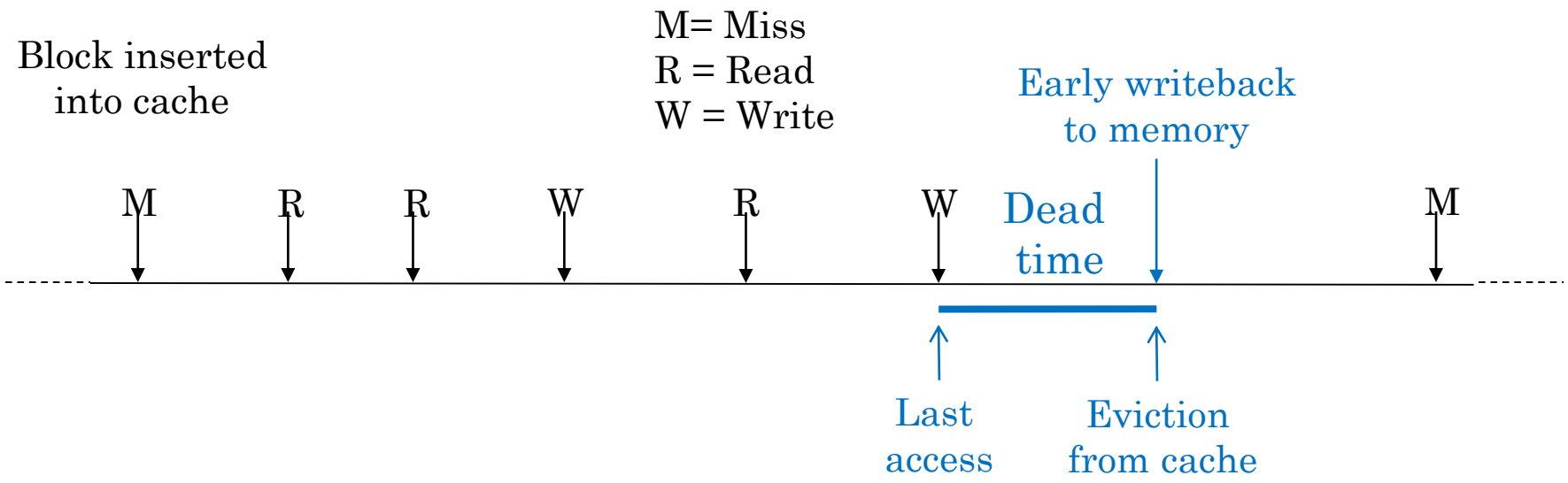


Reducing critical time: early-writeback (1 of 2)



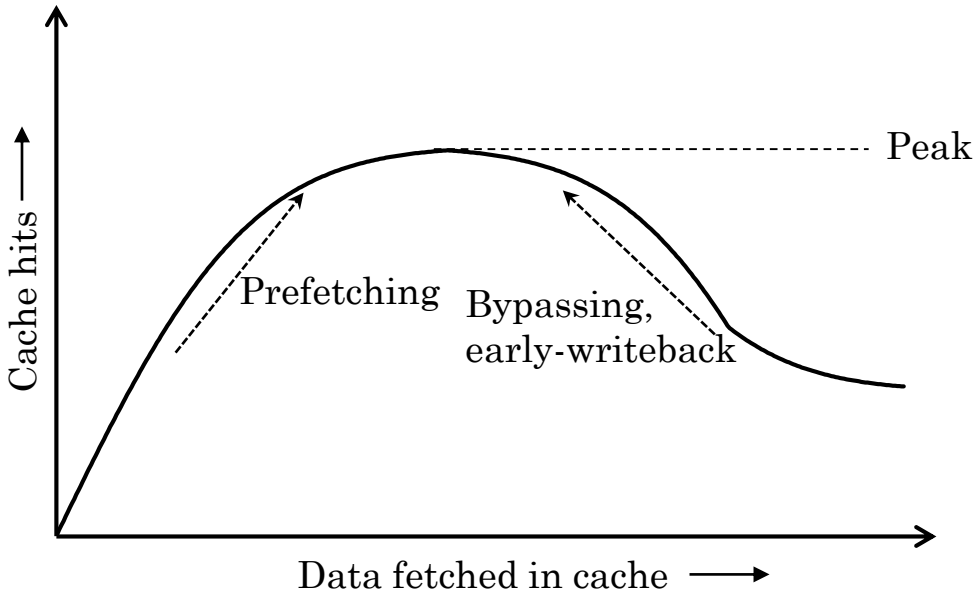
- Dead times are typically long and data remain vulnerable during this time.
- Initiate early writeback of these blocks to reduce critical time

Reducing critical time: early-writeback (2 of 2)



- Requires prediction of which is the last access of a block
- This may not be fully accurate => some performance loss

Reducing critical time: avoid/delay prefetching



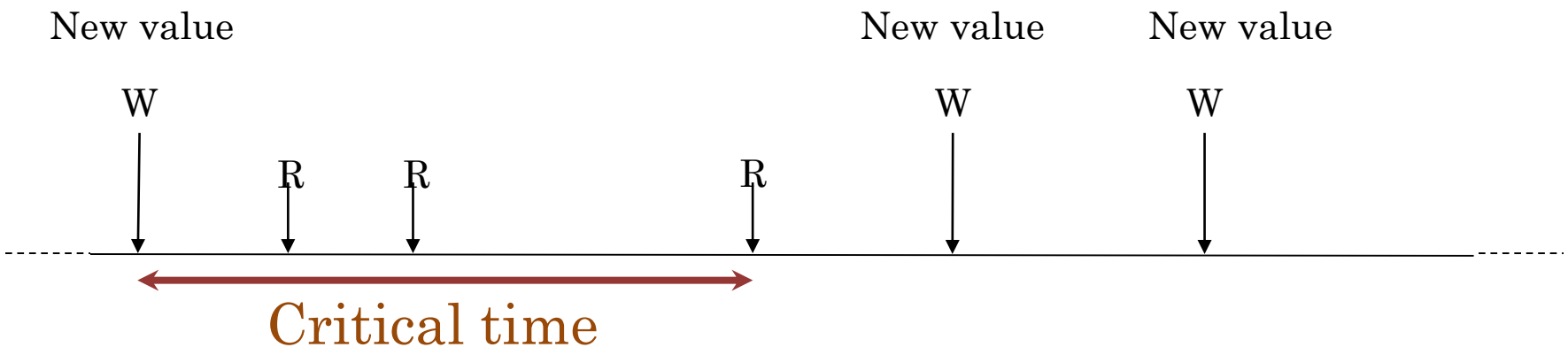
- Prefetching increases valid cache blocks
- Bypassing & early-writeback reduce them
- Need to strike a balance between performance and soft-error resilience

A question

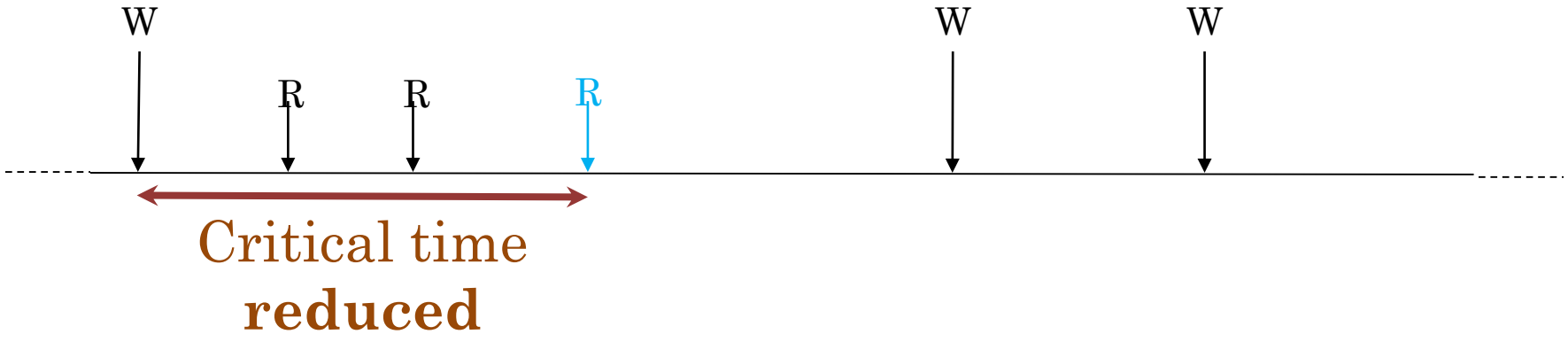
- Can we apply early write-back in RF ?

- No, because unlike caches, RF is not backed up in lower-levels of memory/cache

Reducing RF critical time: instruction rescheduling



- Schedule reads early or delay writes



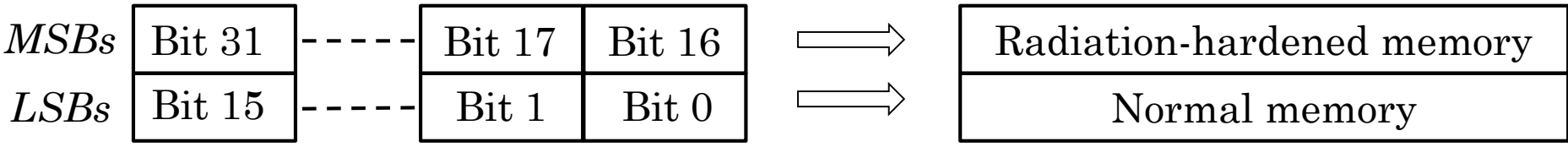
Different vulnerability of clean and dirty data

- Dirty data
  - Only correct copy in memory hierarchy, so cannot be corrected.
  - When written-back, error will also propagate to memory
- Clean data
  - Other copy available in hierarchy, so an error can be corrected by consulting
  - At replacement, clean data will be discarded, without being written-back

Dirty block is more vulnerable than clean block

Different vulnerability of MSB and LSB

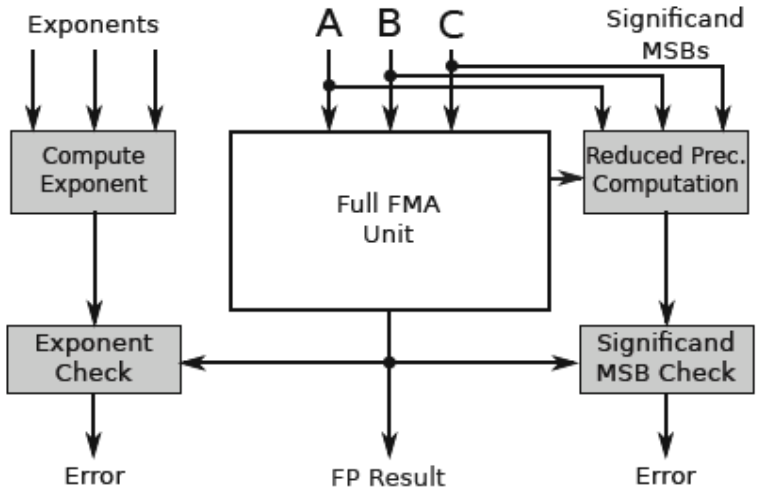
- MSB is generally more critical than LSB
- Store MSB in SE-resilient memory, LSB in normal memory



Precision-aware soft-error protection

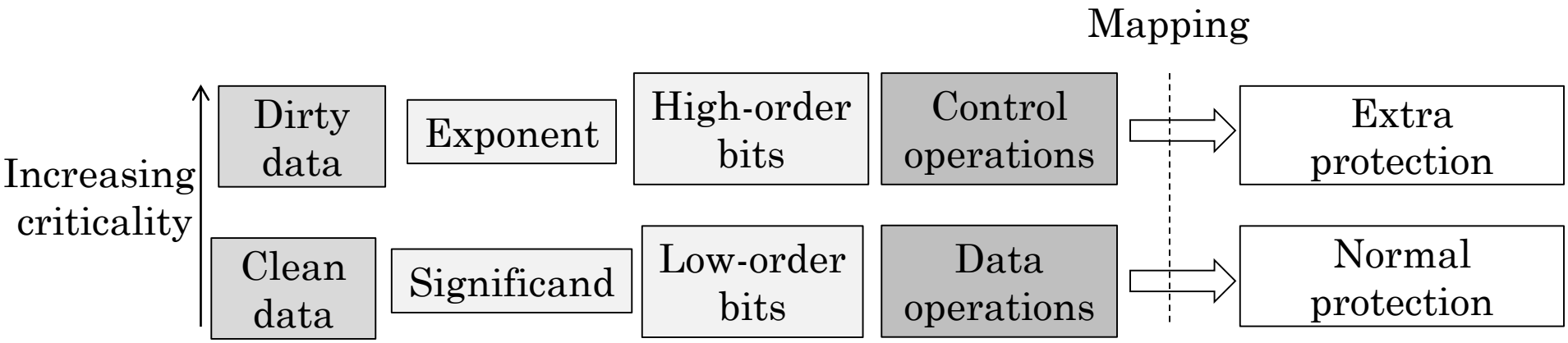
- In floating-point intensive GPU apps
  - large magnitude errors can get further amplified to significantly degrade output
  - small magnitude errors have negligible impact on output.

Verify exponent



Verify only MSB of significand

Accounting for data criticality



Examples of Extra vs. Normal Protection

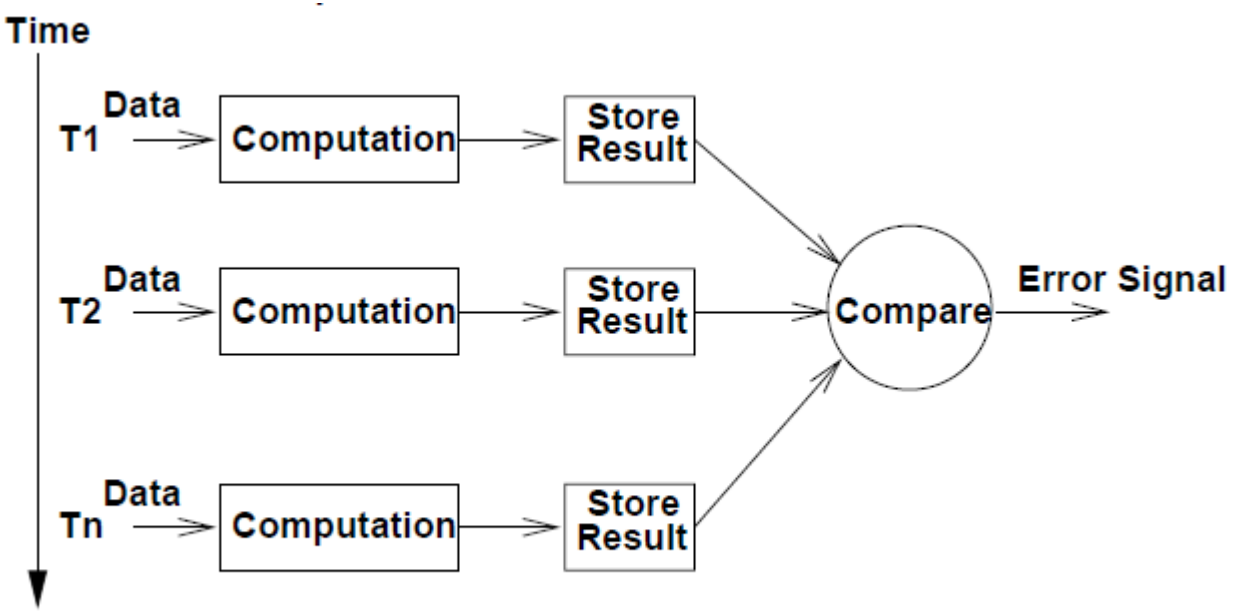
- ECC vs parity (or no protection)
- Duplication vs no-duplication
- Soft-error resilient cells vs normal cells
- Verify correctness vs do-not-verify correctness

Accounting for access frequency

- Provide extra protection to hot blocks
- Identify dead blocks and evict them to make space for duplicating hot blocks
- Copy hot ‘general purpose’ registers into idle ‘special purpose’ registers

Perform Redundant Computation: same thread, but at different time

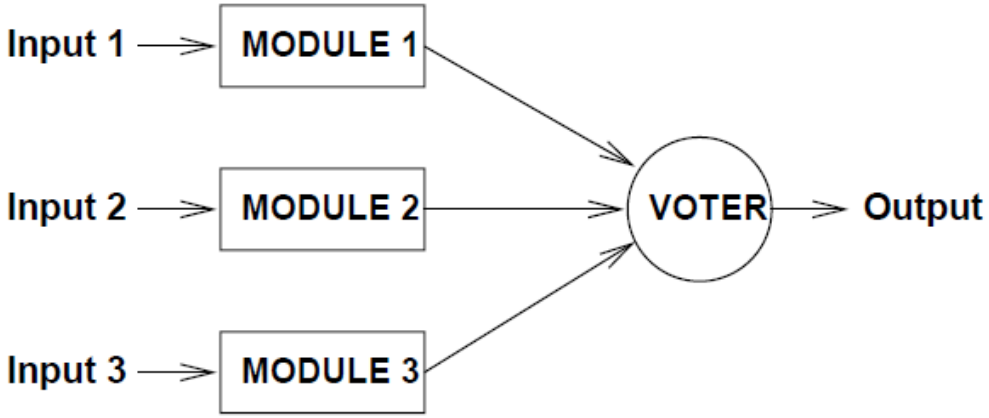
- Computations are repeated at different points in time and then compared to detect soft errors. No extra hardware is required.



28

Perform Redundant Computation: extra thread

- With main thread, use another thread(s) for performing redundant computation
- In 3D processor, a checker core can be implemented on a different die than main core



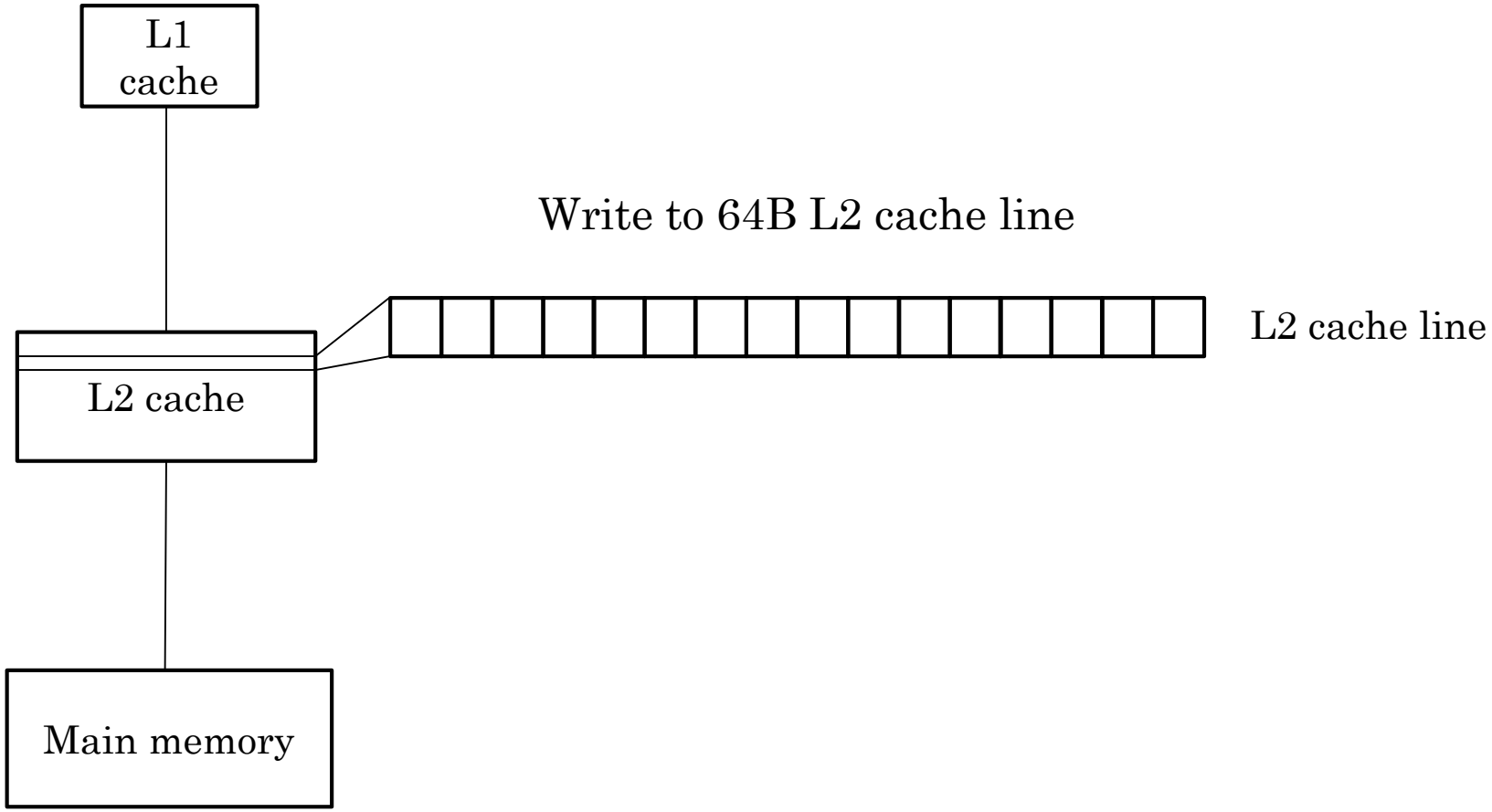
Triple modular redundancy:
Masks failure of a single component.

Using redundant storage

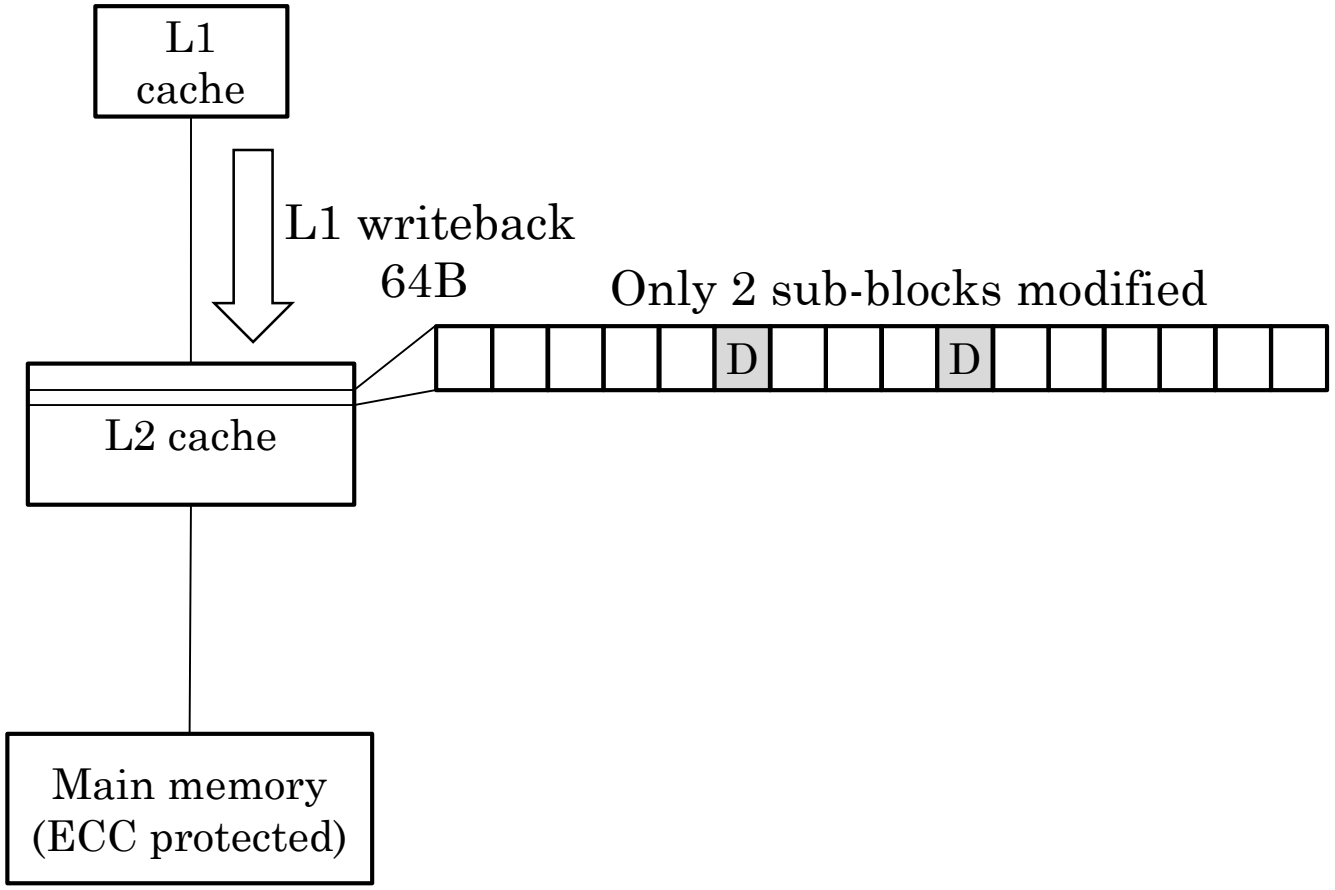
- Full duplication
  - Keep two or three copies of each block
  - Backup data from cache to ECC-protected lower level cache or memory
  - Since usage pattern of blocks is non-uniform, copy hot blocks into unused blocks
  - Use space saved from compression for duplication or storing ECC

- Parity and Error-correcting code
  - Use parity bits or special codes to detect presence of errors and correct them

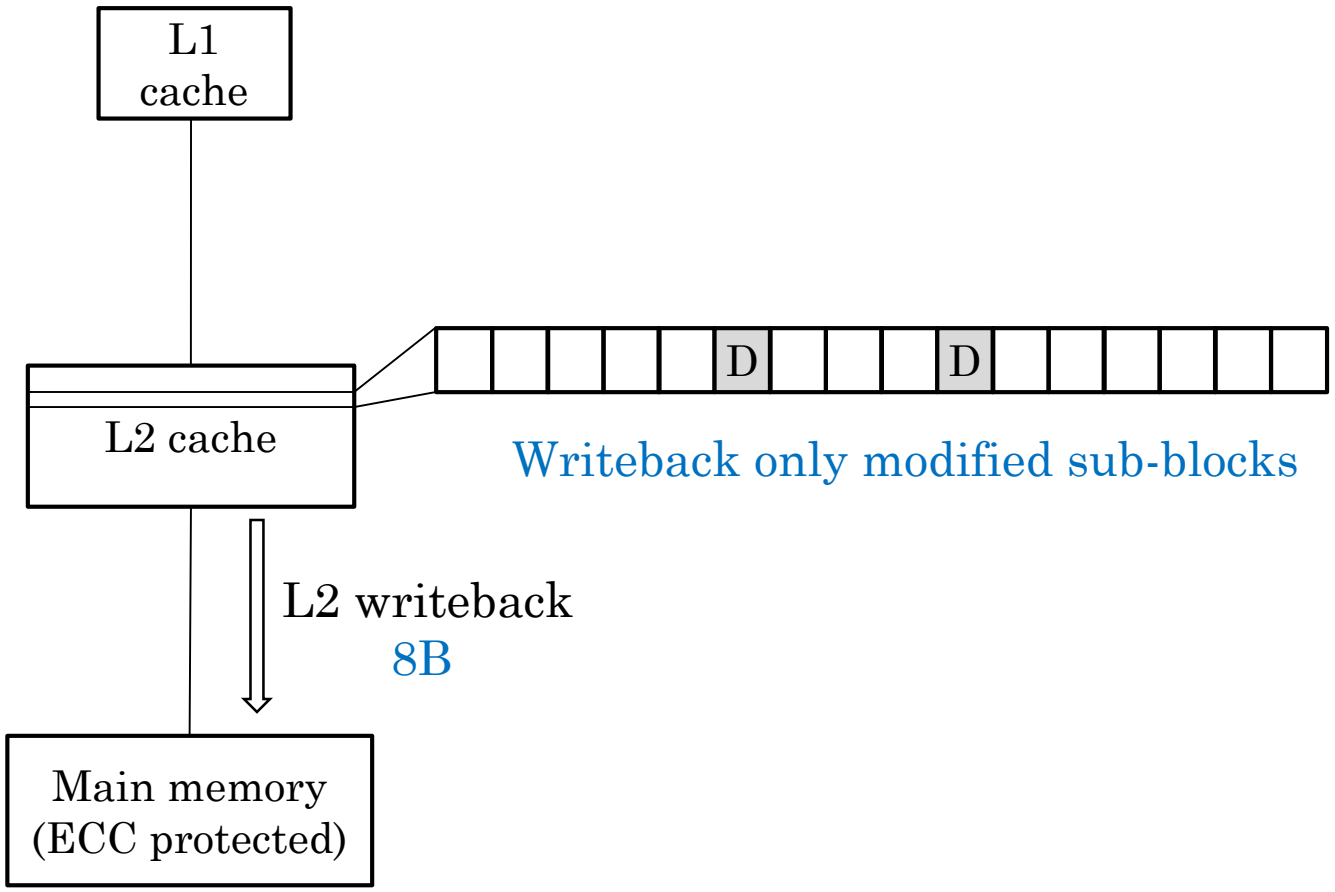
Writing-back at smaller granularity (1/3)



Writing-back at smaller granularity (2/3)

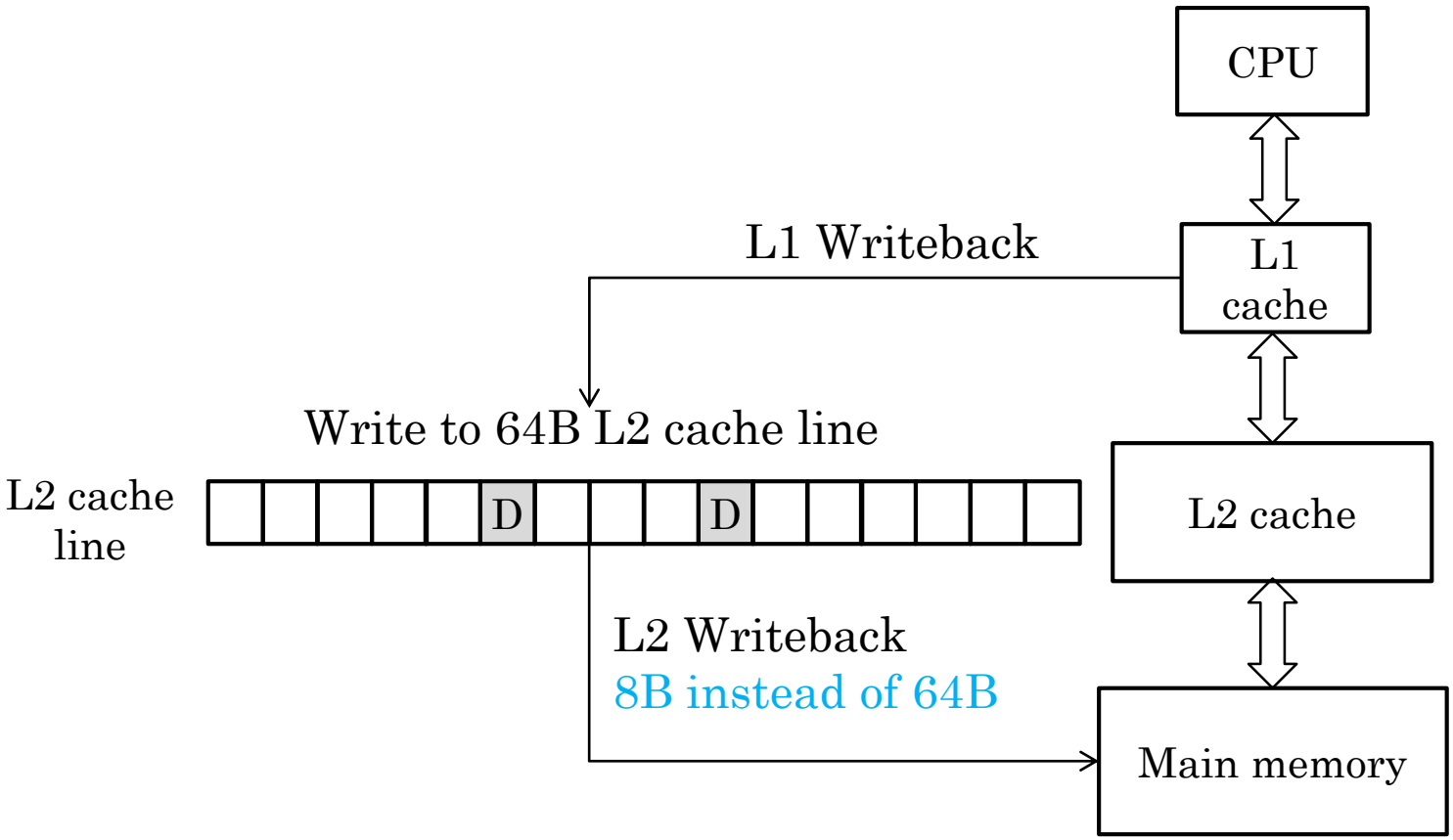


Writing-back at smaller granularity (3/3)



Writeback only modified subblocks instead of entire block.
Error in unmodified block does not propagate

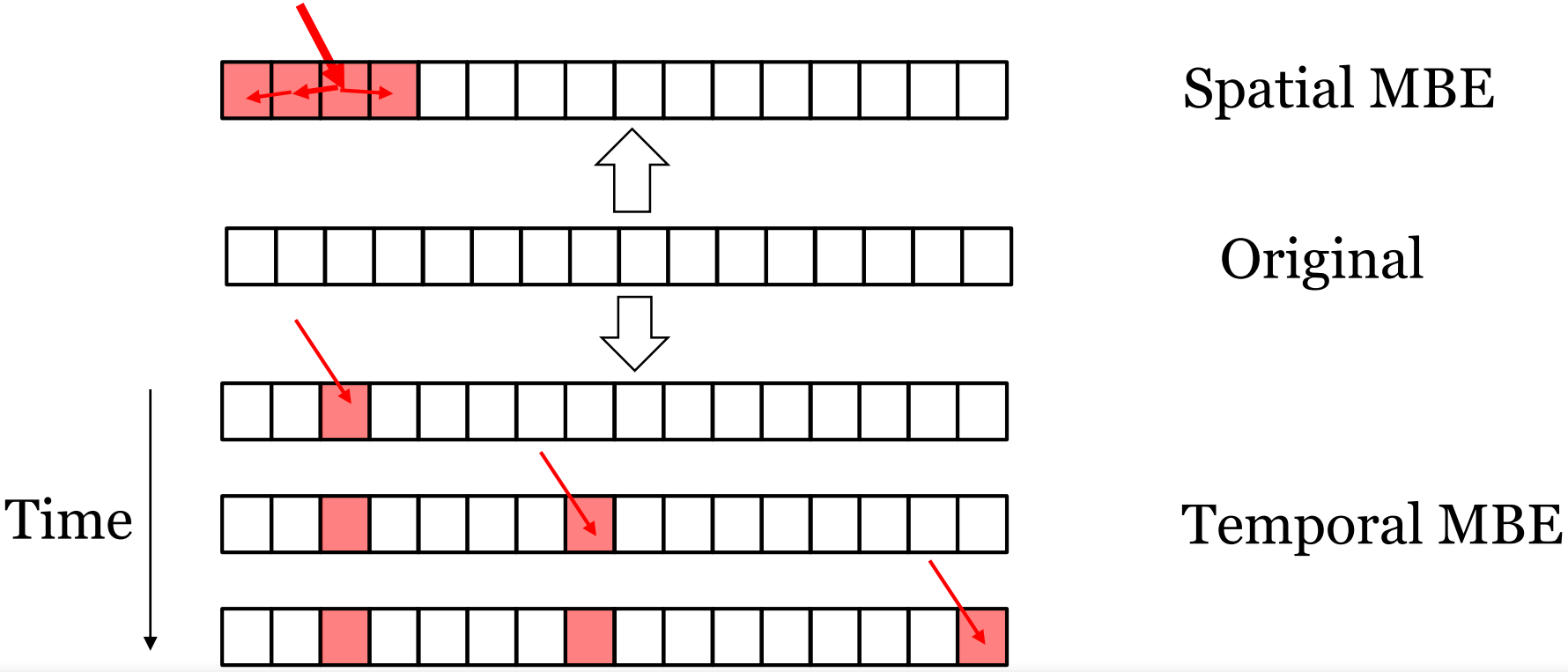
Summary of previous figures



Writeback only modified subblocks instead of entire block.
Error in unmodified block does not propagate

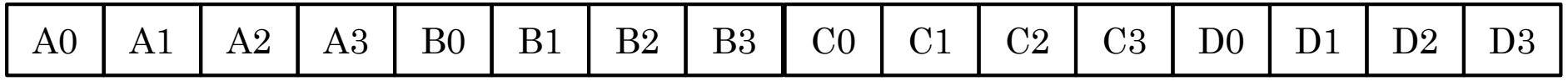
Multi-bit errors (MBEs)

- Spatial MBE: a **single particle** creating MBEs
- Temporal MBE: **multiple independent particles** striking at different times creating MBEs

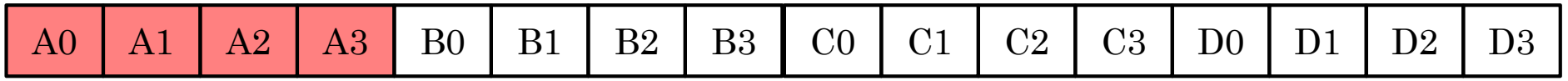


Bit-interleaving to address spatial MBEs

Baseline (no-interleaving)

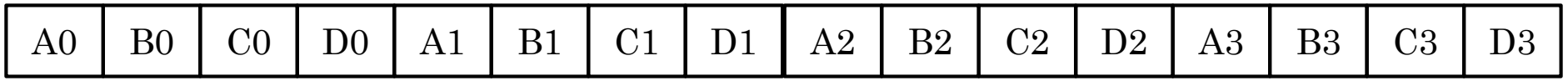


Spatial MBE occurs.

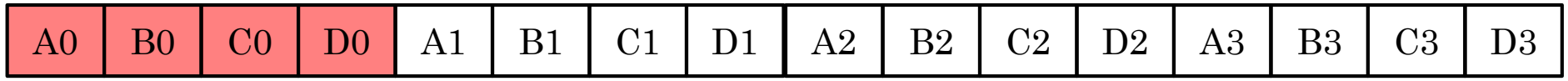


Multi-bit fault in A => Difficult to correct

4:1 Bit-interleaving



Spatial MBE occurs.



Single-bit faults in A, B, C, D => Easier to correct

Memory scrubbing to address temporal MBEs

- Read data, correct error using ECC and write to same location again

- Scrubbing helps in distinguishing b/w soft and hard error
- Scrubbing corrects errors before they exceed correction capability of ECC

References

- S. Mittal et al., “A Survey of Techniques for Modeling and Improving Reliability of Computing Systems”, IEEE TPDS, 2016 (pdf)