InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan† , Jiho Choi† , Dimitrios Skarlatos,
Adam Morrison∗, Christopher W. Fletcher, and Josep Torrellas

University of Illinois at Urbana-Champaign ∗Tel Aviv University

†Authors contributed equally to this work.

ILLINOIS

The Blavatnik School of Computer Science
The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University

Motivation: Speculative Execution Attacks

- Hardware speculative execution offers a big attack surface for covert and side channels

- Speculative execution attacks exploit the side effects of instructions on incorrect speculative paths (squashed)

Compilers and programmers can not reason about it.

- It is crucial to fix this vulnerability efficiently

Outline

- Background
- A comprehensive threat model
- **Invisible Speculation (InvisiSpec)**: the first holistic defense mechanism
- Evaluation results and current work

Speculative Execution Attacks

- An example of Spectre Variant 1 (array bound checking attack).

Victim code

```
1:  if (x < array1_size) {
2:    val = array1[x]
3:    ld array2[val]
4:  }
```

Leaves side effects in cache

Attack to read arbitrary memory:

1) Train branch predictor
2) Trigger branch misprediction
3) Side channel

Speculative execution attacks exploit side effects of instructions on paths that will be squashed

Generalization of Speculative Execution Attacks

- Transient instructions: speculatively-executed instructions that are destined to be squashed.
- Speculative execution attack exploits side effects of transient instructions.

```
1:  if (x < array1_size) {
2:    val = array1[x]
3:    ld array2[val]
4:  }
```

Transient Instructions

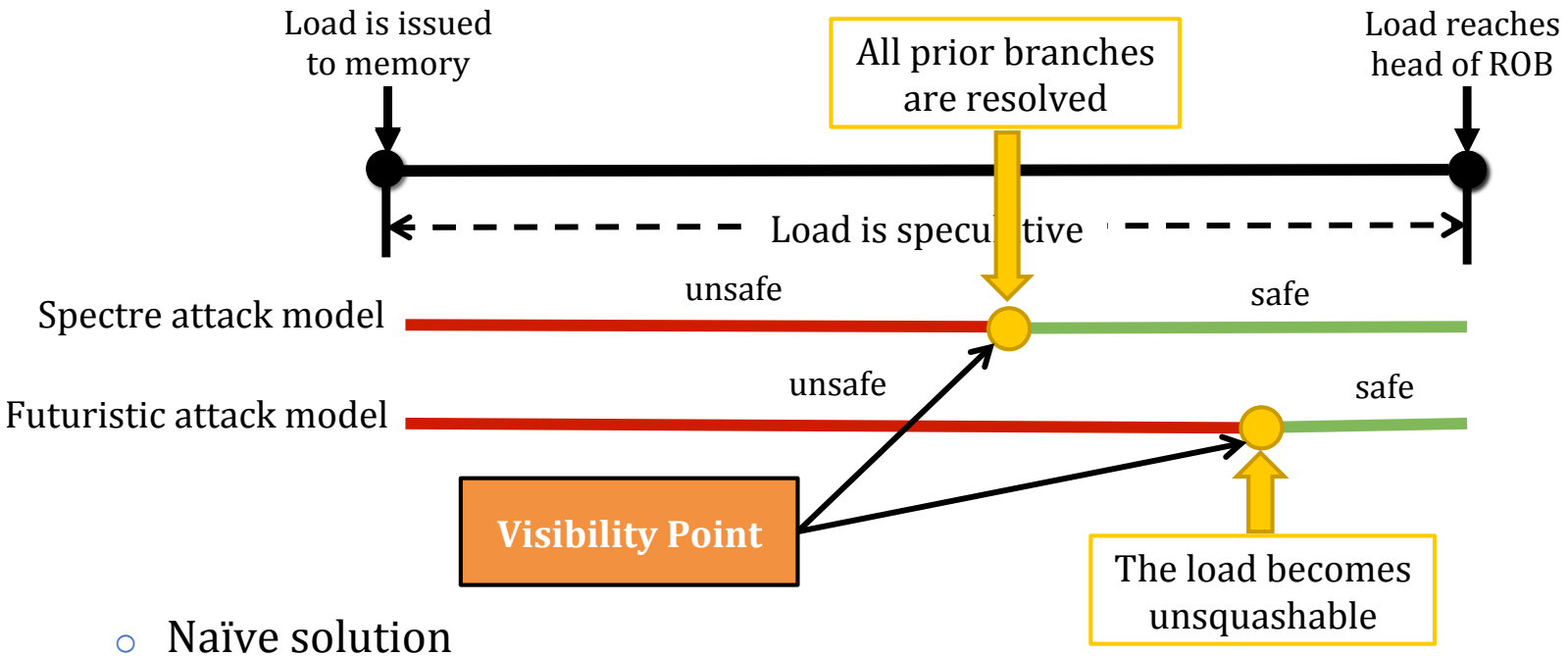
Attack	Sources of Transient Instructions
Spectre	Control-flow misprediction
Meltdown	Virtual memory exception
L1 Terminal Fault	
Speculative Store Bypass	Address alias between a load and an earlier store

Futuristic Speculative Attack Model

- Futuristic speculative attack model
  - An attacker can exploit ***any*** speculative load (load not at the head of ROB).
  - It includes all existing attacks and future speculative execution attacks

Attack Model	Sources of Transient Instructions
Futuristic	Various events, such as: • Exceptions • Control-flow mispredictions • Address alias between a load and an earlier store • Address alias between two loads • Memory consistency model violations • Interrupts

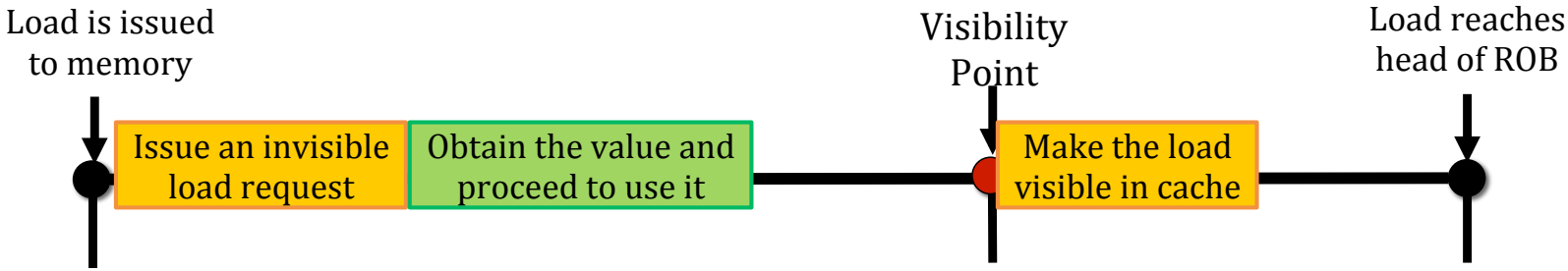
Lifetime of a load instruction



- Naïve solution
  - Delay issuing the load until its visibility point

InvisiSpec

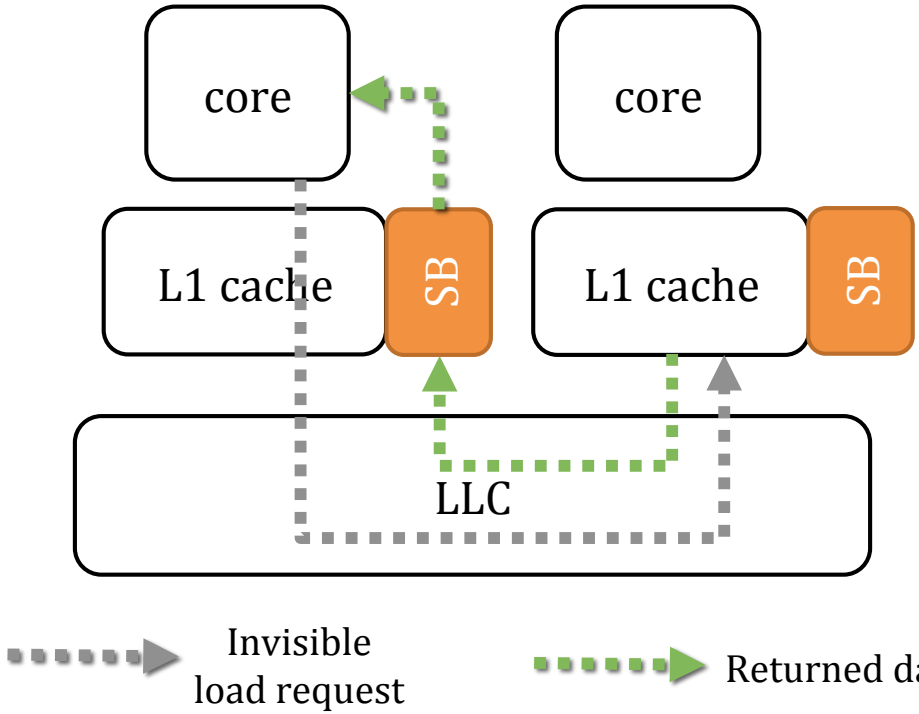
- The first holistic defense mechanism against speculative execution attacks
- Key idea:
  - Make unsafe loads invisible in the cache hierarchy



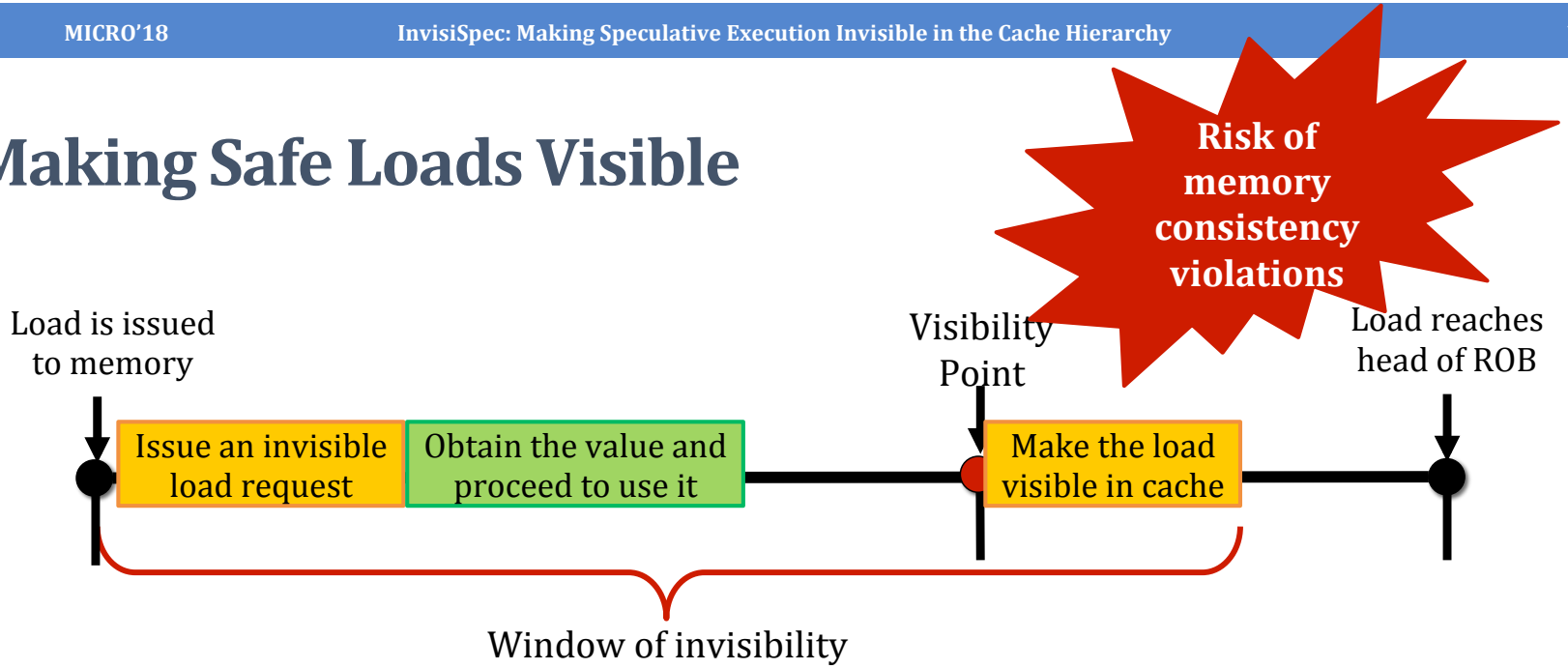
Speculative loads are issued as early as in a conventional machine

Making Unsafe Loads Invisible

- Invisible load request
  - No modification to cache states, including
    - Cache occupancy
    - Replacement information
    - Coherence information
    - TLB state
  - Bring the data, store in Speculative Buffer (SB) and in register



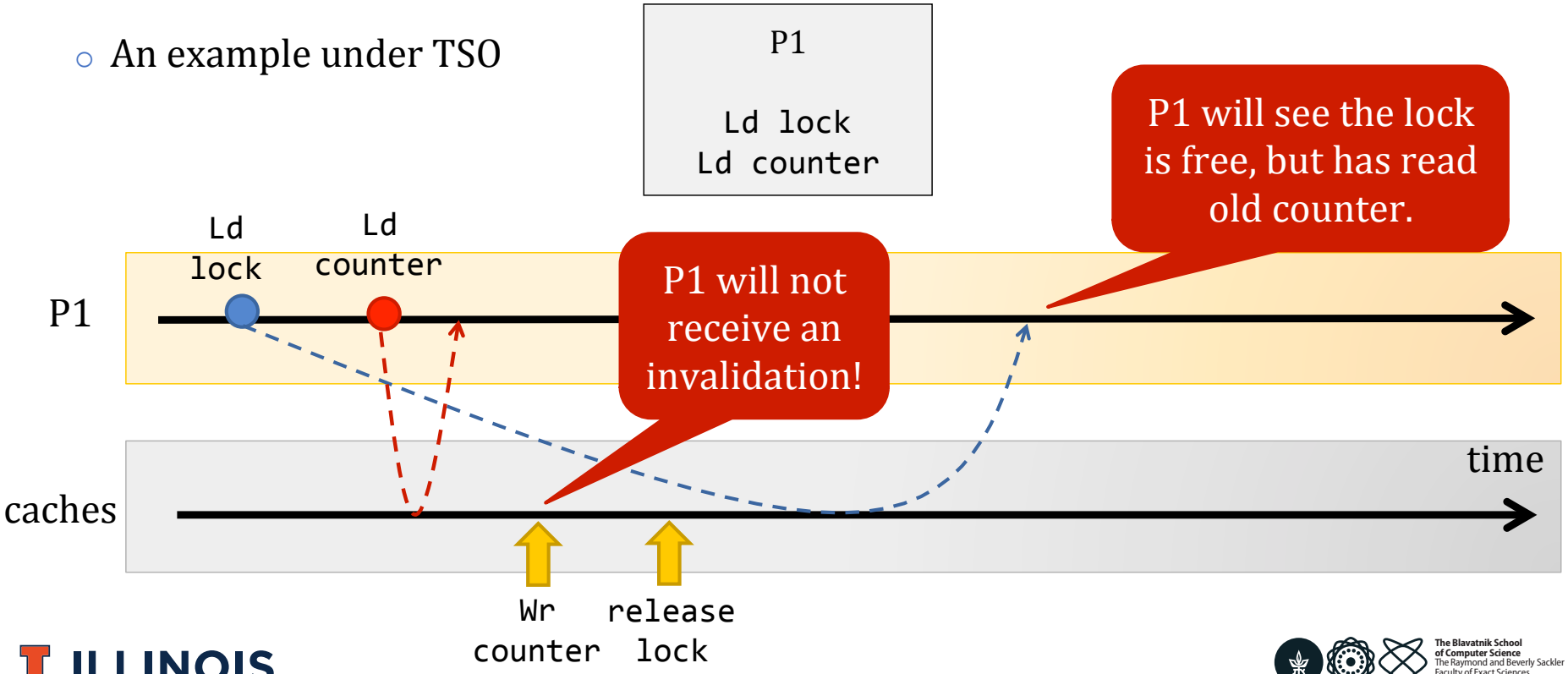
Making Safe Loads Visible



- Make the load visible: HW issues a normal request (which changes the caches)
- While in the window of invisibility, processor does not receive invalidations

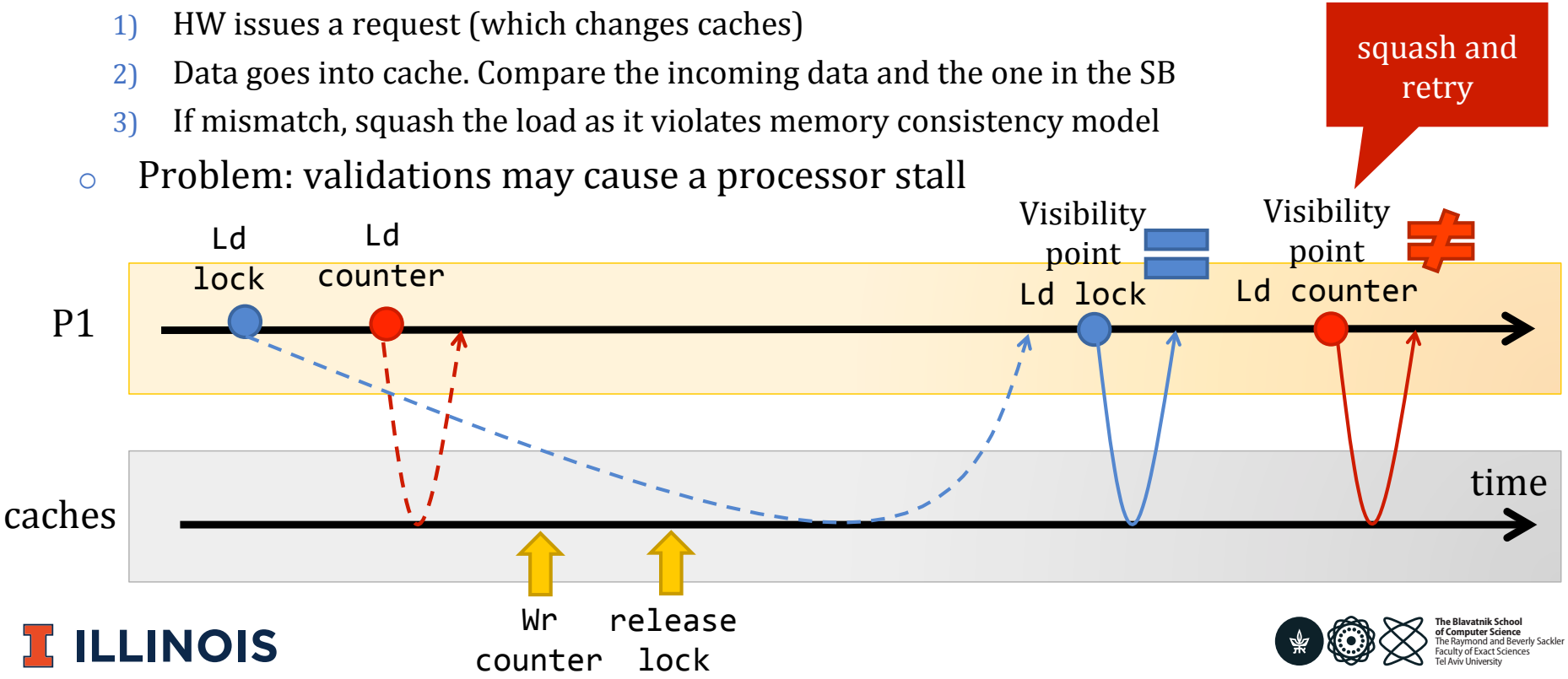
An Example of Memory Consistency Violation

- An example under TSO



Maintaining Memory Consistency

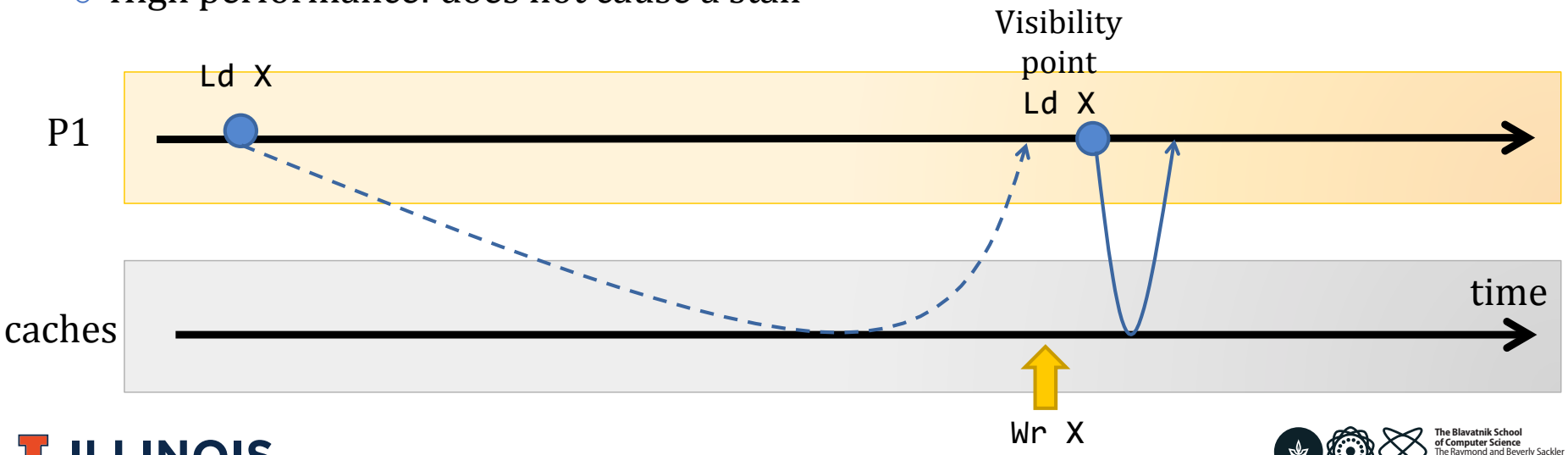
- Need to issue another access at Visibility Point: Validation
  1) HW issues a request (which changes caches)
  2) Data goes into cache. Compare the incoming data and the one in the SB
  3) If mismatch, squash the load as it violates memory consistency model
- Problem: validations may cause a processor stall



Maintaining Memory Consistency (II)

- For loads with no risk of memory consistency violation: Exposure
  - HW issues a request at the Visibility Point
  - Load does not need to wait for response to retire
  - Data goes into cache. No need to compare data
- High performance: does not cause a stall

See the paper for the many cases where a load can use exposures



Pros & Cons of InvisiSpec

- Security
  - Successfully prevent attacks in both Spectre and Futuristic attack models
- High performance
  - Speculative loads are issued as early as in a conventional machine
- Applicability
  - Handle multi-threaded issues
- No software changes

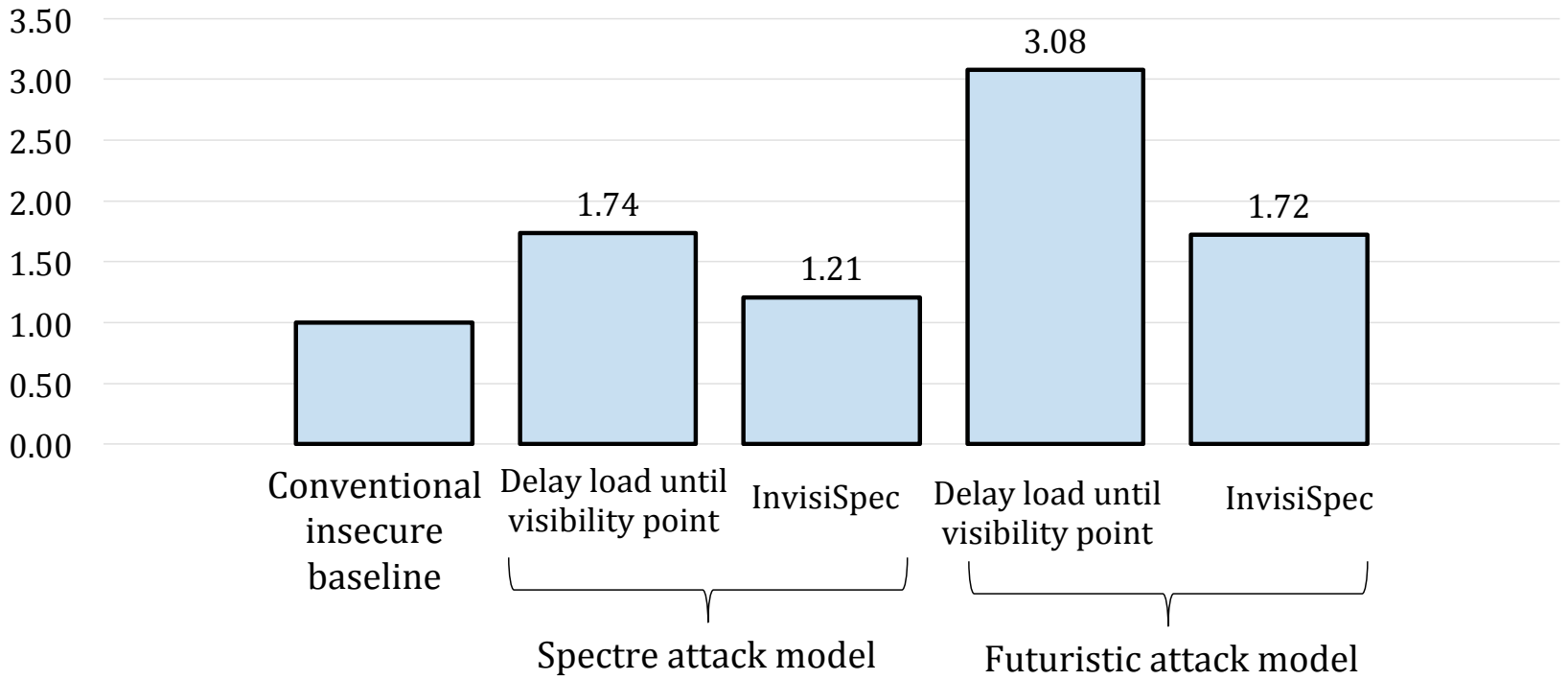
- Performance overhead
  - Double accesses
  - May stall due to validations

BUT:
- Many can be converted to exposures
- Most hit in L1, and return very quickly

ILLINOIS

The Blavatnik School of Computer Science
The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University

Average Execution Time for SPEC and PARSEC (Normalized)



ILLINOIS

The Blavatnik School of Computer Science
The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University

Current Work

- Reduce performance overhead: selectively enable InvisiSpec

More in the paper

- Security analysis
- Details of when to use validations and exposures, and overlapping of them
- Details of implementation
- Detailed performance and area overhead evaluation results

Conclusion

- InvisiSpec is the first comprehensive defense mechanism against speculative execution attacks in the cache hierarchy
- We published the code of our architecture simulator:

https://github.com/mjyan0720/InvisiSpec-1.0

ILLINOIS

The Blavatnik School of Computer Science
The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University