Learning from Mistakes A Comprehensive Study on Real World Concurrency Bug Characteristics

Shan Lu, Soyeon Park, Eunsoo Seo and Yuanyuan Zhou

Presented By Vehbi Esref Bayraktar
(aka Benjamin Turk)

Introduction

- Multicore hardware has made concurrent programs (CP) pervasive.
- Not only for high-end servers but desktop machines
  - The difficulty of hitting the entire SW environment.
  - Requirement of good quality.

Introduction

- Roadblocks to a correct CP
  - Most of programmers think sequentially
  - Non-determinism makes bugs hard to repeat.
- Addressing those roadblocks will require different efforts from multiple aspects

Efforts

- **Concurrency bug detection**
  - **Deadlock bugs:**
    - Two or more operations waiting for each other to release the acquired resources.
  - **Non-deadlock bugs:**
    - Data-race bugs (Order violation)
      - Multiple confliction accesses to one shared variable
    - Atomicity violation bugs
      - Some operations needed to be done without interruption.
    - Other bugs

Efforts

- CP testing and model checking
  - Exposing SW bugs before release
- Model checking refers to test automatically whether the model meets a given specification.
  - Exponential interleaving space for CPs to record in most of **current testing approaches**
  - Even testing a small representative interleaving may still expose most of the concurrency bugs
    - **ConTest**, Application of Synchronization coverage, PPOPP,2005
  - The manifestation conditions of concurrency bugs

Efforts

- **Concurrent programming language design**
  - Good programming languages help programmers correctly express their intentions
    - Transactional memory (one approach to it)
      - Atomicity
      - Consistency
      - Isolation

```
def transfer_money(from_account,
to_account, amount):
    with transaction():
        from_account -= amount
        to_account += amount
```

Outline

- Introduction
- Case Study
  - Bug pattern study
  - Bug manifestation study
  - Bug fix strategies study
- Conclusions

CP Characteristics

- Bug Pattern
  - **Deadlock bugs**
  - **Non-deadlock bugs**
    - Order-violation
      - Data race
      - Happened-before property
    - Atomicity-violation
    - Others

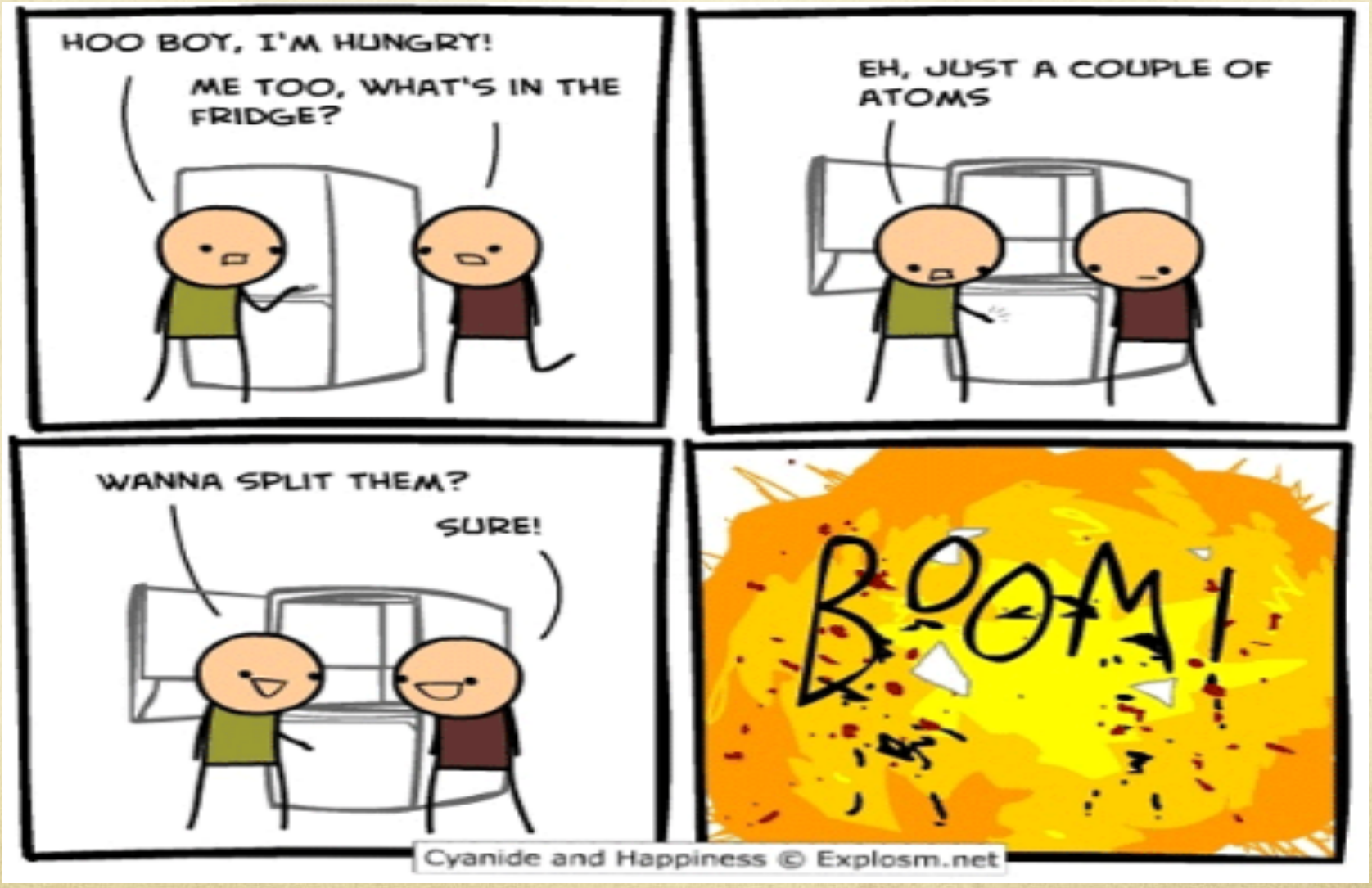
CP Characteristics

- Bug Manifestation
  - Manifestation conditions
- Bug fix strategy
  - Patches
  - Transactional memory (TM)

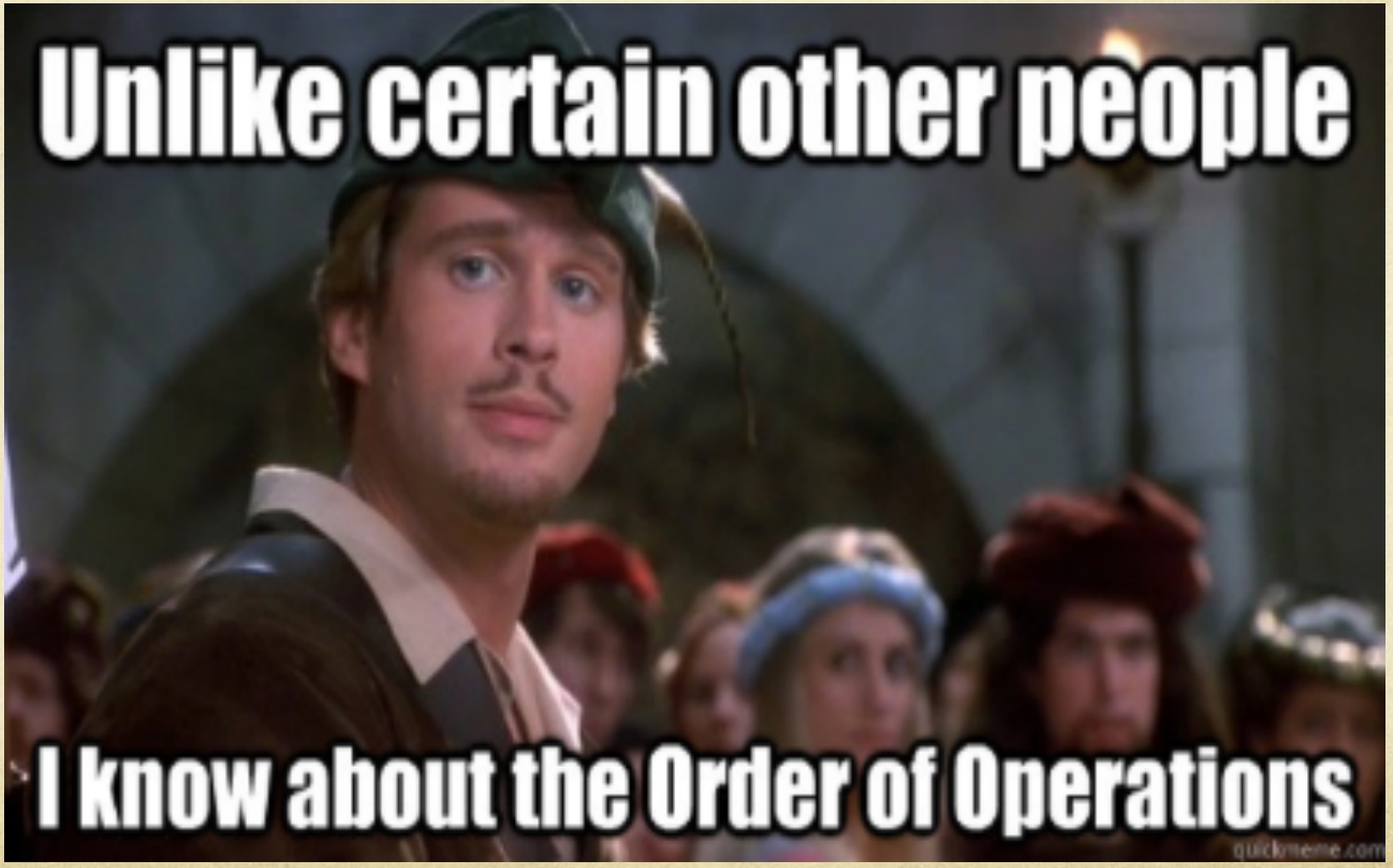
Sources

- MySQL
  - Database server
- Apache
  - Web server
- Mozilla
  - Browser suite
- Open Office
  - Office suite

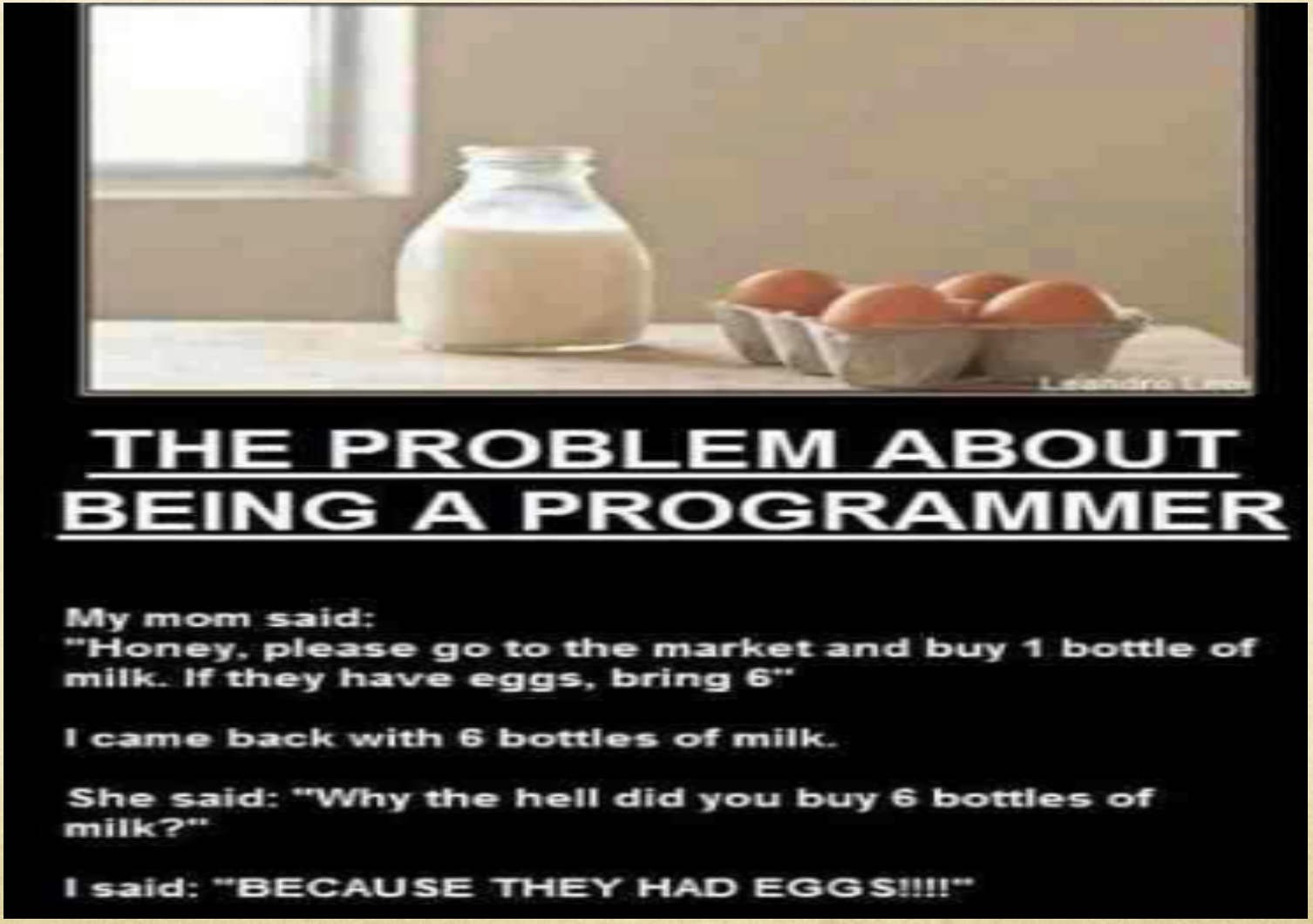
Bug Pattern Study



Bug Pattern Study



Bug Pattern Study



Bug Pattern Study

Finding (2): A significant number (24 out of 74) of the examined non-deadlock concurrency bugs are order bugs, which are *not* addressed by previous bug detection work.

Implications: New bug detection techniques are desired to address order bugs.

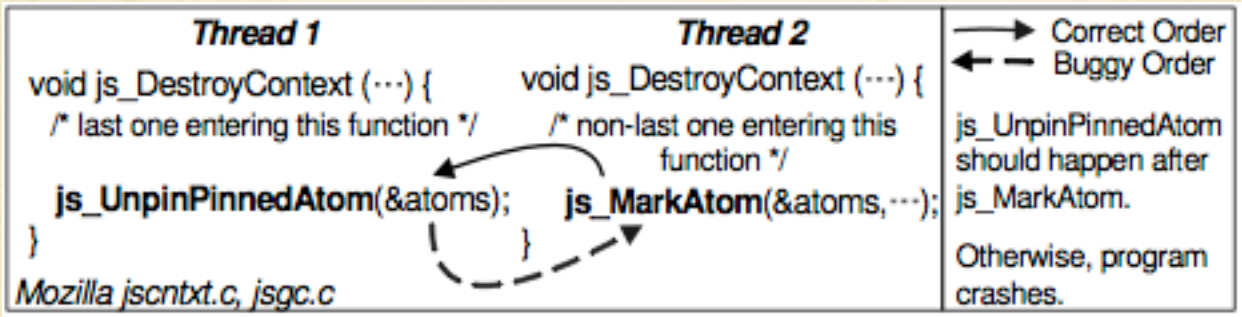


Figure 5. A Mozilla bug that violates the intended order between two groups of operations.

Bug Manifestation Study

- How many threads are involved?

Finding (3): The manifestation of most (101 out of 105) examined concurrency bugs involves no more than two threads.

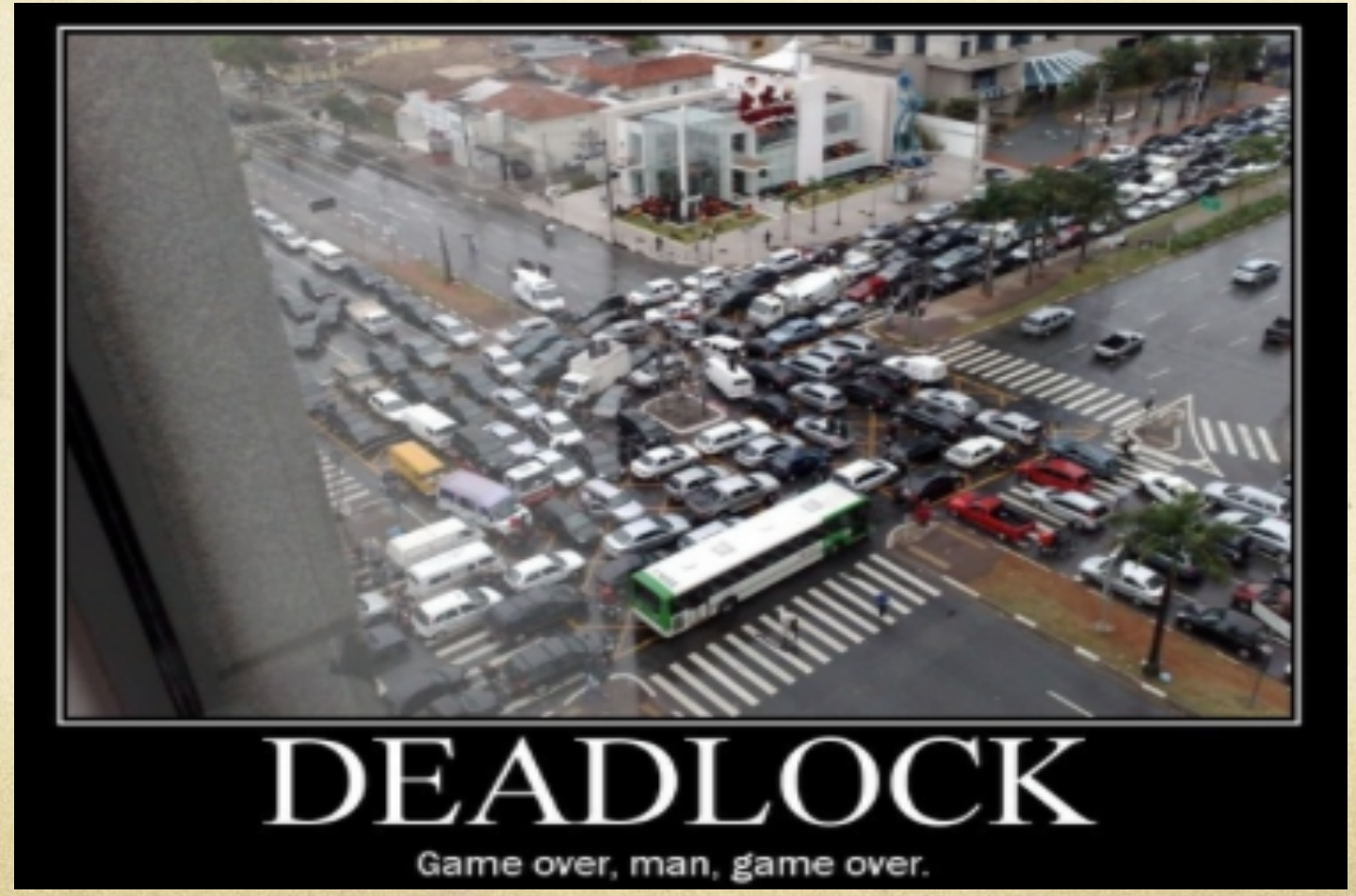
Implications: Concurrent program testing can pairwise test program threads, which reduces testing complexity without losing bug exposing capability much.

Non-deadlock concurrency bugs					
Application	Total	Env.	>2 threads	2 threads	1 thread
MySQL	14	1	1	12	0
Apache	13	0	0	13	0
Mozilla	41	1	0	40	0
OpenOffice	6	0	0	6	0
Overall	74	2	1	71	0
Deadlock concurrency bugs					
Application	Total	Env.	>2 threads	2 threads	1 thread
MySQL	9	0	0	5	4
Apache	4	0	0	4	0
Mozilla	16	0	1	14	1
OpenOffice	2	0	0	0	2
Overall	31	0	1	23	7

Table 5. The number of threads (or environments) involved in concurrency bugs.

Bug Manifestation Study

- How many threads are involved?



Bug Manifestation Study

- How many variables are involved?

Finding (5): 66% (49 out of 74) of the examined non-deadlock

Overall	31	1	23	7

Table 6. The number of variables (resources) involved in concurrency bugs.

Bug Manifestation Study

- How many variables are involved?

Non-deadlock concurrency bugs			
Application	Total	>1 variables	1 variable
MySQL	14	6	8

Finding (7): 97% (30 out of 31) of the examined deadlock concurrency bugs involve at most two resources.

Implications: Deadlock-oriented concurrent program testing can pairwise test the order among acquisition and release of two resources.

OpenOffice	6	0	6	2
Overall	31	1	23	7

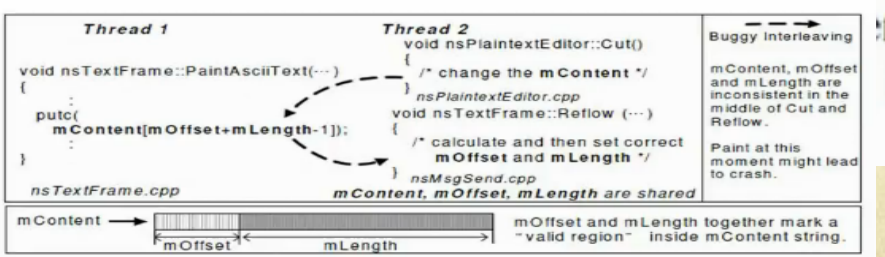


Figure 6. A multi-variable concurrency bug from Mozilla. Accesses to three correlated variables, `mContent`, `mOffset` and `mLength`, should be synchronized.

Non-deadlock concurrency bugs			
Application	Total	>1 variables	1 variable
MySQL	14	6	8
Apache	13	4	9
Mozilla	41	15	26
OpenOffice	6	0	6
Overall	74	25	49

Deadlock concurrency bugs				
Application	Total	>2 resources	2 resources	1 resource
MySQL	9	0	5	4
Apache	4	0	4	0
Mozilla	16	1	14	1
OpenOffice	2	0	0	2
Overall	31	1	23	7

Bug Manifestation Study

- How many accesses are involved?

Finding (8.1): 90% (67 out of 74) of the examined non-deadlock bugs can deterministically manifest, if certain orders among at most four memory accesses are enforced.

Finding (8.2): 97% (30 out of 31) of the examined deadlock bugs can deterministically manifest, if certain orders among at most four resource acquisition/release operations are enforced.

Implications: Concurrent program testing can focus on the partial order among every small groups of accesses. This simplifies the interleaving testing space from exponential to polynomial regarding to the total number of accesses, with little loss of bug exposing capability.

Non-deadlock concurrency bugs						
Application	Total	1 acc.*	2 acc.	3 acc.	4 acc.	>4 acc.
MySQL	14	0	2	7	4	1
Apache	13	0	6	5	2	0
Mozilla	41	0	12	18	5	6
OpenOffice	6	0	2	3	1	0
Overall	74	0	22	33	12	7
Deadlock concurrency bugs						
Application	Total	1 acc.*	2 acc.	3 acc.	4 acc.	>4 acc.
MySQL	9	4	1	4	0	0
Apache	4	0	0	4	0	0
Mozilla	16	1	2	12	0	1
OpenOffice	2	2	0	0	0	0
Overall	31	7	3	20	0	1

Table 7. The number of accesses (or resource acquisition/release) involved in concurrency bugs. (*: "1 acc." case happens only in deadlock bugs, when one thread waits for itself. The bug triggering therefore does not depend on any inter-thread order problem.)

0	0	0	0
3	20	0	1

r resource acquisition/release) in-
c." case happens only in deadlock
The bug triggering therefore does
oblem.)

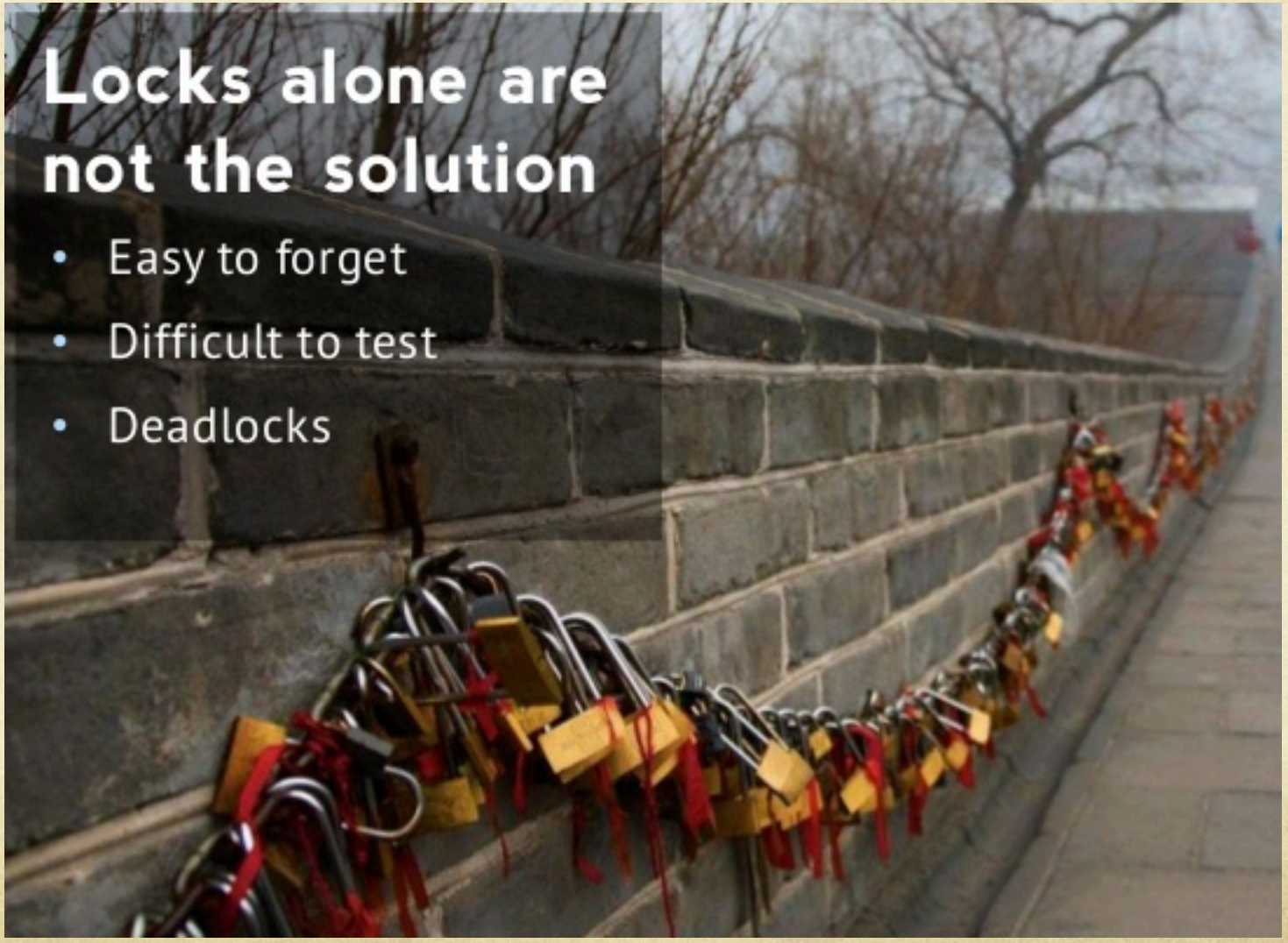
Bug fix Study

- Fix strategies
- Mistakes during bug fixing
- Bug avoidance

Bug fix Study

- Fix Strategies
  - Adding / changing locks
  - Condition check
  - Code switch
  - Algorithm/Data-structure design change
  - Others

Bug fix Study



Bug fix Study

- Fix Strategy
  - Condition check

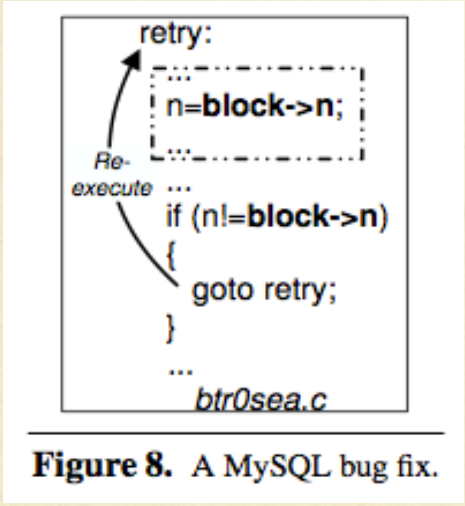
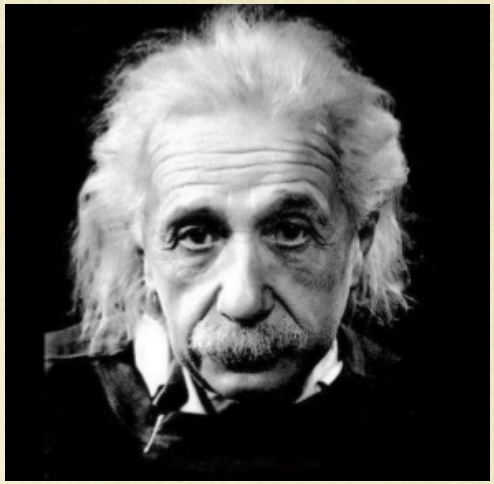


Figure 8. A MySQL bug fix.



Problems can't be solved by the same level of thinking that created them!!!

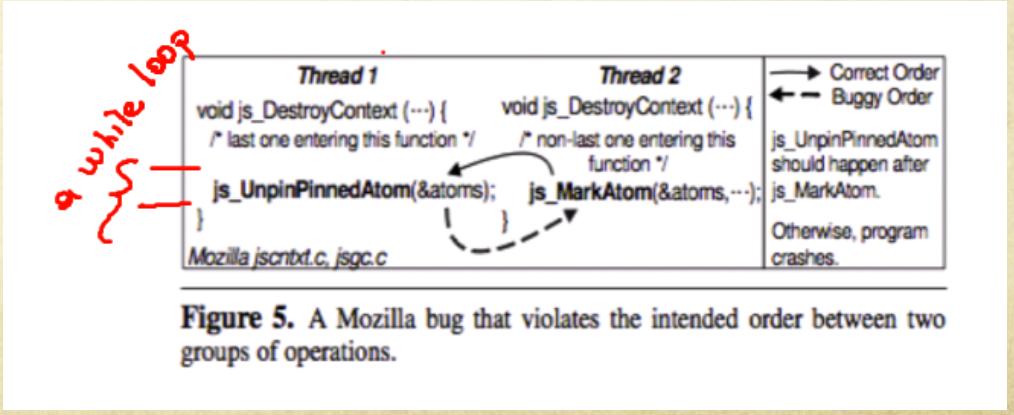


Figure 5. A Mozilla bug that violates the intended order between two groups of operations.

Bug fix Study

- Fix Strategies

Application	Total	GiveUp	Split	AcqOrder	Other
MySQL	9	5	0	2	2
Apache	4	2	0	2	0
Mozilla	16	11	1	3	1
OpenOffice	2	1	0	0	1
Overall	31	19	1	7	4

Table 9. Fix strategies for deadlock bugs (all categories are explained in Table 2)

Finding (10): The most common fix strategy (used in 19 out of 31 cases) for the examined deadlock bugs is to let one thread give up acquiring one resource, such as a lock. This strategy is simple, but it may introduce other non-deadlock bugs.

Implication: We need to pay attention to the correctness of some "fixed" deadlock bugs.

Bug fix Study

- Mistakes during bug fixing
  - Buggy patches
  - Every fix should be re-tested and analyzed.

Bug fix Study

- Bug avoidance

Finding (11): TM can help avoid many concurrency bugs (41 out of the 105 concurrency bugs we examined).

Implication: Although TM is not a panacea, it can ease programmers correctly expressing their synchronization intentions

Finding (12): TM can potentially help avoid many concurrency bugs (44 out of the 105 concurrency bugs we examined), if some concerns can be addressed, as shown in Table 10.

Implication: TM design can combine system supports and other techniques to solve some of these concerns, and further ease the concurrent programming.

Apache	17	7	0	3	1	6
Mozilla	57	25	8	9	5	10
OpenOffice	8	2	0	4	0	2
Overall	105	41	8	30	6	20

Table 10. Can TM help avoid concurrency bugs?

Other Characteristics

- Bug impacts
  - 34 cause program crashes
  - 37 of them cause program hangs
- Non-determinism
  - Some concurrency bugs are very difficult to repeat
    - "Mozilla and occur once in a day bug"
- Test cases are critical to bug diagnosis
  - Good test inputs for repeating bugs.
- Programmers lack diagnosis tools & training

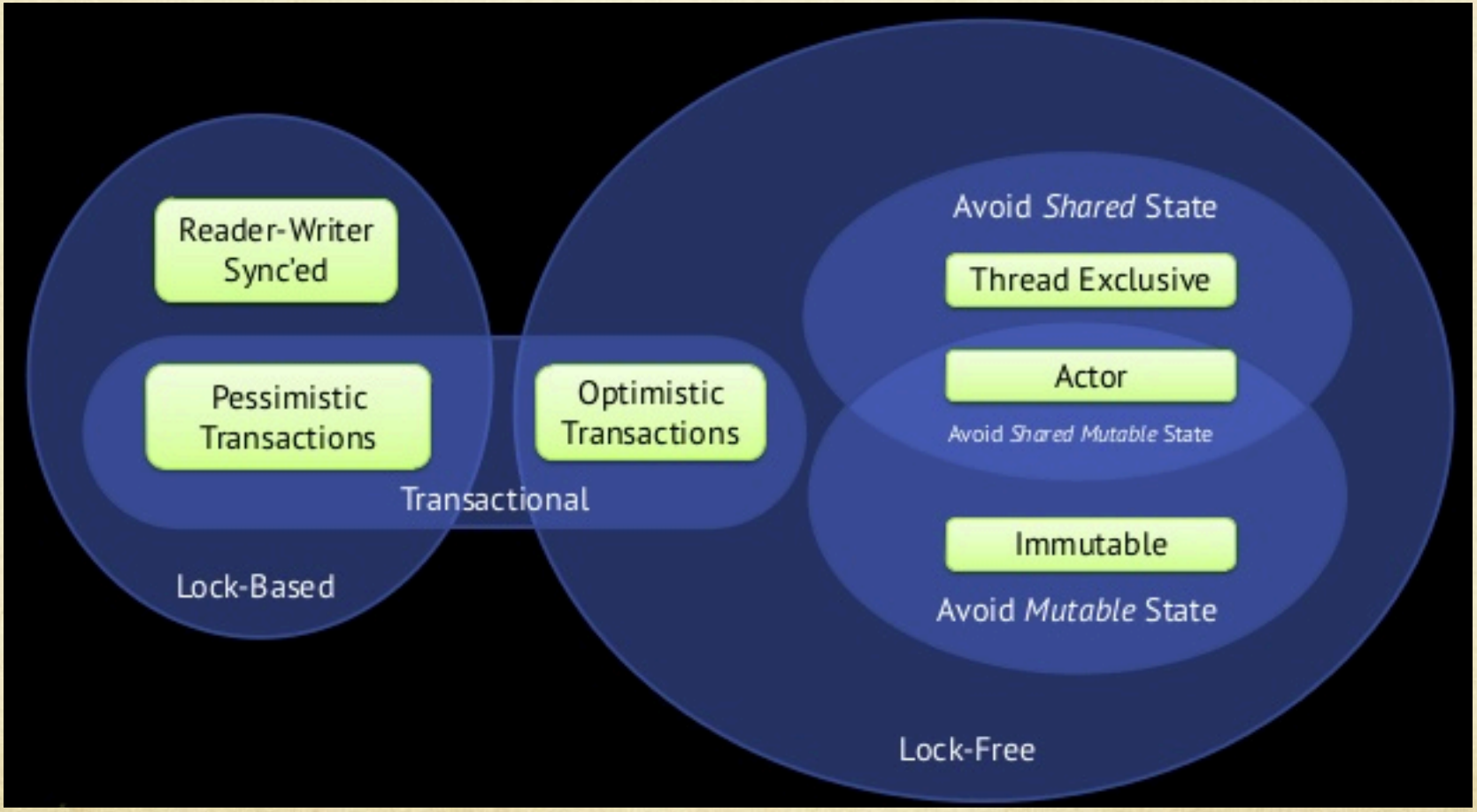
Conclusions

- Comprehensive study of the real world concurrency bugs
- Give us alternative views for
  - Concurrency bug detection
  - Concurrent program testing and model checking
  - Concurrent programming language design

Questions

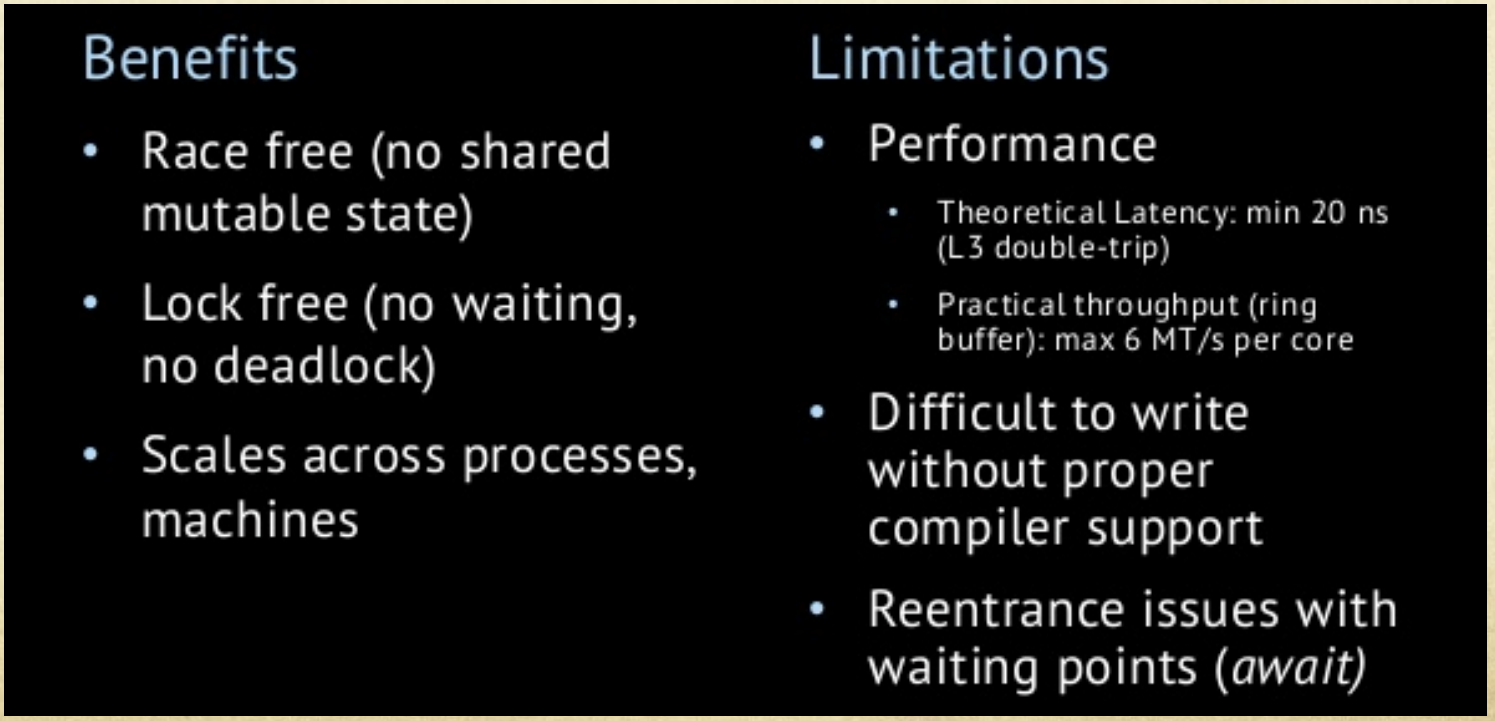
- Don't hesitate to ask!!!

EXTRA - I

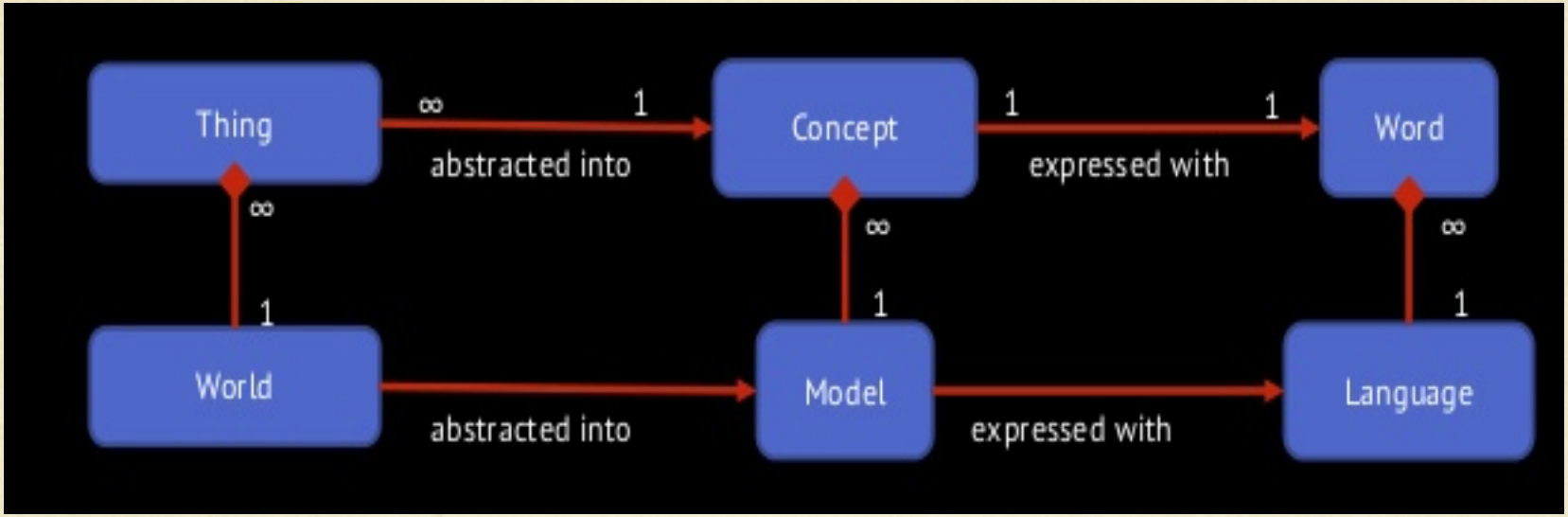

Reader-Writer
Sync'ed
Pessimistic
Transactions
Optimistic
Transactions
Transactional
Lock-Based
Avoid *Shared* State
Thread Exclusive
Actor
Avoid *Shared Mutable* State
Immutable
Avoid *Mutable* State
Lock-Free

EXTRA-II

- Actors
  - Bound to a single thread (at a time)
  - No mutable Shared State (All mutable states are private)



EXTRA-III

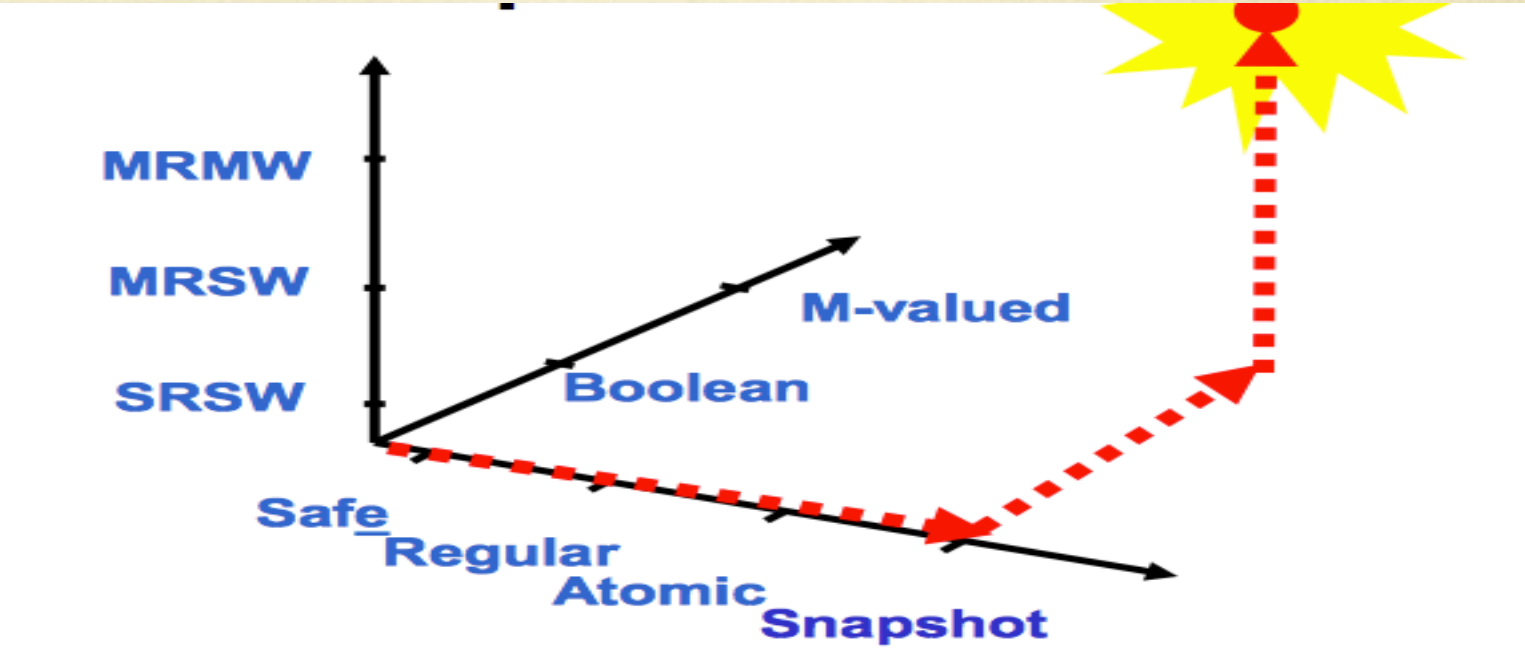


Languages let us express ourselves against a model!!!!

EXTRA-IV

- LOCKS
  - Should be used to protect logical invariants, not memory locations
  - Rules while using them:
    - Avoid them by implicit replication
    - Try to at most one lock at a time : never call other people's code while holding a lock.
    - Always acquire locks on in the same order (Deadlock problem)
      - Stratify locks, Assign each lock a level such that 2 locks on the same level are never locked at the same time, then always acquire locks in level order.
      - Sort the locks to be locked
      - Back off : acquiring a set of locks, if any lock can't be acquired immediately, release all locks already acquired.
      - Don't contend on a lock a lot, "Strangled Scaling"

EXTRA-V



Snapshot means:

- Write any array element
- Read multiple array elements atomically

EXTRA-VI

- Wait-Free : each method call takes a finite number of steps to finish
  - Nicer guarantee, no starvation
- Lock-free : infinitely often some method call finishes
  - Practically good enough
  - Gambling on non-determinism
- Consensus is the synchronization power
  - Universal, meaning from n-thread consensus
    - Wait-free/Lock-free
    - Linearizable
    - N-threaded
    - Implementation
    - Of any sequentially specified object