BugNet

Continuously Recording Program Execution for Deterministic Replay Debugging

Satish Narayanasamy

Gilles Pokam

Brad Calder

UCSD

Motivation

Current Scenario

Increasing Software Complexity

Difficult to guarantee correctness

Released software contain bugs

Problem

Bugs manifest at customer site

Difficult to reproduce bugs at developer site

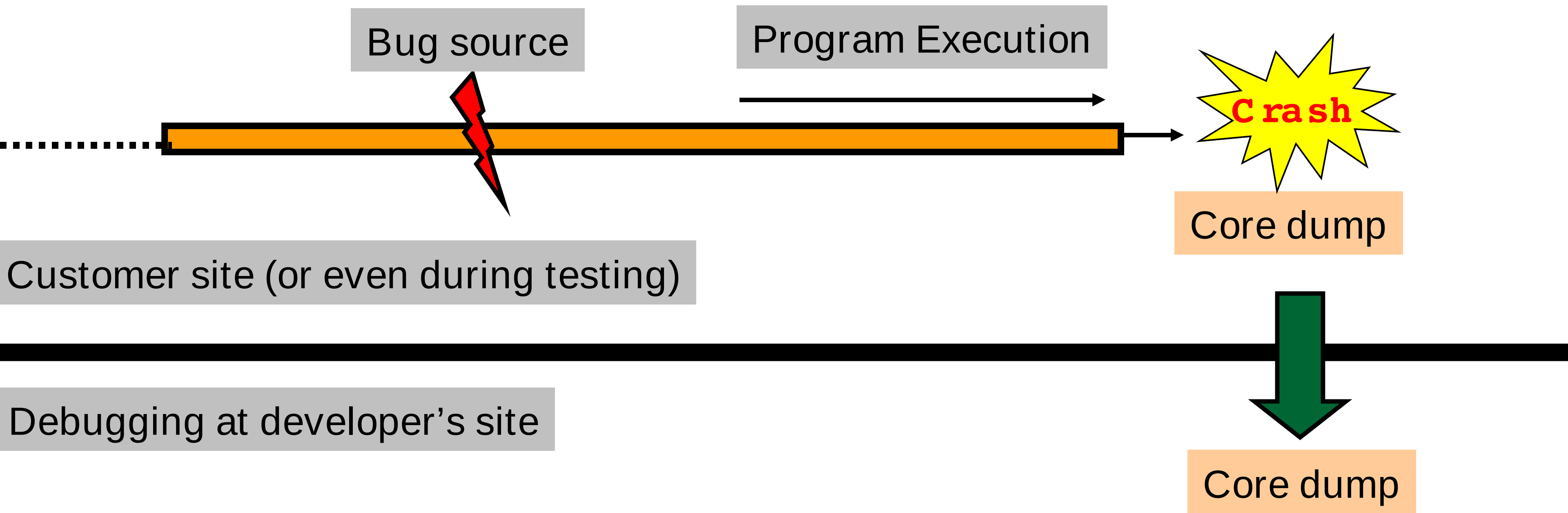
Solution

Continuously record information about program execution, even during ***production runs***

Challenge

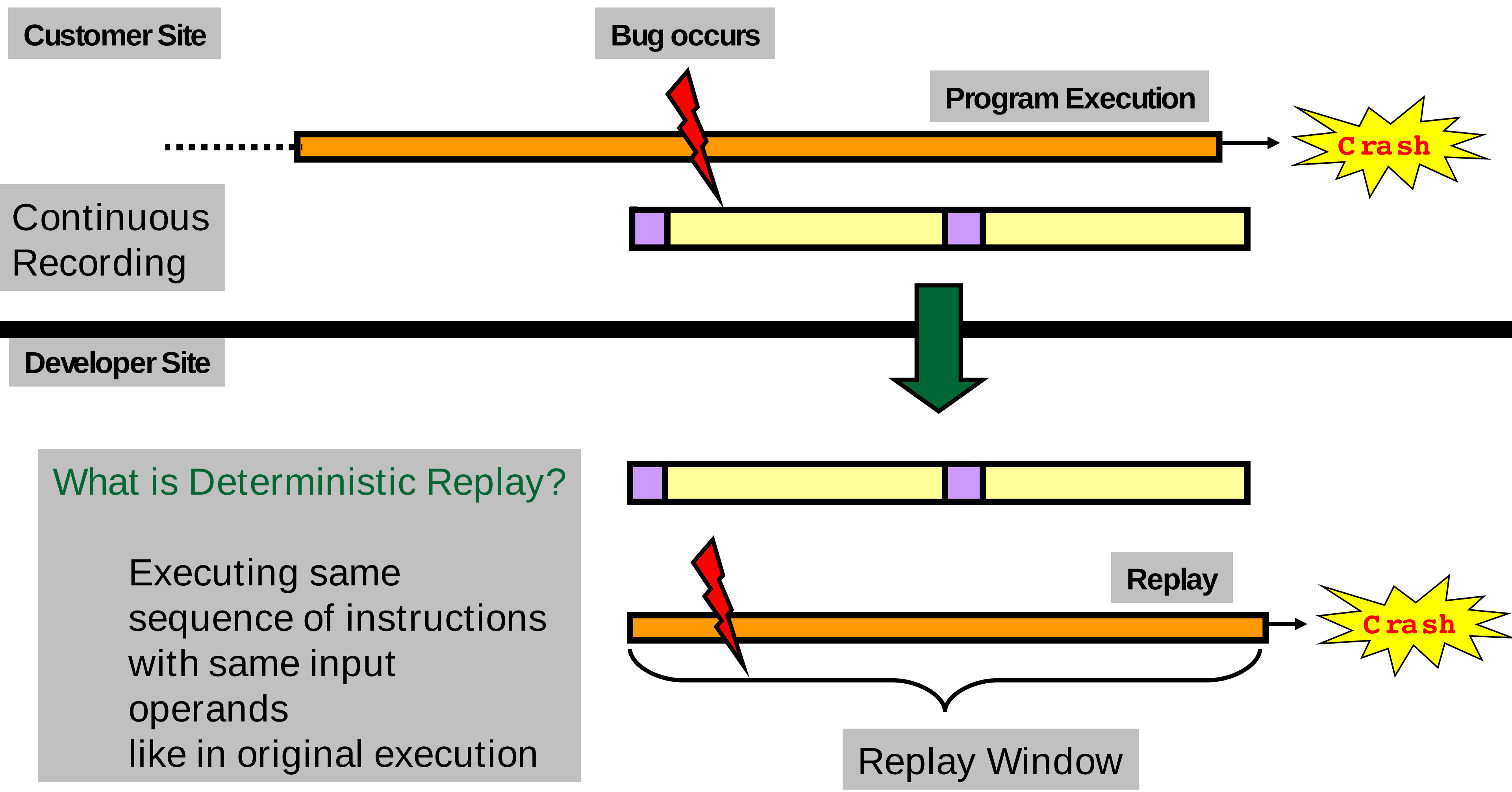
Recording should be transparent to customer –> HW can help!

Conventional Debugging



- Examine core dump
  - Developer can examine final system state just before the crash
  - Very challenging to determine the root cause

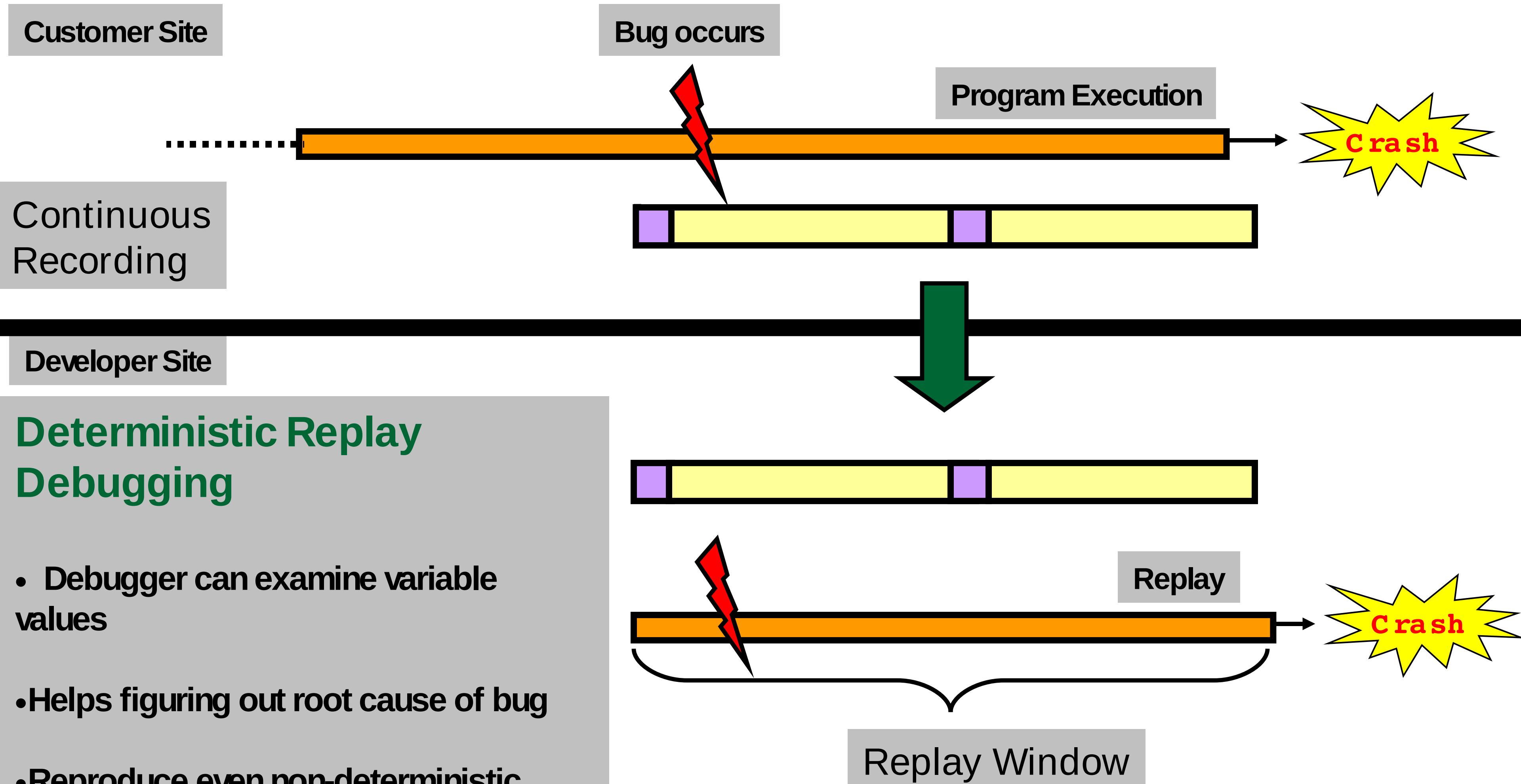
Deterministic Replay Debugging



What is Deterministic Replay?

Executing same sequence of instructions with same input operands like in original execution

Deterministic Replay Debugging



Deterministic Replay Debugging

- Debugger can examine variable values
- Helps figuring out root cause of bug
- Reproduce even non-deterministic Bugs

BugNet

Goal

- Architecture support to enable Deterministic Replay Debugging

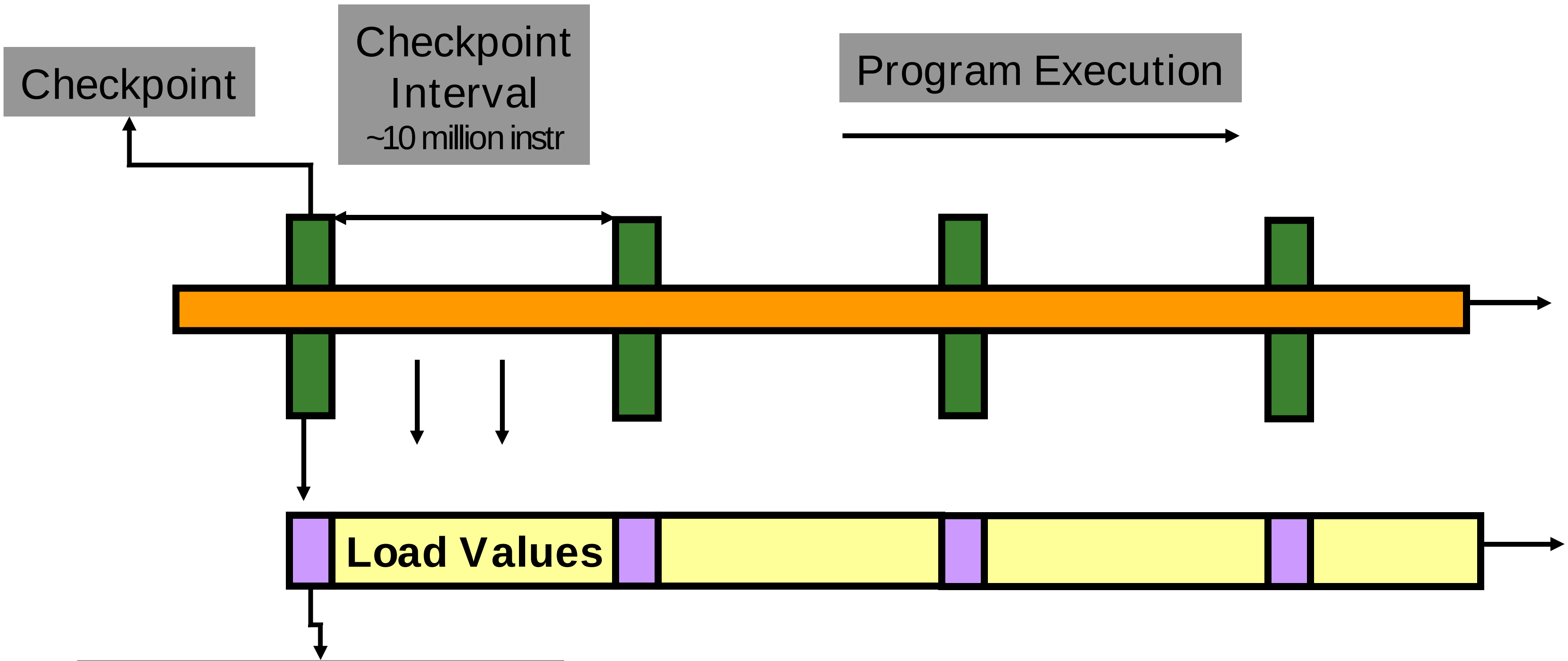
Focus

- Debugging user code
  - Application and shared libraries
  - No logging during execution of system code (interrupt service routines, system calls)

Approach

- Log initial architectural state (registers, PC, etc) and then load values
  - Sufficient to replay user code, *even across interrupts etc..*

Overview

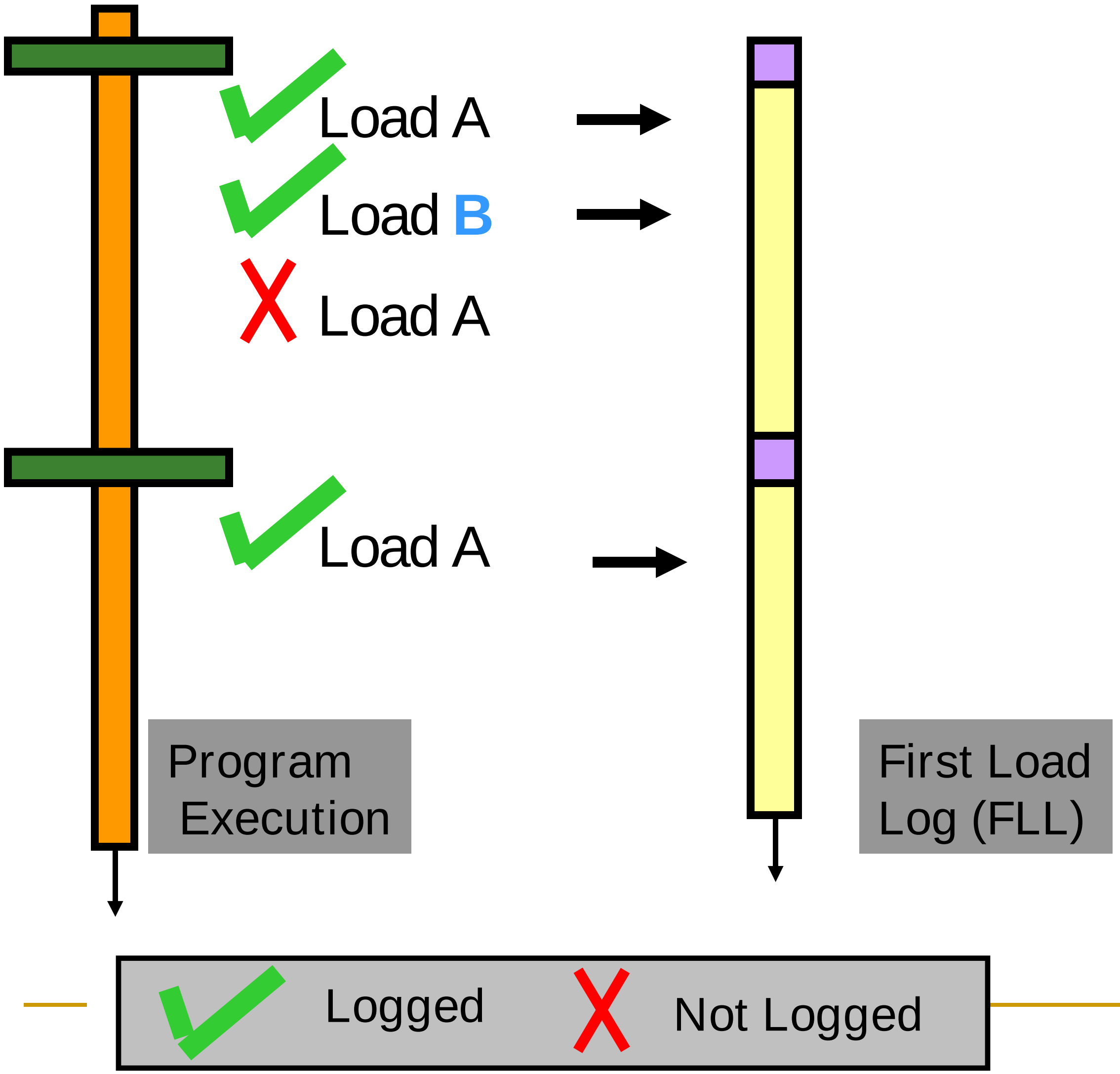


Log Header
- Program Counter
- Arch Register Values
- Process ID, Thread ID
- Checkpoint ID
- ..

Only output of loads need to be logged

Input and output values of other instructions can be *regenerated* during replay

First Load Log

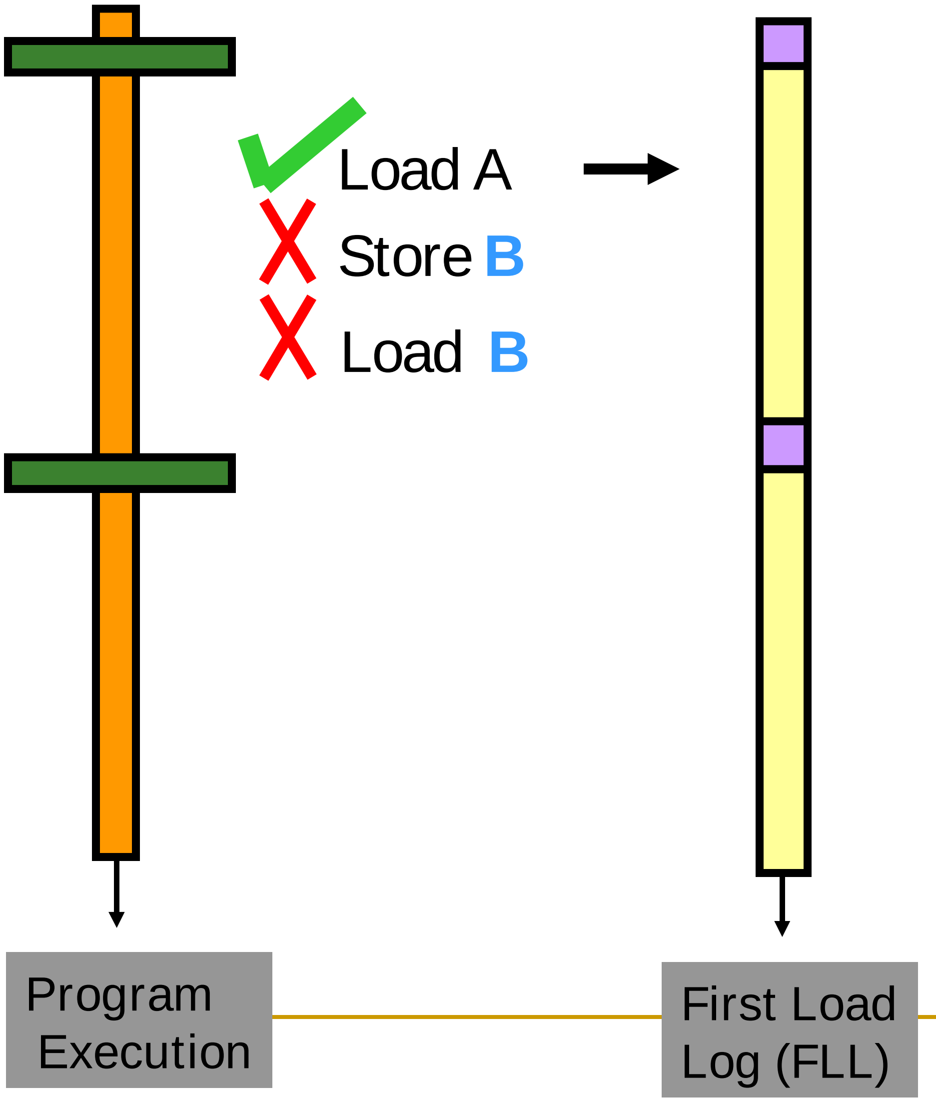


Log load value only if the load is the *first* memory access to a location

HW Support:

- **"FLL bits" for every word in L1 and L2 caches**
  - **Reset at the beginning of a checkpoint interval**
  - **Set on access**

First Load Log



Store values never need to be logged

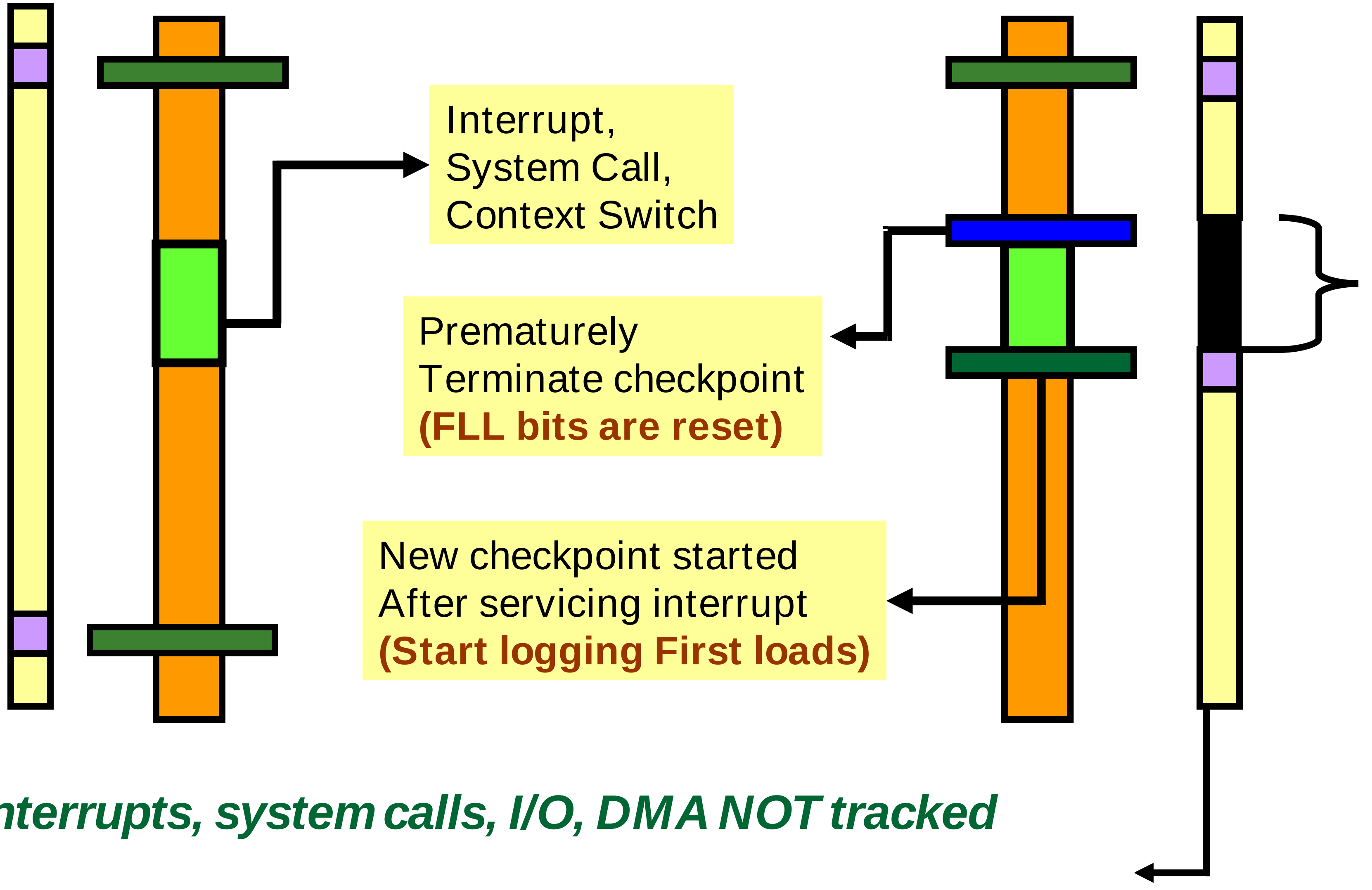
Regenerated during replay

PROBLEMS

Memory location can be modified by stores in

- **Interrupts, system calls**
- **Other threads in multithreaded programs**
  - **DMA transfers**

Interrupts



Interrupts, system calls, I/O, DMA NOT tracked

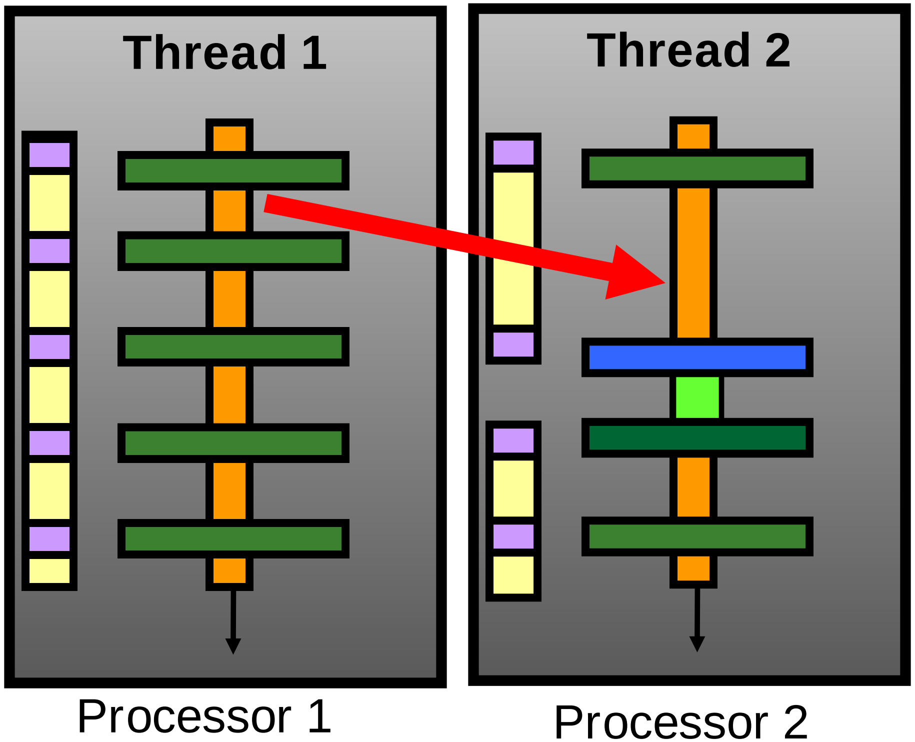
BUT any values consumed later by the application will be logged, ON DEMAND, in the new checkpoint

Support for Multi-threaded Programs

Assumptions for Multithreaded Programs

- Shared Memory Multi-threaded processors
- Sequential Consistency
  - Memory operations form a total order
- Directory based Cache Coherence protocol

Shared Memory Communication

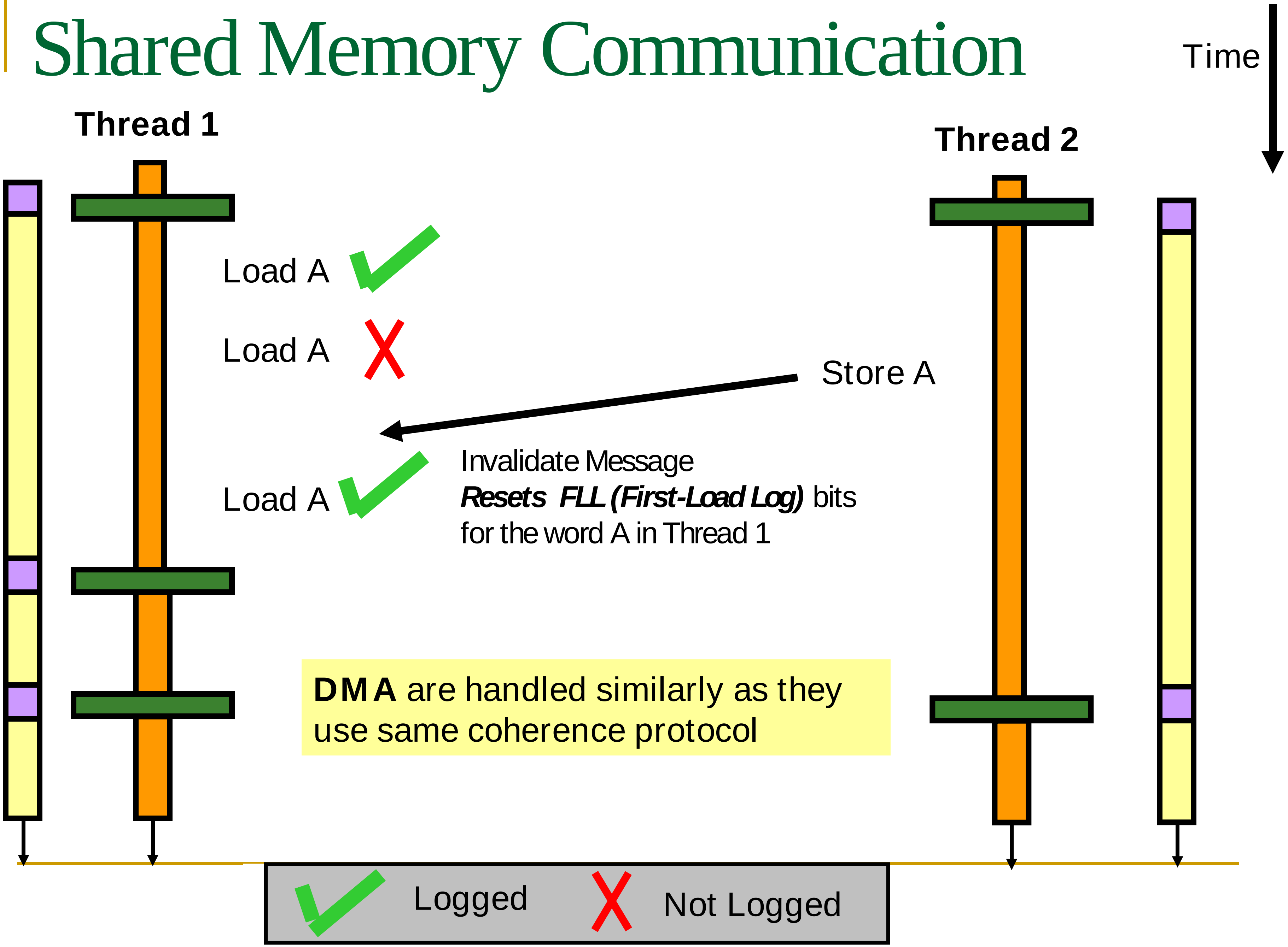


A First Load Log (FLL) for each thread is collected *locally*

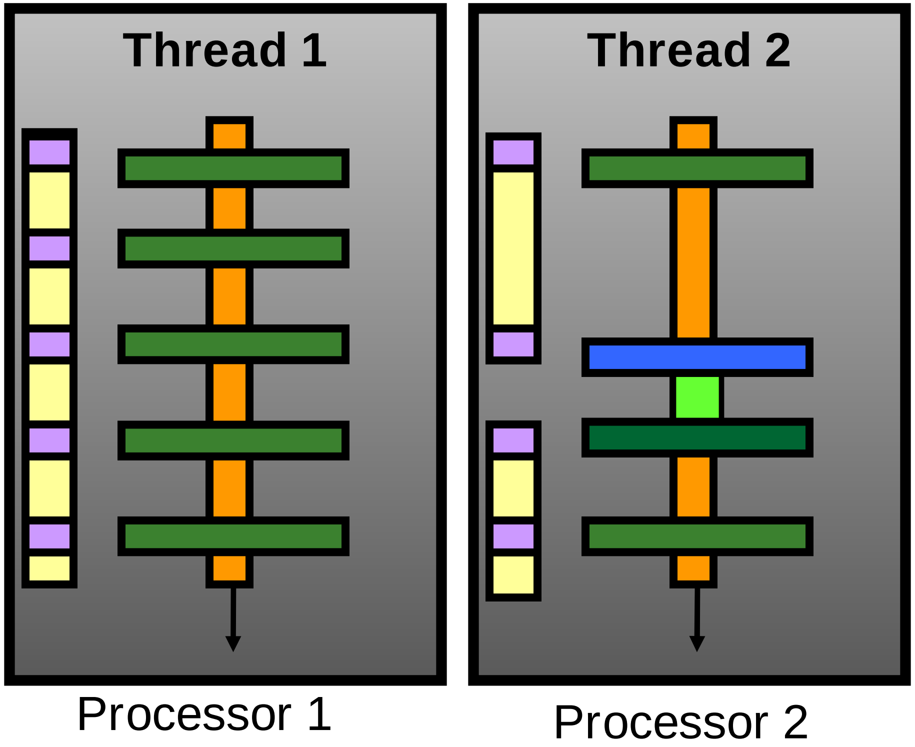
Problem :

- Shared memory communication between threads
  - **Affects First Load optimization**

Shared Memory Communication



Independently Replaying Threads



A thread can be replayed using its local FLL, ***independent of other threads***

FLL checkpoints in different threads need not begin at the same time

- **Prematurely terminating checkpoints for interrupts becomes easier**

Logging Memory Order

Infer and debug data races

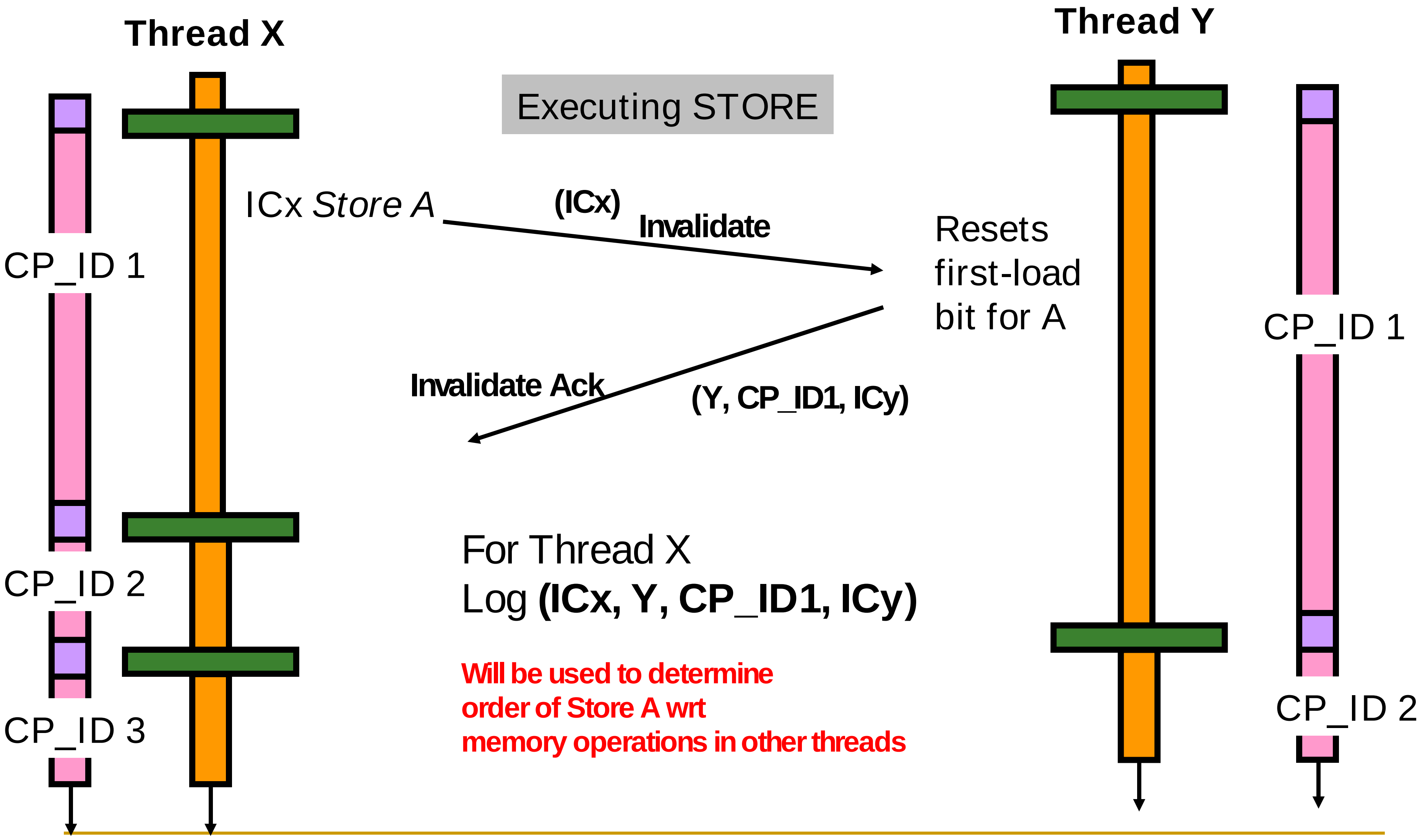
Log order of memory operations executed across all the threads

Adapt **Flight Data Recorder (FDR)**

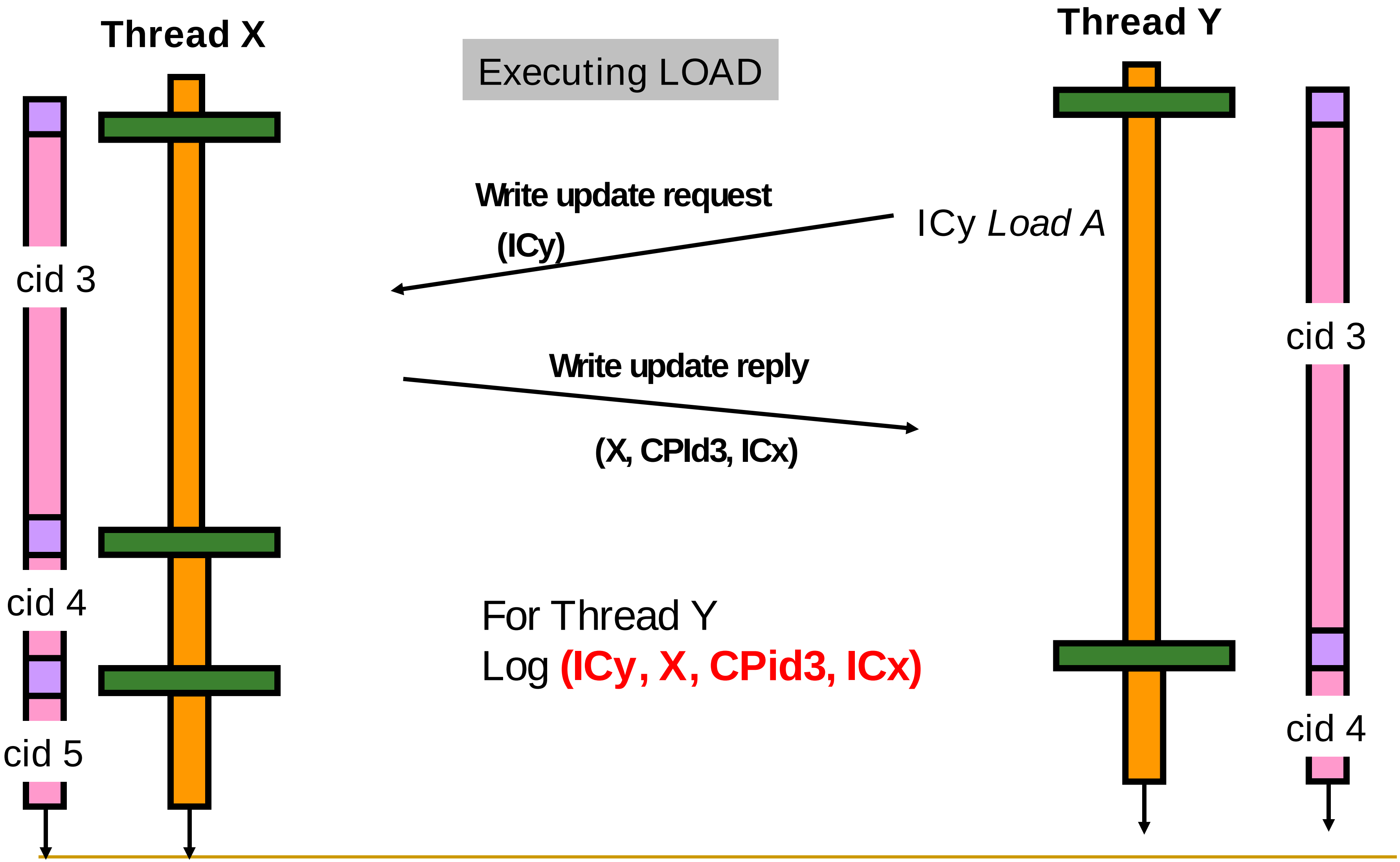
Xu, Bodik, Hill ISCA'03

Piggyback *coherence replies* with *execution states* (Thread-ID, Checkpoint-ID, Inst Count) of sender thread

Memory Race Log



Memory Race Log



Architecture Support Summary

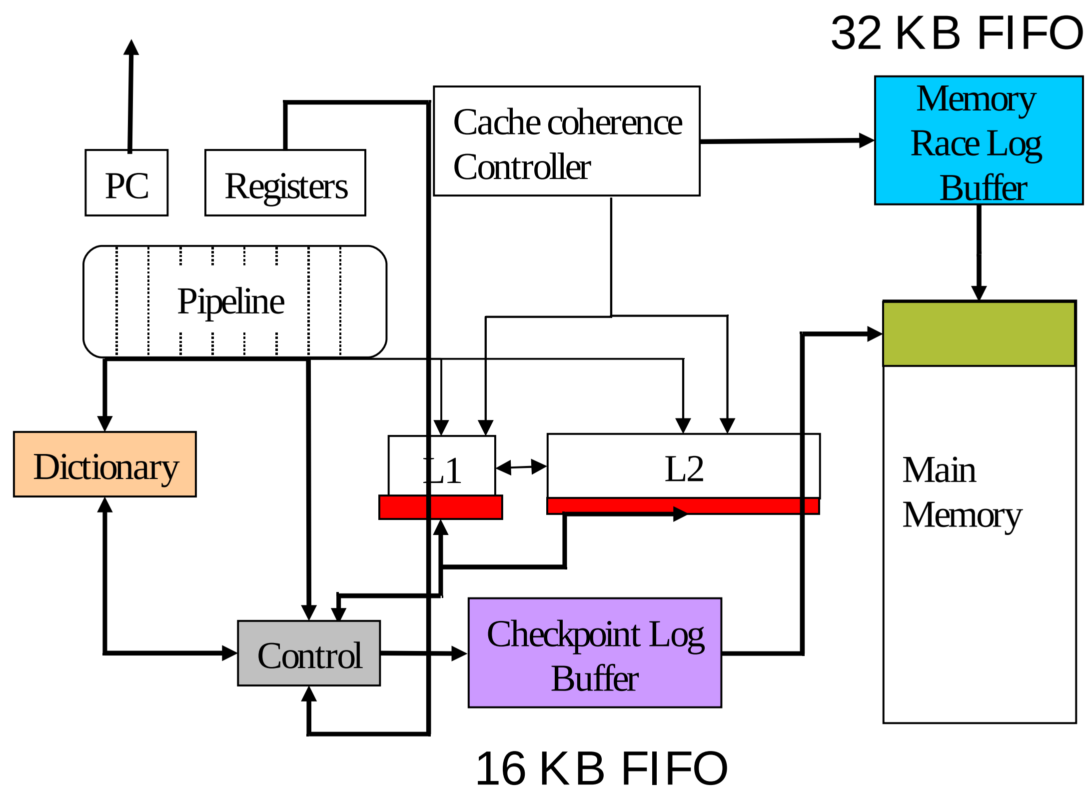
Goal: Deterministically Replay Crash

Checkpoint Mechanism

- **First Load Opt**
- **Online Dictionary Based Compression**

Memory Backed

Support for Multithreading



Memory Back Support

- Handling bursts
  - CB -16 KB; MRB – 32 KB
  - During bursts, CB & MRB buffers can get full
    - Processor stalled OR
    - Flush the buffer and start a new checkpoint

- CB and MRB are memory backed
  - Contents continuously written back to main memory at two separate locations
  - Amount of main memory space allocated determines replay window length

Checkpoint Management

- Oldest checkpoint discarded when allocated main memory space is full
- Checkpoint Interval length chosen based on available main memory space
  - Tradeoff
    - Smaller the checkpoint interval lesser the information loss when a checkpoint is discarded
    - Larger the checkpoint interval lesser the information/instruction that need to be logged
      - Reason: First-Load optimization

Re-player Infrastructure

- Collecting FLL
  - Pin Dynamic Instrumentation **Luk et al., PLDI ’05**
- Replaying program execution using FLL
  - Virtutech Simics
    - A full system functional simulator

How to replay a checkpoint?

- Replay using a functional simulator – eg: Simics
  - Can be integrated into conventional debuggers
- Steps:
  - Load the binaries into the same address locations like in the original location
  - Initialize state of PC and architectural registers
  - Start emulating instructions
  - For first loads, get the value from FLL, else get value from simulated memory
- *Core Dump Not Required*

Re-player Implementation Issues

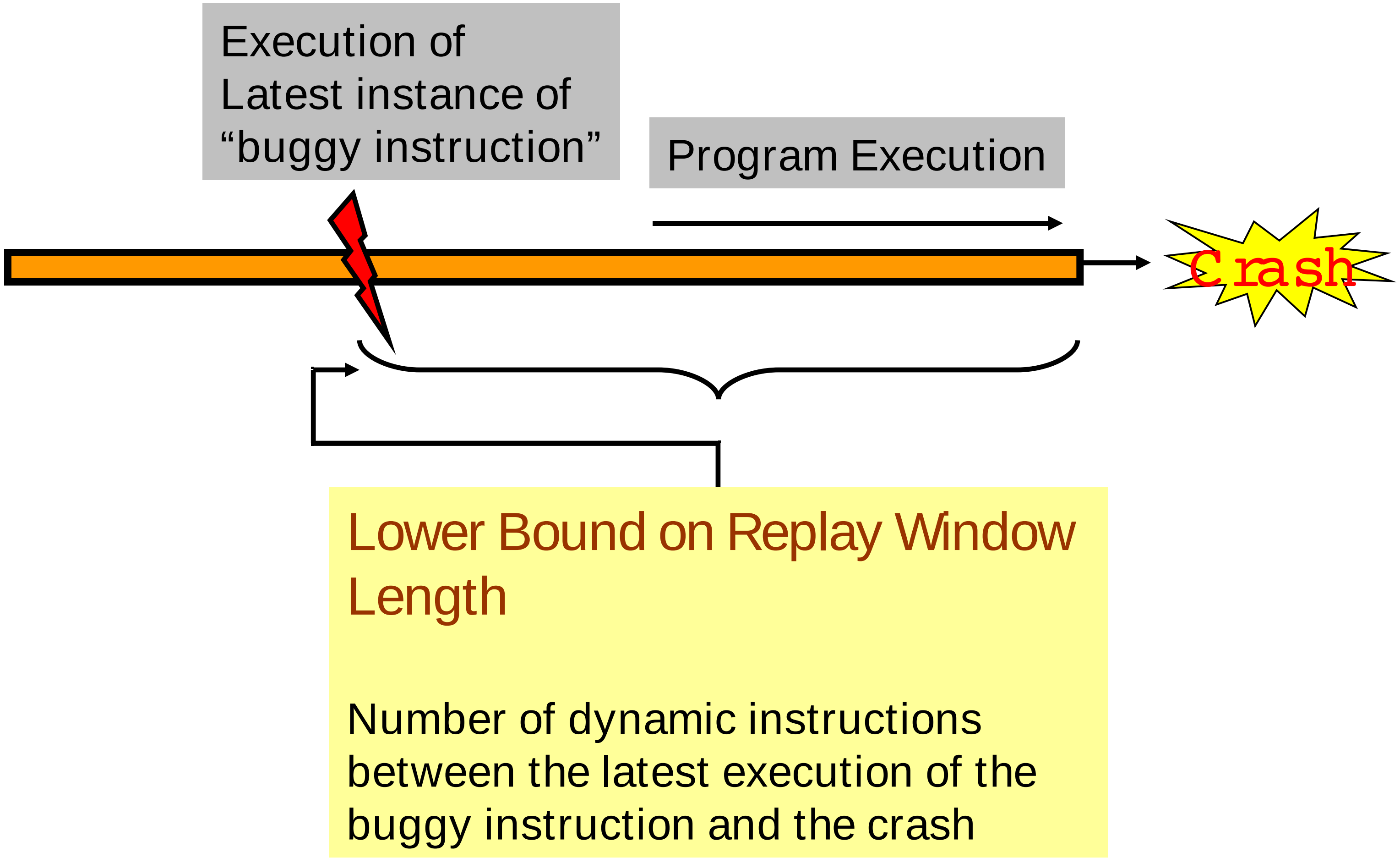
- Code Space
  - Address locations of application code and shared libraries in application's virtual address space need to be same as in the original execution
    - Solution: Include starting locations of user and library code space in the log

  - Developer should have access to binaries and libraries used by the customer

- **Self-Modifying Code**
  - Cannot be handled by BugNet
    - Reason: Instructions are not logged

- **Possible Solution**
  - Log first load (fetch) of instructions

Replay Window Length



Bug Characteristics

Lower bound on required replay window length

Program	Nature of Bug	Replay Window length (in instructions)
gzip	Overflows global variable	32,209
ncompress	Stack Corruption	17,966
tar	Heap object Overflow	6,634
ghostscript	Dangling pointer	*18,030,519*
tidy	Null pointer dereference	2,537,326
xv-3.10a	Buffer overflow	7,543,600
gaim-0.82.1	Null pointer dereference	74,590
napster-1.52	Dangling pointer	189,391
python	Buffer Overflow	92
w3m	Null pointer dereference	79,309
Average		1,594,252

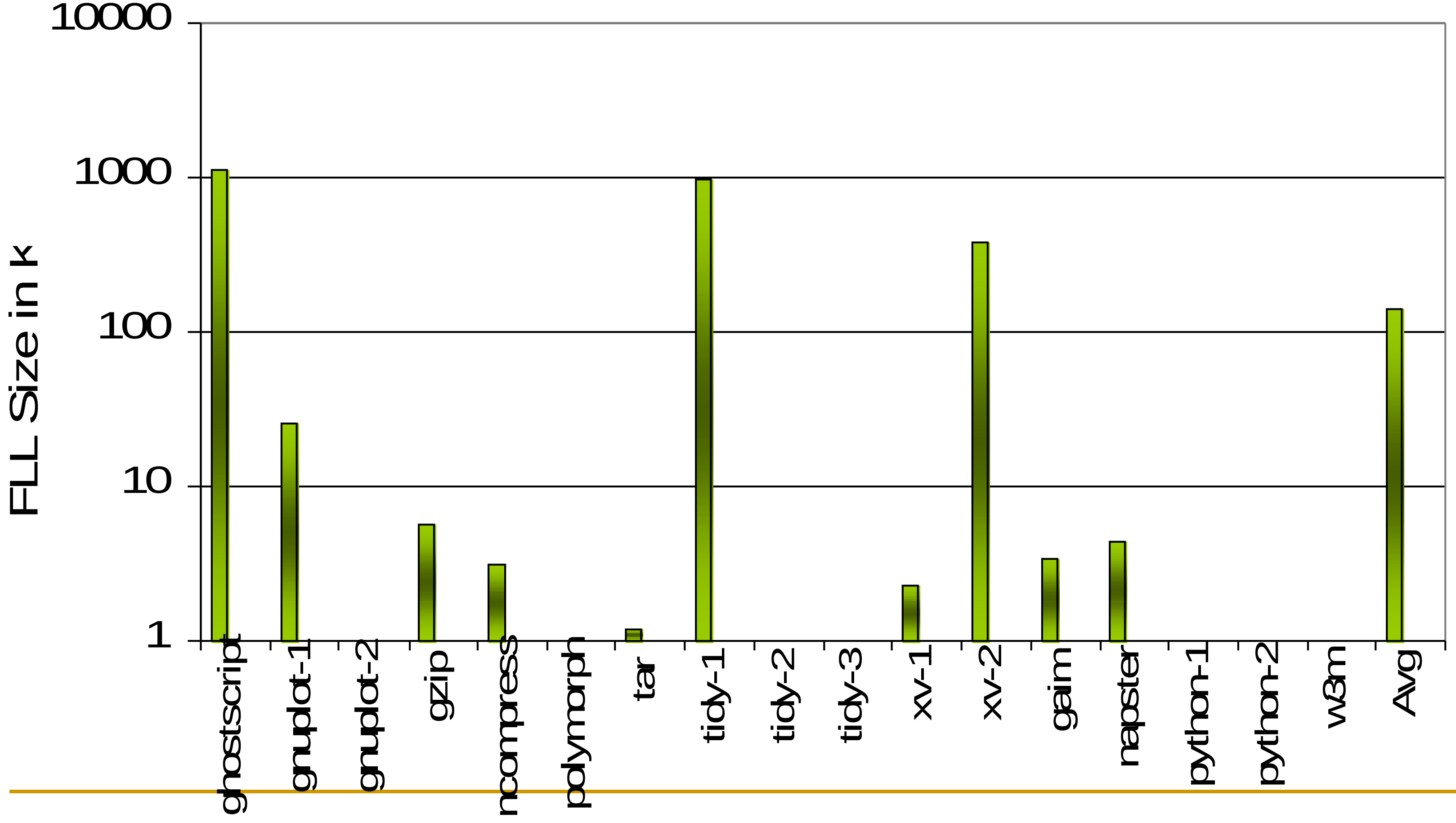
AccMon Zhou et.al. MICRO'04

Sourceforge Single Threaded

Sourceforge Multi-Threaded

FLL Trace Size

Less than 1MB (<20M interval) is required to capture majority of bugs



BugNet Vs FDR (Xu, Bodik & Hill ISCA'03)

Flight Data Recorder (FDR) – Replay ***full system*** for debugging

- Uses **SafetyNet Checkpoint** Mechanism **Sorin et.al. ISCA'02**
  - Logs values replaced by first stores
  - Recover initial full system state from **core dump** and store log
- To enable replay, Interrupt, Prg I/O, DMA are logged separately
- Requires more HW and larger logs than BugNet

BugNet -- Focus on debugging only application code

- First load checkpoint mechanism
- Core dump, Interrupt, I/O, DMA logs NOT required

Performance overhead of both is negligible

- Logging is off the critical path of main computation

Limitation

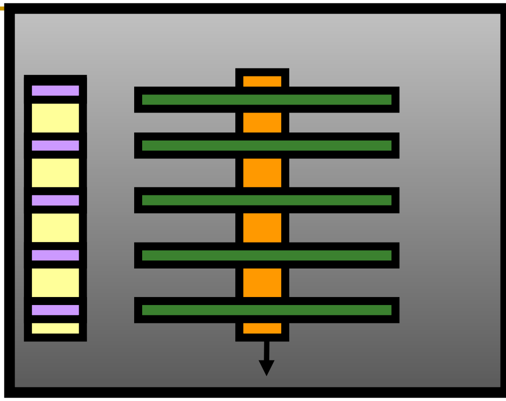
- Debugging ability
  - Debugging OS code not possible
    - BUT, memory values modified during interrupts, I/O and DMA will be captured in FLL
      - Hence, the application with limited interactions with OS can be debugged
  - No Core Dump
    - Values of data structures untouched during replay window are unknown
    - ***BUT, values responsible for bug can be found in the log or reproduced during replay if the replay window is large enough to capture the source of bug***

If a variable is not accessed between the source of bug and the crash then it should not be a reason for the crash

Limitation

- Replay window not long enough
  - Problem:
    - Cause of bug lie outside replay window
  - Reason:
    - Limited storage space -- Depends on amount of main memory to devote to capture logs
  - Solution:
    - OS can fine tune allocation
      - User Input
      - Memory usage at any instant of time

Summary



Bugs in released software are difficult to reproduce

- Goal is to continuously record a light weight trace at the customer's site to capture hard to reproduce bugs

Deterministic Replay Debugging

- On average *at least* 1.5 million instructions need to be replayed to capture bugs that we studied

Recording architectural state and load values are sufficient to enable replay

- Small FLL log size
  - No core dump
  - No I/O, DMA, Interrupt logs

Replay Window	FLL Size
20 Million instr	< 1 MB
100 Million instr	< 3 MB

Limitation

- Debug only user code and shared libraries
  - Though it supports replaying across interrupts