Light64: Lightweight hardware support for data race detection during systematic testing

Adrian Nistor, Darko Marinov, Josep Torrellas

University of Illinois, Urbana – Champaign
http://iacoma.cs.uiuc.edu

Outline

Motivation

Systematic Testing

Light64

Evaluation

Conclusion

Data Races

- Common concurrency bug
- Difficult to detect
- Cause unexpected crashes even in code that is well tested
- Example:

X == 0

Thread A

X += 1

Thread B

X += 1

Depending on the run: X = 2 or X = 1

Contribution: **Light64**

Light64: new data race detection technique

	Software	**Light64**	Hardware
Hardware requirement	**none**	**64 bits**	**72-400 Kbits**
Execution overhead	**8 X**	**1 – 37%**	**0.5%**

NO false positives

Detects **96%** of races

Outline

Motivation

Systematic Testing

Light64

Evaluation

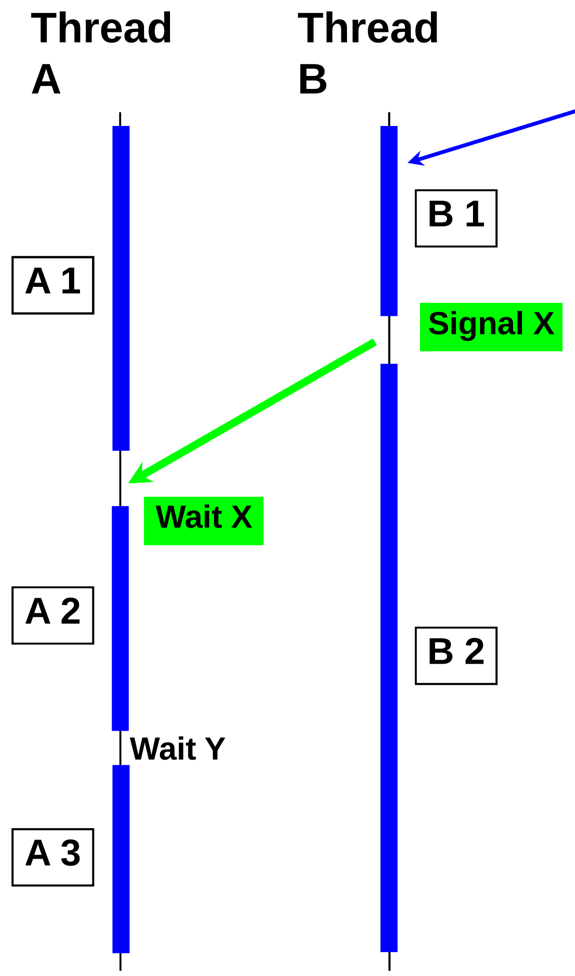
Conclusion

Systematic Testing

- To detect bugs, we need high test coverage
- Very important in parallel programs
    - One input, many thread interleavings
- Systematic testing
    - Systematically execute many thread interleavings
    - Example: CHESS (used by Microsoft testers)
- Systematic testers include data race detection
- Turned **off by default**
    - Due to high runtime overhead
- Light64: Overhead low enough to be **always ON**

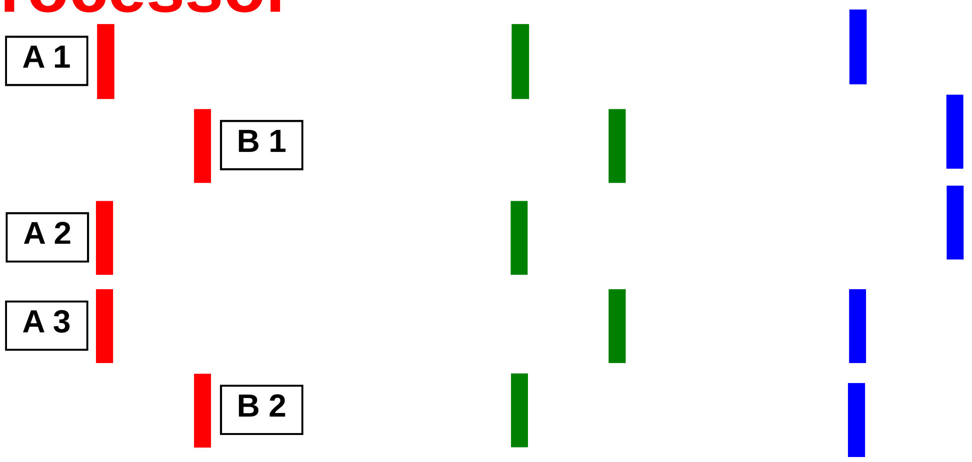
How Systematic Testing Works

SEGMENT == sequence of dynamic instructions



Execute many different interleavings

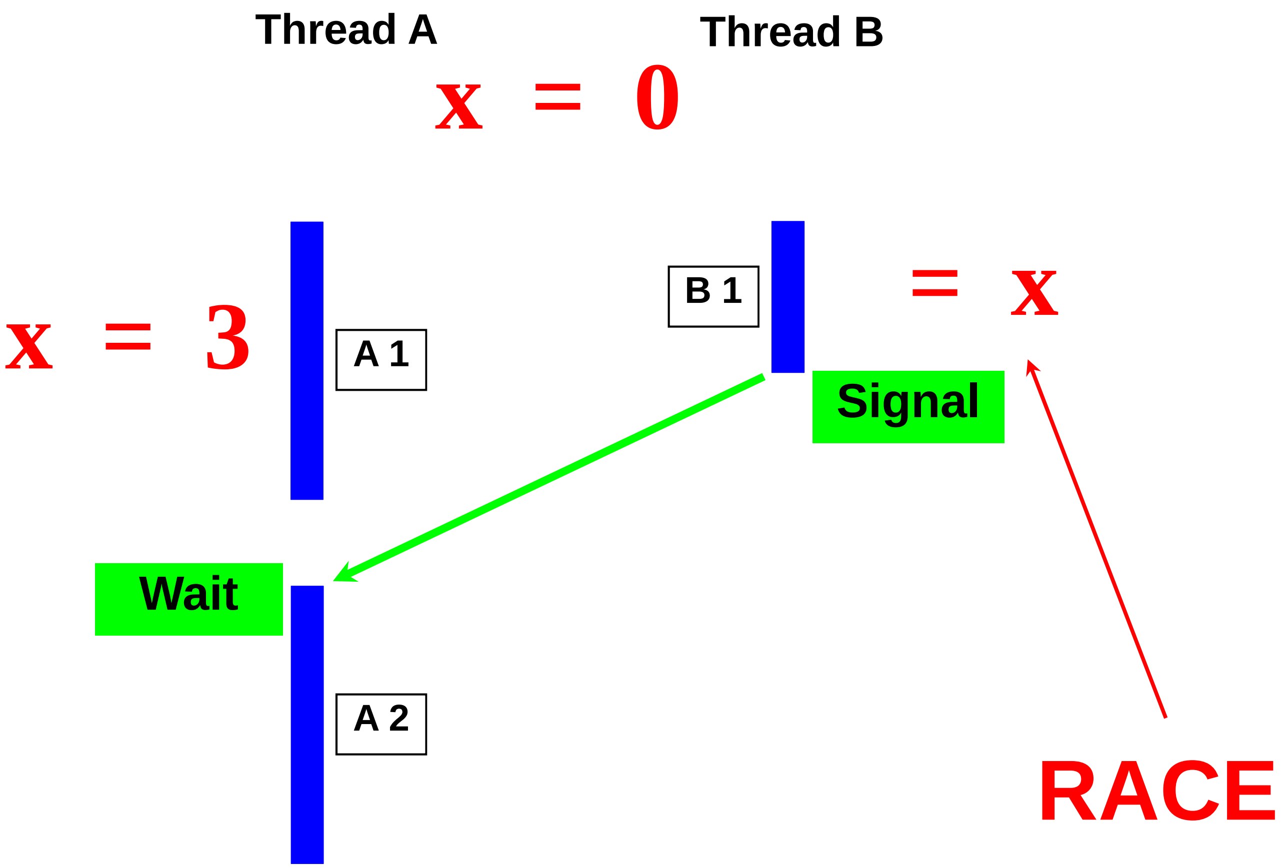
Multiplex segments in a uniprocessor



Outline

- Motivation
- Systematic Testing
- **Light64**
- Evaluation
- Conclusion

Example



Thread A

Thread B

x = 0

x = 3

A 1

B 1

= x

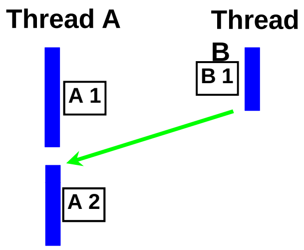
Signal

Wait

A 2

RACE

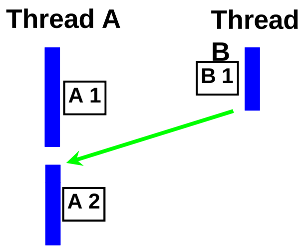
Race on X: because accesses to X are not ordered by synchronization

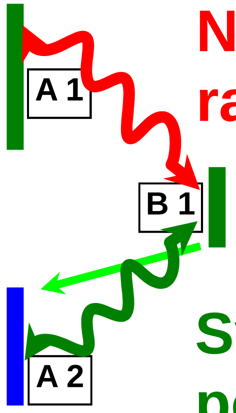


The Idea

Perform two executions flipping the unordered segments

The Idea

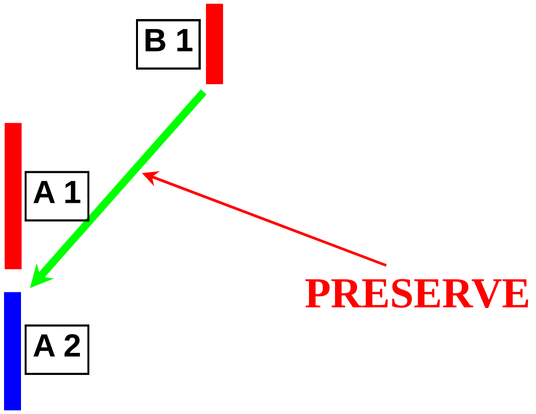




No sync ➔ May have a race

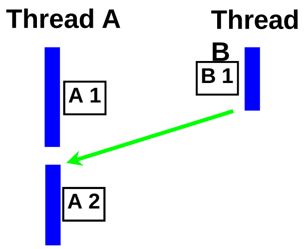
Sync ➔ No race possible

UN – FLIPPED



FLIPPED

If segments A1 and B1
have NO race ➔ they are independent ➔ NOTHING changes
have a RACE ➔ we also flipped the race ➔ Access history changes


Thread A

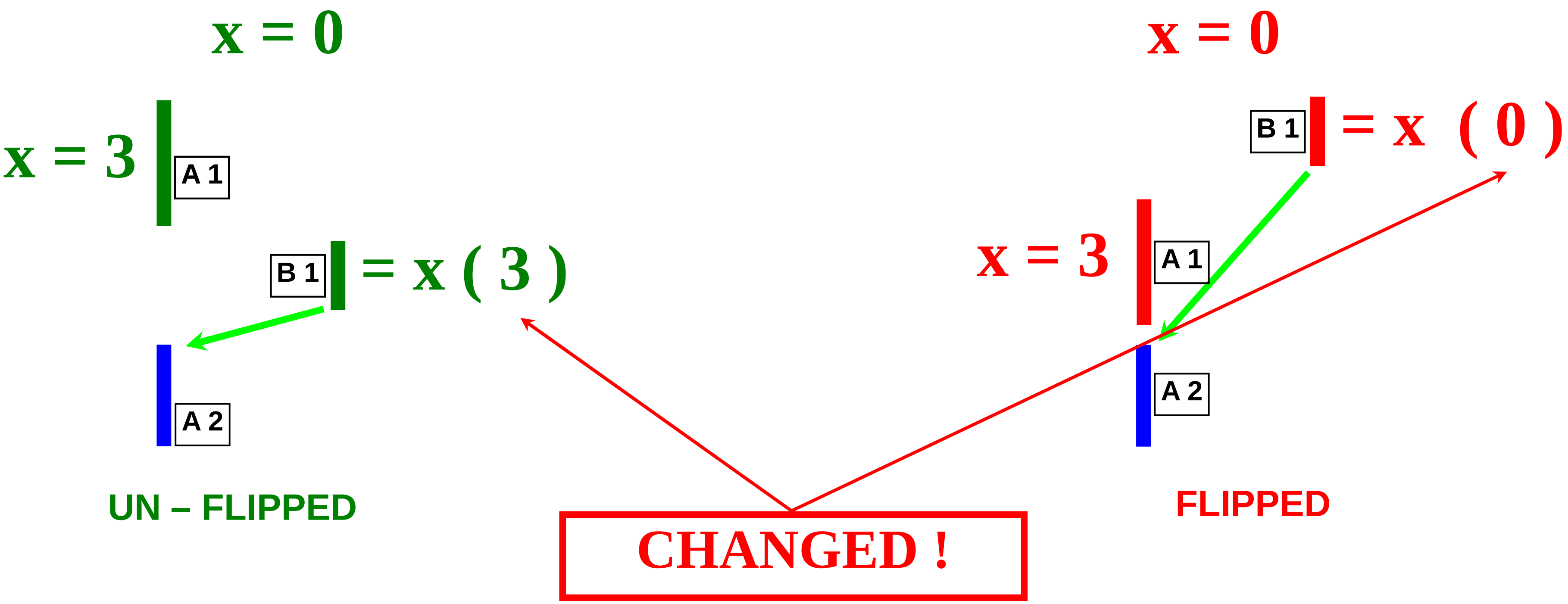
Thread B

A 1

B 1

A 2

The Idea


x = 0

x = 3

A 1

B 1

= x (3)

A 2

UN – FLIPPED

x = 0

B 1

= x (0)

x = 3

A 1

A 2

FLIPPED

CHANGED !

If segments A1 and B1

have NO race ➔ they are independent ➔ NOTHING changes

have a RACE ➔ we also flipped the race ➔ SOMETHING changes

Overview

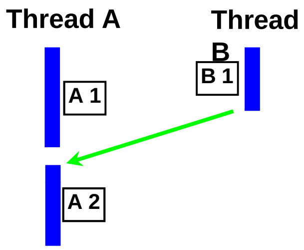
- Use two different executions
- Same synchronization order
  - If change ➔ know for sure there is a race
  - No change ➔ highly probable no race

Phases in **Light64**

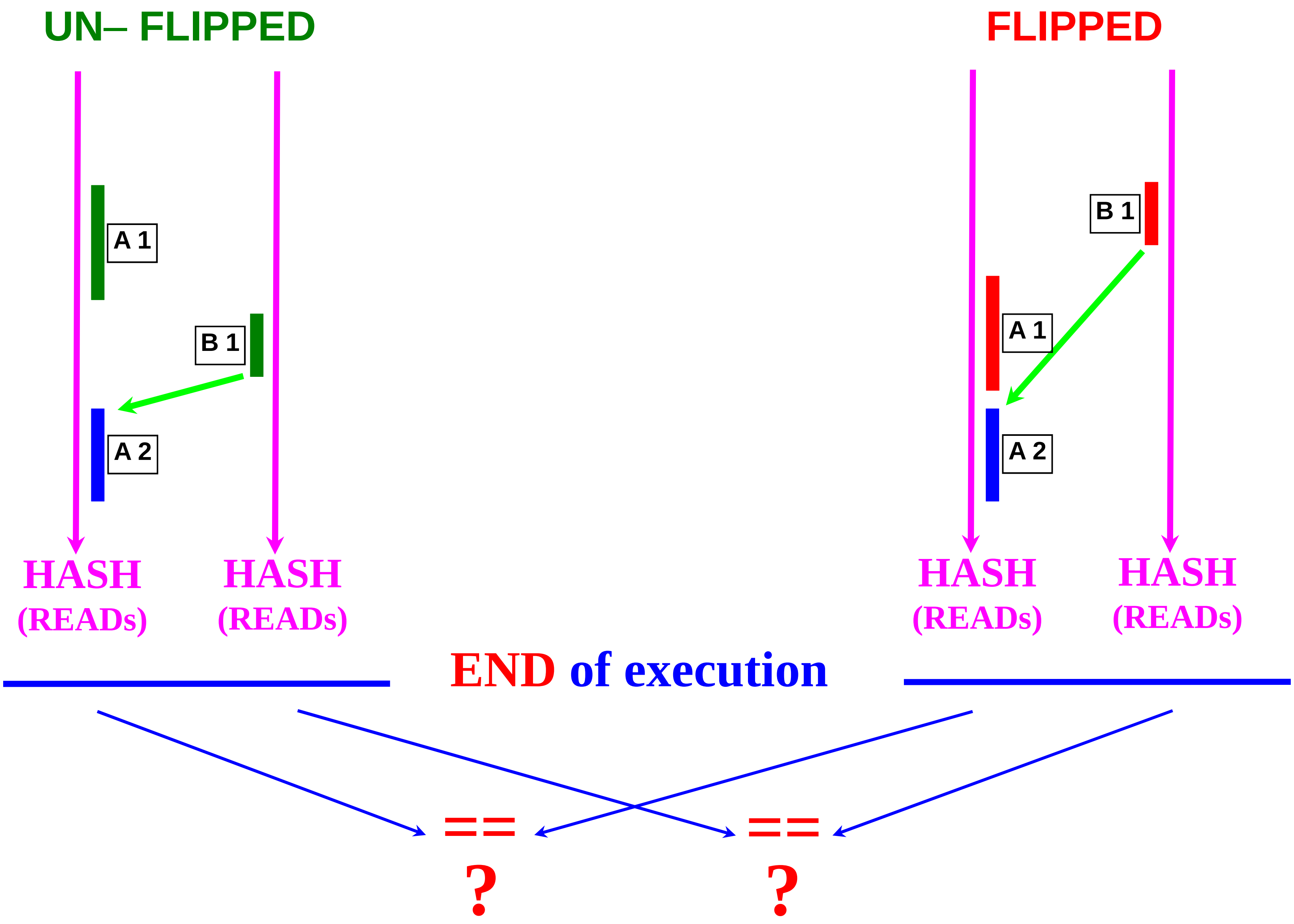
- Detect if races exist
  - Fast, over all thread interleavings executed by the tester
  - Issues
    - How to detect deviations (e.g. from 0 to 3) ➔ HW hash
    - How to flip the segments with low overhead ➔ SW
- Pin – point races
  - Slow, classic data race detection algorithm
  - Only if there are races
  - Only for the racy interleavings
  - Optimization: only for selected racy interleavings

Detecting Deviations

- Per thread: hash all the values read from memory on-the-fly
- Compare hashes of two executions with same sync order
  - Different hashes ➔ know for sure there is a race
  - Identical hashes ➔ high probability no race

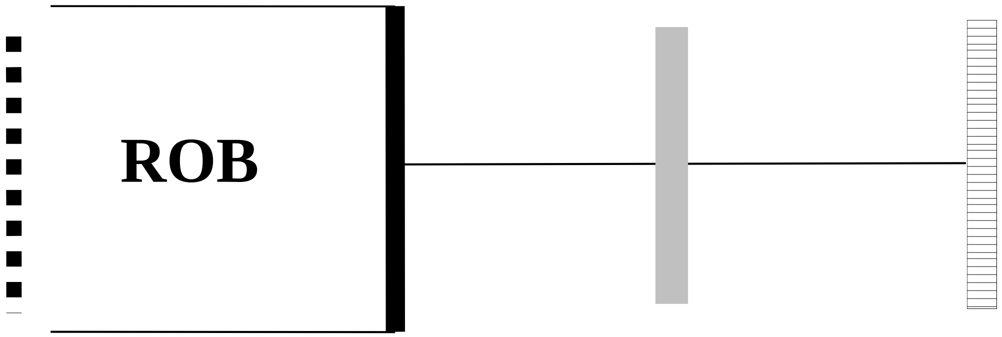


Example: Detecting Deviations



HW System

BONUS



CRC 64
hash logic

ROB

Head of
ROB

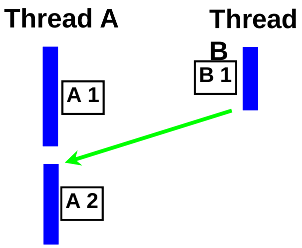
REGISTER

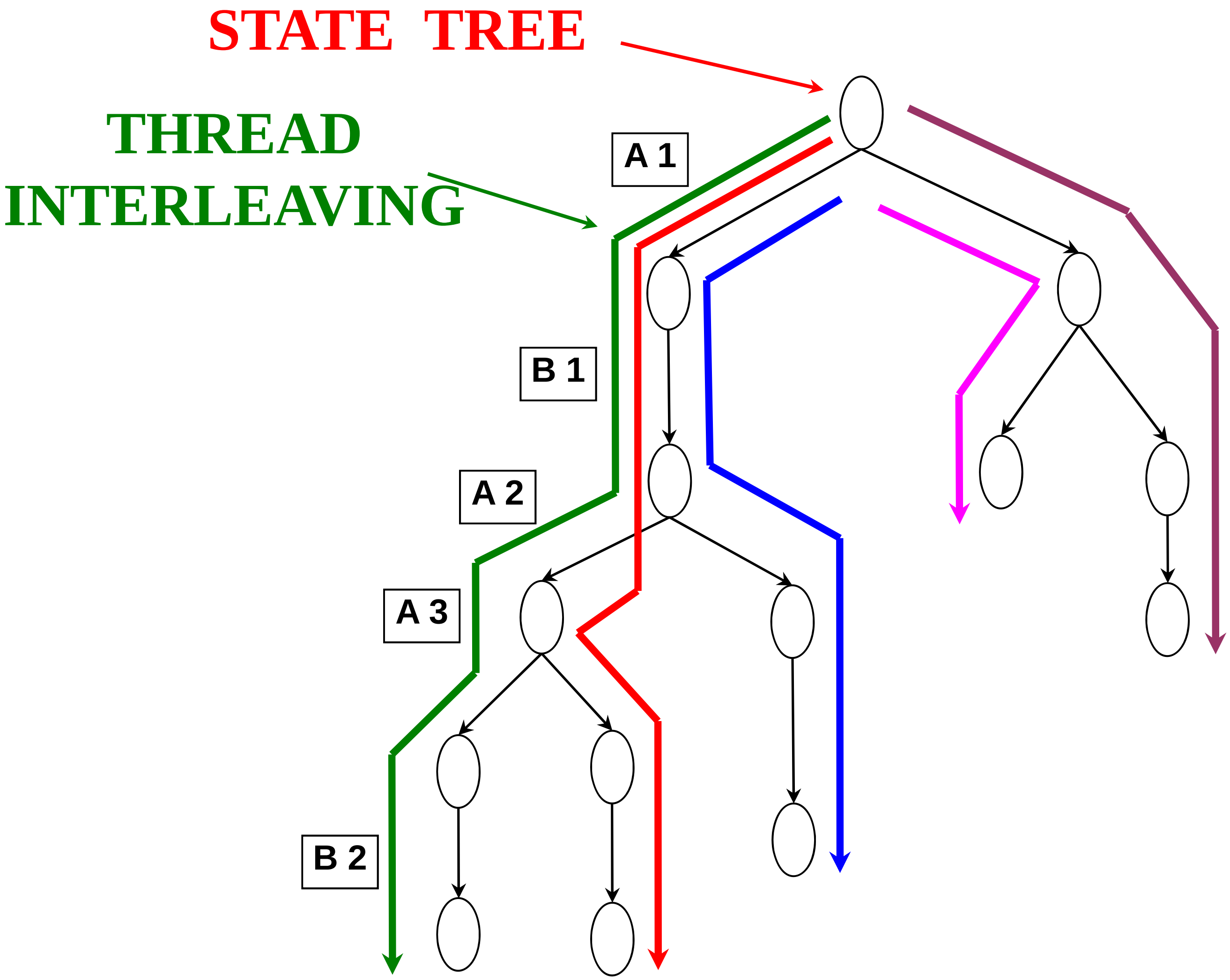
↓

virtualize
migrate
context switch
no cache spills

64 bit register
Accumulates values read from memory

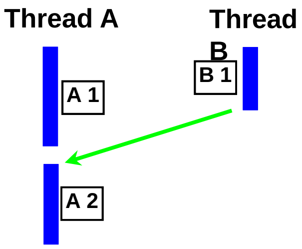
Flip with Low Overhead

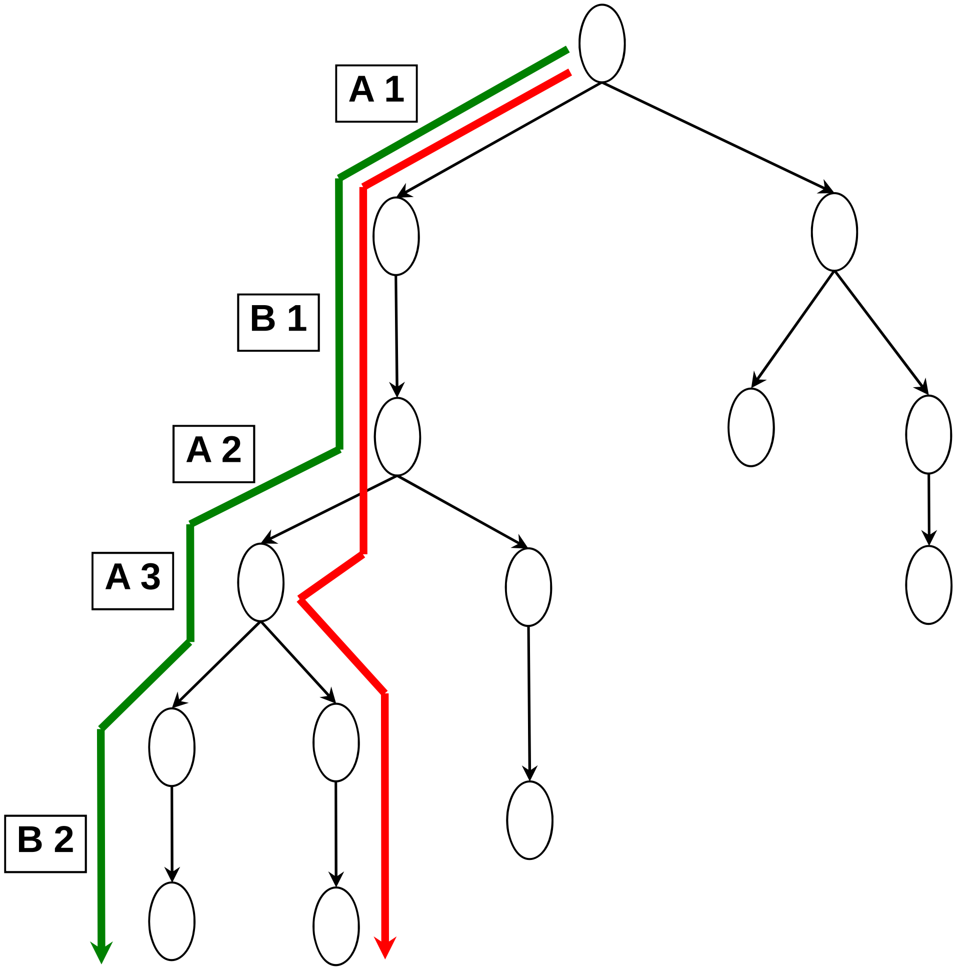



STATE TREE

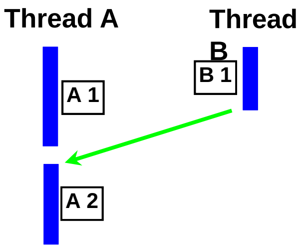
THREAD INTERLEAVING

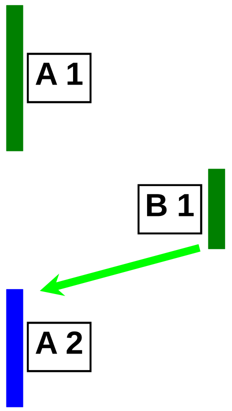
Flip with Low Overhead



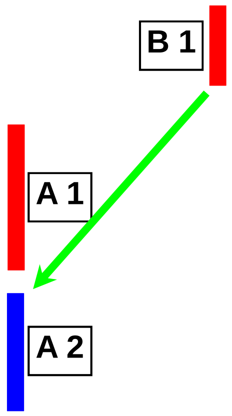


Flip with Low Overhead

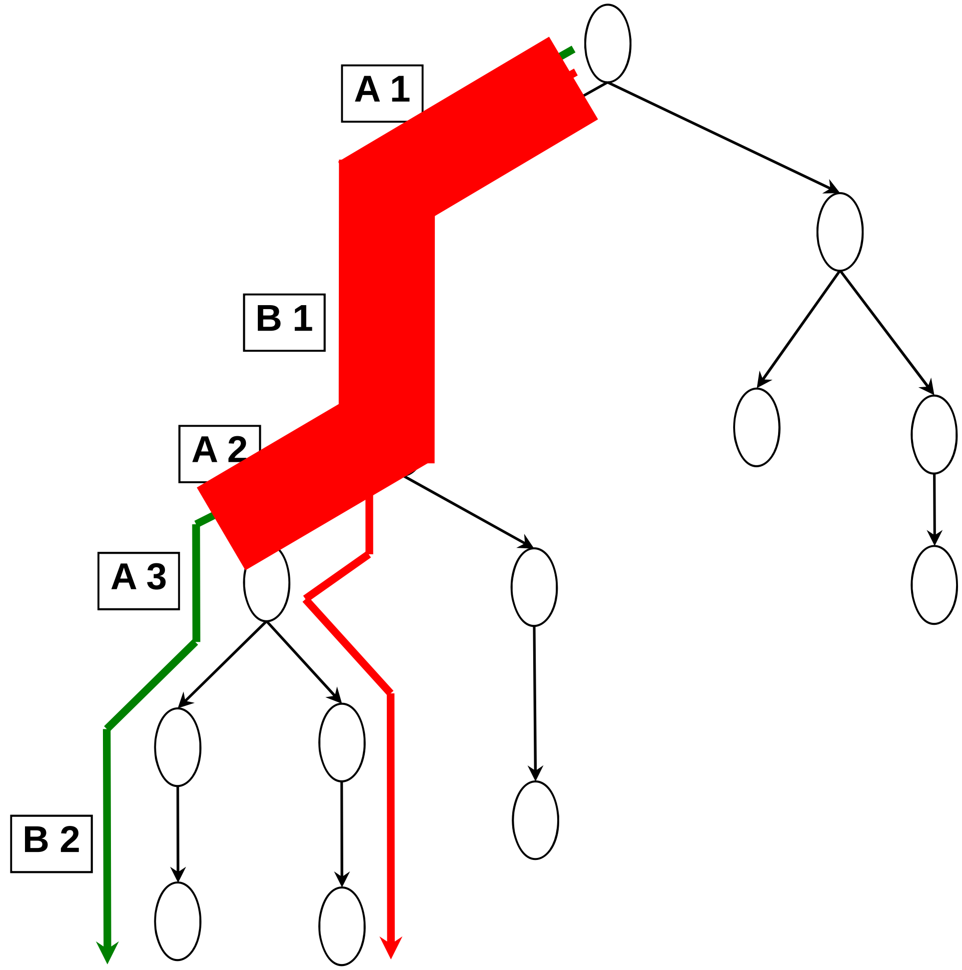




UN– FLIPPED



FLIPPED



Piggy–back on Systematic Testing primitives to reduce overhead

Some synchronization orders are executed multiple times

Outline

Motivation

Systematic Testing

Light64

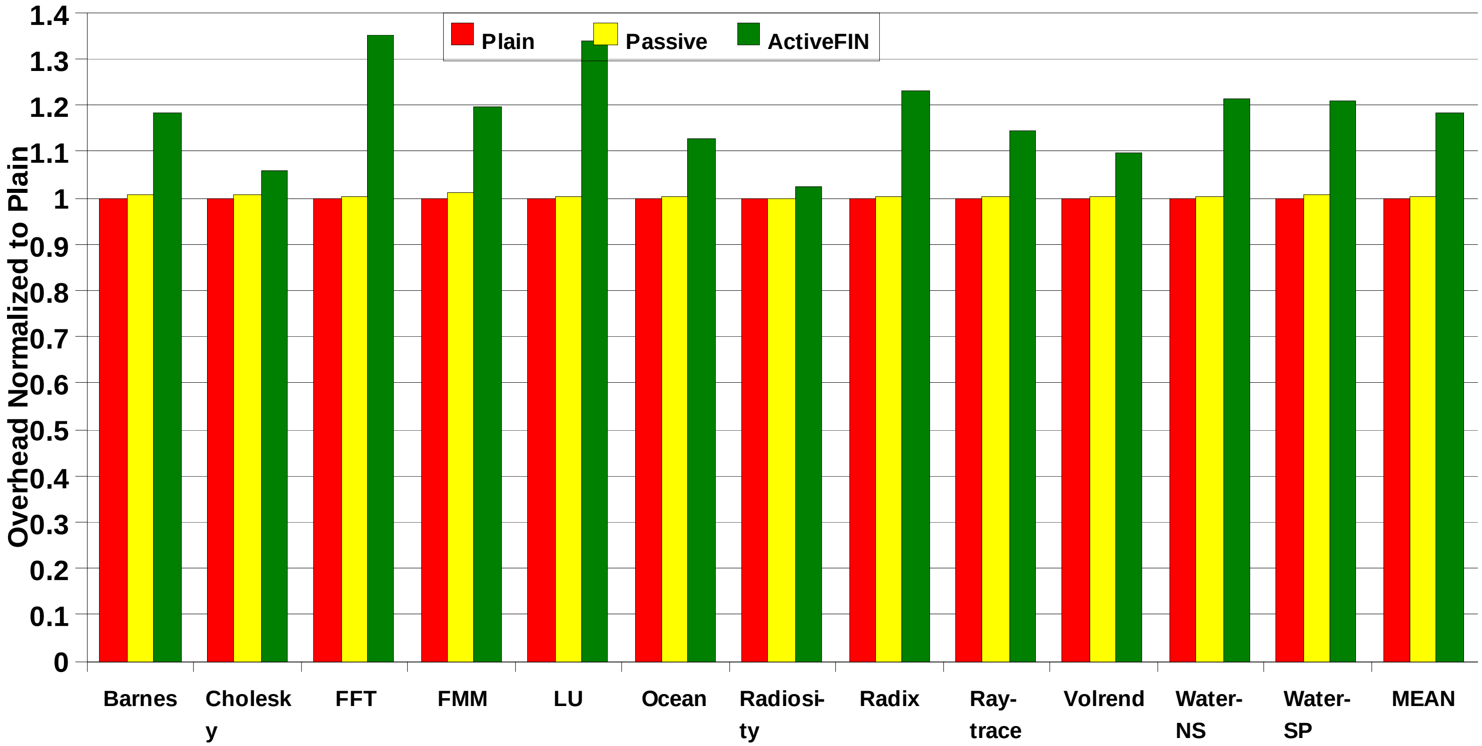
Evaluation

Conclusion

Experimental Setup

- Developed systematic tester in the lines of CHESS
- Tested all SPLASH-2 applications
- Run with 2 and 4 threads
- Execution overhead
  - Compare to a systematic tester with no race detection: **Plain**
- Accuracy
  - Compare to a systematic tester running a SW precise race detector
- Propose two Light64 versions:
  - **Active:** Aggressive flipping for high coverage
  - **Passive:** Modest flipping for minimum overhead

Execution Overhead (4 threads)



- Tradeoff execution overhead vs detection accuracy
  - Active: 37% overhead, 96% races detected
  - Passive: 2 % overhead, 89% races detected
  - SW only: 8 X overhead, 100% races detected

Detection Accuracy (4 threads)

	Original		Inserted Races	
	Light64	Precise SW	Light64	Precise SW
Barnes	311	311	192	192
Cholesky	4	4	14	14
FFT	0	0	42688	42688
FMM	0	0	40	40
LU	0	0	15286	15286
Ocean	2	2	45	45
Radiosity	12	12	16	16
Radix	0	0	15	16
Raytrace	7	7	87	88
Volrend	44	44	30	43
Water-NS	0	0	69	69
Water-SP	0	0	162	162

Average race detection accuracy: 96 %

Also in the Paper

- Additional Light64 versions
- Optimization for the phase that pin-points races
- Characterization of the systematic testing
- Overhead and accuracy for two-threaded runs
- Additional results
- Software-only implementation

Outline

Motivation

Systematic Testing

Light64

Evaluation

Conclusion

Conclusion

- Introduced Light64, new data race detection technique for use during systematic testing

	Light64	Other HW
HW	64 **bits**	72 â€400 **Kbits**
Virtualize	✓	**NO** or **replicate** the **HW**
Cache spill	✓	✗ **except one**
Context switch	✓	✗
Migration	✓	**NO** or **additional HW**
No False Positives	✓	✓
Few False Negatives	✓	✓
Runtime Overhead	**2 â€37%**	**0.5%**

Light64: Lightweight hardware support for data race detection during systematic testing

Adrian Nistor, Darko Marinov, Josep Torrellas

University of Illinois, Urbana – Champaign
http://iacoma.cs.uiuc.edu