Weak Memory Behaviors in GPUs Applications

Tyler Sorensen

Supervisors: Alastair F. Donaldson and James Brotherston

15 July 2015

Imperial Concurrency Workshop

Overview

- Current techniques for reasoning about GPU applications under weak memory models are limited to hand analysis

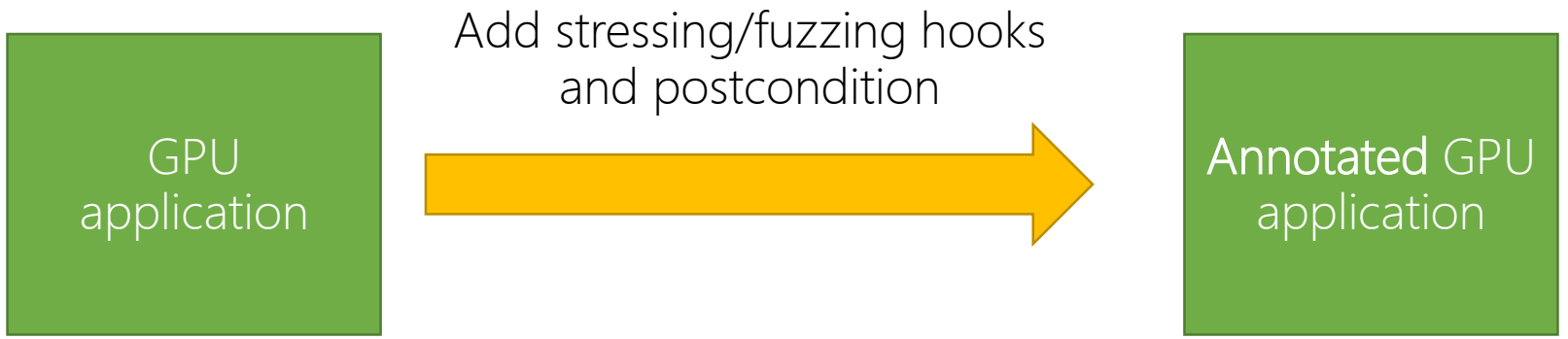
Overview

- Current techniques for reasoning about GPU applications under weak memory models are limited to hand analysis
  - This is laborious, error prone, requires a formal model

Overview

- Current techniques for reasoning about GPU applications under weak memory models are limited to hand analysis
  - This is laborious, error prone, requires a formal model
- We propose a new methodology based on stress and fuzz testing

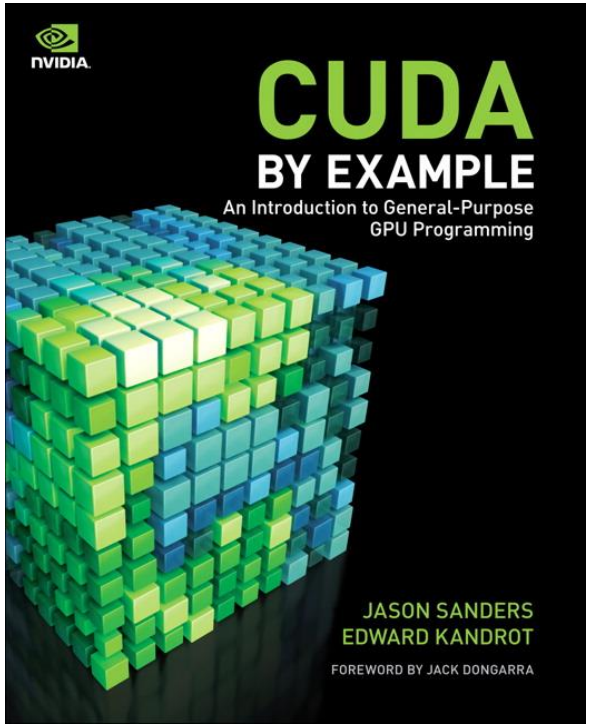
Overview



- Run annotated application for many iterations and check for postcondition violations.

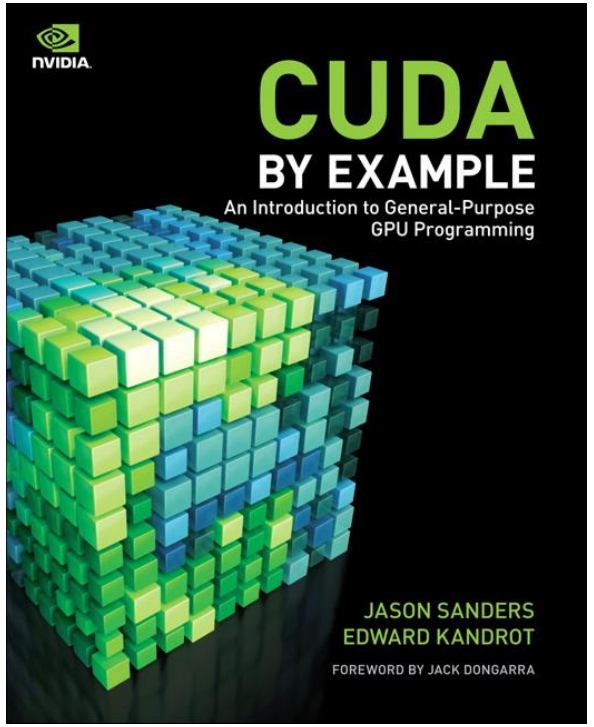
Overview

- Buggy dot product routine



Overview

- Buggy dot product routine

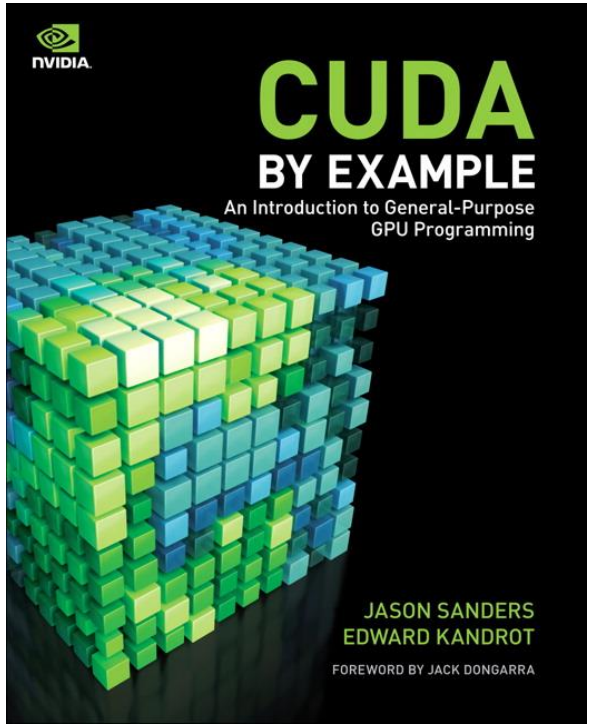


- Running the program for 1 hour (~2 seconds per run) the number of failed postconditions are:

No stress	Stress/fuzzing
0	

Overview

- Buggy dot product routine



- Running the program for 1 hour (~2 seconds per run) the number of failed postconditions are:

No stress	Stress/fuzzing
0	396

Roadmap

- Background
- Stress testing details
- Results

Weak memory models

- consider the test known as *message passing* (MP)

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Weak memory models

- consider the test known as *message passing* (MP)

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Weak memory models

- consider the test known as *message passing* (MP)

initial state: $x = 0, y = 0$	
Thread 0 a: $x \leftarrow 1$; b: $y \leftarrow 1$;	Thread 1 c: $r1 \leftarrow y$; d: $r2 \leftarrow x$;
assert: $r1 = 1 \wedge r2 = 0$	

Weak memory models

- consider the test known as *message passing* (MP)

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Message passing (MP) test

- Tests how to implement a handshake idiom

initial state: x = 0, y = 0	
Thread 0	Thread 1
a: x ← 1;	c: r1 ← y;
b: y ← 1;	d: r2 ← x;
assert: r1 = 1 ∧ r2 = 0	

Data

Data

Message passing (MP) test

- Tests how to implement a handshake idiom

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;` Flag
Flag `b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Message passing (MP) test

- Tests how to implement a handshake idiom

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$ Stale Data	

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Interleaving 1

```
a: x ← 1;
b: y ← 1;
c: r1 ← y;
d: r2 ← x;
```

Final: $r1 = 1 \wedge r2 = 1$

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Interleaving 1	**Interleaving 2**
`a: x ← 1;`	`a: x ← 1;`
`b: y ← 1;`	`c: r1 ← y;`
`c: r1 ← y;`	`b: y ← 1;`
`d: r2 ← x;`	`d: r2 ← x;`
Final: $r1 = 1 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 1$

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Interleaving 1	**Interleaving 2**	**Interleaving 3**
`a: x ← 1;`	`a: x ← 1;`	`a: x ← 1;`
`b: y ← 1;`	`c: r1 ← y;`	`c: r1 ← y;`
`c: r1 ← y;`	`b: y ← 1;`	`d: r2 ← x;`
`d: r2 ← x;`	`d: r2 ← x;`	`b: y ← 1;`
Final: $r1 = 1 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 0$

initial state: x = 0, y = 0	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: r1 = 1 ∧ r2 = 0	

Interleaving 1	**Interleaving 2**	**Interleaving 3**
`a: x ← 1;`	`a: x ← 1;`	`a: x ← 1;`
`b: y ← 1;`	`c: r1 ← y;`	`c: r1 ← y;`
`c: r1 ← y;`	`b: y ← 1;`	`d: r2 ← x;`
`d: r2 ← x;`	`d: r2 ← x;`	`b: y ← 1;`
Final: r1 = 1 ∧ r2 = 1	Final: r1 = 0 ∧ r2 = 1	Final: r1 = 0 ∧ r2 = 0
Interleaving 4	**Interleaving 5**	**Interleaving 6**
`c: r1 ← y;`	`c: r1 ← y;`	`c: r1 ← y;`
`a: x ← 1;`	`a: x ← 1;`	`d: r2 ← x;`
`b: y ← 1;`	`d: r2 ← x;`	`a: x ← 1;`
`d: r2 ← x;`	`b: y ← 1;`	`b: y ← 1;`
Final: r1 = 0 ∧ r2 = 1	Final: r1 = 0 ∧ r2 = 1	Final: r1 = 0 ∧ r2 = 0

this is known as Lamport's *sequential consistency (or SC)*

initial state: $x = 0, y = 0$	
Thread 0	**Thread 1**
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

assertion cannot be satisfied by interleavings

Interleaving 1	**Interleaving 2**	**Interleaving 3**
`a: x ← 1;`	`a: x ← 1;`	`a: x ← 1;`
`b: y ← 1;`	`c: r1 ← y;`	`c: r1 ← y;`
`c: r1 ← y;`	`b: y ← 1;`	`d: r2 ← x;`
`d: r2 ← x;`	`d: r2 ← x;`	`b: y ← 1;`
Final: $r1 = 1 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 0$
Interleaving 4	**Interleaving 5**	**Interleaving 6**
`c: r1 ← y;`	`c: r1 ← y;`	`c: r1 ← y;`
`a: x ← 1;`	`a: x ← 1;`	`d: r2 ← x;`
`b: y ← 1;`	`d: r2 ← x;`	`a: x ← 1;`
`d: r2 ← x;`	`b: y ← 1;`	`b: y ← 1;`
Final: $r1 = 0 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 1$	Final: $r1 = 0 \wedge r2 = 0$

Weak memory models

- can we assume assertion will never pass?

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Weak memory models

- can we assume assertion will never pass? No!

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

Weak memory models

- Alglave et al. report this assertion passes 41 million times out of 5 billion test runs on Tegra2 ARM processor[1]

initial state: $x = 0, y = 0$	
Thread 0	Thread 1
`a: x ← 1;`	`c: r1 ← y;`
`b: y ← 1;`	`d: r2 ← x;`
assert: $r1 = 1 \wedge r2 = 0$	

[1]http://diy.inria.fr/cats/tables.html

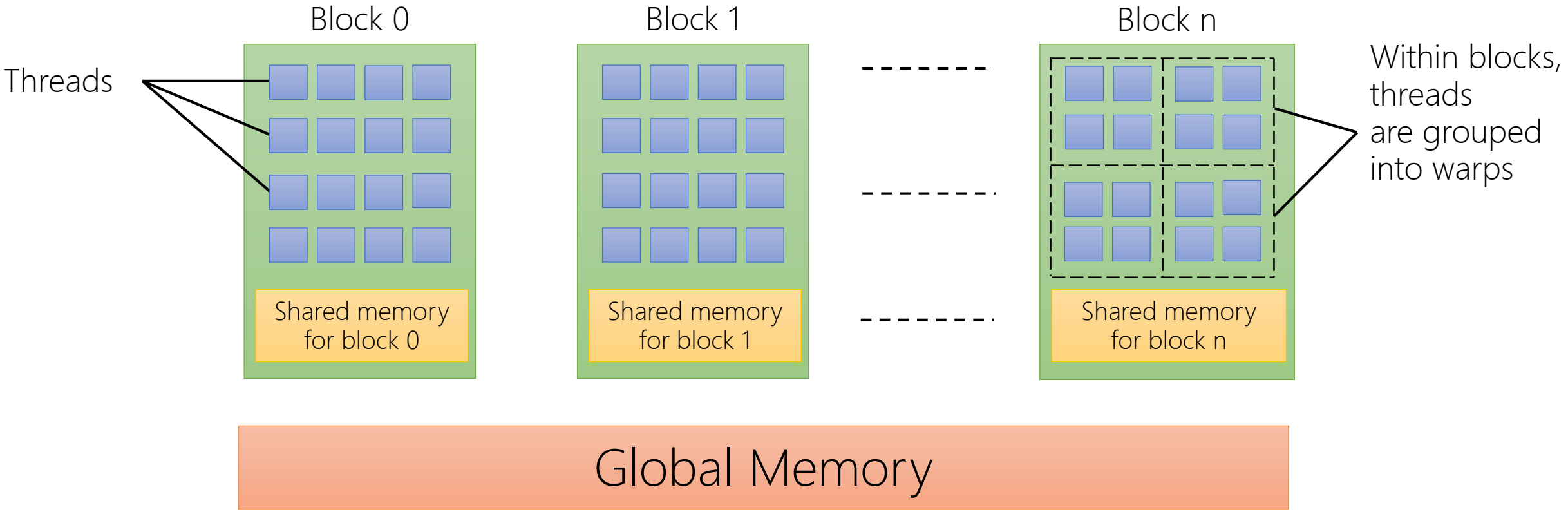
Weak memory models

- what happened?

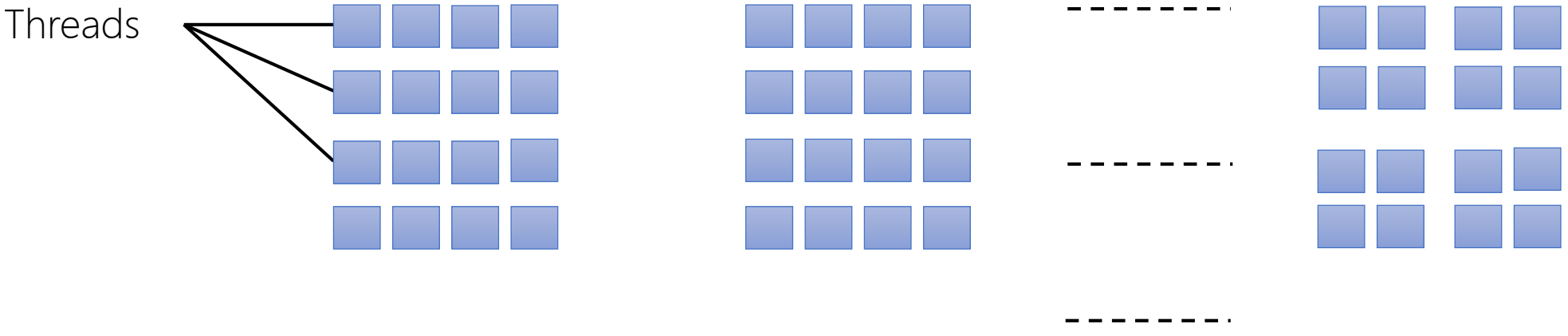
Weak memory models

- what happened?
- architectures implement *weak memory models* where the hardware is allowed to re-order certain memory instructions.
- weak memory models can allow *weak behaviors* (executions that do not correspond to an interleaving)

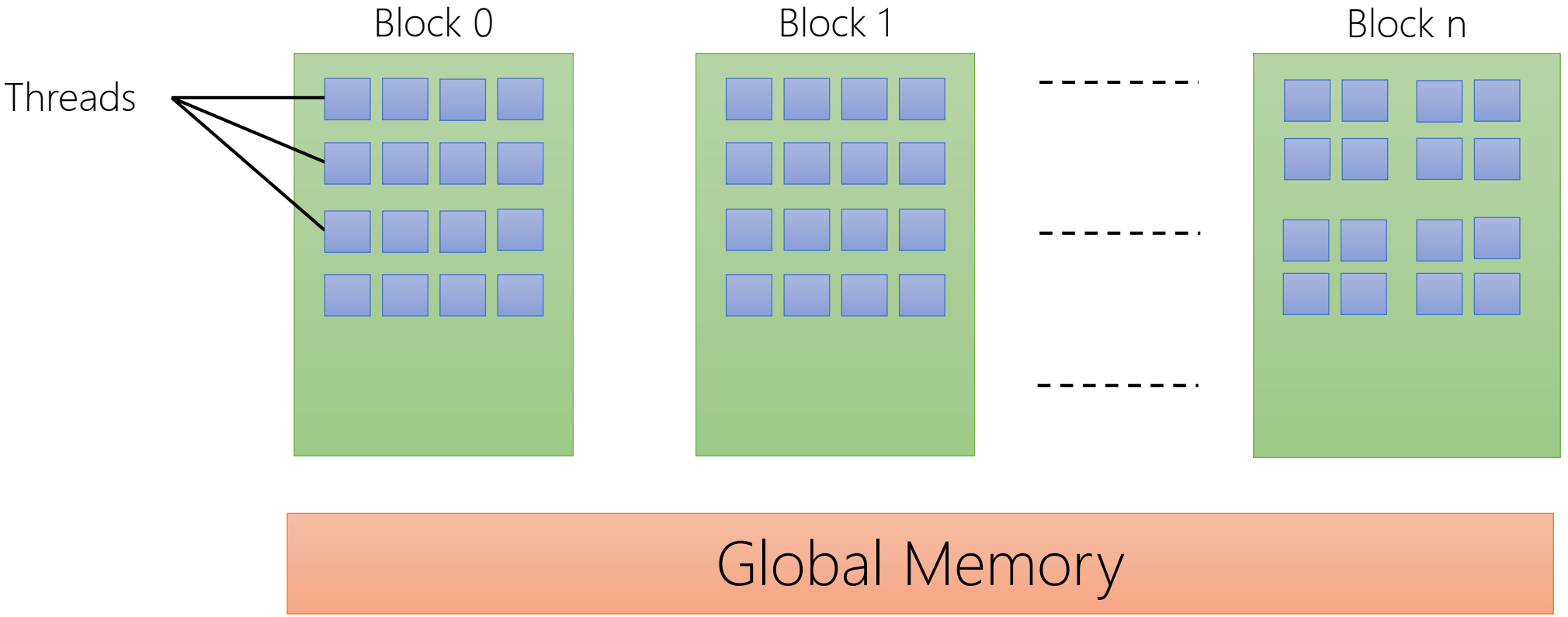
GPU programming



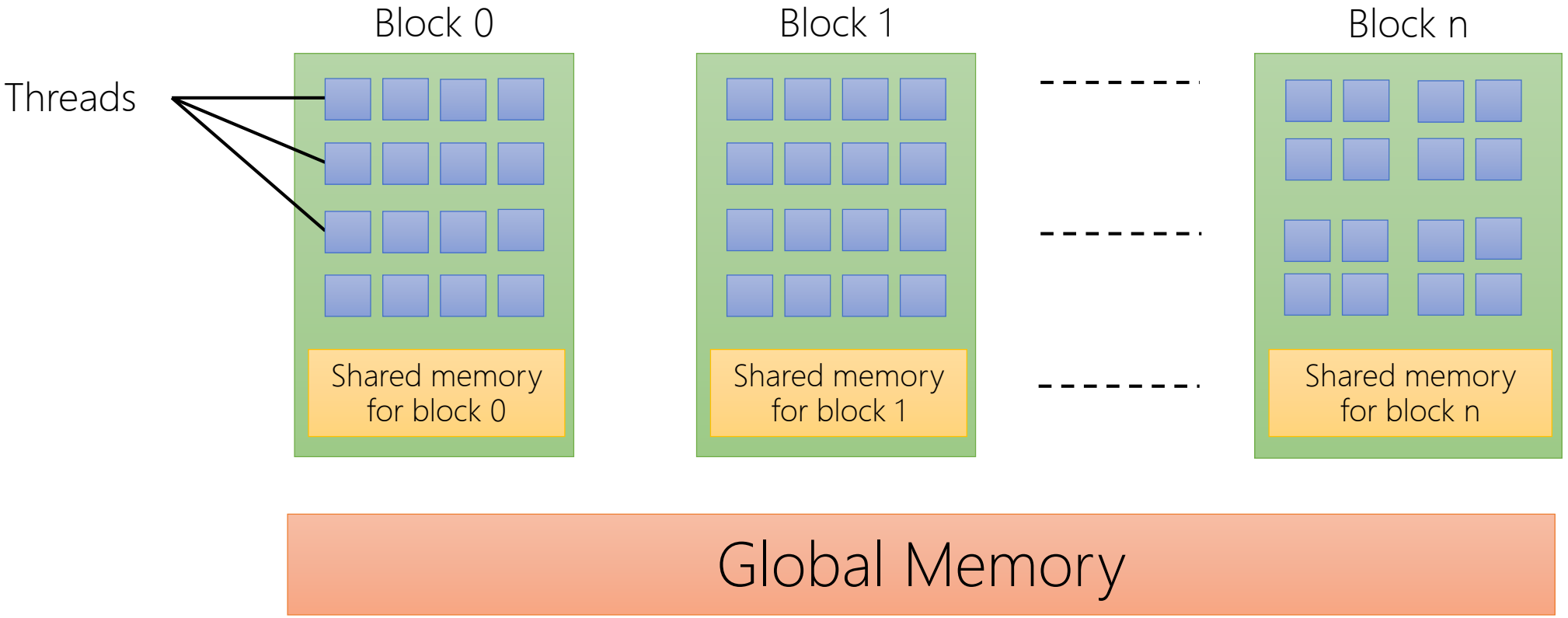
GPU programming



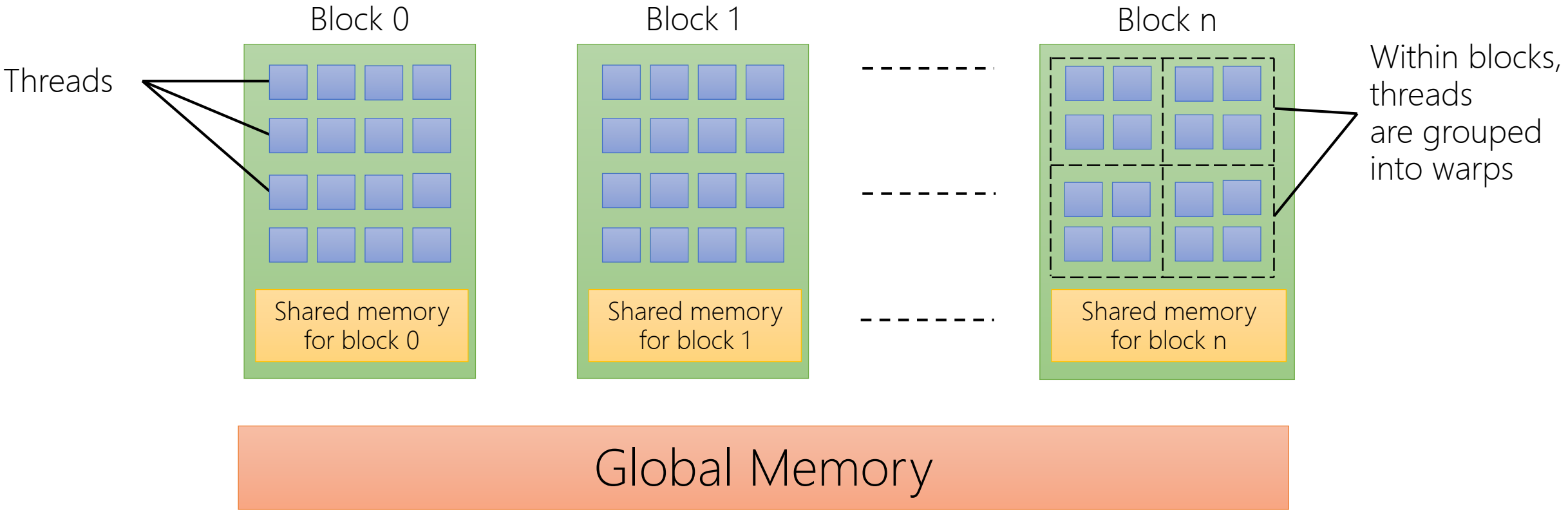
GPU programming



GPU programming



GPU programming



Roadmap

- Background
- Stress testing details
- Results

GPU memory models

- Previous work[1] showed that GPUs empirically have weak memory models.
- Done using a tool which ran litmus tests on GPUs
- Required heuristics for weak behaviors to appear

[1]GPU concurrency: Weak behaviours and programming assumptions. ASPLOS ’15.

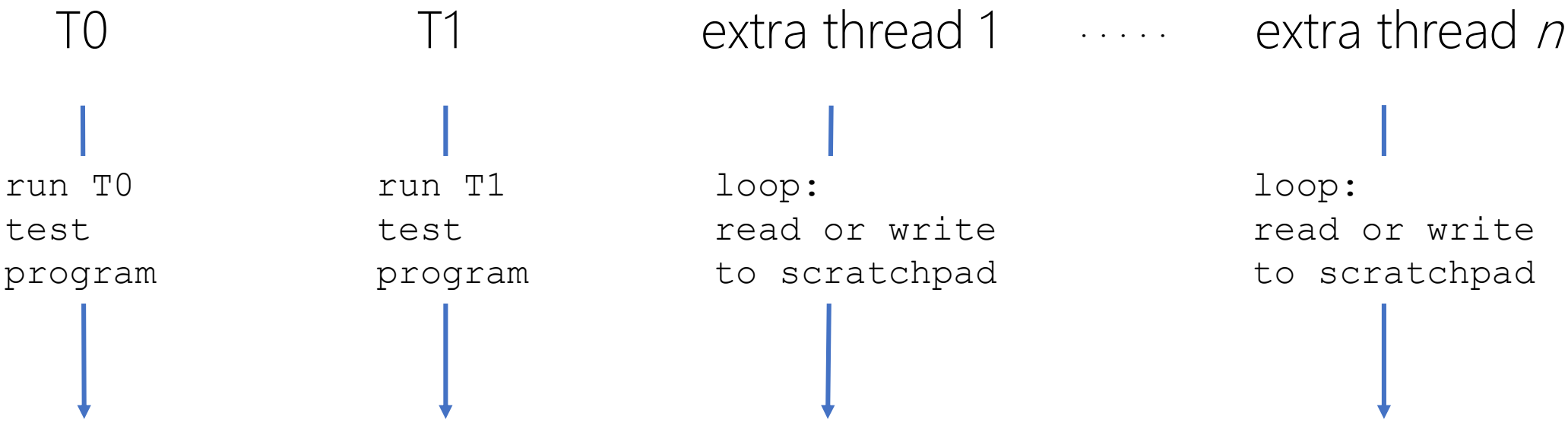
Litmus tests

Message Passing (MP)	
initial state: x = 0, y = 0	
Thread 1	Thread 2
`a: x = 1;`	`c: r1 = y;`
`b: y = 1;`	`d: r2 = x;`
weak behavior: r1 = 1 ∧ r2 = 0	

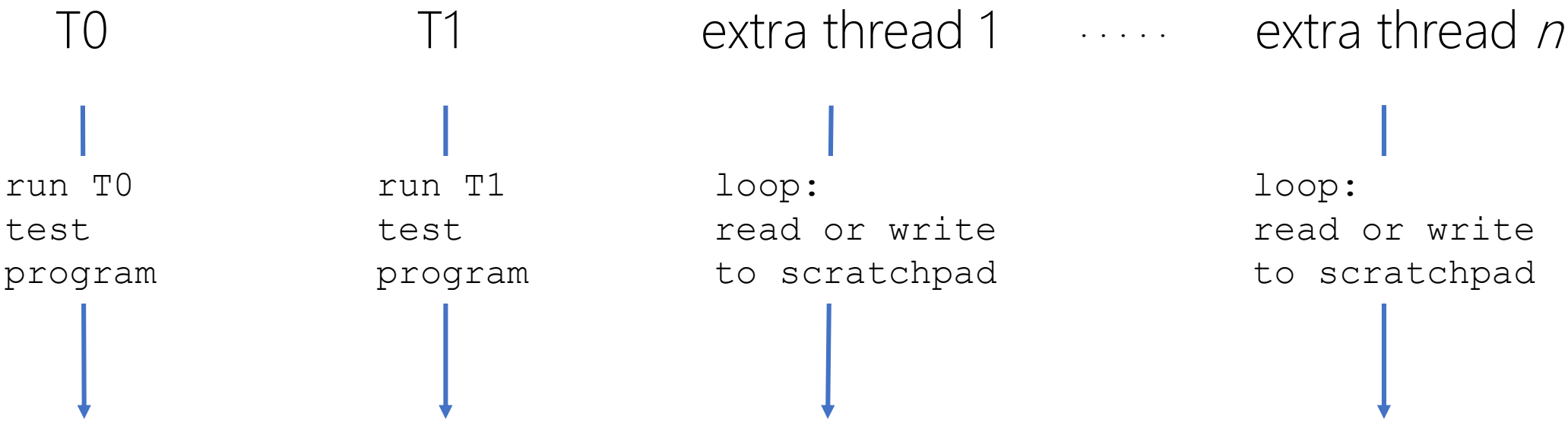
Load Buffering (LB)	
initial state: x = 0, y = 0	
Thread 1	Thread 2
`a: r1 = x;`	`c: r2 = y;`
`b: y = 1;`	`d: x = 1;`
weak behavior: r1 = 1 ∧ r2 = 1	

Store Buffering (SB)	
initial state: x = 0, y = 0	
Thread 1	Thread 2
`a: x = 1;`	`c: y = 1;`
`b: r1 = y;`	`d: r2 = x;`
weak behavior: r1 = 0 ∧ r2 = 0	

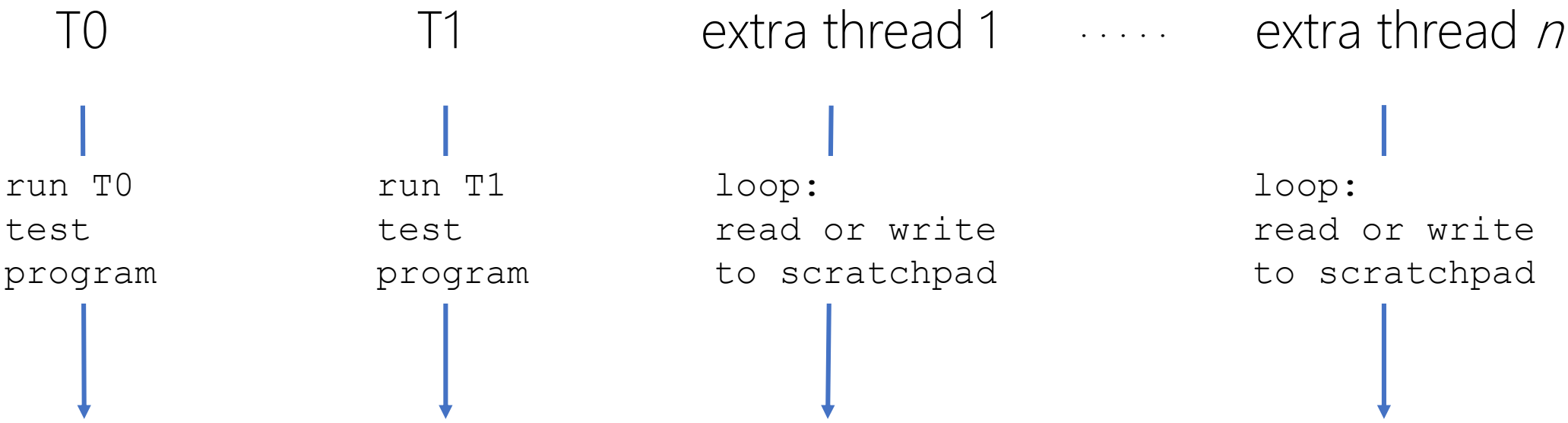
Memory stress



Memory stress

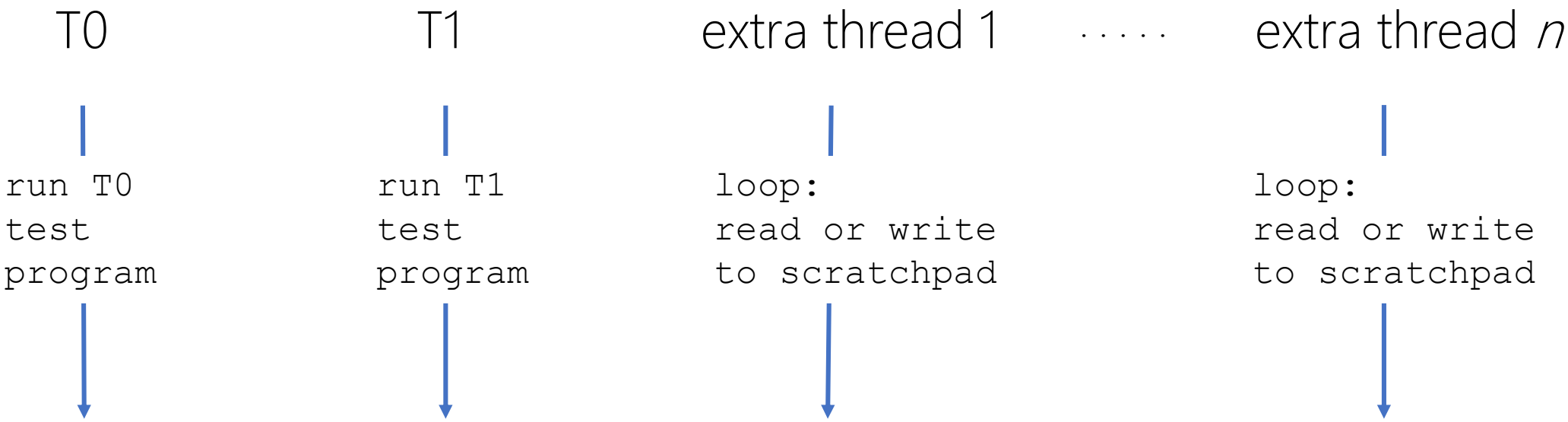


Memory stress



Memory

Memory stress

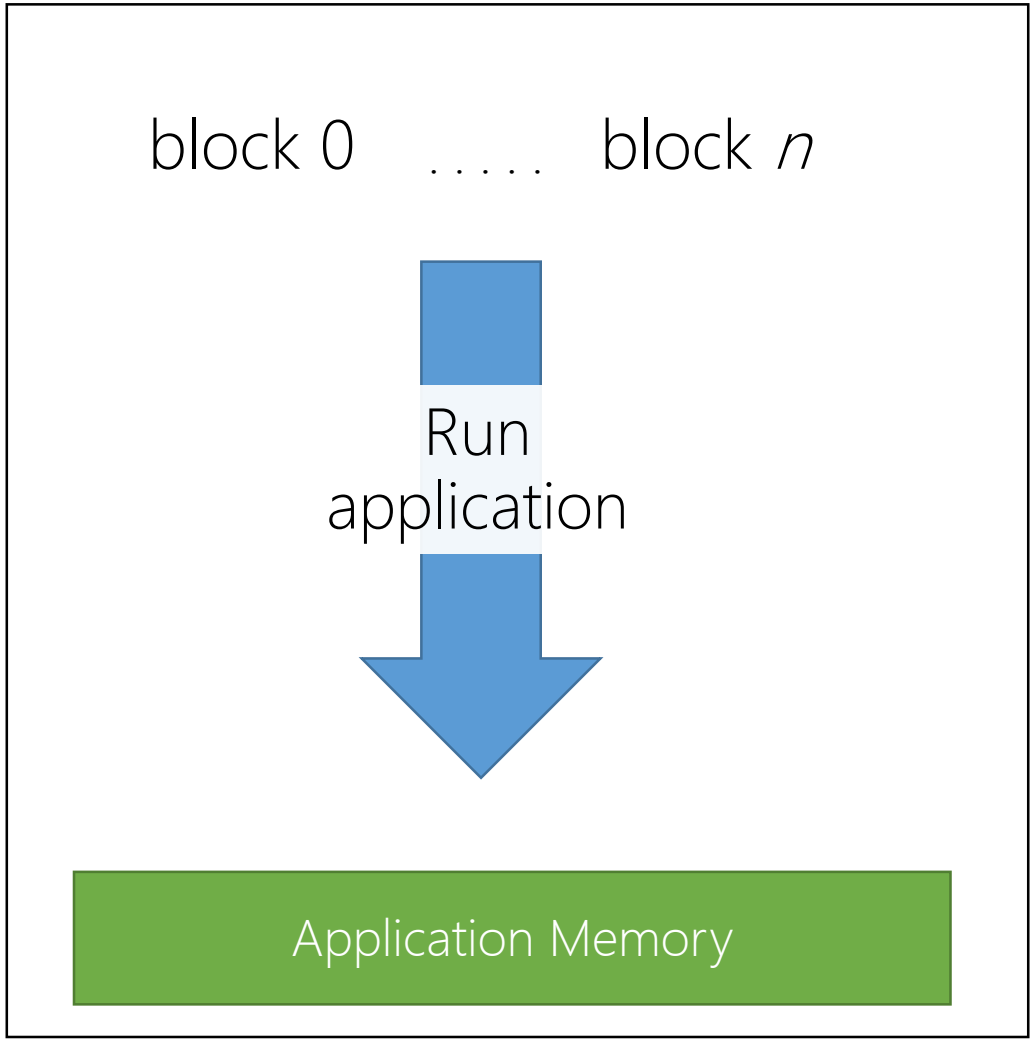


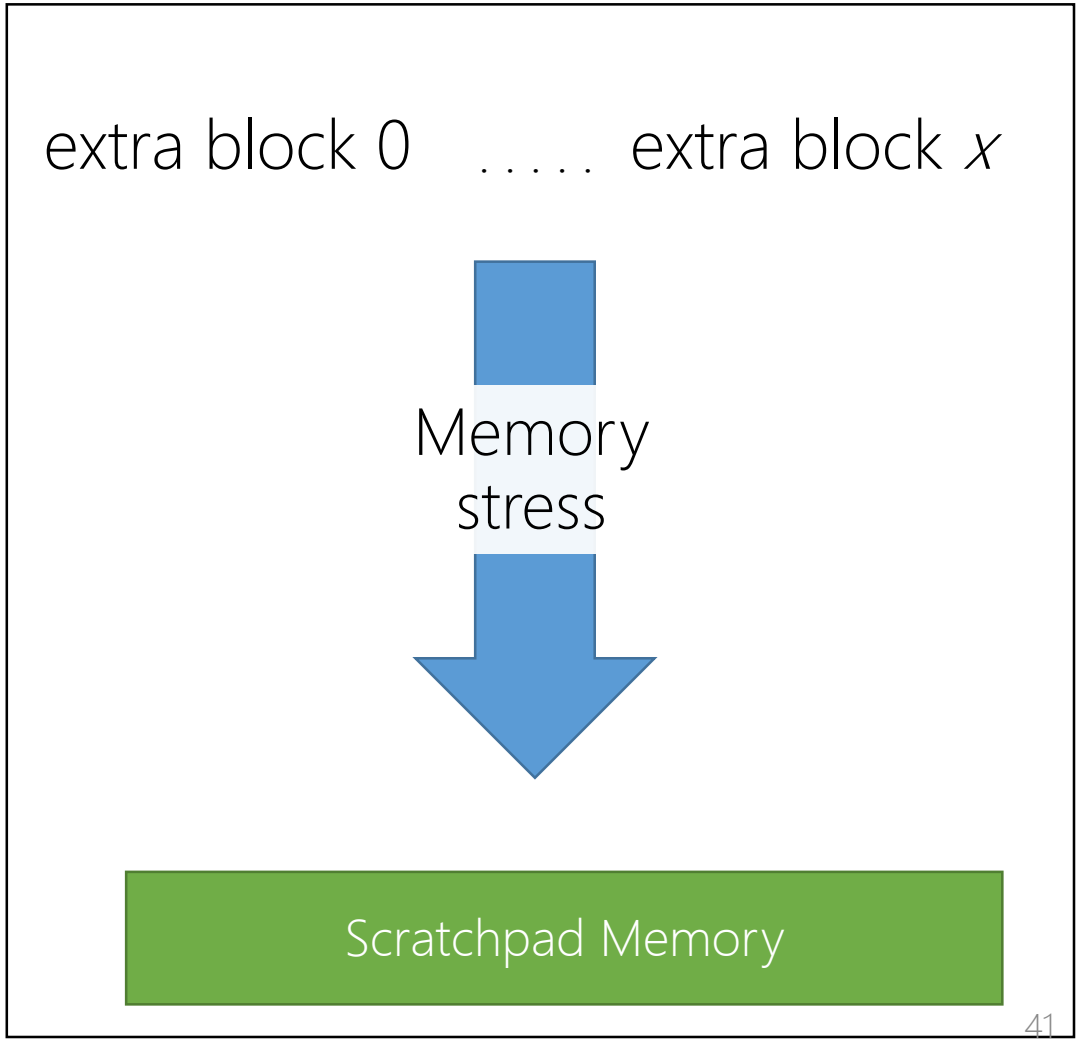
Memory

Memory stress

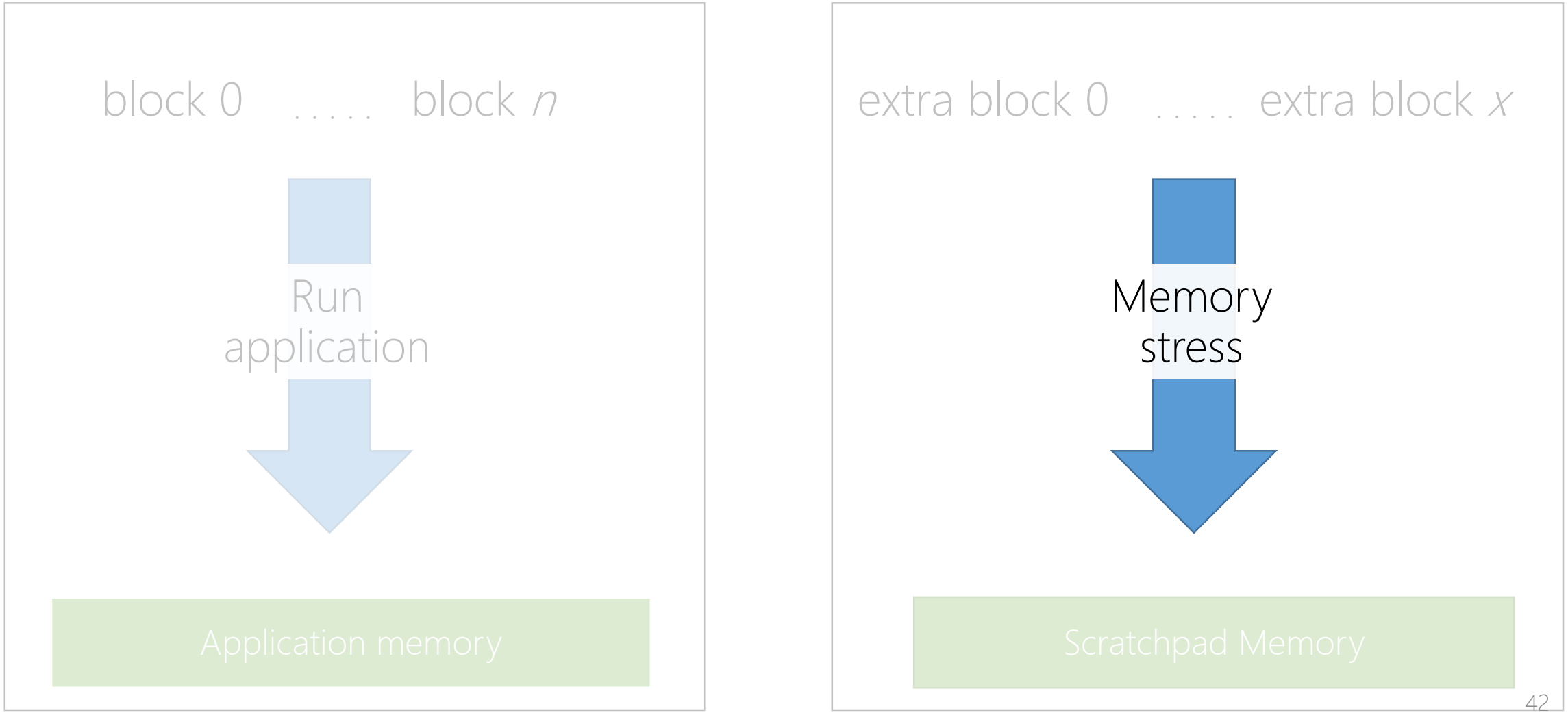
- Can we extend memory stress for testing applications?

Memory stress



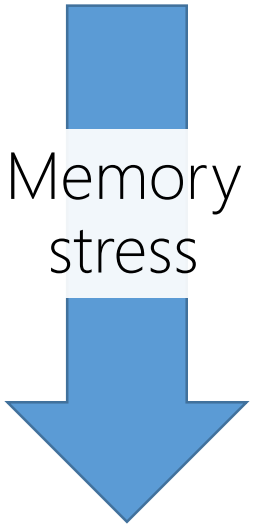


Memory stress



Memory stress

- Goal: design stress to reveal weak behaviors with no a priori knowledge about the application.
- We investigate using litmus tests, MP, SB, and LB



Memory stress

Where to stress:

Memory stress

Where to stress:

- For each distance *D*:

Memory stress

Where to stress:

- For each distance *D*:

Memory stress

Where to stress:

- For each distance *D:*

Memory stress

Where to stress:

- For each distance *D*:

Memory stress

Where to stress:

- For each distance *D*:

Memory stress

Where to stress:

- For each distance *D*:

- For each scratchpad location *l*:

Memory stress

Where to stress:

- For each distance *D*:

- For each scratchpad location *l*:

Memory stress

Where to stress:

- For each distance *D*:

- For each scratchpad location *l*:

Memory stress

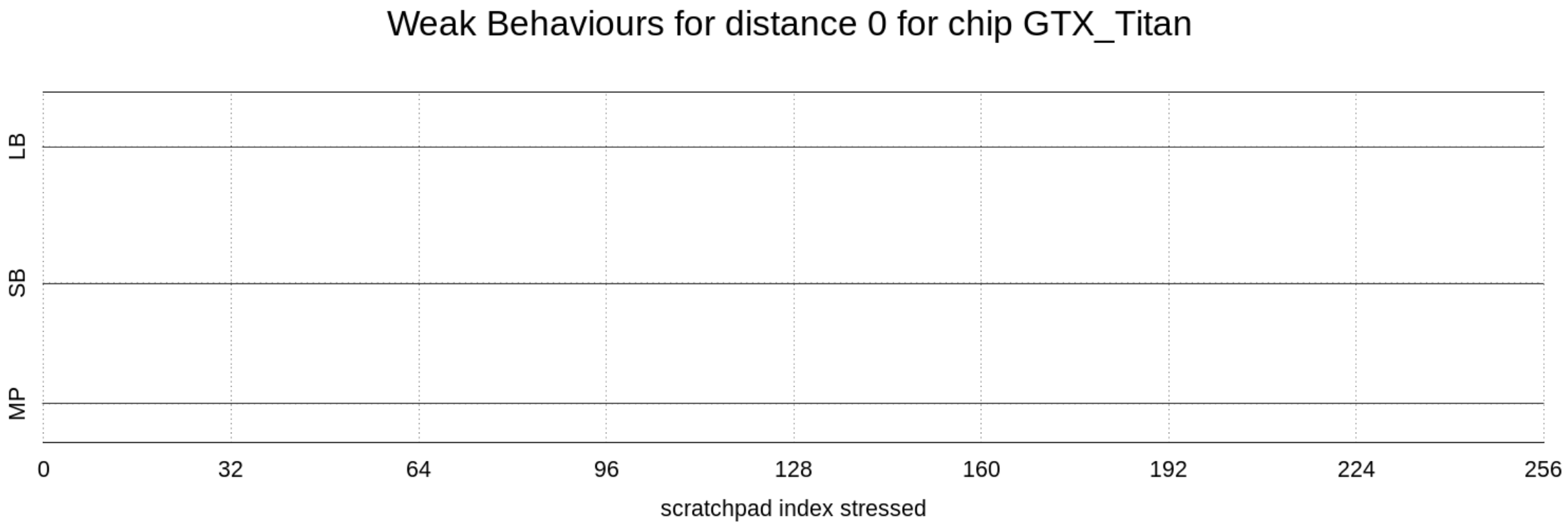
Where to stress:

- For each distance *D*:

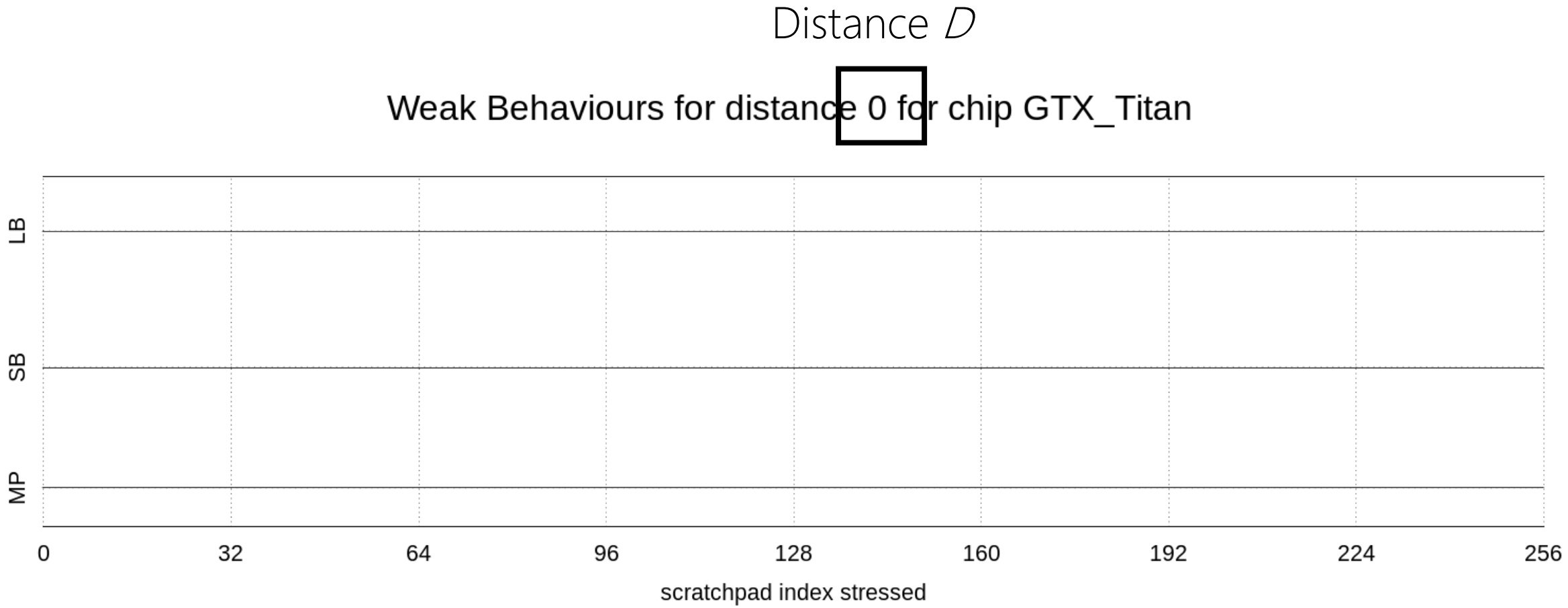
- For each scratchpad location *l*:

- Run MP, SB, LB at distance *D* litmus tests stressing only location *l* for 1000 iterations

Memory stress

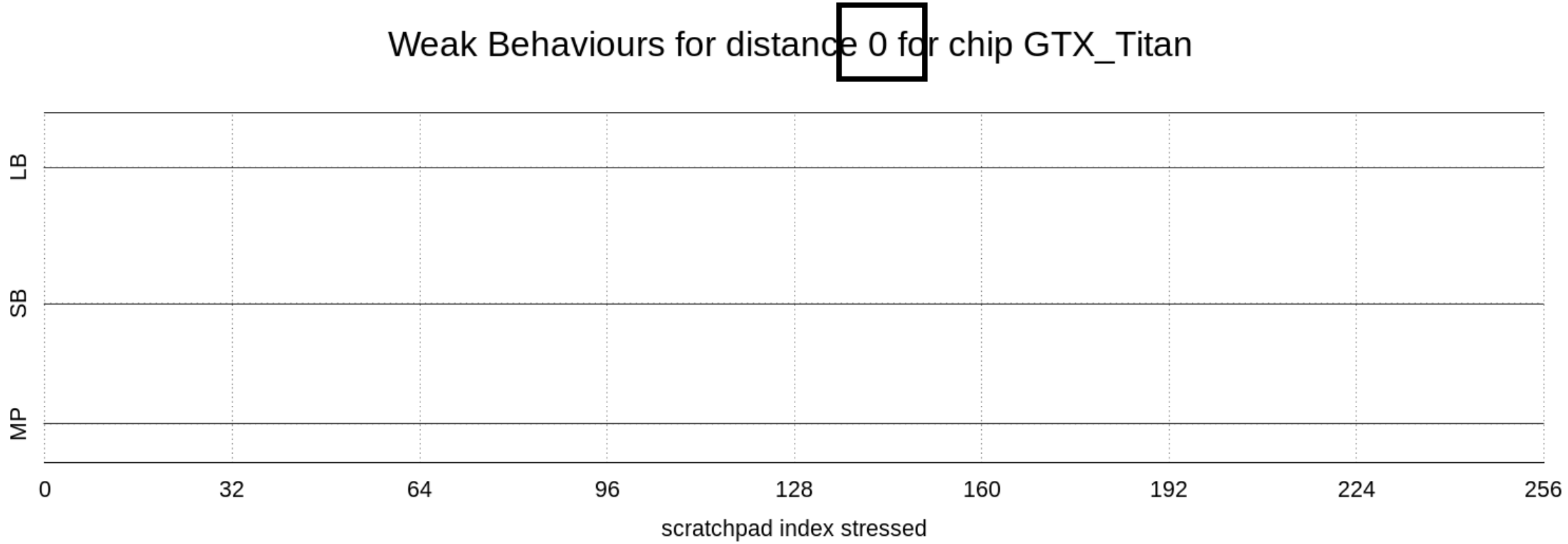


Memory stress



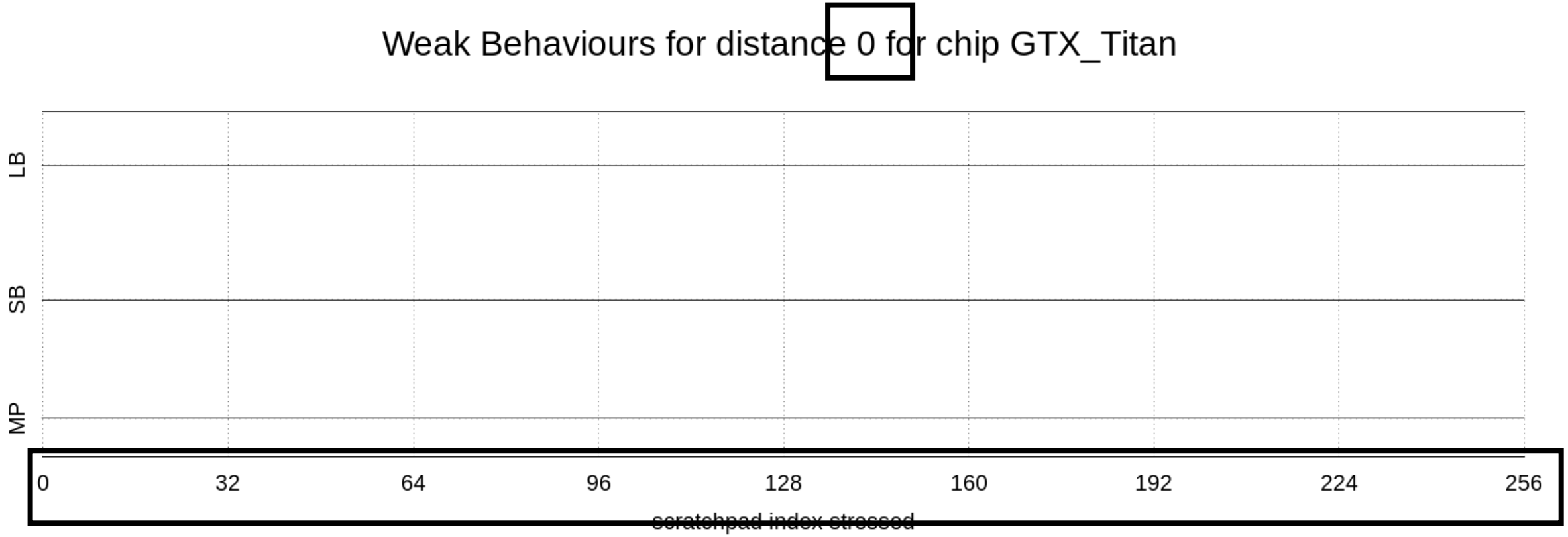
Memory stress

Distance *D*



Memory stress

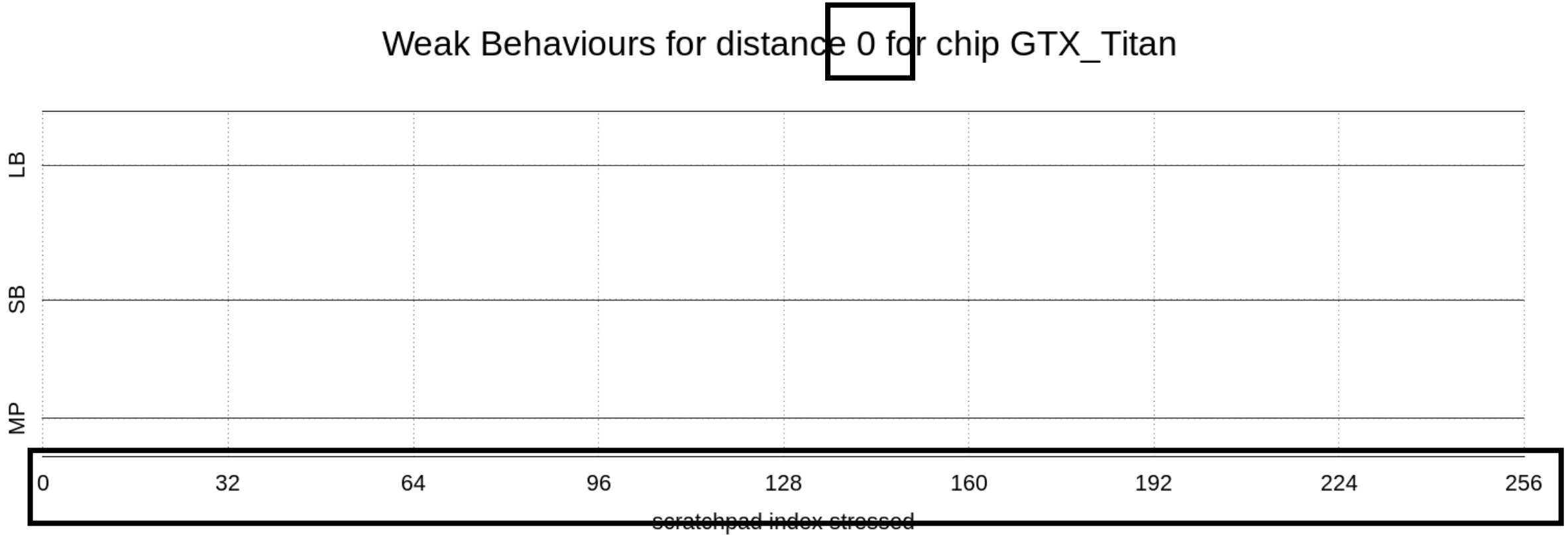
Distance *D*



Index / stressed

Memory stress

Distance *D*

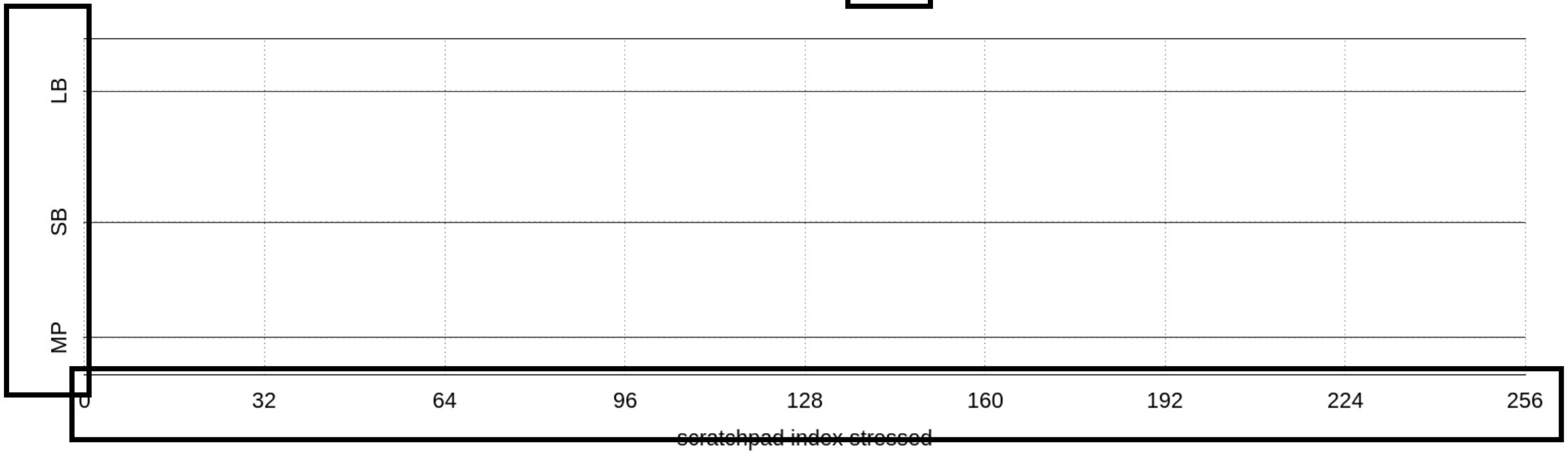


Index / stressed

Memory stress

Distance *D*

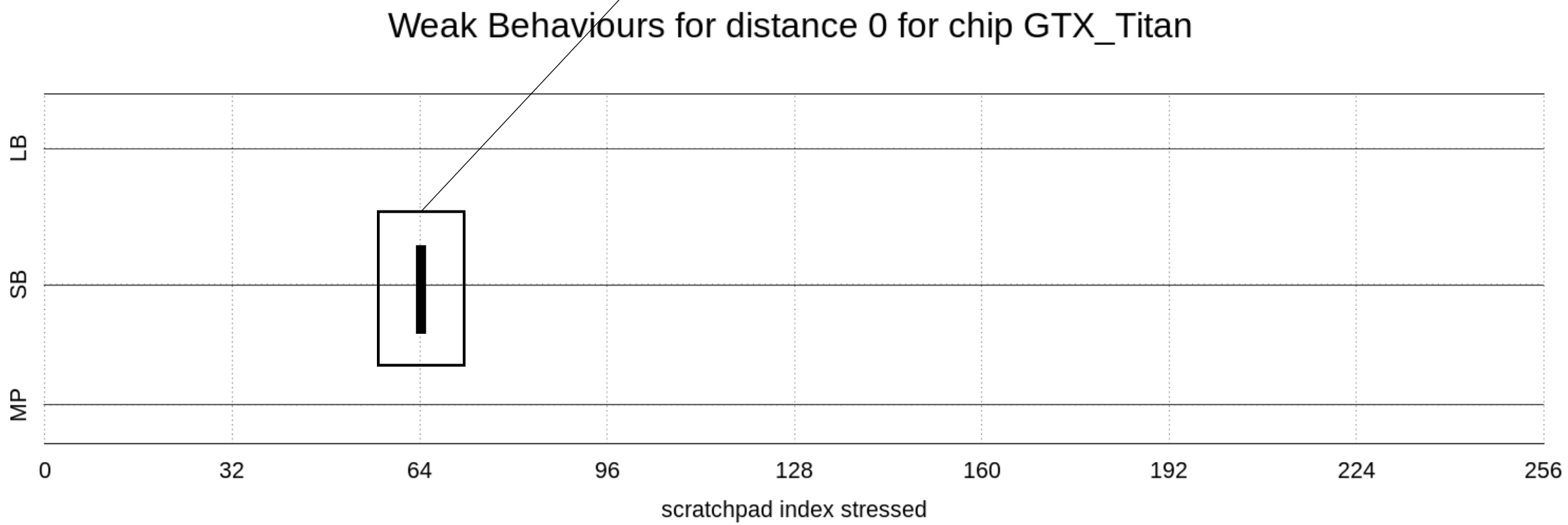
Litmus test



Index / stressed

Memory stress

Vertical bar represents the magnitude of weak behaviors observed

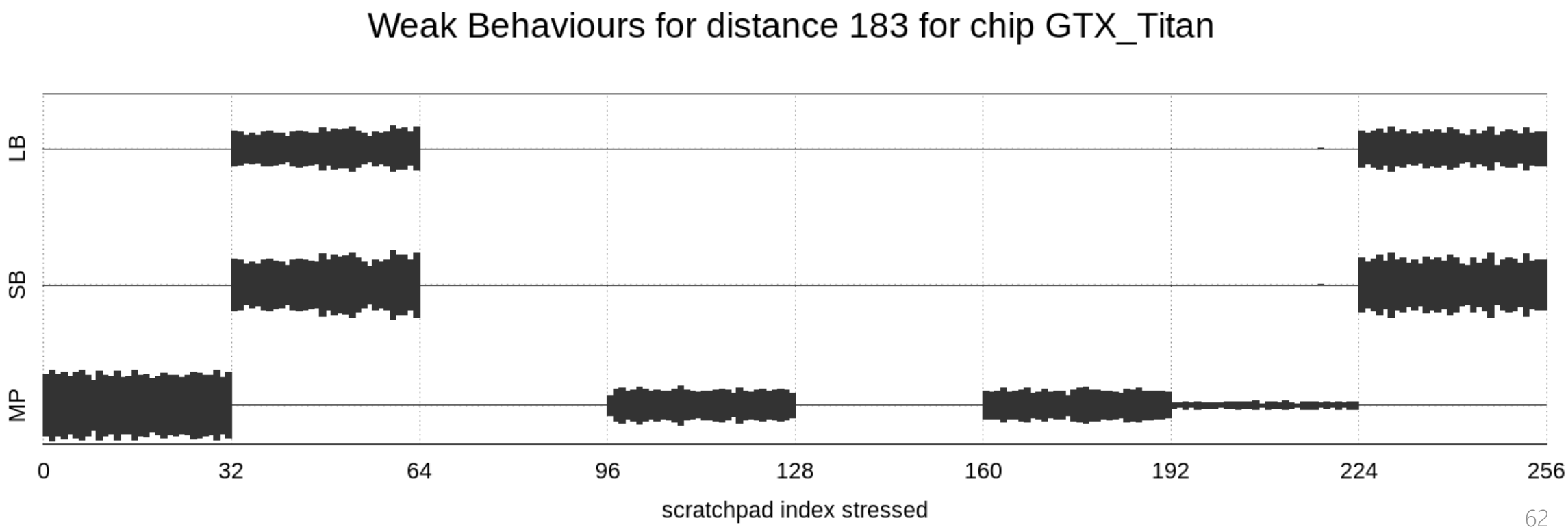


Memory stress

- Visualization samples

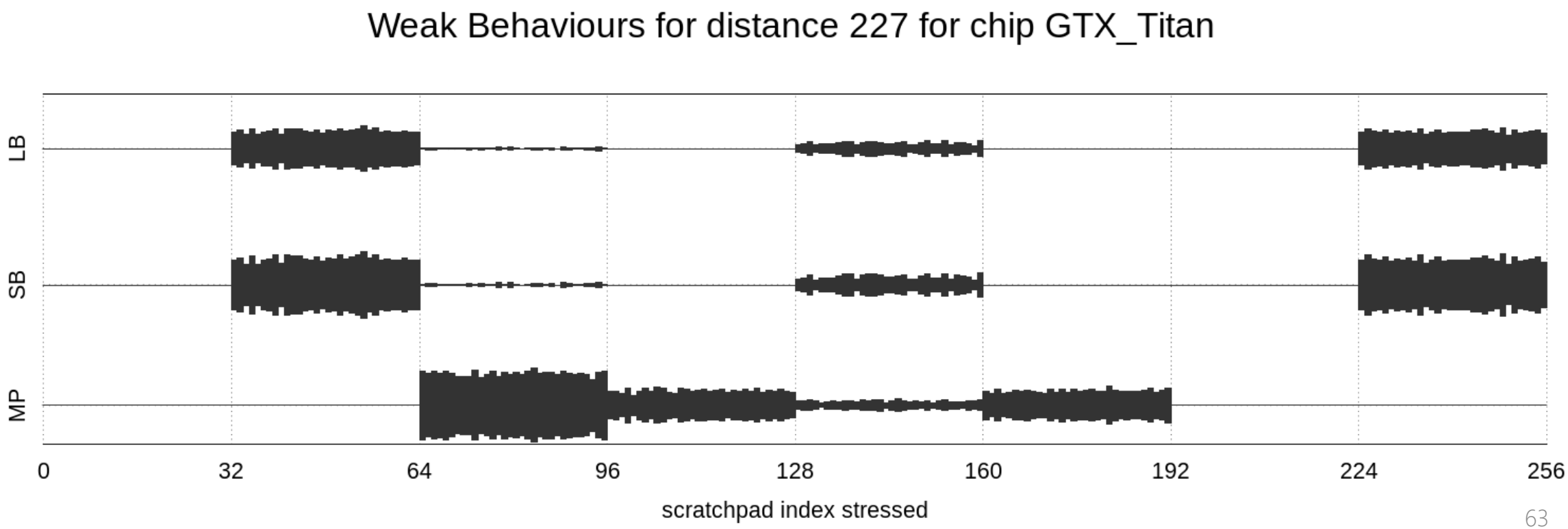
Memory stress

- Visualization samples



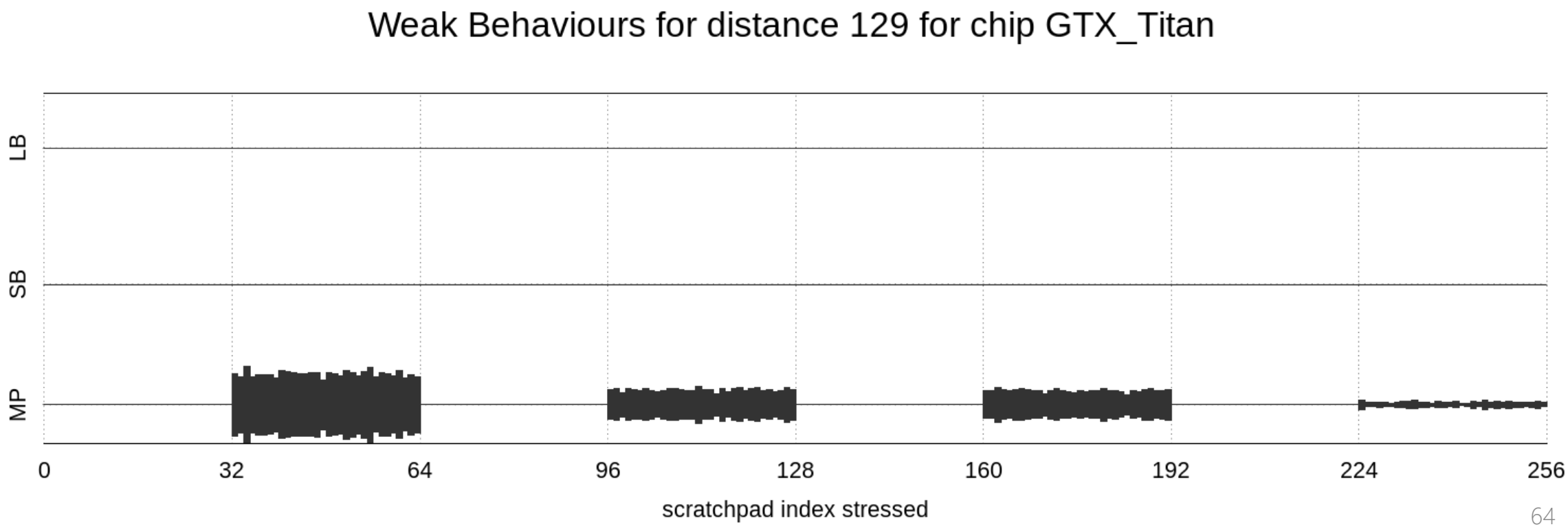
Memory stress

- Visualization samples



Memory stress

- Visualization samples



Memory stress

- What does this tell us?

Memory stress

- What does this tell us?
- *To reveal weak behaviors we only need to stress 1 in every 32 locations**
- We call a contiguous region of 32 elements a ***patch***

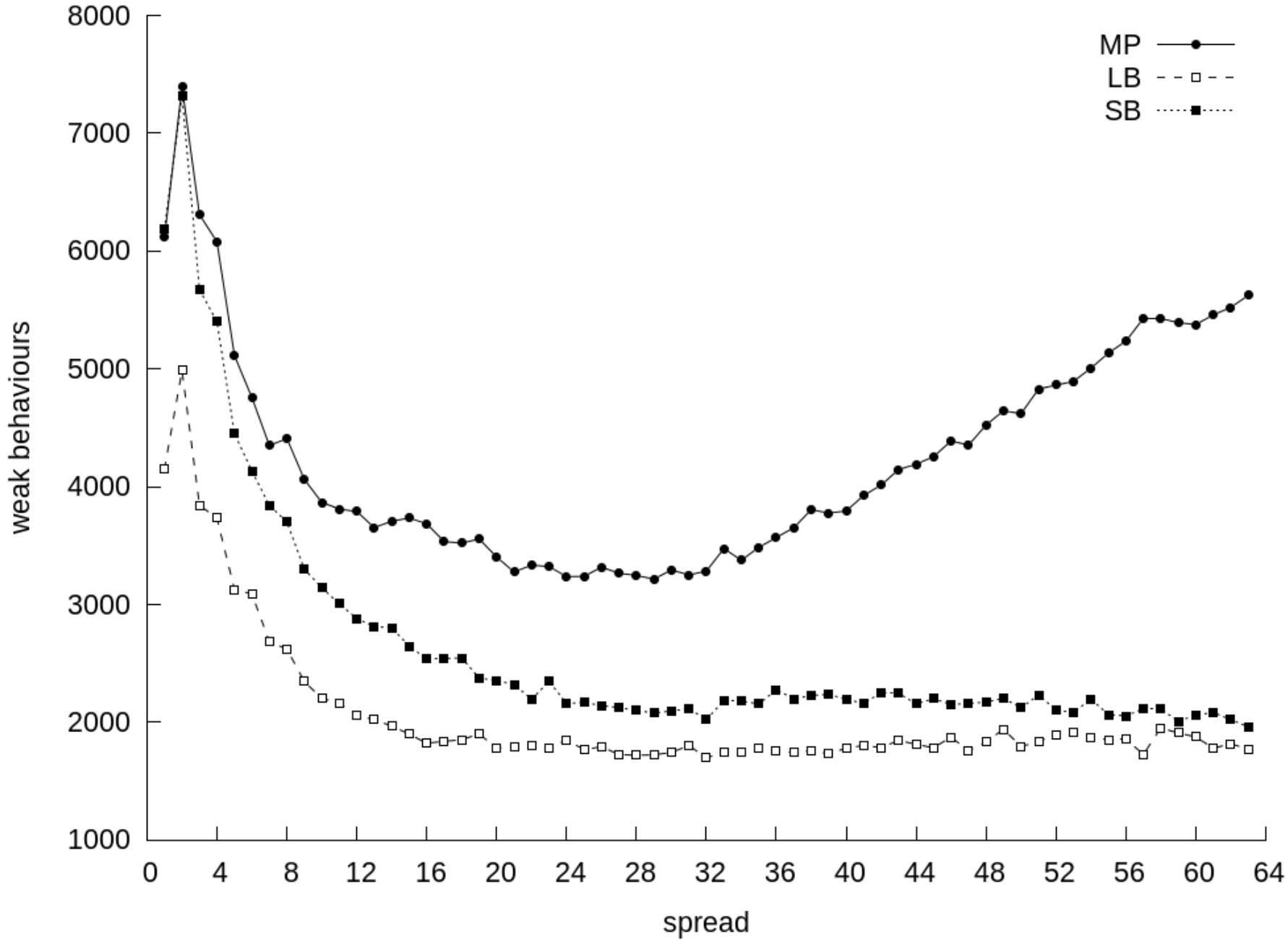
*64 for some chips

Memory stress

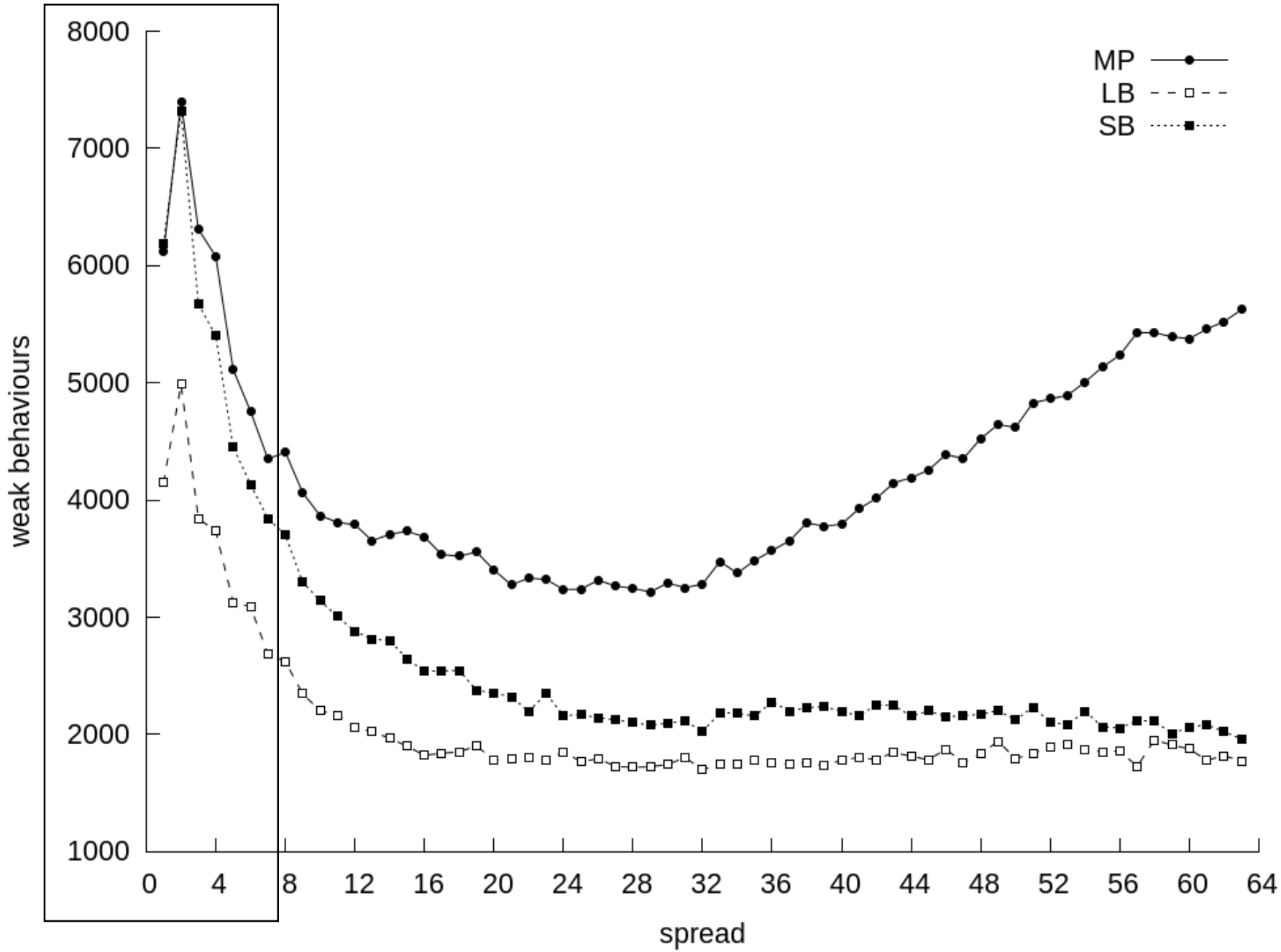
- How many patches can we effectively stress?
- If D is unknown (as in applications), we would like to stress as many disjoint patches as possible

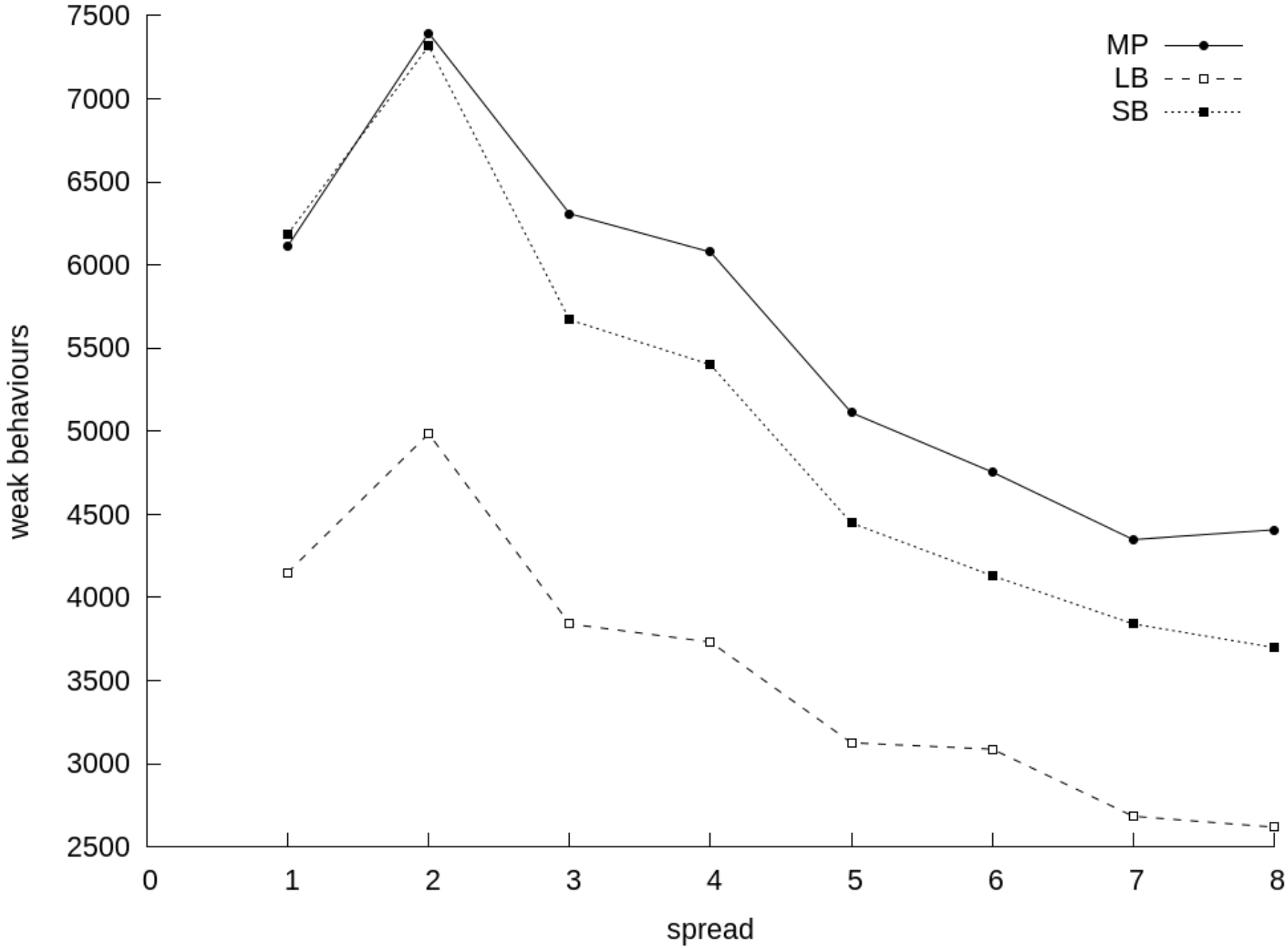
Memory stress

- Scratchpad has size of 64 patches
- We try stressing a randomly selected *n* patches for values 1 – 64 for *n*

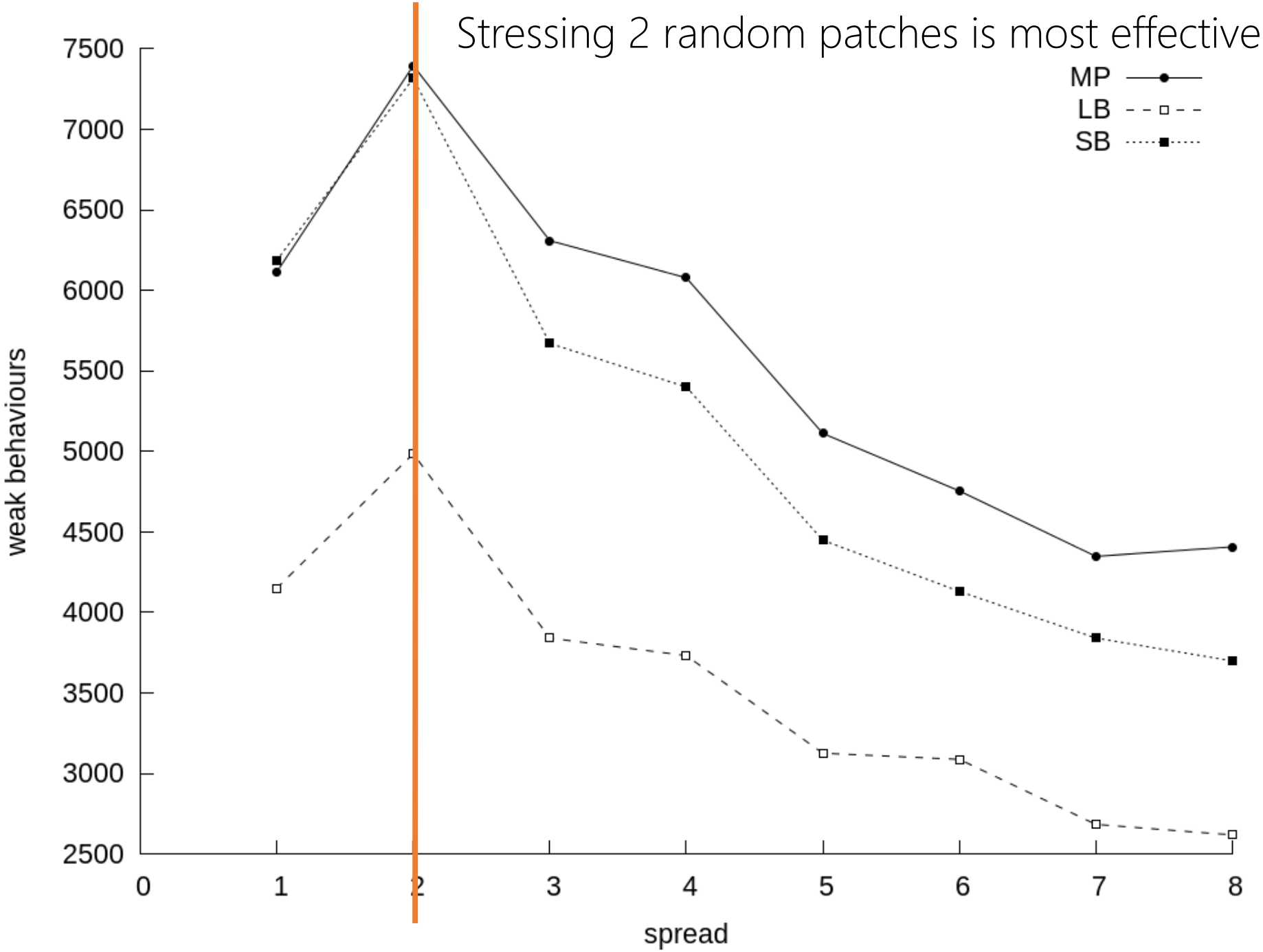

8000
7000
6000
5000
4000
3000
2000
1000
weak behaviours
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
spread
MP
LB
SB

Zoom in
on first 8




7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
weak behaviours
0
1
2
3
4
5
6
7
8
spread
MP
LB
SB

Stressing 2 random patches is most effective



Memory stress

- Now we have a memory stressing strategy!
- Stress two random patches in the scratchpad
- Patch size may change per chip

Roadmap

- Background
- Stress testing details
- Results

Application

N-body particle simulation in Lonestar GPU benchmark[1]

[1]see: http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu

Application

N-body particle simulation in Lonestar GPU benchmark[1]

- Documented to have communication across blocks
- No other information a priori needed for our testing
- Post condition checks the final location of particles

[1]see: http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu

Application

Executing the application for 1 hour (~2 seconds per run), the number of erroneous runs on a Quadro K5200:

Application

Executing the application for 1 hour (~2 seconds per run), the number of erroneous runs on a Quadro K5200:

No stress	With stress
0	

Application

Executing the application for 1 hour (~2 seconds per run), the number of erroneous runs on a Quadro K5200:

No stress	With stress
0	48

Comparing stresses

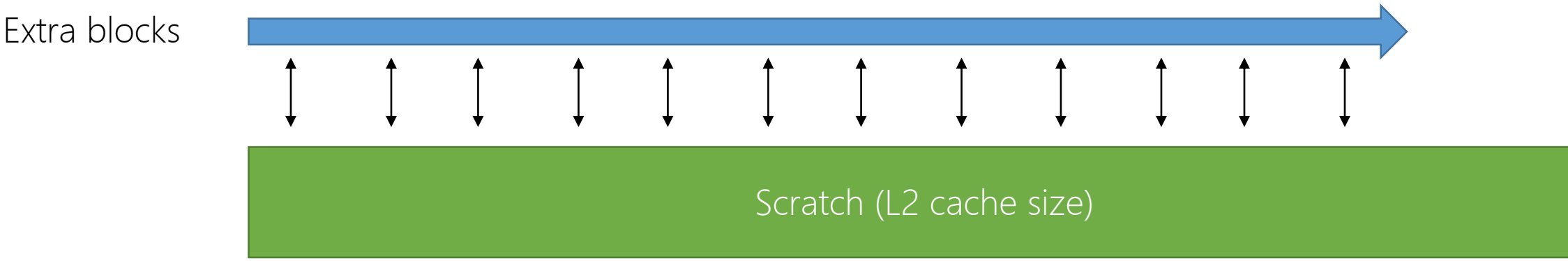
- Does it matter how we stress?

- We compare our systematic stressing method to 2 other stressing strategies

Comparing stresses

Cache stress:

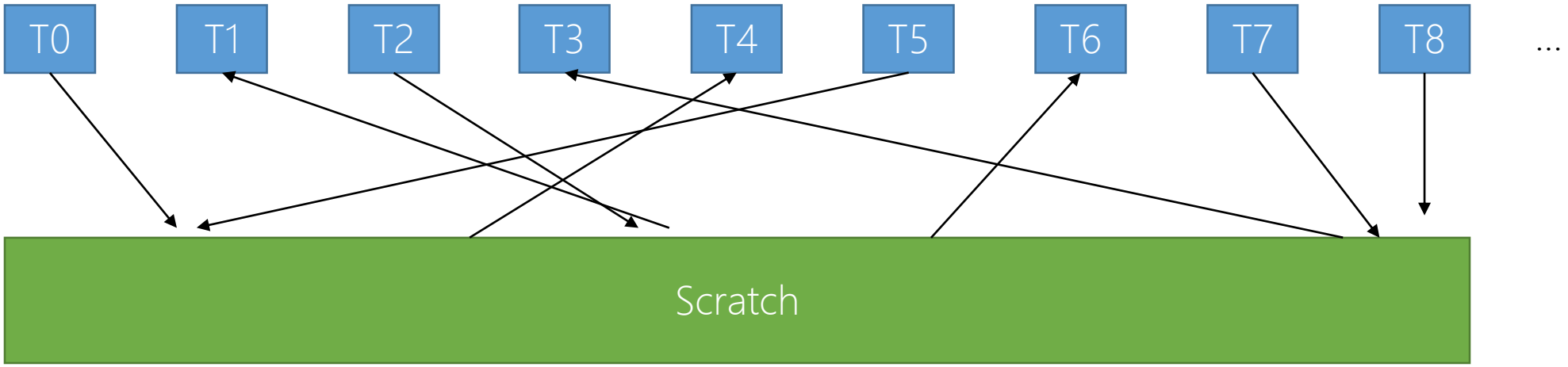
- Each stressing blocks streams over a scratchpad the size of the L2 cache, performing one read and write at each location



Comparing stresses

Random stress:

- Each thread randomly selects a location in the scratchpad and randomly performs a read or write to that location



Application

Executing the application for 1 hour (~2 seconds per run), the number of erroneous runs:

No stress	Systematic stress	Cache stress	Random stress
0	48		

Application

Executing the application for 1 hour (~2 seconds per run), the number of erroneous runs:

No stress	Systematic stress	Cache stress	Random stress
0	48	0	

Application

Executing the application for 1 hour (~2 seconds per run), the number of erroneous runs:

No stress	Systematic stress	Cache stress	Random stress
0	48	0	0

Application

How about the dot product application?

Chip	No stress	S. stress	Cache stress	Random stress
Titan	0	396		
GTX 980	0	495		

Application

How about the dot product application?

Chip	No stress	S. stress	Cache stress	Random stress
Titan	0	396	0	
GTX 980	0	495	2	

Application

How about the dot product application?

Chip	No stress	S. stress	Cache stress	Random stress
Titan	0	396	0	2
GTX 980	0	495	2	1

Full experimental study

- Tested:
  - 4 chips across 3 major Nvidia architectures
  - 10 applications
  - 3 different stress settings for each chip/application combination

Full experimental study

- Total of 40 chip/application combinations
- Observed weak behaviors in 32 of the combinations

	Systematic Stress	Random Stress	Cache Stress
# combinations showing weak memory	32	8	8
# combinations most effective stress	28	2	2

Some results:

- We provided empirical confirmation of 3 bugs reported in prior work
- We discovered unreported weak memory bugs in 2 applications

Future work

- Use our testing framework to automatically insert memory fences

- Benchmark the cost of fences on GPUs

Questions