29th Asia and South Pacific Design Automation Conference
(ASP-DAC 2024)

On Decomposing Complex Test Cases for Efficient Post-silicon Validation

Harshitha C, Sundarapalli Harikrishna, Peddakotla Rohith,
Sandeep Chandran, and Rajshekar Kalayappan

INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

॥ सा विद्या या विमुक्तये ॥
भारतीय प्रौद्योगिकी संस्थान धारवाड
Indian Institute of Technology Dharwad

This work was partly sponsored by Semiconductor Research Corporation (SRC)

Motivation

Scenario 1 (industry-inspired)**:**

- A chip manufacturing company gives an extensively tested chip to an OEM for early evaluation and integration
- A system prototype of OEM which incorporates the chip into it crashes after running for (say) 10 minutes
- The application that triggered the crash cannot be shared with the chip manufacturers due to IP restrictions
- The chip validation team struggles to reproduce the error because their in-house verification and validation techniques and strategies failed to uncover the bug earlier
- Significant time is lost to reproducing the error

Motivation

Scenario 2 (industry-inspired)**:**

- Initial silicon samples are exercised on validation boards in-house
- A server powered with these initial silicon sample crashes after 5 days of operation
- Validation team struggles to reproduce the error because
  - the entire software stack needs to be analyzed
  - bug reproduction takes very long time

Wish-list

- Bug Reproduction

Wish-list

- Bug Reproduction
  - Deterministic

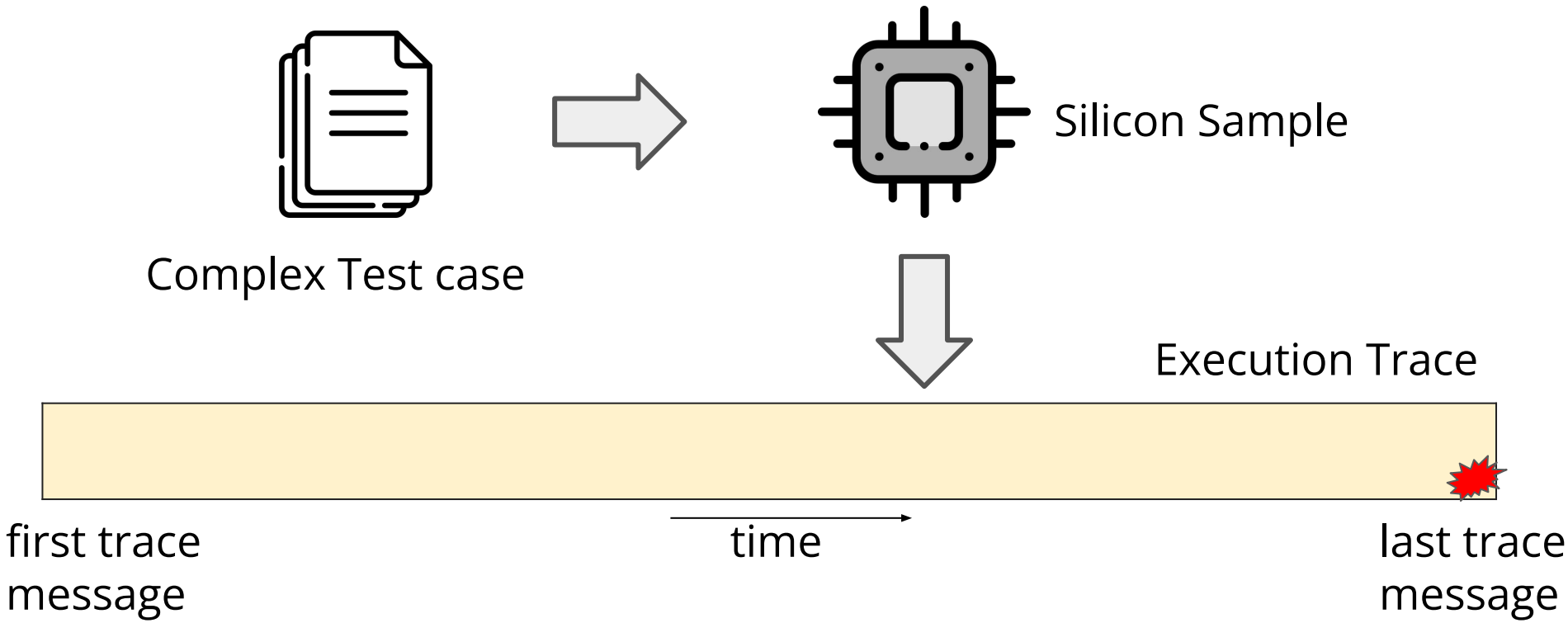
Wish-list

- Bug Reproduction
  - Deterministic
  - Privacy-aware

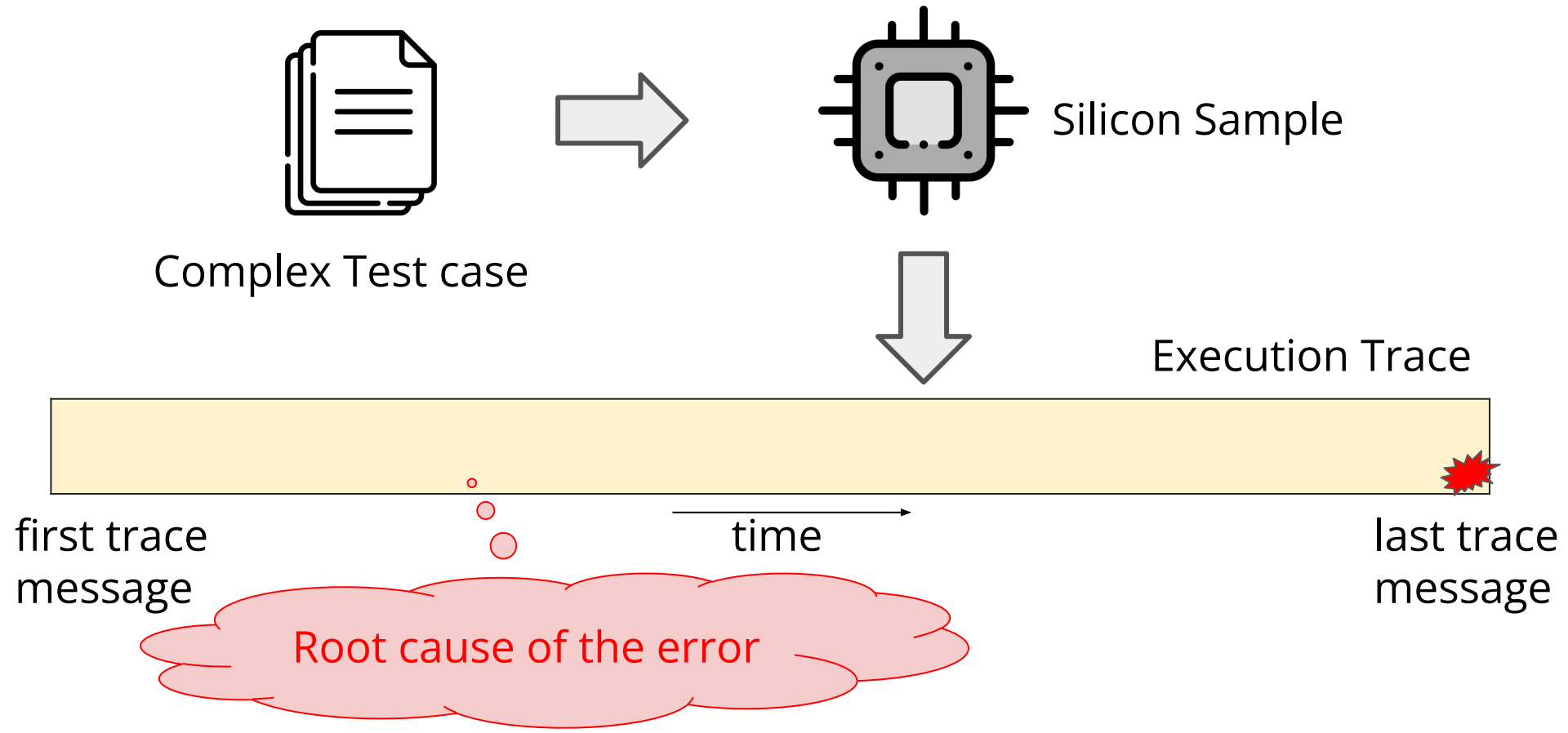
Wish-list

- Bug Reproduction
  - Deterministic
  - Privacy-aware
  - Fast

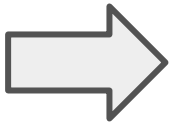
Solution Outline

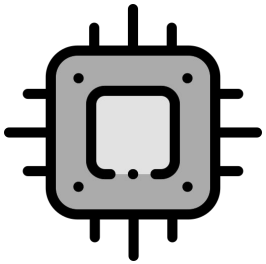


Solution Outline



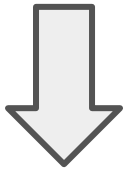
Solution Outline





Silicon Sample

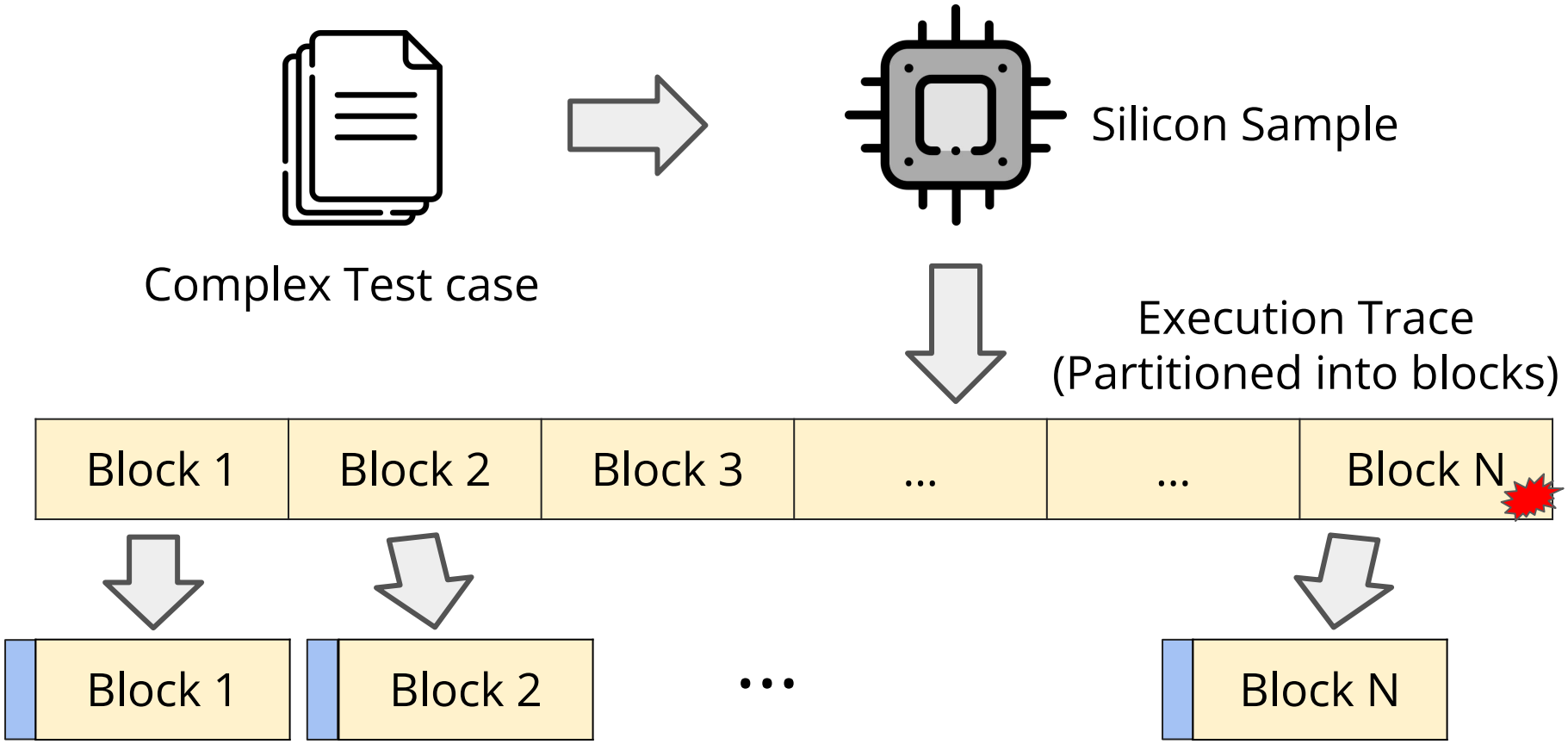
Complex Test case



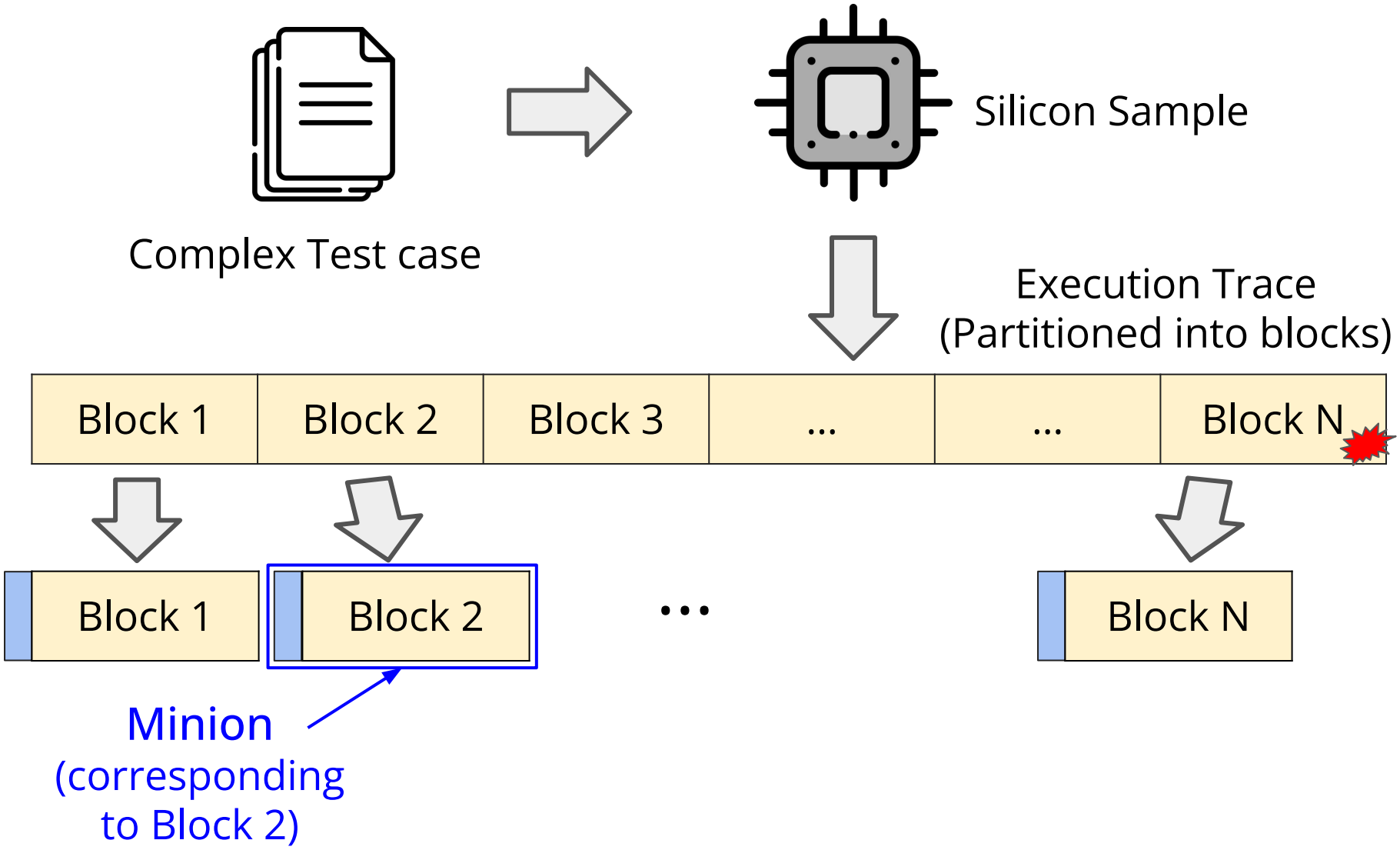
Execution Trace
(Partitioned into blocks
Block size is parameterized)

Block 1	Block 2	Block 3	...	...	Block N

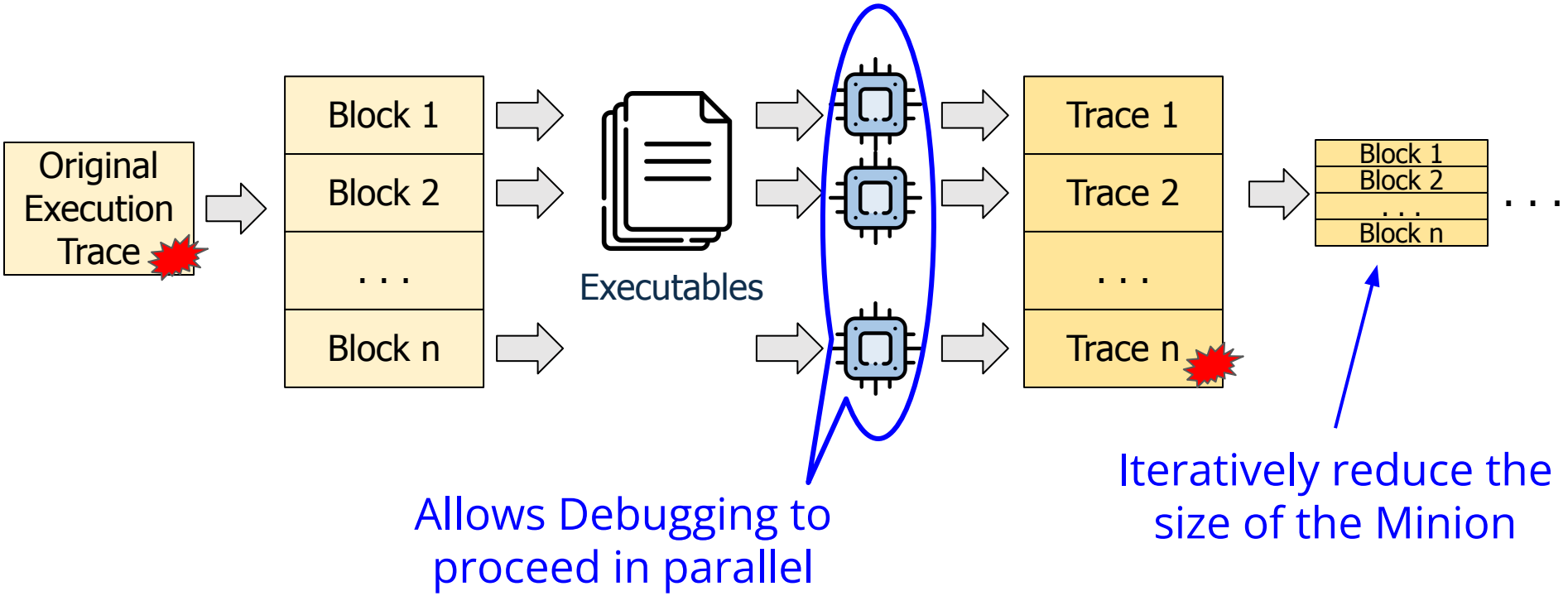
Solution Outline



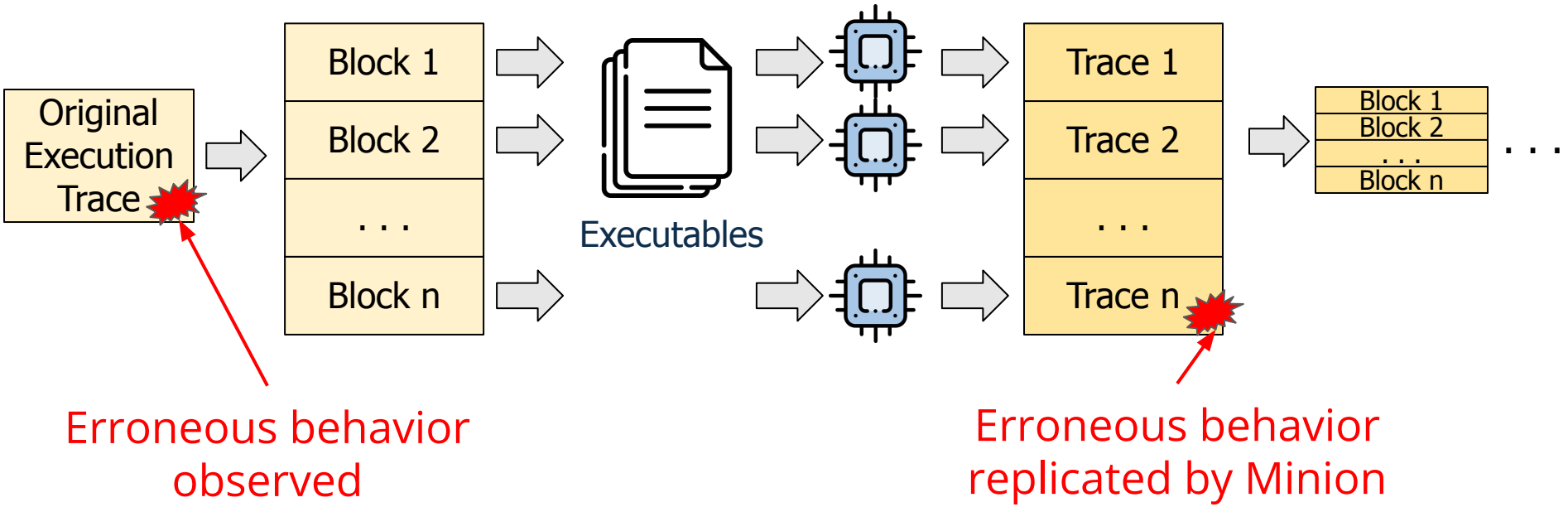
Solution Outline



Benefits of Gru

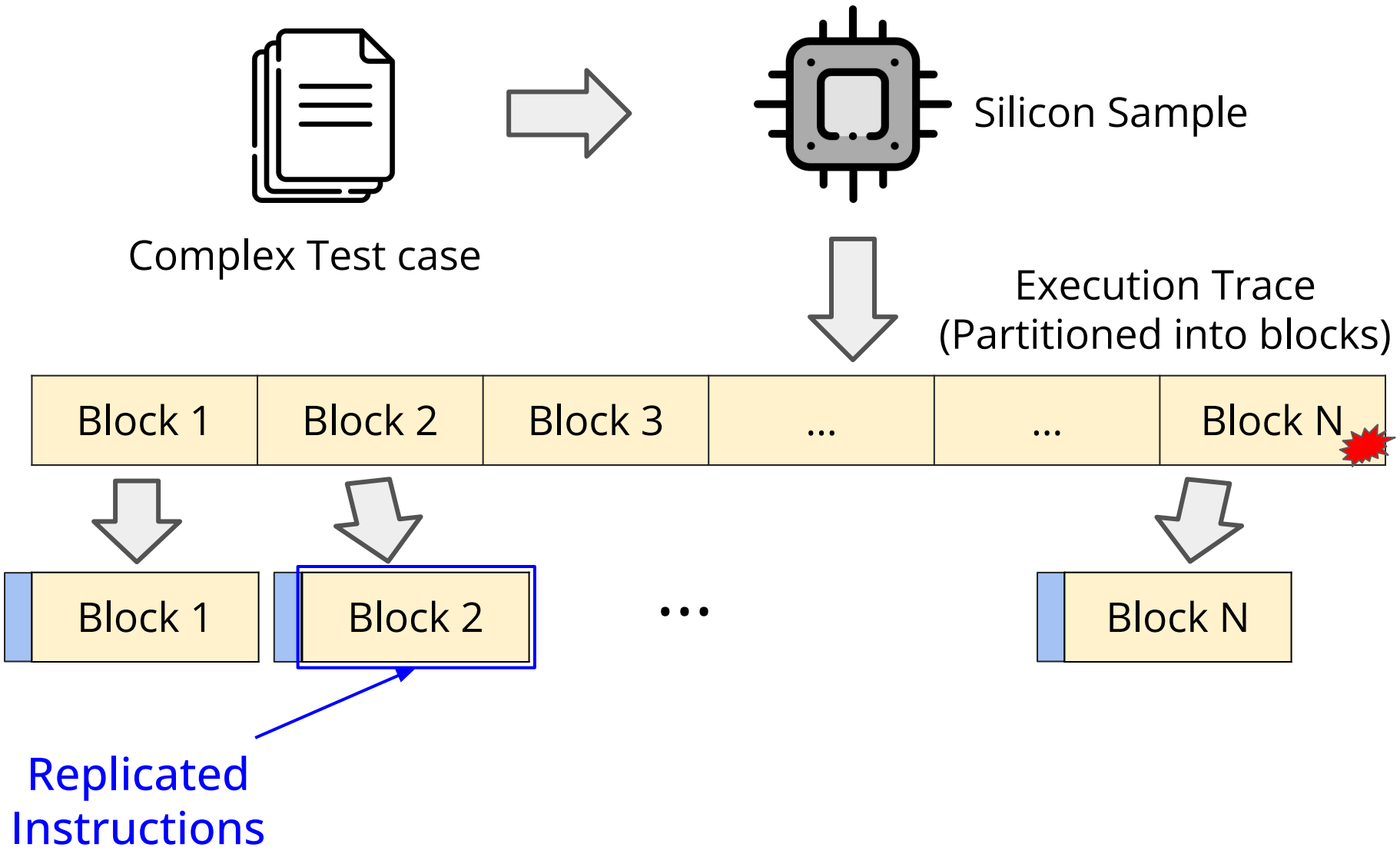


Benefits of Gru

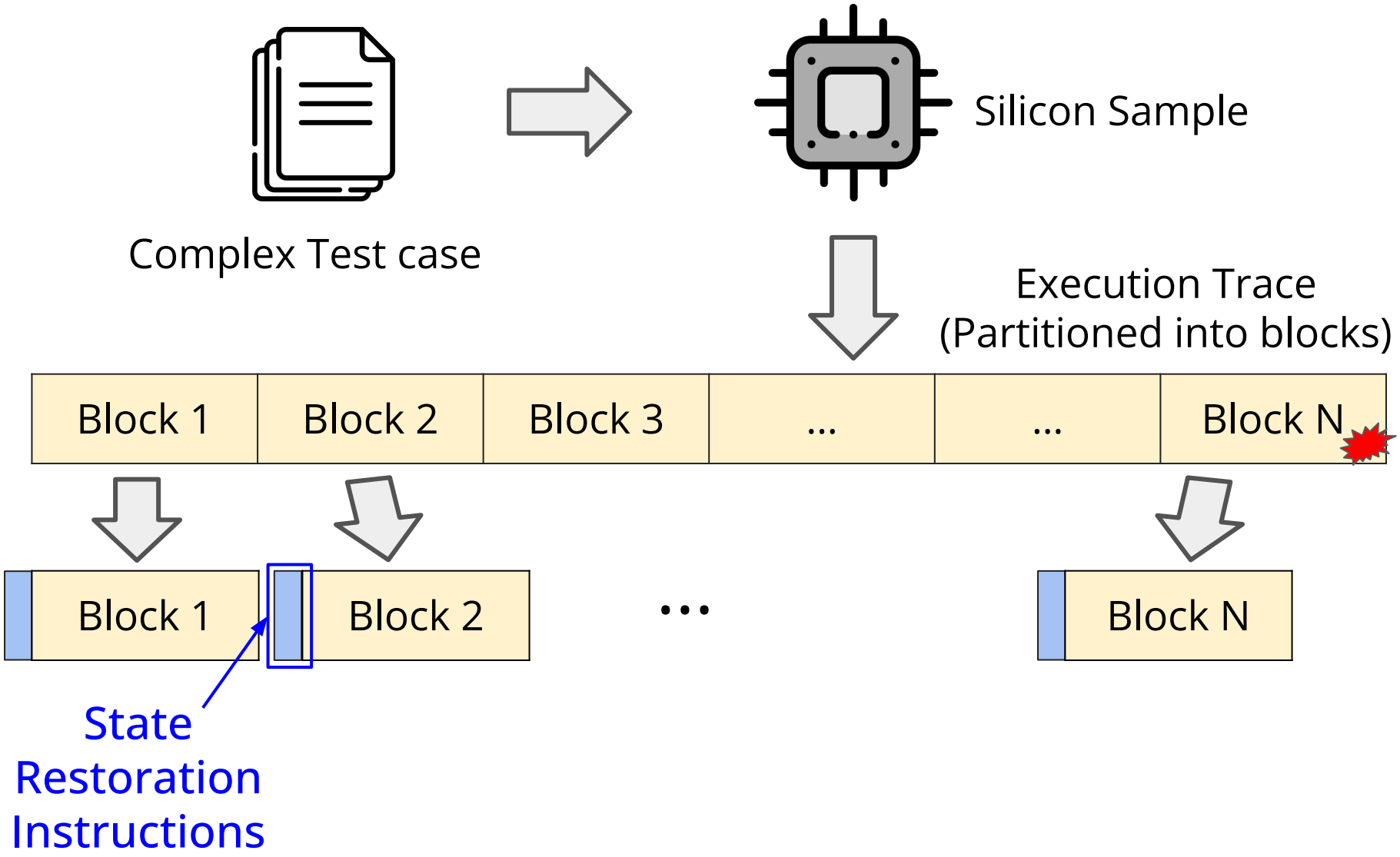


Privacy-aware Debugging

Solution Outline



Solution Outline



Minion Generation - Example

```
0x40010098: ld [%g2+0x000c], %o5 [0x00000300]
0x4001009c: st %o5,[%g1+0x000c]
0x400100a0: add %g2, 0x0010, %g2 [0xfffff020]
0x400100a4: bgu 0x40010078
0x400100a8: add %g1, 0x0010, %g1 [0x400ffe24]
0x400100ac: ld [%g2+0x000c], %o5 [0x00000010]
0x400100b0: srl %i2, 0x0004, %o5 [0x00000001]
0x400100b4: sll %o5, 0x0004, %g1 [0x00000010]
0x400100b8: add %o5, 0x0001, %o5 [0x00000002]
0x400100bc: sub %i2, %g1, %i2 [0x00000000]
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Minion Generation - Example

```
0x40010098: ld [%g2+0x000c], %o5 [0x00000300]
0x4001009c: st %o5,[%g1+0x000c]
```

PC:	Instruction	Output Value
0x400100a8:	add %g1, 0x0010, %g1	[0x400ffe24]

```
0x400100ac: ld [%g2+0x000c], %o5 [0x00000010]
0x400100b0: srl %i2, 0x0004, %o5 [0x00000001]
0x400100b4: sll %o5, 0x0004, %g1 [0x00000010]
0x400100b8: add %o5, 0x0001, %o5 [0x00000002]
0x400100bc: sub %i2, %g1, %i2 [0x00000000]
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Minion Generation - Example

```
0x40010098: ld [%g2+0x000c], %o5 [0x00000300]
0x4001009c: st %o5,[%g1+0x000c]
0x400100a0: add %g2, 0x0010, %g2 [0xfffff020]
0x400100a4: bgu 0x40010078
0x400100a8: add %g1, 0x0010, %g1 [0x400ffe24]
0x400100ac: ld [%g2+0x000c], %o5 [0x00000010]
0x400100b0: srl %i2, 0x0004, %o5 [0x00000001]
0x400100b4: sll %o5, 0x0004, %g1 [0x00000010]
0x400100b8: add %o5, 0x0001, %o5 [0x00000002]
0x400100bc: sub %i2, %g1, %i2 [0x00000000]
```

Block 1

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

Minion Generation - Example

```
sll %o5, 0x0004, %o5
subcc %i2, 0x0003, %g0
ld [%i2+0x000c], %g4
mov %i2, %g3
bgu 0x40010048
add %o5, %o5, %o5
subcc %o5, 0x0040, %g0
be 0x400100f0
mov 0x0000, %g1
ret
```

Minion corresponding to Block 2

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

Minion Generation - Example

```
sll %o5, 0x0004, %o5
subcc %i2, 0x0003, %g0
ld [%i2+0x000c], %g4
mov %i2, %g3
bgu 0x40010048
add %o5, %o5, %o5
subcc %o5, 0x0040, %g0
be 0x400100f0
mov 0x0000, %g1
ret
```

Incorrect

Mi[illegible]
corr[illegible]onding
to Block 2

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

Minion Generation - Example

Incorrect

```
sll %o5, 0x0004, %o5
subcc %i2, 0x0003, %g0
ld [%i2+0x000c], %g4
mov %i2, %g3
bgu 0x40010048
add %o5, %o5, %o5
subcc %o5, 0x0040, %g0
be 0x400100f0
mov 0x0000, %g1
ret
```

The value in %o5 in the minion should match the value that prevailed at the end of Block 1 ([0x00000002]) so that it produces the same value as in the original trace

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

Minion Generation - Example

The Minion should restore the values in the register %o5, %i2 and the memory location [0x0000000c] to the value that prevailed at the end of Block 1 to replicate the behavior observed in the original trace

State: %o5:??, %i2:??, [0x0000000c]:??

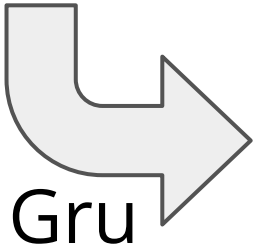
```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

Minion corresponding to Block 2



restore.s

```
 1: .global rstr
 2: .extern test
 3: .text
 4: rstr:
 5:   sethi %hi(0x00000300), %o1
 6:   or %o1, %lo(0x00000300), %o1
 7:   sethi %hi(0x0000000c), %o2
 8:   or %o2, %lo(0x0000000c), %o2
 9:   st %o1, [%o2]
10:   sethi %hi(0x00000002), %o5
11:   or %o5, %lo(0x00000002), %o5
12:   sethi %hi(0x00000000), %i2
13:   or %i2, %lo(0x00000000), %i2
14:   wrpsr 0xff400120
15:   ba test
16:   nop
```

replica.s

```
 1: .global test
 2: .text
 3: test:
 4:   sll %o5, 0x0004, %o5
 5:   subcc %i2, 0x0003, %g0
 6:   ld [%i2+0x000c], %g4
 7:   mov %i2, %g3
 8:   bgu 0x40010048
 9:   add %o5, %o5, %o5
10:   subcc %o5, 0x0040, %g0
11:   be 0x400100f0
12:   mov 0x0000, %g1
13:   nop
14:   nop
15:   nop
16:   ret
```

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

Block 2

Minion corresponding to Block 2

Gru

restore.s

```
 1: .global rstr
 2: .extern test
 3: .text
 4: rstr:
 5:   sethi %hi(0x00000300), %o1
 6:   or %o1, %lo(0x00000300), %o1
 7:   sethi %hi(0x0000000c), %o2
 8:   or %o2, %lo(0x0000000c), %o2
 9:   st %o1, [%o2]
10:   sethi %hi(0x00000002), %o5
11:   or %o5, %lo(0x00000002), %o5
12:   sethi %hi(0x00000000), %i2
13:   or %i2, %lo(0x00000000), %i2
14:   wrpsr 0xff400120
15:   ba test
16:   nop
```

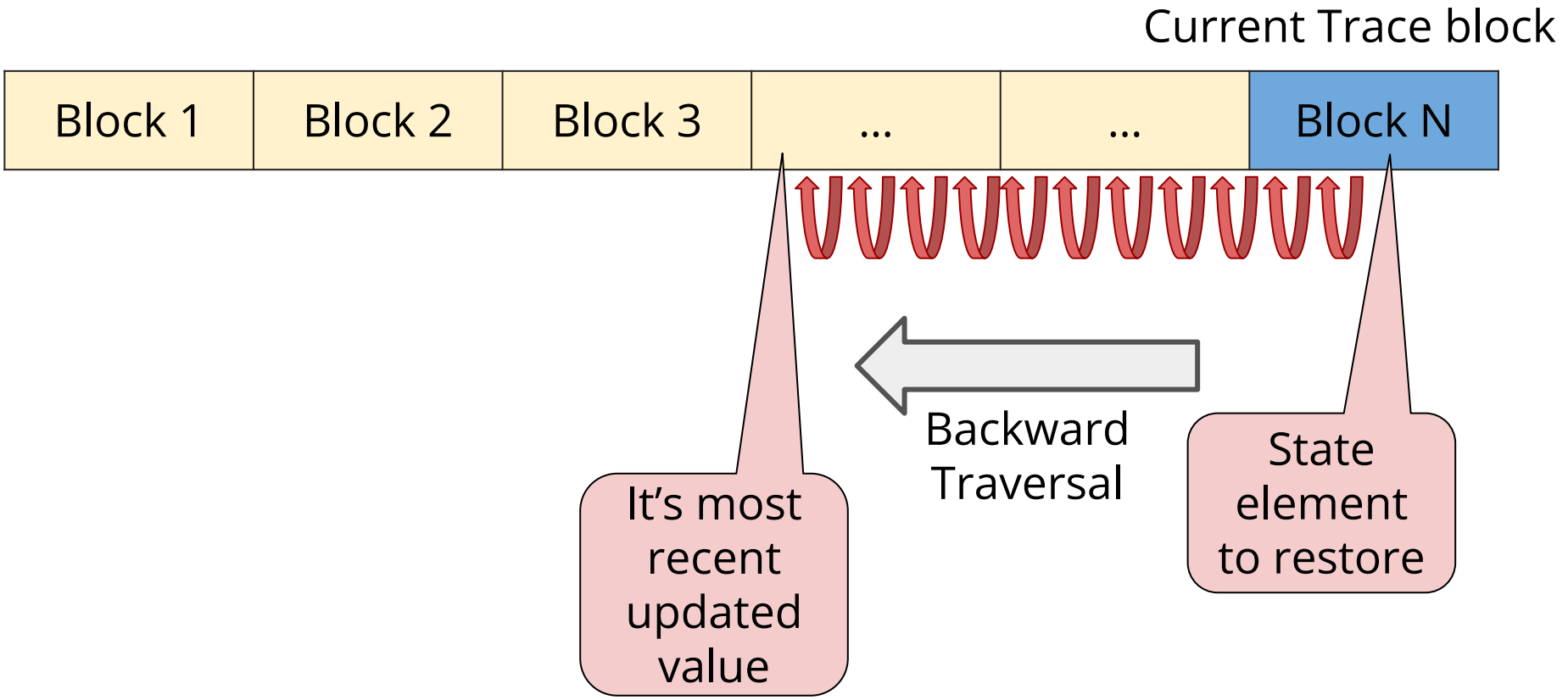
replica.s

```
 1: .global test
 2: .text
 3: test:
 4:   sll %o5, 0x0004, %o5
 5:   subcc %i2, 0x0003, %g0
 6:   ld [%i2+0x000c], %g4
 7:   mov %i2, %g3
 8:   bgu 0x40010048
 9:   add %o5, %o5, %o5
10:   subcc %o5, 0x0040, %g0
11:   be 0x400100f0
12:   mov 0x0000, %g1
13:   nop
14:   nop
15:   nop
16:   ret
```

Minion Generation - Procedure

- **Step 1**: Identify registers that are read before being written to in the current trace block
- **Step 2**: Infer the values of these registers using State-inference techniques
- **Step 3**: Identify memory locations that are read before being written to in the current trace block (using reg values identified above, if required)
- **Step 4**: Infer the values of these memory locations using State-inference techniques
- **Step 5**: Emit State Restoration Instructions (SRIs) into the Minion (`restore.s`) for the memory locations and registers identified in Steps 3 and 1 respectively.
- **Step 6**: Copy instructions in the current trace block into the Minion (`replica.s`).
- **Step 7:** Compile and link `restore.s` and `replica.s`

State Inference Technique - Brute Force



State Inference Technique - Brute Force

```
0x40010098: ld [%g2+0x000c], %o5 [0x00000300]
0x4001009c: st %o5,[%g1+0x000c]
0x400100a0: add %g2, 0x0010, %g2 [0xffffff020]
0x400100a4: bgu 0x40010078
0x400100a8: add %g1, 0x0010, %g1 [0x400ffe24]
0x400100ac: ld [%g2+0x000c], %o5 [0x00000010]
0x400100b0: srl %i2, 0x0004, %o5 [0x00000001]
0x400100b4: sll %o5, 0x0004, %g1 [0x00000010]
0x400100b8: add %o5, 0x0001, %o5 [0x00000002]
0x400100bc: sub %i2, %g1, %i2 [0x00000000]
```

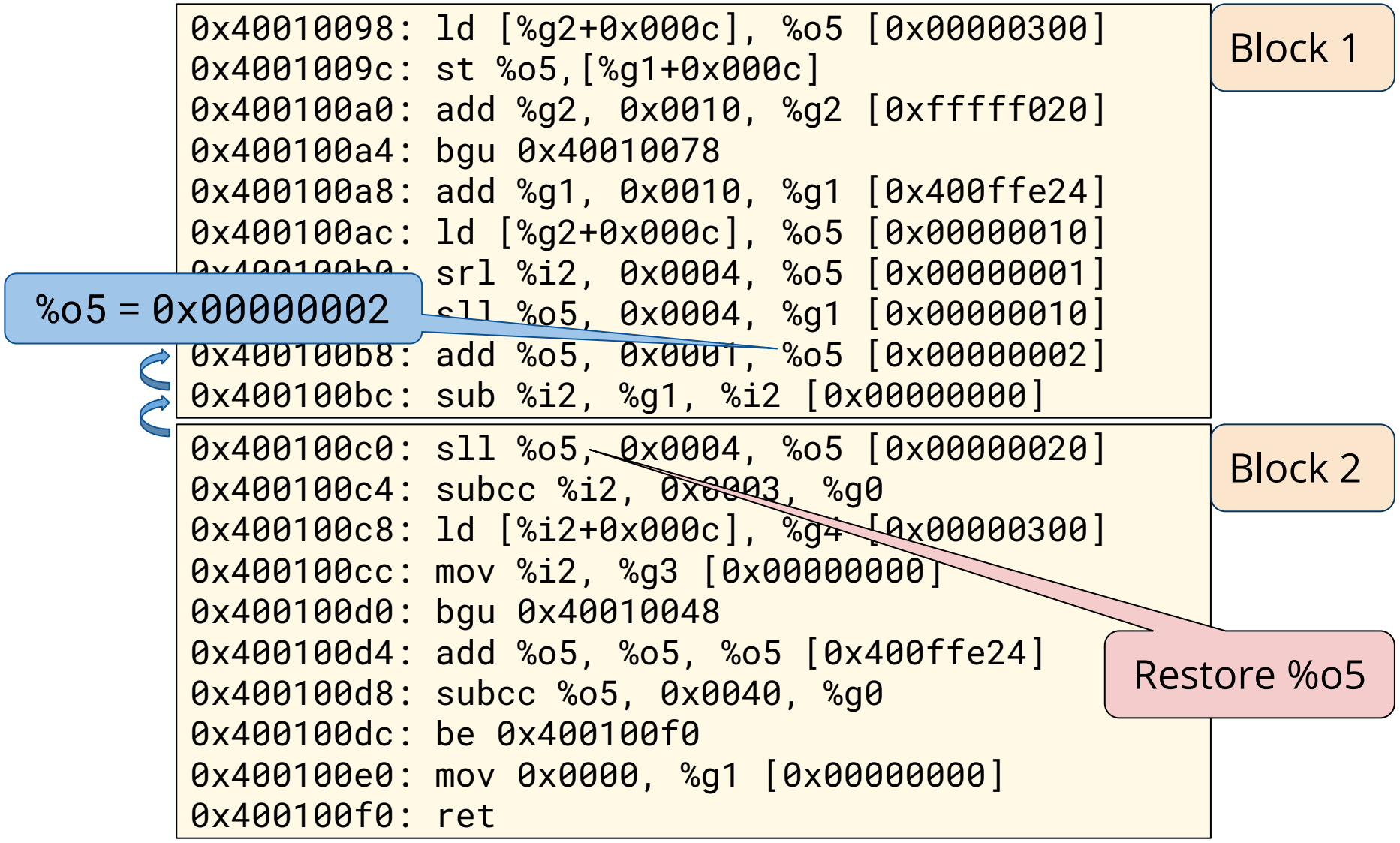
Block 1

```
0x400100c0: sll %o5, 0x0004, %o5 [0x00000020]
0x400100c4: subcc %i2, 0x0003, %g0
0x400100c8: ld [%i2+0x000c], %g4 [0x00000300]
0x400100cc: mov %i2, %g3 [0x00000000]
0x400100d0: bgu 0x40010048
0x400100d4: add %o5, %o5, %o5 [0x400ffe24]
0x400100d8: subcc %o5, 0x0040, %g0
0x400100dc: be 0x400100f0
0x400100e0: mov 0x0000, %g1 [0x00000000]
0x400100f0: ret
```

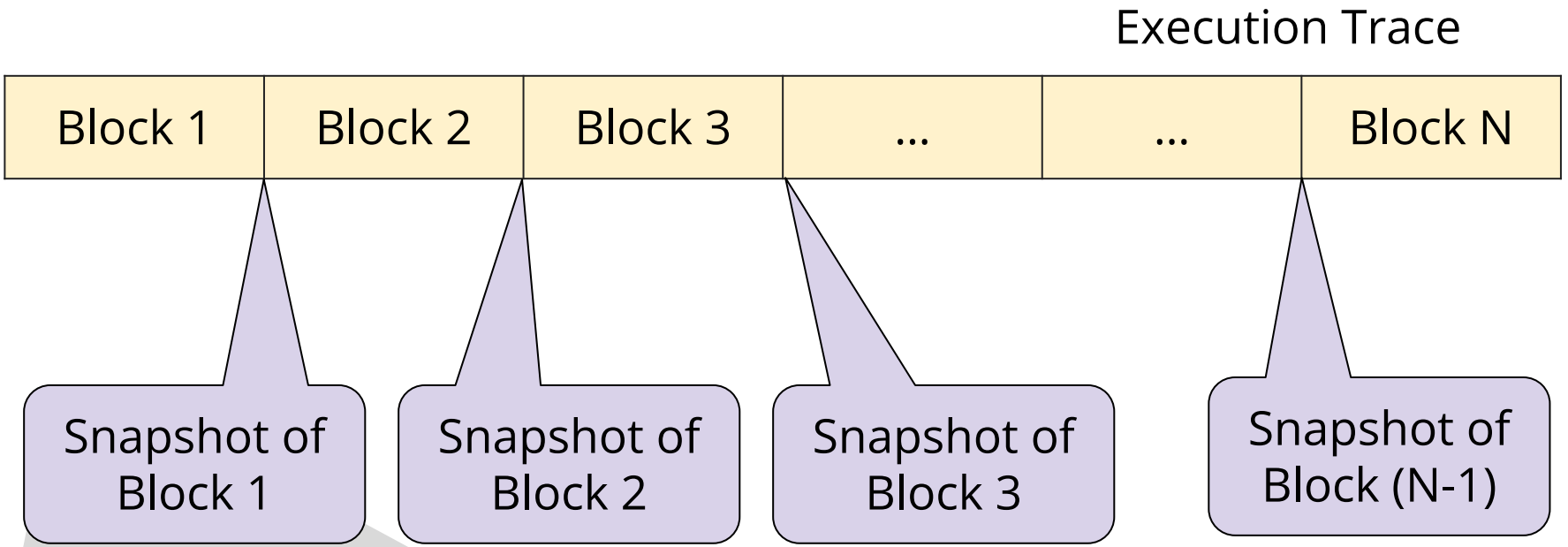
Block 2

Restore %o5

State Inference Technique - Brute Force



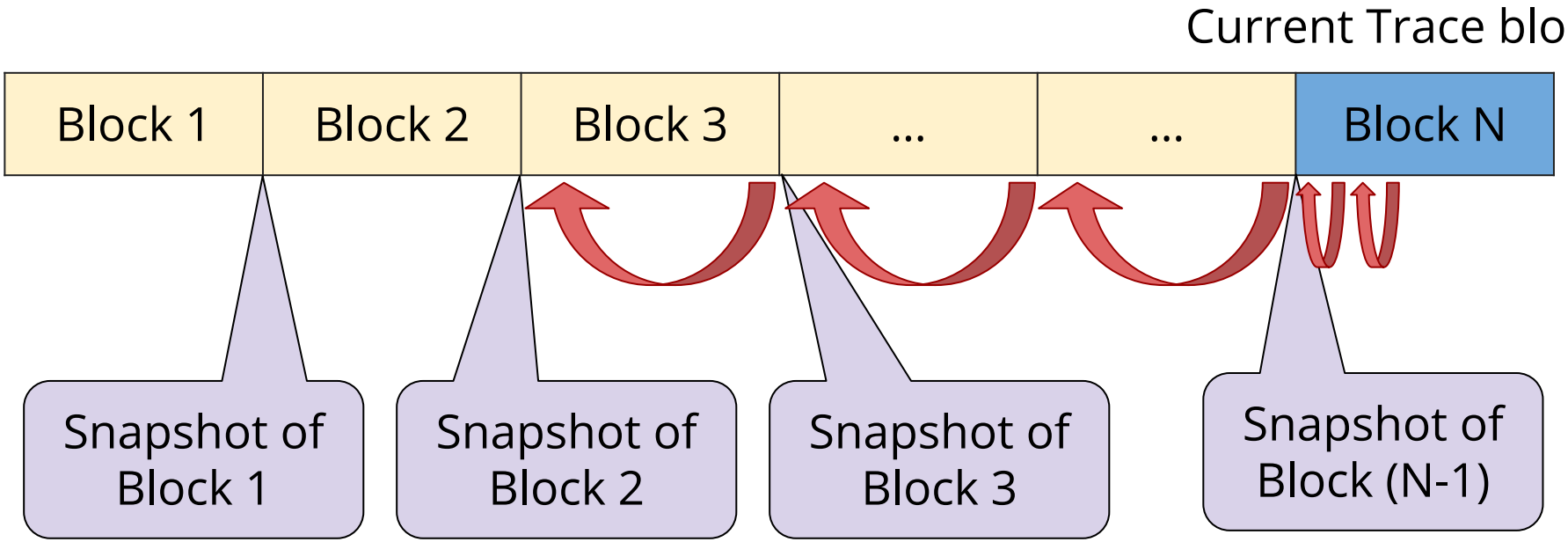
State Inference Technique - Snapshots-saving



State Element	Updated Value
%o5	0x00000002
%g1	0x00000010
%g2	0xfffff020

- Snapshots capture state-updates in each trace block
- Snapshots created as trace is decoded

State Inference Technique - Snapshots-saving



- Backward traversal jumps from one snapshot to an earlier snapshot till the most recently updated value is found

State Inference Technique - Inverted Map

Block 1	Block 2	Block 3	...	...	Block N

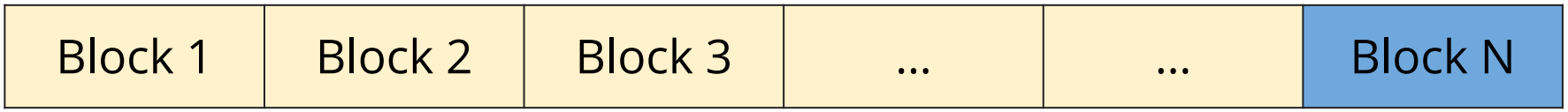
Inverted Map

State Element	Current Value
%o5	0x00000002
%g1	0x00000010
%g2	0xfffff020
%i2	0x00000000
%g4	0xfffff020

- Inverted Map is constructed as the execution trace is decoded

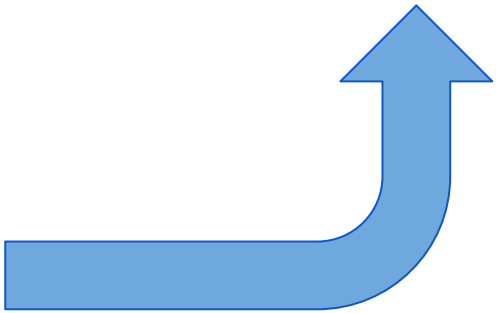
State Inference Technique - Inverted Map

Current Trace block



Inverted Map

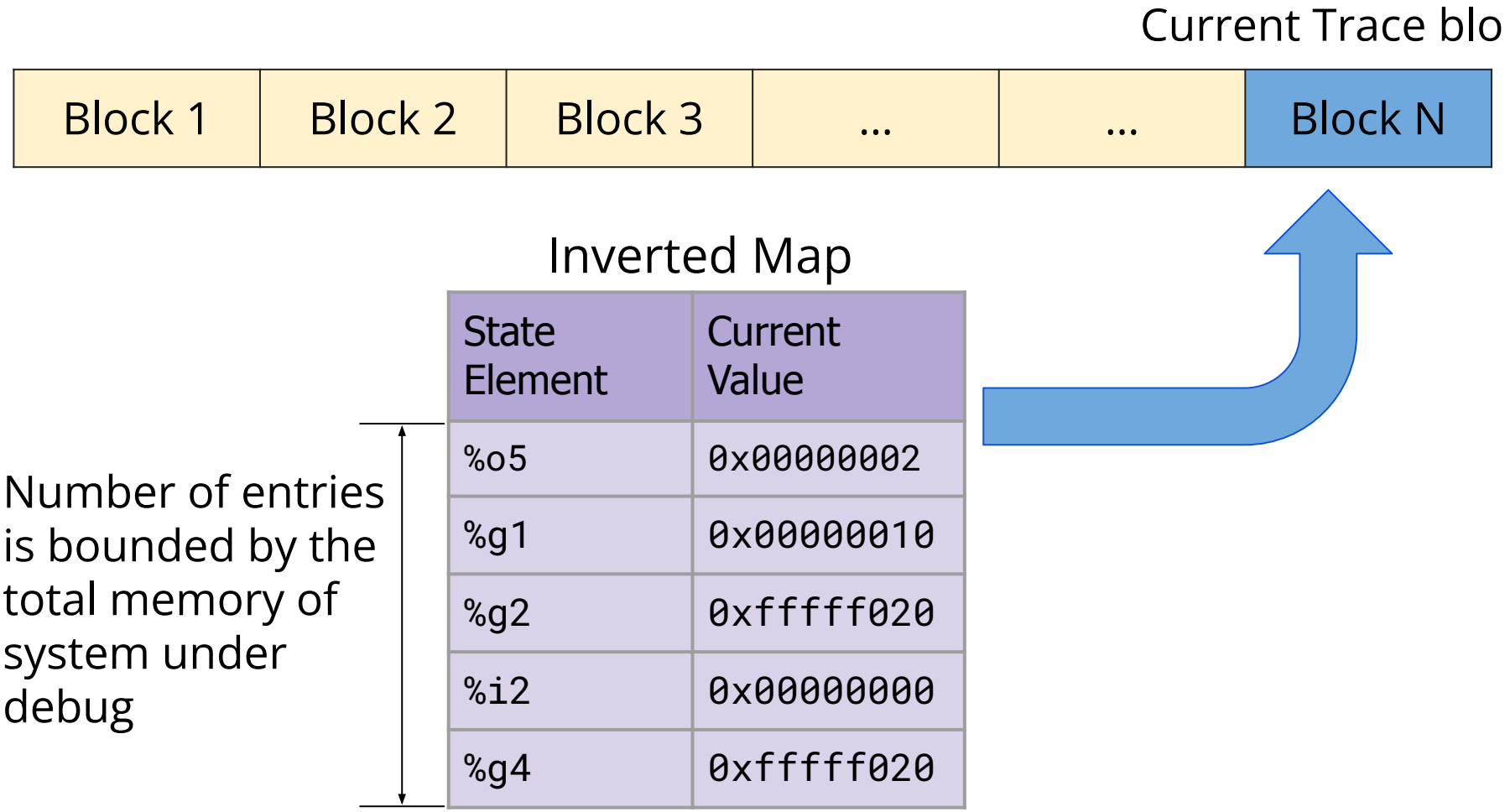
State Element	Current Value
%o5	0x00000002
%g1	0x00000010
%g2	0xfffff020
%i2	0x00000000
%g4	0xfffff020



Inverted Map is looked up to determine the most recent value of a state element

Avoids backward passes over the trace

State Inference Technique - Inverted Map

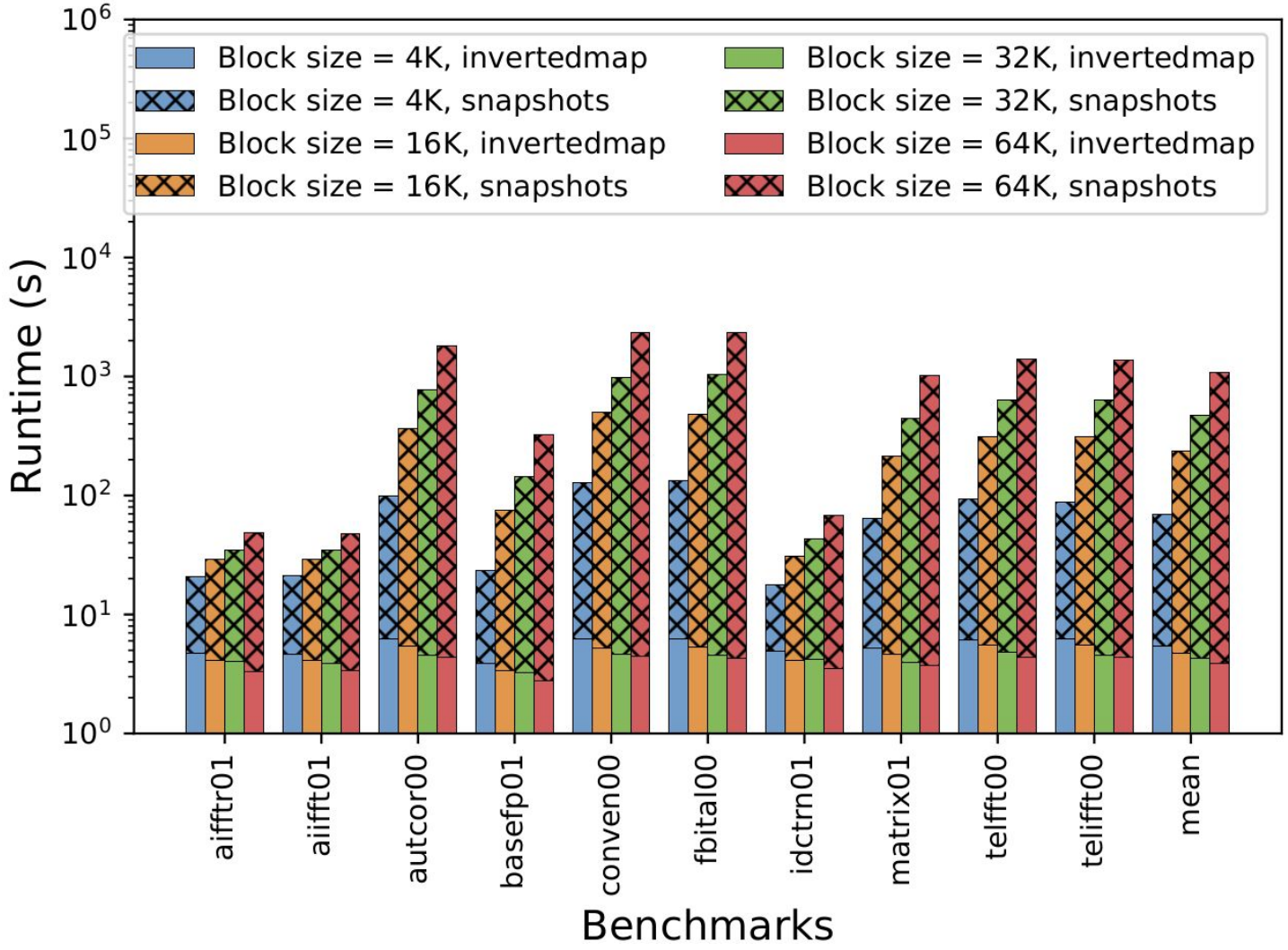


Experimental Setup

- LEON3 SoC with one SPARCv8 core, AHB, DDR and a Debug Support Unit (DSU)
  - 4KB Instruction Trace Buffer and 4KB AHB Trace Buffer
- Executed 10 (out of 17) benchmarks from the EEMBC AutoBench and TeleBench suites
  - Compiled using Bare-C Cross Compiler (BCC) - Bare-metal runs
- Execution time of 10ms
  - Captured traces had 369K messages (on average)
  - Approx. size of 5.6 MB
- Gru implemented in Python v3.10 and executed on an Intel Xeon Gold 5218R processor (2.1 GHz base frequency) with 32 GB RAM
  - Varied block size from 4KB to 64KB

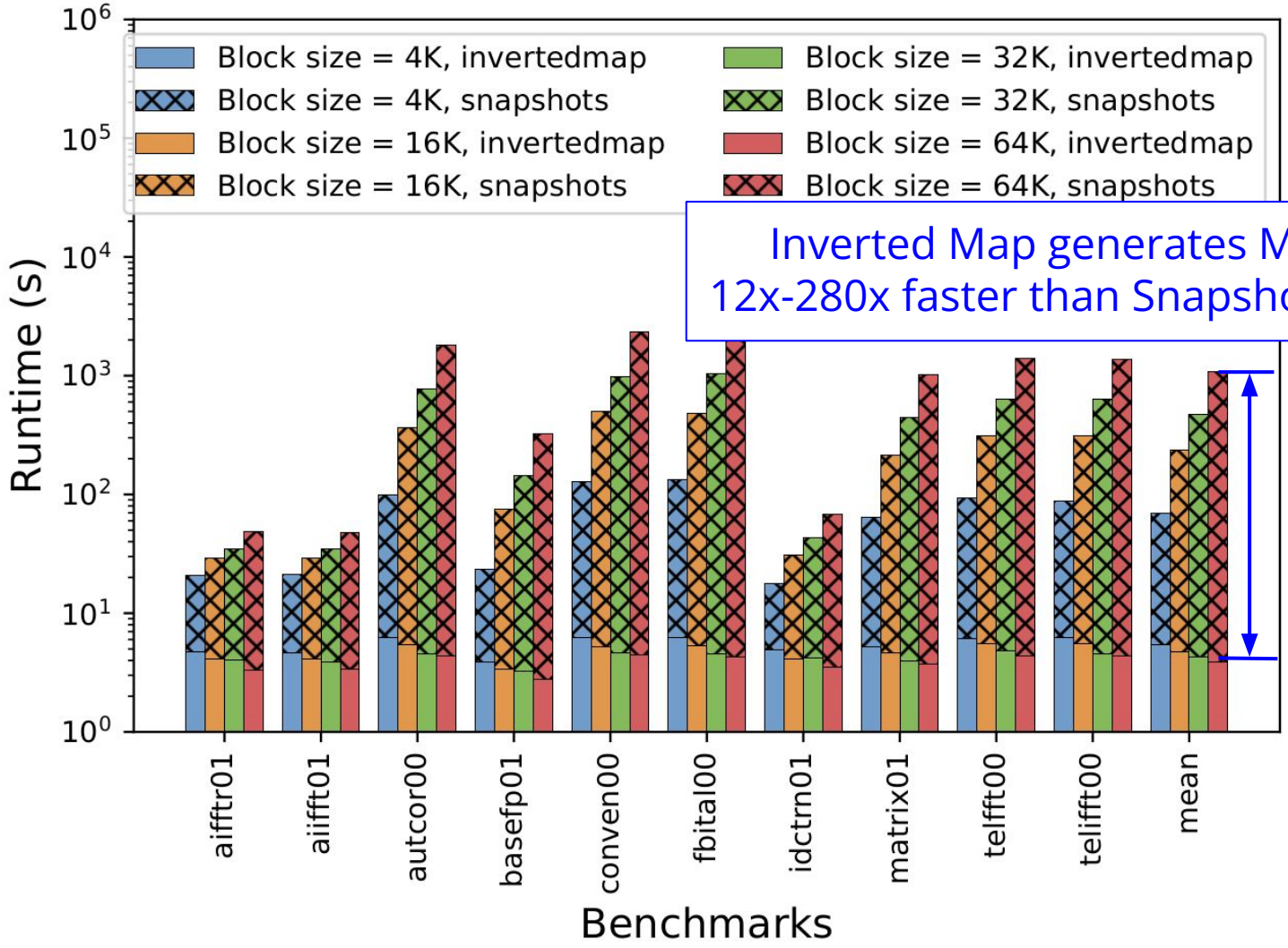
Results

Minion Generation Times



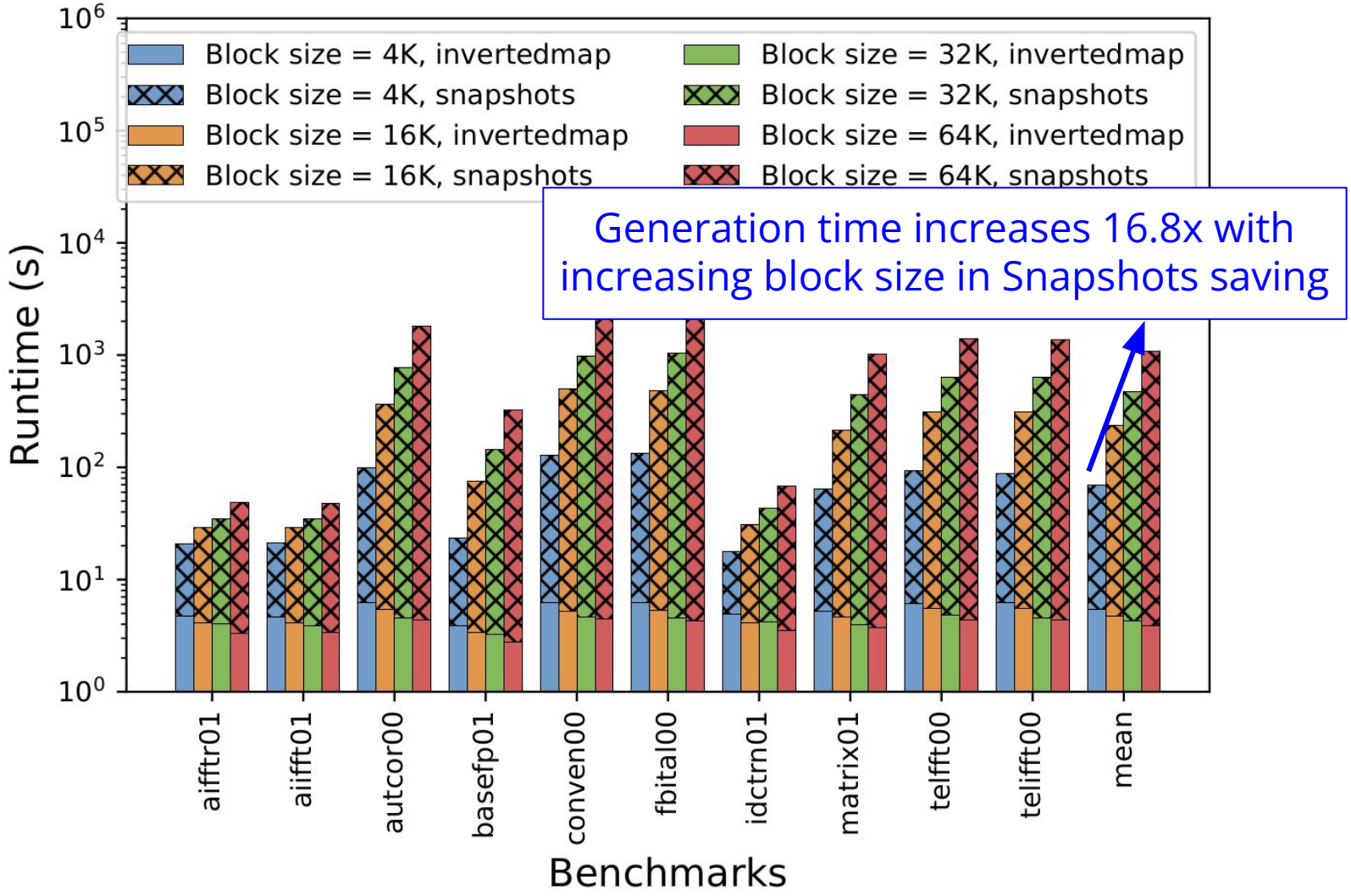
Results

Minion Generation Times



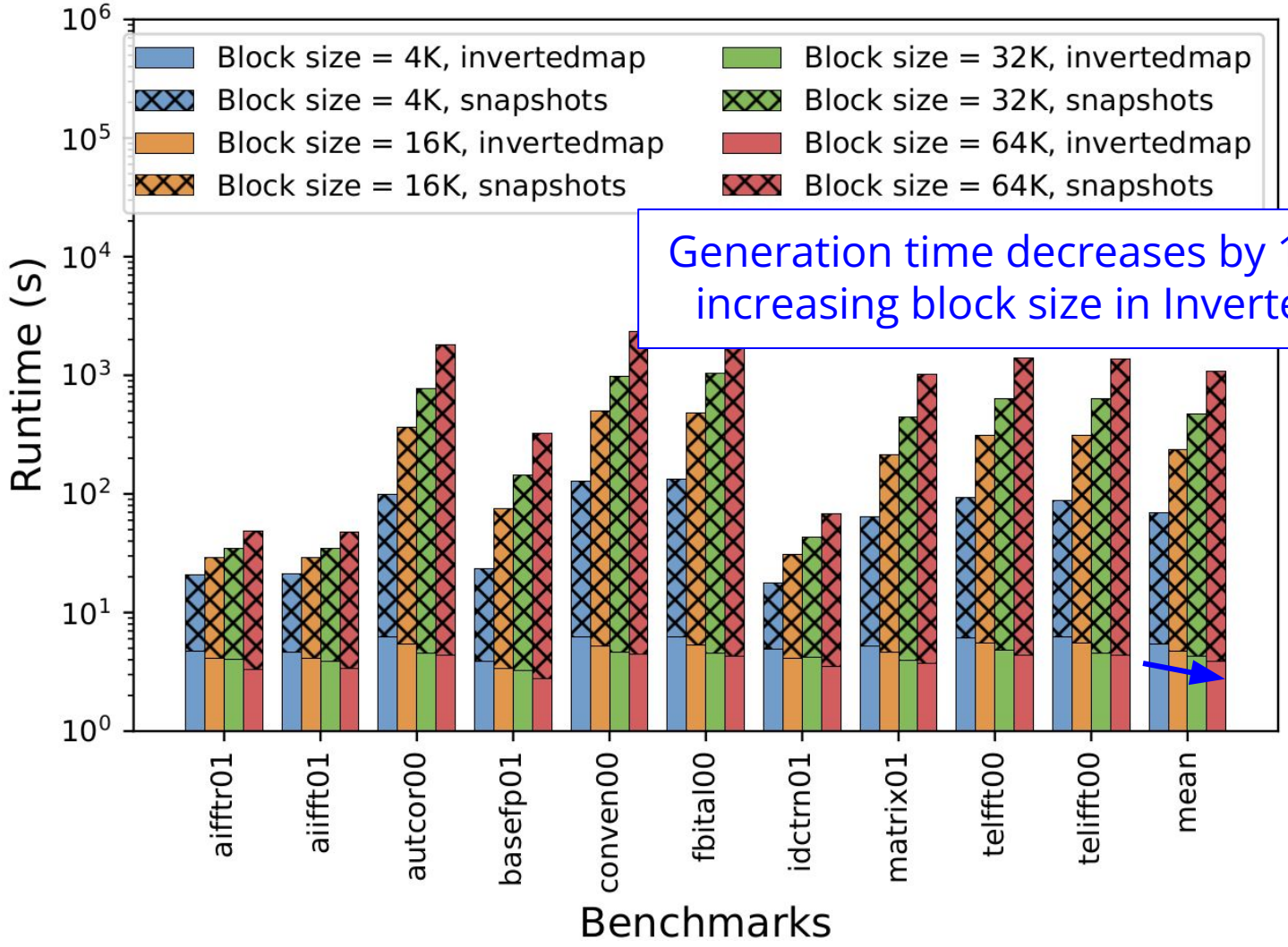
Results

Minion Generation Times



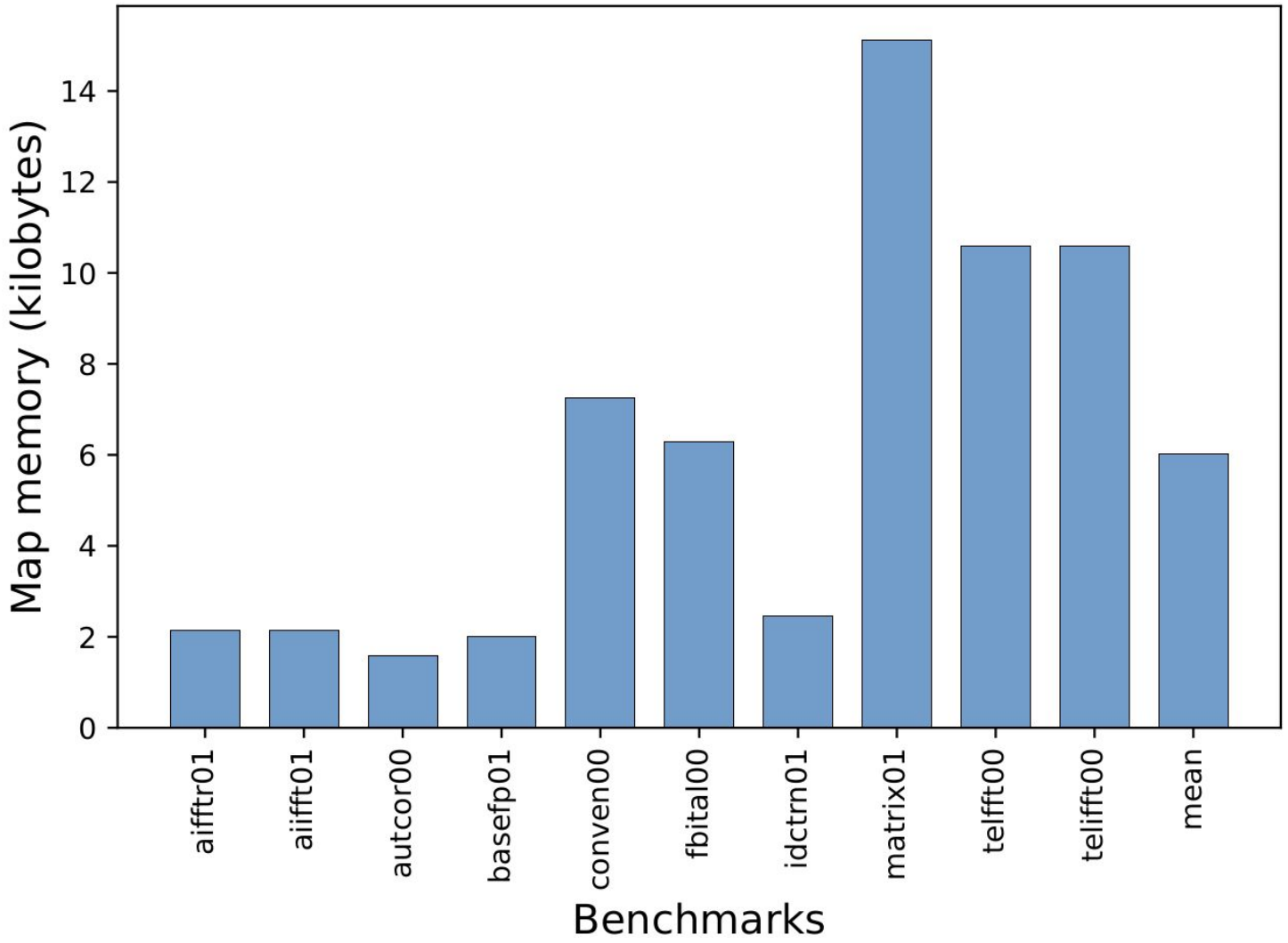
Results

Minion Generation Times



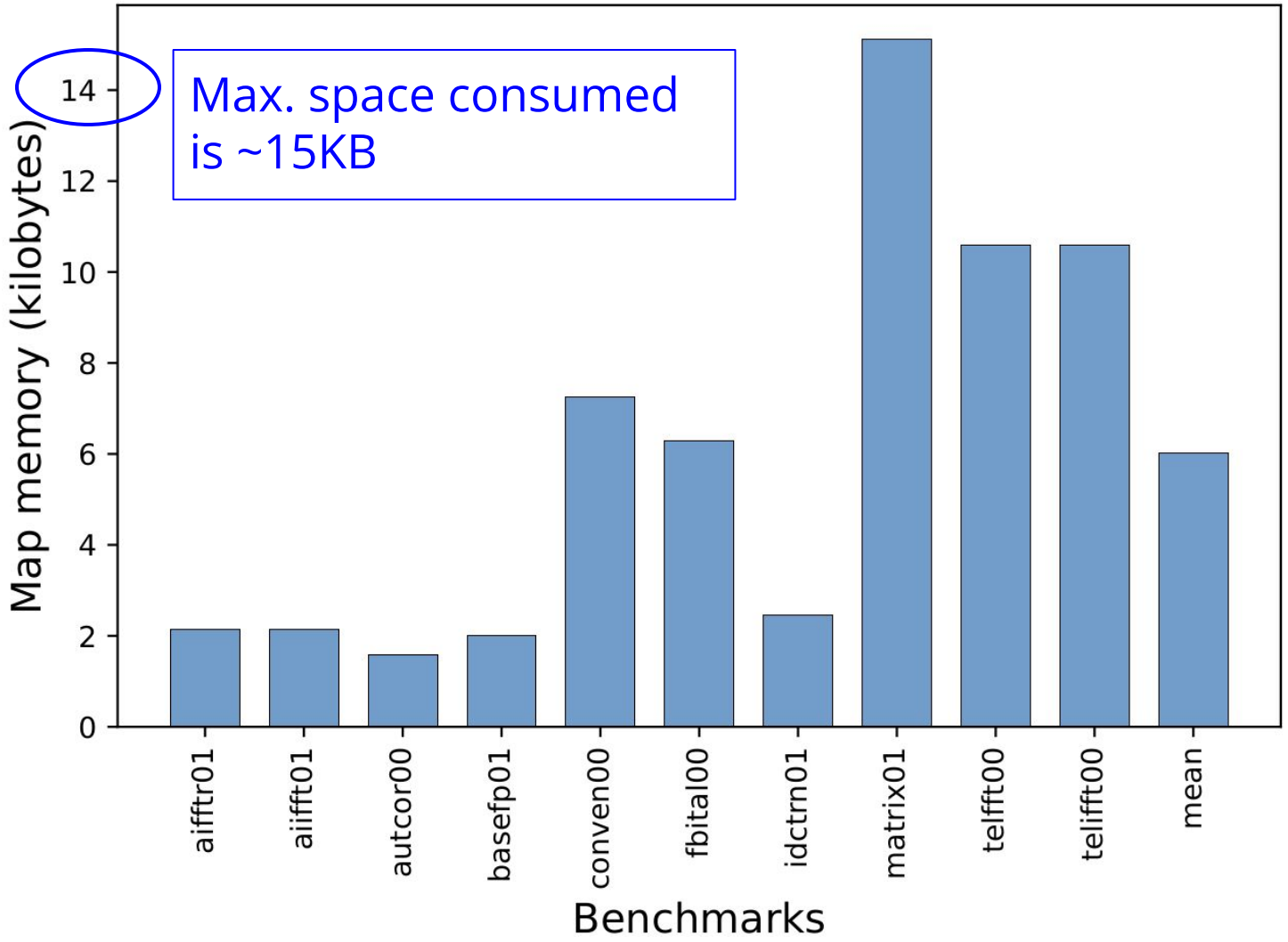
Results

Space Consumed by Inverted Map



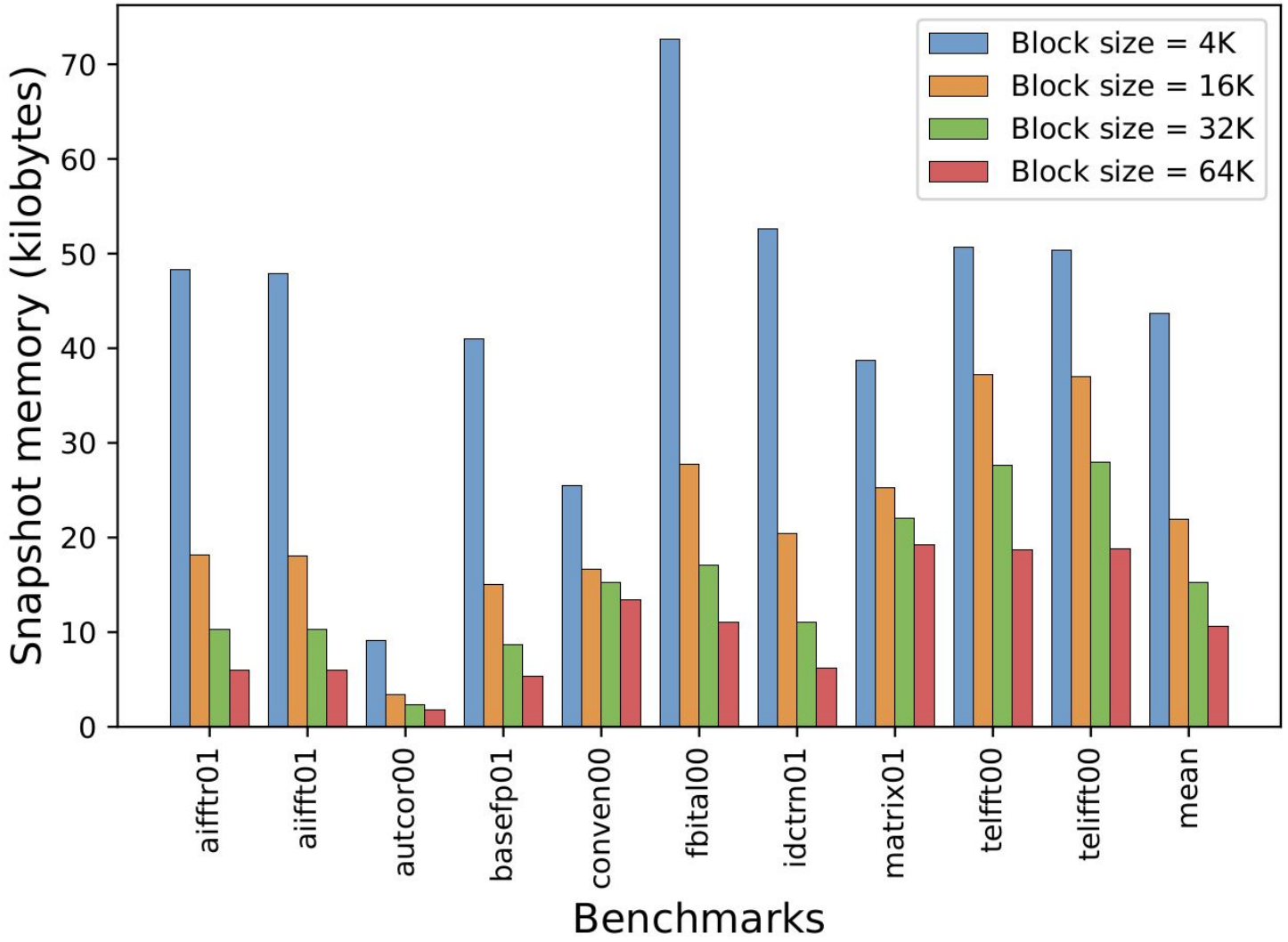
Results

Space Consumed by Inverted Map



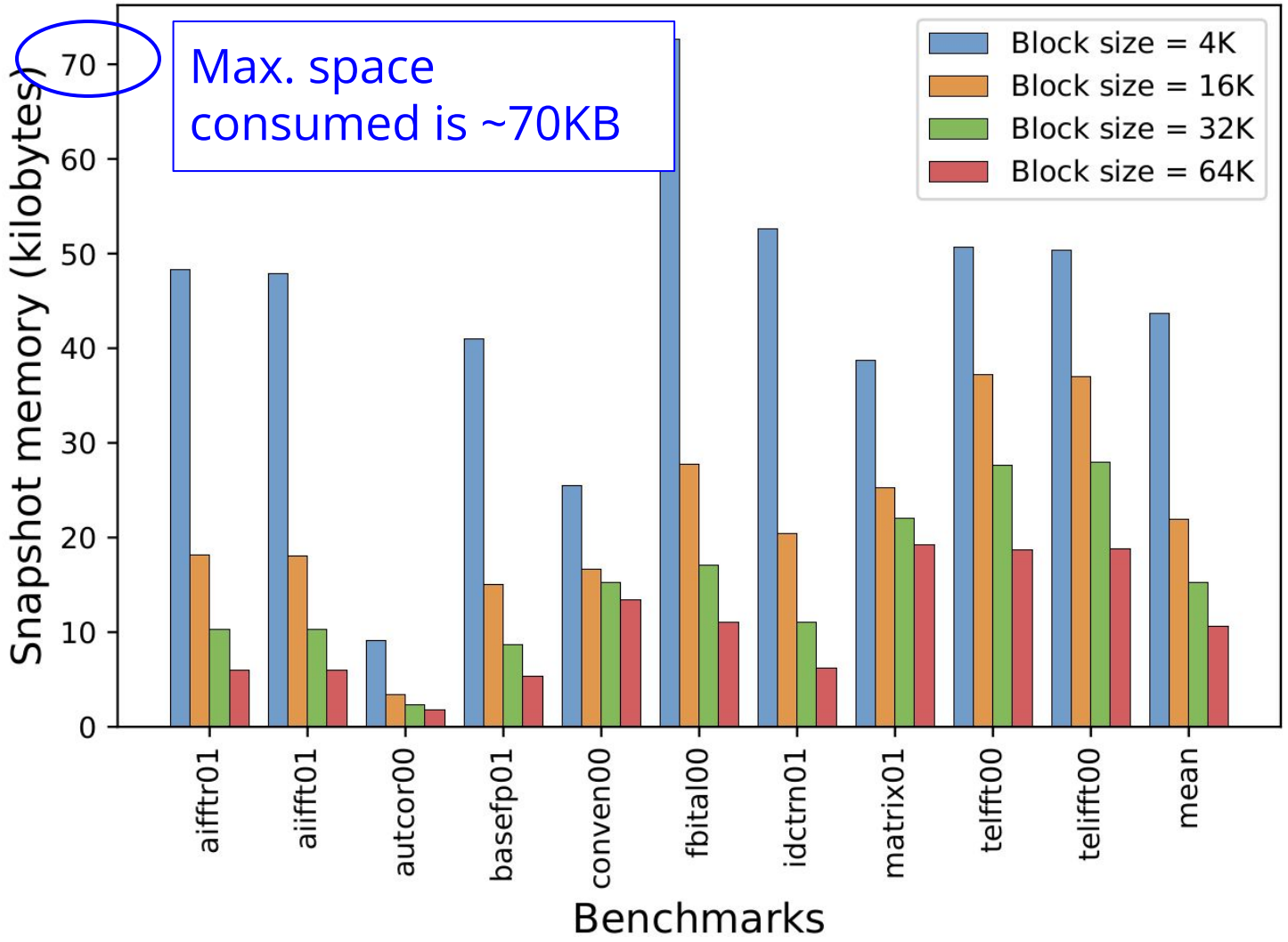
Results

Space Consumed by Snapshots (cumulative)



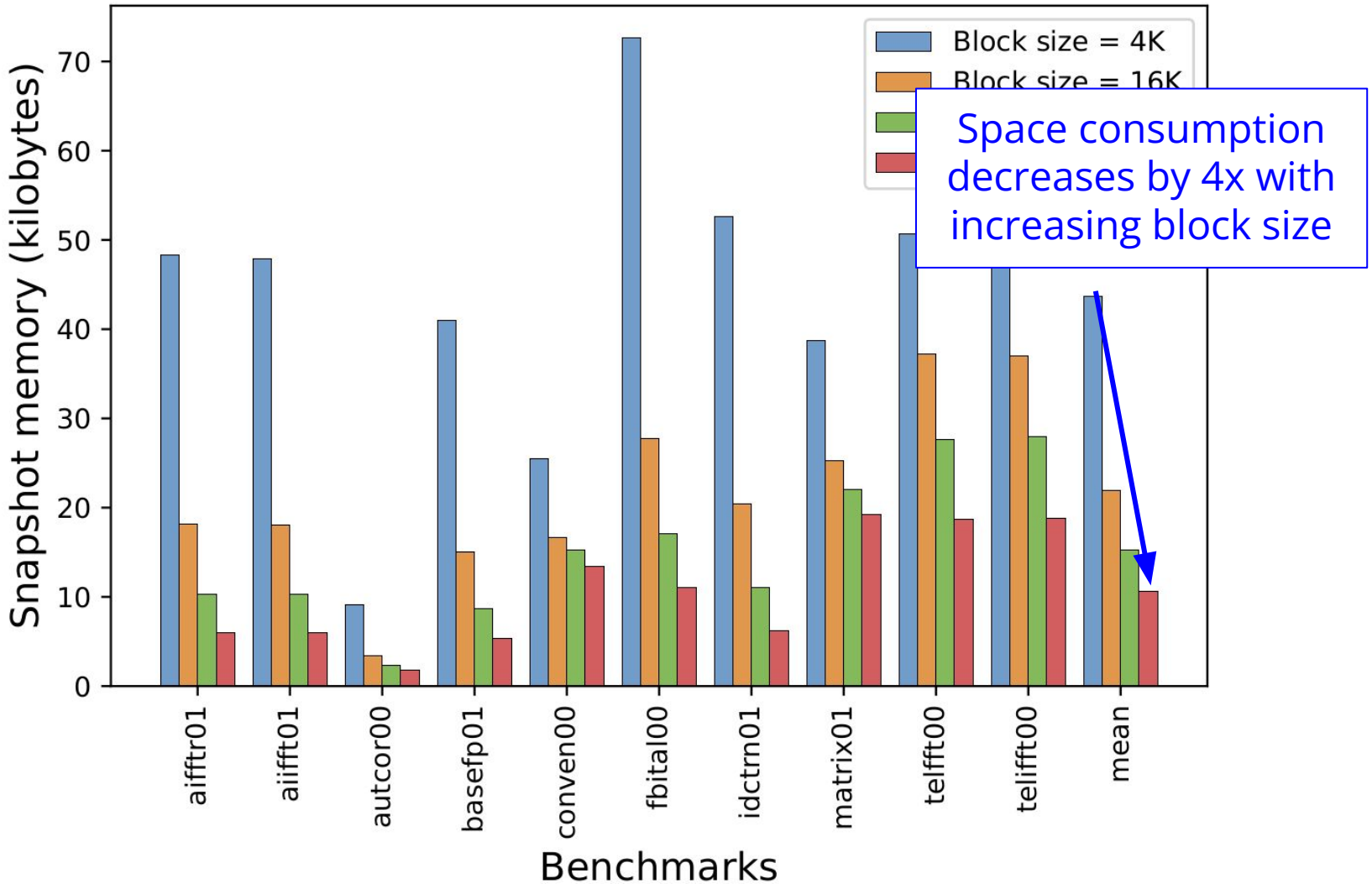
Results

Space Consumed by Snapshots (cumulative)



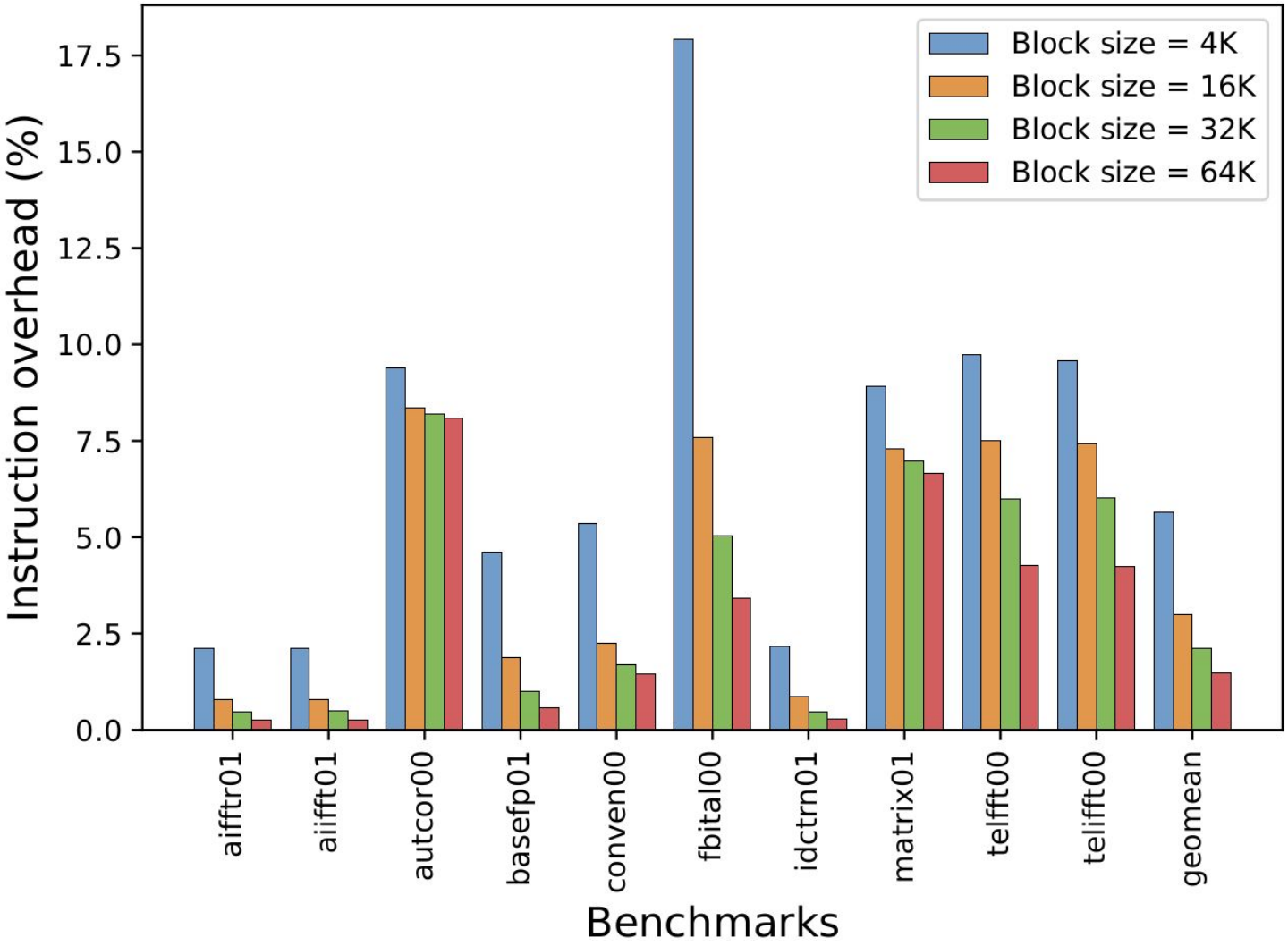
Results

Space Consumed by Snapshots (cumulative)



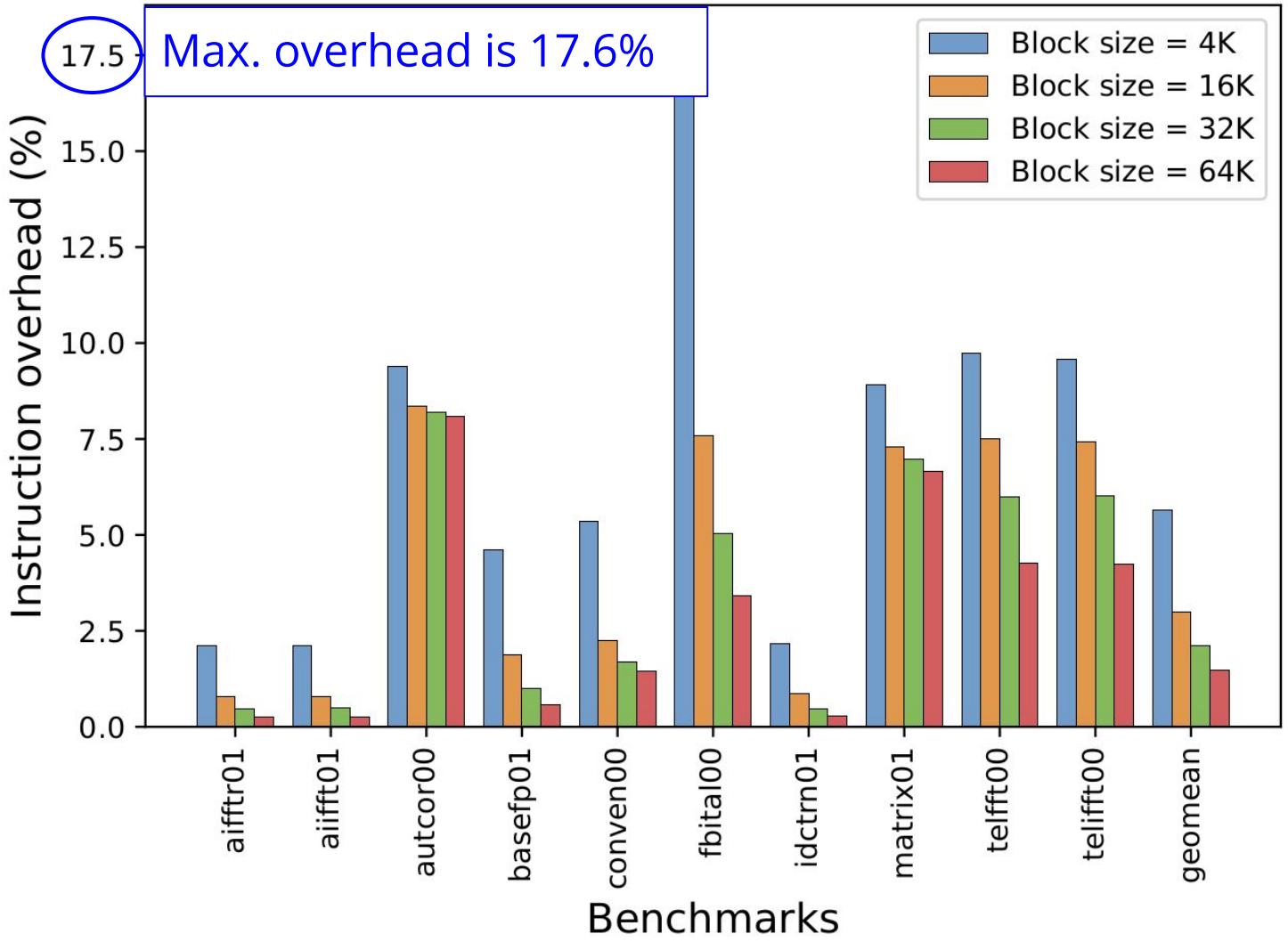
Results

Overhead of State Restoration Instructions



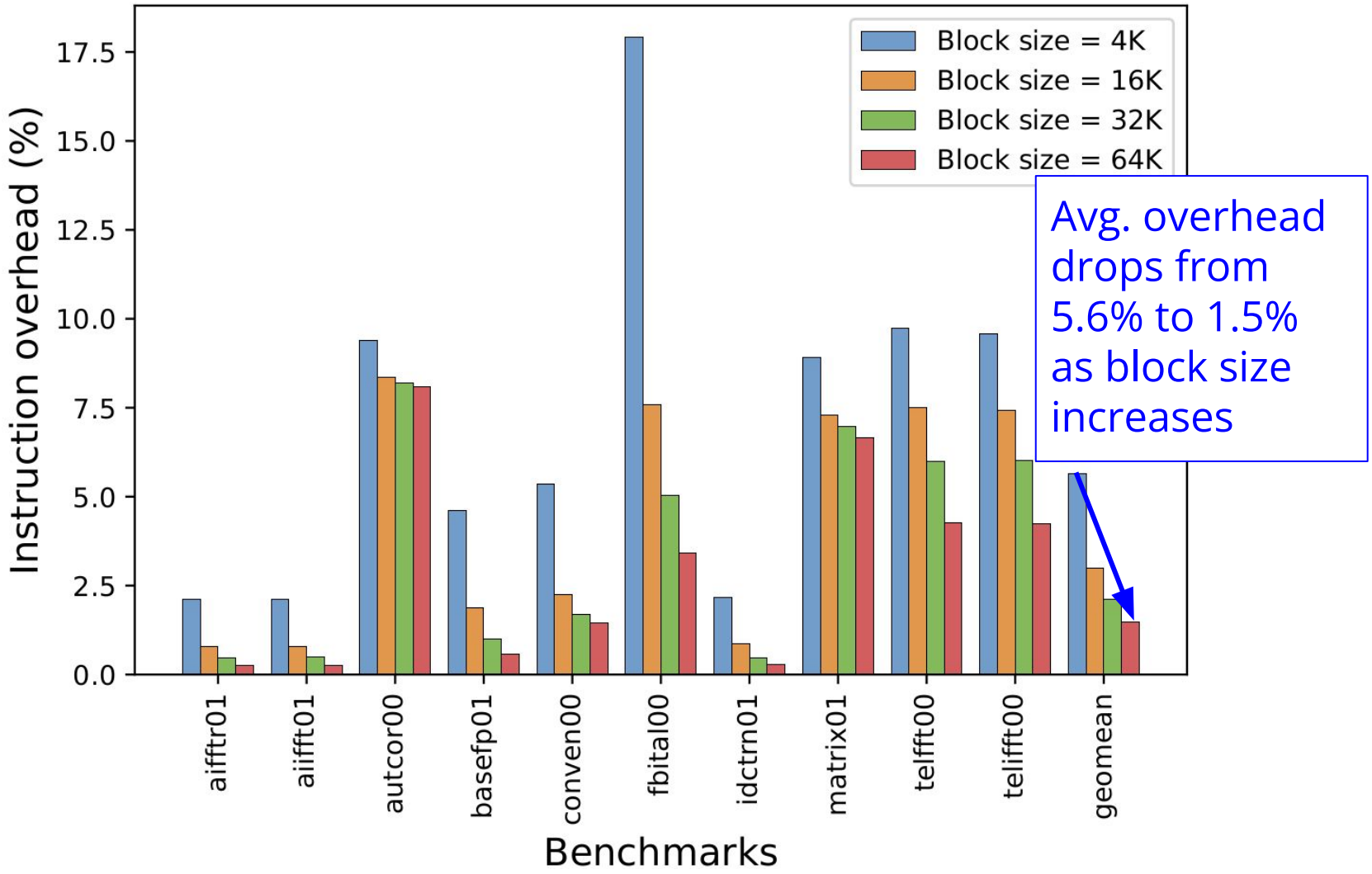
Results

Overhead of State Restoration Instructions



Results

Overhead of State Restoration Instructions



Conclusion

- We proposed a tool called, Gru, that
  - generates Minion executables (of parameterizable size) which reproduce behavior observed in the corresponding sections of an earlier run of the system
  - relies only on execution traces captured during the earlier execution
  - adds only 1.5% additional instructions

Conclusion

- Benefits:
  - Supports privacy-aware bug replication
  - Debugging activities parallelized
  - Bug replication using smaller executables makes it amenable to use other tools for bug localization and root-cause analysis

- Current Limitations and Future Directions
  - Micro-architectural State restoration
  - Replicating multi-core behavior

Conclusion

- Benefits:
  - Supports privacy-aware bug replication
  - Debugging activities parallelized
  - Bug replication using smaller executables makes it amenable to use other tools for bug localization and root-cause analysis

- Current Limitations and Future Directions
  - Micro-architectural State restoration
  - Replicating multi-core behavior

Thank You!

sandeepchandran@iitpkd.ac.in
rajshekar.k@iitdh.ac.in