InstantCheck: Checking the determinism of parallel programs using on-the-fly incremental hashing

Adrian Nistor, Darko Marinov, Josep Torrellas

MICRO '10

i-acoma group

Problem: Nondeterminism

Problem: Nondeterminism

- Parallel programming is difficult
  - One input → many different outputs

Problem: Nondeterminism

- Parallel programming is difficult
  - One input → many different outputs
- However, programmers like serial thinking
  - One input → one output

Problem: Nondeterminism

- Parallel programming is difficult
  - One input → many different outputs
- However, programmers like serial thinking
  - One input → one output
- The problem with one input, many outputs
  - **Uncertain** what your own code does

Problem: Nondeterminism

- Parallel programming is difficult
  - One input → many different outputs
- However, programmers like serial thinking
  - One input → one output
- The problem with one input, many outputs
  - **Uncertain** what your own code does
- The problem: Nondeterminism
  - Due to parallel execution
  - Presume input is fixed (like in serial execution)

External determinism

Capturing state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

Conclusions

Our proposal: External Determinism

Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state

Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state

- Determinism that we are currently getting
  - Fix order of inter-thread communications

Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state
  - **External Determinism** – our proposal
- Determinism that we are currently getting
  - Fix order of inter-thread communications

Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state
  - **External Determinism** – our proposal
- Determinism that we are currently getting
  - Fix order of inter-thread communications
  - **Internal Determinism** – some of the current proposals

Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state
  - **External Determinism** – our proposal
- Determinism that we are currently getting
  - Fix order of inter-thread communications
  - **Internal Determinism** – some of the current proposals
- External determinism includes Internal determinism

Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state
  - **External Determinism** – our proposal
- Determinism that we are currently getting
  - Fix order of inter-thread communications
  - **Internal Determinism** – some of the current proposals
- External determinism includes Internal determinism
  - If fix order of communication — Internal
    - You implicitly get same output memory state — External

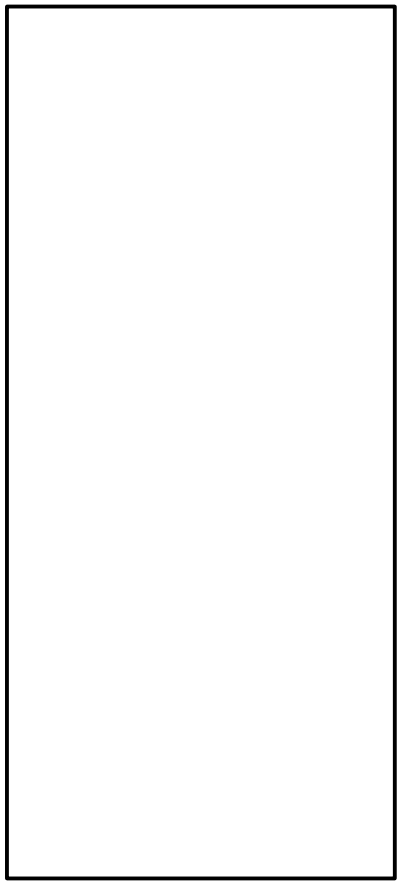
Our proposal: External Determinism

- Determinism that we would like (for relevant algorithms)
  - Same input MEM state → same output MEM state
  - **External Determinism** – our proposal
- Determinism that we are currently getting
  - Fix order of inter-thread communications
  - **Internal Determinism** – some of the current proposals
- External determinism includes Internal determinism
  - If fix order of communication — Internal
    - You implicitly get same output memory state — External
  - But can get same output memory state — External
    - Without enforcing inter-thread communication order — Internal

Example

Example

CODE



Example

CODE

Global G

Example

CODE

Global	**G**
Local	**L**

Example

CODE

Global G
Local L

G = G + L

Example

CODE

```
Global      G
Local       L


LOCK


  G = G + L


UN__LOCK
```

Example

CODE

EXEC—1

```
Global      G
Local       L

LOCK

  G = G + L

UN__LOCK
```

Example

CODE **EXEC—1** **EXEC—2**

```
Global      G
Local       L


LOCK


  G = G + L


UN__LOCK
```

Example

CODE

```
Global      G
Local       L

LOCK

 G = G + L

UN__LOCK
```

EXEC—1

G == 2

EXEC—2

G == 2

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7

EXEC—2

G == 2

L_0 == 7

Example

CODE

```
Global      G
Local       L


LOCK


 G = G + L


UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 L_1 == 3

EXEC—2

G == 2

L_0 == 7 L_1 == 3

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 L_1 == 3

thread 0

EXEC—2

G == 2

L_0 == 7 L_1 == 3

thread 0

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7	L_1 == 3
thread 0	thread 1

EXEC—2

G == 2

L_0 == 7	L_1 == 3
thread 0	thread 1

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7	L_1 == 3
thread 0	thread 1
	G = 2 + 7

EXEC—2

G == 2

L_0 == 7	L_1 == 3
thread 0	thread 1

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7	L_1 == 3
thread 0	thread 1
G = 2 + 7	
	G = 9 + 3

EXEC—2

G == 2

L_0 == 7	L_1 == 3
thread 0	thread 1

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

G = 5 + 7

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

G = 5 + 7

G == 12

Example

CODE

Global G
Local L

LOCK

G = G + L

UN__LOCK

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

G = 5 + 7

G == 12

SAME STATE

Example

CODE

```
Global        G
Local         L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

G = 5 + 7

G == 12

SAME STATE

Execution order does not matter

Example

CODE

```
Global      G
Local       L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

G = 5 + 7

G == 12

SAME STATE = EXTERNAL DET

Execution order does not matter

Example

CODE

```
Global        G
Local         L

LOCK

G = G + L

UN__LOCK
```

EXEC—1

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 7

G = 9 + 3

G == 12

EXEC—2

G == 2

L_0 == 7 | L_1 == 3

thread 0 | thread 1

G = 2 + 3

G = 5 + 7

G == 12

SAME STATE = EXTERNAL DET

Execution order does not matter = NO internal det

Internal Determinism

Internal Determinism

- Internal determinism is **extremely intuitive**

Internal Determinism

- Internal determinism is **extremely intuitive**
- But has costs

Internal Determinism

- Internal determinism is **extremely intuitive**
- But has costs
- Hardware cost
  - Special HW that commits transactions in order

Internal Determinism

- Internal determinism is **extremely intuitive**
- But has costs
- Hardware cost
  - Special HW that commits transactions in order
- Programmability cost
  - Disallow some synchronization (e.g., locks)
  - Deterministic languages

Internal Determinism

- Internal determinism is **extremely intuitive**
- But has costs
- Hardware cost
  - Special HW that commits transactions in order
- Programmability cost
  - Disallow some synchronization (e.g., locks)
  - Deterministic languages
- Performance cost
  - Software runtime fixes order – slows down execution

InstantCheck

InstantCheck

- Check external determinism

InstantCheck

- Check external determinism
- **Fast checking of memory–state identity**
  - **On-the-fly incremental hashing**

InstantCheck

- Check external determinism
- **Fast checking of memory–state identity**
  - **On-the-fly incremental hashing**
- Do the checking during code testing

InstantCheck

- Check external determinism
- **Fast checking of memory–state identity**
  - **On-the-fly incremental hashing**
- Do the checking during code testing
- Parallel code testing already runs the code many times
  - Piggyback on these many runs
  - Insert some very fast checks for external determinism

InstantCheck

- Check external determinism
- **Fast checking of memory–state identity**
  - **On-the-fly incremental hashing**
- Do the checking during code testing
- Parallel code testing already runs the code many times
  - Piggyback on these many runs
  - Insert some very fast checks for external determinism
- Check that memory states are identical
  - Program end, barriers, other points in the program

External determinism

Capture state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

Conclusions

How it works

How it works

- Have a hash of the Memory State

How it works

- Have a hash of the Memory State
- Each time memory is updated (via Store)

How it works

- Have a hash of the Memory State
- Each time memory is updated (via Store)
- Update the hash
  - To reflect the update to memory

How it works

- Have a hash of the Memory State
- Each time memory is updated (via Store)
- Update the hash
  - To reflect the update to memory
- At any time the hash already summarizes state
  - No need to compute it from scratch
  - No need to traverse the entire memory state

How it works

- Have a hash of the Memory State
- Each time memory is updated (via Store)
- Update the hash
  - To reflect the update to memory
- At any time the hash already summarizes state
  - No need to compute it from scratch
  - No need to traverse the entire memory state
- At any time, **Instantly ready** for identity checking

Example

Example

EXEC—1

Example

EXEC—1

G == 2

Example

EXEC—1

G == 2

thread 0

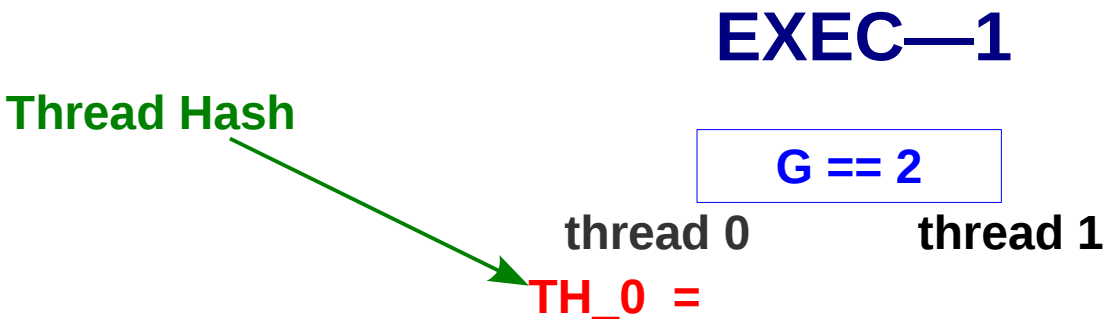
Example

EXEC—1

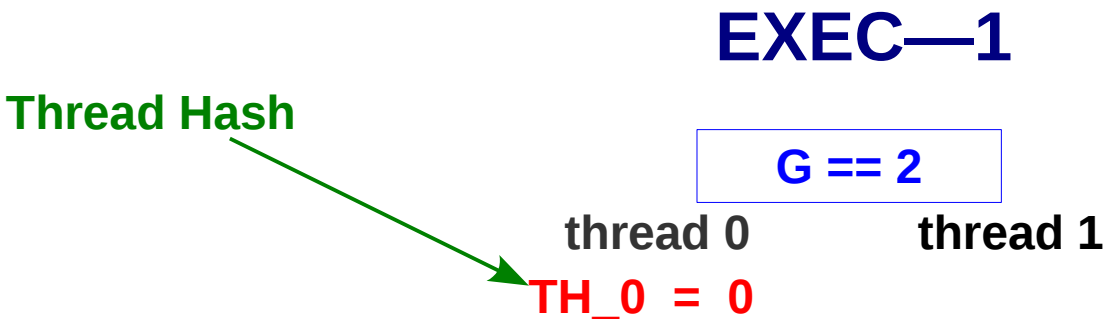
G == 2

thread 0 thread 1

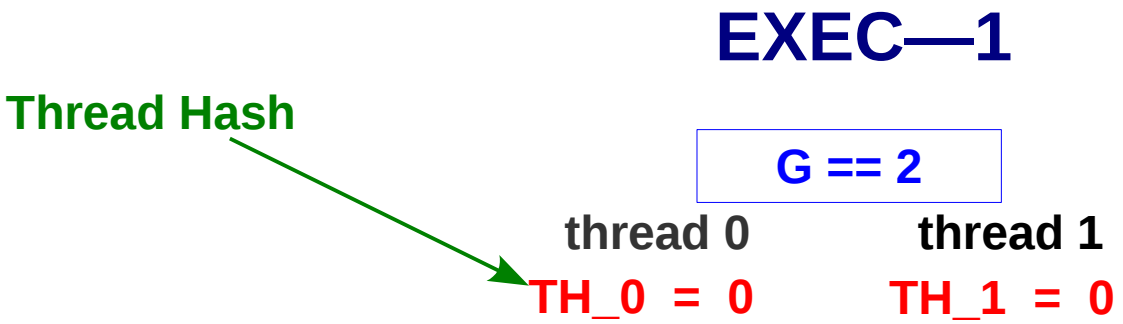
Example



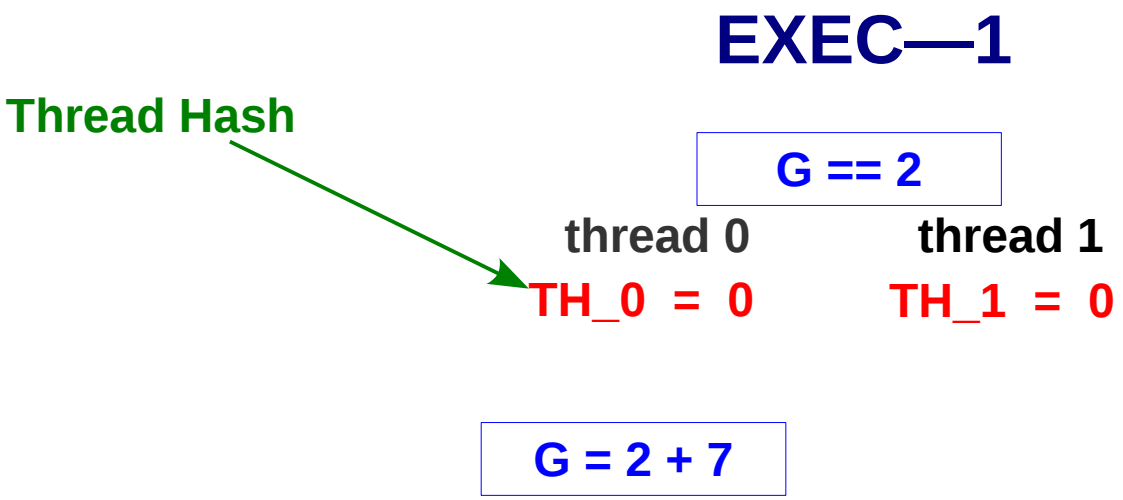
Example



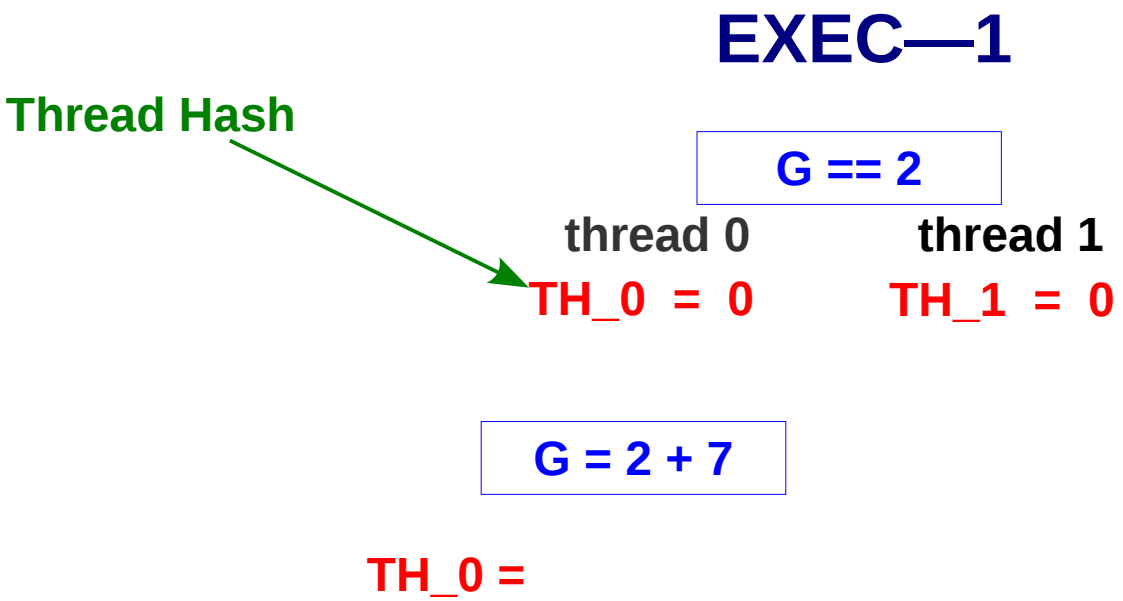
Example



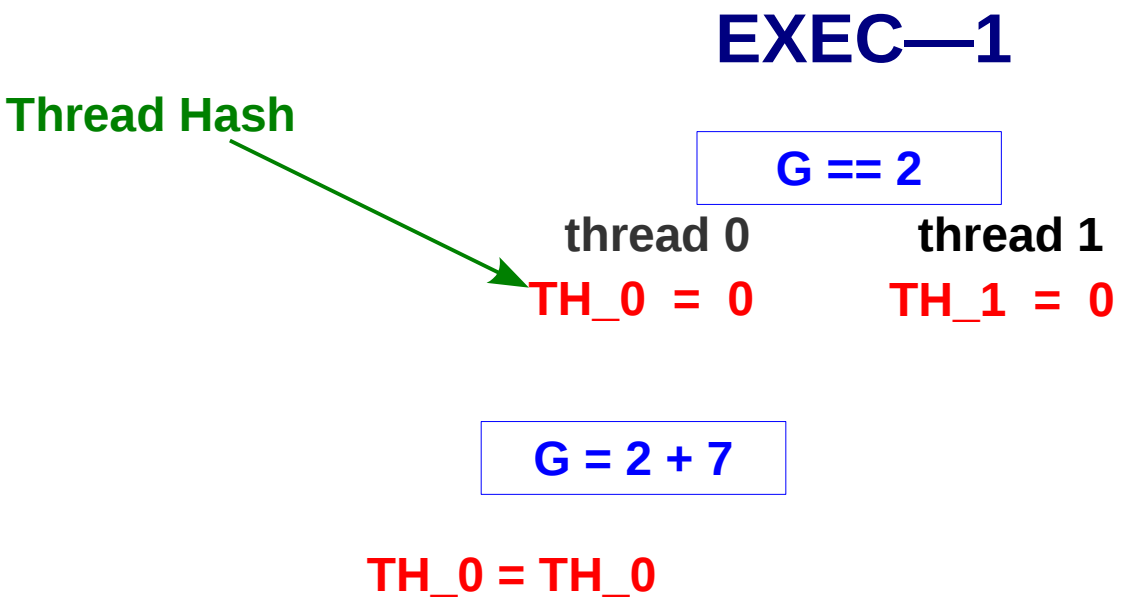
Example



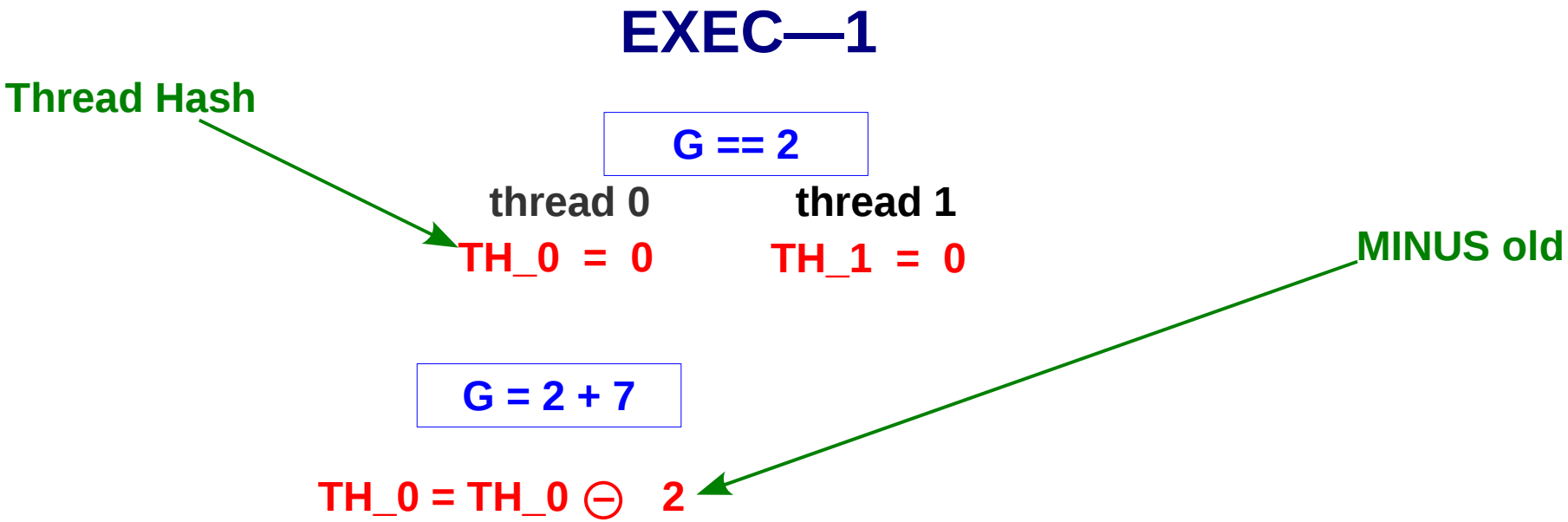
Example



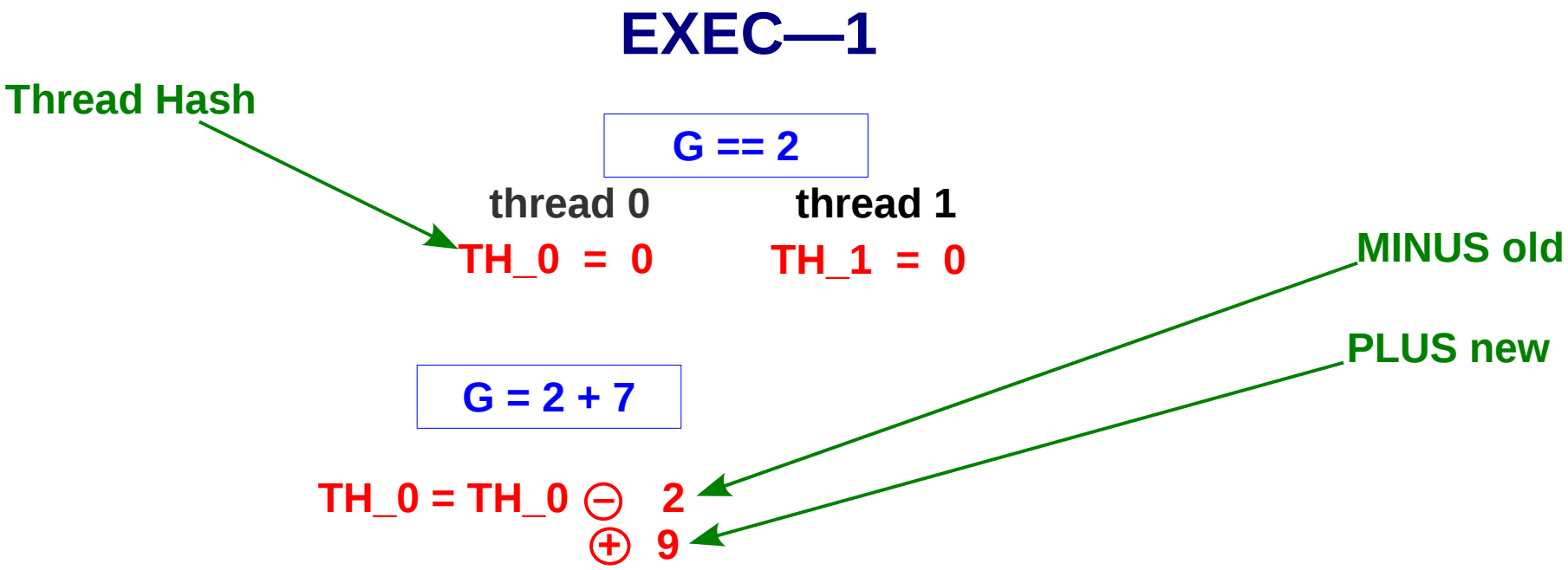
Example



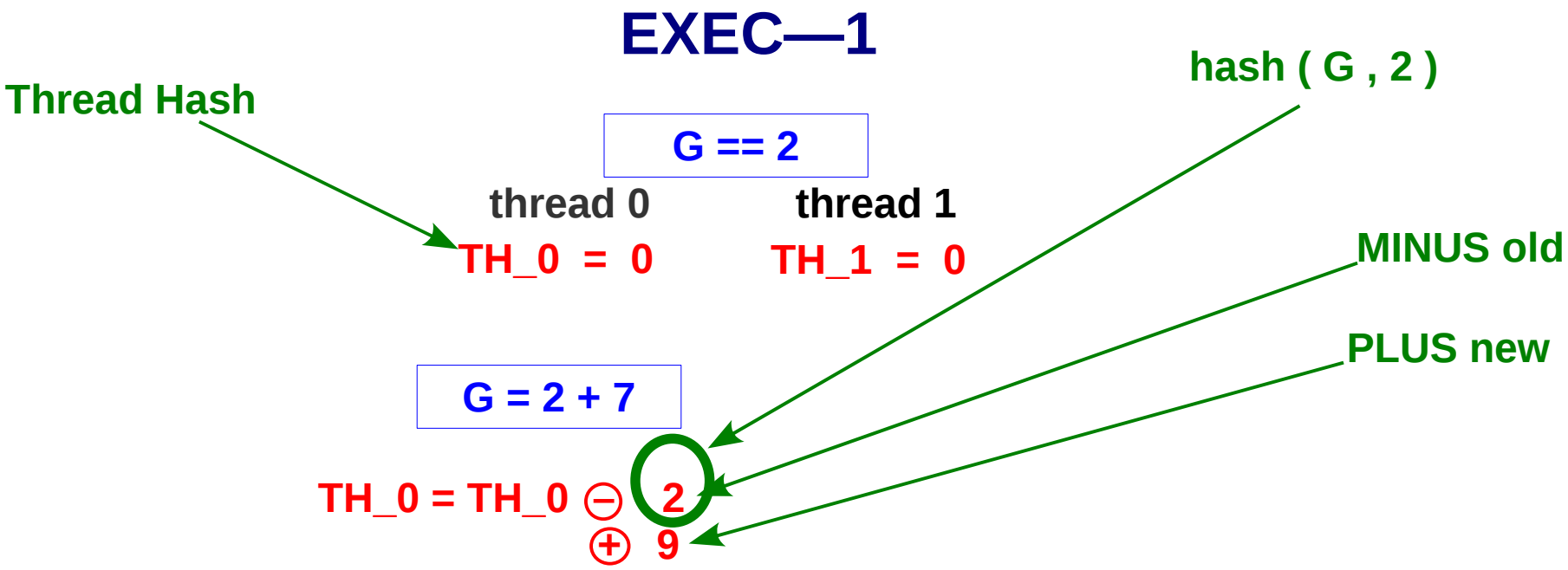
Example


EXEC—1
Thread Hash
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
G = 2 + 7
TH_0 = TH_0 ⊖ 2

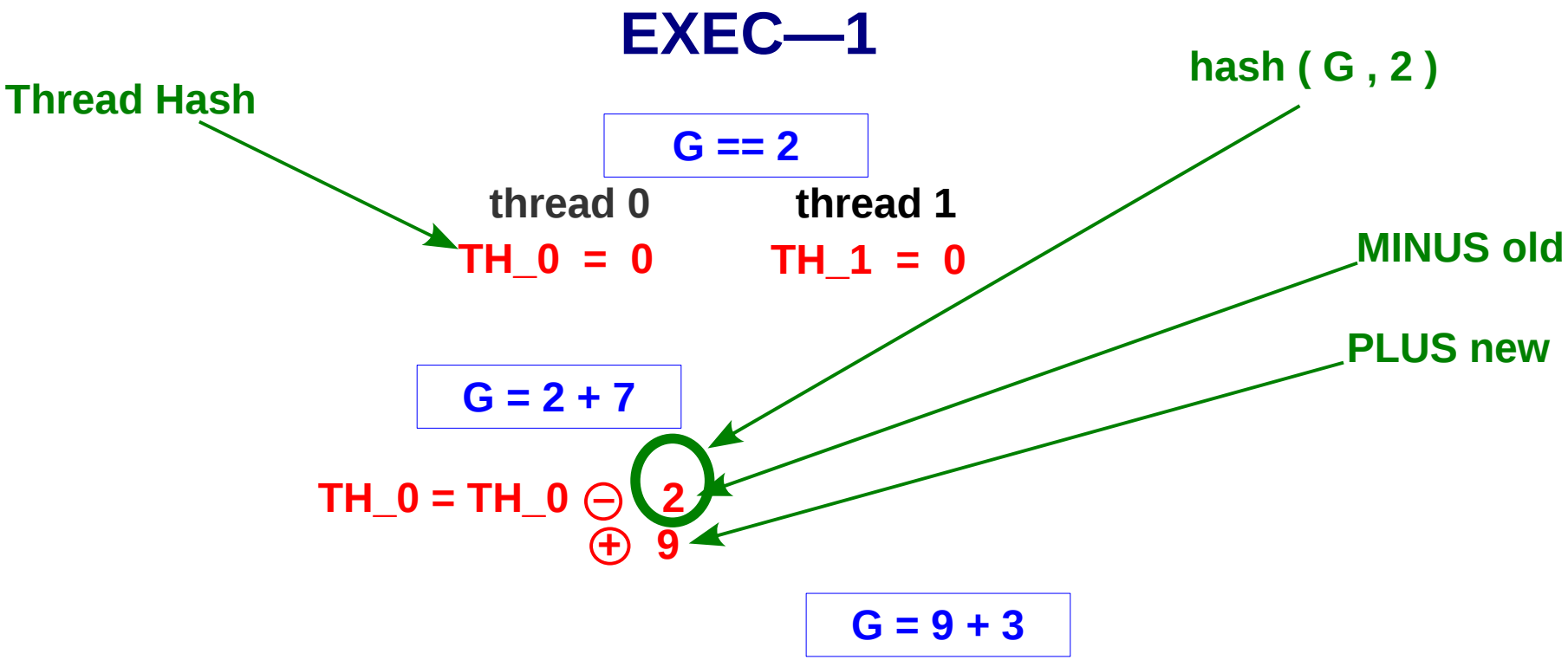
Example


EXEC—1
Thread Hash
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9

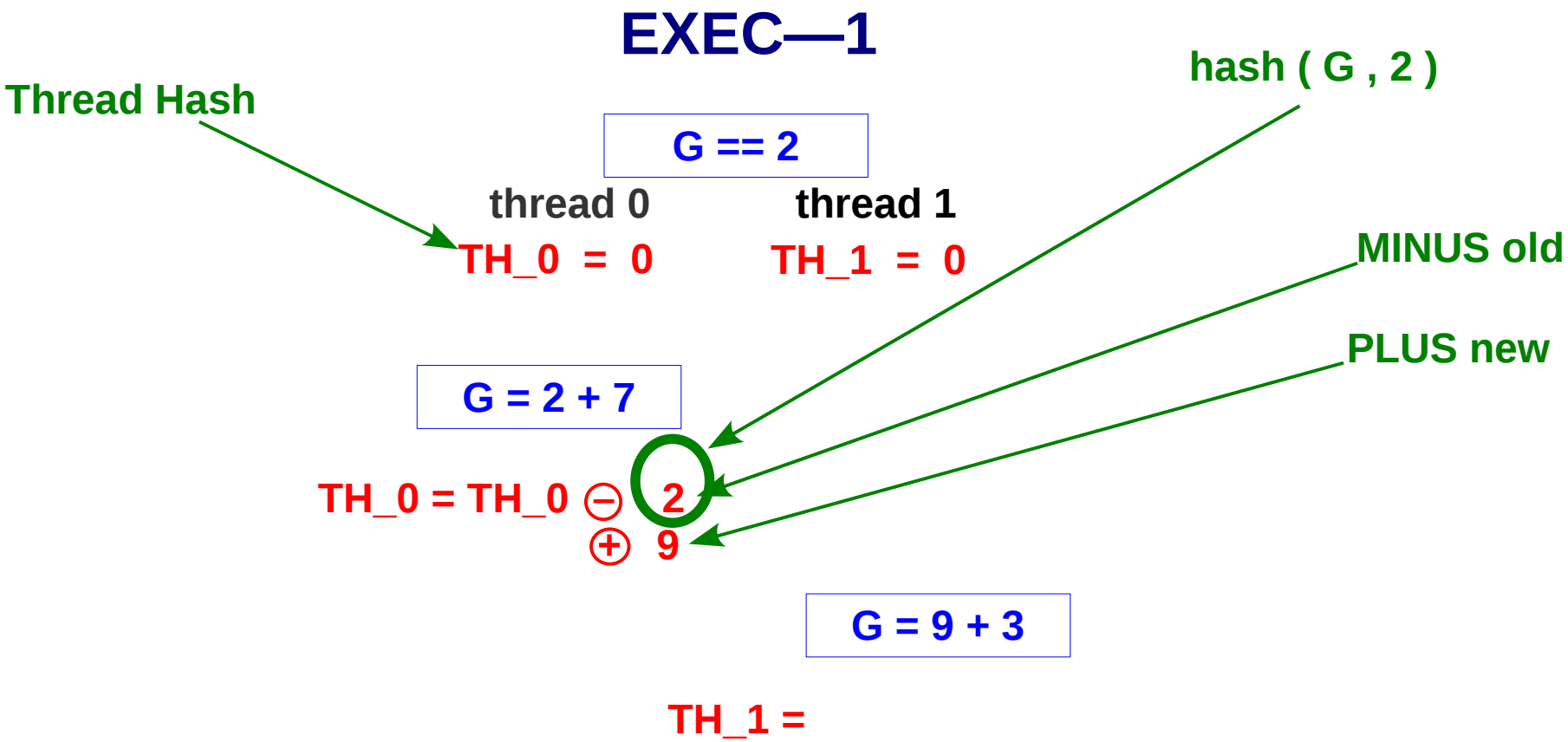
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9

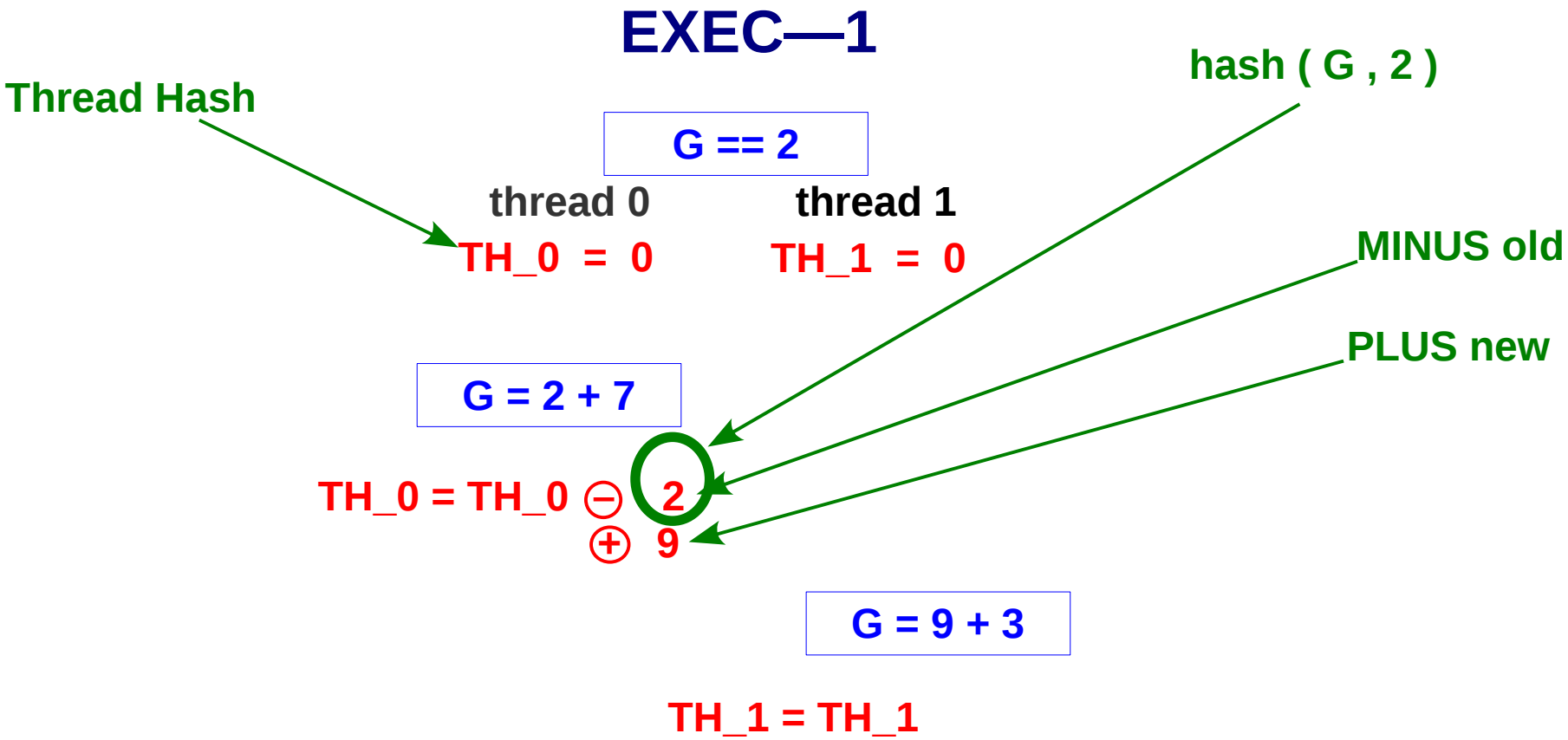
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3

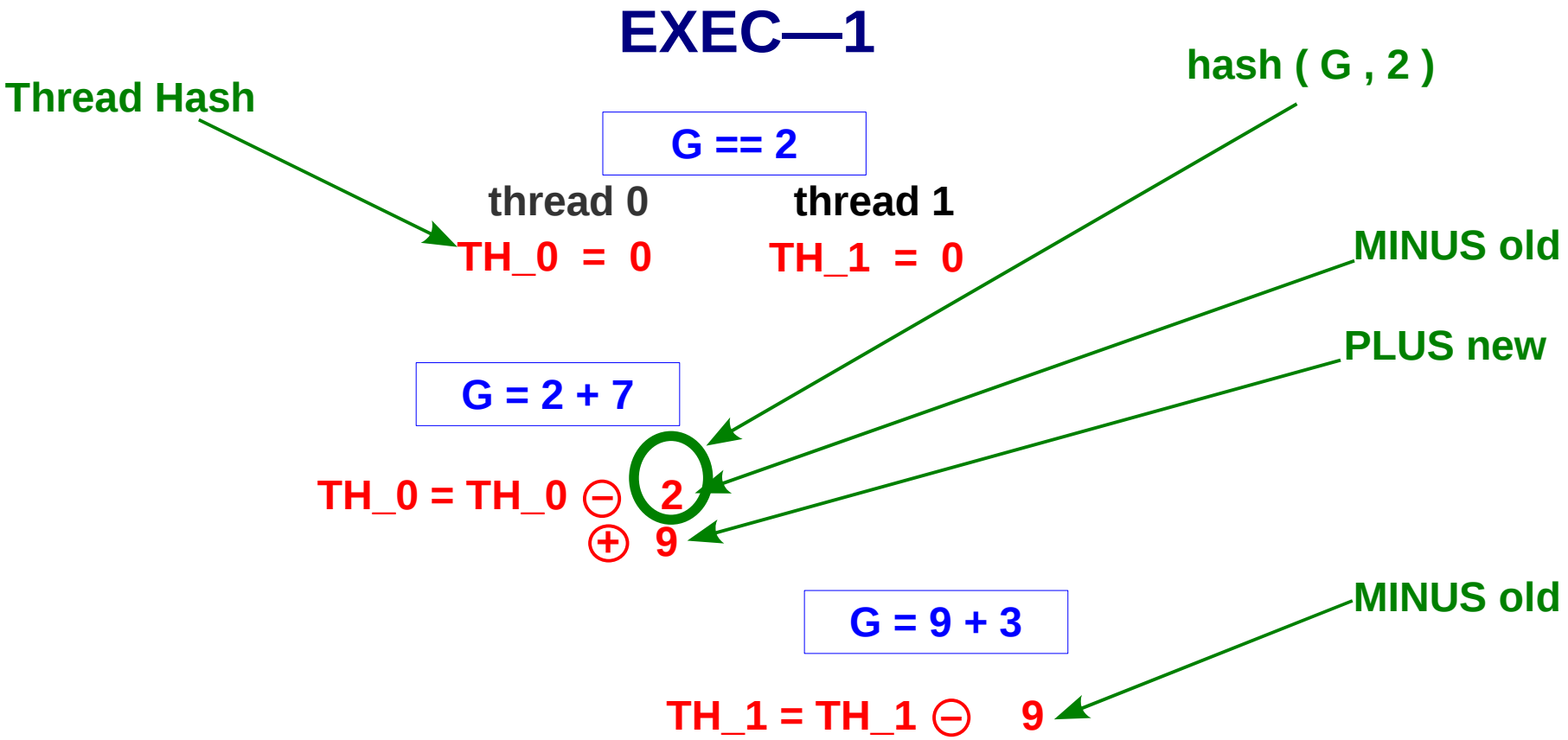
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
TH_1 =

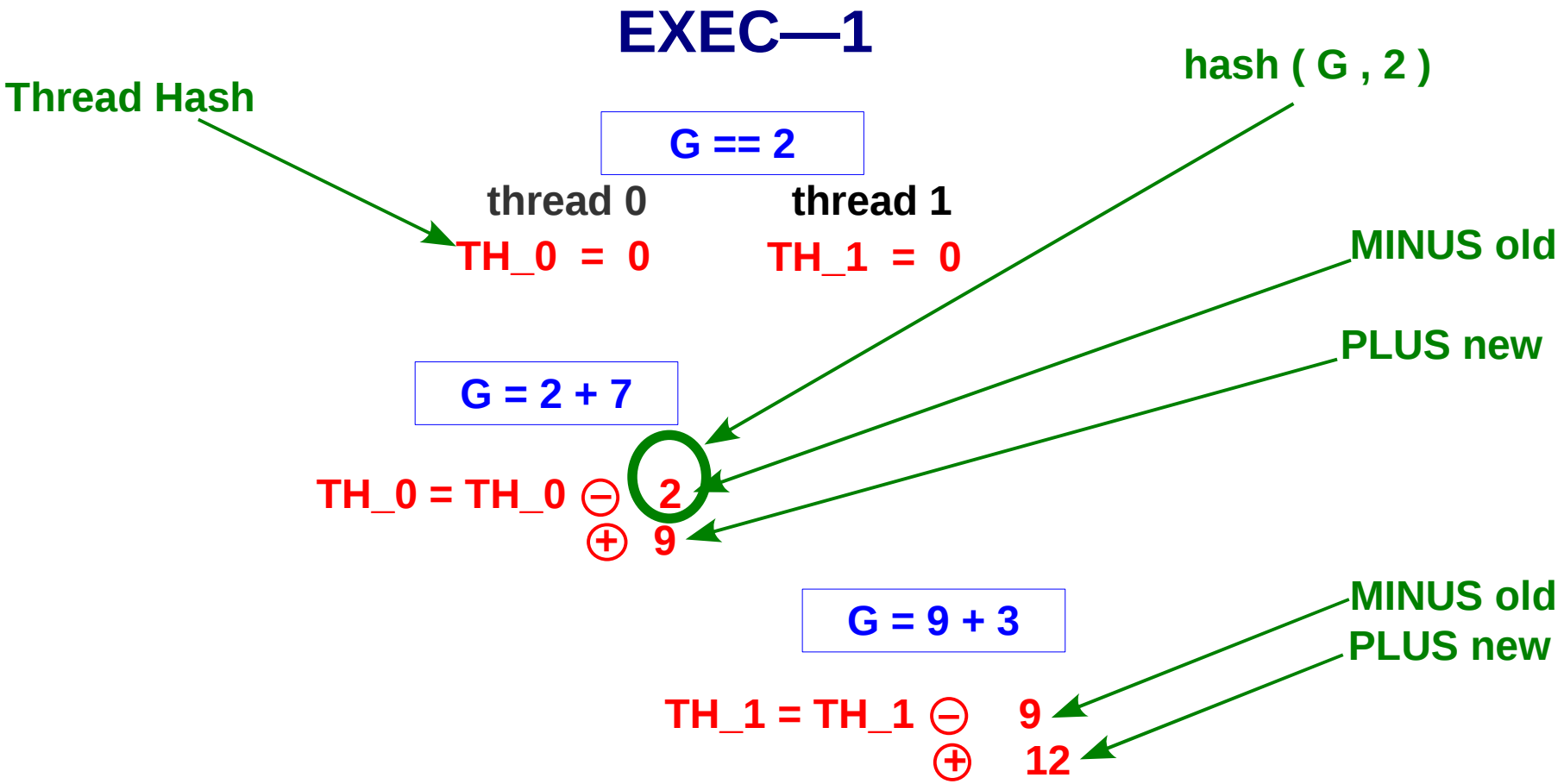
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
TH_1 = TH_1

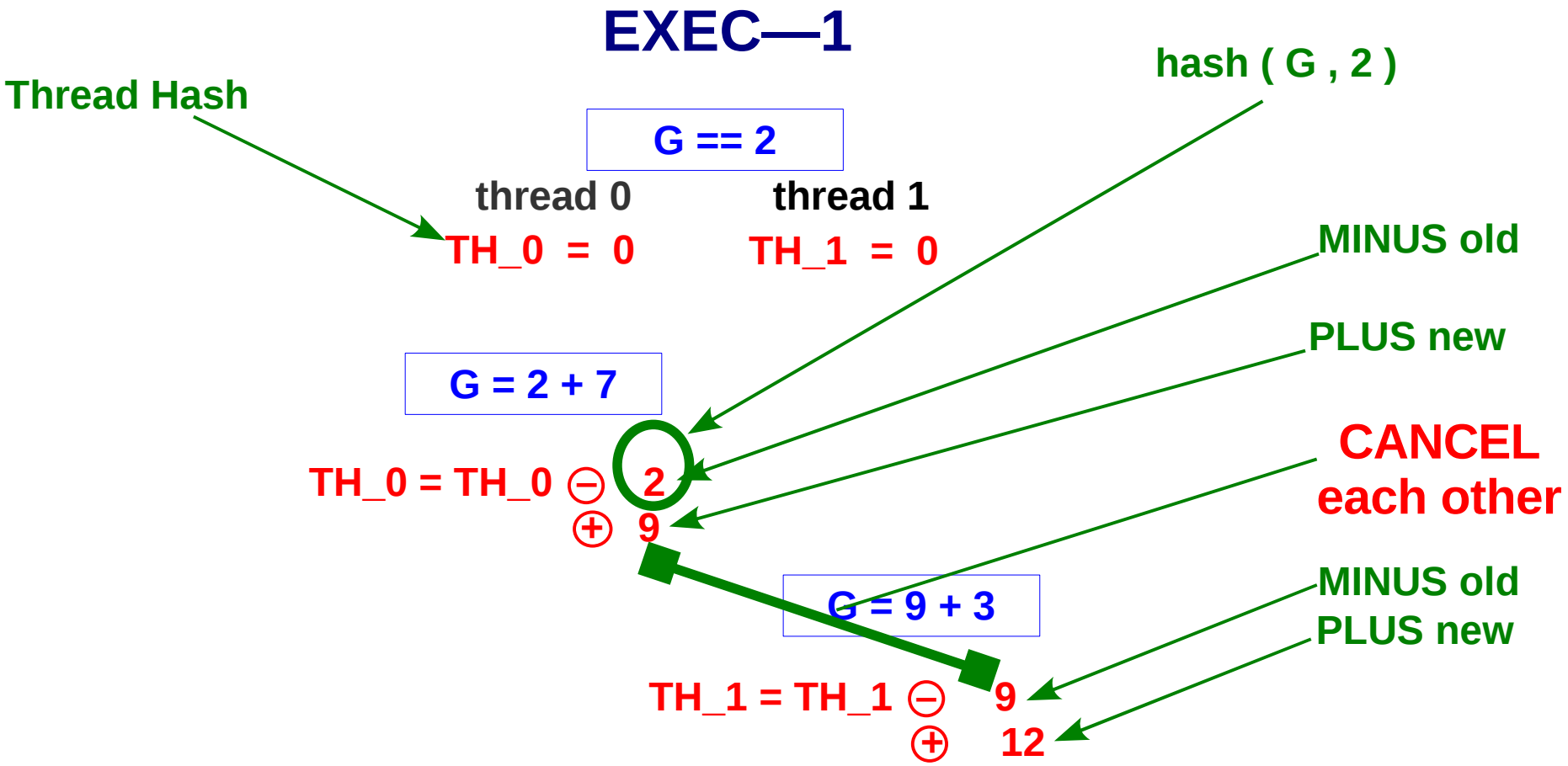
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
TH_1 = TH_1 ⊖ 9

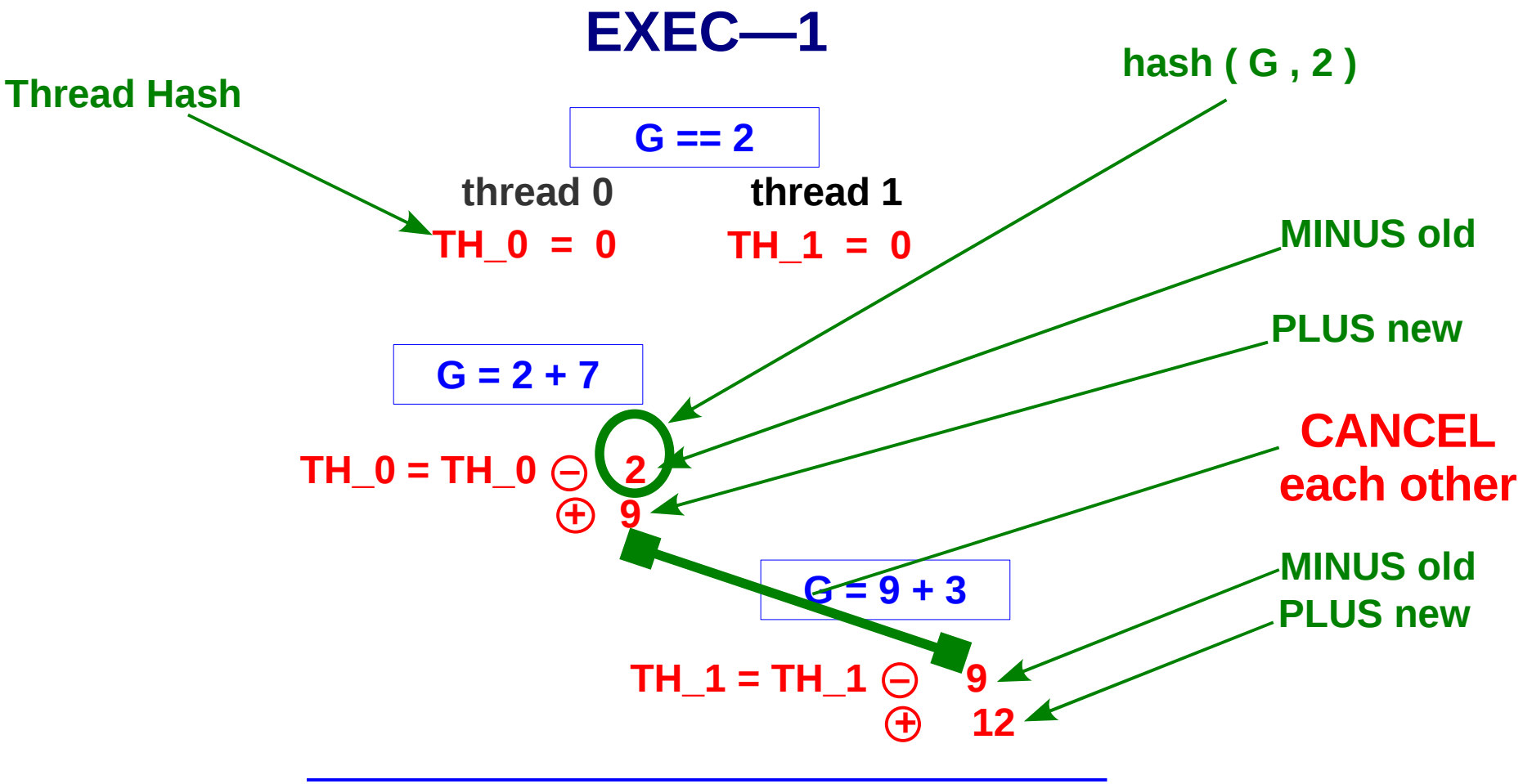
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12

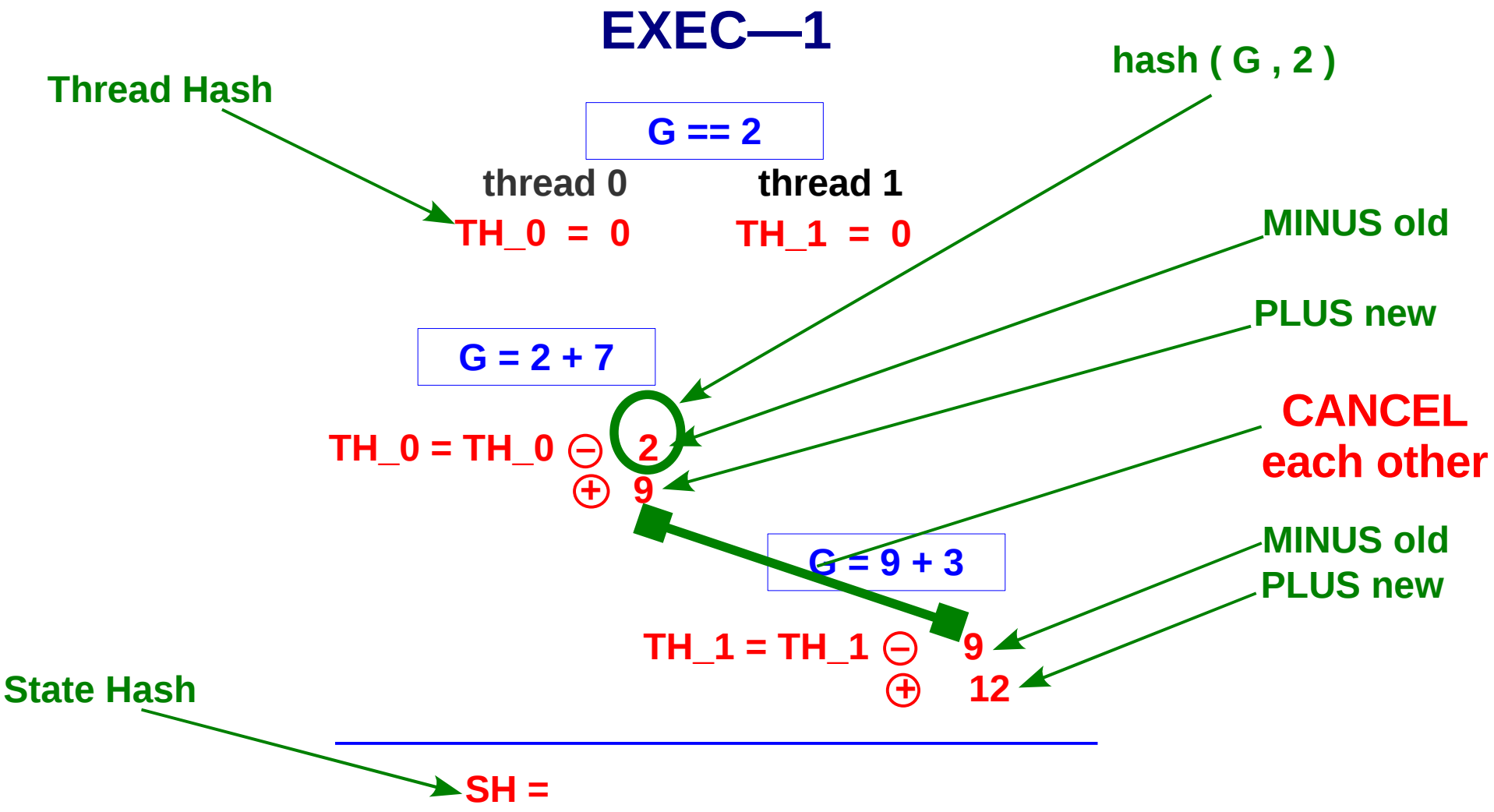
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12

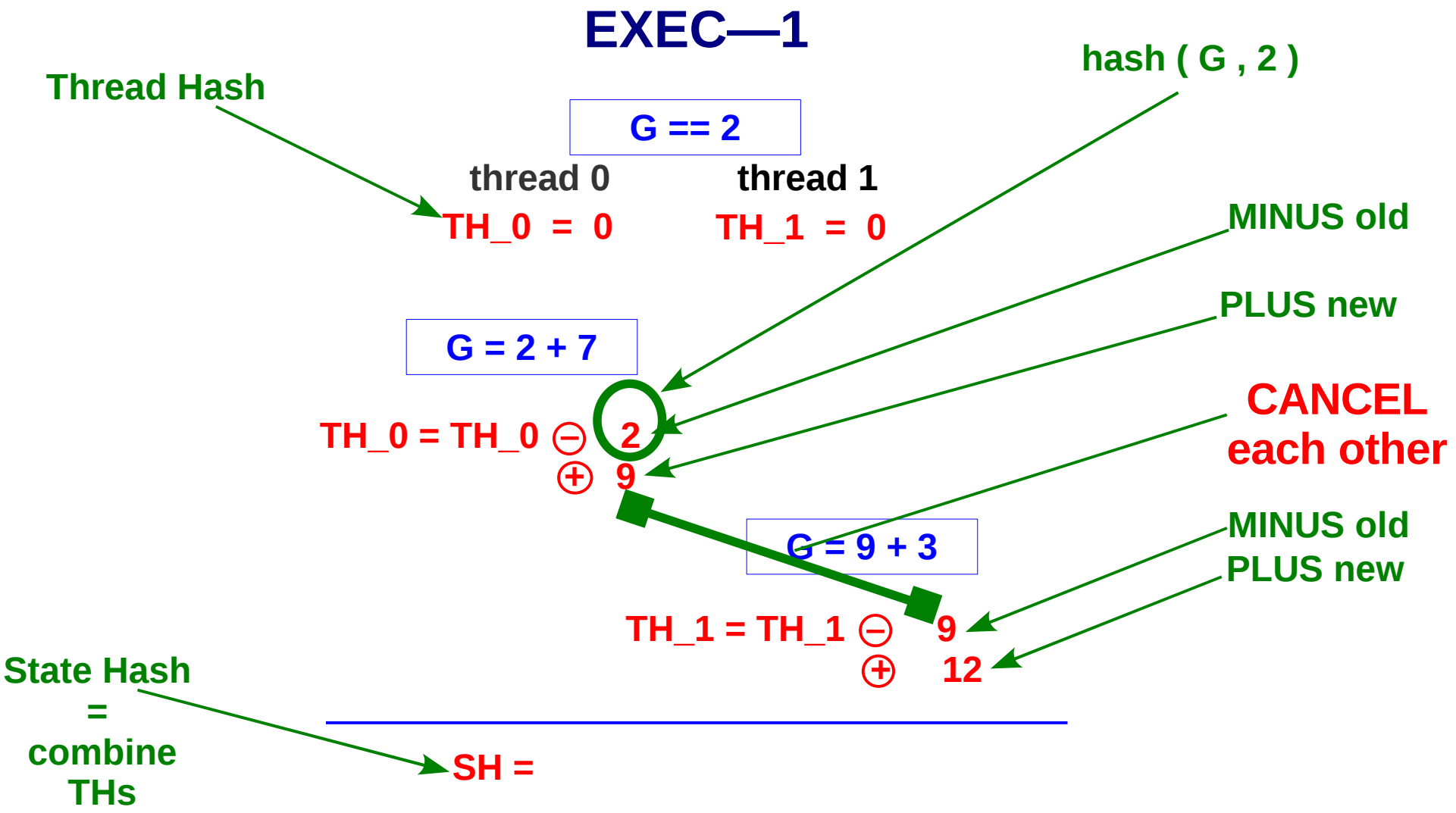
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12

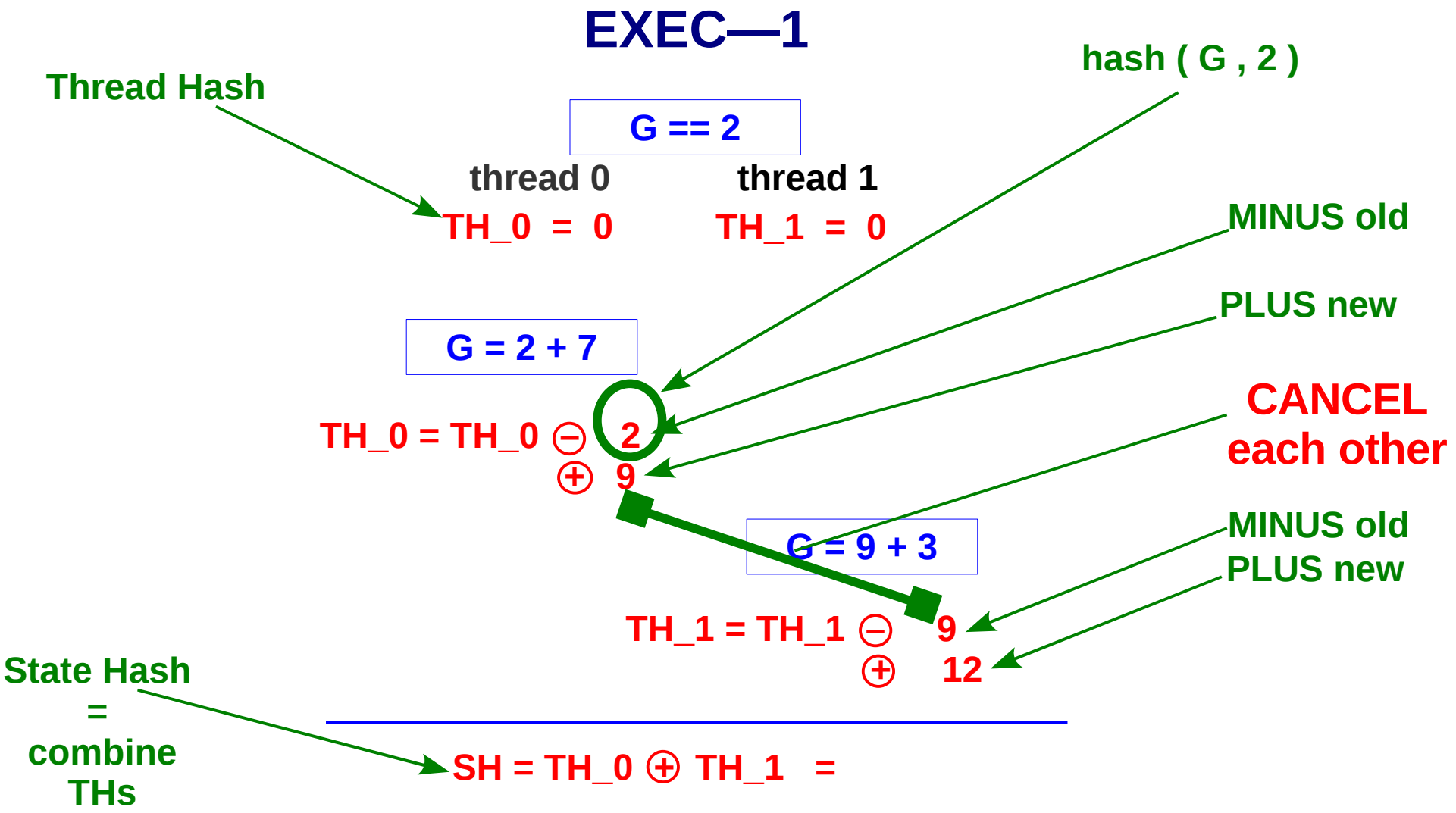
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12
State Hash
SH =

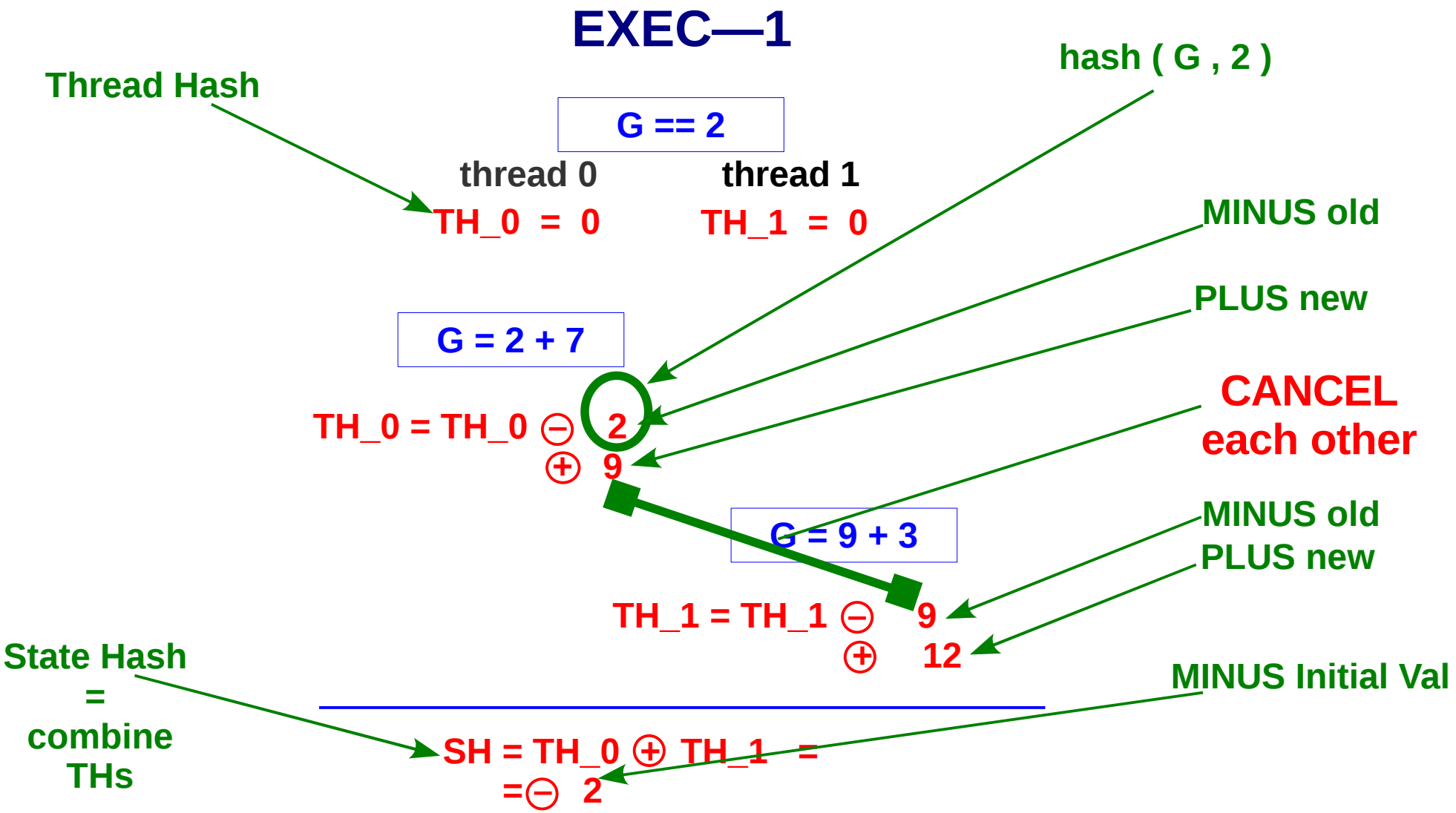
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12
State Hash
=
combine
THs
SH =

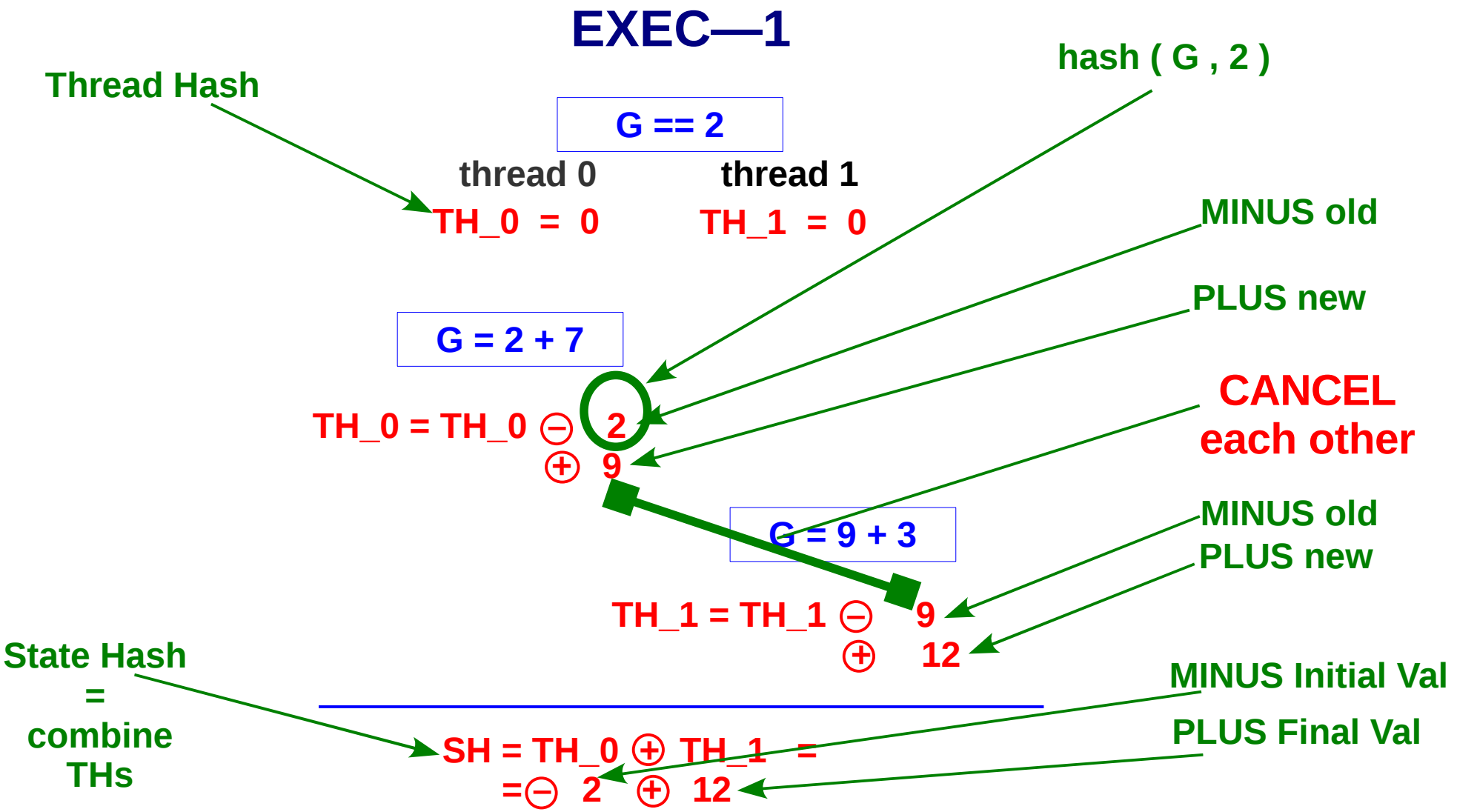
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12
State Hash
=
combine
THs
SH = TH_0 ⊕ TH_1 =

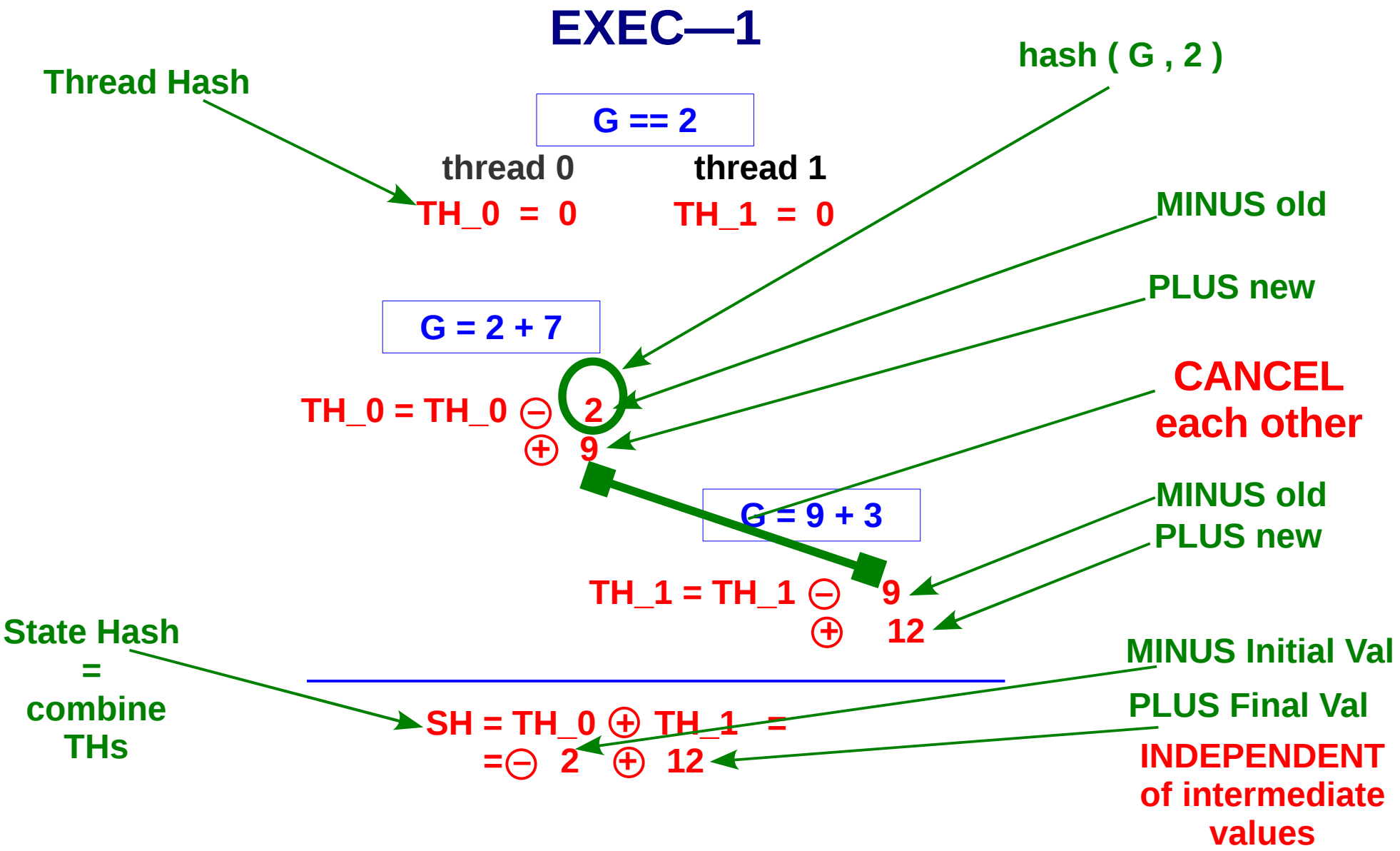
Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12
State Hash
=
combine
THs
MINUS Initial Val
SH = TH_0 ⊕ TH_1 =
=⊖ 2

Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
CANCEL
each other
TH_0 = TH_0 ⊖ 2
⊕ 9
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12
State Hash
=
combine
THs
MINUS Initial Val
PLUS Final Val
SH = TH_0 ⊕ TH_1 =
=⊖ 2 ⊕ 12

Example


EXEC—1
Thread Hash
hash (G , 2)
G == 2
thread 0
thread 1
TH_0 = 0
TH_1 = 0
MINUS old
PLUS new
G = 2 + 7
TH_0 = TH_0 ⊖ 2
⊕ 9
CANCEL
each other
G = 9 + 3
MINUS old
PLUS new
TH_1 = TH_1 ⊖ 9
⊕ 12
State Hash
=
combine
THs
MINUS Initial Val
PLUS Final Val
SH = TH_0 ⊕ TH_1 =
=⊖ 2 ⊕ 12
INDEPENDENT
of intermediate
values

Overview

Overview

- The computed hash is:

  MINUS (Initial__State) PLUS (Final__State)

Overview

- The computed hash is:

  MINUS (Initial__State) PLUS (Final__State)
- Independent of the intermediate steps

Overview

- The computed hash is:

  MINUS (Initial__State) PLUS (Final__State)
- Independent of the intermediate steps
- At each Write operation do:

  MINUS (Old) PLUS (New)

Overview

- The computed hash is:

  MINUS (Initial__State) PLUS (Final__State)
- Independent of the intermediate steps
- At each Write operation do:

  MINUS (Old) PLUS (New)
- Cancels the intermediate values

Advantages of Incremental Hashing

Advantages of Incremental Hashing

- Highly efficient hardware implementation

Advantages of Incremental Hashing

- Highly efficient hardware implementation
  - Thread Hash = locally computed = HW

Advantages of Incremental Hashing

- Highly efficient hardware implementation
  - Thread Hash = locally computed = HW
  - Global Hash = global operation = SW
    - Very rare ~ 10 – 10,000 times

Advantages of Incremental Hashing

- Highly efficient hardware implementation
  - Thread Hash = locally computed = HW
  - Global Hash = global operation = SW
    - Very rare ~ 10 – 10,000 times
- Associate & Commutative
  - The +/- operations can be: in Parallel & Out of Order
  - Flexible implementation choices for HW module

Advantages of Incremental Hashing

- Highly efficient hardware implementation
  - Thread Hash = locally computed = HW
  - Global Hash = global operation = SW
    - Very rare ~ 10 – 10,000 times
- Associate & Commutative
  - The +/- operations can be: in Parallel & Out of Order
  - Flexible implementation choices for HW module
- Can delete some variables:

$$\oplus \text{initial\_value} \ominus \text{final\_value}$$

External determinism

Capturing state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

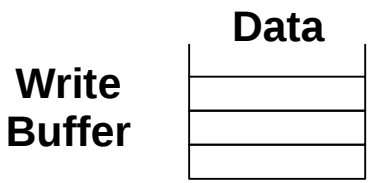
Conclusions

Basic design

Basic design

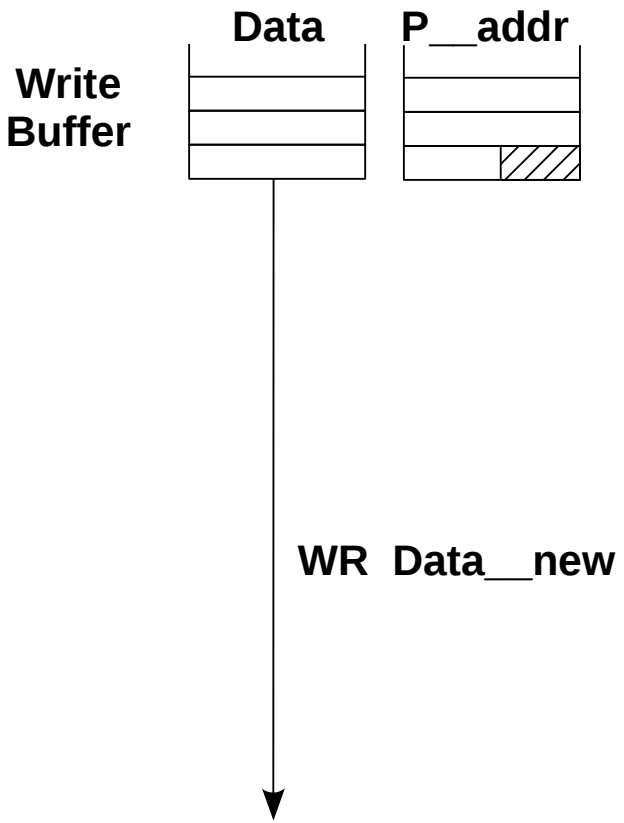
Write Buffer

Basic design

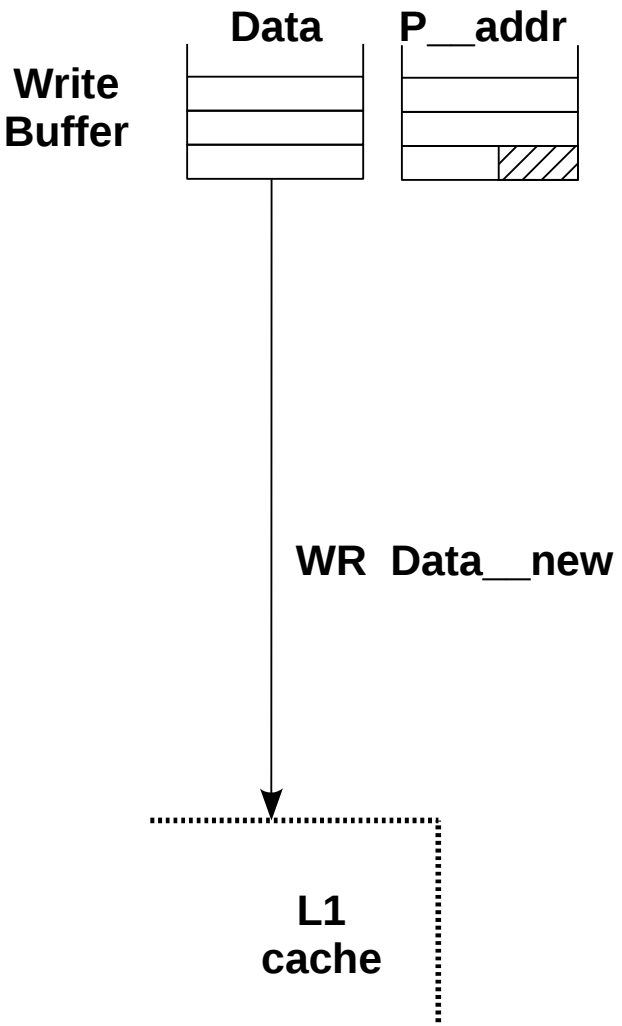


Basic design

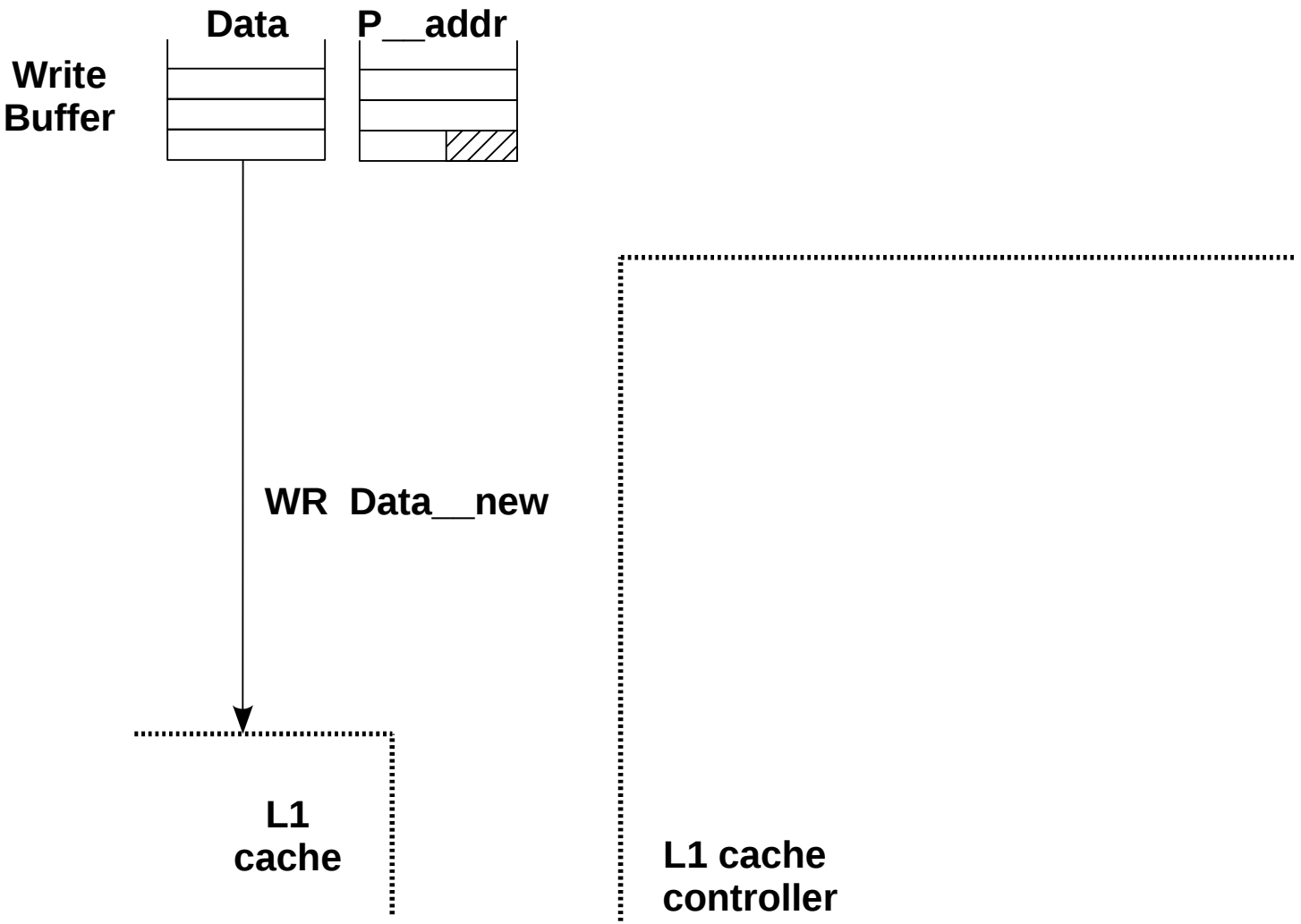
Basic design



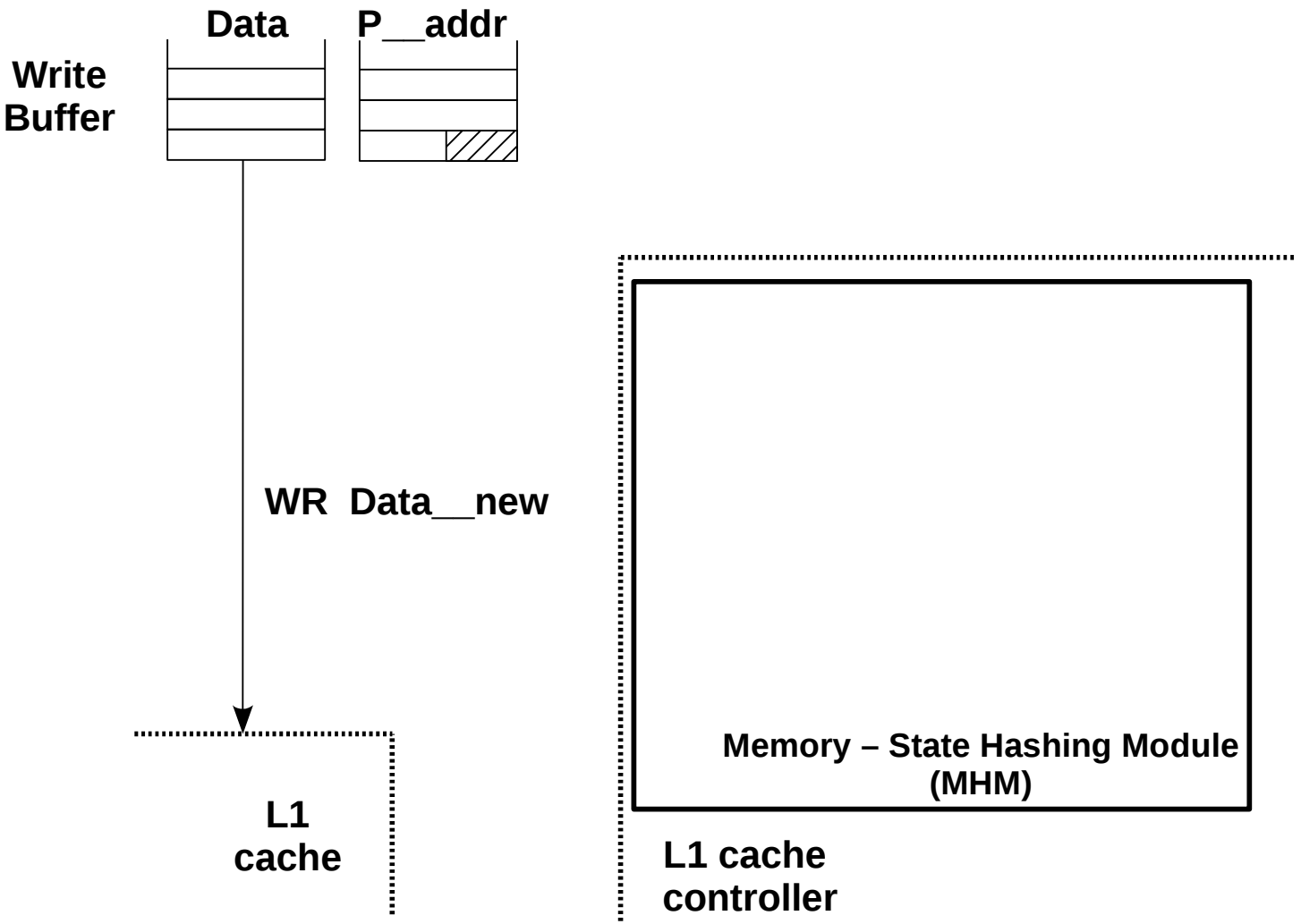
Basic design



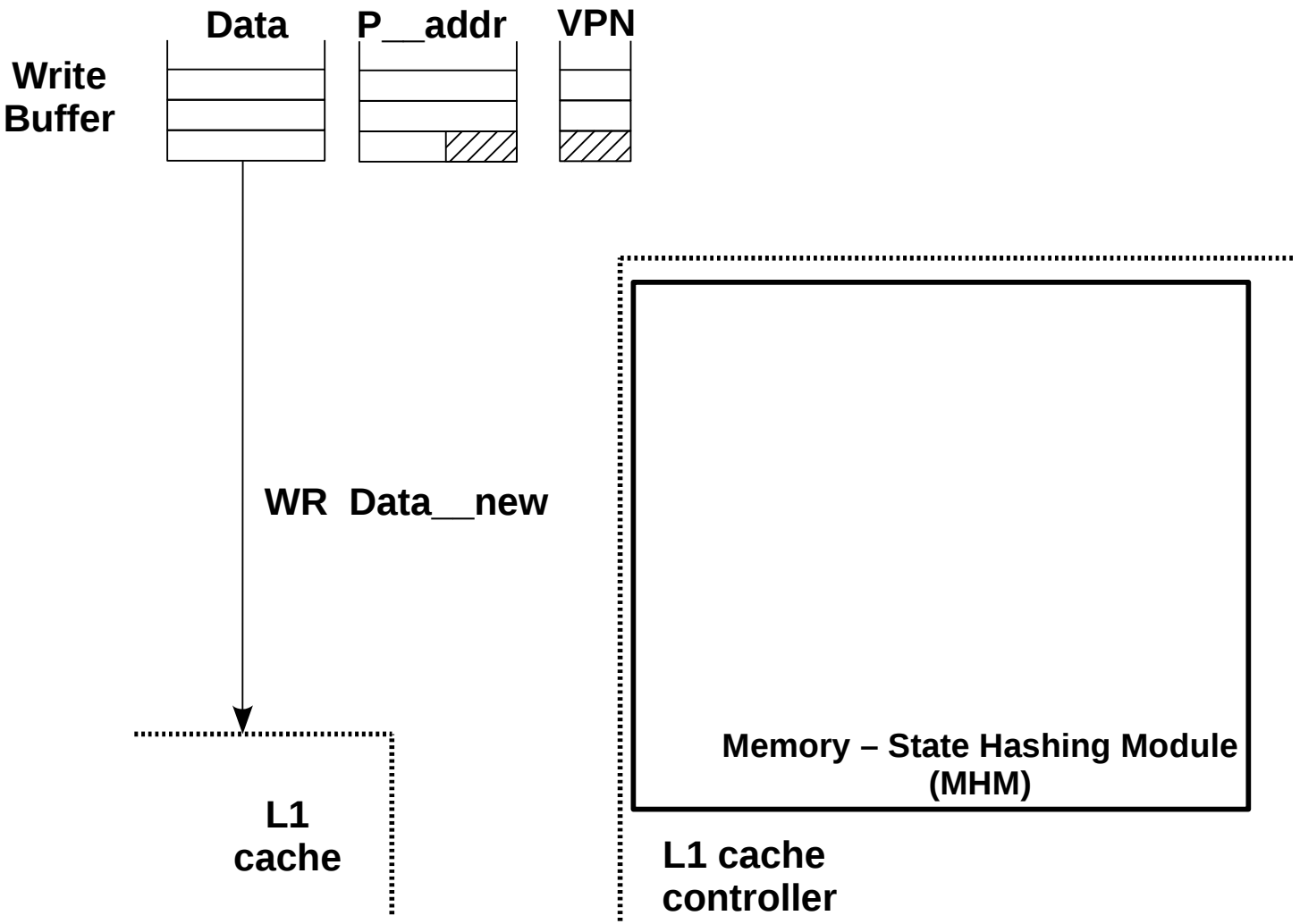
Basic design



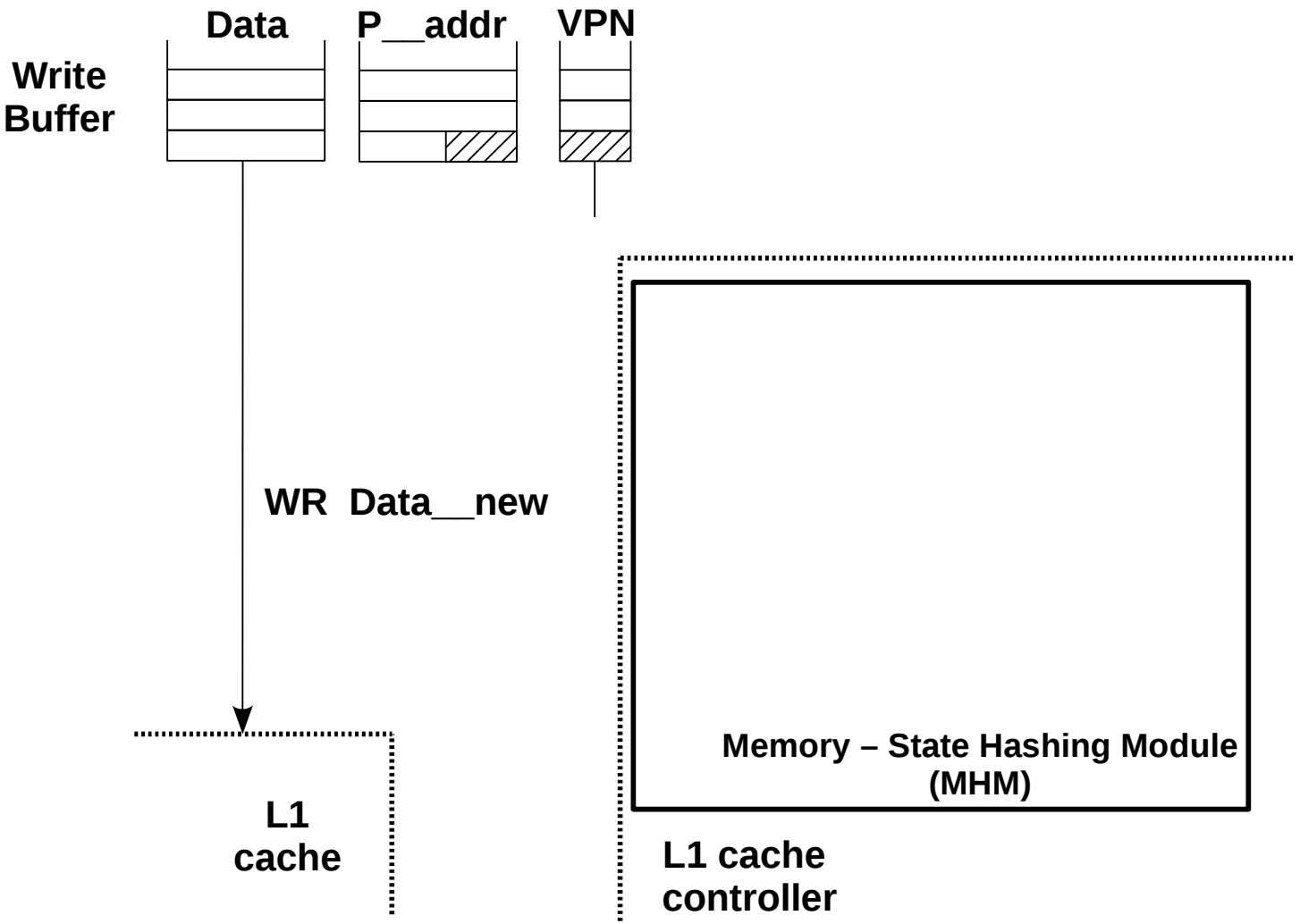
Basic design



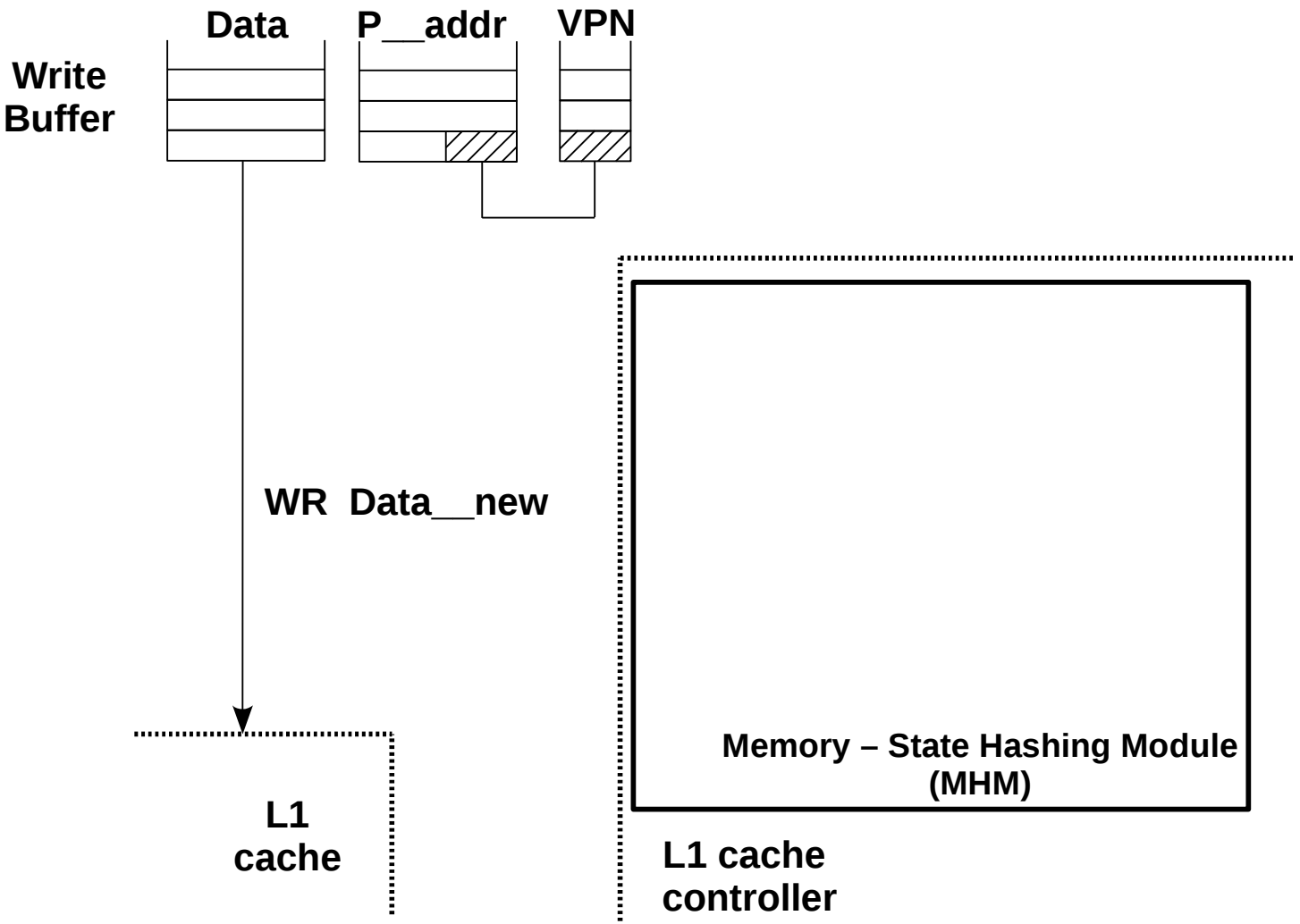
Basic design



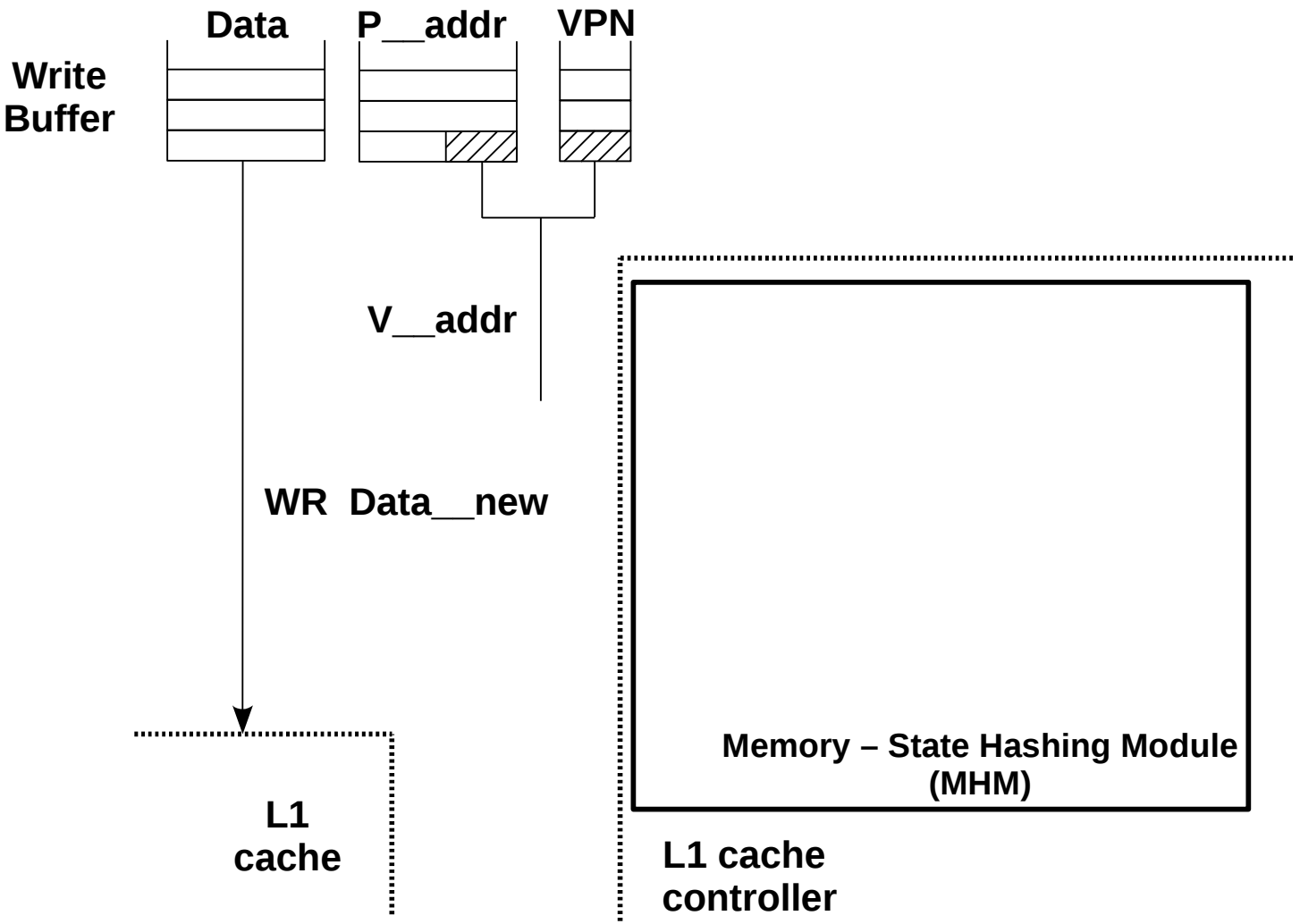
Basic design



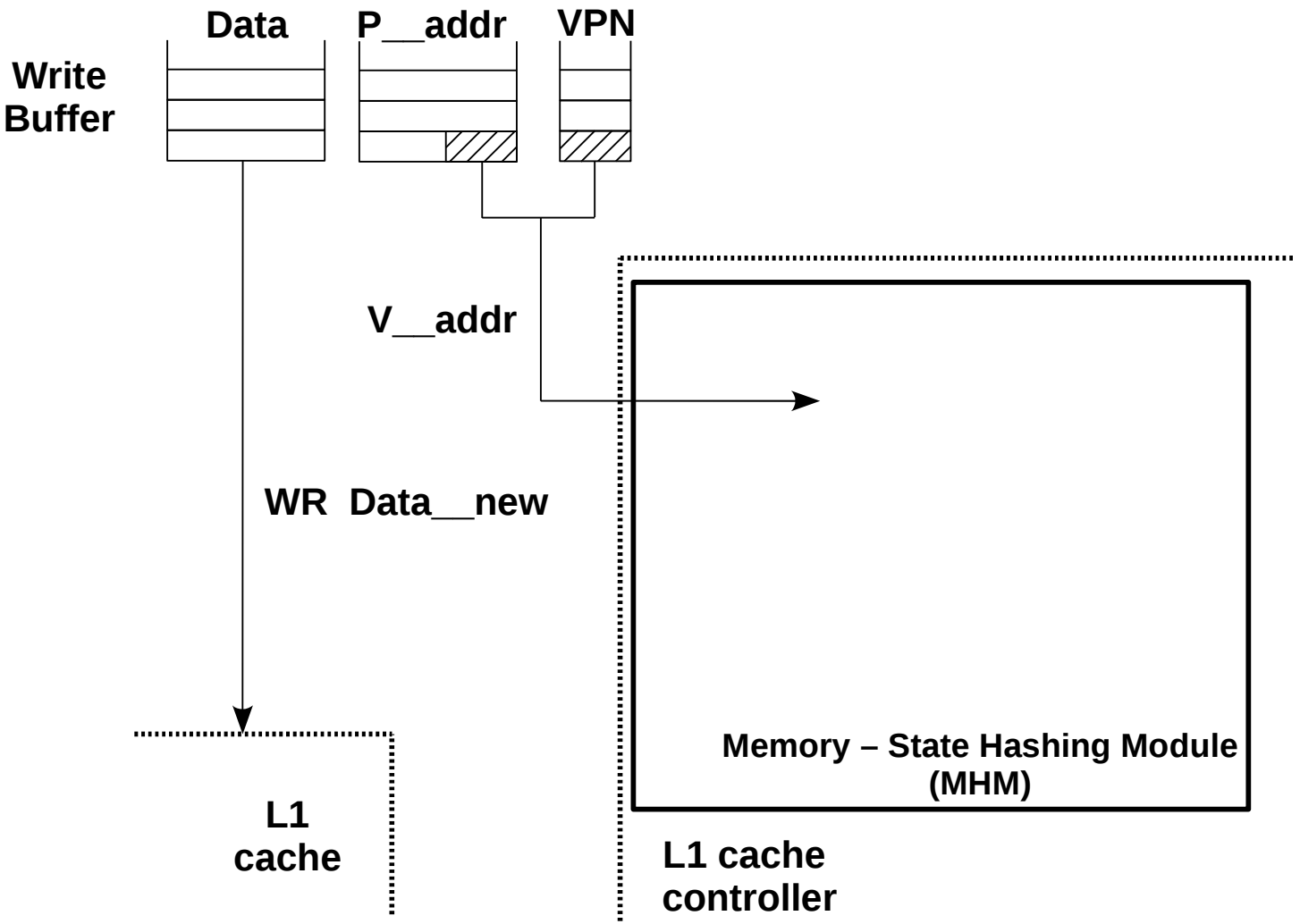
Basic design



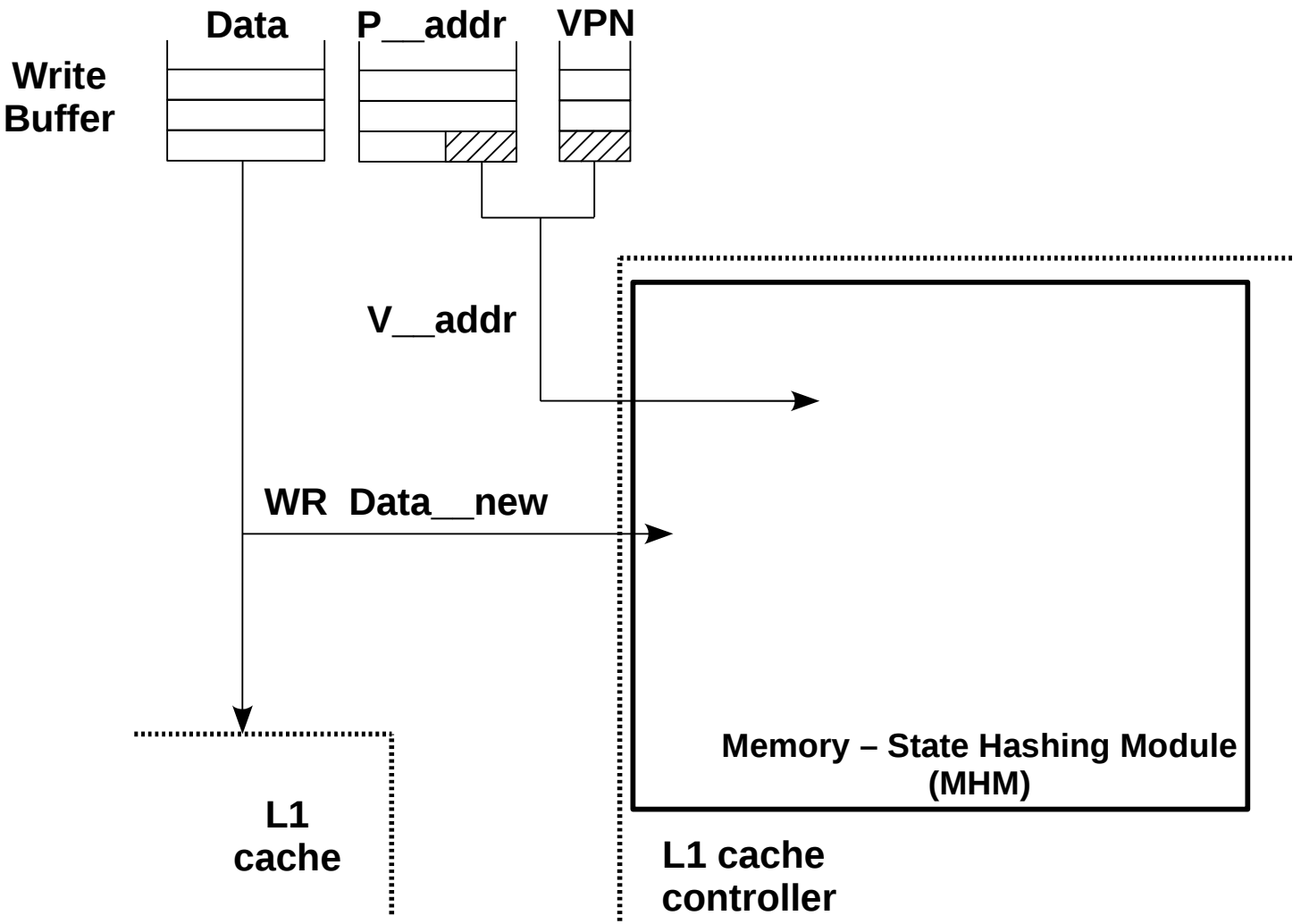
Basic design



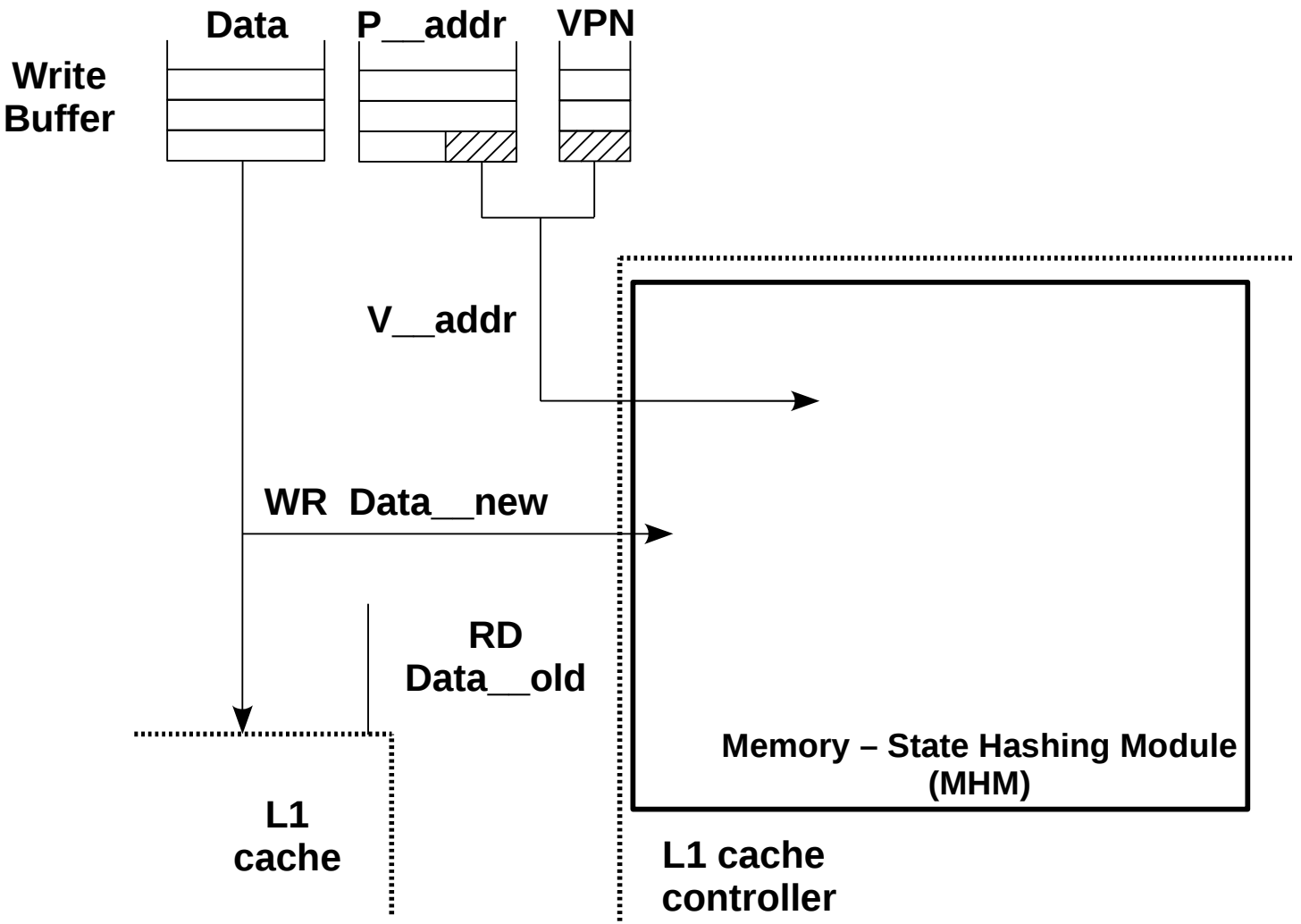
Basic design



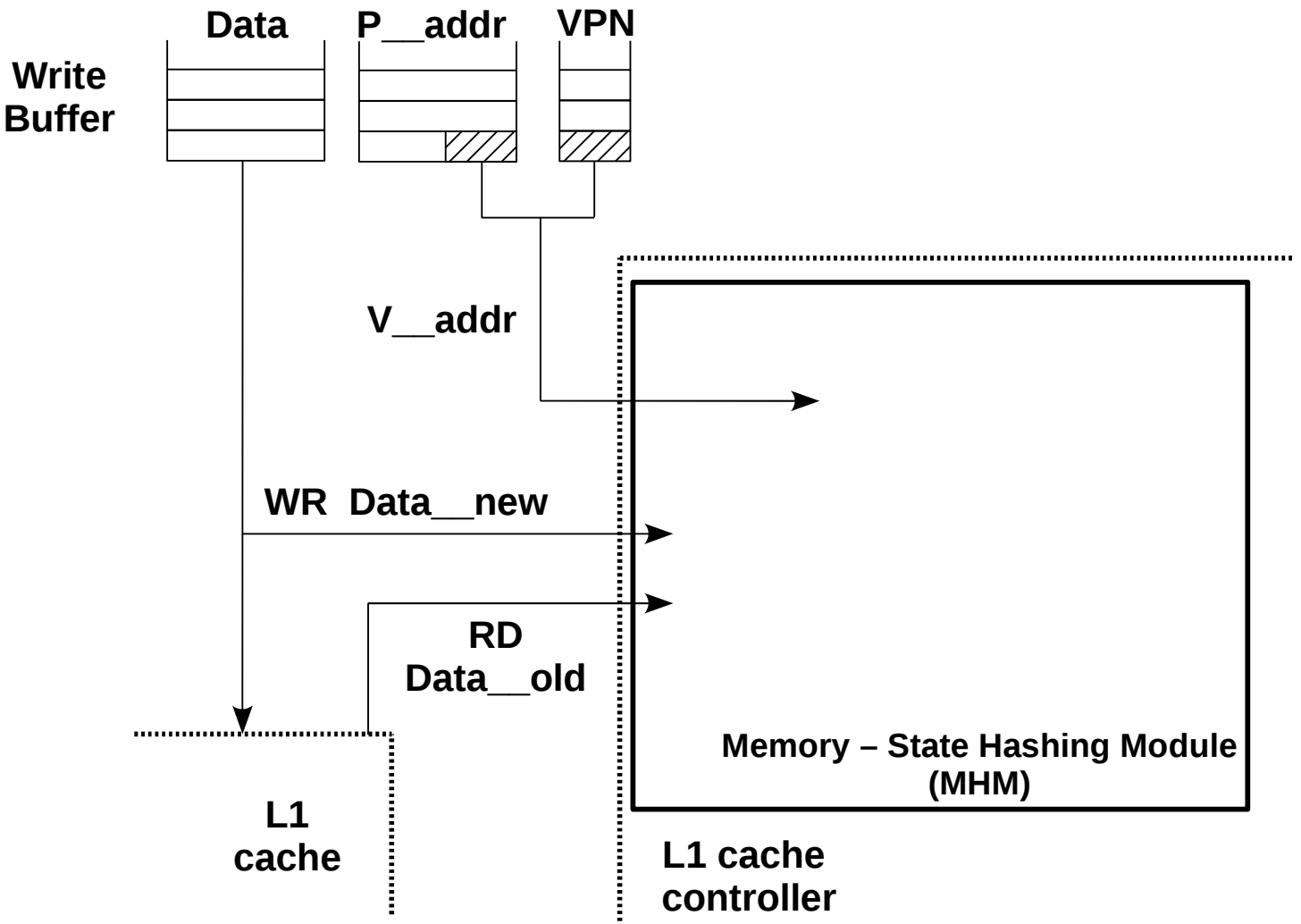
Basic design



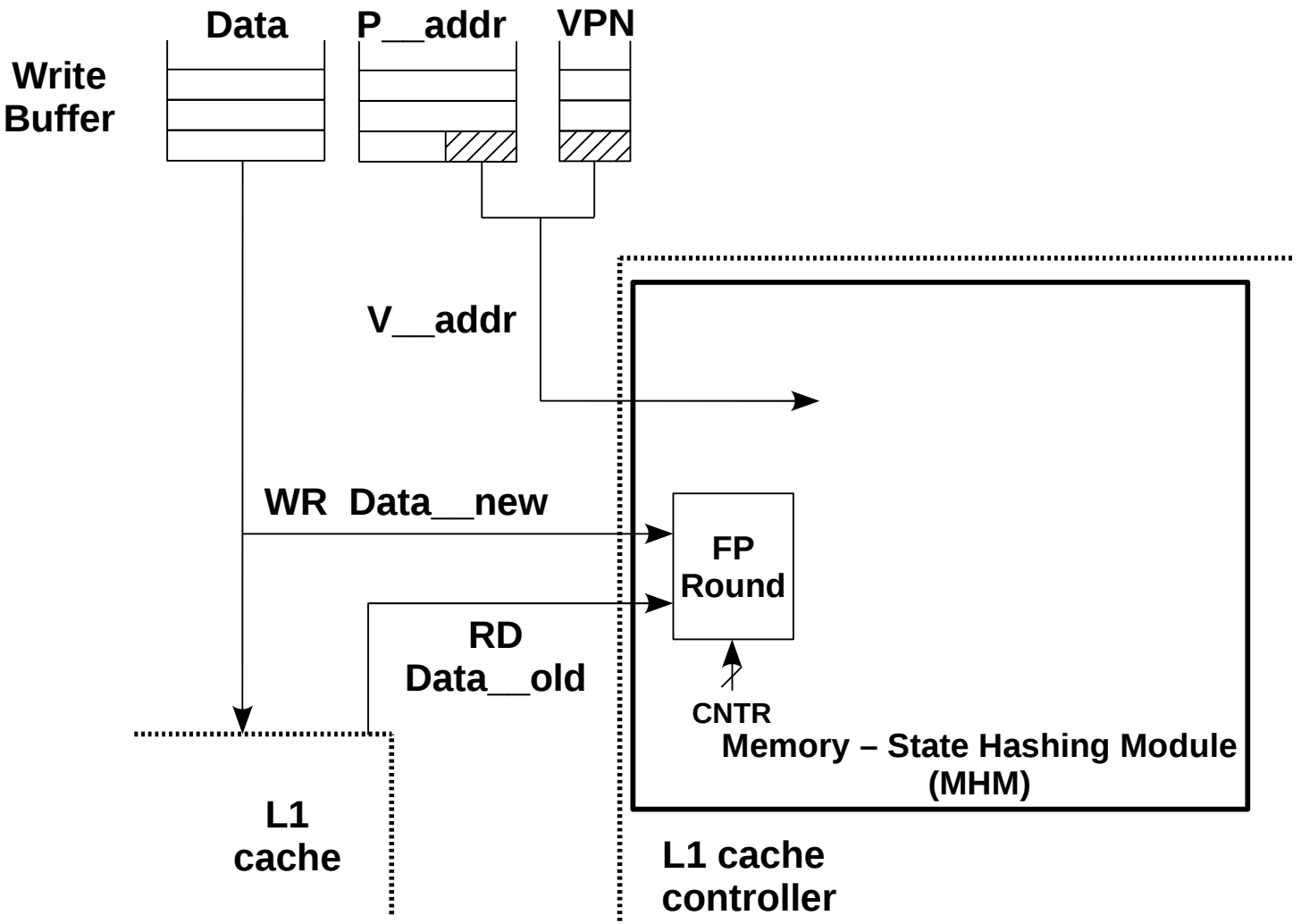
Basic design



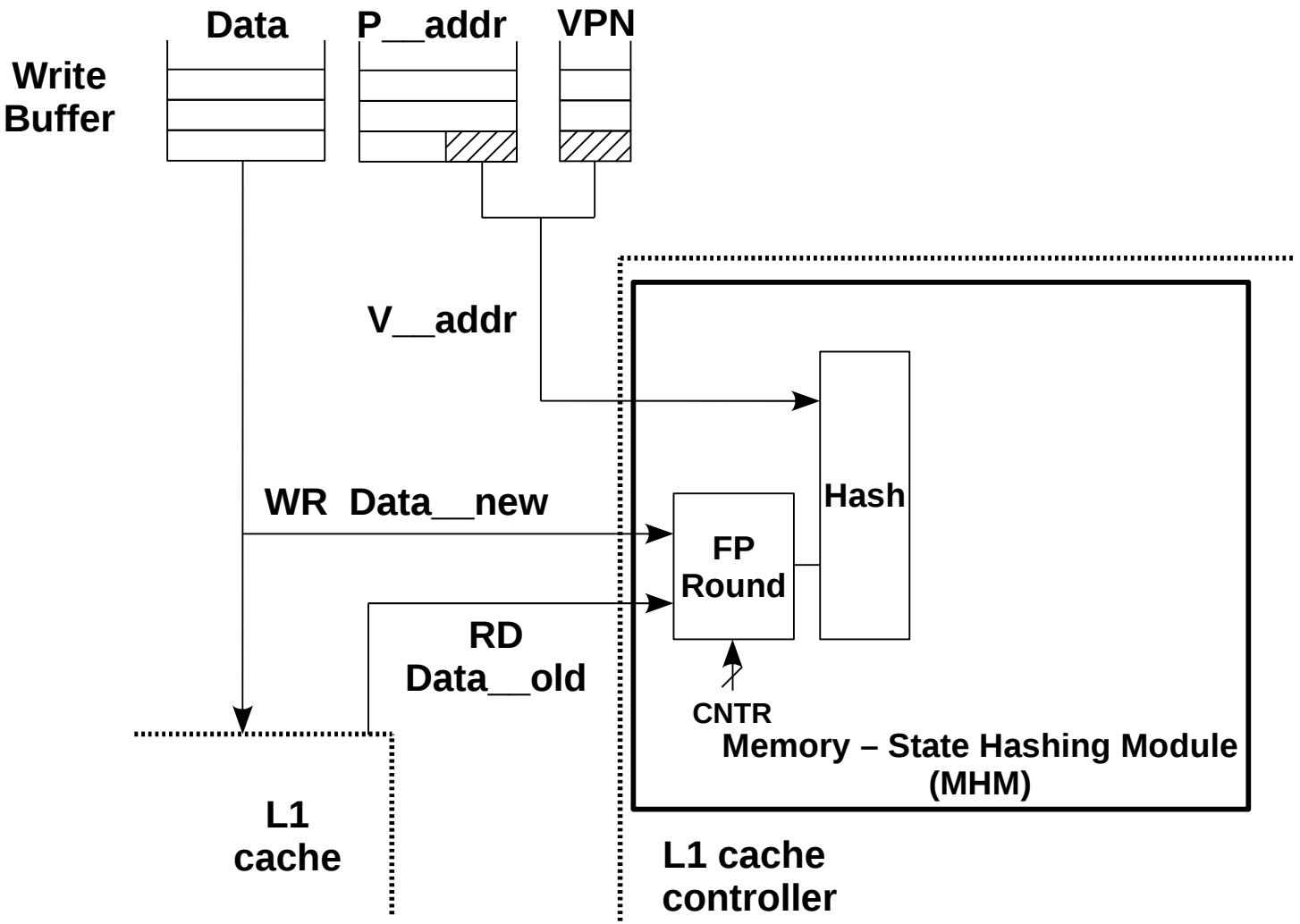
Basic design



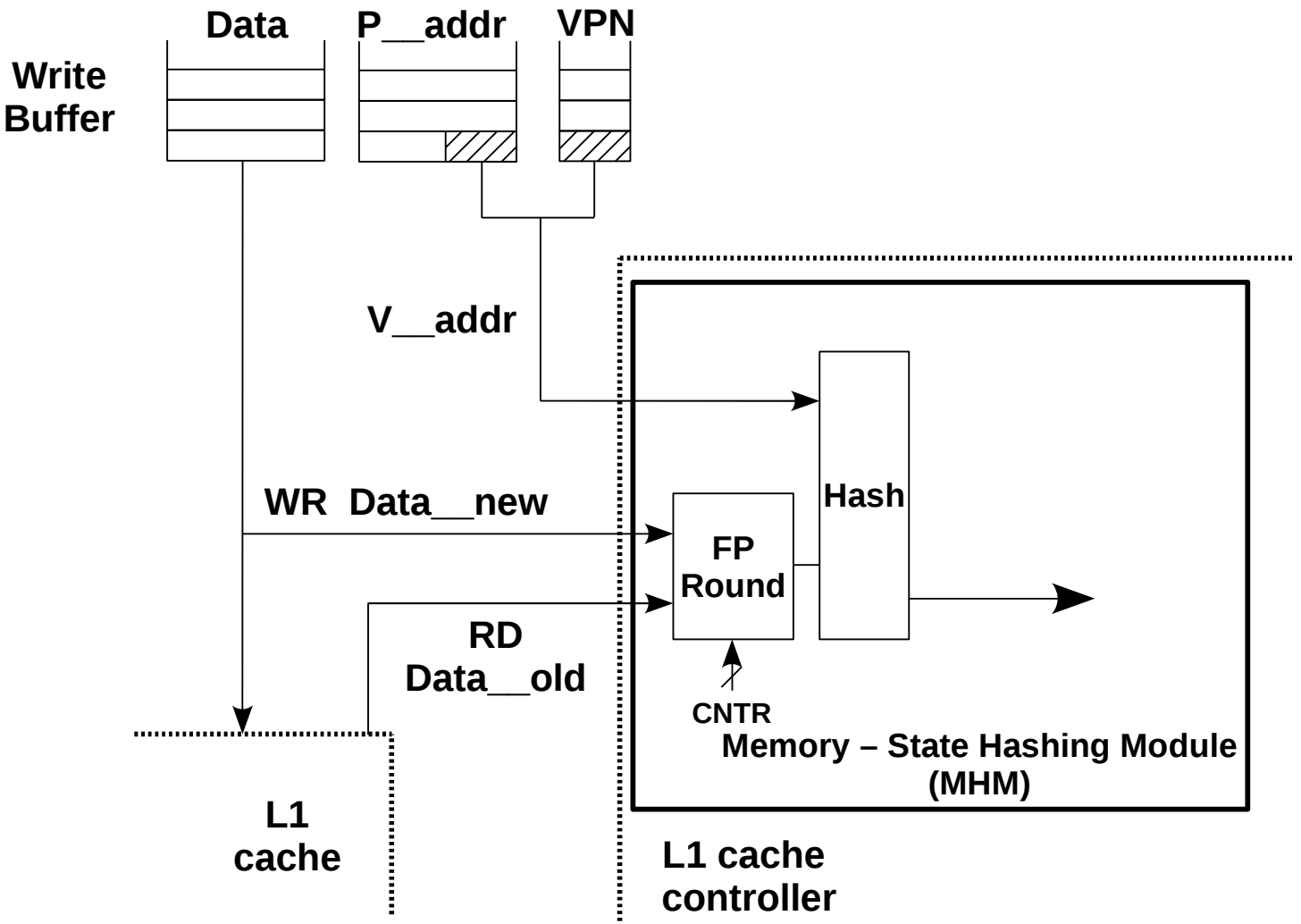
Basic design



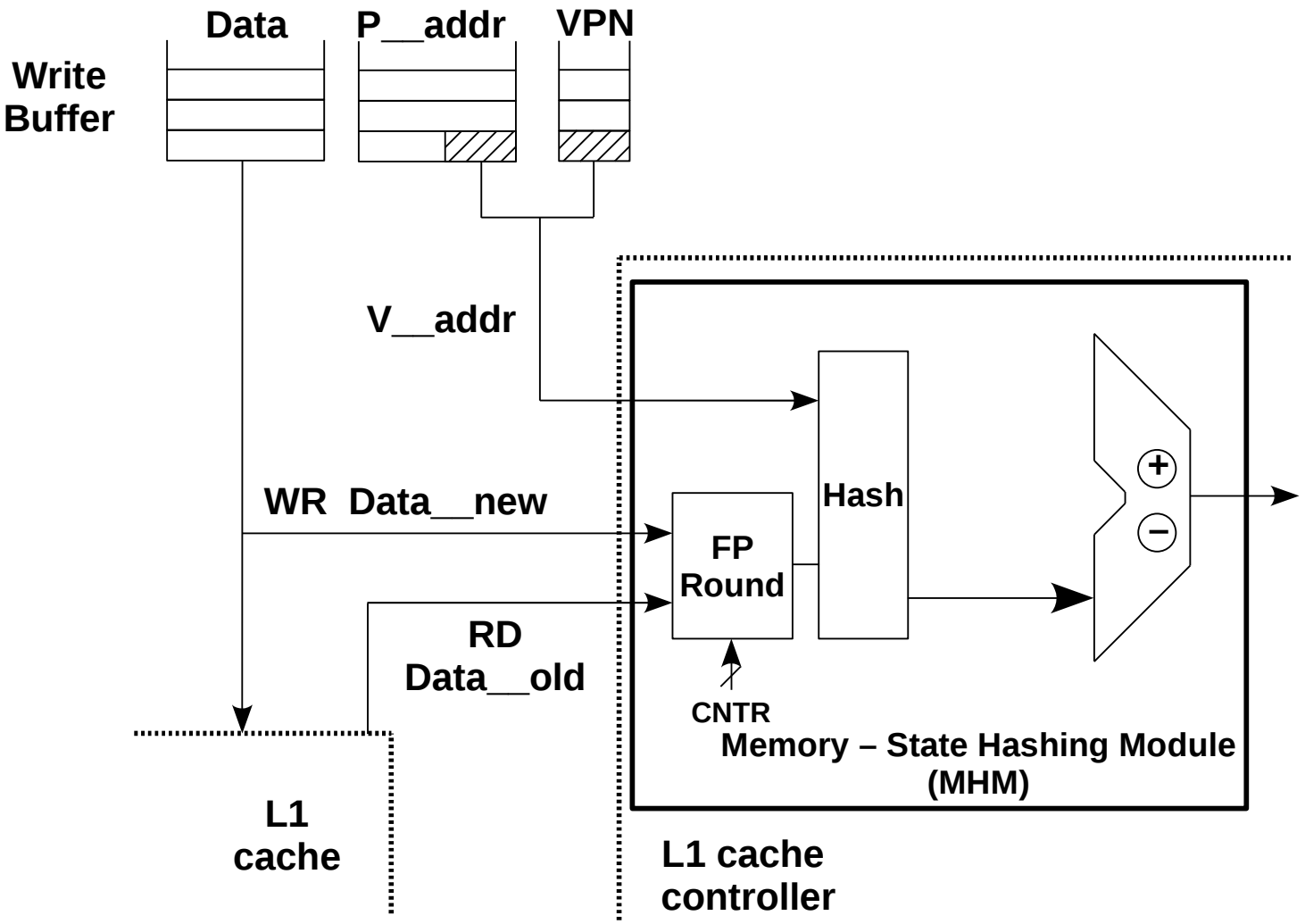
Basic design



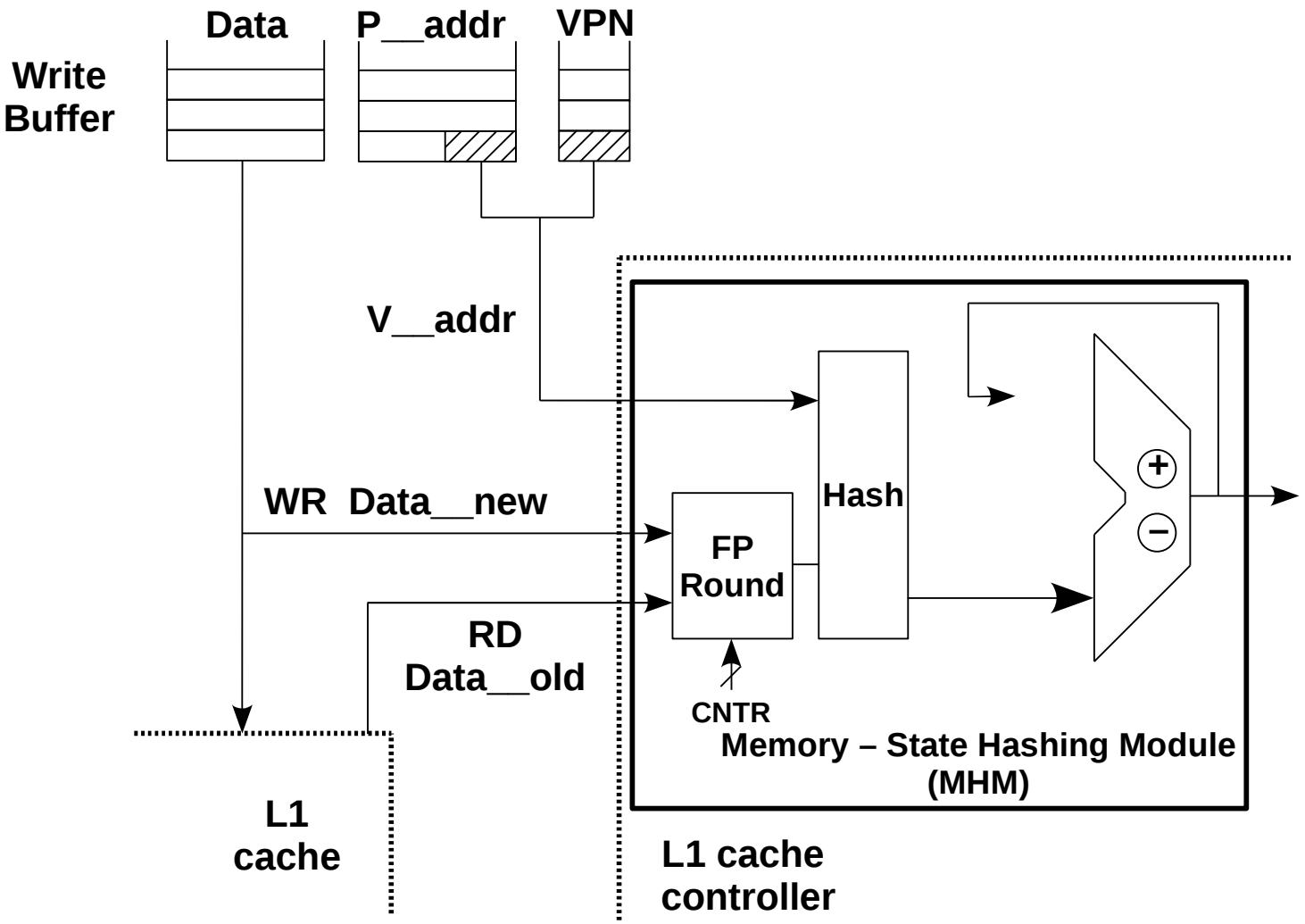
Basic design



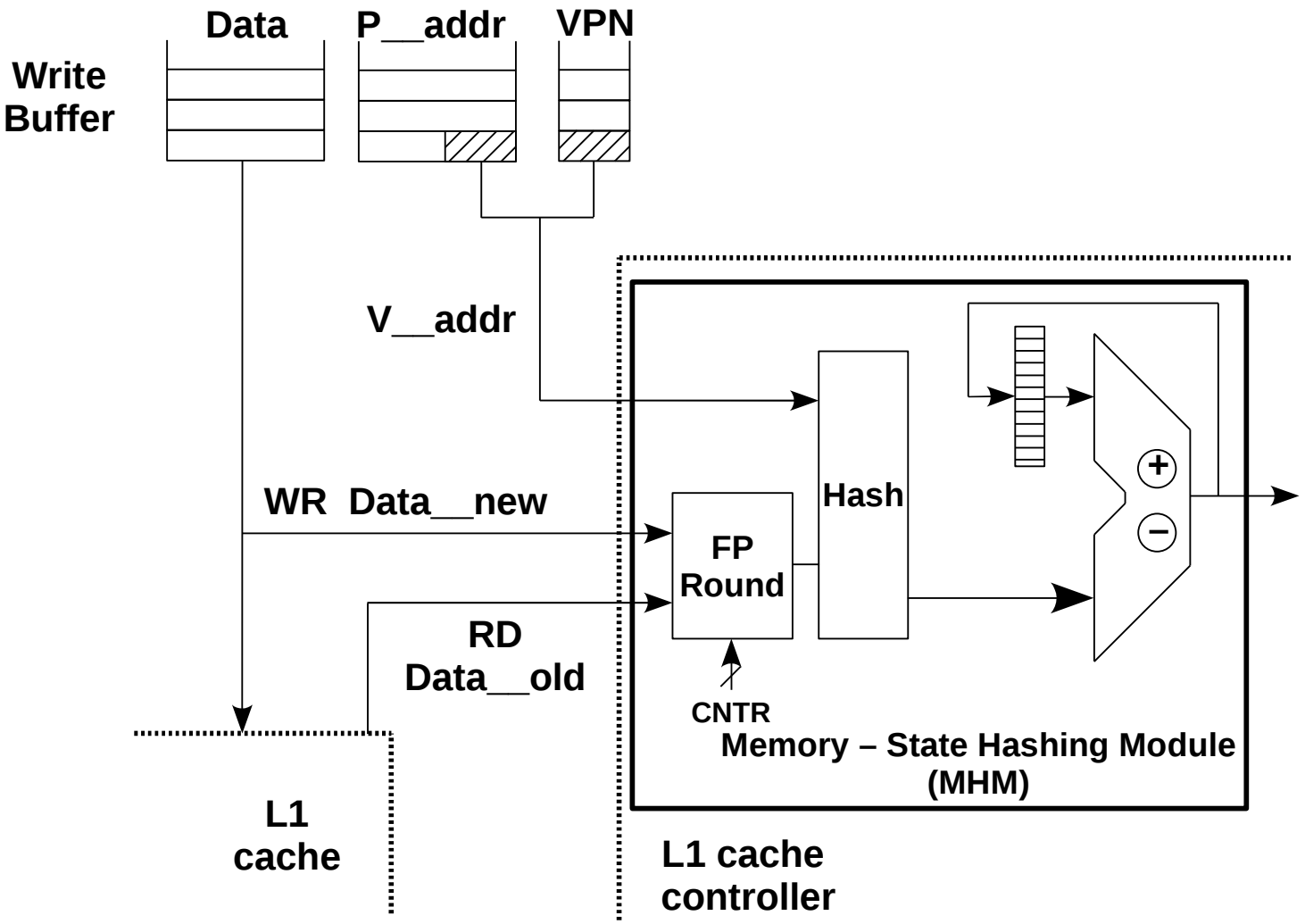
Basic design



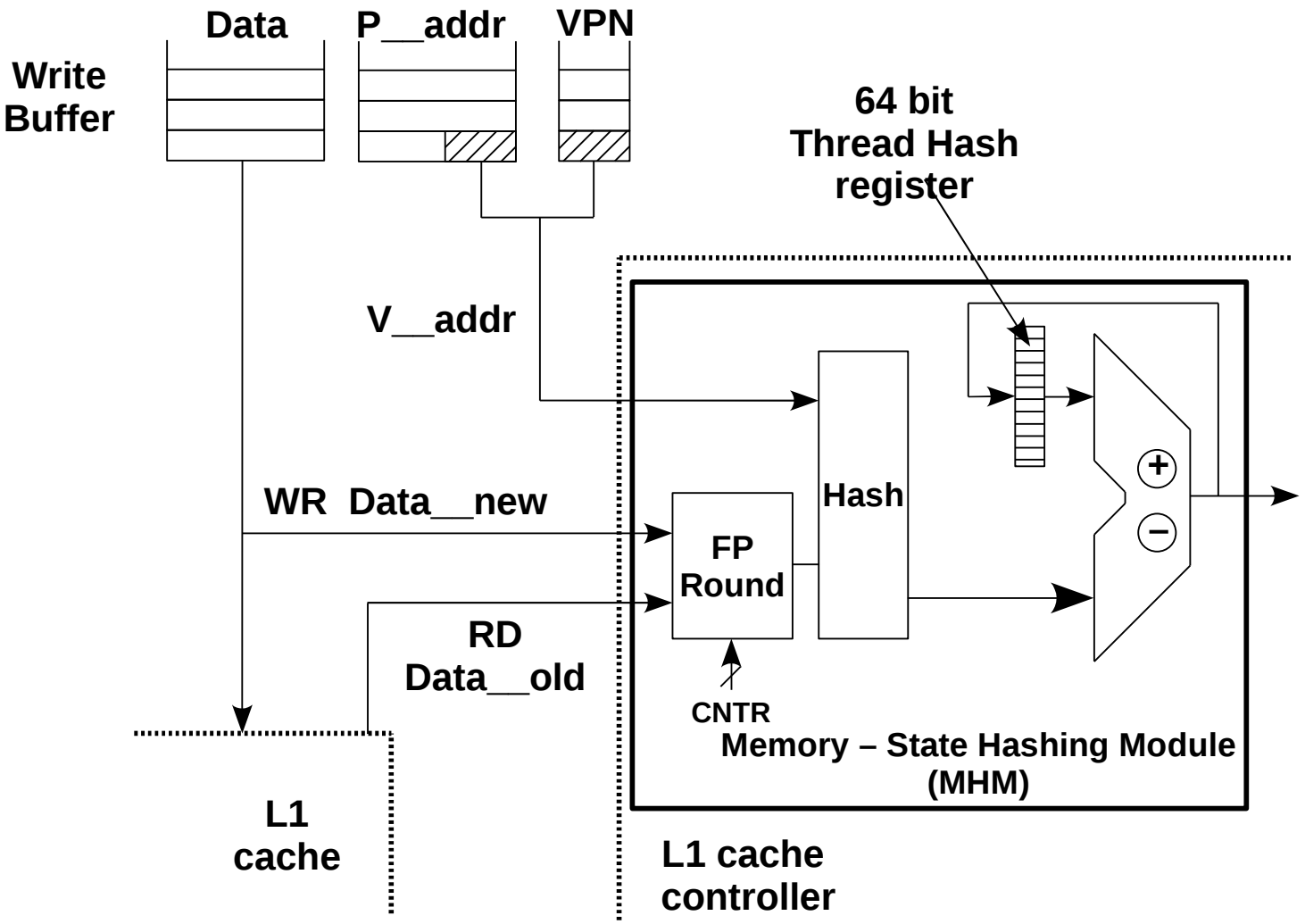
Basic design



Basic design



Basic design



Flexible Implementation Choices

Flexible Implementation Choices

Commutative
Associative

Flexible Implementation Choices

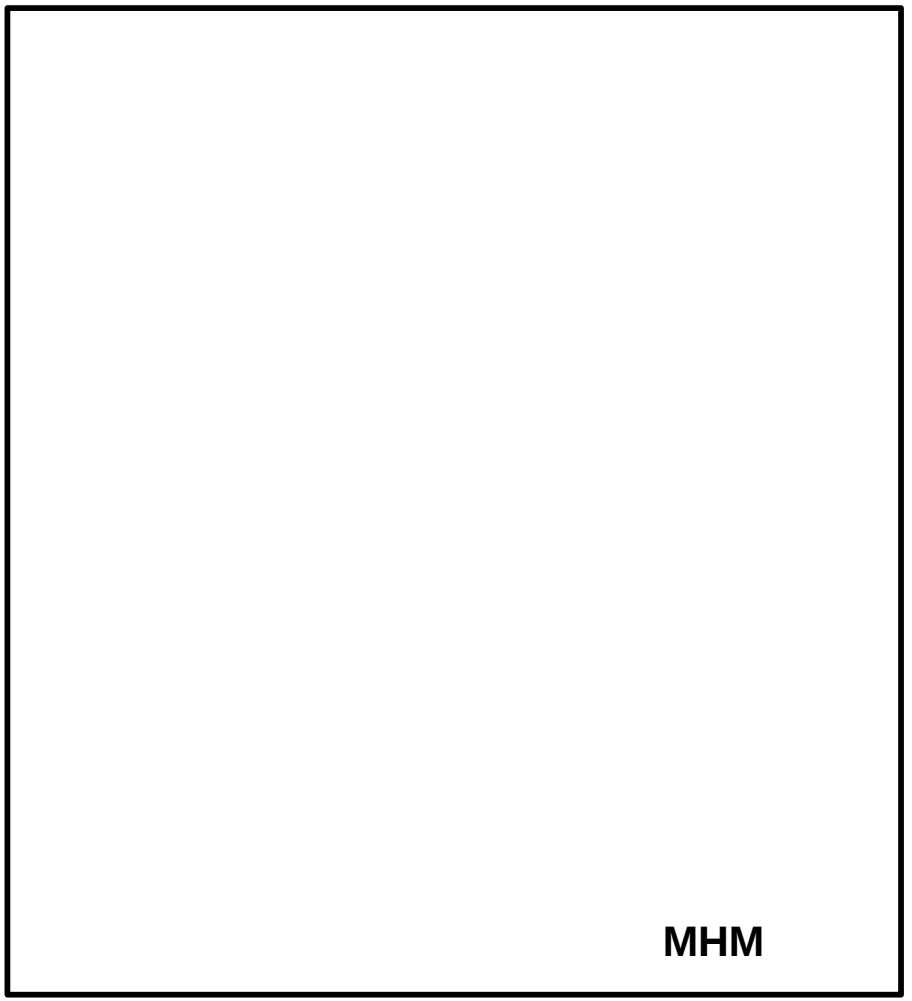
Commutative
Associative

Flexible Implementation Choices

Commutative
Associative

Out of Order
In parallel

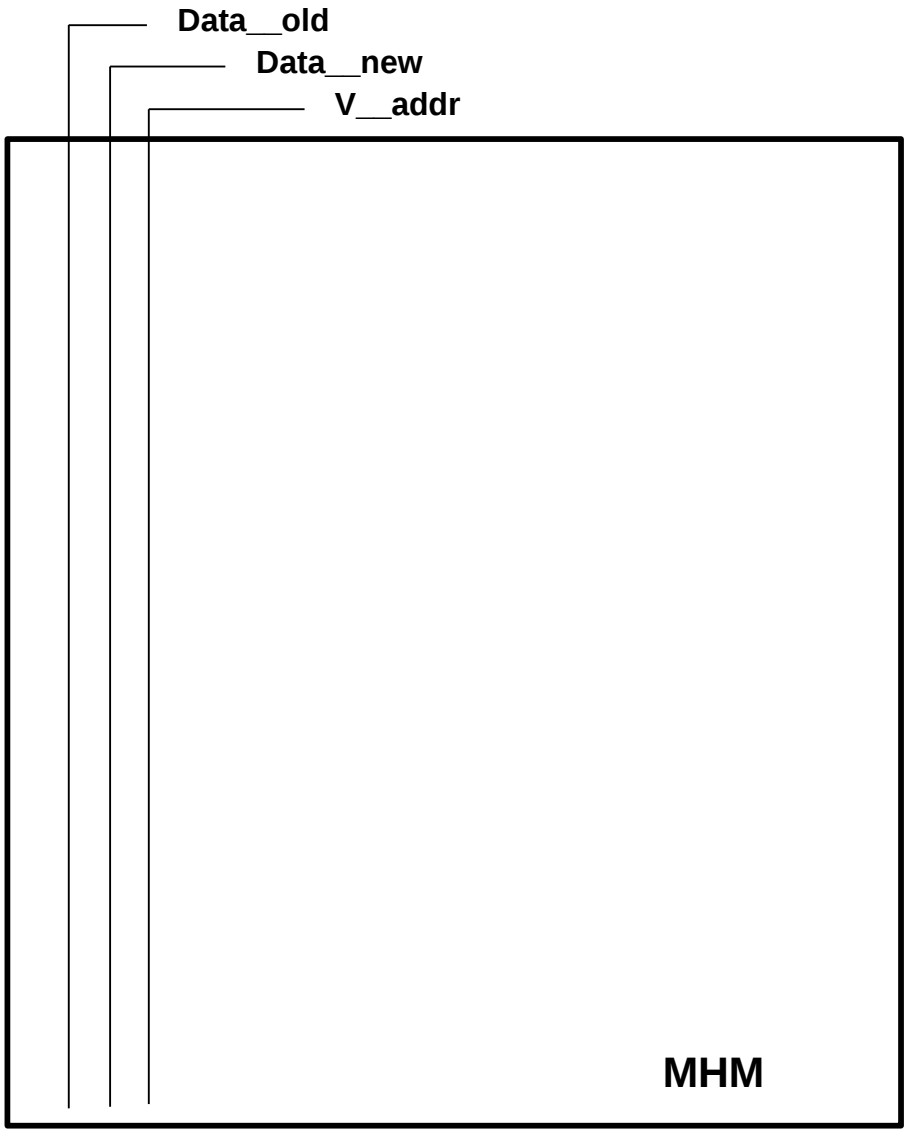
Flexible Implementation Choices



Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices



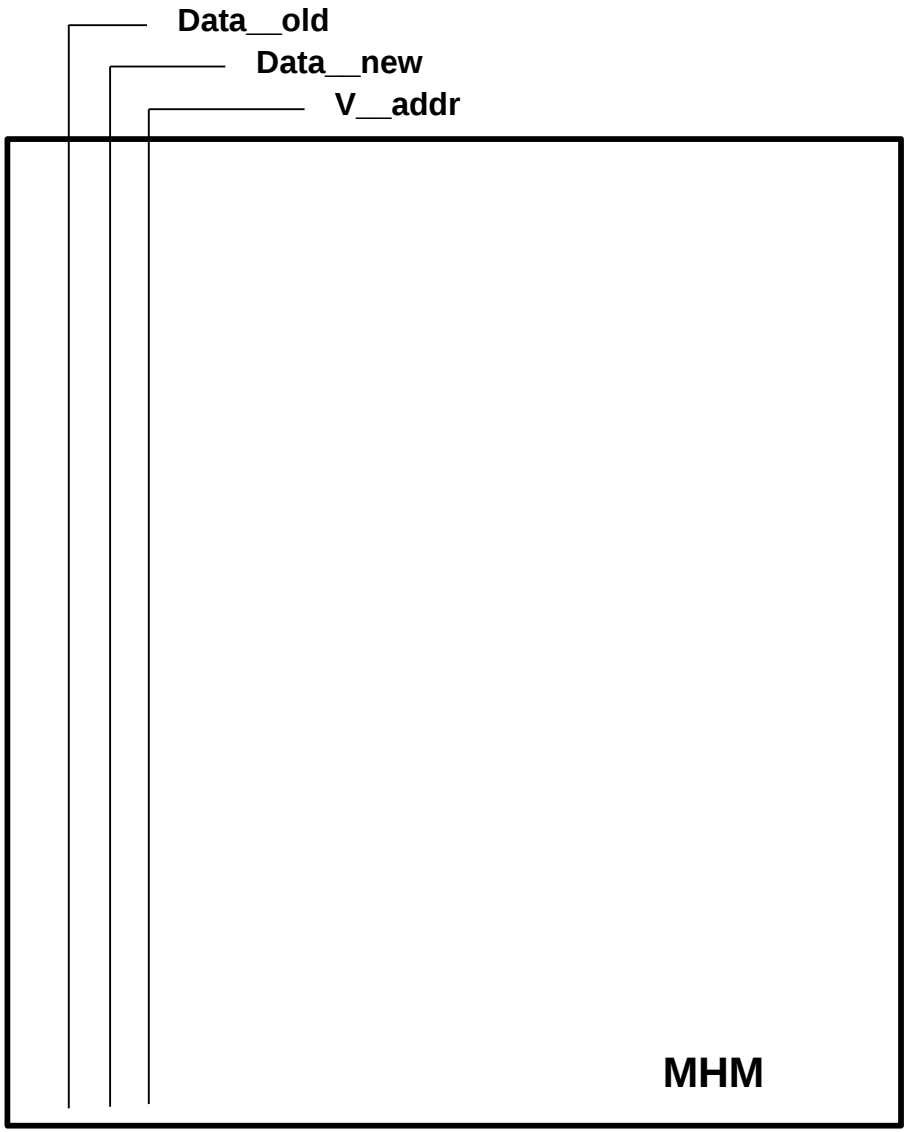
Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices

CLUSTERS

Compute Intermediate Results



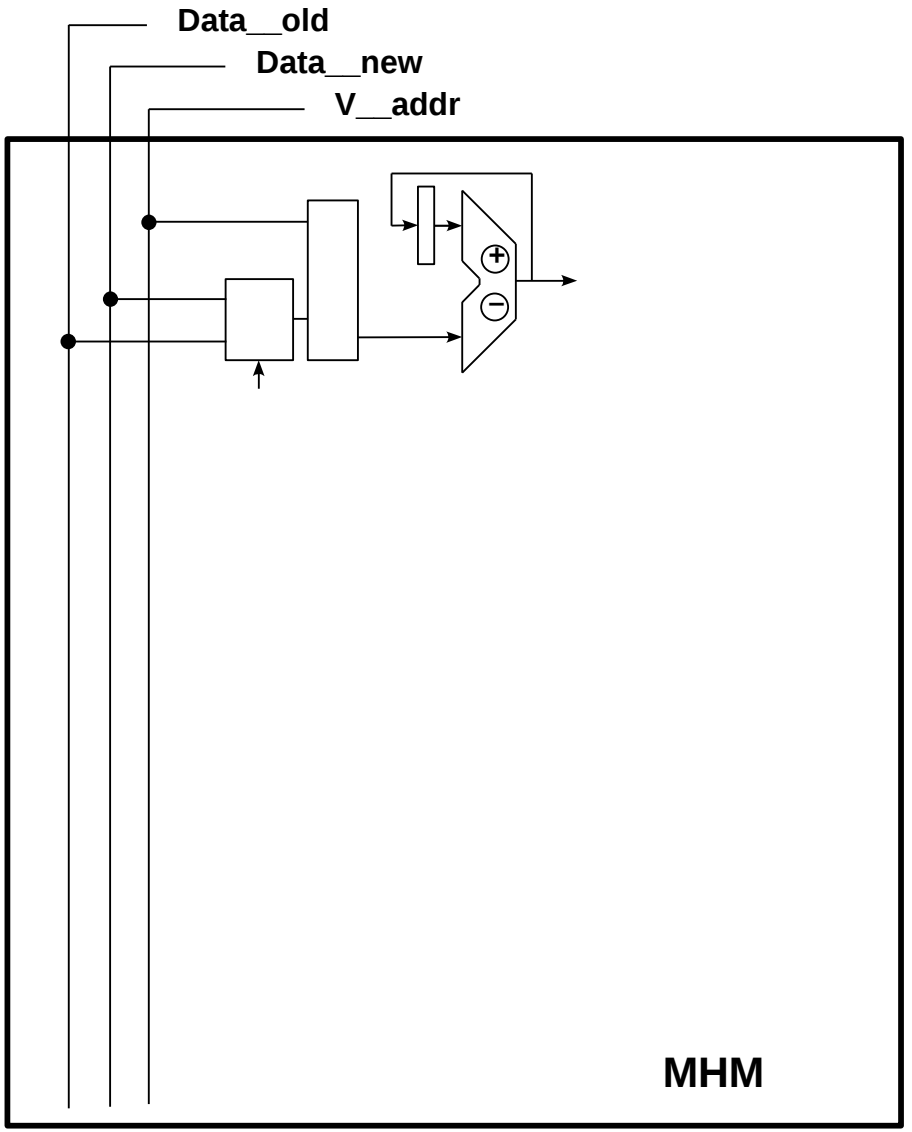
Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices

CLUSTERS

Compute Intermediate Results



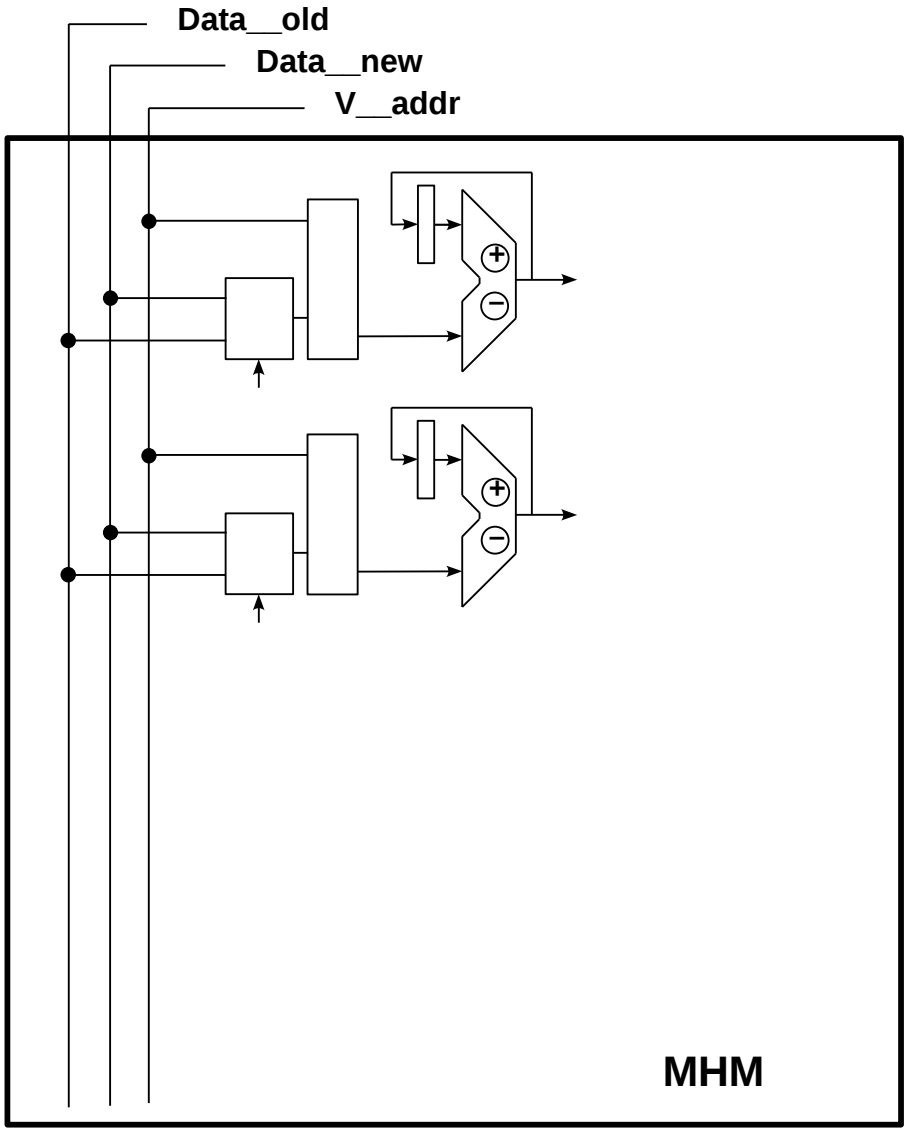
Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices

CLUSTERS

Compute Intermediate Results



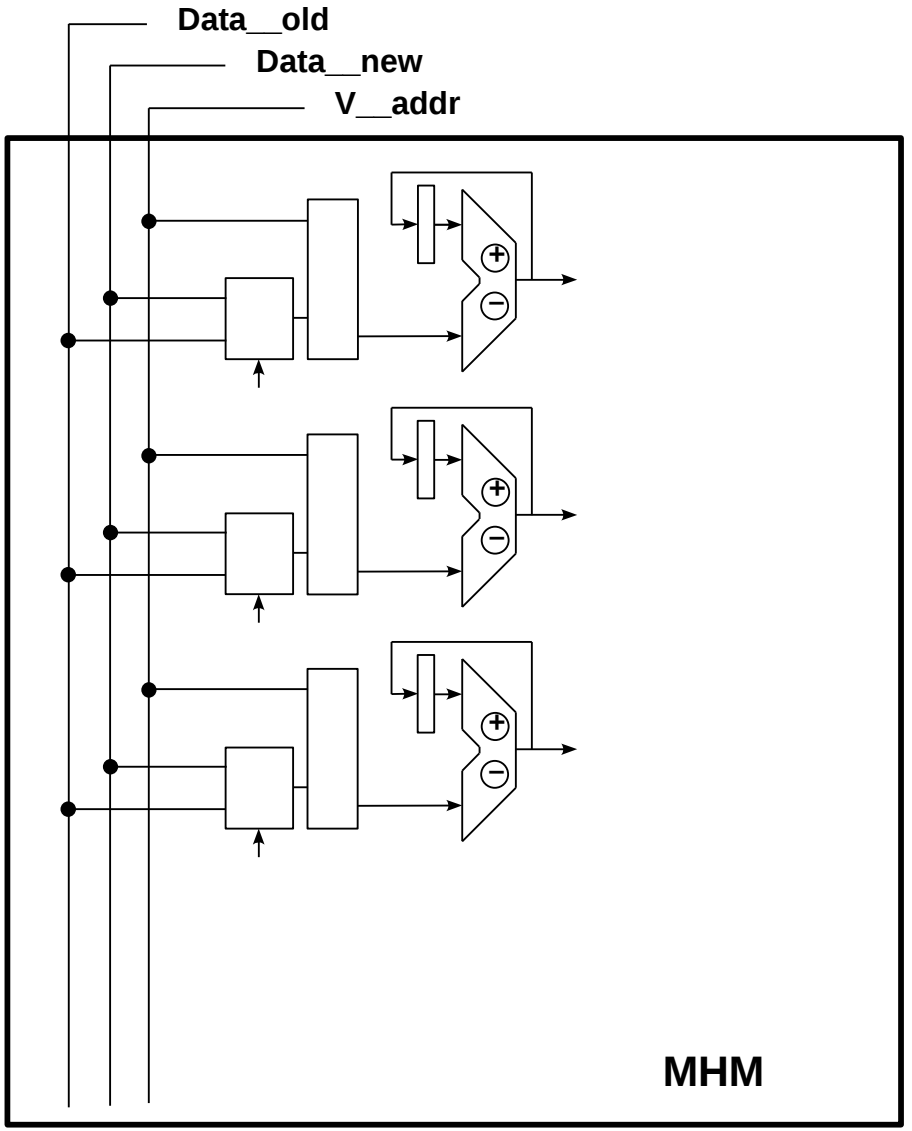
Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices

CLUSTERS

Compute Intermediate Results

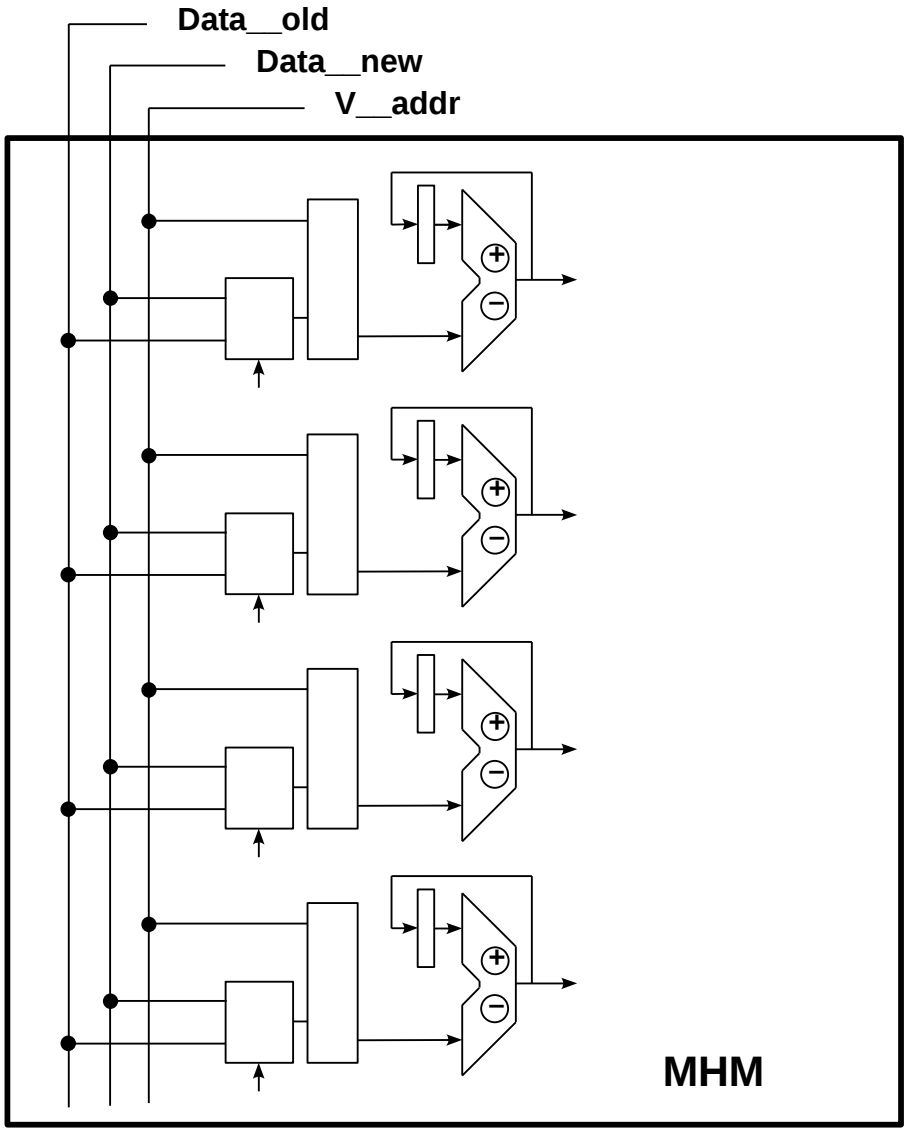


Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices

CLUSTERS
Compute Intermediate Results



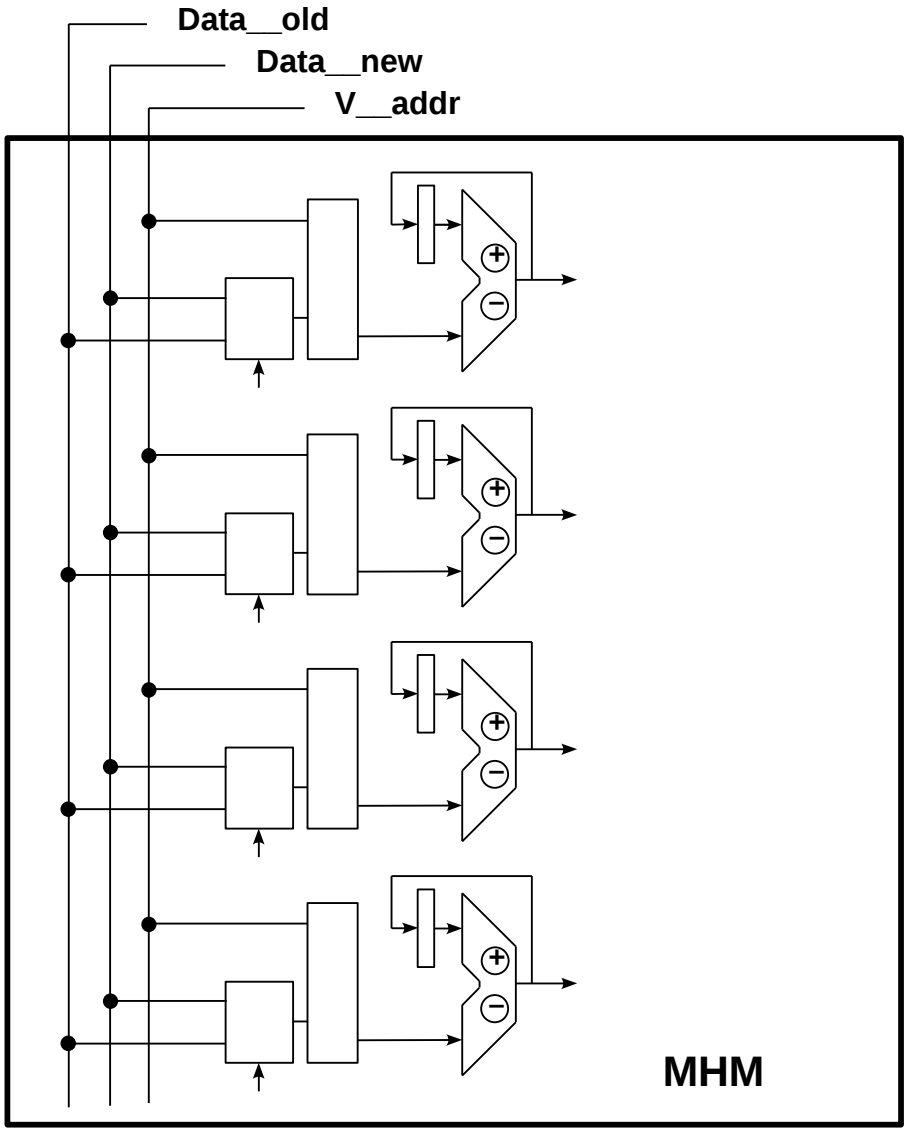
Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices

CLUSTERS
Compute Intermediate Results

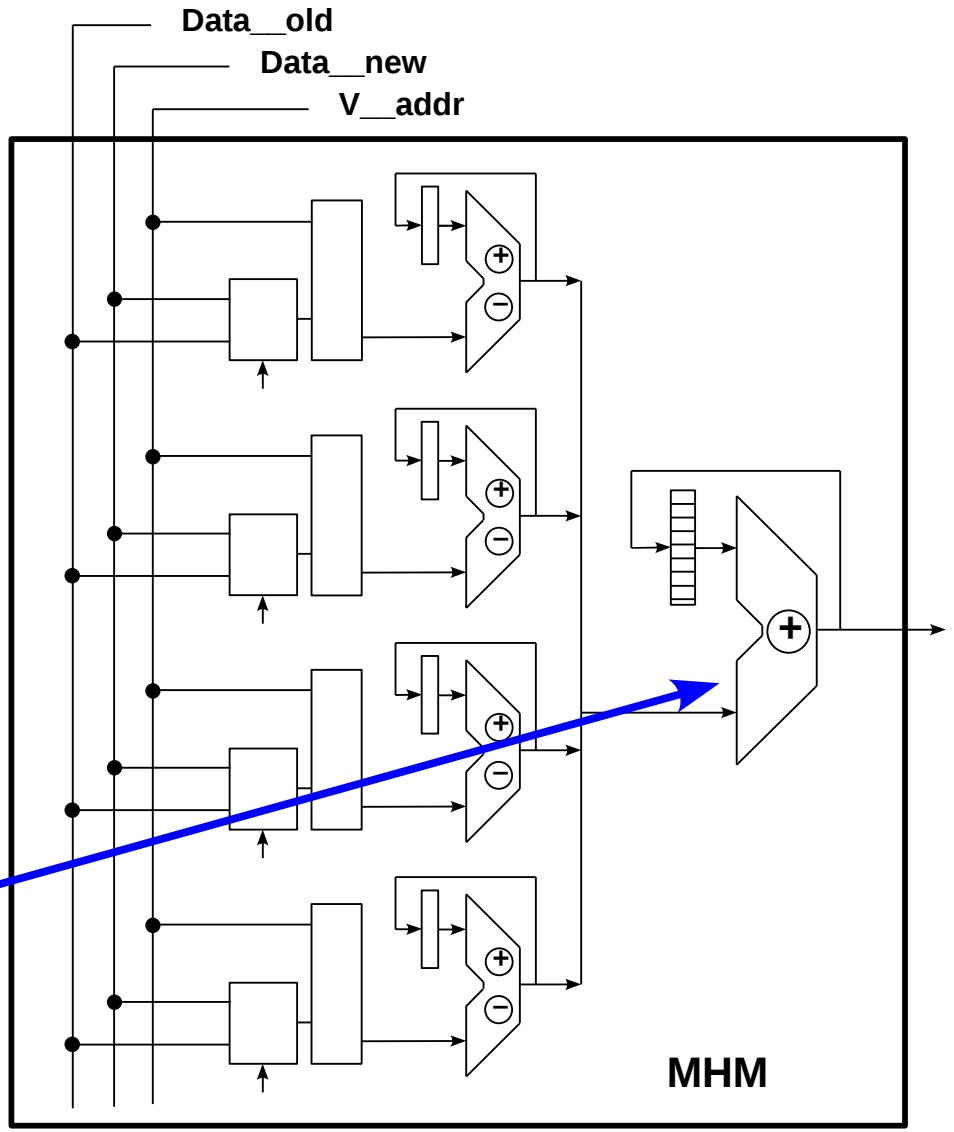
merge / sum
The Intermediate Results



Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices



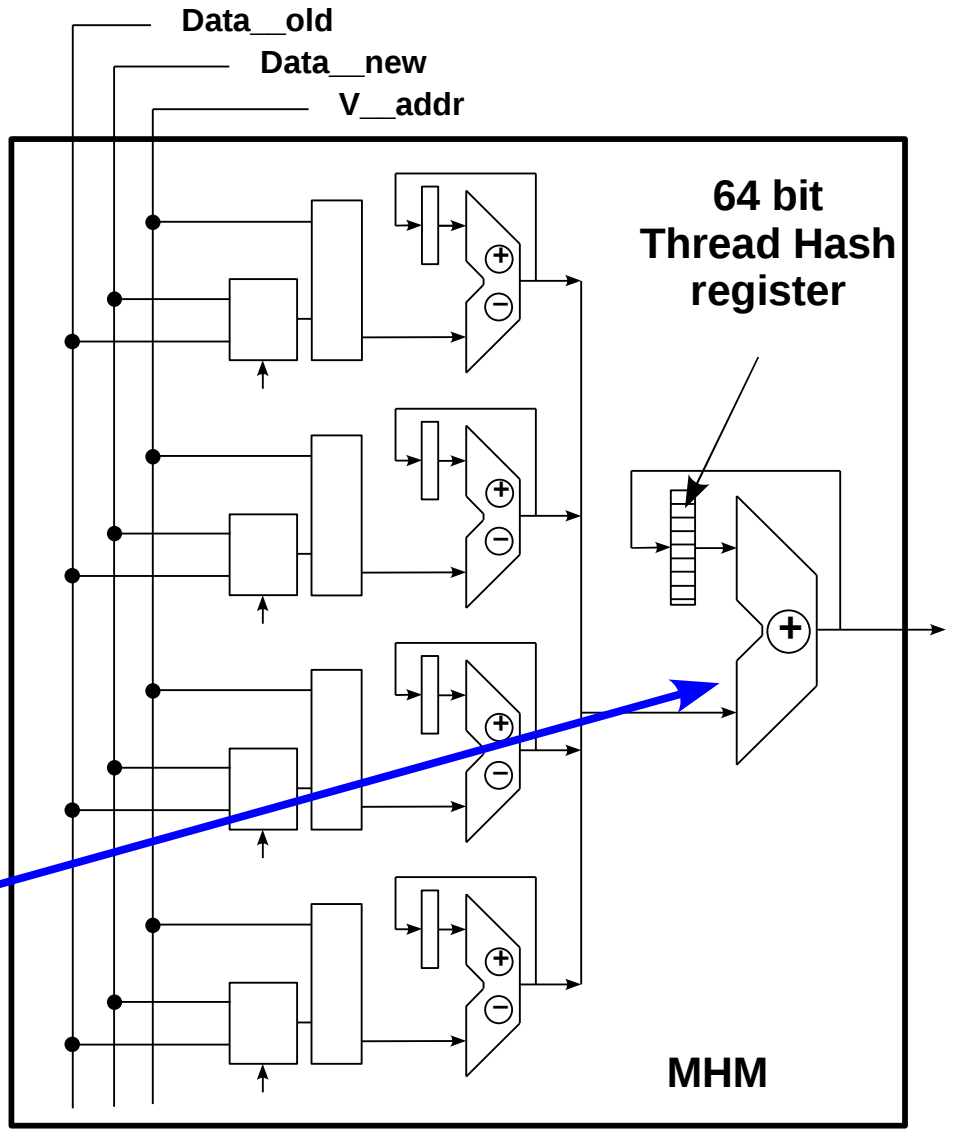
CLUSTERS
Compute Intermediate Results

merge / sum
The Intermediate Results

Commutative
Associative

Out of Order
In parallel

Flexible Implementation Choices



CLUSTERS
Compute
Intermediate
Results

merge / sum
The
Intermediate
Results

Commutative
Associative

Out of Order
In parallel

Lightweight Hardware

Lightweight Hardware

- ✓ Migrate

Lightweight Hardware

- ✓ Migrate
- ✓ Virtualize

Lightweight Hardware

- ✓ Migrate
- ✓ Virtualize
- ✓ Context-switch

Lightweight Hardware

- ✓ Migrate
- ✓ Virtualize
- ✓ Context-switch

} 1 reg = Save & Restore

Lightweight Hardware

- ✓ Migrate
- ✓ Virtualize
- ✓ Context-switch

} 1 reg = Save & Restore

- ✓ Scalable

} Core – local operations

Lightweight Hardware

- ✓ Migrate
- ✓ Virtualize
- ✓ Context-switch

} 1 reg = Save & Restore

- ✓ Scalable

} Core – local operations

- ✓ Flexible implementation choices

External determinism

Capturing state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

Conclusions

Hashing in Software

Hashing in Software

- Can hash in SW
  - No special HW
  - Slower, but still reasonable

Hashing in Software

- Can hash in SW
  - No special HW
  - Slower, but still reasonable
- Two software only implementations

Hashing in Software

- Can hash in SW
  - No special HW
  - Slower, but still reasonable
- Two software only implementations
- Incremental hashing – SW-InstantCheck-Inc
  - Atomically get : Data_Old, Data_New

Hashing in Software

- Can hash in SW
  - No special HW
  - Slower, but still reasonable
- Two software only implementations
- Incremental hashing – SW-InstantCheck-Inc
  - Atomically get : Data_Old, Data_New
- Traversal base hashing – SW-InstantCheck-Tr
  - Need type information: float, double, other

Hashing in Software

- Can hash in SW
  - No special HW
  - Slower, but still reasonable
- Two software only implementations
- Incremental hashing – SW-InstantCheck-Inc
  - Atomically get : Data_Old, Data_New
- Traversal base hashing – SW-InstantCheck-Tr
  - Need type information: float, double, other
- I/O
  - Hash (e.g., CRC) the output stream

Sources of nondeterminism (1)

Sources of nondeterminism (1)

- We want to be aware and understand them

Sources of nondeterminism (1)

- We want to be aware and understand them
- Maybe even ignore some benign nondeterminism

Sources of nondeterminism (1)

- We want to be aware and understand them
- Maybe even ignore some benign nondeterminism
- Software bugs

Sources of nondeterminism (1)

- We want to be aware and understand them
- Maybe even ignore some benign nondeterminism
- Software bugs
  - **Important** source of nondeterminism

Sources of nondeterminism (1)

- We want to be aware and understand them
- Maybe even ignore some benign nondeterminism
- Software bugs
  - **Important** source of nondeterminism
  - But this is very positive:
    - Can detect software bugs
    - If they create nondeterminism

Sources of nondeterminism (1)

- We want to be aware and understand them
- Maybe even ignore some benign nondeterminism
- Software bugs
  - **Important** source of nondeterminism
  - But this is very positive:
    - Can detect software bugs
    - If they create nondeterminism
  - Concurrency bugs create nondeterminism

Sources of nondeterminism (1)

- We want to be aware and understand them
- Maybe even ignore some benign nondeterminism
- Software bugs
  - **Important** source of nondeterminism
  - But this is very positive:
    - Can detect software bugs
    - If they create nondeterminism
  - Concurrency bugs create nondeterminism
  - In our experiments, we found **real bug in PARSEC**

Sources of nondeterminism (2)

Sources of nondeterminism (2)

- Floating point precision limitation

Sources of nondeterminism (2)

- Floating point precision limitation
  - Previous example, if we used FP:

$$(G\_0 + L\_0) + L\_1 \quad != \quad G\_0 + (L\_0 + L\_1)$$

Sources of nondeterminism (2)

- Floating point precision limitation
  - Previous example, if we used FP:

    (G_0 + L_0) + L_1 != G_0 + (L_0 + L_1)
  - Maybe the user wants to ignore this (or not)
    - Different users may have different preferences
    - We offer several options

Sources of nondeterminism (2)

- Floating point precision limitation
  - Previous example, if we used FP:

    (G_0 + L_0) + L_1 != G_0 + (L_0 + L_1)
  - Maybe the user wants to ignore this (or not)
    - Different users may have different preferences
    - We offer several options
  - Default option: round to nearest 0.001
    - In our experiments (many FP) this was effective

Sources of nondeterminism (3)

Sources of nondeterminism (3)

- Small auxiliary data structures

Sources of nondeterminism (3)

- Small auxiliary data structures
  - e.g., 1 integer out of 10 MB of state
    - Reasonable to say that it is almost deterministic

Sources of nondeterminism (3)

- Small auxiliary data structures
  - e.g., 1 integer out of 10 MB of state
    - Reasonable to say that it is almost deterministic
  - Cholesky: list free “task nodes”
    - Truly nondeterministic
      - Different size, different values, different order
    - But just one structure -> ignore it

Sources of nondeterminism (3)

- Small auxiliary data structures
  - e.g., 1 integer out of 10 MB of state
    - Reasonable to say that it is almost deterministic
  - Cholesky: list free “task nodes”
    - Truly nondeterministic
      - Different size, different values, different order
    - But just one structure -> ignore it
  - PBZip2: dangling pointers to nondeterministic memory
    - The nondeterministic memory is deallocated
    - But the dangling pointers remain part of the state

Sources of nondeterminism (3)

- Small auxiliary data structures
  - e.g., 1 integer out of 10 MB of state
    - Reasonable to say that it is almost deterministic
  - Cholesky: list free “task nodes”
    - Truly nondeterministic
      - Different size, different values, different order
    - But just one structure -> ignore it
  - PBZip2: dangling pointers to nondeterministic memory
    - The nondeterministic memory is deallocated
    - But the dangling pointers remain part of the state
  - Solution: delete from hash (with the +/– technique)

Sources of nondeterminism (4)

Sources of nondeterminism (4)

- Truly nondeterministic algorithms
  - e.g.: Simulated Annealing in PARSEC
  - Expected to be nondeterministic

Sources of nondeterminism (4)

- Truly nondeterministic algorithms
  - e.g.: Simulated Annealing in PARSEC
  - Expected to be nondeterministic
- Monte-Carlo implementation in PARSEC
  - Maybe expect to be nondeterministic

Sources of nondeterminism (4)

- Truly nondeterministic algorithms
  - e.g.: Simulated Annealing in PARSEC
  - Expected to be nondeterministic
- Monte-Carlo implementation in PARSEC
  - Maybe expect to be nondeterministic
  - In fact, turns out this implementation is deterministic
    - From parallel execution point of view
    - **Local** random number generator (no shared state)
    - Number sequence: independent of other threads

Sources of nondeterminism (4)

- Truly nondeterministic algorithms
  - e.g.: Simulated Annealing in PARSEC
  - Expected to be nondeterministic
- Monte-Carlo implementation in PARSEC
  - Maybe expect to be nondeterministic
  - In fact, turns out this implementation is deterministic
    - From parallel execution point of view
    - **Local** random number generator (no shared state)
    - Number sequence: independent of other threads
  - Input nondeterminism = like in serial program

External determinism

Capturing state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

Conclusions

Instantly Compare Memory States (1)

Instantly Compare Memory States (1)

- Small hardware

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?
- In techniques that use memory state comparison

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?
- In techniques that use memory state comparison
- Detect bugs

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?
- In techniques that use memory state comparison
- Detect bugs
  - Concurrency bugs create nondeterminism

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?
- In techniques that use memory state comparison
- Detect bugs
  - Concurrency bugs create nondeterminism
  - Atom viol, order viol, data races, some semantic bugs

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?
- In techniques that use memory state comparison
- Detect bugs
  - Concurrency bugs create nondeterminism
  - Atom viol, order viol, data races, some semantic bugs
  - Detect that a bug did occur = very fast
    - Keep bug detection always – on

Instantly Compare Memory States (1)

- Small hardware
- How can we use this ?
- In techniques that use memory state comparison
- Detect bugs
  - Concurrency bugs create nondeterminism
  - Atom viol, order viol, data races, some semantic bugs
  - Detect that a bug did occur = very fast
    - Keep bug detection always – on
  - If bug occurred (hopefully rare case) => go pin–point

Instantly Compare Memory States (2)

Instantly Compare Memory States (2)

- Systematic testing

Instantly Compare Memory States (2)

- Systematic testing
  - CHESS, Microsoft Research
    - Found bugs that stress testing missed for **months**

Instantly Compare Memory States (2)

- Systematic testing
  - CHESS, Microsoft Research
    - Found bugs that stress testing missed for **months**
  - State comparison is very frequent operation

Instantly Compare Memory States (2)

- Systematic testing
  - CHESS, Microsoft Research
    - Found bugs that stress testing missed for **months**
  - State comparison is very frequent operation
  - Currently, very conservative
    - This results in unnecessary exploring some states
    - state == synchronization – order
    - previous example: same state, different sync – order

Instantly Compare Memory States (2)

- Systematic testing
  - CHESS, Microsoft Research
    - Found bugs that stress testing missed for **months**
  - State comparison is very frequent operation
  - Currently, very conservative
    - This results in unnecessary exploring some states
    - state == synchronization – order
    - previous example: same state, different sync – order
  - May miss states (coverage):
    - Sync – order the same,
    - but if races => not same state

Instantly Compare Memory States (3)

Instantly Compare Memory States (3)

- Filtering out benign data races

Instantly Compare Memory States (3)

- Filtering out benign data races
  - You know about a race, and ask:
    - “is this race benign?”

Instantly Compare Memory States (3)

- Filtering out benign data races
  - You know about a race, and ask:
    - "is this race benign?"
  - If you flip that race

Instantly Compare Memory States (3)

- Filtering out benign data races
  - You know about a race, and ask:
    - "is this race benign?"
  - If you flip that race
  - But after some time you reach the same state

Instantly Compare Memory States (3)

- Filtering out benign data races
  - You know about a race, and ask:
    - "is this race benign?"
  - If you flip that race
  - But after some time you reach the same state
  - This means the race is benign

Instantly Compare Memory States (4)

Instantly Compare Memory States (4)

- Deterministic replay

Instantly Compare Memory States (4)

- Deterministic replay
  - Classical approach:
    - save a very precise execution log
    - Replay using that log

Instantly Compare Memory States (4)

- Deterministic replay
  - Classical approach:
    - save a very precise execution log
    - Replay using that log
  - Several recent proposals
    - Save partial log (e.g., just SYNCs)
    - At replay, search total log that generated the state

Instantly Compare Memory States (4)

- Deterministic replay
  - Classical approach:
    - save a very precise execution log
    - Replay using that log
  - Several recent proposals
    - Save partial log (e.g., just SYNCs)
    - At replay, search total log that generated the state
  - Can have hashes as part of the partial log
    - Guide the search at replay time
    - Detect replay failure

External determinism

Capturing state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

Conclusions

Setup

Setup

- Simulate the HW hashing module with PIN

Setup

- Simulate the HW hashing module with PIN
- 17 applications:
  - Sphinx3, PBZip2, PARSEC, SPLASH–2

Setup

- Simulate the HW hashing module with PIN
- 17 applications:
  - Sphinx3, PBZip2, PARSEC, SPLASH–2
- Run each application 30 times

Setup

- Simulate the HW hashing module with PIN
- 17 applications:
  - Sphinx3, PBZip2, PARSEC, SPLASH–2
- Run each application 30 times
- 8 Threads

Setup

- Simulate the HW hashing module with PIN
- 17 applications:
  - Sphinx3, PBZip2, PARSEC, SPLASH–2
- Run each application 30 times
- 8 Threads
- Compare memory state at:
  - Program end, Barriers, Loop iteration (for 2 apps)

Setup

- Simulate the HW hashing module with PIN
- 17 applications:
  - Sphinx3, PBZip2, PARSEC, SPLASH–2
- Run each application 30 times
- 8 Threads
- Compare memory state at:
  - Program end, Barriers, Loop iteration (for 2 apps)
- Randomizing thread scheduler
  - Like tools from Microsoft Research: PCT, CHESS
  - But simple scheduling policy: random

Determinism Characteristics

Determinism Characteristics

Application
blackscholes
fft
lu
radix
streamcluster
swaptions
volrend
fluidanimate
ocean
waterNS
waterSP
cholesky
pbzip2
sphinx3
barnes
canneal
radiosity

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
	lu
	radix
	streamcluster
	swaptions
	volrend
	fluidanimate
	ocean
	waterNS
	waterSP
	cholesky
	pbzip2
	sphinx3
	barnes
	canneal
	radiosity

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
	ocean
	waterNS
	waterSP
	cholesky
	pbzip2
	sphinx3
	barnes
	canneal
	radiosity

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
FP	ocean
prec	waterNS
	waterSP
	cholesky
	pbzip2
	sphinx3
	barnes
	canneal
	radiosity

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
FP	ocean
prec	waterNS
	waterSP
Ignore	cholesky
Small	pbzip2
Structs	sphinx3
	barnes
	canneal
	radiosity

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
FP	ocean
prec	waterNS
	waterSP
Ignore	cholesky
Small	pbzip2
Structs	sphinx3
	barnes
	canneal
	radiosity

Free list

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
FP	ocean
prec	waterNS
	waterSP
Ignore	cholesky
Small	pbzip2
Structs	sphinx3
	barnes
	canneal
	radiosity

Free list
Dangling ptr

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
FP	ocean
prec	waterNS
	waterSP
Ignore	cholesky
Small	pbzip2
Structs	sphinx3
	barnes
	canneal
	radiosity

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application
	blackscholes
	fft
Bit	lu
By	radix
Bit	streamcluster
	swaptions
	volrend
	fluidanimate
FP	ocean
prec	waterNS
	waterSP
Ignore	cholesky
Small	pbzip2
Structs	sphinx3
	barnes
NDet	canneal
	radiosity

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)
	blackscholes	
	fft	
Bit	lu	
By	radix	
Bit	streamcluster	
	swaptions	
	volrend	
	fluidanimate	
FP	ocean	
prec	waterNS	
	waterSP	
Ignore	cholesky	
Small	pbzip2	
Structs	sphinx3	
	barnes	
NDet	canneal	
	radiosity	

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)
	blackscholes	–
	fft	–
Bit	lu	–
By	radix	–
Bit	streamcluster	–
	swaptions	–
	volrend	–
	fluidanimate	
FP	ocean	
prec	waterNS	
	waterSP	
Ignore	cholesky	
Small	pbzip2	
Structs	sphinx3	
	barnes	
NDet	canneal	
	radiosity	

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)
	blackscholes	–
	fft	–
Bit	lu	–
By	radix	–
Bit	streamcluster	–
	swaptions	–
	volrend	–
	fluidanimate	2
FP	ocean	3
prec	waterNS	3
	waterSP	2
Ignore	cholesky	3
Small	pbzip2	2
Structs	sphinx3	2
	barnes	2
NDet	canneal	2
	radiosity	2

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points
	blackscholes	–	
	fft	–	
Bit	lu	–	
By	radix	–	
Bit	streamcluster	–	
	swaptions	–	
	volrend	–	
	fluidanimate	2	
FP	ocean	3	
prec	waterNS	3	
	waterSP	2	
Ignore	cholesky	3	
Small	pbzip2	2	
Structs	sphinx3	2	
	barnes	2	
NDet	canneal	2	
	radiosity	2	

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points Det	# Dyn check points NDet
	blackscholes	–		
	fft	–		
Bit	lu	–		
By	radix	–		
Bit	streamcluster	–		
	swaptions	–		
	volrend	–		
	fluidanimate	2		
FP	ocean	3		
prec	waterNS	3		
	waterSP	2		
Ignore	cholesky	3		
Small	pbzip2	2		
Structs	sphinx3	2		
	barnes	2		
NDet	canneal	2		
	radiosity	2		

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points Det	NDet
	blackscholes	–	101	0
	fft	–	13	0
Bit	lu	–	68	0
By	radix	–	12	0
Bit	streamcluster	–	12928	**74**
	swaptions	–	2501	0
	volrend	–	6	0
	fluidanimate	2	41	0
FP	ocean	3	871	0
prec	waterNS	3	21	0
	waterSP	2	21	0
Ignore	cholesky	3	4	0
Small	pbzip2	2	1	0
Structs	sphinx3	2	4265	0
	barnes	2		
NDet	canneal	2		
	radiosity	2		

PARSEC BUG

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points Det	NDet
	blackscholes	–	101	0
	fft	–	13	0
Bit	lu	–	68	0
By	radix	–	12	0
Bit	streamcluster	–	12928	**74**
	swaptions	–	2501	0
	volrend	–	6	0
	fluidanimate	2	41	0
FP	ocean	3	871	0
prec	waterNS	3	21	0
	waterSP	2	21	0
Ignore	cholesky	3	4	0
Small	pbzip2	2	1	0
Structs	sphinx3	2	4265	0
	barnes	2	2	16
NDet	canneal	2	0	64
	radiosity	2	0	19

PARSEC BUG

Free list
Dangling ptr
4% mem

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points Det	NDet
	blackscholes	–	101	0
	fft	–	13	0
Bit	lu	–	68	0
By	radix	–	12	0
Bit	streamcluster	–	12928	**74**
	swaptions	–	2501	0
	volrend	–	6	0
	fluidanimate	2	41	0
FP	ocean	3	871	0
prec	waterNS	3	21	0
	waterSP	2	21	0
Ignore	cholesky	3	4	0
Small	pbzip2	2	1	0
Structs	sphinx3	2	4265	0
	barnes	2	2	16
NDet	canneal	2	0	64
	radiosity	2	0	19

PARSEC BUG

Free list
Dangling ptr
4% mem

14 out of 17 apps are externally deterministic

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points Det	NDet
	blackscholes	–	101	0
	fft	–	13	0
Bit	lu	–	68	0
By	radix	–	12	0
Bit	streamcluster	–	12928	**74**
	swaptions	–	2501	0
	volrend	–	6	0
	fluidanimate	2	41	0
FP	ocean	3	871	0
prec	waterNS	3	21	0
	waterSP	2	21	0
Ignore	cholesky	3	4	0
Small	pbzip2	2	1	0
Structs	sphinx3	2	4265	0
	barnes	2	2	16
NDet	canneal	2	0	64
	radiosity	2	0	19

PARSEC BUG

Free list
Dangling ptr
4% mem

14 out of 17 apps are externally deterministic
Even though they were not written for determinism (but high performance)

Determinism Characteristics

Det Type	Application	First Ndet Run (b-b-b)	# Dyn check points Det	NDet
	blackscholes	–	101	0
	fft	–	13	0
Bit	lu	–	68	0
By	radix	–	12	0
Bit	streamcluster	–	12928	**74**
	swaptions	–	2501	0
	volrend	–	6	0
	fluidanimate	2	41	0
FP	ocean	3	871	0
prec	waterNS	3	21	0
	waterSP	2	21	0
Ignore	cholesky	3	4	0
Small	pbzip2	2	1	0
Structs	sphinx3	2	4265	0
	barnes	2	2	16
NDet	canneal	2	0	64
	radiosity	2	0	19

PARSEC BUG

Free list
Dangling ptr
4% mem

14 out of 17 apps are externally deterministic

Even though they were not written for determinism (but high performance)

Variety of thread communication, global variables, benign races, floating point

Distribution of Nondeterminism

Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs

Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?

Distribution of Nondeterminism

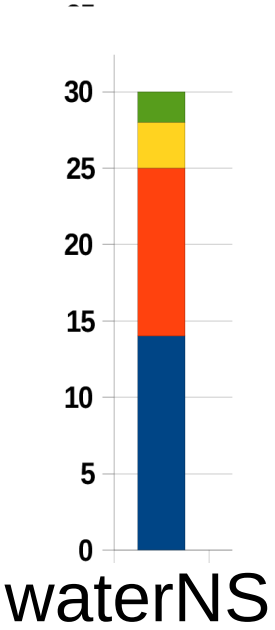
- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism

Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism

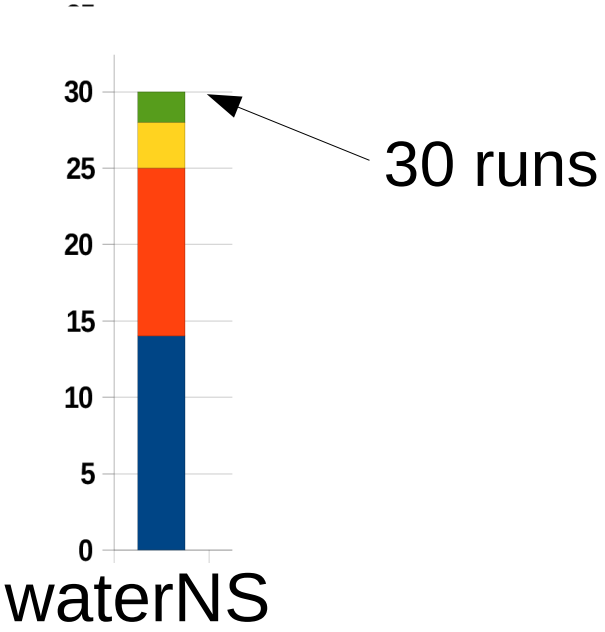
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



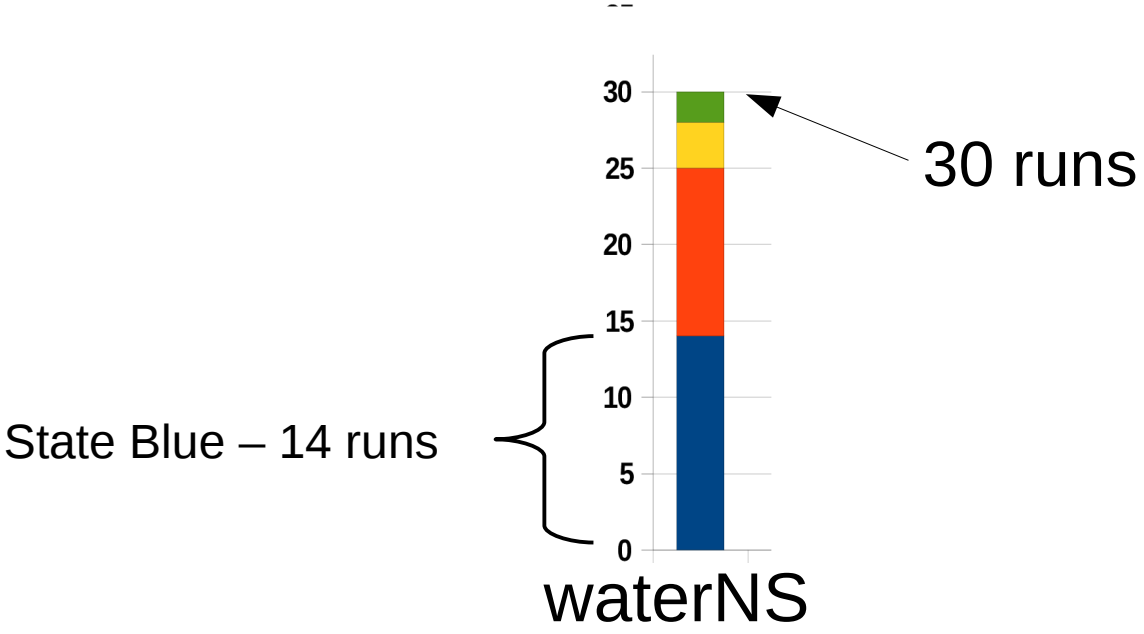
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



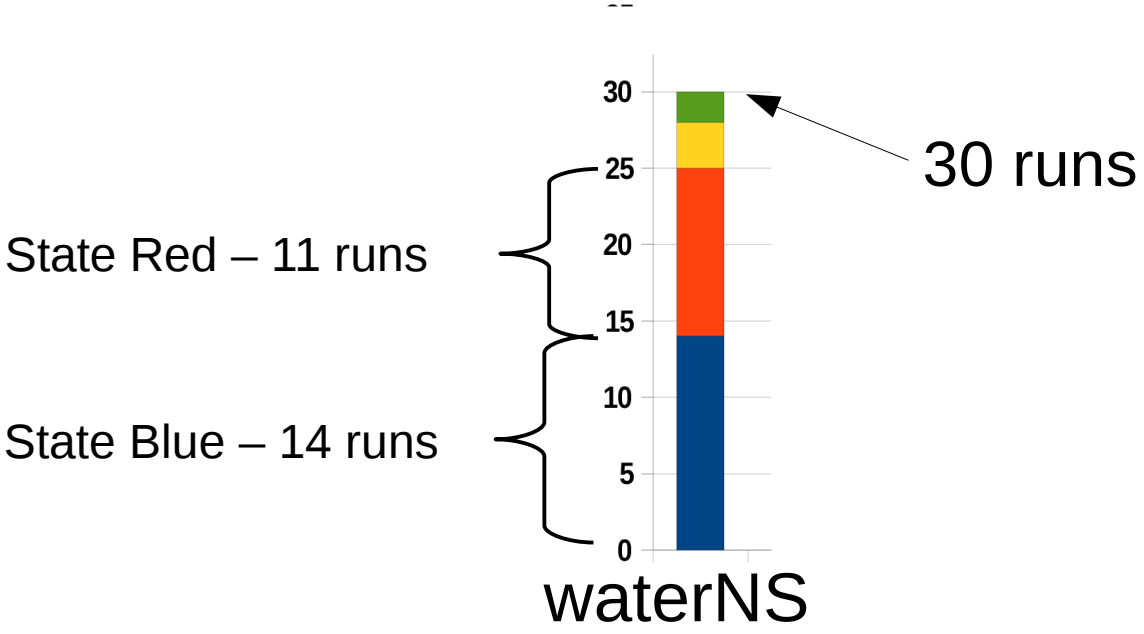
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



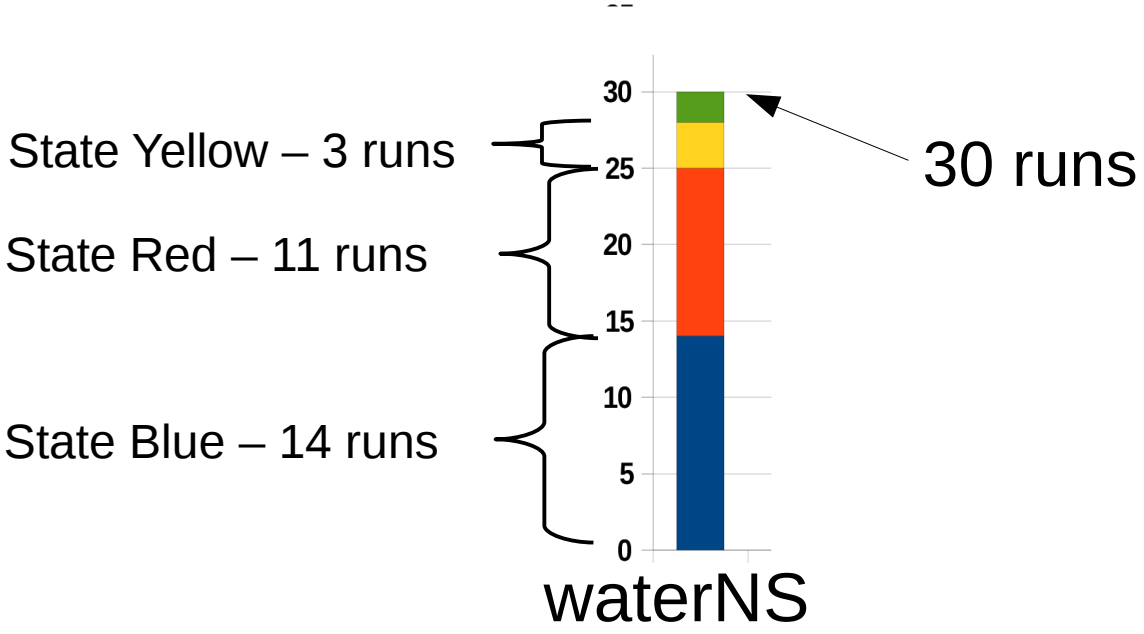
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



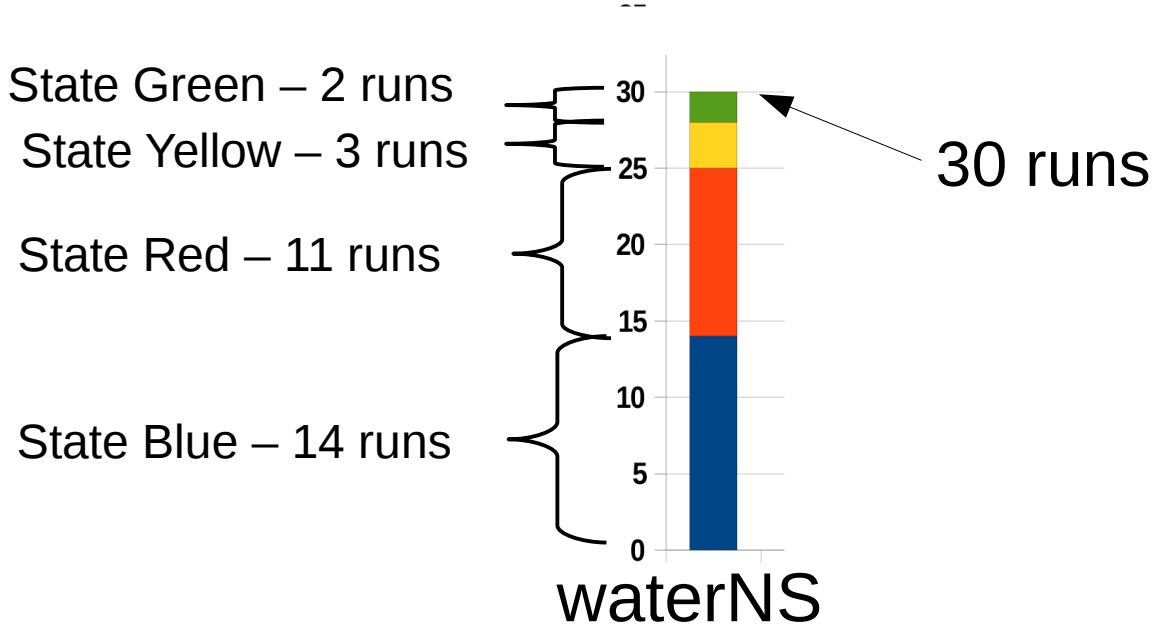
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



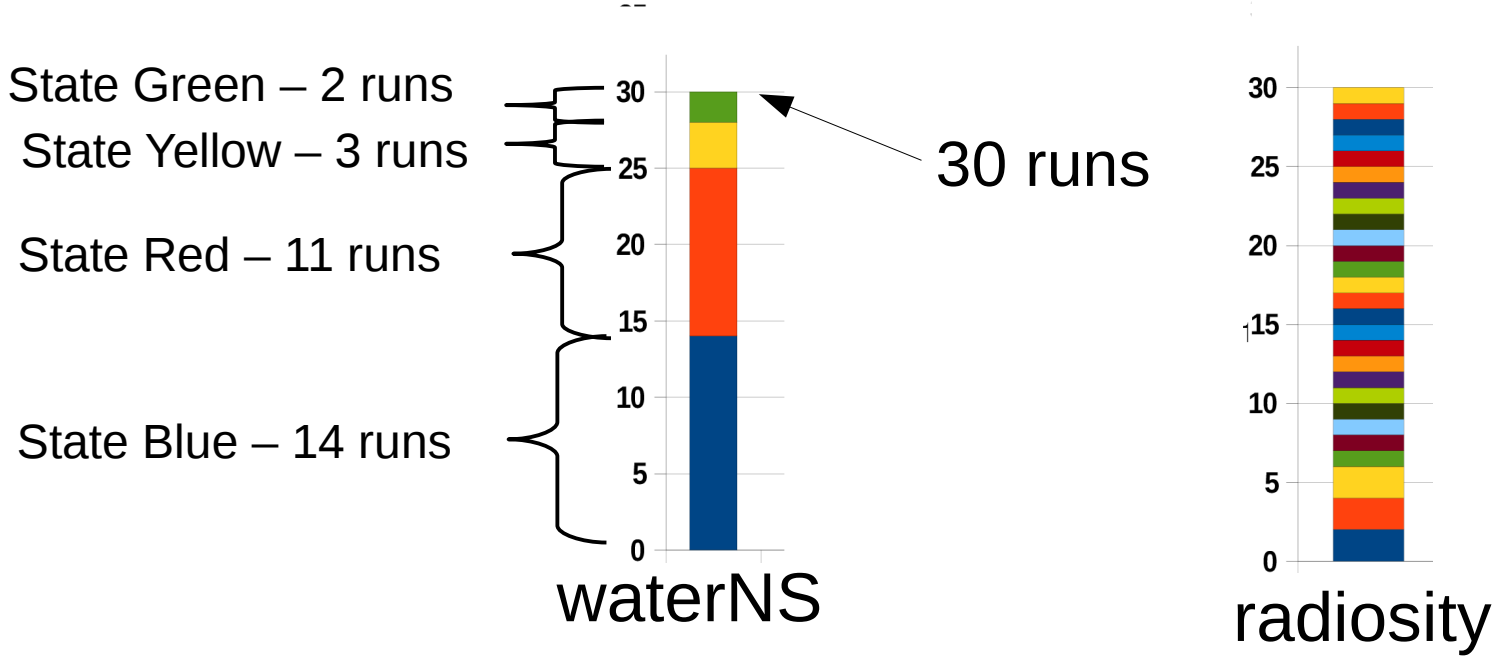
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



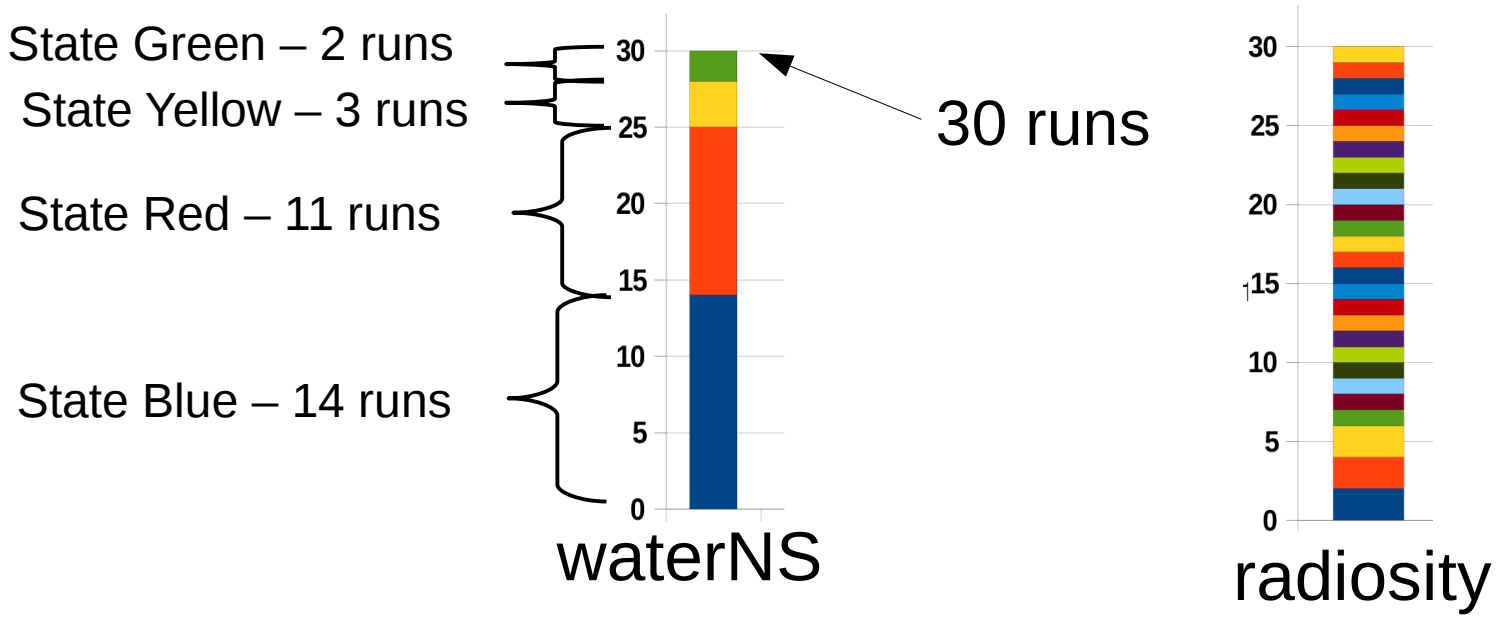
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism



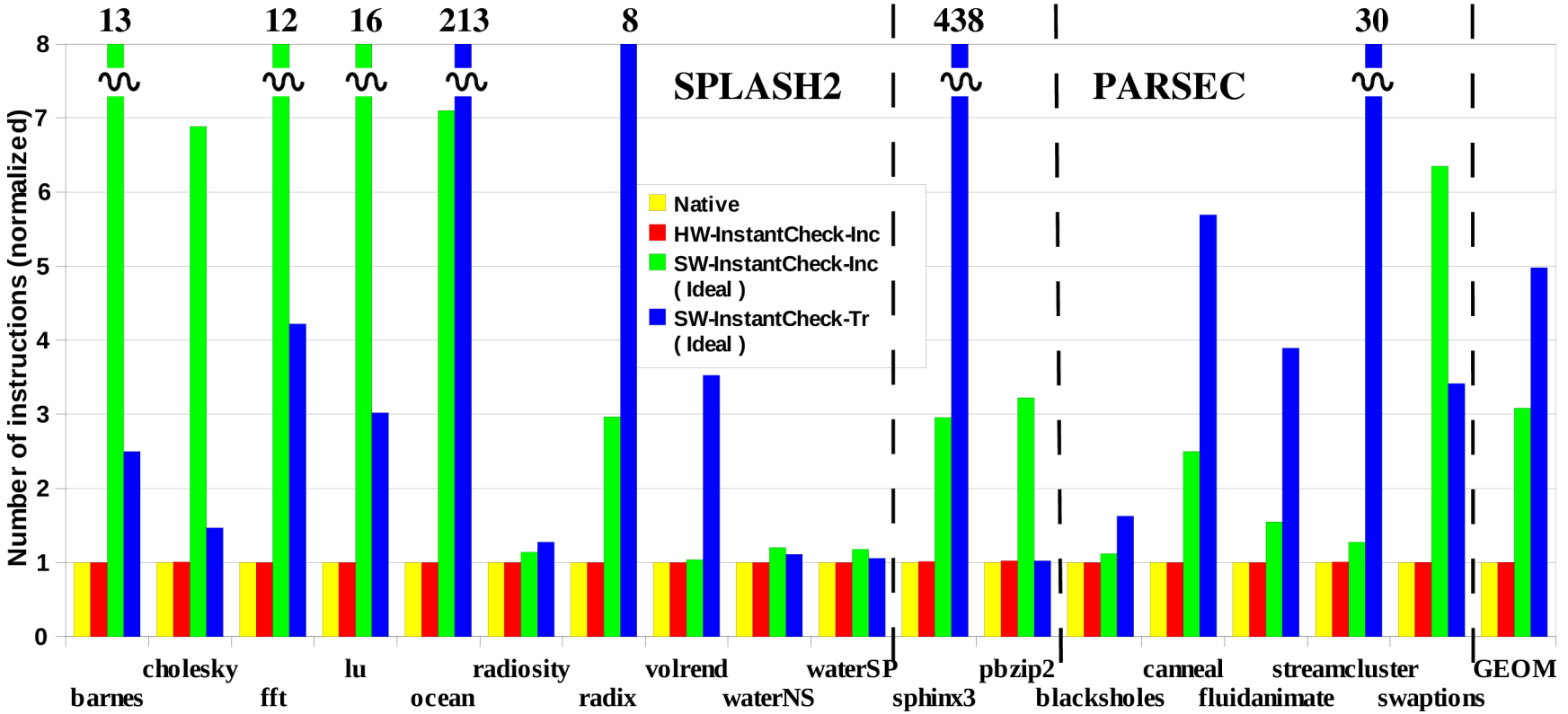
Distribution of Nondeterminism

- Previous slide: find NonDet in 2 – 3 runs
- But maybe just luck: what is the distribution of NonDet?
  - Scattered – high changes of finding nondeterminism
  - Biased – low chances of finding nondeterminism
- **Scattered => high chances of finding Nondeterminism**

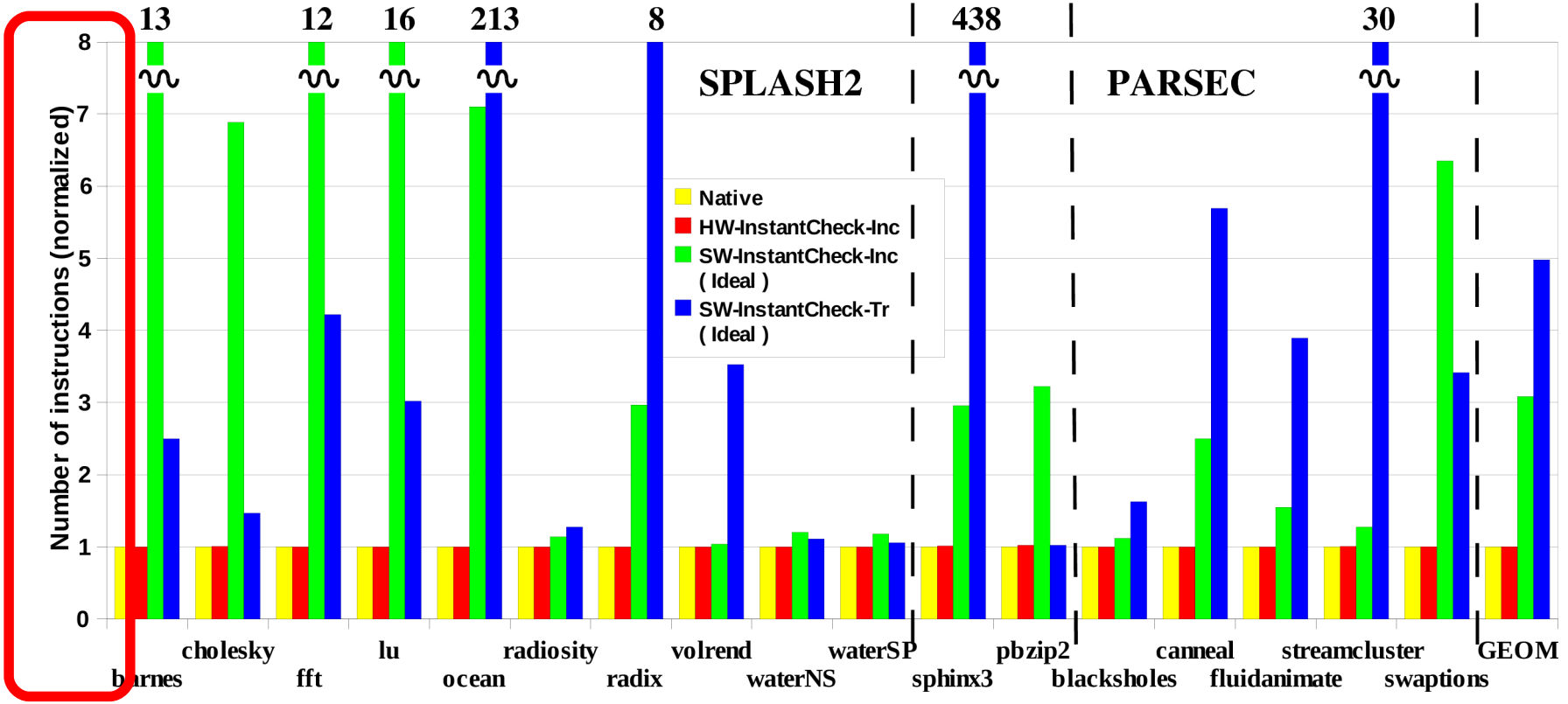


Overhead

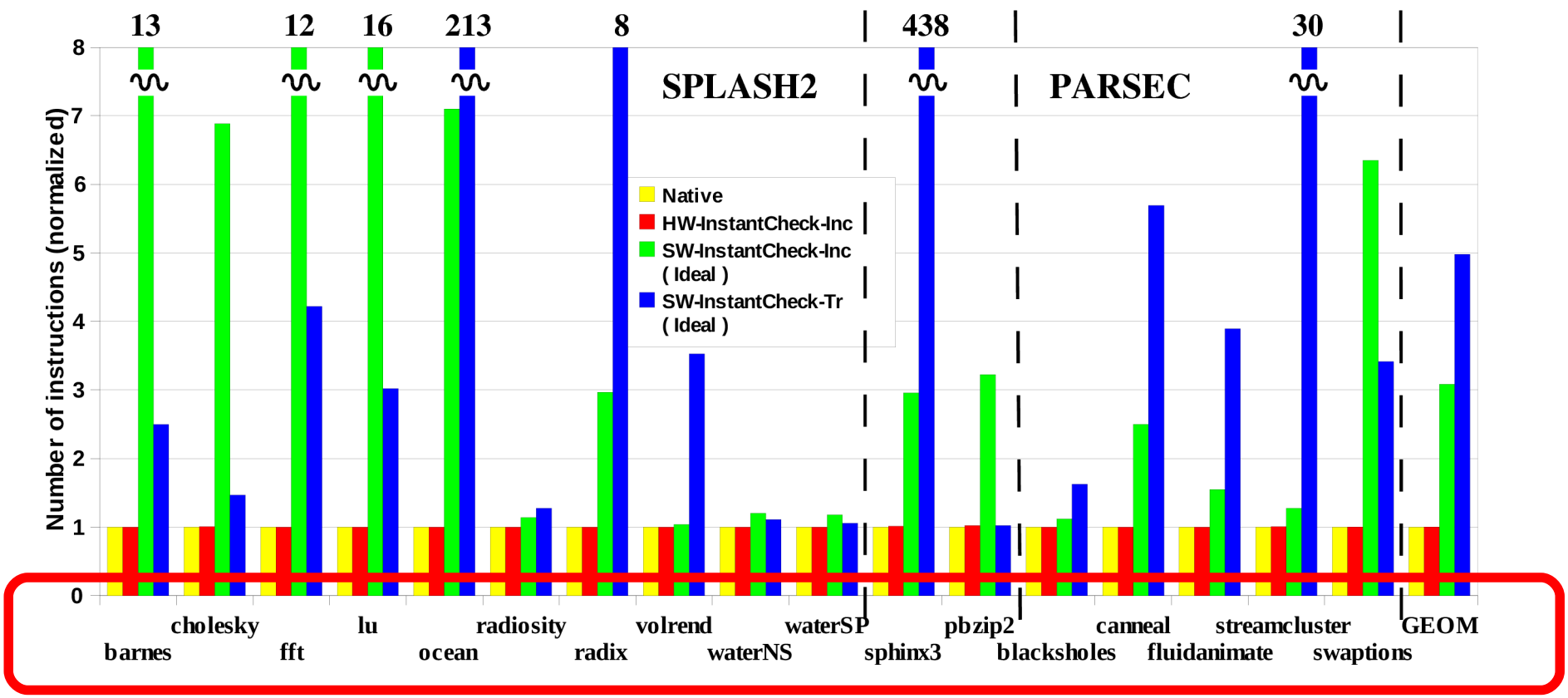
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blacksholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

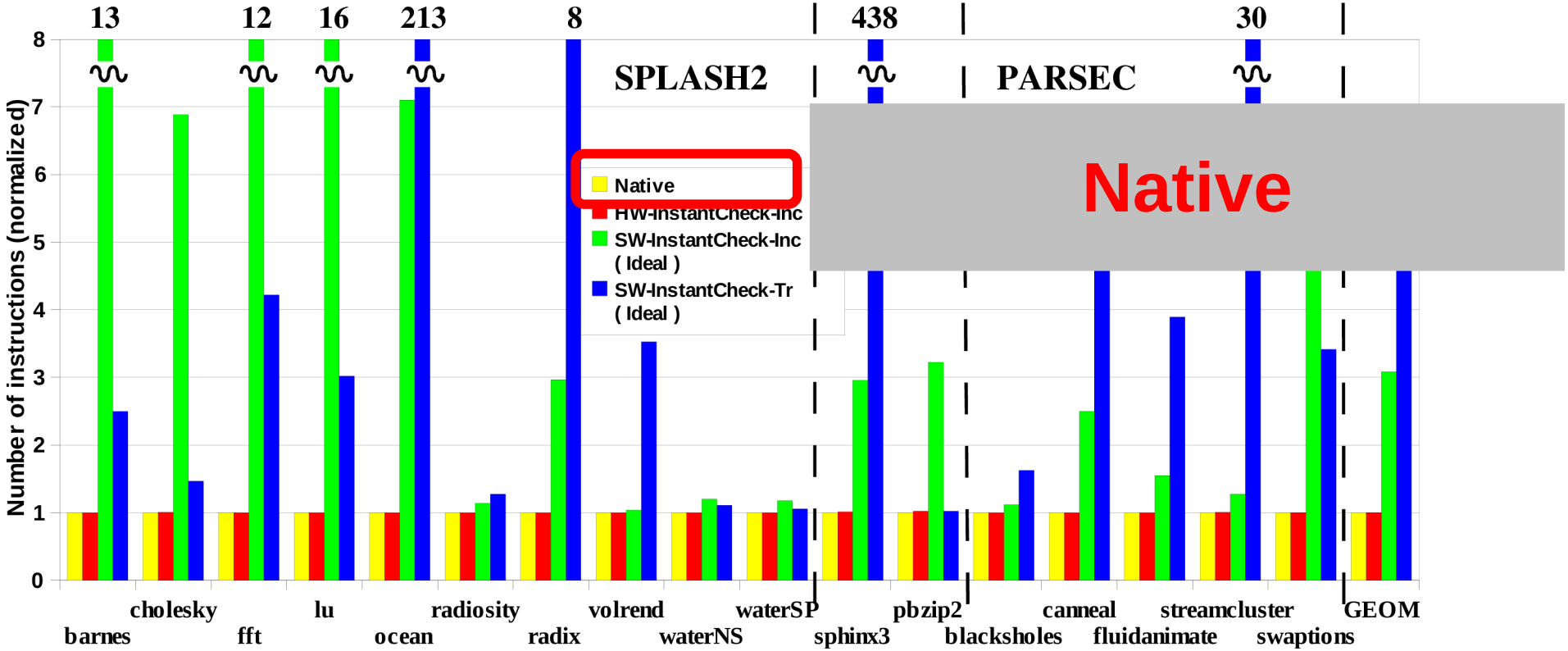
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blacksholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

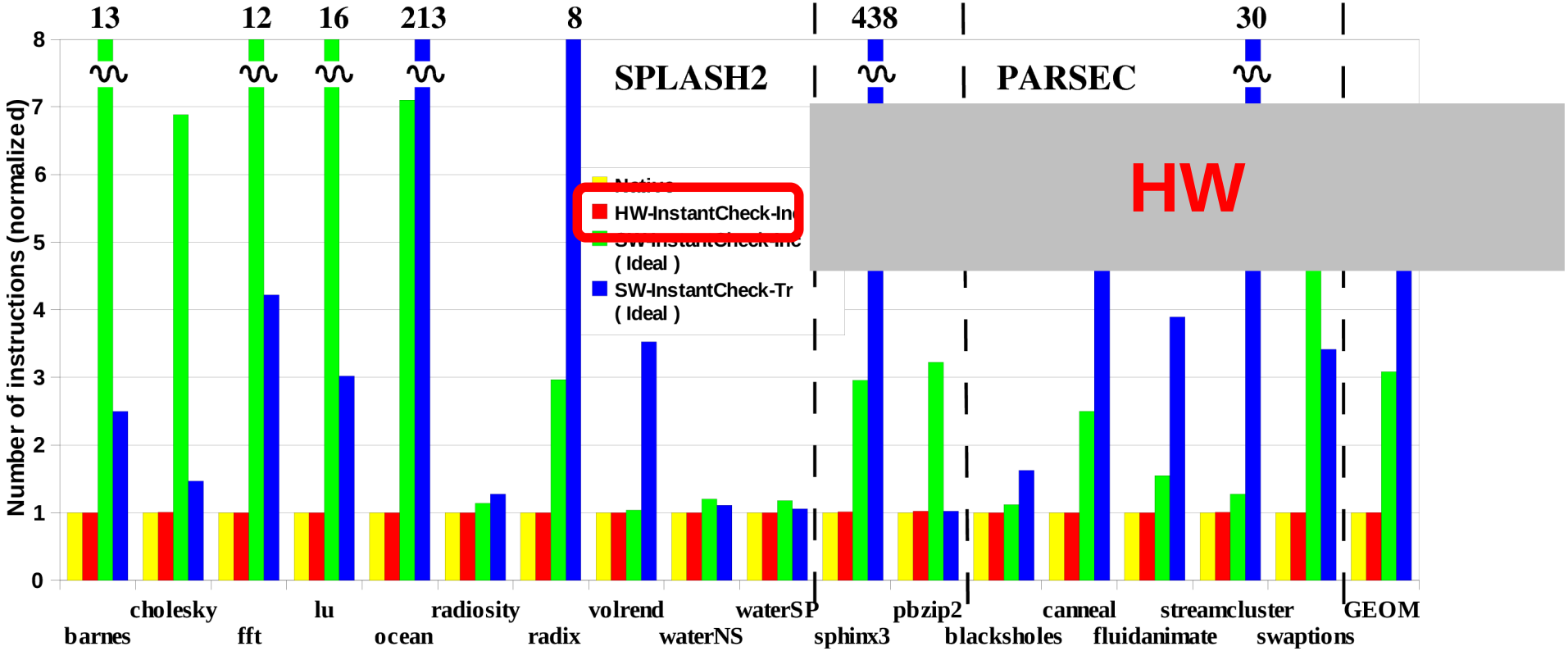
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blacksholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

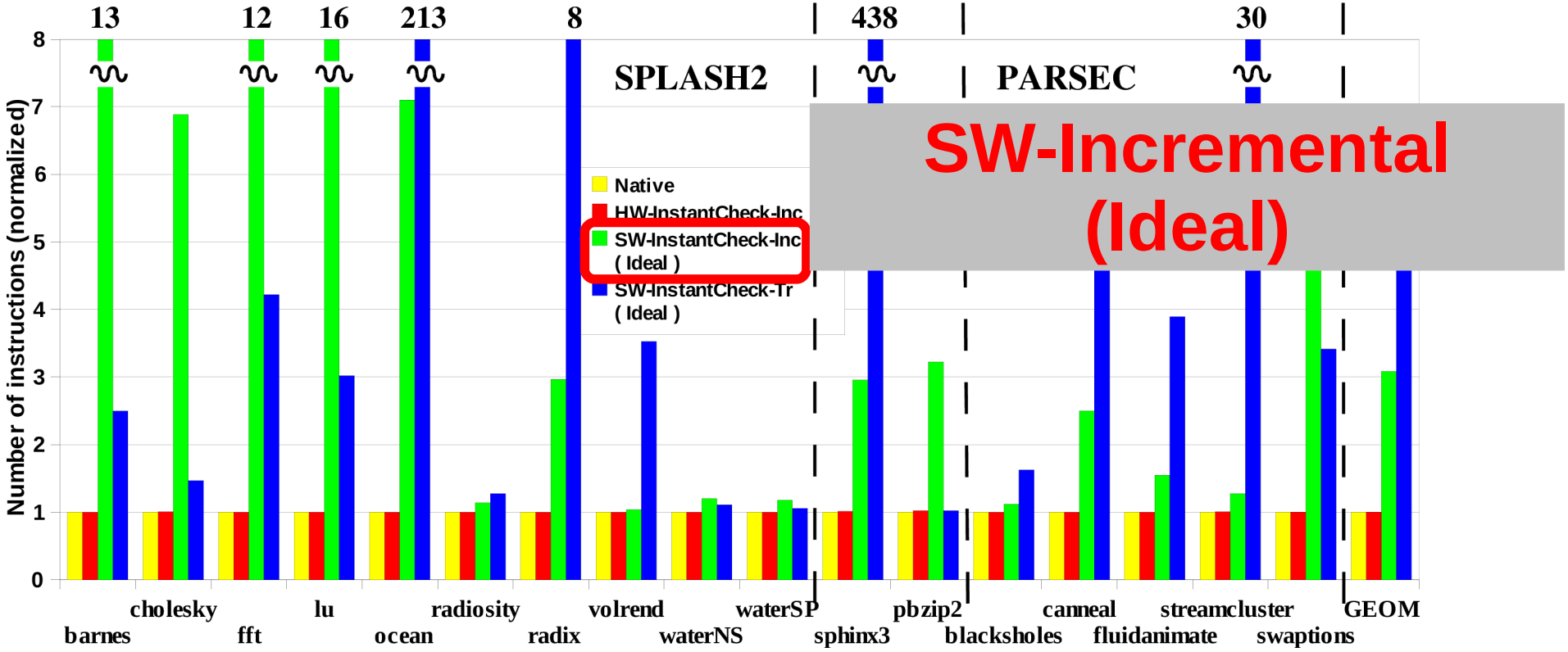
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blacksholes
canneal
fluidanimate
streamcluster
swaptions
GEOM
Native

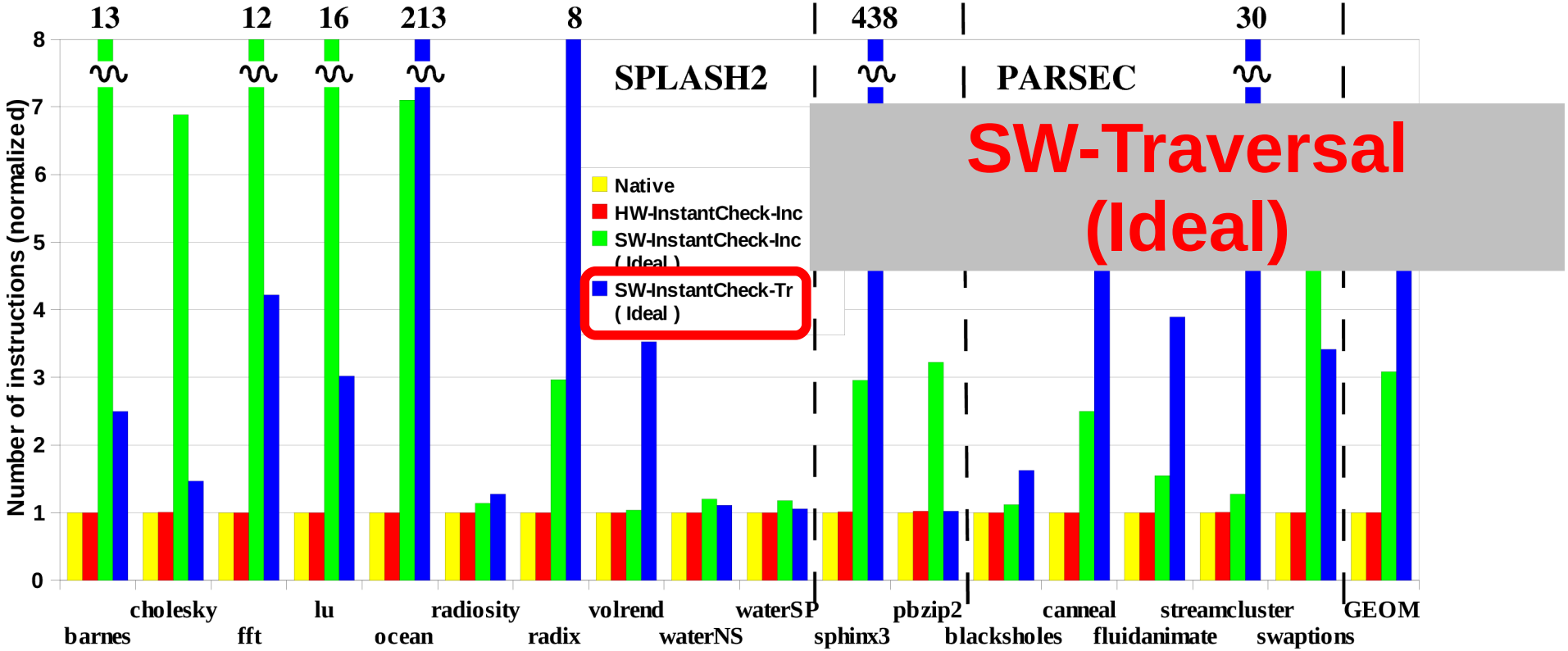
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
HW
HW-InstantCheck-In
(Ideal)
SW-InstantCheck-Tr
(Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blacksholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

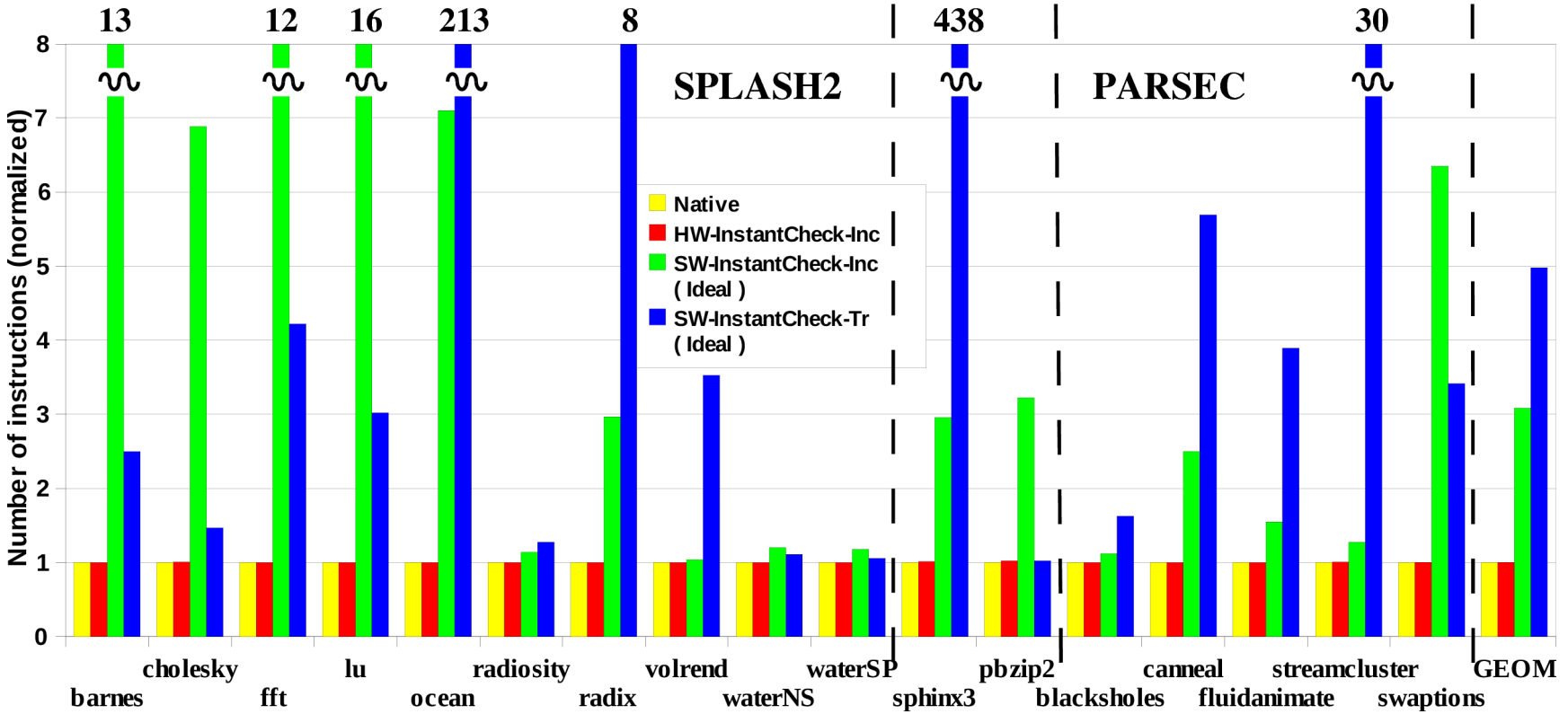
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
SW-Incremental (Ideal)
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blackscholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

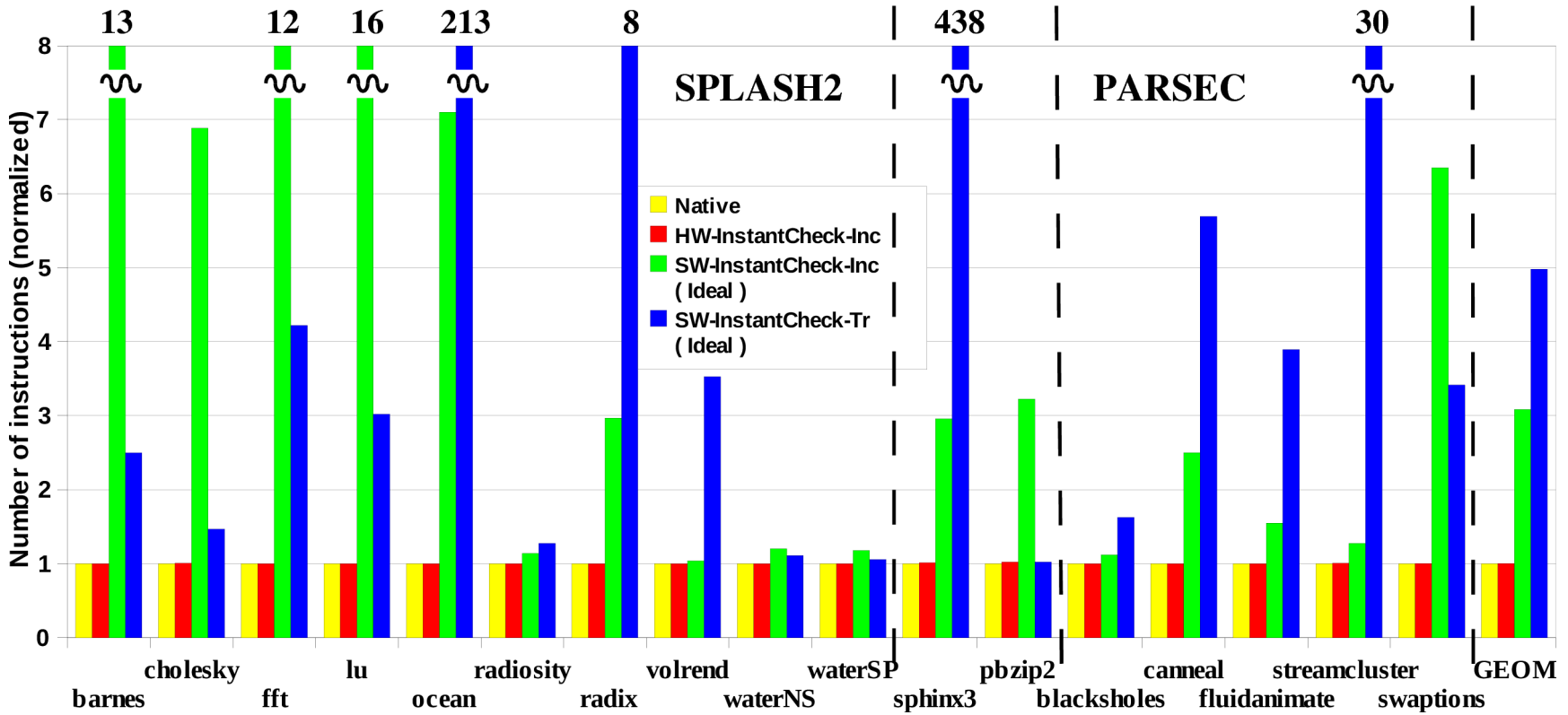
Overhead


13
12
16
213
8
438
30
SPLASH2
PARSEC
SW-Traversal (Ideal)
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blackscholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

Overhead

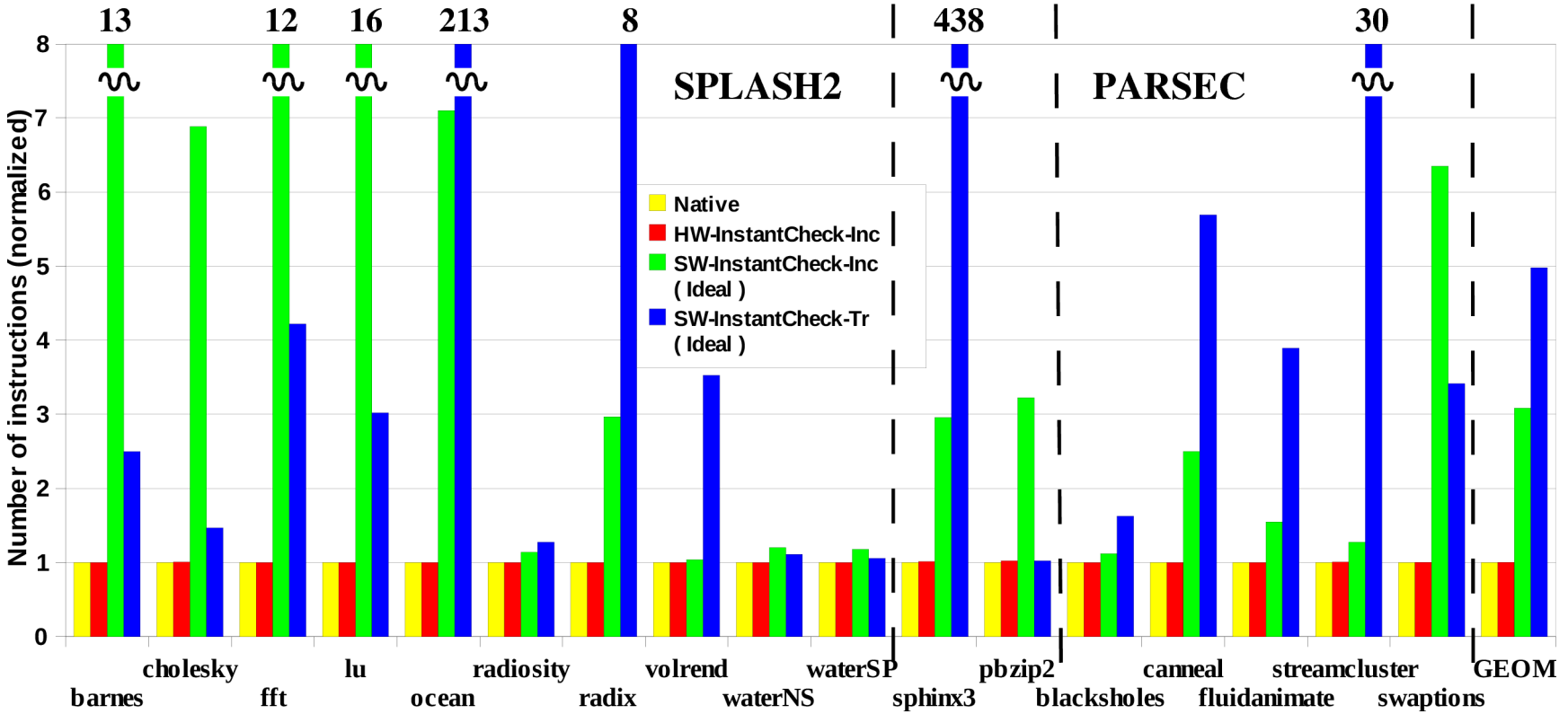

13
12
16
213
8
438
30
SPLASH2
PARSEC
Native
HW-InstantCheck-Inc
SW-InstantCheck-Inc (Ideal)
SW-InstantCheck-Tr (Ideal)
Number of instructions (normalized)
0
1
2
3
4
5
6
7
8
barnes
cholesky
fft
lu
ocean
radiosity
radix
volrend
waterNS
waterSP
sphinx3
pbzip2
blacksholes
canneal
fluidanimate
streamcluster
swaptions
GEOM

Overhead



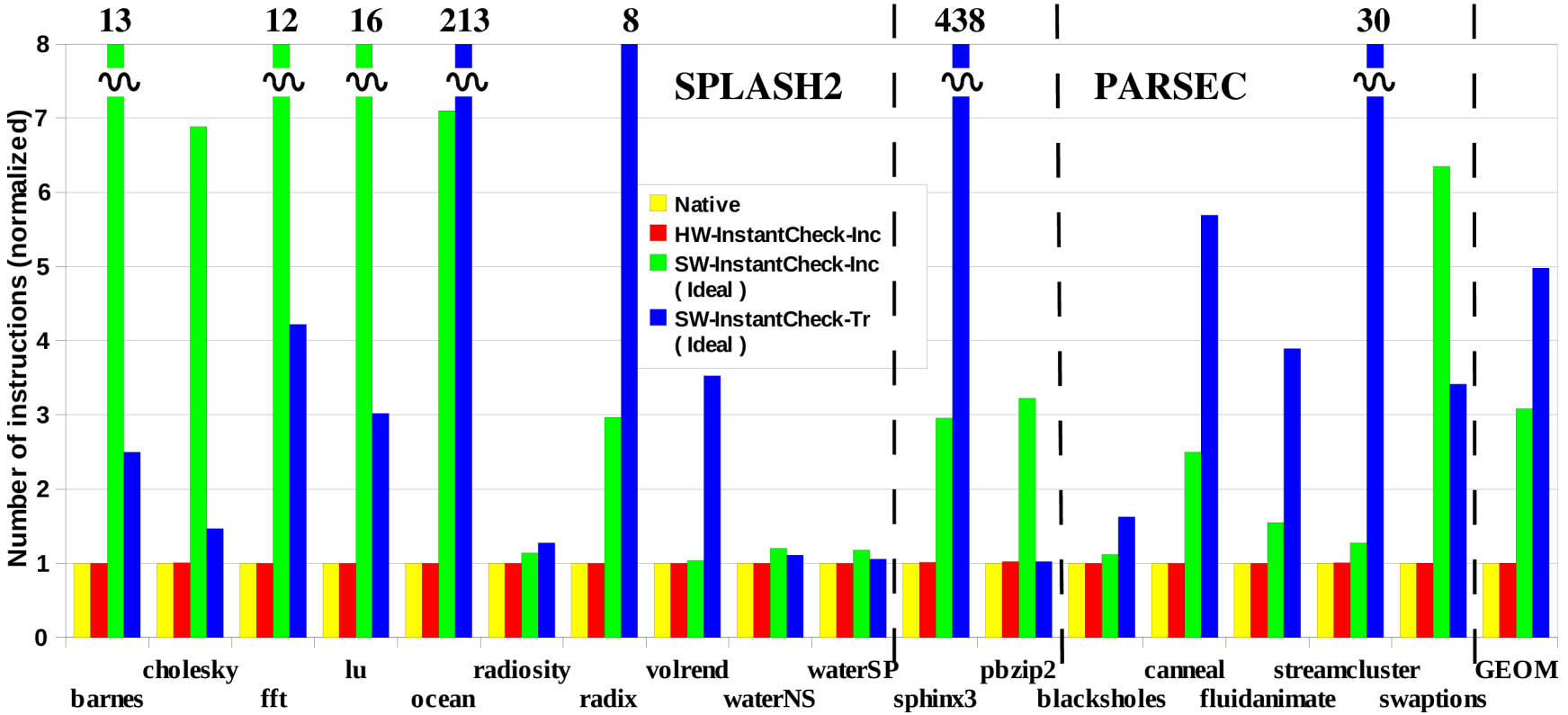
- HW: 0.3 % -> always-on

Overhead



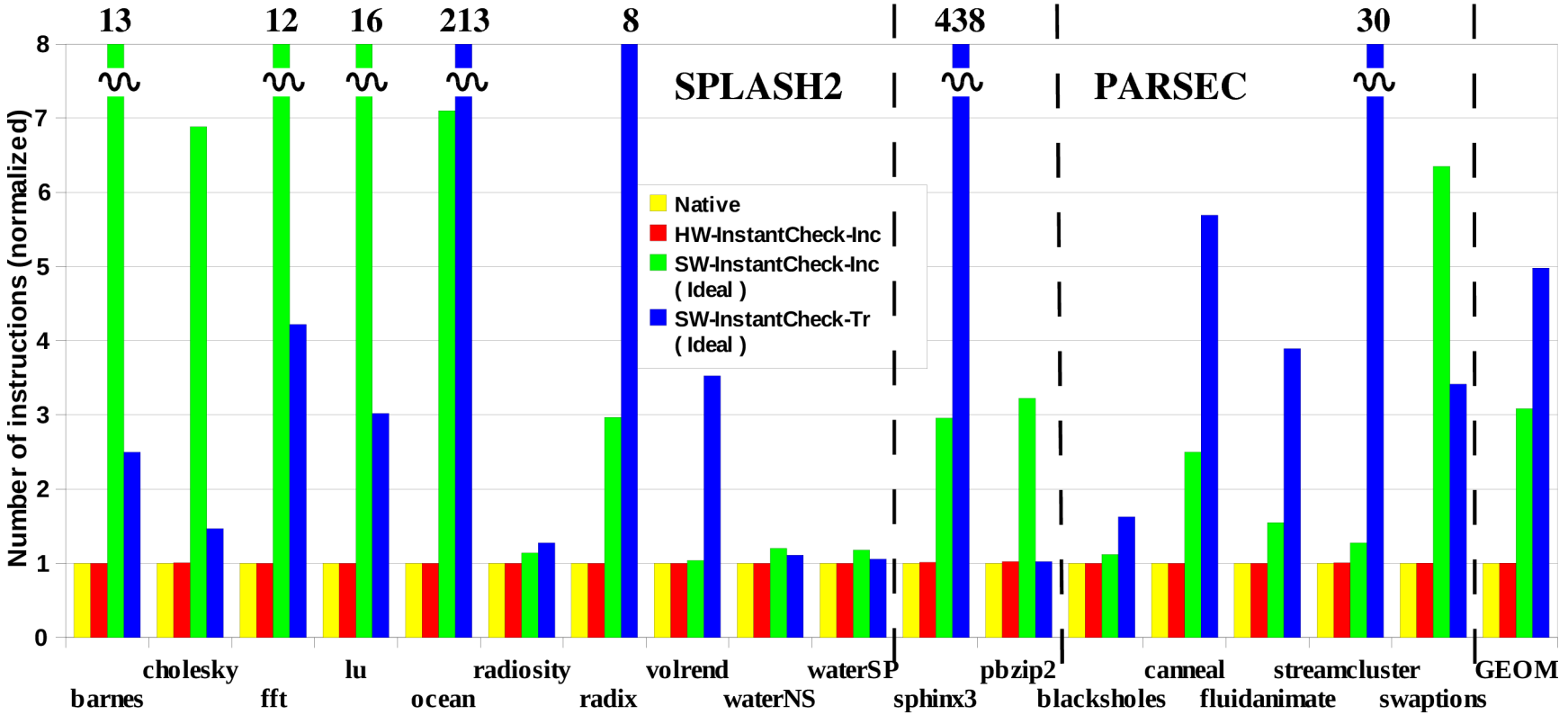
- HW: 0.3 % -> always-on
- Geometric Mean: SW-Incremental = 3X , SW-Traversal = 5X

Overhead



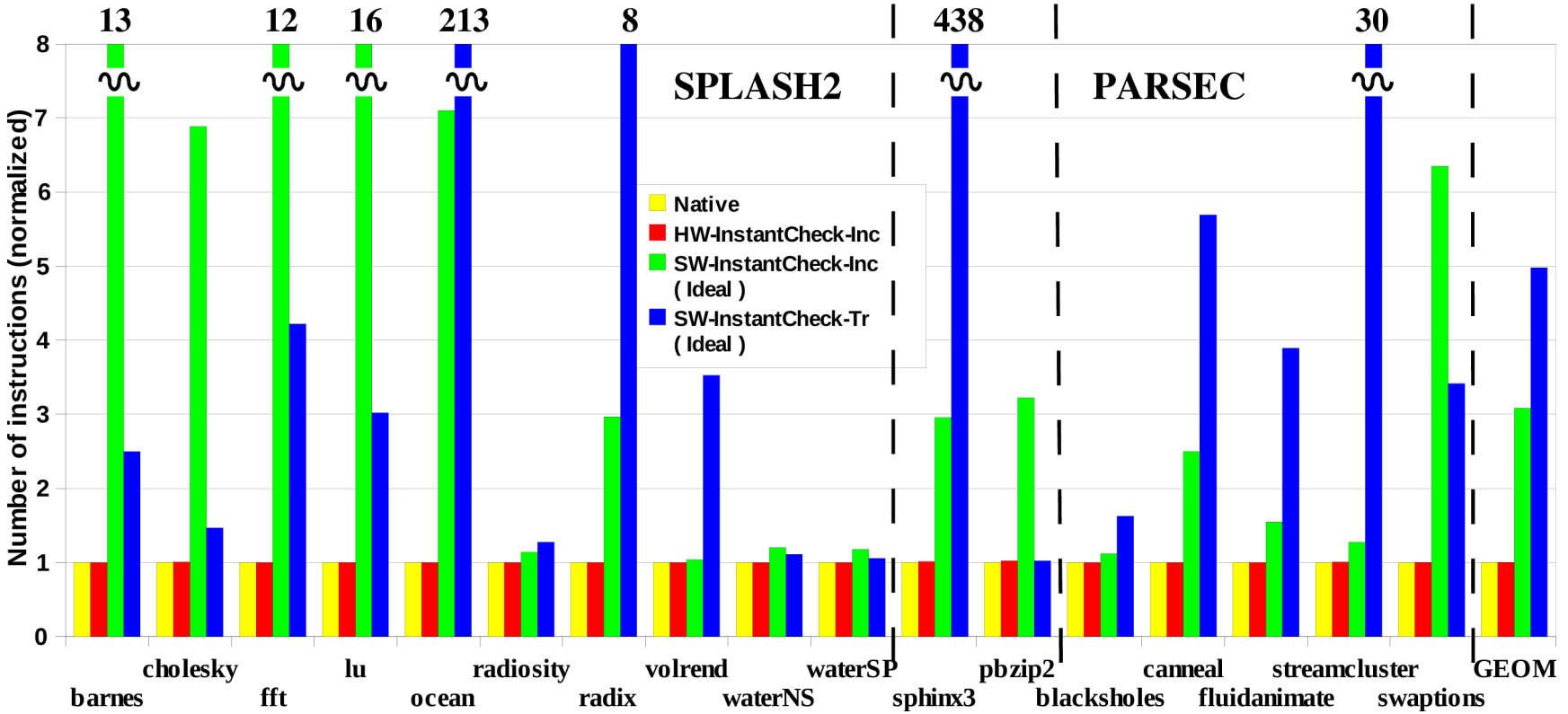
- HW: 0.3 % -> always-on
- Geometric Mean: SW-Incremental = 3X , SW-Traversal = 5X
  - Very reasonable if you don't have the HW support

Overhead



- HW: 0.3 % -> always-on
- Geometric Mean: SW-Incremental = 3X , SW-Traversal = 5X
  - Very reasonable if you don't have the HW support
  - For some apps, one of SW-Incremental or SW-Traversal is clearly better

Overhead



- HW: 0.3 % -> always-on
- Geometric Mean: SW-Incremental = 3X , SW-Traversal = 5X
  - Very reasonable if you don't have the HW support
  - For some apps, one of SW-Incremental or SW-Traversal is clearly better
  - Should choose the appropriate one

External determinism

Capturing state w/ Incremental Hashing

Hardware system

Software system

Other uses of hardware primitive

Evaluation

Conclusions

Hardware Hashing

Hardware Hashing

- Powerful, versatile and lightweight primitive

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**, read (data)

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**, read (data)
  - Detect data races during systematic testing

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**, read (data)
  - Detect data races during systematic testing
- Bond & McKinley — **execution context**

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**, read (data)
  - Detect data races during systematic testing
- Bond & McKinley — **execution context**, RetAddr

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**, read (data)
  - Detect data races during systematic testing
- Bond & McKinley — **execution context**, RetAddr
  - Java serial programs
  - Several testing and debugging tasks

Hardware Hashing

- Powerful, versatile and lightweight primitive
- This paper — **execution state**, write (addr, data)
  - 5 applications
  - Checking determinism, detecting software bugs
  - Systematic testing, benign races, deterministic replay
- Our previous paper — **execution history**, read (data)
  - Detect data races during systematic testing
- Bond & McKinley — **execution context**, RetAddr
  - Java serial programs
  - Several testing and debugging tasks
- Other operations, other information, other applications ?

InstantCheck

InstantCheck

- Check external determinism with low overhead

InstantCheck

- Check external determinism with low overhead
- Lightweight hardware
  - **Instantly** compare memory state

InstantCheck

- Check external determinism with low overhead
- Lightweight hardware
  - **Instantly** compare memory state
- Incremental hashing
  - High – performance hardware implementation
  - Enables flexible implementation choices

InstantCheck

- Check external determinism with low overhead
- Lightweight hardware
  - **Instantly** compare memory state
- Incremental hashing
  - High – performance hardware implementation
  - Enables flexible implementation choices
- Present four other applications of HW primitive

InstantCheck

- Check external determinism with low overhead
- Lightweight hardware
  - **Instantly** compare memory state
- Incremental hashing
  - High – performance hardware implementation
  - Enables flexible implementation choices
- Present four other applications of HW primitive
- Some applications are externally deterministic
  - Not written for determinism (but high performance)

InstantCheck

- Check external determinism with low overhead
- Lightweight hardware
  - **Instantly** compare memory state
- Incremental hashing
  - High – performance hardware implementation
  - Enables flexible implementation choices
- Present four other applications of HW primitive
- Some applications are externally deterministic
  - Not written for determinism (but high performance)
- Helped us find a real bug in PARSEC

InstantCheck: Checking the determinism of parallel programs using on-the-fly incremental hashing

Adrian Nistor, Darko Marinov, Josep Torrellas

MICRO '10

i-acoma group