A Midsummer Night's Tree: Efficient and High Performance Secure SCM

Samuel Thomas[1], Kidus Workneh[2], Jac McCarty[3], Joseph Izraelevitz[2], Tamara Lehman[2], R. Iris Bahar[4]

1 BROWN

2 University of Colorado Boulder

3

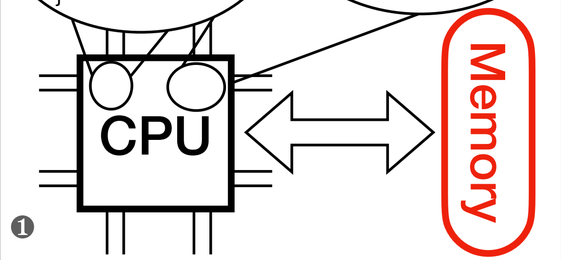
4 School of Mines Colorado 1874

Problem

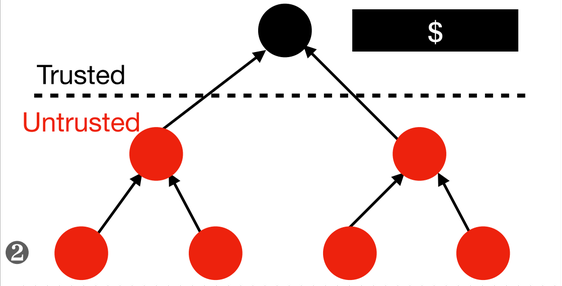
```
arr[i] = n;

main() {
  int x = arr[i];
}
```

```
main() {
  while (1)
    clflush(arr[i]);
}
```



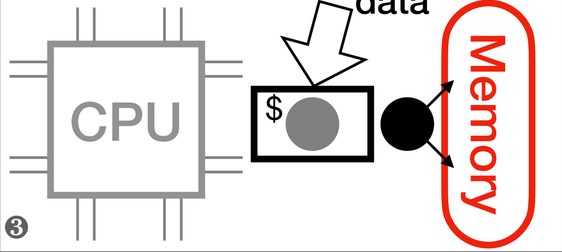
❶



❷

Power loss

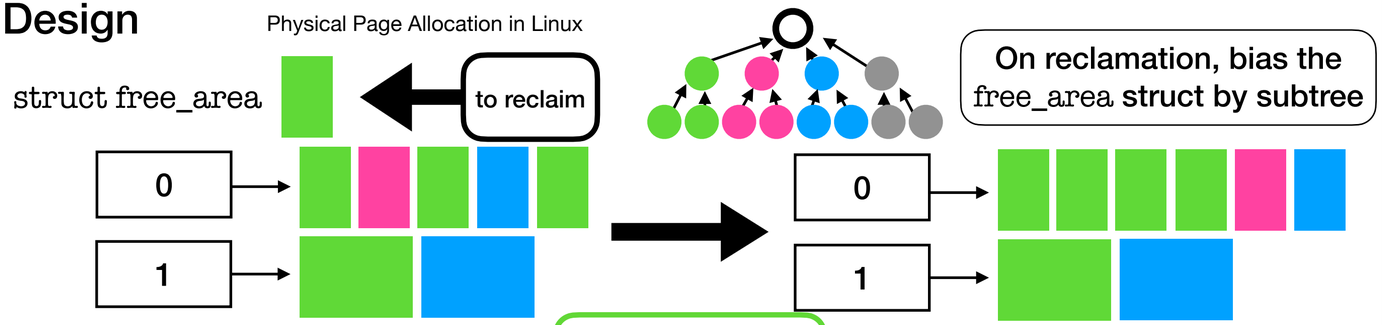
Metadata inconsistent with data



❸

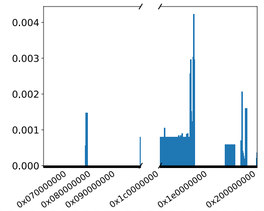
Design

Physical Page Allocation in Linux



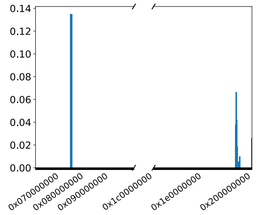
On reclamation, bias the free_area struct by subtree

Multiprogram workloads have multiple hot address spaces in parallel



AMNT++ offloads complexity to software

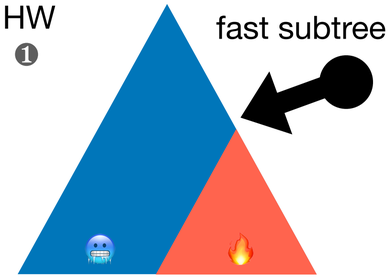
Hardware Overhead



Biased physical page allocation increases in memory locality

SW SW

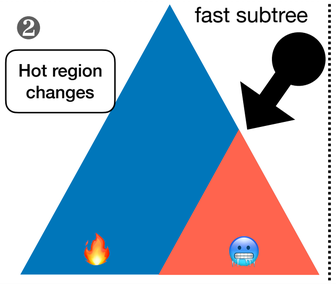
HW ❶



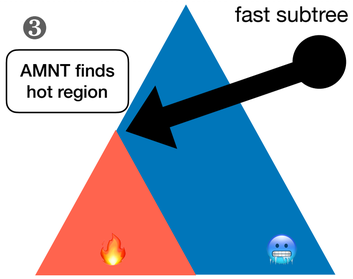
Performance

AMNT performs hot region tracking so that most accesses use the fast persistence protocol

❷ Hot region changes



❸ AMNT finds hot region



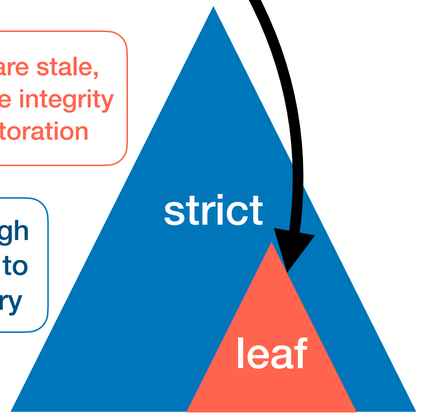
Recovery

HW

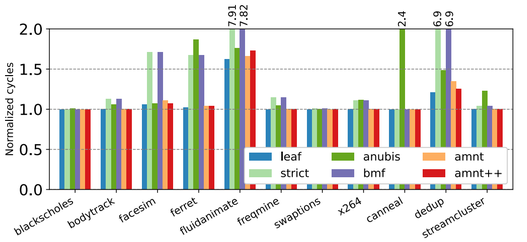
Small percentage of nodes need to be recovered in AMNT

Leaf: inner nodes are stale, must reconstruct the integrity tree on power restoration

Strict: write through all cached nodes to persistent memory



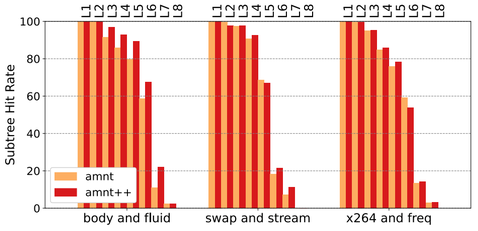
Evaluation



Performance: AMNT shows improvements of 41% on average

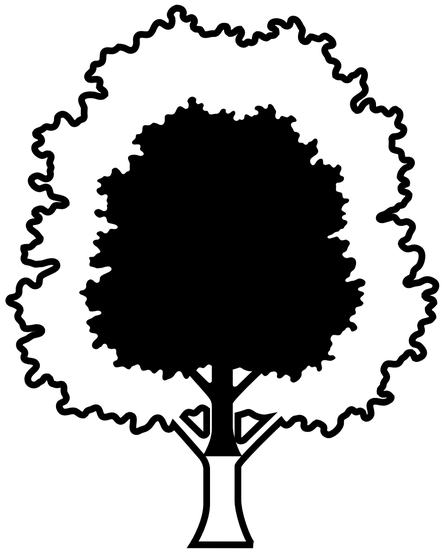
	Volatile Overhead	Nonvolatile Overhead
AMNT	**96 bytes**	**64 bytes**
Anubis, ISCA19	37kB	64 bytes
BMF, MICRO21	768 bytes	4kB

Area: AMNT requires less hardware overhead



HWSW: AMNT++ can benefit subtree locality in multiprogram workloads

1 2 3 4
BROWN
University of Colorado Boulder

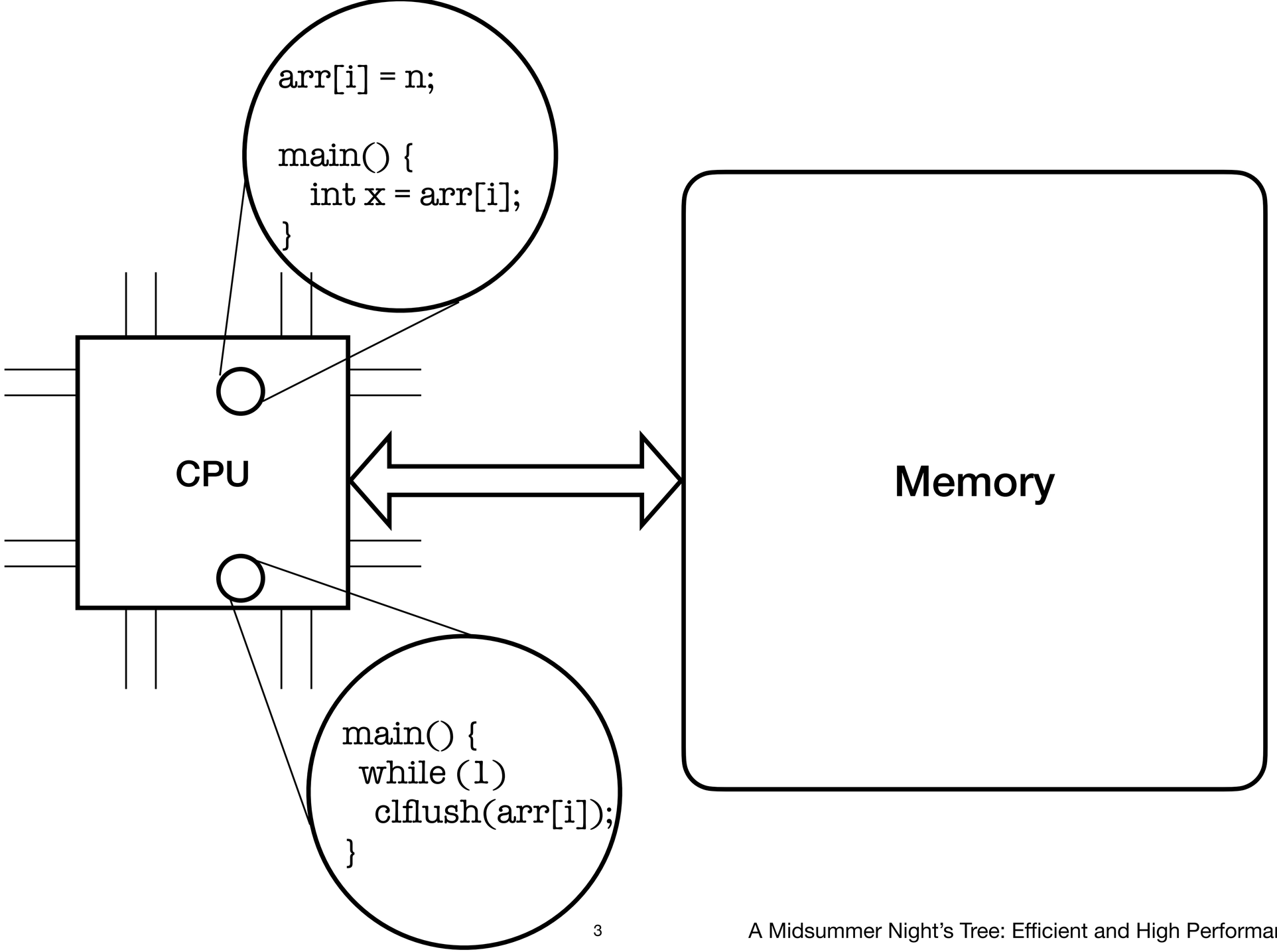


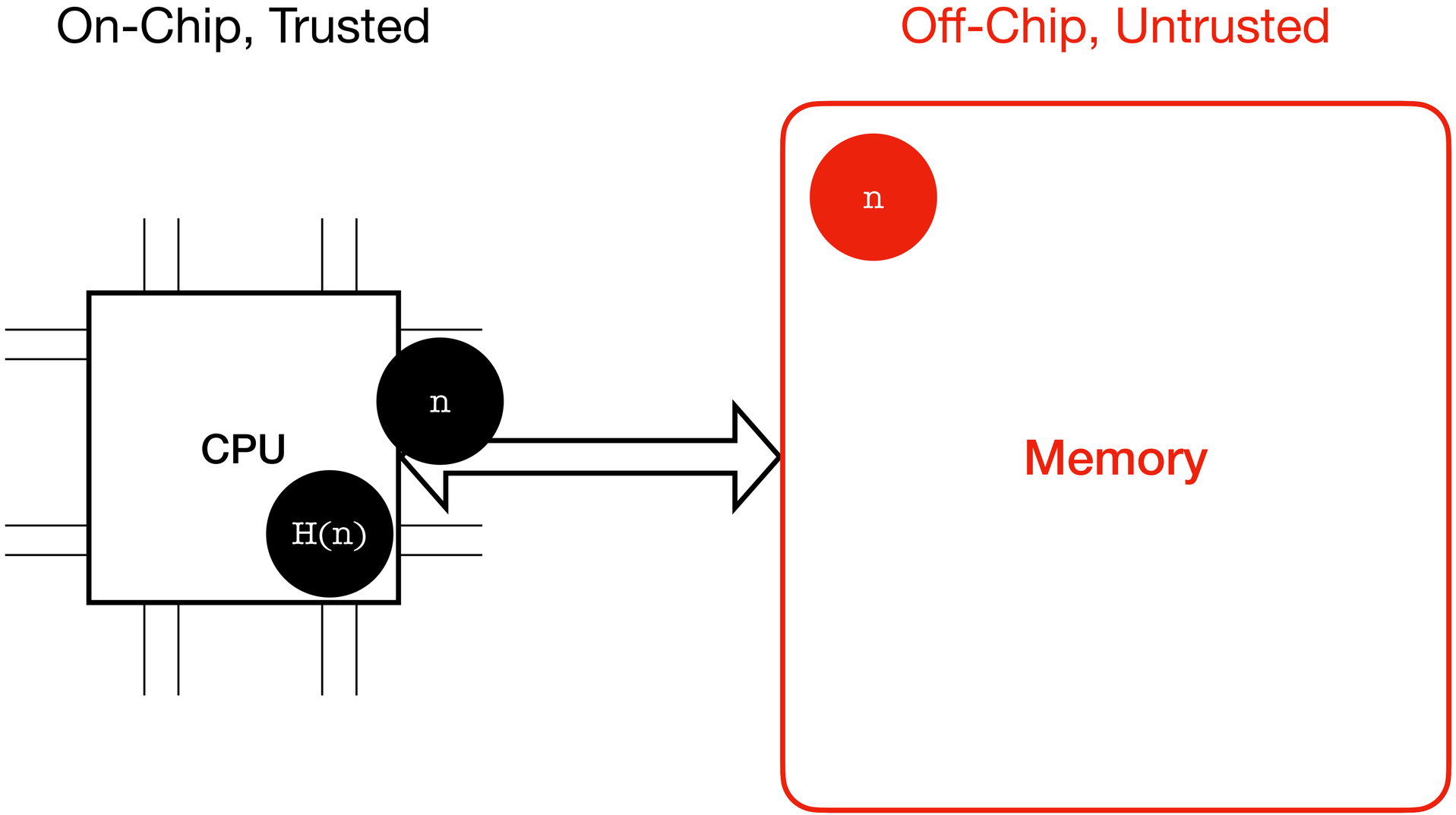
A Midsummer Night's Tree

Efficient and High Performance Secure SCM

Samuel Thomas[1], Kidus Workneh[2], Jac McCarty[3], Joseph Izraelevitz[2], Tamara Lehman[2], Iris Bahar[4]

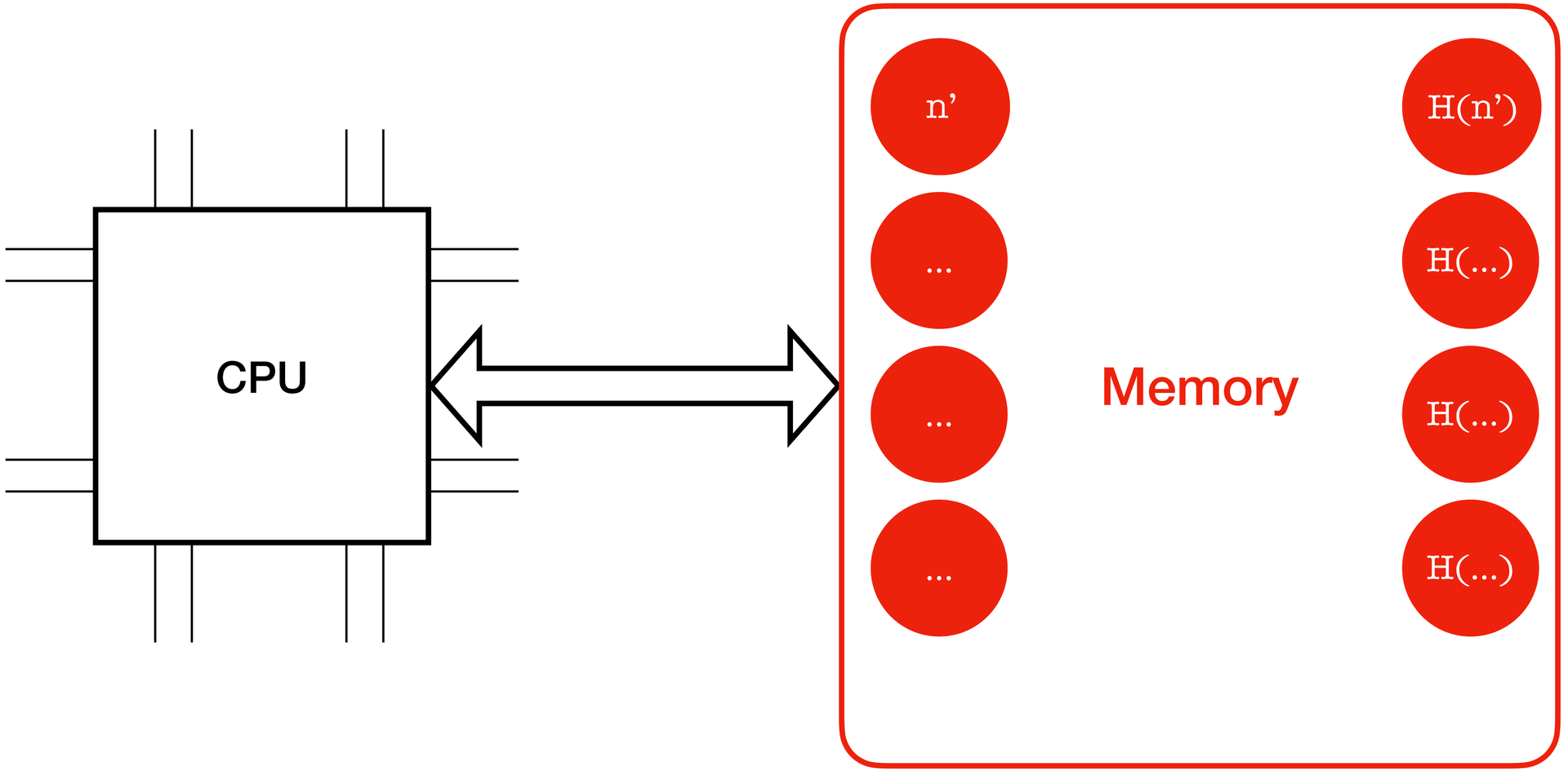
Background


arr[i] = n;
main() {
int x = arr[i];
}
CPU
Memory
main() {
while (1)
clflush(arr[i]);
}

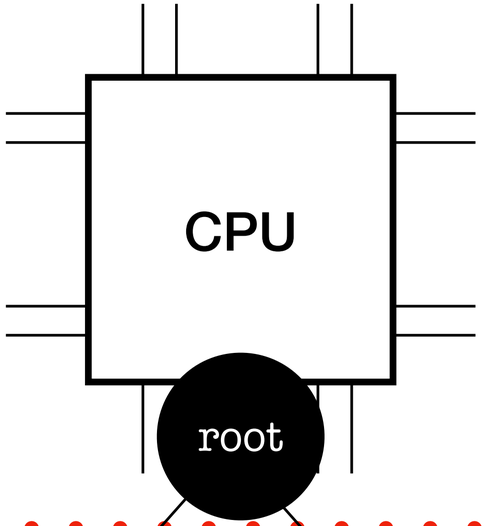

On-Chip, Trusted
Off-Chip, Untrusted
n
CPU
n
H(n)
Memory

On-Chip, Trusted

Off-Chip, Untrusted

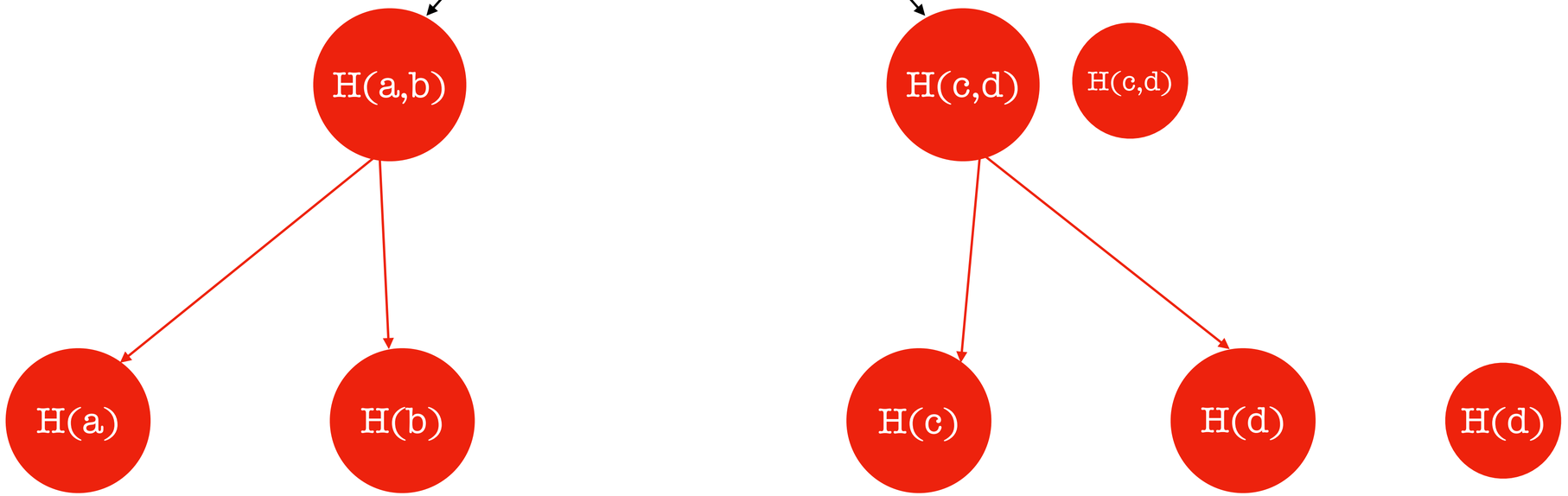


On-Chip, Trusted

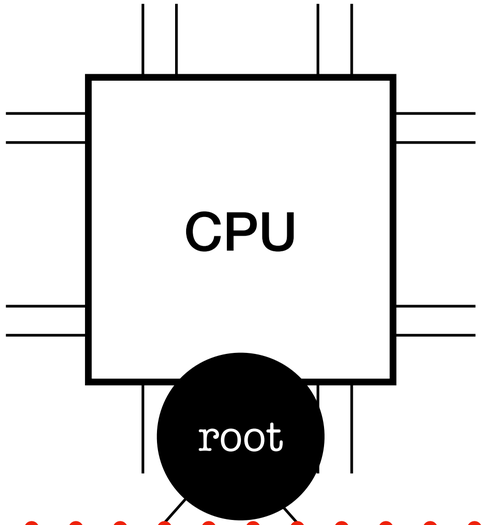


Fetch d!

Off-Chip, Untrusted

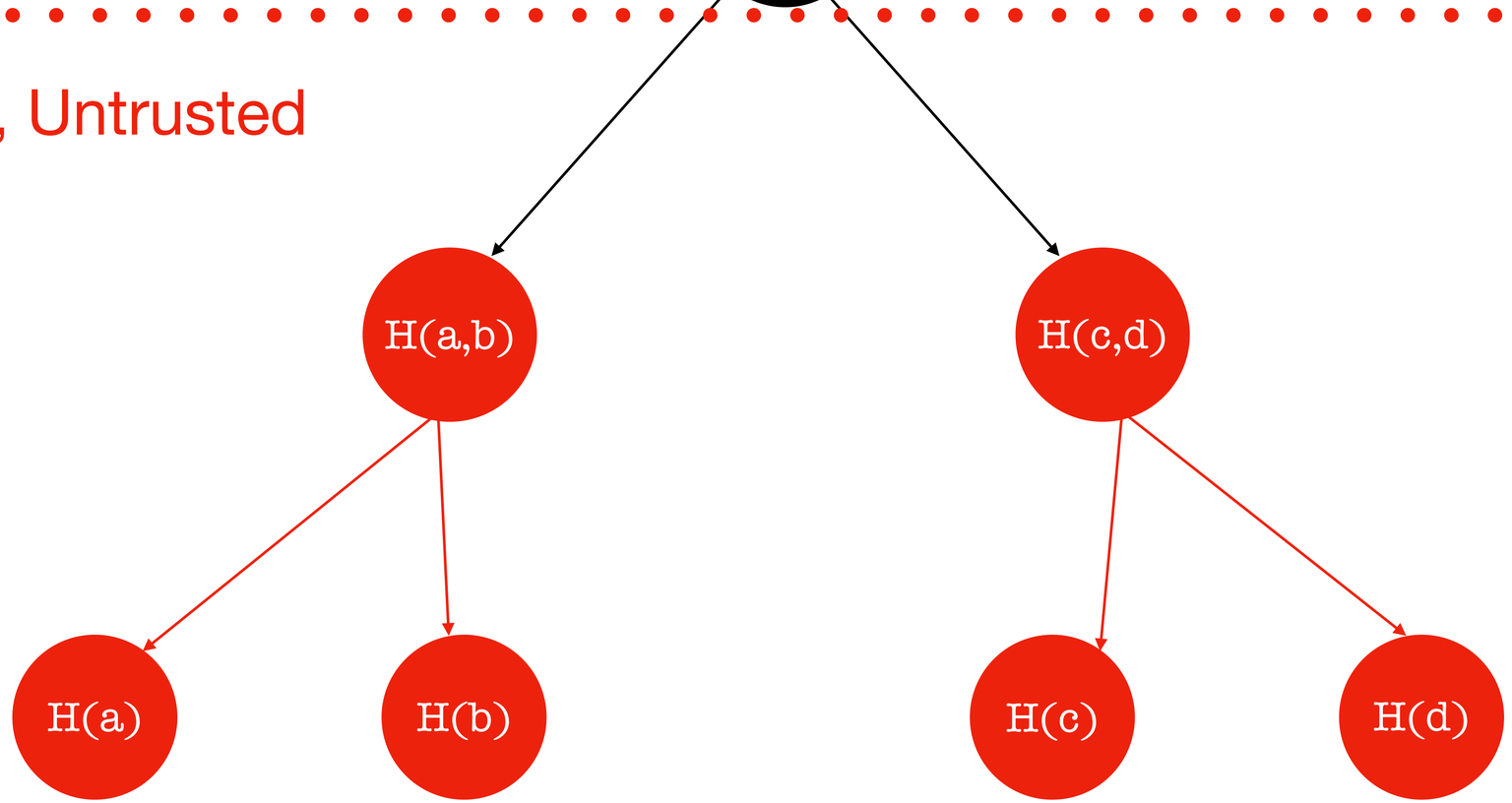


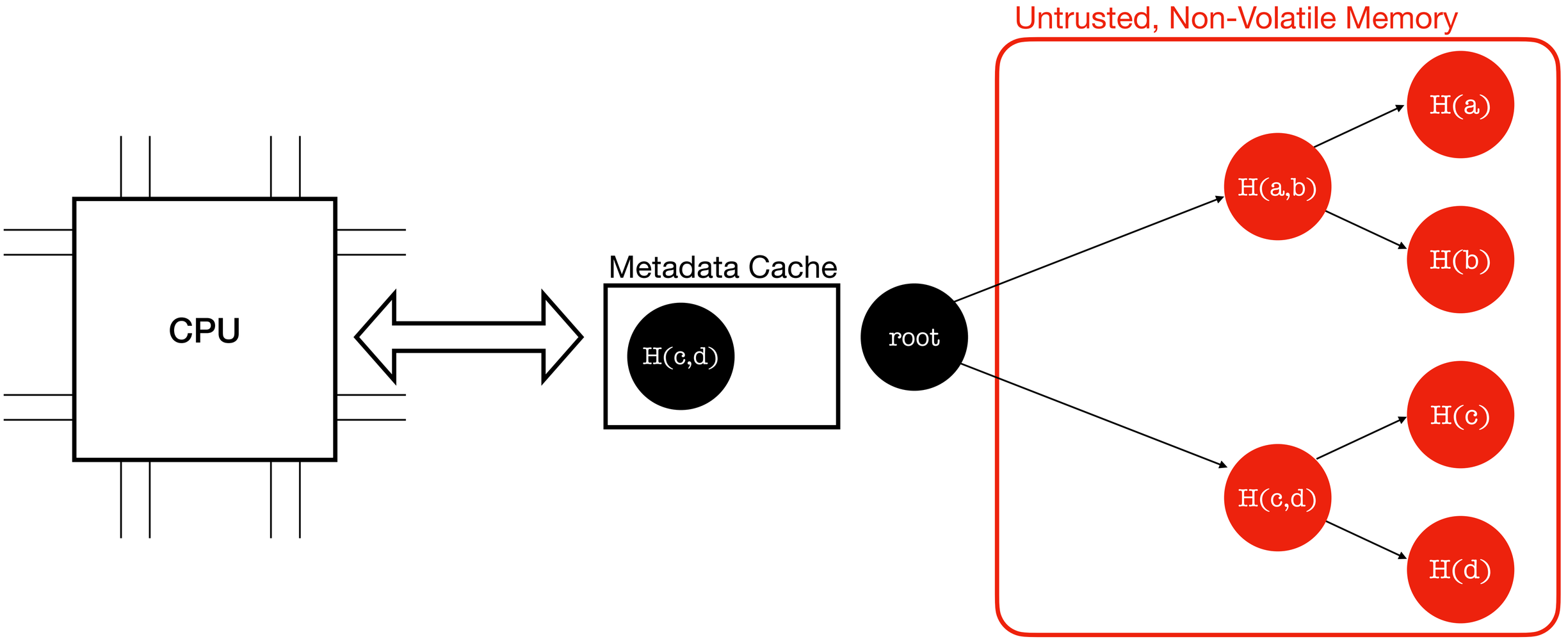
On-Chip, Trusted

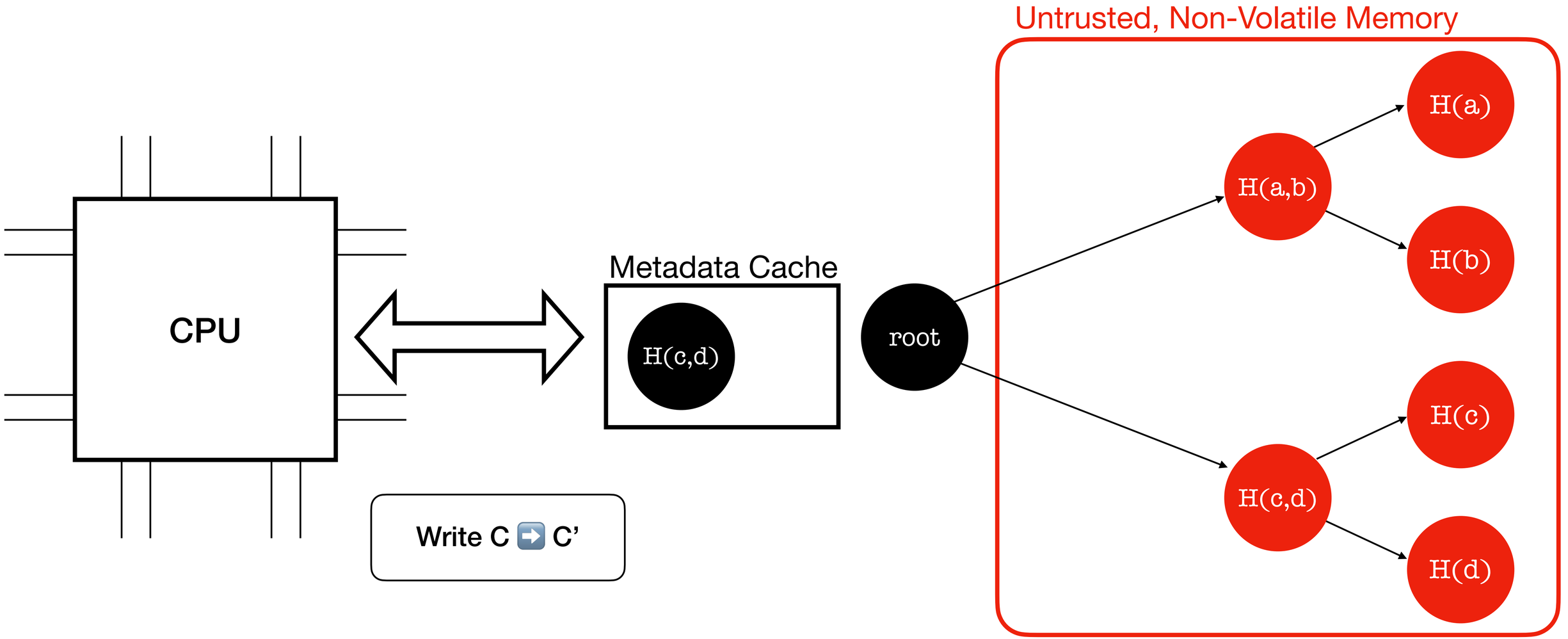


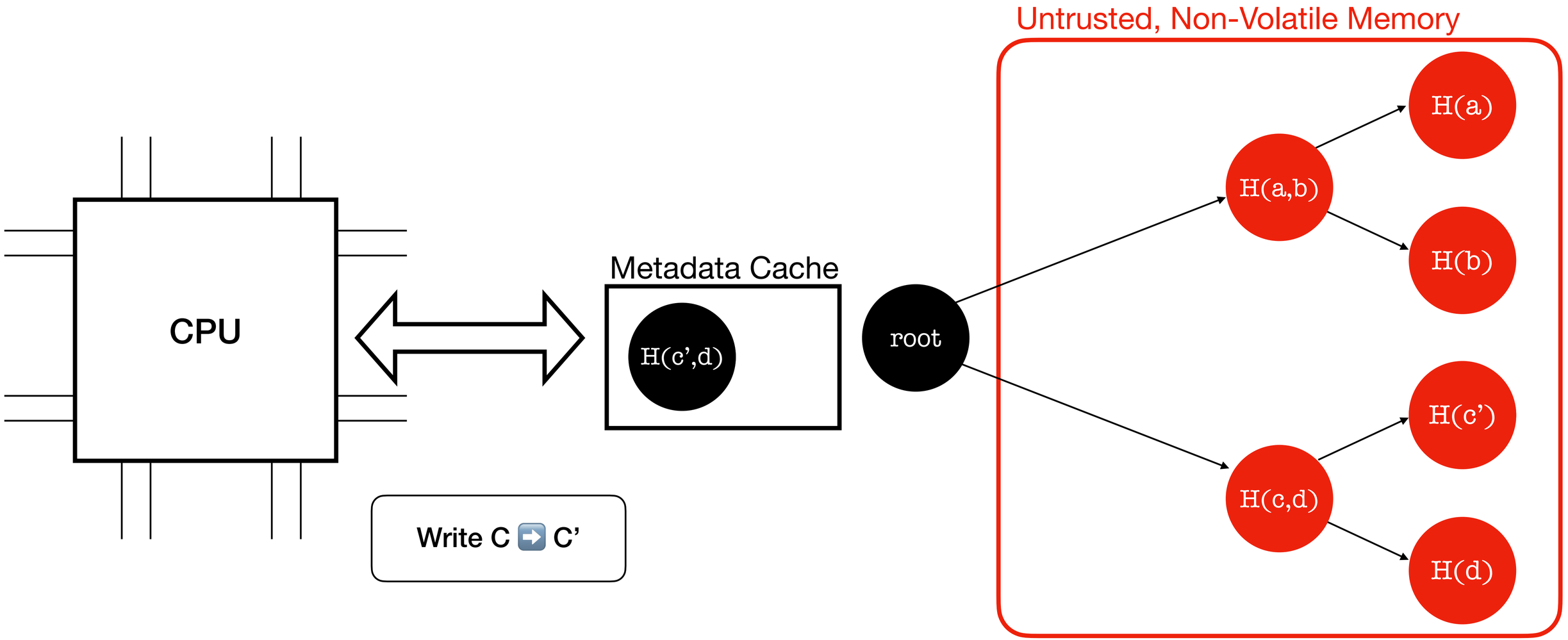
Metadata Cache

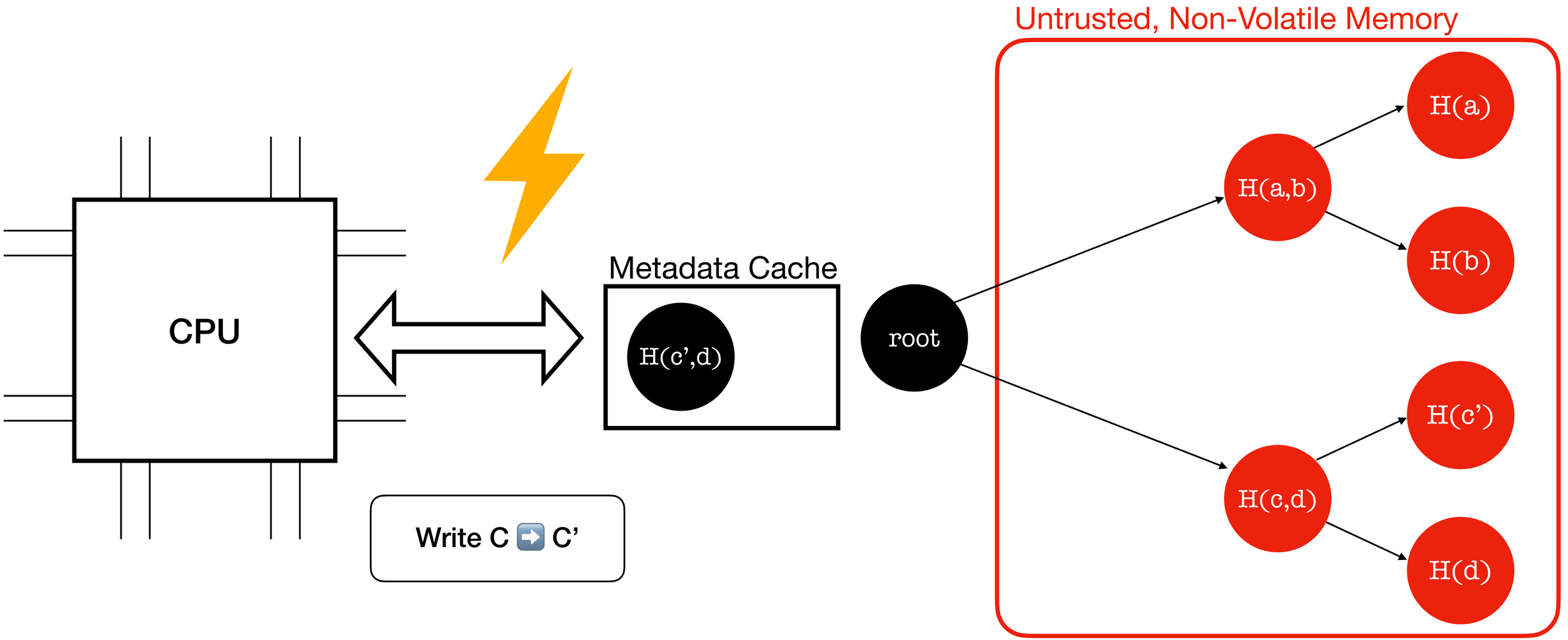
Off-Chip, Untrusted

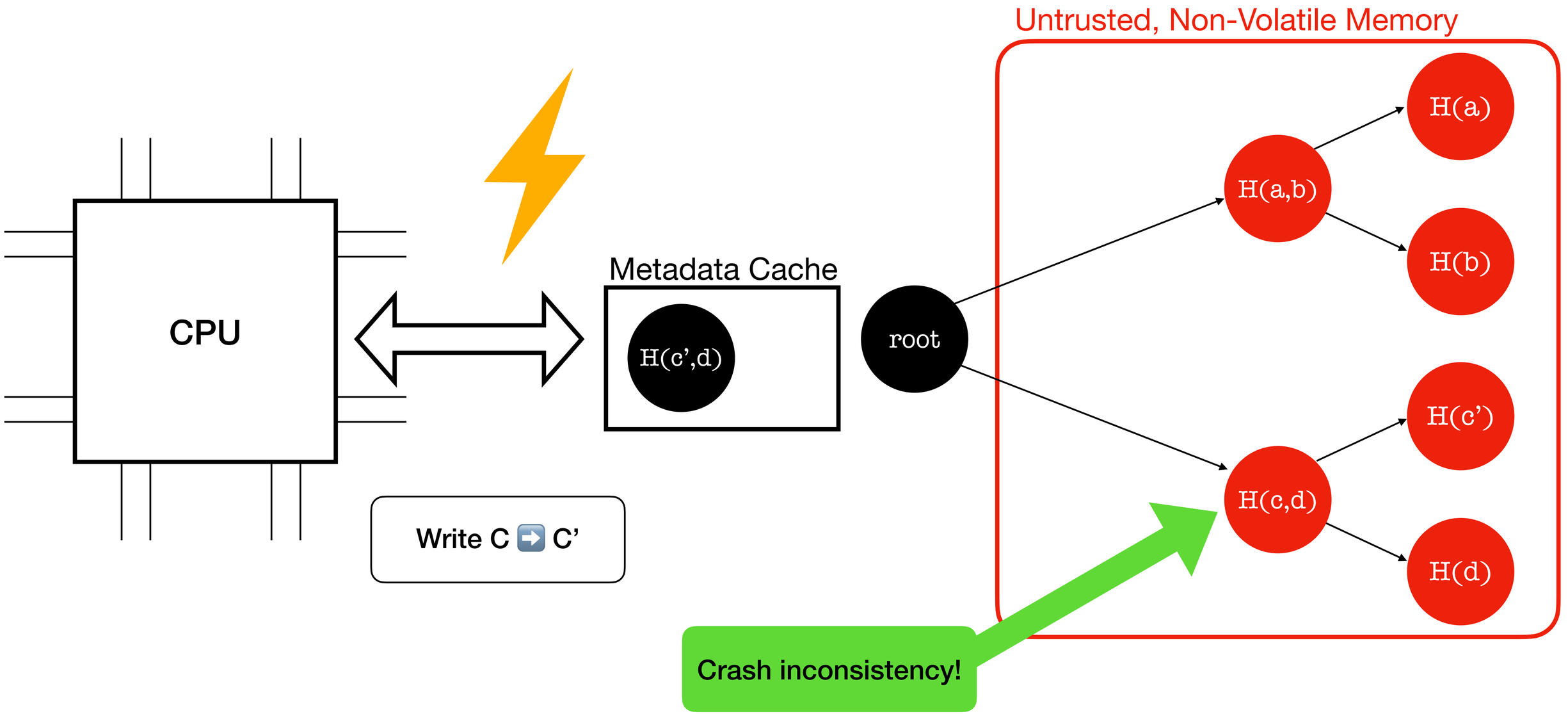


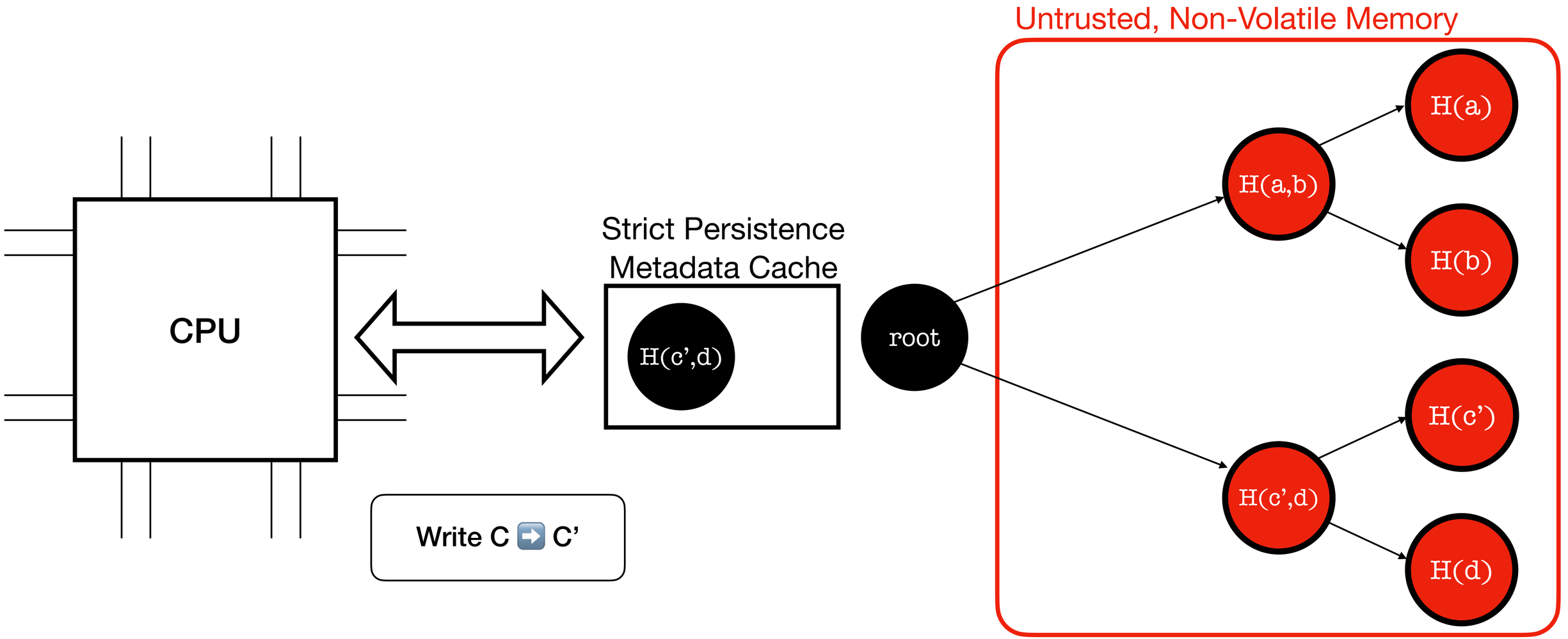

Untrusted, Non-Volatile Memory
CPU
Metadata Cache
H(c,d)
root
H(a)
H(a,b)
H(b)
H(c)
H(c,d)
H(d)

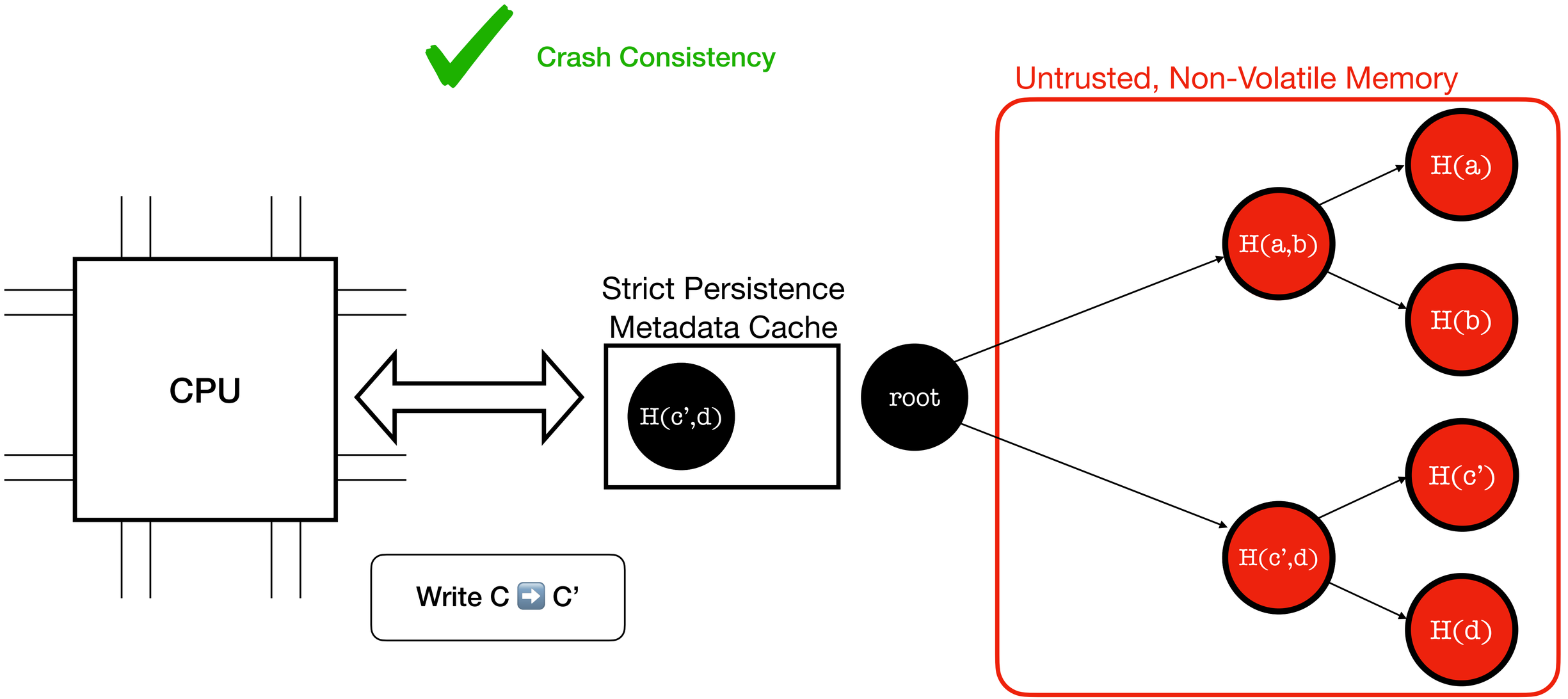

Untrusted, Non-Volatile Memory
CPU
Metadata Cache
H(c,d)
root
H(a)
H(a,b)
H(b)
H(c)
H(c,d)
H(d)
Write C ➡️ C’

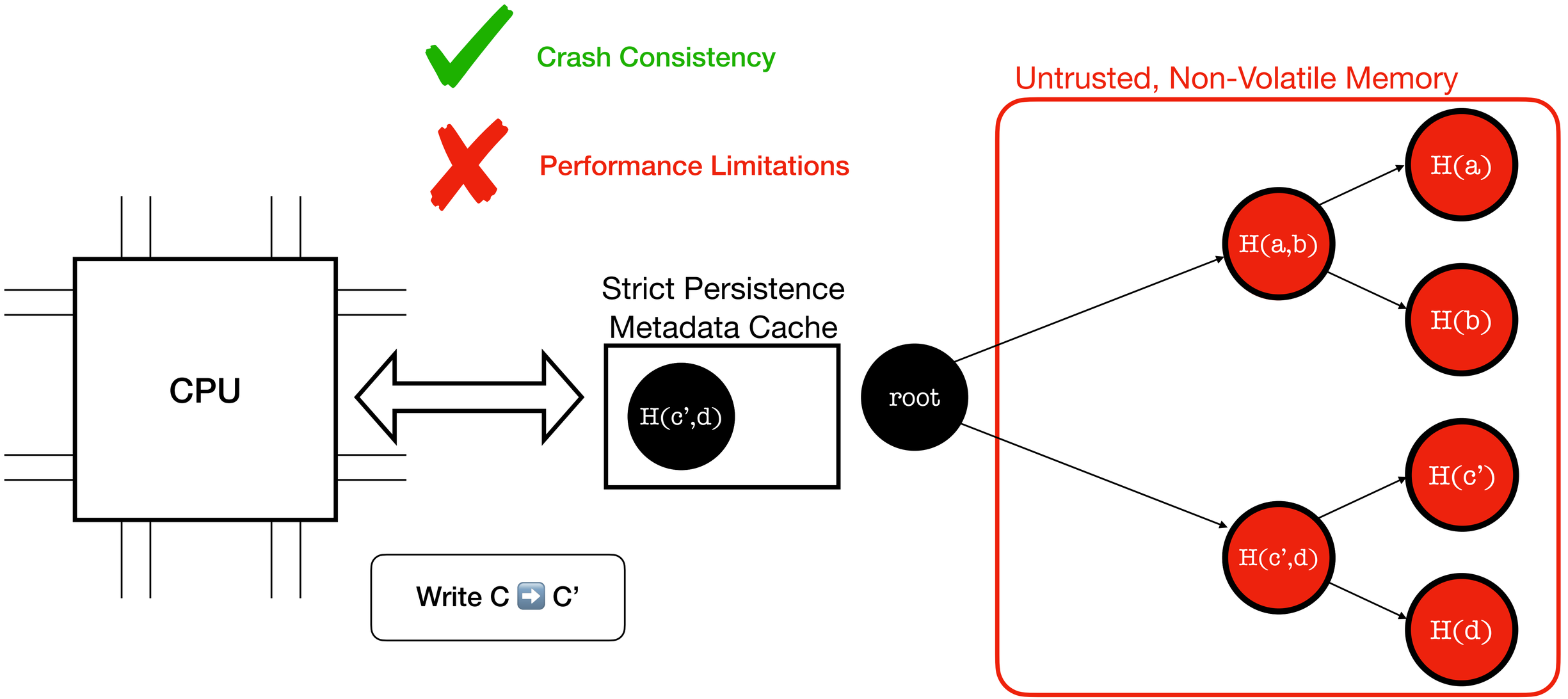

Untrusted, Non-Volatile Memory
CPU
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)
Write C ➡️ C'

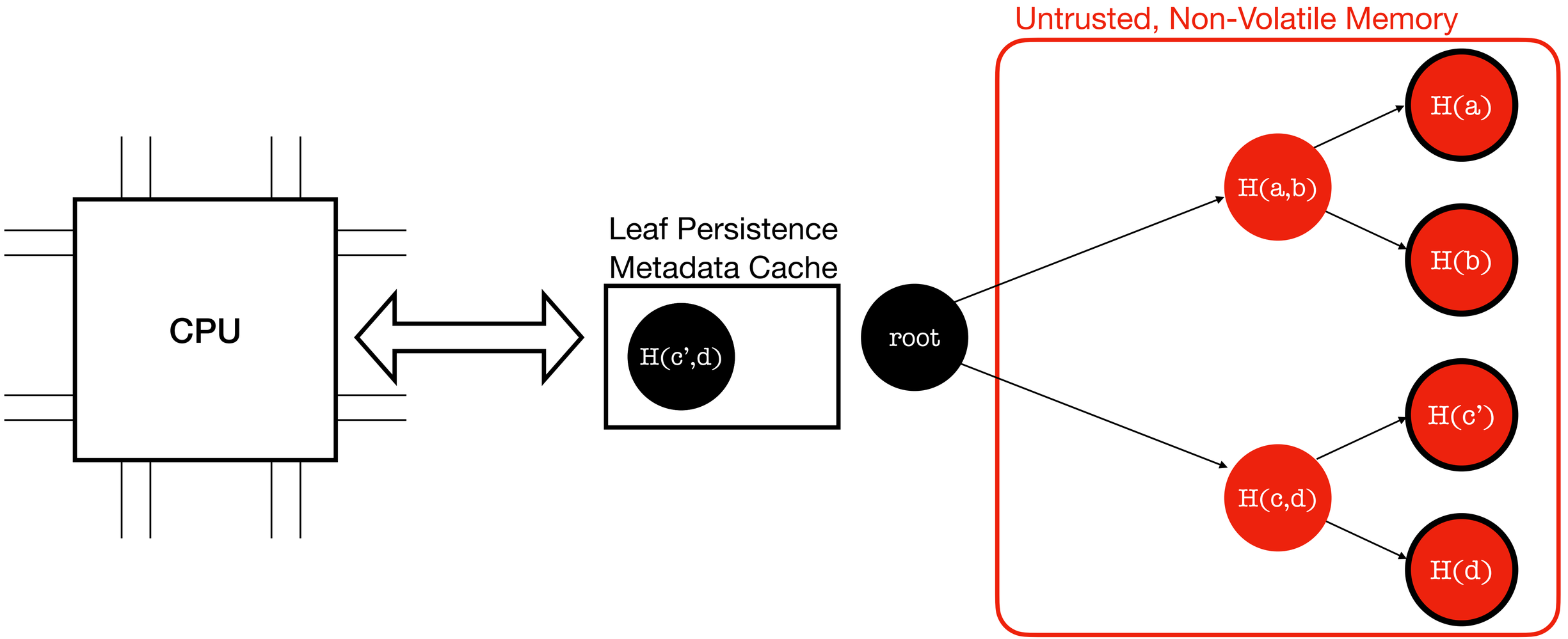

Untrusted, Non-Volatile Memory
CPU
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)
Write C ➡️ C'

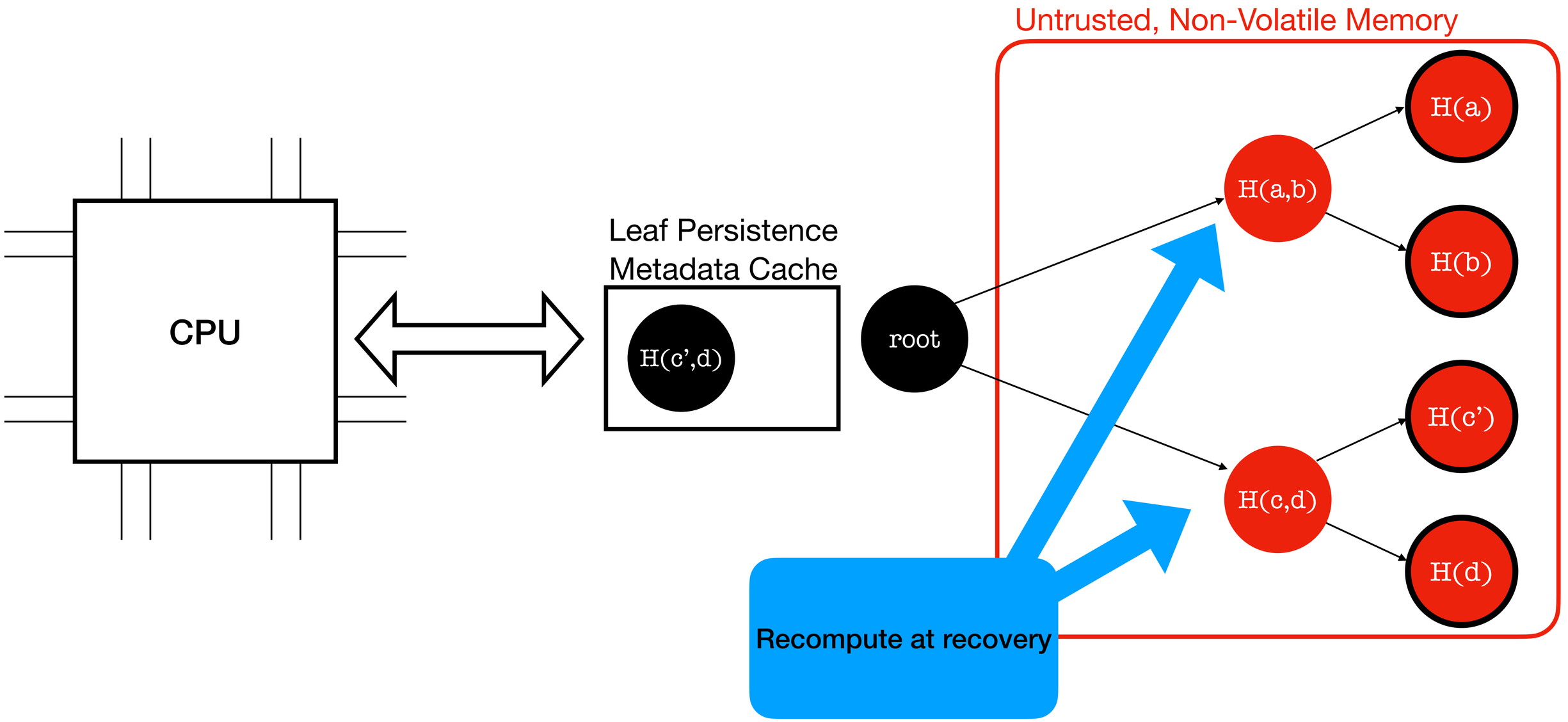

Untrusted, Non-Volatile Memory
CPU
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)
Write C ➡️ C'
Crash inconsistency!

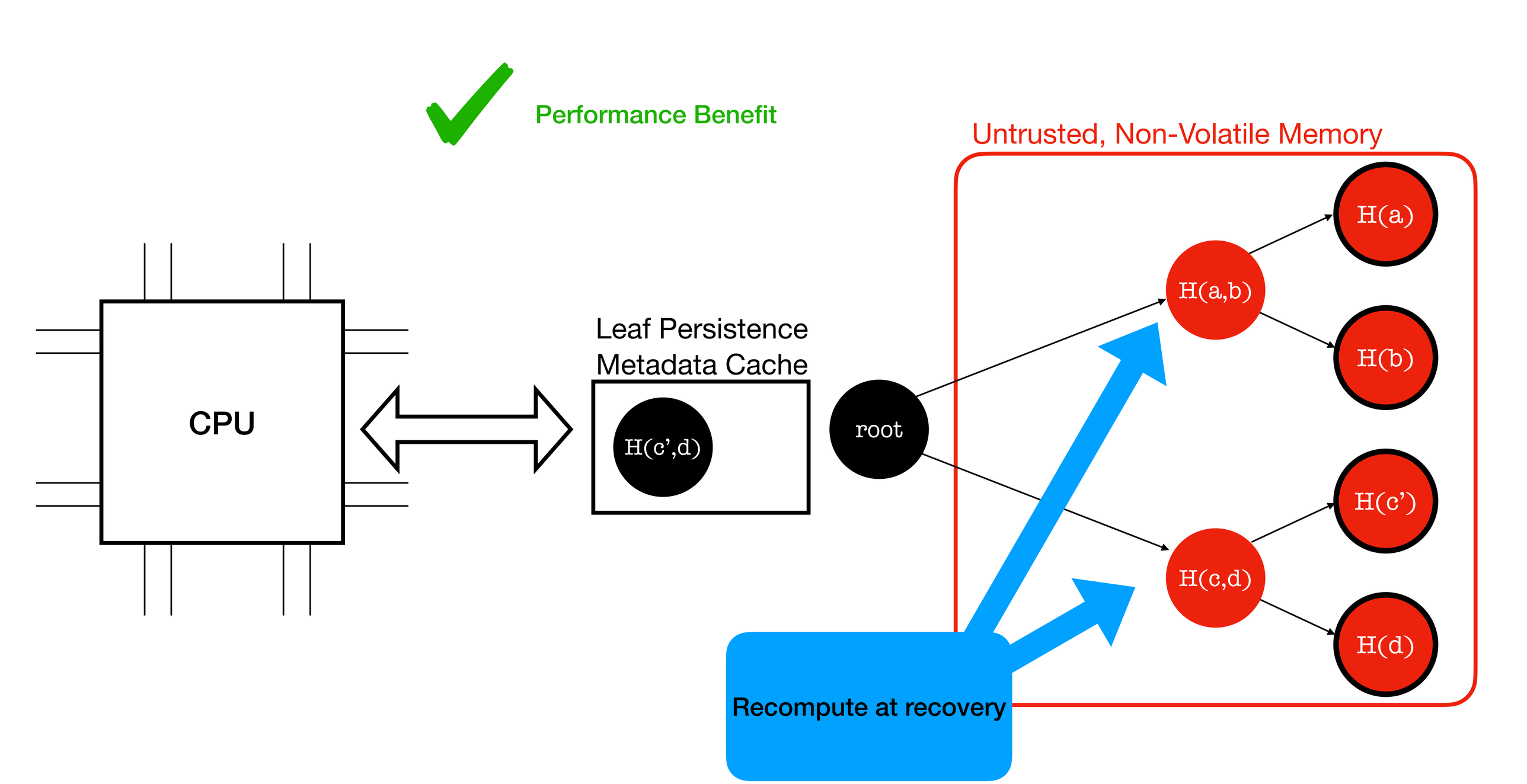

Untrusted, Non-Volatile Memory
CPU
Strict Persistence
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c',d)
H(d)
Write C ➡️ C'

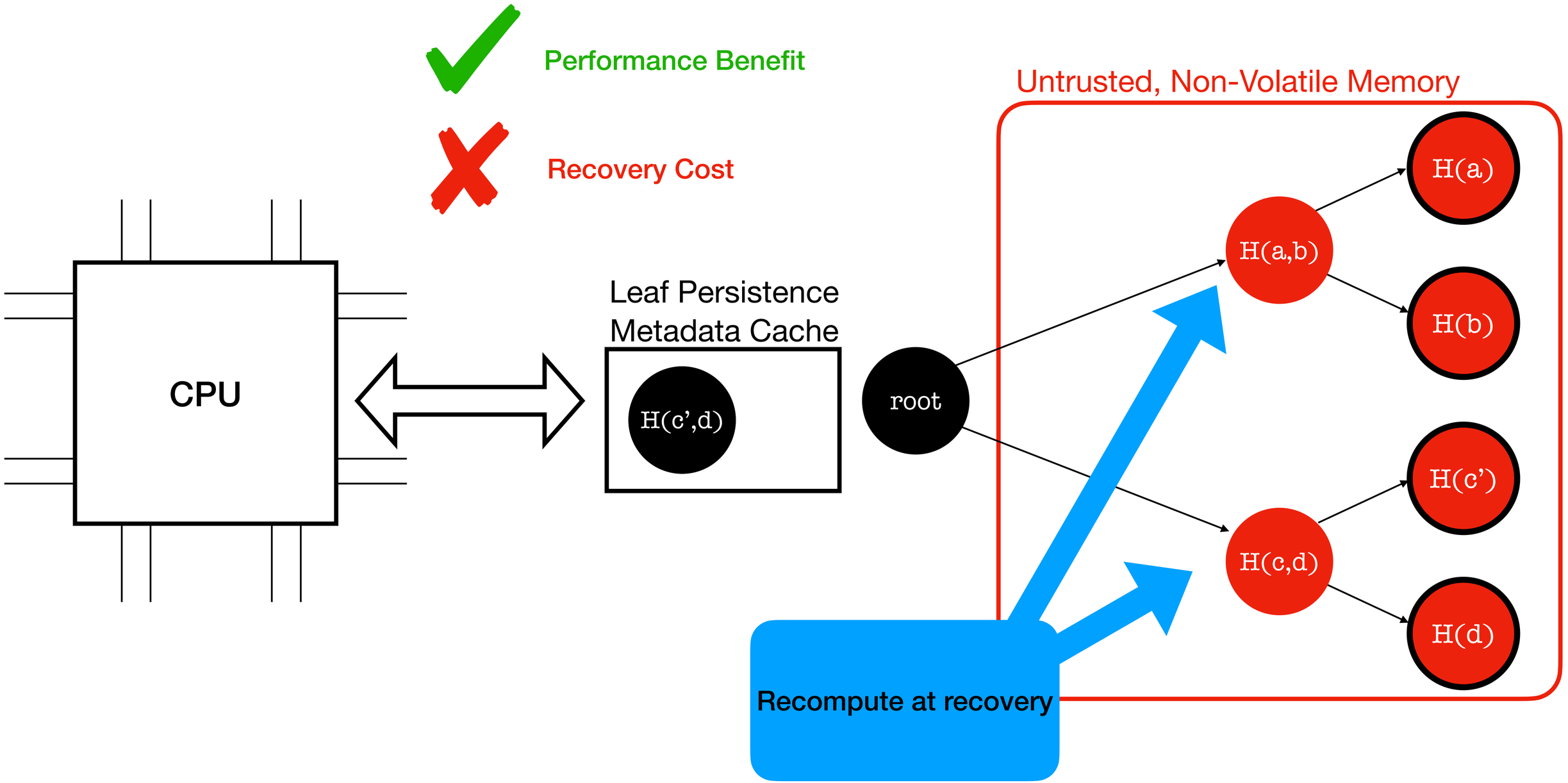

Crash Consistency
Untrusted, Non-Volatile Memory
CPU
Strict Persistence
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c',d)
H(d)
Write C ➡️ C'


Crash Consistency
Performance Limitations
CPU
Strict Persistence
Metadata Cache
H(c',d)
Write C ➡️ C'
Untrusted, Non-Volatile Memory
root
H(a)
H(a,b)
H(b)
H(c')
H(c',d)
H(d)


Untrusted, Non-Volatile Memory
CPU
Leaf Persistence
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)


Untrusted, Non-Volatile Memory
CPU
Leaf Persistence
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)
Recompute at recovery


Performance Benefit
Untrusted, Non-Volatile Memory
CPU
Leaf Persistence
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)
Recompute at recovery


Performance Benefit
Recovery Cost
Untrusted, Non-Volatile Memory
CPU
Leaf Persistence
Metadata Cache
H(c',d)
root
H(a)
H(a,b)
H(b)
H(c')
H(c,d)
H(d)
Recompute at recovery

Contribution

Samuel Thomas

Kidus Workneh, Joseph Izraelevitz, Tamara Lehman

Jac McCarty

R. Iris Bahar

QR code to the paper!

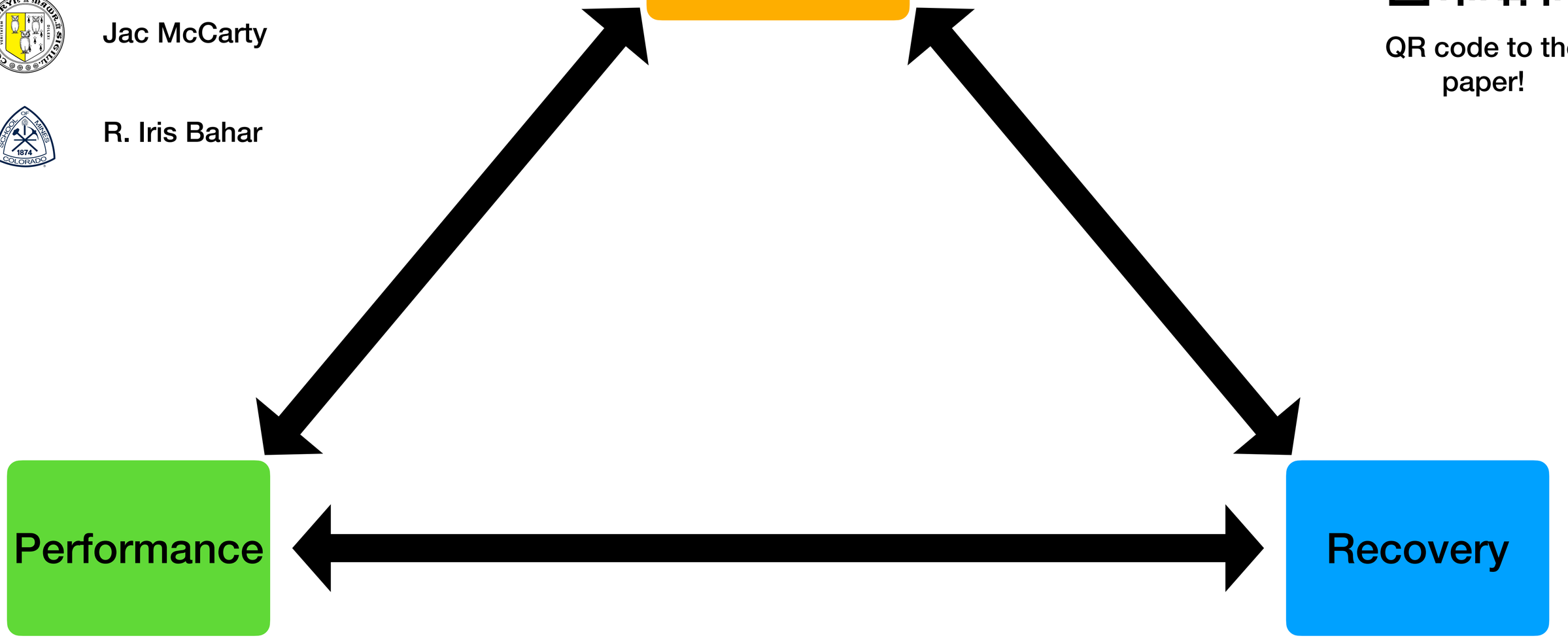
Samuel Thomas

Kidus Workneh, Joseph Izraelevitz, Tamara Lehman

Jac McCarty

R. Iris Bahar

QR code to the paper!



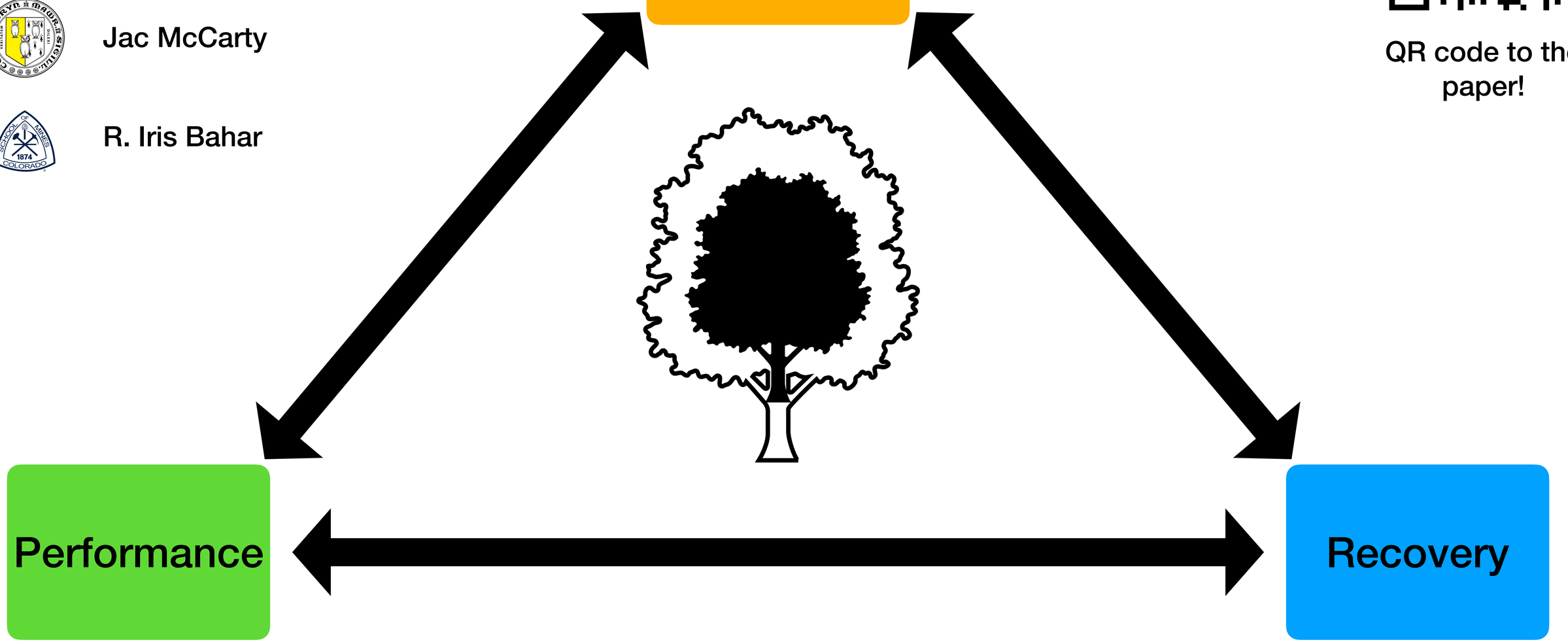
Samuel Thomas

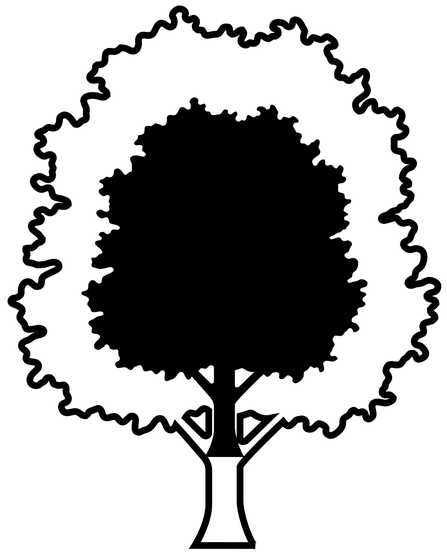
Kidus Workneh, Joseph Izraelevitz, Tamara Lehman

Jac McCarty

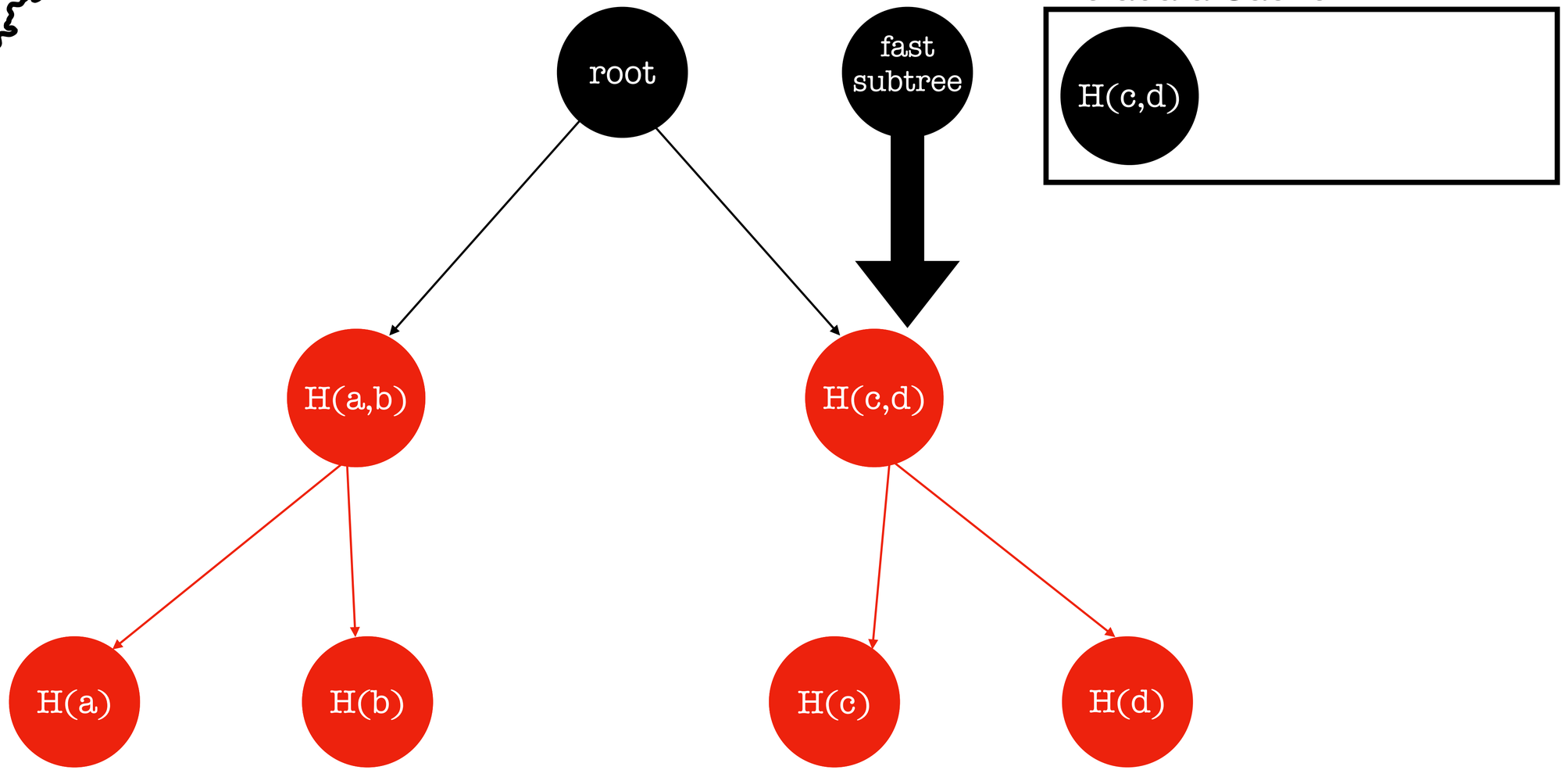
R. Iris Bahar

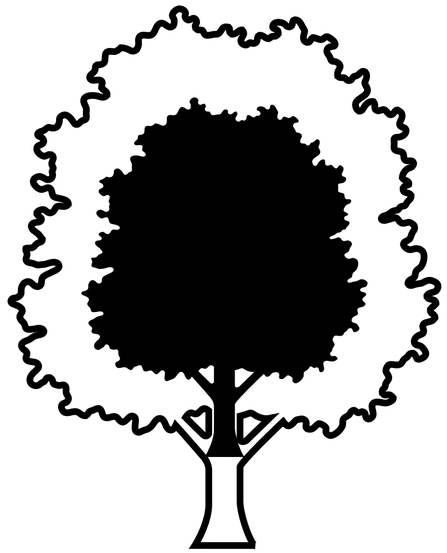
QR code to the paper!



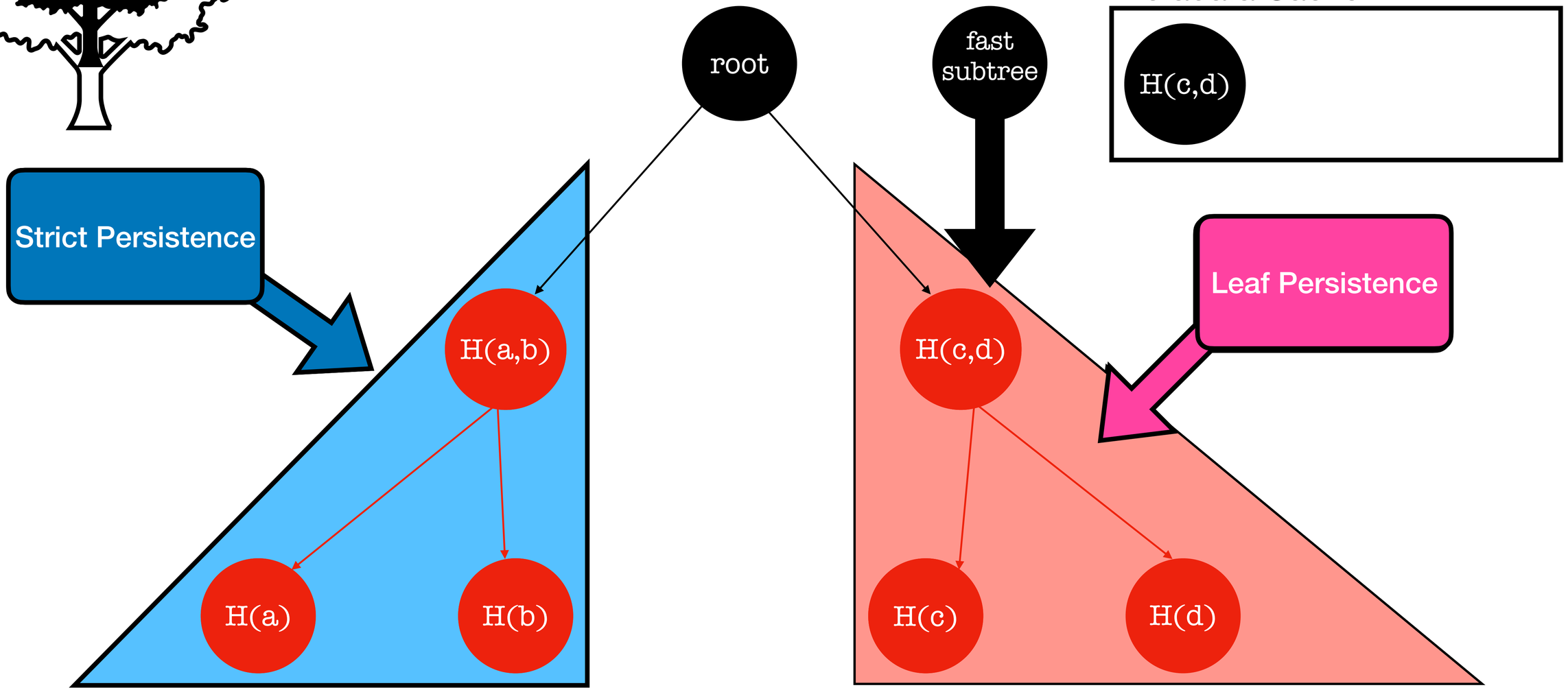


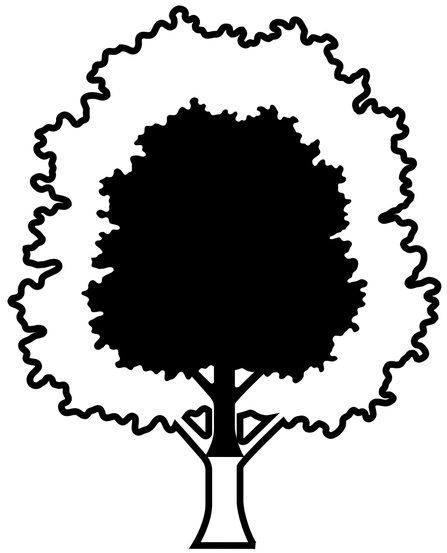
A Midsummer Night's Tree



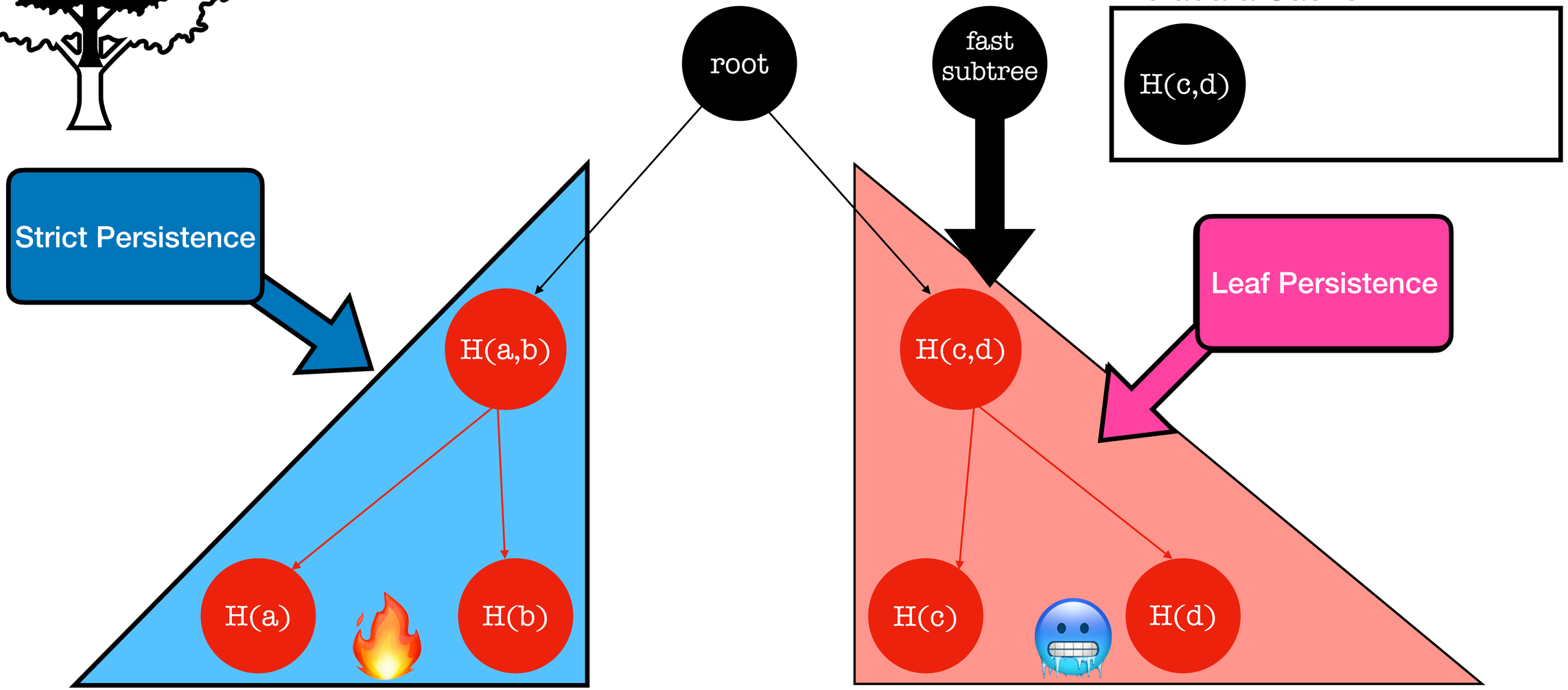


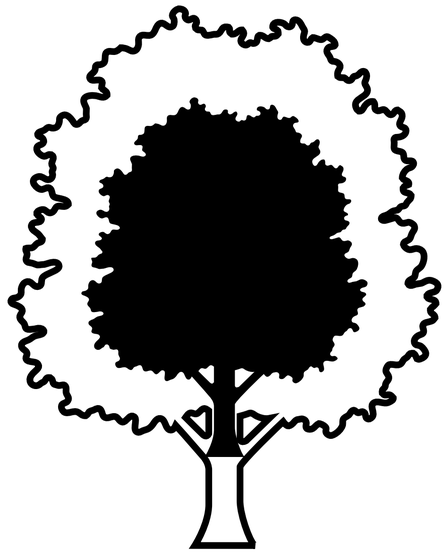
A Midsummer Night's Tree



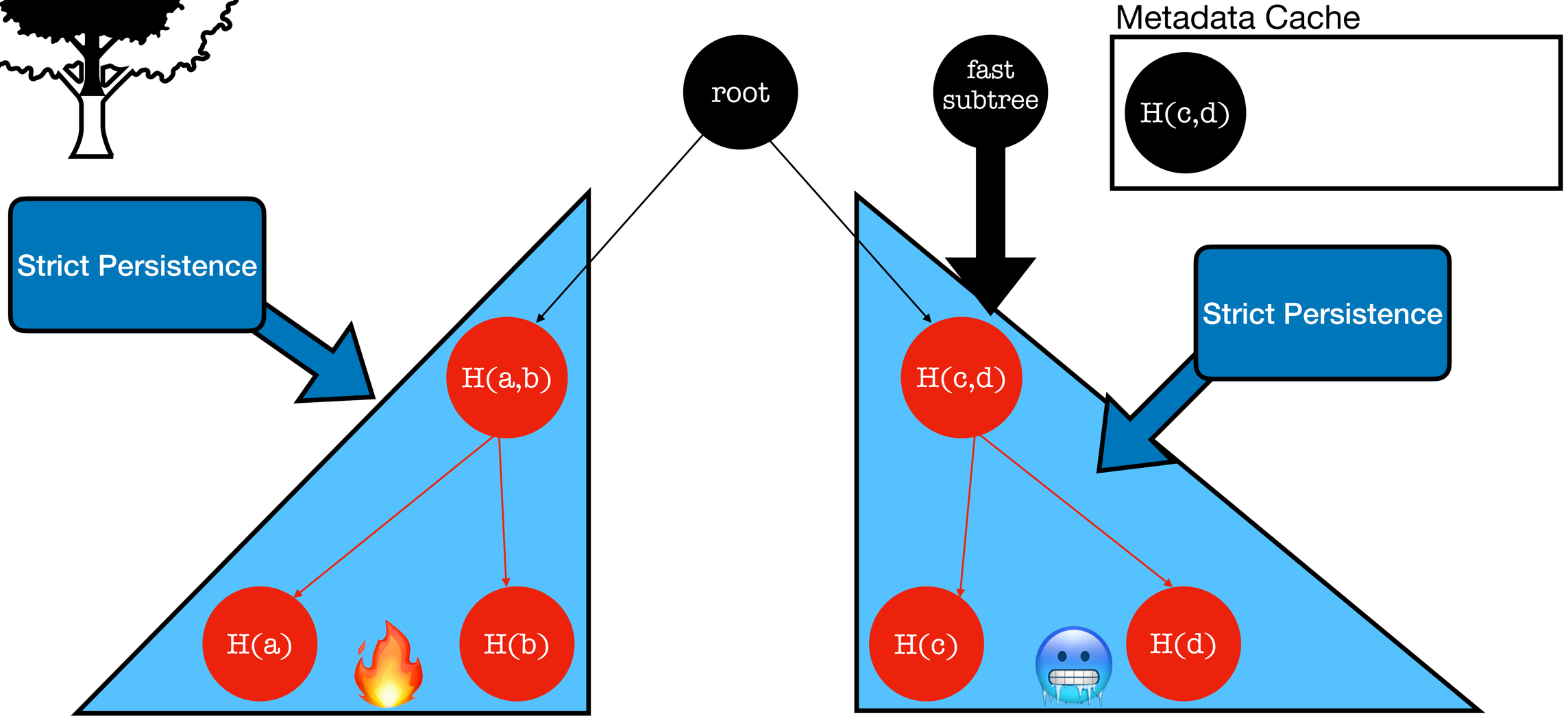


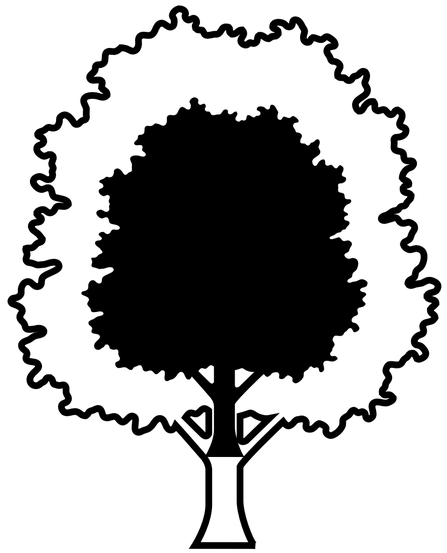
A Midsummer Night's Tree



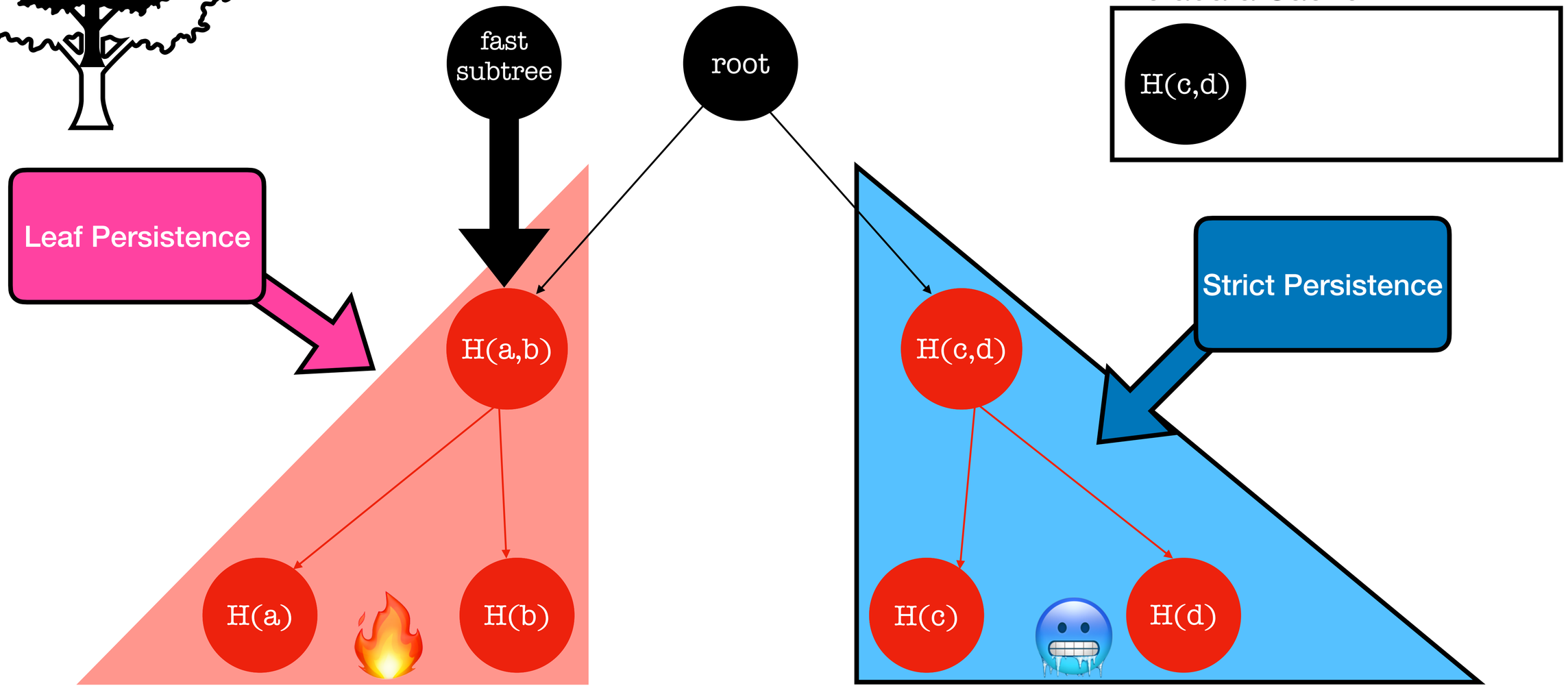


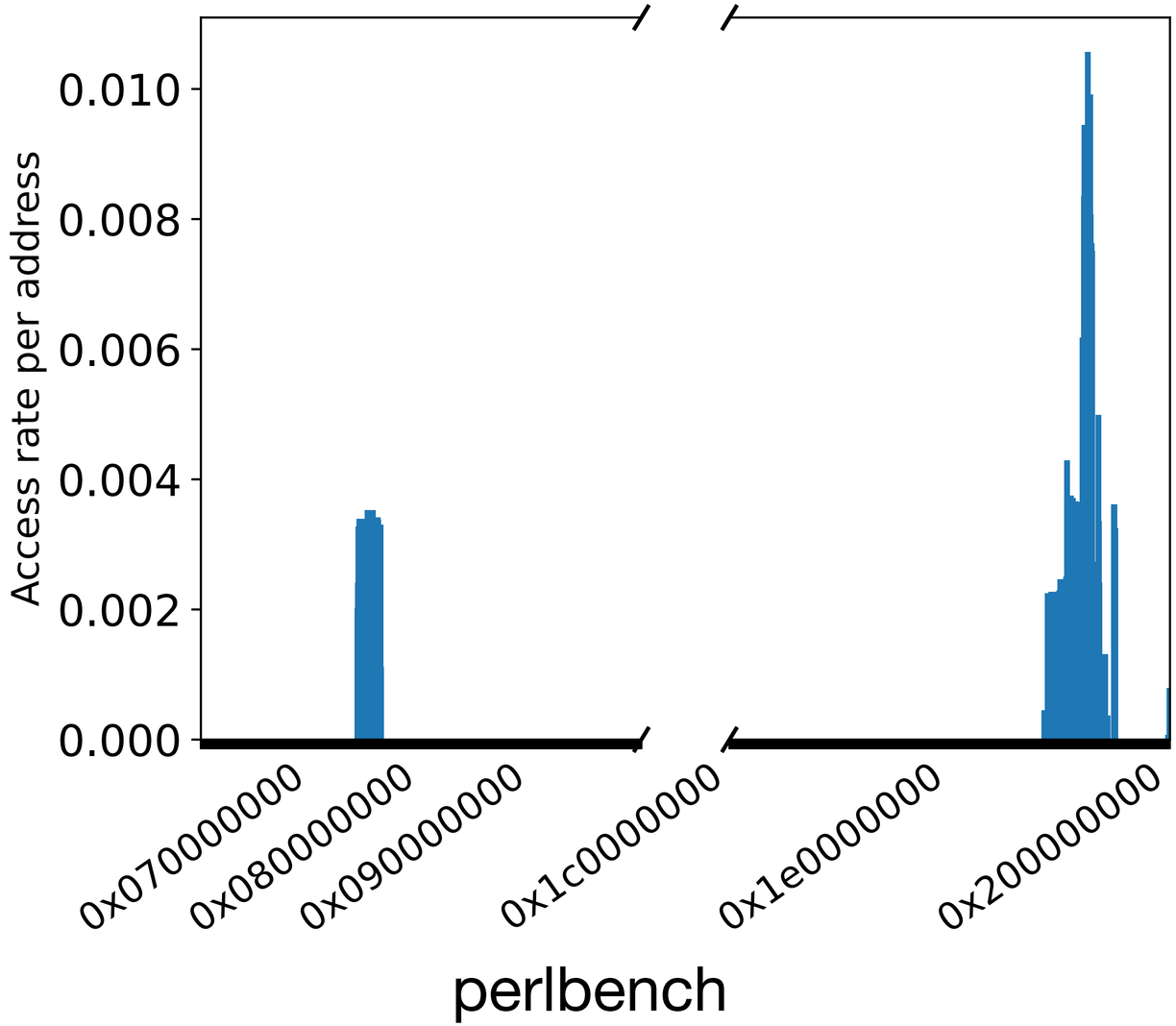
A Midsummer Night's Tree

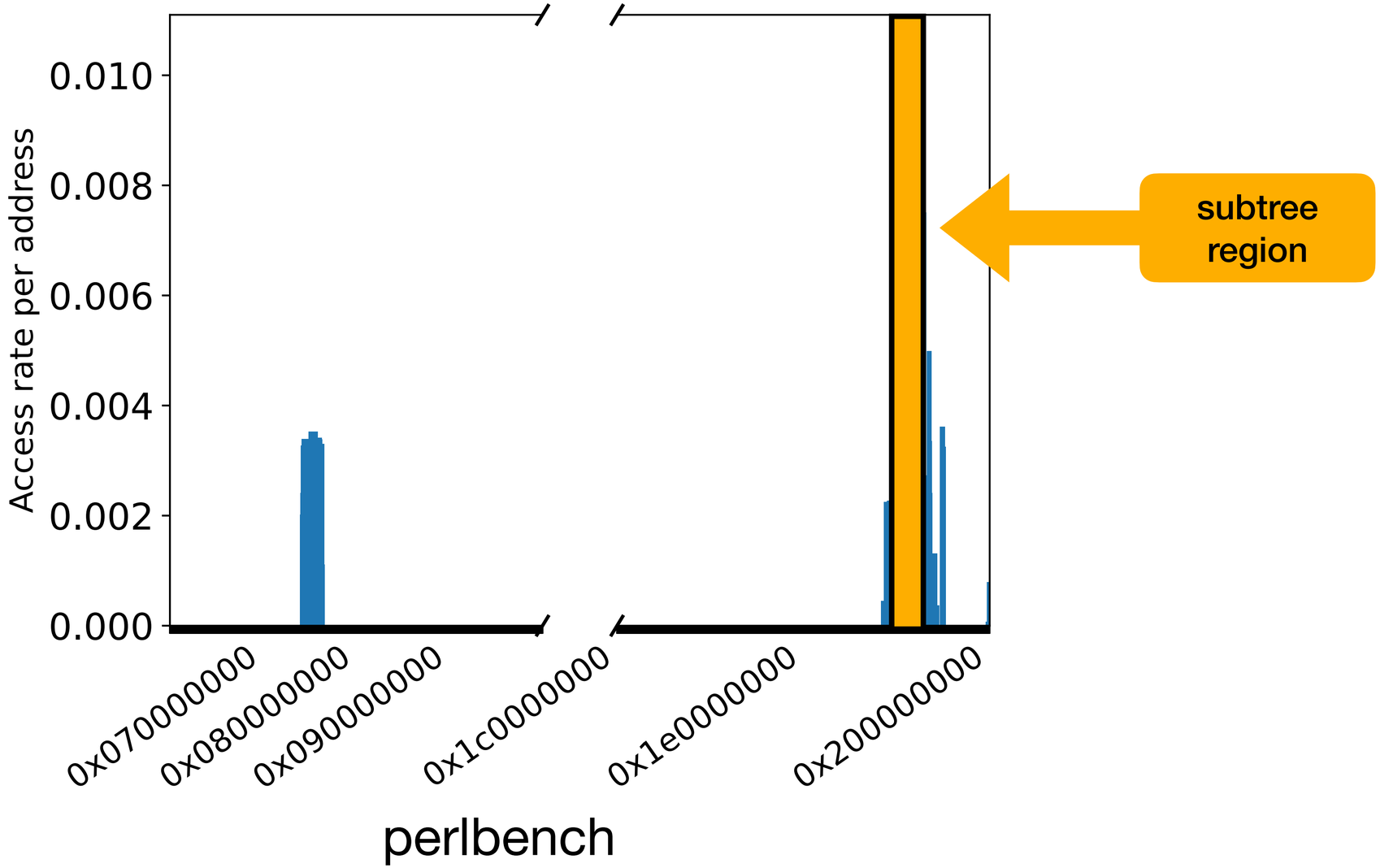


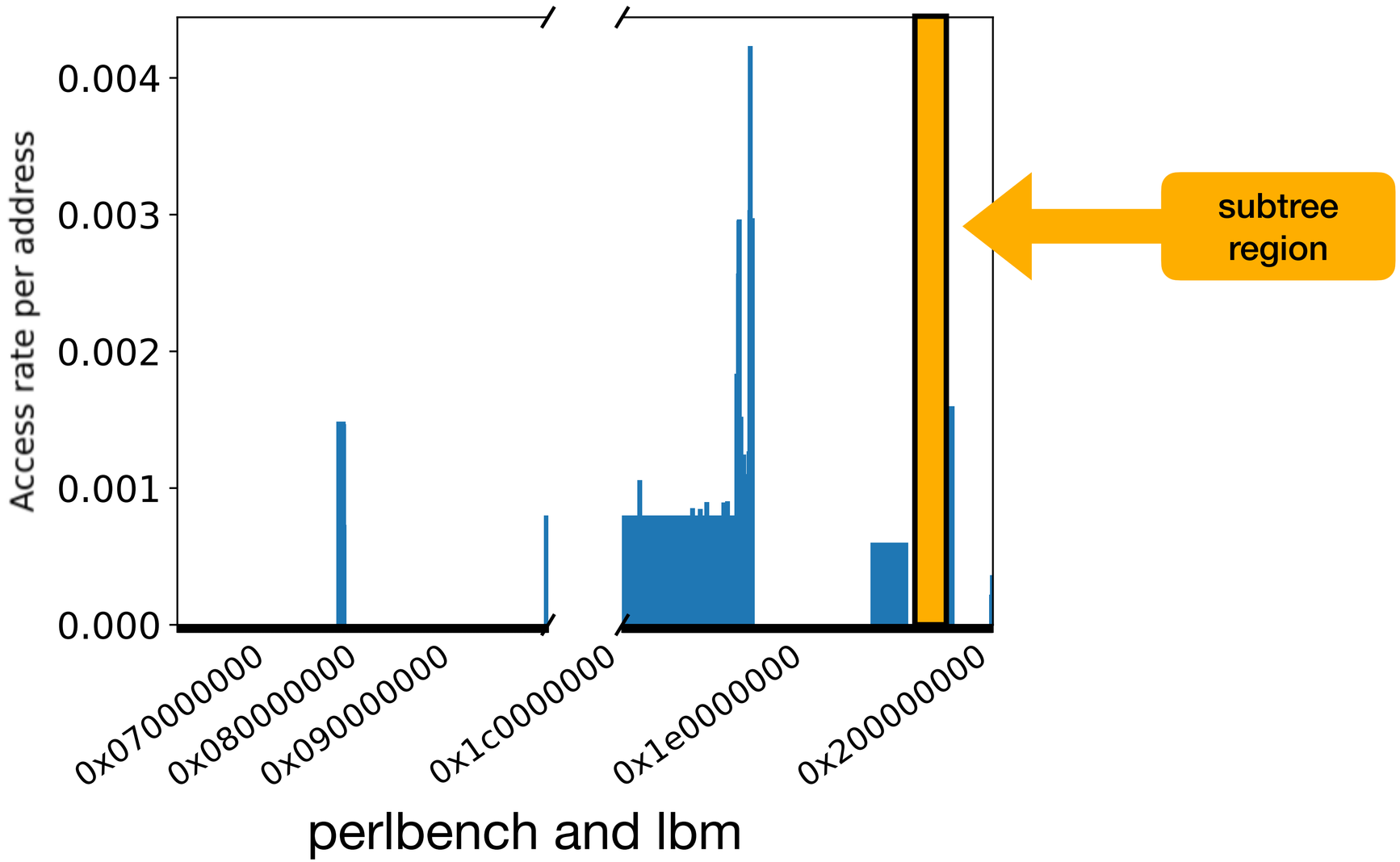


A Midsummer Night's Tree

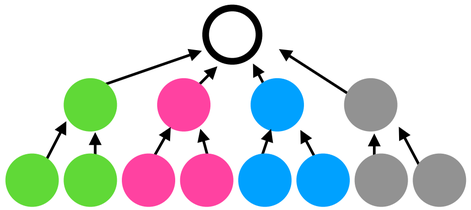



Access rate per address
0.010
0.008
0.006
0.004
0.002
0.000
0x070000000
0x080000000
0x090000000
0x1c0000000
0x1e0000000
0x200000000
perlbench

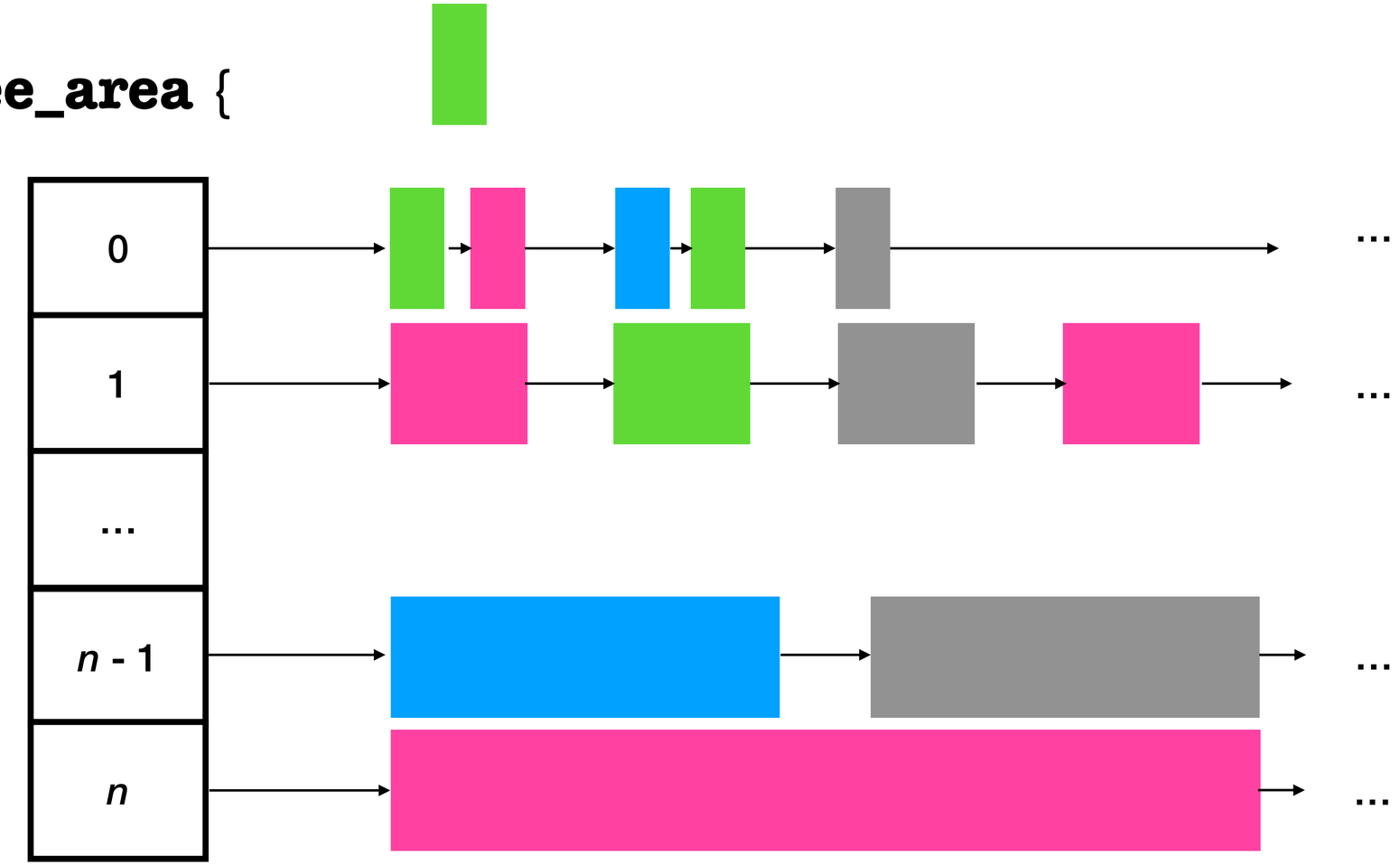

Access rate per address
0.010
0.008
0.006
0.004
0.002
0.000
0x070000000
0x080000000
0x090000000
0x1c0000000
0x1e0000000
0x200000000
perlbench
subtree region


Access rate per address
0.004
0.003
0.002
0.001
0.000
0x070000000
0x080000000
0x090000000
0x1c0000000
0x1e0000000
0x200000000
perlbench and lbm
subtree region

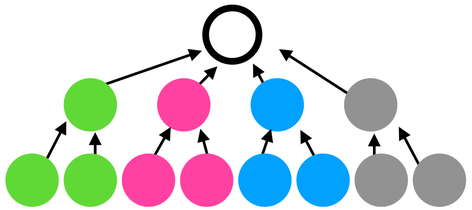
Biased Physical Page Allocation



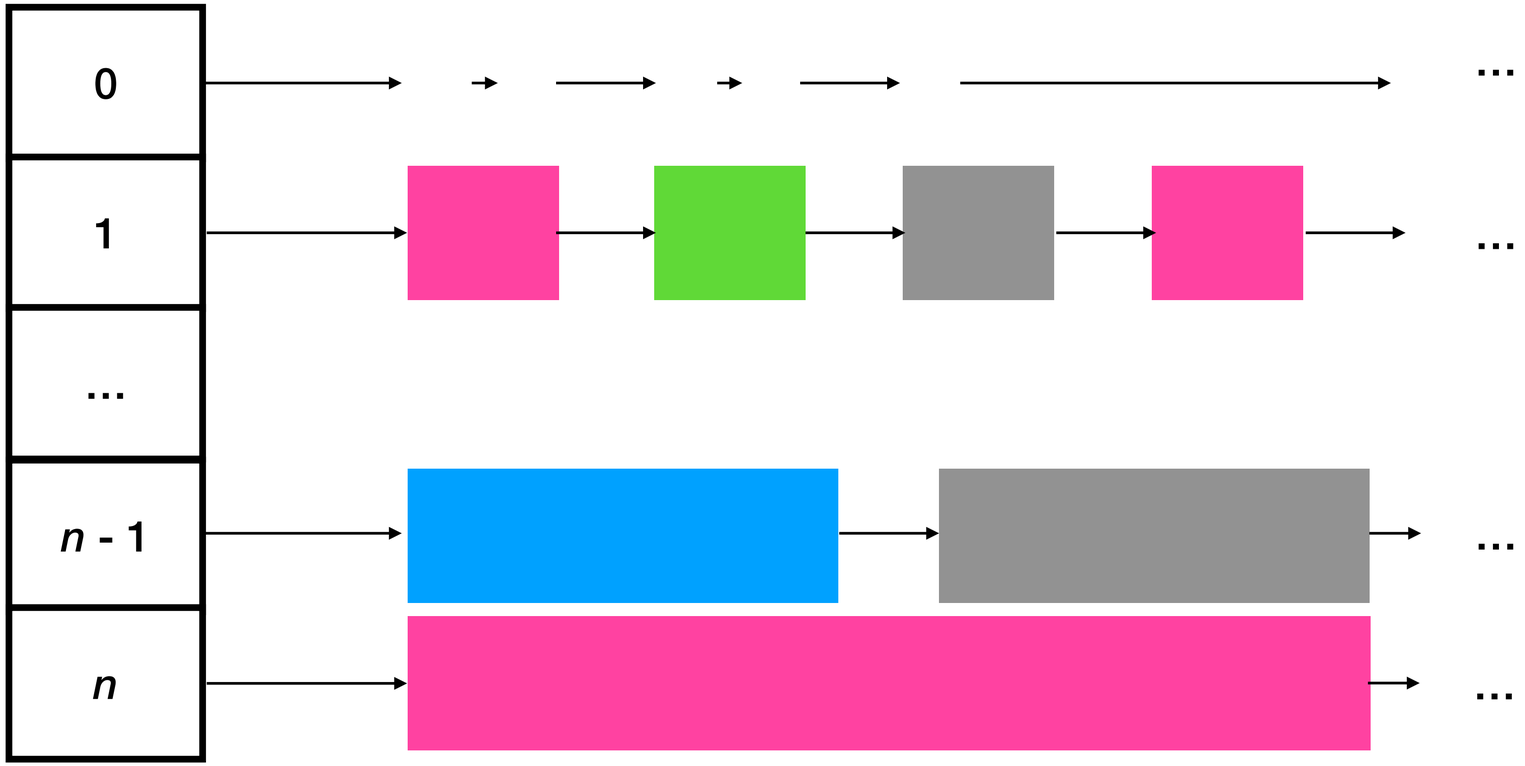
```
struct free_area {
    ...
};
```



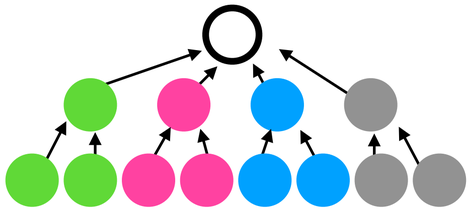
Biased Physical Page Allocation



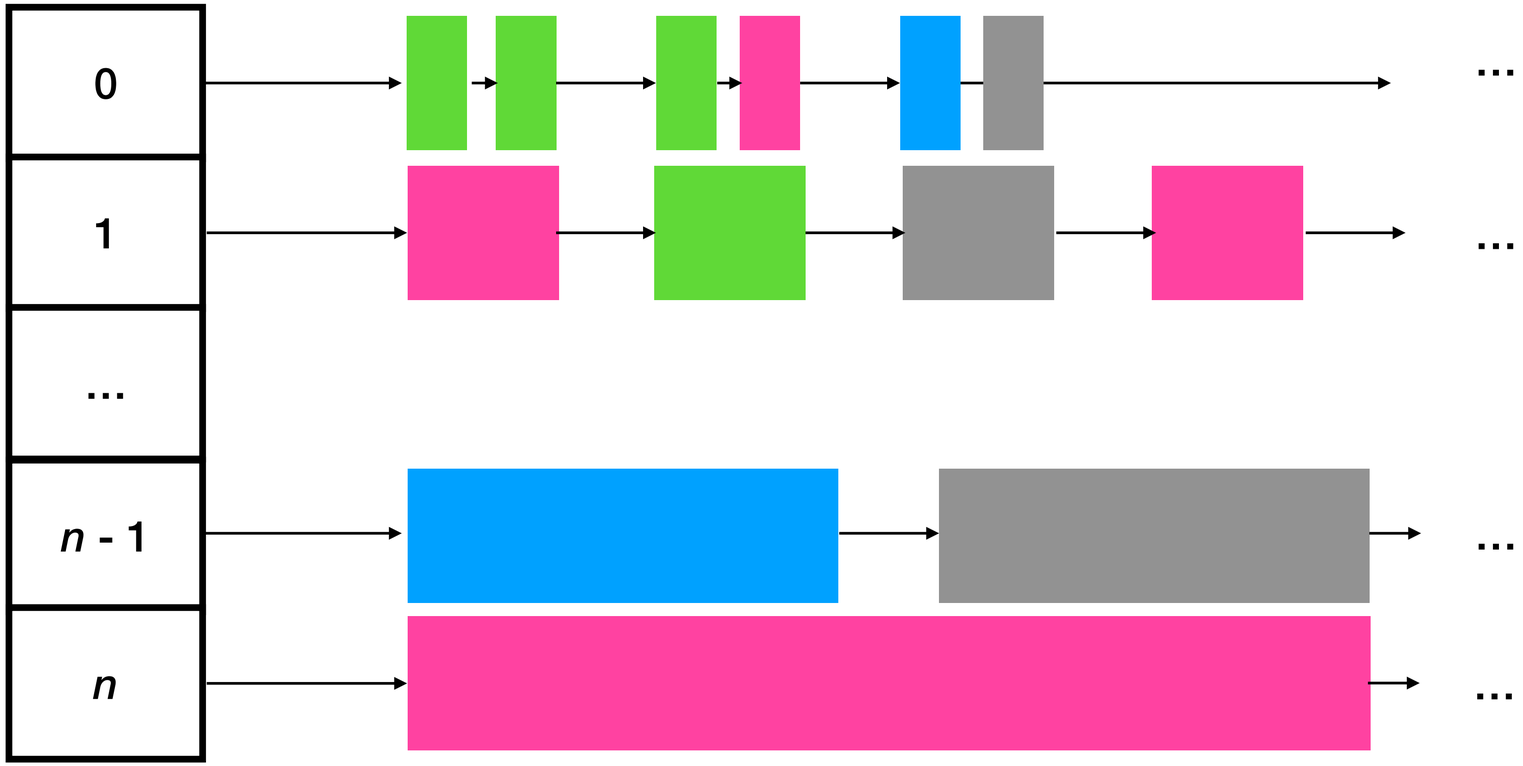
```
struct free_area {
    ...
};
```

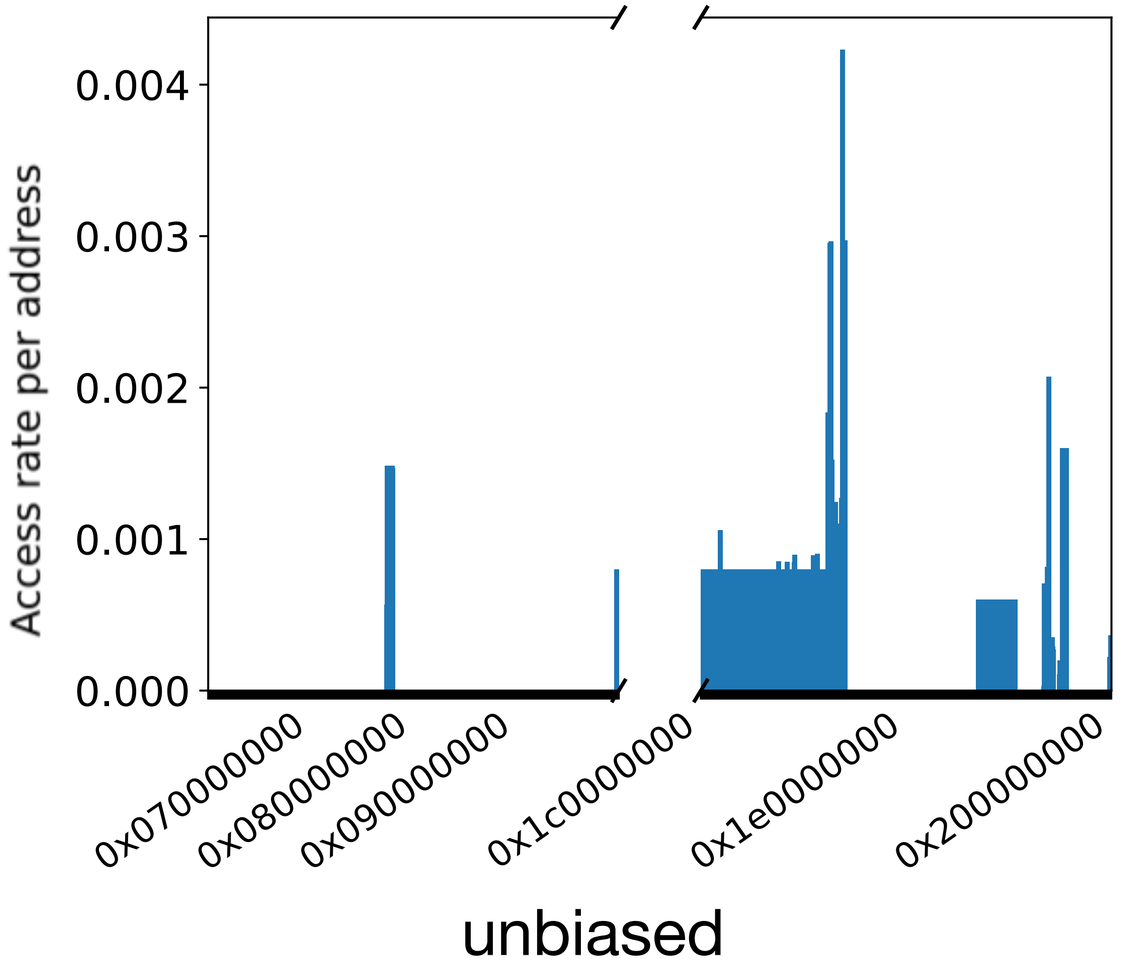


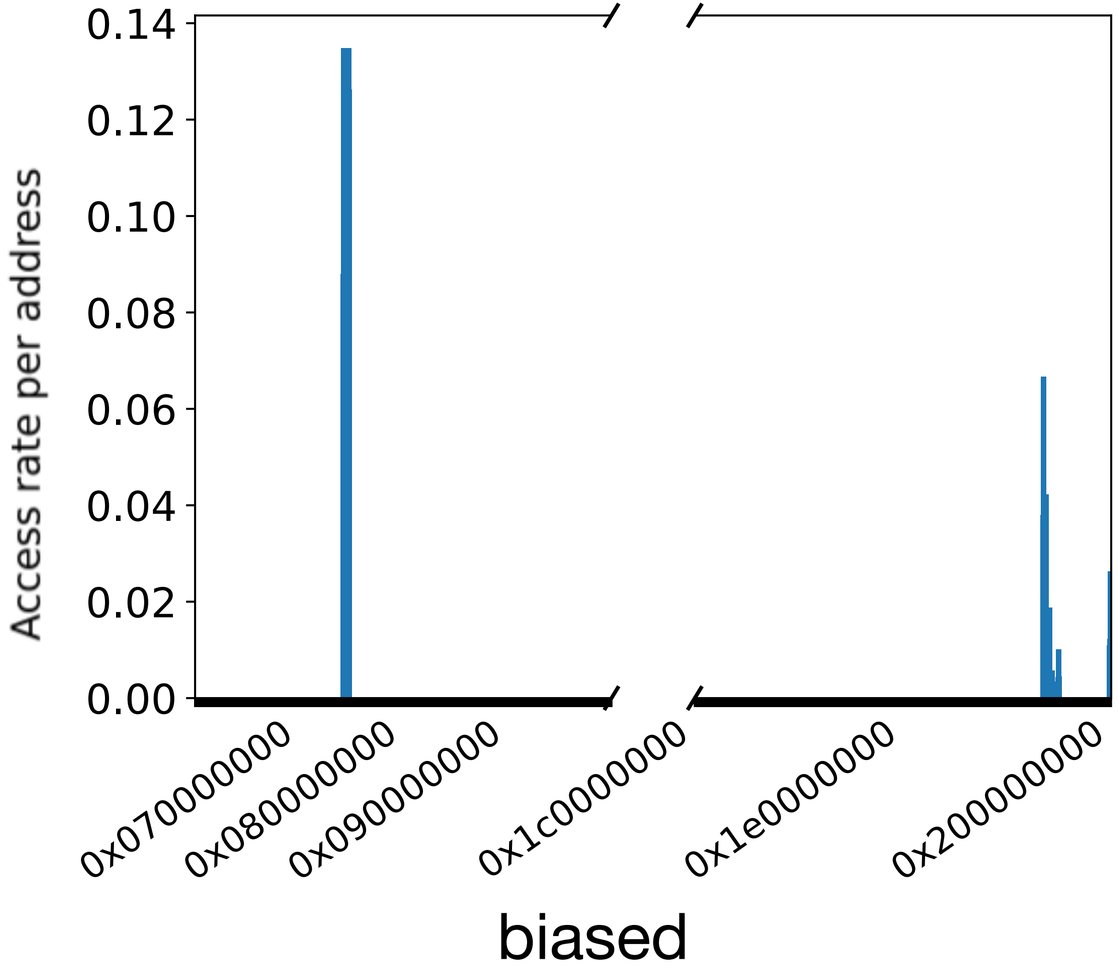
Biased Physical Page Allocation



```
struct free_area {
    ...
};
```




Access rate per address
0.004
0.003
0.002
0.001
0.000
0x070000000
0x080000000
0x090000000
0x1c0000000
0x1e0000000
0x200000000
unbiased

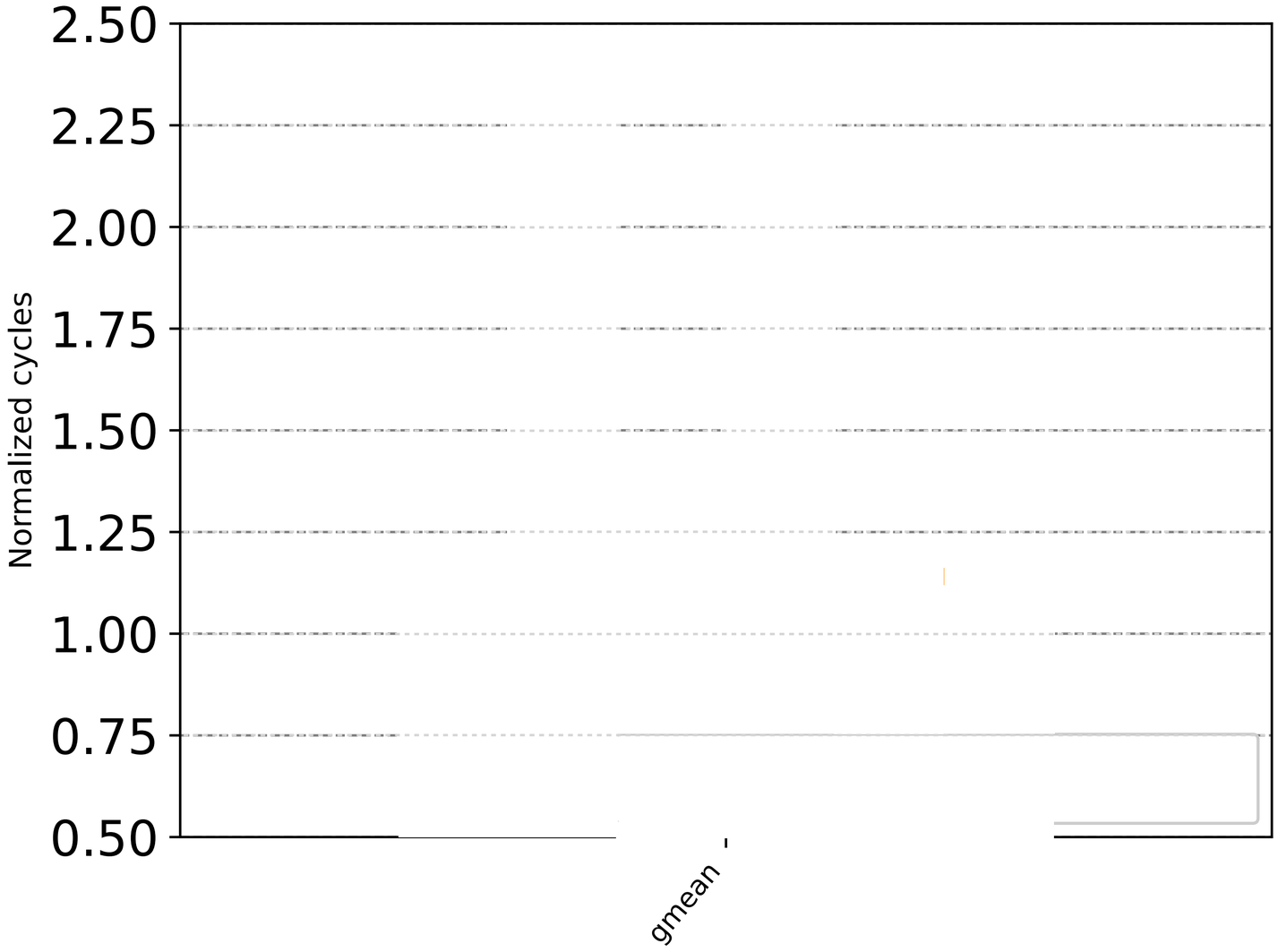

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Access rate per address
0x070000000
0x080000000
0x090000000
0x1c0000000
0x1e0000000
0x200000000
biased

Evaluation

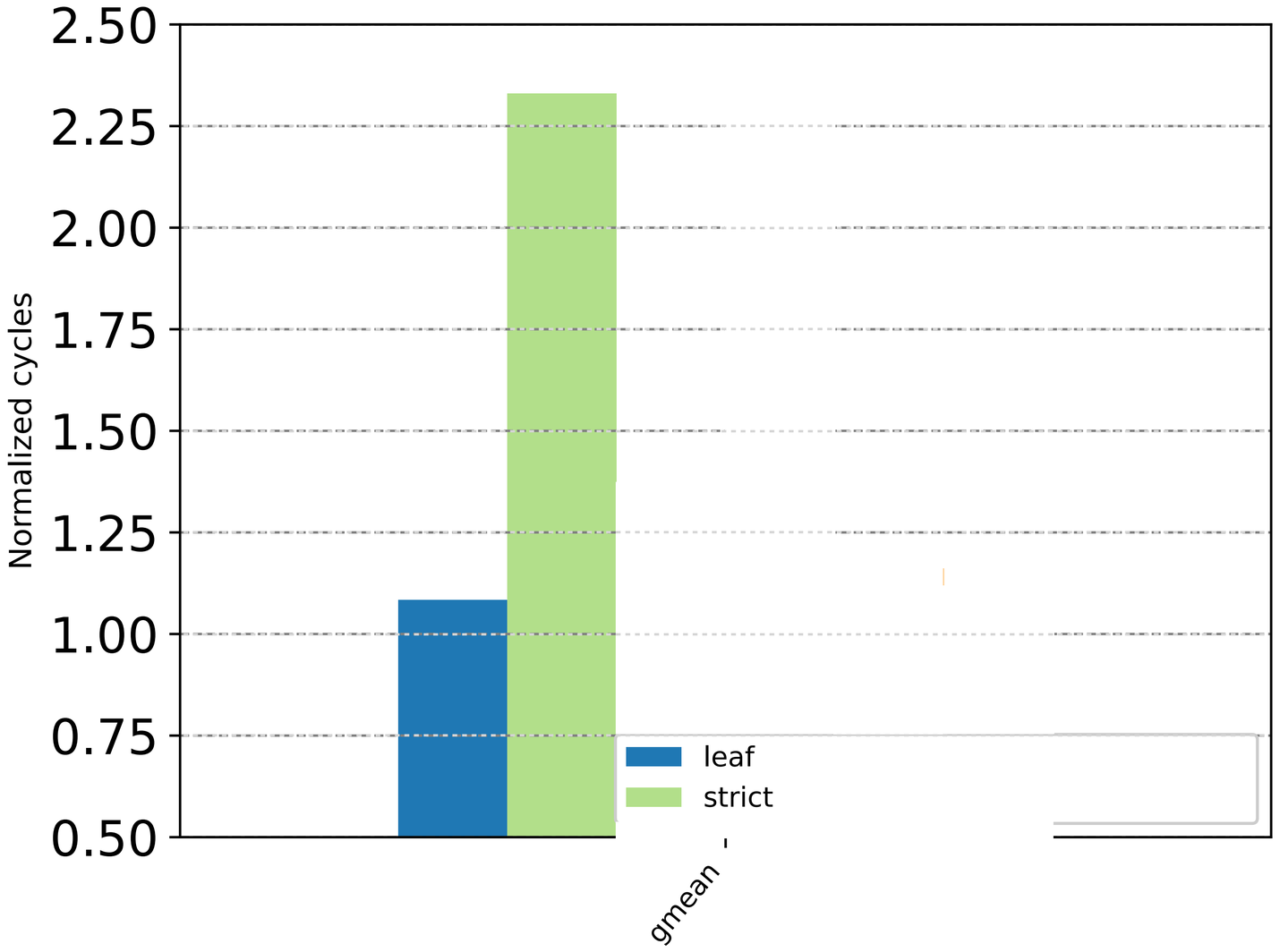
Hardware Overhead

	Volatile Overhead	Non-Volatile Overhead
AMNT	96 bytes	64 bytes
Anubis, ISCA19	37 kB	64 bytes
BMF, MICRO21	768 bytes	4 kB

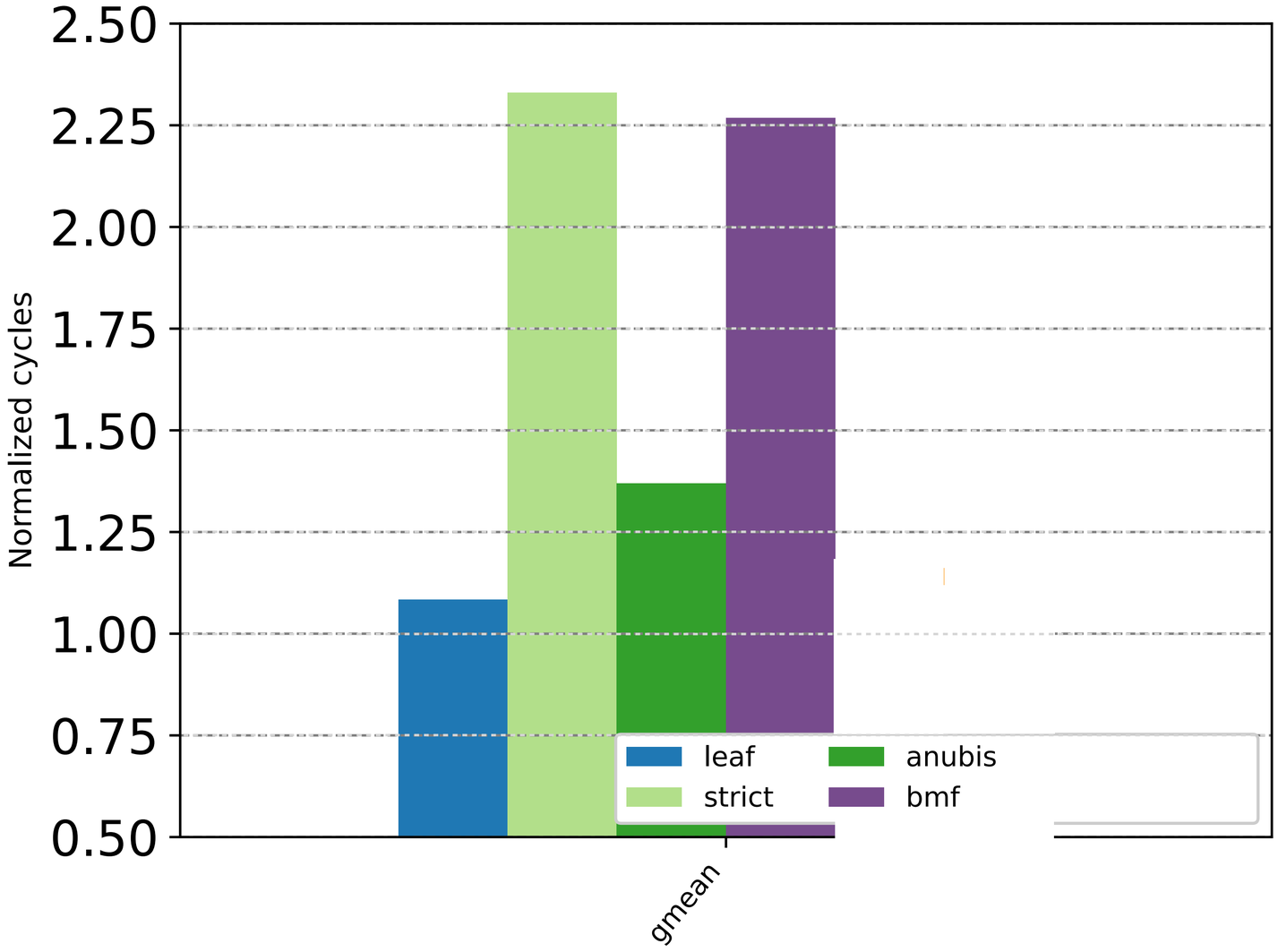
Performance Overhead



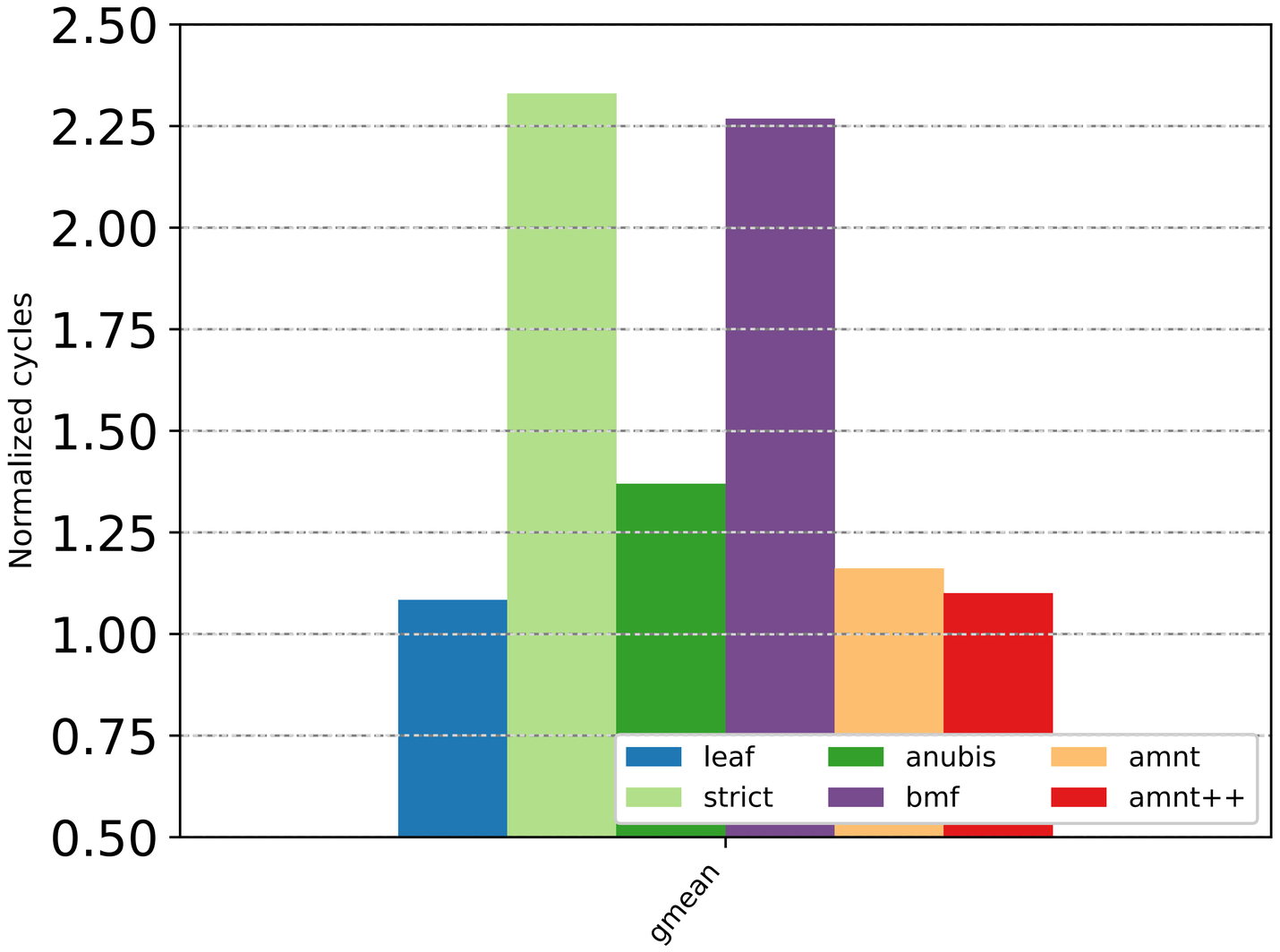
Performance Overhead



Performance Overhead



Performance Overhead

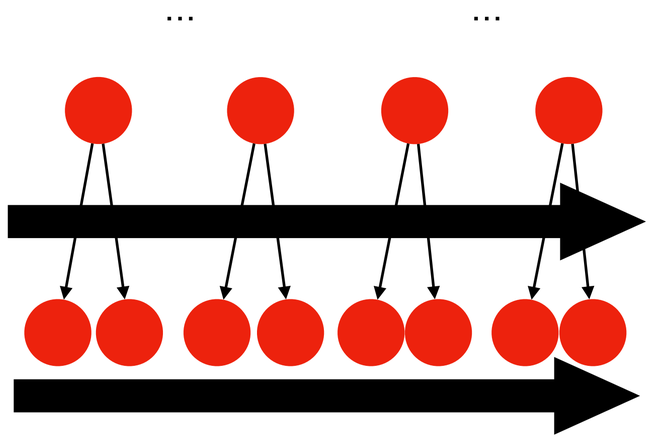


Recovery

https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html

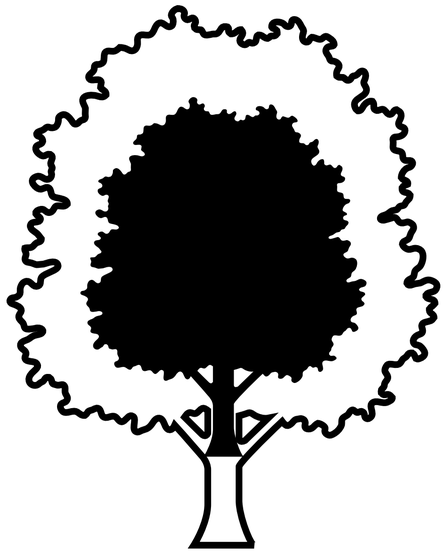
Intel® Optane™ Persistent Memory 200 Series Enables Fast Tiered Memory, Delivering 32 Percent More Bandwidth on Average[1] with up to 6 TB Total Memory per Socket[2].

SKU+	128 GB	256 GB	512 GB
USER CAPACITY+	126.7 GB	253.7 GB	507.7 GB
BANDWIDTH 67% READ; 33% WRITE 15W 64B	1.06 GB/s	1.41 GB/s	1.15 GB/s



	128GB	256GB	512GB	...	128TB
64B words to fetch	38.3M	76.7M	153.4M	...	39.3B
time to recover (leaf)	1.89 sec	2.84 sec	6.9 sec	...	30 min
time to recover (AMNT)	0.03 sec	0.04 sec	0.11 sec	...	32 sec

Conclusion



Requires less hardware than prior state of the art

Hardware Complexity

Offload complexity to software

For questions, please email me at samuel_thomas@brown.edu!

10% overhead versus leaf persistence

98.6% reduction in recovery versus leaf persistence

Performance

Recovery

Hot Region Tracking

Strictly Persist Cold Data