DAMON Updates and Future Plans:

Monitoring Parameters Auto-tuning and Memory Tiering

SeongJae (SJ) Park <sj@kernel.org>

https://damonitor.github.io

From: SJ, speaking instead of thankful DAMON community

- Authors of DAMON commits merged after LSFMM+BPF'24

Usama Arif <usamaarif642@gmail.com> Andrew Morton <akpm@linux-foundation.org>
Honggyu Kim <honggyu.kim@sk.com>
Linus Torvalds <torvalds@linux-foundation.org> Nhat Pham <nphamcs@gmail.com>
David Hildenbrand <david@redhat.com>
Zheng Yejian <zhengyejian@huaweicloud.com>
Anna-Maria Behnsen <anna-maria@linutronix.de>
Hyeongtak Ji <hyeongtak.ji@sk.com> Taotao Chen <chentaotao@didiglobal.com>
Marcelo Moreira <marcelomoreira1905@gmail.com>
Thorsten Blum <thorsten.blum@linux.dev> Joshua Hahn <joshua.hahnjy@gmail.com>
Maximilian Heyne <mheyne@amazon.de> Andrew Paniakin <apanyaki@amazon.com>
James Houghton <jthoughton@google.com> Ba Jing <bajing@cmss.chinamobile.com>
Leo Stone <leocstone@gmail.com> Jinjie Ruan <ruanjinjie@huawei.com>
Diederik de Haas <didi.debian@cknow.org>
Liam R. Howlett <Liam.Howlett@oracle.com> Peng Hao <flyingpeng@tencent.com>
Christophe Leroy <christophe.leroy@csgroup.eu>
Alex Rusuf <yorha.op@gmail.com>

DAMON in a Nutshell

- DAMON: a kernel subsystem for Data Access MONitoring
  - “DAMON-region” abstraction for
    - Access pattern information (which address range is how frequently accessed for how long time)
    - Overhead-accuracy best-effort tradeoff
- DAMOS: another side of DAMON for access-aware system operations
  - Executes monitoring results-based memory management operations
    - e.g., reclaim regions that not accessed for >=2 minutes
  - Allows fine control of resource consumption and behavior: quotas and filters
- Publicly known industry usages: Proactive reclaim (AWS) and CXL memory tiering (SK hynix)

Major DAMON Changes Since LSFMM+BPF’24 in a Glance

- Shared in LSFMM+BPF’24 and merged into mainline
  - DAMOS young page filter (6.10-rc1)
  - memory tiering support (`migrate_{hot,cold}` DAMOS actions) (6.11-rc1)
- Developed after LSFMM+BPF’24 and merged into mm-stable so far
  - Monitoring intervals tuning guide documentation (6.14-rc1)
  - **Page level monitoring** support (6.14-rc1)
  - **Monitoring intervals auto-tune** (mm-stable as of this talk)
- Followup of LSFMM+BPF’24-shared one
  - **Self-tuned memory tiering** prototype and evaluation results are recently shared
- (Important and significant amount of cleanups and documentations are also made)

Page Level Monitoring

Page Level Monitoring, a.k.a sz_ops_filter_passed

- Motivation: “I want to know access pattern of huge pages only”
- DAMOS knows it (DAMOS filters), but doesn’t expose it to monitoring results
- Idea: Expose it via API and ABI
- Implementation: Count and expose ‘`->sz_filter_passed`’ per-region, per-scheme

Page Level Monitoring Example: Detailed Information

- Shows how much of the region of a specific access pattern have passed the given filter

```
$ sudo ./damo report access --snapshot_damos_filter allow active --sort_regions_by temperature
heatmap: 988888888888888863333333222222210000000000000000000000000000000000000000
# min/max temperatures: -50,872,986,756, 905,086,383, column size: 766.312 MiB
# damos filters (df): allow active
df-pass: 9988888888888888888888888888887777777777777777777778[...]800000000006[...]40000000000004
# min/max temperatures: -49,530,000,000, 2,570,000,500, column size: 236.492 MiB
0   addr 63.165 GiB   size 719.684 MiB access 0 %   age 8 m 29.600 s  df-passed 0 B
1   addr 57.185 GiB   size 2.990 GiB   access 0 %   age 8 m 15.300 s  df-passed 0 B
2   addr 60.175 GiB   size 2.990 GiB   access 0 %   age 8 m 15.300 s  df-passed 3.324 MiB
[...]
302 addr 5.013 GiB    size 4.000 KiB   access 35 %  age 400 ms        df-passed 4.000 KiB
303 addr 5.013 GiB    size 8.000 KiB   access 50 %  age 400 ms        df-passed 8.000 KiB
304 addr 5.012 GiB    size 12.000 KiB  access 95 %  age 400 ms        df-passed 12.000 KiB
305 addr 5.013 GiB    size 8.000 KiB   access 95 %  age 400 ms        df-passed 8.000 KiB
[...]

447 addr 5.363 GiB    size 52.000 KiB  access 100 % age 25.500 s     df-passed 52.000 KiB
448 addr 5.363 GiB    size 36.000 KiB  access 100 % age 25.500 s     df-passed 36.000 KiB
449 addr 4.000 GiB    size 518.094 MiB access 5 %   age 25.700 s     df-passed 156.574 MiB
memory bw estimate: 5.650 GiB per second  df-passed: 2.117 GiB per second
total size: 59.868 GiB  df-passed 1.305 GiB
```

Page Level Monitoring Example: Recency Histogram

- Page level information can be accumulated in multiple visualization

```
$ sudo ./damo report access --snapshot_damos_filter allow active \
                            --style recency-sz-hist
<last accessed time (us)> <df-passed size>
[0 ns, 7.380 s)             180.168 MiB |*****                 |
[7.380 s, 14.760 s)         891.234 MiB |********************|
[14.760 s, 22.140 s)        234.203 MiB |******              |
[22.140 s, 29.520 s)        17.480 MiB  |*                   |
[...]
[1 m 6.420 s, 1 m 13.800 s) 9.250 MiB   |*                   |

<last accessed time (us)> <total size>
[0 ns, 7.380 s)             531.879 MiB |*                   |
[7.380 s, 14.760 s)         5.989 GiB   |***                 |
[14.760 s, 22.140 s)        732.332 MiB |*                   |
[22.140 s, 29.520 s)        5.631 GiB   |***                 |
[...]
[1 m 6.420 s, 1 m 13.800 s) 41.073 GiB  |********************|
memory bw estimate: 10.645 GiB per second  df-passed: 3.787 GiB per second
total size: 59.868 GiB  df-passed 1.301 GiB
```

Page Level Monitoring: Continu{ed,ing} Developments

- More page level DAMOS filter types are implemented:
  - Usama Arif (THP shrinker developer) contributed ‘hugepage_size’ DAMOS filter type
  - Nhat Pham (zswap developer) contributed ‘active’ LRU pages DAMOS filter type
- Future Work: Reducing overhead
  - Overhead is controllable, but still high
    - Controlled usage: apply page level monitoring to only interesting regions
    - e.g., damo report access --address 7.2G 7.4G --snapshot_damos_filter hugeapge_size 2M max
  - Sampling approaches might help

Monitoring Intervals Auto-tuning

Monitoring Intervals Tuning

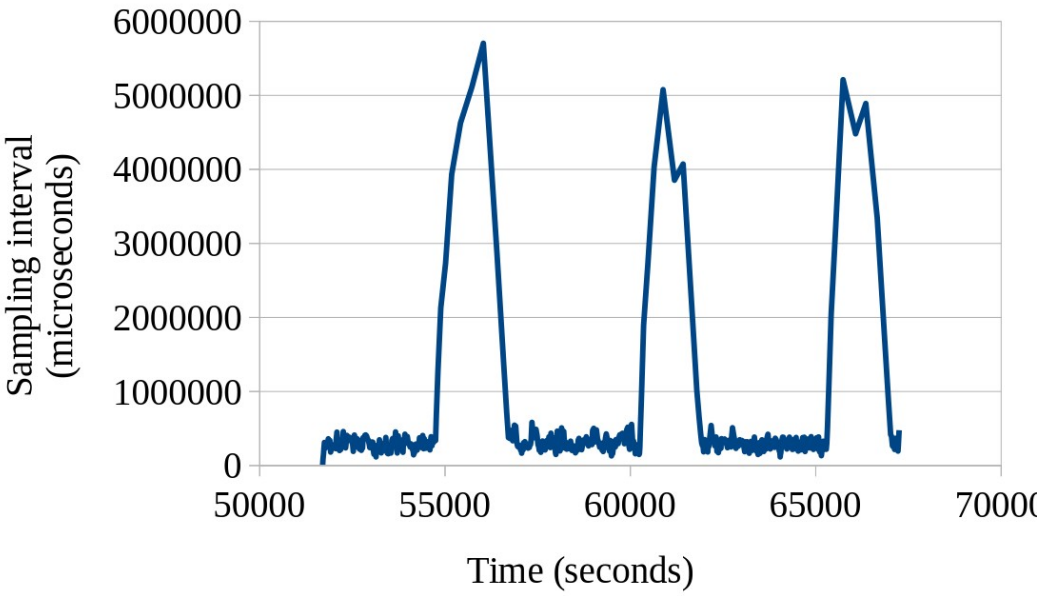
- Aggregation interval need to be tuned
  - If it is too short, everything looks cold
  - If it is too long, everything looks hot
  - The default value (100 ms) is far from a magic value
- The tuning is required for different systems and workloads (repetitive and time consuming)
  - Every known DAMON production users did the tuning on their own ways
- Tuning guide is finally documented on v6.14
  - Readable one is good, executable (automated) one is better

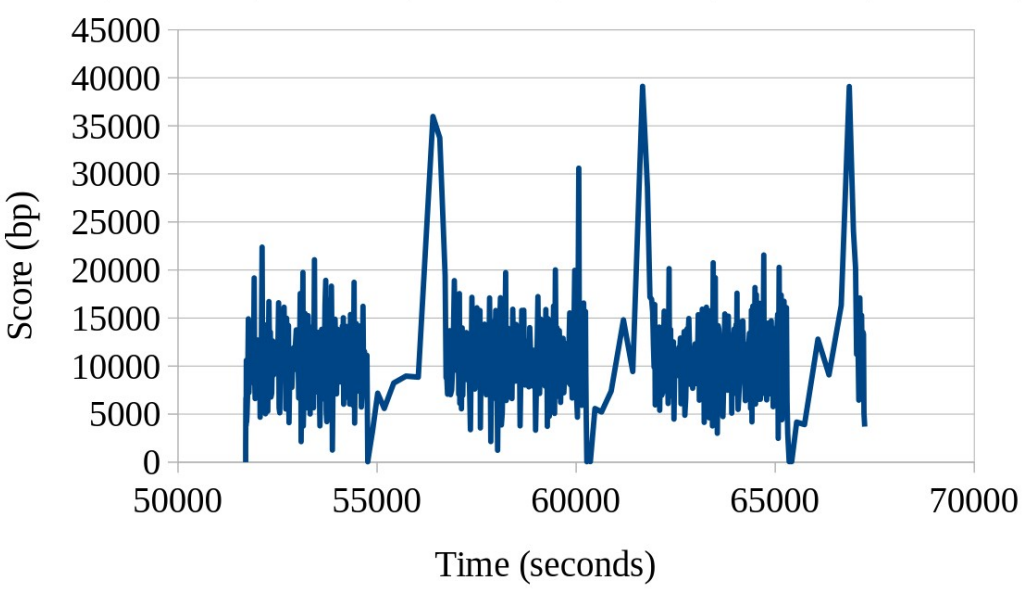
Monitoring Intervals Tuning Automation

- Better question: “how much access events each snapshot should capture?”
  - Metric: access events ratio (DAMON-observed : DAMON-could-observe-in-max)
- Run feedback loop for the target value, feeding access events ratio of current snapshot
- Let users set how frequently adaptation should happen, and min/max sampling interval
- Still questions exist, but easier to answer
  - Target access events ratio depend on users’ interest: higher target to find more hot regions
    - 4 % maybe a good default value (by 20:80 rule, can capture 64% meaningful accesses)
  - Min/max sampling interval can naively set
    - [5ms, 10s] maybe a good default range
- Merged into mm tree during 6.15 development cycle

Intervals Auto-tuning on a Real-world Server Workload

- Sampling interval and tuning score continuously change, and converge for given situation
  - Sampling interval convertges to 370ms under usual load, ~4-5 seconds under light load
  - Tuning score converges to the goal (10,000 bp)





Results and Next Steps

- Results: Always colorful snapshot with low overhead (0.1% CPU on the real workload)
- Next step: CONFIG_DAMON_ALWAYS_ON?

Self-tuned Memory Tiering

Recap of LSFMM+BPF 24 Shared DAMON-based Memory Tiering Idea

- For each CPU-unattached NUMA node,
  - If the node has a lower node,
    - Demote cold pages of the current node to the lower node,
      aiming little fraction (e.g. 5%) of free memory of the current node
  - If the node has an upper node,
    - Promote hot pages of the current node to the upper node,
      aiming big fraction (e.g., 96%) of used memory of the _upper_ node
- Use upper tier as much as possible for hotter data with controlled aggressiveness
- Support N (CPU-unattached) tiers

```
node 0 (fast):   Demote cold pages in node 0 aiming 5% free memory of node 0
node 1 (slow):   Promote hot pages in node 1 aiming 96% used memory of node 0
                 Demote cold pages in node 1 aiming 5% free memory of node 1
node 2 (slowoo): Promote hot pages in node 2 aiming 96% used memory of node 1
[...]
```

Self-tuned DAMON-based Memory Tiering Development After LSFMM+BPF

- Essential patches for user-tuned tiering are merged into v6.11
- Implemented a DAMON module for self-tuned memory tiering
  - With auto-tuned intervals, promote/demote hot/cold pages up to 200 MiB/s quota that auto-tuned aiming 99.7%/0.5% utilization/free ratio of upper tier memory
- RFC implementation: https://lore.kernel.org/20250320053937.57734-1-sj@kernel.org
  - Introduce NUMA nodes utilization/free space ratio DAMOS tuning goal
  - Introduce sample DAMON kernel module for easily using memory tiering

Evaluation Setup

- Machine: 250 GiB DIMM-connected node 0 and 55 GiB CXL-connected node 1
- Workload
  - Taobench/DCPerf (open source benchmark for in-memory caching workload)
  - Parameters set to use ~90% of total memory (~270 GiB)
- Configs
  - Baseline: Occasionally promote cold and demote hot pages, to simulate dynamic access pattern
  - Numab_tiering: Enable NUMAB-2 promotion and LRU-based demotion on Baseline
  - DAMON_tiering: Enable DAMON-based memory tiering on Baseline

Evaluation Result: Performance

Config	Score	Stdev	(%)	Normalized
Baseline	1.6165	0.0319	1.9764	1.0000
Numab_tiering	1.4976	0.0452	3.0209	0.9264
DAMON_tiering	1.6881	0.0249	1.4767	1.0443

- DAMON_tiering improves performance by 4.4 %
- Numab_tiering degrades performance by 7.3 %
  - More investigation is needed, but current suspects are:
    - Direct migration has blocked Taobench’s progress
    - Took long time to find hot pages scattered in large area
- Disclaimer: Artificial frequent access pattern change

Next Steps

- Land NUMA nodes utilization / free space ratios DAMOS goal metrics
- Implement DAMON module for general memory tiering that just works
  - Automated tiers and promotion/demotion path identification
- General heterogeneous (NUMA-abstract) memory management
- CPU/Bandwidth-aware migrations

Discussion Time

Discussion Points

- Page Level Monitoring
- Monitoring Intervals Auto-tuning
- Self-tuned Memory Tiering

Backup Slides

Self-tuned DAMON-based Memory Tiering vs NUMAB-2 and LRU-based Demotion

DAMON/S vs NUMAB-2 promotion

- Advantages
  - Migrate in async thread
  - Handle unmapped pages
  - Adapt promotion aggressiveness; minimize unnecessary demotion-promotion ping-pong
  - Can detect hot pages in any location
- Disadvantages
  - Fairness: DAMON loves hot threads!
  - Could take longer time to find small hot pages

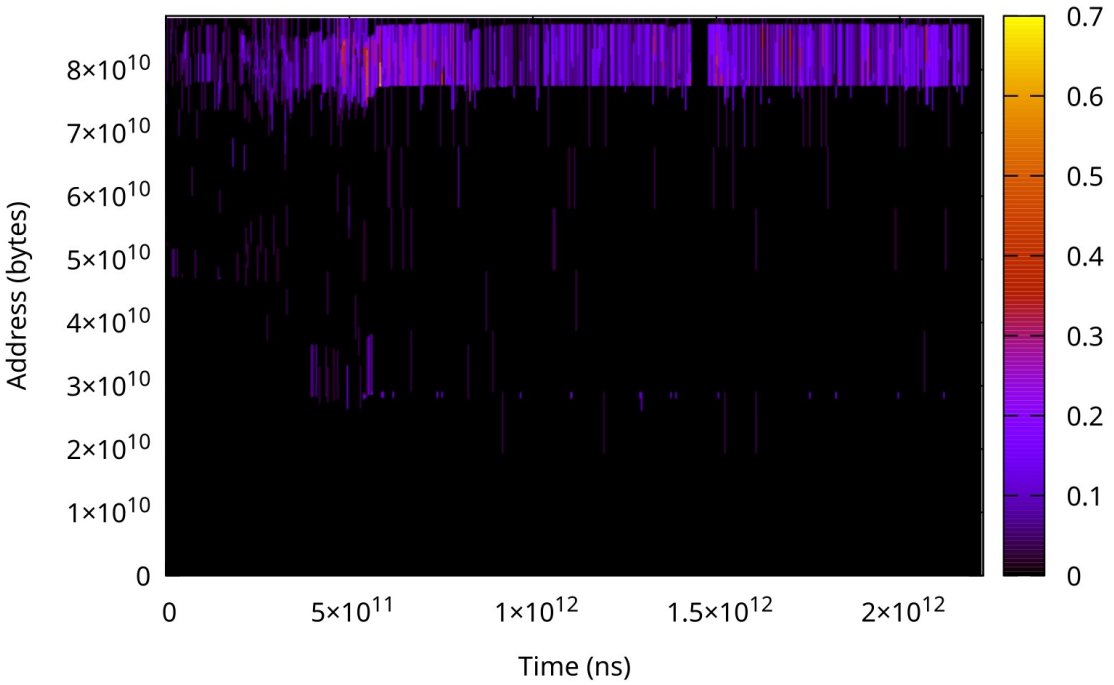
DAMON/S vs LRU-based Demotion

- Advantages
  - Balance better with DAMOS auto-tuning based promotion; minimize unnecessary ping-pong
  - Can find colder pages faster, if memory pressure happens only occasionally
- Disadvantage
  - DAMOS maybe slower than LRU-based one at handling significant memory pressure
- Running LRU-based direct demotion and DAMON/S-based proactive one together may be good team work

Memory Tiering Evaluation: More Details

Taobench (64GiB memsize) Access Pattern

- Only small amount of memory is continue being hot
- Only demotion benefits this access pattern



Evaluation Result: Performance

- Baseline_0: Taobench without artificial access pattern changes (hot pages demotion and cold pages promotion)

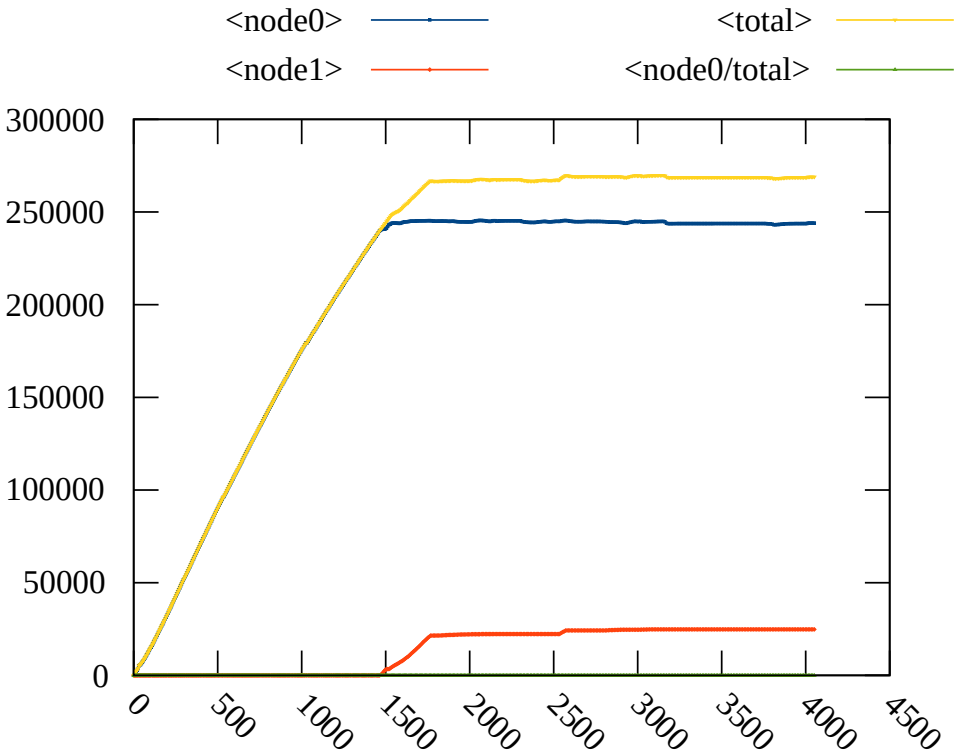
Config	Score	Stdev	(%)	Normalized
Baseline_0	1.5989	0.0131	0.8193	0.9891
Baseline	1.6165	0.0319	1.9764	1.0000
Numab_tiering	1.4976	0.0452	3.0209	0.9264
DAMON_tiering	1.6881	0.0249	1.4767	1.0443

Taobench_static Performance Results

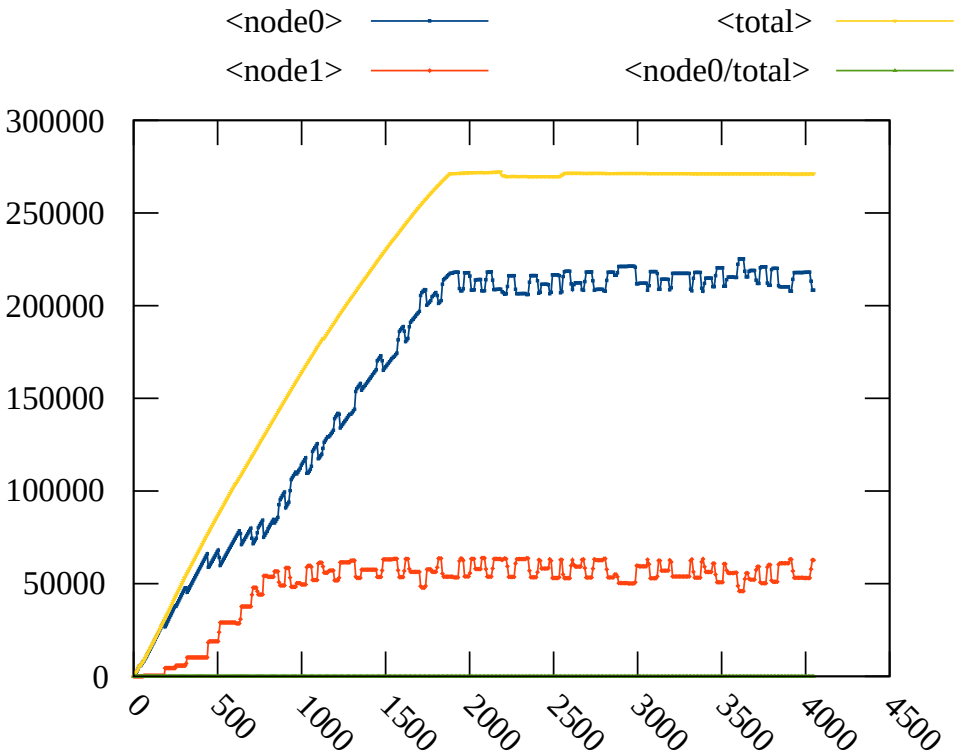
- Why damon_tiering degrade?
  - DAMON-demotion is too slow (200 MiB/second) to handle the memory pressure
  - Since LRU-demotion is turned off, reclaim happened
- Why {numab,damon}-promotion doesn’t improve?
  - Static access pattern: No value to promote anything

Config	Score	Stdev	(%)	Normalized
Baseline_0	1.6175	0.0131	0.8112	1.0000
demote_only	1.7178	0.0166	0.9635	1.0620
numab_tiering	1.7126	0.0298	1.7399	1.0588
damon_tiering	1.5754	0.0134	0.8509	0.9994
demote+damon_tiering	1.6939	0.0102	0.5998	1.0472

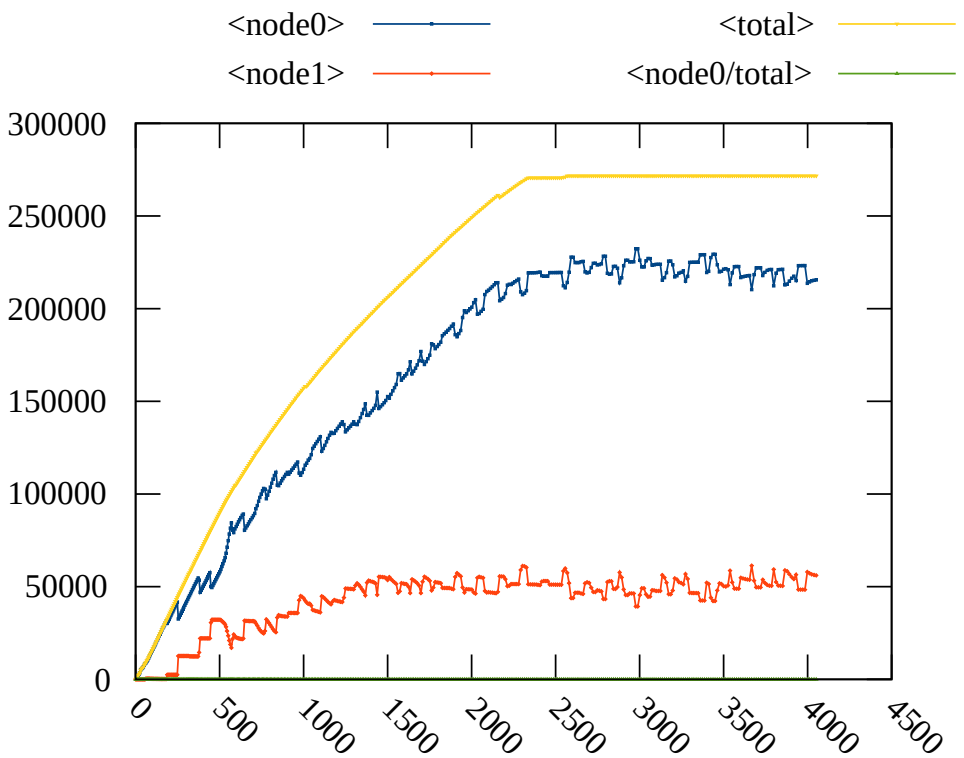
Taobench Memory Usage: Baseline0



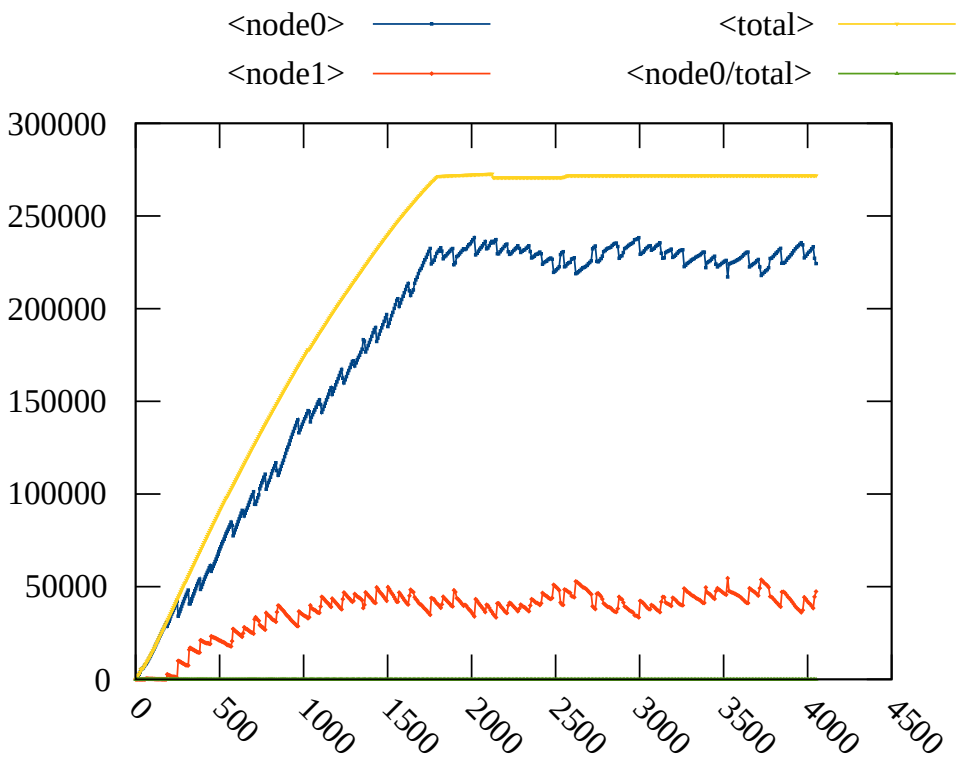
Taobench Memory Usage: Baseline



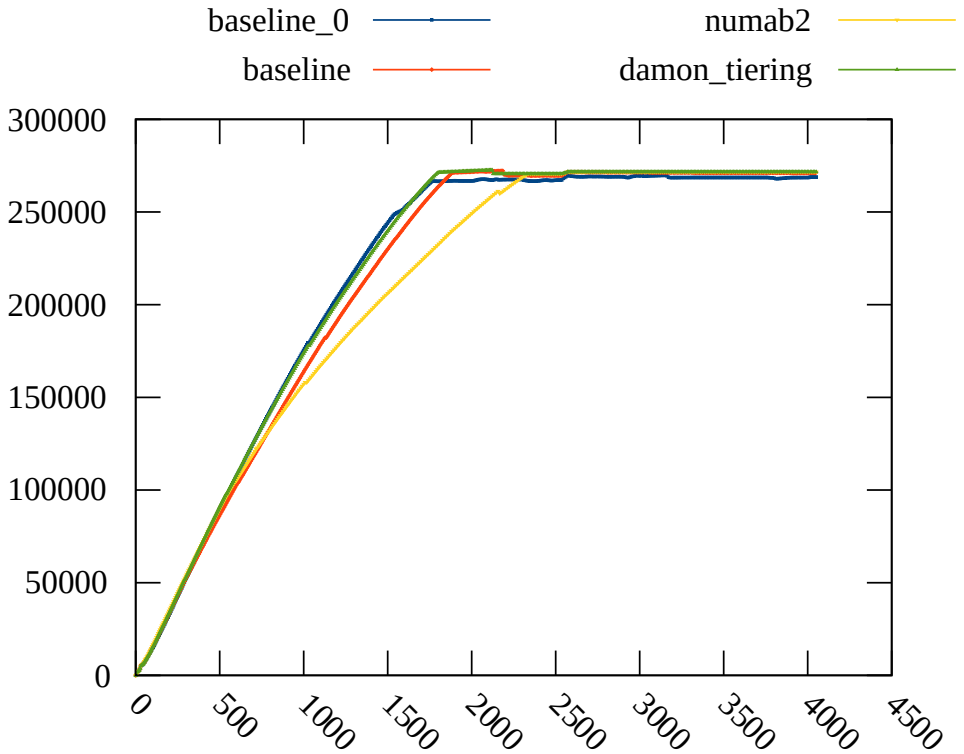
Taobench Memory Usage: Numab2



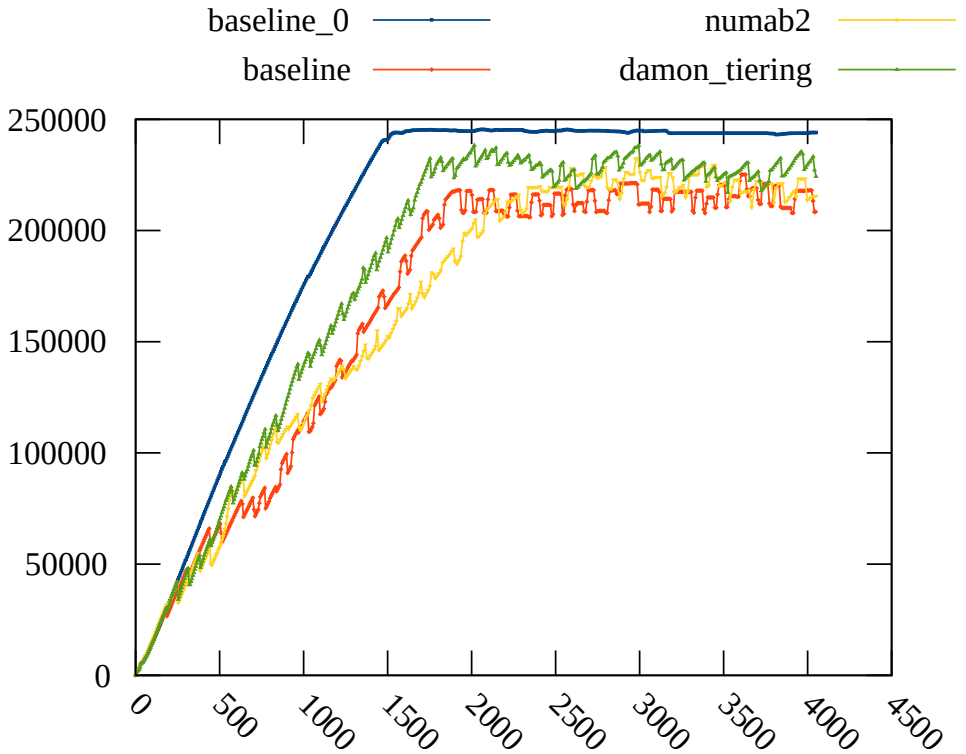
Taobench Memory Usage: DAMON-tiering



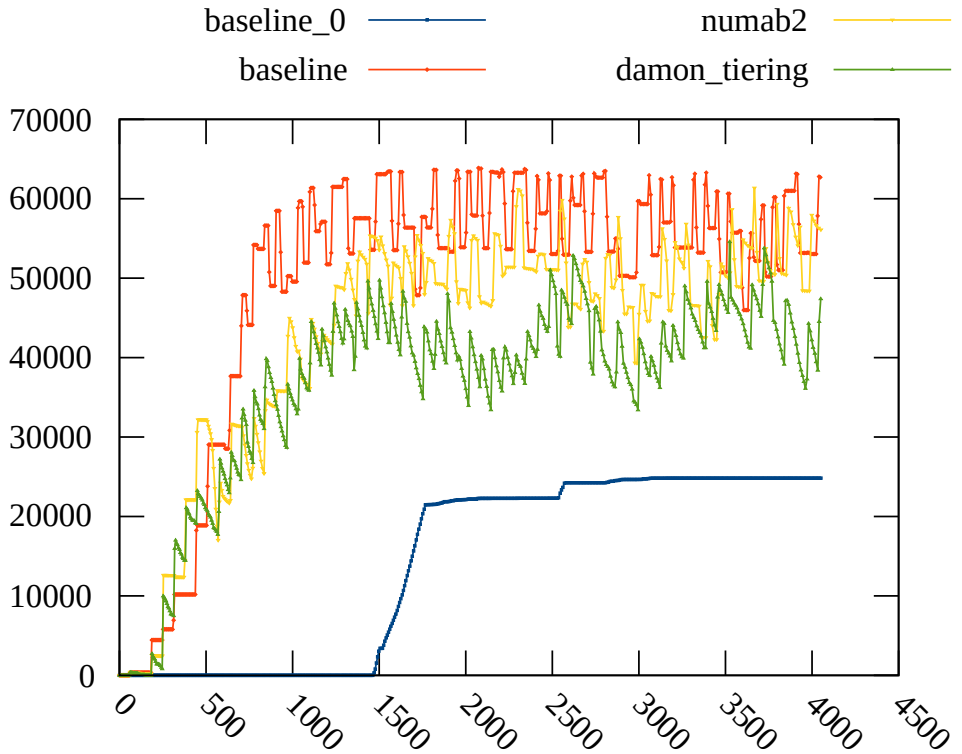
DAMON-Tiering: Memory Usage: Total



DAMON-Tiering: Memory Usage: Node 0



DAMON-Tiering: Memory Usage: Node 1



More Updates

Updates to LSFMM+BPF’24 Shared Projects

- DAMOS Auto-tuning based Tiered Memory Management
  - A prototype is implemented and evaluated: 4.4% performance improvement achieved
- Access/Contiguity-aware Memory Auto-scaling
  - No progress at all, sorry; It’s unclear if it will be prioritized
- Monitoring improvements
  - Auto-tuning: Intervals auto-tuning is actually for this
  - higher accuracy: A plan for regions adjustment improvement is shared
- Write-only monitoring: Beyond Accessed bits is for this
- LRU-sort auto-tuning: No progress yet; stay tuned
- Access-aware THP assistant: THP access monitoring is for preparation of this
- CPU-aware monitoring and NUMA-balancing: Beyond Accessed bits is for this

More Plans

Keep The Monitoring On

- DAMON ABI users (user-space) and API callers (kernel-space) should start DAMON for them
  - Difficult to “just” use DAMON
- Planning to be able to “just” be used
  - Start DAMON for physical address space with intervals auto-tuning at boot, keep it on always
    - CONFIG_DAMON_ALWAYS ?
  - Let users and kernel components use the system-started DAMON
    - struct damon_ctx *damon_get_always_on_ctx(void);
    - Read monitoring results from it
    - Add and remove (stacking) DAMOS schemes to it
    - Report observed access to it

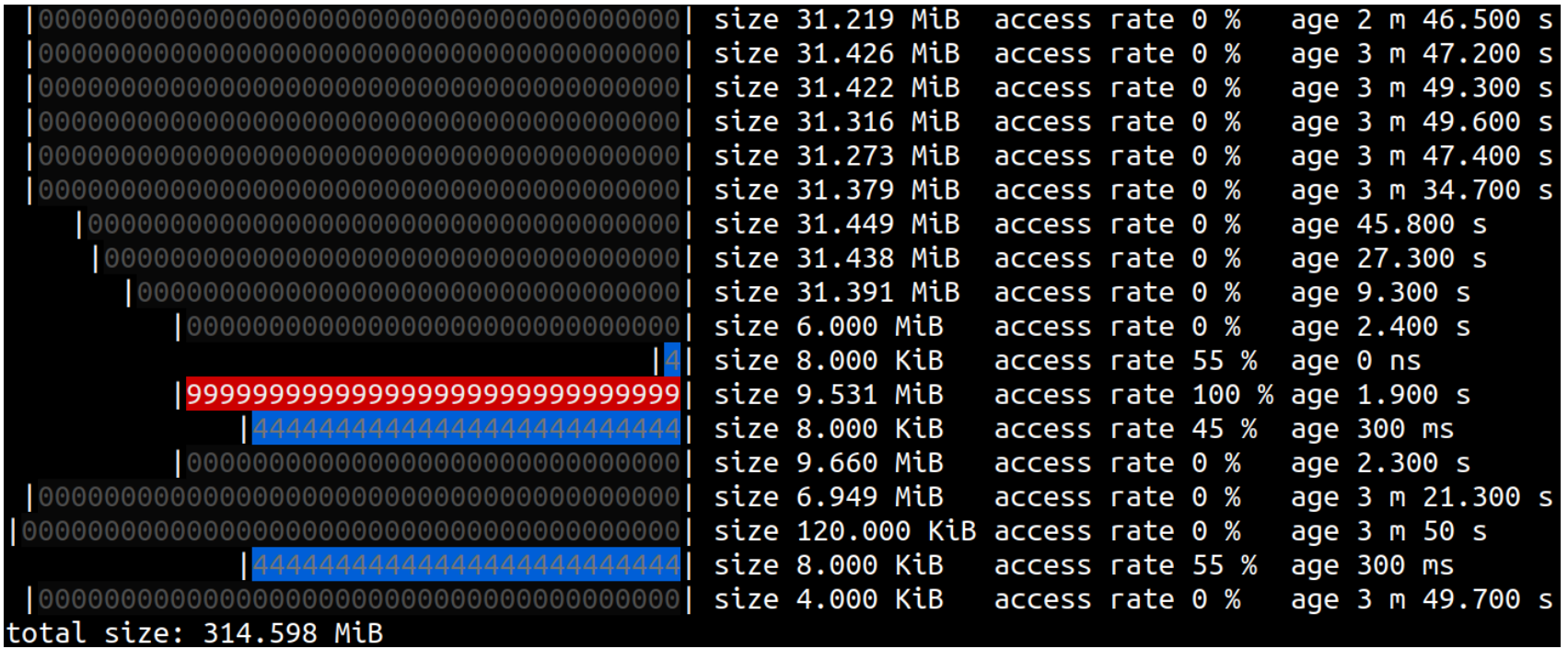
Beyond Accessed Bits

- Page accessed bits and PG_Idle are the main source of DAMON today
- Desired Usages of DAMON
  - Per-CPU access for cache-aware scheduling
  - Write-only monitoring for live migration VM target selection
  - Memory bandwidth monitoring for process/jobs scheduling
- Planning to expand with more source of information
  - Page faults
  - AMD IBS and similar h/w features
  - Memory bandwidth PMU

DAMON in a Nutshell

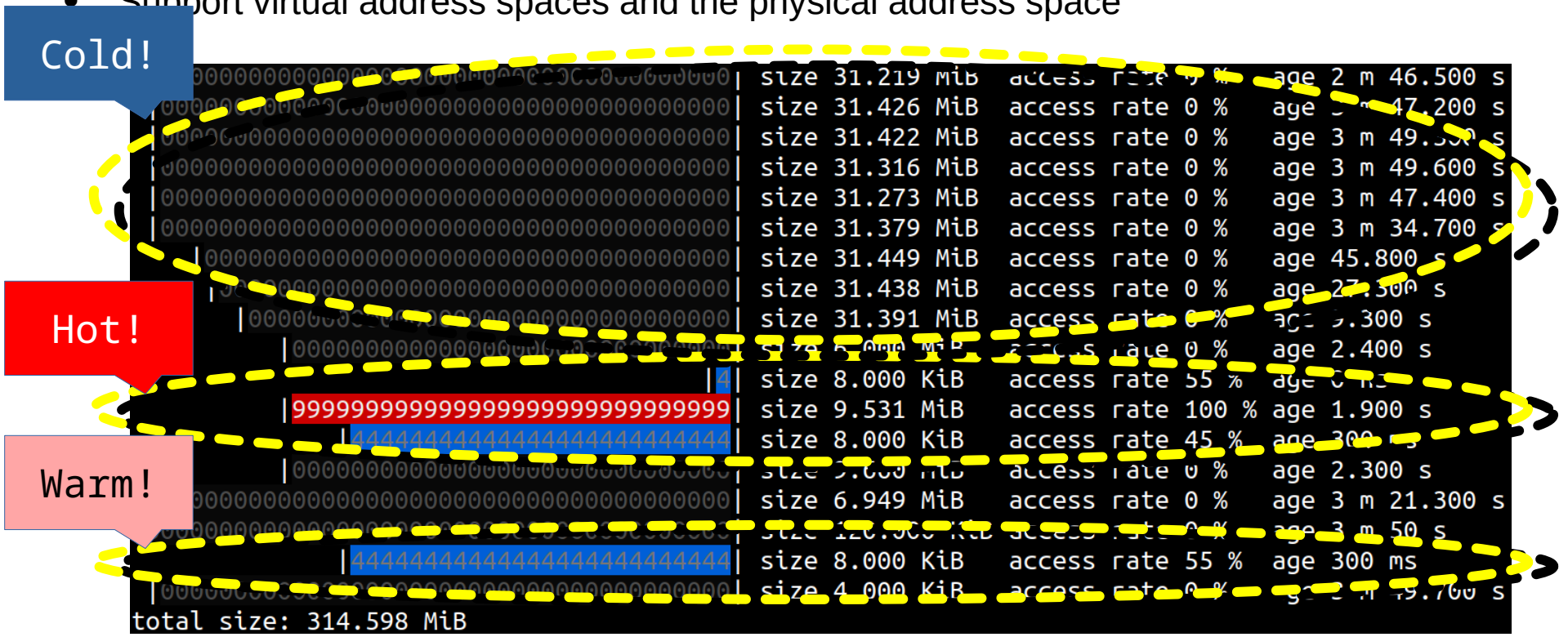
DAMON: Access Pattern Snapshot Generator

- Inform which *address range* is how *frequently* accessed for how *long* time
- Support virtual address spaces and the physical address space



DAMON: Access Pattern Snapshot Generator

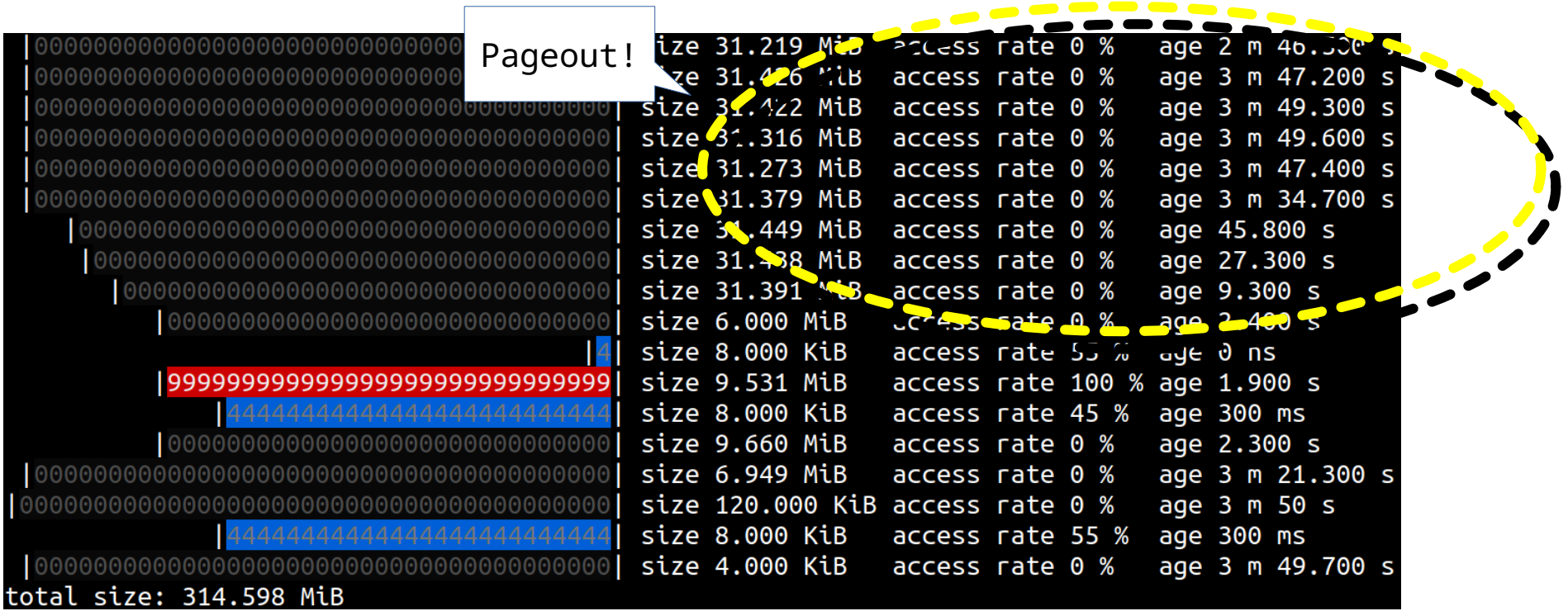
- Inform which *address range* is how *frequently* accessed for how *long* time
- Support virtual address spaces and the physical address space



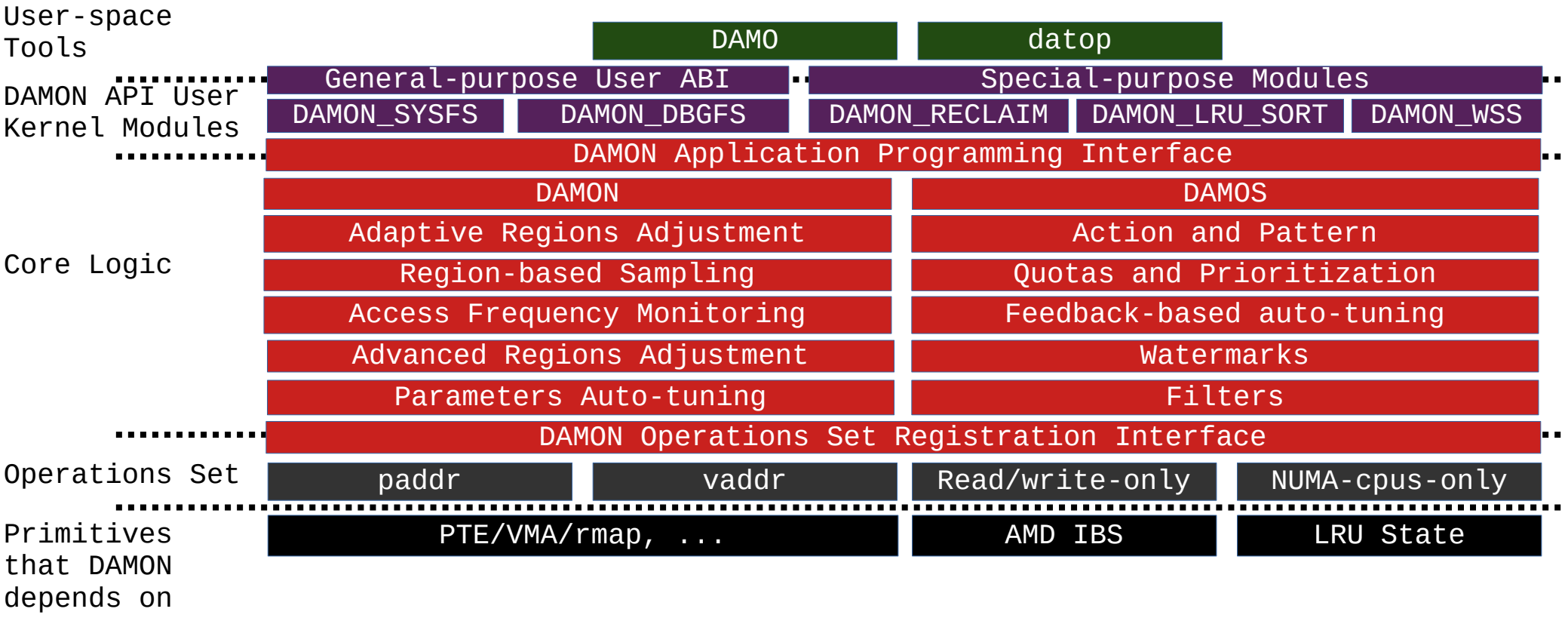
DAMOS: DAMON-based Operation Scheme

- Apply memory operation actions to regions of interesting access pattern

```
# # pageout memory regions that not accessed for >=5 seconds
# damo start --damos_action pageout --damos_access_rate 0% 0% --damos_age 5s max
```

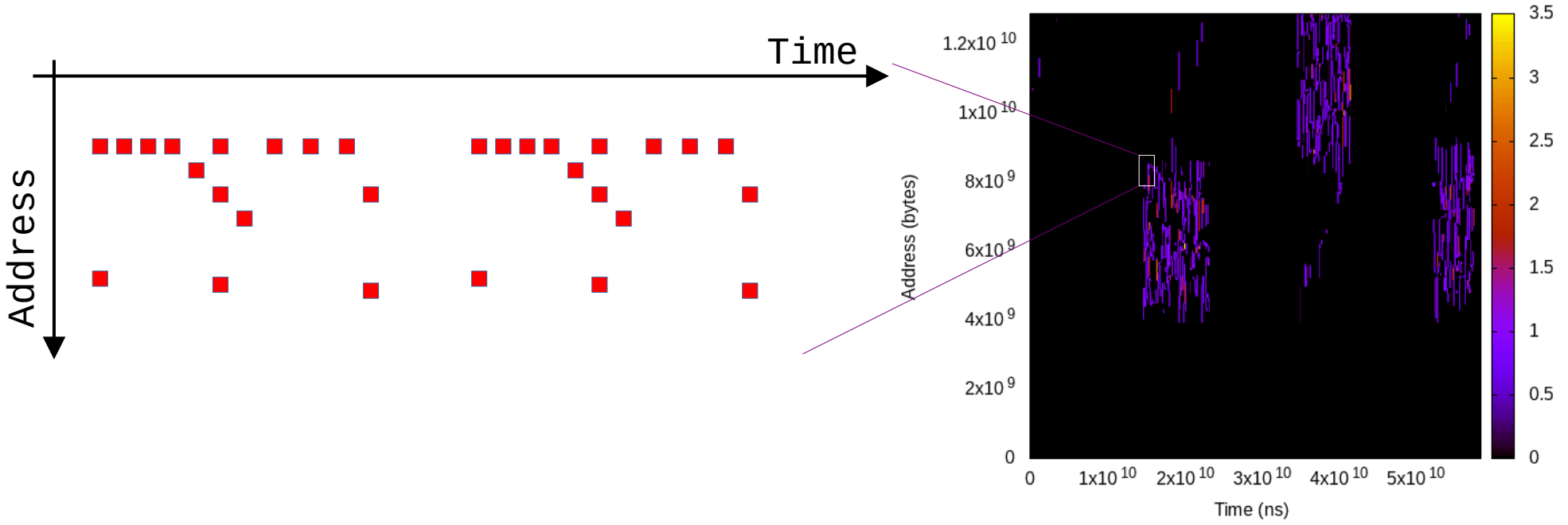


DAMON Stack, In a Future



DAMON: Kernel Subsystem for Data Access Monitoring and Access-aware System Operations

Data Accesses: Events on Space/Time of Memory



Data Access Monitoring: Hope, Real, and DAMON

- Hope: Precise, Complete, Light
  - Time granularity: CPU cycle / # CPUs (or, speed of light)
  - Space granularity: bit (or, electron)
  - Keep complete history (from Bigbang)
  - Lightweight enough to run on production systems
- Real: Expensive, YAGNI without tradeoff
  - O(memory size) time, O(memory size * total monitoring time) space overhead
  - Do you really care if a bit was accessed five years ago?
- DAMON: accuracy vs overhead tradeoff
  - Scalable, best-effort, aim real-world products usage

Region-based Space Handling

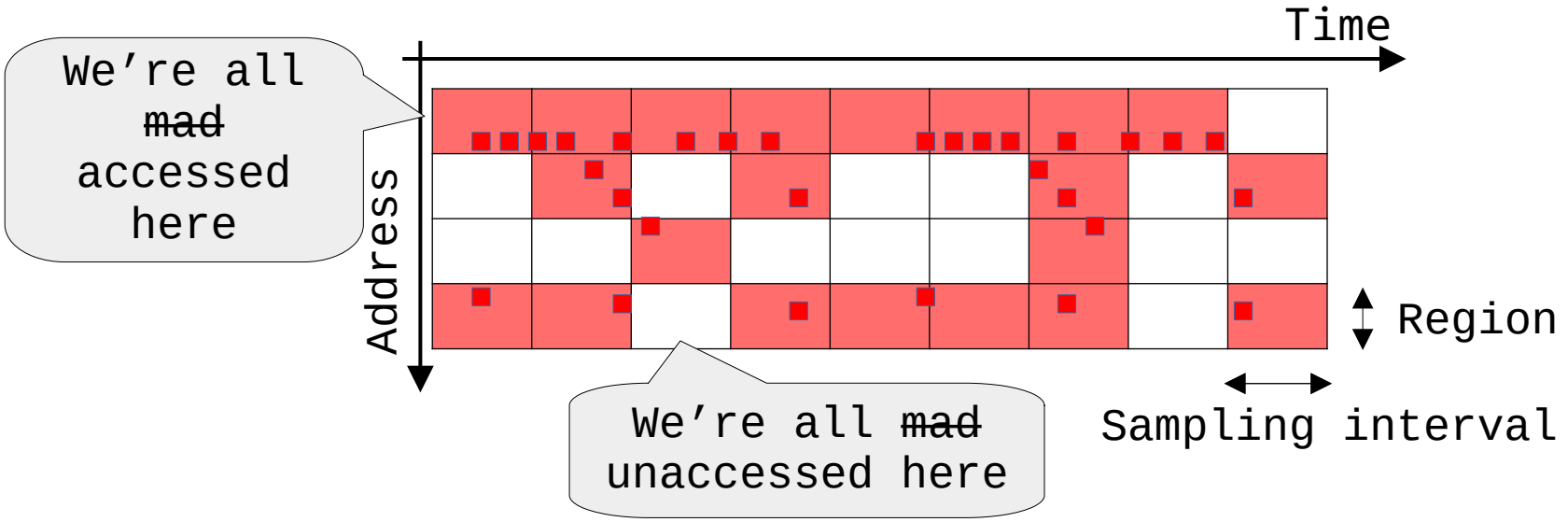
Region: Access Monitoring Unit

- Defined as
  - A sub-area of the memory's space-time
  - A collection of adjacent elements that having similar access pattern
- Access check of one element per region is enough
- e.g., "This page is accessed within last 1 second; a cacheline is checked"

```
$ cat wonder_region_1
We're all ~~mad~~ [un]accessed here
```

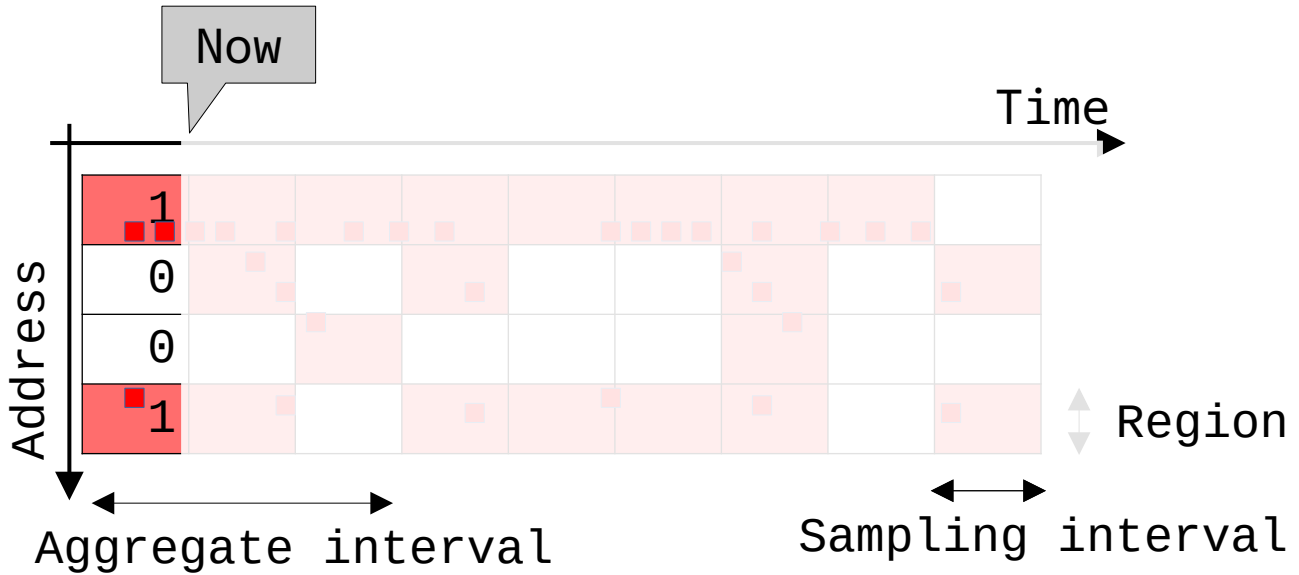
Fixed Space/Time Granularity, <=1 Access Frequency

- Sort of periodic fixed granularity idleness monitoring
- Time overhead: “memory size / space granularity”
- Space overhead: “time overhead * monitoring time / time granularity”
- Reduced, but still ruled by memory size and total monitoring time



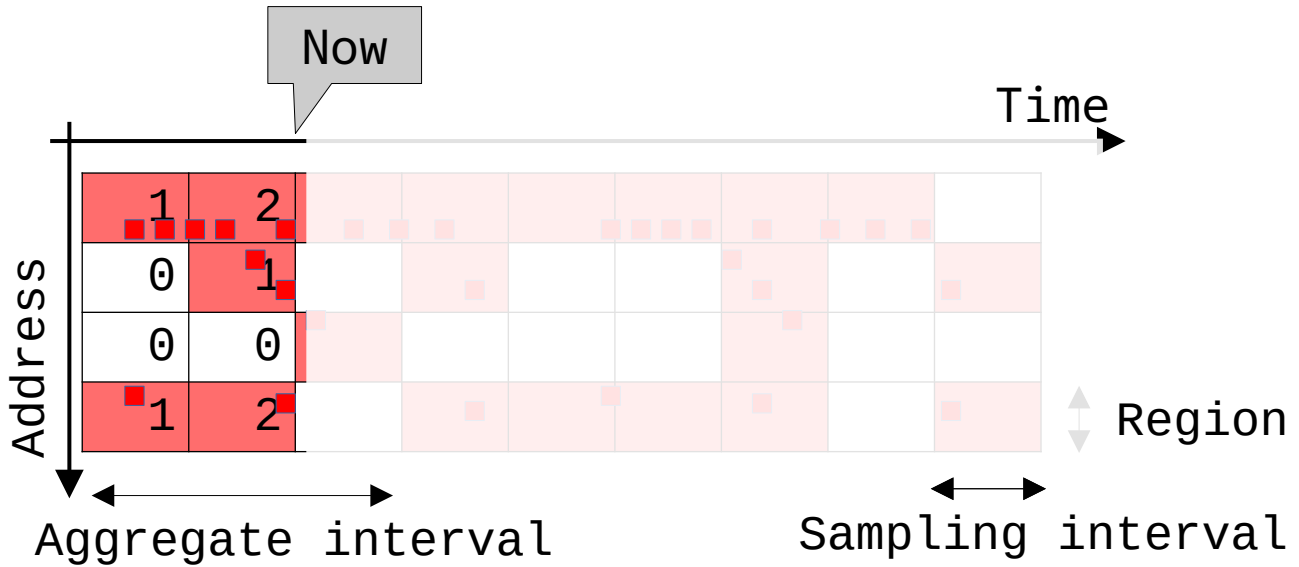
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter



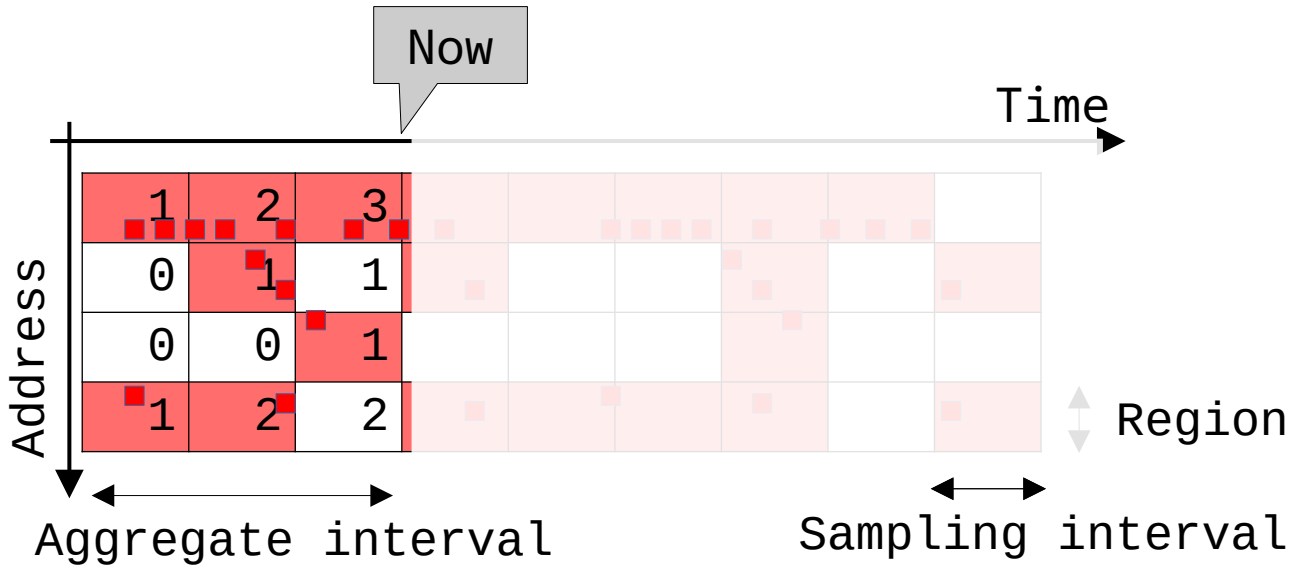
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter



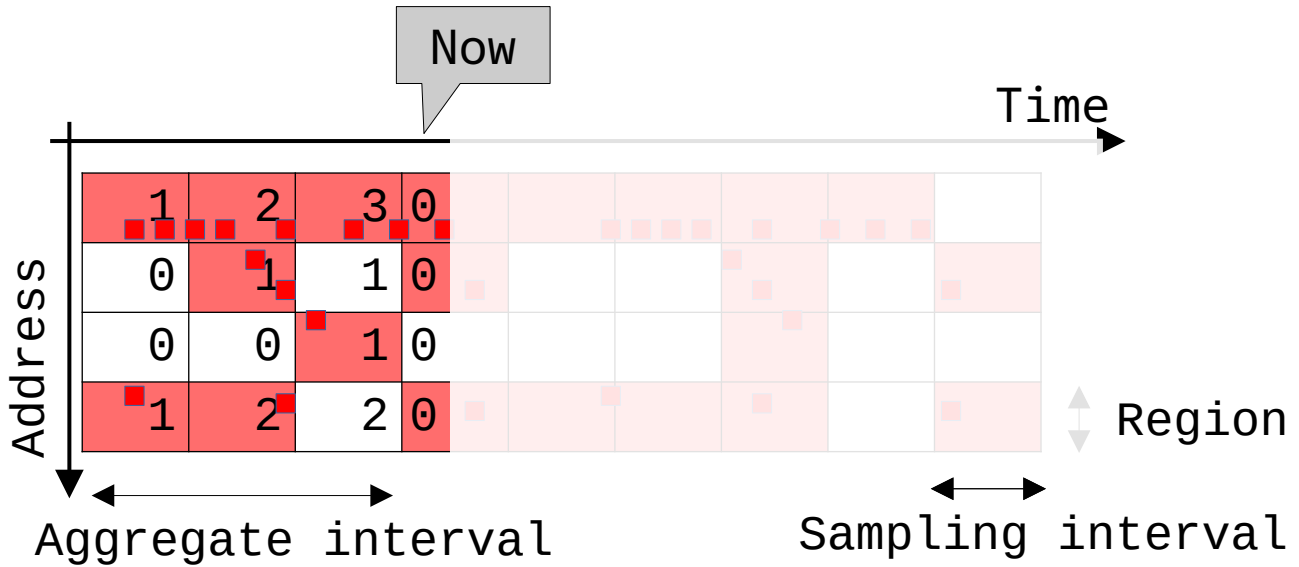
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter



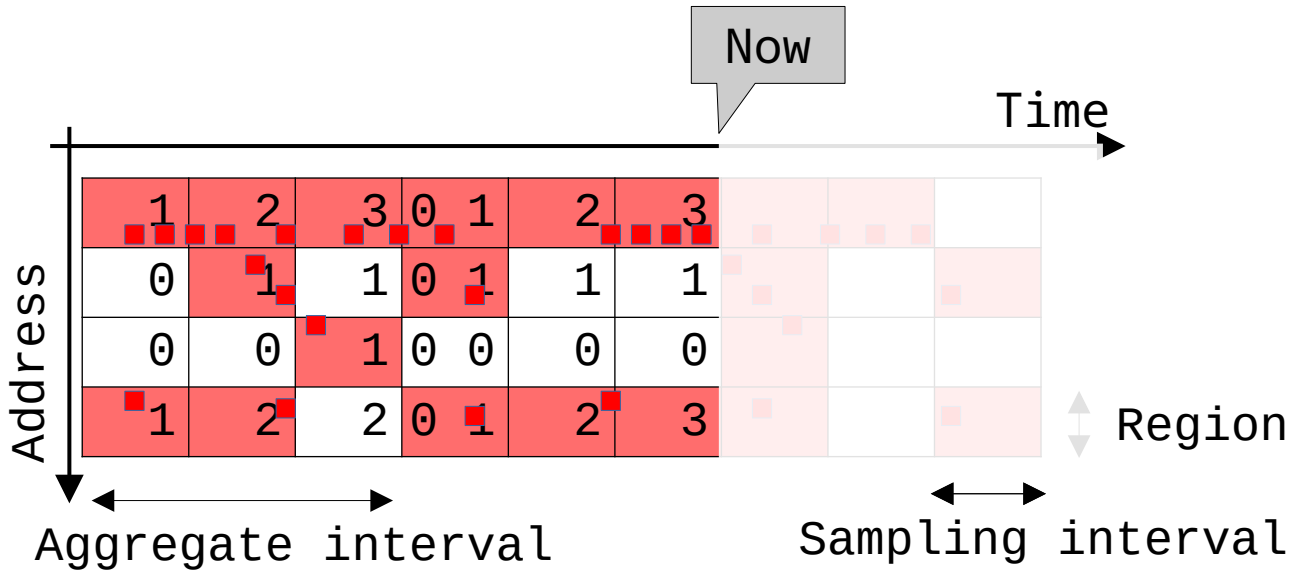
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter



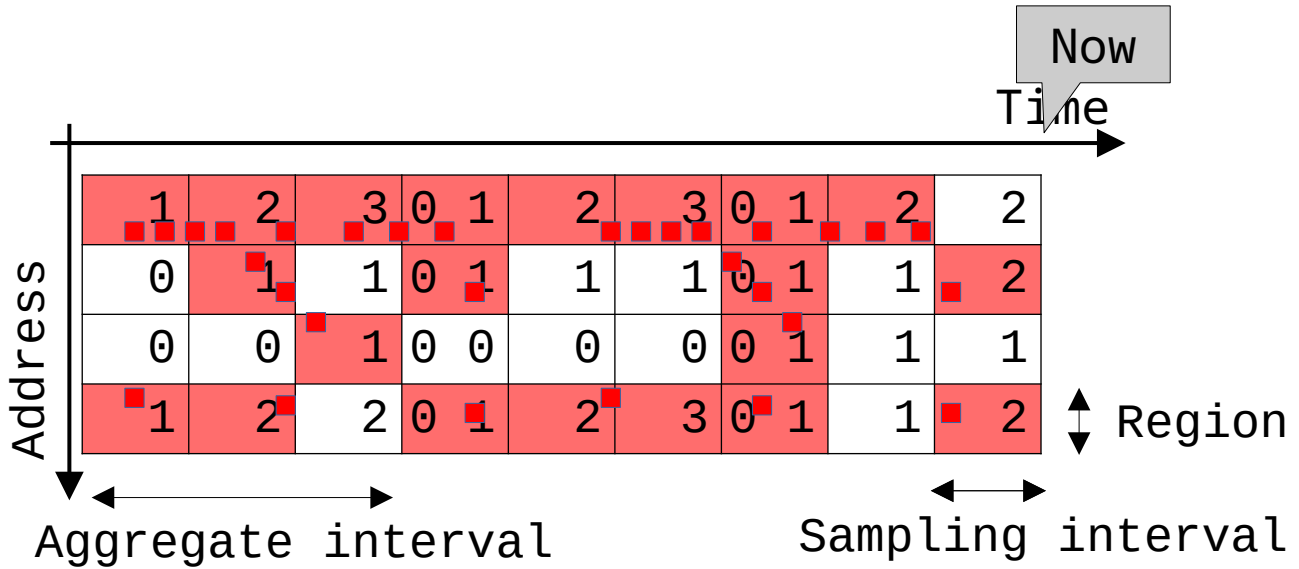
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter



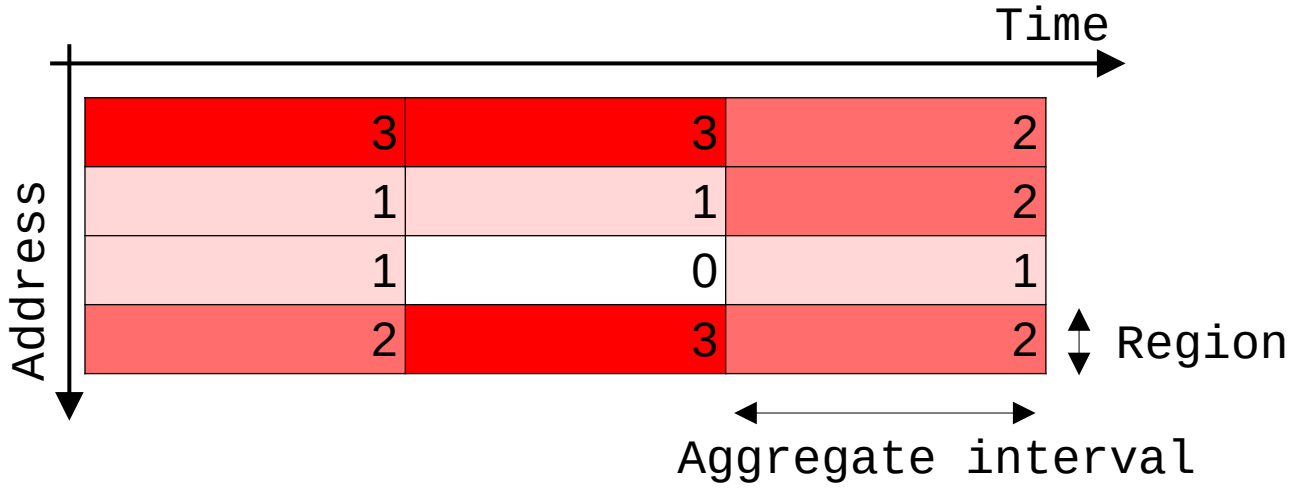
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter



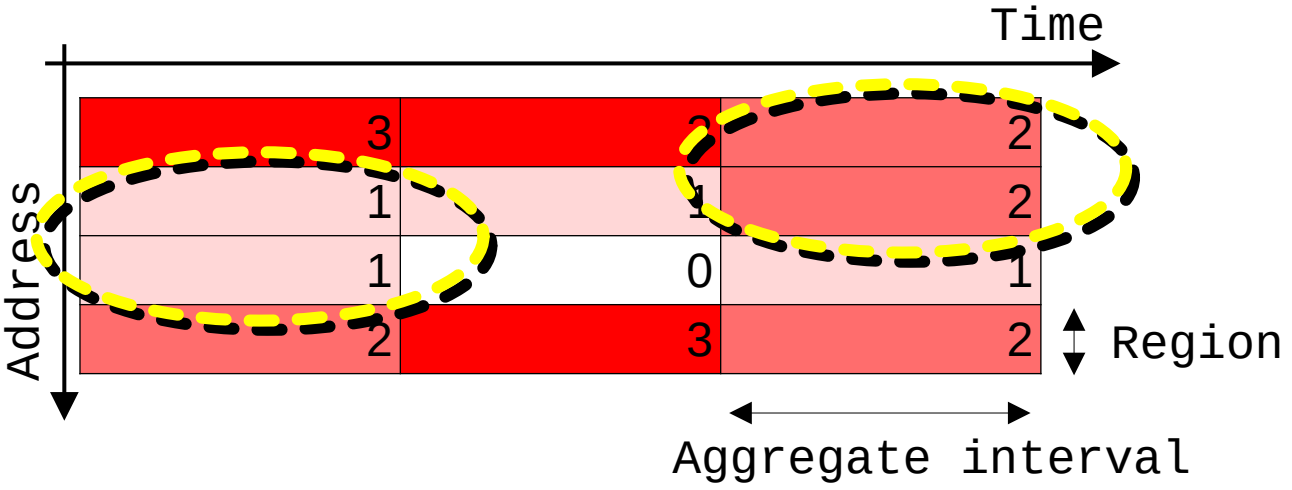
Fixed Space/Time Granularity, <=N Access Frequency

- Accumulate access checks via per-region counter
- Reduce space overhead to “1/N”
- Still, O(memory size * total monitoring time)



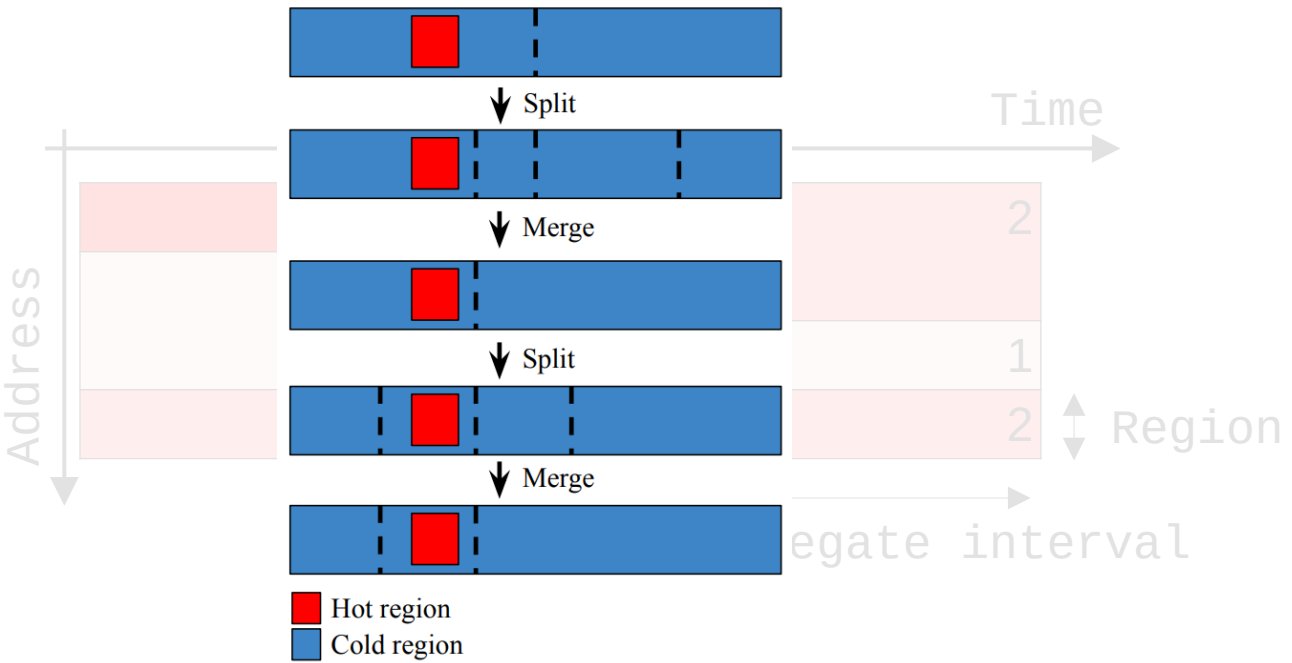
Inefficiency of Fixed Space Granularity

- Adjacent regions of similar hotness



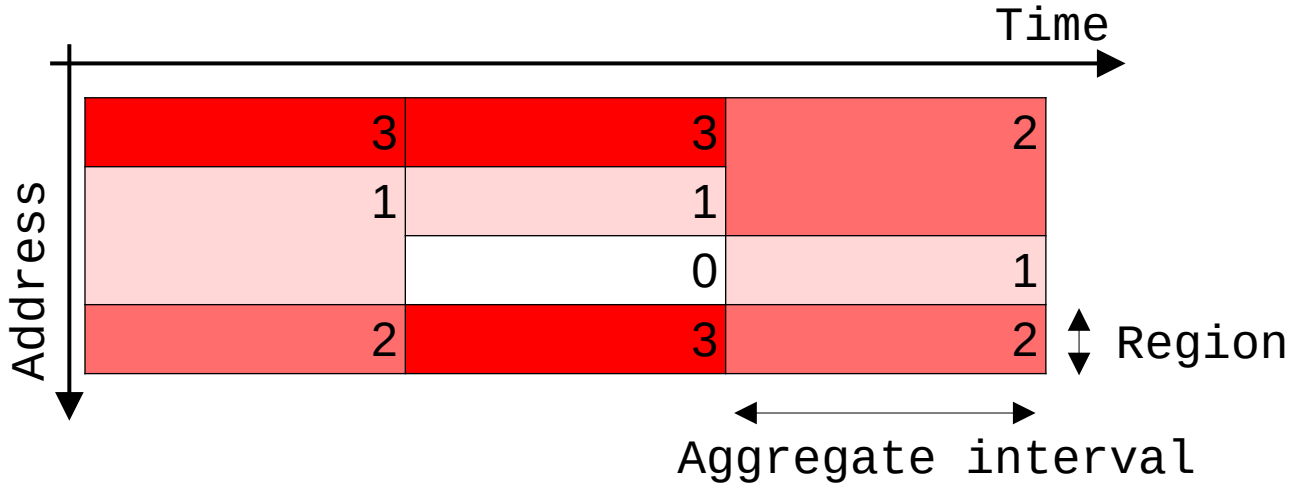
Dynamic Space Granularity: Mechanisms

- Repeat merging the wasteful regions and randomly splitting regions
  - The number of region == number of different access patterns
- Let user set min/max number of total regions



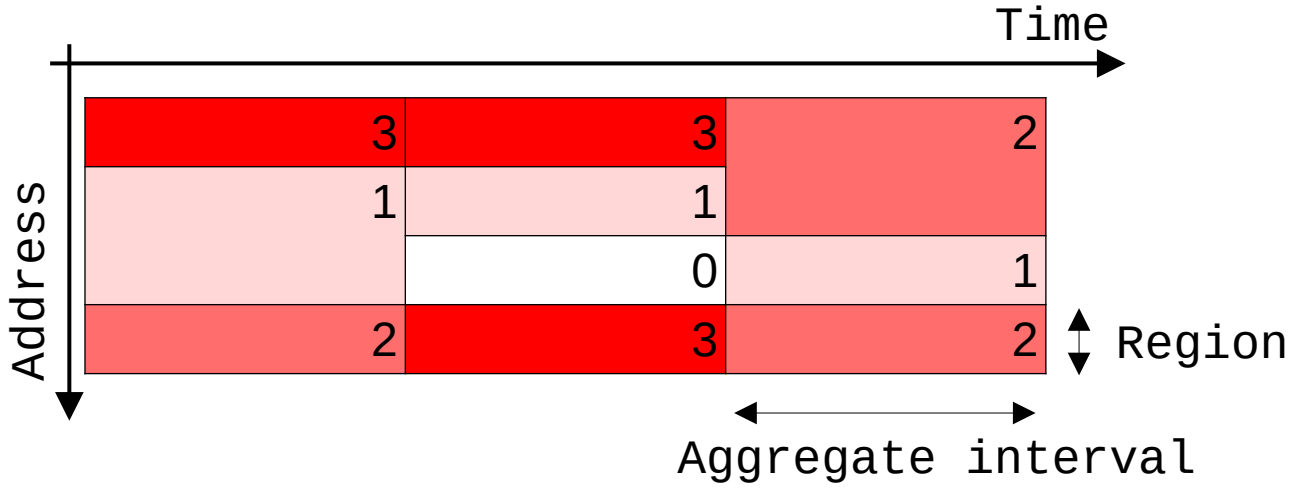
Dynamic Space Granularity: Mechanisms

- Merge the wasteful regions and randomly split region
  - The number of region == number of different access patterns
- Let user set min/max number of regions



Dynamic Space Granularity: Overhead/Accuracy

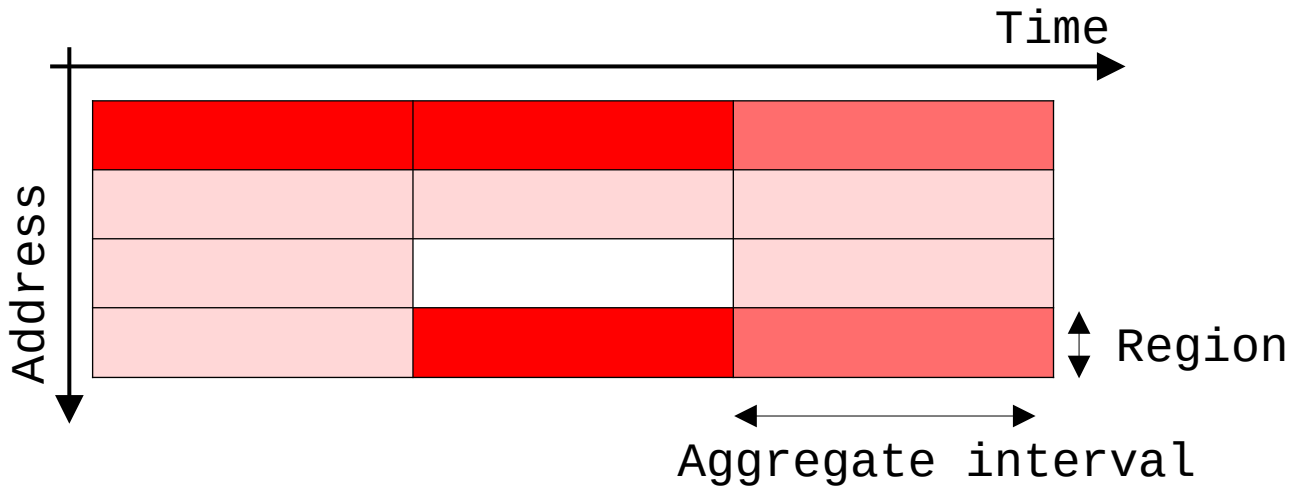
- Time overhead: min(different access patterns, max number of regions)
  - No more ruled by arbitrary memory size
- Accuracy: best-effort high
  - Lower-bound accuracy can be set via min_nr_regions



Age Counting for History Handling

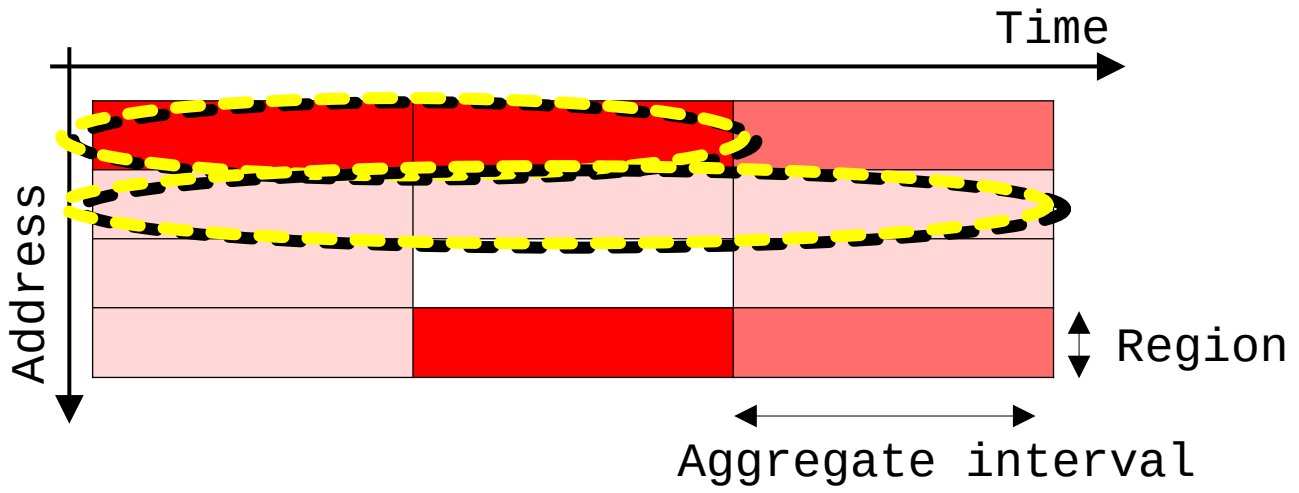
Inefficiency of Fixed Time Granularity Regions (1/2)

- The definition of regions is not only about space, but also about time



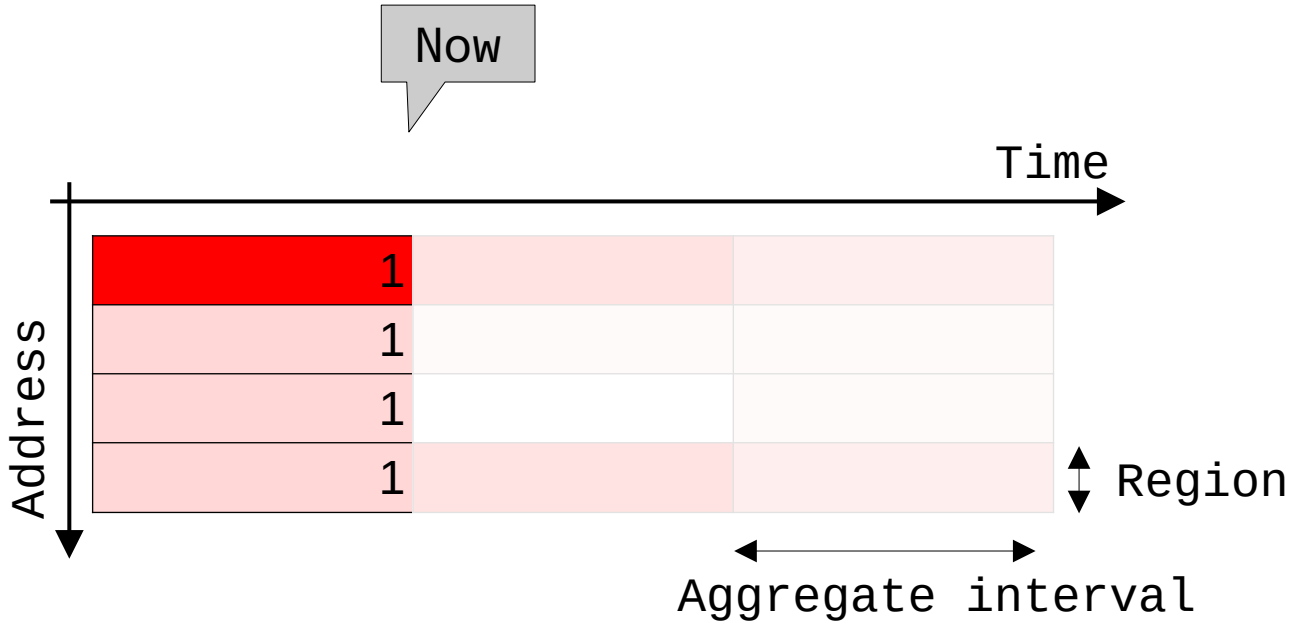
Inefficiency of Fixed Time Granularity Regions (2/2)

- The definition of regions is not only about space, but also about time
- Multiple time-adjacent regions of similar hotness: only waste



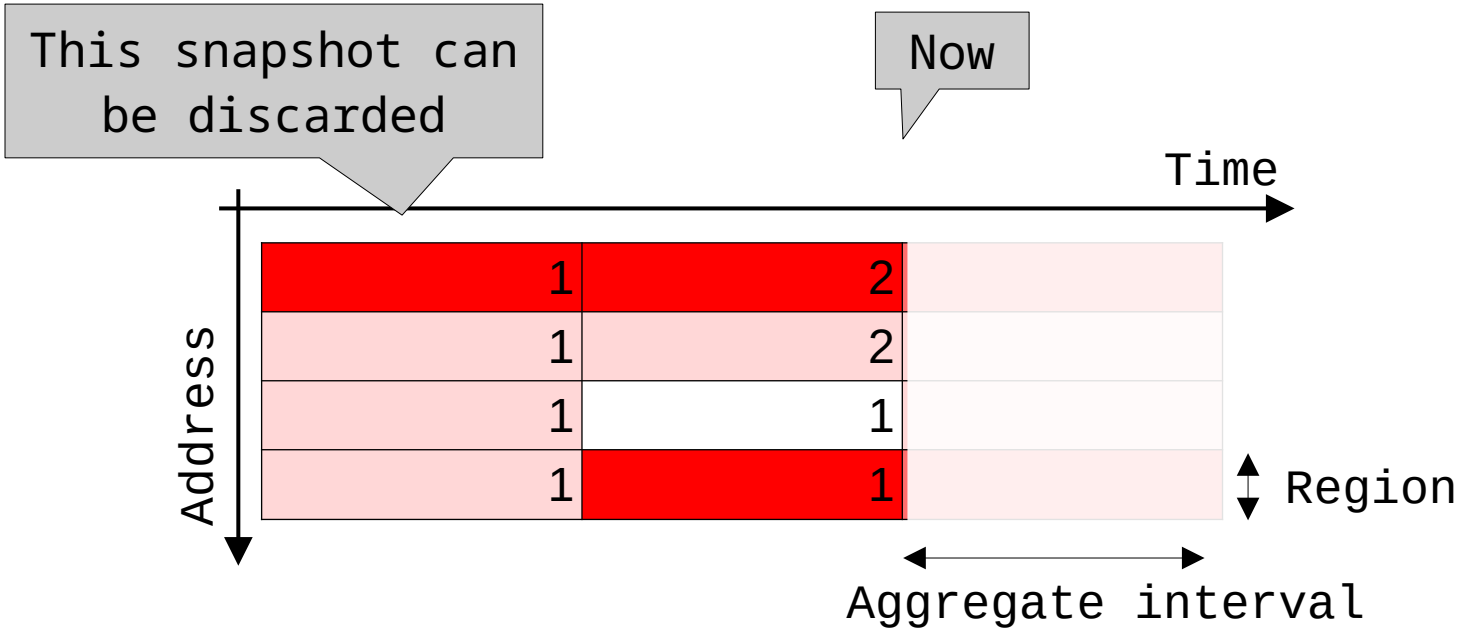
Dynamic Time Granularity (1/3)

- Count how long the hotness has kept
- Snapshot contains history of useful length



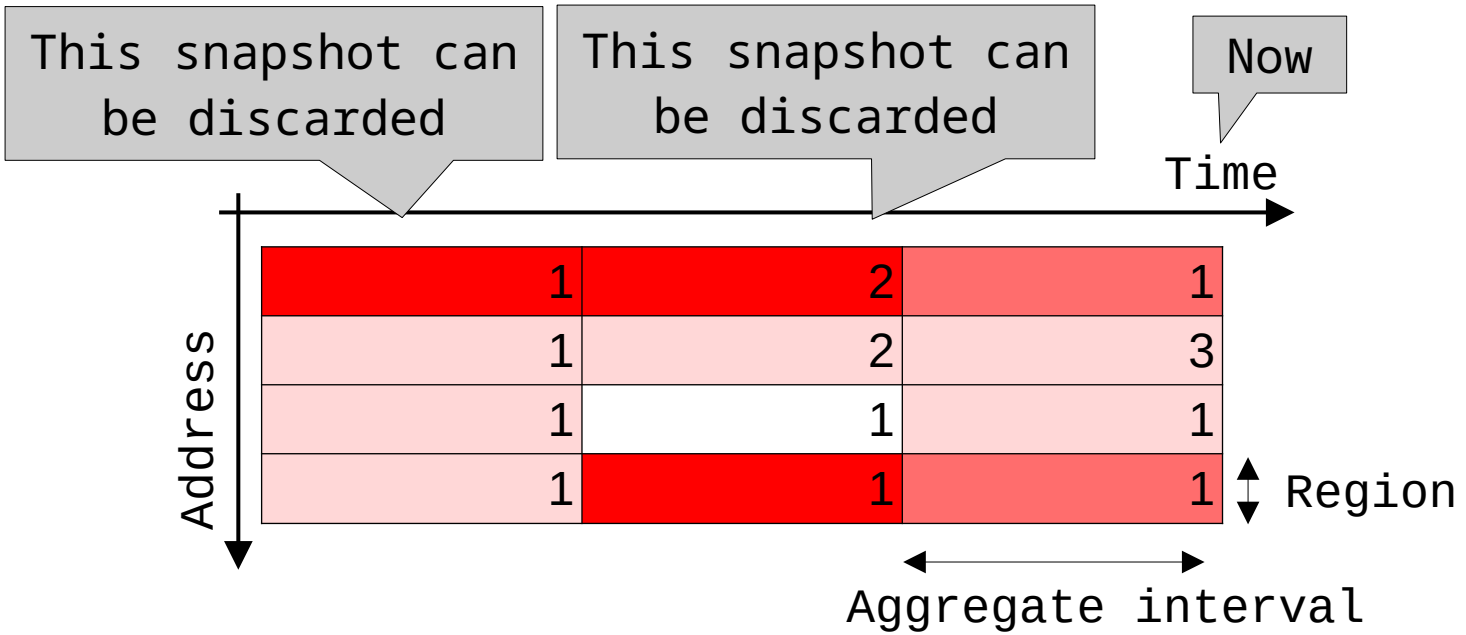
Dynamic Time Granularity (2/3)

- Count how long the hotness has kept
- Snapshot contains history of useful length



Dynamic Time Granularity (3/3)

- Count how long the hotness has kept
- Snapshot contains history of useful length



Snapshot: The Output of DAMON

- O(max_nr_regions) time/space overhead
- Both time/space overheads are not ruled by memory size/monitoring time

```
$ sudo ./damo report access --style simple-boxes
  |0000000000000000000000000000000000000000| size 53.340 MiB  access rate 0 %   age 58.300 s
|0000000000000000000000000000000000000000| size 53.020 MiB  access rate 0 %   age 1 m 16.800 s
 |0000000000000000000000000000000000000000| size 52.711 MiB  access rate 0 %   age 1 m 16.700 s
 |0000000000000000000000000000000000000000| size 53.289 MiB  access rate 0 %   age 1 m 16.500 s
 |0000000000000000000000000000000000000000| size 52.484 MiB  access rate 0 %   age 1 m 16.300 s
 |0000000000000000000000000000000000000000| size 52.887 MiB  access rate 0 %   age 1 m 15.800 s
 |0000000000000000000000000000000000000000| size 53.211 MiB  access rate 0 %   age 1 m 13.700 s
  |0000000000000000000000000000000000000000| size 52.777 MiB  access rate 0 %   age 59.200 s
            |00000000000000000000000000000| size 17.125 MiB  access rate 0 %   age 8.800 s
                                |55555555| size 8.000 KiB   access rate 60 %  age 400 ms
                     |9999999999999999999| size 7.672 MiB   access rate 100 % age 2.200 s
                     |8888888888888888888| size 1.922 MiB   access rate 95 %  age 2.200 s
              |0000000000000000000000000| size 53.121 MiB  access rate 0 %   age 7.200 s
              |0000000000000000000000000| size 23.238 MiB  access rate 0 %   age 8 s
    |0000000000000000000000000000000000000000| size 6.727 MiB   access rate 0 %   age 45.900 s
 |0000000000000000000000000000000000000000| size 124.000 KiB access rate 0 %   age 1 m 13 s
                                       |4| size 8.000 KiB   access rate 55 %  age 100 ms
total size: 533.660 MiB
```