MEGA: Overcoming Traditional Problems with OS Huge Page Management

Theodore Michailidis, Alex Delis, Mema Roussopoulos
University of Athens

Motivation

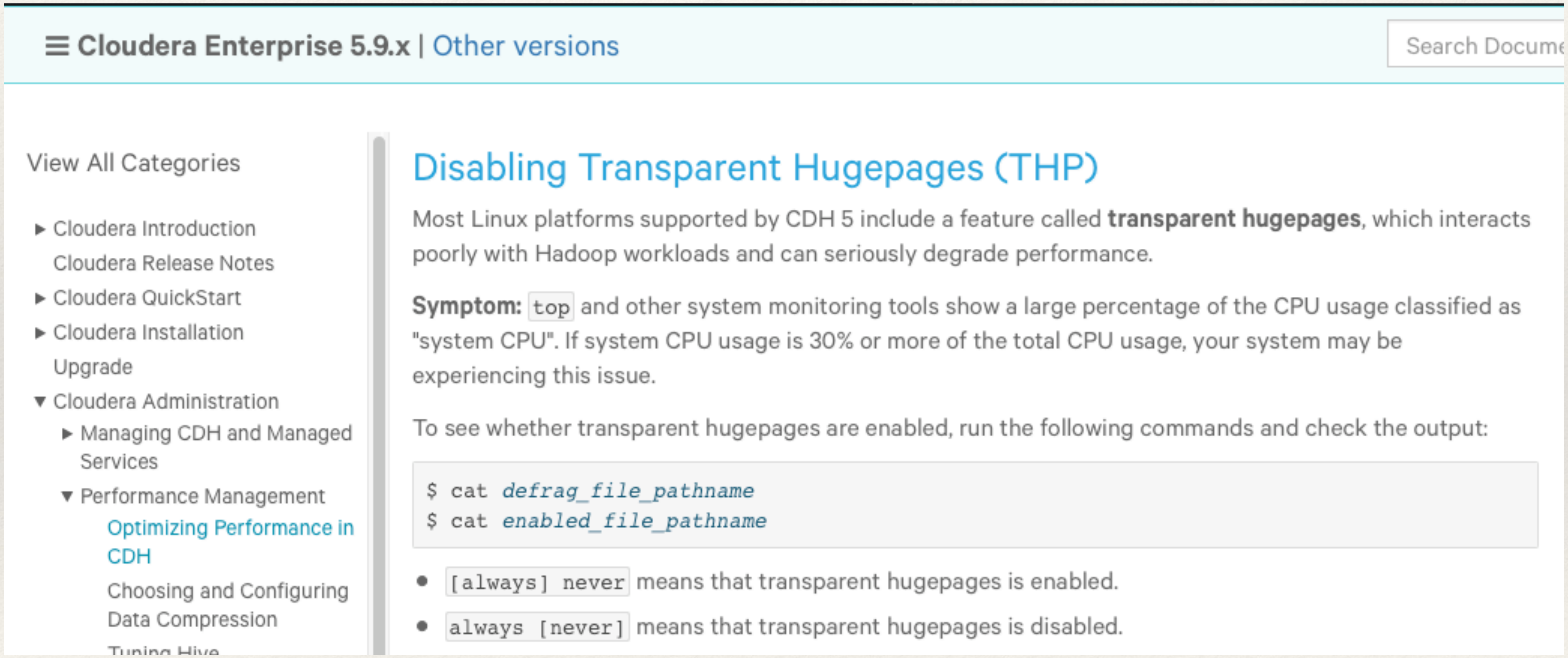
- Capacity of memory is ever growing, TLBs do not scale.
  - Problem: Increased TLB misses, cause up to 50% overhead
- Idea: **Huge pages** (usually 2MB/1GB), proposed in the 1990s.
- Until recently, TLBs had limited number of HP entries (up to 64MB)
- Since 2013, TLBs have more entries for HP (3GB)
  → Sophisticated software is needed.
- Linux has the Transparent Huge Pages (THP) feature.

Benefit from using huge pages

- Experiment: On a machine with 16GB of RAM, 2 million **set** requests with 4KB objects on **Redis** key-value store, analyze performance with **perf.**

	THP disabled	THP enabled
TLB data loads	15,172,995,558	12.162.832.618
TLB data load misses	70,996,819	315.154
TLB instruction load misses	36,694,469	87,874
TLB data store misses	9,496,490	40,932
Total cycles	30,369,768,113	14,871,159,636
Data cycles for page walking	1,358,301,181	18,422,086
Instruction cycles from page walking	656,749,586	3,645,584
Data reads from main memory for page walking	227,534,040	421,743
Instruction reads from main memory for page walking	120,997,735	465,317
Total execution time	11.722s	7.065s (-40%)

THP does not come for free


≡ Cloudera Enterprise 5.9.x | Other versions
Search Docume
View All Categories
▶ Cloudera Introduction
Cloudera Release Notes
▶ Cloudera QuickStart
▶ Cloudera Installation
Upgrade
▼ Cloudera Administration
▶ Managing CDH and Managed Services
▼ Performance Management
Optimizing Performance in CDH
Choosing and Configuring Data Compression
Tuning Hive

Disabling Transparent Hugepages (THP)

Most Linux platforms supported by CDH 5 include a feature called **transparent hugepages**, which interacts poorly with Hadoop workloads and can seriously degrade performance.

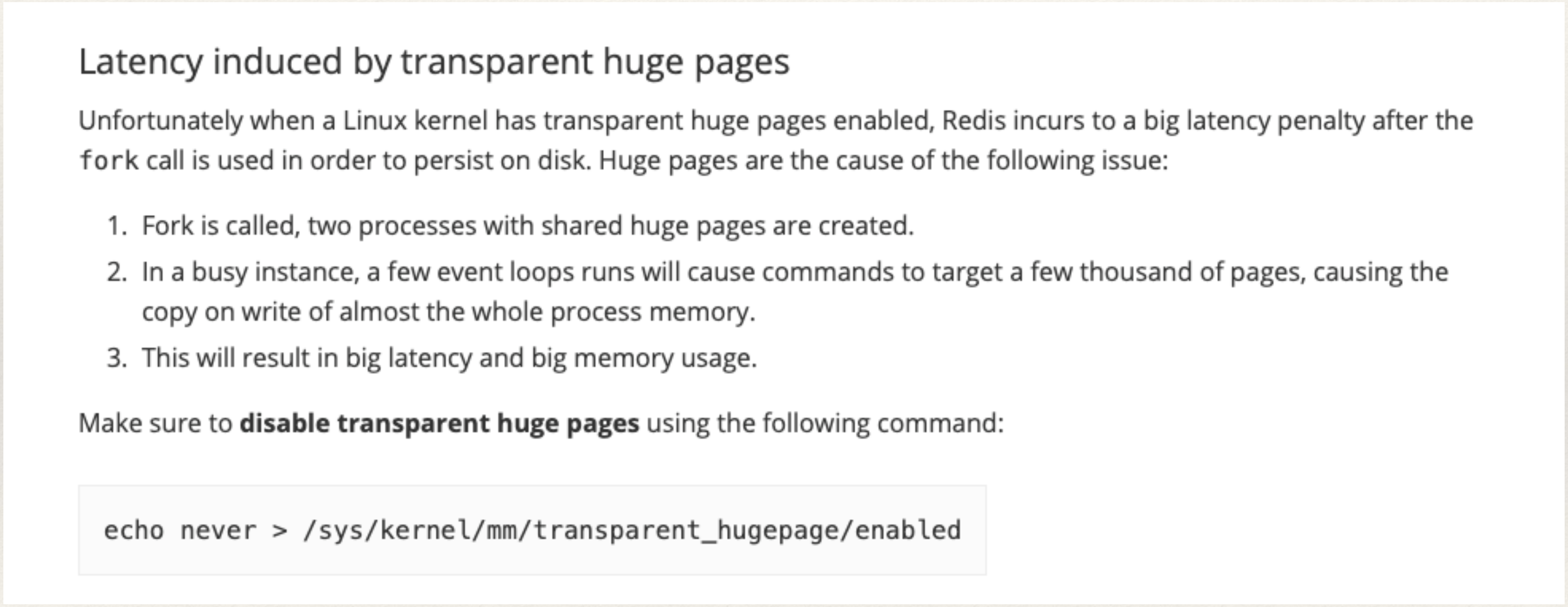
Symptom: `top` and other system monitoring tools show a large percentage of the CPU usage classified as "system CPU". If system CPU usage is 30% or more of the total CPU usage, your system may be experiencing this issue.

To see whether transparent hugepages are enabled, run the following commands and check the output:

```
$ cat defrag_file_pathname
$ cat enabled_file_pathname
```

- `[always] never` means that transparent hugepages is enabled.
- `always [never]` means that transparent hugepages is disabled.

THP does not come for free



Latency induced by transparent huge pages

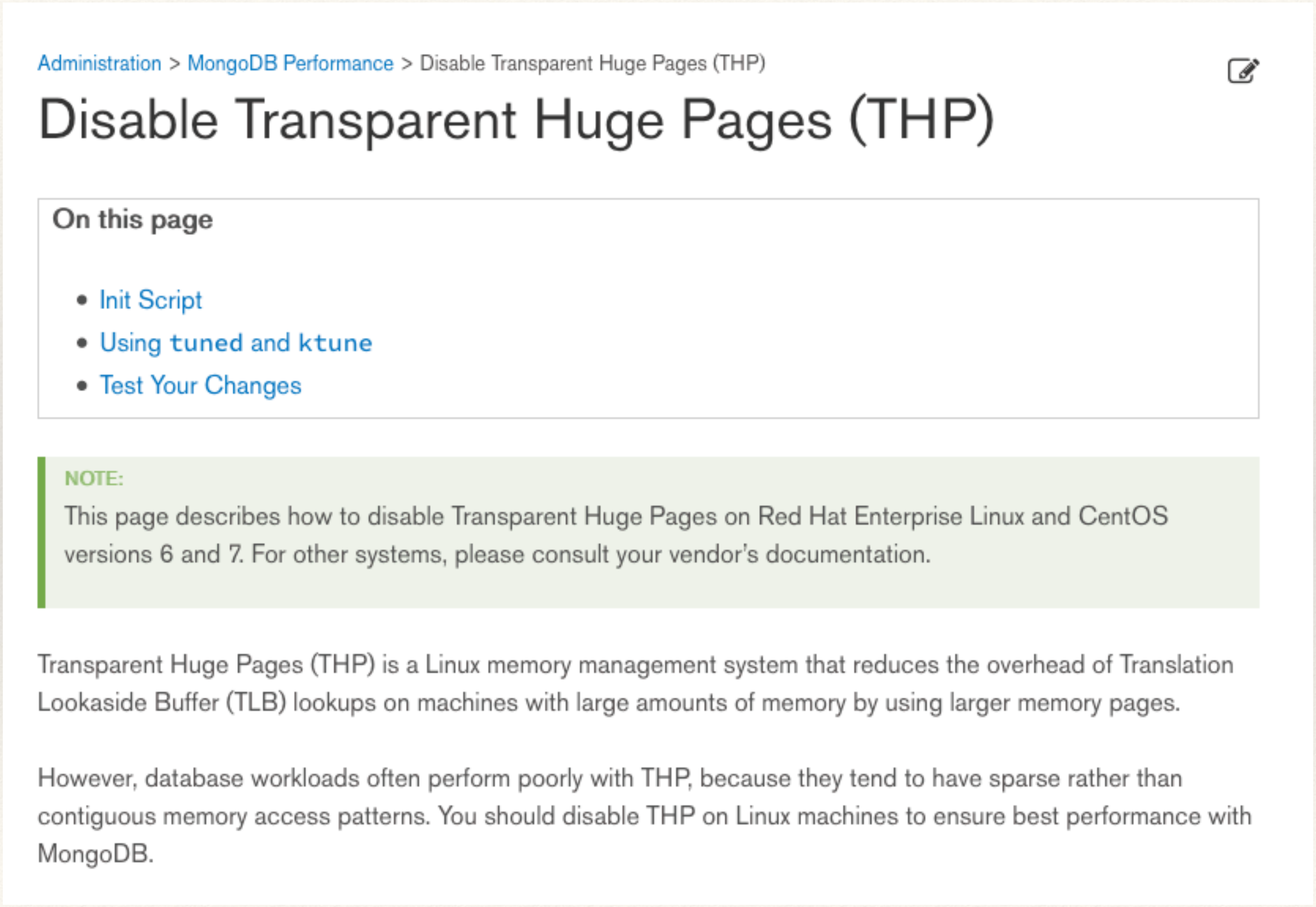
Unfortunately when a Linux kernel has transparent huge pages enabled, Redis incurs to a big latency penalty after the `fork` call is used in order to persist on disk. Huge pages are the cause of the following issue:

1. Fork is called, two processes with shared huge pages are created.
2. In a busy instance, a few event loops runs will cause commands to target a few thousand of pages, causing the copy on write of almost the whole process memory.
3. This will result in big latency and big memory usage.

Make sure to **disable transparent huge pages** using the following command:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

THP does not come for free



Administration > MongoDB Performance > Disable Transparent Huge Pages (THP)

Disable Transparent Huge Pages (THP)

On this page

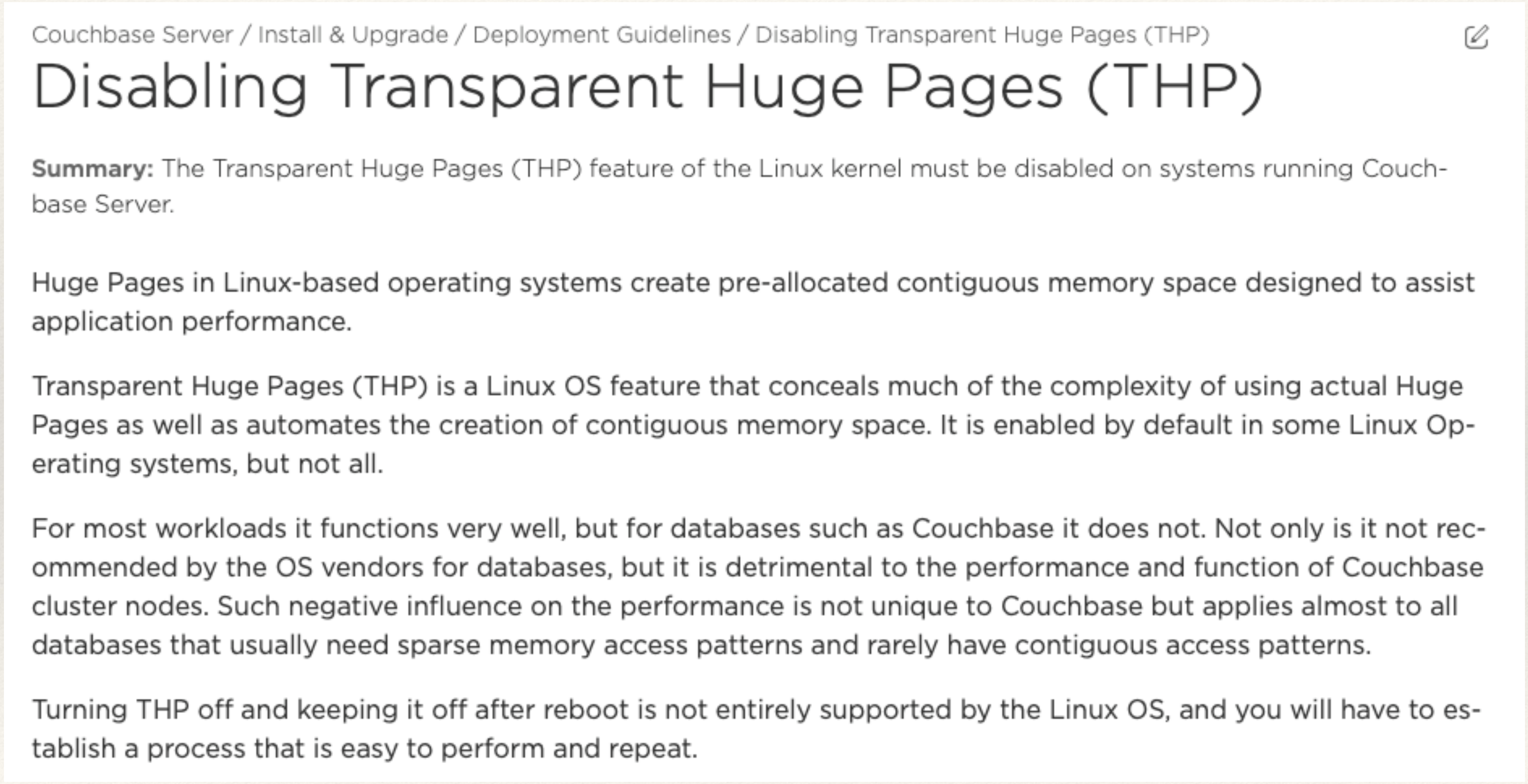
- Init Script
- Using `tuned` and `ktune`
- Test Your Changes

NOTE:
This page describes how to disable Transparent Huge Pages on Red Hat Enterprise Linux and CentOS versions 6 and 7. For other systems, please consult your vendor's documentation.

Transparent Huge Pages (THP) is a Linux memory management system that reduces the overhead of Translation Lookaside Buffer (TLB) lookups on machines with large amounts of memory by using larger memory pages.

However, database workloads often perform poorly with THP, because they tend to have sparse rather than contiguous memory access patterns. You should disable THP on Linux machines to ensure best performance with MongoDB.

THP does not come for free



Couchbase Server / Install & Upgrade / Deployment Guidelines / Disabling Transparent Huge Pages (THP)

Disabling Transparent Huge Pages (THP)

Summary: The Transparent Huge Pages (THP) feature of the Linux kernel must be disabled on systems running Couchbase Server.

Huge Pages in Linux-based operating systems create pre-allocated contiguous memory space designed to assist application performance.

Transparent Huge Pages (THP) is a Linux OS feature that conceals much of the complexity of using actual Huge Pages as well as automates the creation of contiguous memory space. It is enabled by default in some Linux Operating systems, but not all.

For most workloads it functions very well, but for databases such as Couchbase it does not. Not only is it not recommended by the OS vendors for databases, but it is detrimental to the performance and function of Couchbase cluster nodes. Such negative influence on the performance is not unique to Couchbase but applies almost to all databases that usually need sparse memory access patterns and rarely have contiguous access patterns.

Turning THP off and keeping it off after reboot is not entirely supported by the Linux OS, and you will have to establish a process that is easy to perform and repeat.

But…why?

- Current Linux kernel’s huge page management is greedy and aggressive.
- Every time a page fault occurs in a huge page region (i.e. 2MB), the kernel tries to **promote** to a huge page.
- If a small chunk of memory inside a huge page is freed, the kernel **demotes** it instantly to multiple base pages.
- Problems:
  - Promotion and demotion are **synchronous**.
  - Promotion and demotion are **costly,** mainly due to TLB invalidations.
  - Memory compaction is **synchronous**.

Problems with THP

- Increased page fault latency
- Memory bloating
- Memory fragmentation
- Huge pages are not swappable
- Huge pages are not migratable

Increased page fault latency

- Experiment: 2 million **set** requests with 4KB objects on **Redis.**
- Trace the __do_page_fault function using the **ftrace** tool.

	8GB base	8GB huge
#page faults	2,731,657	291,098
Average	0.9 µs	2.9 µs
90th	1.5 µs	1.8 µs
99th	2.8 µs	118.2 µs
99.9th	4.2 µs	123.8 µs

Memory bloating

- When a process reserves more memory than it uses, resulting in increased memory footprint.
- Experiment:
  - 2 million **hset** requests with 4KB objects in Redis.
  - remove 1.5 million objects.
  - trigger **hgetall** command.

Base pages only	Huge pages enabled
7.6 GB	11.1 GB (+ 46%)

Memory fragmentation

- Aggressive promotion to huge pages rapidly fragments memory.
- Severe memory fragmentation leads to increased page fault latency and other issues.

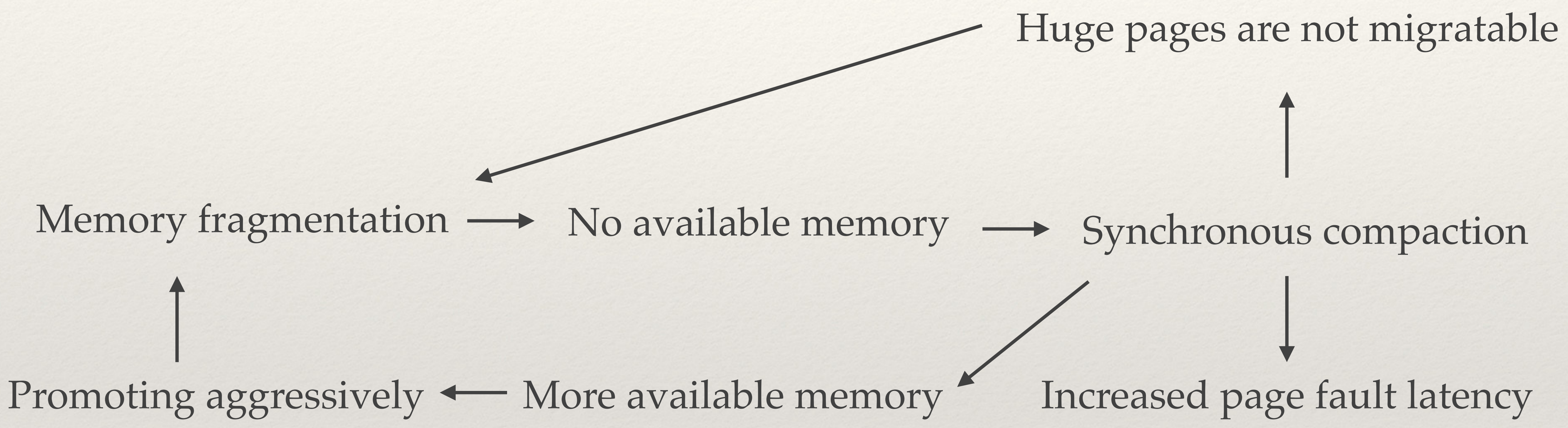
Huge pages are not swappable

- In current Linux, huge pages cannot be swapped.
  - To reclaim memory from a huge page, kernel demotes it into base pages and swaps them out.
  - When base pages are swapped in, kernel must promote them again to huge pages.

Huge pages are not migratable

- Huge pages are not moved (migrated) during the memory compaction algorithm.
- This leads to additional fragmentation.

Interconnected problems



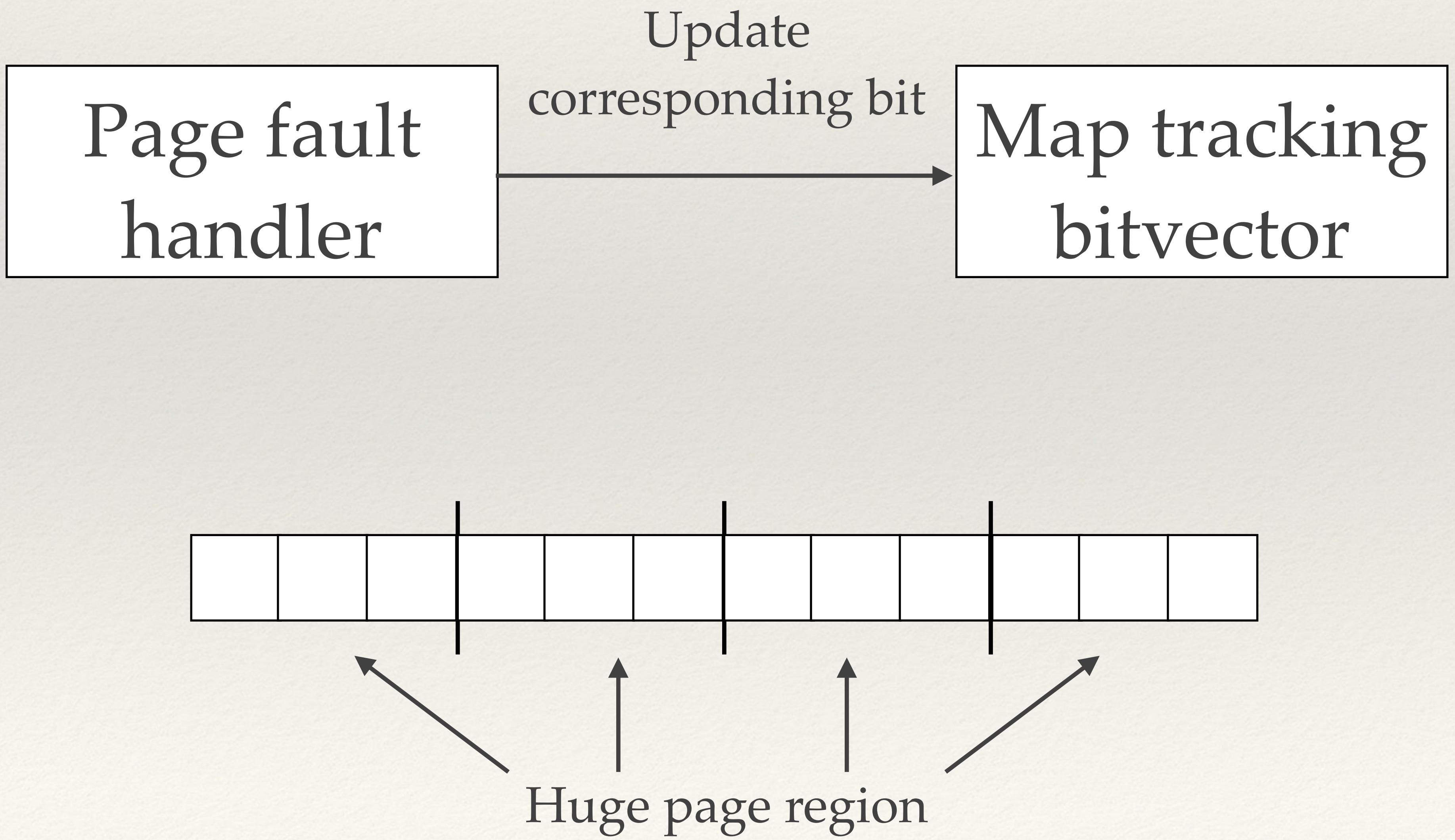
Our framework for huge page management

- MEGA: **M**anaging **E**fficiently Hu**g**e P**a**ges[1]
- MEGA manages 2MB huge pages.
- Based on the following:
  - Base pages map tracking (space).
  - Huge page region utilization tracking (time).
  - New memory compaction algorithm.

[1]Also, from the Ancient Greek word **μέγα**, which means large

Base pages map tracking in MEGA

- In page fault handler, record which base pages in which huge page region are mapped.



Page Utilization Tracking

- Idle page tracking API (since Linux kernel 4.3)
- Associated **idle** flag (in software) with **access** bit (in hardware)
  - Set the idle flag (and clear the access bit).
  - Wait for some predefined time for the page to be accessed.
  - Check the idle flag.
- Setting the **access** bit clears the **idle** flag.
- Clearing the **access** bit causes a TLB invalidation.

Huge page region utilization tracking in MEGA

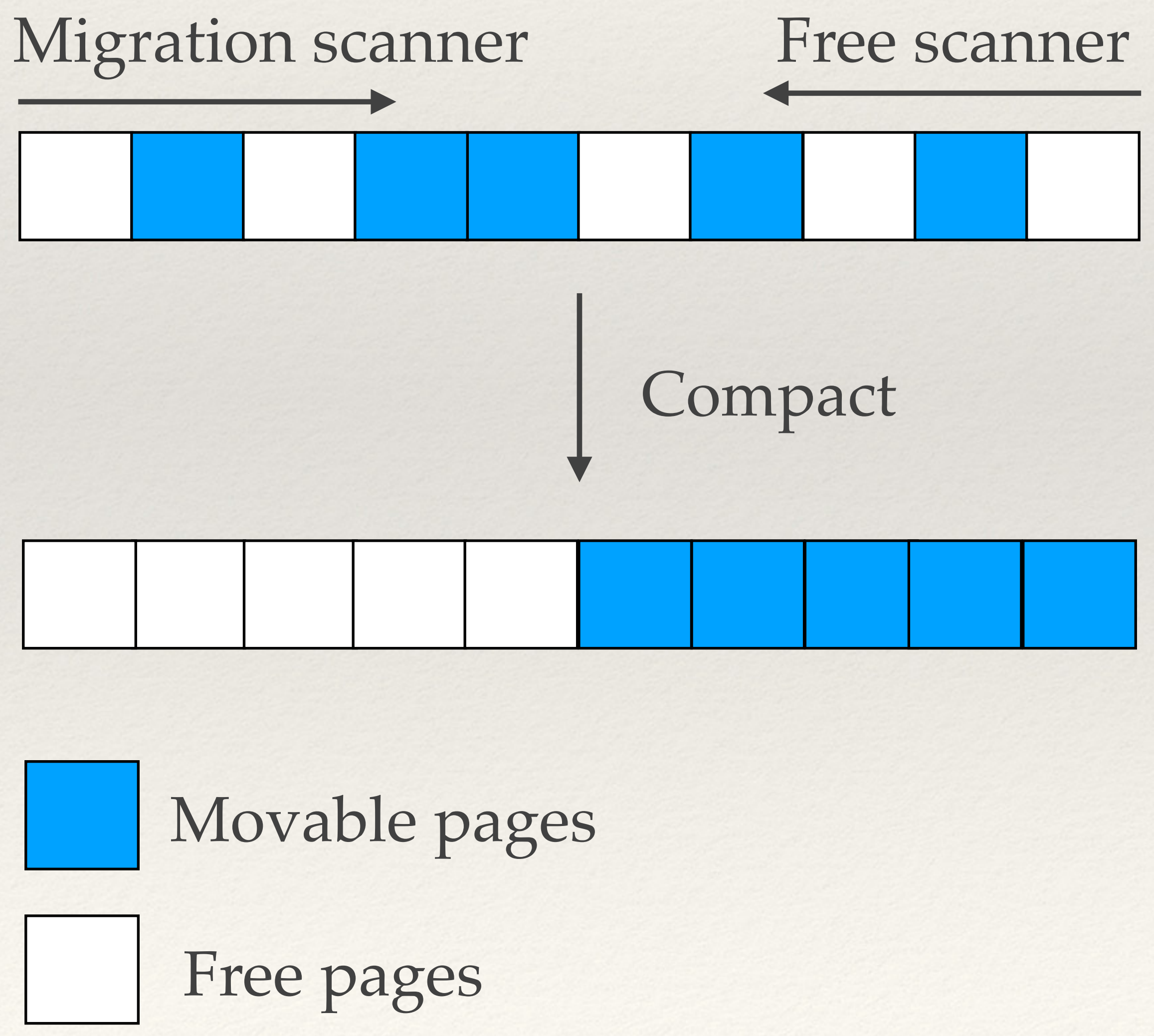
- Periodically **scan** to track pages' utilization, and **store** last 10 utilization numbers (utilization history buffer).
- Due to high cost (TLB invalidation):
  - Only track huge page regions with **50%** base pages mapped.
  - If %mapped base pages drops under **25%**, stop tracking utilization of huge page region.

Asynchronous promotion/demotion in MEGA

- **Promote**, when #base_pages_mapped > **90% and** utilization > **50%**.
- **Demote**, when #base_pages_mapped < **50% or** utilization < **25%**.
- Thresholds chosen to reduce memory bloating and frequent promotions and demotions.

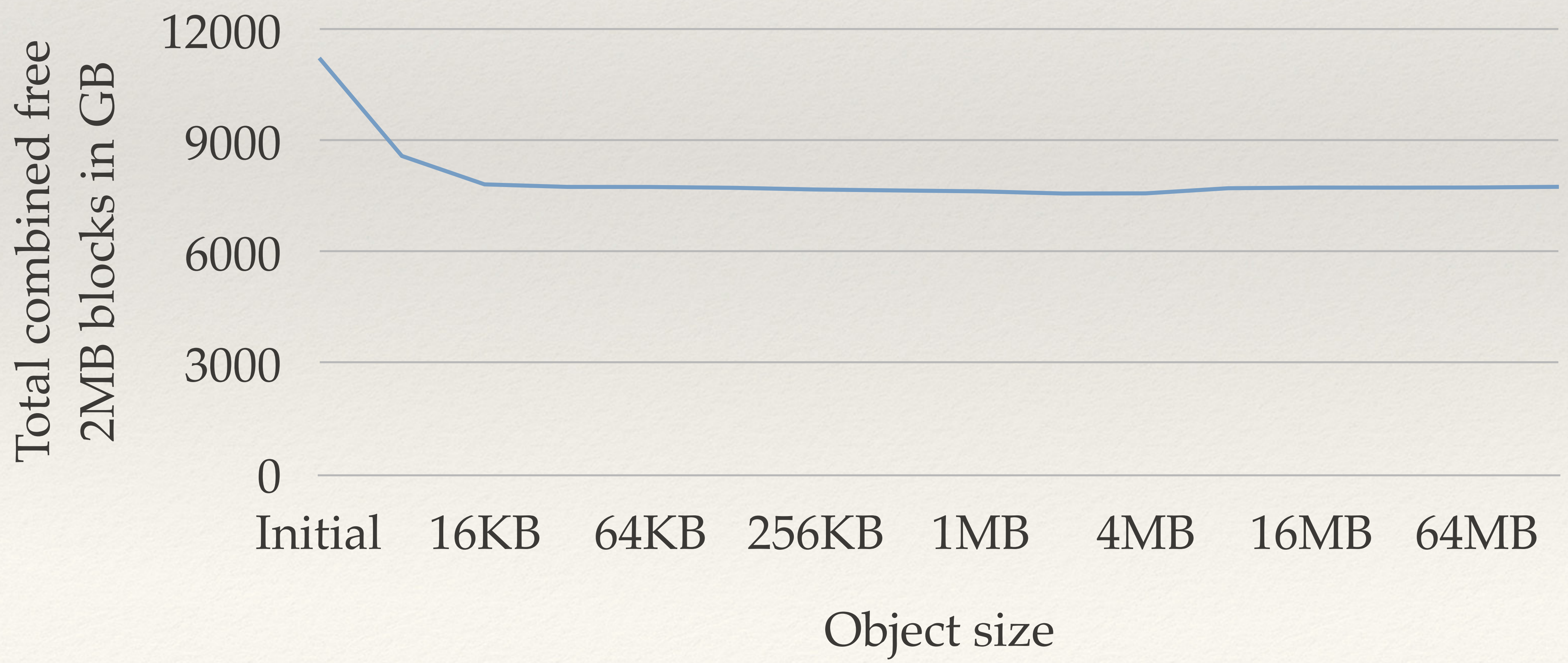
Linux memory compaction algorithm

- Scan ⟶ Compact/Migrate



Linux memory compaction algorithm

- Current memory compaction done when it is too late.
- After compaction, memory does not fully recover.
- Experiment: Continuously allocate/free 10GB of memory and record total free 2MB blocks after the memory is freed.



Memory compaction in MEGA

- Prioritize **physical** huge page regions that:
  - **Are “cold”/utilized less** (less interference).
  - **Have fewer base pages mapped**
    - Less costly to move.
    - Easier to find free space to move → reduces the risk of failed migration.
  - **Are “older”, in terms of mapping.** Newly created data (memory) is more likely to “die” (be freed) in the near future.

Memory compaction in MEGA

- Cost-benefit approach used for segment cleaning in LFS.
- Proactive compaction of up to 200MB of memory, to avoid high compaction costs.

$$\frac{benefit}{cost} = \frac{age * (1 - \%bpagesMapped) * (1 - \%bpagesAccessed)}{(2 * \%bpagesMapped)}$$

Evaluation

- 16GB DDR3 RAM
- 500GB SSD
- Intel i7 2.3GHz
  - L1 Data 32KB
  - L1 Instruction 32KB
  - Shared L2 256KB
  - Shared L3 6MB

Evaluation

- Compare MEGA, Linux kernel 4.16.8 and Ingens [Kwon, 2016], the state-of-the-art framework for huge pages.
- Our evaluation includes experiments for:
  - Page fault latency
  - Utilization based promotion/demotion
  - Memory compaction
  - Performance impact for compute-intensive workloads
  - Big-memory workloads

Ingens

- **Promotes** a huge page region if #base_pages_mapped > **90%** and **demotes** a huge page if any number of base pages are freed within it.
- Checks the utilization of a process' previously allocated huge pages, to determine if it will "get" another huge page (fairness).
- Periodically compacts 100MB of memory, using the default memory compaction algorithm.

Evaluation - Page fault latency

- 2 million **set** requests with 4KB objects on **Redis.**

Latency	Linux 4.16.8 THP disabled	Linux 4.16.8 THP enabled	Ingens	MEGA
Average	0.9 μs	2.9 μs (x3.22)	1.6 μs (x1.78)	2.5 μs (x2.78)
90th	1.5 μs	1.8 μs (x1.2)	1.7 μs (x1.13)	3.1 μs (x2.06)
99th	2.8 μs	118.2 μs (x42.21)	4.5 μs (x1.6)	6.1 μs (x2.17)
99.9th	4.2 μs	123.8 μs (x29.46)	400.8 μs (x95.42)	15.1 μs (x3.59)

Evaluation - Utilization based promotion/demotion

- Allocate 8GB, iterate over it, then free it.
- We do this 50 times and measure the total execution time in seconds.

	Total execution time in seconds
Ingens	47.614s (+59%)
MEGA	29.98s

Evaluation - Utilization based promotion/demotion

- We demonstrate an extreme case: Allocate 6GB of memory, iterate over it with step 32 * 1024 (L1 data cache size).
- We do this 10 times and measure the total execution time in seconds.
- In MEGA, the utilization is not high enough to exceed the threshold.

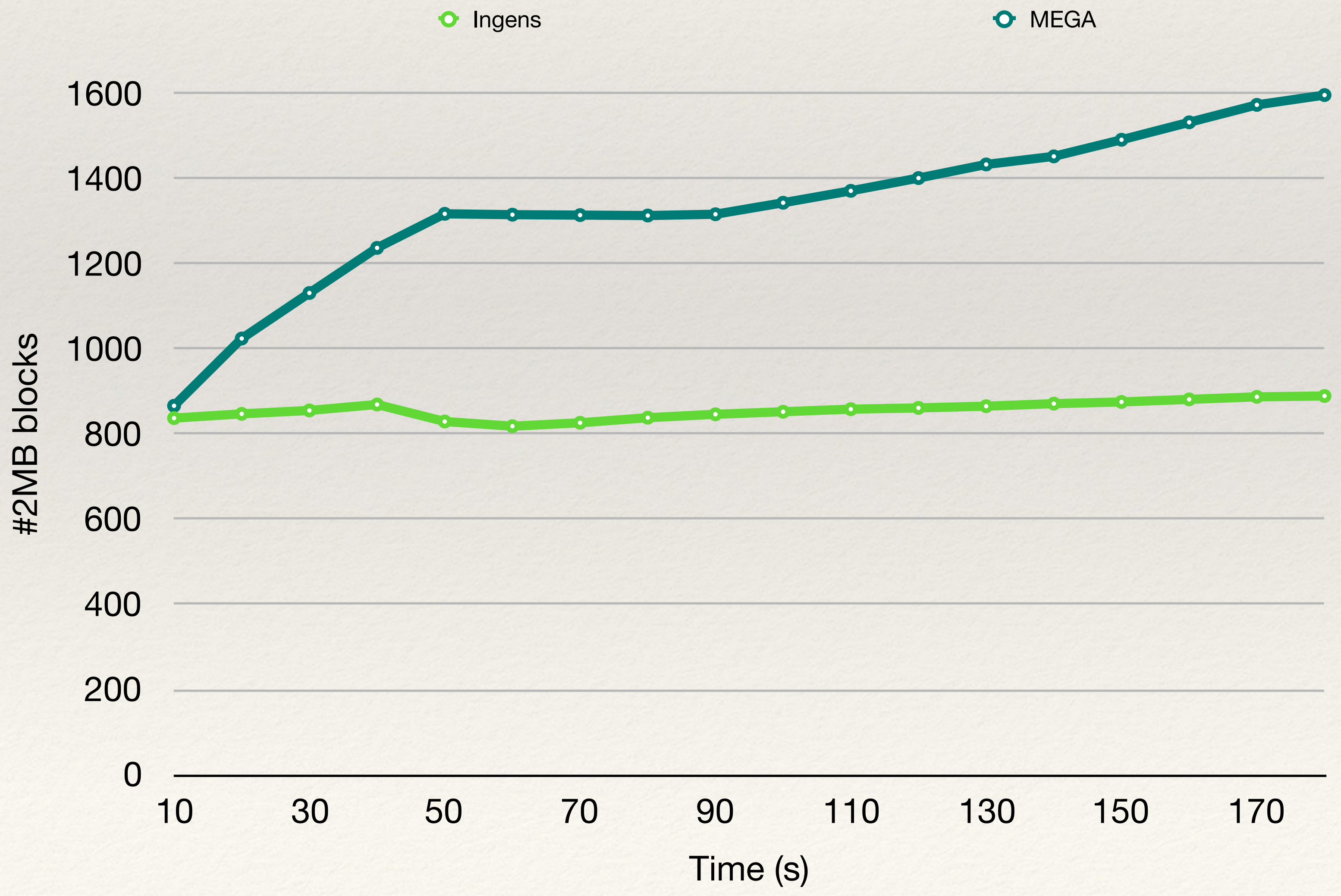
	Total execution time in seconds
Ingens	158.78s
MEGA	324.11s

Evaluation - Memory compaction

- We allocate 12GB of memory and iterate once through it.
- Free 50% of allocated memory in chunks of 1MB.
- We run this experiment for 2 minutes and then observe in the next 1 minute how fast the system restores 2MB blocks.
- We record the number of 2MB blocks available throughout the 3 minutes.
- Increase the compaction limit in Ingens to 200MB (every 5 seconds).

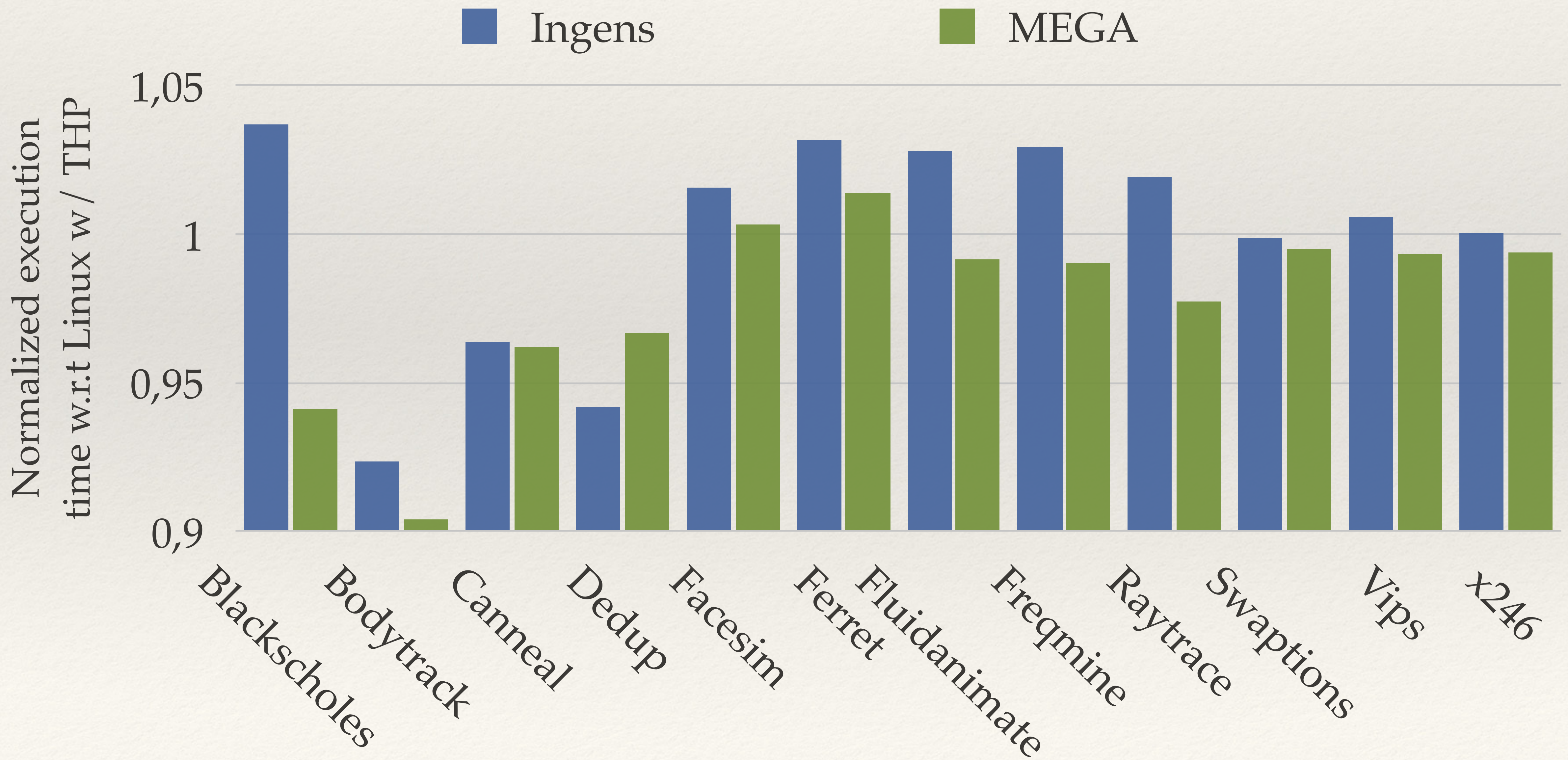
Evaluation - Memory compaction

- MEGA recovers faster and has nearly 2x the number of 2MB available blocks Ingens has.
- MEGA has 5x Ingens' #successfully migrated pages (7,352 vs 1,432)
- Ingens has a small decline in 2MB blocks (at 40s).



Evaluation - Performance Impact

- Measure performance impact of MEGA on compute-intensive workloads (PARSEC 3.0 benchmark suite).



Evaluation - Big-Memory Workloads

- 256GB DDR4 RAM, 2.8TB of SATA 3 SSD, Intel Xeon Processor E5-2650 v4 @2.2GHz
- 2 workloads:
    - We run a workload that allocates/manipulates/frees at least 100GB of memory at a time.
    - Then, we run **redis-benchmark** and measures latency, throughput and the number of free 2MB blocks.

Evaluation - Big-Memory Workloads

Step 1

$$\begin{tikzcd}
\text{Allocate of 200GB in 1MB chunks} \arrow[r] & \text{Free 100GB} \arrow[d] \\
& \text{Allocate 100GB and free the rest (100GB) from the previous iteration} \arrow[loop right, "\text{X5}"]
\end{tikzcd}$$

Result: The memory becomes fragmented, simulating cloud systems that run real client workloads.

Step 2

Run **redis-benchmark** with 40000 **set** operations with randomly selected 12-byte sized keys, values of 2MB and 50 parallel clients.

Evaluation - Big-Memory Workloads

Stats	Ingens	MEGA
Throughput (req/s)	501.54	638.05
99th latency	278ms	104ms
99.9th latency	421ms	260ms
99.99th latency	505ms	266ms
Execution Time	79.75s (+27%)	62.69s

Memory block size	Ingens	MEGA
#2MB	42	14
#4MB	58	1146
Total available 2MB blocks	158	2306

- Note: increasing the number of **set** operations in Ingens, causes the first workload to be killed due to extreme memory starvation.

Summary

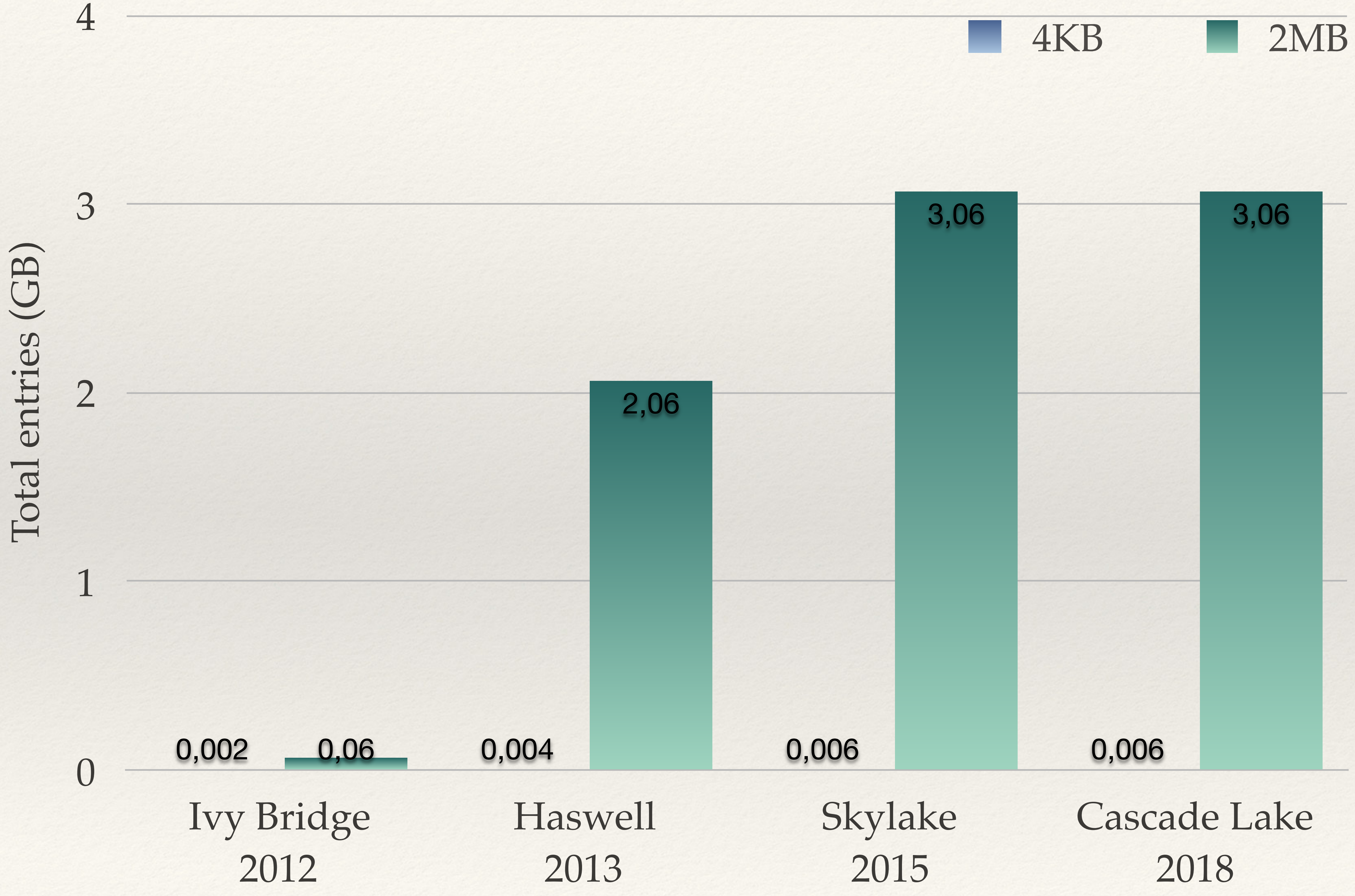
- MEGA combines spatial and temporal tracking to make decisions about promotions/demotions.
  - Utilization tracking is critical for system and workload performance.
- MEGA compaction algorithm moves old, cold and partially used physical huge page regions.
  - Achieves better memory state than Linux or Ingens and minimizes workload interference.

Thank you!

Questions?

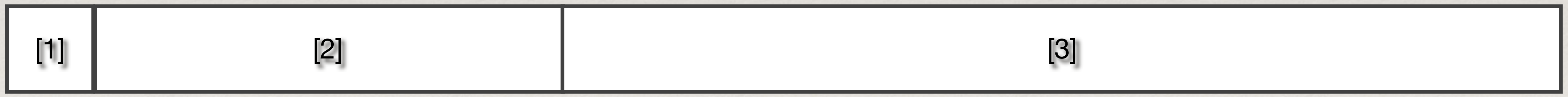
Backup Slides

TLB entries coverage



Memory compaction algorithm

- Linux in x86_64 divides physical memory in 3 zones:
  - [1] **ZONE_DMA** (0 - 16MB). Primarily for devices that can DMA only into 24-bit addresses.
  - [2] **ZONE_DMA32 (**16MB - 4GB). Primarily for devices that can DMA only into 32-bit addresses
  - [3] **ZONE_NORMAL (**4GB - End of memory). Contains normal addressable pages.
- The kernel tries to satisfy user-level memory requests from ZONE_NORMAL, and if there is no available memory, it tries to allocate memory from ZONE_DMA32.



Evaluation - Utilization based promotion/demotion

- Workload that allocates 6GB of memory and uses only 1GB.
- Run concurrently 2 instances of this workload to put pressure in the system.
- Record number of THP used and number of 2MB blocks of memory before and after **one** minute of execution.
- Ingens tries to allocate 3,967 more huge pages, but the memory is too fragmented.

	Ingens	MEGA
#THP used	2,447	1,024
#2MB blocks before execution	6,710	6,733
#2MB blocks after one minute of execution	50	282

Evaluation - Performance Impact

- Measuring the performance and latency of MySQL using the **sysbench** benchmark suite.
- Run for 1 minute a read-only workload on a table with 30 million rows, executed by 8 threads
- Ingens experiences the biggest average, max and 95th latency and the lowest transaction throughput
- MEGA's number of transactions per second is close to the number of transactions per second that Linux with huge pages achieves, while keeping the latency at low levels.

	MEGA	Ingens	Linux base pages	Linux huge pages
Min	0,76 ms	0,68 ms	0,74 ms	0,85 ms
Average	1,44 ms	1,71 ms	1,63 ms	1,32 ms
Max	54,01 ms	108,12 ms	11,66 ms	64,94 ms
95th percentile	2,18 ms	3,62 ms	3,07 ms	1,76 ms
Transactions per second	5556,29	4661,36	4895,21	6056,84