adaptTo()

APACHE SLING & FRIENDS TECH MEETUP
BERLIN, 25-27 SEPTEMBER 2017

TarMK: Facts and Figures

Michael Dürig / Valentin Olteanu, Adobe

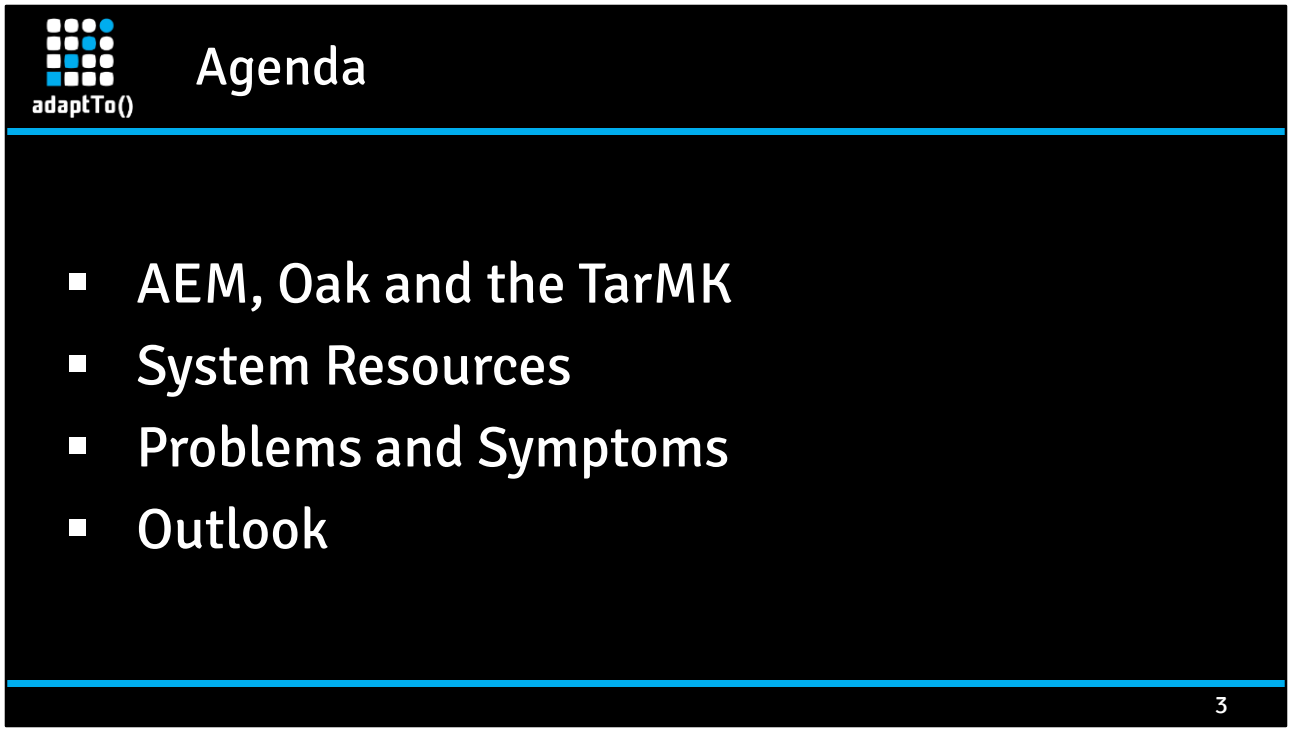
Understanding how the TarMK uses system resources like RAM, CPU, disk space and IO is crucial for effective deployments and operations. This session focuses on the capabilities and characteristics of the TarMK by presenting the relation between content, load and consumed resources. We show how TarMK behaves under various operation conditions, illustrate with data from our internal testing and explain how the numbers can be interpreted. Moreover, we examine the impact and effect of running online revision garbage collection and explain how to monitor, detect

and recover from anomalies.

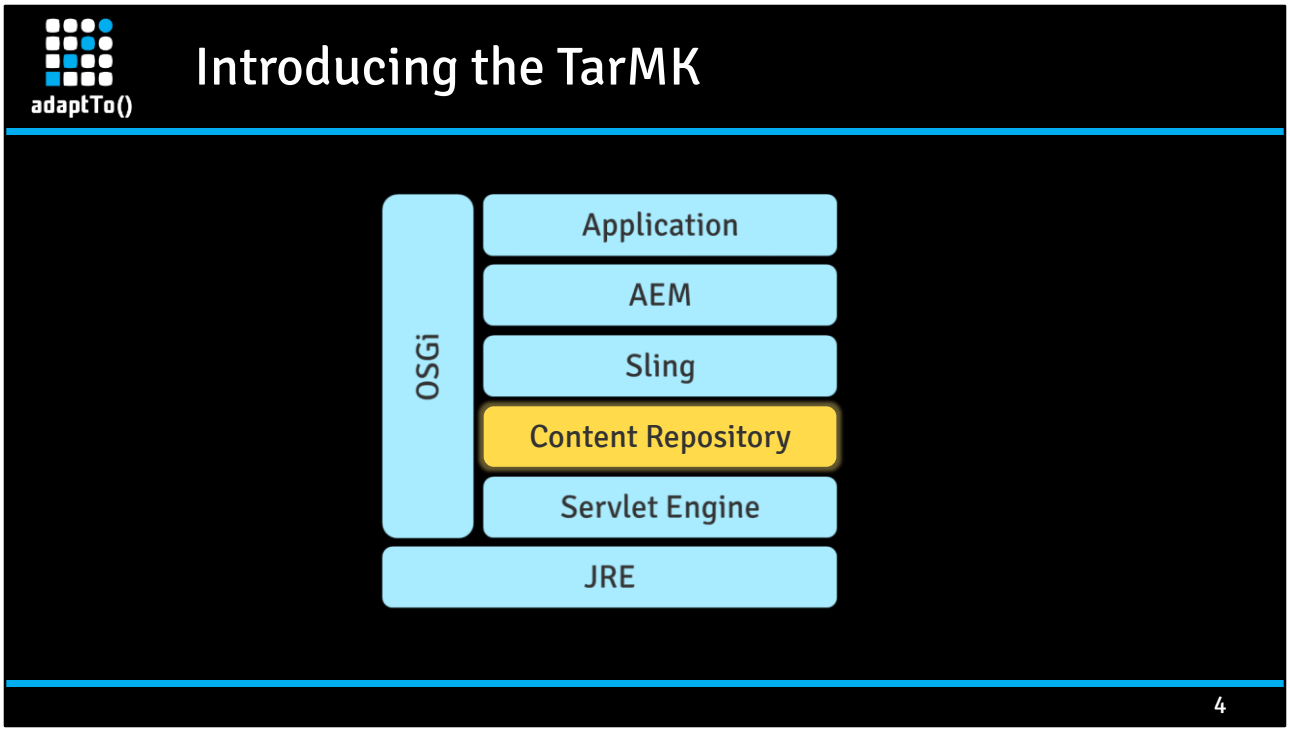
adaptTo()

Slooooow

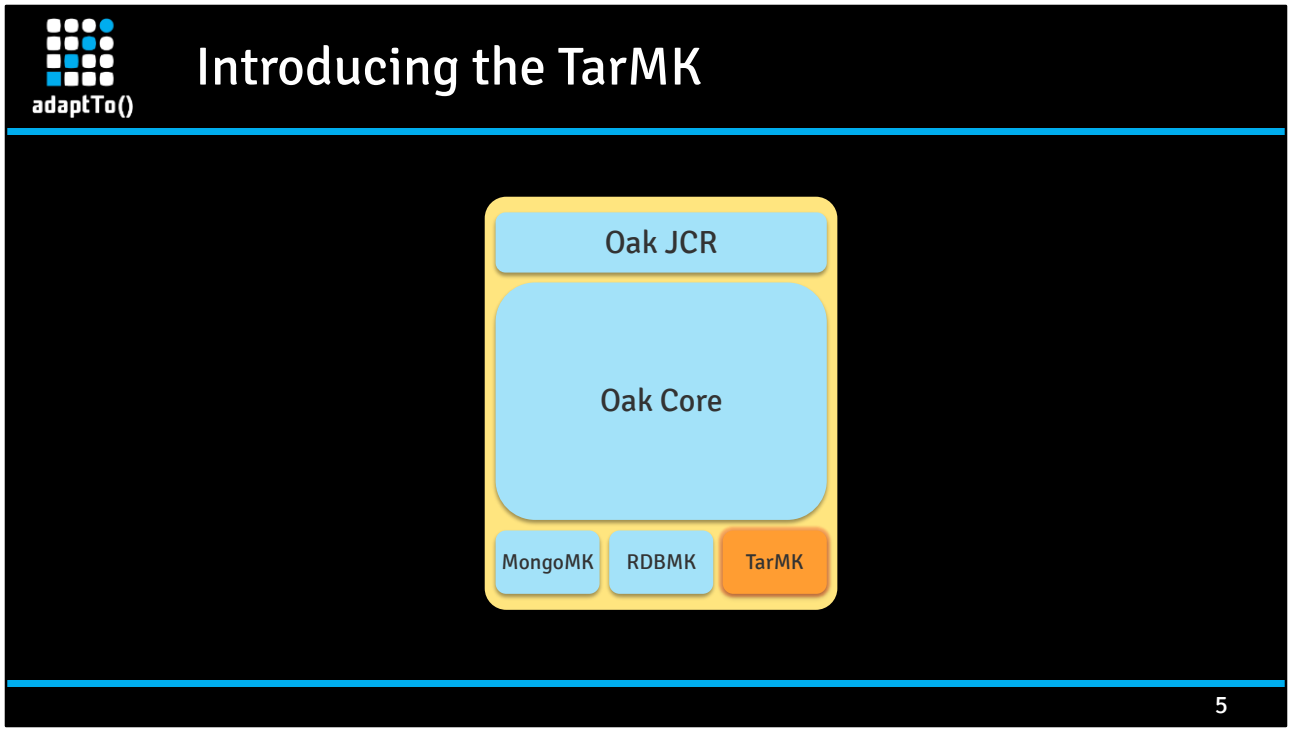
Why is my instance slow? What is it doing?
Where is the bottleneck? How do I fix it?



After an introduction of the inner workings of Oak and the TarMK , this presentation shows how system resources are used and what tools are available to monitor them. It presents a case study of typical problems its symptoms and possible remedies. Finally it concludes with an outlook on future areas of improvement.



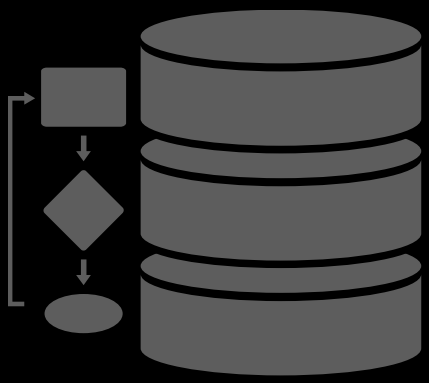
The TarMK is a tiny part of the whole AEM stack. It is one of multiple persistence options of the Java Content Repository implementation Jackrabbit Oak.



The TarMK is a tiny part of the whole AEM stack. It is one of multiple persistence options of the Java Content Repository implementation Jackrabbit Oak.

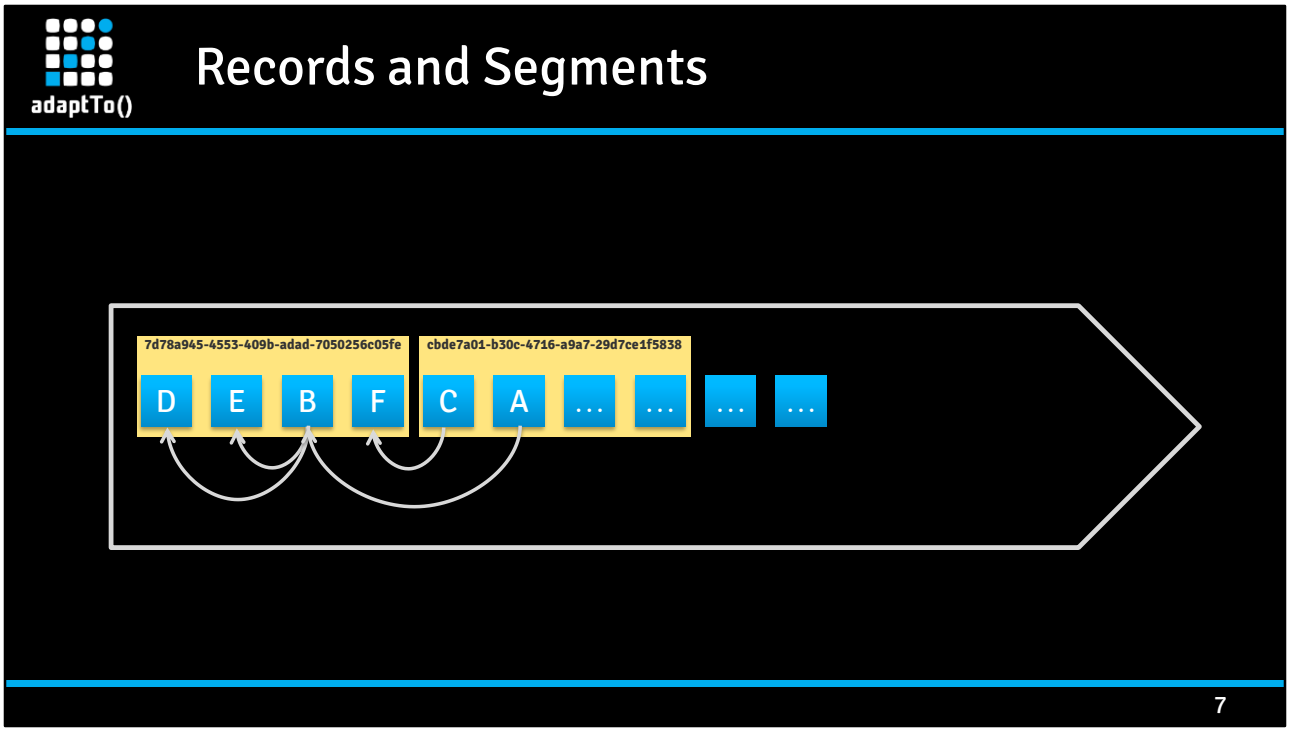
Features of the TarMK

- Embedded Database
  - Hierarchical
  - Fast / Small
  - Vertical scalability
  - MVCC / append only

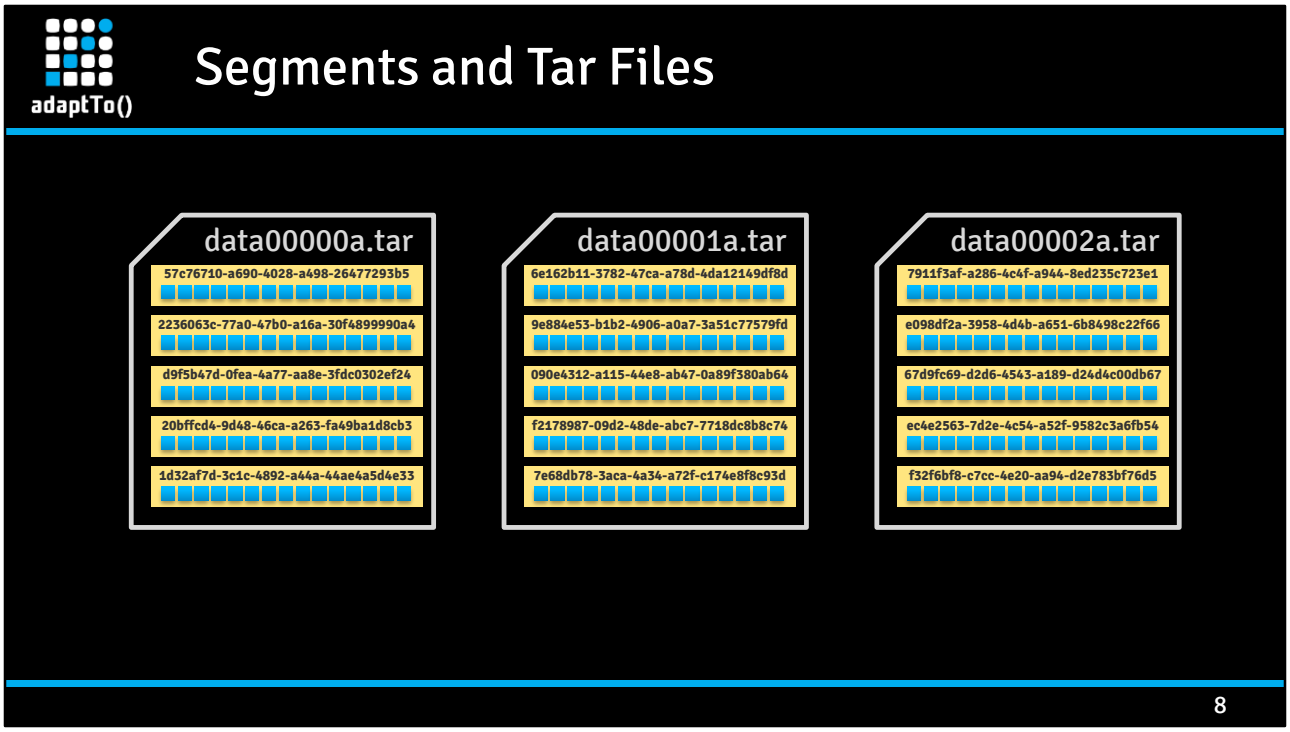


6

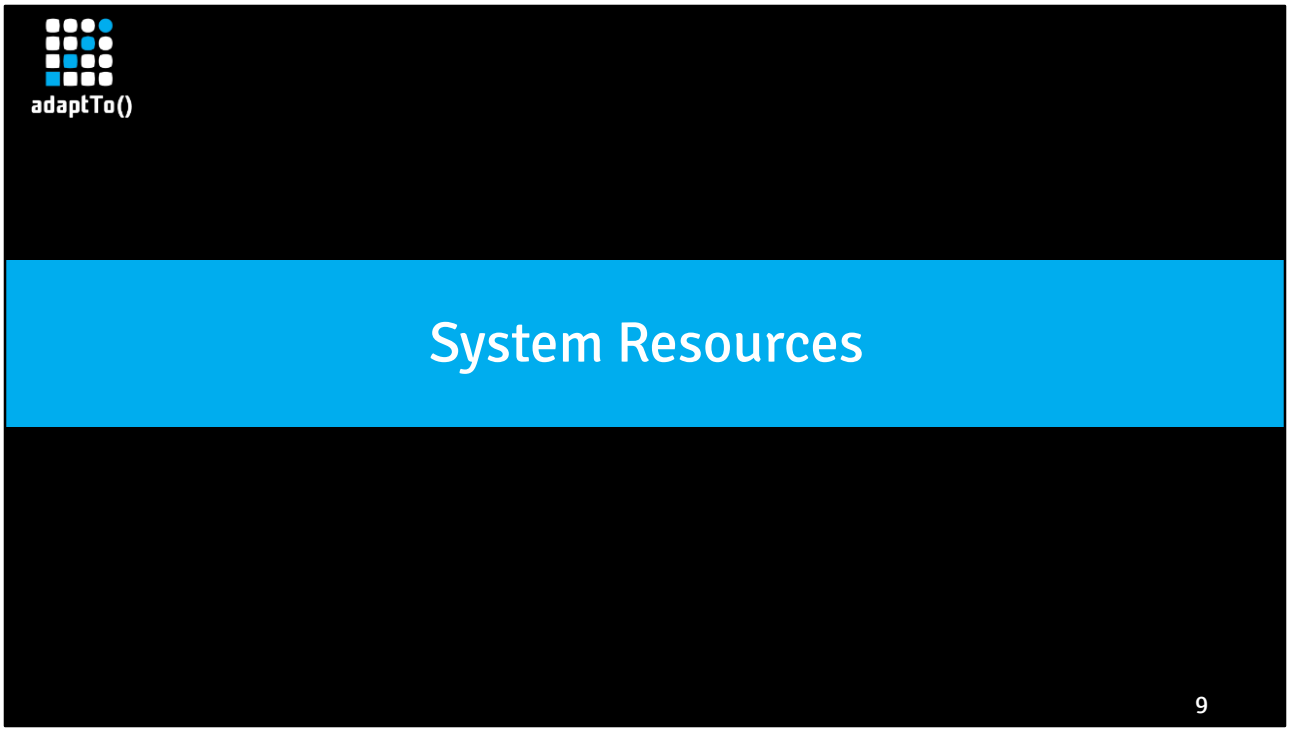
The TarMK is a fast, small and simple embedded hierarchical database engine serving as a persistence backend for the Jackrabbit Oak Java Content Repository. It implements multi-version concurrency control and stores all data in tar files in an append only way.



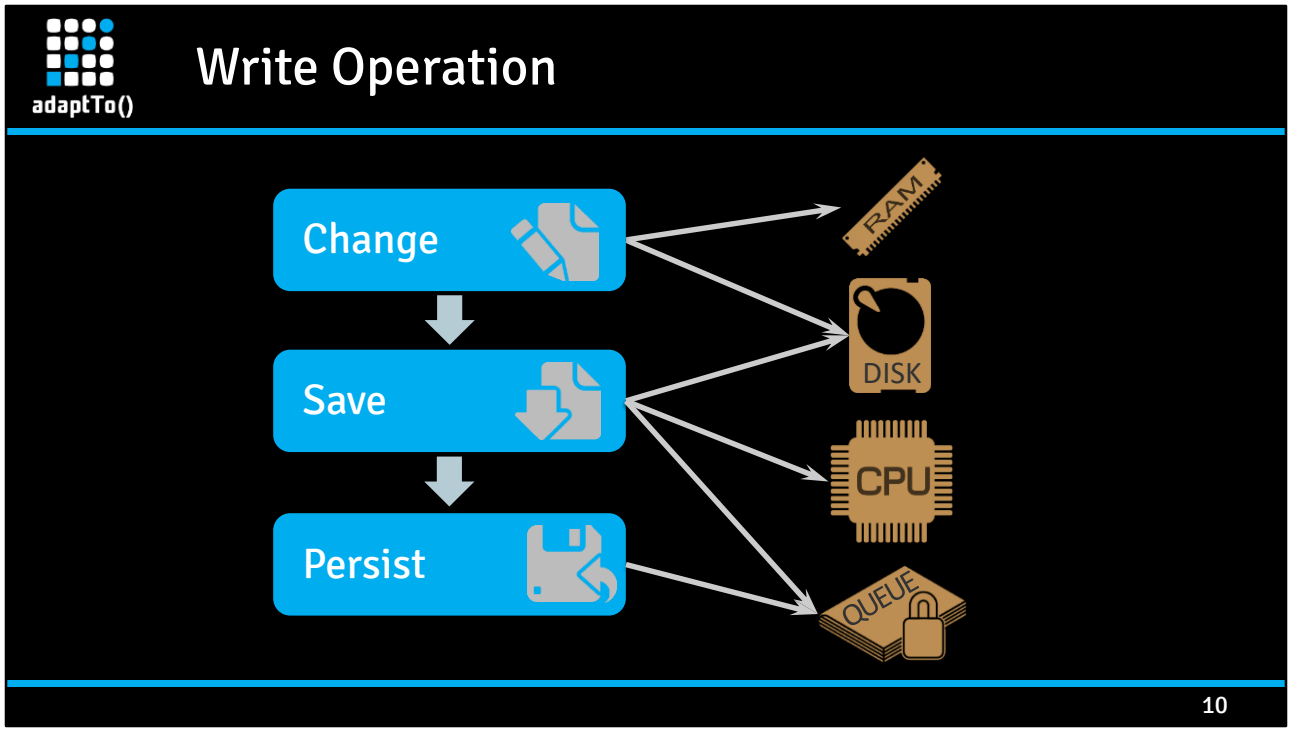
Each change to a node or a property is written to a record and appended to the list of existing record. Records can reference older records for data deduplication. Unreferenced records can be removed to reduce disk usage, increase data locality and reduce fragmentation. Records are grouped into segments, which are the smallest unit of persistence.



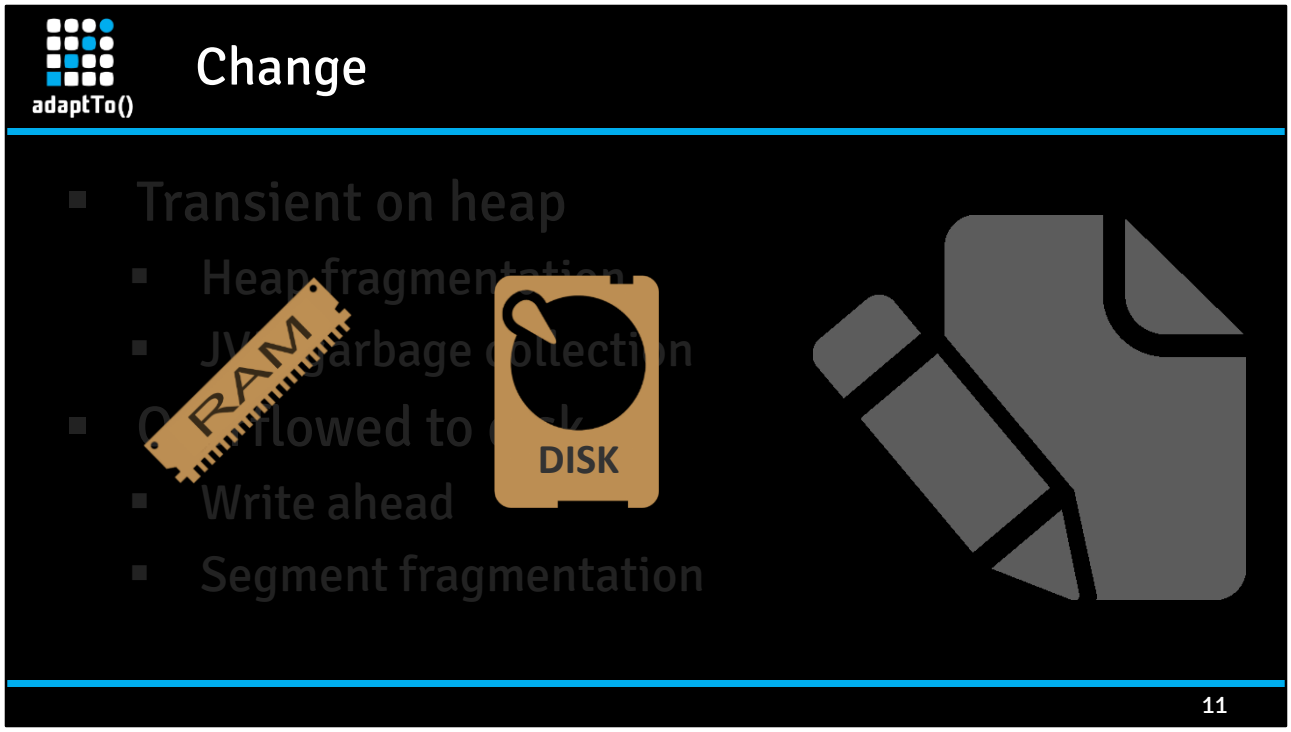
Segments are appended into tar files. Once a tar file becomes full (256MB by default) a new tar file is started. By default tar files are memory mapped for fast access. So it is important to avoid allocating all available RAM to the JVM (e.g. heap) as otherwise the OS would not have enough space for memory mapping the tar files, which could lead to disk thrashing.



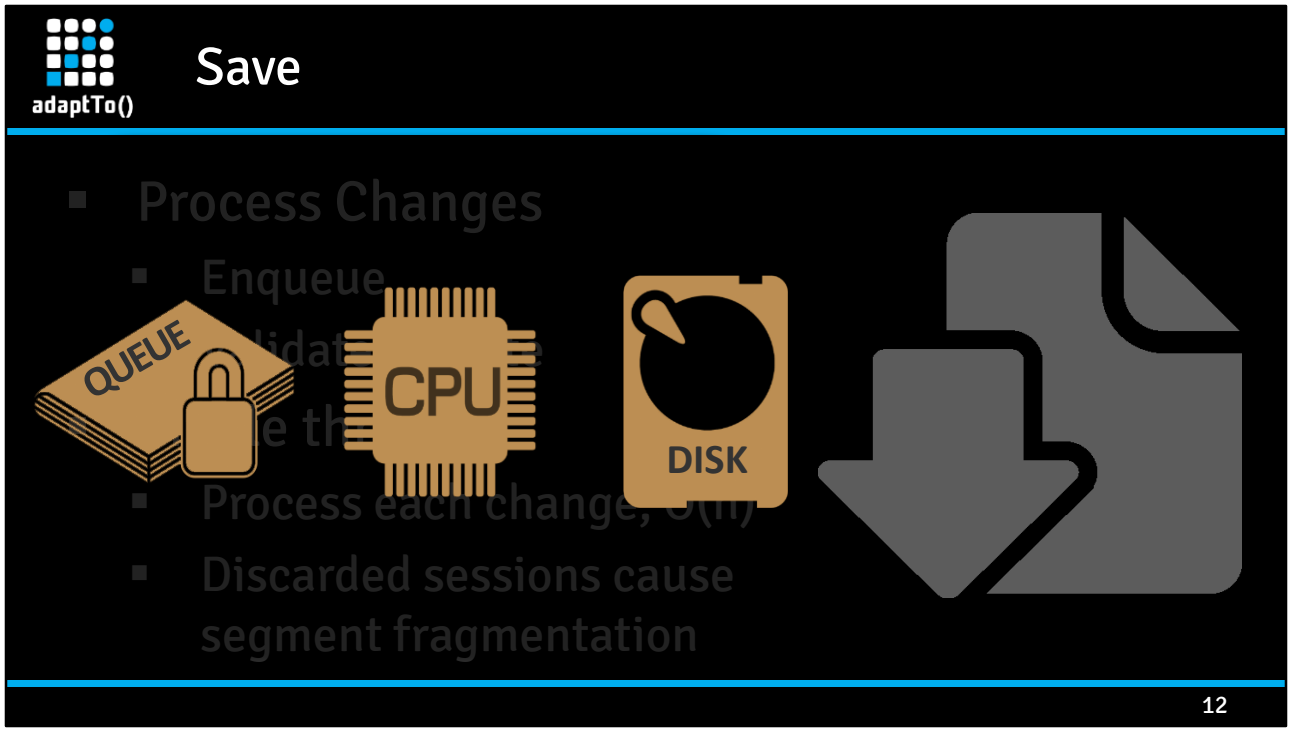
Oak requires various system resources like disk, memory, IO, etc. to operate properly and efficiently. This section explains how these are used during a typical write operation and how to monitor for anomalies.



A write operation roughly consists of three phases: first changes are transiently accumulated in the user's session. Subsequently when saving the session the commit phase processes all changes. Finally the persist phase atomically makes those changes durable by committing them to the journal.



Changes are transiently accumulated on the heap and overflowed to disk to ensure memory usage is bounded. Transient overflows are implemented as write ahead logic of the respective records. Transient changes can thus cause the disk footprint to increase, might result in a new tar file being created and mapped into memory (off-heap memory usage). The transient changes cached on the heap contribute to heap fragmentation thus increasing the workload of the the JVM's garbage collector.

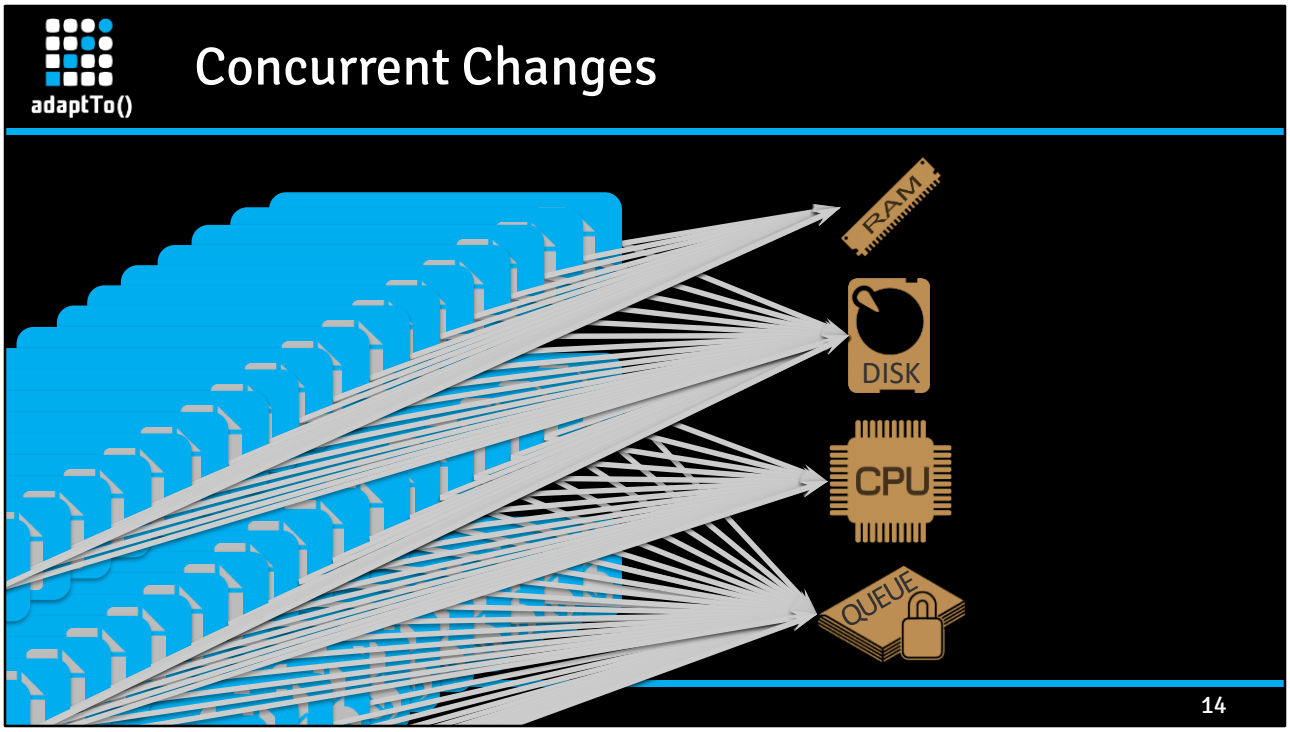


To save its transient changes a session is placed onto a queue of pending save operations. Once it arrives at the head of the queue all its changes are processed in order to ensure validity (referential integrity, uniqueness constraints, type soundness, etc.) and to update secondary data (indexes, auto generated items and values, etc.). During this processing the CPU is utilized by a single thread handling each single change. This often cause other nodes to be read from disk thus utilizing some of the disk IO bandwidth. Overall resource requirements are in the order of the

number of changes in the session.

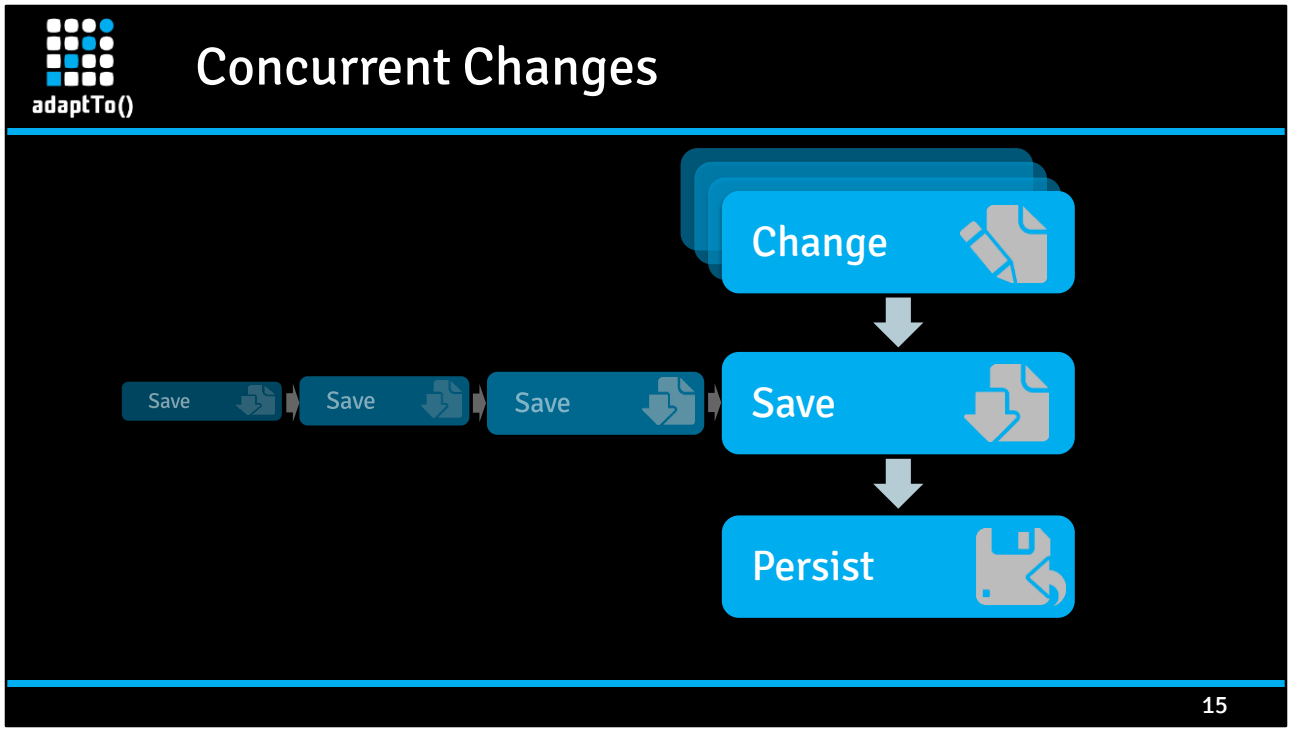
Discarding a session without saving this causes all written ahead records to become garbage, which contributes to segment fragmentation.

To make the processed changes durable Oak updates its journal with the id of the new root node and removes the session from the queue of pending save operations afterwards. This step does not directly involve a considerable amount of system resources. It often causes a considerable fan out subsequently though: updates to asynchronous indexes, generation and sending of observation events, workflow processing, asset ingestion, replication, etc.



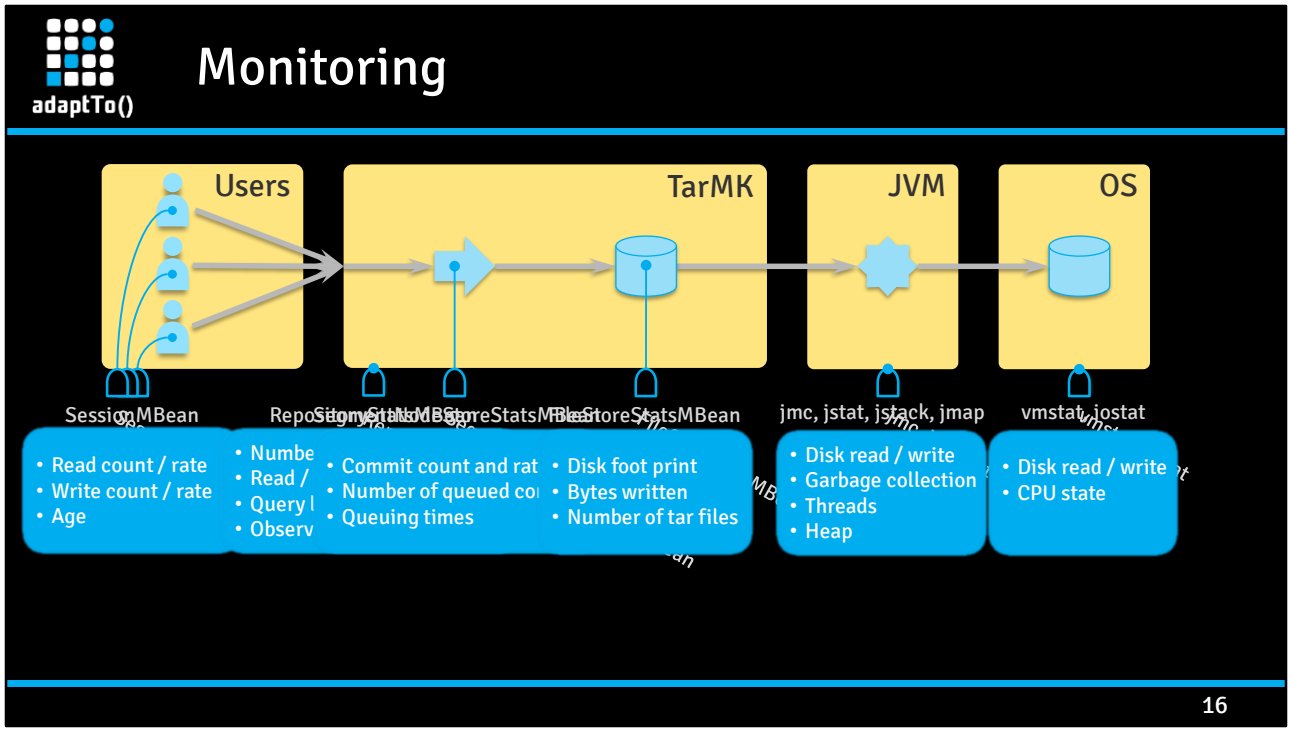
In a busy system there are many requests being processed in parallel and concurrently. The level of parallelism is limited by the available resources: e.g. the number of CPU cores limit how many threads can be in the runnable state at any given point in time. Once that limit is met parallel request start contending for resources, which adds overhead and reduces overall throughput. (E.g. CPU context switches, waiting for disk IO, etc).

Thread dumps are useful to examine concurrency and contention at system level.



It is the change phase that has the highest level of parallelism as individual sessions are mostly independent. However as they share the same memory and disk they may end up waiting for those resources on a heavily contended system.
In contrast the save and the persist phases have the highest level of concurrency as they are effectively executed serially contending for the TarMK commit queue. (An optimistic approach to waiting on the queue – seemingly more attractive at first – is actually inferior: in case of a conflict between concurrent saves all

but one participants need to reattempt the operation. This involves re-processing all changes thus adding more load to an already contended system, which in turn increases the risk for again running into a conflict).



Oak exposes various endpoints for monitoring the resource it uses:

- Each session exposes an SessionMBean instance, which contains counters like the number and rate of reads and writes to the session.
- The RepositoryStatsMBean exposes endpoints to monitor parallel requests like the number of open sessions, the session login rate, the overall read and write load across all sessions, the overall read and write timings across all sessions and overall load and timings for queries and

observation.

- The SegmentNodeStoreStatsMBean exposes endpoints to monitor commits: number and rate, number of queued commits and queuing times.
- The FileStoreStatsMBean exposes endpoints reflecting the amount of data written to disk, the number of tar files on disk and the total footprint on disk.

In addition to those endpoints there is many JMV and OS specific tools that help gaining further insight in what the system is busy with:

- Java Mission Control (jmc) is a very powerful tool to collect about every performance aspect of a running JVM. Its ability to record IO per Java process can sometimes be invaluable.
- The command line tools jstat, jstack, and jmap are useful to get inside into the JVM's garbage collector, the JVM's threads and the JMV's heap, respectively.
- The OS level tools vmstat and iostat can be used to examine IO and CPU usage at the operating system level.

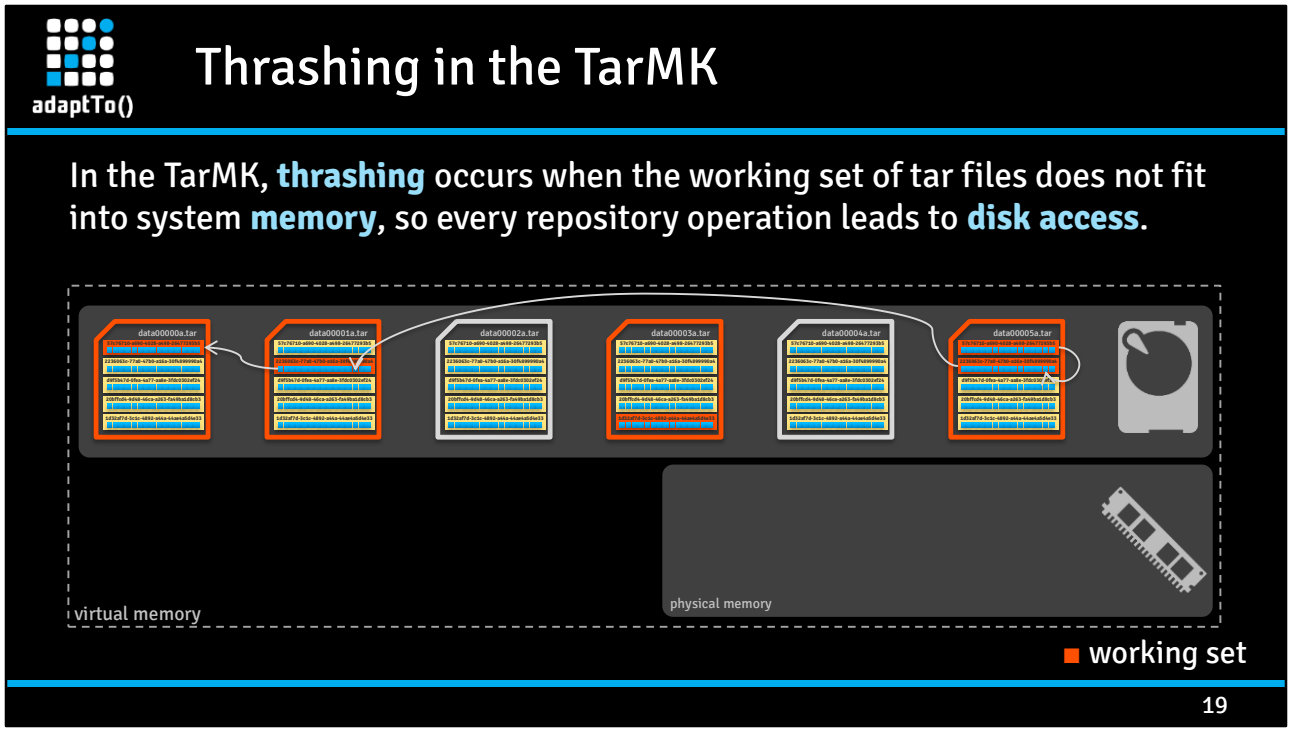
Together these monitoring endpoints provide different perspectives on the overall throughput in the system at the various layers: from JCR sessions to commits in the TarMK to disk IO of the TarMK. Combined with information collected with JVM and OS level tooling they provide a wealth of information about the system’s health and to help finding bottlenecks.


adaptTo()
Case Study: Thrashing
17

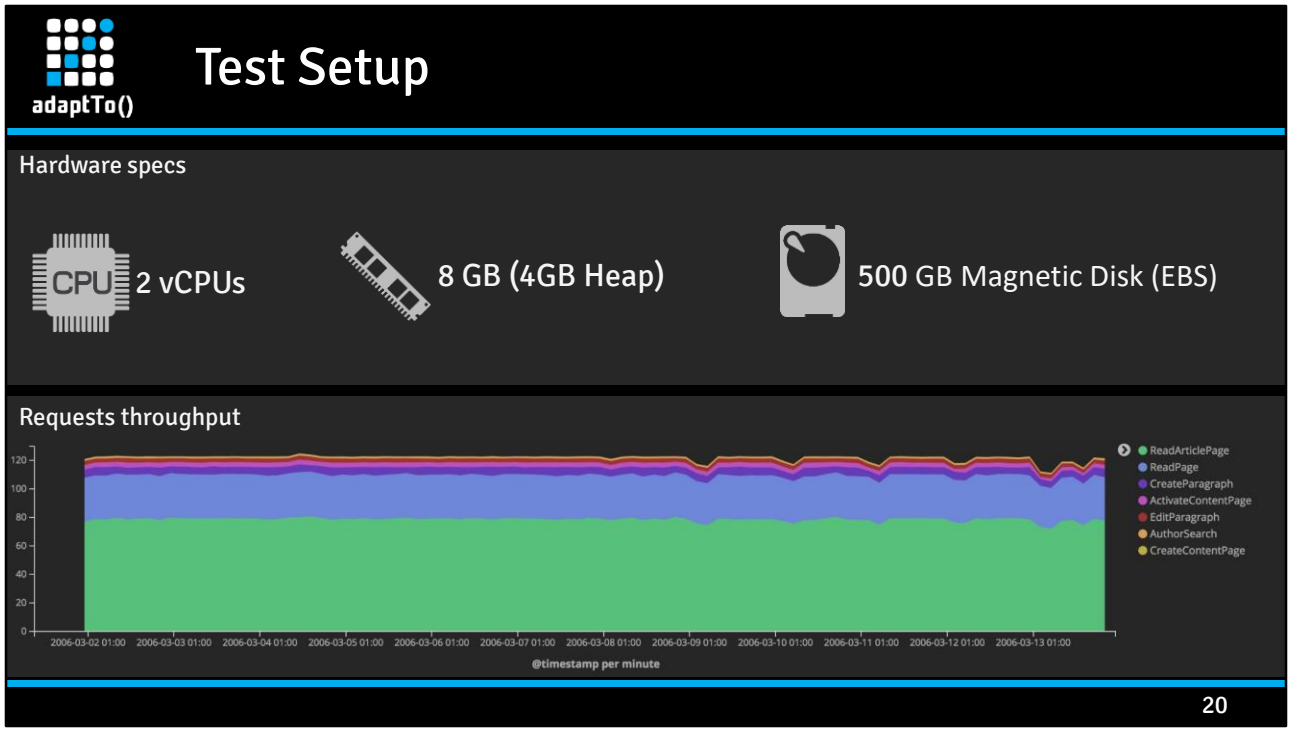
Thrashing

In computer science, **thrashing** occurs when a computer's virtual memory subsystem is in a **constant state of paging**, rapidly exchanging data in memory for data on disk, to the exclusion of most application-level processing. This causes the performance of the computer to **degrade or collapse**.

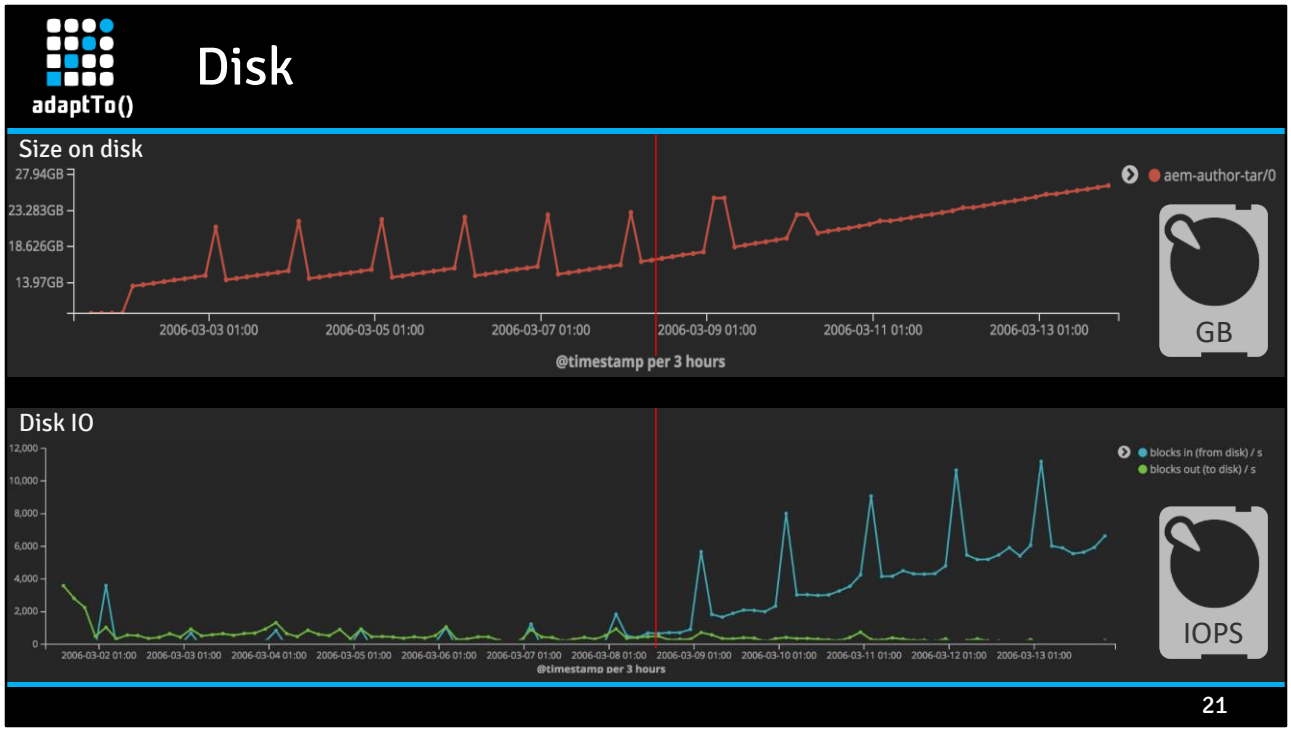
https://en.wikipedia.org/wiki/Thrashing_(computer_science)



TarMK leverages memory mapping mechanisms provided by the OS to cache the tar files. It means that when a segment is read, the corresponding tar is loaded into memory and kept for future access. This goes on until the available RAM is filled and at that point, old tars have to be unloaded for newly accessed ones. This results in extra disk reads, which makes the instance slow. When the set of tars that are frequently accessed, also called the working set, is way bigger than the cache size, almost all the processing time is spent waiting for the disk.

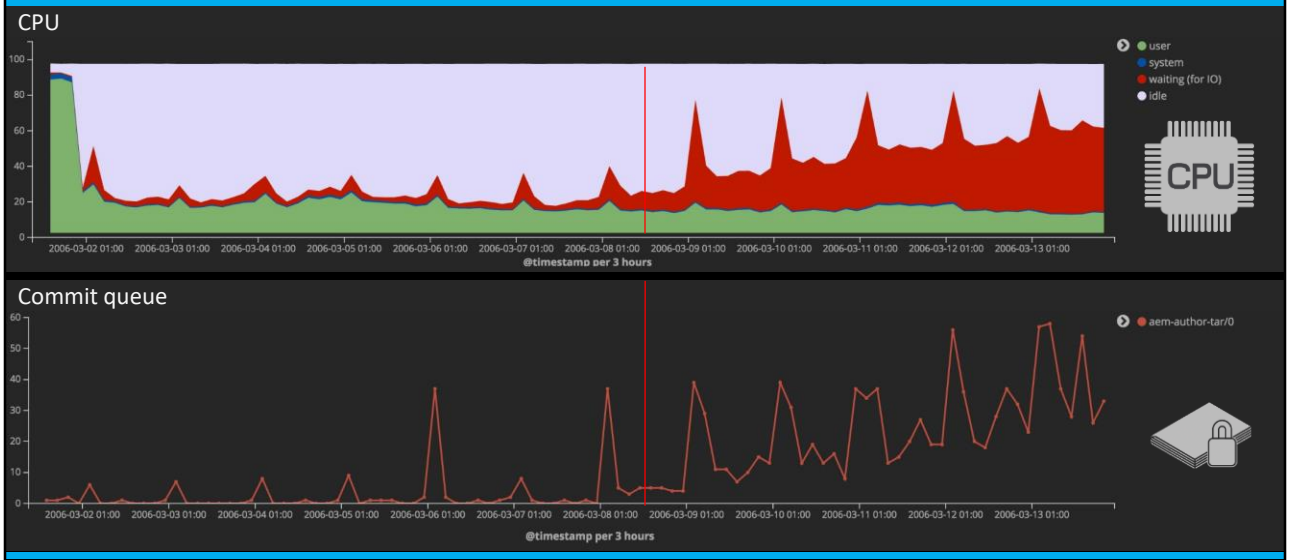


We have created a test to expose the limit at which the system is starting thrashing due to insufficient memory. Hardware specs are on purpose low to reach this limit faster.
Incoming requests simulate a typical sites authoring scenario (browsing and editing content), but with constant throughput over two weeks.



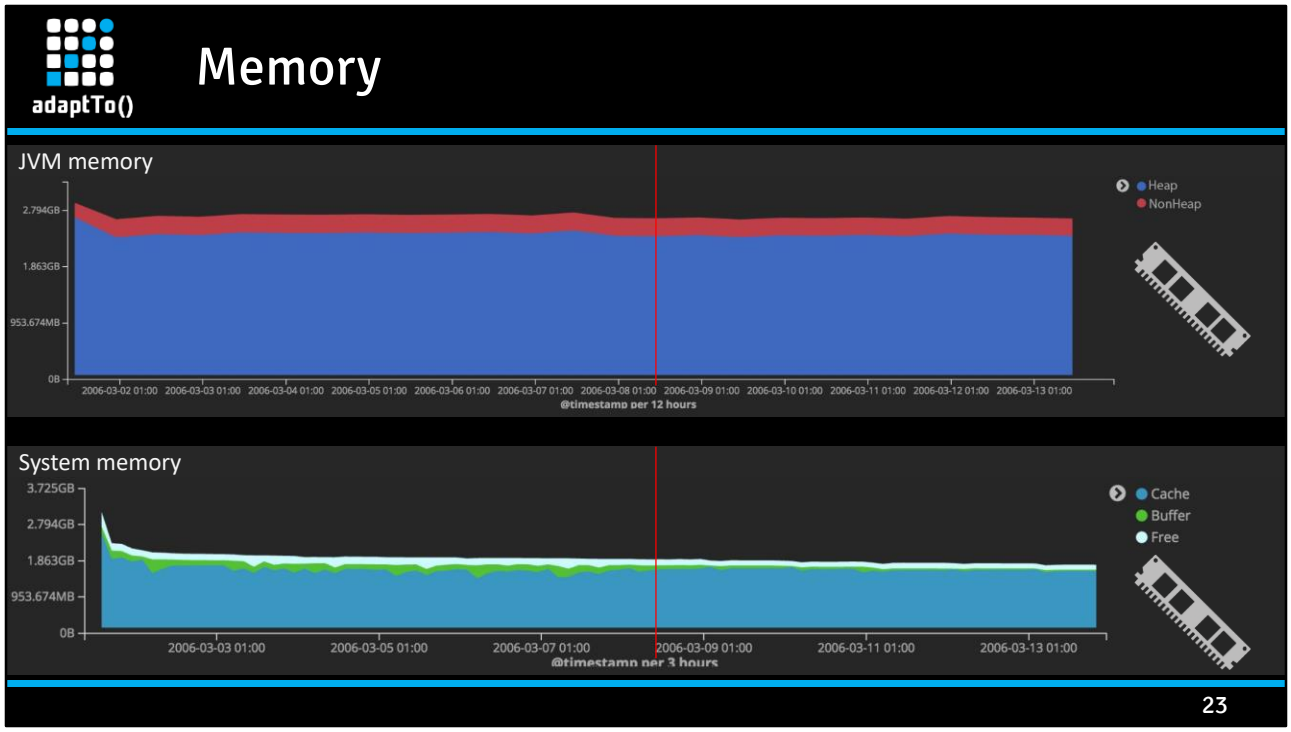
Tipping point marked with red line – size on disk > 16GB and reads > writes

CPU and Commits



22

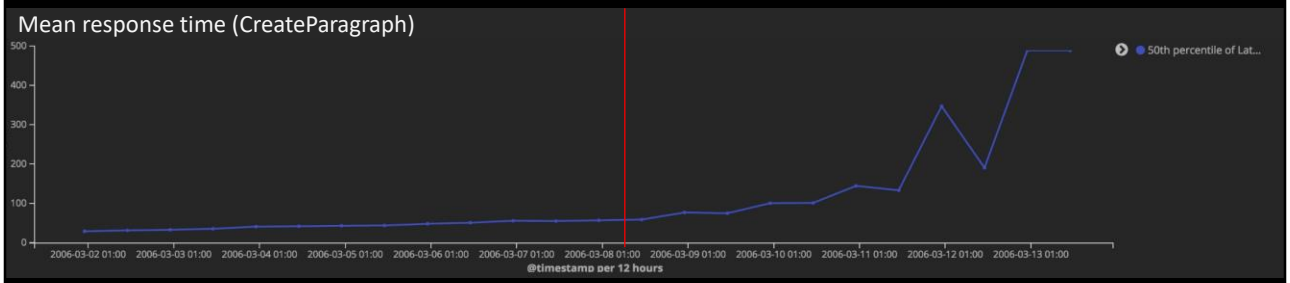
After tipping point, CPU spends most of the time waiting for the disk.
After tipping point, the commit queue increases, which results in higher response times.



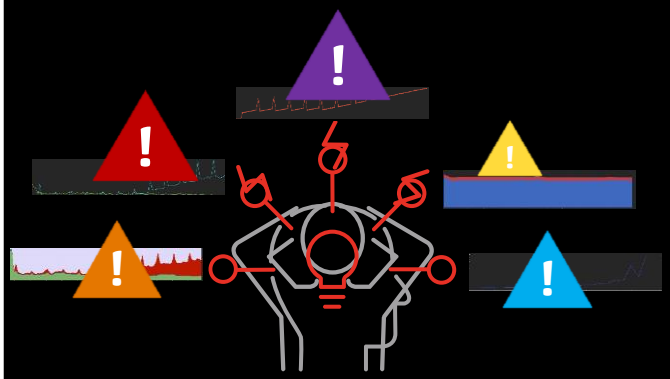
Not much to comment on memory, the expected (constant) levels of usage.

Response Times

Mean response time (CreateParagraph)



Now What?



NOW WHAAAAAAAT?!

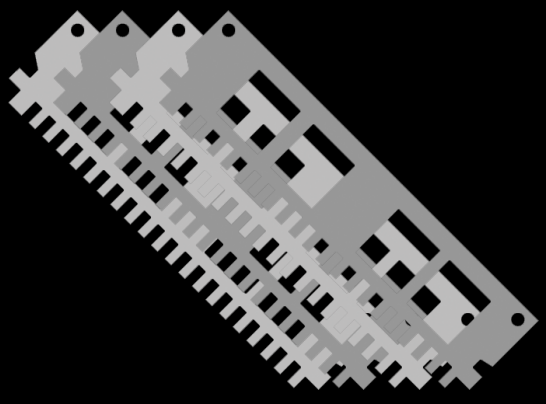
1. 

2. Qualify the problem

3. Take prompt actions

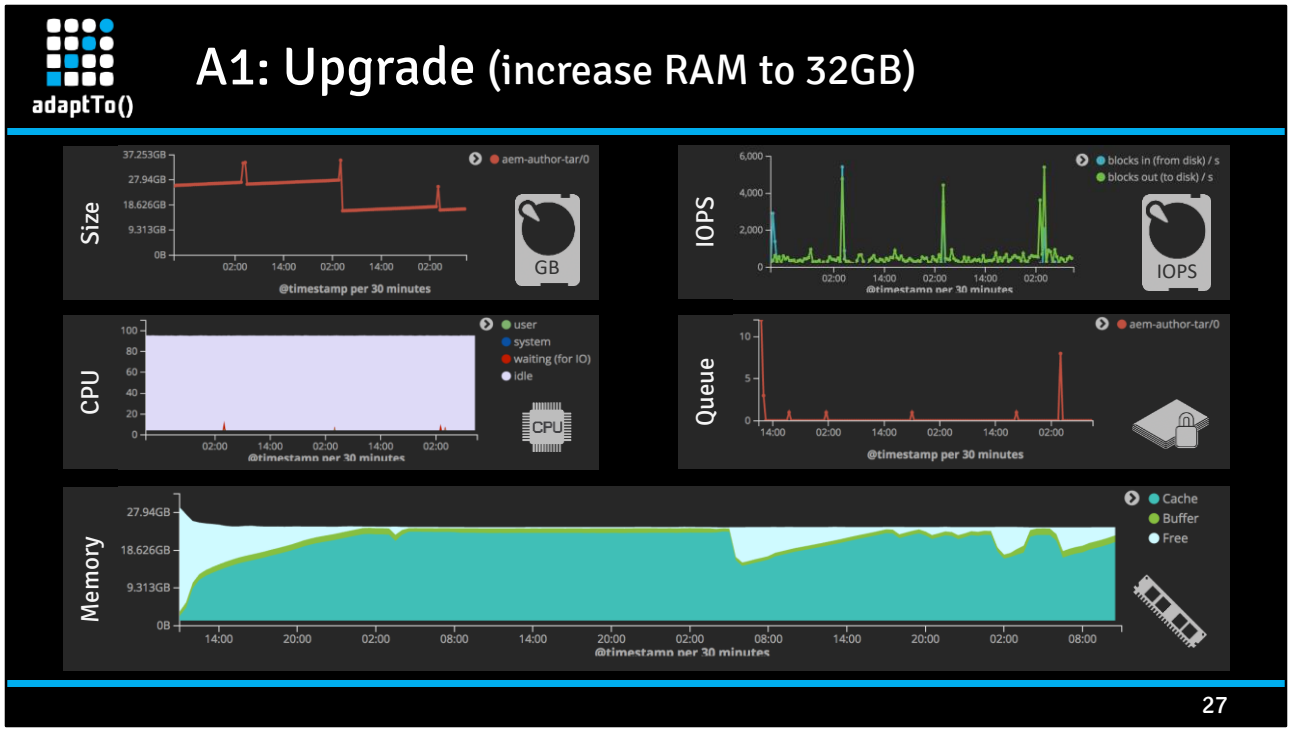
Potential actions (1)

1. Upgrade hardware
   - Add RAM
   - Optimize IO



26

Ideally, the system should have enough free memory to cache the full segmentstore. But, if this is not possible, any increase in RAM will help mitigate the problem.
To further optimize IO consider using a dedicated data disk holding the tar files. Reduce read-ahead and turn off transparent huge pages on that disk. Also consider using a dedicated disk for the Lucene NRT indexes.



After provisioning enough RAM to the instance to hold the whole repository in memory, all the parameters go back to normal values, even with the same segmentstore that created problems:

- CPU utilization is below 5%
- disk is used mostly for writes (triggered by creating pages)
- and the commit queue is almost all the time empty.

The Cache in Memory graph reflects the use of memory mapped files: it progressively grows to

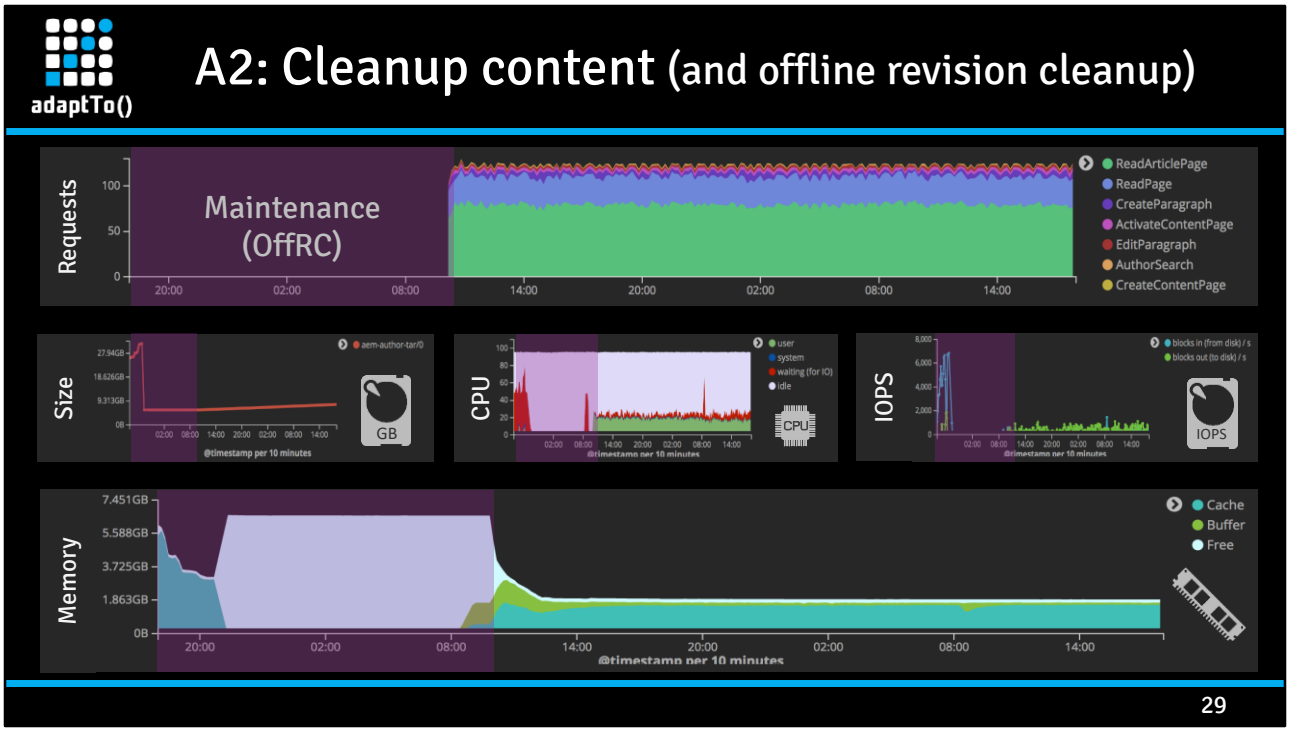
the same size as the segmentstore (as more and more tars are accessed and cached by the test), and shrinks after OnRC reduces the footprint.

Potential actions (2)

2. Reduce repository
   - Use a blob store
   - Manage inactive content (Content hygiene)
   - Optimize indexes

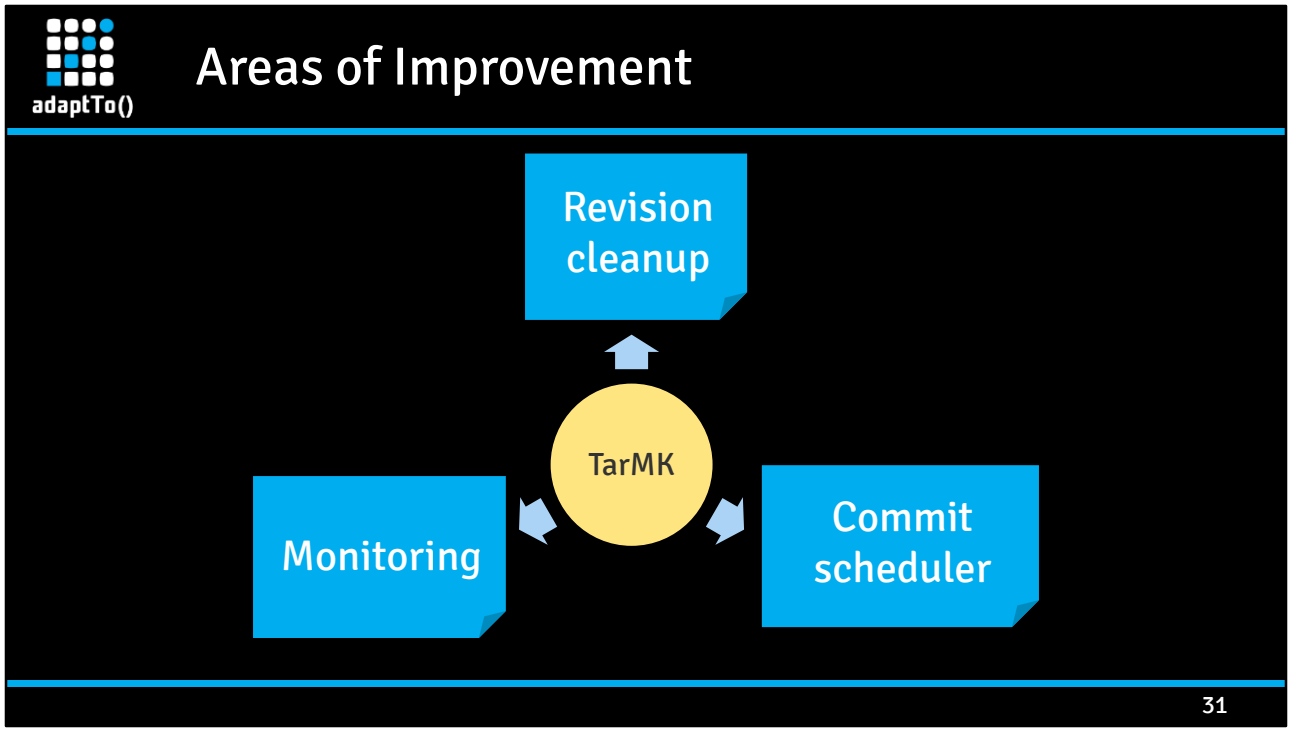
28

To reduce the disk footprint regularly schedule version purges. Also remove any temporary and not needed content from the repository.

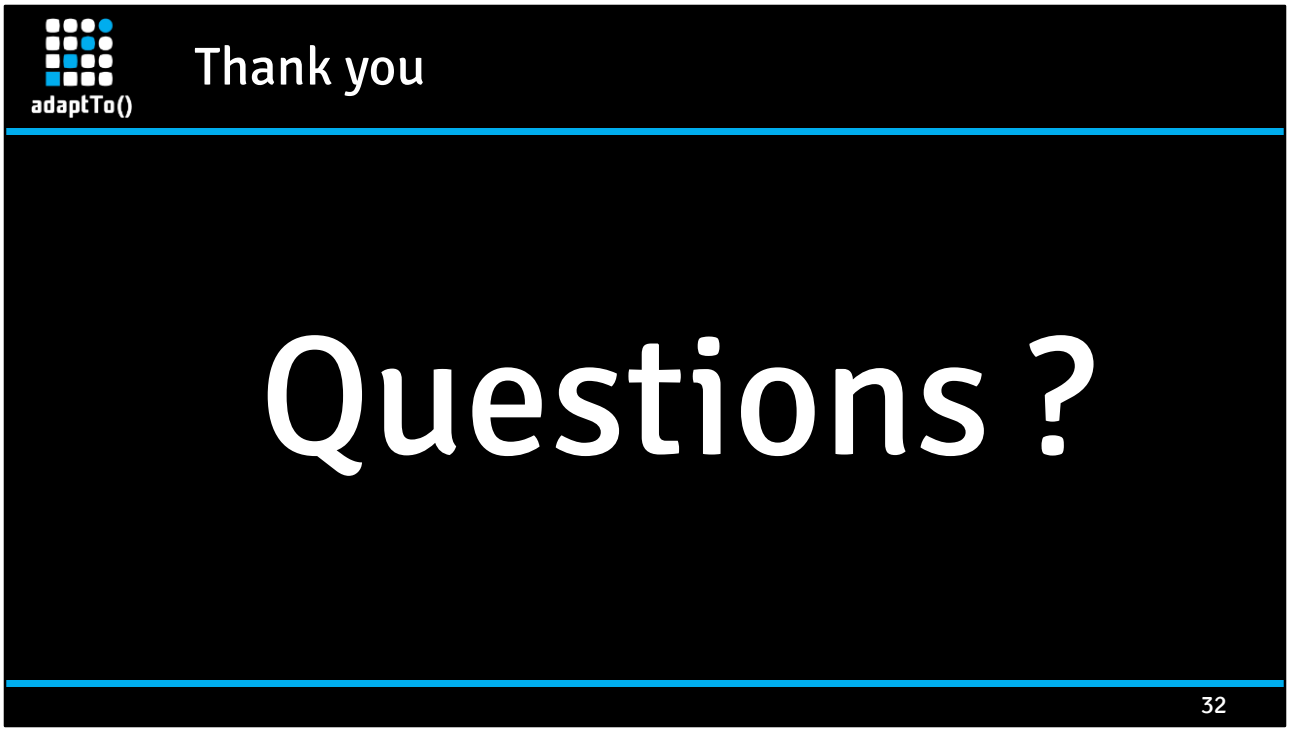


Another solution to recover is to reduce the segmentstore by cleaning up unnecessary content. In this case, we delete some of the previously created pages then run offline revision cleanup (OffRC is needed here). As observed, the segmentstore size drops to about 5GB and when the test is restarted, the instance goes back to the normal state. Also, the system's cache grows to the maximum allowed by the physical RAM.

Outlook



- Revision cleanup is an effective way to reduce the TarMK's disk footprint thereby keeping locality high. By letting revision cleanup focus on volatile content the process is able to complete faster while at the same time using fewer recourses.
- By introducing a commit queue and a scheduler to prioritize and schedule commits the TarMK can select an optimal strategy to maximize throughput.
- Detailed monitoring endpoints for the commit queue enables early detection of system overload.


adaptTo()
Thank you
Questions ?
32

Appendix

Typical Segment Store Composition