When Linux Memory Accounting Goes Wrong

Minhaj Ahammed

October 12, 2021

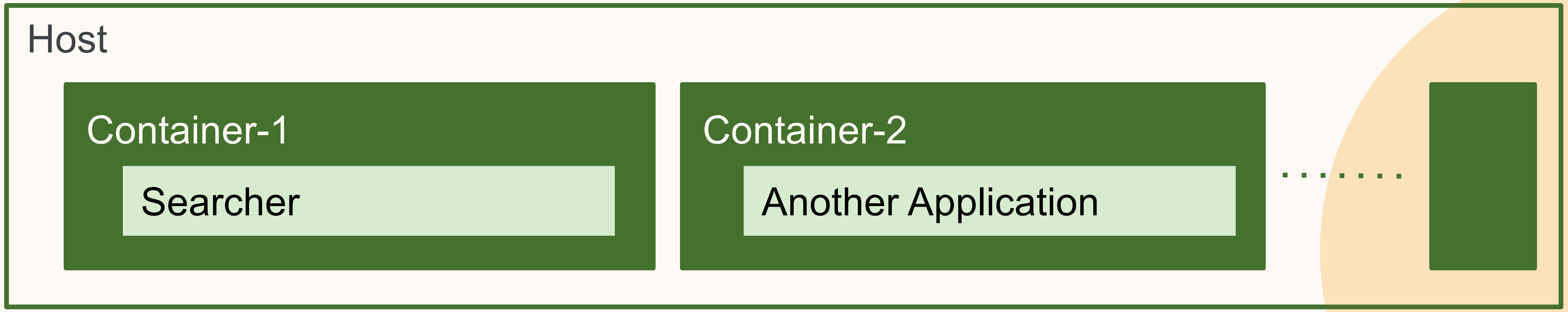
Search at LinkedIn

Galene

Search-as-a-Service (SeaS) infrastructure that powers search at LinkedIn.

Searcher

- Queries the search indices.
- Loads index files into memory.
- Use cgroups to limit CPU, Memory.



Concepts

01 Container

A standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

02 Control Groups (cgroups)

Kernel feature that limits, accounts for, and isolates the CPU, memory, disk I/O and network usage of one or more processes. This helps us co-host multiple applications on the same host while ensuring that they don't use more resources than allocated to them

03 mlockall()

System call that locks part or all of the calling process's virtual address space into RAM, preventing that memory from being paged to the swap area.

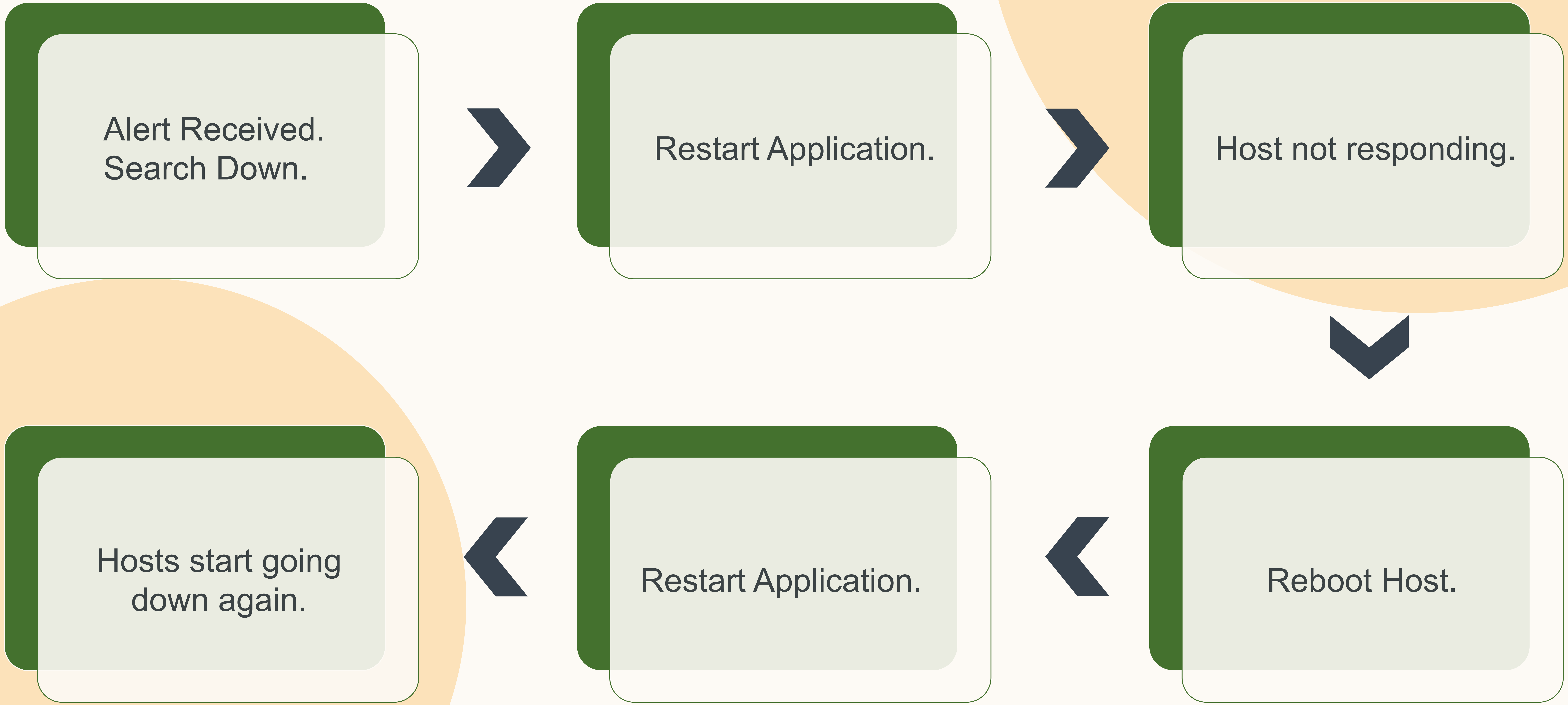
04 Out of Memory Killer (OOMKiller)

- Process that the Linux kernel employs when the system is critically low on memory.
- Also called when cgroup memory limits are breached.

05 Resident Set Size (RSS)

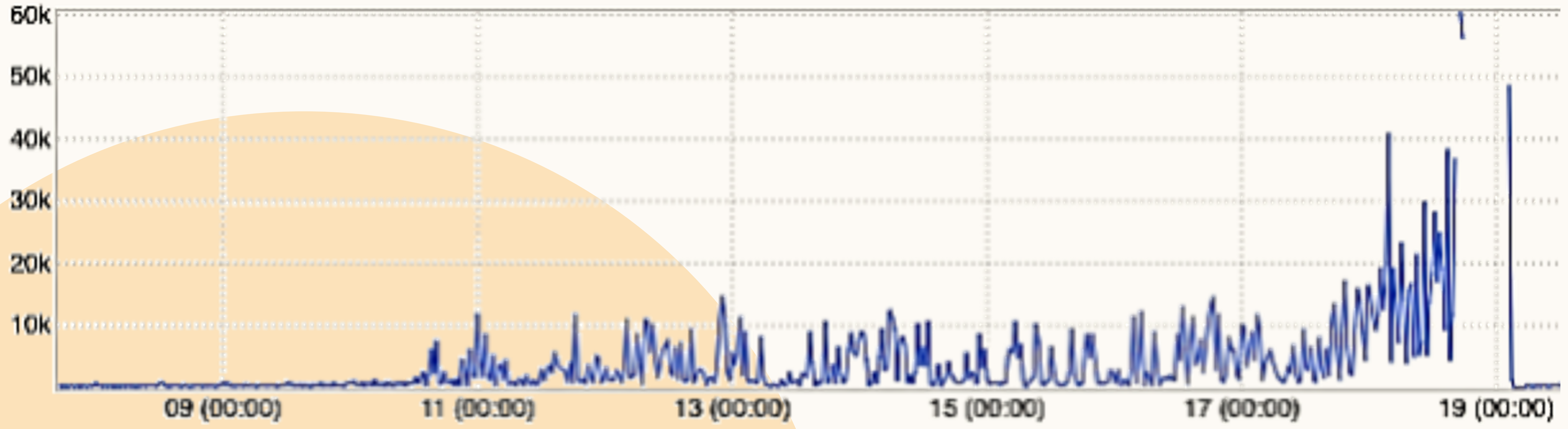
- Portion of memory occupied by a process that is held in main memory (RAM)

The Problem

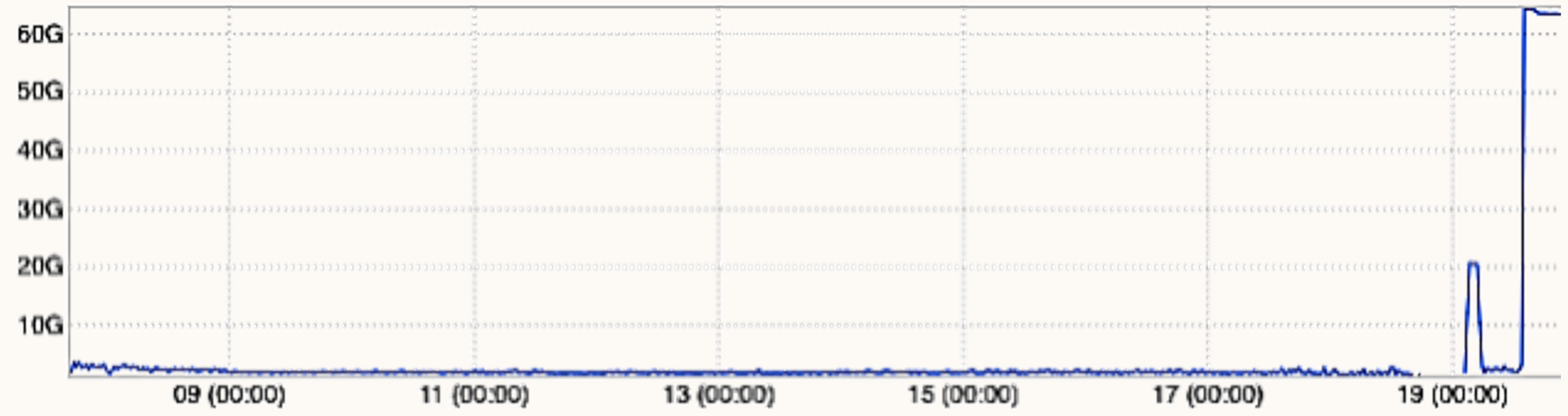

Alert Received. Search Down.
Restart Application.
Host not responding.
Reboot Host.
Restart Application.
Hosts start going down again.

Initial Debugging and Resolution

- The hosts were low on memory.
- There was lots of swapping going on.
- No logs of any kind were generated on the host once it was unreachable.



Host Disk Read time Graph



Host Available memory graph

Initial Debugging and Resolution (contd)

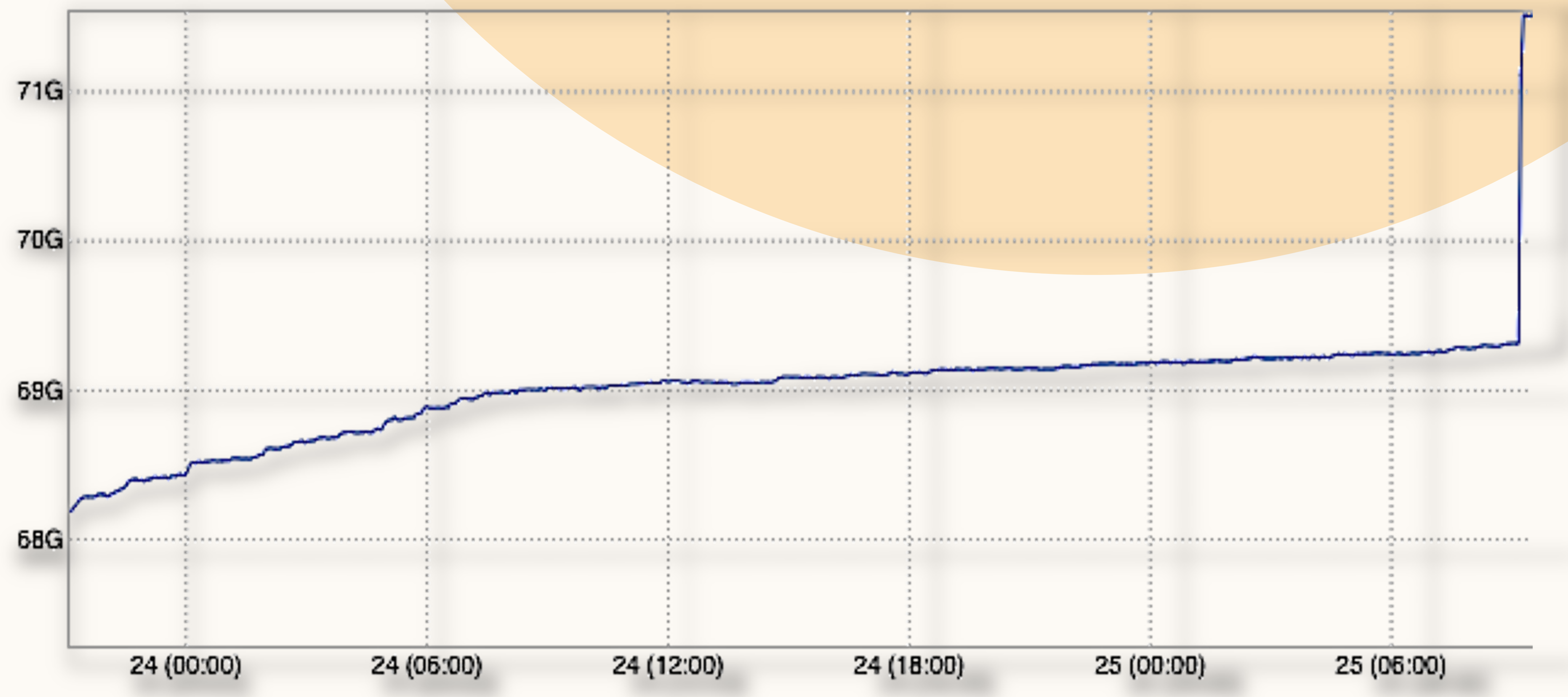
- Did not find the real culprit hogging system memory.

- Resolution
  - Optimized the searchers memory utilization
  - Reduced the cgroup memory limit for the application

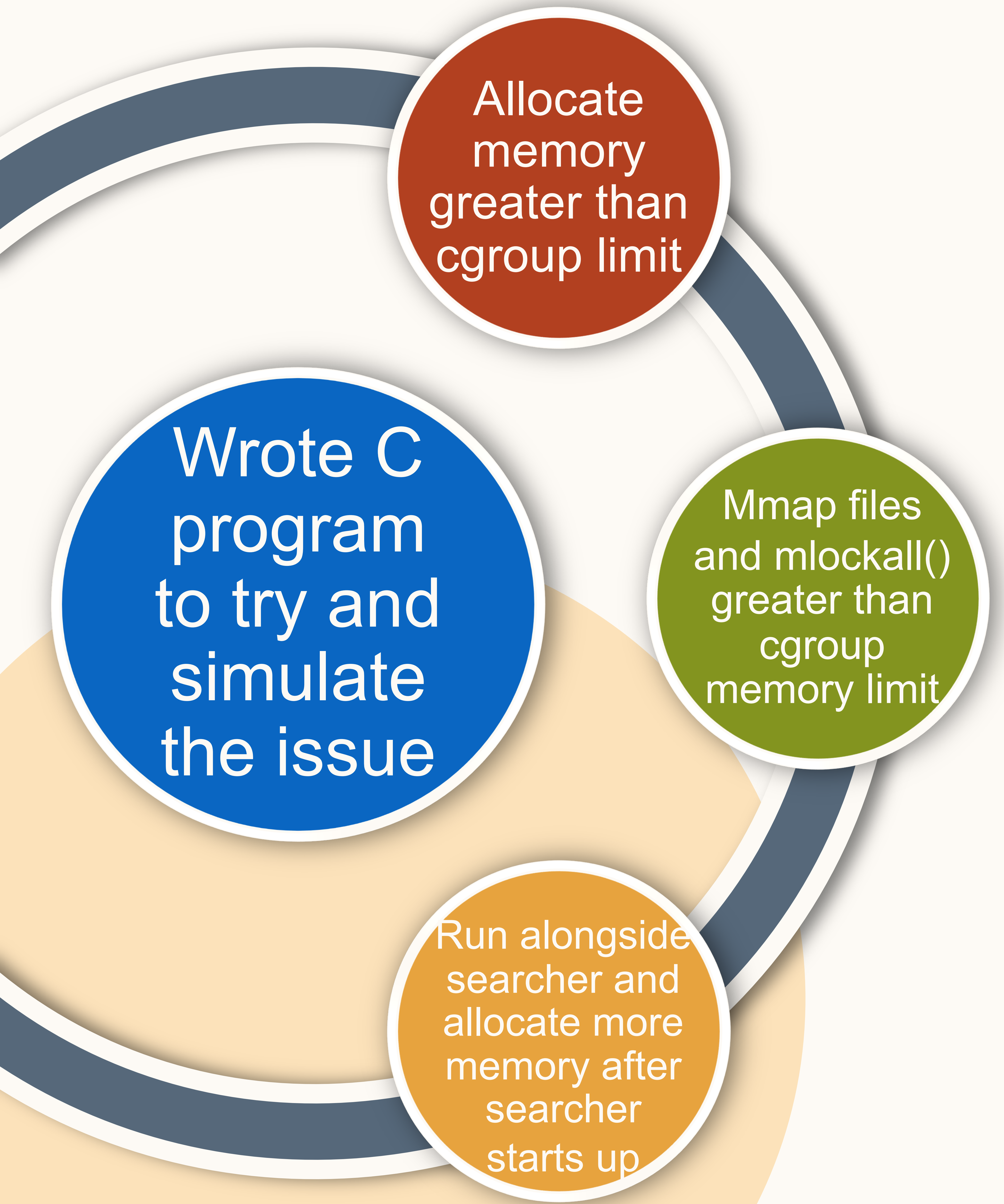
Issue recurrence

- ➢ Application requests big chunk of memory before going down.
- ➢ Prime suspect = Linux's cgroup Memory Enforcement

Application cgroup memory_usage_in_bytes graph



Issue recurrence


Wrote C program to try and simulate the issue
Allocate memory greater than cgroup limit
Mmap files and mlockall() greater than cgroup memory limit
Run alongside searcher and allocate more memory after searcher starts up

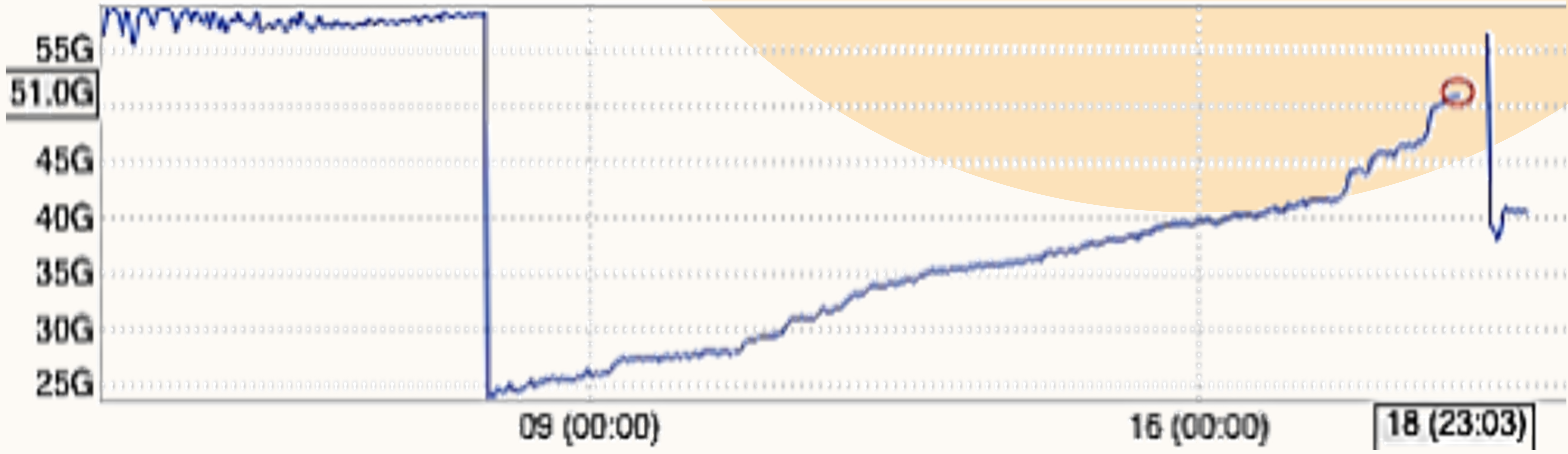
OOMkiller invoked correctly

Since replicating the problem had proved unsuccessful we went back to take a deeper look at cgroups.

Troubleshooting

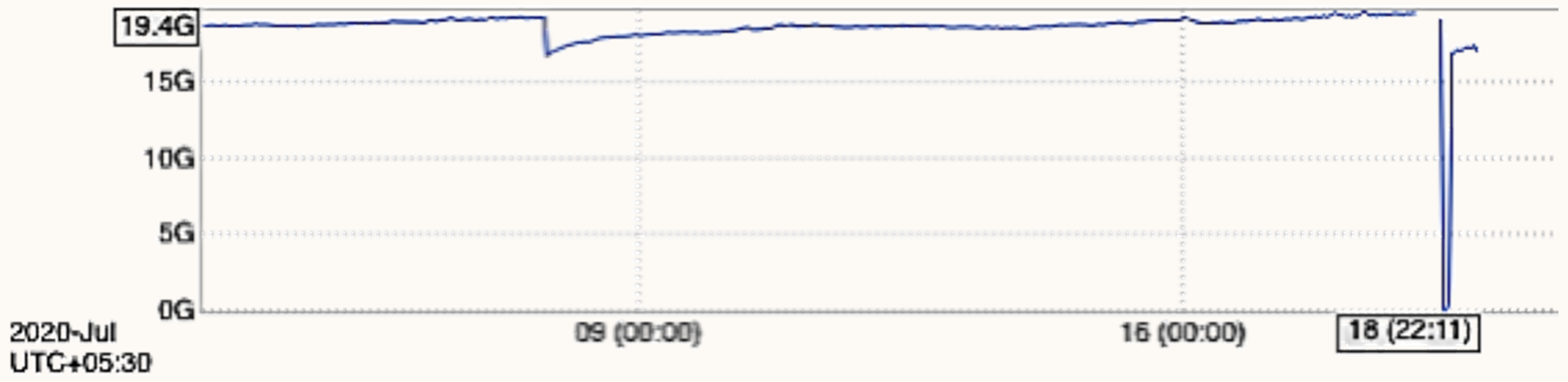
Investigating the memory usage pattern on cgroup.

Application cgroup memory_usage_in_bytes graph

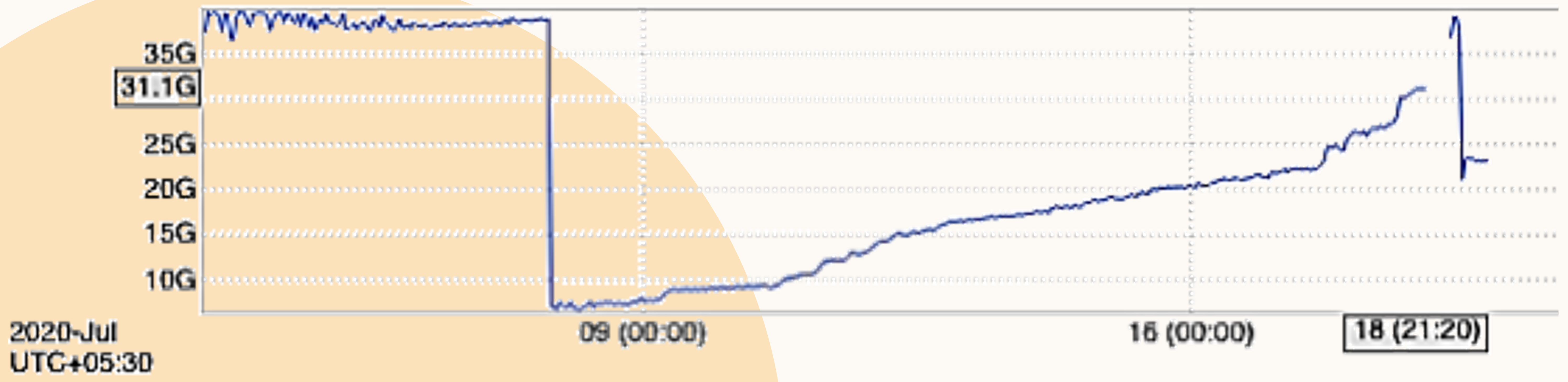


Memory usage = RSS + page-cache.

Troubleshooting (contd)



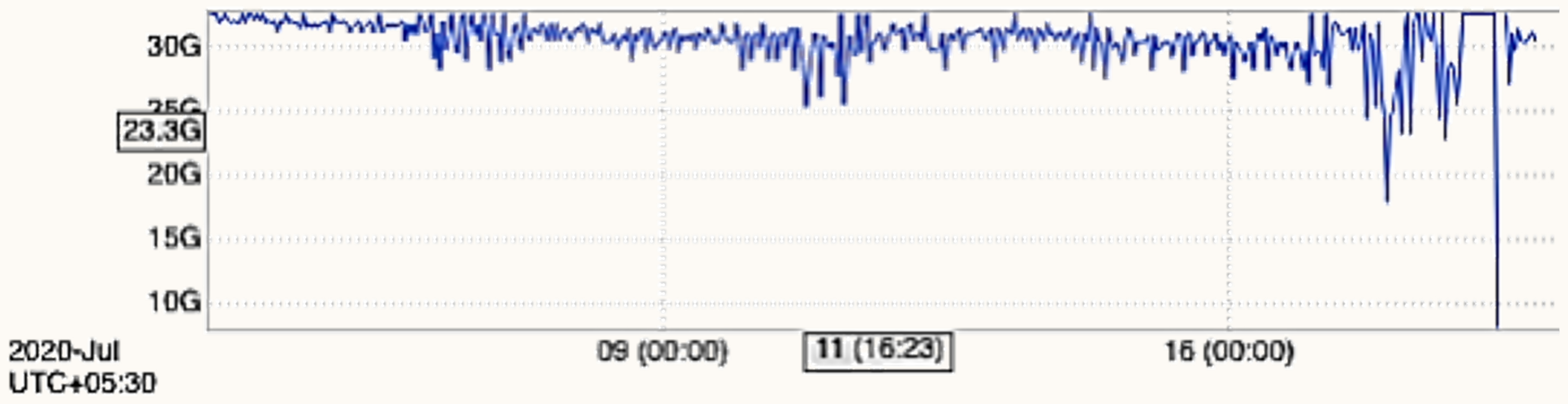
Application cgroup RSS usage graph



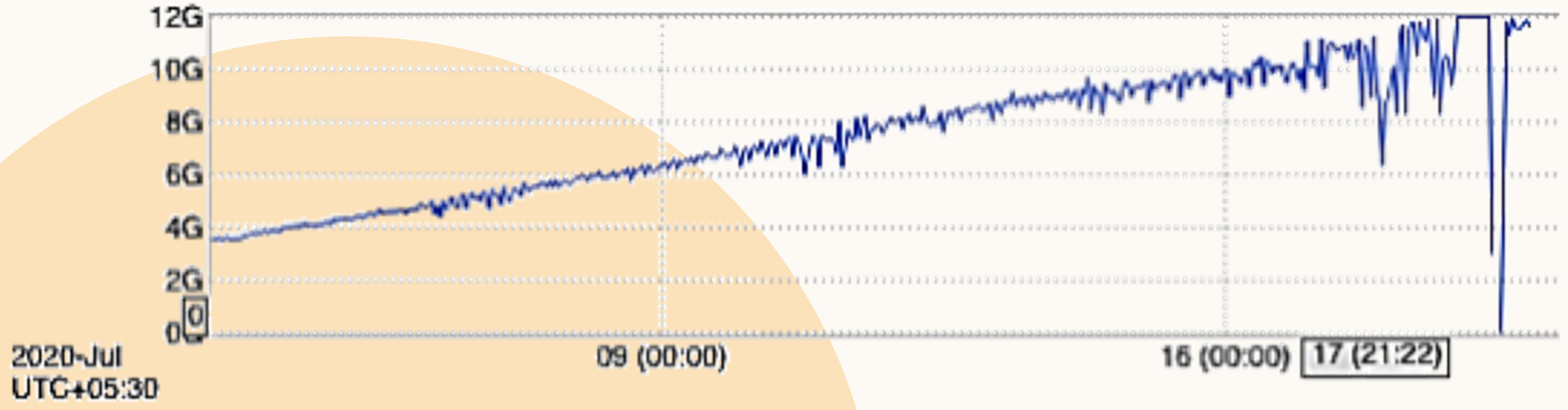
Application cgroup cache usage graph

Adding memory usage for RSS and cache, 19.4 + 31.1 ≈ 51GB.

Troubleshooting (contd)



Searcher Application base index size graph



Searcher Application middle index size graph

Adding index sizes. 32 + 12 = 44GB.
Adding RSS value to it, 44 + 19 = 63GB

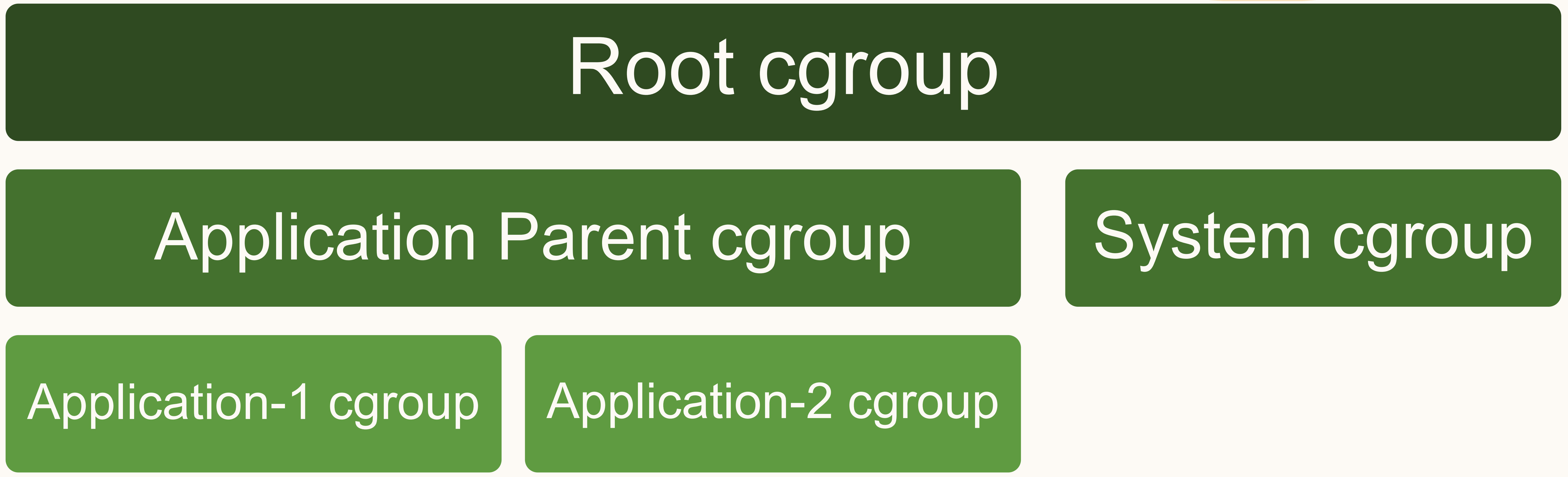
So, the application is using 63GB.

Troubleshooting (contd)

Actual page cache usage - 44GB

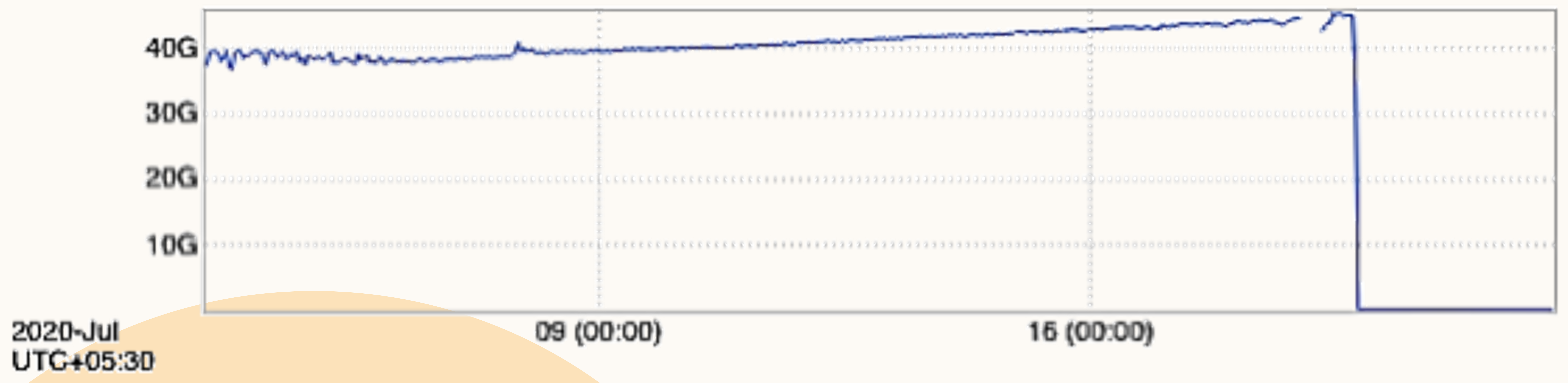
Usage reported by cgroup - 31 GB

Hierarchy of our cgroups

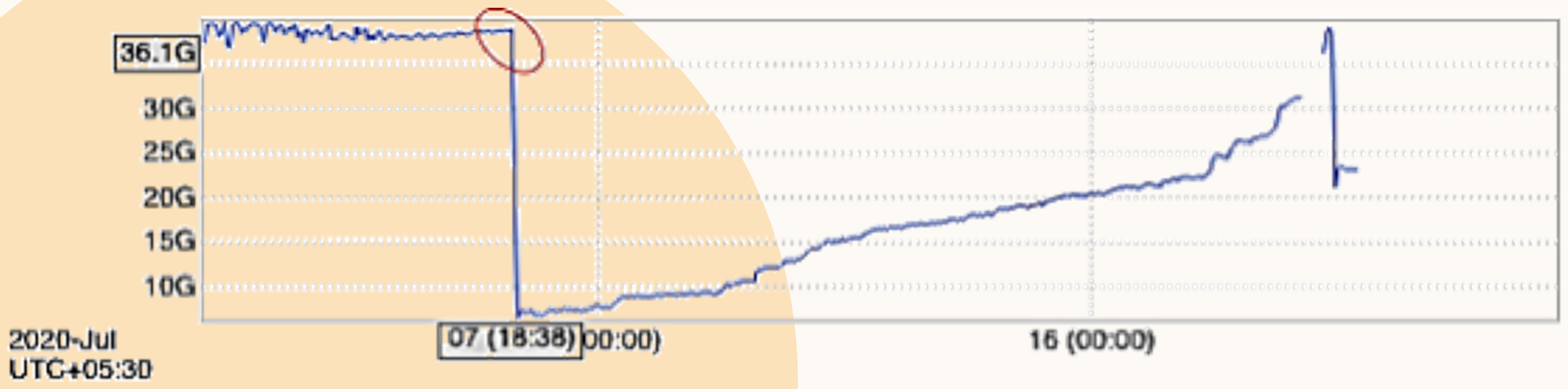


Troubleshooting (contd)

Let's compare page cache with parent cgroup,



Parent cgroup page cache usage graph



Application cgroup page cache usage graph

Dip in cache usage by the application cgroup is due to a restart.

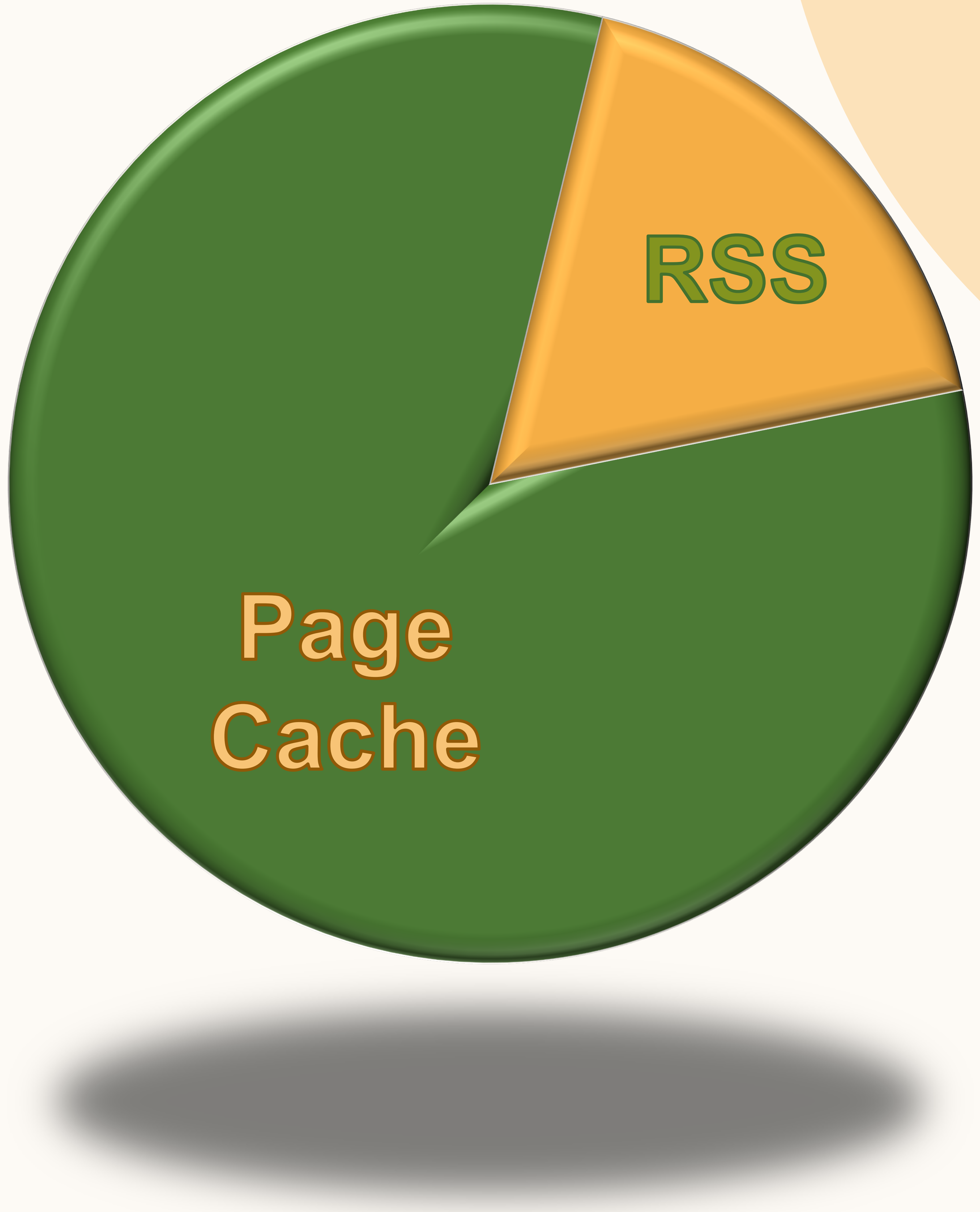
Troubleshooting

Here is how restarts or deploys work in our stack,

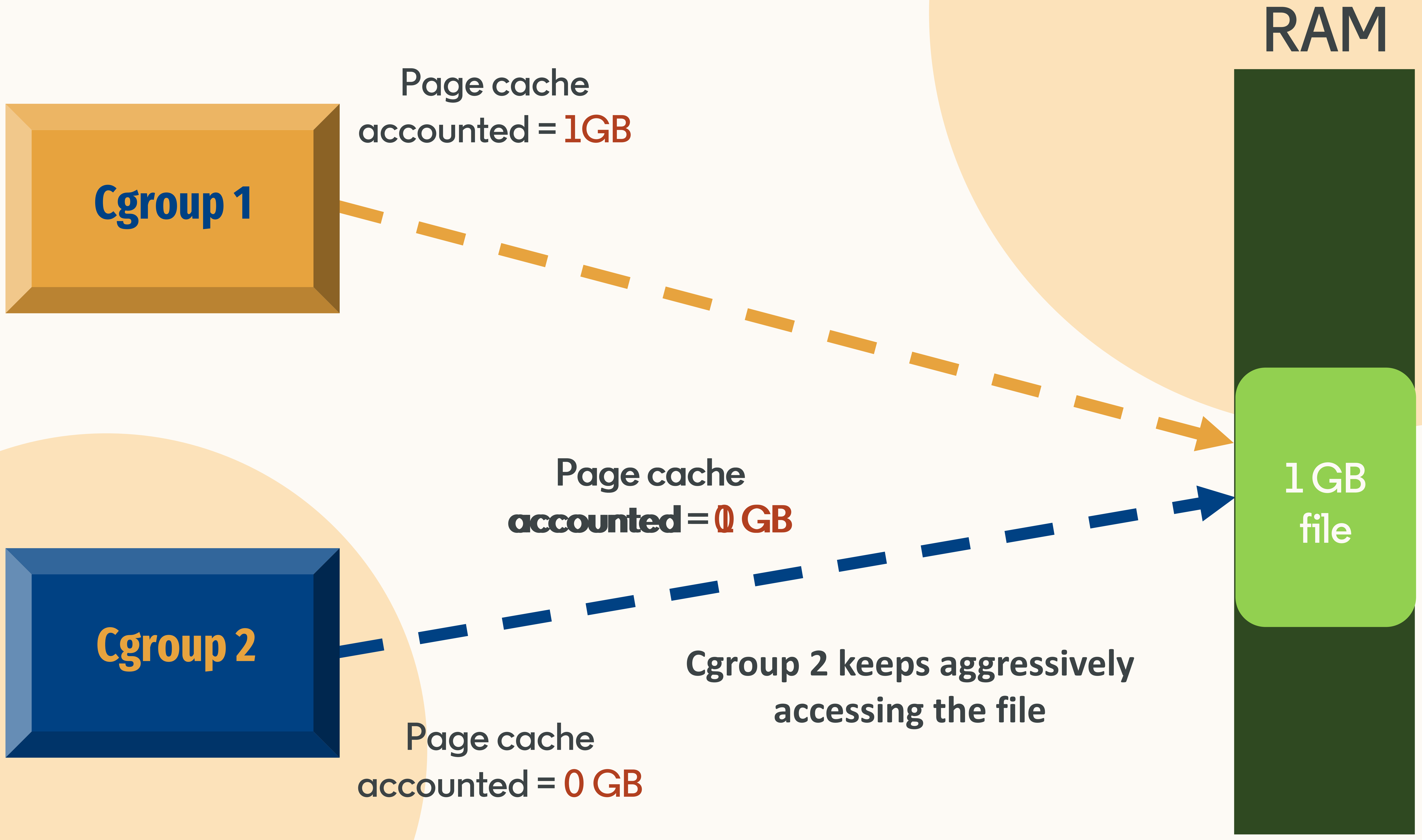
Stop application. ❯ Delete Application cgroup. ❯ Create Application cgroup. ❯ Start application.

Memory accounting in cgroups

- Shared pages are accounted for on First touch basis.
- If page exists in memory, it will eventually get accounted to the cgroup accessing it aggressively.



Sum of RSS of all the processes under that cgroup.


RAM
Page cache accounted = 1GB
Cgroup 1
Page cache accounted = 0 GB
1 GB file
Cgroup 2
Page cache accounted = 0 GB

Cgroup 2 keeps aggressively accessing the file

What happened in our Case

New Index Deployed
Cgroup limit reached
Automation restarts application
Actual Cgroup limit reached
Host low on memory
Host unreachable

Application Cgroup reports correct memory usage.

OOMkiller invoked and application is killed.

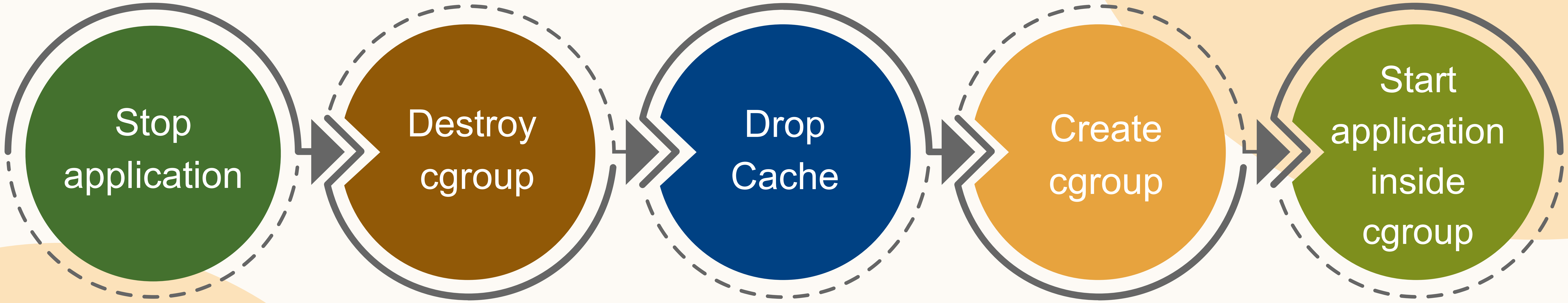
Application cgroup me mory reported is wrong.
New cgroup does not account for indexes already in memory.

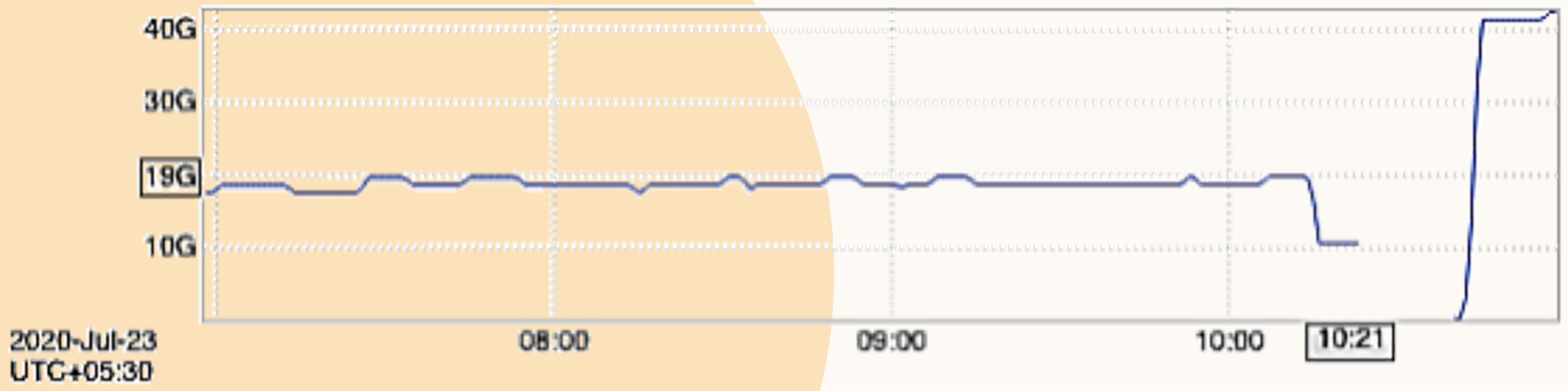
The OOMkiller is not invoked by Application cgroup.

Searcher uses mlockall() so cannot be swapped. Other system applications get swapped.

Validation

Did a small experiment to validate our findings.





Cgroup shows right amount of memory after the steps.

Verified that the issue was because a new application cgroup did not charge pages to itself even if the application inside it is the only one using it.

Mitigation

01

Set up proper monitoring

- Use sum of index size and RSS to set up an alert.
- Gives us enough time to react in cases of index growth

02

Limit memory of parent cgroup

- Total memory used shown for Parent cgroup is correct.
- OOMkiller will be invoked when the parent is breaching its limits.

Conclusion

01 **Memory Accounting in Cgroups**

Page Cache accounting is complicated

02 **mlockall()**

Can lead to critical services being swapped out.

03 **Never Assume Anything**

Thank You

Questions?
linkedin.com/in/minhajahammed94

Linked in