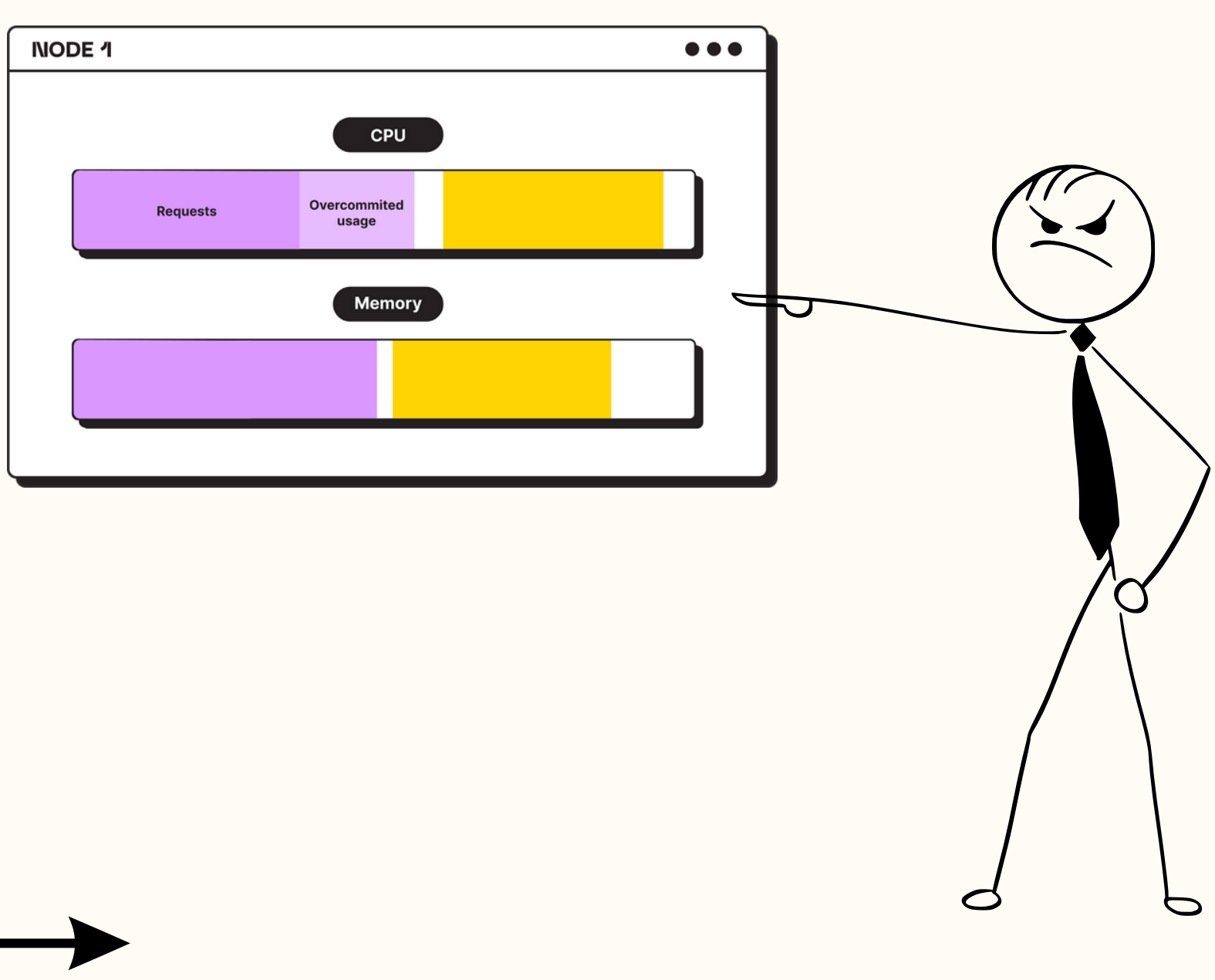
Preventing OOMKilled Errors in Kubernetes

If you don't set the correct memory limits and requests, and/or if K8s doesn't assign proper QoS classes for some reason...

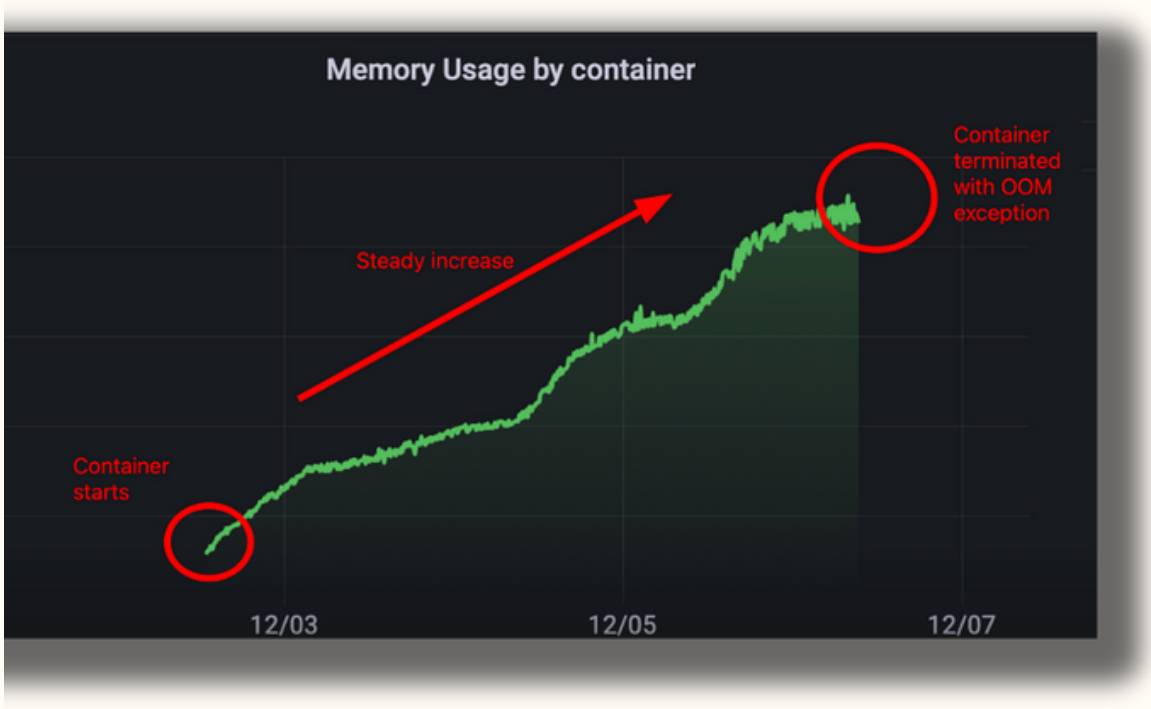


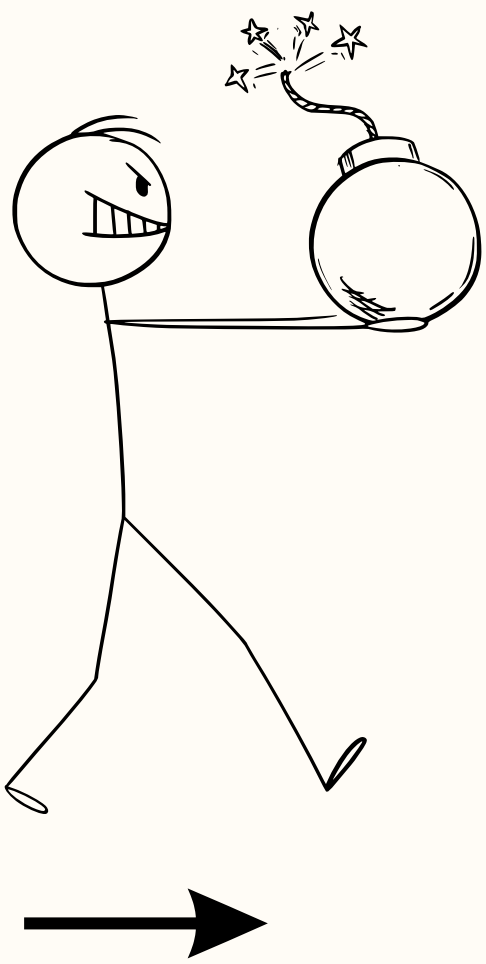
You could end up with an OOMkilled error

(or Out of Memory Killed error)

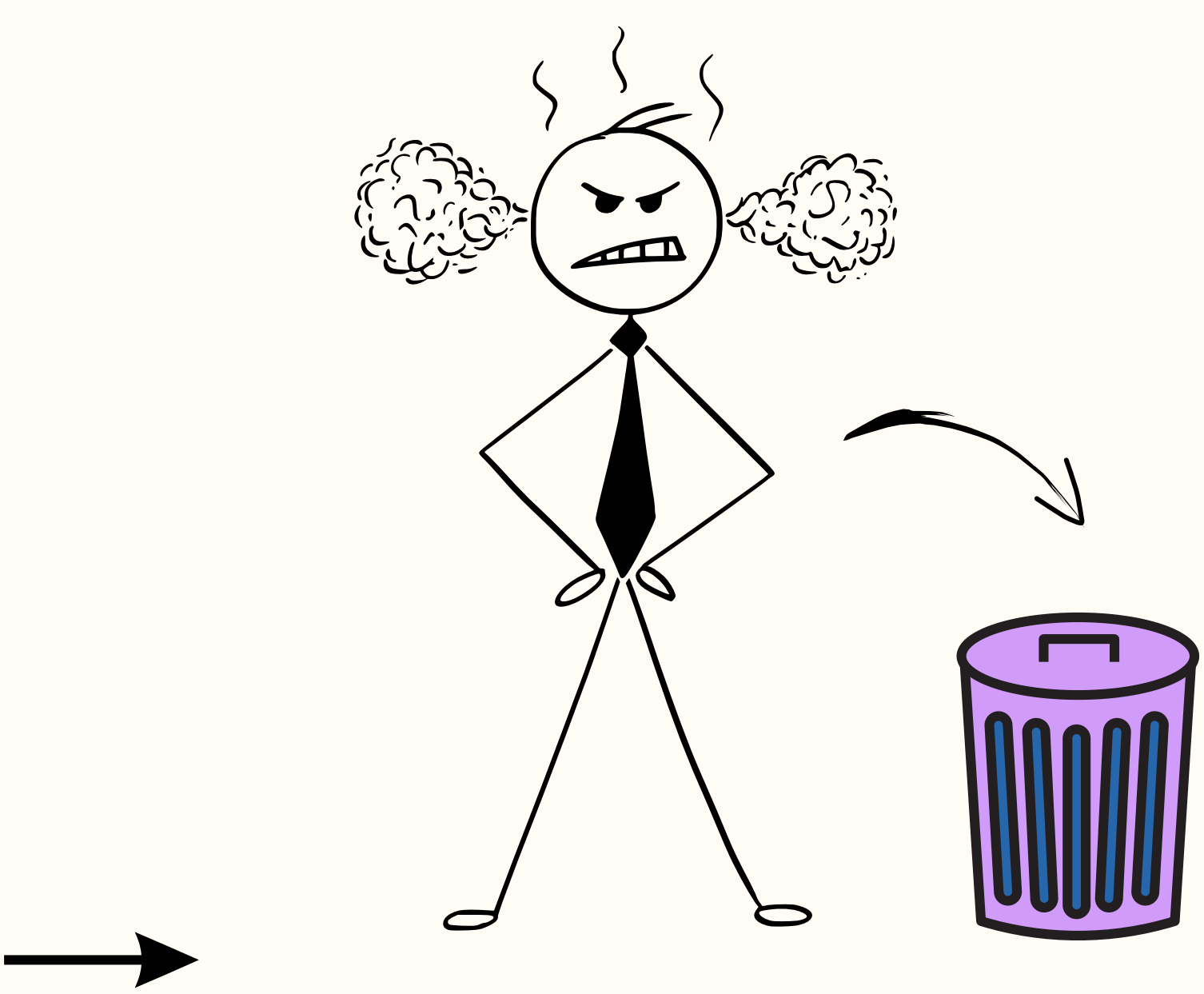
```
State:          Running
  Started:      Wed, 10 Mar 2021 11:35:31 +1100
Last State:     Terminated
  Reason:       OOMKilled
  Exit Code:    137
  Started:      Wed, 10 Mar 2021 08:06:03 +1100
  Finished:     Wed, 10 Mar 2021 11:35:30 +1100
```

OOMKilled errors occur when the Linux kernel's OOM Killer decides to terminate a process because it needs to free up some memory.



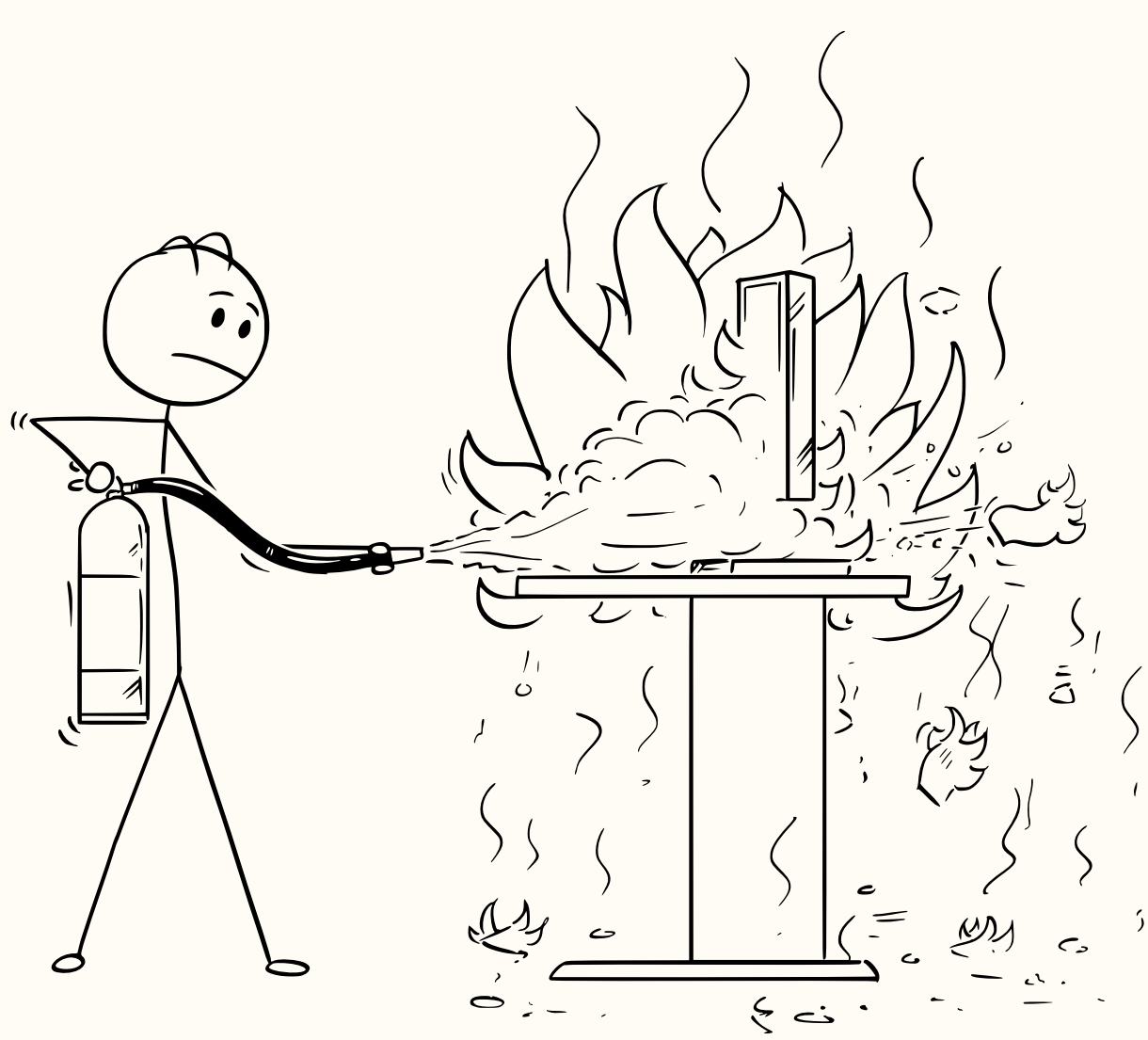


K8s should automatically move Pods to different nodes based on memory availability before it becomes necessary for one Pod's kernel to start terminating Pods due to low memory availability, but you still need to manage things right.



The way to manage memory inside K8s via limits and requests in such a way that OOMkilled errors won't happen because memory will always be distributed properly between containers and Pods.

Kubernetes can track the available memory on each node in a cluster and distribute Pods on different nodes accordingly.

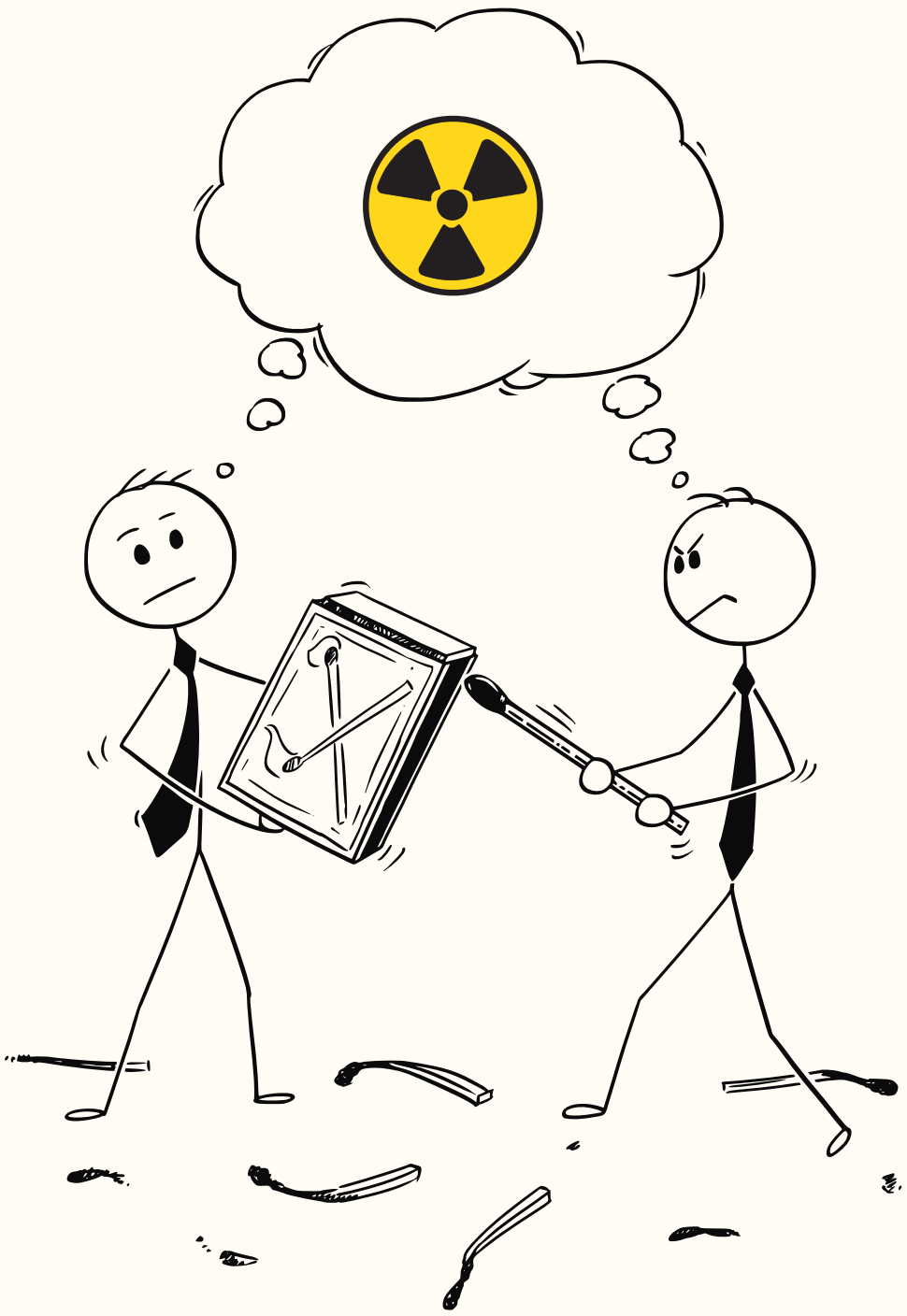


Set the limit too high?

You might find yourself in a “noisy neighbors” situation when some applications affect the performance of others.

Set the limit too low?

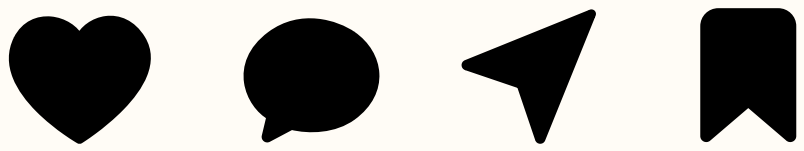
And your application pods might crash spontaneously without you even knowing.



As long as you set the right memory limits and requests – and assuming your cluster's total memory availability is sufficient to meet the needs of your workloads – you should be able to avoid OOMkilled errors, and still keep your deployments separated and stable.

Thanks for reading!

If you find this useful, don't forget to save it!



groundcover