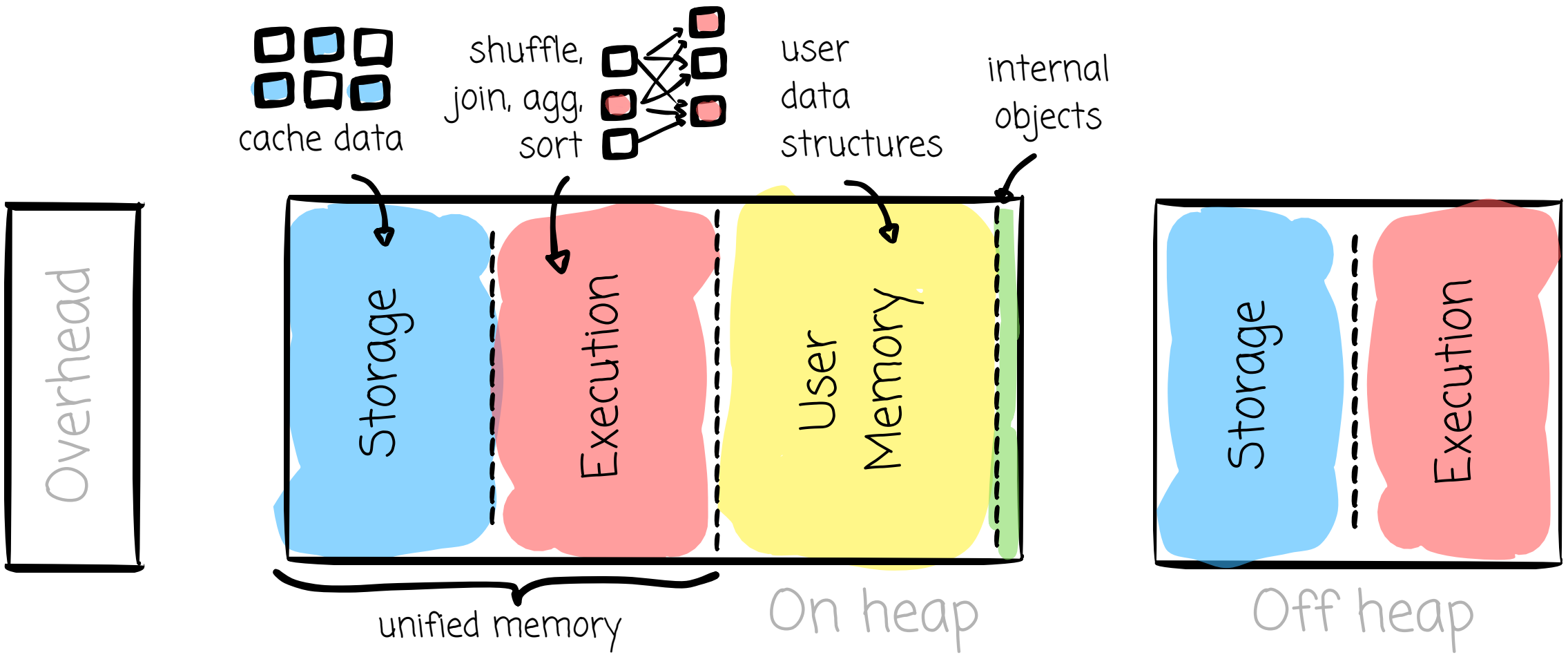


I spent 8 hours understanding Apache Spark's memory management

vutr.substack.com

Executors (ones that execute your Spark application) are JVMs processes

The executor has main regions for memory: on-heap, off-heap, and overhead.

On Heap

The JVM heap size define by
the spark .executor .memory.

The heap has:

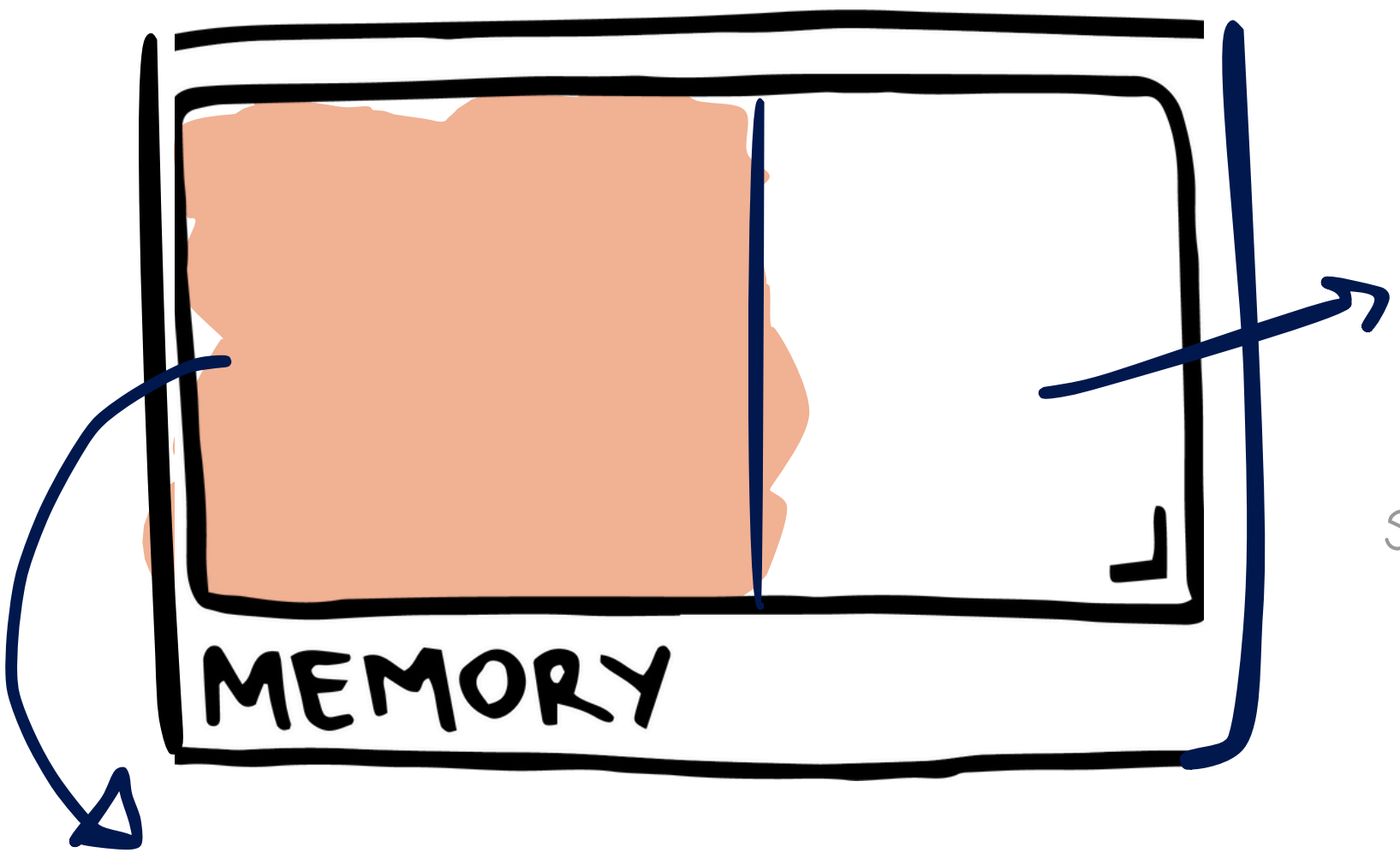
The reserved memory: for internal objects, hardcode 300MB

The user memory: user data structures (e.g., your hash tables or arrays) and Spark's internal metadata. Specified by:

spark.executor.memory * (1-spark.memory.fraction)

The remain is used for storage and execution, which is called the unified memory

spark.executor.memory



for internal metadata and user data structures.

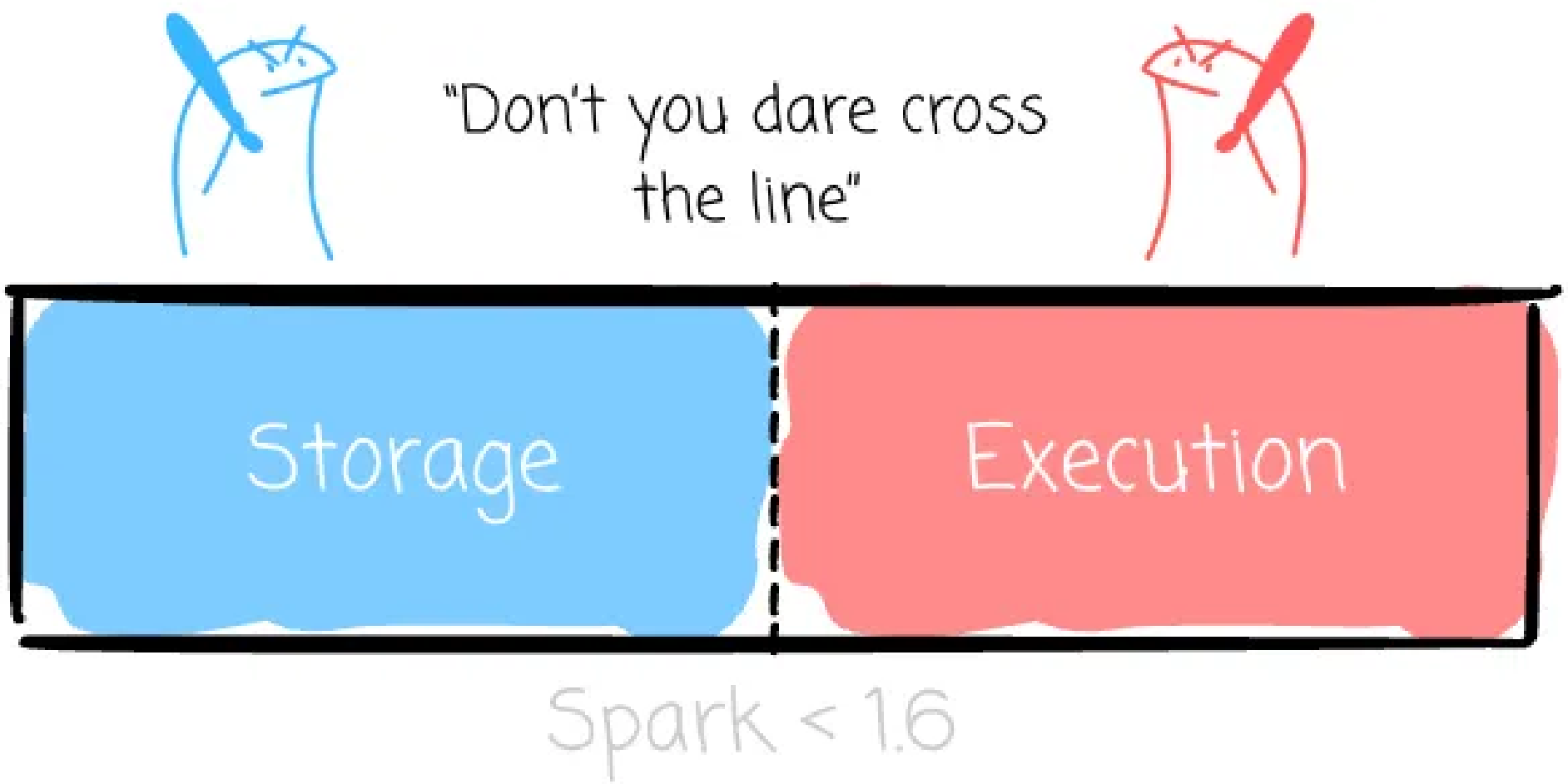
unified = spark.executor.memory * spark.memory.fraction

The execution memory is used for computing Spark transformations

while storage memory used for caching

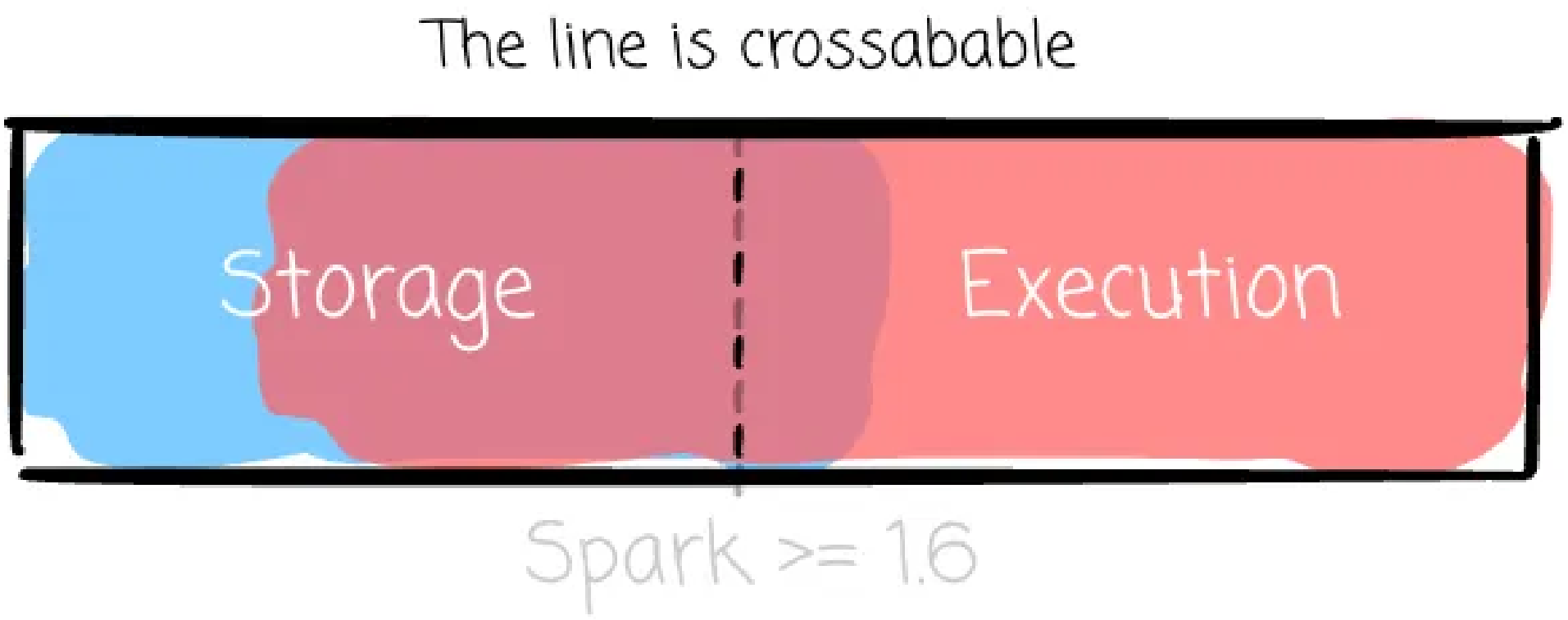
The
spark.memory.storageFraction
defines the boundary between
storage and execution.

Initially, this boundary is fixed; storage can't use space from execution, and vice versa.



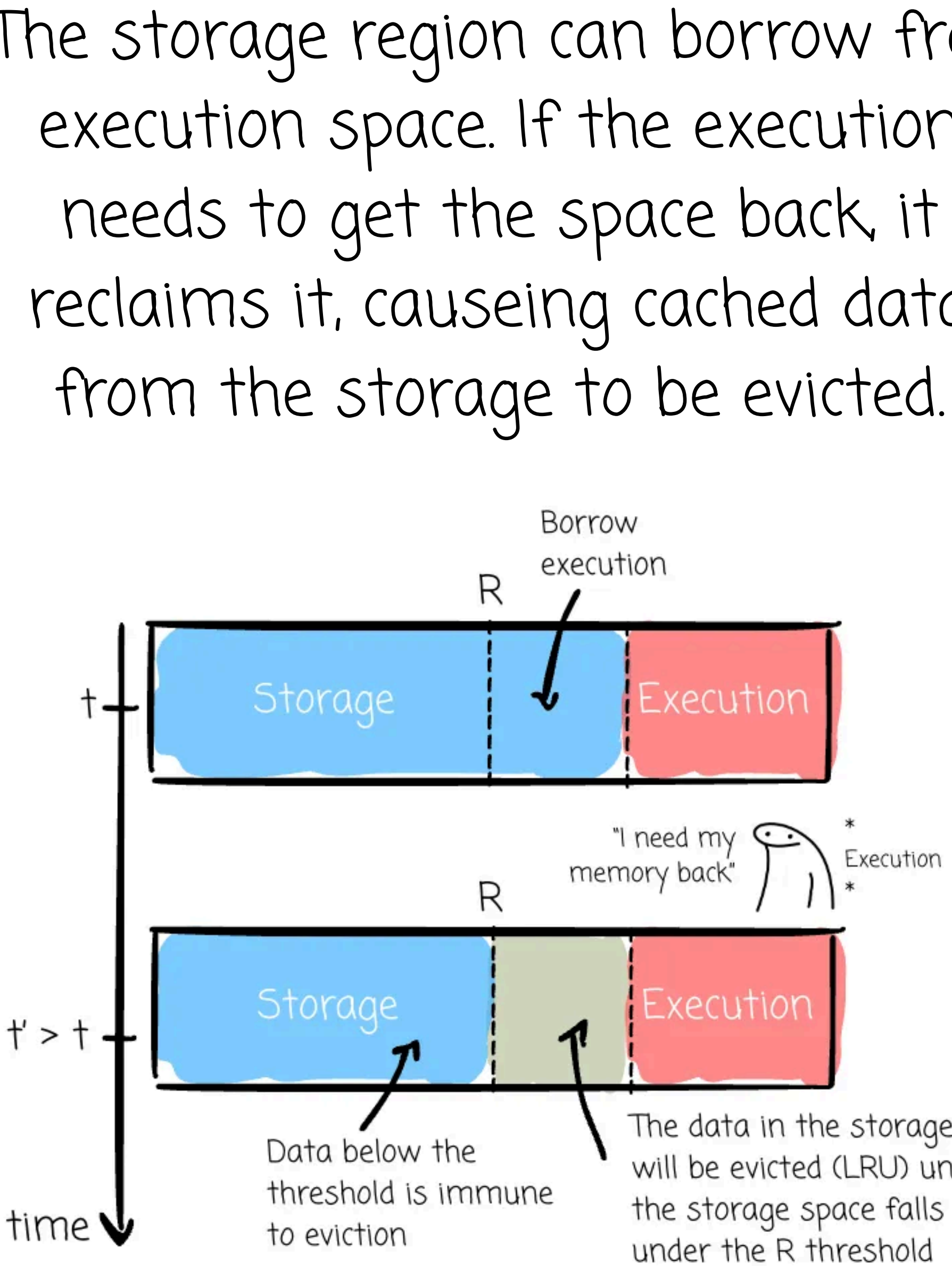
Spark < 1.6

Since Spark 1.6, the boundary between the execution and storage regions is crossable.

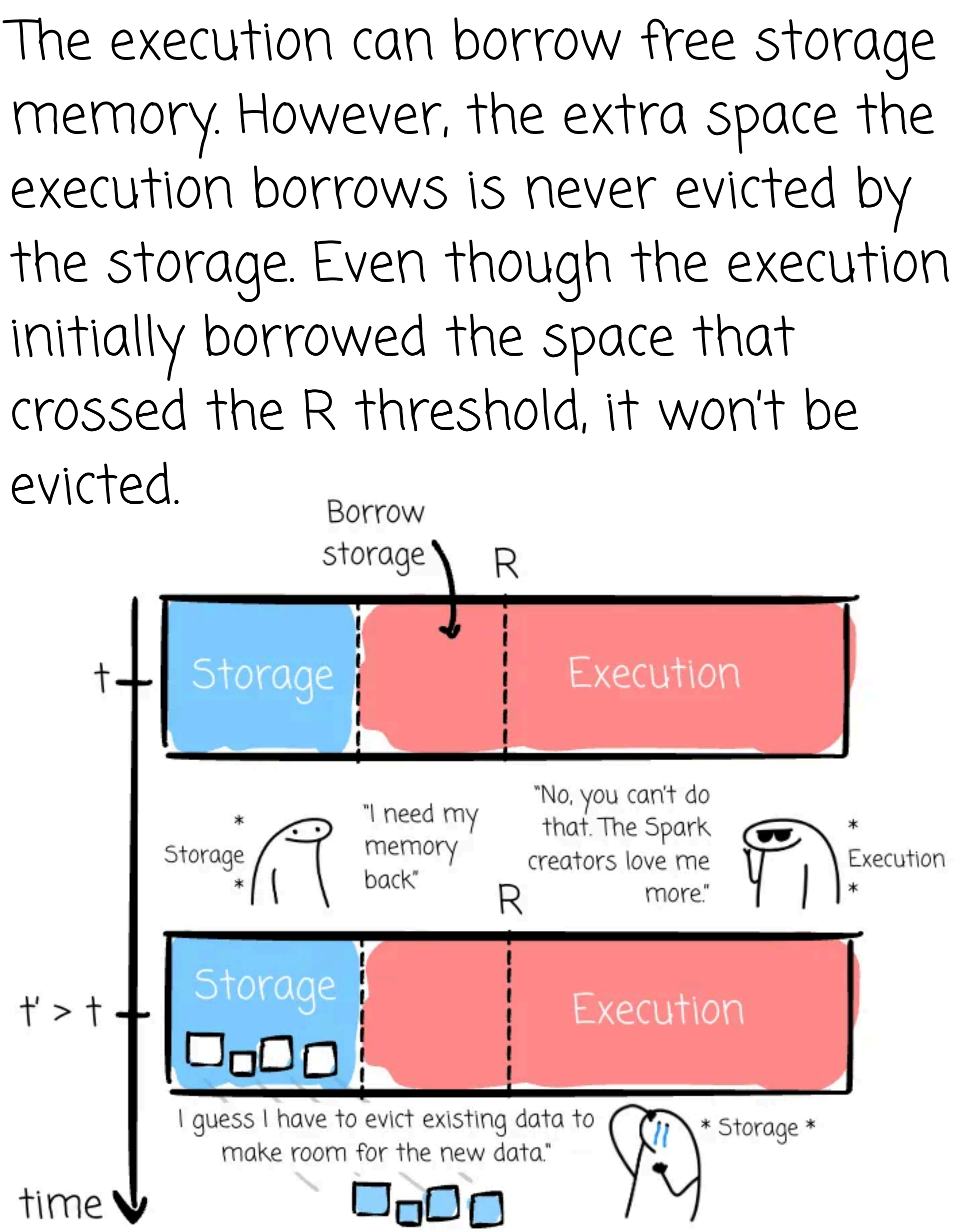


Essentially, the goal of the unified approach is to help the executor leverage resources more efficiently.

The storage region can borrow free execution space. If the execution needs to get the space back, it reclaims it, causeing cached data from the storage to be evicted.



The execution can borrow free storage memory. However, the extra space the execution borrows is never evicted by the storage. Even though the execution initially borrowed the space that crossed the R threshold, it won't be evicted.



Off Heap

The on-heap data is subject to the JVM garbage collection (GC) process. In addition, JVM's object has a significant memory overhead. A 4-byte string would have over 48 bytes in the JVM object.

To address the GC inefficiency and JVM object overhead, the project Tungsten introduces a memory manager that operates directly against binary data rather than Java objects.

Tungsen can work with the off-heap mode, which directly manages data outside the JVM.

Overhead
memory

This is the amount of additional memory to be allocated per executor process. It stores things like interned strings, VM overheads, other native overheads, etc.

👉 If you like this document, you will enjoy my 110+ data engineering articles repo and upcoming exclusive content that helps you enter the field and become a more impactful data engineer.

👉 Join 13,000+ DEs with the 50% discount here: https://vutr.substack.com