My implementation without CON is making ropies of values in the tight loop. This made the solution take a long time to finish.

Profiling shows my tode is spending a lot of time destroying objects.

This was happening in the tight loop.

STACK OF PAIRS OF PACKETS

We store references in the stuck. We derive an object and own it in the same stack. Making copies of all objects in the stack is costly.

COW is able to store the reference to an object. But it can also become the owner of a copy of the object. I can instruct a COW to construct and own an object as well.

It's the first and third use cases that would come in handy for implementing the stack.

Storing references in CoW requires use of std: reference_wrapper.

Alternatively, objects and references could live in the stock, wropped in shared ptr. Objects will reside in the heap. Their copies will not be necessary. Destroying the shared pointers will be cheap until the final references leave the scope.

An added borns is I save time implementing CoW. But there may be an overhead of keeping extra copies of references in memory.

This is probably not very big.

We derive a value into

a list of a single element

yhis is according

sometimes.

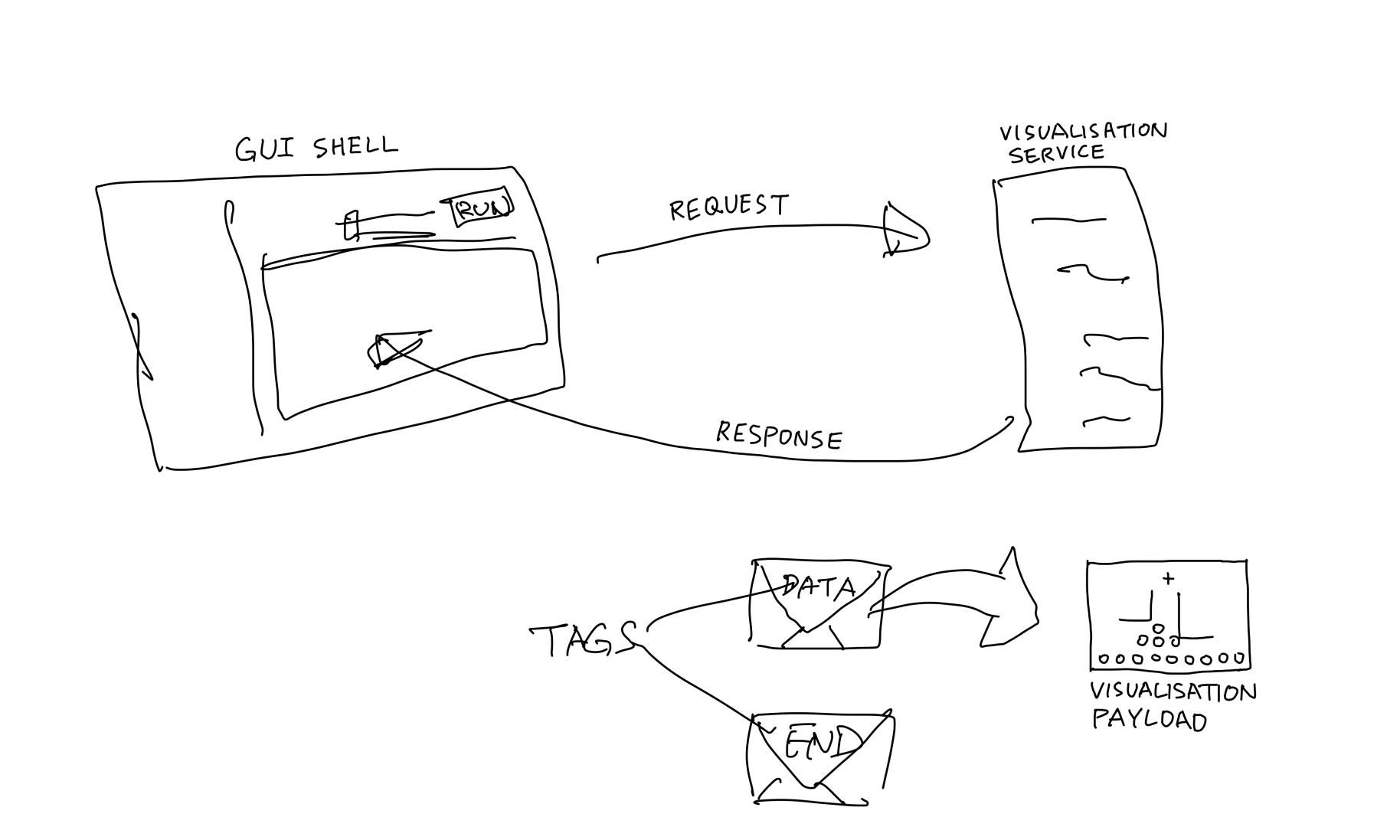
requirements.

d can implement a COW as a variant of the type and its reference type. The size of a COW object would be the size of the type plus the size of the metadata.

WATERFALLS CAVE

How easy is it to visualise the state of the algorithm?

How useful, or important?



CLASS HIERACHY OF CAVE, PRINTING PRESS AND ITERATOR

