# The Butcher

Applications that use this framework require what we call a *Butcher*. It is responsible for segmenting your inputs and merging your outputs in a way that allows for compatibility with the Parallel network. It consists of at least 2 required files, *"Split.py"* and *"Merge.py"*, you may include any other helper files. Split will provide the ability for the chief to serve data fragments on demand. Merge will combine your sub results into a combined result.

The guidelines described below are to help you understand the purpose this module plays in the system, and how to use it properly/safely.

## Split.py

The main concept is to use python generators to serve your inputs on demand. What is a generator you may ask. Put simply generators *yield* data instead of returning it like a normal function. That means specifically that the generator loads part of the data into memory and saves its state before yielding it and iterates to the next state. So if you know anything about python its that its super easy to do this in practice.

It involves defining a function called *splitter()* that segments your data or defines some kind of iterable input space. When you would normally append data to some kind of buffer in a parsing loop you can just yield the data to the caller like a super user.

## Merge.py

Merge takes all the work submitted back to the chief and combines the data however the user sees fit. You are responsible for implementing the strategy you think is appropriate for your application. Again, we are doing this lazily so that means another python generator.

This involves defining a function called *merger()* that takes result segments and maps them to a final result.

Our backend will import the methods you defined to segment data and merge results that are associated with the butcher when it receives requests from workers.

# General Requirements

NO malicious applications

No component should exceed some maximum recursion depth enforced by the chief

NO Trying to break out of the docker container or gain access to the local system.

NO running a server or proxy through workers.

NO creating docker containers or virtual machines.

NO access to external networks, like the internet

*Splitter()* and *merger()* methods each define a python generator using the *yield* keyword, this is like your main function. No other method in your package should yield anything.

Your generators do not produce infinite sequences.

The fragment size is smaller than some maximum size enforced by the chief.

## Consequences

Your application will not work, and the network will refuse to execute the job.

You will waste time writing an application that is not compatible with our architecture.

Suspicious activity might trigger the containers to reset.

You will waste resources and the Chief might blacklist your device’s MAC address.