"return" versus "yield" in CherryPy 2.1

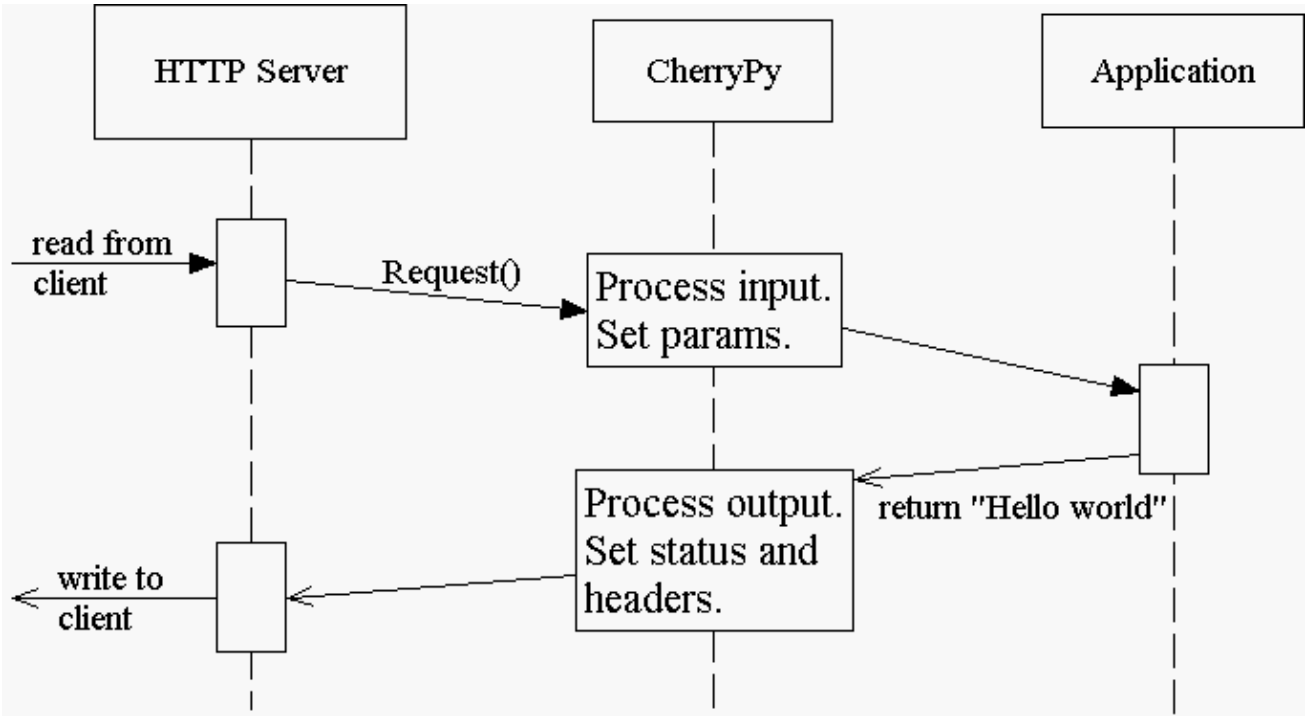
CherryPy handles HTTP requests, packing and unpacking the low−level details, then passing control to your application's *page handlers*, which generate the body of the response. CherryPy 2.1 allows you to return body content in a variety of types: a string, a list of strings, a file. CherryPy also allows you to *yield* content, rather than *return* content.

In general, you should use "return" instead of "yield".

The Generators and Yield tutorial should probably be updated to reflect this recommendation.

How "return" works with CherryPy

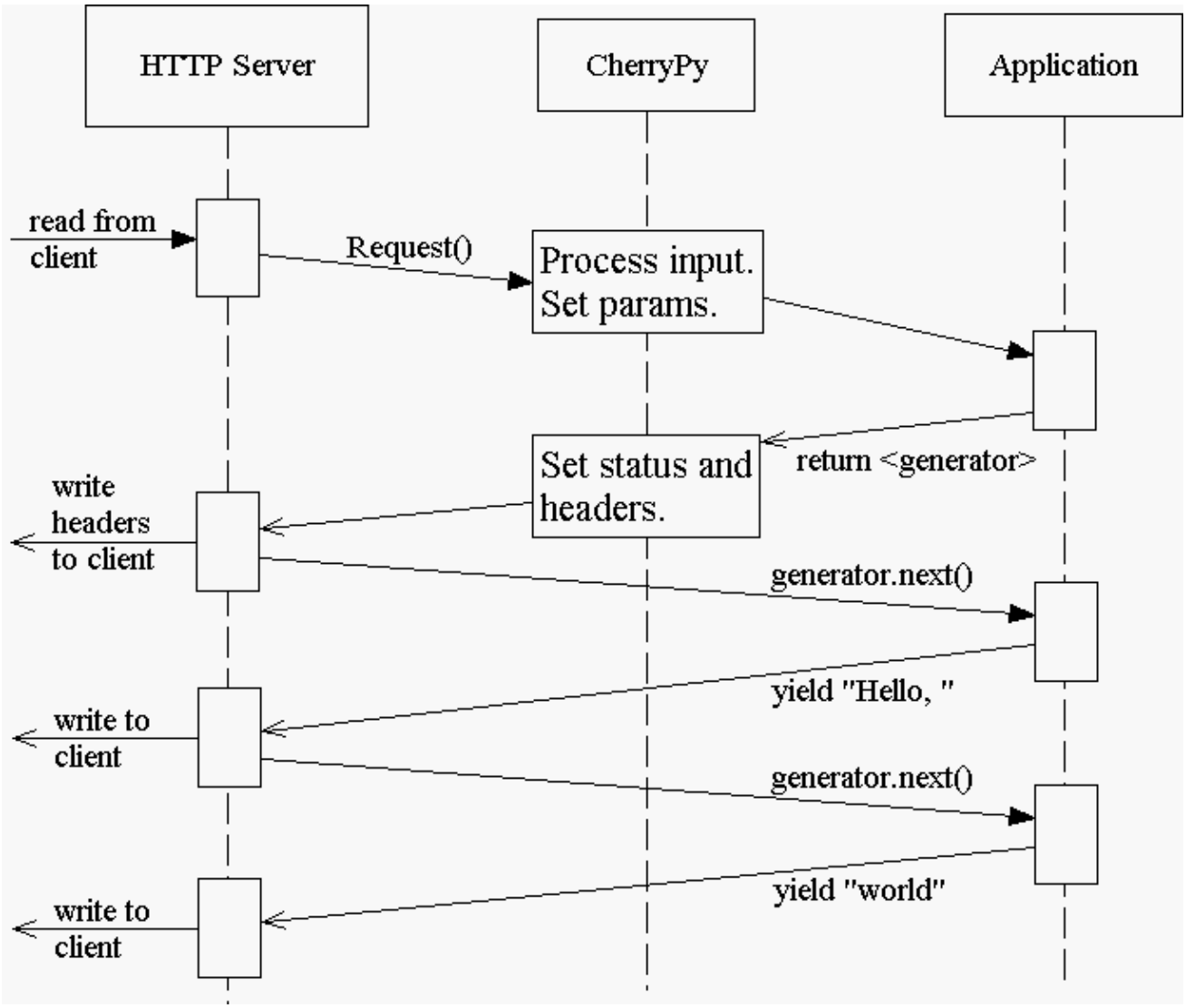
When you provide content from your page handler using the "return" statement, CherryPy manages the conversation between the HTTP server and your code like this:



Notice that the HTTP server writes everything to the client at once: status, headers, and body. This works well for static or simple pages, since the entire response can be changed at any time, either in your application code, or by the CherryPy framework.

How "yield" works with CherryPy

When you provide content from your page handler using the "yield" statement, CherryPy manages the conversation between the HTTP server and your code like this:



When you use yield, your application doesn't immediately pass raw body content back to CherryPy or to the HTTP server. Instead, it passes back a generator. At that point, CherryPy finalizes the status and headers, ***before*** the generator has been consumed, or has produced any output. This is necessary to allow the HTTP server to send the headers and pieces of the body as they become available.

Once CherryPy has set the status and headers, it sends them to the HTTP server, which then writes them out to the client. From that point on, the CherryPy framework steps out of the way, and the HTTP server essentially requests content directly from your application code.

Therefore, if an error occurs within your page handler, CherryPy will not catch it−−the HTTP server will catch it. Because the headers (and potentially some of the body) have already been written to the client, the server *cannot* know a safe means of handling the error, and will therefore simply close the connection (the current, builtin servers actually write out a short error message in the body, but this may be changed, and is not guaranteed behavior for all HTTP servers you might use with CherryPy).

In addition, you cannot manually modify the status or headers within your page handler if that handler method is a generator, because the method will not be iterated over until after the headers have been written to the client. ***This includes raising exceptions like NotFound, RequestHandled, InternalRedirect and***

HTTPRedirect. To use a generator while modifying headers, you would have to return a generator that is separate from (or embedded in) your page handler. For example:

```
class Root:
    def thing(self):
        cherrypy.headerMap['Content-Type'] = 'text/plain'
        if not authorized():
            raise NotFound(cherrypy.request.path)
        def content():
            yield "Hello, "
            yield "world"
        return content()
```

Generators are sexy, but they play havoc with a streaming protocol like HTTP. CherryPy allows you to use generators for specific situations: pages which take many minutes to produce, or pages which need a portion of their content immediately output to the client. Because of the issues outlined above, ***it is usually better to return content rather than yielding content***. Do otherwise only when the benefits of "yield" outweigh the risks.

HTTP/1.0

Final caveat: if either your HTTP server or your client are not HTTP/1.1, then CherryPy will ***not*** allow the server to "make its own requests" for yielded content. Instead, CherryPy will consume your generator, produce a new body (a single string), finalize headers, and pass them all to the HTTP server at once. In other words, it will behave as if you used "return" even though you did not. If you find your yielded pages do not display incrementally, check the HTTP version which your server claims to support; your page content may be being flattened by CherryPy in order to be HTTP/1.0–compliant.