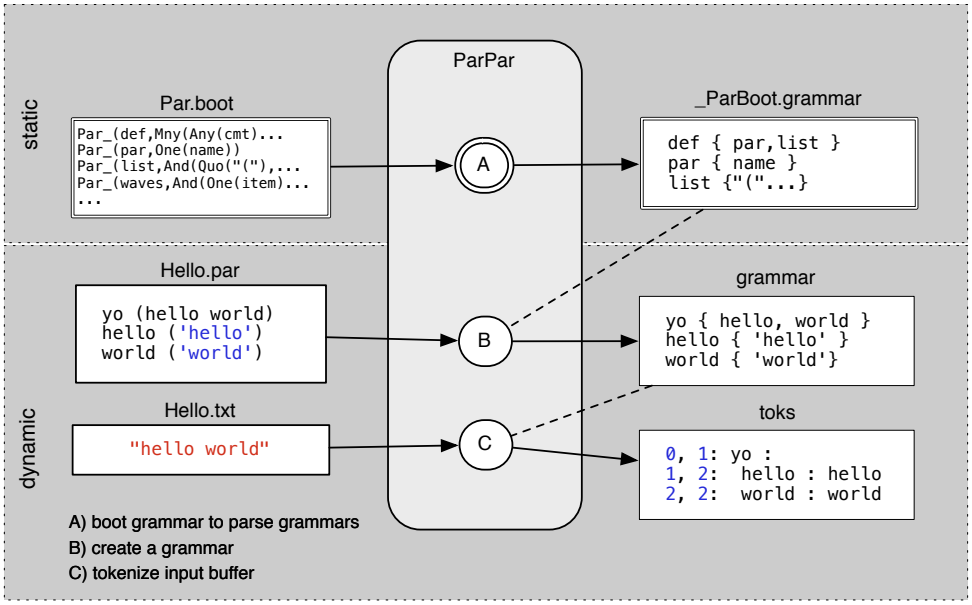
ParTest Walkthrough

The simplest and example of using Par is: **void ParTest::helloWorld()**

Which performs the following steps

A) Load the boot grammar that parses a new grammar. It never changes.
B) Create a new grammar: `parFile2Grammar("HelloWorld.par",…)`
C) Parse and make tokens: `txtFile2Tokens ("HelloWorld.txt",…)`

This is pretty useless example; we have a set of 3 tokens, with nowhere to go.



An example of using tokens is with the following test suite:

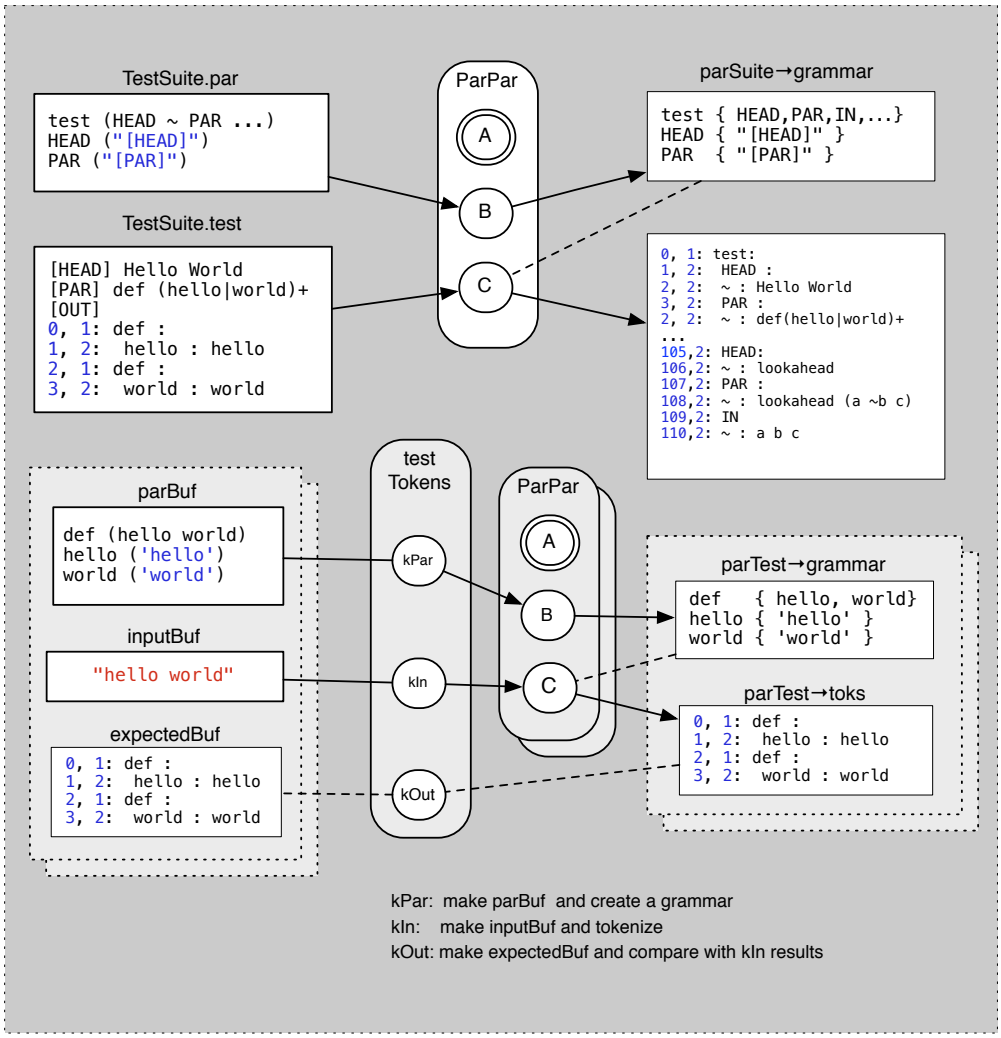
```
void ParTest::testSuite(const char*fname)
void ParTest::testTokens(ParPar *parSuite)
```

Which reads in a file with markdown and constructs a series of tests and compares the result. It is a bit circular in that we're using Par to test out Par with a set of small Par examples.

But now there are two instances of ParPar:

> parSuite: which reads at *.test file with markdown [HEAD], [PAR], [IN], [OUT]
> parTest: which is generated by the tokens resulting from markdown strings

parSuite read in a reads in a markdown file TestSuite.test and generates a set of tokens, which get interpreted by `ParTest::testTokens(ParPar *parSuite)` ,which in turn creates multiple instance of parTest to tokenize and compare.



The definition of the markdown is using an “~” to denote an island grammar

test (HEAD ~ PAR ~ IN ~ OUT ~ END?)+

Which looks for each of the markdown labels HEAD, PAR, etc. The interesting part is the token immediately follows, so we need to process the next token based on the previous context, like so:

```
void ParTest::testTokens(ParPar *parSuite) {
    …
    typedef enum { kHead, kPar, kIn, kOut} Context;
      …
      switch (tok->tokType) {

          case str2int("HEAD"): context = kHead; break;
          case str2int("PAR"):  context = kPar;  break;
          case str2int("IN"):   context = kIn;   break;
          case str2int("OUT"):  context = kOut;  break;
             …
          case str2int("~"): {

              switch (context) {

                  case kHead:
                  case kPar:
                  case kIn:
                  case kOut
```

Notice that we're switching on `case str2int("HEAD"):` for the tokens. Par automatically hashes the name of the token to a string.