S/SL Pipeline PL

- S/SL programs composed of “little” passes pipelined together to make a whole system
- Each pass is completely *isolated*
  - build and forget
  - no implicit dependencies between passes
- pass-based, isolated architecture is the epitome of recursive design (aka divide and conquer)
- passes are “bite-sized” actions
  - with explicit input and output APIs
  - sequenced by input token stream
- e.g. Concurrent Euclid - scanner, parser, semantic analyzer, allocator, emitter

S/SL Pipeline (Con’t)

- Passes emit streams of tokens (e.g. {token-id, text, line, offset})
- AST describes abstract syntax “tree” of valid token phrases
- CST describes concrete syntax “tree” of actual tokens for a given input
- S/SL doesn’t need trees - pipelines of tokens are equivalent to trees
- S/SL actions “hang off” of token CST
  - actions “scripted” by incoming token stream
  - “history” encoded in token stream, *f(t)* does not need to be ignored