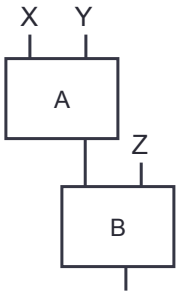
UNIT GENERATOR IMPLEMENTATION

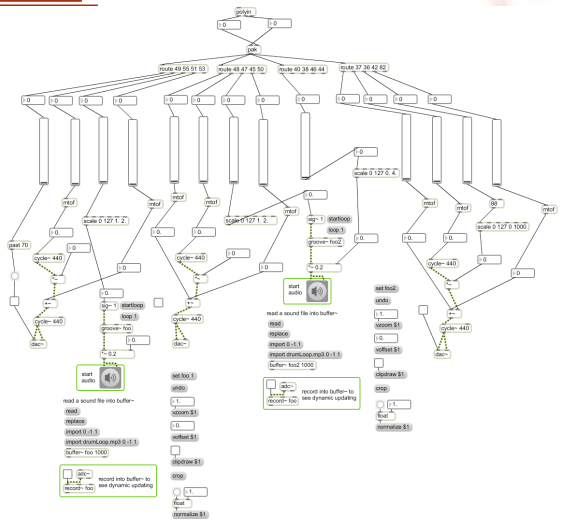
What's inside a Unit Generator and how do we access it?

Unit Generator Implementation

- Have to store the intermediate state somewhere
  - e.g. the current phase and frequency of an oscillator UG
  - therefore, Unit Generators are implemented as *objects* in Nyquist
  - Objects are accessed *implicitly* to provide samples – they are hidden from the user
- Many languages present (expose?) UG's as an *explicit* graph of objects.
  - A pass is made over the graph to propagate the next sample (or block of samples) from input to output



MAX MSP Example



Sounds in Nyquist

- In Nyquist, there's no direct access to the Unit Generators as objects
- Instead, *functions* make and return objects called SOUNDS
- Instances of Unit Generator objects are contained within sounds and called upon when samples are needed
- We'll learn more about SOUNDS later

Playing a Sound

- If you write `play` *sound-expression*
  - A sound is returned
  - Internally, the sound has a graph of unit generators
  - To play the samples, the graph is traversed, generating samples incrementally
  - The samples (in blocks of about 1000) are played in "real time"
- If you write `set` *var* `=` *sound-expression,* the entire sound might be computed, saved, and stored in memory