rf_pipelines: Proposal for a new library, and request for comments

Goals:

1. Make running bonsai easier, by hiding its internals

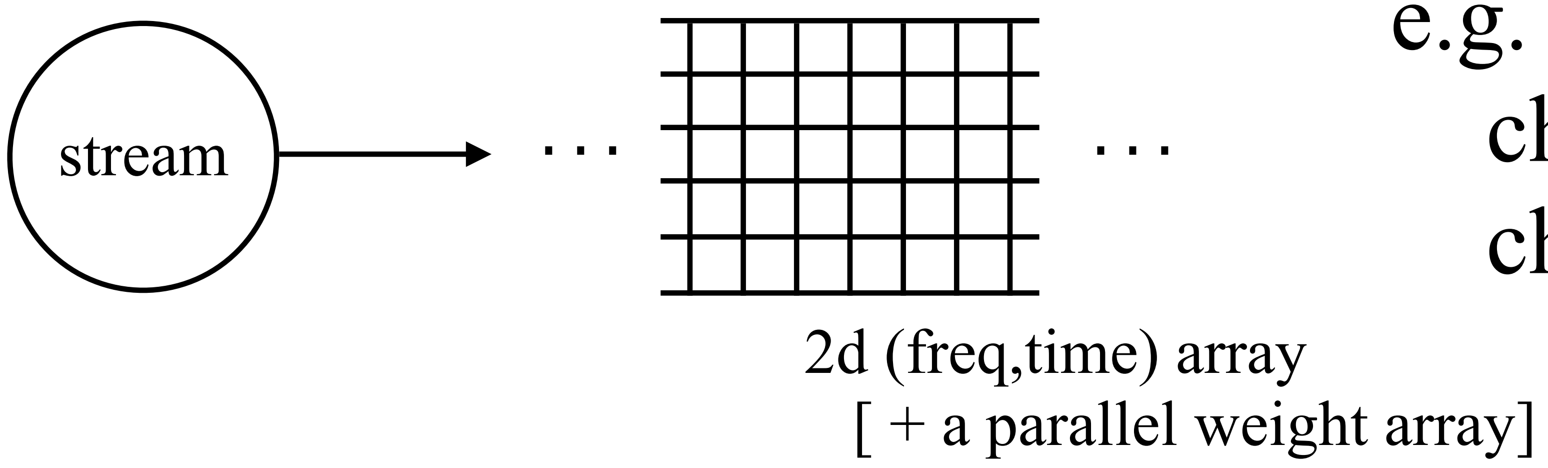
2. "Toolkit" library for RFI removal, oriented toward reuseable code and rapid prototyping

3. Framework for making collaboration easier

4. Help everyone work more efficiently (I hope!)

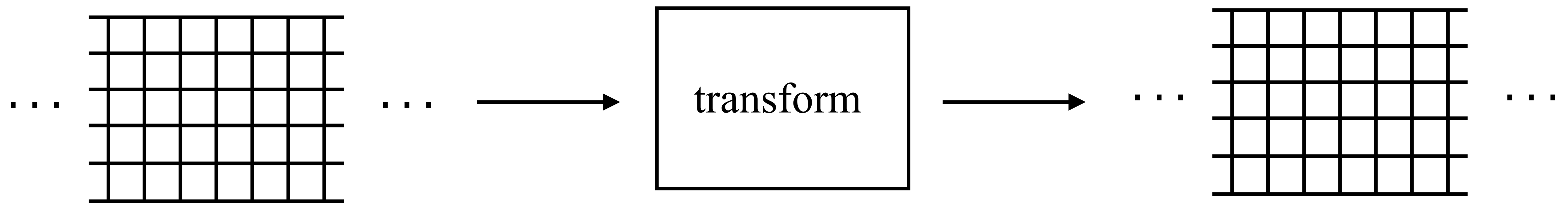
Proposal: library with two types of objects

wi_stream: an object which generates weighted intensity data in incremental chunks

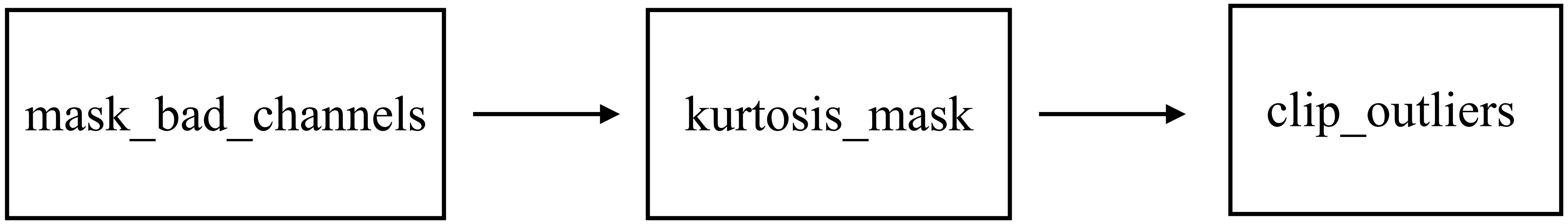


e.g. psrfits_stream
chime_file_stream
chime_network_stream

wi_transform: an object which operates on and modifies weighted intensity data in incremental chunks



wi_transforms can be composed.
e.g. RFI removal algorithm built out of building blocks



Over time, we can build up a library of reuseable components, allowing rapid prototyping in the future.

Facilitates collaboration by making it easier for everyone to use each other's code.

wi_transforms can be prototyped in python and later re-implemented in C++ for speed.

Library requirements:

- Streams/transforms can be written in either C++ or python, and freely intermix.
- No performance penalty if all streams/transforms are C++
- Transforms should be allowed to define their chunk size, prepad size, and postpad size. The rf_pipelines library will provide the needed buffering/rechunking.
- A chain of existing streams/transforms should be runnable with a few-line python script. Something like this:

```
s = rf_transforms.chime_file_stream('acqdir')
t1 = rf_transforms.simple_detrender()
t2 = rf_transforms.burst_search_rfi_remover()
b = rf_transforms.bonsai_search('file.cfg')
rf_transforms.run(s,t1,t2,b)
```

Some obvious streams/transforms:

- file stream
- network stream
- detrending transform
- RFI masking transform (or building block)
- FRB search “transform” (doesn’t actually modify intensity)

Some not-so-obvious streams/transforms:

- plotter “transform”

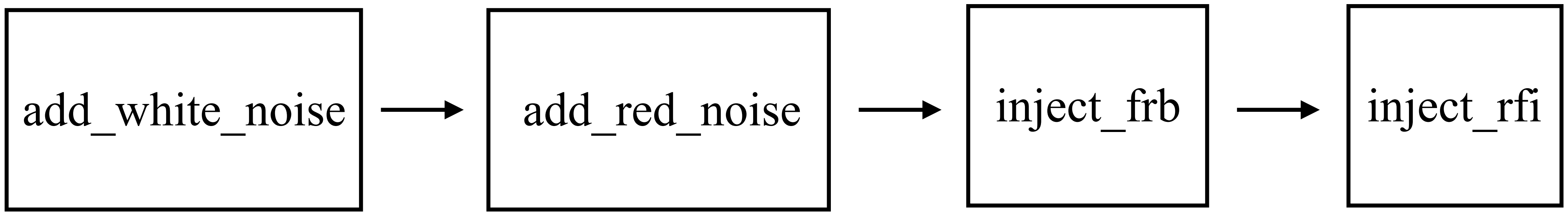
Doesn’t actually modify the intensity data, but writes summary plots (waterfalls, RFI histograms) to disk.

By inserting multiple plotters in a transform chain, can watch the filtering progress.

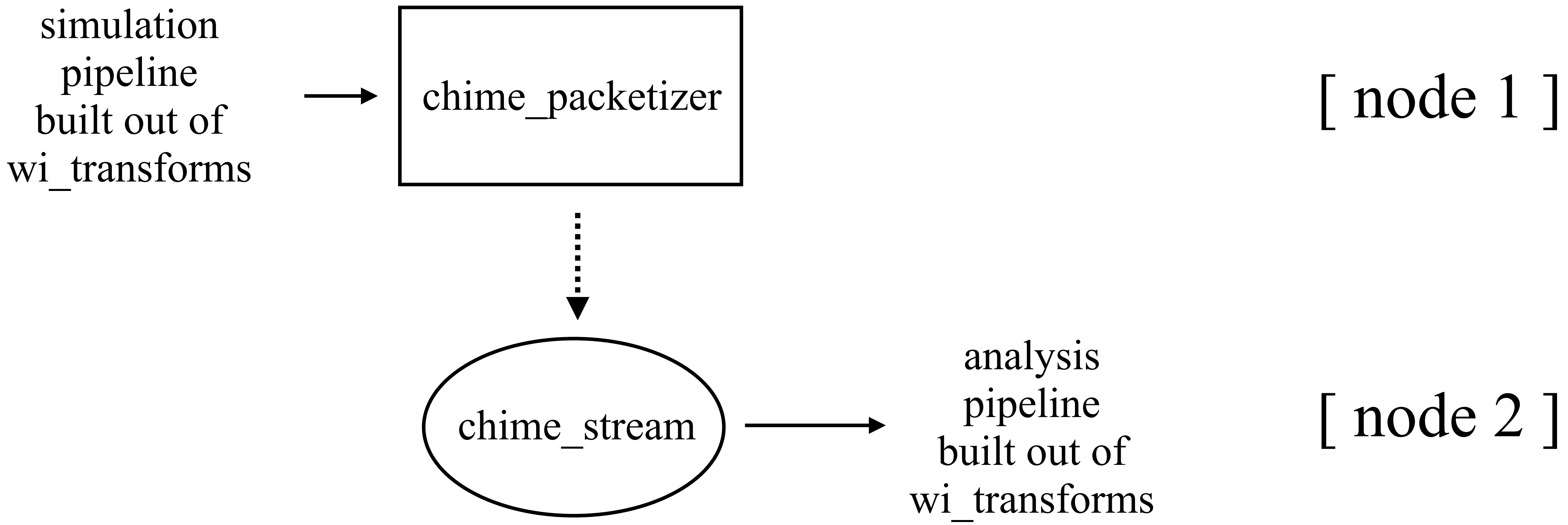
“Polished” plotting code can be written once and run at any intermediate point, e.g. for debugging

Some not-so-obvious streams/transforms:

- transforms for simulating data



- network packetizing transform


simulation pipeline built out of wi_transforms
chime_packetizer
[node 1]
chime_stream
analysis pipeline built out of wi_transforms
[node 2]

Proposed next steps:

- Write core rf_transforms library: base classes, buffering/ rechunking logic, C++/python interoperability
- Move some code from bonsai to rf_transforms: the input_stream class, simple detrender, psrfits reader, chime hdf5 reader, plotters. These become wi_streams or wi_transforms.
- Write bonsai_search transform

Existing code which could be added right away:

- FRB simulations (Kendrick frb_olympics)
- Detrenders, RFI-removing transforms (Kiyo burst_search)
- gbncc-inspired RFI removal (Alex J, Andy)
- FDMT? (Alex J)
- Ambitious: presto rfifind logic?

Going forward, try to implement new code as wi_transform whenever possible, build up library of useful transforms.

Once this gets traction, a task like

> "I want to take gbncc data, add simulated FRB's, run Kiyo's detrender, run Alex's RFI remover, then run a bonsai search"

should be implementable with a few-line python script.

Should be particularly useful for newcomers to the project, or for distributing tasks.

[Your comments here!]