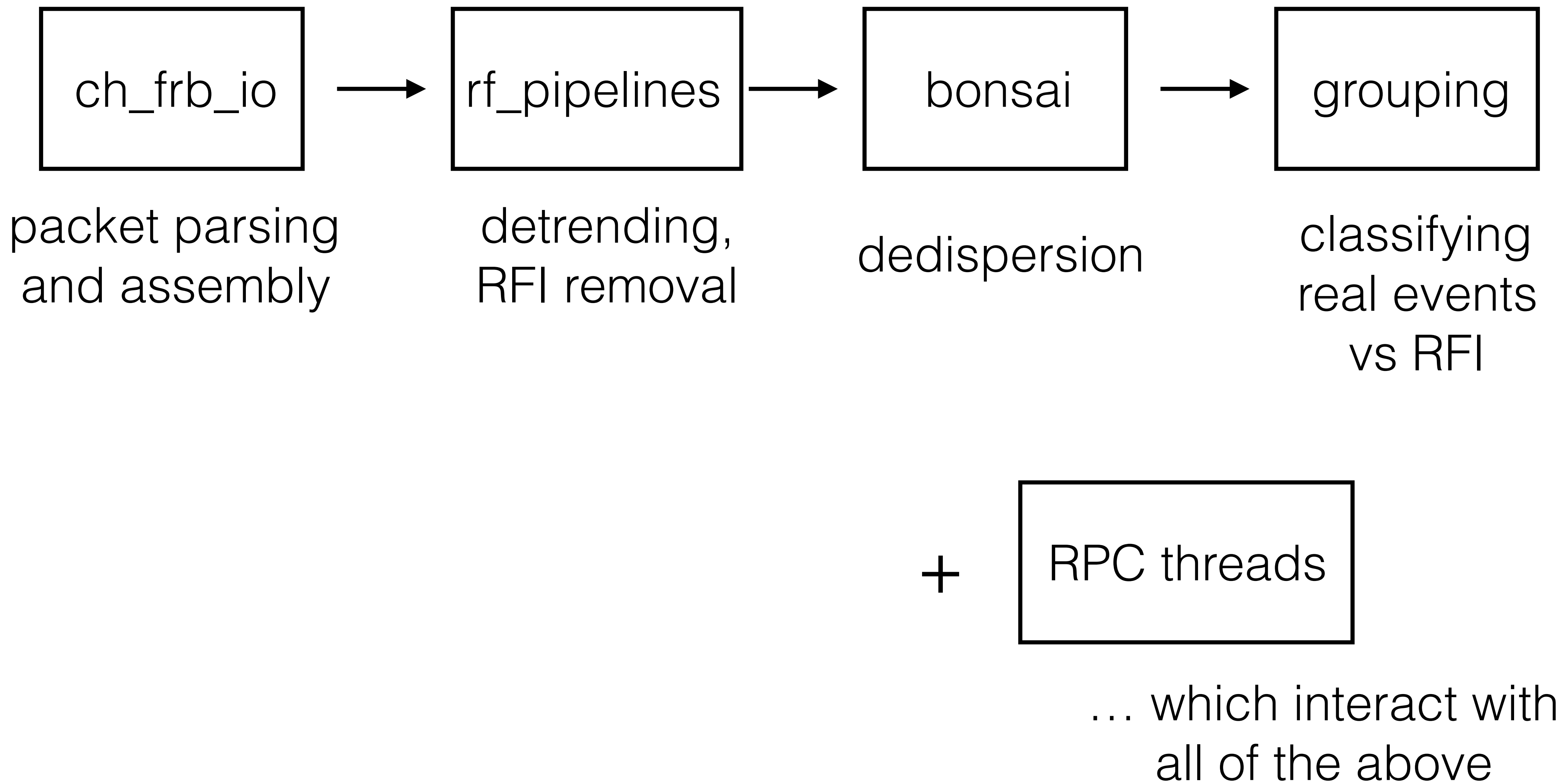
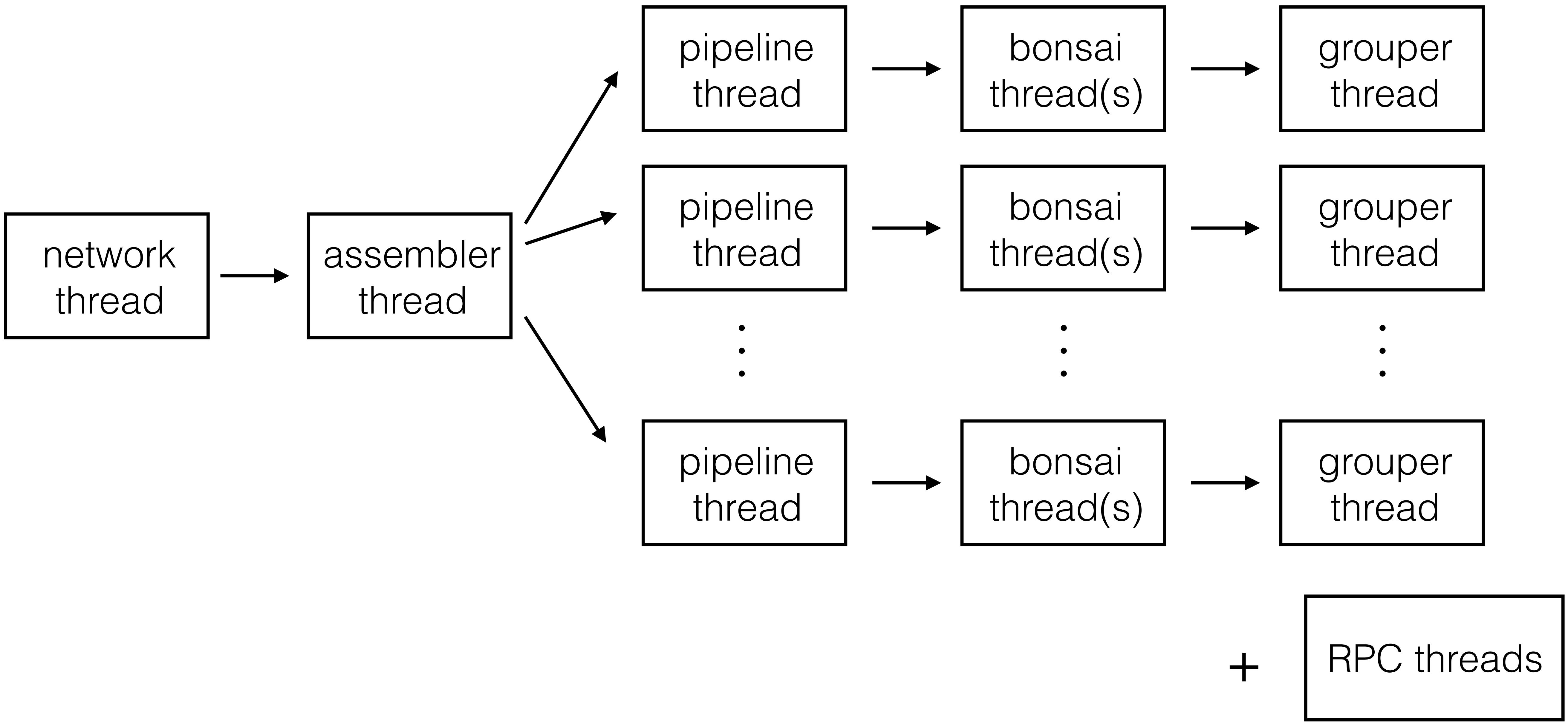
L1 network code

KMS, Sept 2016

L1 software components



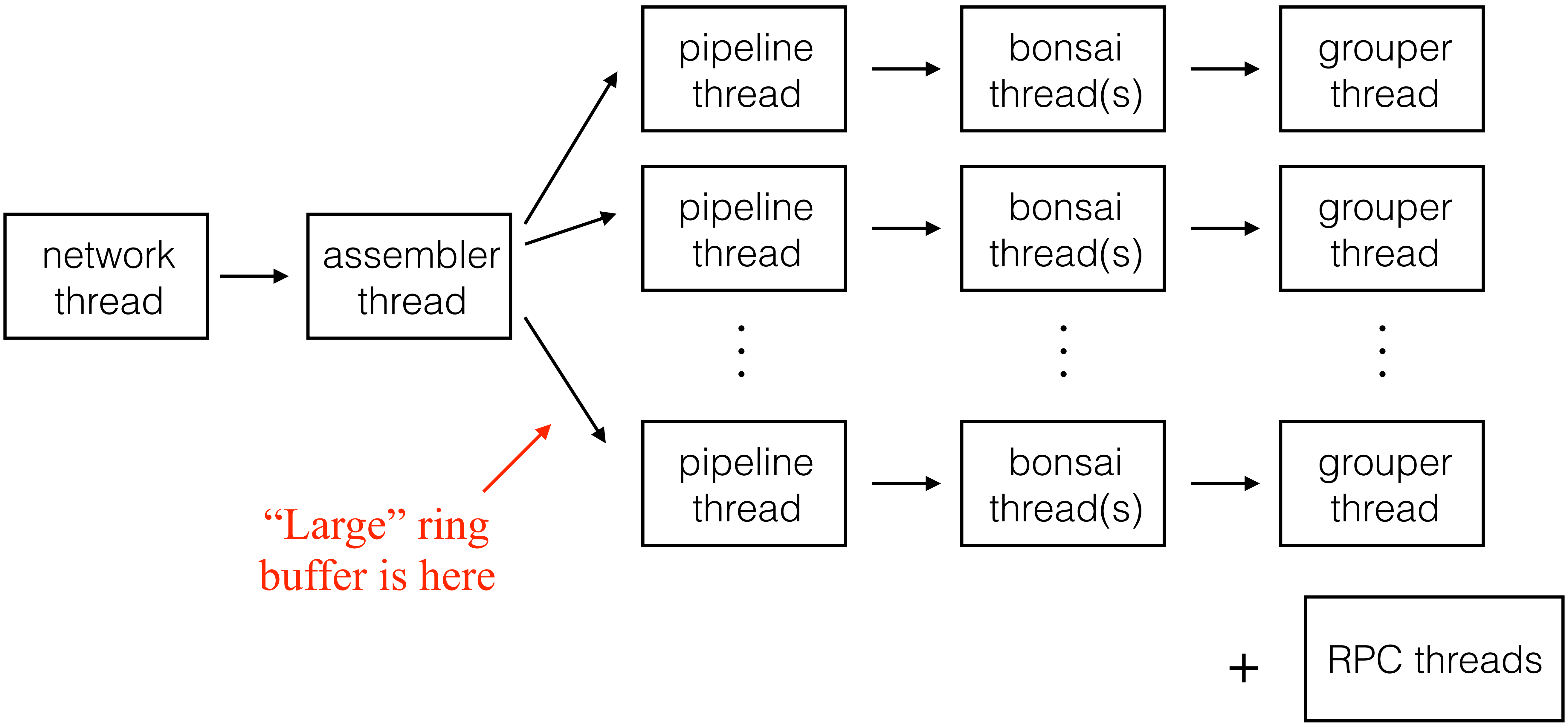
L1 thread model



Network and assembler threads now fully implemented:

- Network thread just reads packets from network into a “small” ring buffer
- Assembler thread parses packets and “assembles” into regular per-beam arrays

L1 thread model



A "large" ring buffer has been implemented between the assembler and pipeline threads. As discussed last time, this will give maximum flexibility when we implement RPC's:

- RPC can flush beam to local disk, or retrieve over network
- RPC can request an arbitrary subarray, including "slice" around fiducial DM.

Performance

After a lot of optimization, the network and assembler threads take 0.5 cores on frb1 (3.5 GHz CPU) at the full CHIME data rate (1 Gbps).

Assuming we use a 2.2 GHz CPU in the real L1 nodes, predict that the networking code will use 0.8 cores.

Optimizations include assembly language kernels which assume AVX2 instruction set and nt_per_packet=16 (would need to be revisited if this number changes).

Event counting

The network code counts events of different types:

```
[kendrick@psrcontrol ch_frb_l1]$ ./ch-frb-l1
ch_frb_io: listening for packets on port 10252
ch_frb_io: assembler thread exiting
    bytes received (GB): 10.0002
    packets received: 1177049
    good packets: 1177048
    bad packets: 0
    dropped packets: 0
    end-of-stream packets: 1
    beam id mismatches: 0
    first-packet mismatches: 0
    assembler hits: 9416384
    assembler misses: 0
    assembled chunks dropped: 0
    assembled chunks queued: 576
```

Right now we don't do anything with them except print them at the end, but there is a hook in the C++ API for real-time retrieval via RPC.

The idea is that we eventually want a web-based "dashboard" which can show these event counts and other diagnostic info, to give a visual summary of the FRB backend status.

Networking code to do list

Short-term goal: develop the L1 pipeline enough that pipeline timings can be used to make final decisions about what hardware we need.

Current networking code should be good enough for this purpose.
Looking slightly beyond this short-term goal, there are some loose ends!

- implement RPC's
- do something useful with event counts
- implement bitshuffle-compression of packets
- handling of "soft" failure modes, such as temporary correlator crash or full ring buffer somewhere in chain
- simulation code which can send "interesting" simulated timestreams to the L1 node for testing (code exists but is too slow, needs parallelization)

+ lots of minor or "internal" loose ends (see ch_frb_io/README)