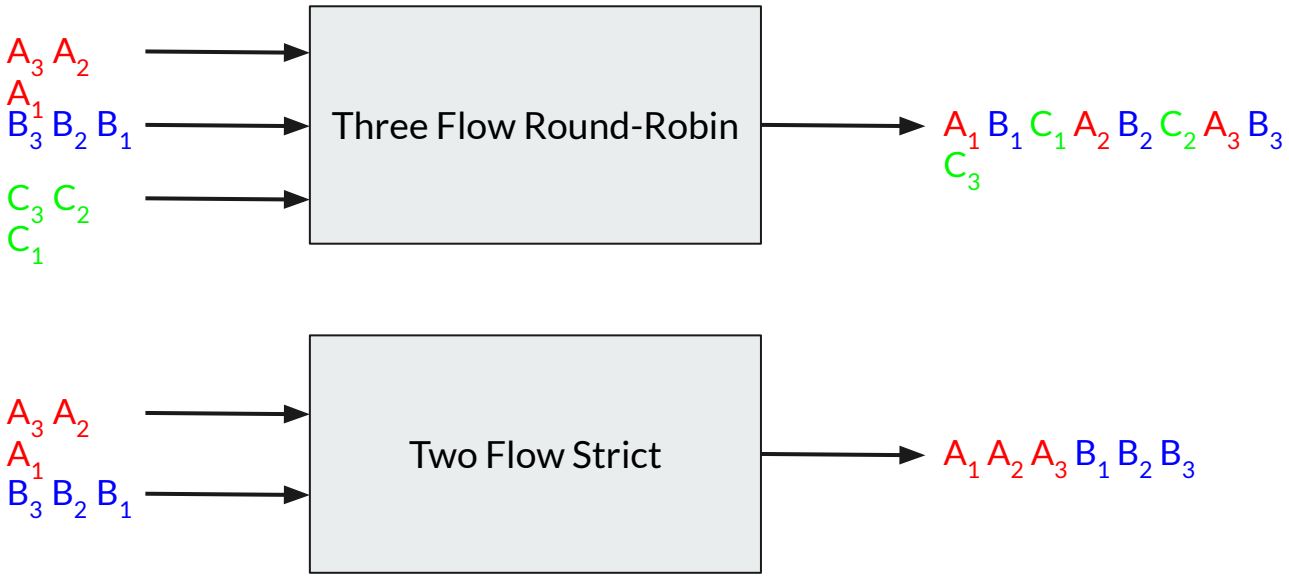
Rio – Programmable Packet Scheduling

Akash & Kabir

Scheduling Policies



Types of Policies

Set-To-Stream Policies

Work with all packets and all classes.

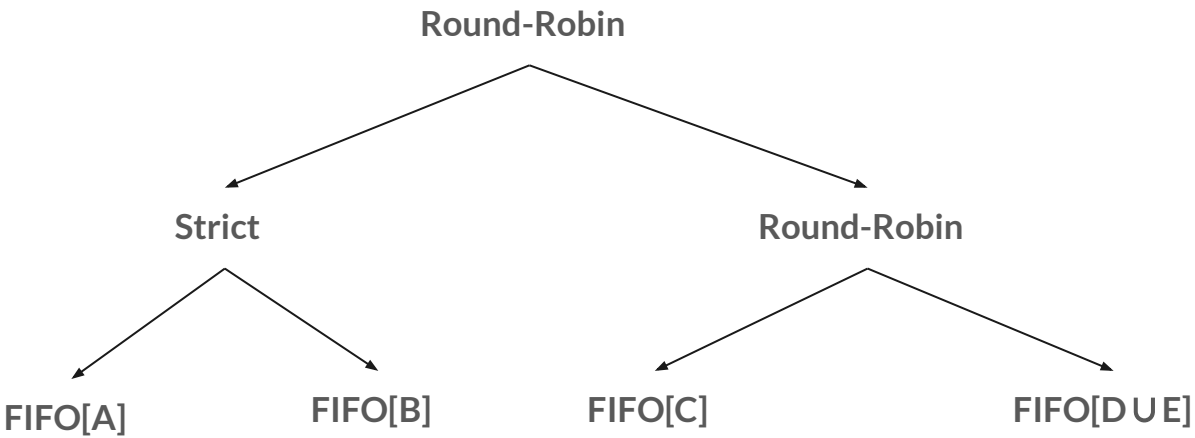
- **First-In First-Out**
- **Earliest-Deadline First**
- **Shortest-Journey Next**

Stream-To-Stream Policies

Operate over certain flows depending on the policy.

- **Round-Robin**
- **Strict**
- **Weighted-Fair Queueing**

Trees?



Rio Syntax

```
classes A, B, C, D, E;

left = strict[fifo[A], fifo[B]];
right = rr[fifo[C], fifo[union[D, E]]];

policy = rr[left, right];

return policy
```

- Names for collections of packets
- Partitions set of all packets: i.e.
  - each class is disjoint
  - every packet belongs to some class

- Classes are sets (of packets)
- We can take unions of sets to build larger sets

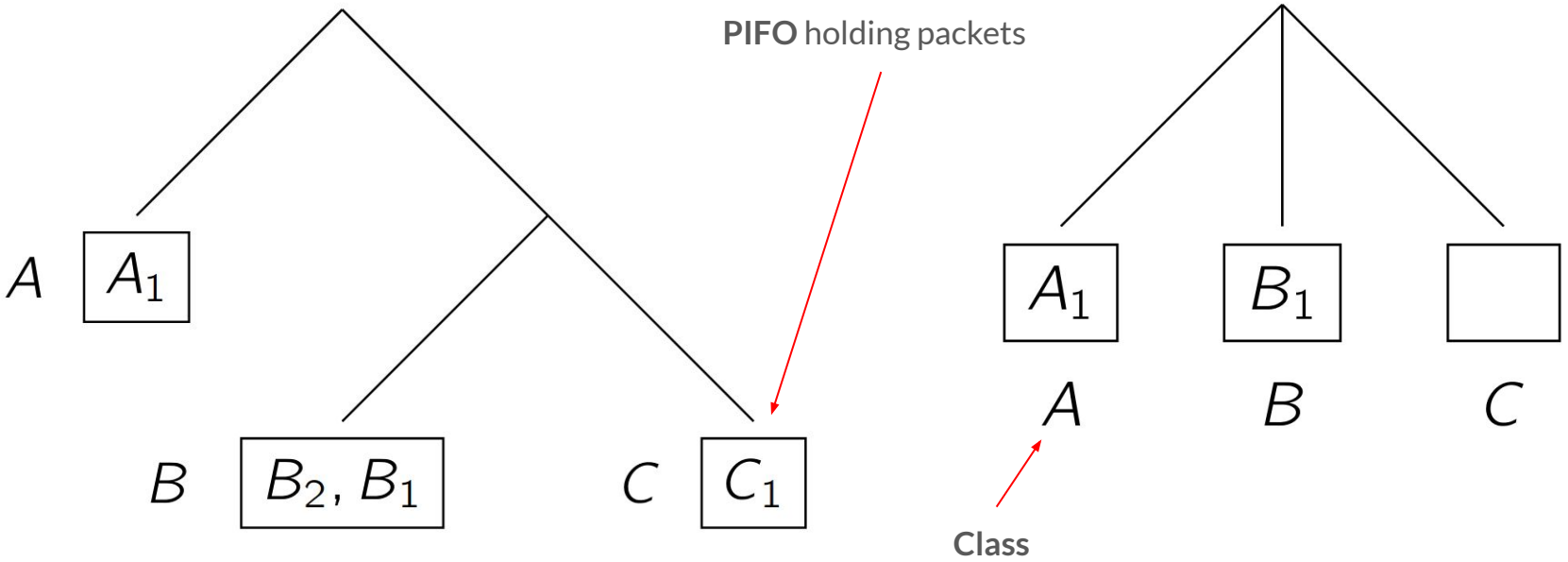
set2stream transformer:
- gives order to an unordered set (turning it into a stream)
- takes *one* set as input

stream2stream transformer:
- takes multiple *streams* as inputs

Must return stream at the end!

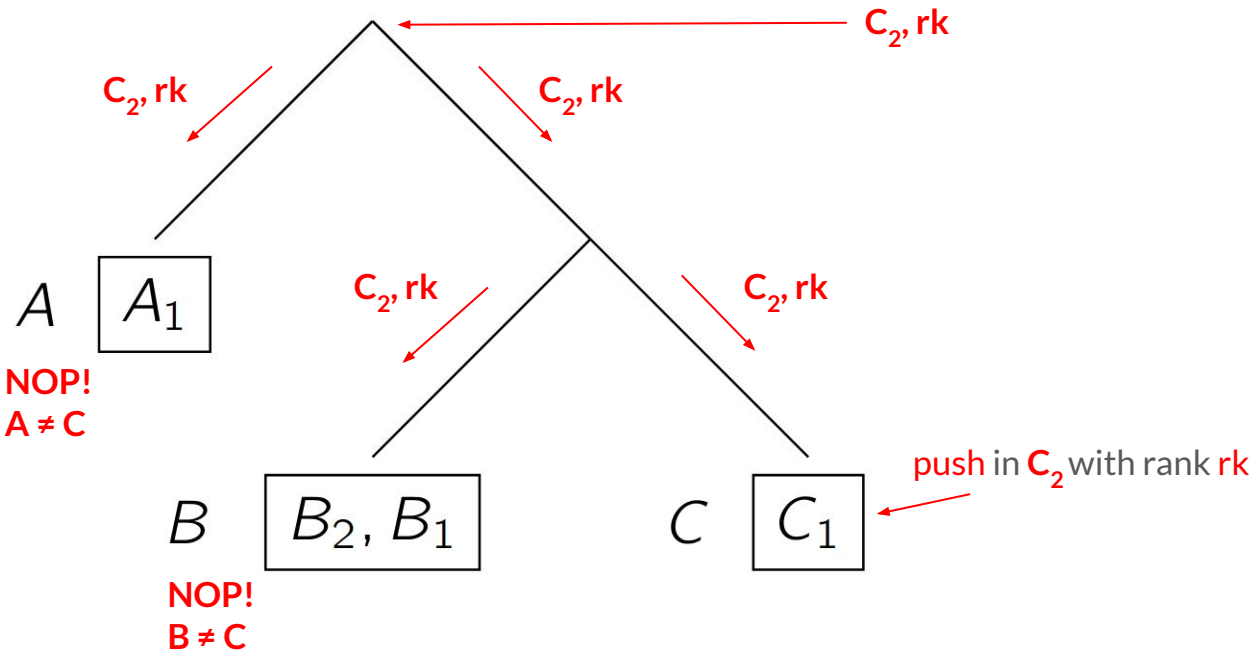
Rio Semantics

Rio Trees

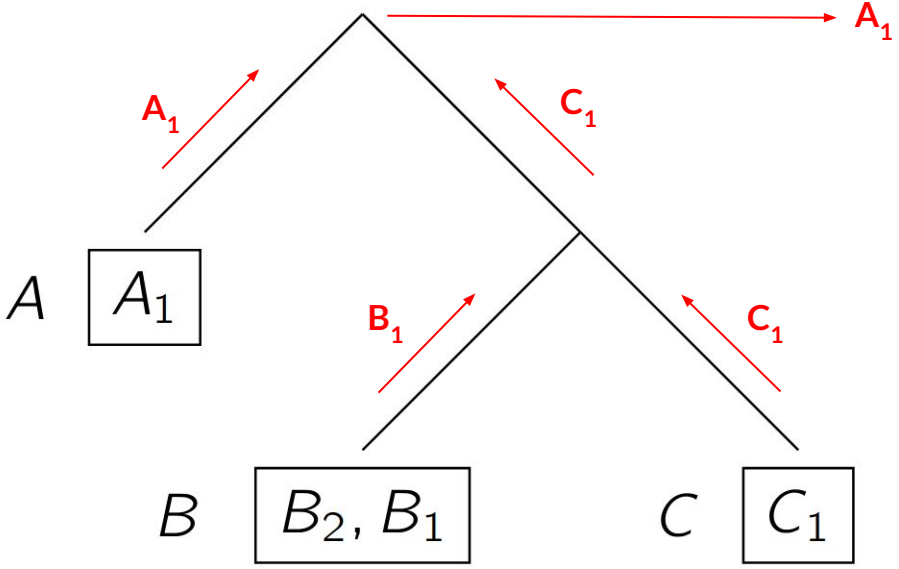


$$\text{push} : \textbf{RioTree} \times \textbf{Pkt} \times \textbf{Rk} \rightharpoonup \textbf{RioTree}$$

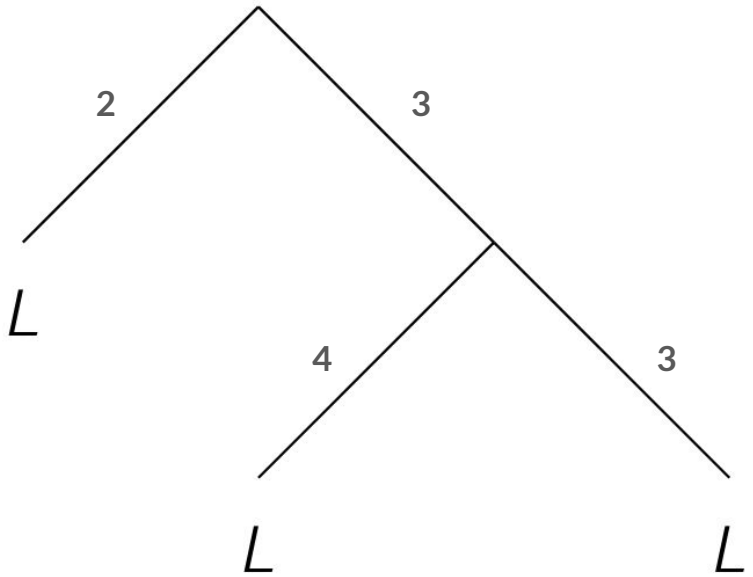
How to Push to Rio Trees



How to Pop Rio Trees



$$\text{pop} : \textbf{RioTree} \times \textbf{OrdTree} \rightharpoonup \textbf{Pkt} \times \textbf{RioTree}$$



Rio Controls

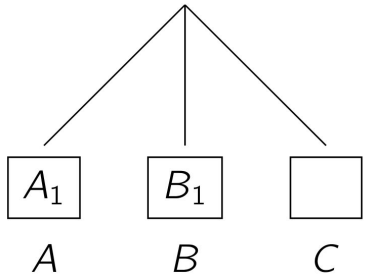
$$z_{\text{pre-push}} : \mathbf{St} \times \mathbf{Pkt} \rightharpoonup \mathbf{Rk} \times \mathbf{St}$$

$$z_{\text{pre-pop}} : \mathbf{St} \rightharpoonup \mathbf{OrdTree} \times \mathbf{St}$$

$$z_{\text{post-pop}} : \mathbf{St} \times \mathbf{Pkt} \rightharpoonup \mathbf{St}$$

Scheduling Transactions

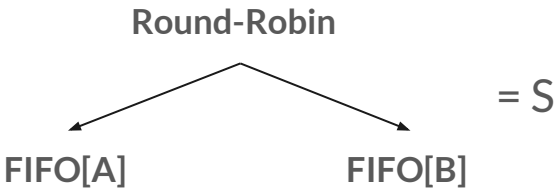
\+ State +



$$\text{push} : \mathbf{RioControl} \times \mathbf{Pkt} \rightharpoonup \mathbf{RioControl}$$

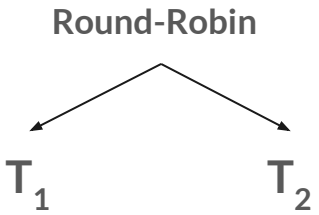
$$\text{pop} : \mathbf{RioControl} \rightharpoonup \mathbf{Pkt} \times \mathbf{RioControl}$$

Rio Programs to Controls

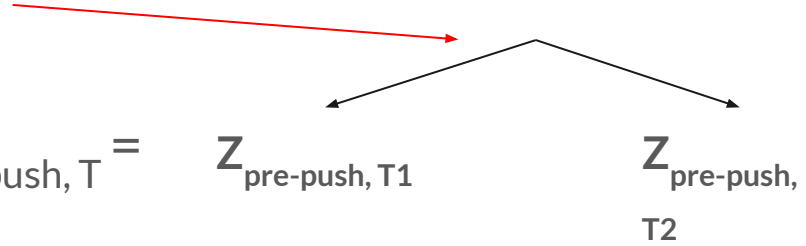


= S

T =



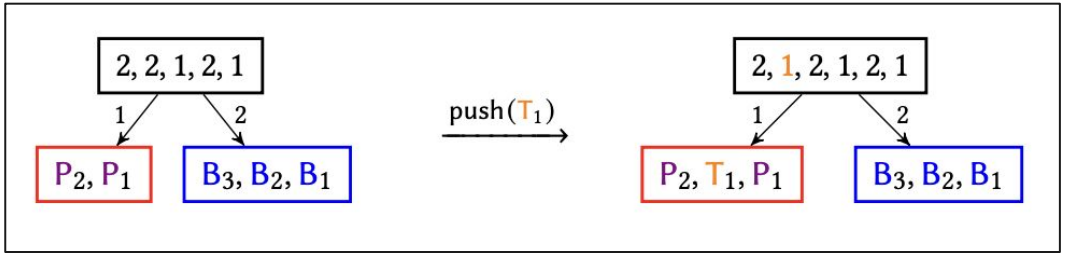
Rank children same way S would



$Z_{pre\text{-}push,\ T} = Z_{pre\text{-}push,\ T1}$ $Z_{pre\text{-}push,\ T2}$

PIFO Trees

PIFO Trees Recap



$$\frac{\textsc{pop}(p) = (pkt, p')}{\text{pop}(\text{Leaf}(p)) = (pkt, \text{Leaf}(p'))} \qquad \frac{\textsc{pop}(p) = (i, p') \qquad \text{pop}(qs[i]) = (pkt, q')}{\text{pop}(\text{Internal}(qs, p)) = (pkt, \text{Internal}(qs[q'/i], p'))}$$

$$\frac{\textsc{push}(p, pkt, r) = p'}{\text{push}(\text{Leaf}(p), pkt, r) = \text{Leaf}(p')} \qquad \frac{\text{push}(qs[i], pkt, pt) = q' \qquad \textsc{push}(p, i, r) = p'}{\text{push}(\text{Internal}(qs, p), pkt, (i, r) :: pt) = \text{Internal}(qs[q'/i], p')}$$

PIFO Tree Recap

Implementing Policies On Hardware

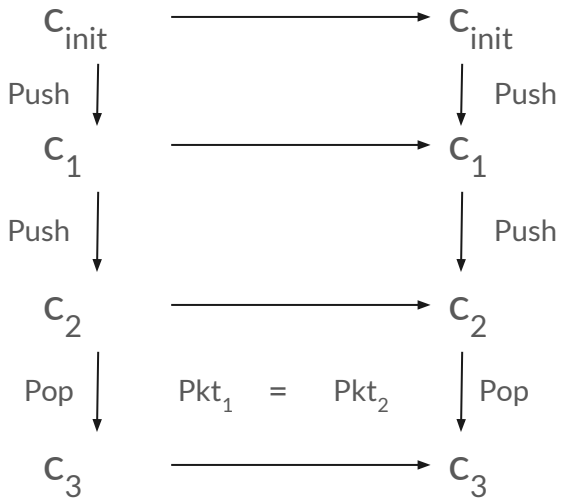
- Currently, we have two strategies for implementing PIFO Trees on Hardware.
  - **Queues**, which work well for policies like Round Robin, but fail on others.
  - **Binary Heaps**

- We've made significant progress towards representing Round Robin policies on hardware!
- Further work involves using our Binary Heaps to implement policies like Weighted-Fair Queueing.

Enqueue-Dequeue Side Equivalence

Math

PIFO Control **Rio Control**




```
def z_pre-pop(st):
    turn = int(s["turn"])
    rs = []
    for i in range(n):
        rs.append((i - turn) % n)
    return Internal(rs, [Leaf] * n)
```

```
def z_pre-push(st, pkt):
    r = arrival_time(pkt)
    f = flow(pkt)
    rank_ptr = "r_" + str(f)
    r_i = int(st[rank_ptr])
    st["rank_ptr"] += float(n)
    return ((f, r_i) :: r, st)
```

```
def z_post-pop(st, pkt):
    f = flow(pkt)
    turn = int(s["turn"])
    i = turn
    while i != f:
        st["r_" + str(i)] += n
        i = (i + 1) % n
    st["turn"] = float((f + 1) % n)
    if turn >= st["turn"]:
        st["cycle"] += 1
    return st
```

Simulator

DEMO time!