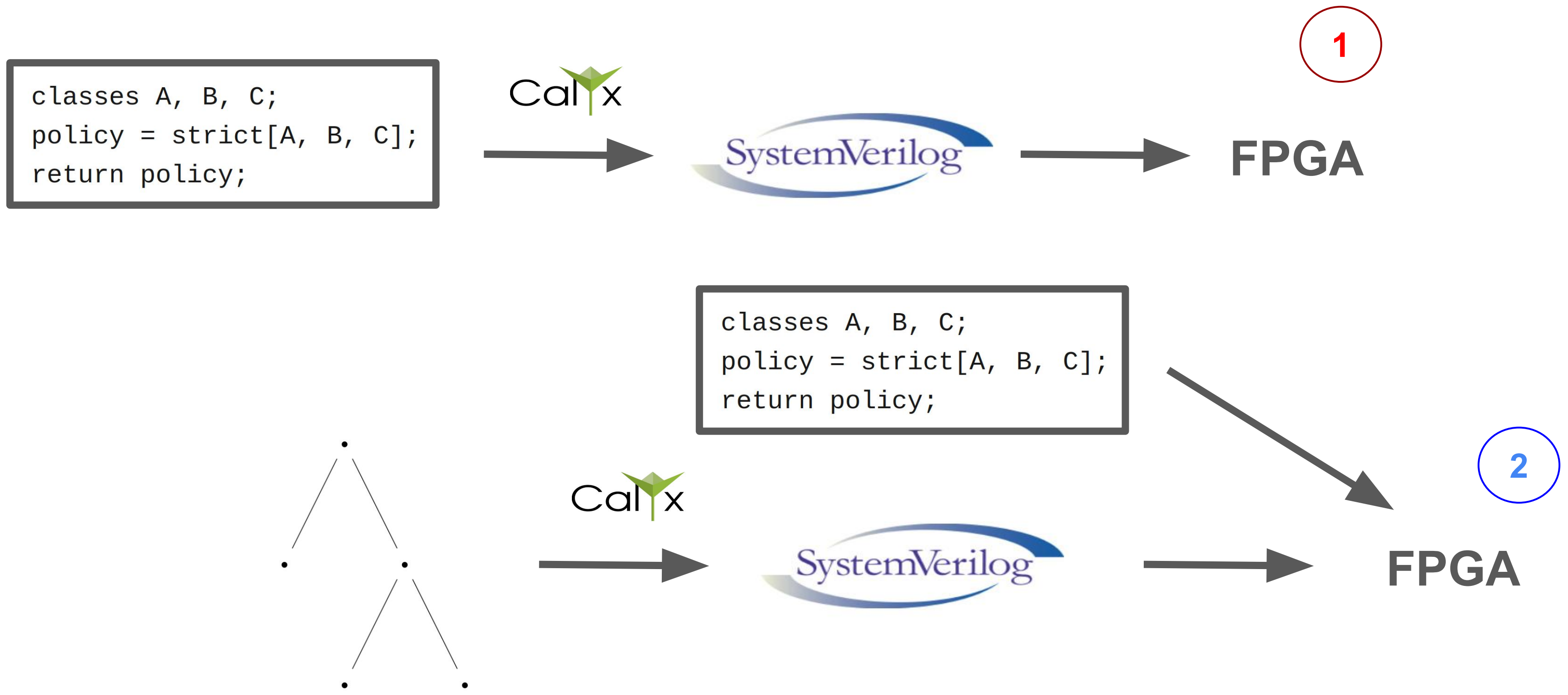
Rio: DSL for programmable packet scheduling

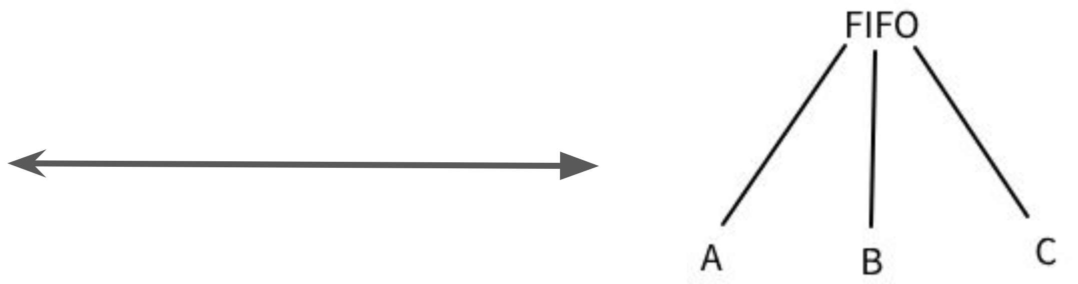


```
classes A, B, C;
policy = strict[A, B, C];
return policy;
```

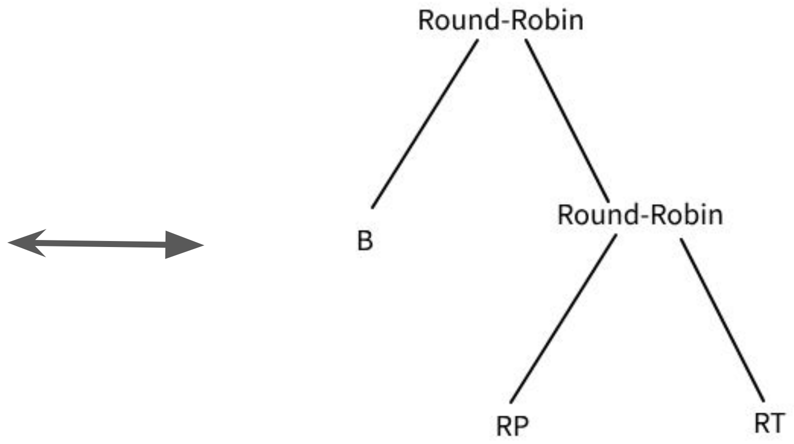
```
classes A, B, C;
policy = strict[A, B, C];
return policy;
```

Rio Snippets

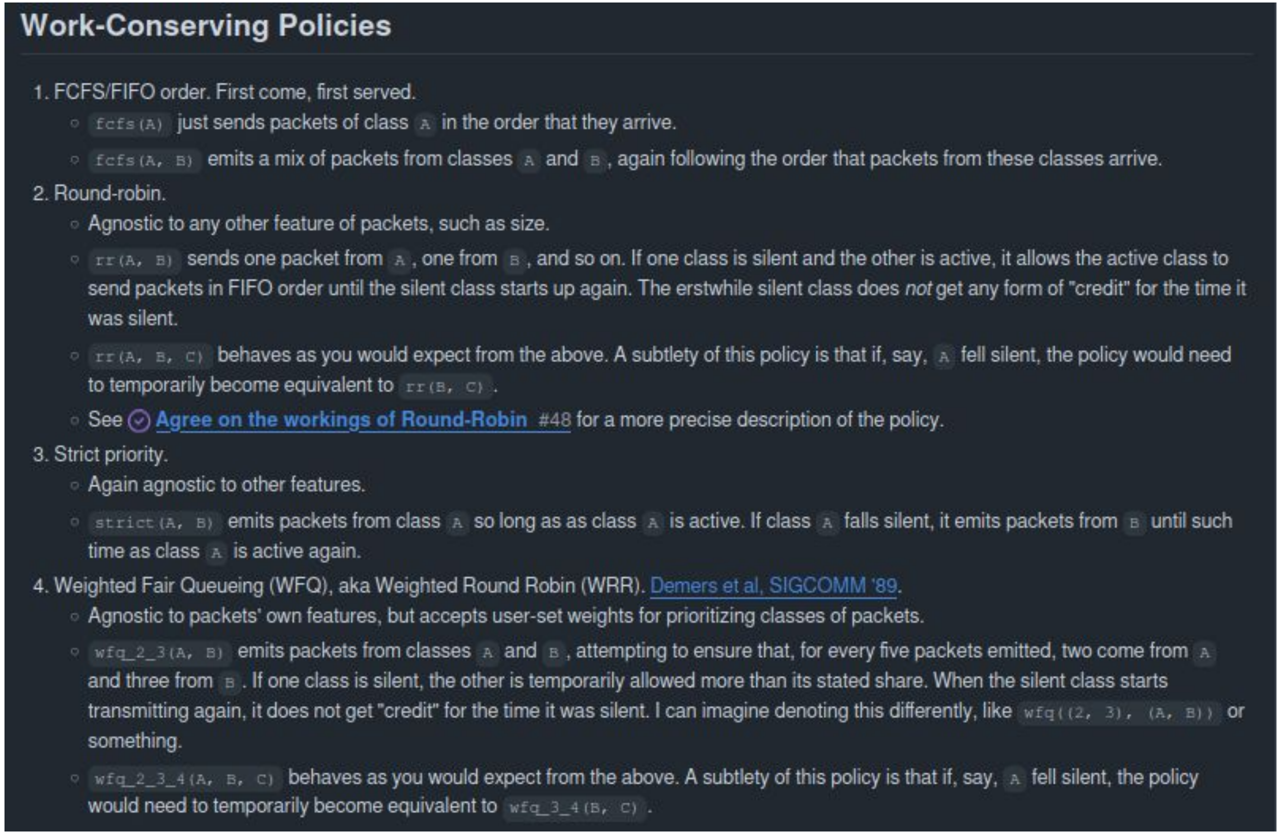
```
classes A, B, C;
policy = fifo[A, B, C];
return policy
```



```
classes B, RP, RT;
// B represents packets that came from Buffalo
// R? represents packets that came from Rochester
// RP packets are destined for Pittsburgh
// RT packets are destined for Toronto
r_policy = rr[RP, RT];
policy = rr[B, r_policy];
return policy
```



Semantics – non-hierarchical setting



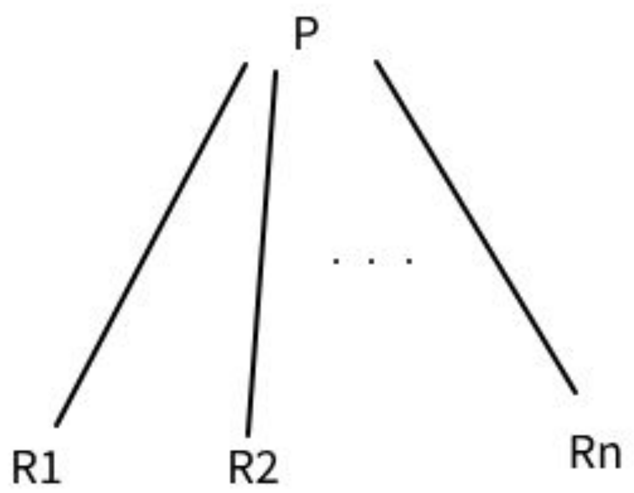
Work-Conserving Policies

1. FCFS/FIFO order. First come, first served.
   - `fcfs(A)` just sends packets of class `A` in the order that they arrive.
   - `fcfs(A, B)` emits a mix of packets from classes `A` and `B`, again following the order that packets from these classes arrive.
2. Round-robin.
   - Agnostic to any other feature of packets, such as size.
   - `rr(A, B)` sends one packet from `A`, one from `B`, and so on. If one class is silent and the other is active, it allows the active class to send packets in FIFO order until the silent class starts up again. The erstwhile silent class does *not* get any form of "credit" for the time it was silent.
   - `rr(A, B, C)` behaves as you would expect from the above. A subtlety of this policy is that if, say, `A` fell silent, the policy would need to temporarily become equivalent to `rr(B, C)`.
   - See Agree on the workings of Round-Robin #48 for a more precise description of the policy.
3. Strict priority.
   - Again agnostic to other features.
   - `strict(A, B)` emits packets from class `A` so long as as class `A` is active. If class `A` falls silent, it emits packets from `B` until such time as class `A` is active again.
4. Weighted Fair Queueing (WFQ), aka Weighted Round Robin (WRR). Demers et al, SIGCOMM '89.
   - Agnostic to packets' own features, but accepts user-set weights for prioritizing classes of packets.
   - `wfq_2_3(A, B)` emits packets from classes `A` and `B`, attempting to ensure that, for every five packets emitted, two come from `A` and three from `B`. If one class is silent, the other is temporarily allowed more than its stated share. When the silent class starts transmitting again, it does not get "credit" for the time it was silent. I can imagine denoting this differently, like `wfq((2, 3), (A, B))` or something.
   - `wfq_2_3_4(A, B, C)` behaves as you would expect from the above. A subtlety of this policy is that if, say, `A` fell silent, the policy would need to temporarily become equivalent to `wfq_3_4(B, C)`.

glossar

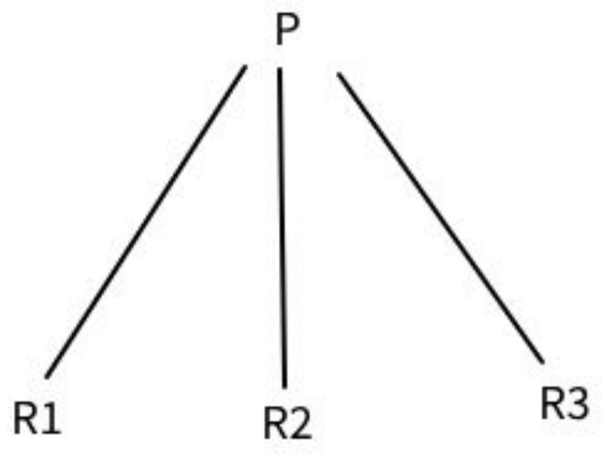
Semantics – hierarchical setting

Given a policy P and DSL programs R1, R2, … , Rn, what is this?



Current Strategy

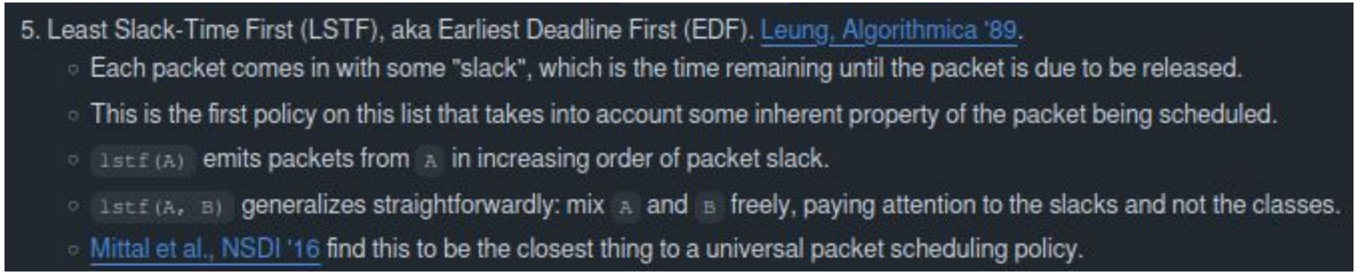
Let's consider n = 3 for instance.



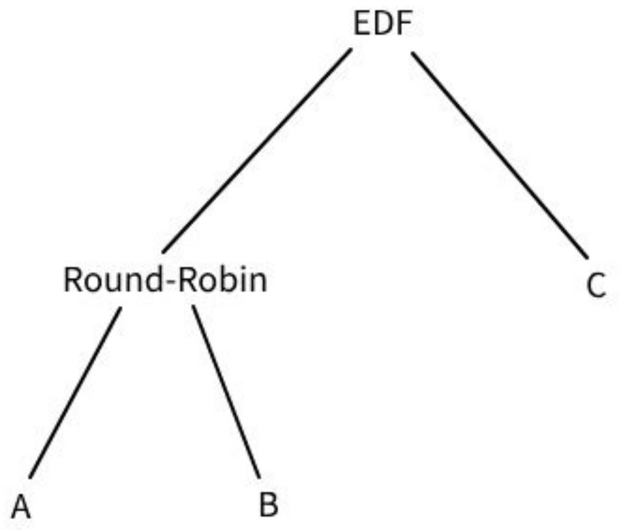
Suppose we encounter a *pkt*, how do we push to our PIFO tree?

1. Flow inference
   a. This determines which subtree *pkt*'s class is in. Let's say it's the *i*th subtree
2. Push i to the root with rank *r* as though each *R1*, *R2*, *R3* are all classes
3. Recursively push to *pkt* to *Ri*

This is a bit broken



5. Least Slack-Time First (LSTF), aka Earliest Deadline First (EDF). Leung, Algorithmica '89.
    - Each packet comes in with some "slack", which is the time remaining until the packet is due to be released.
    - This is the first policy on this list that takes into account some inherent property of the packet being scheduled.
    - `lstf(A)` emits packets from `A` in increasing order of packet slack.
    - `lstf(A, B)` generalizes straightforwardly: mix `A` and `B` freely, paying attention to the slacks and not the classes.
    - Mittal et al., NSDI '16 find this to be the closest thing to a universal packet scheduling policy.



Suppose we encounter the following sequence of packets before any pops:

# Notation: (X_i, j) is the ith packet from flow X, with slack j
(A_1, 1), (A_2, 2), (A_3, 3), (B_1, 1000), (C_1, 4)

Flushing then yields

A_1, B_1, A_2, C_1, A_3