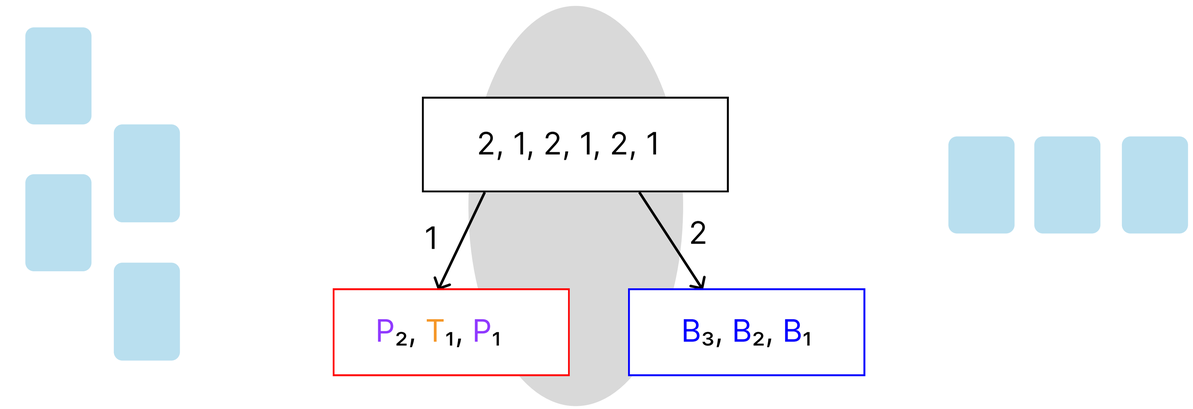
A DSL for Programmable Packet Scheduling

Cassandra Sziklai

Background

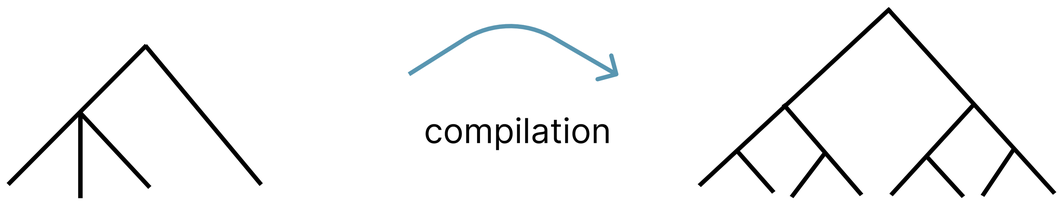
High level languages → Calyx → Hardware description languages



incoming packets

scheduler

outgoing packets



the user wants a specific type of tree

compilation

the hardware supports one type of tree

Work Conserving Policies in Calyx

Round Robin generalized to n flows
n = 4

1. push 52		Q1 = [52] ≤ 100
2. peek	→ 52	Q2 = [127, 110, 143] ≤ 200
3. push 127		Q3 = [208, 281] ≤ 300
4. peek	→ 52	Q4 = [374] ≤ 400
5. push 374		
6. push 110		answer = [52, 52, 52,
7. push 208		127, 127, 208, 374, 81]
8. push 143		
9. pop	→ 52	
10. peek	→ 127	
11. pop	→ 127	
12. push 281		
13. pop	→ 208	
14. pop	→ 374	
15. pop	→ 110	

Strict Queues

- maintain a strict priority among classes
- Eg. Q3 > Q2 > Q1

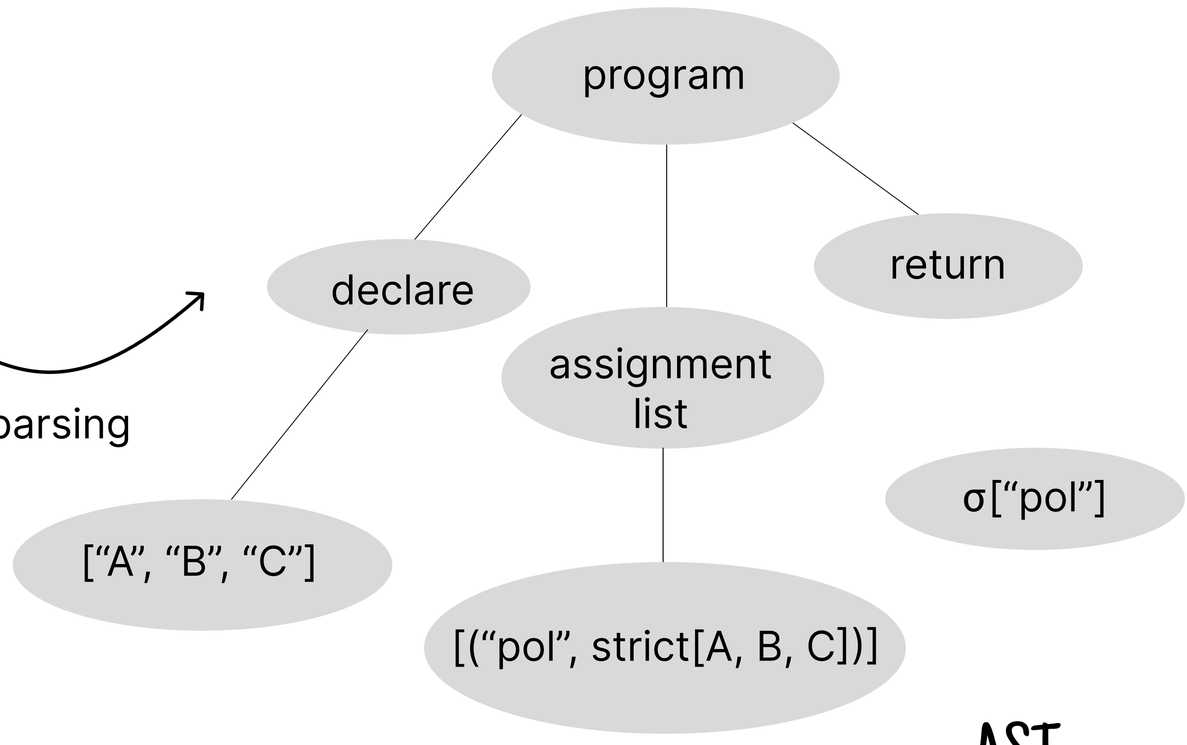
Creating the DSL

```
pol = strict[A, B, C];
```

lexing →

VAR(“pol”) EQUALS
STRICT LBRACKET
CLSS(“A”) COMMA
CLSS(“B”) COMMA
CLSS(“C”) RBRACKET
SEMICOLON

parsing →



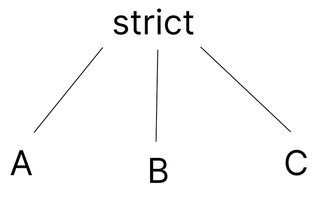
program

```
classes A, B, C;

pol = strict[A, B, C];

return pol
```

tokens

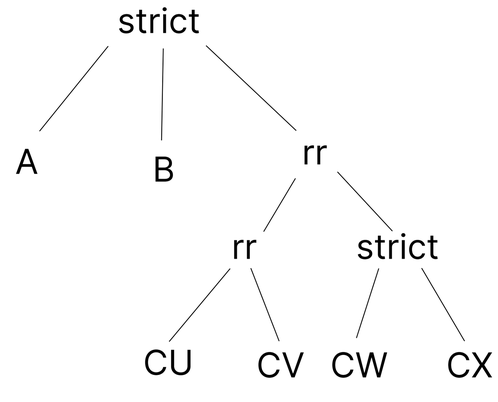


AST

```
classes A, B, CU, CV, CW, CX;

c_policy = rr[rr[CU, CV], strict[CW, CX]];
policy = strict[A, B, c_policy];

return policy
```



Checking well-formedness

- substitute variables into policies
- only declared classes can be used
- prevent unbound variables
- check for duplicate classes

Interested in more? Contact cns58@cornell.edu

view the code