Formal Abstractions for Packet Scheduling

Mohan, Liu, Foster, Kappé, Kozen

SDN made networks programmable.

SDN made networks programmable.

Early goal: routing.

SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

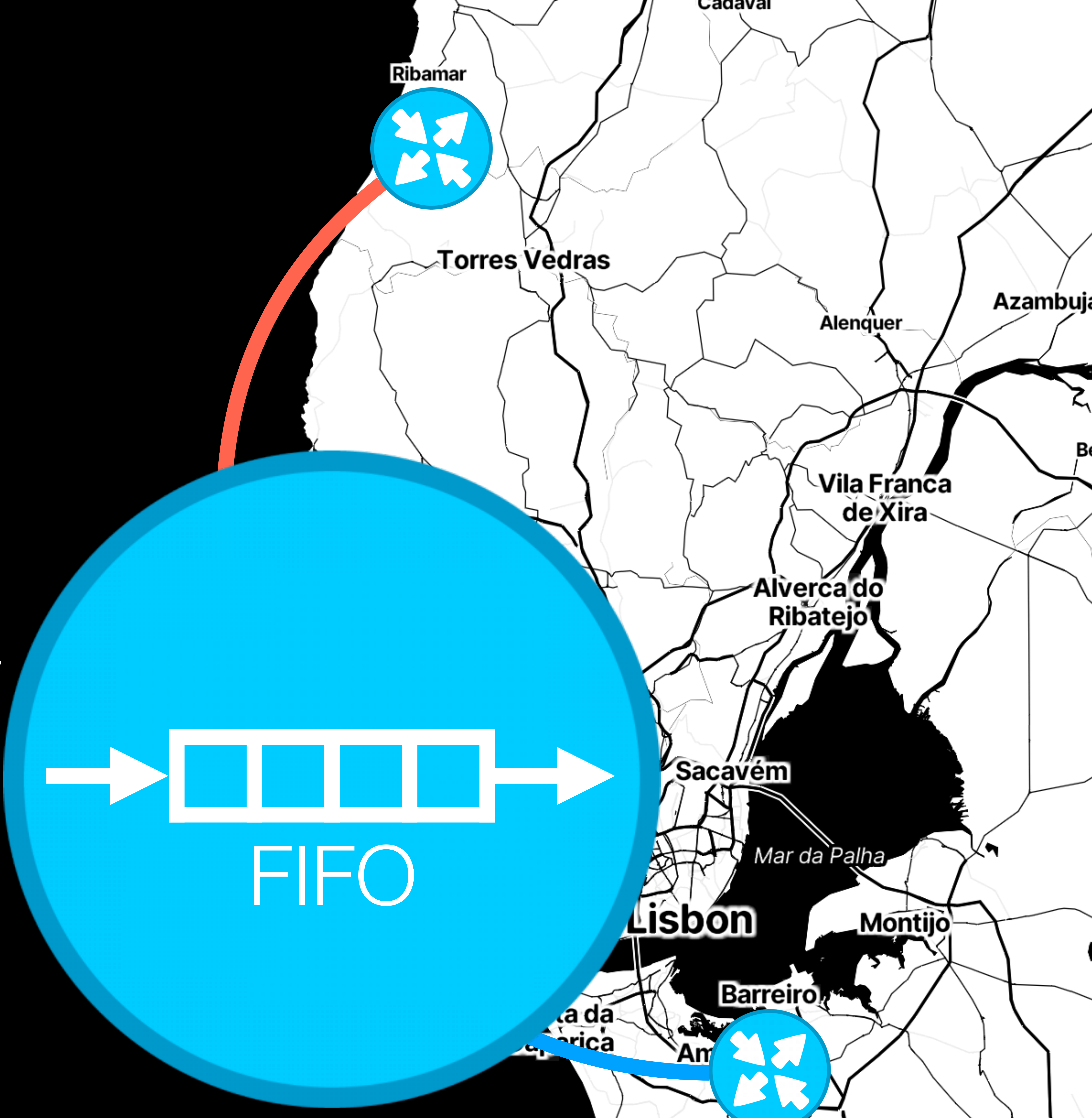
Basic tools work fine…

SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

Basic tools work fine…

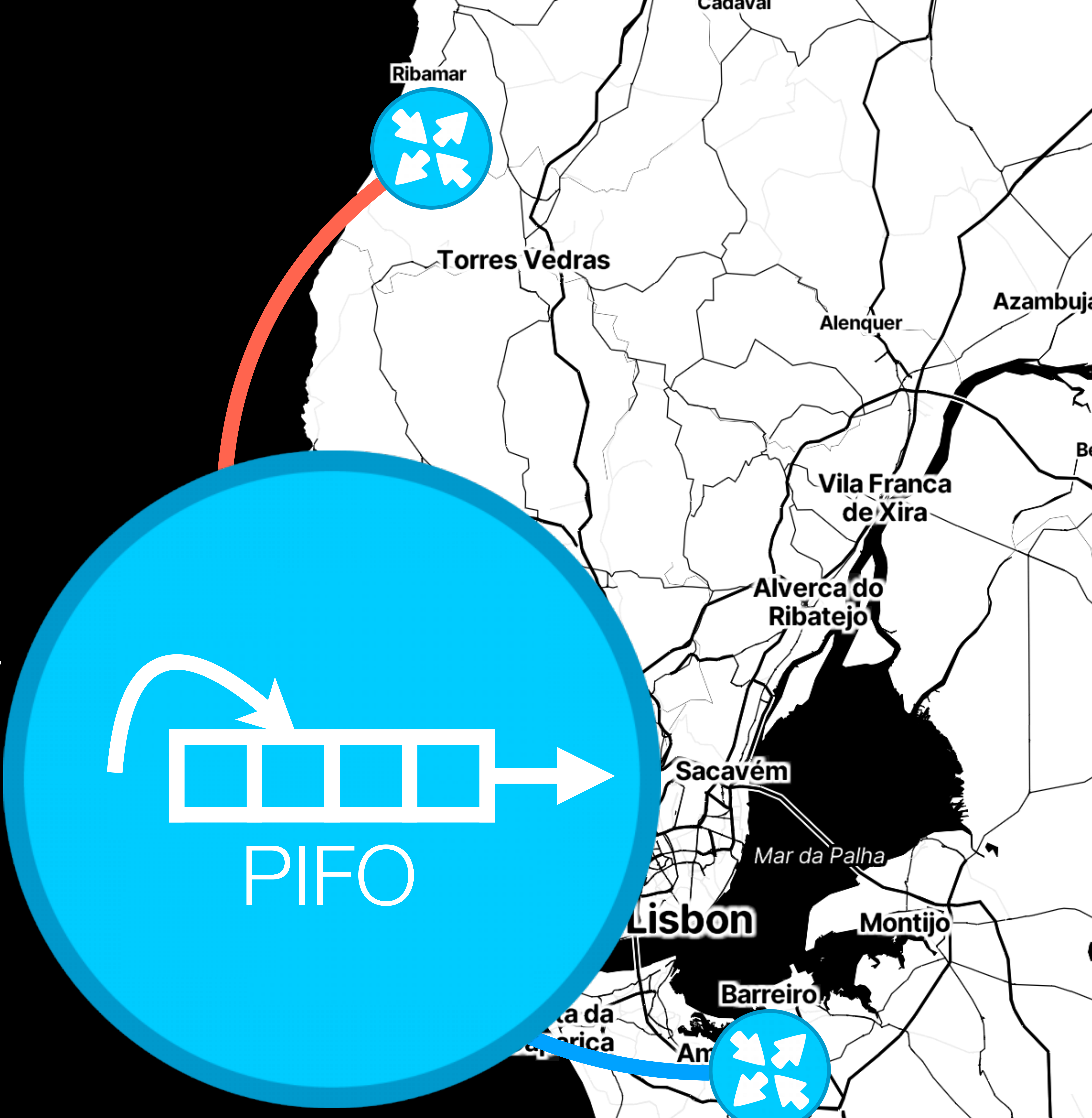


SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

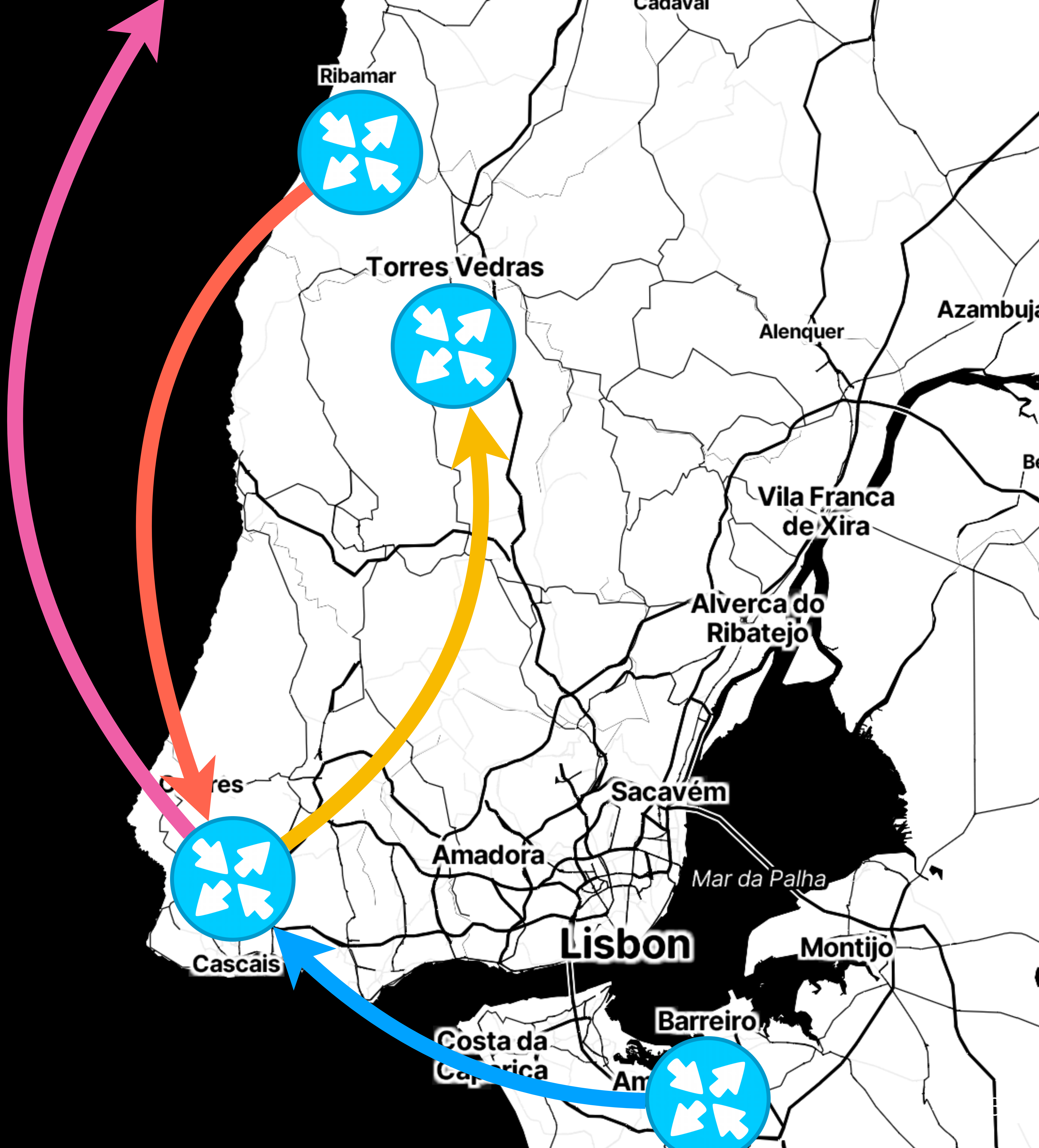
Basic tools work fine…



But modern scheduling requires more.

But modern scheduling requires more.

R traffic goes to either Porto or Torres Vedras.

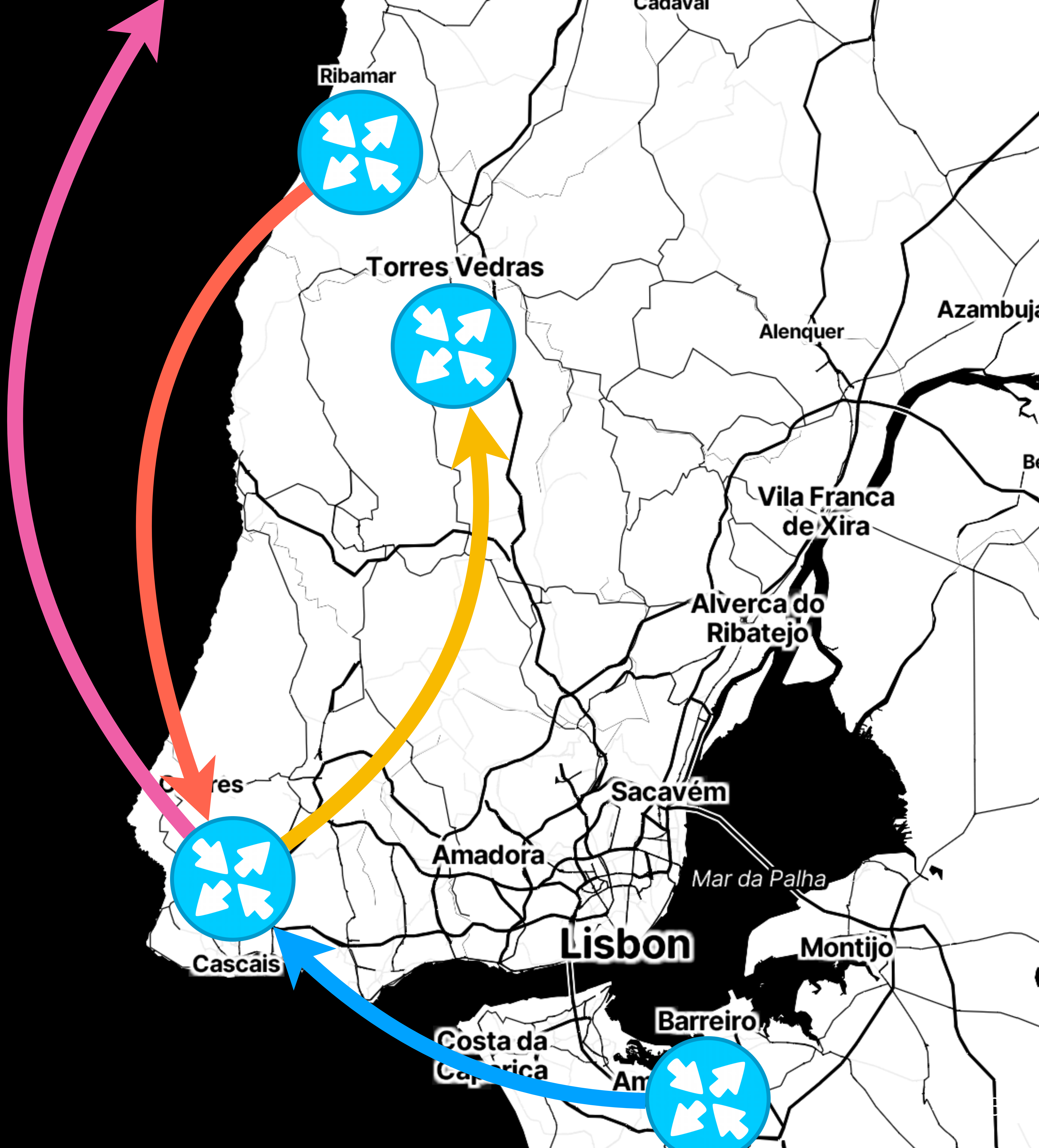


But modern scheduling requires more.

R traffic goes to either Porto or Torres Vedras.

Goal:

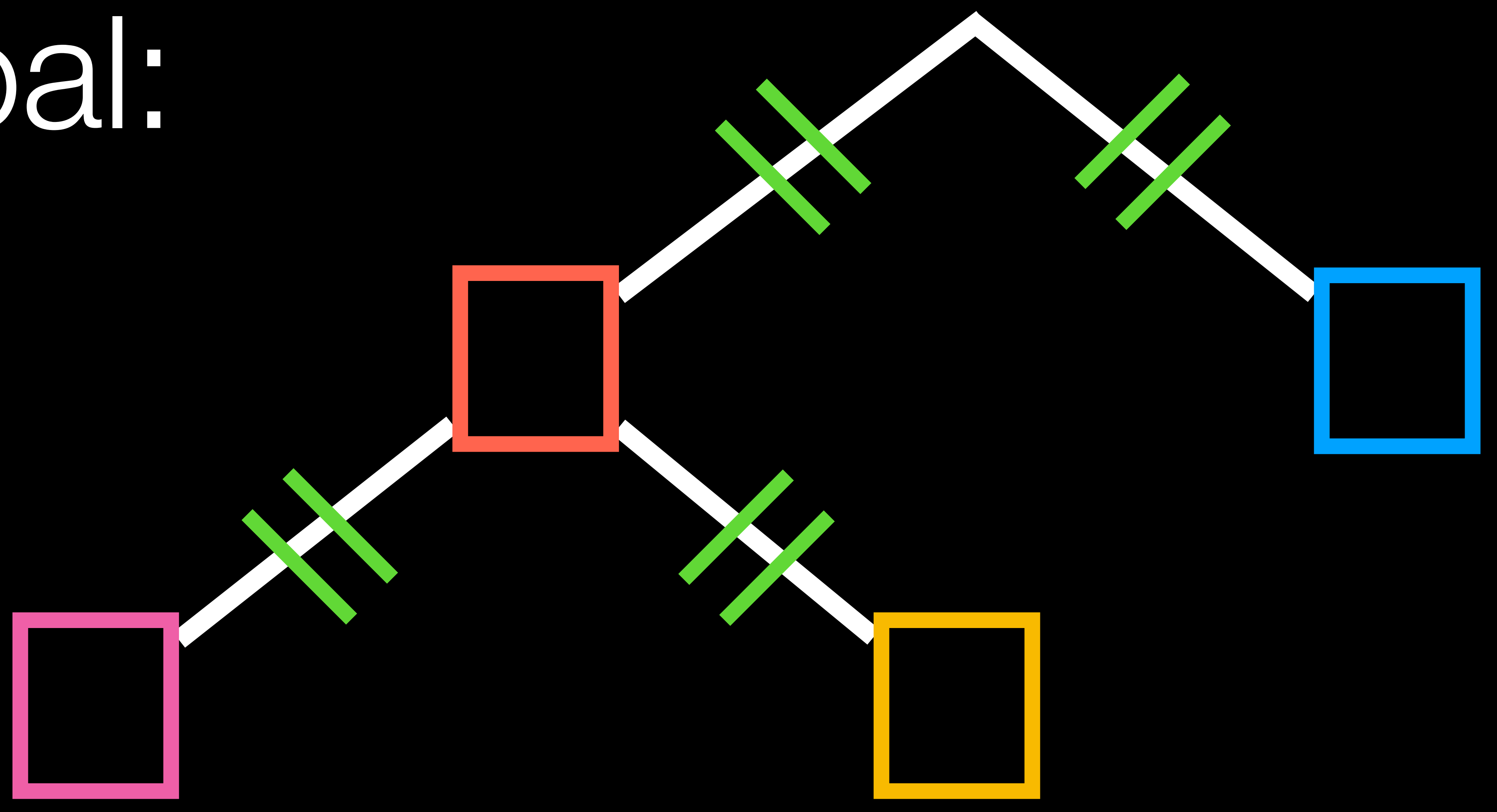
Interleave R and B; interleave P and T.

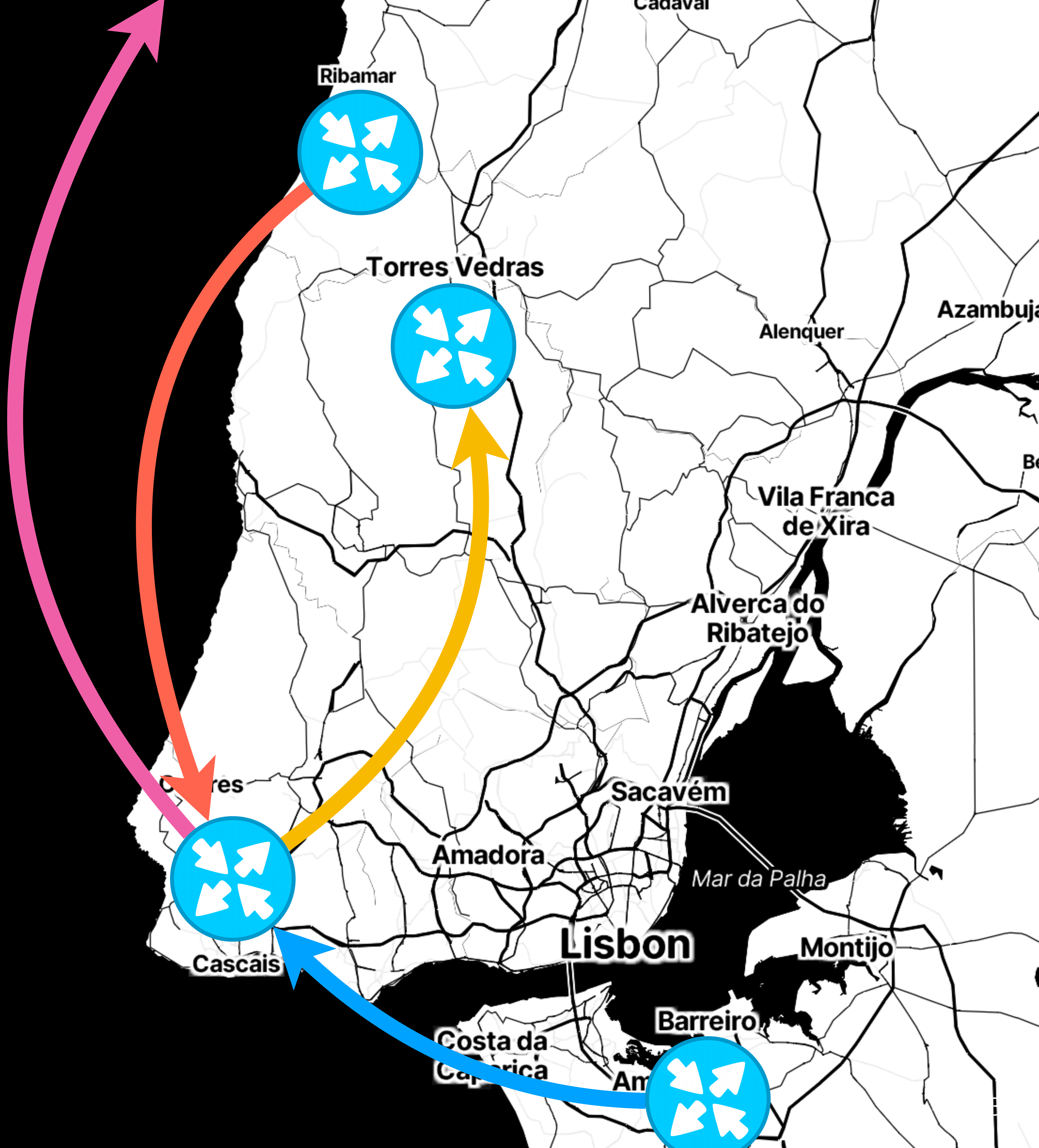


But modern scheduling requires more.

R traffic goes to either Porto or Torres Vedras.

Goal:

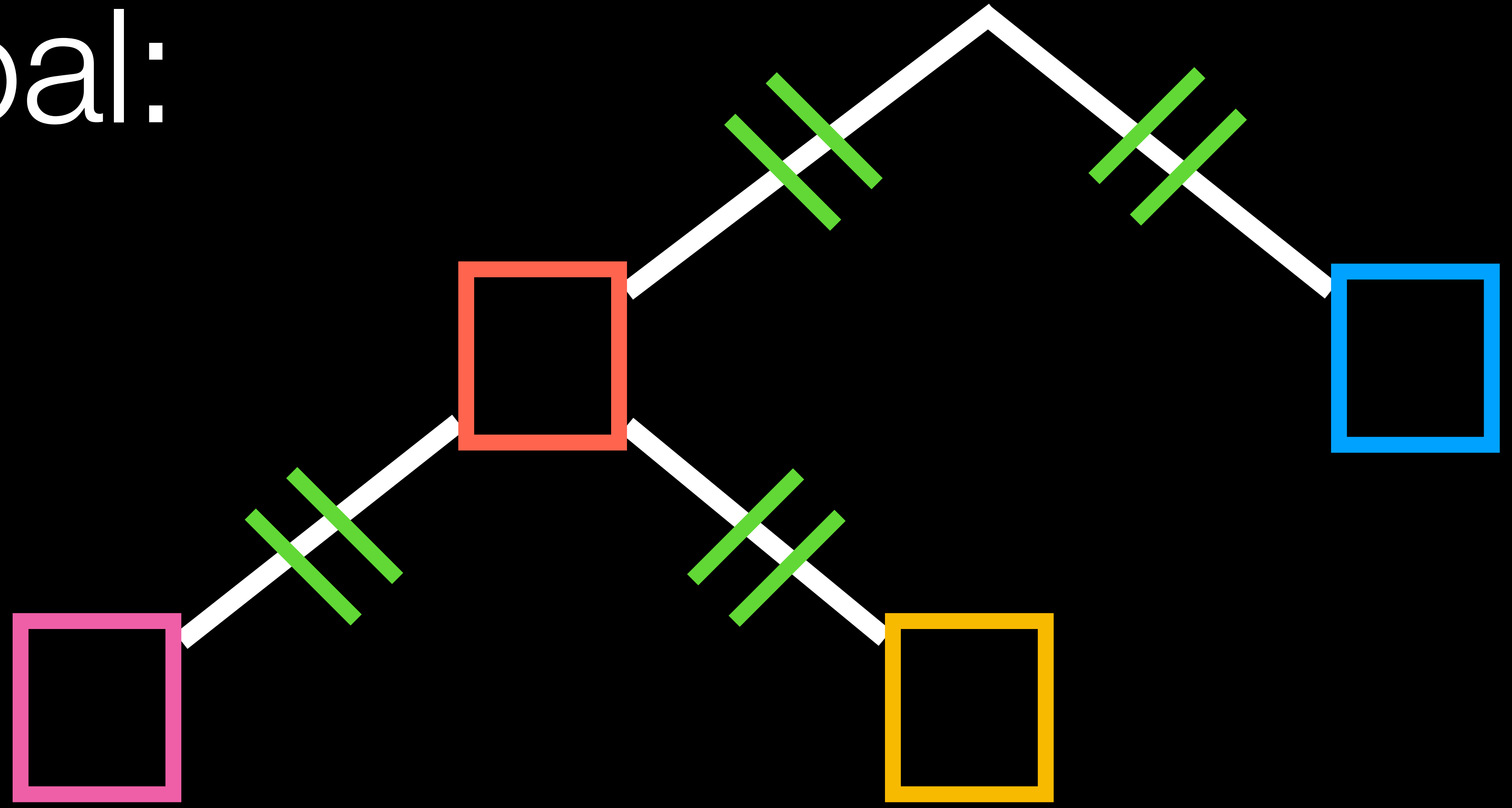


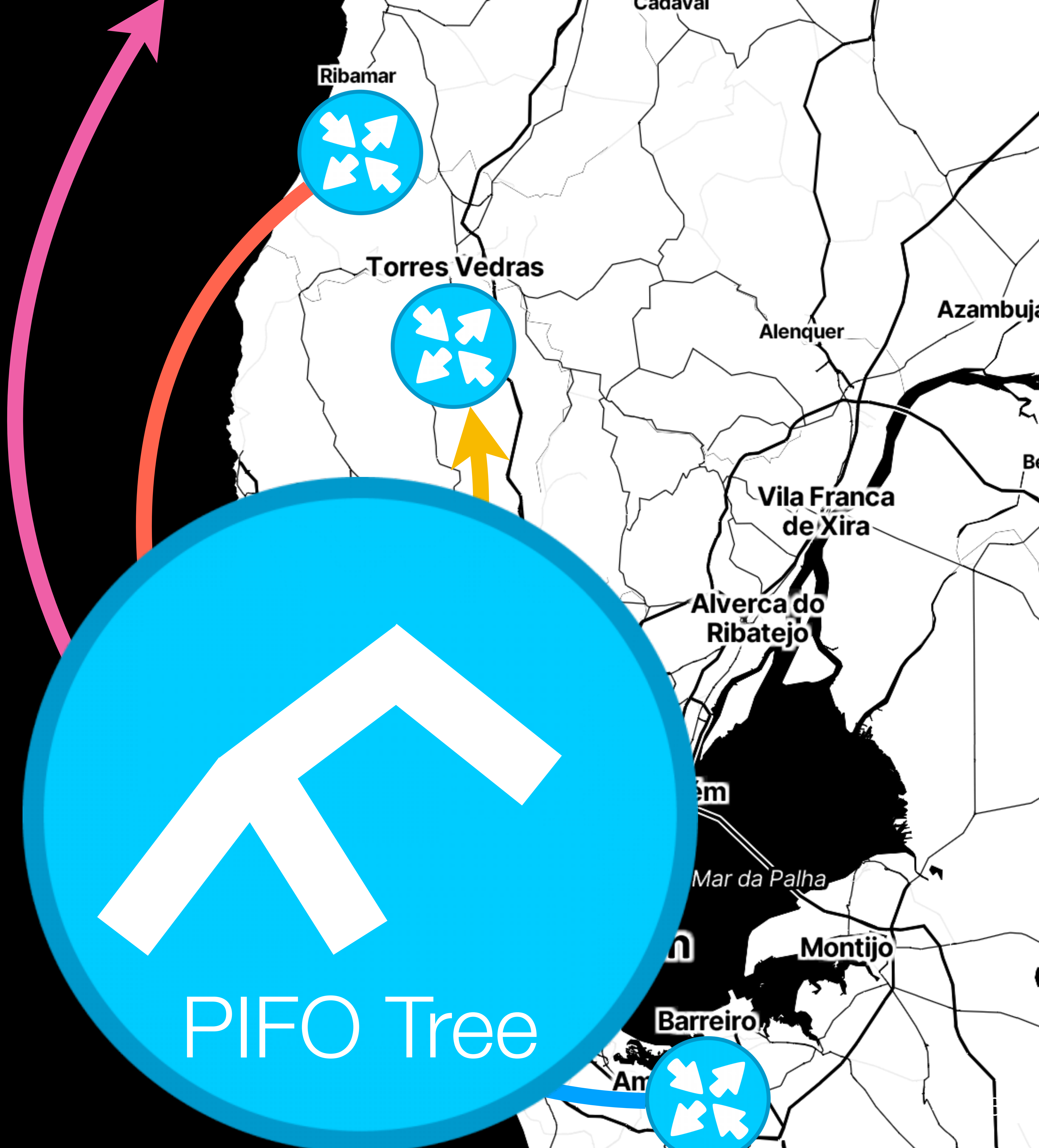


But modern scheduling requires more.

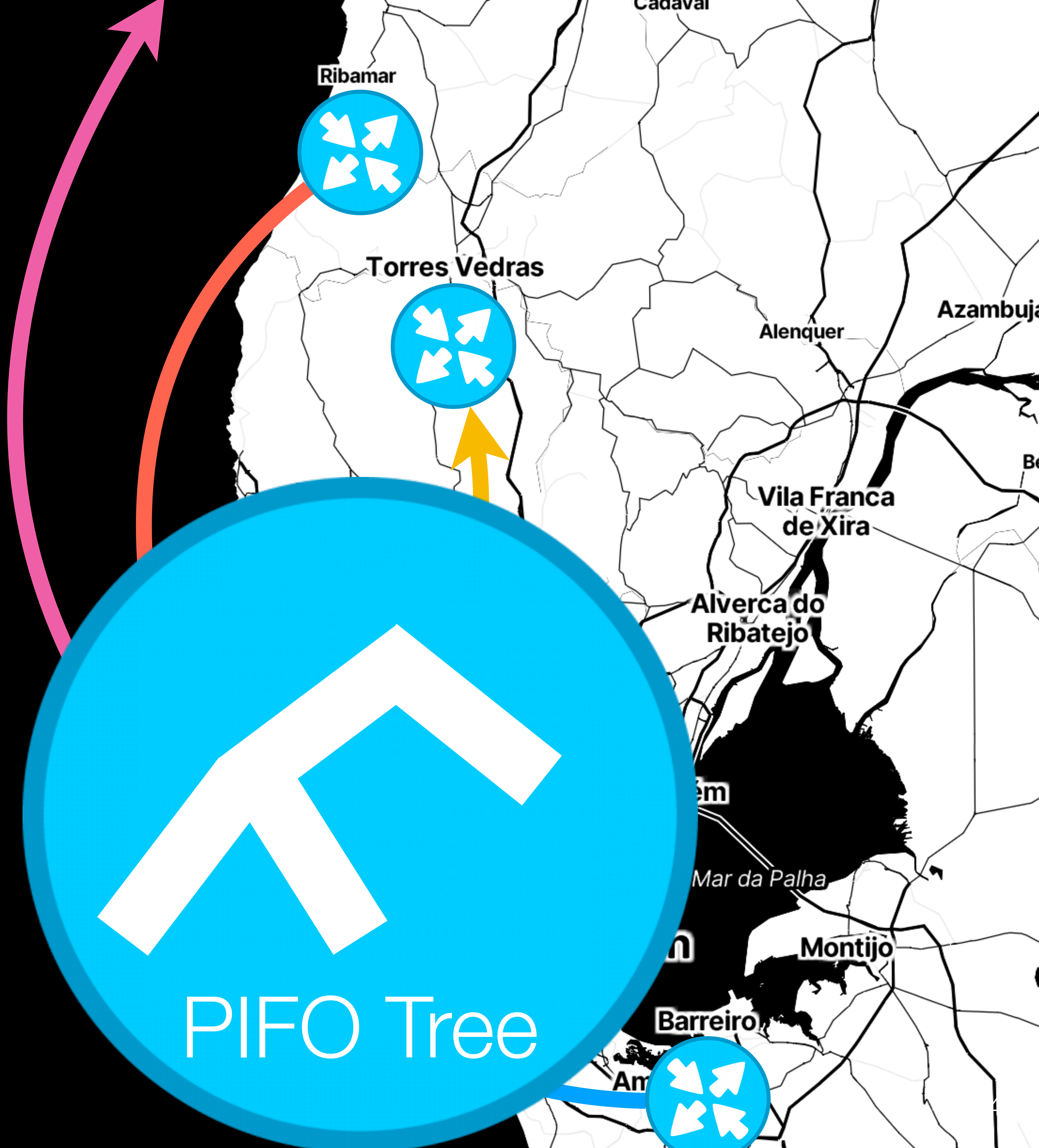
R traffic goes to either Porto or Torres Vedras.

Goal:



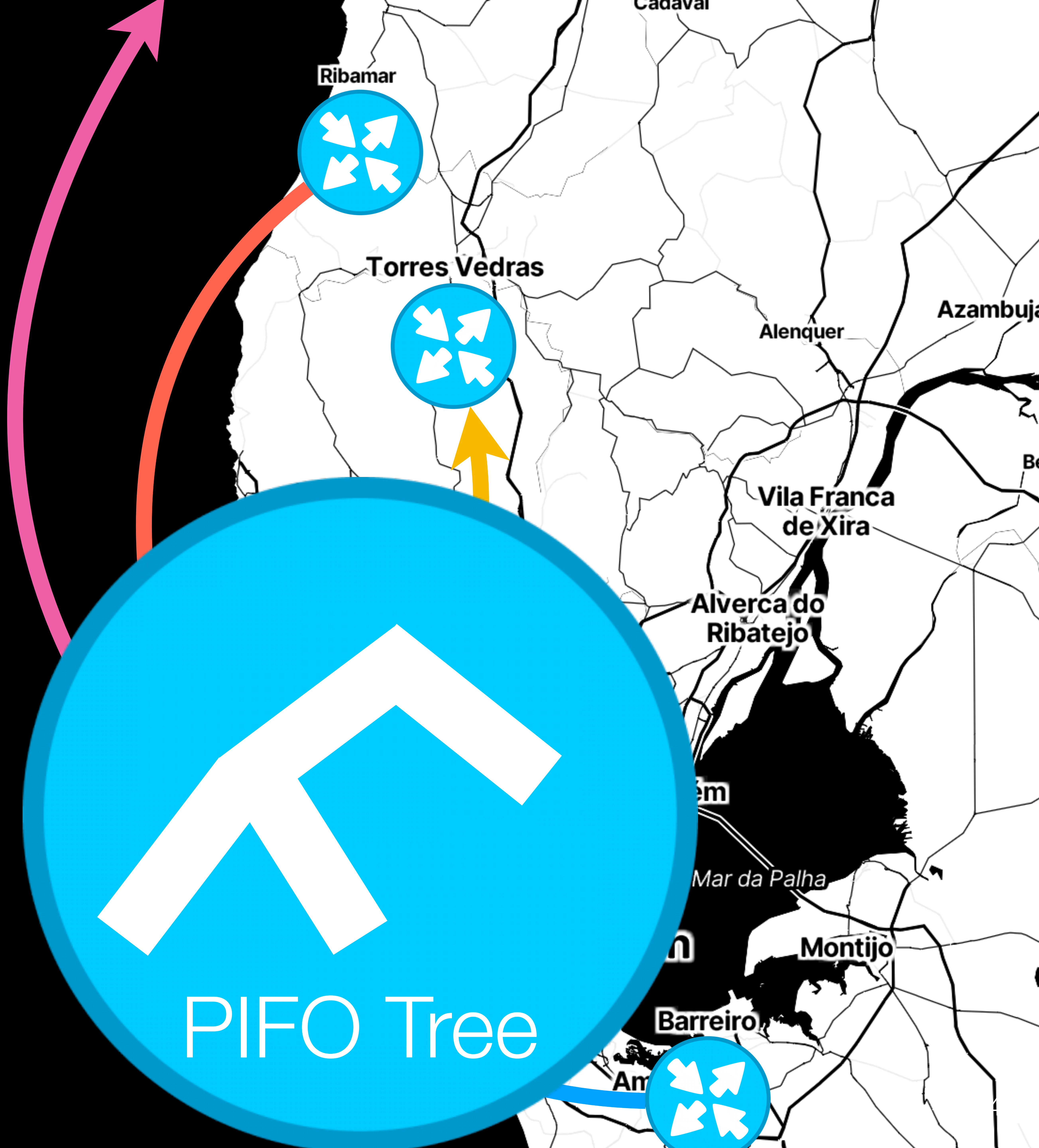


New plan!


Ribamar
Torres Vedras
Alenquer
Vila Franca de Xira
Alverca do Ribatejo
Mar da Palha
Montijo
Barreiro
PIFO Tree

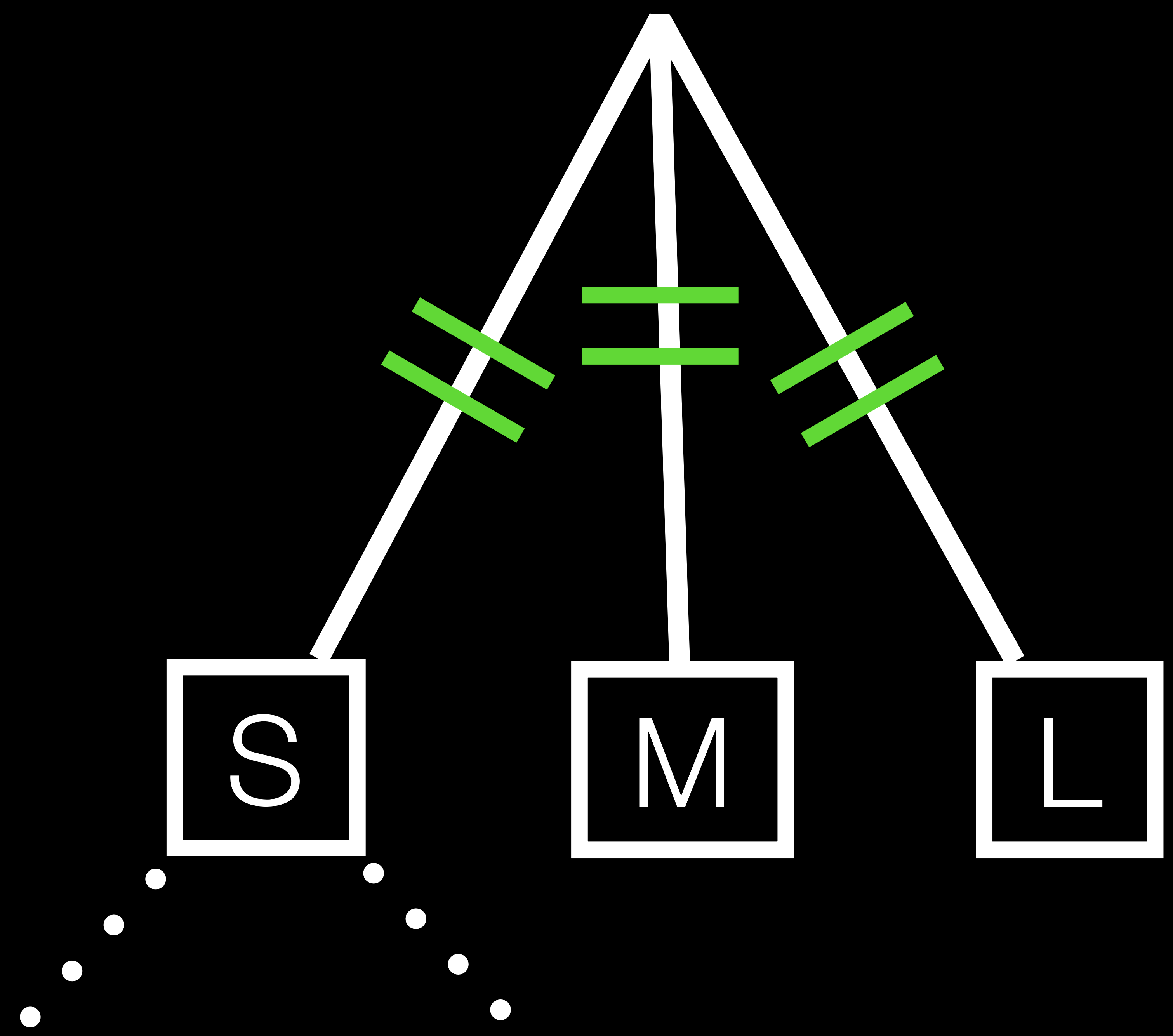
New plan!

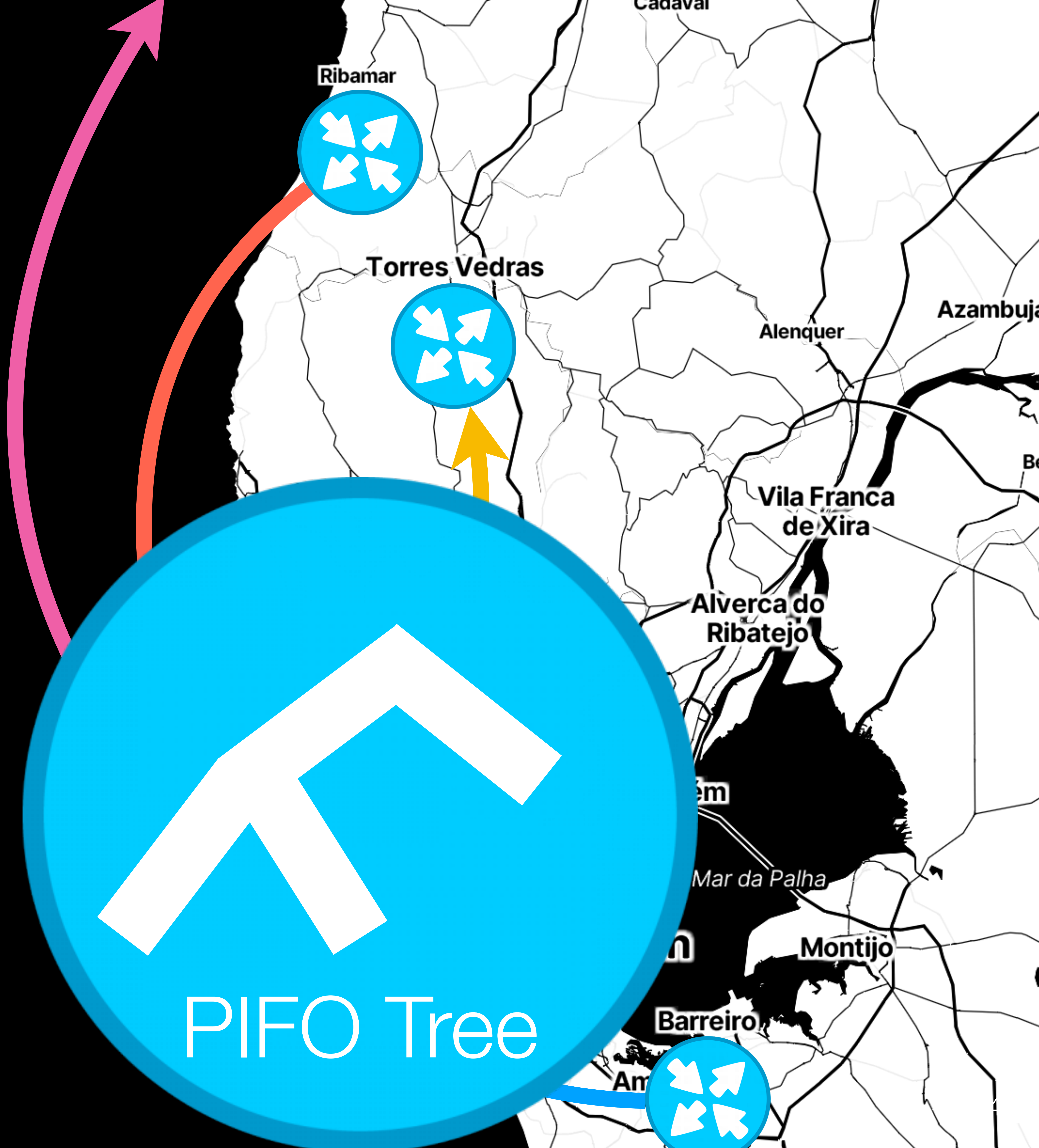
Interleave
small, medium, and large
packets.



New plan!

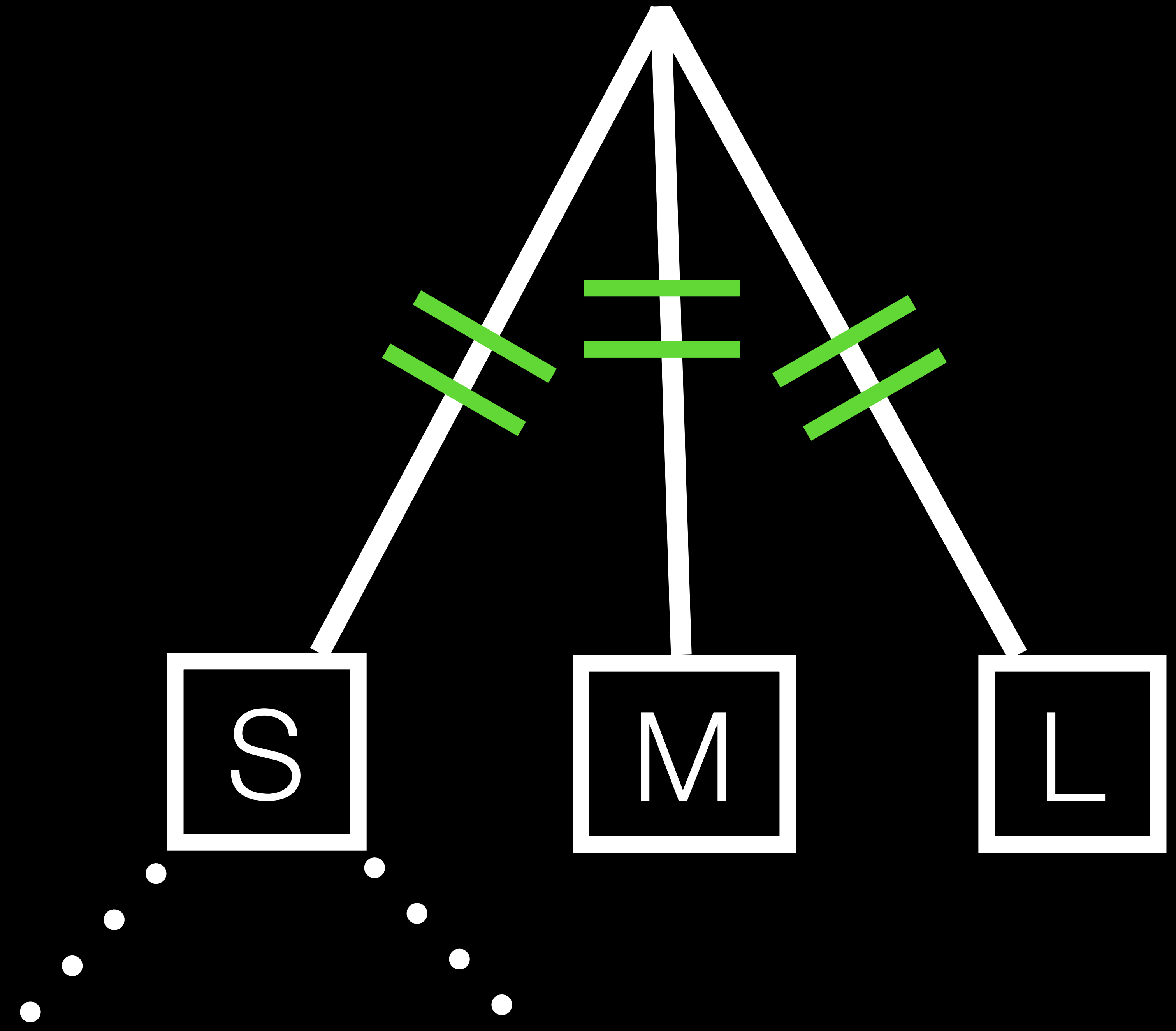
Interleave
small, medium, and large
packets.

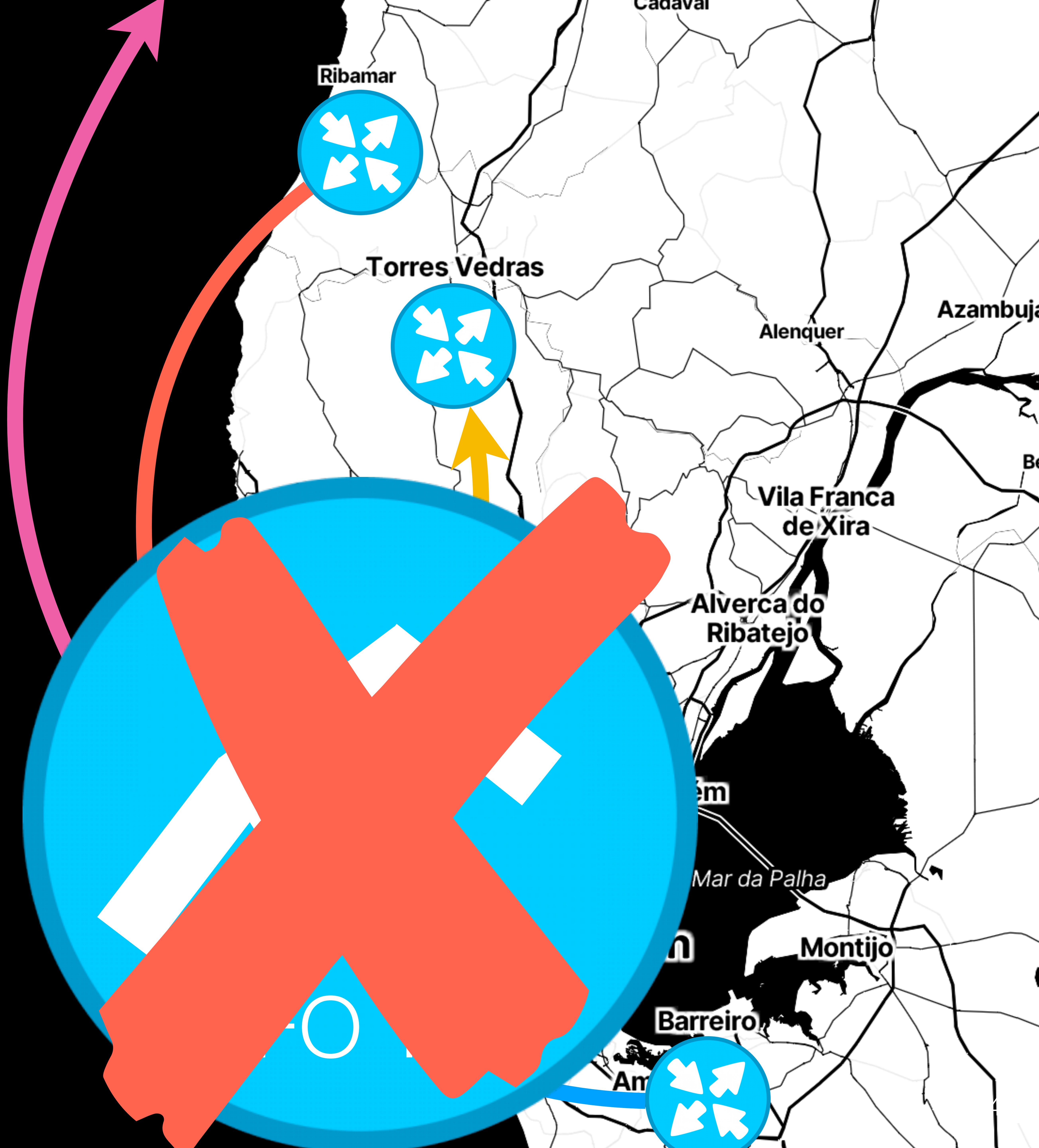




New plan!

Interleave
small, medium, and large
packets.





No general way to deploy our gadget.

No general way to deploy our gadget.

A human needs a *range* of trees.

No general way to deploy our gadget.

A human needs a *range* of trees.

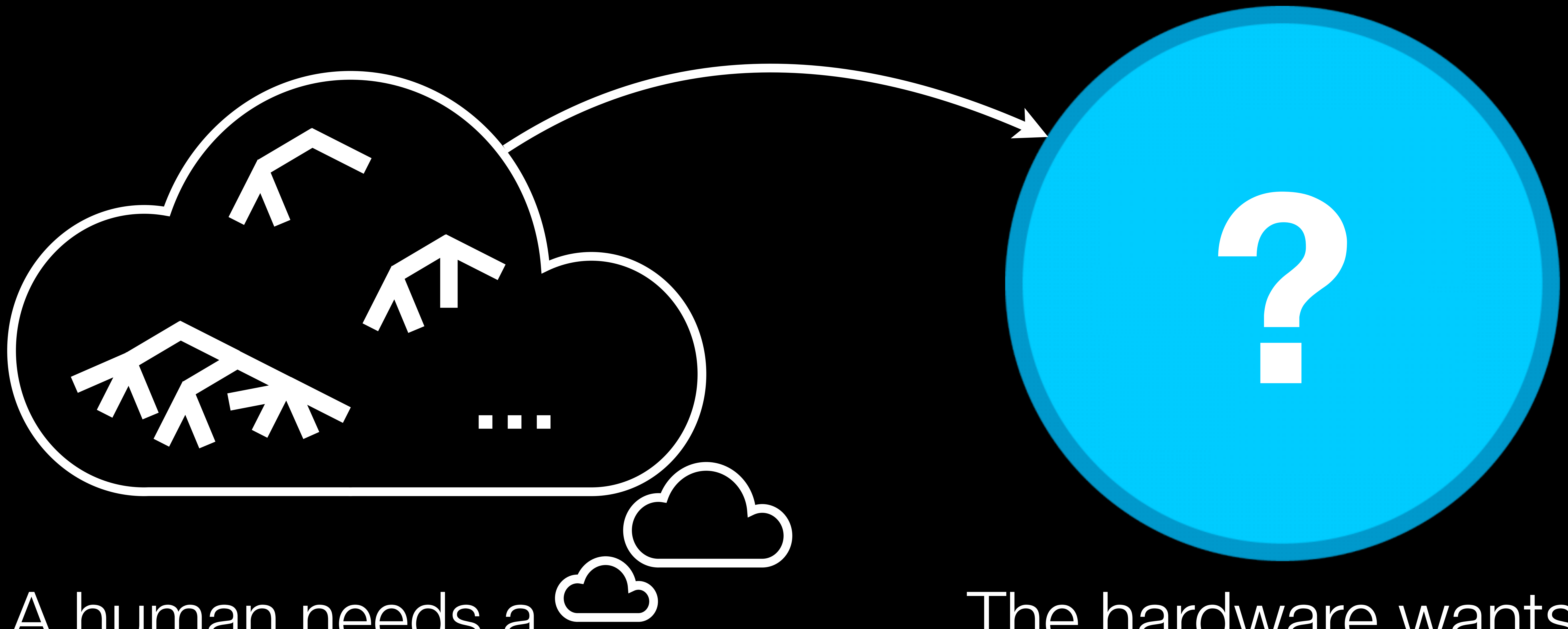
The hardware wants to support *one* tree.

No general way to deploy our gadget.

A human needs a *range* of trees.

The hardware wants to support *one* tree.

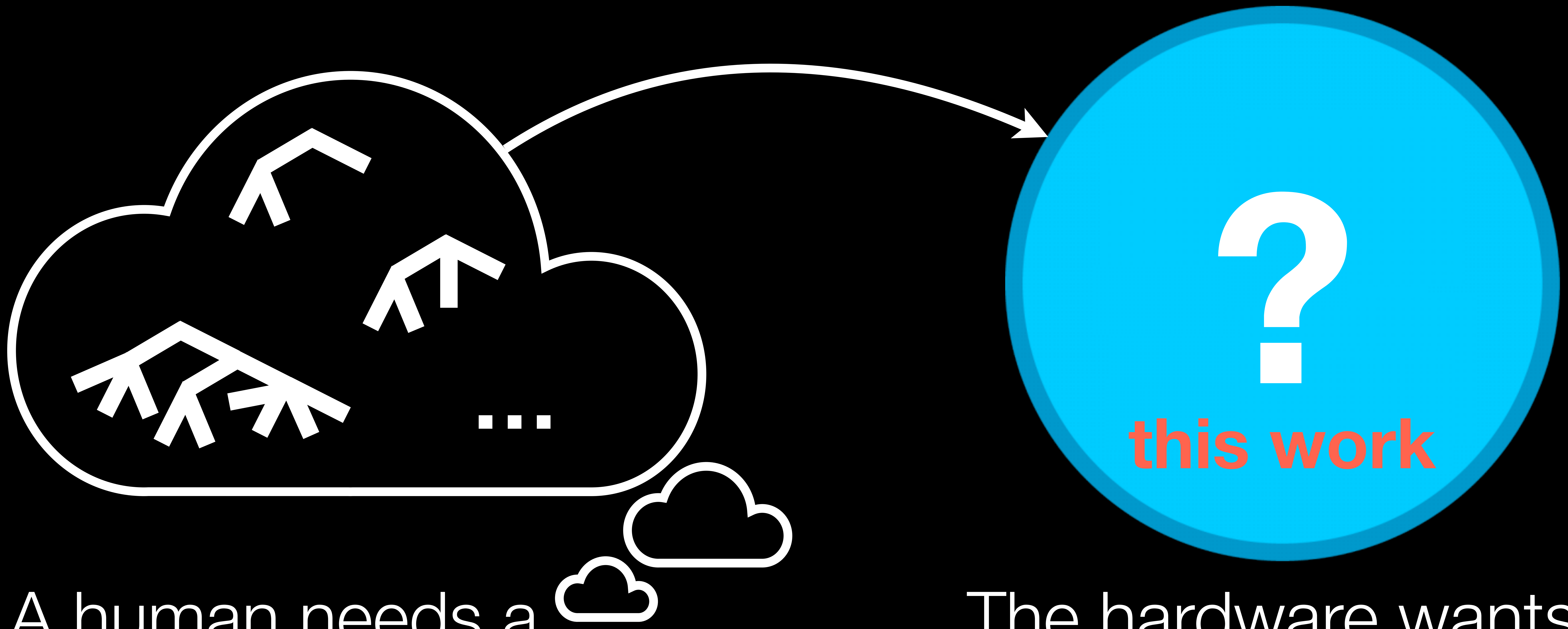
No general way to deploy our gadget.



A human needs a *range* of trees.

The hardware wants to support *one* tree.

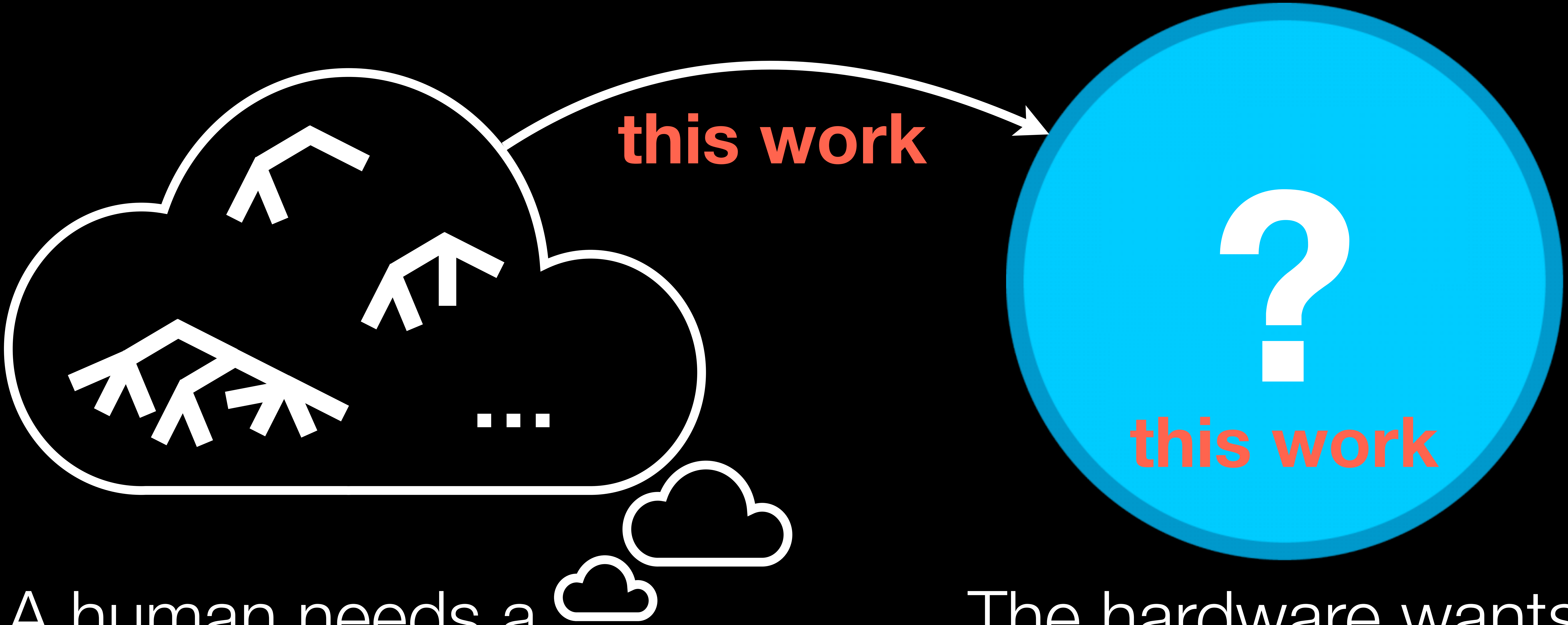
No general way to deploy our gadget.



A human needs a *range* of trees.

The hardware wants to support *one* tree.

No general way to deploy our gadget.

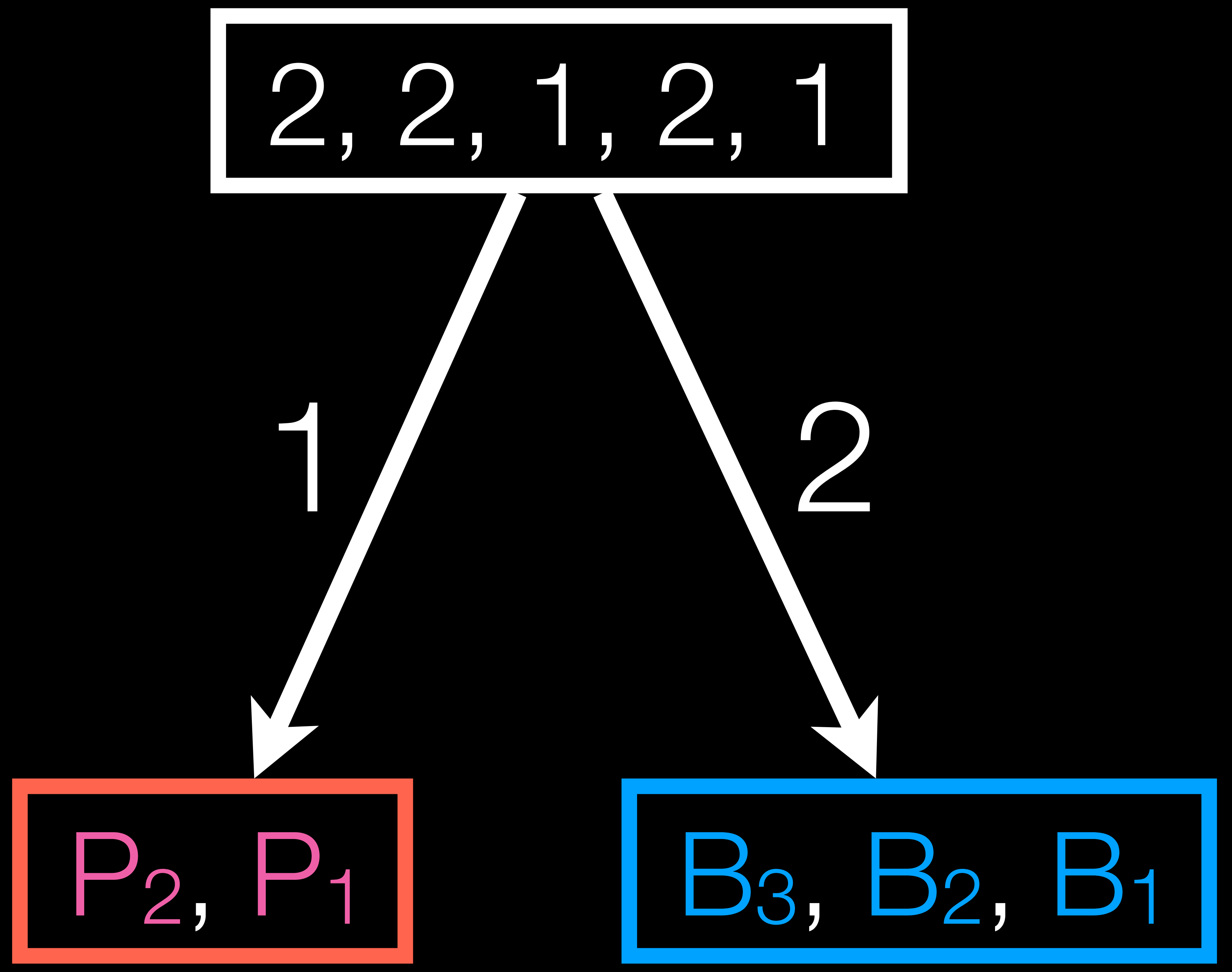


A human needs a *range* of trees.

The hardware wants to support *one* tree.

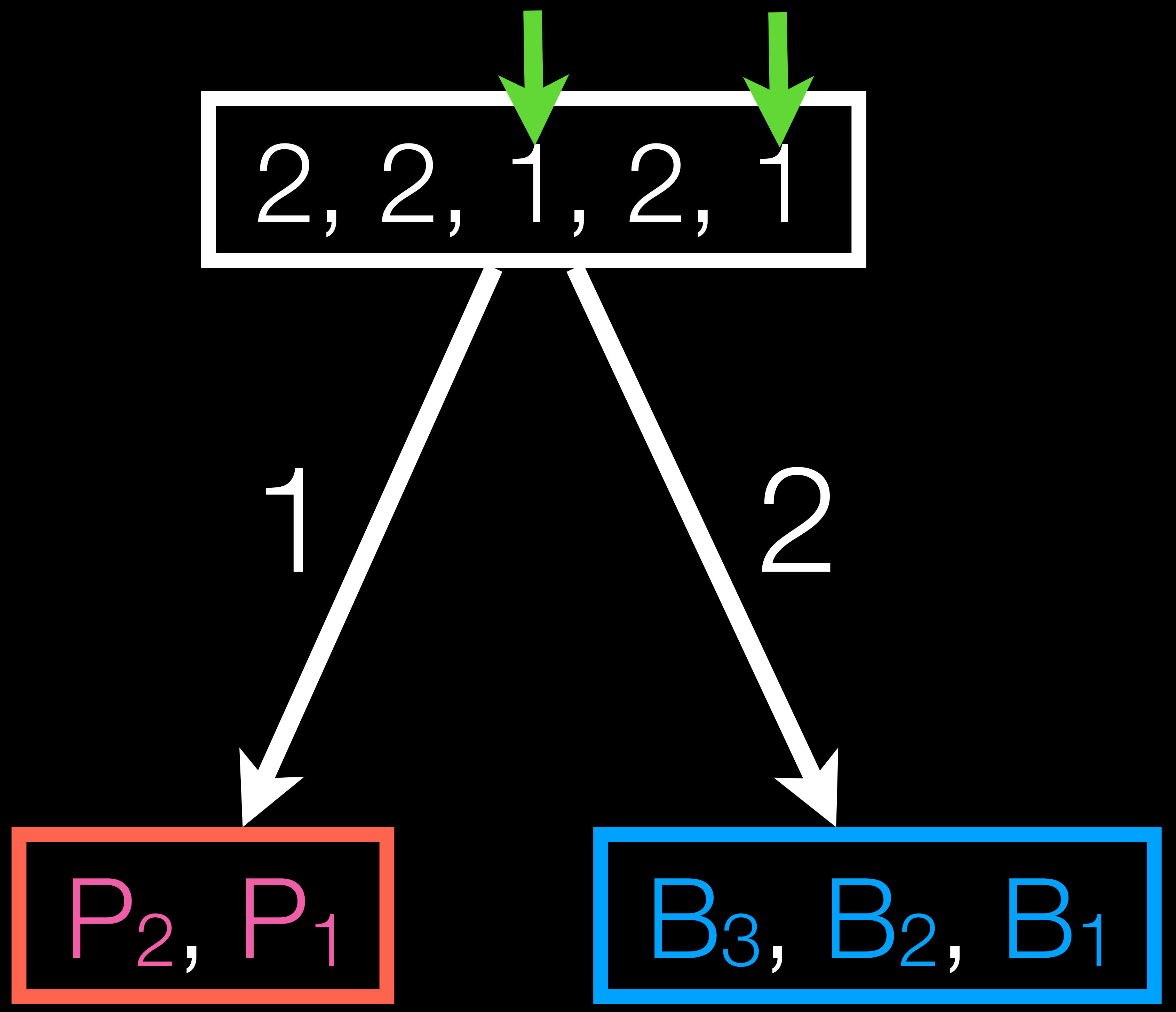
Aside: PIFO trees

interleave R and B;
interleave P and T.



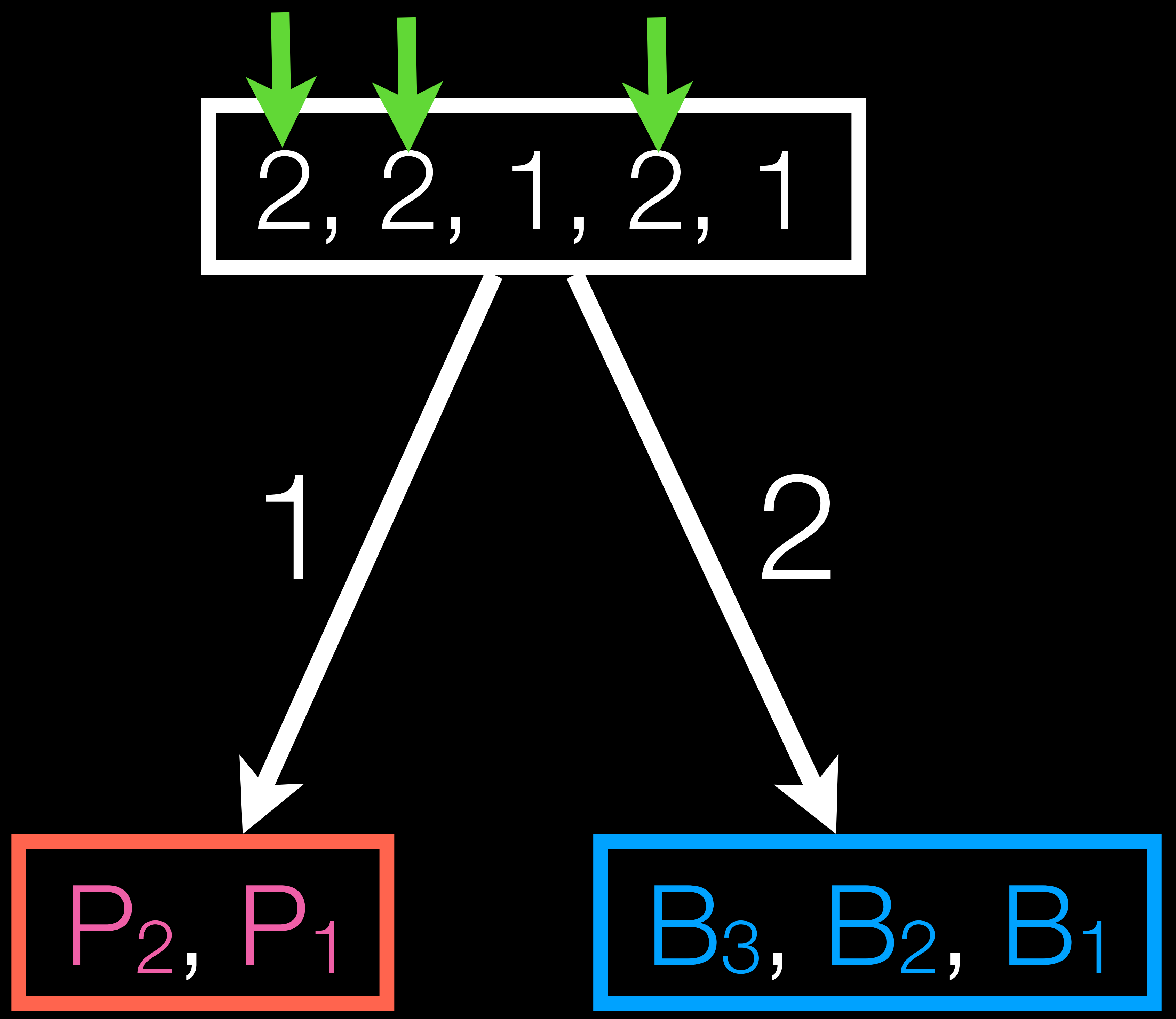
Aside: PIFO trees

interleave R and B;
interleave P and T.



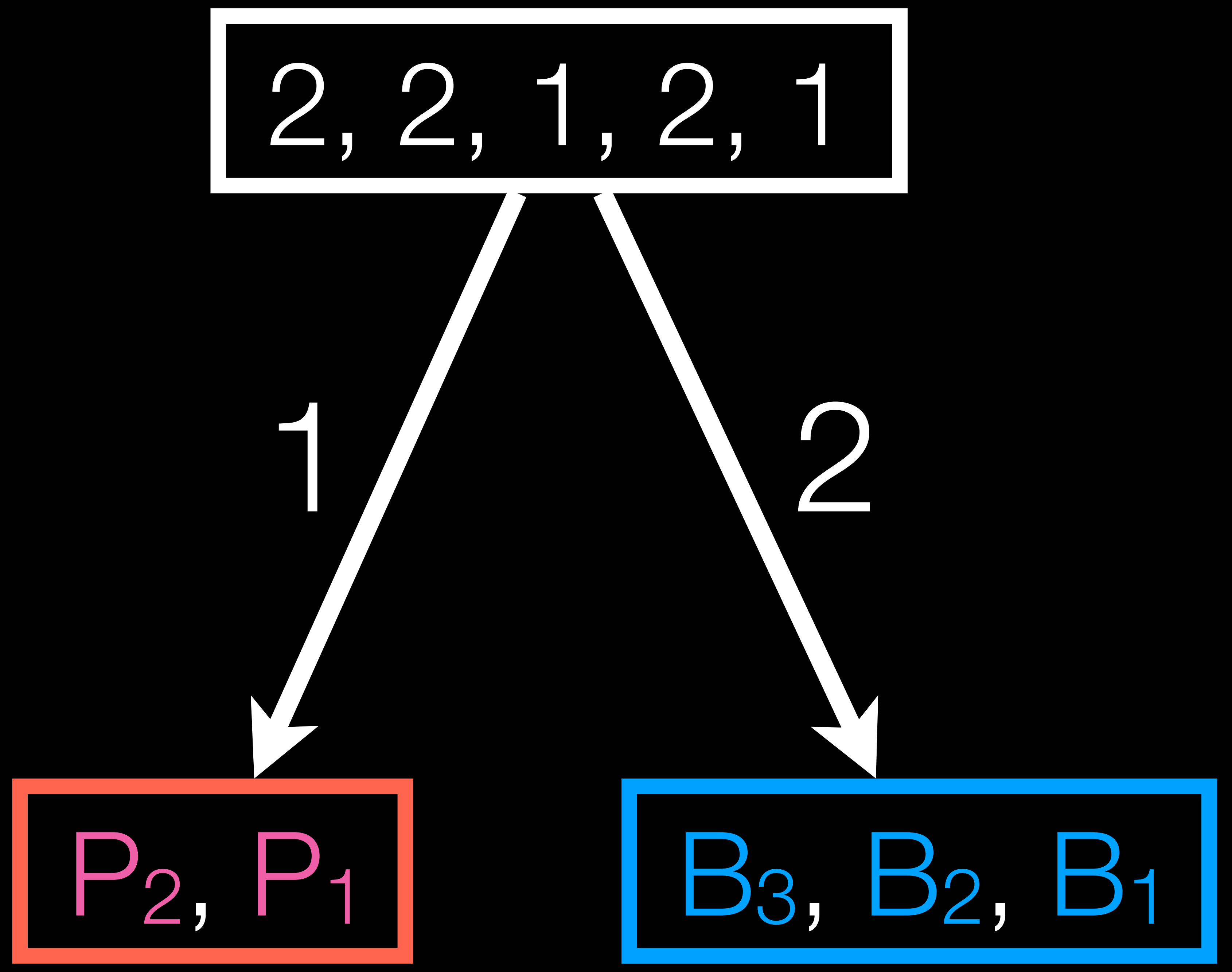
Aside: PIFO trees

interleave R and B;
interleave P and T.



Aside: PIFO trees

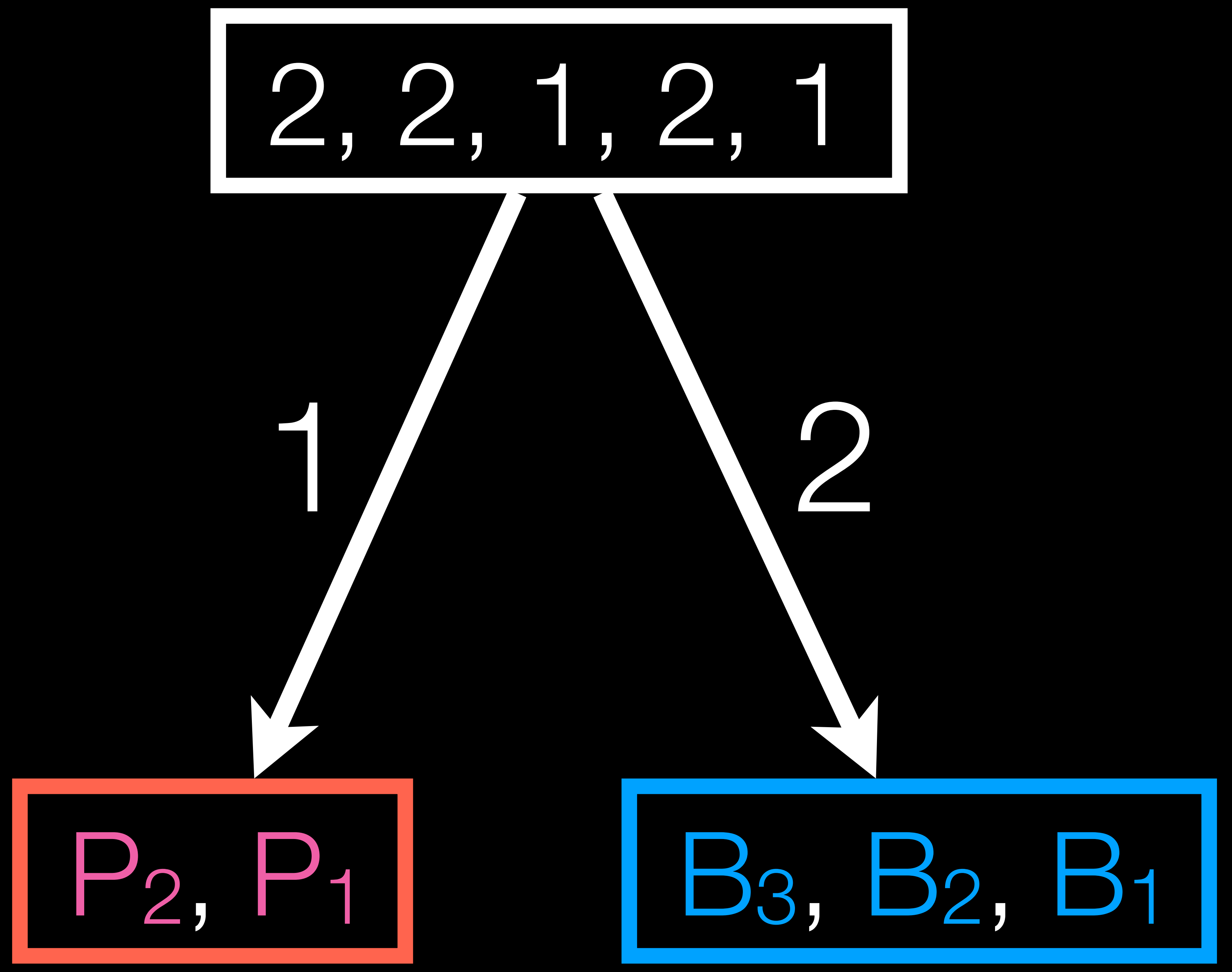
interleave R and B;
interleave P and T.



This behaves like a queue!

Aside: PIFO trees

interleave R and B;
interleave P and T.

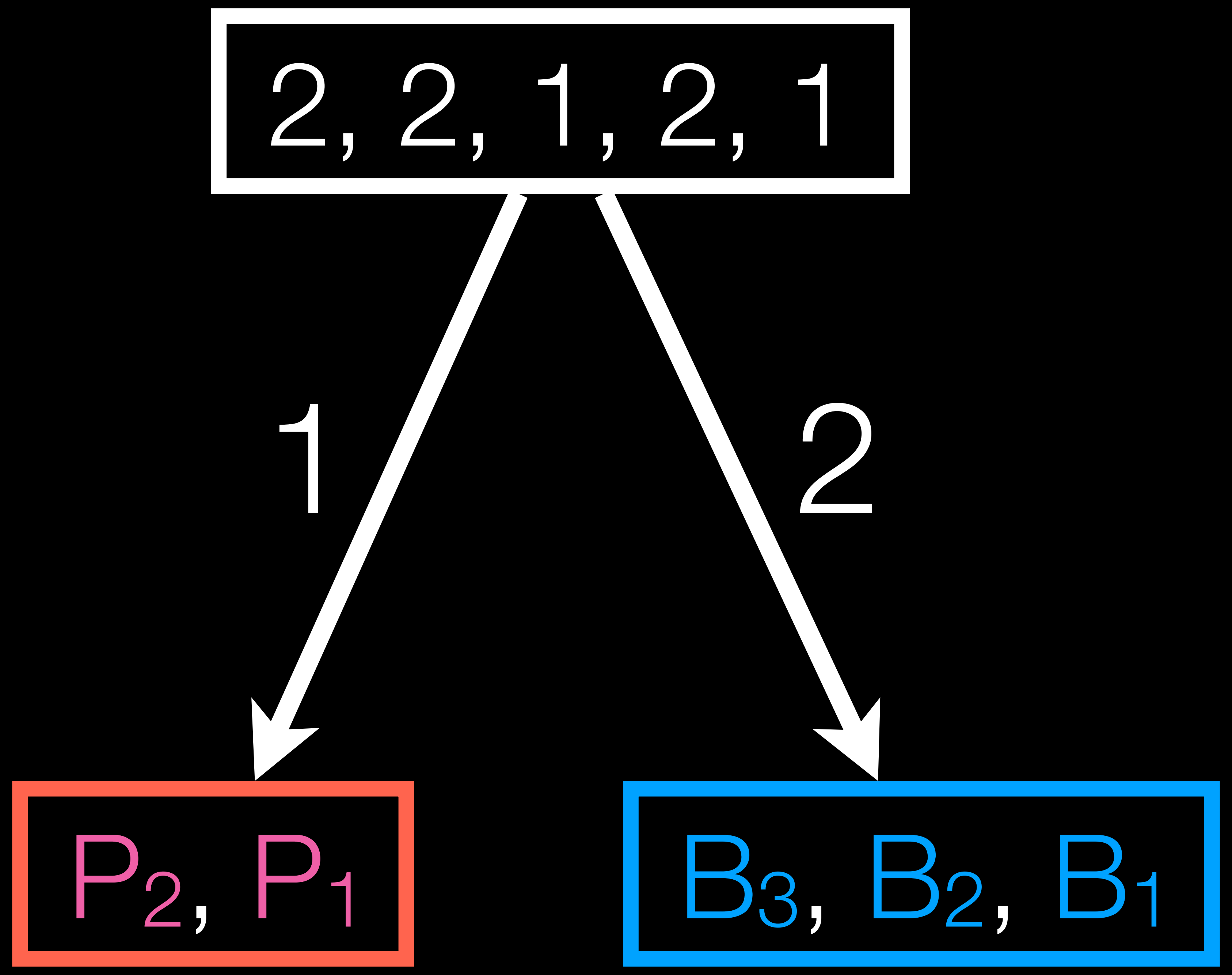


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

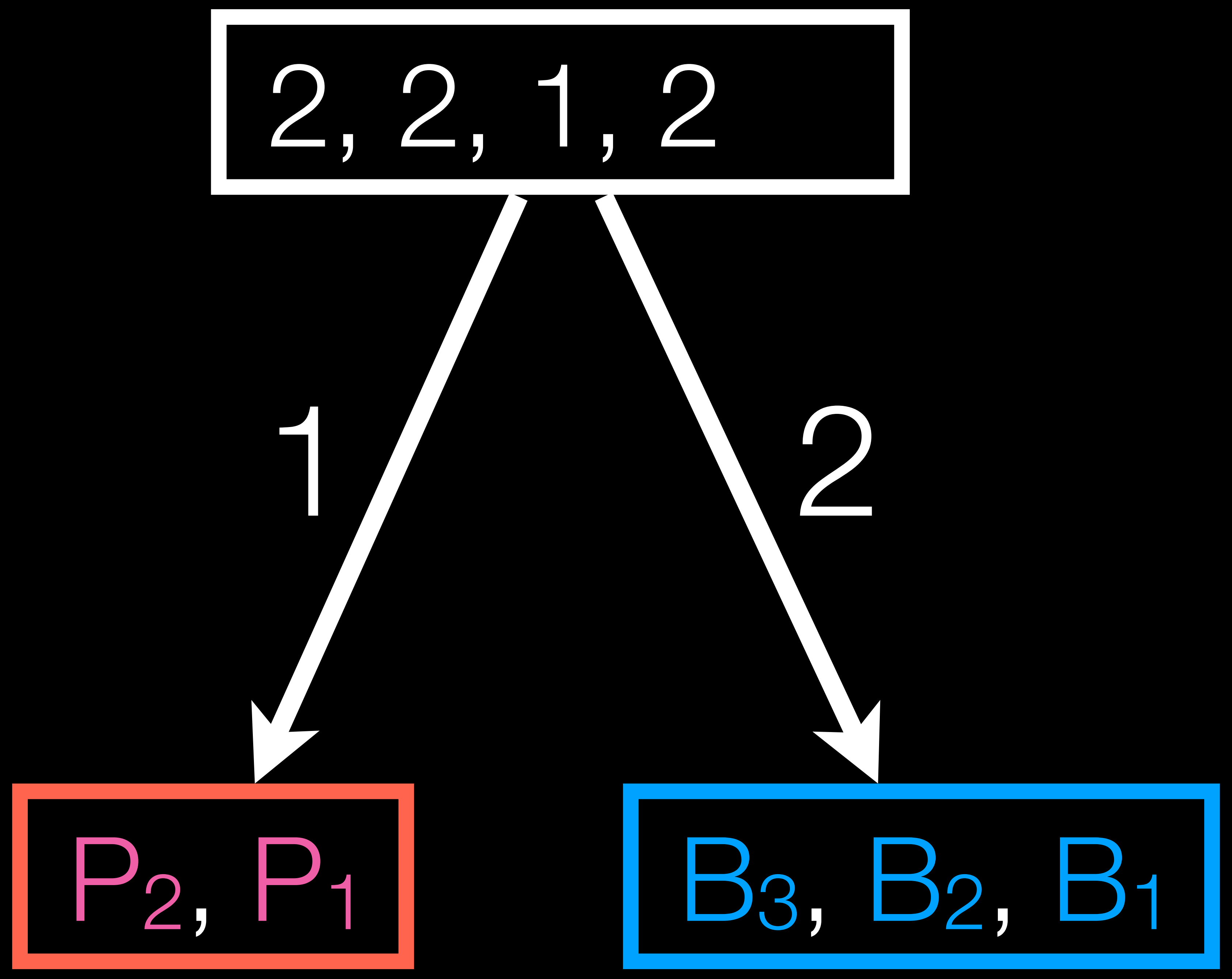


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

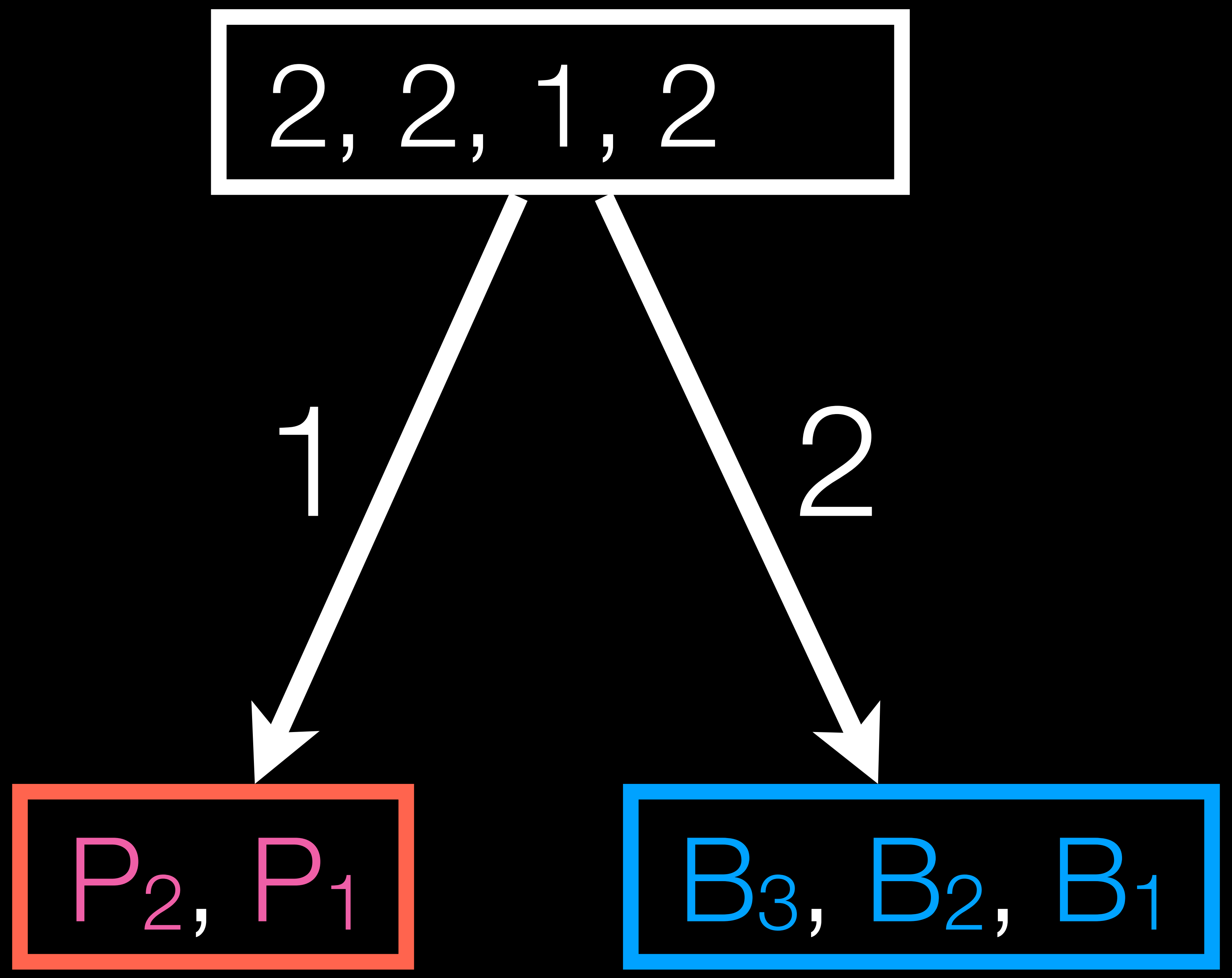


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

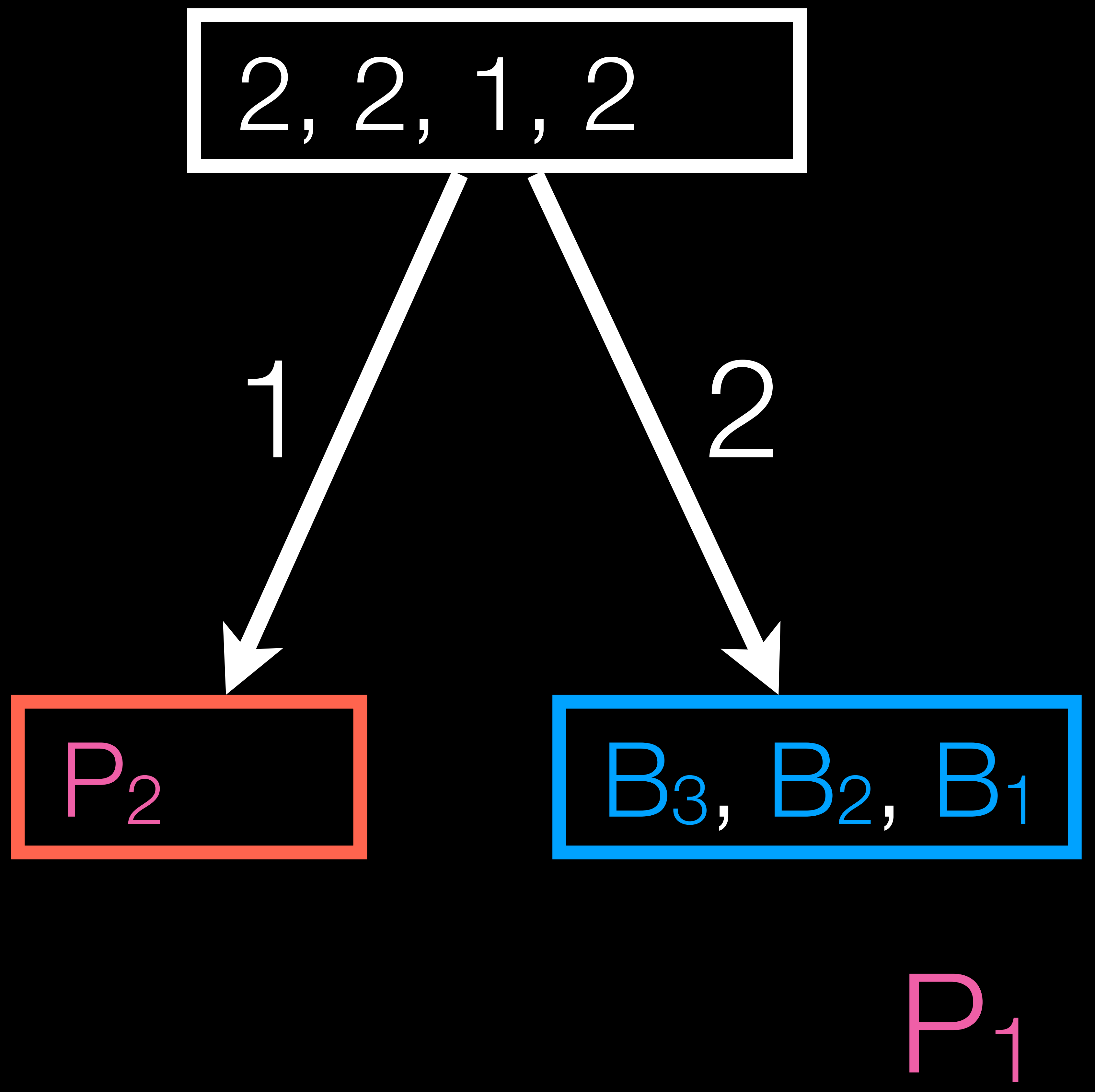


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

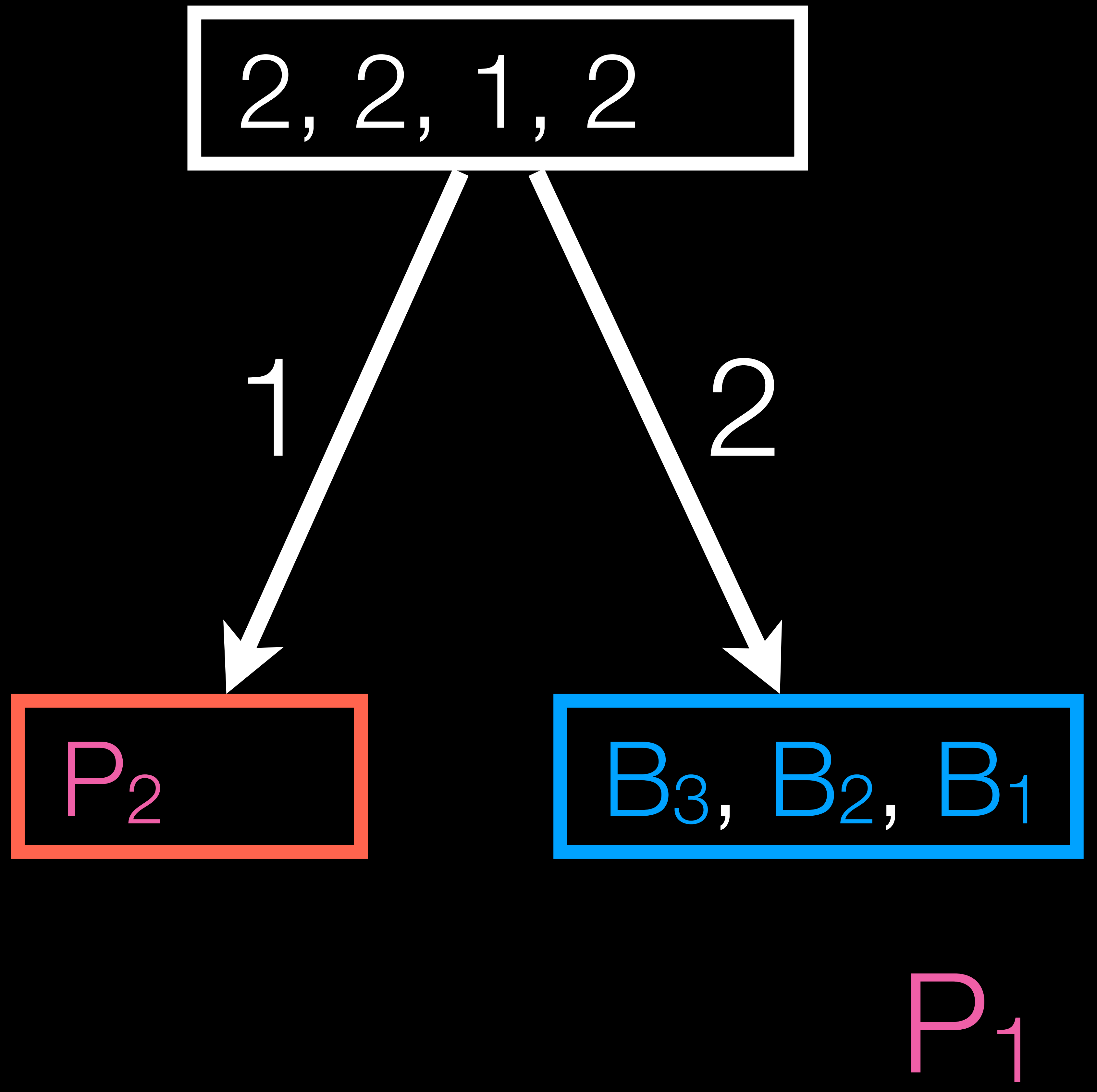


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

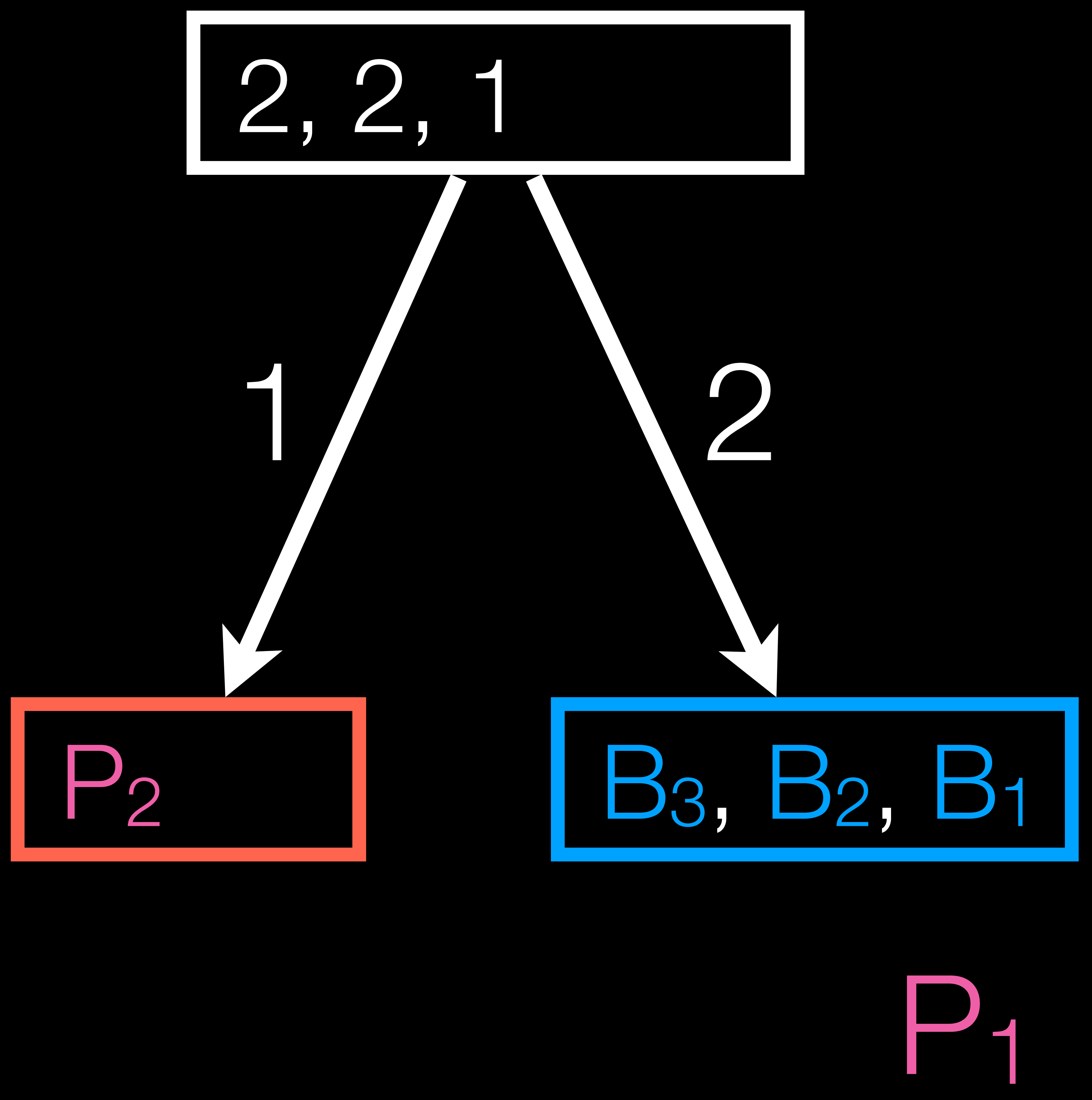


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

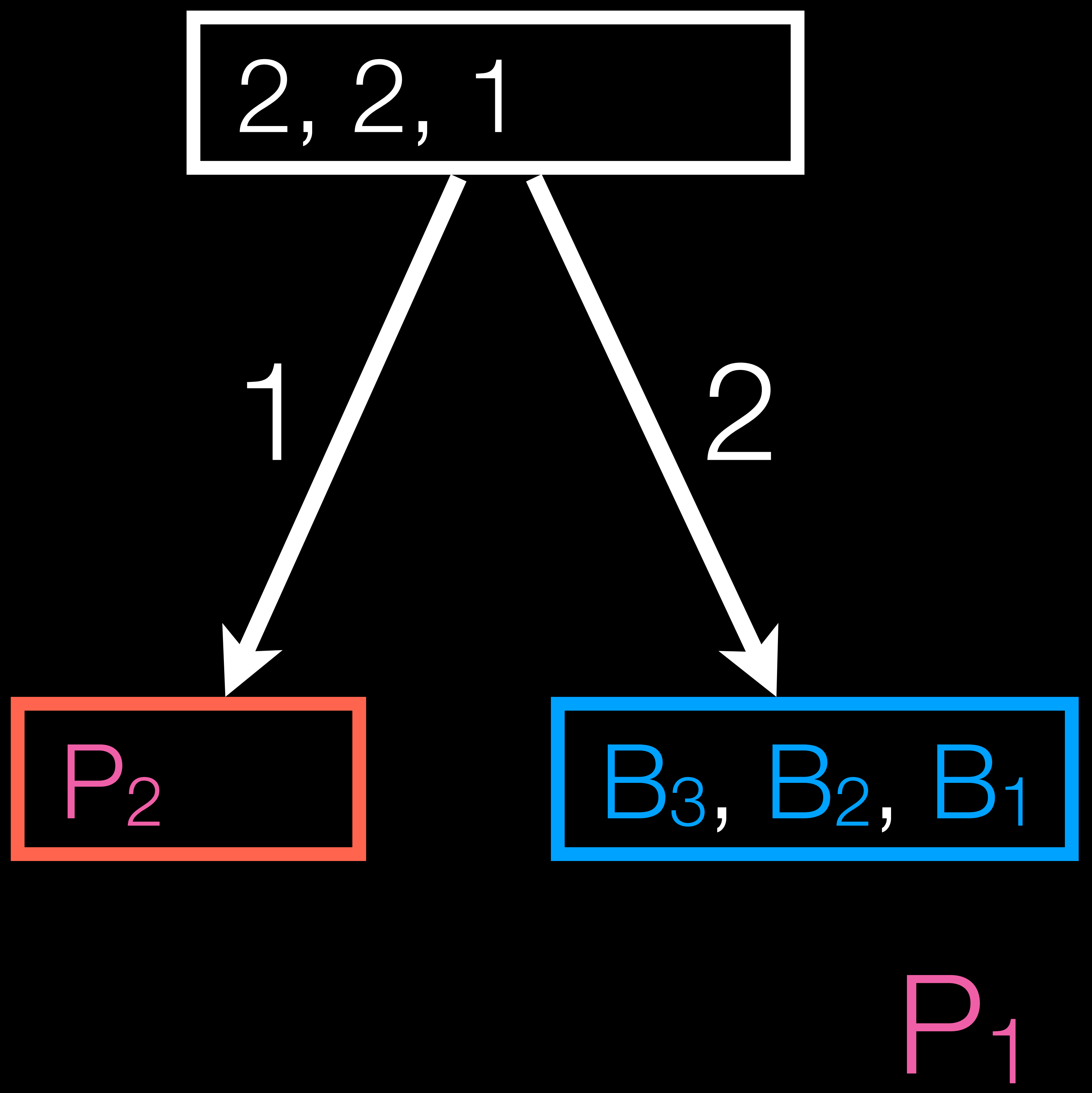


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

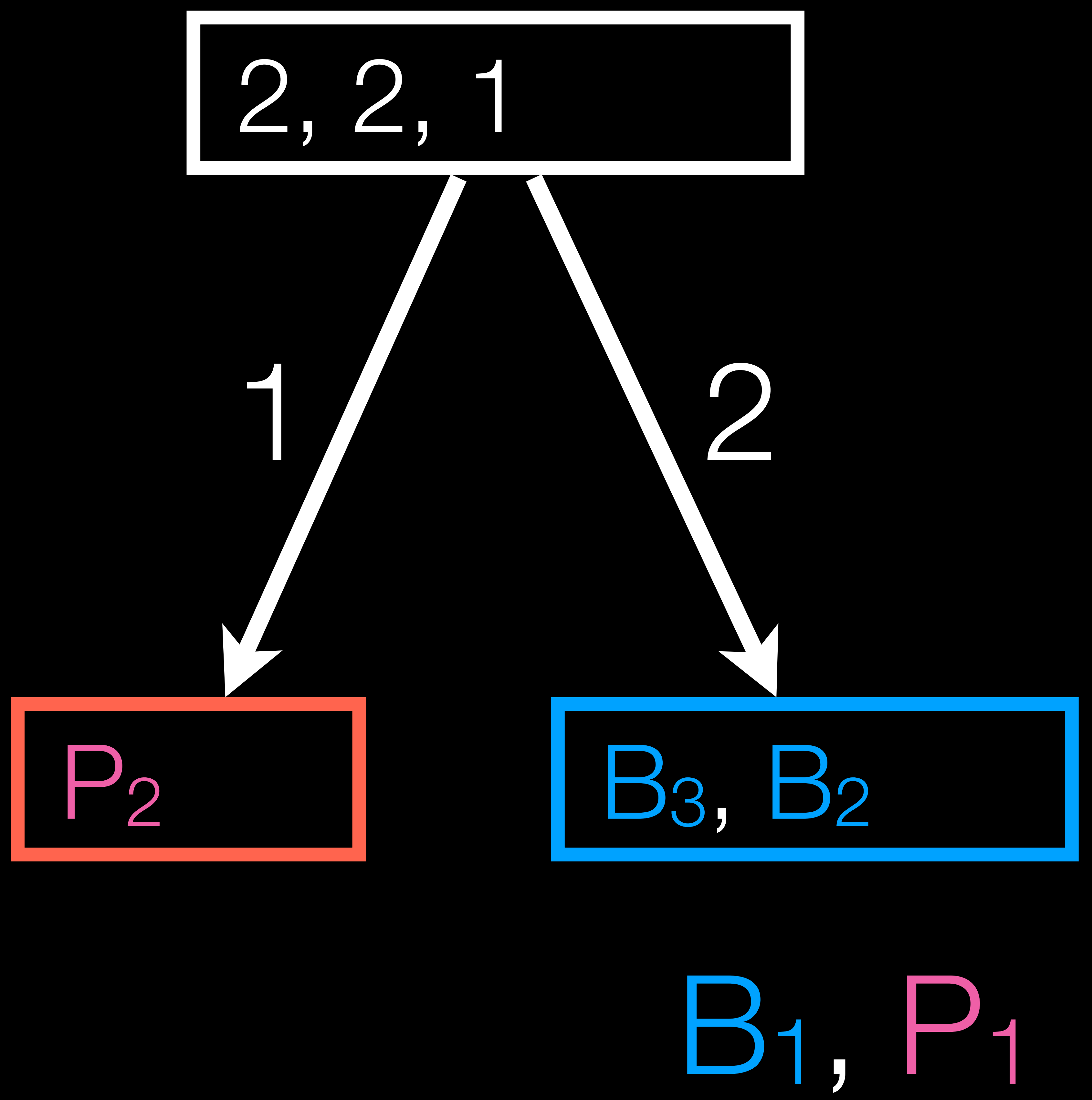


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

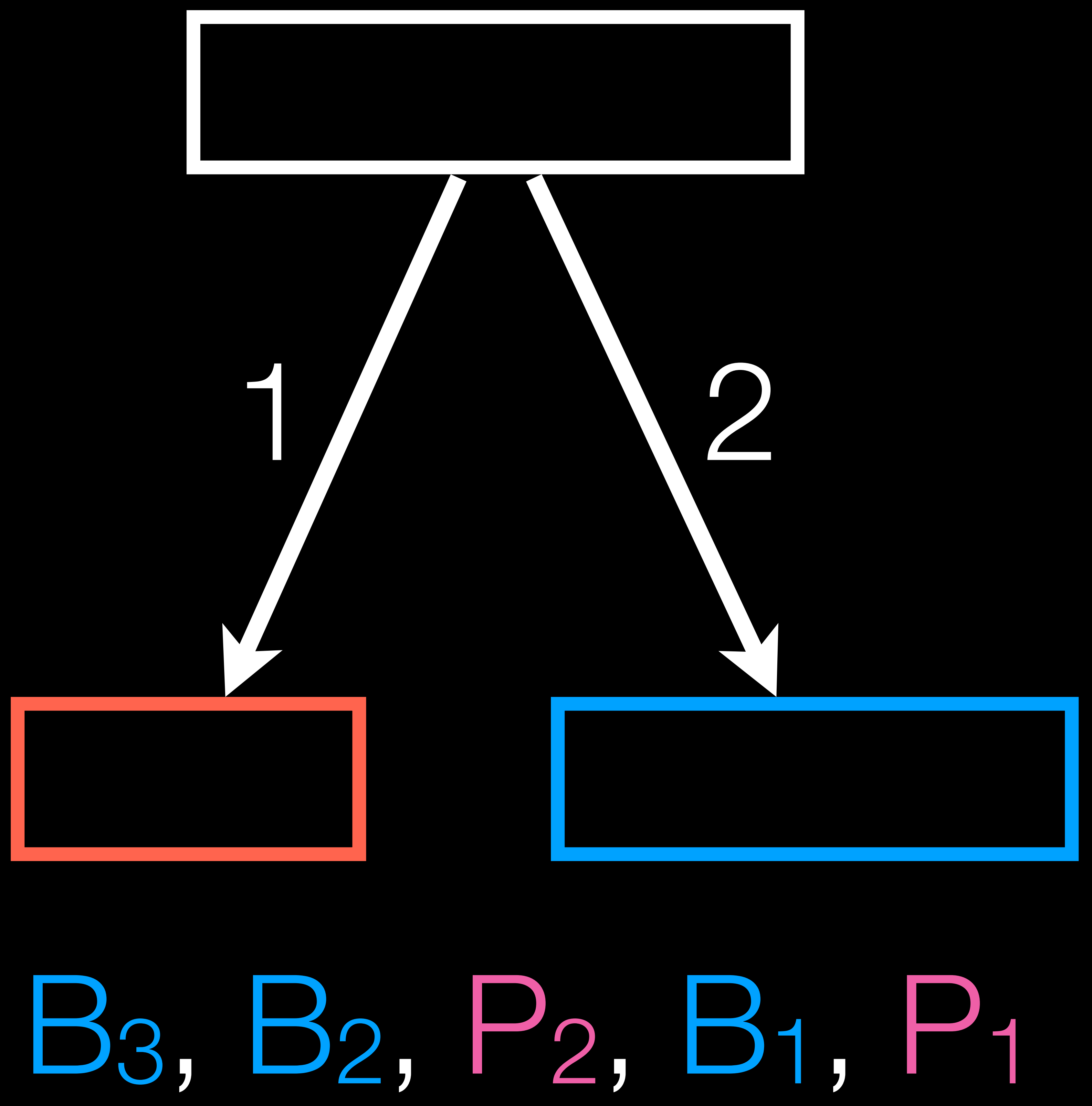


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.

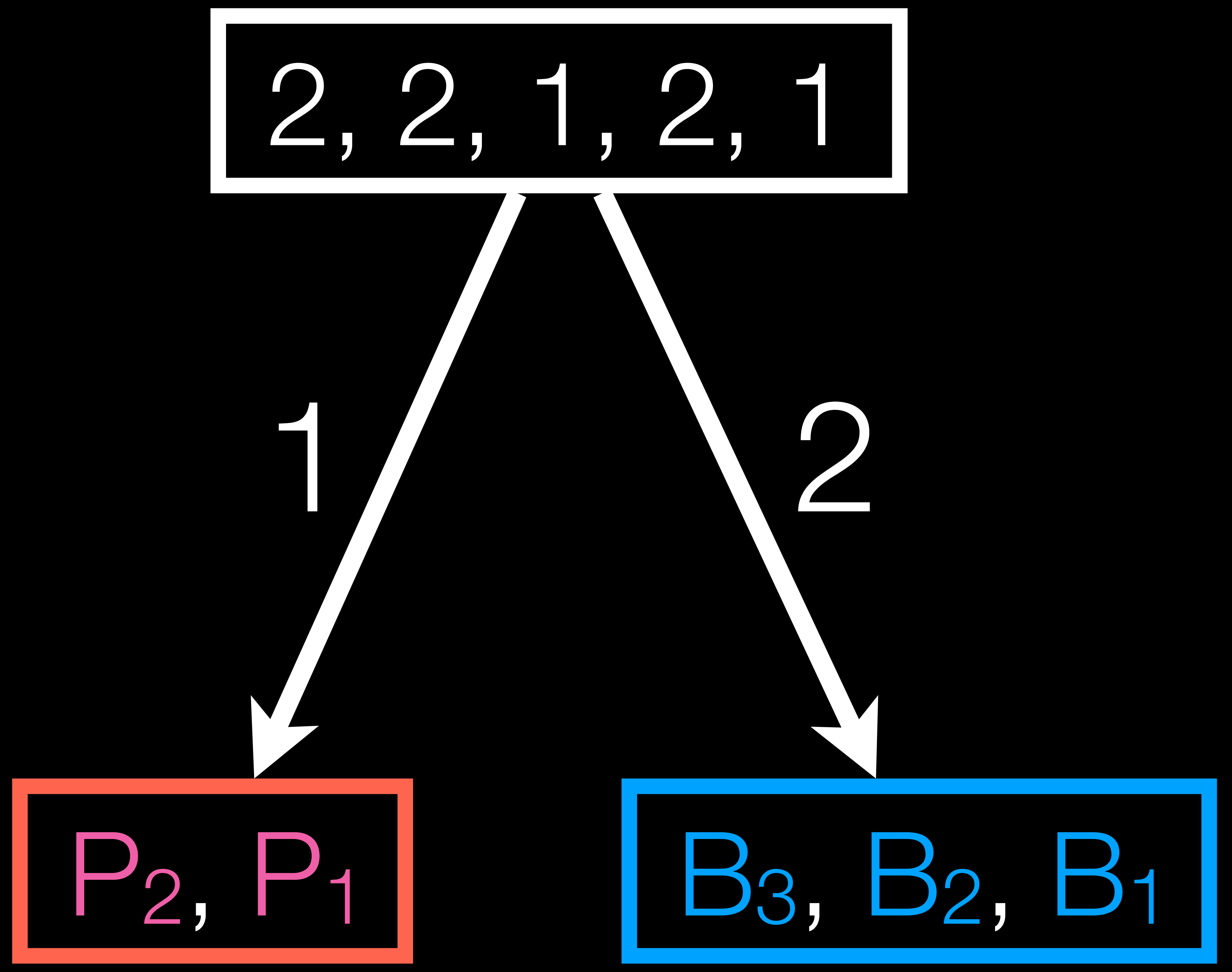


This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.



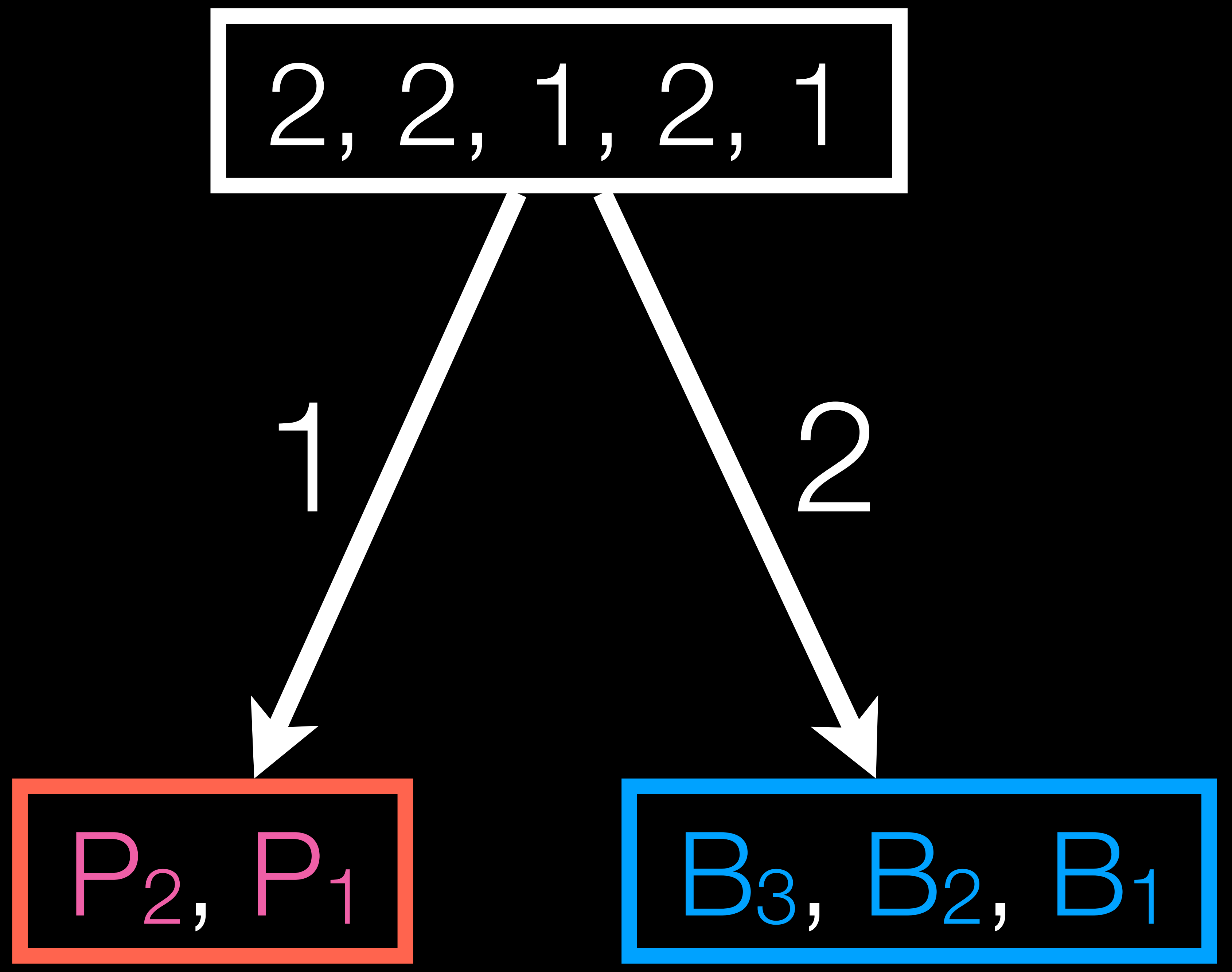
B_3, B_2, P_2, B_1, P_1

This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.



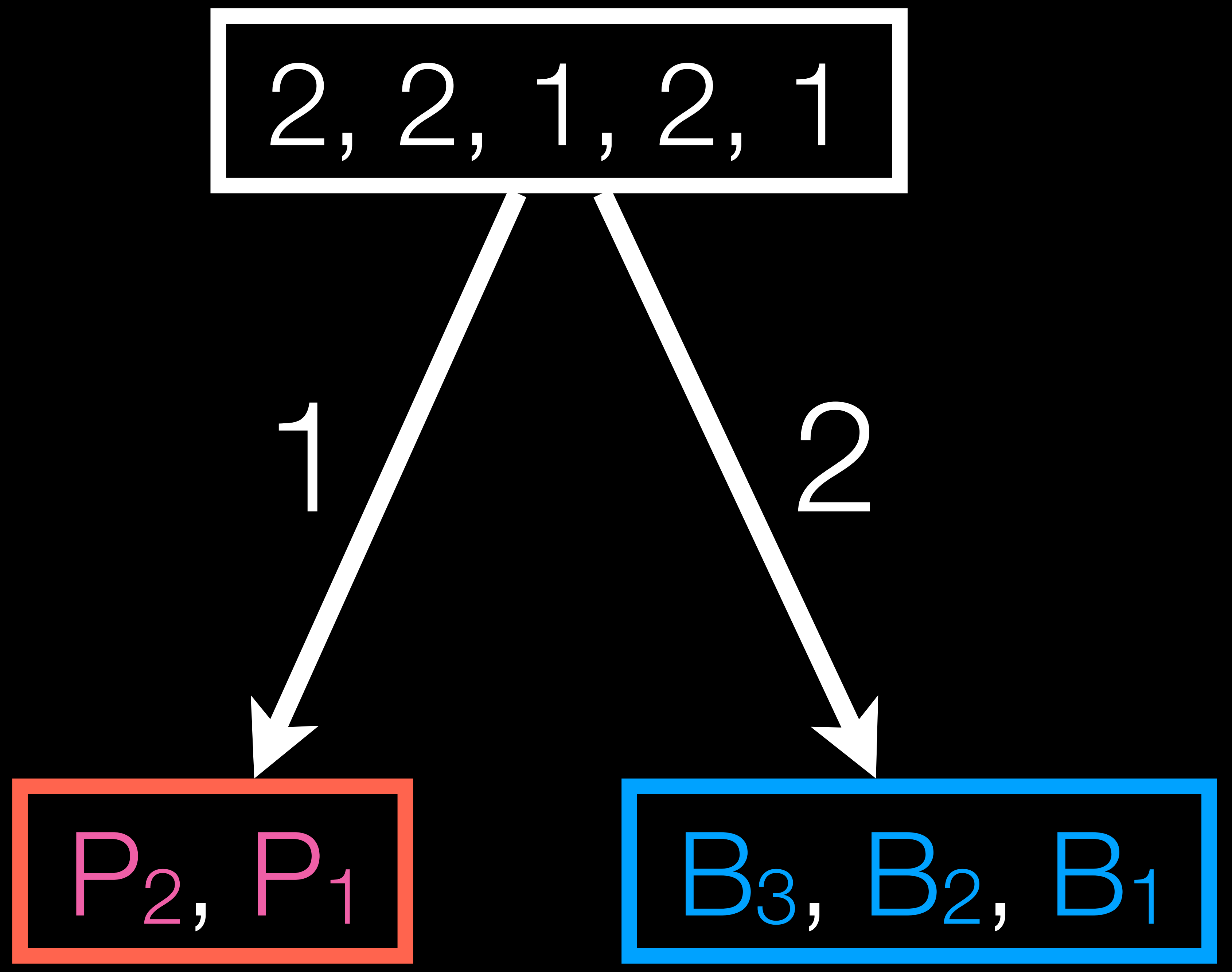
$B_3, B_2, \overline{P_2}, B_1, \overline{P_1}$

This behaves like a queue!

How do we pop it?

Aside: PIFO trees

interleave R and B;
interleave P and T.



$B_3, B_2, \overline{P_2}, B_1, \overline{P_1}$

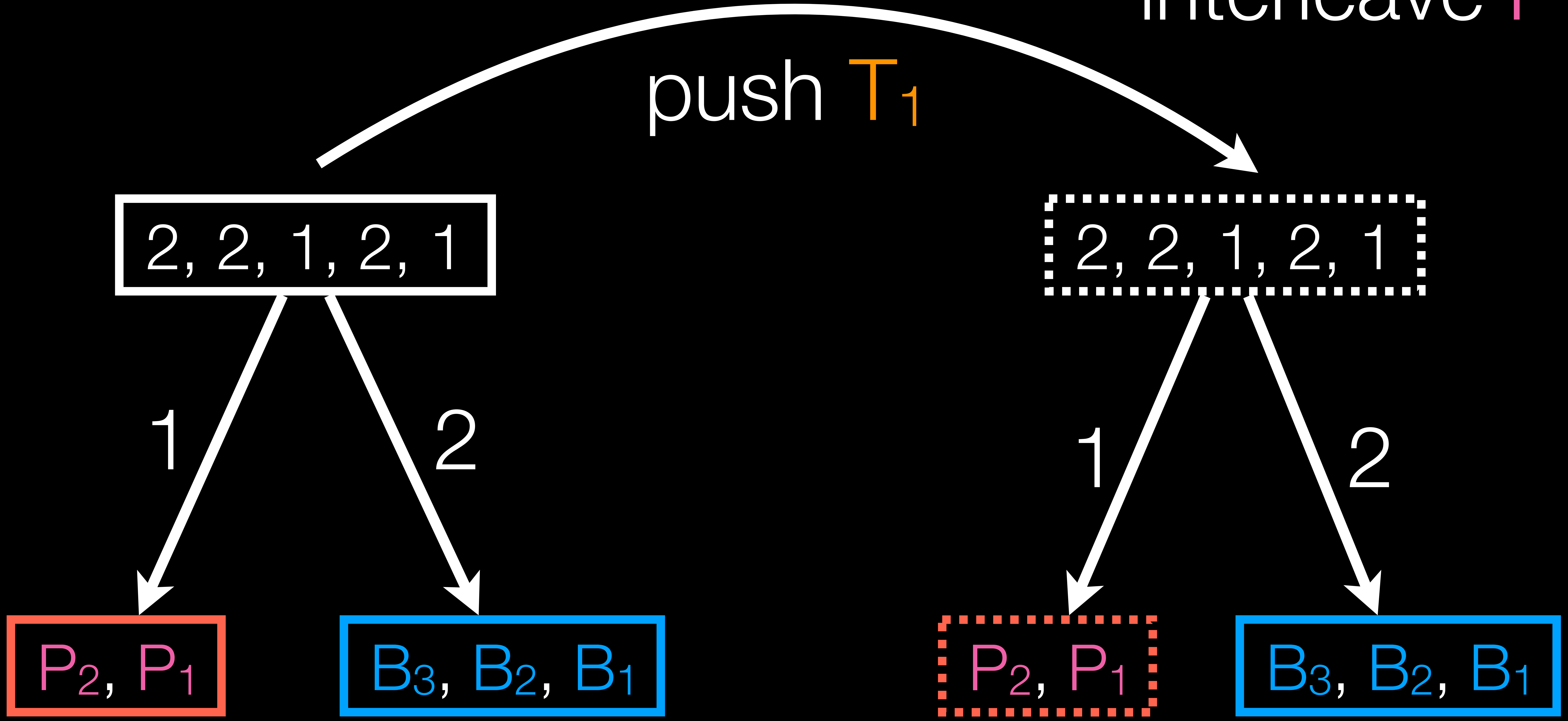
This behaves like a queue!

How do we pop it?

How do we push into it?

Aside: PIFO trees

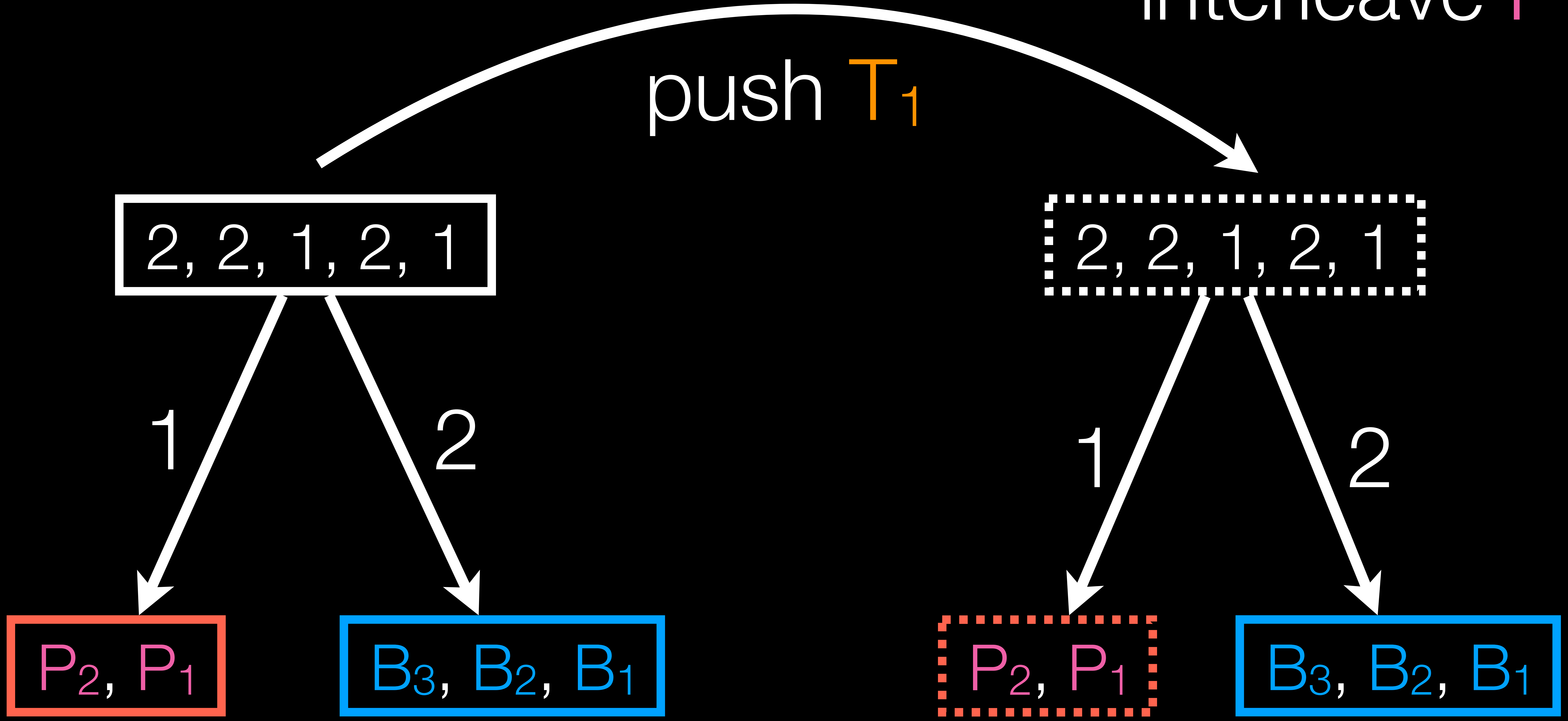
interleave R and B;
interleave P and T.



B_3, B_2, $\overline{P_2}$, B_1, $\overline{P_1}$

Aside: PIFO trees

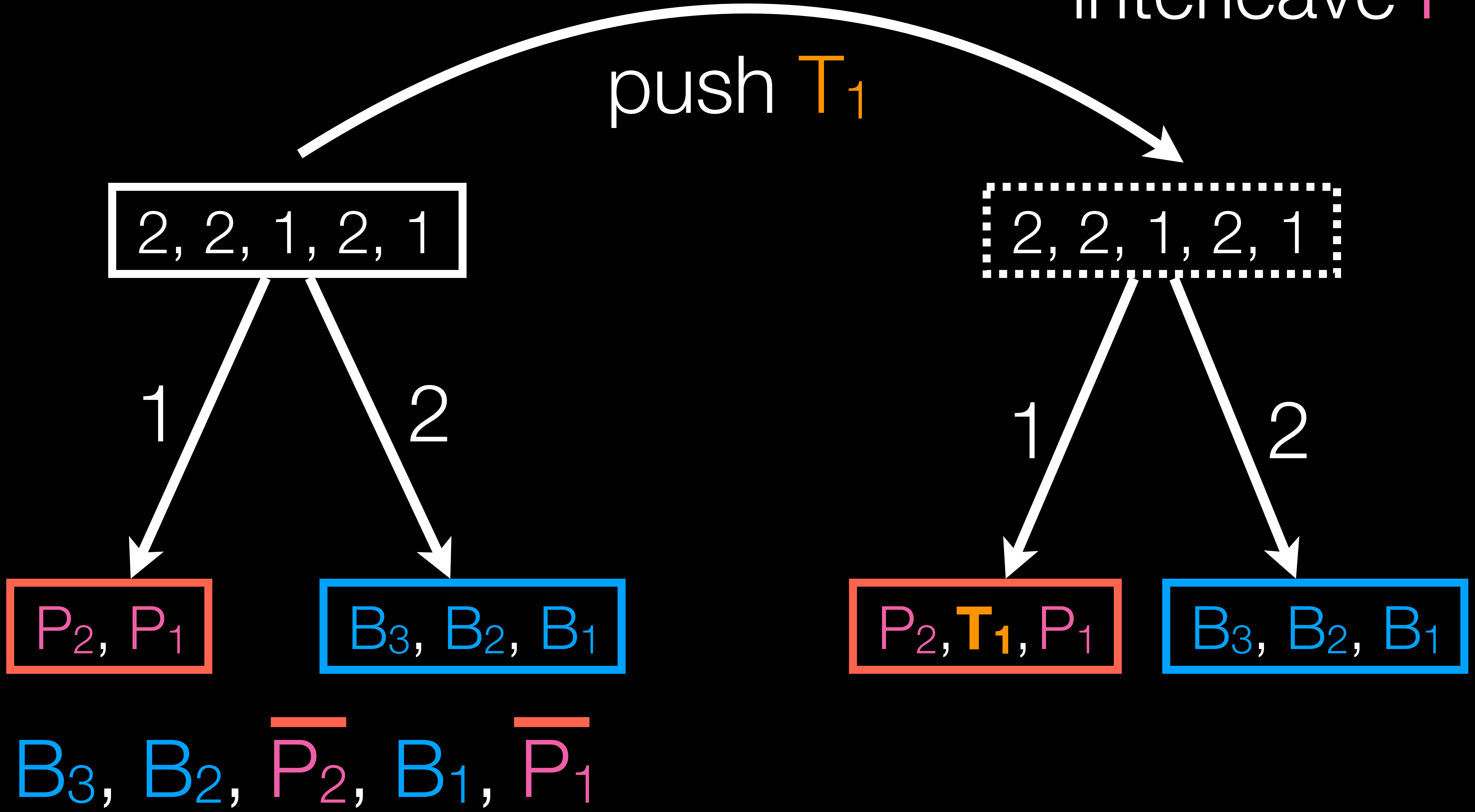
interleave R and B;
interleave P and T.



$B_3, B_2, \overline{P_2}, B_1, \overline{P_1}$

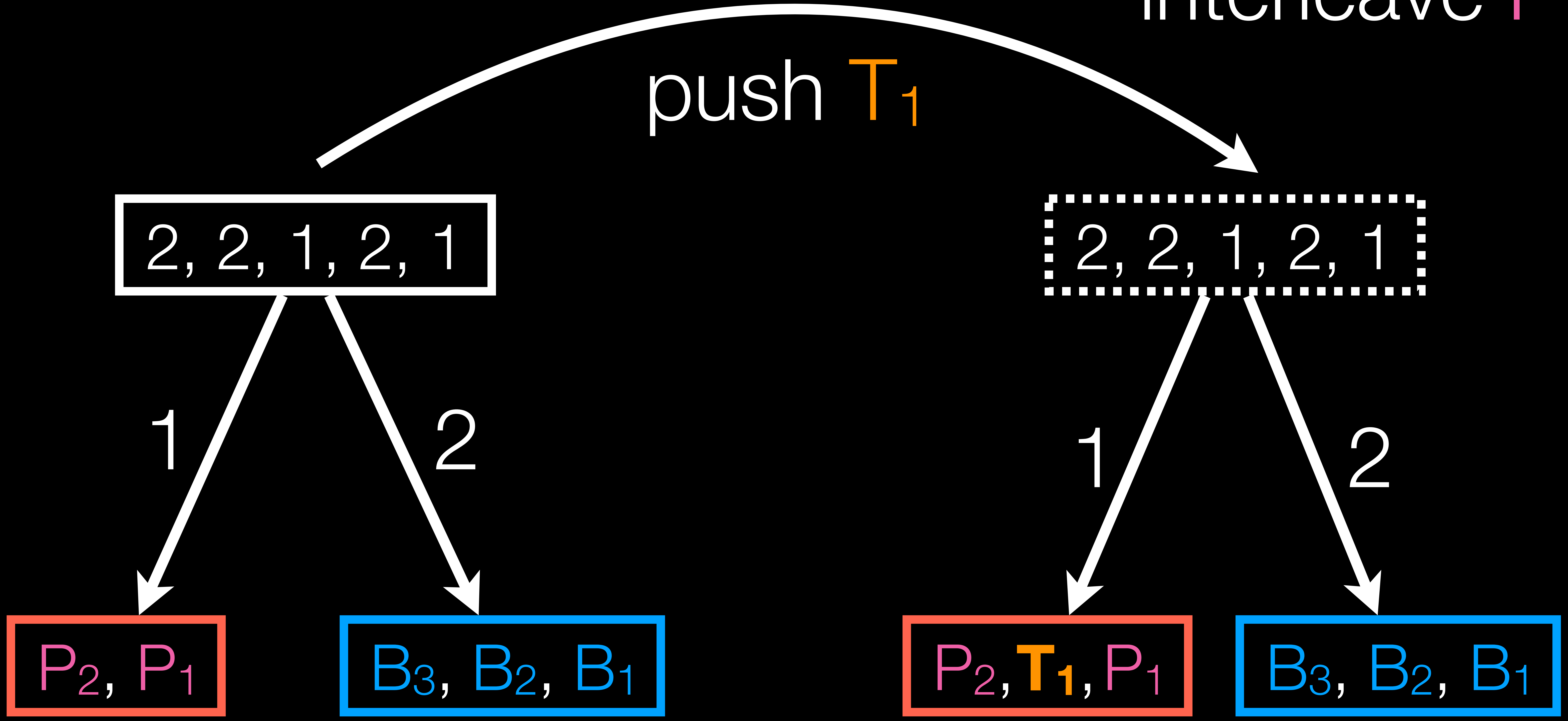
Aside: PIFO trees

interleave R and B;
interleave P and T.



Aside: PIFO trees

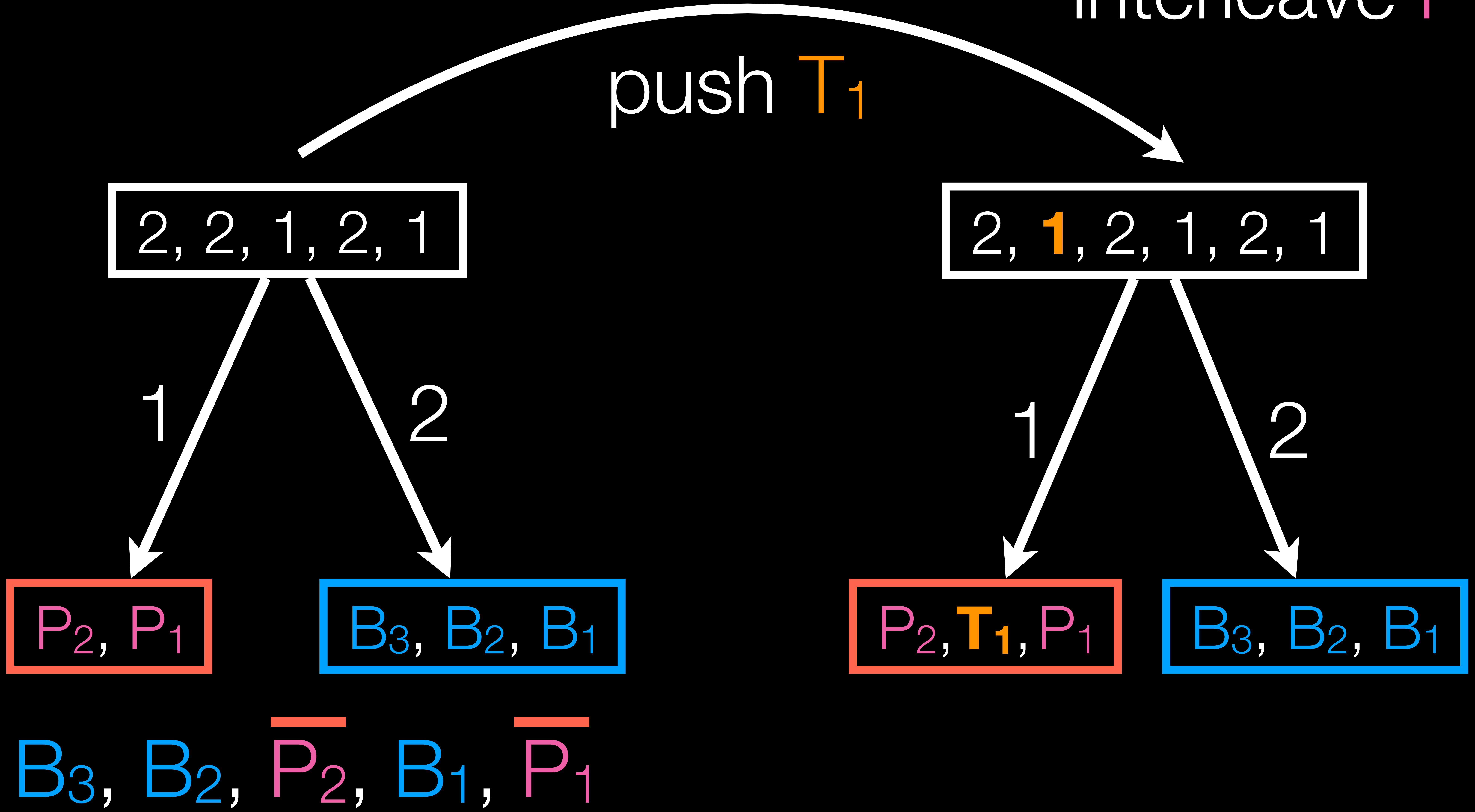
interleave R and B;
interleave P and T.



B_3, B_2, $\overline{P_2}$, B_1, $\overline{P_1}$

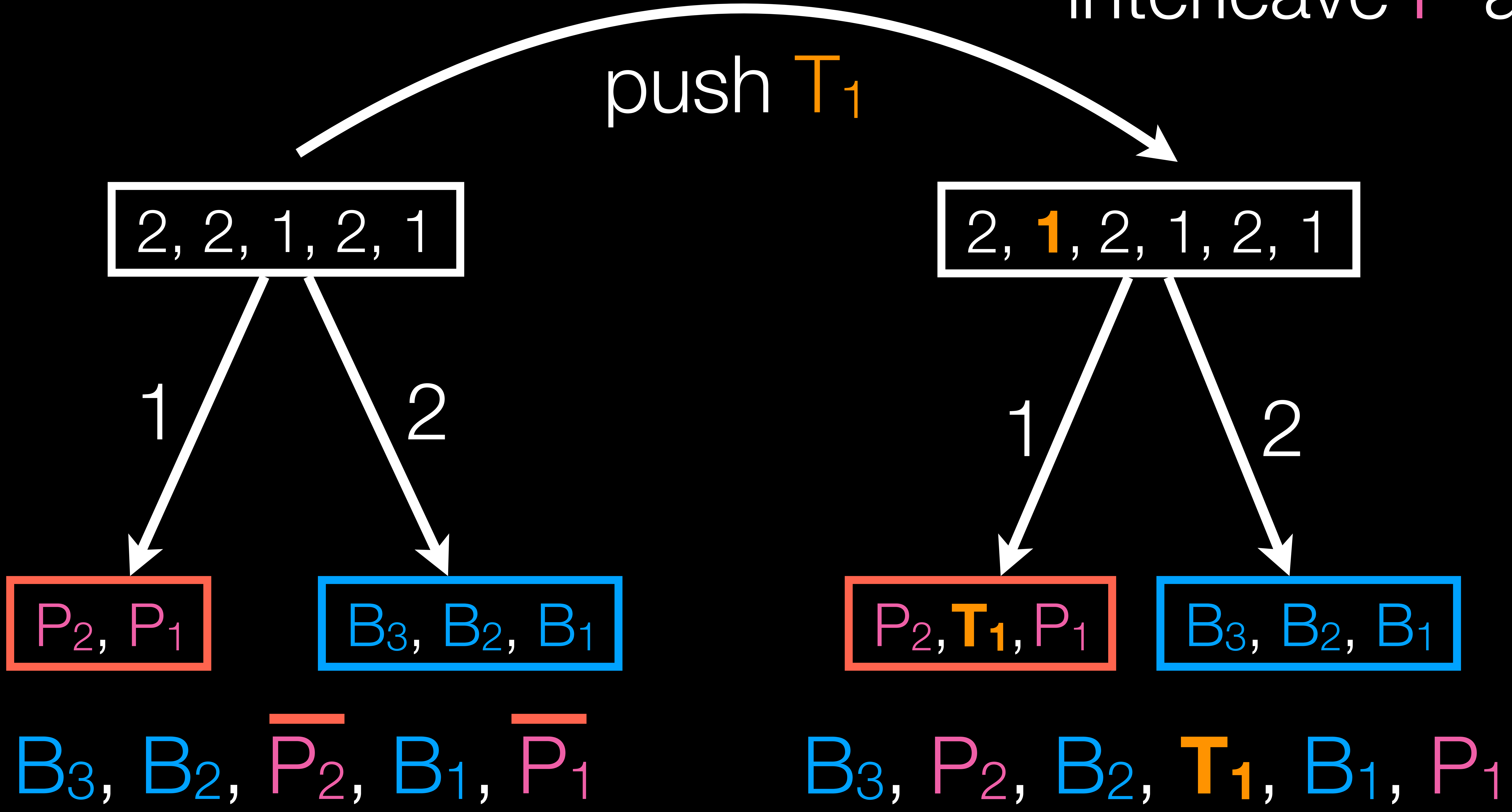
Aside: PIFO trees

interleave R and B;
interleave P and T.



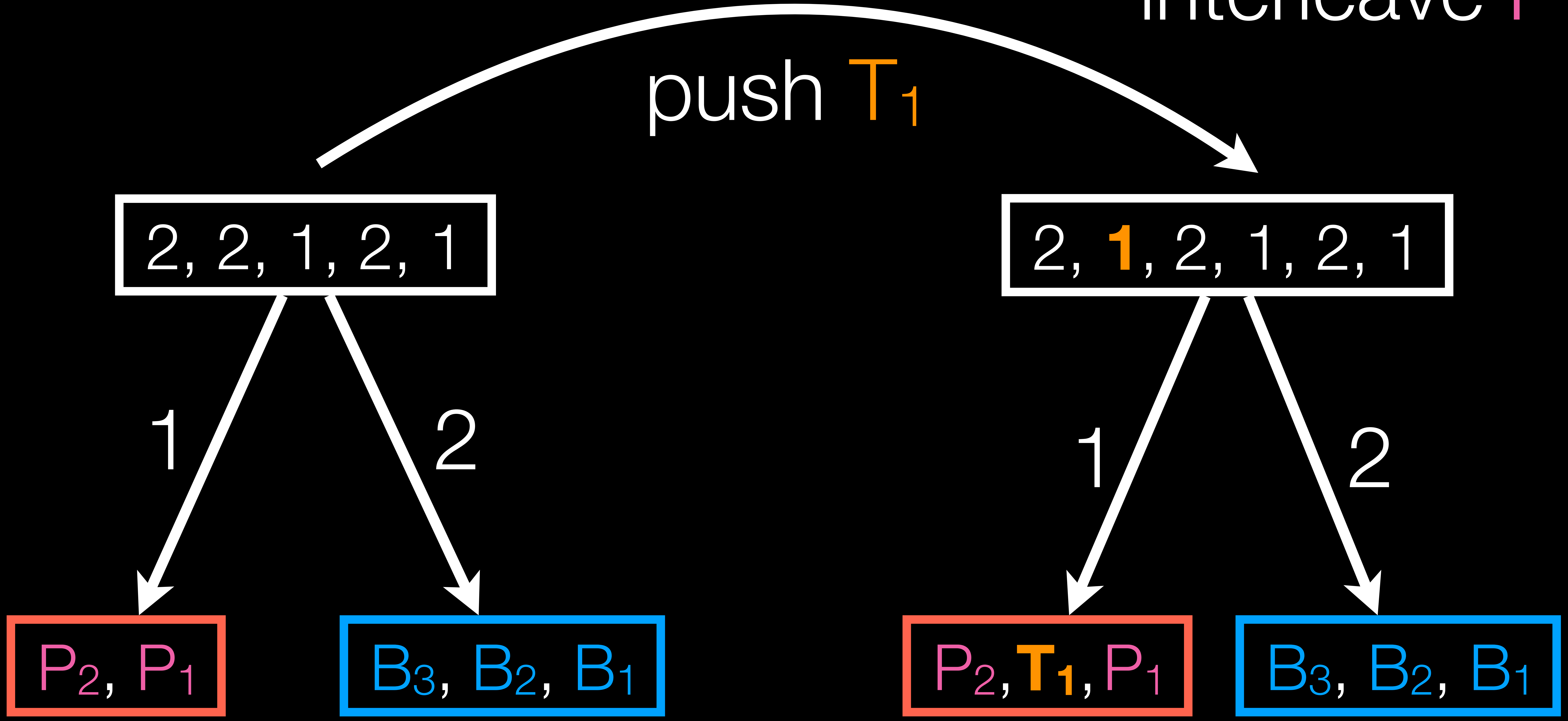
Aside: PIFO trees

interleave R and B;
interleave P and T.



Aside: PIFO trees

interleave R and B;
interleave P and T.

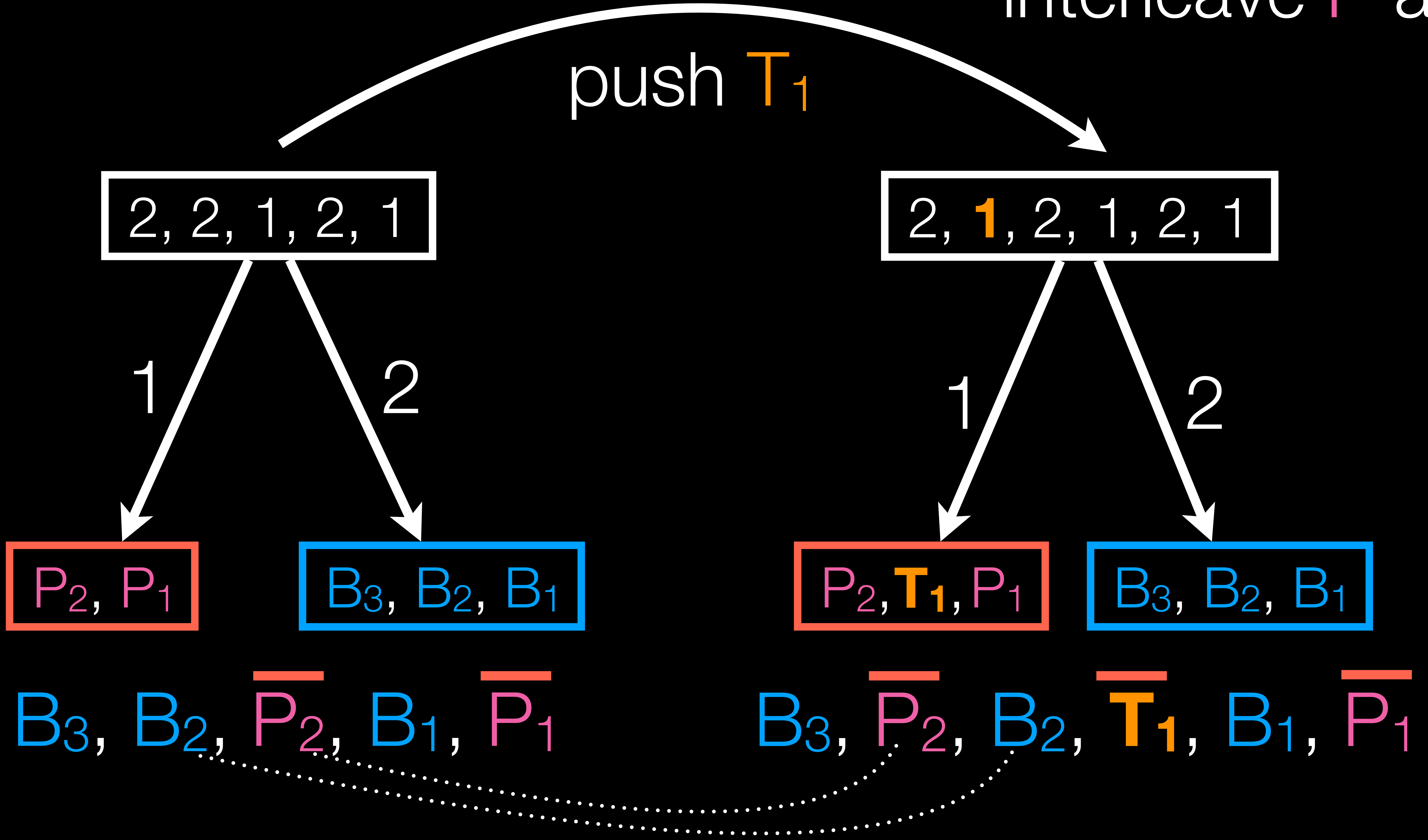


B3, B2, P2, B1, P1

B3, P2, B2, T1, B1, P1

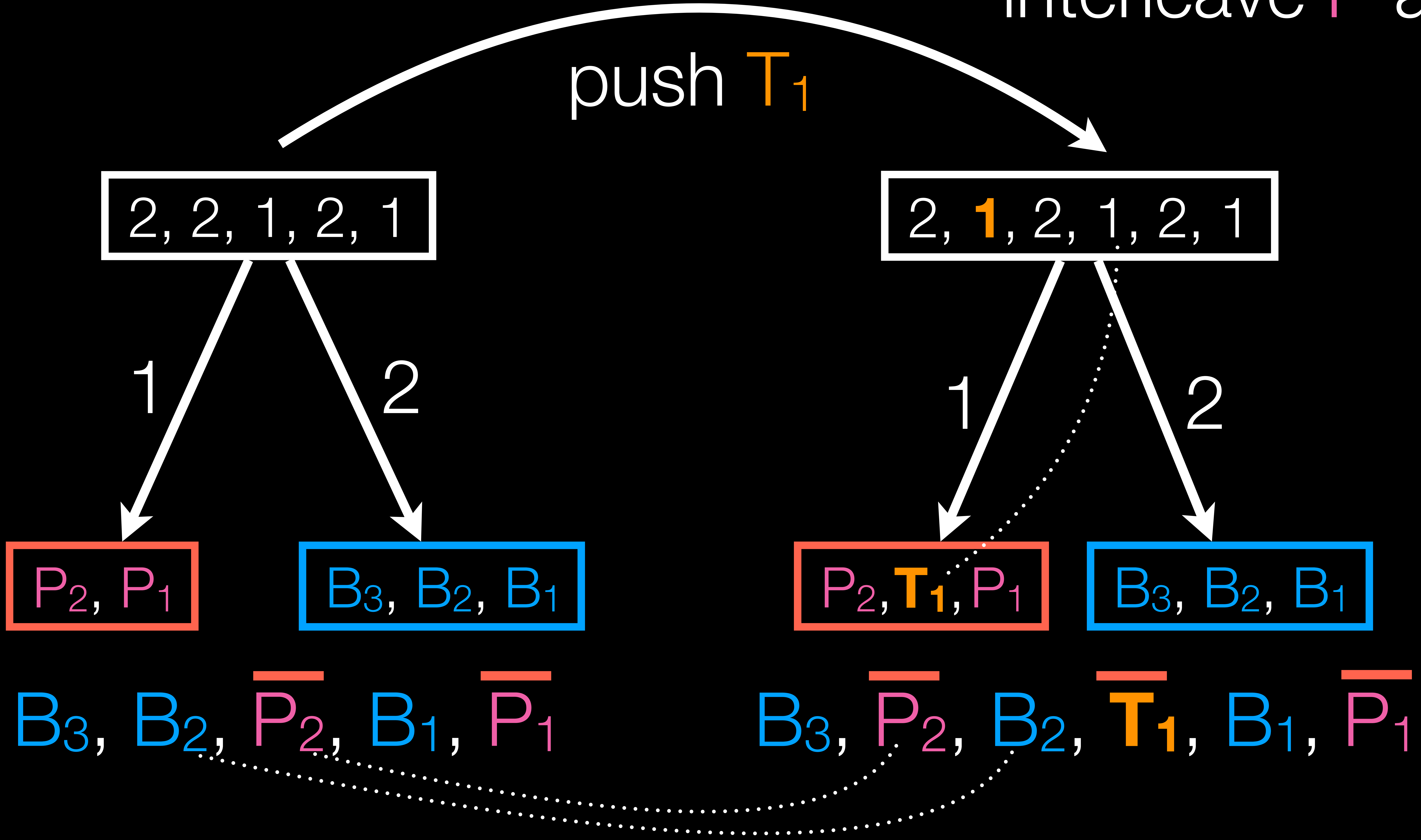
Aside: PIFO trees

interleave R and B;
interleave P and T.



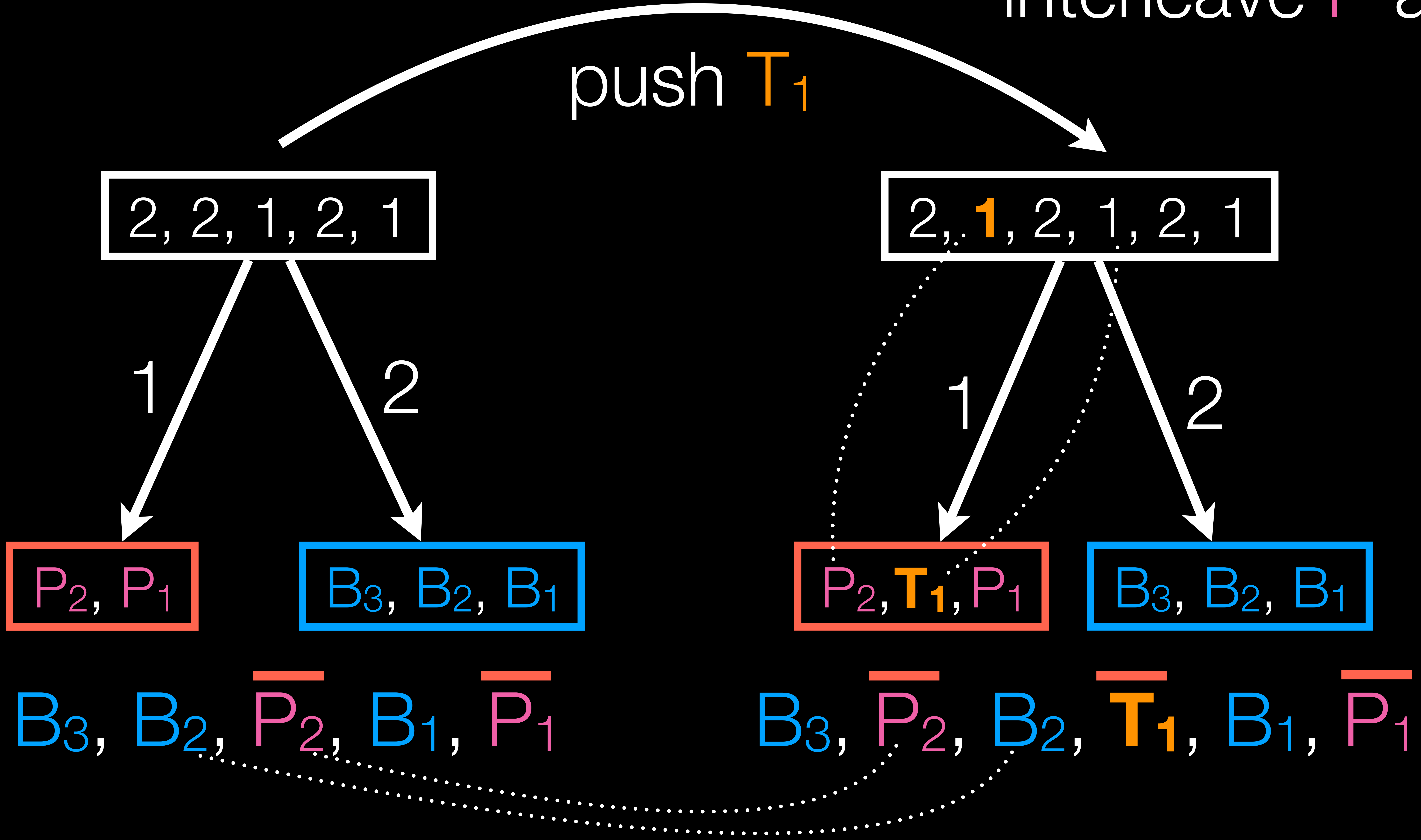
Aside: PIFO trees

interleave R and B;
interleave P and T.

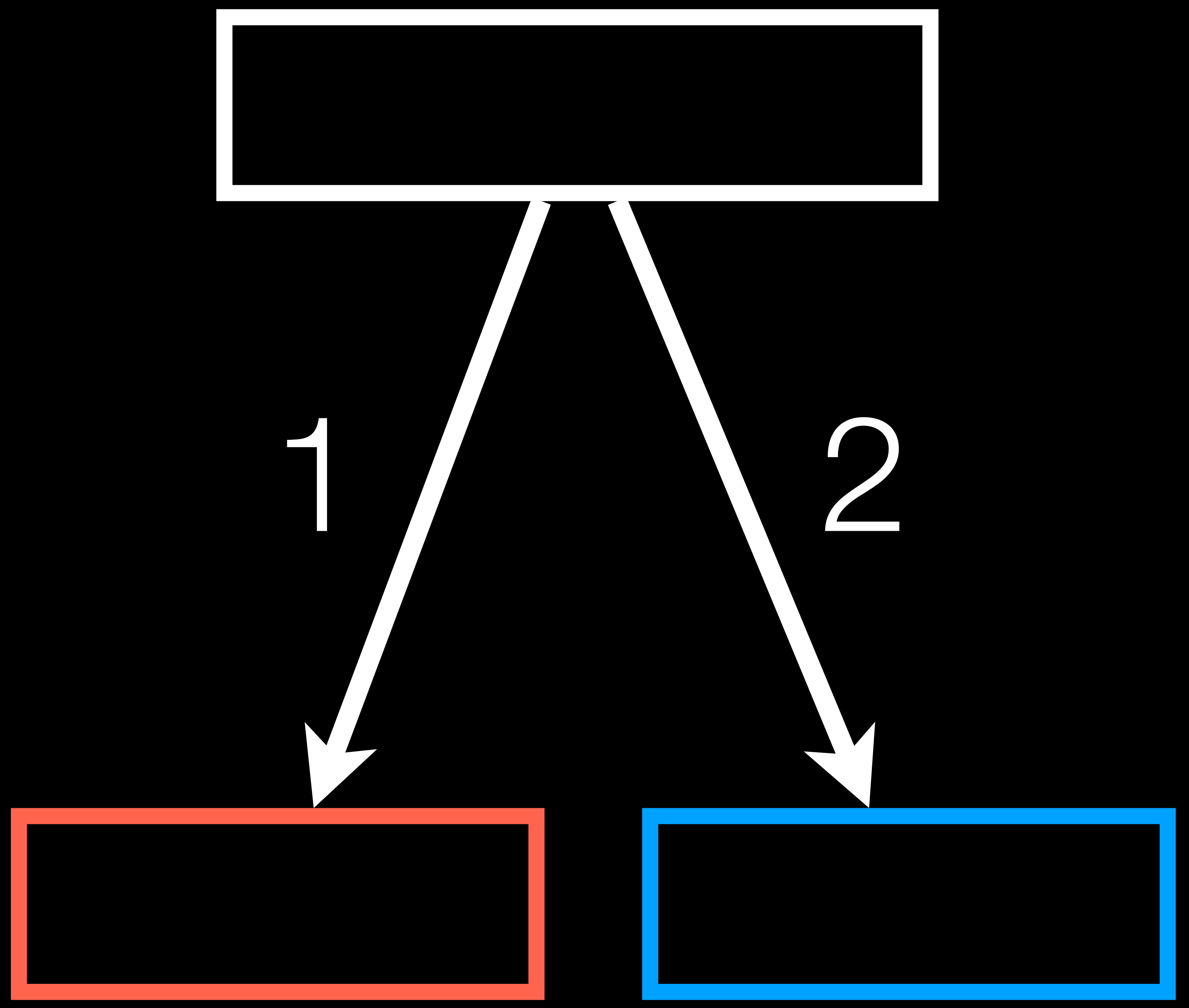


Aside: PIFO trees

interleave R and B;
interleave P and T.

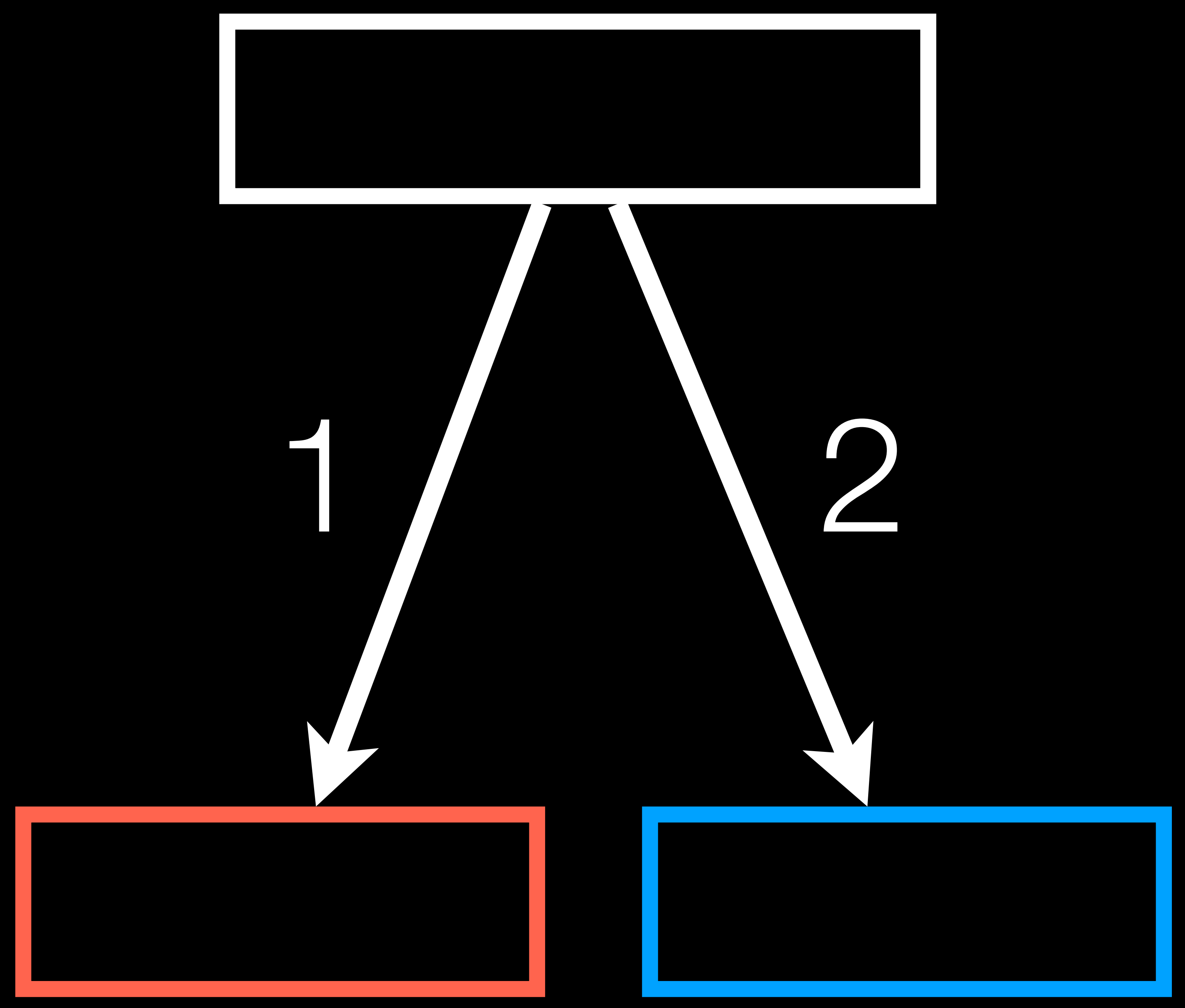


Key Insight



Key Insight

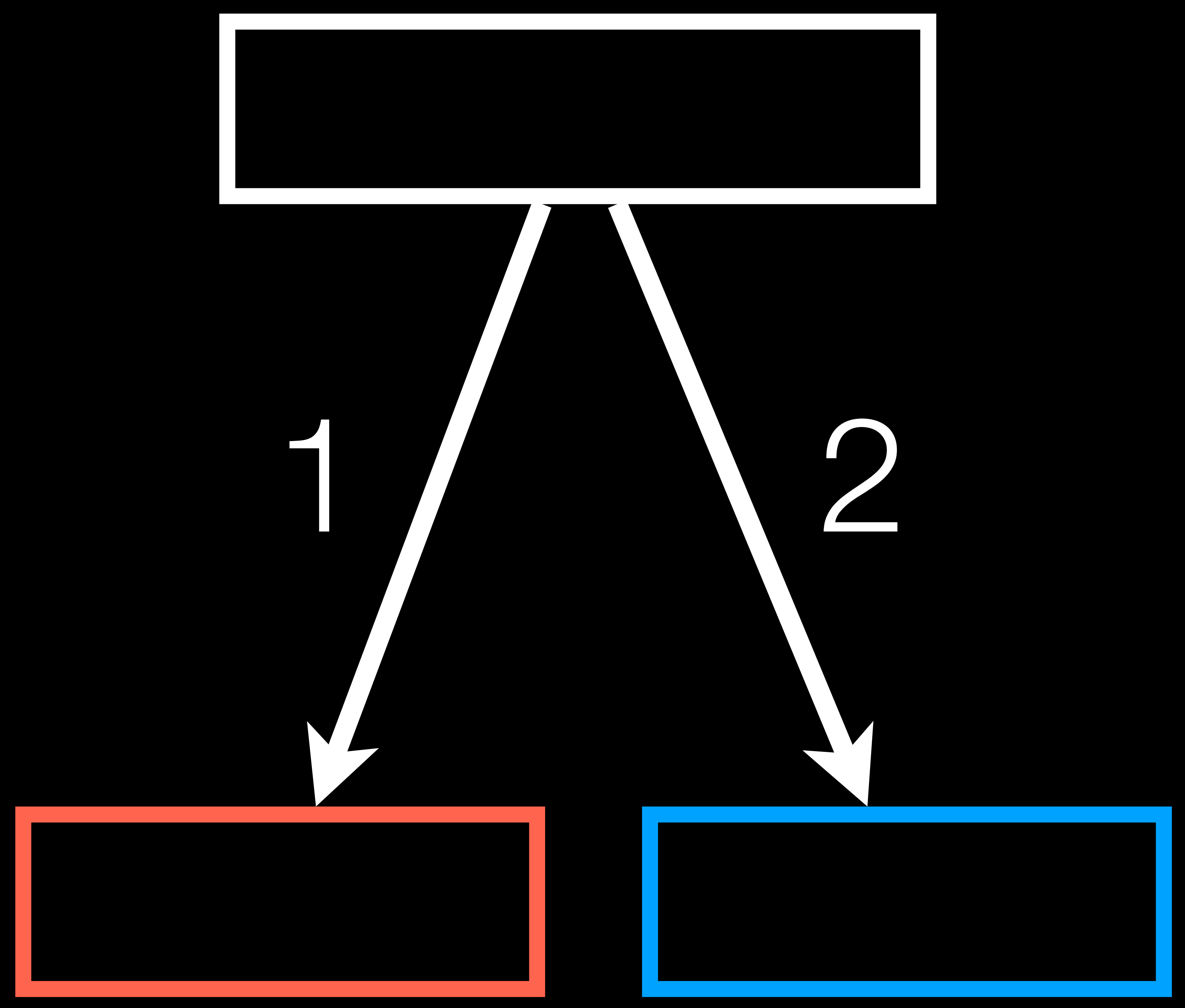
A PIFO tree manifests a *programming language.*



Key Insight

A PIFO tree manifests a *programming language.*

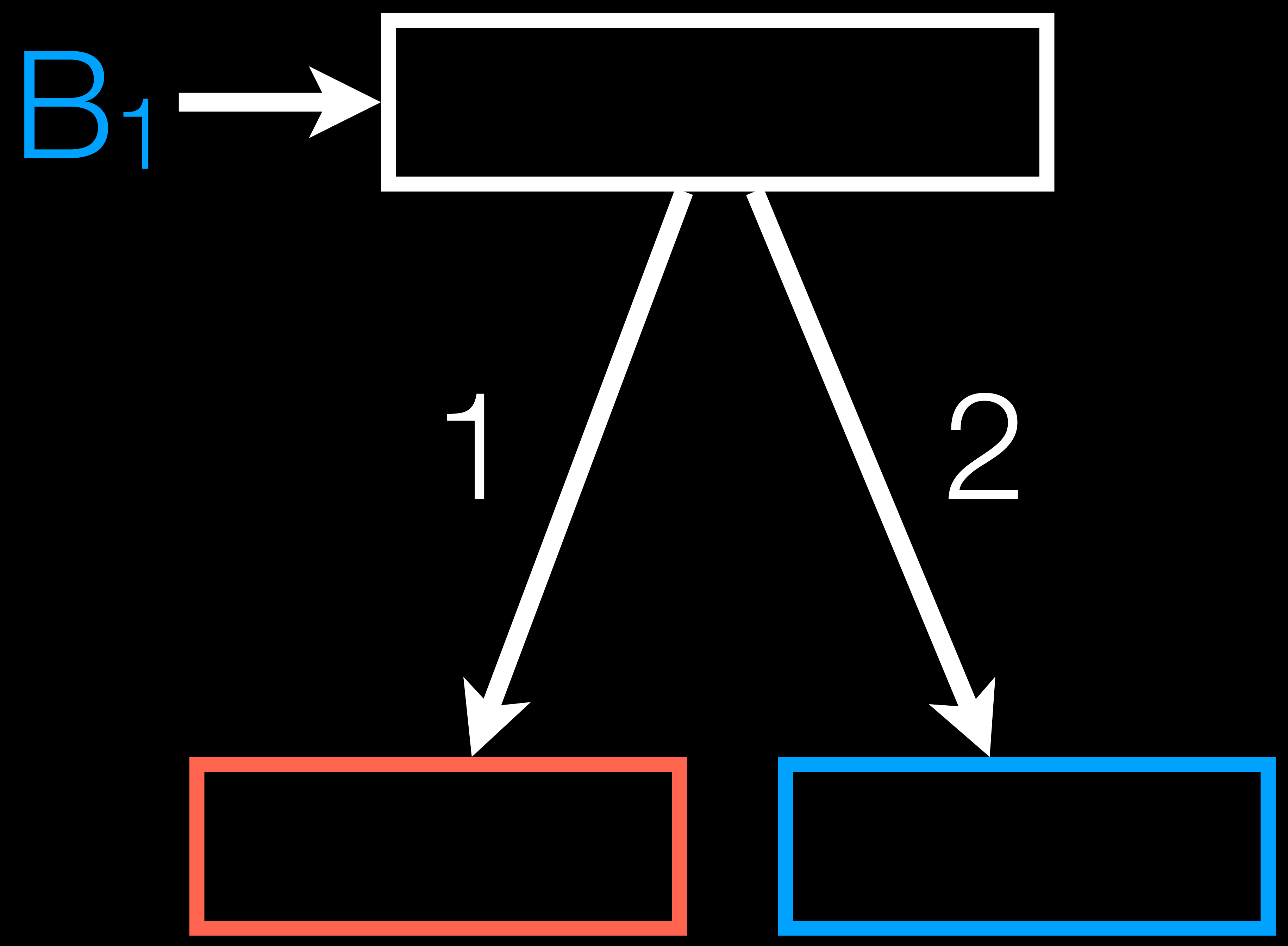
A program is precisely a *scheduling algorithm.*



Key Insight

A PIFO tree manifests a *programming language.*

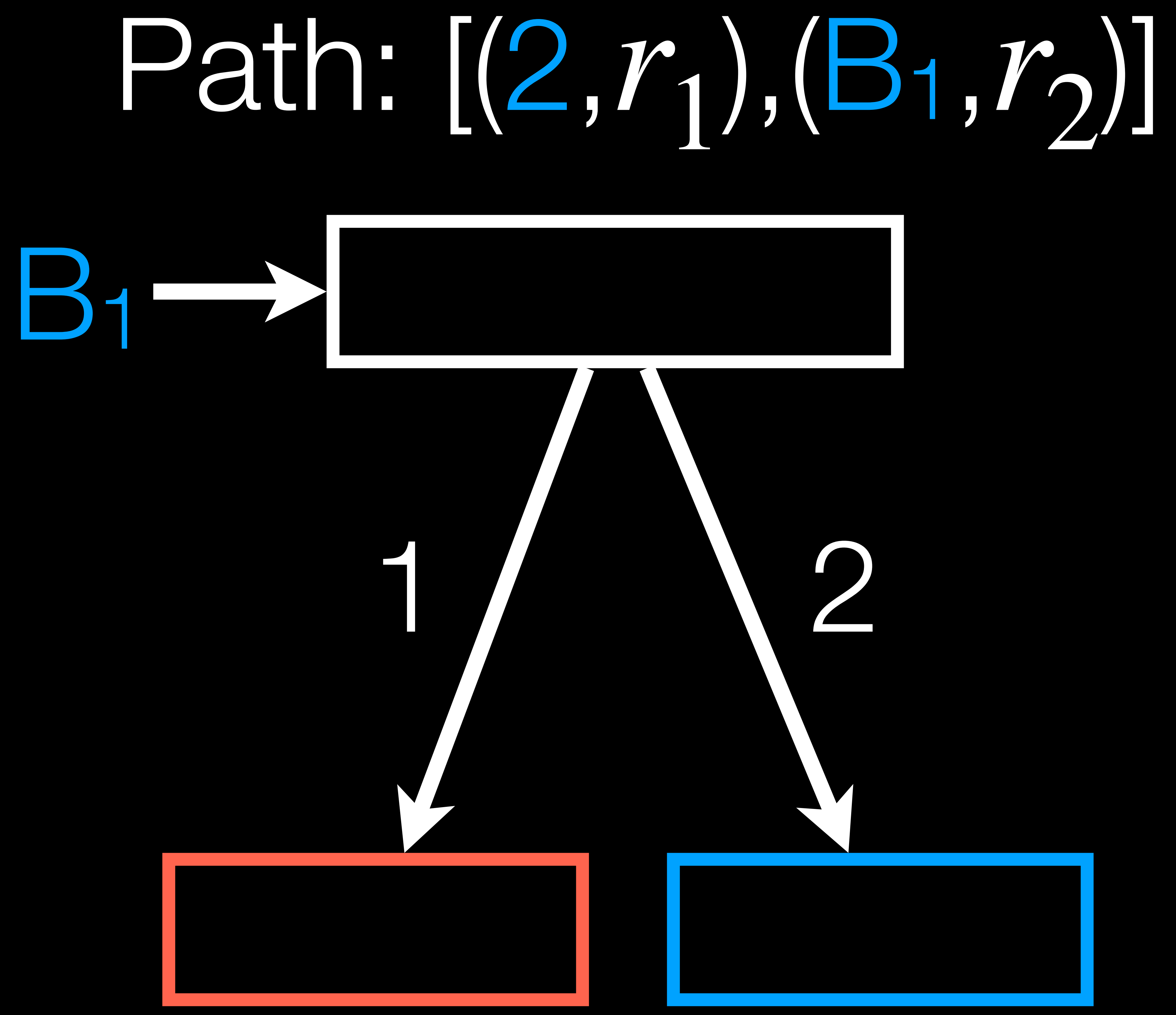
A program is precisely a *scheduling algorithm.*



Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

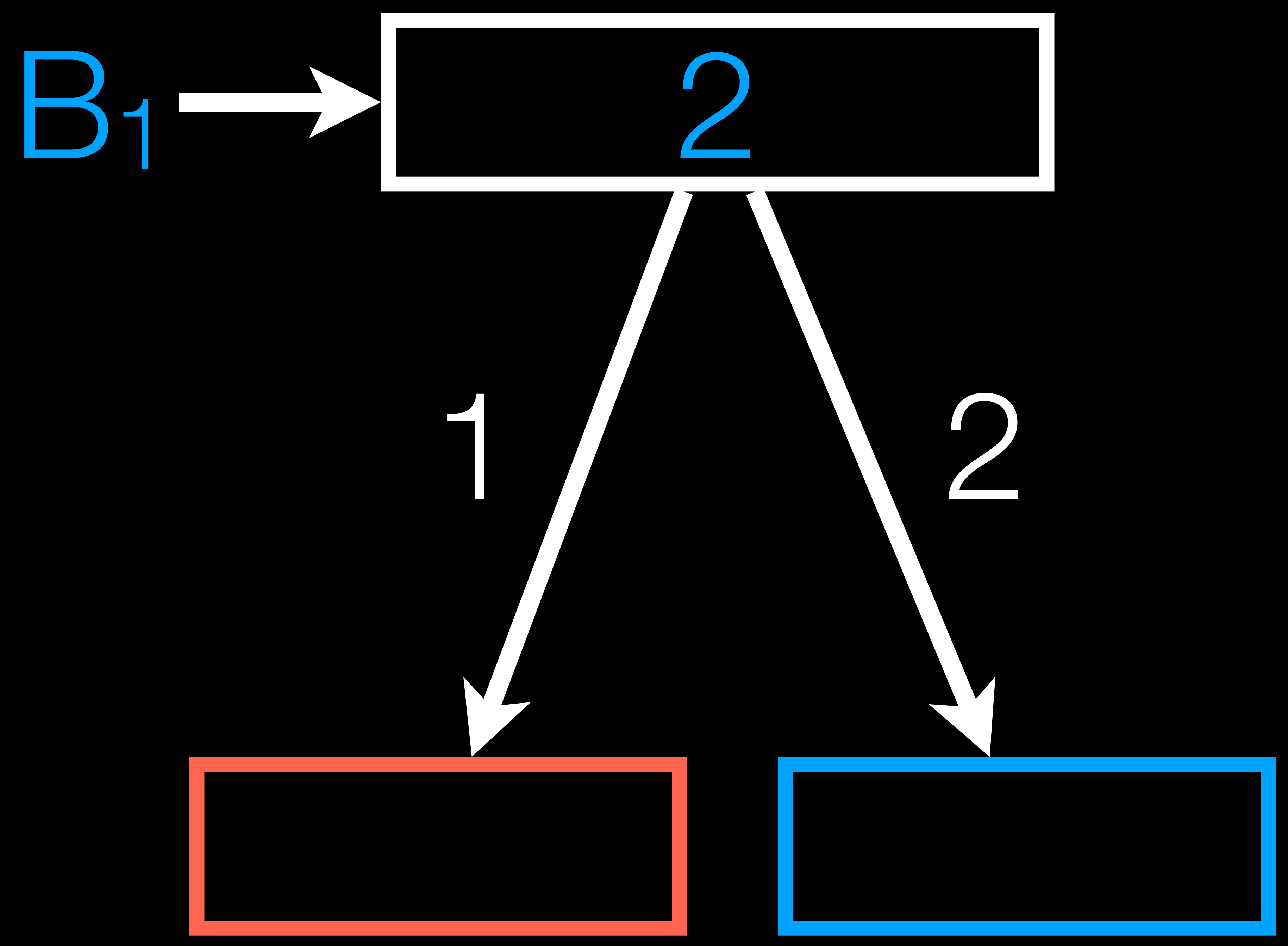


Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

Path: $[(2, r_1), (B_1, r_2)]$

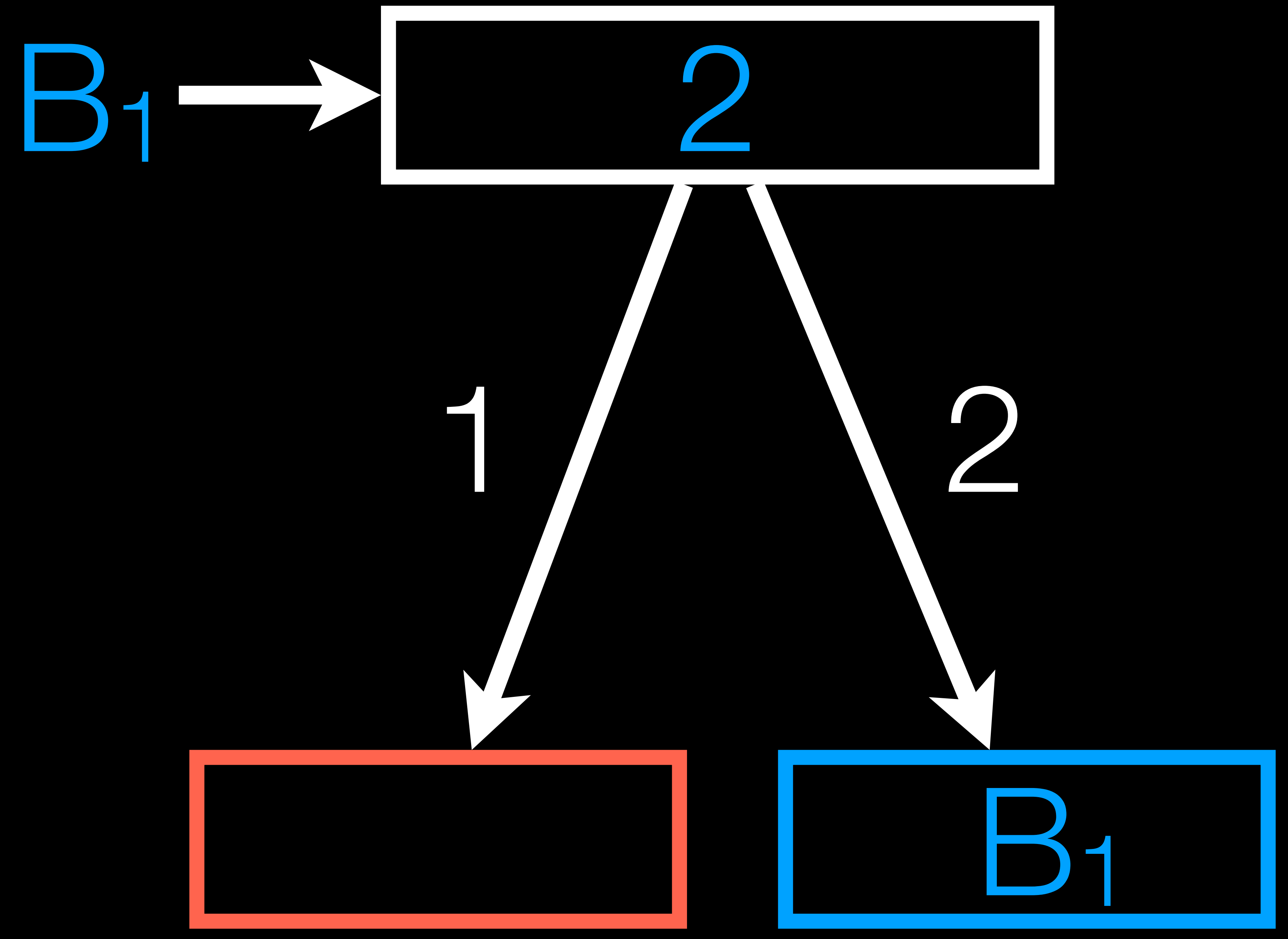


Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

Path: [(2, r_1),(B1, r_2)]

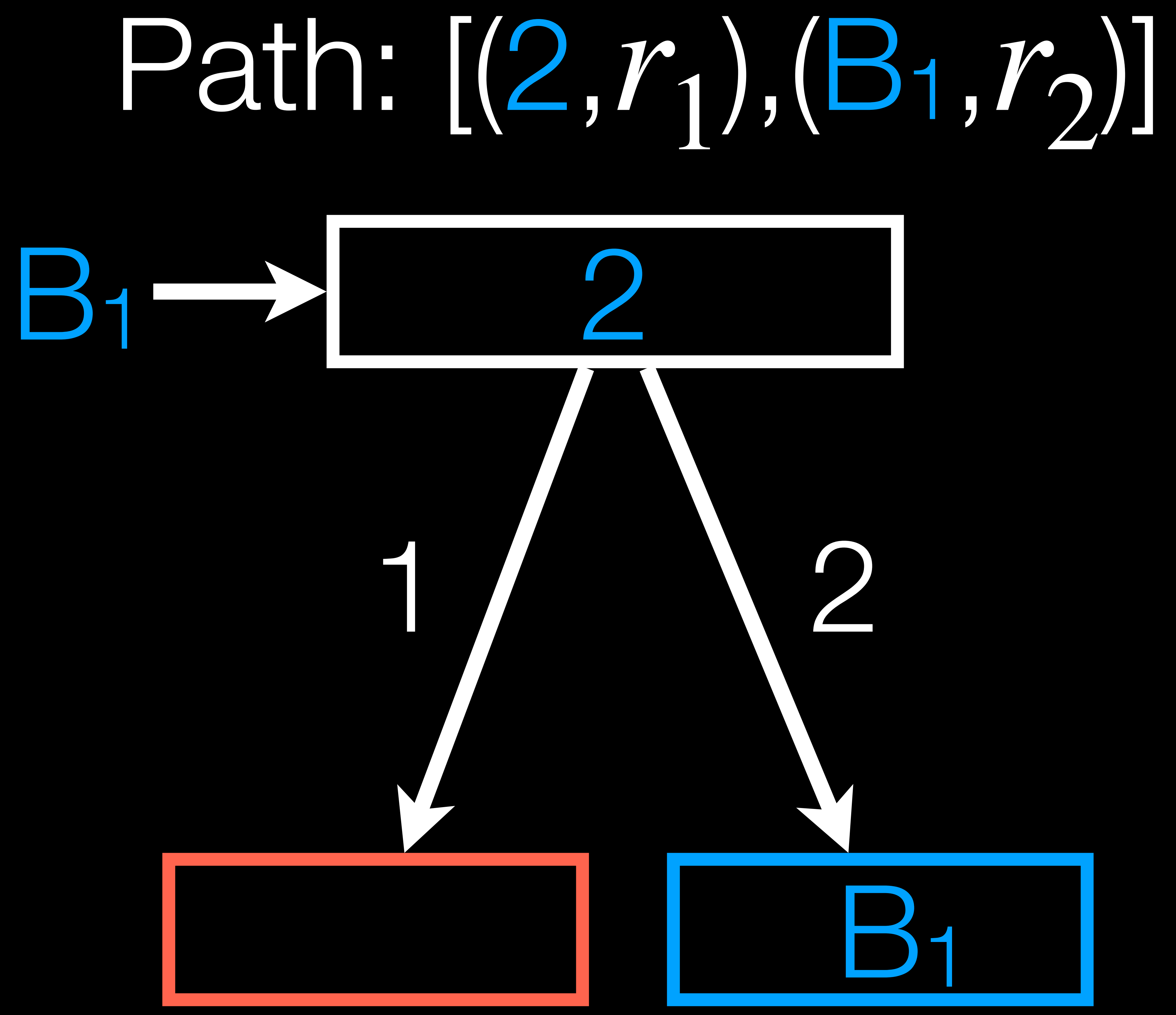


Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

tree shape ⟷ language expressivity



Which leads to some very PL-ey questions:

tree shape ⟷ language expressivity

Which leads to some very PL-ey questions:

tree shape ⟷ language expressivity

Compare expressivity of languages?

Which leads to some very PL-ey questions:

tree shape ⟷ language expressivity

Compare expressivity of languages?
Compare expressivity of *trees*?

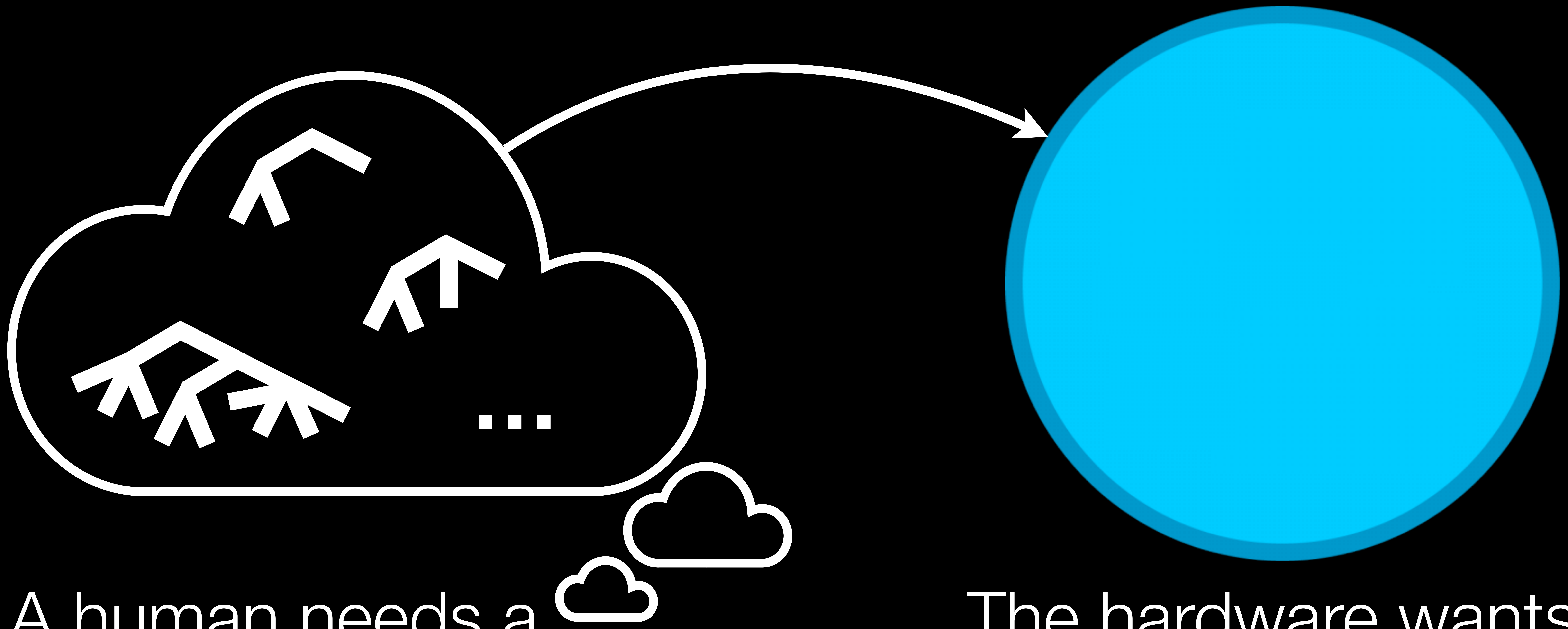
Which leads to some very PL-ey questions:

tree shape ⟷ language expressivity

Compare expressivity of languages?
Compare expressivity of *trees*?

Compile a program so it runs against a new tree?

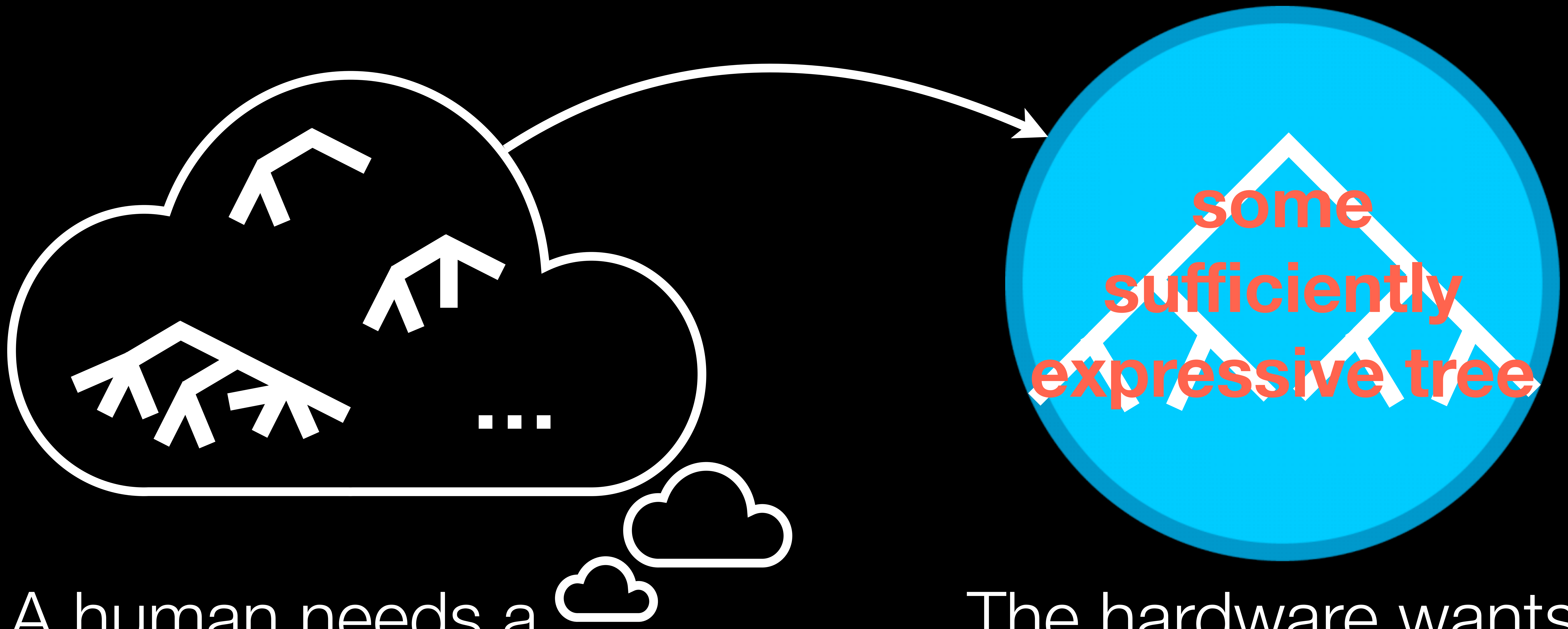
~~No general way to deploy our gadget.~~



A human needs a *range* of trees.

The hardware wants to support *one* tree.

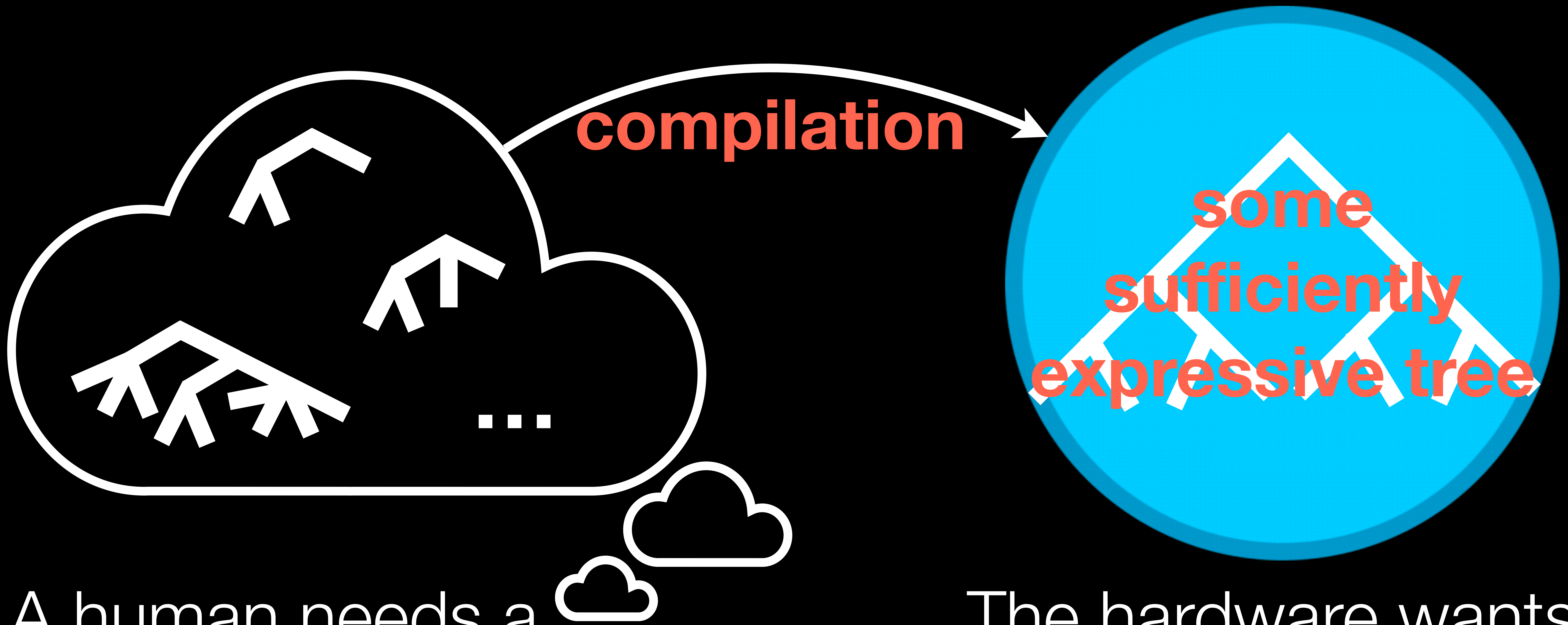
~~No general way to deploy our gadget.~~



A human needs a *range* of trees.

The hardware wants to support *one* tree.

~~No general way to deploy our gadget.~~



A human needs a *range* of trees.

The hardware wants to support *one* tree.

Contributions

Contributions

Formal model of PIFO trees

Contributions

Formal model of PIFO trees

General theorems of expressiveness w.r.t. tree shape

Contributions

Formal model of PIFO trees

General theorems of expressiveness w.r.t. tree shape

Compiler

Contributions

Formal model of PIFO trees

General theorems of expressiveness w.r.t. tree shape

Compiler

Simulator

Expressivity of trees

Trees with more leaves are more expressive.

Taller trees are more expressive.

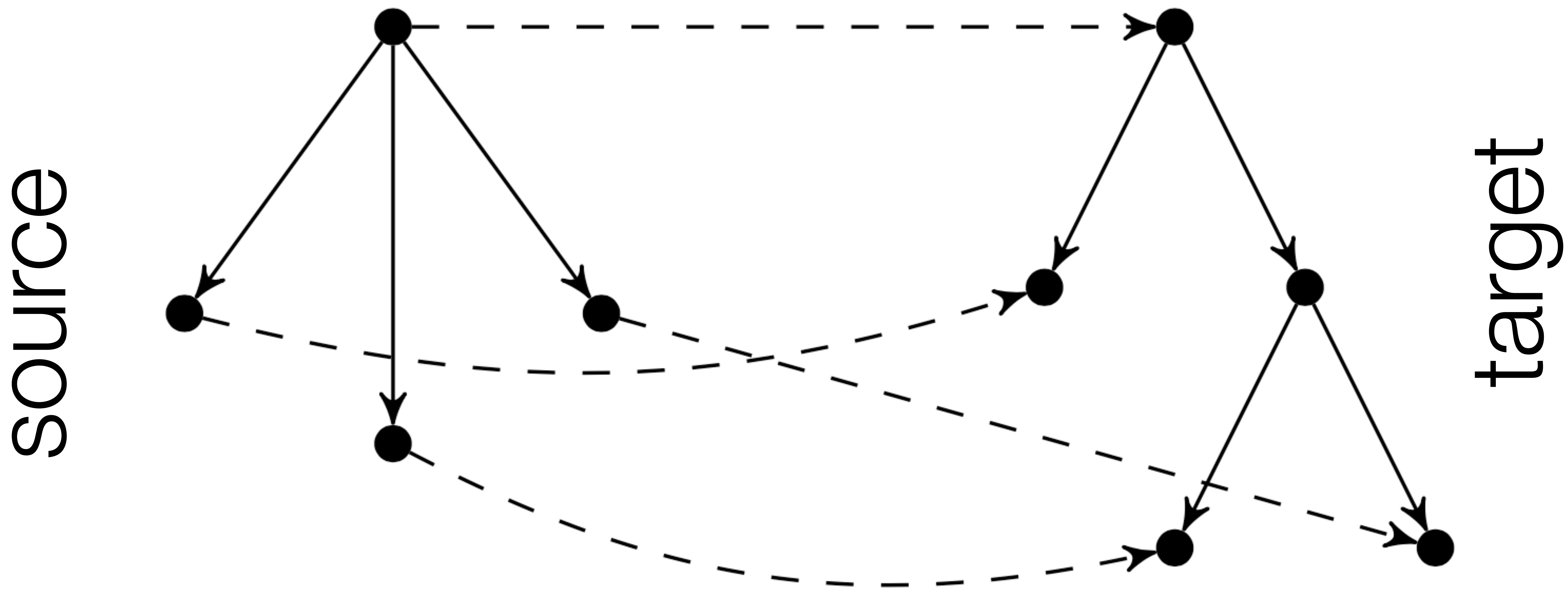
Expressivity of trees

Trees with more leaves are more expressive.
Taller trees are more expressive.

Captured elegantly by:

Homomorphic embedding.
Map root to root, leaves to leaves. Respect ancestry.

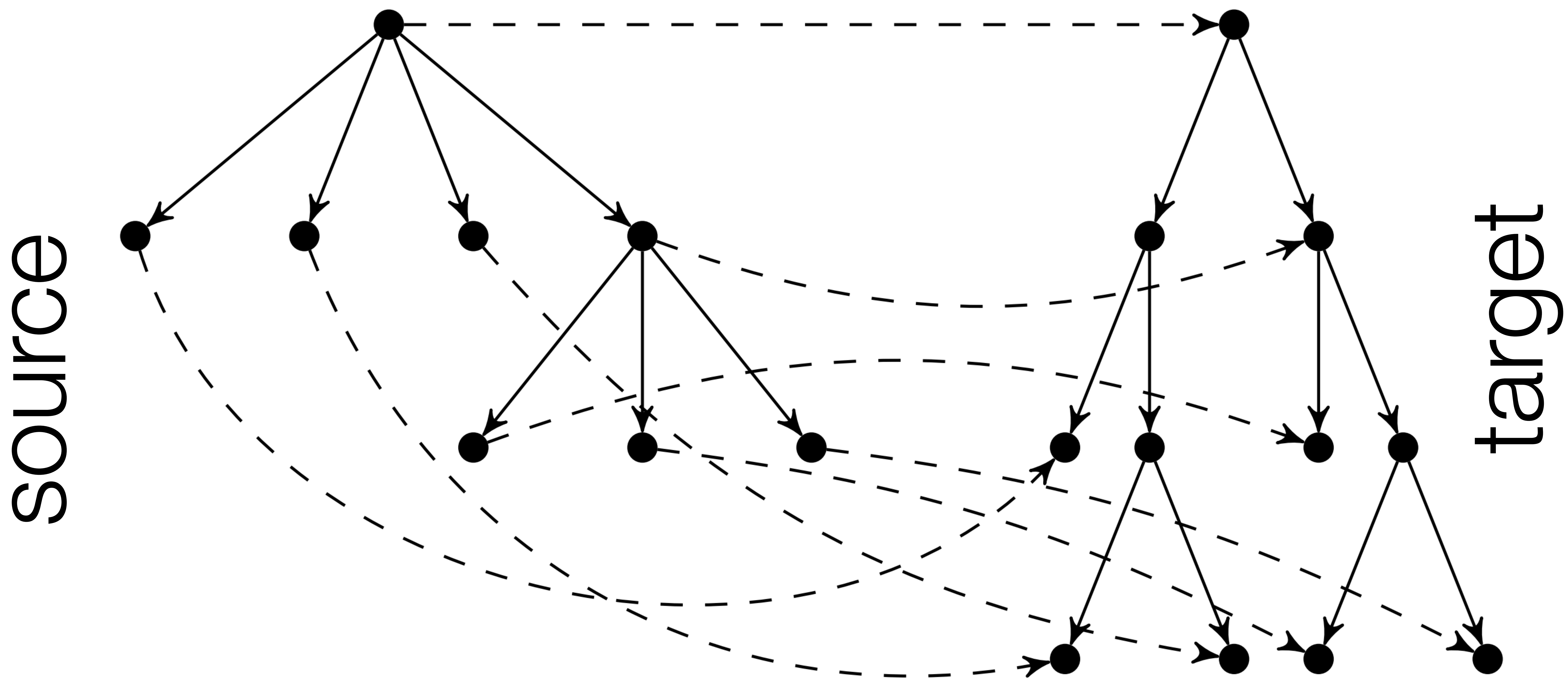
Expressivity of trees



Homomorphic embedding.

Map root to root, leaves to leaves. Respect ancestry.

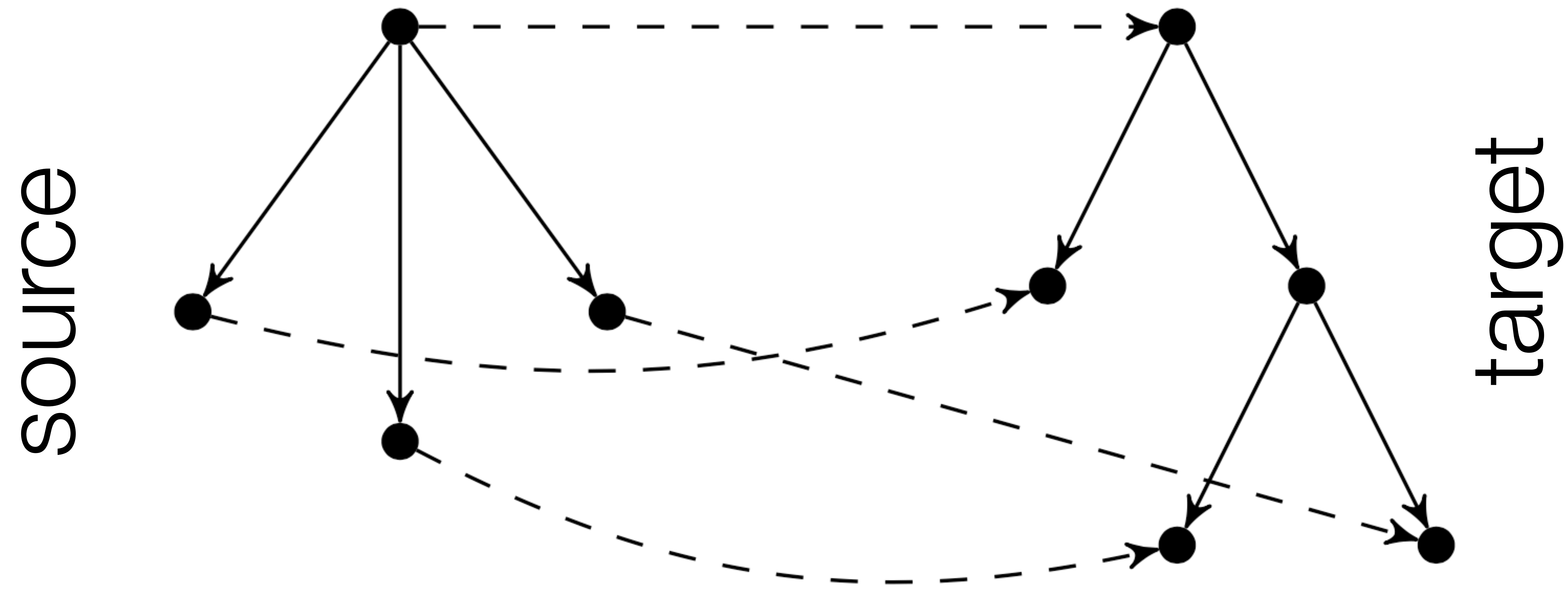
Expressivity of trees



Homomorphic embedding.

Map root to root, leaves to leaves. Respect ancestry.

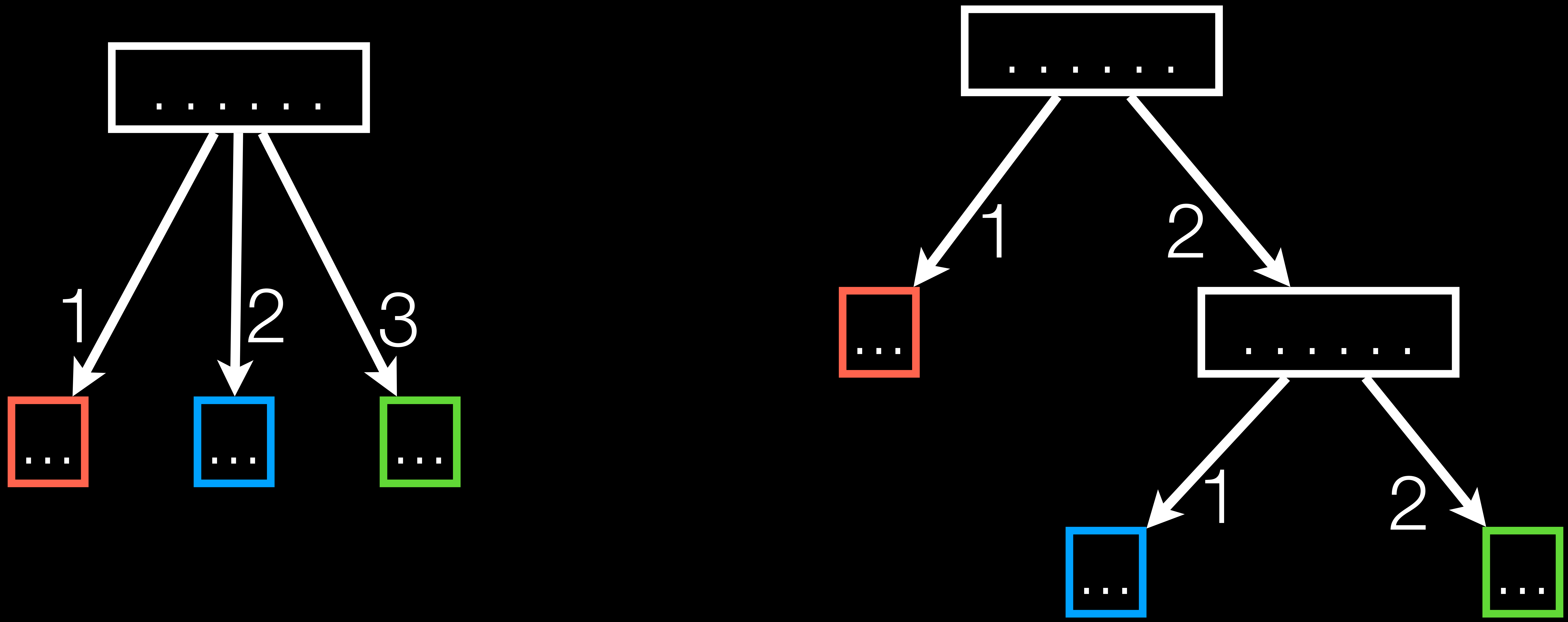
Expressivity of trees



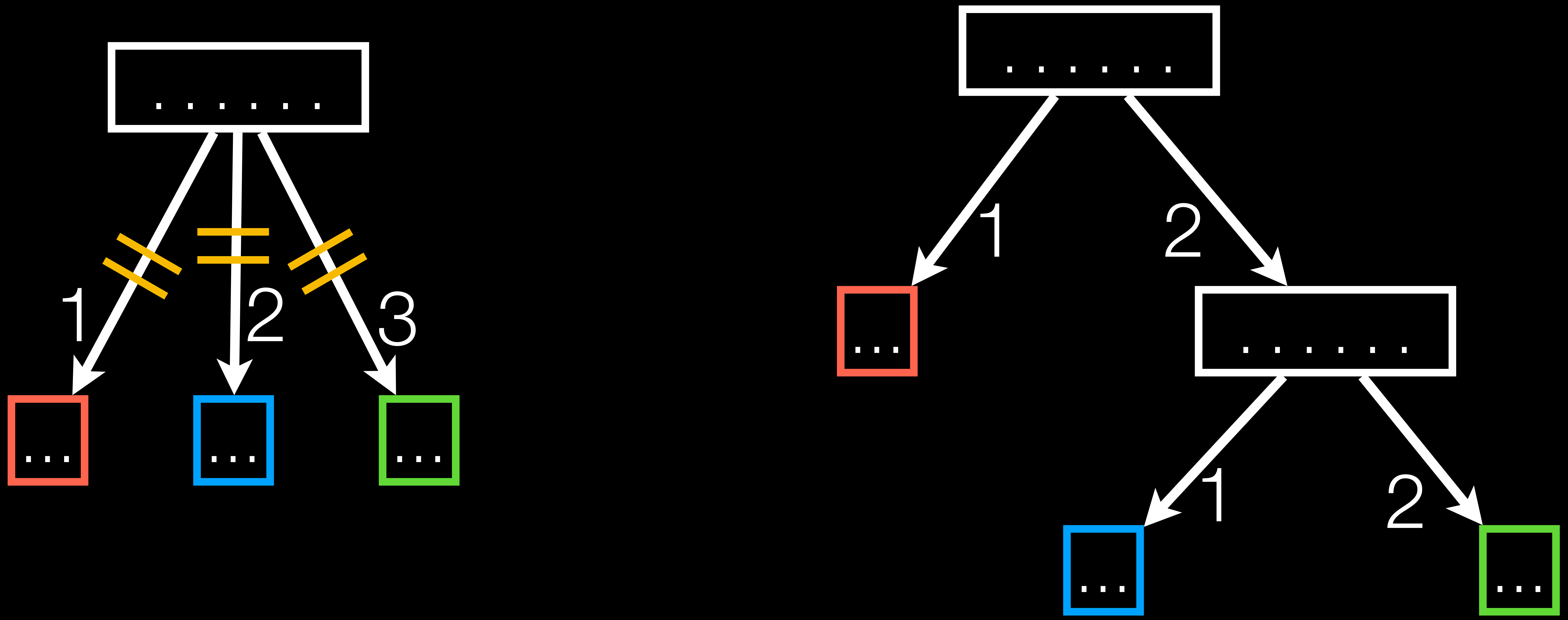
Homomorphic embedding.

Map root to root, leaves to leaves. Respect ancestry.

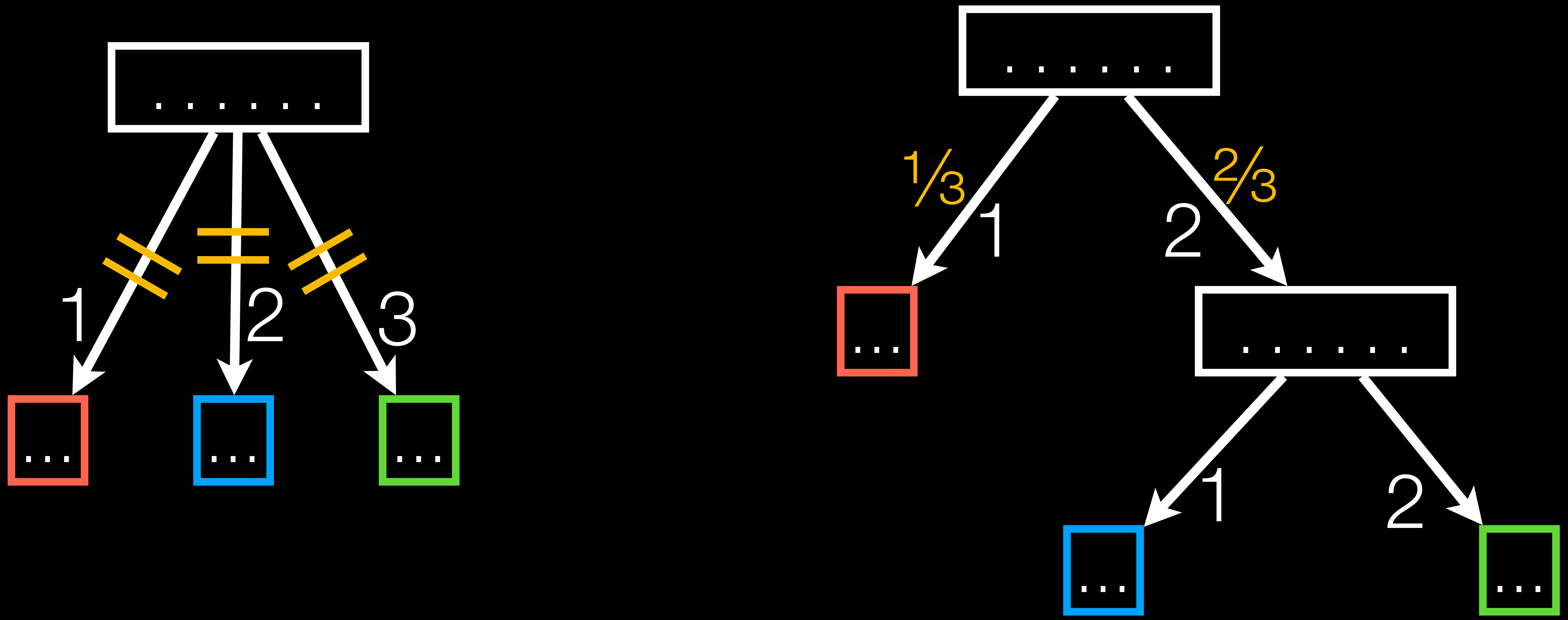
Compiling programs



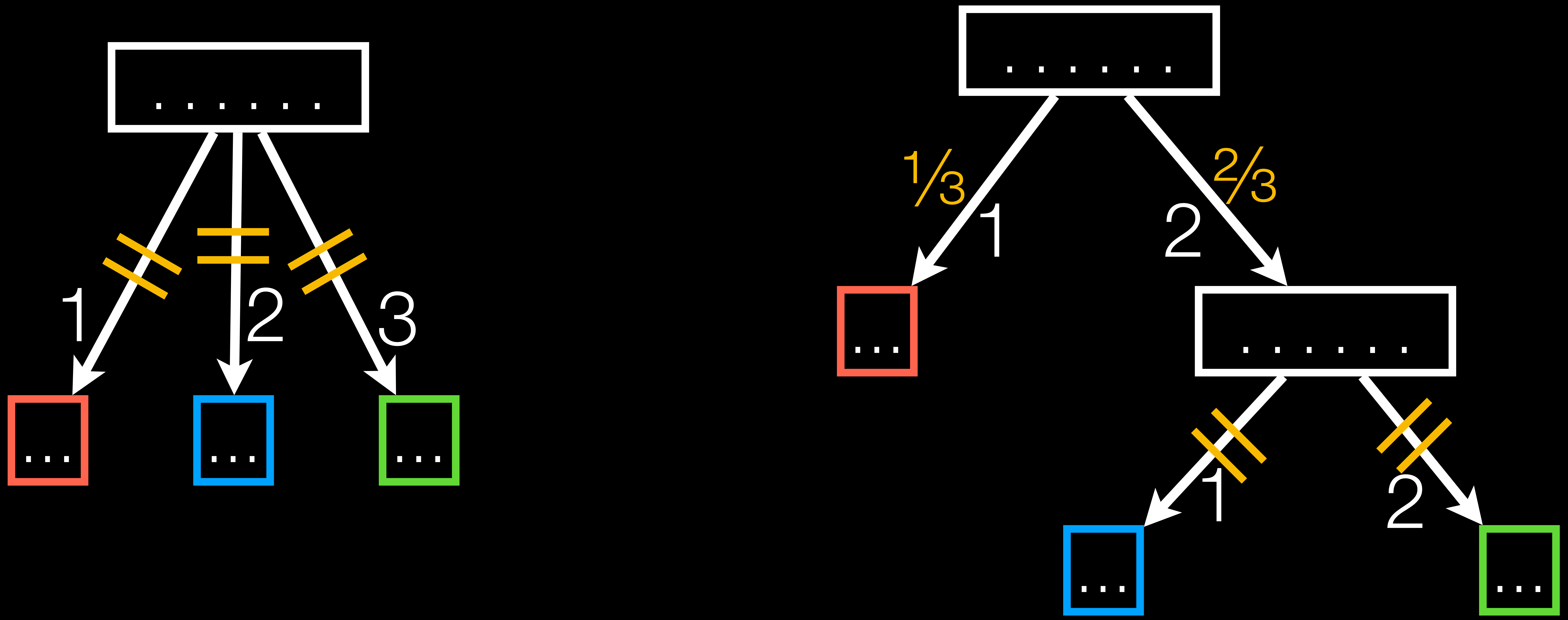
Compiling programs



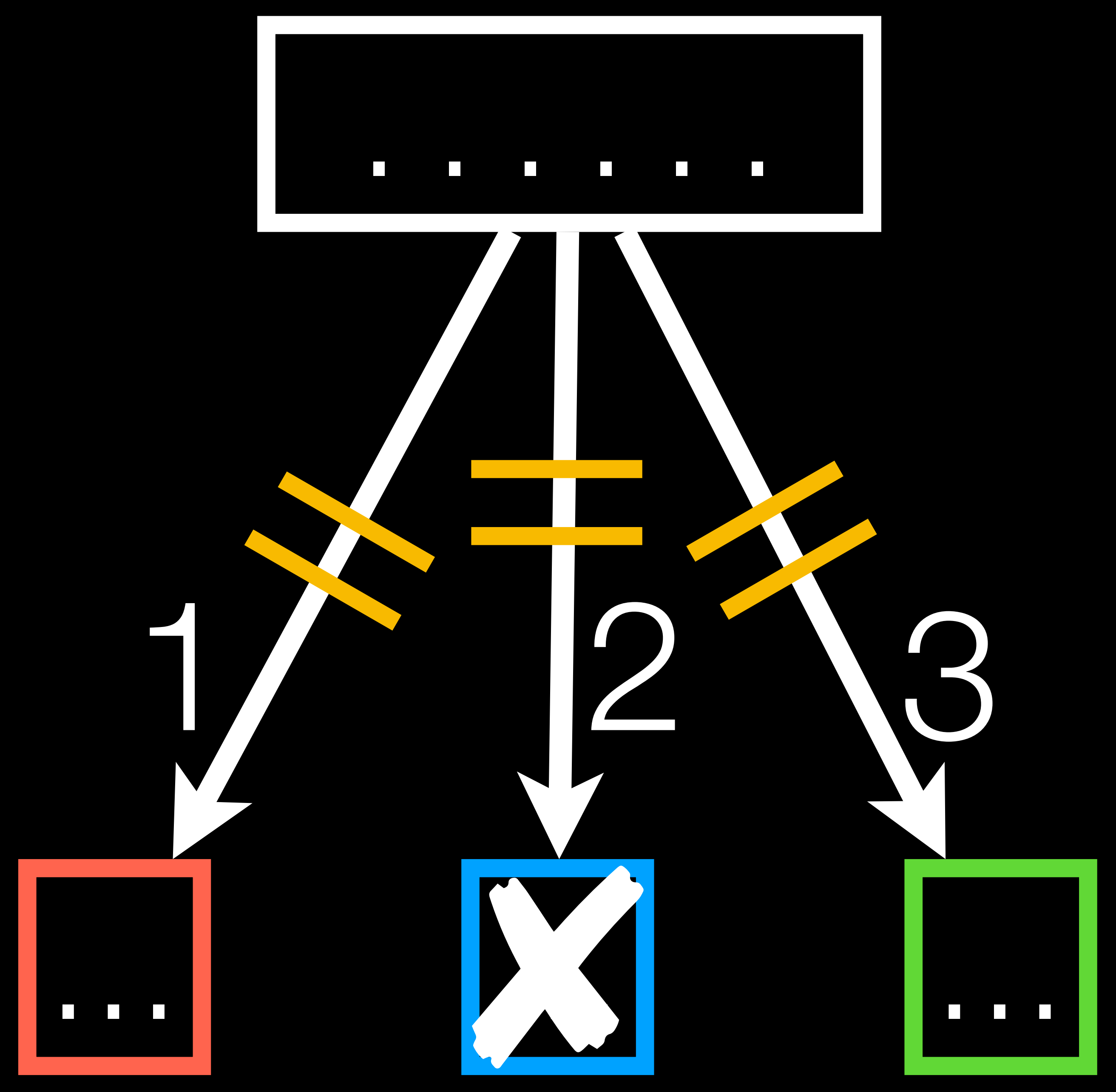
Compiling programs

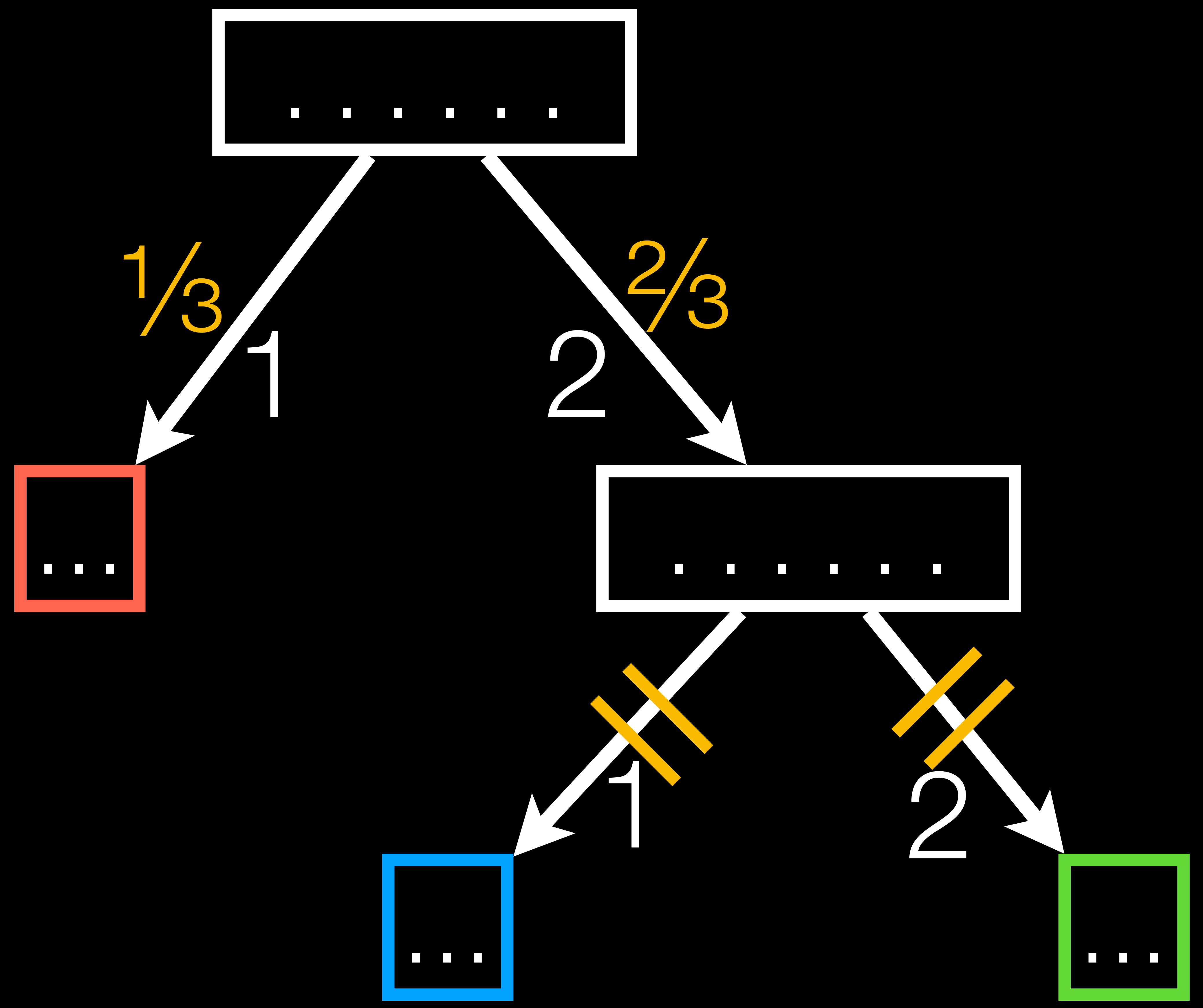


Compiling programs

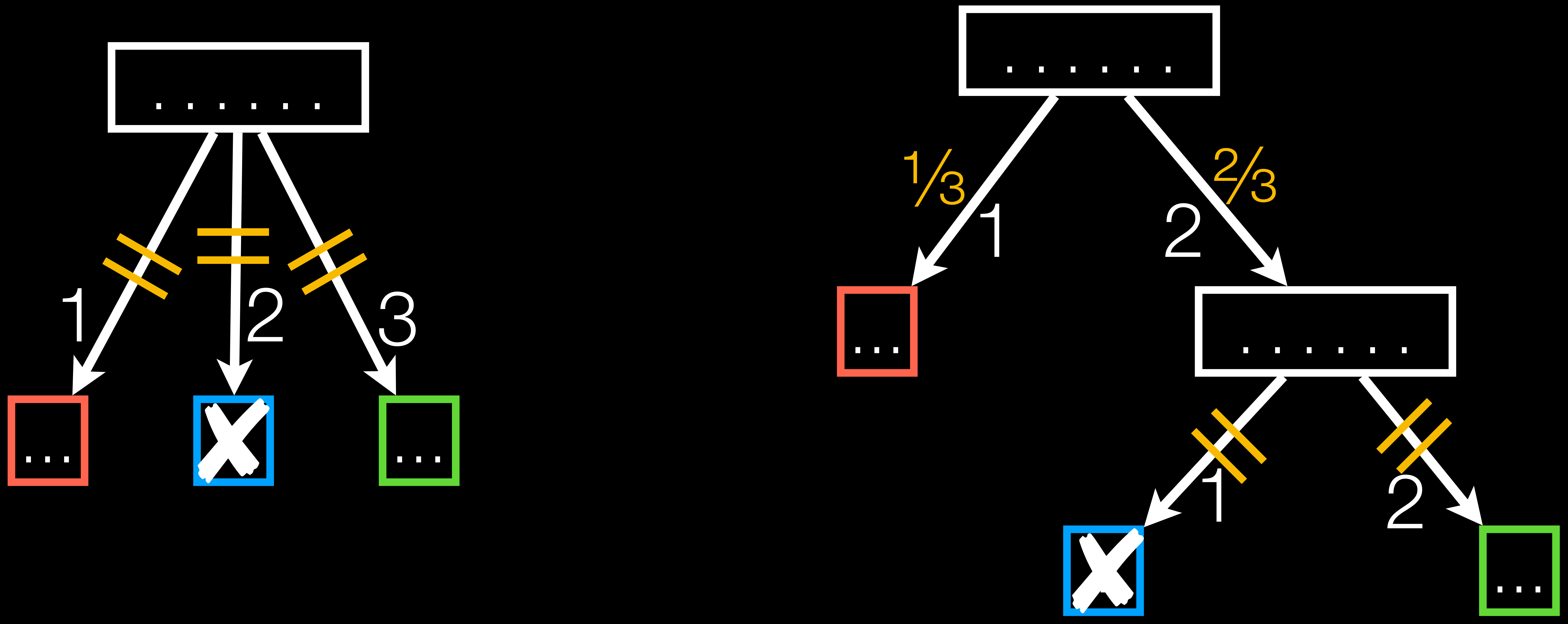


Compiling programs

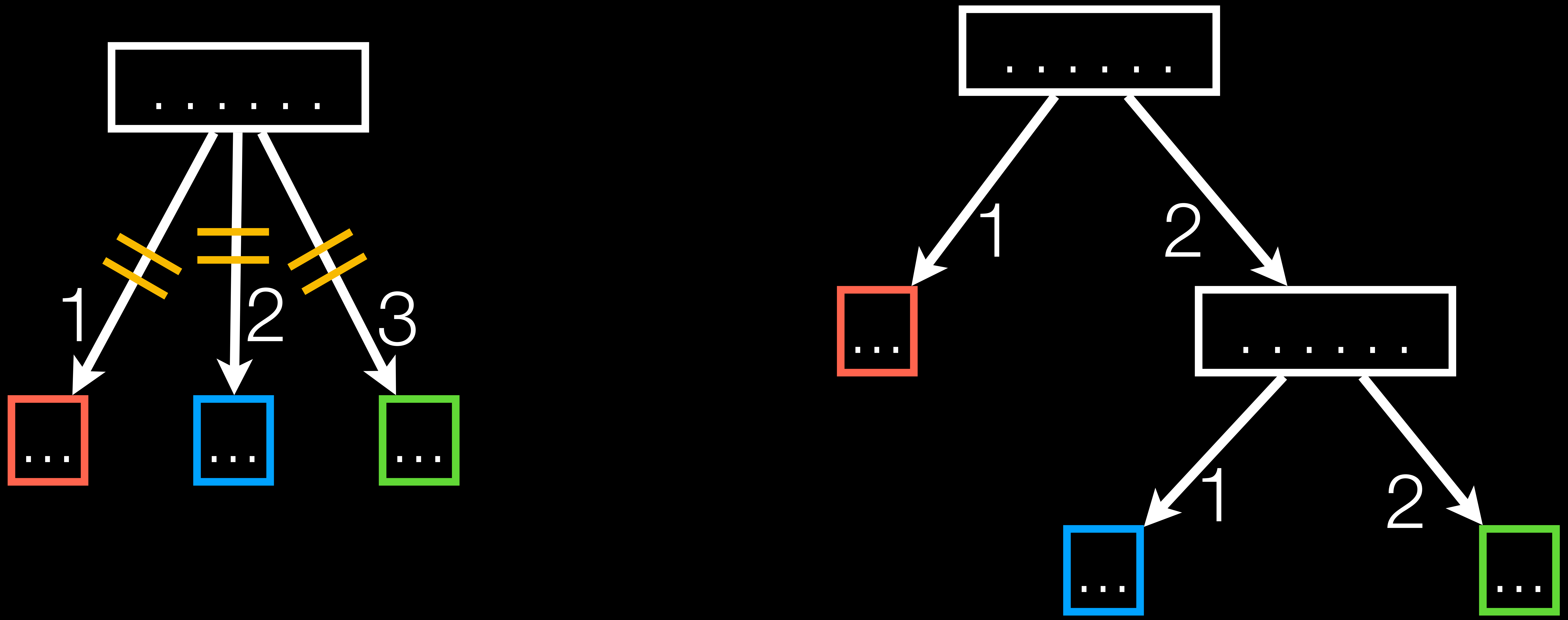




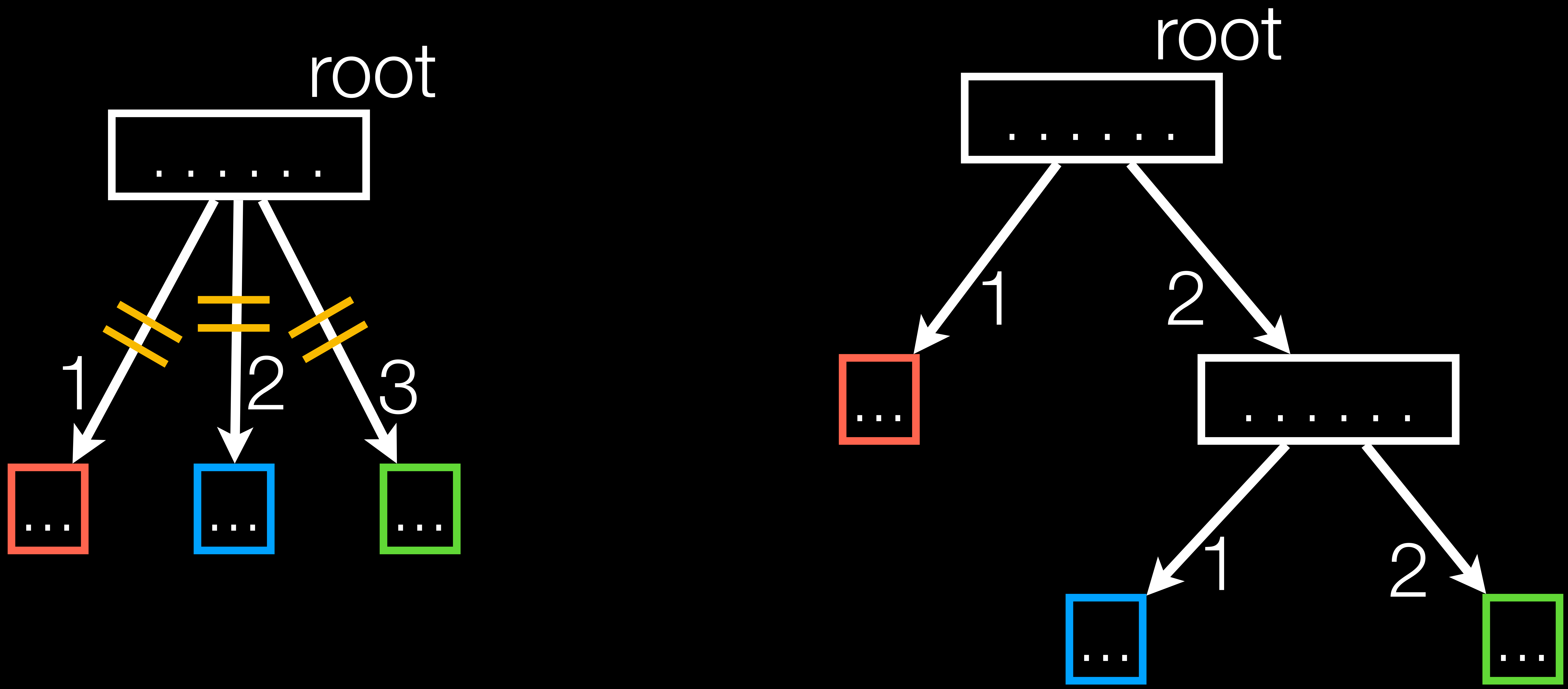
Compiling programs



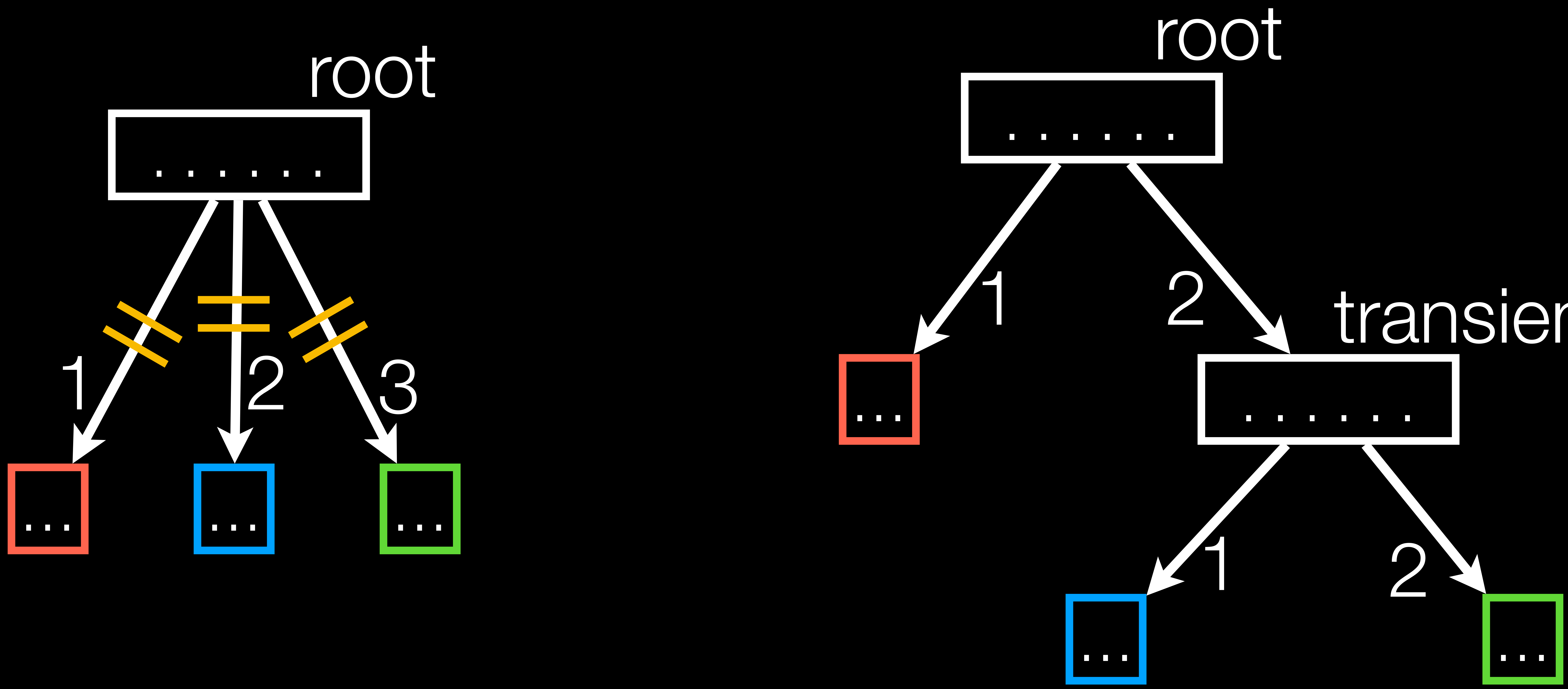
Compiling programs



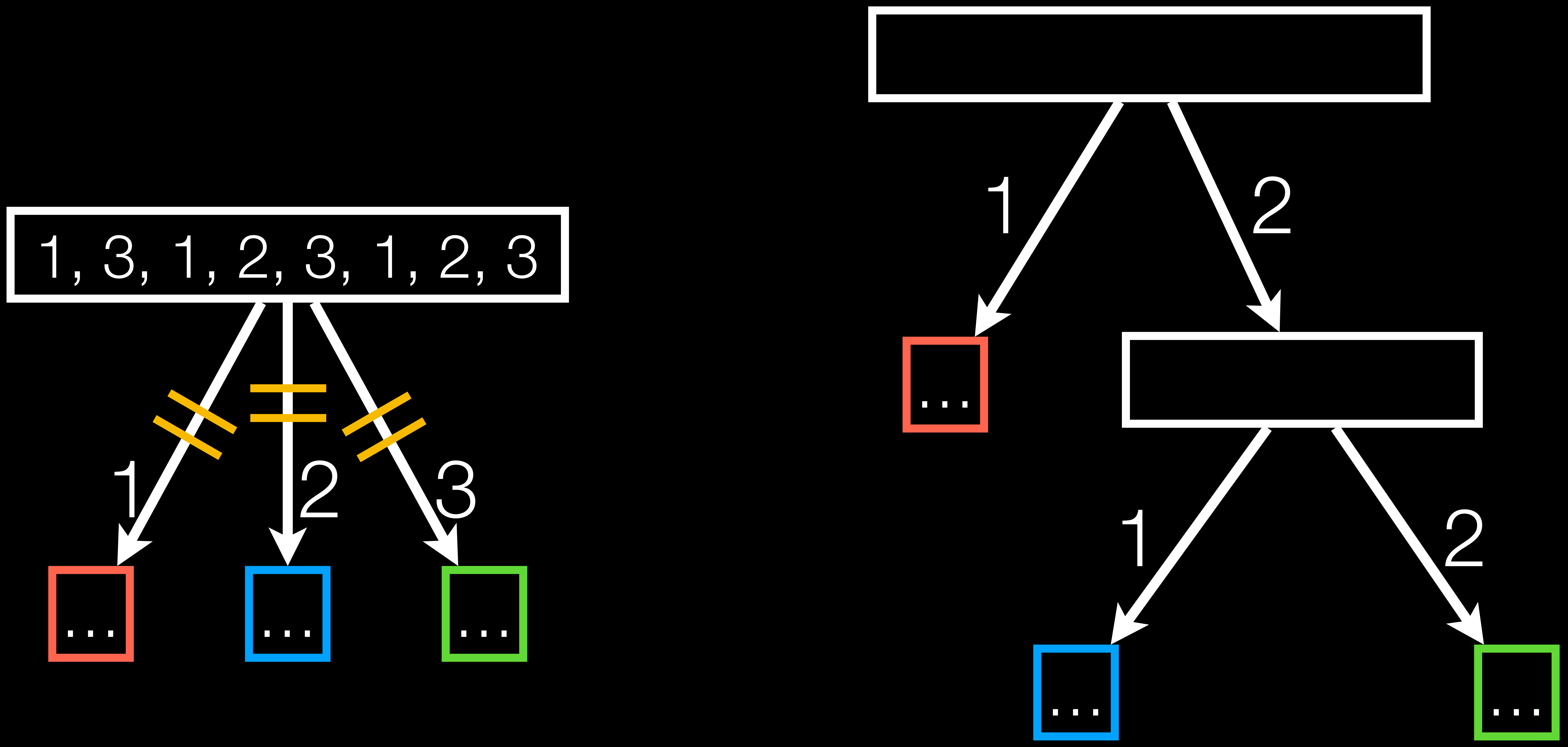
Compiling programs



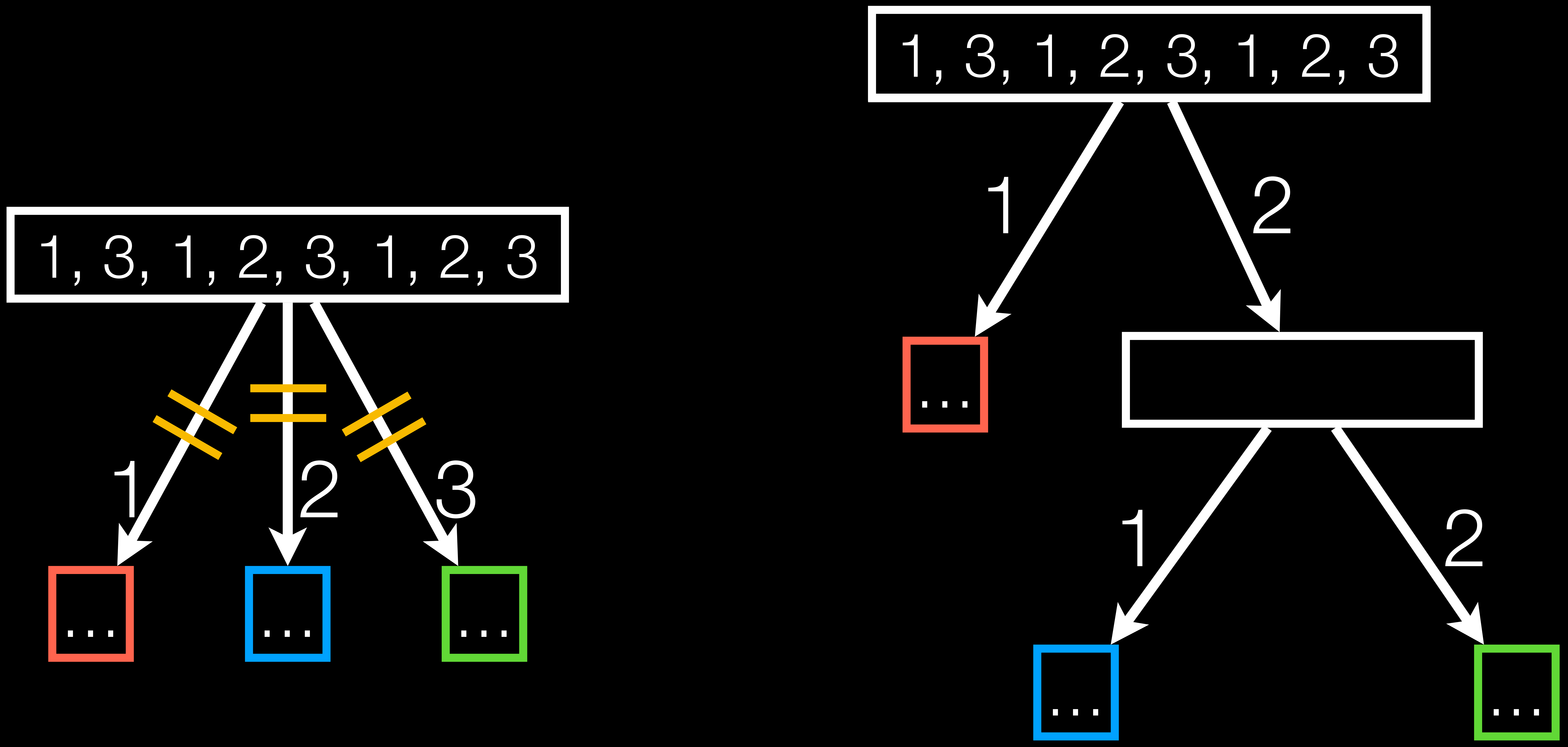
Compiling programs



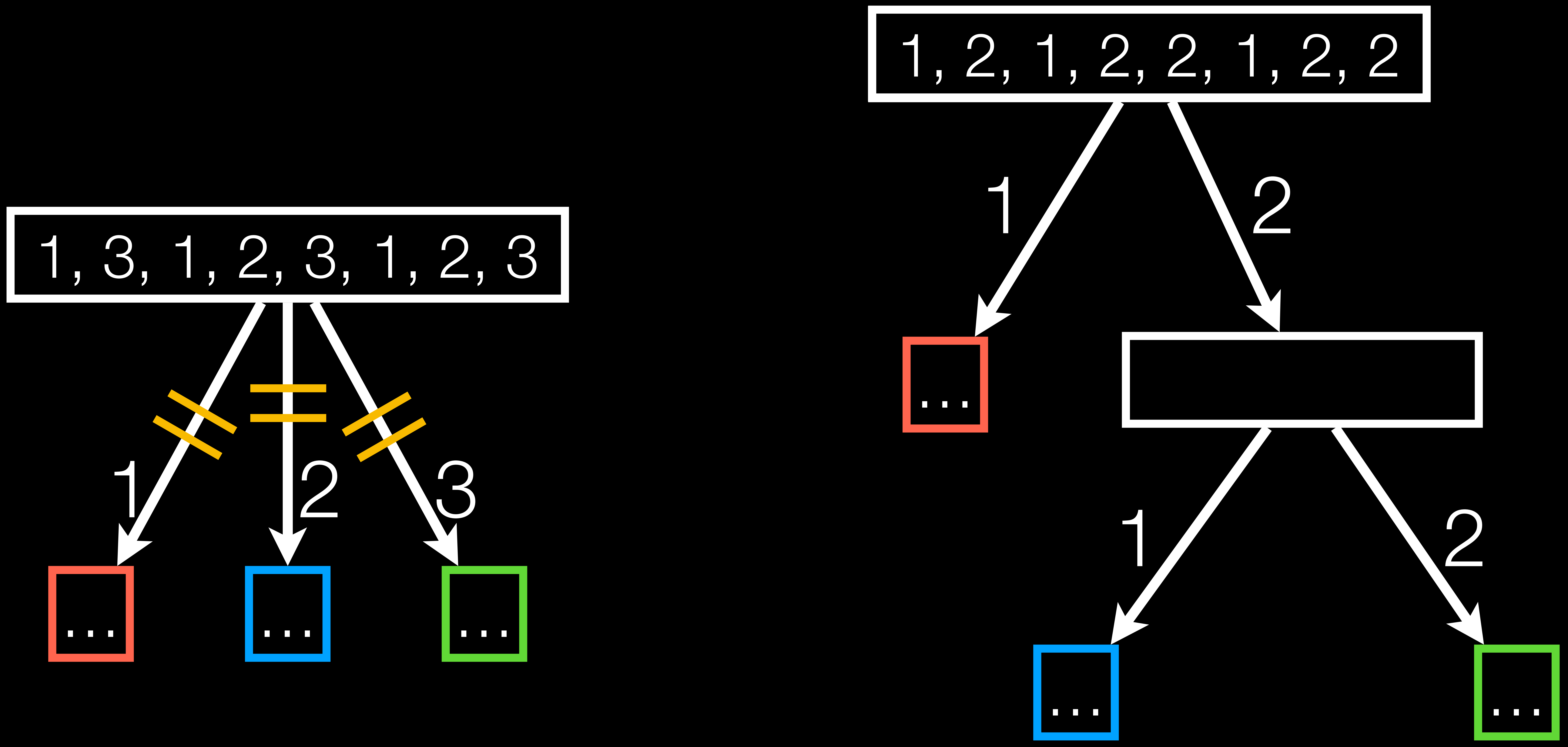
Compiling programs



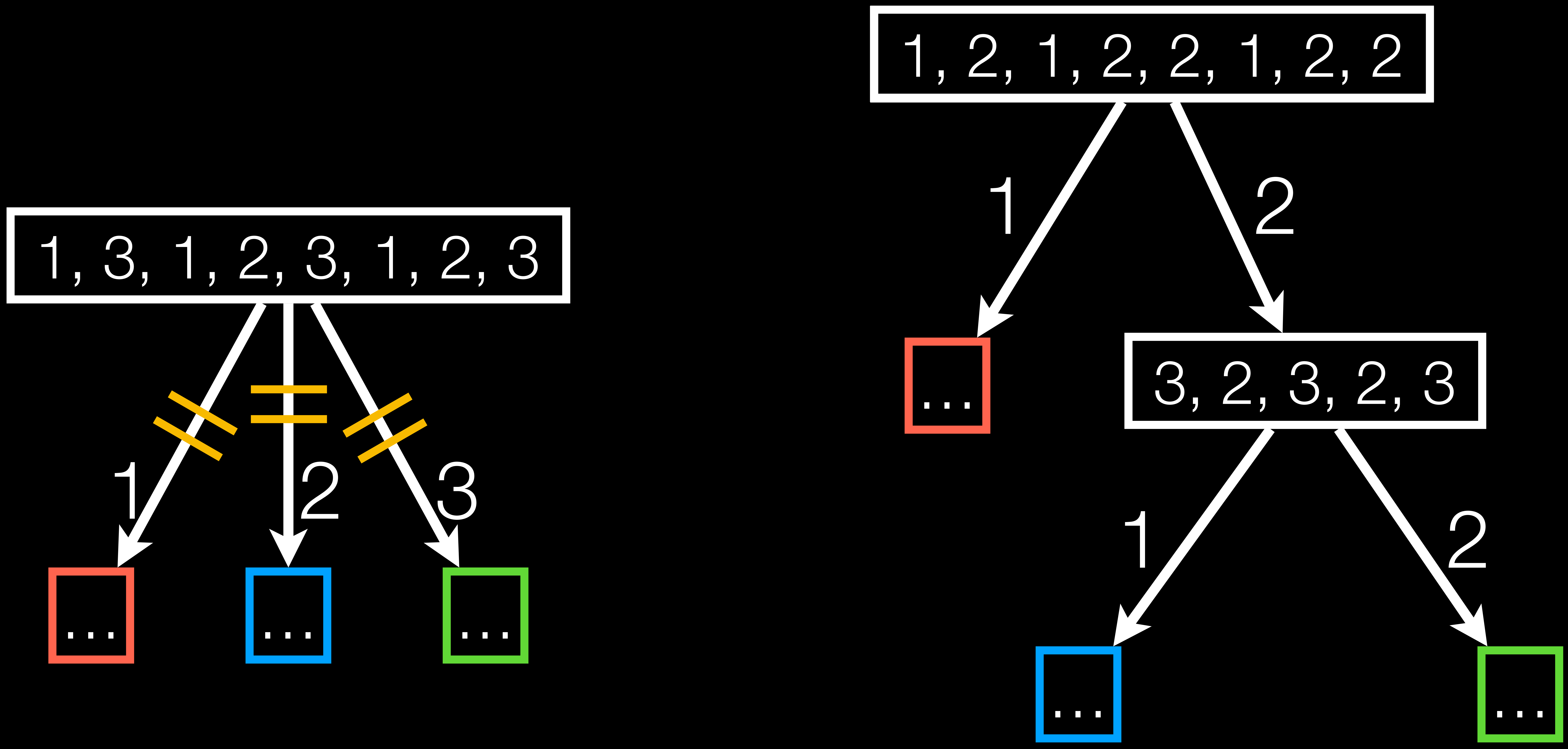
Compiling programs



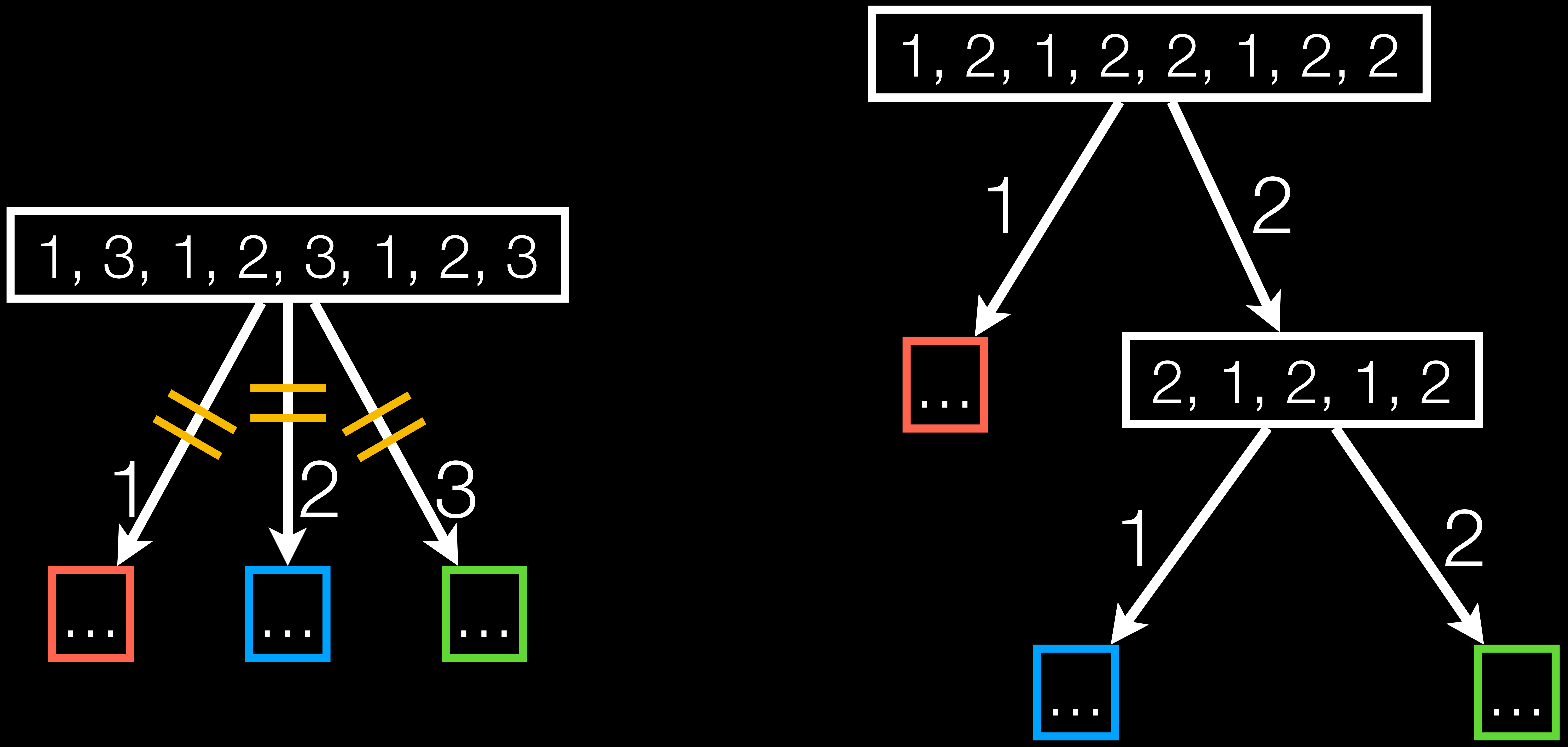
Compiling programs



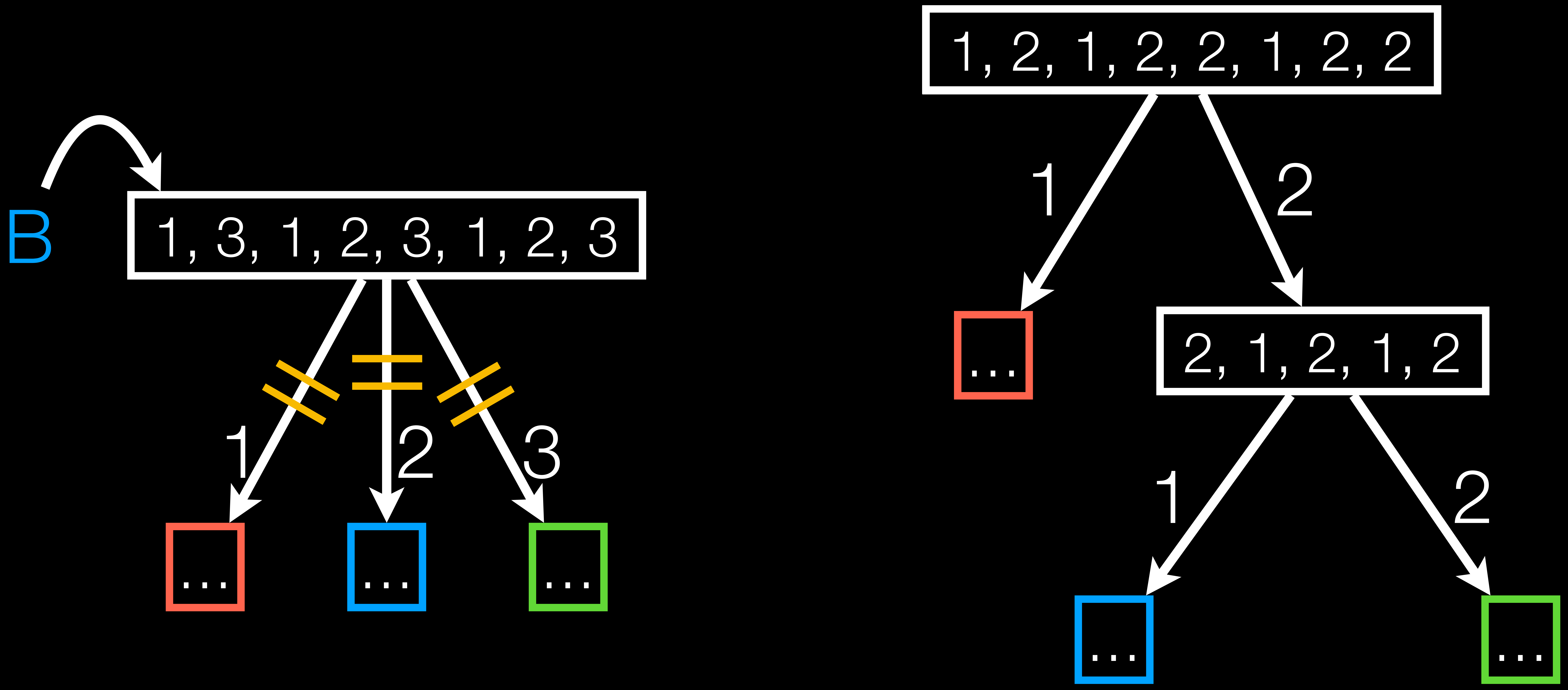
Compiling programs



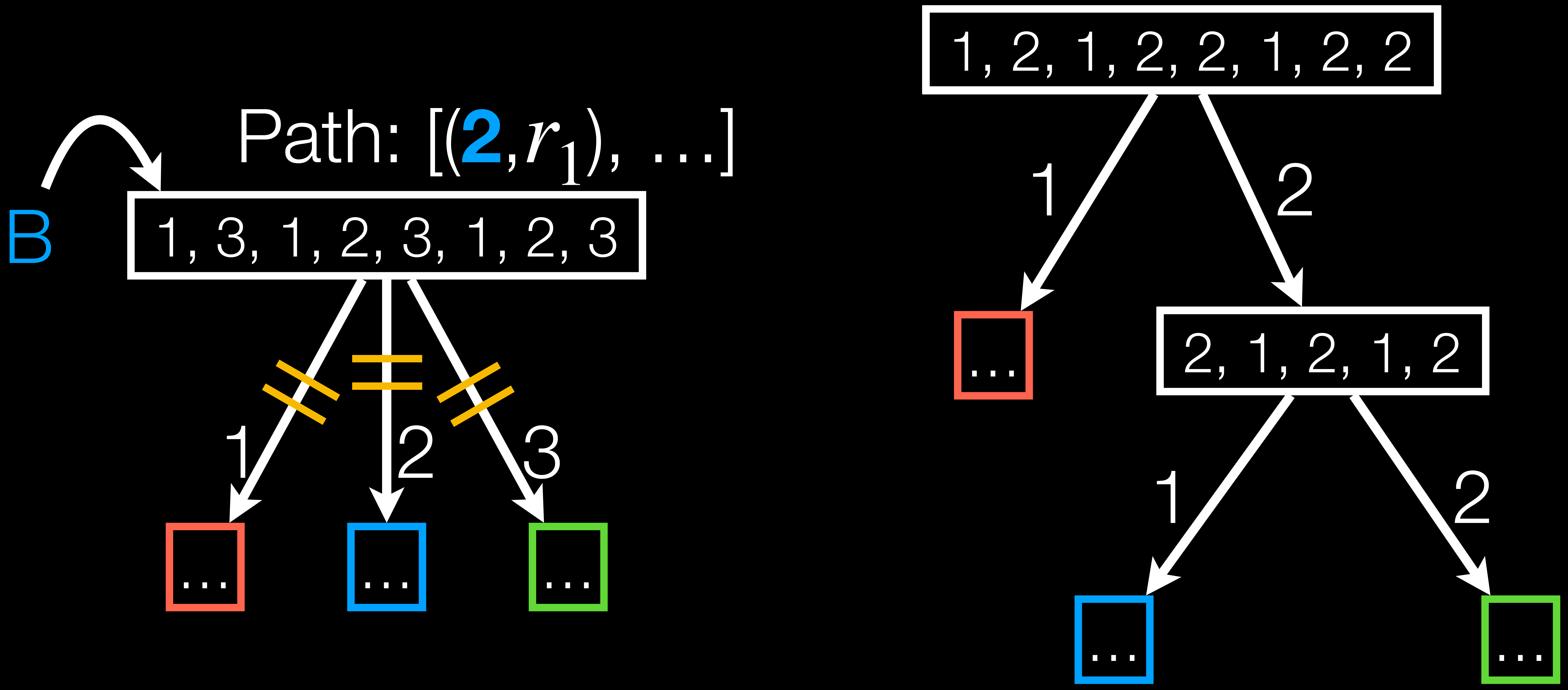
Compiling programs



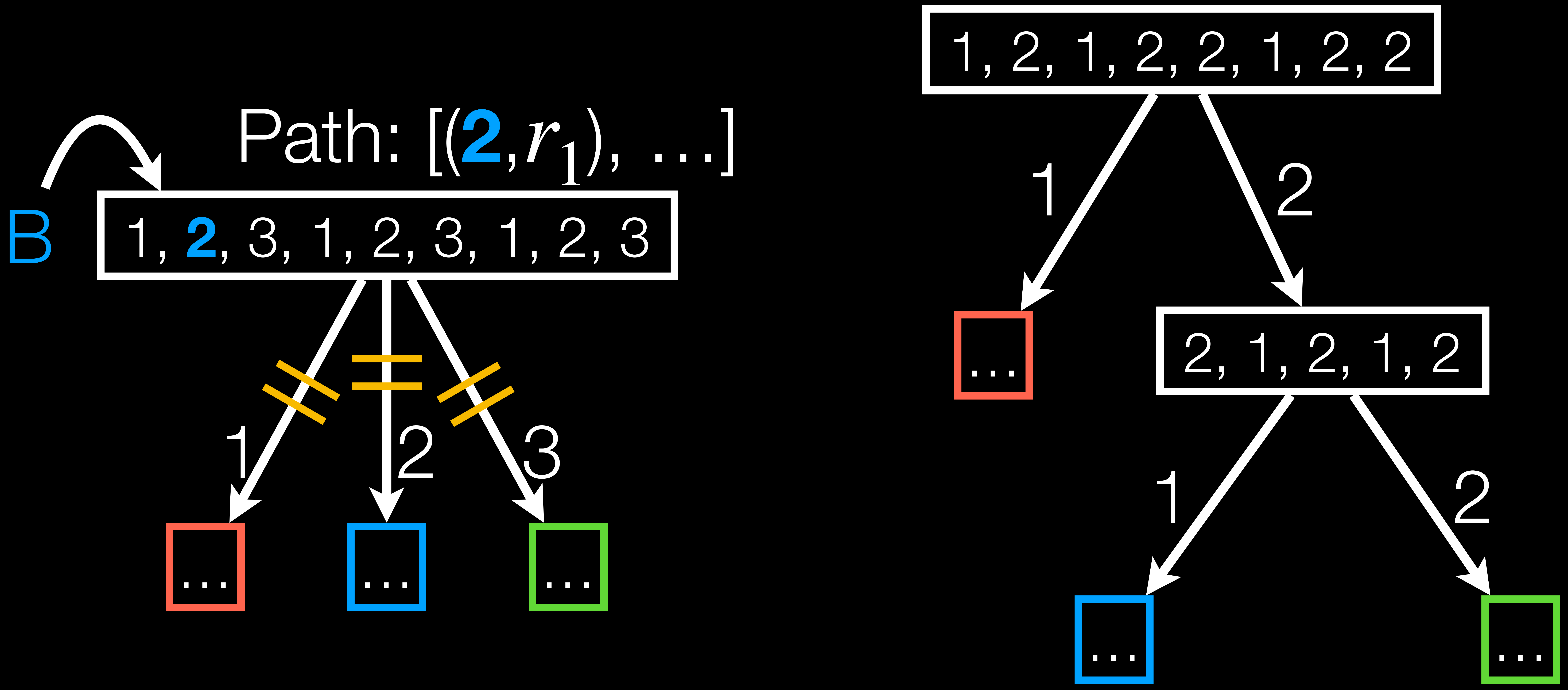
Compiling programs



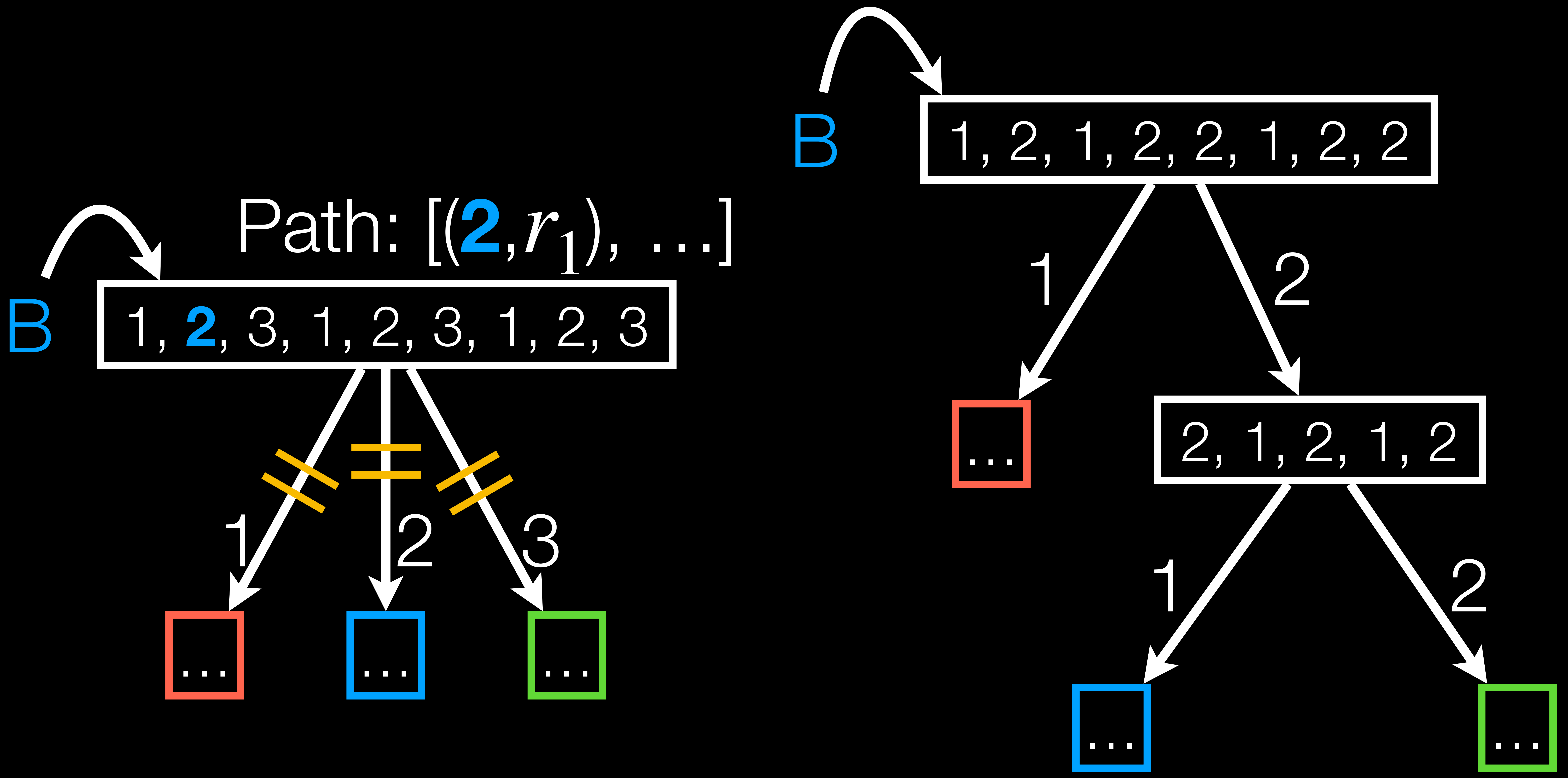
Compiling programs



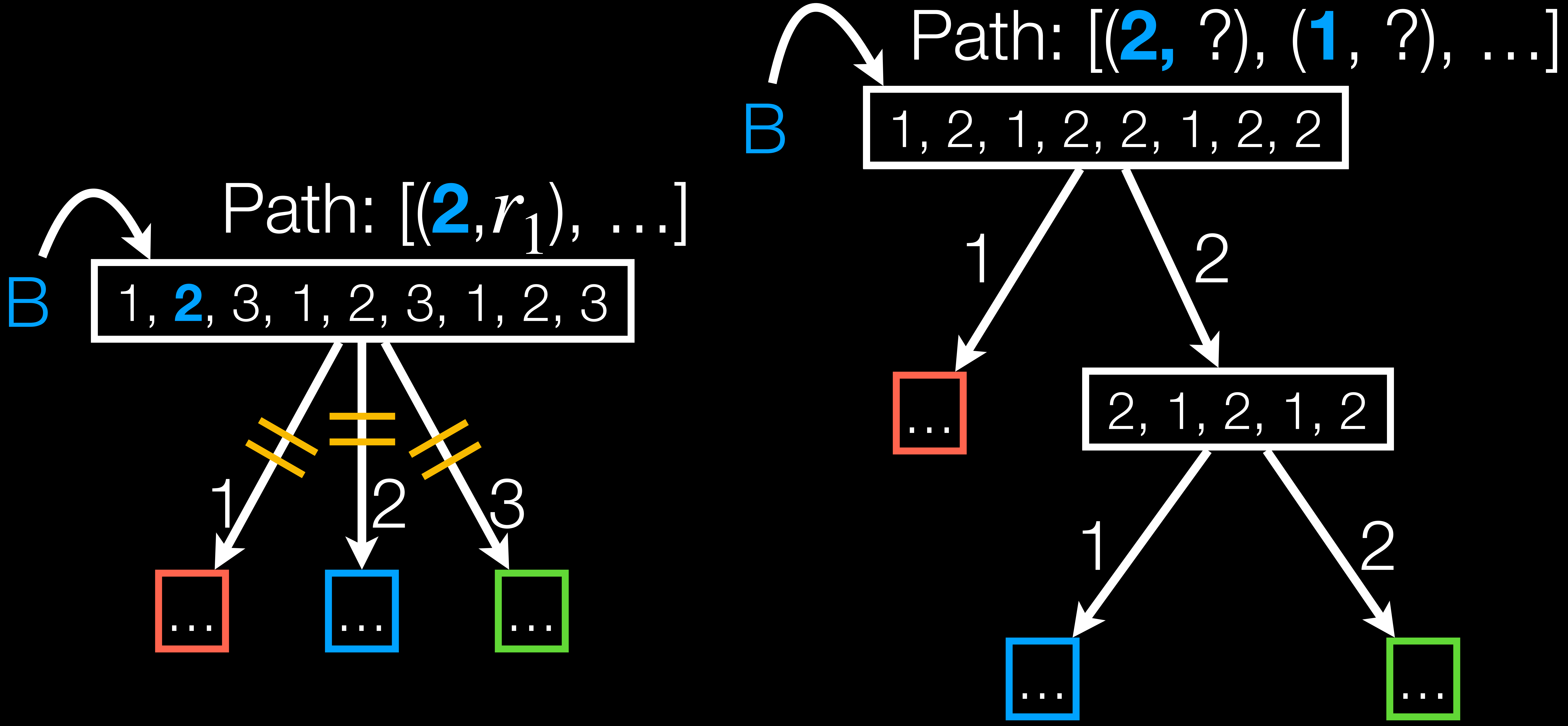
Compiling programs



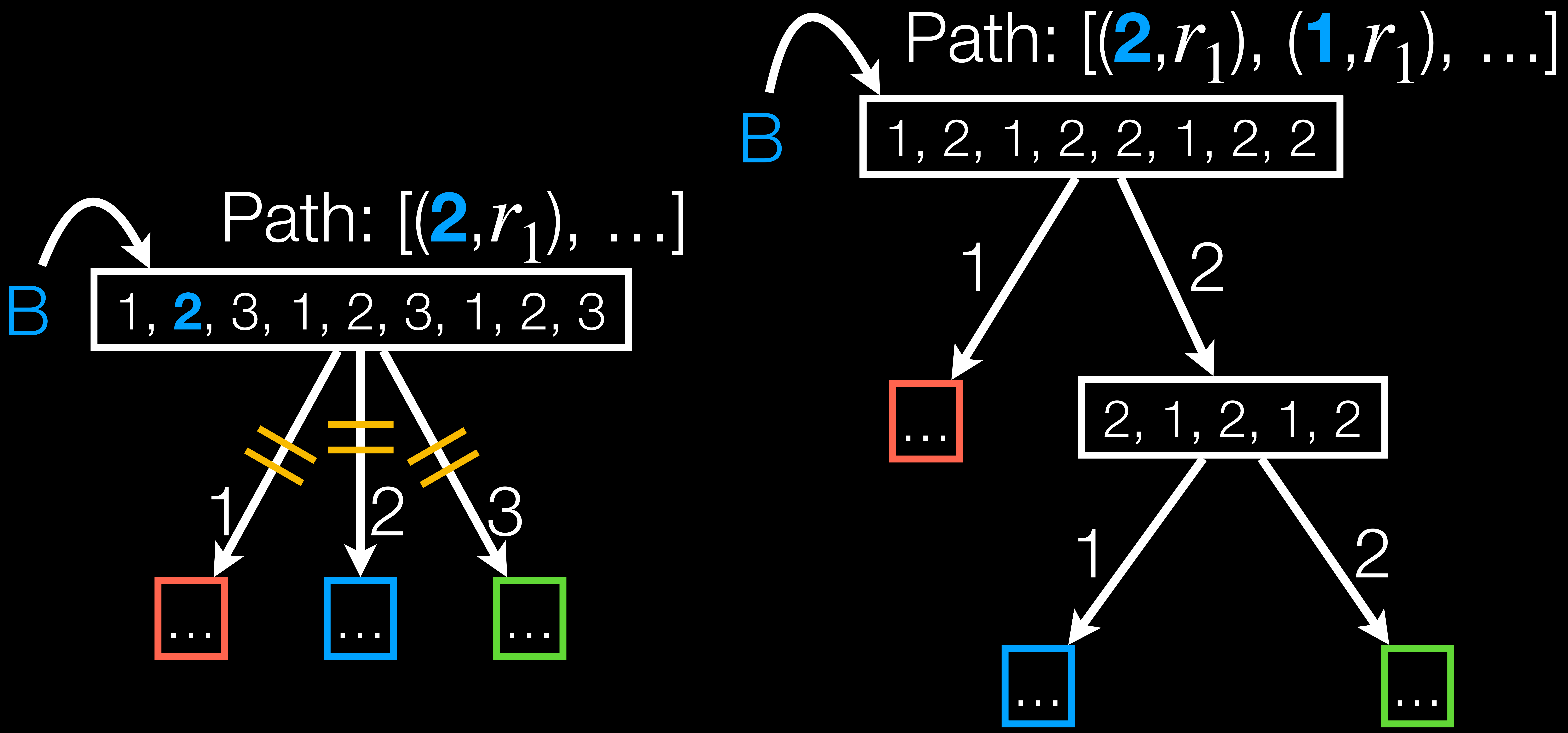
Compiling programs



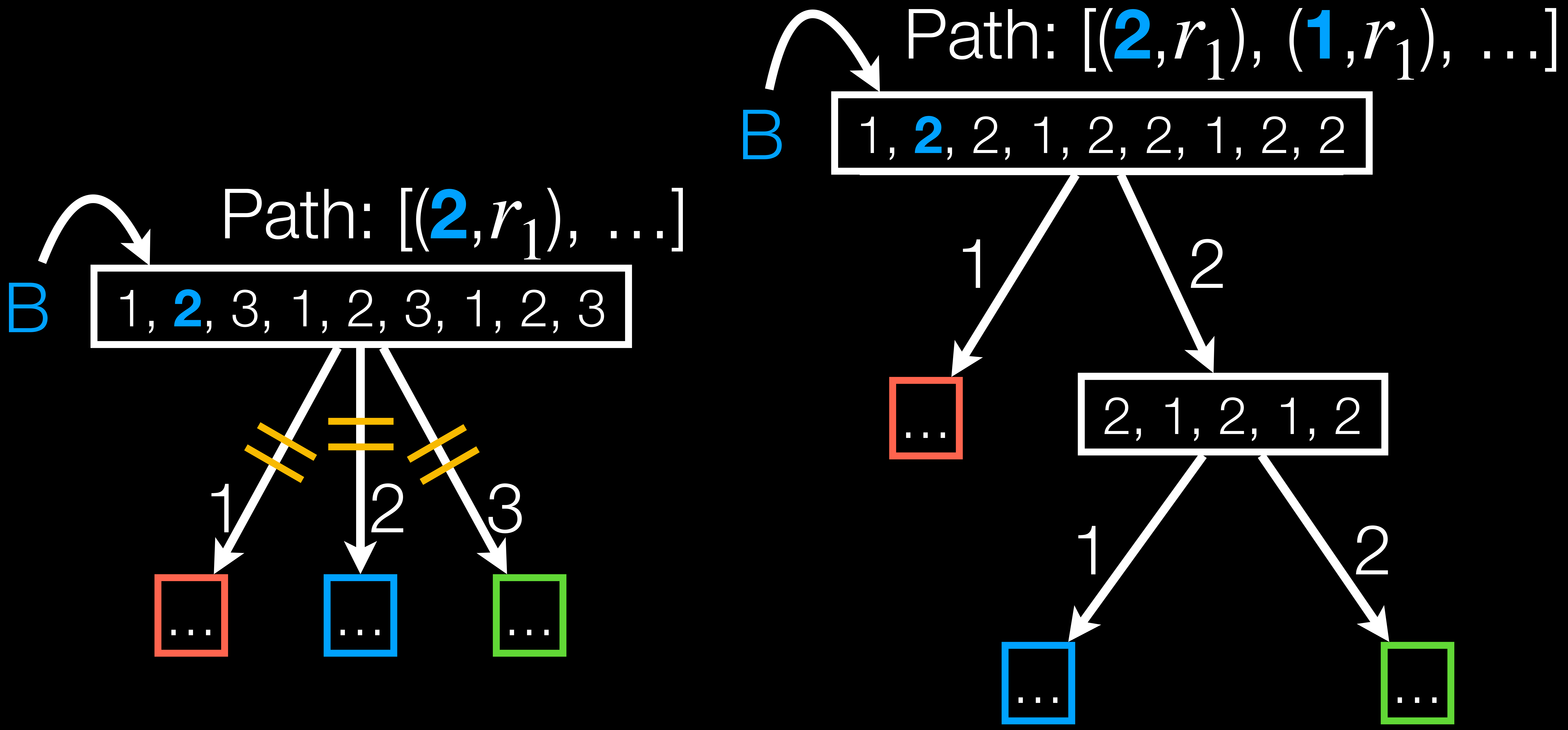
Compiling programs



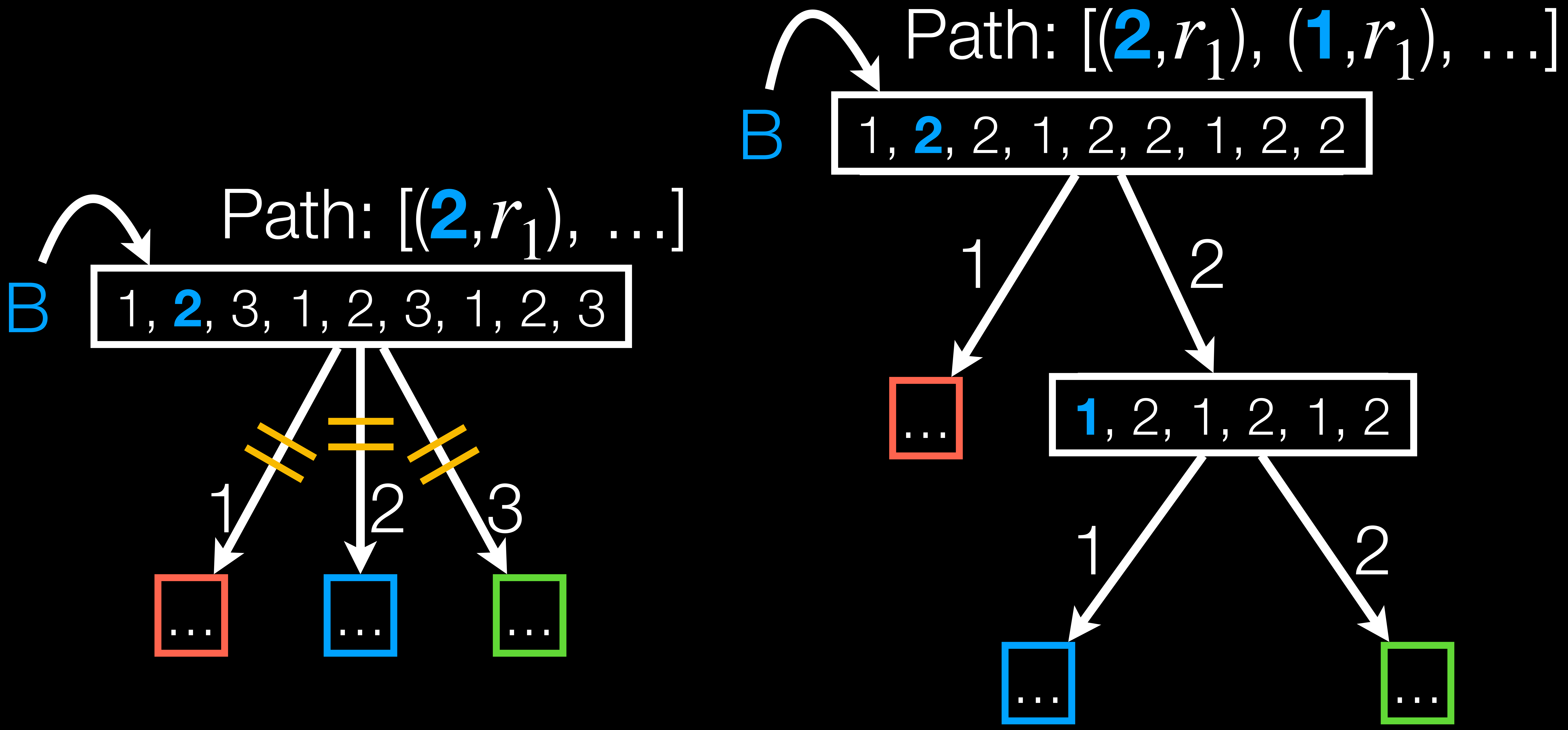
Compiling programs



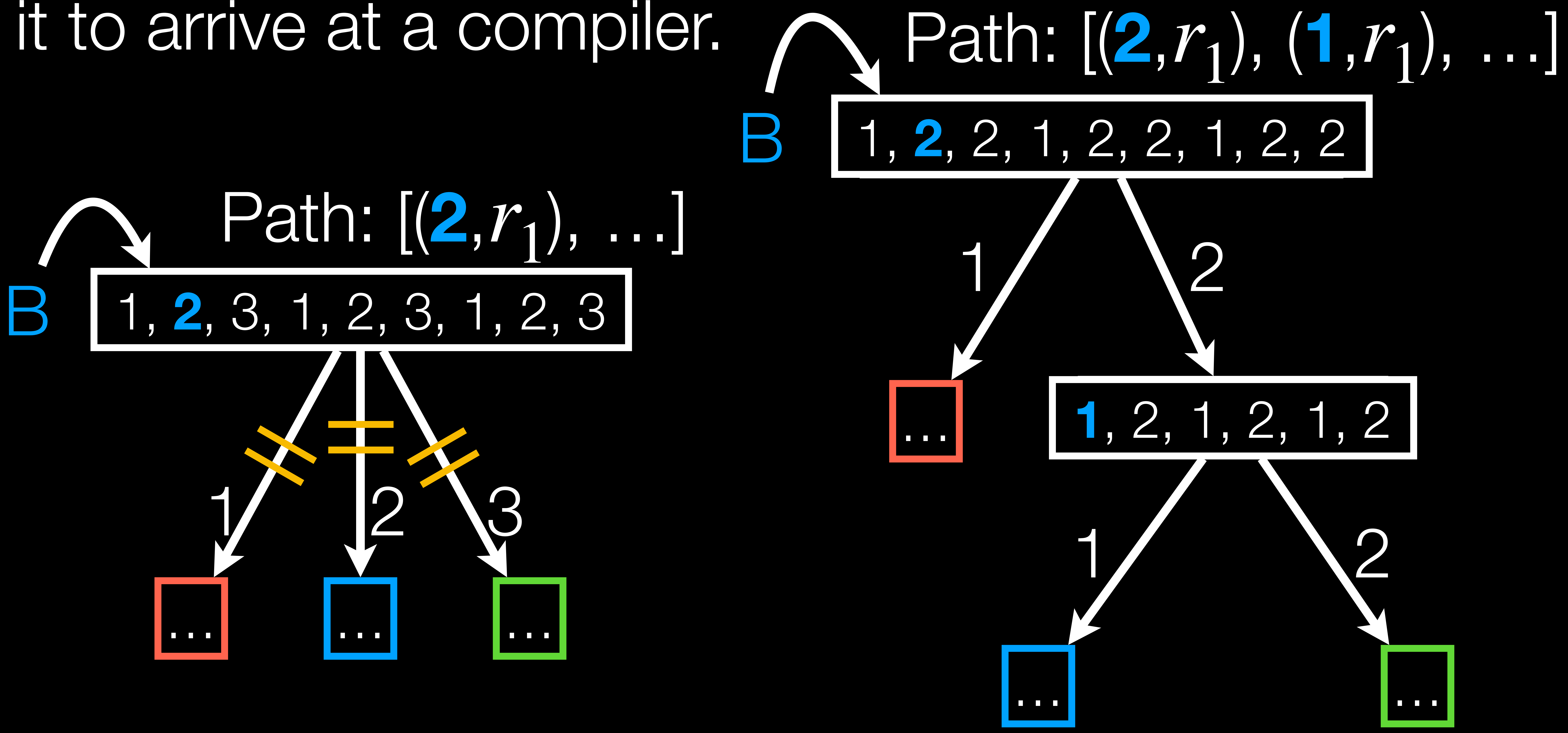
Compiling programs



Compiling programs



Given an embedding, we *lift* it to arrive at a compiler.

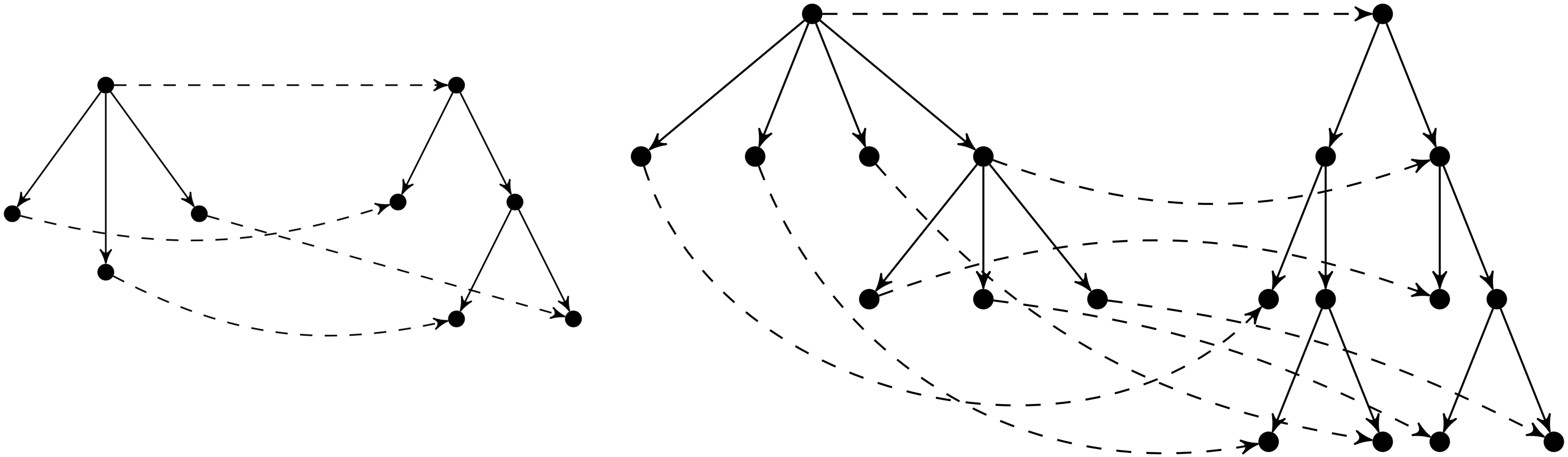


Generating embeddings automatically!

Generating embeddings automatically!

Homomorphic embedding.

Map root to root, leaves to leaves. Respect ancestry.



Generating embeddings automatically!

Homomorphic embedding.
Map root to root, leaves to leaves. Respect ancestry.

Two new algorithms,
both starting with heterogeneous source trees.

Generating embeddings automatically!

Homomorphic embedding.
Map root to root, leaves to leaves. Respect ancestry.

Two new algorithms,
both starting with heterogeneous source trees.

1. If target tree is regular *d*-ary for some *d*.

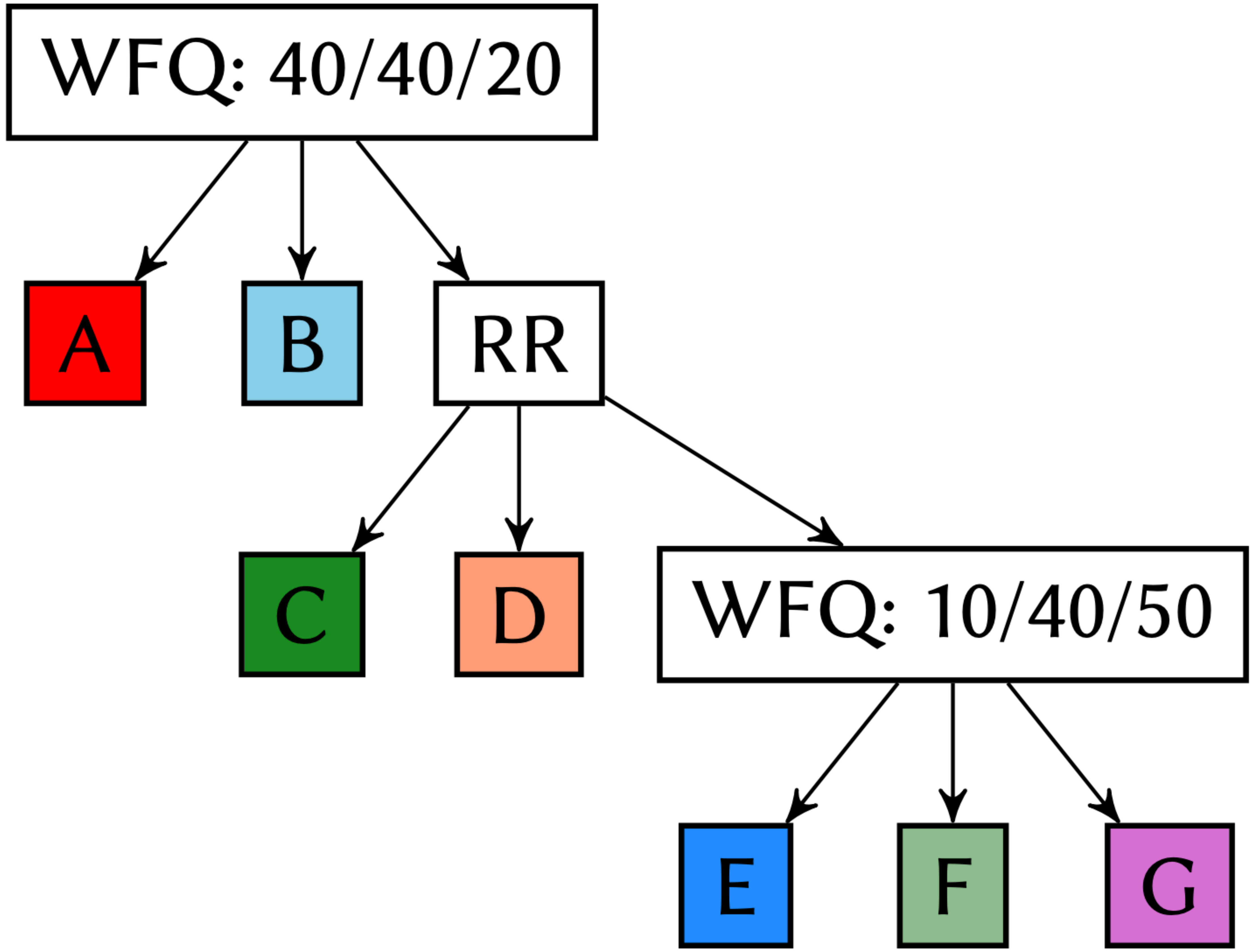
Generating embeddings automatically!

Homomorphic embedding.
Map root to root, leaves to leaves. Respect ancestry.

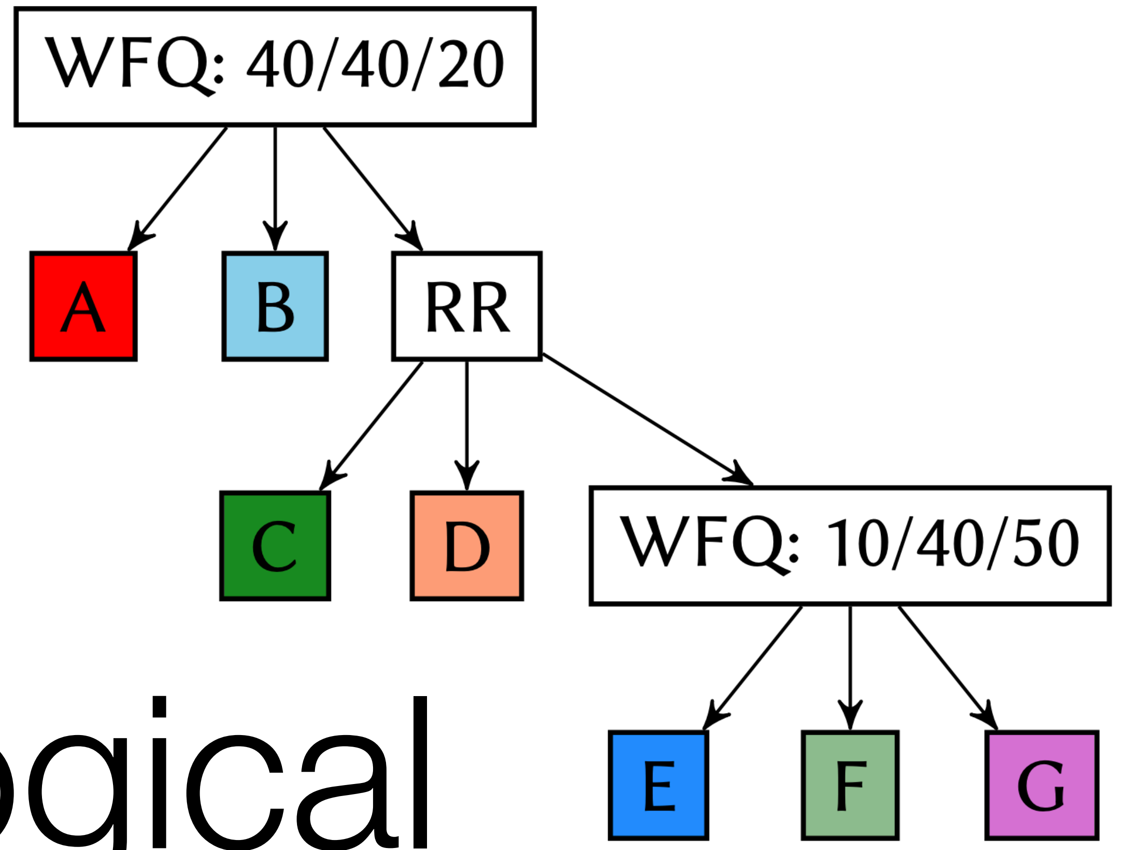
Two new algorithms,
both starting with heterogeneous source trees.

1. If target tree is regular *d*-ary for some *d*.
2. If target tree is itself heterogeneous.

Workflow

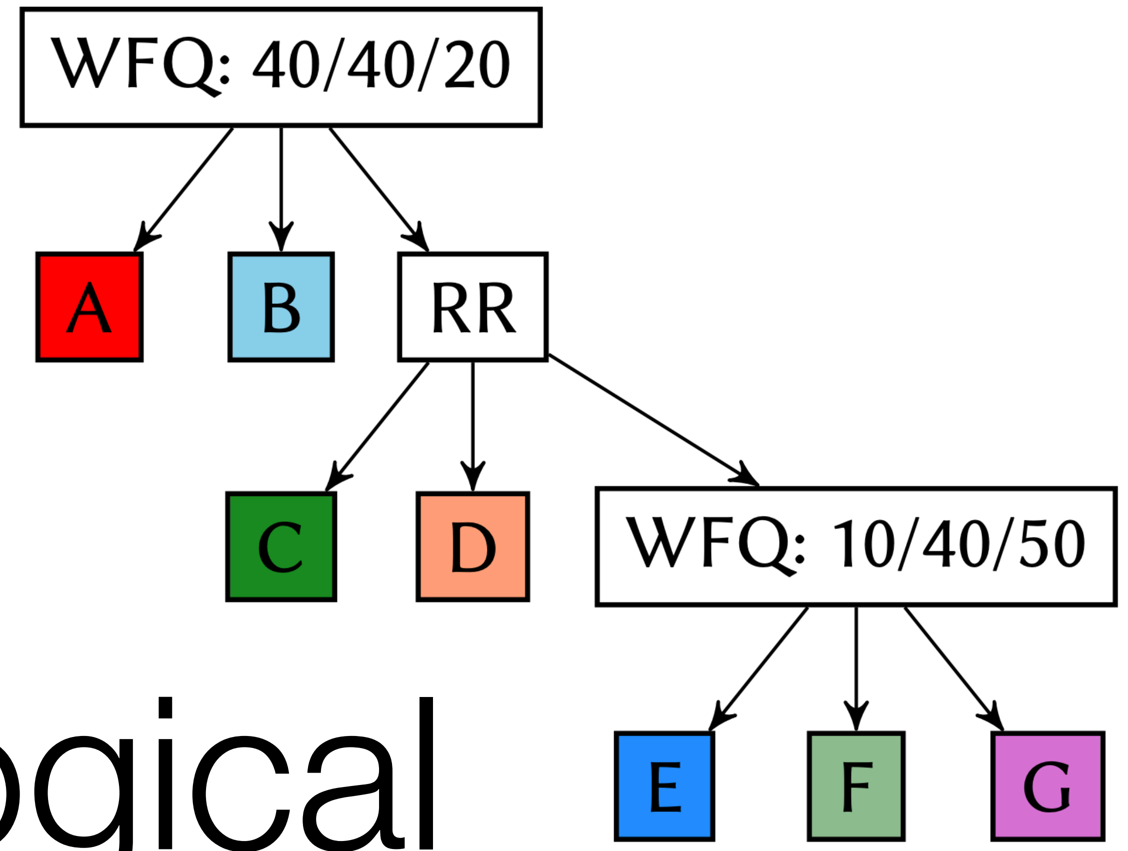


Workflow



logical

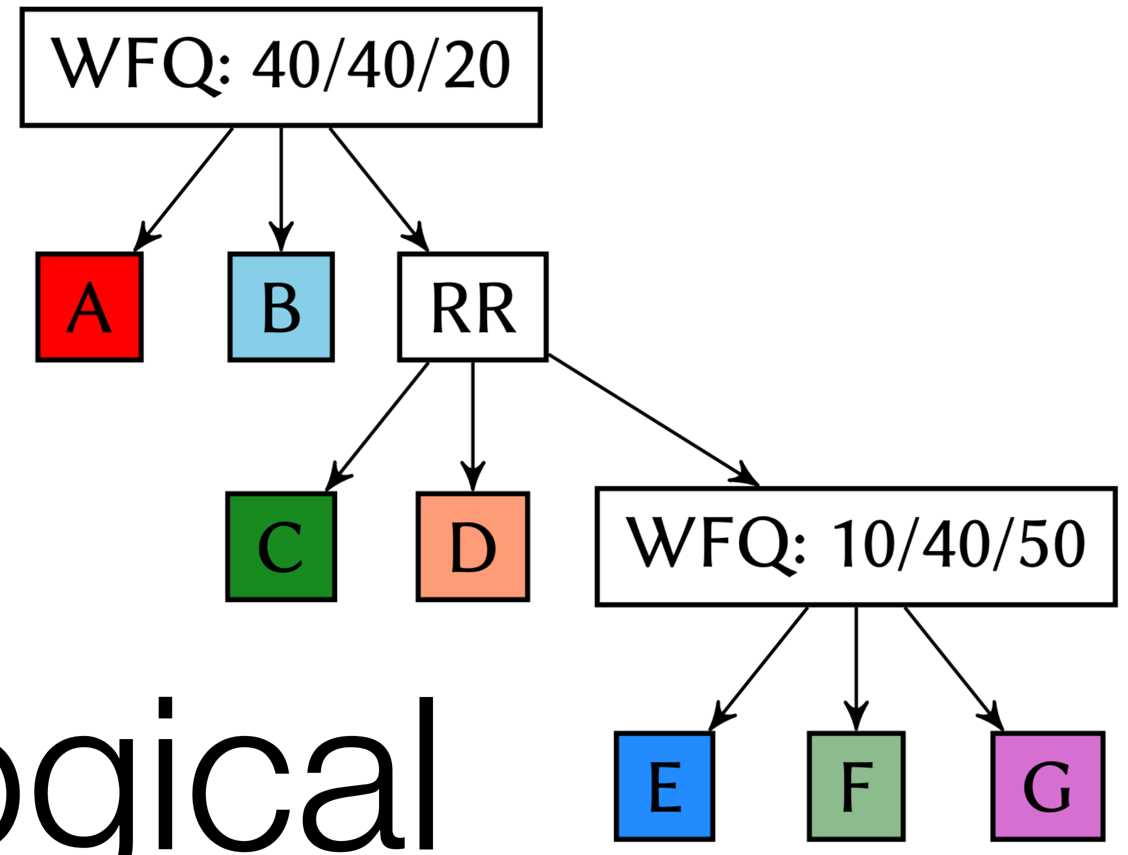
Workflow



logical

But the hardware supports
a regular-branching binary tree.

Workflow

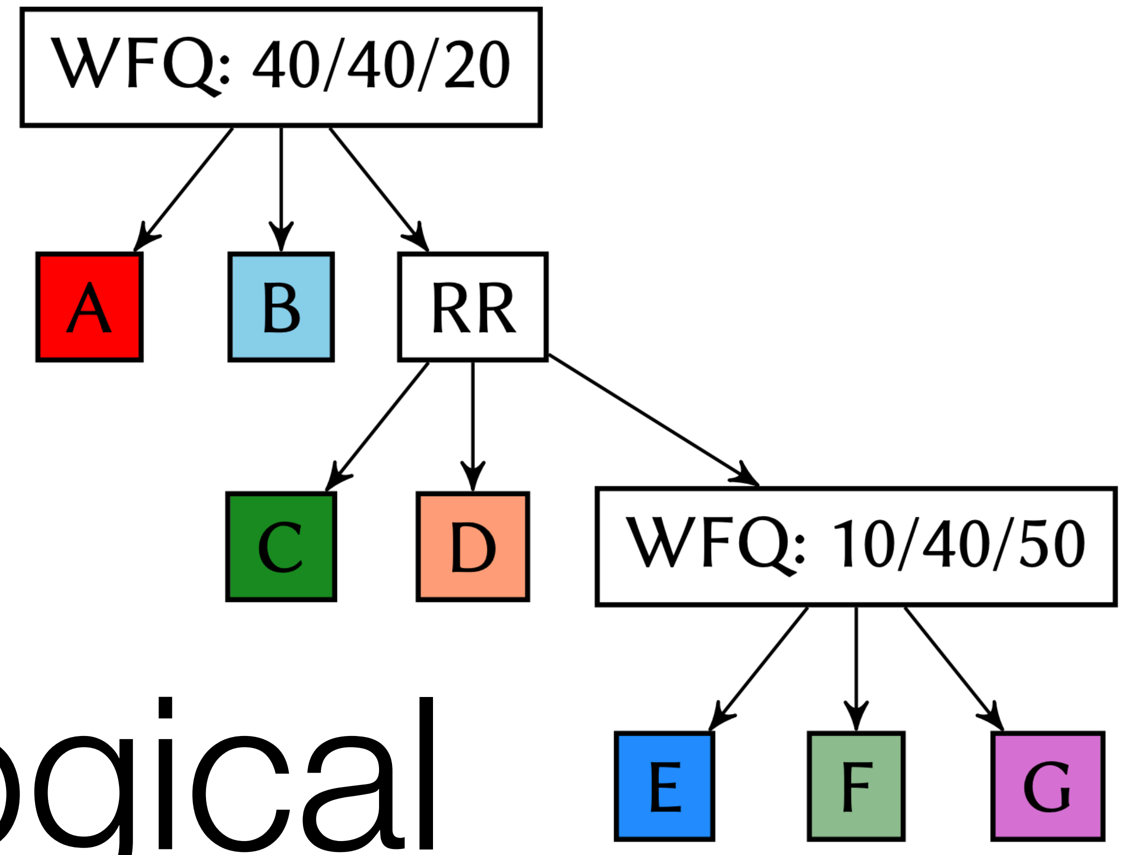


logical

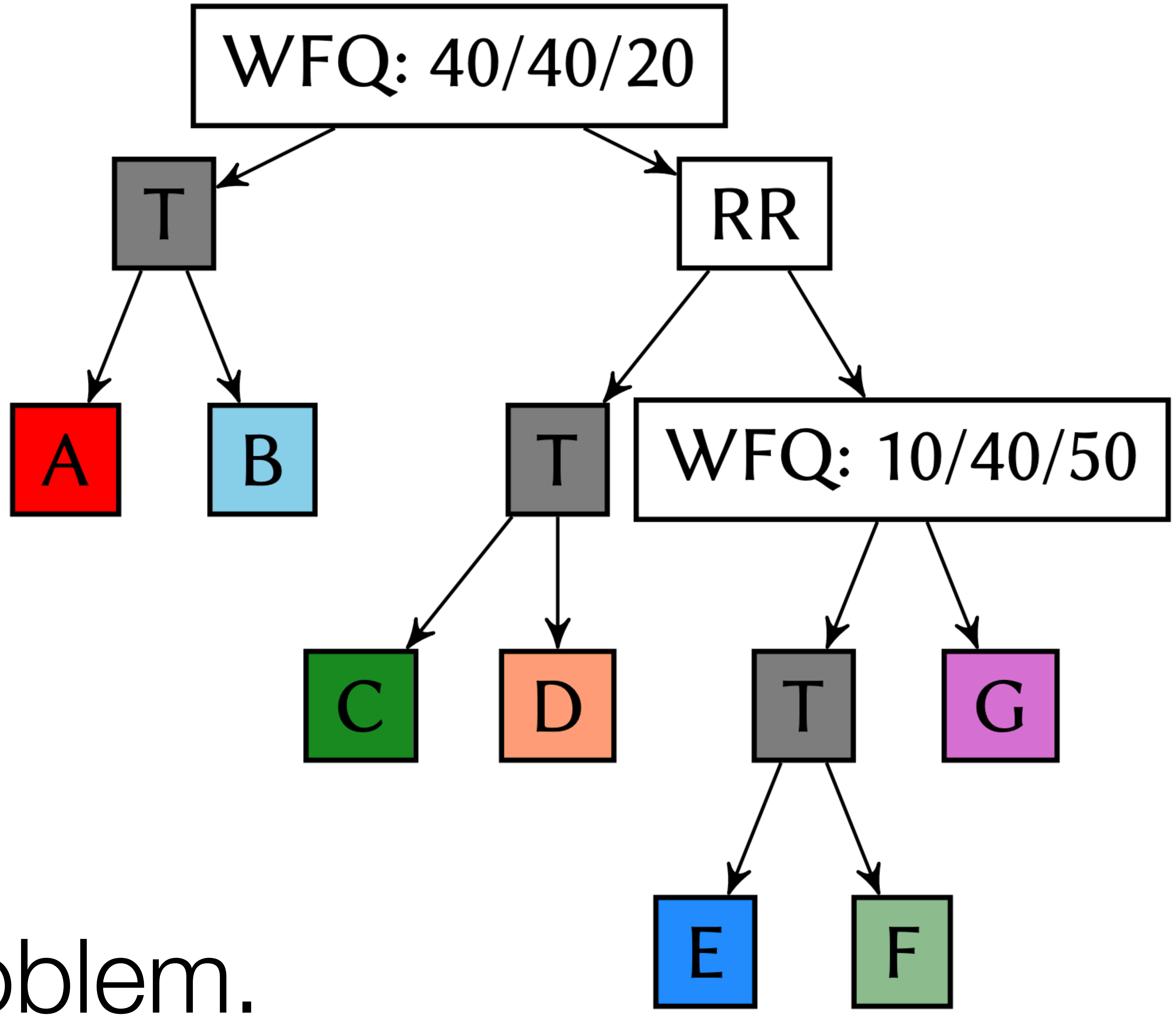
But the hardware supports
a regular-branching binary tree.

No problem.
Here's how I'll use that tree.

Workflow



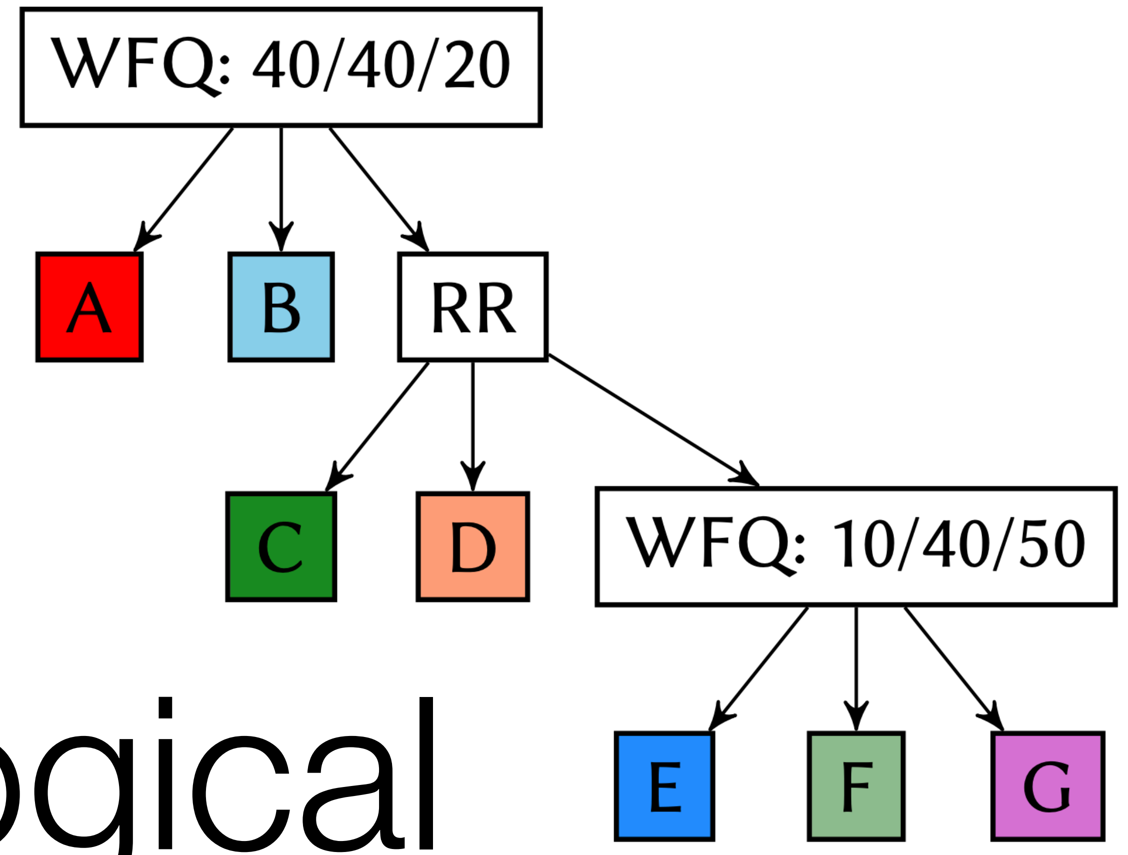
logical



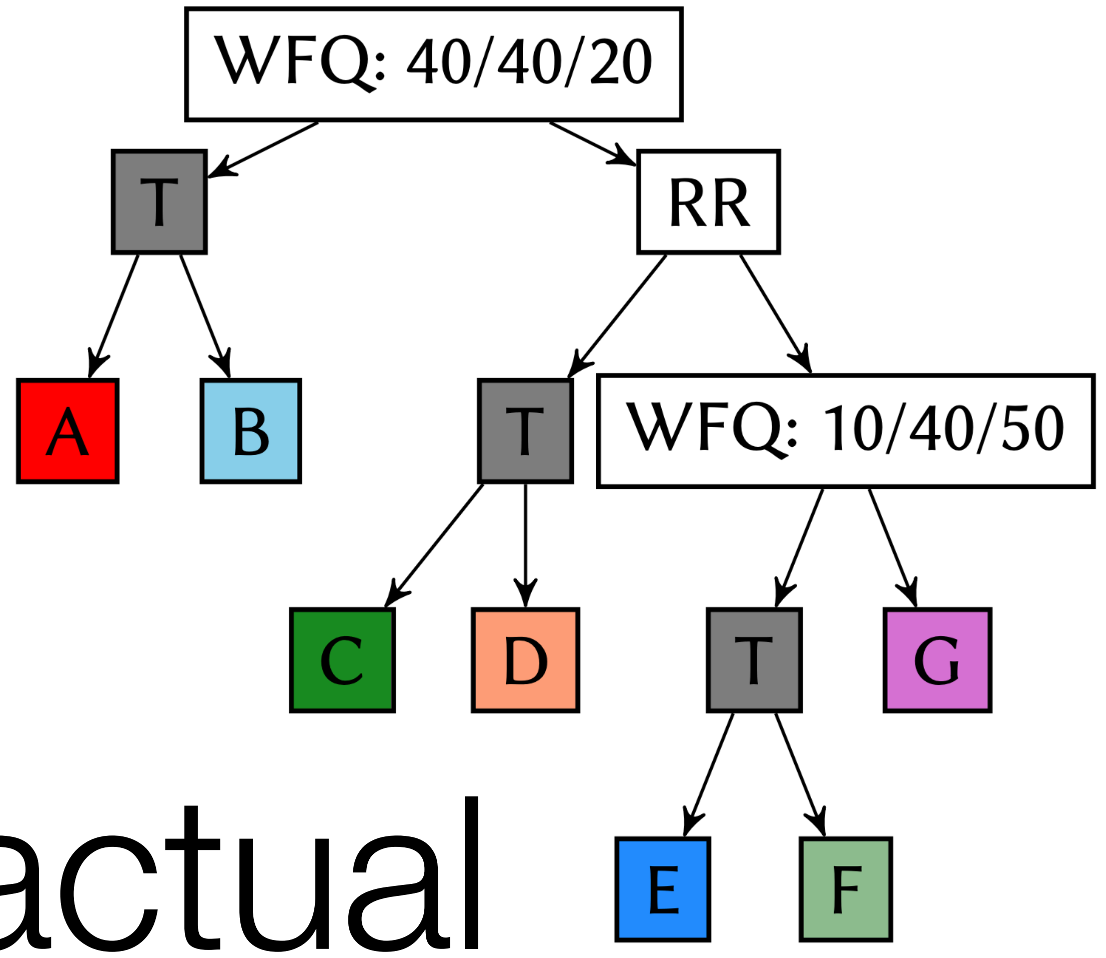
No problem.

Here's how I'll use that tree.

Workflow

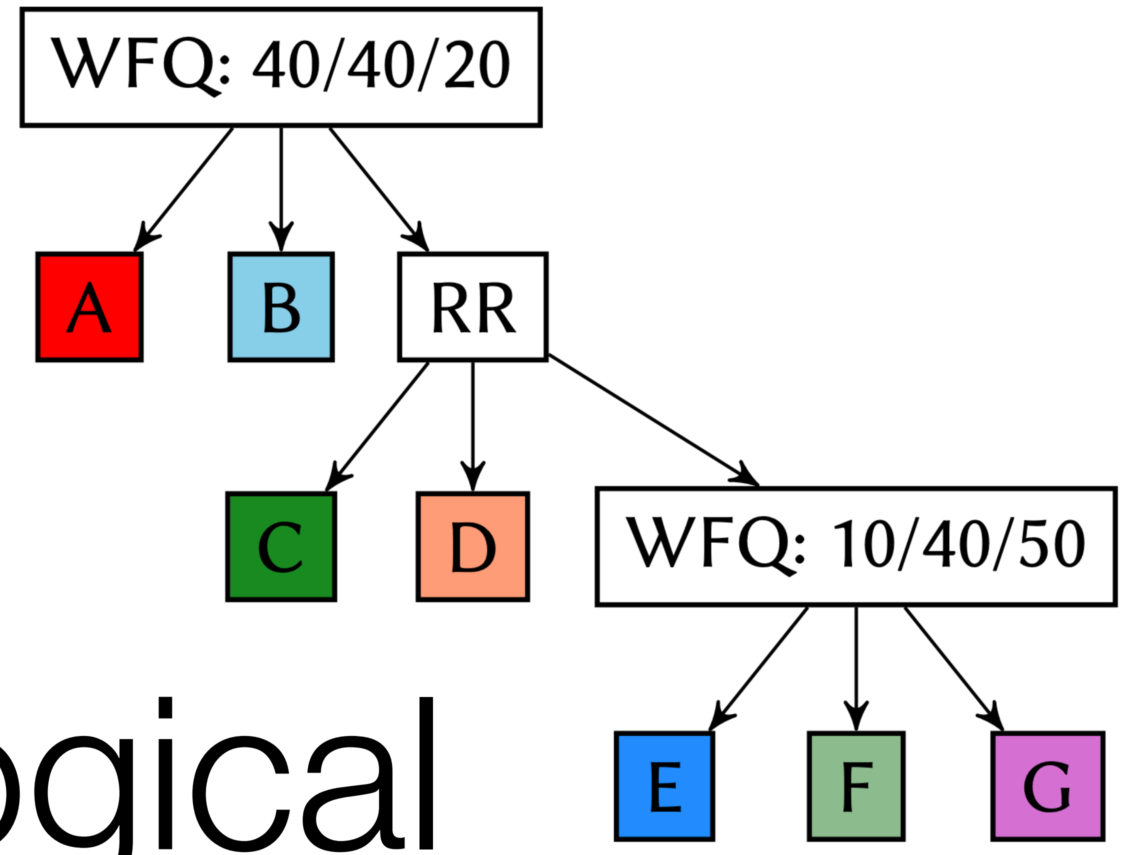


logical

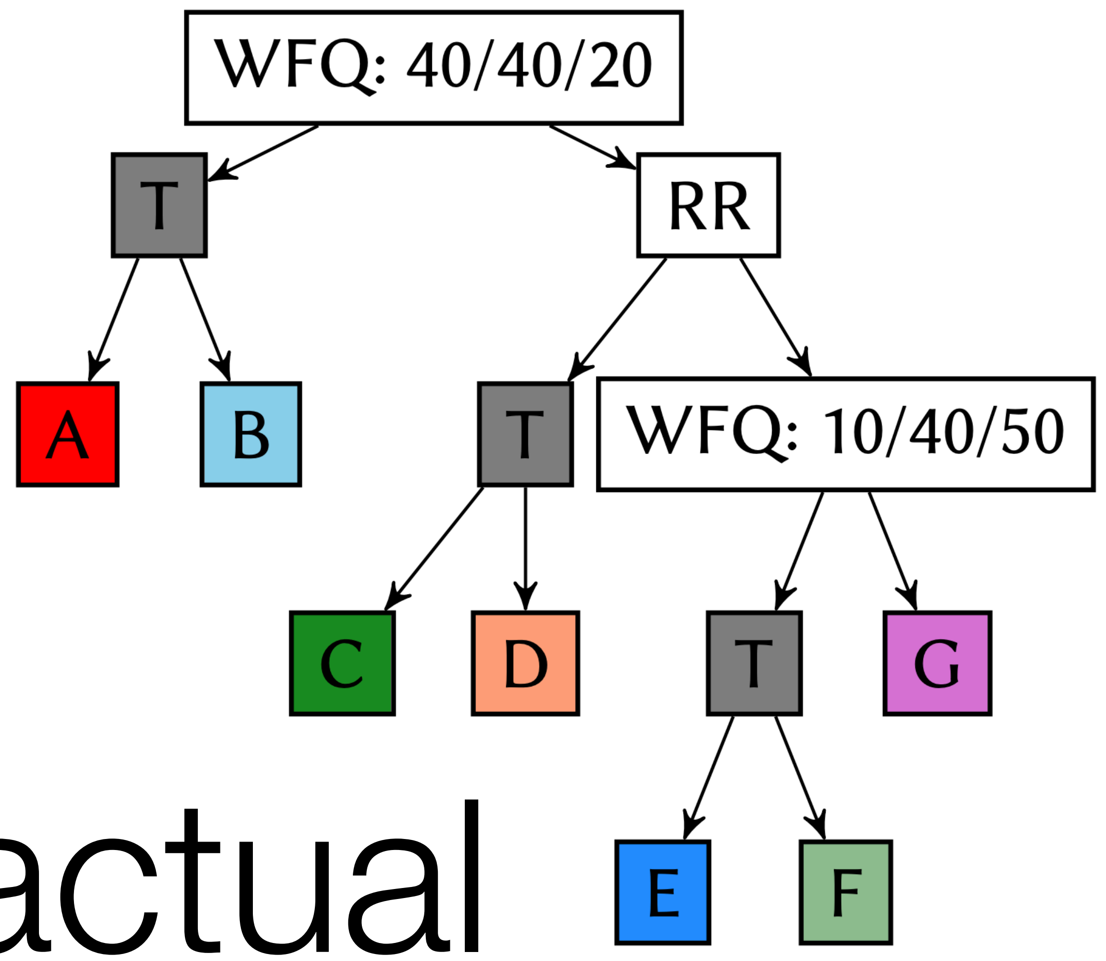


actual

Simulation

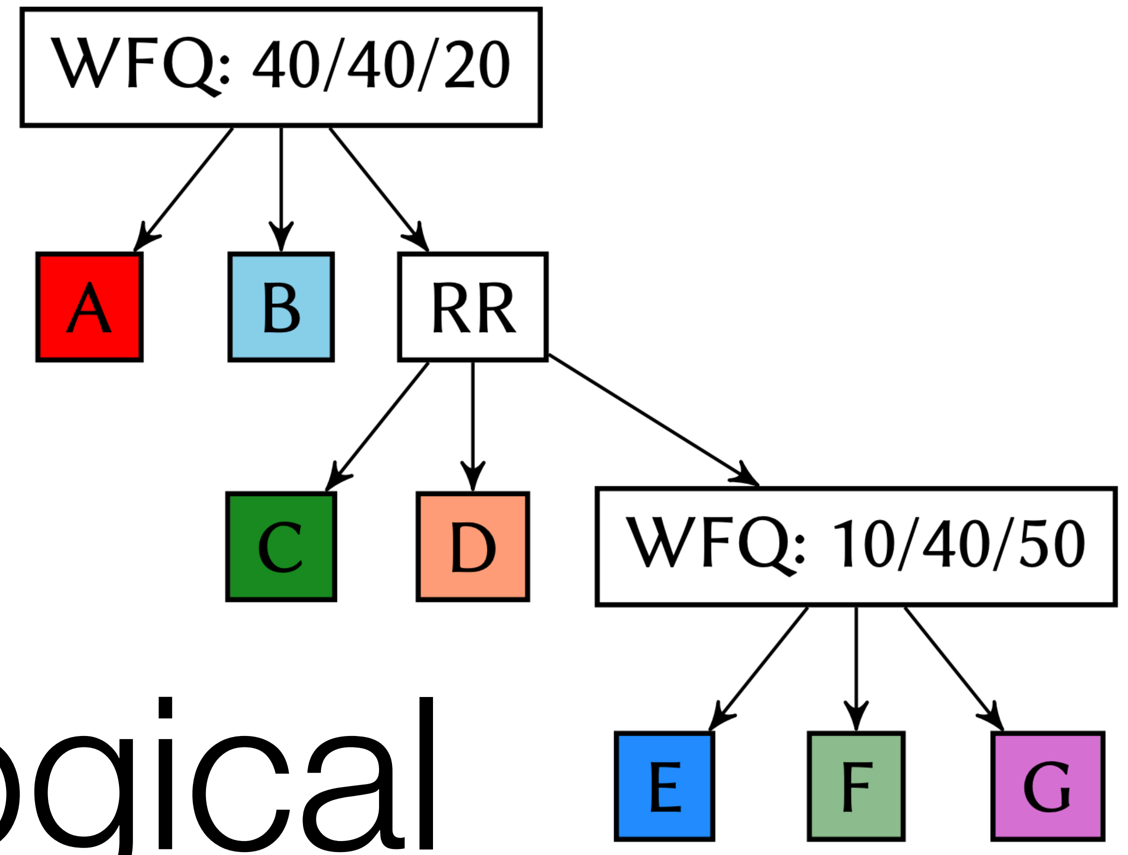


logical

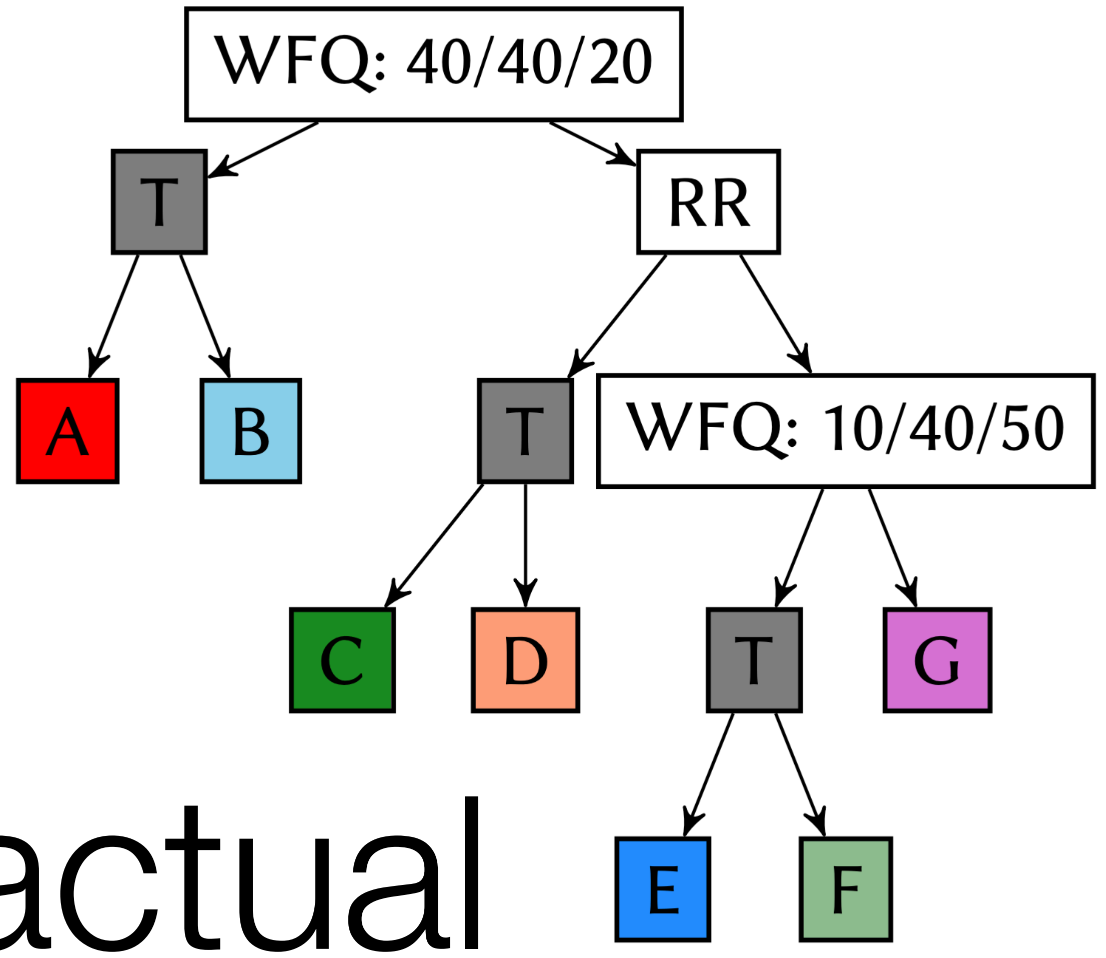


actual

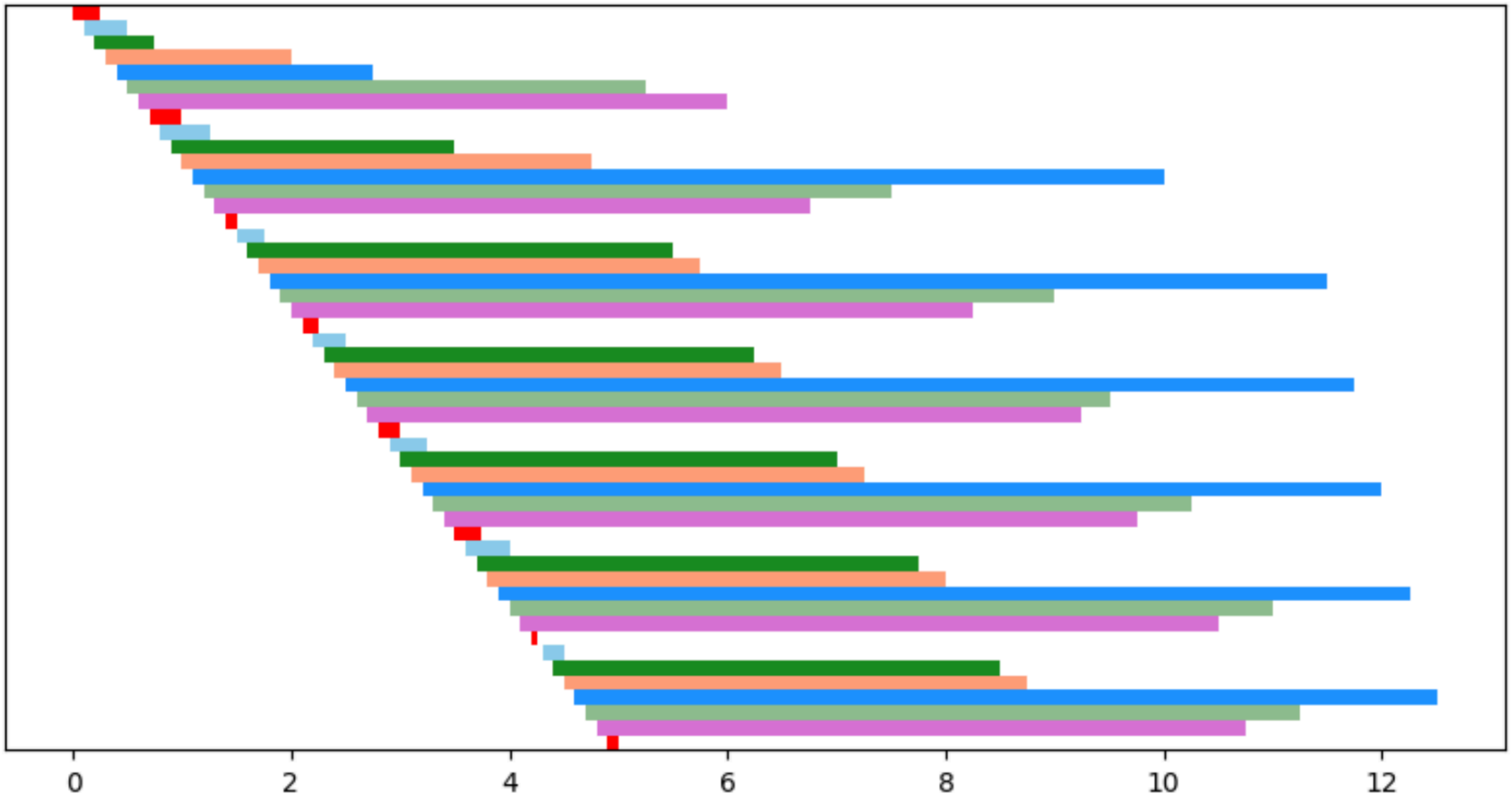
Simulation



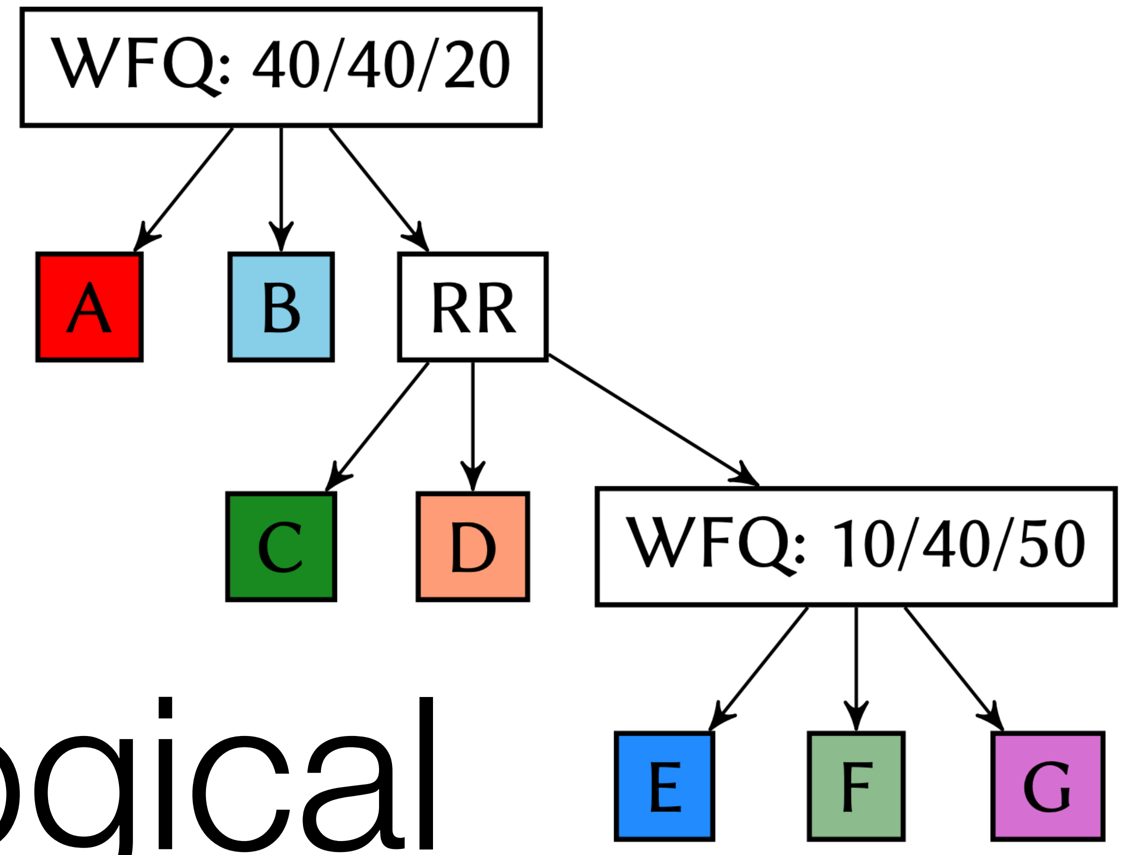
logical



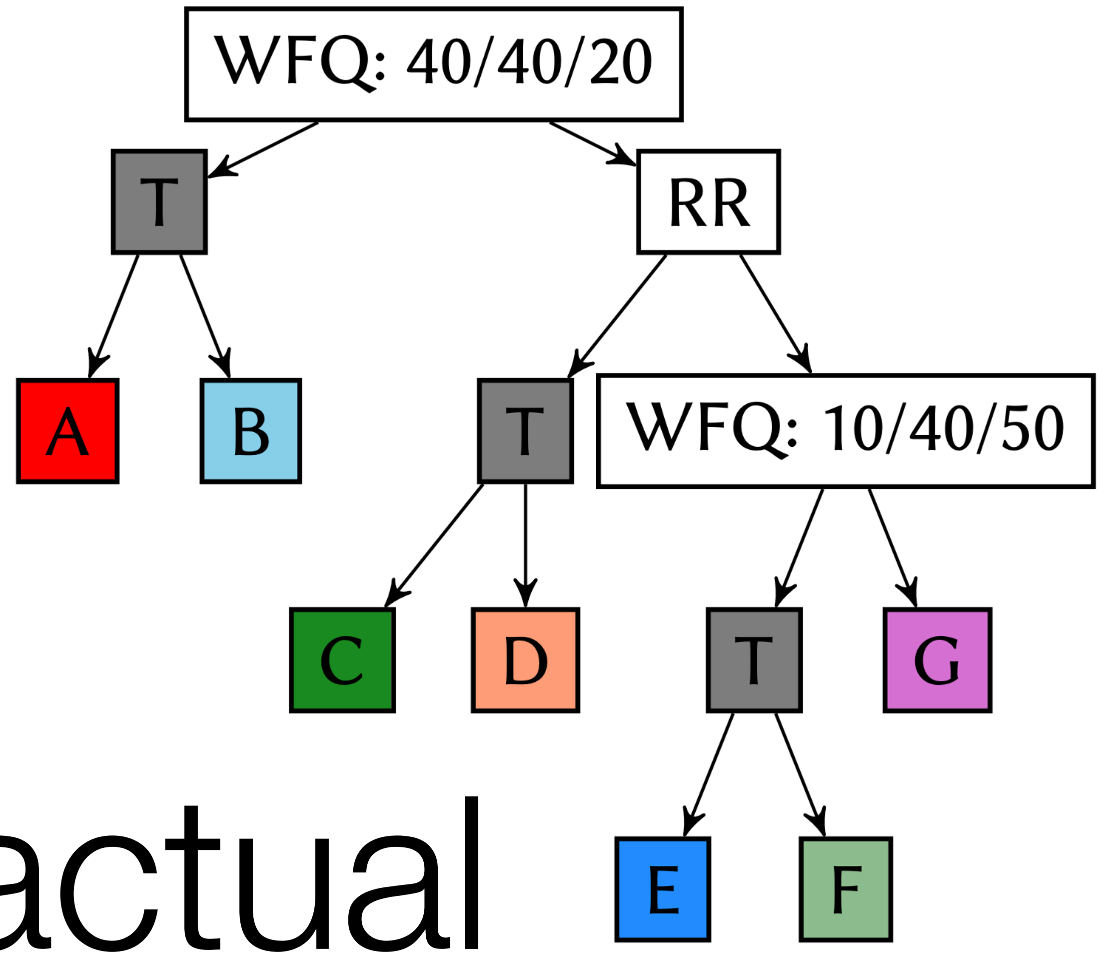
actual



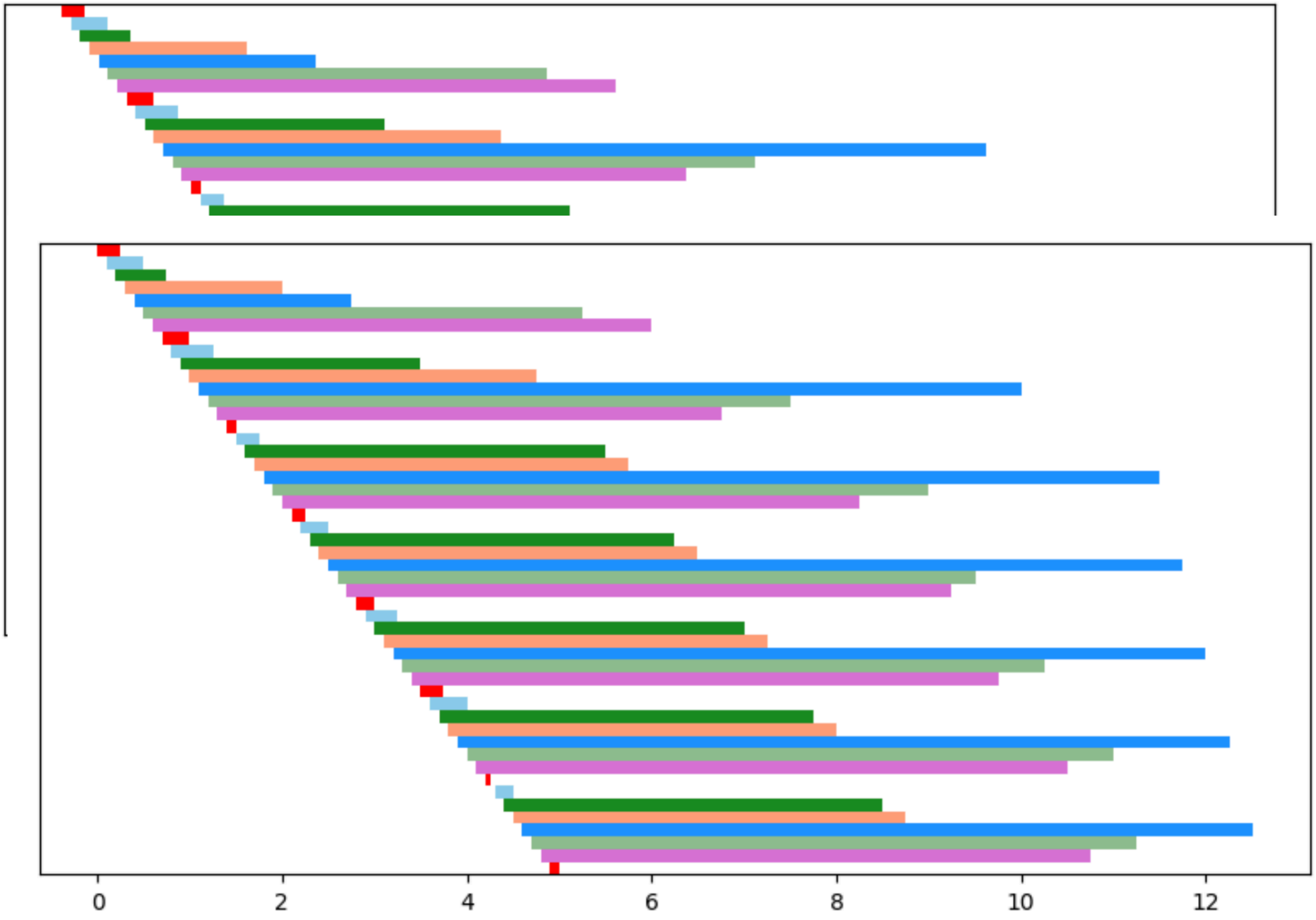
Simulation



logical



actual



Underlying formalism

Underlying formalism

$$\frac{}{* \in \mathsf{Topo}} \qquad \frac{n \in \mathbb{N} \qquad ts \in \mathsf{Topo}^n}{\mathsf{Node}(ts) \in \mathsf{Topo}}$$

$$\frac{p \in \mathsf{PIFO}(\mathsf{Pkt})}{\mathsf{Leaf}(p) \in \mathsf{PIFOTree}(\underline{*})} \qquad \frac{n \in \mathbb{N} \qquad ts \in \mathsf{Topo}^n \qquad p \in \mathsf{PIFO}(\{1,\ldots,n\}) \quad \forall 1 \leq i \leq n.\ qs[i] \in \mathsf{PIFOTree}(ts[i])}{\mathsf{Internal}(qs, p) \in \mathsf{PIFOTree}(\underline{\mathsf{Node}(ts)})}$$

Topo Topo

$$\frac{r \in \mathsf{Rk}}{r \in \mathsf{Path}(\underline{*})} \qquad \frac{ts \in \mathsf{Topo}^n \qquad 1 \leq i \leq n \qquad r \in \mathsf{Rk} \qquad pt \in \mathsf{Path}(ts[i])}{(i, r) :: pt \in \mathsf{Path}(\underline{\mathsf{Node}(ts)})}$$

Topo Topo

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

PIFOTree Path PIFOTree

A general way to deploy PIFO trees.

A general way to deploy PIFO trees.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

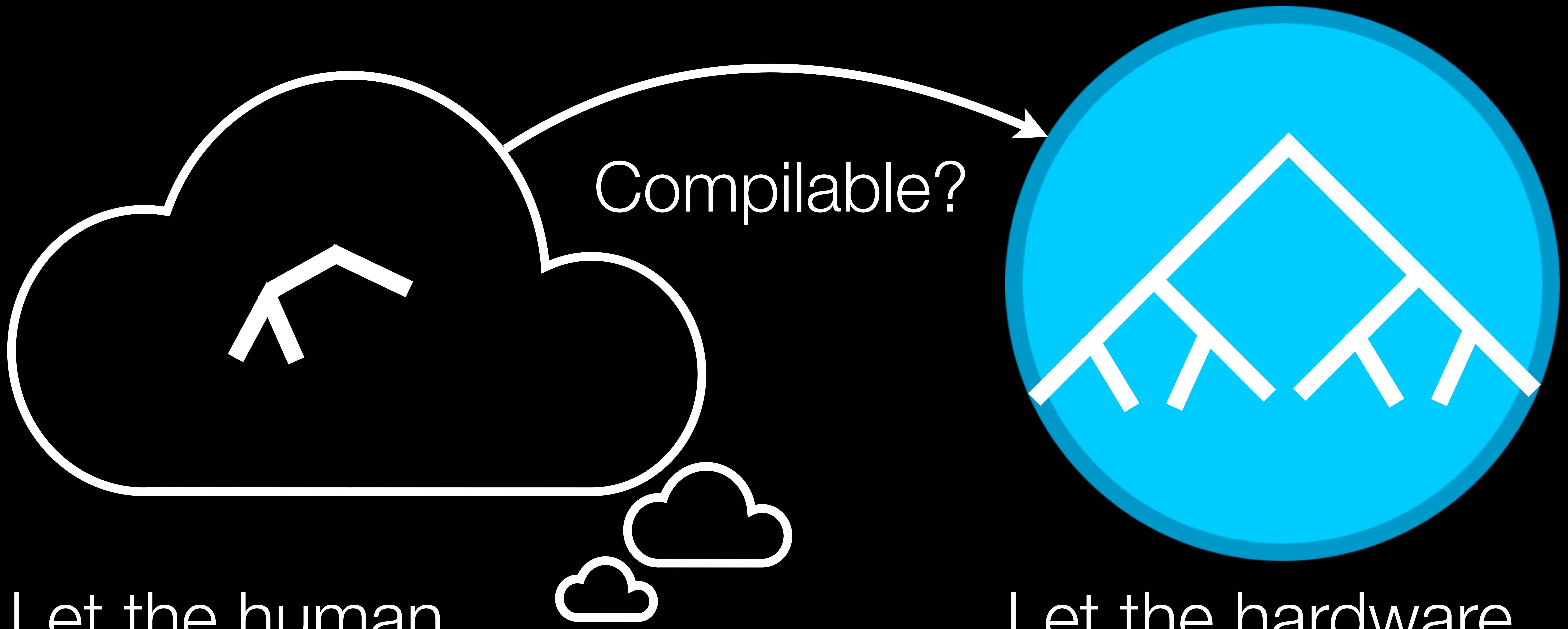
Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.



Let the human program against some tree.

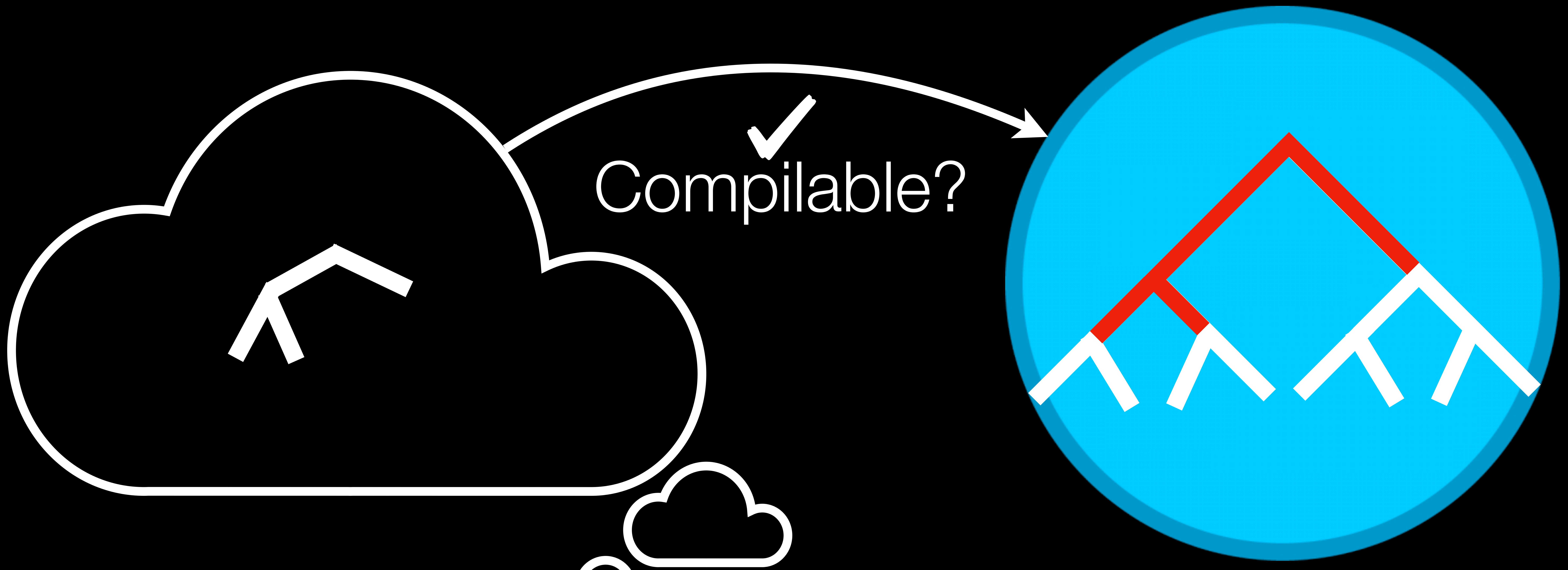
Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.



Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees.

Let the human program against some tree.

Let the hardware support some tree.

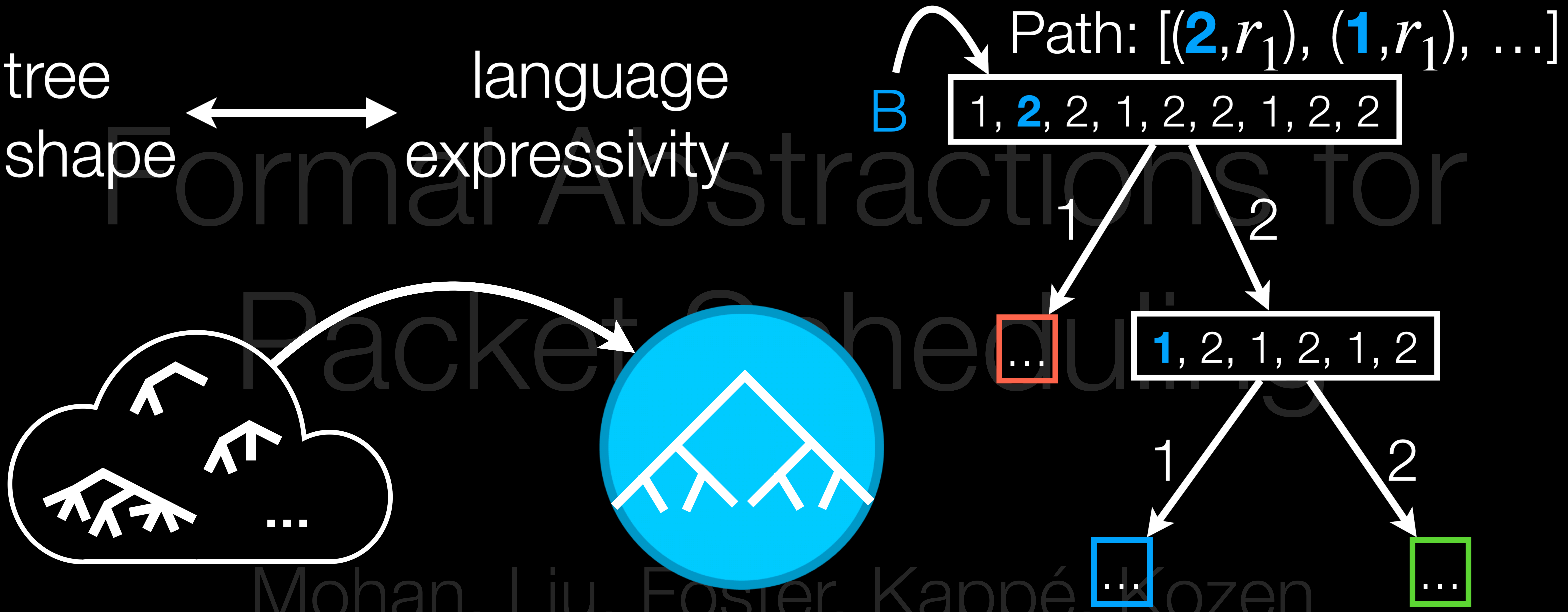
Formal Abstractions for Packet Scheduling

Mohan, Liu, Foster, Kappé, Kozen

cs.cornell.edu/~amohan

Formal Abstractions for Packet Scheduling

Mohan, Liu, Foster, Kappé, Kozen



cs.cornell.edu/~amohan