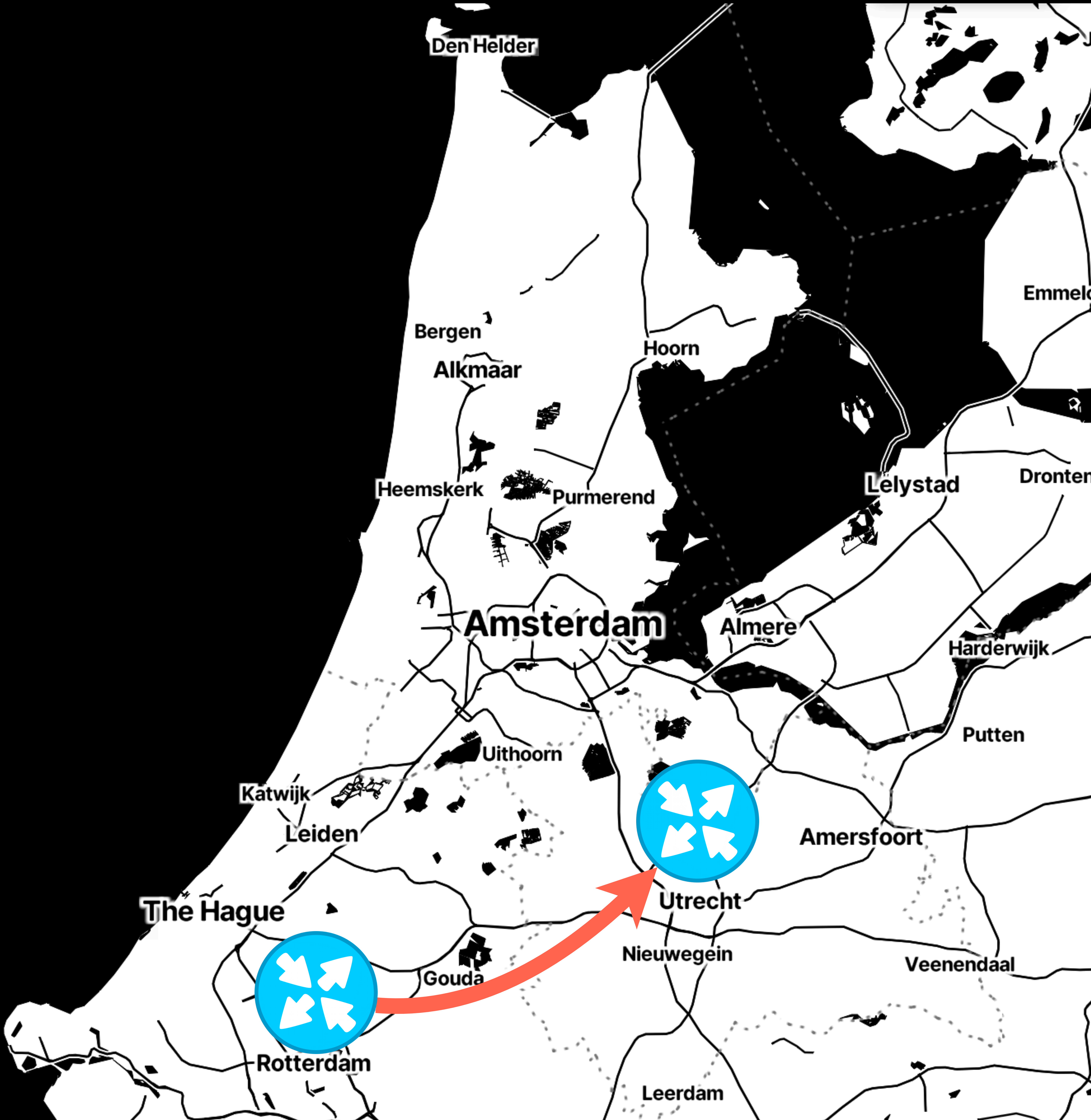
Formal Abstractions for Packet Scheduling

Mohan, Liu, Foster, Kappé, Kozen

SDN made networks programmable.

SDN made networks programmable.

Early goal: routing.



SDN made networks programmable.

Early goal: routing.

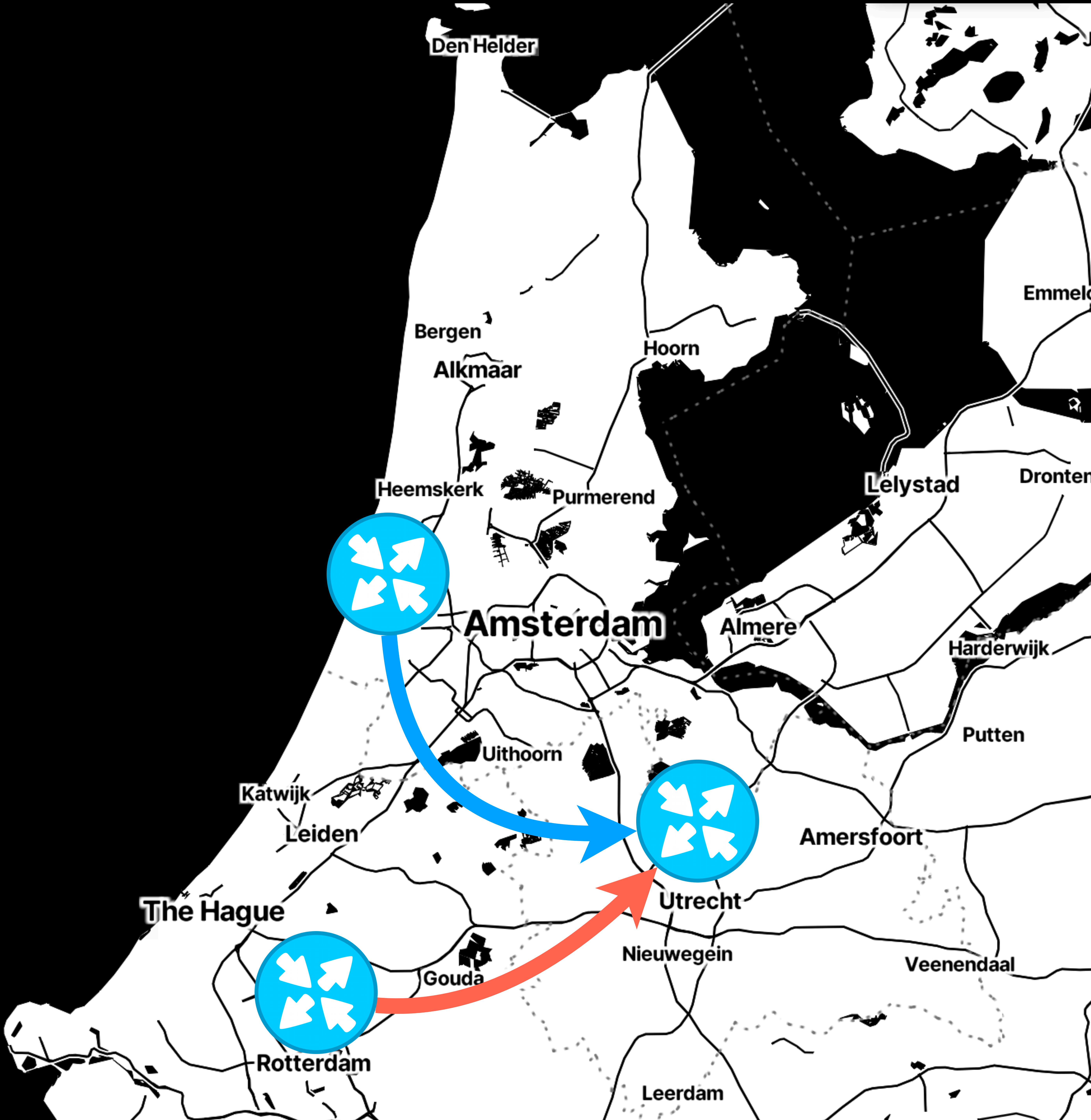
But now we need control over *scheduling.*

Den Helder
Bergen
Alkmaar
Hoorn
Emmel
Heemskerk
Purmerend
Lelystad
Dronten
Amsterdam
Almere
Harderwijk
Putten
Uithoorn
Katwijk
Leiden
Amersfoort
The Hague
Utrecht
Nieuwegein
Gouda
Veenendaal
Rotterdam
Leerdam

SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling*.

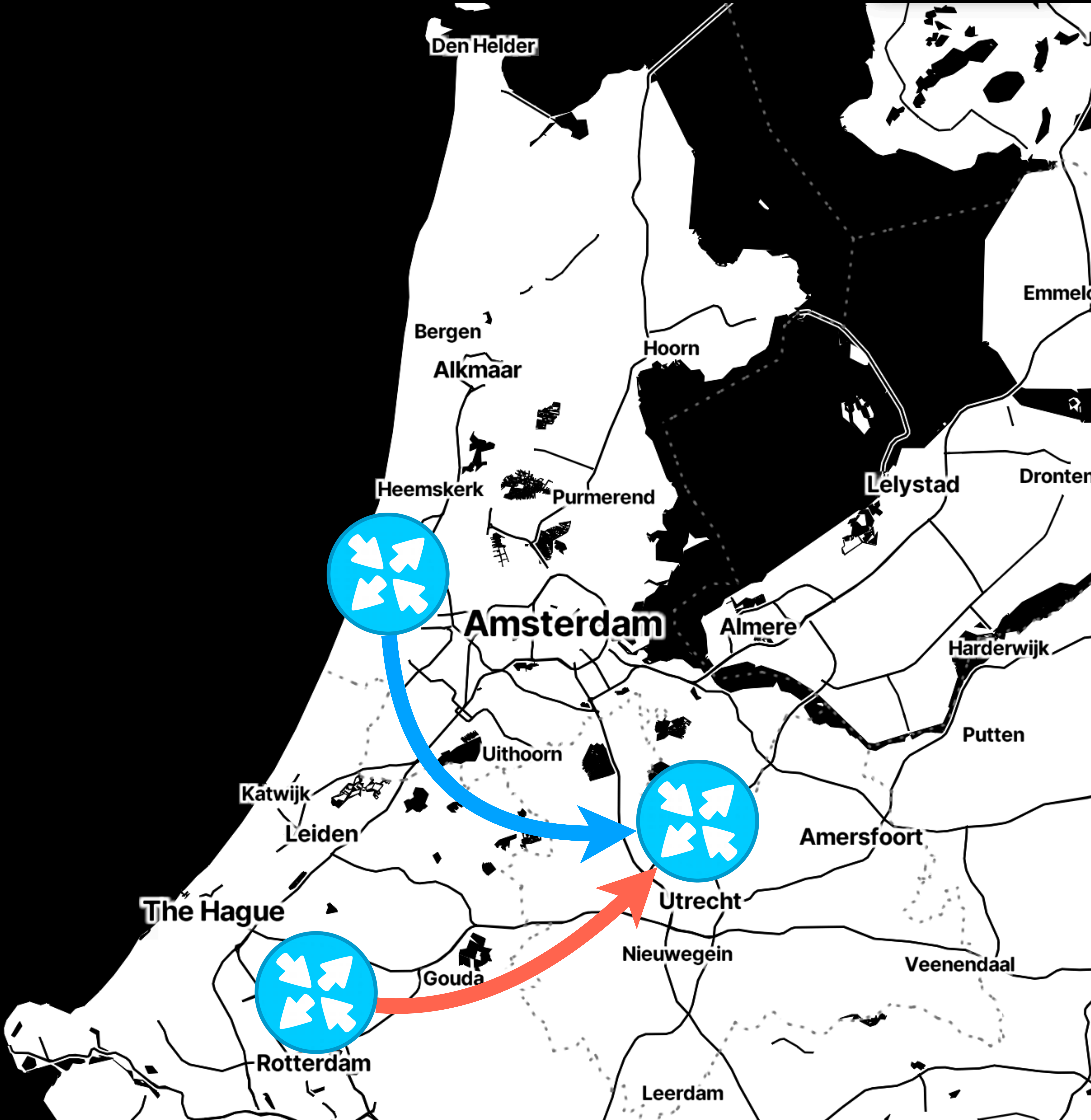


SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

Basic tools work fine…

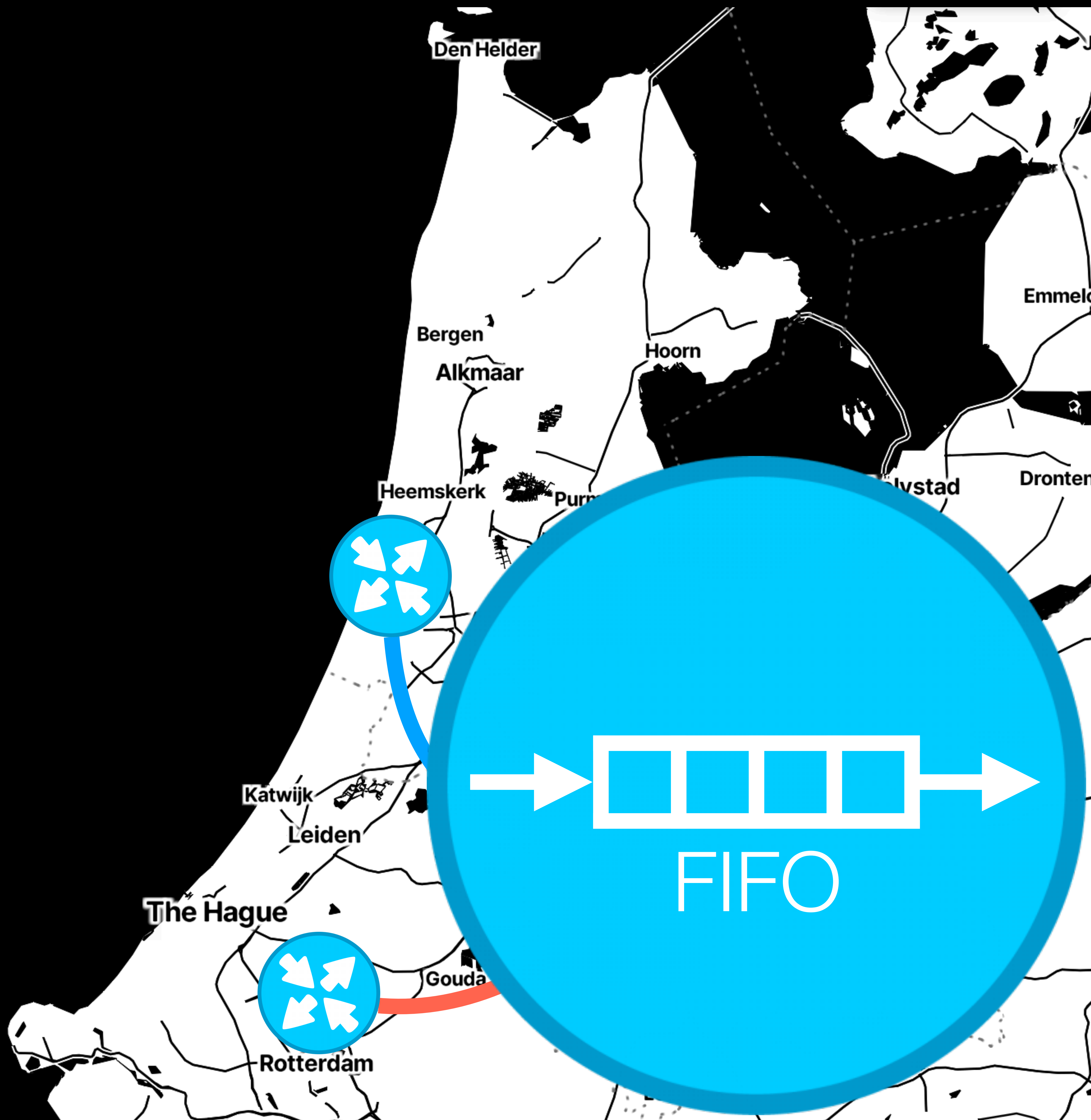


SDN made networks programmable.

Early goal: routing.

But now we need control over *scheduling.*

Basic tools work fine…



SDN made networks programmable.

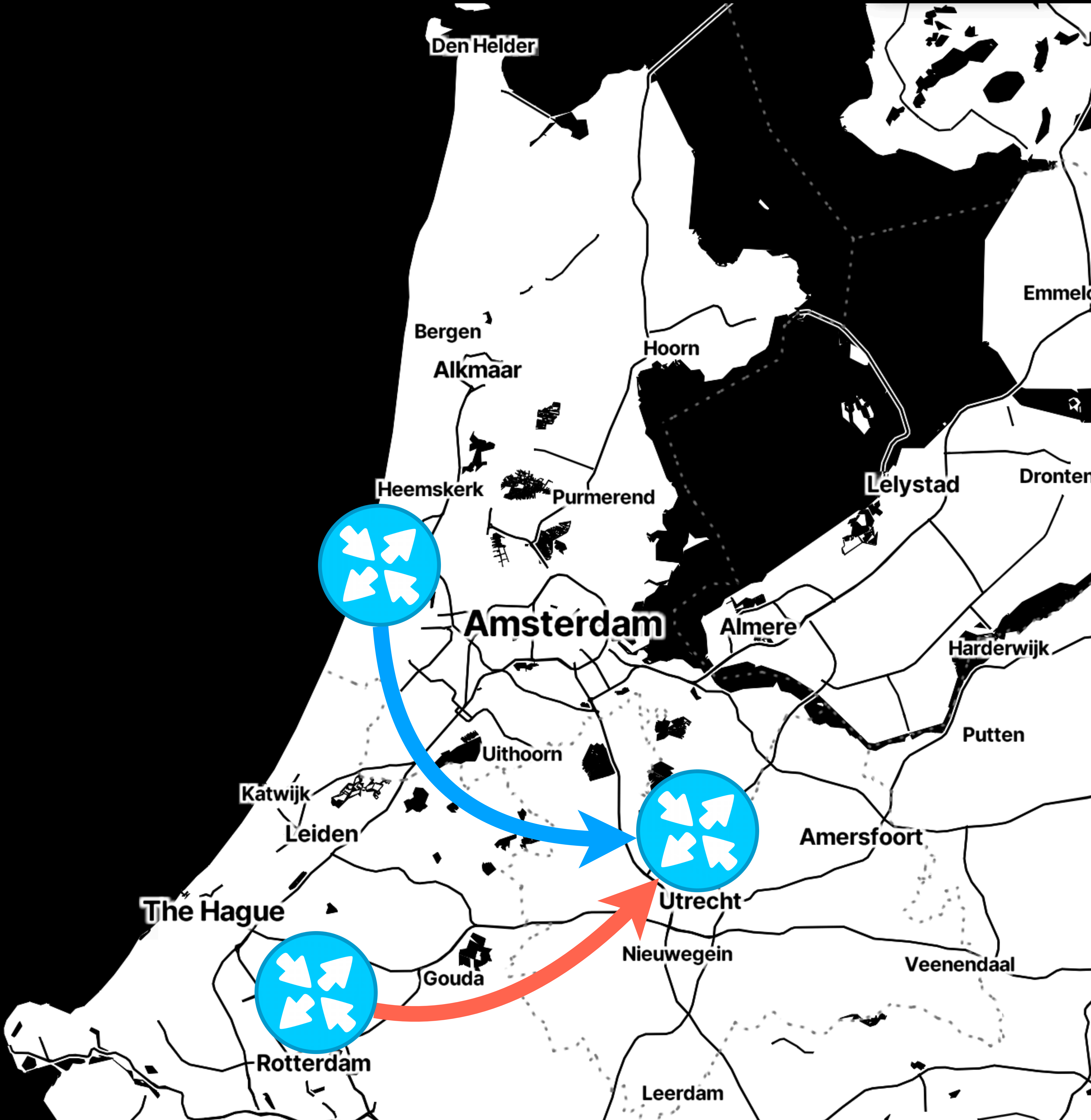
Early goal: routing.

But now we need control over *scheduling.*

Basic tools work fine…

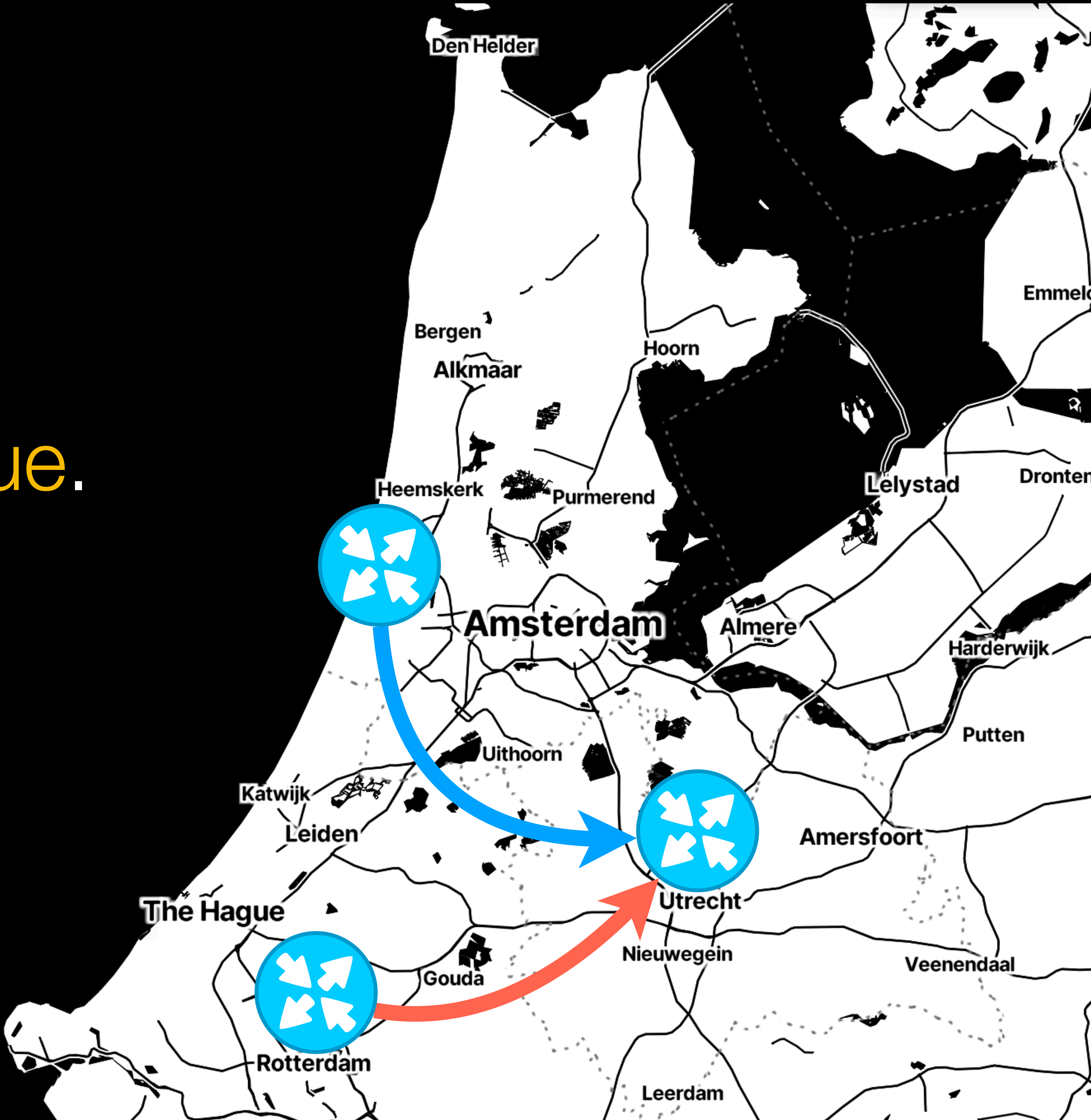
Den Helder
Bergen
Alkmaar
Hoorn
Emmel
Heemskerk
Pur
lystad
Dronten
Katwijk
Leiden
The Hague
Gouda
Rotterdam
PIFO

But modern scheduling requires more.



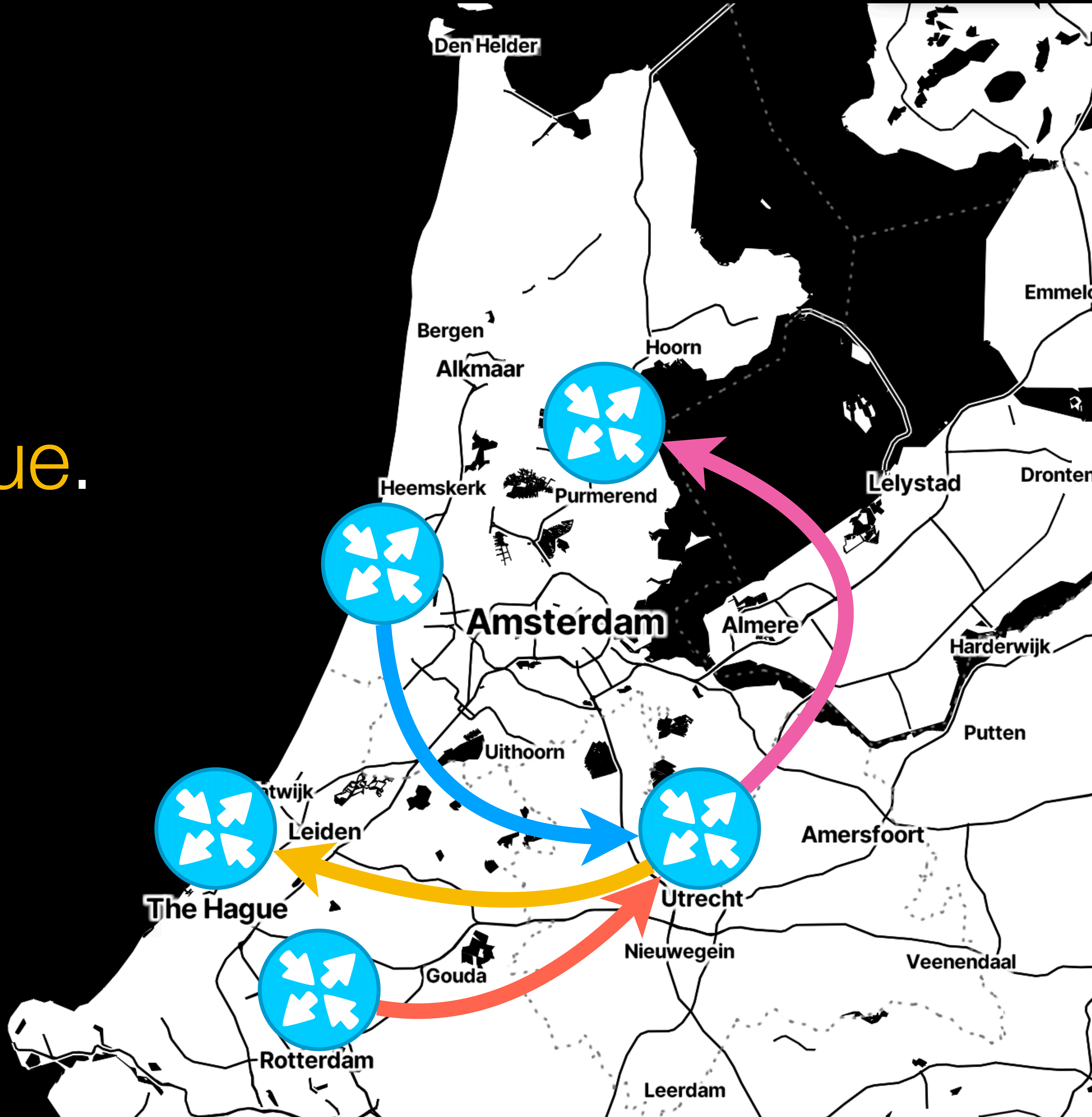
But modern scheduling requires more.

R traffic goes to either Purmerend or The Hague.



But modern scheduling requires more.

R traffic goes to either Purmerend or The Hague.

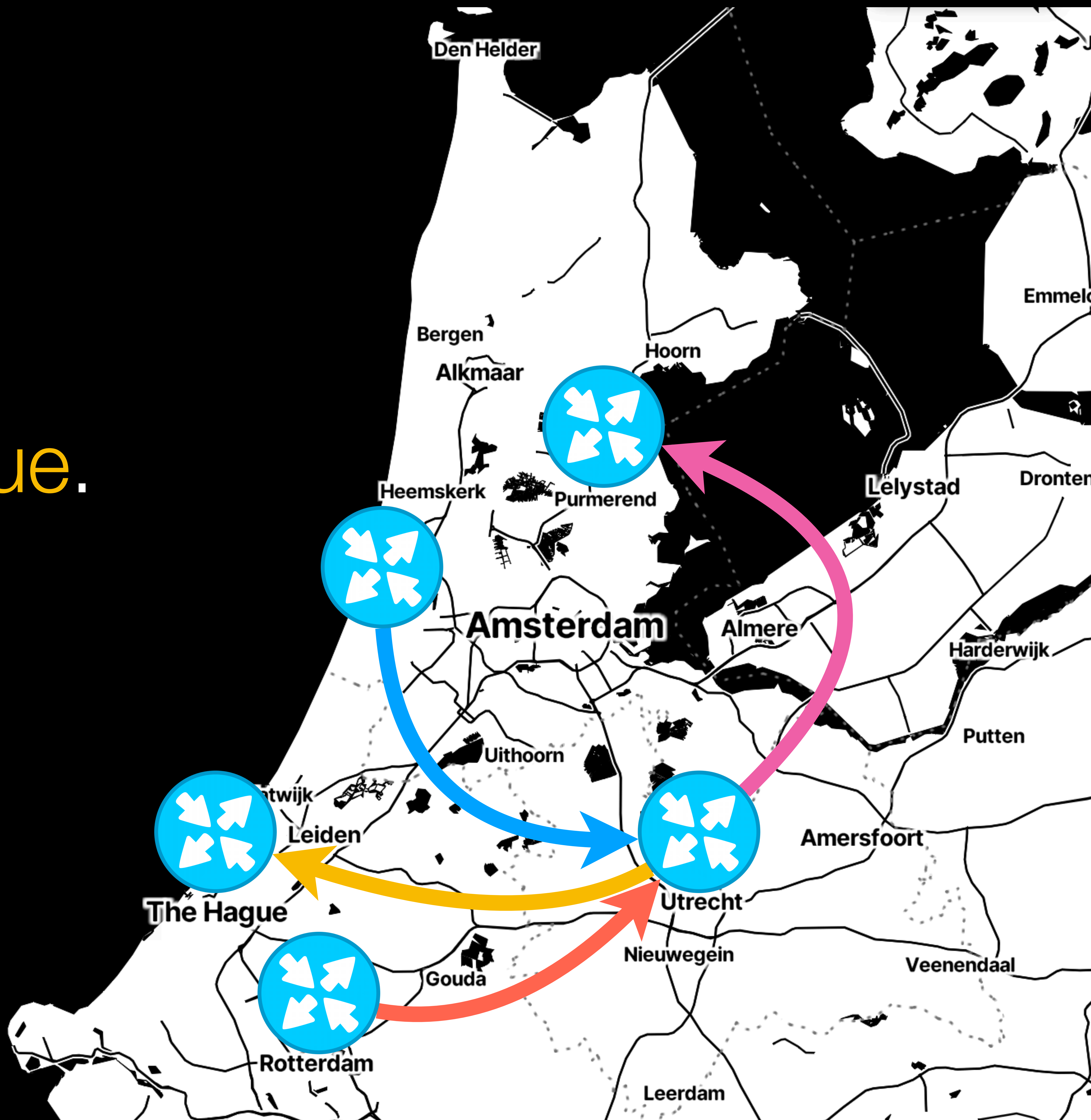


But modern scheduling requires more.

R traffic goes to either Purmerend or The Hague.

Goal:

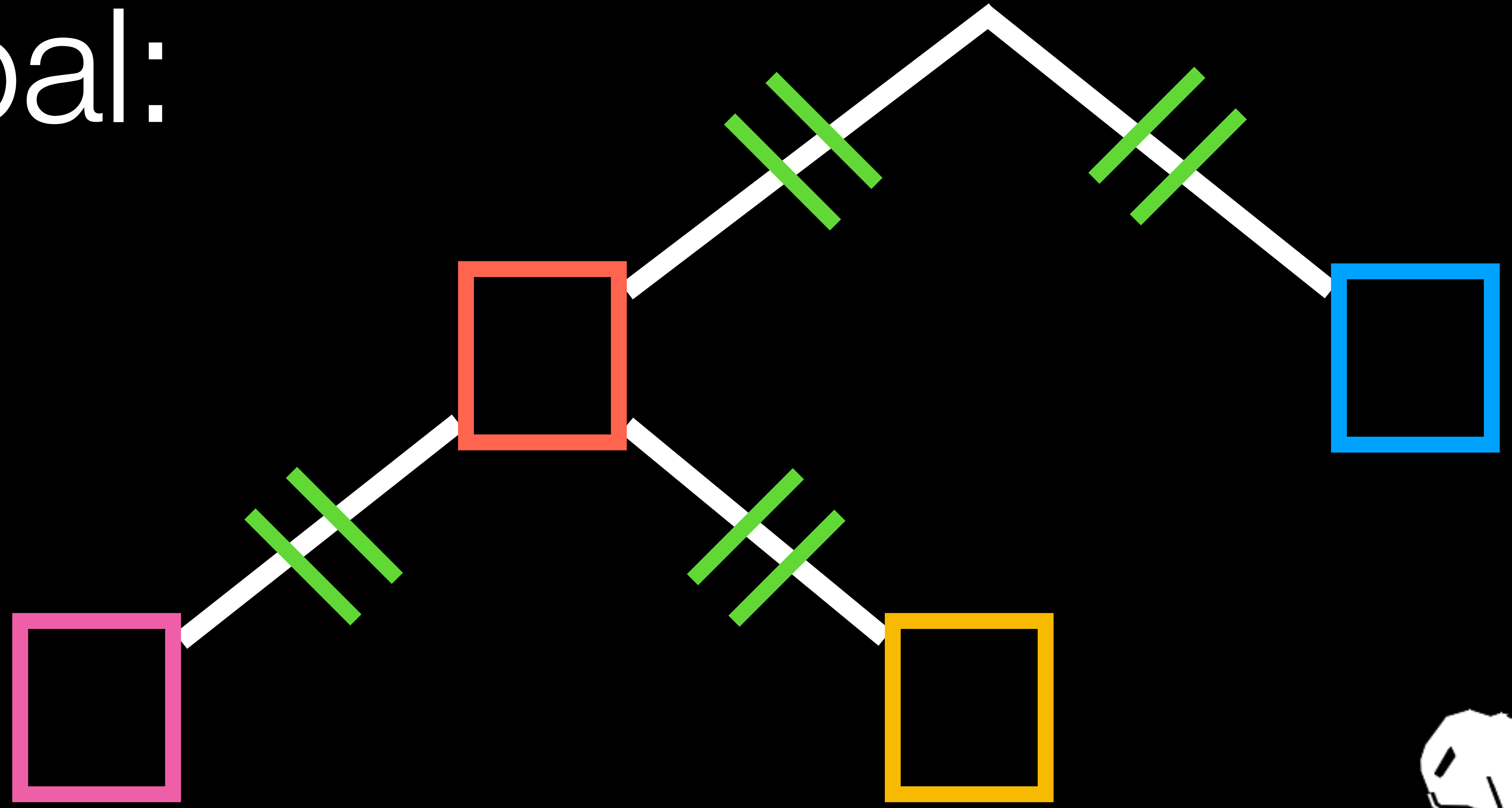
Interleave R and B;
interleave P and T.

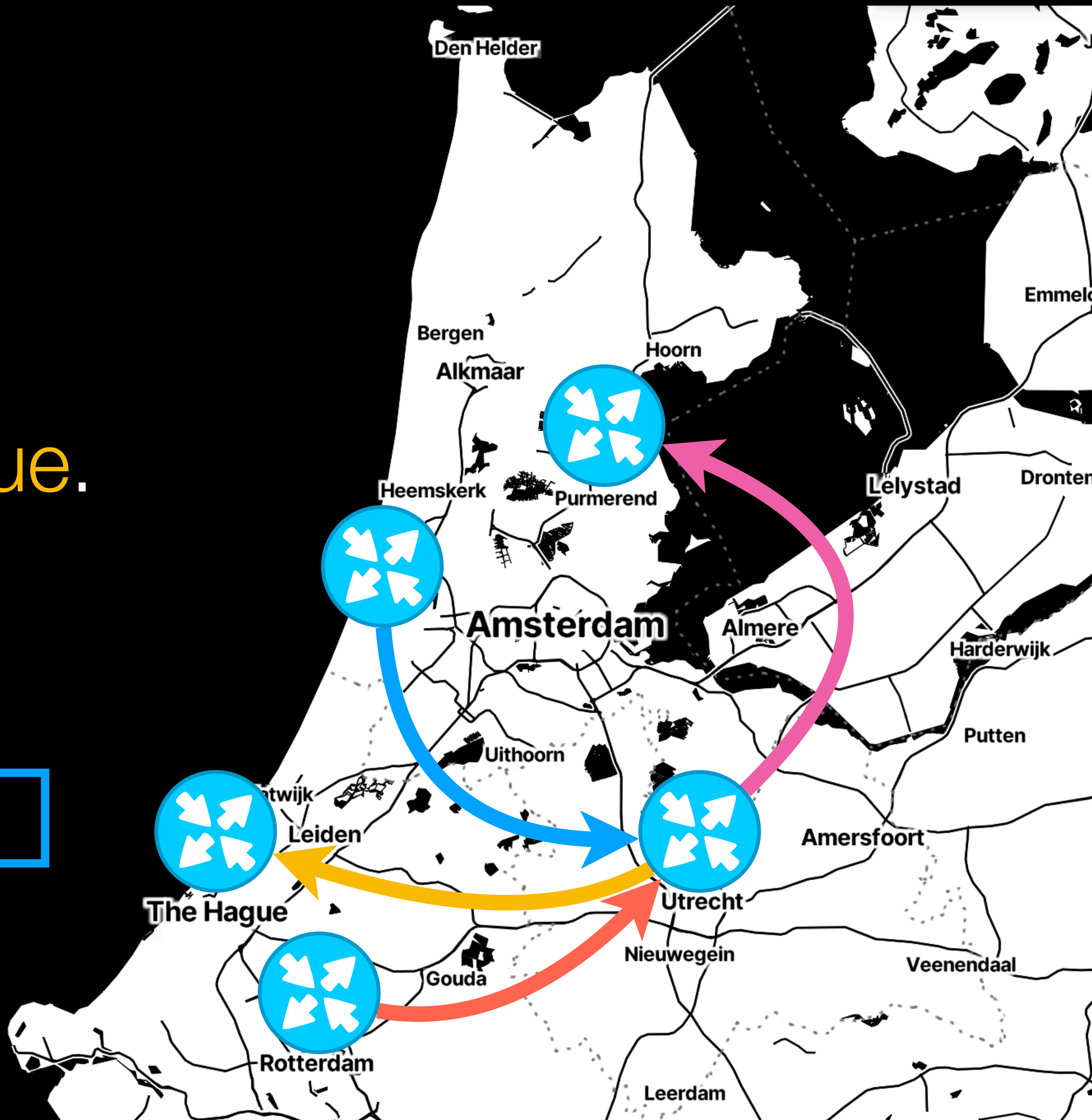


But modern scheduling requires more.

R traffic goes to either Purmerend or The Hague.

Goal:

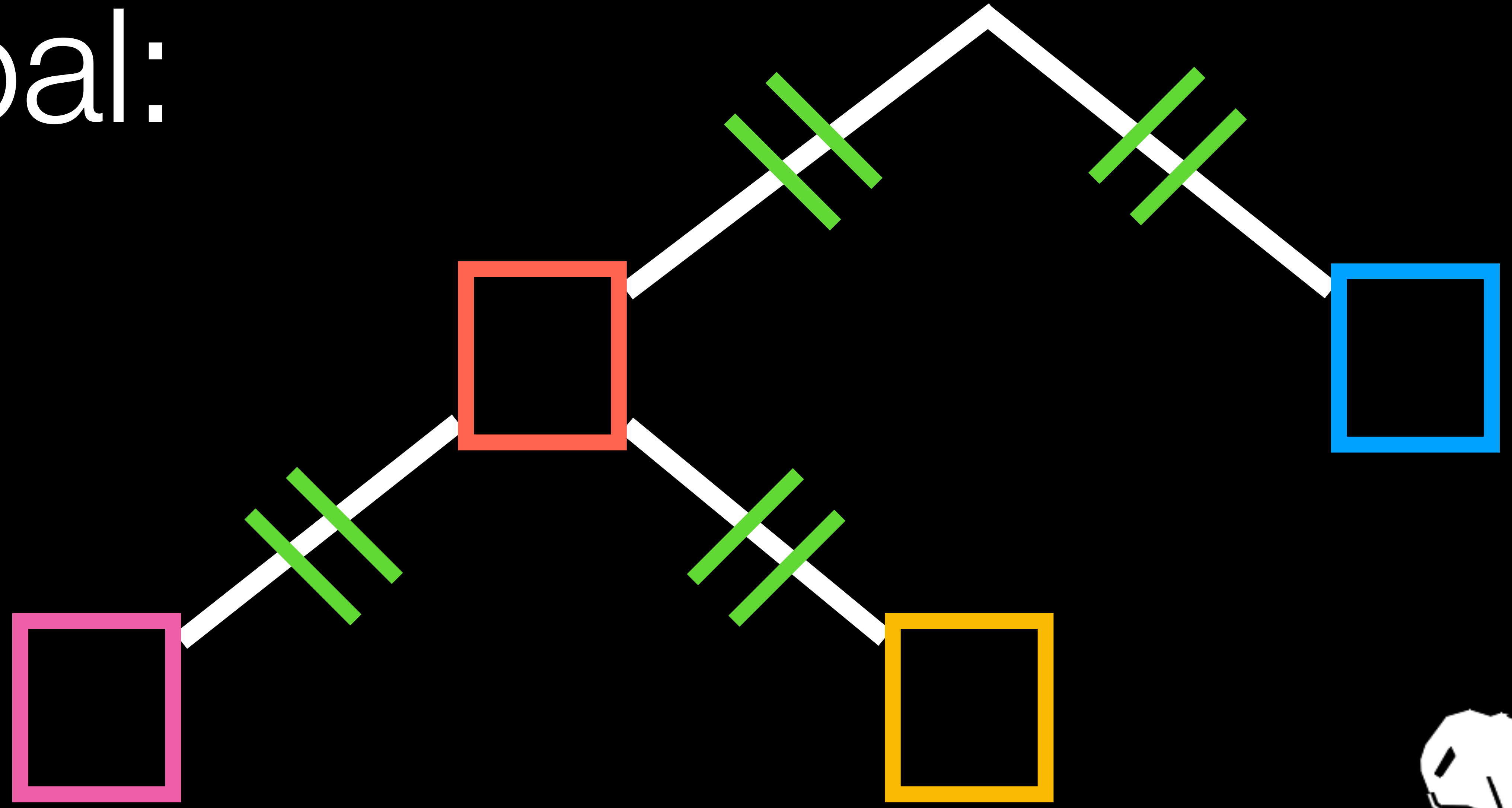




But modern scheduling requires more.

R traffic goes to either Purmerend or The Hague.

Goal:



Den Helder
Emmel
Bergen
Hoorn
Alkmaar
Heemskerk
Purm
lystad
Dronten
twijk
Leiden
The Hague
Gouda
Rotterdam
PIFO Tree

New plan!

Den Helder
Bergen
Alkmaar
Hoorn
Heemskerk
Purm
lystad
Dronten
Emmel
twijk
Leiden
The Hague
Gouda
Rotterdam
PIFO Tree

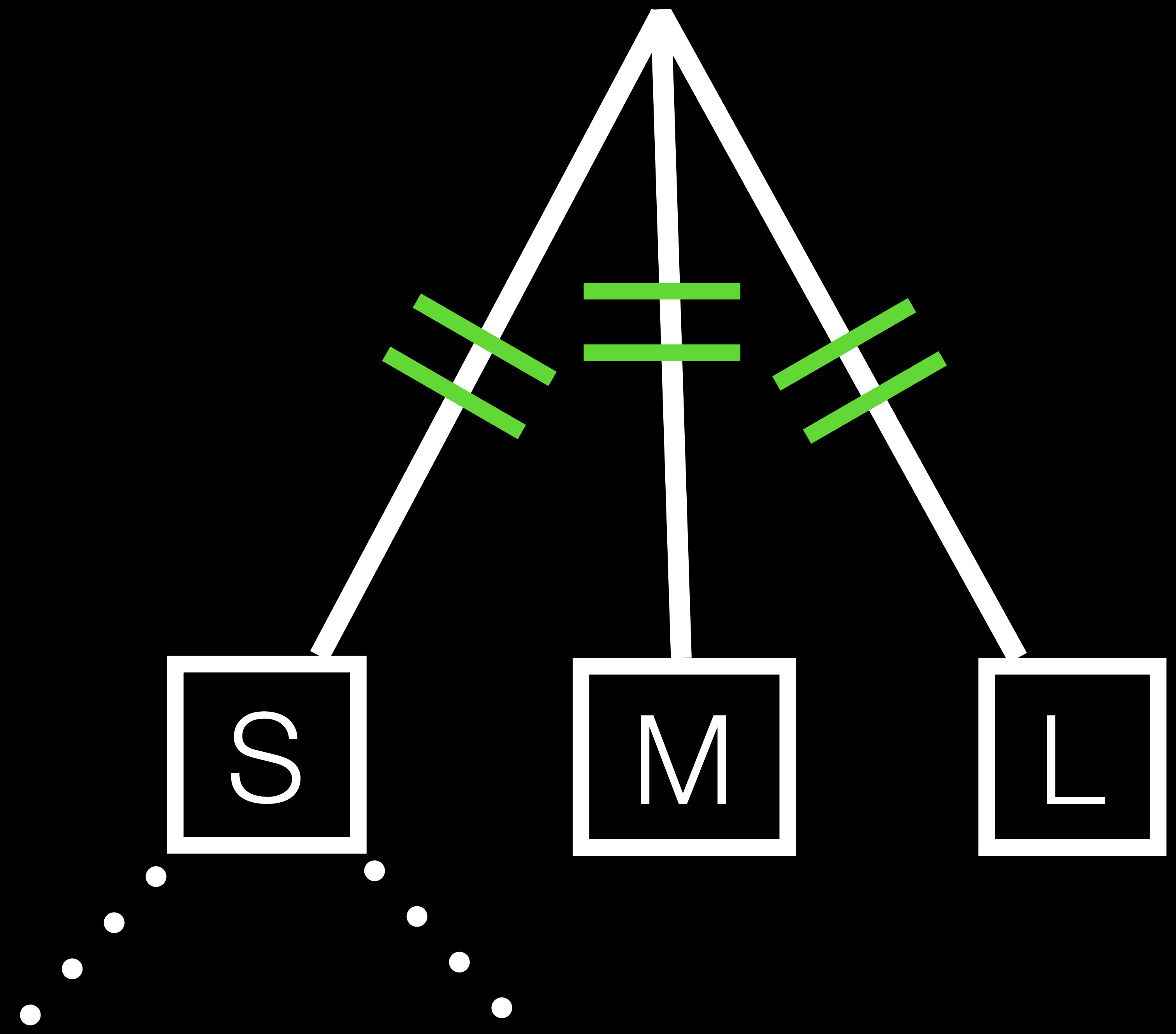
New plan!

Interleave small, medium, and large packets.

Den Helder
Emmelo
Bergen
Hoorn
Alkmaar
Heemskerk
Purm
lystad
Dronten
twijk
Leiden
The Hague
Gouda
Rotterdam
PIFO Tree

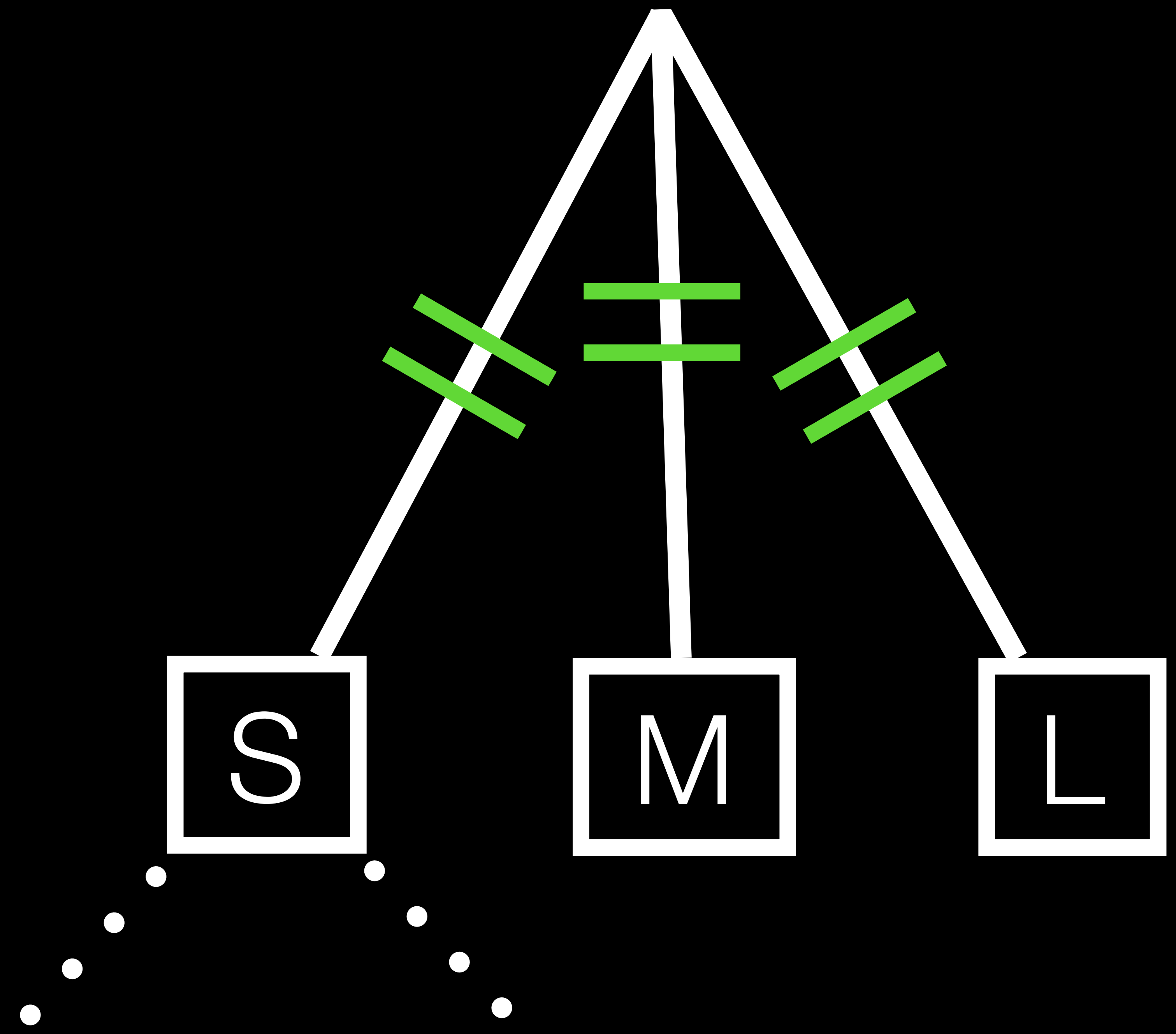
New plan!

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.

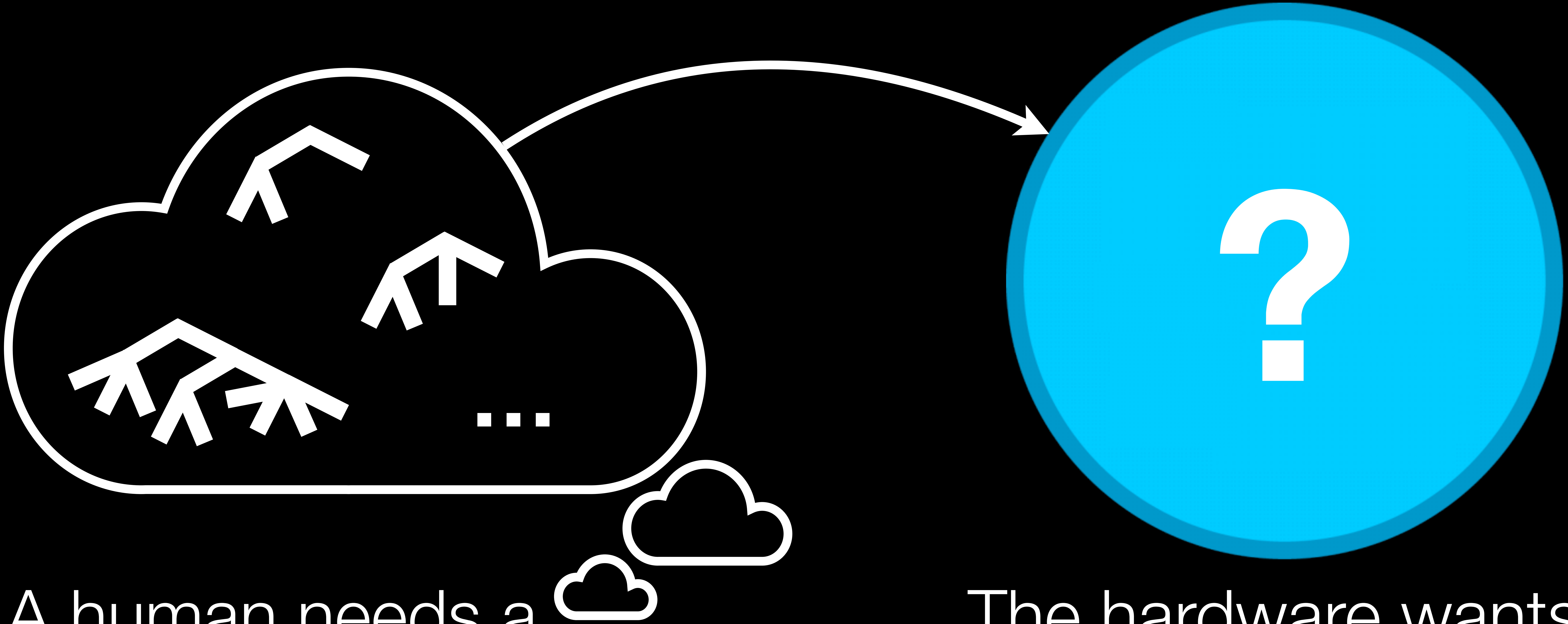
The hardware wants to support *one* tree.

No general way to deploy our gadget.

A human needs a *range* of trees.

The hardware wants to support *one* tree.

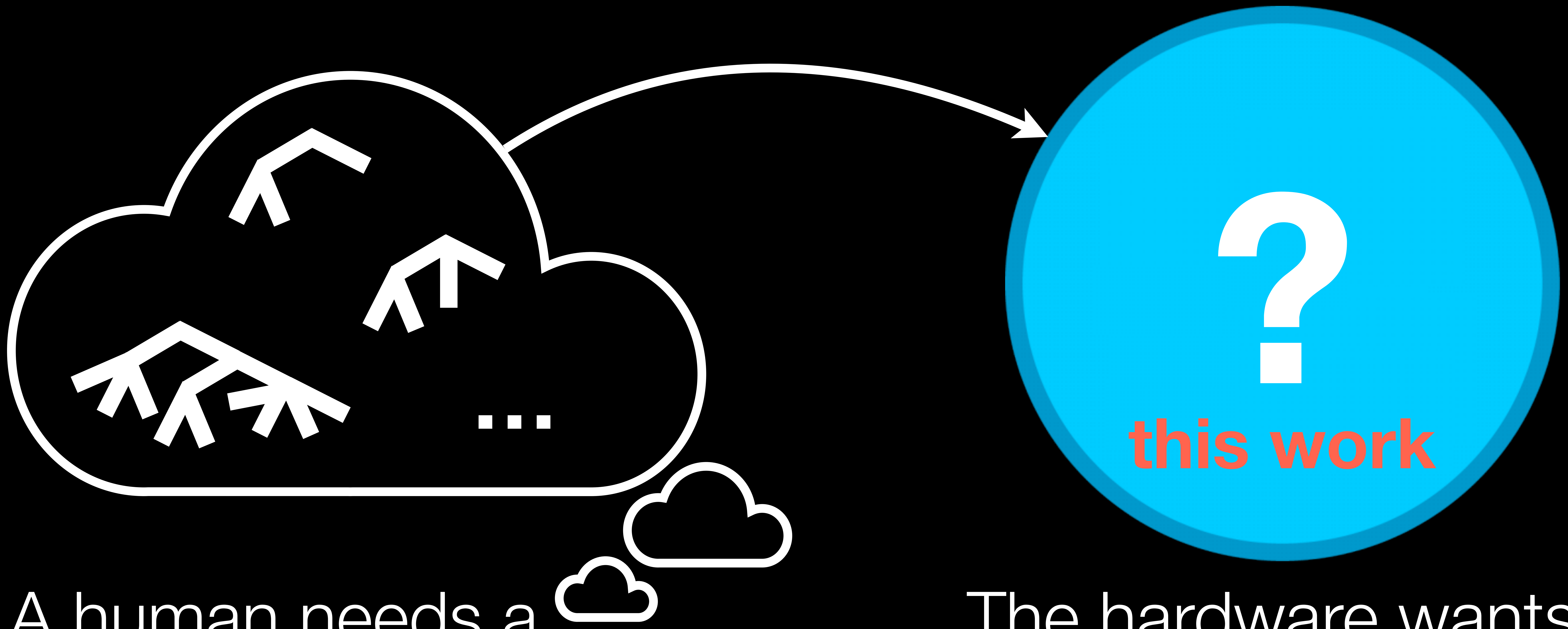
No general way to deploy our gadget.



A human needs a *range* of trees.

The hardware wants to support *one* tree.

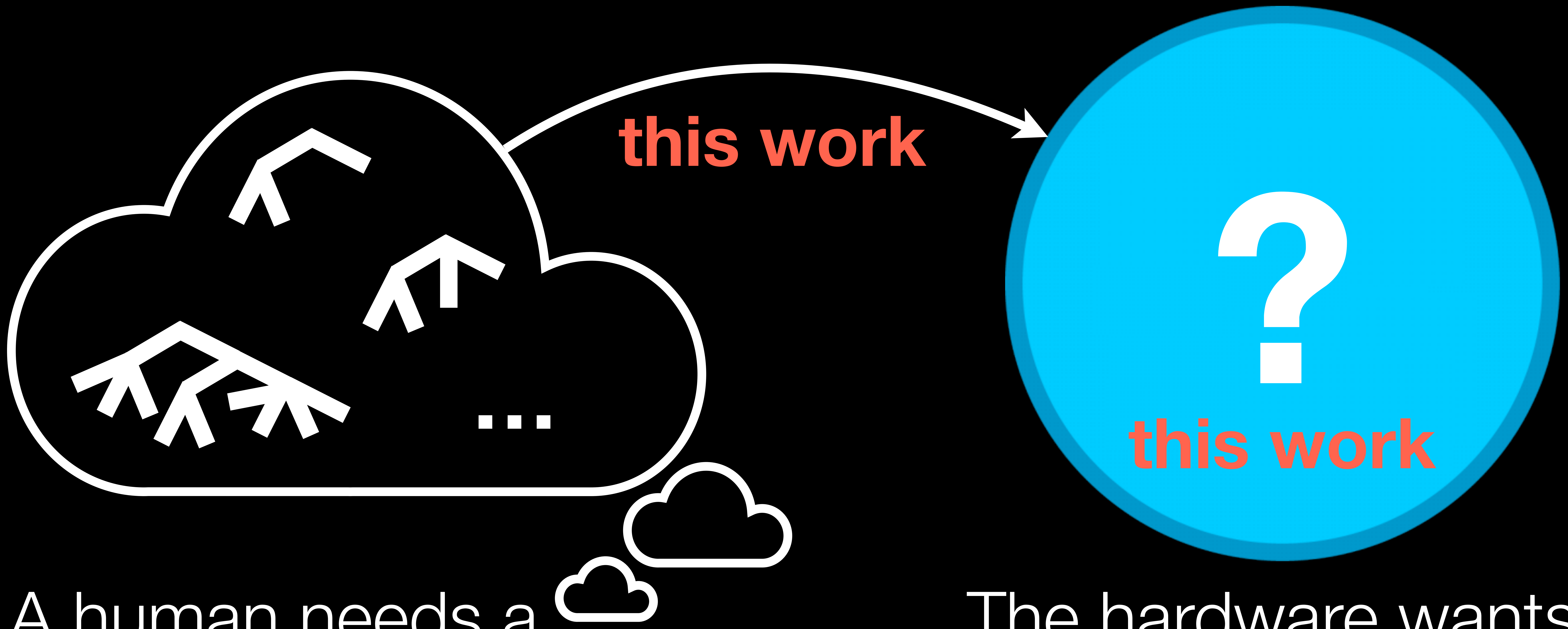
No general way to deploy our gadget.



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.

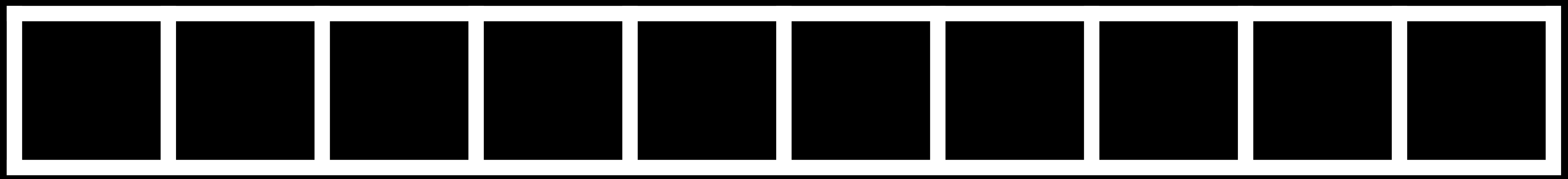
The hardware wants to support *one* tree.

Aside: PIFO Trees

Sivaraman et al. at SIGCOMM ’16

Review: FIFO

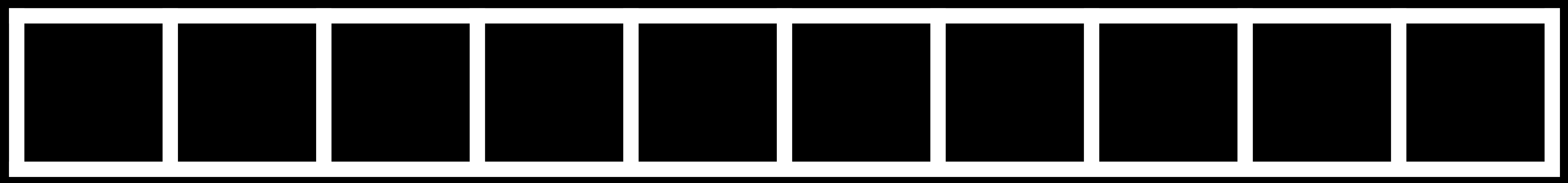
Just an ordered collection.



Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

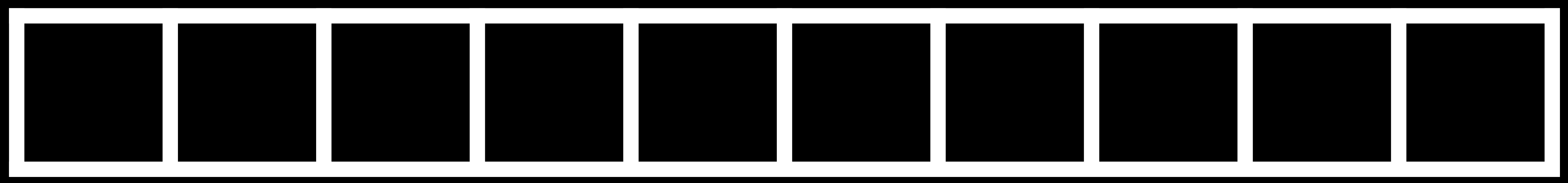


Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push

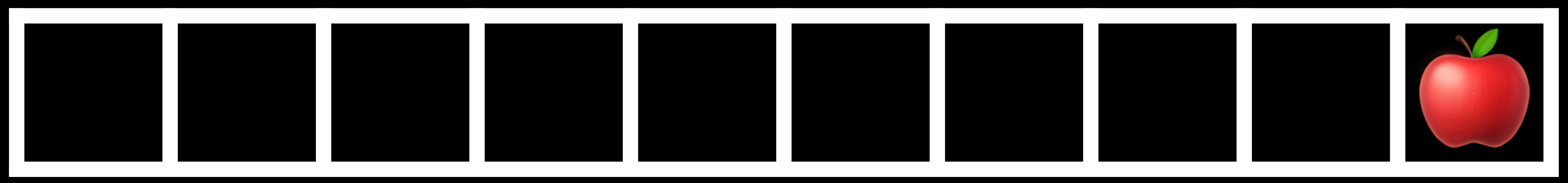


Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push

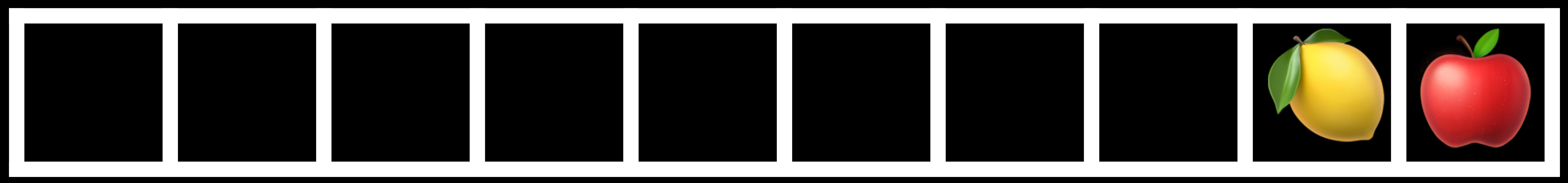


Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push



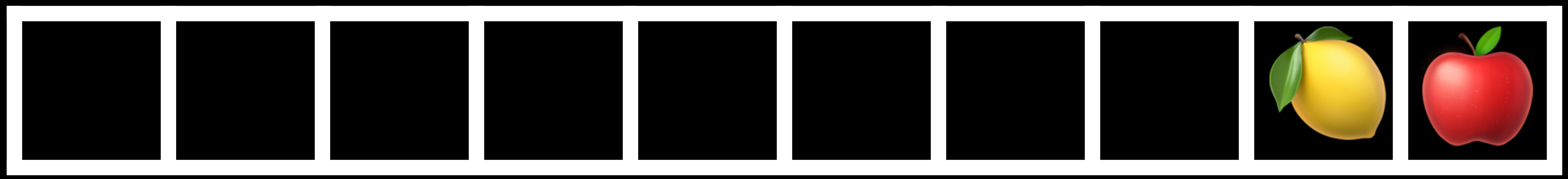
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push

pop



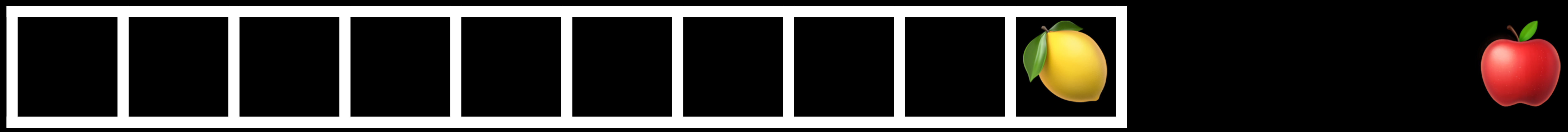
Review: FIFO

Just an ordered collection.

Two ways of interacting with the collection:

push

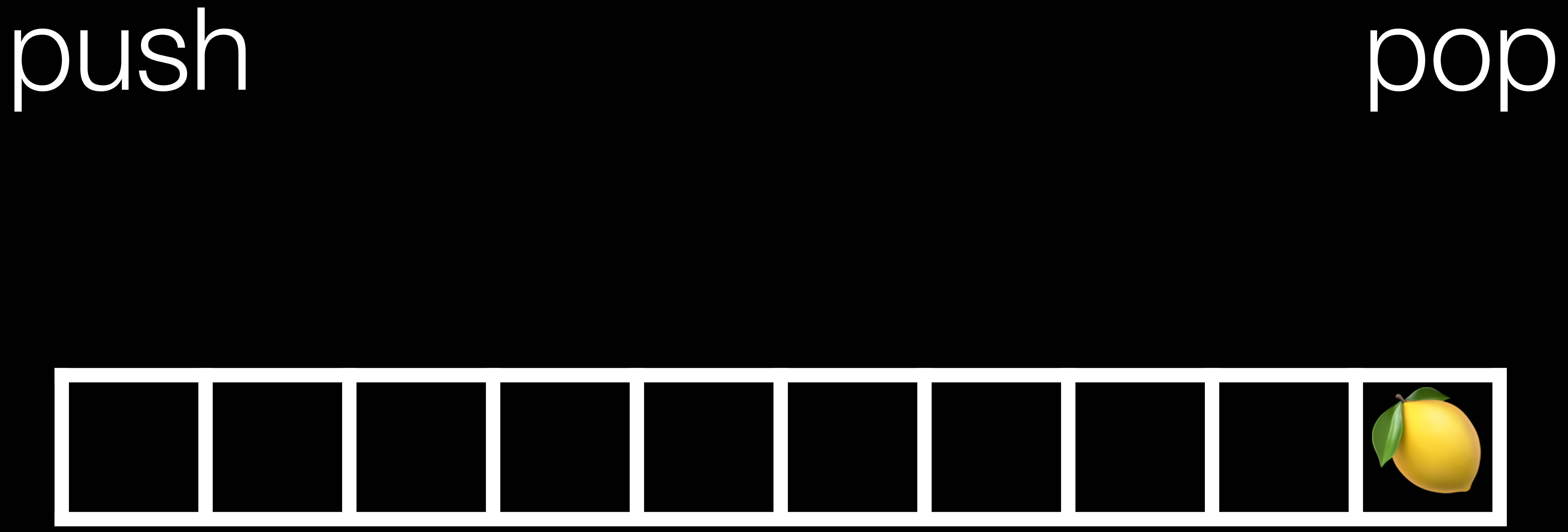
pop



Review: FIFO

Just an ordered collection.

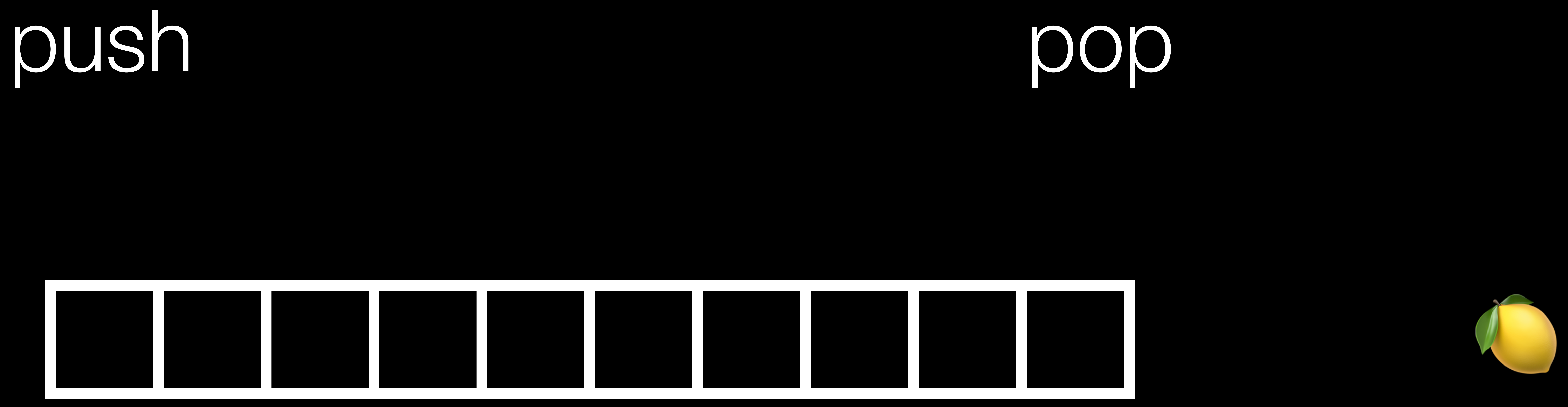
Two ways of interacting with the collection:



Review: FIFO

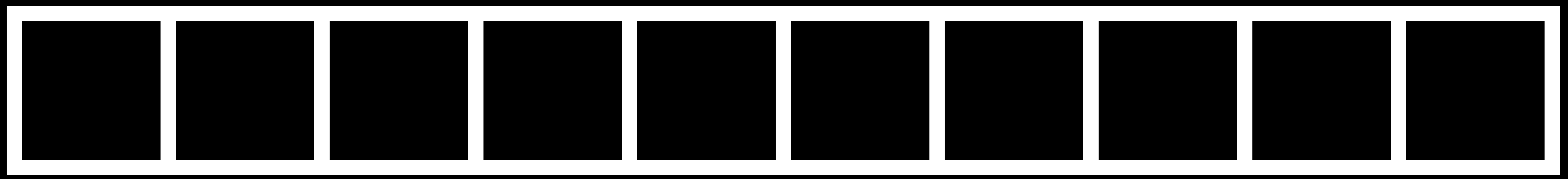
Just an ordered collection.

Two ways of interacting with the collection:



Review: priority queue

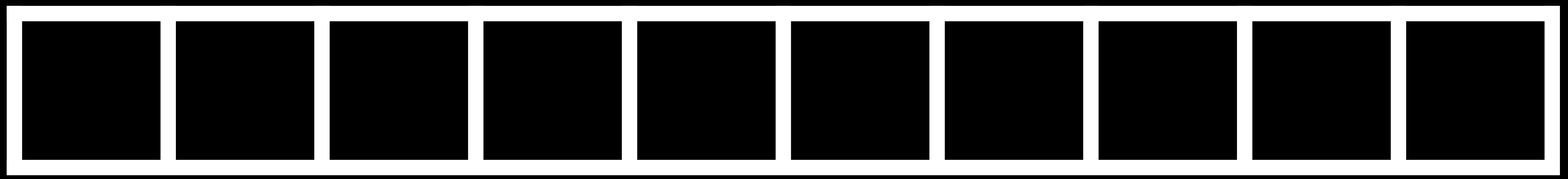
Everything from before holds,
but we have a little more control.



Review: priority queue

Everything from before holds,
but we have a little more control.

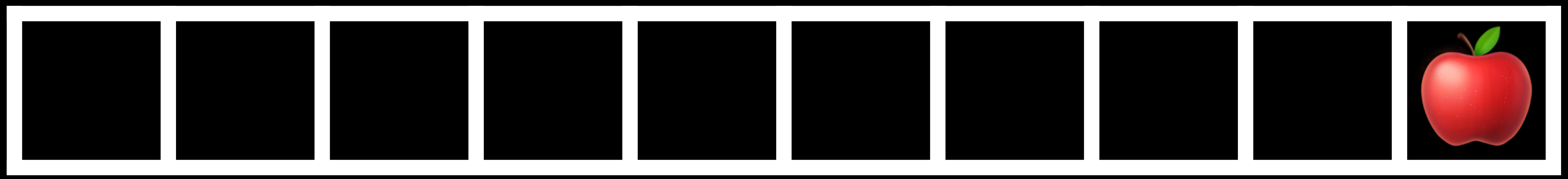
Say we have a queue *prioritized by pH.*



Review: priority queue

Everything from before holds,
but we have a little more control.

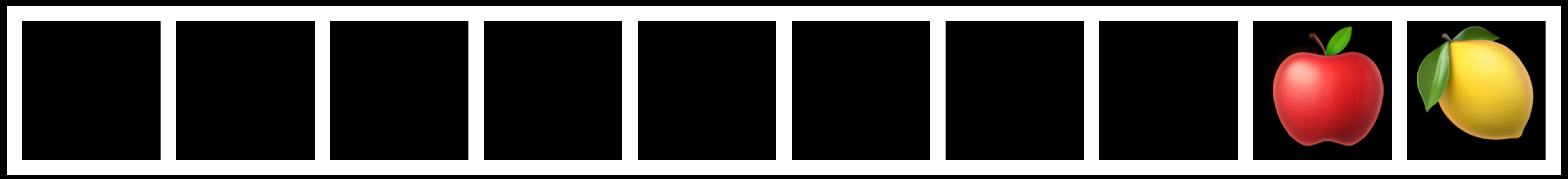
Say we have a queue *prioritized by pH.*



Review: priority queue

Everything from before holds,
but we have a little more control.

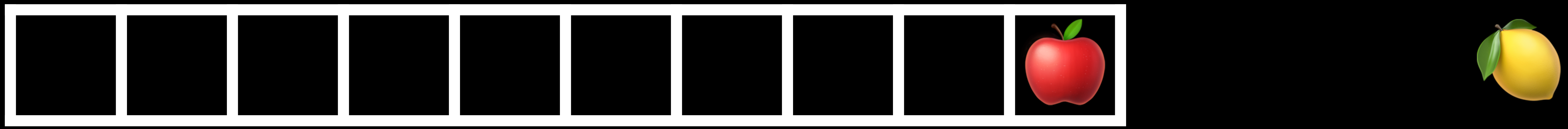
Say we have a queue *prioritized by pH.*



Review: priority queue

Everything from before holds,
but we have a little more control.

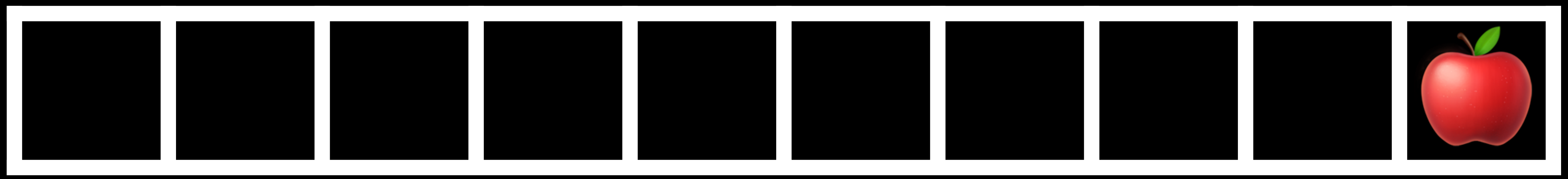
Say we have a queue *prioritized by pH.*



Review: priority queue

Everything from before holds,
but we have a little more control.

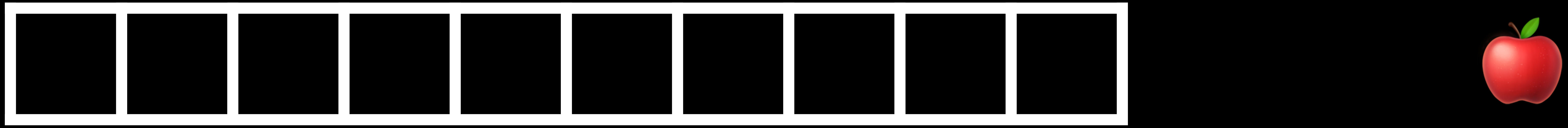
Say we have a queue *prioritized by pH.*



Review: priority queue

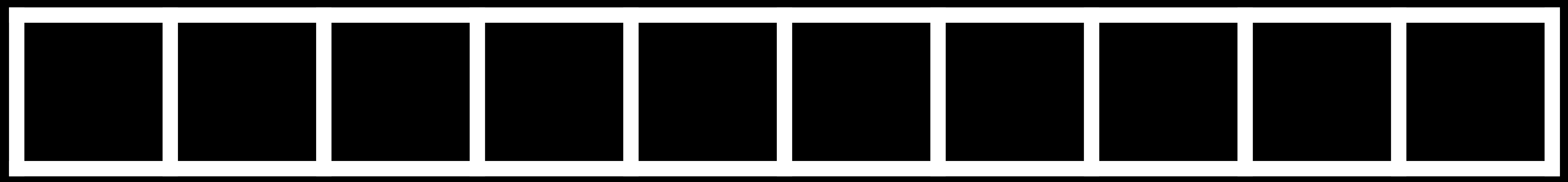
Everything from before holds,
but we have a little more control.

Say we have a queue *prioritized by pH.*



Review: priority queue

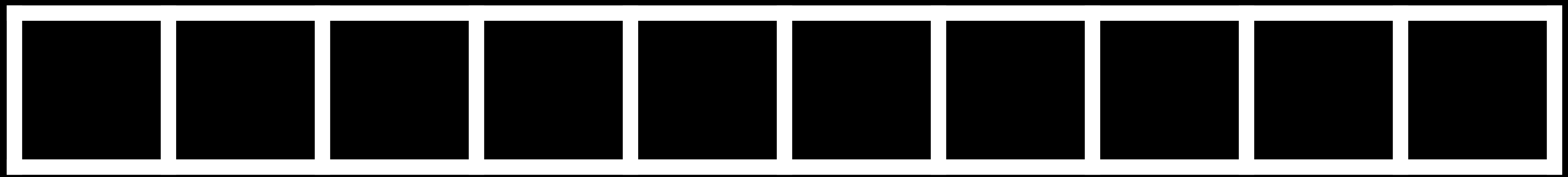
The priority need not be inherent to the item!



Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*



Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*

Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*

Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*

Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*

Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*

Review: priority queue

The priority need not be inherent to the item!

We can have a *ranking function:*

Review: priority queue

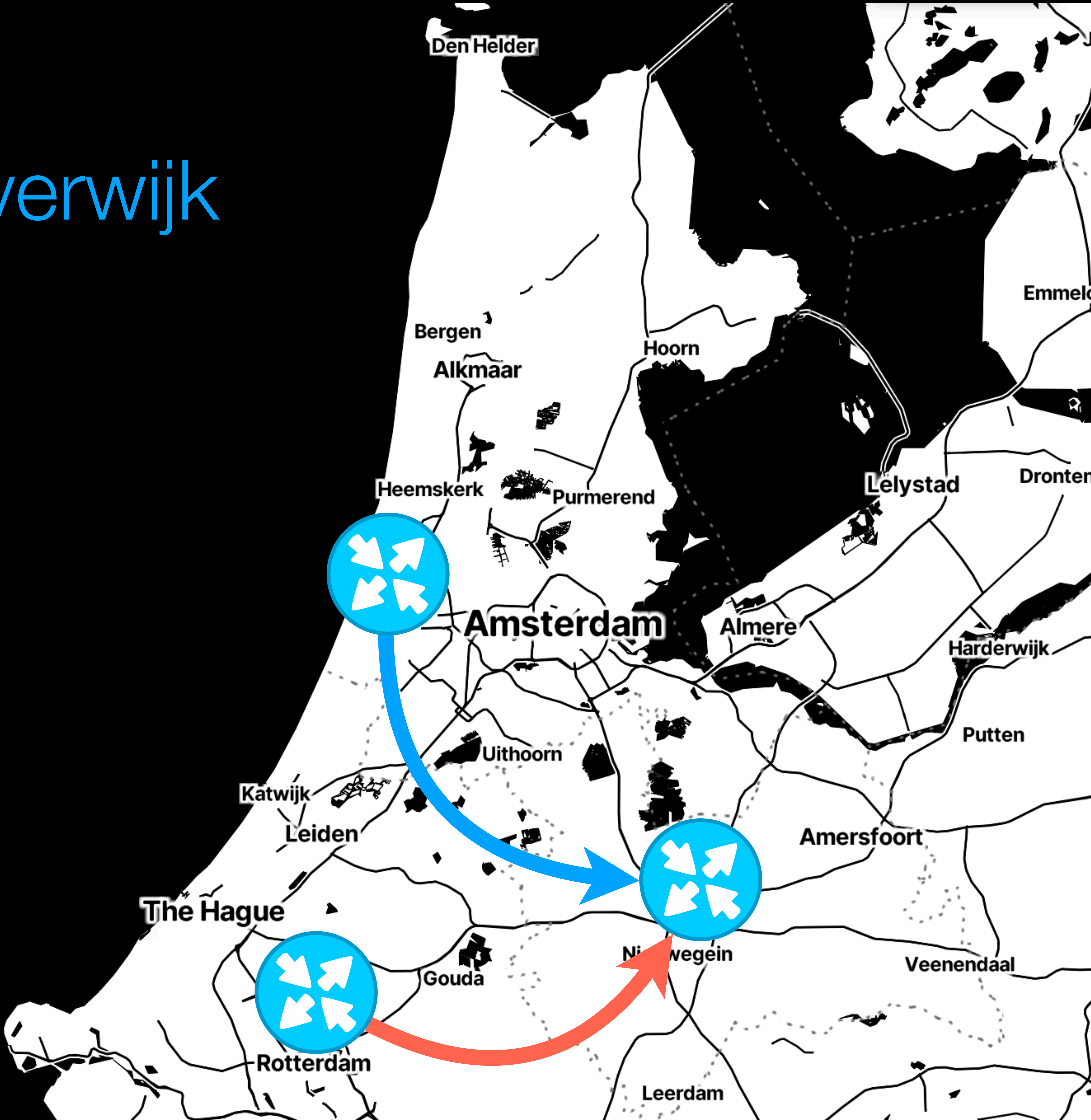
The priority need not be inherent to the item!

We can have a *ranking function:*

Introducing: PIFO

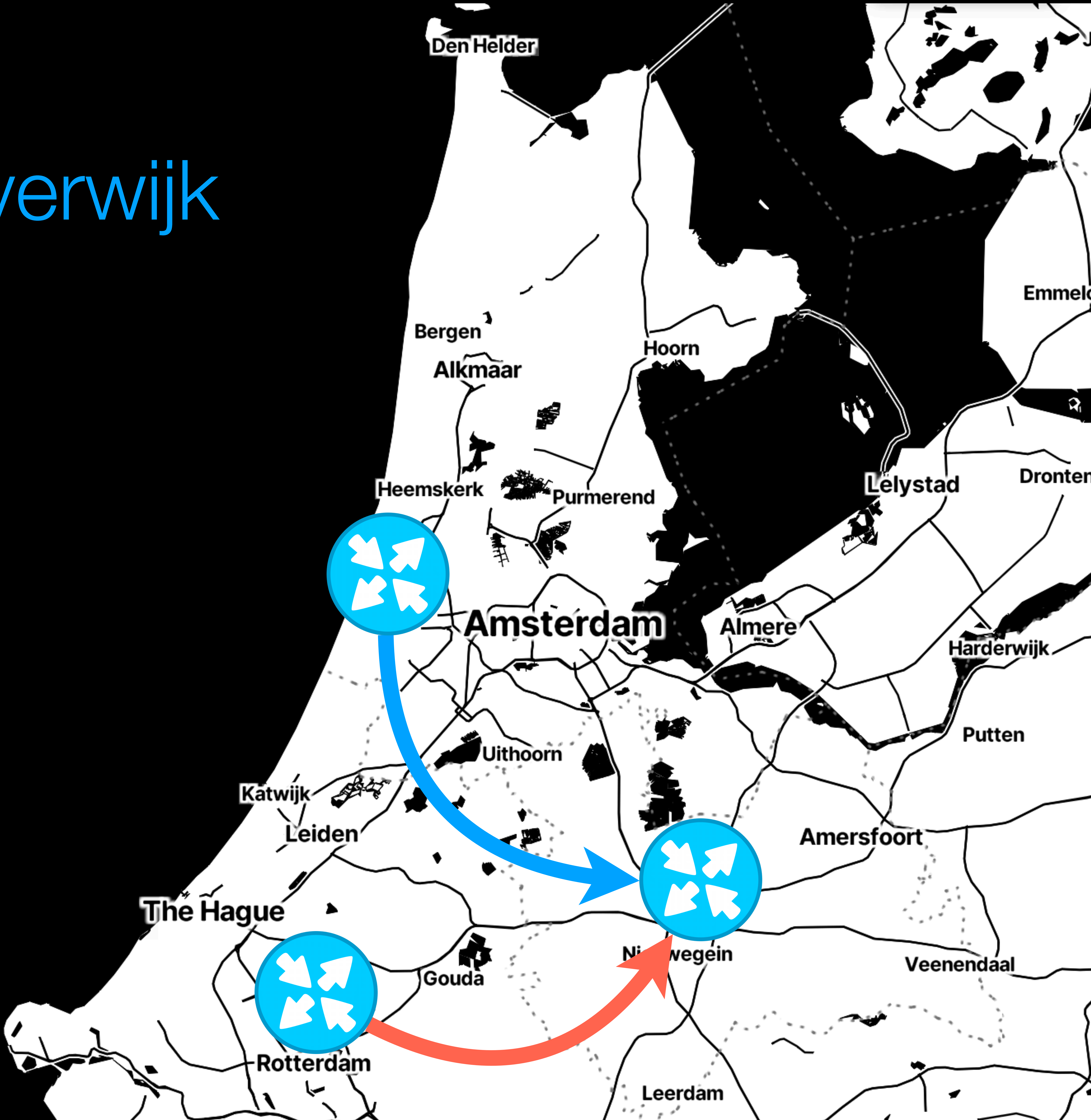
Just a PQ, with a ranking function,
but with *rank-ties* broken in FIFO order.

Traffic incoming from Rotterdam and Beverwijk



Traffic incoming from Rotterdam and Beverwijk

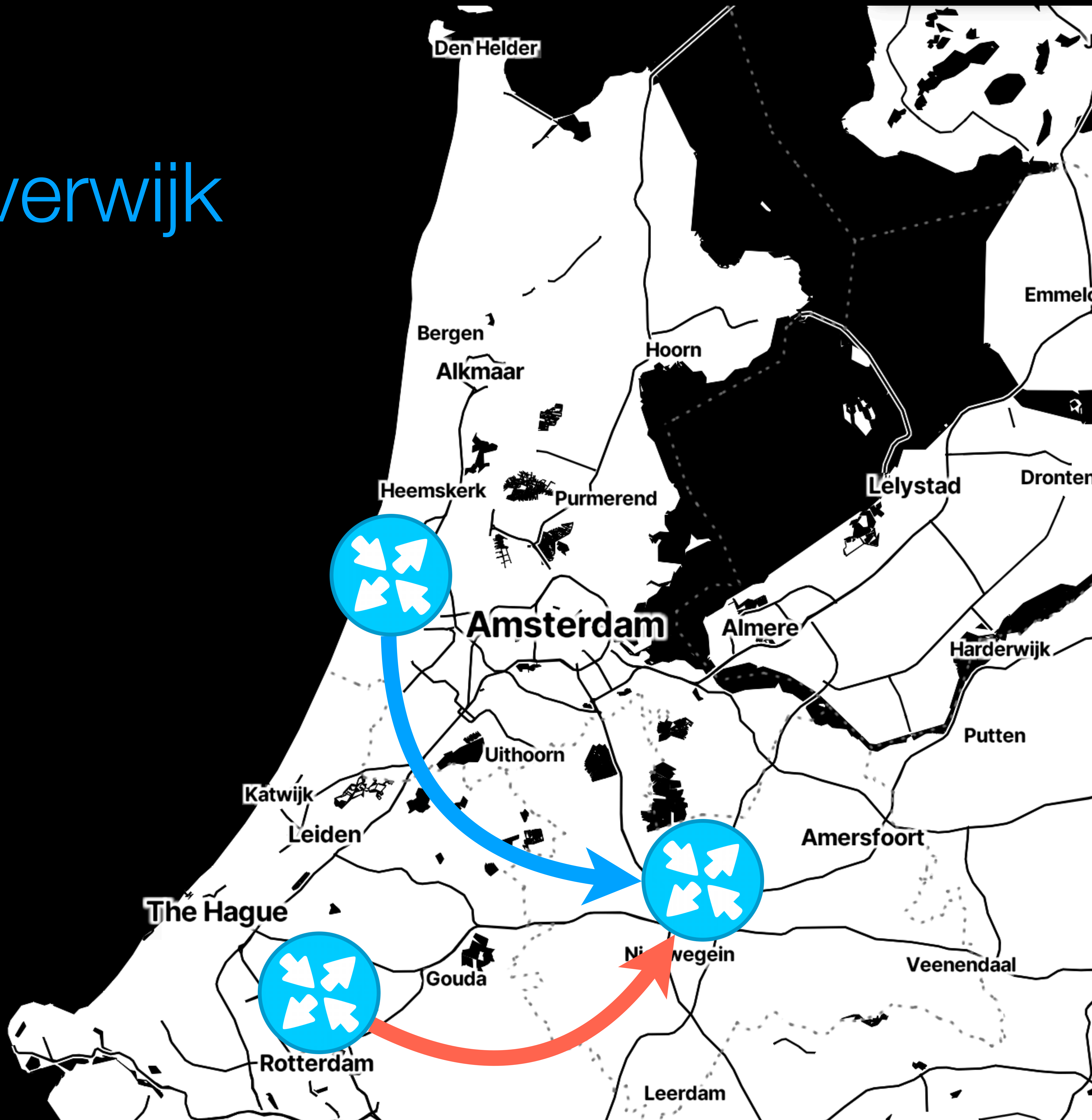
Goal: interleave R and B



Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

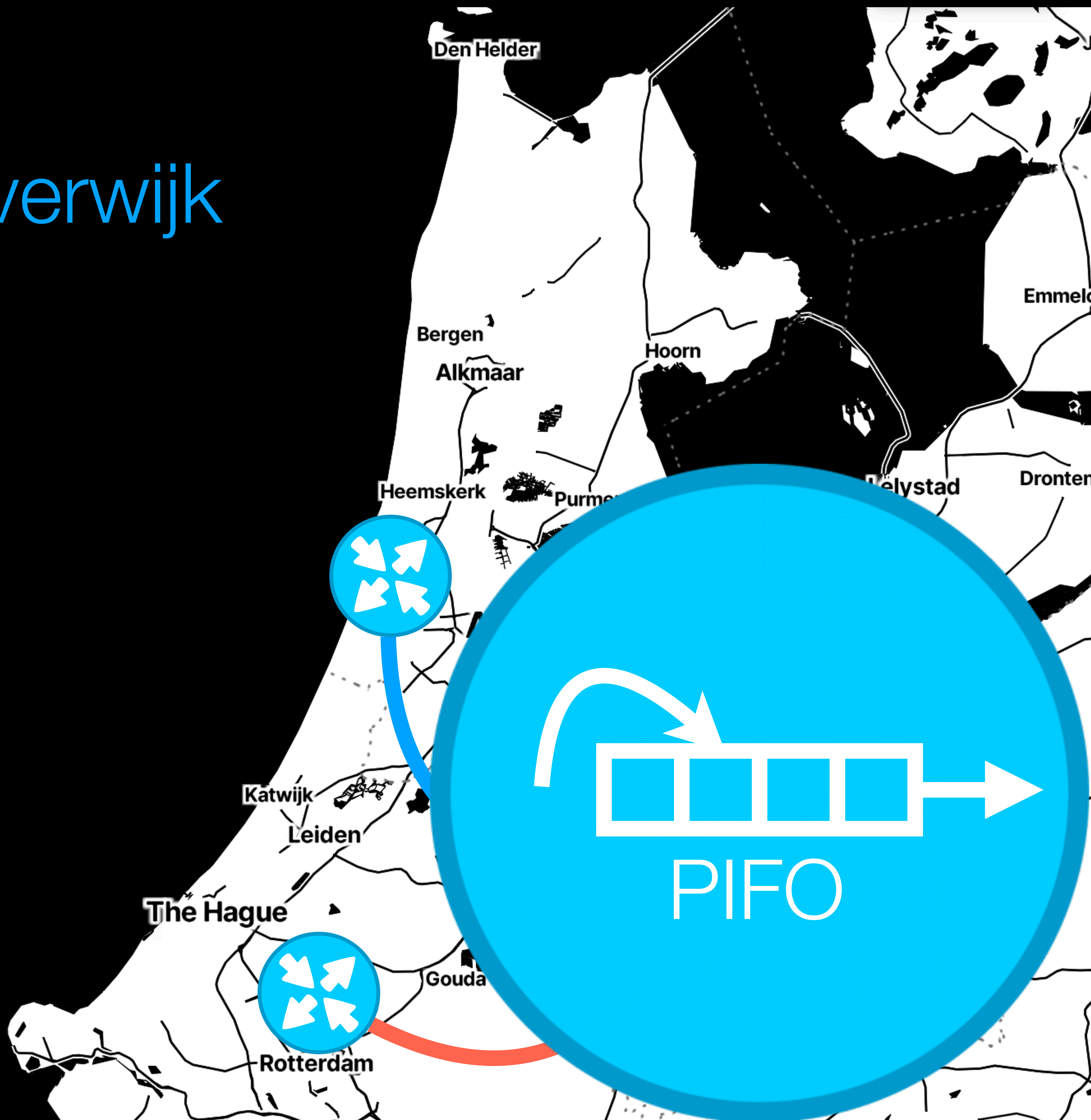
A PIFO will suffice.



Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

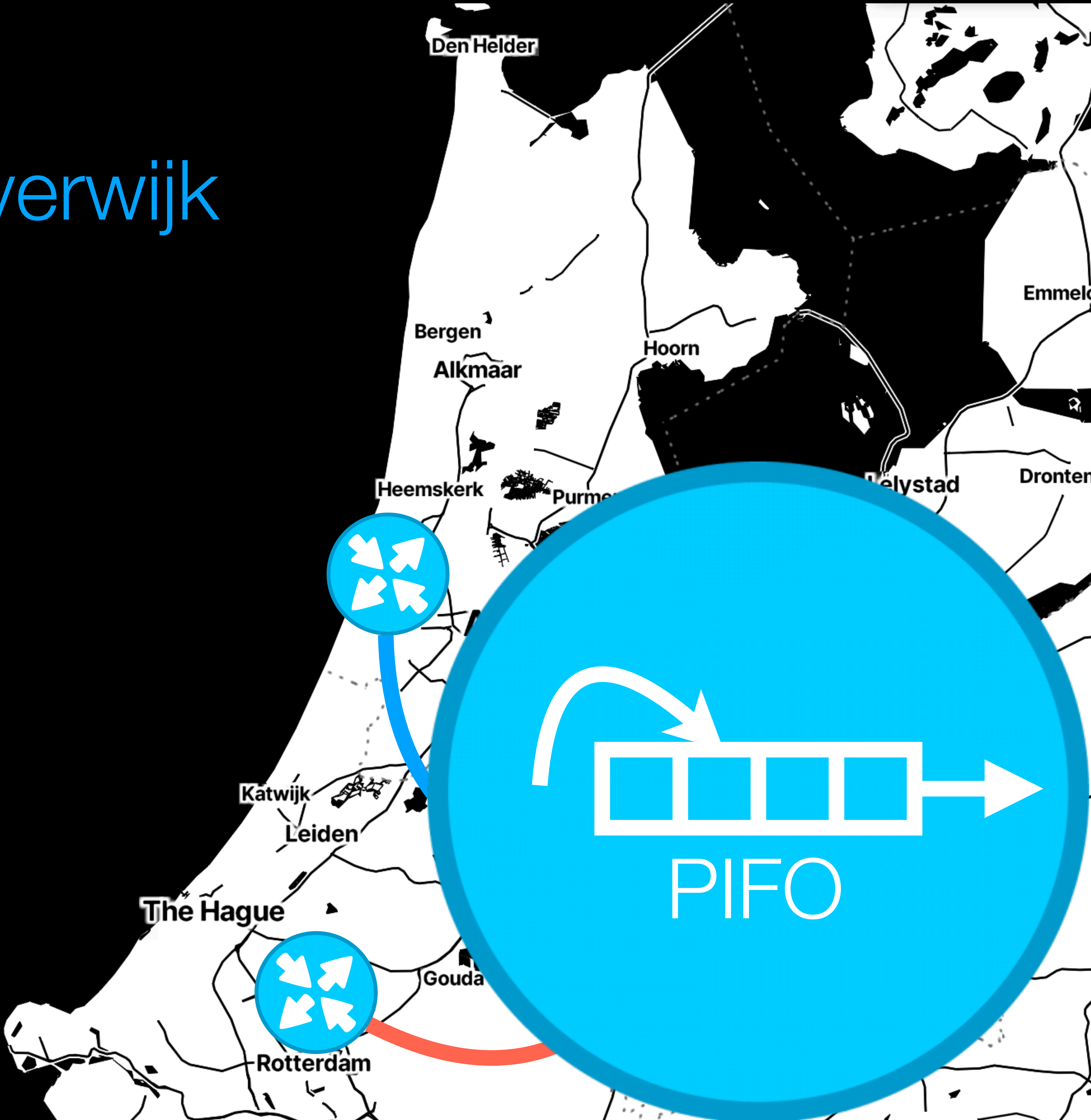


Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

$B_n, \ldots, B_1, (R,B)^*$

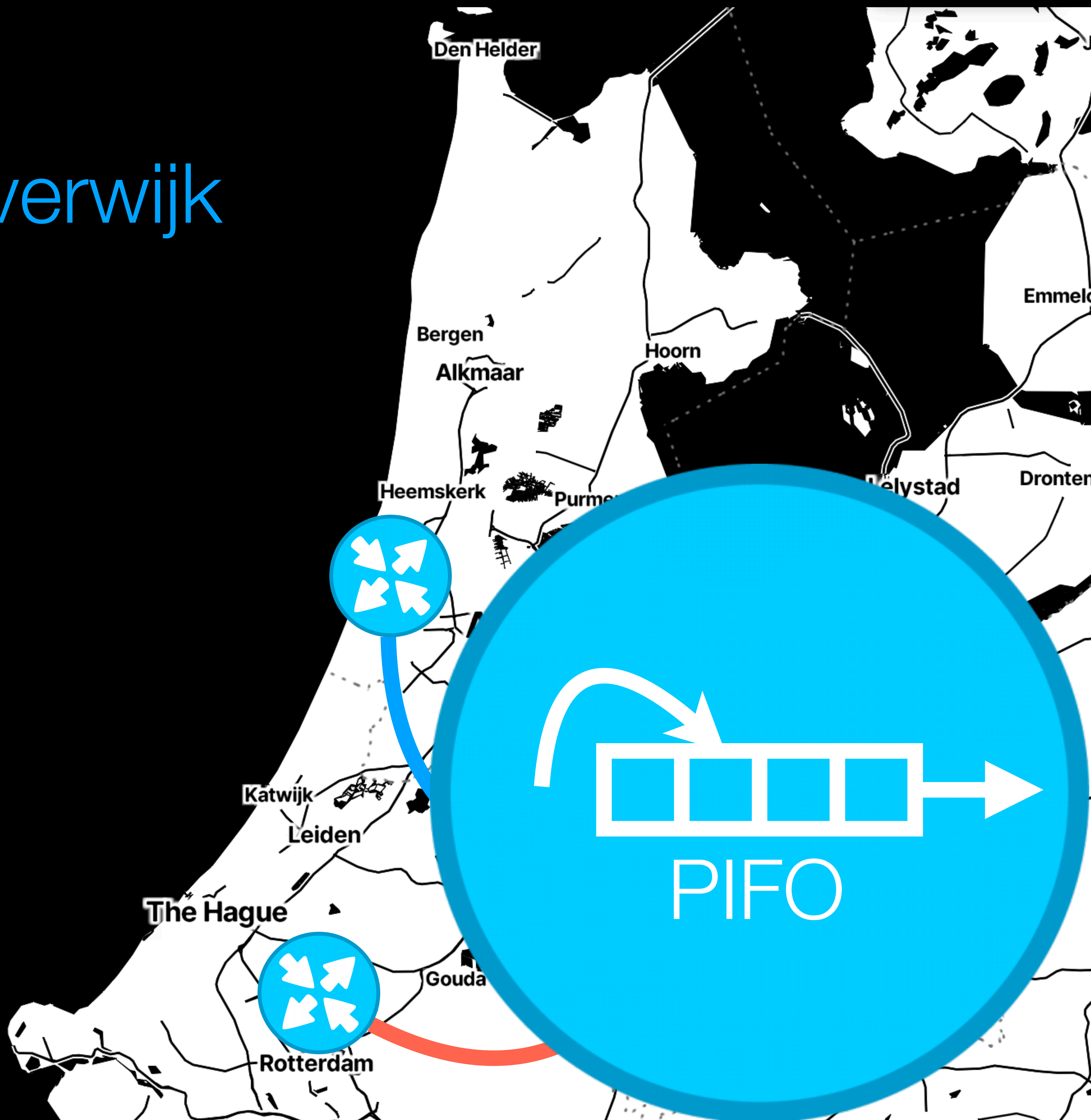


Traffic incoming
from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

$$\begin{tikzcd}
R \arrow[dr, bend left] & \\
B_n, \ldots, & B_1, (R,B)^*
\end{tikzcd}$$

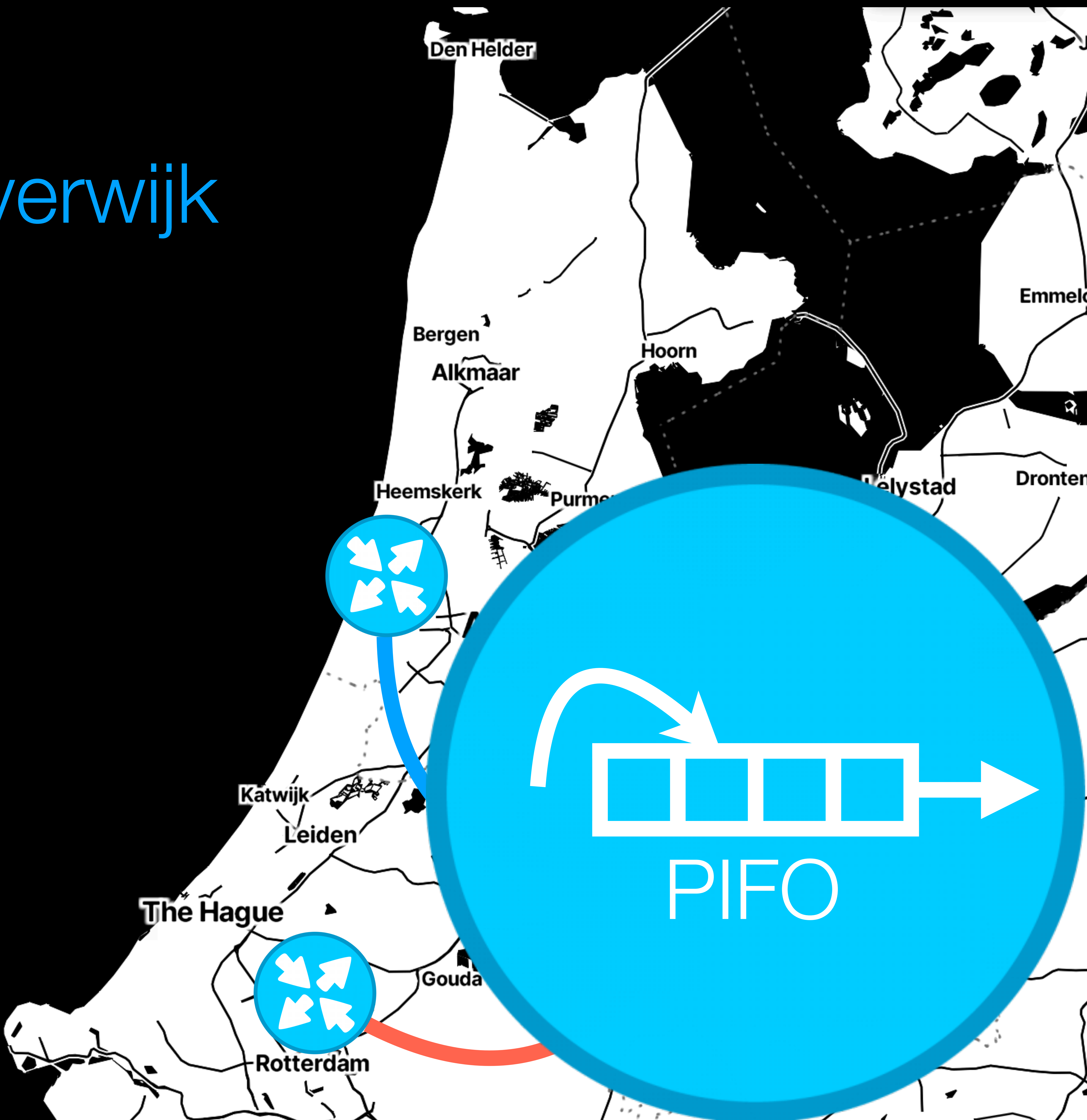


Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

$$\begin{tikzcd} B \arrow[dr, bend right] & \\ & B_n, \ldots, B_1, (R,B)^* \end{tikzcd}$$

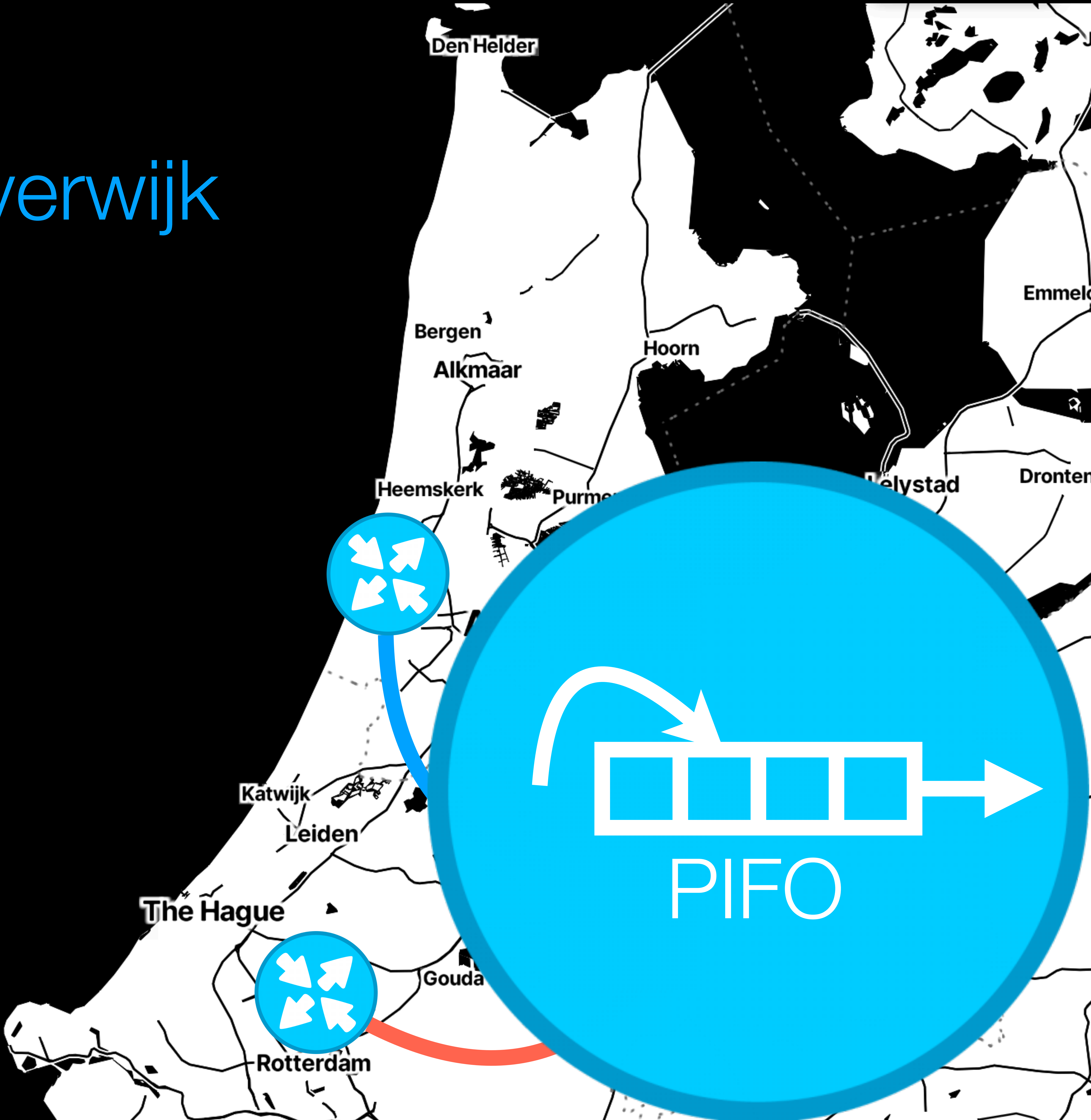


Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

B_n, …, B_1, (R,B)*



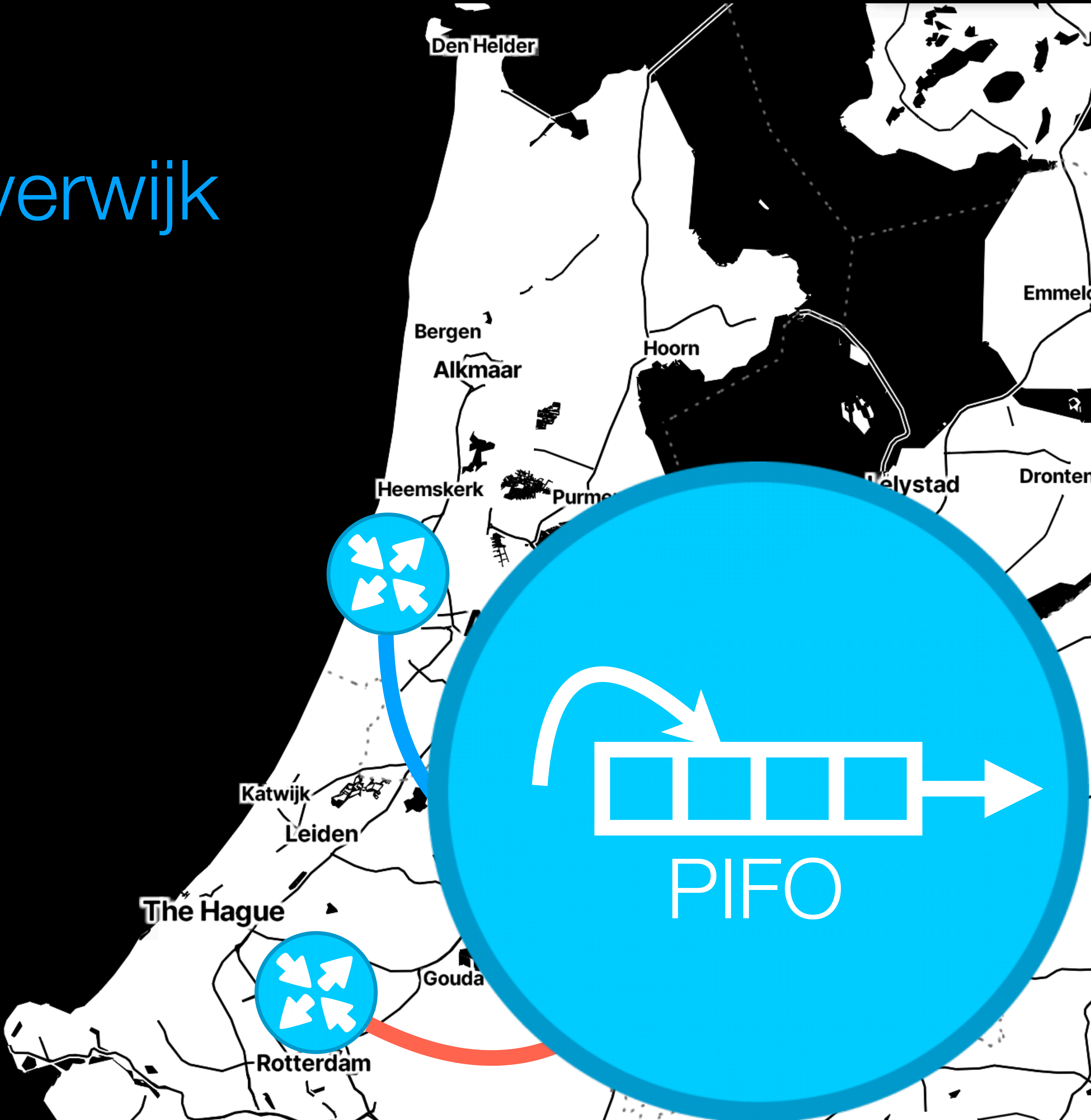
Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

B_n, …, B_1, (R,B)*

R_n, …, R_1, (R,B)*



Traffic incoming from Rotterdam and Beverwijk

Goal: interleave R and B

A PIFO will suffice.

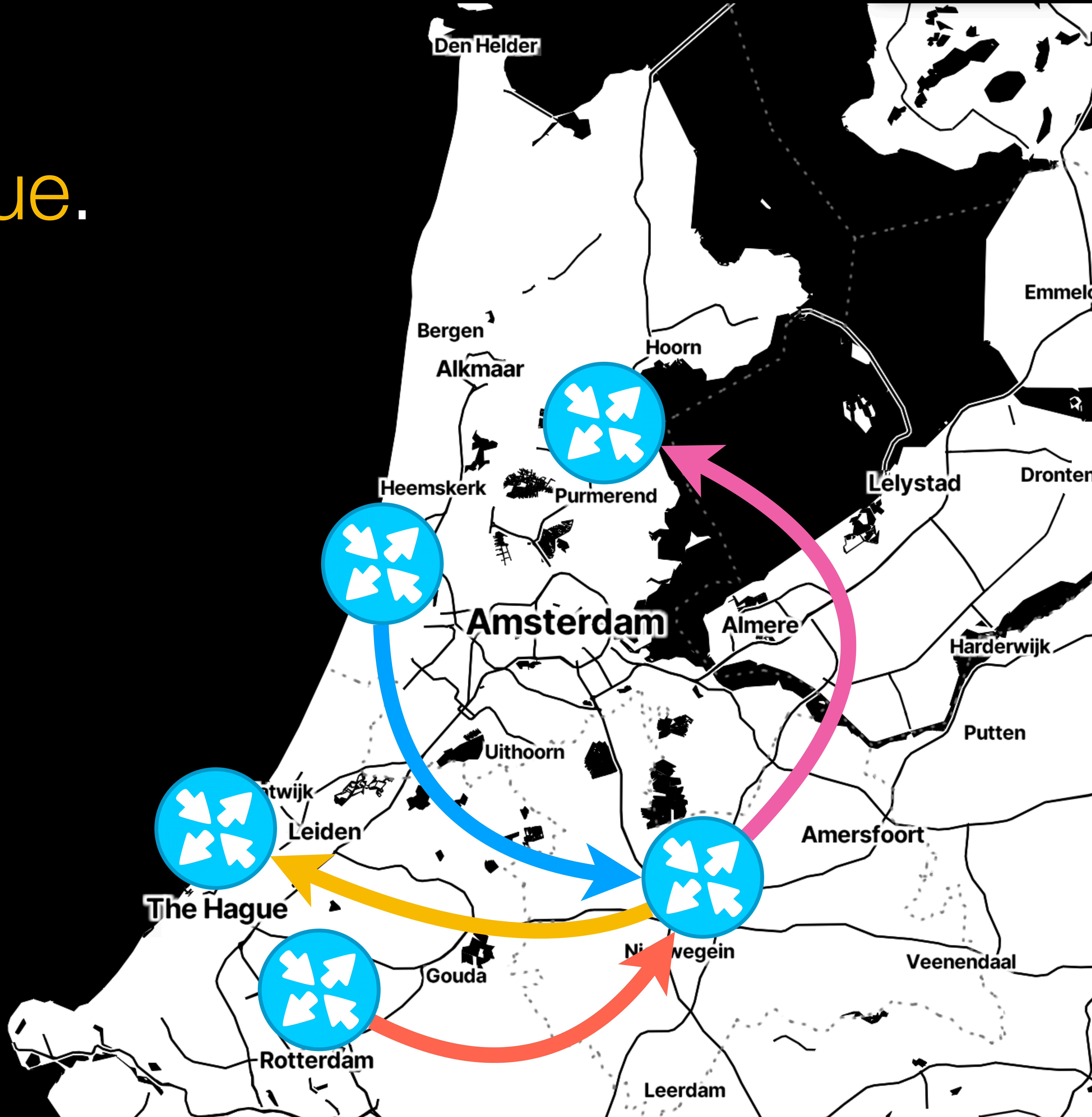
B_n, …, B_1, (R,B)*

R_n, …, R_1, (R,B)*

(R,B)*

Den Helder
Bergen
Alkmaar
Hoorn
Emmeloord
Heemskerk
Purmerend
Lelystad
Dronten
Katwijk
Leiden
The Hague
Gouda
Rotterdam
PIFO

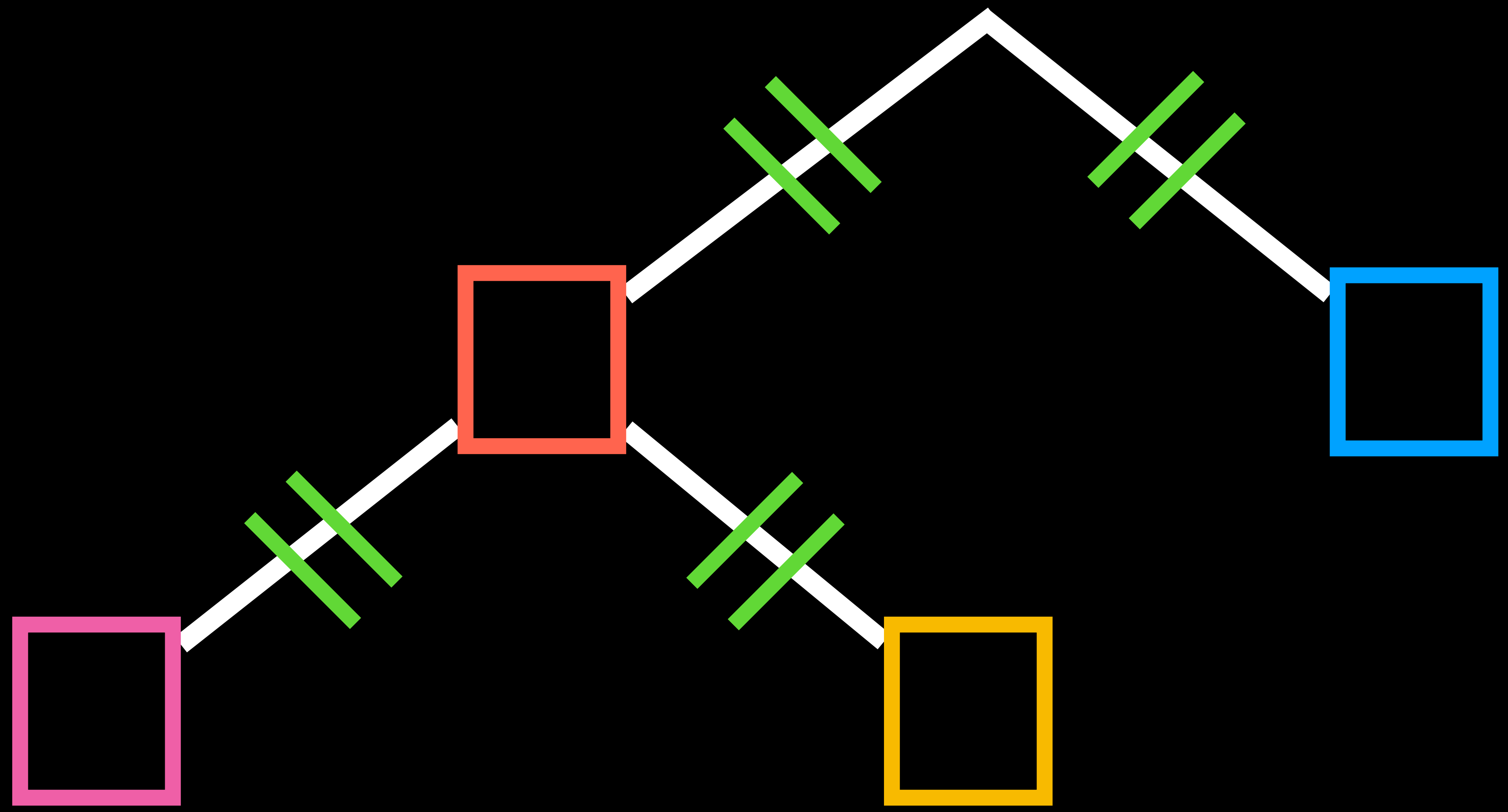
R traffic goes to either Purmerend or The Hague.

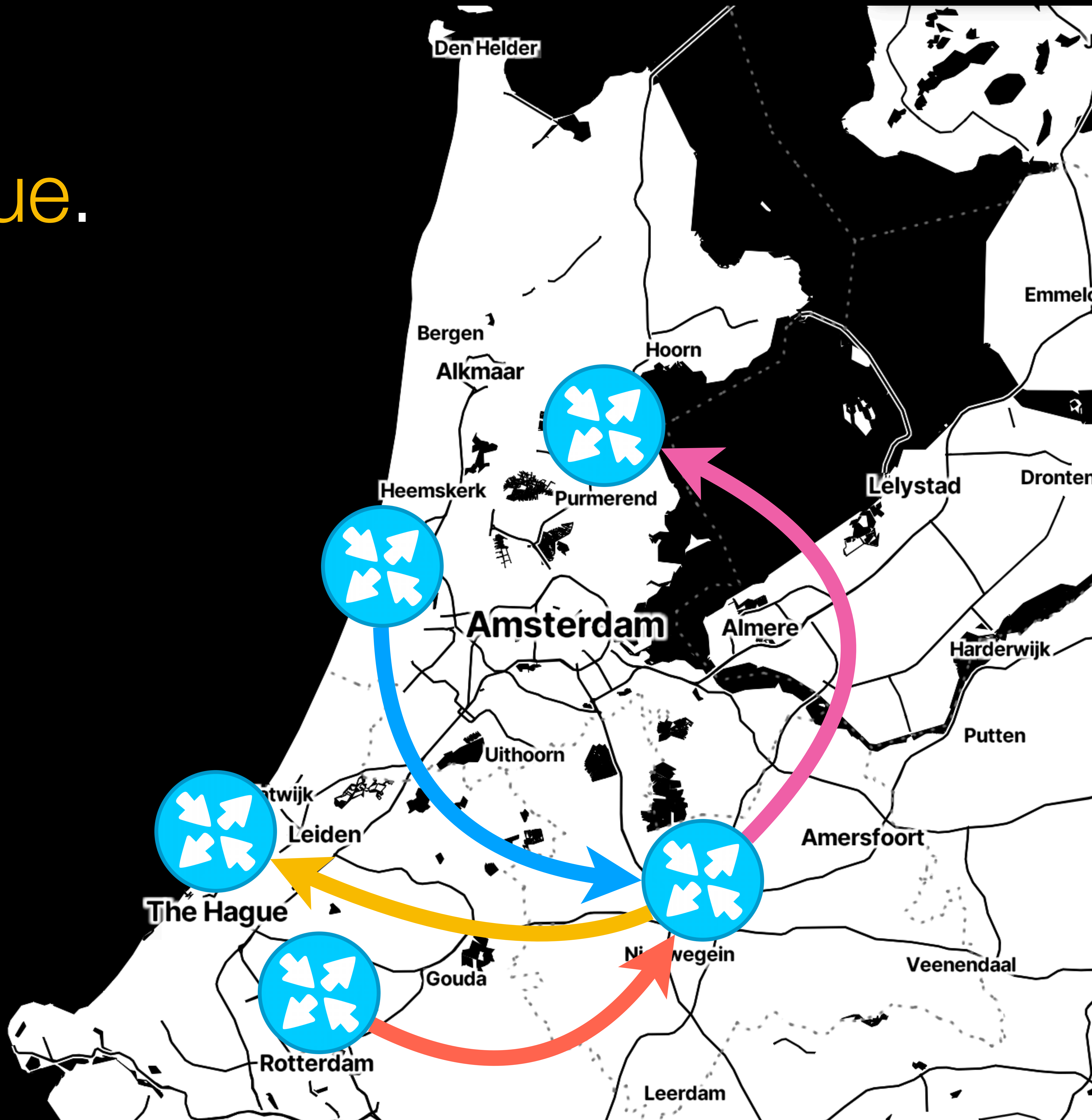


R traffic goes to either Purmerend or The Hague.

Goal:

Interleave R and B;
interleave P and T.





Dual goals:
interleave R and B;
  interleave P and T.

Dual goals:

interleave R and B;

interleave P and T.

B_3, B_2, P_2, B_1, P_1

Dual goals:
interleave R and B;
  interleave P and T.

$$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$$

Dual goals:
interleave R and B;
  interleave P and T.

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, B_2, P_2, T_1, B_1, P_1$

Dual goals:
interleave R and B;
  interleave P and T.

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, B_2, P_2, T_1, B_1, P_1$

$B_3, B_2, P_2, B_1, T_1, P_1$

Dual goals:

interleave R and B;

interleave P and T.

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, B_2, P_2, T_1, B_1, P_1$

$B_3, B_2, P_2, B_1, T_1, P_1$

$B_3, B_2, P_2, B_1, P_1, T_1$

Dual goals:
interleave R and B;
interleave P and T.

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

B_3, B_2, 🟥, 🟥, B_1, 🟥
B_3, B_2, 🟥, B_1, 🟥, 🟥
B_3, B_2, 🟥, B_1, 🟥, 🟥

Dual goals:
interleave R and B;
interleave P and T.

$B_3, B_2, P_2, T_1, B_1, P_1$
$B_3, B_2, P_2, B_1, T_1, P_1$
$B_3, B_2, P_2, B_1, P_1, T_1$

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

Dual goals:
interleave R and B;
interleave P and T.

~~$B_3, B_2, P_2, T_1, B_1, P_1$~~

~~$B_3, B_2, P_2, B_1, T_1, P_1$~~

~~$B_3, B_2, P_2, B_1, P_1, T_1$~~

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

Dual goals:
interleave R and B;
interleave P and T.

~~$B_3, B_2, P_2, T_1, B_1, P_1$~~
~~$B_3, B_2, P_2, B_1, T_1, P_1$~~
~~$B_3, B_2, P_2, B_1, P_1, T_1$~~

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, P_2, B_2, T_1, B_1, P_1$

Dual goals:
interleave R and B;
interleave P and T.

~~$B_3, B_2, P_2, T_1, B_1, P_1$~~
~~$B_3, B_2, P_2, B_1, T_1, P_1$~~
~~$B_3, B_2, P_2, B_1, P_1, T_1$~~

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, P_2, B_2, T_1, B_1, P_1$
$B_3, P_2, B_2, P_1, B_1, T_1$

Dual goals:
interleave R and B;
interleave P and T.

~~B_3, B_2, P_2, T_1, B_1, P_1~~
~~B_3, B_2, P_2, B_1, T_1, P_1~~
~~B_3, B_2, P_2, B_1, P_1, T_1~~

T_1 → B_3, B_2, P_2, B_1, P_1

B_3, 🟥, B_2, 🟥, B_1, 🟥
B_3, 🟥, B_2, 🟥, B_1, 🟥

Dual goals:
interleave R and B;
interleave P and T.

~~$B_3, B_2, P_2, T_1, B_1, P_1$~~
~~$B_3, B_2, P_2, B_1, T_1, P_1$~~
~~$B_3, B_2, P_2, B_1, P_1, T_1$~~

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, P_2, B_2, T_1, B_1, P_1$
$B_3, P_2, B_2, P_1, B_1, T_1$

Dual goals:
interleave R and B;
interleave P and T.

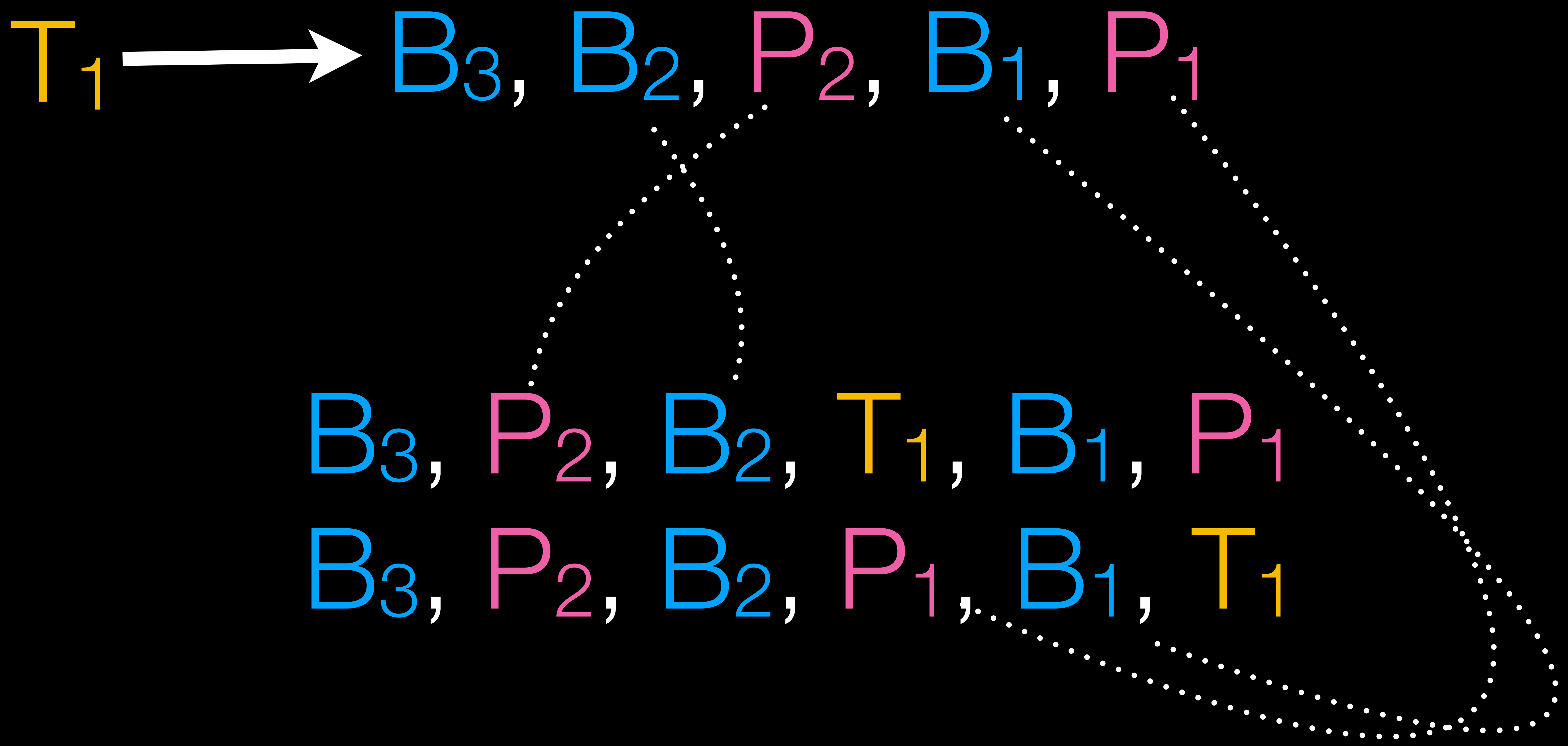
~~$B_3, B_2, P_2, T_1, B_1, P_1$~~
~~$B_3, B_2, P_2, B_1, T_1, P_1$~~
~~$B_3, B_2, P_2, B_1, P_1, T_1$~~

$T_1 \longrightarrow B_3, B_2, P_2, B_1, P_1$

$B_3, P_2, B_2, T_1, B_1, P_1$
$B_3, P_2, B_2, P_1, B_1, T_1$

Dual goals:
interleave R and B;
interleave P and T.

~~B_3, B_2, P_2, T_1, B_1, P_1~~
~~B_3, B_2, P_2, B_1, T_1, P_1~~
~~B_3, B_2, P_2, B_1, P_1, T_1~~



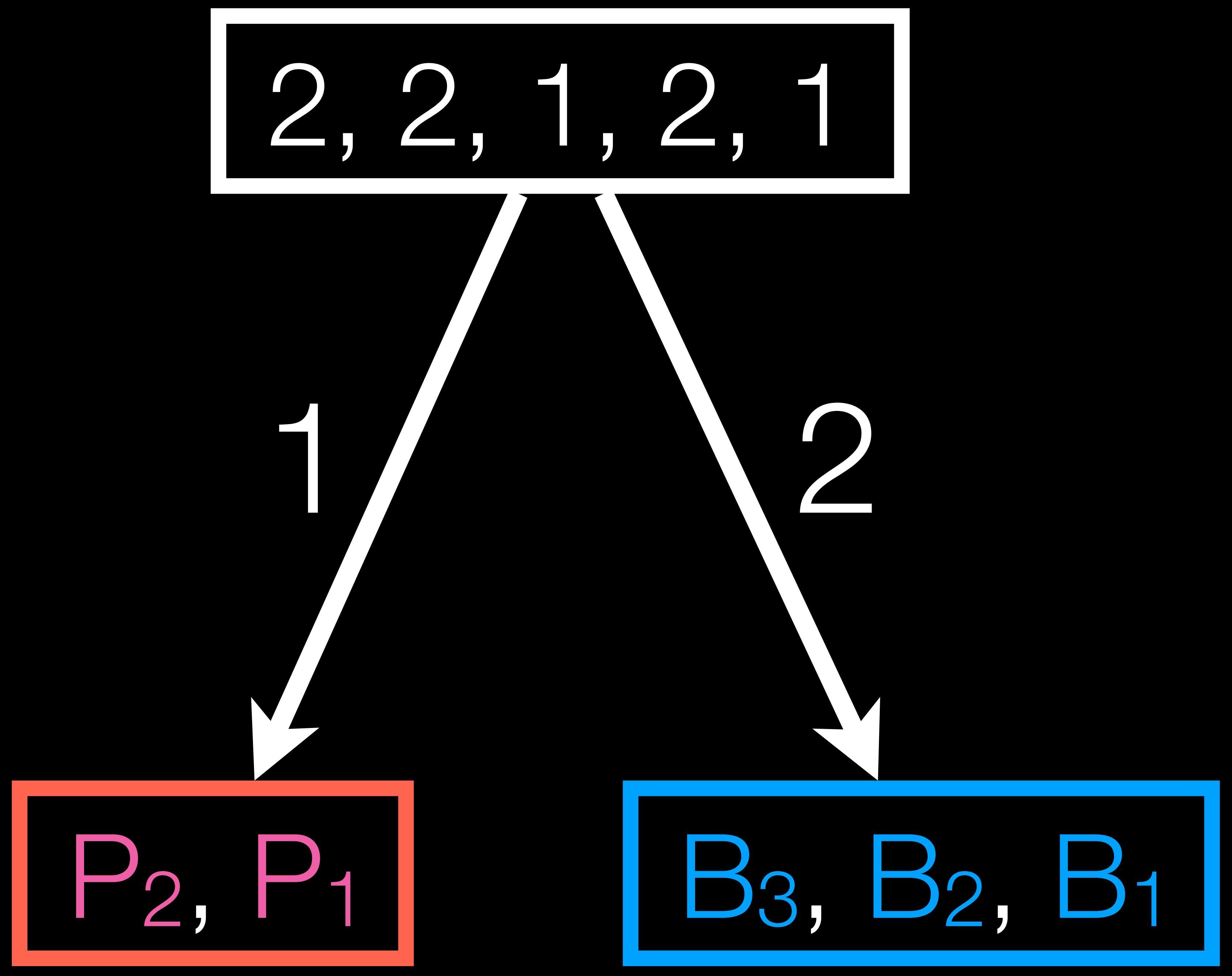
T_1 → B_3, B_2, P_2, B_1, P_1

B_3, P_2, B_2, T_1, B_1, P_1
B_3, P_2, B_2, P_1, B_1, T_1

Enqueueing a packet can require the reordering of buffered packets.

No PIFO can do this.

Introducing: PIFO trees



2, 2, 1, 2, 1

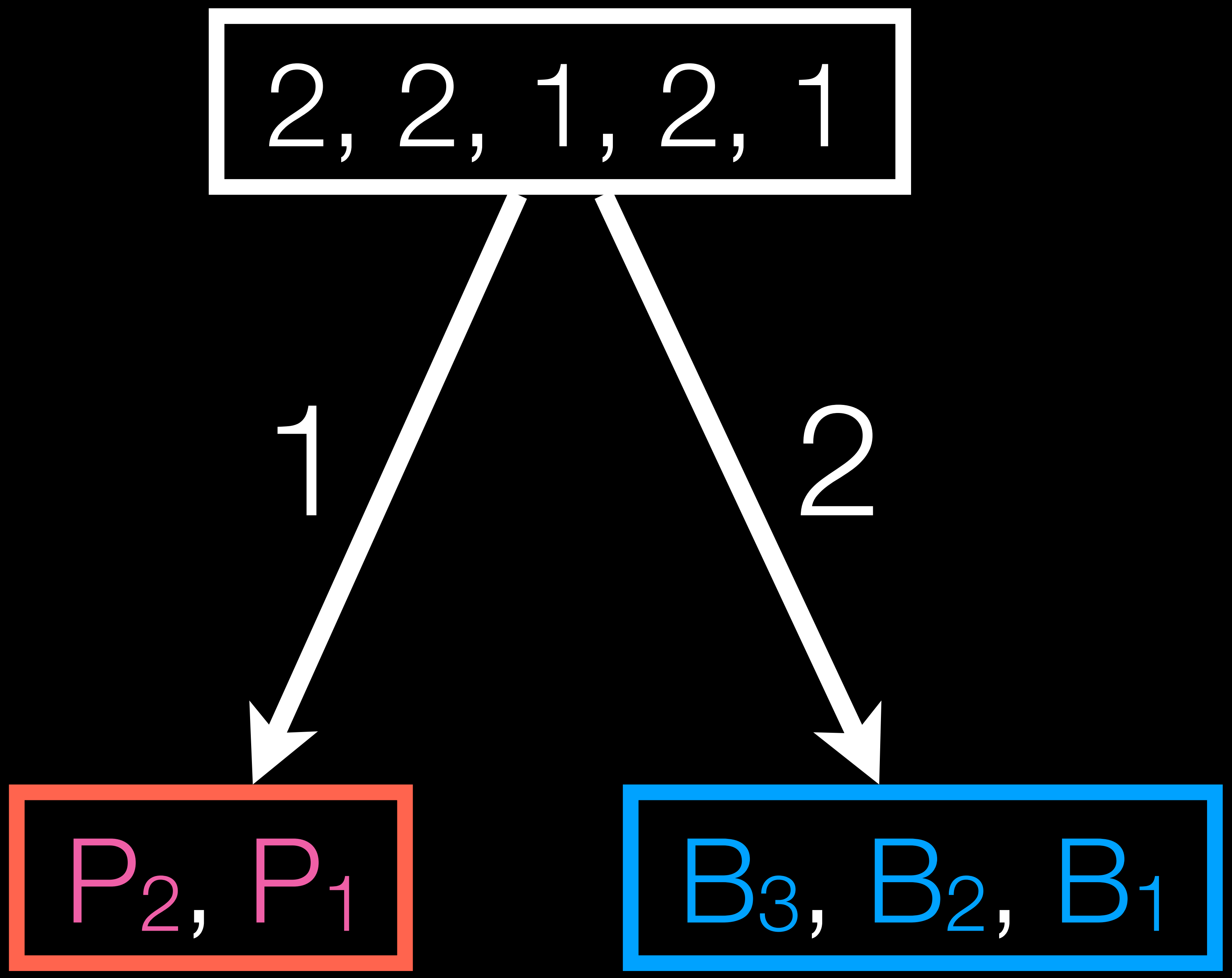
1

2

P_2, P_1

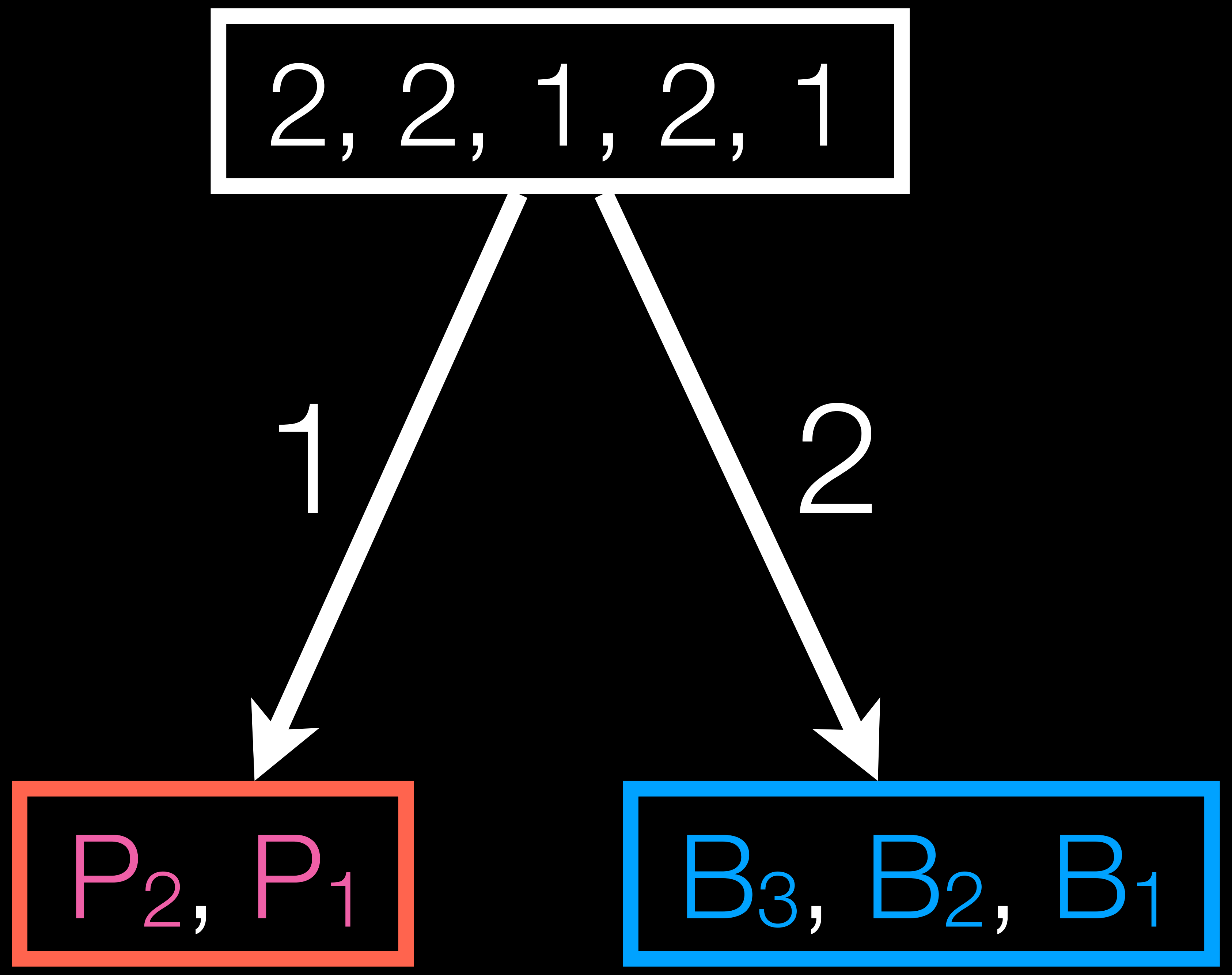
B_3, B_2, B_1

Introducing: PIFO trees



This behaves like a queue!

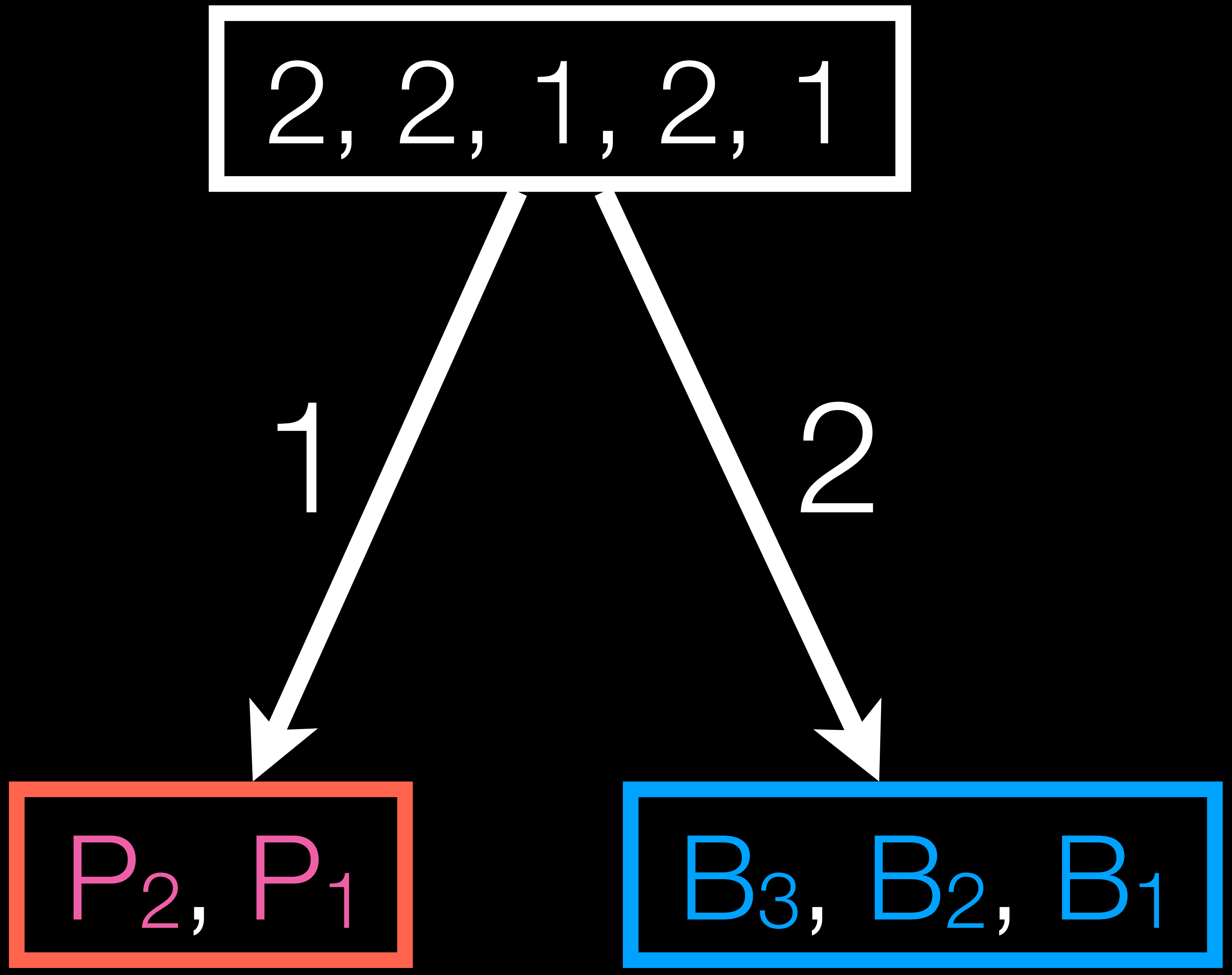
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

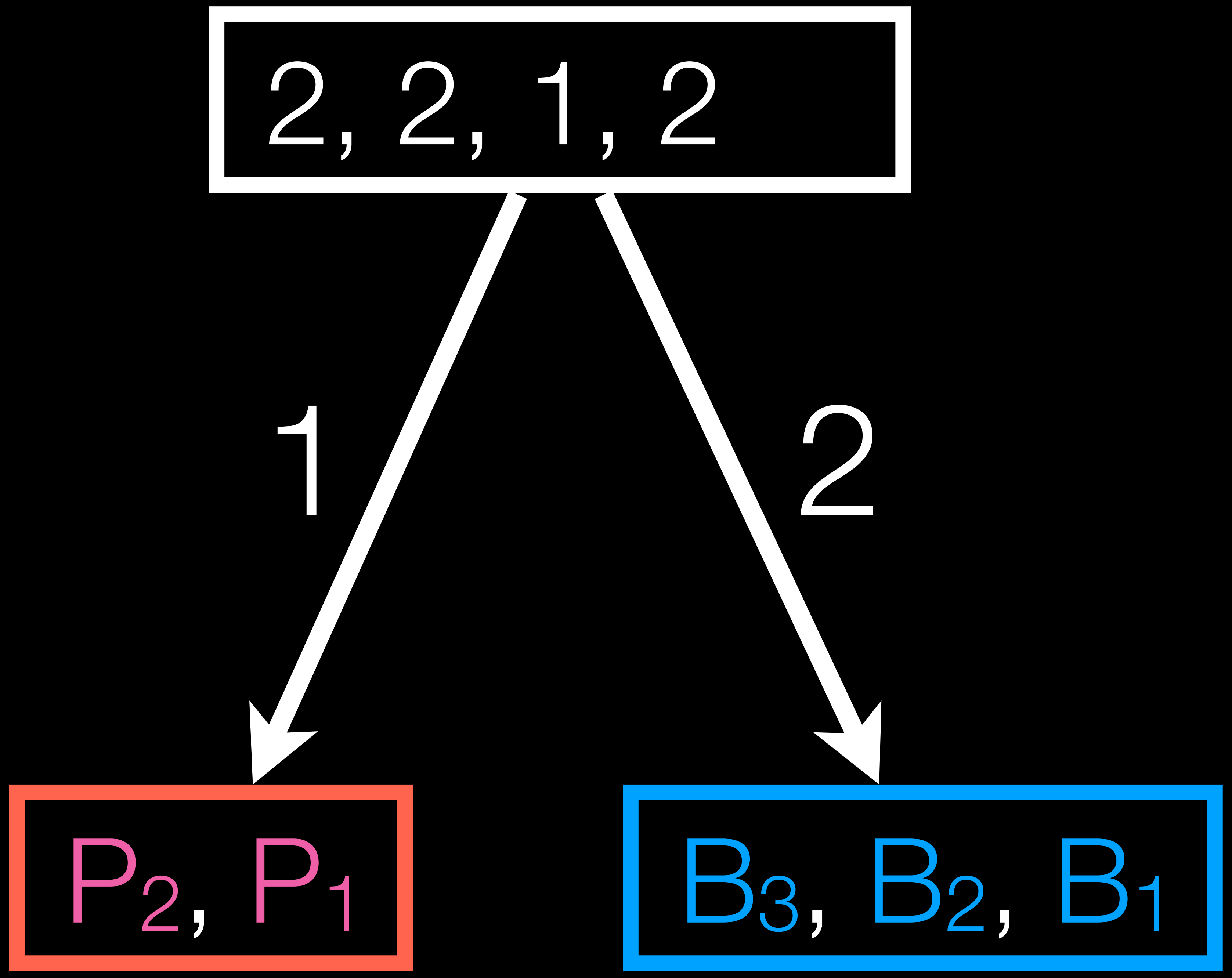
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

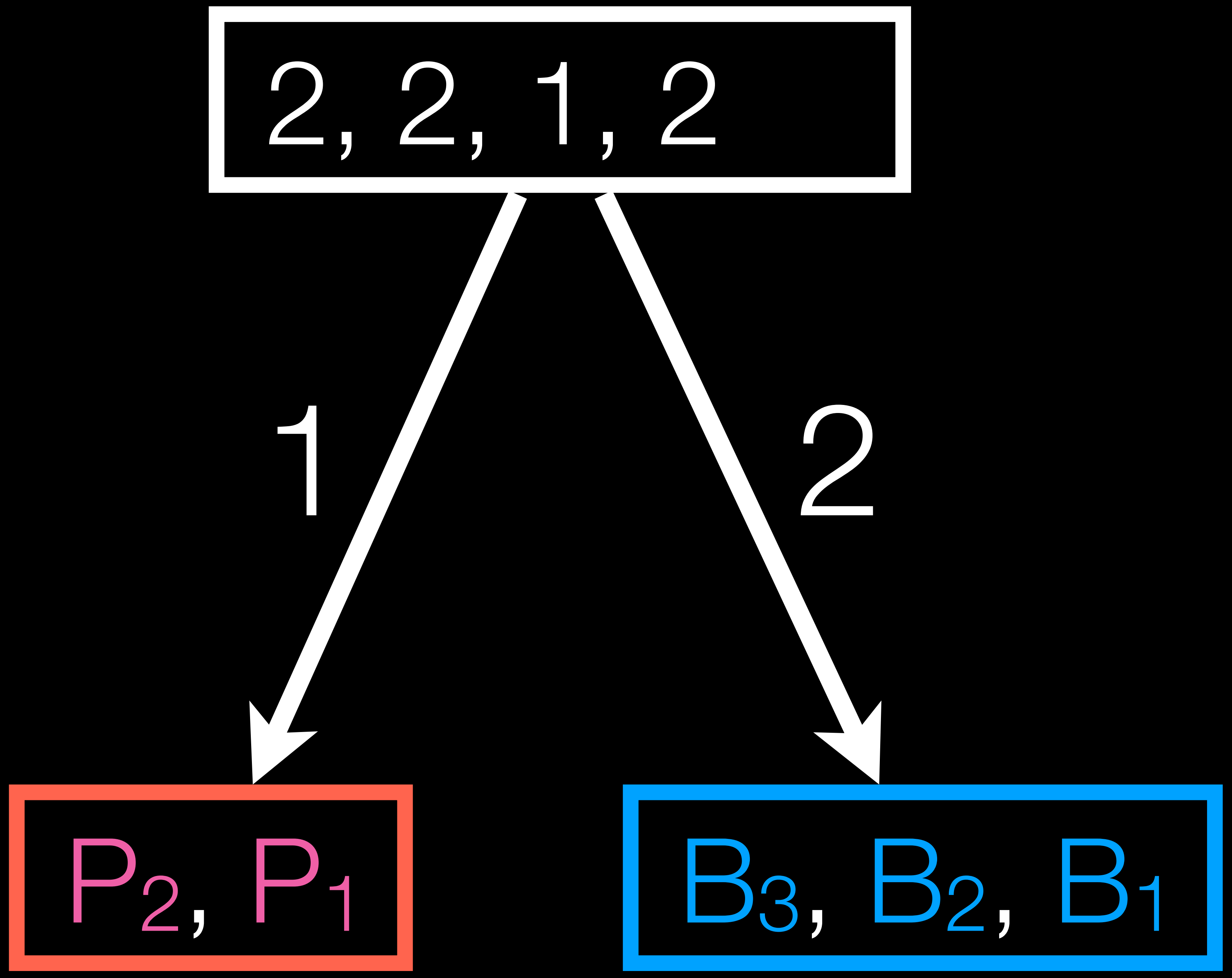
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

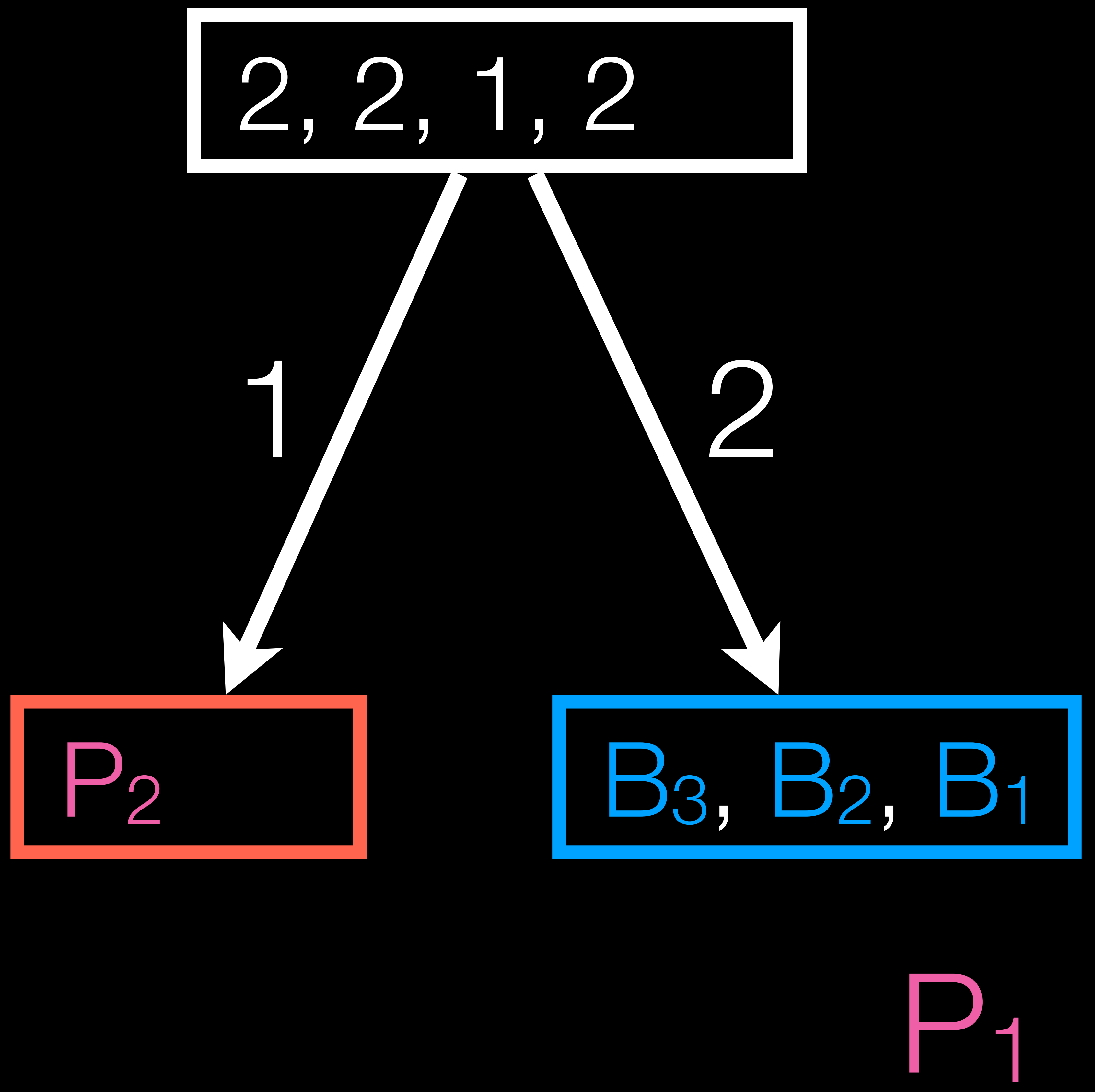
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

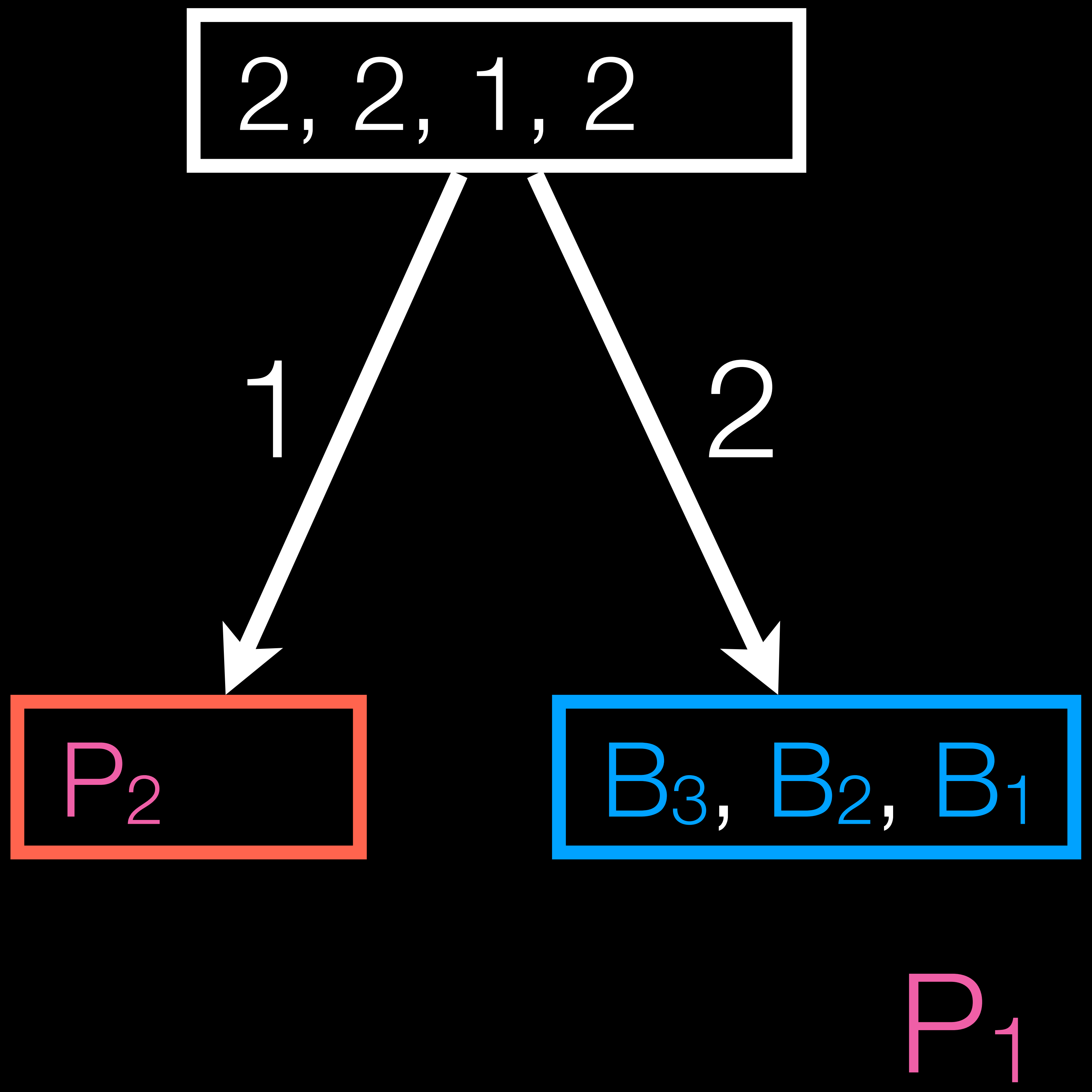
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

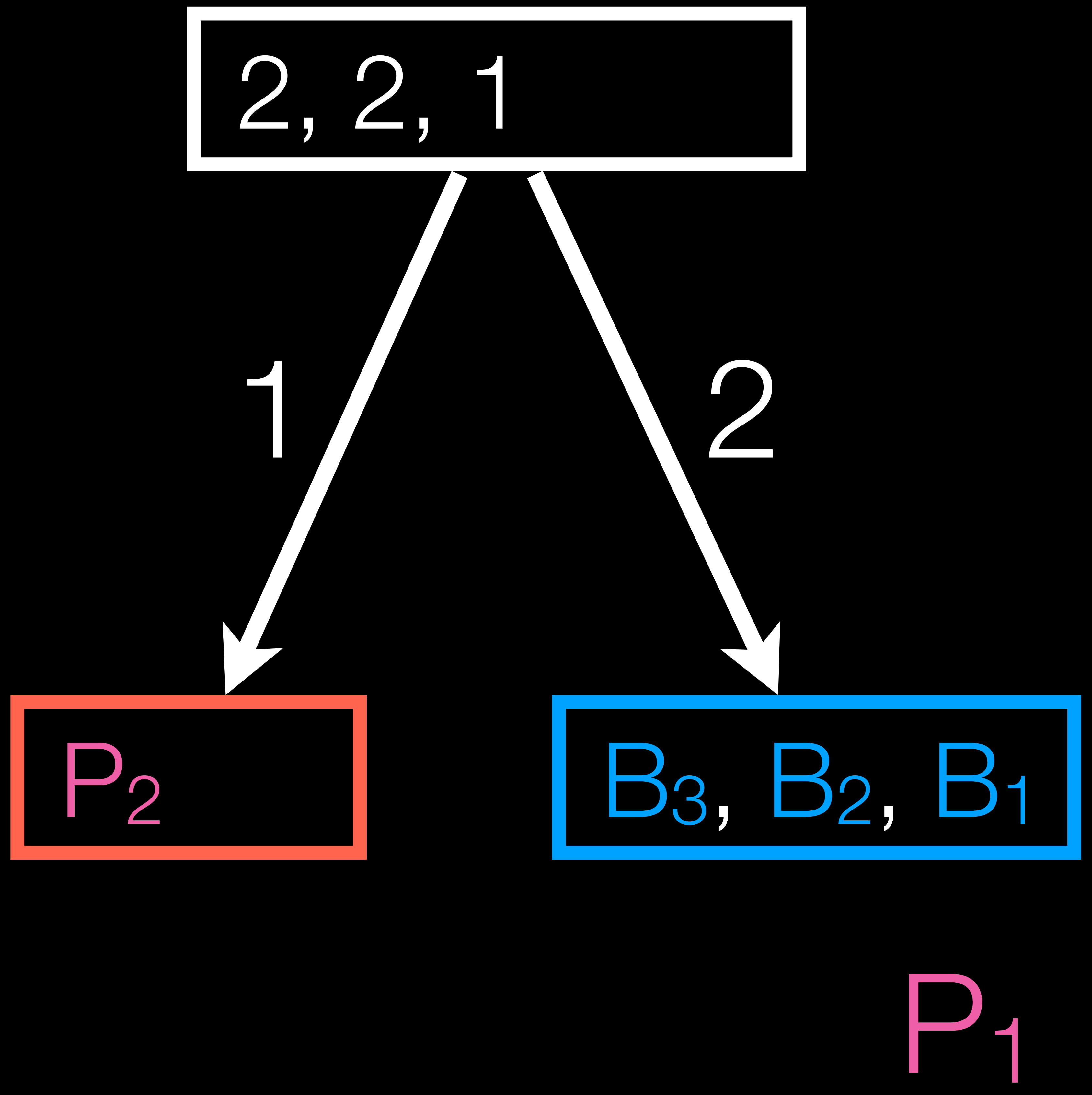
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

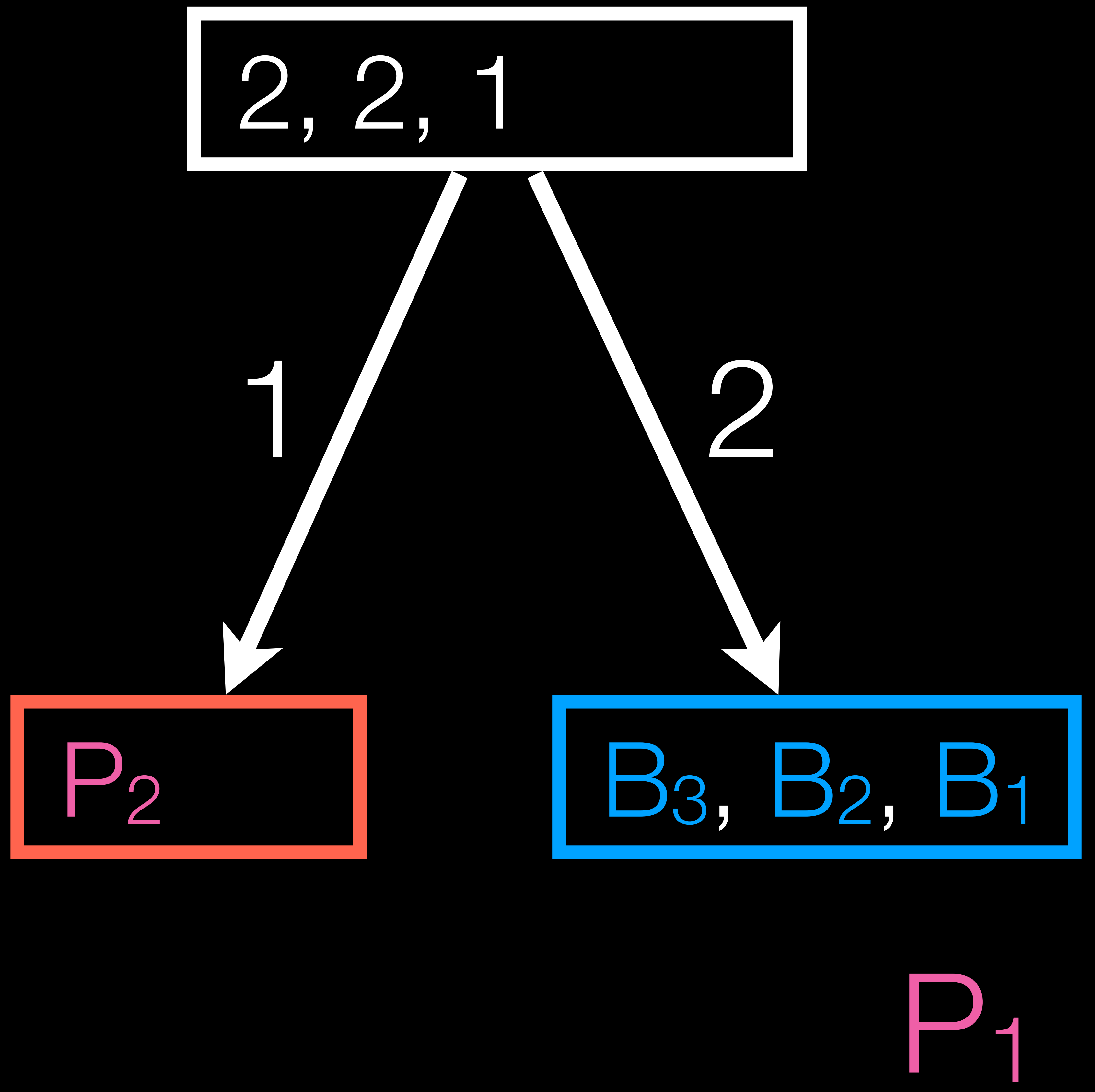
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

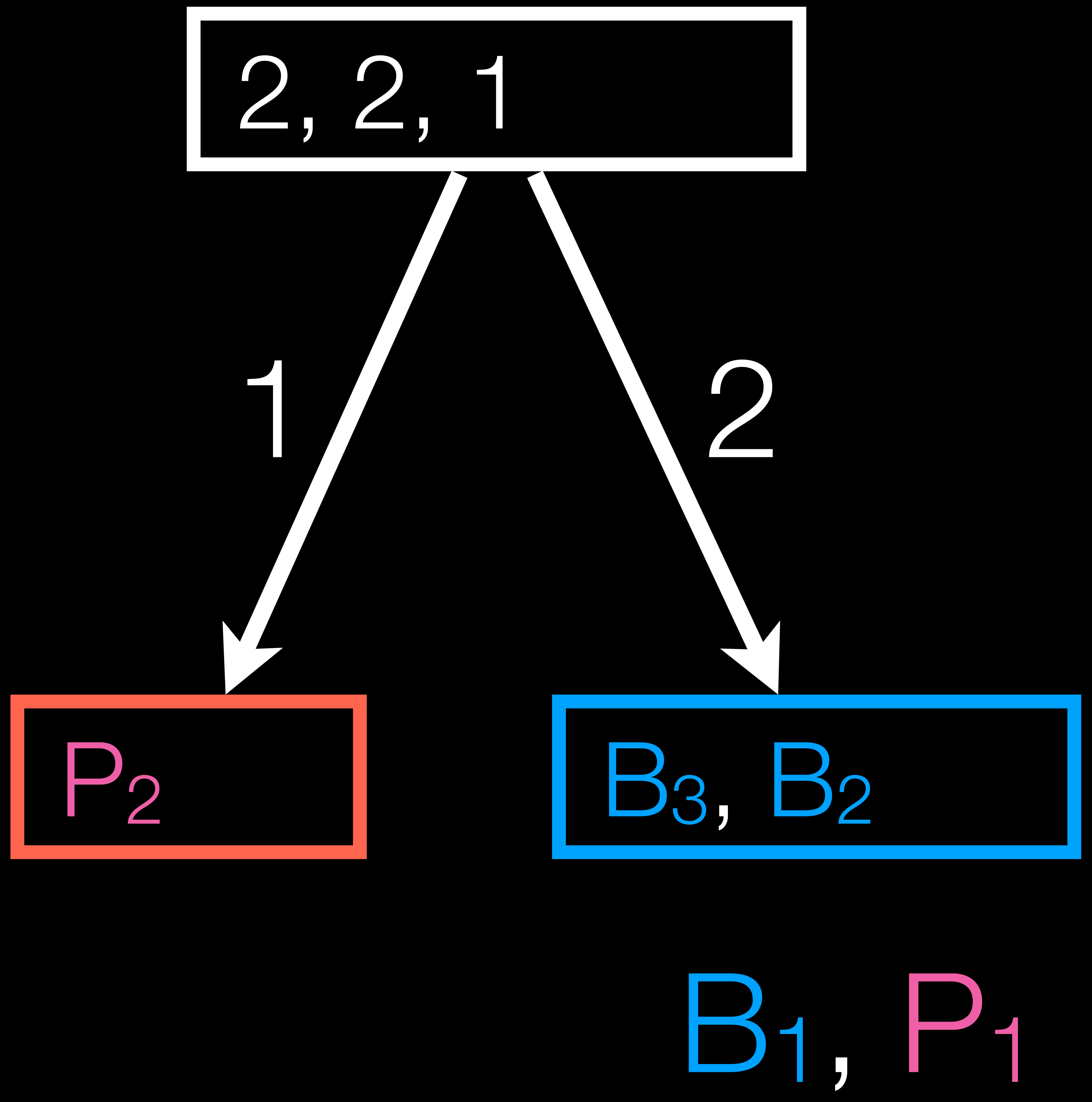
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

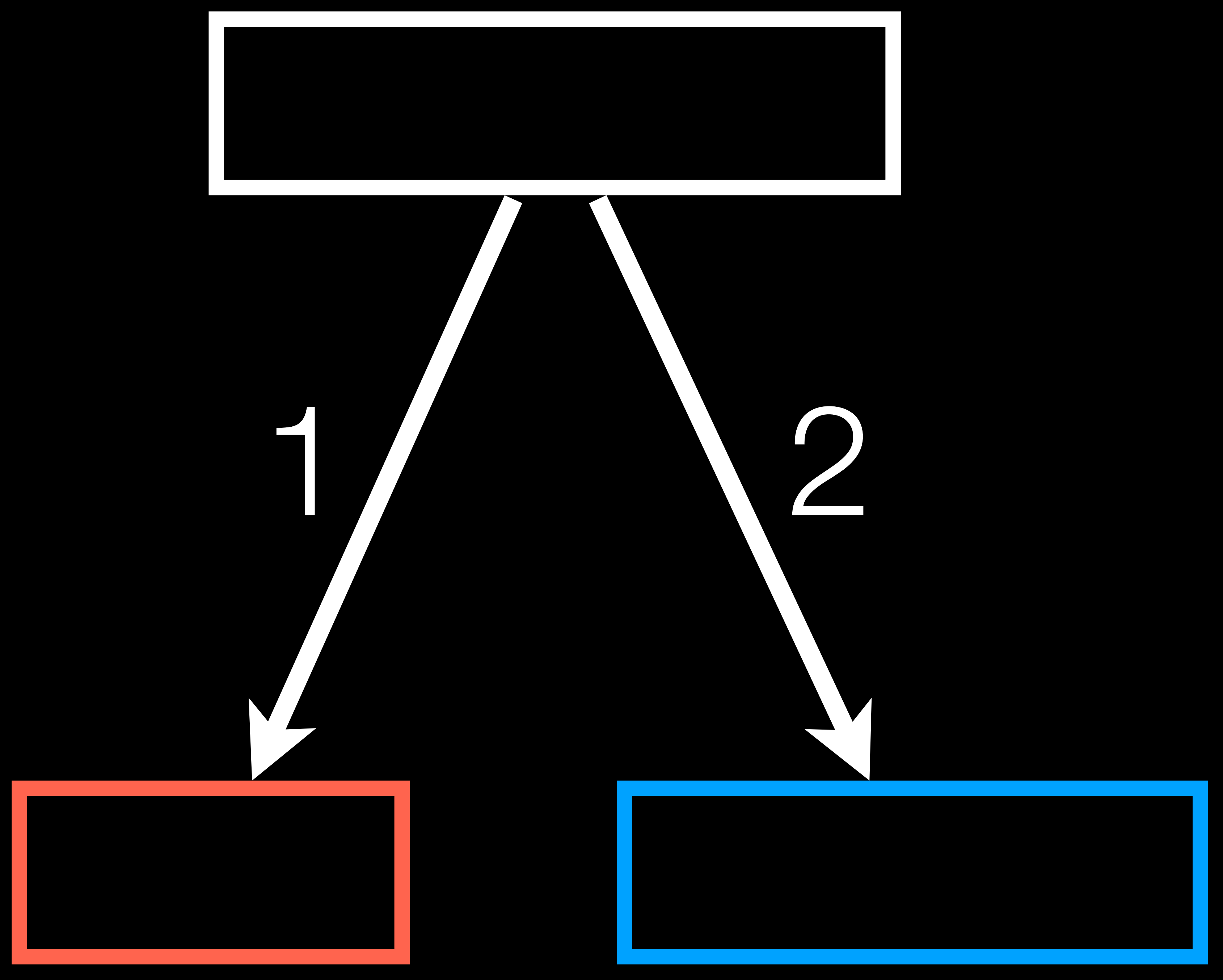
Introducing: PIFO trees



This behaves like a queue!

How do we pop it?

Introducing: PIFO trees

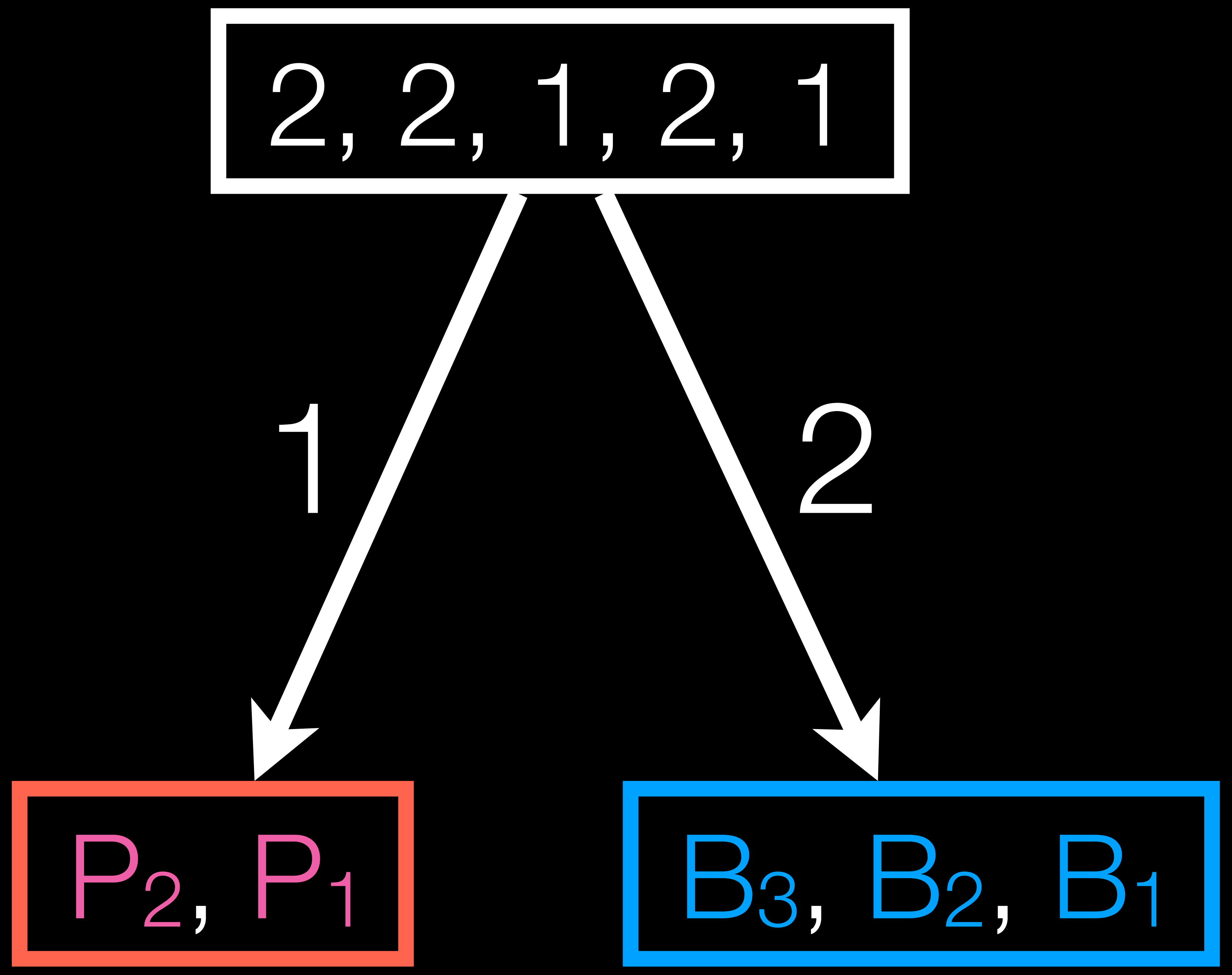


B_3, B_2, P_2, B_1, P_1

This behaves like a queue!

How do we pop it?

Introducing: PIFO trees

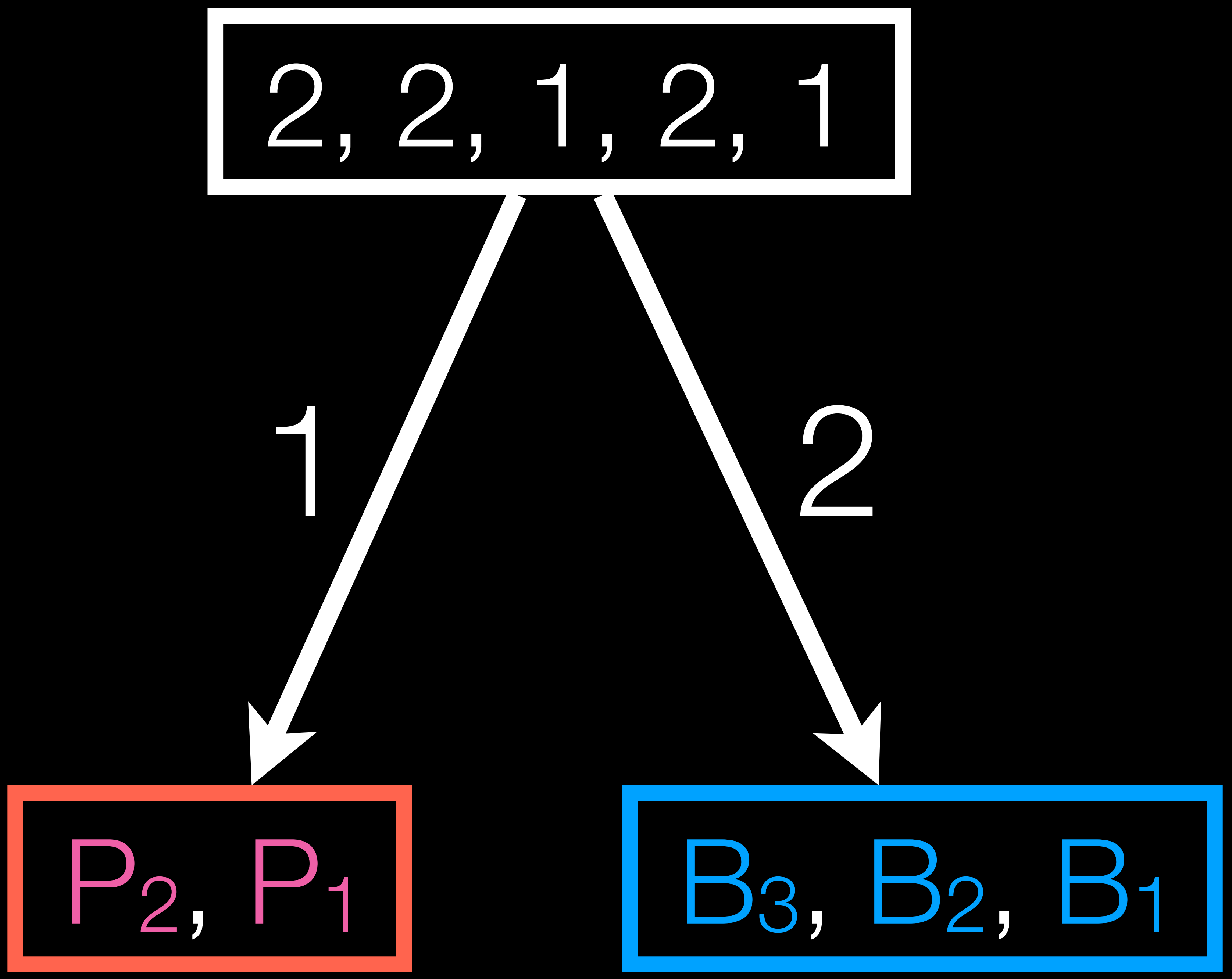


B_3, B_2, P_2, B_1, P_1

This behaves like a queue!

How do we pop it?

Introducing: PIFO trees



2, 2, 1, 2, 1

1

2

P_2, P_1

B_3, B_2, B_1

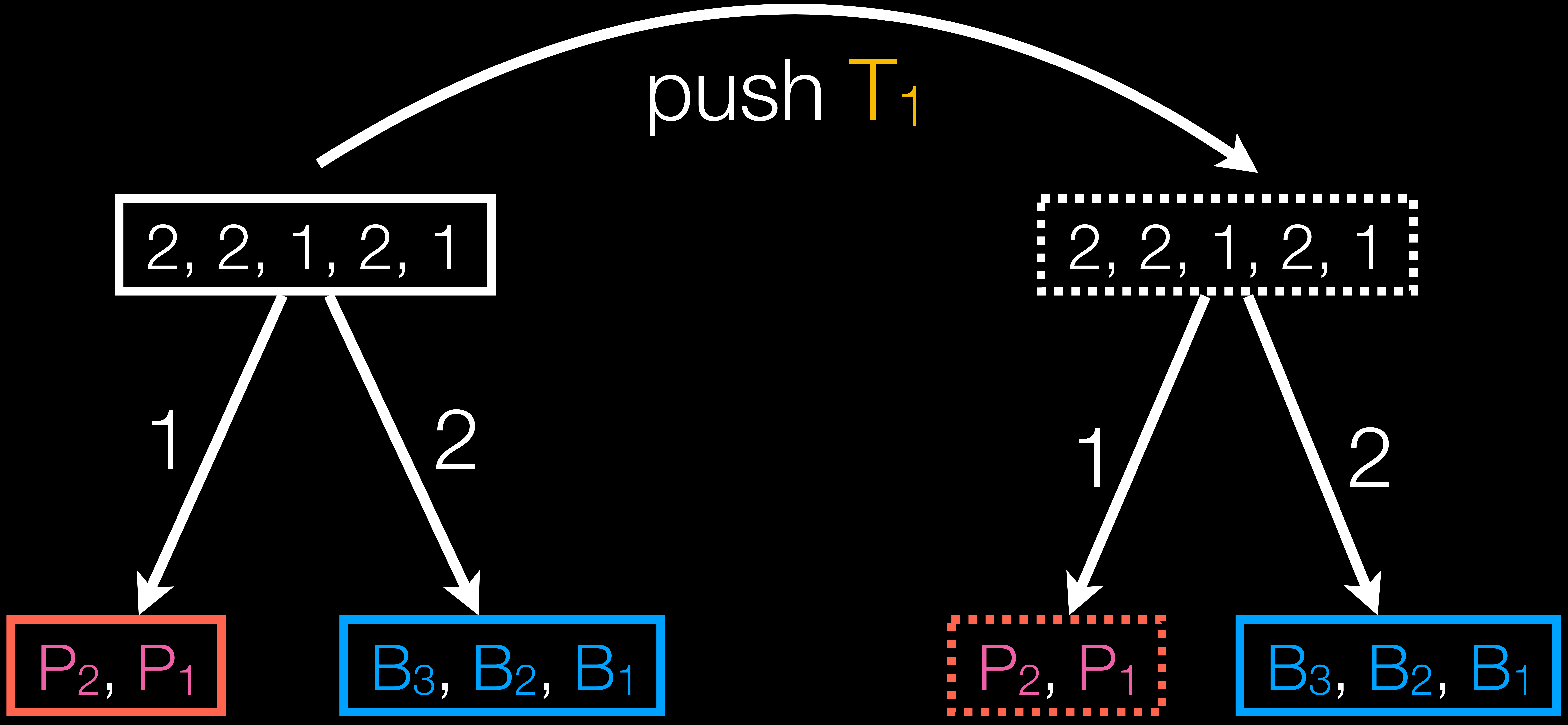
B_3, B_2, P_2, B_1, P_1

This behaves like a queue!

How do we pop it?

How do we push into it?

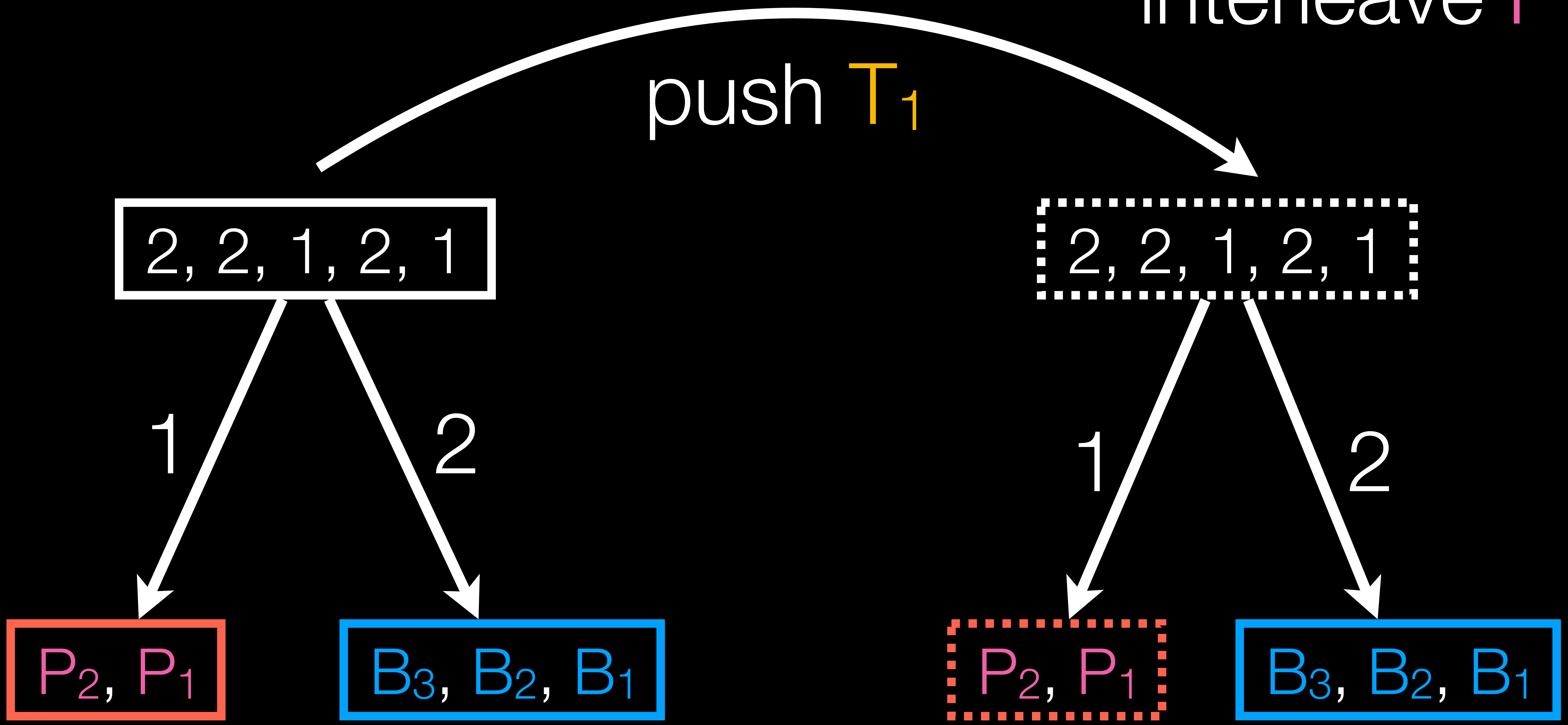
Introducing: PIFO trees



B_3, B_2, P_2, B_1, P_1

Introducing: PIFO trees

interleave R and B;
interleave P and T.



push T_1

2, 2, 1, 2, 1

1

2

P_2, P_1

B_3, B_2, B_1

2, 2, 1, 2, 1

1

2

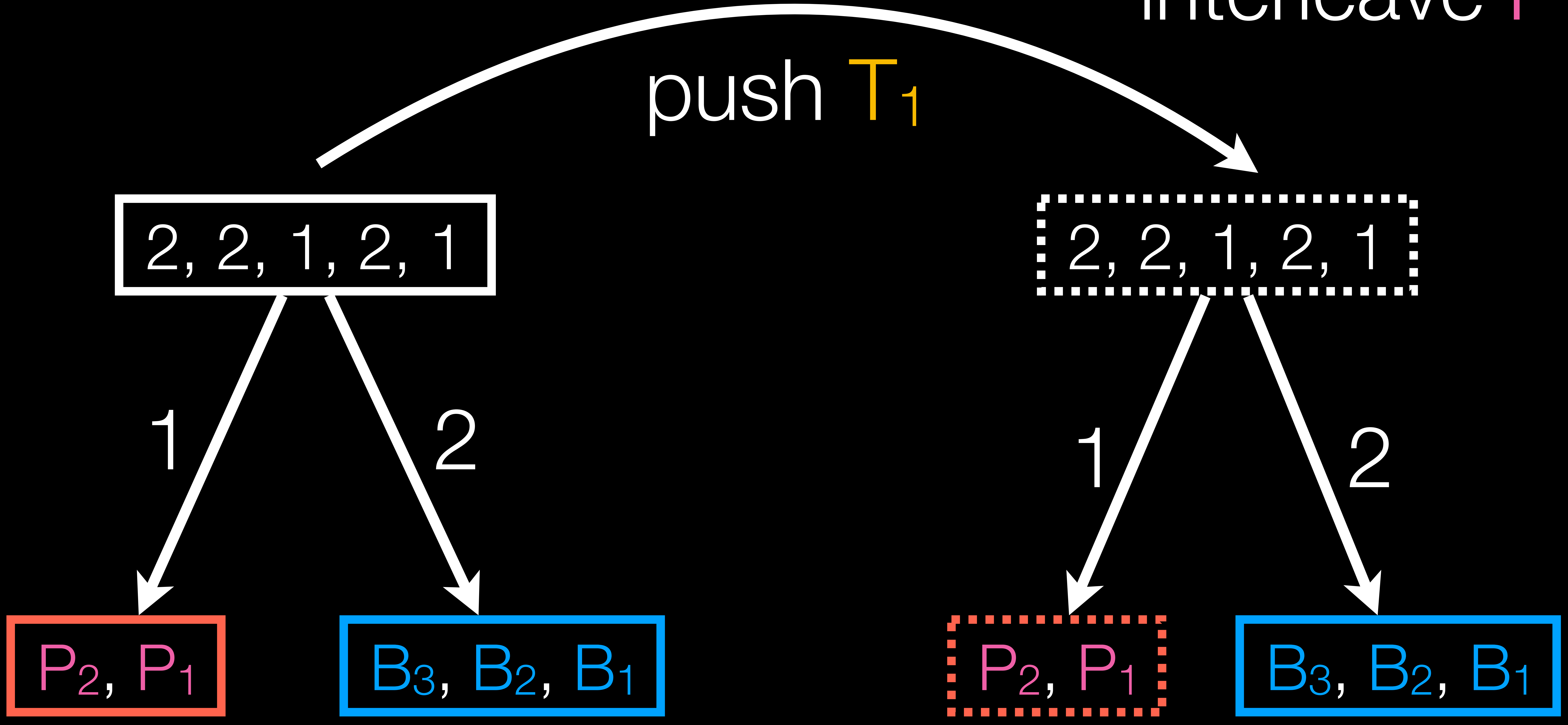
P_2, P_1

B_3, B_2, B_1

B_3, B_2, P_2, B_1, P_1

Introducing: PIFO trees

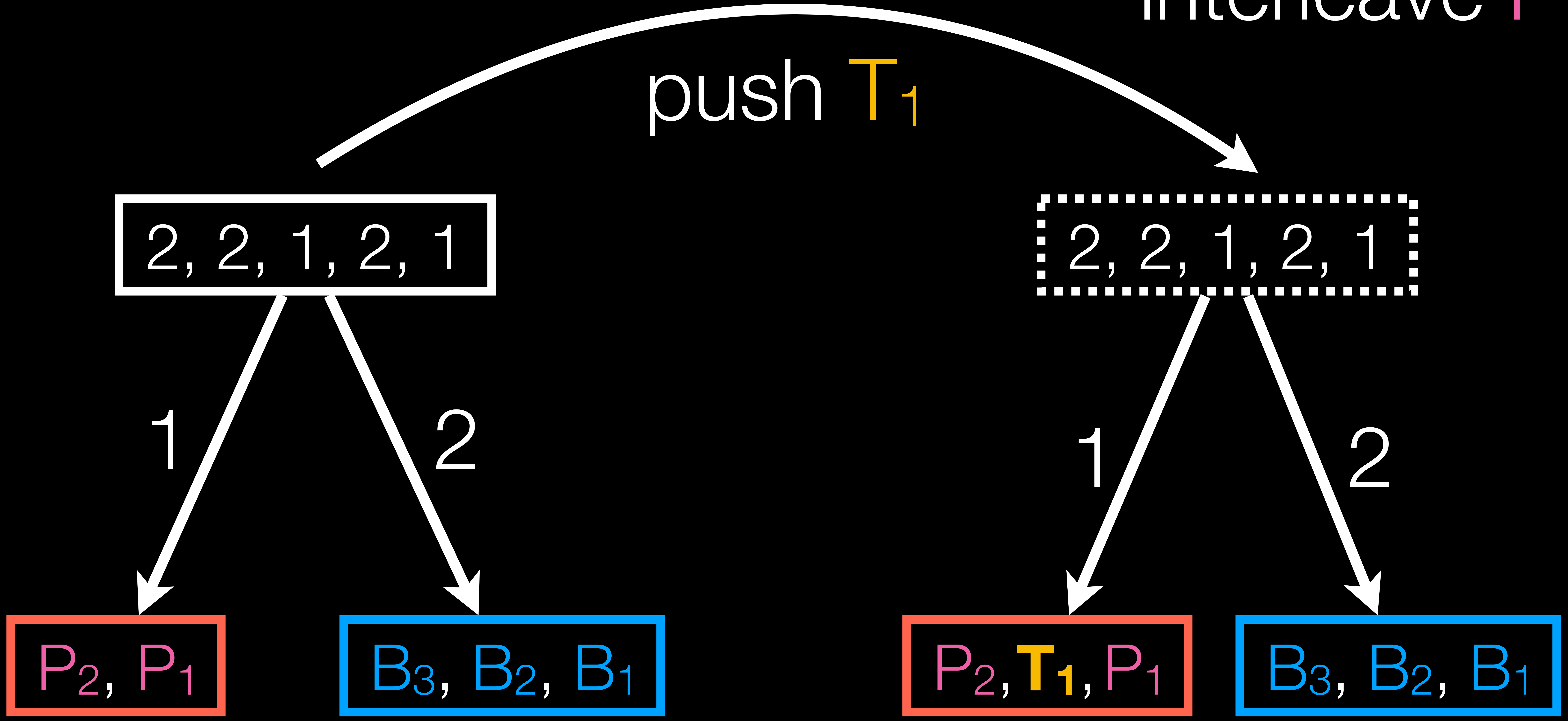
interleave R and B;
interleave P and T.



B_3, B_2, P_2, B_1, P_1

Introducing: PIFO trees

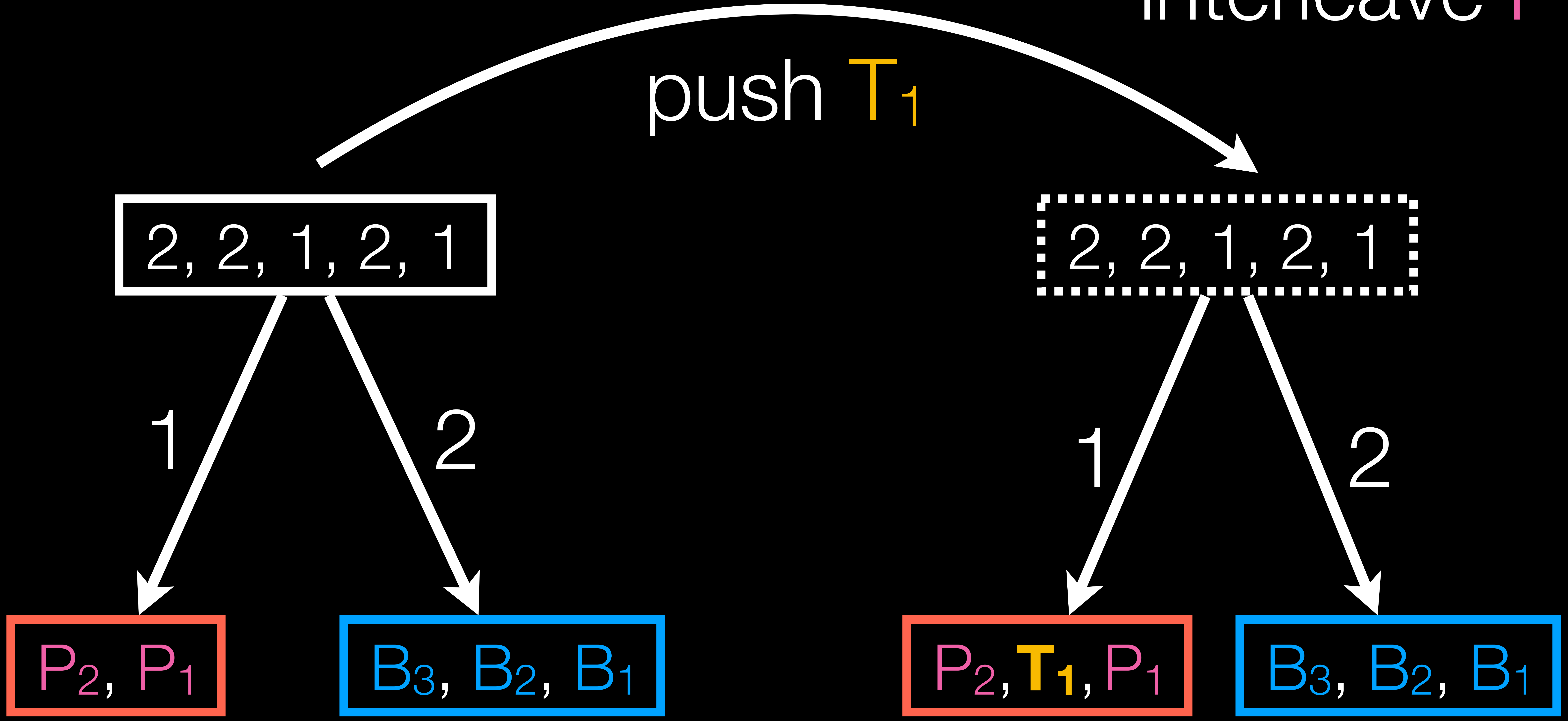
interleave R and B;
interleave P and T.



B_3, B_2, P_2, B_1, P_1

Introducing: PIFO trees

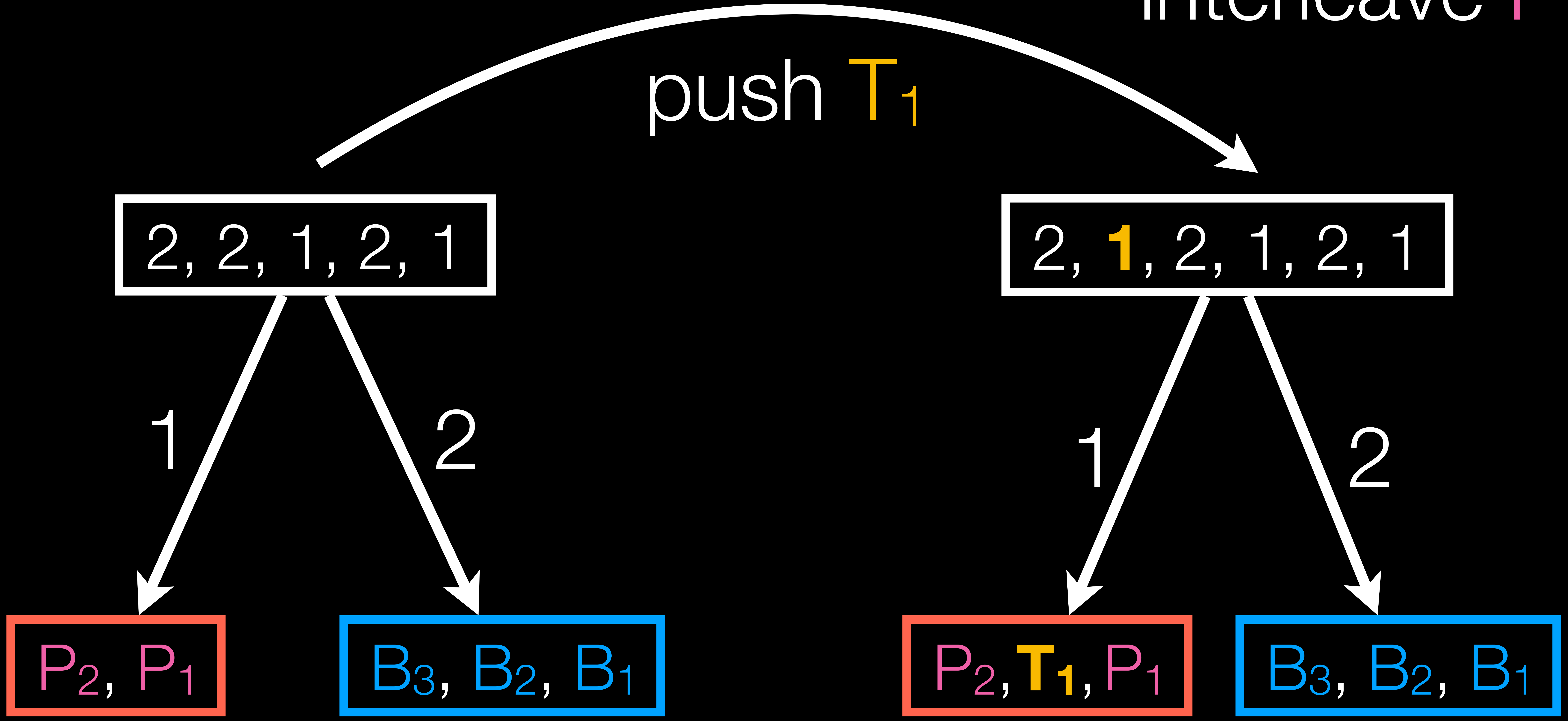
interleave R and B;
interleave P and T.



B_3, B_2, P_2, B_1, P_1

Introducing: PIFO trees

interleave R and B;
interleave P and T.



push T_1

2, 2, 1, 2, 1

1

2

P_2, P_1

B_3, B_2, B_1

2, **1**, 2, 1, 2, 1

1

2

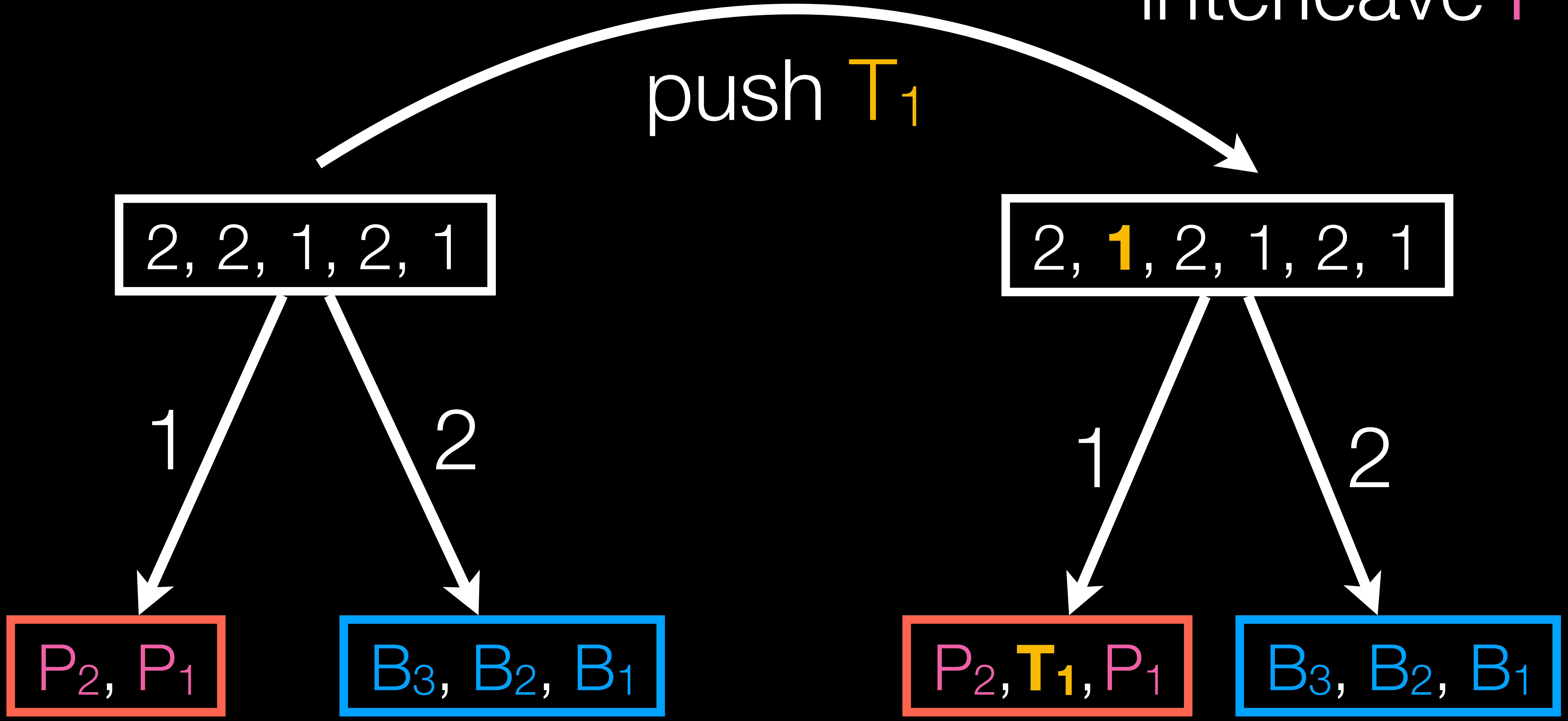
P_2, $\mathbf{T_1}$, P_1

B_3, B_2, B_1

B_3, B_2, P_2, B_1, P_1

Introducing: PIFO trees

interleave R and B;
interleave P and T.

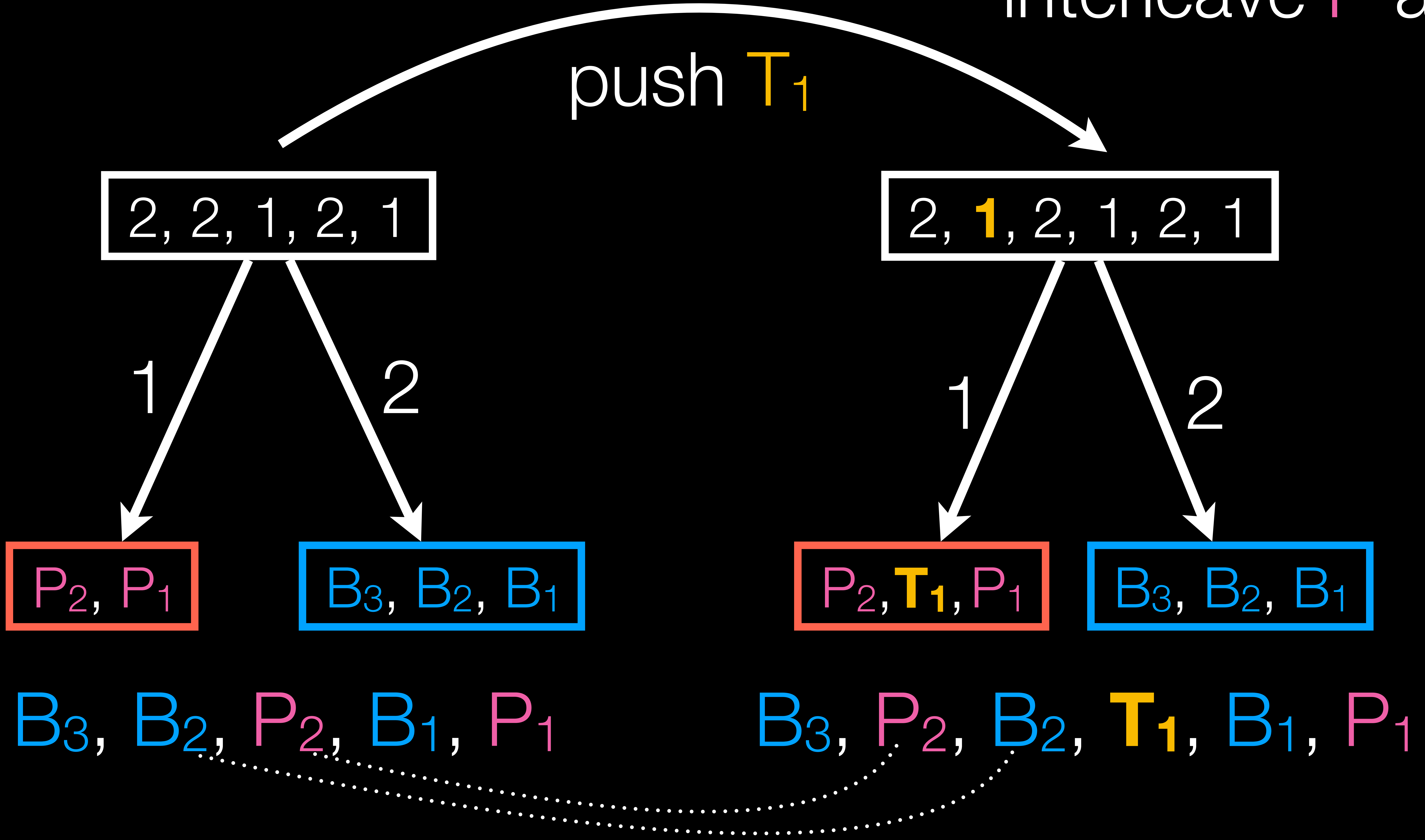


B_3, B_2, P_2, B_1, P_1

B_3, P_2, B_2, $\mathbf{T_1}$, B_1, P_1

Introducing: PIFO trees

interleave R and B;
interleave P and T.



push T_1

2, 2, 1, 2, 1

1

2

P_2, P_1

B_3, B_2, B_1

B_3, B_2, P_2, B_1, P_1

2, **1**, 2, 1, 2, 1

1

2

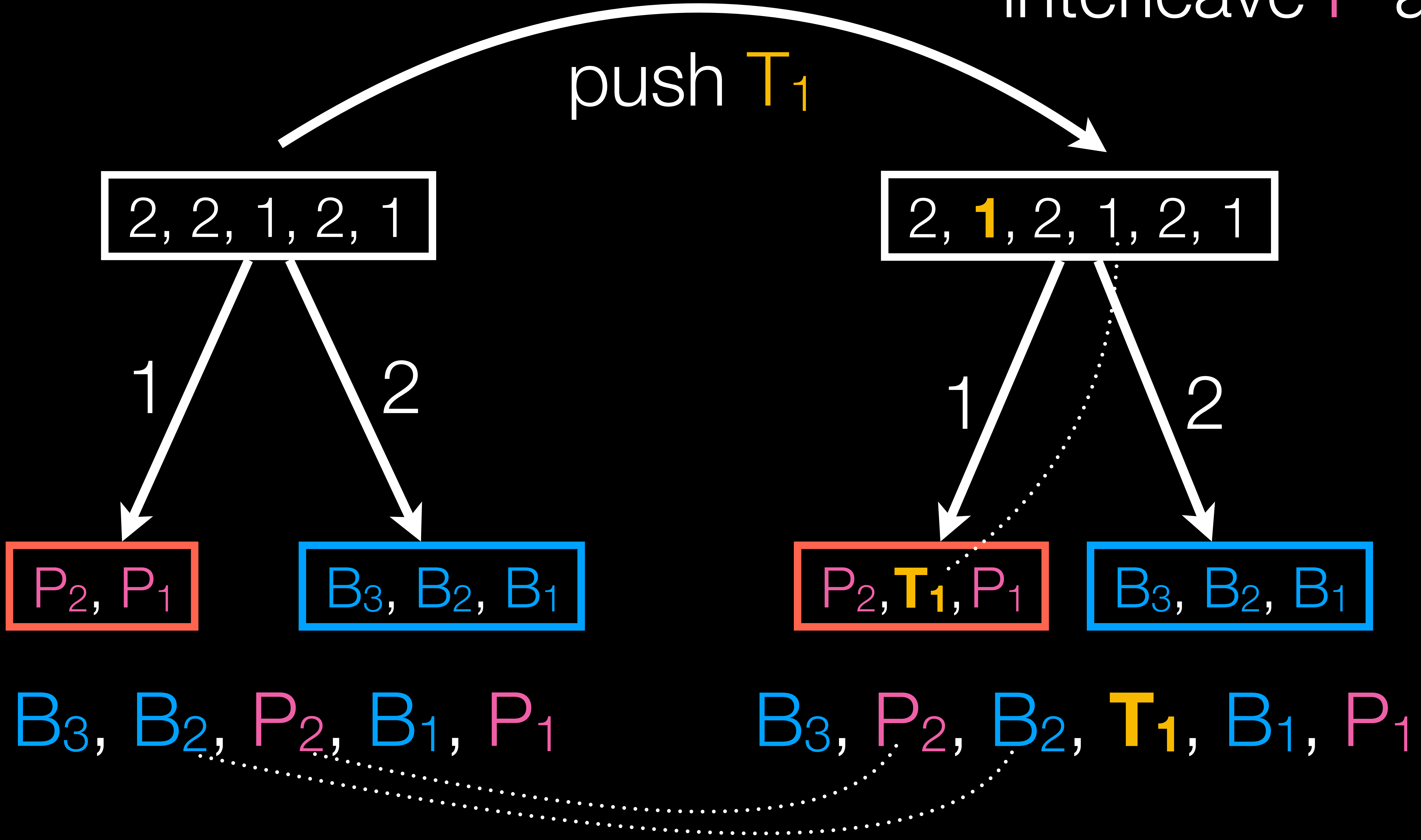
P_2, $\mathbf{T_1}$, P_1

B_3, B_2, B_1

B_3, P_2, B_2, $\mathbf{T_1}$, B_1, P_1

Introducing: PIFO trees

interleave R and B;
interleave P and T.



push T_1

2, 2, 1, 2, 1

1

2

P_2, P_1

B_3, B_2, B_1

B_3, B_2, P_2, B_1, P_1

2, **1**, 2, 1, 2, 1

1

2

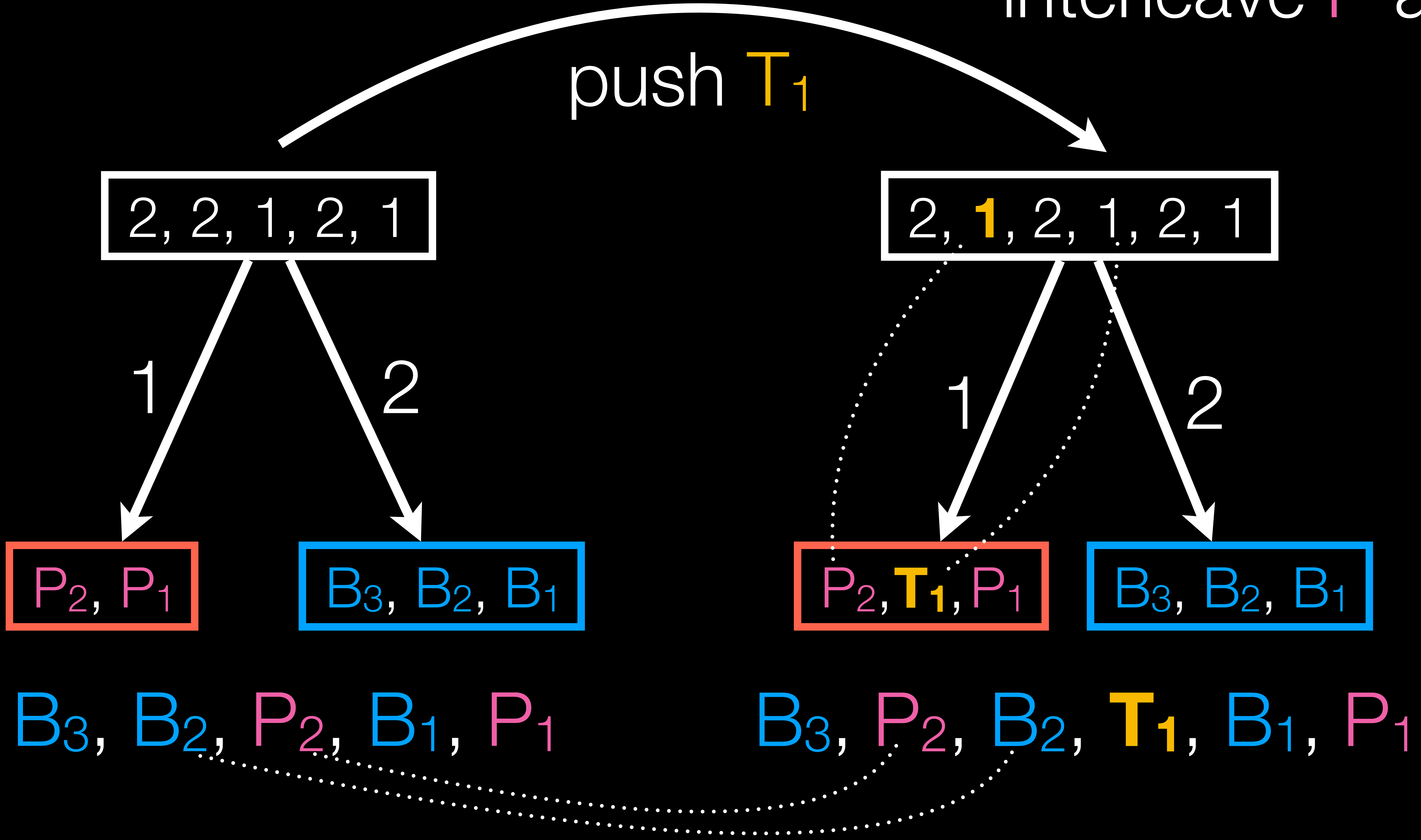
P_2, $\mathbf{T_1}$, P_1

B_3, B_2, B_1

B_3, P_2, B_2, $\mathbf{T_1}$, B_1, P_1

Introducing: PIFO trees

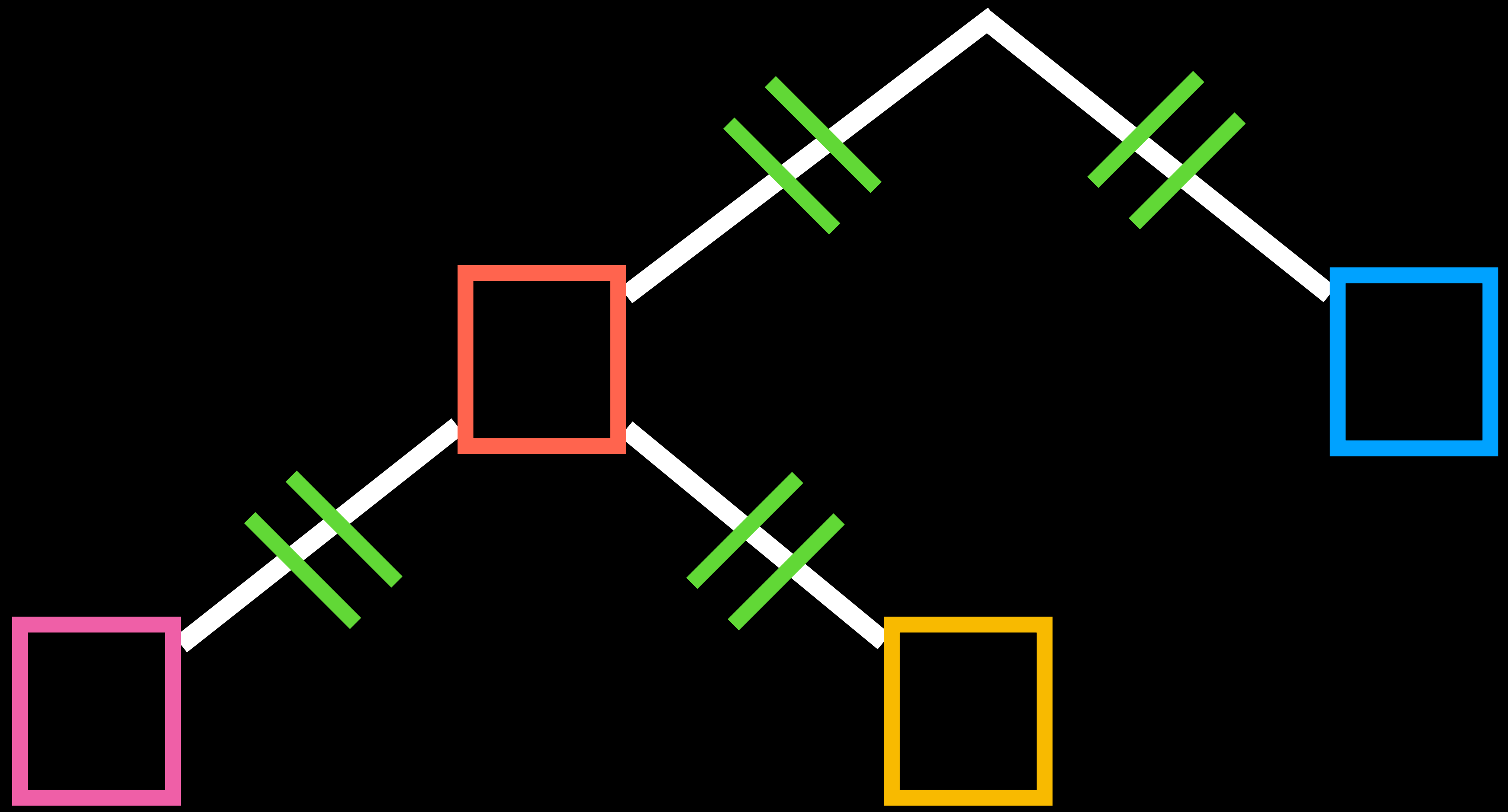
interleave R and B;
interleave P and T.

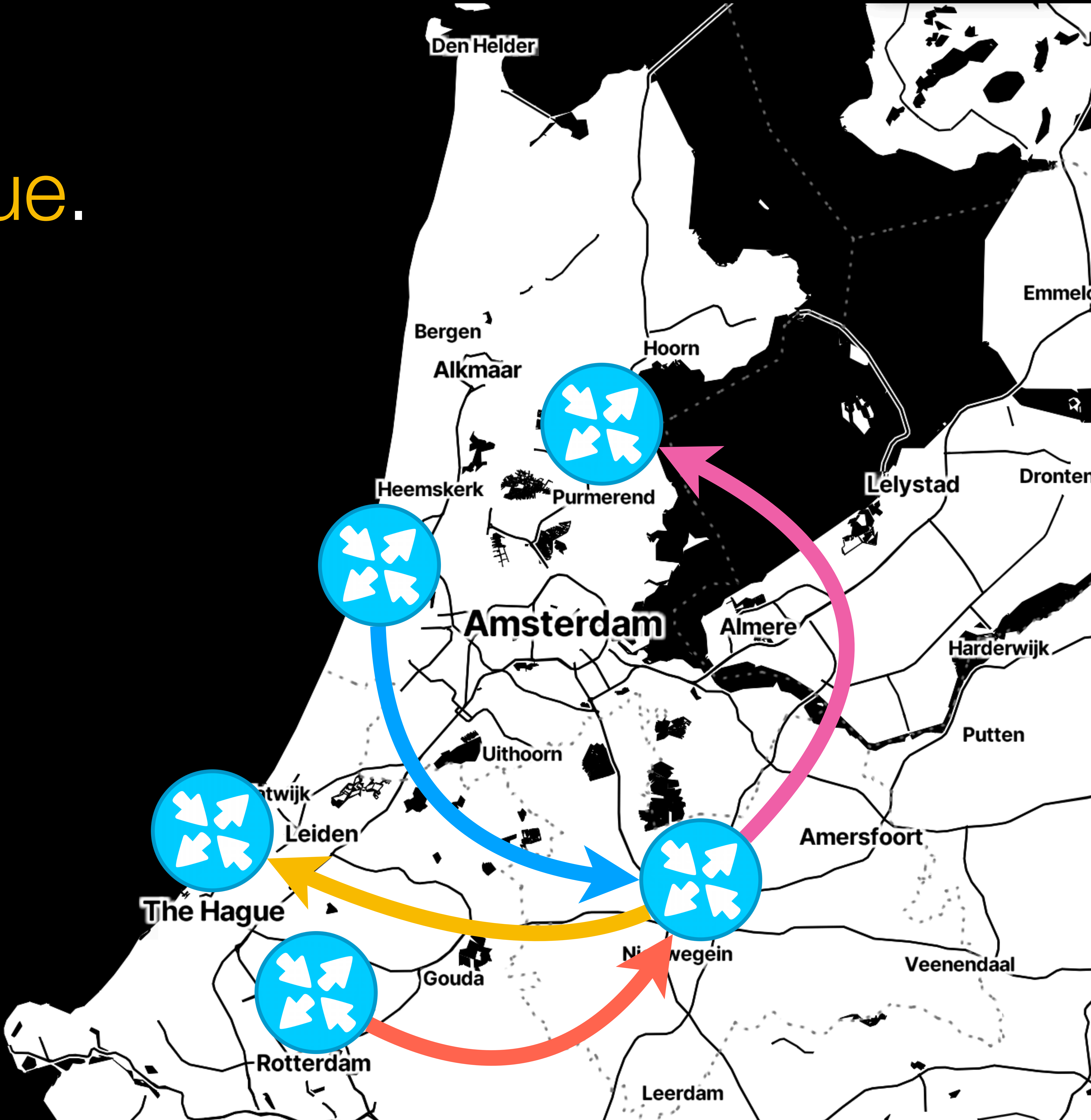

push T_1
2, 2, 1, 2, 1
1
2
P_2, P_1
B_3, B_2, B_1
2, **1**, 2, 1, 2, 1
1
2
P_2, $\mathbf{T_1}$, P_1
B_3, B_2, B_1
B_3, B_2, P_2, B_1, P_1
B_3, P_2, B_2, $\mathbf{T_1}$, B_1, P_1

R traffic goes to either Purmerend or The Hague.

Goal:

Interleave R and B;
interleave P and T.

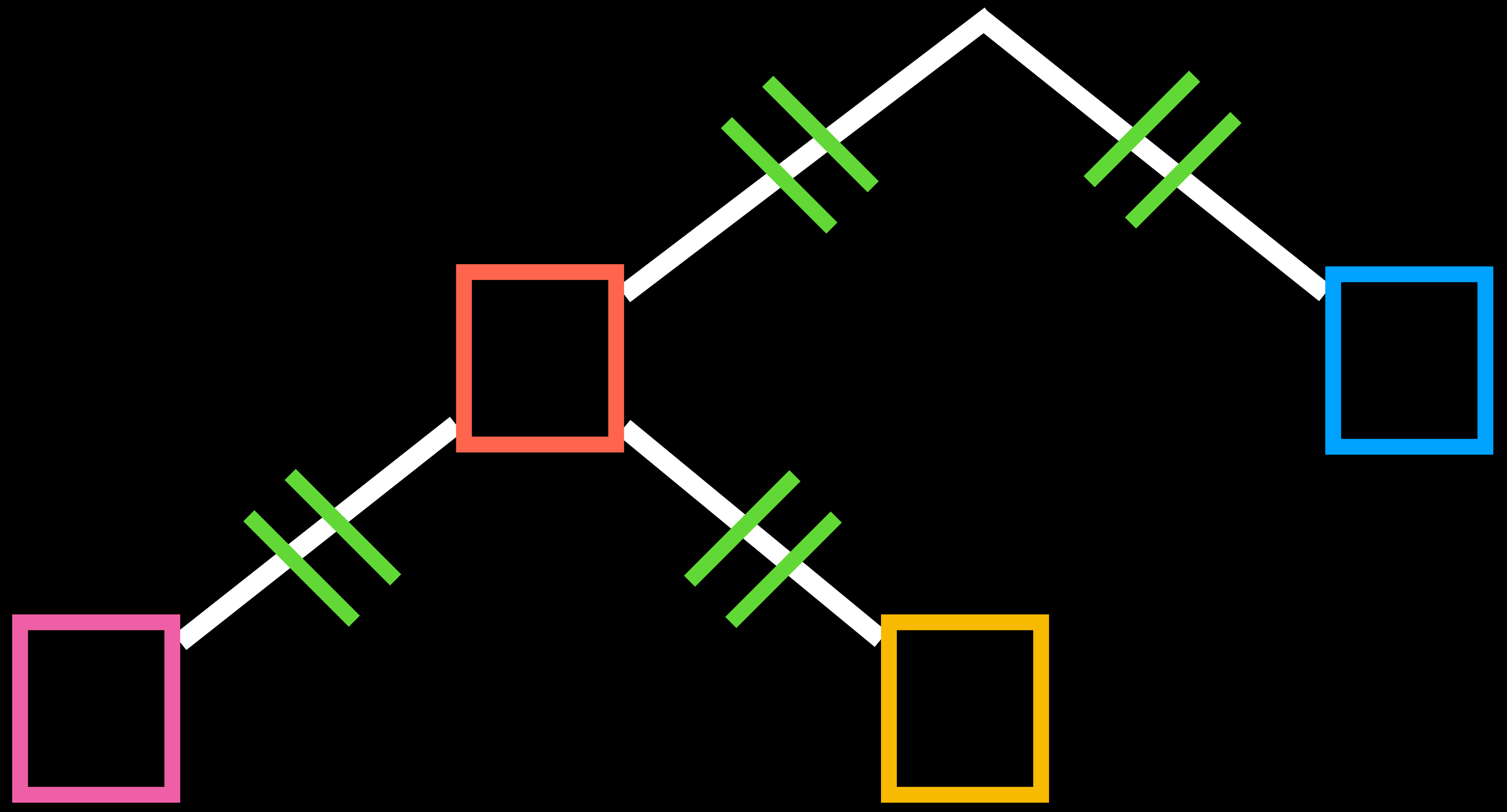


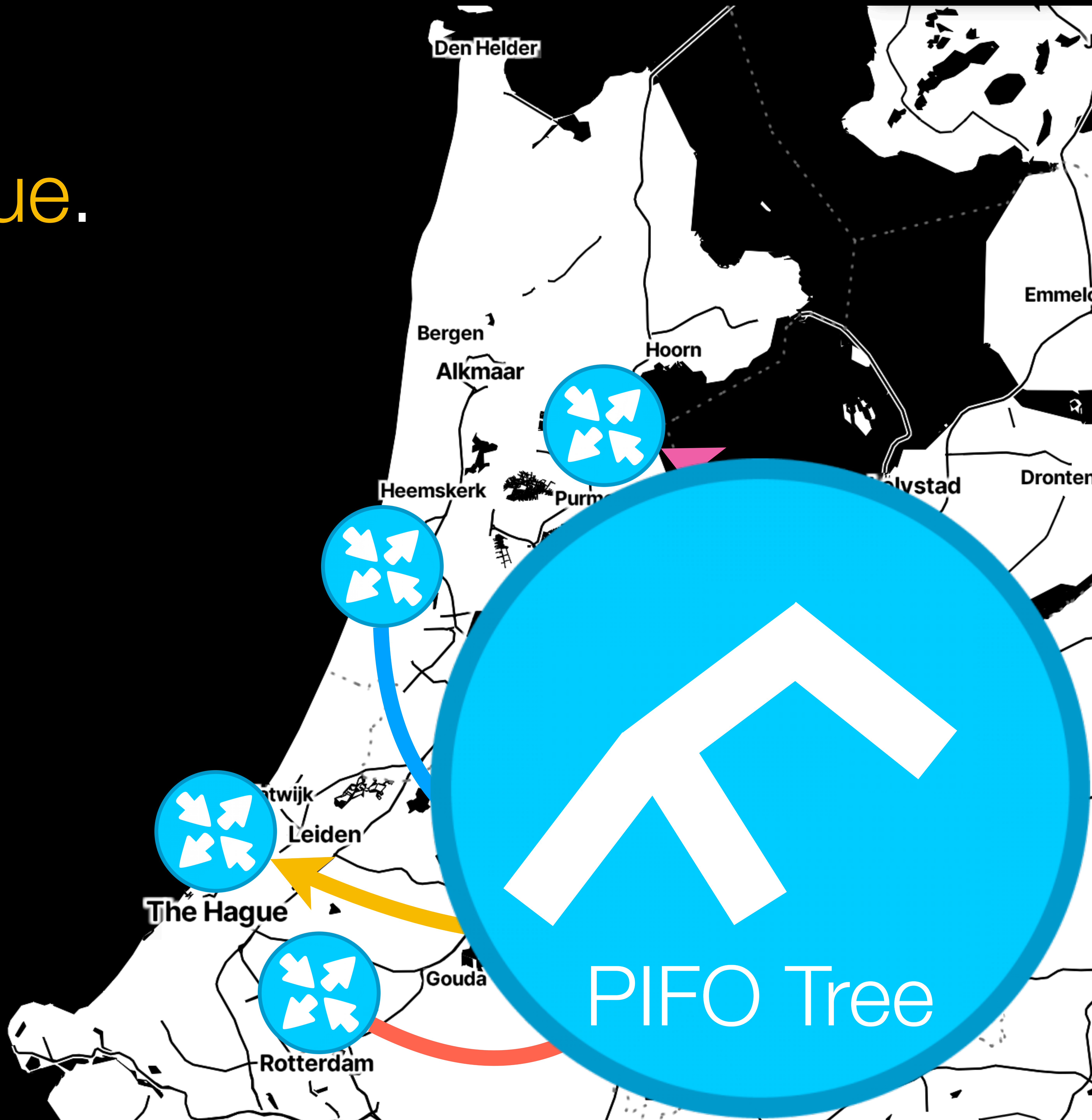


R traffic goes to either Purmerend or The Hague.

Goal:

Interleave R and B;
interleave P and T.



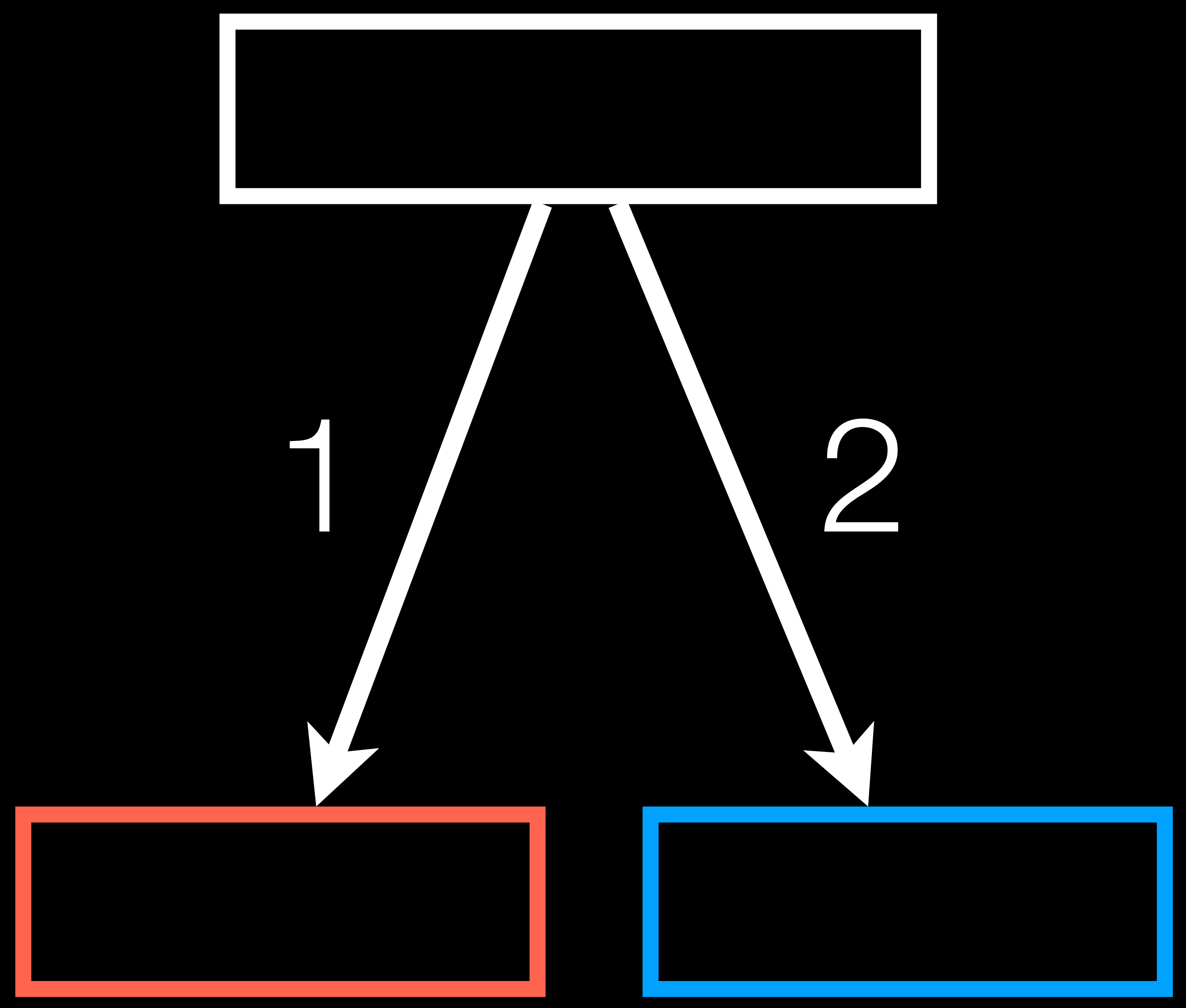


Aside: PIFO Trees

Sivaraman et al. at SIGCOMM ’16

Key Insight

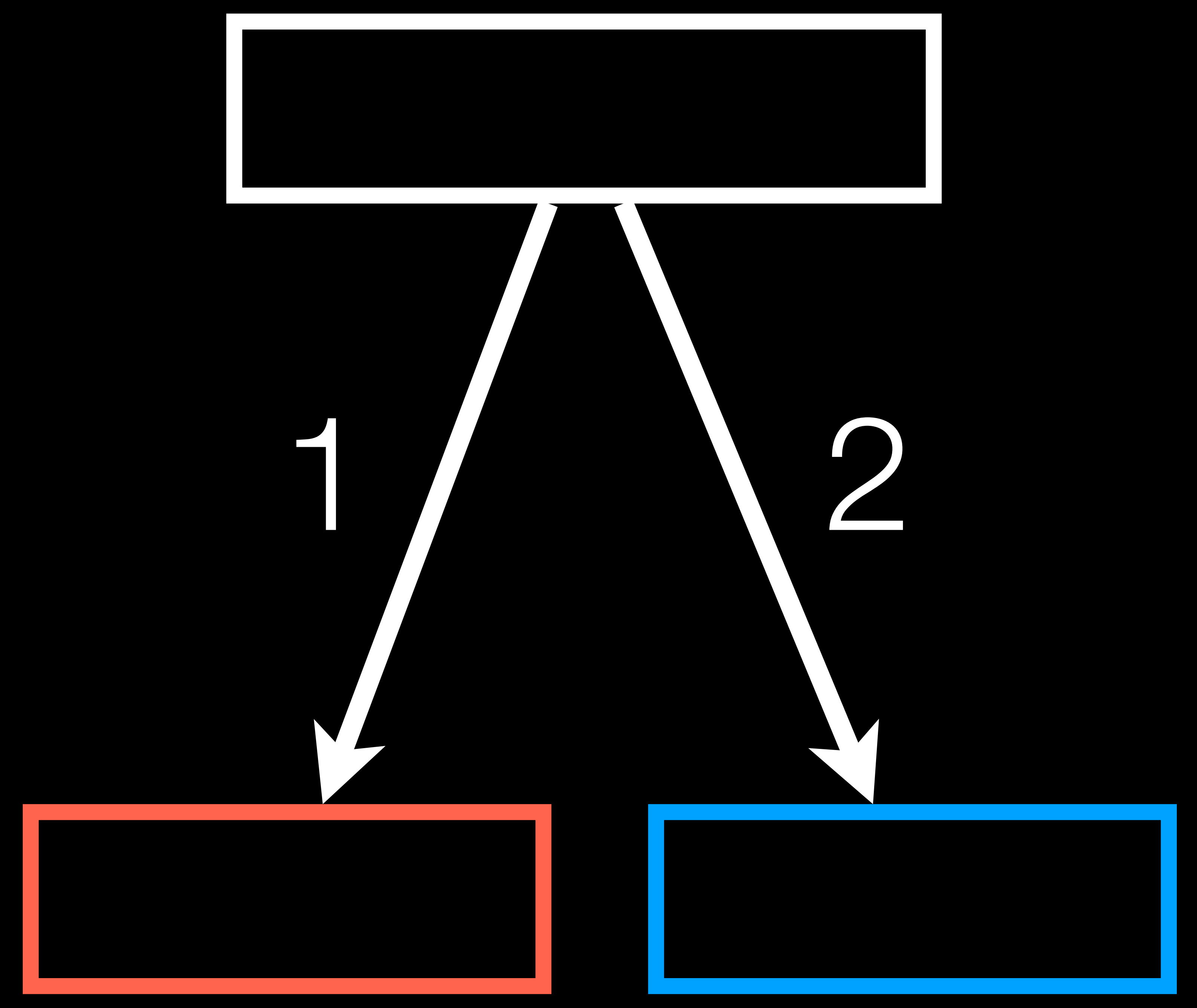
A PIFO tree manifests a *programming language.*



Key Insight

A PIFO tree manifests a *programming language.*

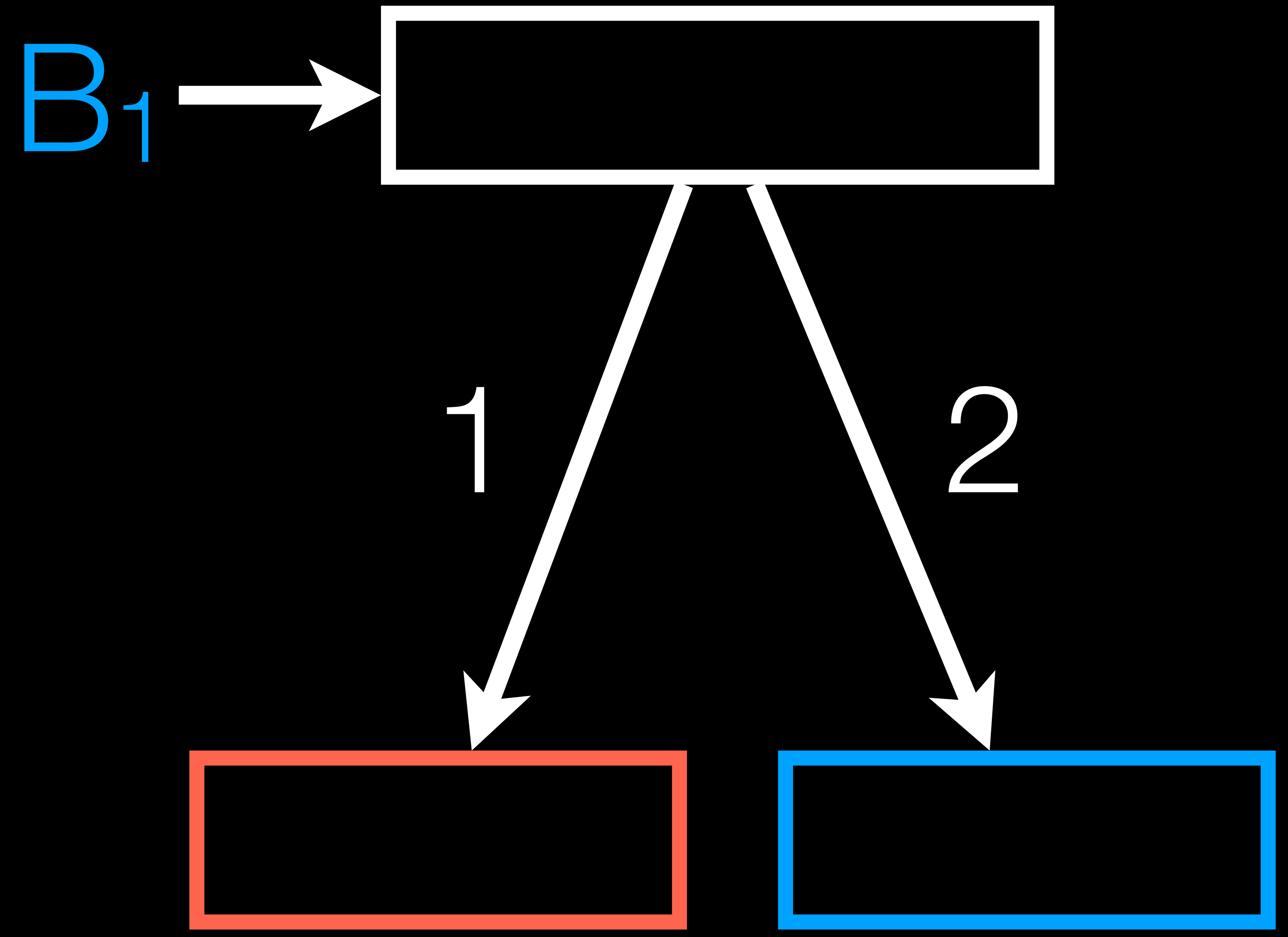
A program is precisely a *scheduling algorithm.*



Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

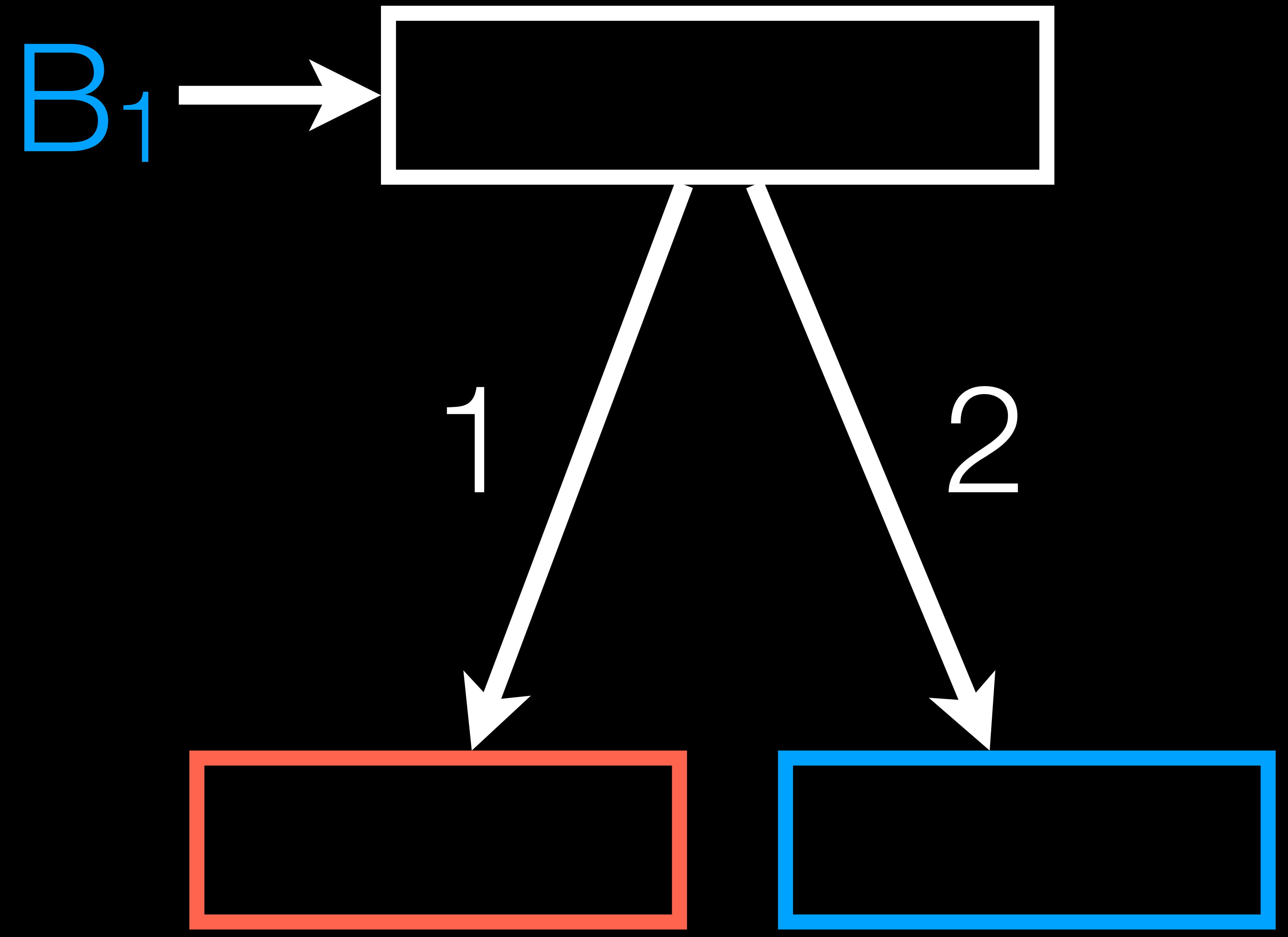


Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

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

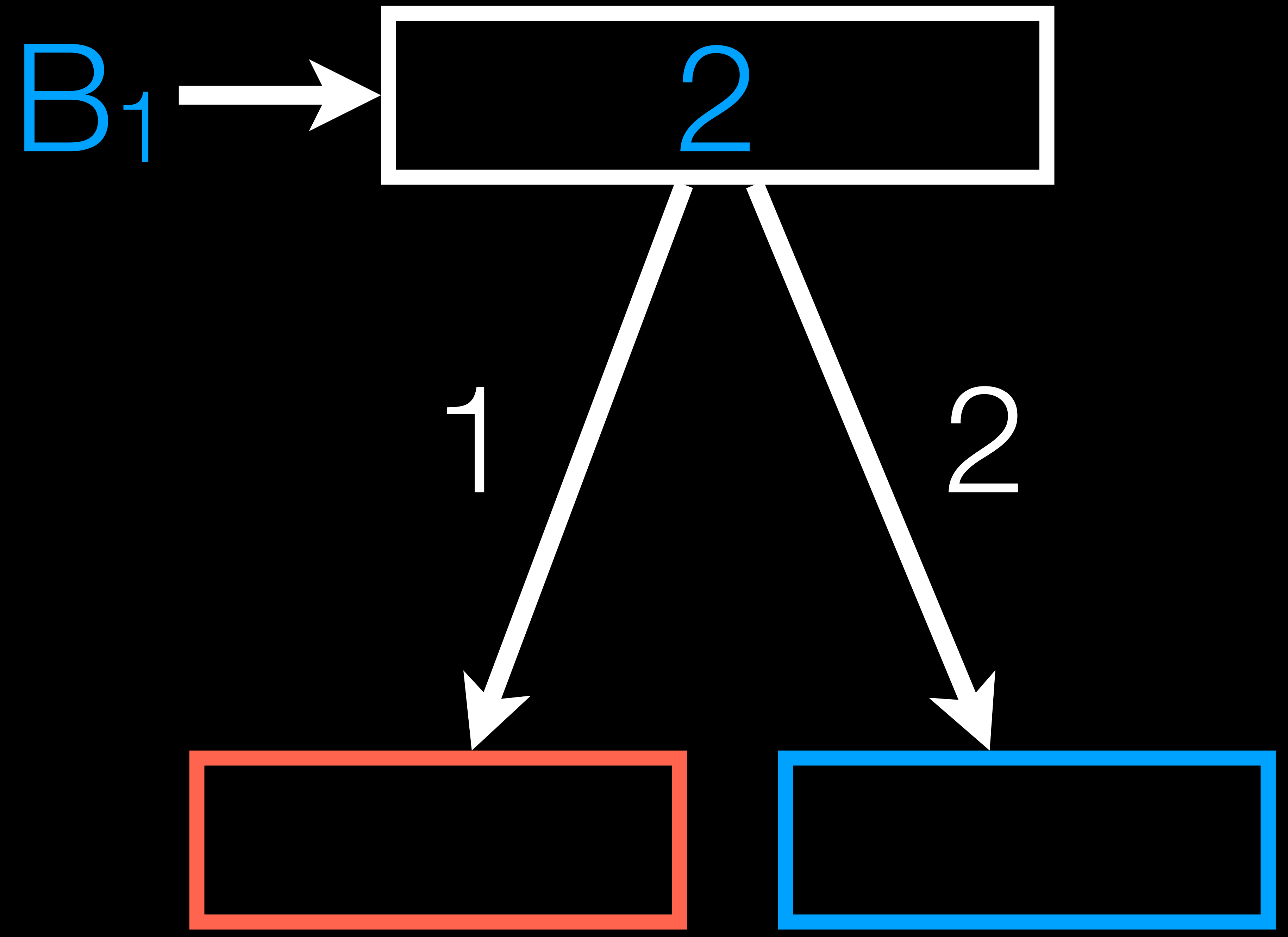


Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

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

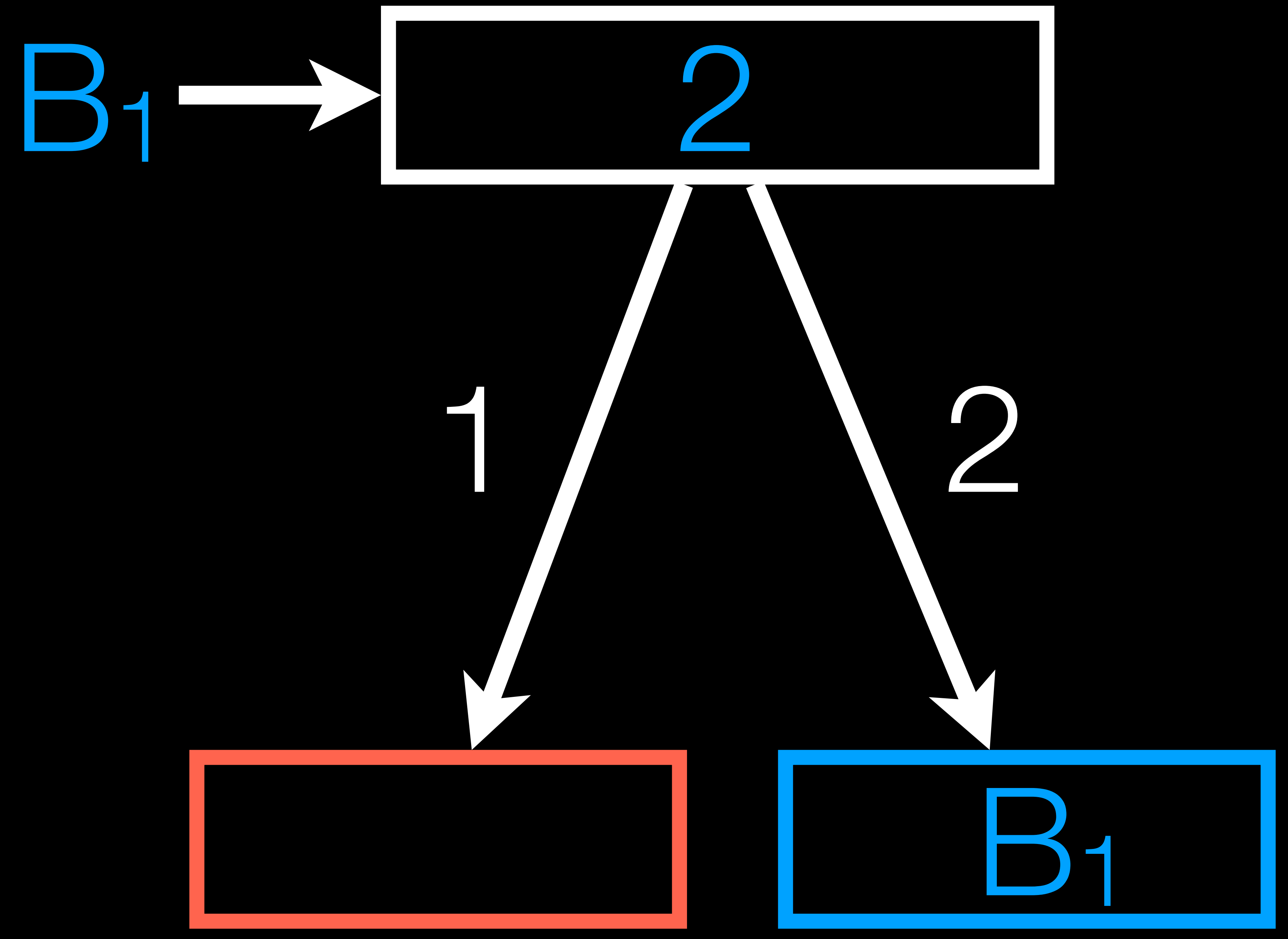


Key Insight

A PIFO tree manifests a *programming language.*

A program is precisely a *scheduling algorithm.*

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



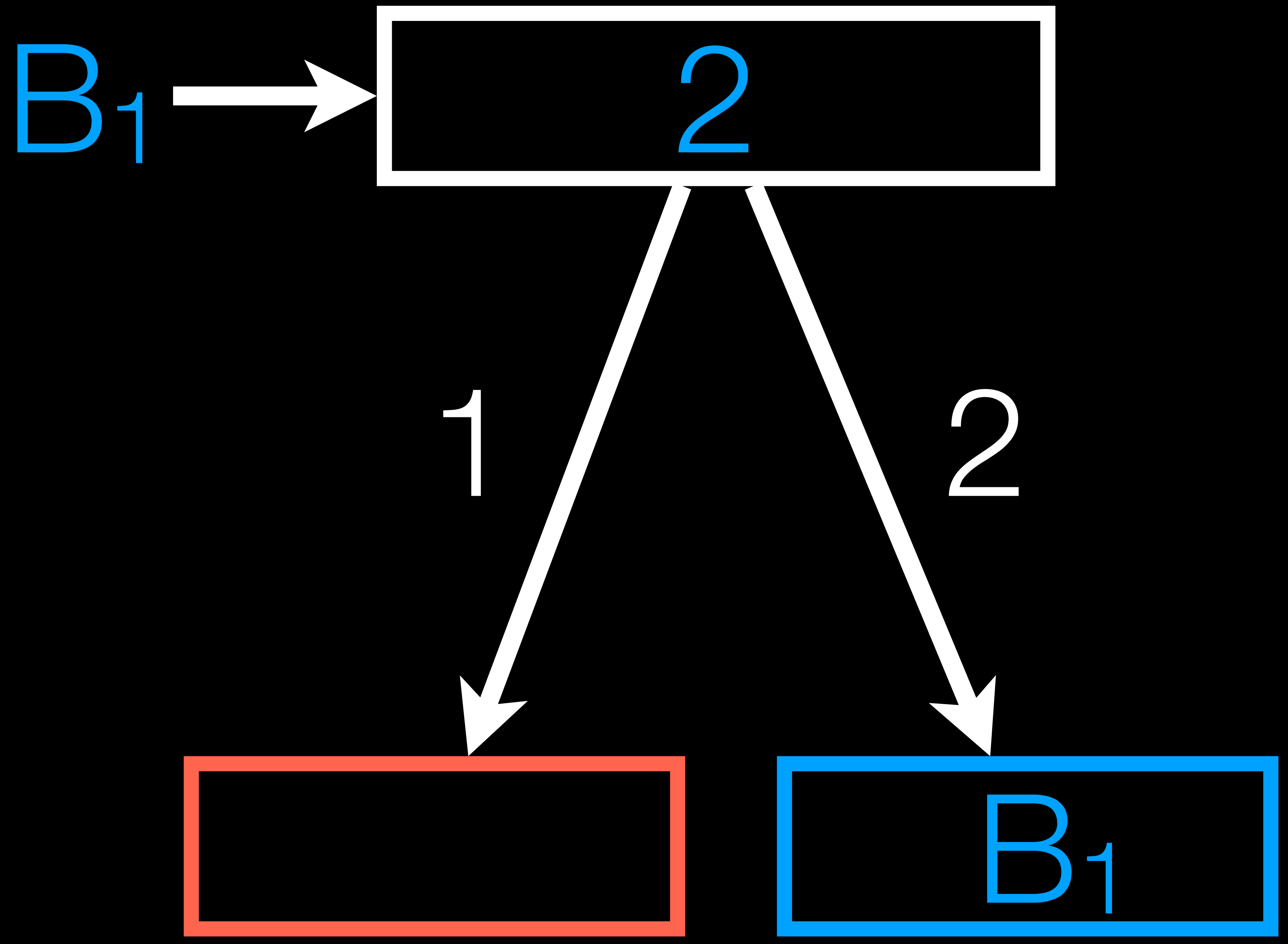
Key Insight

A PIFO tree manifests a *programming language.*

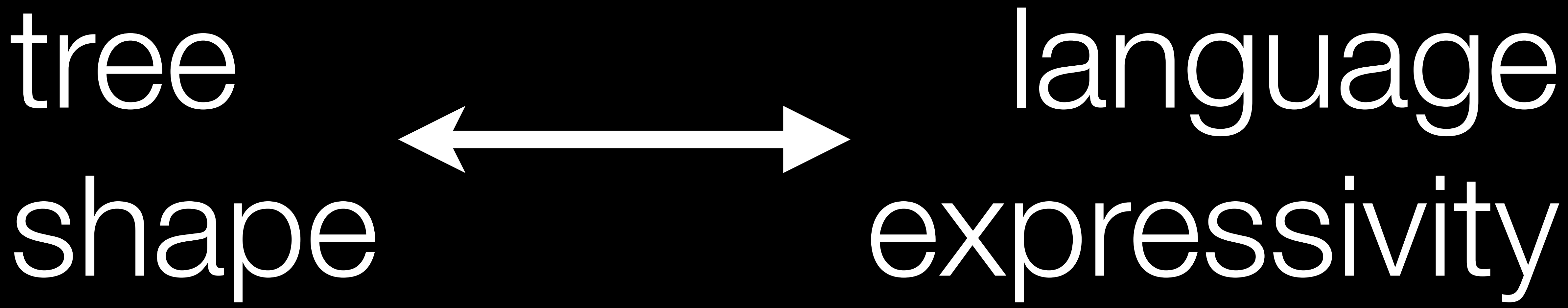
A program is precisely a *scheduling algorithm.*

tree shape ⟷ language expressivity

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



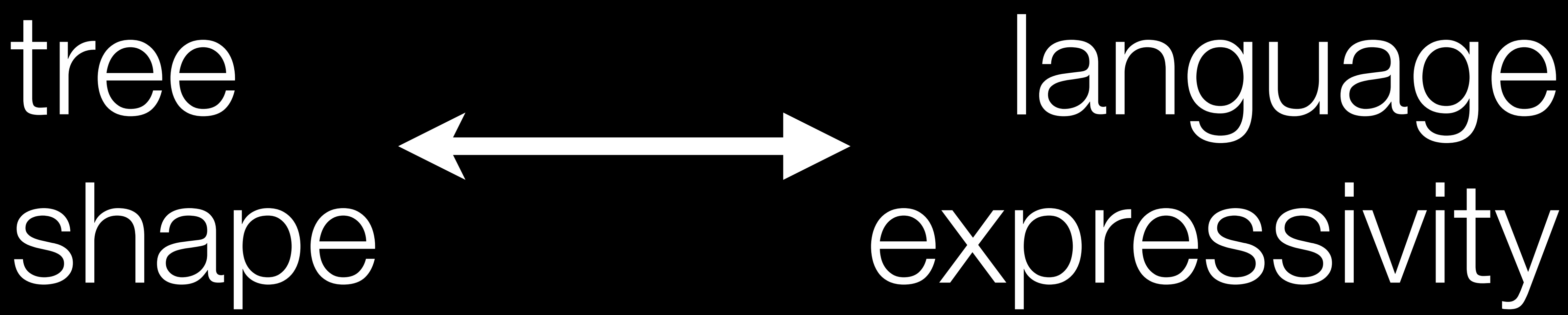
Which leads to some very PL-ey questions:



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*?

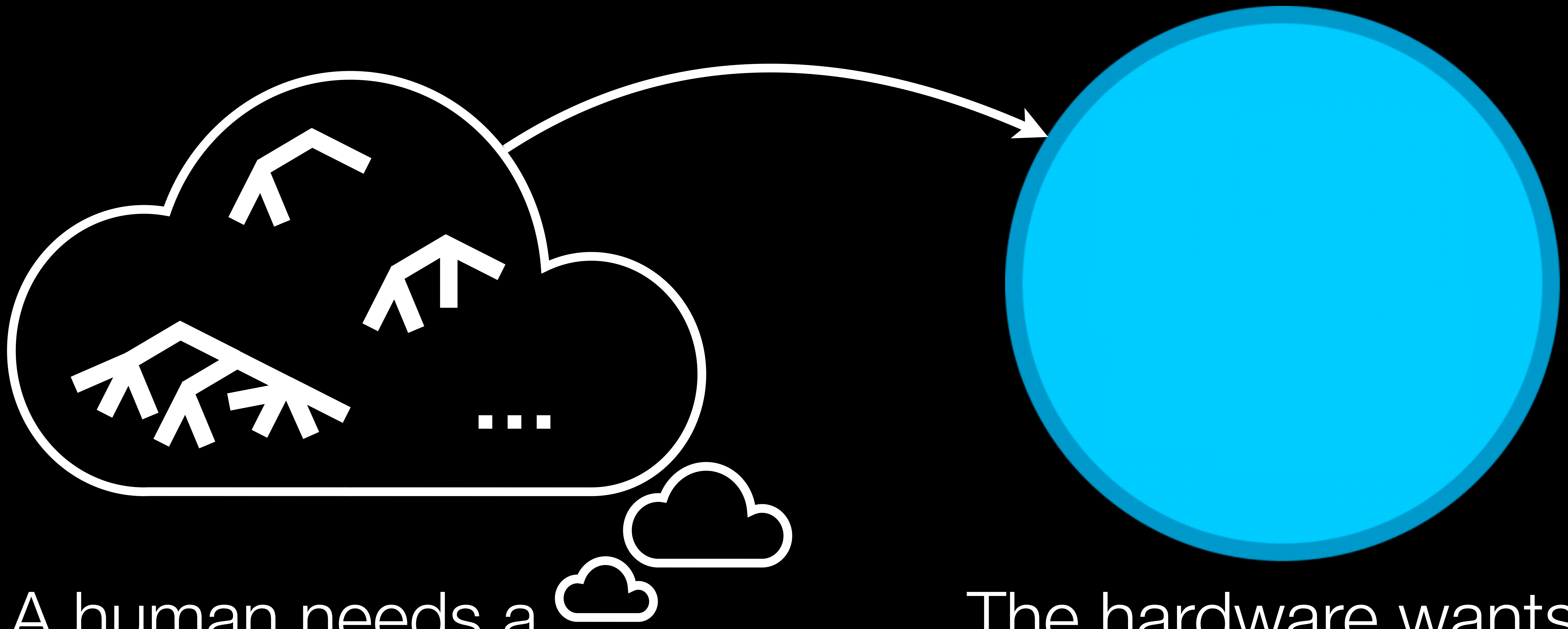
Which leads to some very PL-ey questions:

tree shape ↔ language expressivity

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

Compile a program so it runs against a new tree?

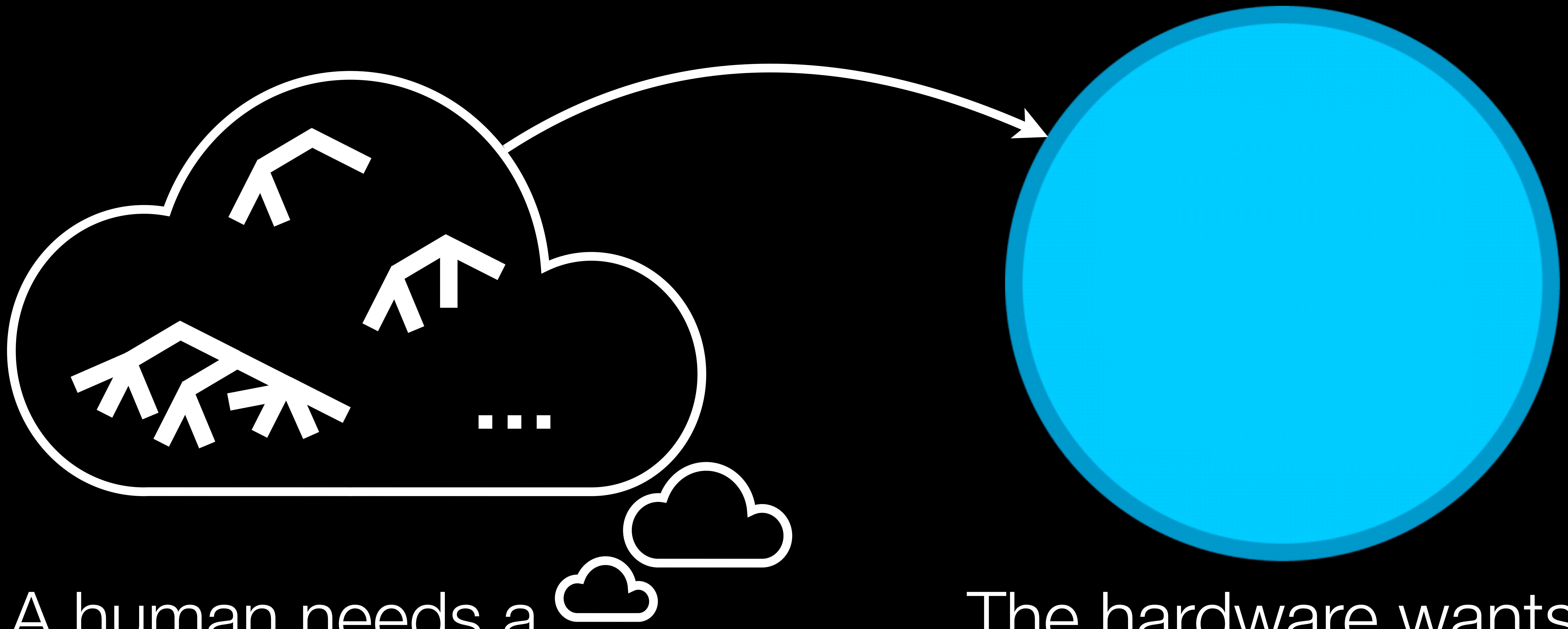
No general way to deploy our gadget.



A human needs a *range* of trees.

The hardware wants to support *one* tree.

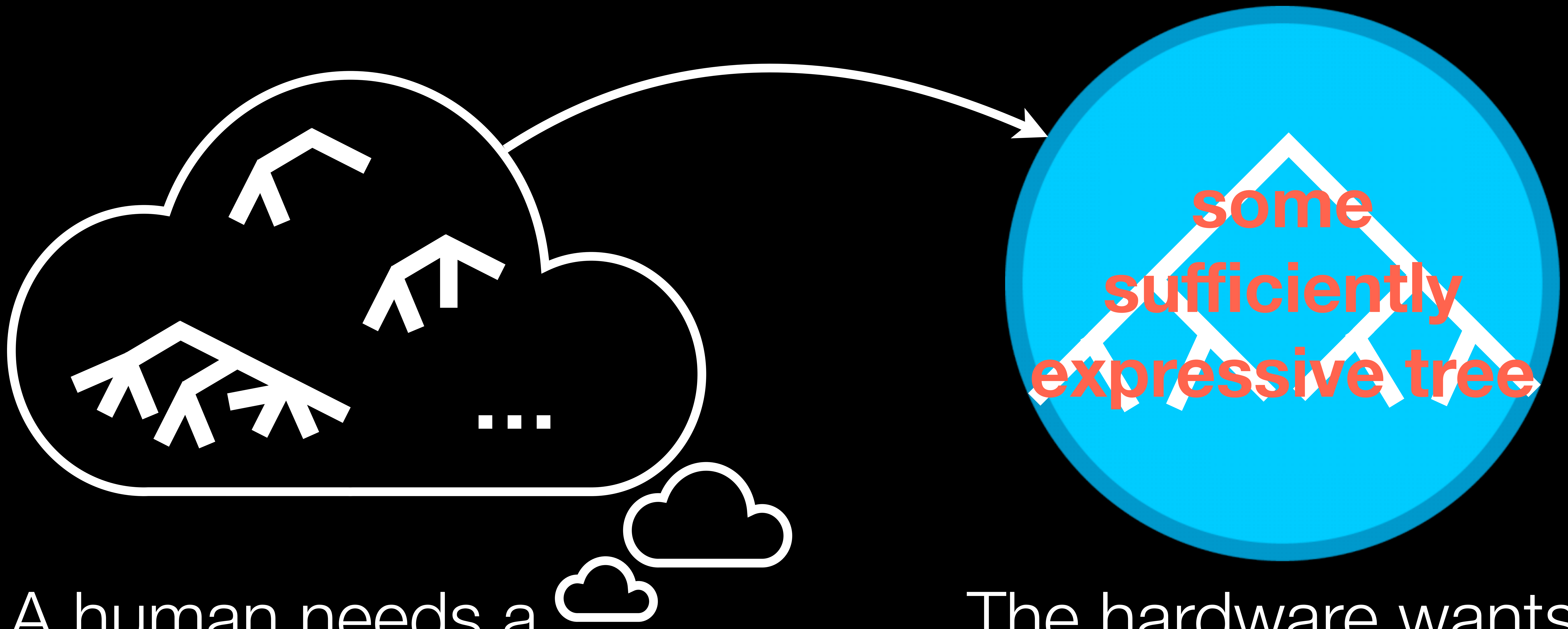
~~No general way to deploy our gadget.~~



A human needs a *range* of trees.

The hardware wants to support *one* tree.

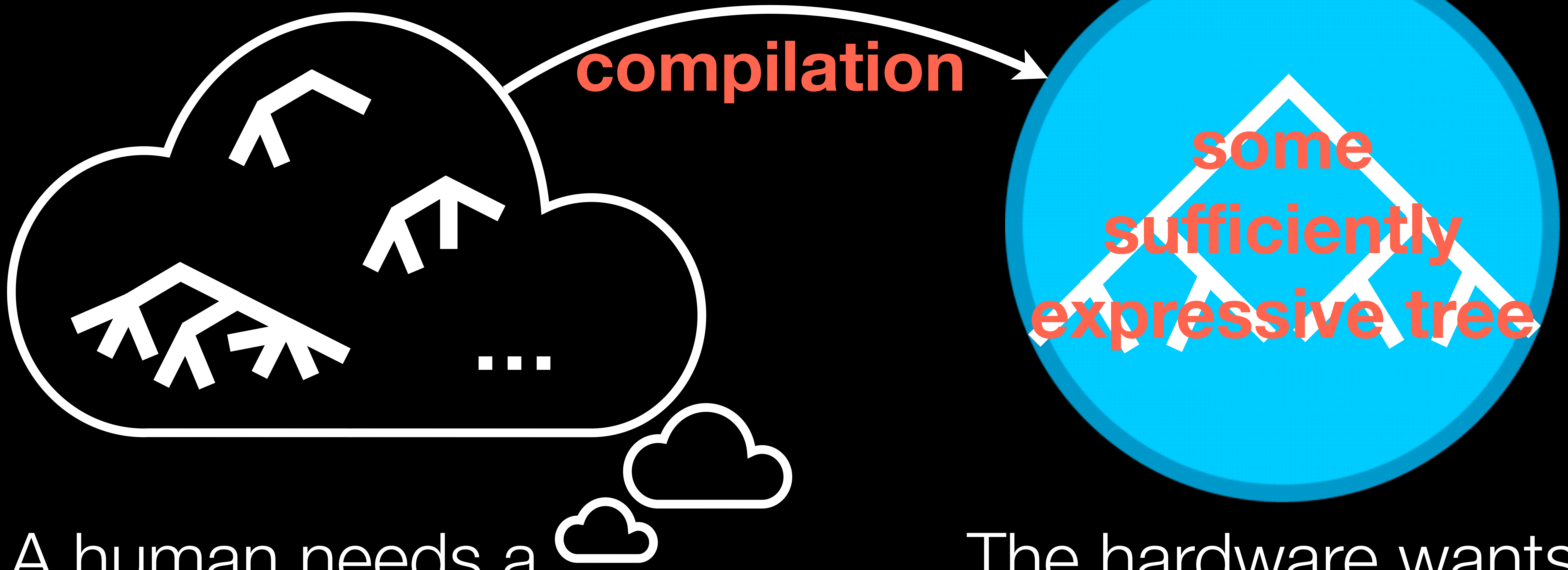
~~No general way to deploy our gadget.~~



A human needs a *range* of trees.

The hardware wants to support *one* tree.

~~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:

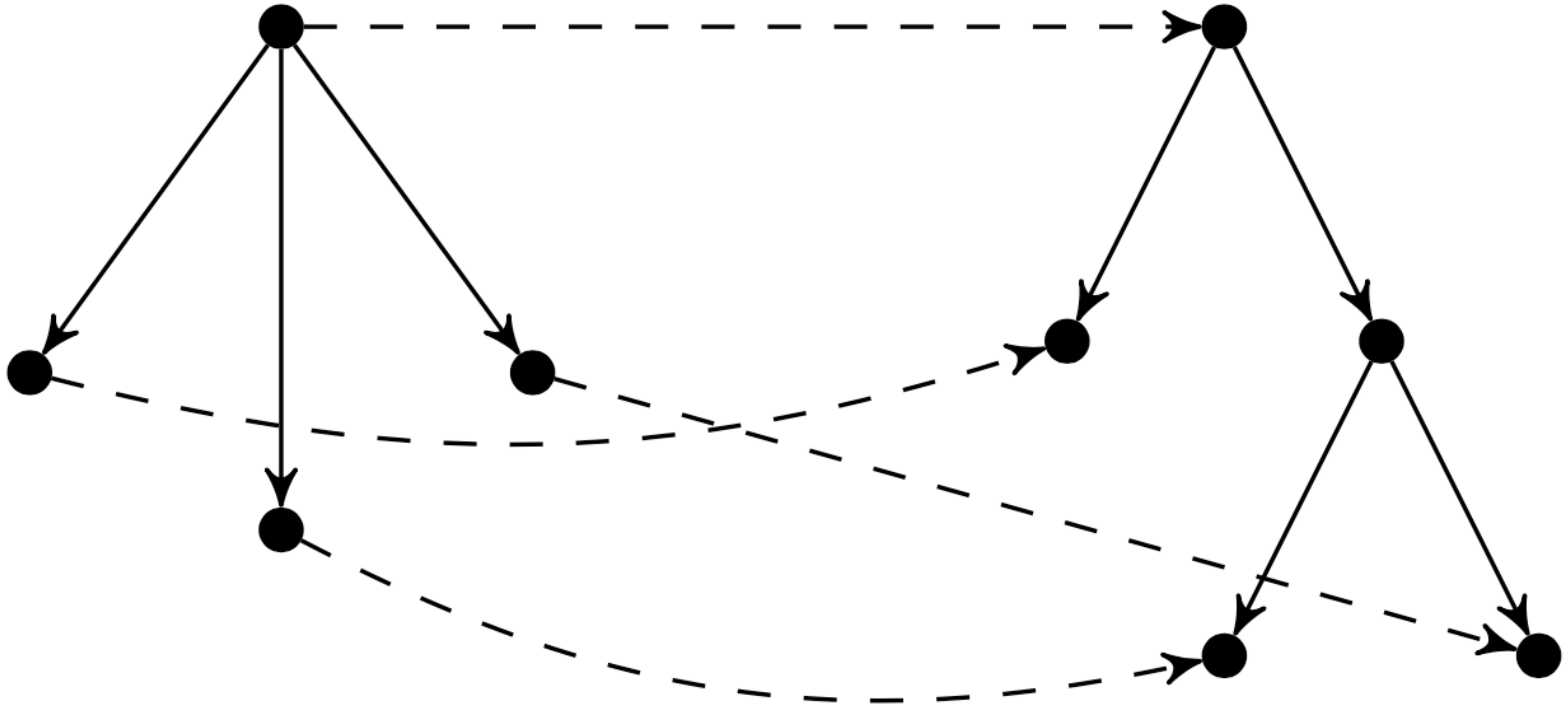
Expressivity of trees

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.

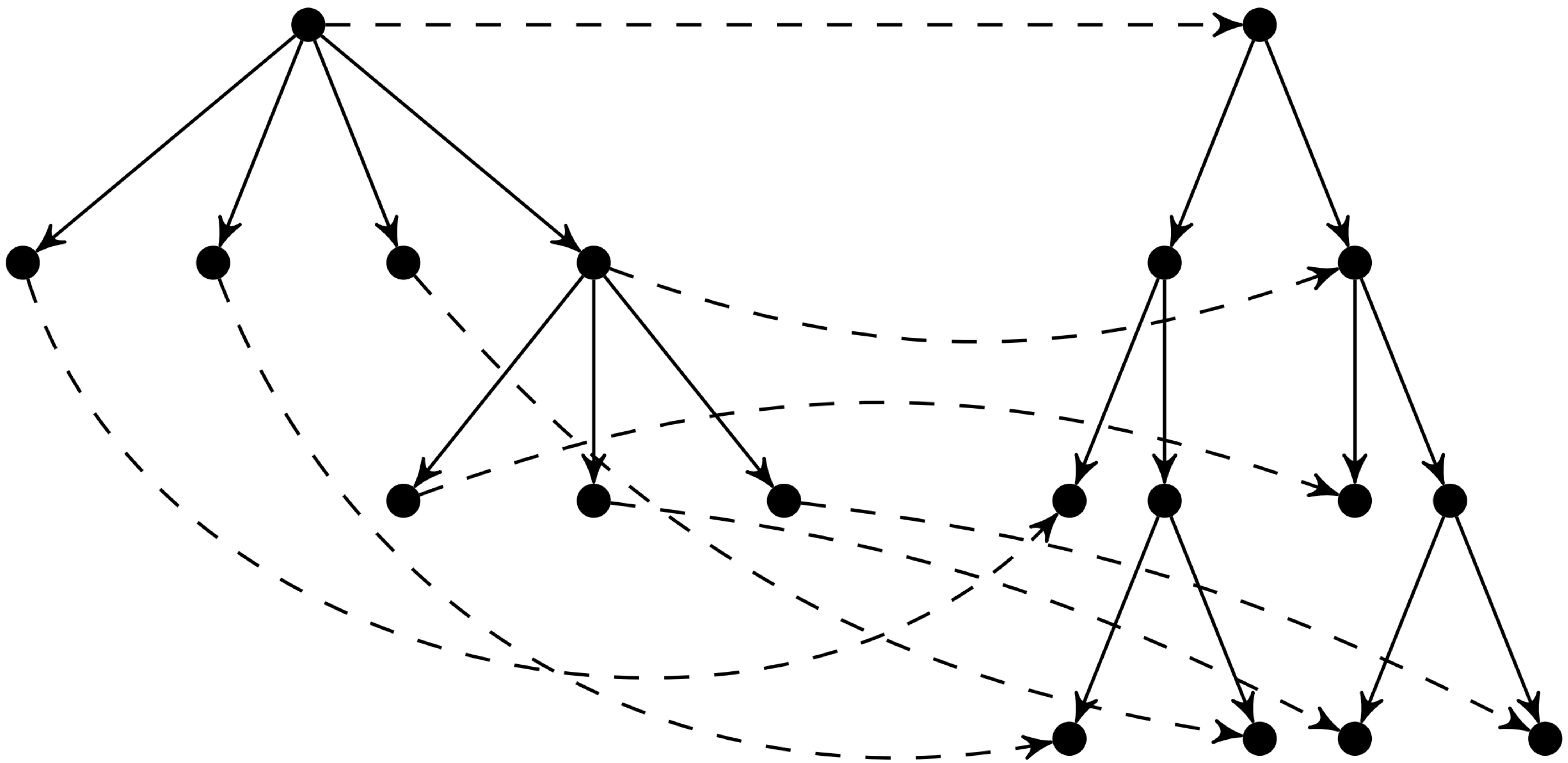
Expressivity of trees



Homomorphic embedding.

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

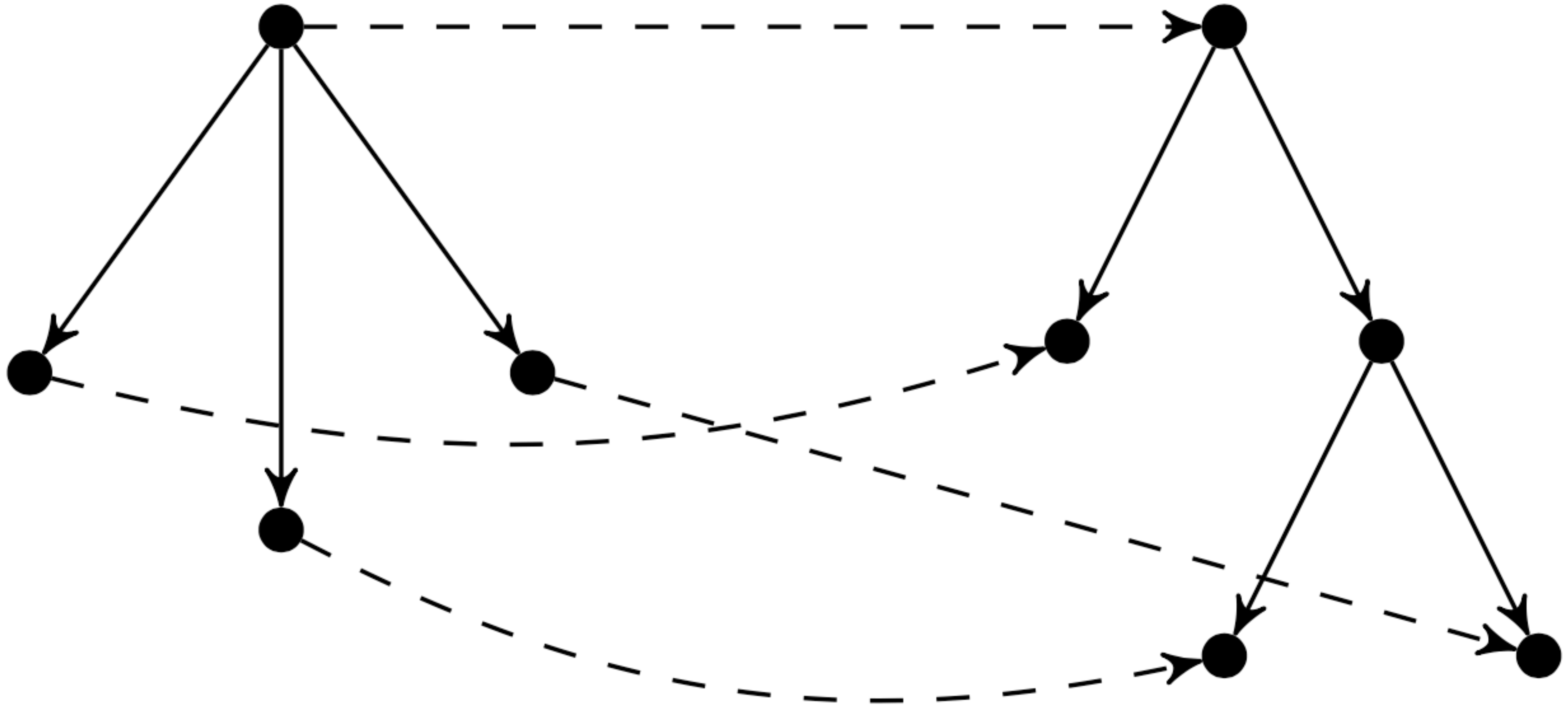
Expressivity of trees



Homomorphic embedding.

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

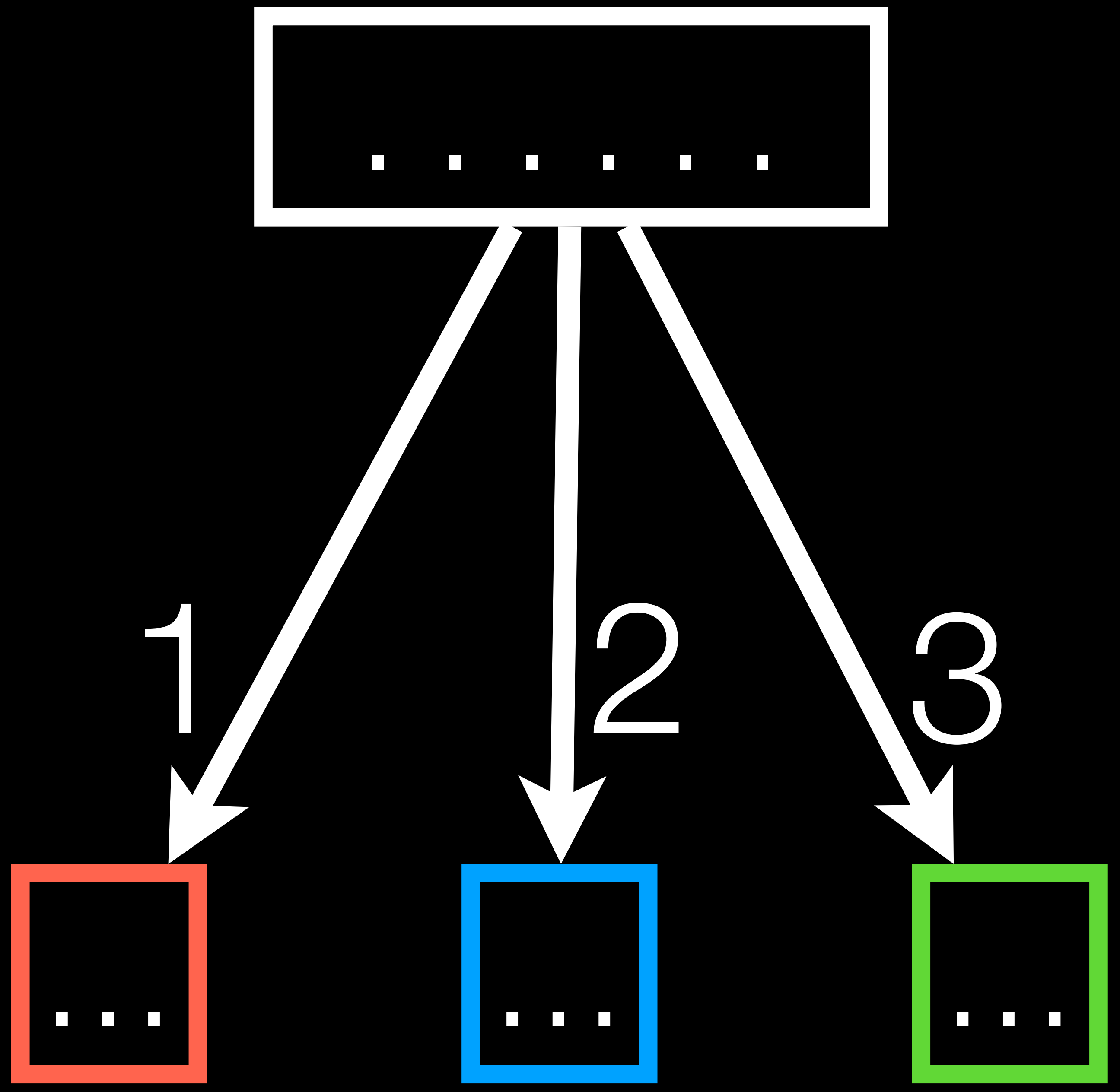
Expressivity of trees

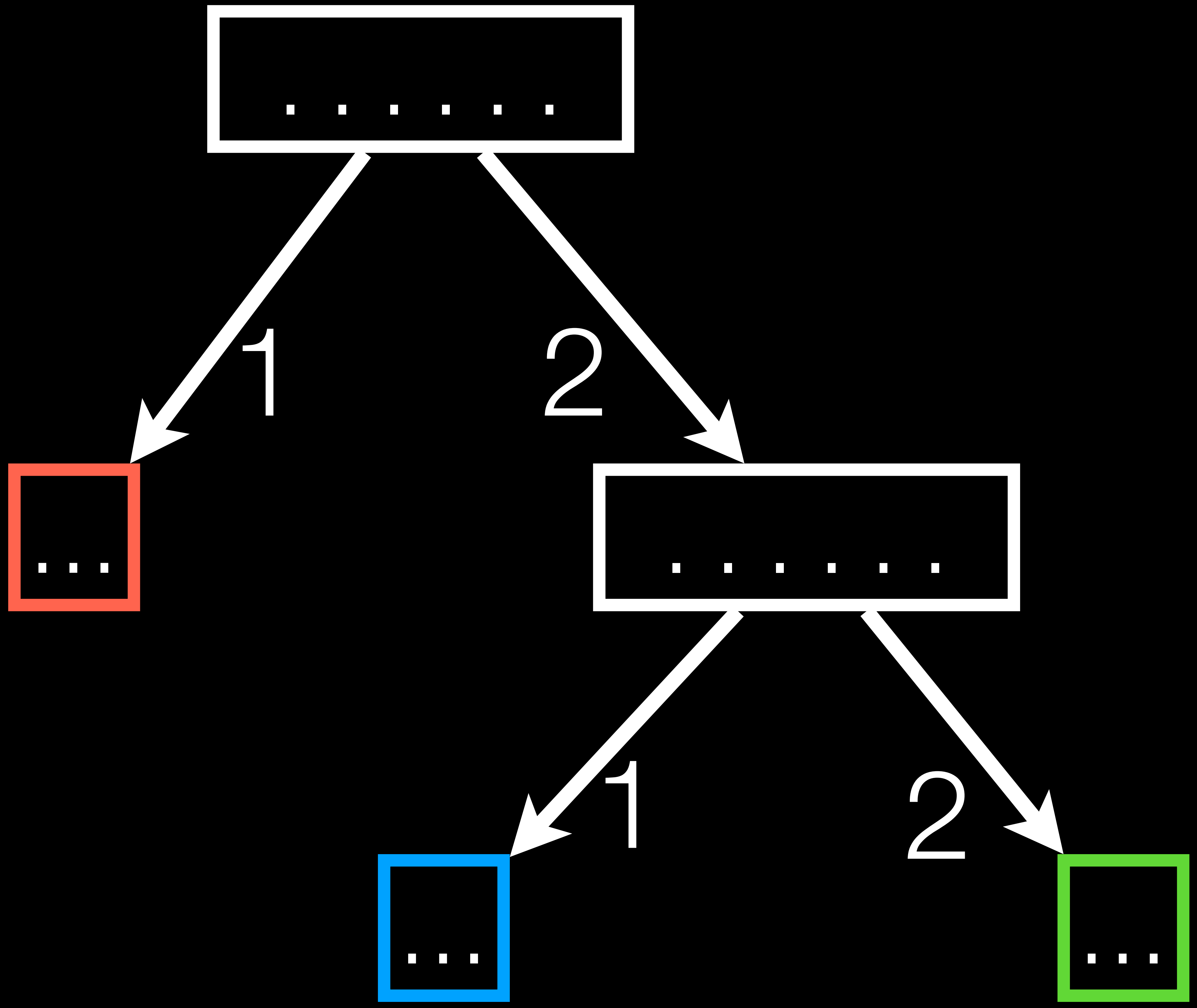


Homomorphic embedding.

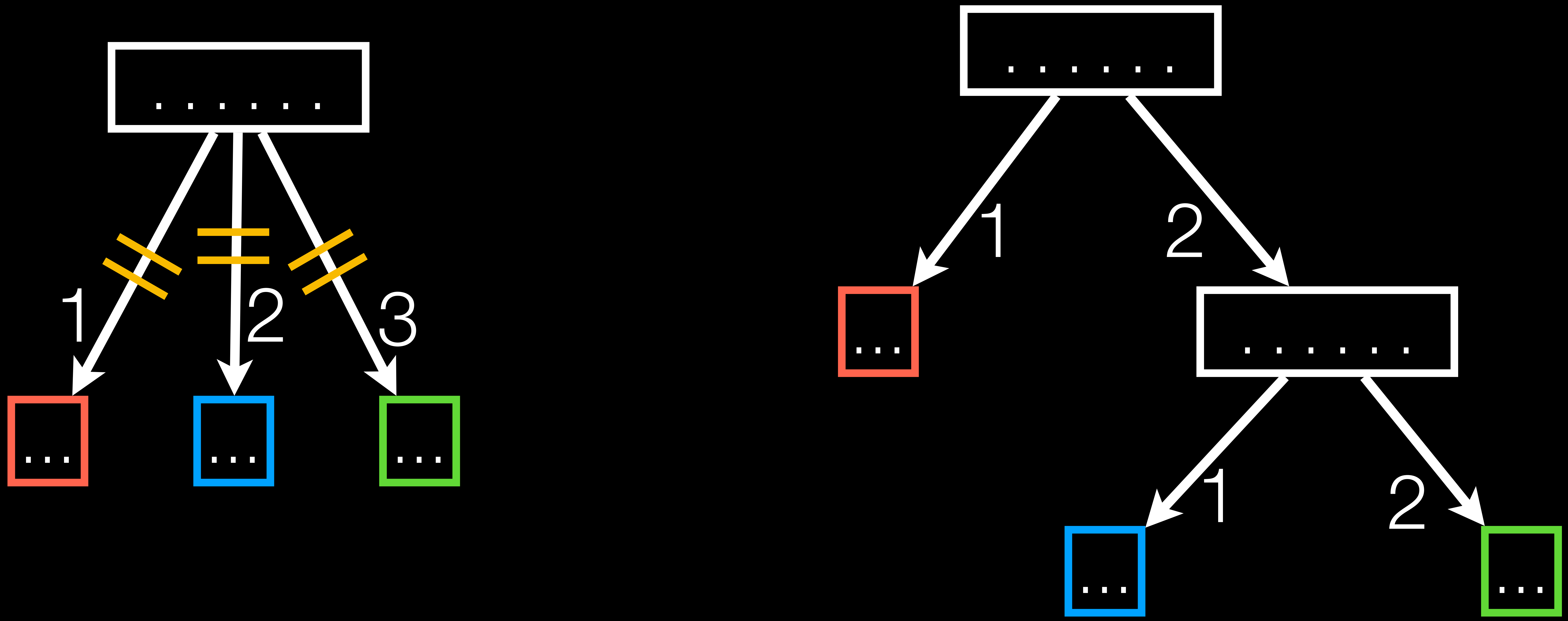
Map root to root, leaves to leaves. Respect ancestry.

Compiling programs

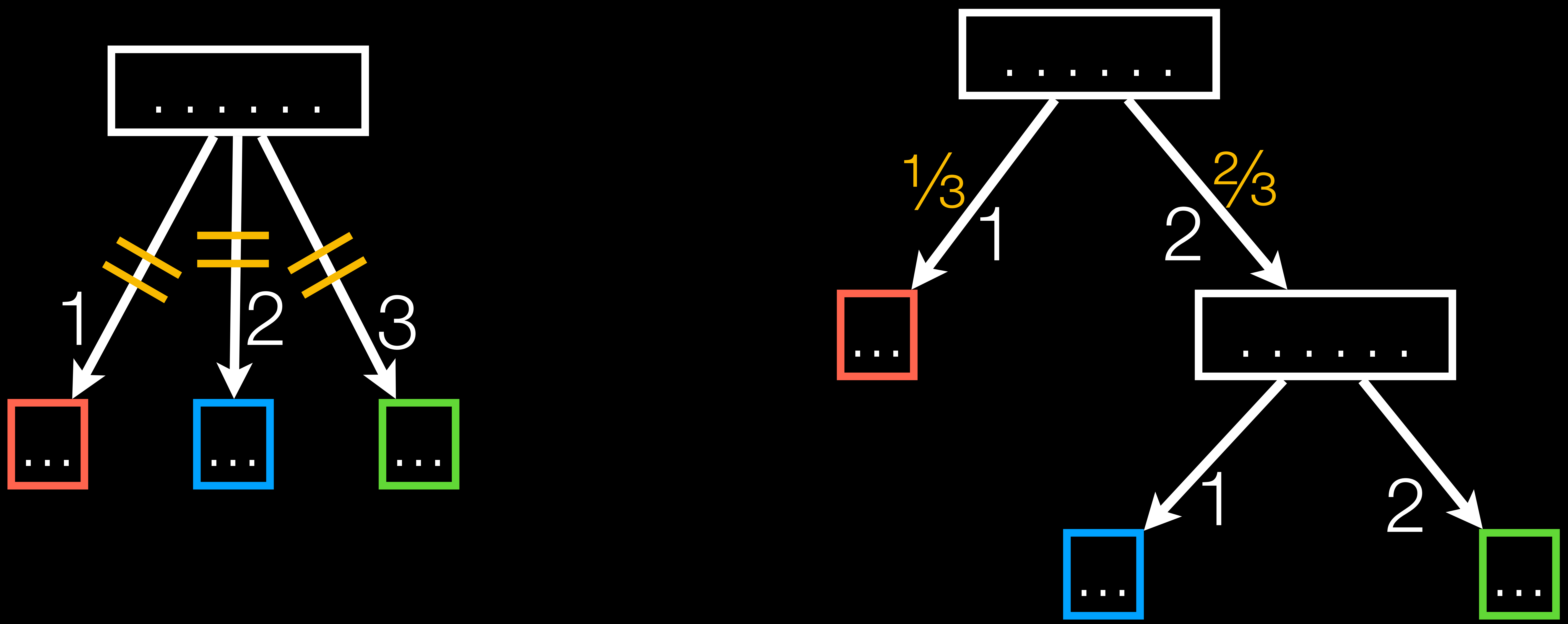




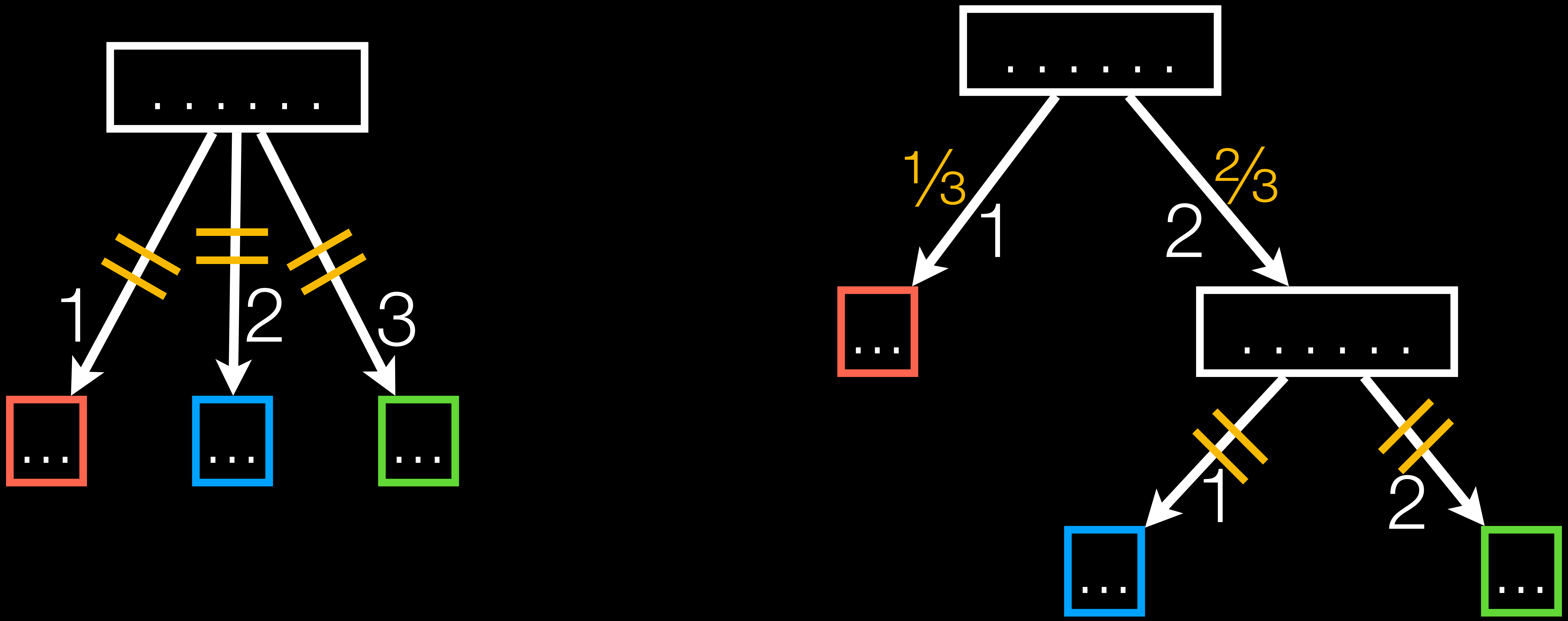
Compiling programs



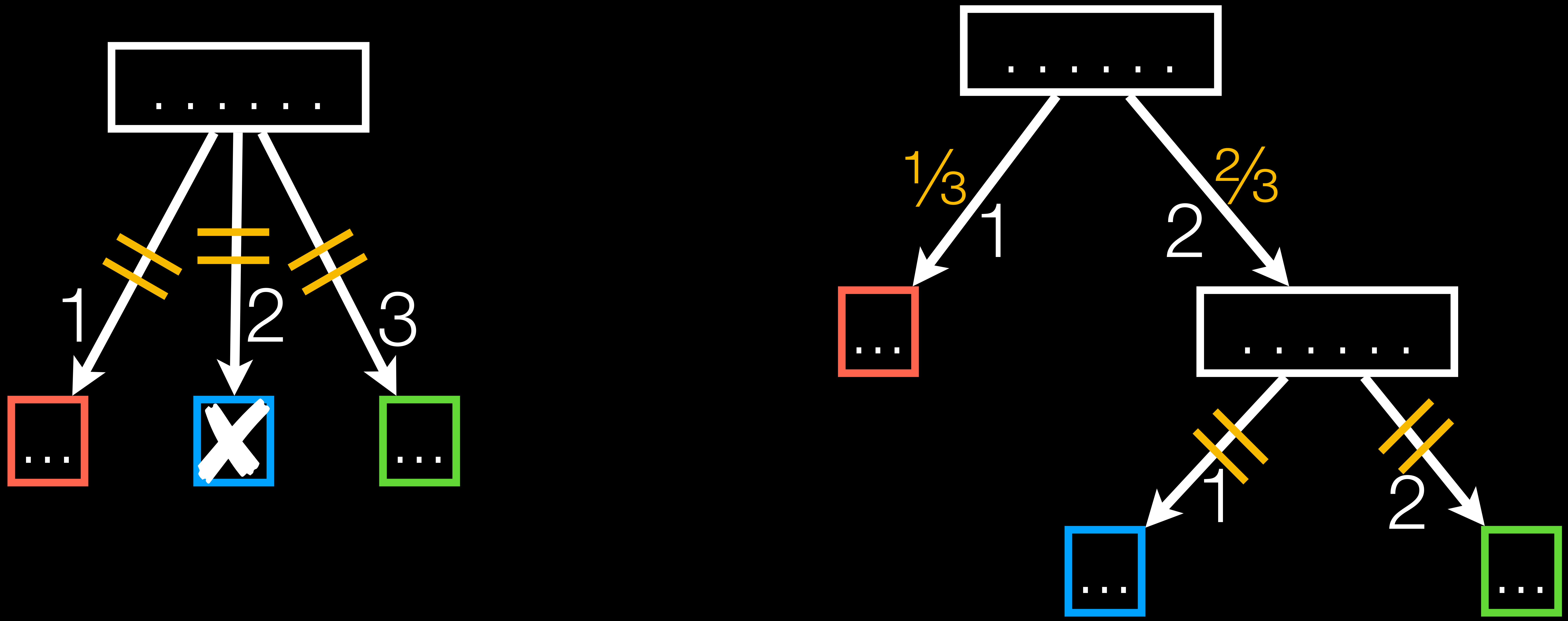
Compiling programs



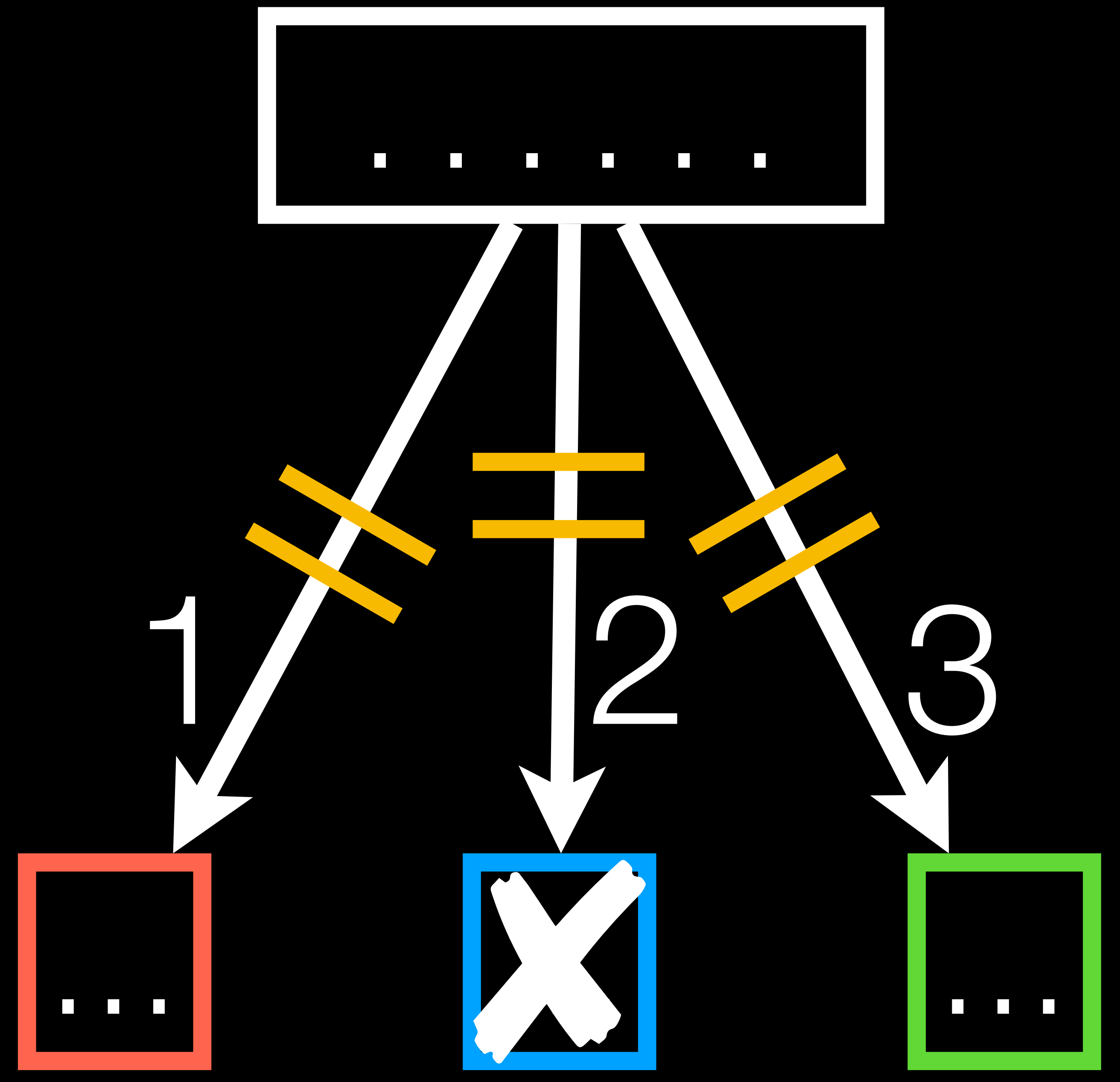
Compiling programs

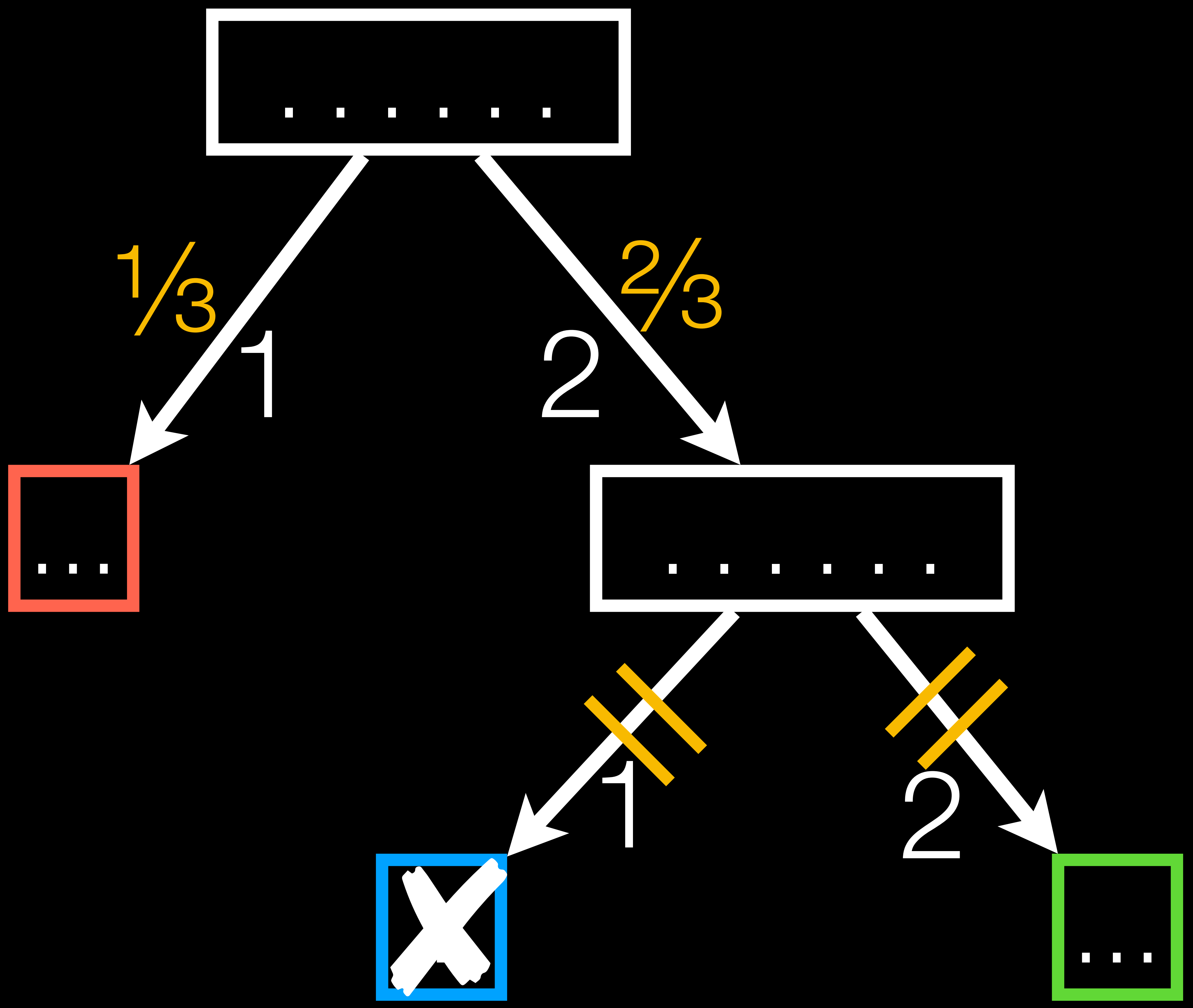


Compiling programs

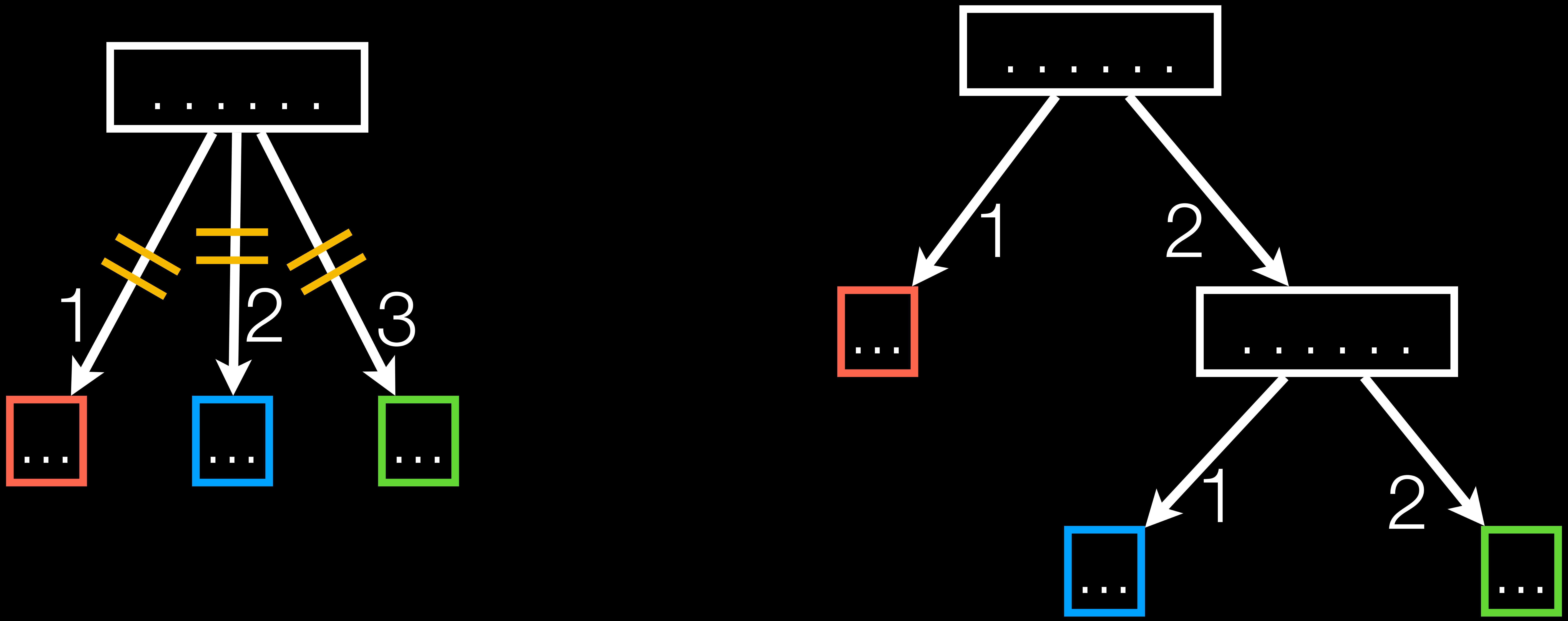


Compiling programs

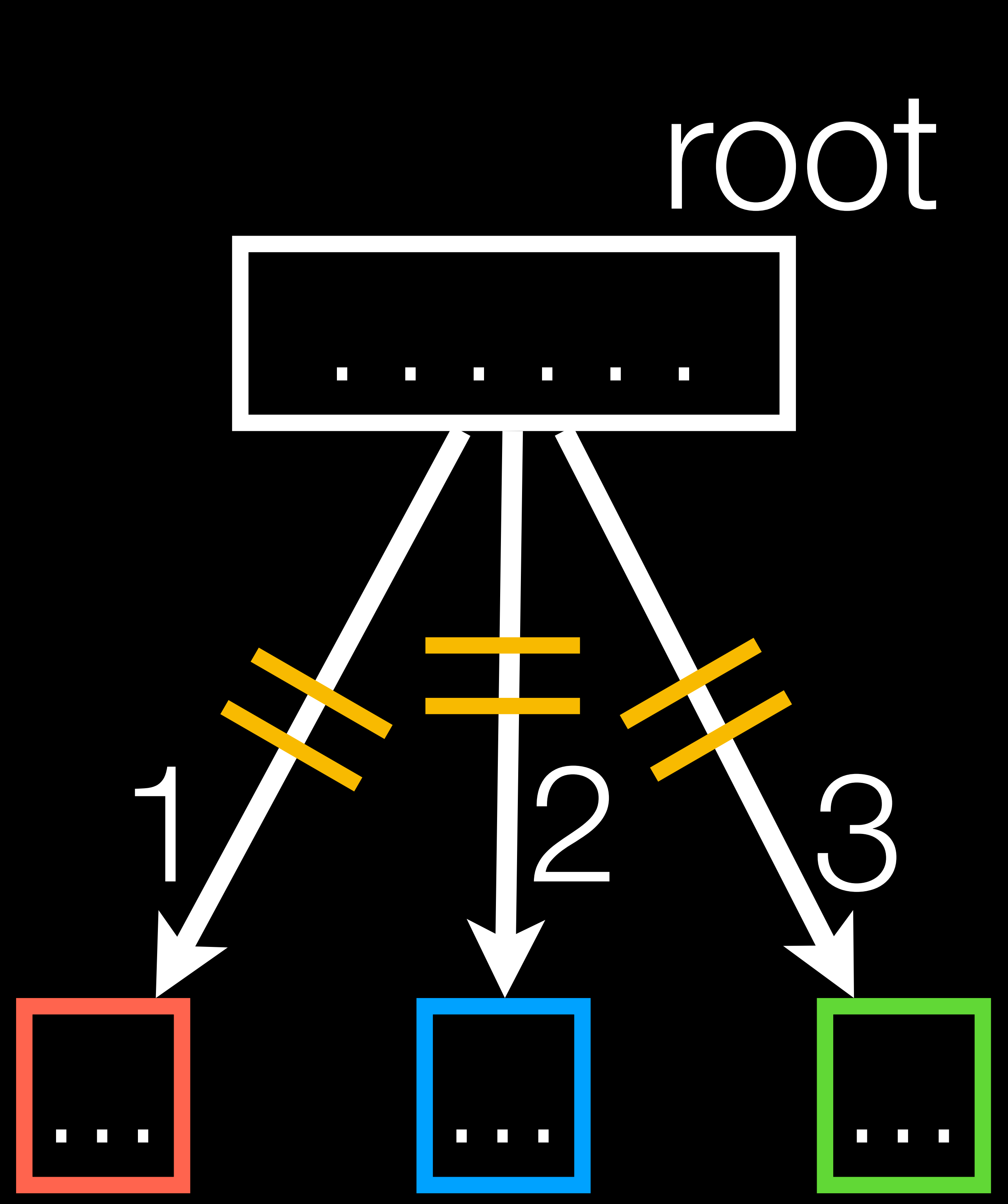


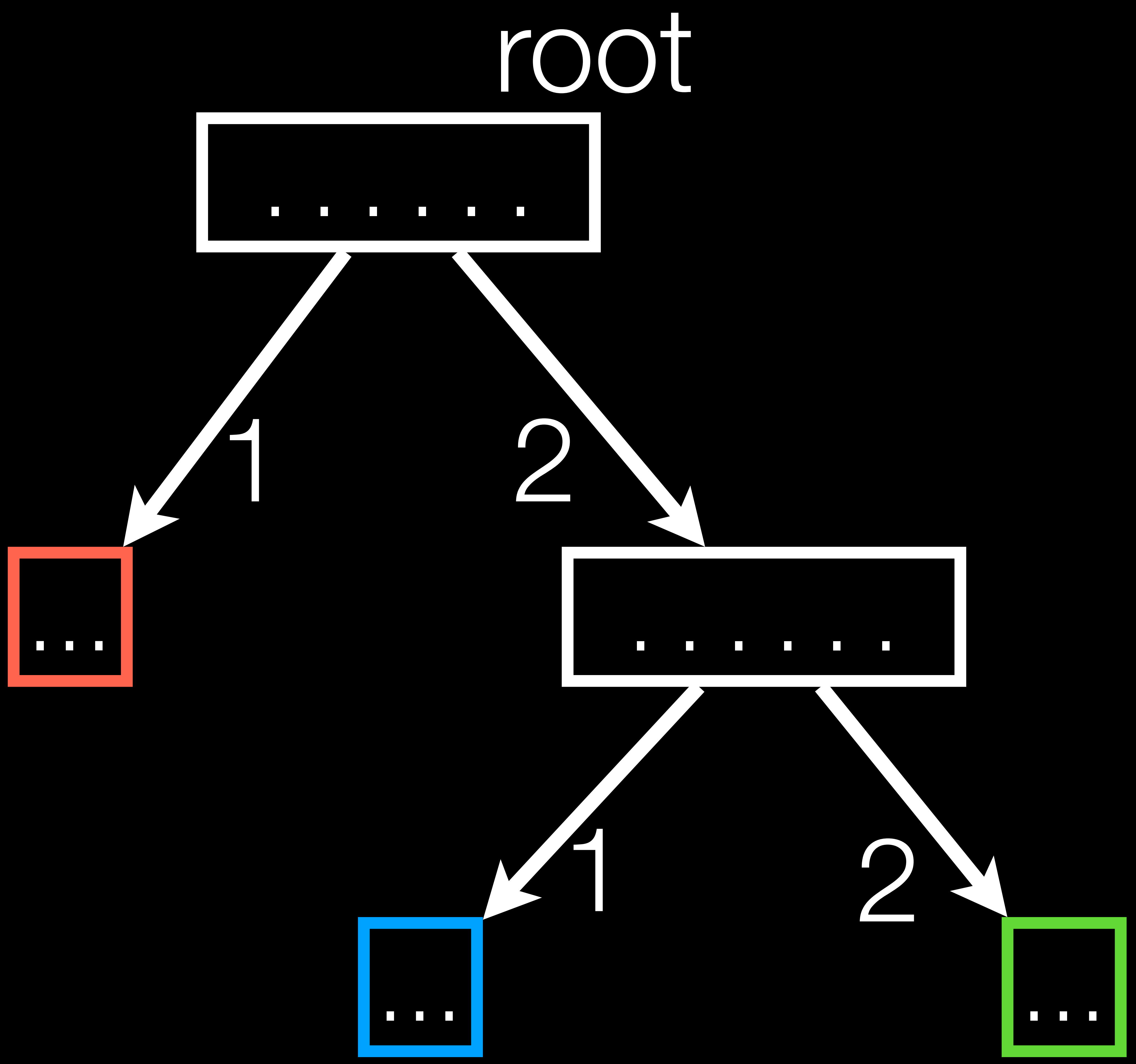


Compiling programs

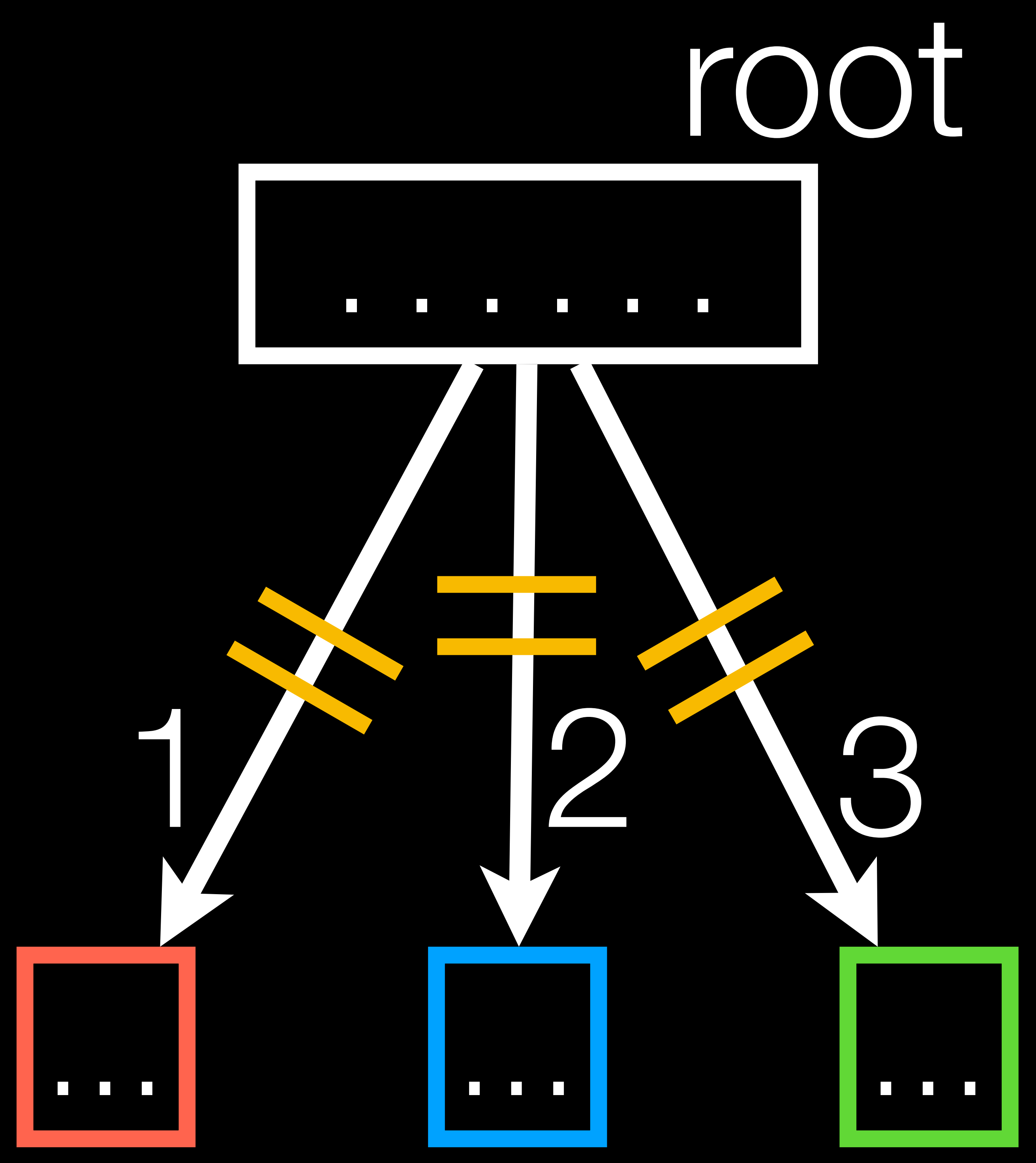


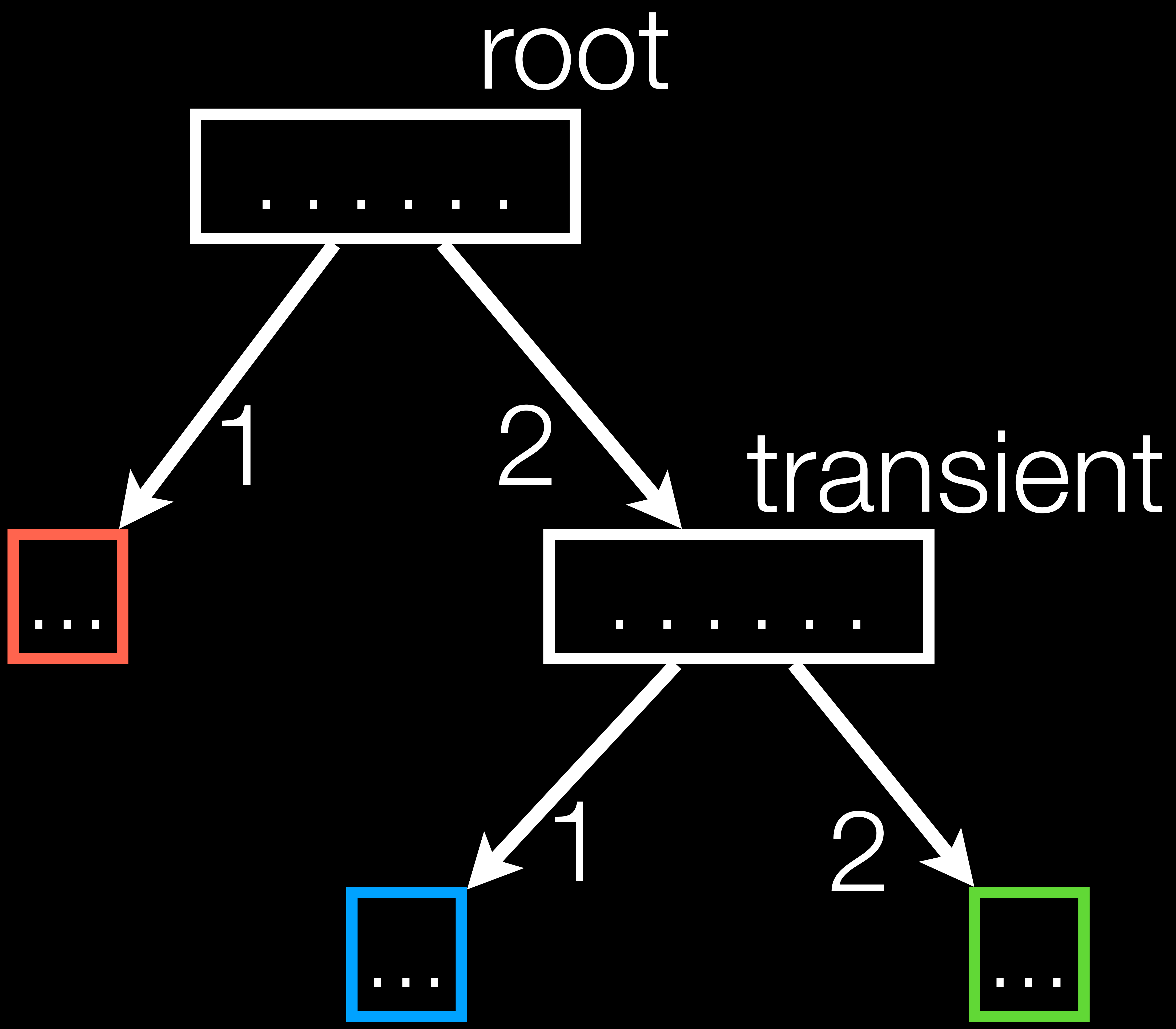
Compiling programs



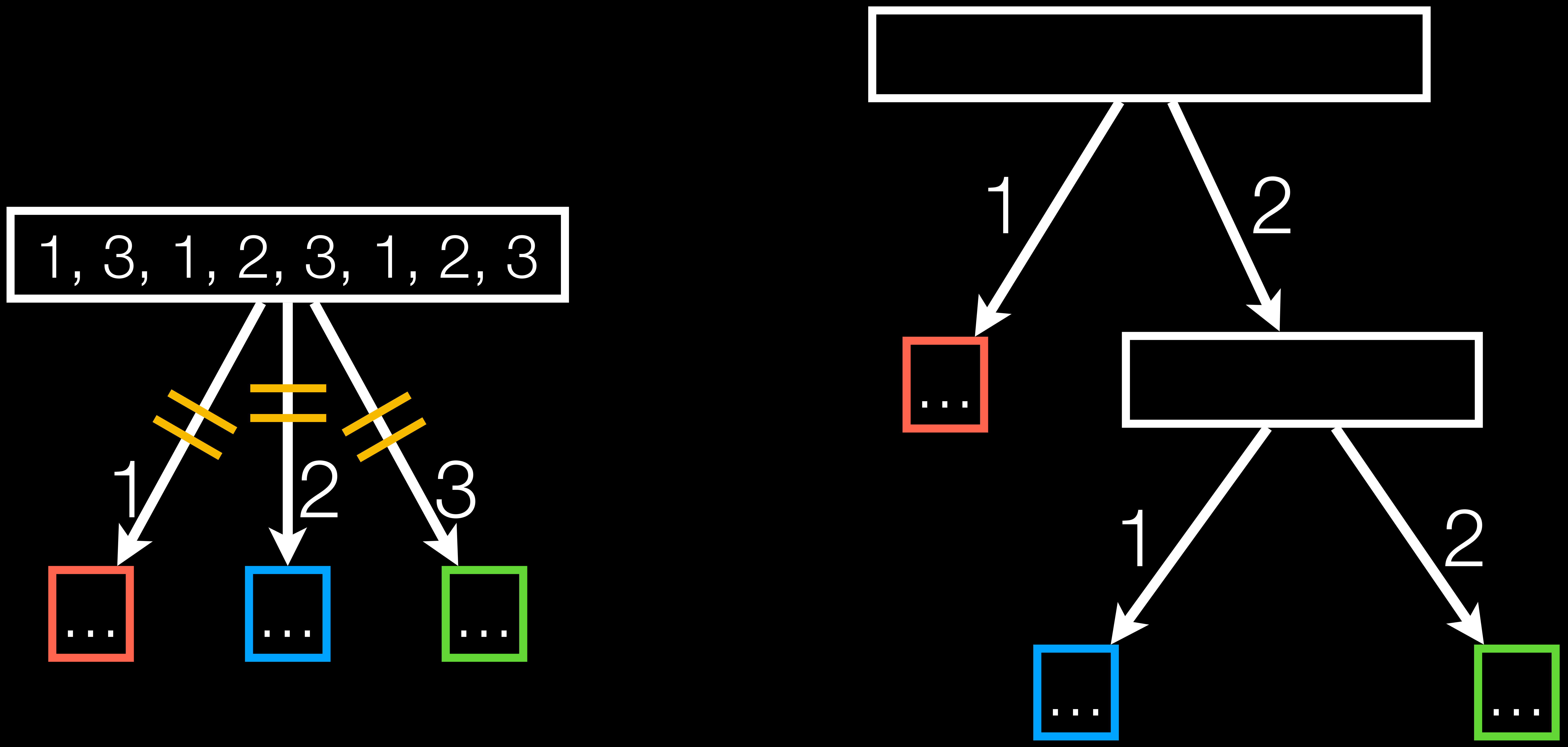


Compiling programs

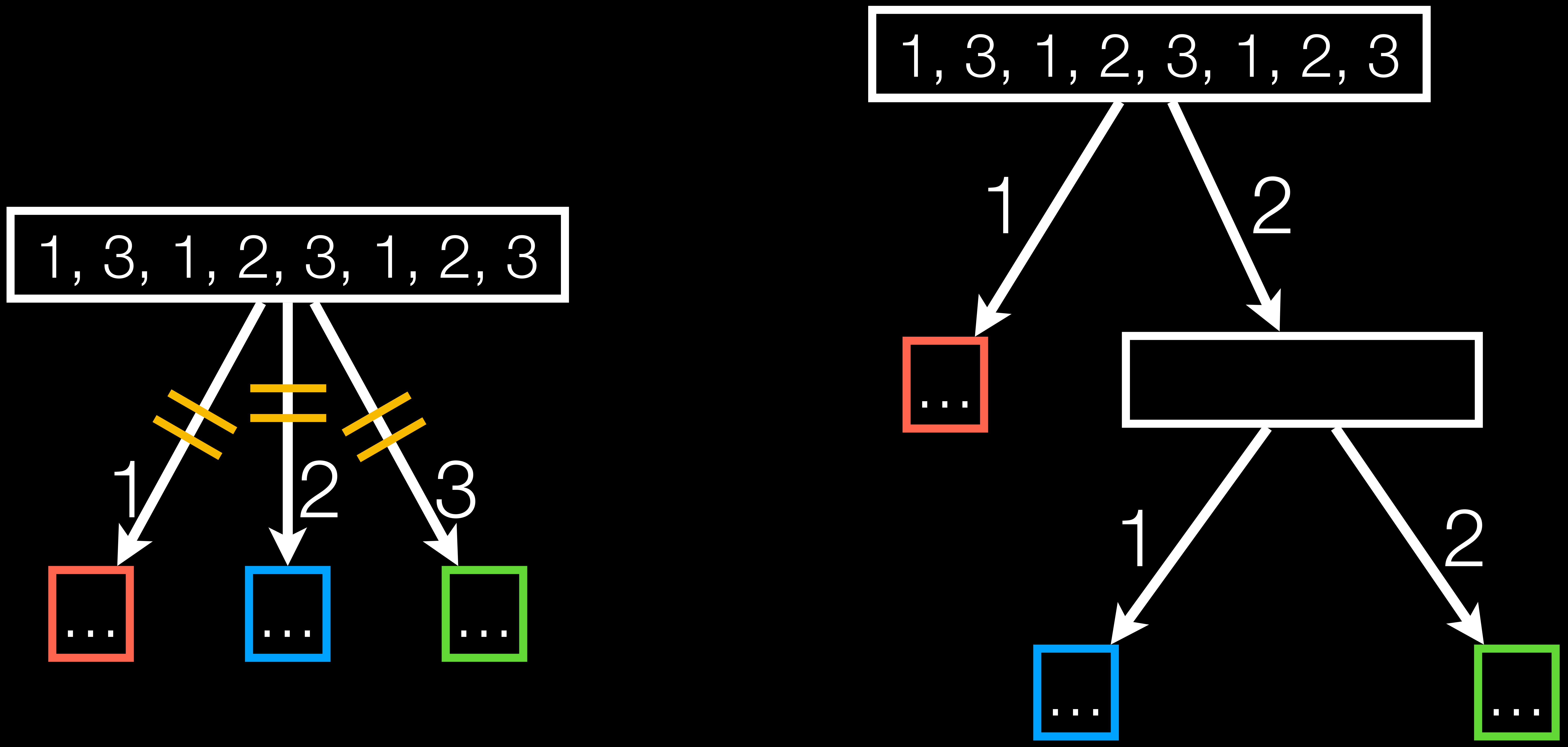




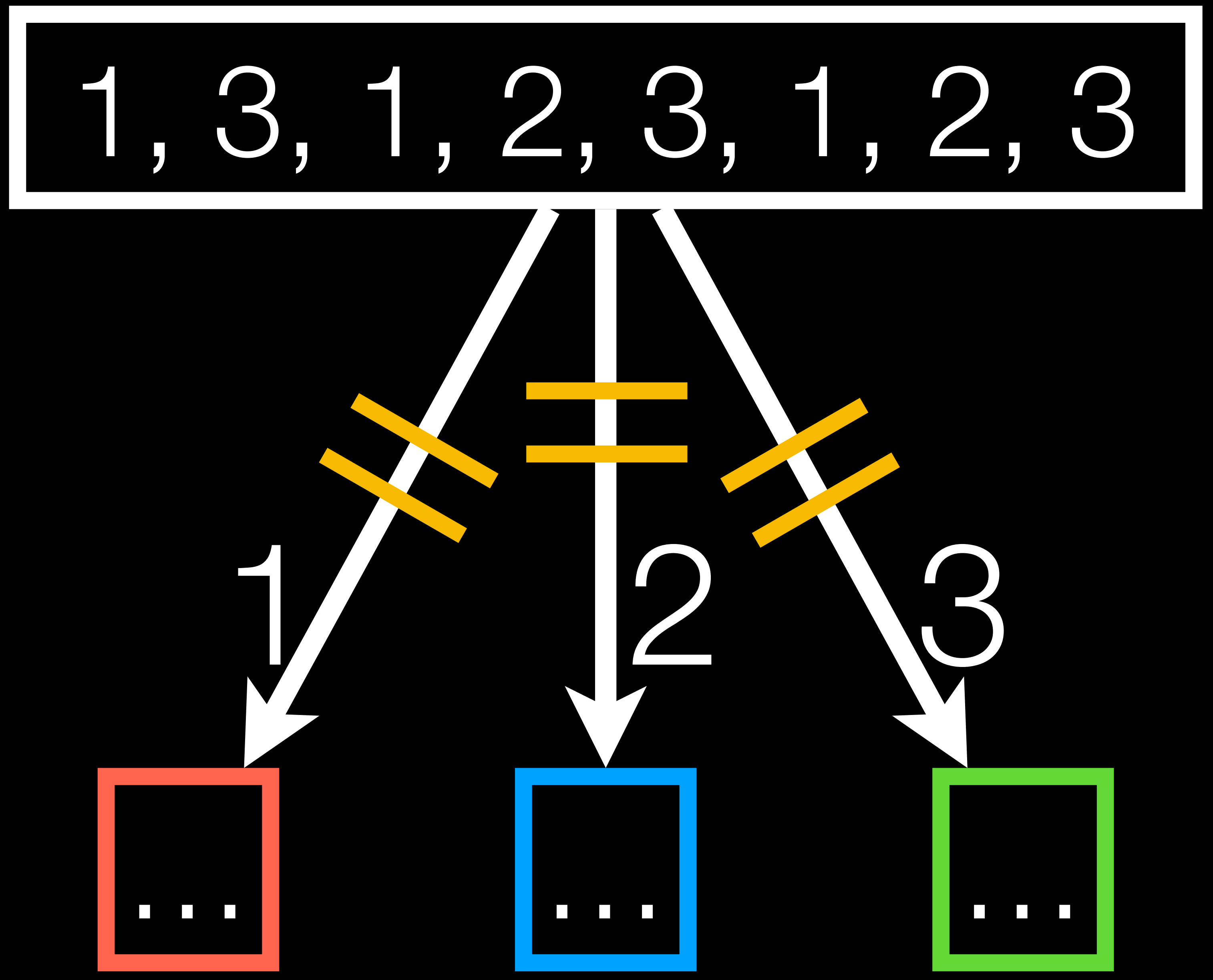
Compiling programs

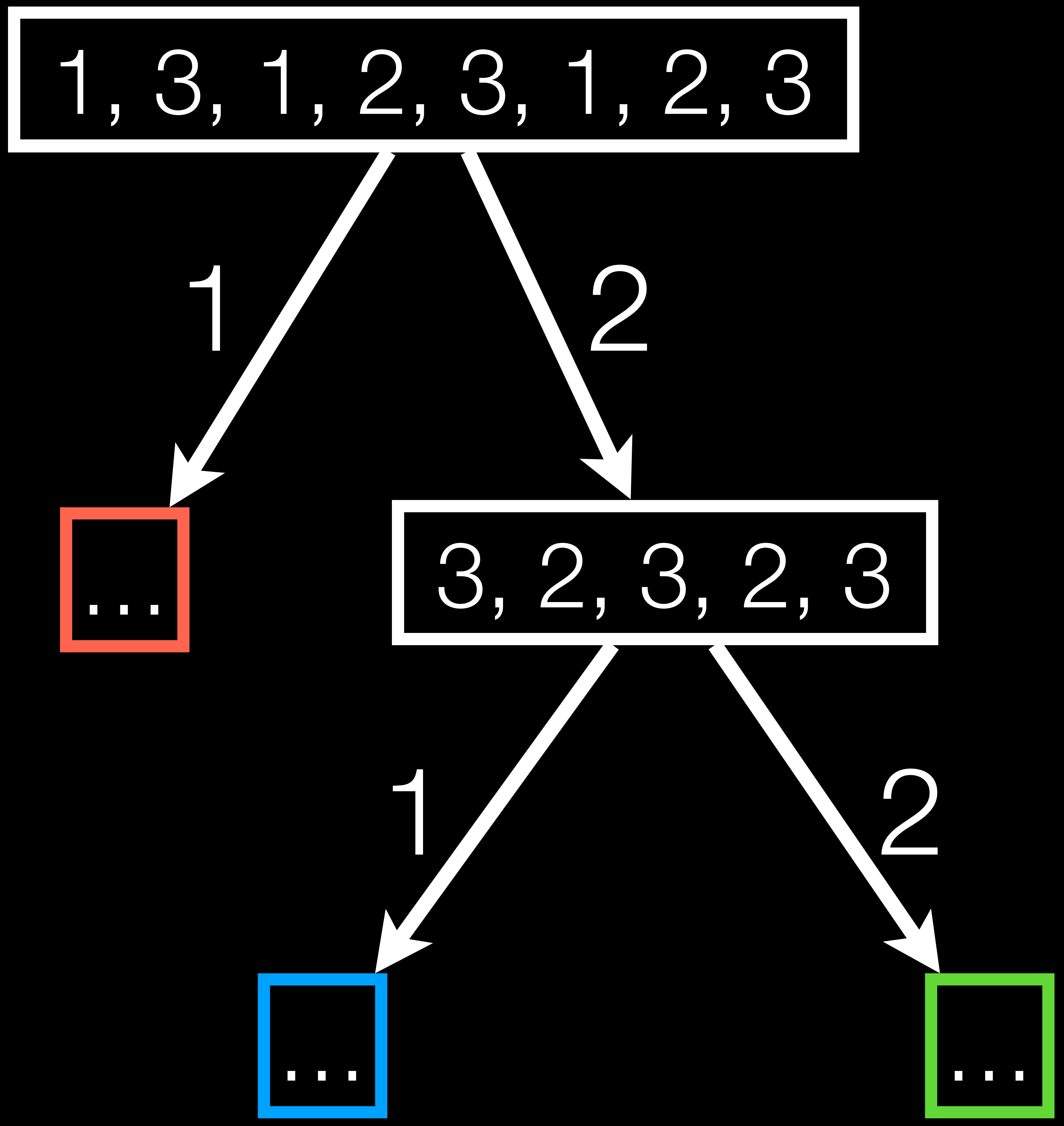


Compiling programs

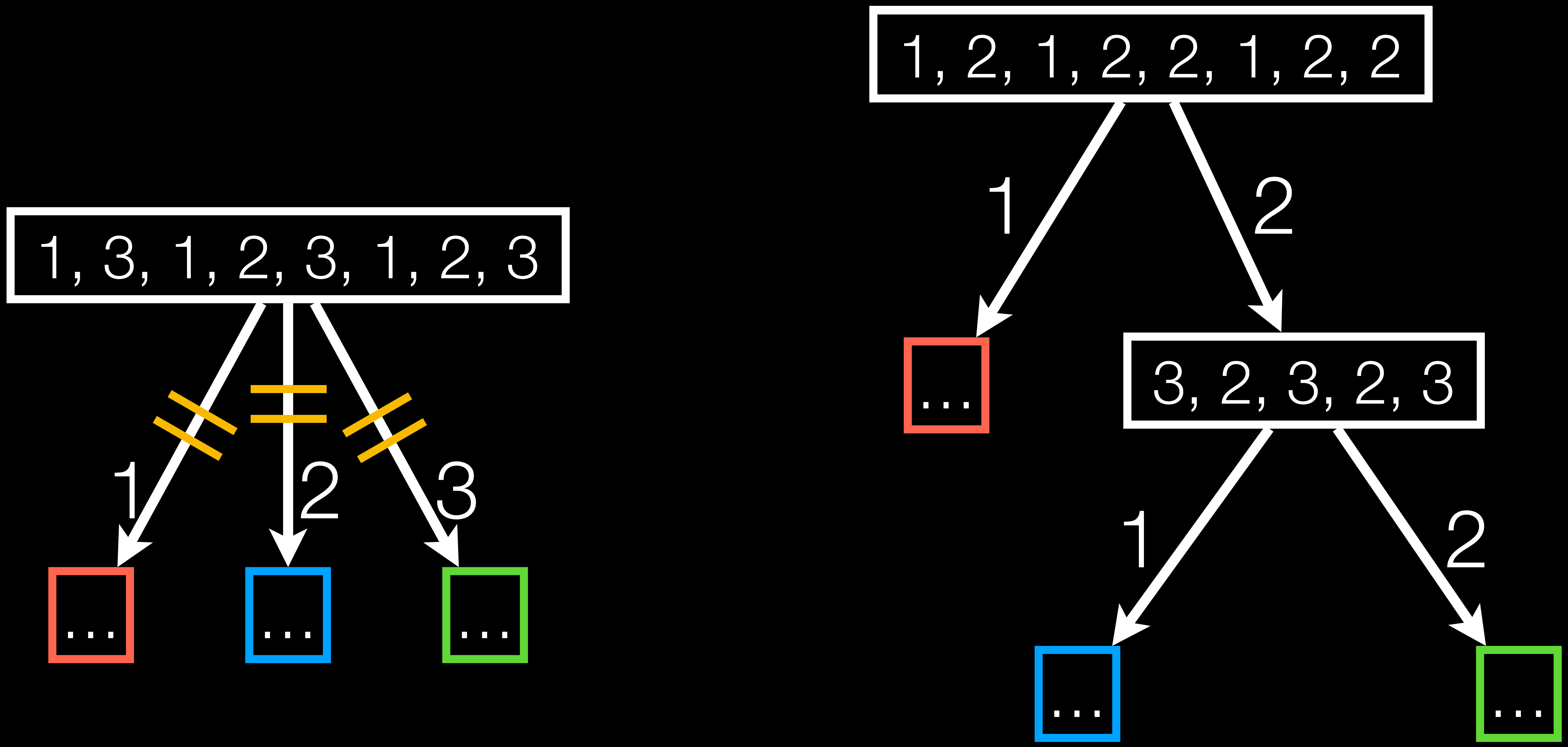


Compiling programs

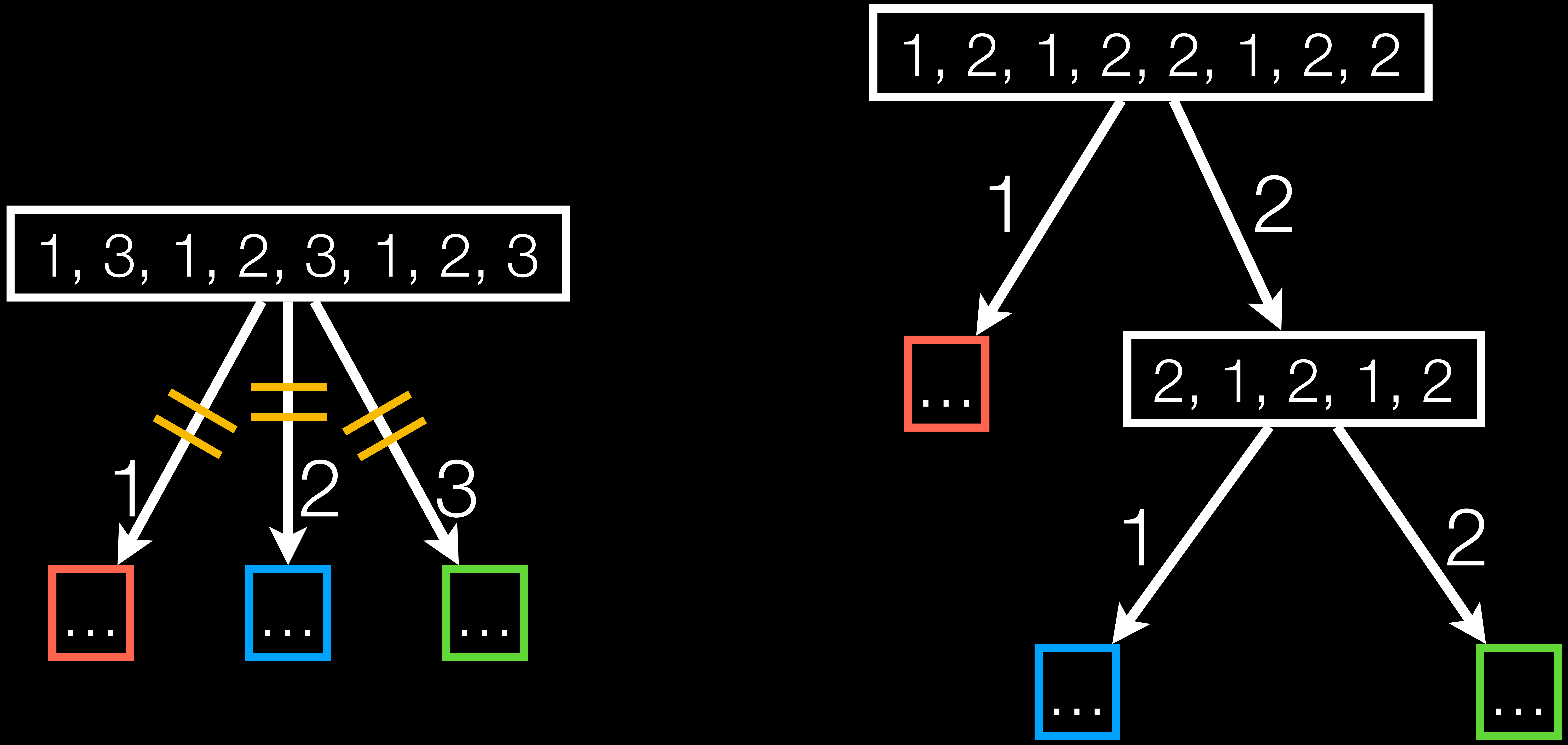




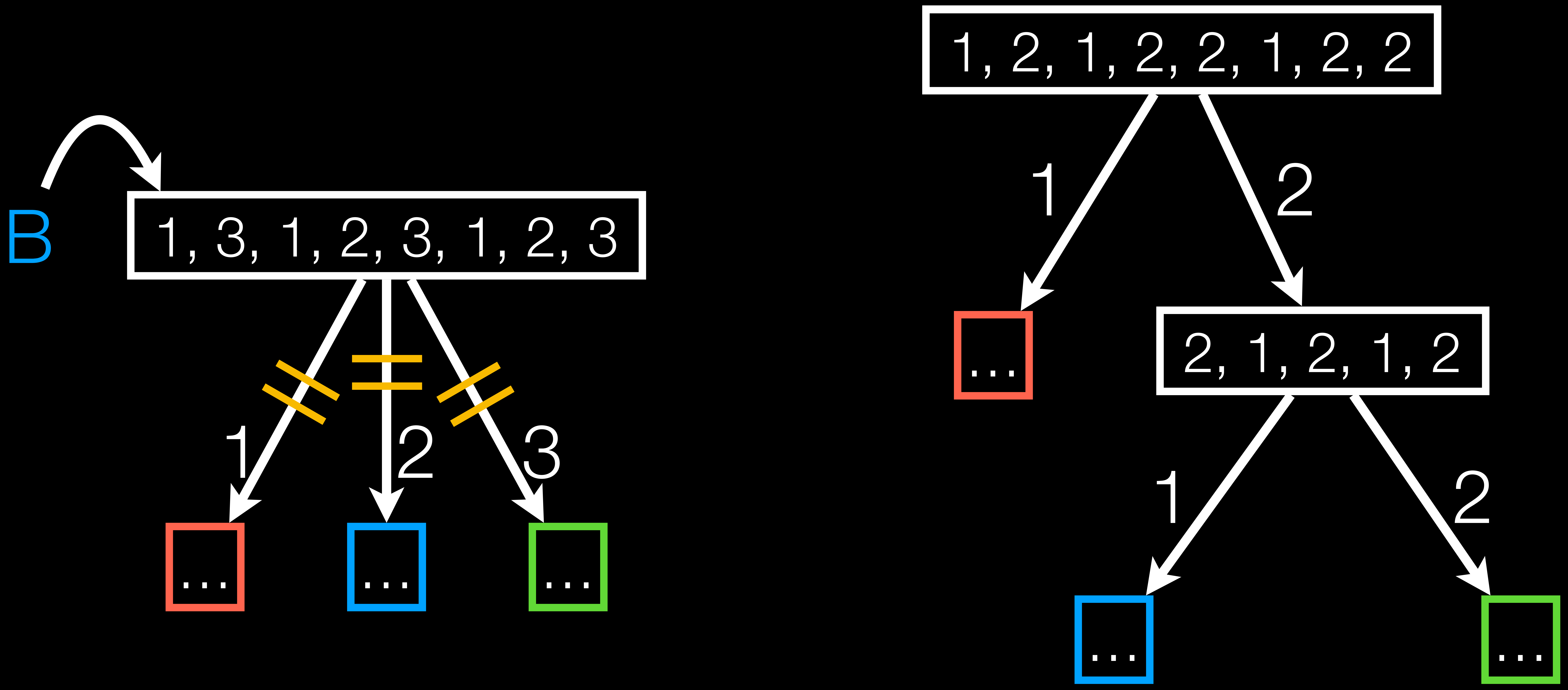
Compiling programs



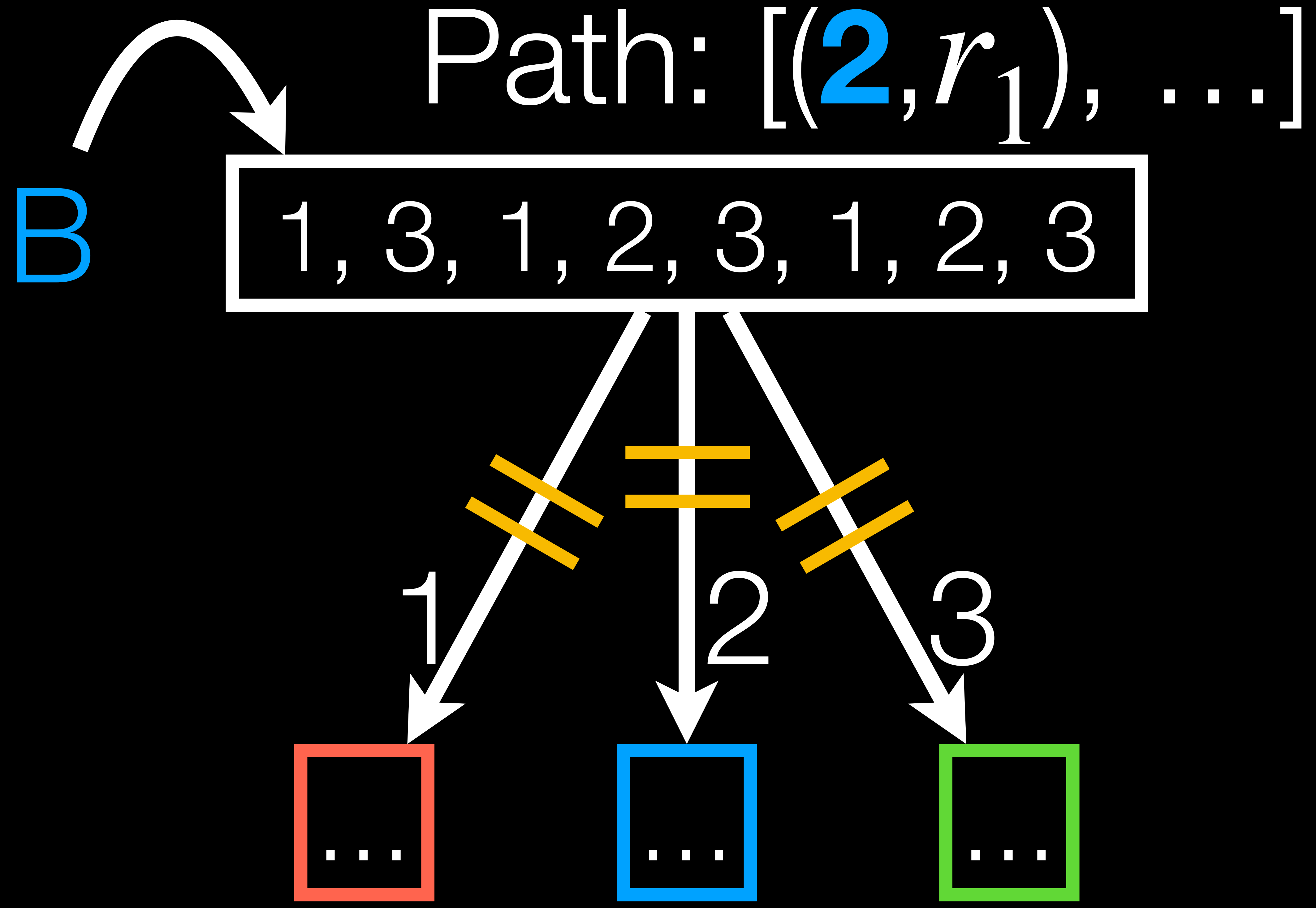
Compiling programs

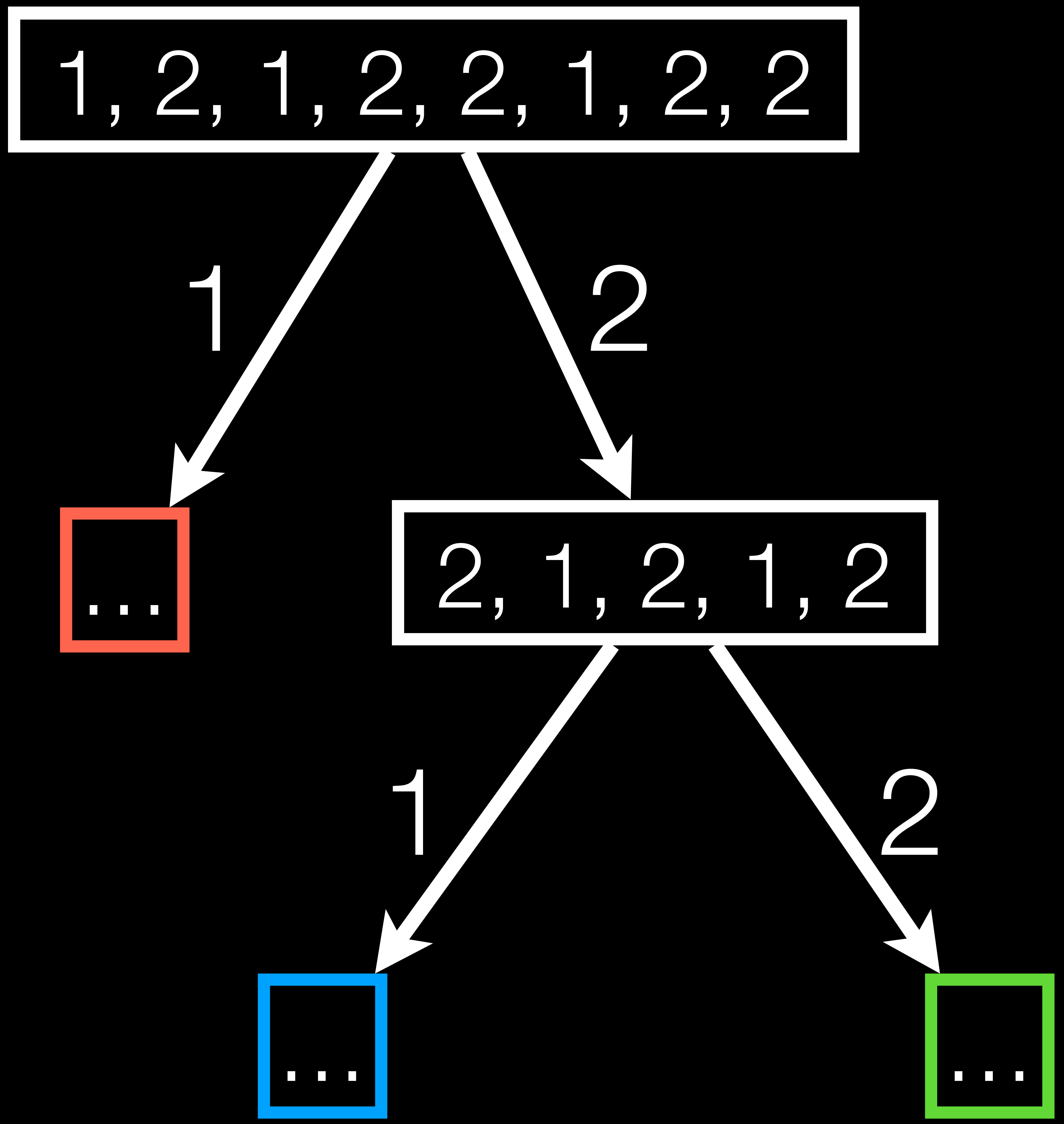


Compiling programs

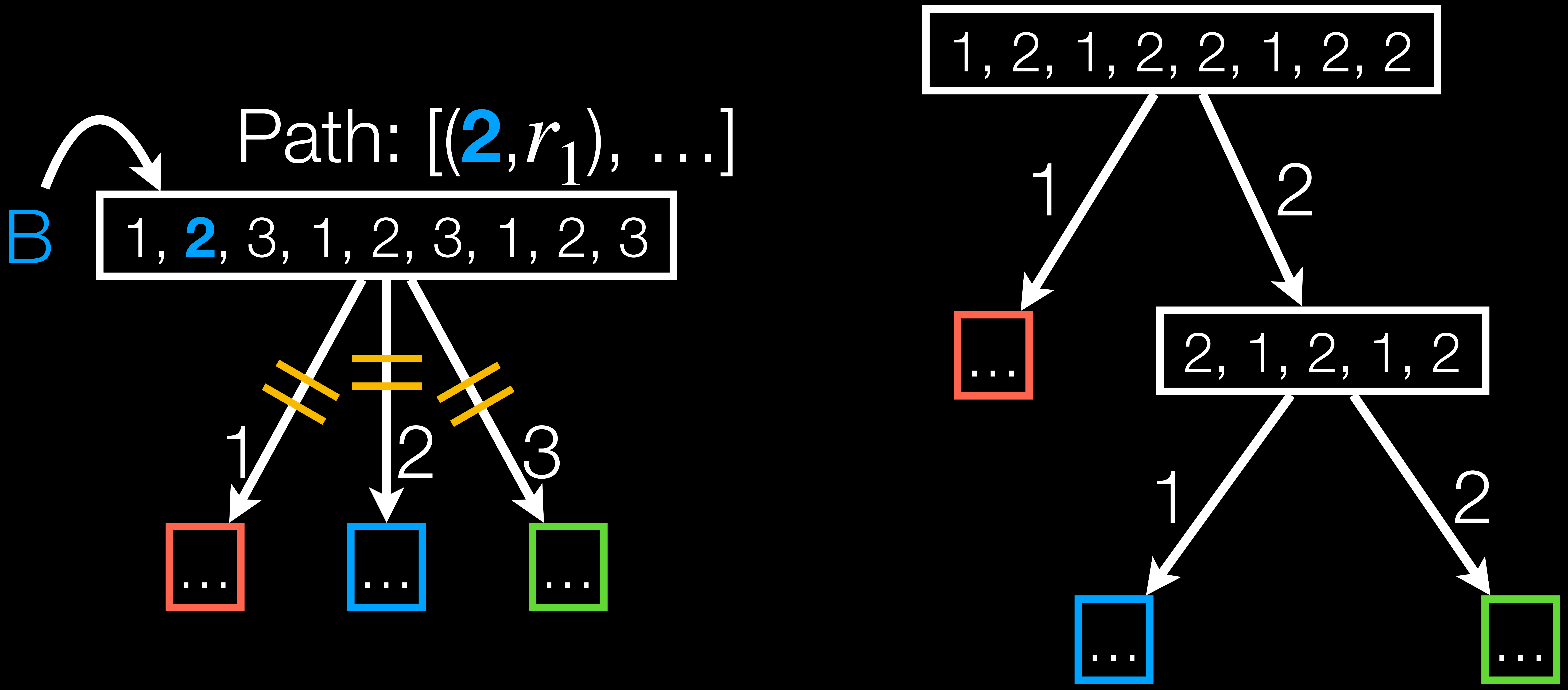


Compiling programs

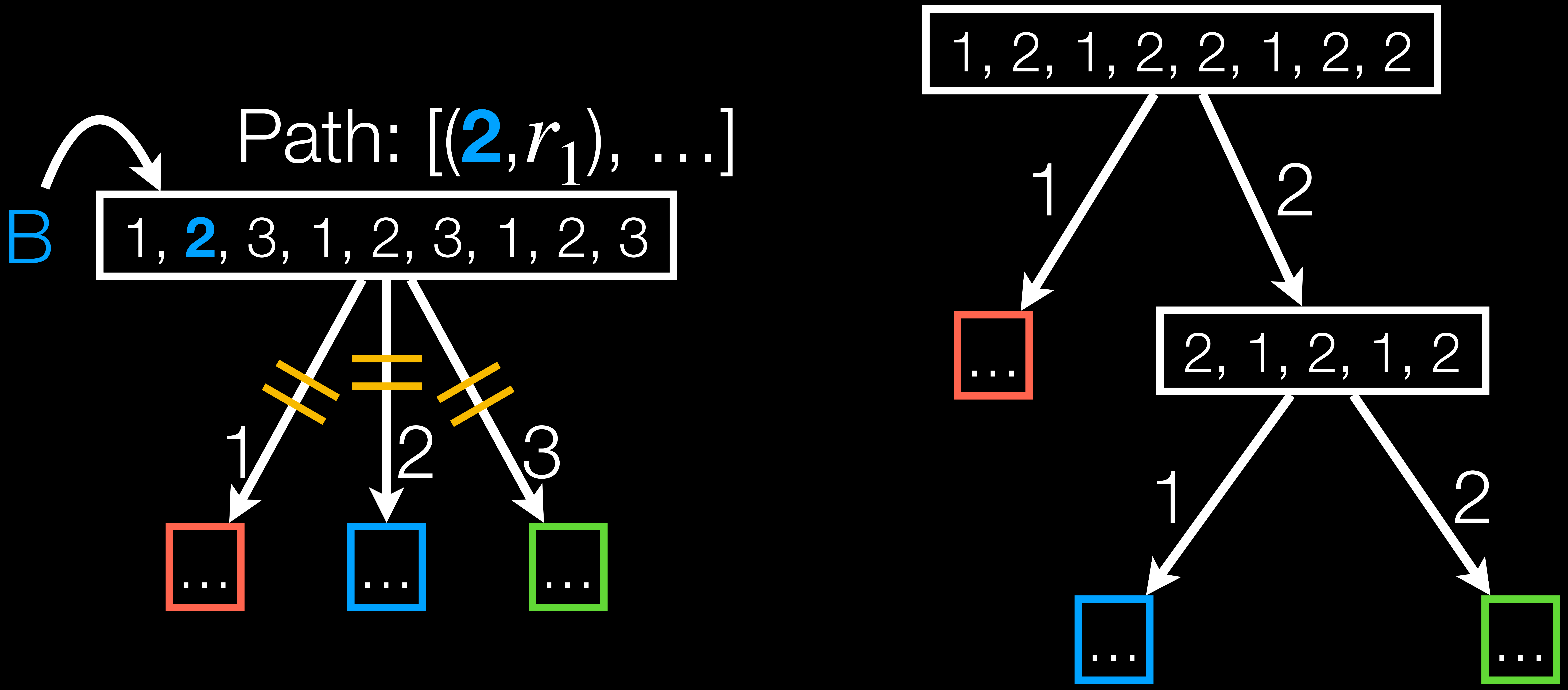




Compiling programs

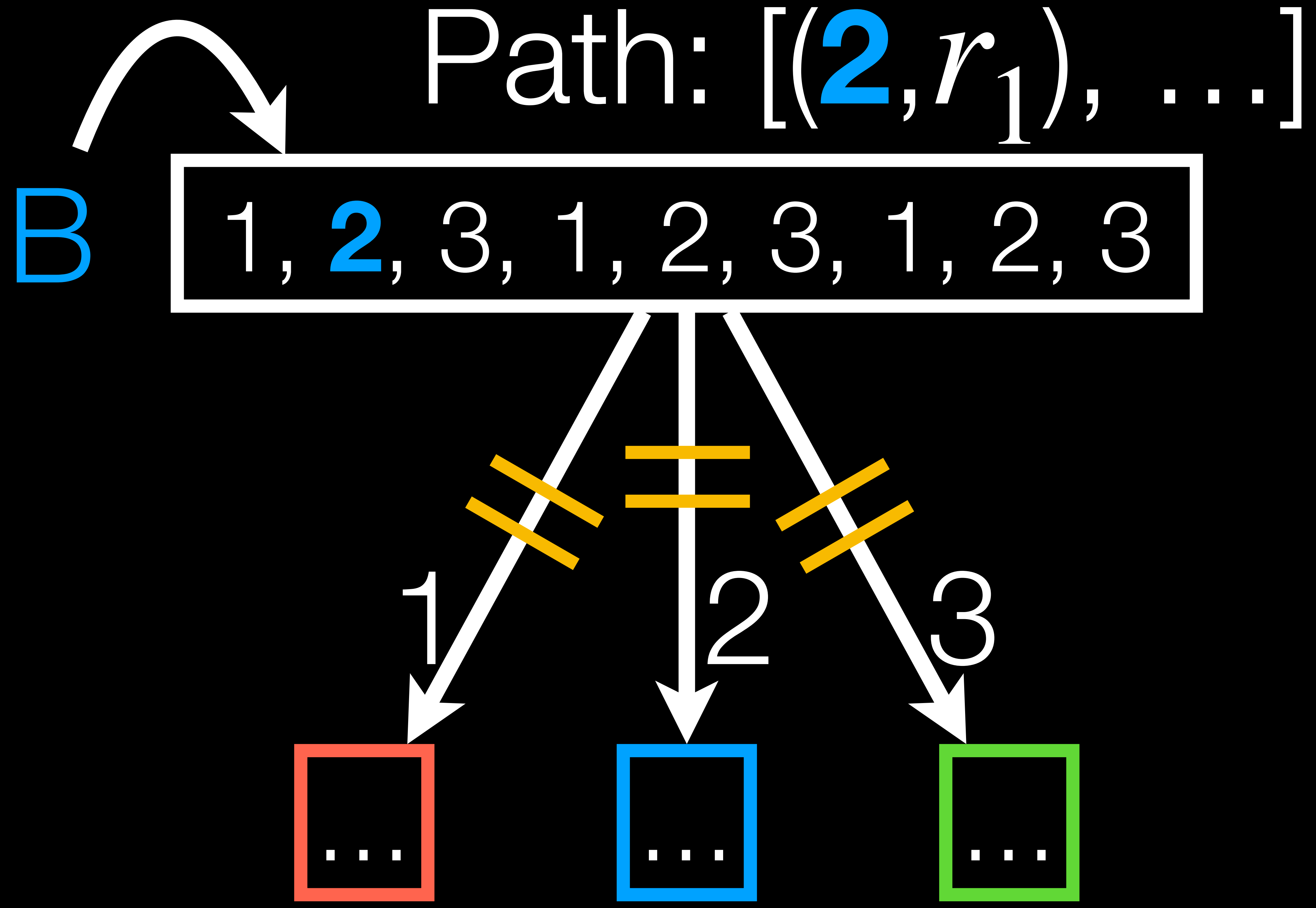


Compiling programs

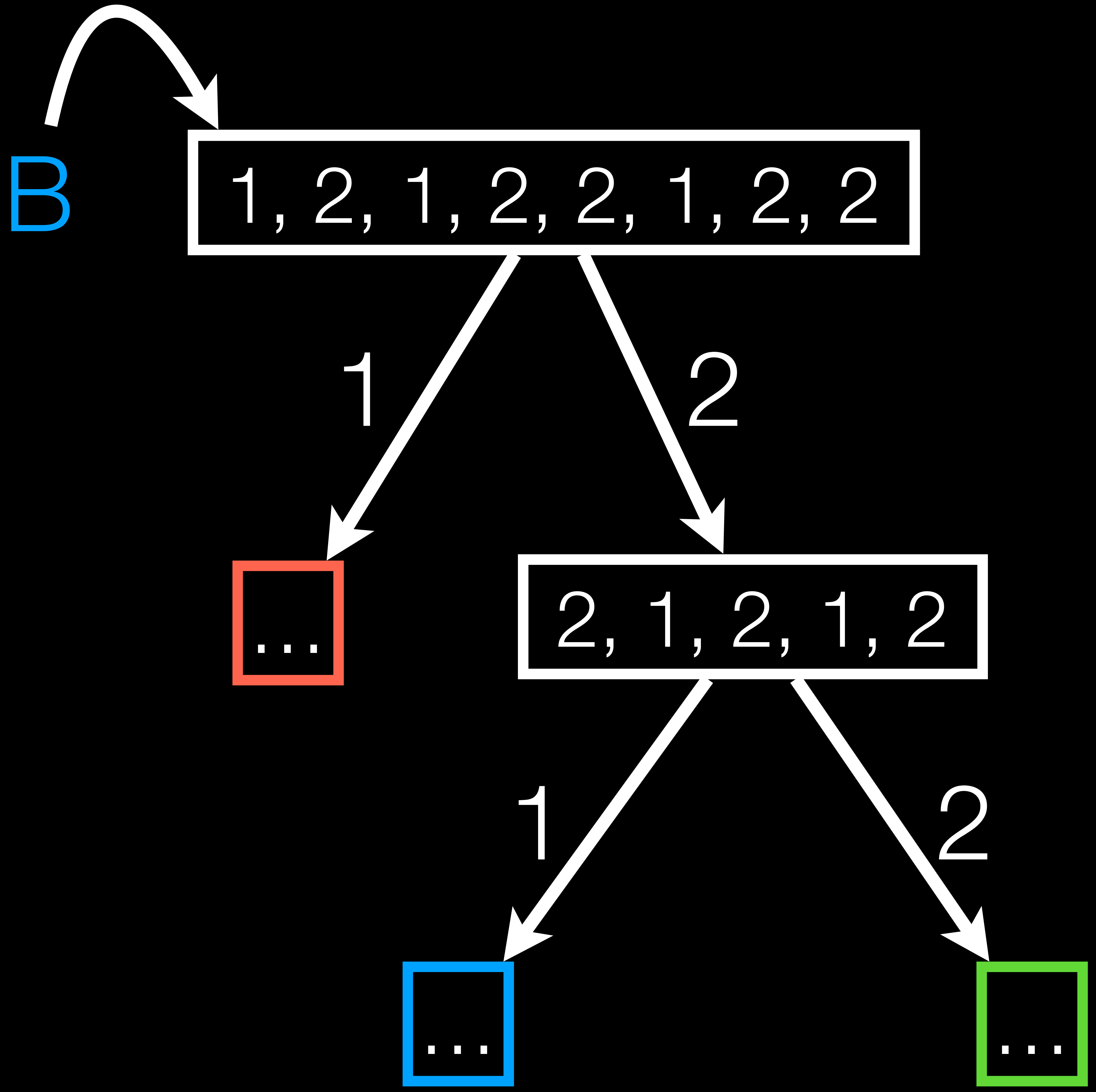


Path: [($\mathbf{2}$, r_1), …]

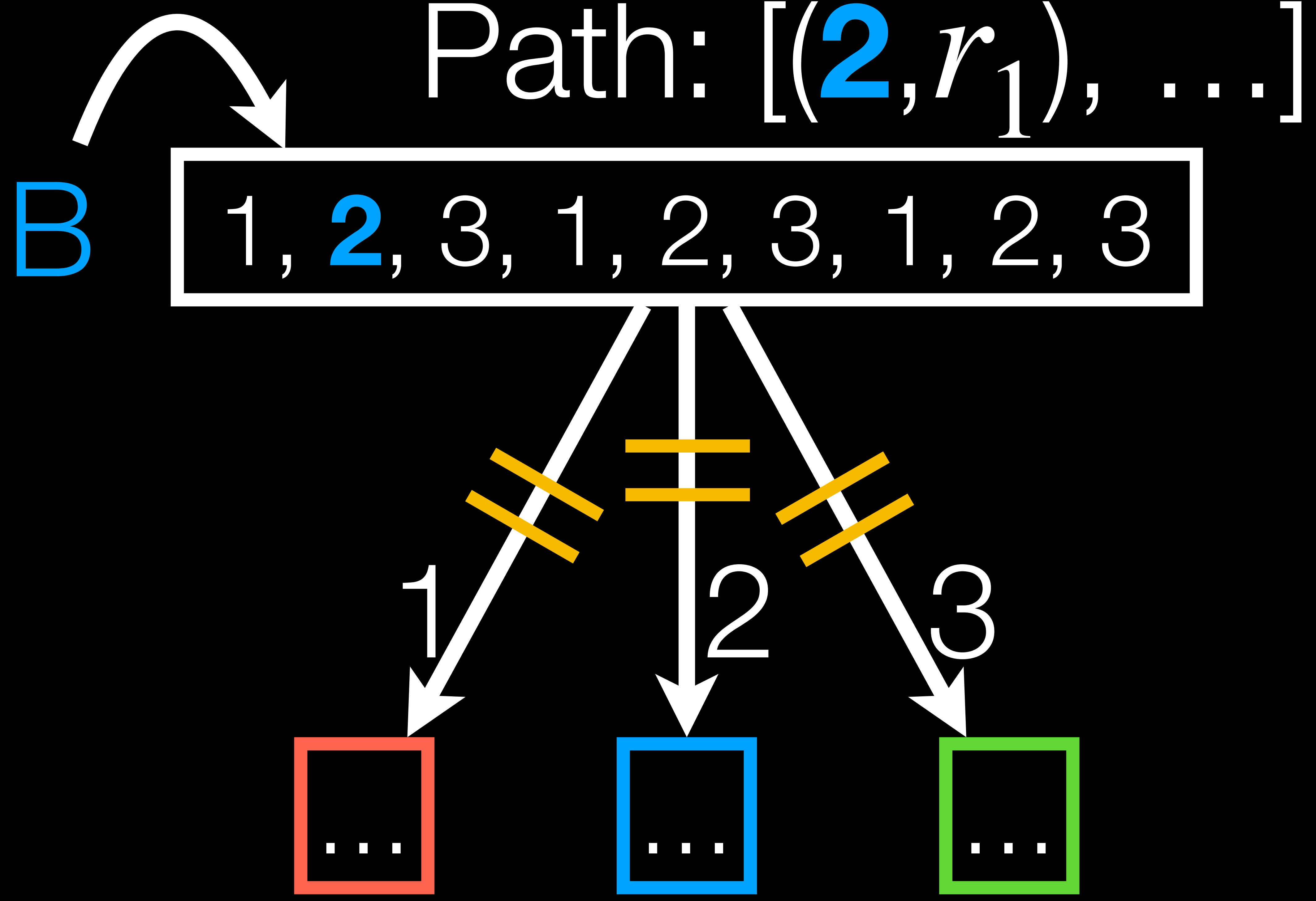
Compiling programs



Path: [($\mathbf{2}$,r_1), …]



Compiling programs



Path: [(**2**,r_1), …]

B

1, **2**, 3, 1, 2, 3, 1, 2, 3

1

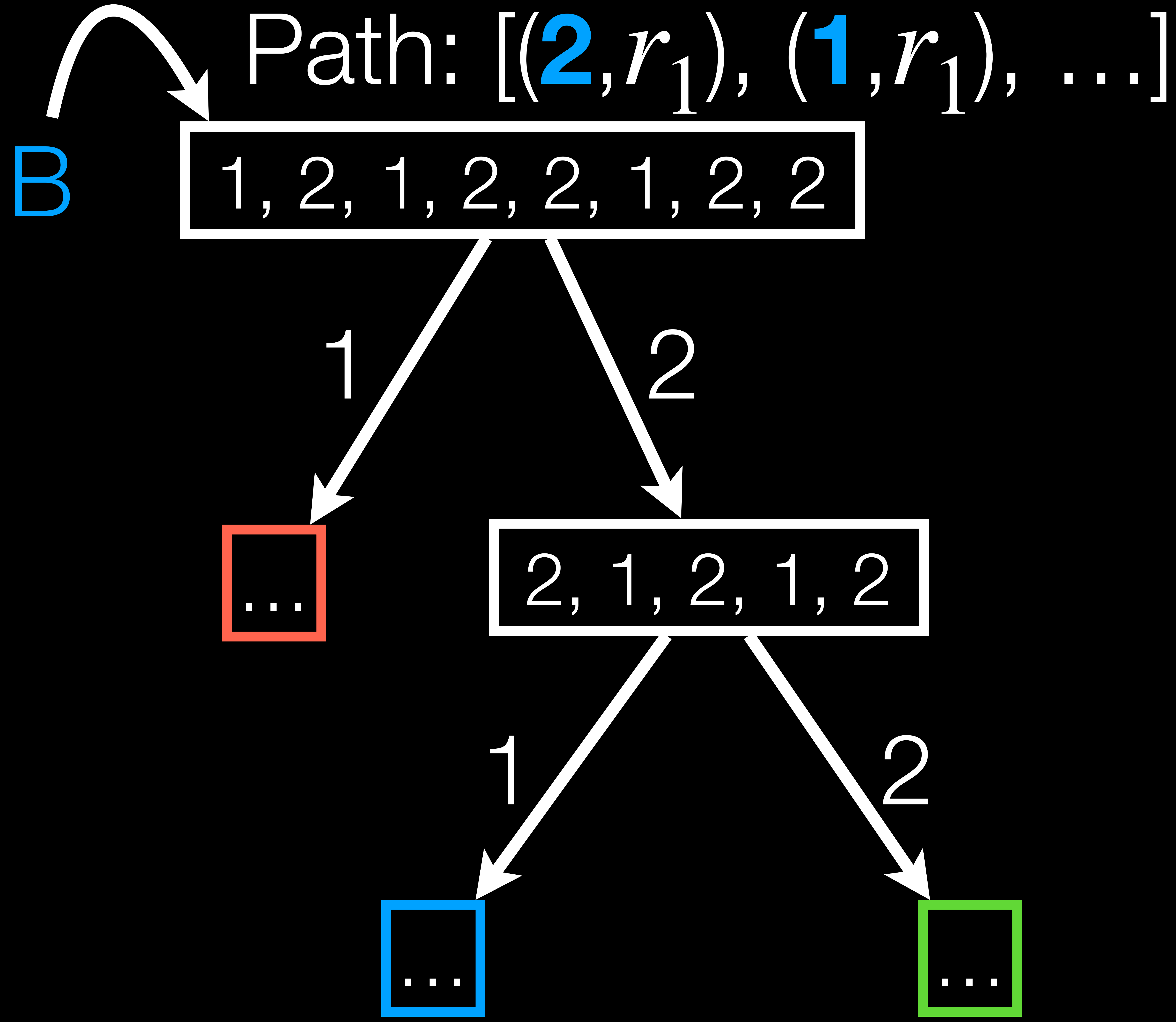
2

3

…

…

…



Path: [(**2**,r_1), (**1**,r_1), …]

B

1, 2, 1, 2, 2, 1, 2, 2

1

2

…

2, 1, 2, 1, 2

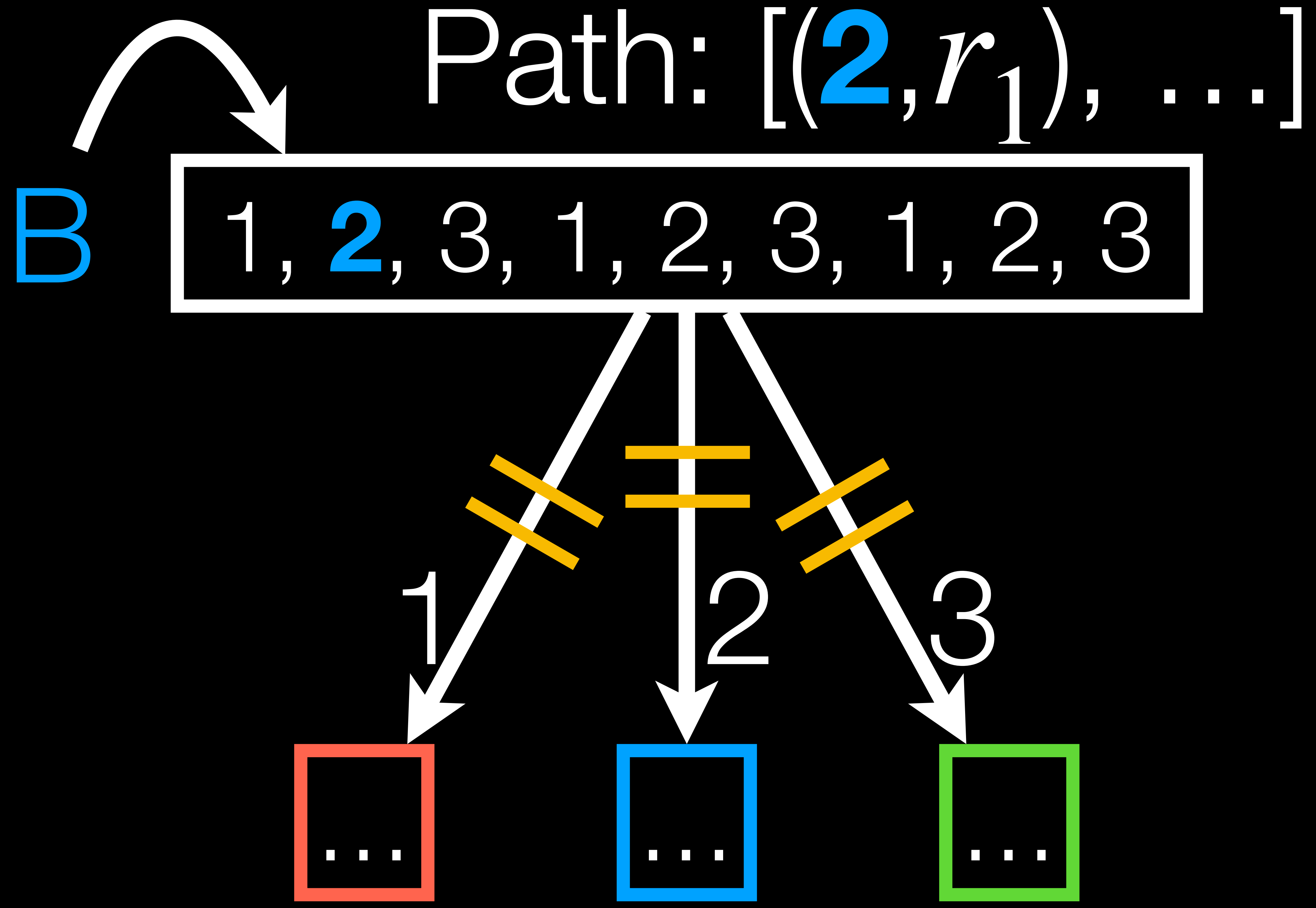
1

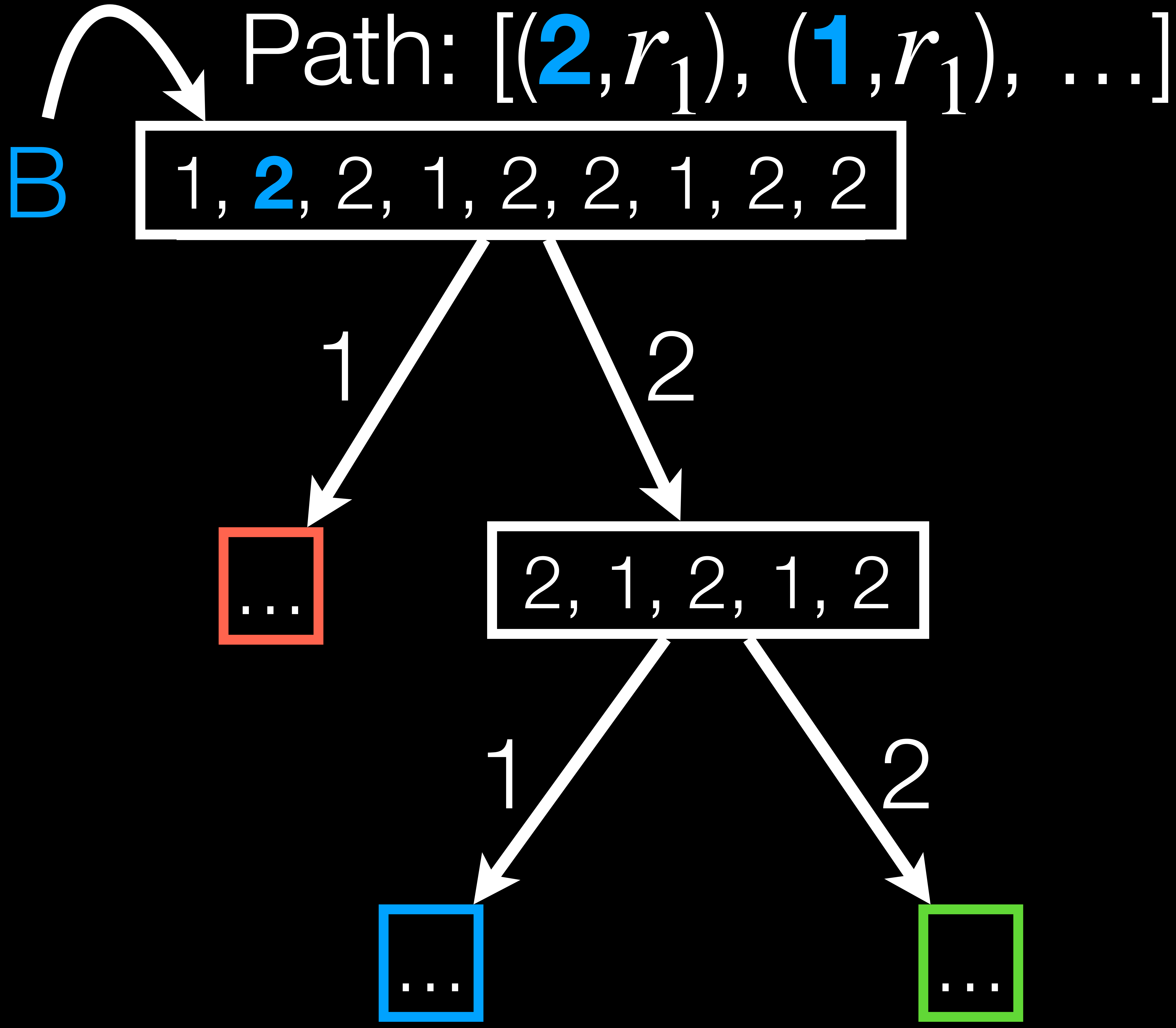
2

…

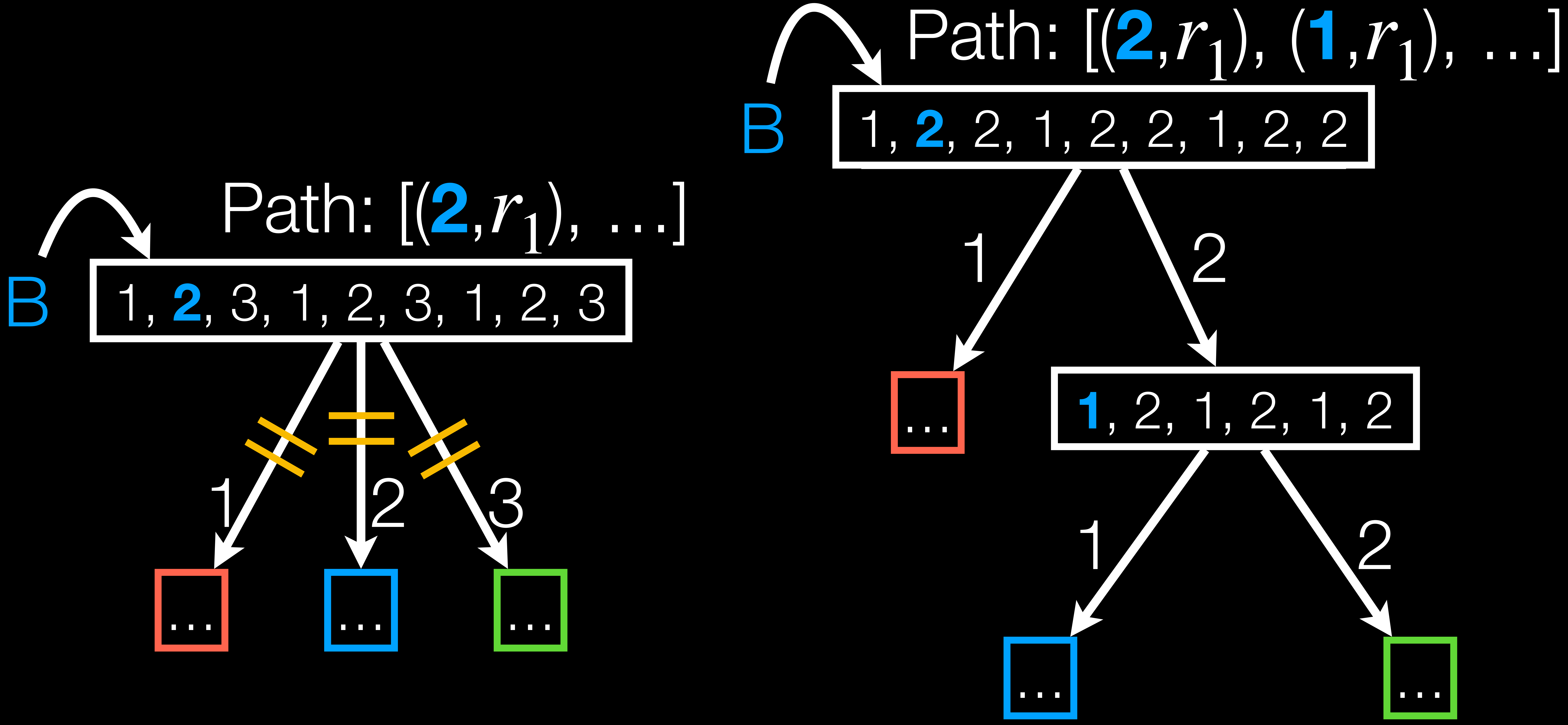
…

Compiling programs

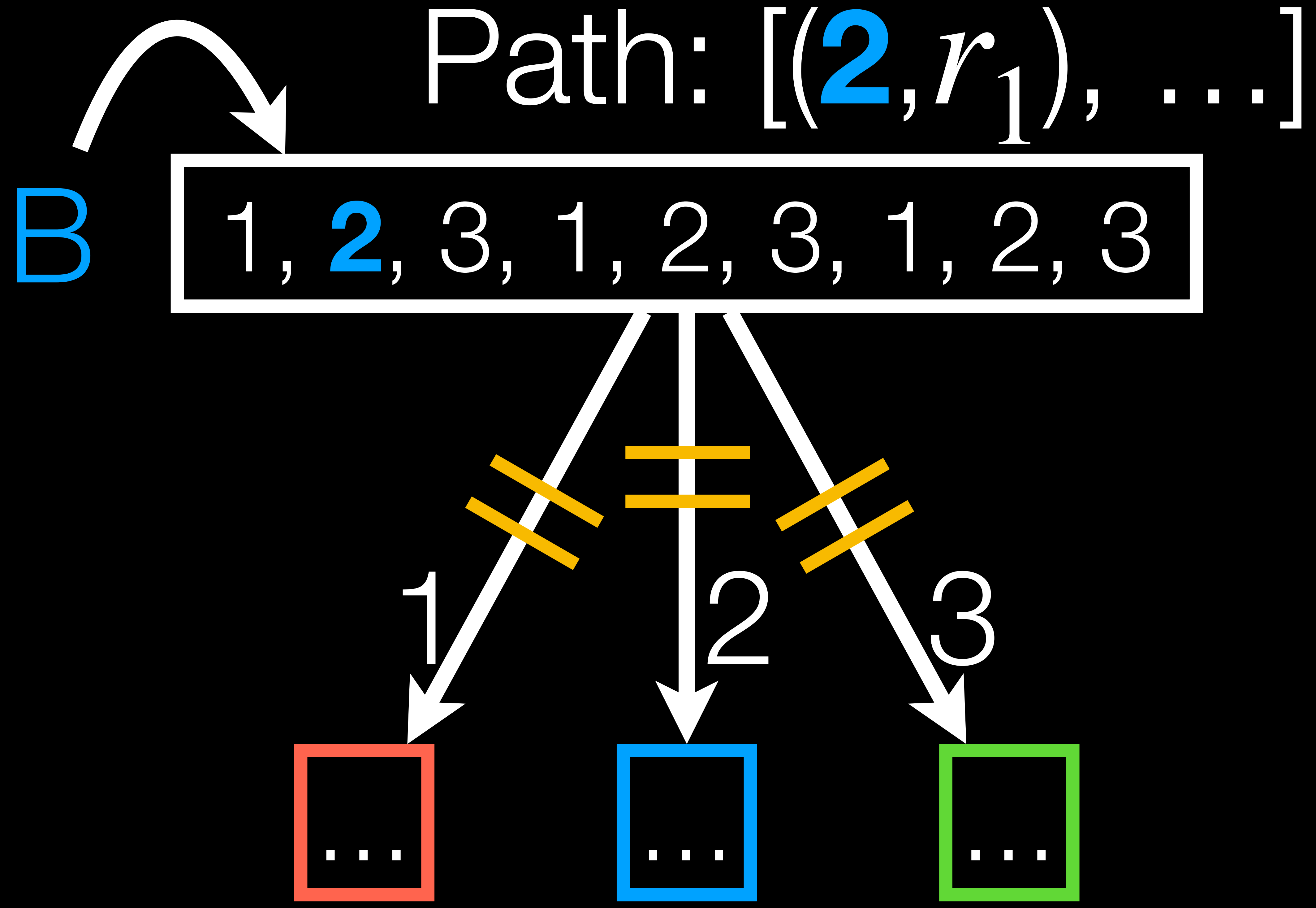


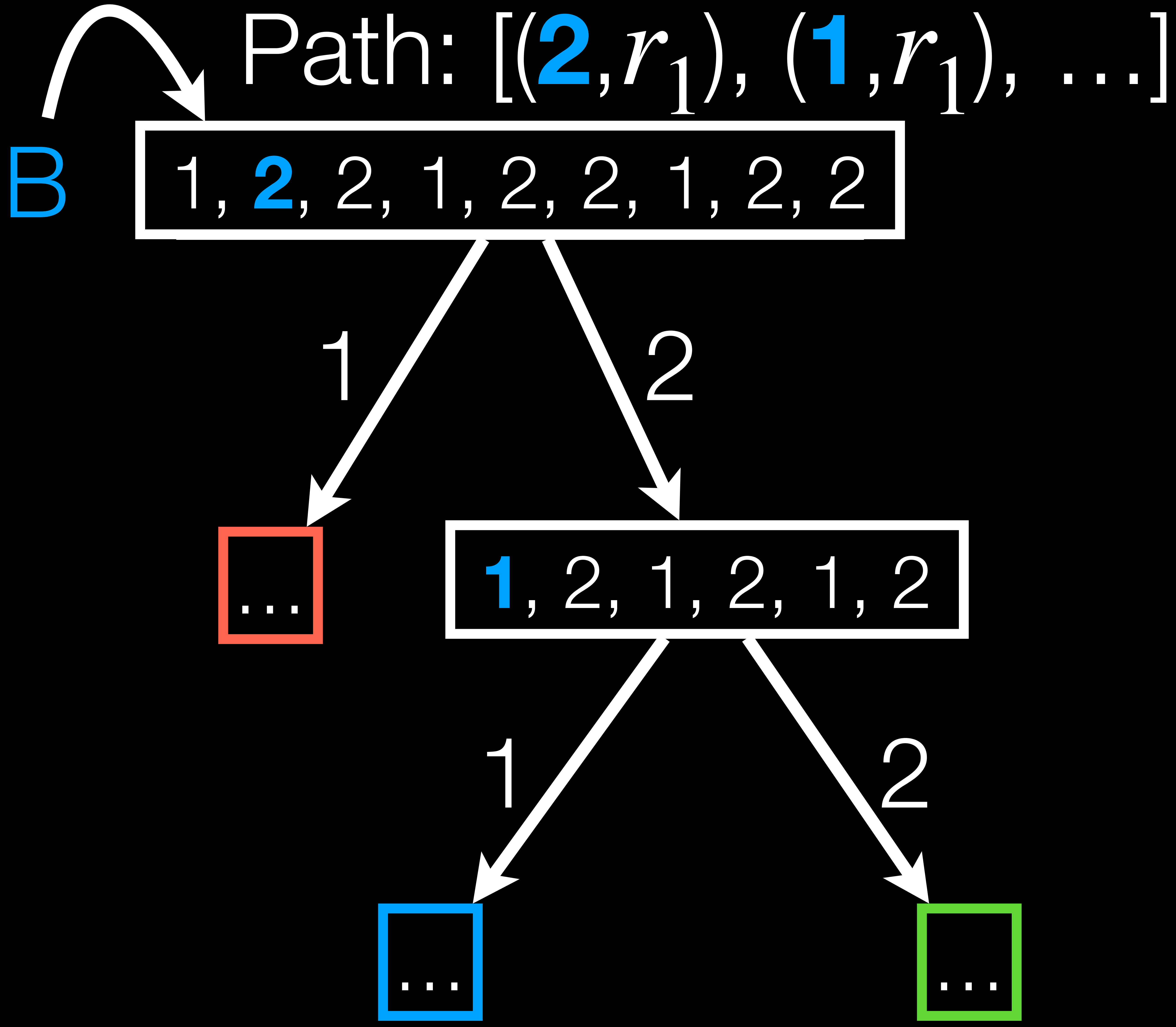


Compiling programs

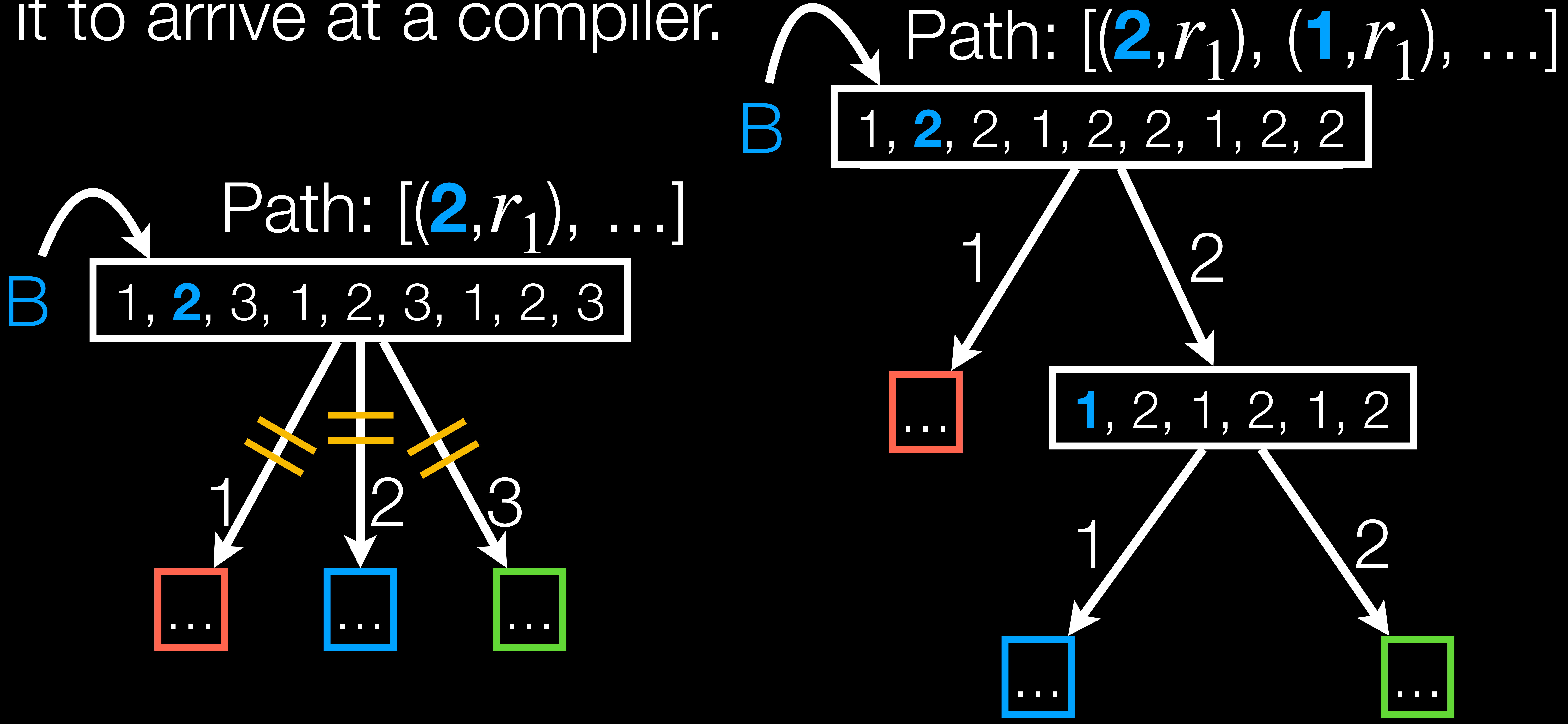


Compiling programs





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

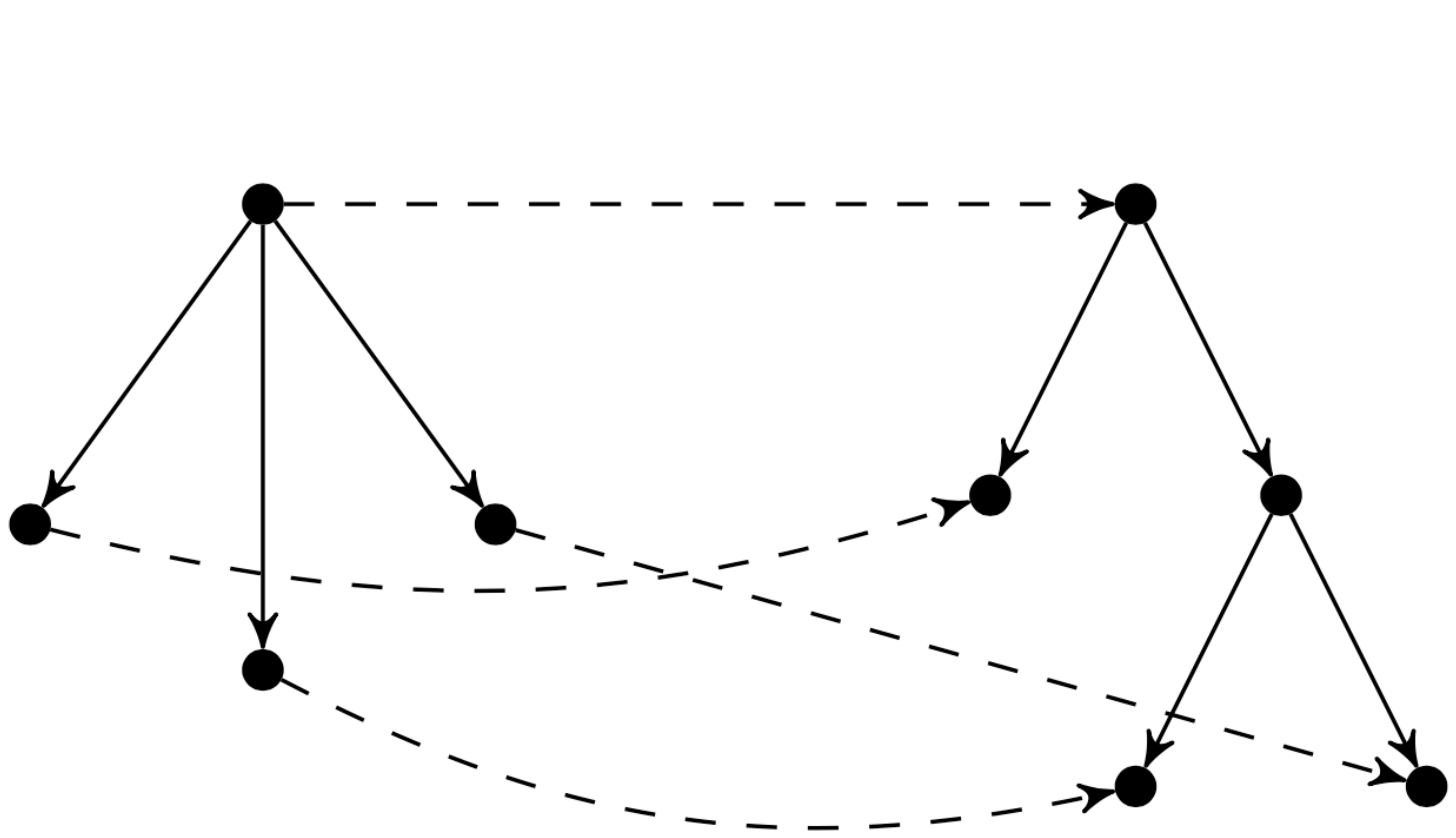


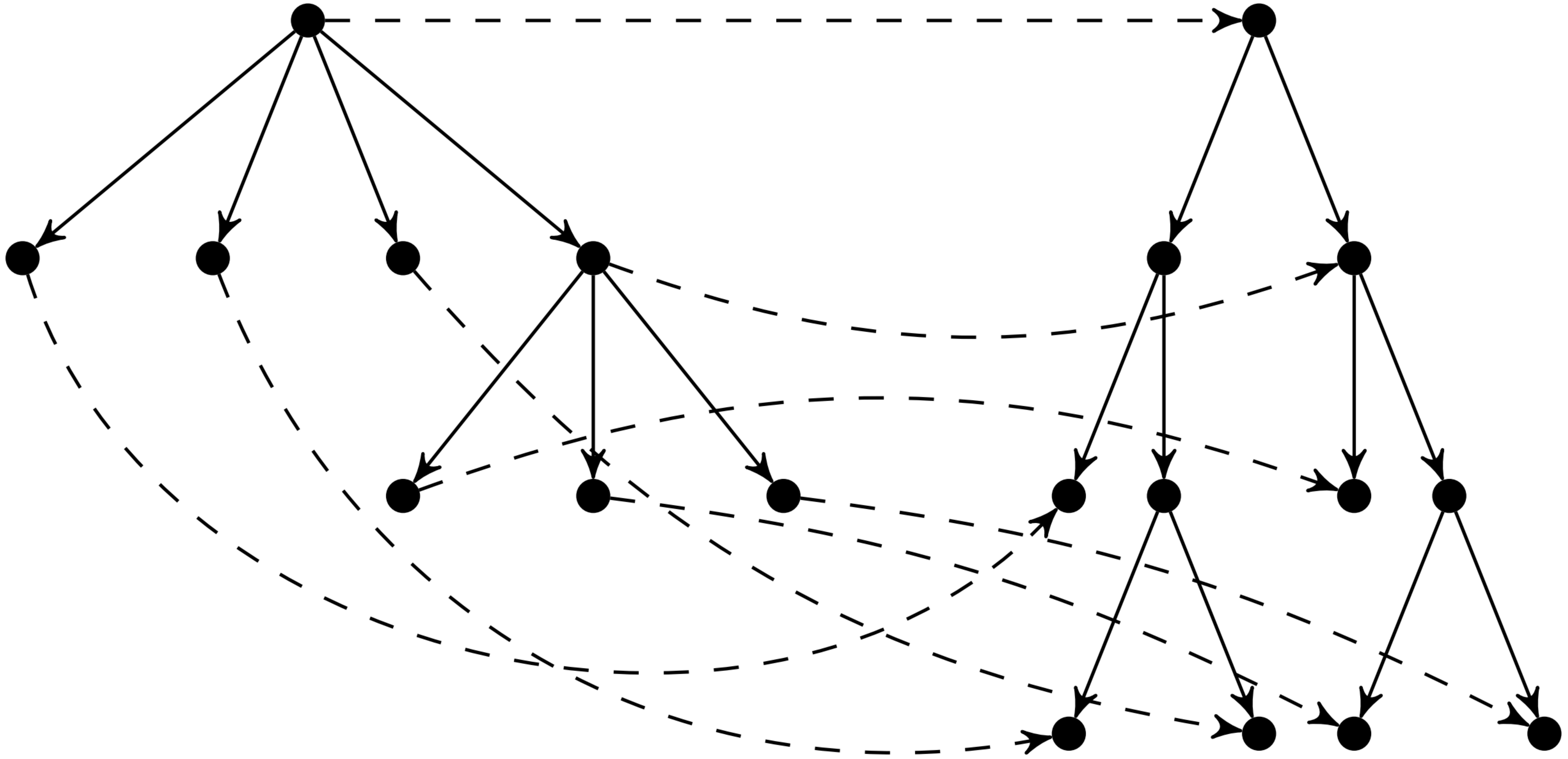
Generating embeddings automatically

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.*

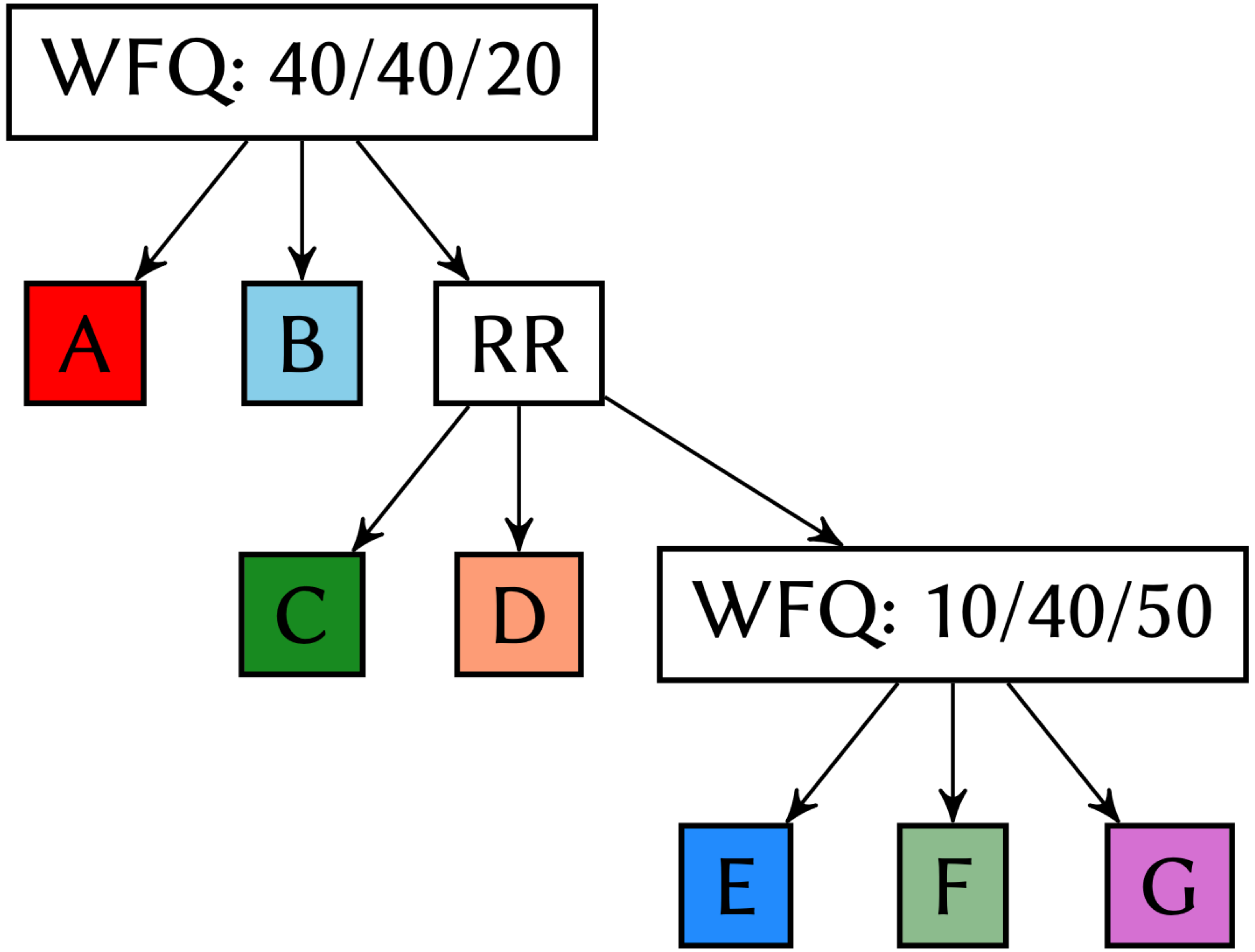
Generating embeddings automatically

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

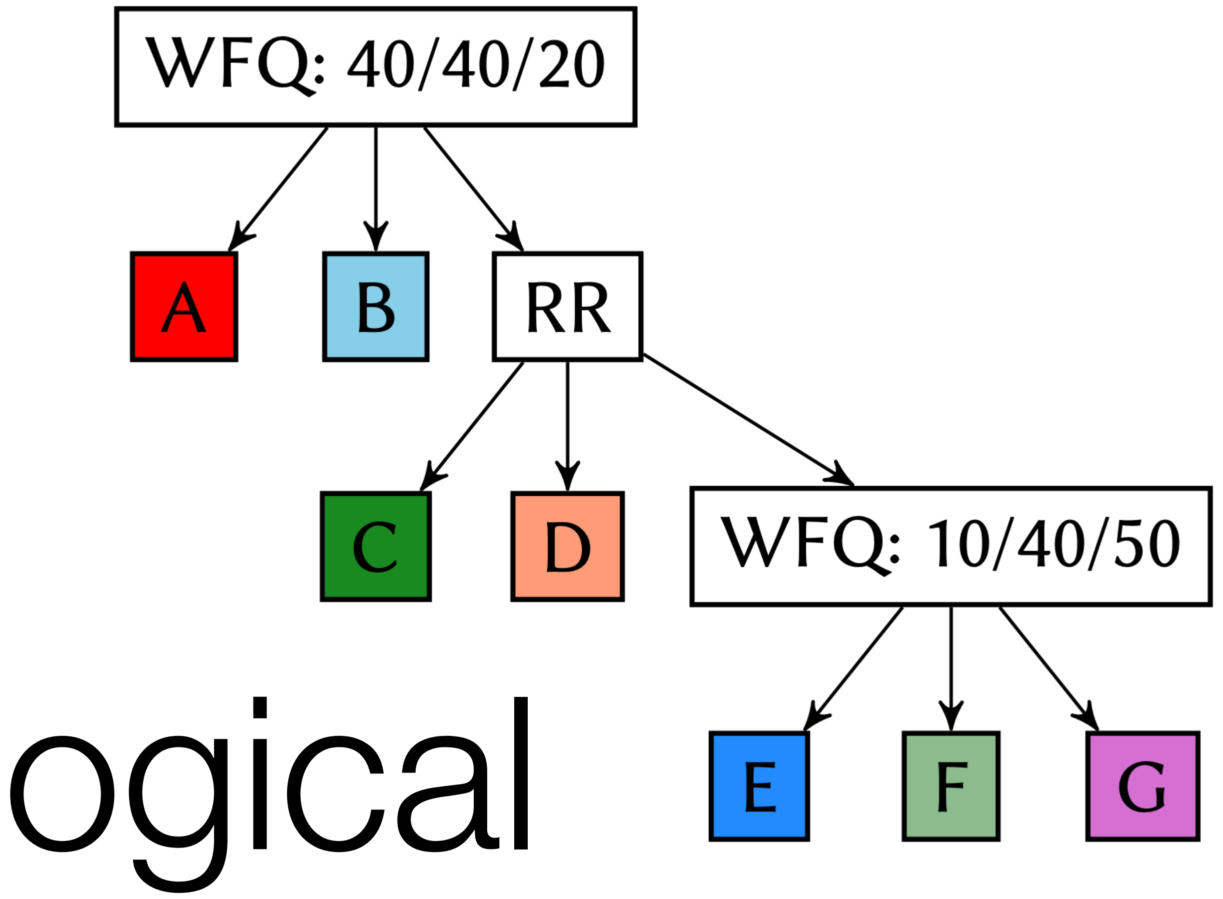
Two new algorithms,
both starting with heterogeneous source trees.

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

Workflow

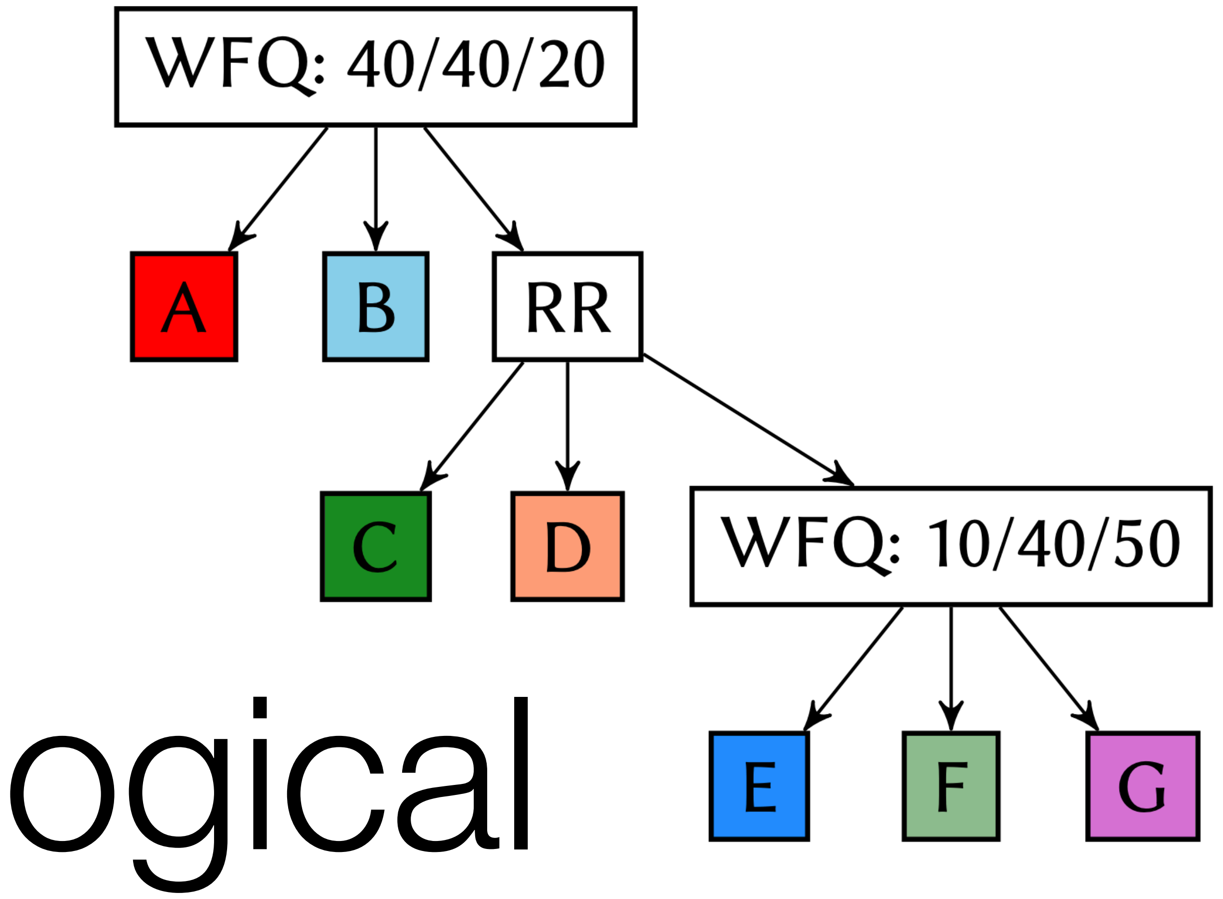


Workflow



logical

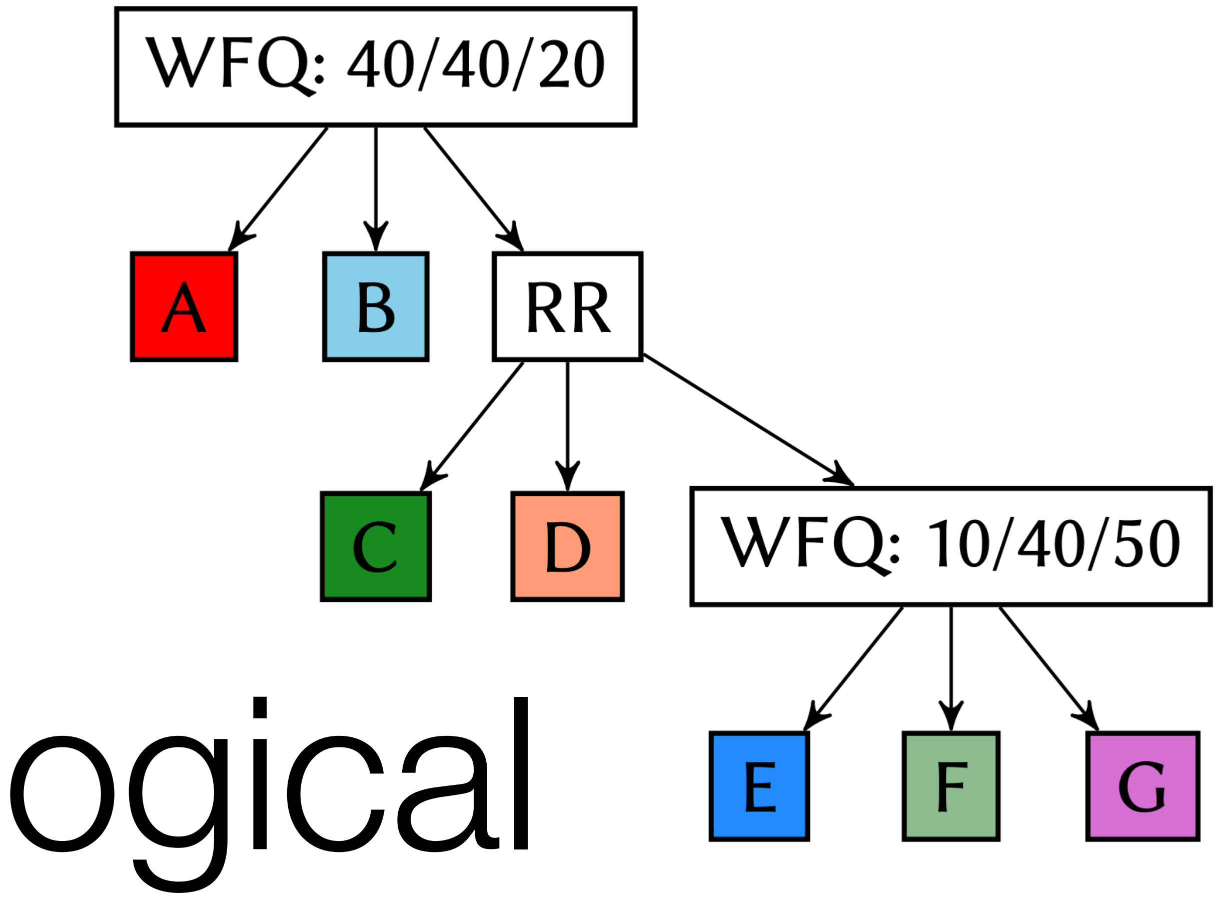
Workflow



logical

But the hardware supports
a regular-branching binary tree.

Workflow

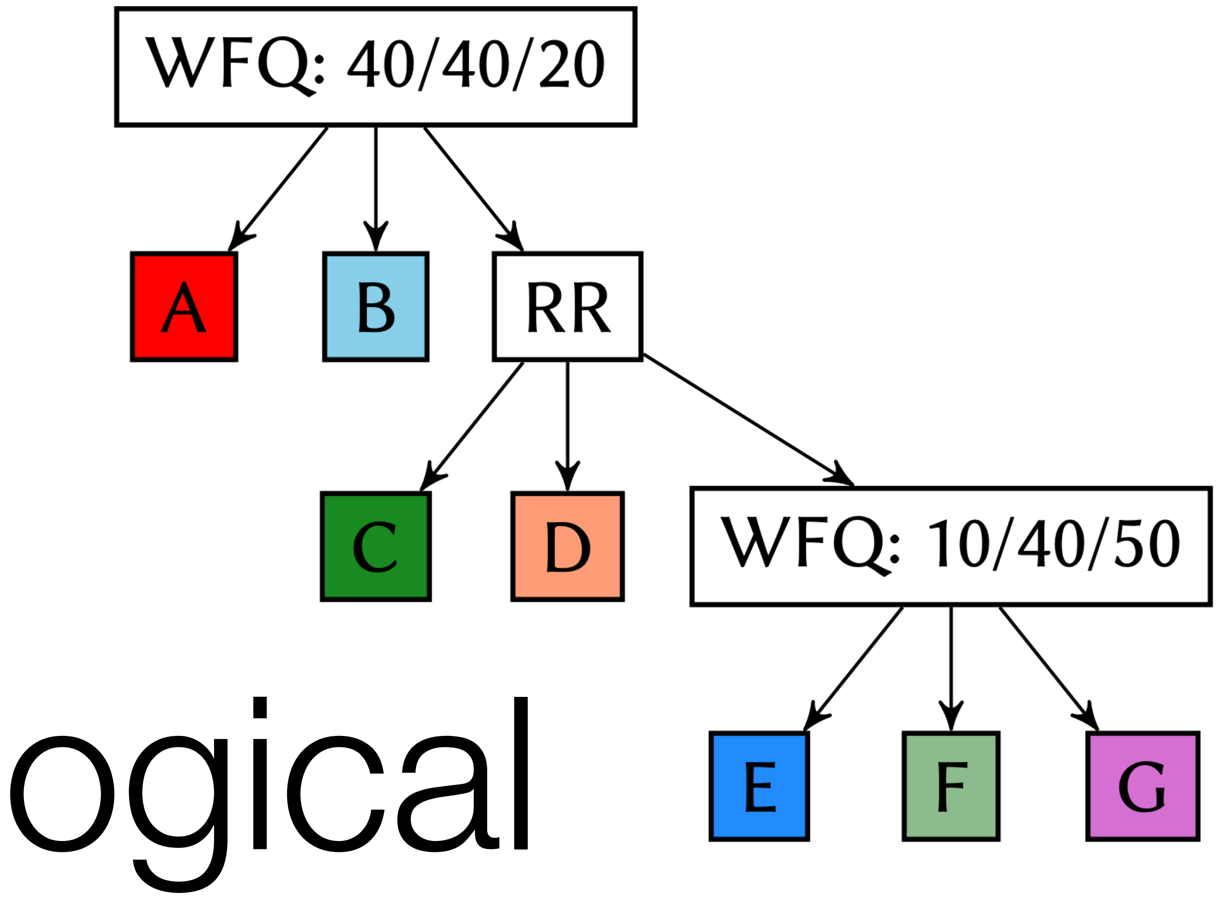


logical

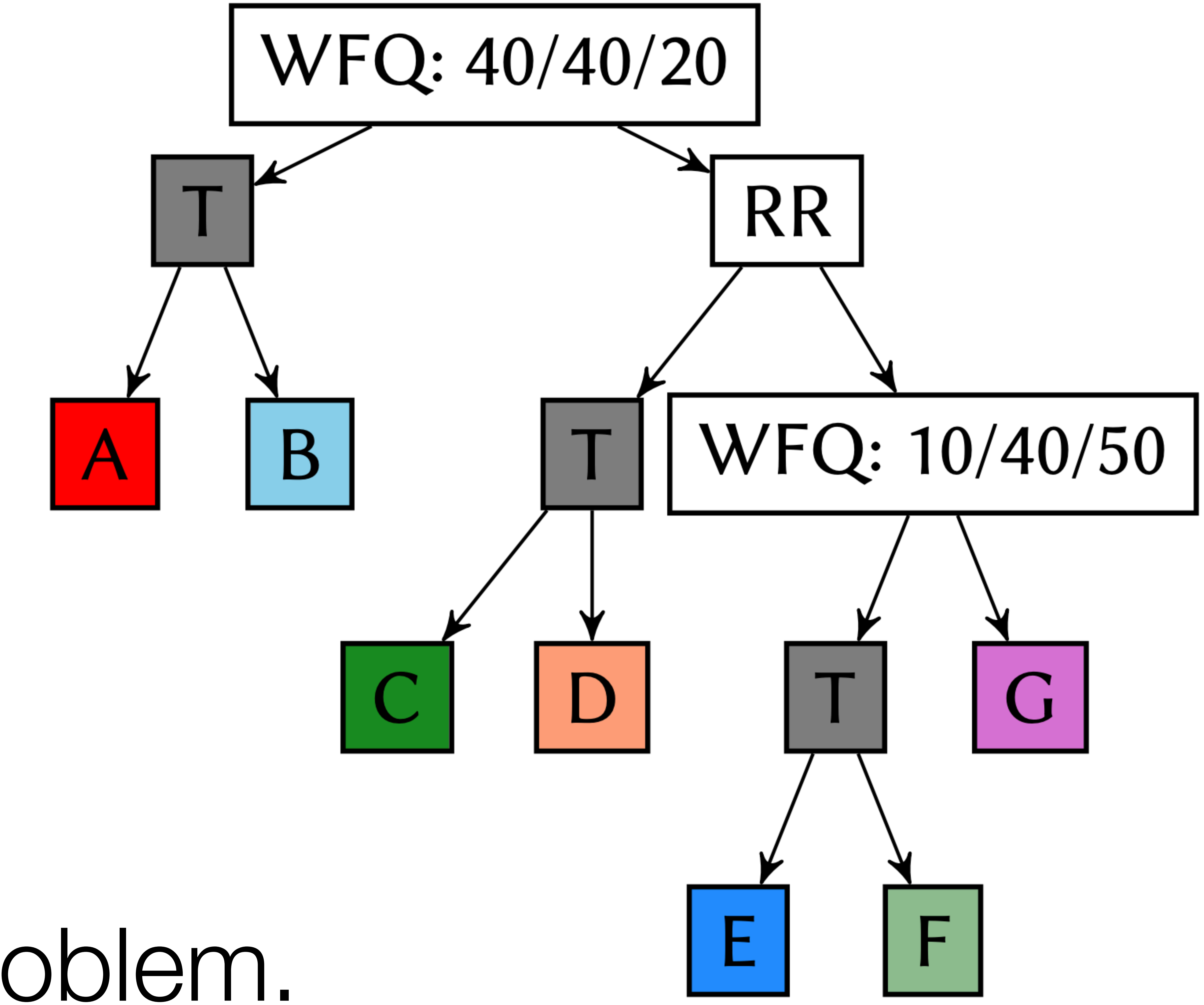
But the hardware supports
a regular-branching binary tree.

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

Workflow



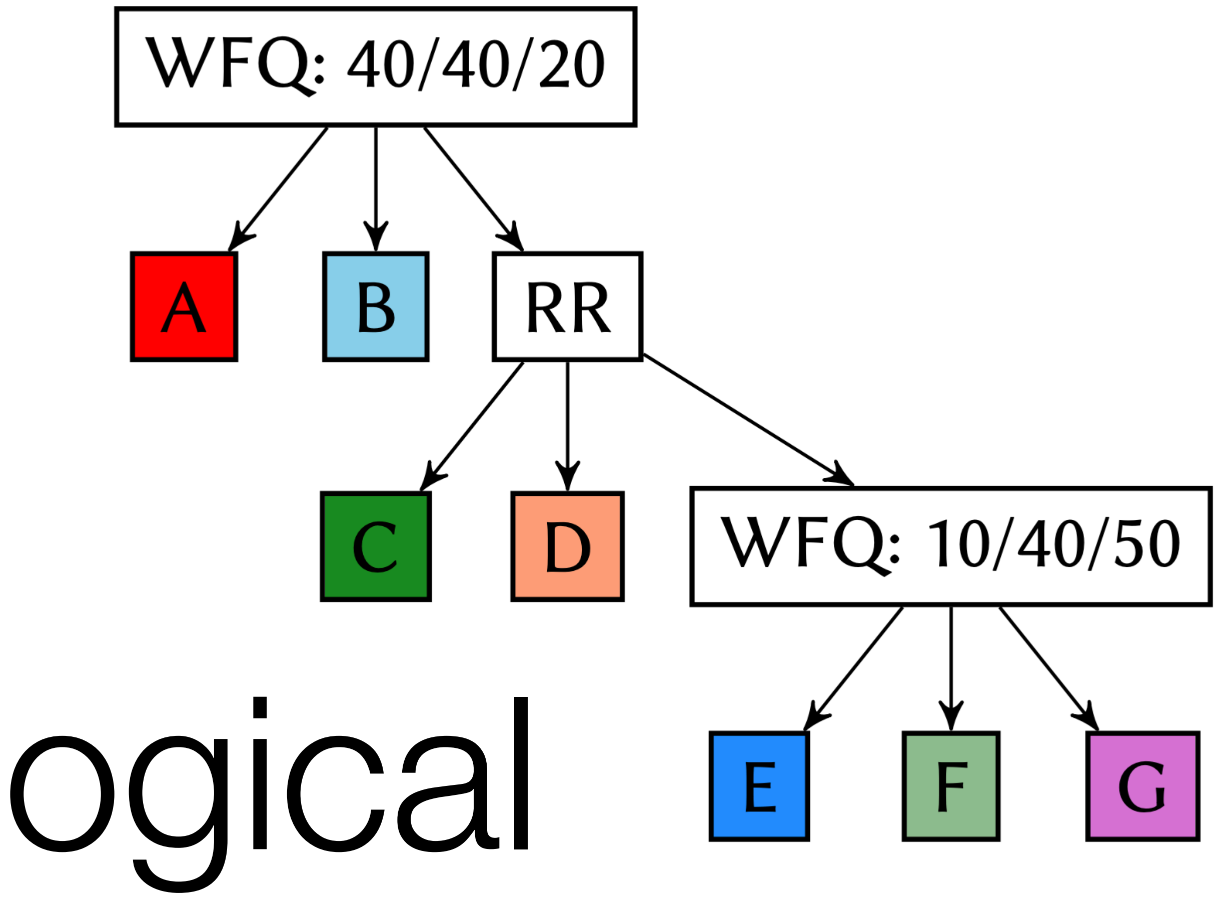
logical



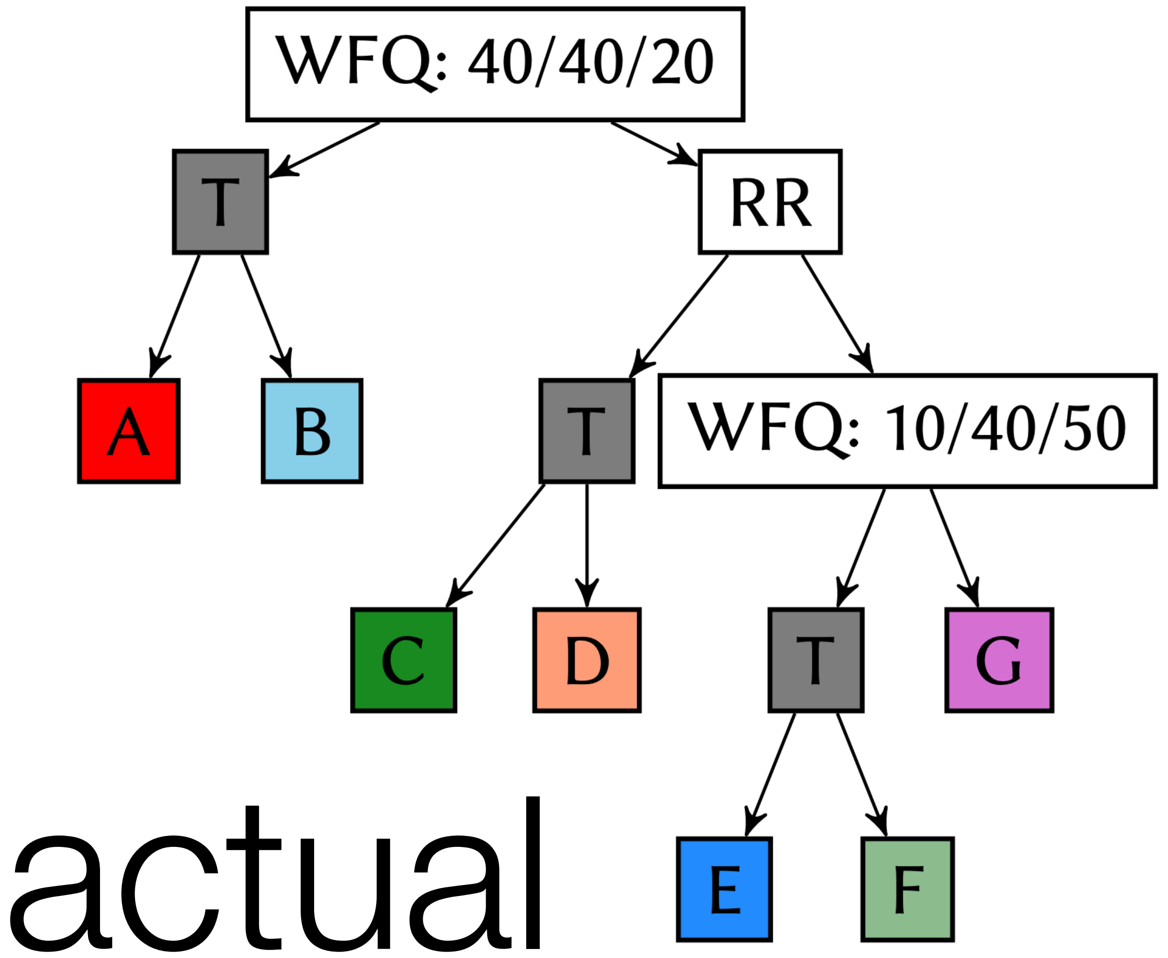
No problem.

Here’s how I’ll use that tree.

Workflow

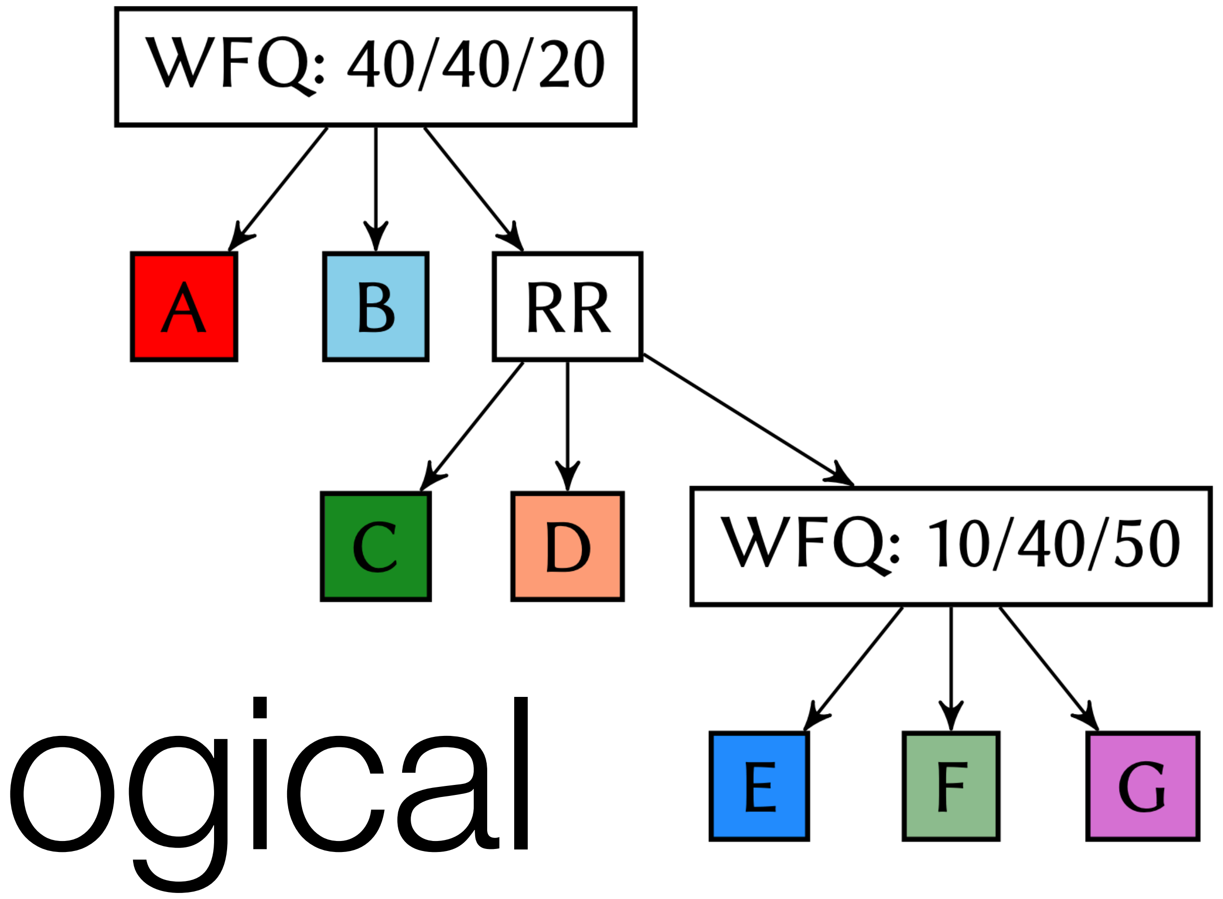


logical

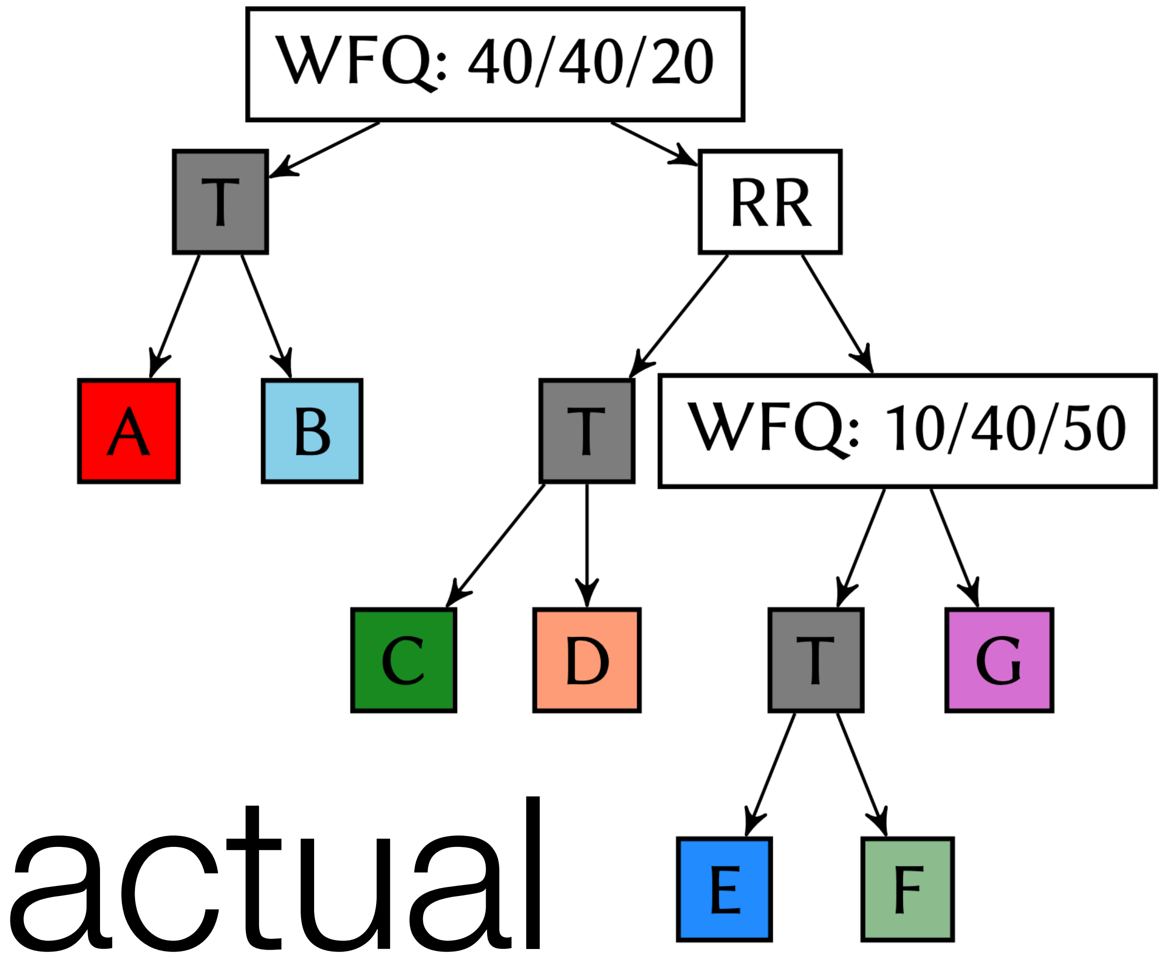


actual

Simulation

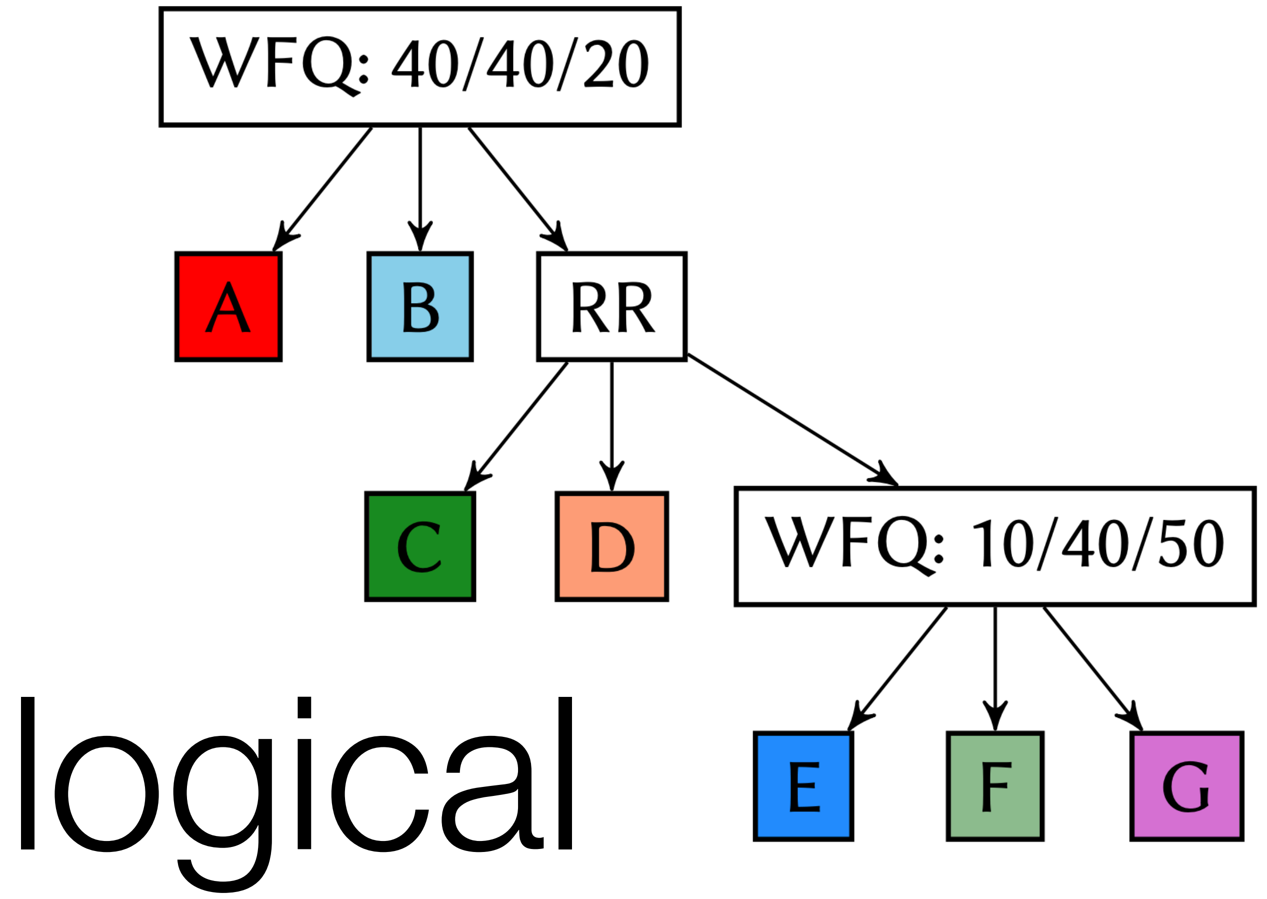


logical

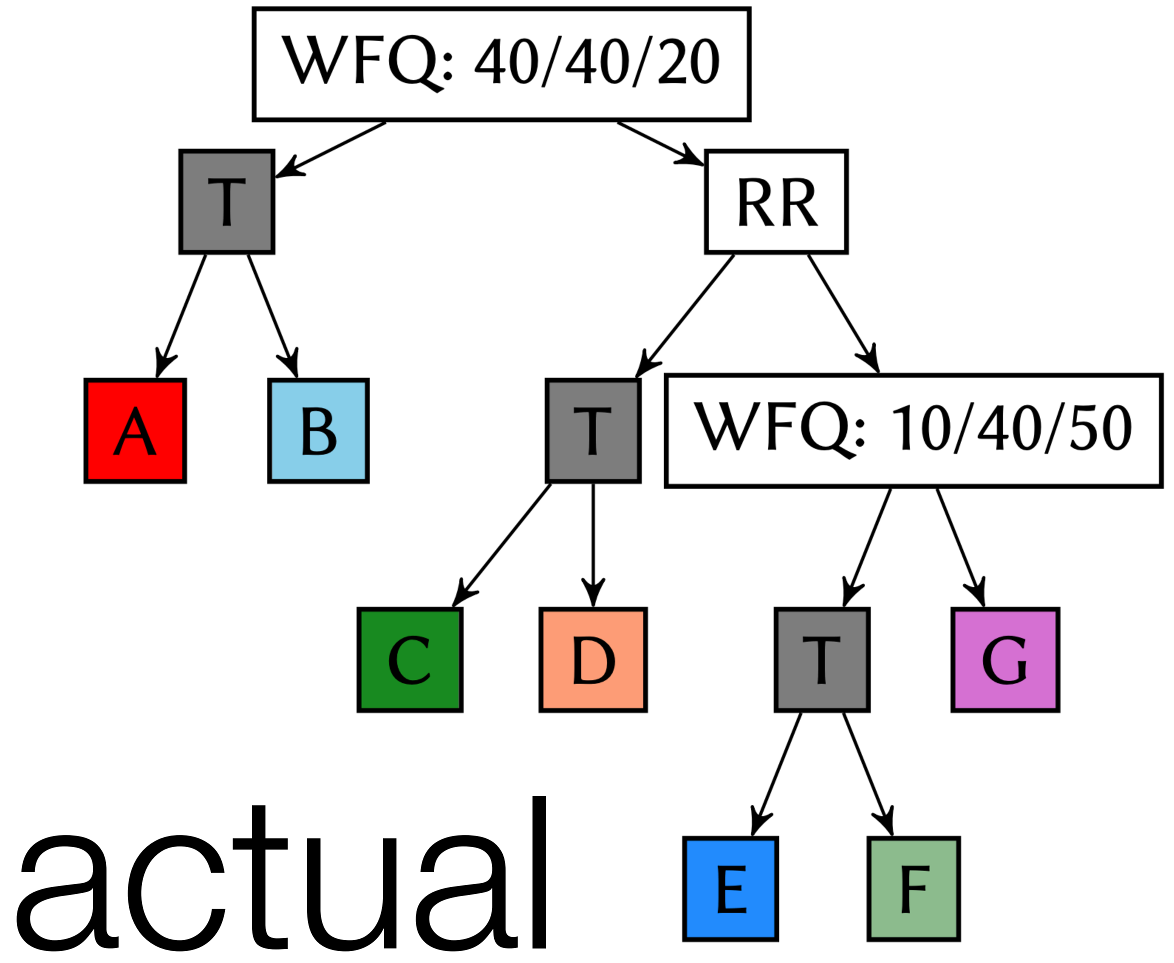


actual

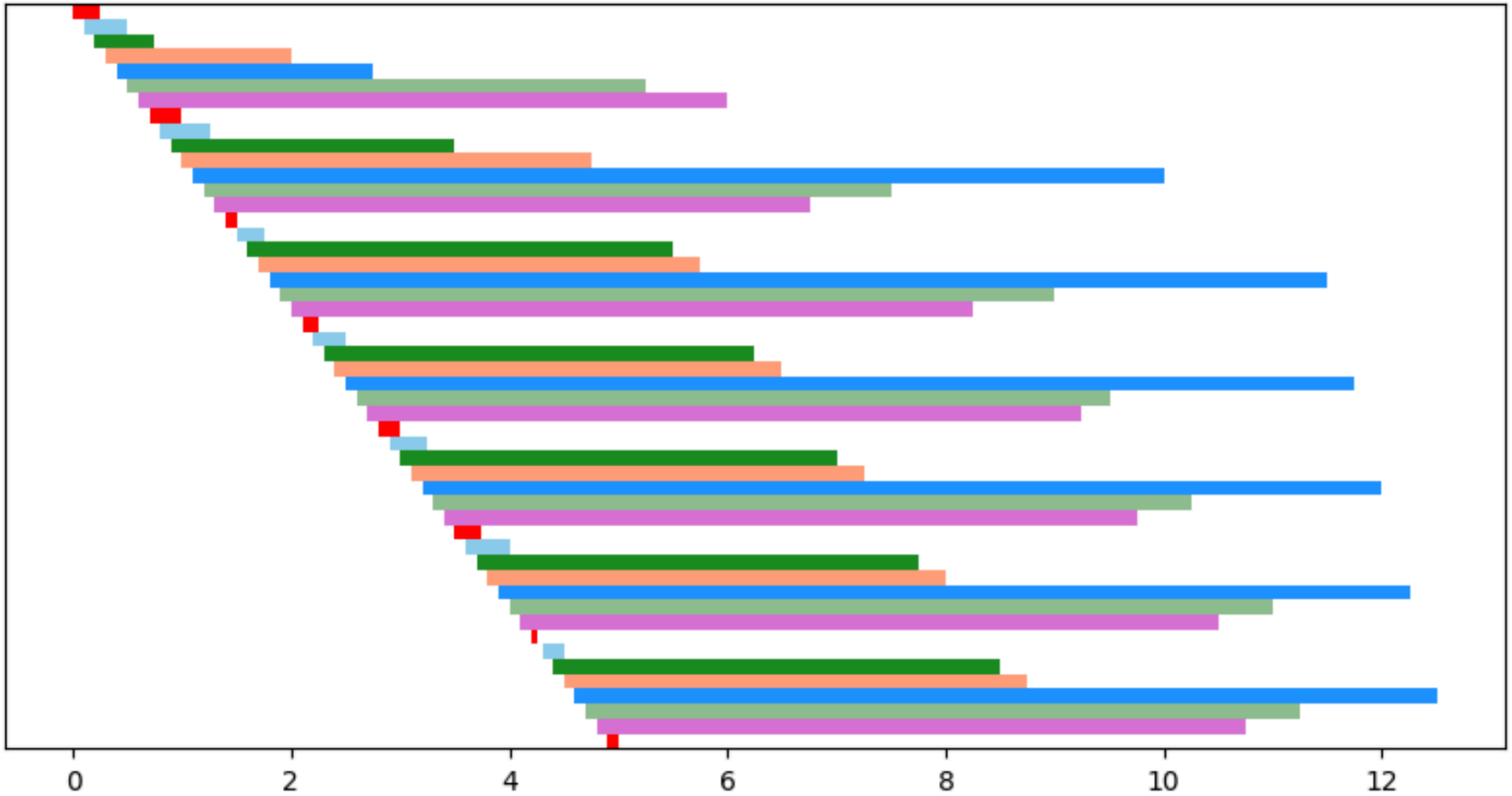
Simulation



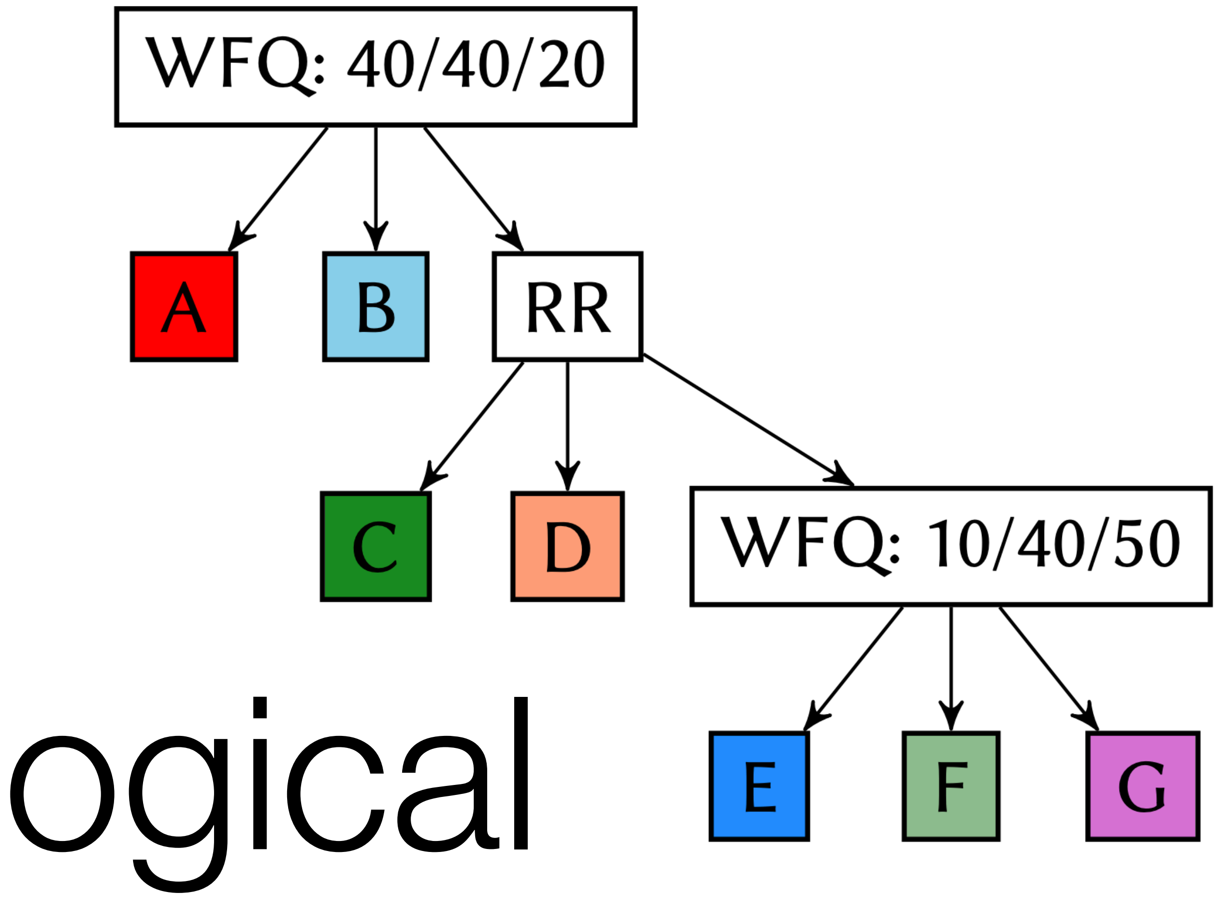
logical



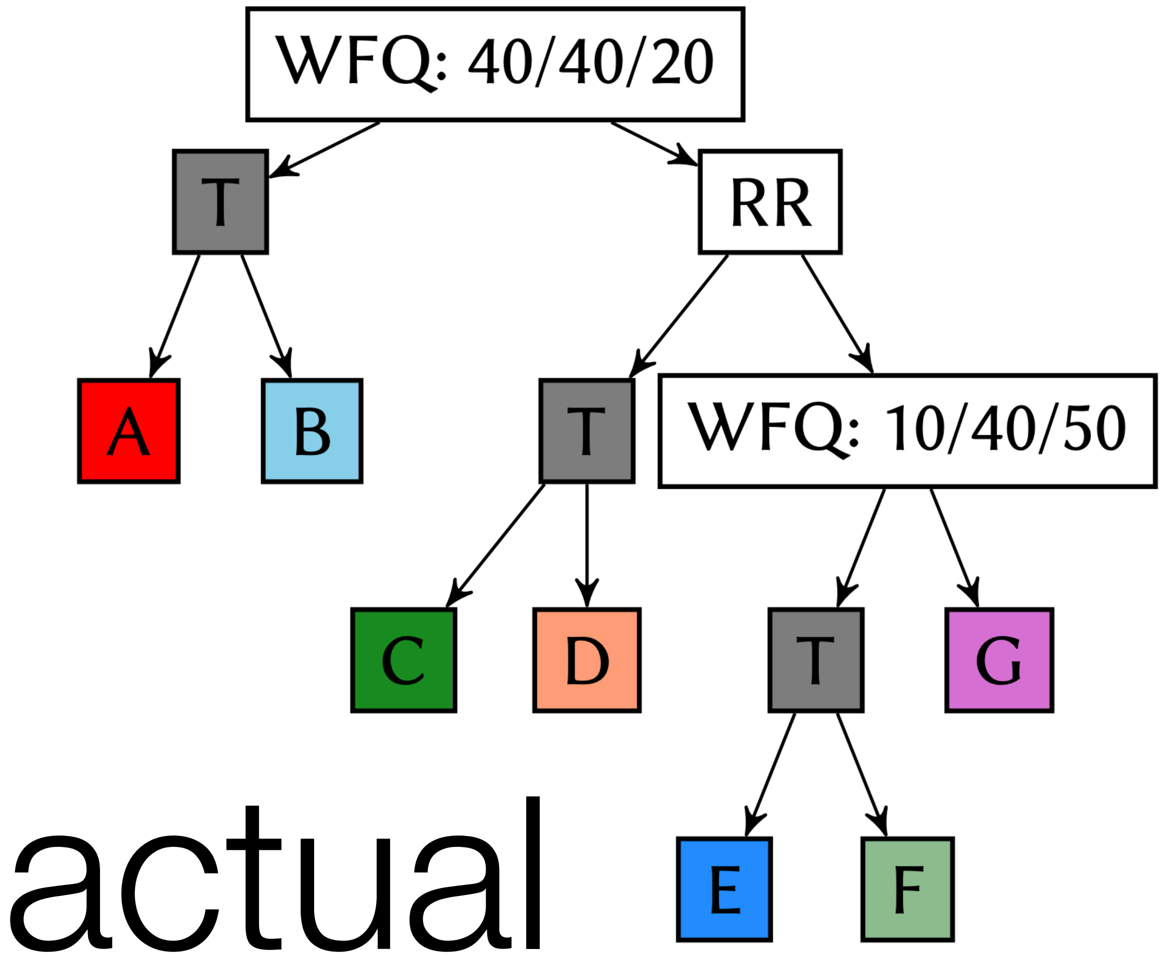
actual



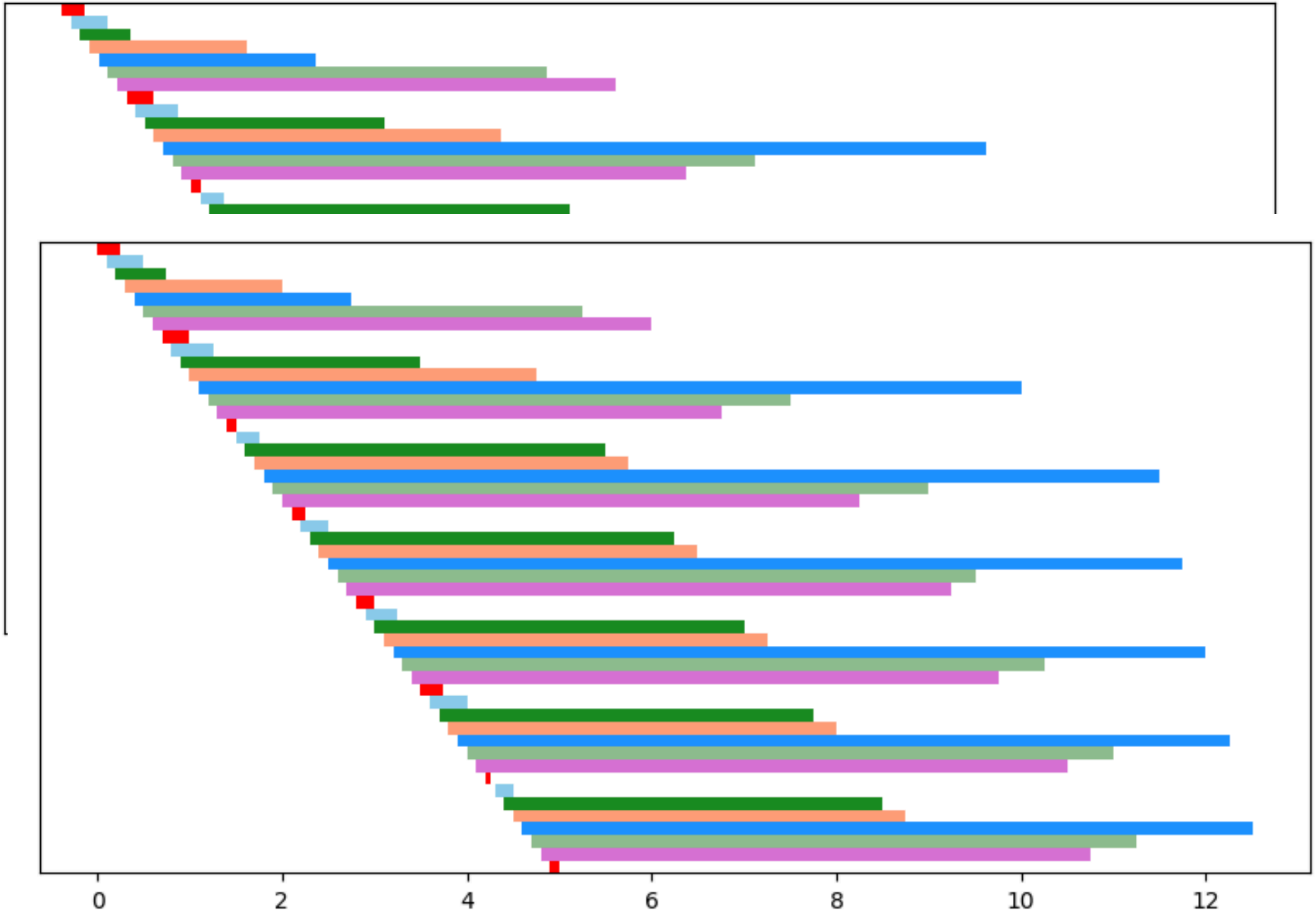
Simulation



logical



actual



Underlying formalism

Underlying formalism

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

Underlying formalism

$$\frac{\textsc{push}(p, pkt, r) = p'}{\text{push}(\text{Leaf}(p), pkt, r) = \text{Leaf}(p')}$$

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

Underlying formalism

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

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

Underlying formalism

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

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

Underlying formalism

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

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

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

Underlying formalism

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

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

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

Underlying formalism

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

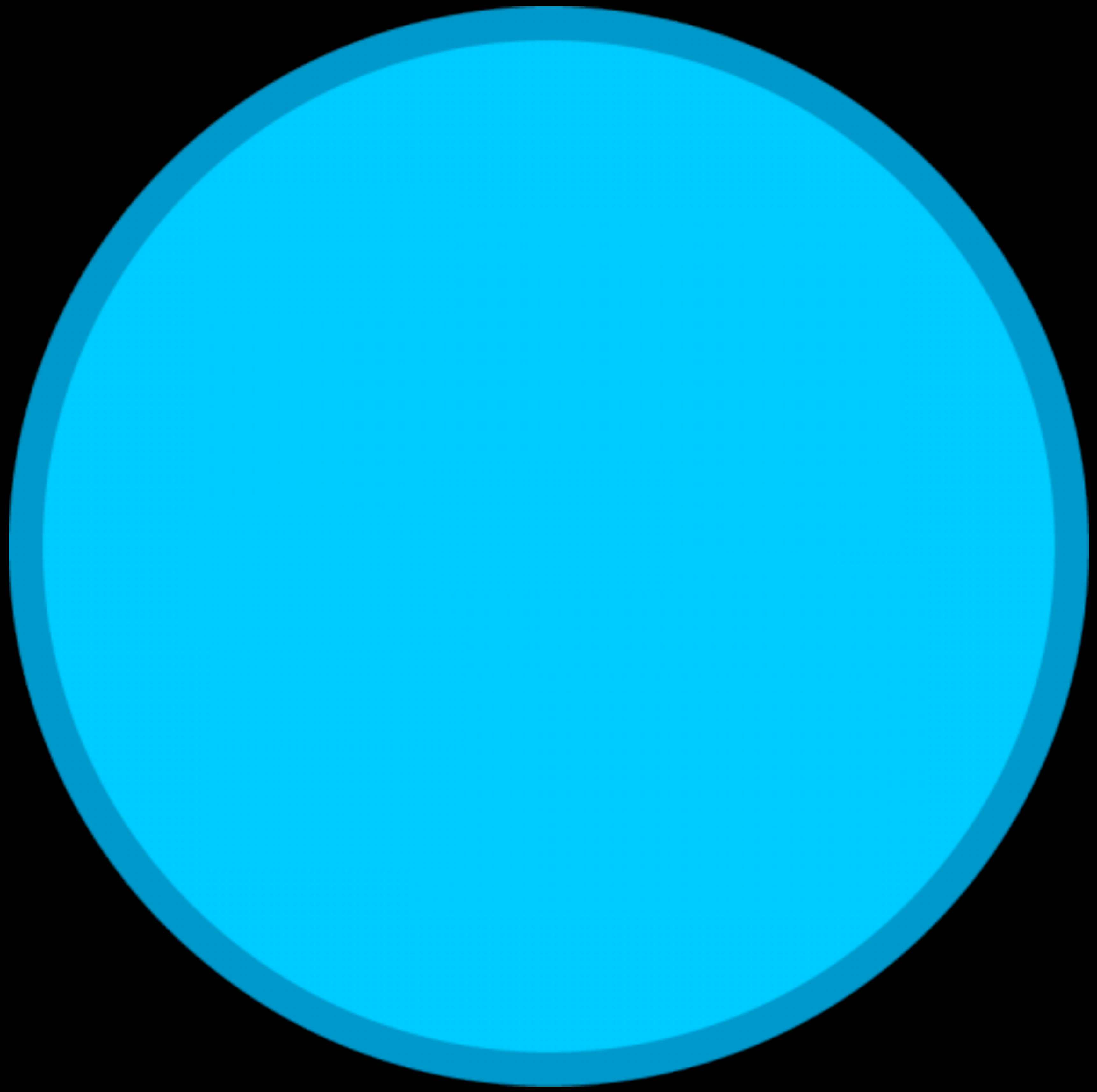
$$\frac{p \in \mathsf{PIFO}(\mathsf{Pkt})}{\mathsf{Leaf}(p) \in \mathsf{PIFOTree}\underbrace{(*)}_{\text{Topo}}} \qquad \frac{\begin{array}{c} n \in \mathbb{N} \qquad ts \in \mathsf{Topo}^{n} \qquad p \in \mathsf{PIFO}(\{1, \ldots, n\}) \\ \forall 1 \leq i \leq n.\ qs[i] \in \mathsf{PIFOTree}(ts[i]) \end{array}}{\mathsf{Internal}(qs, p) \in \mathsf{PIFOTree}(\underbrace{\mathsf{Node}(ts)}_{\text{Topo}})}$$

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

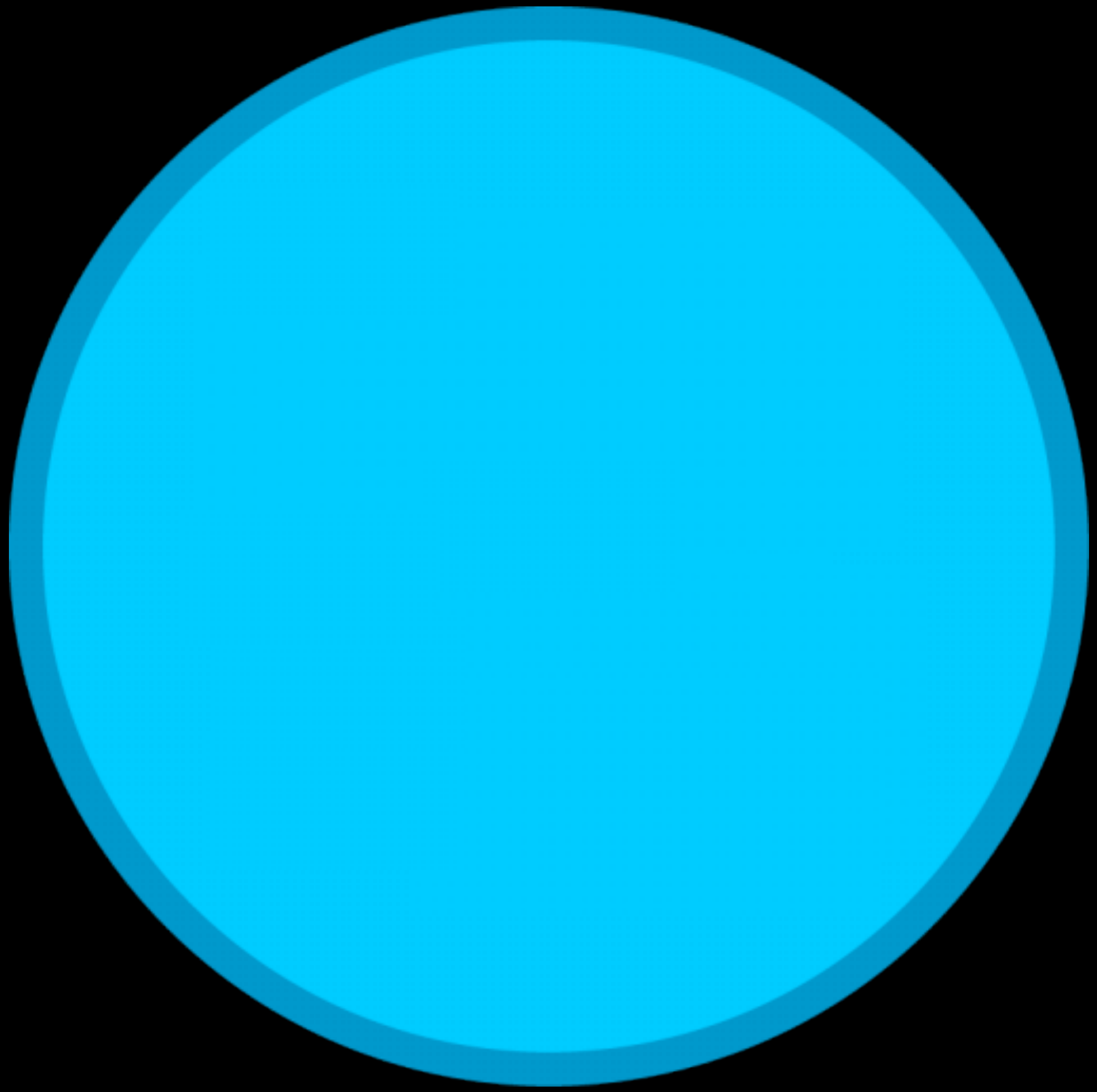
$$\frac{\textsc{push}(p, pkt, r) = p'}{\mathsf{push}(\mathsf{Leaf}(p), pkt, r) = \underbrace{\mathsf{Leaf}(p')}_{\text{PIFOTree}}} \qquad \frac{\mathsf{push}(qs[i], pkt, pt) = q' \qquad \textsc{push}(p, i, r) = p'}{\mathsf{push}(\mathsf{Internal}(qs, p), pkt, \underbrace{(i, r) :: pt}_{\text{Path}})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

A general way to deploy PIFO trees

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 hardware support some tree.

A general way to deploy PIFO trees

Let the human program against some tree.

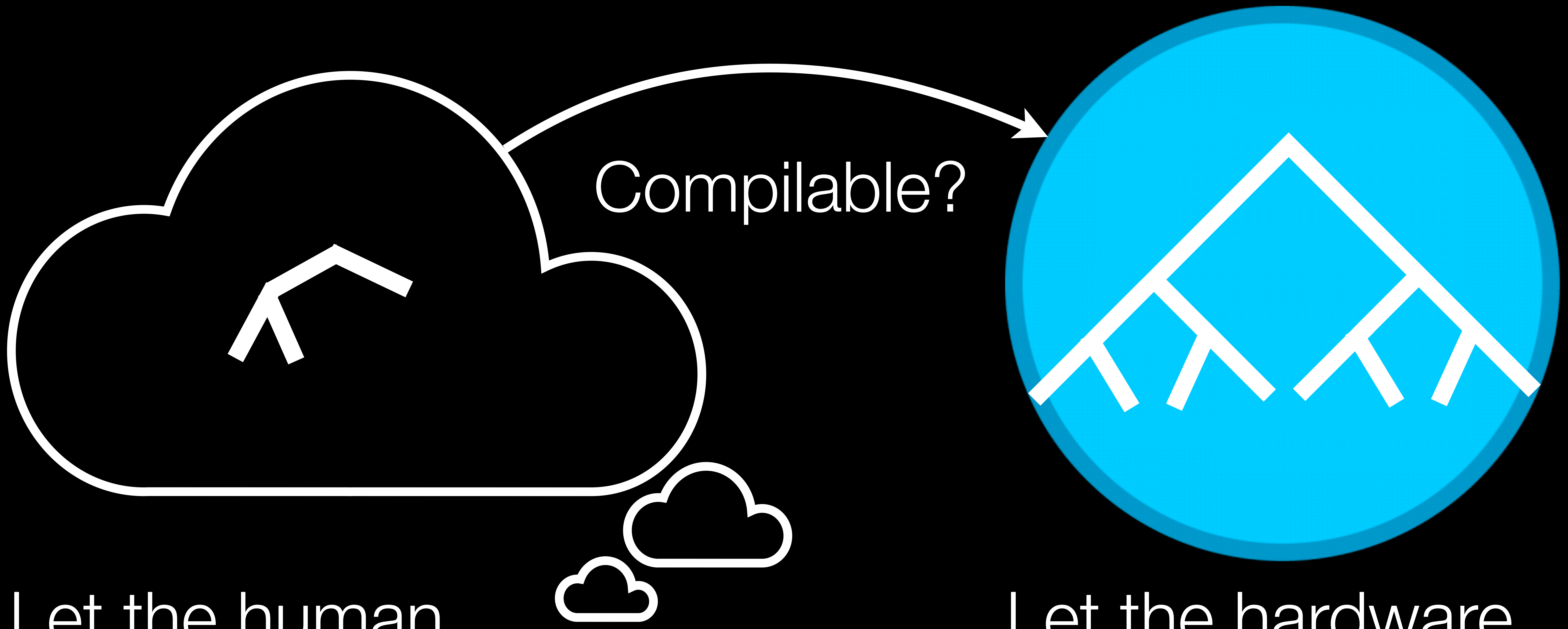
Let the hardware support some tree.

A general way to deploy PIFO trees

Let the human
program against some tree.

Let the hardware
support some tree.

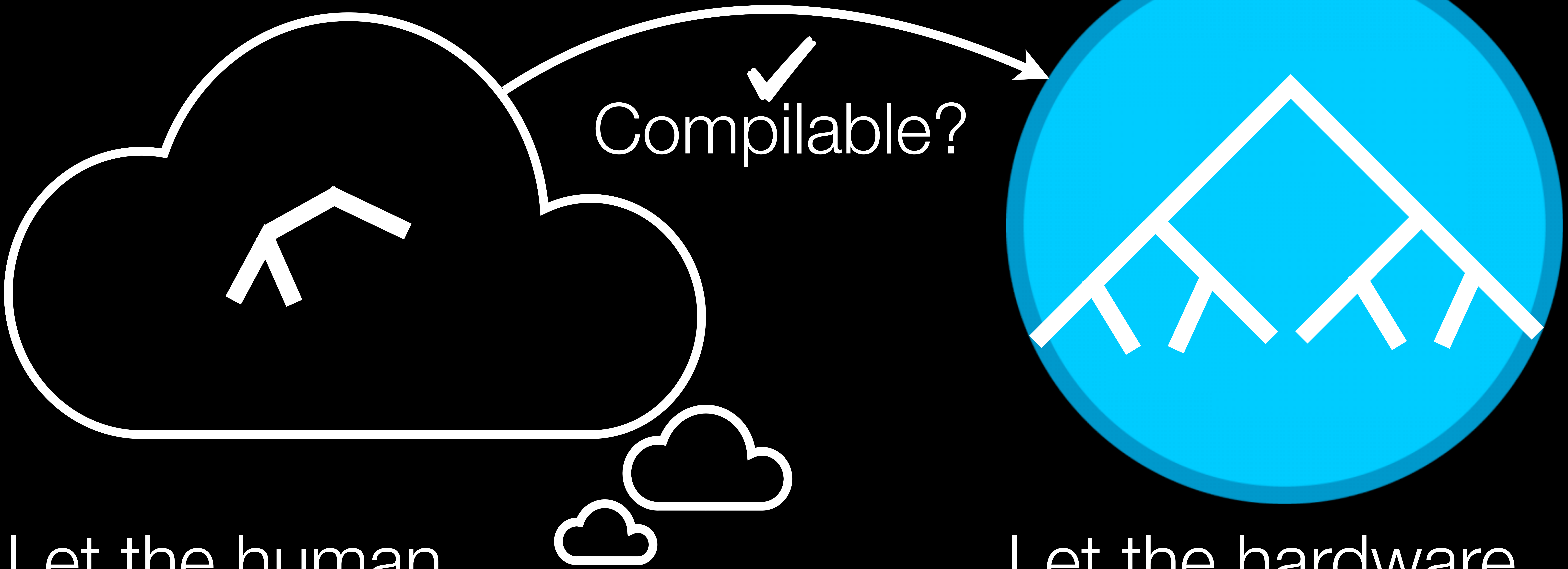
A general way to deploy PIFO trees



Let the human program against some tree.

Let the hardware support some tree.

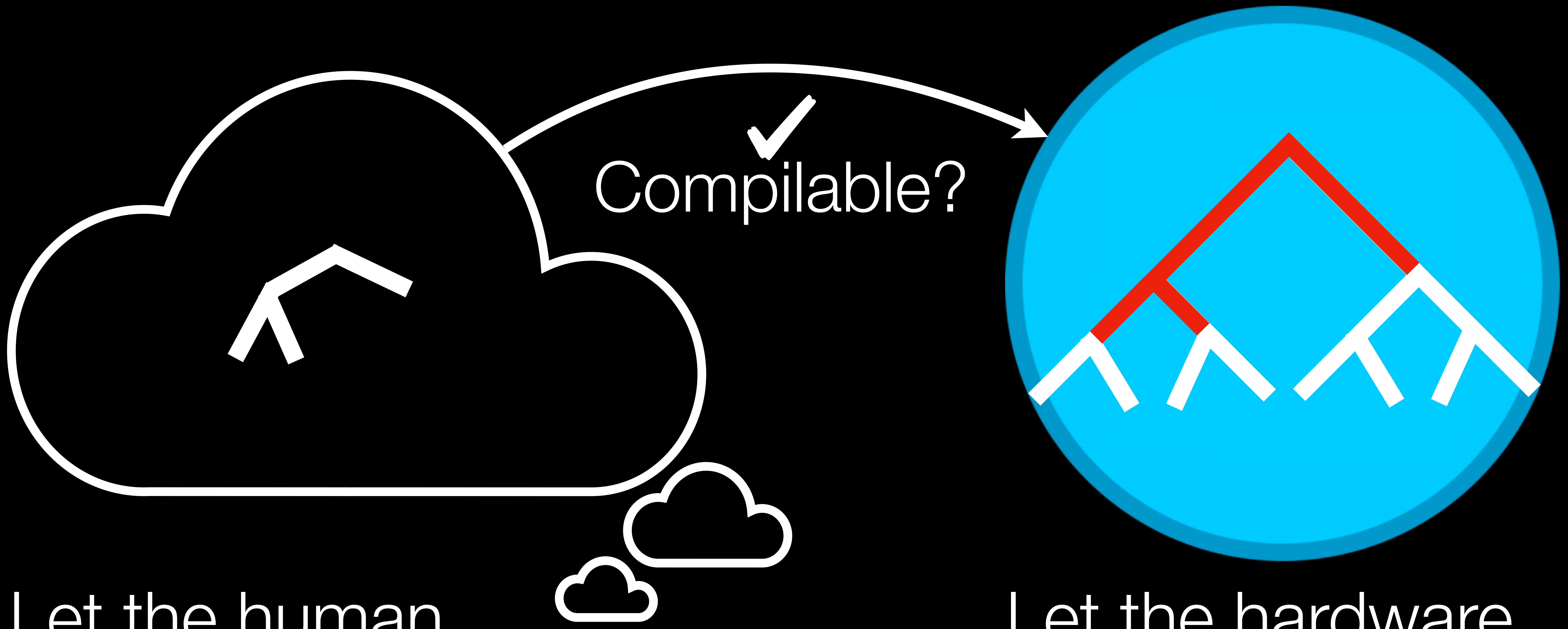
A general way to deploy PIFO trees



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.

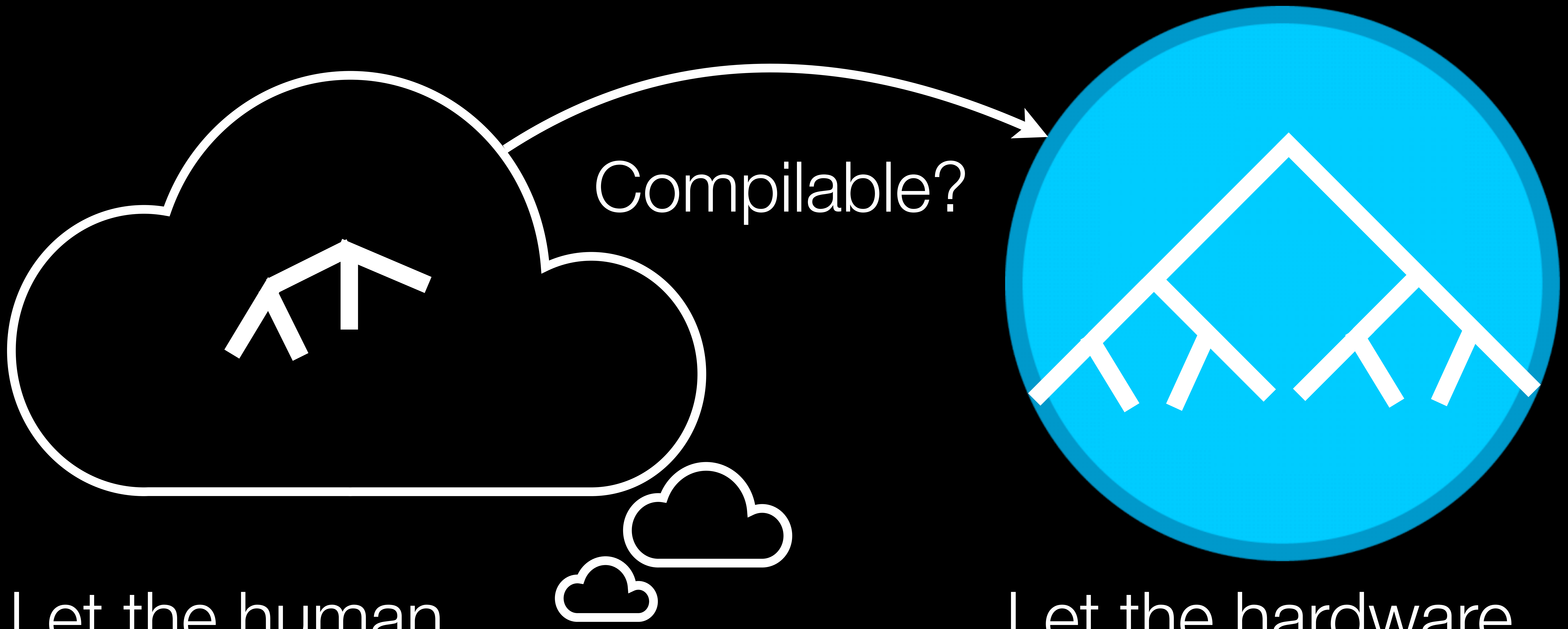
Let the hardware
support some tree.

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

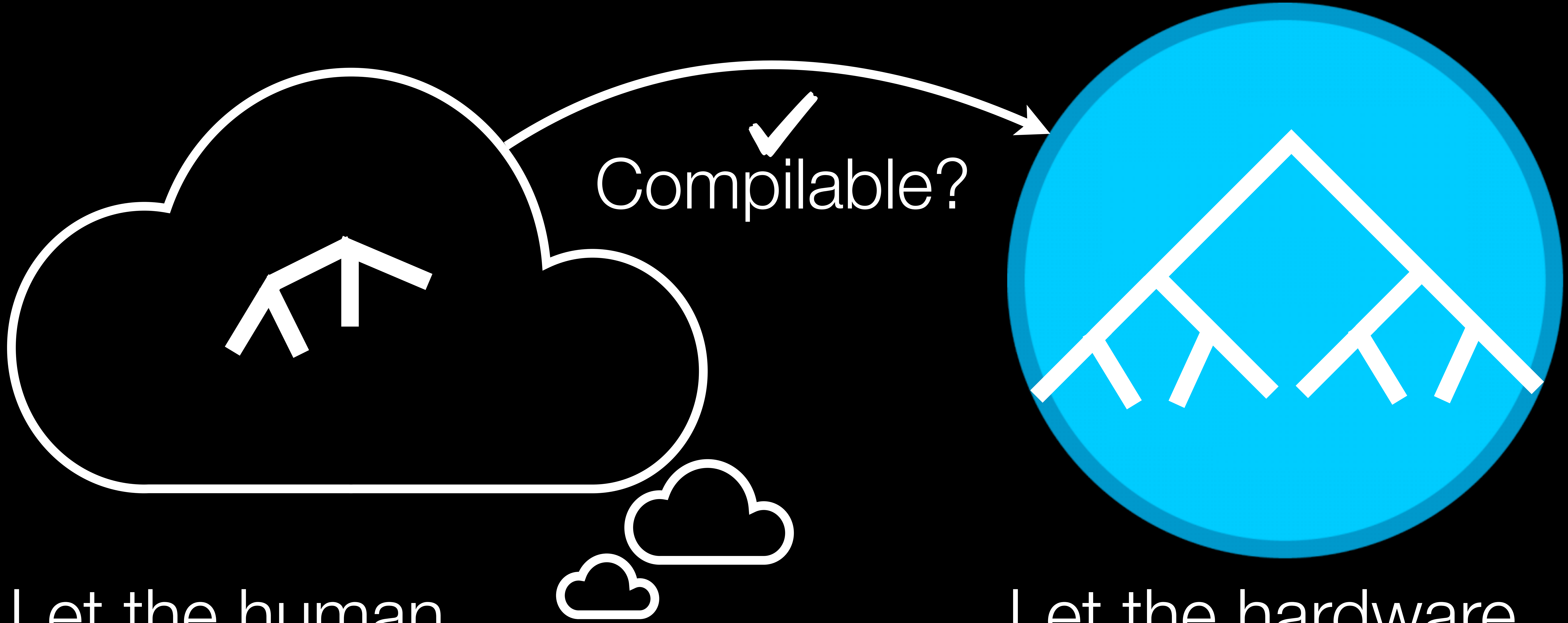


Compilable?

Let the human program against some tree.

Let the hardware support some tree.

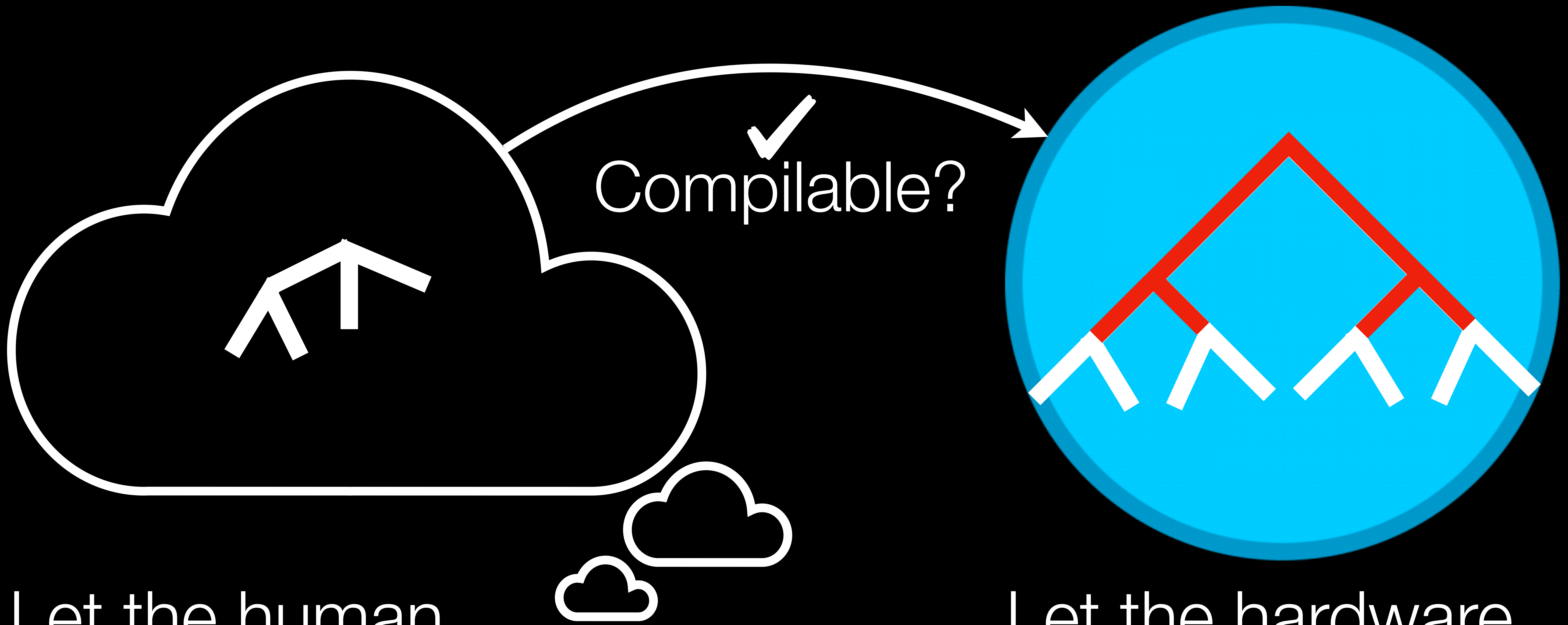
A general way to deploy PIFO trees



Let the human program against some tree.

Let the hardware support some tree.

A general way to deploy PIFO trees



✓ Compilable?

Let the human program against some tree.

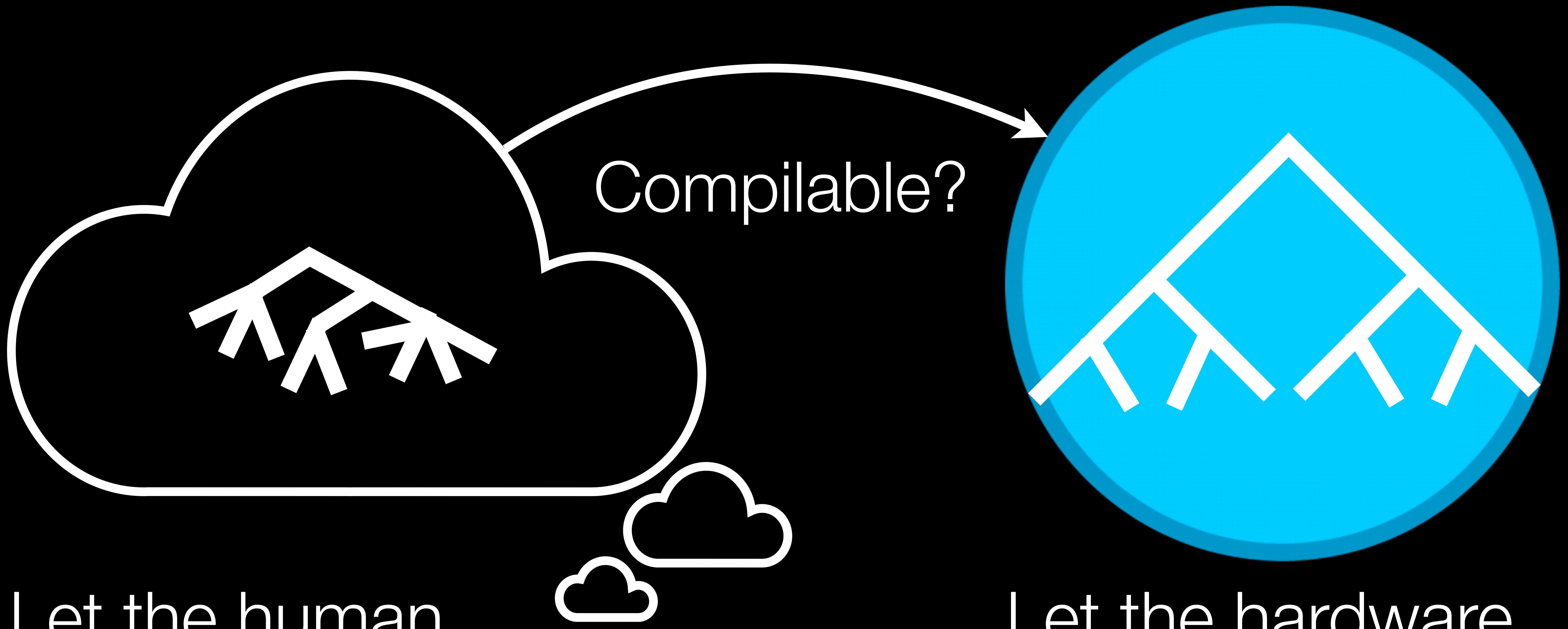
Let the hardware support some tree.

A general way to deploy PIFO trees

Let the human program against some tree.

Let the hardware support some tree.

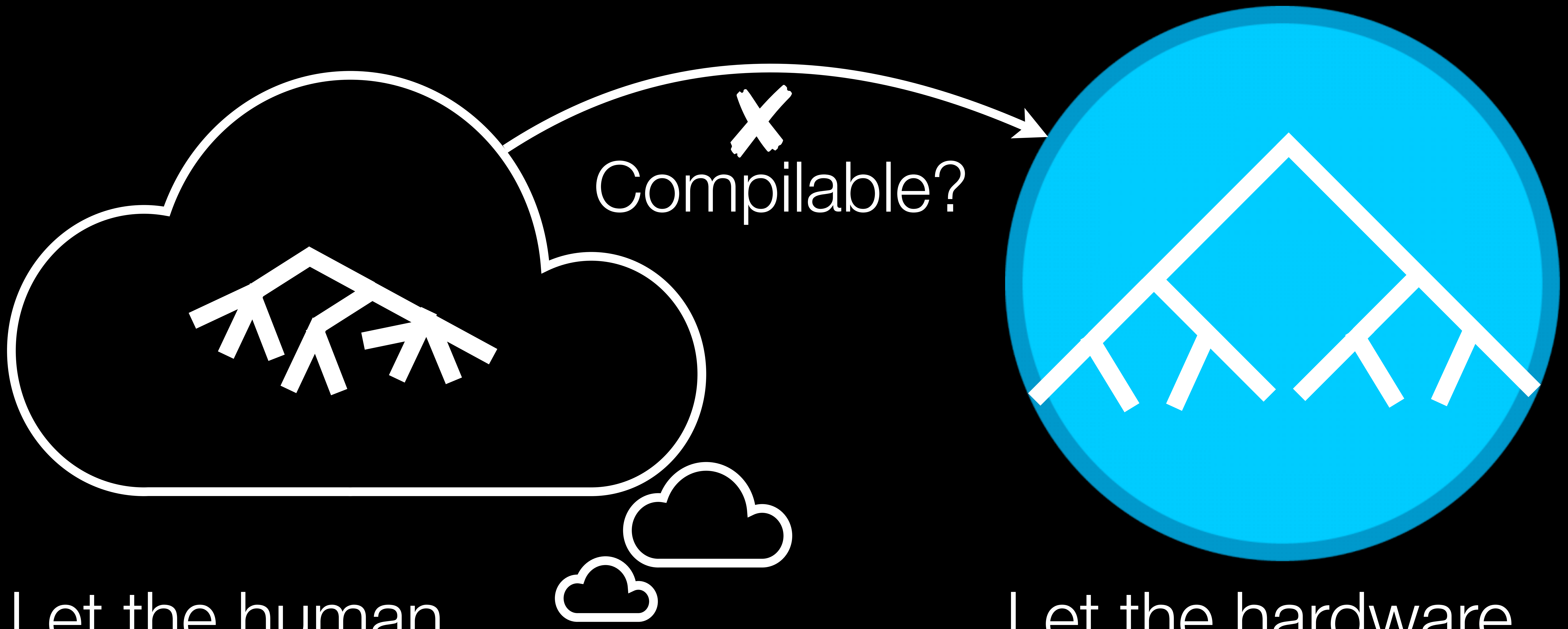
A general way to deploy PIFO trees



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