Scaling Data Plane Verification via Parallelization

Sisi Wen, Anubhavnidhi Abhashkumar, Chenyang Zhao, Weirong Jiang

ByteDance 字节跳动

Outline

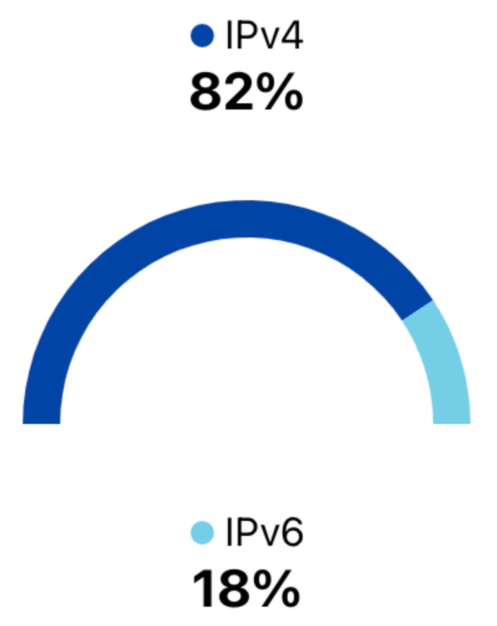
- Background & Motivation
- Challenges
- RANGESET
- Workflow
- Evaluation
- Future Work & Summary

Background

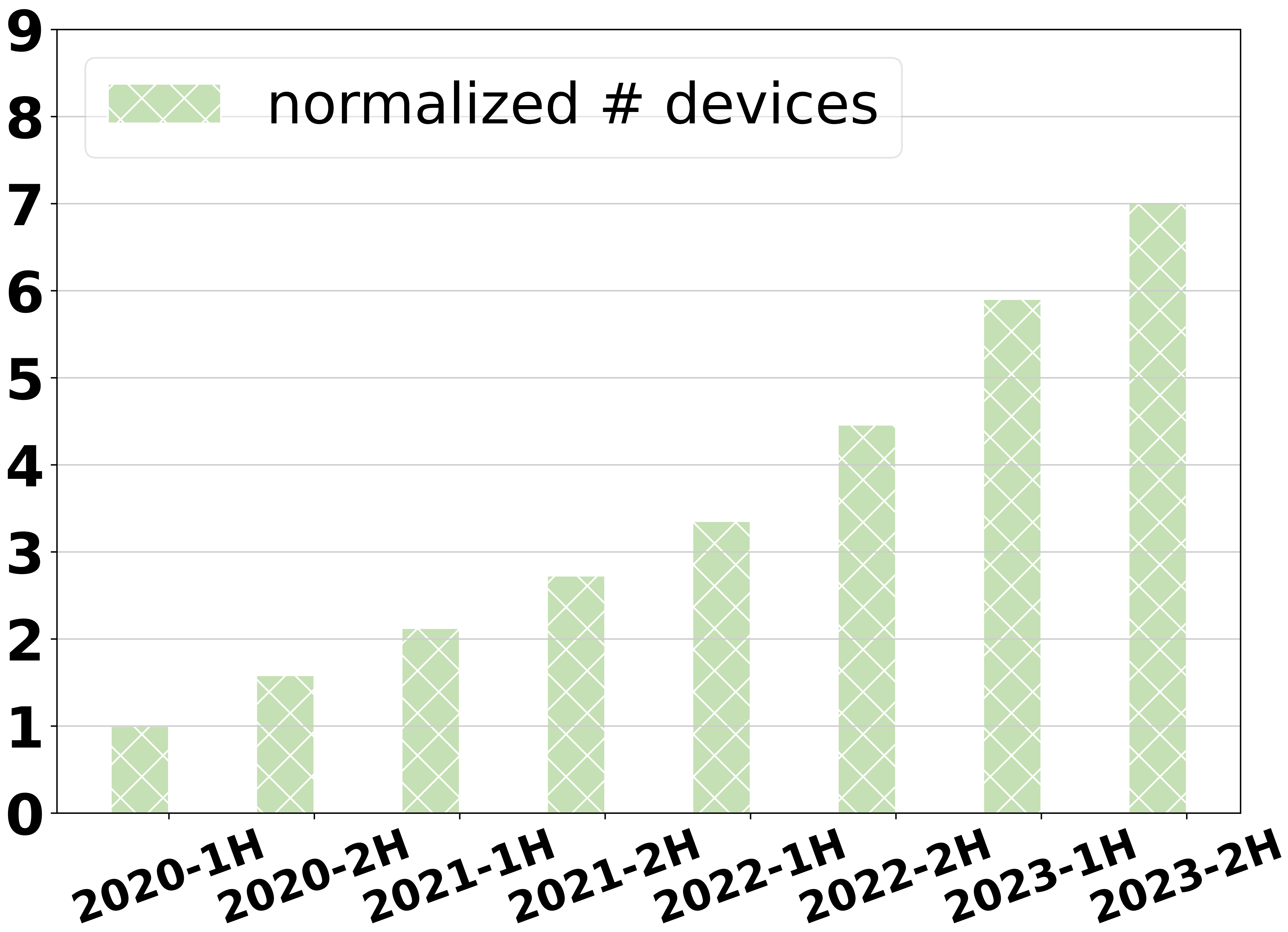
Routing Statistics

Statistics about relevant global routing table entries

ASes	Prefixes	Routes
110.1k	**1.3M**	**1.3M**
IPv4: 76,254 IPv6: 33,882	IPv4: 1,040,542 IPv6: 225,693	IPv4: 1,048,795 IPv6: 233,985



The network data plane is huge[1]



ByteDance's network scale increases steadily[2]

[1] https://radar.cloudflare.com/routing
[2] Gao, Zhaoyu, Anubhavnidhi Abhashkumar, Zhen Sun, Weirong Jiang, and Yi Wang. "Crescent: Emulating Heterogeneous Production Network at Scale." In 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), pp. 1045-1062. 2024.

Motivation

OOM > 32GB, TIMEOUT > 1h

State-of-Art	Summary	Cons	Result
Delta-net[NSDI17]	Incrementally maintain a compact representation using Atoms	$O(n^2)$ space complexity	OOM
APKeep[NSDI20]	Incrementally compute the minimum ECs	Redundant predicate merge and split operations	TIMEOUT
Flash[SIGCOMM22]	Handle update storms and long-tail update arrivals	Uneven subspace partitioning	TIMEOUT
Tulkun[SIGCOMM23]	Decompose DPV into distributed, on-device verification	Computation cost of DPVNet	TIMEOUT

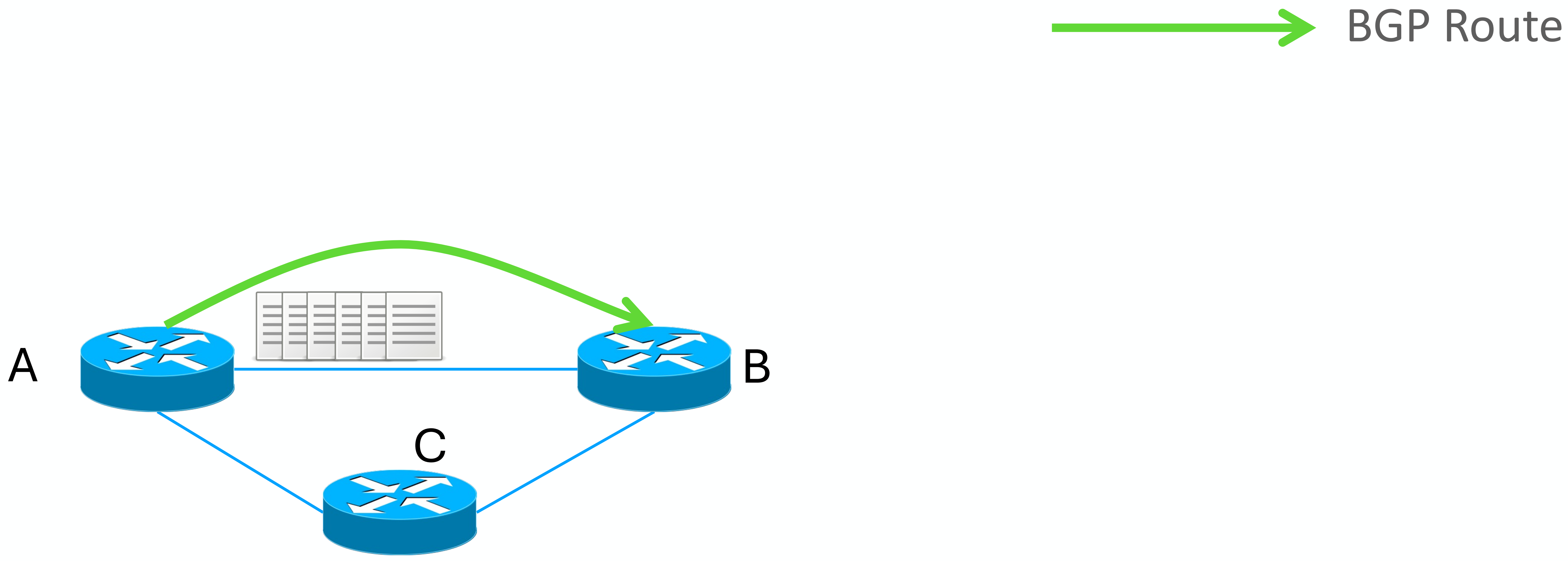
The existing methods fail to analyze large-scale networks

ByteDance字节跳动

Motivation

Why do we need high performance?

Case1: The controller does not catch routing changes in time leading to loops

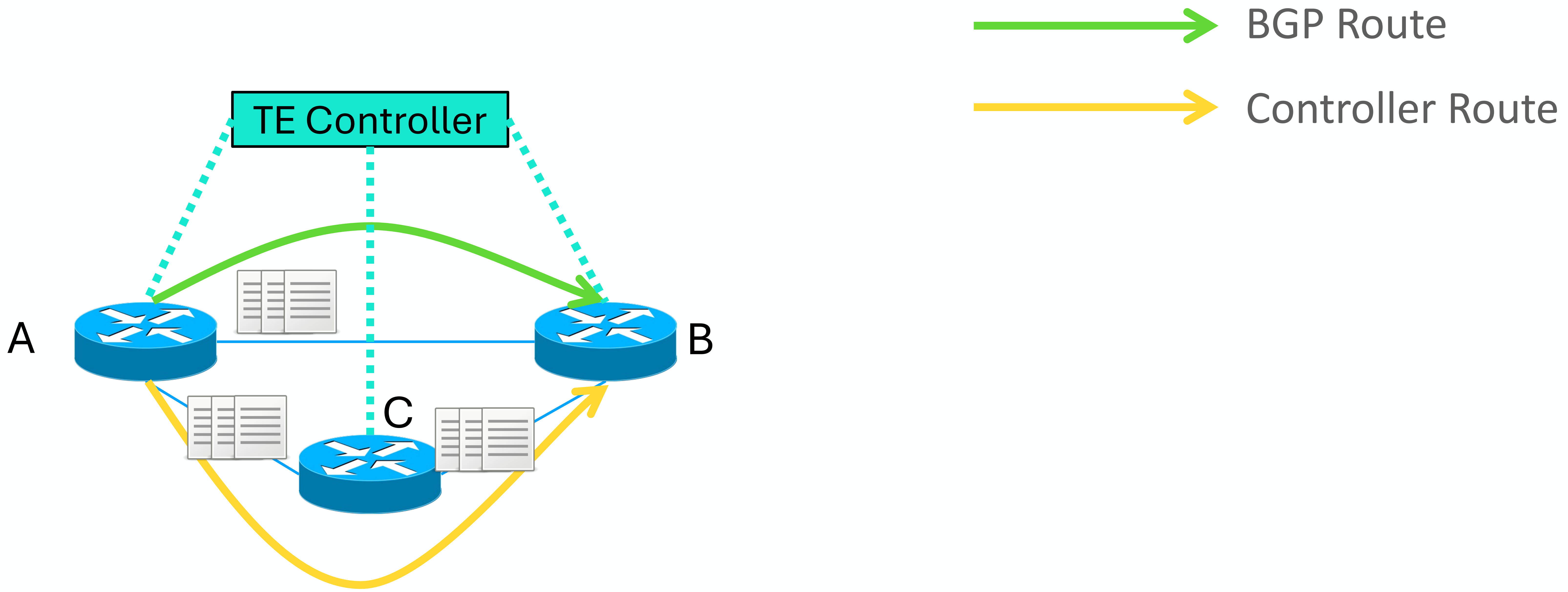


Normal scenario

Motivation

Why do we need high performance?

Case1: The controller does not catch routing changes in time leading to loops

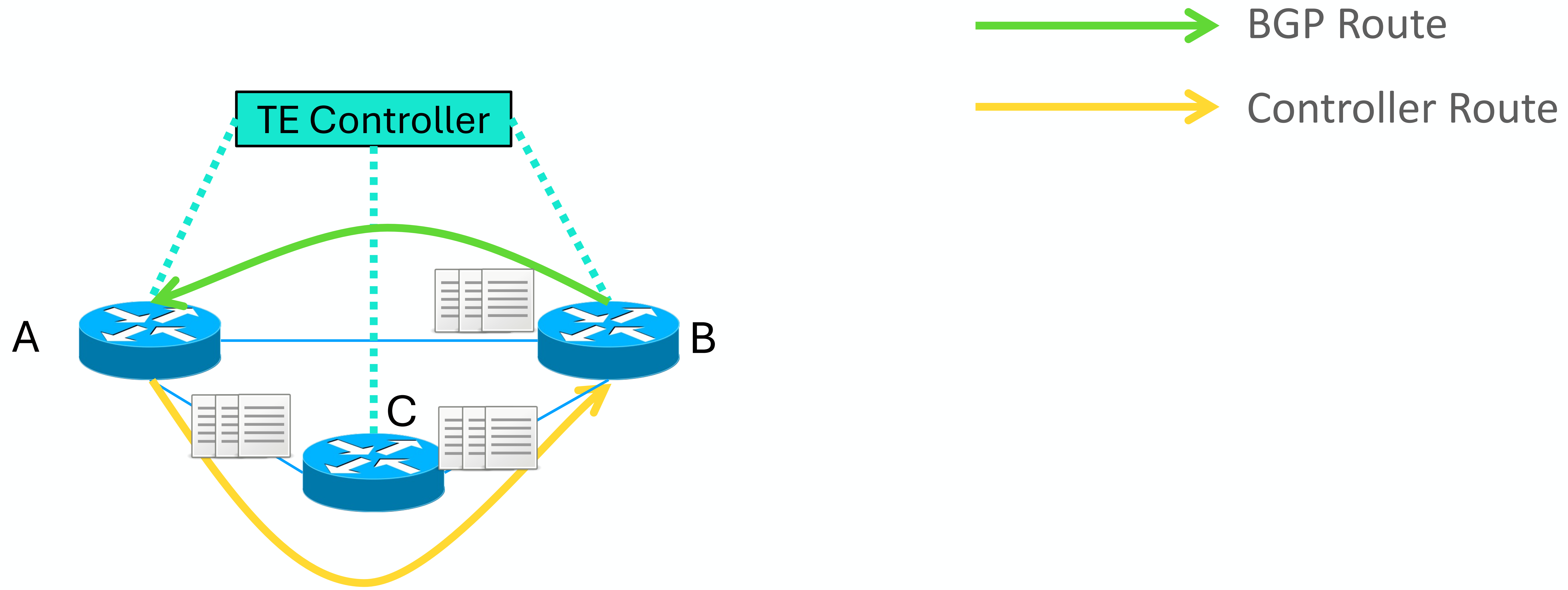


Normal scenario

Motivation

Why do we need high performance?

Case1: The controller does not catch routing changes in time leading to loops

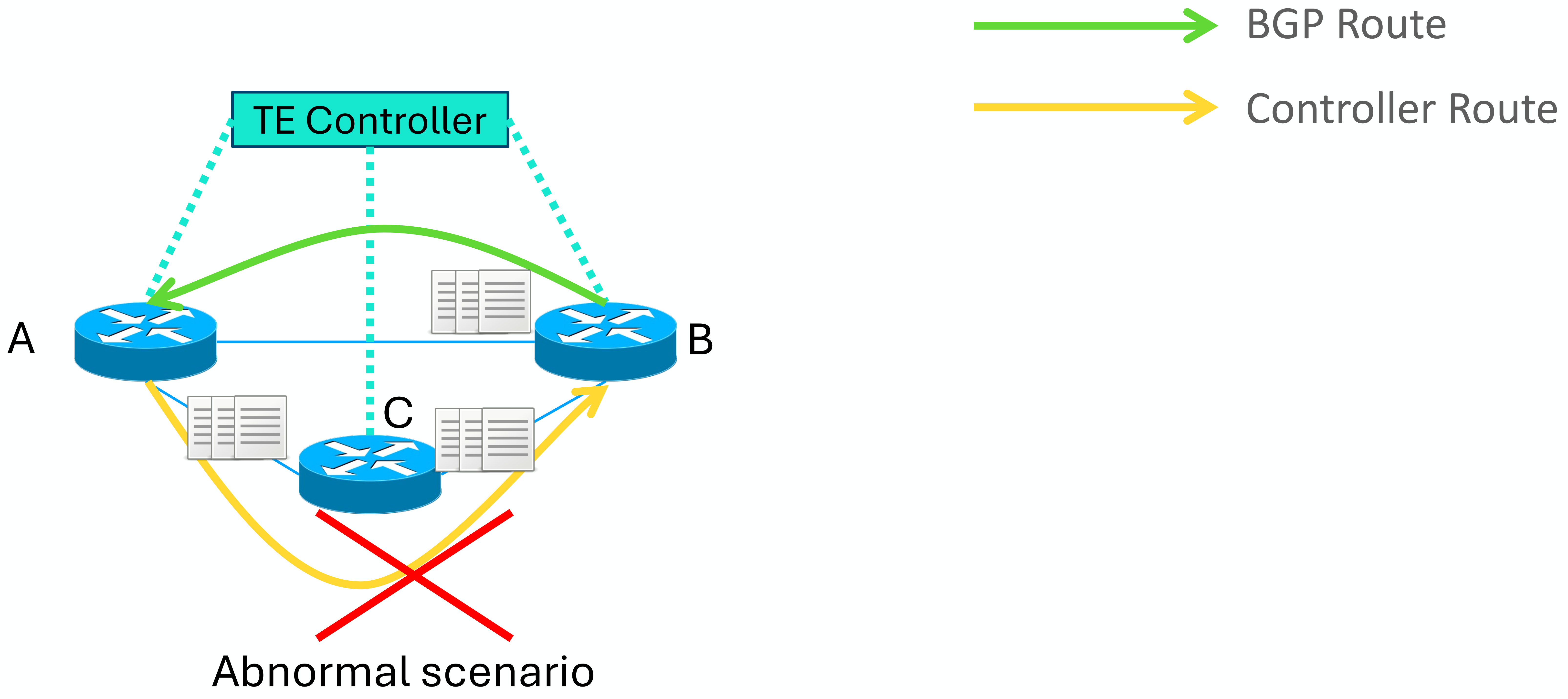


Abnormal scenario

Motivation

Why do we need high performance?

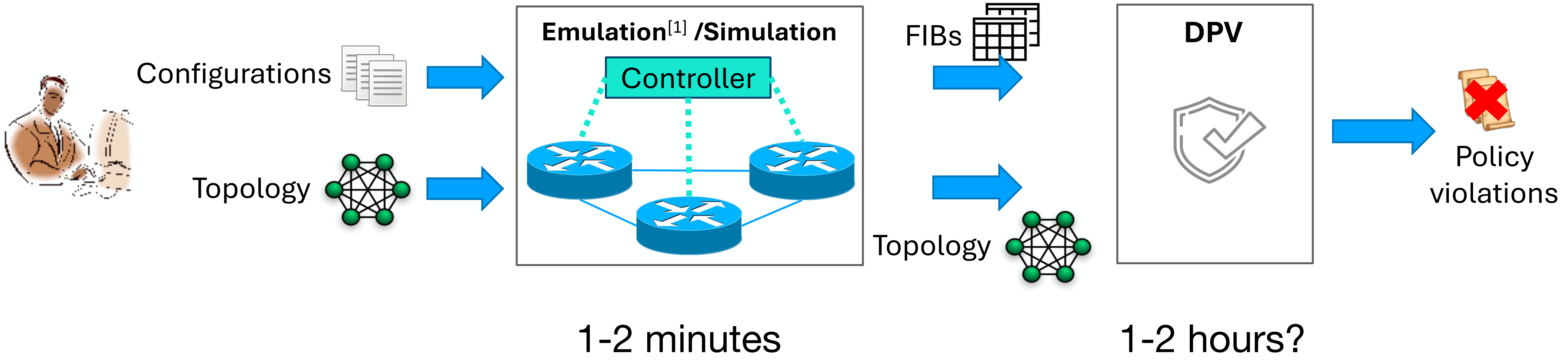
Case1: The controller does not catch routing changes in time leading to loops



Motivation

Why do we need high performance?

Case2: Fast analyze the impact of configuration changes

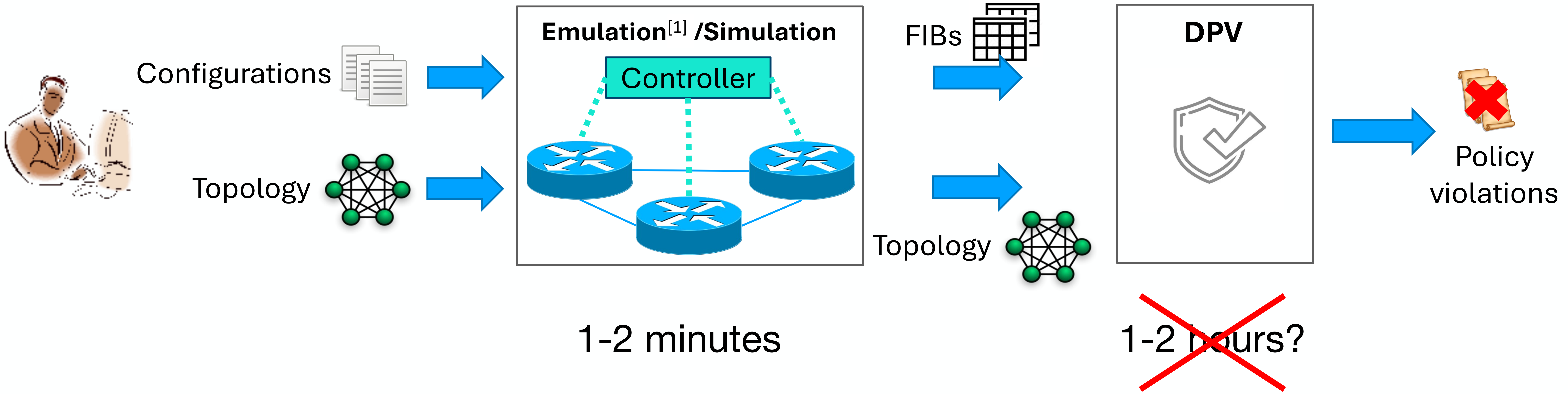


[1] Gao, Zhaoyu, Anubhavnidhi Abhashkumar, Zhen Sun, Weirong Jiang, and Yi Wang. "Crescent: Emulating Heterogeneous Production Network at Scale." In 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), pp. 1045-1062. 2024.

Motivation

Why do we need high performance?

Case2: Fast analyze the impact of configuration changes



[1] Gao, Zhaoyu, Anubhavnidhi Abhashkumar, Zhen Sun, Weirong Jiang, and Yi Wang. "Crescent: Emulating Heterogeneous Production Network at Scale." In 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), pp. 1045-1062. 2024.

Goals

Low Performance AND High Memory

High Performance AND Low Memory

Goals

Low Performance AND High Memory

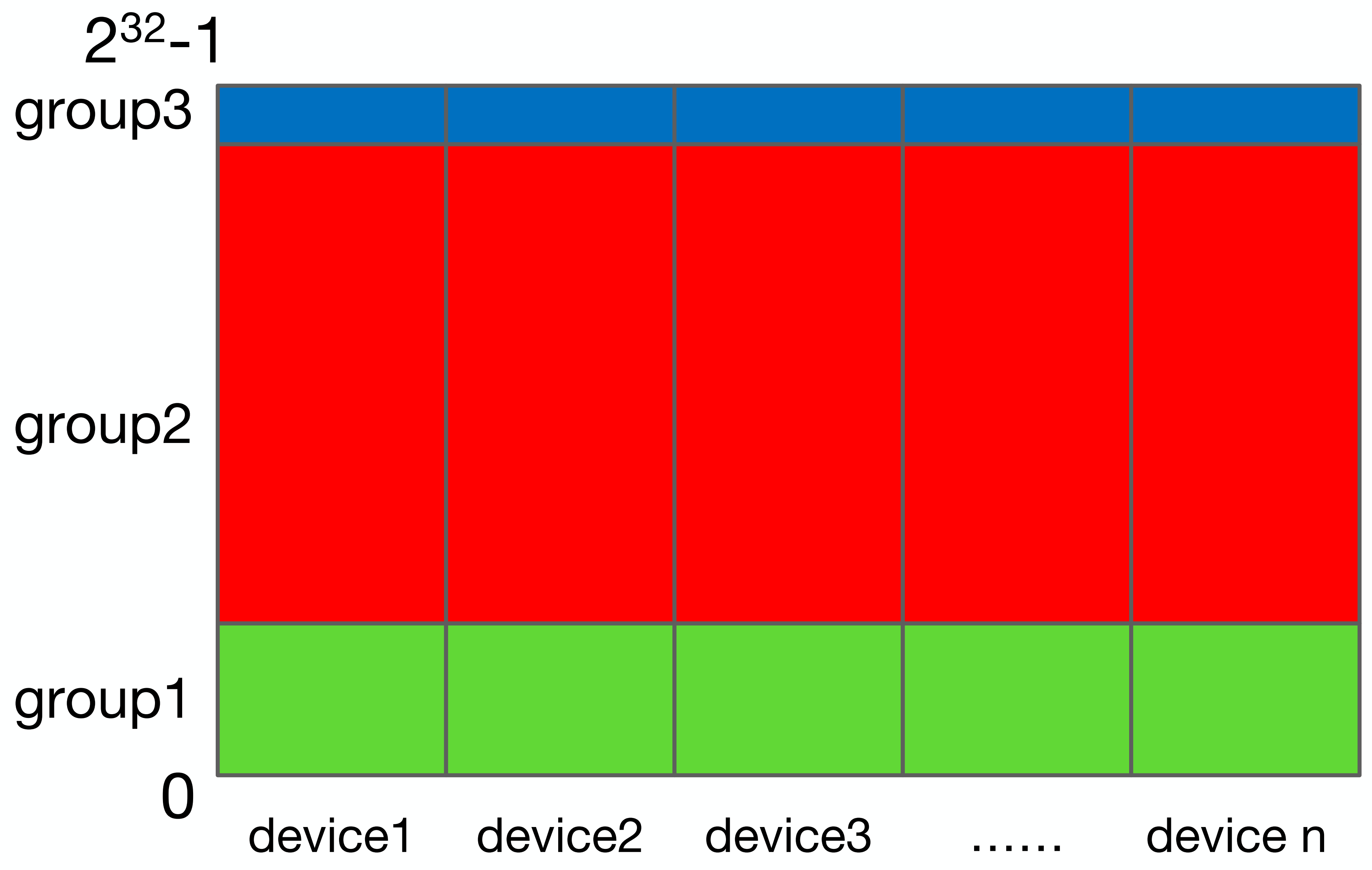
From Serial to Parallel

From EC to RANGESET

High Performance AND Low Memory

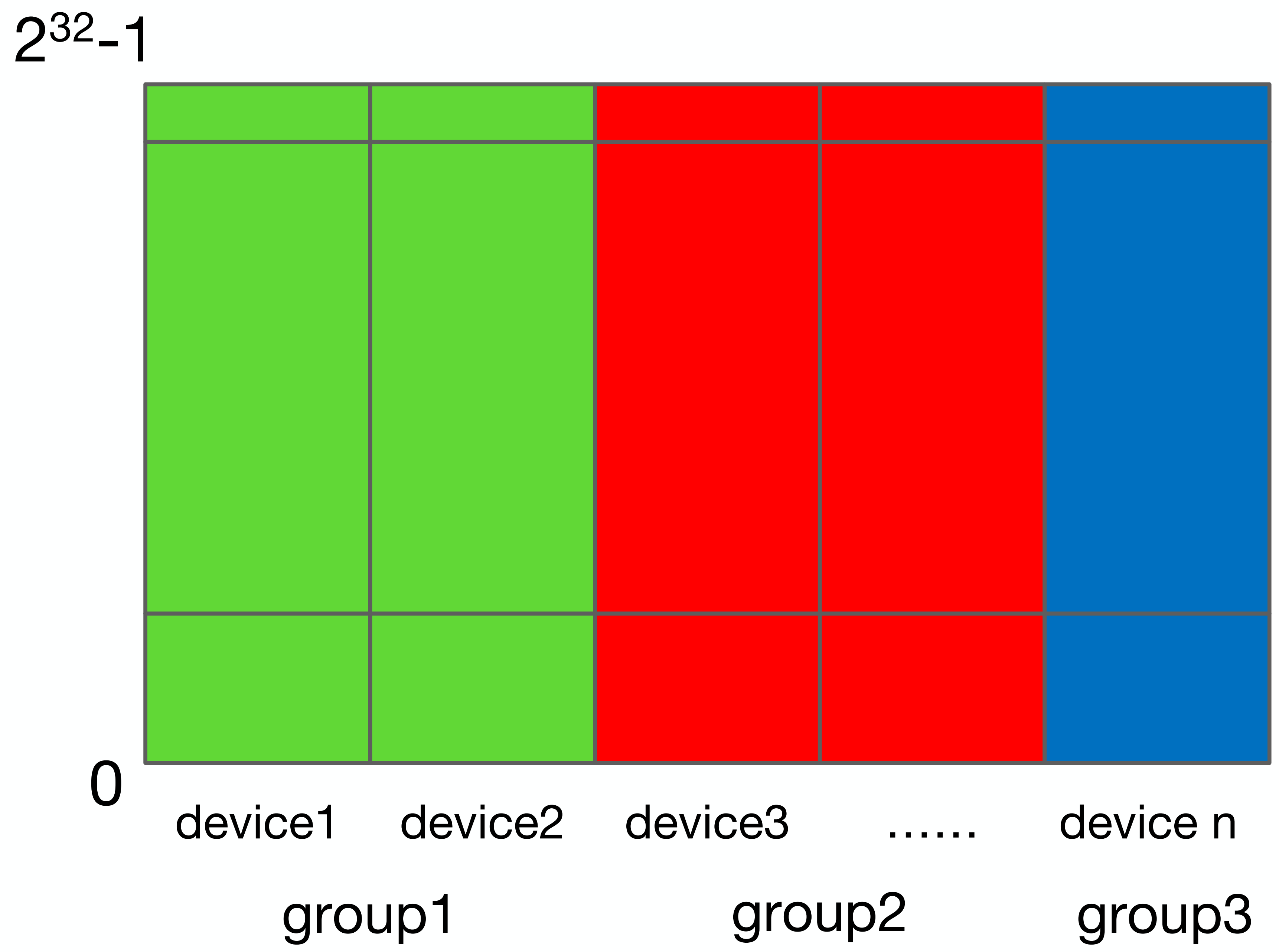
Challenges

How to divide the network model?



Divide IP space

- Uneven IP space partition
- Divide the same EC into different groups

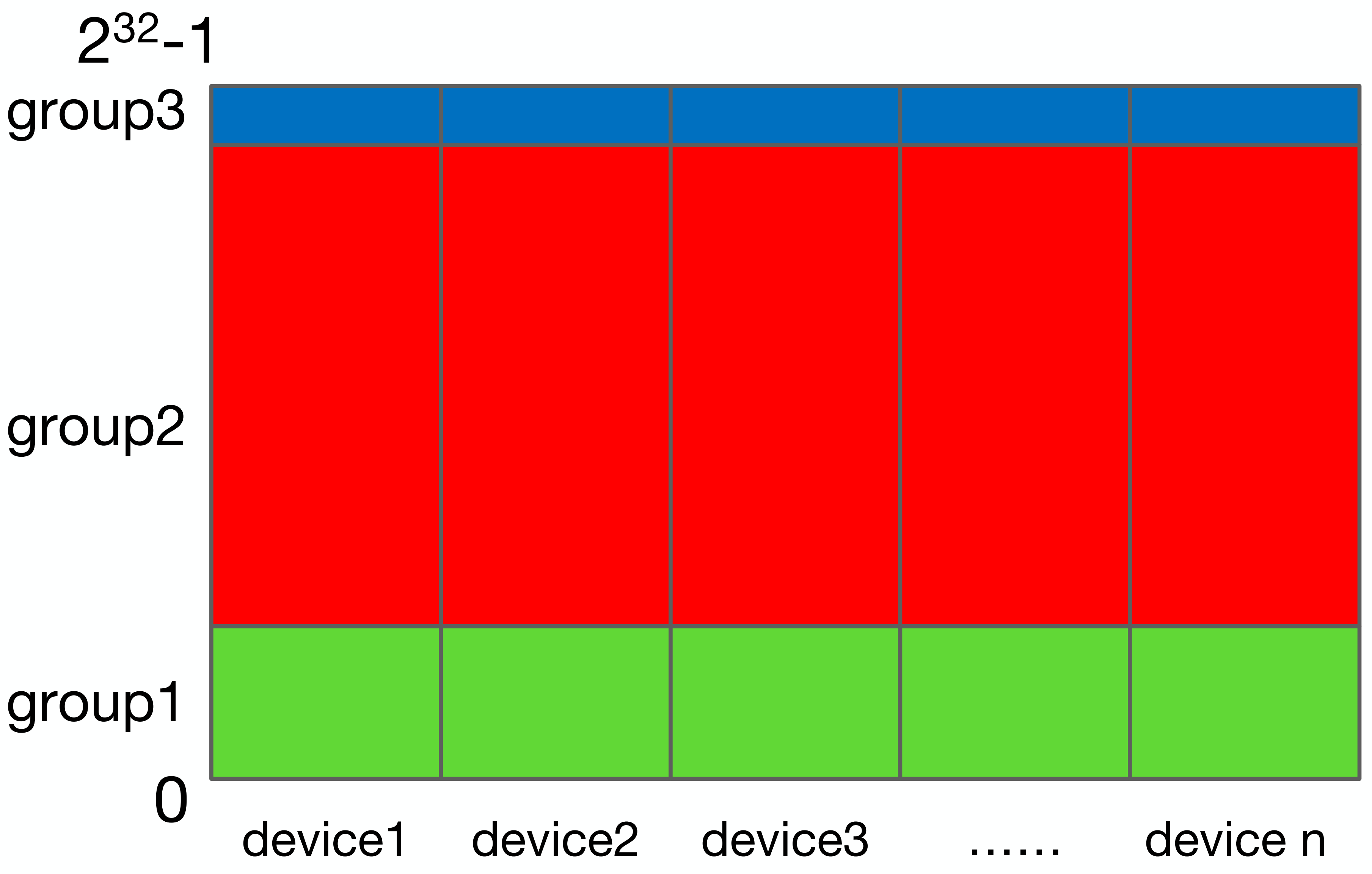


Divide devices

+ Fewer ECs due to fewer devices
+ Fewer memory due to fewer ECs

Challenges

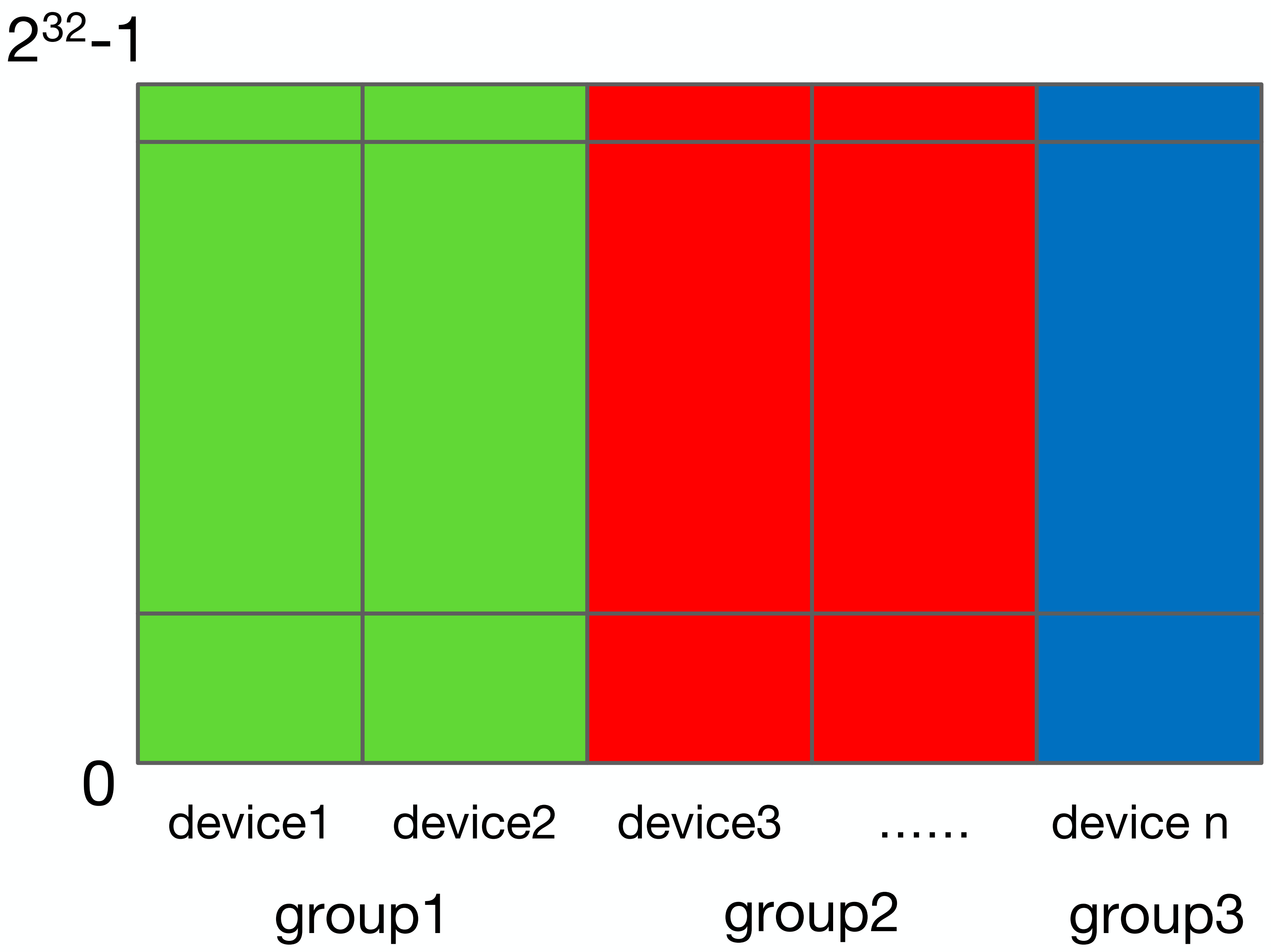
How to divide the network model?



Divide IP space

- Uneven IP space partition

- Divide the same EC into different groups



Divide devices ✓

+ Fewer ECs due to fewer devices

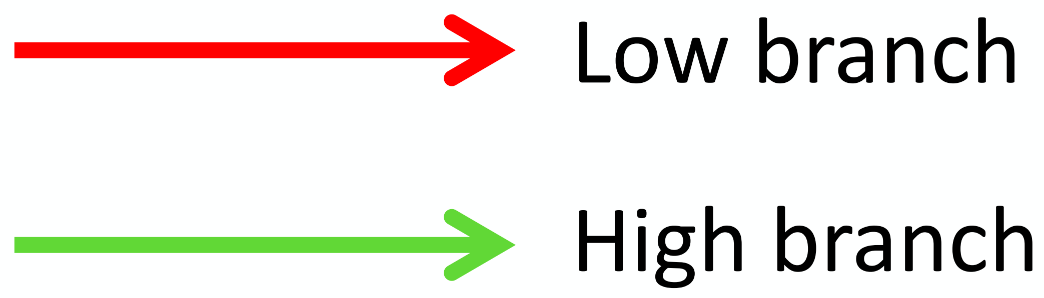
+ Fewer memory due to fewer ECs

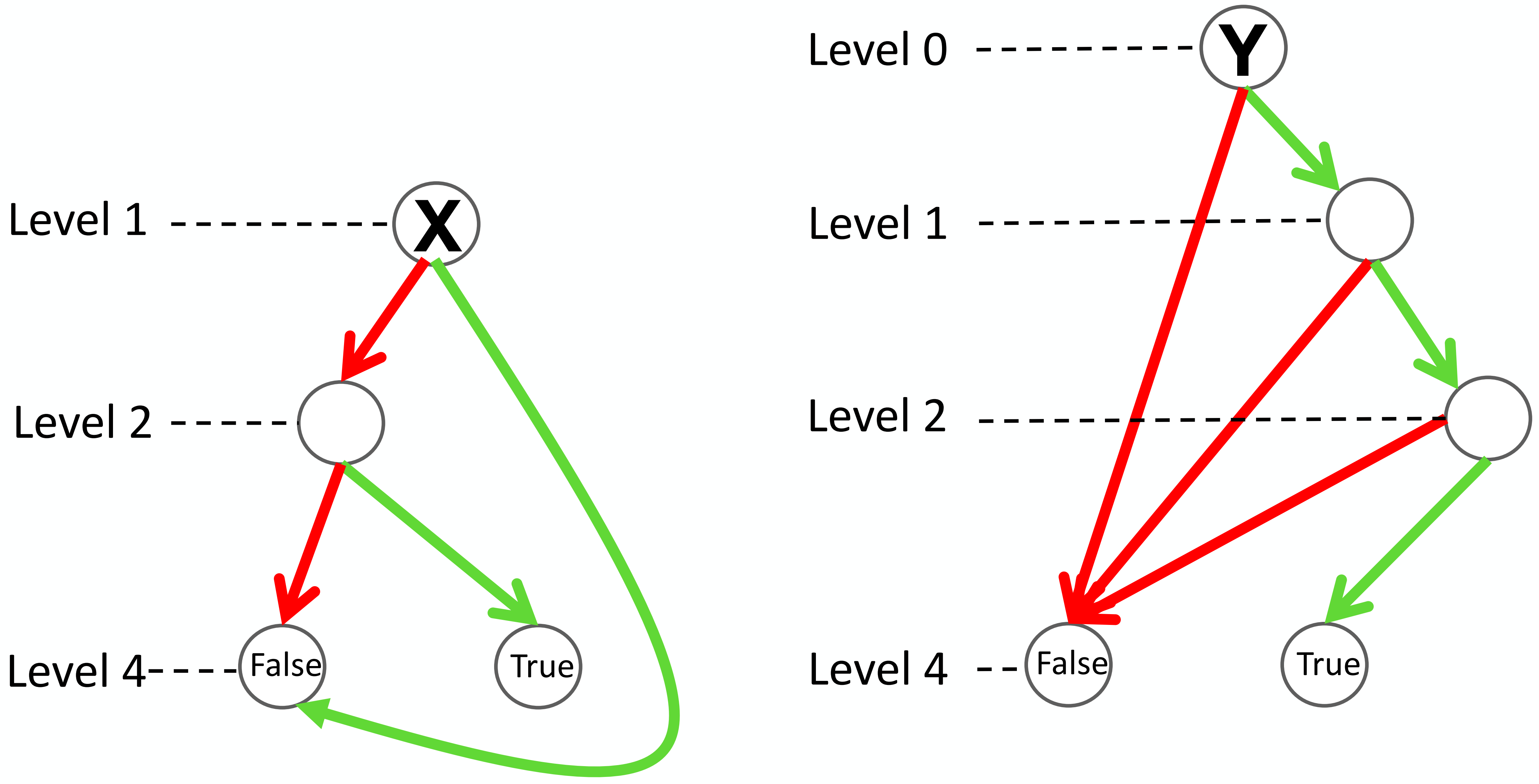
Challenges

What data structure to use for parallelism?

The primary data structure BDD

X = *01*   Y = 111*

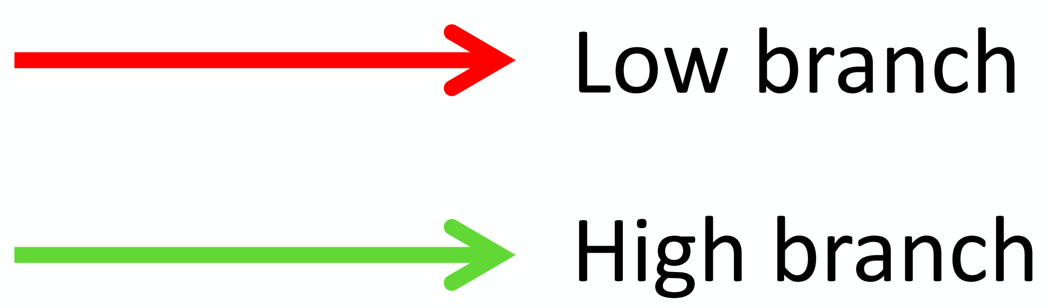




Challenges

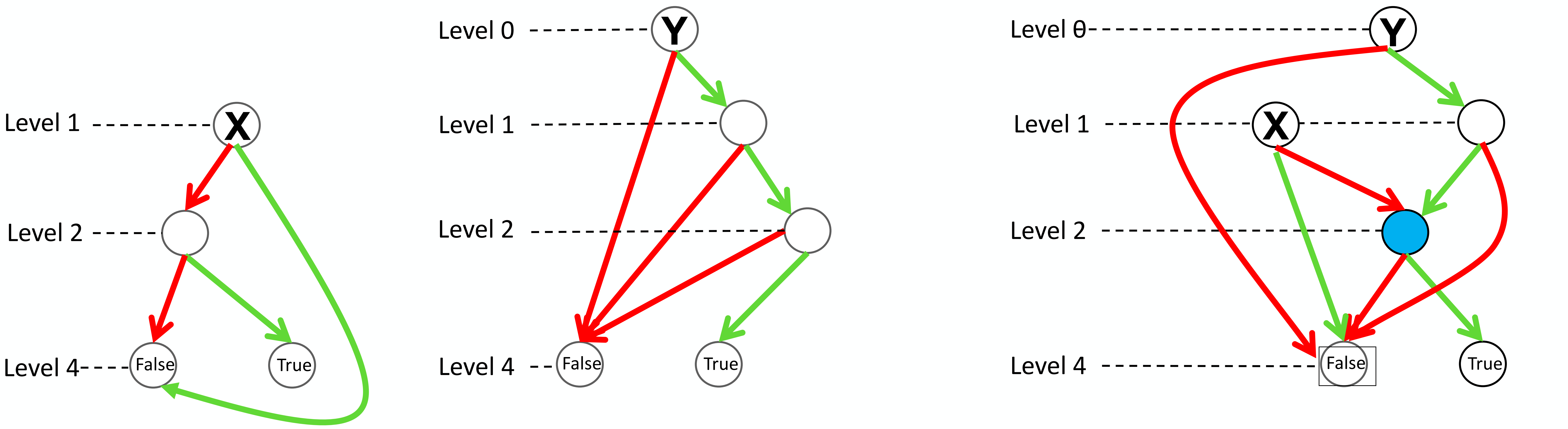
What data structure to use for parallelism?

The primary data structure BDD



X = *01*

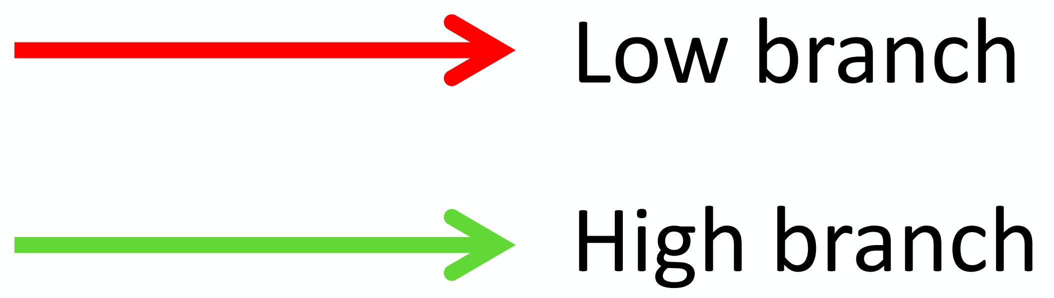
Y = 111*



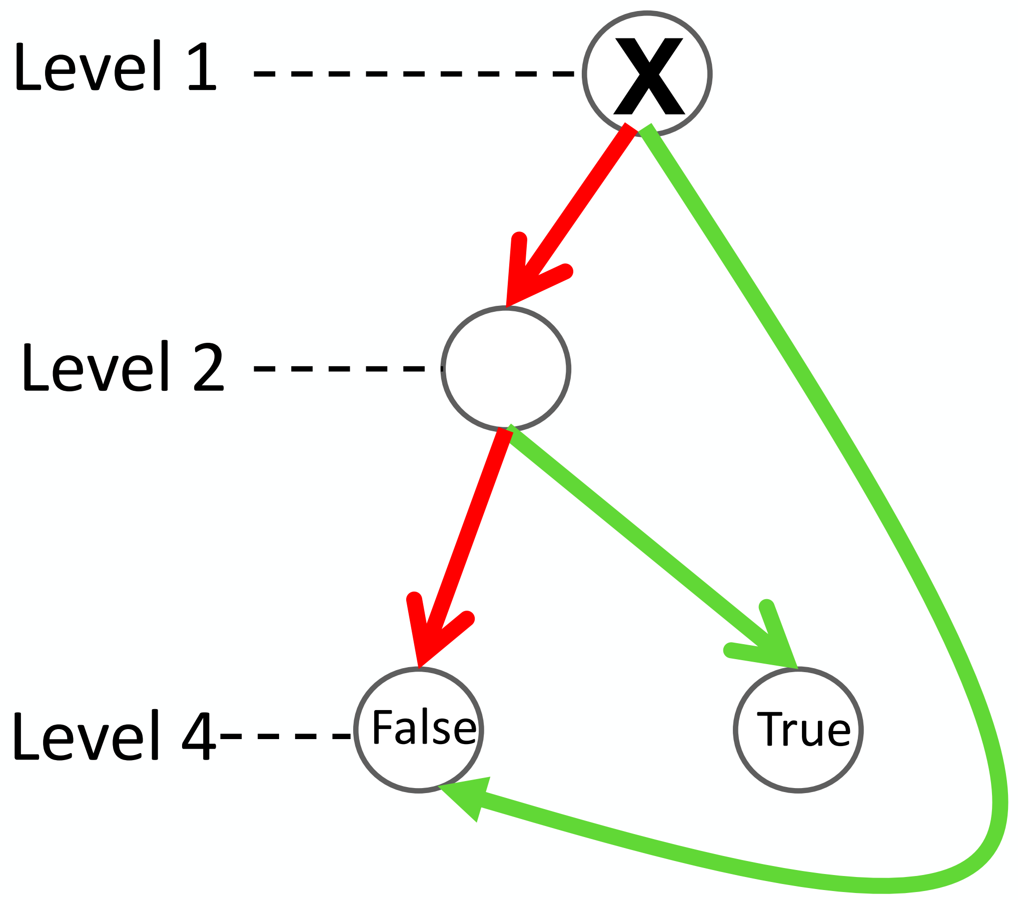
Challenges

What data structure to use for parallelism?

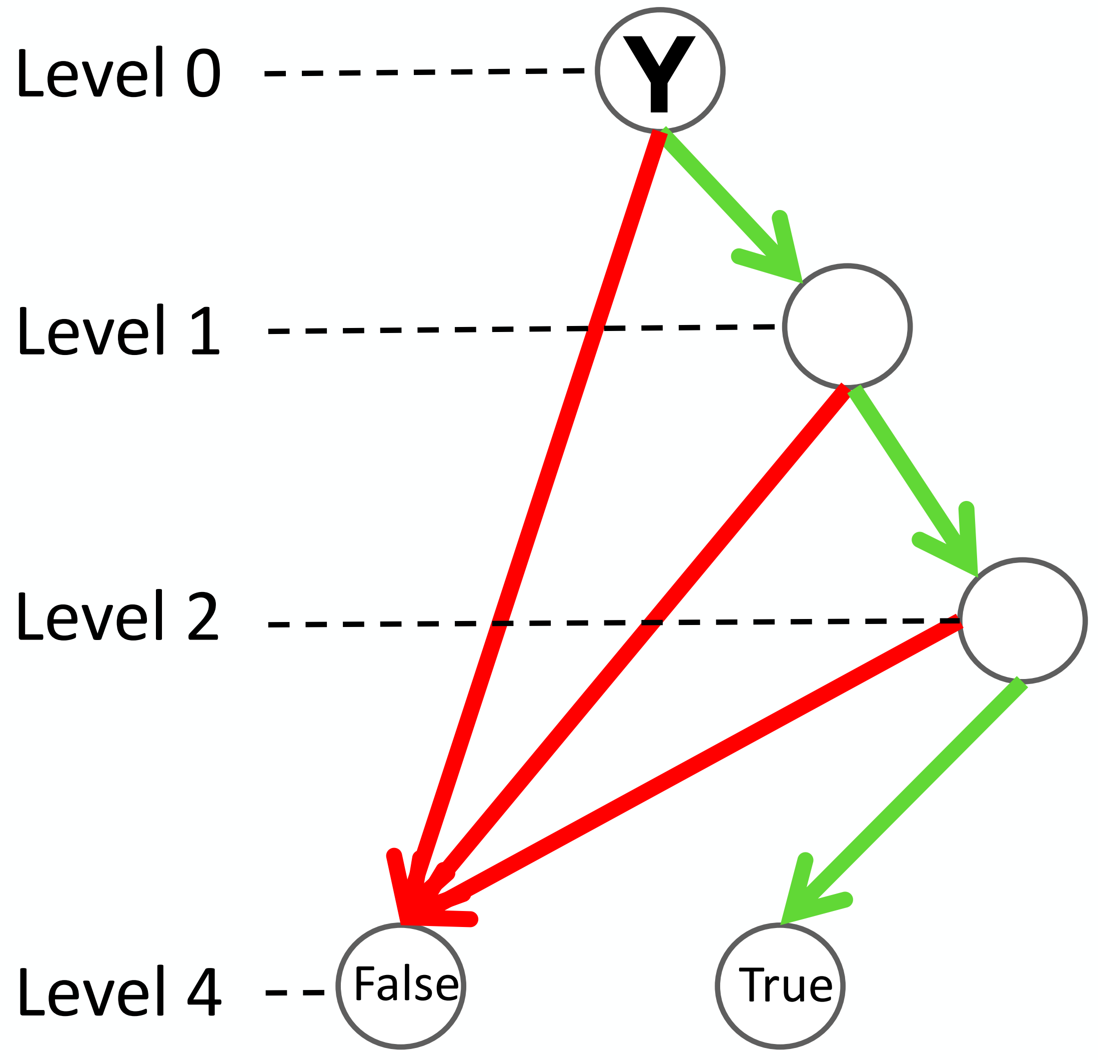
The primary data structure BDD

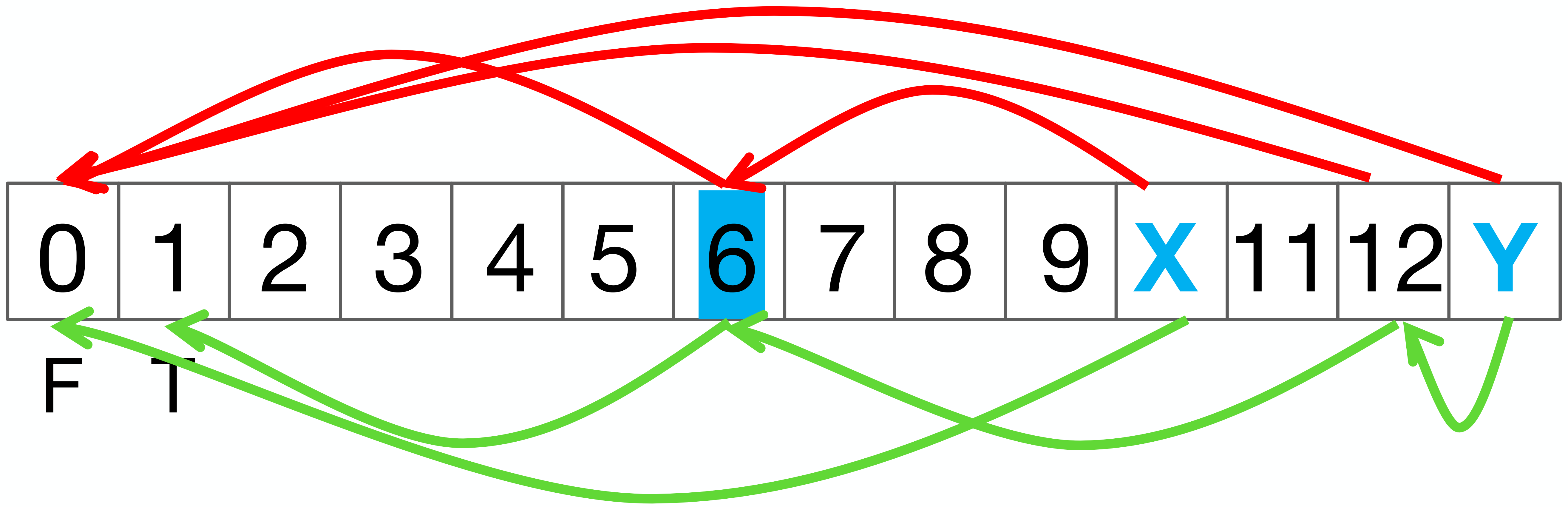


X = *01*



Y = 111*

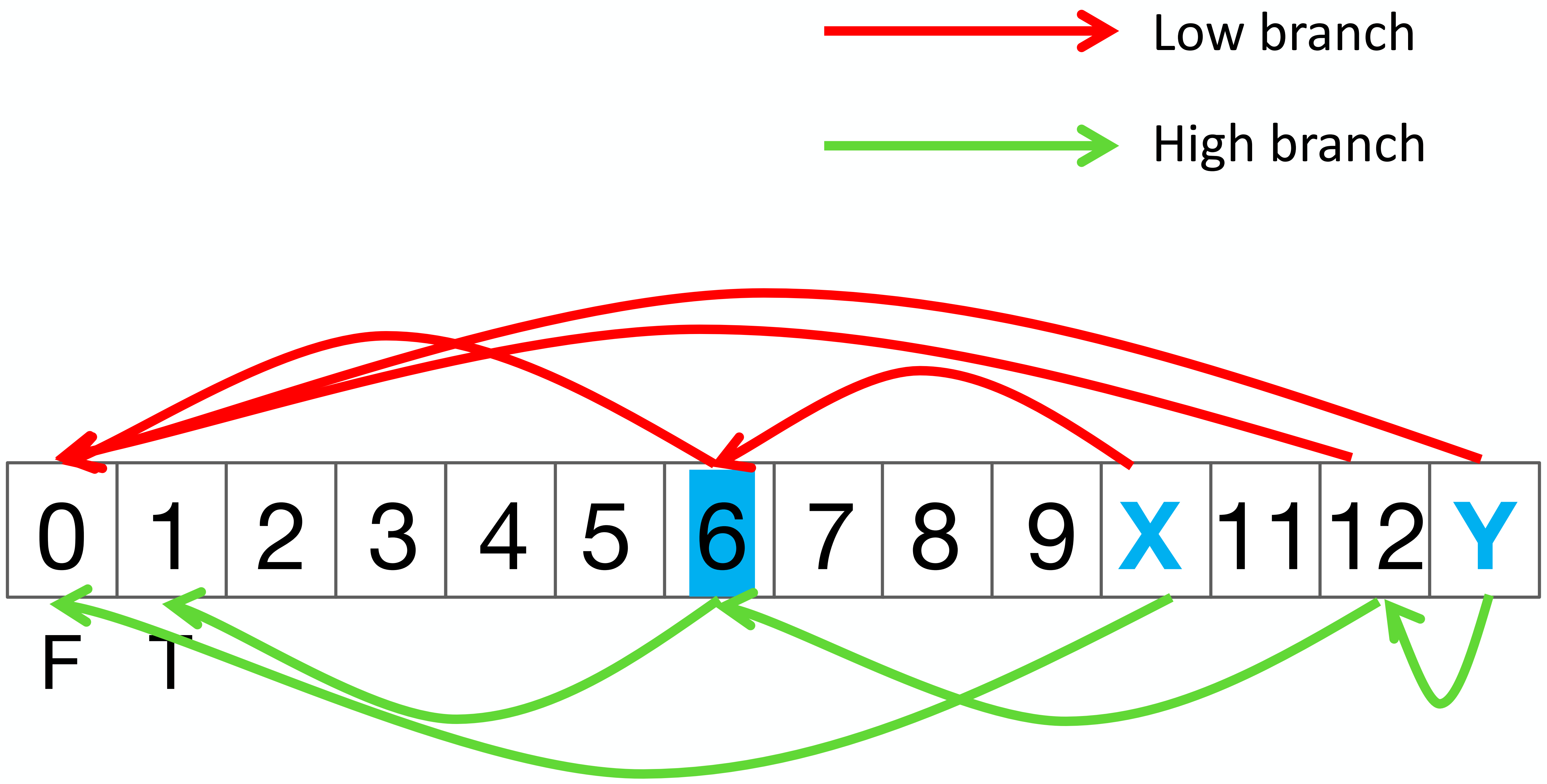




Challenges

What data structure to use for parallelism?

The primary data structure BDD



Hard to parallelize

- Dependencies
  - ➢ Different BDDs may share the same subgraph
- Dynamic nature
  - ➢ Resize the node table
- …

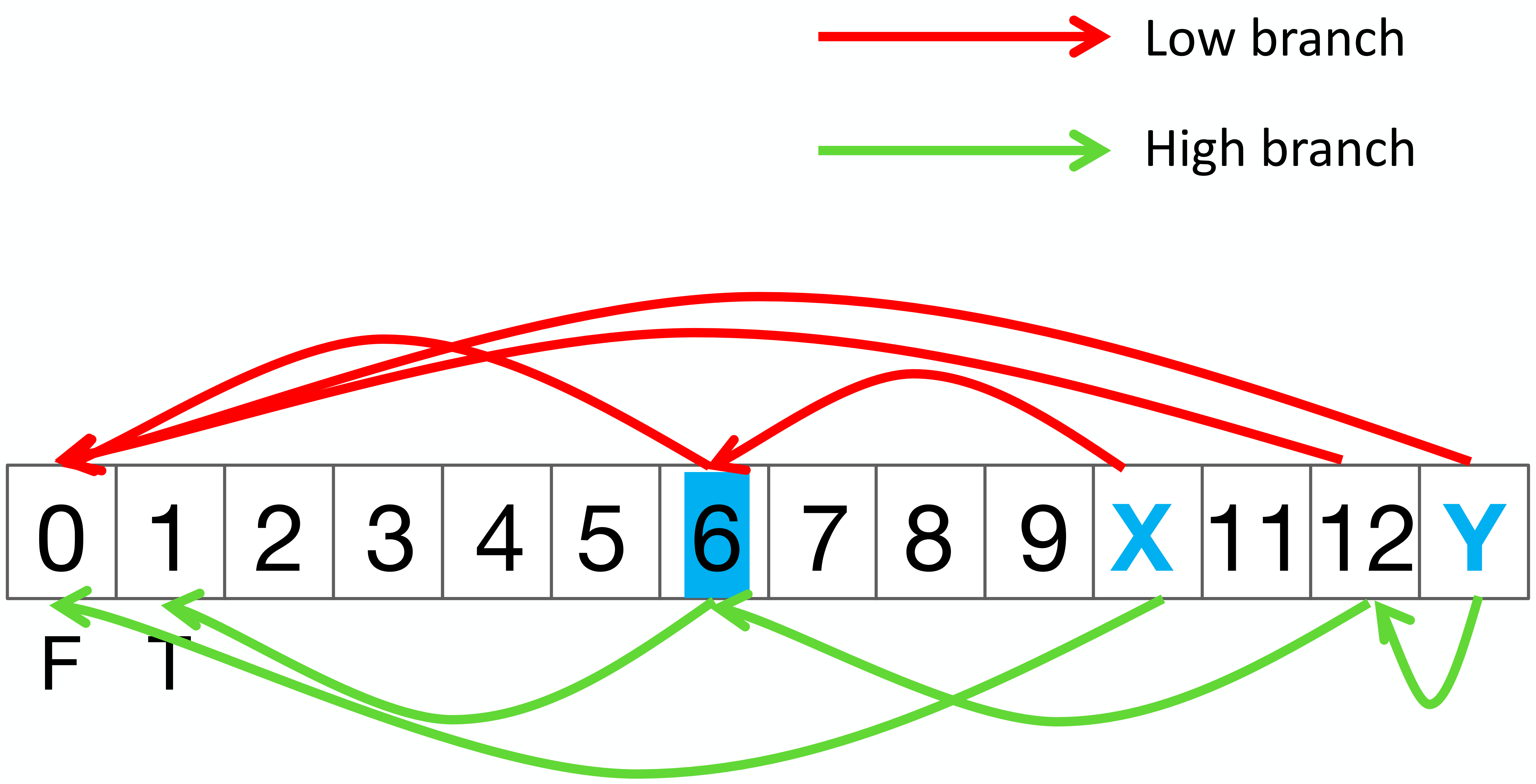
BDD operations are more expensive

- The size of a BDD can grow exponentially
- Garbage Collection
- …

Challenges

What data structure to use for parallelism?

The primary data structure BDD



Hard to parallelize

- Dependencies
  - Different BDDs may share the same subgraph
- Dynamic nature
  - Resize the node table
- …

BDD operations are more expensive

- The size of a BDD can grow exponentially
- Garbage Collection
- …

RANGESET ✓

ByteDance字节跳动

RANGESET

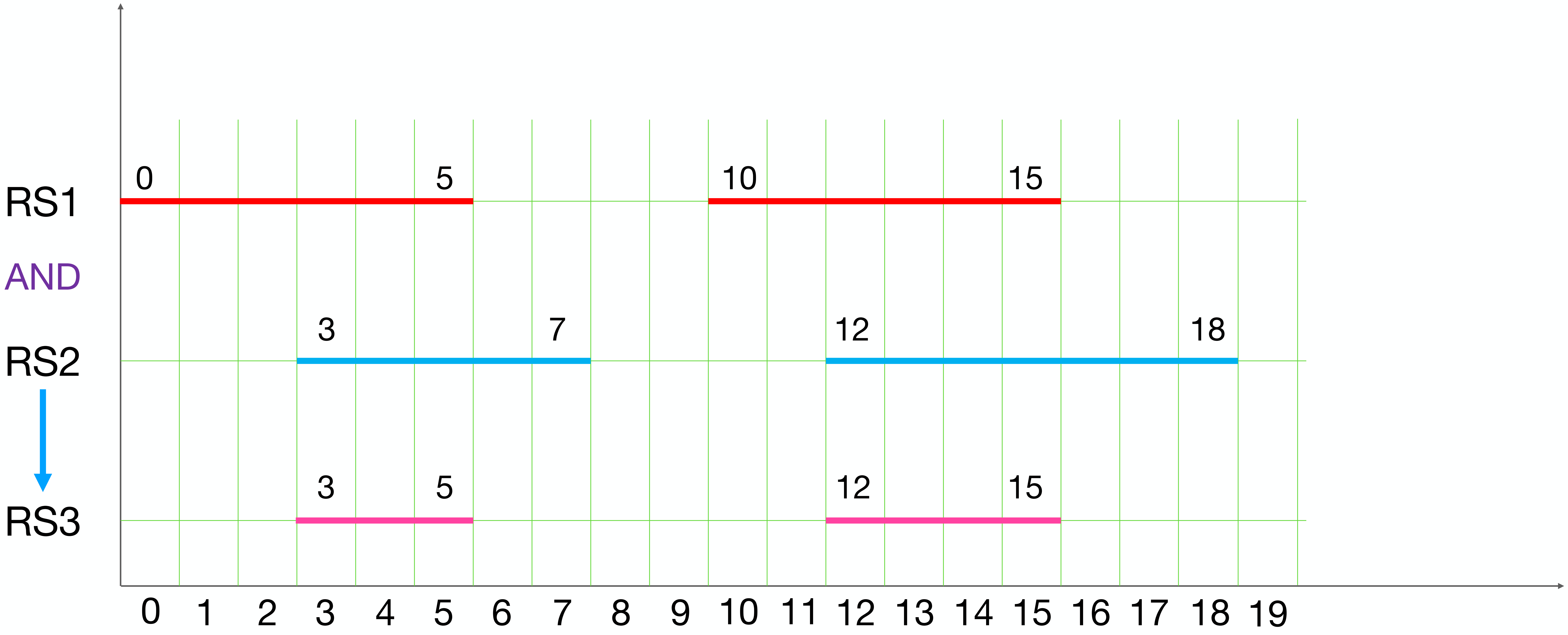
Definition

$$\boldsymbol{RANGESET} = \bigcup_{\boldsymbol{i}=1}^{\boldsymbol{n}} (\boldsymbol{LB_i}, \boldsymbol{UB_i})$$

where $LB_i \leq UB_i$ and $UB_i + 1 < LB_{i+1}$

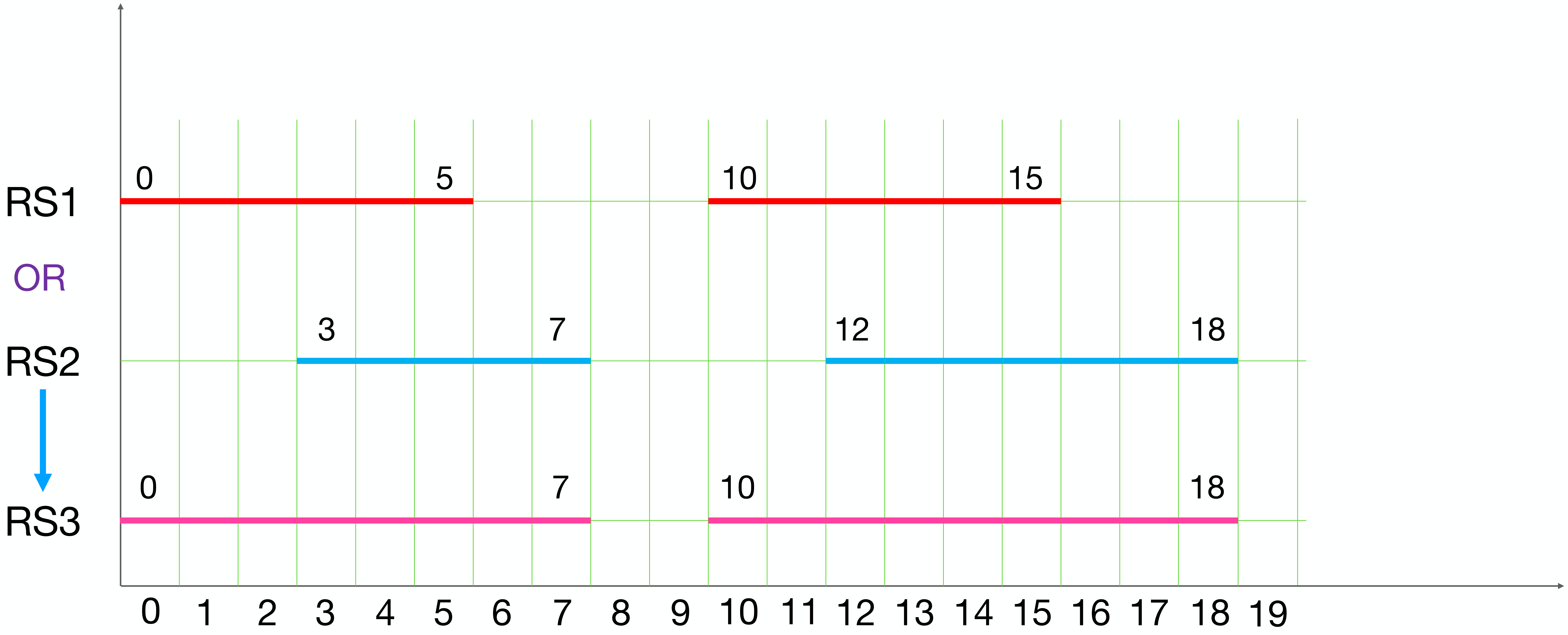
RANGESET

RS1 AND RS2 = RS3



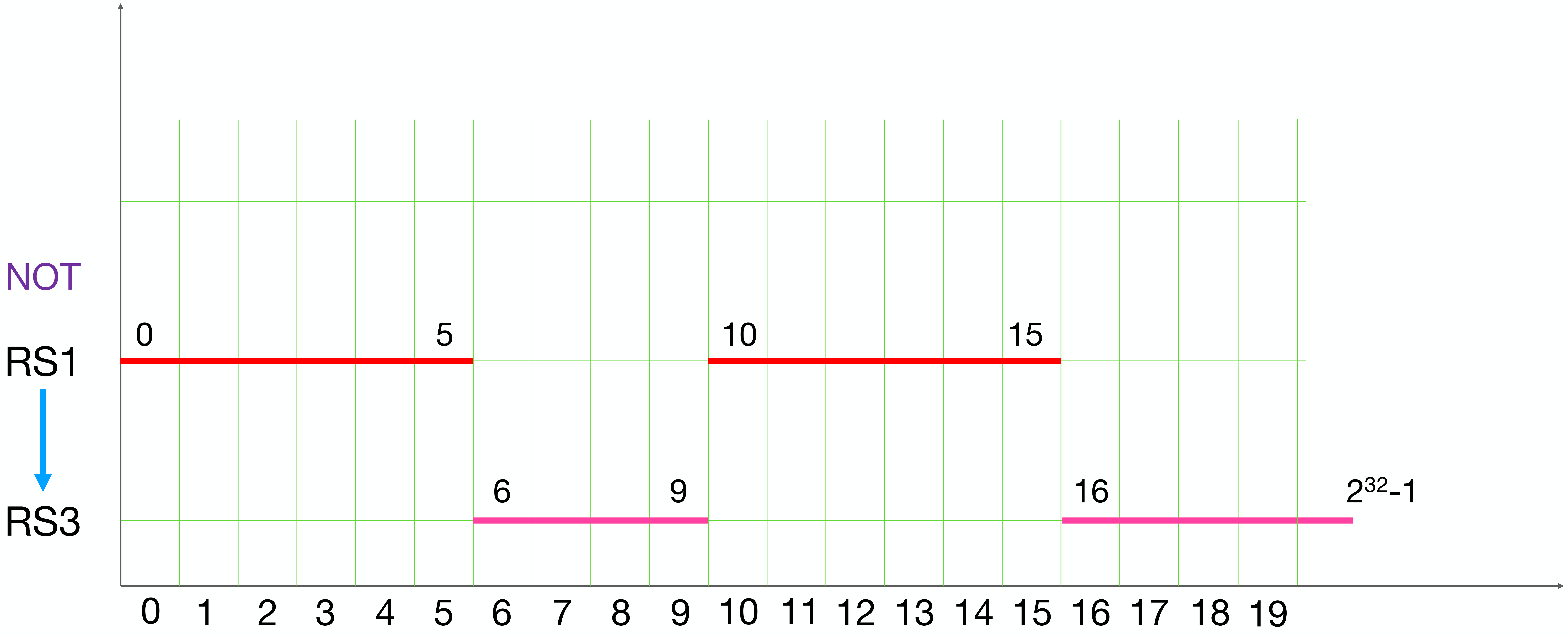
RANGESET

RS1 OR RS2 = RS3



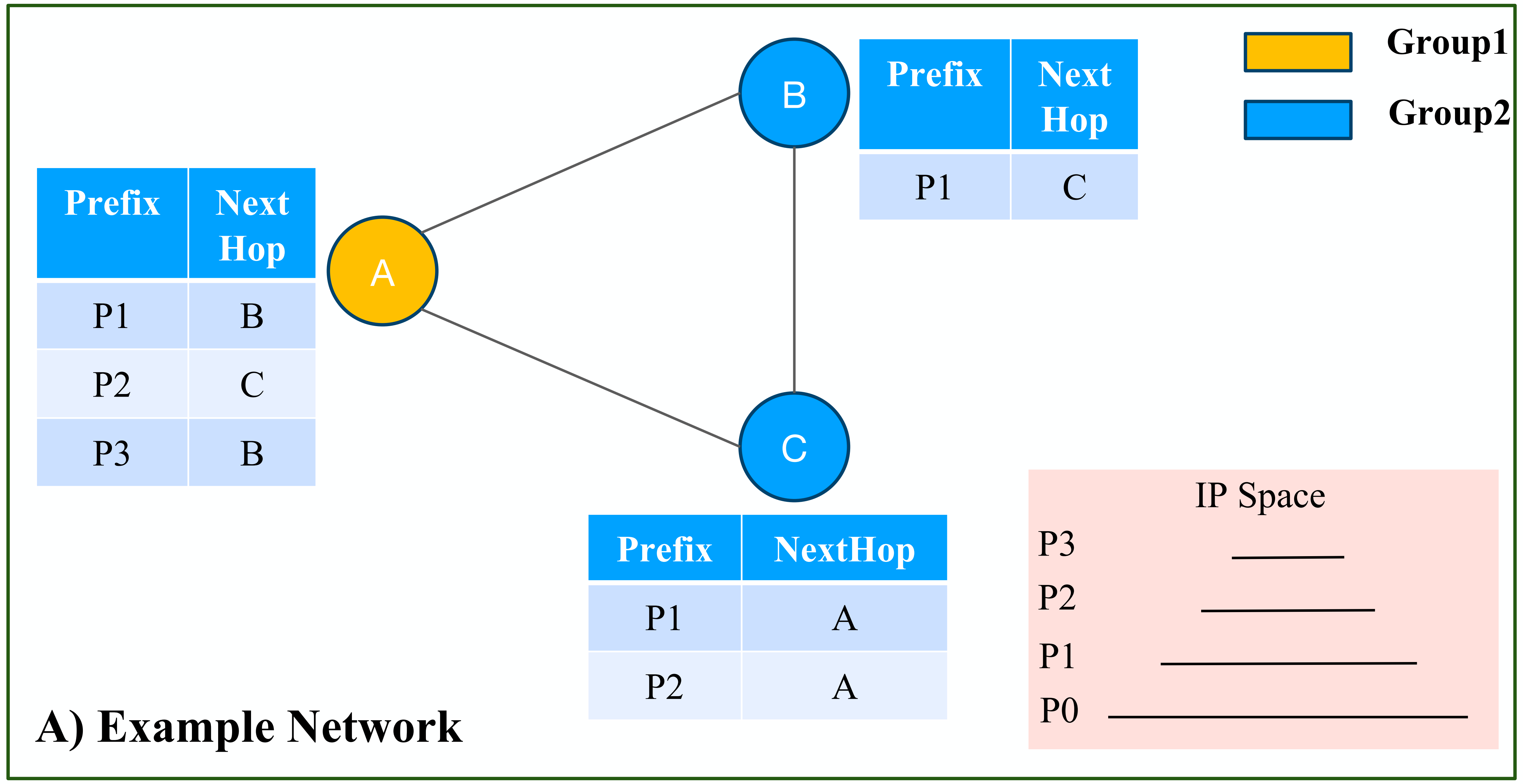
RANGESET

NOT RS1 = RS3



WORKFLOW

Example Network

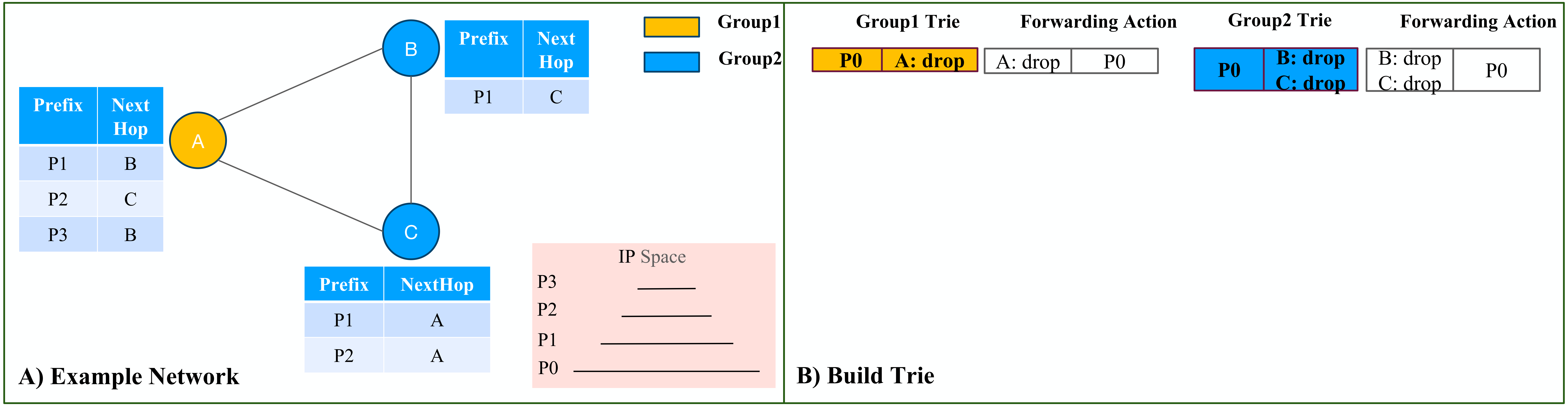


A) Example Network

ByteDance字节跳动

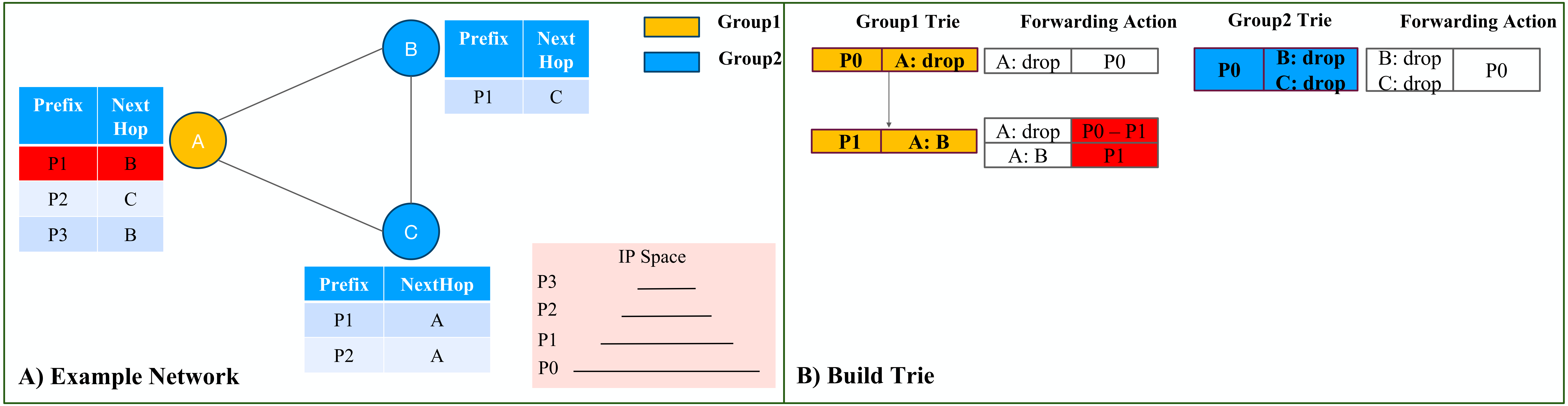
WORKFLOW

Example Network >> Build Trie



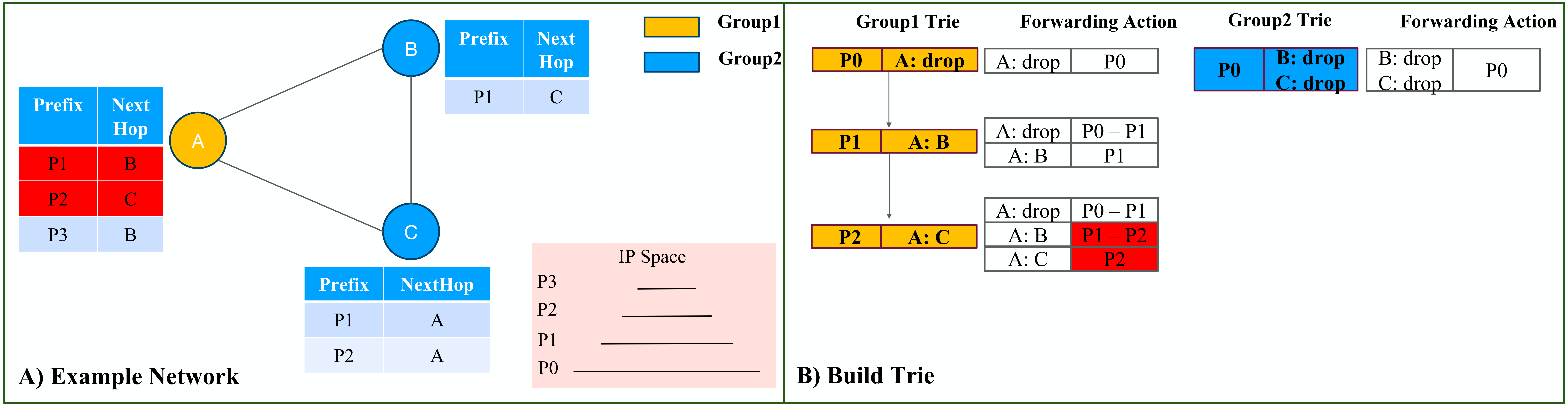
WORKFLOW

Example Network >> Build Trie



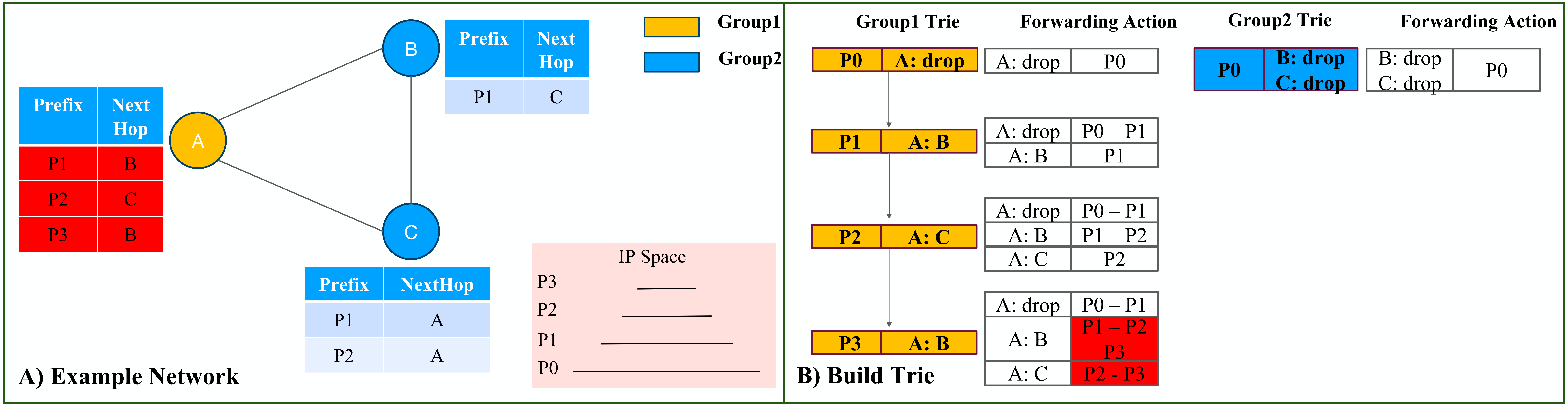
WORKFLOW

Example Network >> Build Trie



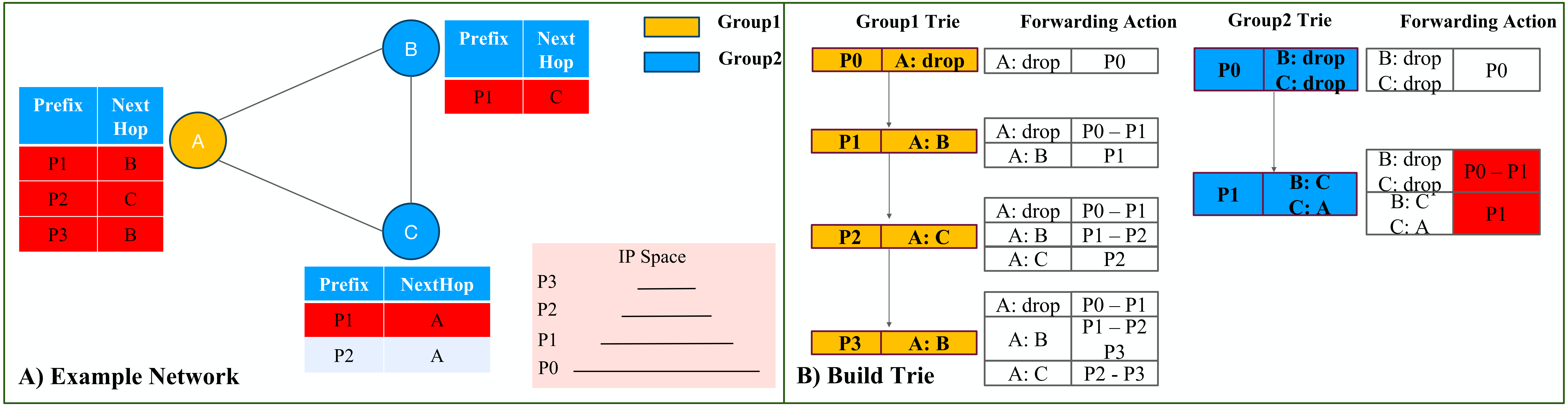
WORKFLOW

Example Network >> Build Trie



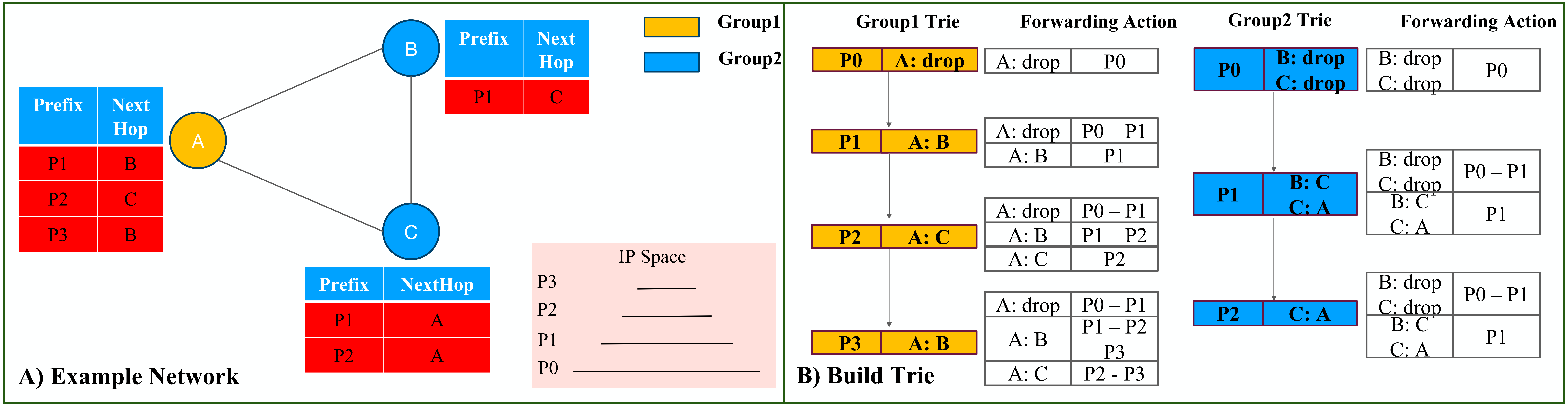
WORKFLOW

Example Network >> Build Trie



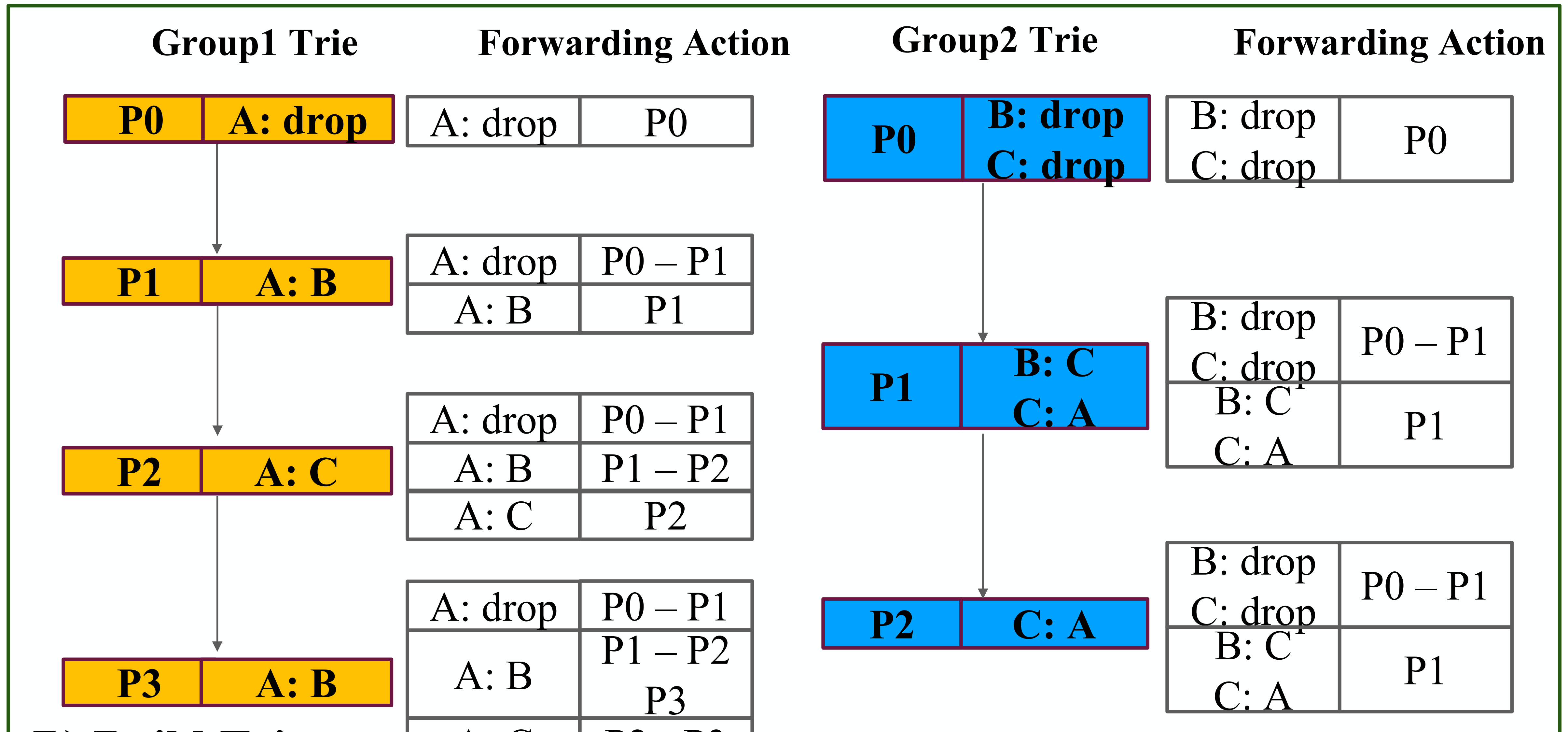
WORKFLOW

Example Network >> Build Trie



WORKFLOW

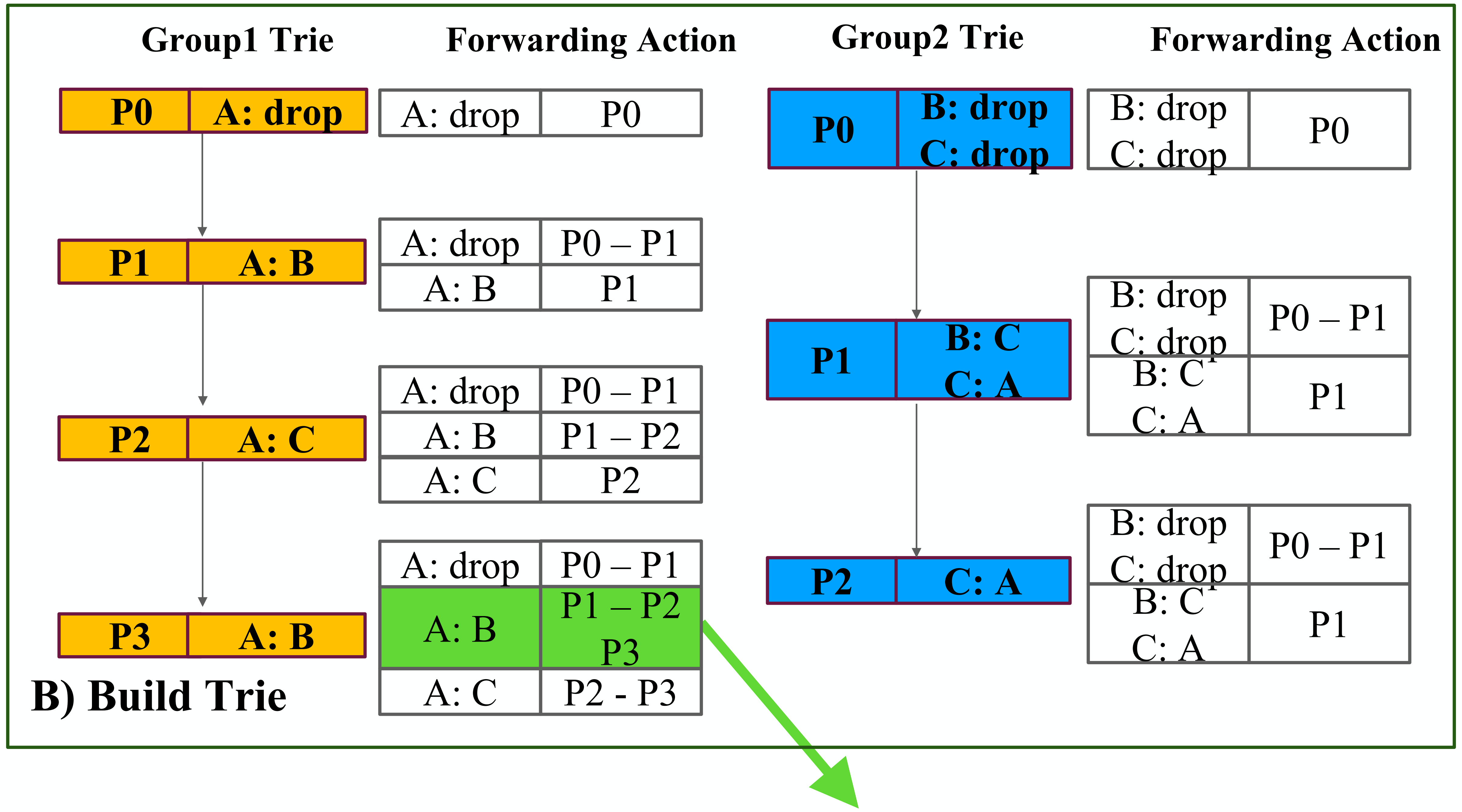
Example Network >> Build Trie >> Compute RANGESET

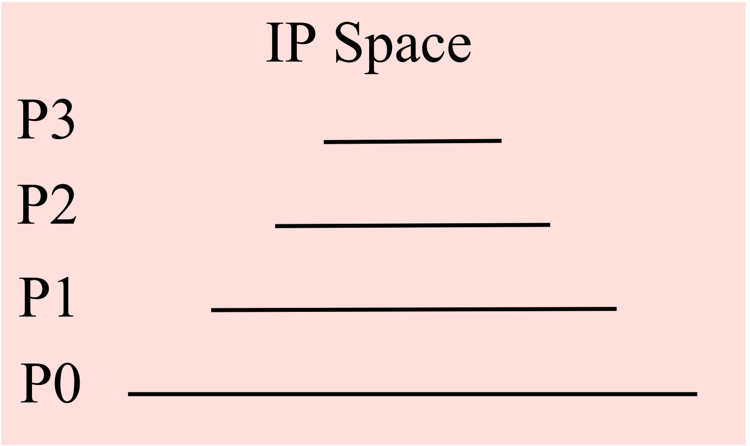


B) Build Trie

WORKFLOW

Example Network >> Build Trie >> Compute RANGESET

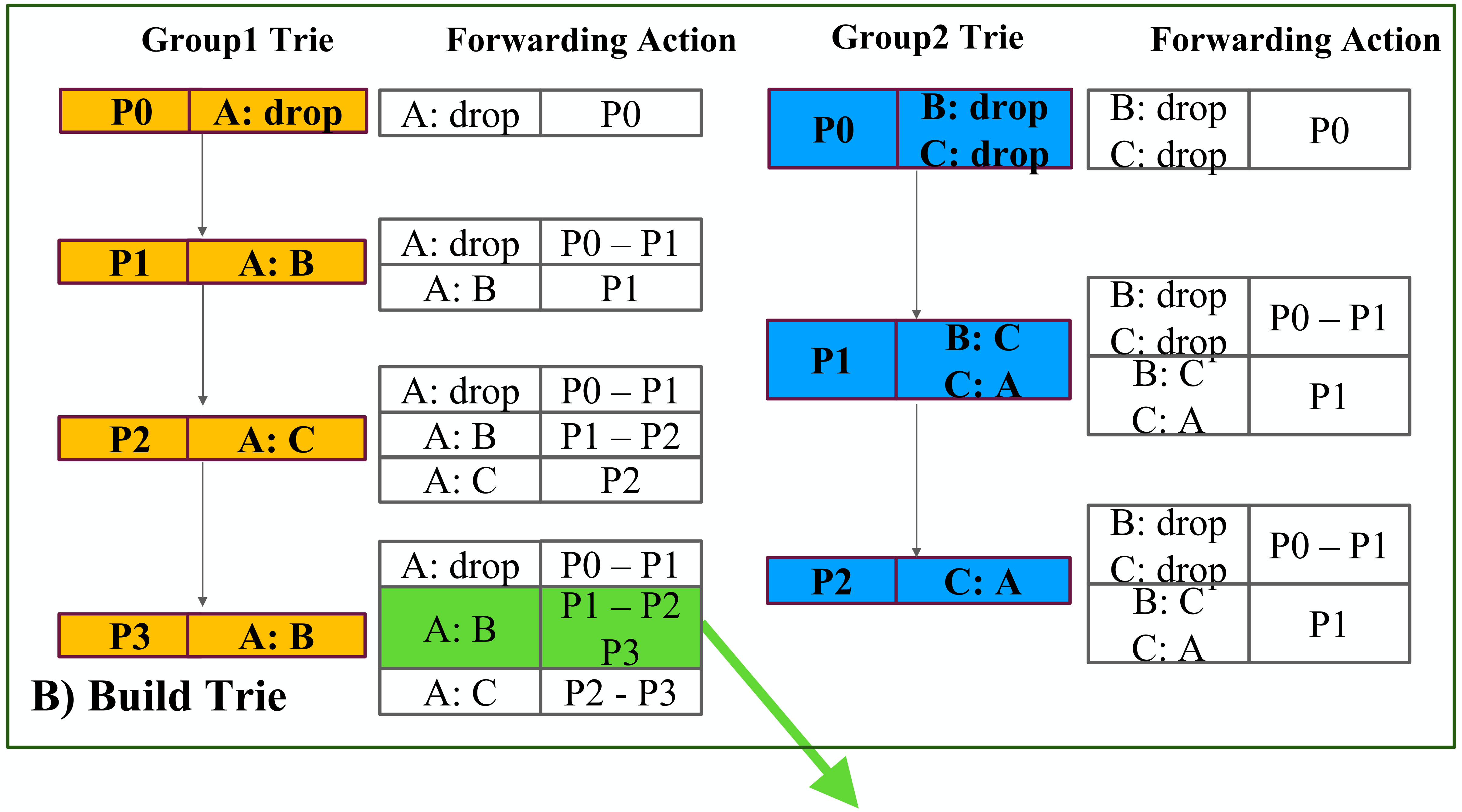




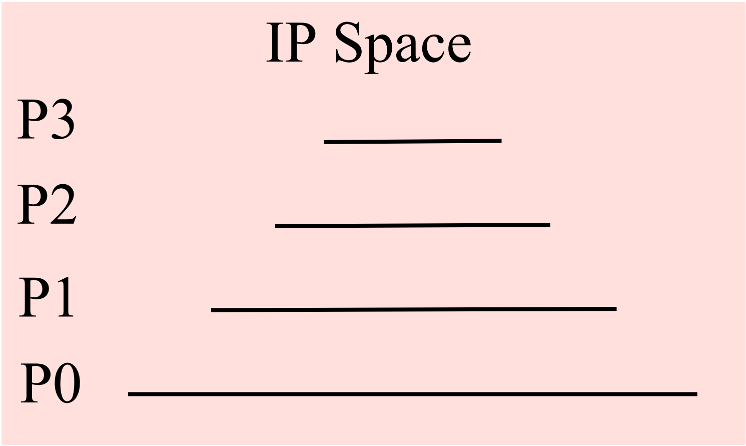
ByteDance字节跳动

WORKFLOW

Example Network >> Build Trie >> Compute RANGESET

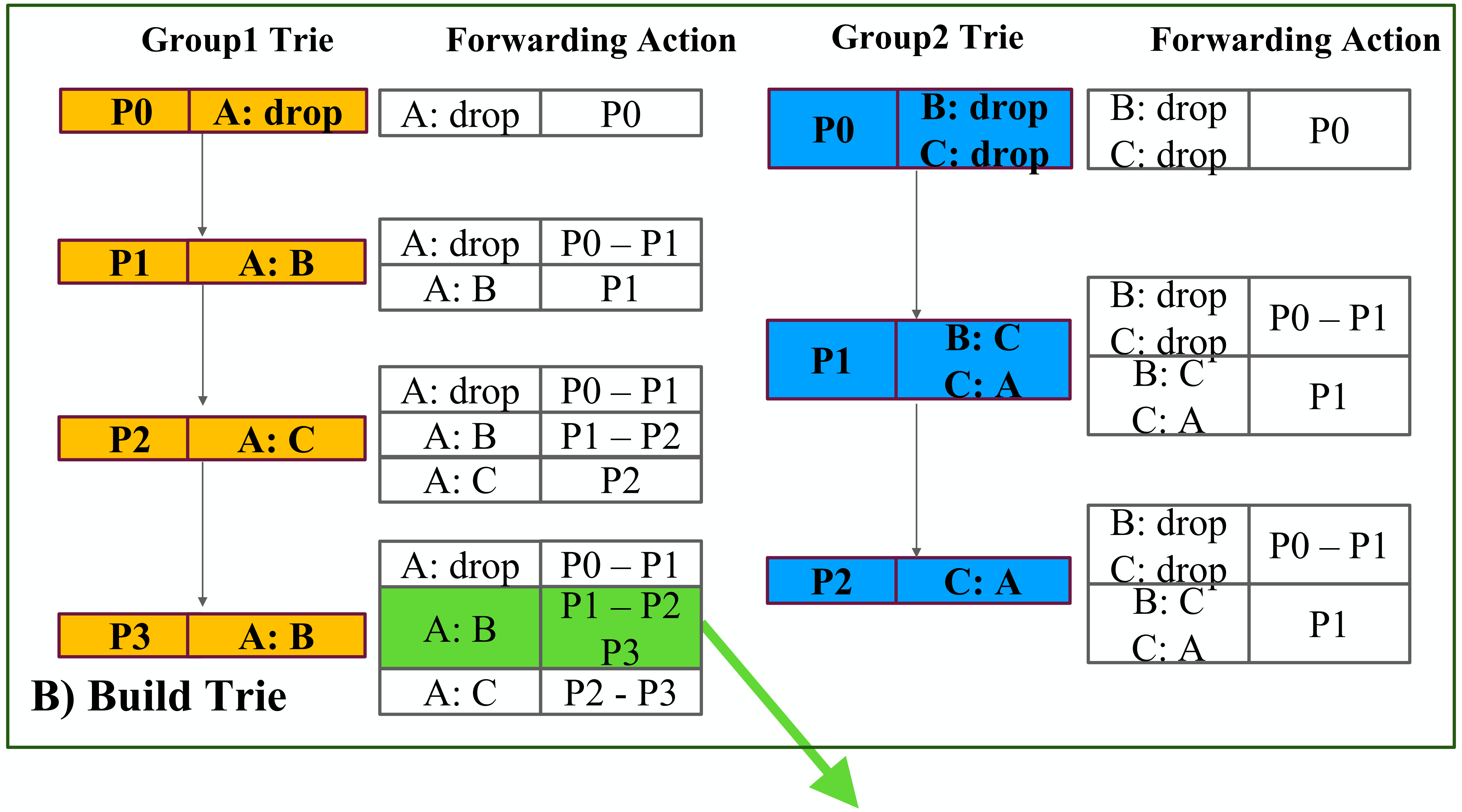


P1 – P2 + P3

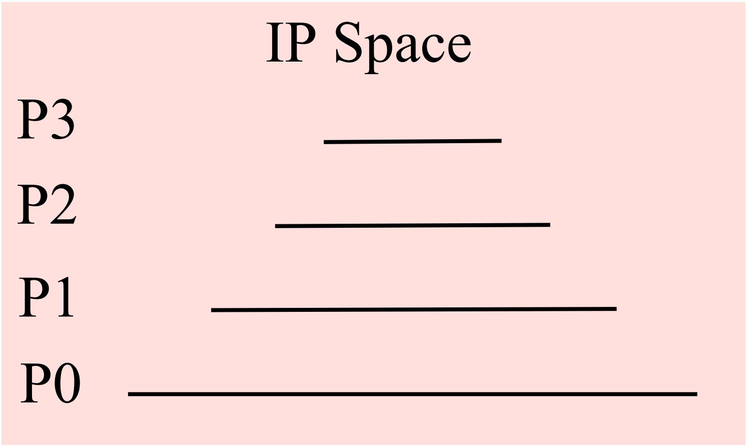


WORKFLOW

Example Network >> Build Trie >> Compute RANGESET

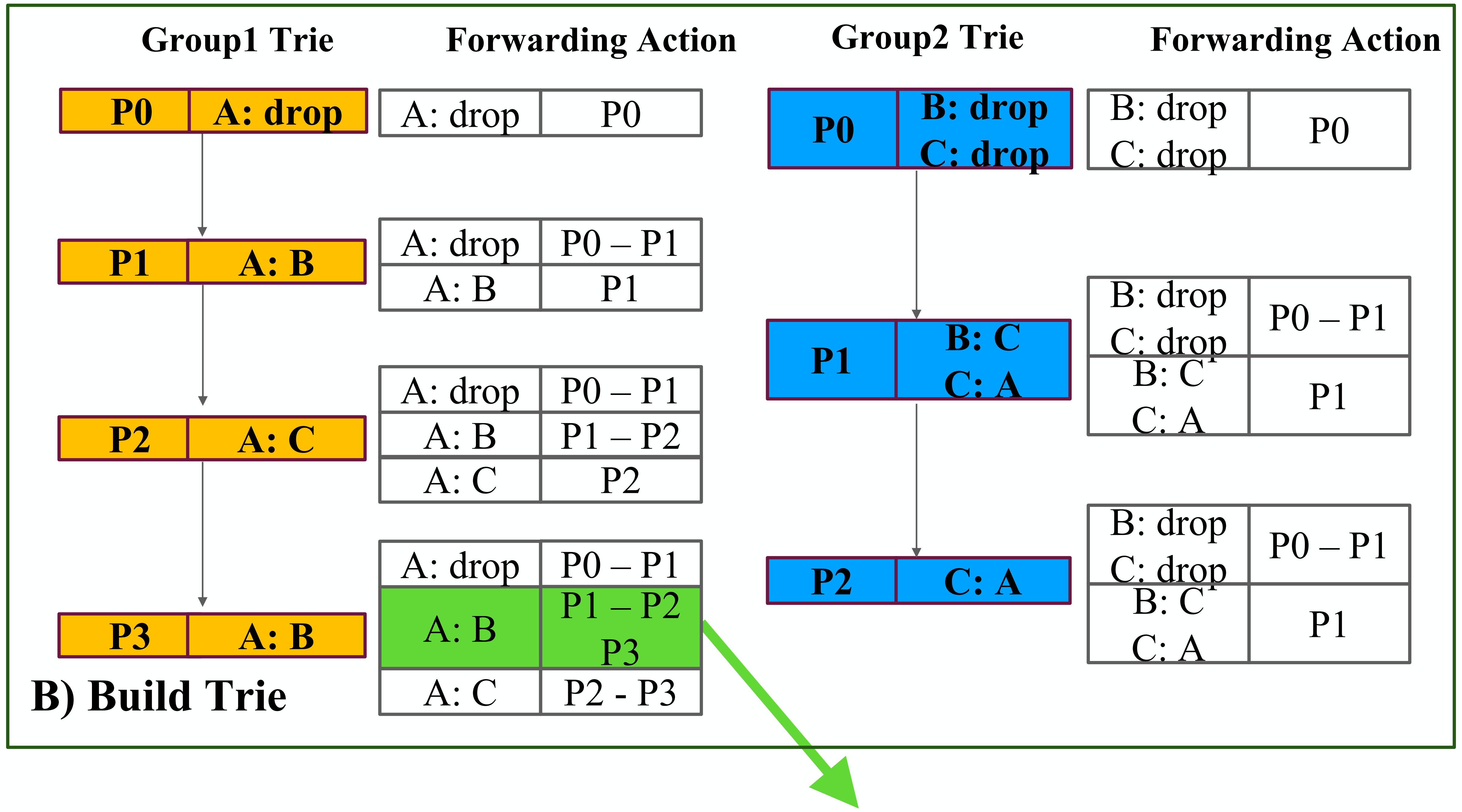


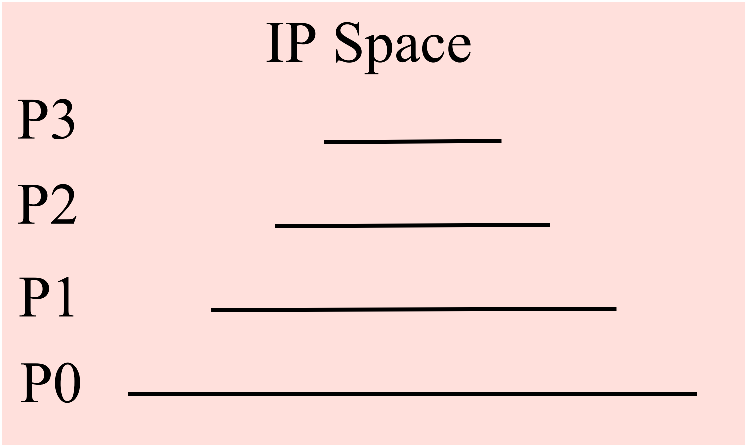
P1 – P2 + P3 = {(LB(P1), UB(P1))} – {(LB(P2), UB(P2))} + {(LB(P3), UB(P3))}



WORKFLOW

Example Network >> Build Trie >> Compute RANGESET



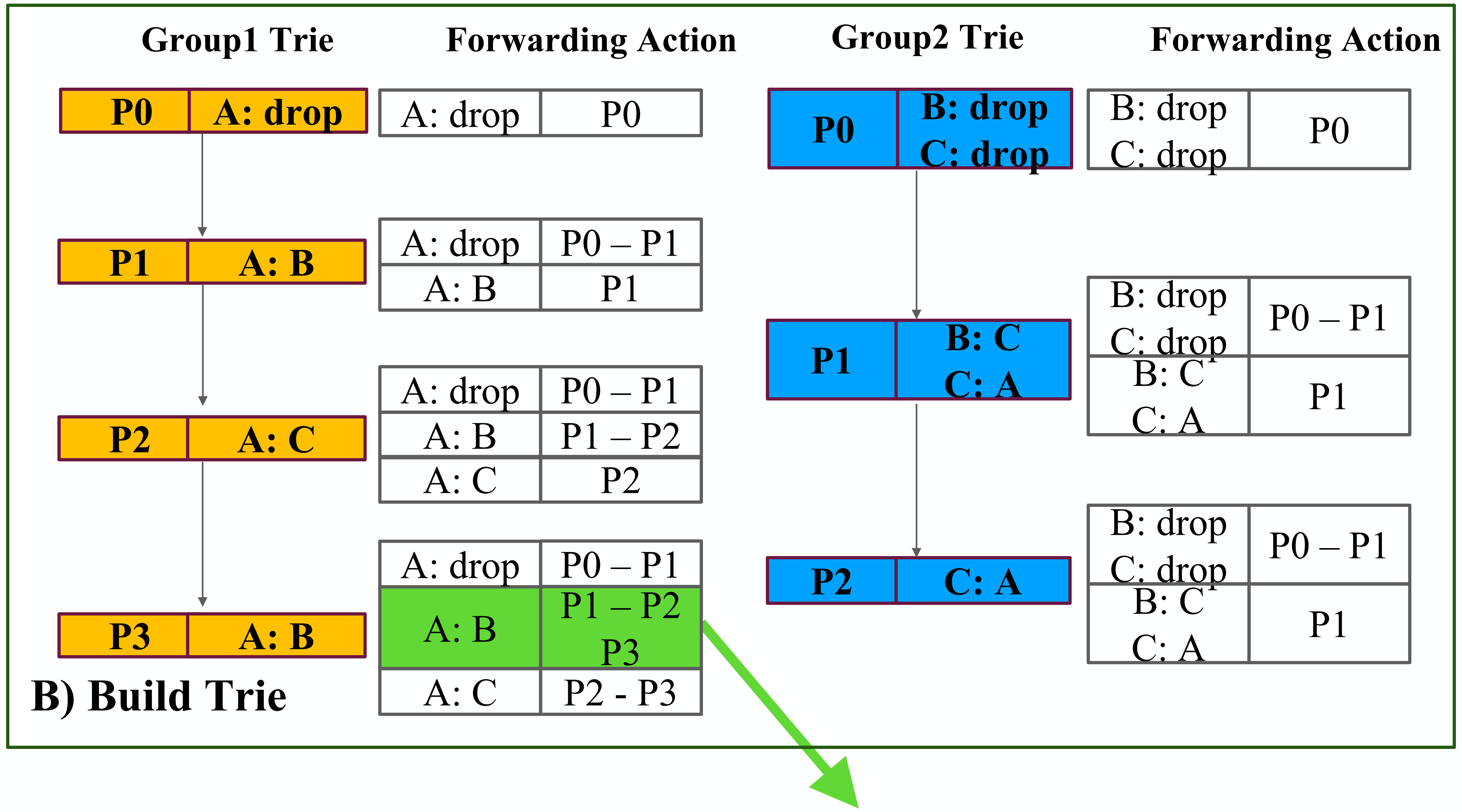


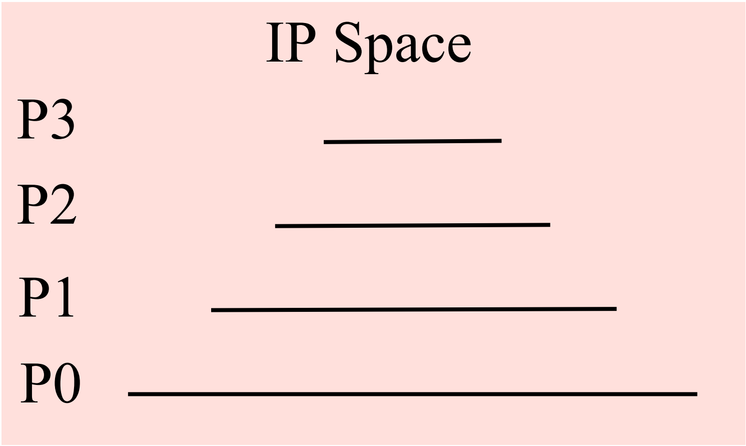
$$P1 - P2 + P3 = \{(LB(P1), UB(P1))\} - \{(LB(P2), UB(P2))\} + \{(LB(P3), UB(P3))\}$$

$$= \mathbf{AND}(\{(LB(P1), UB(P1))\}, \mathbf{NOT}(\{(LB(P2), UB(P2))\})) + \{(LB(P3), UB(P3))\}$$

WORKFLOW

Example Network >> Build Trie >> Compute RANGESET





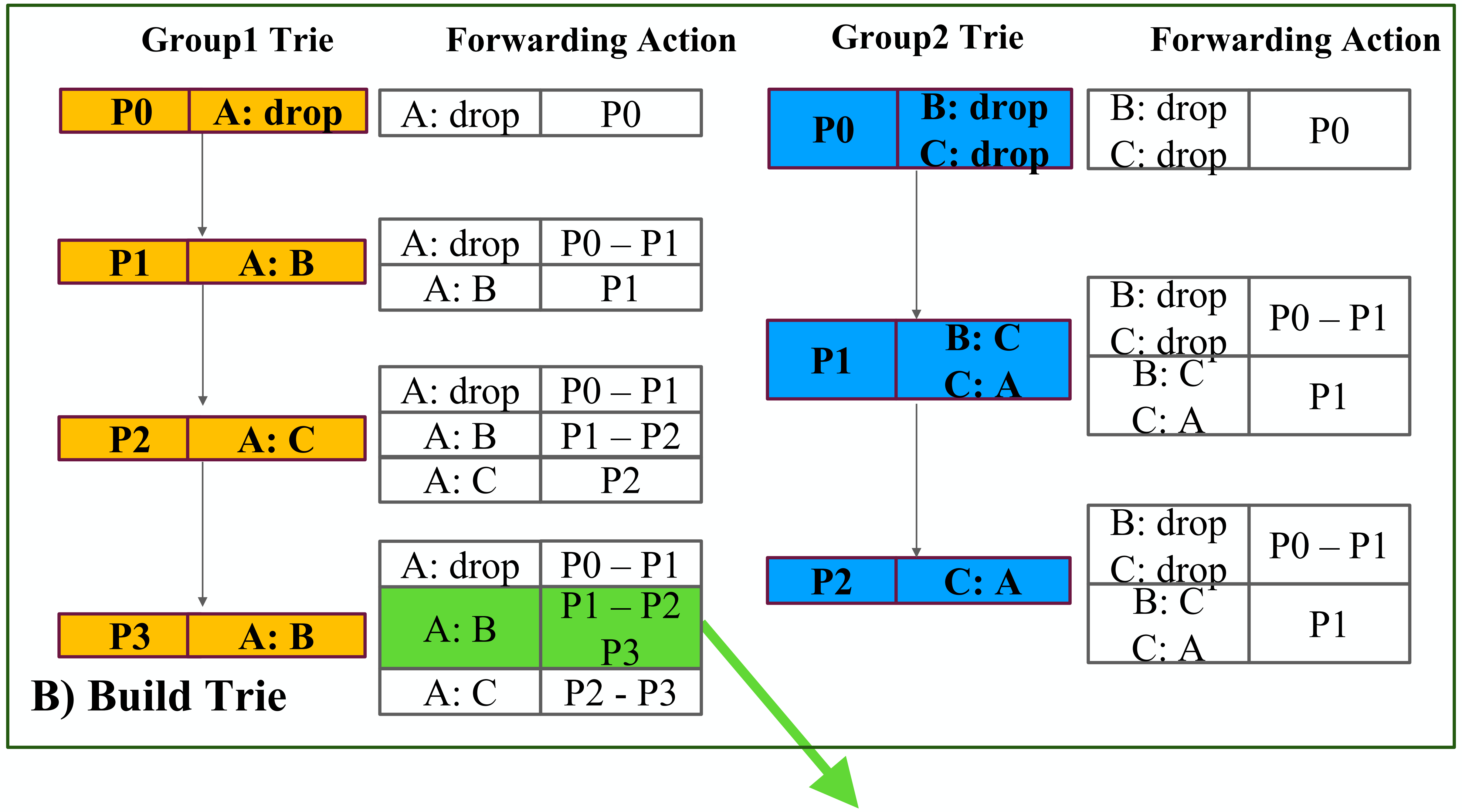
$$P1 - P2 + P3 = \{(LB(P1), UB(P1))\} - \{(LB(P2), UB(P2))\} + \{(LB(P3), UB(P3))\}$$

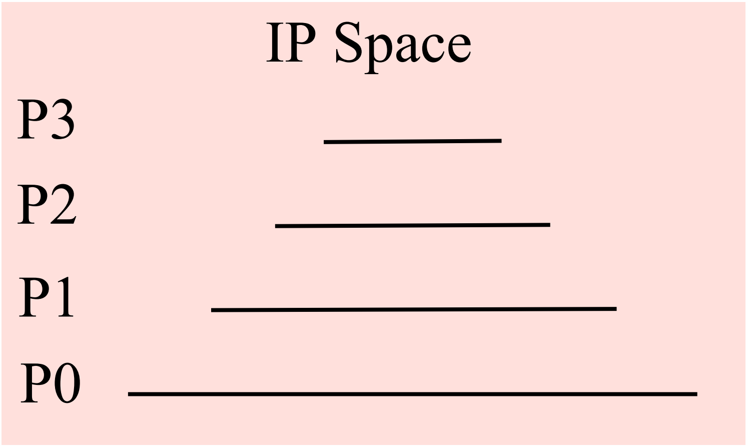
$$= AND(\{(LB(P1), UB(P1))\}, NOT(\{(LB(P2), UB(P2))\})) + \{(LB(P3), UB(P3))\}$$

$$= OR(AND(\{(LB(P1), UB(P1))\}, NOT(\{(LB(P2), UB(P2))\})), \{(LB(P3), UB(P3))\})$$

WORKFLOW

Example Network >> Build Trie >> Compute RANGESET





$$P1 - P2 + P3 = \{(LB(P1), UB(P1))\} - \{(LB(P2), UB(P2))\} + \{(LB(P3), UB(P3))\}$$

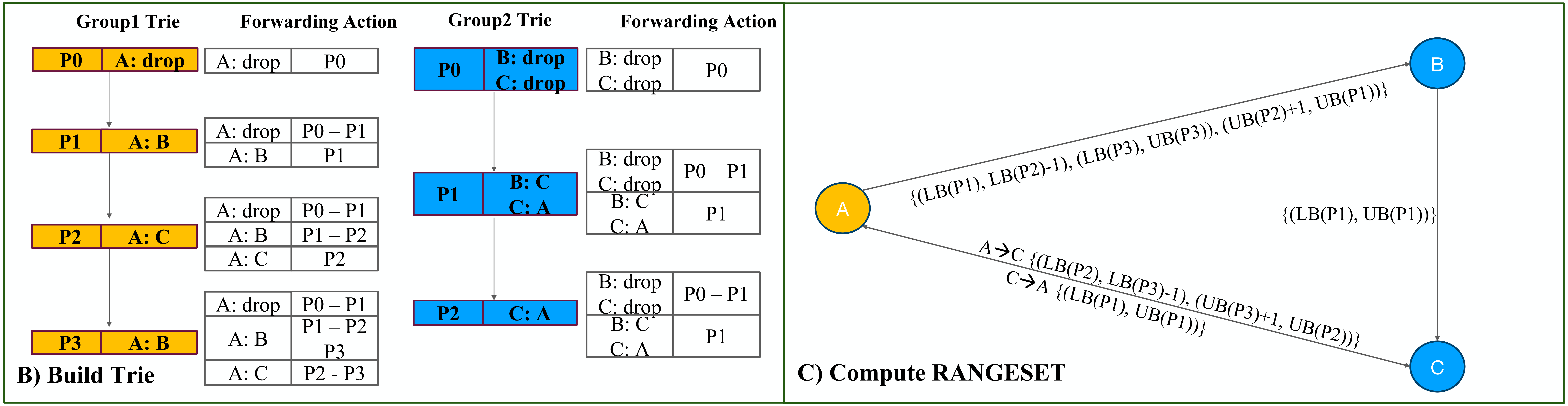
$$= AND(\{(LB(P1), UB(P1))\}, NOT(\{(LB(P2), UB(P2))\})) + \{(LB(P3), UB(P3))\}$$

$$= OR(AND(\{(LB(P1), UB(P1))\}, NOT(\{(LB(P2), UB(P2))\})), \{(LB(P3), UB(P3))\})$$

$$= \{(LB(P1), LB(P2)-1), (LB(P3), UB(P3)), (UB(P2)+1, UB(P1))\}$$

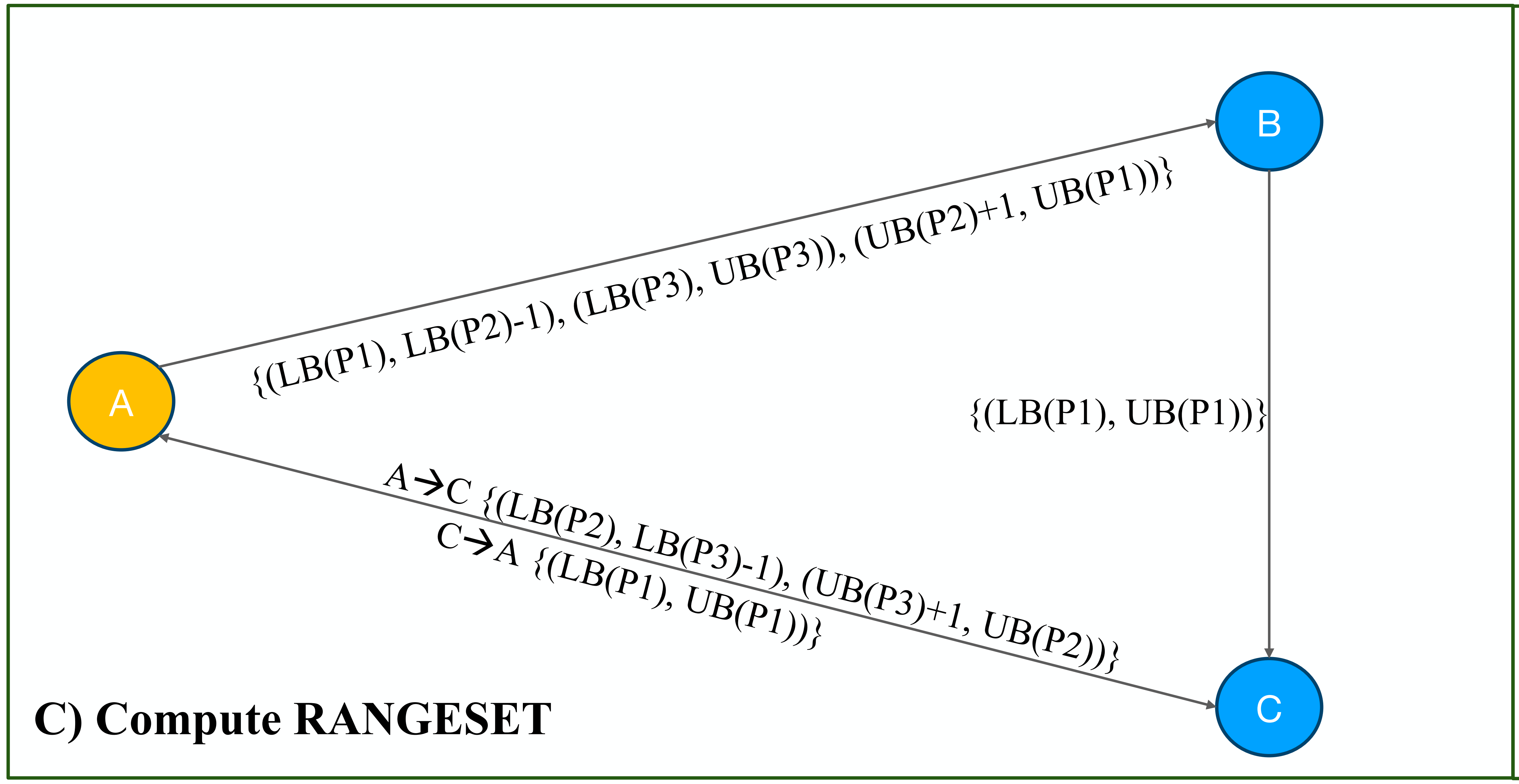
WORKFLOW

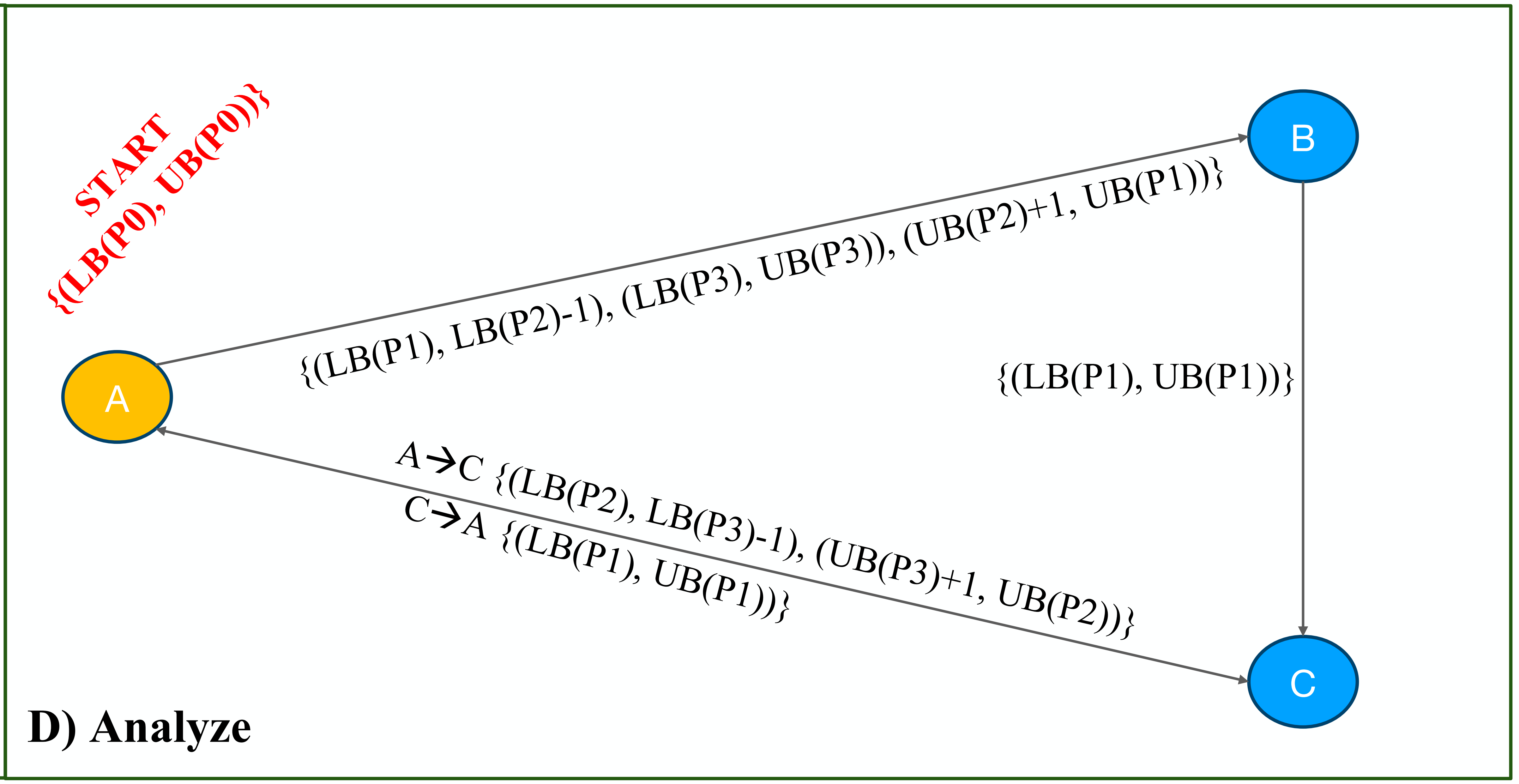
Example Network >> Build Trie >> Compute RANGESET



WORKFLOW

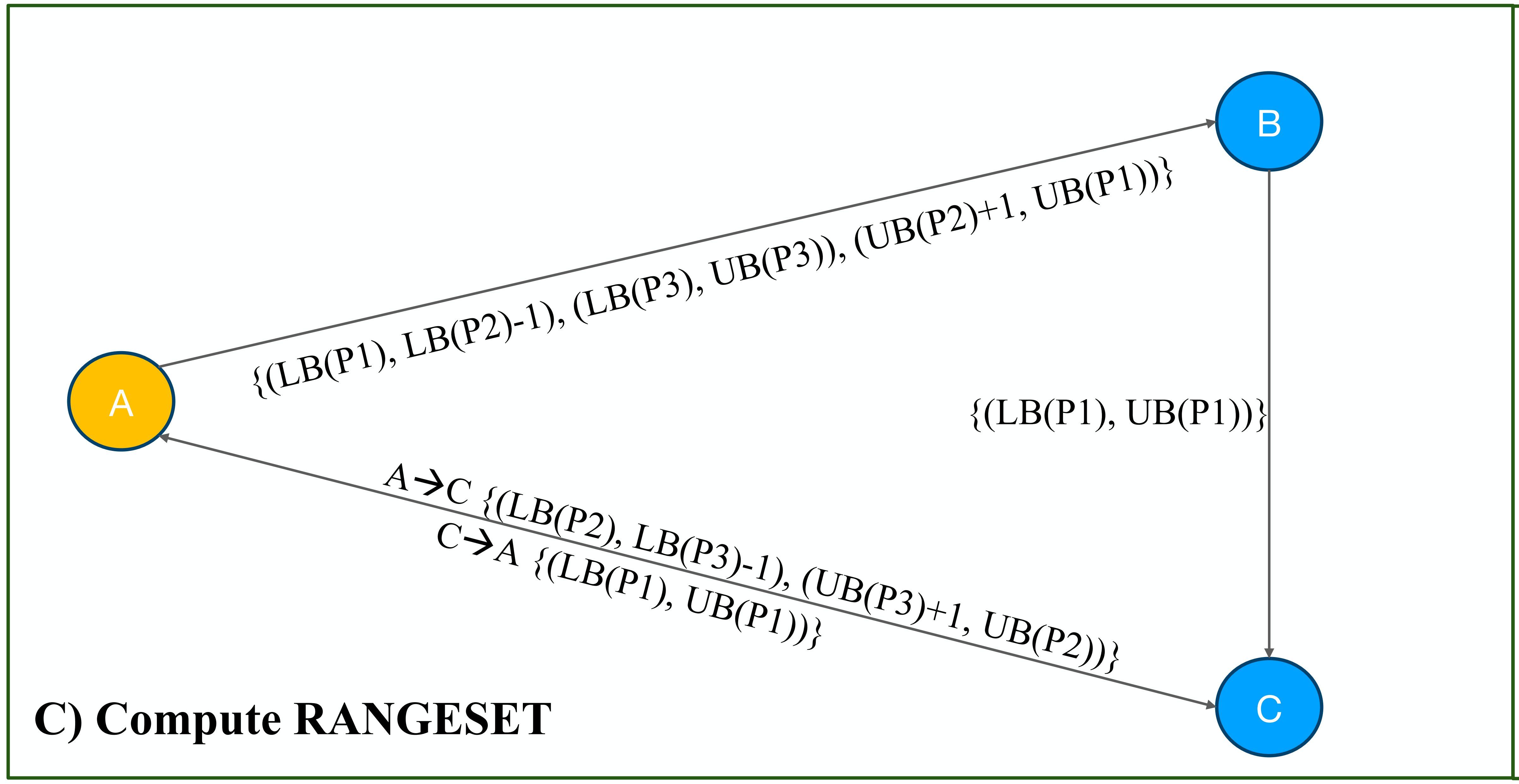
Example Network >> Build Trie >> Compute RANGESET >> Analyze

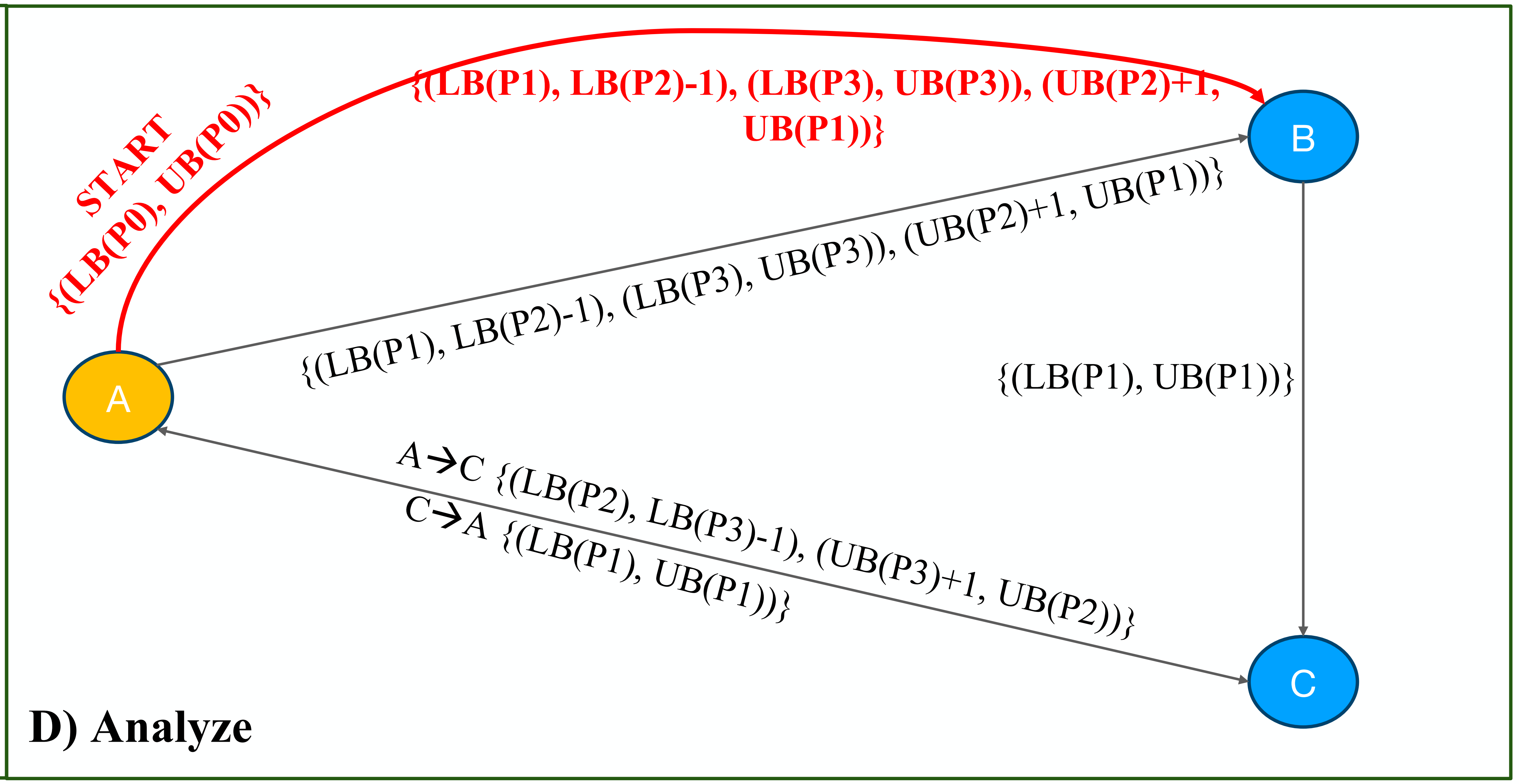




WORKFLOW

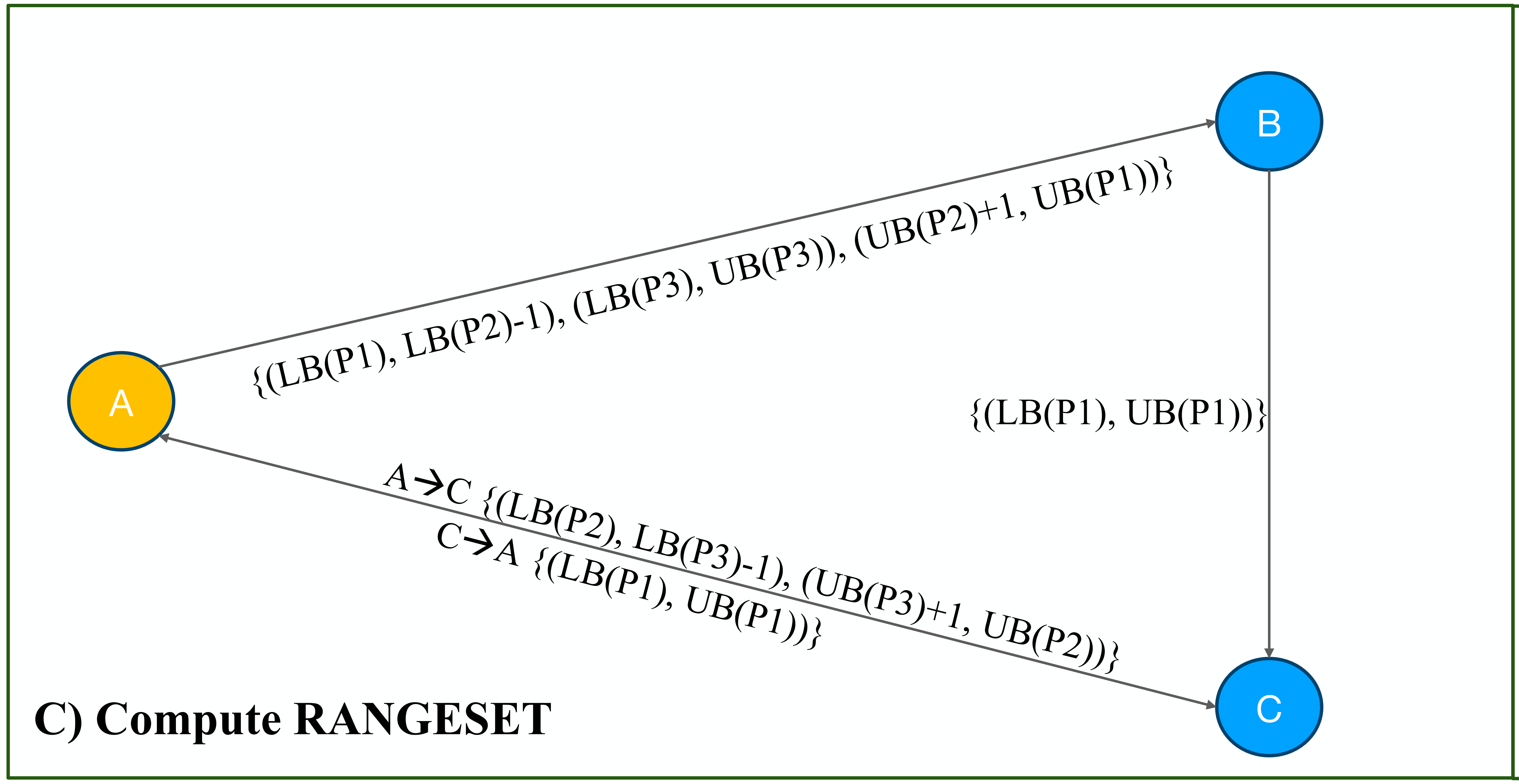
Example Network >> Build Trie >> Compute RANGESET >> Analyze

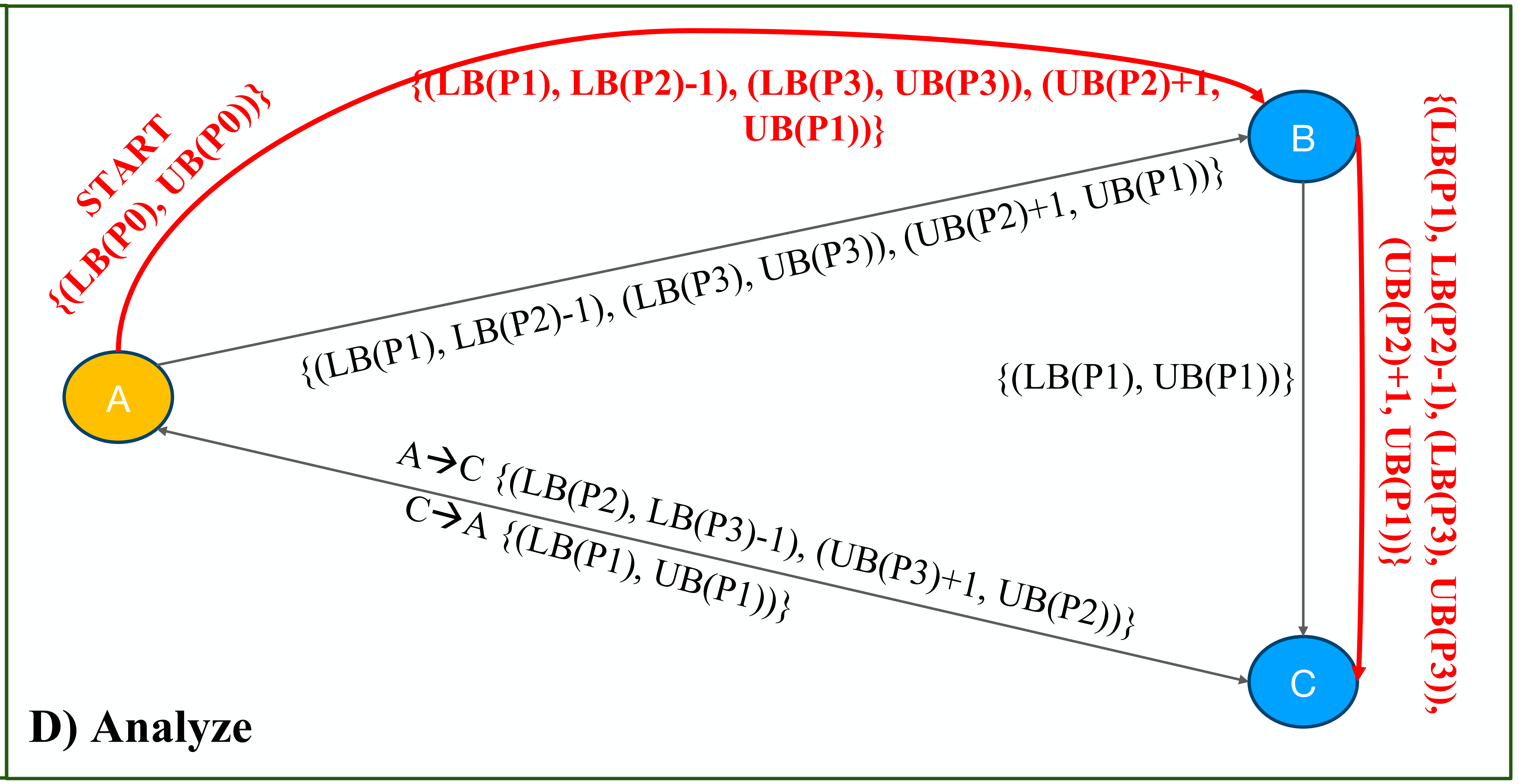




WORKFLOW

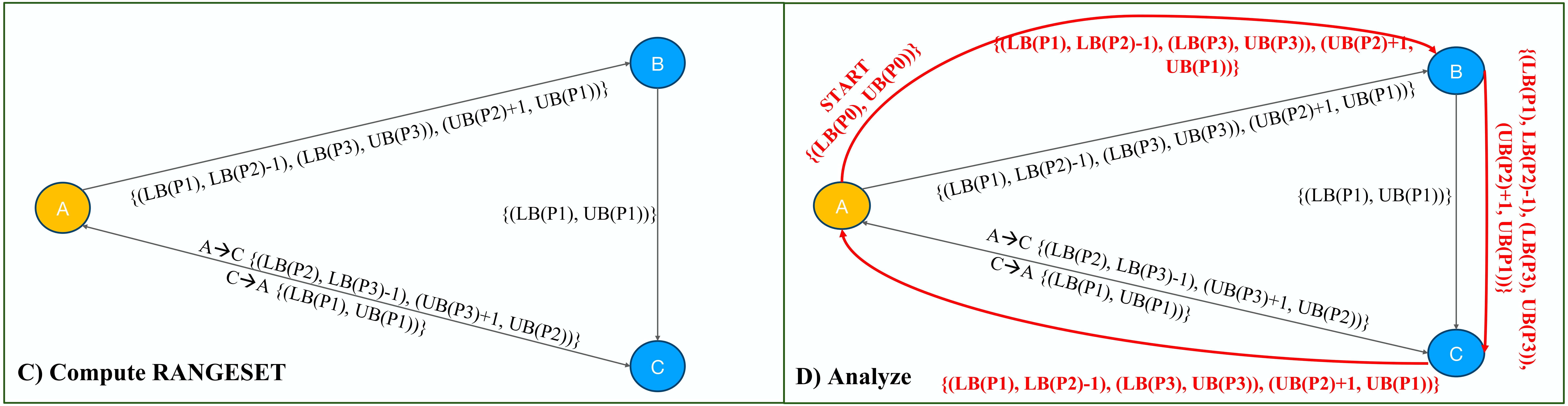
Example Network >> Build Trie >> Compute RANGESET >> Analyze





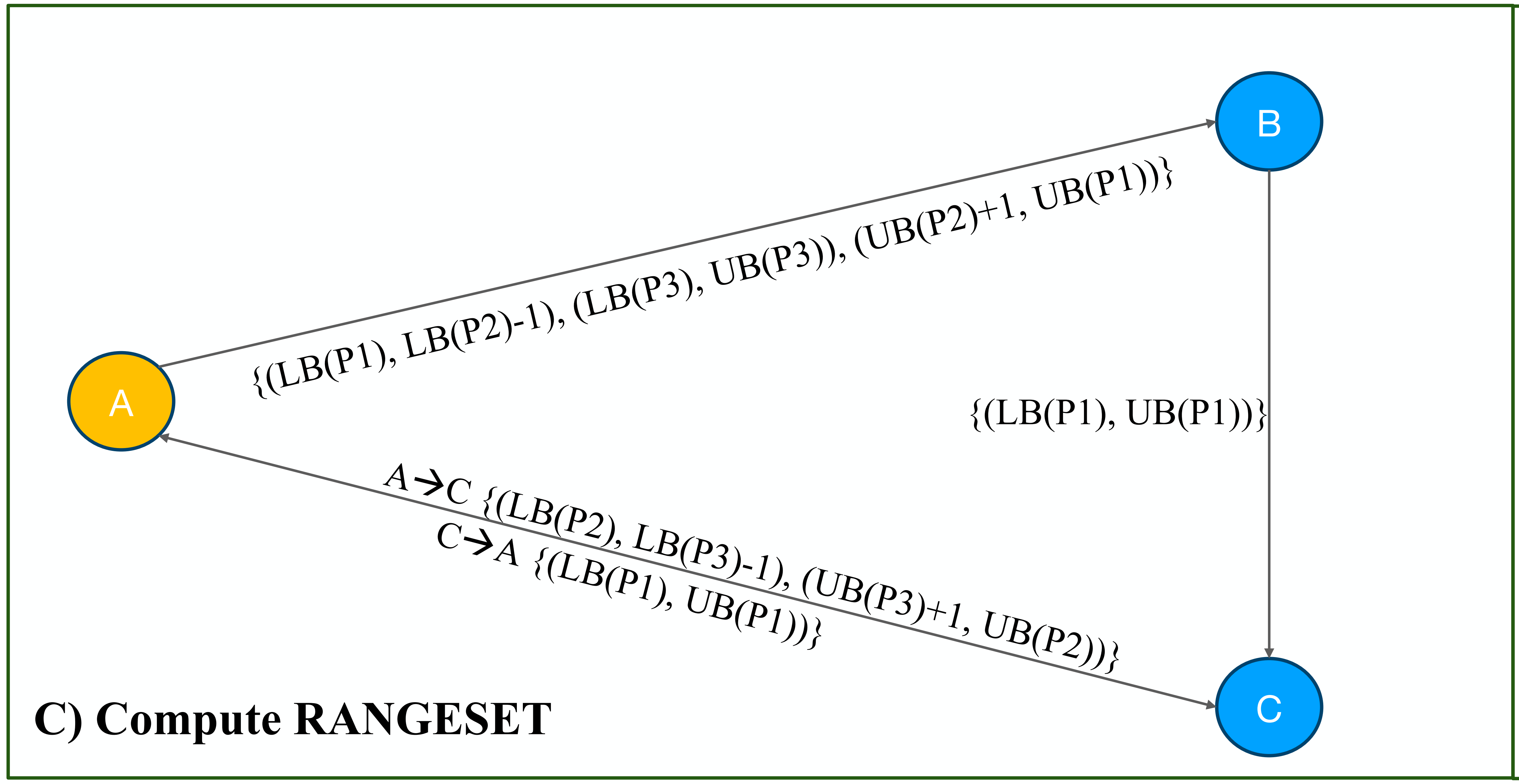
WORKFLOW

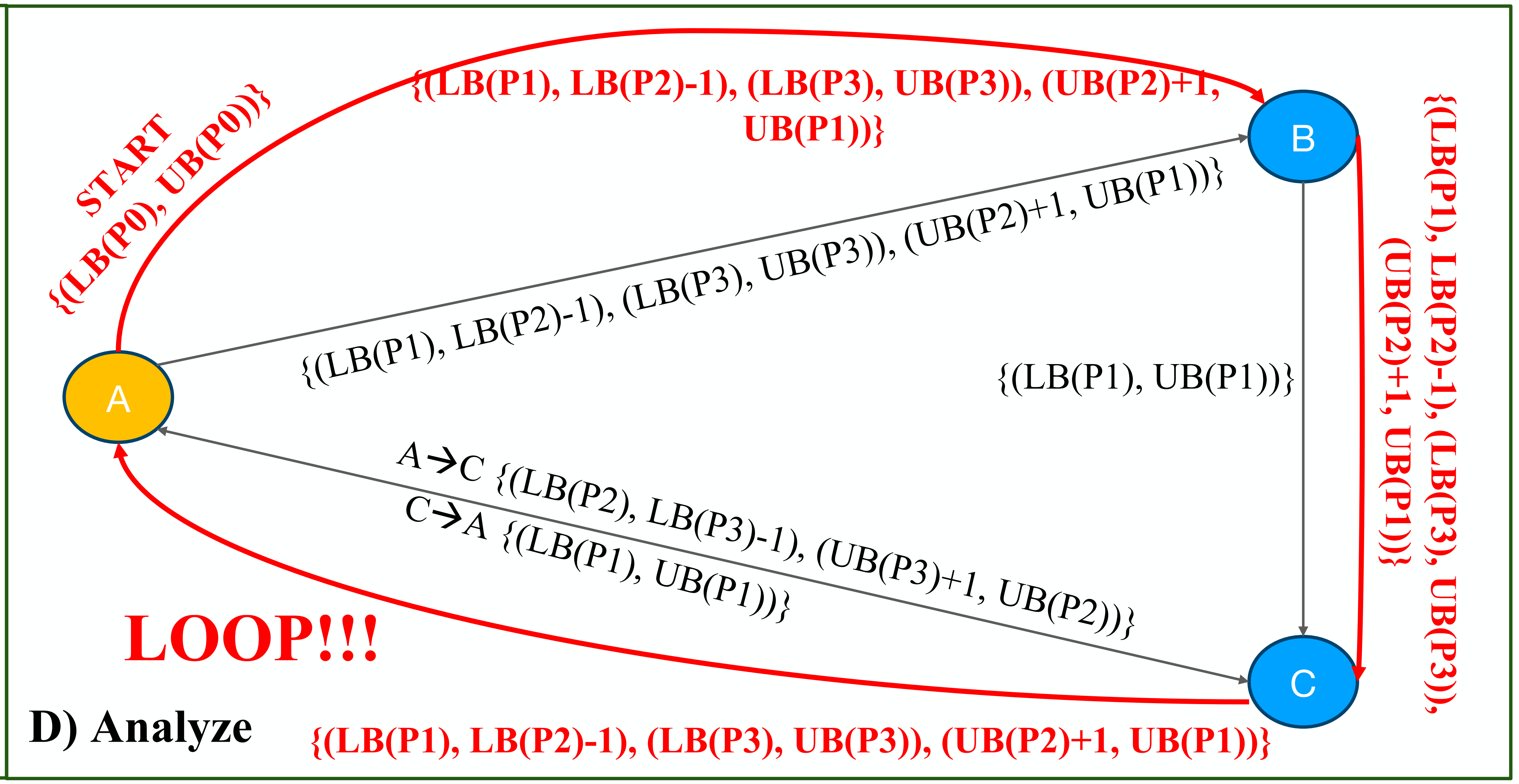
Example Network >> Build Trie >> Compute RANGESET >> Analyze



WORKFLOW

Example Network >> Build Trie >> Compute RANGESET >> Analyze





Evaluation - Dataset

- 3 popular public datasets(Airtel, I2 and Stanford)
- 4 datasets from our private data centers.

Dataset	Nodes	Links	Forwarding Rules
Airtel	O(10)	O(10)	O(100K)
I2	O(1)	O(10)	O(10K)
Stanford	O(10)	O(10)	O(1K)
PDC1	O(10)	O(100)	O(100K)
PDC2	O(100)	O(1K)	O(100K)
PDC3	O(1K)	O(10K)	O(1M)
PDC4	O(10K)	O(100K)	O(1M)

Evaluation - Runtime

- Loop verification using different DPVs
- 64 threads for Flash[java], Tulkun[java] and Medusa[cpp]
- >100x performance improvement
- Larger network, greater improvement

Tool	Runtime in seconds (speedup)			
	Flash	APKeep	Tulkun	*Medusa*
Airtel	2.76 (1.8)	32.99 (22.2)	1206 (814.8)	1.48
I2	0.55 (2.7)	5.54 (27.7)	1.46 (7.3)	0.20
Stanford	0.25 (25)	1.00 (100)	1.63 (163)	0.01
PDC1	1.13 (11.3)	14.73 (147.3)	4.01 (40.1)	0.10
PDC2	6.26 (20.2)	243.24 (784.6)	TO	0.31
PDC3	18.89 (48.4)	1629.83 (4179)	TO	0.39
PDC4	3002 (613.9)	TO	TO	4.89

Timeout: > 1 h

ByteDance字节跳动

Note: Tulkun timeout is due to the overhead of computing the DPVNet

Evaluation - Memory

- Less memory due to saving the extra overhead of ECs and BDDs

Tool	Memory in GB (memory reduction)			
	Flash	APKeep	Tulkun	*Medusa*
Airtel	4.23 (4.45)	1.26 (1.32)	3.91 (4.11)	0.95
I2	0.99 (5.82)	0.30 (1.76)	0.68 (4)	0.17
Stanford	1.01 (101)	0.11 (11)	0.62 (62)	0.01
PDC1	2.04 (10.2)	0.48 (2.4)	1.54 (7.7)	0.20
PDC2	5.56 (10.9)	1.16 (2.27)	TO	0.51
PDC3	7.10 (8.98)	2.93 (3.7)	TO	0.79
PDC4	93.51 (4.56)	TO	TO	20.49

Timeout: > 1 h

Future Work

- Integrate multiple parallel techniques
- Update the model incrementally
- Support more features like ACL

Summary

- The network data plane is getting much larger.
- The existing method fails to analyze such scale of network.
- We propose a new parallel framework to improve scalability.
- We divide the network into distinct groups and assign each group to a separate thread for computation. The results are then integrated for comprehensive verification.
- We achieve performance improvements of hundreds of times compared to start-of-the-art.

THANKS

ByteDance字节跳动