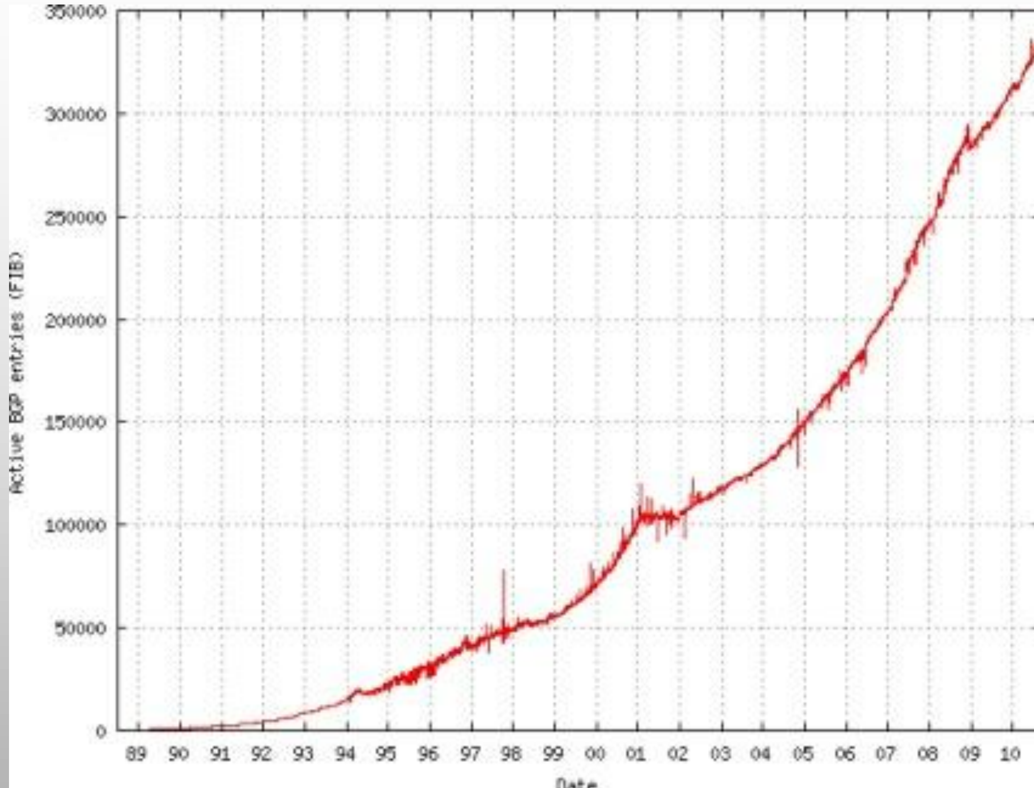
Online FIB Aggregation without Update Churn

Stefan Schmid
(TU Berlin & T-Labs)

joint work with
Marcin Bienkowski
Nadi Sarrar
Steve Uhlig

Growth of Routing Tables



Reasons: scale, virtualization, IPv6 may not help, …

Local FIB Compression: 1-Page Overview

Routers or SDN Switches

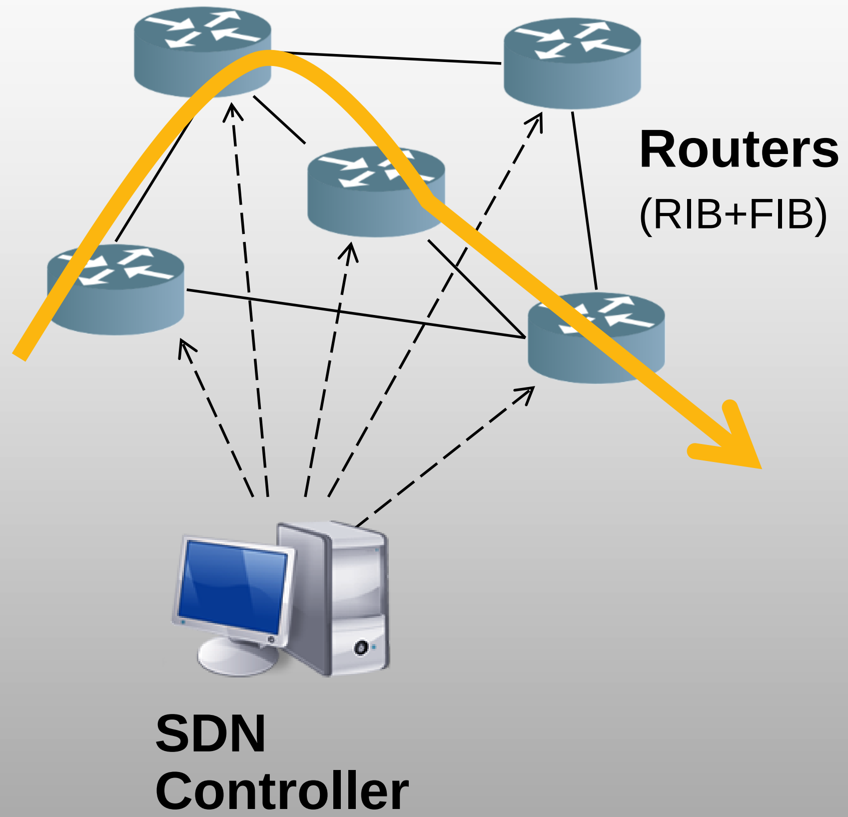
- RIB: Routing Information Base
- FIB: Forwarding Information Base
- FIB consists of
  - set of <prefix, next-hop>

Basic Idea

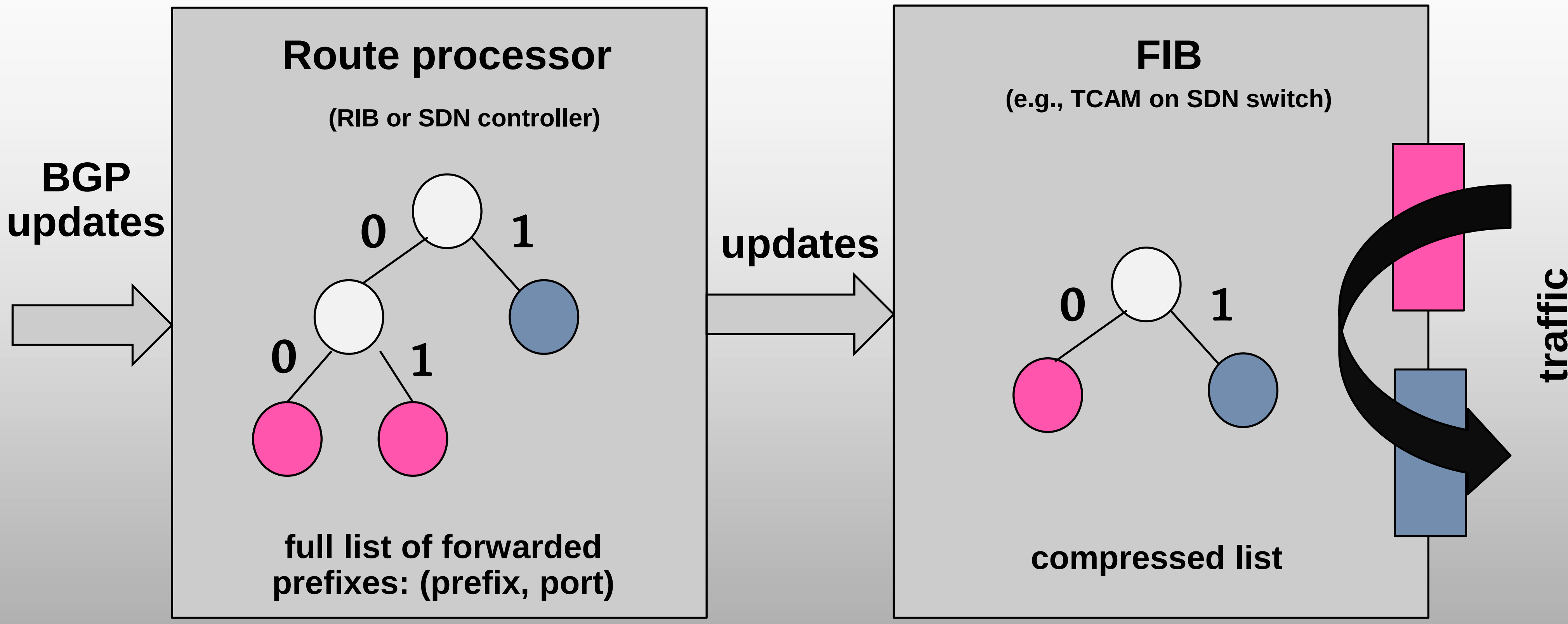
- Dynamically aggregate FIB
  - "Adjacent" prefixes with same next-hop (= color): one rule only!
- But be aware that BGP updates (next-hop change, insert, delete) may change forwarding set, need to deaggregate again
- Additional churn is bad: rebuild internal FIB structures, traffic between controller and switch, etc.

Benefits

- Only single router affected
- Other routers do not notice
- Aggregation = simple software update



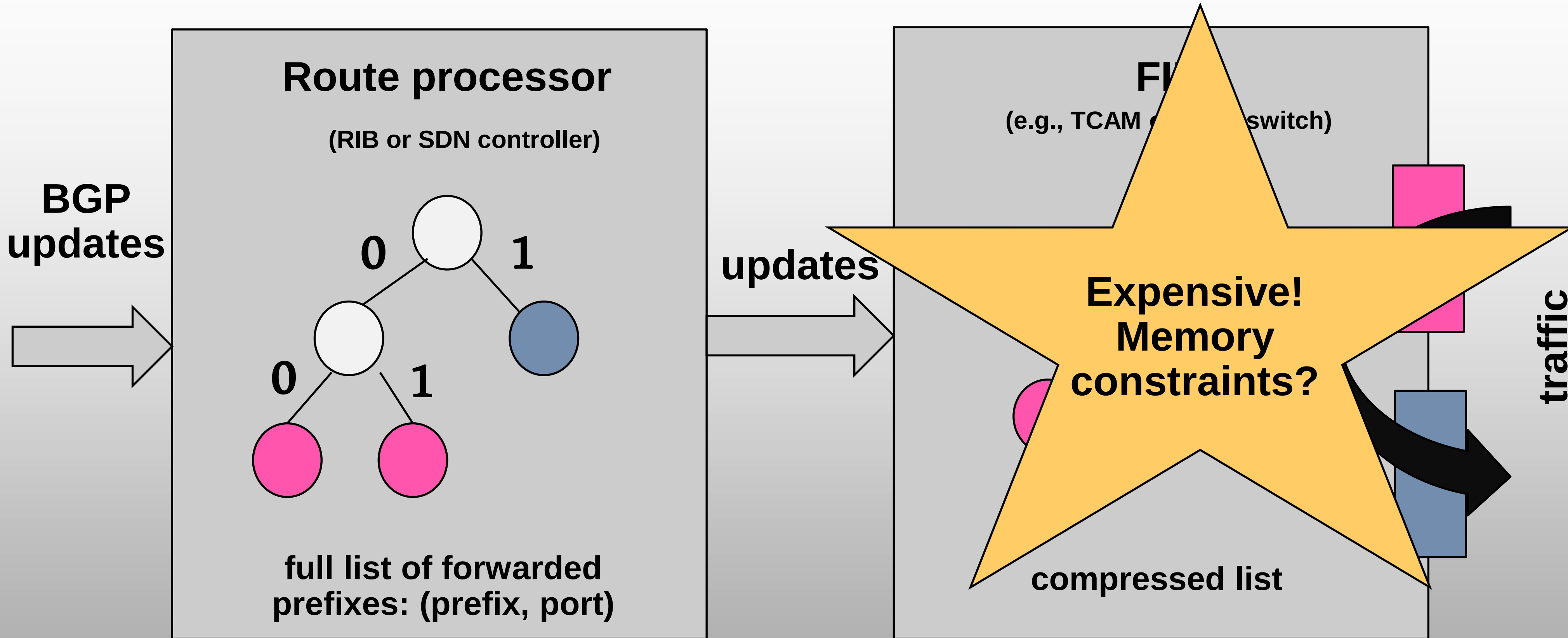
Setting: A Memory-Efficient Switch/Router



Goal: keep FIB small but consistent!

Without sending too many additional updates.

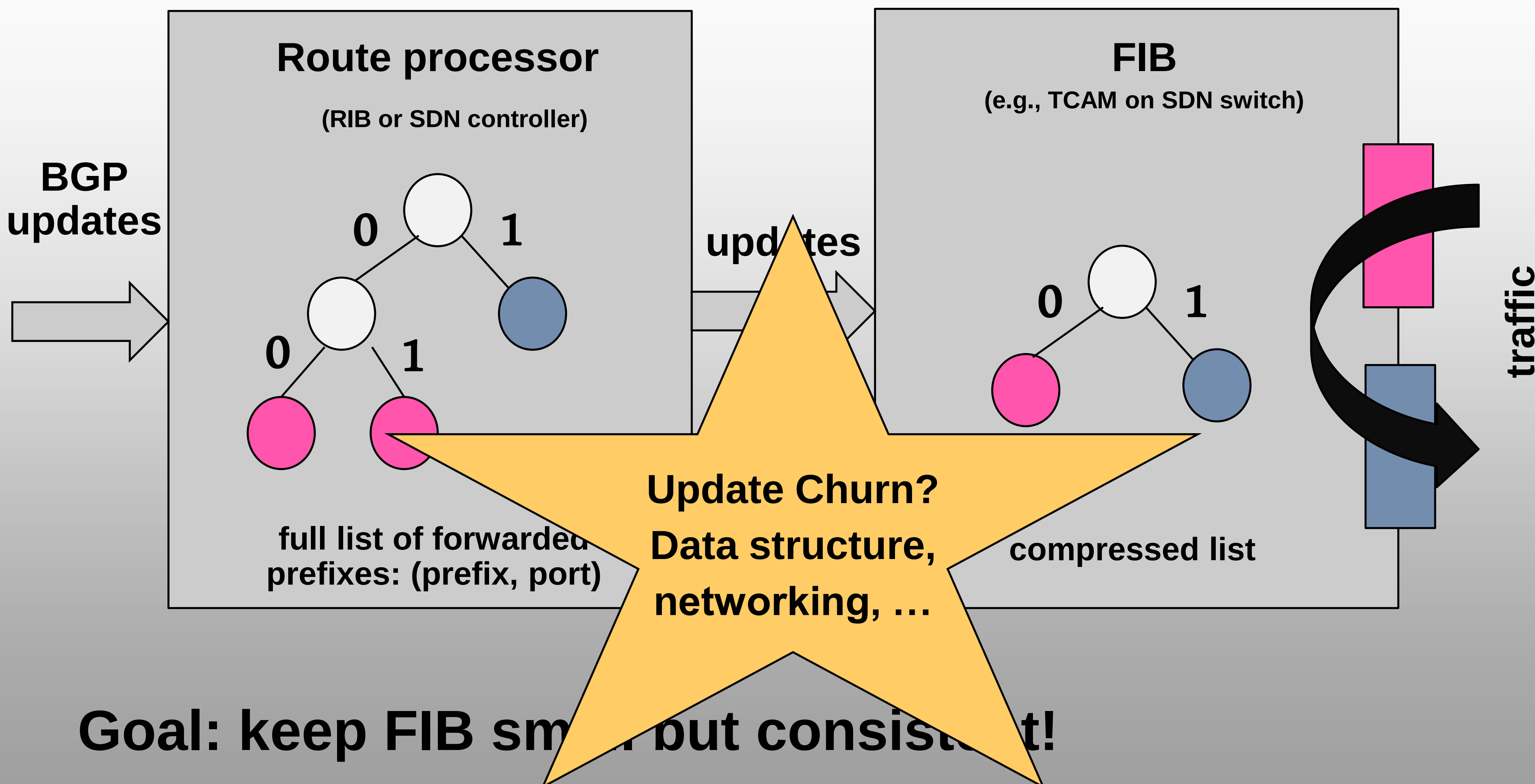
Setting: A Memory-Efficient Switch/Router



Goal: keep FIB small but consistent!

Without sending too many additional updates.

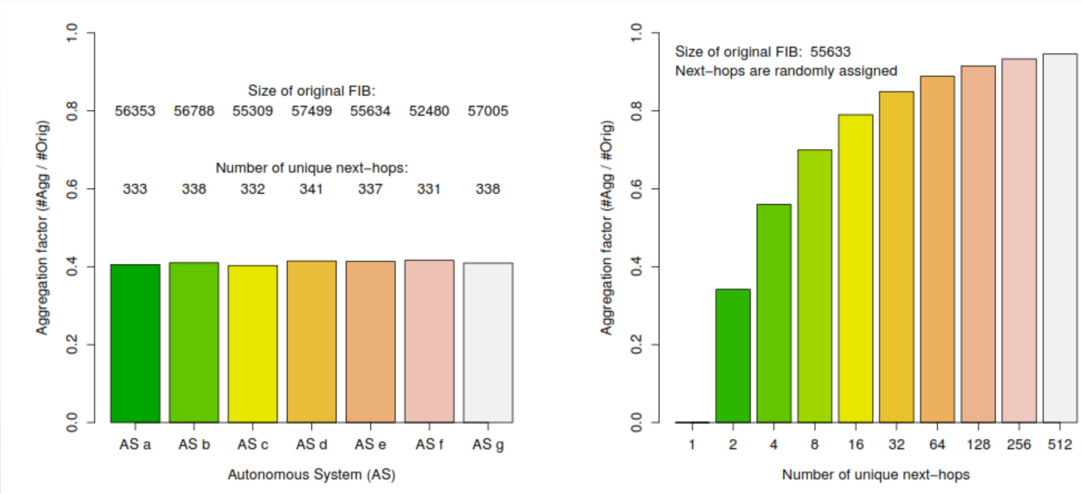
Setting: A Memory-Efficient Switch/Router



Goal: keep FIB small but consistent!

Without sending too many additional updates.

Motivation: FIB Compression and Update Churn

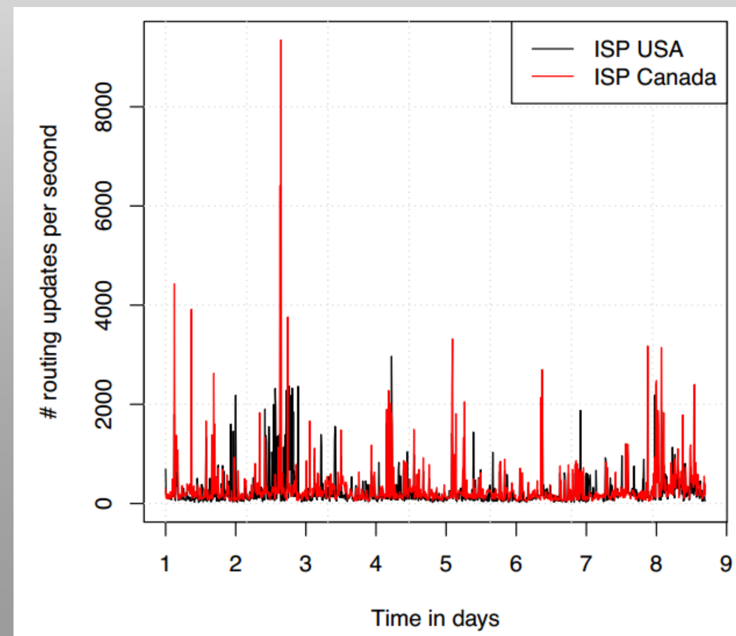


Benefits of FIB aggregation

- Routeview snapshots indicate 40% memory gains
- More than under uniform distribution
- But depends on number of next hops

Churn

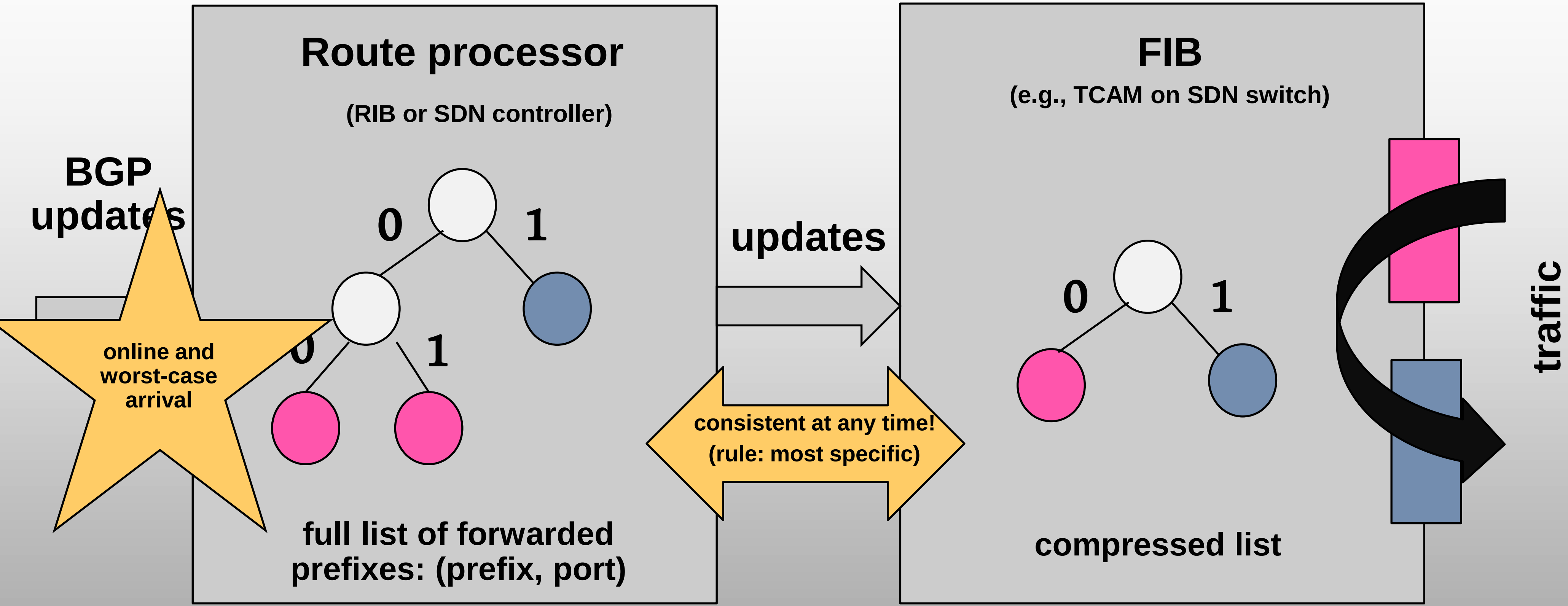
- Thousands of routing updates per second
- Goal: do not increase more



Stefan Schmid (T-Labs)

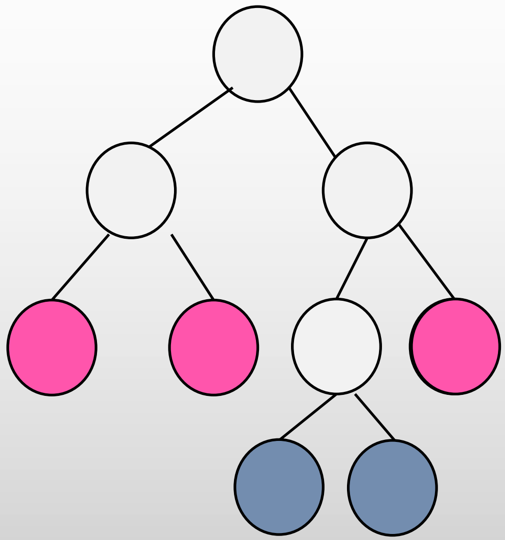
Model: Costs

Ports = Next-Hops = Colors

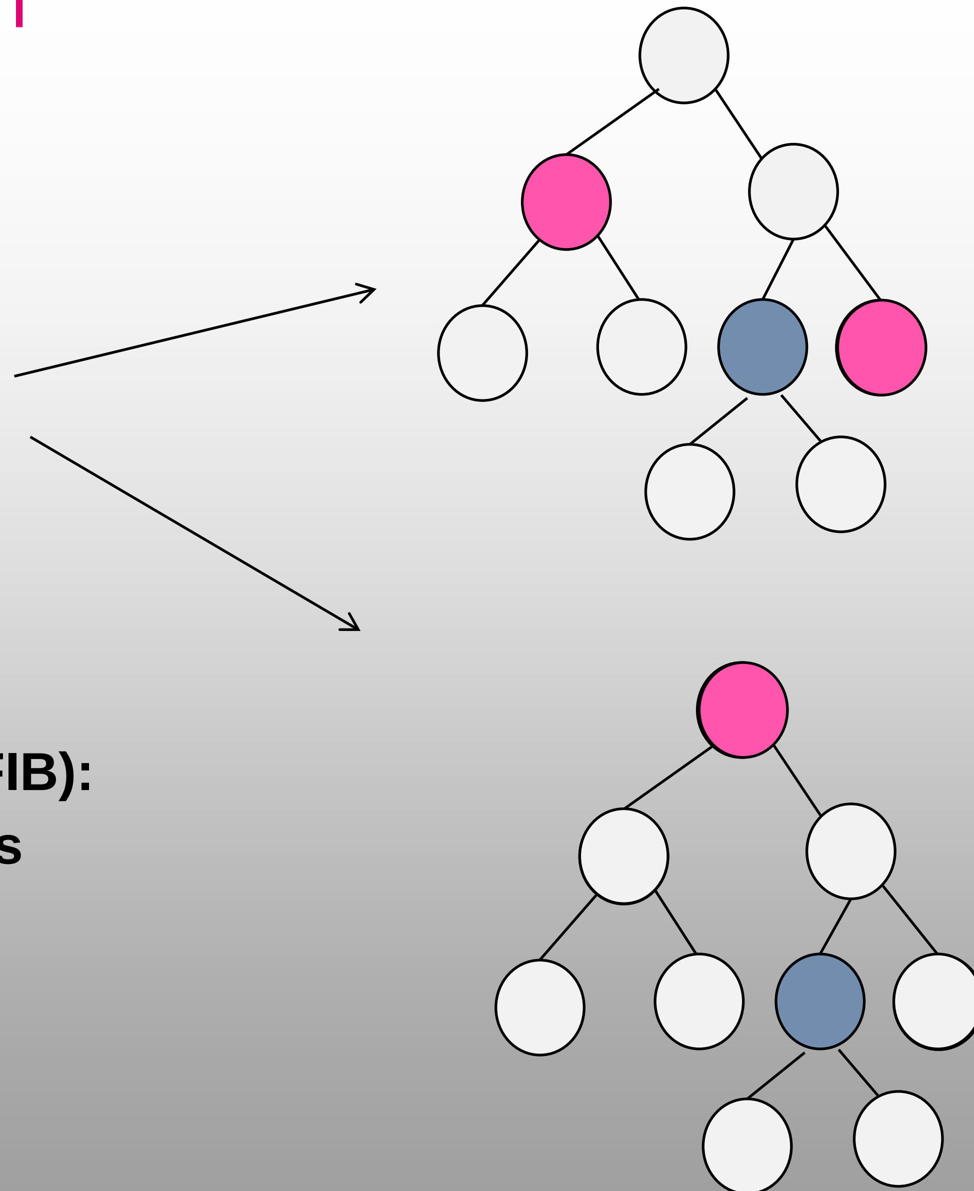


$$\text{Cost} = \alpha\ (\#\ \text{updates to FIB}) + \int_t \text{memory}$$

Model: Aggregation



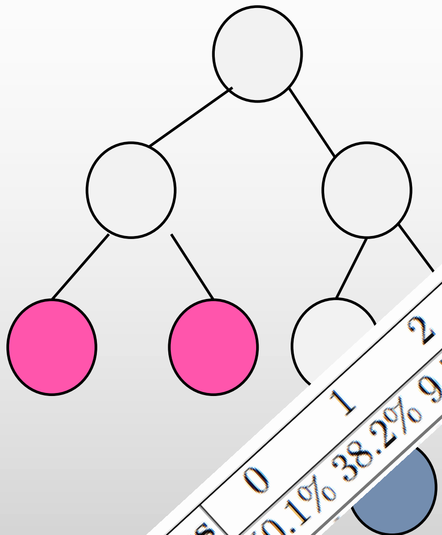
Uncompressed FIB (UFIB):
independent prefixes
size 5



FIB w/o exceptions
size 3

FIB w/ exceptions
size 2

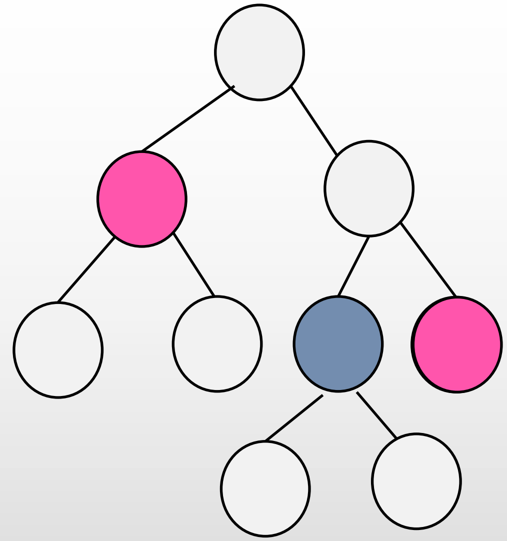
Model: Aggregation



# less specifics	0	1	2	3	4	5	6
% of prefixes	50.1%	38.2%	9.5%	1.7%	0.4%	0.1%	0.01%

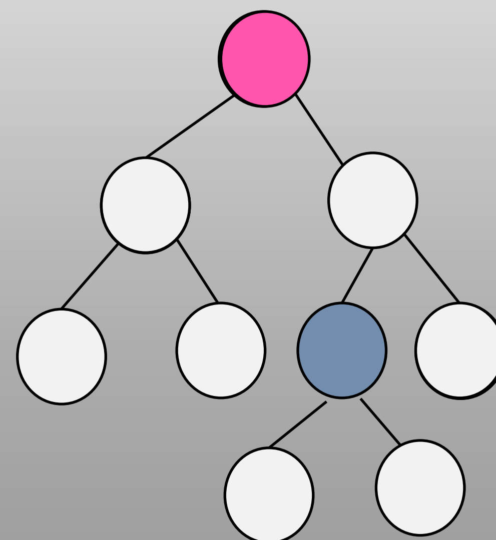
Uncompressed FIB (UFIB): independent prefixes

size 5



FIB w/o exceptions

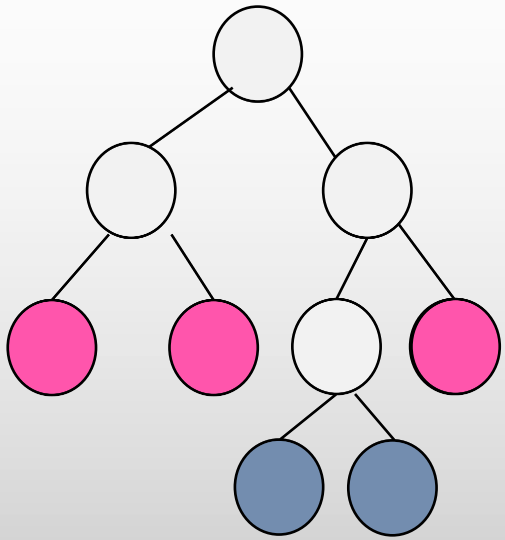
size 3



FIB w/ exceptions

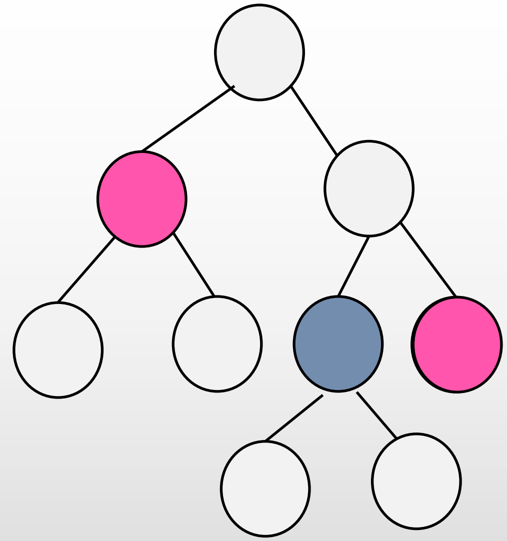
size 2

Model: Aggregation



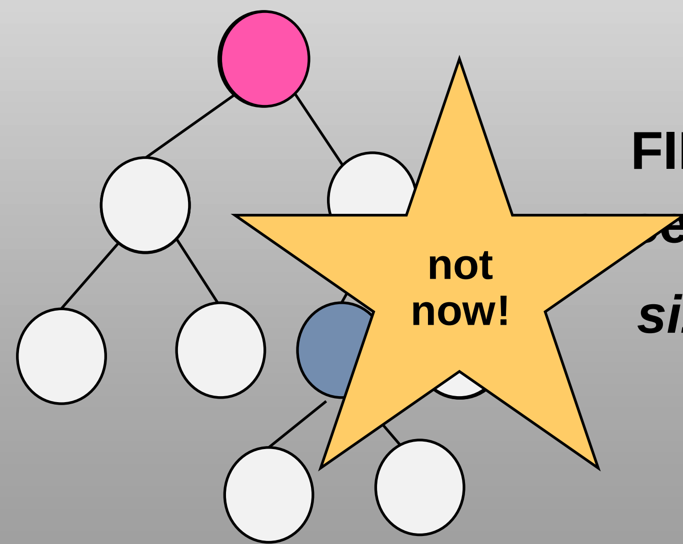
Uncompressed FIB (UFIB): independent prefixes

size 5



FIB w/o exceptions

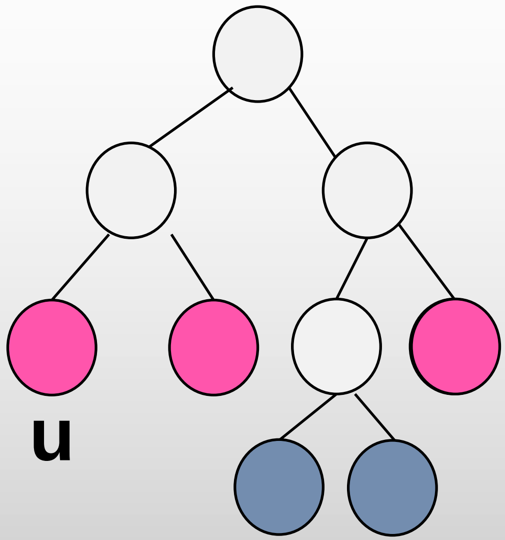
size 3

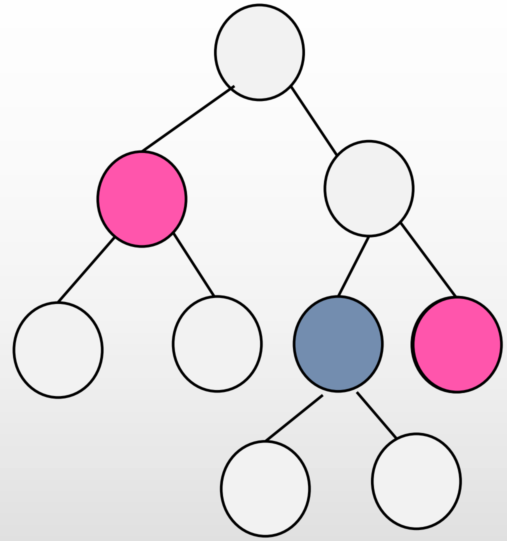


FIB w/ eptions

size 2

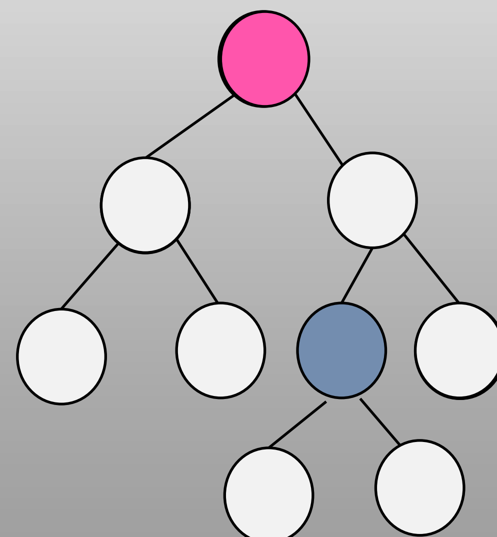
Model: Aggregation





FIB w/o exceptions

size 3



FIB w/ exceptions

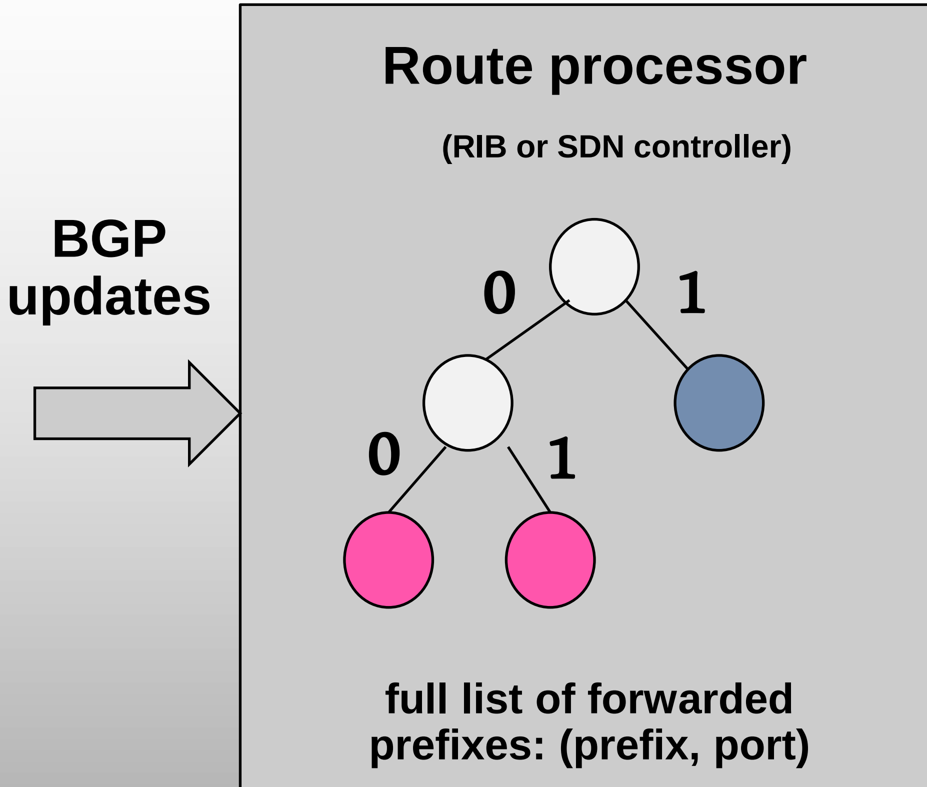
size 2

Uncompressed FIB (UFIB): independent prefixes

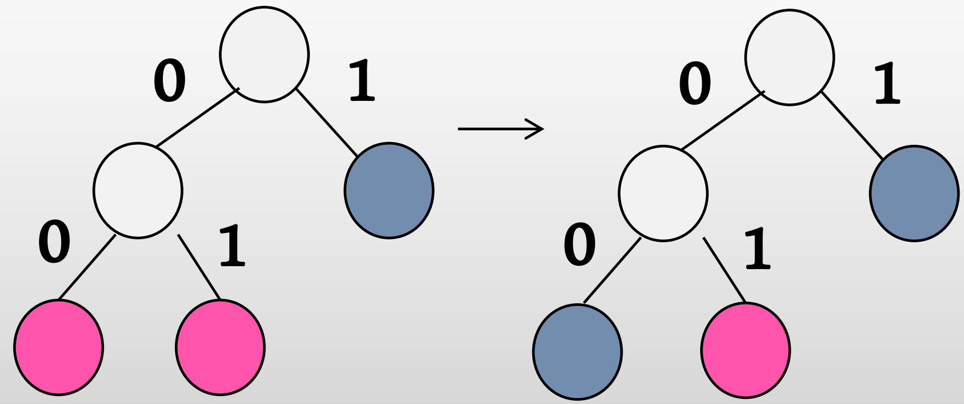
size 5

Note: if node u changes color to blue, three updates are required in the compressed tries! (remove one, insert two)

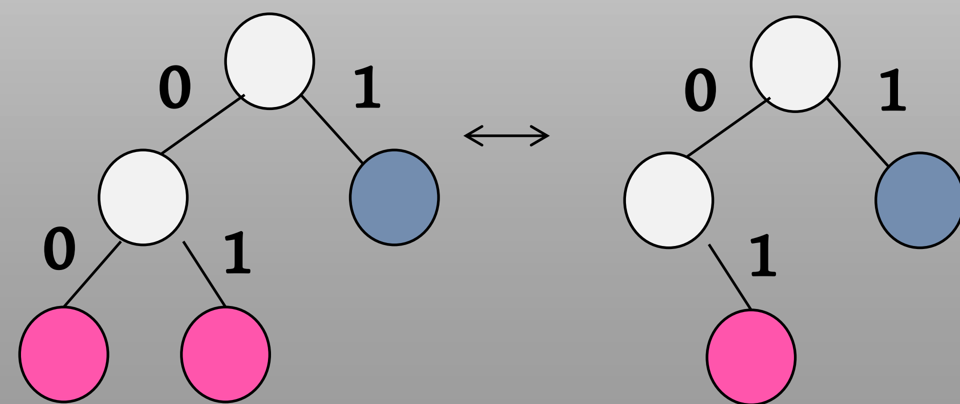
Model: Online Input Sequence



Update: Color change



Update: Insert/Delete



Model: Online Perspective

Competitive analysis framework:

Online Algorithm

Online algorithms make decisions at time t without any knowledge of inputs at times t'>t.

Competitive Ratio

Competitive ratio r,

$$r = Cost(ALG) / cost(OPT)$$

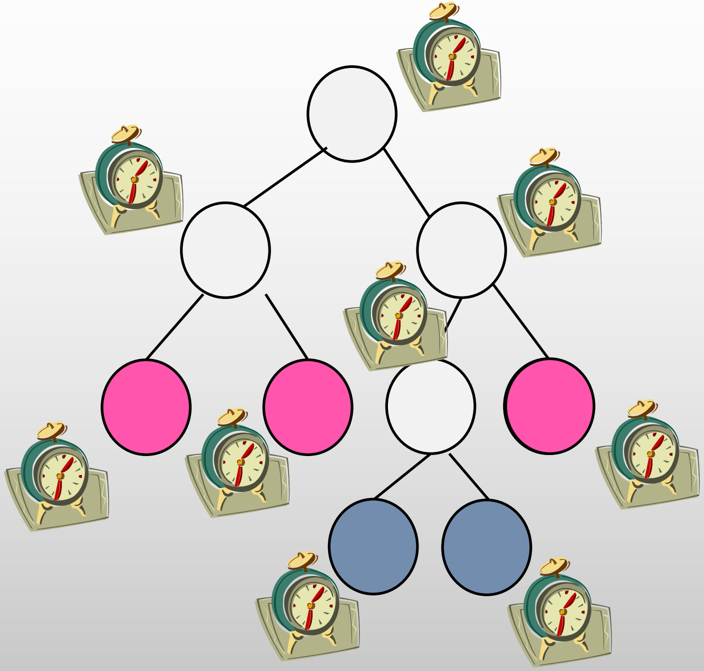
The price of not knowing the future!

Competitive Analysis

An *r-competitive online algorithm* ALG gives a worst-case performance guarantee: the performance is at most a factor r worse than an optimal offline algorithm OPT!

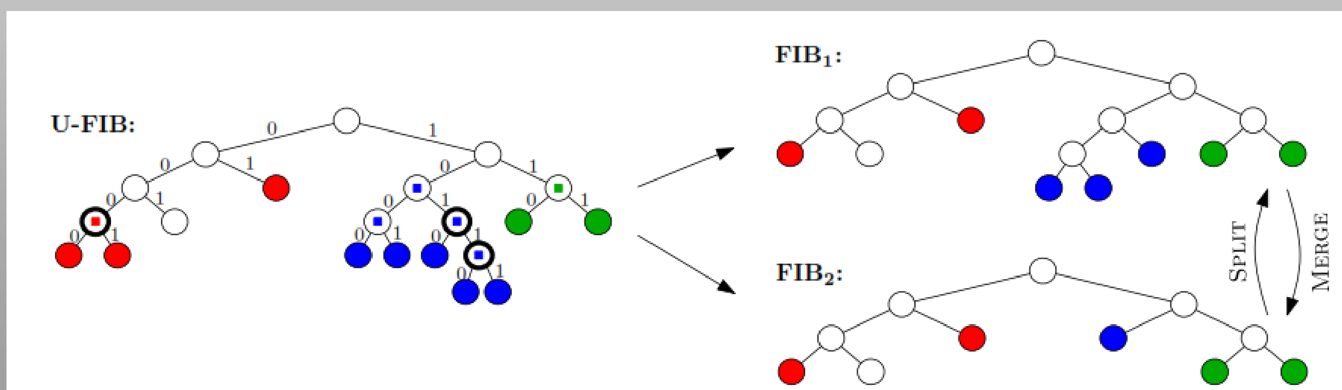
No need for complex predictions but still good!

Algorithm BLOCK(A,B)



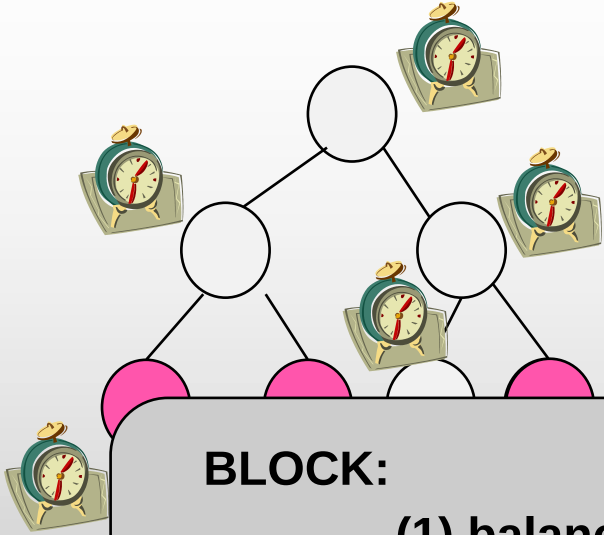
BLOCK(A,B) operates on trie:

- Two parameters A and B for amortization (A ≥ B)
- Definition: internal node v is c-mergeable if subtree T(v) only constains color c leaves
- Trie node v monitors: how long was subtree T(v) c-mergeable without interruption? Counter C(v).
- If C(v) ≥ A α, then aggregate entire tree T(u) where u is furthest ancestor of v with C(u) ≥ B α. (Maybe v is u.)
- Split lazily: only when forced.



Nodes with square inside: mergeable. Nodes with bold border: suppressed for FIB1.

Algorithm BLOCK(A,B)



BLOCK(A,B) operates on trie:

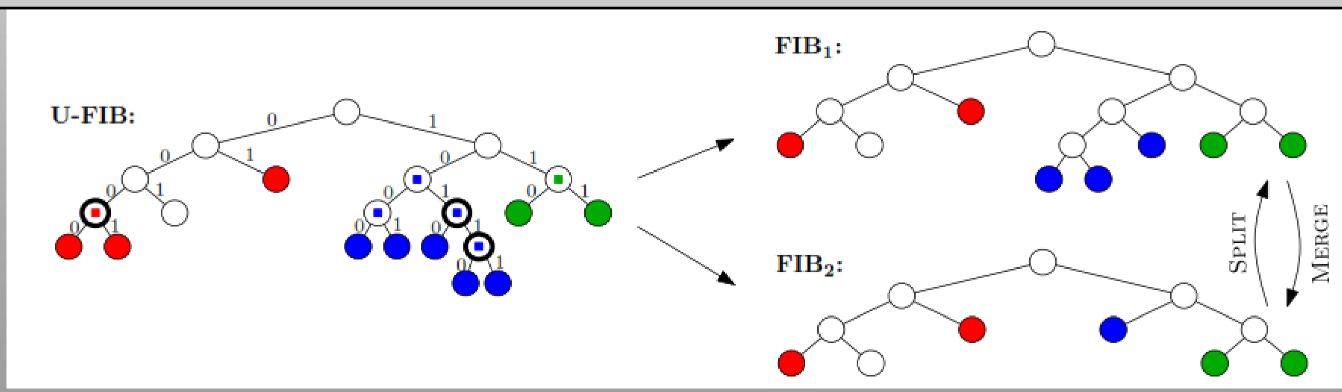
- Two parameters A and B for amortization (A ≥ B)
- Definition: internal node v is c-mergeable if subtree T(v) only constains color c leaves
- Trie node v monitors: how long was subtree T(v) c-mergeable without interruption? Counter C(v).

here
aybe

BLOCK:

(1) balances memory and update costs

(2) exploits possibility to merge multiple tree nodes simultaneously at lower price (threshold A and B)



Nodes with square inside: mergeable. Nodes with bold border: suppressed for FIB1.

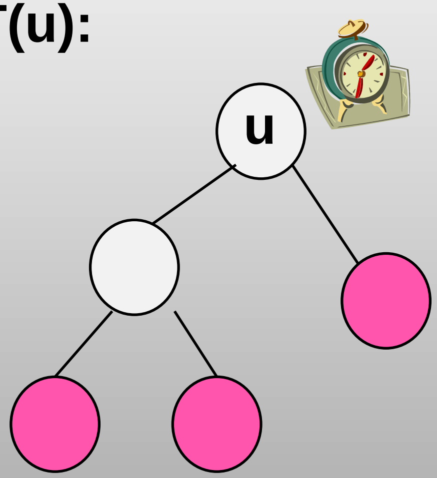
Analysis

Theorem: BLOCK(A,B) is 3.603-competitive.

Proof idea (a bit technical):

- Time events when ALG merges k nodes of T(u) at u
- **Upper bound ALG cost:**
  - k+1 counters between B α and A α
  - Merging cost at most (k+3) α: remove k+2 leaves, insert one root
  - Splitting cost at most (k+1) 3α: in worst case, remove-insert-remove individually
- **Lower bound OPT cost:**
  - Time period from t- α to t
  - If OPT does not merge anything in T(u) or higher: high memory costs
  - If OPT merges ancestor of u: counter there must be smaller than B α, memory and update costs
  - If OPT merges subtree of T(u): update cost and memory cost for in- and out-subtree
- Optimal choice: A = $\sqrt{13} - 1$, B = $(2\sqrt{13})/3 - 2/3$
- Add event costs (inserts/deletes) later!

T(u):



QED

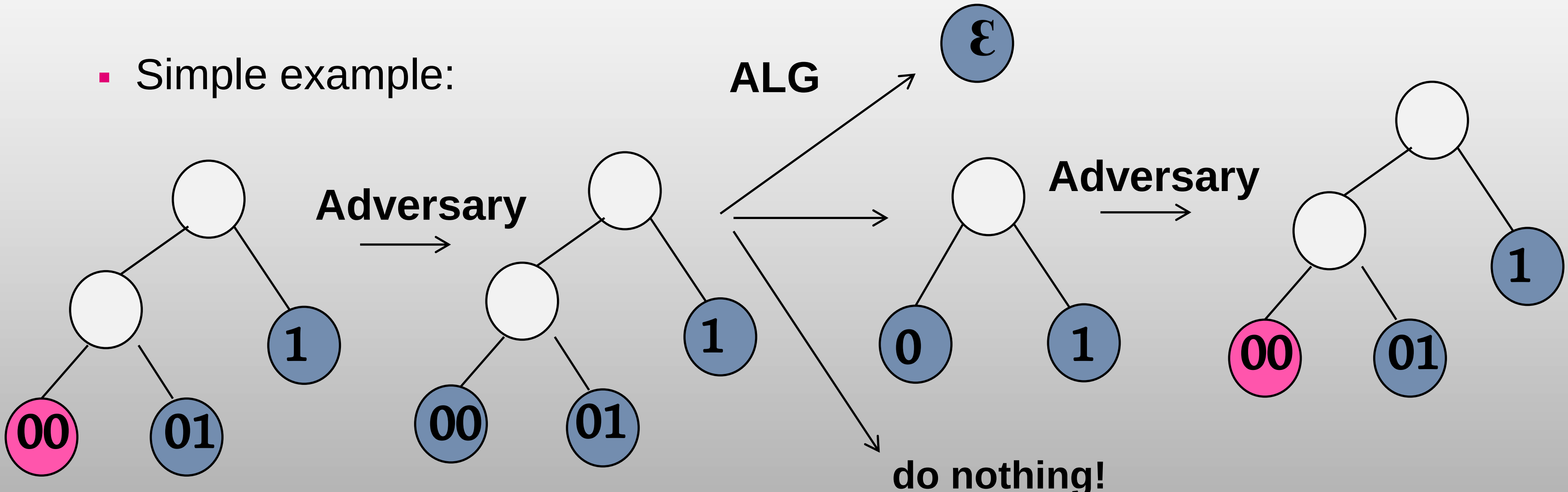
Lower Bound

Theorem:

Any online algorithm is at least 1.636-competitive.

Proof idea:

- Simple example:



(1) If ALG does never changes to single entry, competitive ratio is at least 2 (size 2 vs 1).

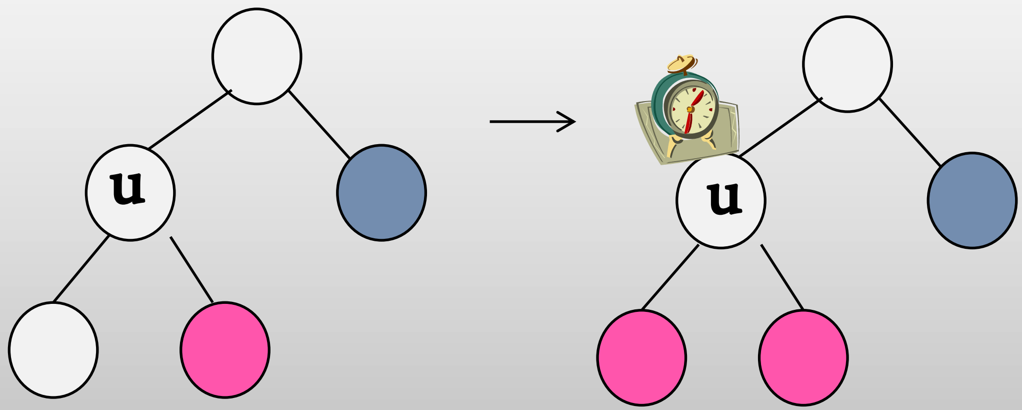
(2) If ALG changes before time α, adversary immediately forces split back! Yields costly inserts...

(3) If ALG changes after time α, the adversary resets color as soon as ALG for the first time has a single node. Waiting costs too high.

Note on Adding Insertions and Deletions

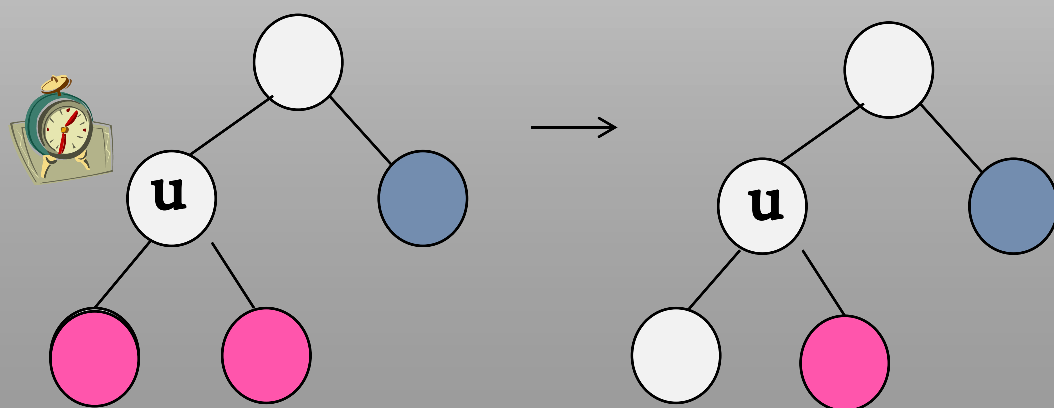
- Algorithm can be extended to insertions/deletions

Insert:



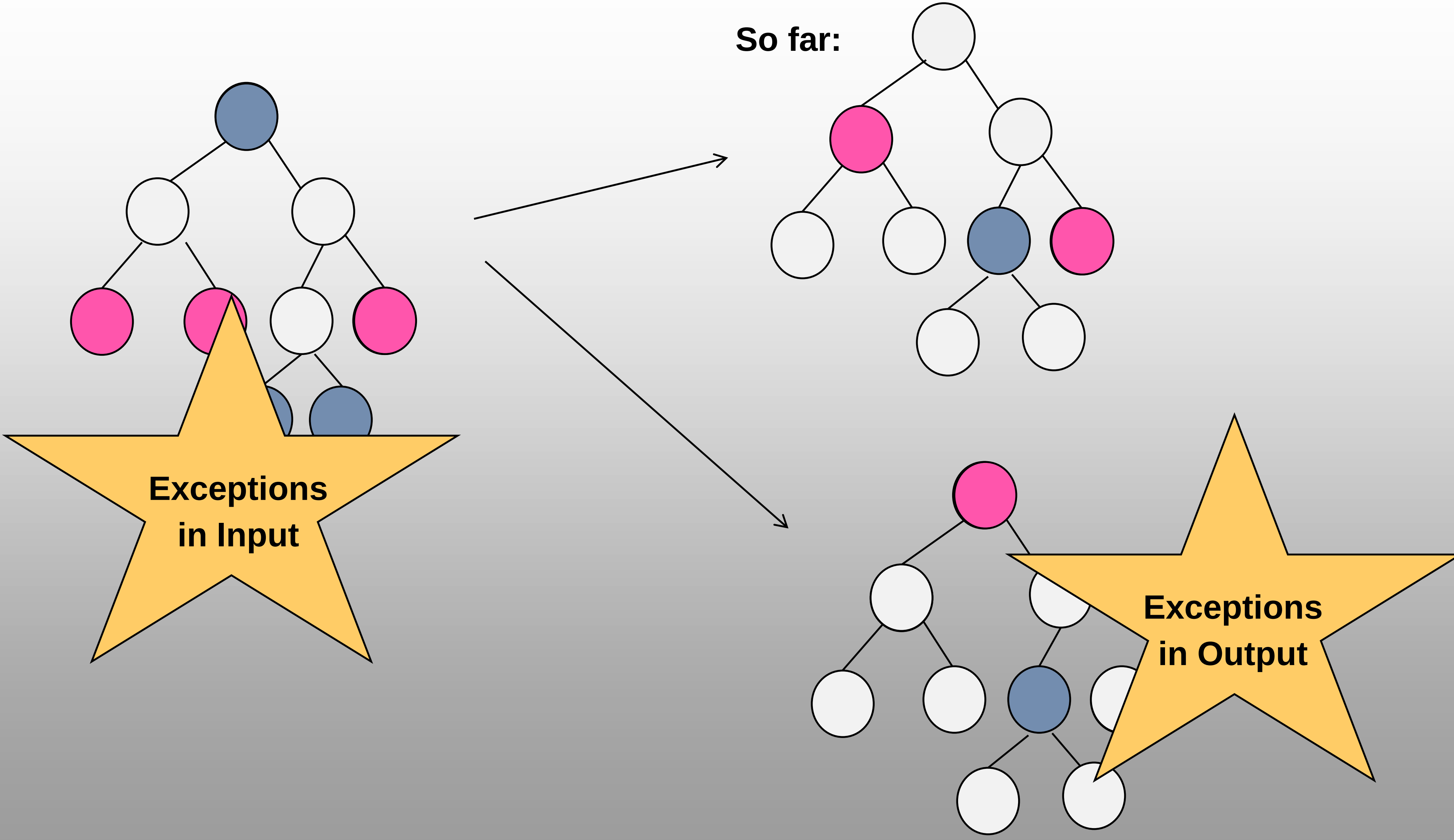
u becomes mergeable!

Delete:



u no longer mergeable!

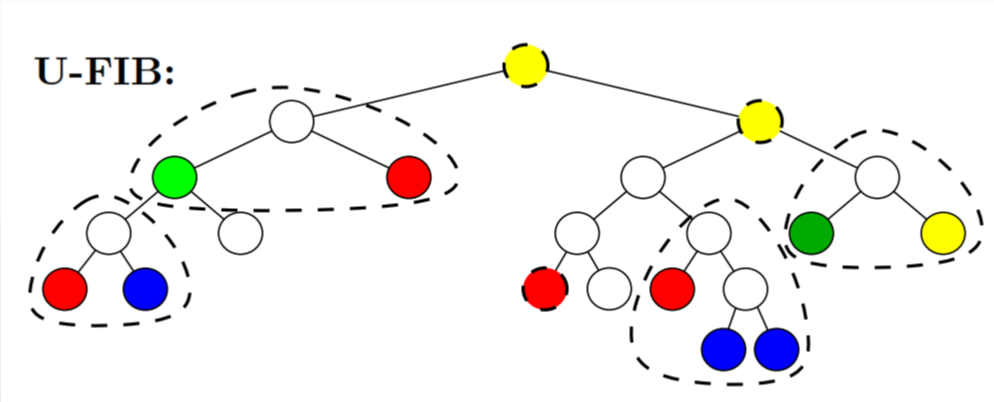
Allowing for Exceptions



Exceptions: Concepts and Definitions

Sticks

Maximal subtrees of UFIB with colored leaves and blank internal nodes.

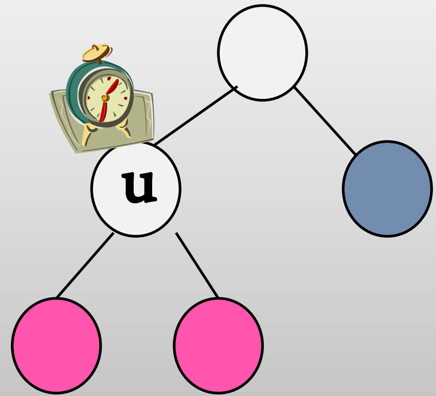


Idea: if all leaves in Stick have same color, they would become mergeable.

The HIMS Algorithm

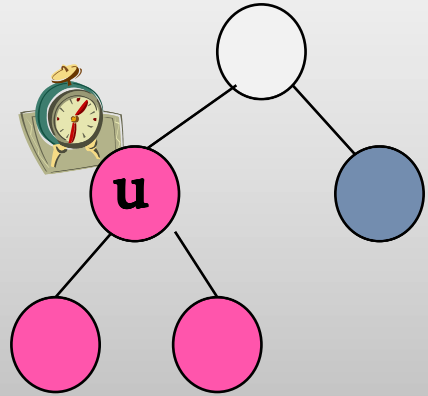
- Hide Invisibles Merge Siblings (HIMS)
- Two counters in Sticks:

Merge Sibling Counter:



C(u) = time since Stick descendants are unicolor

Hide Invisible Counter:



H(u) = how long do nodes have same color as the least colored ancestor?

Note: C(u) ≥ H(u), C(u) ≥ C(p(u)), H(u) ≥ H(p(u)), where p() is parent.

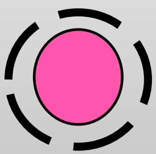
The HIMS Algorithm

Keep rule in FIB if and only if all three conditions hold:

(1) H(u) < α (do not hide yet)
(2) C(u) ≥ α or u is a stick leaf (do not aggregate yet if ancestor low)
(3) C(p(u)) < α or u is a stick root

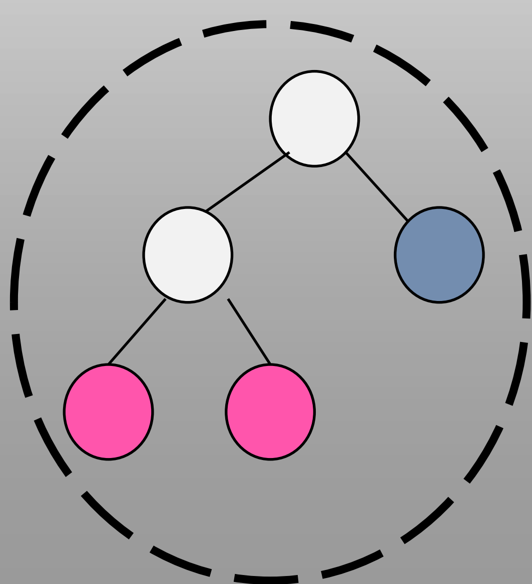
Examples:

Ex 1.



Trivial stick: node is both root and leaf (Conditions 2+3 fulfilled). So HIMS simply waits until invisible node can be hidden.

Ex 2.



Stick without colored ancestors: H(u)=0 all the time (Condition 1 fulfilled). So everything depends on counters inside stick. If counters large, only root stays.

Analysis

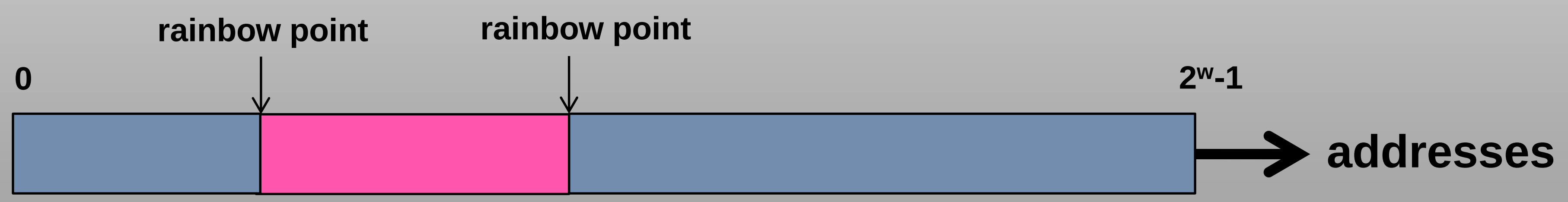
Theorem:

HIMS is O(w) -competitive.

Proof idea:

- In the absence of further BGP updates
  (1) HIMS does not introduce any changes after time α
  (2) After time α, the memory cost is at most an factor O(w) off

- In general: for any snapshot at time t, either HIMS already started aggregating or changes are quite new
- Concept of rainbow points and line coloring useful



- A rainbow point is a “witness” for a FIB rule
- Many different rainbow points over time give lower bound

Lower Bound

Theorem:

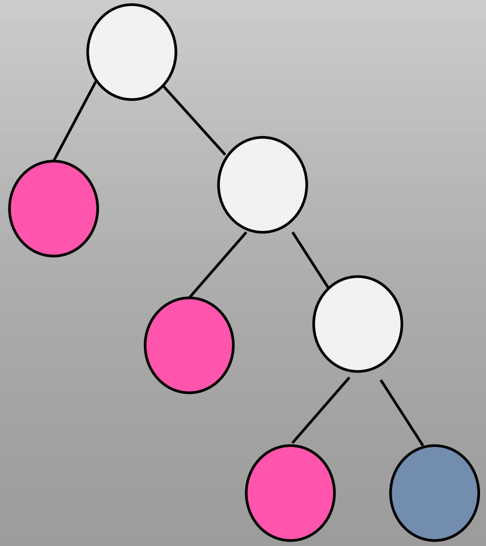
Any (online or offline) Stick-based algo is Ω(w) -competitive.

Proof idea:

Stick-based: (1) never keep a node outside a stick

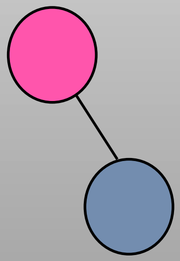
(2) inside a stick, for any pair u,v in ancestor-descendant relation, only keep one

Consider single stick: prefixes representing lengths 2^{w-1}, 2^{w-2}, ..., 2^1, 2^0, 2^0



Cannot aggregate stick!

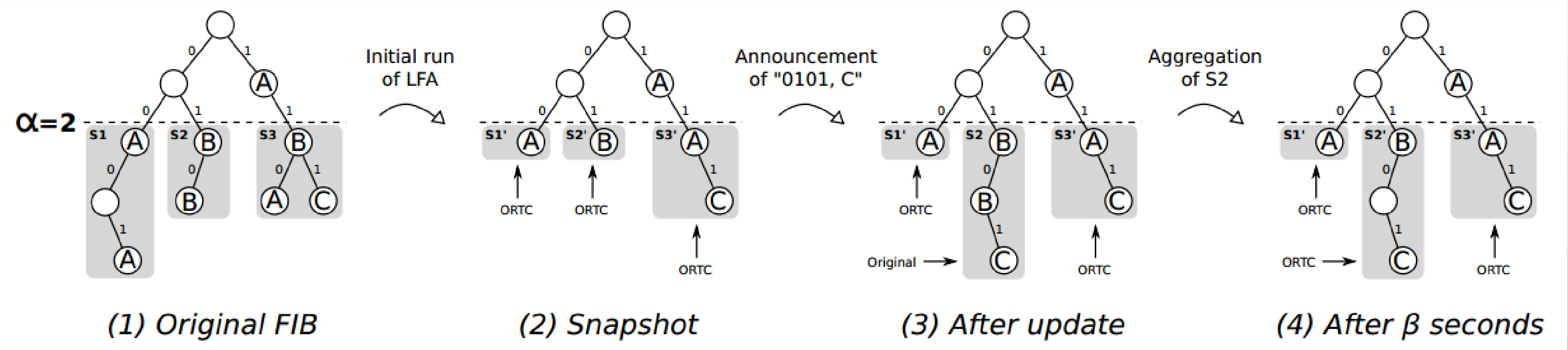
But OPT could use FIB:



QED

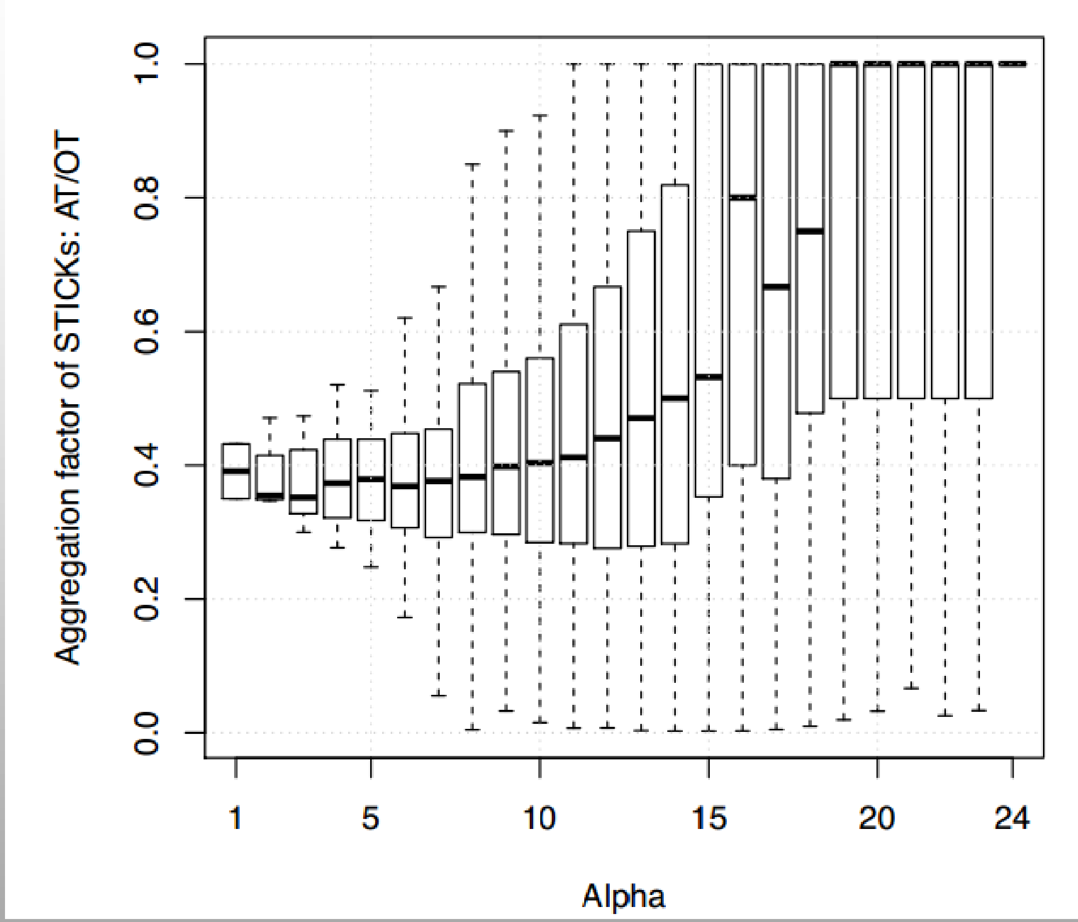
LFA: A Simplified Implementation

- LFA: Locality-aware FIB aggregation



- Combines stick aggregation with offline optimal ORTC
  - Parameter α: depth where aggregation starts
  - Parameter β: time until aggregation

LFA Simulation Results



For small alpha, Aggregated Table (AT) significantly smaller than Original Table (OT)

Conclusion

- Without exceptions in input and output: BLOCK is constant competitive
- With exceptions in input and output: HIMS is O(w)-competitive
- Note on offline variant: fixed parameter tractable, runtime of dynamic program in $f(\alpha)\, n^{O(1)}$

Thank you! Questions?