NTNU

Norwegian University of Science and Technology

LiteBTB: The Path to BTB Storage Efficiency Goes Through Tags

Roman K. Brunner, Rakesh Kumar
Computer Architecture Laboratory (CAL),
Norwegian University of Science and Technology (NTNU)

LiteBTB: What to Remember



Problem: The Frontend is a Major Performance Limiter

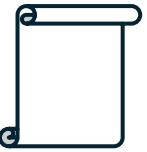
- Instruction working sizes overwhelm frontend structure
- BTB sizes keep growing at a rapid rate

LiteBTB: What to Remember



Problem: The Frontend is a Major Performance Limiter

- Instruction working sizes overwhelm frontend structure
- BTB sizes keep growing at a rapid rate



Prior Work: Improving BTB target representation

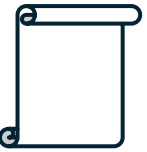
[PDede, MICRO'21. / BTB-X, HPCA'23]

LiteBTB: What to Remember



Problem: The Frontend is a Major Performance Limiter

- Instruction working sizes overwhelm frontend structure
- BTB sizes keep growing at a rapid rate



Prior Work: Improving BTB target representation

[PDede, MICRO'21. / BTB-X, HPCA'23]



Observation: Tags now consume most of BTB storage

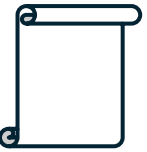
- Most tags stored in a BTB are duplicated

LiteBTB: What to Remember



Problem: The Frontend is a Major Performance Limiter

- Instruction working sizes overwhelm frontend structure
- BTB sizes keep growing at a rapid rate



Prior Work: Improving BTB target representation

[PDede, MICRO'21. / BTB-X, HPCA'23]



Observation: Tags now consume most of BTB storage

- Most tags stored in a BTB are duplicated



Idea: Store each unique tag only once

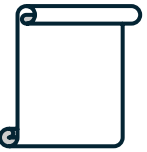
- Store mapping between tags and targets
- Lookup tags and mapping in parallel to keep access time constant

LiteBTB: What to Remember



Problem: The Frontend is a Major Performance Limiter

- Instruction working sizes overwhelm frontend structure
- BTB sizes keep growing at a rapid rate



Prior Work: Improving BTB target representation

[PDede, MICRO'21. / BTB-X, HPCA'23]



Observation: Tags now consume most of BTB storage

- Most tags stored in a BTB are duplicated

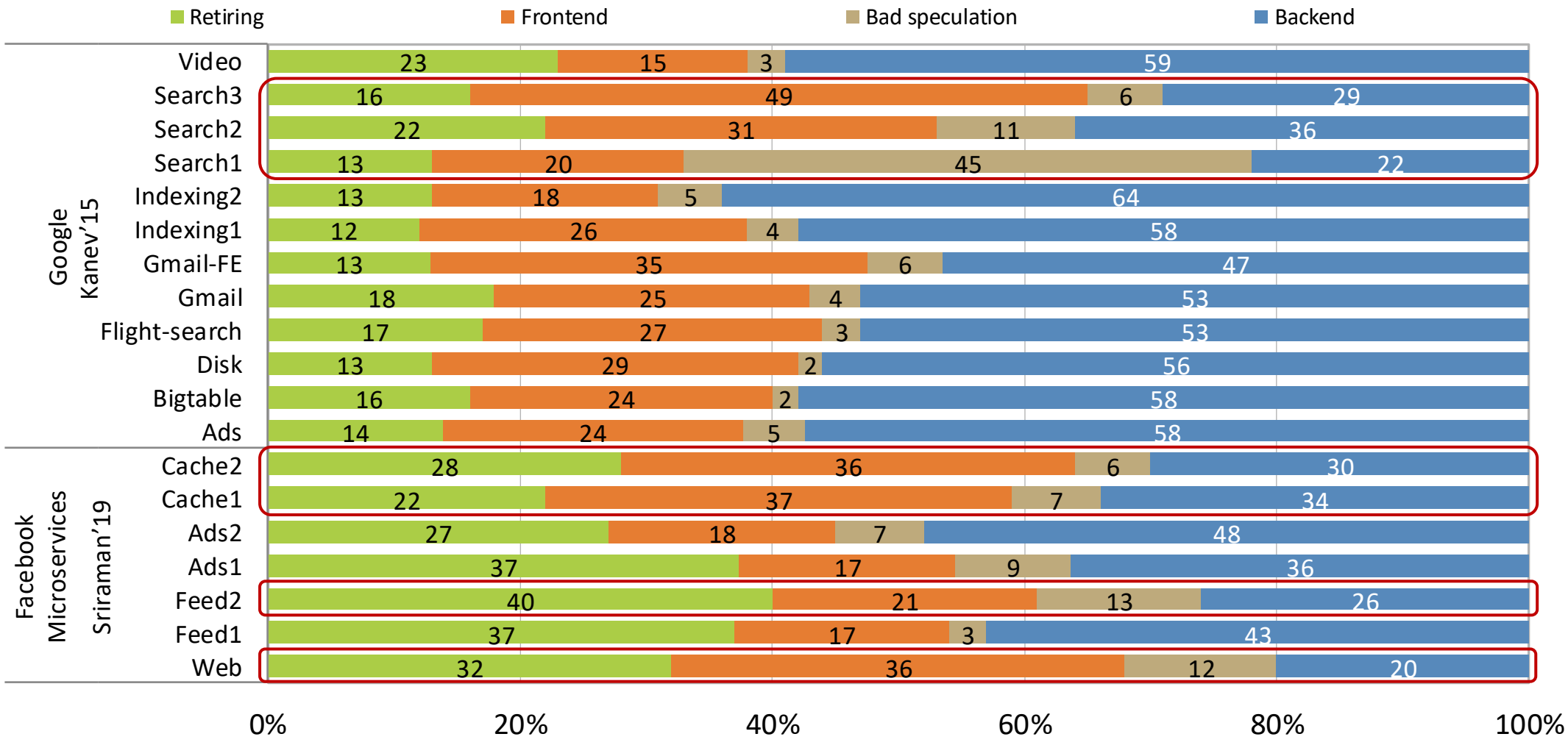


Idea: Store each unique tag only once

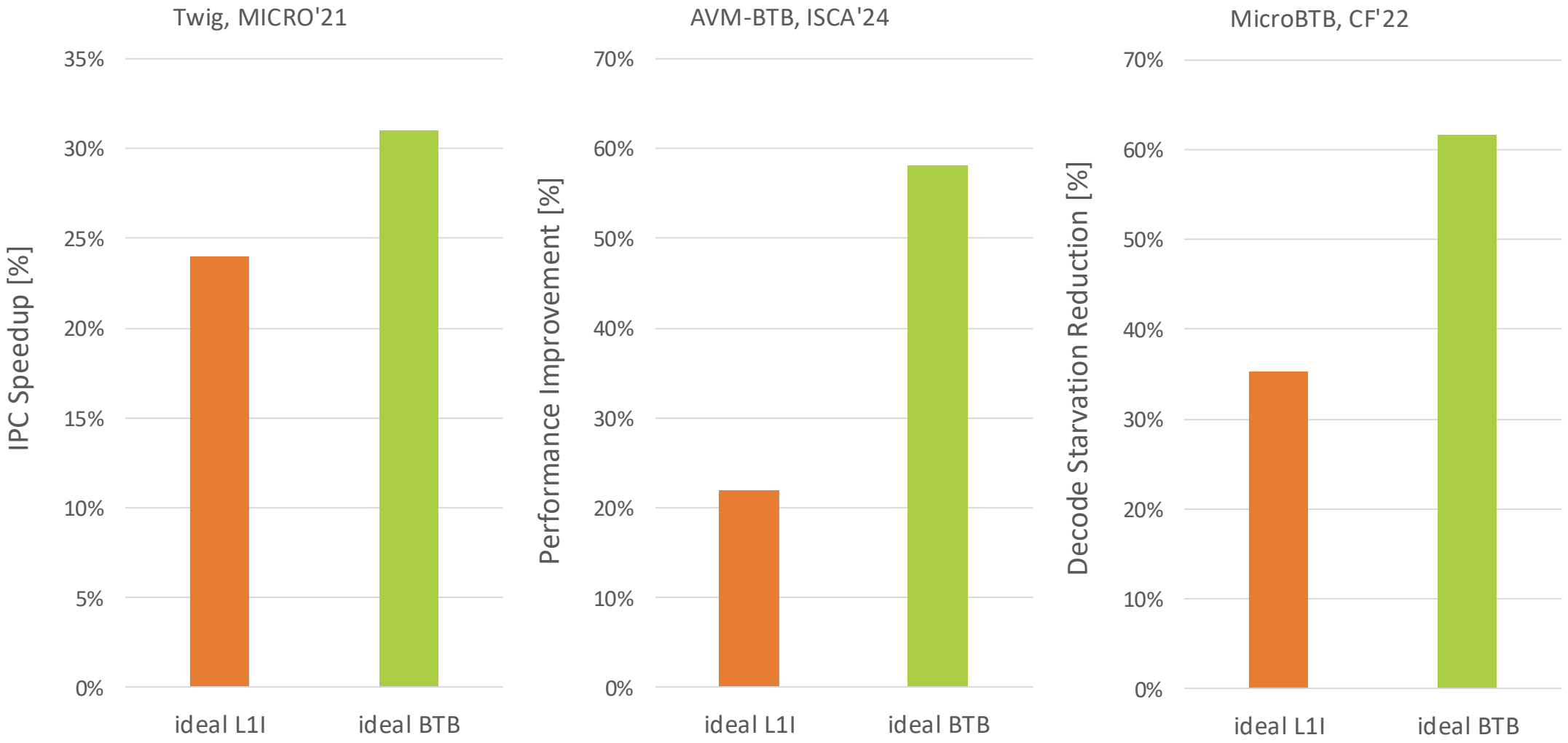
- Store mapping between tags and targets
- Lookup tags and mapping in parallel to keep access time constant

LiteBTB achieves comparable performance with up to 13.1% less storage

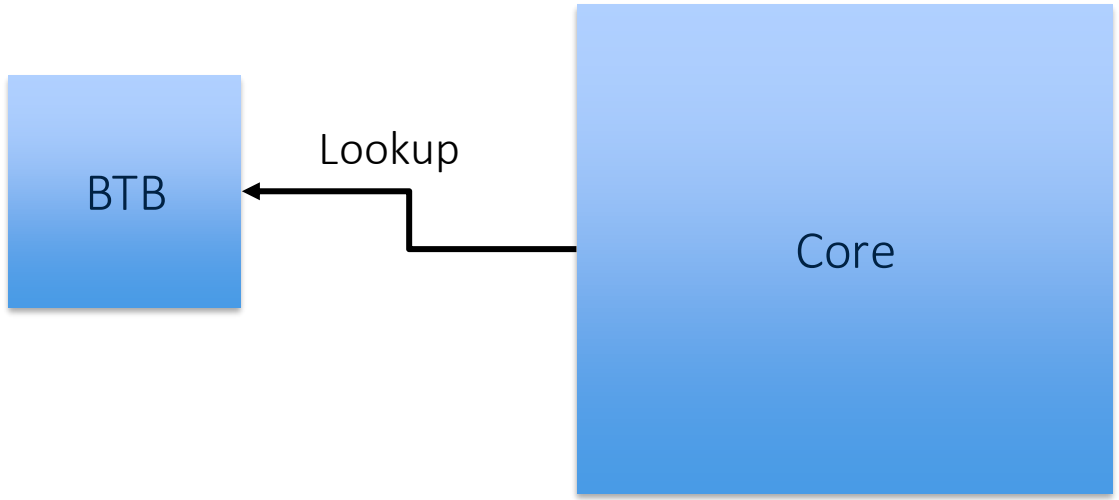
The Frontend Bottleneck



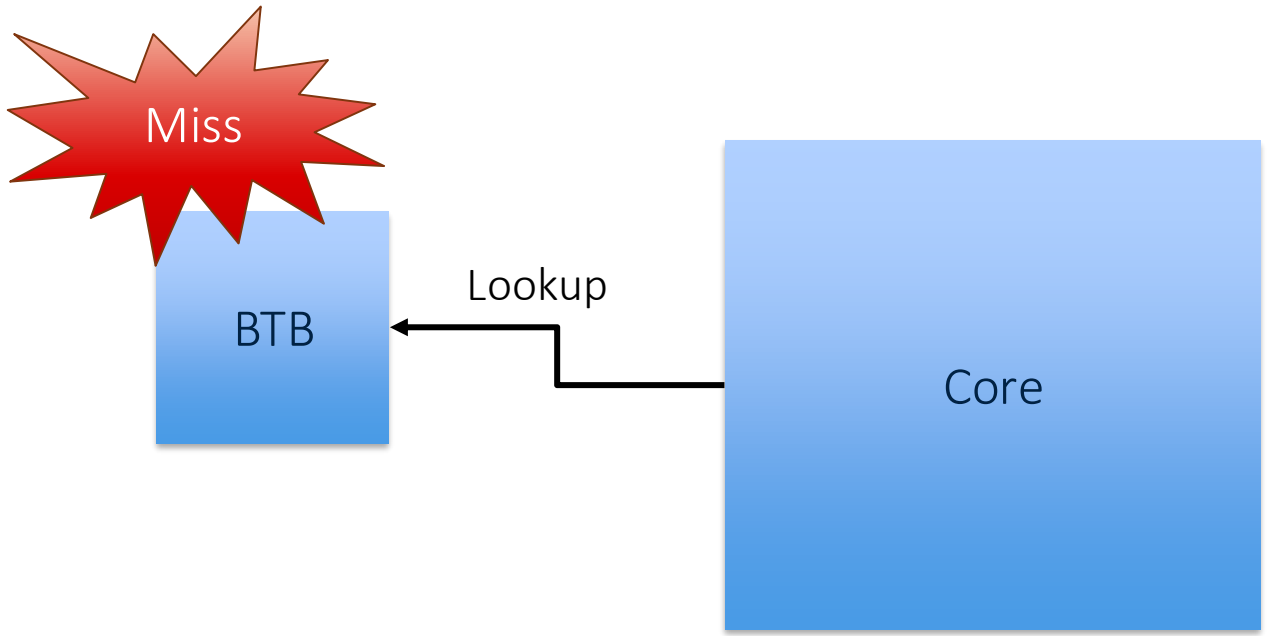
BTB's Contribution to the Frontend Bottleneck



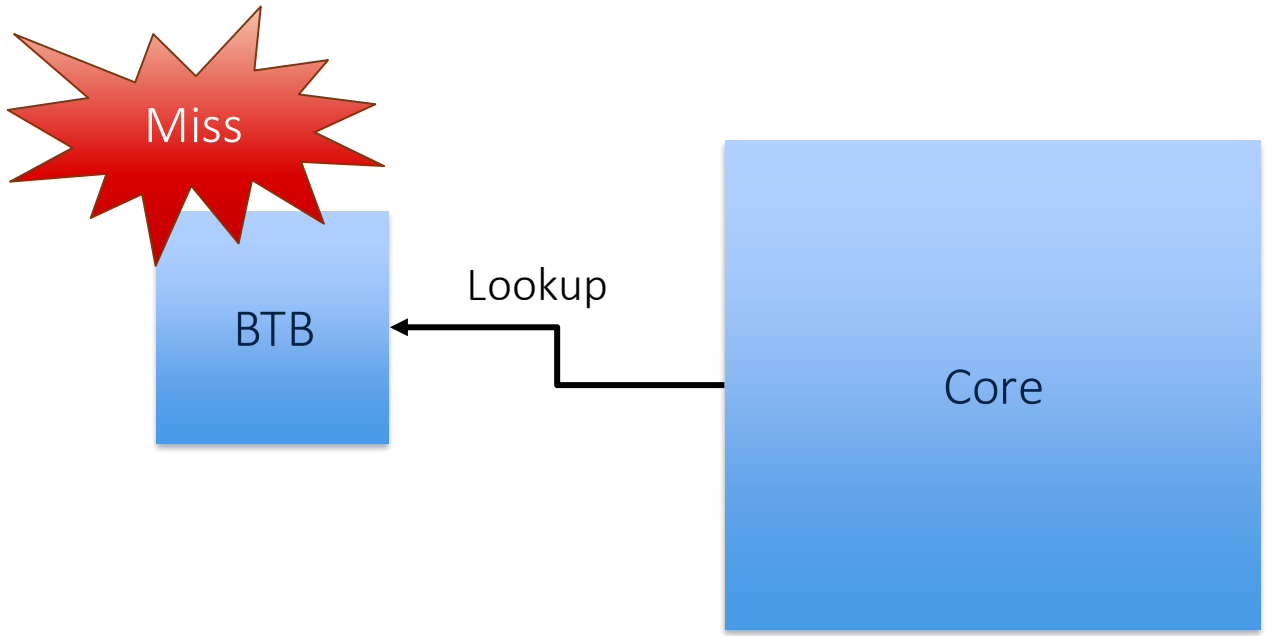
BTB's Contribution to the Frontend Bottleneck



BTB's Contribution to the Frontend Bottleneck

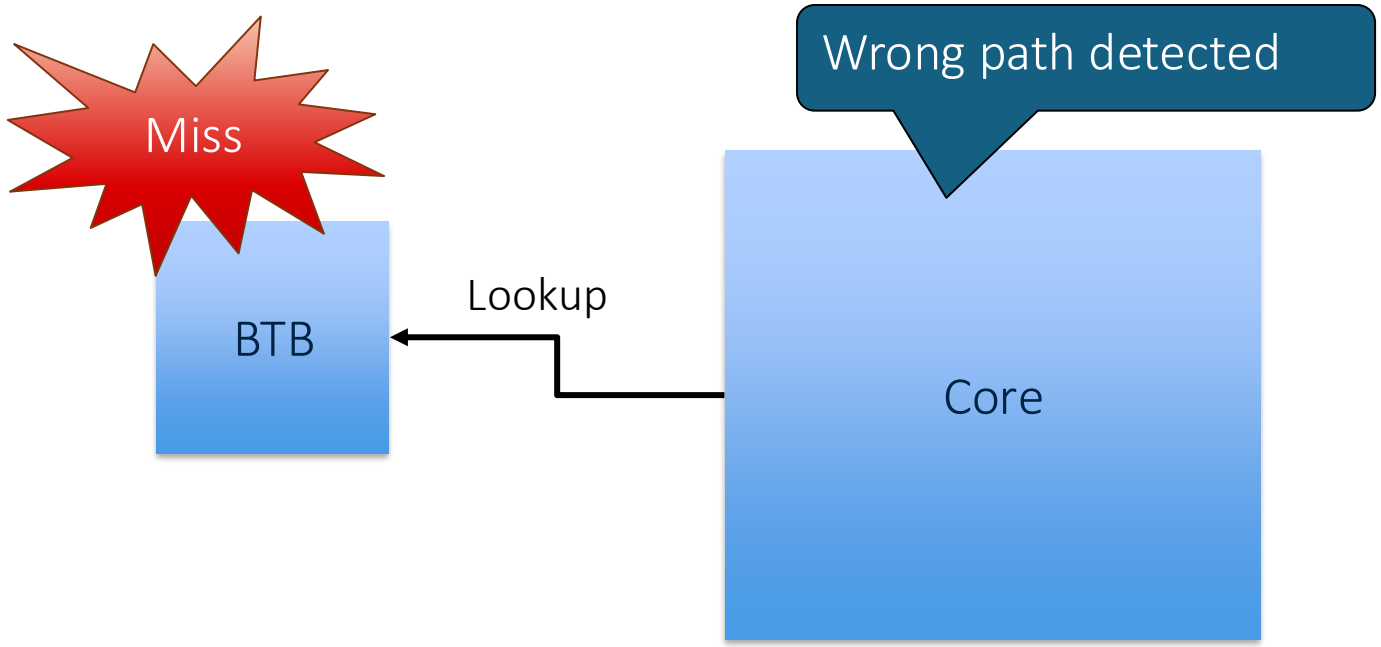


BTB's Contribution to the Frontend Bottleneck



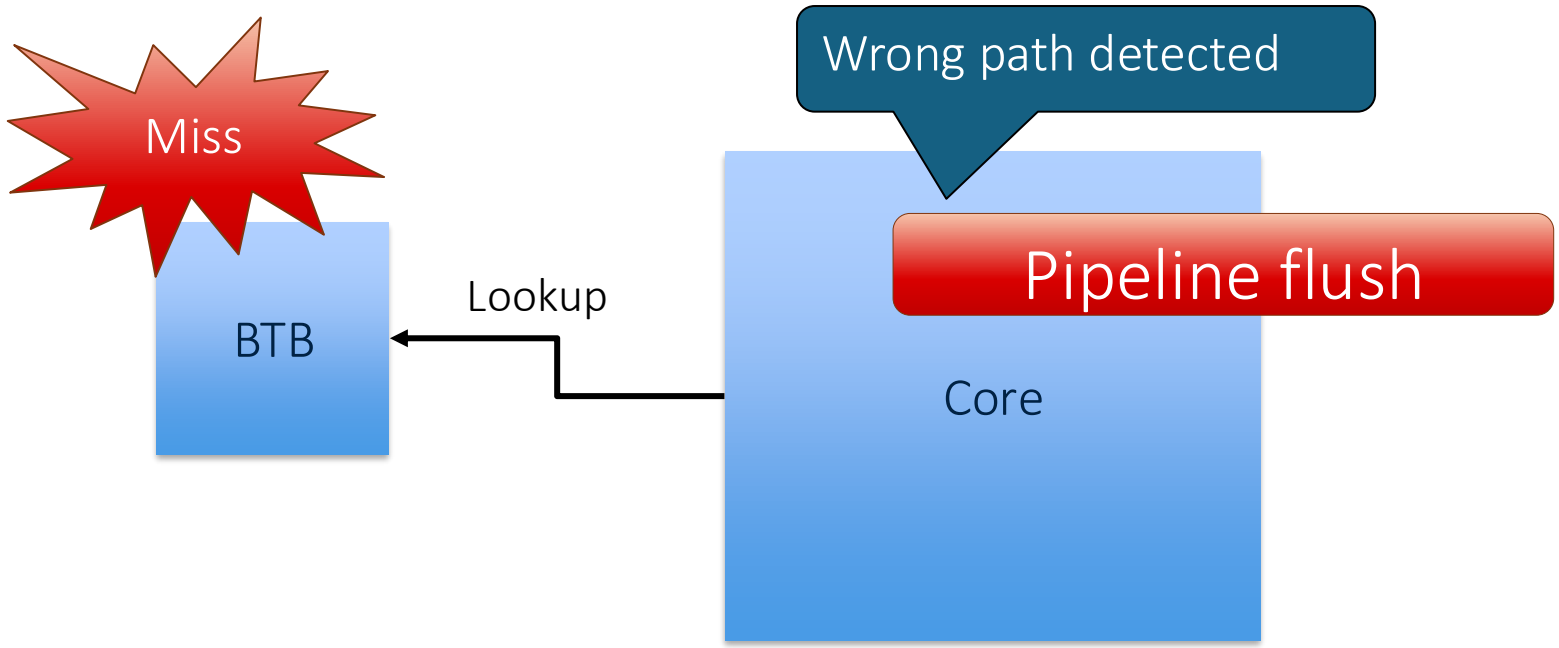
Execution cycles: 10+

BTB's Contribution to the Frontend Bottleneck



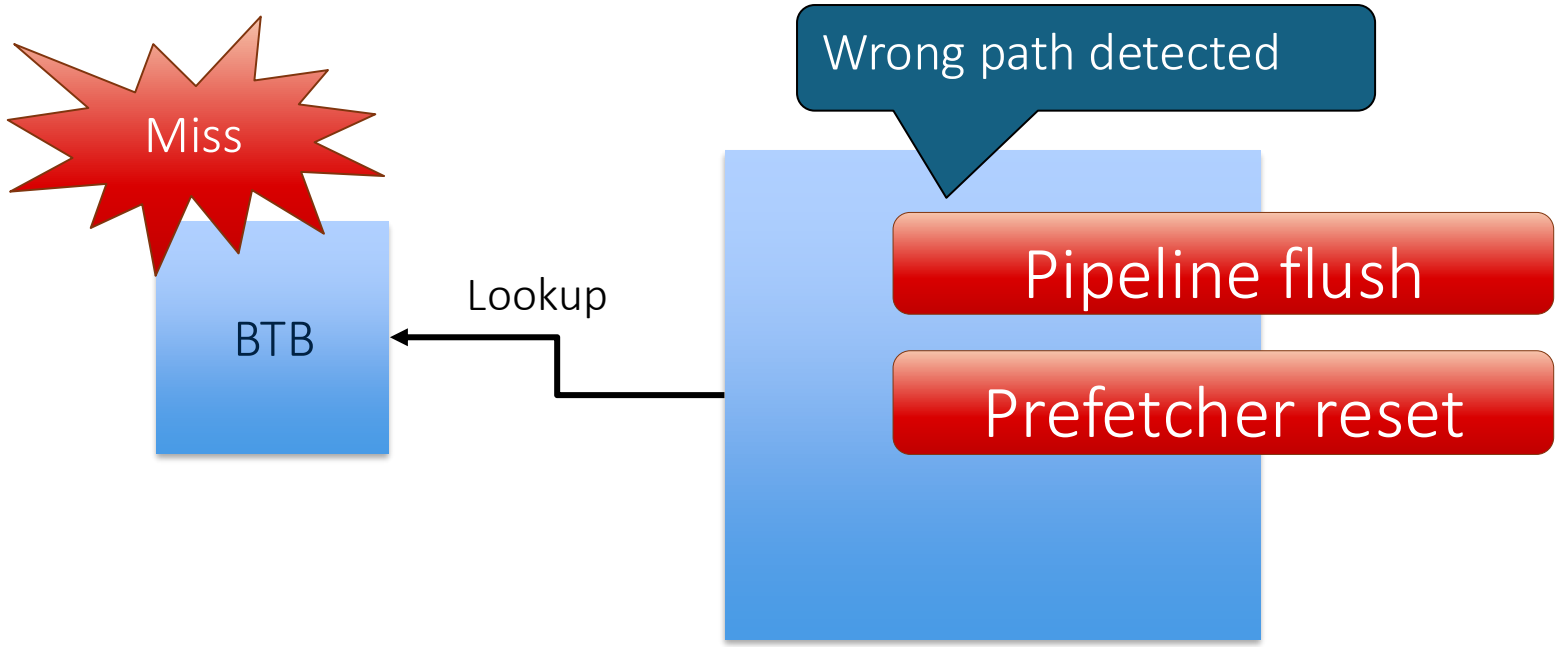
Execution cycles: 10+

BTB's Contribution to the Frontend Bottleneck



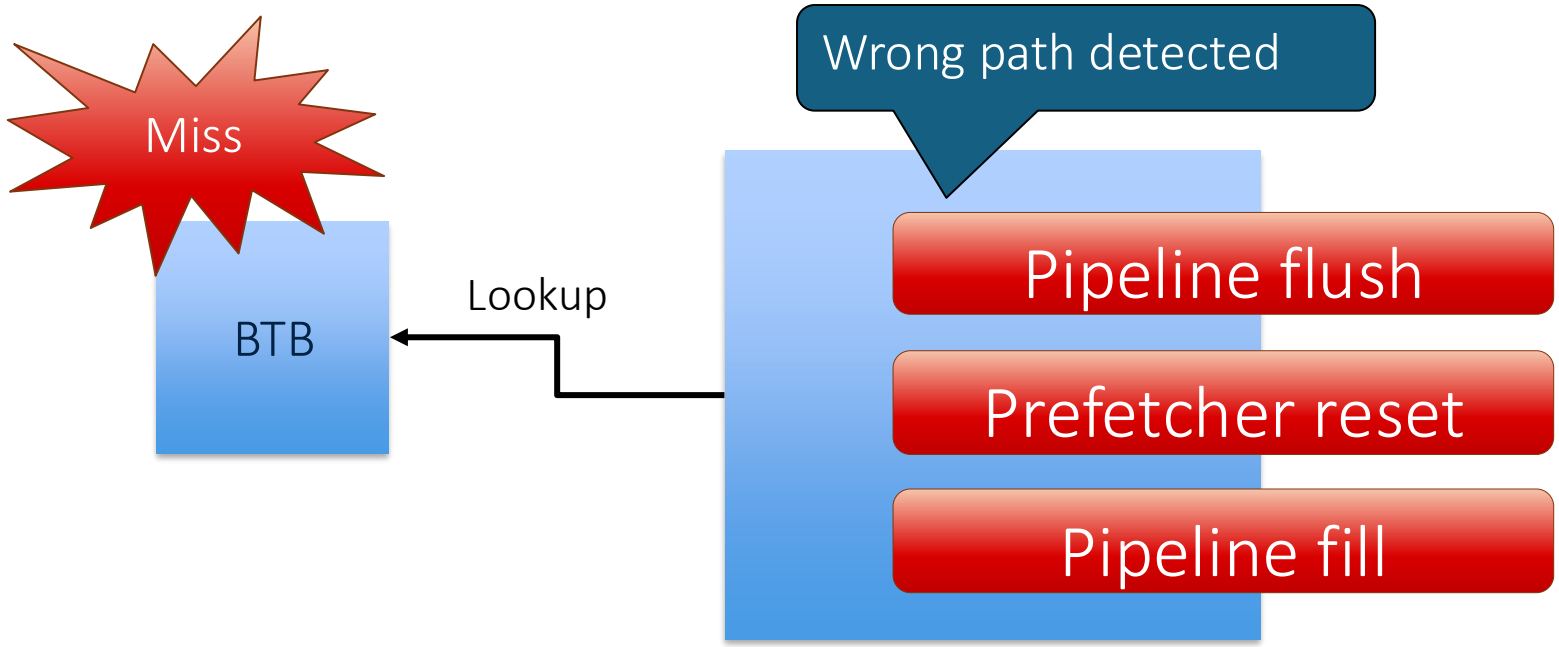
Execution cycles: 10+

BTB's Contribution to the Frontend Bottleneck



Execution cycles: 10+

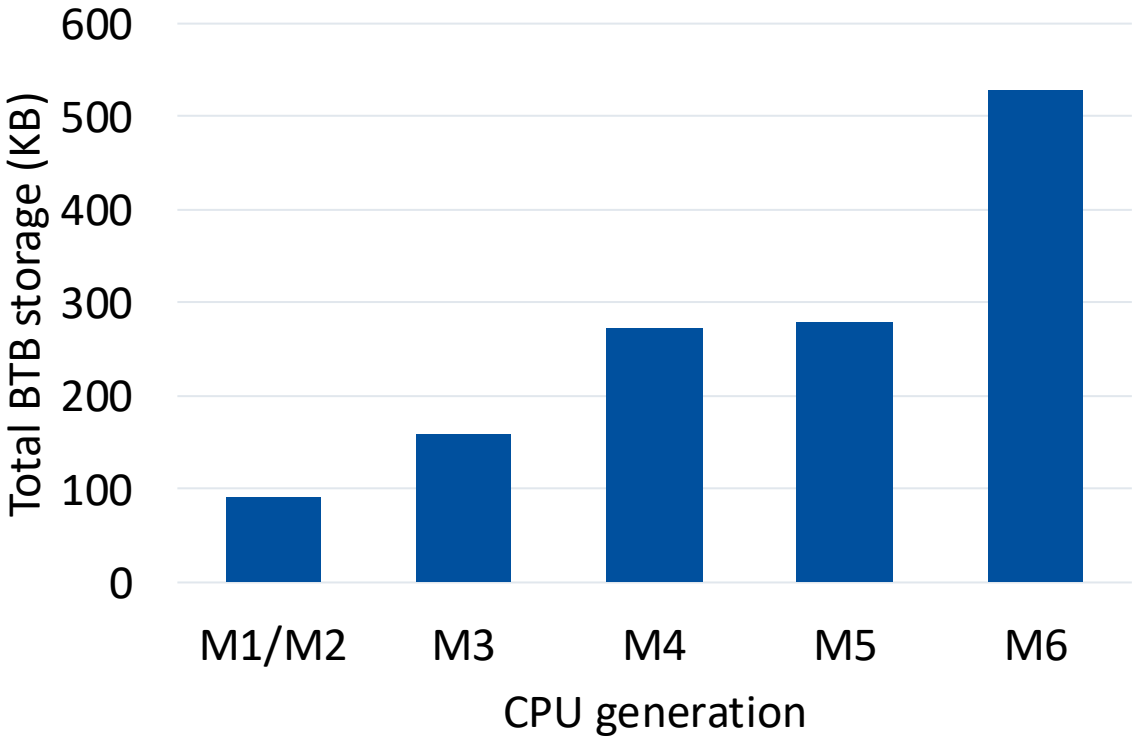
BTB's Contribution to the Frontend Bottleneck



Pipeline fill latency: 10+ cycles

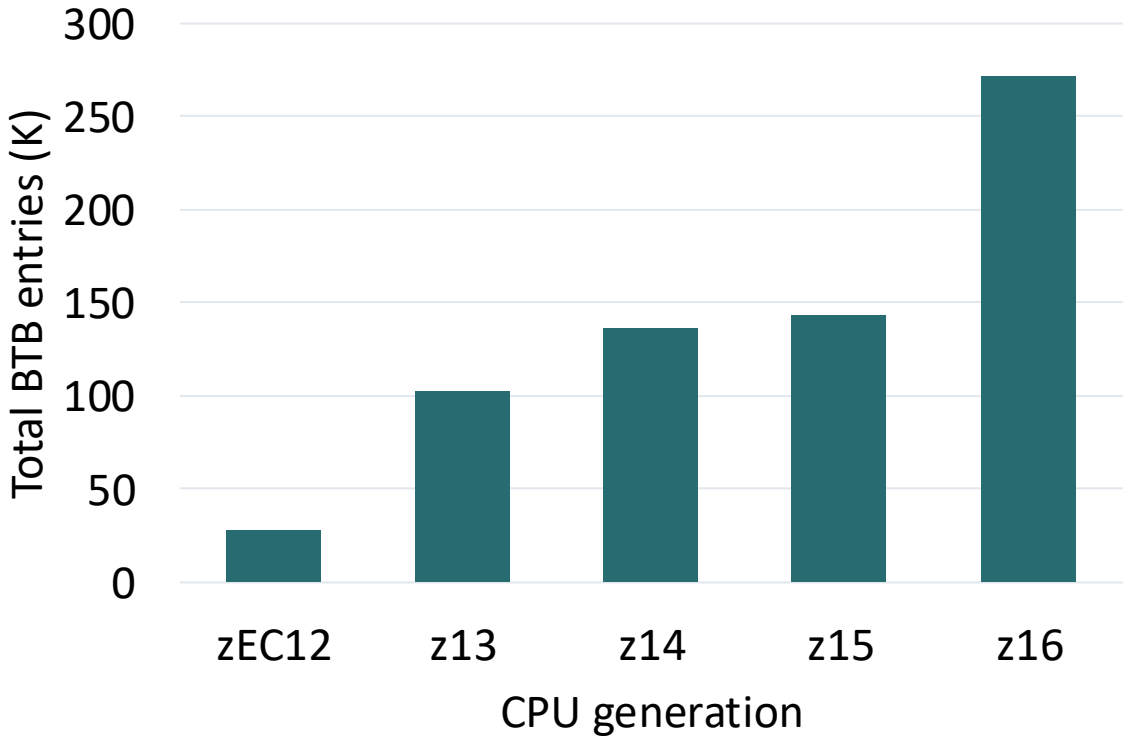
BTBs Keep Growing

Samsung Exynos



5.8× storage

IBM zSeries

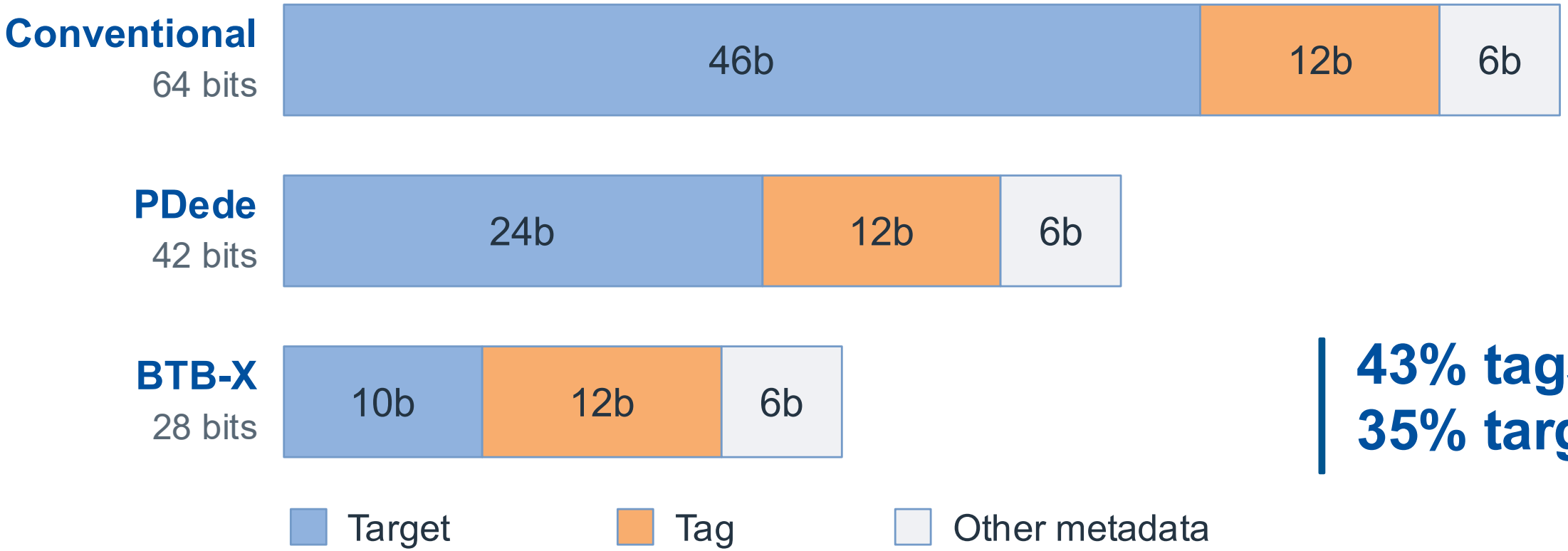


9.7× entries

Grayson et al., ISCA 2020.

Adiga et al., ISCA 2020.

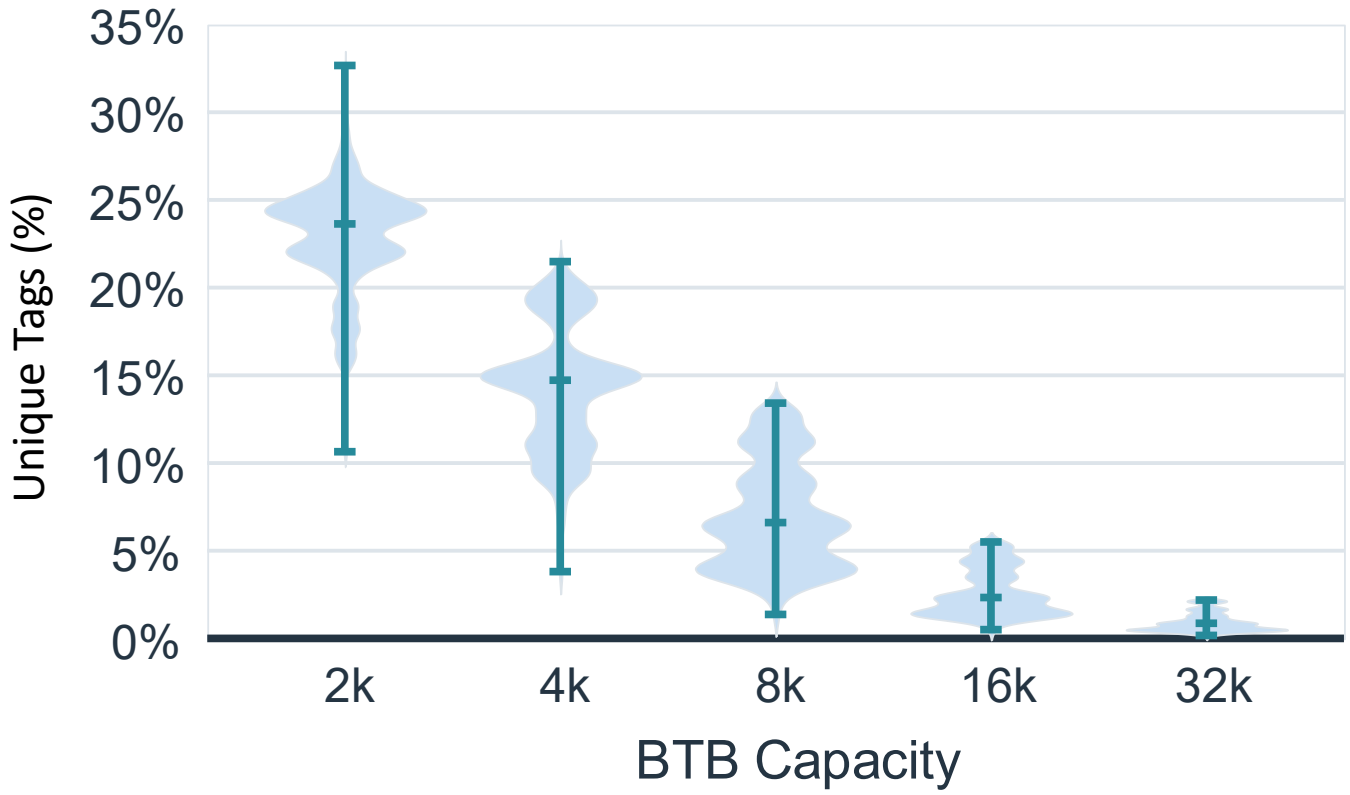
Tags Dominate BTB Storage



43% tags
35% targets

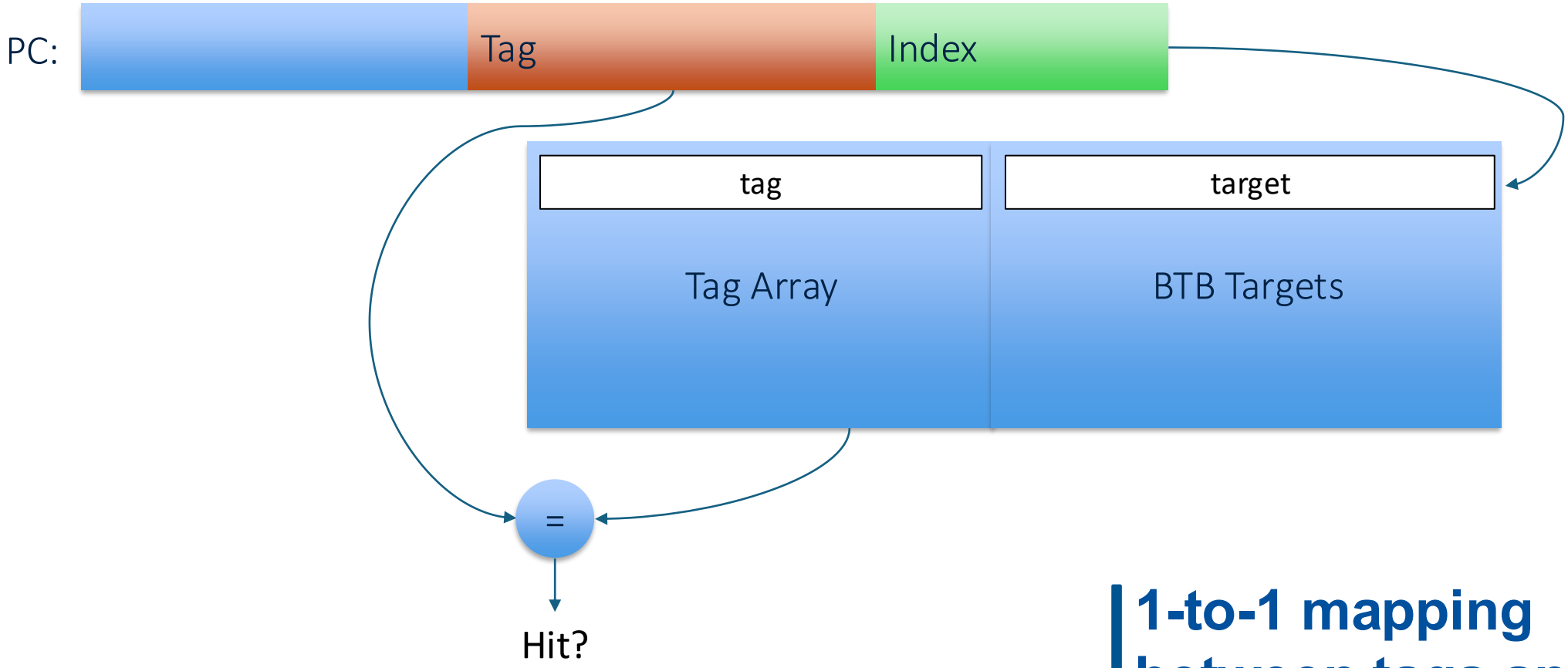
PDede: Soundararajan et al., MICRO 2021.
BTB-X: Asheim et al., HPCA 2023.

BTBs Duplicate Tags



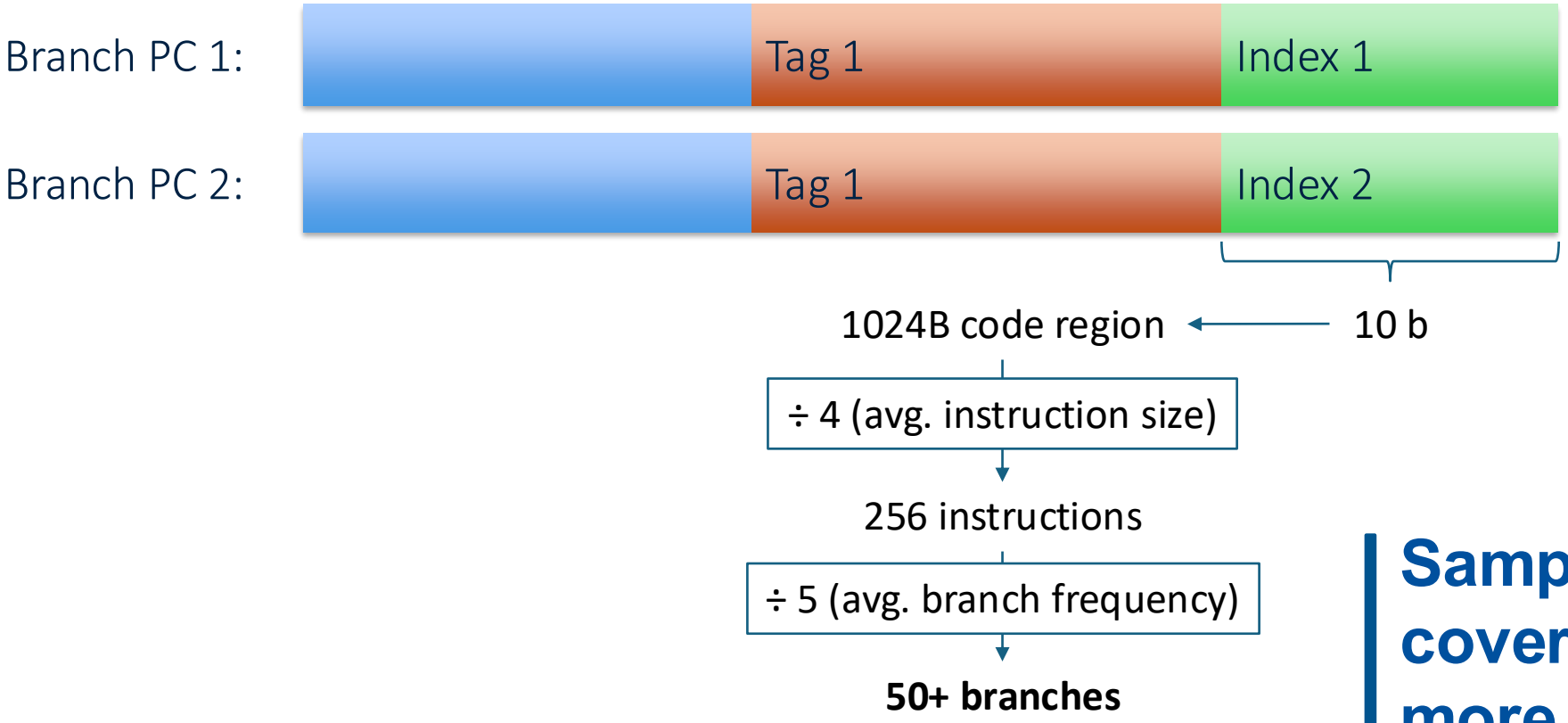
Deduplicating tags reduces tag storage requirements

BTB Operation



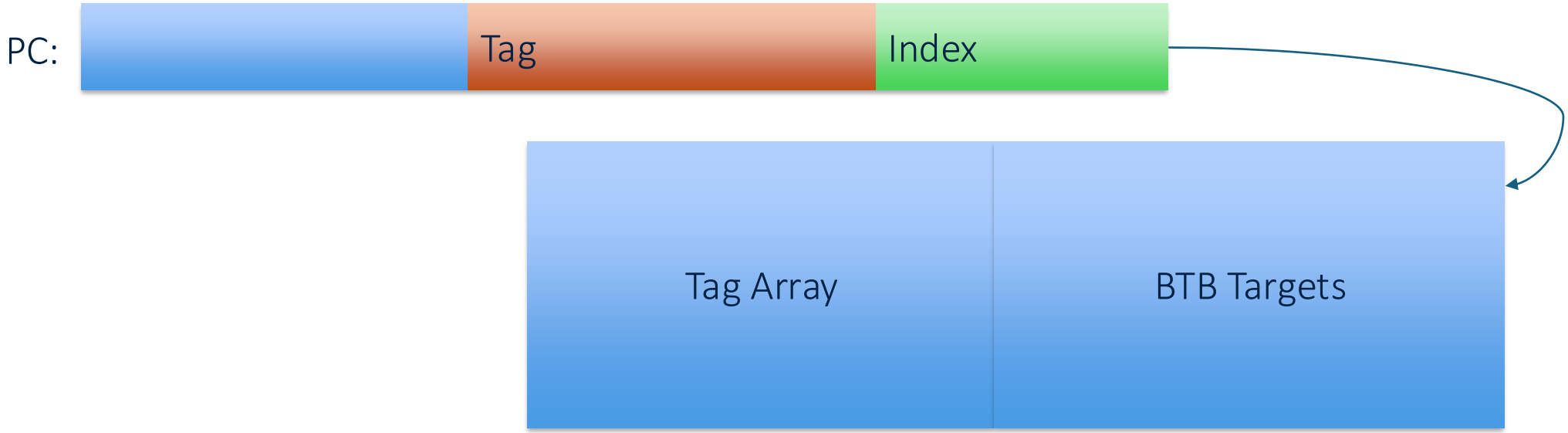
1-to-1 mapping between tags and targets

Why Are Tags Duplicated?

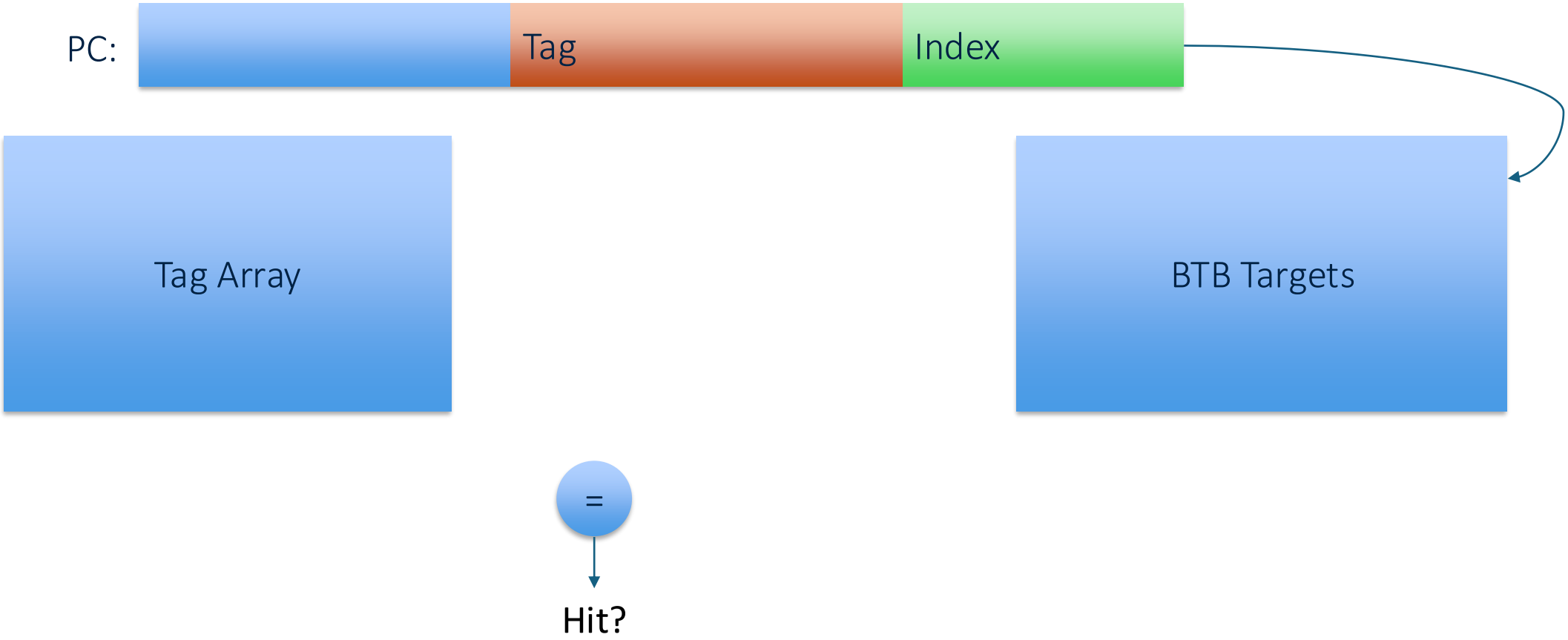


Sample: One tag covers on average more than 50 branches

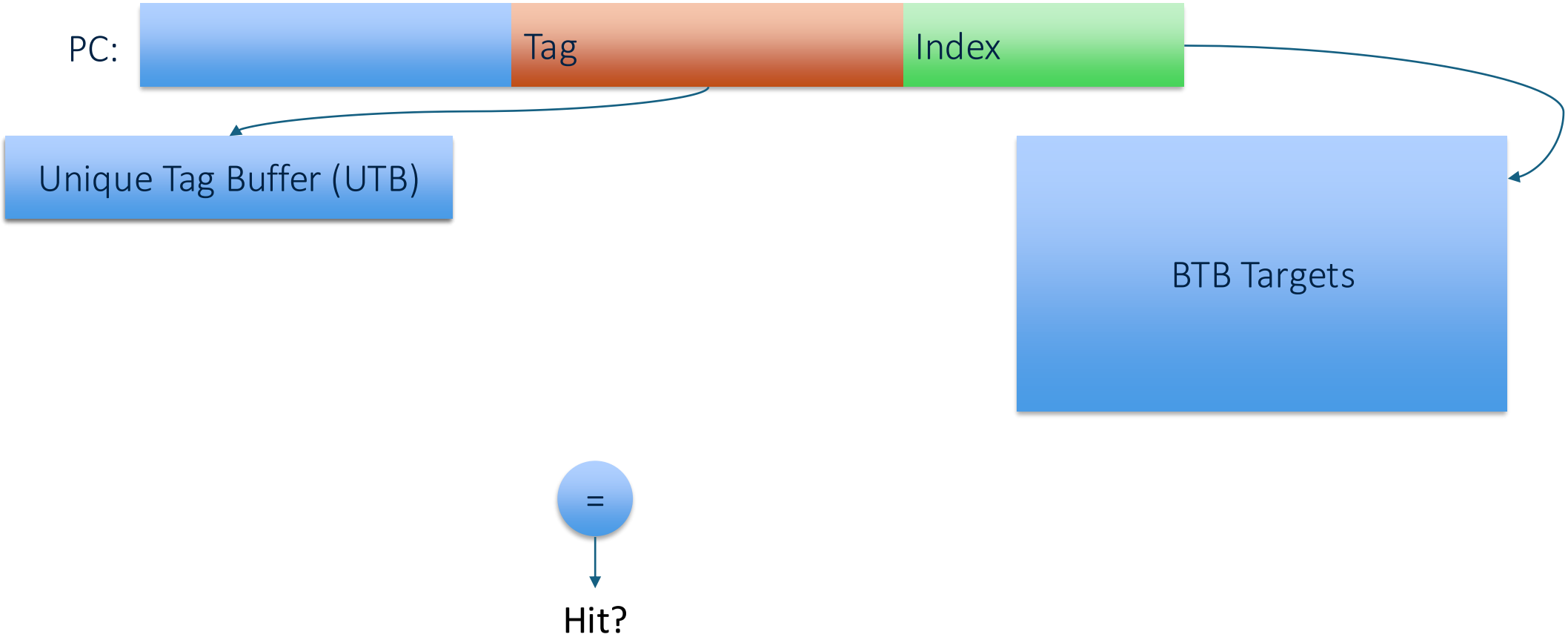
Deduplicating Tags



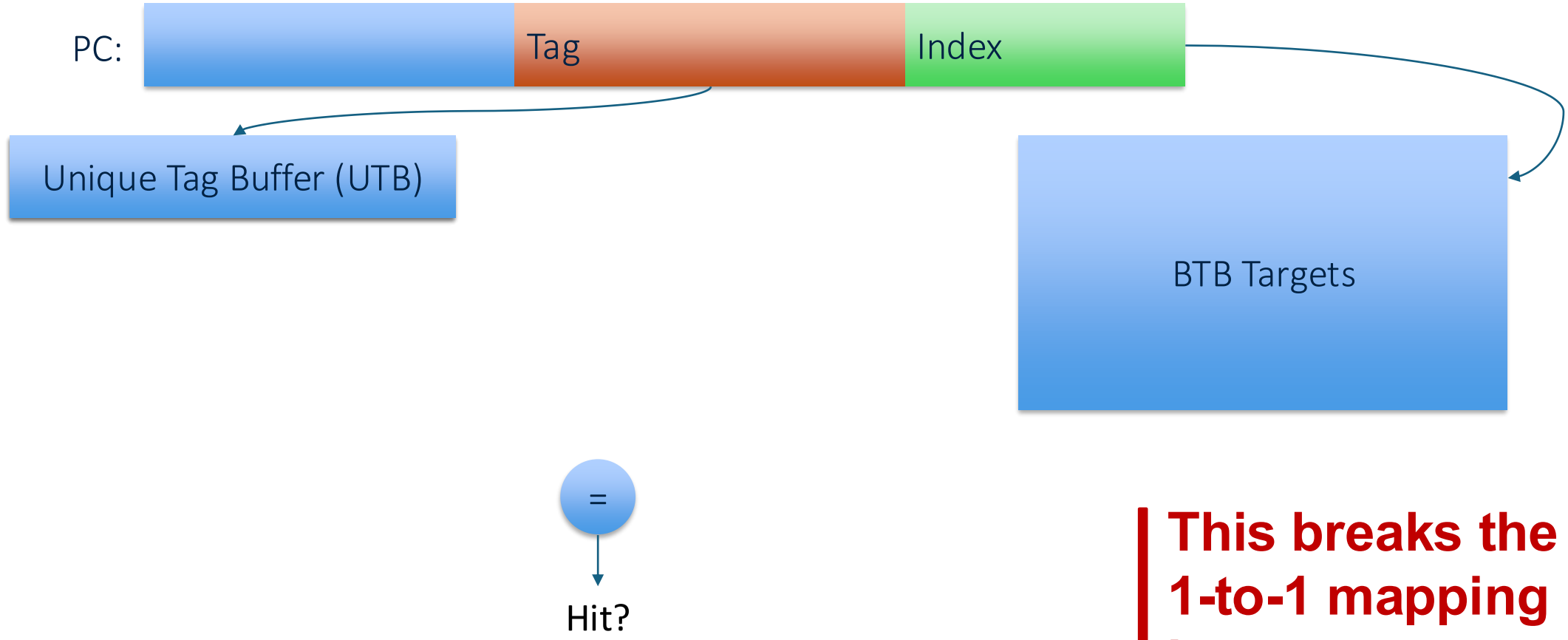
Deduplicating Tags



Deduplicating Tags

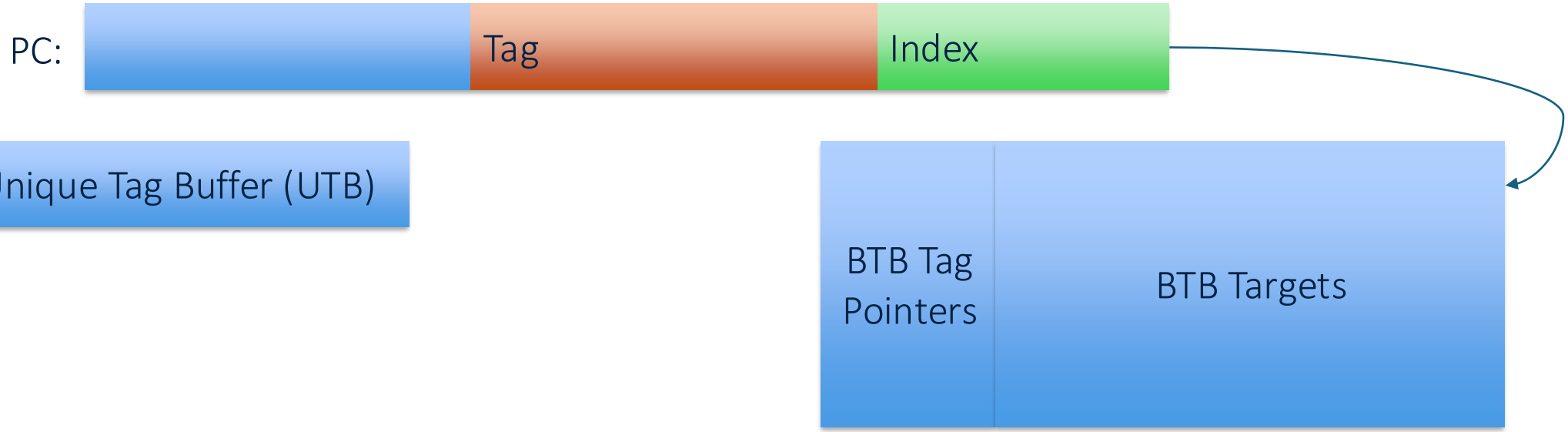


Deduplicating Tags

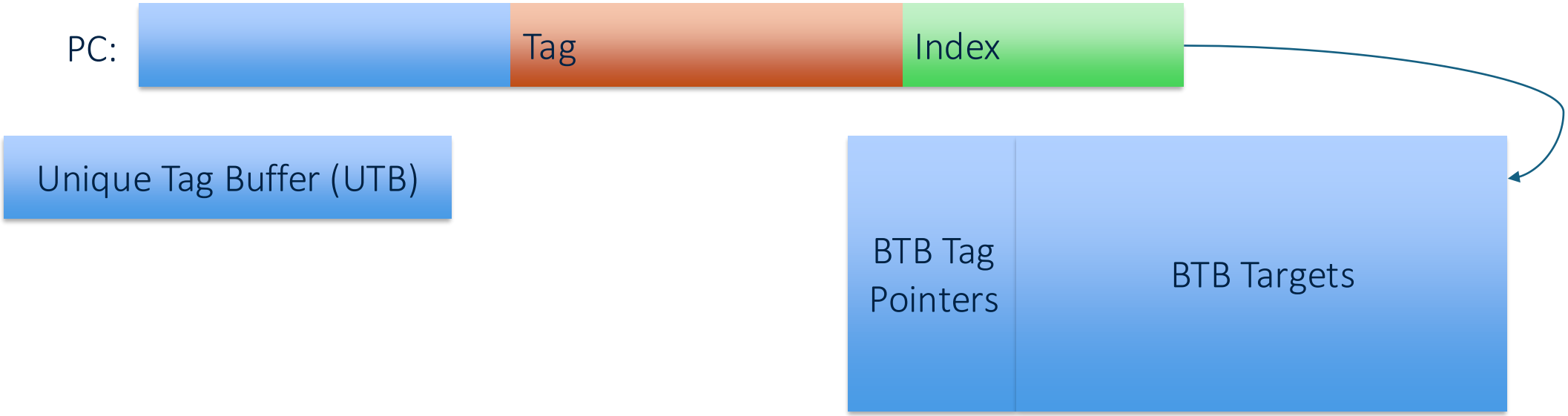


This breaks the 1-to-1 mapping between tags and targets

Deduplicating Tags



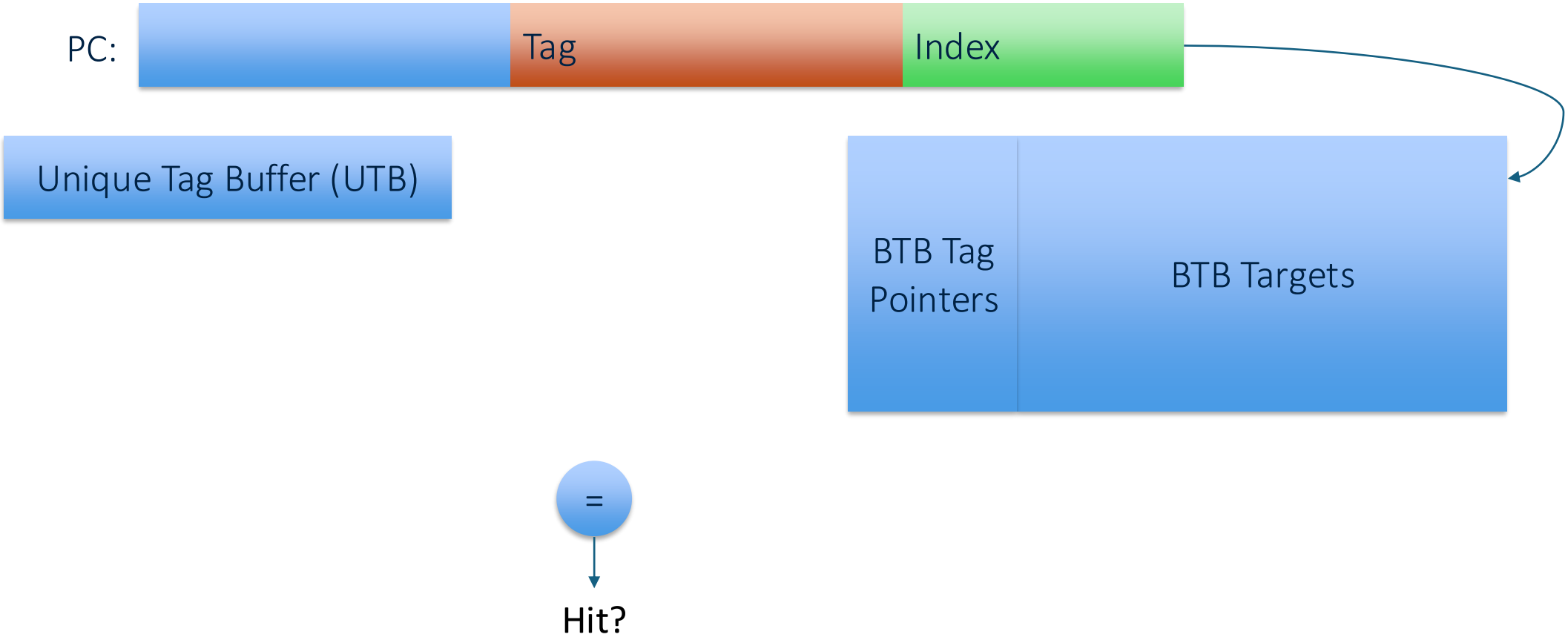
Deduplicating Tags



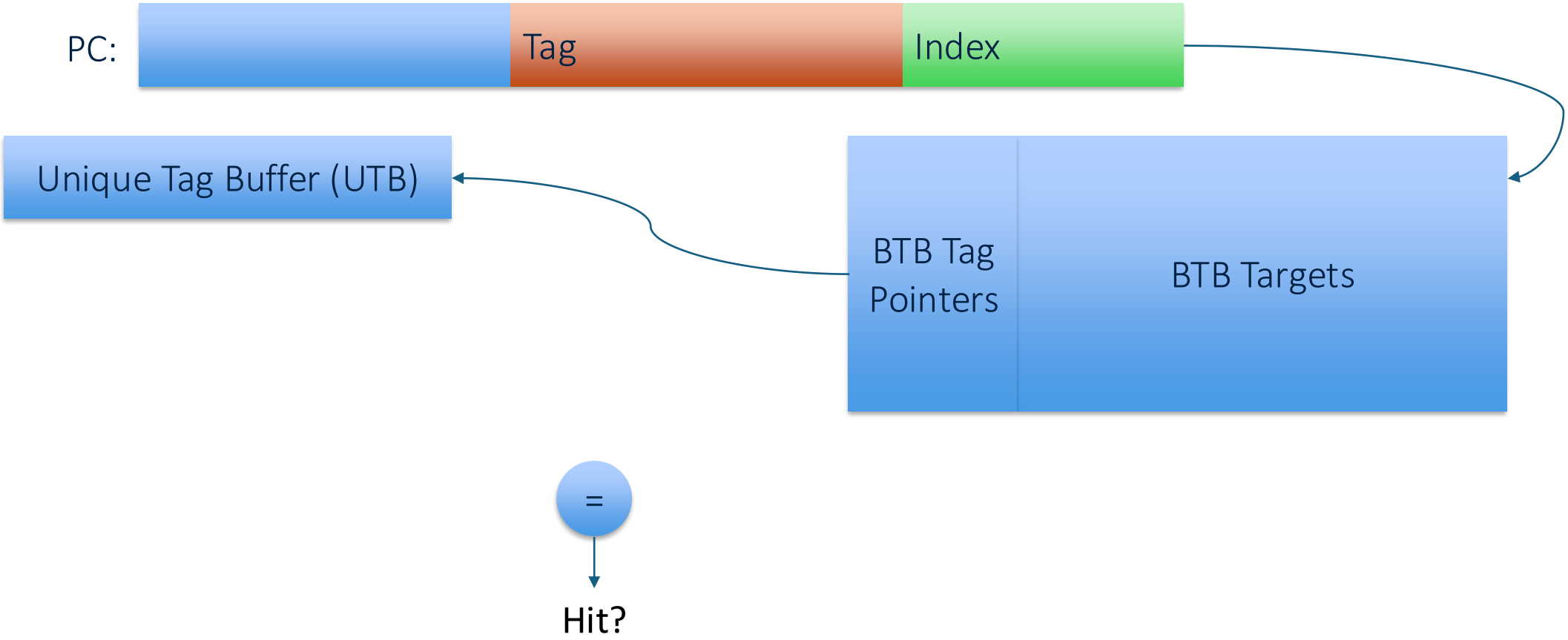
LiteBTB saves storage:

BTB tag pointers are short since they point to few unique tags

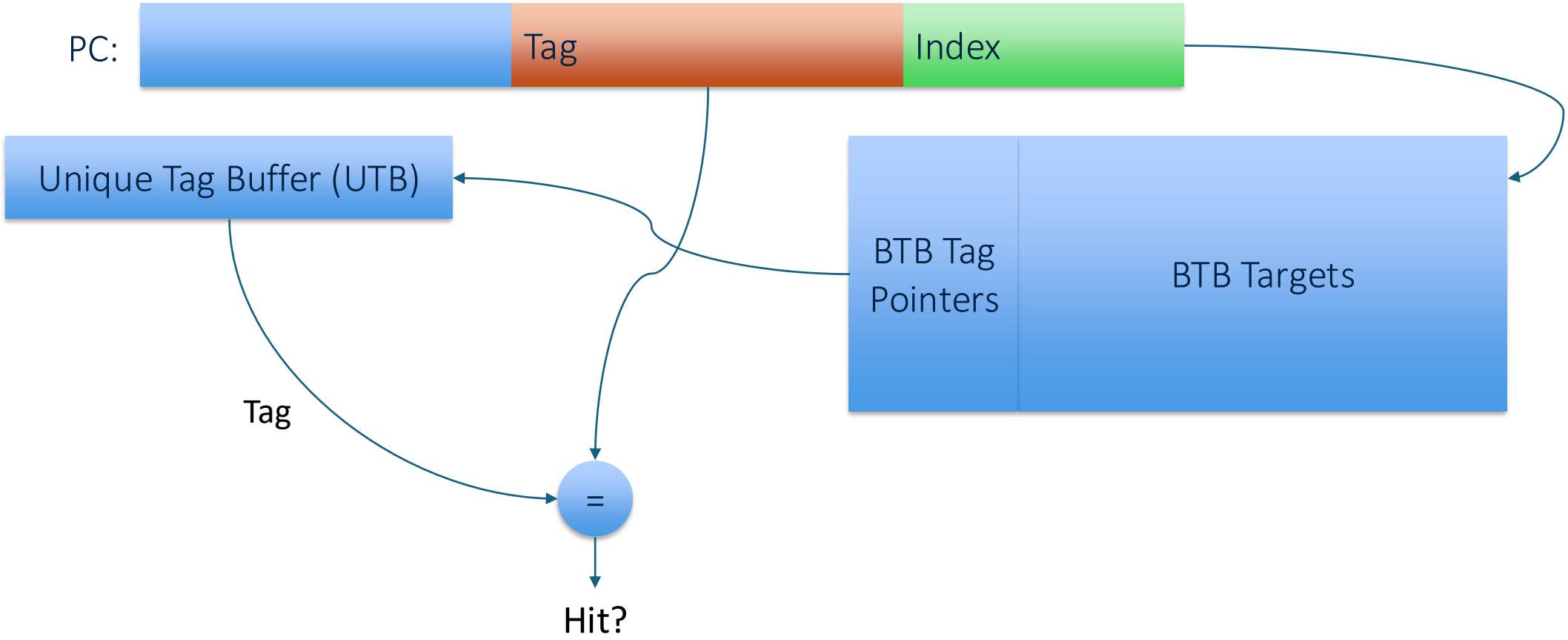
Accessing UTB: Naïve Approach



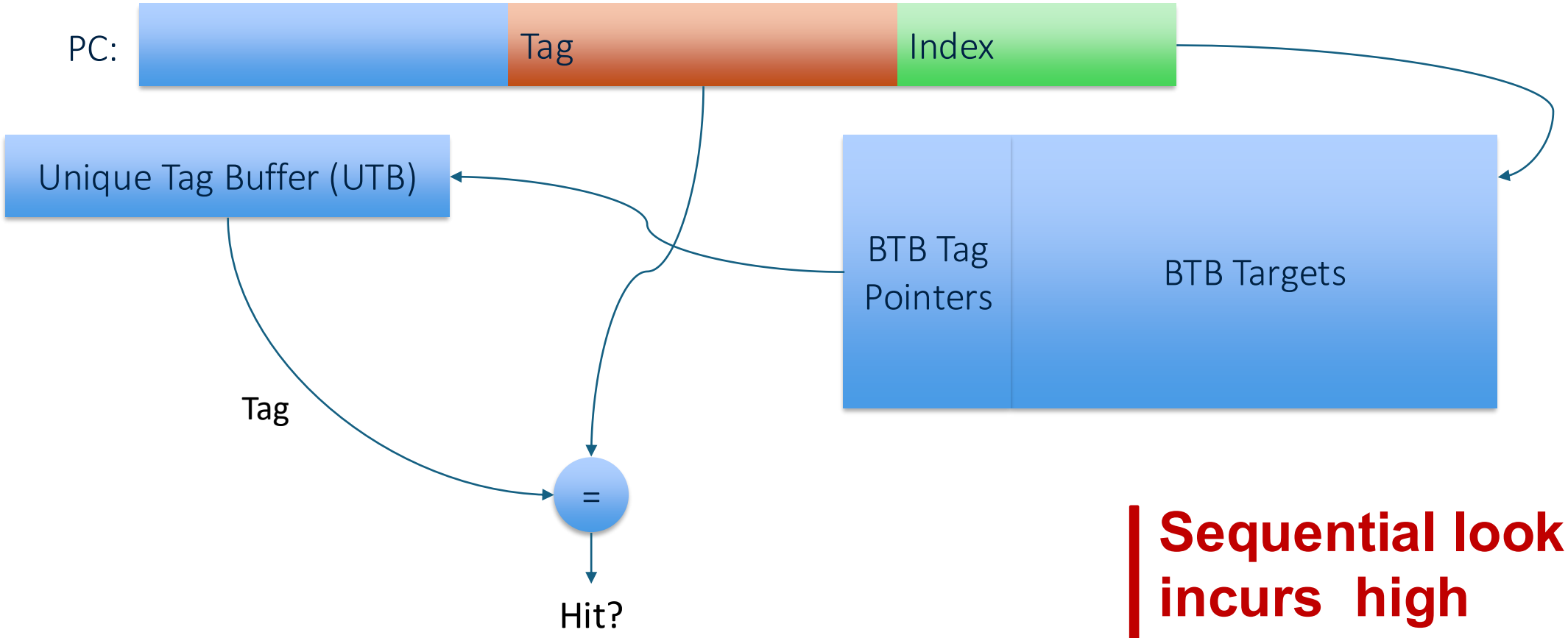
Accessing UTB: Naïve Approach



Accessing UTB: Naïve Approach

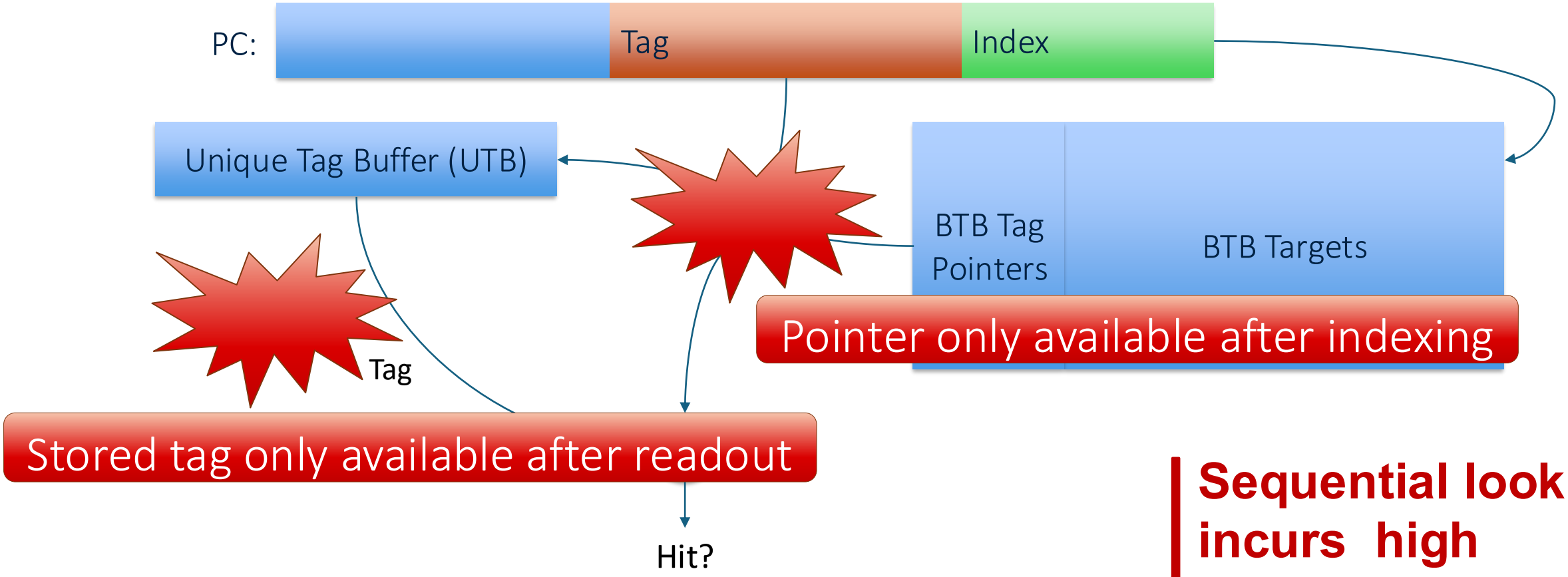


Accessing UTB: Naïve Approach



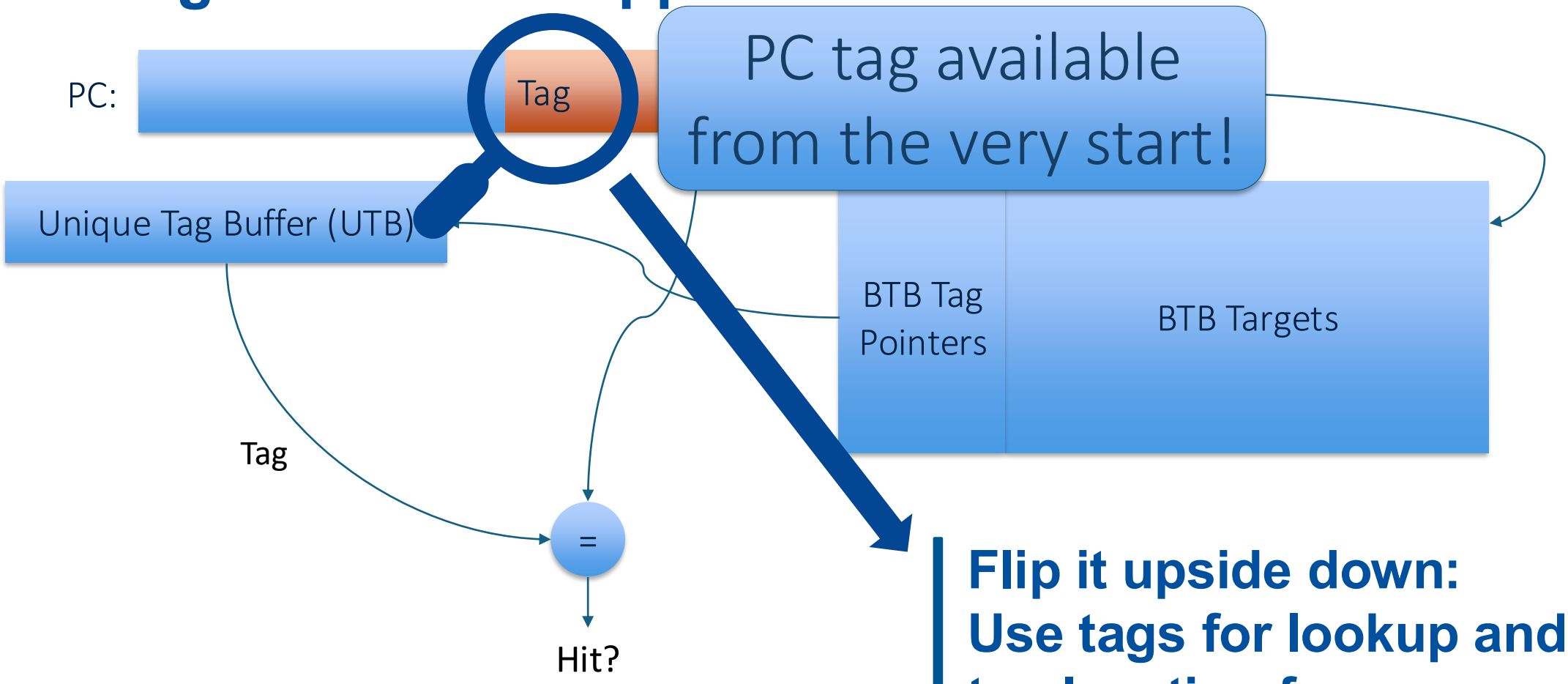
Sequential lookup incurs high lookup latency

Accessing UTB: Naïve Approach



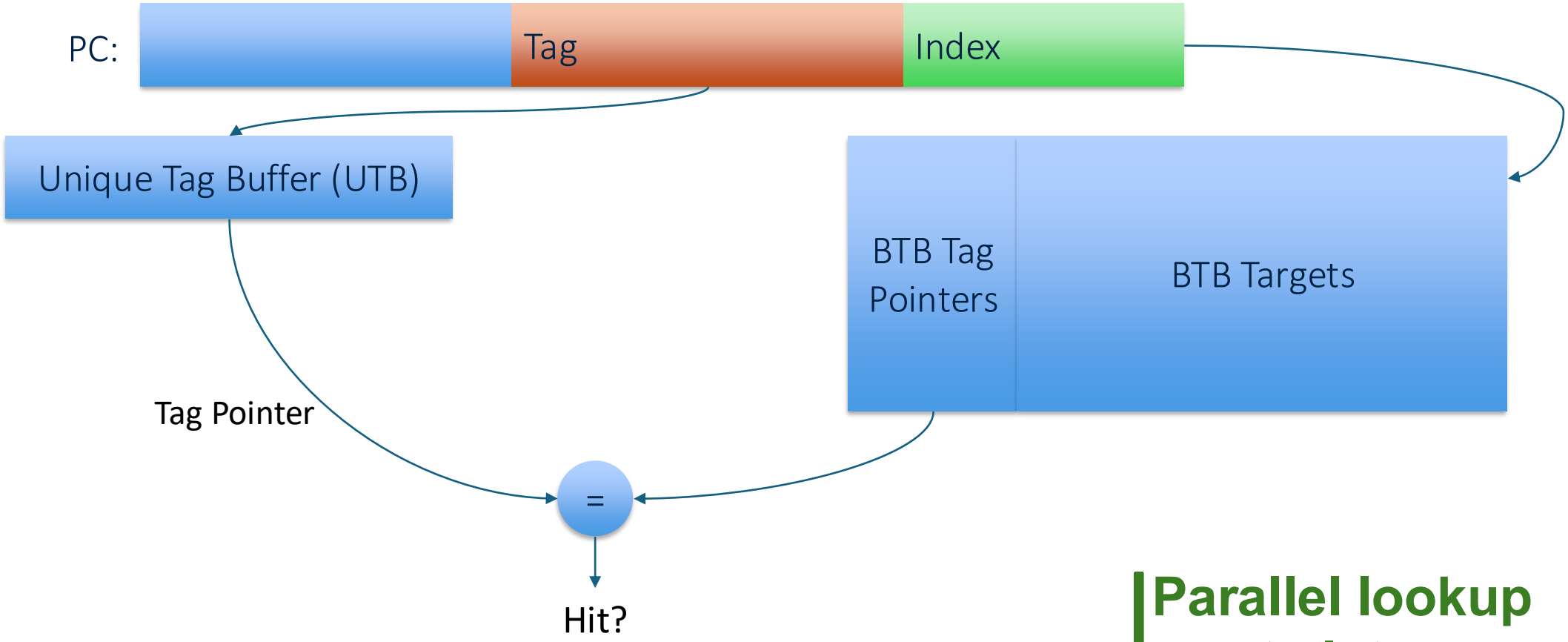
Sequential lookup incurs high lookup latency

Accessing UTB: Naïve Approach



Flip it upside down: Use tags for lookup and tag location for comparison

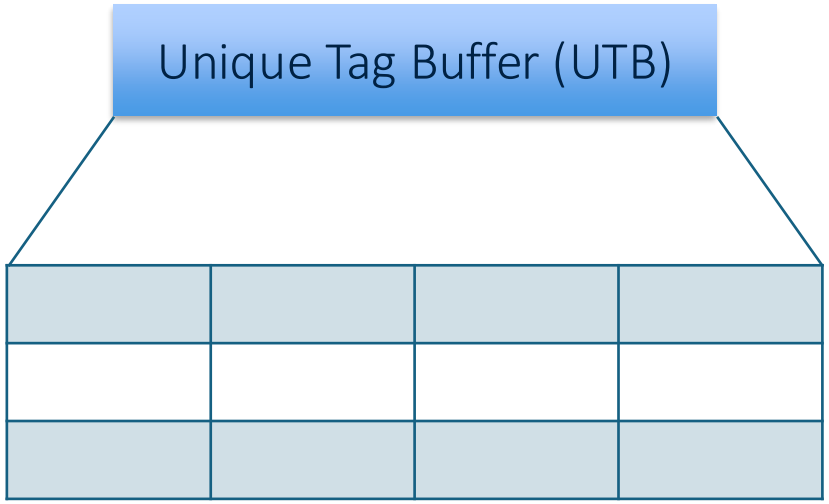
Accessing UTB: LiteBTB Approach



Parallel lookup meets latency requirements

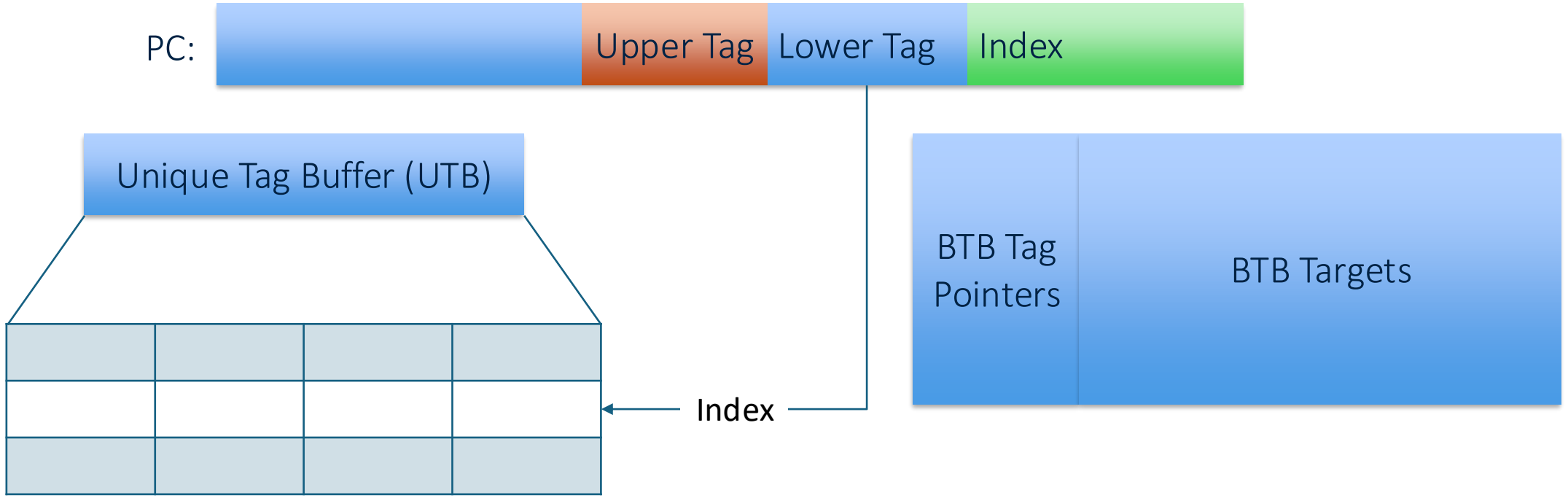
UTB Organisation



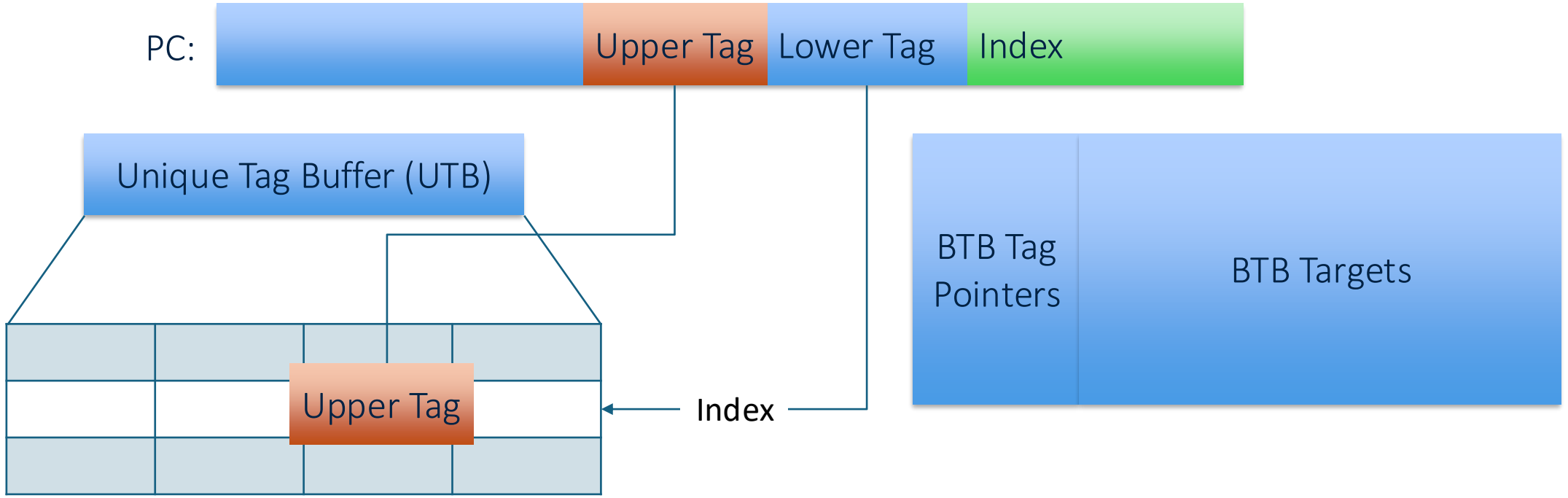




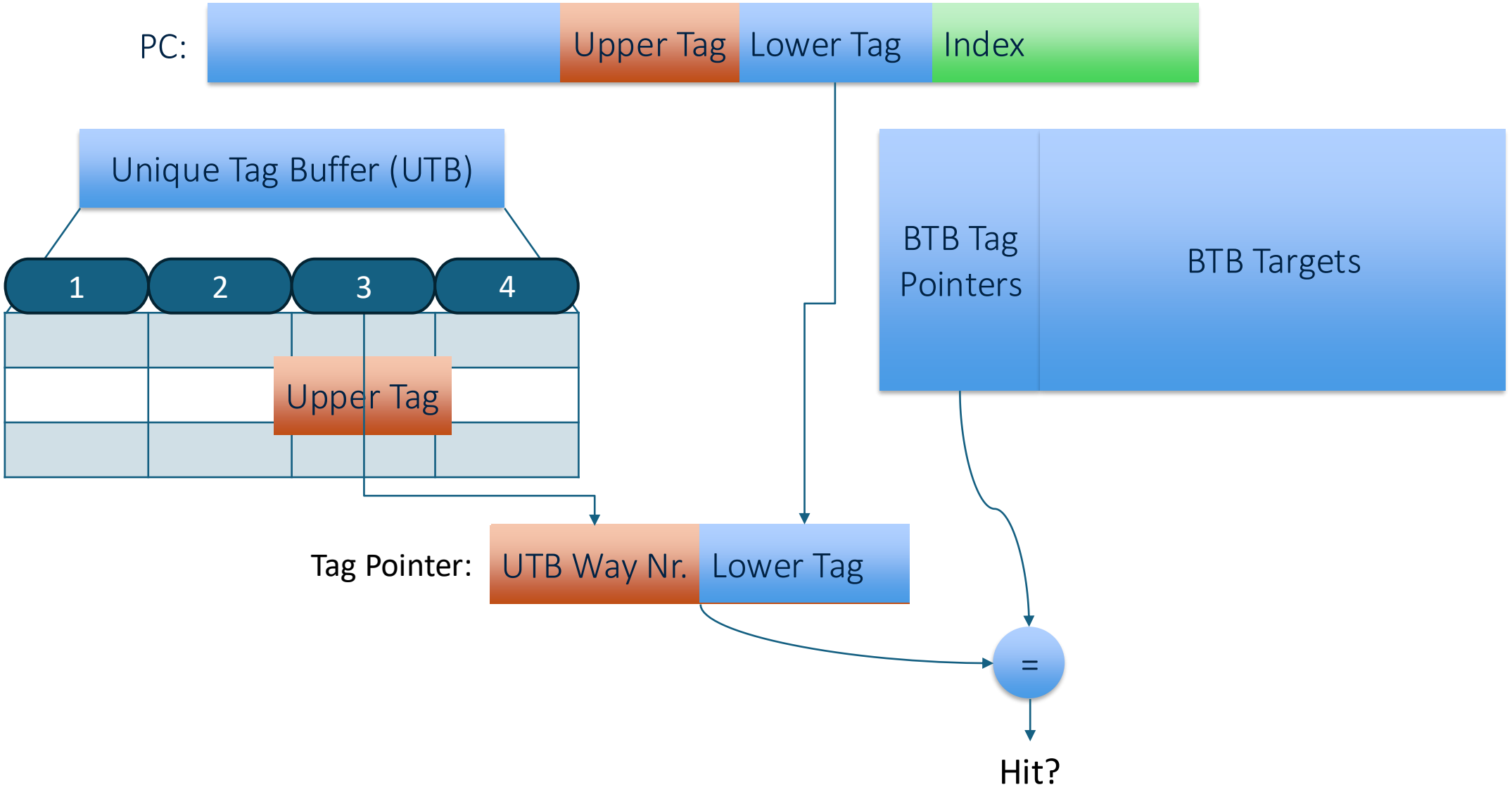
UTB Insertion



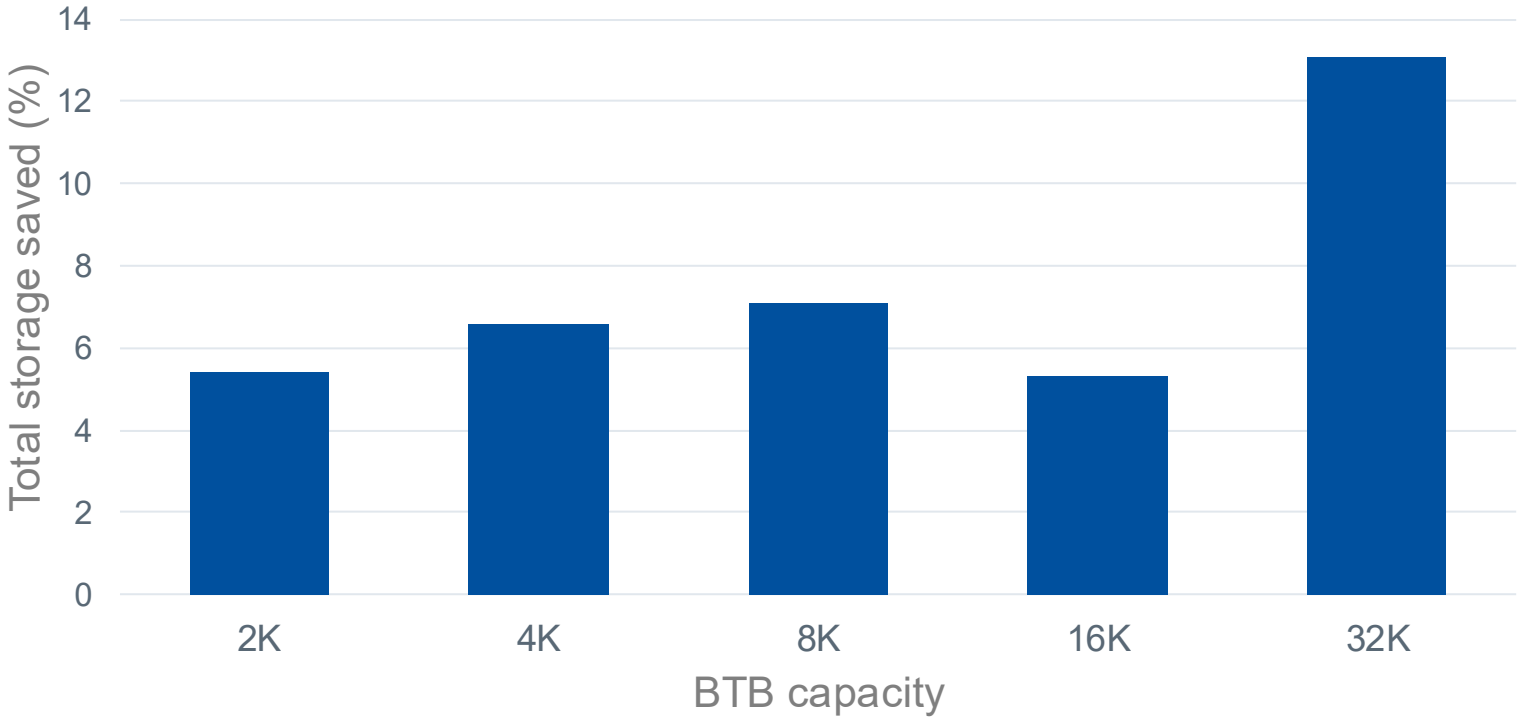
UTB Insertion



UTB Lookup


PC:
Upper Tag
Lower Tag
Index
Unique Tag Buffer (UTB)
1
2
3
4
Upper Tag
BTB Tag Pointers
BTB Targets
Tag Pointer:
UTB Way Nr.
Lower Tag
=
Hit?

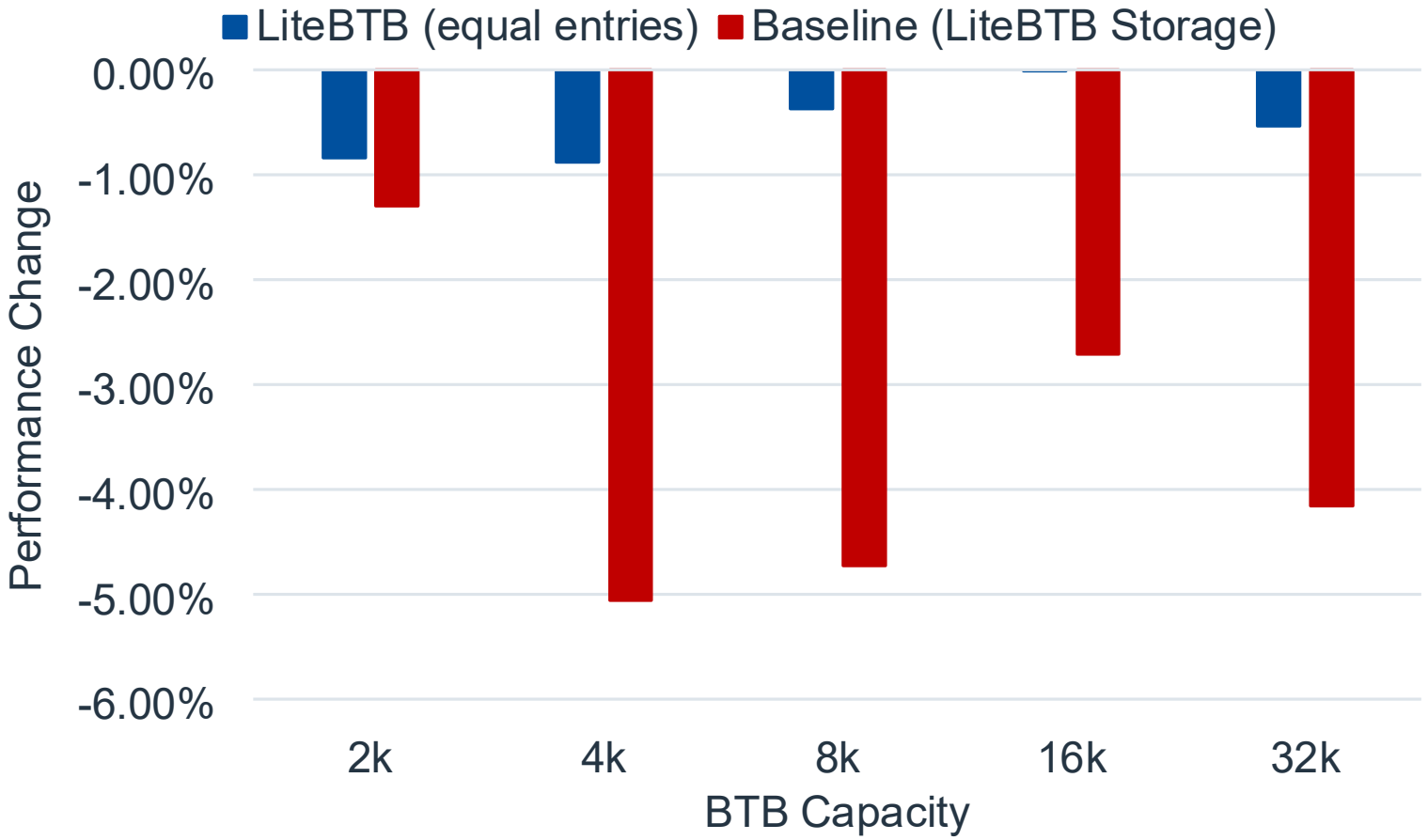
LiteBTB reduces storage beyond BTB-X



Up to 13.1%

less total storage,
at comparable
performance

Performance Impact



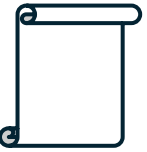
LiteBTB retains Performance

Baseline at LiteBTB storage capacity looses up to 5% of performance

Conclusion



BTBs keep growing at rapid rates to keep up with application demand



Prior work on BTB targets make tags the major storage bottleneck



BTB tags are highly redundant

- Tags cover large regions of code, potentially covering hundreds of branches
- We observed as few as 1.1% unique tags in a 32k entry BTB



LiteBTB saves up to 13.1% of storage by deduplicating tags

- Unique Tag Buffer (UTB) stores each tag once
- Parallel lookup of the BTB and the UTB keep latency in bounds

NTNU

Norwegian University of Science and Technology

Thanks!

The Path to BTB Storage Efficiency Goes Through Tags

Roman K. Brunner
roman.k.brunner@ntnu.no

Rakesh Kumar
rakesh.kumar@ntnu.no

Computer Architecture Laboratory, NTNU