IvLeague: *Side Channel-resistant Secure Architectures* Using Isolated Domains of Dynamic Integrity Trees

Md Hafizul Islam Chowdhuryy and **Fan Yao**

Computer Architecture and Systems Research Lab (CASRL)

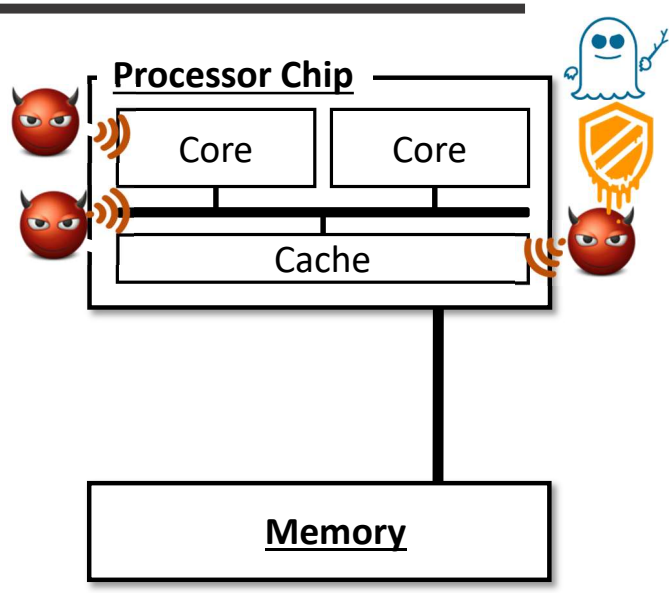
University of Central Florida

Orlando, USA

57th IEEE/ACM International Symposium on Microarchitecture (**MICRO 2024**)
November 2nd – November 6th, 2024

µArch Security Problems in Secure Processors

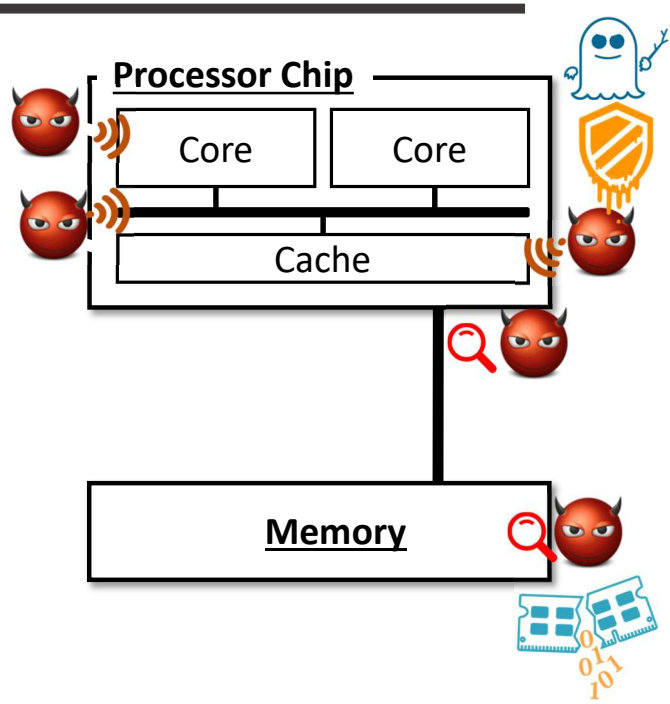
- Two different attack domains are possible:
  - **On-chip:** uArch side channel attacks (e.g., cache attacks, BPU attacks).



UCF

µArch Security Problems in Secure Processors

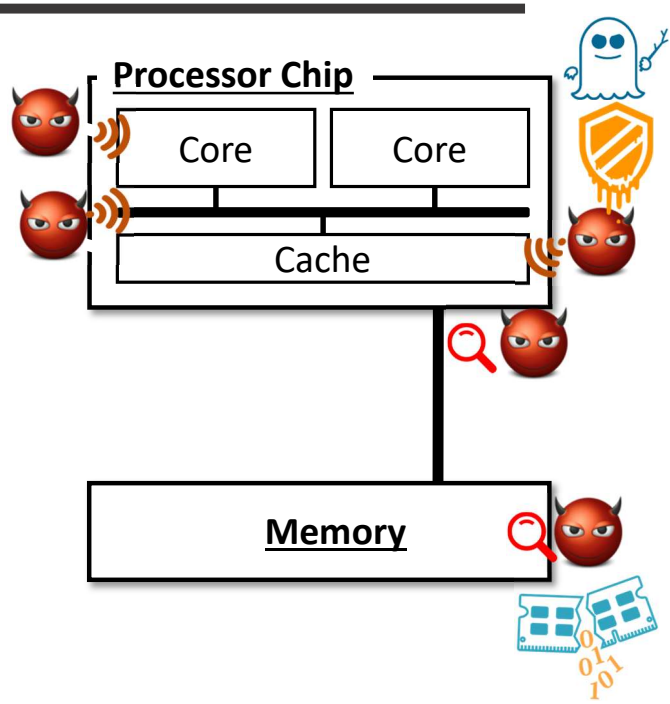
- Two different attack domains are possible:
  - **On-chip:** uArch side channel attacks (e.g., cache attacks, BPU attacks).
  - **Off-chip:** Physical attacks (e.g., bus snooping, data tampering attacks).



UCF

µArch Security Problems in Secure Processors

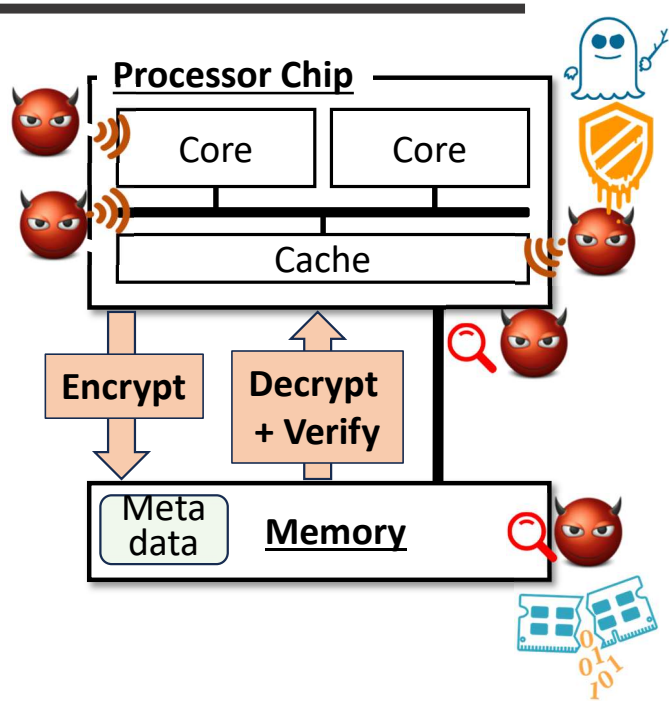
- Two different attack domains are possible:
  - **On-chip:** uArch side channel attacks (e.g., cache attacks, BPU attacks).
  - **Off-chip:** Physical attacks (e.g., bus snooping, data tampering attacks).
- Increasing demand for both on-chip and off-chip security.



UCF

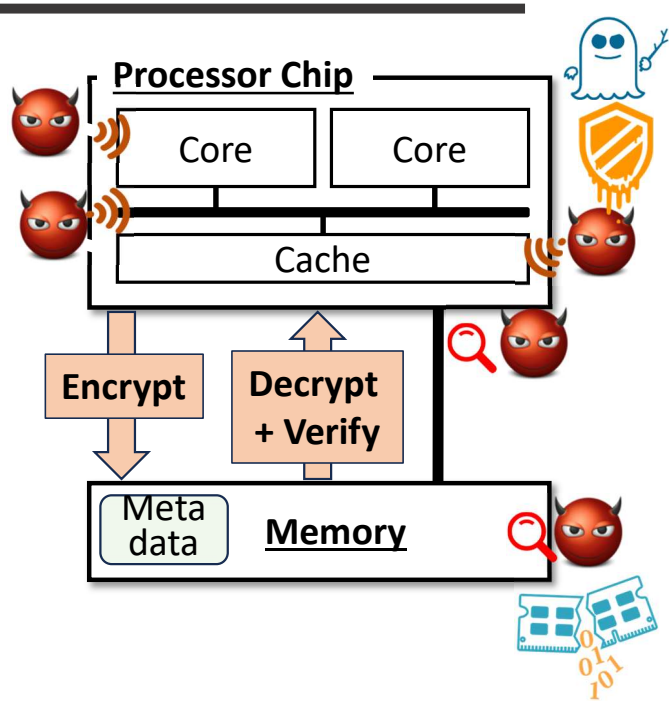
µArch Security Problems in Secure Processors

- Two different attack domains are possible:
  - **On-chip:** uArch side channel attacks (e.g., cache attacks, BPU attacks).
  - **Off-chip:** Physical attacks (e.g., bus snooping, data tampering attacks).
- Increasing demand for both on-chip and off-chip security.
- ***Secure processor design objectives:*** Protect off-chip data.
  - ***Encryption*** when data leave processor chip.
  - ***Decryption and integrity verification*** when data enter on chip.
- Secure processors are the foundation of *Trusted execution environments*.
  - ***Industry solutions:*** Intel SGX/TDX, Arm TrustZone, AMD SEV.



µArch Security Problems in Secure Processors

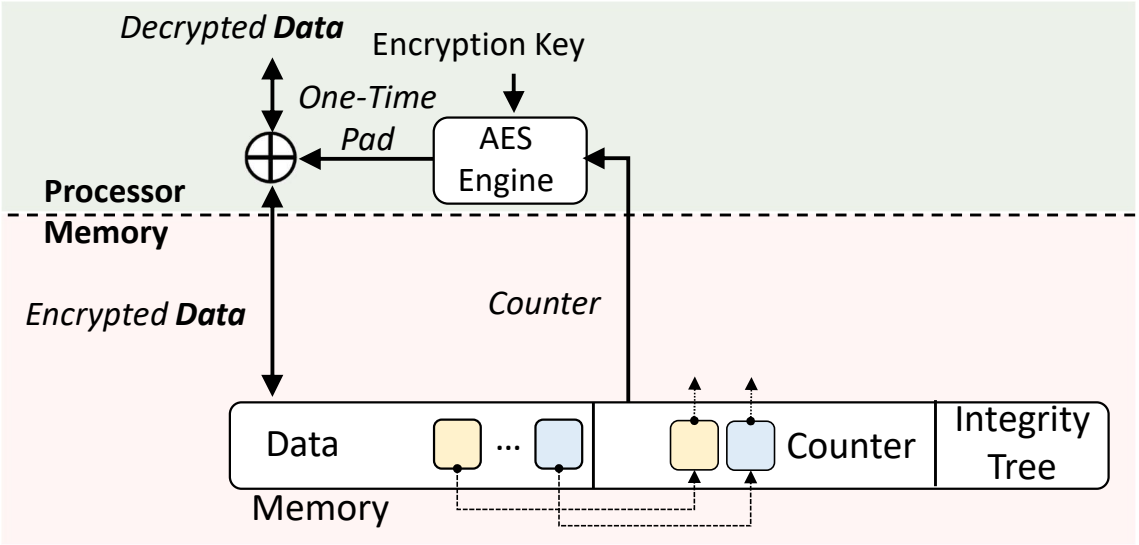
- Two different attack domains are possible:
  - **On-chip:** uArch side channel attacks (e.g., cache attacks, BPU attacks).
  - **Off-chip:** Physical attacks (e.g., bus snooping, data tampering attacks).
- Increasing demand for both on-chip and off-chip security.
- ***Secure processor design objectives:*** Protect off-chip data.
  - ***Encryption*** when data leave processor chip.
  - ***Decryption and integrity verification*** when data enter on chip.
- Secure processors are the foundation of *Trusted execution environments*.
  - ***Industry solutions:*** Intel SGX/TDX, Arm TrustZone, AMD SEV.



Traditional **secure processor designs** only consider security of **off-chip data**.

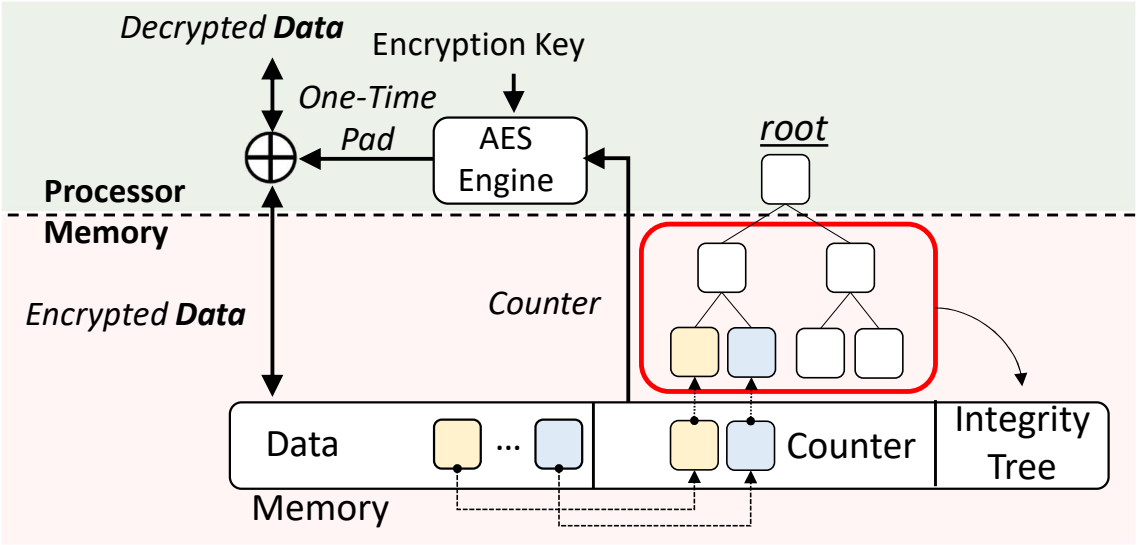
UCF

Side Channel Security of Secure Processor Metadata



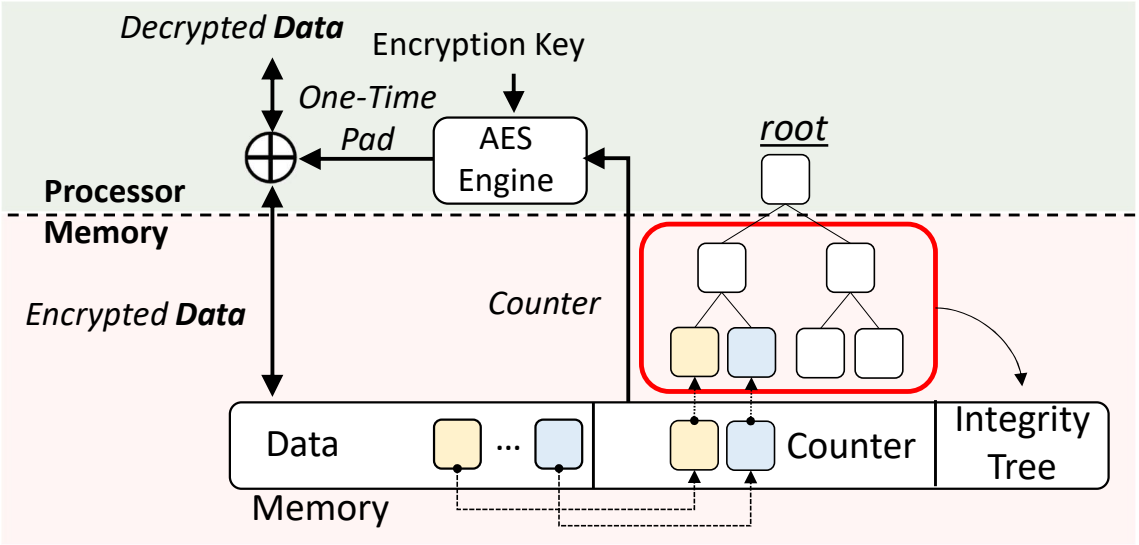
UCF

Side Channel Security of Secure Processor Metadata



UCF

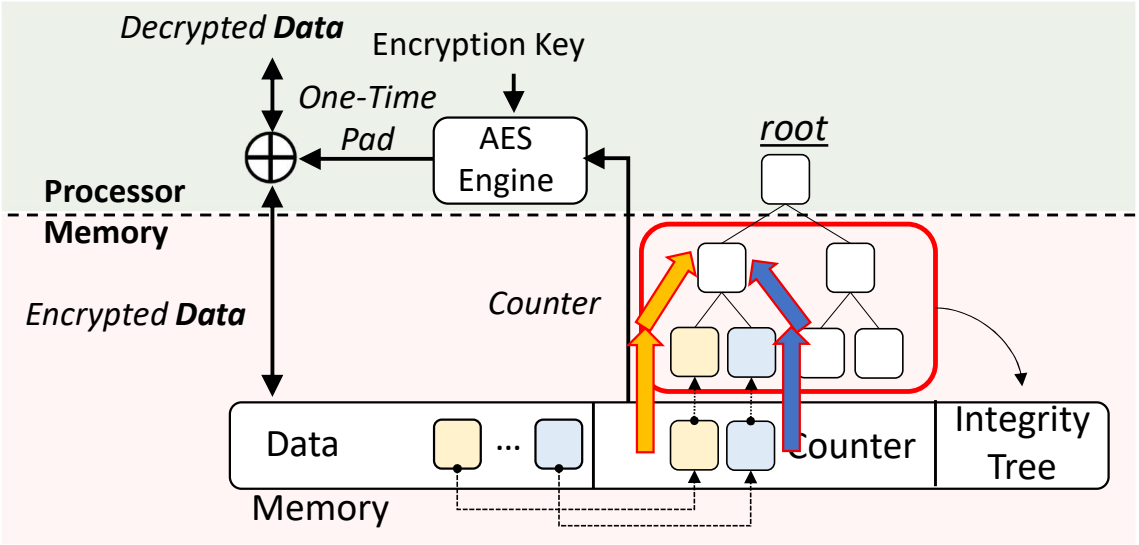
Side Channel Security of Secure Processor Metadata



- **Implicit memory sharing:**
  - Integrity tree creates ***shared tree blocks*** *among pages* (across processes).

UCF

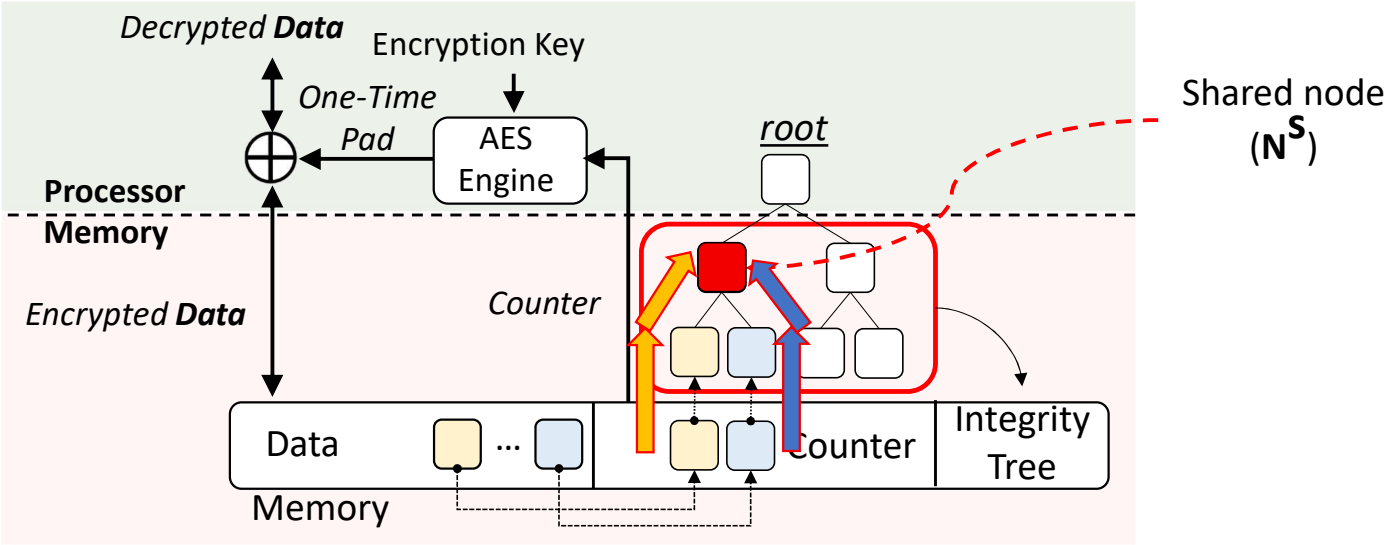
Side Channel Security of Secure Processor Metadata



- **Implicit memory sharing:**
  - Integrity tree creates ***shared tree blocks*** *among pages* (across processes).

UCF

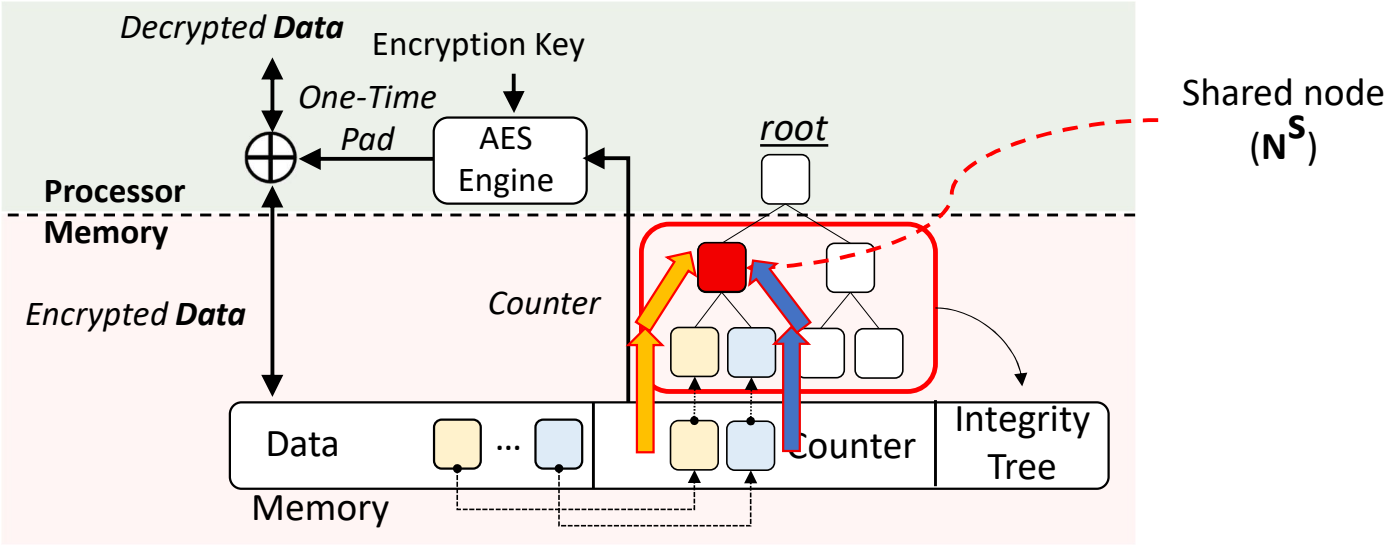
Side Channel Security of Secure Processor Metadata



- **Implicit memory sharing:**
  - Integrity tree creates ***shared tree blocks*** *among pages* (across processes).

UCF

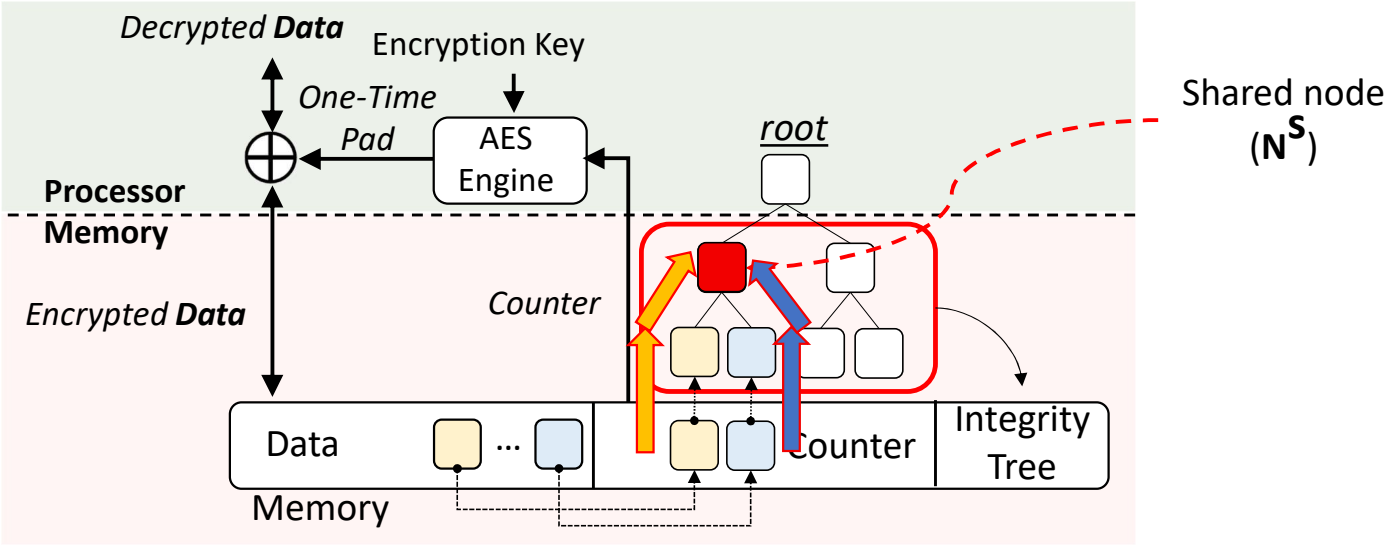
Side Channel Security of Secure Processor Metadata



- **Implicit memory sharing:**
  - Integrity tree creates ***shared tree blocks*** *among pages* (across processes).
  - Creates practical **shared-memory** side channel **without explicit data sharing**.

UCF

Side Channel Security of Secure Processor Metadata

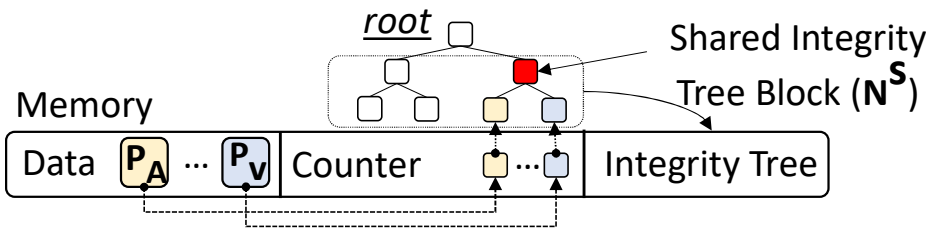


- **Implicit memory sharing:**
  - Integrity tree creates ***shared tree blocks*** *among pages* (across processes).
  - Creates practical **shared-memory** side channel **without explicit data sharing**.

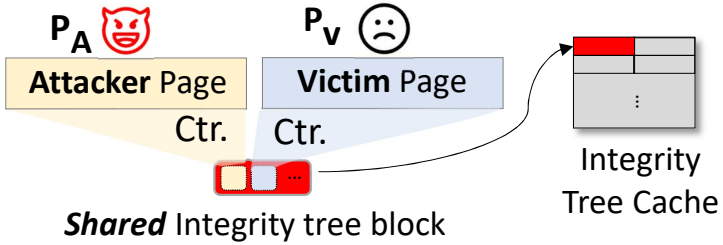
Secure processors introduce **new source of side channel leakage** (MetaLeak [ISCA'24]).

UCF

Side Channel Exploiting Tree Metadata Sharing



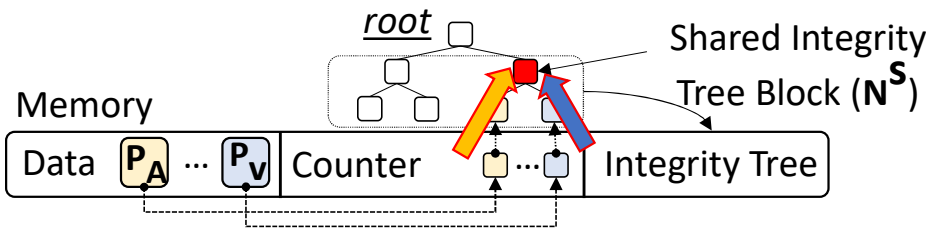
Implicit sharing of integrity tree metadata



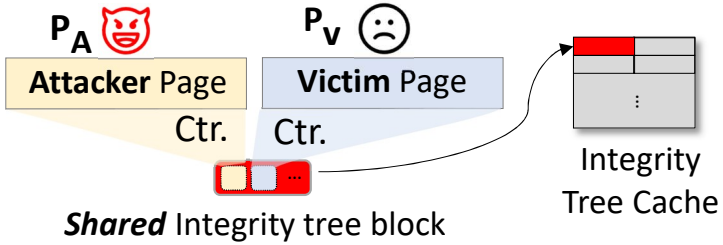
Shared Integrity tree block

UCF

Side Channel Exploiting Tree Metadata Sharing



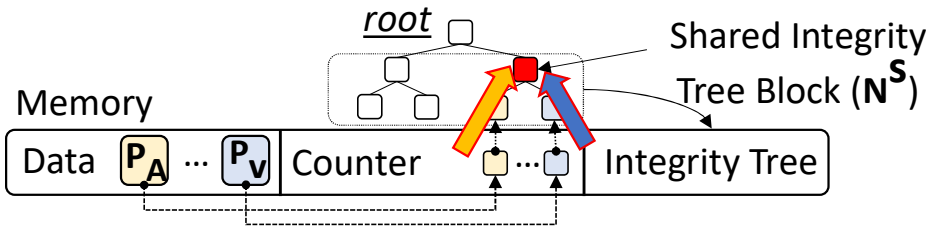
Implicit sharing of integrity tree metadata



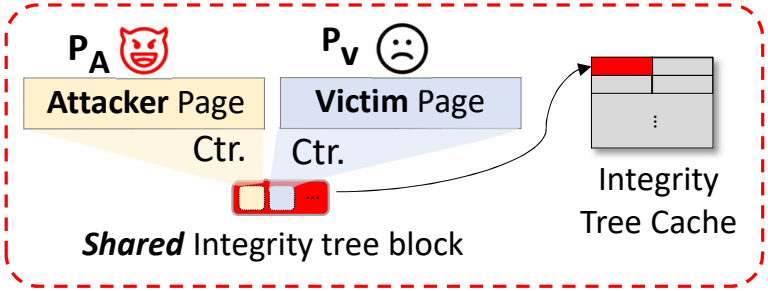
Shared Integrity tree block

UCF

Side Channel Exploiting Tree Metadata Sharing

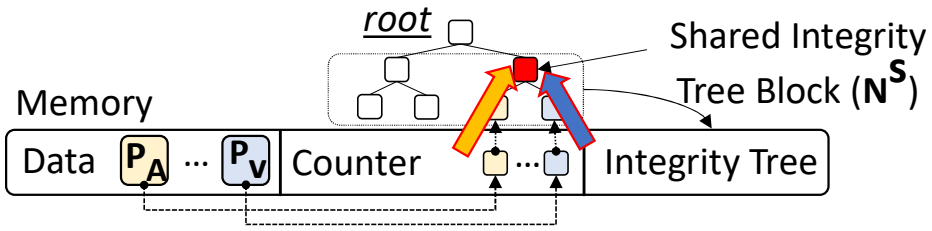


Implicit sharing of integrity tree metadata

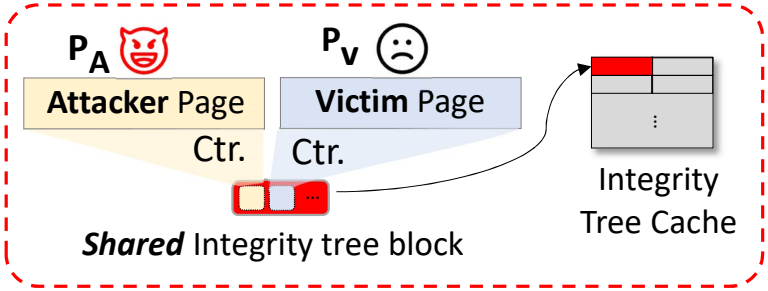


UCF

Side Channel Exploiting Tree Metadata Sharing



Implicit sharing of integrity tree metadata



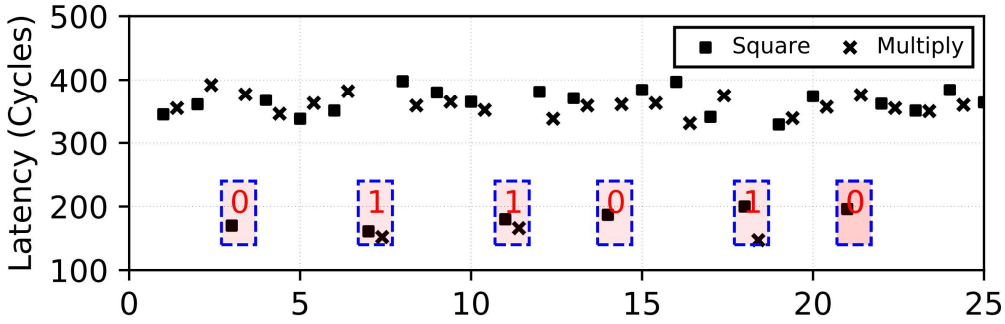
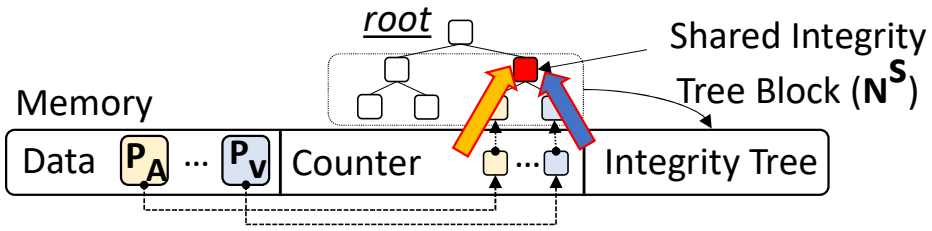


Figure: RSA private exponent recovery in **Intel SGX** (Core i7-9700K)*

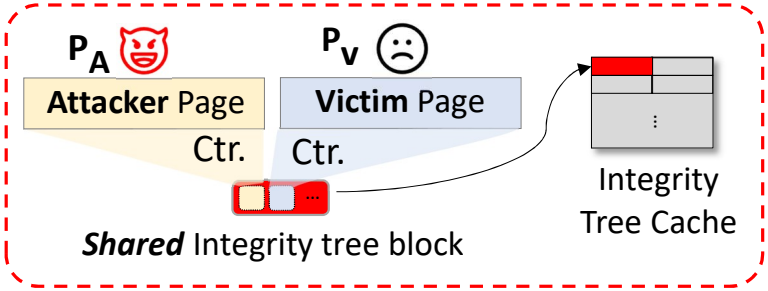
* Reproduction of attack demonstrated in MetaLeak [ISCA'24]

UCF

Side Channel Exploiting Tree Metadata Sharing



Implicit sharing of integrity tree metadata



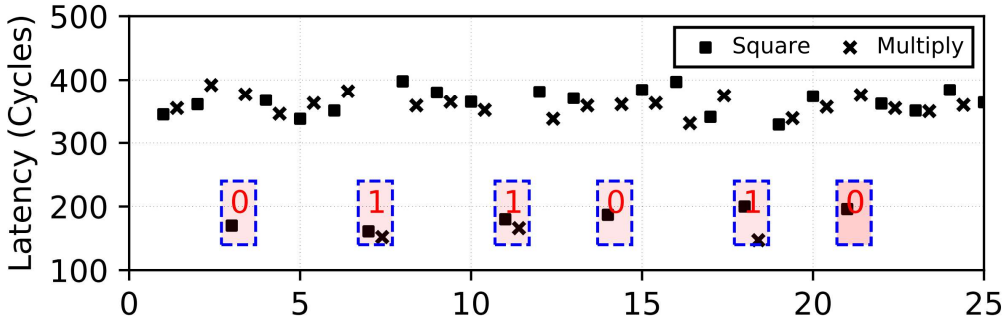


Figure: RSA private exponent recovery in **Intel SGX** (Core i7-9700K)*

Key vulnerability: **Integrity verification metadata is shared** across domains, leading to **shared-memory side channels** even when regular data sharing is prohibited.

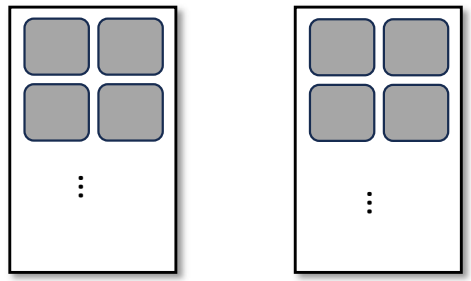
* Reproduction of attack demonstrated in MetaLeak [ISCA'24]

UCF

Uniqueness of Metadata-based Exploits

- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.

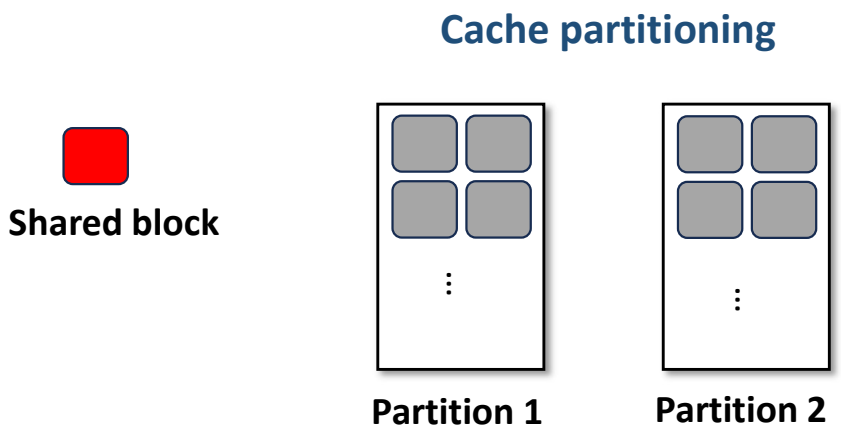
Cache partitioning



UCF

Uniqueness of Metadata-based Exploits

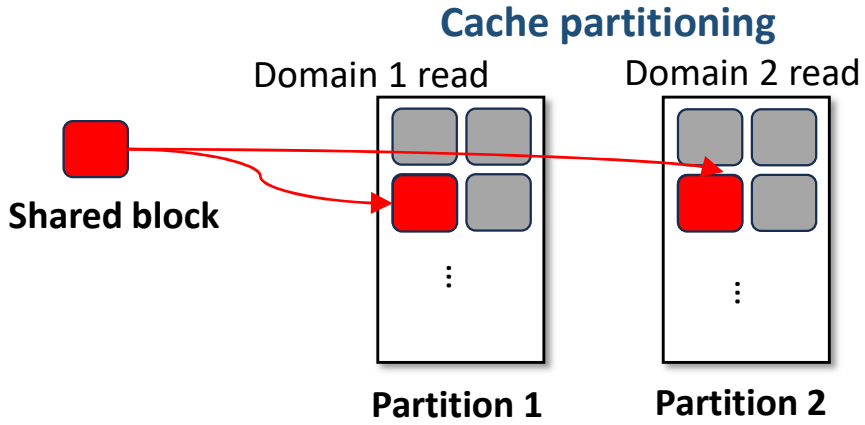
- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.



UCF

Uniqueness of Metadata-based Exploits

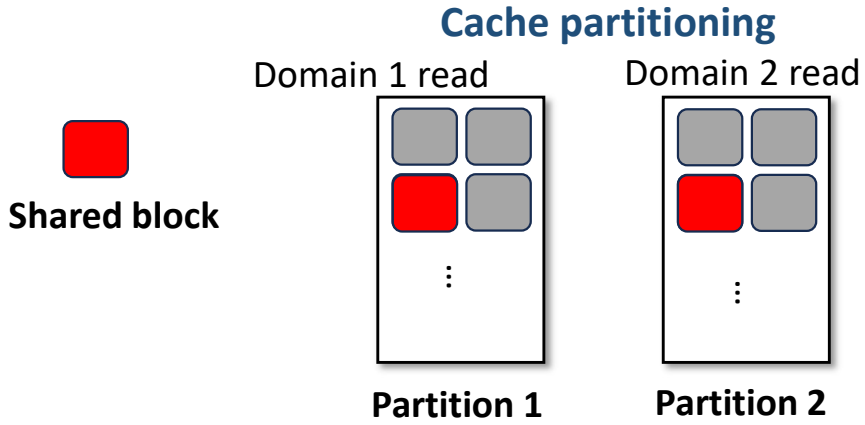
- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.



UCF

Uniqueness of Metadata-based Exploits

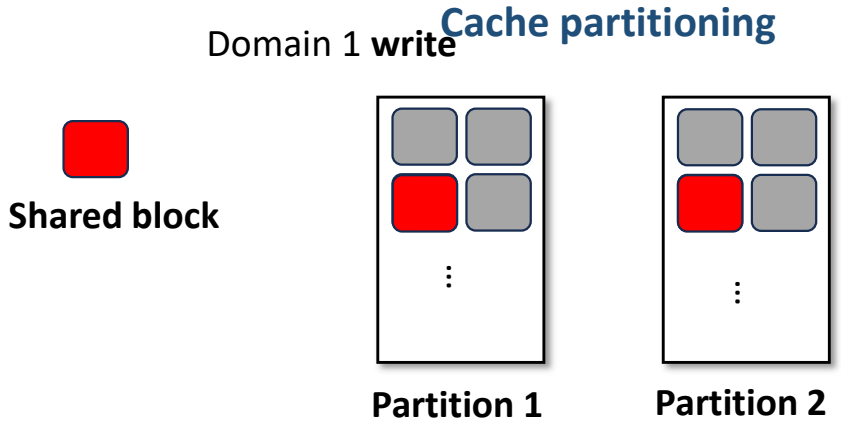
- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.



UCF

Uniqueness of Metadata-based Exploits

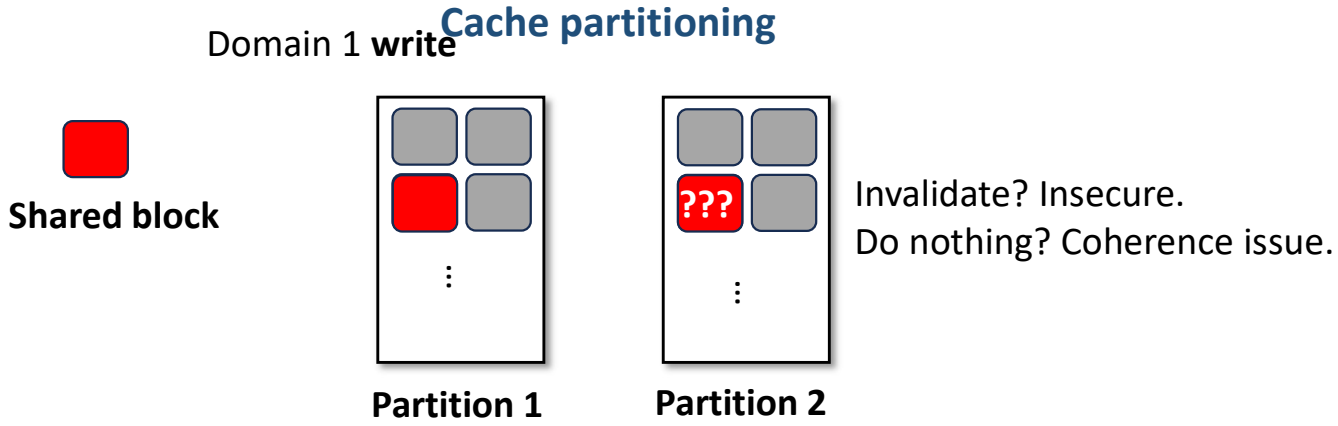
- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.



UCF

Uniqueness of Metadata-based Exploits

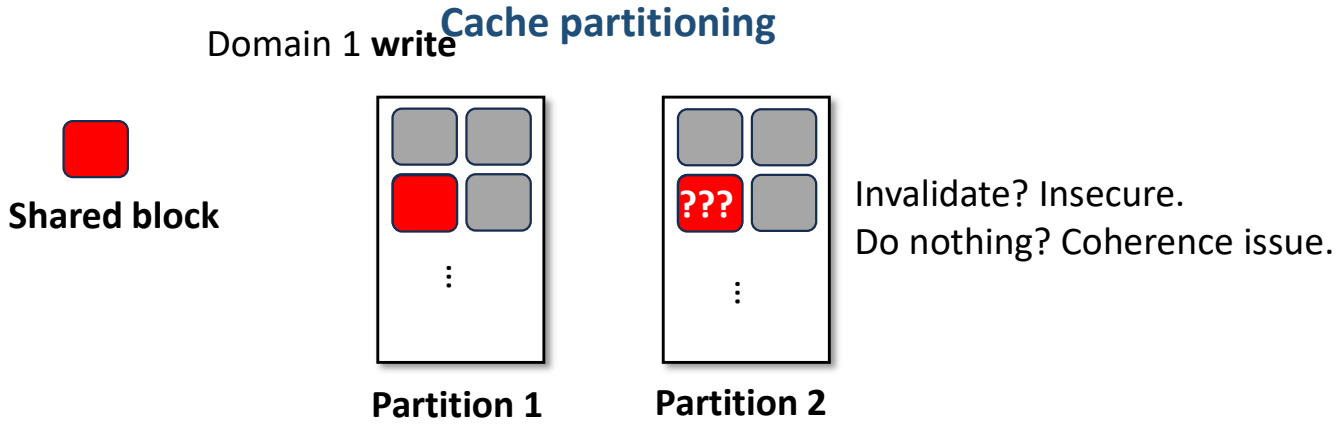
- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.



UCF

Uniqueness of Metadata-based Exploits

- MetaLeak breaks assumptions the fundamental assumptions of uArch defenses (**sharing**).
- Existing µArch defenses are typically based on ***randomization*** or ***partitioning***.
  - Mainstream protection schemes against ***shared-memory attack requires read-only sharing***.
  - Secure processor metadata is ***shared and updated during runtime***.



Integrity tree metadata is ***shared and writable*** → cannot be protected using classical defenses.

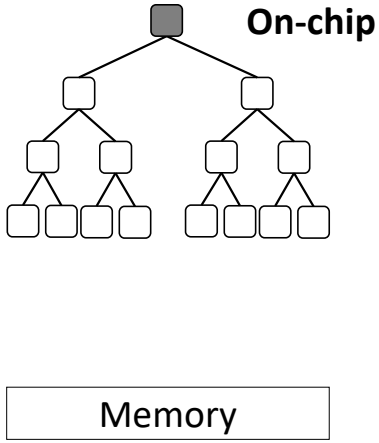
UCF

This work

Architecture support for **side channel-resistant** secure processor metadata to prevent metadata-sharing side channel.

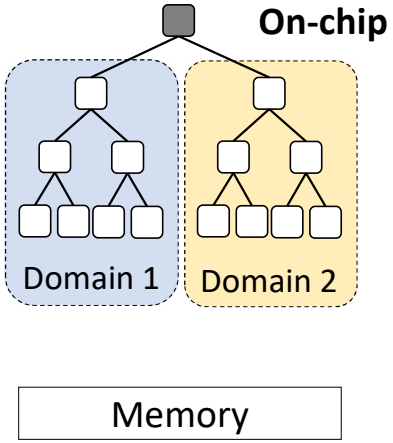
Design Objectives of Isolated Integrity Tree

- **Main idea:** Metadata-level isolation for integrity verification.
  - Ensure no tree node sharing in memory between domains.



Design Objectives of Isolated Integrity Tree

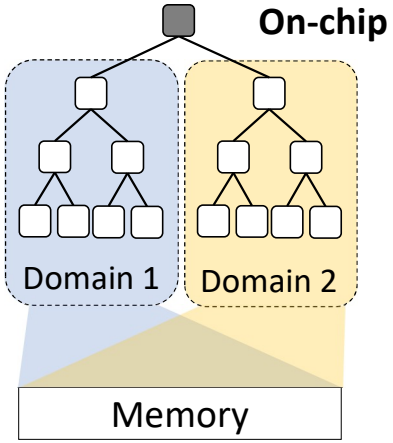
- **Main idea:** Metadata-level isolation for integrity verification.
  - Ensure no tree node sharing in memory between domains.



UCF

Design Objectives of Isolated Integrity Tree

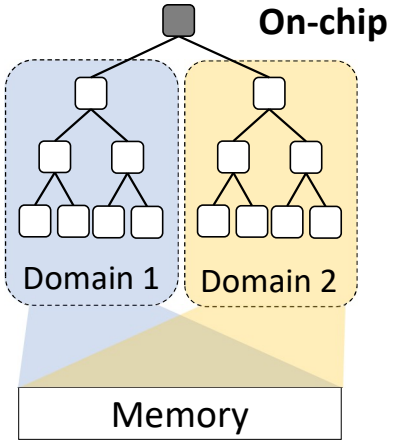
- **Main idea:** Metadata-level isolation for integrity verification.
  - Ensure no tree node sharing in memory between domains.



UCF

Design Objectives of Isolated Integrity Tree

- **Main idea:** Metadata-level isolation for integrity verification.
  - Ensure no tree node sharing in memory between domains.



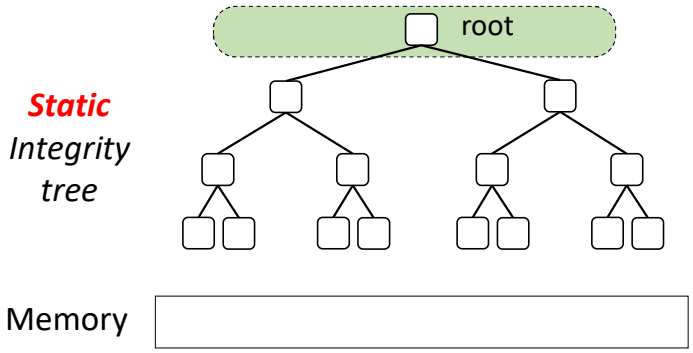
Obj. (#) : Support for a *sufficient number of isolated integrity domains*.

Obj. (⤡) : Ability to *resize integrity coverage of each domain* on-demand.

Obj. ($) : *Low-overhead maintenance* of tree nodes during runtime.

UCF

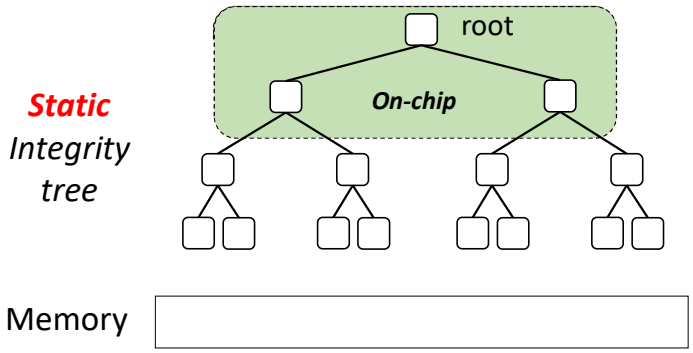
Design Space Exploration: Fully Static Partitioning



- Statically partition the global tree.

UCF

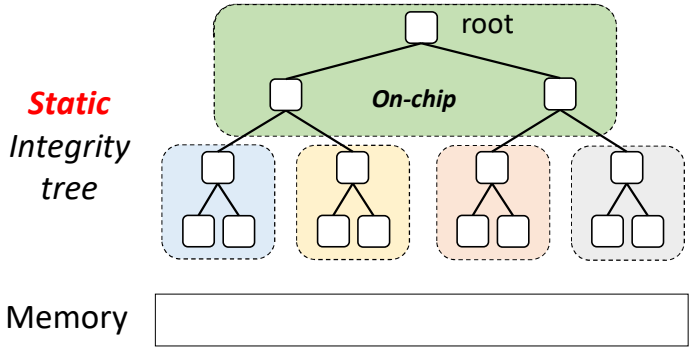
Design Space Exploration: Fully Static Partitioning



- Statically partition the global tree.

UCF

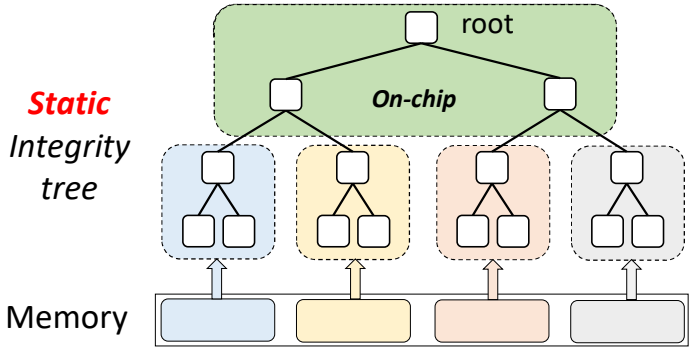
Design Space Exploration: Fully Static Partitioning



- Statically partition the global tree.

UCF

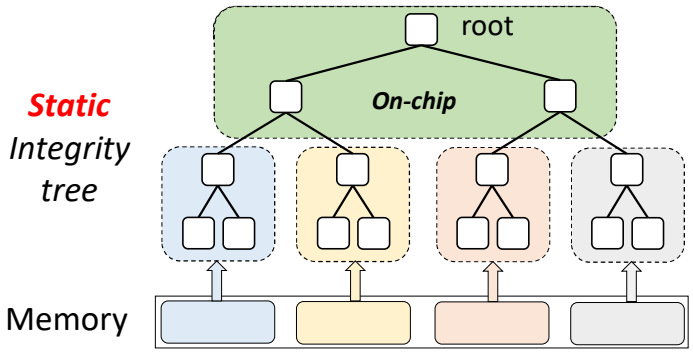
Design Space Exploration: Fully Static Partitioning



- Statically partition the global tree.
- Each sub-tree covers a fixed chunk of memory.

UCF

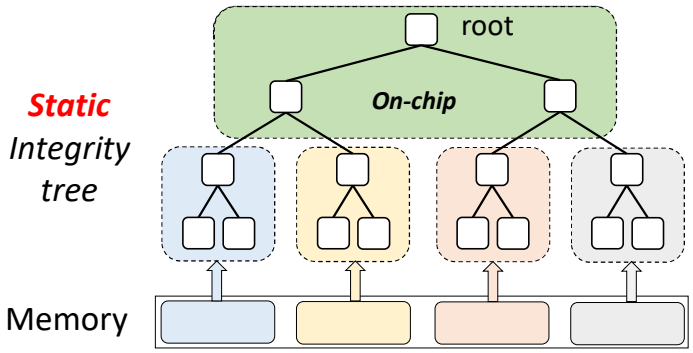
Design Space Exploration: Fully Static Partitioning



- Statically partition the global tree.
- Each sub-tree covers a fixed chunk of memory.
- Static one-to-one mapping between data to sub-tree node:
  - Low-overhead maintenance of tree nodes (**Obj. $ Satisfied**).

UCF

Design Space Exploration: Fully Static Partitioning

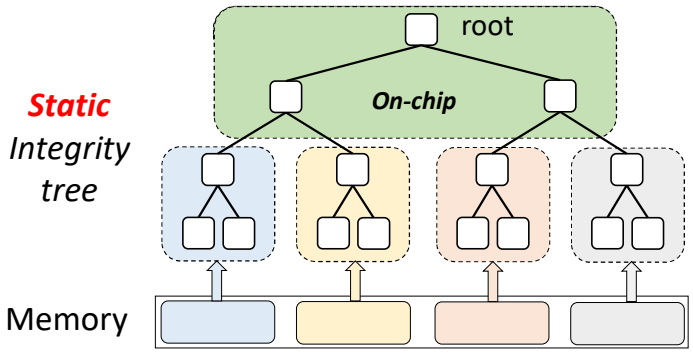


- Statically partition the global tree.
- Each sub-tree covers a fixed chunk of memory.
- Static one-to-one mapping between data to sub-tree node:
  - Low-overhead maintenance of tree nodes (**Obj. $ Satisfied**).

Number of security domains does not scale → limited number of domains (**Obj. # Not-satisfied**).

UCF

Design Space Exploration: Fully Static Partitioning



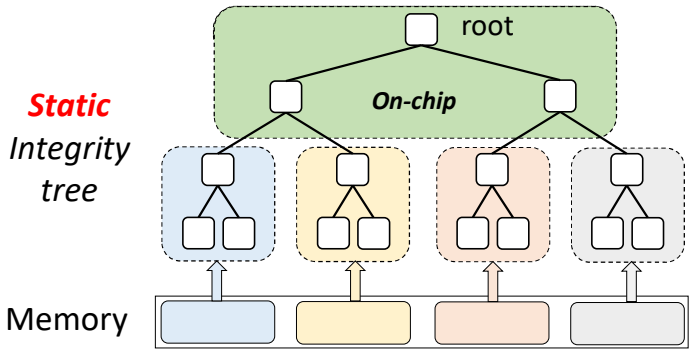
- Statically partition the global tree.
- Each sub-tree covers a fixed chunk of memory.
- Static one-to-one mapping between data to sub-tree node:
  - Low-overhead maintenance of tree nodes (**Obj. $ Satisfied**).

Number of security domains does not scale → limited number of domains (**Obj. # Not-satisfied**).

Fixed memory coverage of each partition → cannot resize memory footprint on-demand (**Obj. ⤡ Not-satisfied**).

UCF

Design Space Exploration: Fully Static Partitioning



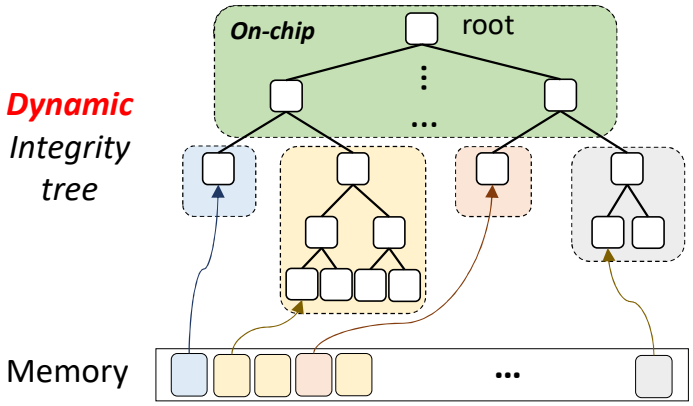
- Statically partition the global tree.
- Each sub-tree covers a fixed chunk of memory.
- Static one-to-one mapping between data to sub-tree node:
  - Low-overhead maintenance of tree nodes (**Obj. $ Satisfied**).
- Relies on the OS to strictly *allocate continuous physical pages* to domains.

Number of security domains does not scale → limited number of domains (**Obj. # Not-satisfied**).

Fixed memory coverage of each partition → cannot resize memory footprint on-demand (**Obj. ⤡ Not-satisfied**).

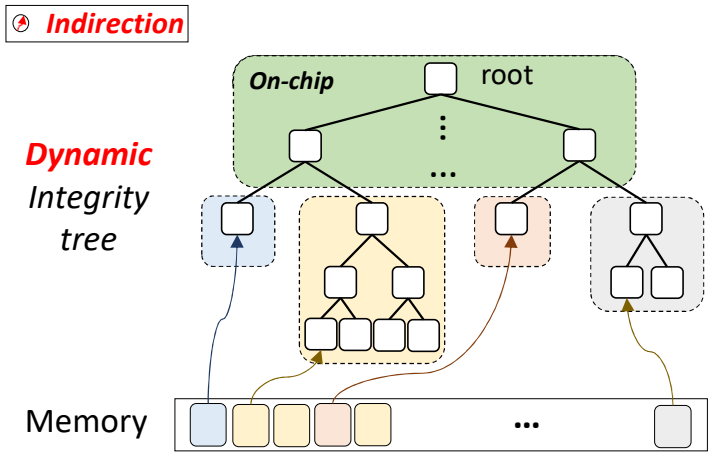
UCF

Design Space Exploration: Fully Dynamic Partitioning



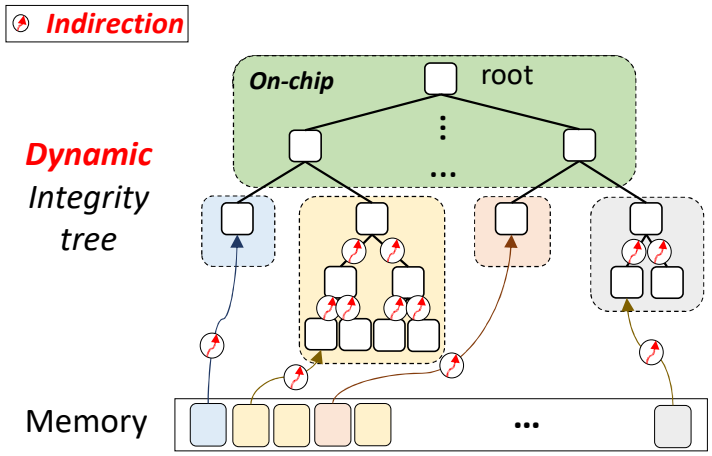
UCF

Design Space Exploration: Fully Dynamic Partitioning



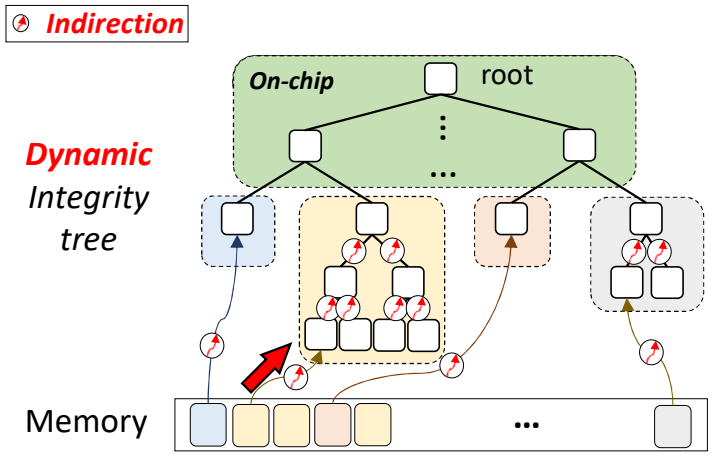
UCF

Design Space Exploration: Fully Dynamic Partitioning



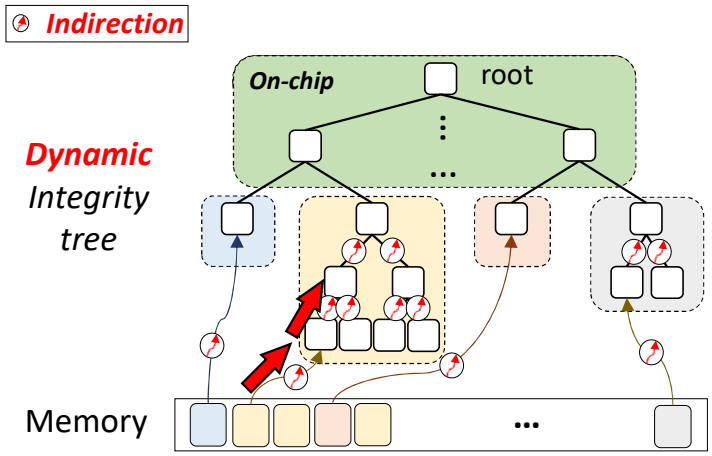
UCF

Design Space Exploration: Fully Dynamic Partitioning



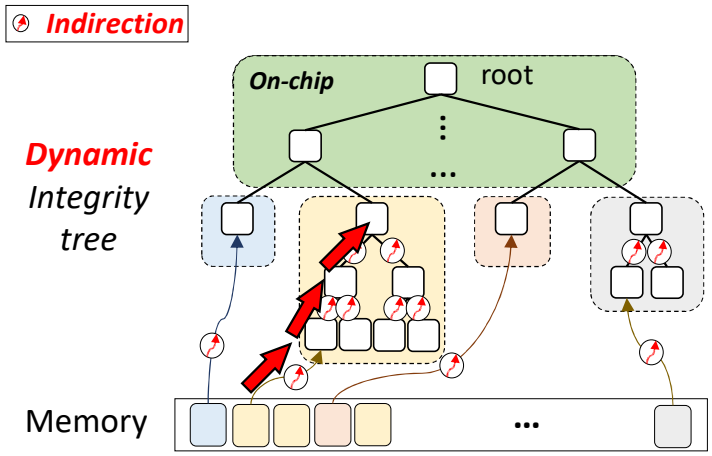
UCF

Design Space Exploration: Fully Dynamic Partitioning



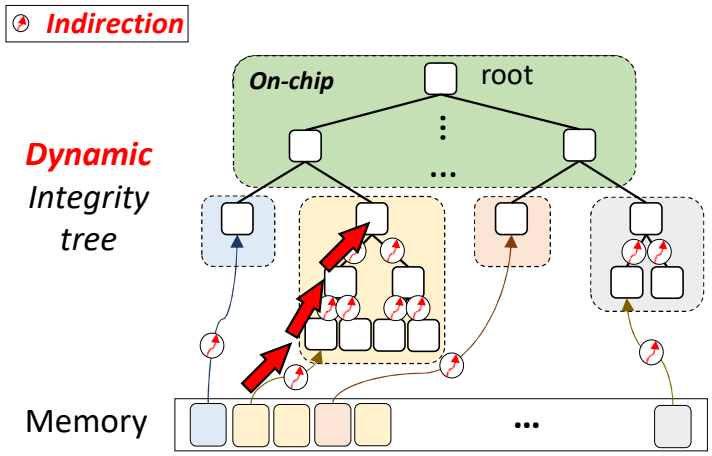
UCF

Design Space Exploration: Fully Dynamic Partitioning



UCF

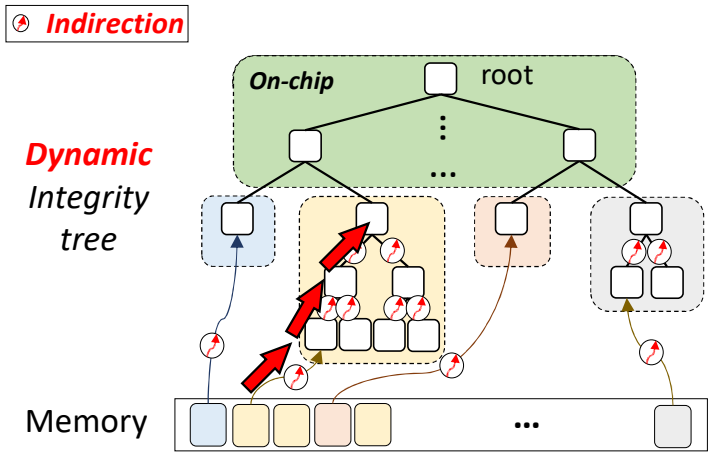
Design Space Exploration: Fully Dynamic Partitioning



- Each domain hosts individual integrity sub-tree which can dynamically grow or shrink (**Obj. # Satisfied**).

UCF

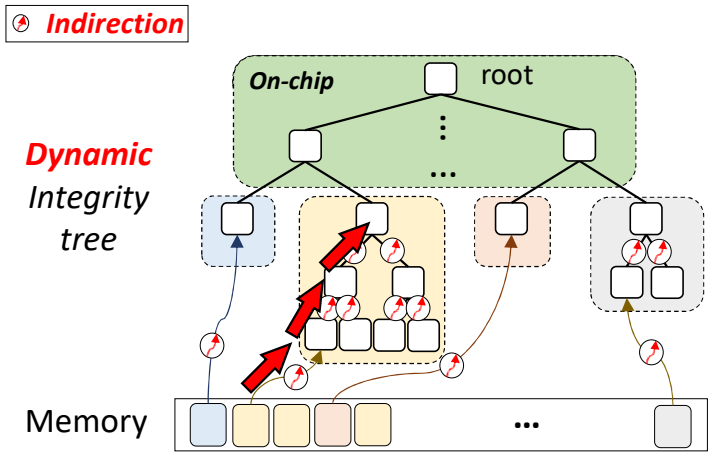
Design Space Exploration: Fully Dynamic Partitioning



- Each domain hosts individual integrity sub-tree which can dynamically grow or shrink (**Obj. # Satisfied**).
- Can resize integrity coverage of domains based on memory footprint (**Obj. Satisfied**).

UCF

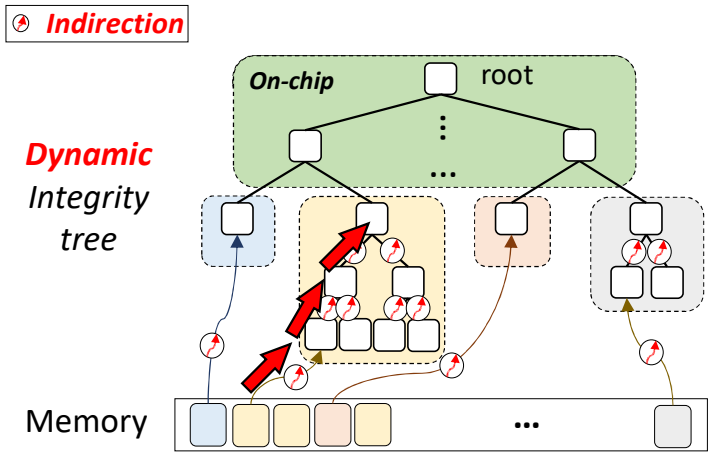
Design Space Exploration: Fully Dynamic Partitioning



- Each domain hosts individual integrity sub-tree which can dynamically grow or shrink (**Obj. # Satisfied**).
- Can resize integrity coverage of domains based on memory footprint (**Obj. Satisfied**).
- **Multiple indirections for single integrity verification** (1x data-to-leaf mapping, n-x leaf-to-root mapping).

UCF

Design Space Exploration: Fully Dynamic Partitioning



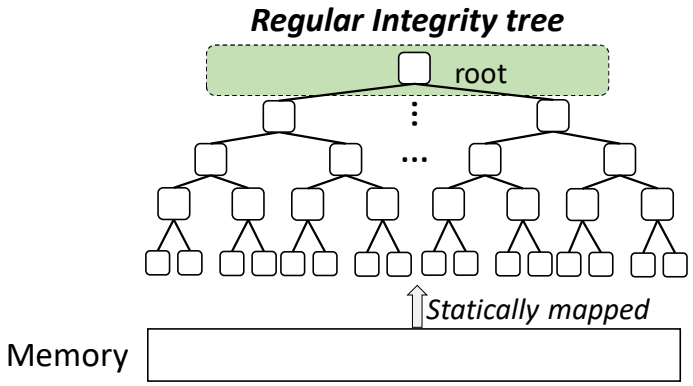
- Each domain hosts individual integrity sub-tree which can dynamically grow or shrink (**Obj. # Satisfied**).
- Can resize integrity coverage of domains based on memory footprint (**Obj. Satisfied**).
- **Multiple indirections for single integrity verification** (1x data-to-leaf mapping, n-x leaf-to-root mapping).

Prohibitive overheads of runtime integrity verification→ high-overhead of tree maintenance (**Obj. $ not-satisfied**).

UCF

IvLeague Conceptual Overview

Split the integrity tree into many ***small but fixed-sized*** sub-trees (**TreeLing**).
Roots of the TreeLings are always ***kept on-chip***.

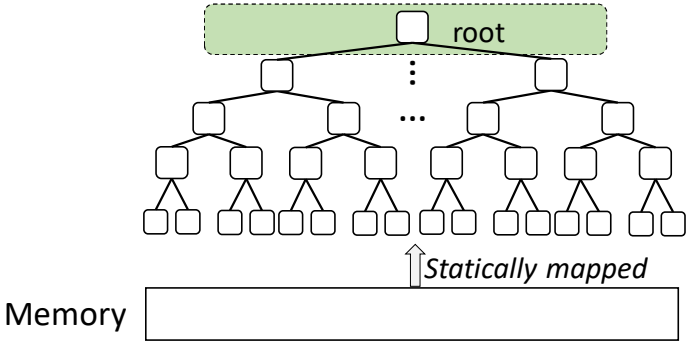


UCF

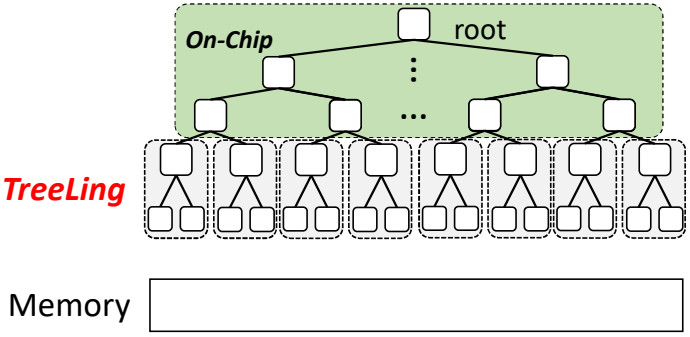
IvLeague Conceptual Overview

Split the integrity tree into many ***small but fixed-sized*** sub-trees (**TreeLing**).
Roots of the TreeLings are always ***kept on-chip***.

Regular Integrity tree



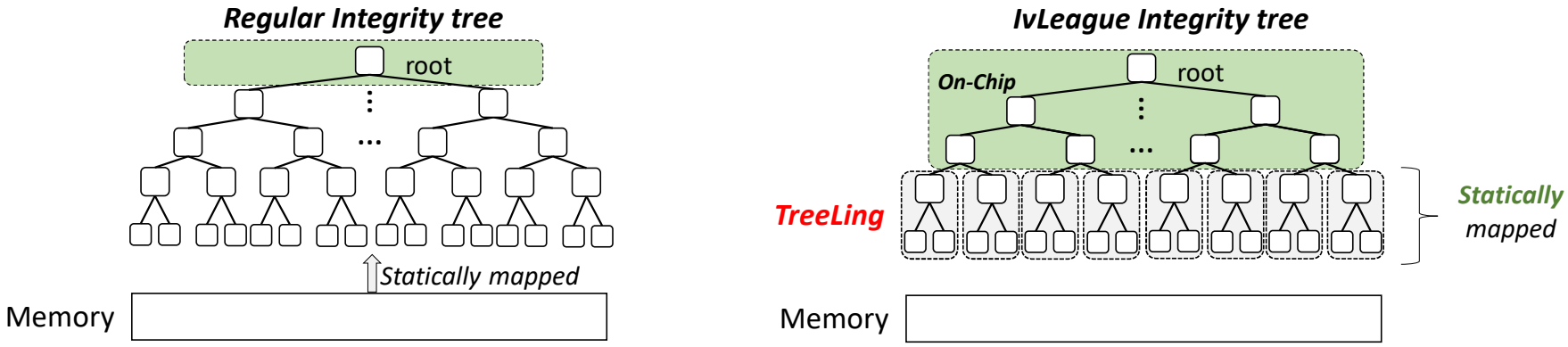
IvLeague Integrity tree



UCF

IvLeague Conceptual Overview

Split the integrity tree into many ***small but fixed-sized*** sub-trees (**TreeLing**).
Roots of the TreeLings are always ***kept on-chip***.

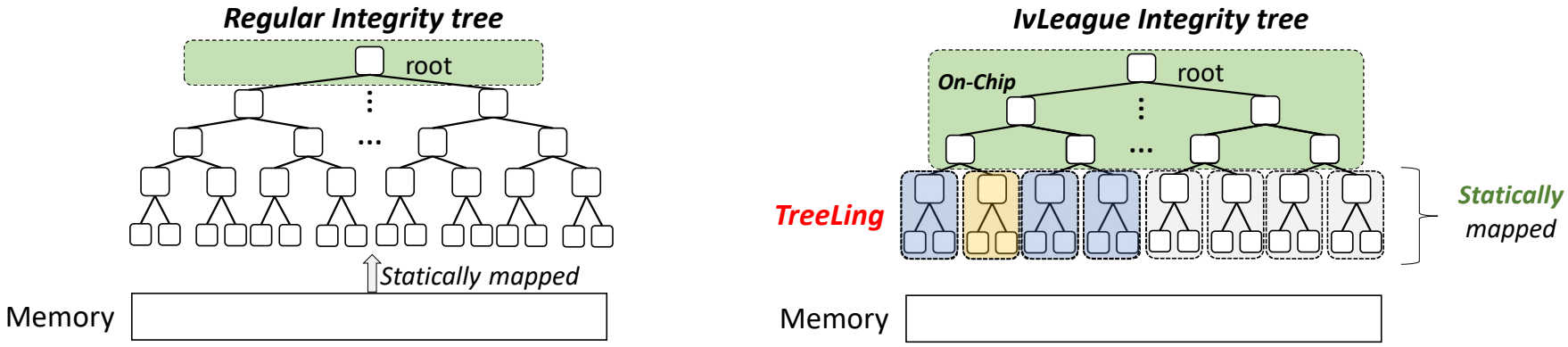


Each sub-tree (TreeLing) is statically mapped, ***no indirection needed for leaf-to-root traversal***.

UCF

IvLeague Conceptual Overview

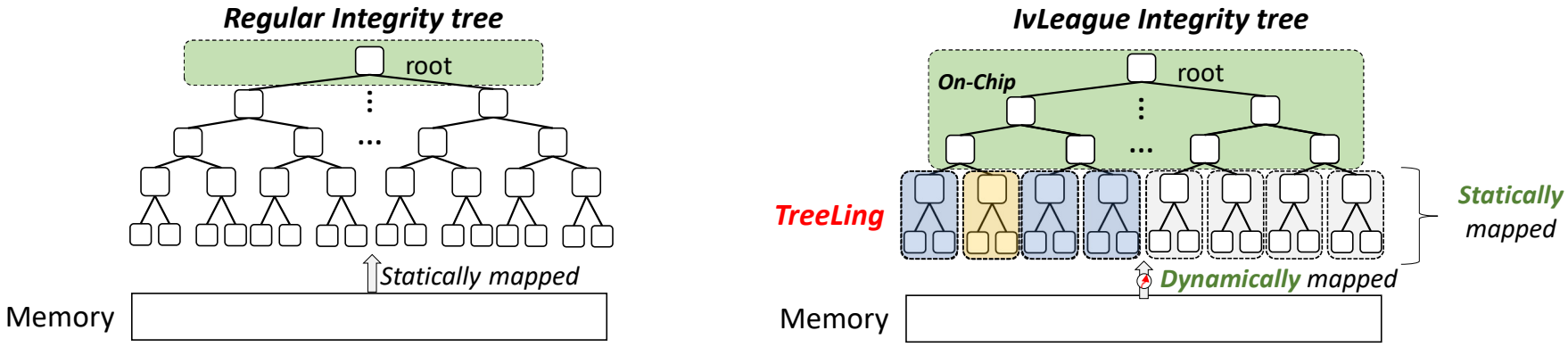
Split the integrity tree into many ***small but fixed-sized*** sub-trees (**TreeLing**).
Roots of the TreeLings are always ***kept on-chip***.



- Each sub-tree (TreeLing) is statically mapped, ***no indirection needed for leaf-to-root traversal***.
- TreeLings are allocated to domain on-demand, ***resize integrity coverage during runtime***.

UCF

IvLeague Conceptual Overview

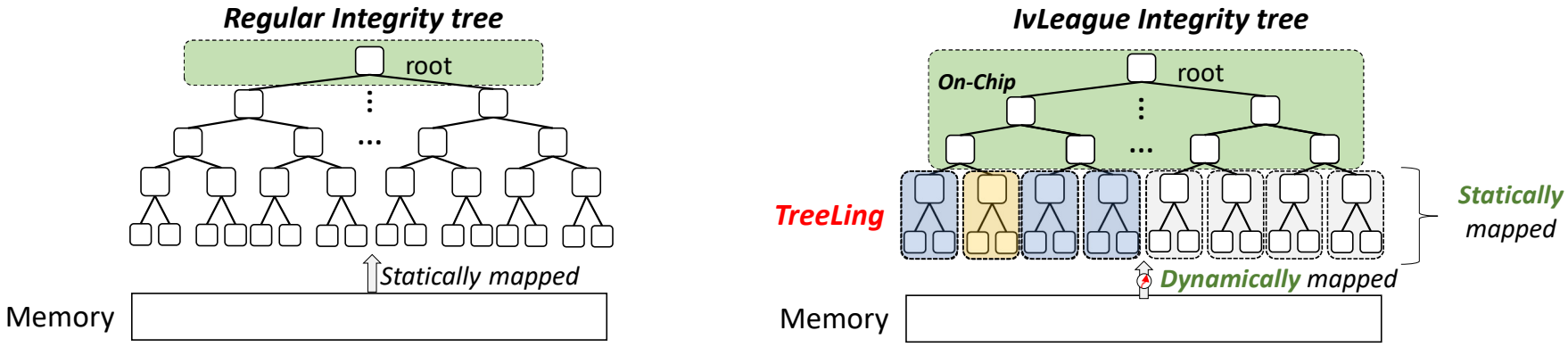


- Each sub-tree (TreeLing) is statically mapped, ***no indirection needed for leaf-to-root traversal***.
- TreeLings are allocated to domain on-demand, ***resize integrity coverage during runtime***.
- Supports as many domains as fully dynamic partitioning, ***scalable for larger domain support***.

UCF

IvLeague Conceptual Overview

Split the integrity tree into many ***small but fixed-sized*** sub-trees (**TreeLing**).
Roots of the TreeLings are always ***kept on-chip***.



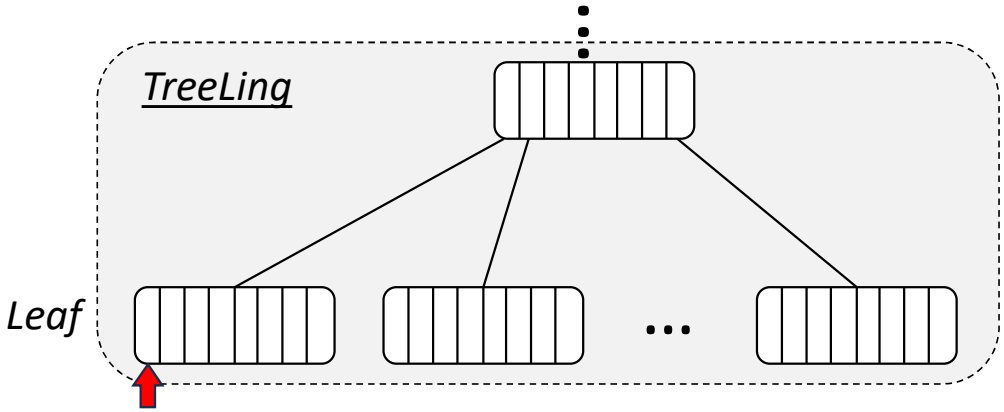
Challenges:

- ***Intra-TreeLing management:*** Efficient hardware-controlled allocation policy for TreeLing nodes.
- ***Inter-TreeLing management:*** TreeLing configuration (# and size of TreeLing) considering diverse memory footprints.

UCF

Need for Efficient Page to Tree Node Mapping Mechanism

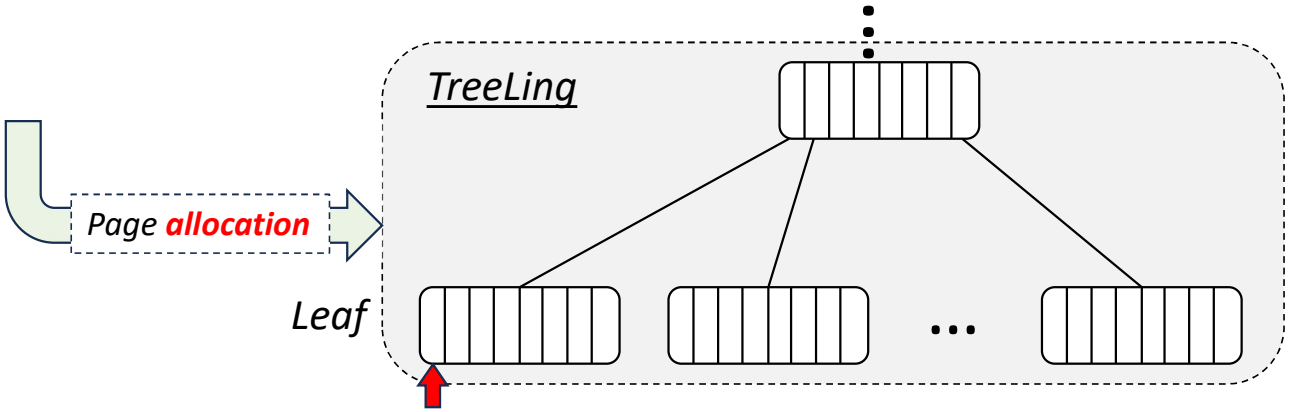
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

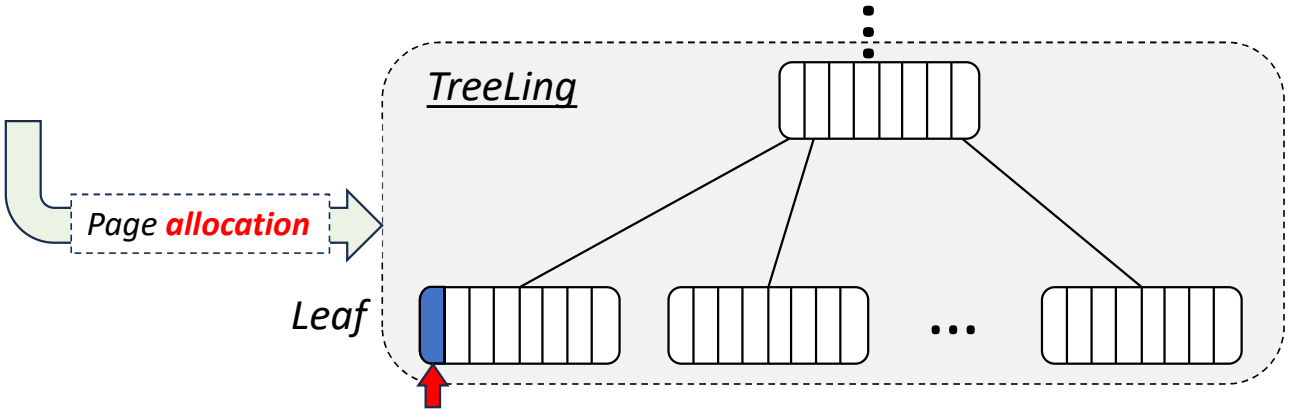
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

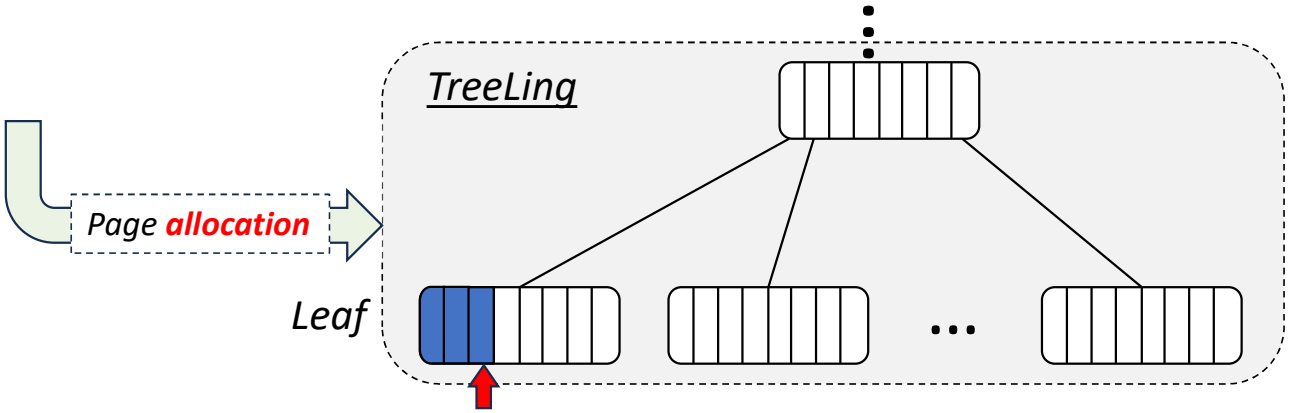
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

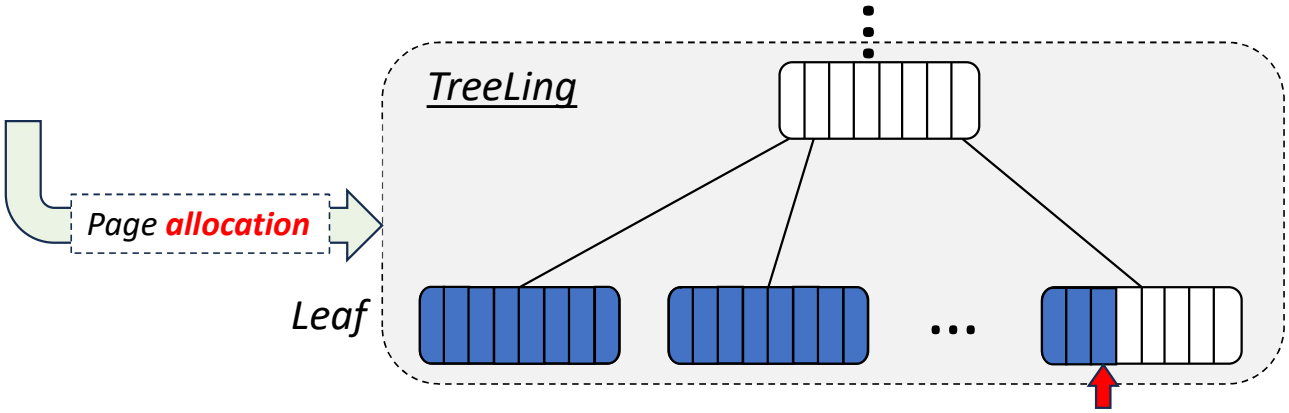
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

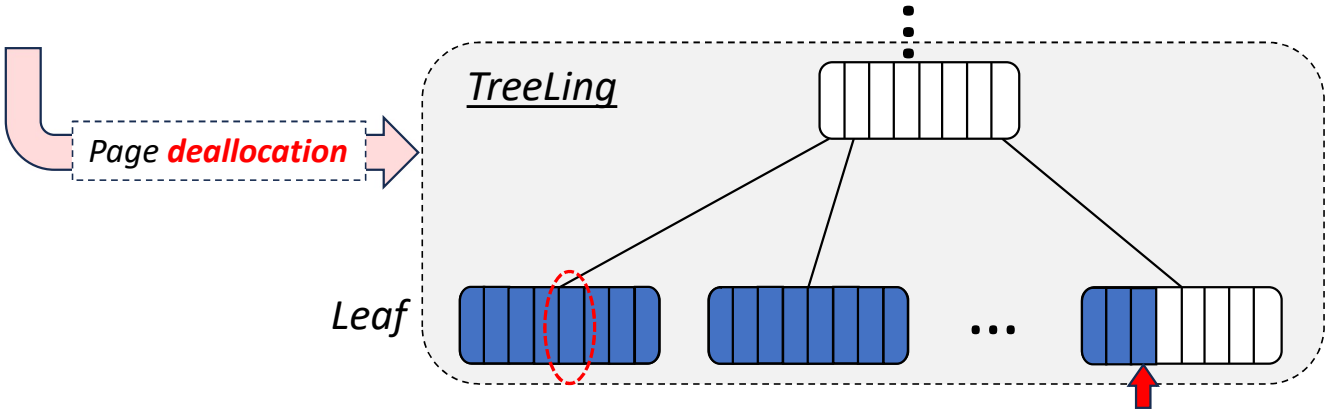
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

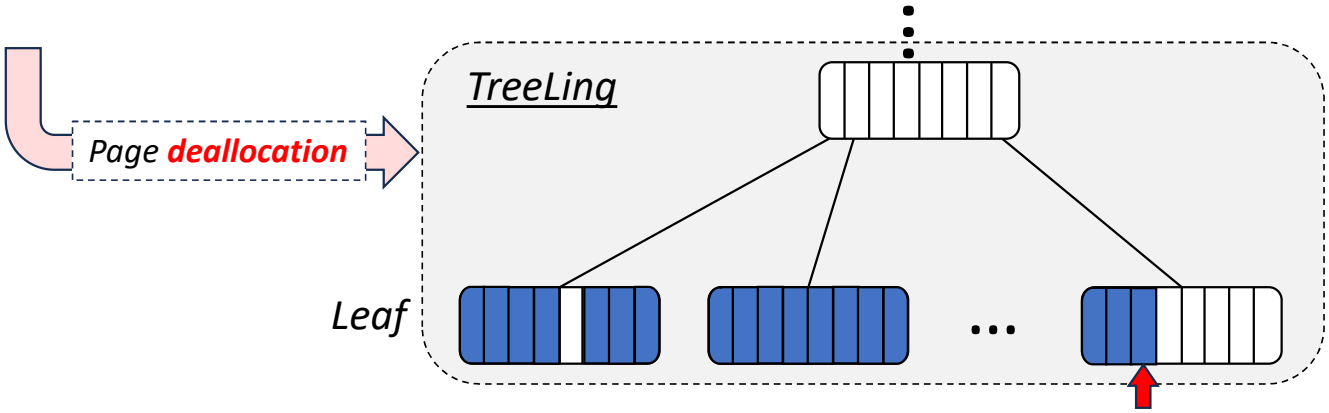
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

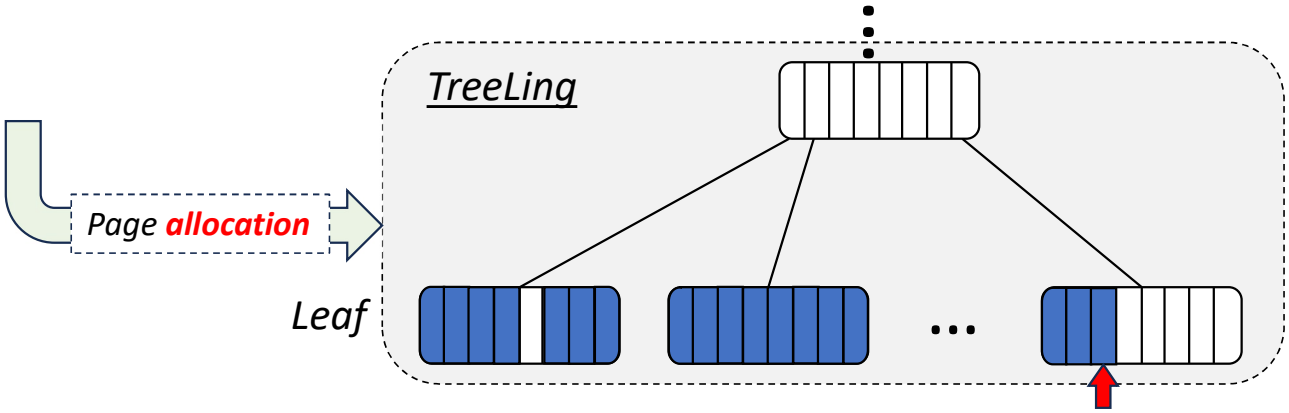
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

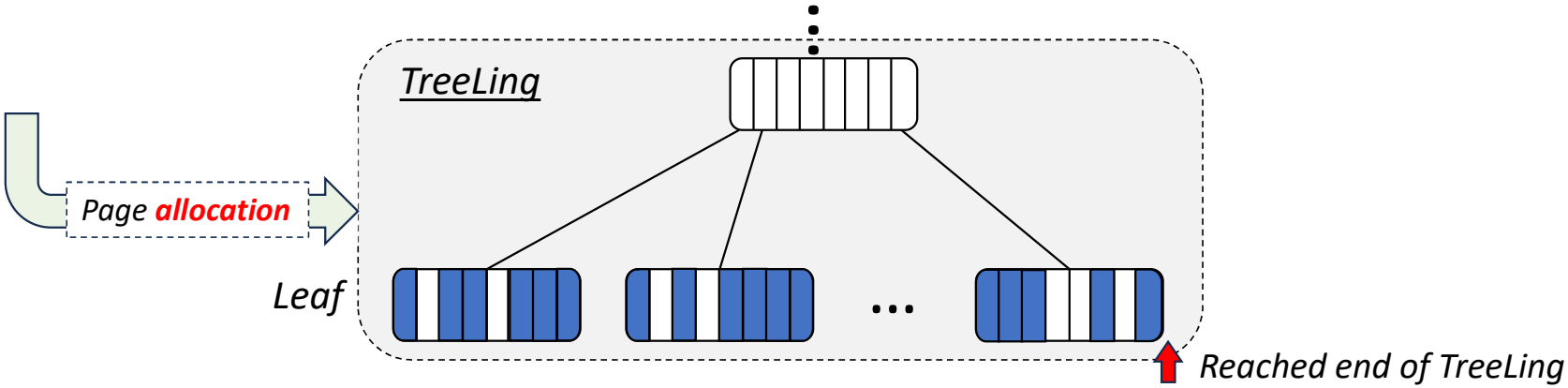
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

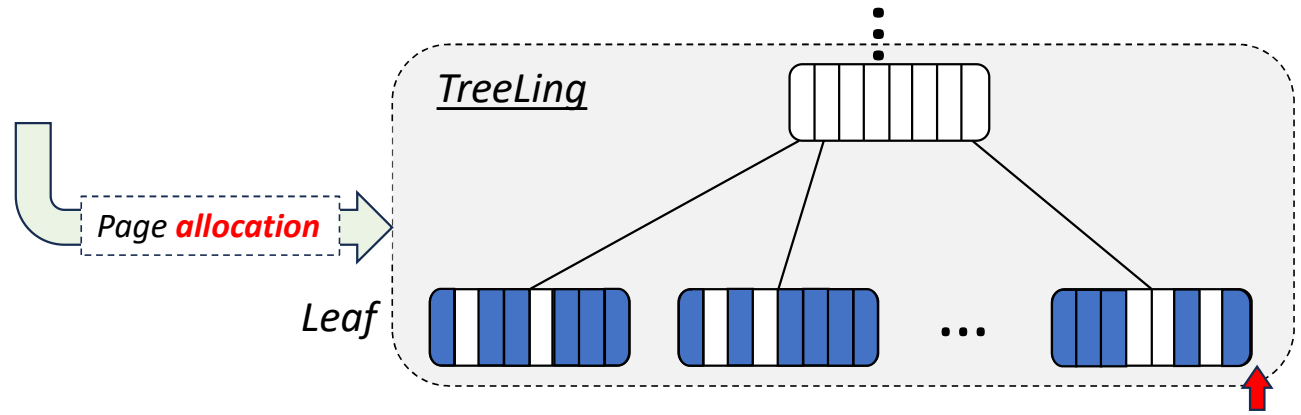
Problems with simple page to tree node mapping mechanism:



UCF

Need for Efficient Page to Tree Node Mapping Mechanism

Problems with simple page to tree node mapping mechanism:

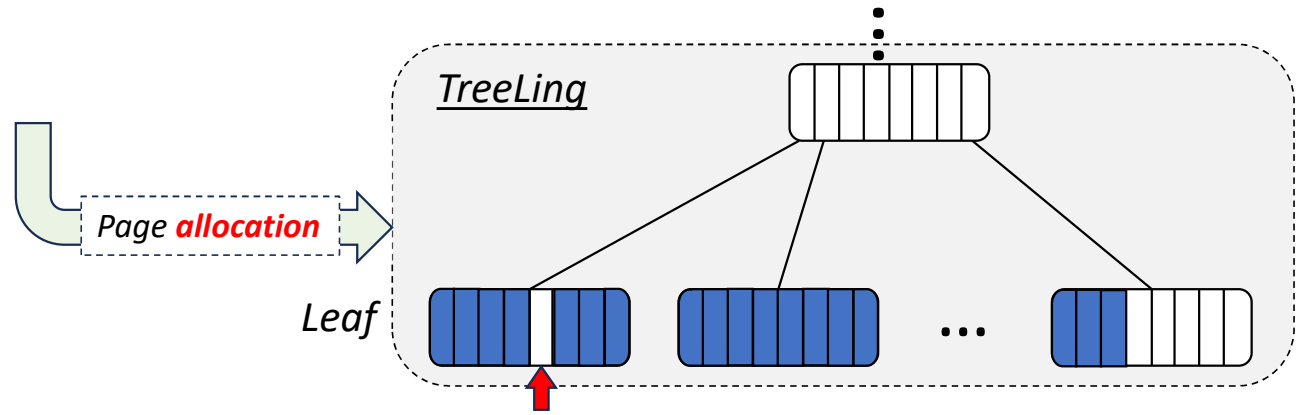


Runtime memory deallocation behavior will result in ***underutilization*** of the tree.

UCF

Need for Efficient Page to Tree Node Mapping Mechanism

Problems with simple page to tree node mapping mechanism:

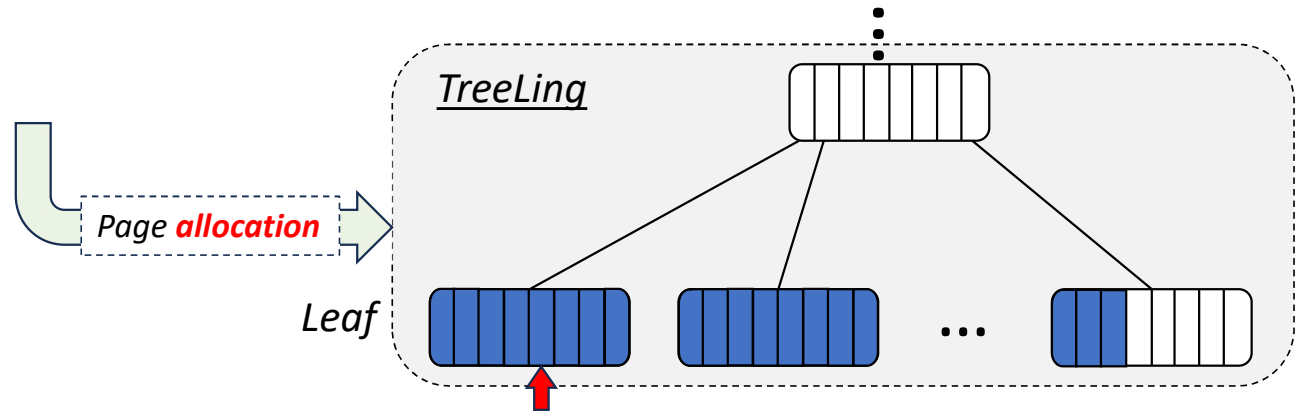


Runtime memory deallocation behavior will result in ***underutilization*** of the tree.

UCF

Need for Efficient Page to Tree Node Mapping Mechanism

Problems with simple page to tree node mapping mechanism:

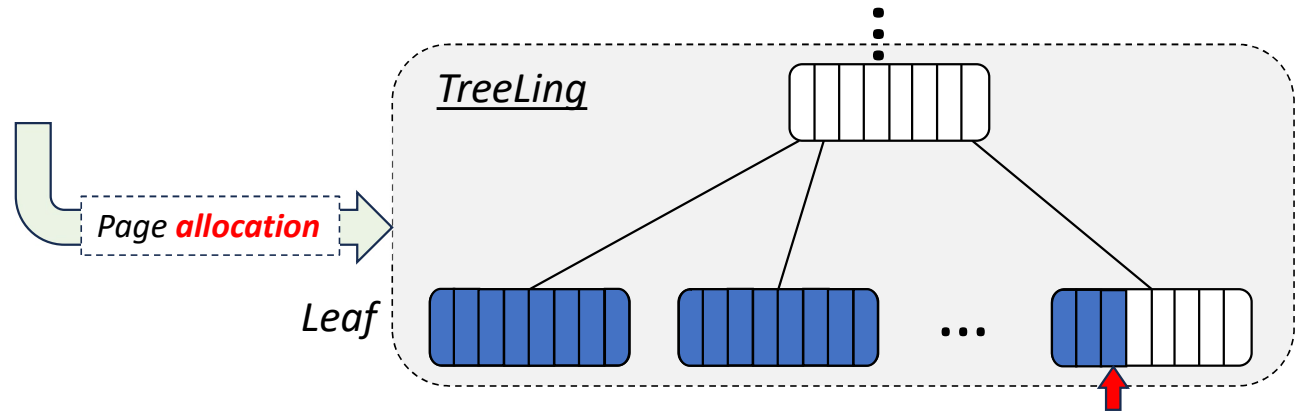


Runtime memory deallocation behavior will result in ***underutilization*** of the tree.

UCF

Need for Efficient Page to Tree Node Mapping Mechanism

Problems with simple page to tree node mapping mechanism:

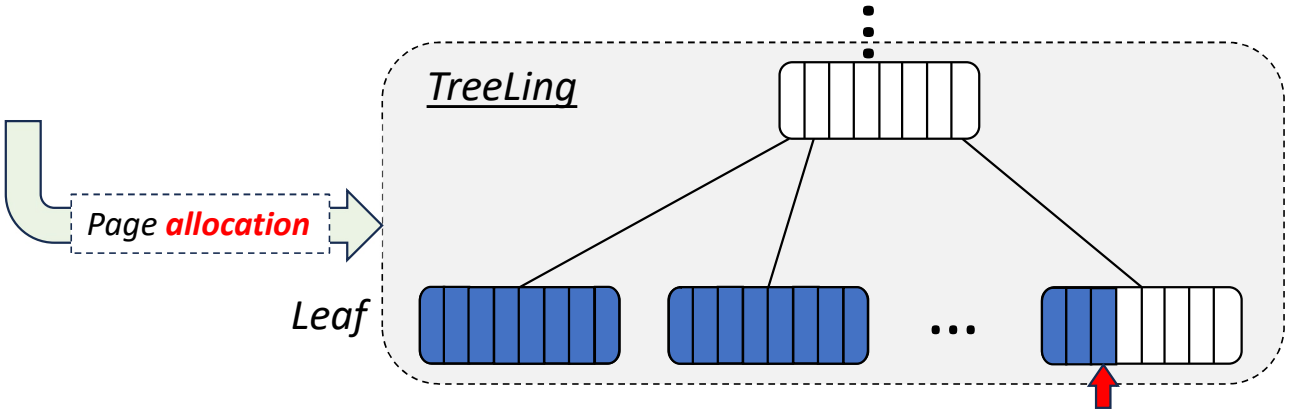


Runtime memory deallocation behavior will result in ***underutilization*** of the tree.

UCF

Need for Efficient Page to Tree Node Mapping Mechanism

Problems with simple page to tree node mapping mechanism:



Runtime memory deallocation behavior will result in ***underutilization*** of the tree.

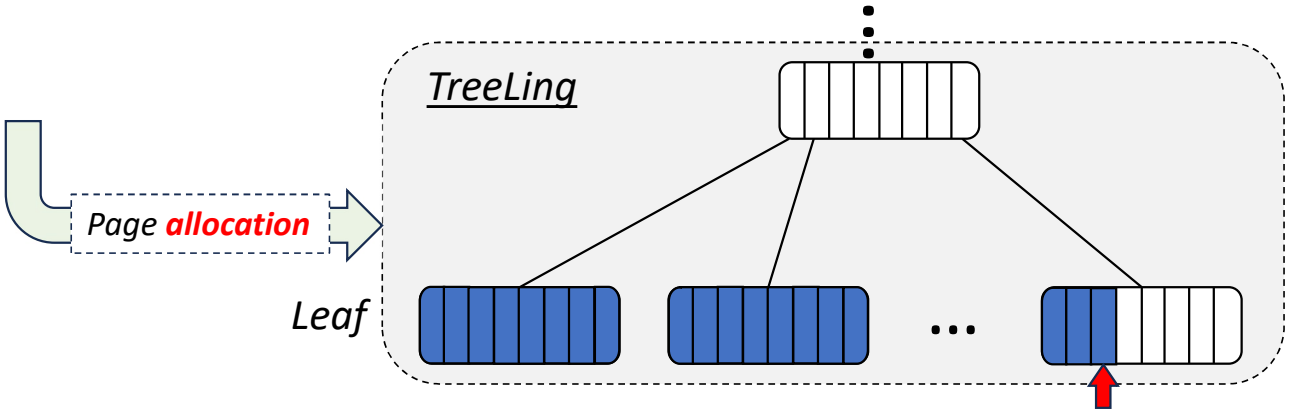
Alternative: *Scan through all leaf nodes* to find an available node **(up to $O(N = \#\ of\ nodes)$** overhead**).**

In critical path of program execution

UCF

Need for Efficient Page to Tree Node Mapping Mechanism

Problems with simple page to tree node mapping mechanism:



Runtime memory deallocation behavior will result in ***underutilization*** of the tree.

Alternative: *Scan through all leaf nodes* to find an available node **(up to $O(N = \#\ of\ nodes)$** overhead**).**

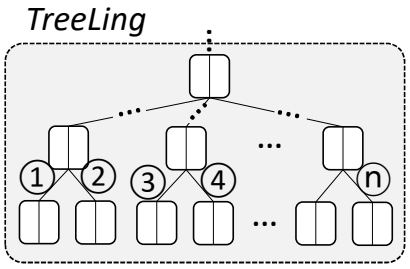
In critical path of program execution

Need ***hardware-only*** design to provide ***efficient and scalable*** TreeLing node mapping to pages.

UCF

Node Free-List: Efficient Mapping of Leaf

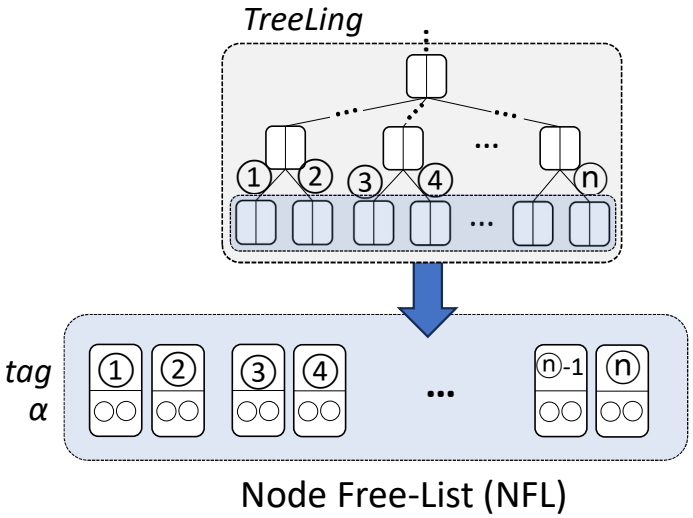
- Track *available leaf nodes* in a Node Free-List (NFL).



UCF

Node Free-List: Efficient Mapping of Leaf

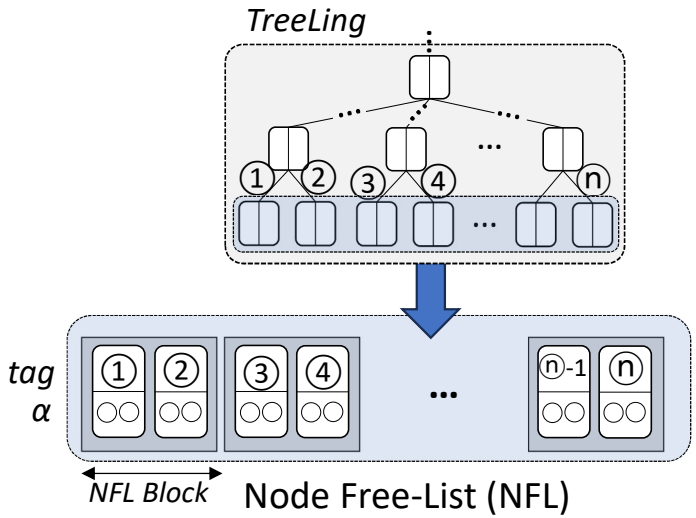
- Track *available leaf nodes* in a Node Free-List (NFL).
- NFL contains *tree node availability vector* for all leaf nodes (○→Available, ●→Unavailable).



UCF

Node Free-List: Efficient Mapping of Leaf

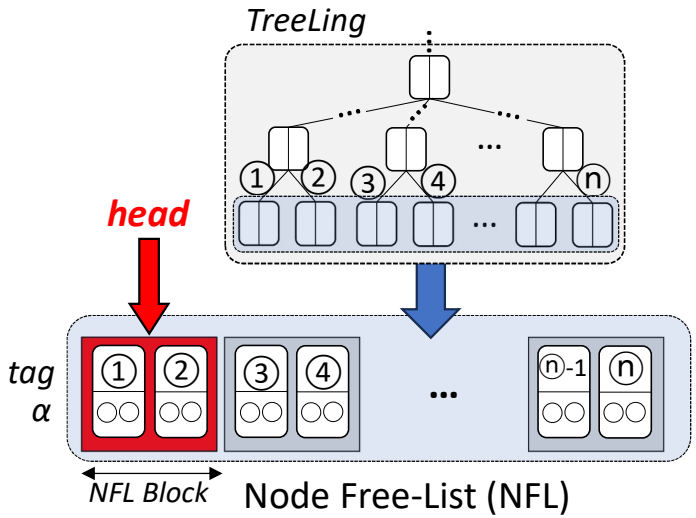
- Track *available leaf nodes* in a Node Free-List (NFL).
- NFL contains *tree node availability vector* for all leaf nodes (○→Available, ●→Unavailable).
- NFLs are grouped together in memory as *NFL blocks*.



UCF

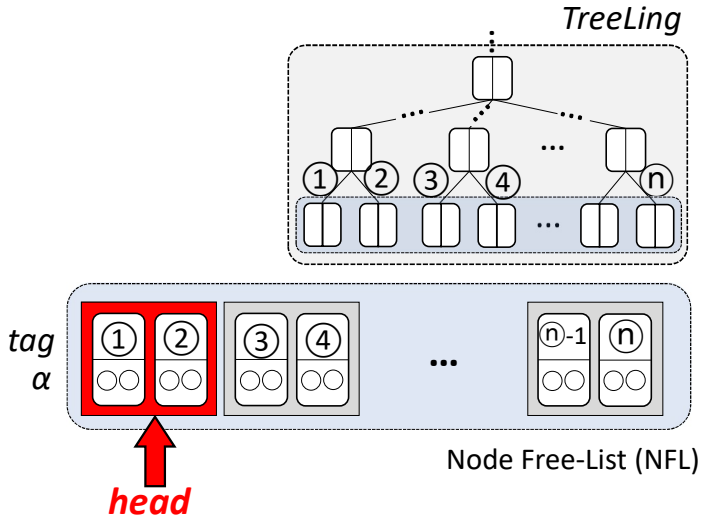
Node Free-List: Efficient Mapping of Leaf

- Track *available leaf nodes* in a Node Free-List (NFL).
- NFL contains *tree node availability vector* for all leaf nodes (○→Available, ●→Unavailable).
- NFLs are grouped together in memory as *NFL blocks*.
- ***head*** register denotes currently active NFL block.



UCF

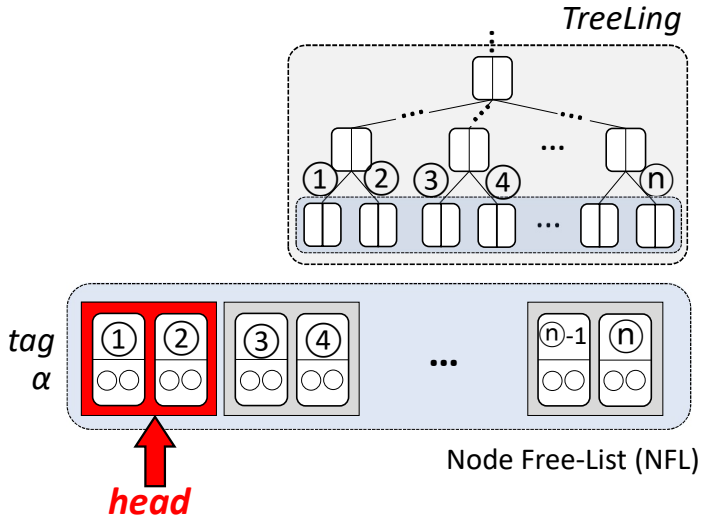
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.

UCF

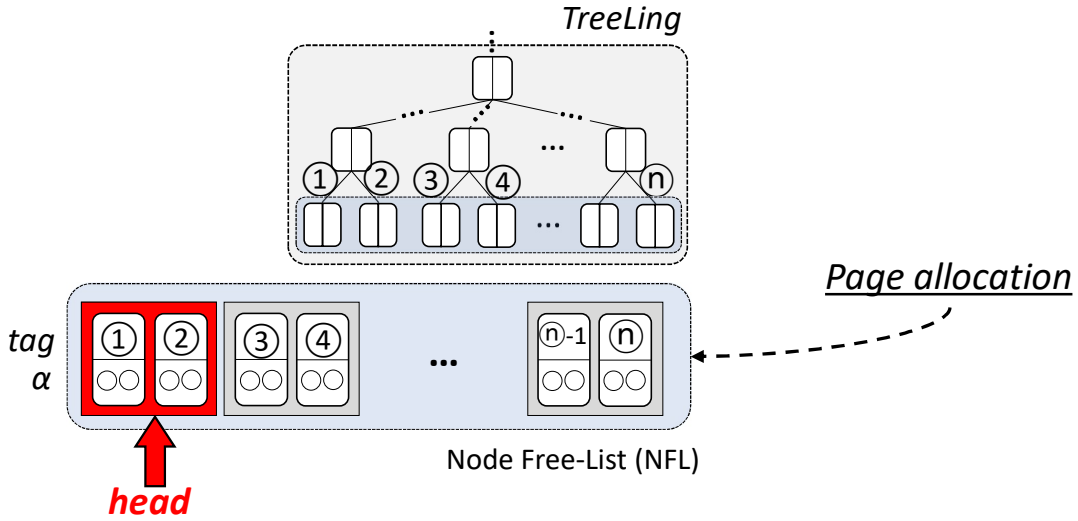
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.

UCF

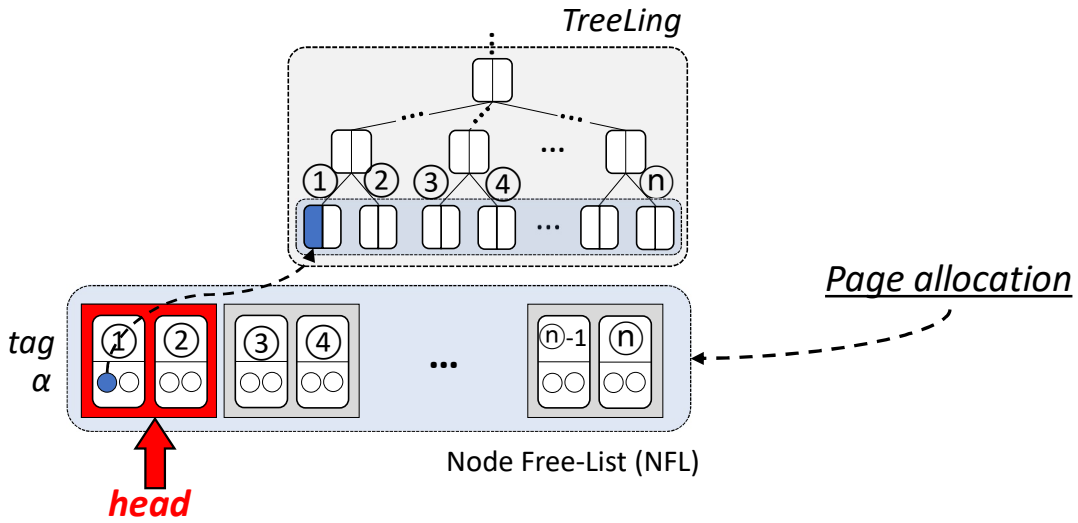
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.

UCF

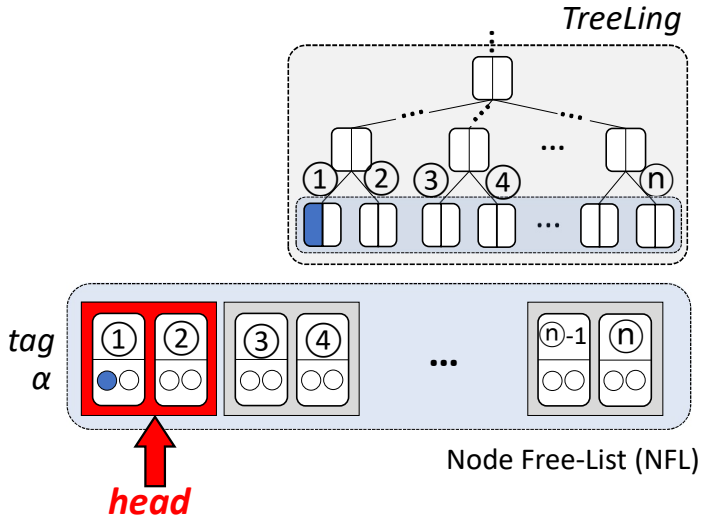
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.

UCF

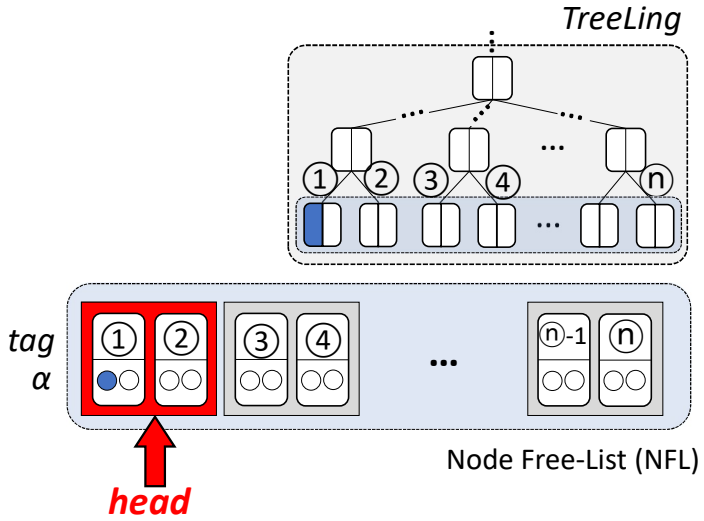
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.

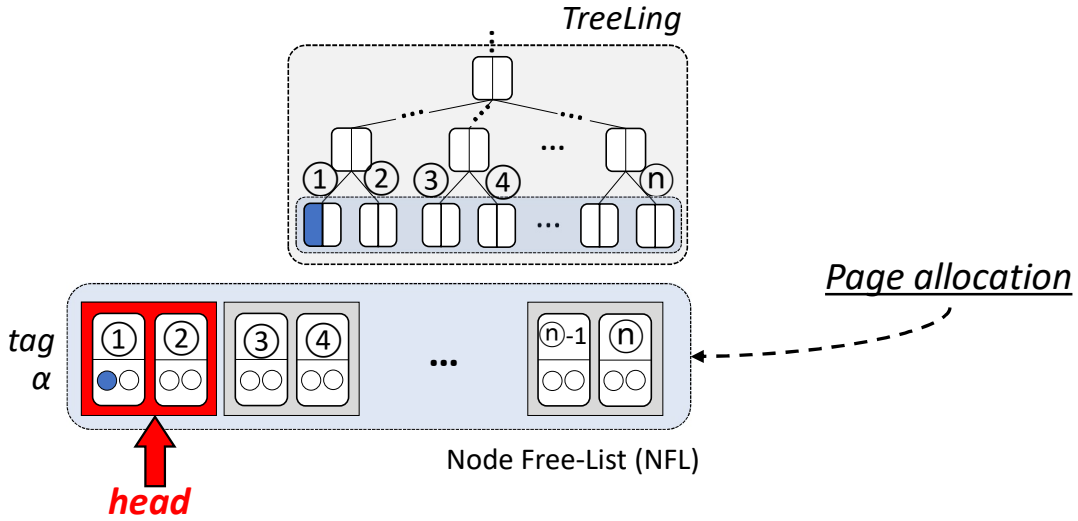
UCF

Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.

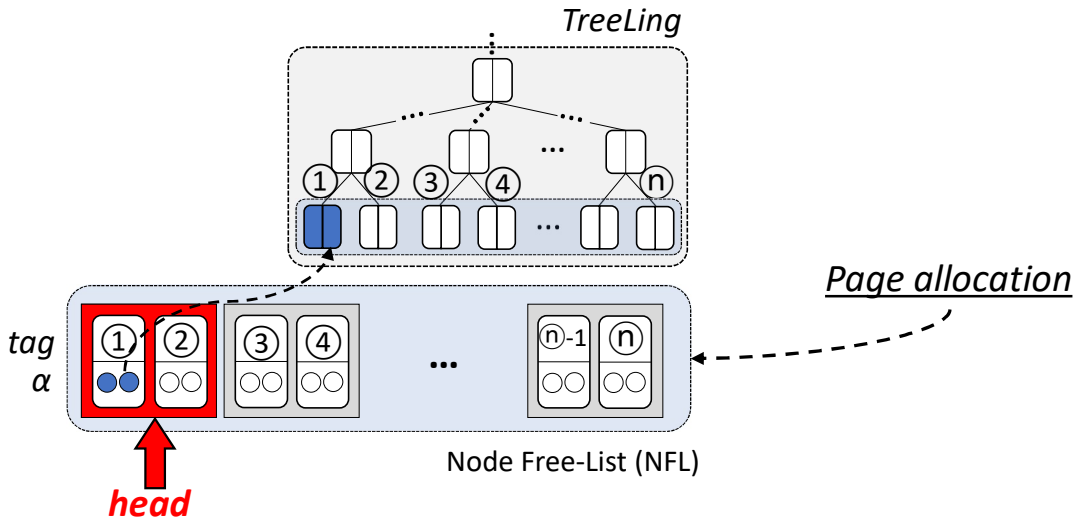
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.

UCF

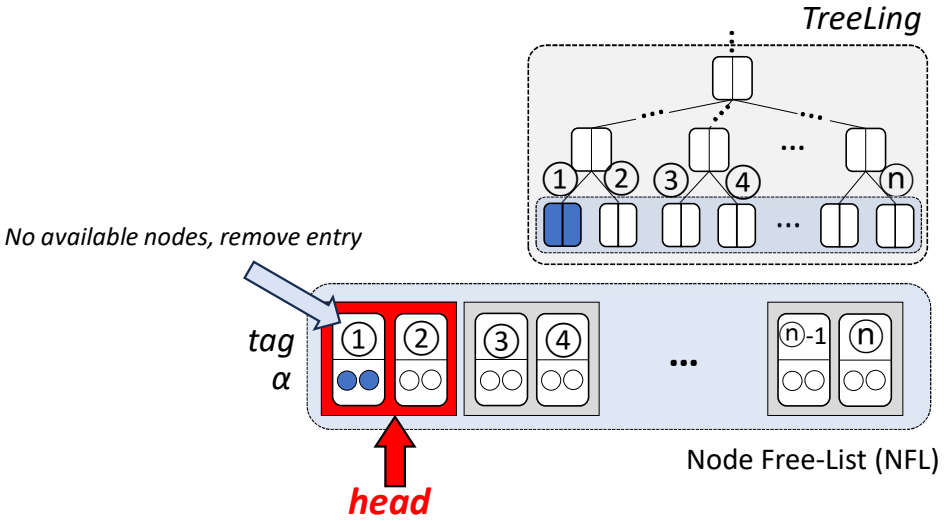
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.

UCF

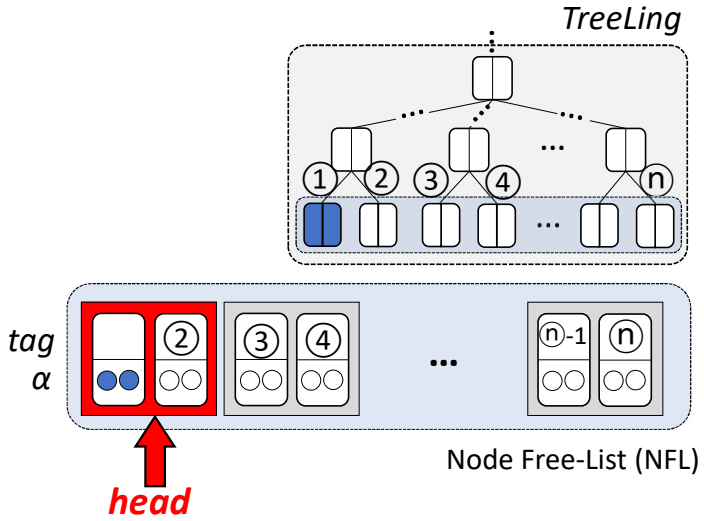
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.

UCF

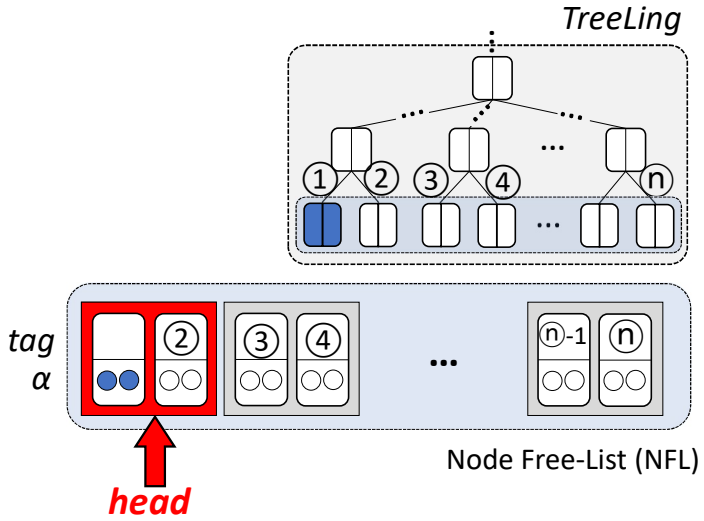
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.

UCF

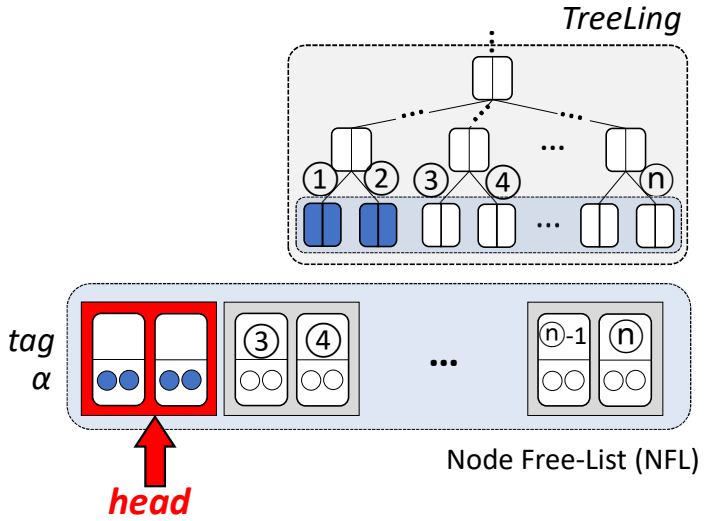
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.
  - Active NFL block ***does not have available node*** → move head to next block (right) and assign.

UCF

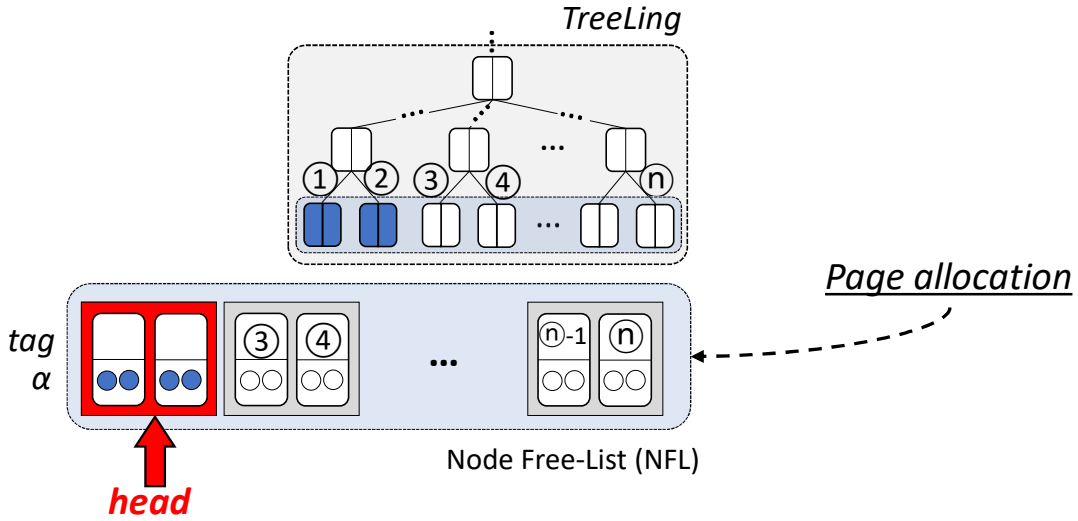
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.
  - Active NFL block ***does not have available node*** → move head to next block (right) and assign.

UCF

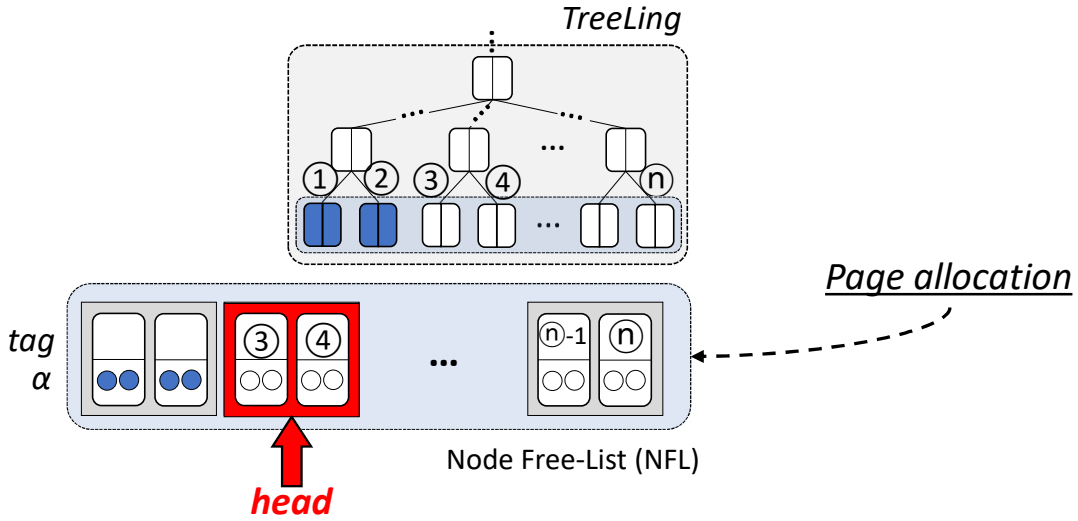
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.
  - Active NFL block ***does not have available node*** → move head to next block (right) and assign.

UCF

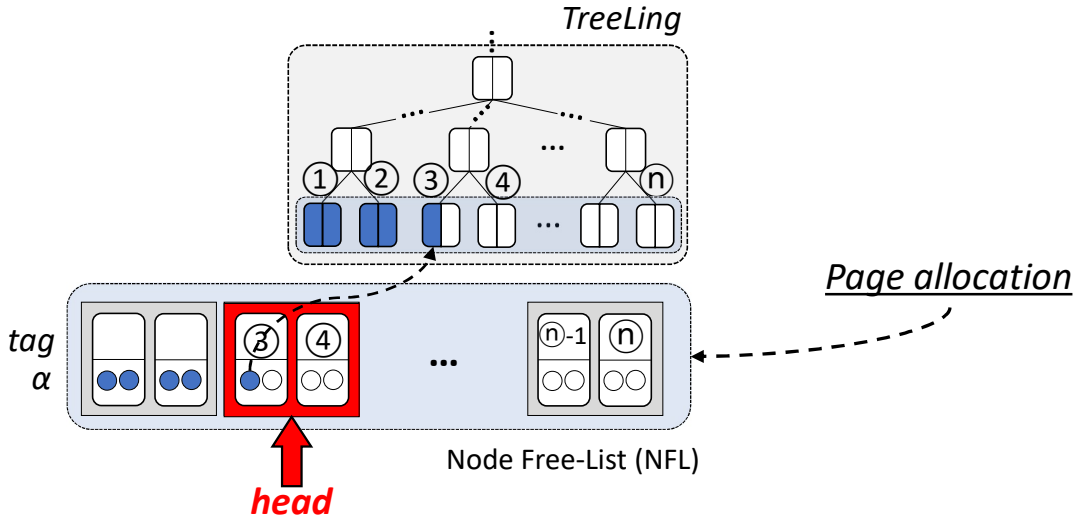
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.
  - Active NFL block ***does not have available node*** → move head to next block (right) and assign.

UCF

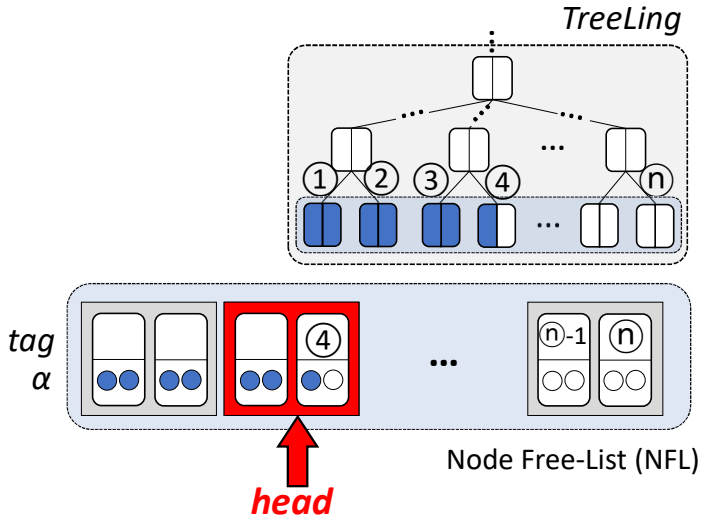
Efficient Mapping of Tree Nodes via NFL



- During program page allocation, NFL tracks *available tree nodes*.
  - Active NFL block ***has available node*** → assign node to page.
    - NFL only tracks available tree nodes. If an NFL entry has no available node, that entry is removed from NFL.
  - Active NFL block ***does not have available node*** → move head to next block (right) and assign.

UCF

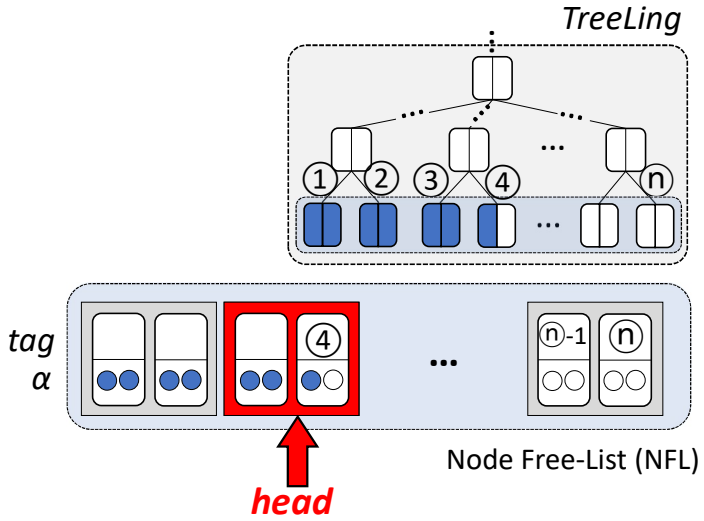
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.

UCF

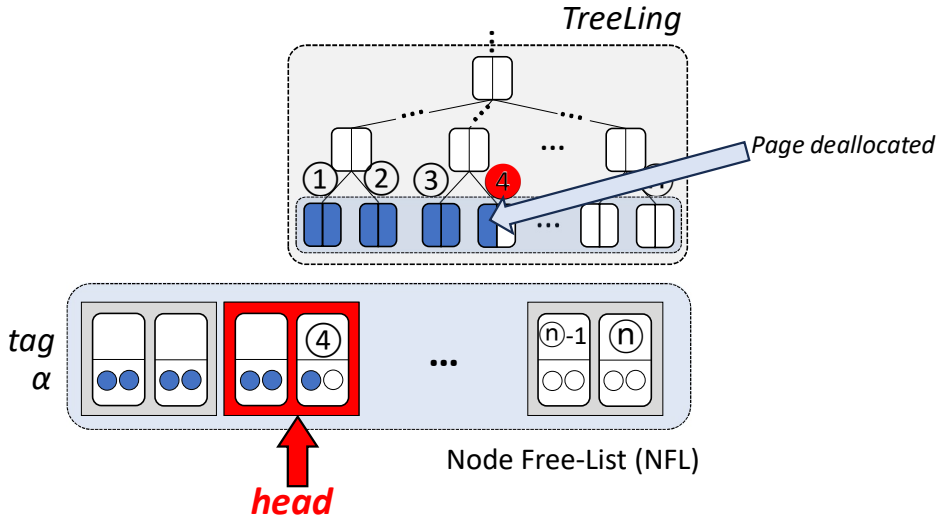
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.

UCF

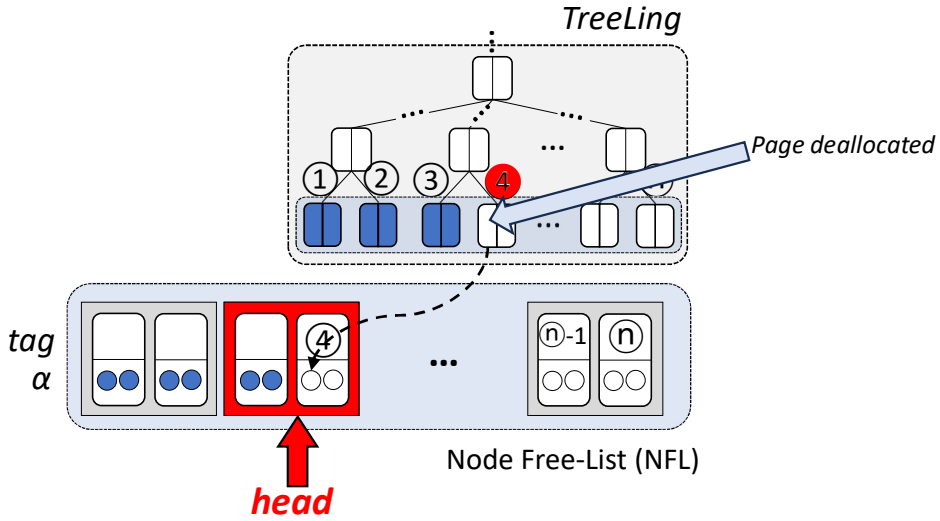
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.

UCF

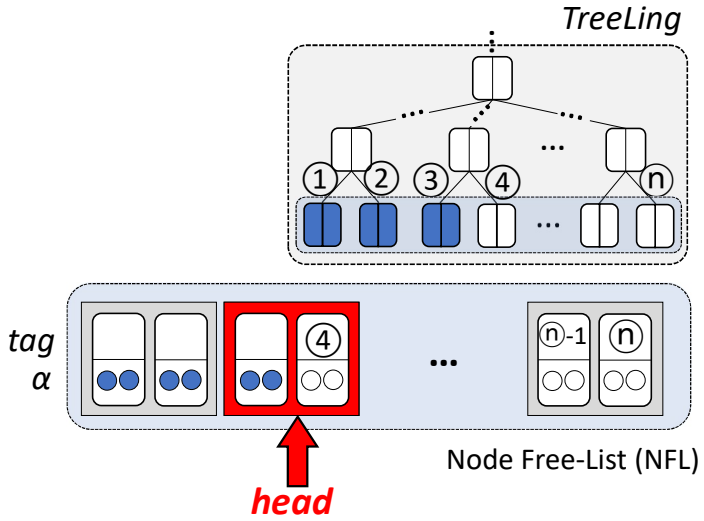
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.

UCF

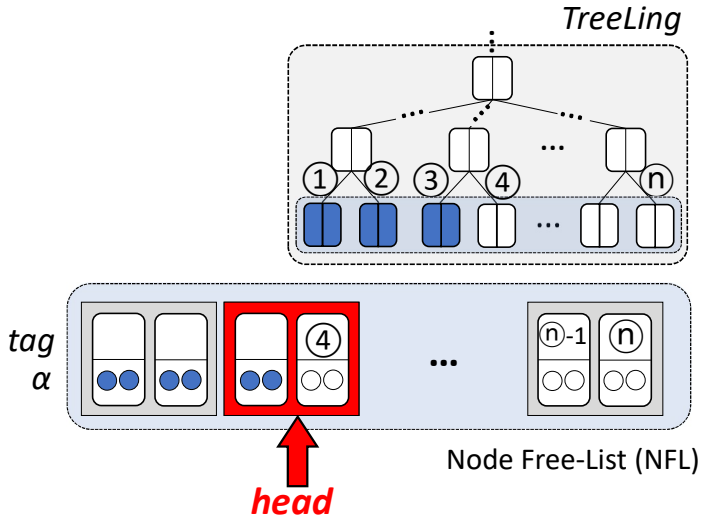
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.

UCF

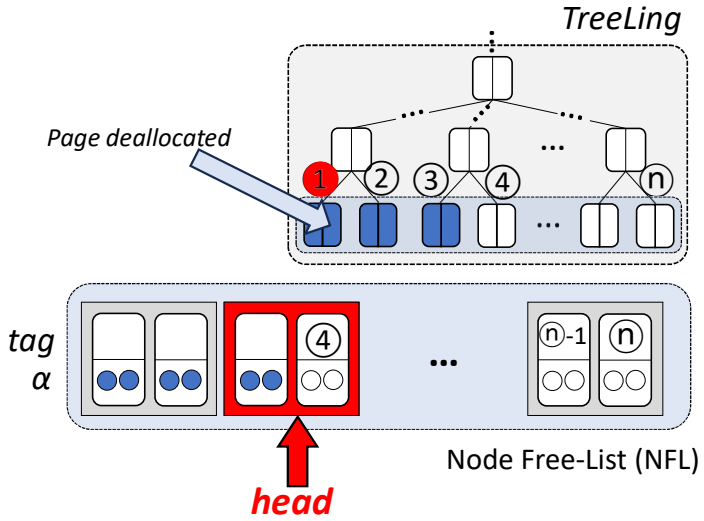
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:

UCF

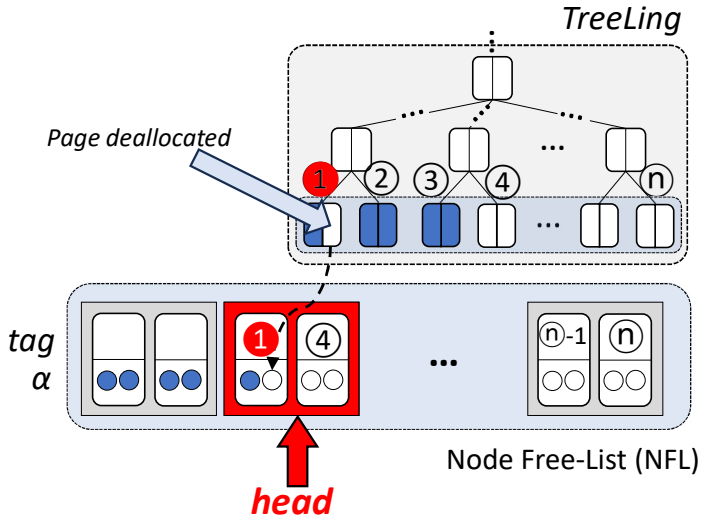
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.

UCF

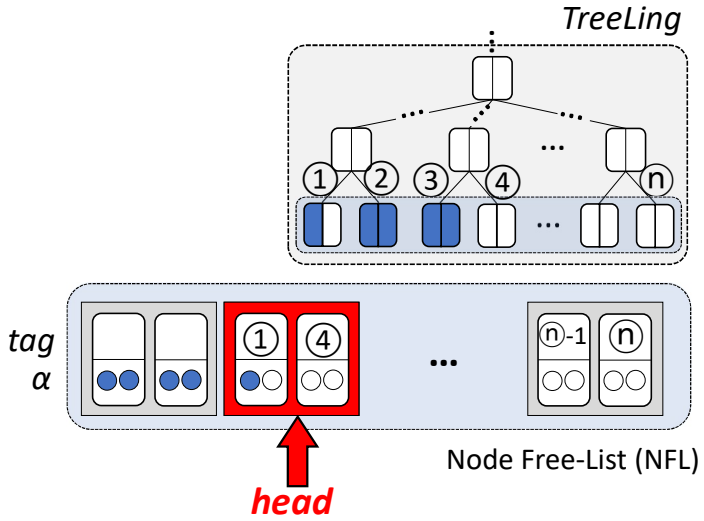
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.

UCF

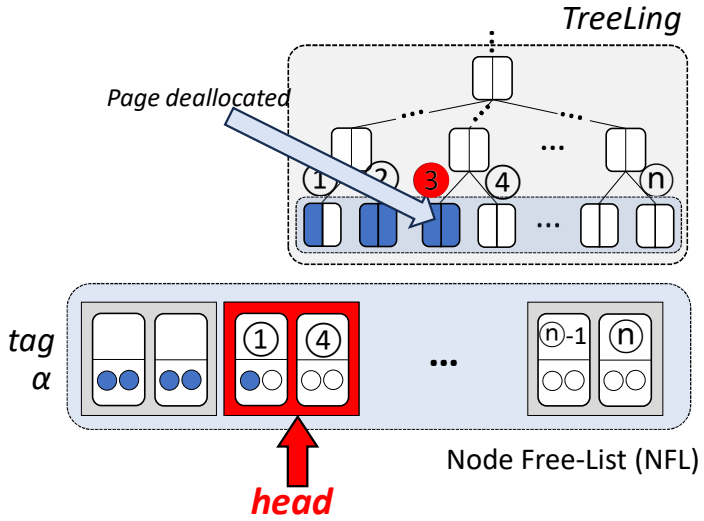
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.

UCF

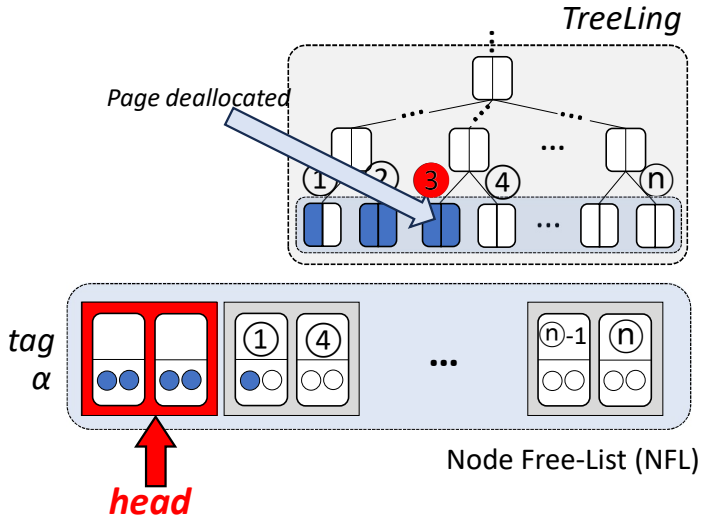
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.
    - Active NFL block ***does not have empty entry*** → move head to *previous block* (left) and track.

UCF

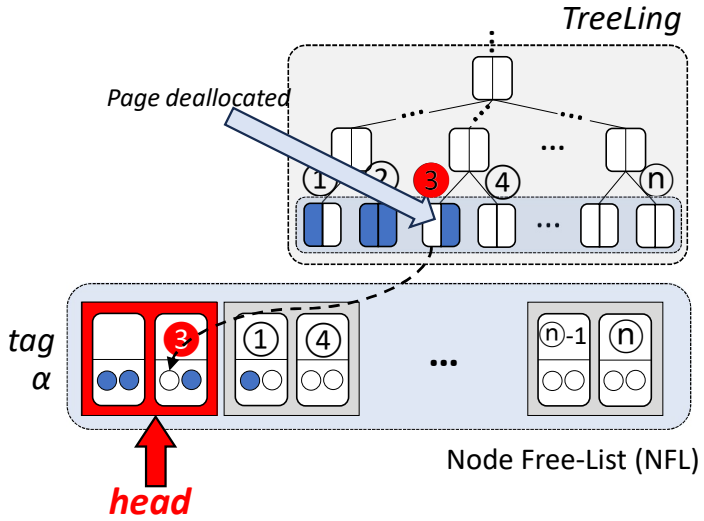
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.
    - Active NFL block ***does not have empty entry*** → move head to *previous block* (left) and track.

UCF

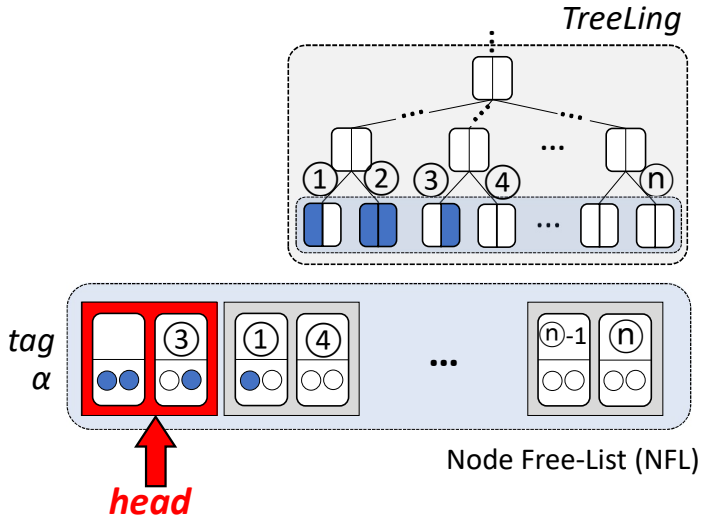
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.
    - Active NFL block ***does not have empty entry*** → move head to *previous block* (left) and track.

UCF

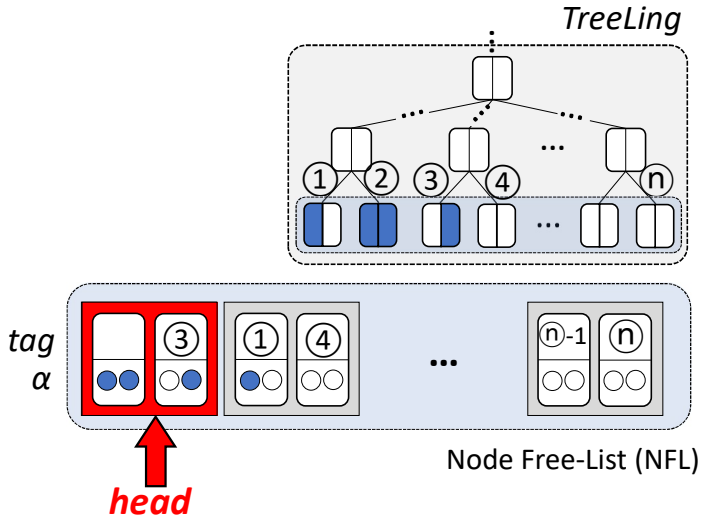
Tracking of Deallocated Tree Nodes via NFL



- When program page is deallocated, NFL marks the tree node as available.
  - Active NFL block ***has tag of deallocated node*** → track deallocation in-place.
  - Active NFL block does not have tag of deallocated node:
    - Active NFL block ***has empty entry*** → track in place of empty entry.
    - Active NFL block ***does not have empty entry*** → move head to *previous block* (left) and track.

UCF

Tracking of Deallocated Tree Nodes via NFL



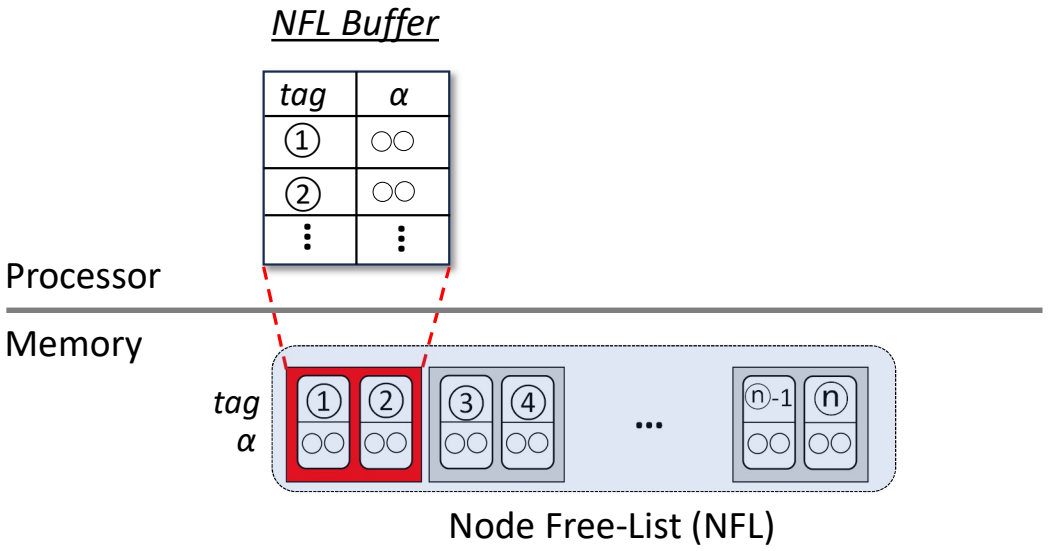
- When program page is deallocated, NFL marks the tree node as available.
  - Ac
  - Active NFL block does not have tag of deallocated node.
    - Active NFL block ***has empty entry*** → track in place of empty entry.
    - Active NFL block ***does not have empty entry*** → move head to *previous block* (left) and track.

Maximum of ***O(1) overhead*** for locating an available/empty slot for allocation/deallocation.

UCF

Runtime Maintenance of NFL Blocks

- During runtime, NFL Buffer caches most recently used NFL blocks (i.e., *active NFL block).*
- *NFL buffer* is a CAM structure.



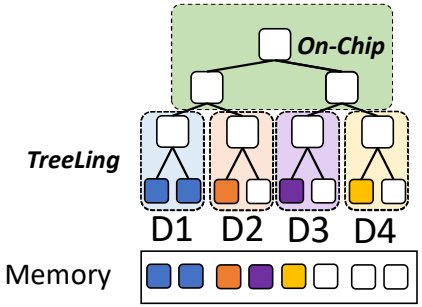
UCF

Determining Optimal TreeLing Configuration

- Real-world workloads has skewed memory distributions.

Determining Optimal TreeLing Configuration

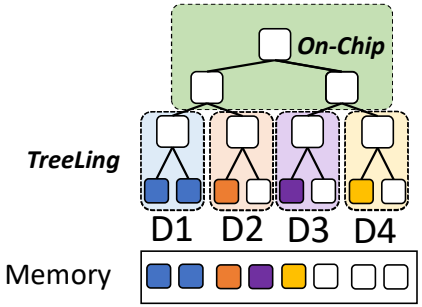
- Real-world workloads has skewed memory distributions.



UCF

Determining Optimal TreeLing Configuration

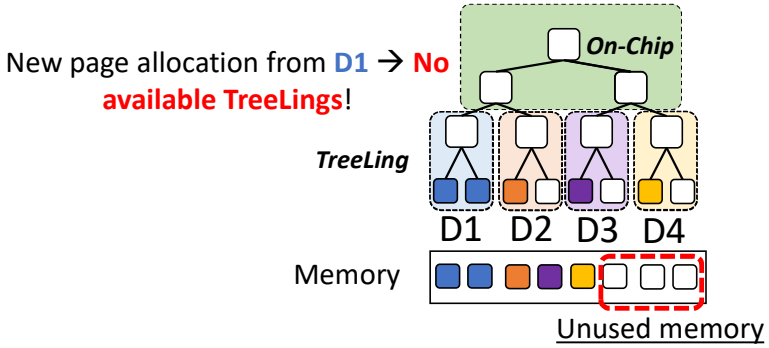
- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.



UCF

Determining Optimal TreeLing Configuration

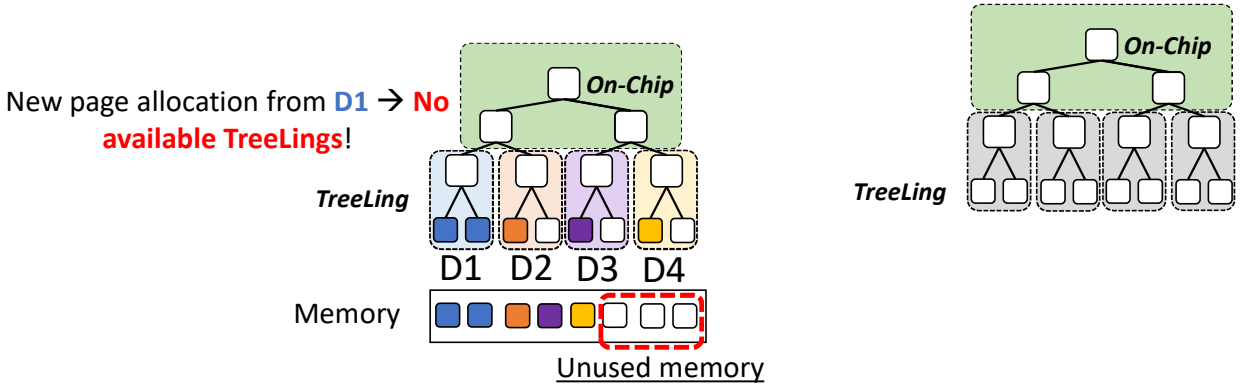
- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.



UCF

Determining Optimal TreeLing Configuration

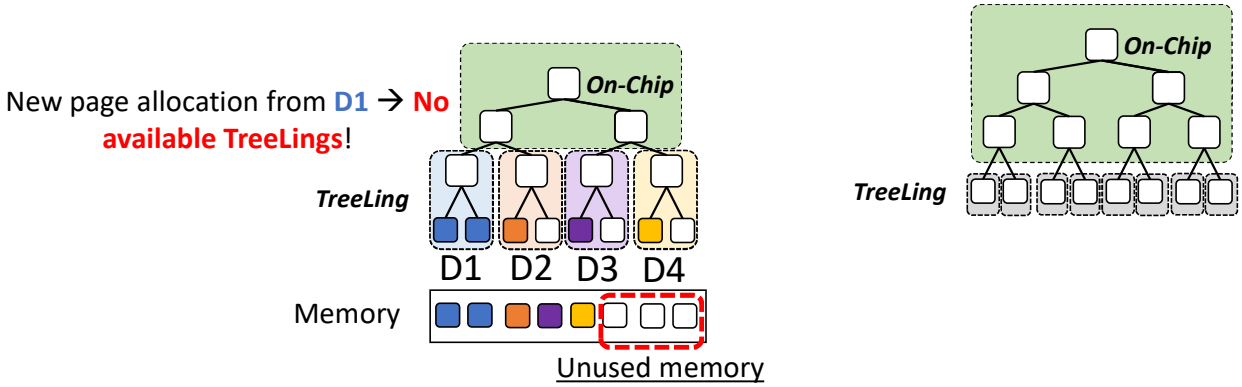
- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.



UCF

Determining Optimal TreeLing Configuration

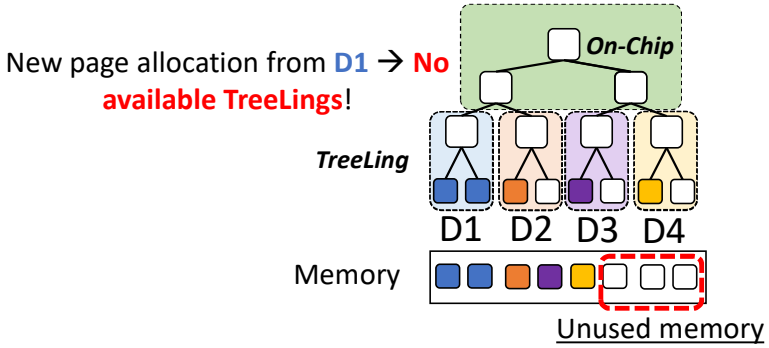
- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.

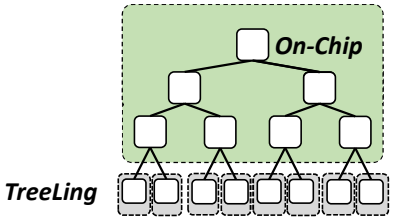


UCF

Determining Optimal TreeLing Configuration

- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.



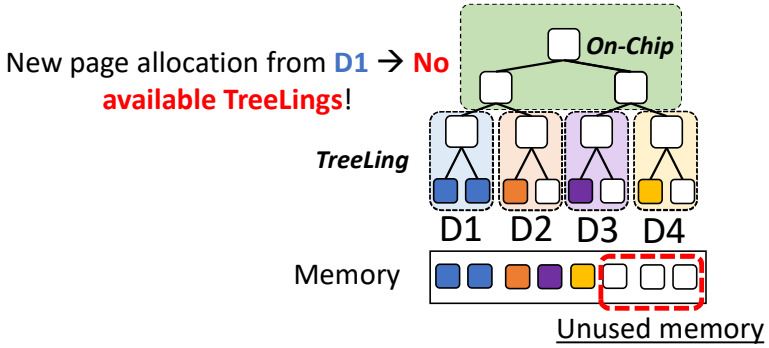


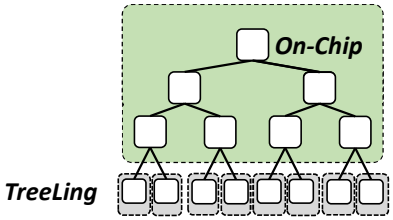
More TreeLings available.
Higher on-chip overheads.

UCF

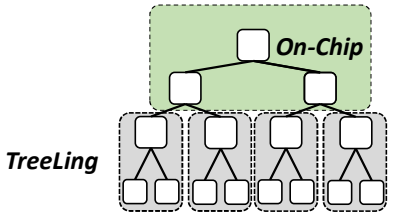
Determining Optimal TreeLing Configuration

- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.





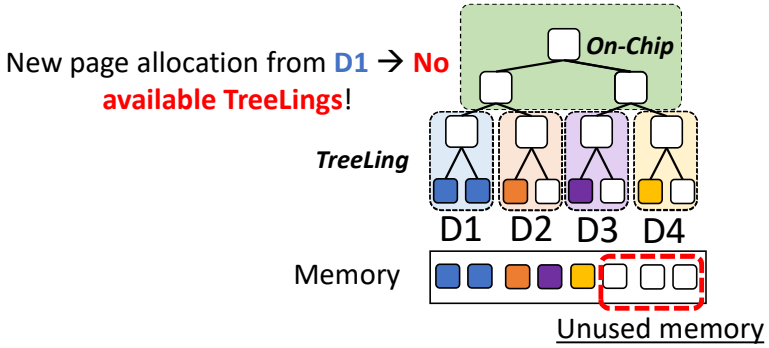
More TreeLings available.
Higher on-chip overheads.

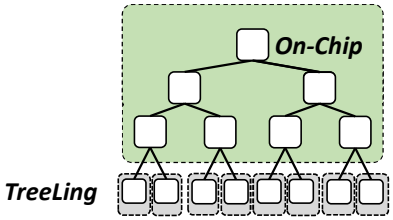


UCF

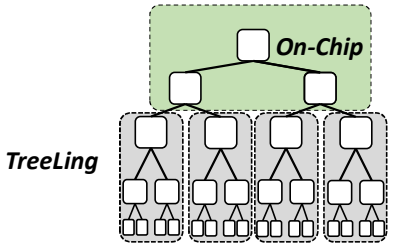
Determining Optimal TreeLing Configuration

- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.





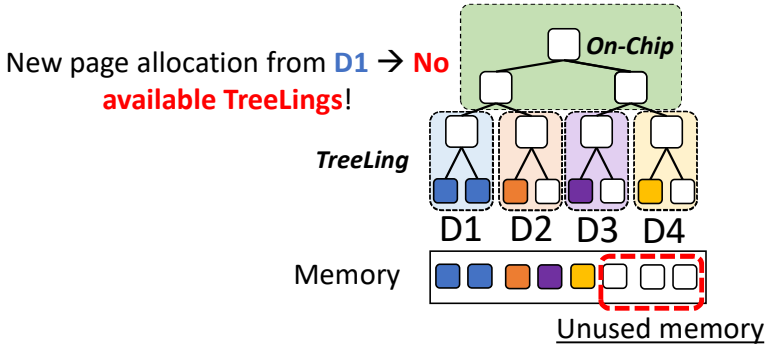
More TreeLings available.
Higher on-chip overheads.

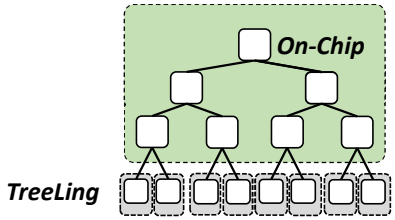


UCF

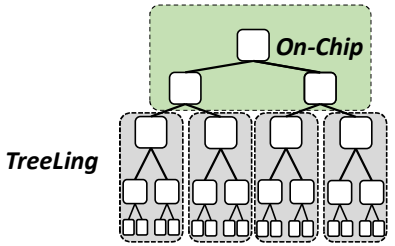
Determining Optimal TreeLing Configuration

- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.





More TreeLings available.
Higher on-chip overheads.

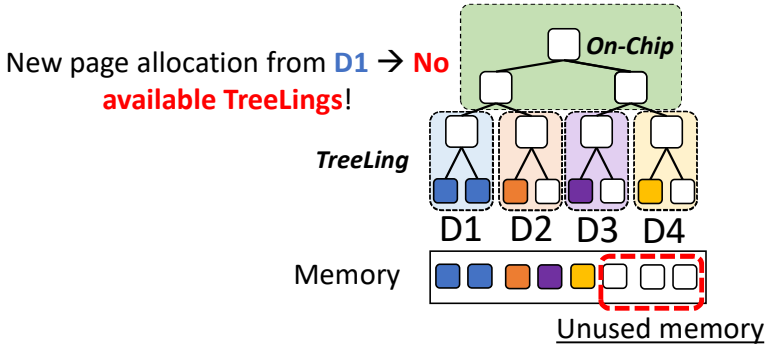


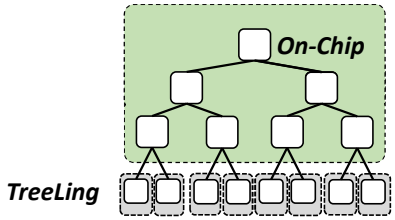
More per-TreeLing coverage.
Higher memory storage overhead.

UCF

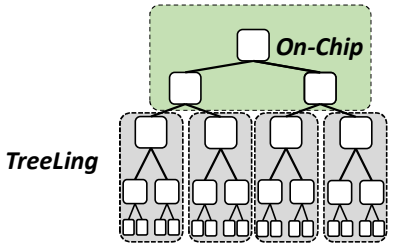
Determining Optimal TreeLing Configuration

- Real-world workloads has skewed memory distributions.
- May suffer ***TreeLing starvation*** if number of TreeLings are low.





More TreeLings available.
Higher on-chip overheads.



More per-TreeLing coverage.
Higher memory storage overhead.

Determining *optimal number of TreeLings* require *multi-dimensional optimization*.
(considering memory size, number of domains and available on-chip storage)

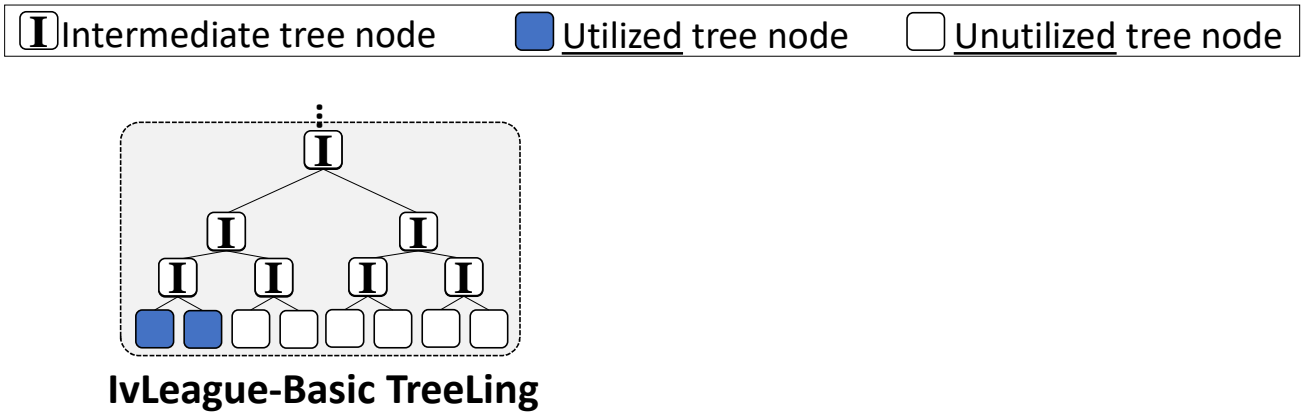
UCF

IvLeague-Invert: Dynamic TreeLing Extension

- Classical integrity tree structure has sub-optimal allocation policy.
  - Larger than required *tree height*.

IvLeague-Invert: Dynamic TreeLing Extension

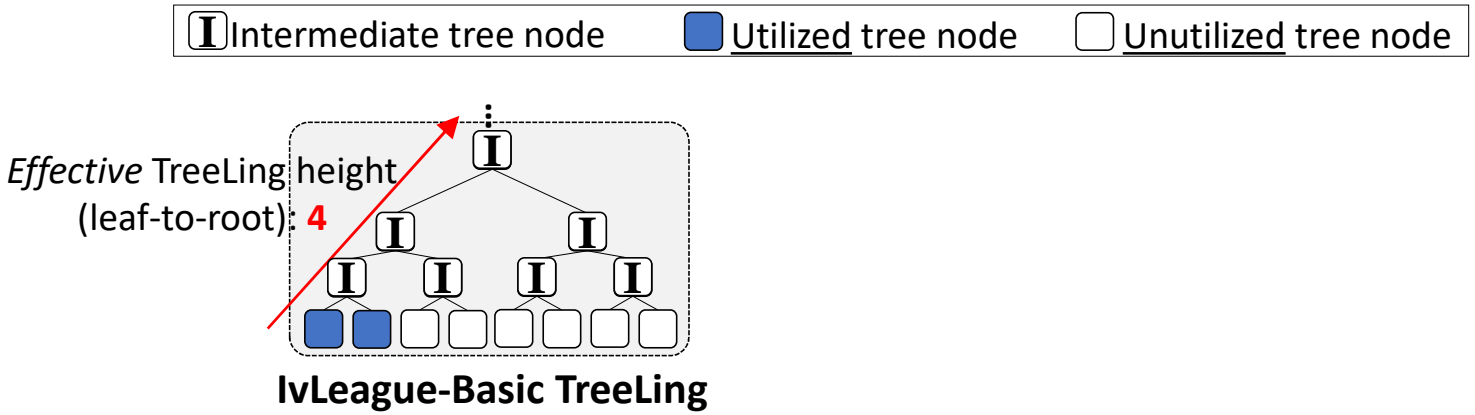
- Classical integrity tree structure has sub-optimal allocation policy.
  - Larger than required *tree height*.



IvLeague-Basic TreeLing

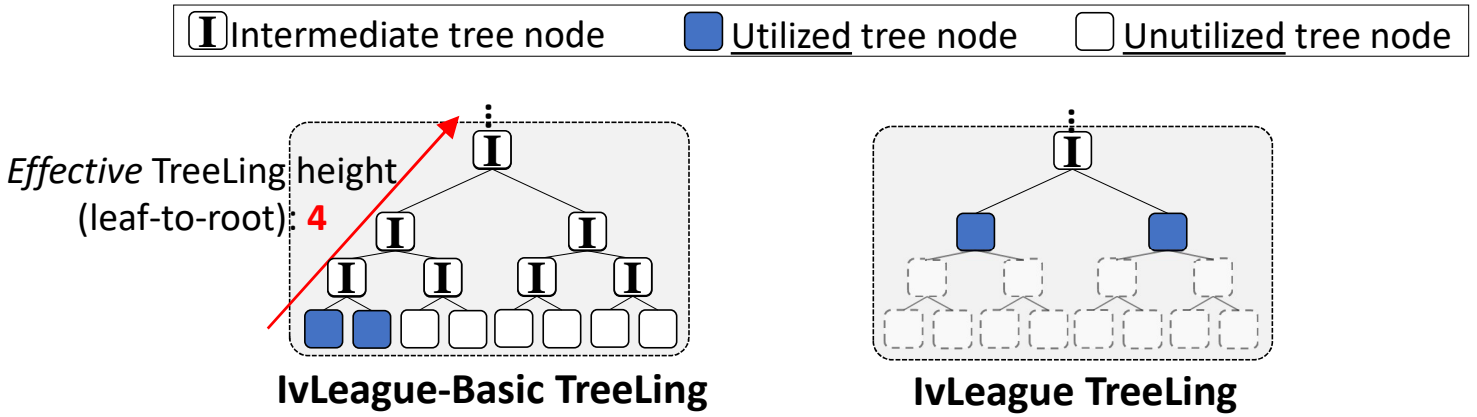
IvLeague-Invert: Dynamic TreeLing Extension

- Classical integrity tree structure has sub-optimal allocation policy.
  - Larger than required *tree height*.



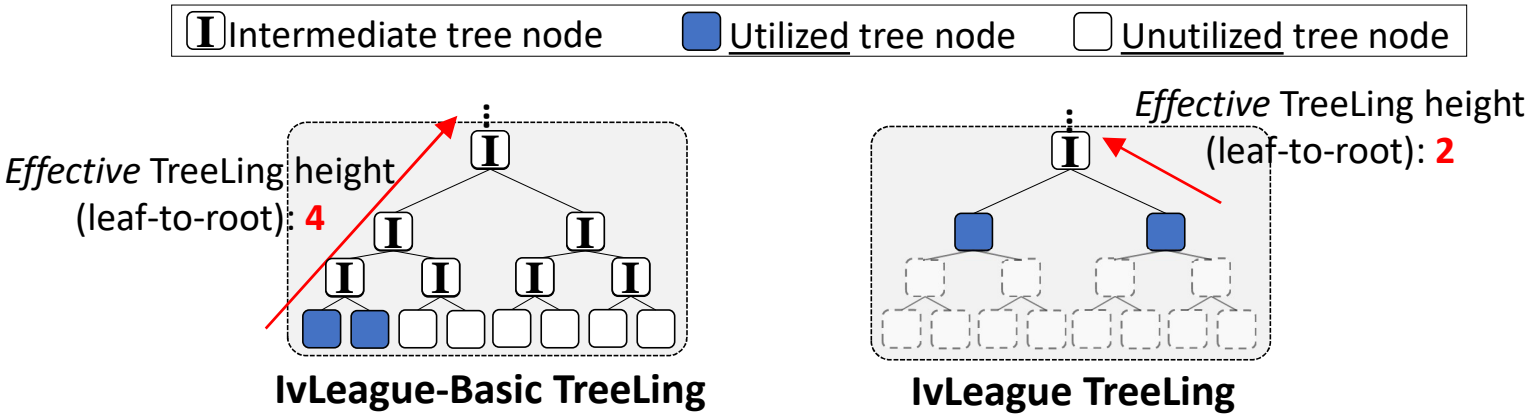
IvLeague-Invert: Dynamic TreeLing Extension

- Classical integrity tree structure has sub-optimal allocation policy.
  - Larger than required *tree height*.
- IvLeague-Invert → **Top-down allocation policy**.
  - Reduced tree height for low utilization TreeLings.



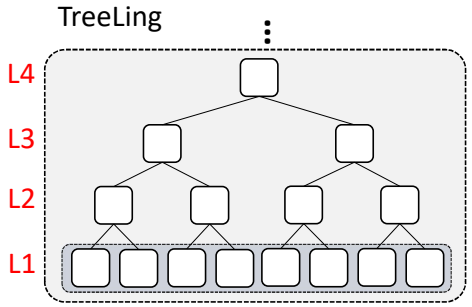
IvLeague-Invert: Dynamic TreeLing Extension

- Classical integrity tree structure has sub-optimal allocation policy.
  - Larger than required *tree height*.
- IvLeague-Invert → **Top-down allocation policy**.
  - Reduced tree height for low utilization TreeLings.



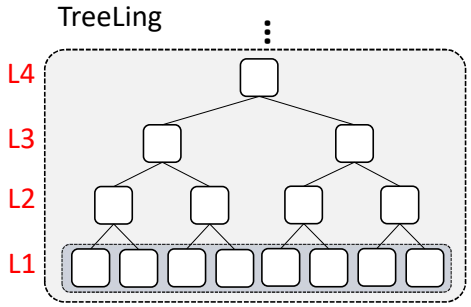
UCF

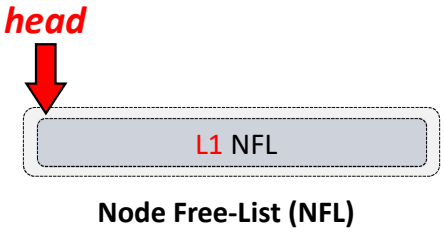
Dynamic TreeLing Extension of IvLeague-Invert



UCF

Dynamic TreeLing Extension of IvLeague-Invert

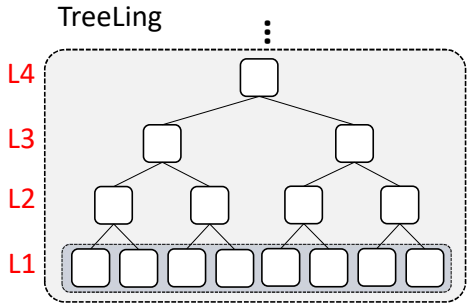


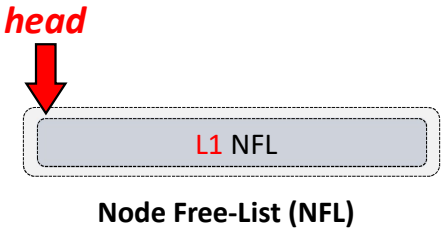


Node Free-List (NFL)

UCF

Dynamic TreeLing Extension of IvLeague-Invert

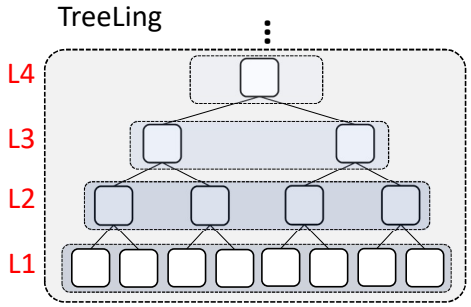


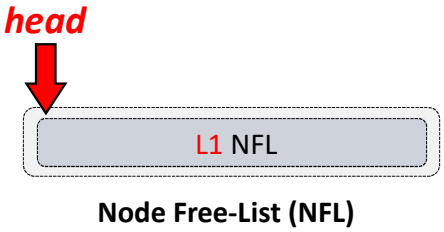


- NFL of IvLeague-Invert contains ***availability vector of all tree nodes***.

UCF

Dynamic TreeLing Extension of IvLeague-Invert

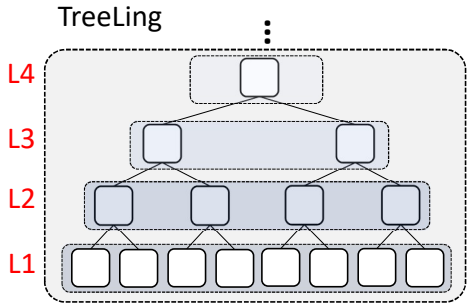




- NFL of IvLeague-Invert contains ***availability vector of all tree nodes***.

UCF

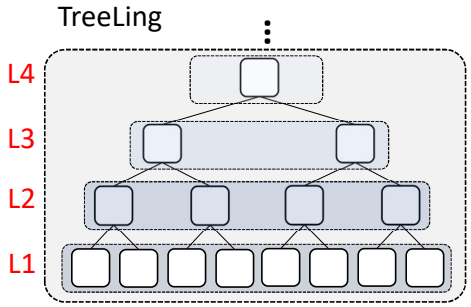
Dynamic TreeLing Extension of IvLeague-Invert



- NFL of IvLeague-Invert contains ***availability vector of all tree nodes***.

UCF

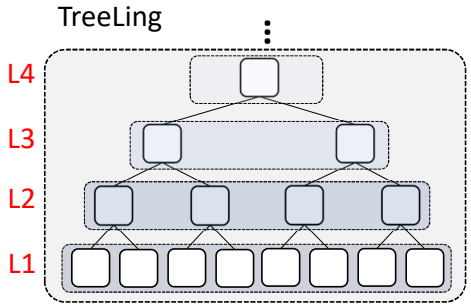
Dynamic TreeLing Extension of IvLeague-Invert



- NFL of IvLeague-Invert contains ***availability vector of all tree nodes***.
- During page allocation, head starts allocating from the top-level.

UCF

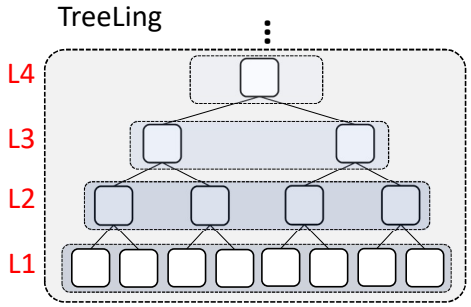
Dynamic TreeLing Extension of IvLeague-Invert



- NFL of IvLeague-Invert contains ***availability vector of all tree nodes***.
- During page allocation, head starts allocating from the top-level.
- TreeLings are statically addressed, ***root traversal for any node in tree is fixed***.

UCF

Dynamic TreeLing Extension of IvLeague-Invert



- NFL of IvLeague-Invert contains ***availability vector of all tree nodes***.
- During page allocation, head starts allocating from the top-level.
- TreeLings are statically addressed, ***root traversal for any node in tree is fixed***.

Can significantly reduce integrity verification overhead by ***decreasing the number of levels to be traversed***.

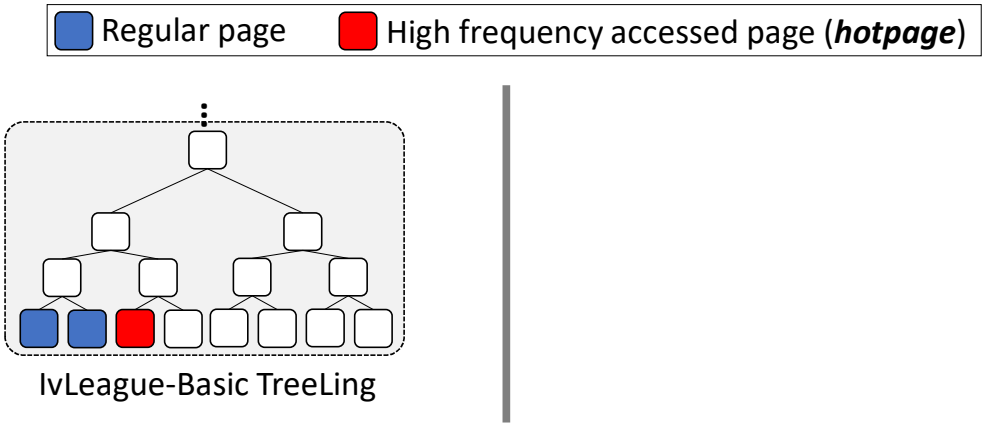
UCF

IvLeague-Pro: Optimize High-Frequency Pages

- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.

IvLeague-Pro: Optimize High-Frequency Pages

- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.

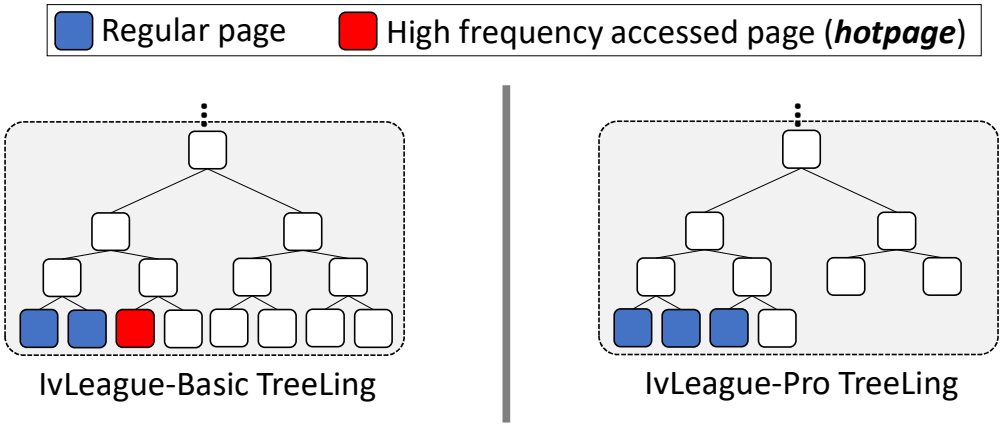


IvLeague-Basic TreeLing

UCF

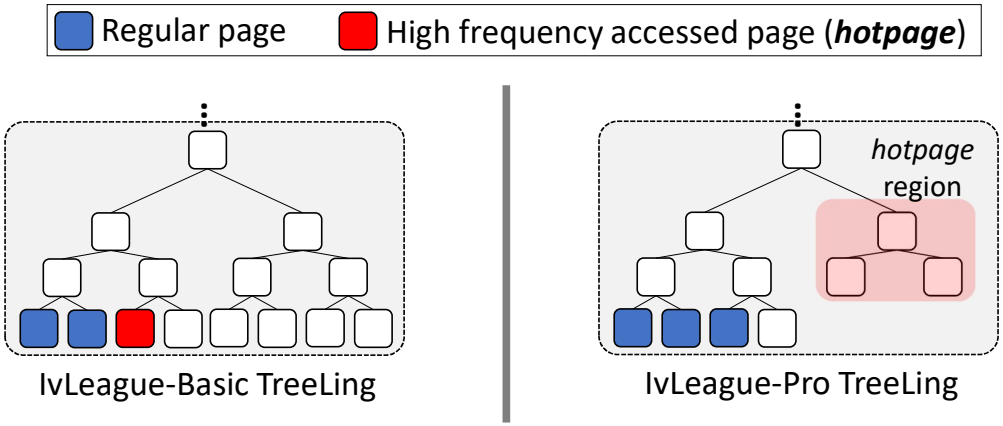
IvLeague-Pro: Optimize High-Frequency Pages

- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.



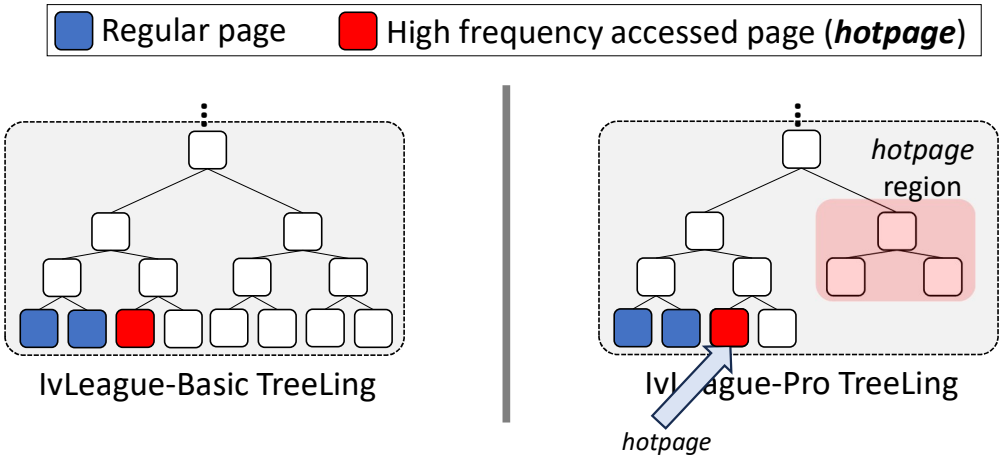
IvLeague-Pro: Optimize High-Frequency Pages

- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.



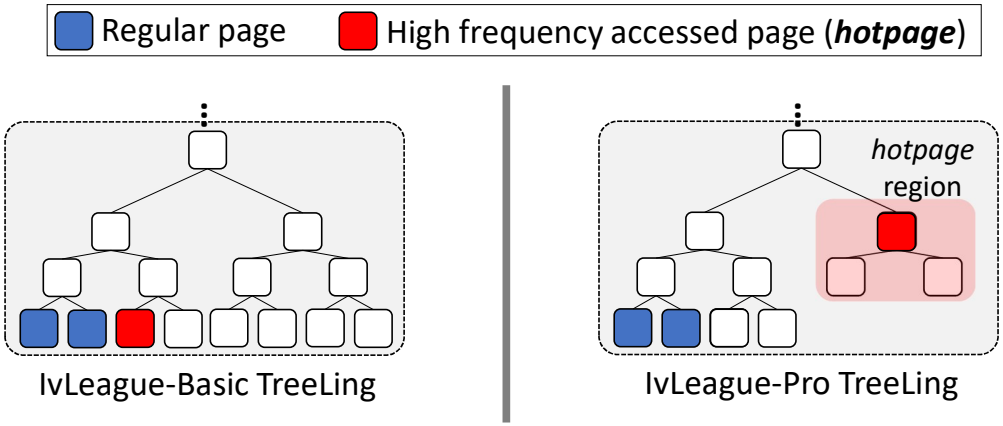
IvLeague-Pro: Optimize High-Frequency Pages

- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.



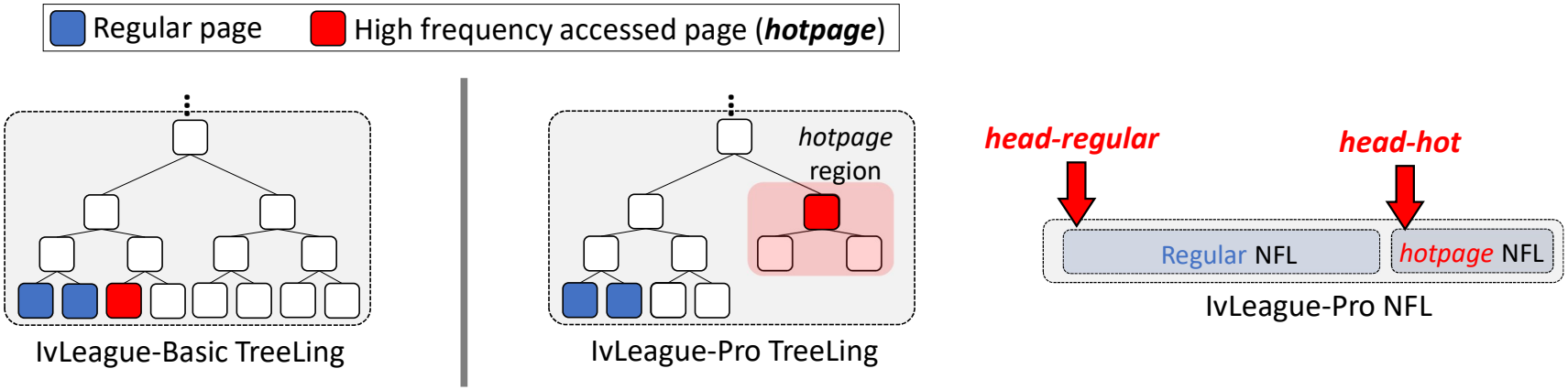
IvLeague-Pro: Optimize High-Frequency Pages

- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.

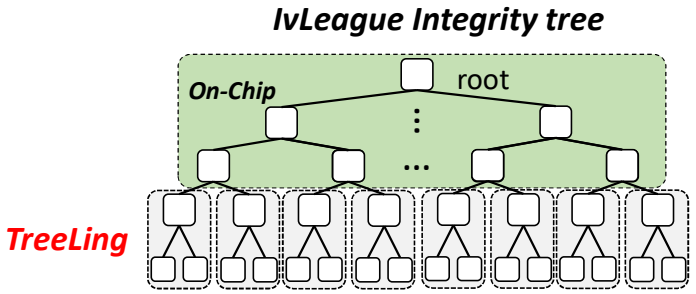


IvLeague-Pro: Optimize High-Frequency Pages

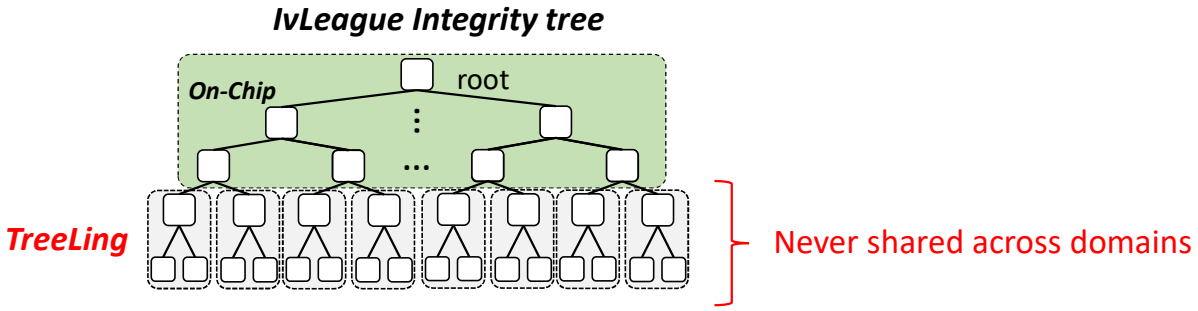
- In typical real-world workloads, a small portion of memory is accessed very frequently (i.e., ***hotpages***).
- Place *hotpages closer to root* → ***reduced integrity verification overhead*** for high frequency pages.



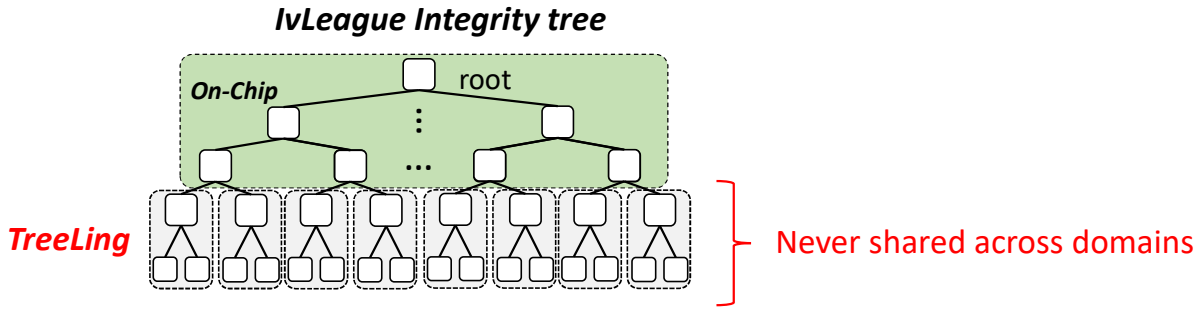
Security Analysis



Security Analysis



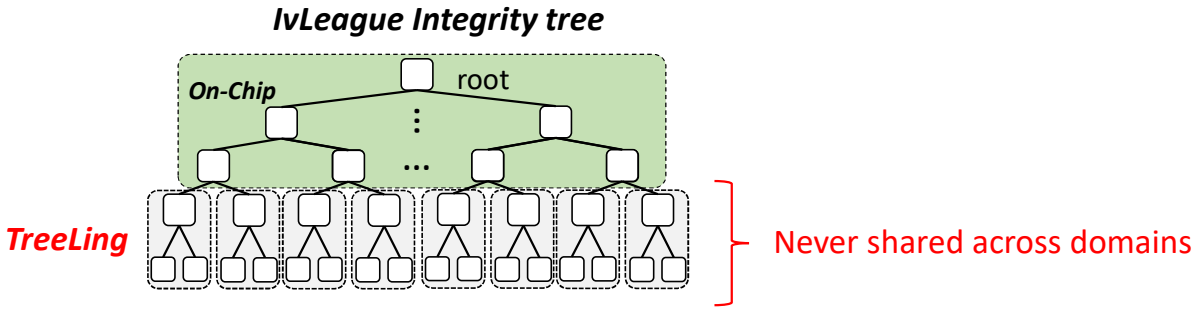
Security Analysis



- Off-chip nodes of IvLeague integrity tree (i.e., TreeLing) are **never shared across domains**.
- TreeLing roots are always kept on-chip → **no timing channel possible** for shared roots.

UCF

Security Analysis

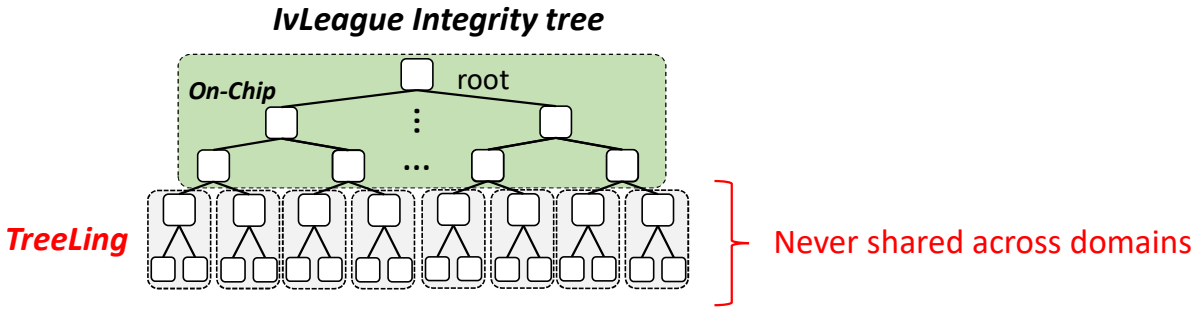


- Off-chip nodes of IvLeague integrity tree (i.e., TreeLing) are **never shared across domains**.
- TreeLing roots are always kept on-chip → **no timing channel possible** for shared roots.

- All TreeLing roots are brought on-chip during system startup → no runtime leakage due to TreeLing allocation.

UCF

Security Analysis



- Off-chip nodes of IvLeague integrity tree (i.e., TreeLing) are **never shared across domains**.
- TreeLing roots are always kept on-chip → **no timing channel possible** for shared roots.

- All TreeLing roots are brought on-chip during system startup → no runtime leakage due to TreeLing allocation.

- IvLeague is designed specifically to thwart shared metadata across security domains.
  - Contention-based attacks on non-shared part of integrity tree is prevented using classical side channel protection.

UCF

Experimental Setup

- **Simulator:** Gem5 full system simulation.
- **Architecture configuration:** 8-core, Out-of-order, x86.
- **Memory:** 32GB, 2-rank.
- **Secure processor configuration:**
  - *Integrity tree:* 8-ary Bonsai Merkle Tree.
  - *Tree cache:* 8-way 256KB.
- **IvLeague configuration:**
  - 16-way 204KB *leaf-mapping metadata* cache.
  - 2-entry (per-domain) *NFL buffer*.
  - Size of TreeLing: 64MB; # of TreeLings: 4K.

- **Schemes:**
  - ***Baseline***: State-of-the-art secure processor design following Bonsai Merkle Tree.
  - ***IvLeague-Basic***.
  - ***IvLeague-Invert***.
  - ***IvLeague-Pro***.

Performance of IvLeague Schemes

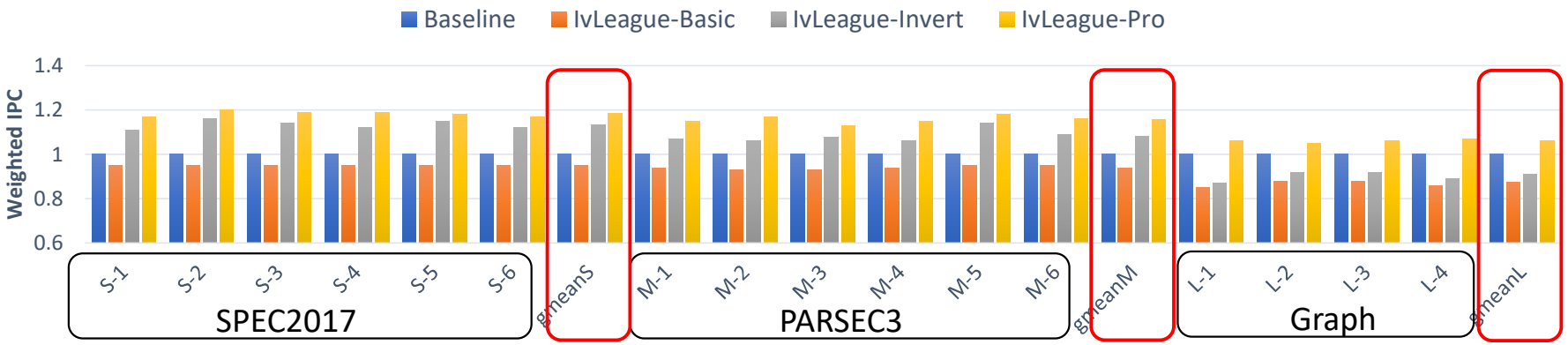


Figure: Comparison of performance (i.e., Weighted IPC normalized to Baseline) under different schemes.

Performance of IvLeague-Basic: Compared to *baseline*	↓2.7% Small	↓5.5% Medium	↓17.4% Large

UCF

Performance of IvLeague Schemes

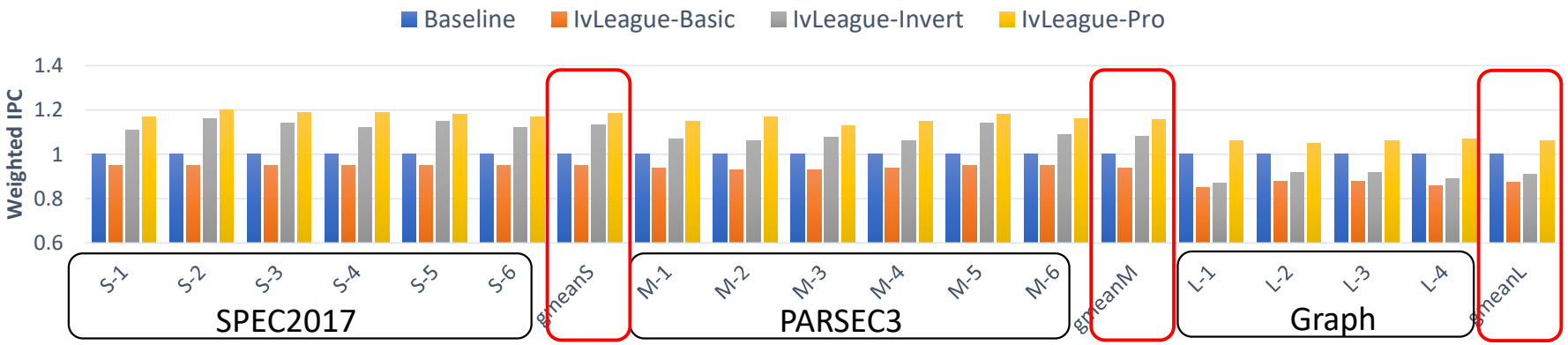


Figure: Comparison of performance (i.e., Weighted IPC normalized to Baseline) under different schemes.

Performance of IvLeague-Basic: Compared to *baseline*	↓2.7% Small	↓5.5% Medium	↓17.4% Large

Performance of IvLeague-Invert/IvLeague-Pro: Compared to *baseline*	↑8.2%/↑13.5% Small	↑3.4%/↑9.3% Medium	↓13.2%/↑3.4% Large

UCF

Effectiveness of NFL Design

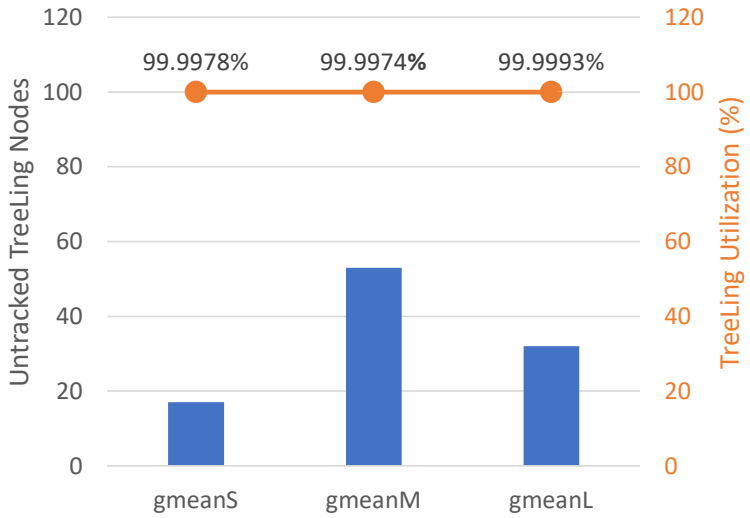


Figure: Utilization of TreeLing nodes using NFL.

UCF

Effectiveness of NFL Design

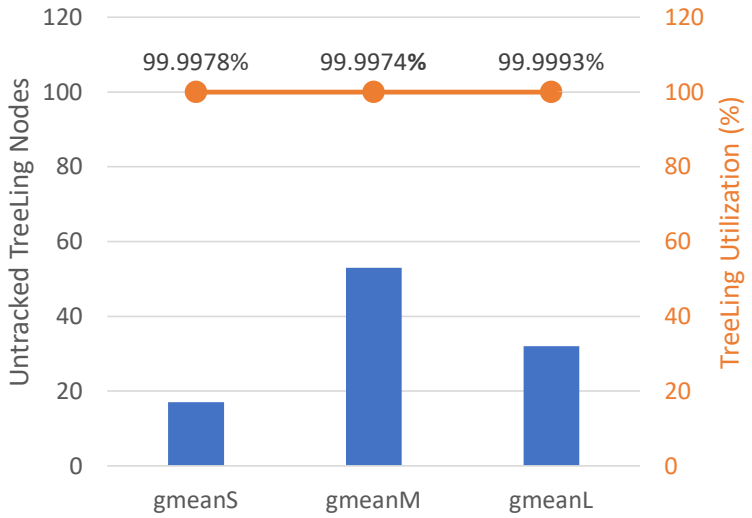
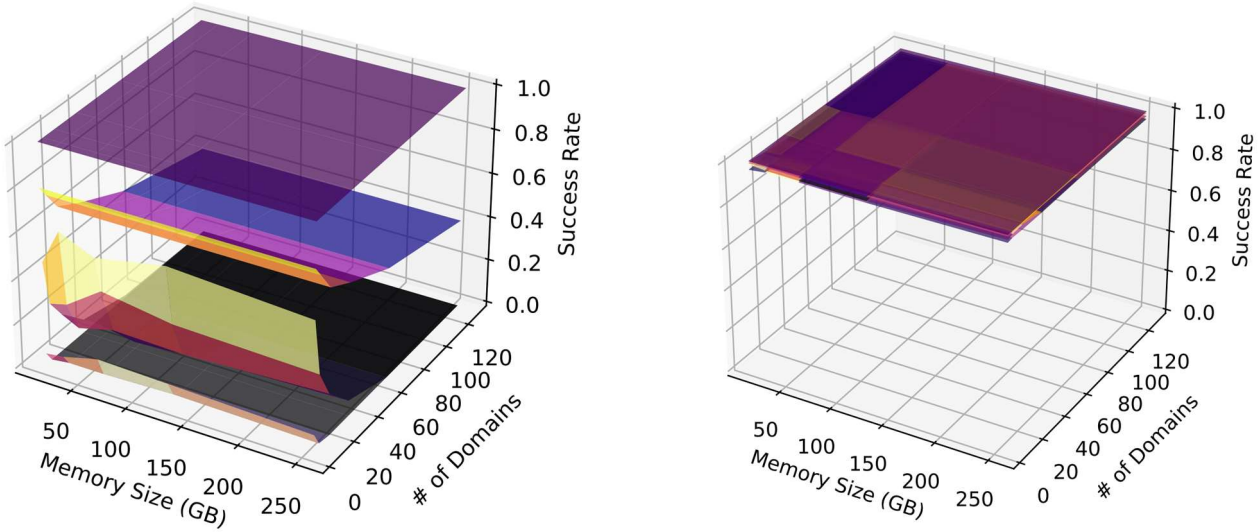


Figure: Utilization of TreeLing nodes using NFL.

IvLeague can achieve near-optimal tree node utilization (> 99.99%) with NFL.

UCF

Scalability of IvLeague Scheme



Static partitioning scheme ***IvLeague scheme***

Figure: Comparison of different number of domain support.

- Static partitioning has significantly low success rate when system memory utilization is high.
- IvLeague shows consistently high success rate under any give system memory utilization.

UCF

Sensitivity Analysis of IvLeauge

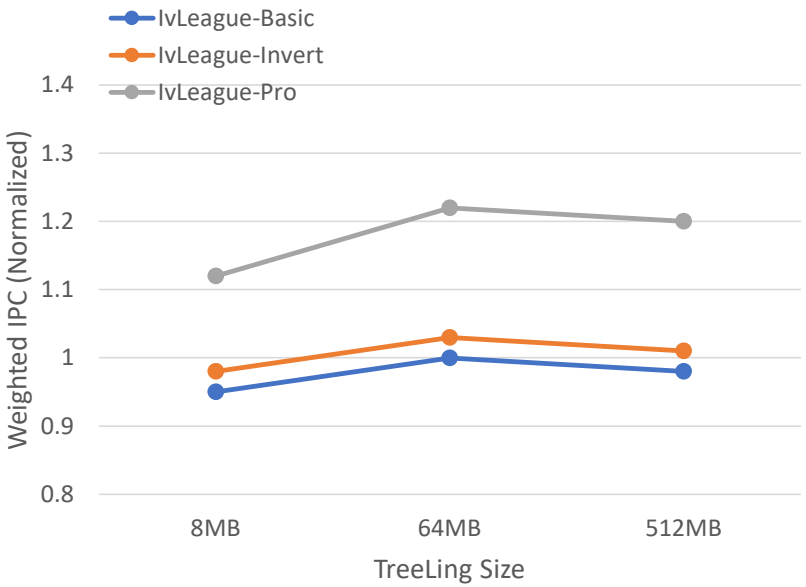


Figure: Sensitivity to TreeLing size (normalized to 64MB TreeLing size).

UCF

Sensitivity Analysis of IvLeauge

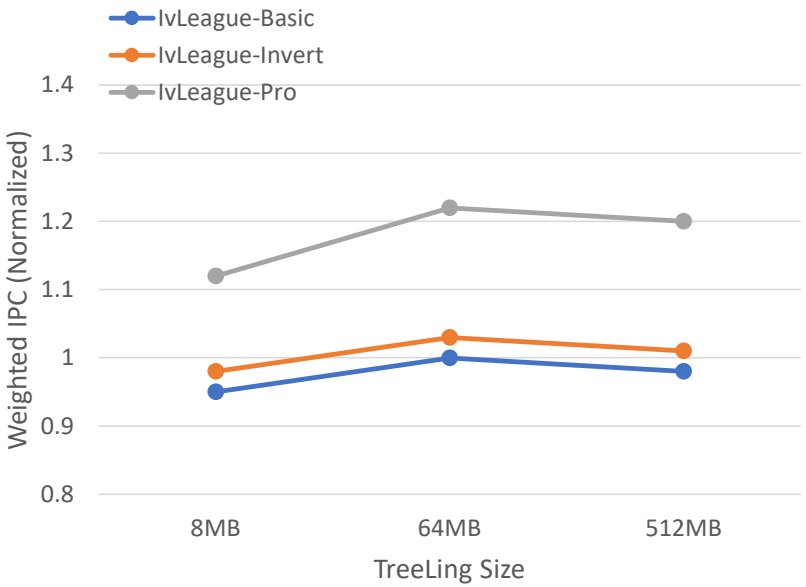


Figure: Sensitivity to TreeLing size (normalized to 64MB TreeLing size).

64MB TreeLing configuration offers the best balance between levels inside TreeLing and on-chip locked blocks.

UCF

More on Paper…

- Additional discussion on IvLeauge design.
- NFL buffer performance analysis.
- Analysis of additional memory accesses.
- Analytical investigation of TreeLing configurations.
- Hardware overhead analysis.
- And more…

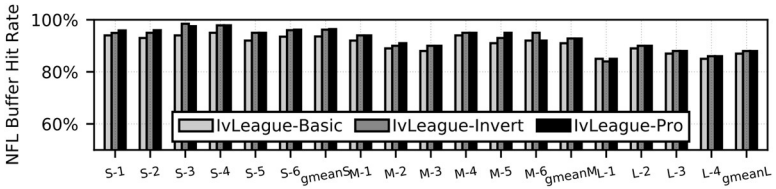
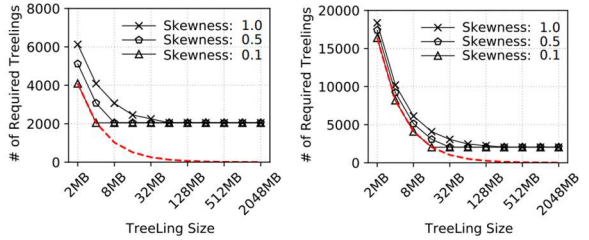


Figure: NFLB hit rate for all workloads.



(a) System memory: 8GB (b) System memory: 32GB

Figure: TreeLings required under different memory allocation distributions across programs (number of IV Domains: 2^{12}). Red dashed line represents *minimum* number of TreeLing that covers the entire available memory (i.e., assuming all TreeLings are fully utilized).

Component	Storage	Area
NFL Logic and Buffer	528-byte	$0.0071mm^2$
LMM Cache	204KB	$0.33mm^2$
Hotpage Predictor (IvLeague-Pro)	848-byte	$0.018mm^2$

TABLE III: On-chip hardware cost for IvLeague components.

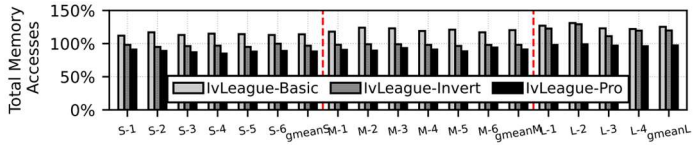


Figure: Additional memory accesses due to IvLeague operations (normalized to *baseline* scheme).

UCF

Key Takeaways

- Existing integrity verification mechanisms use a global integrity tree shared across security domains.
  - Introduce side channel leakage through integrity tree metadata sharing.
- IvLeague provides side channel-resistant isolated integrity trees among dynamic domains.
  - Splits the global tree into multiple fixed-size subtrees.
  - Dynamically allocating these subtrees to domains during runtime.
- IvLeague optimizations shortens the integrity verification path significantly.
- IvLeague provides 3%-13% speedup over the insecure baseline.
  - With minimal hardware and logic overheads.

Thanks! Questions?

Md Hafizul Islam Chowdhuryy
CASR Lab (https://casr.ece.ucf.edu)
Email: hafizul.islam@ucf.edu