

Revelator

Rapid Data Fetching
via System-Software-Guided Hash-based
Speculative Address Translation

Konstantinos Kanellopoulos Konstantinos Sgouras
Harsh Songara Andreas Kosmas Kakolyris Vlad-Petru Nitu
Spiros Galanopoulos Rahul Bera Konstantina Koliogeorgi
Rakesh Kumar Onur Mutlu

SAFARI **ETH** zürich  NTNU

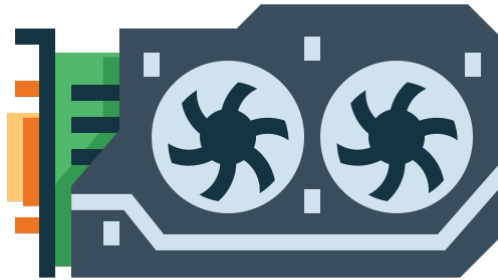
Background

Virtual Memory is a Cornerstone of Modern Computing Systems

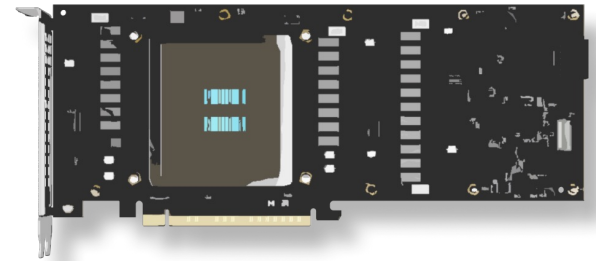
CPUs



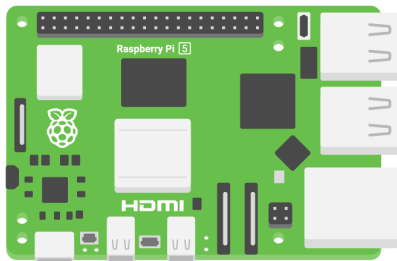
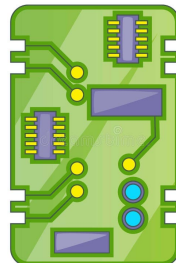
GPUs



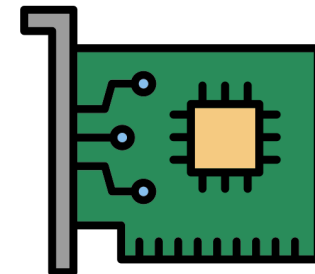
AI Accelerators



Mobile/ Embedded SoCs



NICs



Emerging Workloads

Genomics



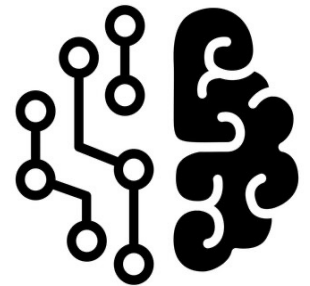
Graph Analytics



Databases



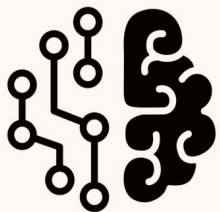
Sparse Machine Learning



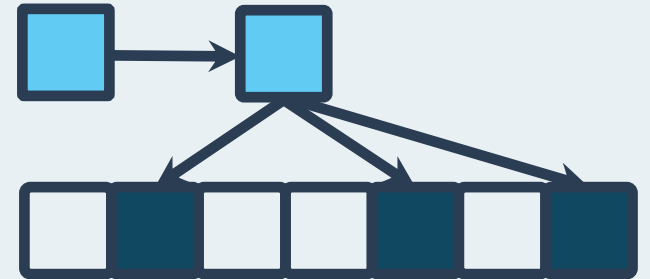
**Performance bottleneck: 10-30%
of cycles spent on address translation**

Zao et al. "[Contiguitas: The Pursuit of Physical Memory Contiguity in Datacenters](#)" ISCA 2023

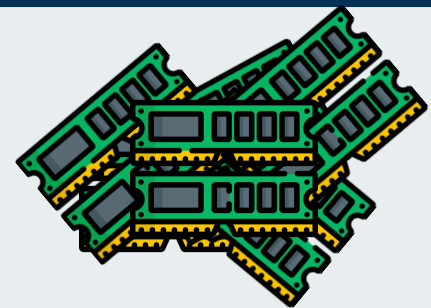
Address Translation Overheads



**Irregular Memory
Accesses**



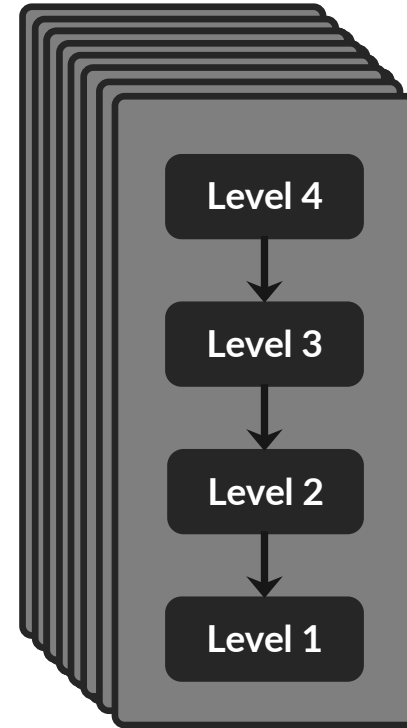
**Large Data
Footprints**



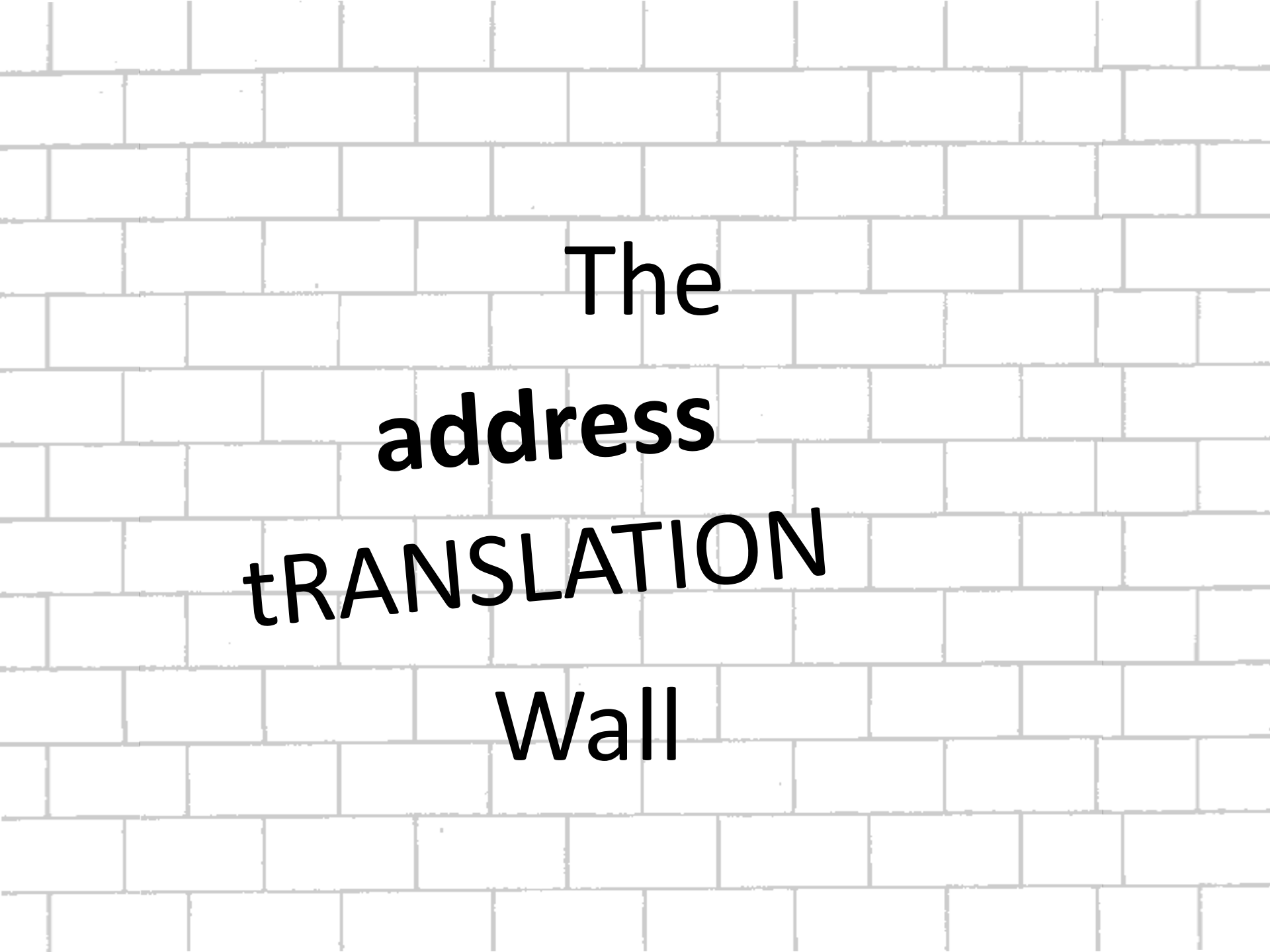
Low Spatial and Temporal Locality



*Frequent
TLB Misses*

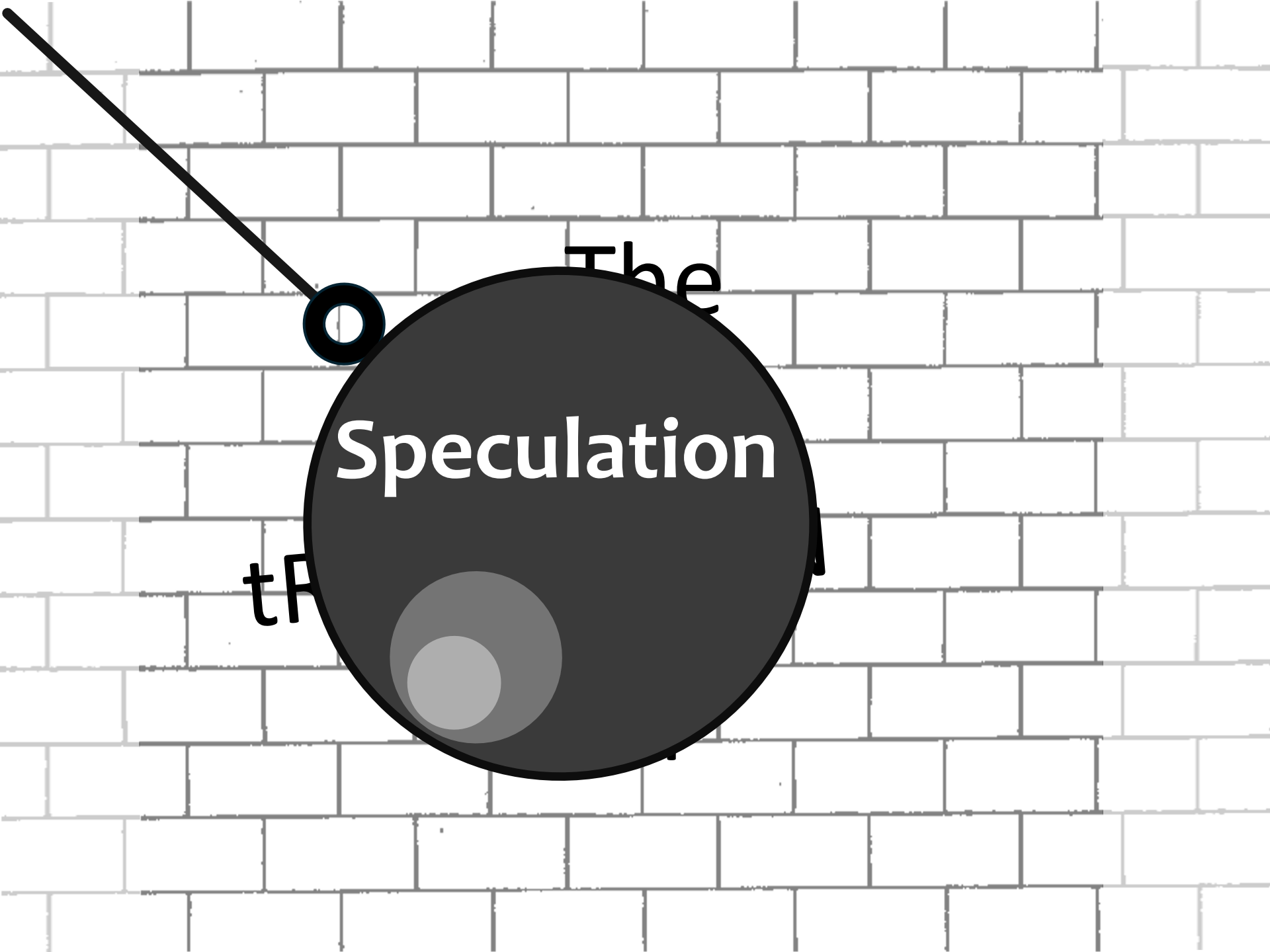


*Frequent High-Latency
Page Table Walks*

The background of the image is a light gray brick wall with dark gray mortar lines. The bricks are arranged in a standard running bond pattern.

The address tRANSLATION Wall

The
address
t**RANS**LATION
Wall



The
Speculation

tA

Speculative Address Translation

Conventional Address Translation

TLB Miss

Page Table Walk

Data Fetch

Physical
address
is known



Likely DRAM access

Speculative Address Translation

Conventional Address Translation

TLB Miss

Page Table Walk

Data Fetch

Speculative Address Translation

TLB Miss

Page Table Walk

Data Fetch

Likely
cache hit

**Predict
Physical
Address**

Speculative Data Fetch

Speculative Address Translation

Conventional Address Translation

TLB Miss

Page Table Walk

Data Fetch

Speculative Address Translation

TLB Miss

Page Table Walk

Data Fetch

Saved
Cycles

**Predict
Physical
Address**

Speculative Data Fetch

Likely
cache hit

Predicting the PA is **Challenging**

Virtual-to-physical mapping depends on **unpredictable** runtime conditions and system-level factors

- 1 Scheduling Decisions
- 2 Co-running Applications
- 3 Memory Utilization
- 4 Memory Fragmentation

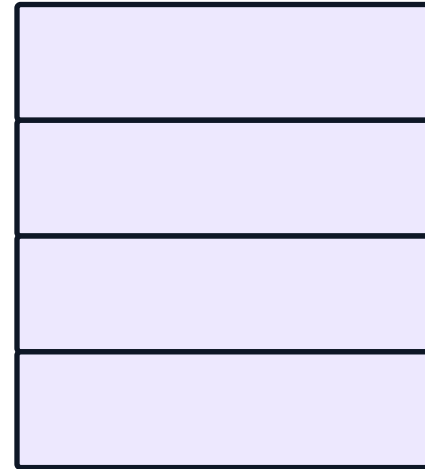
Example:

Same Process, Different Instances

Virtual Address Space



Physical Address Space

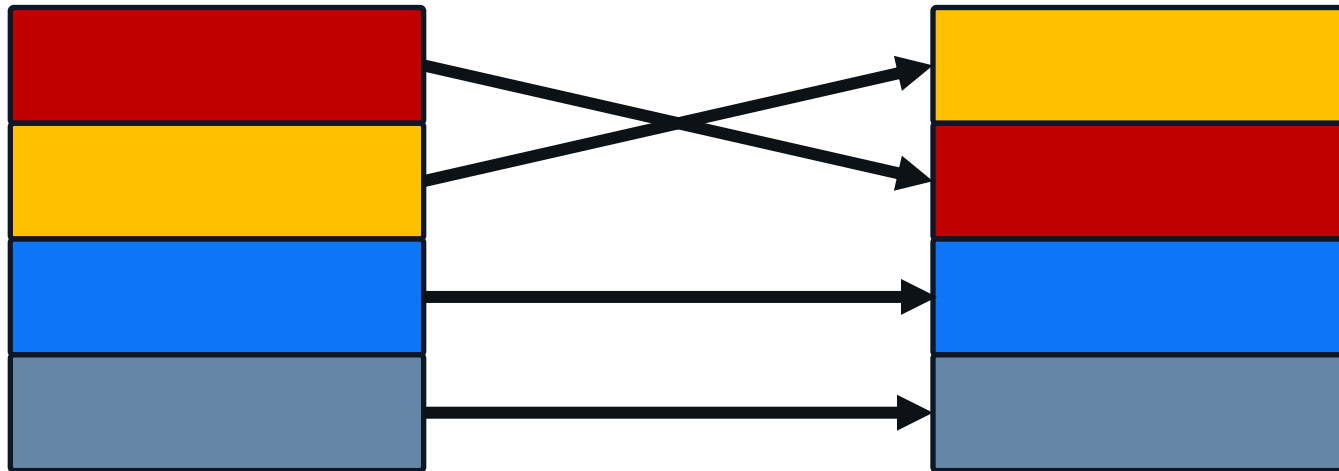


Example:

Same Process, Different Instances

Virtual Address Space

Physical Address Space



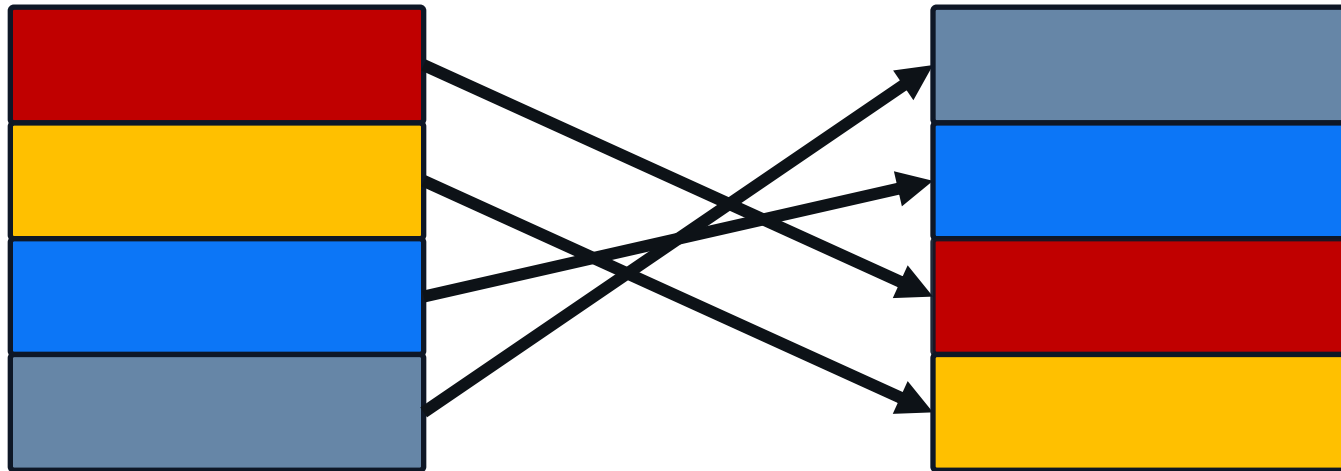
Process A – Instance 1

Example:

Same Process, Different Instances

Virtual Address Space

Physical Address Space



Process A – Instance 2

Example:

Same Process, Different Instances

Virtual Address Space

Physical Address Space

Takeaway: VA-to-PA mapping is established by modern OSes in an unpredictable manner

Process A – Instance 3

Motivation: Shortcomings of Prior Works

To improve VA-to-PA predictability,
prior speculative translation schemes *require*:

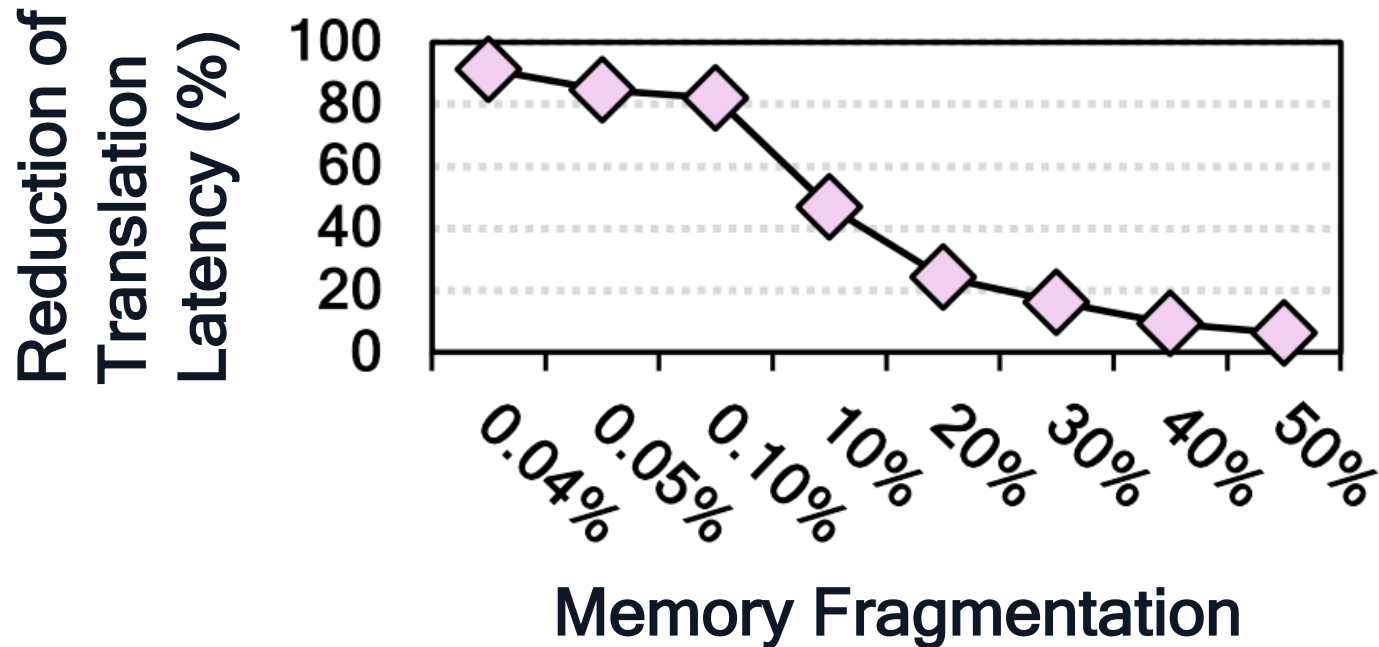
1

Free large pages and/or VA-to-PA contiguity

2

Costly hardware changes for speculation metadata

Prior Work: Contiguity-based Speculation¹



As memory fragmentation increases, reduction of translation latency drops significantly

¹Alverti et al. "Enhancing and Exploiting Contiguity for Fast Memory Virtualization", ISCA 2020

Our Goal

Enable *highly-accurate* speculative address translation without requiring:

Large page availability 

VA-to-PA contiguity 

Costly hardware for metadata 

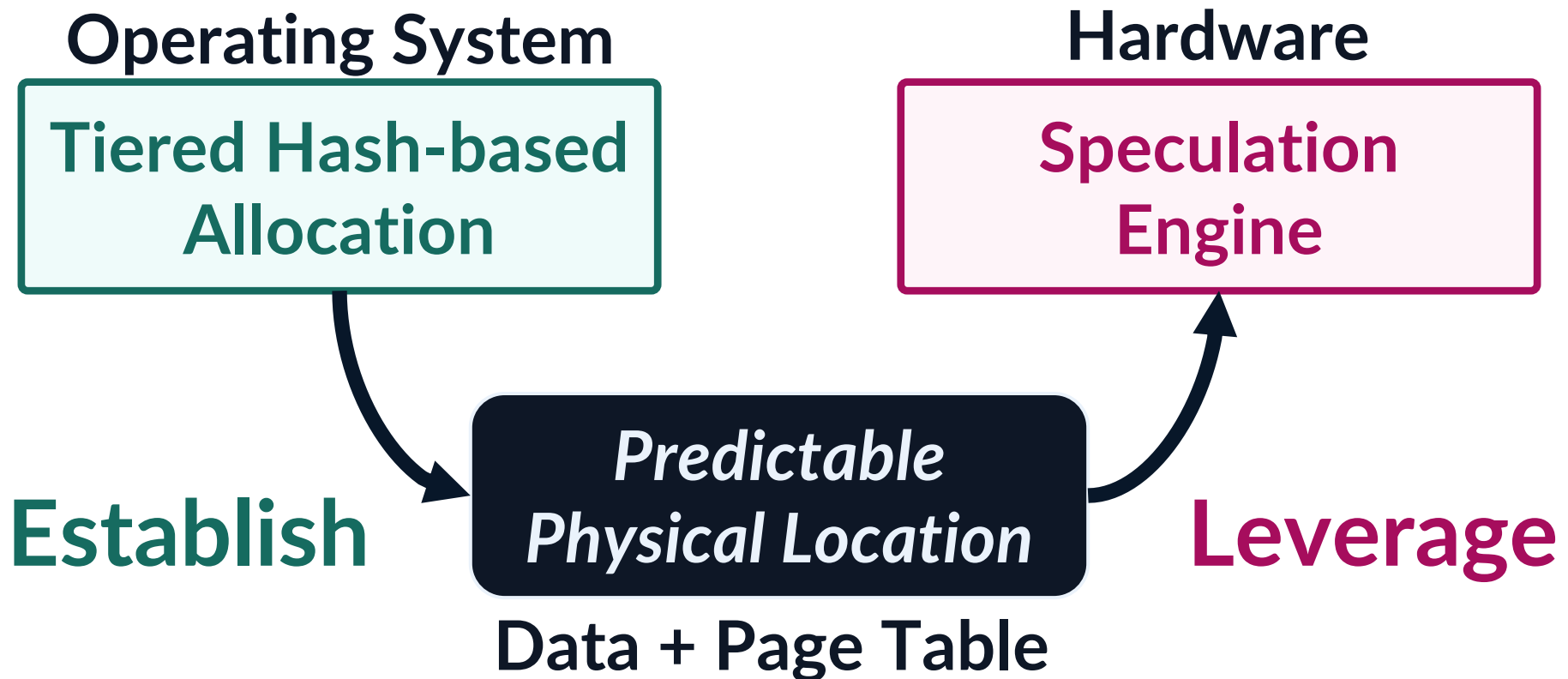
Revelator

Revelator: Key Approach

Co-design hardware and OS to perform
highly accurate speculative translation

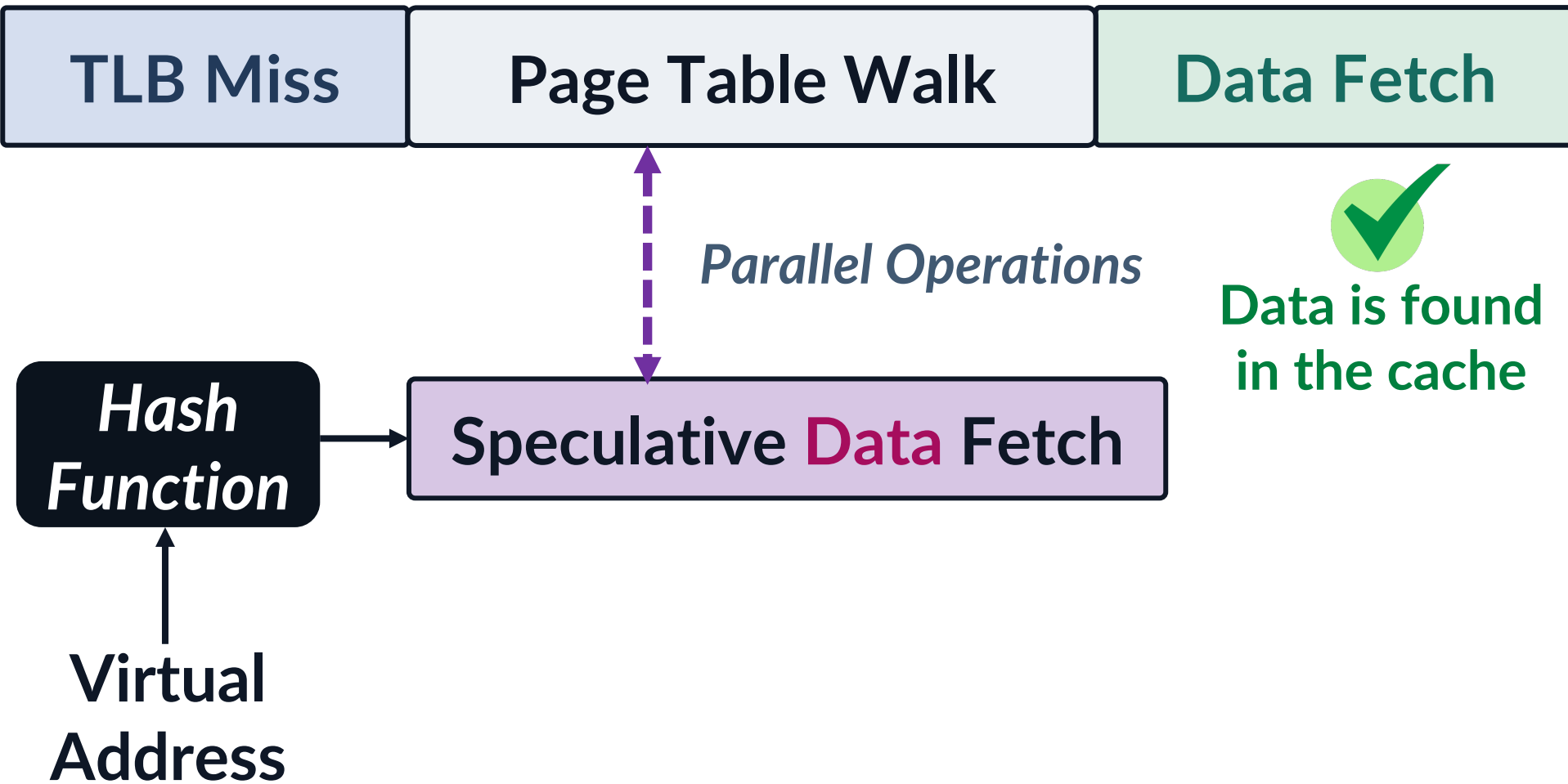
Revelator: Key Idea

Employ *hashing* to enable **highly accurate** speculative address translation with **small system modifications**

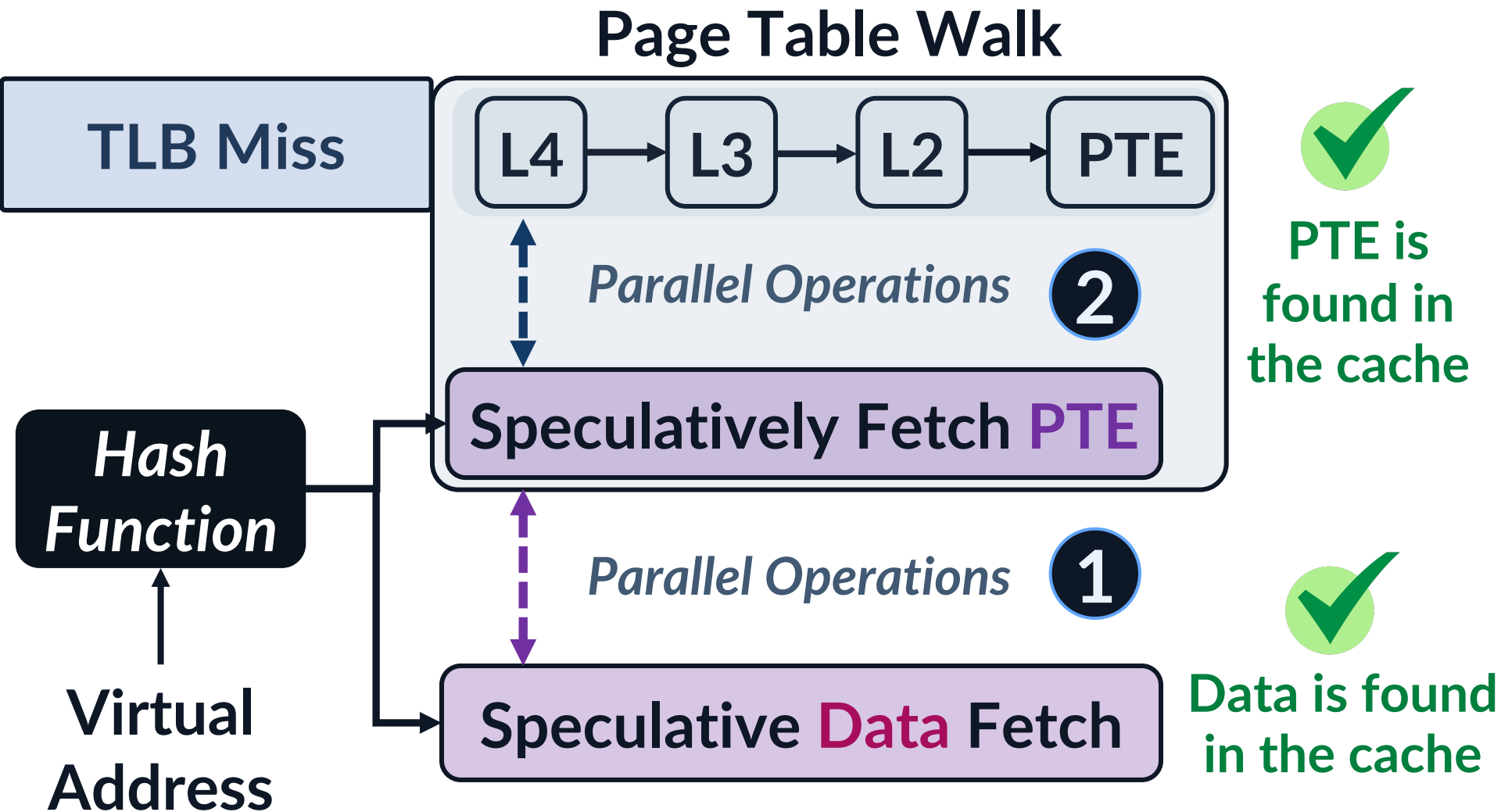


Revelator: Overview

Revelator: Overview



Revelator: Overview



Revelator: Key Benefit

Co-design hardware and OS to perform highly accurate speculative translation

Operating System

Hardware

Hides & reduces address translation latency **without** depending on memory fragmentation (i.e., large pages, contiguity)

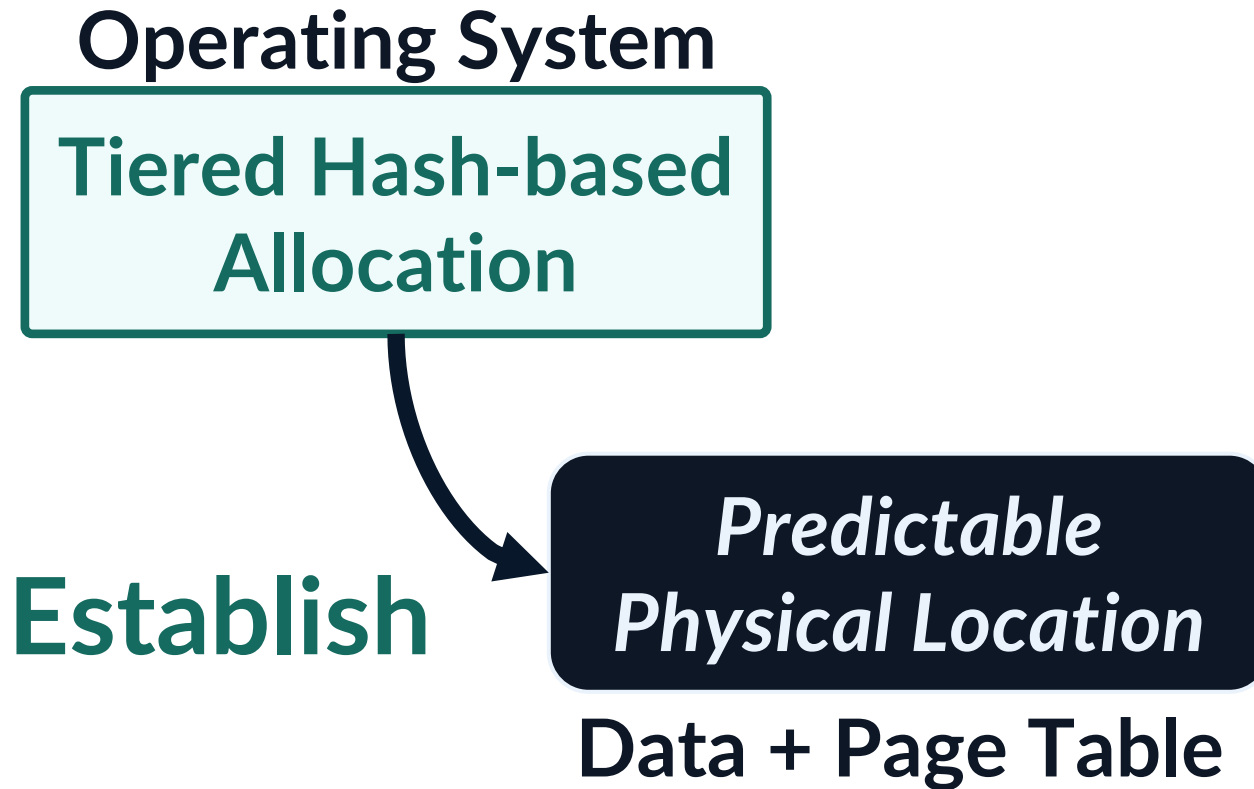
Establish

*Predictable
Physical Location*

Leverage

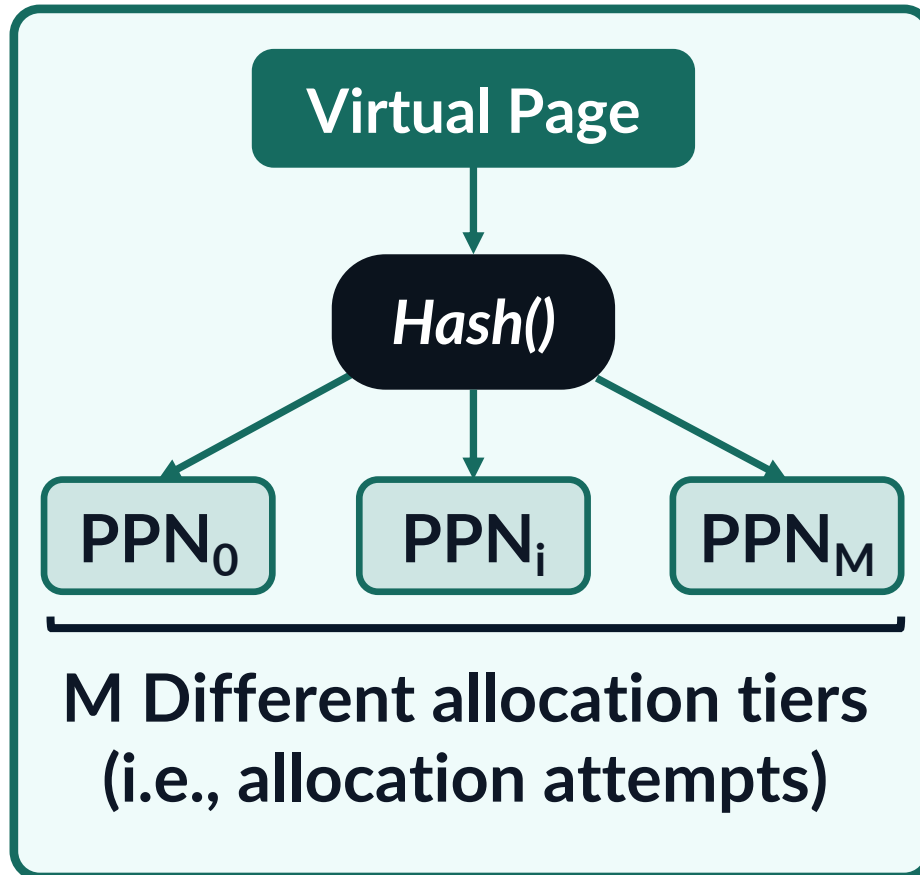
Data + Page Table

Revelator: Key Components



Revelator: OS Side

Operating System



Establish

*Predictable
Physical Location*
Data + Page Table

Revelator: OS Side

Small OS modifications with
low impact on allocation latency

Hardware

Virtual Page

PPN_0

..

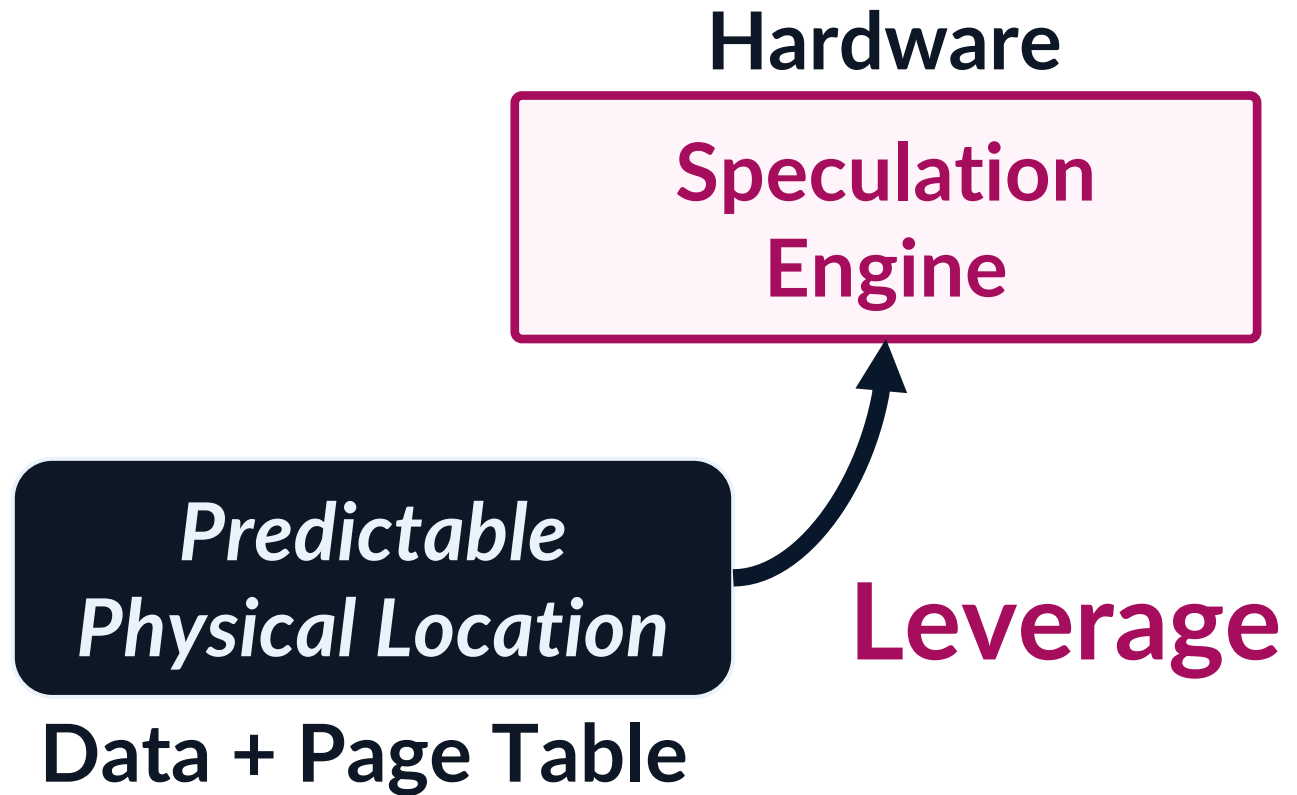
PPN_i

..

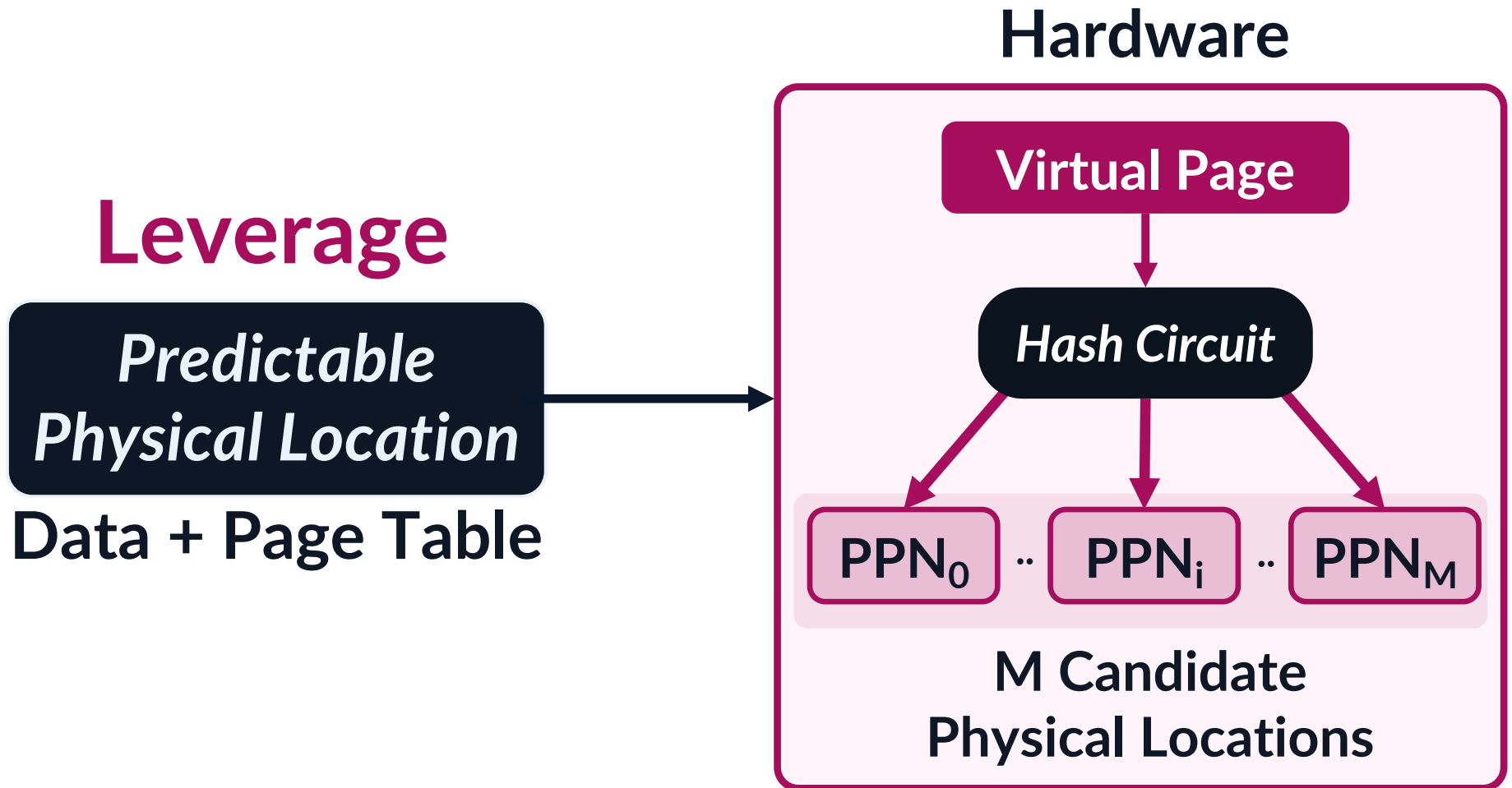
PPN_M

M Candidate
Physical Locations

Revelator: Key Components



Revelator: Hardware Side



Revelator: Hardware Side

Operating System

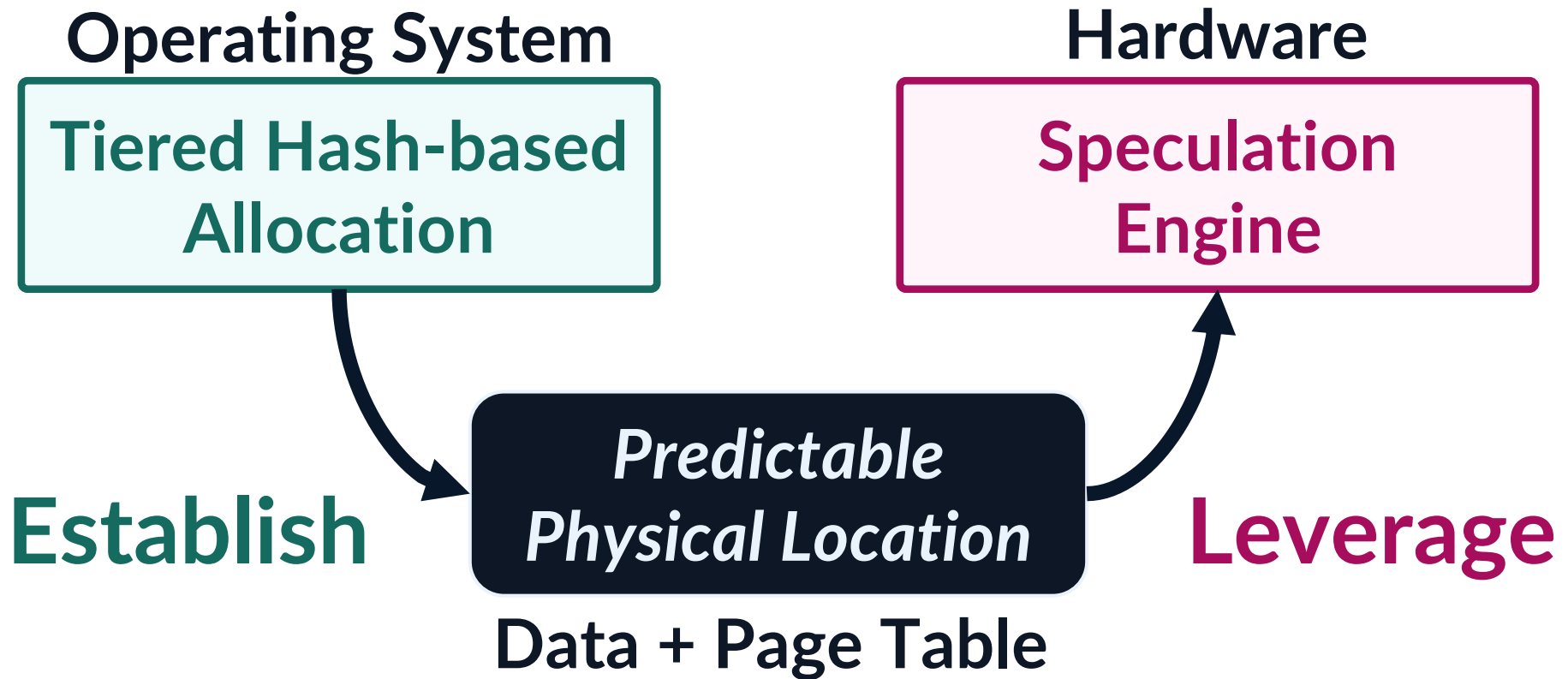
Virtual Page

Low-cost hardware
to generate candidate PAs
without the need for **extra** metadata

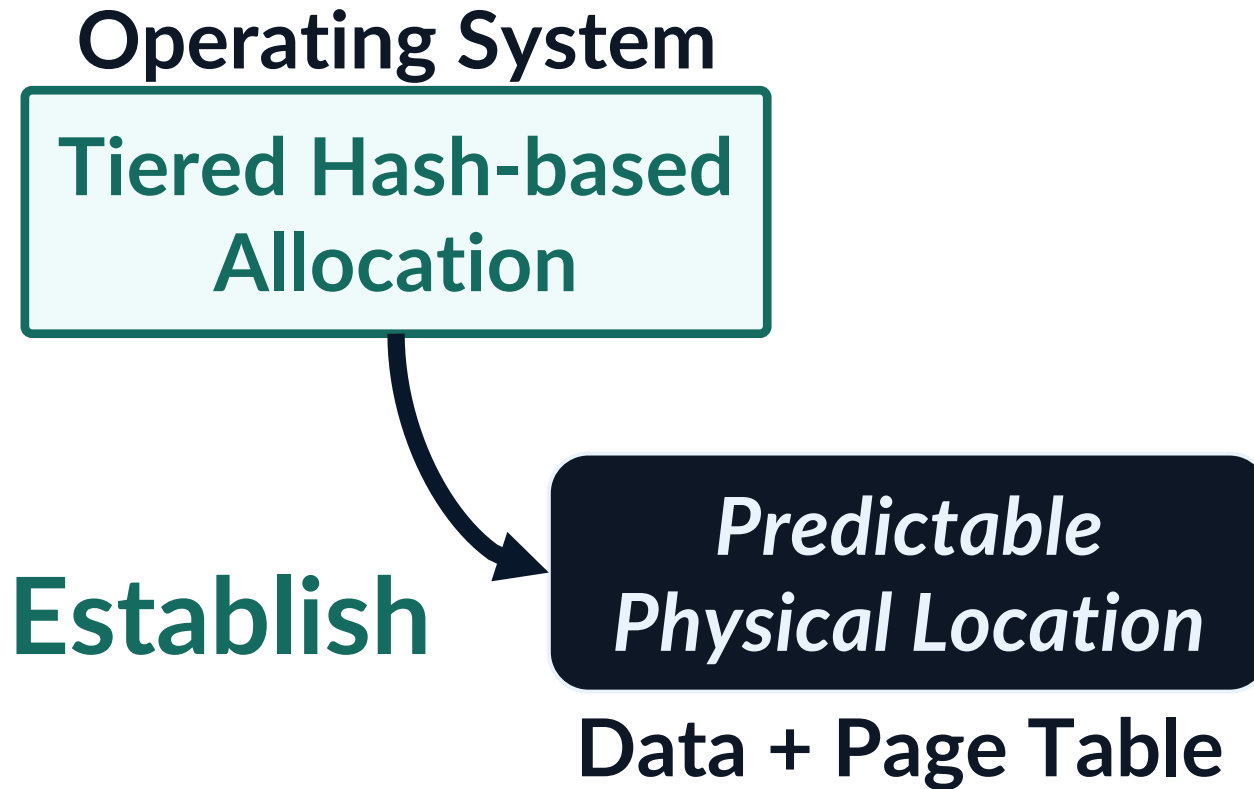
M Different Tiers
(i.e., Hash Attempts)

Revelator: Detailed Design

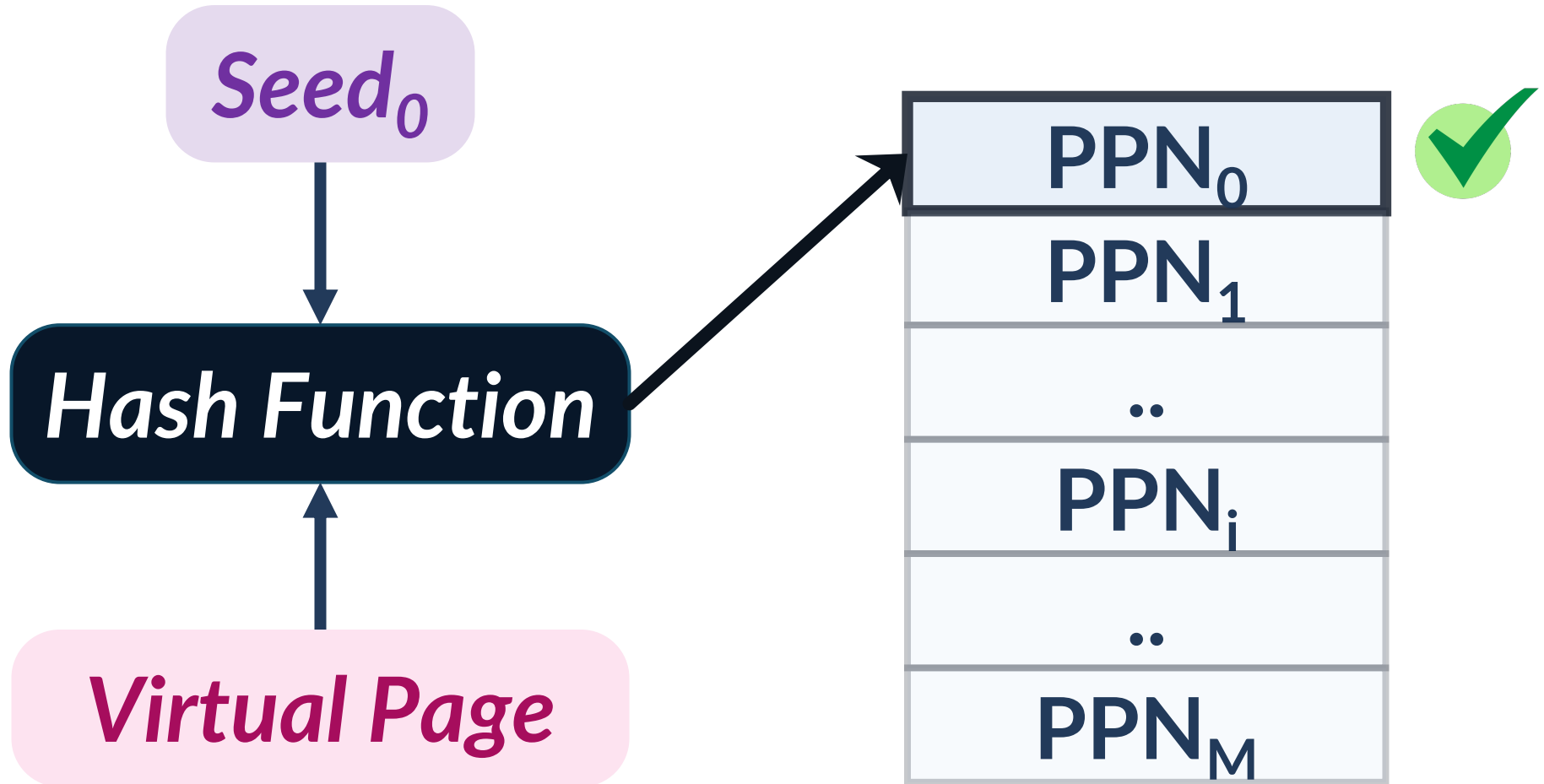
Revelator: Key Components



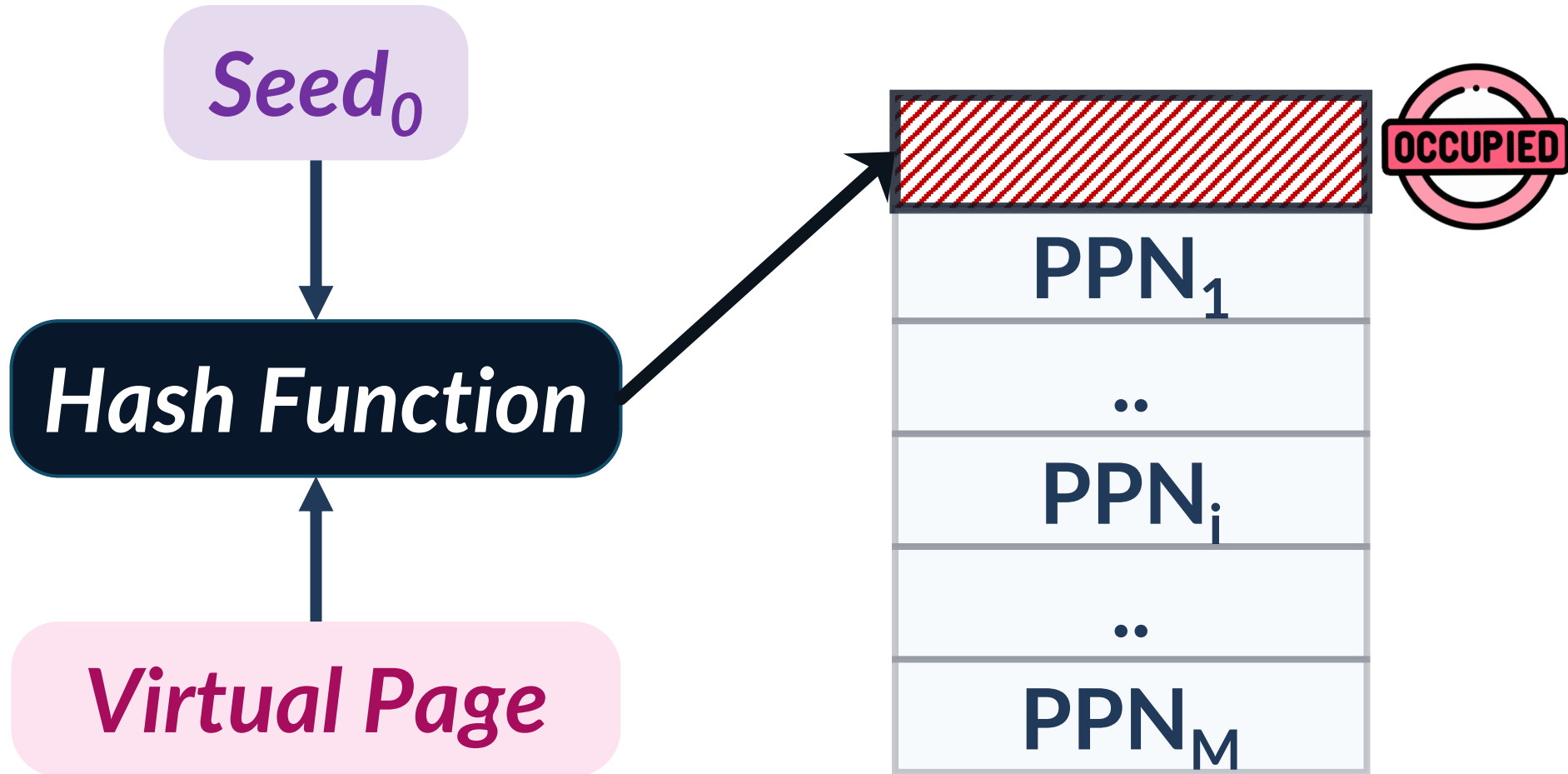
Revelator: Key Components



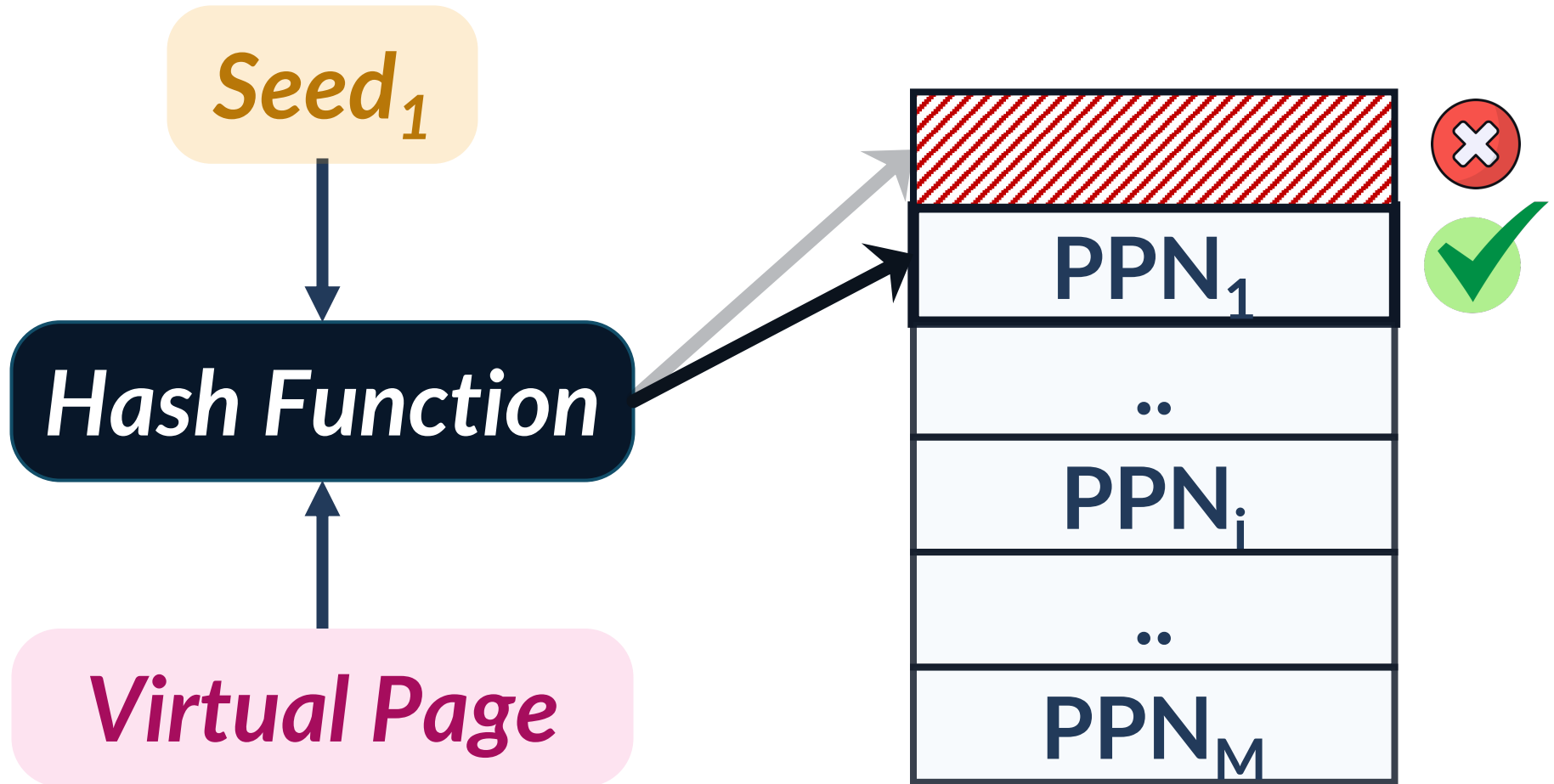
OS: Tiered Hash-based Allocation



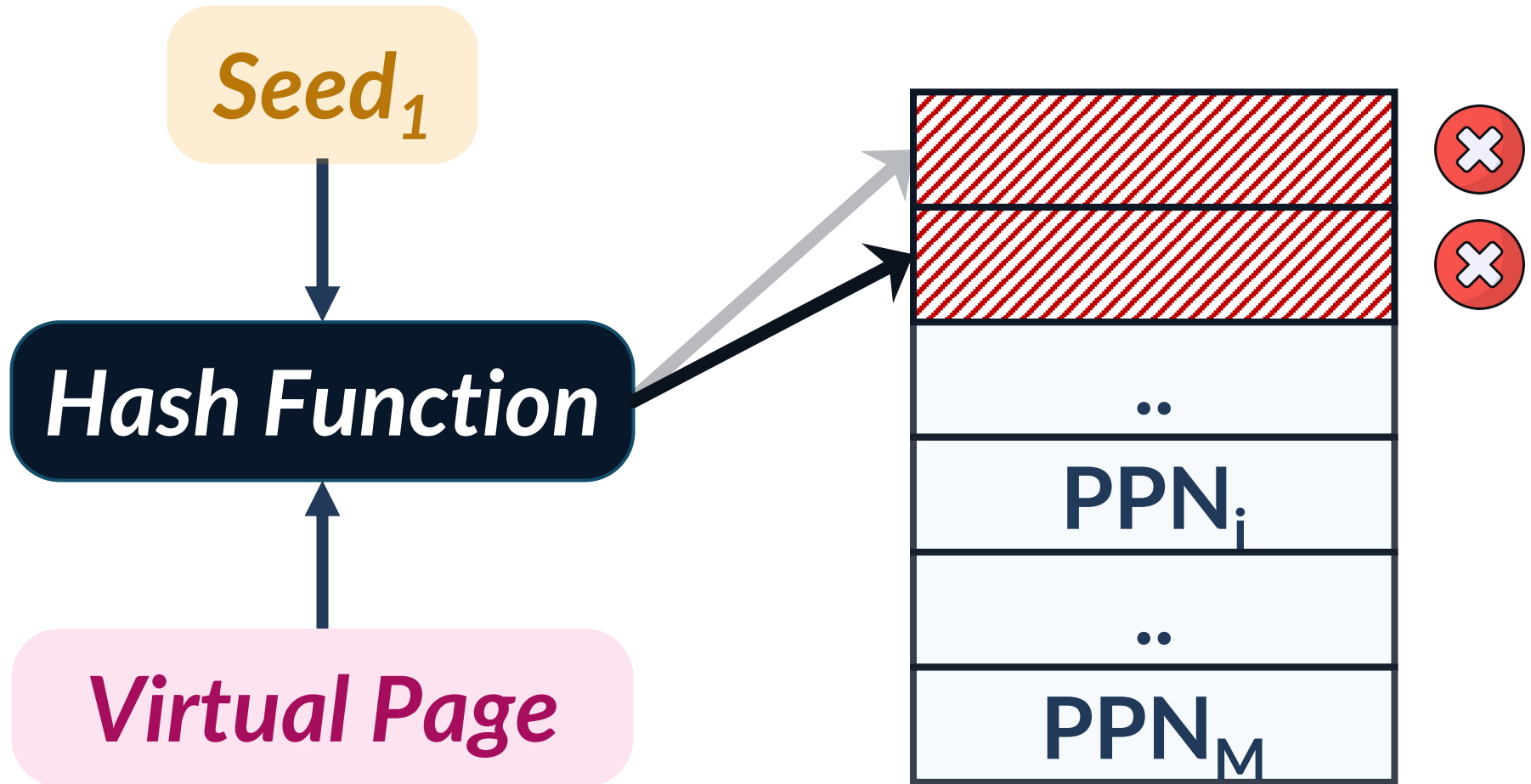
OS: Tiered Hash-based Allocation



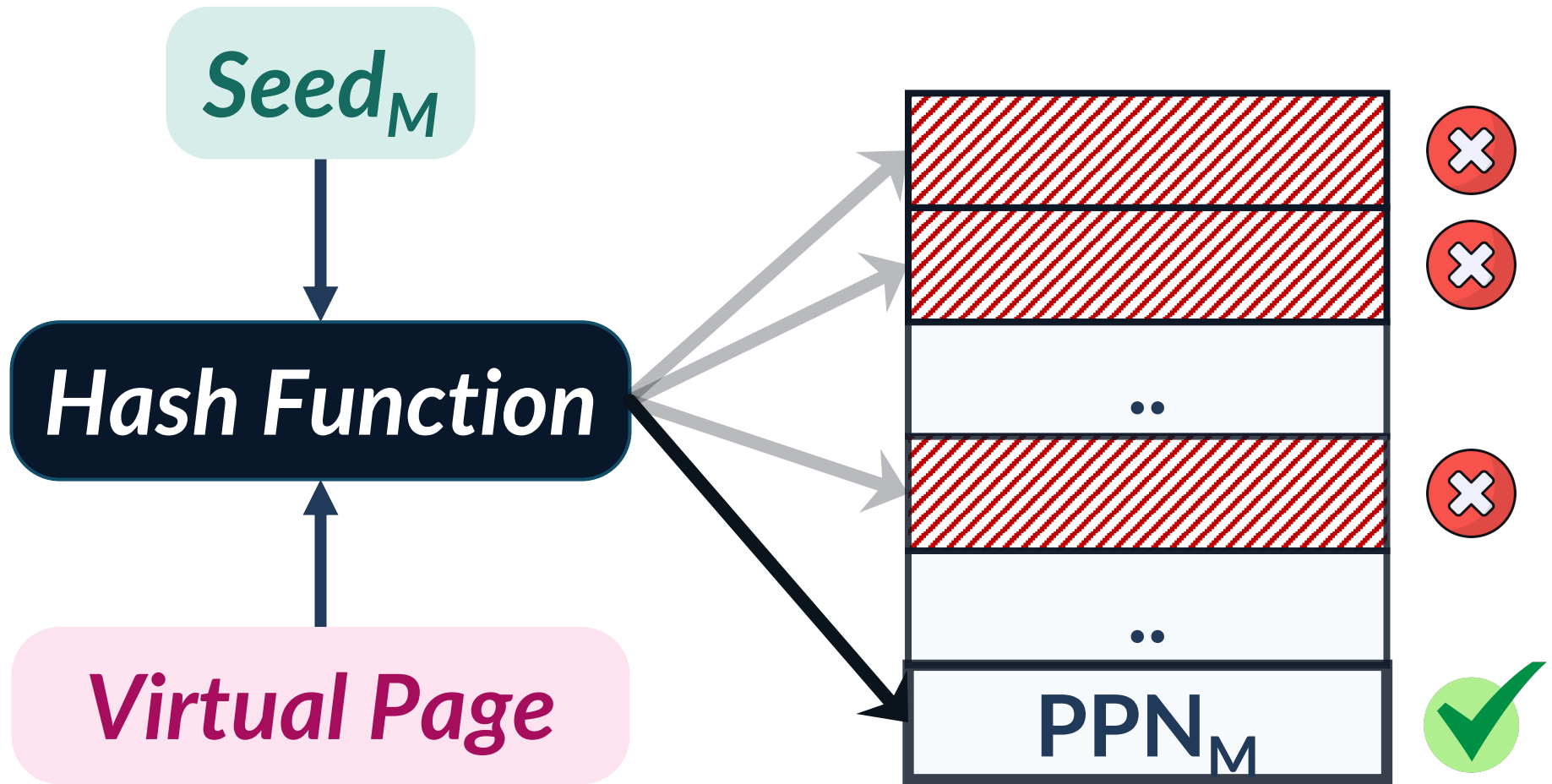
OS: Tiered Hash-based Allocation



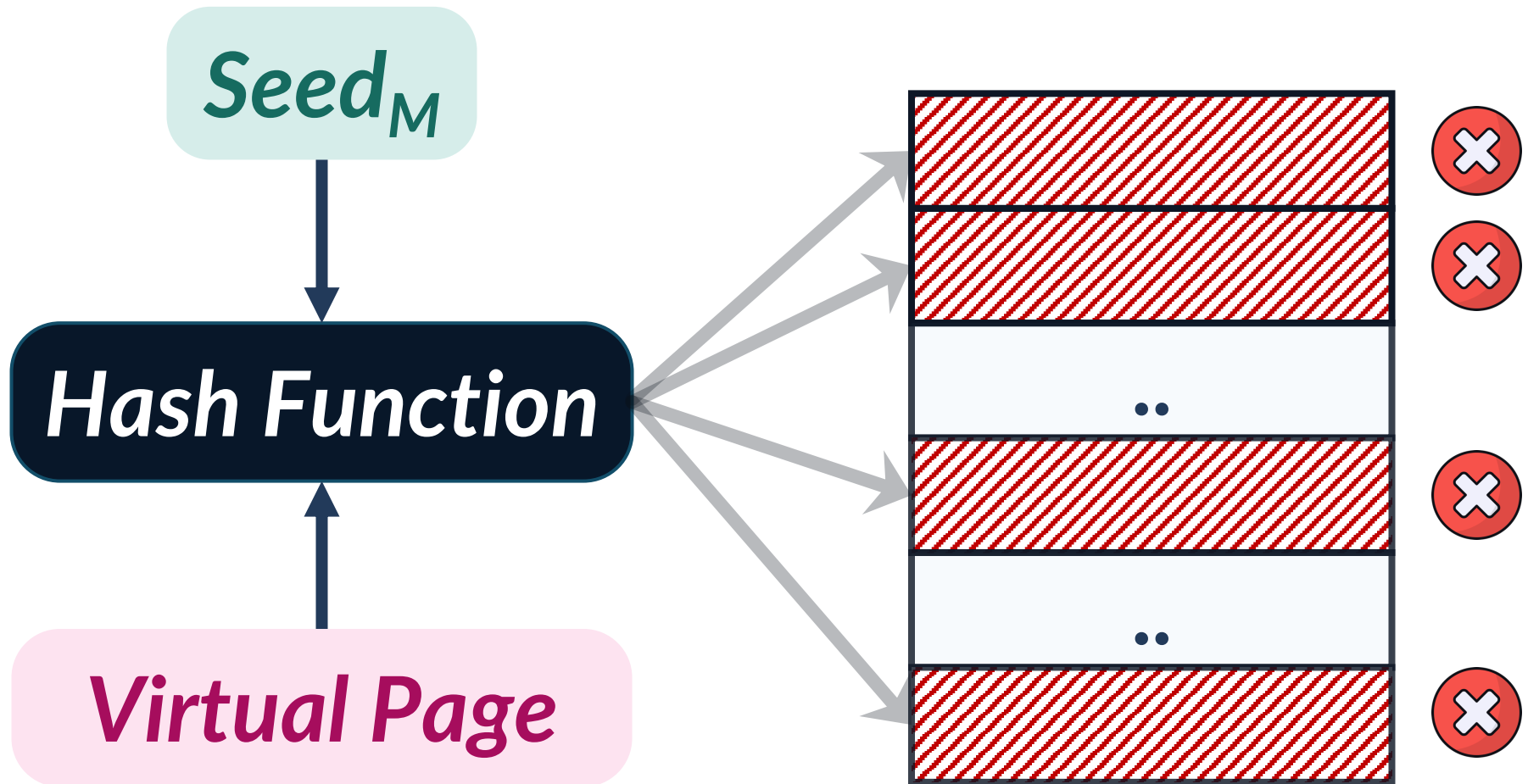
OS: Tiered Hash-based Allocation



OS: Tiered Hash-based Allocation

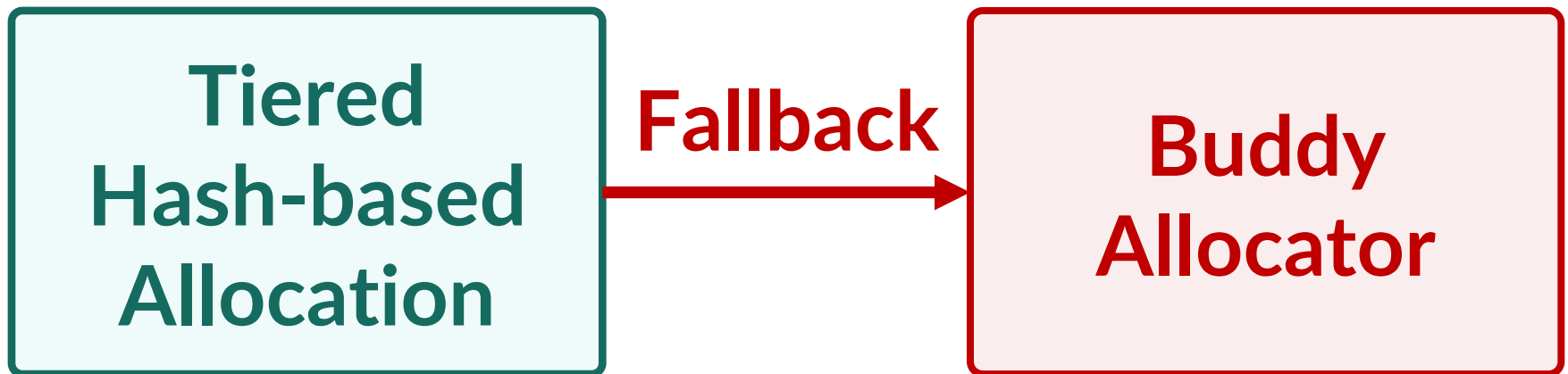


OS: Tiered Hash-based Allocation



OS: Tiered Hash-based Allocation

If all M hash attempts **fail**
Revelator **falls back**
to the conventional allocator



OS: Tiered Hash-based Allocation

If all N hash attempts fail
Revelator falls back

Does tiered hash-based
allocation work well?



OS: Tiered Hash-based Allocation

L = Number of occupied pages N = Total number of pages

- Each allocation attempt is a **Bernoulli trial**: a slot is free with probability $P = L / N$
- Each request makes k independent hash attempts
- Allocation fails only if **all k attempts** land on taken slots

Probability data is allocated with a hash function

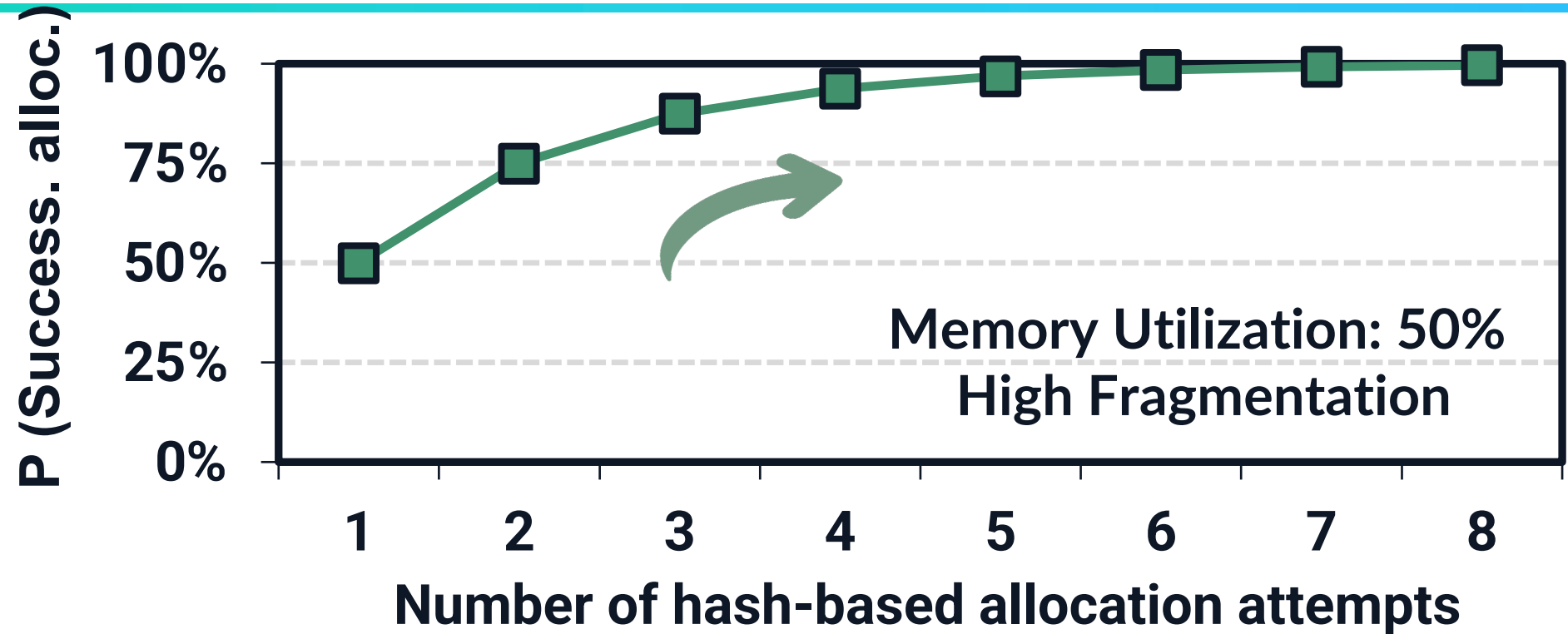
$$P(\text{success}) = 1 - (L / N)^k$$

Revelator: Allocation Properties

1

Number of hash-based allocation attempts

Number of Hash-based Alloc. Attempts



Increasing the number of hash-based allocation attempts **exponentially increases** the probability of successful allocation

Revelator: Allocation Properties

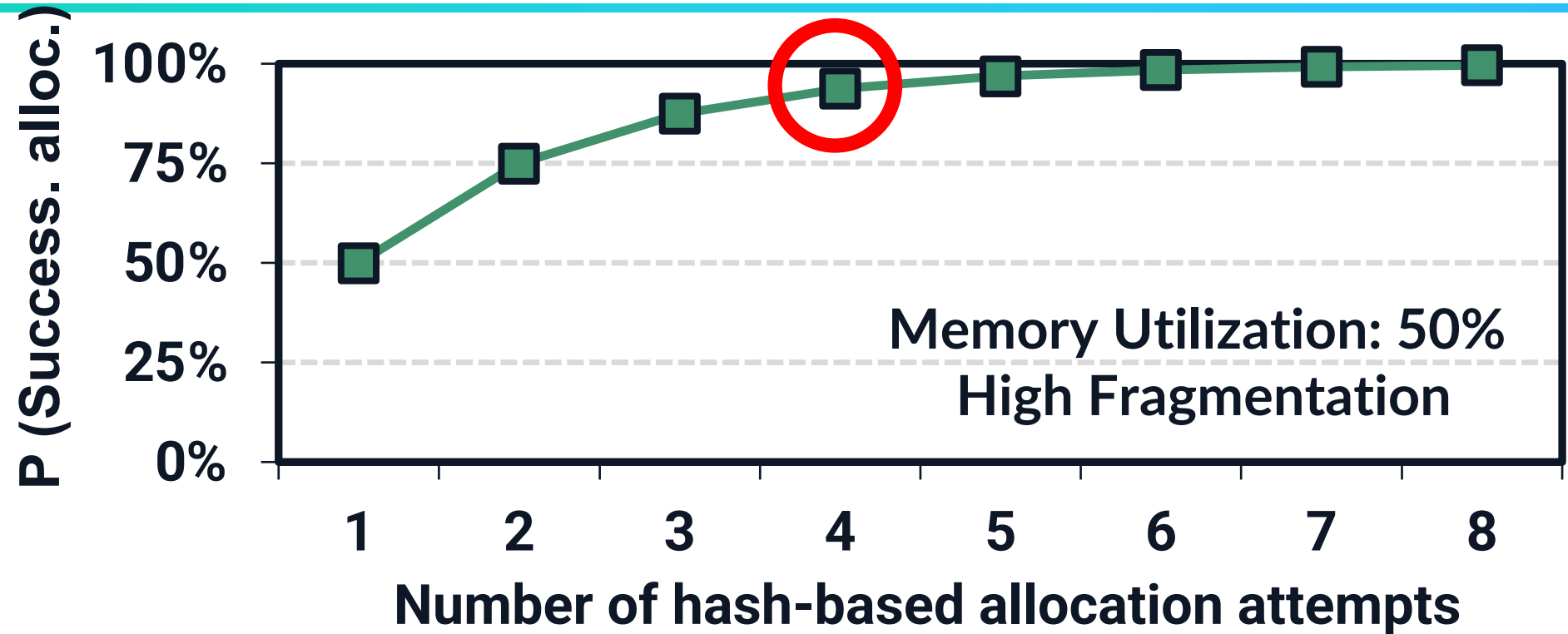
1

Number of hash-based allocation attempts

2

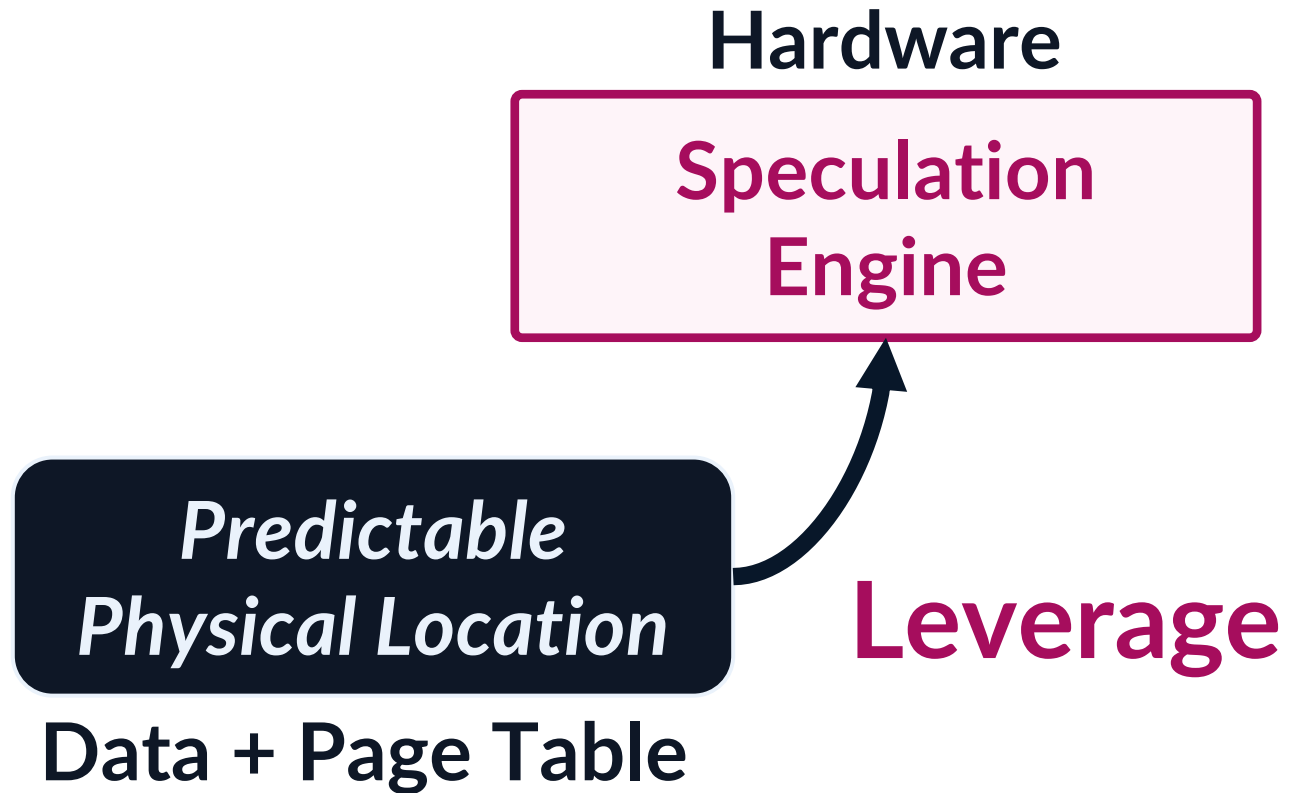
Allocation success is independent
from memory fragmentation

Allocation Success is Independent from Memory Fragmentation

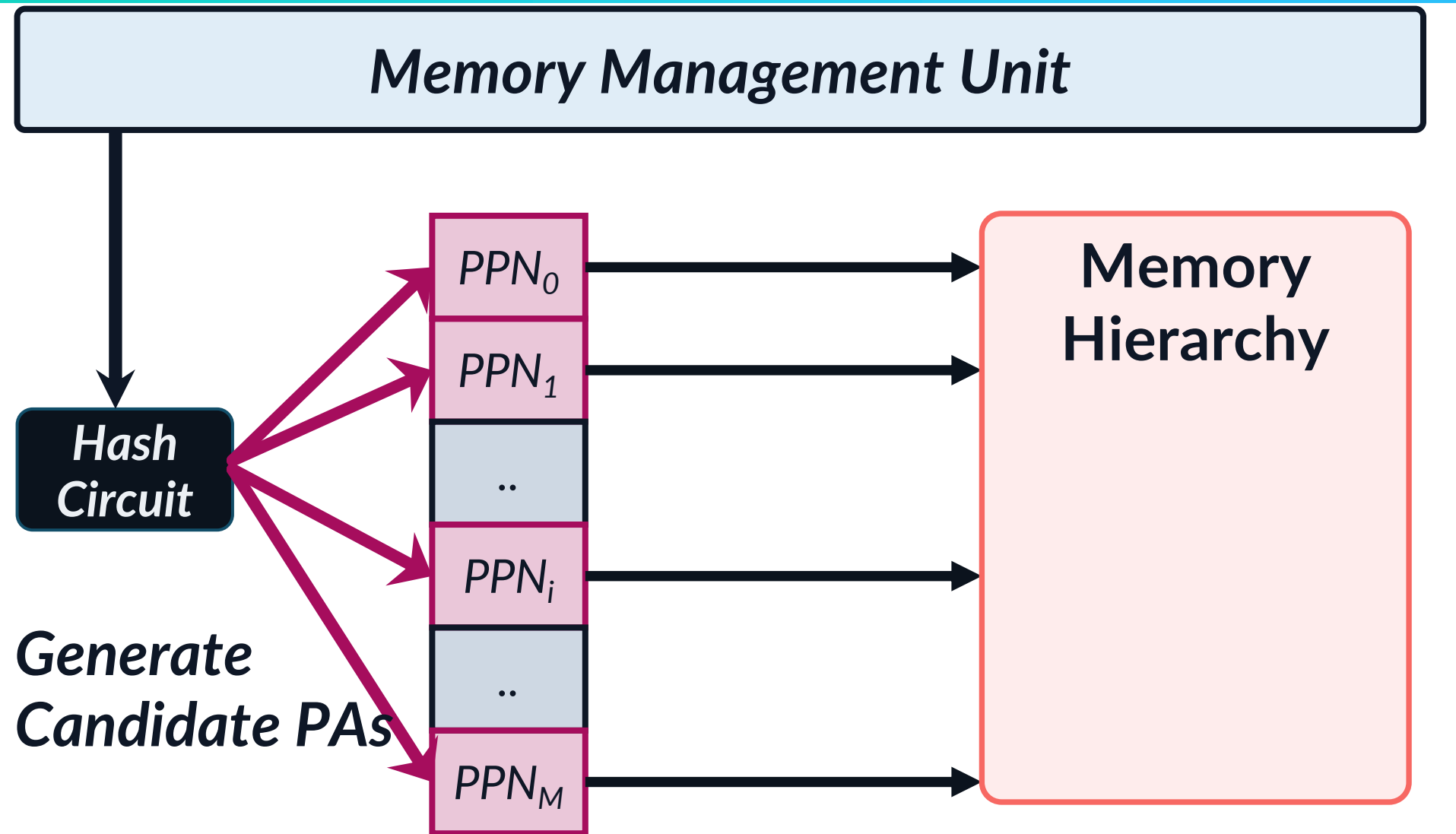


Even at **high fragmentation**, using 4 hash attempts leads to **>90% successful allocation**

Revelator: Key Components



Hardware: Speculation Engine



Revelator: Speculation Engine

Memory Management Unit

*Hash
Circuit*

PPN_0

PPN_1

..

PPN_i

..

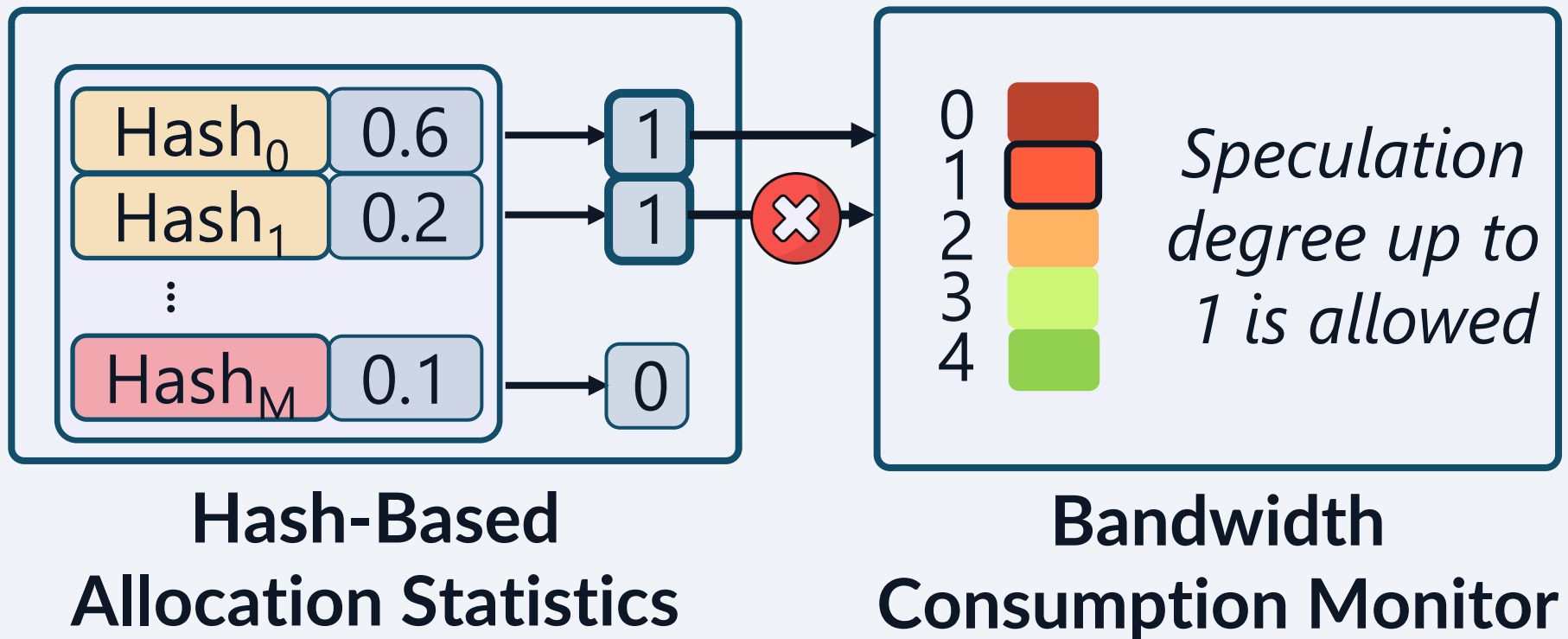
PPN_M

Filter

**Memory
Hierarchy**

Speculation Degree Filter

Two-level Degree Filtering



Revelator: Speculation Engine

Memory Management Unit

Hash
Circuit

PPN_0

PPN_1

..

PPN_i

..

PPN_M

Filter

Memory
Hierarchy

Data **stored in PPN_0** is
fetched into the cache
before PTW completes

Extensions of Revelator

Revelator: Extensions

1

Synergy with large pages

2

Virtualized environments

3

NUMA/CXL-like systems

Revelator: Extensions

1

Synergy with large pages

2

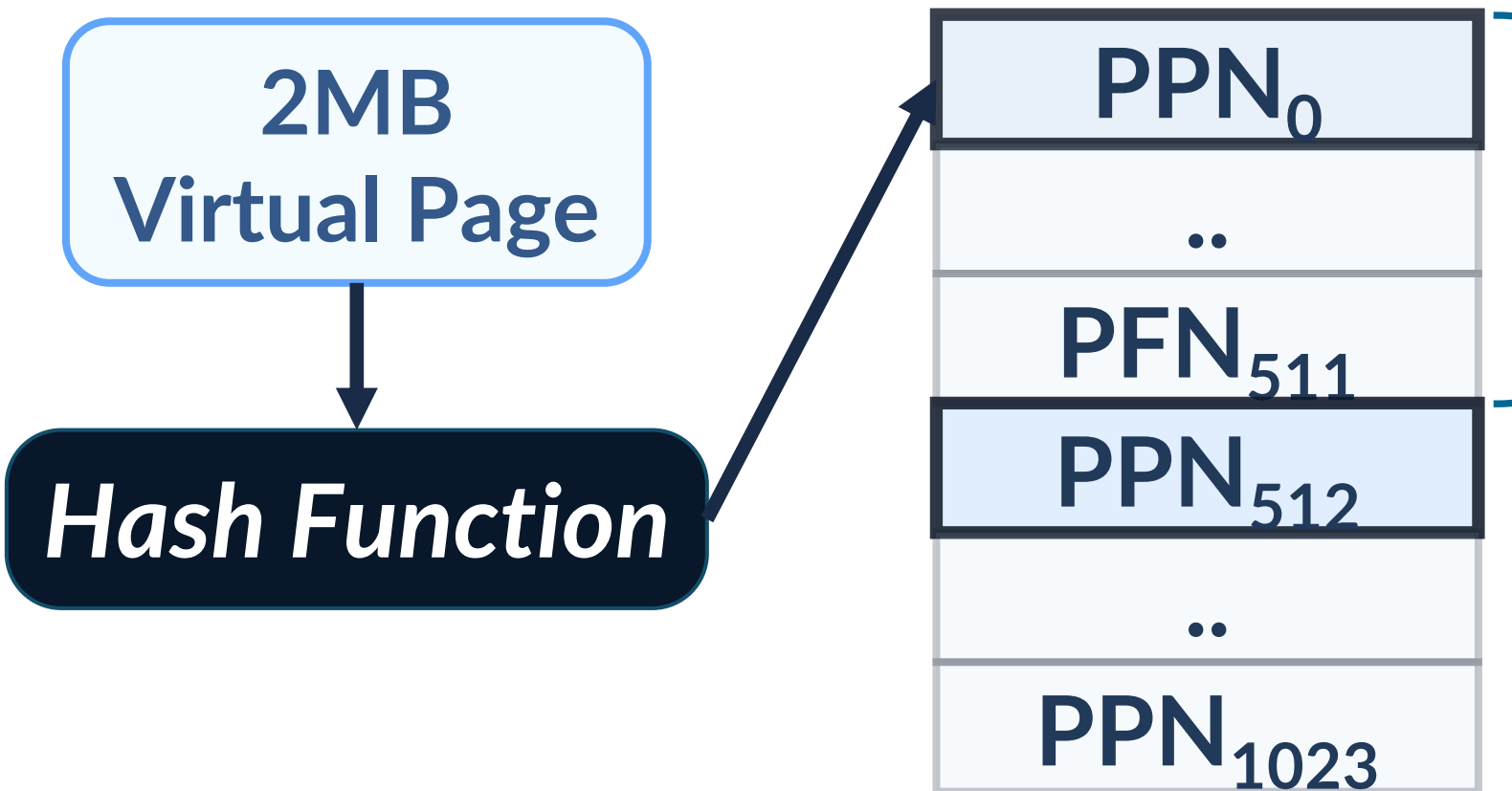
Virtualized environments

3

NUMA/CXL-like systems

Synergy with Large Pages

- 1 Try tiered hash-based allocation for 2MB pages



Synergy with Large Pages

- 1 *Try tiered hash-based allocation for 2MB pages*

Hide address translation
latency of 2MB pages



Hash Function

PPN₁₀₂₃

Synergy with Large Pages

1 *Try tiered hash-based allocation for 2MB pages*

2 *Try to allocate with conventional THP*

Use large pages when
fragmentation levels allow it



Synergy with Large Pages

1 *Try tiered hash-based allocation for 2MB pages*

2 *Try to allocate with conventional THP*

3 *Try 4KB hash-based data allocation*

**Maintain fast address translation
when large pages are not available**



Synergy with Large Pages

1 *Try tiered hash-based allocation for 2MB pages*

2 *Try to allocate with conventional THP*

3 *Try 4KB hash-based data allocation*

4 *Fallback to conventional allocation*

Synergy with Large Pages

1 *Try tiered hash-based allocation for 2MB pages*

Revelator uses large pages 
when possible while being effective
even when they are not available

4 *Fallback to conventional allocation*

Revelator: Extensions

1

Synergy with large pages

2

Virtualized environments

3

NUMA/CXL-like systems

Revelator: Extensions

Revelator: Rapid Data Fetching via System-Software-Guided Hash-based Speculative Address Translation

Konstantinos Kanellopoulos¹ Konstantinos Sgouras¹ Harsh Songara¹
Andreas Kosmas Kakolyris¹ Vlad-Petru Nitu¹ Spiros Galanopoulos¹ Rahul Bera¹
Konstantina Koliogeorgi¹ Rakesh Kumar² Onur Mutlu¹

¹ETH Zürich ²NTNU

Address translation is a significant performance bottleneck in modern computing systems. Predicting the physical address (PA) of the requested data before address translation completes is a promising technique to hide the latency of address translation. However, accurately predicting the PA based on the virtual address (VA) is highly challenging due to the inherent unpre-

dictable nature of the address translation process, which can take up to 40-45% of total execution time (15-20% with large pages) [1, 2, 5, 6, 14–23]. Address translation overheads are projected to grow as architectures transition toward larger physical address spaces, driven by paradigms like disaggregated memory [24–48], tiered memory systems [49–65], and unified virtual memory (UVM) in heterogeneous architectures [66–87].

<https://arxiv.org/abs/2508.02007>

Evaluation Results

Evaluation Methodology

Prototyped the SW component of Revelator in **Linux 6.10.8**

- Modifications in the buddy allocator
- Extensions on top of THP

Evaluation using **Virtuoso [Kanellopoulos+ ASPLOS'25]** ported on top of **Sniper [Carlson+ SC'11]**

Open source: <https://github.com/CMU-SAFARI/Virtuoso>

Workloads: Executed for 300M instructions

GraphBIG: PR, BFS, BC, GC, CC

HPCC: Randacc

XSbench: Particle Simulation

DLRM: Sparse-Length Sum

GenomicsBench: k-mer count

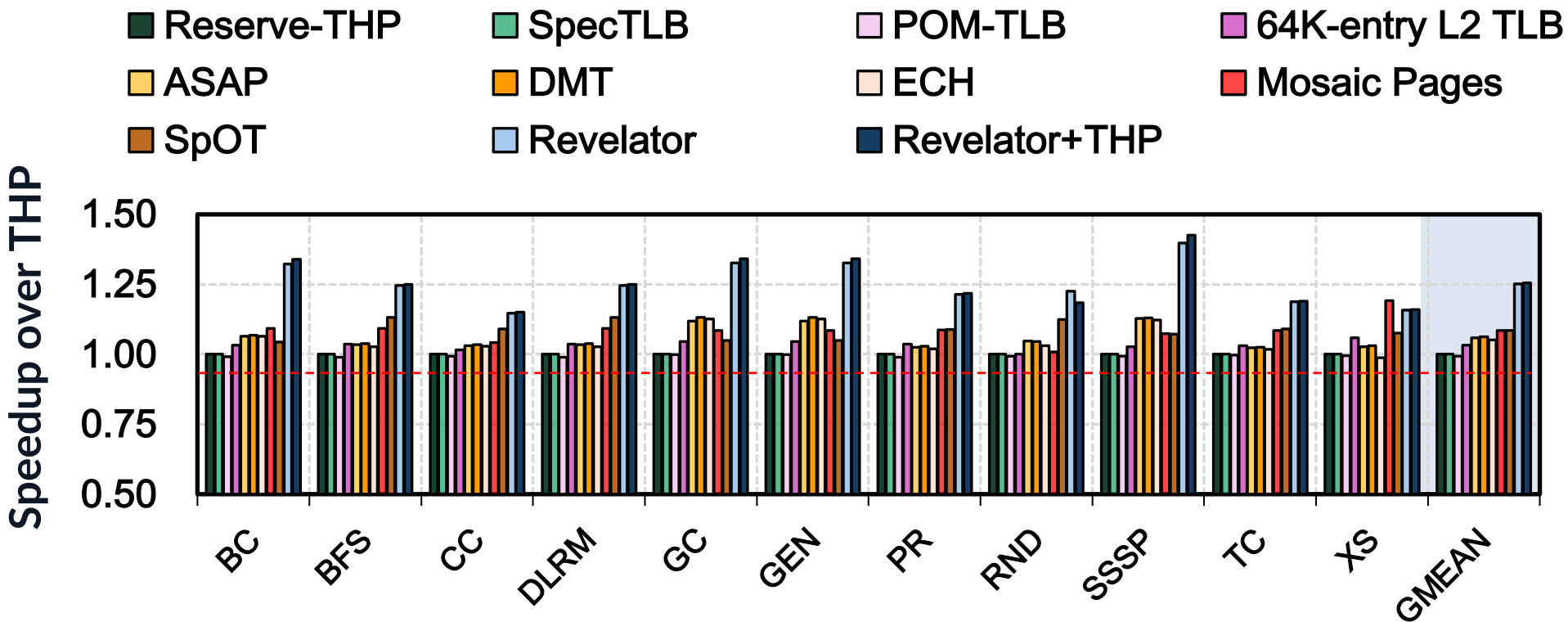
Google DPC-4 mixes in multicore systems

Single-core systems

Evaluated Systems

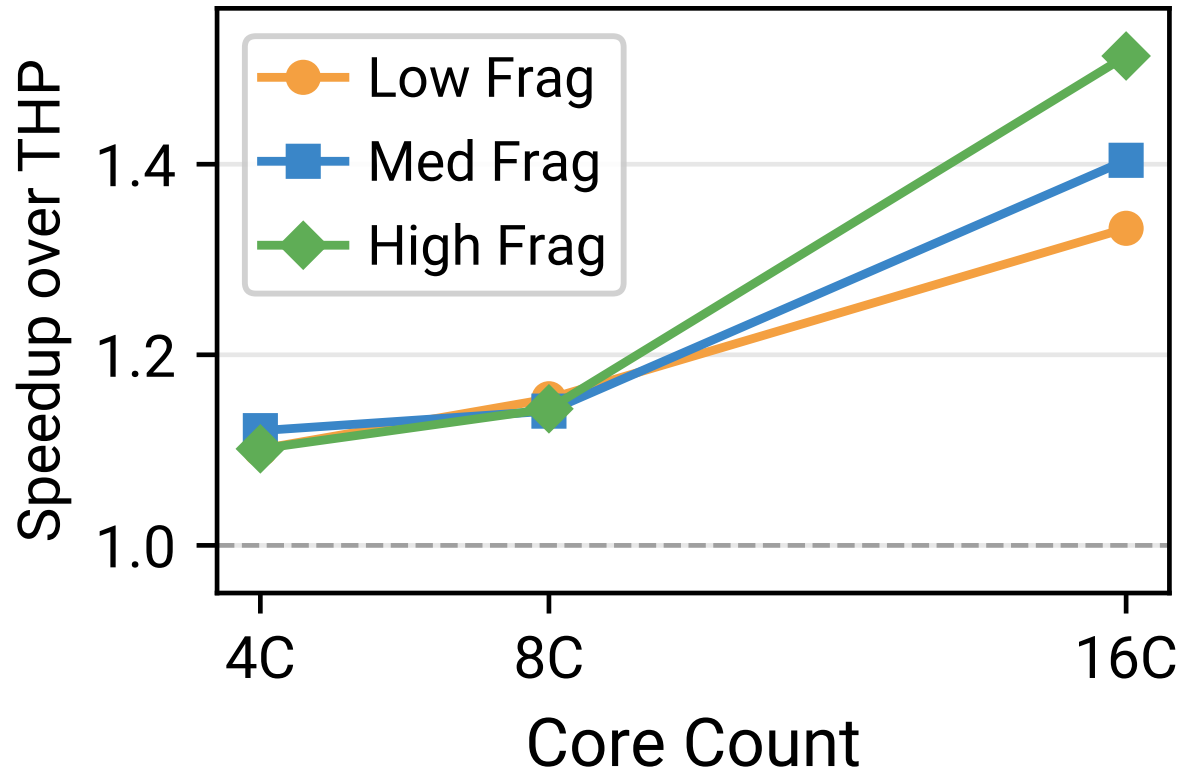
1. **THP**: Baseline with THP enabled
2. **ReserveTHP**: System with reservation-based THP
3. **SpecTLB [Barr+ ISCA'11]**: Speculative translation scheme for reserved large pages
4. **POM-TLB [Ryoo+ ISCA'17]**: Large software-managed 64K-entry L3 TLB
5. **64K-entry L2 TLB**: L2 TLB with optimistic 12-cycle access latency
6. **ASAP [Margaritov+ MICRO'19]**: Intra-page table walk prefetching
7. **DMT [Zhang+ ASPLOS'24]**: Direct memory translation via VA-to-PTE mappings
8. **ECHT [Skarlatos+ ASPLOS'20]**: State of the art page table design
9. **Mosaic Pages [Gosakan+ ASPLOS'24]**: Large TLB reach via hash-based allocation
10. **SpOT [Alverti+ ISCA'20]**: Speculative translation via contiguous allocation
11. **Revelator**: Tiered hash-based allocation + speculation engine
12. **Revelator+THP**

Single-Core: Performance Speedup



Revelator outperforms the best-performing scheme by 17% on average when memory is highly fragmented

Multi-Core: Performance Speedup



Revelator achieves increasing performance speedups (up to 1.5x) against THP with increasing core counts

Area and Power Overhead

We implement the speculation engine in Chisel and **synthesize it** with Yosys using the NanGate 45nm open-source cell library

0.02%

Additional Area

0.0149 mm²

0.03%

Additional Power

14.723 mW static power

*Compared with a high-performance
Intel Cascade Lake core*

More in the Paper

- Analysis on the shortcomings of prior works
- Deep dive into Revelator's performance
- Evaluation in NUMA and virtualized systems
- Revelator combined with other schemes
- Energy reduction achieved by Revelator
- Sensitivity to the number of allocation attempts
- Cache pollution caused by Revelator
- Security implications

More in the Paper

Revelator: Rapid Data Fetching via System-Software-Guided Hash-based Speculative Address Translation

Konstantinos Kanellopoulos¹ Konstantinos Sgouras¹ Harsh Songara¹
Andreas Kosmas Kakolyris¹ Vlad-Petru Nitu¹ Spiros Galanopoulos¹ Rahul Bera¹
Konstantina Koliogeorgi¹ Rakesh Kumar² Onur Mutlu¹

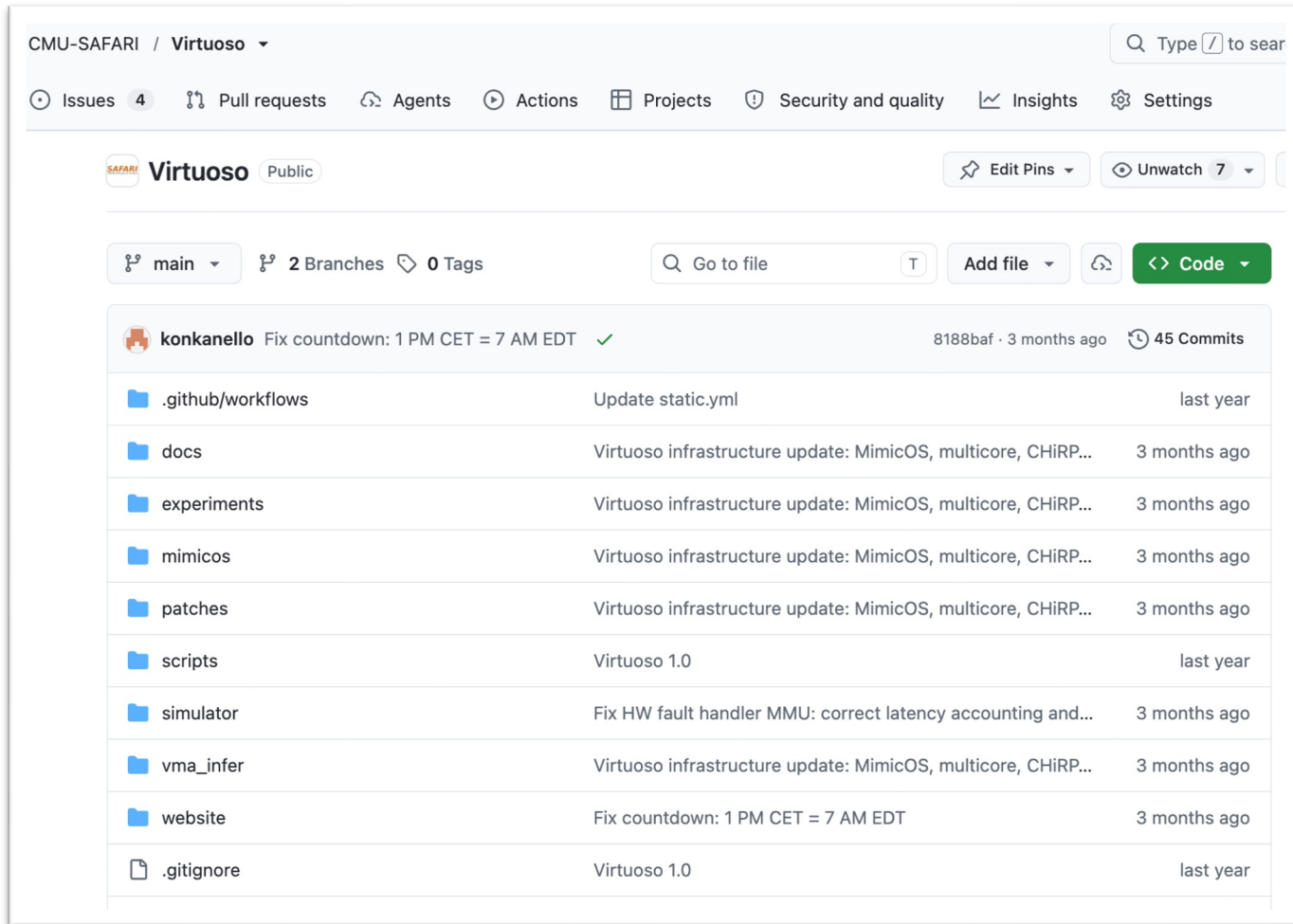
¹ETH Zürich ²NTNU

Address translation is a significant performance bottleneck in modern computing systems. Predicting the physical address (PA) of the requested data before address translation completes is a promising technique to hide the latency of address translation. However, accurately predicting the PA based on the virtual address (VA) is highly challenging due to the inherent unpre-

dictable nature of address translation, which can take up to 40-45% of total execution time (15-20% with large pages) [1, 2, 5, 6, 14–23]. Address translation overheads are projected to grow as architectures transition toward larger physical address spaces, driven by paradigms like disaggregated memory [24–48], tiered memory systems [49–65], and unified virtual memory (UVM) in heterogeneous architectures [66–87].

<https://arxiv.org/abs/2508.02007>

Revelator is Open-Source



CMU-SAFARI / Virtuoso

Issues 4 Pull requests Agents Actions Projects Security and quality Insights Settings

Virtuoso Public Edit Pins Unwatch 7

main 2 Branches 0 Tags Go to file Add file Code

konkanello Fix countdown: 1 PM CET = 7 AM EDT ✓ 8188baf · 3 months ago 45 Commits

.github/workflows	Update static.yml	last year
docs	Virtuoso infrastructure update: MimicOS, multicore, CHiRP...	3 months ago
experiments	Virtuoso infrastructure update: MimicOS, multicore, CHiRP...	3 months ago
mimicos	Virtuoso infrastructure update: MimicOS, multicore, CHiRP...	3 months ago
patches	Virtuoso infrastructure update: MimicOS, multicore, CHiRP...	3 months ago
scripts	Virtuoso 1.0	last year
simulator	Fix HW fault handler MMU: correct latency accounting and...	3 months ago
vma_infer	Virtuoso infrastructure update: MimicOS, multicore, CHiRP...	3 months ago
website	Fix countdown: 1 PM CET = 7 AM EDT	3 months ago
.gitignore	Virtuoso 1.0	last year

<https://github.com/CMU-SAFARI/Virtuoso>

Conclusion

Revelator: HW/OS Cooperation

Employ *hashing* to enable **highly accurate** speculative address translation with **small system modifications**

Operating System

Tiered Hash-
based Allocation



*Predictable
Physical Location*



Hardware

Speculation
Engine

Hides & reduces address translation latency **without** depending on memory fragmentation (i.e., large pages, contiguity)

**Outperforms all state-of-the-art schemes
in both single-core and multi-core systems**



Revelator



Rapid Data Fetching
via System-Software-Guided Hash-based
Speculative Address Translation

Konstantinos Kanellopoulos Konstantinos Sgouras
Harsh Songara Andreas Kosmas Kakolyris Vlad-Petru Nitu
Spiros Galanopoulos Rahul Bera Konstantina Koliogeorgi
Rakesh Kumar Onur Mutlu

SAFARI **ETH** zürich  NTNU

Revelator: Allocation Properties

1

Number of hash-based allocation attempts

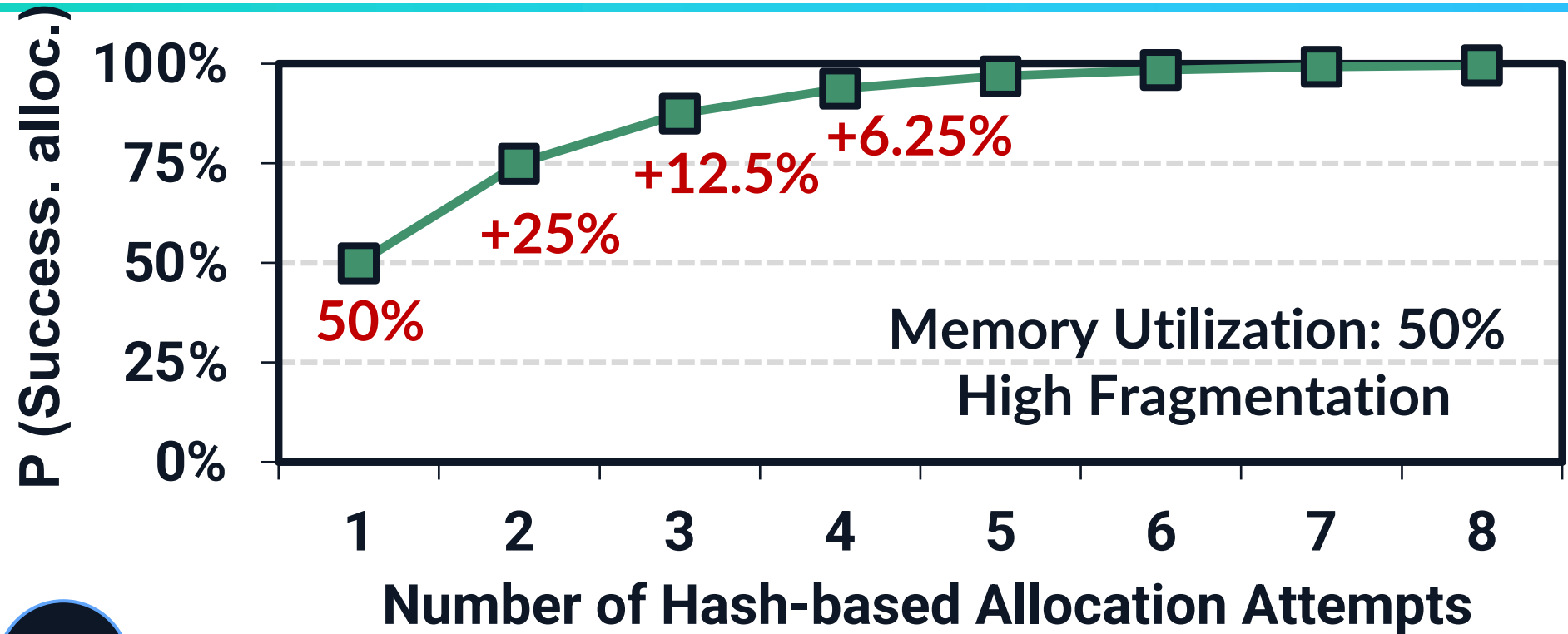
2

Dependence on memory fragmentation

3

Allocation bias

Allocation Bias

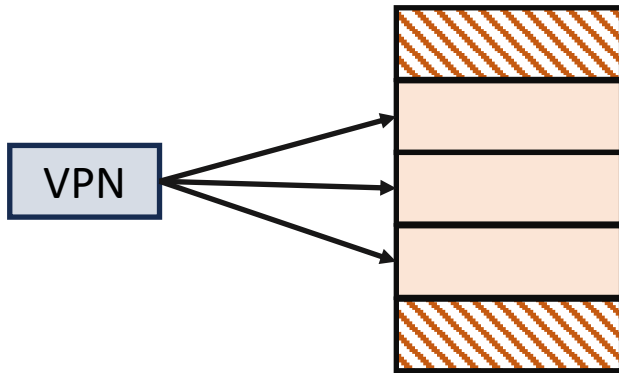


3

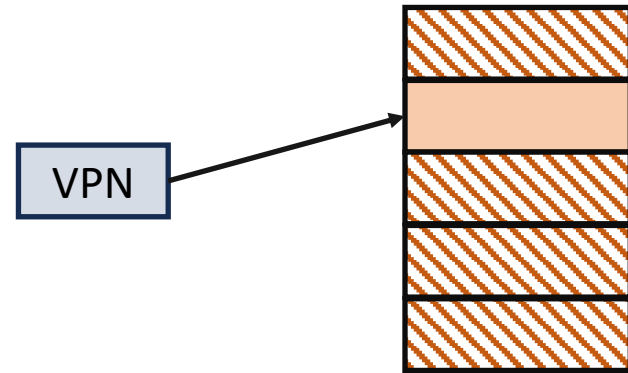
Sequential hashing biases allocations to succeed toward earlier hash functions

Example Scenarios of Allocation

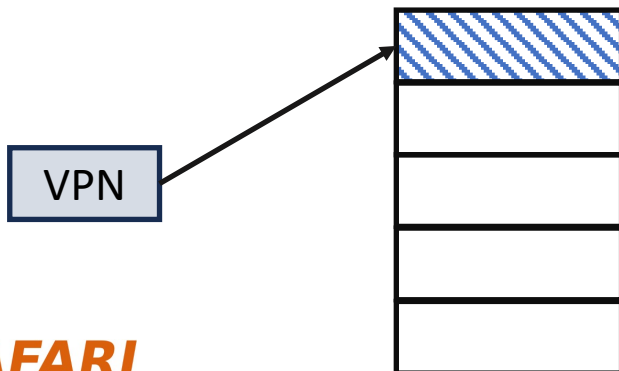
1 Low Fragmentation



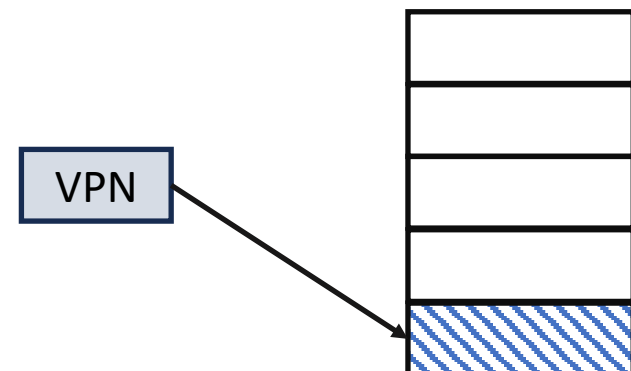
3 High Fragmentation



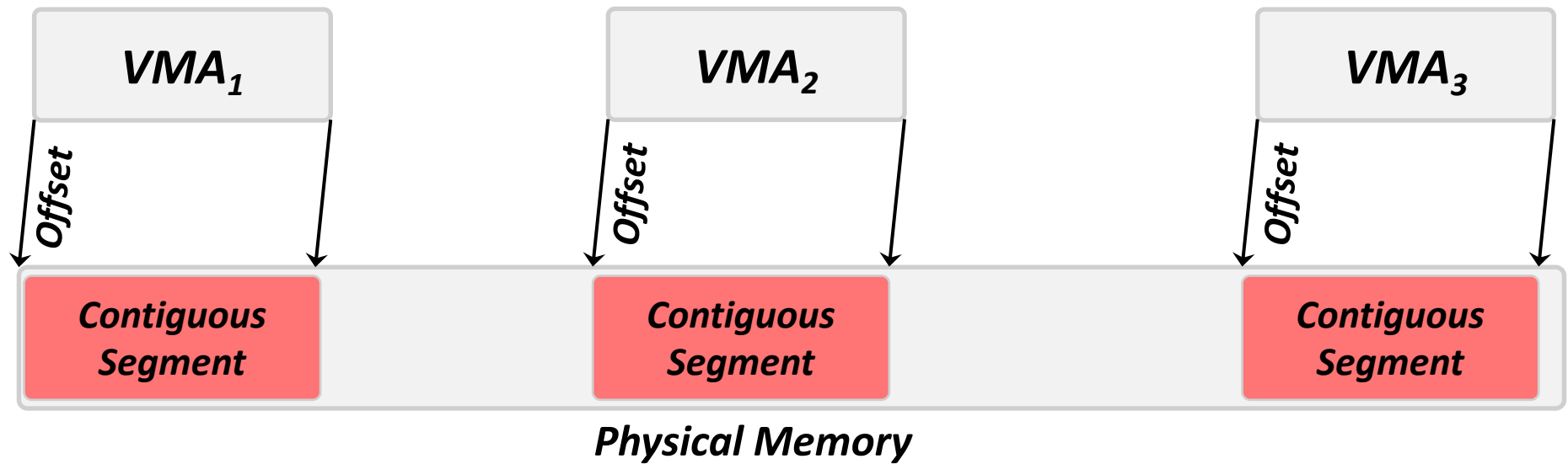
2 Allocated Early



4 Allocated Late



SpOT [Alverti et al. ISCA 2020]

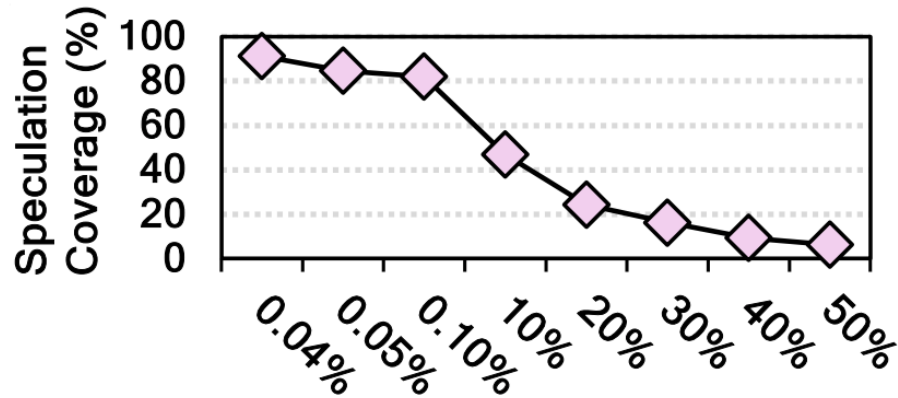
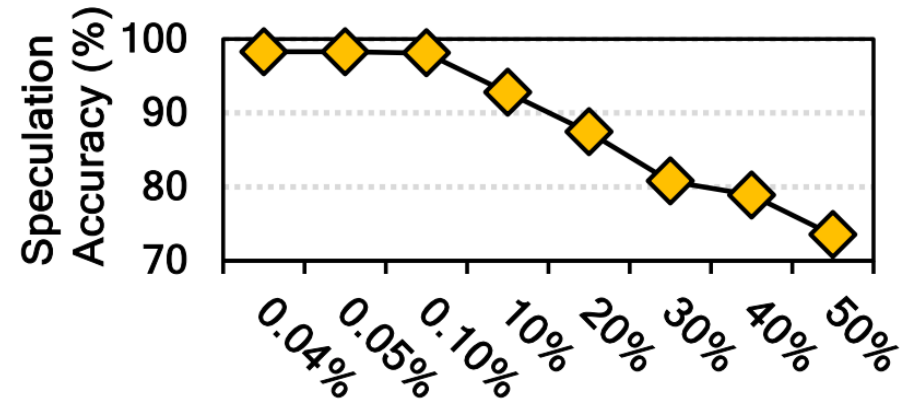


Hardware Table

VMA_1	$Offset_1$
VMA_2	$Offset_2$
VMA_3	$Offset_3$

Speculate PA assuming
VMA-Offset mapping is stable

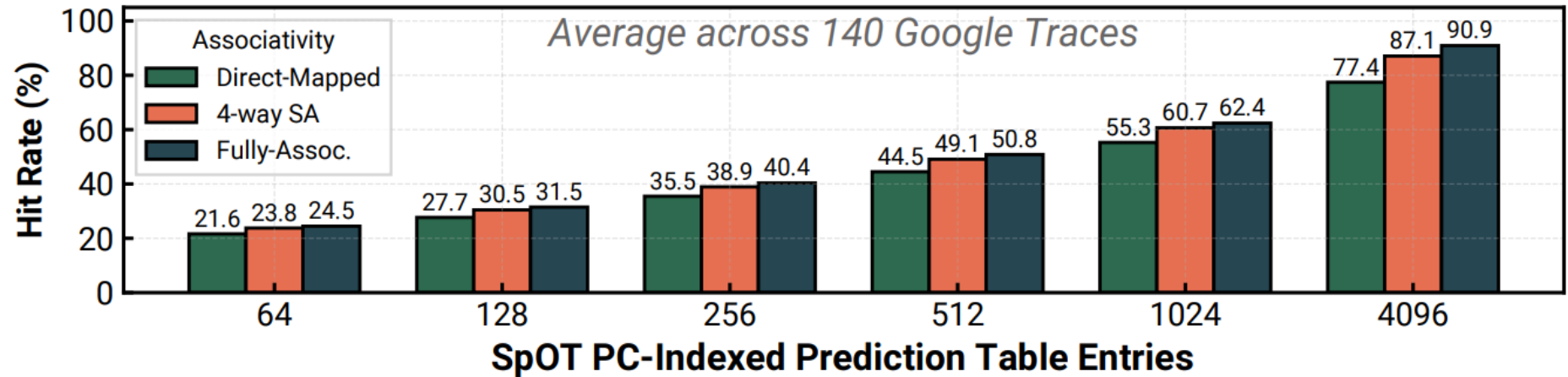
Limitations of SpOT (1)



Memory Fragmentation

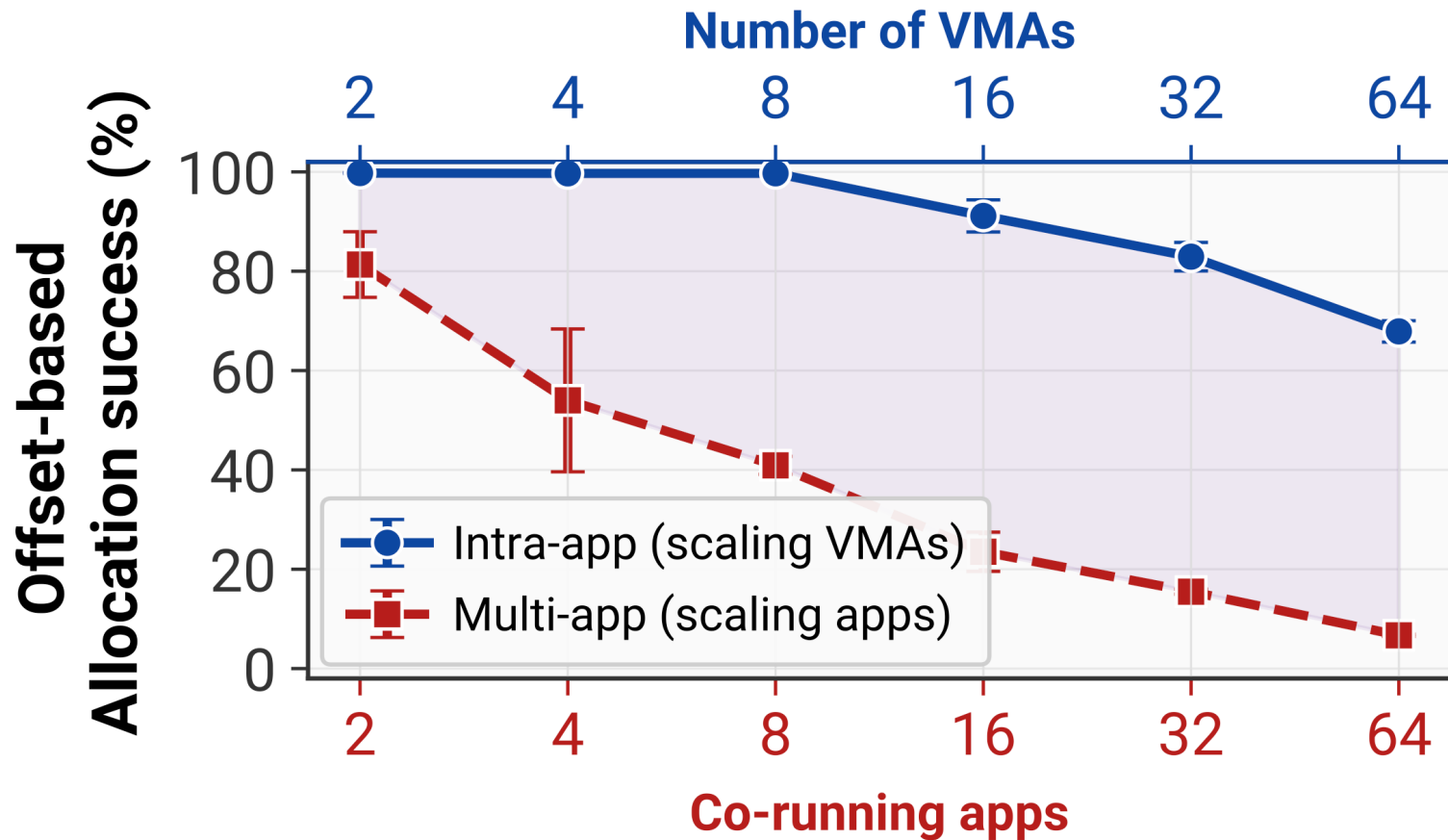
SpOT is sensitive to the amount of fragmentation in the system

Limitations of SpOT (2)



SpOT is sensitive to the number of allocation contexts

Limitations of Prior Work (3)



SpOT is sensitive to allocation interference

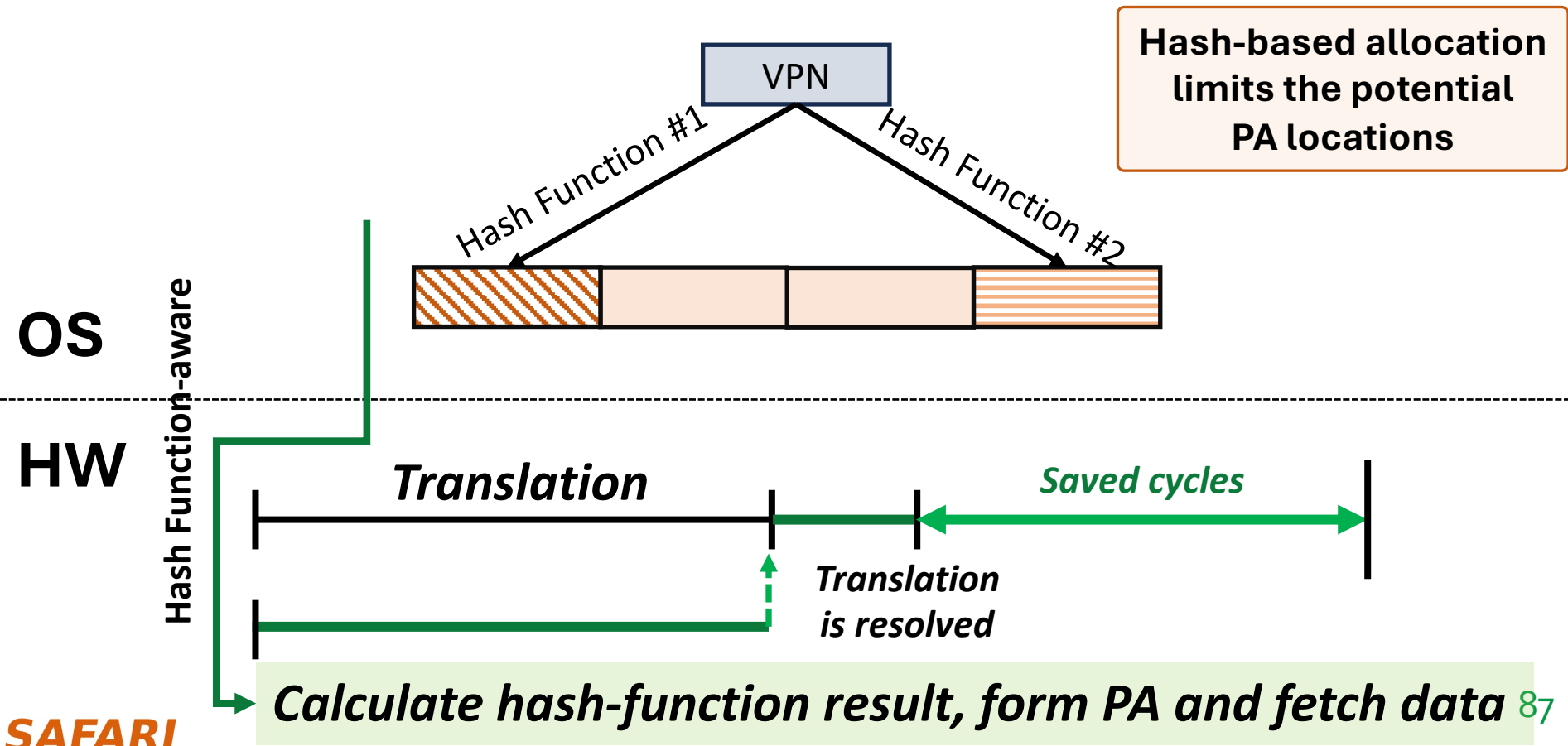
Our Goal

Hide address translation latency in modern systems

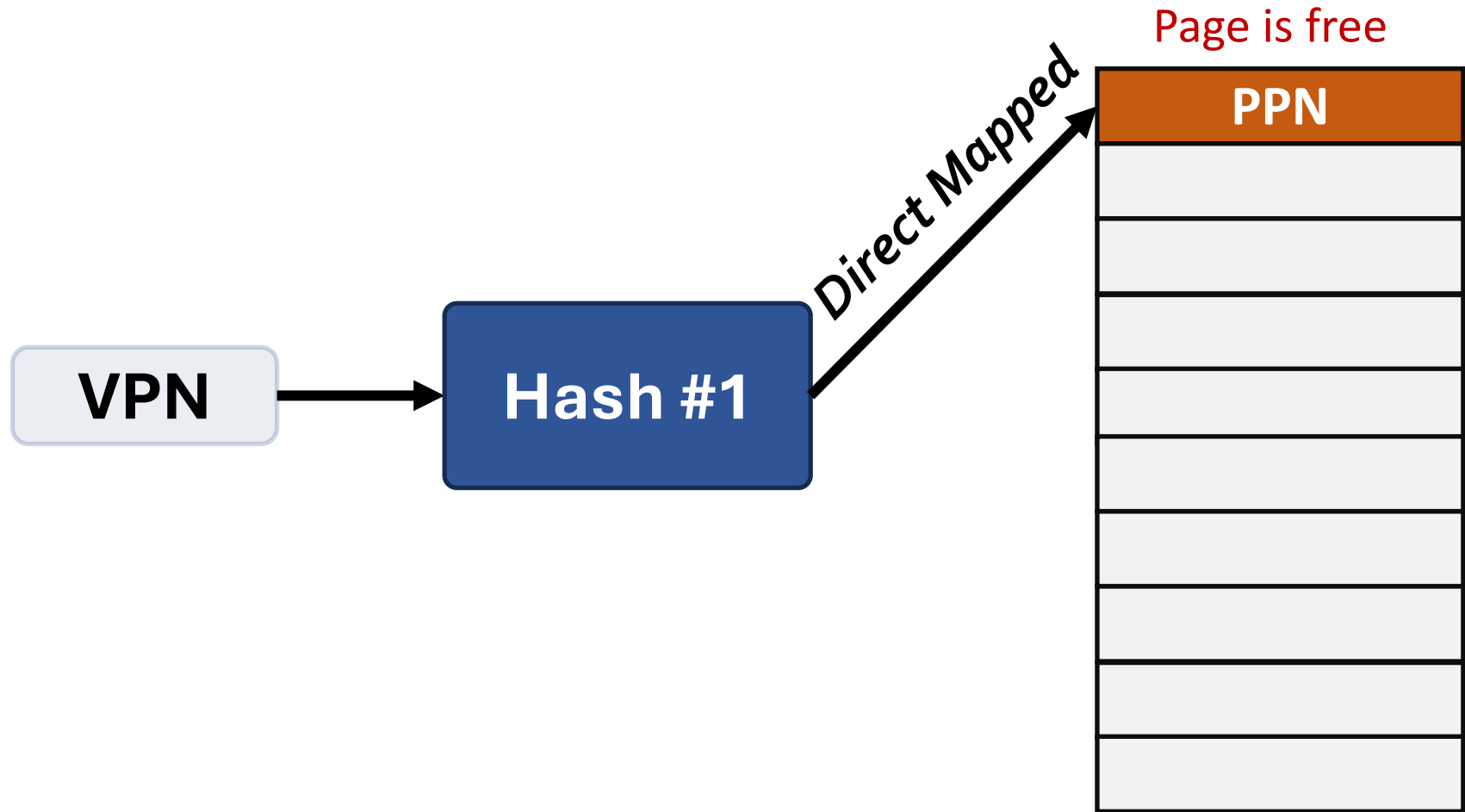
- (1) Without relying on **large pages**
- (2) Without relying on **VA-to-PA contiguity**
- (3) Without **costly hardware** for metadata

Revelator: Key Idea

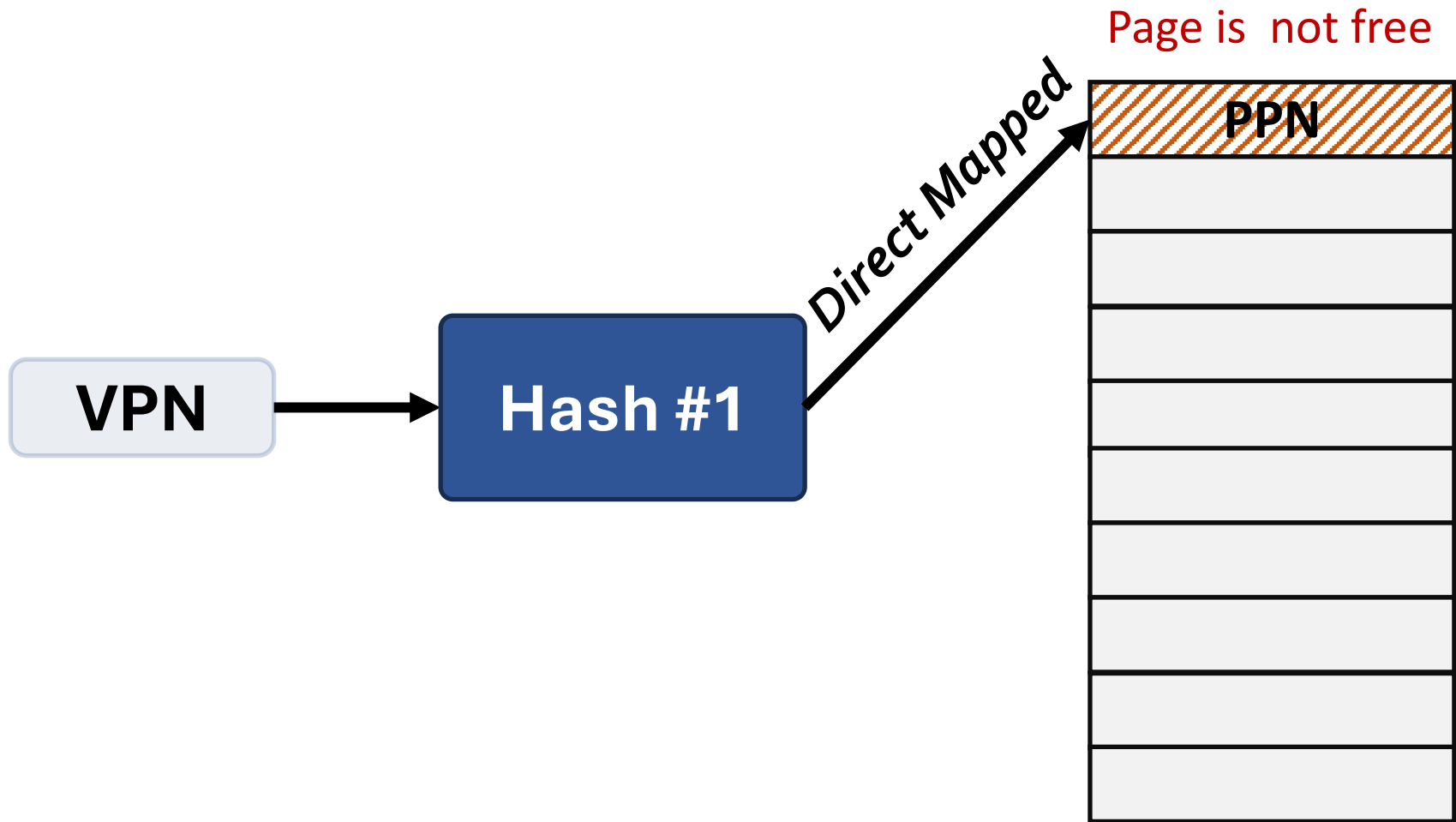
Co-design hardware and OS to perform highly accurate speculative translation



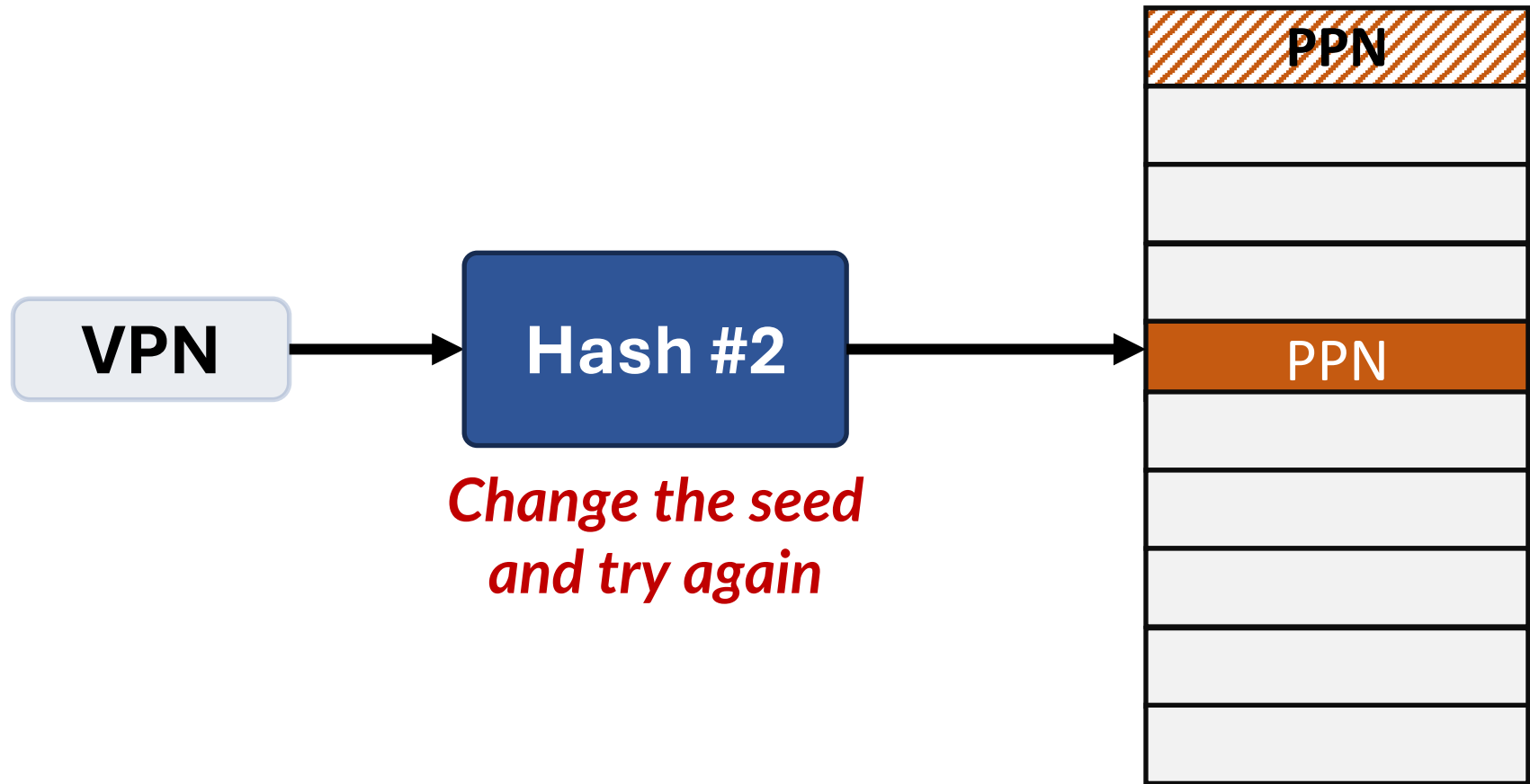
OS: Page Allocation Algorithm



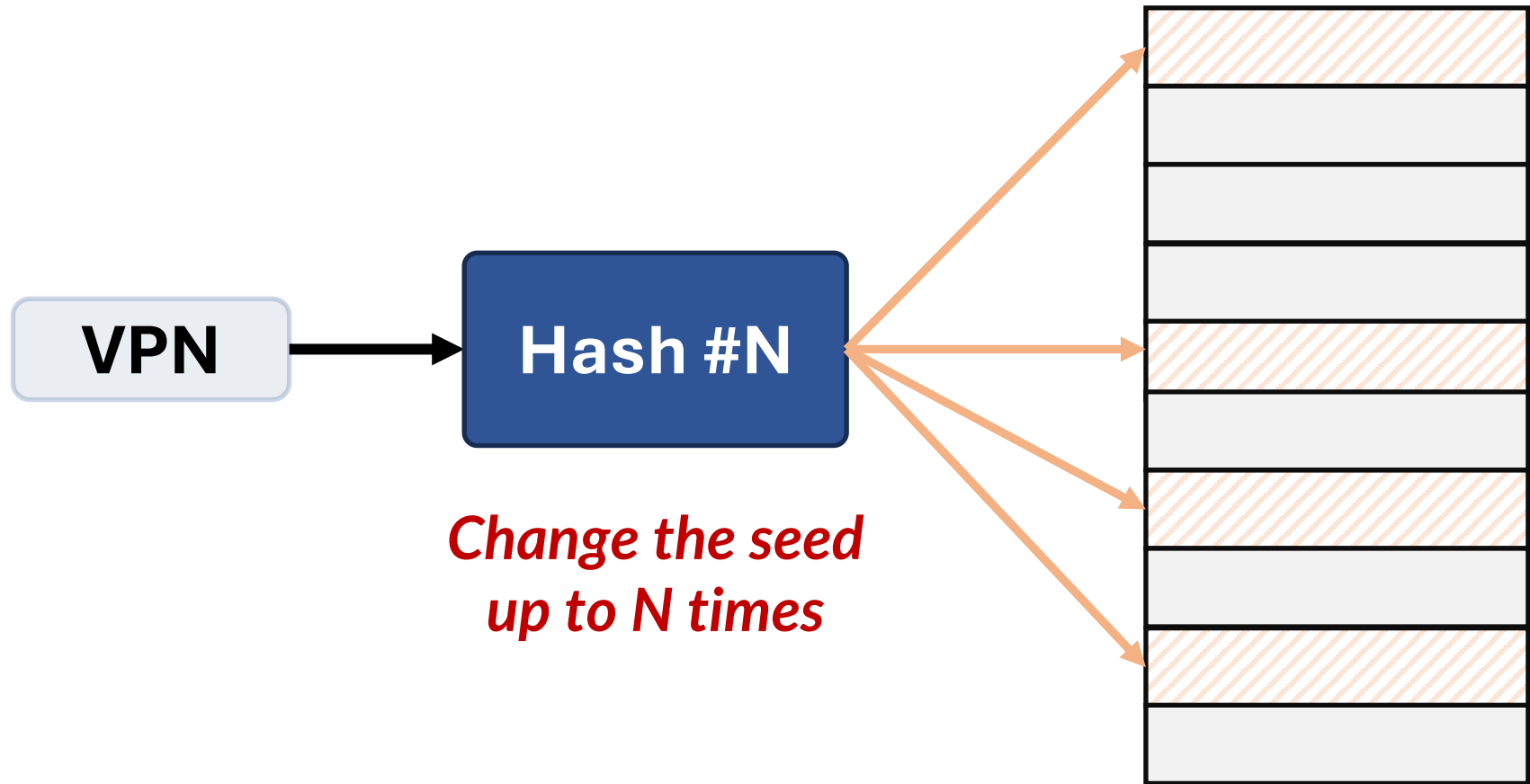
OS: Page Allocation Algorithm



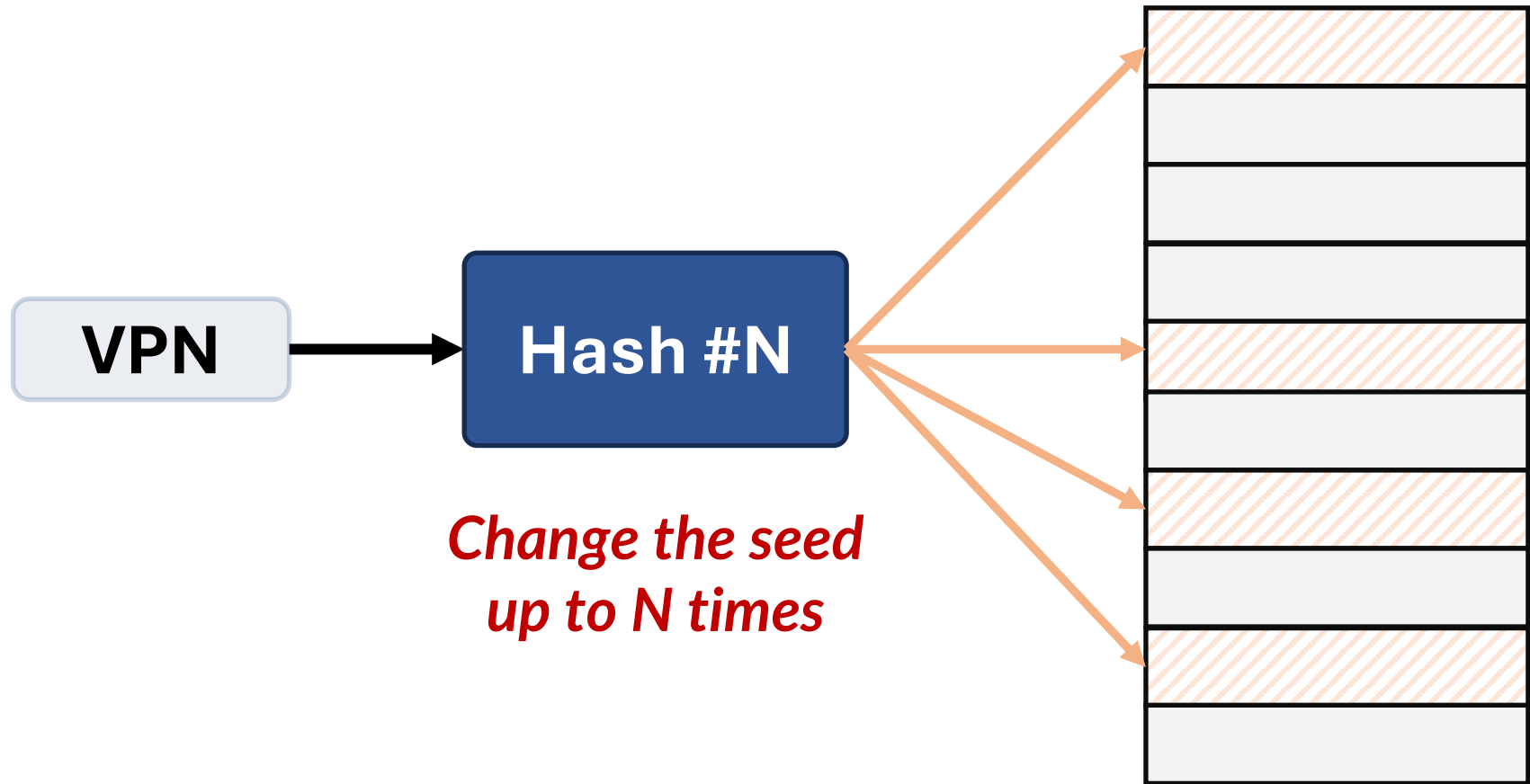
OS: Page Allocation Algorithm



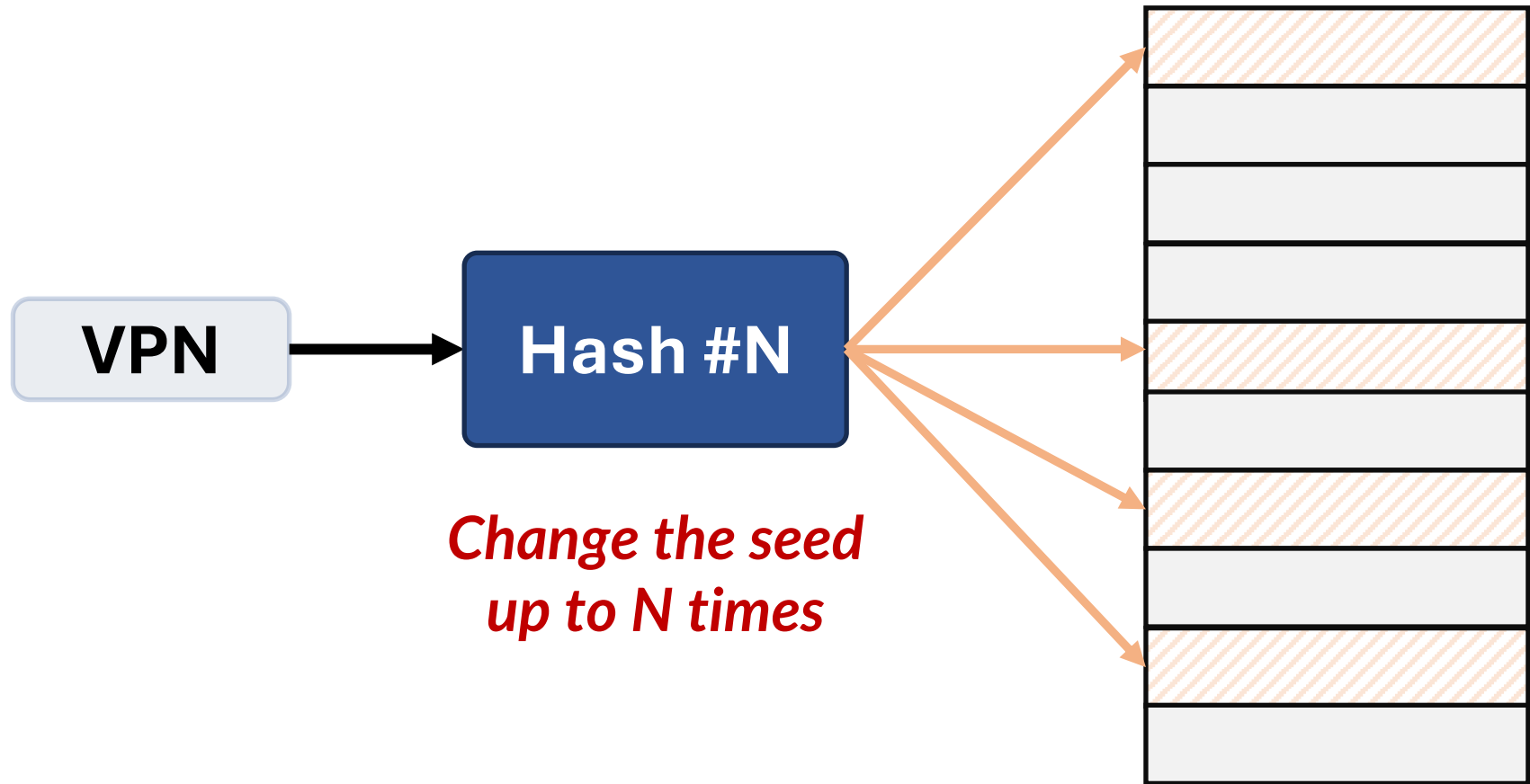
OS: Page Allocation Algorithm



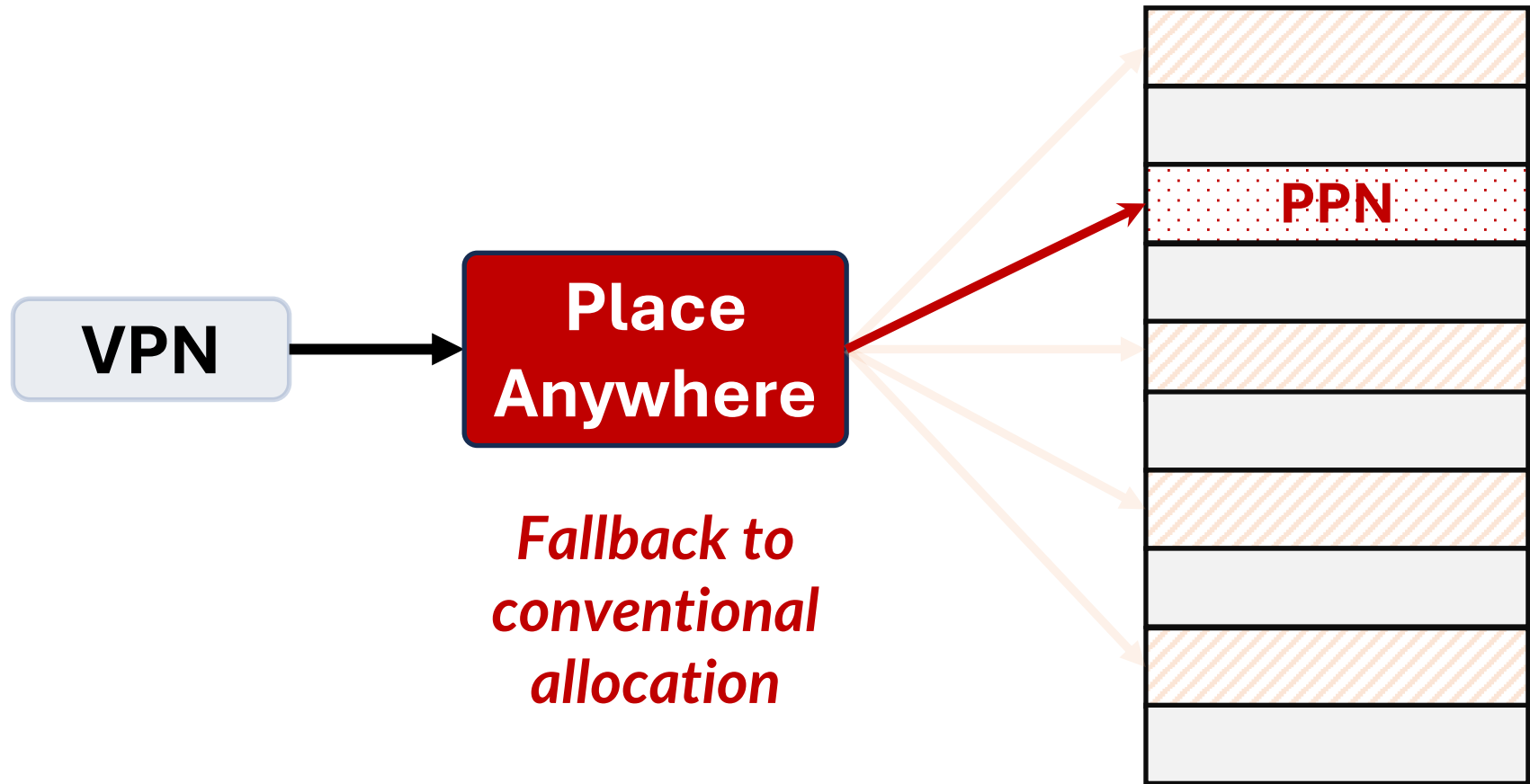
OS: Page Allocation Algorithm



OS: Page Allocation Algorithm



OS: Page Allocation Algorithm



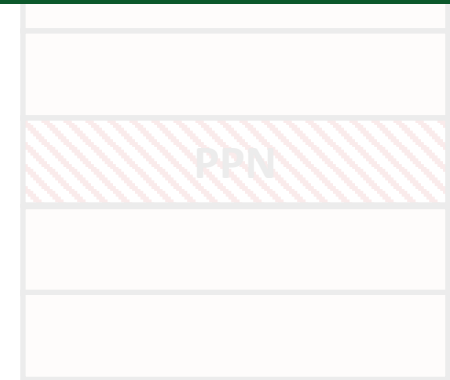
OS: Page Allocation Algorithm

Page is not free

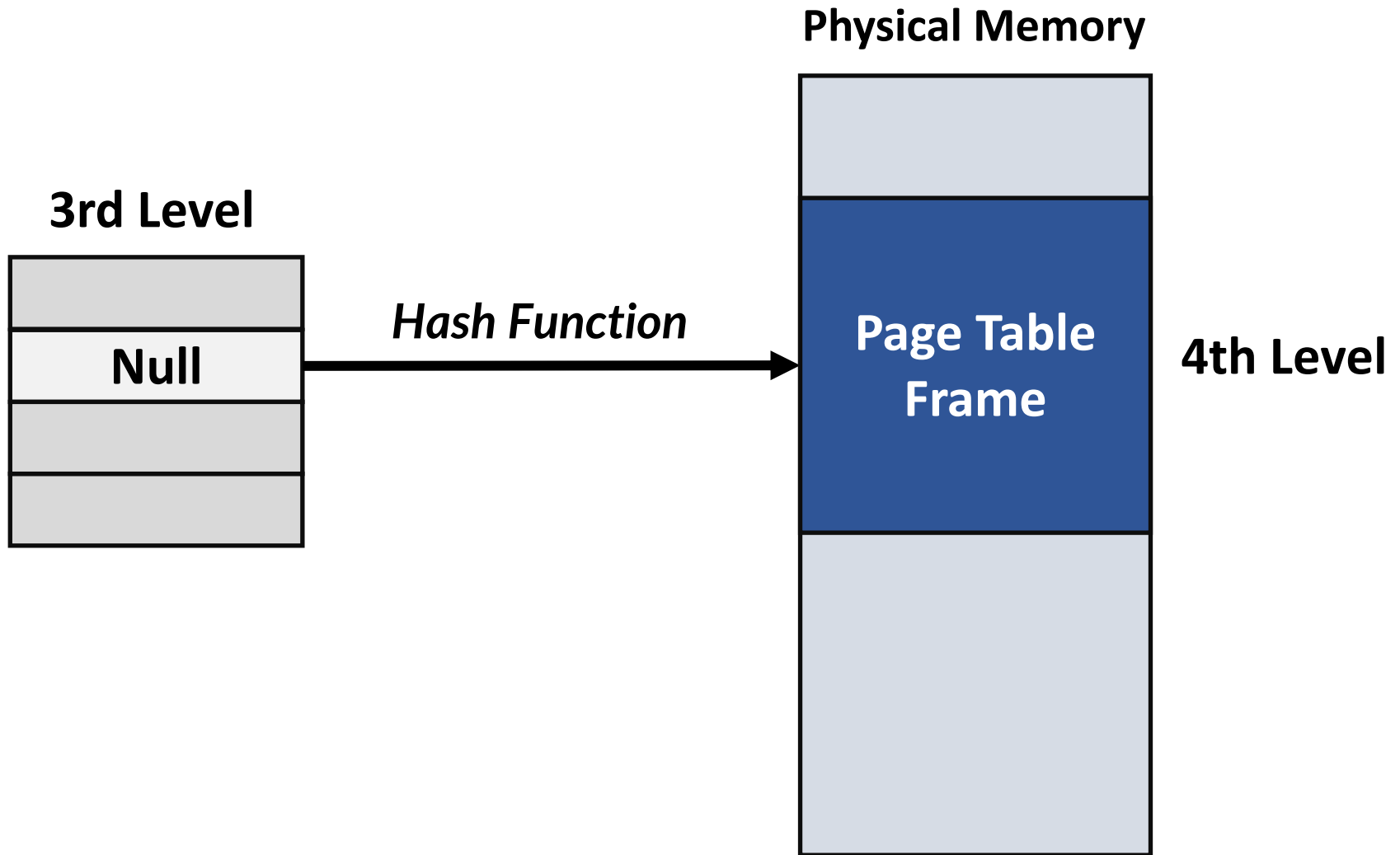


Can we allocate the page table in the same way to enable faster pointer-chasing?

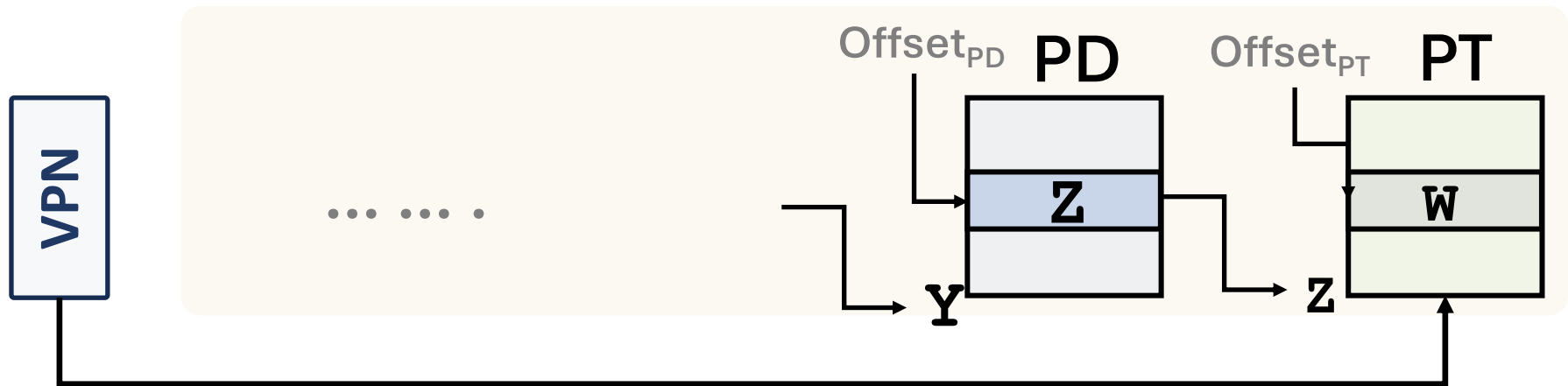
Fallback to conventional fault mechanism before N+1



OS: Allocating the Page Table



OS: Allocating the Page Table



Enables overlapping the access to 3rd and 4th level of the walk

Probability of Successful Allocation

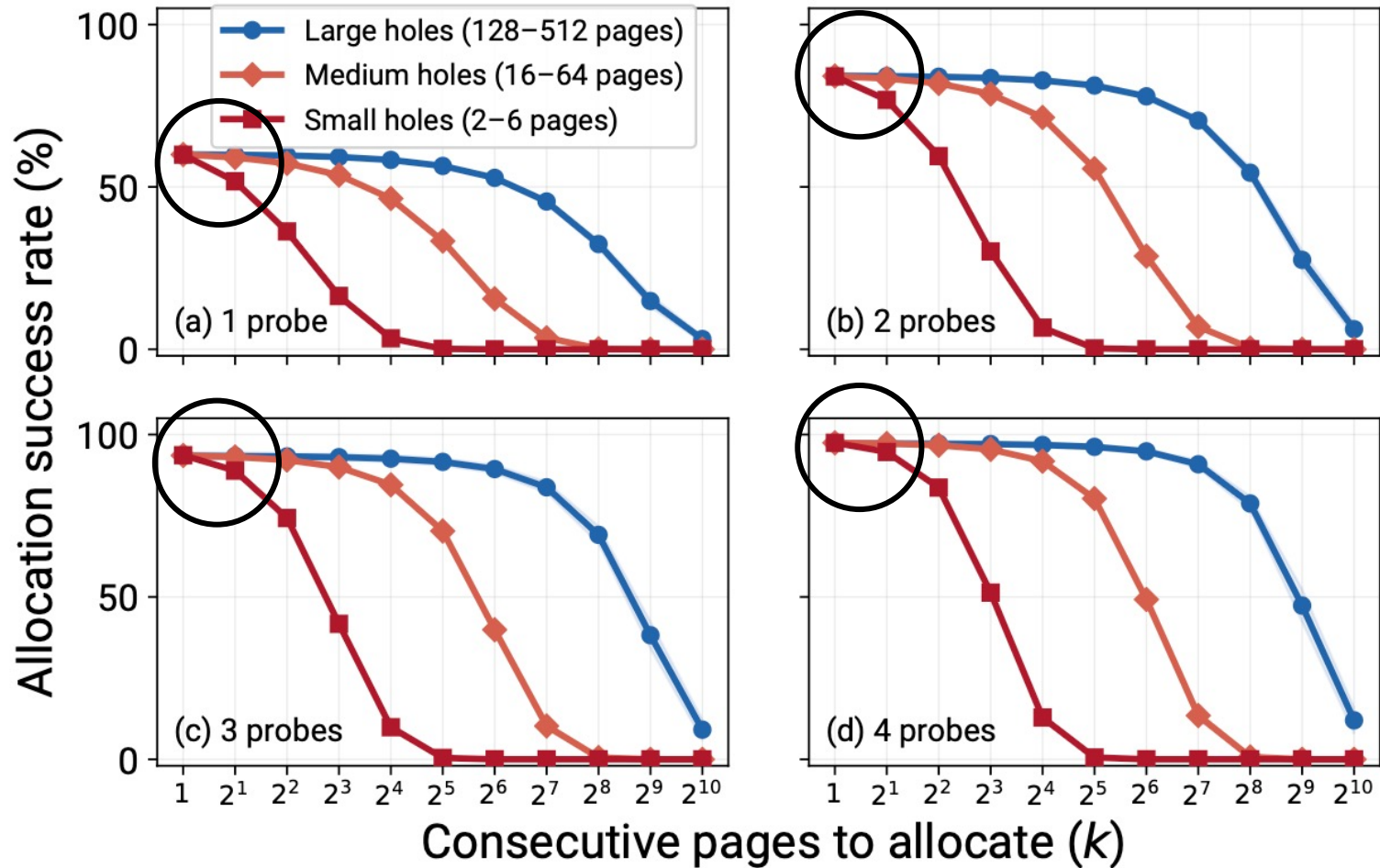
Setup: each VPN allocation attempt = a Bernoulli trial

- Trial outcome: *success* (slot is free) with probability p , *failure* (collision) with probability $1 - p$
- Per-slot occupancy: $p = L / N$, where L = active VPNs, N = total available slots
- Trials are independent across the k hash probes (speculation degree)

Probability of successful allocation

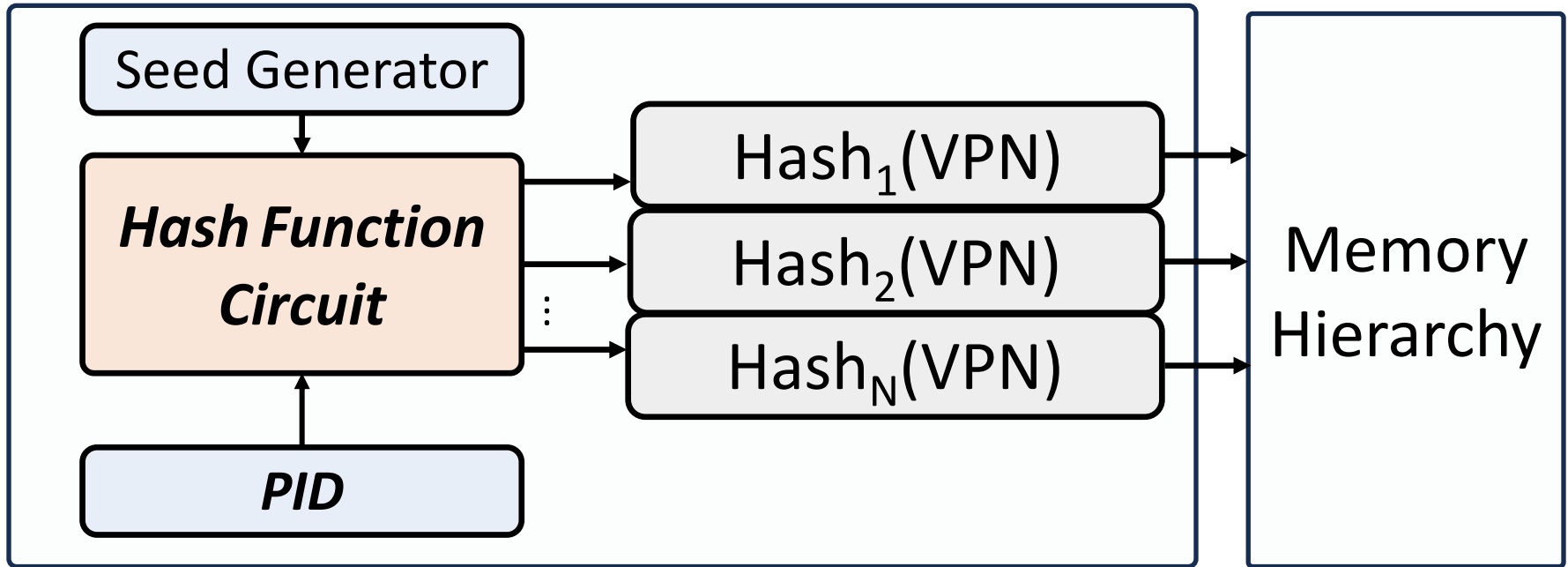
- All k probes fail (allocation rejected): $P(\text{fail}) = p^k$
- At least one probe succeeds: $P(\text{success}) = 1 - p^k = 1 - (L / N)^k$
- Implication: increasing k drives $P(\text{success}) \rightarrow 1$ exponentially, at the cost of bandwidth contention

Probability of Successful Allocation



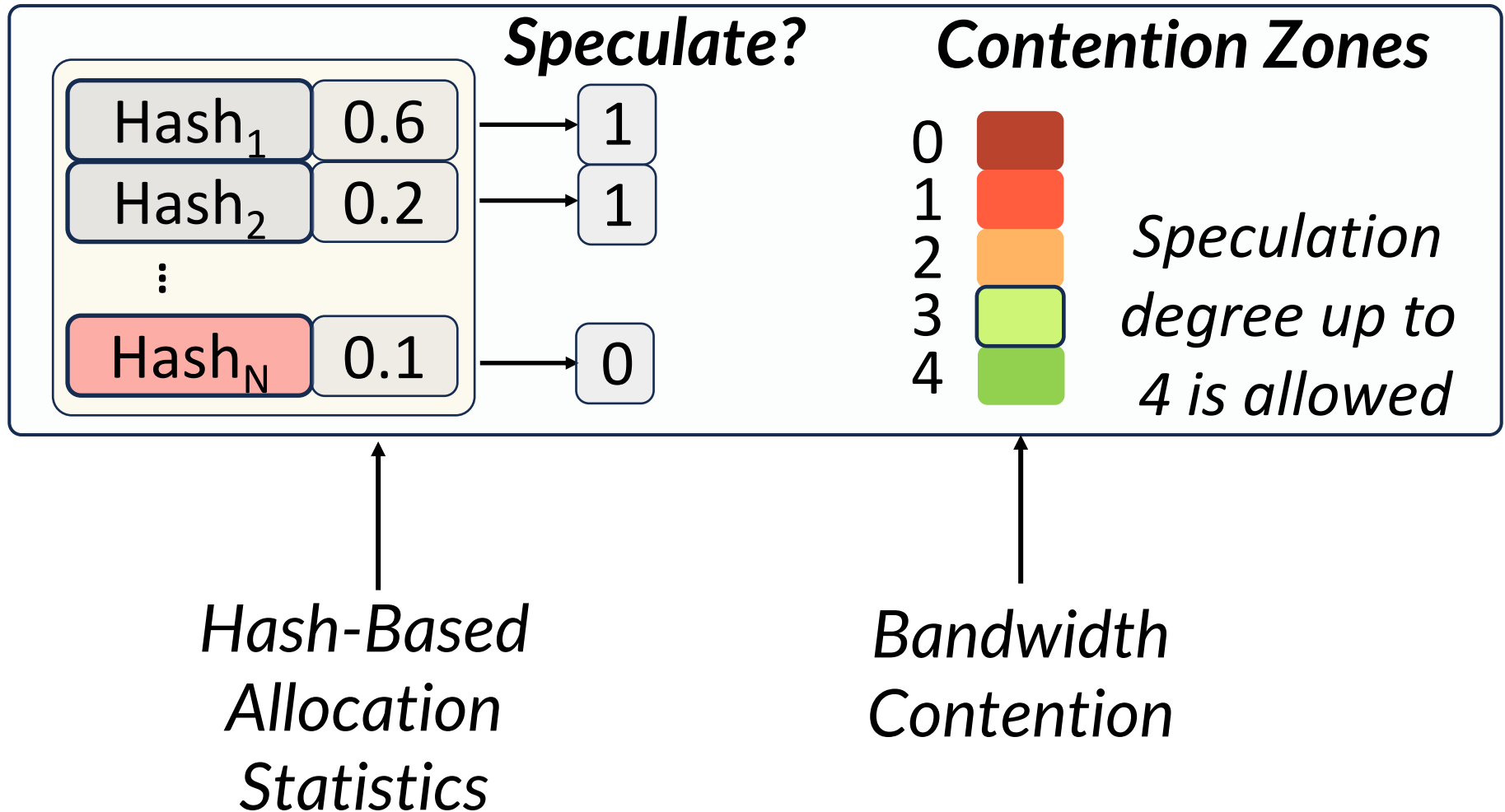
Revelator: Hardware Side

Speculation Engine



Bandwidth consumption is going to explode if N is large

Speculation Degree Filter



Revelator with Large Pages

- ① *Try tiered hash-based allocation for 2MB pages*
- ② *Try 2MB allocation via THP*
- ③ *Try tiered hash-based allocation for 4KB page*
- ④ *Fallback to conventional allocation*

Revelator with Large Pages

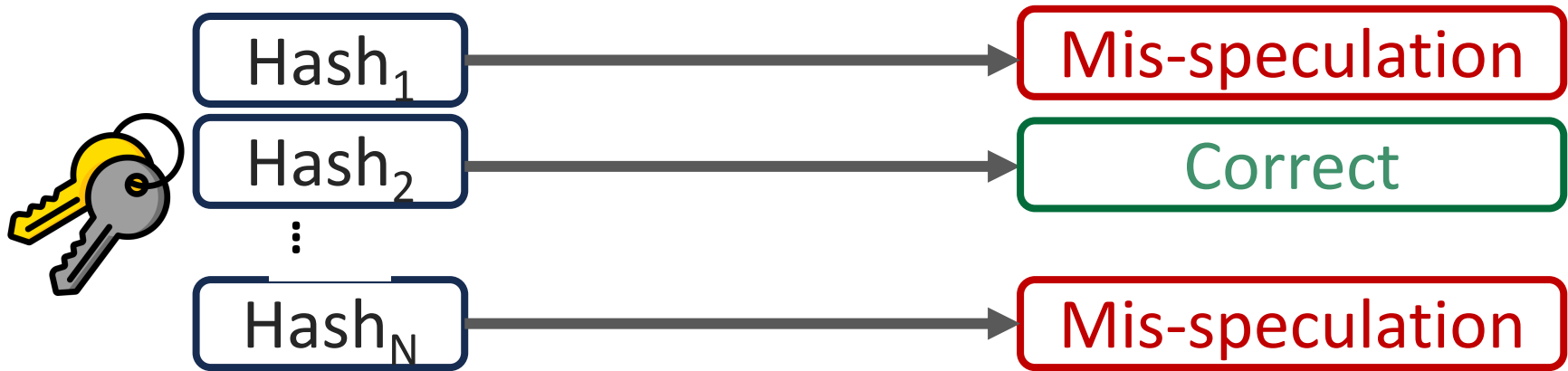
1 *Try tiered hash-based allocation for 2MB pages*

Leverages available contiguity when possible while maintaining the benefits of hash-based allocation in the absence of contiguity

4 *Fallback to conventional allocation*

Security Analysis

- We do not introduce more vulnerabilities than existing data-dependent prefetchers used in modern processors



Mis-speculated data can not be used by the CPU as an index to another data structure -> no new Meltdown/Spectre variants

**However, mis-speculated data remain in the caches
-> an attacker can try to leak data**

Security Analysis

Victim code

The programmer believes the bounds check prevents illegal access.

```
array1 = public_array  
array2 = probe_array  
secret = sensitive_data
```

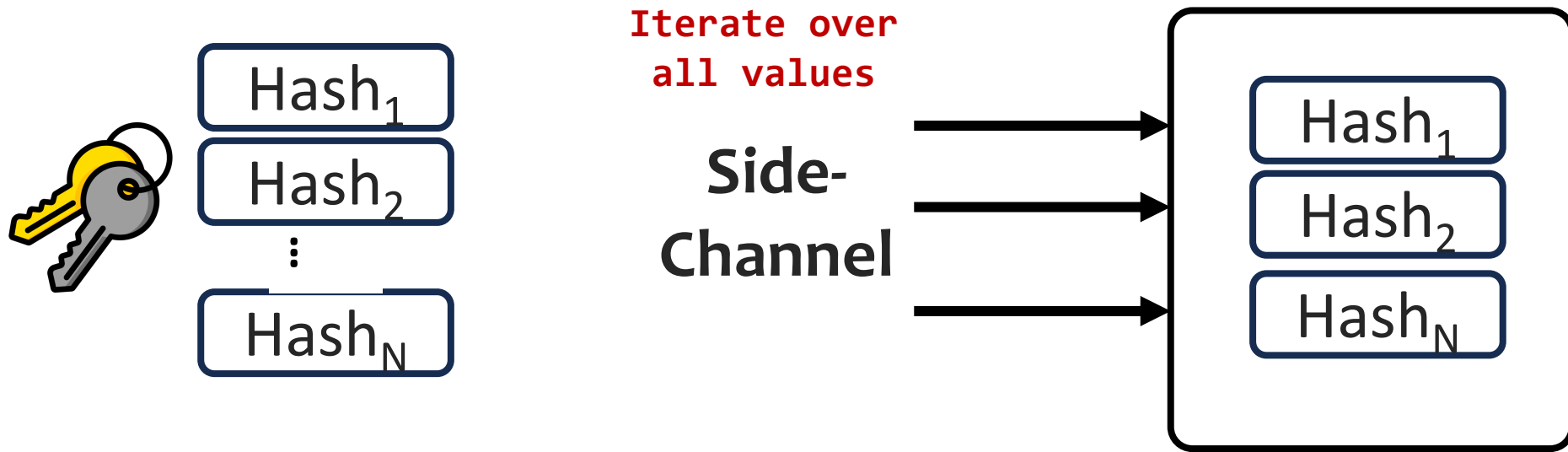
```
function victim(index):  
    if index < length(array1):  
        value = array1[index]
```

```
# Use value to touch a location in another array  
    touch(array2[value])
```

This creates a microarchitectural trace, such as a cache effect.

Security Analysis

touch(array2[value])



Threat Model:

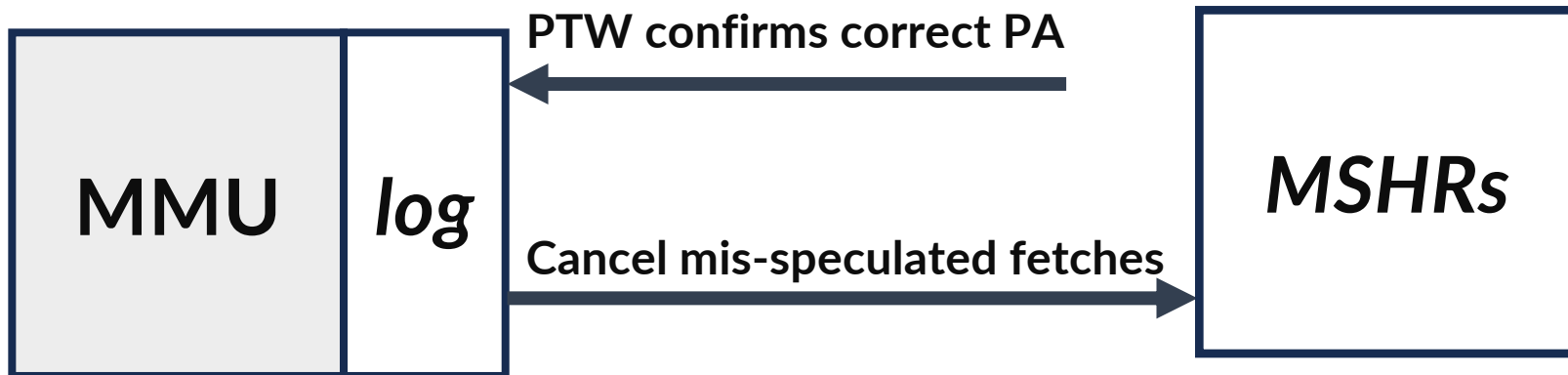
Attacker knows the key of the victim and somehow the defender keeps all cache lines of array2 hot **but not** the TLB ones

Security Analysis

1

Cleanup mis-speculated fetches

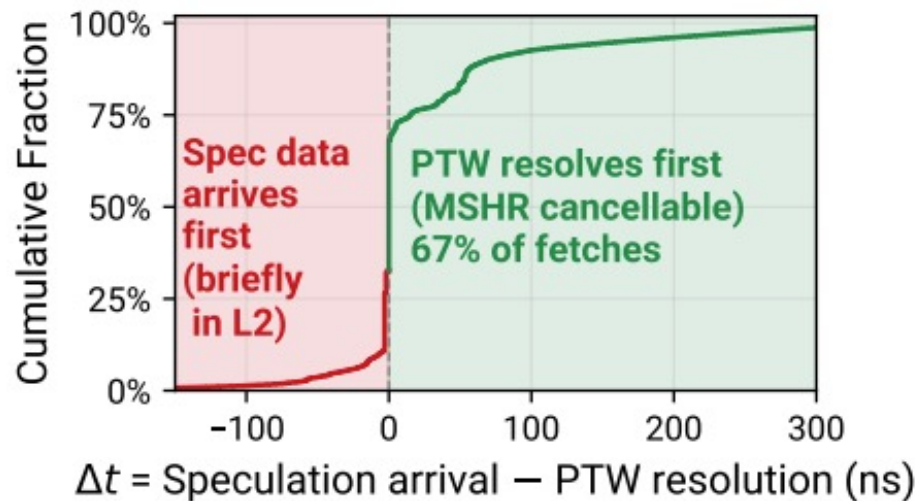
If the PTW completes before all data fetches complete, we can cancel them from the MSHRs so that they are **not installed** in the L2 cache



Security Analysis

1

Cleanup mis-speculated fetches

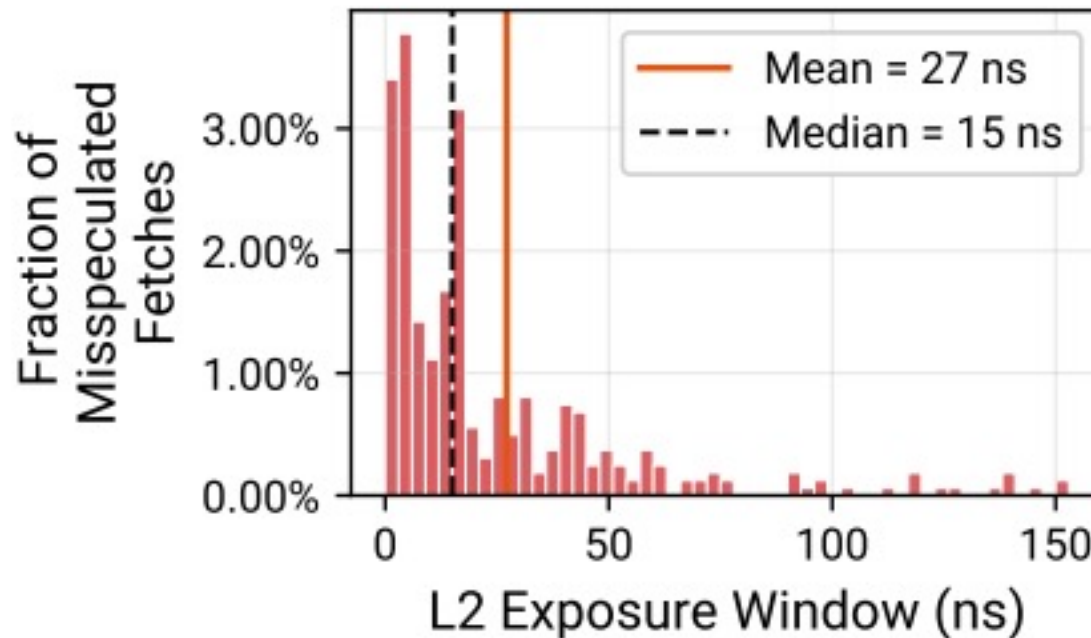


We can cancel 67% of mis-speculated fetches

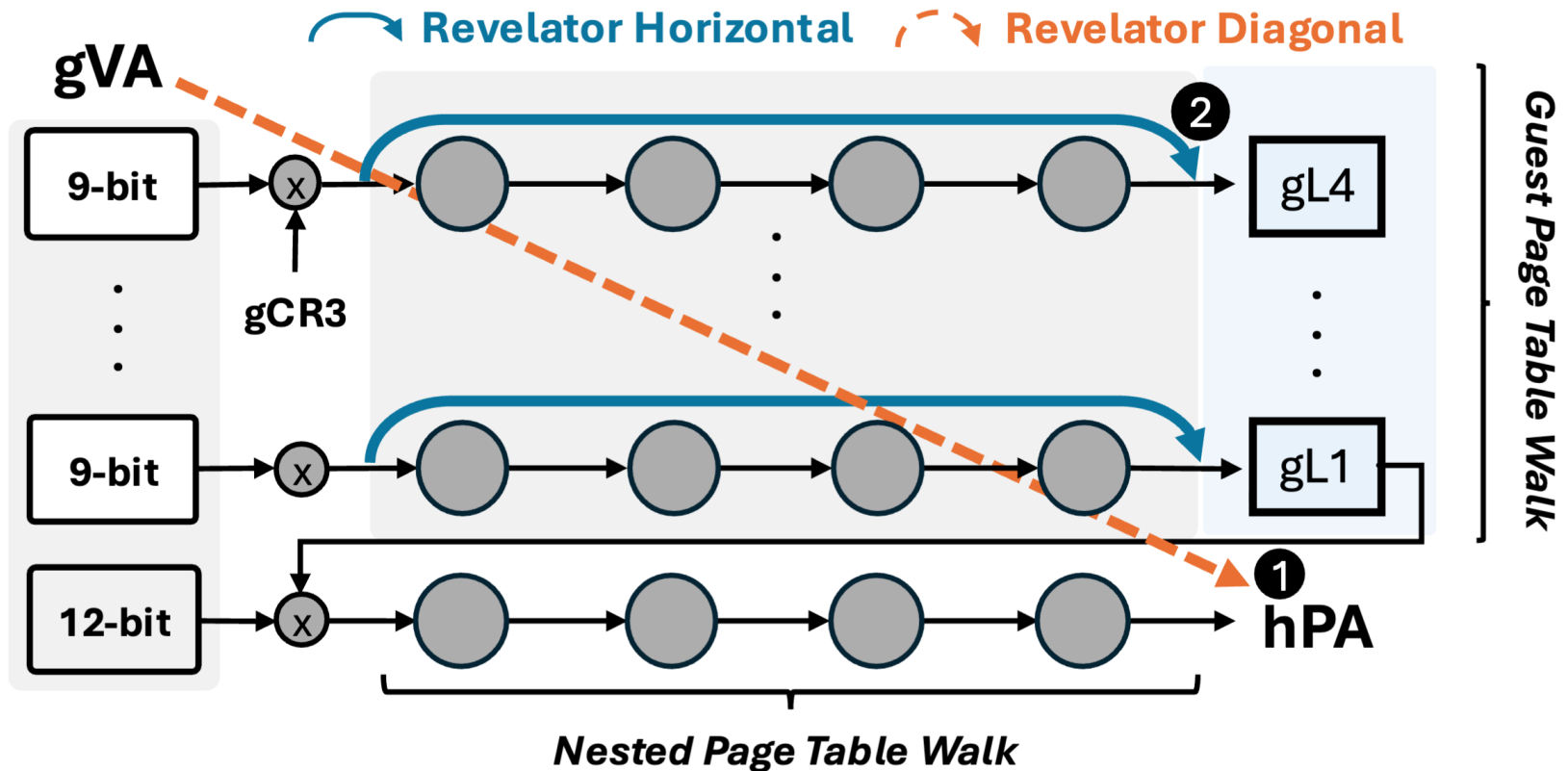
Security Analysis

2

What happens with the rest that remain in the caches until they are invalidated?



Revelator in Virtualized Systems



Revelator in NUMA

Counters

Probability that data will be fetched from Node_i

1

Local Node

59%

Remote Node 1

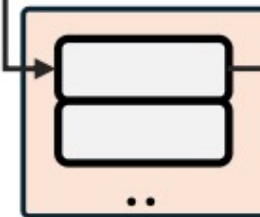
41%

Virtual Address

.. 9 bits 14 bits

3

2

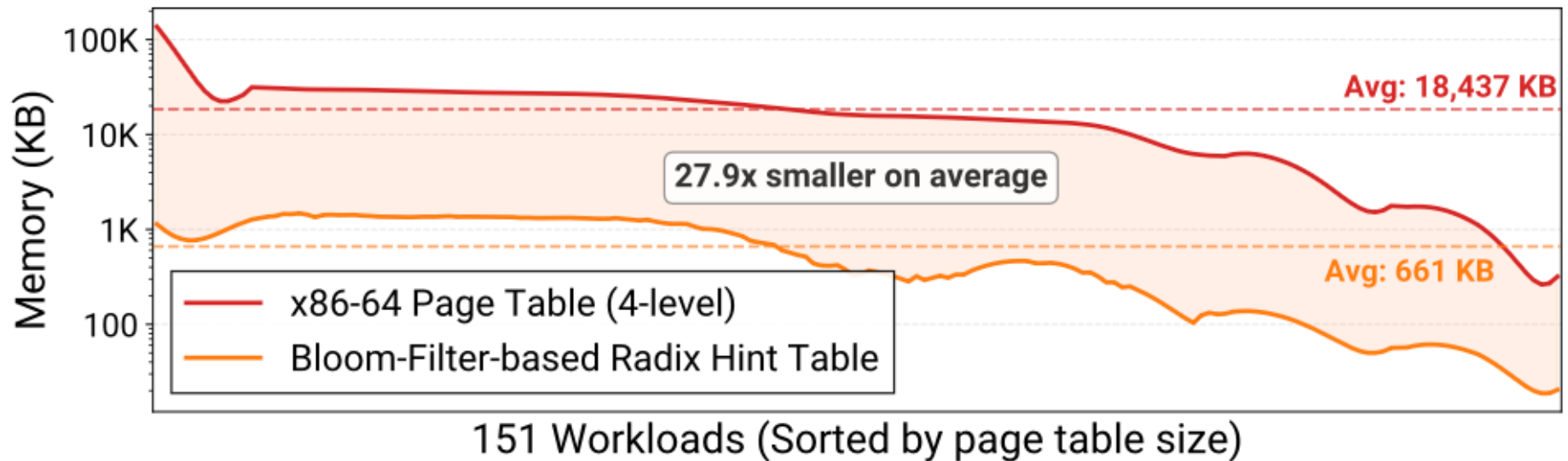


Data is in Remote Node 1

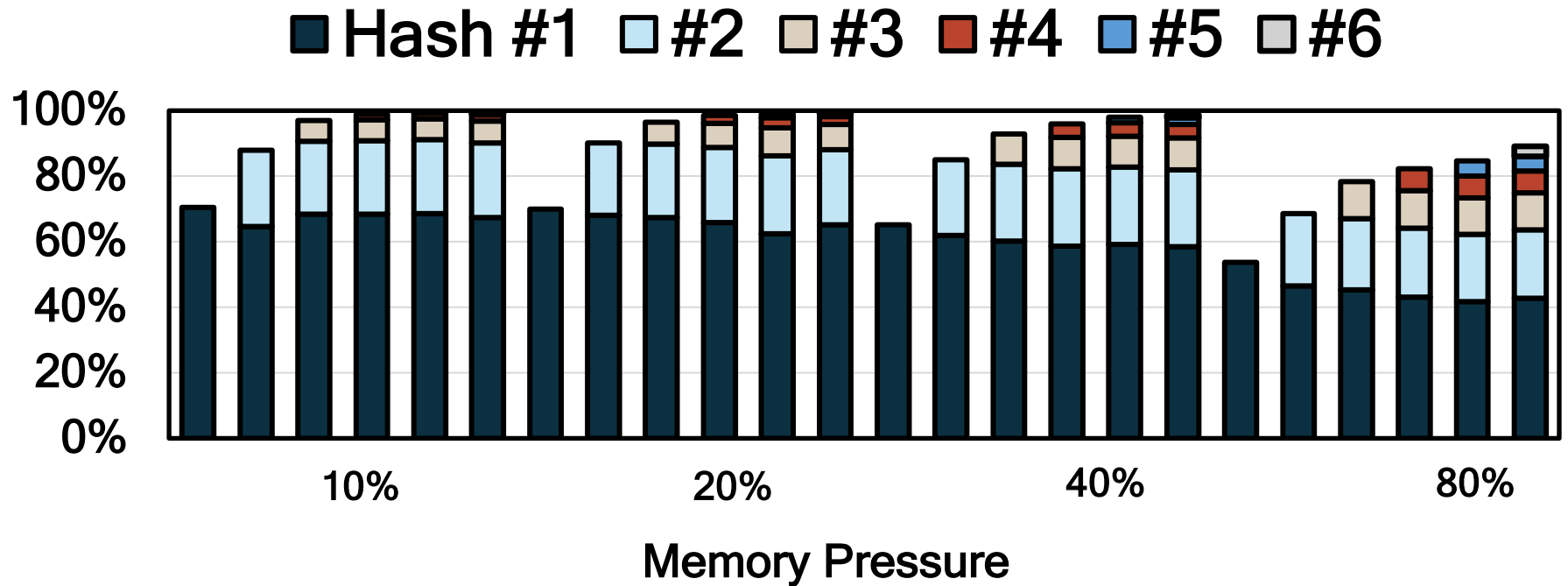
4

High probability data resides in a remote node

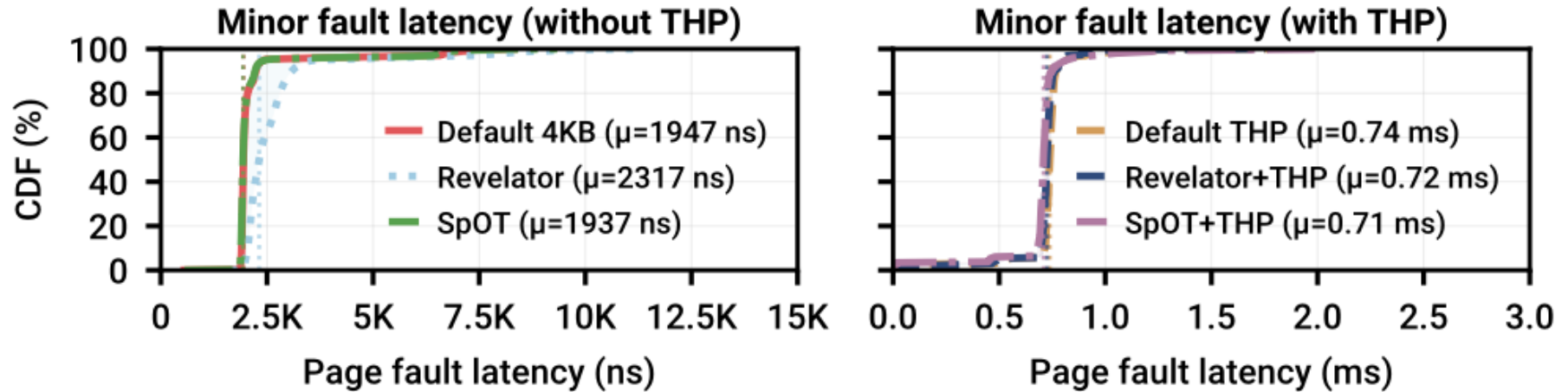
Bloom-Filter-based Radix Table



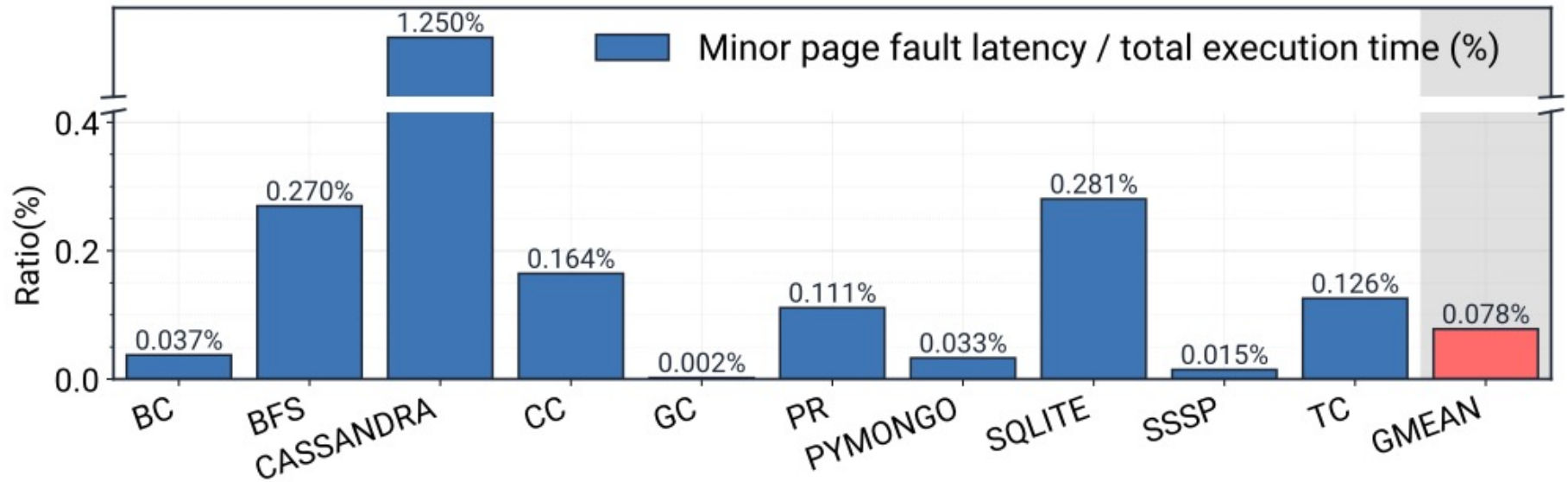
Successful Allocations



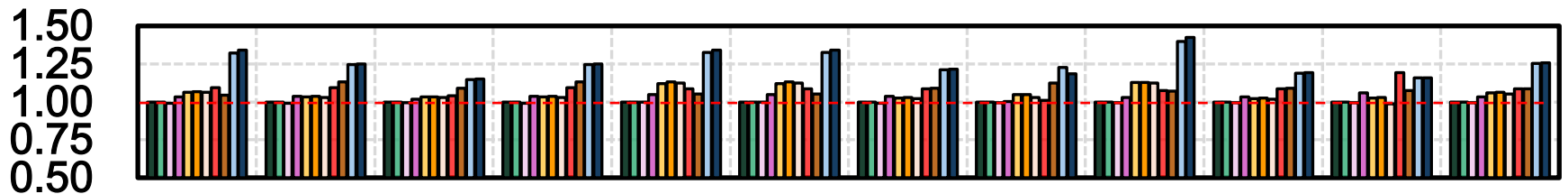
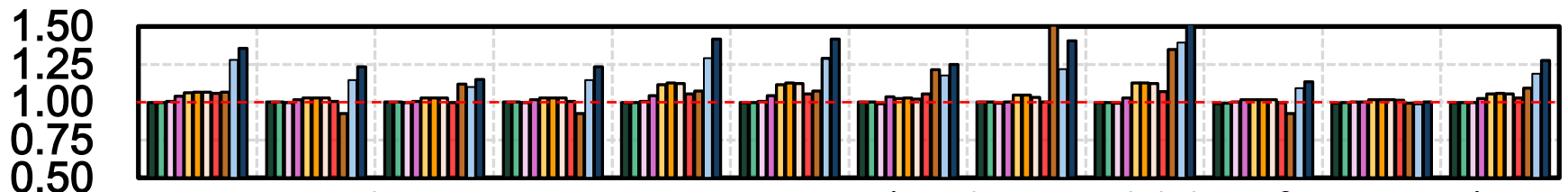
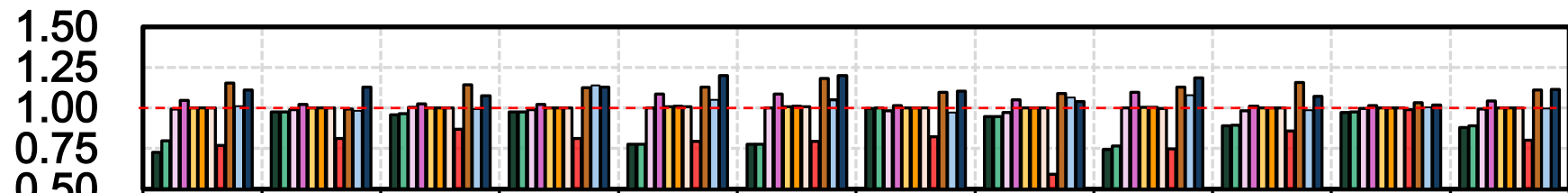
Effect on Allocation Latency



Effect on Allocation Latency

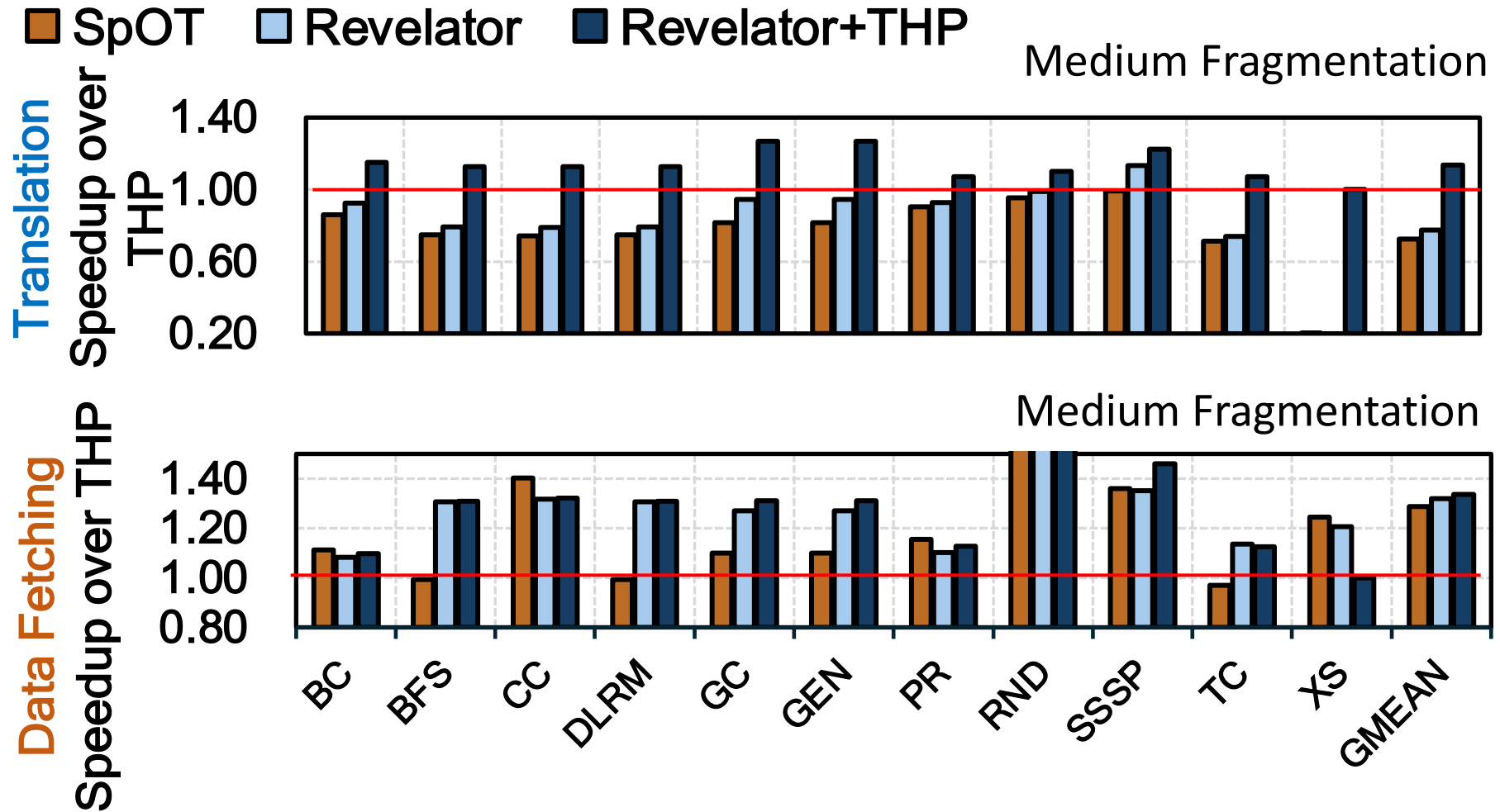


Performance Speedup over THP

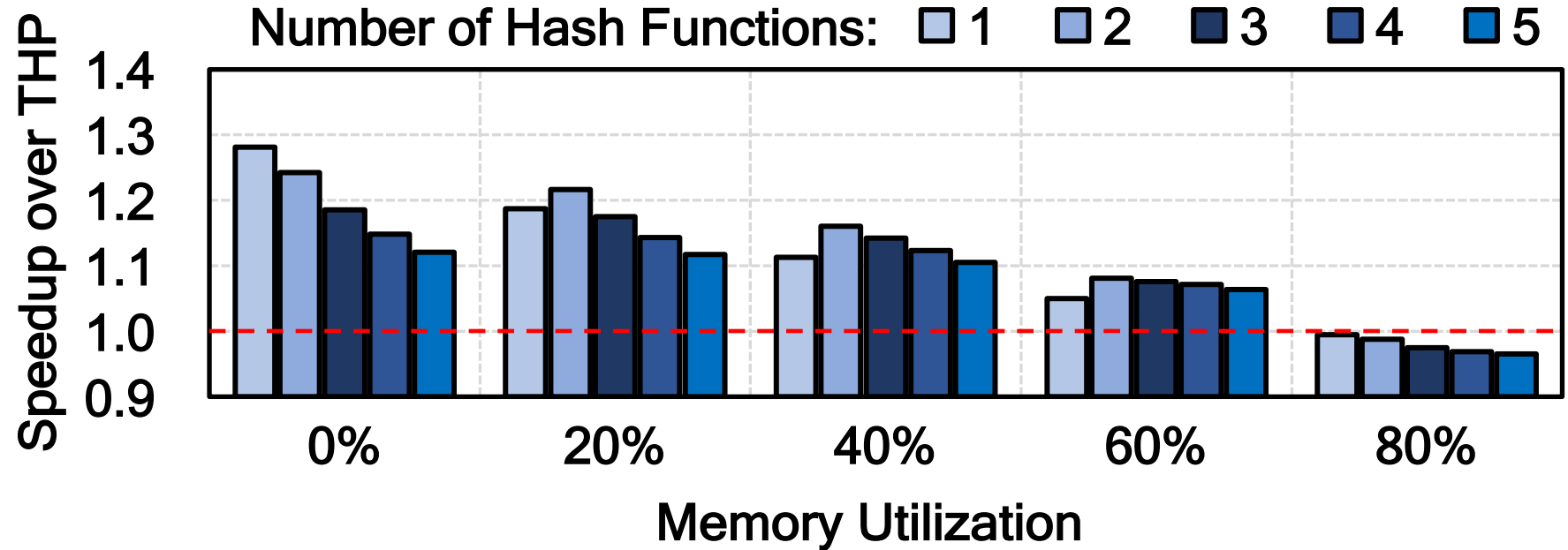


Very Low Fragmentation (Maximum contiguity)

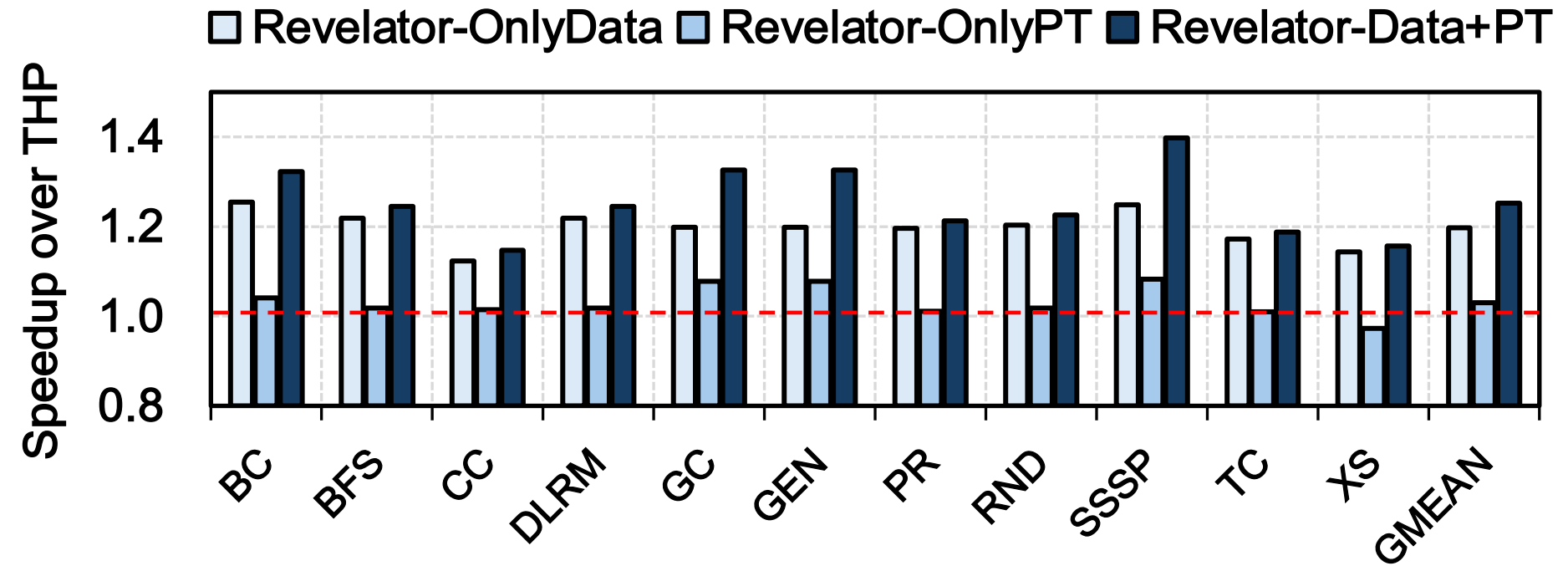
Deep dive into Revelator's benefits



Sweeping Memory Utilization



Ablation Study



Bandwidth Filter

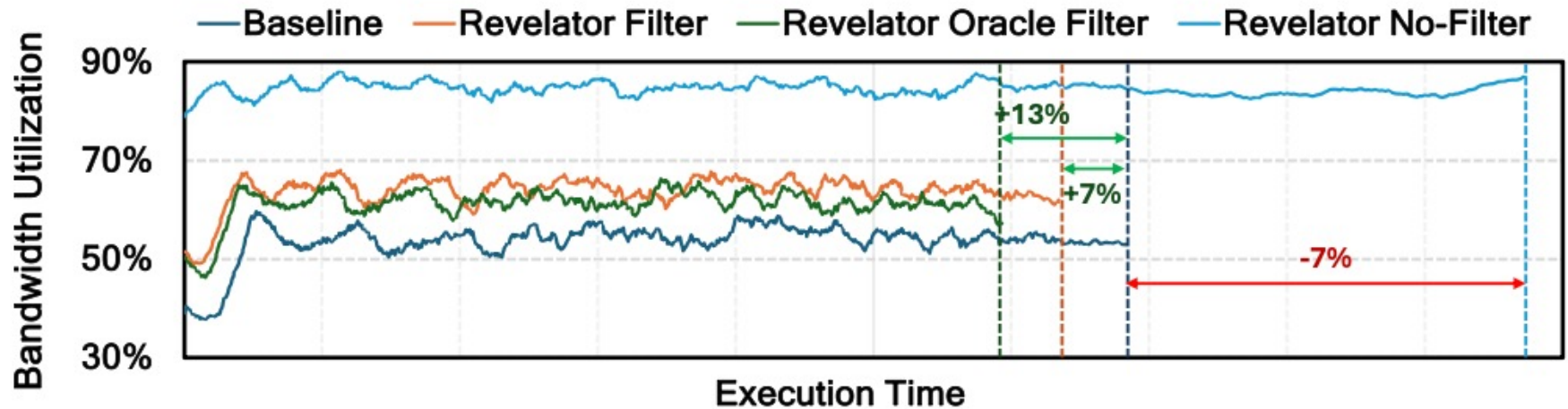


Fig. 21: Impact of Revelator filter in bandwidth utilization and execution time in a memory bandwidth constrained system.

L2 Cache Pollution

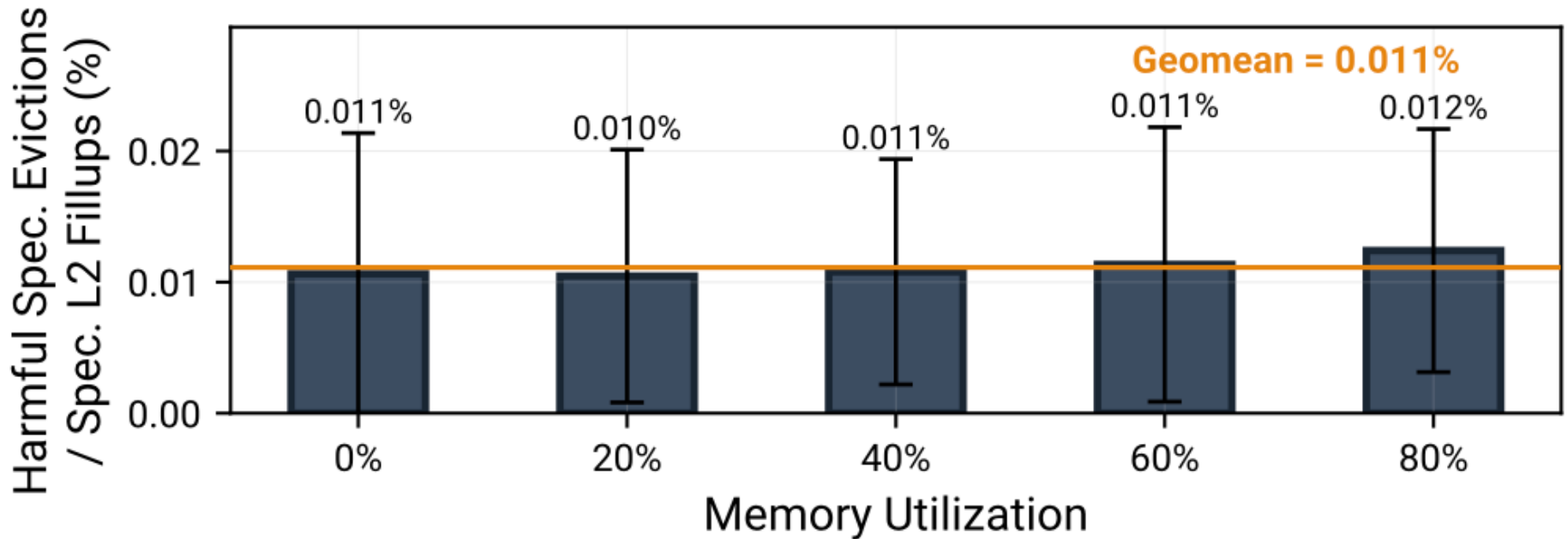


Fig. 22: Percentage of total speculative L2 fillups that lead to harmful evictions across different memory utilization levels.

Energy Reduction

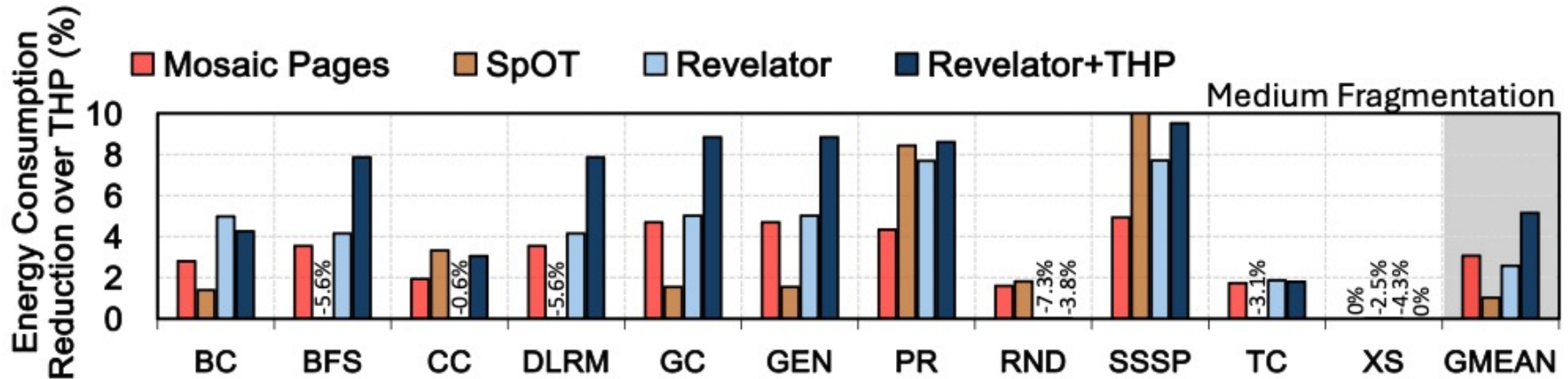


Fig. 23: Energy consumption reduction achieved by Mosaic Pages [23], SpOT [13], Revelator and Revelator+THP over THP in a system with medium fragmentation.

Multicore Environments

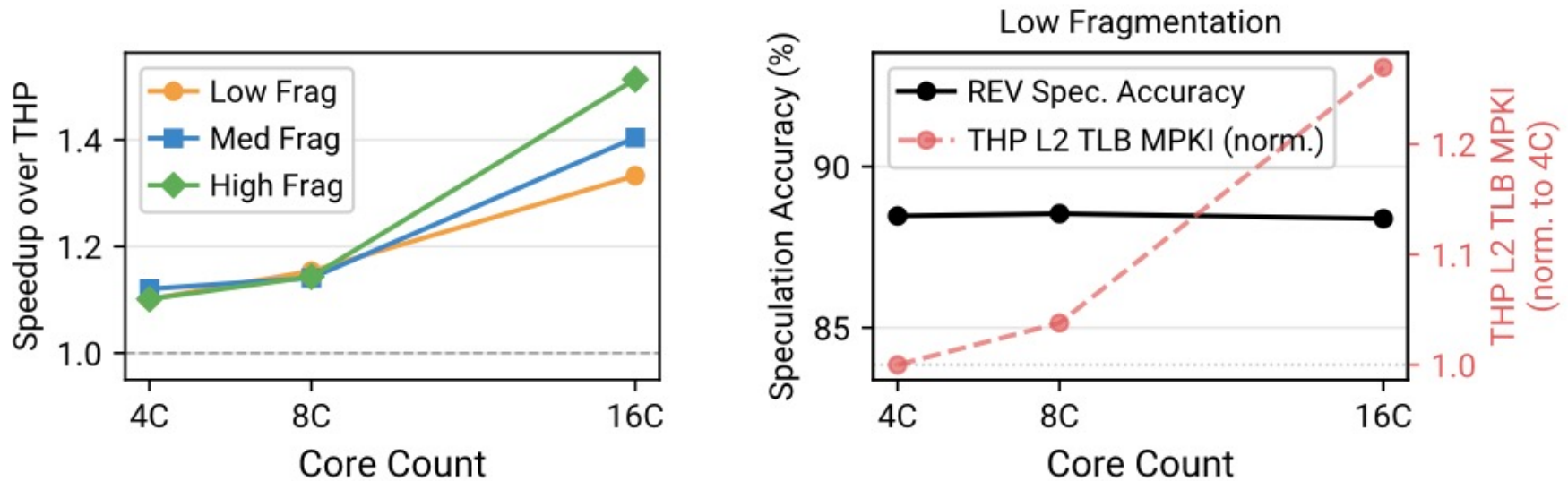


Fig. 25: (Left) Weighted speedup of Revelator over THP under low/med/high fragmentation. and (Right) L2 TLB MPKI increase for THP versus Speculation Accuracy for Revelator under low fragmentation - across 4/8/16-core systems.

Virtualized Environments

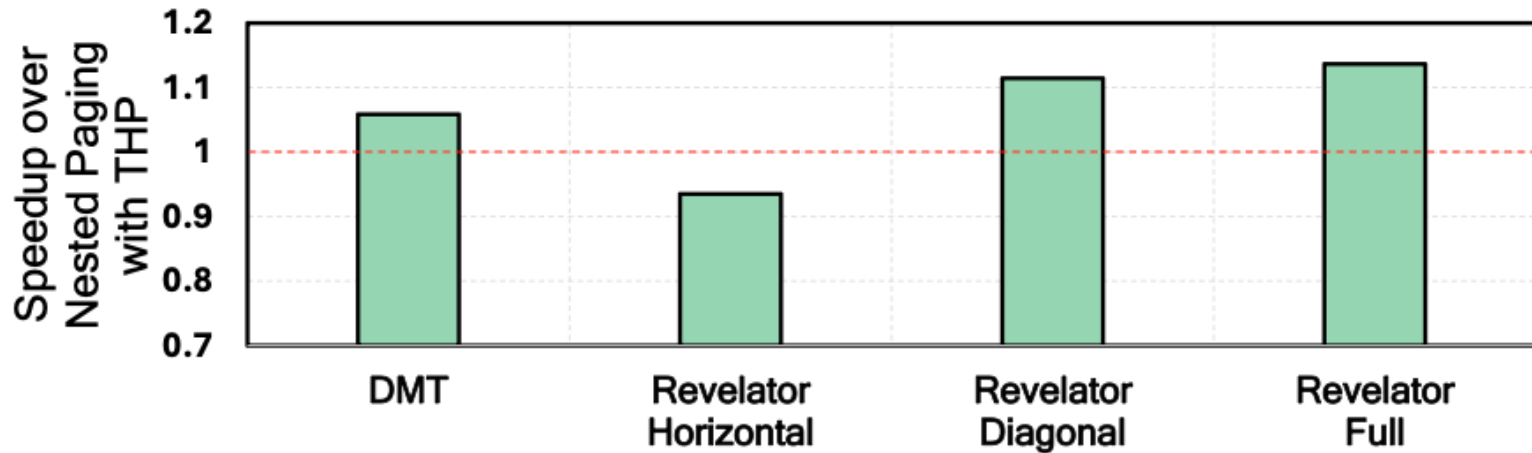
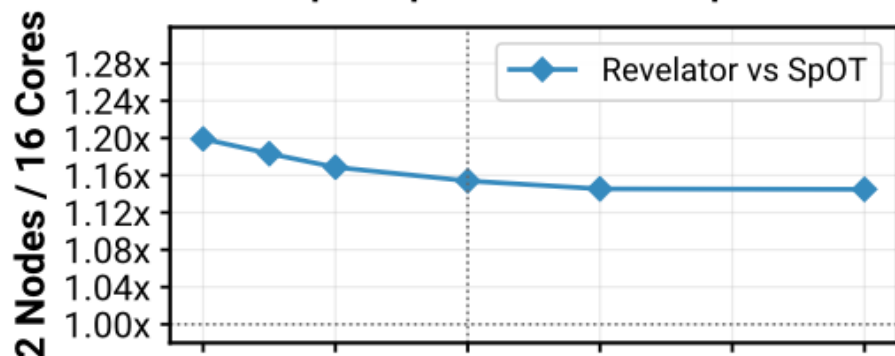


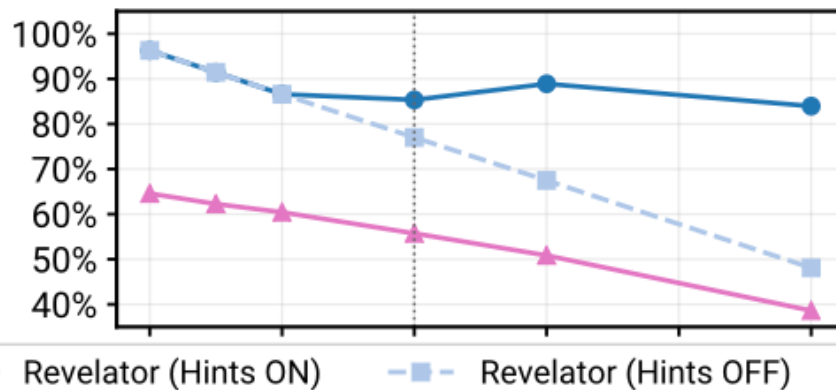
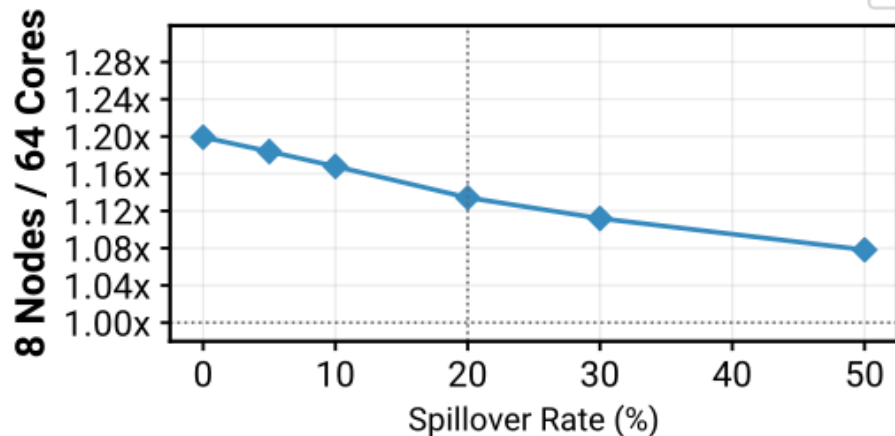
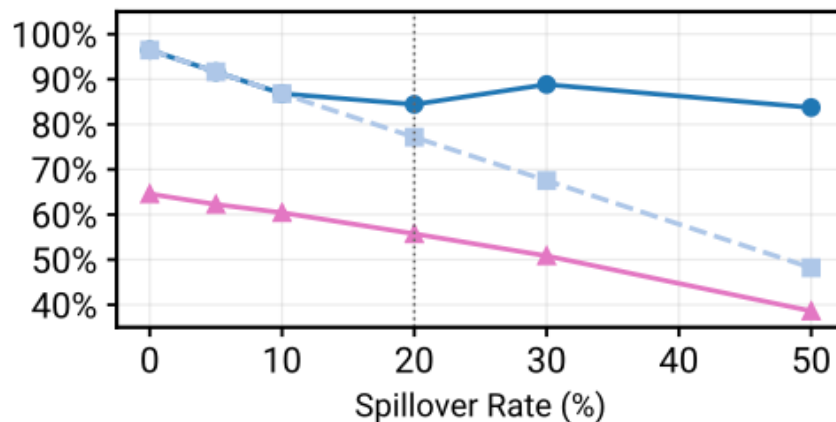
Fig. 24: Performance speedup of Revelator configurations and DMT [22] over Nested Paging with THP.

NUMA

Speedup of Revelator vs SpOT

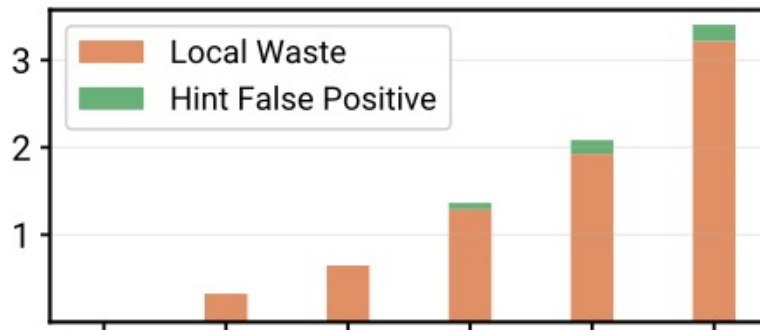


Hidden Page Table Walk Latency

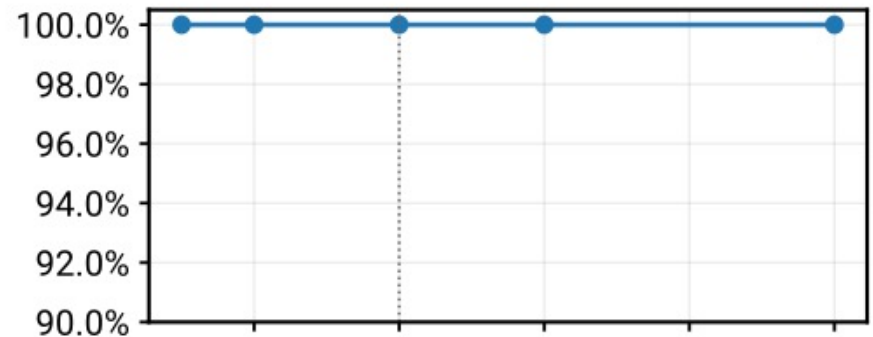
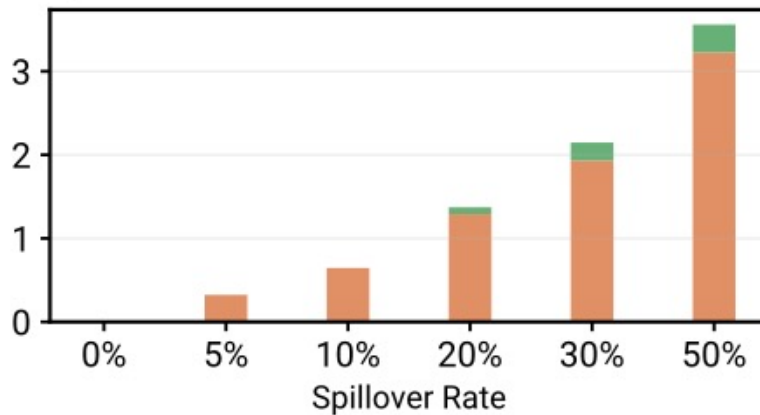
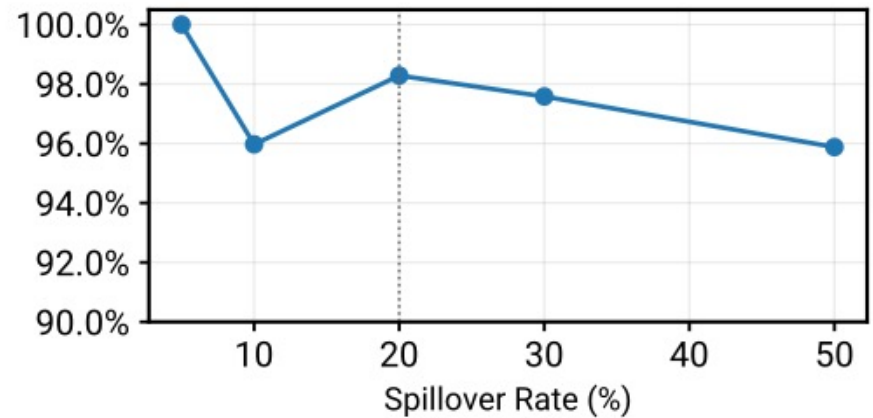


NUMA Details

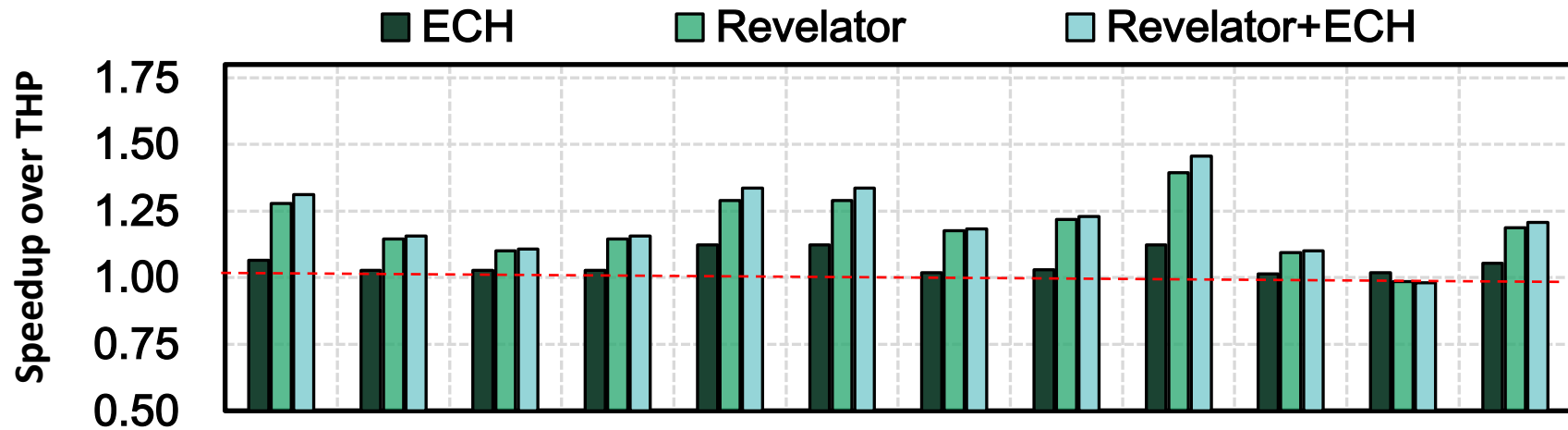
Bandwidth Wasted on Mis-speculation in Revelator
(Aggregate (across cores) waste per kilo memory accesses)



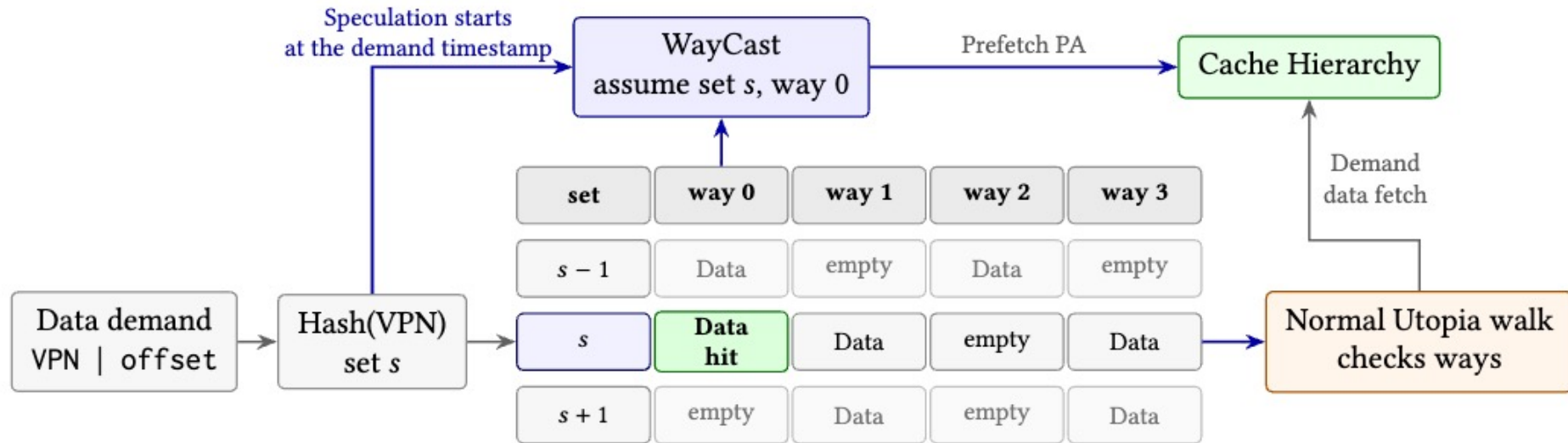
Bloom-Filter-based Radix Hint Accuracy



Revelator + ECHT



WayCast: Utopia vs Revelator



Modularity and Composability

