Scaling CHERI Use-After-Free Protection to Millions of Allocations

CHERI Blossoms Conference, 26 – 27 March 2026

Merve Gülmez[*], Ruben Sturm[†], Hossam ElAtali[§], Håkan Englund[*], Jonathan Woodruff[‖], N. Asokan[§,¶], Thomas Nyman[*]

[*]Ericsson, [†]DistriNet, KU Leuven, [§]University of Waterloo, [‖]University of Cambridge, [¶]KTH Royal Institute of Technology

CHERI avoids indirection

CHERI, *by design*, avoids **table-based lookups**

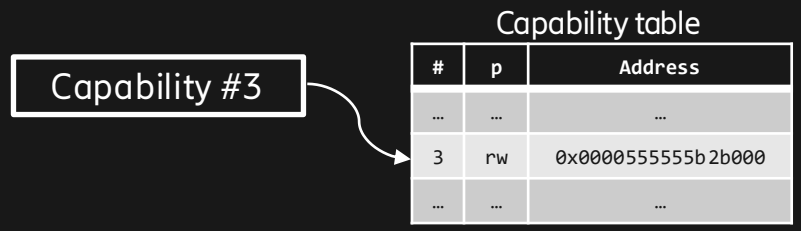
CHERI avoids indirection

CHERI, *by design*, avoids **table-based lookups**

Capability #3

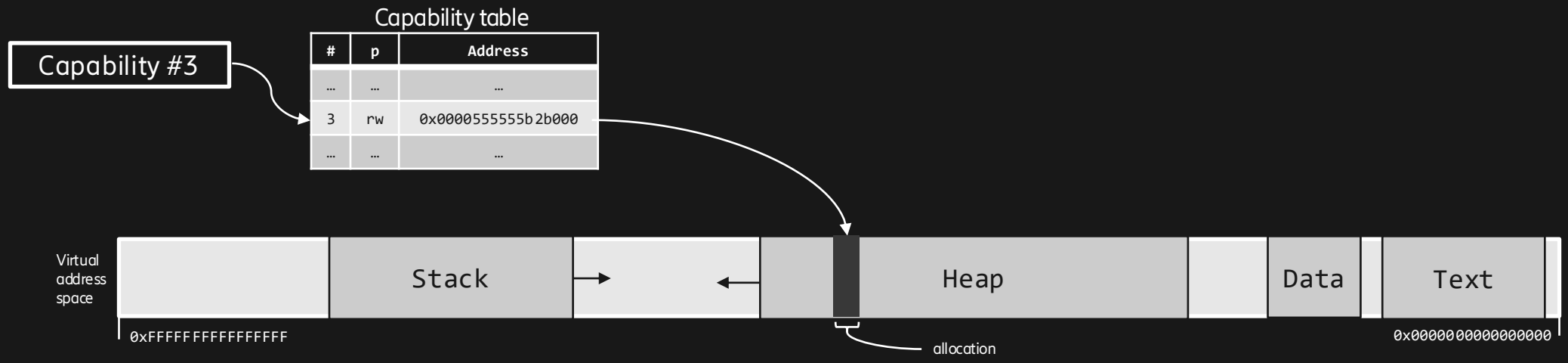
CHERI avoids indirection

CHERI, *by design*, avoids **table-based lookups**



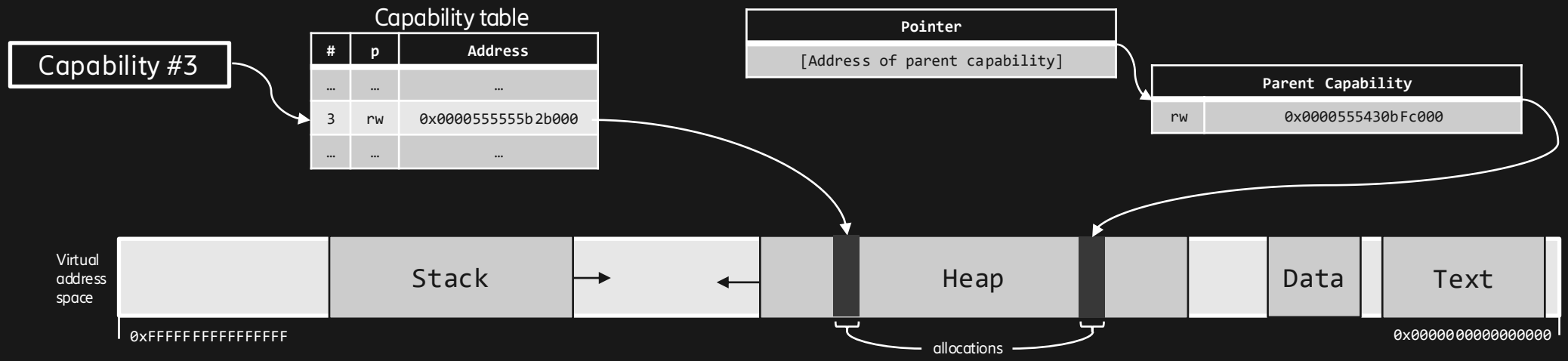
CHERI avoids indirection

CHERI, *by design*, avoids **table-based lookups**



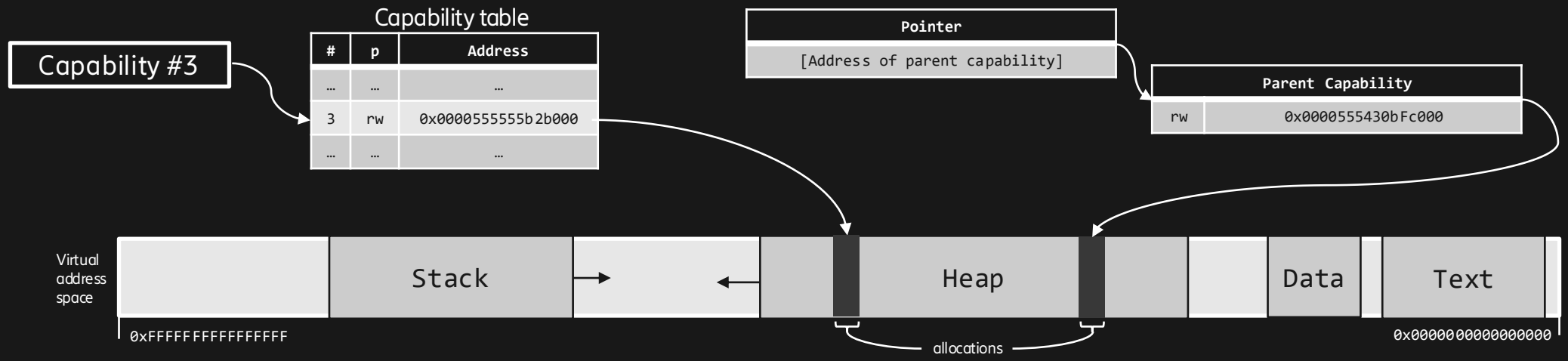
CHERI avoids indirection

CHERI, *by design*, avoids **table-based lookups** and **indirection on pointer operations**



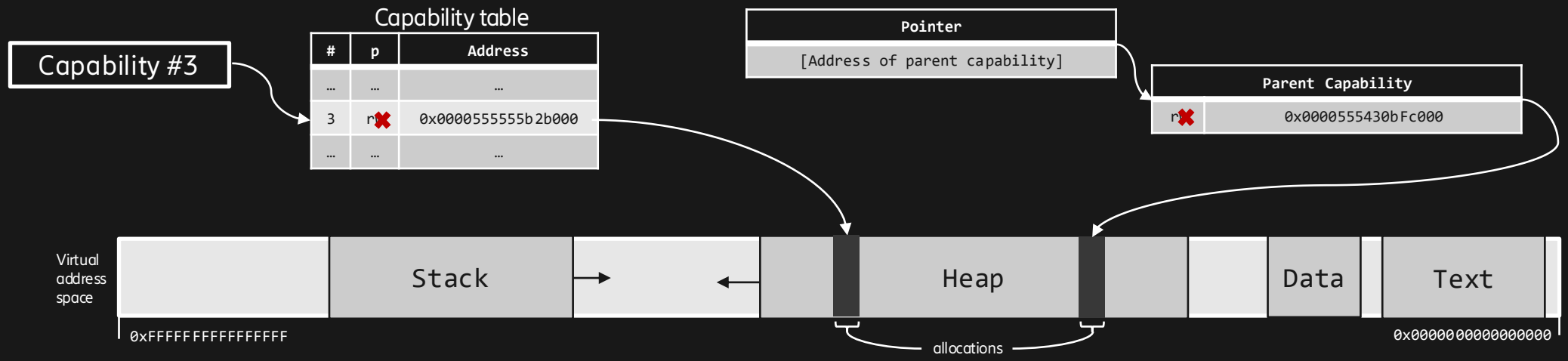
Benefits of indirection

Indirection, however, allows to alter...



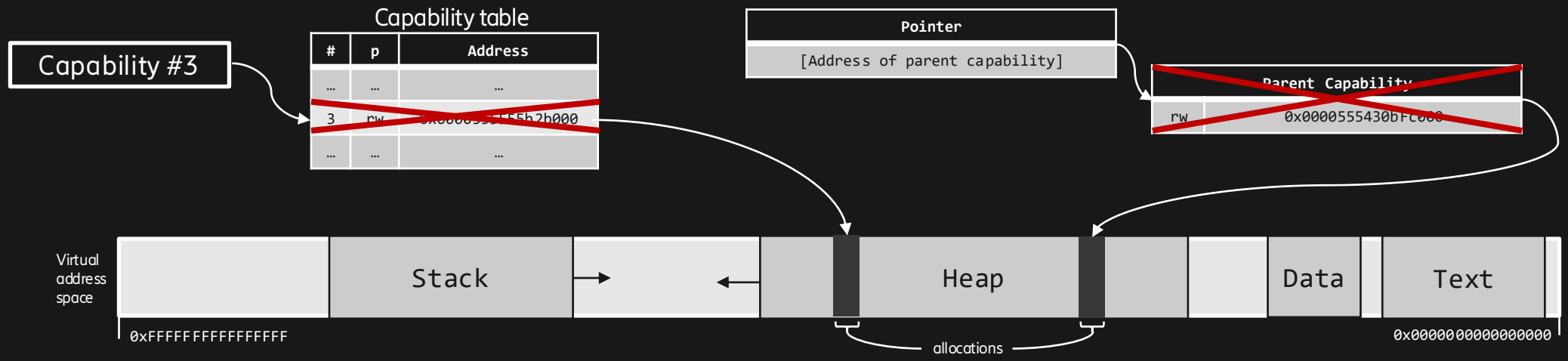
Benefits of indirection

Indirection, however, allows to alter...



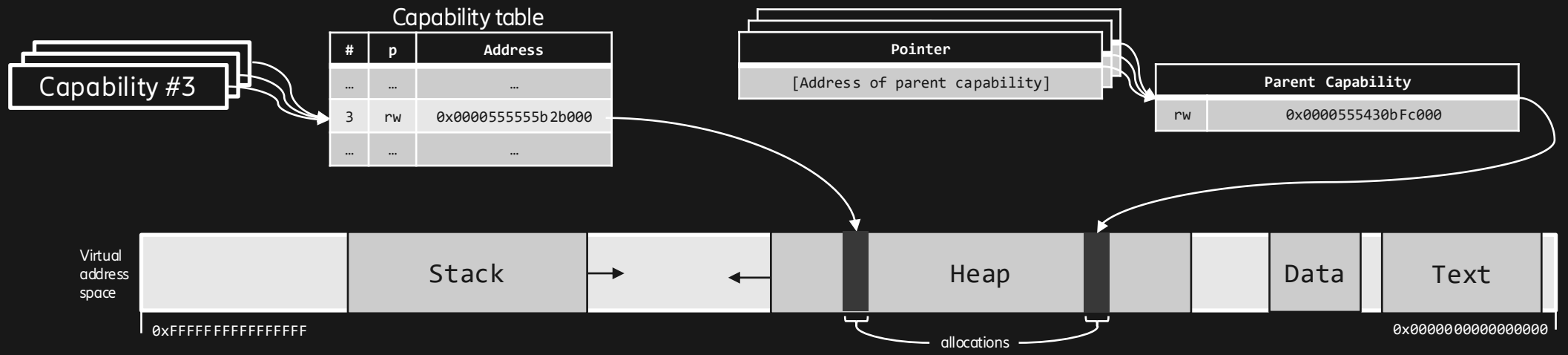
Benefits of indirection

Indirection, however, allows to alter or revoke capabilities...



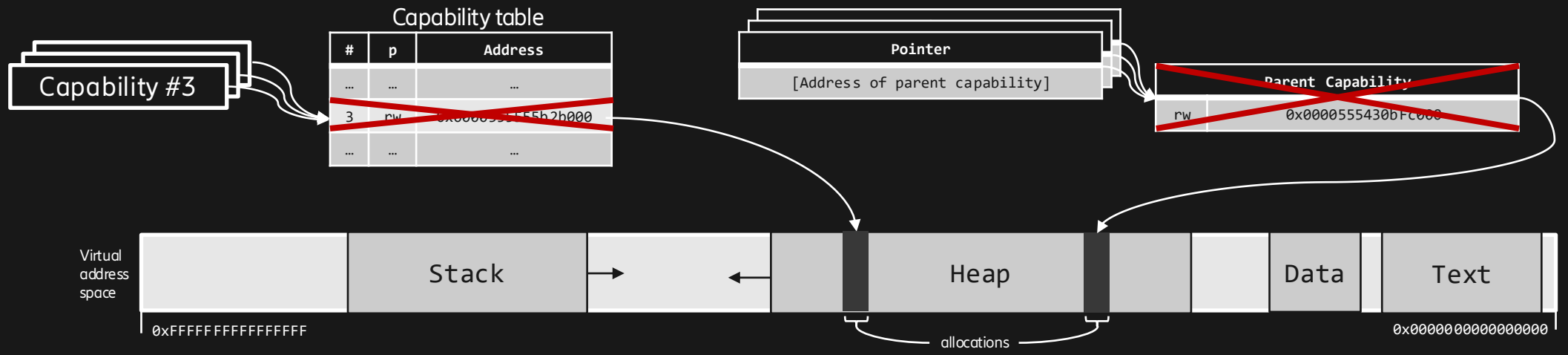
Benefits of indirection

Indirection, however, allows to alter or revoke capabilities **in bulk**



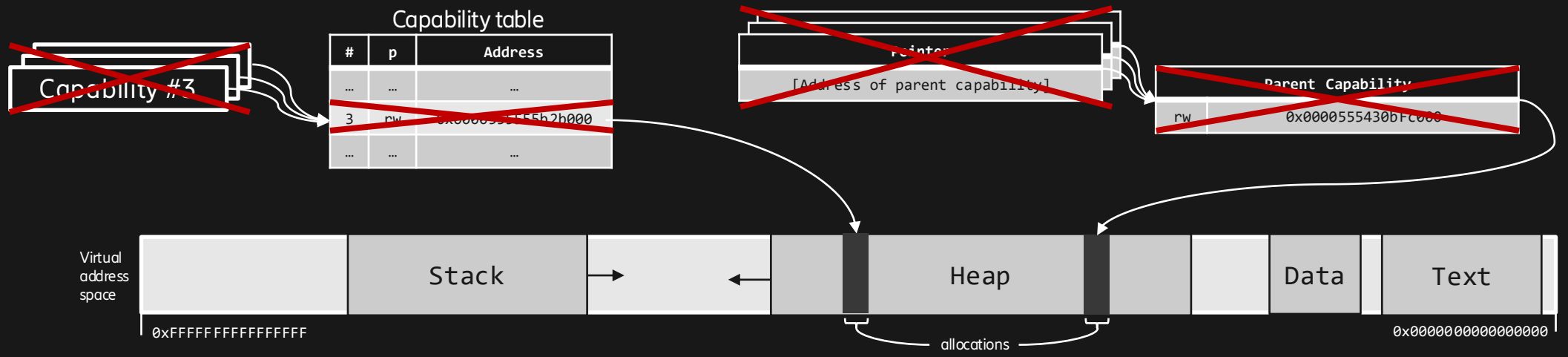
Benefits of indirection

Indirection, however, allows to alter or revoke capabilities **in bulk**



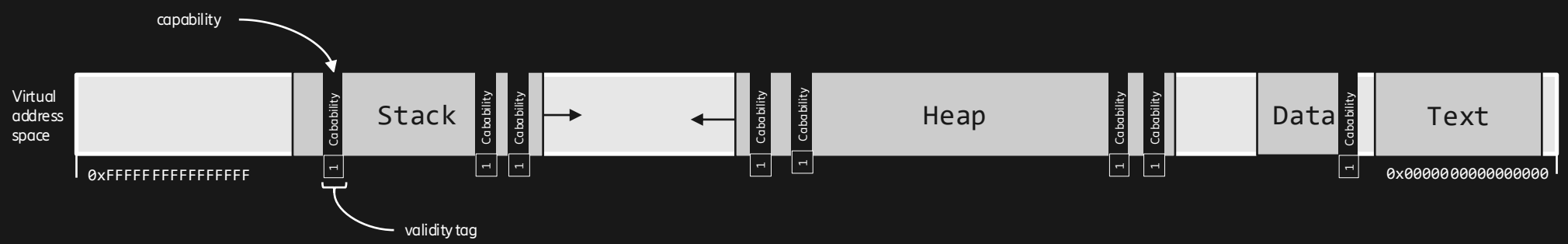
Benefits of indirection

Indirection, however, allows to alter or revoke capabilities **in bulk**



Tradeoffs of avoiding indirection: revocation sweep

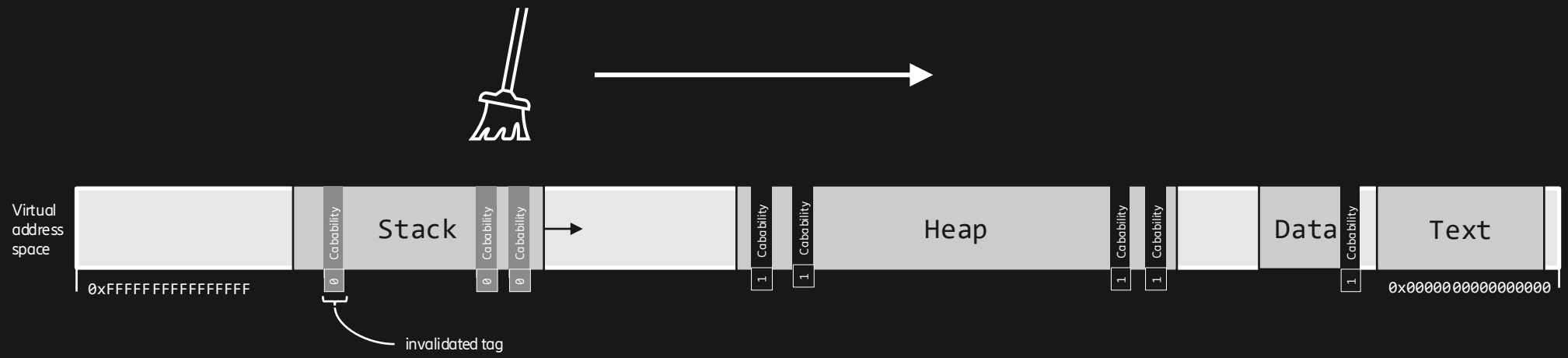
In CHERI, revocation requires knowing a capability's (and its validity tag's) location in memory



Tradeoffs of avoiding indirection: revocation sweep

In CHERI, revocation requires knowing a capability's (and its validity tag's) location in memory

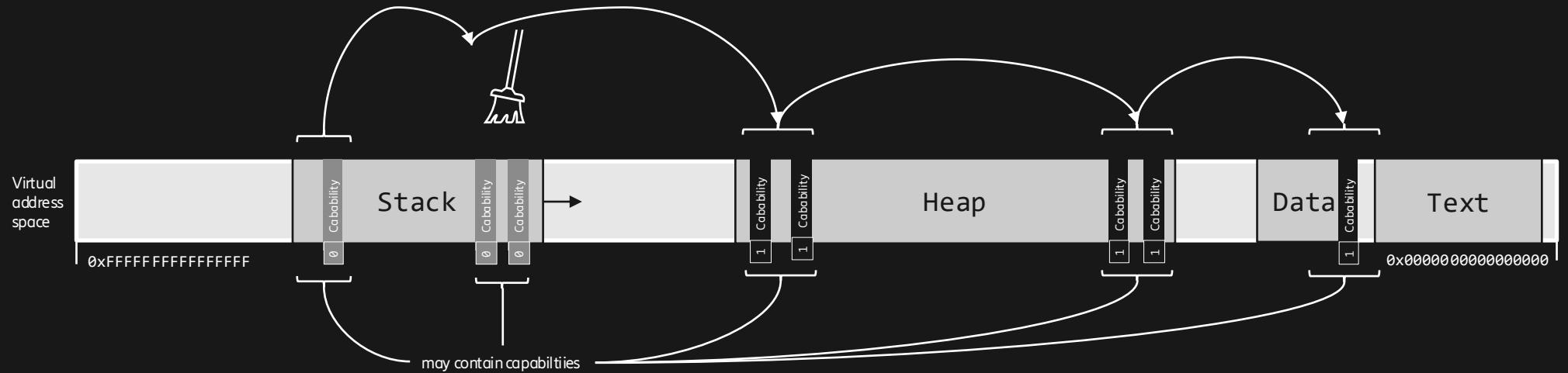
- Invalidating *sets of capabilities* relies on *sweeping revocation,* e.g., for enforcing temporal safety



Tradeoffs of avoiding indirection: revocation sweep

In CHERI, revocation requires knowing a capability's (and its validity tag's) location in memory

- Invalidating *sets of capabilities* relies on *sweeping revocation*, e.g., for enforcing temporal safety
- Can be sped up by limiting sweeps to pages with capabilities (Cornucopia Reloaded, Zcheri)

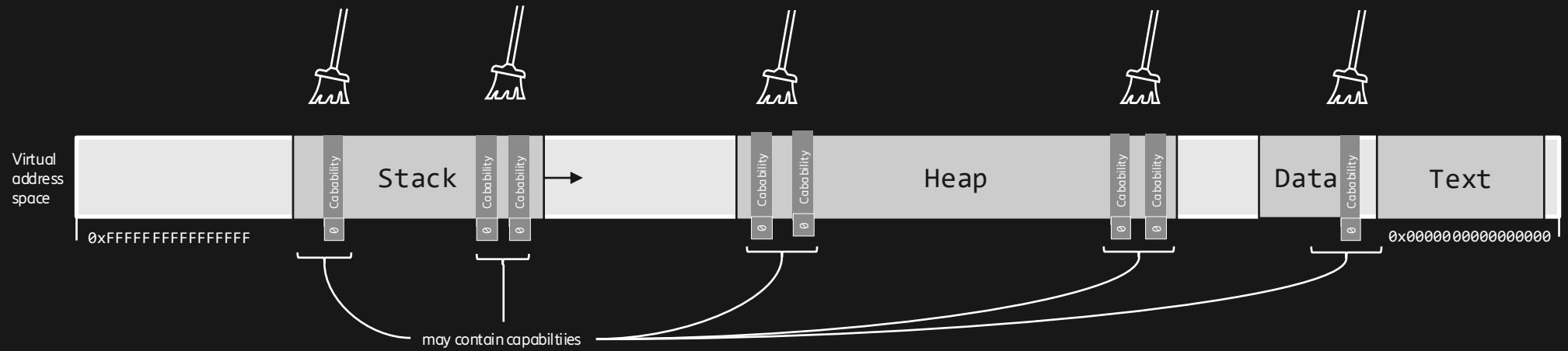


Filardo, N. W. et al. 2024. Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety. ASPLOS '24
Richardson A. et al. 2025. RISC-V Specification for CHERI Extensions. RVG-148 CHERI TG

Tradeoffs of avoiding indirection: revocation sweep

In CHERI, revocation requires knowing a capability's (and its validity tag's) location in memory

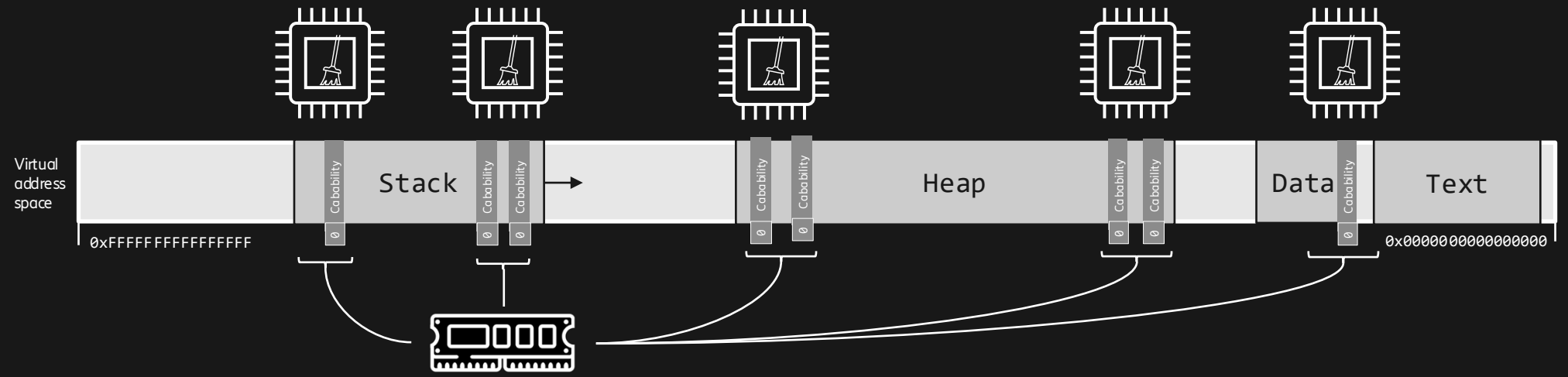
- Invalidating *sets of capabilities* relies on *sweeping revocation*, e.g., for enforcing temporal safety
- Can be sped up by limiting sweeps to pages with capabilities and parallelizing sweeps…



Tradeoffs of avoiding indirection: revocation sweep

In CHERI, revocation requires knowing a capability's (and its validity tag's) location in memory

- Invalidating *sets of capabilities* relies on *sweeping revocation*, e.g., for enforcing temporal safety
- Can be sped up by limiting sweeps to pages with capabilities and parallelizing sweeps…
- …but sweeps still consume **overall** system resources: CPU, memory bandwidth..



Rethinking indirection?

We are doing fine without indirection; do we even need it?

Rethinking indirection?

ERICSSON

We are doing fine without indirection; do we even need it?

In practice, all CHERI temporal-safety schemes have **indirect ways** of keeping track of invalid capabilities

Rethinking indirection?

We are doing fine without indirection; do we even need it?

In practice, all CHERI temporal-safety schemes have **indirect ways** of keeping track of invalid capabilities

- Shadow bitmap in Cornucopia / Cornucopia Reloaded
- Revocation bitmap and load filter in CHERIoT

Filardo, N. W. et al. 2024. Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety. ASPLOS '24
Amar S. et al. 2023. CHERIoT: Complete Memory Safety for Embedded Devices. MICRO '23

Rethinking indirection?

Indirection on every capability access is costly

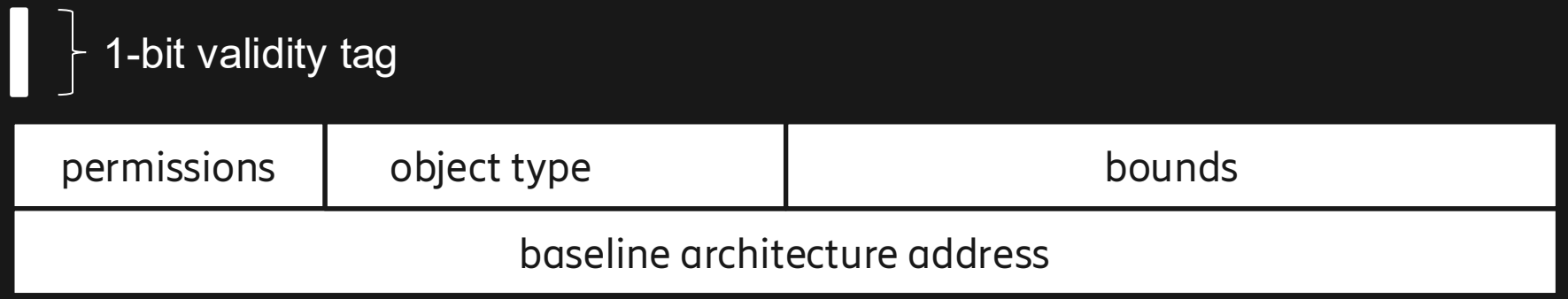
Rethinking indirection?

Indirection on every capability access is costly

Can we apply **indirection selectively** to only certain capabilities?

Colored Capabilities: High-Level Idea

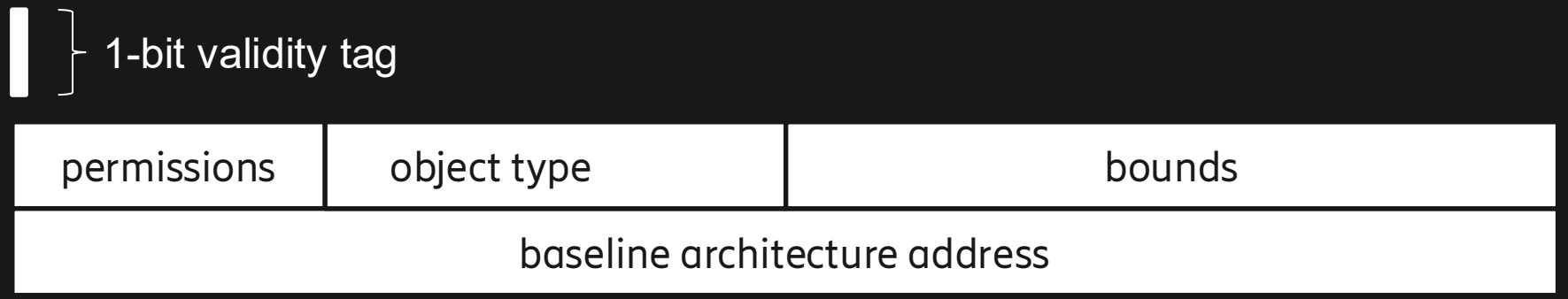
Colored Capabilities introduce a **limited form of indirection** to CHERI



Colored Capabilities: High-Level Idea

Colored Capabilities introduce a **limited form of indirection** to CHERI

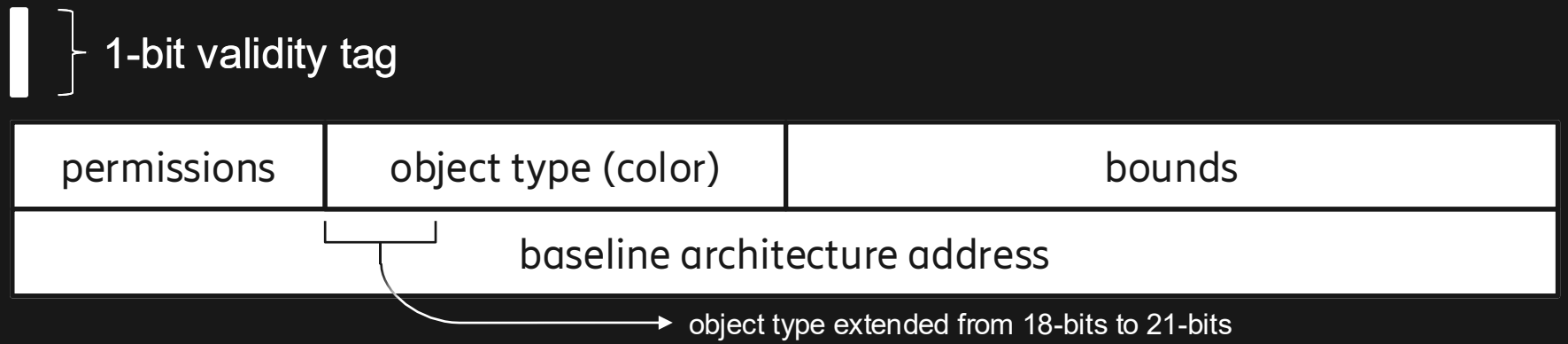
- Track **allocation provenance** of capabilities (**whether** a capability corresponds to valid allocation)
- Does **not** track **which** allocation (no memory tagging)



Colored Capabilities: High-Level Idea

Colored Capabilities introduce a **limited form of indirection** to CHERI

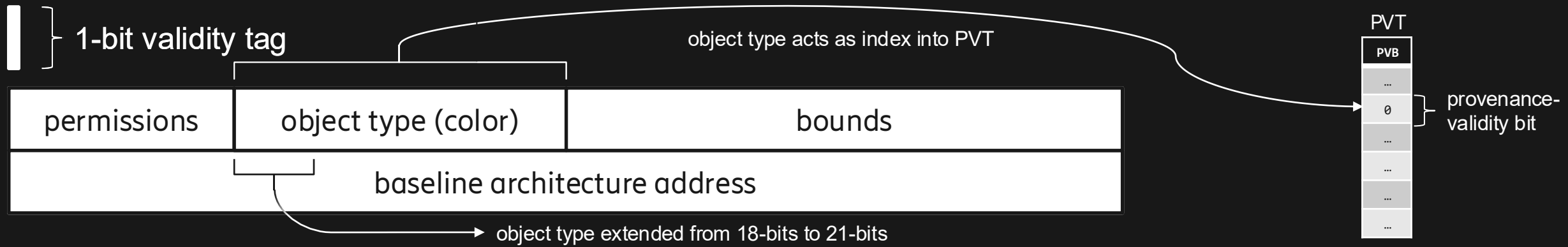
- Track **allocation provenance** of capabilities (**whether** a capability corresponds to valid allocation)
- Does **not** track **which** allocation (no memory tagging)
- All capabilities that share the same allocation provenance also share **provenance ID** (object type)



Colored Capabilities: High-Level Idea

Colored Capabilities introduce a **limited form of indirection** to CHERI

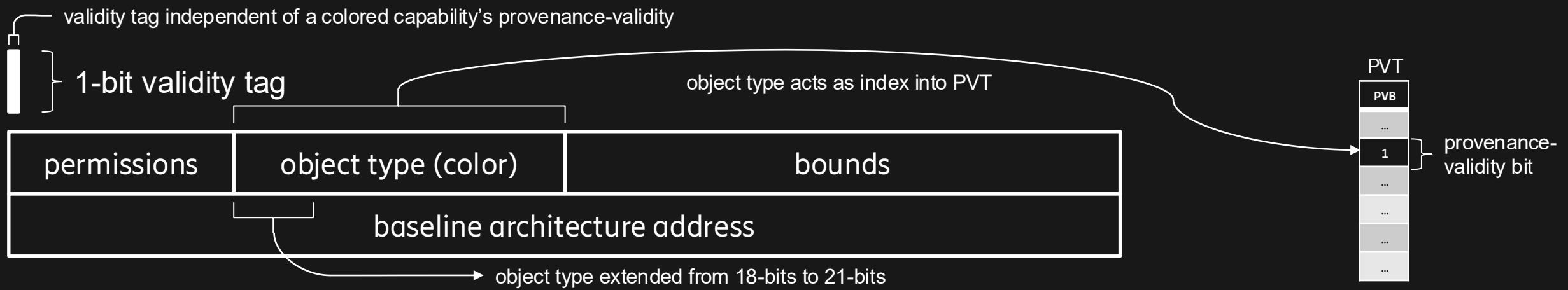
- Track **allocation provenance** of capabilities (**whether** a capability corresponds to valid allocation)
- Does **not** track **which** allocation (no memory tagging)
- All capabilities that share the same allocation provenance also share **provenance ID** (object type)
- The validity of an IDs allocation provenance is tracked by 1-bit entry in provenance-validity table (PVT)



Colored Capabilities: High-Level Idea

Colored Capabilities introduce a **limited form of indirection** to CHERI

- Track **allocation provenance** of capabilities (**whether** a capability corresponds to valid allocation)
- Does **not** track **which** allocation (no memory tagging)
- All capabilities that share the same allocation provenance also share **provenance ID** (object type)
- The validity of an IDs allocation provenance is tracked by 1-bit entry in provenance-validity table (PVT)
- Capabilities with provenance ID, but no provenance are **temporarily retracted** (not fully revoked)



Temporal Safety using Colored Capabilities

Use After Free / Use After Reallocation

Temporal safety relates to memory locations containing different objects at different points in time

Use After Free (UAF)

Accessing memory through a dangling pointer after original allocation has been freed

Use After Reallocation (UAR)

Dereferencing a dangling pointer after original memory has been reallocated and used for another object

```c
int main(void) {
    int *p = malloc(sizeof(int));
    //..
    free(p);  // p becomes a dangling pointer
    p[0] = 1; // use-after-free violation
    //...  p memory is allocated to p2
    p[0] = 1; // use-after-reallocation
}
```

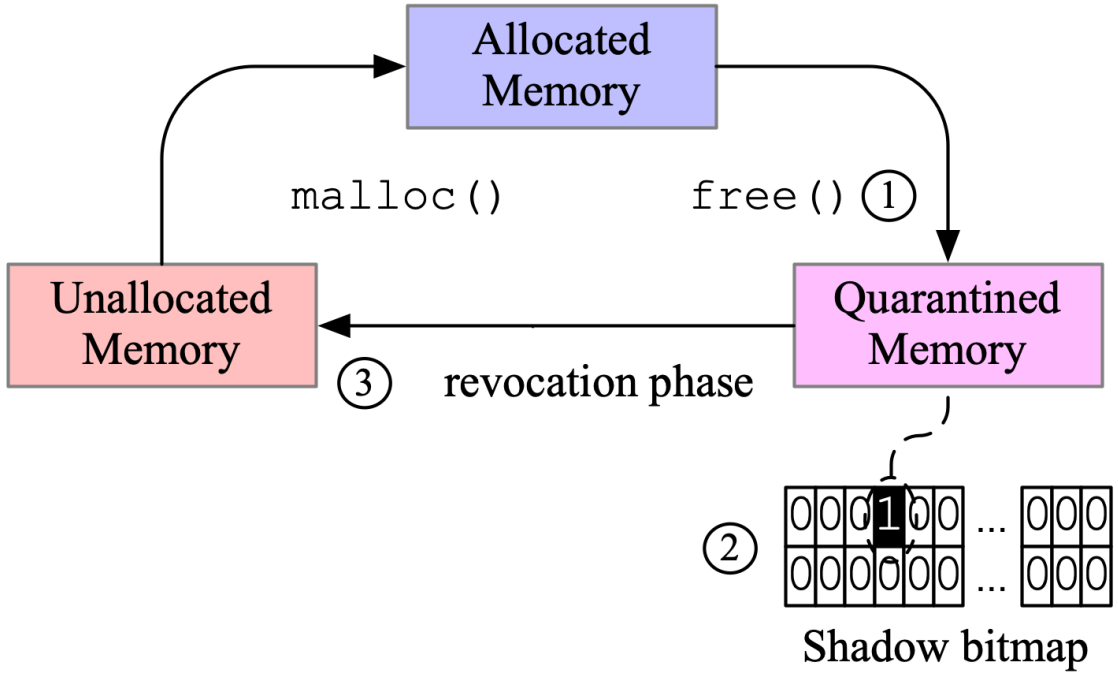
Prior work: Cornucopia

Cornucopia is capability revocation system for CHERI that implements deterministic temporal safety for standard heap allocations

① **Free → Quarantine**: Freed blocks are kept in a quarantine buffer until all capabilities referencing original allocation have been revoked (③)

② **Shadow bitmap**: Freed regions marked at word granularity in shadow bitmap

③ **Periodic sweeping**: Revocation service scans memory for stale (freed) capabilities, clears their tags and frees quarantined memory for reuse



Filardo, N. W. et al. 2020. Cornucopia: Temporal Safety for CHERI Heaps. IEEE S&P '20.
Filardo, N. W. et al. 2024. Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety. ASPLOS '24

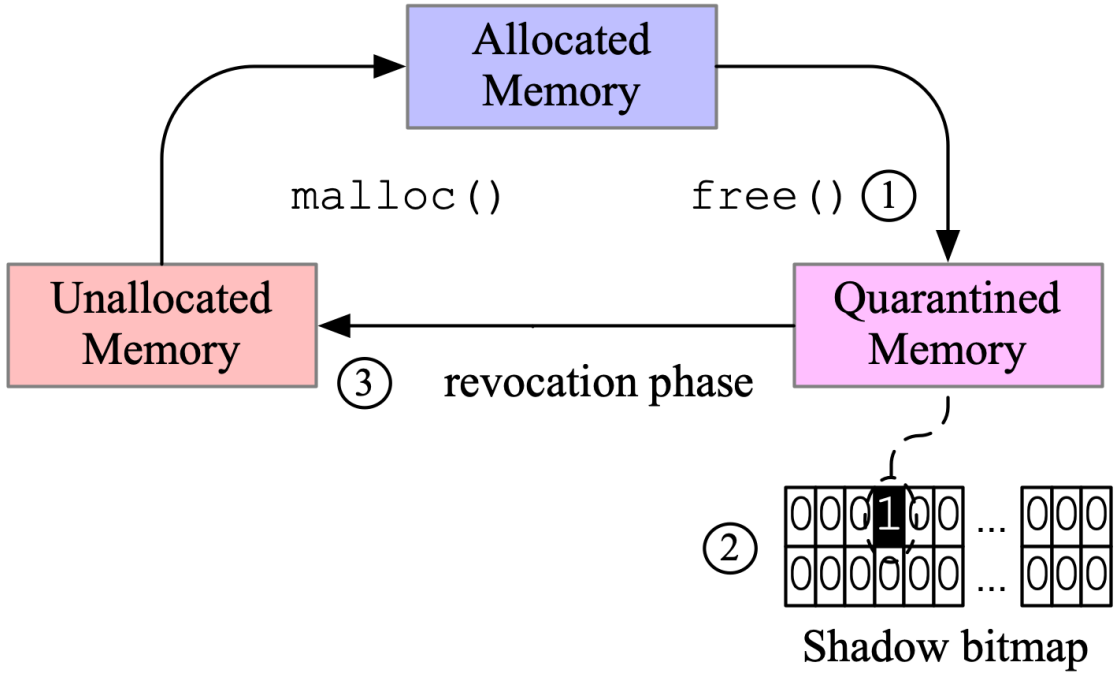
Prior work: Cornucopia

Cornucopia is capability revocation system for CHERI that implements deterministic temporal safety for standard heap allocations

① **Free → Quarantine**: Freed blocks are kept in a quarantine buffer until all capabilities referencing original allocation have been revoked (③)

② **Shadow bitmap**: Freed regions marked at word granularity in shadow bitmap

③ **Periodic sweeping**: Revocation service scans memory for stale (freed) capabilities, clears their tags and frees quarantined memory for reuse



- Only address the use-after-reallocation bugs
- The use of a quarantine buffer increases memory usage
- Revocation frequency scales conversely with memory overhead

Filardo, N. W. et al. 2020. Cornucopia: Temporal Safety for CHERI Heaps. IEEE S&P '20.
Filardo, N. W. et al. 2024. Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety. ASPLOS '24

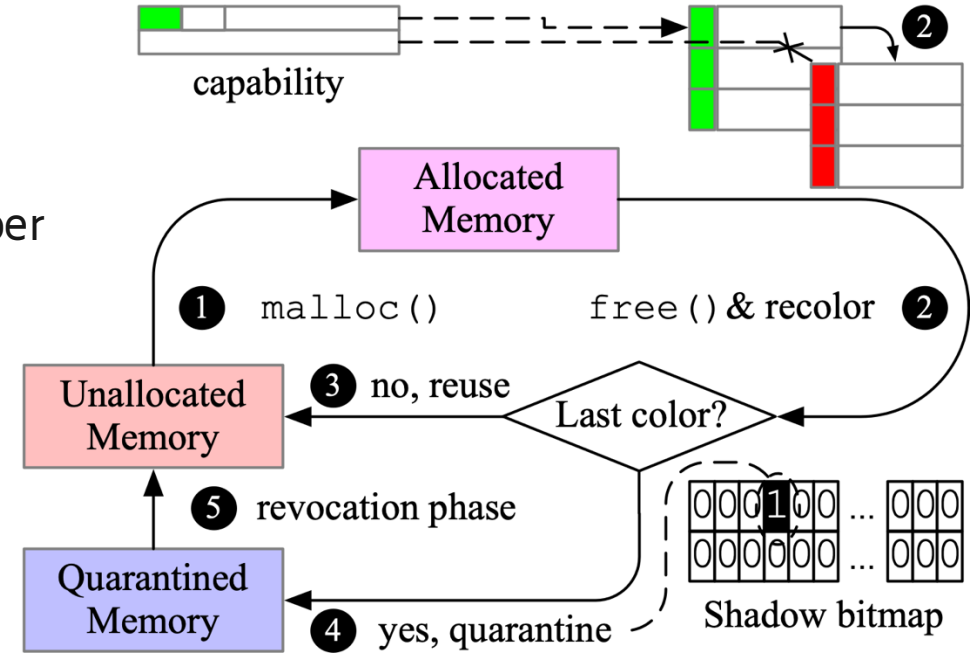
Prior work: Cornucopia + Memory Coloring

Combining CHERI with memory tagging to close the UAR/UAF mitigation gap

1. **Allocation:** capability + memory tag assigned same color ID
2. **Free:** memory is coloring to invalidate stale pointers
3. **Reuse:** freed colors recycled. Detects dangling pointers, but number of colors is limited

When colors IDs (e.g., 16 for MTE) exhausted:

4. Quarantine freed memory
5. Fall back to Cornucopia's capability sweeping



Watson, R. N. M. 2023. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9). UCAM-CL-TR-987

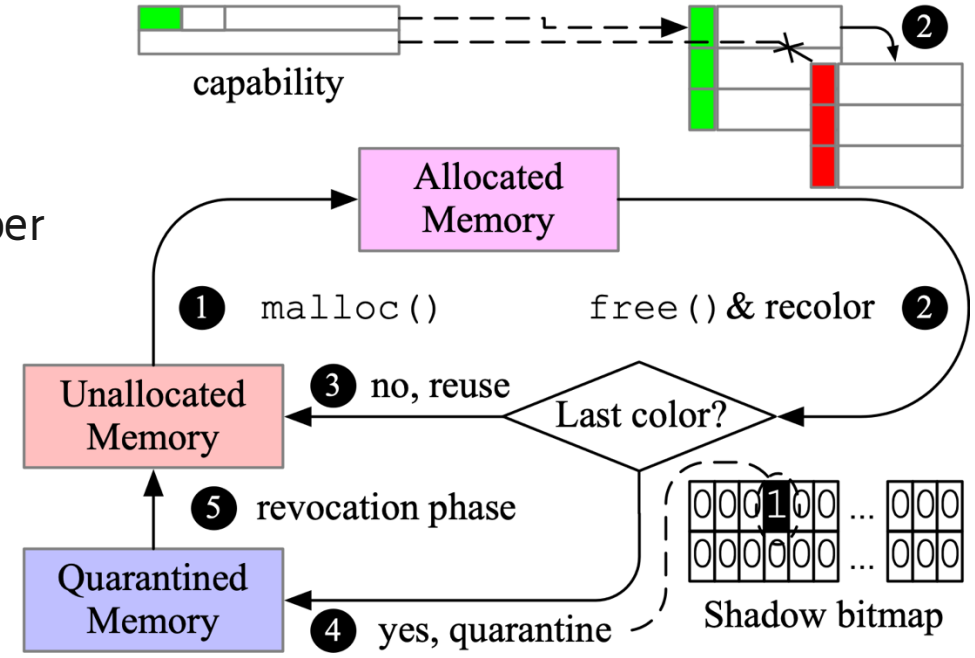
Prior work: Cornucopia + Memory Coloring

Combining CHERI with memory tagging to close the UAR/UAF mitigation gap

1. **Allocation:** capability + memory tag assigned same color ID
2. **Free:** memory is coloring to invalidate stale pointers
3. **Reuse:** freed colors recycled. Detects dangling pointers, but number of colors is limited

When colors IDs (e.g., 16 for MTE) exhausted:

4. Quarantine freed memory
5. Fall back to Cornucopia's capability sweeping



\+ Addresses the UAF/UAR gap

– The use of a quarantine buffer increases memory usage

– Revocation frequency scales conversely with memory overhead

Watson, R. N. M. 2023. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9). UCAM-CL-TR-987

Colored Capability Goals

Deterministically close the UAF/UAR gap

Immediately reuse free memory (no quarantine buffer)

Reduce the frequency of revocation sweeps

Temporal Safety using Colored Capabilities

2 million color

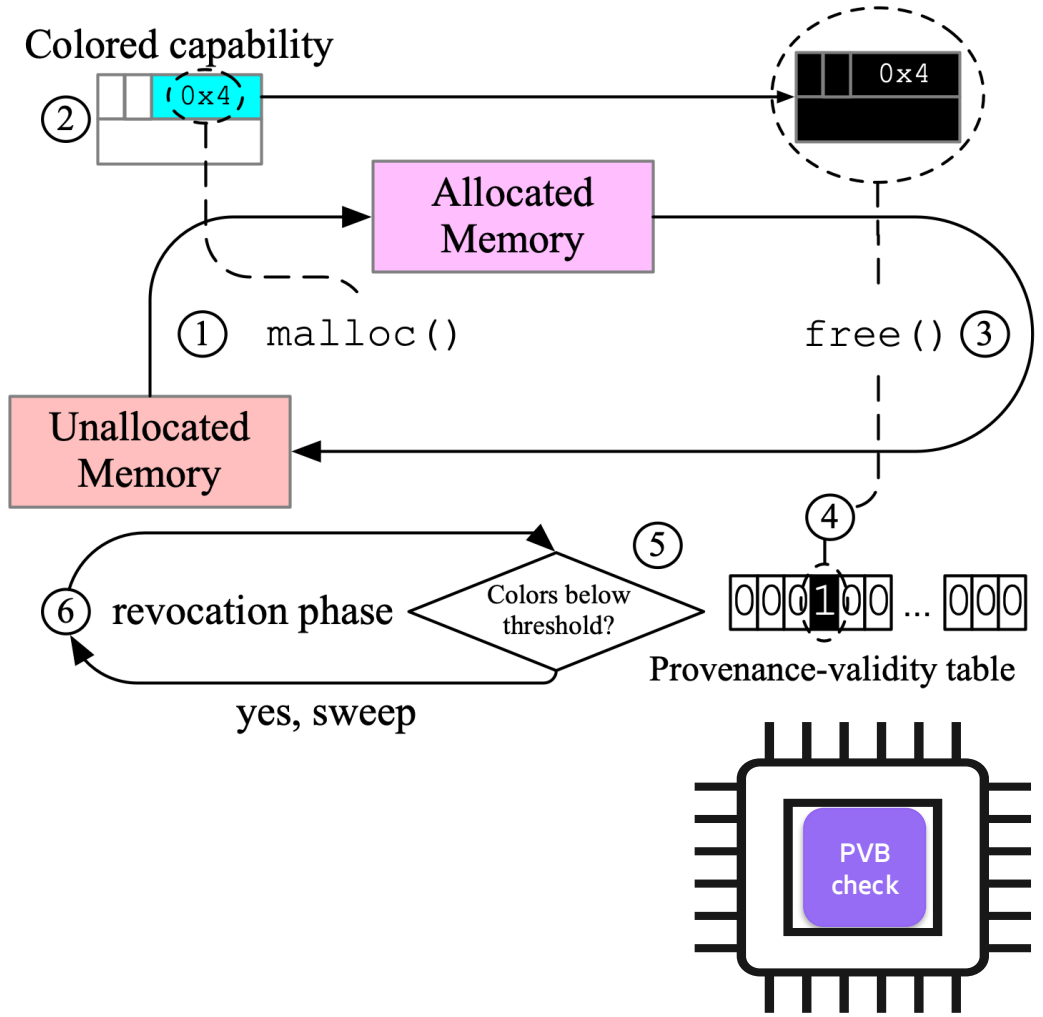
①, ② **On allocation**: Assign a unique provenance ID from available (unassigned) otype (color) values to capability

- Becomes colored capability by setting otype to assigned value

③, ④ **On free**: Mark provenance ID's PVB as invalid in PVT

- CPU prevents dereferences through colored capabilities whose provenance-validity bit (PVB) is invalid

⑤, ⑥ **When available colors below pre-set threshold**: Sweep memory and revoke capabilities marked invalid in PVT



Temporal Safety using Colored Capabilities

2 million color

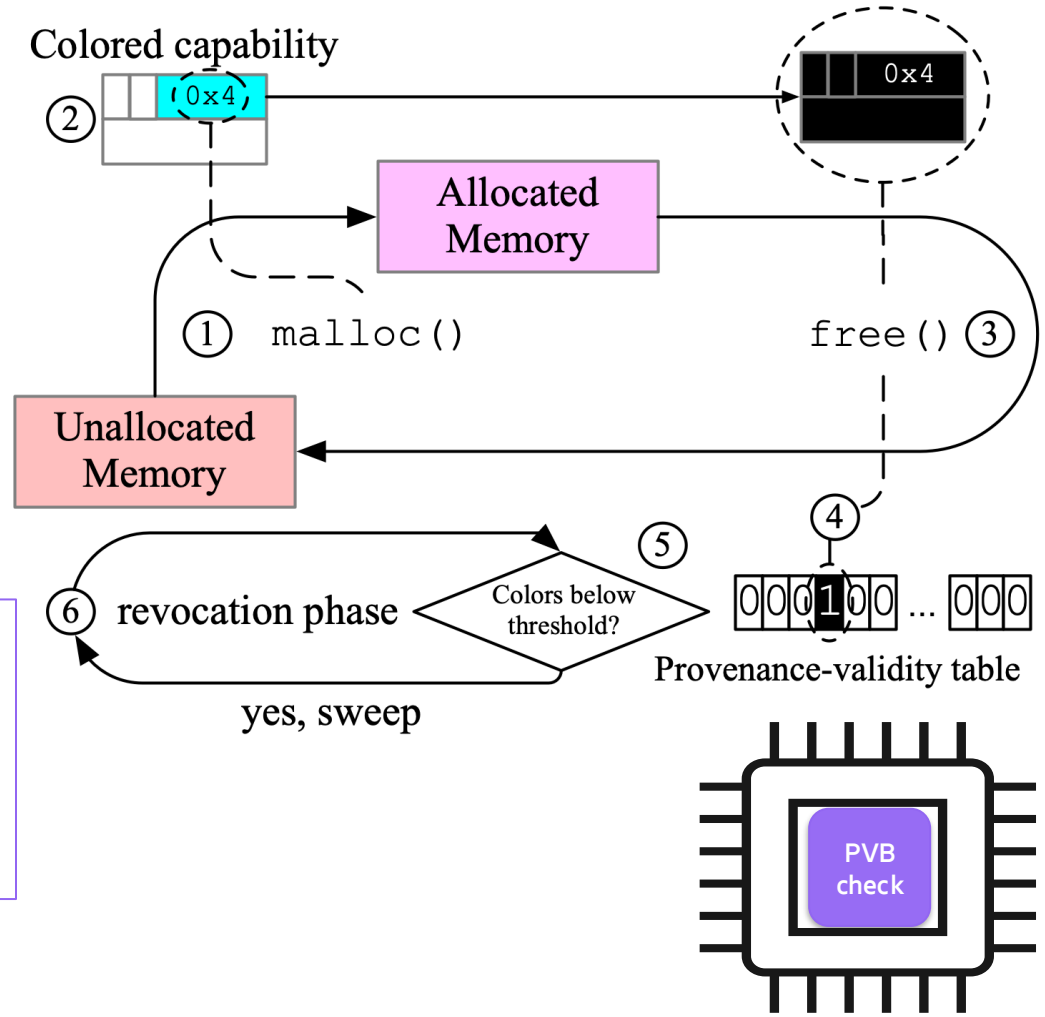
①, ② **On allocation**: Assign a unique provenance ID from available (unassigned) otype (color) values to capability

- Becomes colored capability by setting otype to assigned value

③, ④ **On free**: Mark provenance ID's PVB as invalid in PVT

- CPU prevents dereferences through colored capabilities whose provenance-validity bit (PVB) is invalid

⑤, ⑥ **When available colors below pre-set threshold**: Sweep memory and revoke capabilities marked invalid in PVT



+ Addresses the UAF/UAR gap
+ Immediately reuse freed memory
+ Revocation frequency scales with number of supported colors

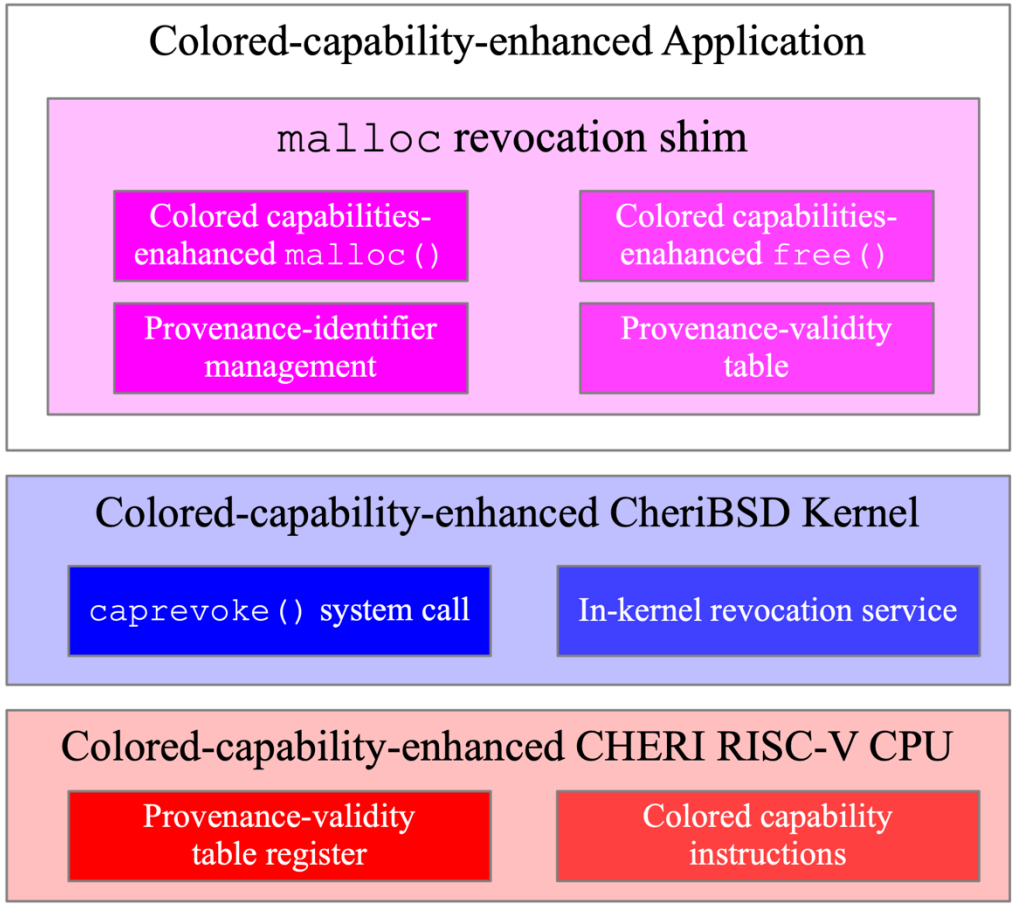
PICASSO: Colored Capability System Architecture

Hardware: We implemented two versions of PICASSO:

1. QEMU-system-CHERI128 full-system emulator, and
2. CHERI-Toooba processor with colored capability support
   - Supporting ~2 million of allocations using 21-bit otype
   - PVB checks on loads and stores using colored capabilities
   - PVB buffer caching PVT fetches, reducing cycle overhead
   - Compatible with CHERI sealing by reserving subset of otypes

Software: We integrated colored capabilities into CheriBSD 25.03.

- Memory allocator support by integrating colored capabilities into CheriBSD malloc revocation shim (MRS)
- Userspace provenance-identifier management (unr allocator)
- Colored Capability-enhanced revocation integrated into CherBSD in-kernel revocation service



Evaluation

RQ1: Can PICASSO reliably detect UAF conditions?

RQ2: What is the performance and memory overhead of enforcing temporal-safety with PICASSO?

RQ3: To what extent does PICASSO reduce the frequency of CHERI revocation sweeps?

RQ1: Can PICASSO reliably detect UAF conditions?

We evaluate PICASSO using Juliet Test Suite:

- All CWE-416 (Use After Free) test cases
- All CWE-415 (Double Free) test cases
- In total: 1385 "bad" test cases
- Equal number of "good" (patched) test cases

PICASSO detects **all bad cases** and **no false positives**

RQ1: Can PICASSO reliably detect UAF conditions?

We evaluate PICASSO using Juliet Test Suite:

- All CWE-416 (Use After Free) test cases
- All CWE-415 (Double Free) test cases
- In total: 1385 "bad" test cases
- Equal number of "good" (patched) test cases

PICASSO detects **all bad cases** and **no false positives**

We compare our results against CheriBSD Cornucopia in two-different configurations:

- Default configuration fails to detect any bad CWE-416 cases (since test cases do not reallocate)
- In **revoke-on-free** configuration Cornucopia detects all CWE-416 cases but at increased perf overhead
- Both configurations detect all CWE-415 bad cases

RQ1: Can PICASSO reliably detect UAF conditions?

We evaluate PICASSO using Juliet Test Suite:

- All CWE-416 (Use After Free) test cases
- All CWE-415 (Double Free) test cases
- In total: 1385 "bad" test cases
- Equal number of "good" (patched) test cases

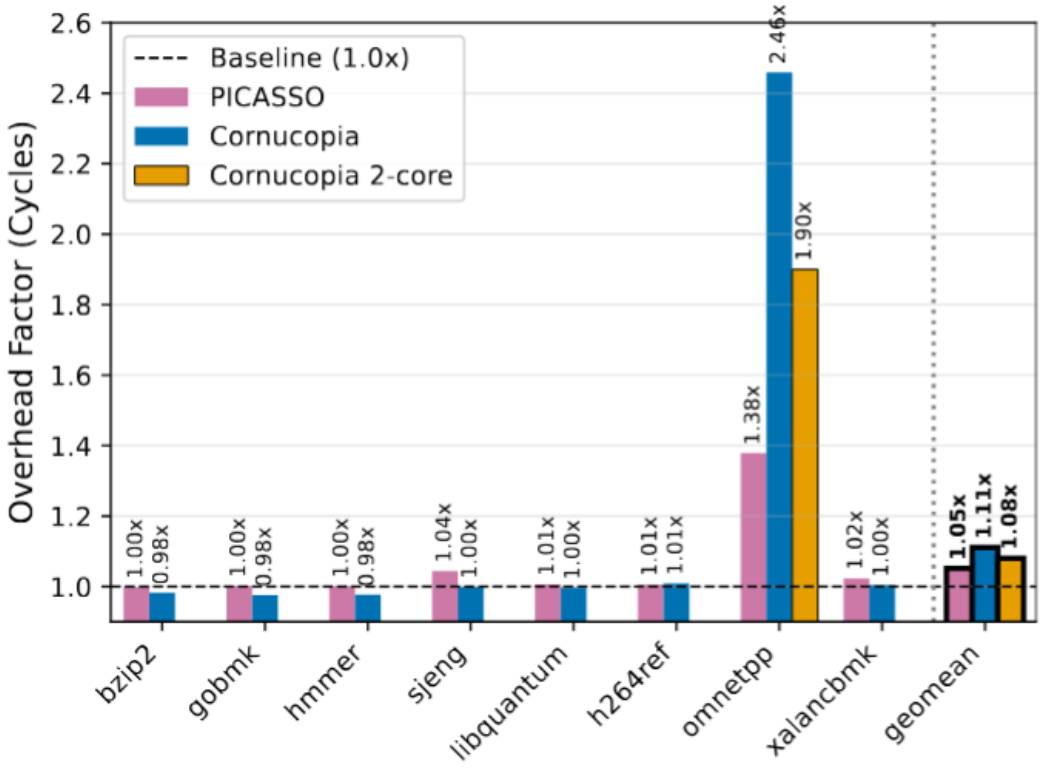
PICASSO detects **all bad cases** and **no false positives**

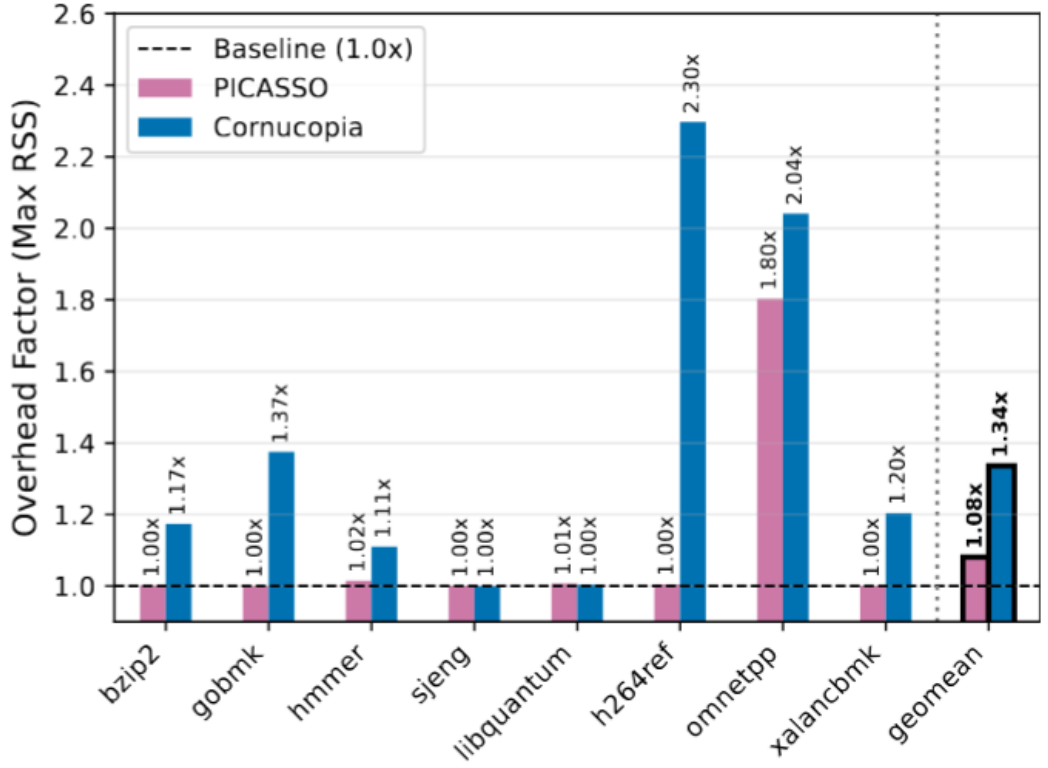
We compare our results against CheriBSD Cornucopia in two-different configurations:

- Default configuration fails to detect any bad CWE-416 cases (since test cases do not reallocate)
- In **revoke-on-free** configuration Cornucopia detects all CWE-416 cases but at increased perf overhead
- Both configurations detect all CWE-415 bad cases

PICASSO achieves perfect, deterministic accuracy in detecting use-after-free conditions
PICASSO successfully closes the UAF / UAR gap

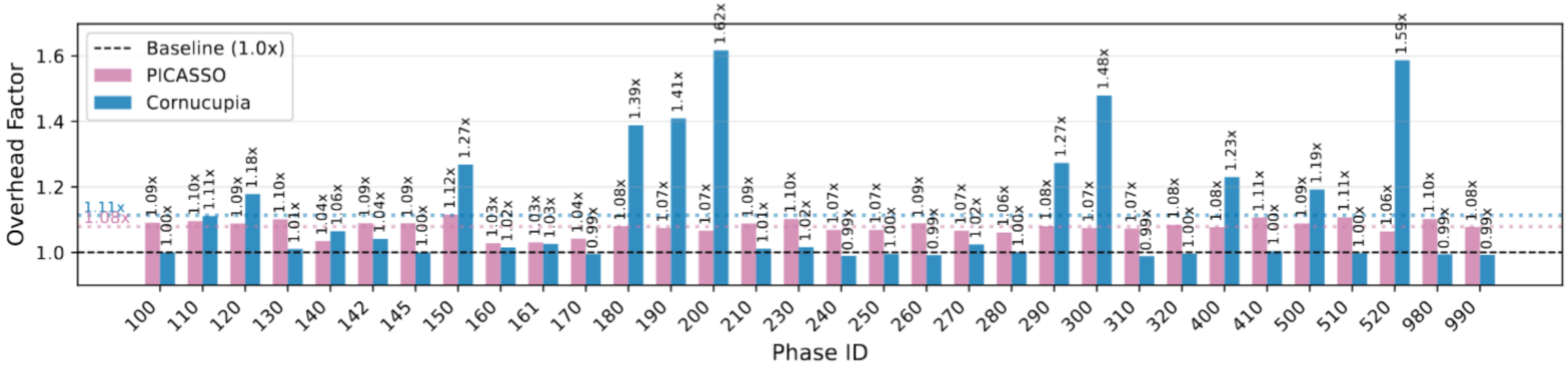
RQ2: SPEC 2006 benchmark Results





- **Performance overhead**: PICASSO (5% g.m.) and CheriBSD 25.03 Cornucopia (11% g.m. single-core, 8% g.m. 2-core)
- **Memory overhead** (measured as maximum resident set size): PICASSO (8% g.m.) and Cornucopia (34% g.m.)

RQ2: SQLite speedtest1



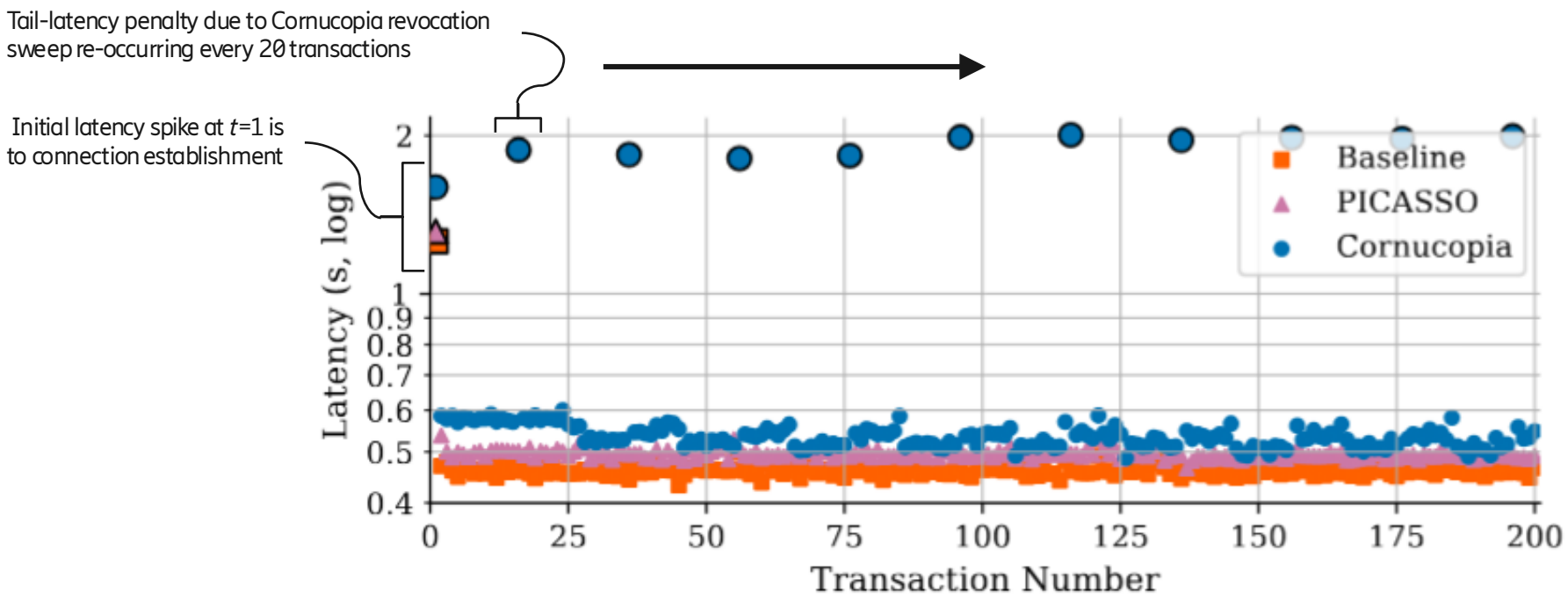
Performance overhead

- PICASSO: 8% avg., 12% max.
- Cornucopia: %12 avg., 62% max.

Memory overhead

- PICASSO: no memory overhead
- Cornucopia: 99% memory overhead across whole benchmark

RQ2: PostgreSQL pgbench



Latency graph for 200 sample pgbench transactions with markers denoting individual response times

Throughput degradation (transactions per second)

- PICASSO: 6.2%
- Cornucopia: 23.1%

RQ3: Reduction in frequency of revocation sweeps?

		# Revocations	
Benchmark	**# Allocations**	**Cornucopia**	**PICASSO**
bzip2	87	3	0
gobmk	121361	17	0
hmmer	170126	16	0
sjeng	5	0	0
libquantum	108	1	0
h264ref	38270	12	0
omnetpp	129620400	2792	76
xalancbmk	1057345	2	0
SQLite	224917	272	0
Benchmark	**# Transactions**	**Cornucopia**	**PICASSO**
PostgreSQL	≈66000	≈ 3300	1
gRPC	≈100000	≈ 400	1

No revocation with PICASSO in majority of benchmarks

- Further experiments with long-running PostgreSQL and gRPC benchmarks show that PICASSO significantly reduces revocation time without increasing the latency percentile even with p999

Conclusion

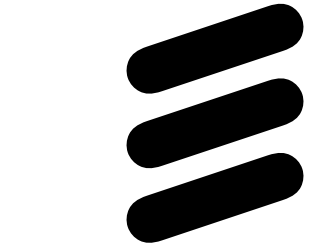
Colored Capabilities enhance CHERI's temporal memory safety through efficient support for **bulk retraction** of capabilities

Colored Capabilities add **limited indirection** to CHERI's capability model and

- Address the **UAR/UAF gap**
- Reduces memory overhead by **eliminating quarantine buffers**
- Significantly **reduce the frequency of needed revocation sweeps**

https://arxiv.org/abs/2602.09131

- Evaluation on **thousands of NIST Juliet test cases**, **SPEC CPU benchmarks**, and **latency-sensitive real-world benchmarks** confirm that **colored capabilities improve temporal safety** with **reasonable performance overhead** (3% – 8% overhead on average compared to CHERI purecap baseline)



ERICSSON

https://www.ericsson.com/en/security