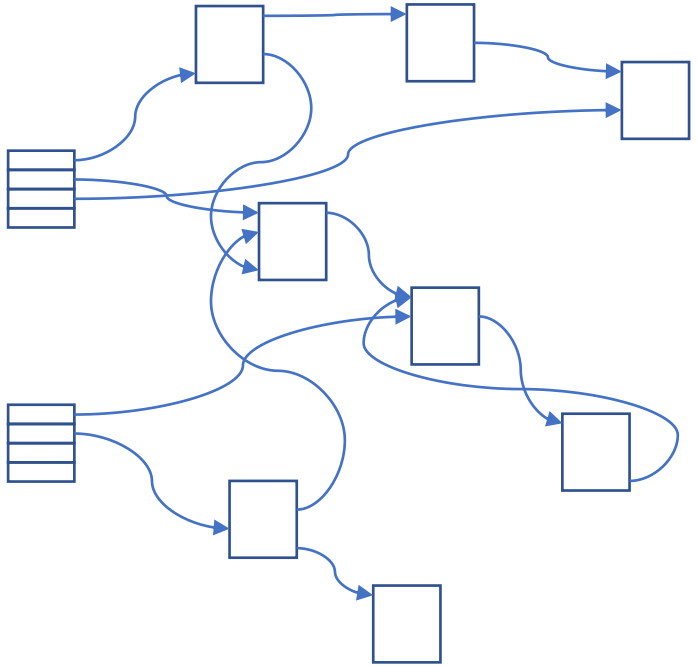
Towards a Safe, High-Performance Heap Allocator

Lessons from CHERIfying snmalloc (so far)

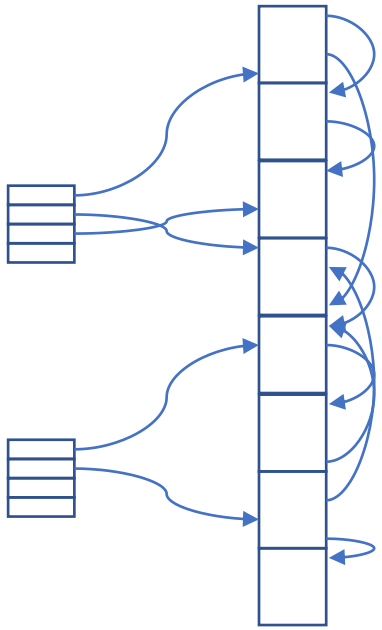
David Chisnall

Building the object abstraction

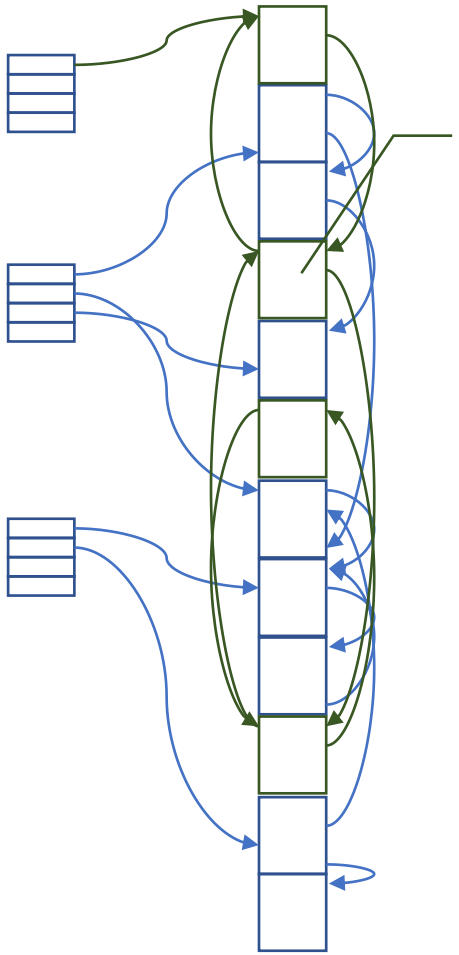
Building the object abstraction



Building the object abstraction

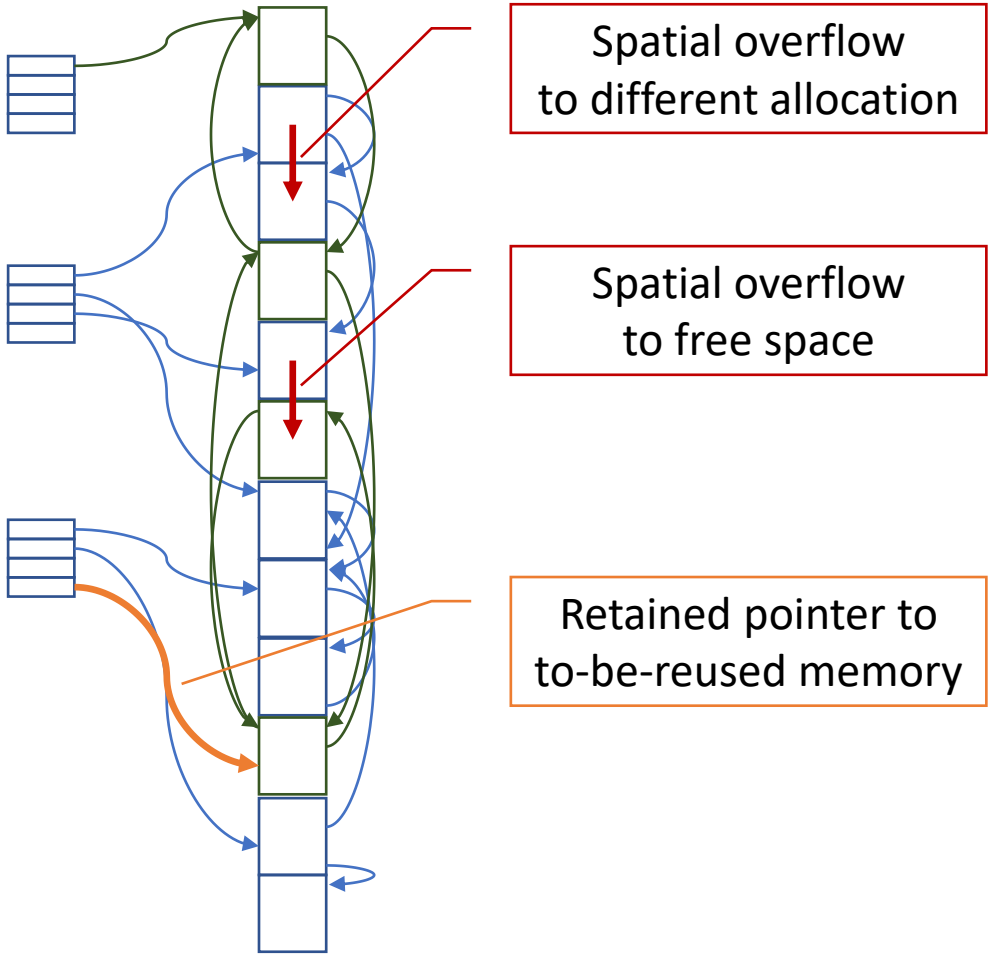


Building the object abstraction

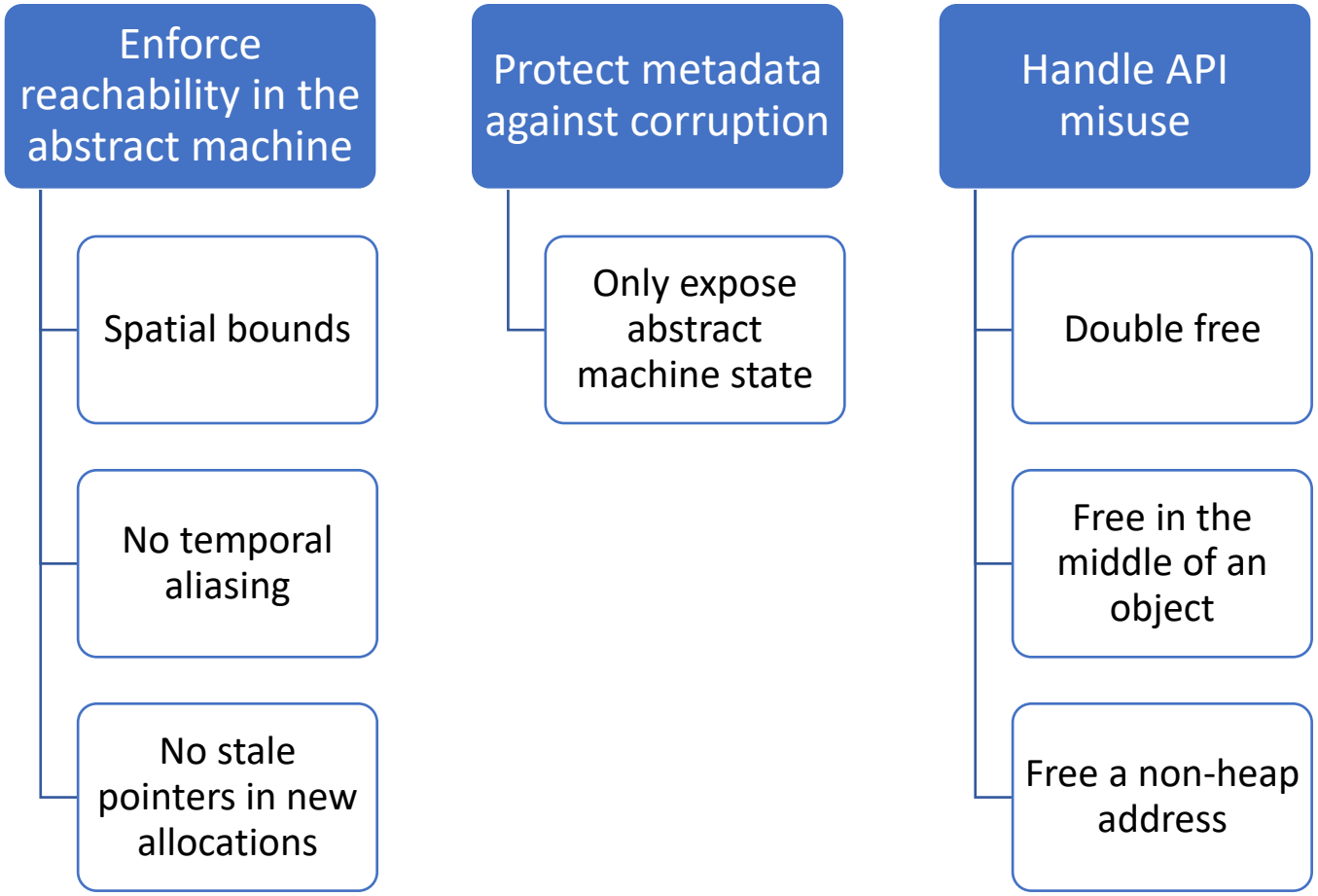


Free regions for reuse must be tracked by allocator

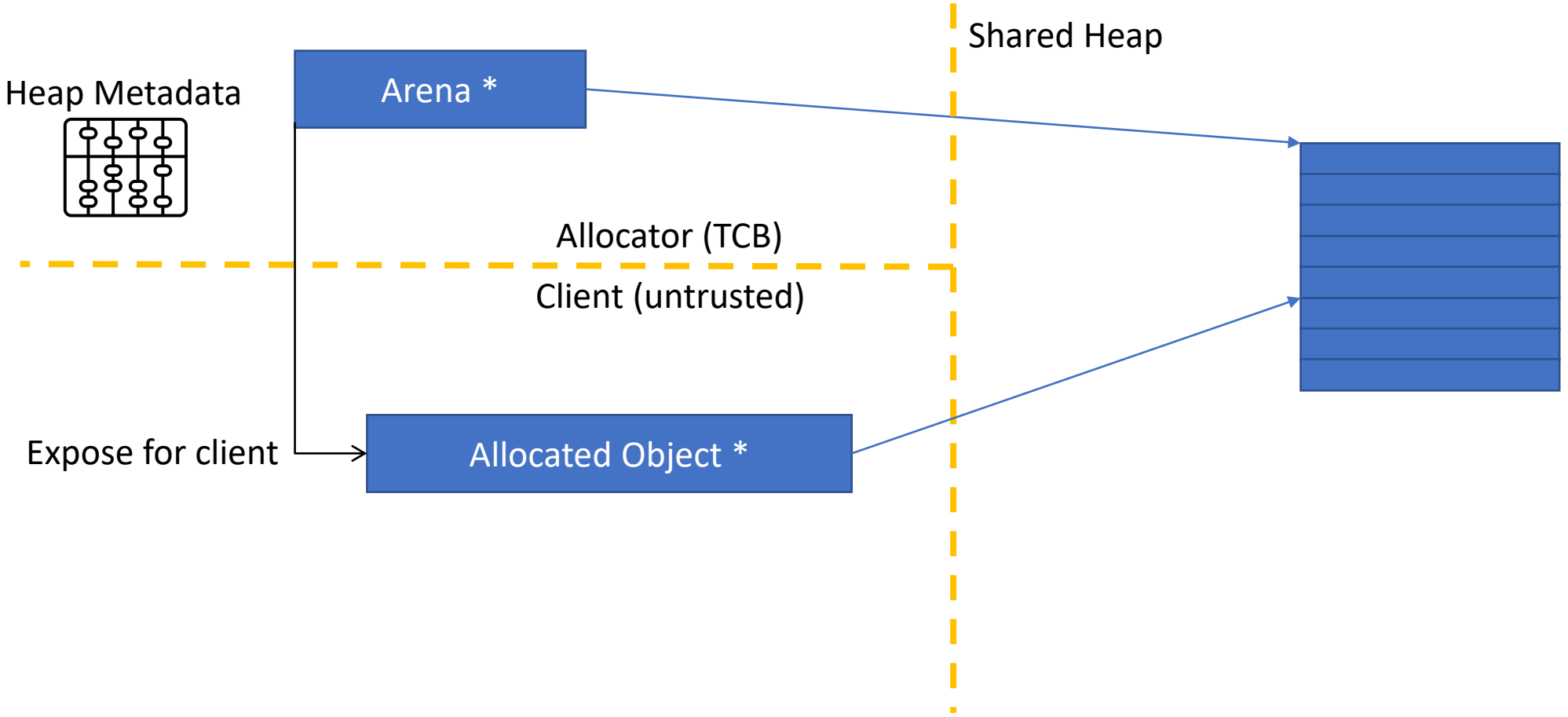
Less-than-full abstraction



Towards full abstraction for the heap



Heap allocator is a core part of the TCB



snmalloc

A High-performance Allocator with Lightweight Hardening

snmalloc pre-CHERI defenses

Threat		Pre-CHERI	
Spatial separation		General	Canaries
Information disclosure			
Temporal aliasing			
Metadata access or corruption	Out-of-band		
	In-band		
Incorrect free			

Canary free objects and occasional tests of liveness

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

snmalloc pre-CHERI defenses

Threat		Pre-CHERI	
Spatial separation		General	Canaries
		`memcpy`	Checked
Information disclosure			
Temporal aliasing			
Metadata access or corruption	Out-of-band		
	In-band		
Incorrect free			

Canary free objects and occasional tests of liveness

Look up allocator metadata for source & dest!

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

snmalloc pre-CHERI defenses

Threat	Pre-CHERI	
Spatial separation	General	Canaries
	`memcpy`	Checked
Information disclosure	0 on alloc (optional)	
Temporal aliasing		
Metadata access or corruption	Out-of-band	
	In-band	
Incorrect free		

Canary free objects and occasional tests of liveness

Look up allocator metadata for source & dest!

Opt-in zero when allocating

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

snmalloc pre-CHERI defenses

Threat		Pre-CHERI	
Spatial separation		General	Canaries
		`memcpy`	Checked
Information disclosure		0 on alloc (optional)	
Temporal aliasing		Randomized free queues	
Metadata access or corruption	Out-of-band	Randomized location & guard pages	
	In-band		
Incorrect free			

Canary free objects and occasional tests of liveness

Look up allocator metadata for source & dest!

Opt-in zero when allocating

Randomization to frustrate attacker's attempts to locate objects of interest

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

snmalloc pre-CHERI defenses

Threat		Pre-CHERI	
Spatial separation		General	Canaries
		`memcpy`	Checked
Information disclosure		0 on alloc (optional)	
Temporal aliasing		Randomized free queues	
Metadata access or corruption	Out-of-band	Randomized location & guard pages	
	In-band	Pointer obfuscation & lightweight MAC	
Incorrect free		↑ & (opt-in) check of ptr to object start	

Canary free objects and occasional tests of liveness

Look up allocator metadata for source & dest!

Opt-in zero when allocating

Randomization to frustrate attacker's attempts to locate objects of interest

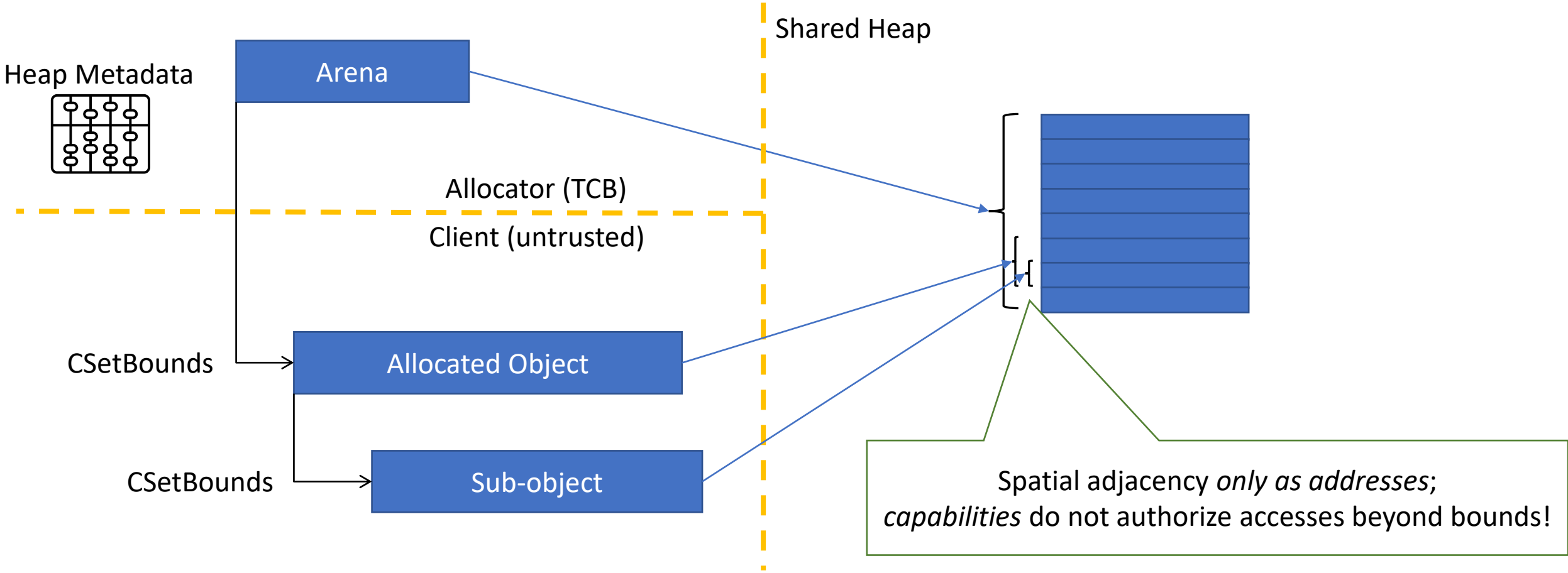
Optional "encrypt and MAC" on in-band metadata: minimizes disclosure and detects tampering (whp)

"Same object twice" DF, not "temporally aliased"

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

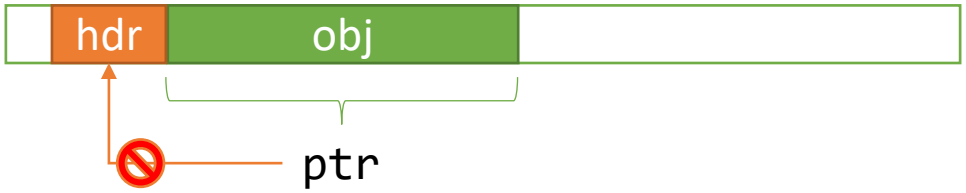

Enter CHERI

CHERI capabilities capture provenance

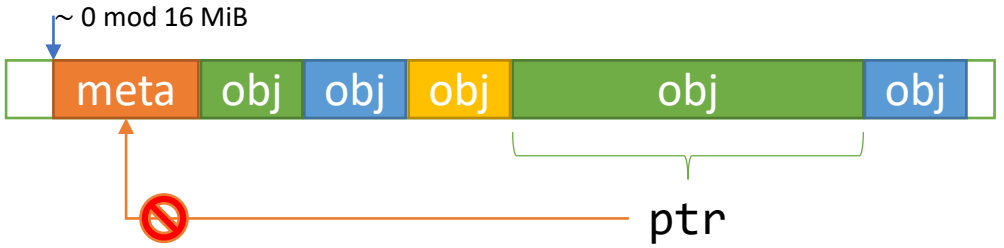


What about `free()`?

Per-object headers?

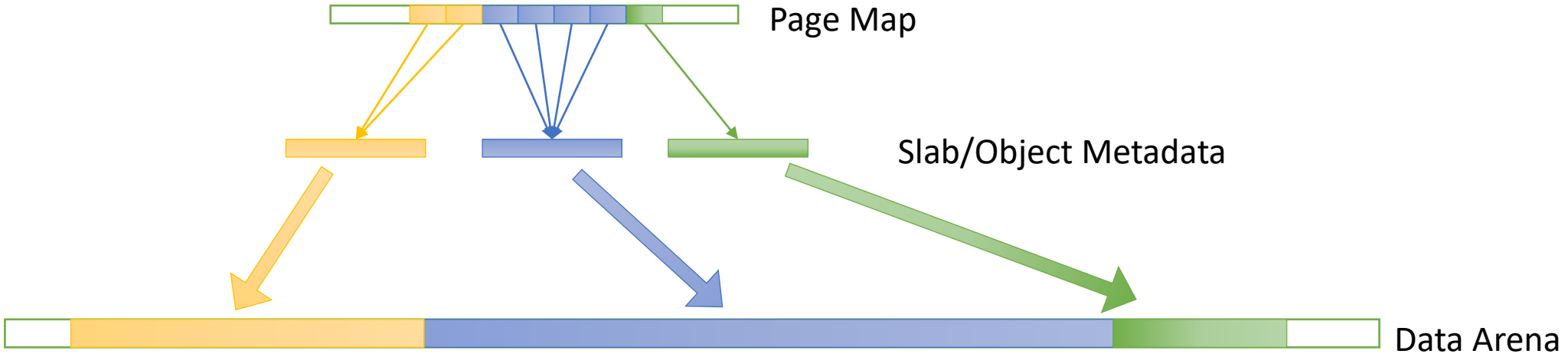


Per-segment headers?



- Bounding in `malloc()` means `free()` can't use argument as pointer!
  - Need to reach metadata via allocator-private state (global/TLS/handle)

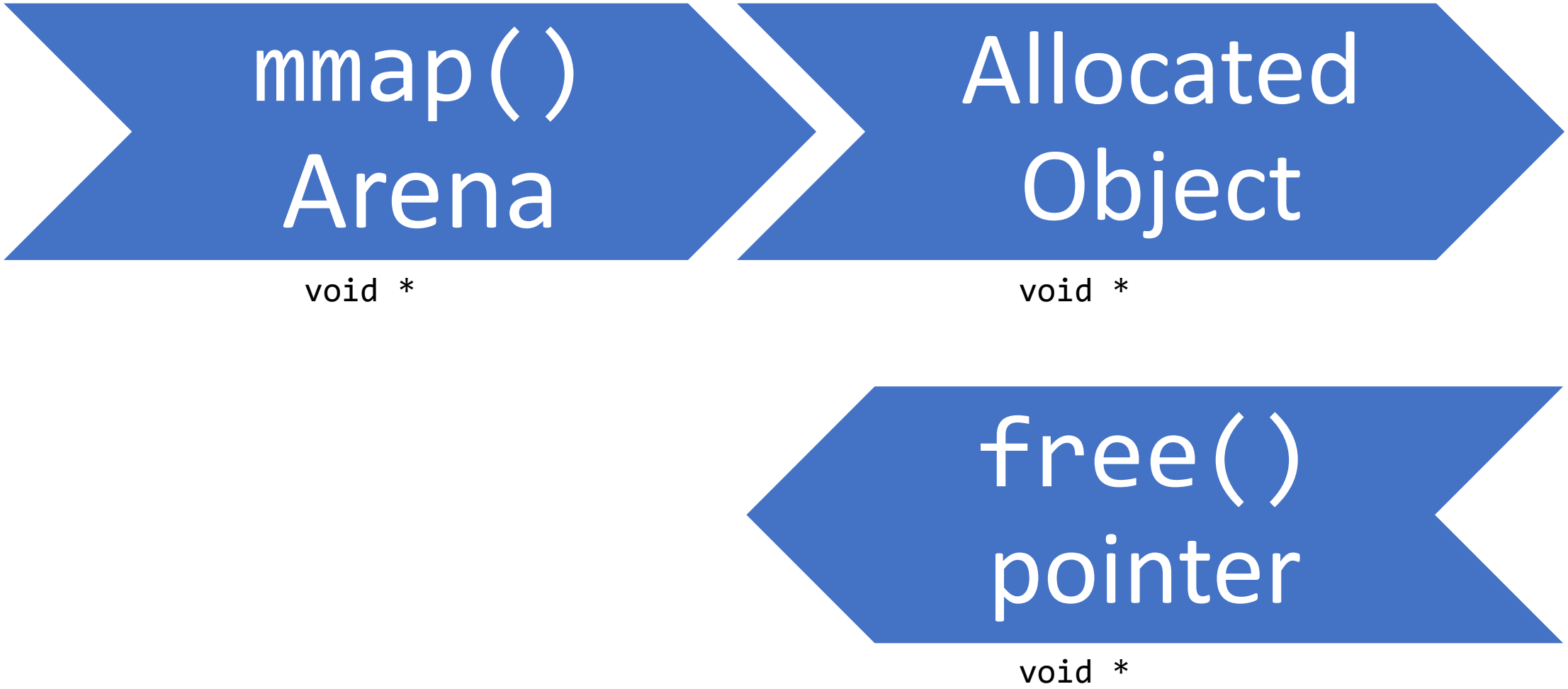
`free()` requires *amplification*



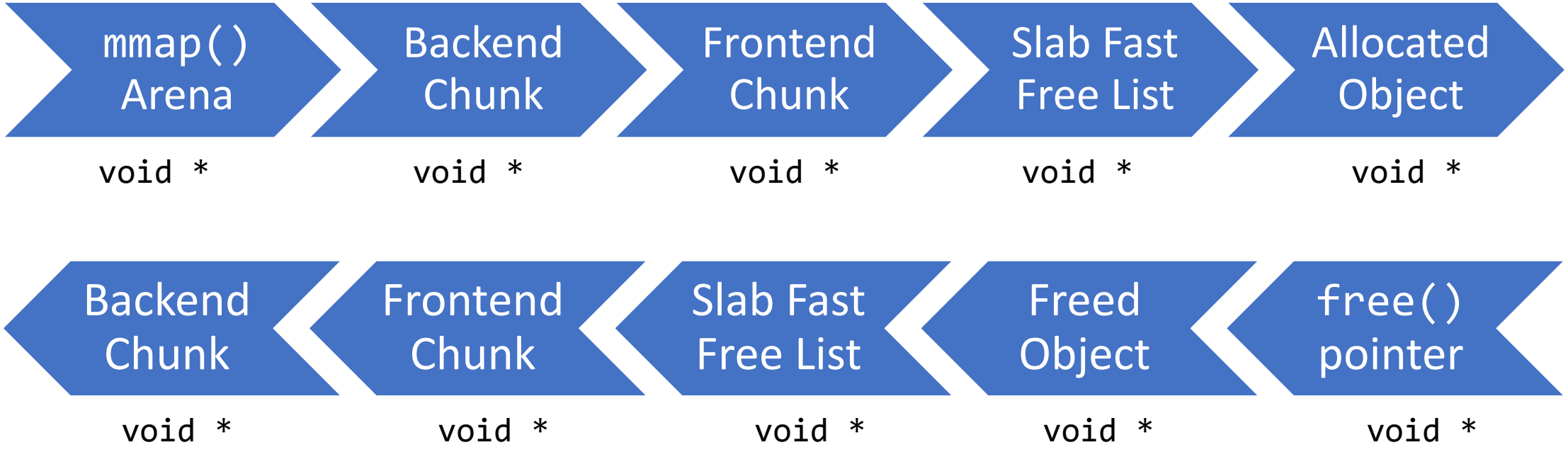
- snmalloc's central internal data structure is its "Page Map"
  (VA / 16KiB) ↦ Per-Slab or Per-large-object Metadata

- Convenient place to stash widely-bounded pointers

Don’t stare into the `void*`

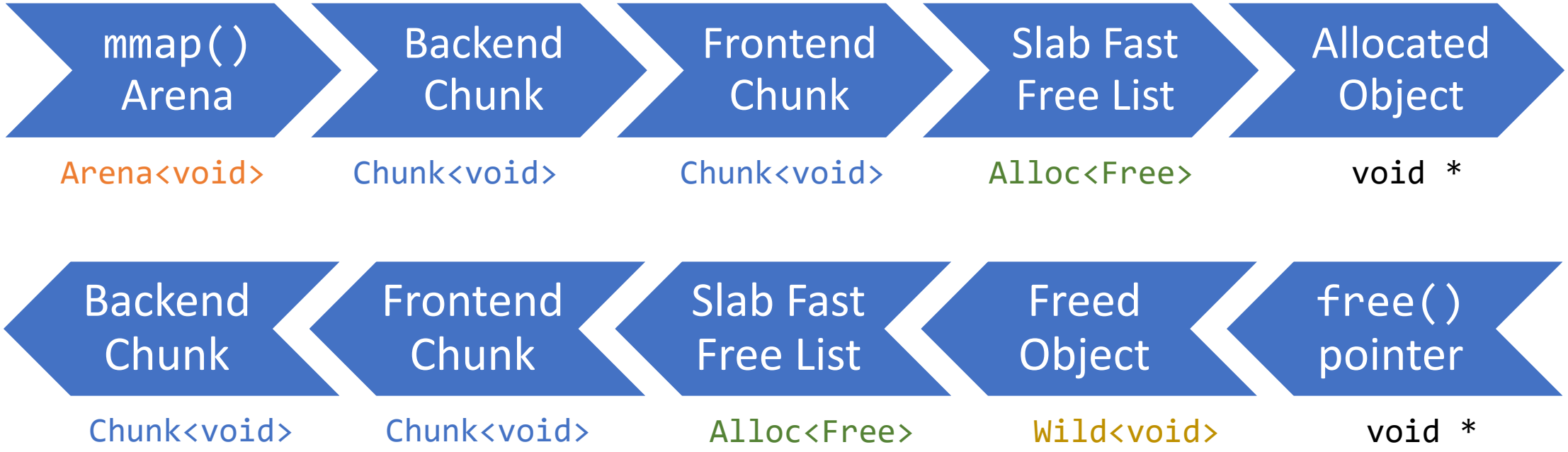


Don’t stare into the `void*`



Don’t stare into the `void*`

`CapPtr<T,B>` aka **`B<T>`**: T* with *static bound annotation* B (`Arena` > `Chunk` > `Alloc` > `Wild`)



Safe affordances:

```
CapPtr<T, BOut> capptr_bound(CapPtr<U, BIn>, size_t); // BOut ≤ BIn
void* capptr_reveal(Alloc<void>);
```

Initial CHERIfication of snmalloc

Threat		Pre-CHERI		CHERI
Spatial separation		General	Canaries	Set bounds
		`memcpy`	Checked	
Temporal aliasing		Randomized free queues		←
Information disclosure		0 on alloc (optional)		←, CapPtr
Metadata access or corruption	Out-of-band	Randomized location & guard pages		Capability reachability
	In-band	Pointer obfuscation & lightweight MAC		Seal* & MAC
Incorrect free		↑ & (opt-in) check of ptr to object start		↑ & ←'s check

CHERI spatial bounds do their thing!

Still just randomized defenses

`CapPtr` eases auditing

Clients not linked to snmalloc globals, but arena caps can still be leaked

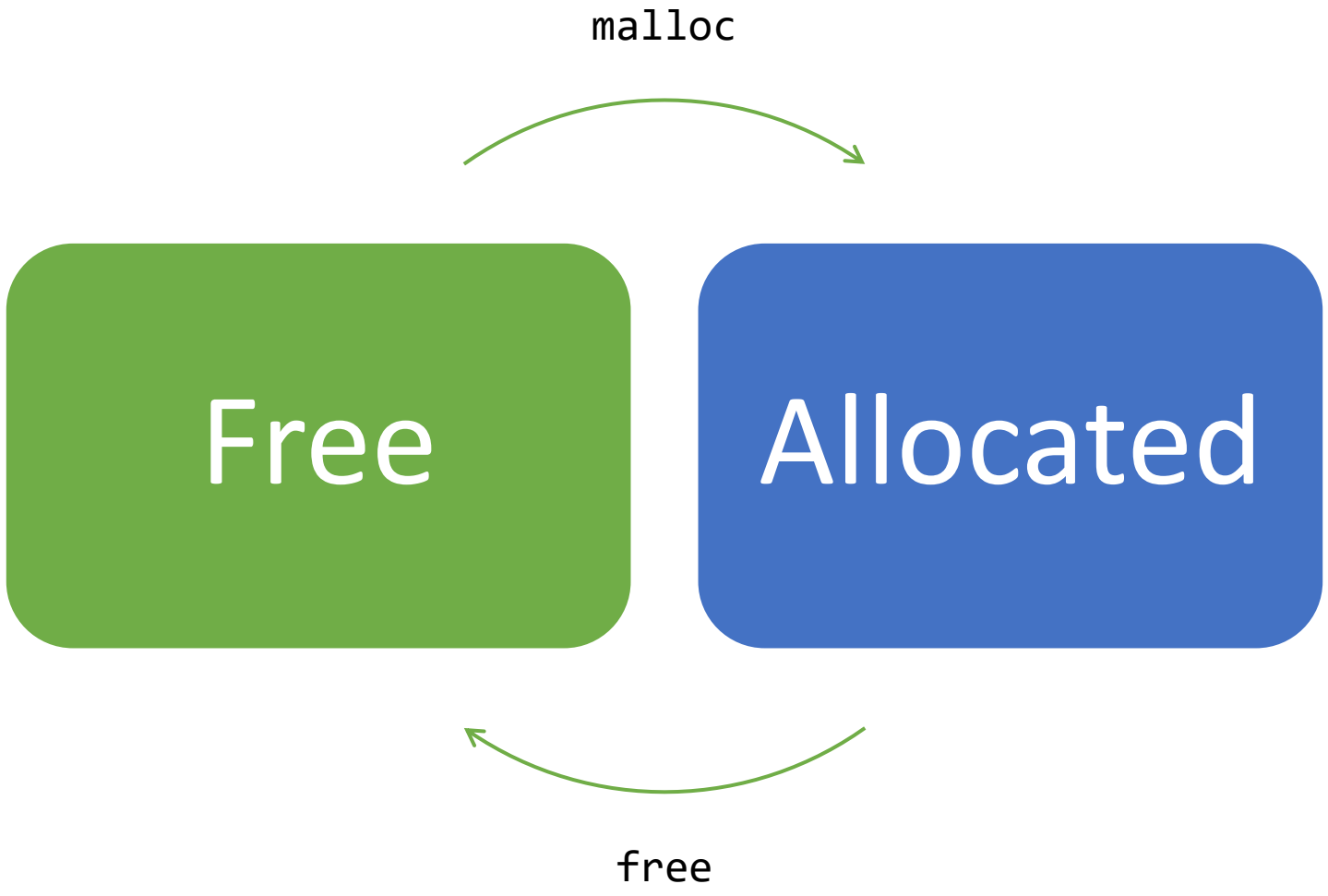
Capability bits are precious; can't obfuscate, but can *seal* (not yet done) and can still MAC

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

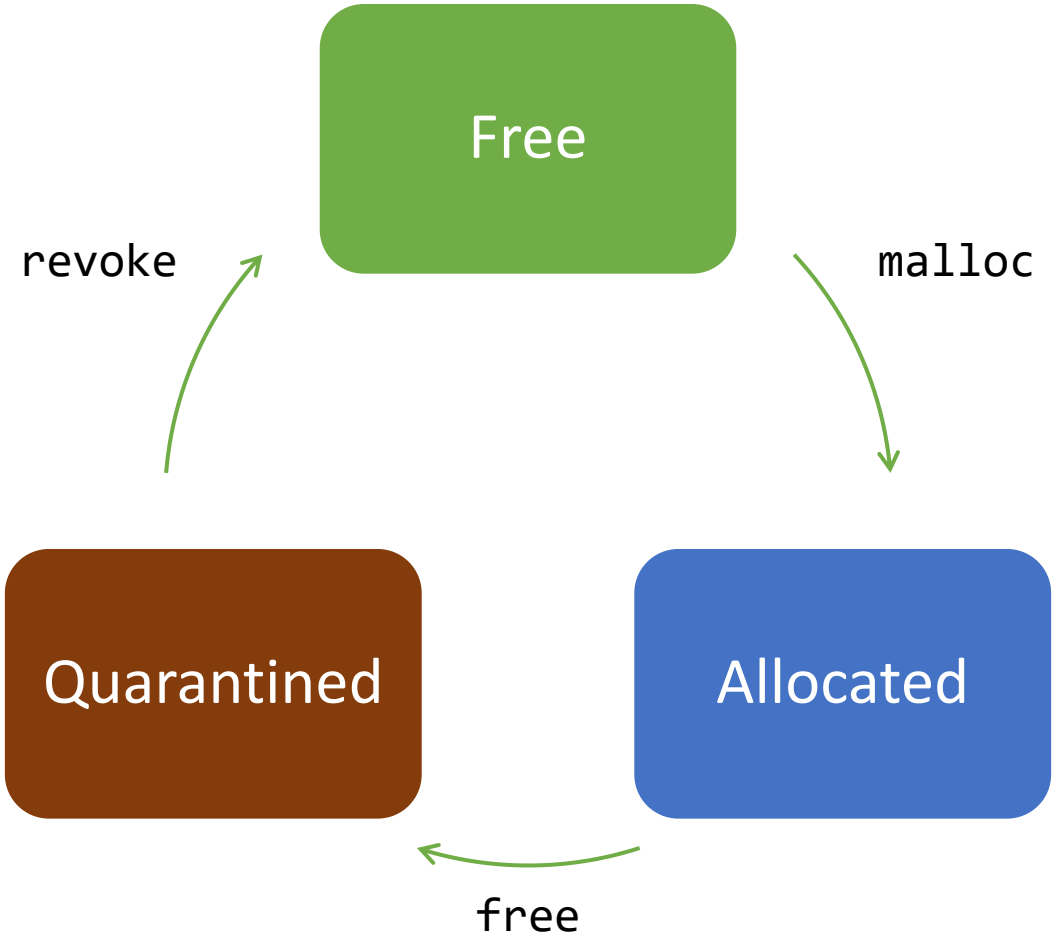
Cornucopia, Take 2

CHERI heap temporal safety

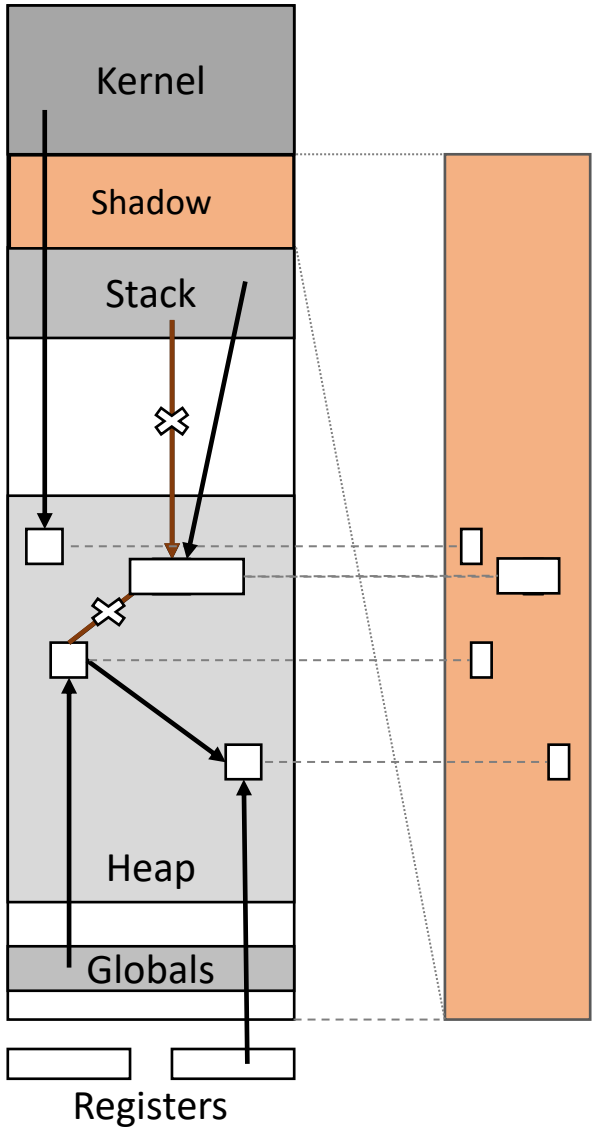
Address-space quarantine



Address-space quarantine

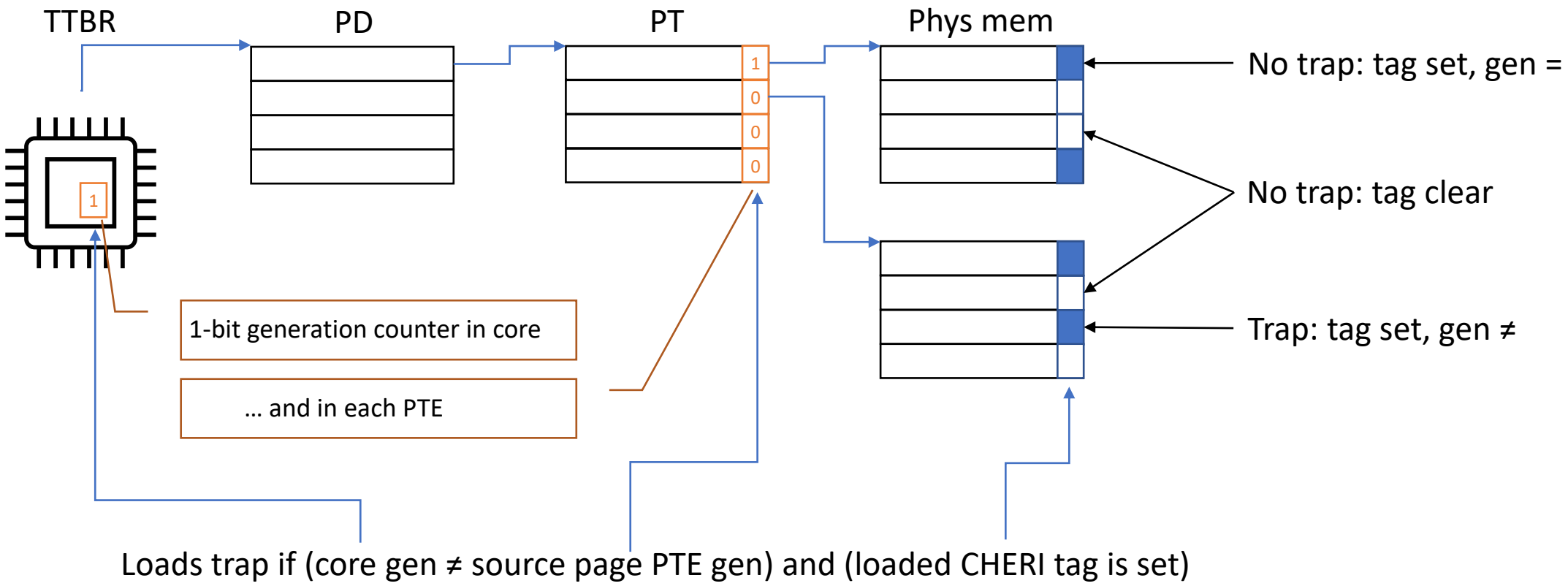


Cornucopia quarantine & revocation

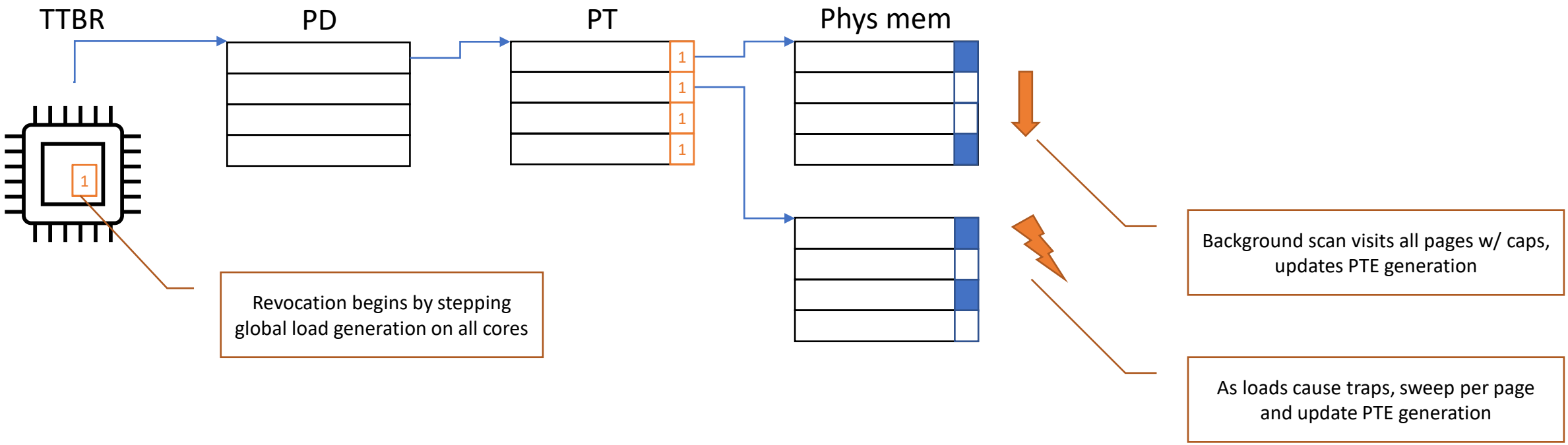


- Application `free()`-s object, might retain references.
- Express quarantine by painting *shadow* bitmap
  - Live and free objects have 0 shadow bits.
- Eventually, ask *kernel* to revoke stale caps
  - Sweep AS & remove caps w/ base address shadow bit set
- After revocation, stale caps gone,
  - Now safe to clear shadow bits, &
  - re-issue *unaliased* address space!

New architecture
Per-page capability load generations



Revoking with capability load generations



Threat assessment w/ CHERI & cornucopia

Threat		CHERI	CHERI+Revocation
Spatial separation		Set bounds	←
Temporal aliasing		Randomized free queues	Quarantine & revocation
Information disclosure		0 on alloc	0 on de-quarantine
Metadata access or corruption	Out-of-band	Capability reachability	←
	In-band	Seal* & MAC	Reuse only after revocation
Incorrect free		↑ & (opt-in) ptr check	Interlocks w/ quarantine

Address space quarantine & revocation eliminates dangling pointers

Zeroing post quarantine implies 0 at alloc, but leaves quarantine full of junk.

Quarantine tracked out-of-band; metadata in-band *only once unaliased*

At entry to quarantine: pointer validation & atomic claim of AS

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

Very early revocation benchmarks

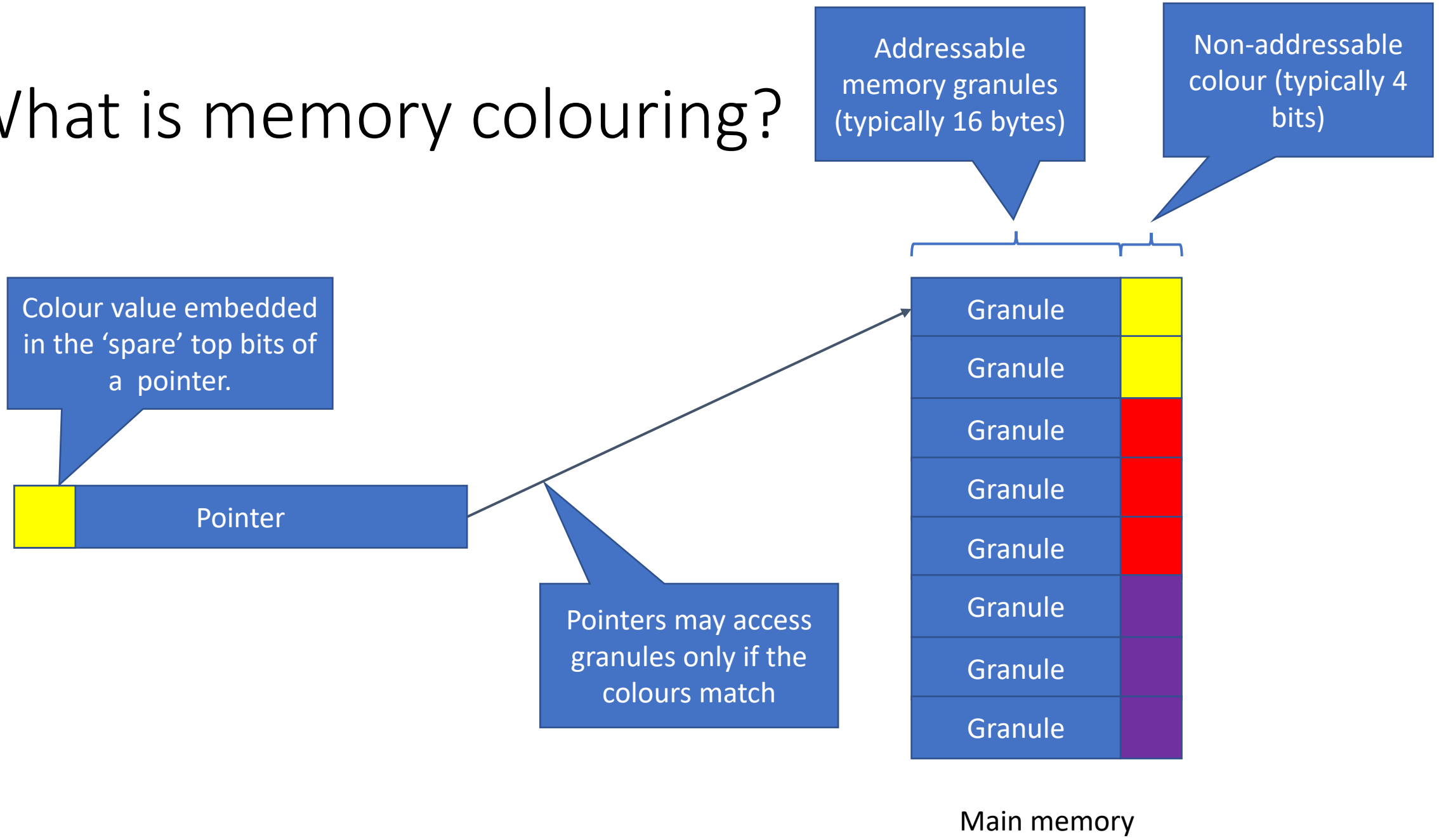
- *An unoptimized implementation, no statistical power; do not quote!*
- SPEC CPU2006 on Morello w/ load generations

Wall time	**gobmk 13x13**	**astar BigLakes2048**	**omnetpp**	**xalancbmk**
Single core	0.69%	1.7%	24%	23%
Revocation offload (SMP)	0.44%	1.1%	12%	20%

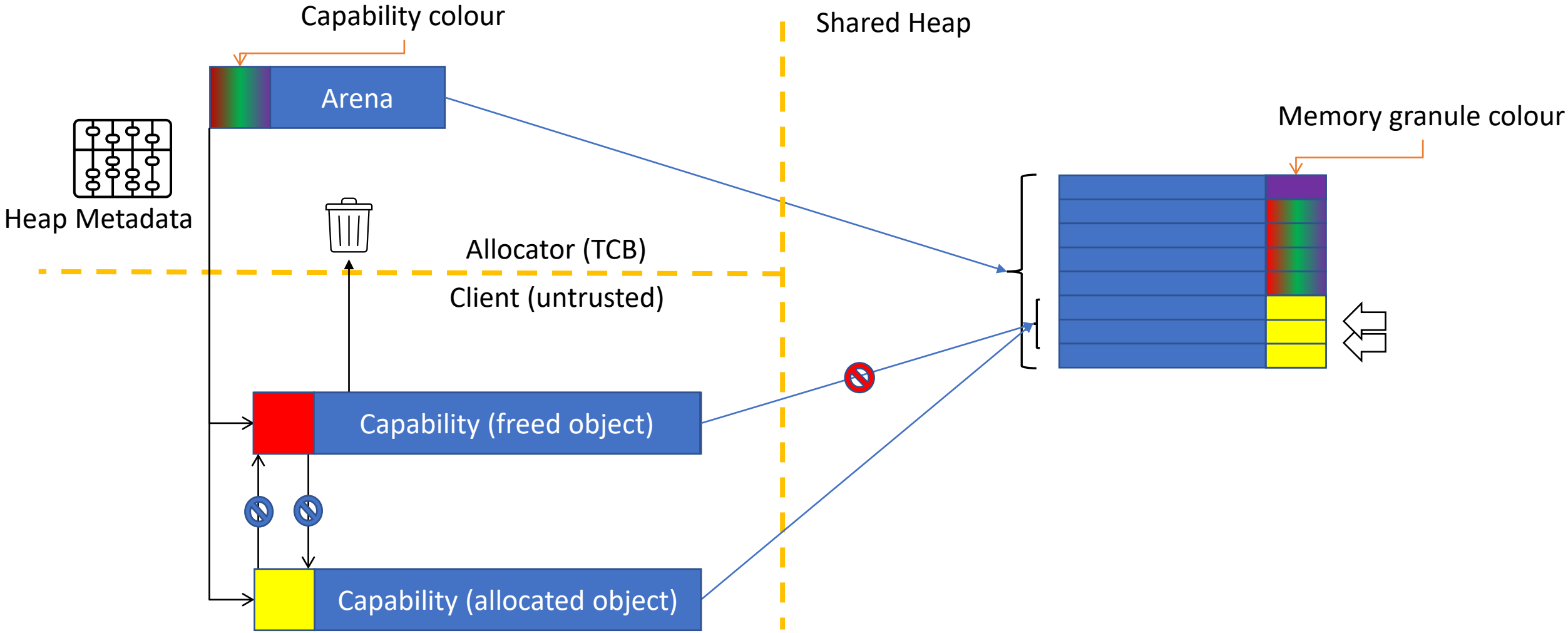
CHERI+MTE

Memory colouring for faster and better temporal safety

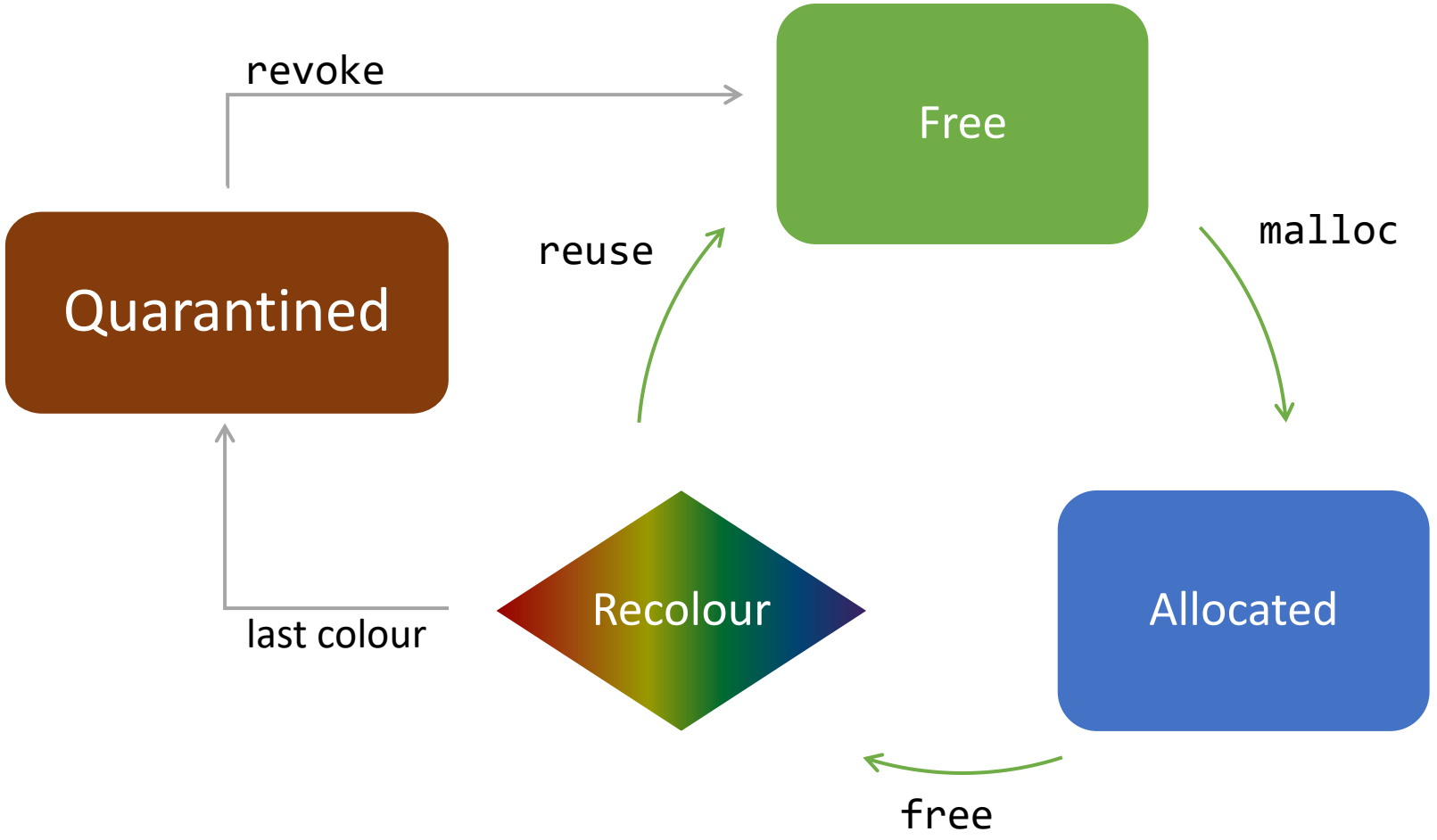
What is memory colouring?



Beyond Morello: non-orthogonal CHERI+MTE in heaps



Colouring & Revocation



snmalloc CHERI+MTE threat assessment

Threat		CHERI+Revocation	CHERI+Rev+MTE
Spatial separation		Set bounds	←
Temporal aliasing		Quarantine & revocation	Recolour & ←
Information disclosure		0 on de-quarantine	0 on free
Metadata access or corruption	Out-of-band	Capability reachability	←
	In-band	Reuse only after revocation	Reuse only after recolouring
Incorrect free		Interlocks w/ quarantine	Interlocks w/ recolouring

Recolouring reduces quarantine pressure

Zero-on-free combines w/ recolouring, clears stale caps, & is safe from client

Quarantine/free state in-band again!

Similar atomic sequence

Randomized defenses | **Deterministic w/ issues** | **Solved (!?)**

snmalloc with CHERI summary

Threat		Pre-CHERI		CHERI	CHERI+Revocation	CHERI+Rev+MTE
Spatial separation		General	Canaries	Set bounds	←	←
		`memcpy`	Checked			
Temporal aliasing		Randomized free queues		←	Quarantine & revocation	Recolour & ←
Information disclosure		0 on alloc (optional)		←	0 on de-quarantine	0 on free
Metadata access or corruption	Out-of-band	Randomized location & guard pages		Capability reachability	←	←
	In-band	Pointer obfuscation & MAC		Seal* & MAC	Reuse only after revocation	Reuse only after recolouring
Incorrect free		↑ & (opt-in) check of ptr to object start		↑ & ←'s check	Interlocks w/ quarantine	Interlocks w/ recolouring

Randomized defenses	Deterministic w/ issues	Solved (!?)

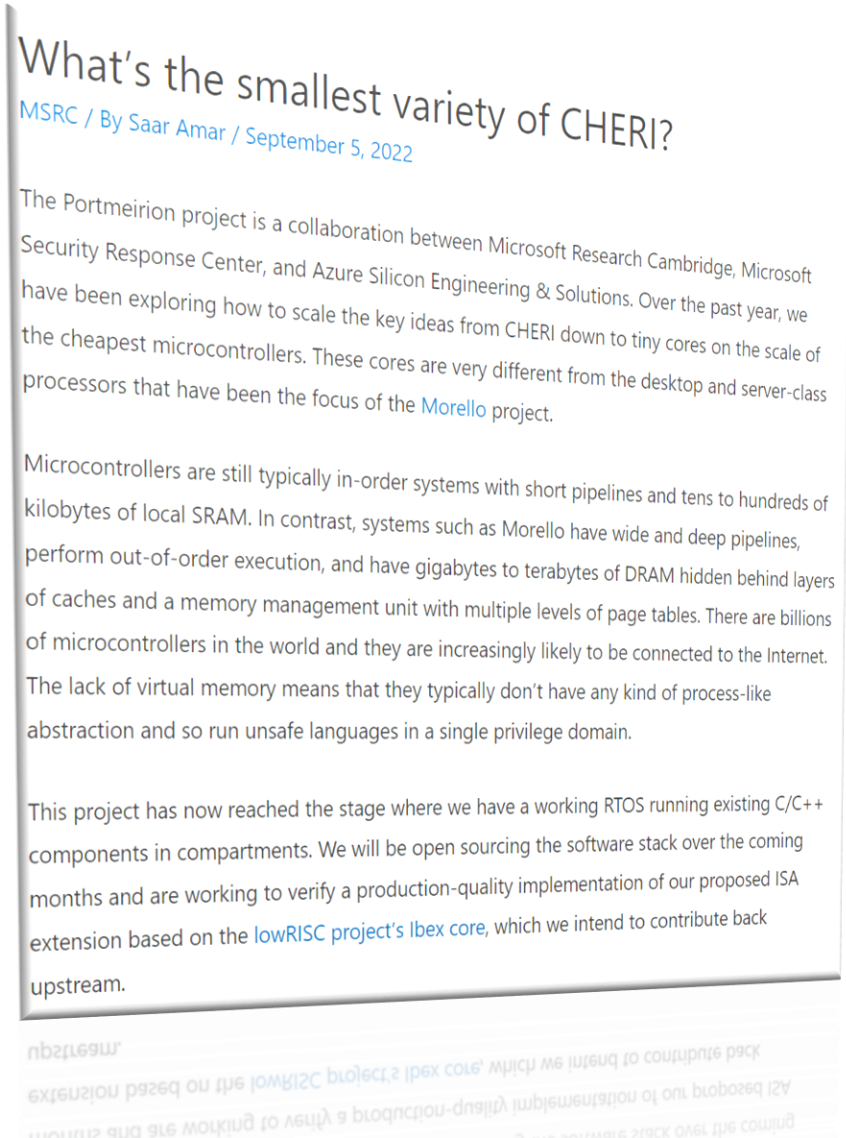
Current state of CheriBSD temporal safety

- snmalloc has baseline CHERI support (no quarantine)
  - Composes with "`mrs` wrapper" library providing a form of quarantine
  - Active work towards *integrated* quarantine

- Available for experimentation now, from source:
  - Kernel, userspace support, `mrs`, & integrated dlmalloc

- Next CheriBSD release (October) should have initial support:
  - Baseline support in userspace, bypassed on default non-Cornucopia kernels
  - 2nd, Cornucopia-enabled kernel as boot option
  - `LD_PRELOAD` malloc(s) and `mrs` wrapper as optional packages

One more thing…



What's the smallest variety of CHERI?

MSRC / By Saar Amar / September 5, 2022

The Portmeirion project is a collaboration between Microsoft Research Cambridge, Microsoft Security Response Center, and Azure Silicon Engineering & Solutions. Over the past year, we have been exploring how to scale the key ideas from CHERI down to tiny cores on the scale of the cheapest microcontrollers. These cores are very different from the desktop and server-class processors that have been the focus of the Morello project.

Microcontrollers are still typically in-order systems with short pipelines and tens to hundreds of kilobytes of local SRAM. In contrast, systems such as Morello have wide and deep pipelines, perform out-of-order execution, and have gigabytes to terabytes of DRAM hidden behind layers of caches and a memory management unit with multiple levels of page tables. There are billions of microcontrollers in the world and they are increasingly likely to be connected to the Internet. The lack of virtual memory means that they typically don't have any kind of process-like abstraction and so run unsafe languages in a single privilege domain.

This project has now reached the stage where we have a working RTOS running existing C/C++ components in compartments. We will be open sourcing the software stack over the coming months and are working to verify a production-quality implementation of our proposed ISA extension based on the lowRISC project's Ibex core, which we intend to contribute back upstream.

https://aka.ms/smallestcheri