SPAM

Stateless Permutation of Application Memory

With LLVM

Mohamed Tarek Ibn Ziad, Miguel A. Arroyo, Simha Sethumadhavan

Columbia University

10/8/20

Columbia | Engineering

The Fu Foundation School of Engineering and Applied Science

About us

Miguel A. Arroyo

5th year PhD Candidate

🐦 @miguelaarroyo12

https://miguel.arroyo.me

Mohamed Tarek

4th year PhD Candidate

🐦 @M_TarekIbnZiad

https://cs.columbia.edu/~mtarek

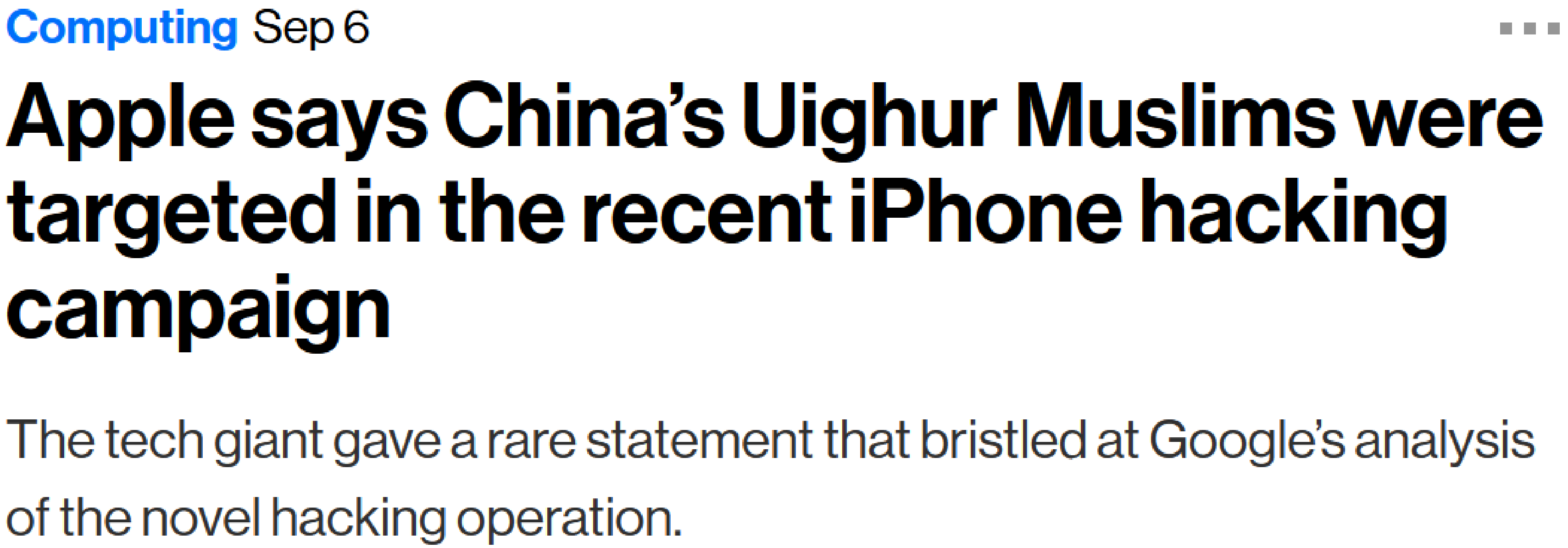
Memory Safety is a serious problem!

Computing Sep 6

Apple says China’s Uighur Muslims were targeted in the recent iPhone hacking campaign

The tech giant gave a rare statement that bristled at Google’s analysis of the novel hacking operation.

Memory Safety is a serious problem!



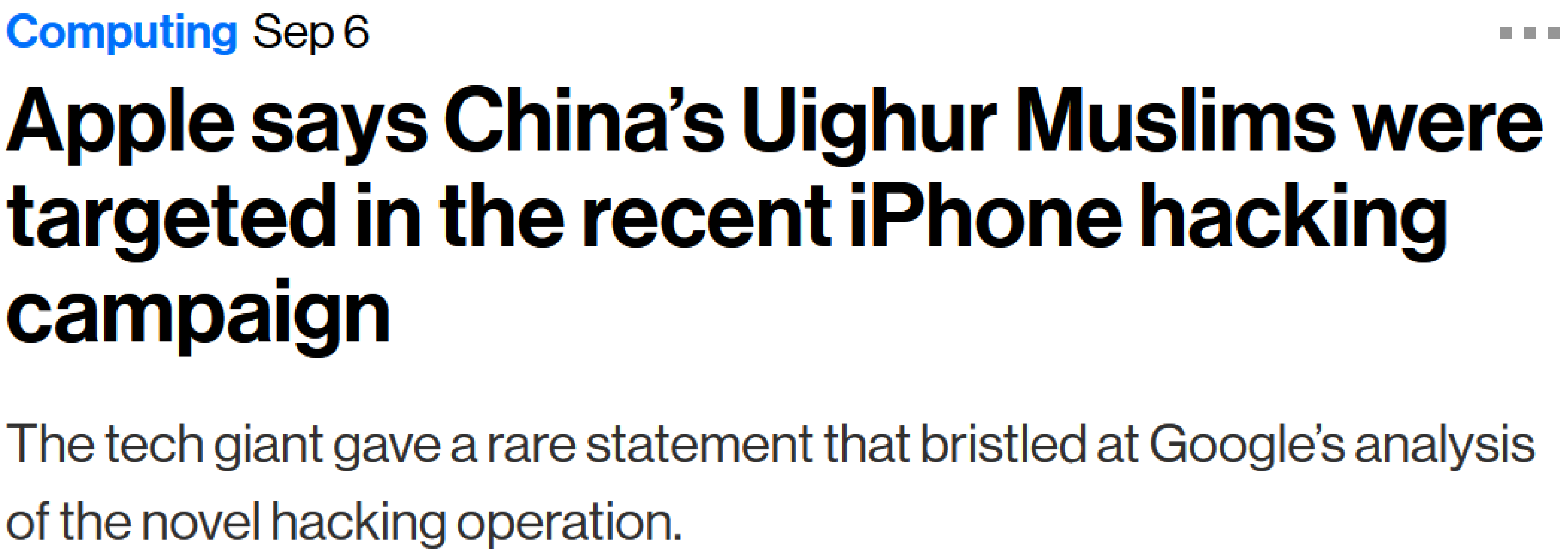
Computing Sep 6

Apple says China's Uighur Muslims were targeted in the recent iPhone hacking campaign

The tech giant gave a rare statement that bristled at Google's analysis of the novel hacking operation.

Exclusive: Saudi Dissidents Hit With Stealth iPhone Spyware Before Khashoggi's Murder

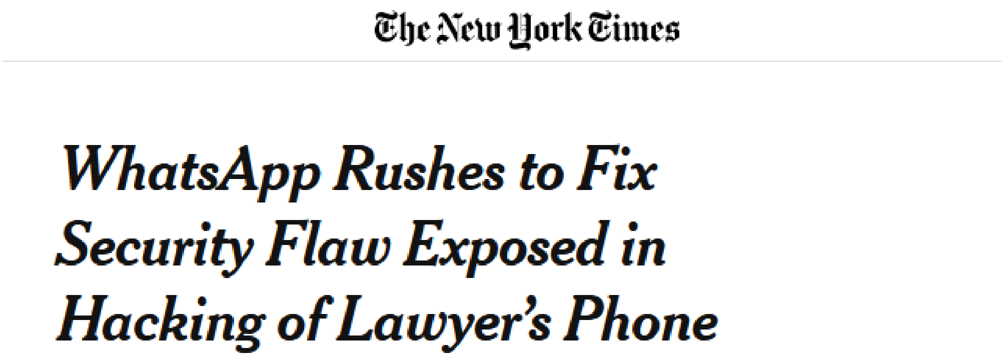
Memory Safety is a serious problem!



Computing Sep 6

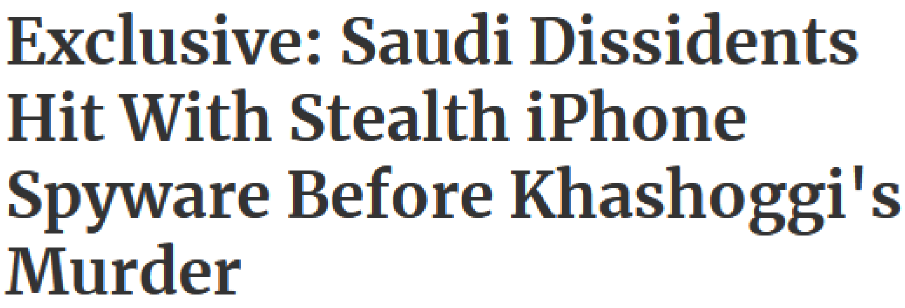
Apple says China’s Uighur Muslims were targeted in the recent iPhone hacking campaign

The tech giant gave a rare statement that bristled at Google’s analysis of the novel hacking operation.



The New York Times

WhatsApp Rushes to Fix Security Flaw Exposed in Hacking of Lawyer’s Phone

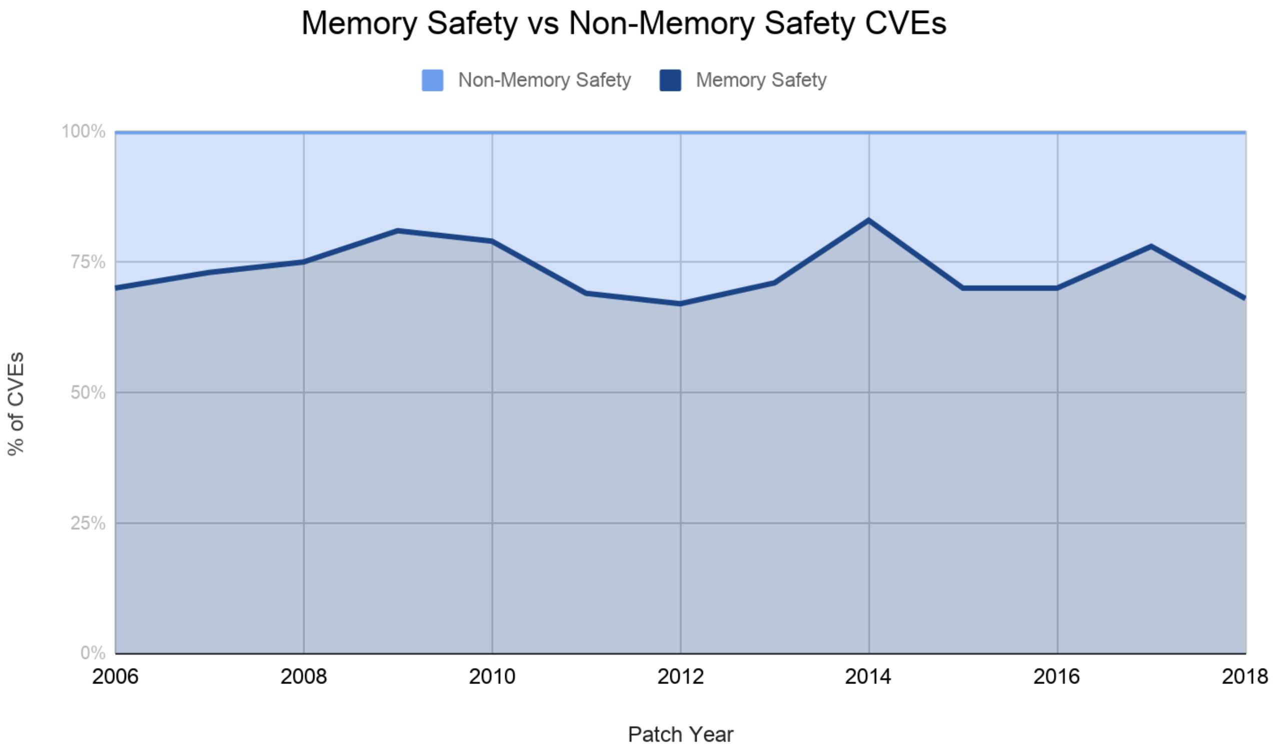


Exclusive: Saudi Dissidents Hit With Stealth iPhone Spyware Before Khashoggi's Murder

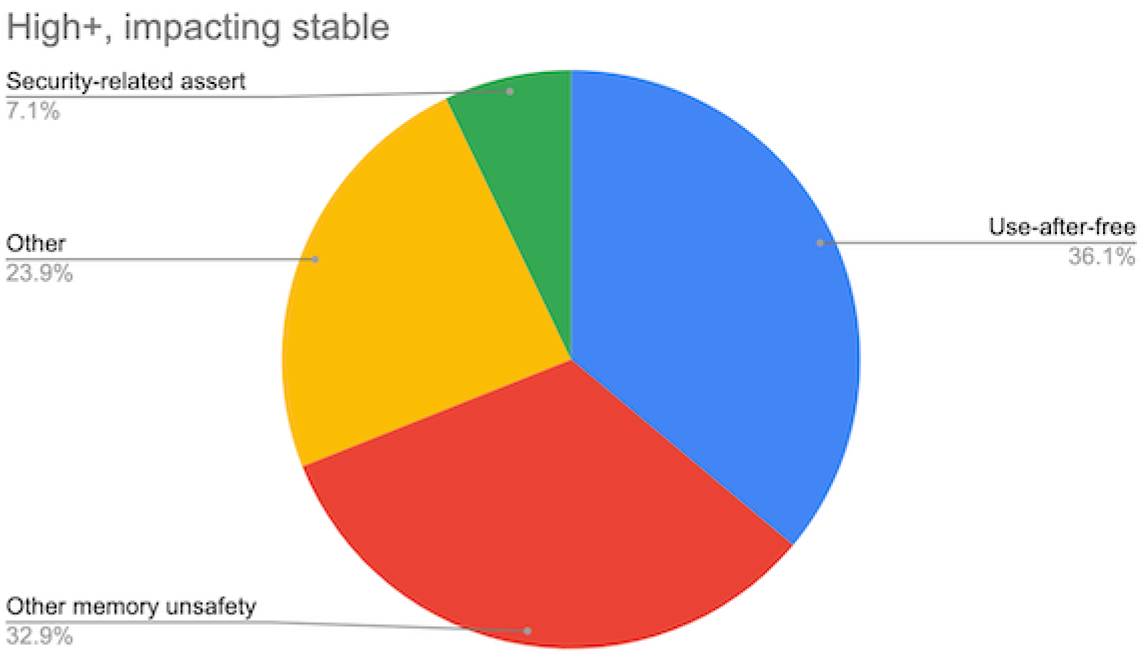
It’s easy to make mistakes

SEGFAULT!

Prevalence of Memory Safety Vulns



Microsoft Product CVEs



Google Chrome Bug Report 2015-2020

Source: *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

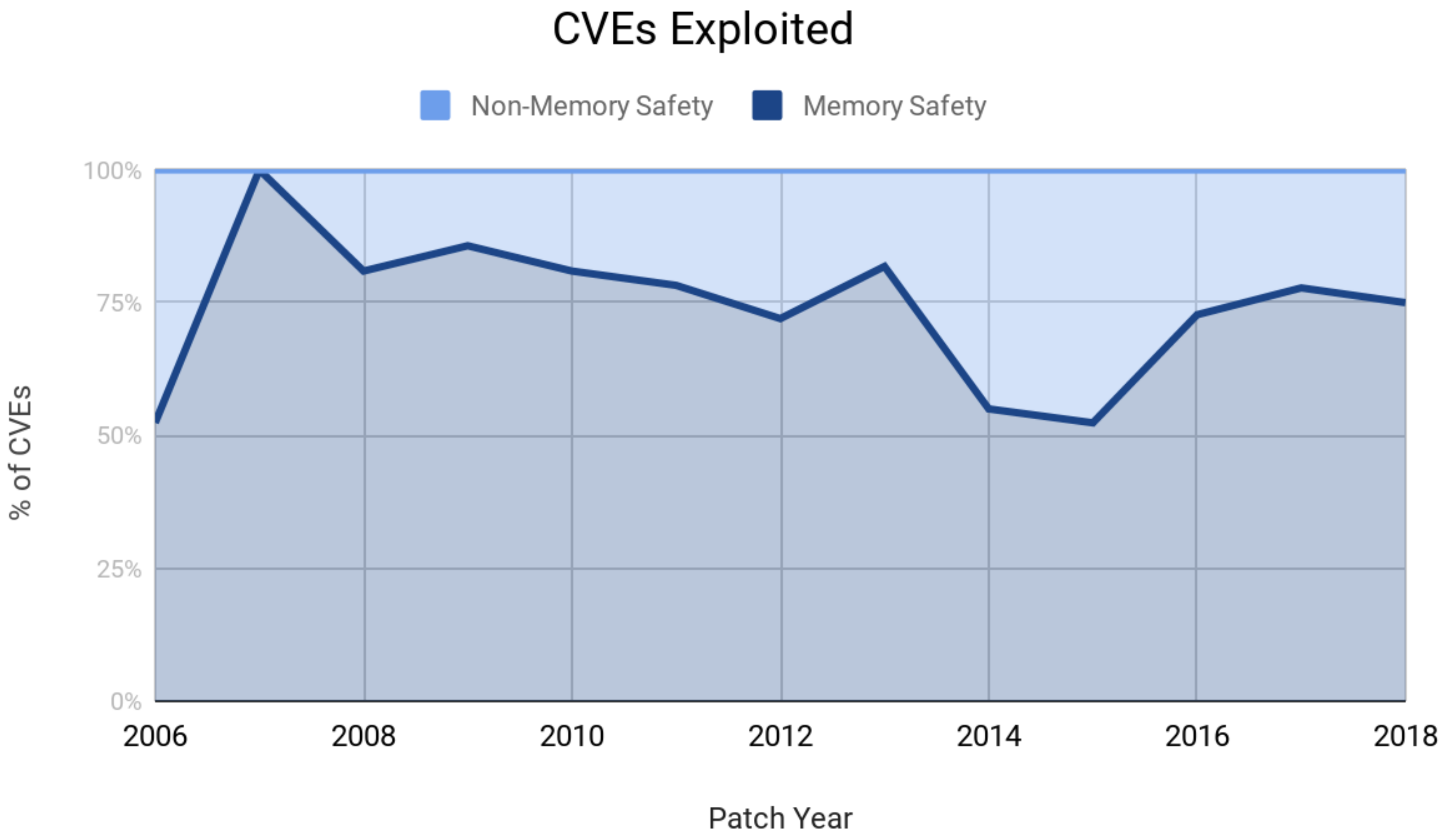
Source: *https://www.chromium.org/Home/chromium-security/memory-safety*

ATTACKERS

❤

MEMORY SAFETY

Attackers Prefer Memory Safety Vulns

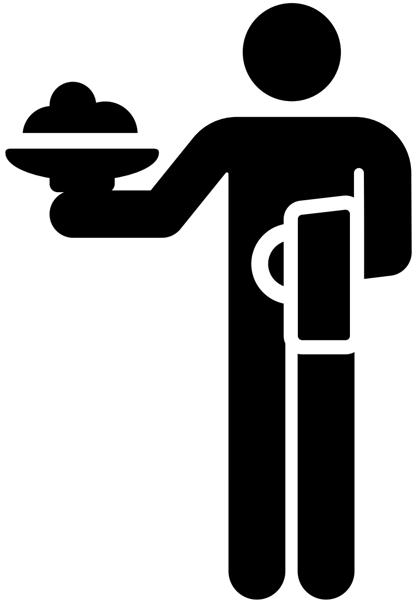


Microsoft Product Exploits

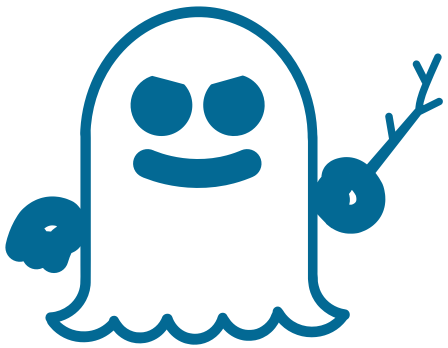
Source: *Matt Miller, Microsoft Security Response Center (MSRC) - BlueHat 2019*

À la carte solutions with additive overheads

Memory Safety Menu	Price
Intra-Object Overflow	$$$
Inter-Object Overflow	$$
Buffer-Overread	$
Control-Flow Hijack	$
Use-after-free	$$
Type Confusion	$$$
Uninitialized Reads	$$



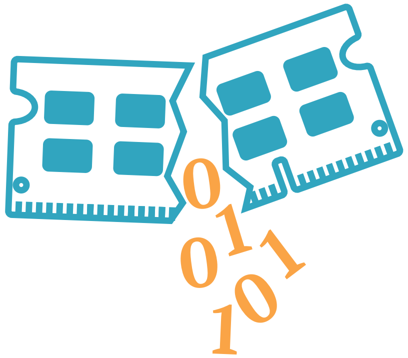
No common solution to all problems



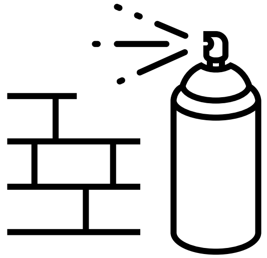
Spectre

Meltdown

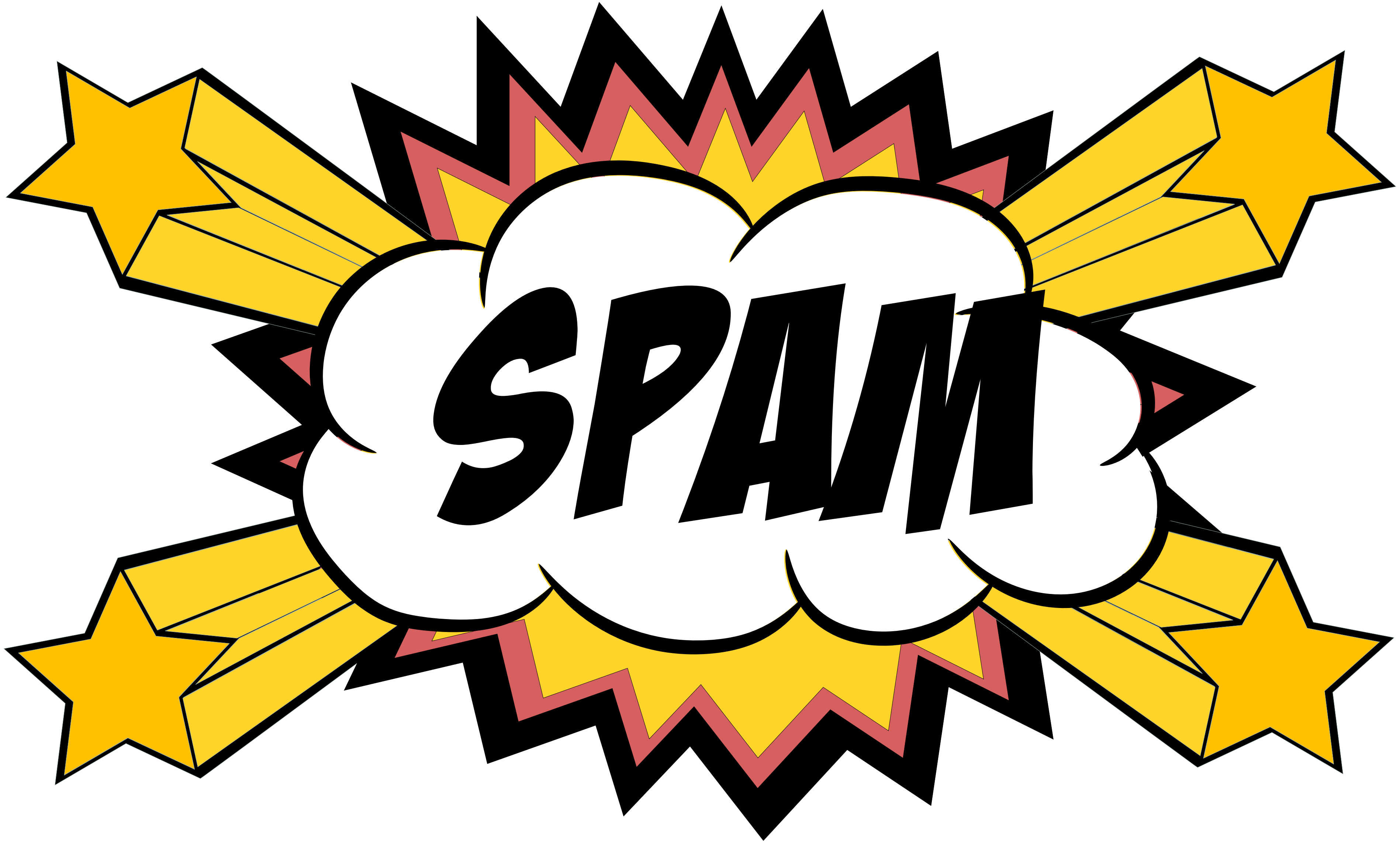
RowHammer



RamBleed



ColdBoot


SPAM

Overview

Overview

Struct Definition

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
A_t *A1 = malloc(
     sizeof(A_t));
A_t *A2 = malloc(
     sizeof(A_t));

free(A1);
A_t *A3 = malloc(
     sizeof(A_t));
```

main.c

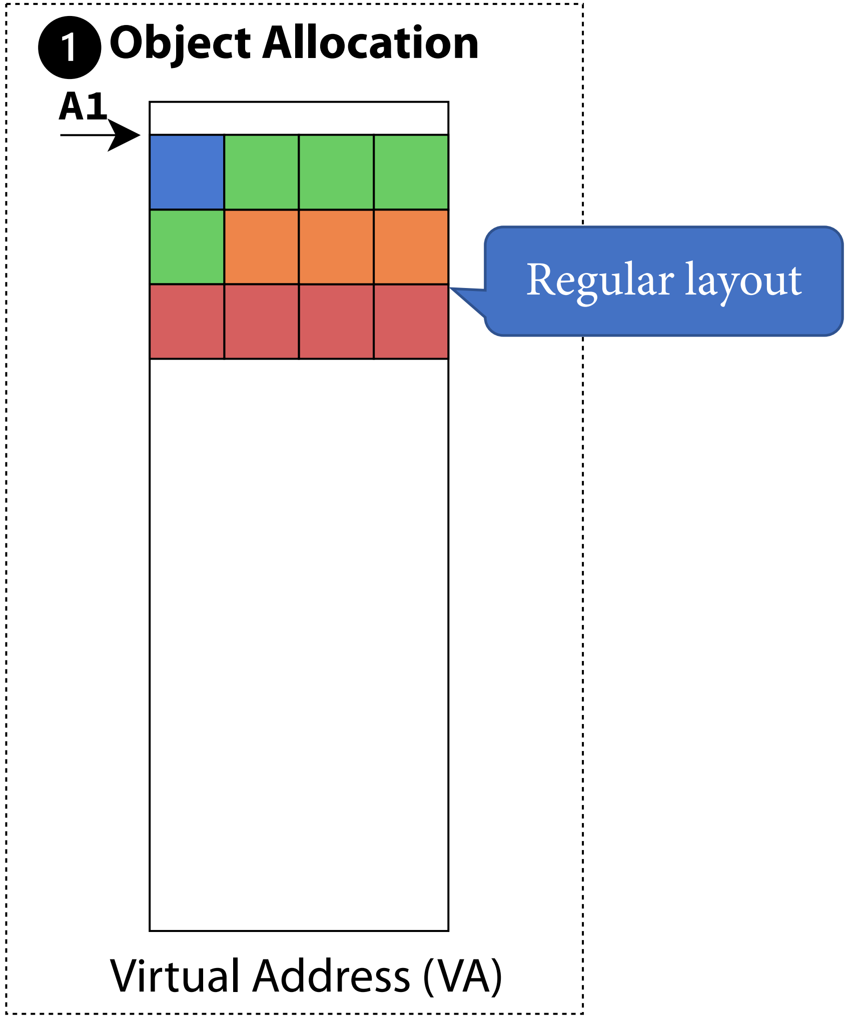
Overview

Struct Definition

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
A_t *A1 = malloc(
      sizeof(A_t));
A_t *A2 = malloc(
      sizeof(A_t));

free(A1);
A_t *A3 = malloc(
      sizeof(A_t));
```

main.c



Overview

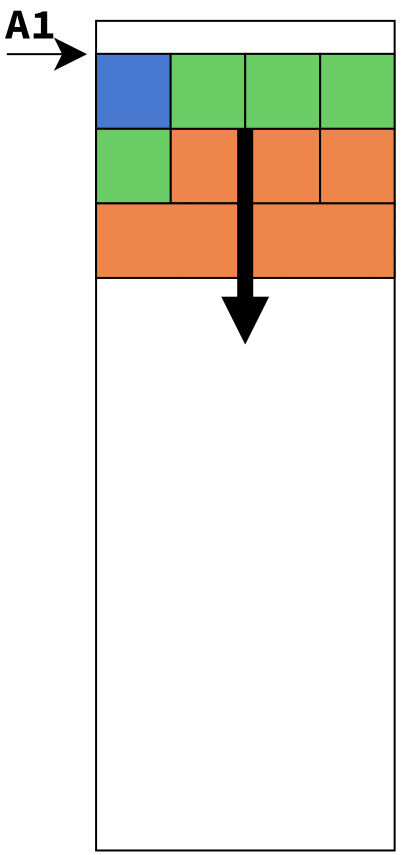
Struct Definition

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
A_t *A1 = malloc(
      sizeof(A_t));
A_t *A2 = malloc(
      sizeof(A_t));

free(A1);
A_t *A3 = malloc(
      sizeof(A_t));
```

main.c

❶ **Object Allocation**



Virtual Address (VA)

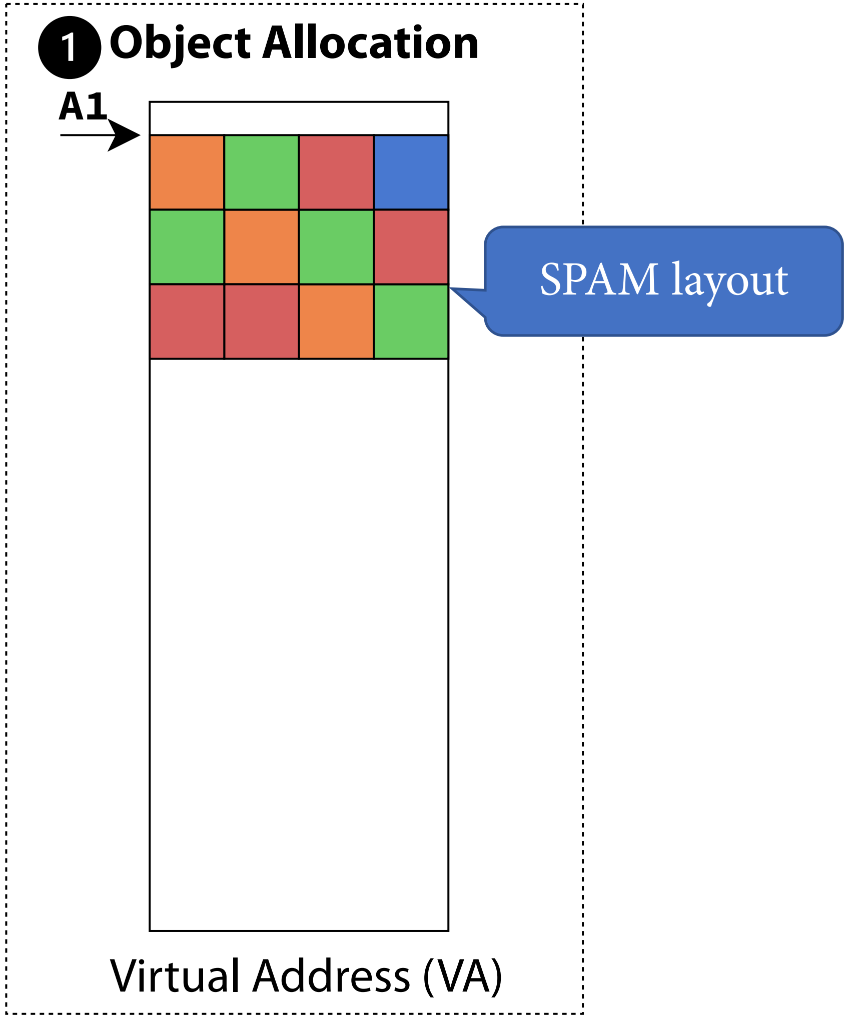
Overview

Struct Definition

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
A_t *A1 = malloc(
      sizeof(A_t));
A_t *A2 = malloc(
      sizeof(A_t));

free(A1);
A_t *A3 = malloc(
      sizeof(A_t));
```

main.c



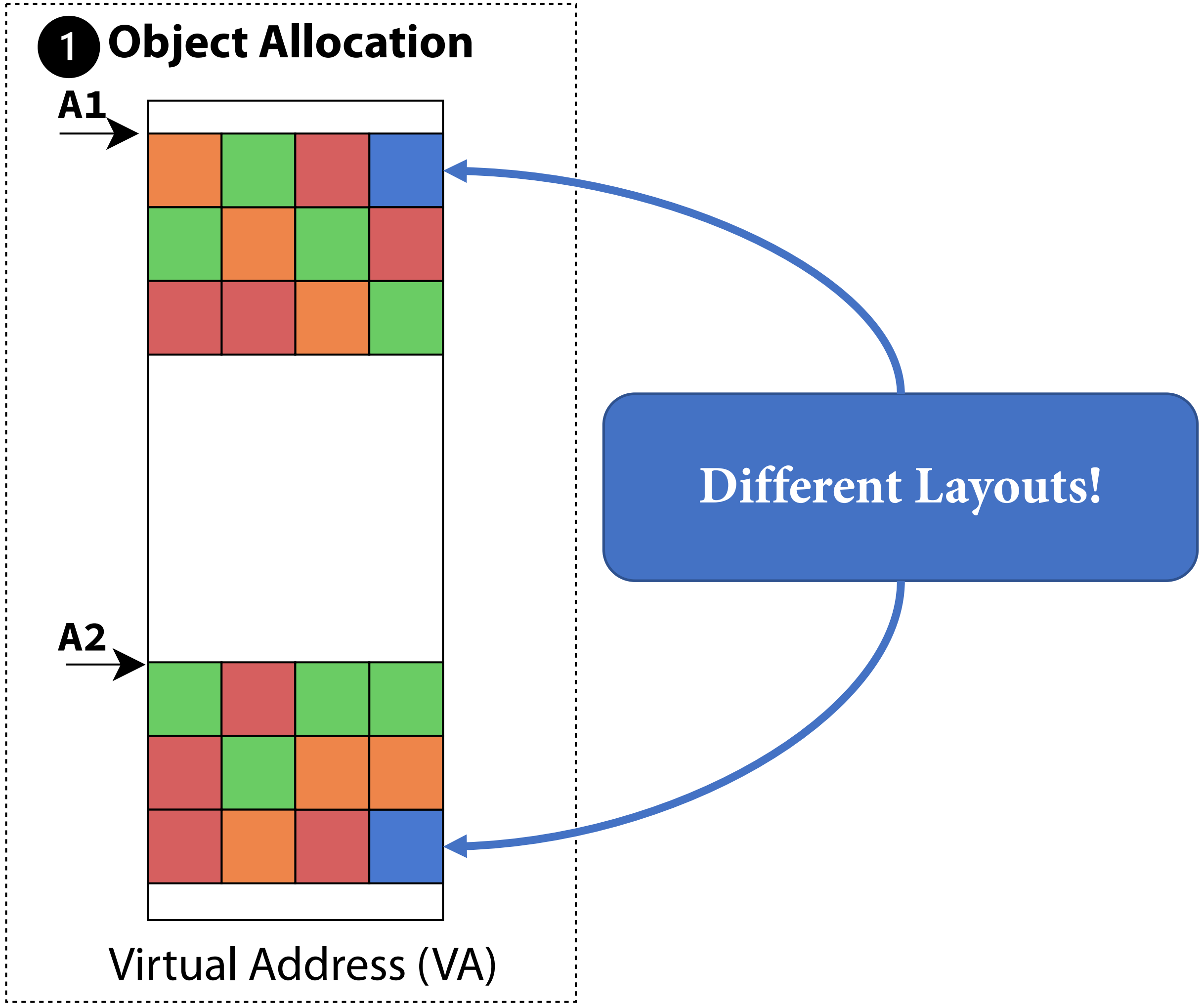
Overview

Struct Definition

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
A_t *A1 = malloc(
      sizeof(A_t));
A_t *A2 = malloc(
      sizeof(A_t));

free(A1);
A_t *A3 = malloc(
      sizeof(A_t));
```

main.c



1 Object Allocation

Virtual Address (VA)

Different Layouts!

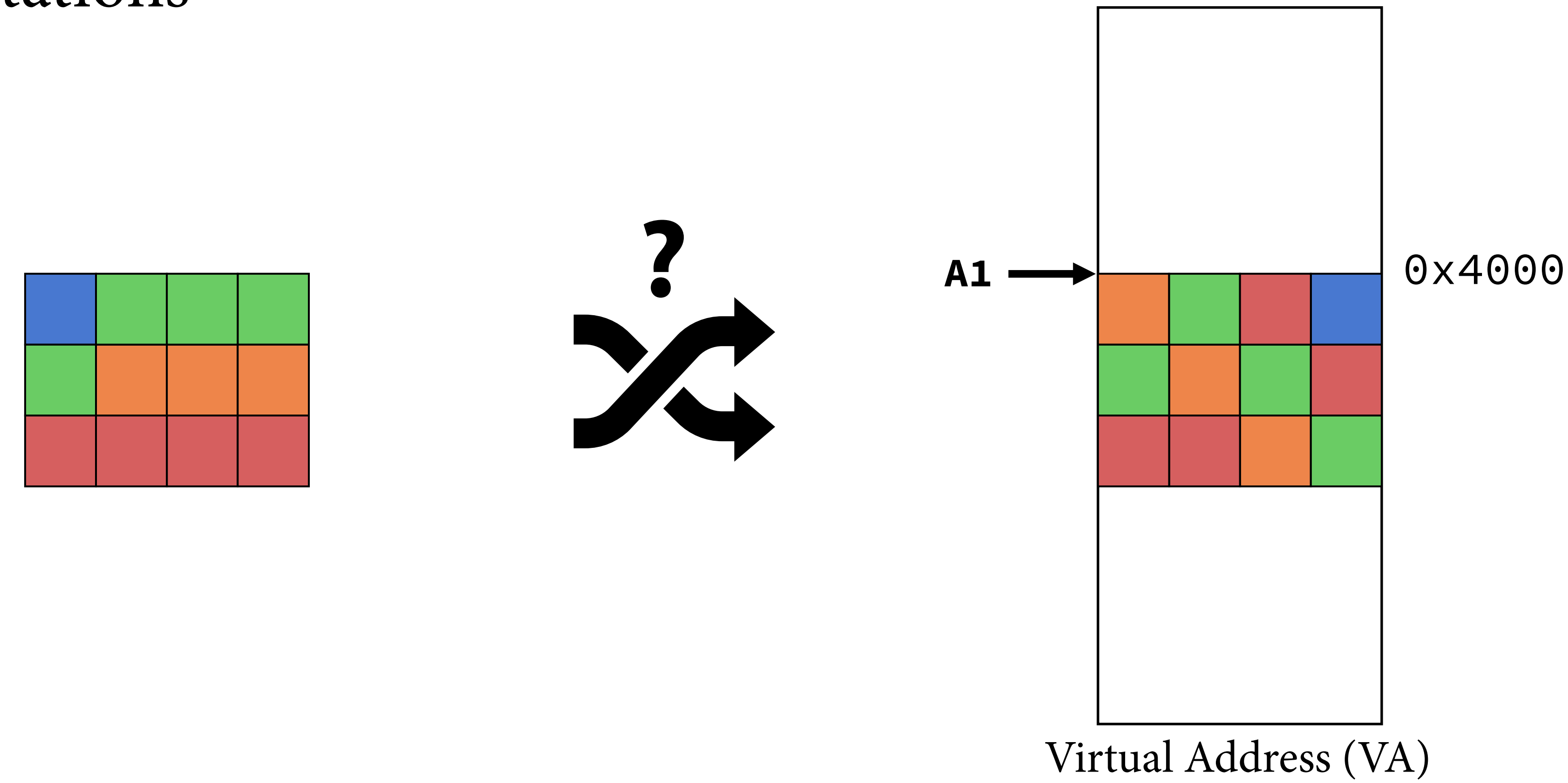
1

Object Allocation

Object Allocation

Generating Permutations

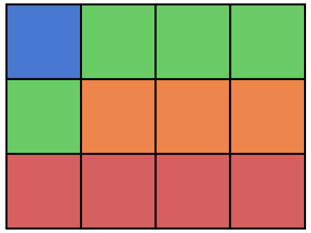
```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

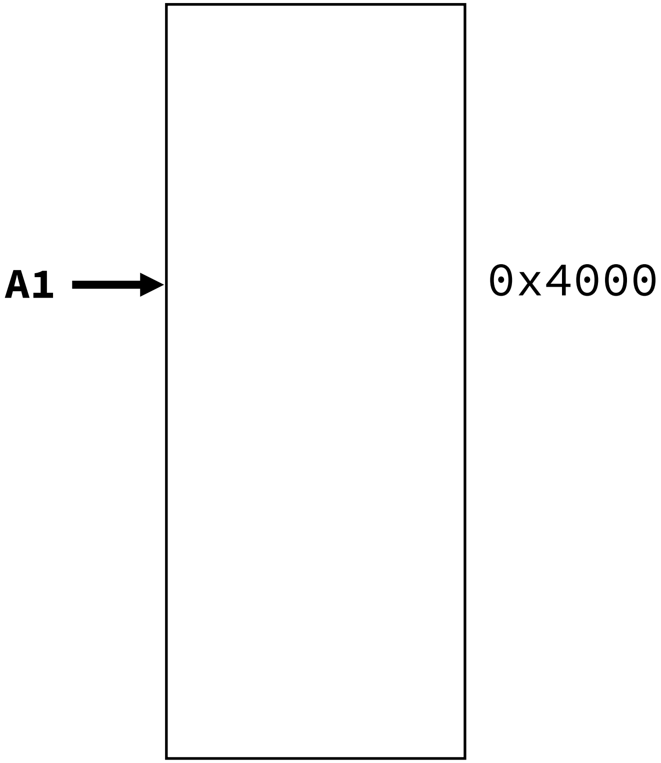


Object Allocation

Generating Permutations

```
typedef struct {
    char a;
    double b;
    char c[3];
    void (*fp)();
} A_t;
```





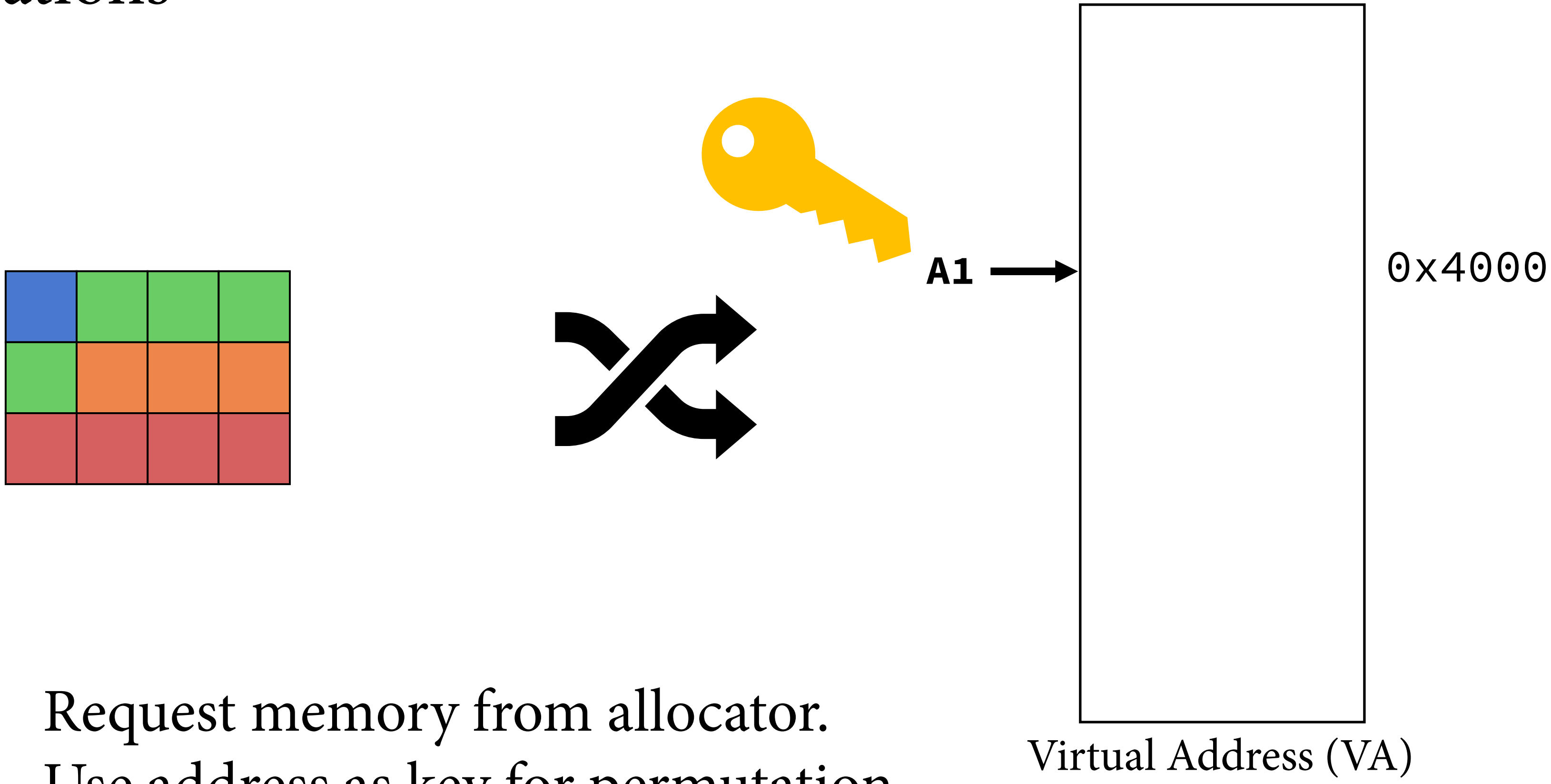
Virtual Address (VA)

1. Request memory from allocator.

Object Allocation

Generating Permutations

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

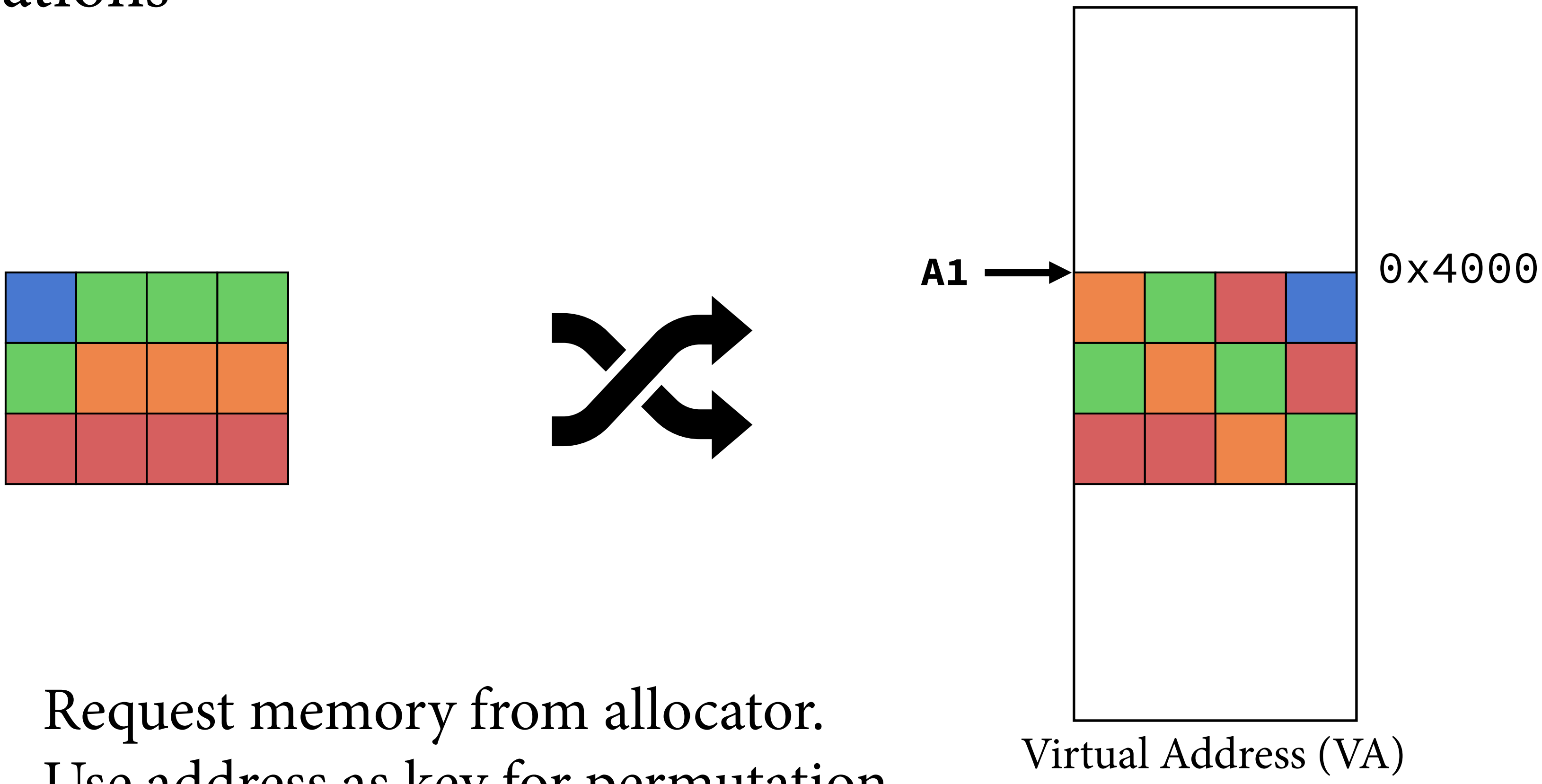


1. Request memory from allocator.
2. Use address as key for permutation.

Object Allocation

Generating Permutations

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

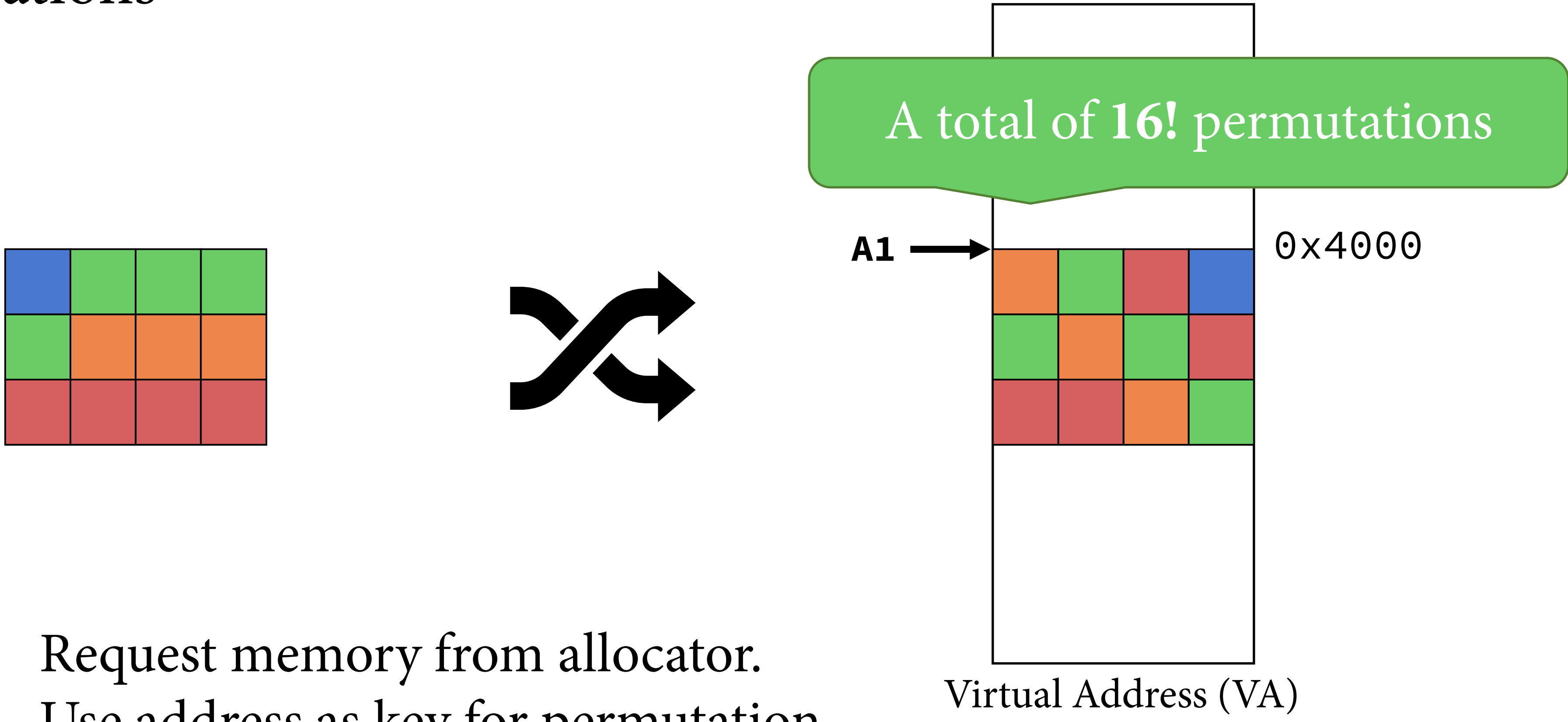


1. Request memory from allocator.
2. Use address as key for permutation.
3. Write to memory in permuted order.

Object Allocation

Generating Permutations

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```


A total of **16!** permutations

1. Request memory from allocator.
2. Use address as key for permutation.
3. Write to memory in permuted order.

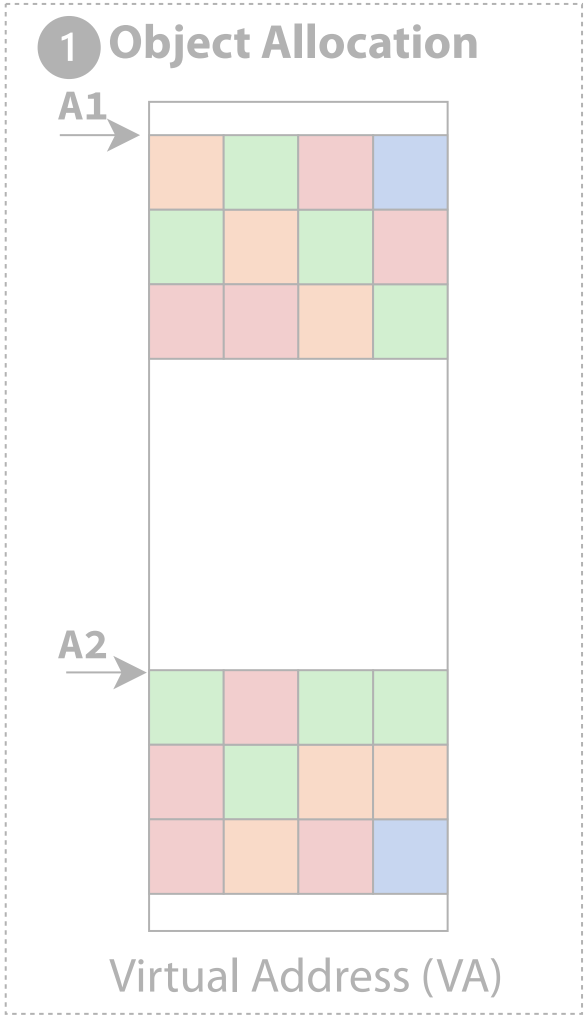
Overview

Struct Definition

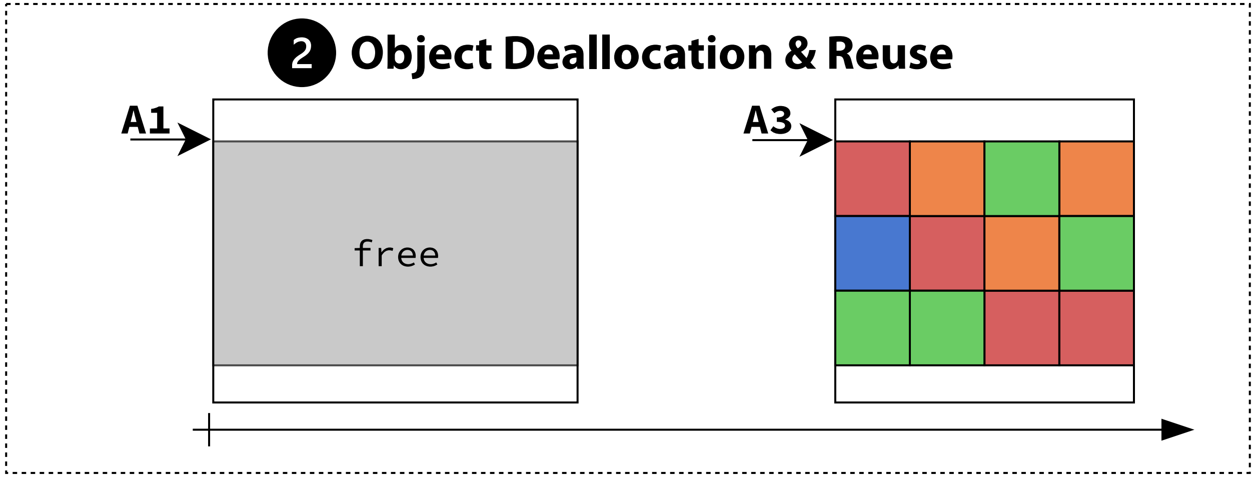
```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
A_t *A1 = malloc(
      sizeof(A_t));
A_t *A2 = malloc(
      sizeof(A_t));

free(A1);
A_t *A3 = malloc(
      sizeof(A_t));
```

main.c



1 Object Allocation

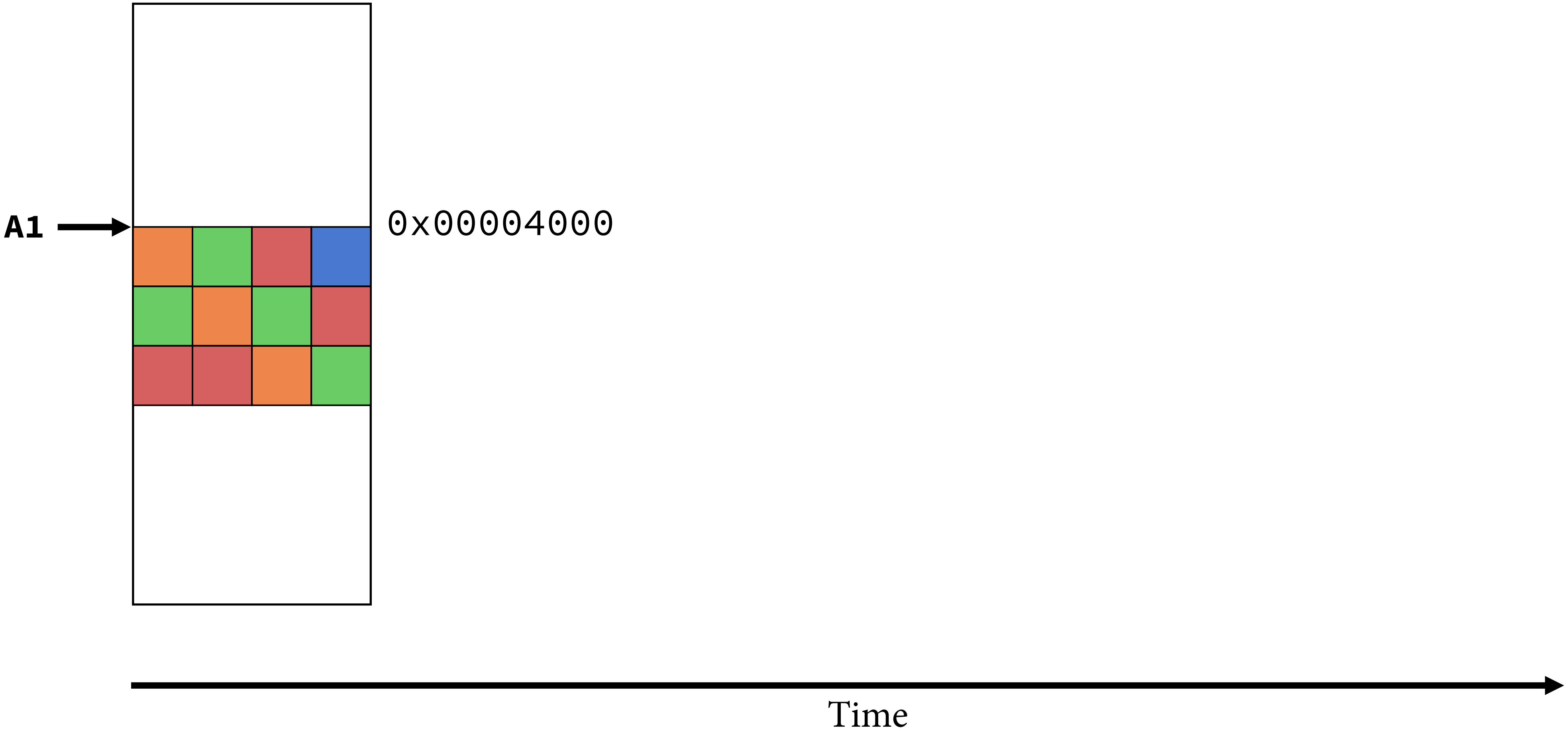


2 Object Deallocation & Reuse

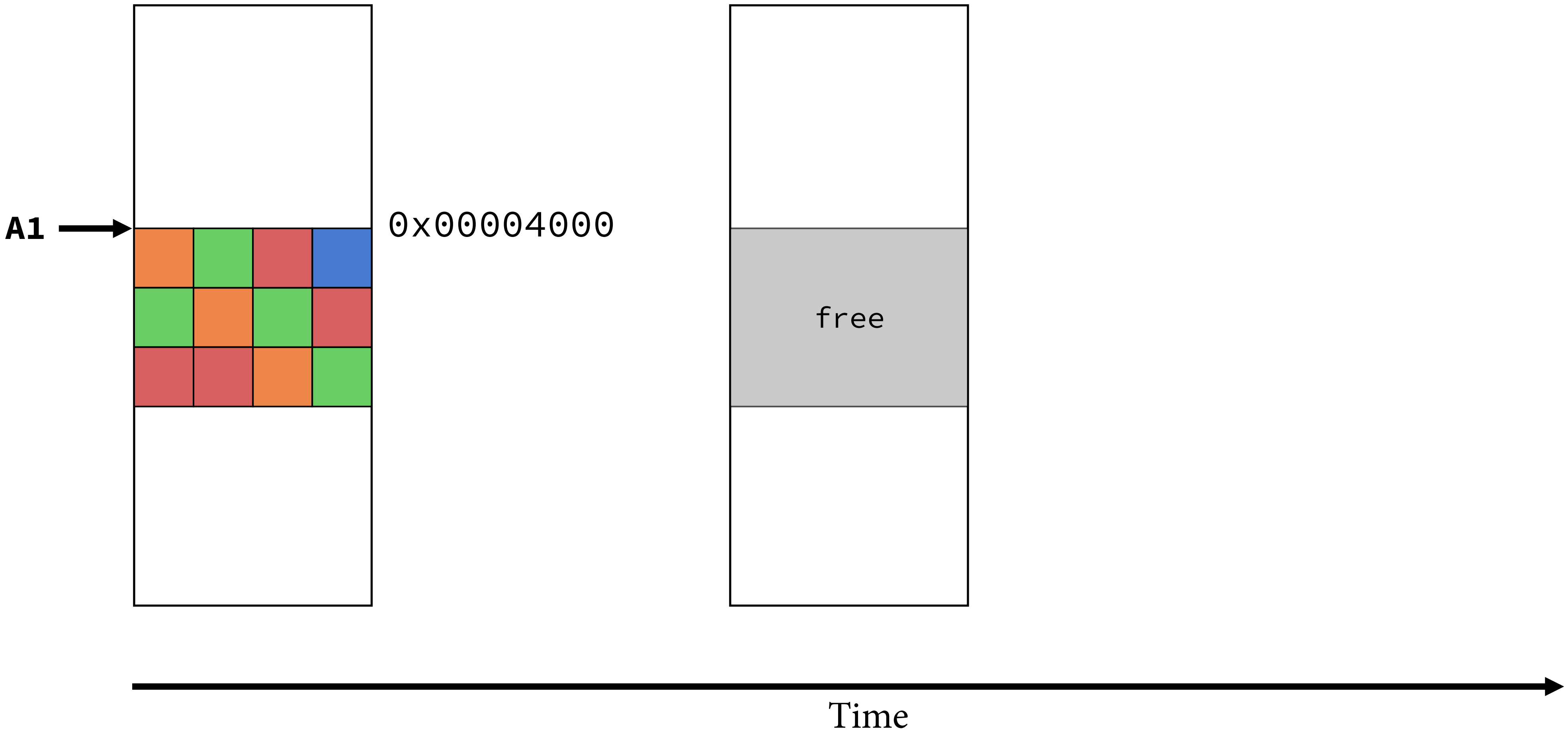
2

Object Deallocation & Reuse

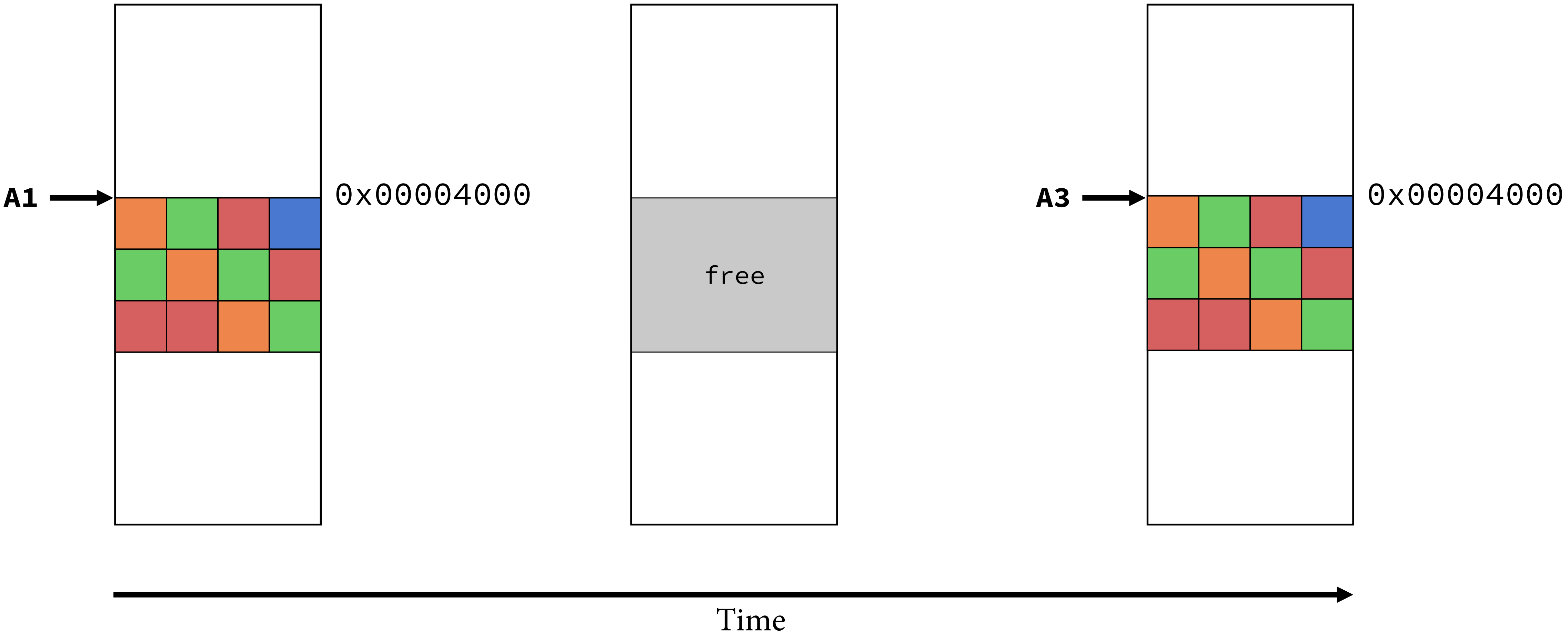
Object Deallocation & Reuse



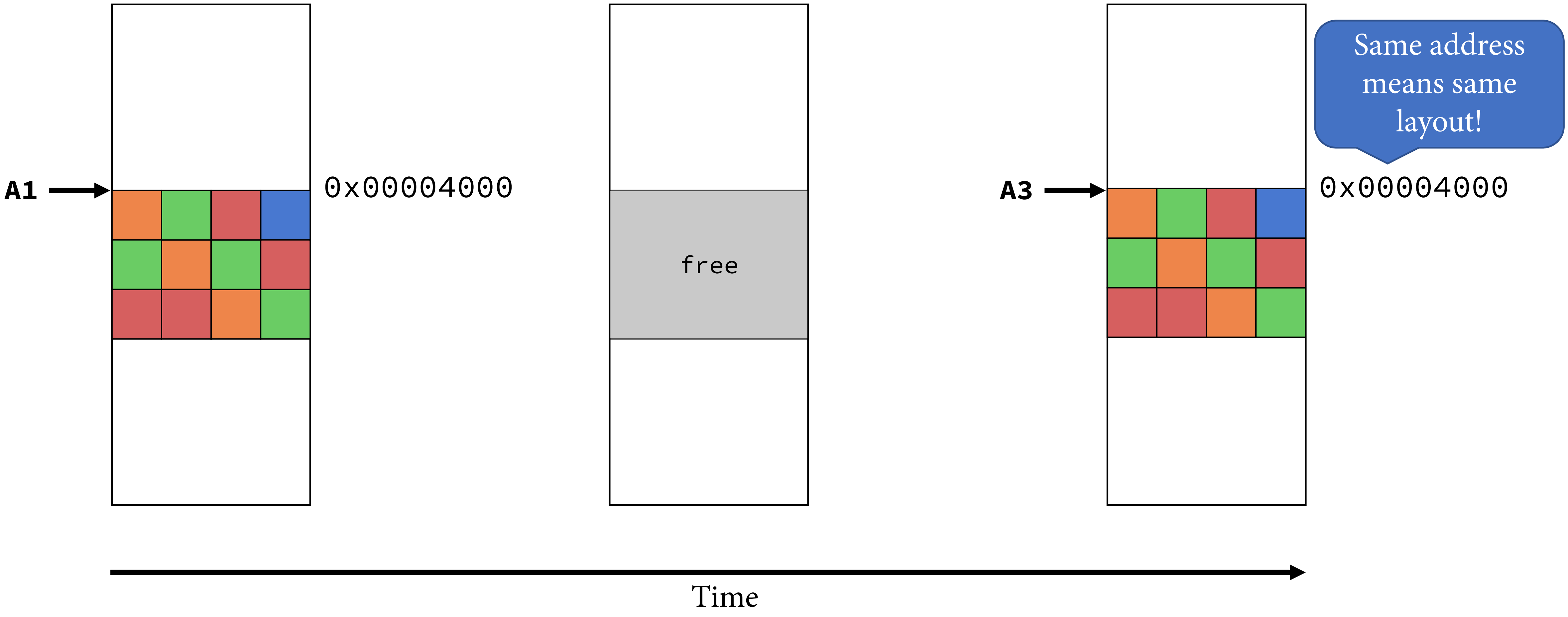
Object Deallocation & Reuse



Object Deallocation & Reuse



Object Deallocation & Reuse



A1 → 0x00004000

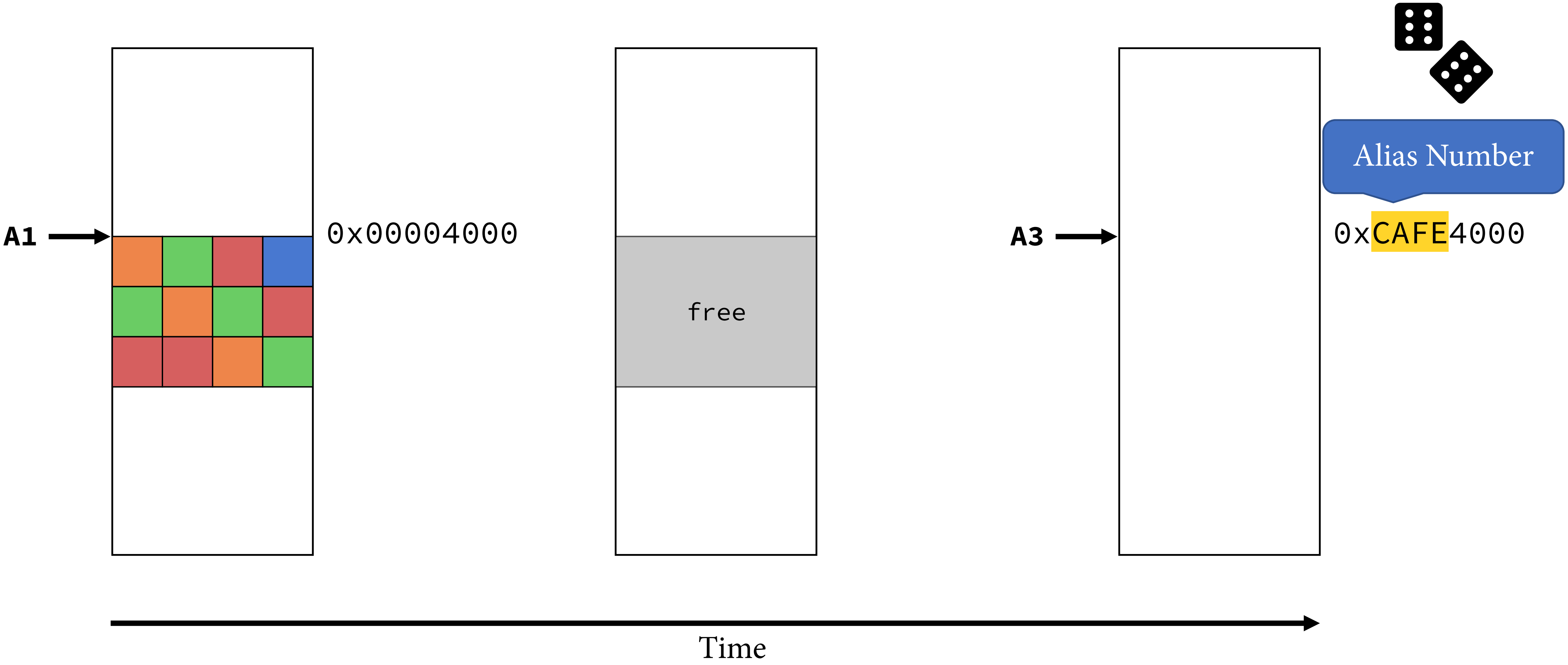
free

A3 → 0x00004000

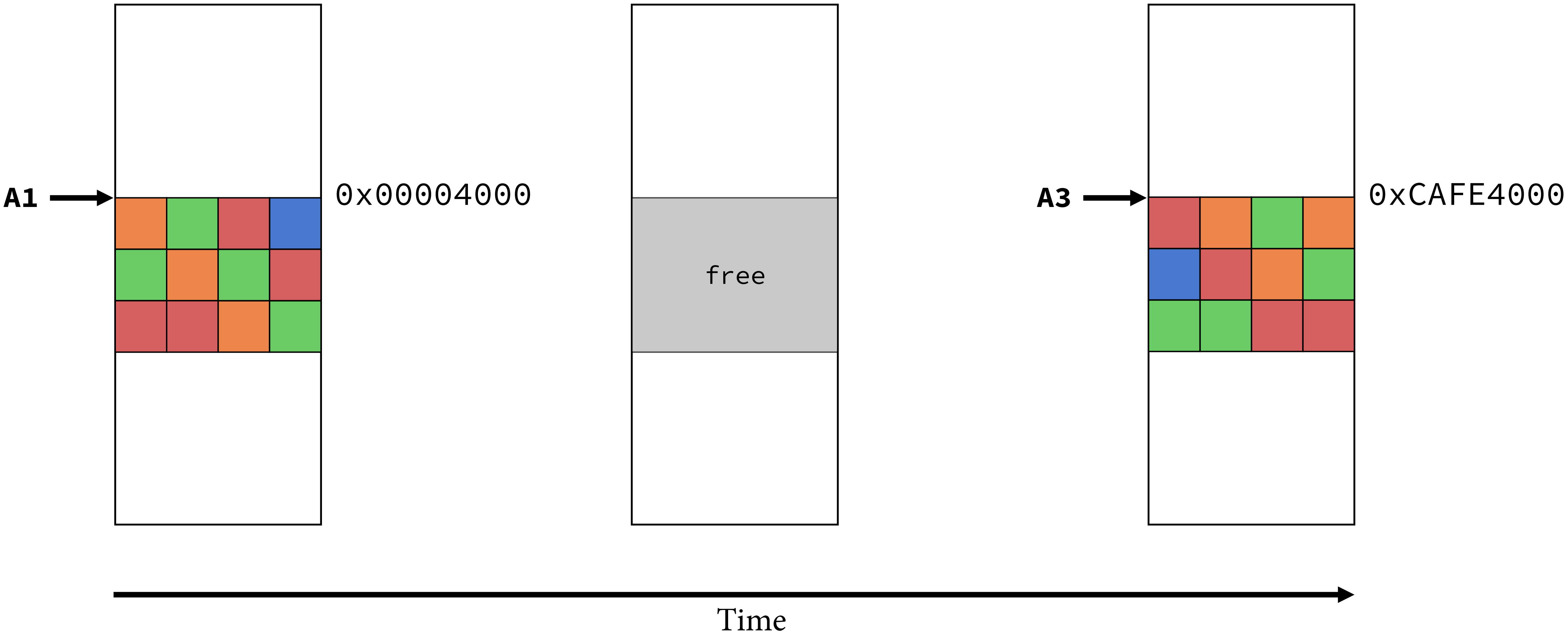
Same address means same layout!

Time

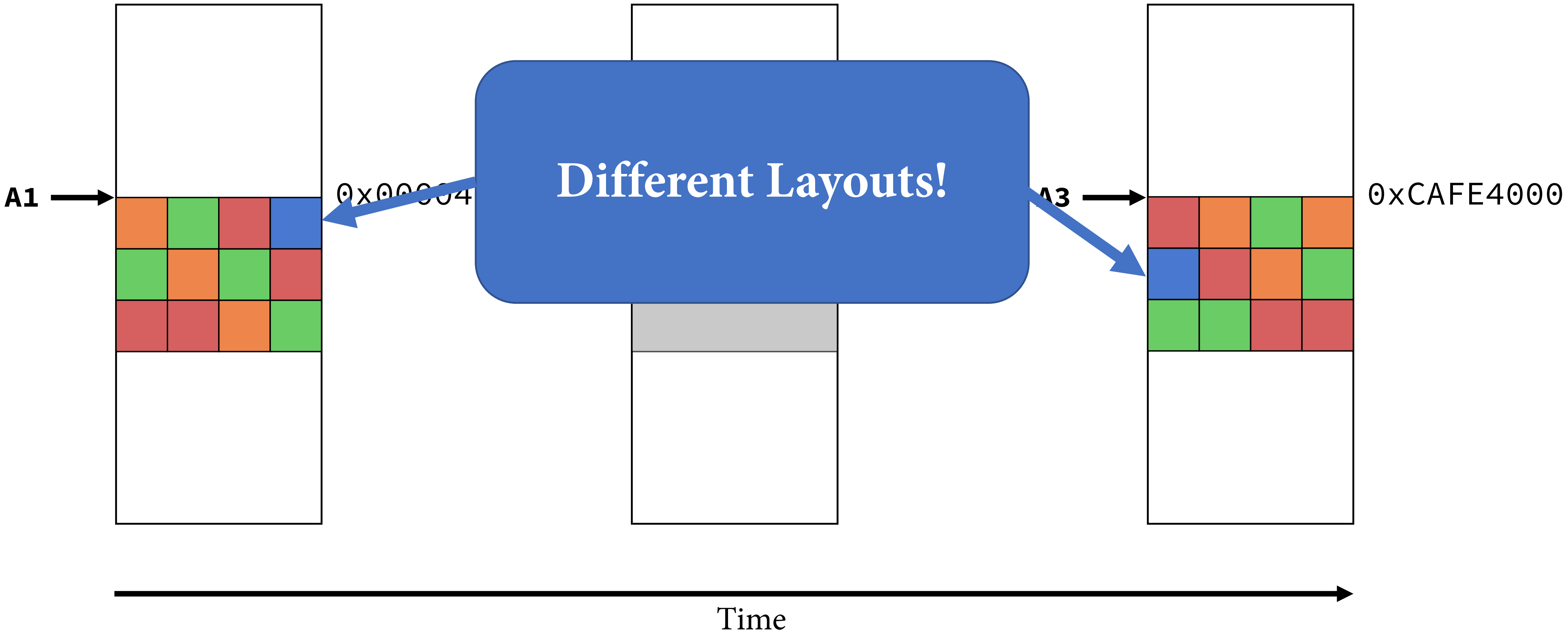
Object Deallocation & Reuse



Object Deallocation & Reuse



Object Deallocation & Reuse



Different Layouts!

Overview

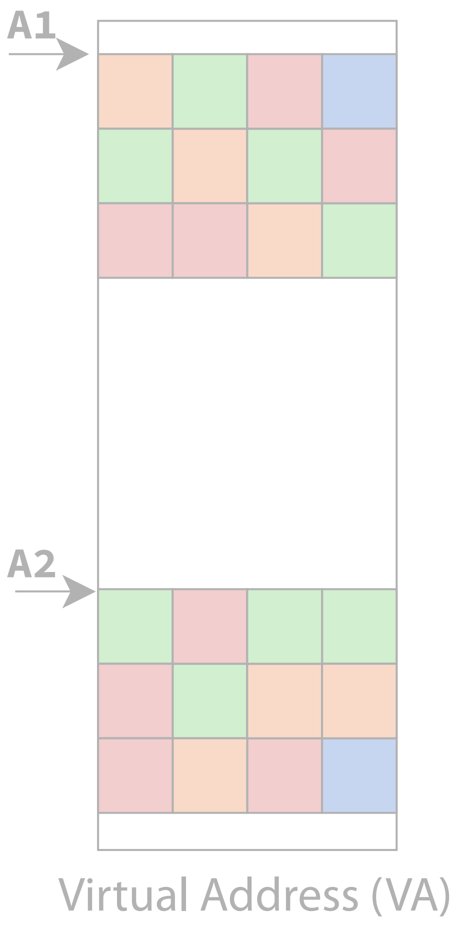
Struct Definition

```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

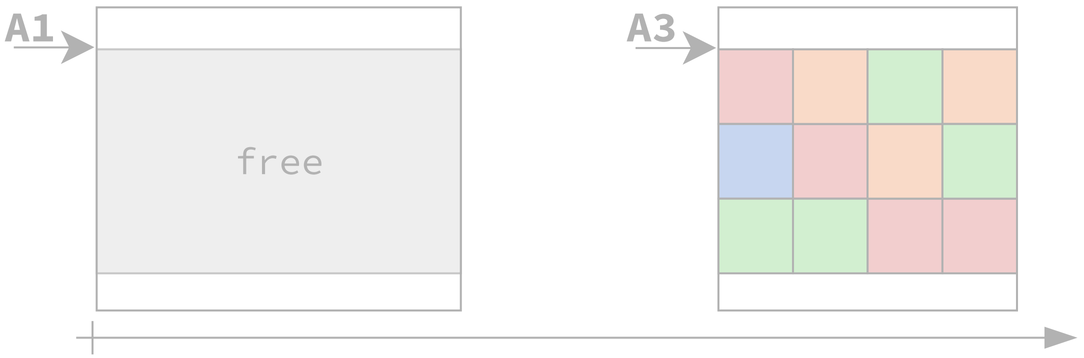
```
A_t *A1 = malloc(
      sizeof(A_t));
A_t *A2 = malloc(
      sizeof(A_t));

free(A1);
A_t *A3 = malloc(
      sizeof(A_t));
```

main.c

1 Object Allocation



2 Object Deallocation & Reuse



3 Multi-Dimensional Objects

```
typedef struct {
  char a;
  double b;
  A_t_c *c_ptr;
  void (*fp)();
} A_t;
```

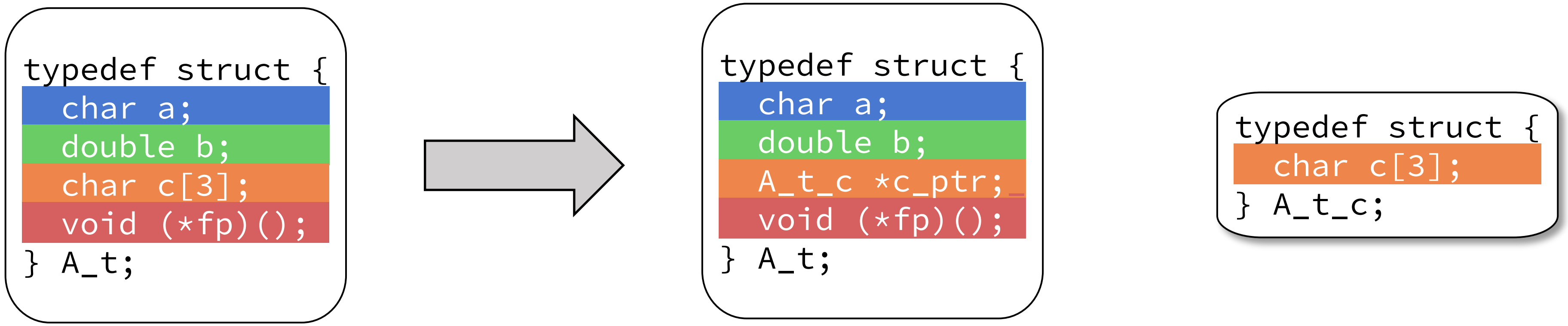
```
typedef struct {
  char c[3];
} A_t_c;
```

3

Multi-Dimensional Objects

Multi-Dimensional Objects

Buf2Ptr Transformation



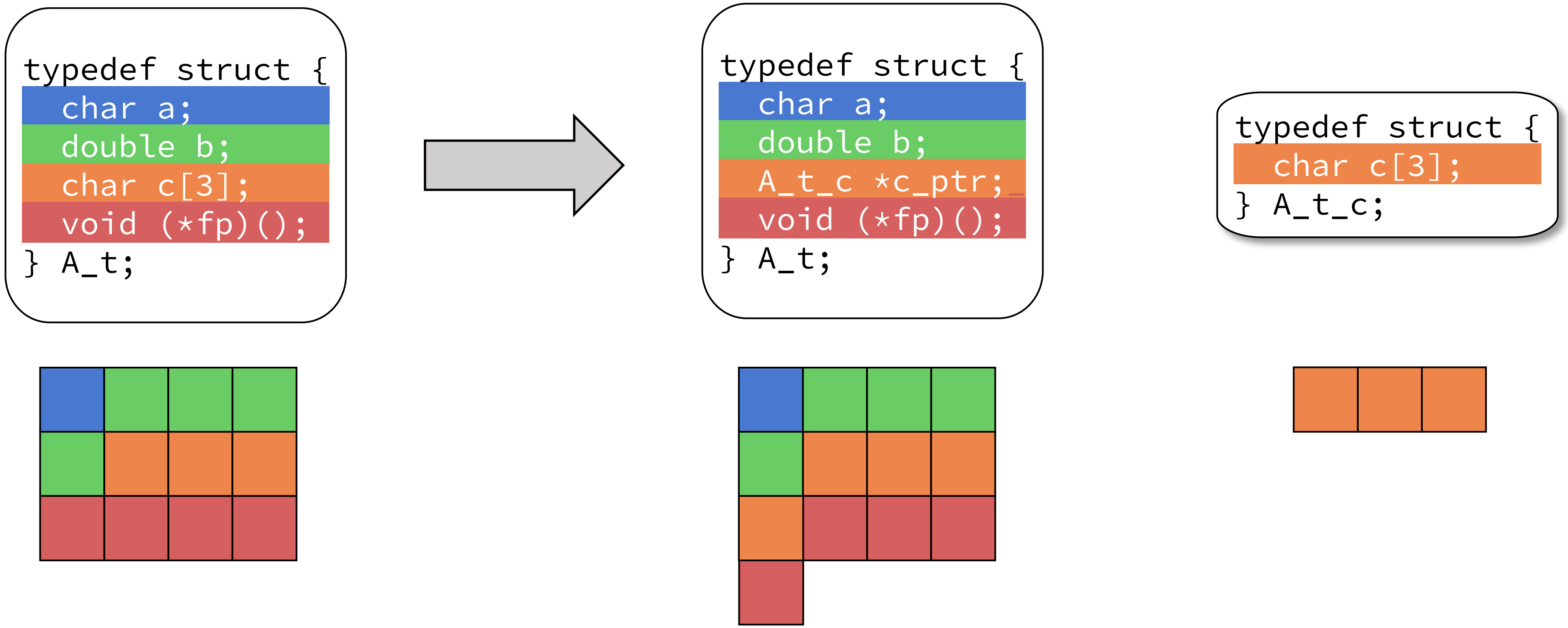
```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

```
typedef struct {
  char a;
  double b;
  A_t_c *c_ptr;
  void (*fp)();
} A_t;
```

```
typedef struct {
  char c[3];
} A_t_c;
```

Multi-Dimensional Objects

Buf2Ptr Transformation



```
typedef struct {
  char a;
  double b;
  char c[3];
  void (*fp)();
} A_t;
```

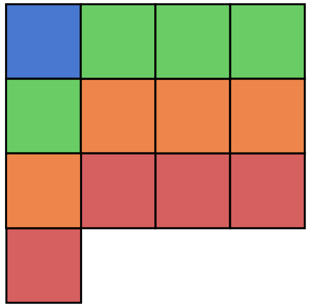
```
typedef struct {
  char a;
  double b;
  A_t_c *c_ptr;
  void (*fp)();
} A_t;
```

```
typedef struct {
  char c[3];
} A_t_c;
```

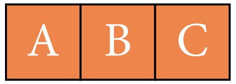
Multi-Dimensional Objects

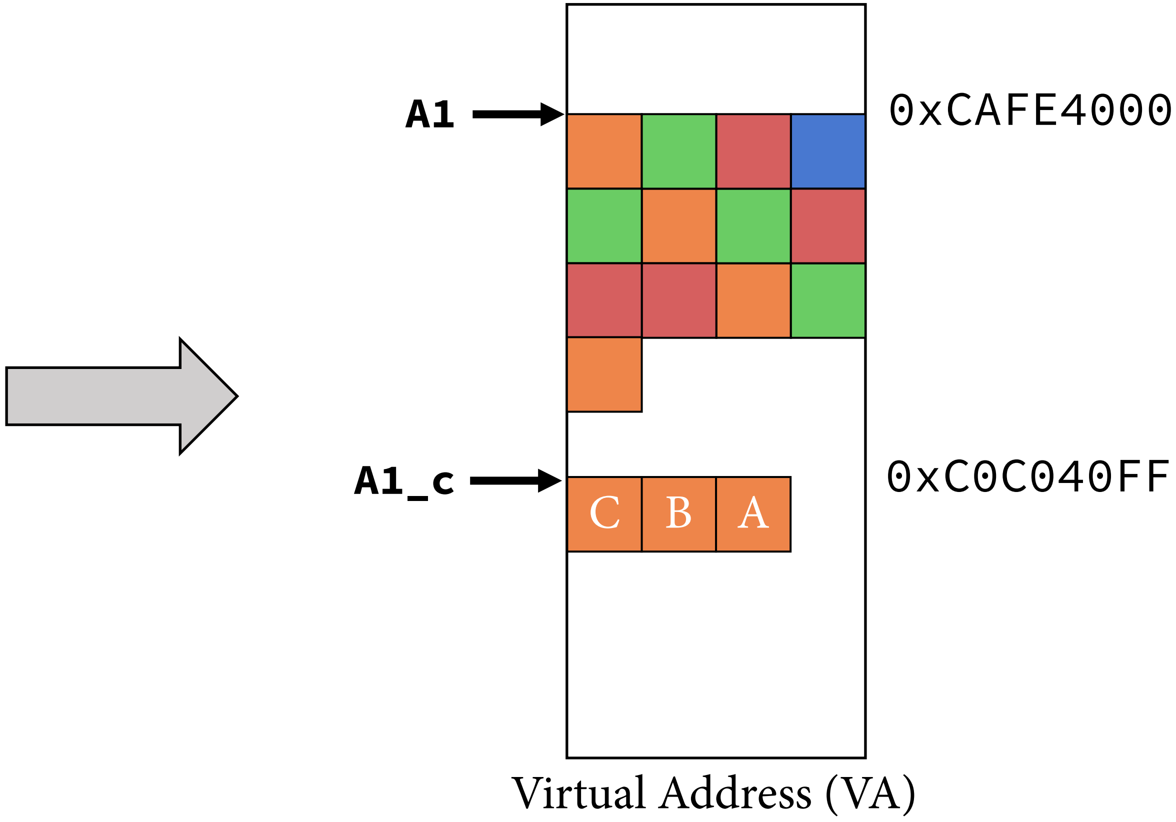
Allocation & Permutation

```
typedef struct {
  char a;
  double b;
  A_t_c *c_ptr;
  void (*fp)();
} A_t;
```



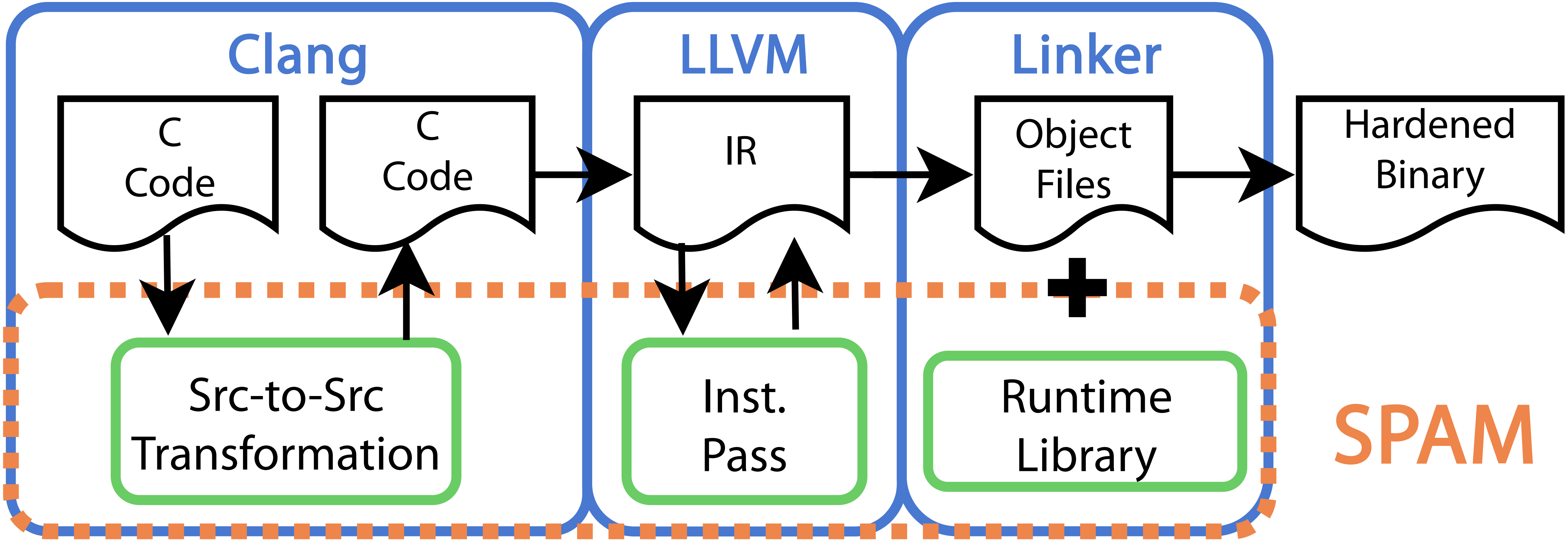
```
typedef struct {
  char c[3];
} A_t_c;
```



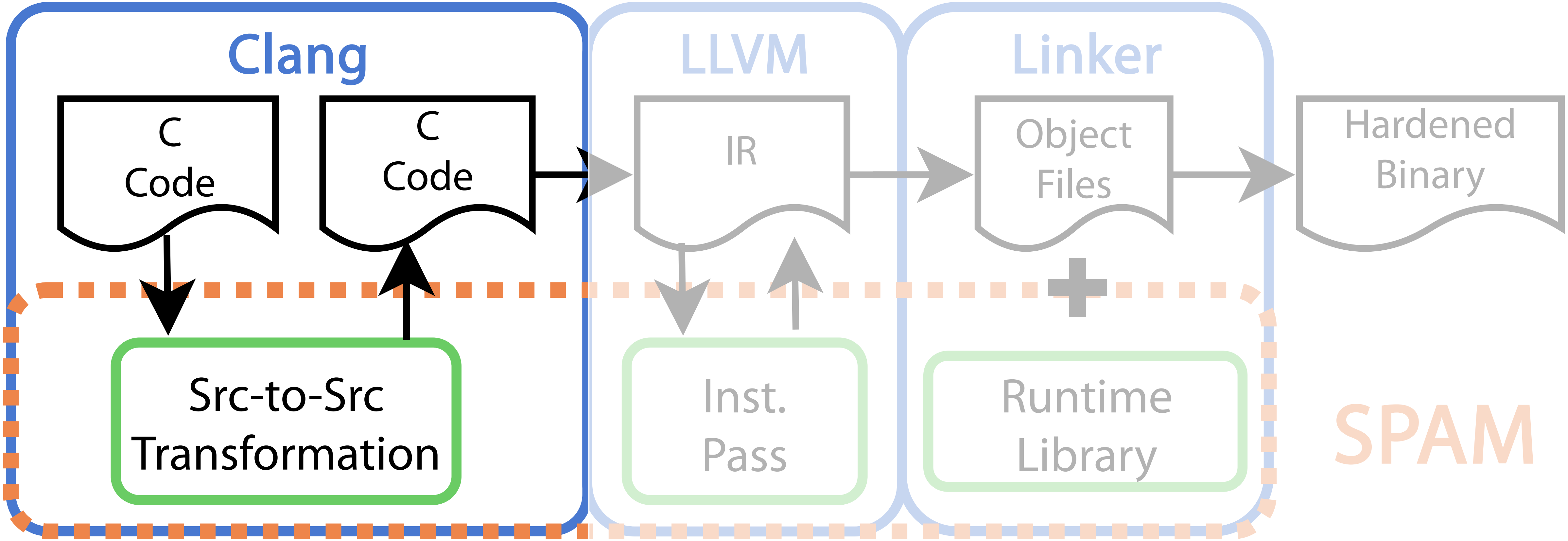


Implementation

Framework


Clang
LLVM
Linker
C Code
C Code
IR
Object Files
Hardened Binary
Src-to-Src Transformation
Inst. Pass
Runtime Library
SPAM

Framework



Clang
C Code
C Code
Src-to-Src Transformation
LLVM
IR
Inst. Pass
Linker
Object Files
Runtime Library
Hardened Binary
SPAM

Buf2Ptr: Source-to-Source Transformation

```c
struct Foo {
  char buf[10];
};
```

(a) Original

```c
// Promoted Type
struct Foo_buf {
  char buf[10];
};
struct Foo {
  struct Foo_buf *p_buf;
};
```

(b) Transformed

Buf2Ptr: Source-to-Source Transformation

```c
struct Foo {
  char buf[10];
};
```

```c
// Promoted Type
struct Foo_buf {
  char buf[10];
};
struct Foo {
  struct Foo_buf *p_buf;
};
```

```c
struct Foo *f = malloc(
  sizeof(struct Foo));
```

```c
// Promoted Allocations
struct Foo *f = malloc(
sizeof(struct Foo));
f->p_buf = malloc(
sizeof(struct Foo_buf));
```

(a) Original

(b) Transformed

Buf2Ptr: Source-to-Source Transformation

```
                                  // Promoted Type
struct Foo {                      struct Foo_buf {
  char buf[10];                     char buf[10];
};                                };
                                  struct Foo {
                                    struct Foo_buf *p_buf;
                                  };
```

```
                                  // Promoted Allocations
struct Foo *f = malloc(           struct Foo *f = malloc(
  sizeof(struct Foo));            sizeof(struct Foo));
                                  f->p_buf = malloc(
                                  sizeof(struct Foo_buf));
```

```
                                  // Promoted Usages
f->buf[7] = 'A’;                  f->p_buf->buf[7] = 'A’;
```

(a) Original (b) Transformed

Buf2Ptr: Source-to-Source Transformation

```
                                  // Promoted Type
struct Foo {                      struct Foo_buf {
  char buf[10];                     char buf[10];
};                                };
                                  struct Foo {
                                    struct Foo_buf *p_buf;
                                  };
```

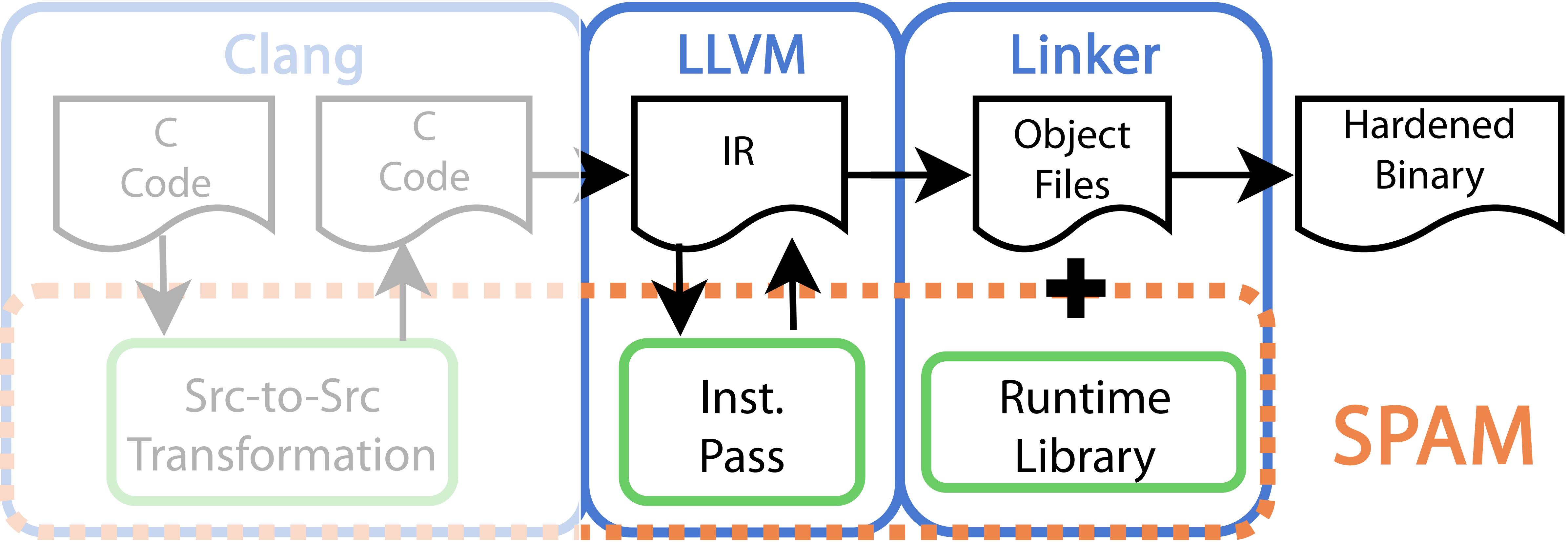
```
                                  // Promoted Allocations
struct Foo *f = malloc(           struct Foo *f = malloc(
  sizeof(struct Foo));            sizeof(struct Foo));
                                  f->p_buf = malloc(
                                  sizeof(struct Foo_buf));
```

```
                                  // Promoted Usages
f->buf[7] = 'A’;                  f->p_buf->buf[7] = 'A’;
```

```
                                  // Promoted Deallocations
                                  free(f->p_buf);
free(f);                          free(f);
```

(a) Original (b) Transformed

Framework


Clang
C
Code
C
Code
Src-to-Src
Transformation
LLVM
IR
Inst.
Pass
Linker
Object
Files
+
Runtime
Library
Hardened
Binary
SPAM

Instrumentation & Runtime

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

Instrumentation & Runtime

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

Baseline Compilation

Flags: `-O0`

Instrumentation & Runtime

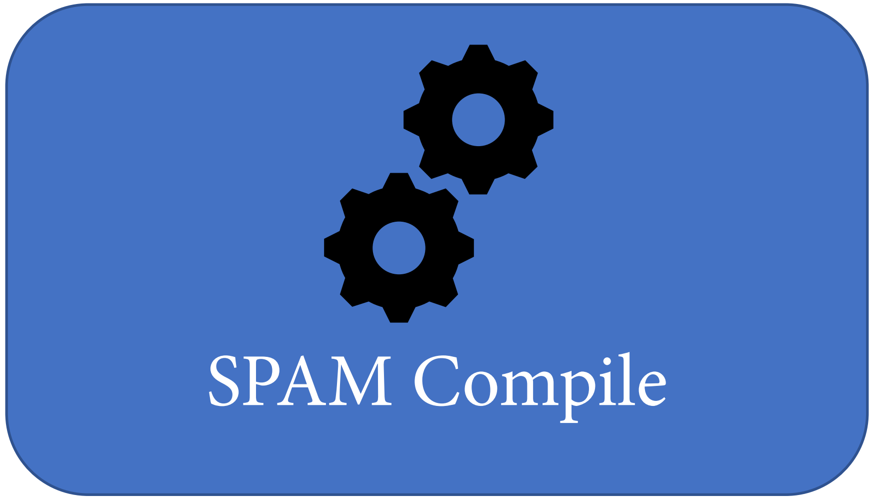
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

```llvm
define i32 @main() {
  %ptr = call i8* @malloc(i64 128)



  store i8 65, i8* %ptr, align 1



  %load = load i8, i8* %ptr, align 1

  %conv = sext i8 %load to i32
  %print = call i32 (i8*, ...) @printf(i8*
           getelementptr inbounds ([4 x
           i8], [4 x i8]* @.str, i32 0, i32 0),
           i32 %conv)
  ret i32 0
}
```

Instrumentation & Runtime

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

```llvm
define i32 @main() {
  %ptr = call i8* @malloc(i64 128)



  store i8 65, i8* %ptr, align 1



  %load = load i8, i8* %ptr, align 1

  %conv = sext i8 %load to i32
  %print = call i32 (i8*, ...) @printf(i8*
              getelementptr inbounds ([4 x
              i8], [4 x i8]* @.str, i32 0, i32 0),
              i32 %conv)
  ret i32 0
}
```


SPAM Compile

Instrumentation & Runtime

SPAM Runtime

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

```
define i32 @main() {
  %ptr = call i8* @spam_malloc(i64 128)

  %store_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

  store i8 65, i8* %store_off, align 1

  %load_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

  %load = load i8, i8* %load_off, align 1

  %conv = sext i8 %load to i32
  %print = call i32 (i8*, ...) @printf(i8*
           getelementptr inbounds ([4 x
           i8], [4 x i8]* @.str, i32 0, i32 0),
           i32 %conv)
  ret i32 0
}
```

Instrumentation & Runtime

To tag/untag Alias Number from pointer.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

```llvm
define i32 @main() {
  %ptr = call i8* @spam_malloc(i64 128)

  %store_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

  store i8 65, i8* %store_off, align 1

  %load_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

  %load = load i8, i8* %load_off, align 1

  %conv = sext i8 %load to i32
  %print = call i32 (i8*, ...) @printf(i8*
           getelementptr inbounds ([4 x
           i8], [4 x i8]* @.str, i32 0, i32 0),
           i32 %conv)
  ret i32 0
}
```

Instrumentation & Runtime

Returns pointer with calculated permuted offset.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

```
define i32 @main() {
  %ptr = call i8* @spam_malloc(i64 128)

  %store_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

  store i8 65, i8* %store_off, align 1

  %load_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

  %load = load i8, i8* %load_off, align 1

  %conv = sext i8 %load to i32
  %print = call i32 (i8*, ...) @printf(i8*
           getelementptr inbounds ([4 x
           i8], [4 x i8]* @.str, i32 0, i32 0),
           i32 %conv)
  ret i32 0
}
```

Instrumentation & Runtime

Global Support

```
void RegisterGlobal(void *Ptr)
```

For `.data` section hook into `.ctor` to permute on program load.

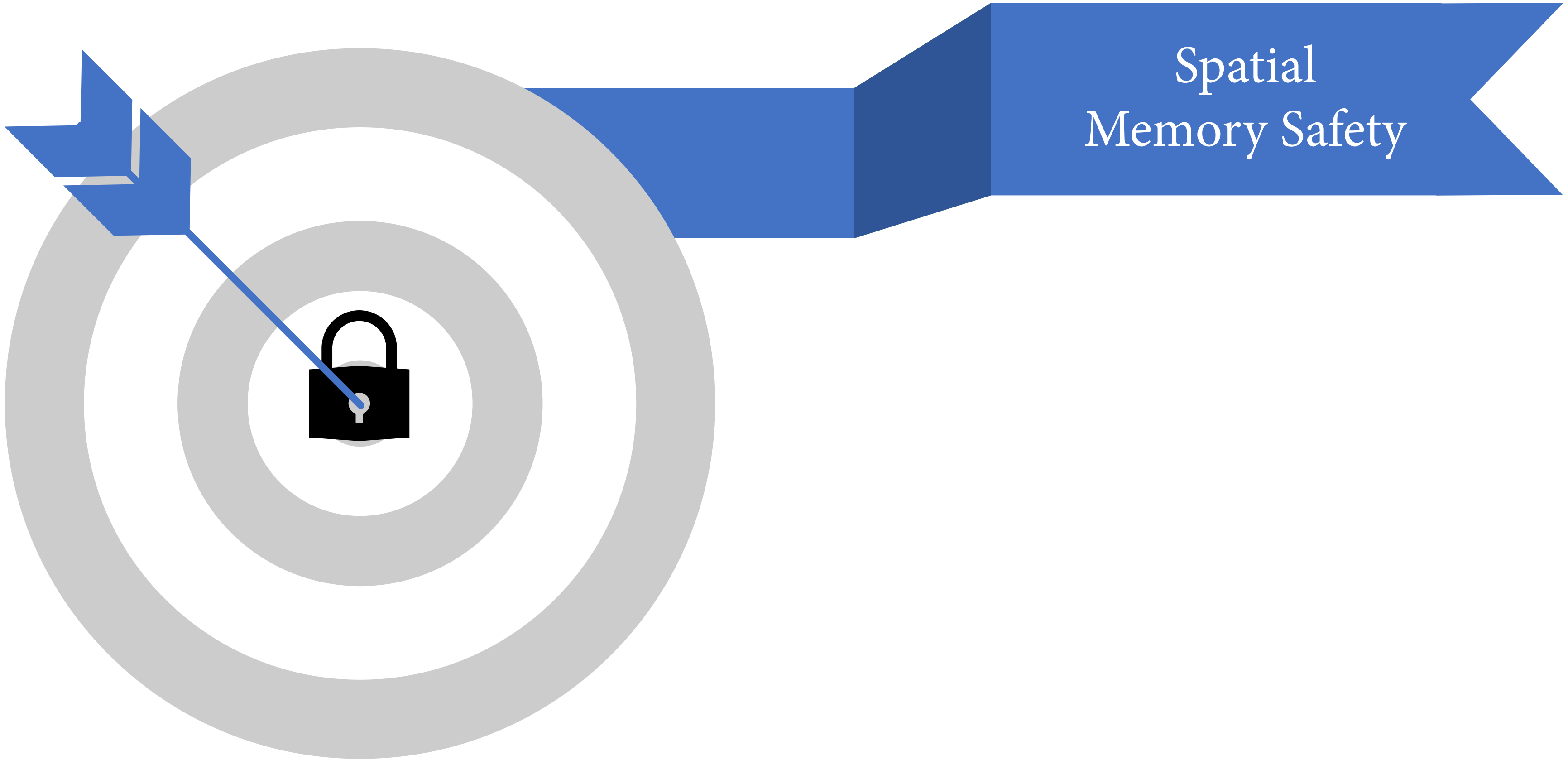
Stack Support

```
void *RegisterStack(void *Ptr)
```

For variables passed by OS (e.g. `argv`) hook into `main` to permute on start.

Why SPAM?

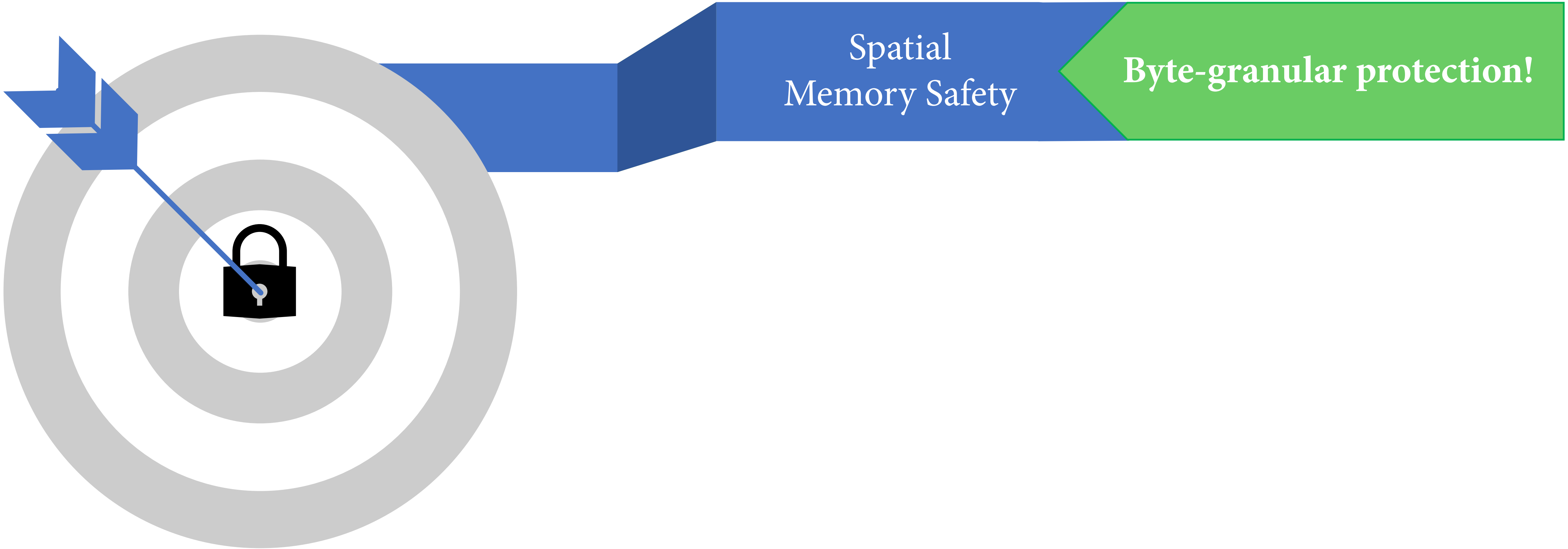
SPAM Benefits



Spatial Memory Safety

- Every object instance (allocation) is permuted independently.
- Overflows within an object (intra) are transformed.

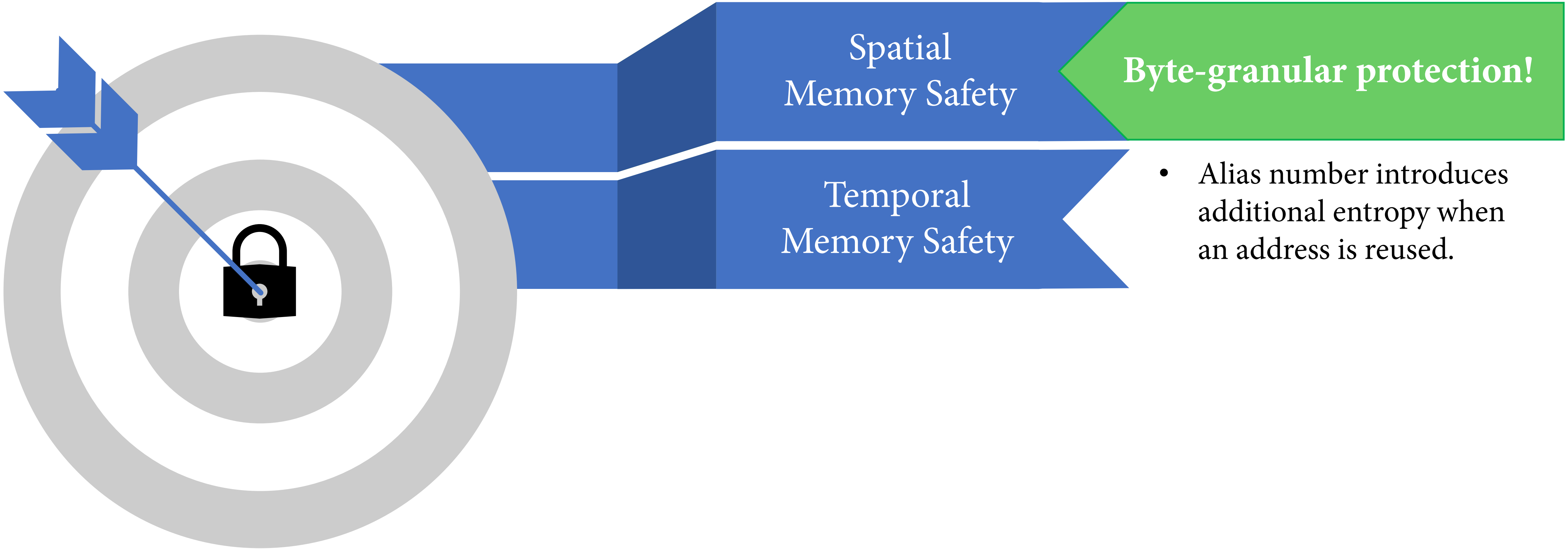
SPAM Benefits



Spatial
Memory Safety

Byte-granular protection!

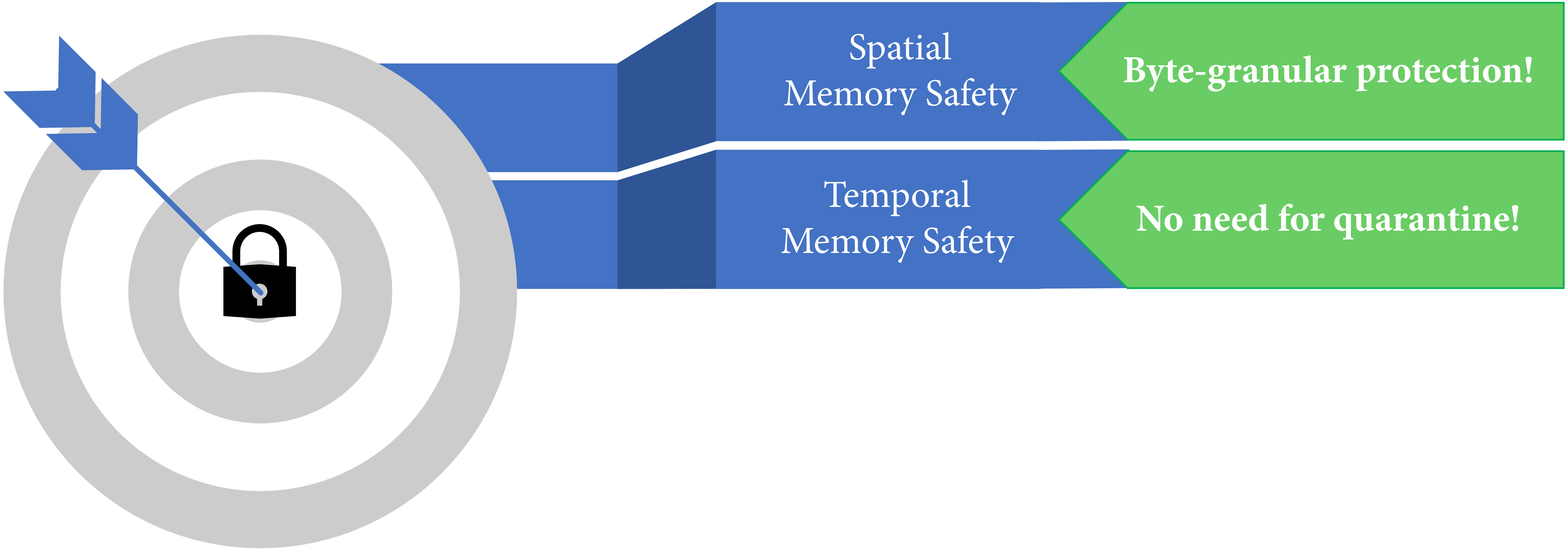
SPAM Benefits



Byte-granular protection!

- Alias number introduces additional entropy when an address is reused.

SPAM Benefits



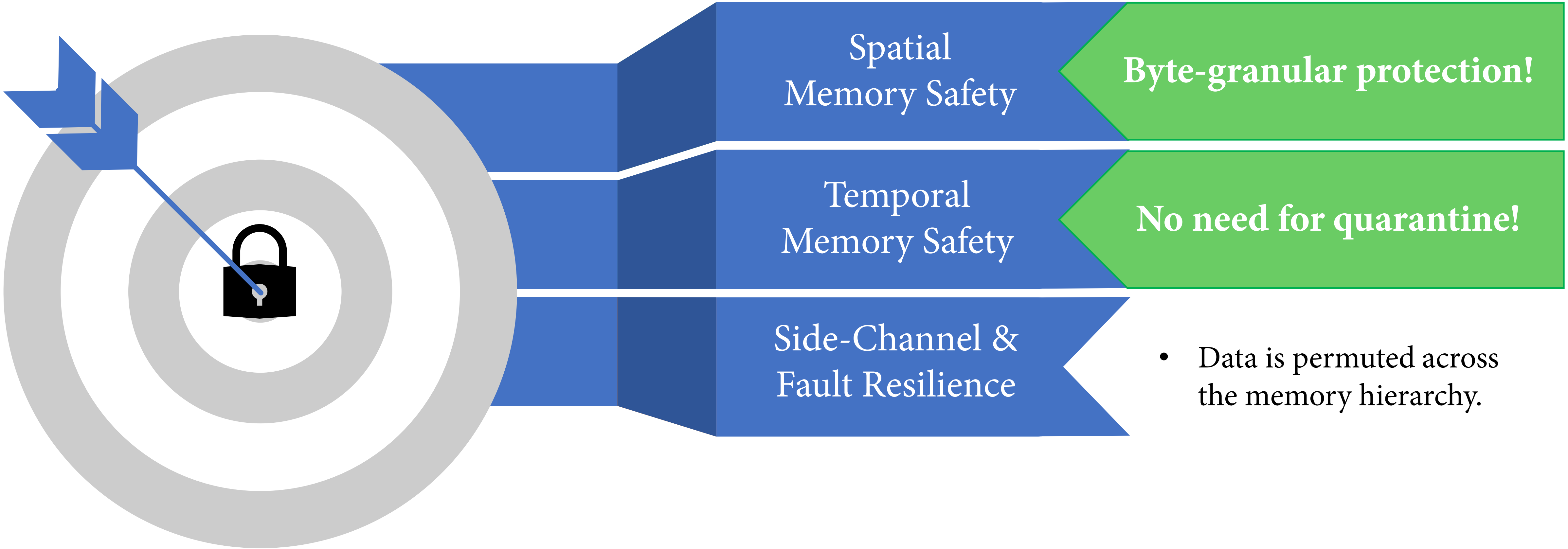
Spatial Memory Safety

Byte-granular protection!

Temporal Memory Safety

No need for quarantine!

SPAM Benefits



Spatial Memory Safety

Byte-granular protection!

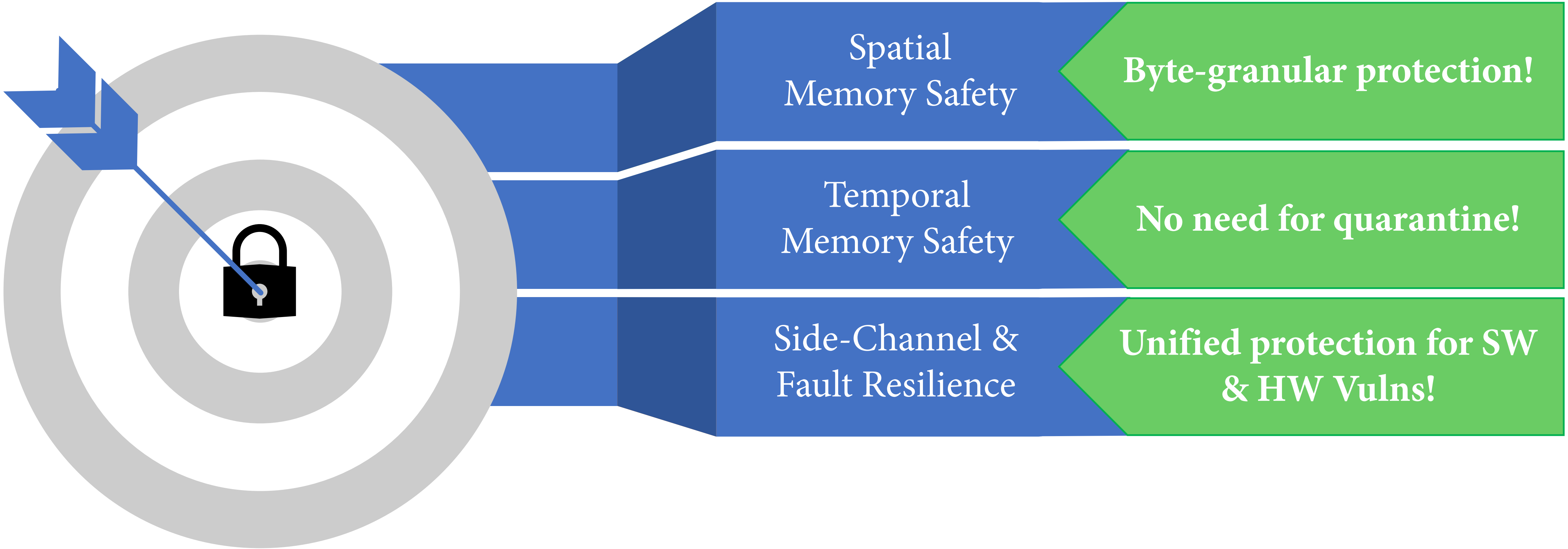
Temporal Memory Safety

No need for quarantine!

Side-Channel & Fault Resilience

- Data is permuted across the memory hierarchy.

SPAM Benefits



Spatial Memory Safety

Byte-granular protection!

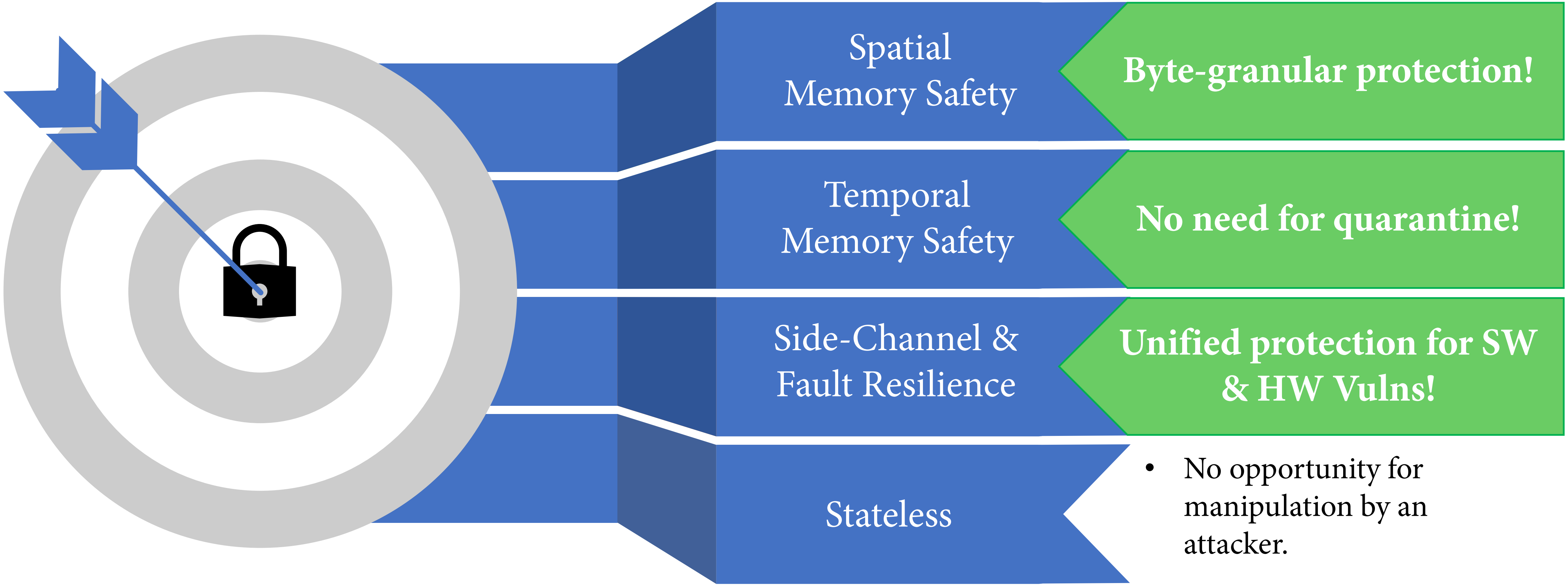
Temporal Memory Safety

No need for quarantine!

Side-Channel & Fault Resilience

Unified protection for SW & HW Vulns!

SPAM Benefits



Spatial Memory Safety

Byte-granular protection!

Temporal Memory Safety

No need for quarantine!

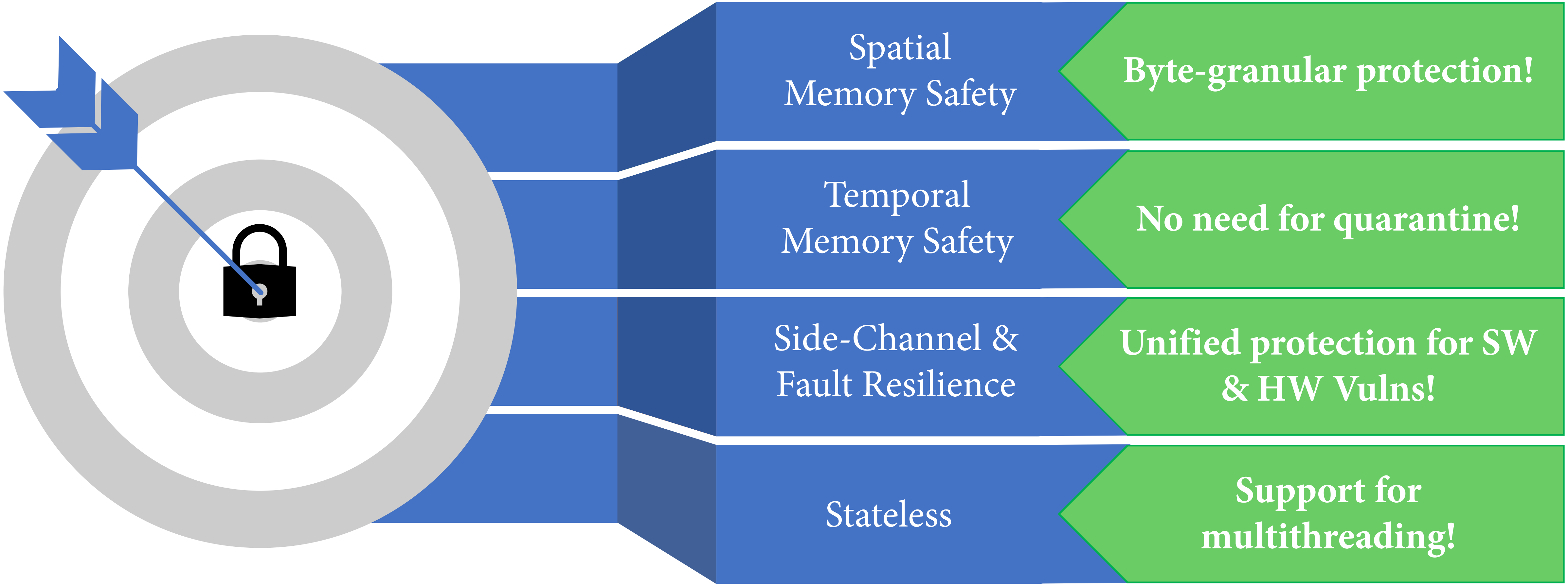
Side-Channel & Fault Resilience

Unified protection for SW & HW Vulns!

Stateless

- No opportunity for manipulation by an attacker.

SPAM Benefits



Spatial Memory Safety	Byte-granular protection!
Temporal Memory Safety	No need for quarantine!
Side-Channel & Fault Resilience	Unified protection for SW & HW Vulns!
Stateless	Support for multithreading!

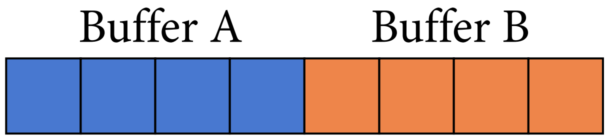
Resilience to Common Exploits

Resilience to Common Exploits

1

Buffer Over-/Under-flows

Cannot reliably corrupt memory.



Buffer A

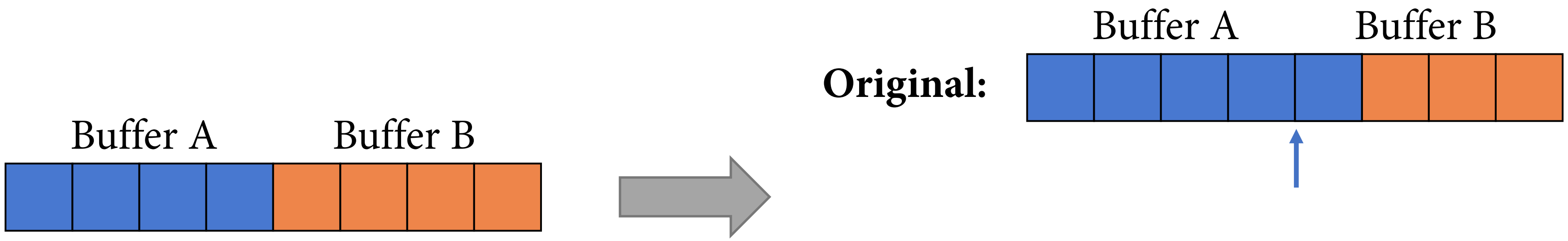
Buffer B

Resilience to Common Exploits

1

Buffer Over-/Under-flows

Cannot reliably corrupt memory.

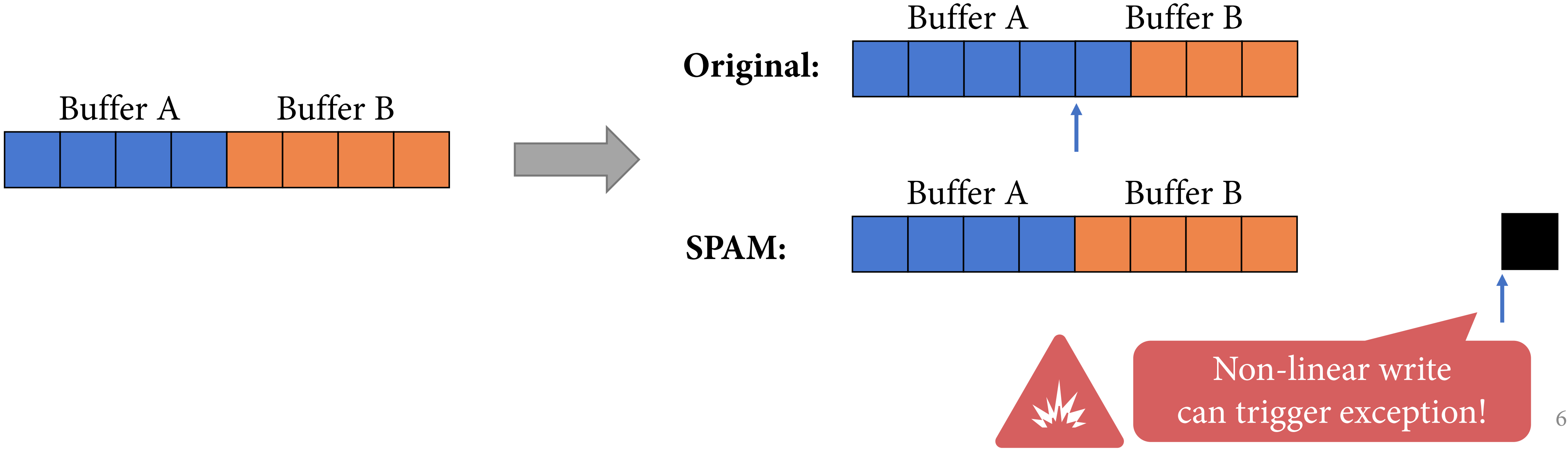


Resilience to Common Exploits

1

Buffer Over-/Under-flows

Cannot reliably corrupt memory.



Non-linear write can trigger exception!

Resilience to Common Exploits

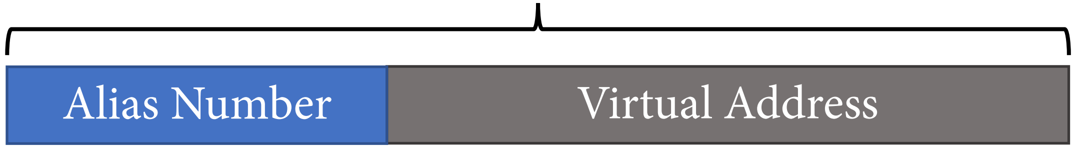
1 **Buffer Over-/Under-flows**

Cannot reliably corrupt memory.

2 **Use-after-free**

Each instance permutated independently.

Alias number provides multiple permutations.



Resilience to Common Exploits

1. **Buffer Over-/Under-flows**

   Cannot reliably corrupt memory.

2. **Use-after-free**

   Each instance permutated independently.

3. **Speculative Attacks**

   Speculative load uses a different permutation to access the permuted data.

```
// mispredicted branch
if (i < sizeof(a)) {

  secret = a[i];

  // secret is leaked
  val = b[64 * secret];
}
```

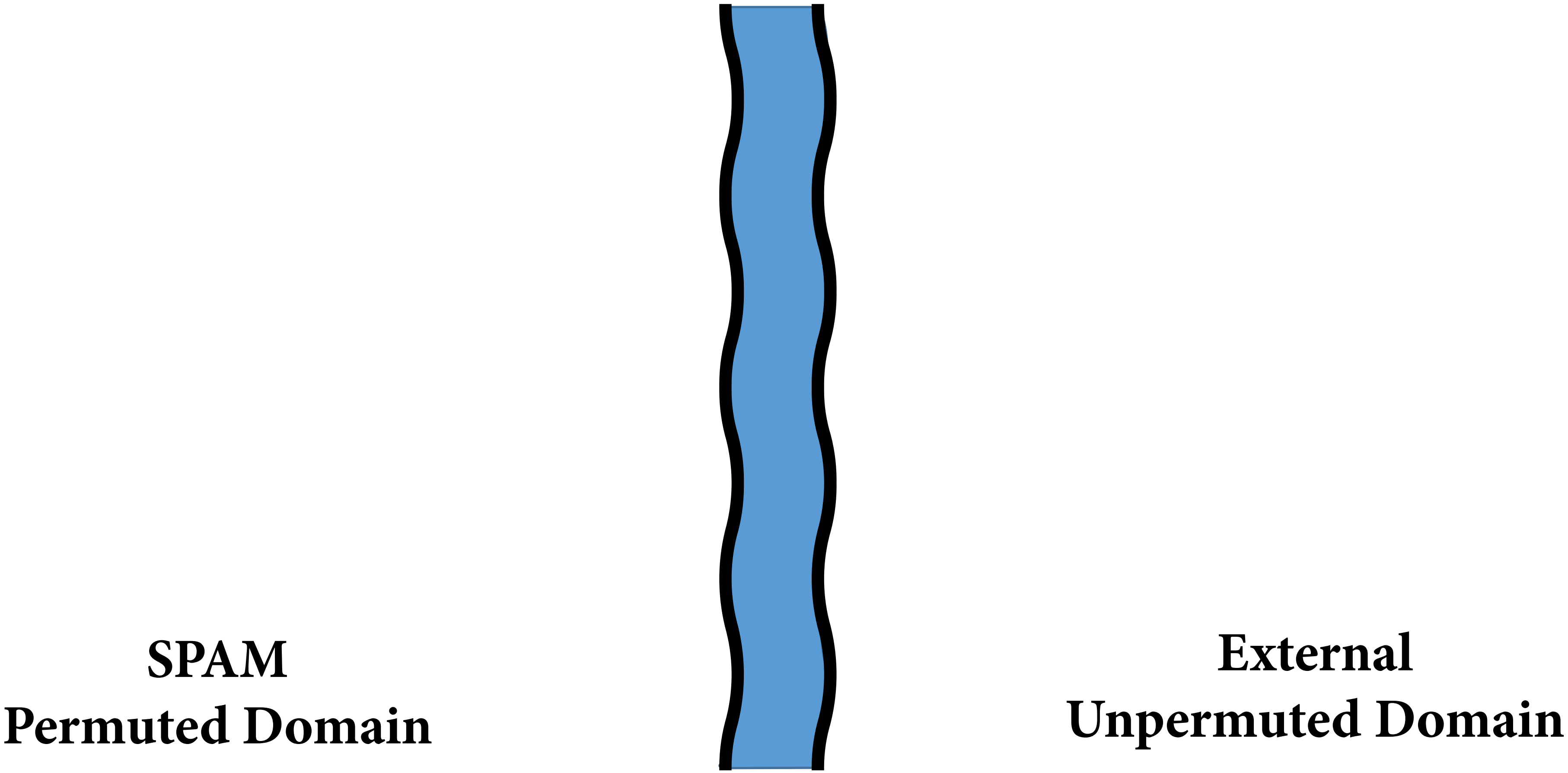
Unpredictable!

- Attacker will end up with an unpredictable value in `secret` due as the permutation depends on the address of `a[i]`.

SPAM Meets Reality

SPAM Meets Reality

Compatibility with Uninstrumented Code



SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *p = malloc(128);
    *p = 'A';
    printf("%c\n", *p);
    return 0;
}
```

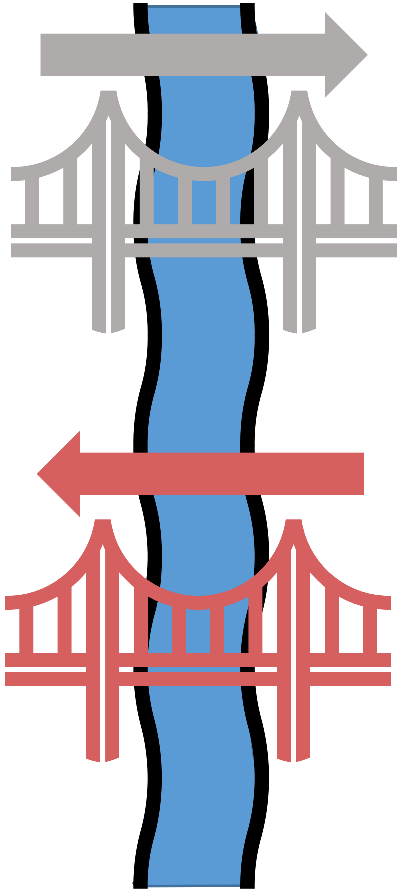
SPAM
Permuted Domain

```c
int printf(const char *fmt, ...) {
    int err;

    va_list ap;
    va_start(ap, fmt);
    err = _dvprintf(fmt, ap);
    va_end(ap);

    return err;
}
```

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

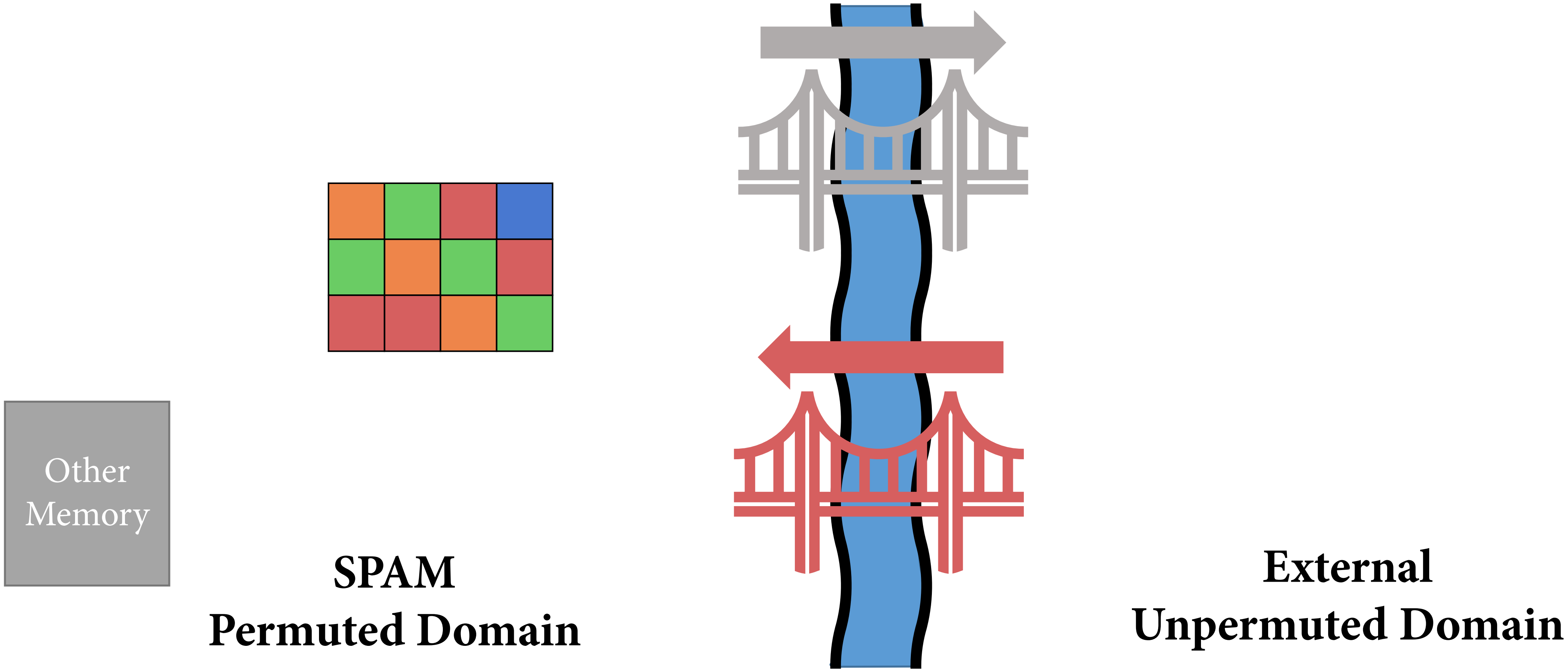


SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



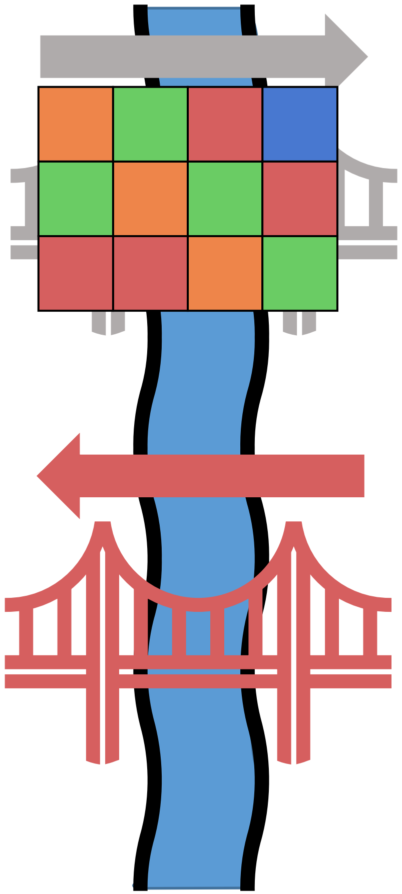
Other Memory

SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



Other Memory

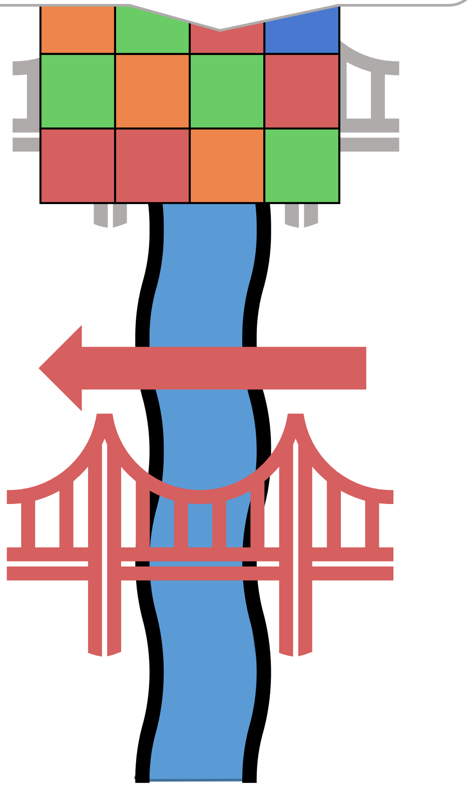
SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

```
void *Unpermute(void *Ptr)
```



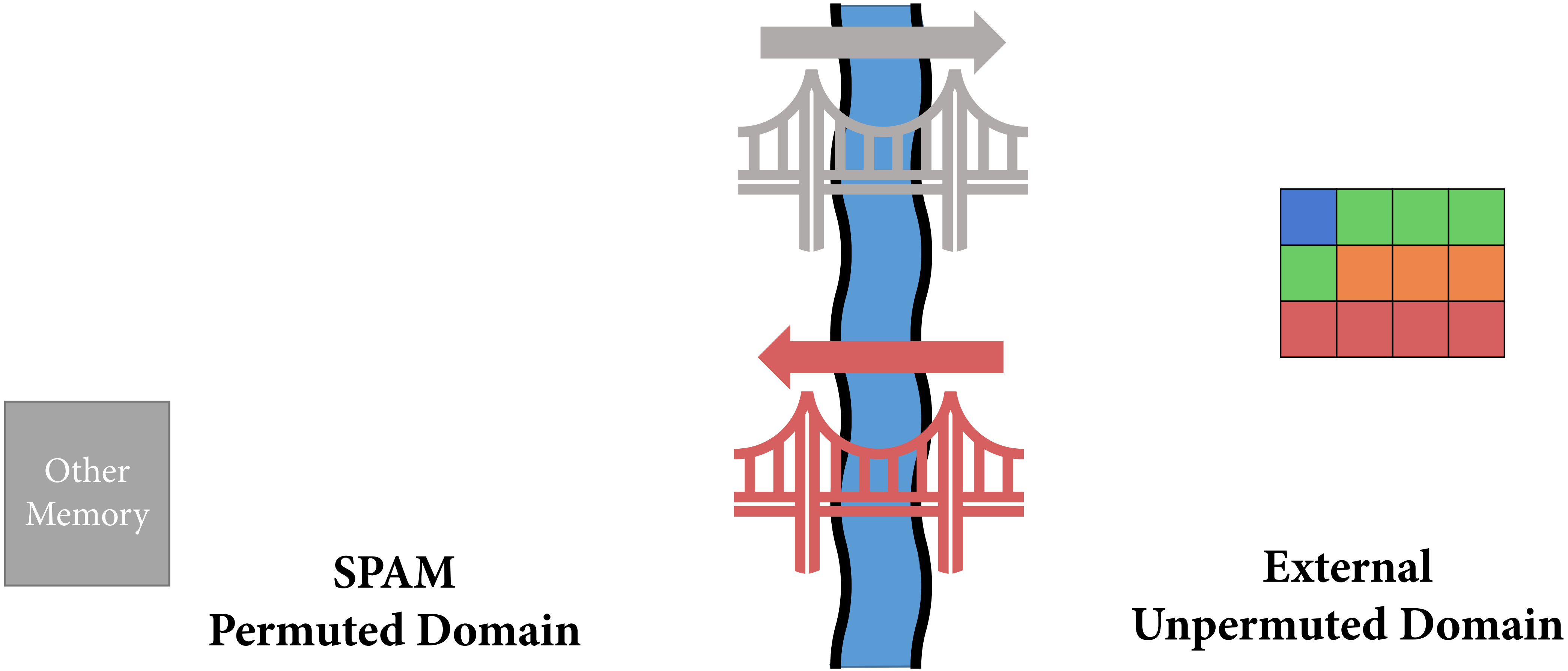
Other Memory

SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



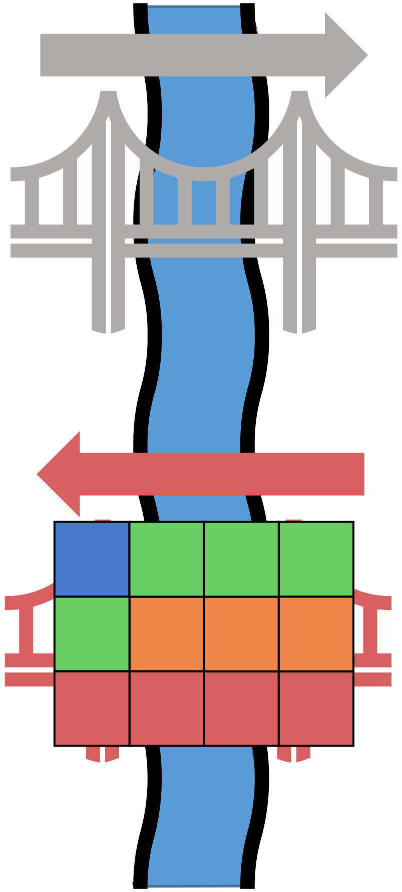
Other Memory

SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



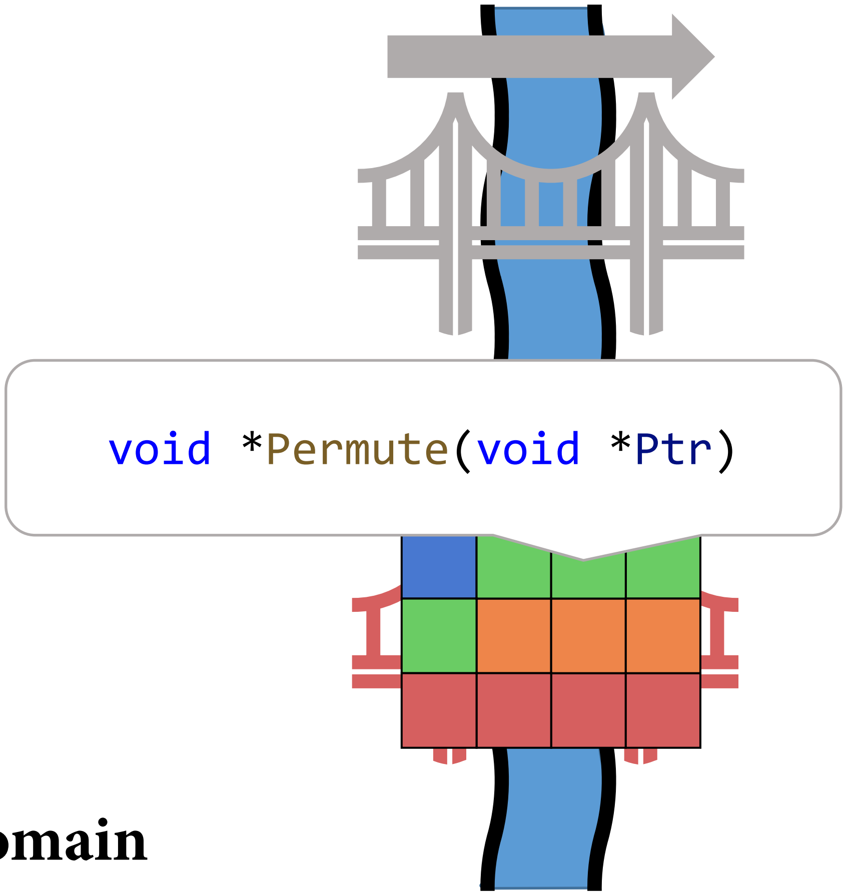
Other Memory

SPAM Permuted Domain

External Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



```
void *Permute(void *Ptr)
```

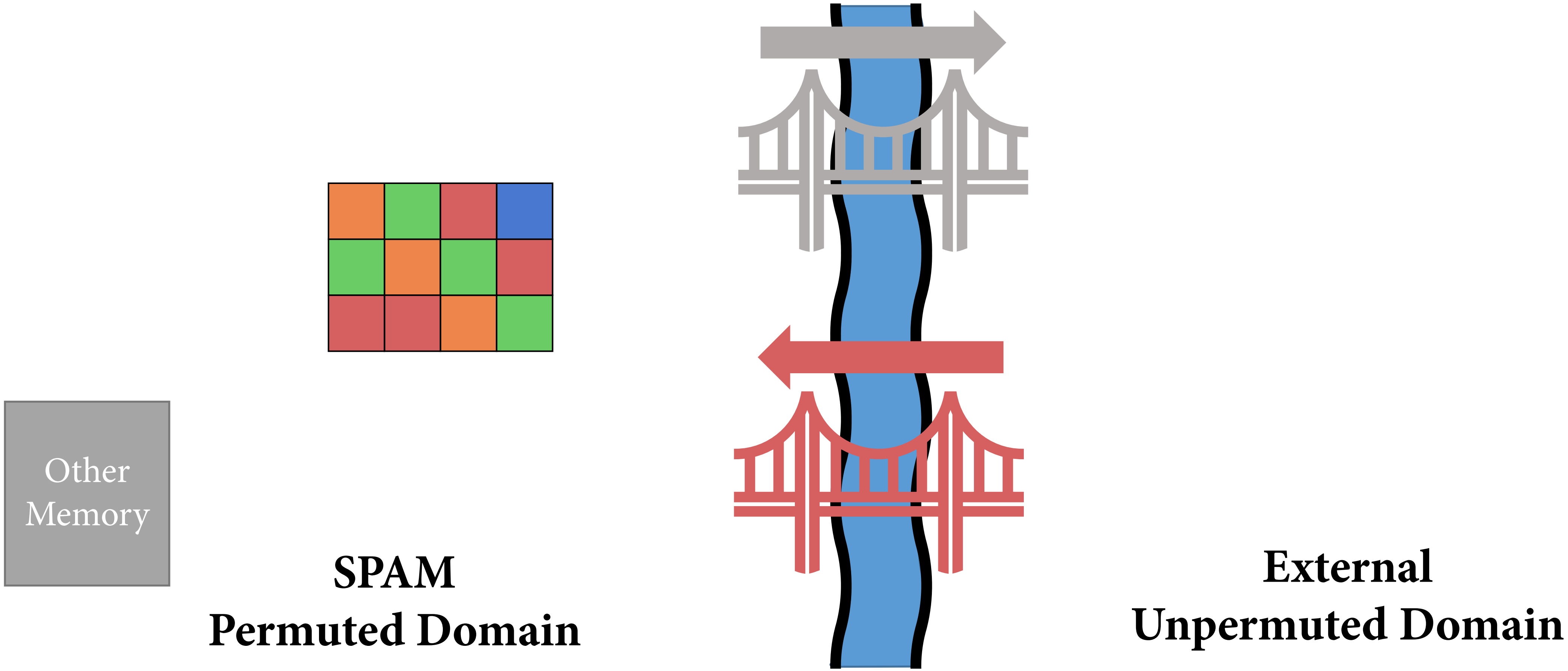
Other Memory

SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



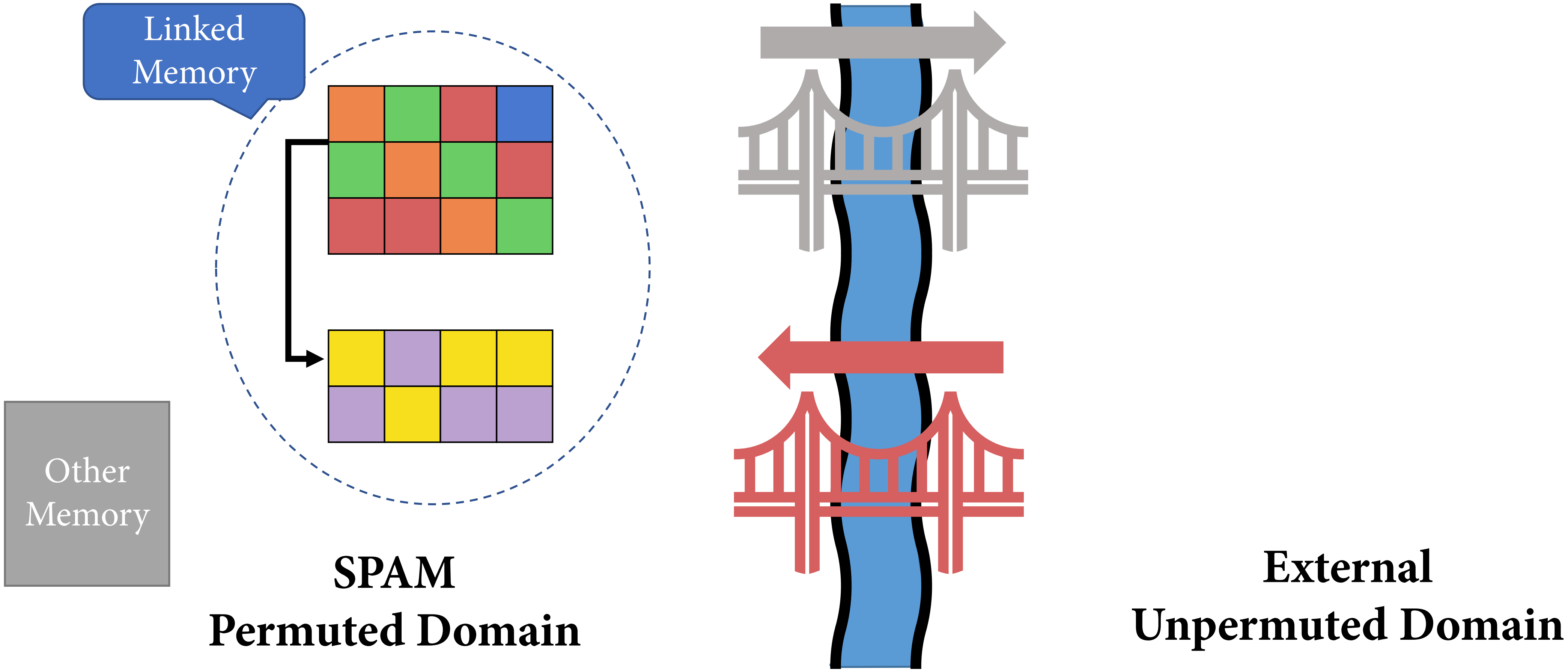
Other Memory

SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code



Linked Memory

Other Memory

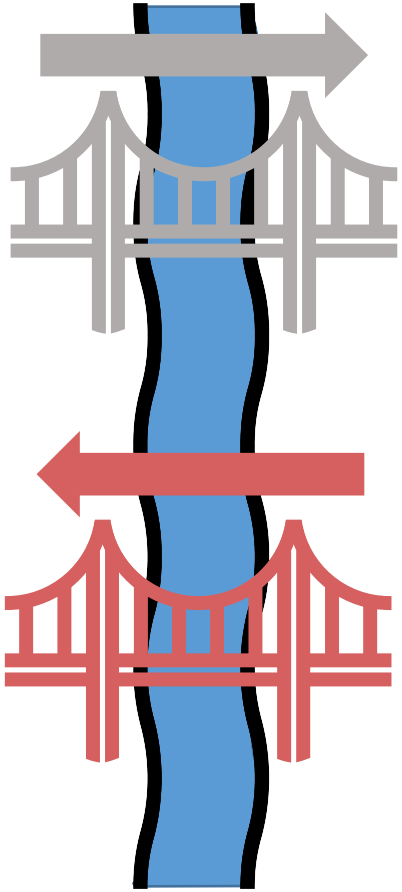
SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

```
int cmp (const void * a,
         const void * b) {...}

int main() {
    ...
    qsort(b, 10, 10, cmp);
    ...
}
```



```
void qsort(void *base,
           size_t nitems,
           size_t size,
int (*cmp)(const void *, const void*))
{...}
```

SPAM
Permuted Domain

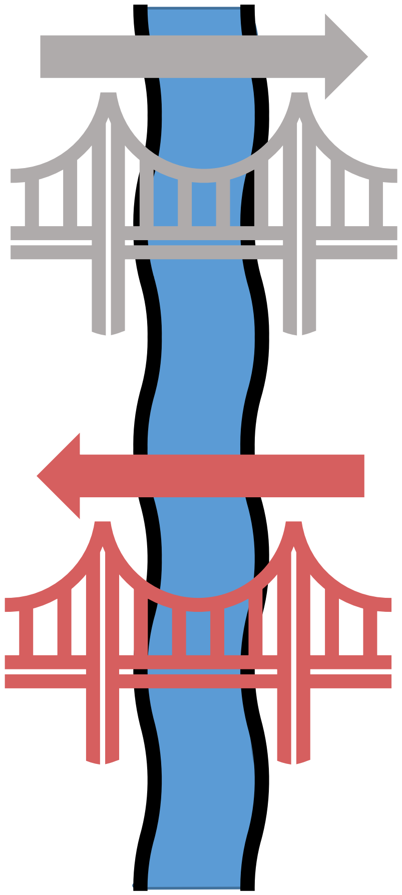
External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

```
int cmp (const void * a,
         const void * b) {...}

int main() {
    ...
    qsort(b, 10, 10, cmp);
    ...
}
```

SPAM
Permuted Domain



```
void qsort(void *base,
           size_t nitems,
           size_t size,
int (*cmp)(const void *, const void*))
{...}
```

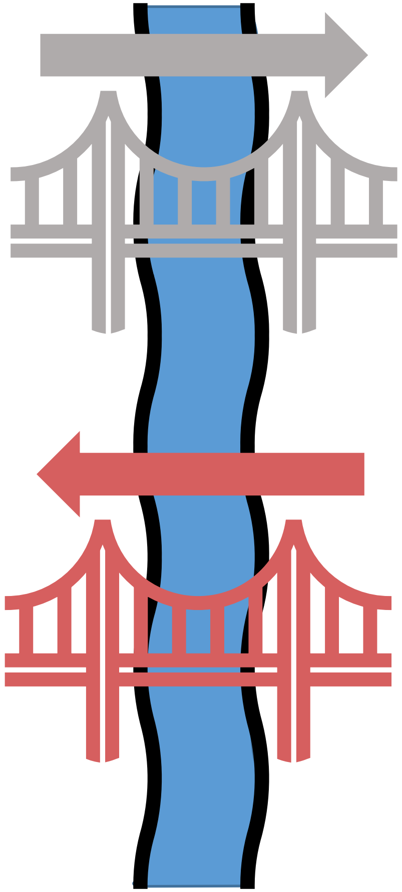
External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

```
int cmp (const void * a,
         const void * b) {...}
```

```
int main() {
    ...
    qsort(b, 10, 10, cmp);
    ...
}
```



```
void qsort(void *base,
           size_t nitems,
           size_t size,
int (*cmp)(const void *, const void*))
{...}
```

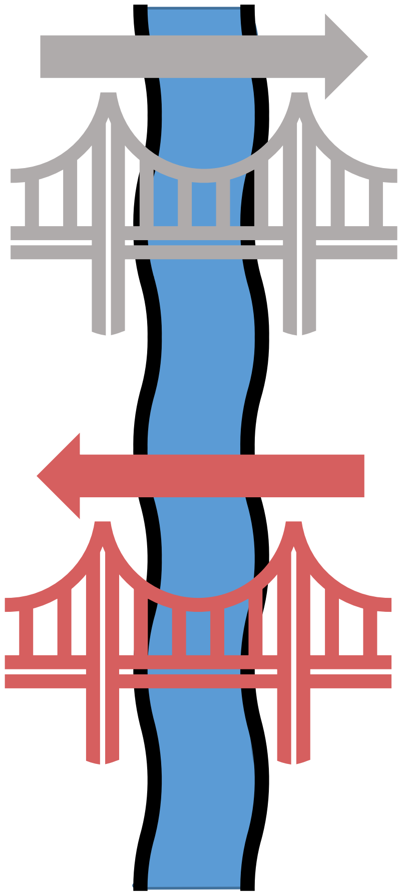
SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Compatibility with Uninstrumented Code

```
int main() {
    ...
    qsort(b, 10, 10, cmp);
    ...
}
```



```
int cmp (const void * a,
         const void * b) {...}
```

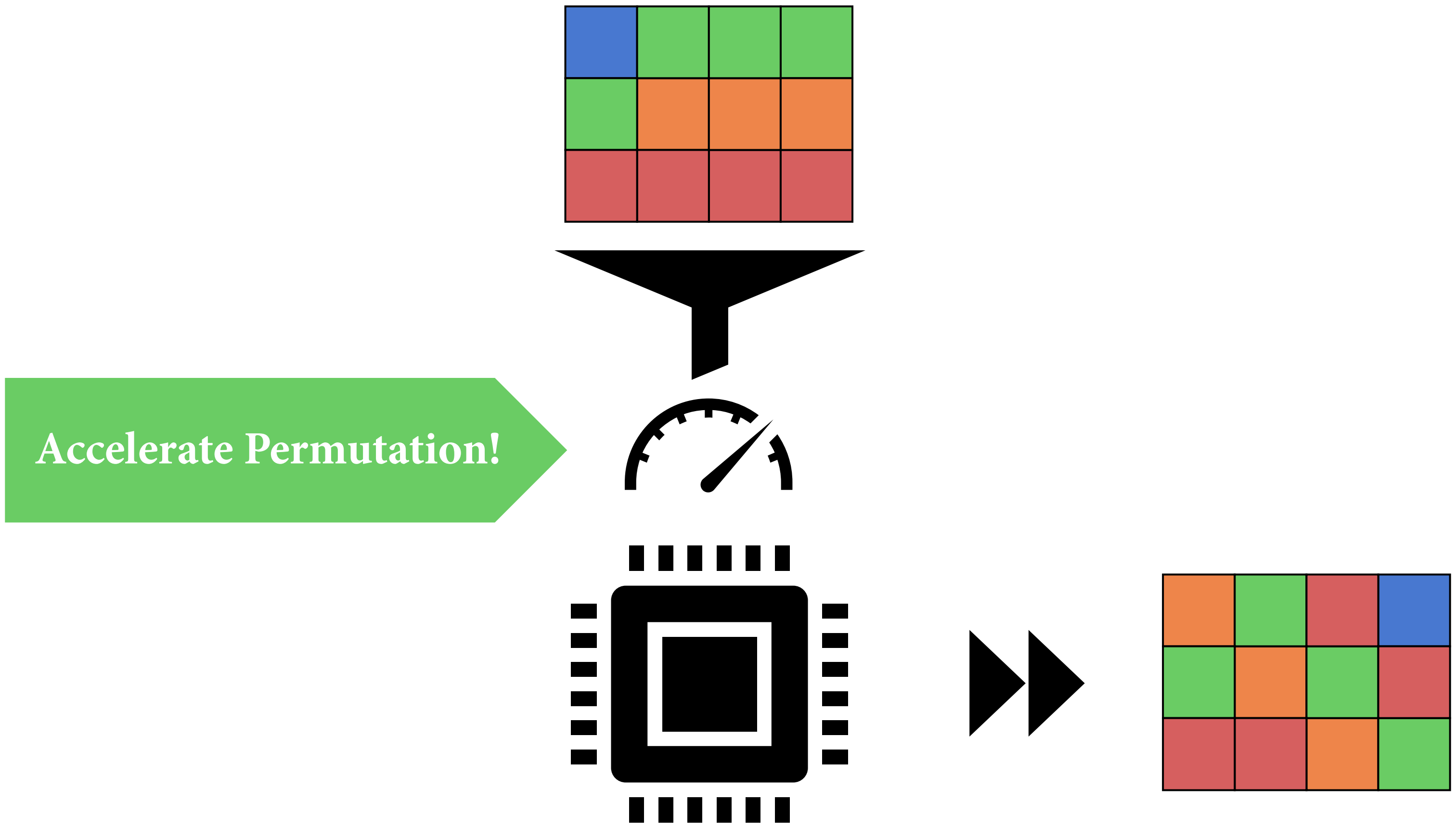
```
void qsort(void *base,
           size_t nitems,
           size_t size,
int (*cmp)(const void *, const void*))
{...}
```

SPAM
Permuted Domain

External
Unpermuted Domain

SPAM Meets Reality

Hardware Support



Accelerate Permutation!

SPAM Meets Reality

Hardware Support

```
%store_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

 store i8 65, i8* %store_off, align 1

 %load_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

 %load = load i8, i8* %load_off, align 1
```

SPAM Meets Reality

Hardware Support

```
%store_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

 store i8 65, i8* %store_off, align 1

 %load_off = call i8* @spam_get_perm_offset(i8* %ptr, i8* %ptr)

 %load = load i8, i8* %load_off, align 1
```

Reduce resource pressure!

```
spam_store i8 65, i8* %ptr, align 1

%load = spam_load i8, i8* %ptr, align 1
```

Other Mitigations

Other Mitigations

- *ARM MTE*
  - Memory & pointers are tagged with colors.

Source: *ARM v8.5A - https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Arm_Memory_Tagging_Extension_Whitepaper.pdf*

Other Mitigations

- *ARM MTE*
  - Memory & pointers are tagged with colors.

Limited set of colors.

Source: *ARM v8.5A - https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Arm_Memory_Tagging_Extension_Whitepaper.pdf*

Other Mitigations

- *ARM MTE*
  - Memory & pointers are tagged with colors.

Vulnerable to intra-object
&
type confusion.

Source: *ARM v8.5A - https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Arm_Memory_Tagging_Extension_Whitepaper.pdf*

Other Mitigations

- *ARM MTE*
  - Memory & pointers are tagged with colors.

- *Checked C*
  - Adds new pointer and array types that are bounds checked.

Source: *Elliott, Archibald Samuel, et al. "Checked C: Making C safe by extension." 2018 IEEE Cybersecurity Development (SecDev). IEEE, 2018.*

Other Mitigations

- *ARM MTE*
  - Memory & pointers are tagged with colors.

- *Checked C*
  - Adds new pointer and array types that are bounds checked.

No temporal protection.

Source: *Elliott, Archibald Samuel, et al. "Checked C: Making C safe by extension." 2018 IEEE Cybersecurity Development (SecDev). IEEE, 2018.*

Other Mitigations

No Hardware Side-Channel Resilience!

Prototype Results

Prototype Results

Average Performance Overheads

~2.11x overhead

spec

2017

- C only subset of programs.

~1.4x overhead

NGINX

- 2019 HTTP Archive Web Almanac workload.

~3.15x overhead

JS

((o) Duktape

- Google Chrome's Octane 2 Benchmark Suite

~2.48x overhead

wolfSSL

- Included Wolfcrypt benchmarks.

Unsupported Functionality

Unsupported Functionality

- *Inline Assembly*
  - Can be handled with lifting or (un)permute primitives.

Unsupported Functionality

- *Inline Assembly*
  - Can be handled with lifting or (un)permute primitives.

- *Variadic Functions*
  - Invoking functions with `va_list` as an argument (e.g. `vsprintf`) are unsupported.

```
void my_printf(const char *fmt, ...
) {
    char buffer[256];

    va_list ap;
    va_start(ap, fmt);
    vsprintf(buffer, fmt, ap);
    va_end(ap);
}
```

Unsupported Functionality

- *Inline Assembly*
  - Can be handled with lifting or (un)permute primitives.

- *Variadic Functions*
  - Invoking functions with `va_list` as an argument (e.g. `vsprintf`) are unsupported.

```
void my_printf(const char *fmt, ...
) {
    char buffer[256];

    va_list ap;
    va_start(ap, fmt);
    vsprintf(buffer, fmt, ap);
    va_end(ap);
}
```

`va_list` is passed to external functions!

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.
    - Key Features
        - ✓ *Metadata-less:* enabled by permuting based on allocation address and a salt.

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.
  - Key Features
    - ✓ *Metadata-less:* enabled by permuting based on allocation address and a salt.
    - ✓ *Out-of-the-box compatibility with MT code:* due to metadata-less nature.

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.
  - Key Features
    - ✓ *Metadata-less:* enabled by permuting based on allocation address and a salt.
    - ✓ *Out-of-the-box compatibility with MT code:* due to metadata-less nature.
    - ✓ *Compatible with non-SPAM code:* allows incremental adoption.

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.
  - Key Features
    - ✓ *Metadata-less:* enabled by permuting based on allocation address and a salt.
    - ✓ *Out-of-the-box compatibility with MT code:* due to metadata-less nature.
    - ✓ *Compatible with non-SPAM code:* allows incremental adoption.
    - ✓ *Suitable for HW acceleration:* localized changes within the pipeline.

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.
  - Key Features
    - ✓ *Metadata-less:* enabled by permuting based on allocation address and a salt.
    - ✓ *Out-of-the-box compatibility with MT code:* due to metadata-less nature.
    - ✓ *Compatible with non-SPAM code:* allows incremental adoption.
    - ✓ *Suitable for HW acceleration:* localized changes within the pipeline.

Future Work

- C++ support
- HW support (including 32-bit systems)

SPAM (in a nutshell)

- Unified solution to multiple software **and** hardware memory security issues.
    - Key Features
        - ✓ *Metadata-less:* enabled by permuting based on allocation address and a salt.
        - ✓ *Out-of-the-box compatibility with MT code:* due to metadata-less nature.
        - ✓ *Compatible with non-SPAM code:* allows incremental adoption.
        - ✓ *Suitable for HW acceleration:* localized changes within the pipeline.

Future Work

- C++ support
- HW support (including 32-bit systems)

PAPER

Currently available upon request!

Checkout our technical report on Arxiv!
https://arxiv.org/abs/2007.13808