Verification of OS-level Cache Management

BU

Renato Mancuso

Mentor Graphics

Sagar Chaki

OSPERT 2018

Goal + Approach

Colored Lockdown
for deterministic cache management

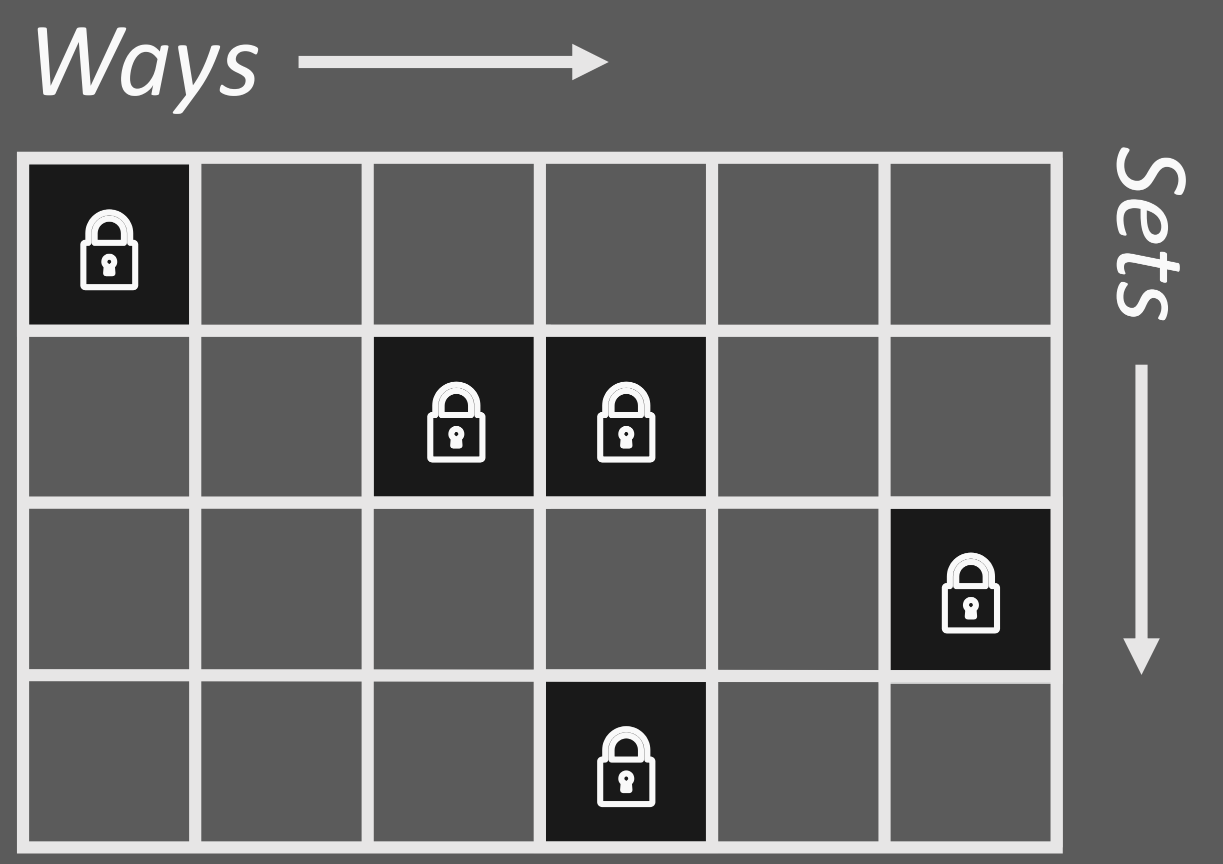
C source code

via CBMC

Linux Kernel Module

Last Level Cache

Management Model



- ✔ Addresses all the sources of interference
- ✔ Converts the LLC cache in a deterministic object at the granularity of a single memory page
- ✔ Allows the use of legacy code
- ✔ Provides flexibility in cache assignment

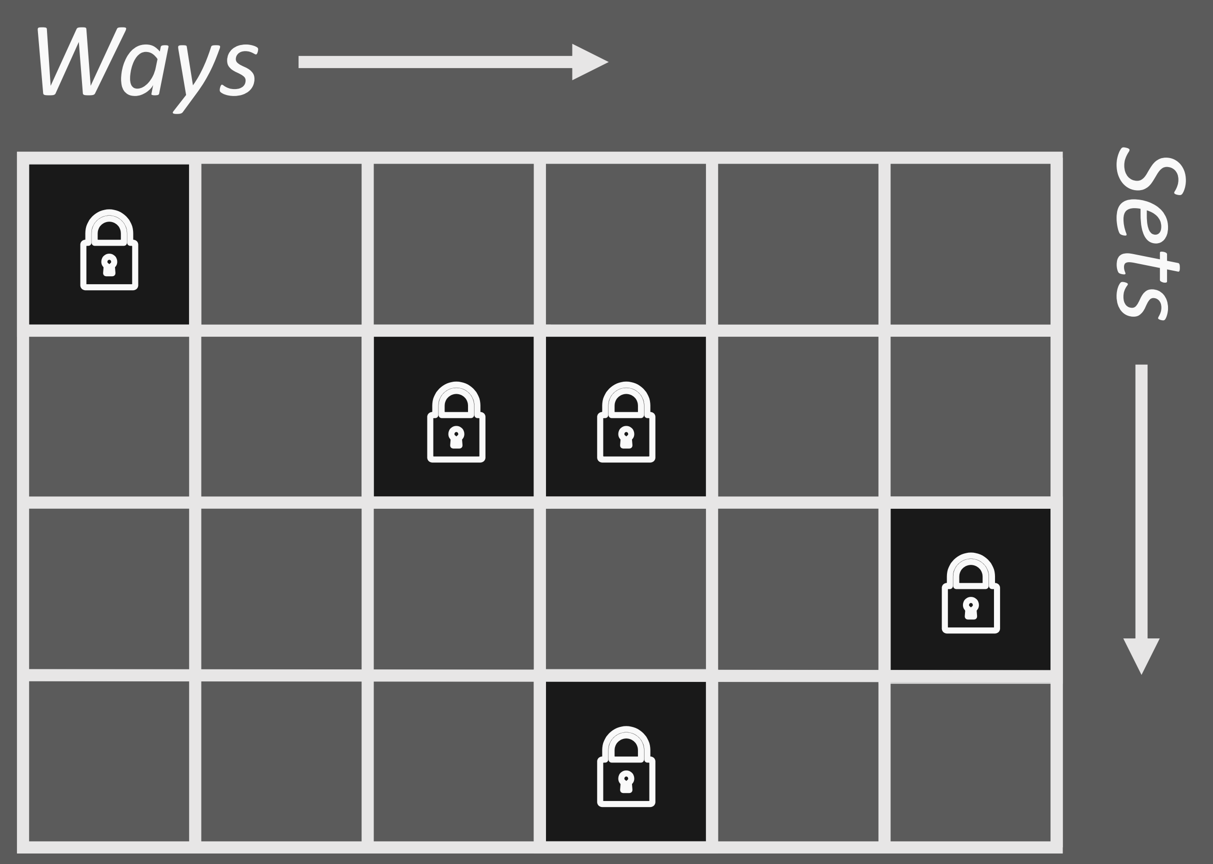
Profile → Remap → Allocate

Background

Colored Lockdown

Last Level Cache

Management Model



- Addresses all the sources of interference
- Converts the LLC cache in a deterministic object at the granularity of a single memory page
- Allows the use of legacy code
- Provides flexibility in cache assignment

Profile → Remap → Allocate

100% hits
on allocated pages

100% misses
on non-allocated pages

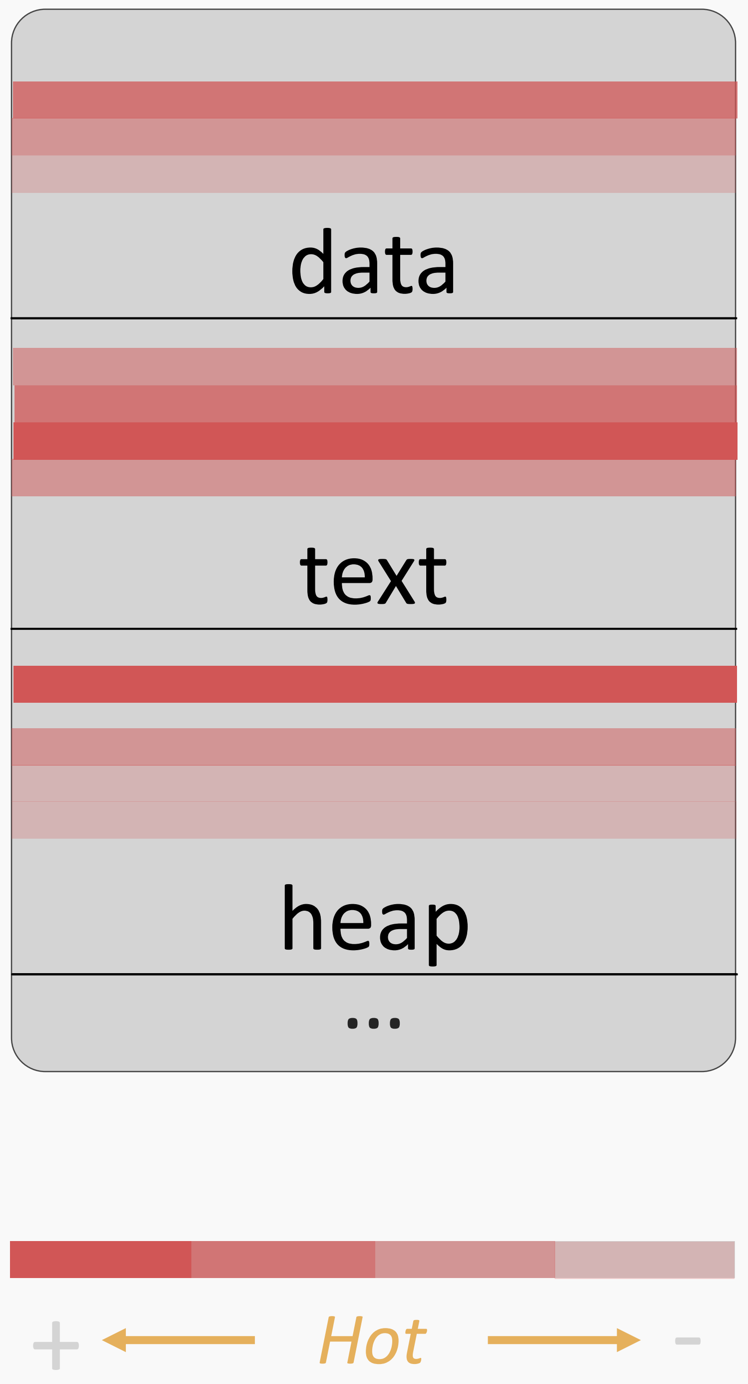
Profile-Driven

Cache Allocation

PROBLEM

Caches are critical,
constrained resources.
Optimal allocation ?

Process Address Space



Location of hot
region(s) is unknown

Absolute virtual
memory addresses
may change

Profile-Driven

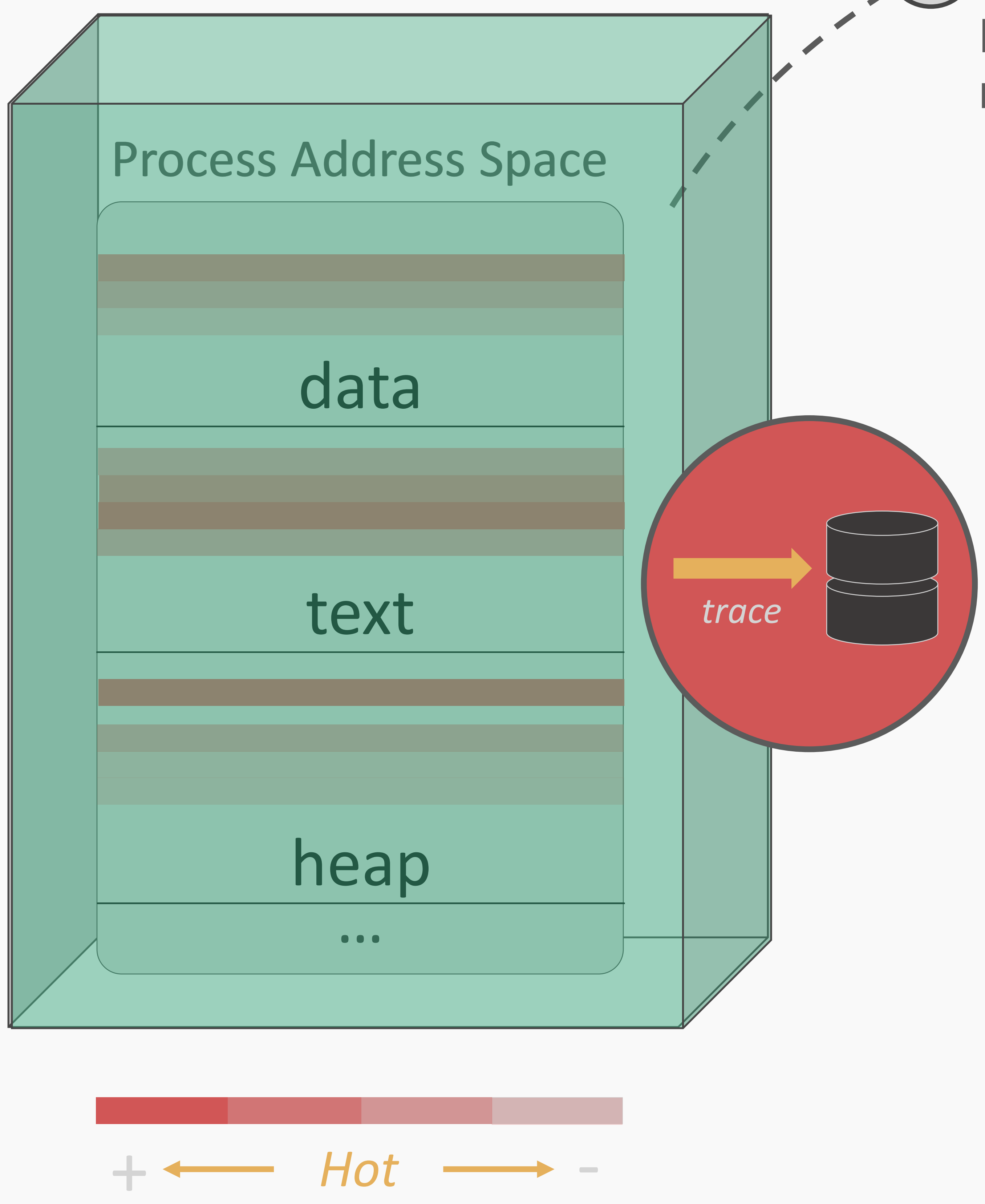
Cache Allocation

PROBLEM

Caches are critical, constrained resources. Optimal allocation ?

PROFILE MEMORY

Extract memory traces and produce memory usage profile.



Location of hot region(s) is unknown

Absolute virtual memory addresses may change

Application Profile

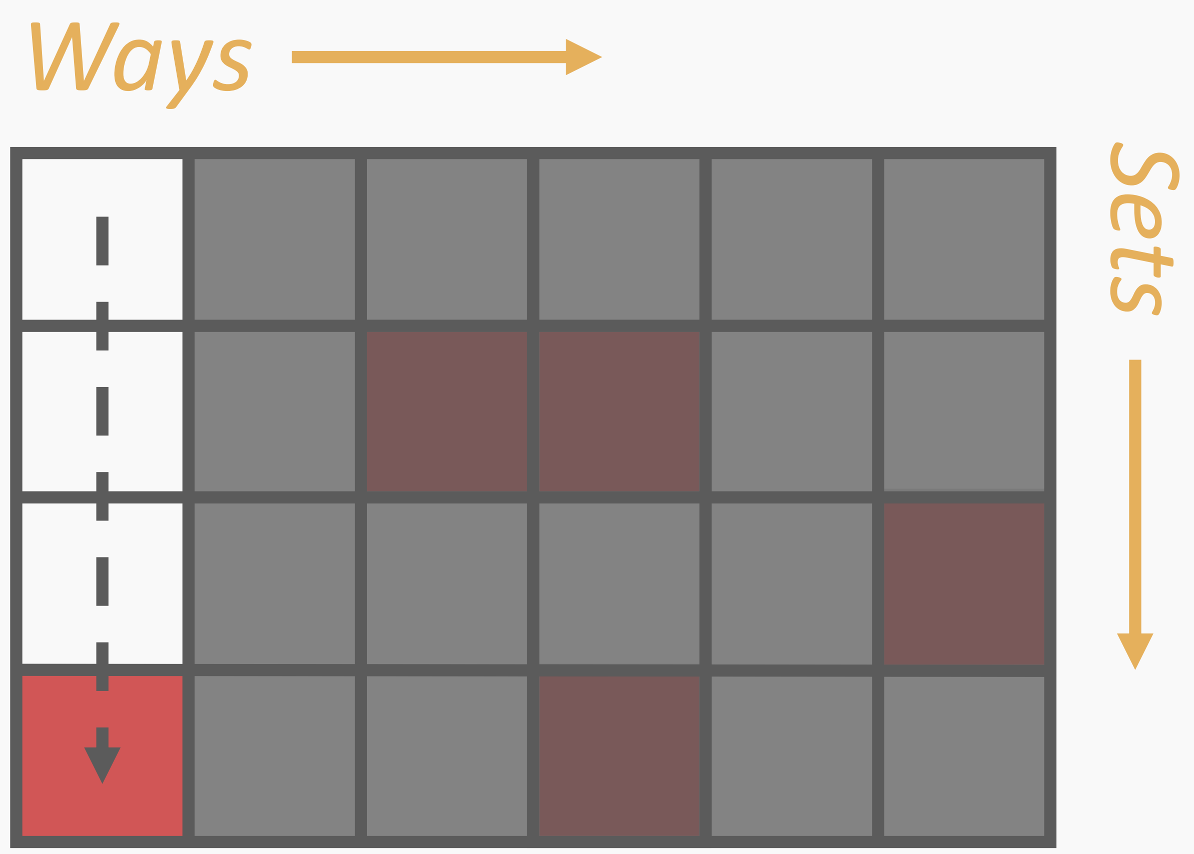
Page	Accesses
A	100 K
B	10 K
C	1 K
D	700
E	500
F	90
C	50
D	10
E	5

threshold

μ

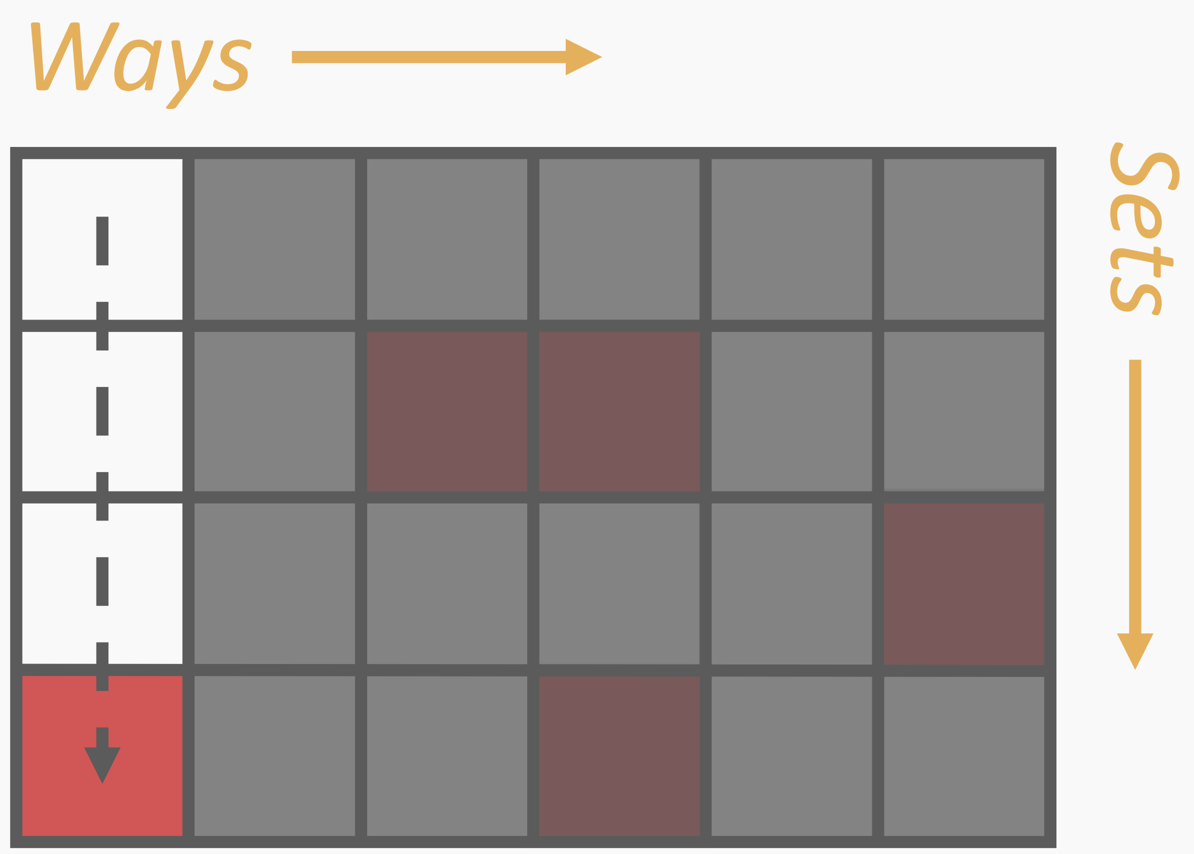
residual cache misses

COLORING



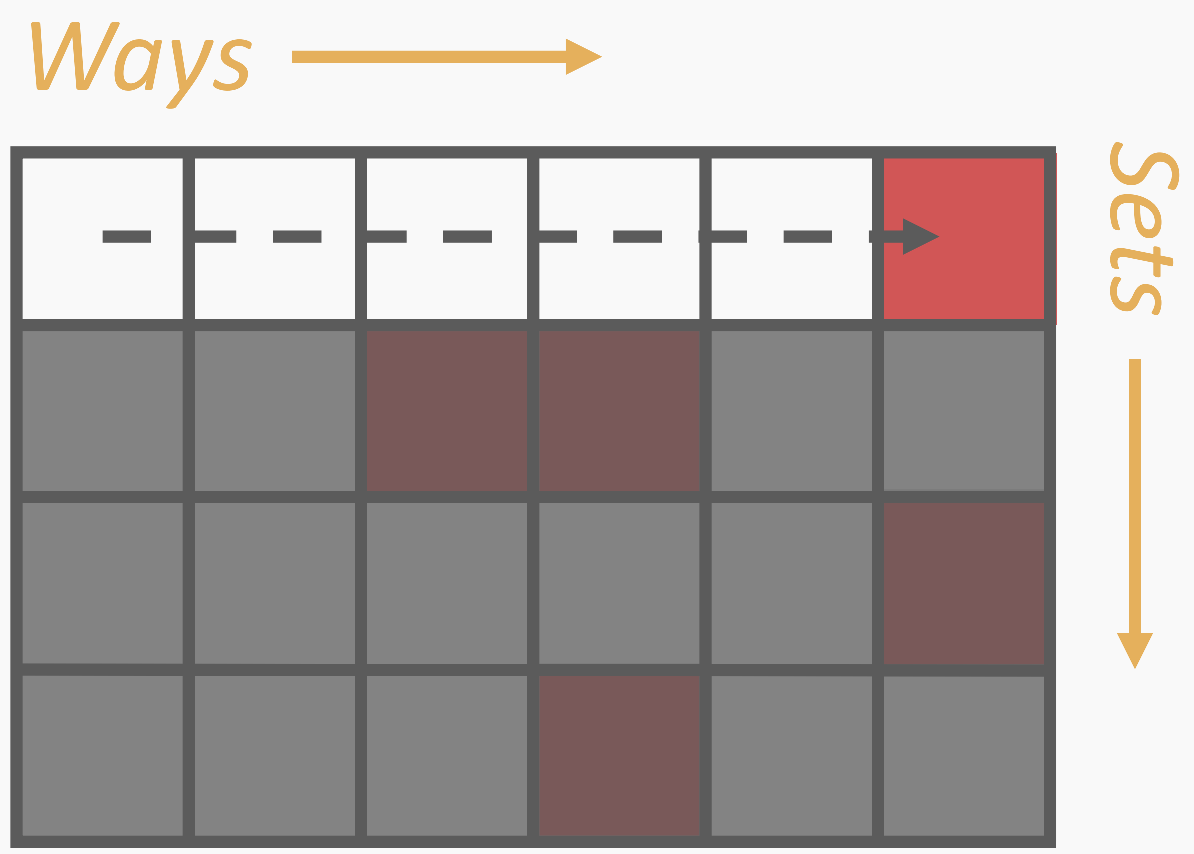
- Leverages on the **virtual** → **physical** translation layer
- Used to **move page mapping** across sets (up/down)
- Transparent to the **application**

COLORING



- Leverages on the **virtual** → **physical** translation layer
- Used to **move page mapping** across sets (up/down)
- Transparent to the **application**

LOCKDOWN



- Uses **architecture-specific** lockdown features
- Used to **allocate pages** on selected ways (left/right)
- Once allocated, pages trigger **cache hits** until deallocation


offline
online


offline
Profile generation
Task tracing + analysis
online


offline
online
Task tracing +
analysis
Profile
generation
Profile load
Color detection +
re-assignment
Cache locking
Task cleanup


offline
online
Task tracing + analysis
Profile generation
Profile load
Color detection + re-assignment
Cache locking
Task cleanup
1
2
3
4
VERIFIED PROPERTIES
If cache is **large enough**, allocation is performed
BU


offline
online
Task tracing +
analysis
Profile
generation
Profile load
Color detection +
re-assignment
Cache locking
Task cleanup
1
2
3
4
VERIFIED PROPERTIES
If allocation is performed, corresponding physical addresses **locked in cache**
If cache is **large enough**, allocation is performed
BU


offline
online
Task tracing + analysis
Profile generation
Profile load
Color detection + re-assignment
Cache locking
Task cleanup
VERIFIED PROPERTIES
1
2
3
4
No **extra** memory locked in cache
If allocation is performed, corresponding physical addresses **locked in cache**
If cache is **large enough**, allocation is performed

offline
online
Task tracing + analysis
Profile generation
Profile load
Color detection + re-assignment
Cache locking
Task cleanup
VERIFIED PROPERTIES
1
2
3
4
All temporary **kernel resources** released at the end
No **extra** memory locked in cache
If allocation is performed, corresponding physical addresses **locked in cache**
If cache is **large enough**, allocation is performed

what was Verified

4 All temporary **kernel resources** released at the end

3 No **extra** memory locked in cache

2 If allocation is performed, corresponding physical addresses **locked in cache**

1 If cache is **large enough**, allocation is performed

1 After locking issued, line is in cache

what was Assumed

what was Verified

4 All temporary **kernel resources** released at the end

3 No **extra** memory locked in cache

2 If allocation is performed, corresponding physical addresses **locked in cache**

1 If cache is **large enough**, allocation is performed

1 After locking issued, line is in cache

2 Profile is correct

what was Assumed

what was Verified

4. All temporary **kernel resources** released at the end
3. No **extra** memory locked in cache
2. If allocation is performed, corresponding physical addresses **locked in cache**
1. If cache is **large enough**, allocation is performed

what was Assumed

1. After locking issued, line is in cache
2. Profile is correct
3. Kernel descriptors correct

what was **Verified**

4. All temporary **kernel resources** released at the end

3. No **extra** memory locked in cache

2. If allocation is performed, corresponding physical addresses **locked in cache**

1. If cache is **large enough**, allocation is performed

1. After locking issued, line is in cache

2. Profile is correct

3. Kernel descriptors correct

4. Kernel routines correct

what was **Assumed**

what was Verified

4 All temporary **kernel resources** released at the end

3 No **extra** memory locked in cache

2 If allocation is performed, corresponding physical addresses **locked in cache**

1 If cache is **large enough**, allocation is performed

1. After locking issued, line is in cache
2. Profile is correct
3. Kernel descriptors correct
4. Kernel routines correct
5. Initial status of lockdown bit is known

what was Assumed

Verification
Boundaries

Verified
Logic

Verification **Boundaries**

User-Space

Template Generic Profile

Verified Logic

Verification **Boundaries**

Template
Generic
Profile

User-Space

Verified
Logic

Hardware

Instantiate
memory
model

Abstract
cache
model

Verification **Boundaries**

Set Associative Cache

Cache Model

& Locking

Set Associative Cache

```
typedef struct {
        void * addr;
        char locked;
} cache_line_t;
```

Cache Model

& Locking

Set Associative Cache

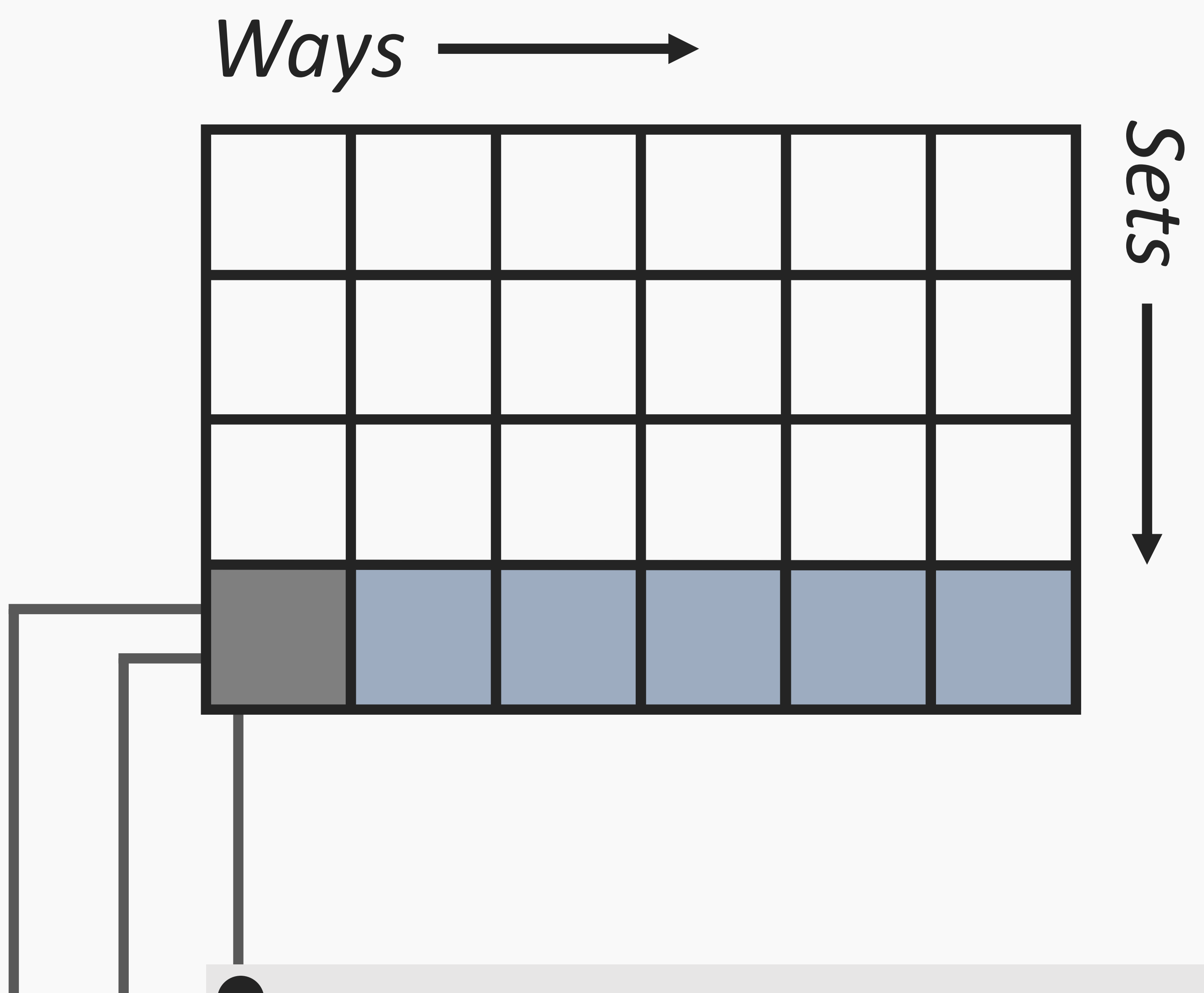
```
typedef cache_line_t cache_set_t [CACHE_ASSOC];
```

```
typedef struct {
        void * addr;
        char locked;
} cache_line_t;
```

Cache Model

& Locking

Set Associative Cache




```
typedef cache_set_t cache_t [CACHE_NSETS];
```

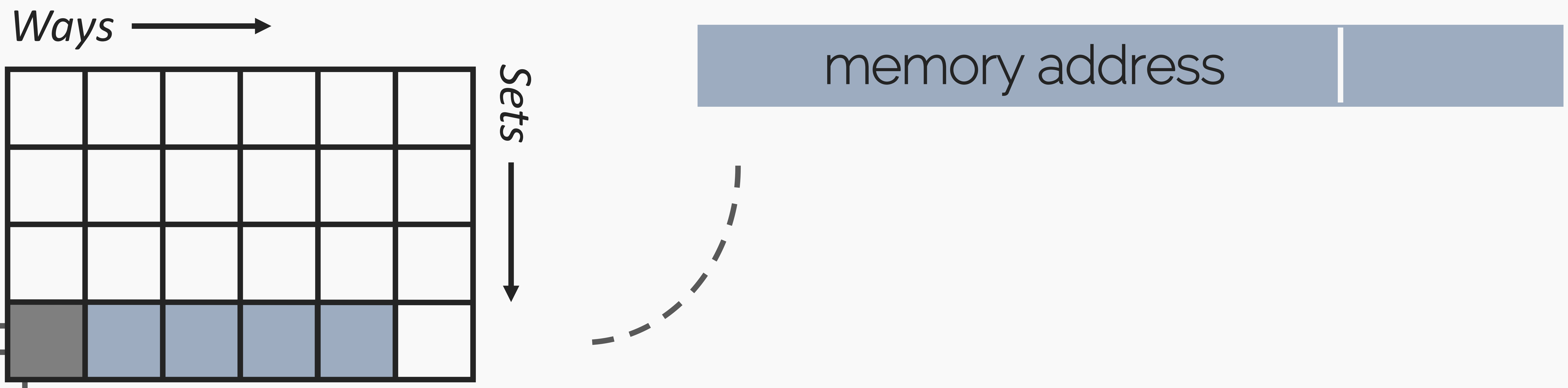
```
typedef cache_line_t cache_set_t [CACHE_ASSOC];
```

```
typedef struct {
        void * addr;
        char locked;
} cache_line_t;
```

Cache Model

& Locking

Set Associative Cache



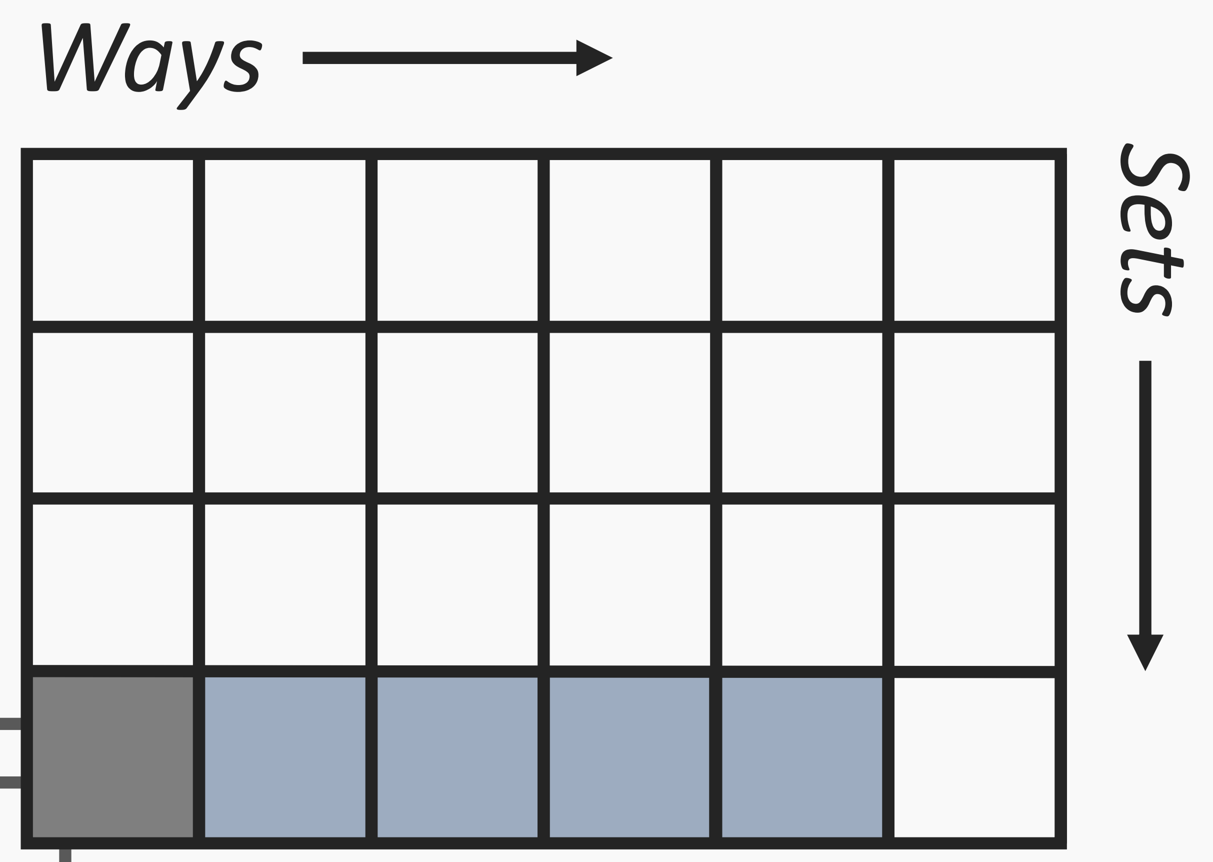

```
typedef cache_set_t cache_t [CACHE_NSETS];

typedef cache_line_t cache_set_t [CACHE_ASSOC];

typedef struct {
        void * addr;
        char locked;
} cache_line_t;
```

Cache Model
& Locking

Set Associative Cache



memory address | lock bit

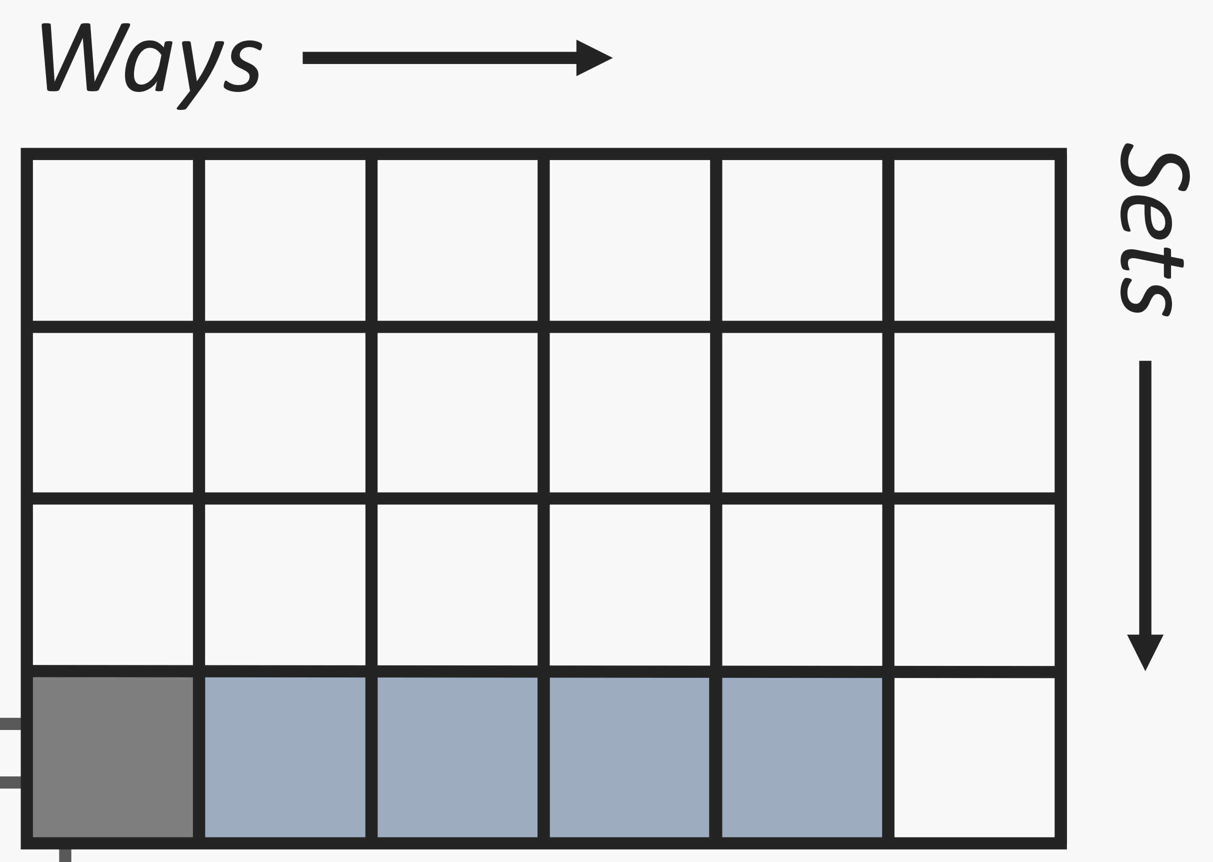
```
typedef cache_set_t cache_t [CACHE_NSETS];
```

```
typedef cache_line_t cache_set_t [CACHE_ASSOC];
```

```
typedef struct {
        void * addr;
        char locked;
} cache_line_t;
```

Cache Model
& Locking

Set Associative Cache



memory address | lock bit

DCBTLS in HW

Data Cache Block Touch & Lock Set

```
typedef cache_set_t cache_t [CACHE_NSETS];

typedef cache_line_t cache_set_t [CACHE_ASSOC];

typedef struct {
        void * addr;
        char locked;
} cache_line_t;
```

Cache Model

& Locking

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

Abstraction of **Kernel Routine**

main parameters
(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of
Kernel Routine

{

}

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters

(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
  struct page * page_ptr;

}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct task_struck** * tsk
- **struct mm_struct** * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct page** ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct** **task_struck** * tsk
- **struct** **mm_struct** * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** **page** ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct** **task_struck** * tsk
- **struct** **mm_struct** * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** **page** ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters

(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
}
```

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters

(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
    assert(mm == tsk->mm);
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
    assert(mm == tsk->mm);
    page_ptr = __CPROVER_ui_void_ptr(...);
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
    assert(mm == tsk->mm);
    page_ptr = __CPROVER_ui_void_ptr(...);
    __CPROVER_assume(page_ptr >= mem_map &&
                     page_ptr < (mem_map + MAX_PAGES));
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters

(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
    assert(mm == tsk->mm);
    page_ptr = __CPROVER_ui_void_ptr(...);
    __CPROVER_assume(page_ptr >= mem_map &&
                page_ptr < (mem_map + MAX_PAGES));
    __CPROVER_assume(ALIGNED_TO(page_ptr,
                sizeof(struct page)));
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters
(as of kernel 3.0-rc7)

- **struct** task_struck * tsk
- **struct** mm_struct * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct** page ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
    assert(mm == tsk->mm);
    page_ptr = __CPROVER_ui_void_ptr(...);
    __CPROVER_assume(page_ptr >= mem_map &&
                page_ptr < (mem_map + MAX_PAGES));
    __CPROVER_assume(ALIGNED_TO(page_ptr,
                sizeof(struct page)));
    *pages = page_ptr;
}
```

BU

get_user_pages(...)

used to pin physical pages, returns page descriptors from virtual addresses

main parameters

(as of kernel 3.0-rc7)

- **struct task_struck** * tsk
- **struct mm_struct** * mm
- **unsigned long** start
- **unsigned long** nr_pages
- **int** write, **int** force
- **struct page** ** pages

Abstraction of **Kernel Routine**

```
{
    struct page * page_ptr;
    assert(nr_pages == 1);
    assert(write == 0);
    assert(force == 0);
    assert(tsk == current);
    assert(mm == tsk->mm);
    page_ptr = __CPROVER_ui_void_ptr(...);
    __CPROVER_assume(page_ptr >= mem_map &&
                page_ptr < (mem_map + MAX_PAGES));
    __CPROVER_assume(ALIGNED_TO(page_ptr,
                sizeof(struct page)));
    *pages = page_ptr;
    return 1;
}
```

BU

Evaluation

Memory Instance #1

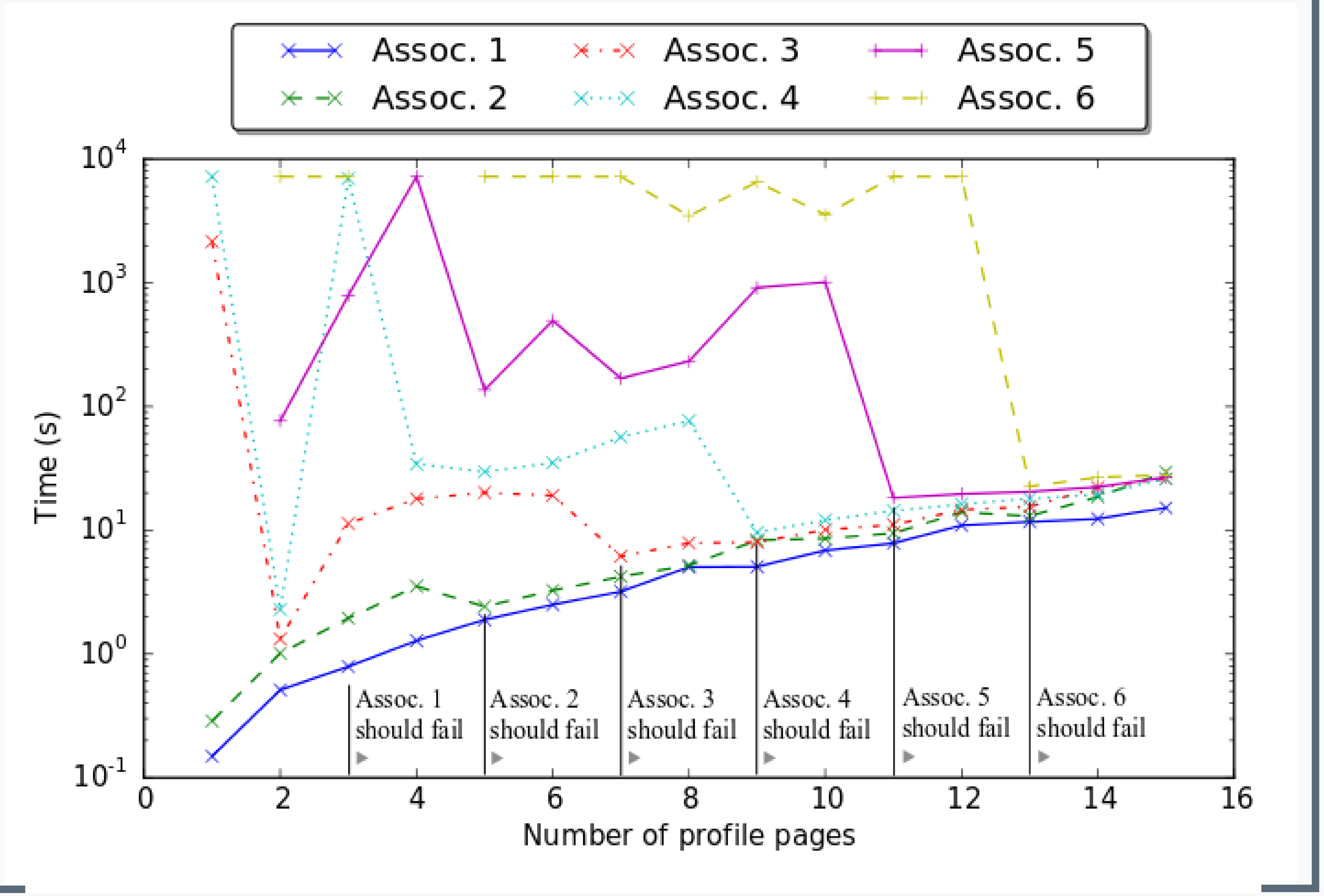
From cache controller's perspective

Tag (23) | Index (3) | Offset (6)

From OS's perspective

Page Frame Number (24) | Offset (8)

Color (1)



Evaluation

Memory Instance #1

From cache controller's perspective

Tag (23) | Index (3) | Offset (6)

From OS's perspective

Page Frame Number (24) | Offset (8)

Color (1)

Memory Instance #2

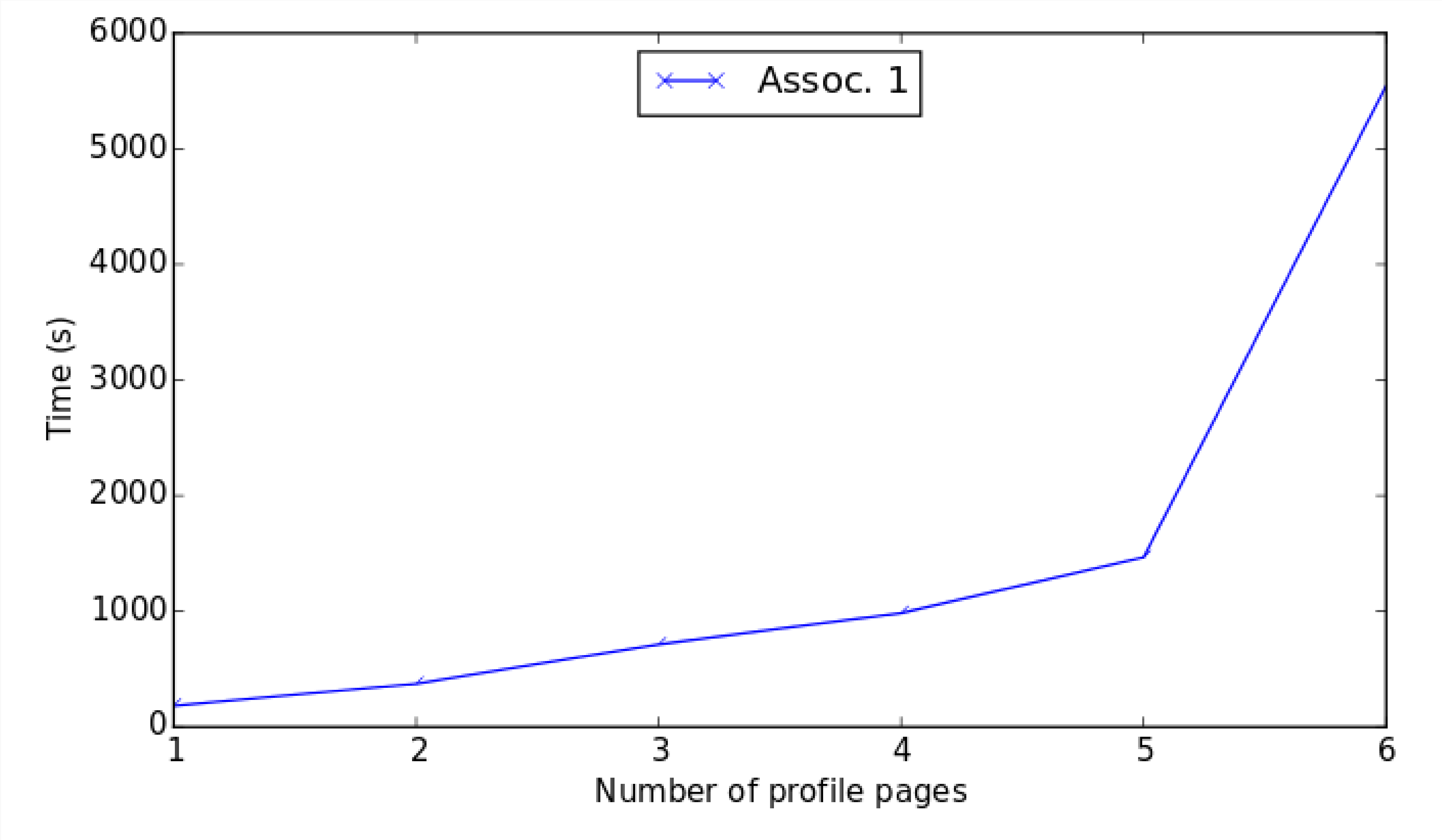
From cache controller's perspective

Tag (16) | Index (10) | Offset (6)

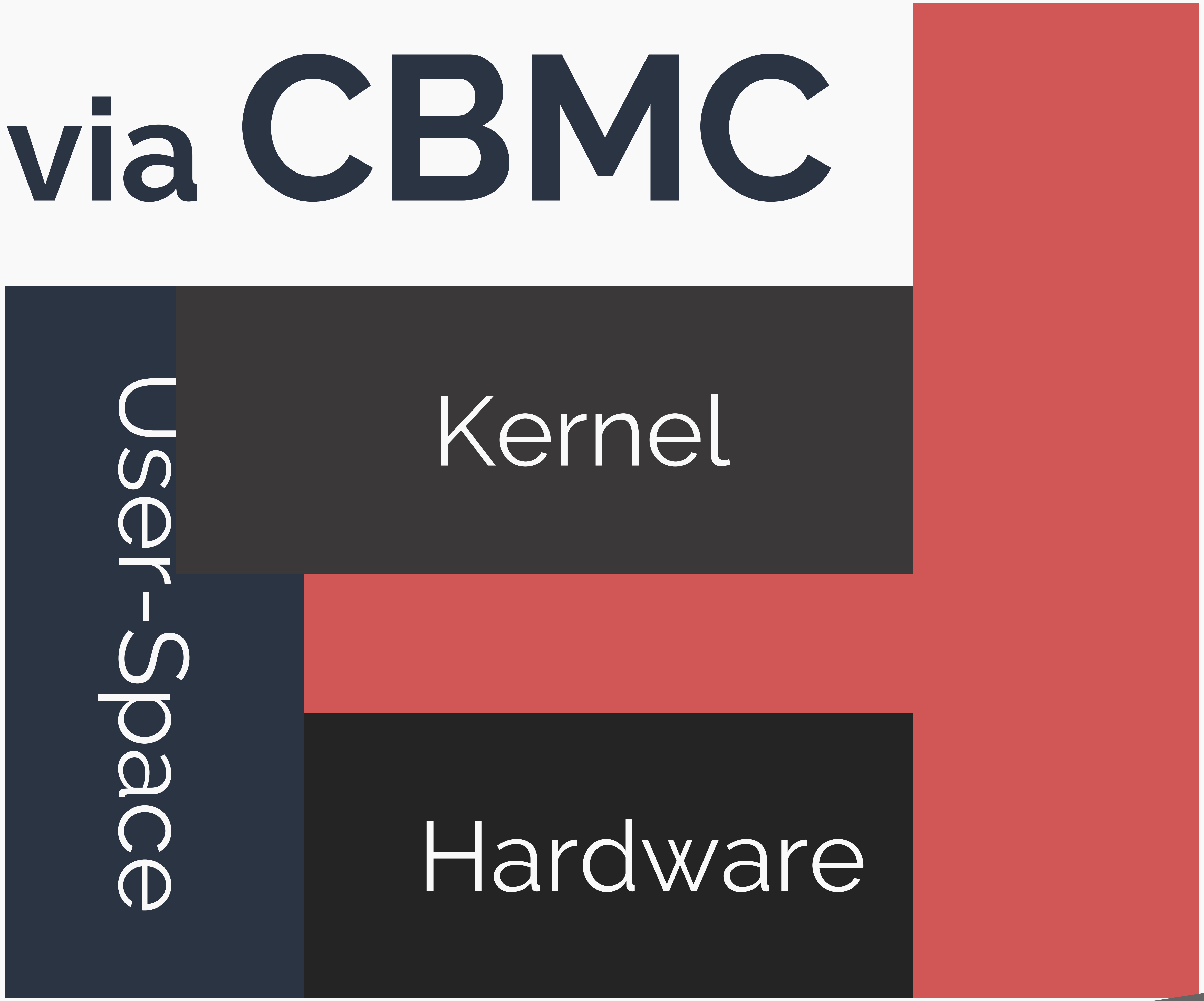
From OS's perspective

Page Frame Number (20) | Offset (12)

Color (4)

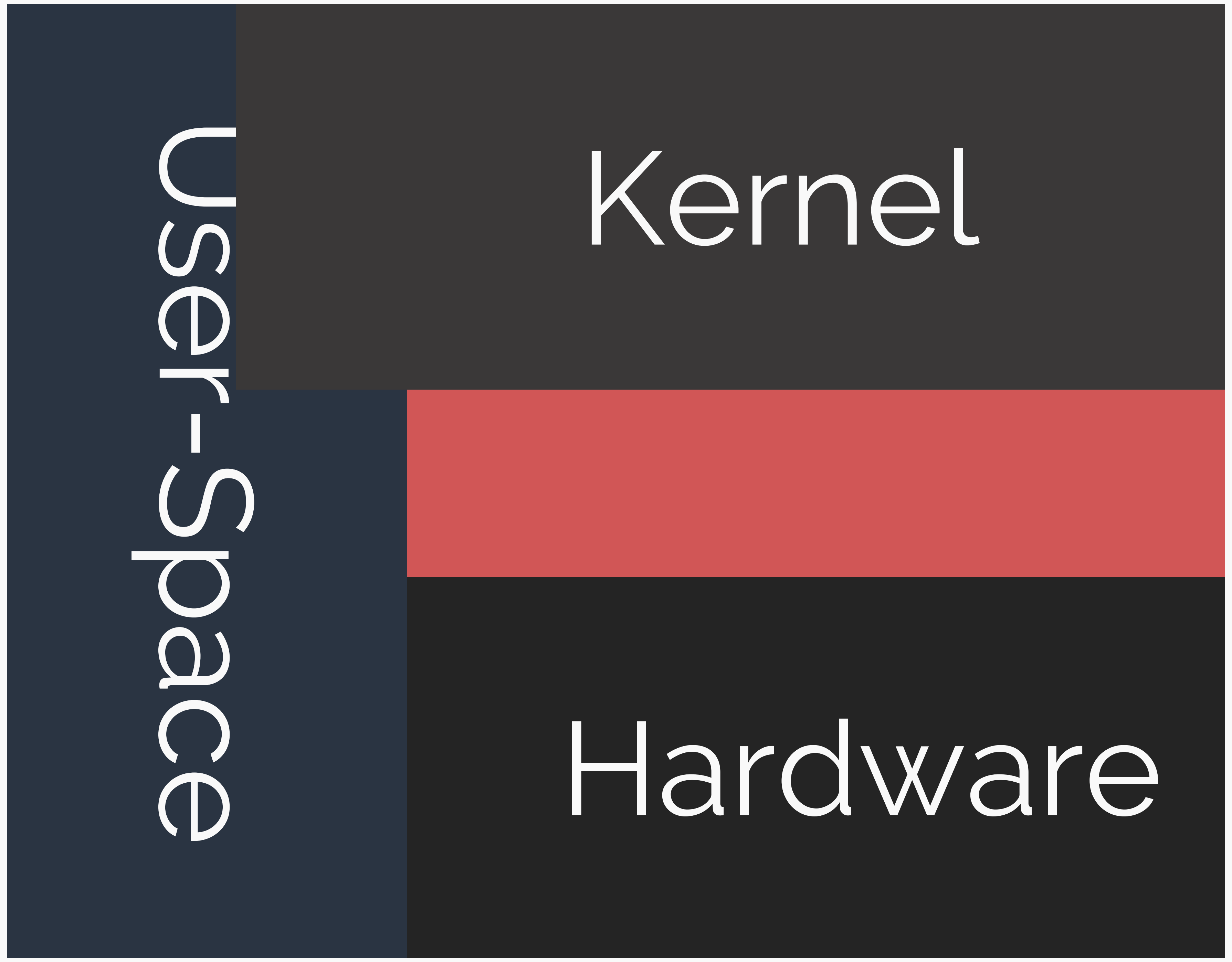


Colored Lockdown via CBMC



Summary

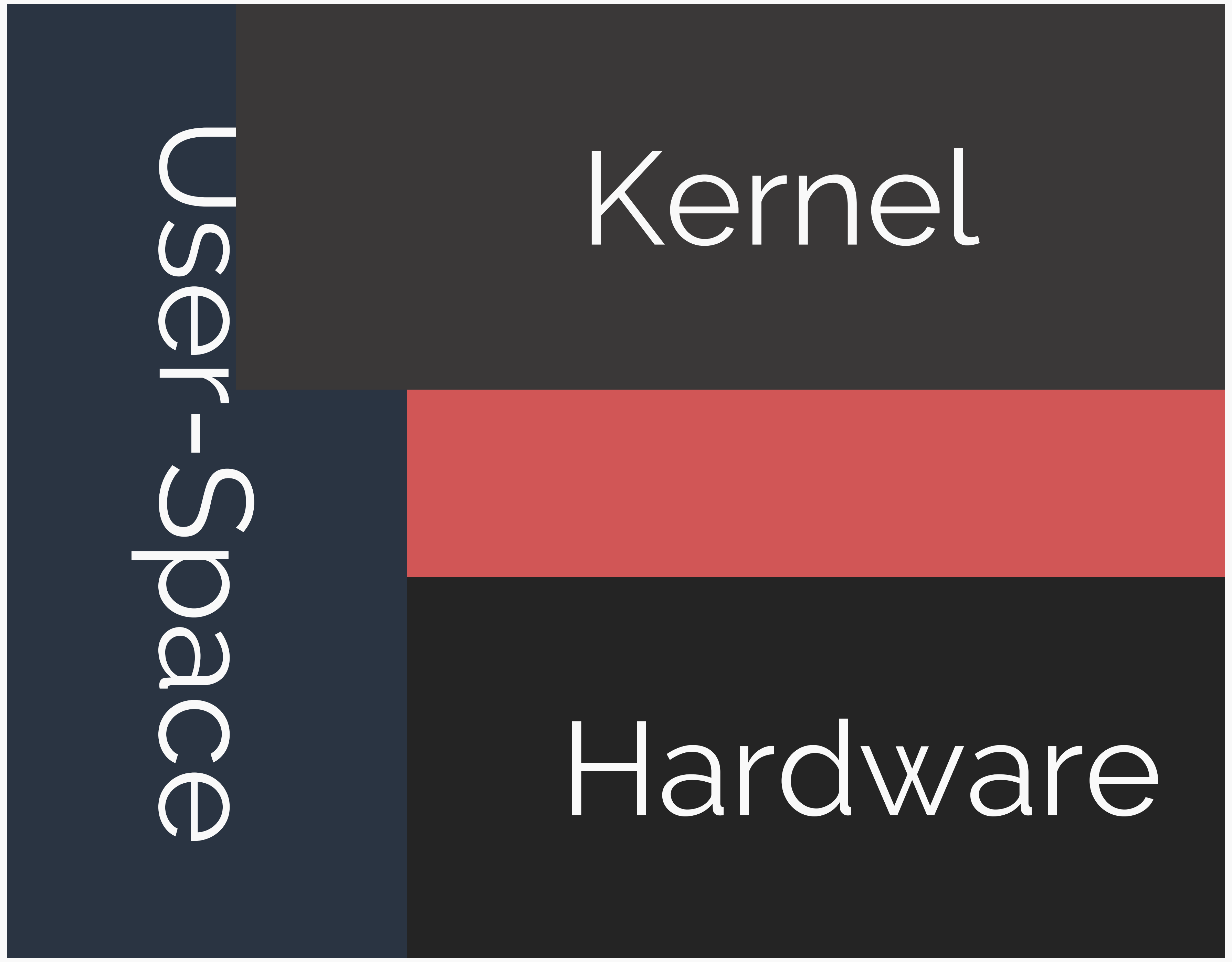
Colored Lockdown via CBMC



succesful with
realistic memory model

Summary

Colored Lockdown via CBMC



scalability needed

+ + + + + + +

Summary

Thanks.

rmancuso@bu.edu