A Compacting Real-Time Memory Management System

Silviu S. Craciunas, Christoph M. Kirsch, Hannes Payer, Ana Sokolova, Horst Stadler, Robert Staudinger

Hannes Payer

Computational Systems Group, University of Salzburg

July 1, 2008

Overview

Motivation

Traditional dynamic memory management systems are typically non-deterministic:

- unpredictable response times of memory operations
- unpredictable memory fragmentation

$\Rightarrow$ Dynamic memory management systems are typically not used in time-critical software components
(hard real-time systems, device drivers . . .)

Predictable Memory Management System

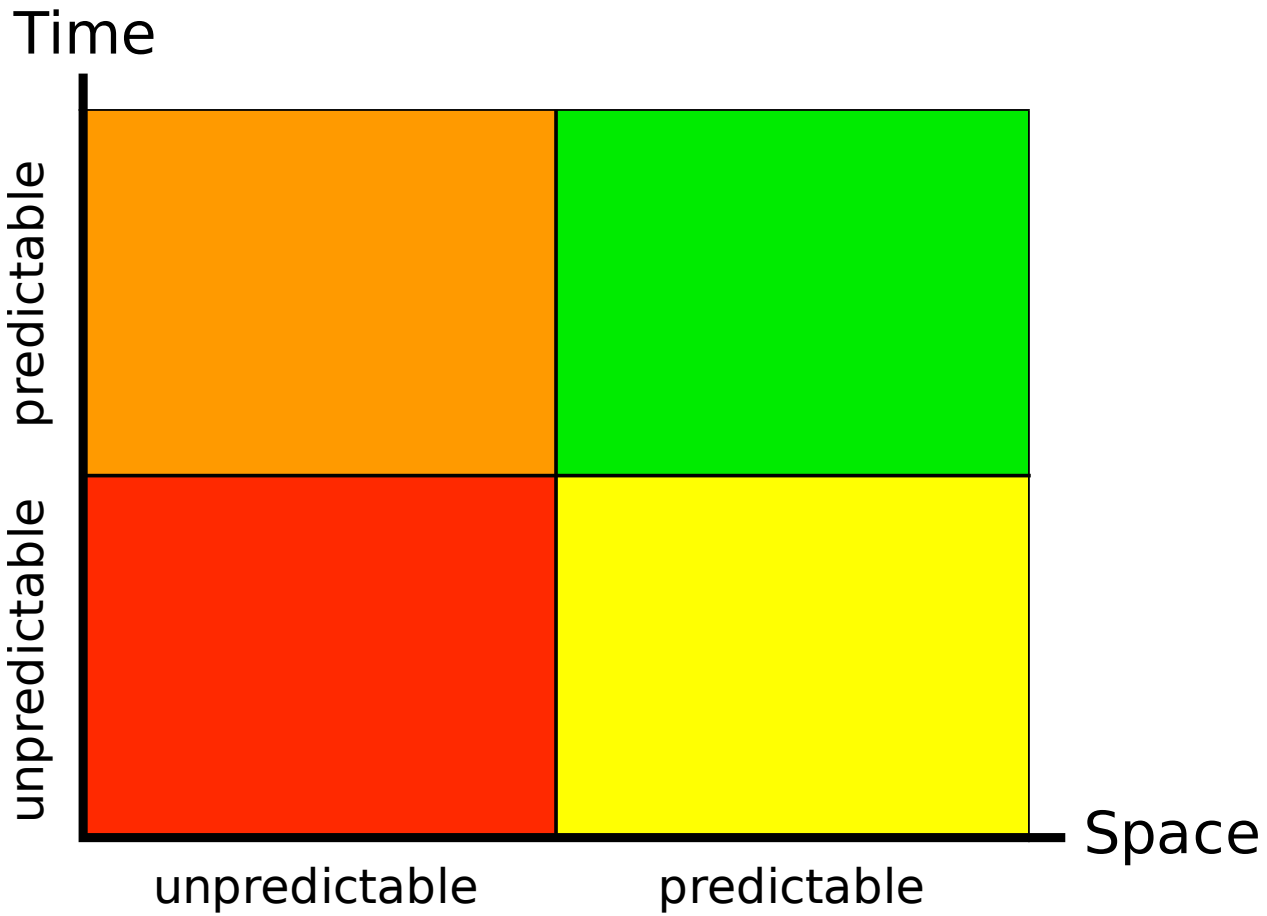
Predictability in Time

The time a memory management operation takes is determined by the size of the object involved in the operation (allocation, deallocation, and dereference).

Predictability in Space

The number of actual allocations together with their sizes (not the order of invocations) determines how many more allocations of a given size will succeed before running out of memory.

Solution Space



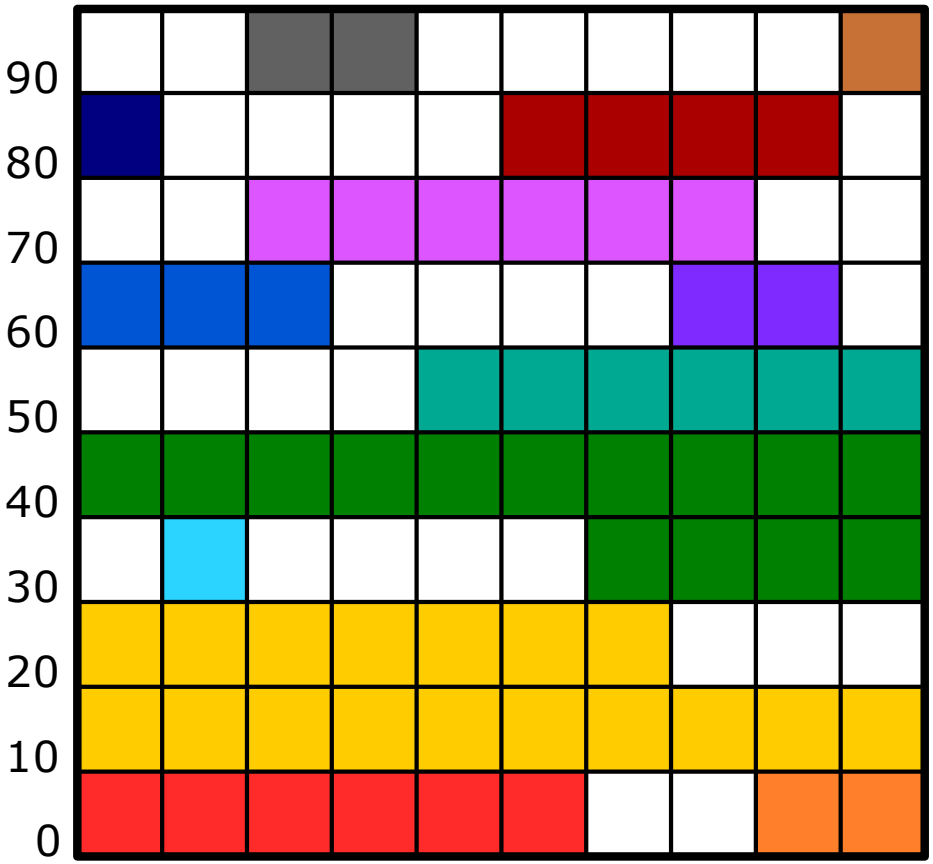
What We Want?

A memory management system predictable in time and space
(component of the real-time operating system Tiptoe)

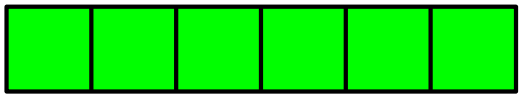
Properties:

- malloc(n) takes at most $\mathcal{O}(n)$ time
- free(n) takes at most $\mathcal{O}(n)$ time
- memory access (dereference) takes small constant time
- small and predictable memory fragmentation bound

Fragmentation Problem



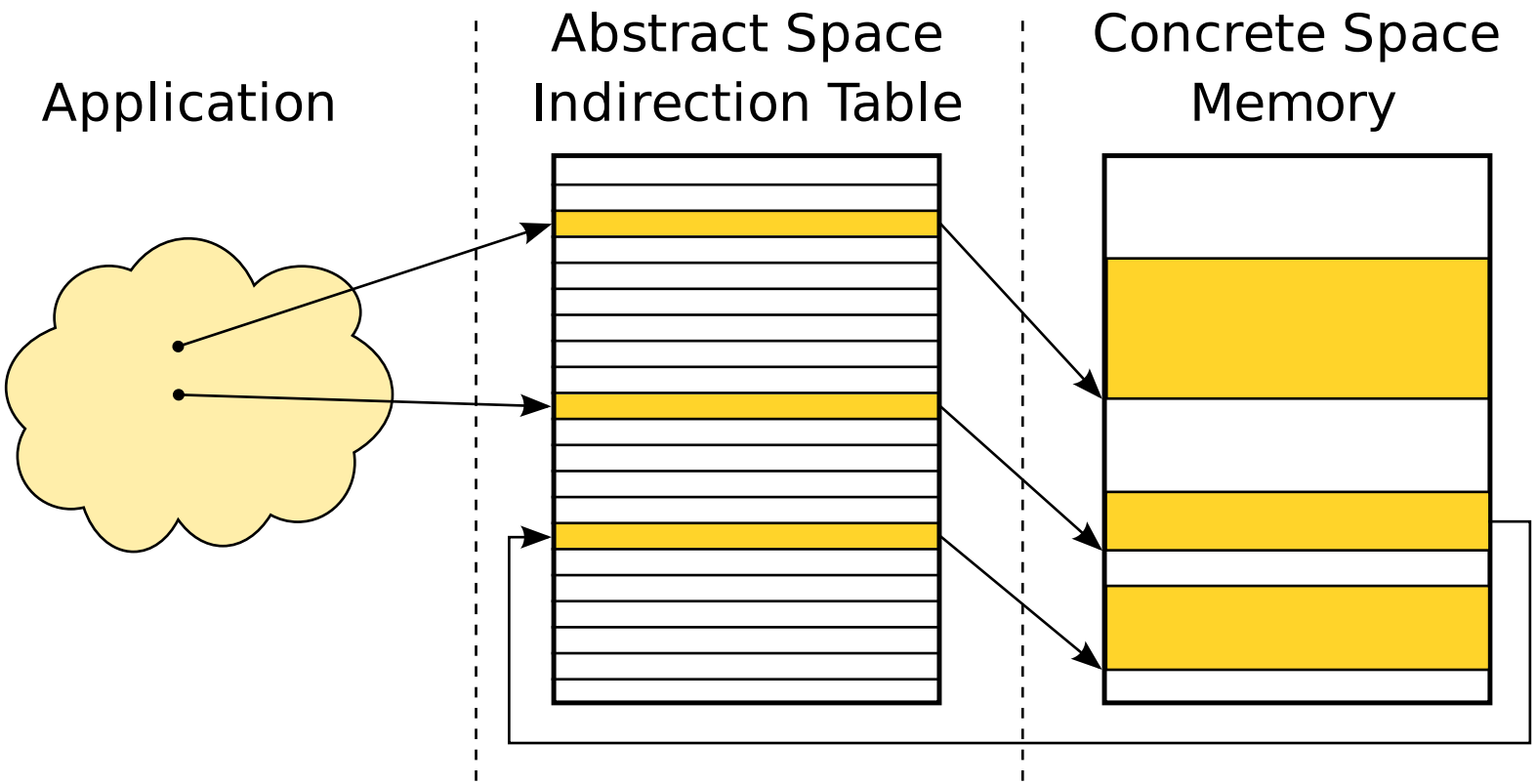
35% free



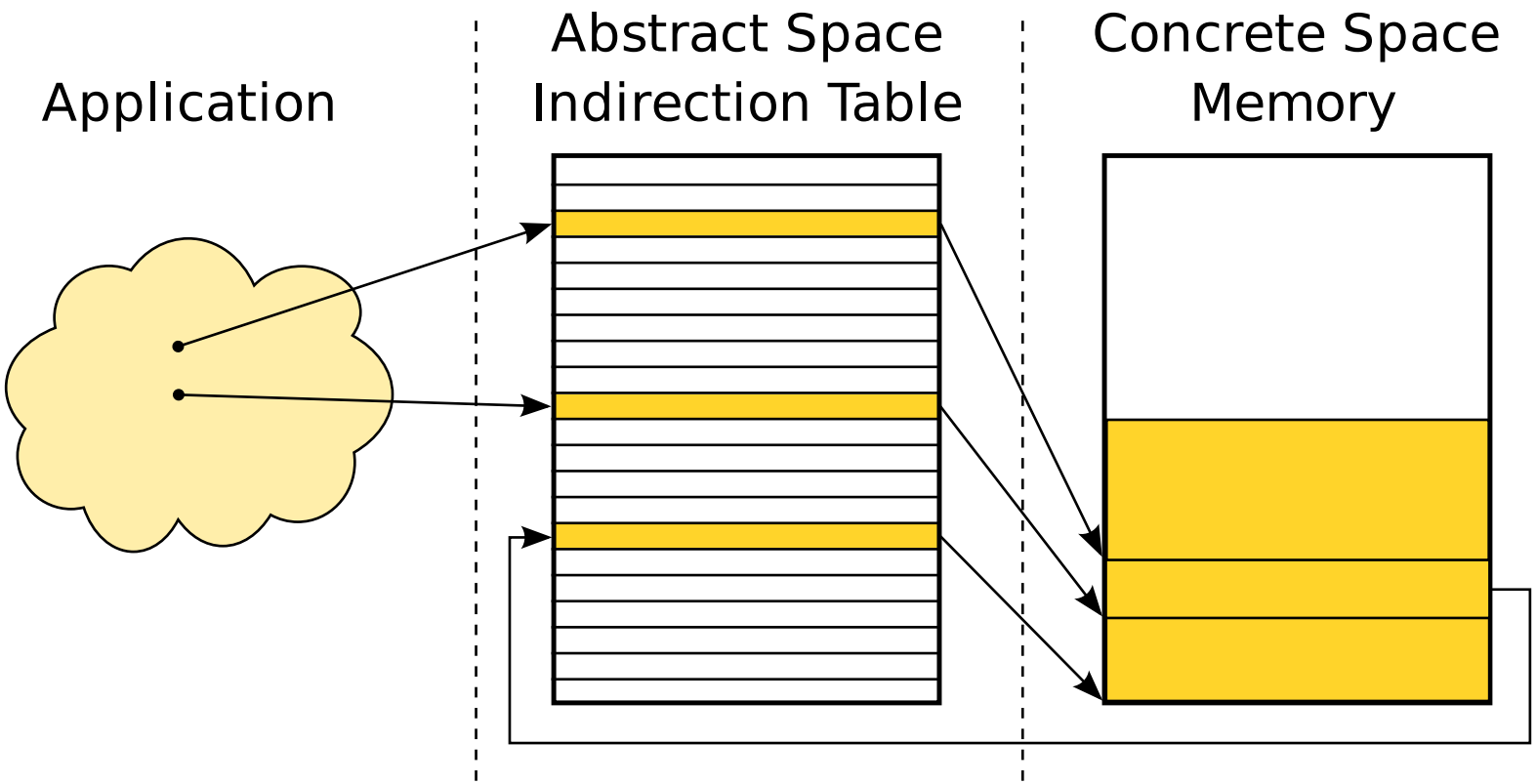
not allocatable

fragmentation in a contiguous space ⇒
compaction ⇒ reference updates

Solution to Reference Updates



Solution to Reference Updates



Compaction

Trade-Off:

speed versus memory fragmentation

Requirement:

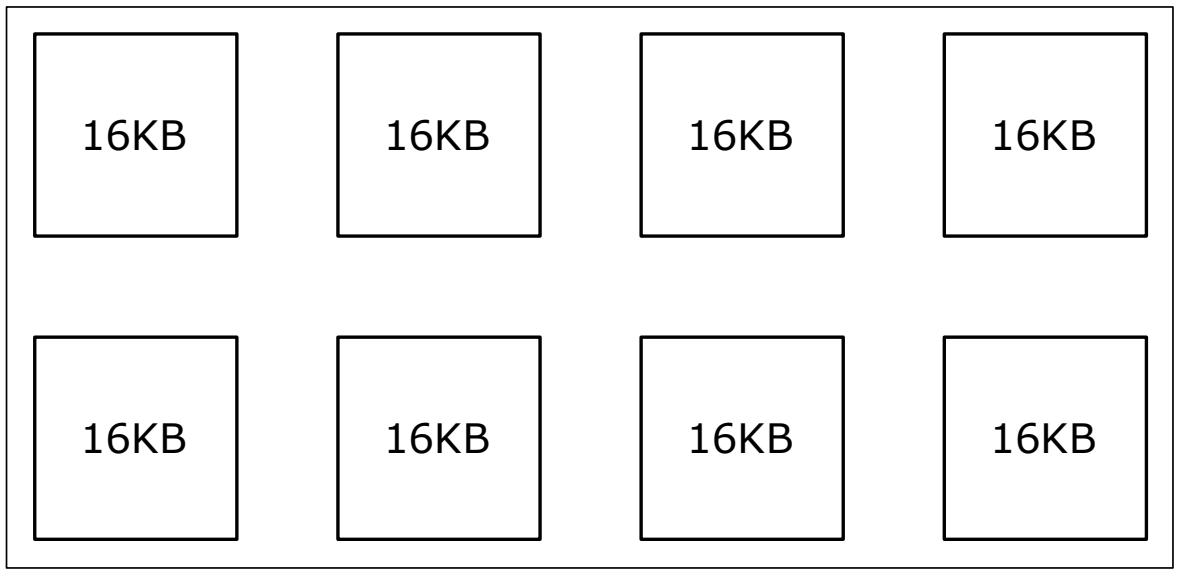
keep speed and memory fragmentation bounded and predictable

2 Extreme Non-Solutions:

- keep memory perfectly compact
- perform memory operations in constant time without considering memory fragmentation

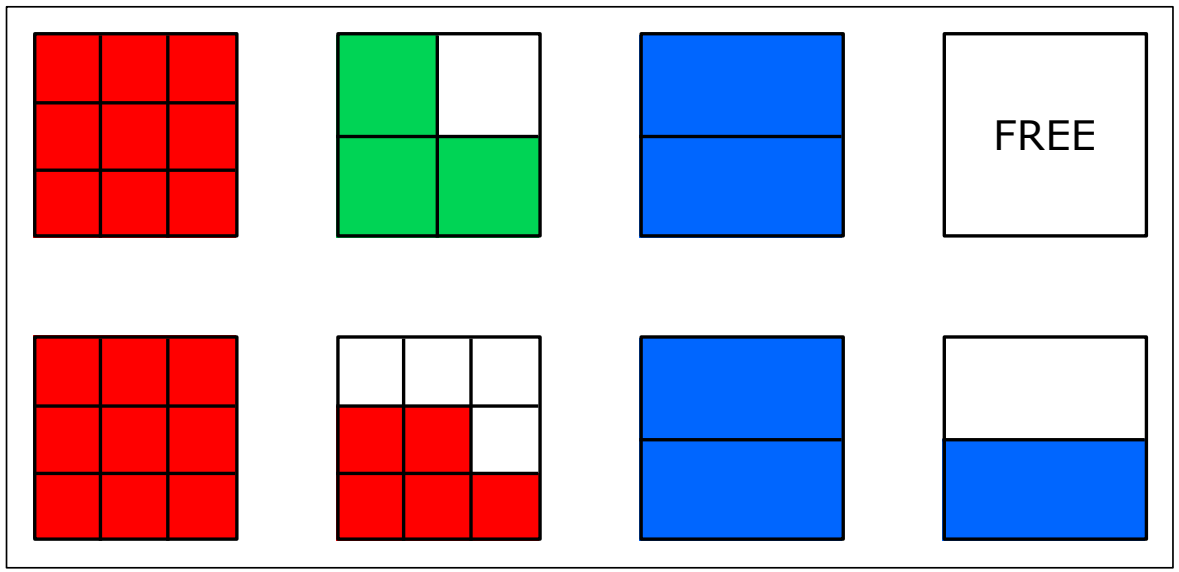
Concrete Address Space

- concrete address space is divided into **pages** of equal size



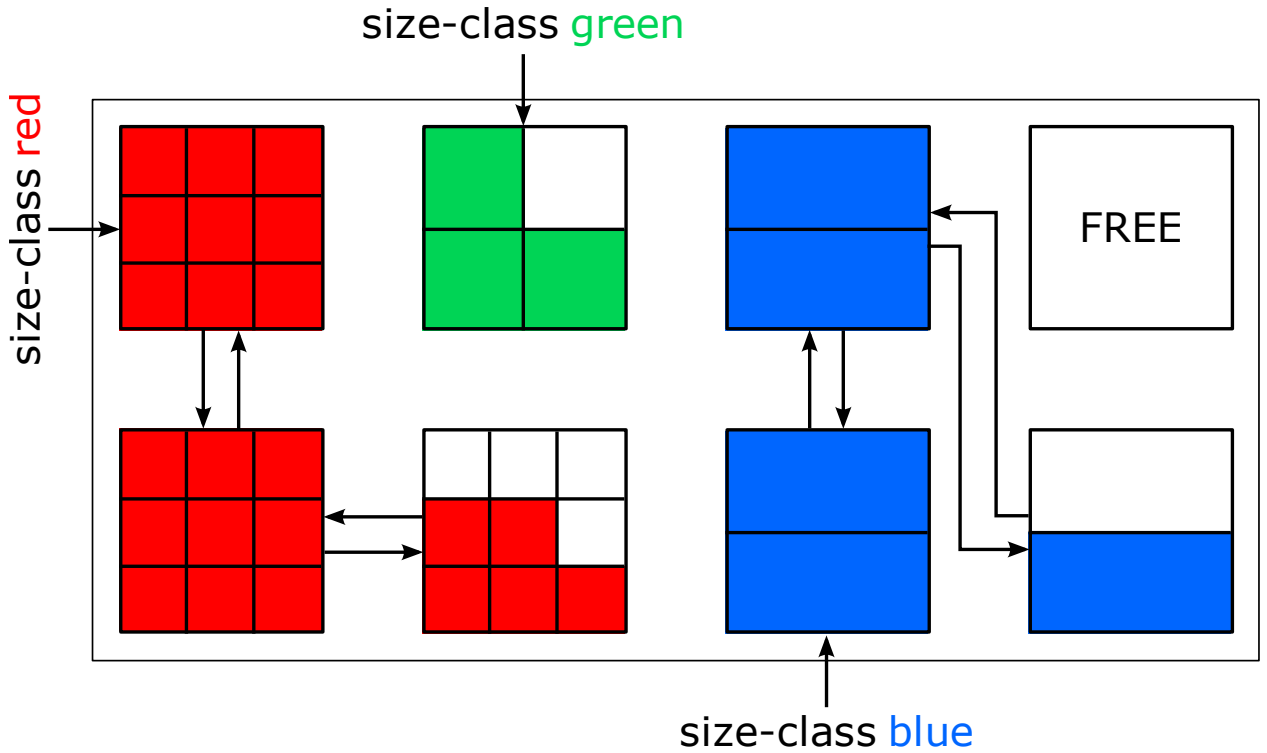
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**



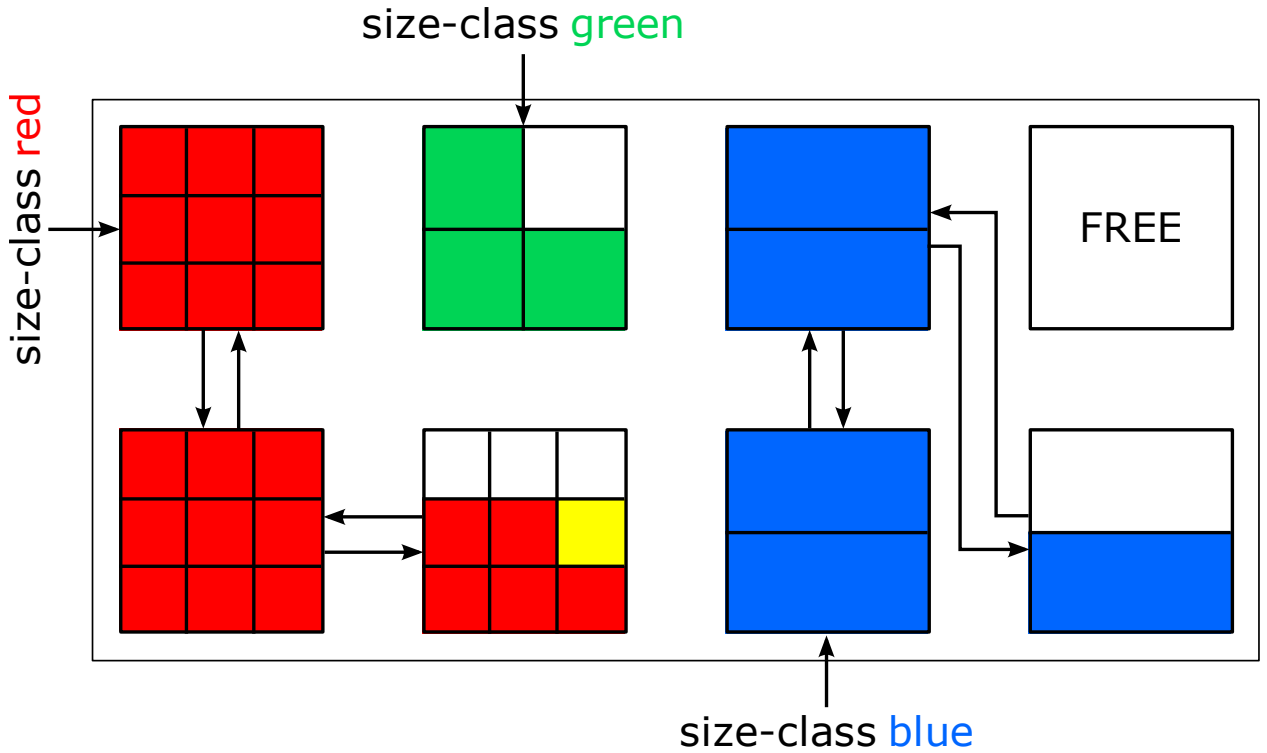
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



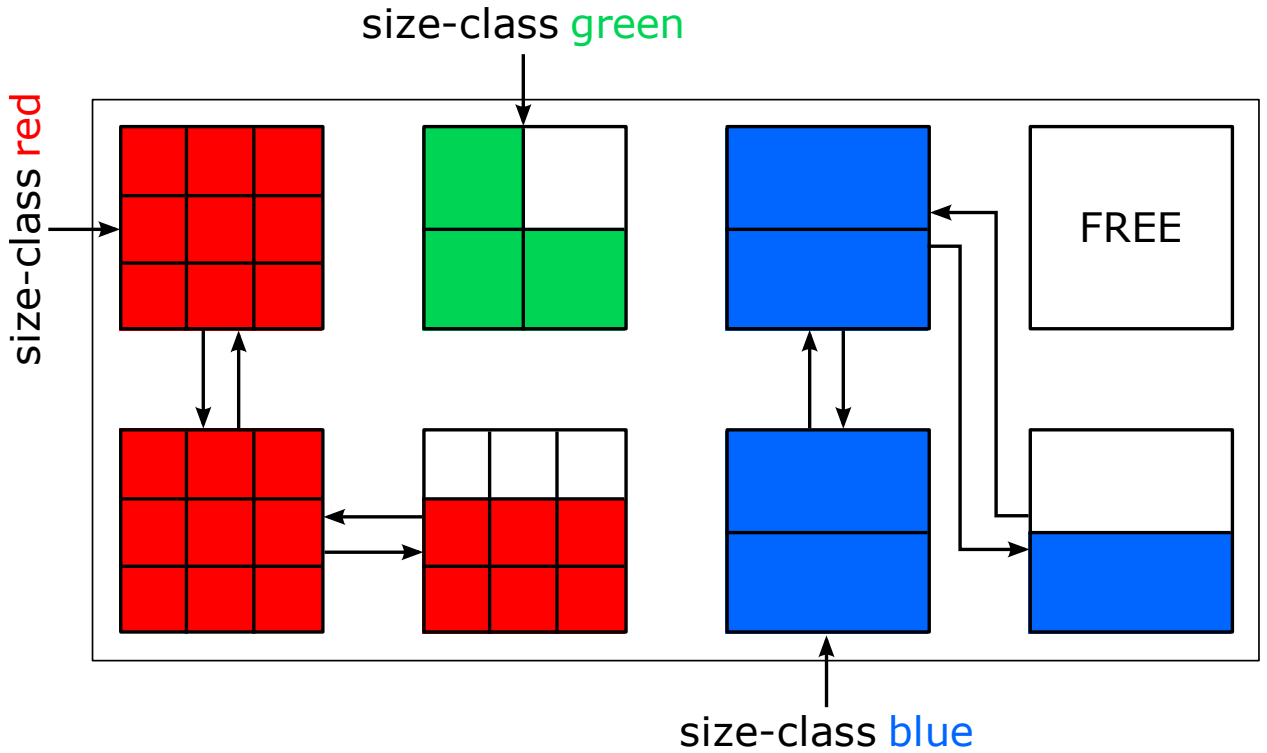
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



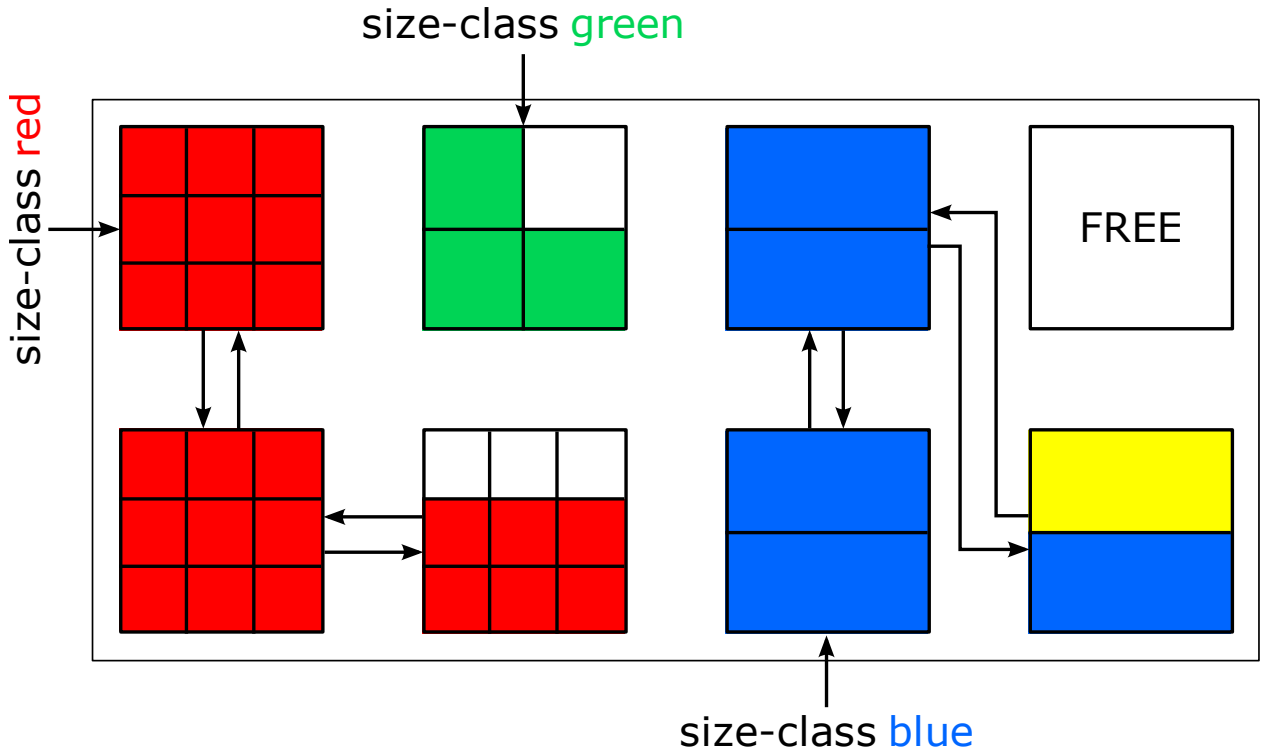
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



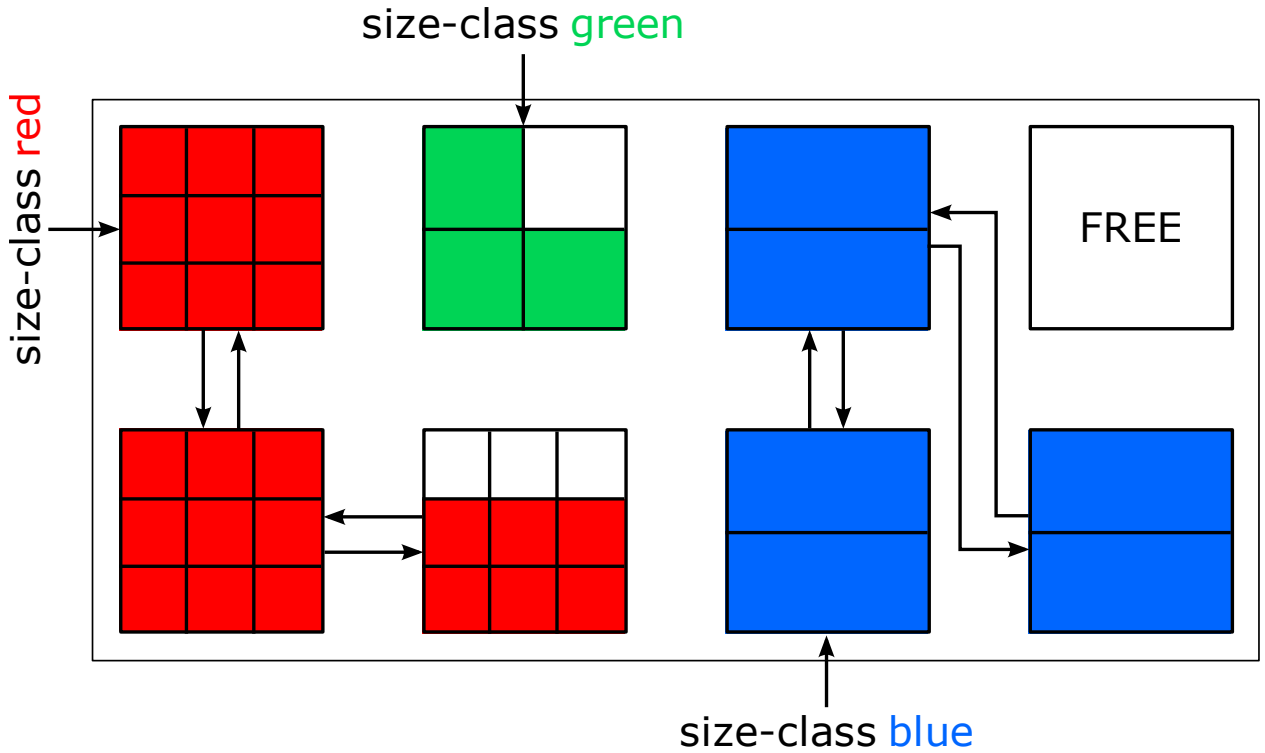
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



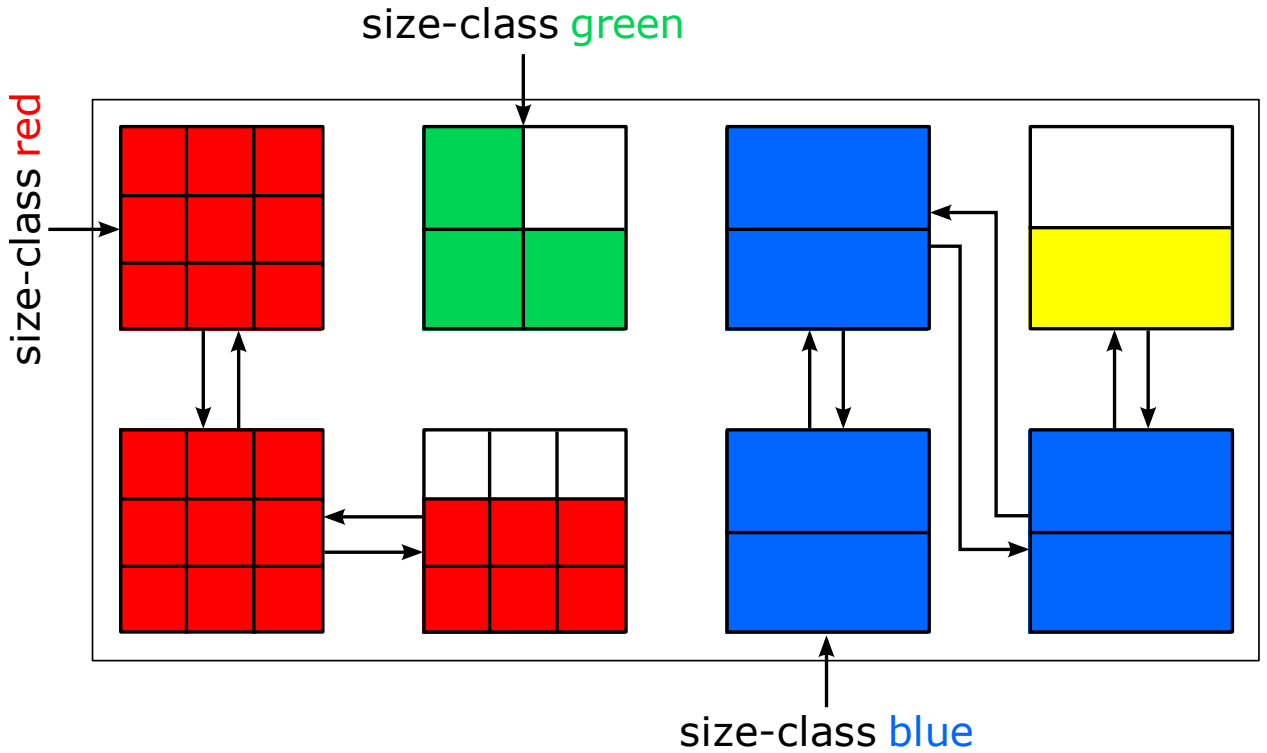
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



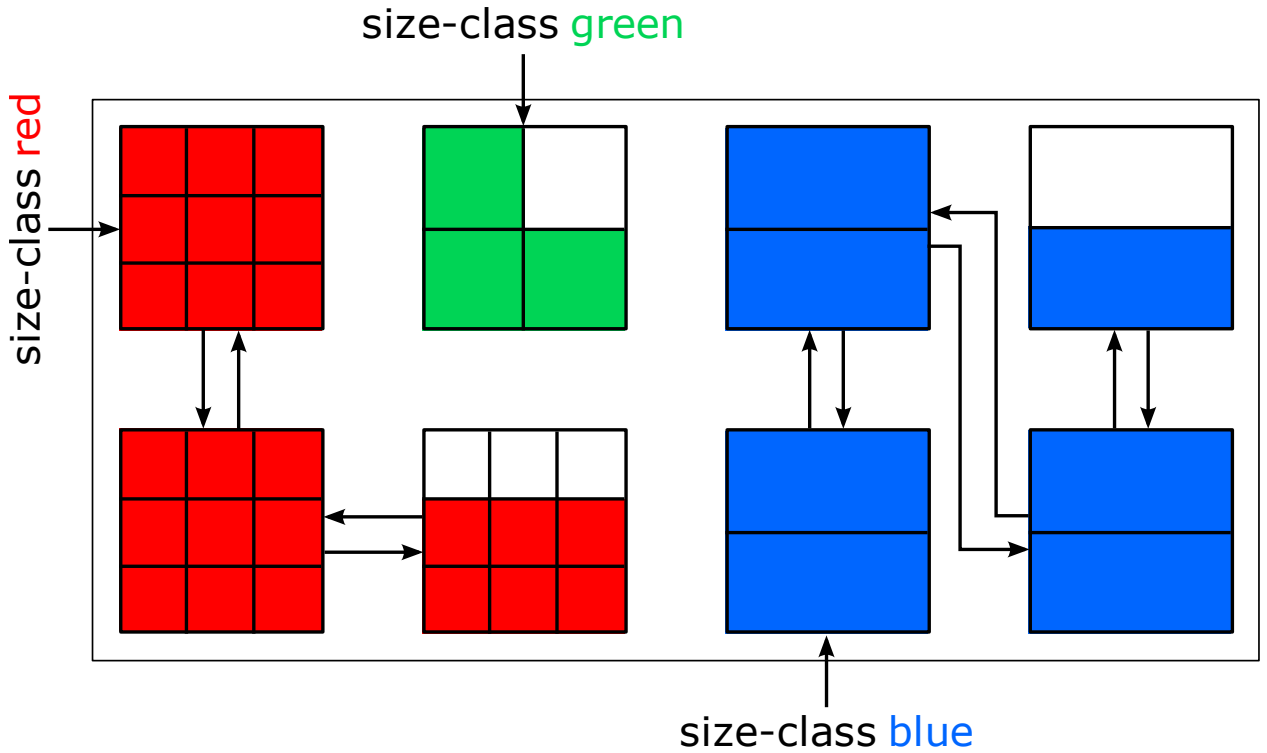
Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



Concrete Address Space

- concrete address space is divided into **pages** of equal size
- each page itself is divided into fixed-sized **page-blocks**
- n predefined page-block sizes $\Rightarrow$ n different **size-classes**



Deallocation May Involve Compaction

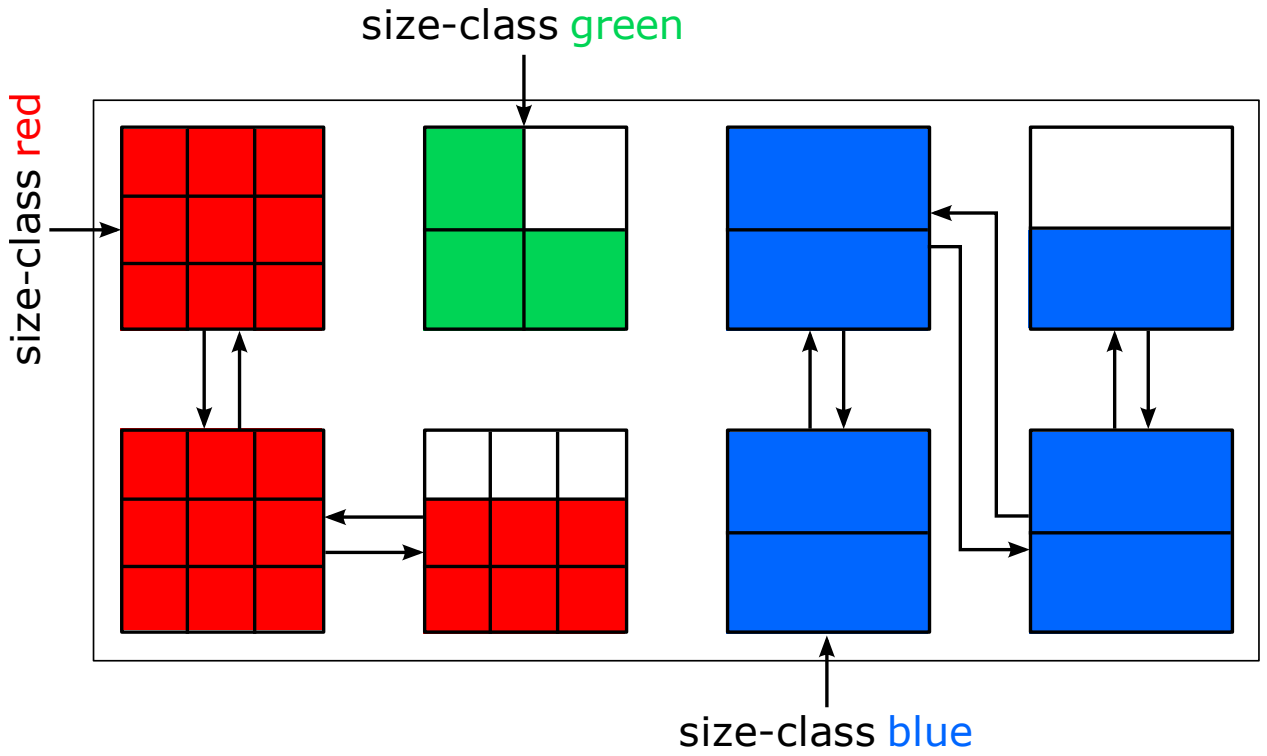
Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.

Deallocation May Involve Compaction

Size-Class Compact Invariant:

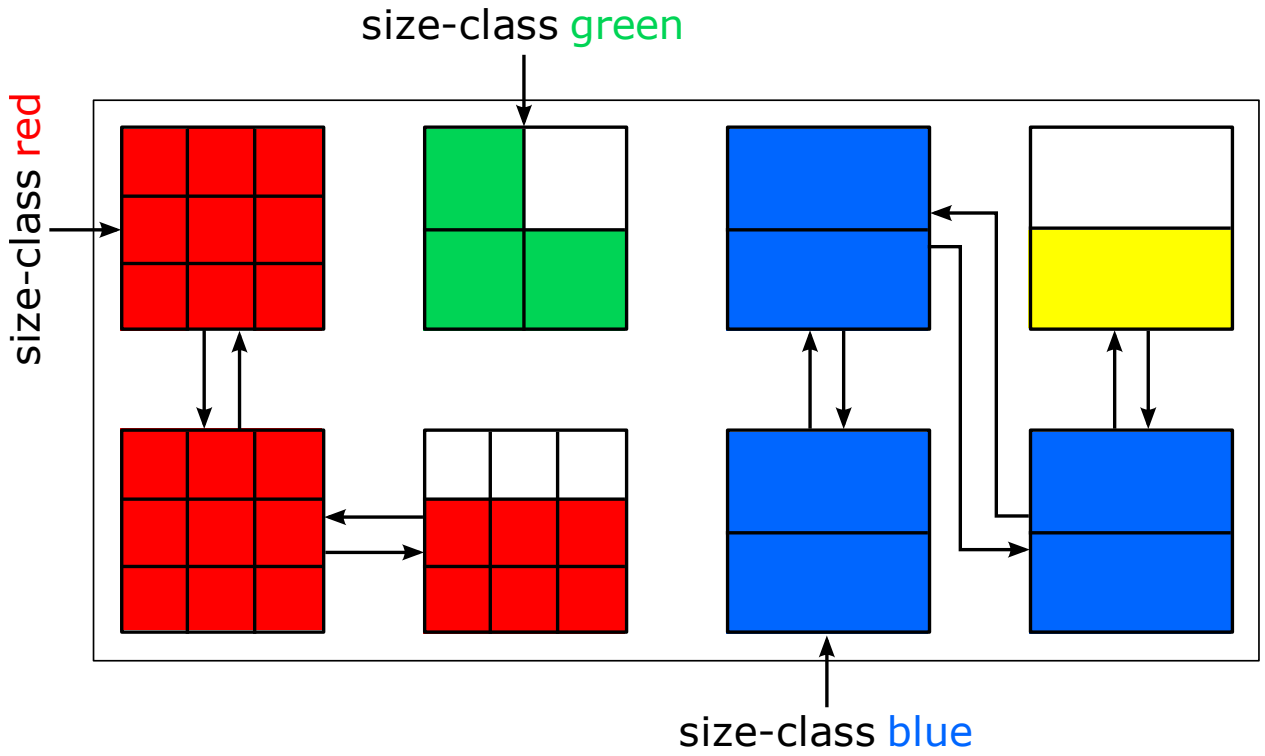
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

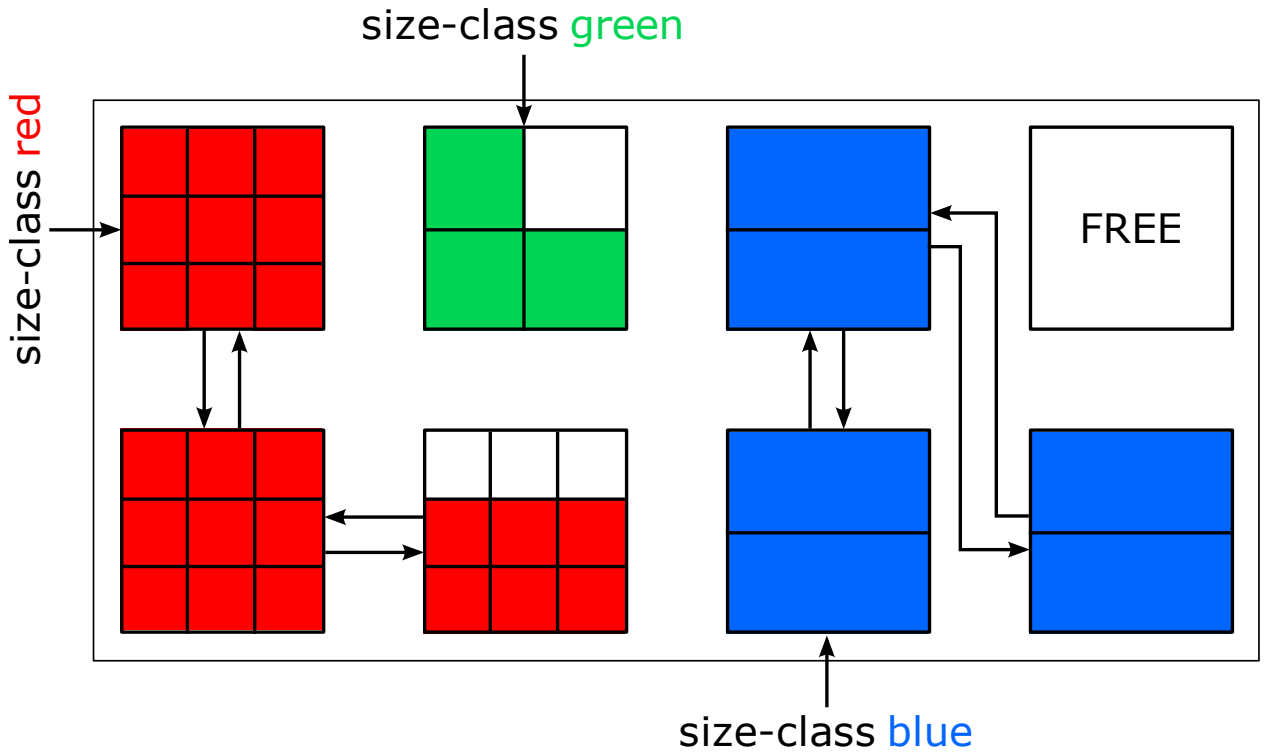
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

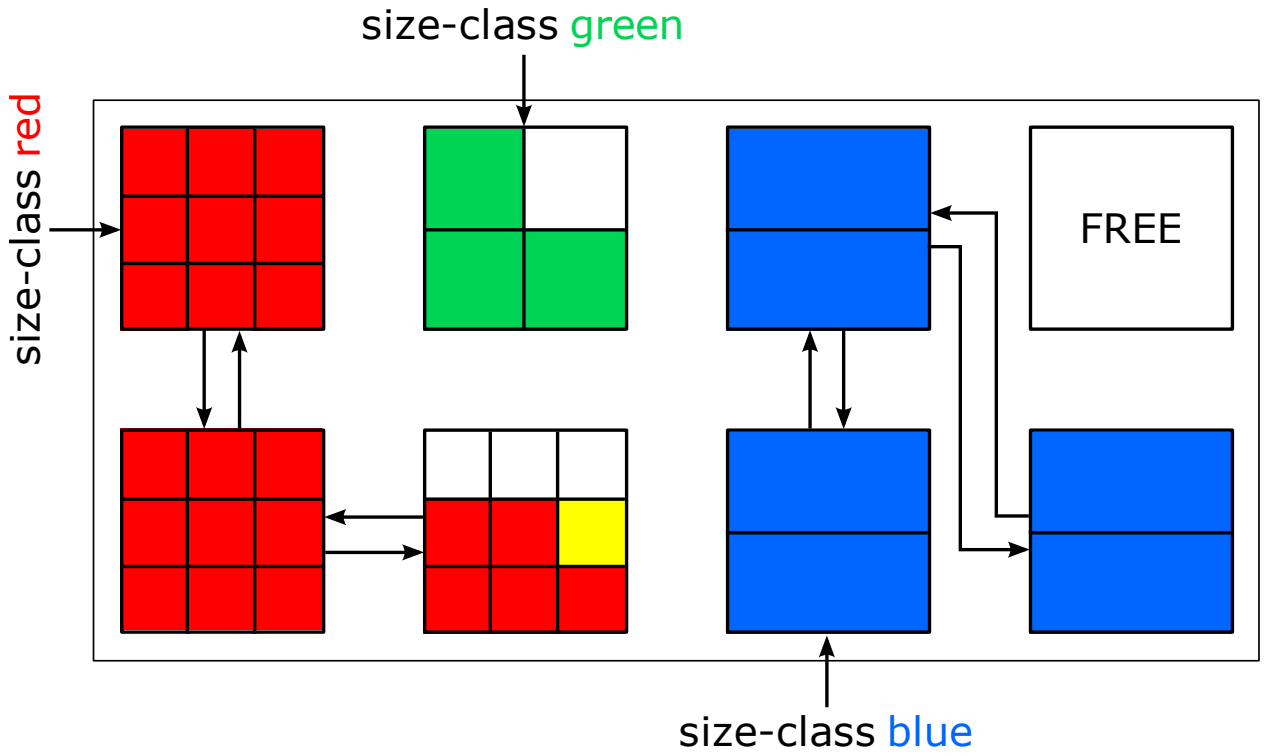
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

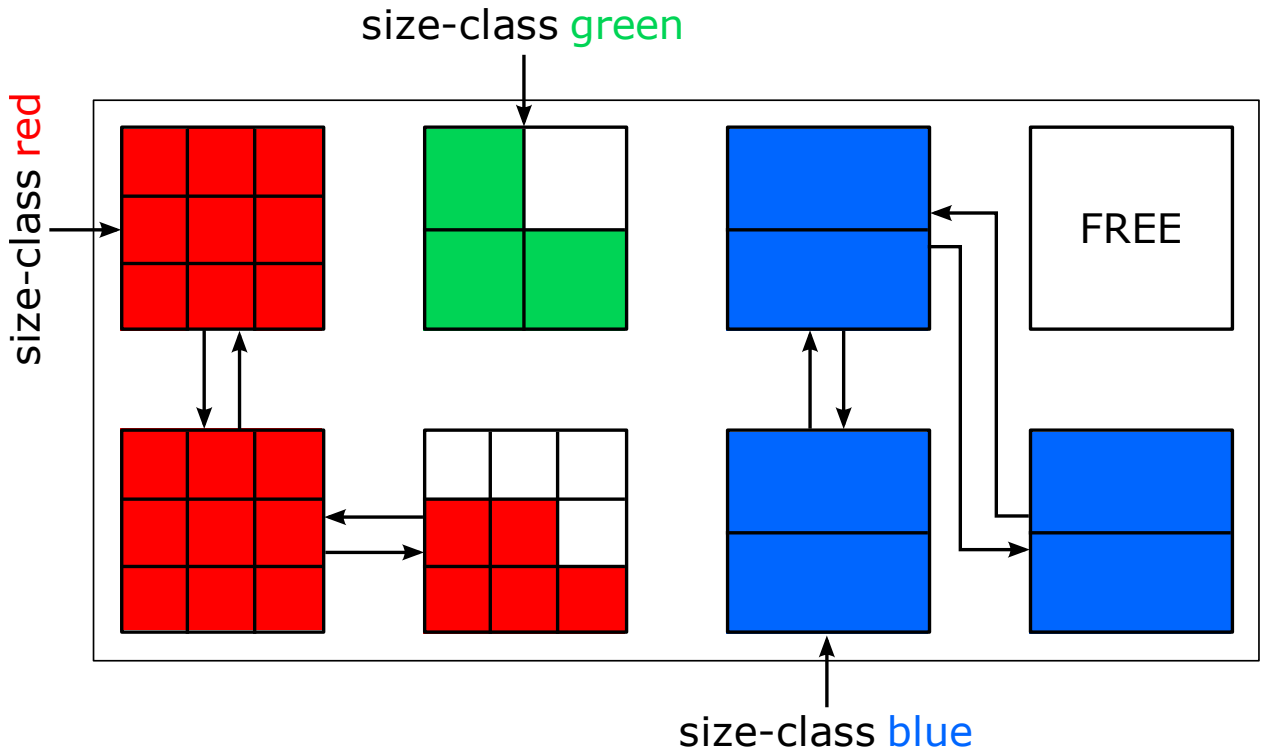
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

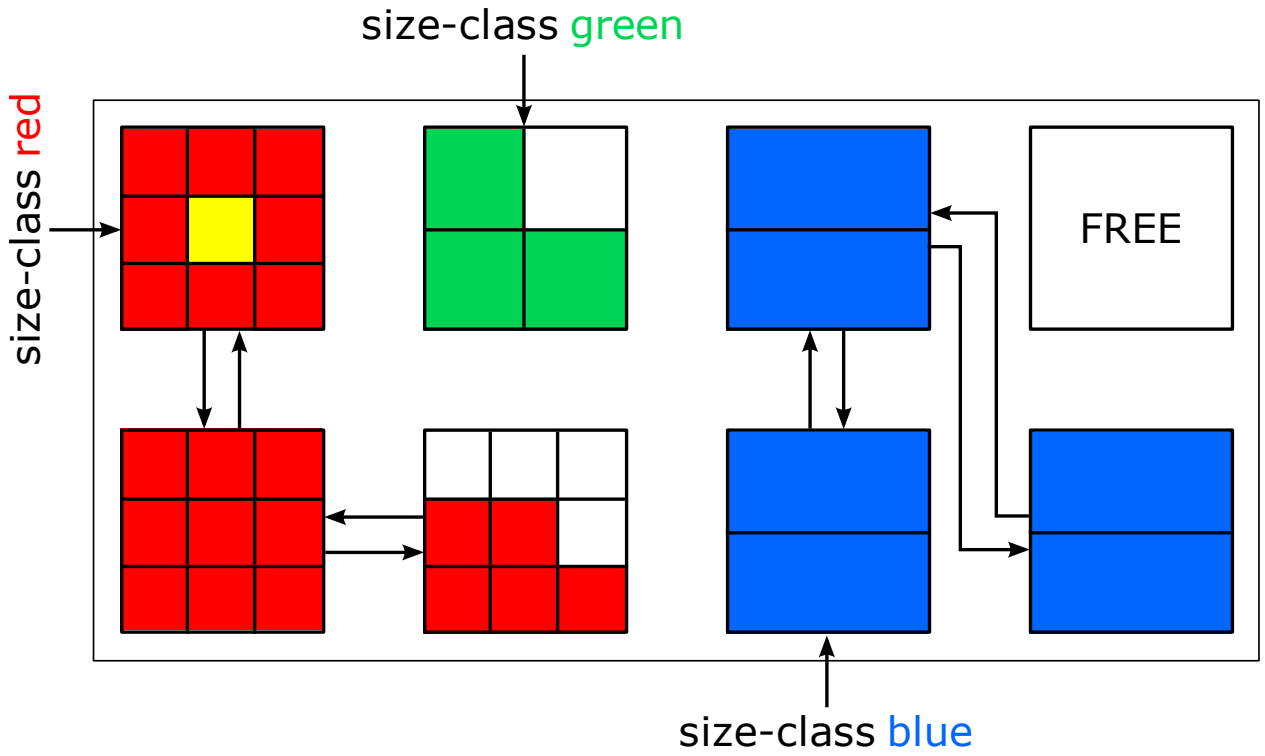
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

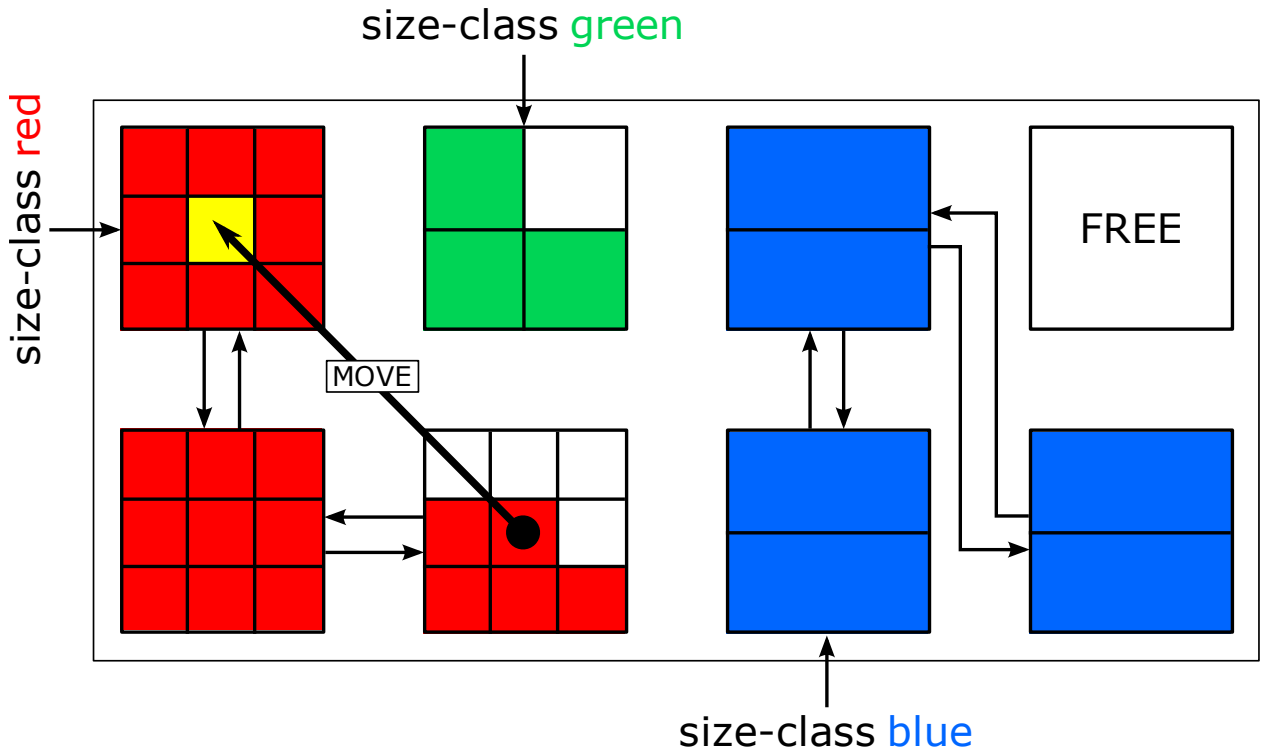
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

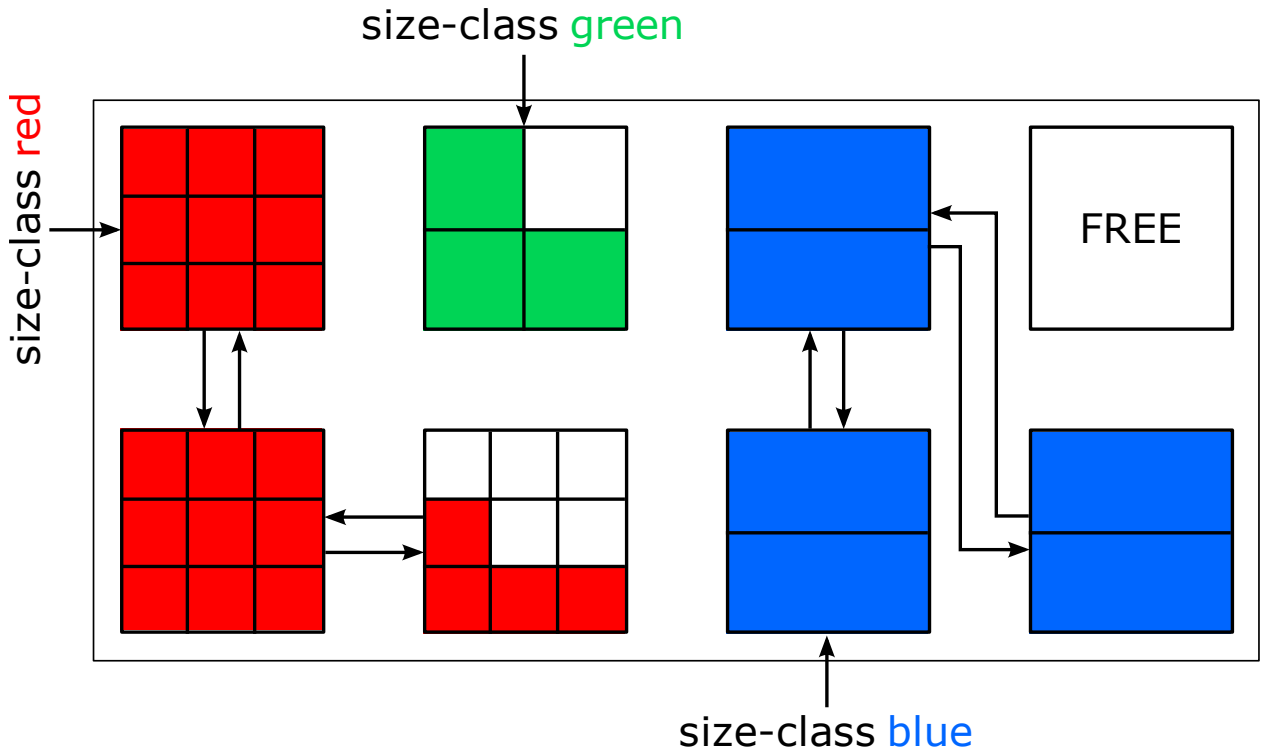
Each size-class can contain at most one not-full page.



Deallocation May Involve Compaction

Size-Class Compact Invariant:

Each size-class can contain at most one not-full page.



Compact-Fit Versions

- Compact-fit moving version (CFM)
  - concrete space = physical memory
  - allocated objects are **contiguous** in physical memory
  - compaction: leads to movements in physical memory
- Compact-fit non-moving version (CFNM)
  - concrete space = virtual memory (blocks)
  - allocated objects are **not contiguous** in physical memory, but are contiguous in virtual memory
  - compaction: reprogramming block table

Compact-Fit Moving Version Complexity

- malloc(n) takes $\Theta(1)$ time
- free(n) takes $\mathcal{O}(n)$ time
  because of compaction
- memory access (dereference) takes $\Theta(1)$ time
  because of abstract address space
- memory fragmentation is bounded and predictable

Compact-Fit Non-Moving Version Complexity

- malloc(n) takes $\Theta(n)$ time
  because of maintaining the virtual memory
- free(n) takes $\Theta(n)$ time
  because of maintaining the virtual memory and compaction
- memory access (dereference) takes $\Theta(1)$ time
  because of abstract address space and virtual memory
- memory fragmentation is bounded and predictable

Partial Compaction

Idea:

Allow an arbitrary number k of not-full pages within a size-class.

Result:

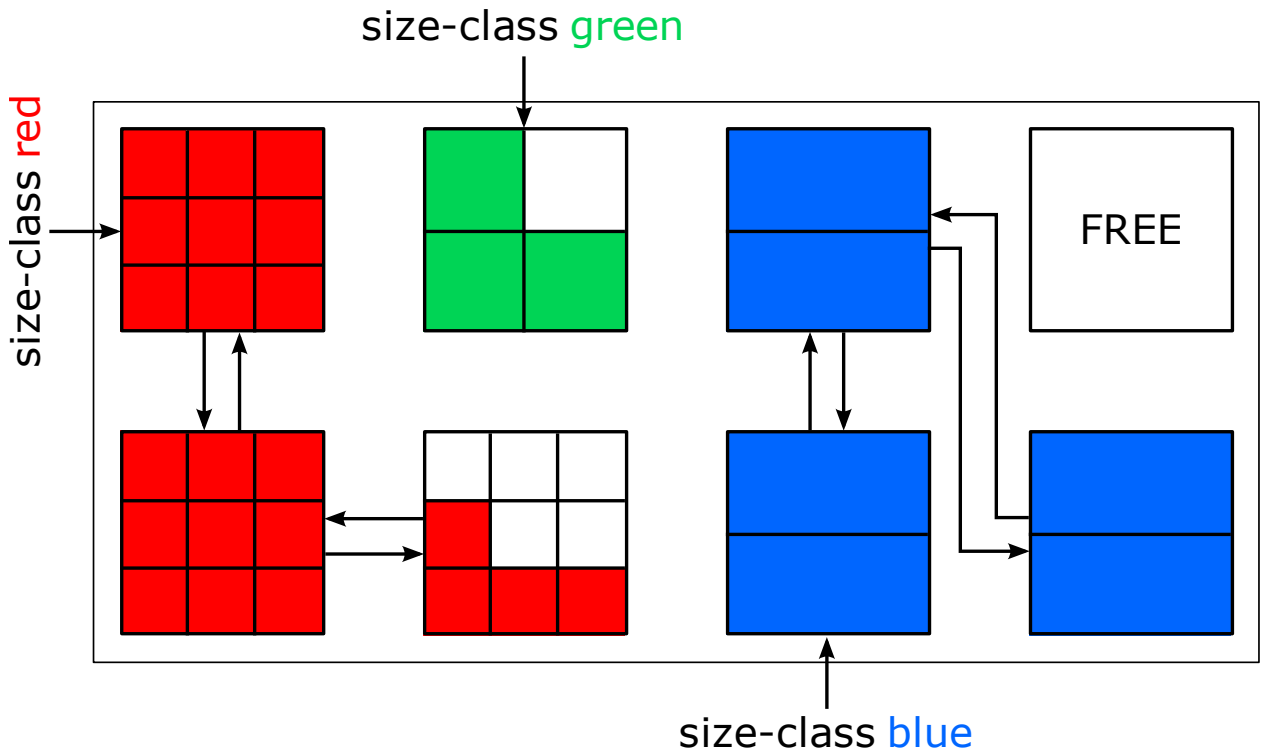
Each deallocation that happens when $number\_not\_full\_pages \leq k$ takes constant time, but fragmentation increases with k.

Effect:

This way we formalize, control, and implement the trade-off between temporal performance and memory fragmentation.

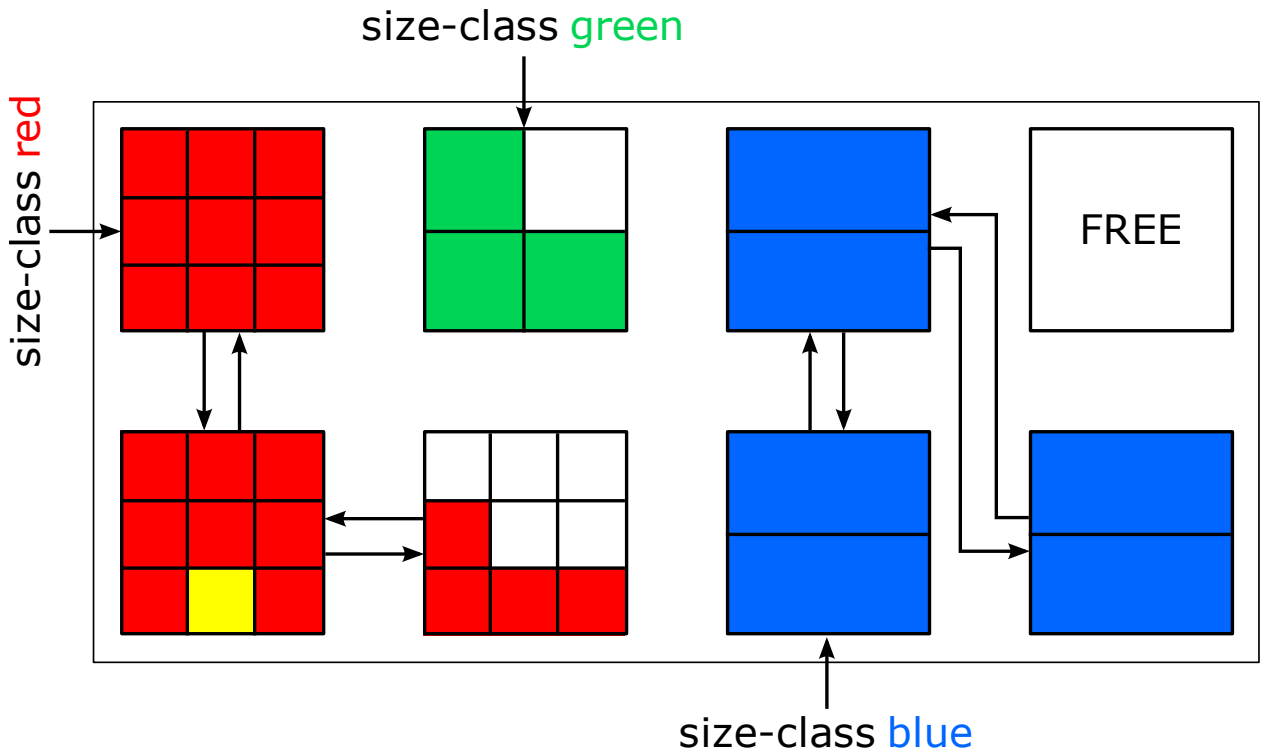
Partial Compaction

size-class red: k=2



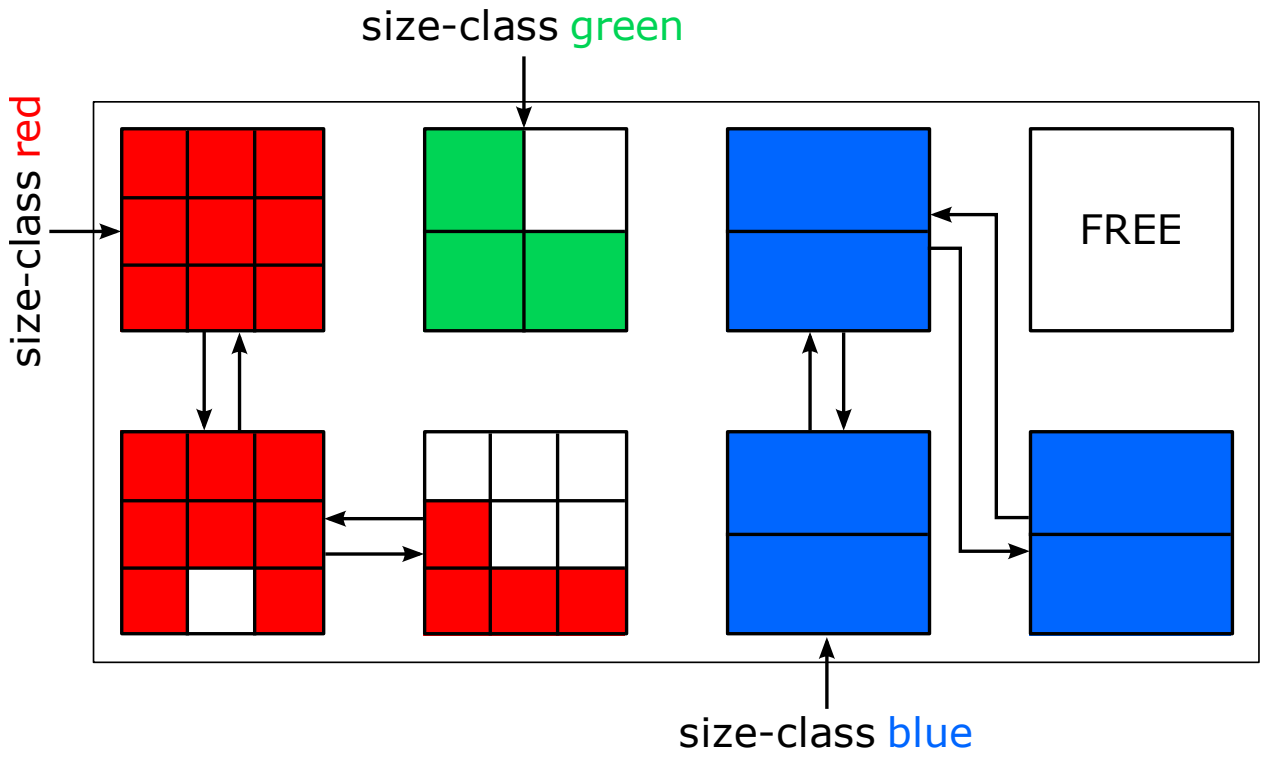
Partial Compaction

size-class red: k=2



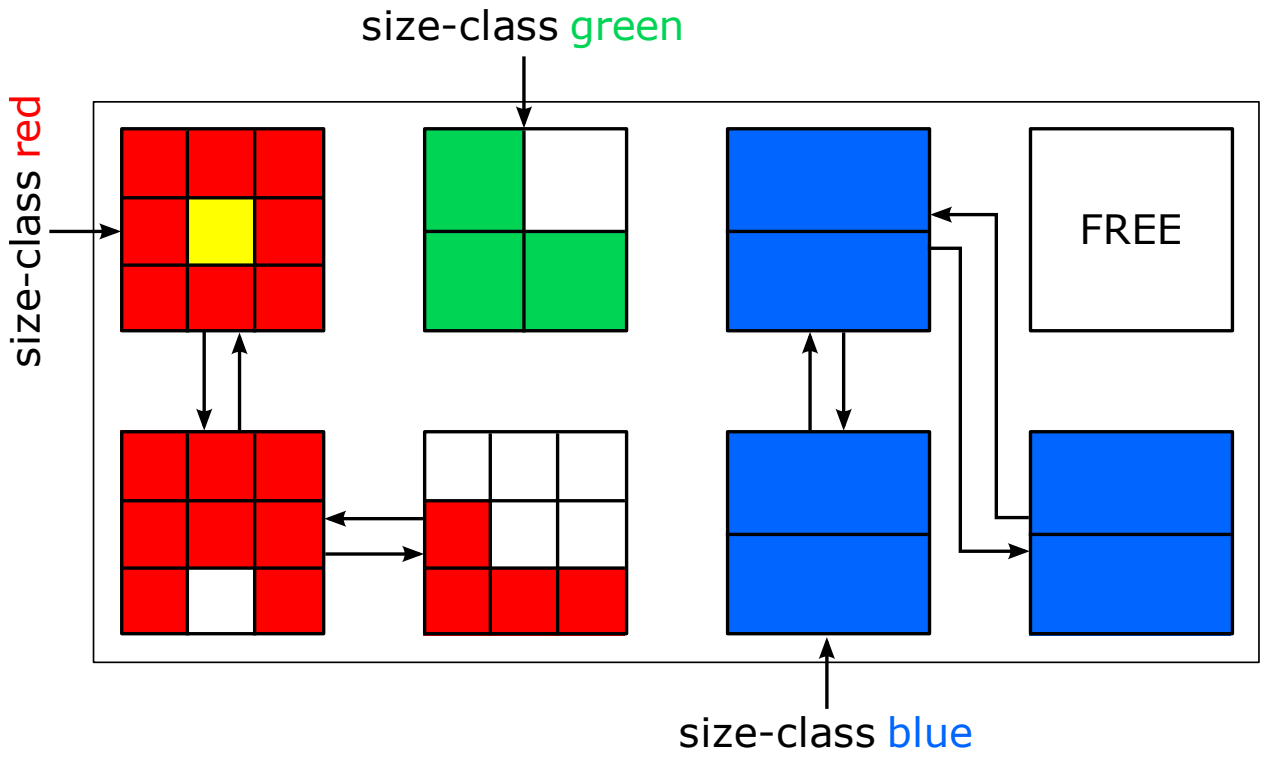
Partial Compaction

size-class red: k=2



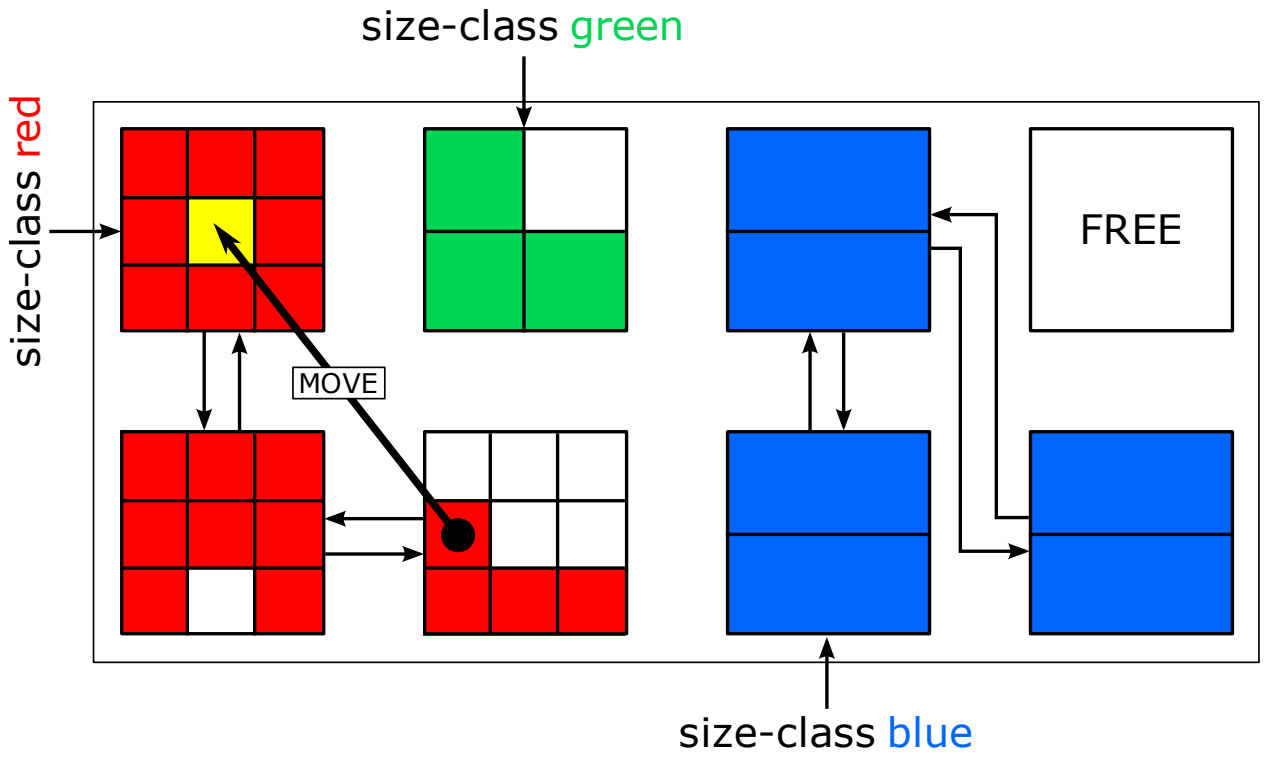
Partial Compaction

size-class red: k=2



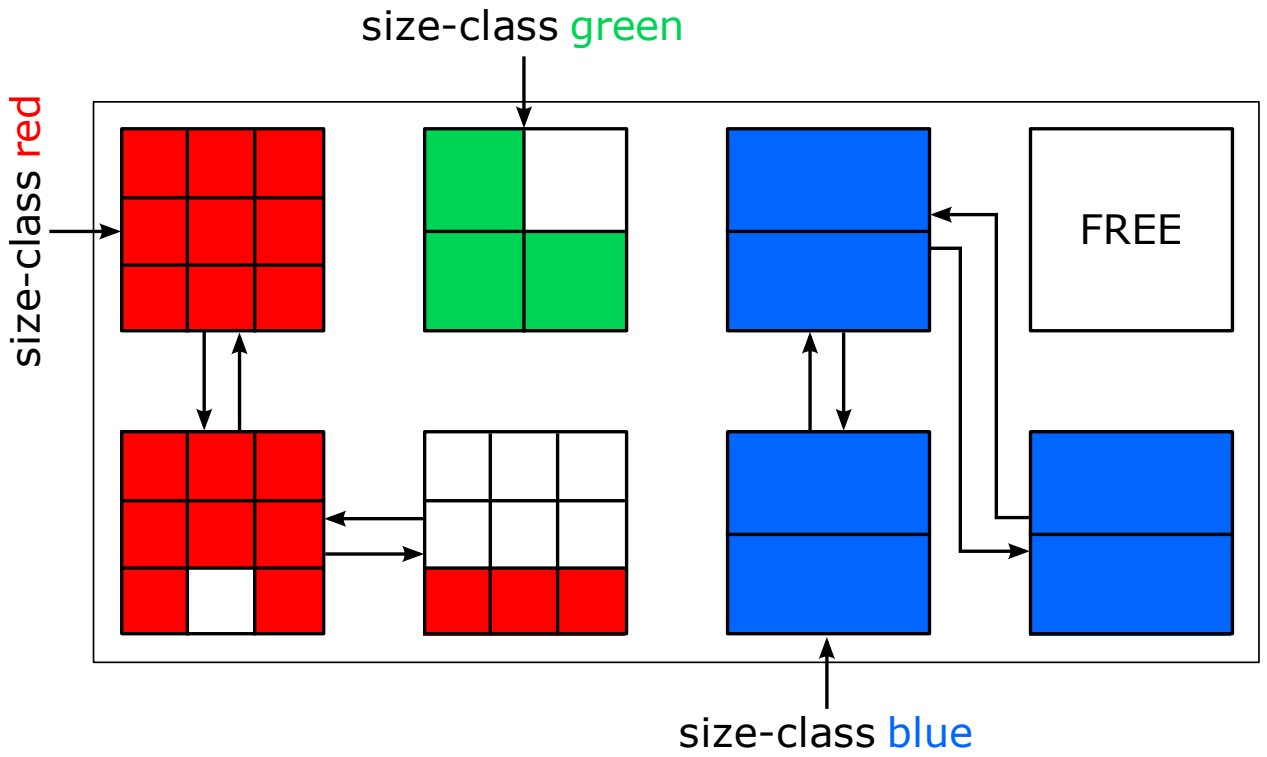
Partial Compaction

size-class red: k=2



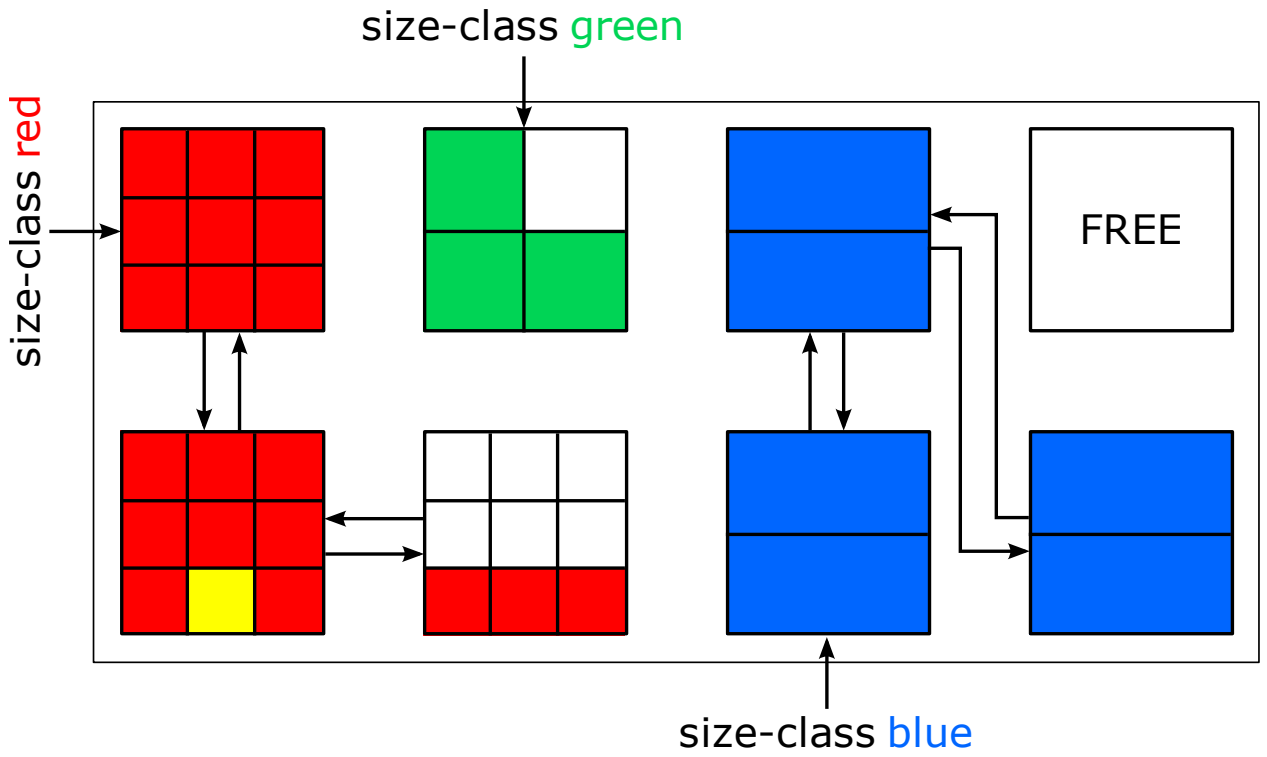
Partial Compaction

size-class red: k=2



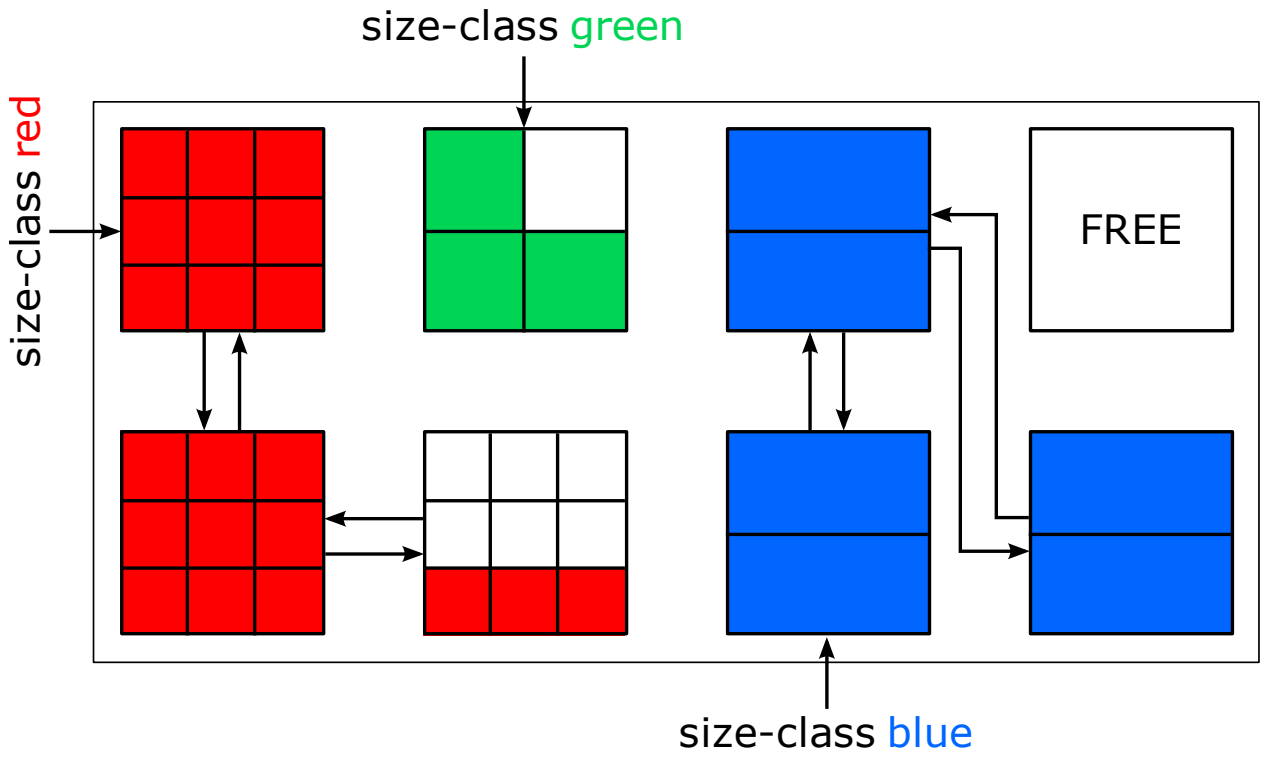
Partial Compaction

size-class red: k=2

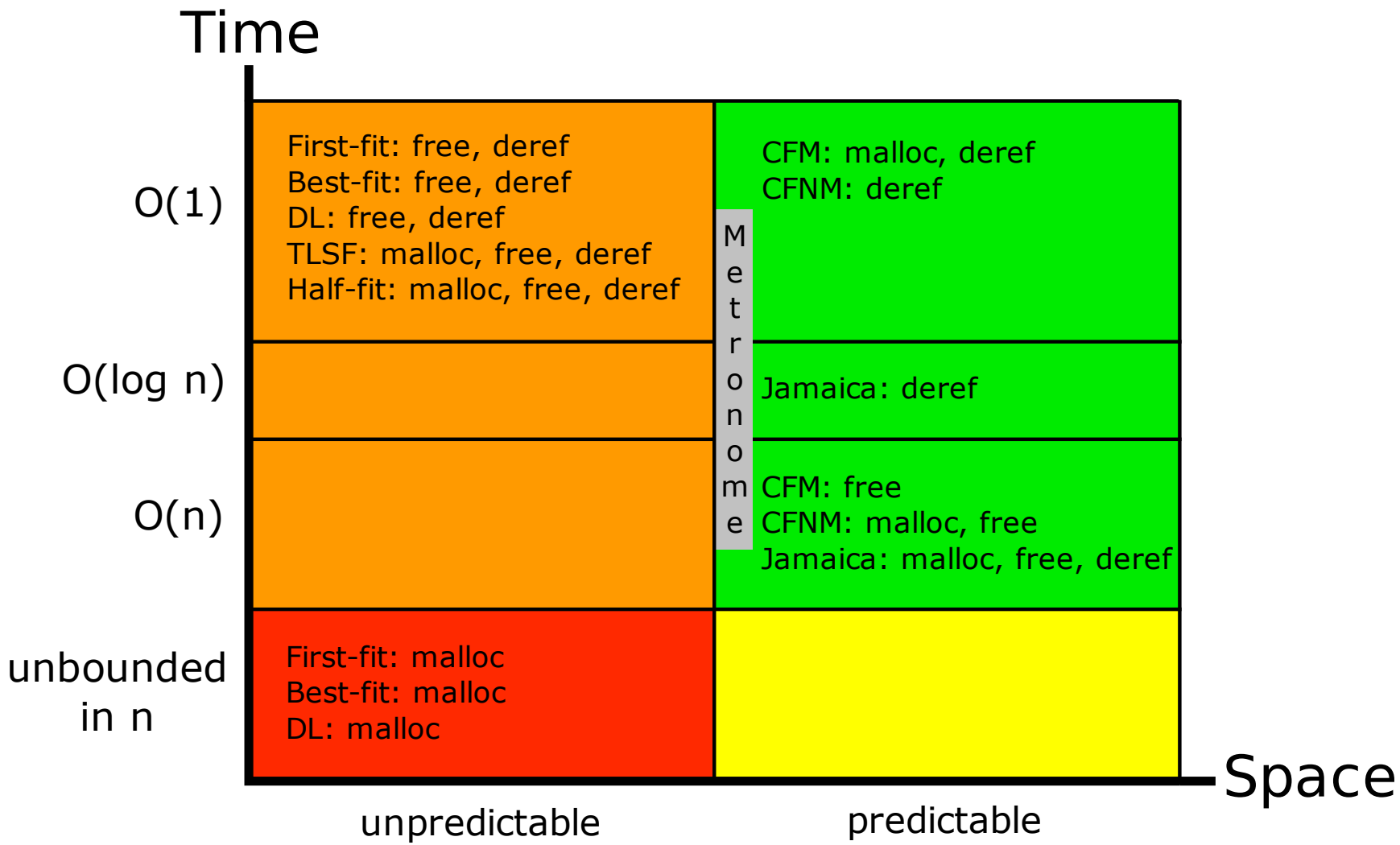


Partial Compaction

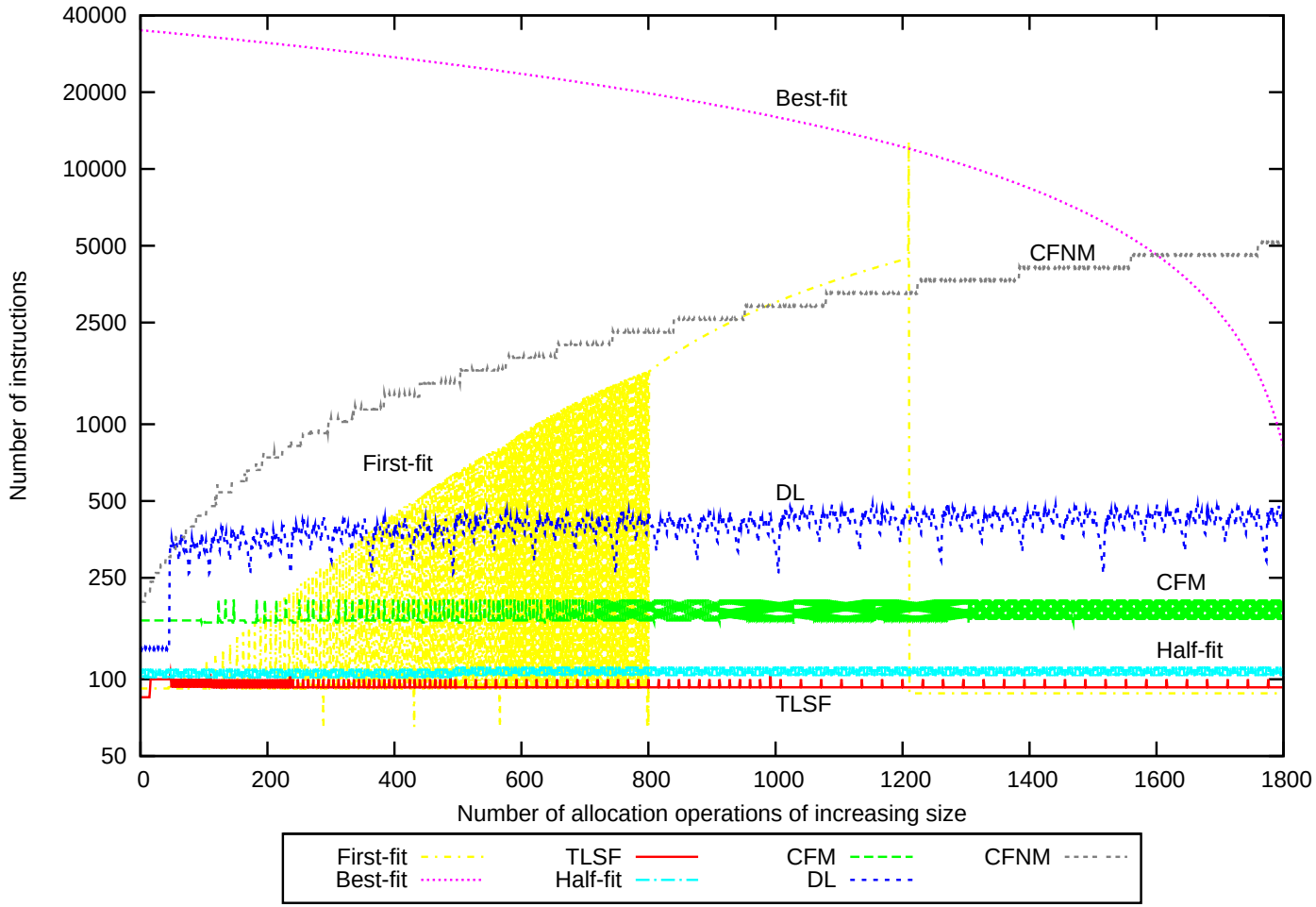
size-class red: k=2



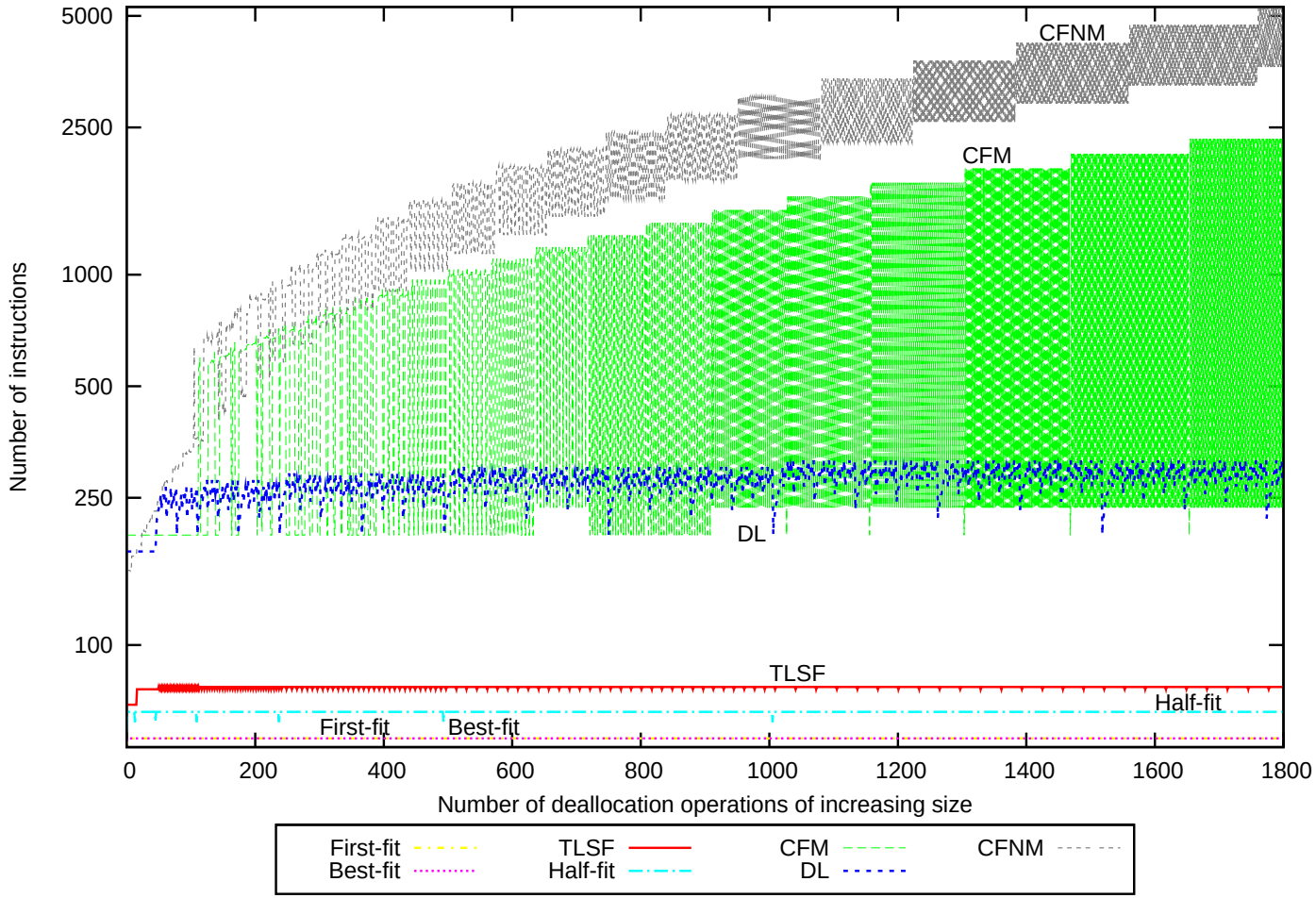
Related Work



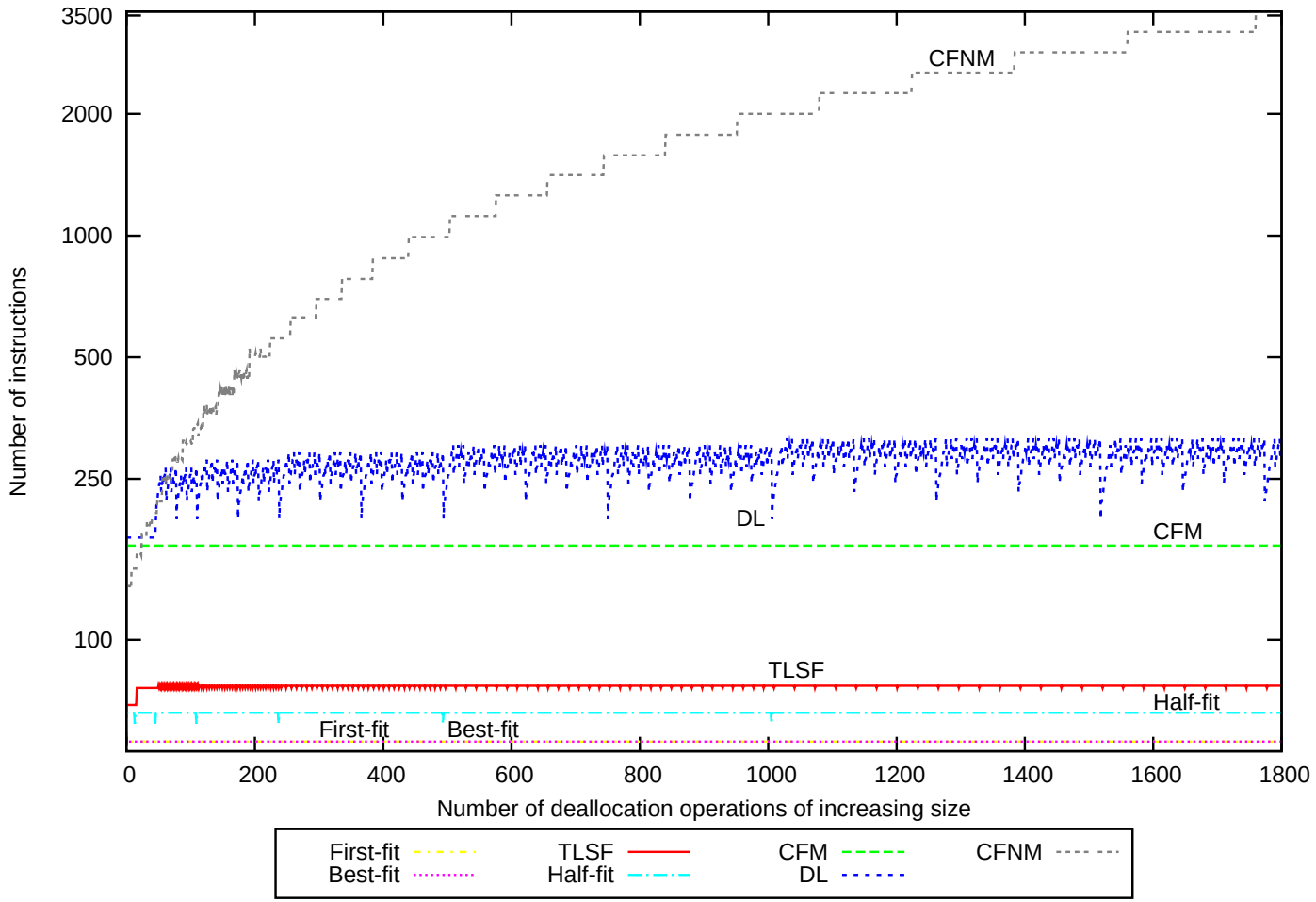
Incremental Allocation Benchmark



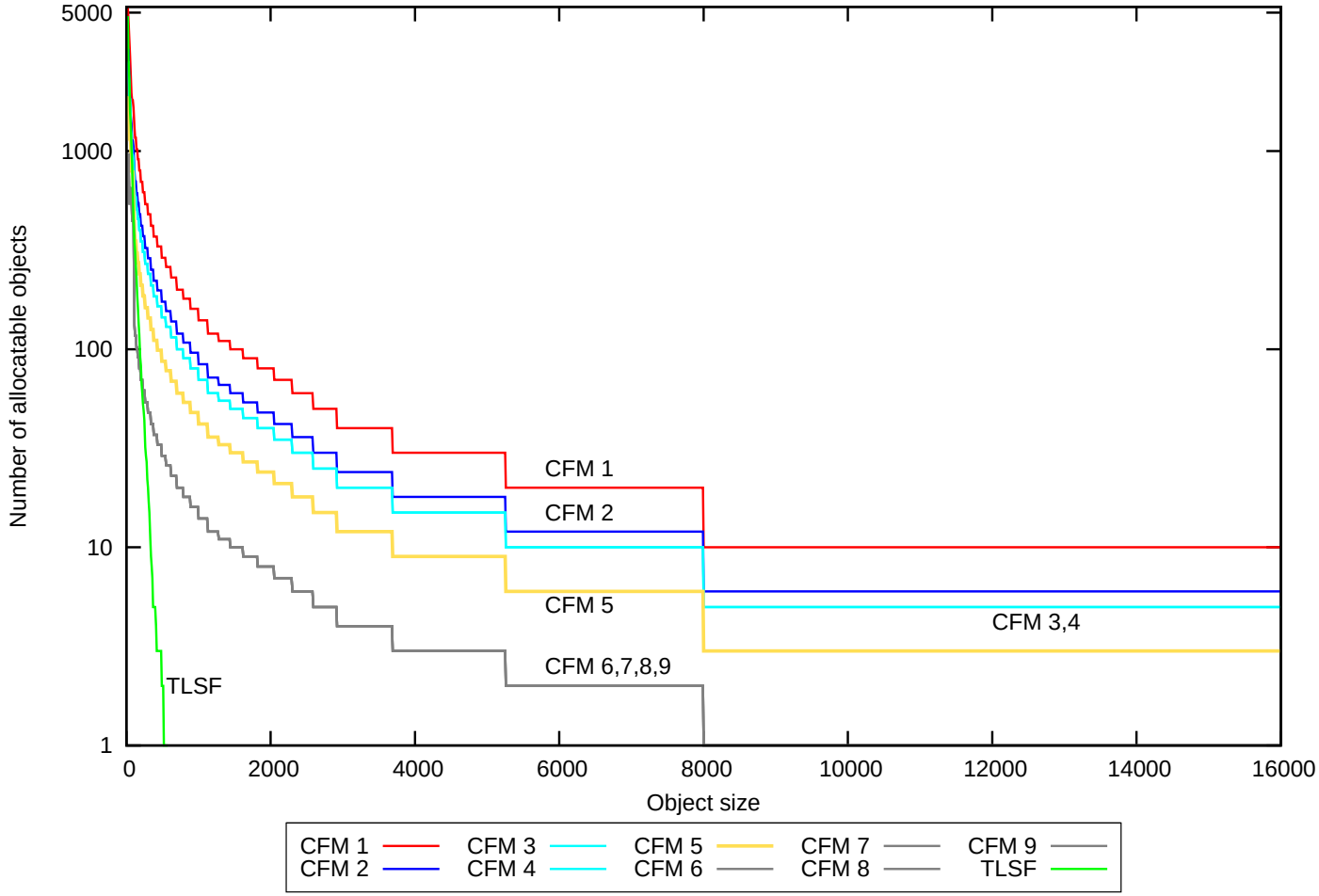
Incremental Free Benchmark



Incremental Free Partial Compaction



Fragmentation



Conclusion and Future Work

Contribution:

- Compact-fit is predictable in time and space
- moving and non-moving Compact-fit implementations

Future work:

- virtual machine implementation
- source-to-source translator
- concurrency and multi-processor support
- static program analysis can help to optimize the k for the partial compaction strategy

`http://tiptoe.cs.uni-salzburg.at/compact-fit`