Explicit, Dynamic Memory Management with Temporal and Spatial Guarantees

Christoph Kirsch and Ana Sokolova
Universität Salzburg

Artist Summer School 2009
Tsinghua University, Beijing, China

Memory Management

- Allocation:
  - `malloc`

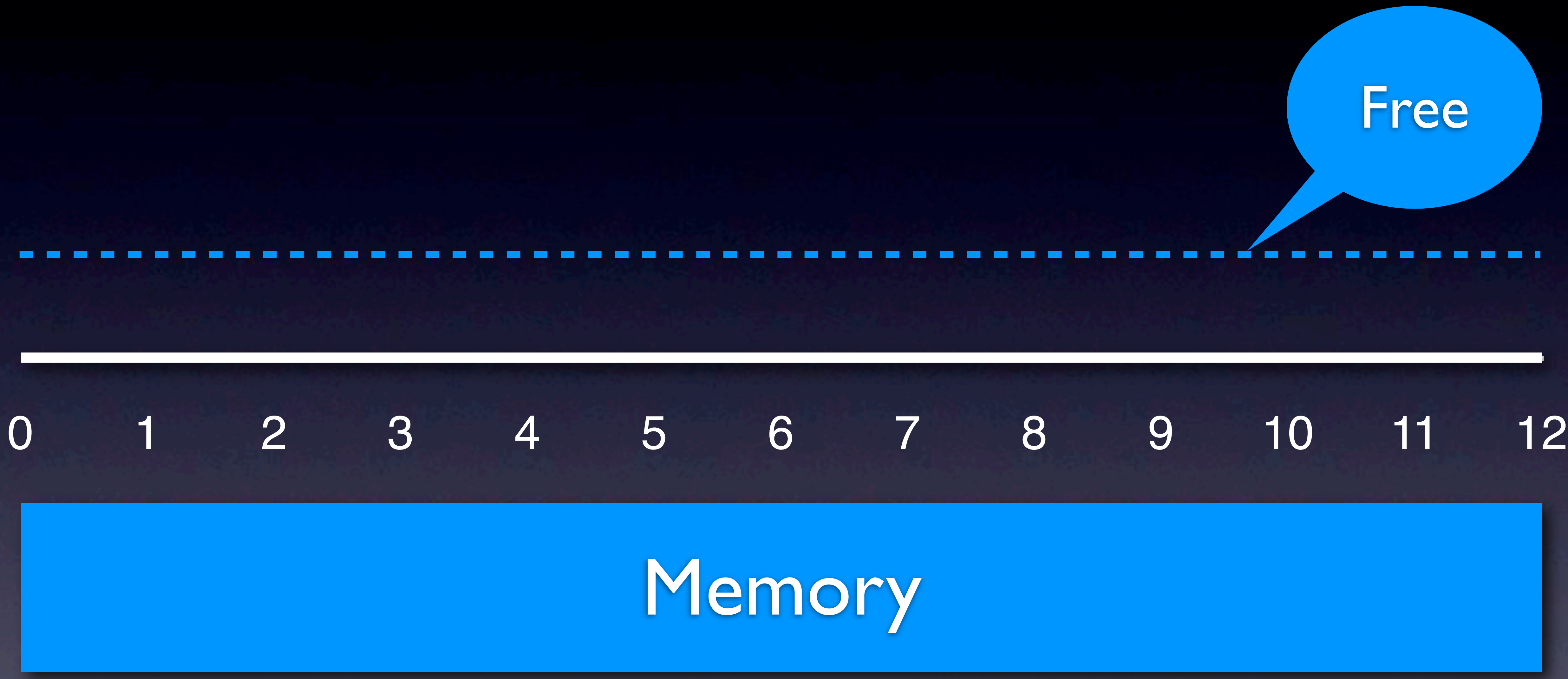
Memory Management

- Allocation:
  - `malloc`
- Deallocation:
  - `free`

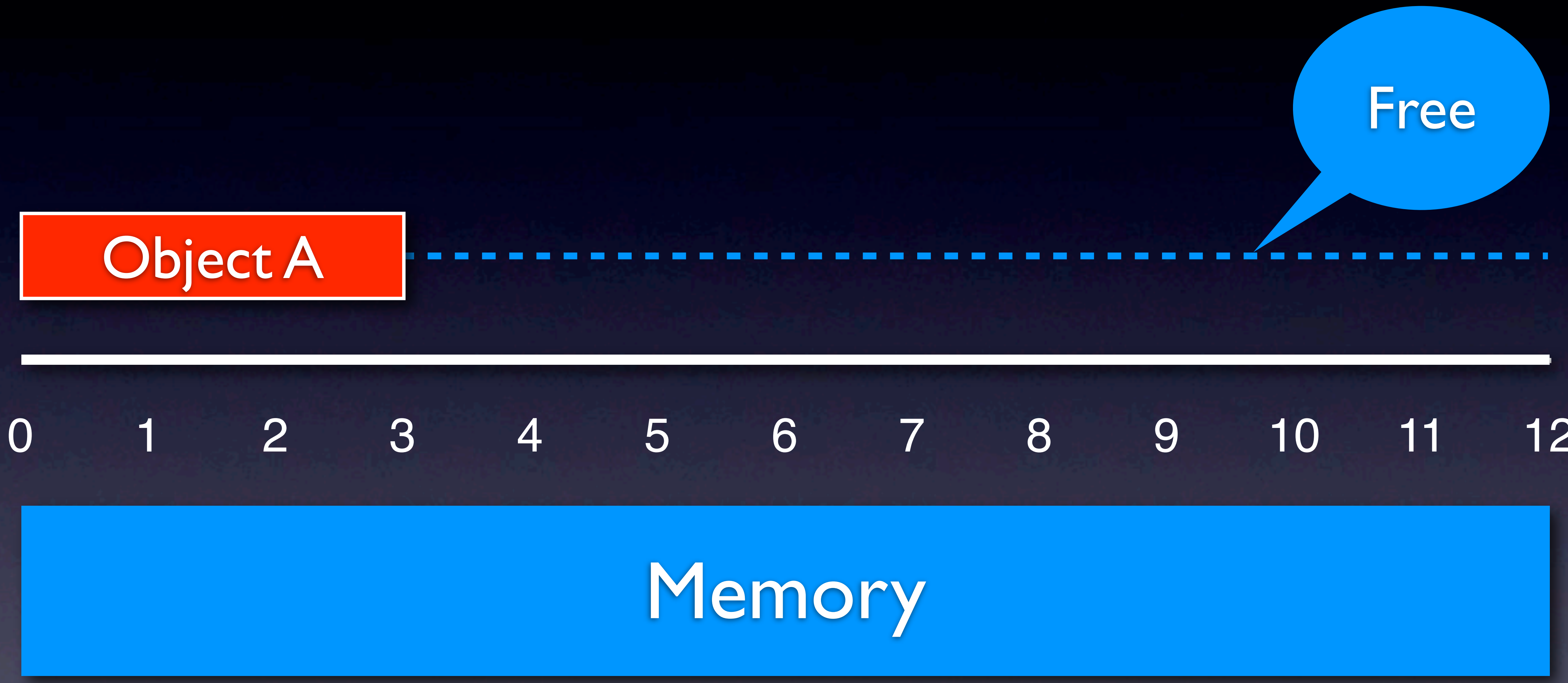
Memory Management

- Allocation:
  - `malloc`
- Deallocation:
  - `free`
- Access:
  - `read` and `write`

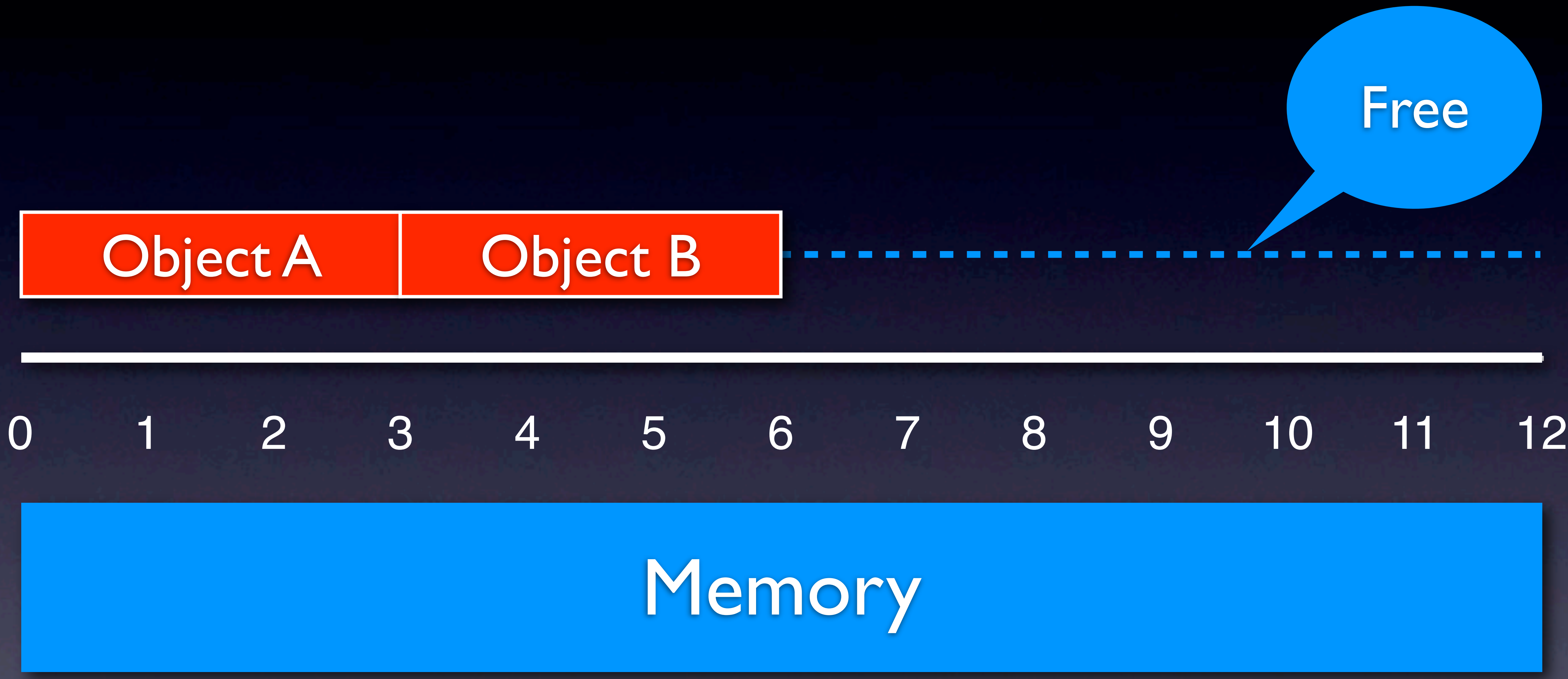
Allocation



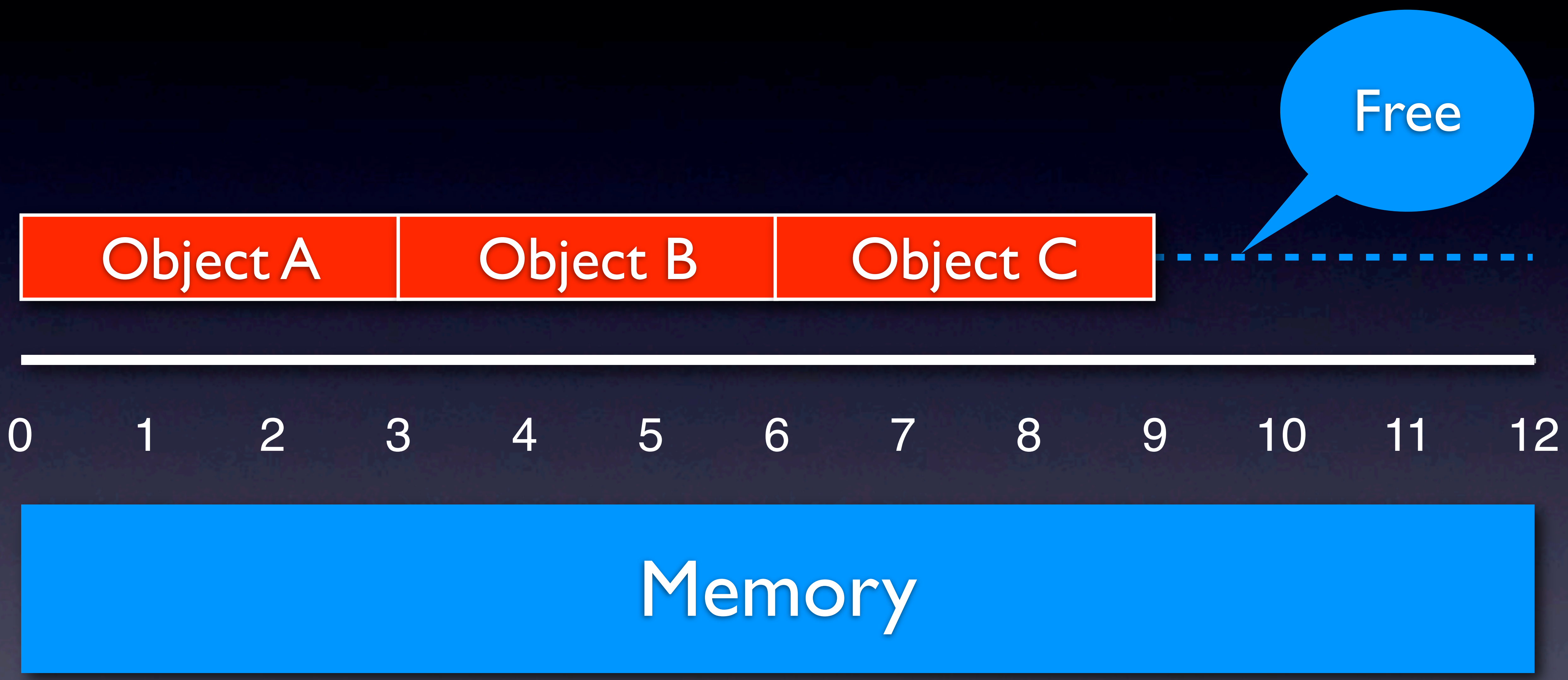
Allocation



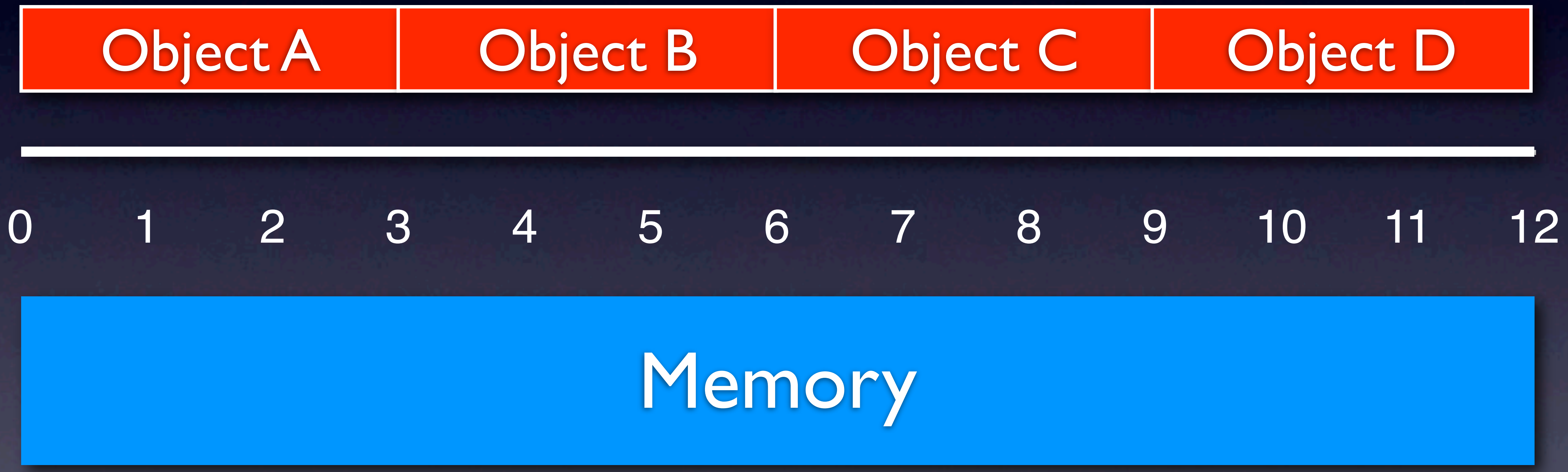
Allocation



Allocation



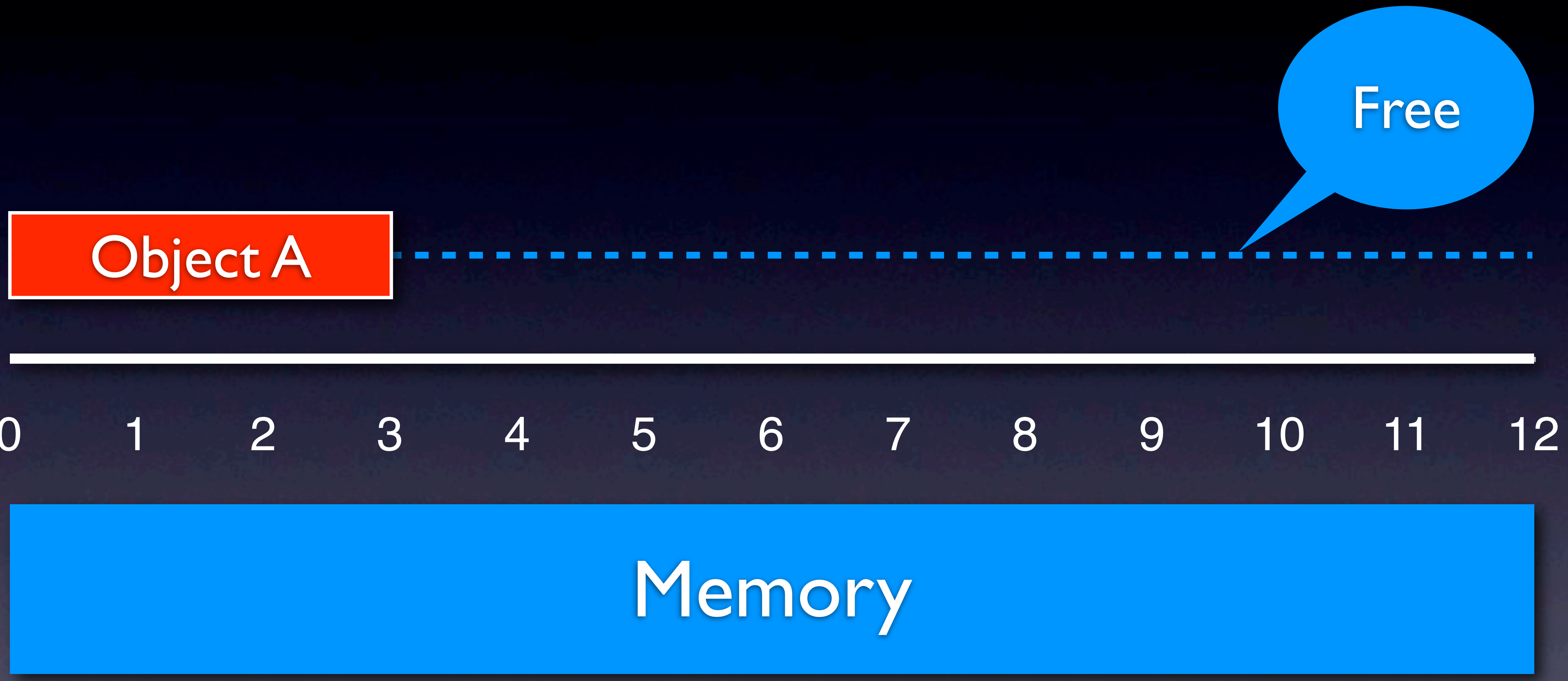
Allocation


Object A
Object B
Object C
Object D
0
1
2
3
4
5
6
7
8
9
10
11
12
Memory

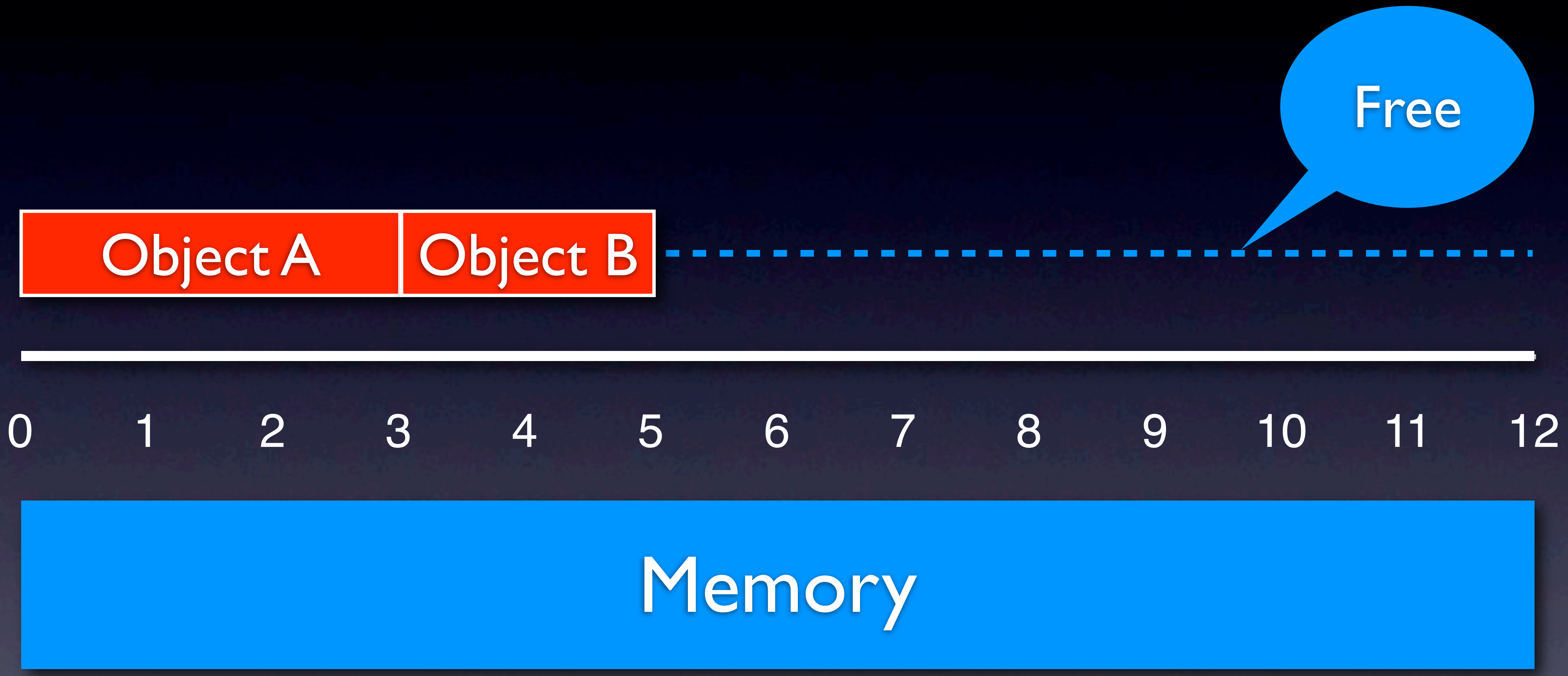
1. Assumption:

Objects may have different sizes

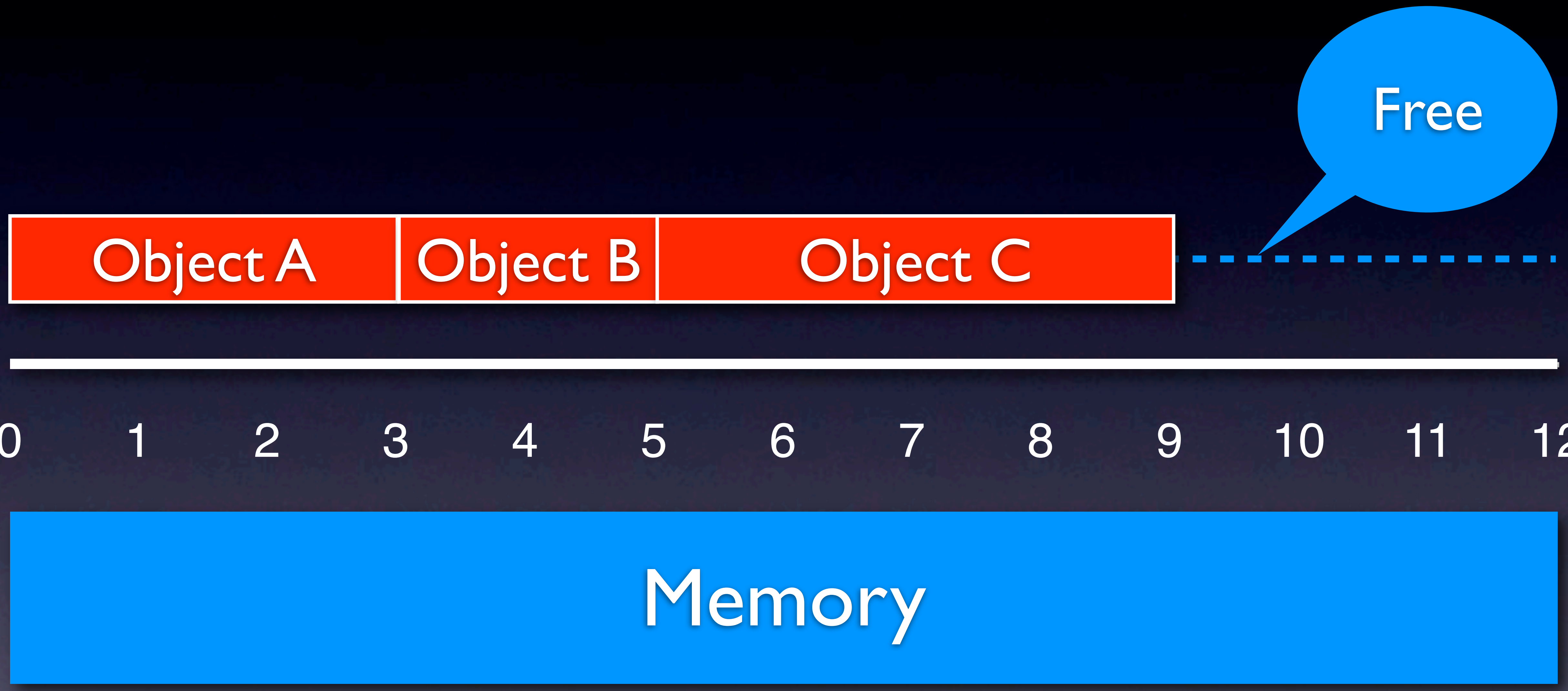
Allocation



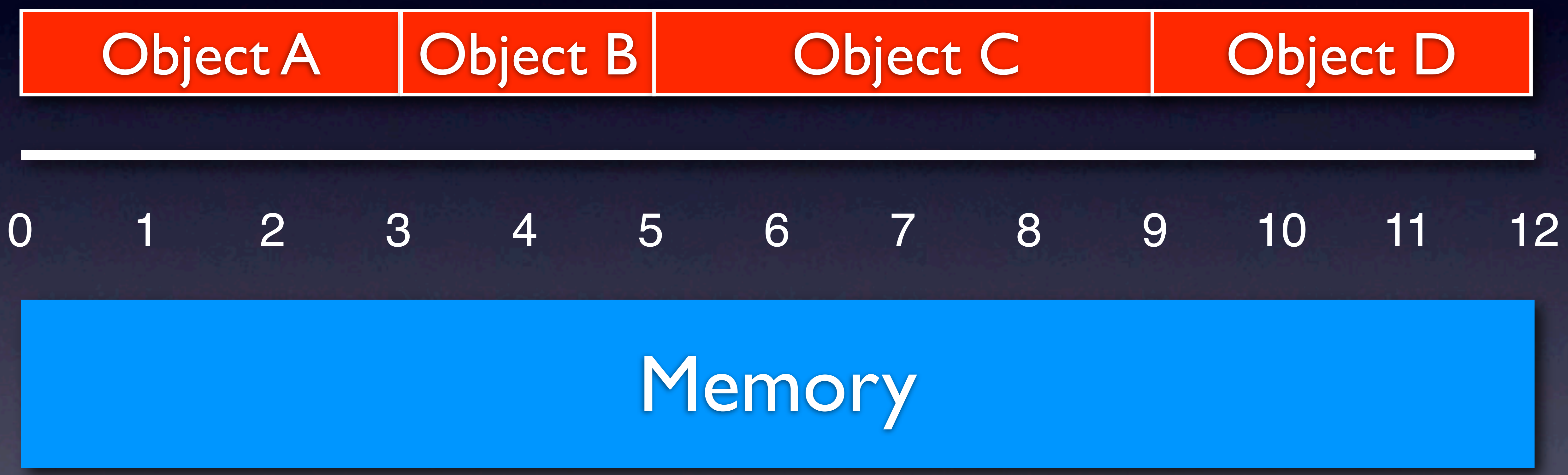
Allocation



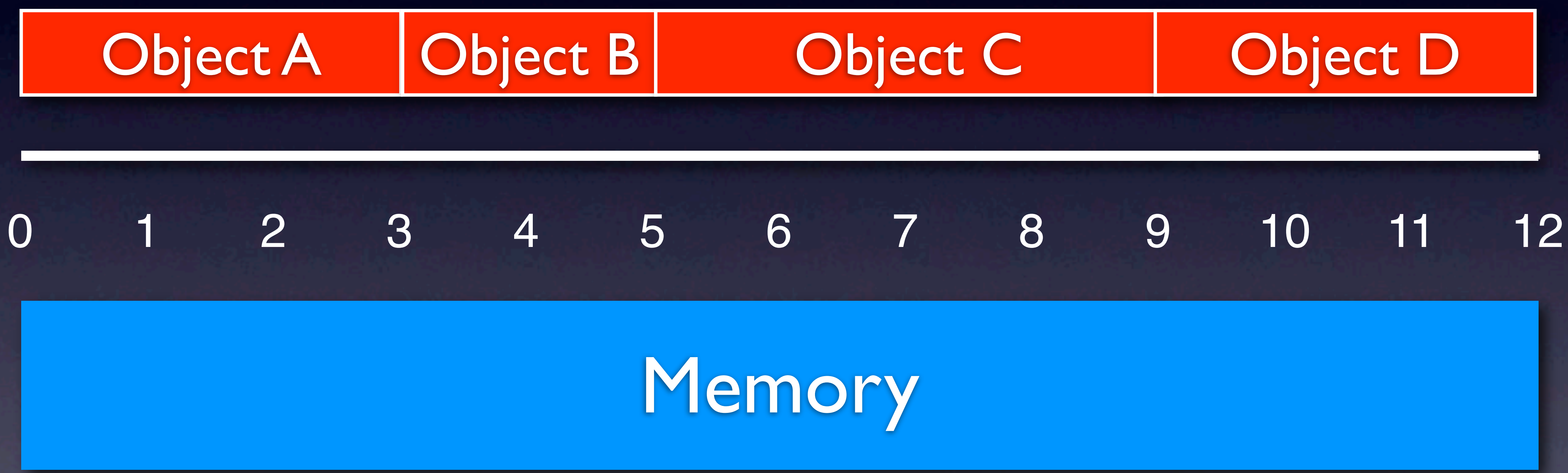
Allocation



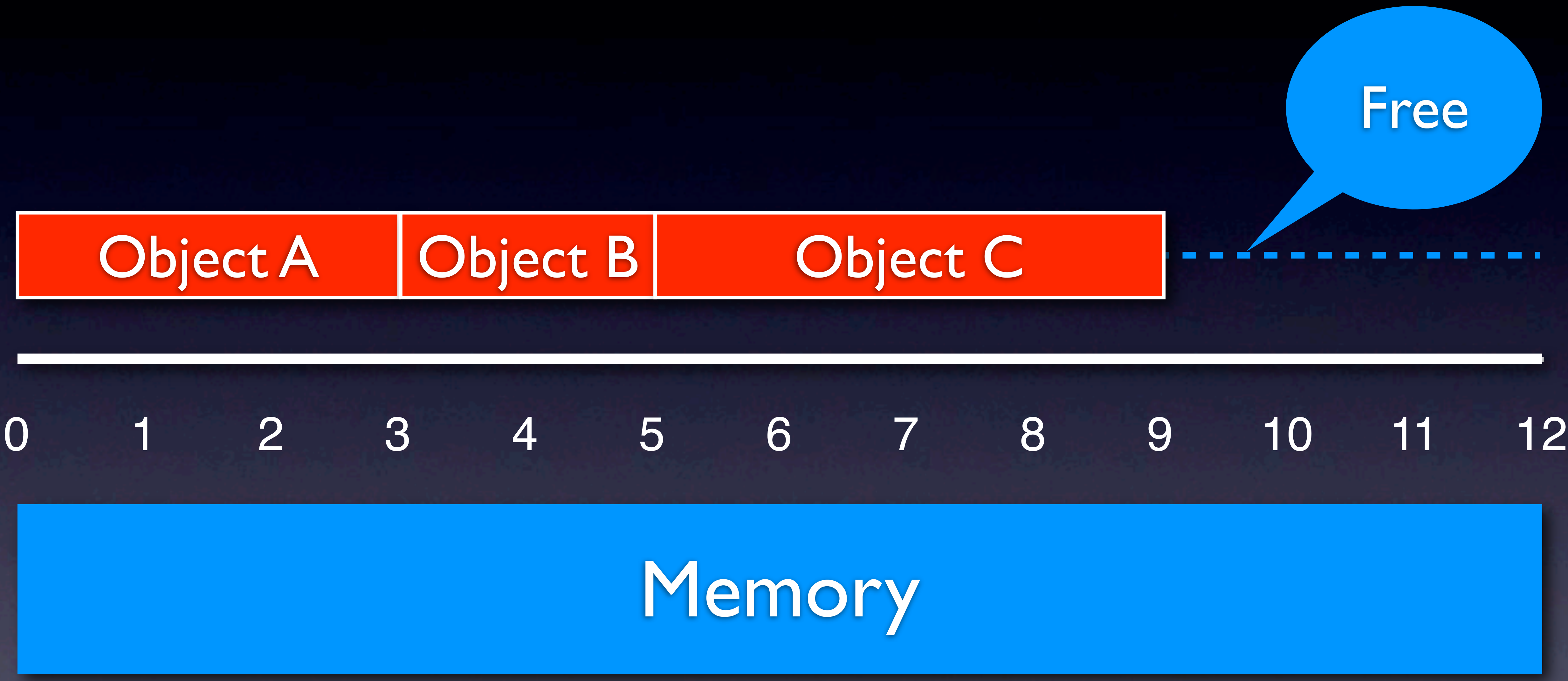
Allocation


Object A
Object B
Object C
Object D
0
1
2
3
4
5
6
7
8
9
10
11
12
Memory

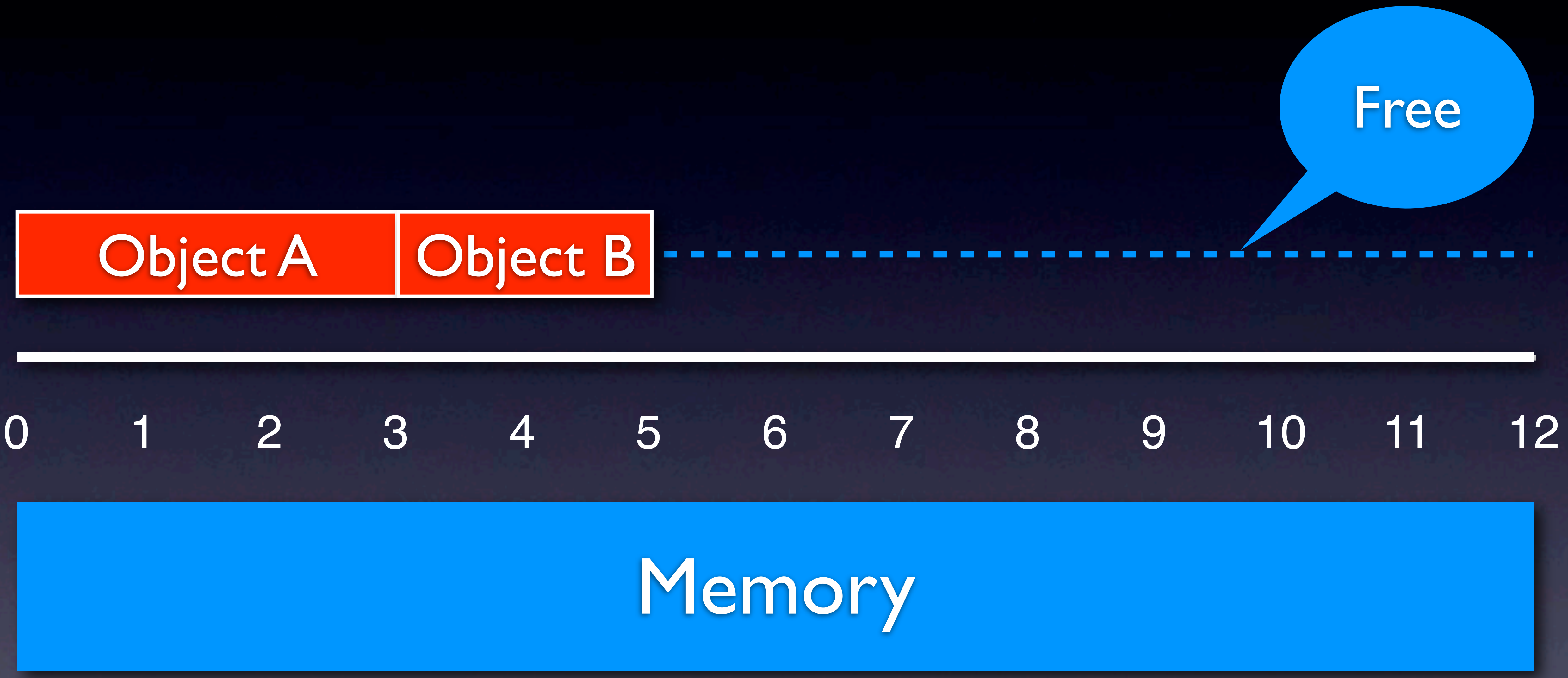
Deallocation



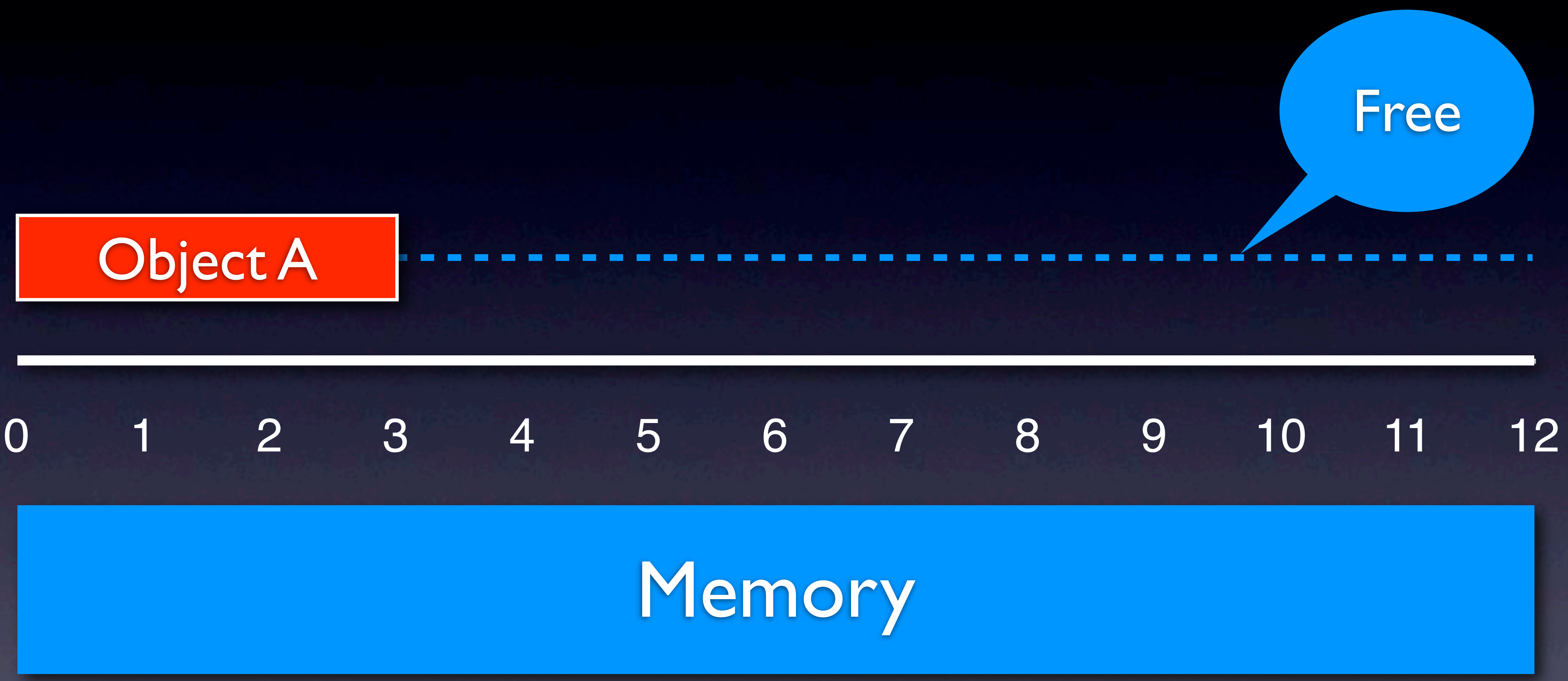
Deallocation



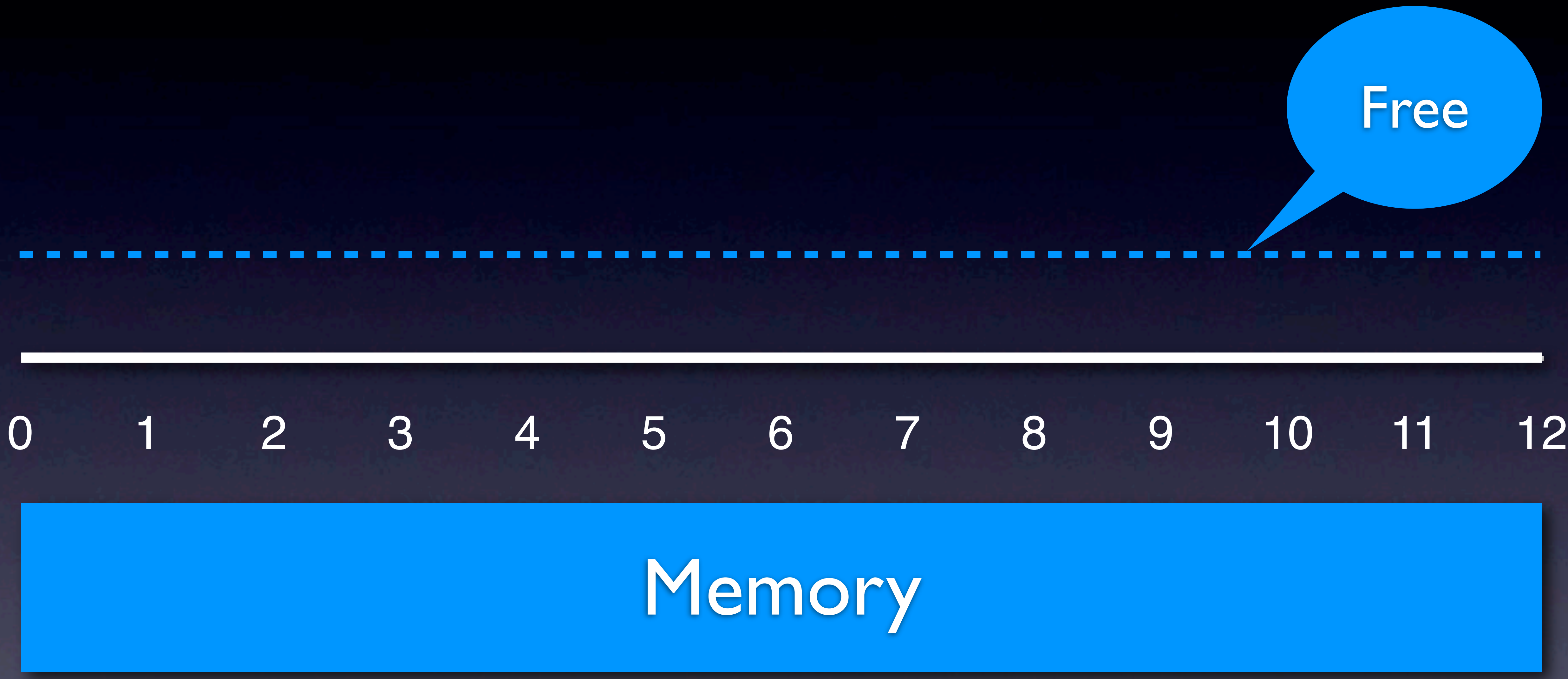
Deallocation



Deallocation



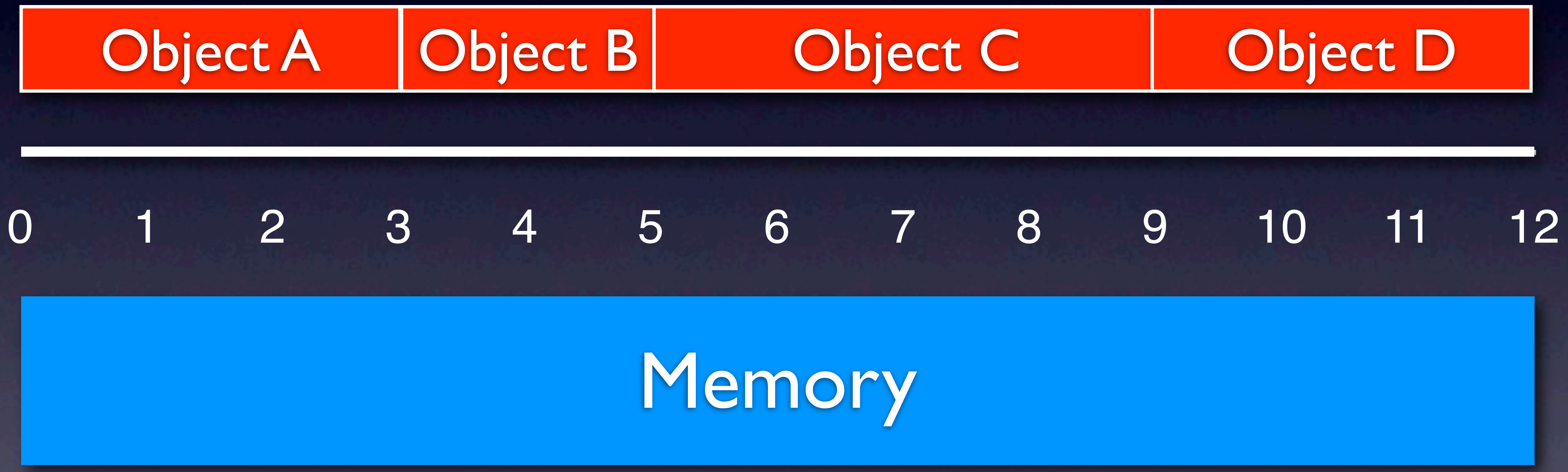
Deallocation



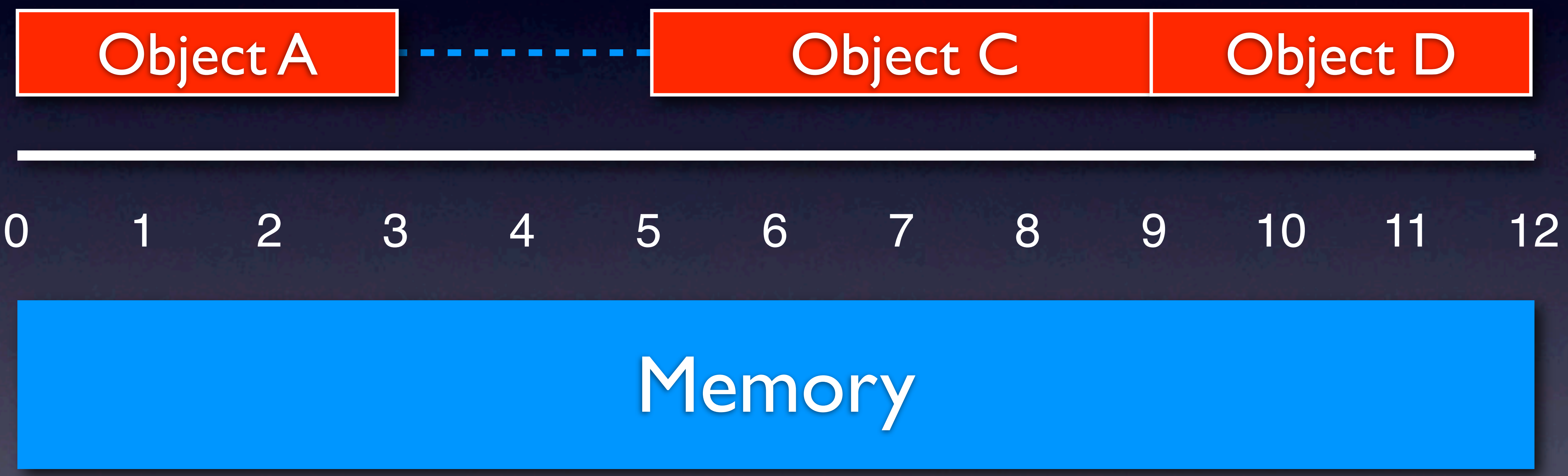
2. Assumption:

Objects may be allocated and deallocated in random order

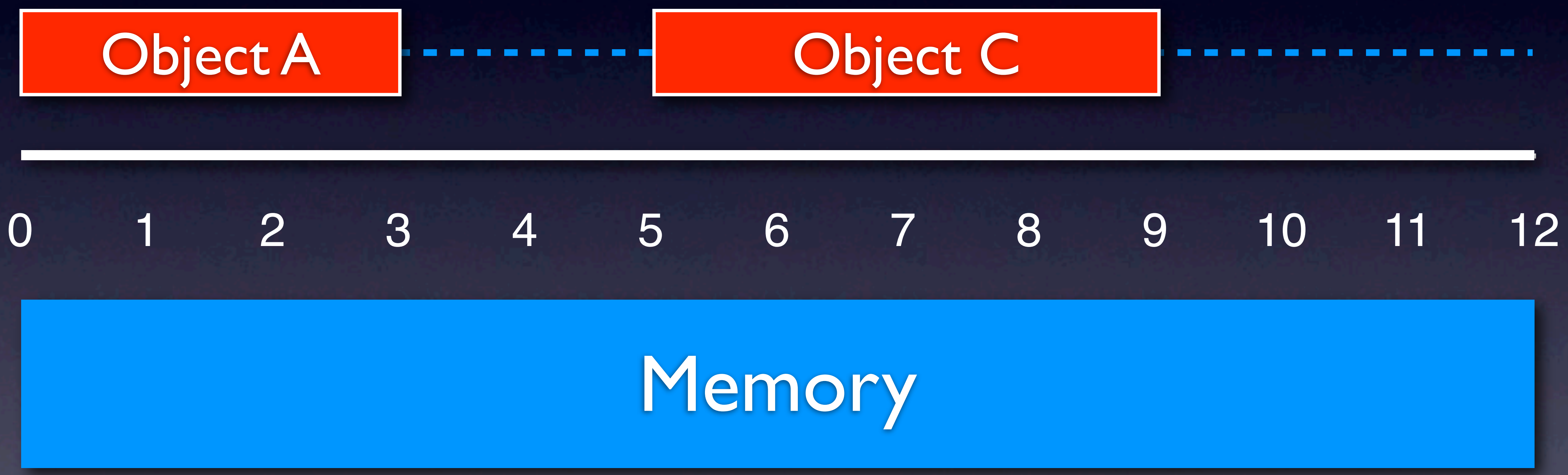
Deallocation



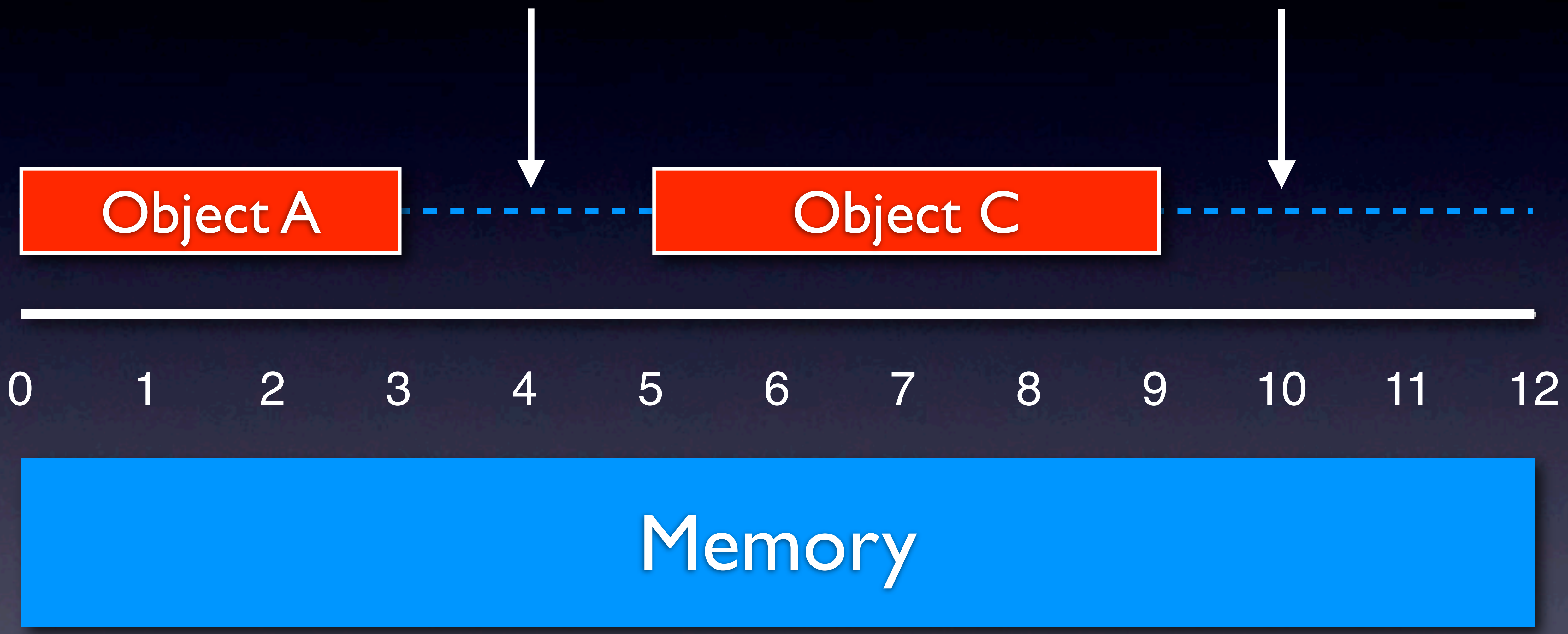
Deallocation



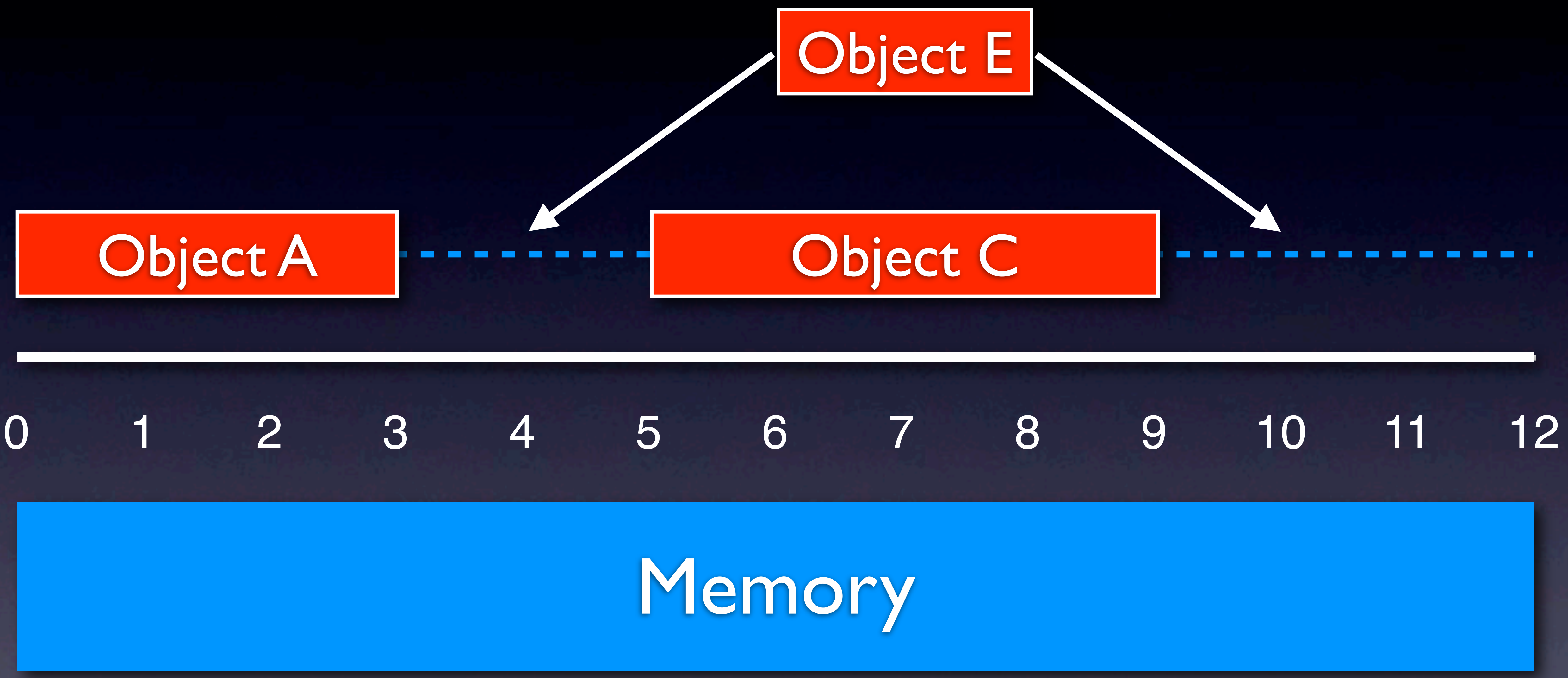
Deallocation



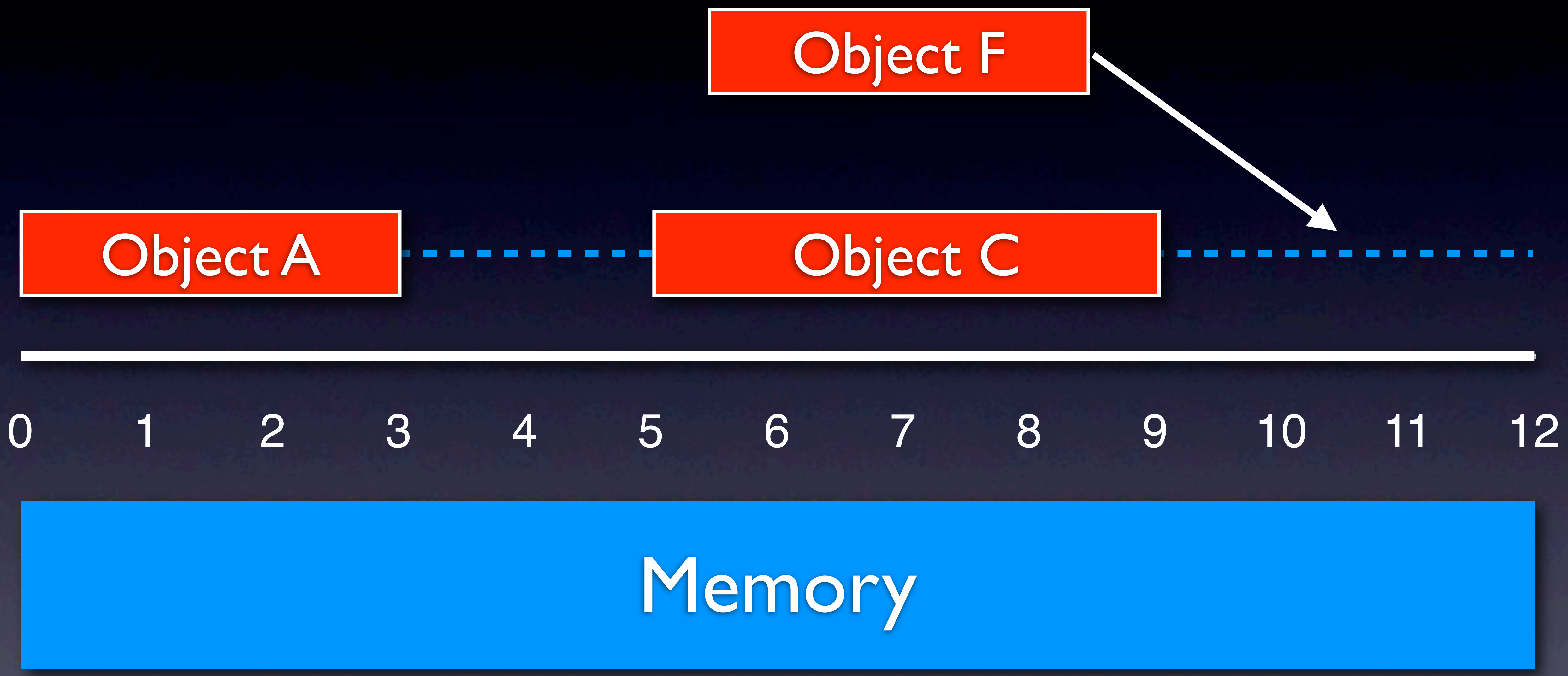
External Fragmentation



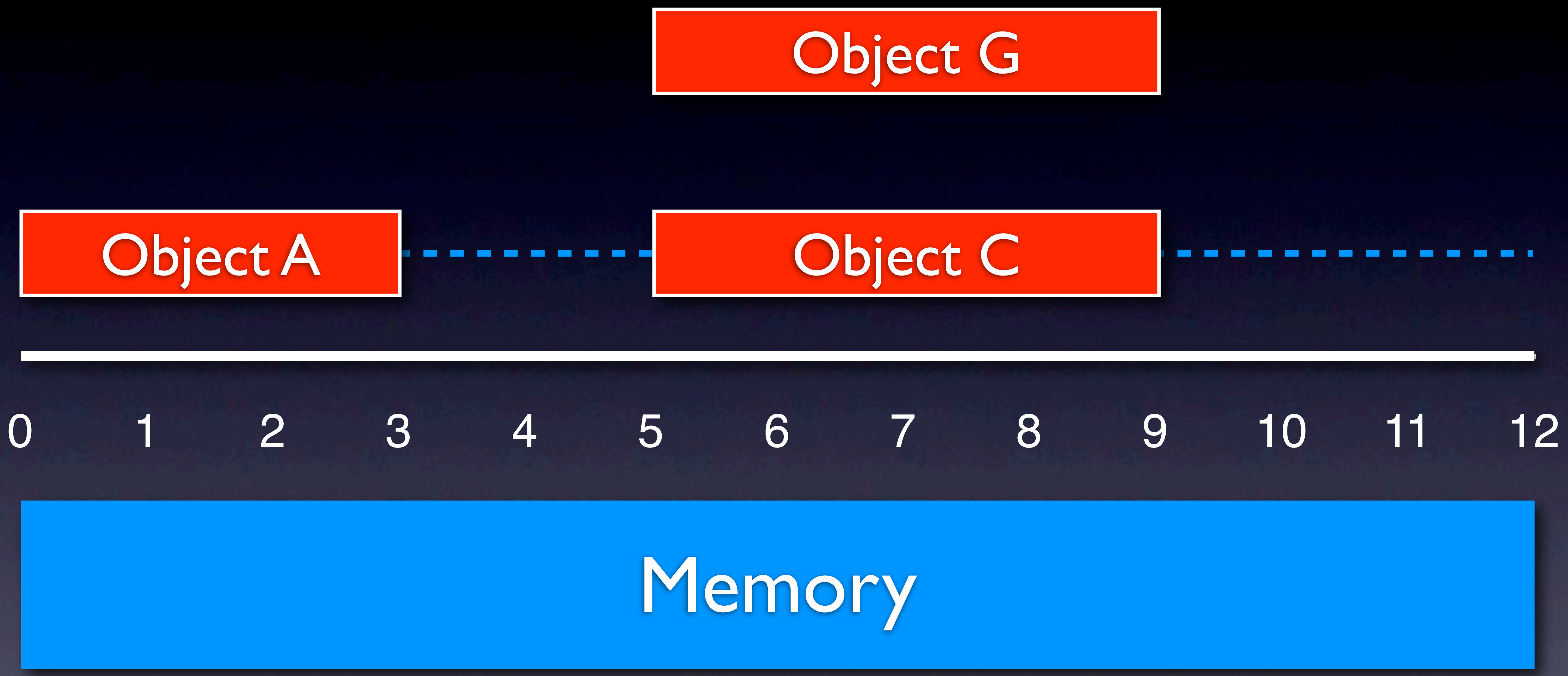
Allocation



Allocation



Allocation



Memory is *fragmented* if the largest, contiguous piece of available space is smaller than the total available space

Fragmentation

- Memory objects may have different sizes
- Memory objects may be allocated and deallocated in random order
  - creates the problem of memory fragmentation!

Explicit, Dynamic Memory Management with Temporal and Spatial Guarantees

Static versus Dynamic

- Static memory management:
  - Preallocate all memory at compile time

Static versus Dynamic

- Static memory management:
  - Preallocate all memory at compile time
- Dynamic memory management:
  - Allocate and deallocate memory at run time

Explicit, Dynamic Memory Management with Temporal and Spatial Guarantees

Implicit versus Explicit

- Implicit, dynamic memory management:
  - Garbage collector (GC) deallocates objects, not programmer (implicit `free` calls by GC)

Implicit versus Explicit

- Implicit, dynamic memory management:
  - Garbage collector (GC) deallocates objects, not programmer (implicit `free` calls by GC)
- Explicit, dynamic memory management:
  - Objects are deallocated by programmer (explicit `free` calls)

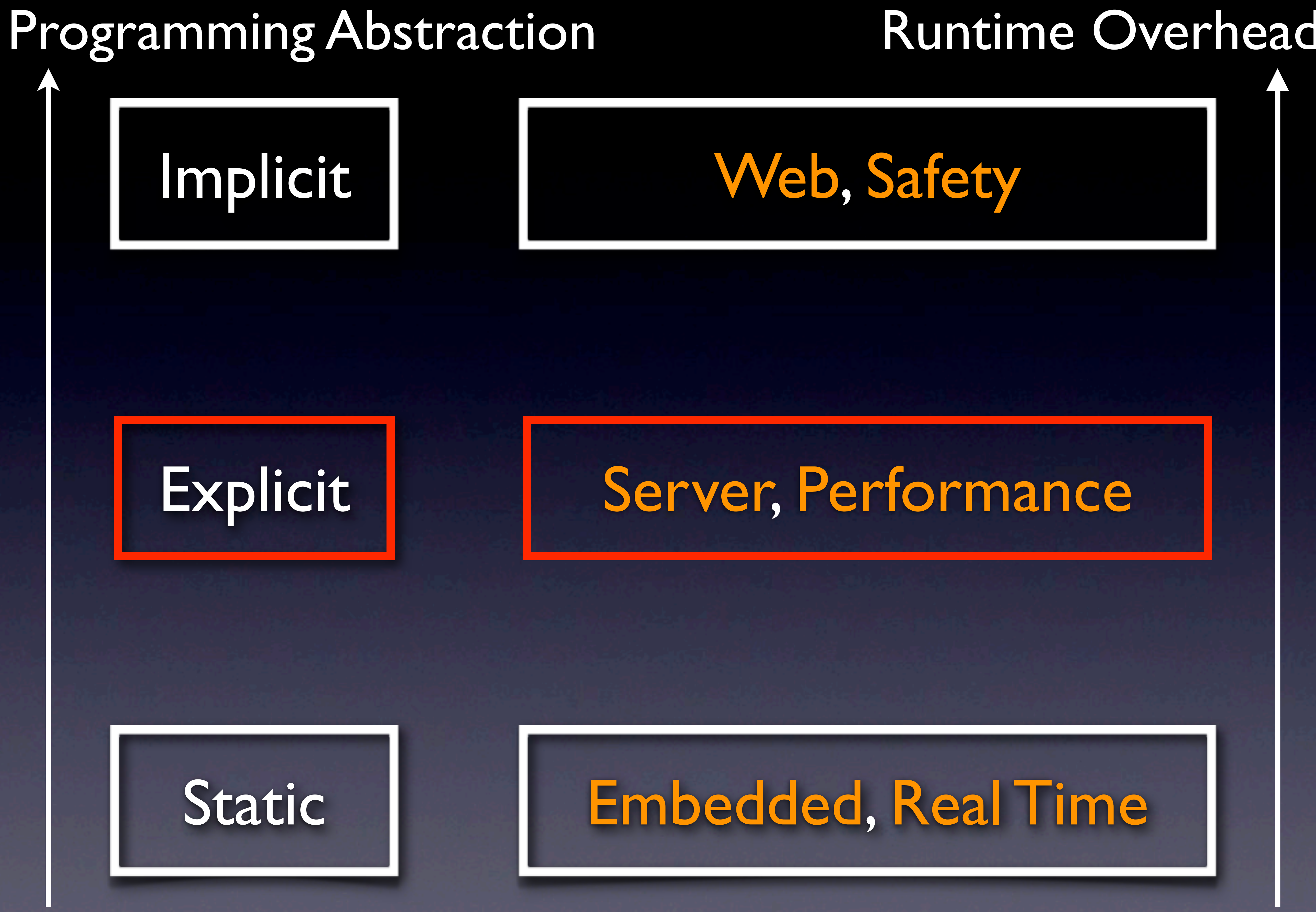
Programming Abstraction

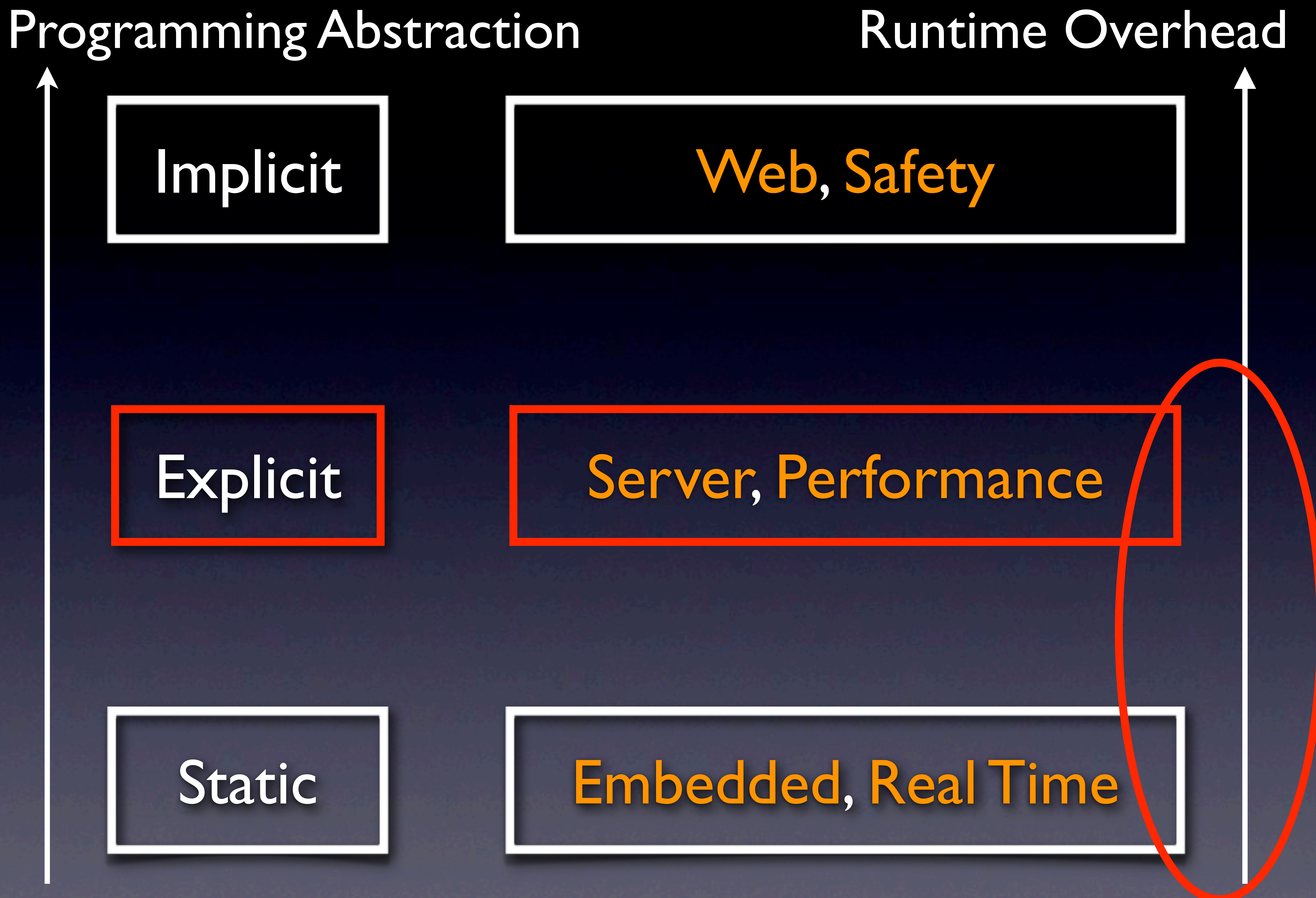
Runtime Overhead

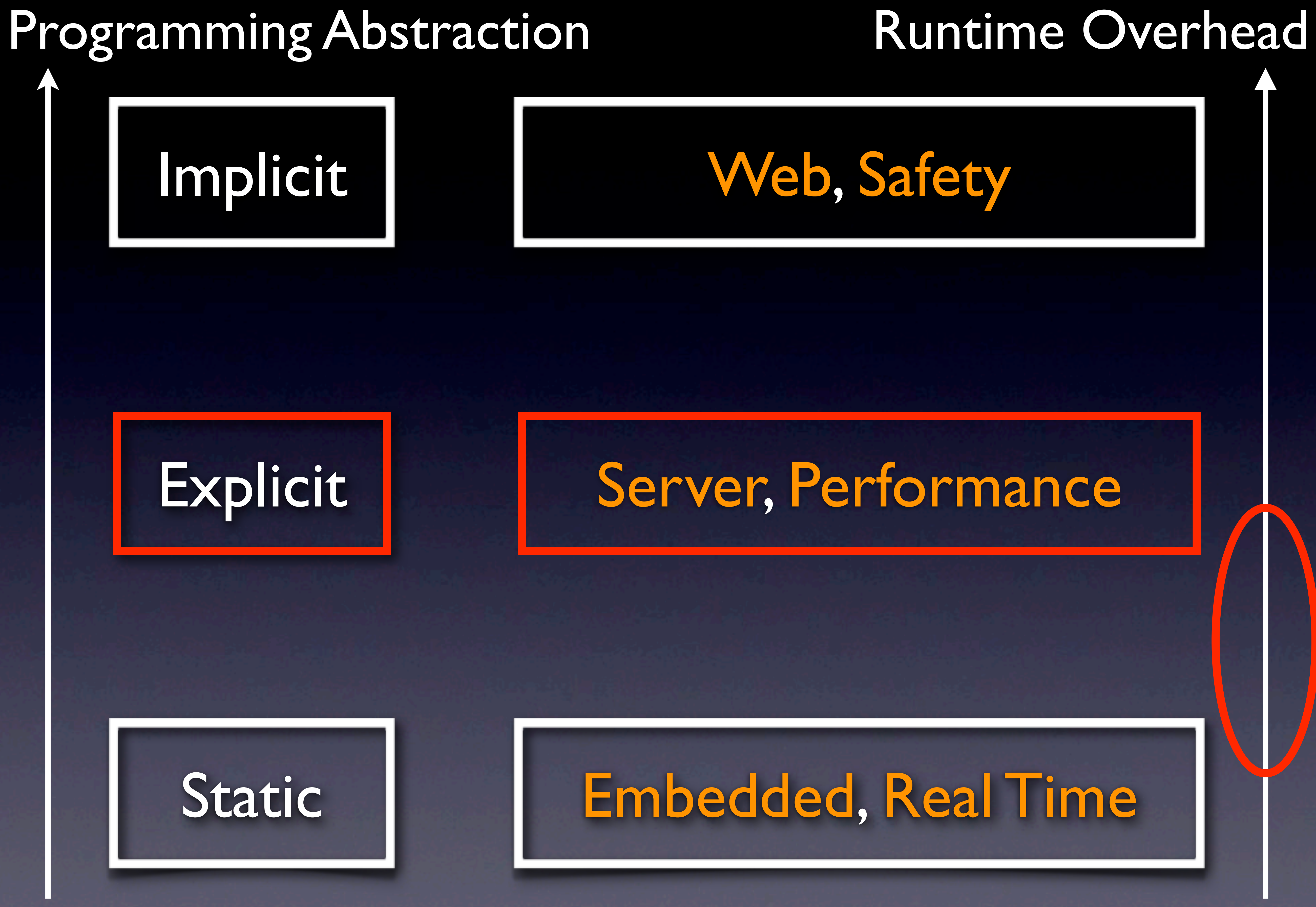
Implicit, Dynamic Memory Management

Explicit, Dynamic Memory Management

Static Memory Management


Programming Abstraction
Runtime Overhead
Implicit
Web, Safety
Explicit
Server, Performance
Static
Embedded, Real Time


Programming Abstraction
Runtime Overhead
Implicit
Web, Safety
Explicit
Server, Performance
Static
Embedded, Real Time


Programming Abstraction
Runtime Overhead
Implicit
Web, Safety
Explicit
Server, Performance
Static
Embedded, Real Time

Temporal Performance

- Throughput:
  - 10MB/s allocation rate
  - 10MB/s deallocation rate

Temporal Performance

- Throughput:
  - 10MB/s allocation rate
  - 10MB/s deallocation rate
- Latency/Responsiveness:
  - 1ms execution time (malloc/free)
  - 0.1ms preemption time (malloc/free)

Spatial Performance

- Degree of fragmentation:
  - The number of contiguous pieces of memory of a given size that can still be allocated

Spatial Performance

- Degree of fragmentation:
  - The number of contiguous pieces of memory of a given size that can still be allocated
- Administrative space:
  - meta data structures (used, free lists)

There is a <u>trade-off</u> between temporal and spatial performance

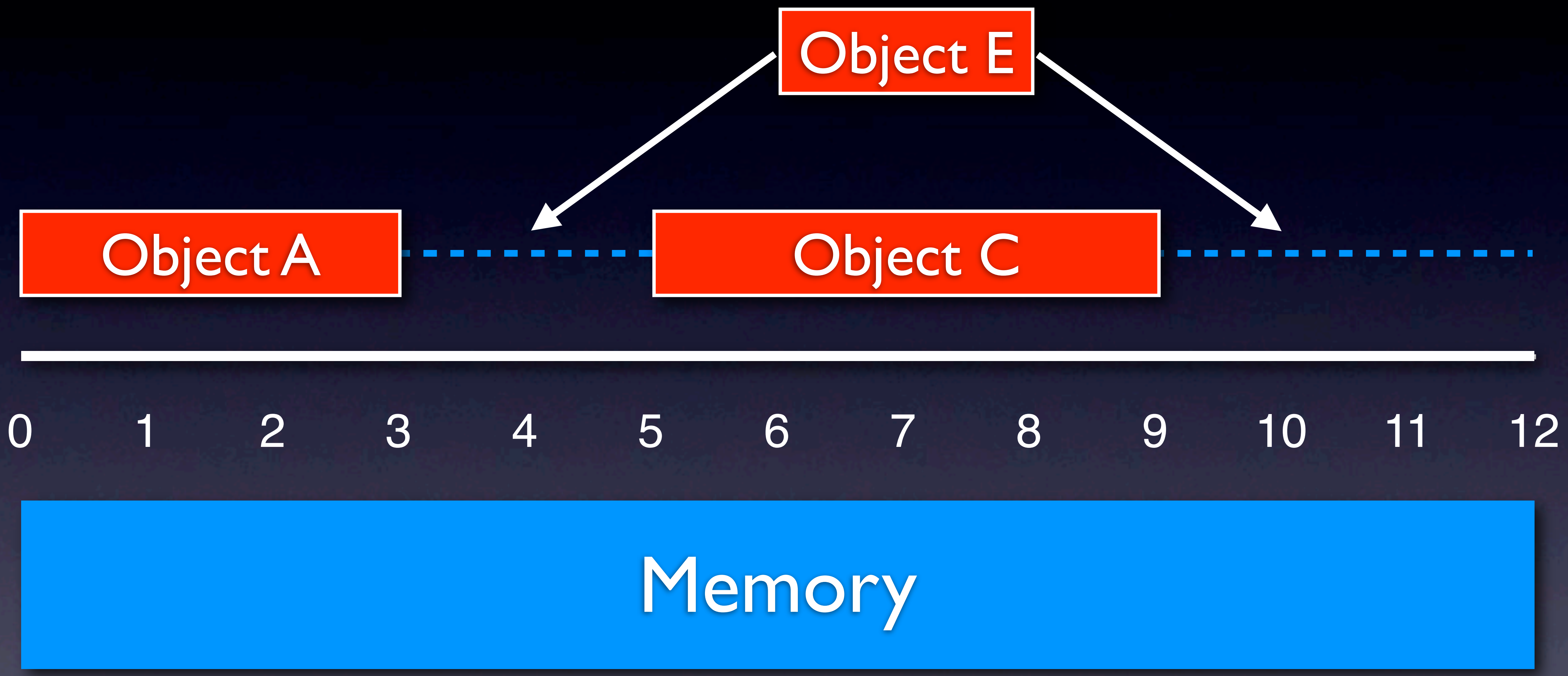
Temporal Predictability

- Unpredictable complexity (in terms of input):
  - allocation/deallocation may take time proportional to the total size of memory

Temporal Predictability

- Unpredictable complexity (in terms of input):
  - allocation/deallocation may take time proportional to the total size of memory
- Predictable complexity (in terms of input):
  - allocation/deallocation takes time at most proportional to the size of involved object
  - access takes time at most proportional to the size of involved object

Allocation Complexity



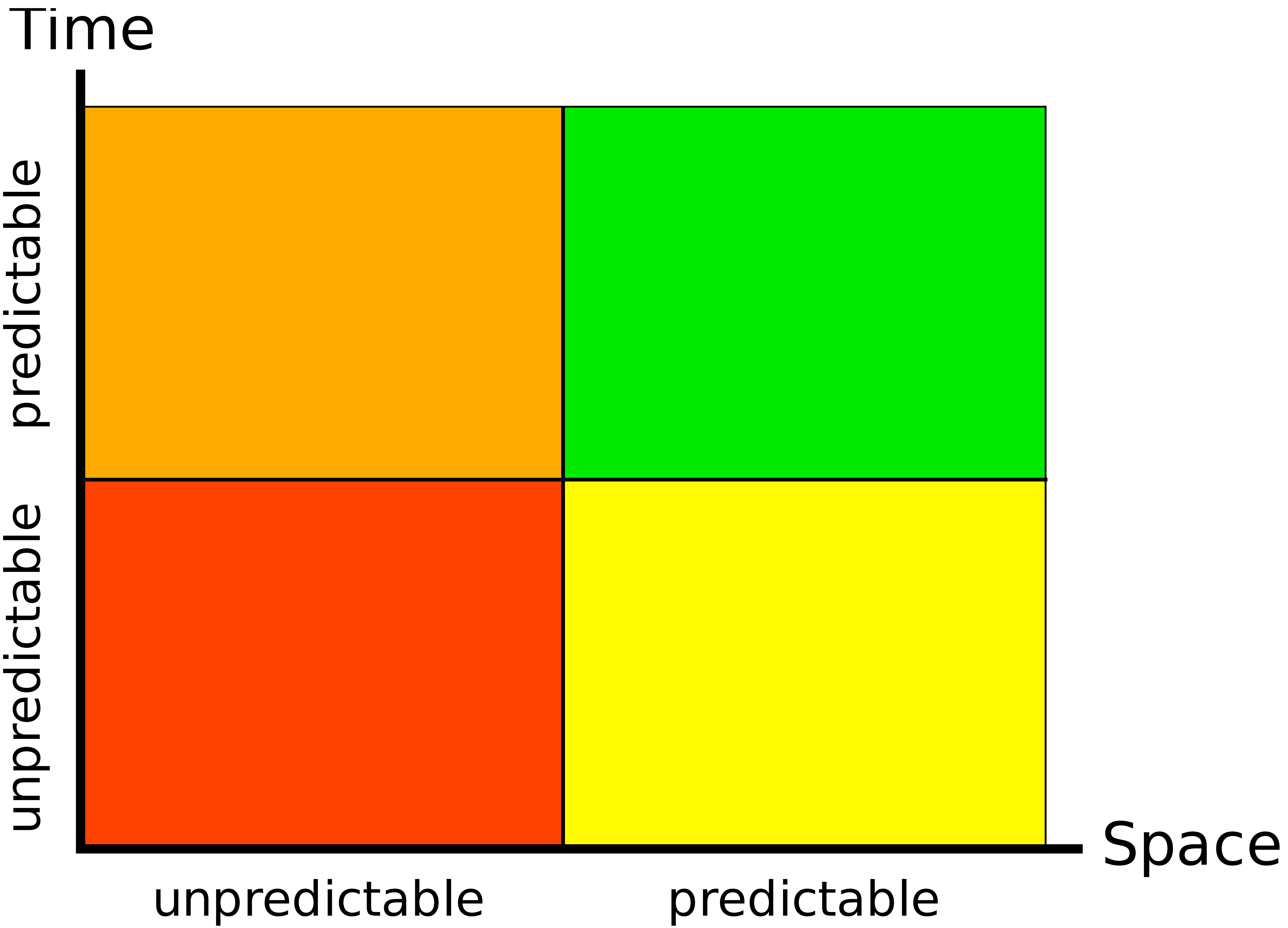
It may be <u>difficult</u> to improve
average performance
but it may still be <u>possible</u> to improve
predictability
without loosing too much
performance

Spatial Predictability

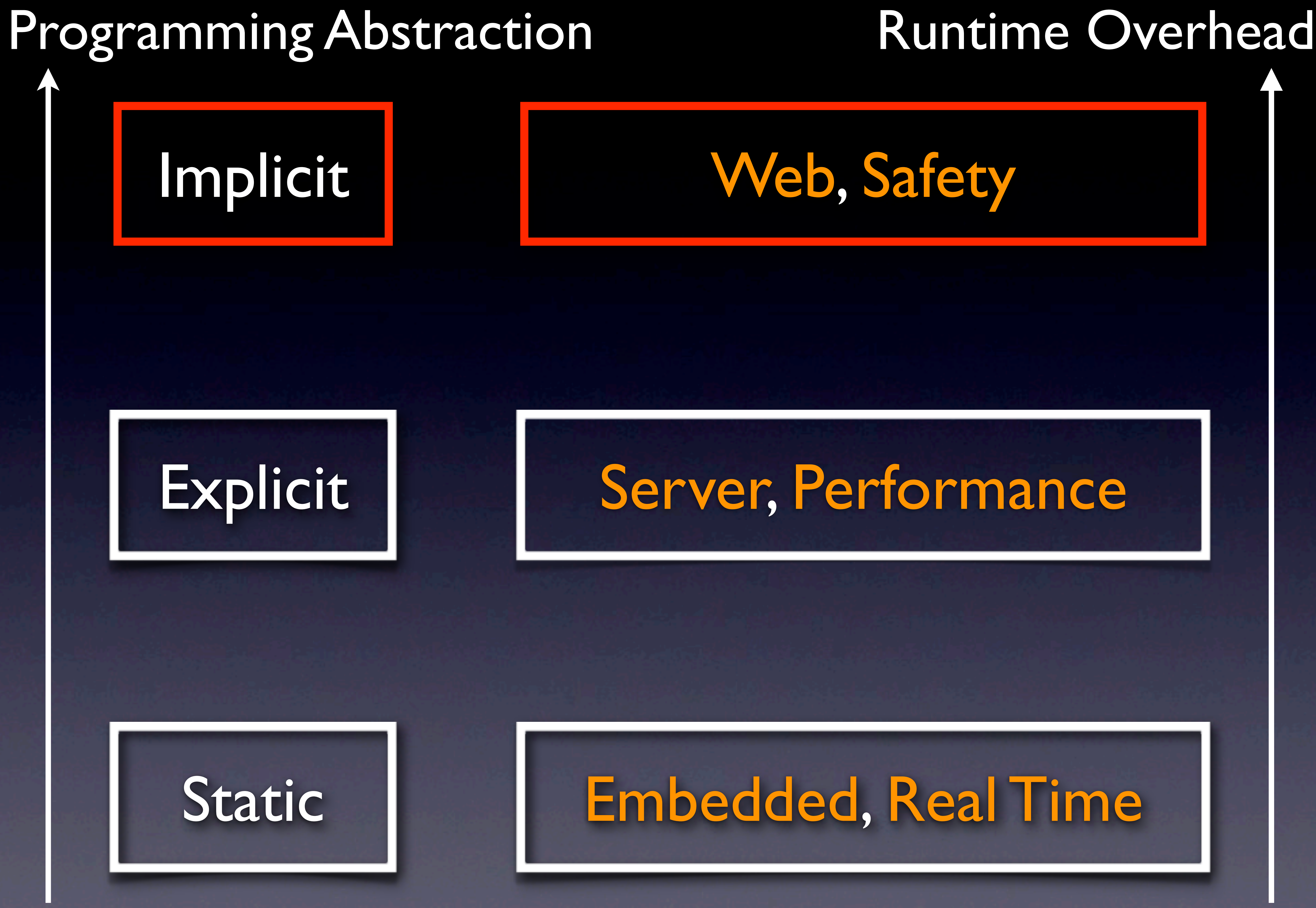
- Unpredictable fragmentation:
  - the degree of fragmentation may depend on the full allocation and deallocation history, i.e., the order of invocations

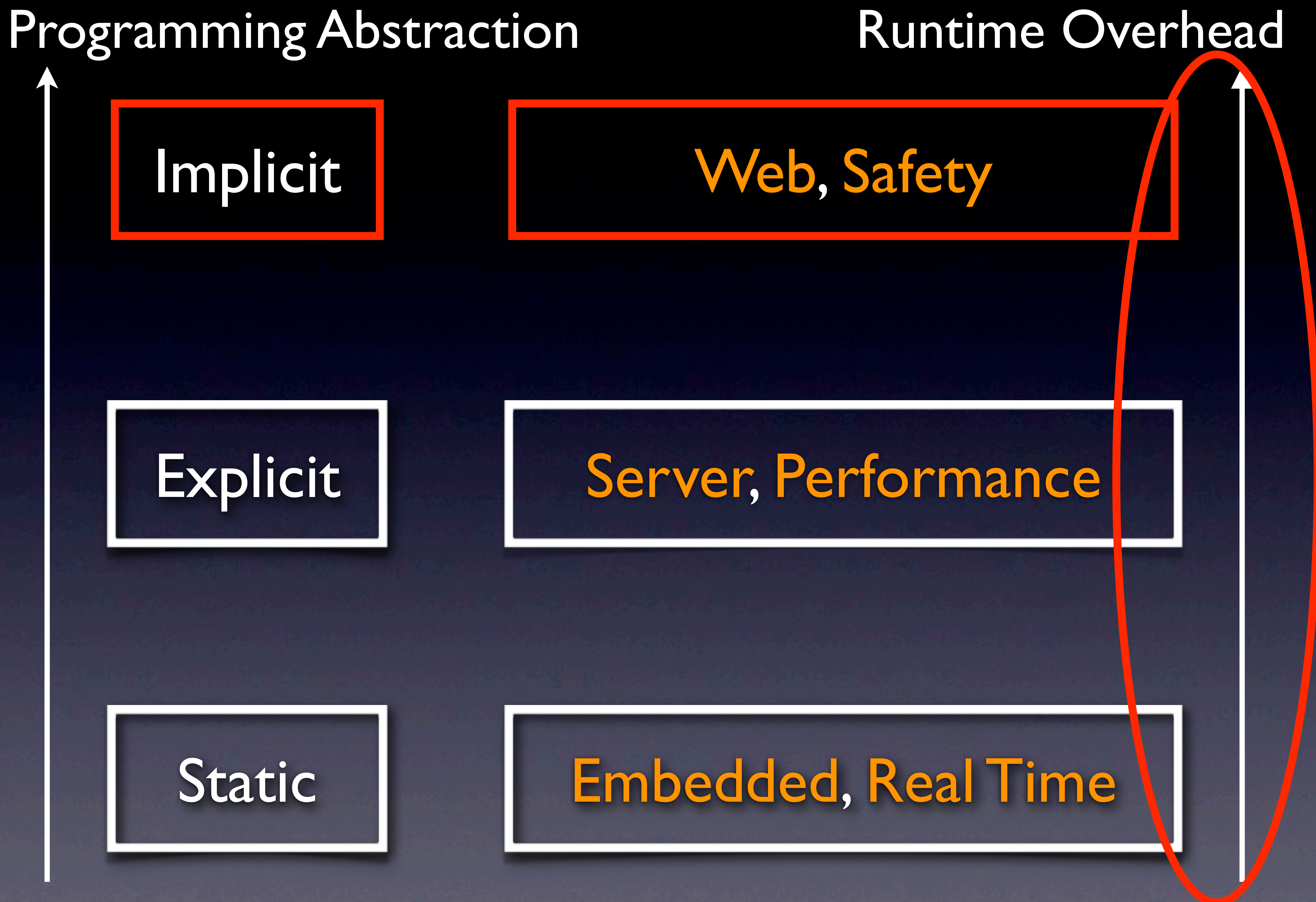
Spatial Predictability

- Unpredictable fragmentation:
  - the degree of fragmentation may depend on the full allocation and deallocation history, i.e., the order of invocations
- Predictable fragmentation:
  - the degree of fragmentation only depends on the number of allocations and deallocations, independently of the order of invocations

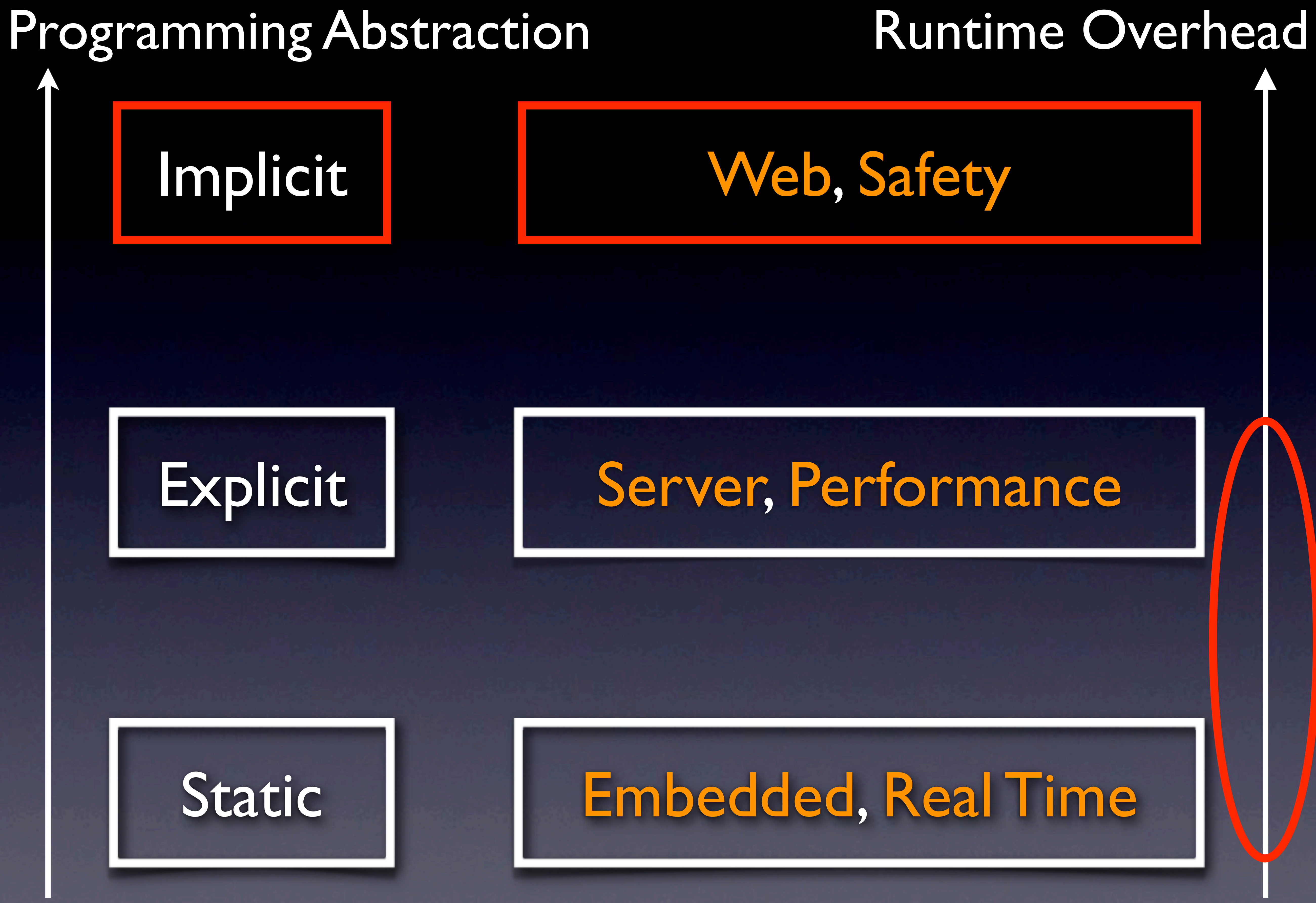

Time
predictable
unpredictable
Space
unpredictable
predictable

Explicit, Dynamic Memory Management with Temporal and Spatial Guarantees


Programming Abstraction
Runtime Overhead
Implicit
Web, Safety
Explicit
Server, Performance
Static
Embedded, Real Time



Programming Abstraction
Runtime Overhead
Implicit
Web, Safety
Explicit
Server, Performance
Static
Embedded, Real Time


Programming Abstraction
Runtime Overhead
Implicit
Web, Safety
Explicit
Server, Performance
Static
Embedded, Real Time

tiptoe.cs.uni-salzburg.at[#]

- Silviu Craciunas[#] (Programming Model)
- Andreas Haas (Memory Management)
- Hannes Payer[#] (Memory Management)
- Harald Röck (VM, Scheduling)
- Ana Sokolova* (Theoretical Foundation)

[#]Supported by a 2007 IBM Faculty Award, the EU ArtistDesign Network of Excellence on Embedded Systems Design, and Austrian Science Fund Project P18913-N15.
*Supported by Austrian Science Fund Project V00125.

Tiptoe

- Tiptoe is a microkernel-based virtual machine and process monitor for embedded systems

Tiptoe

- Tiptoe is a microkernel-based virtual machine and process monitor for embedded systems
- Tiptoe virtualizes the host platform (system VM) and provides infrastructure to run process VMs and processes in real time

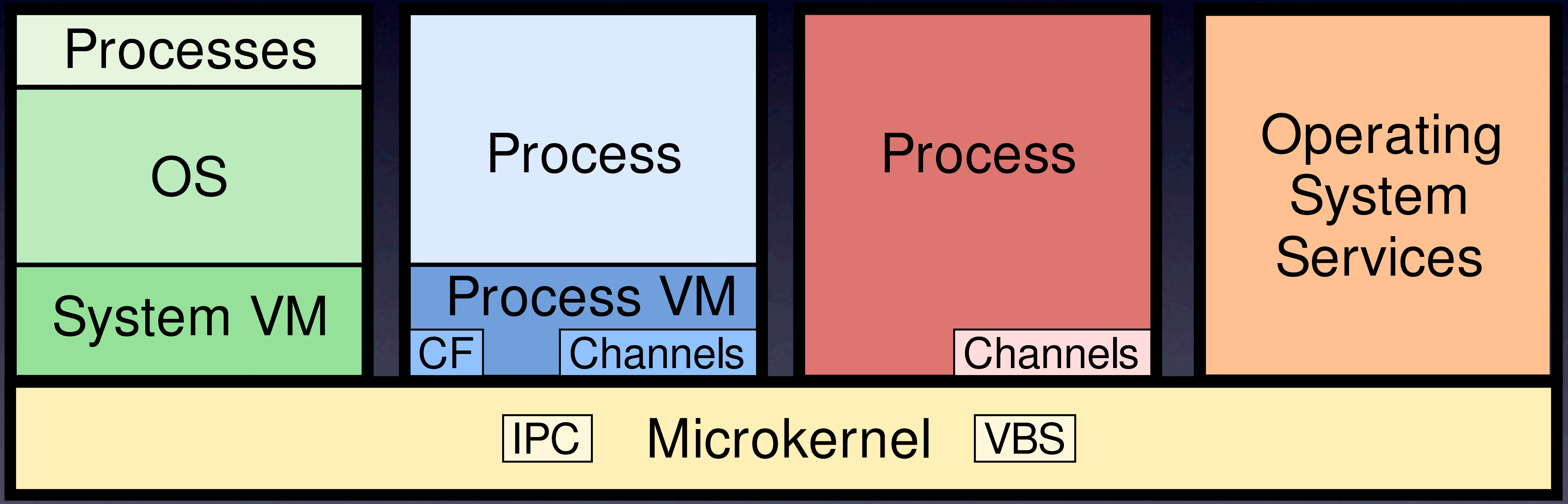
Tiptoe

- Tiptoe is a microkernel-based virtual machine and process monitor for embedded systems
- Tiptoe virtualizes the host platform (system VM) and provides infrastructure to run process VMs and processes in real time
- Tiptoe controls throughput and latency of CPU, memory, and I/O

Tiptoe

- Tiptoe is a microkernel-based virtual machine and process monitor for embedded systems
- Tiptoe virtualizes the host platform (system VM) and provides infrastructure to run process VMs and processes in real time
- Tiptoe controls throughput and latency of CPU, memory, and I/O
- I/O is multiplexed through IPC to a system VM running Linux

Tiptoe

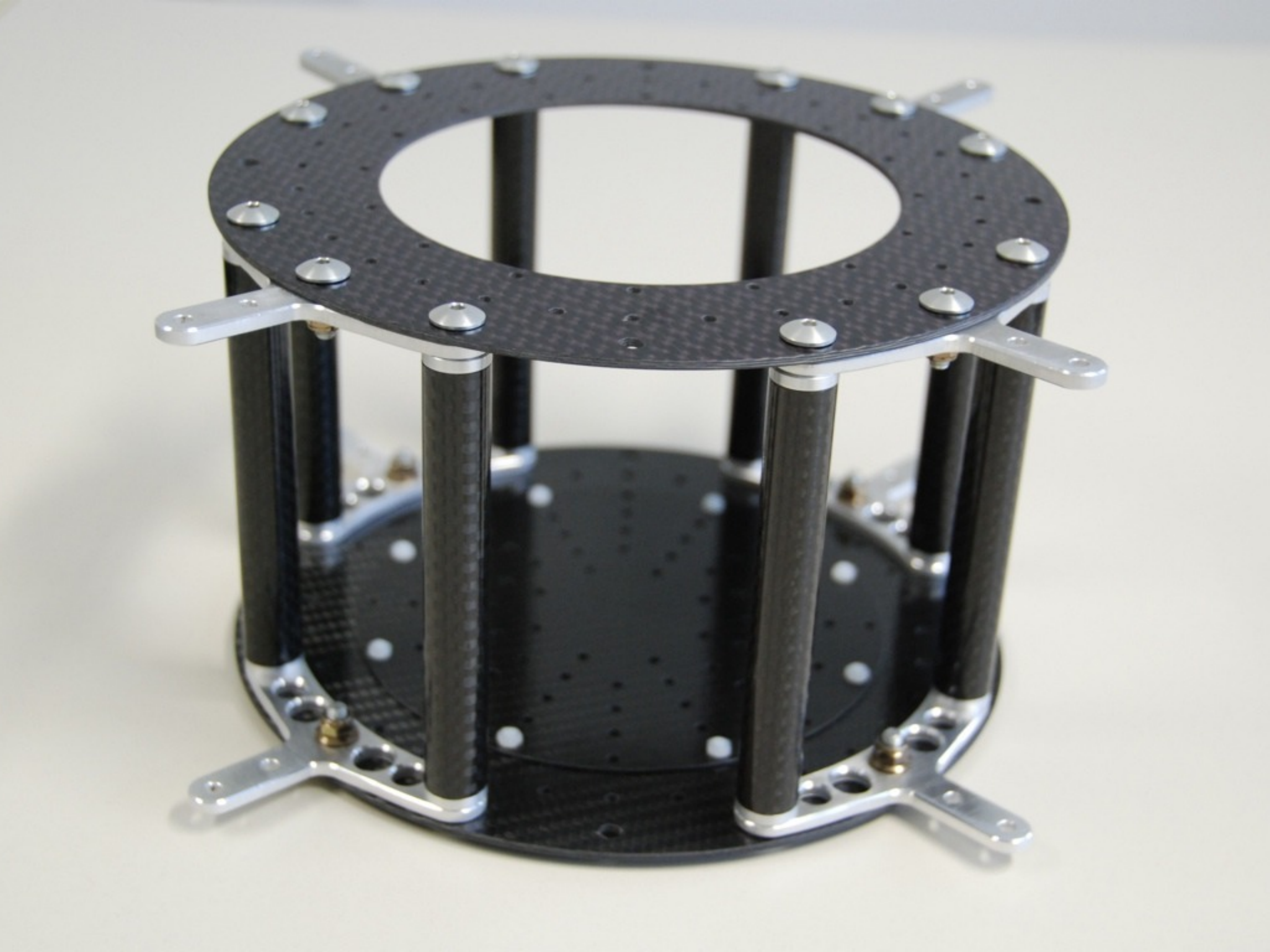


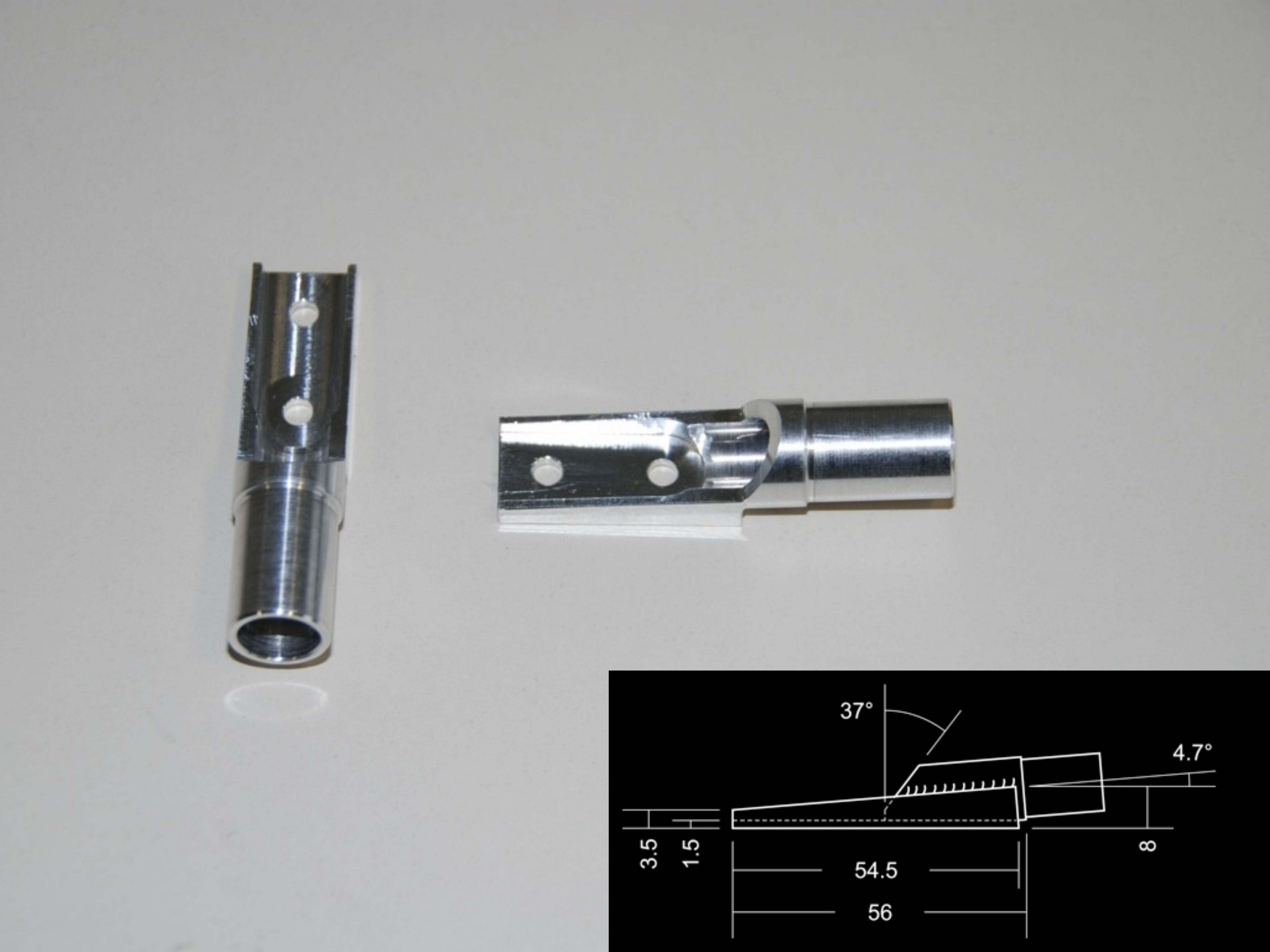
Processes
OS
System VM
Process
Process VM
CF
Channels
Process
Channels
Operating System Services
IPC
Microkernel
VBS

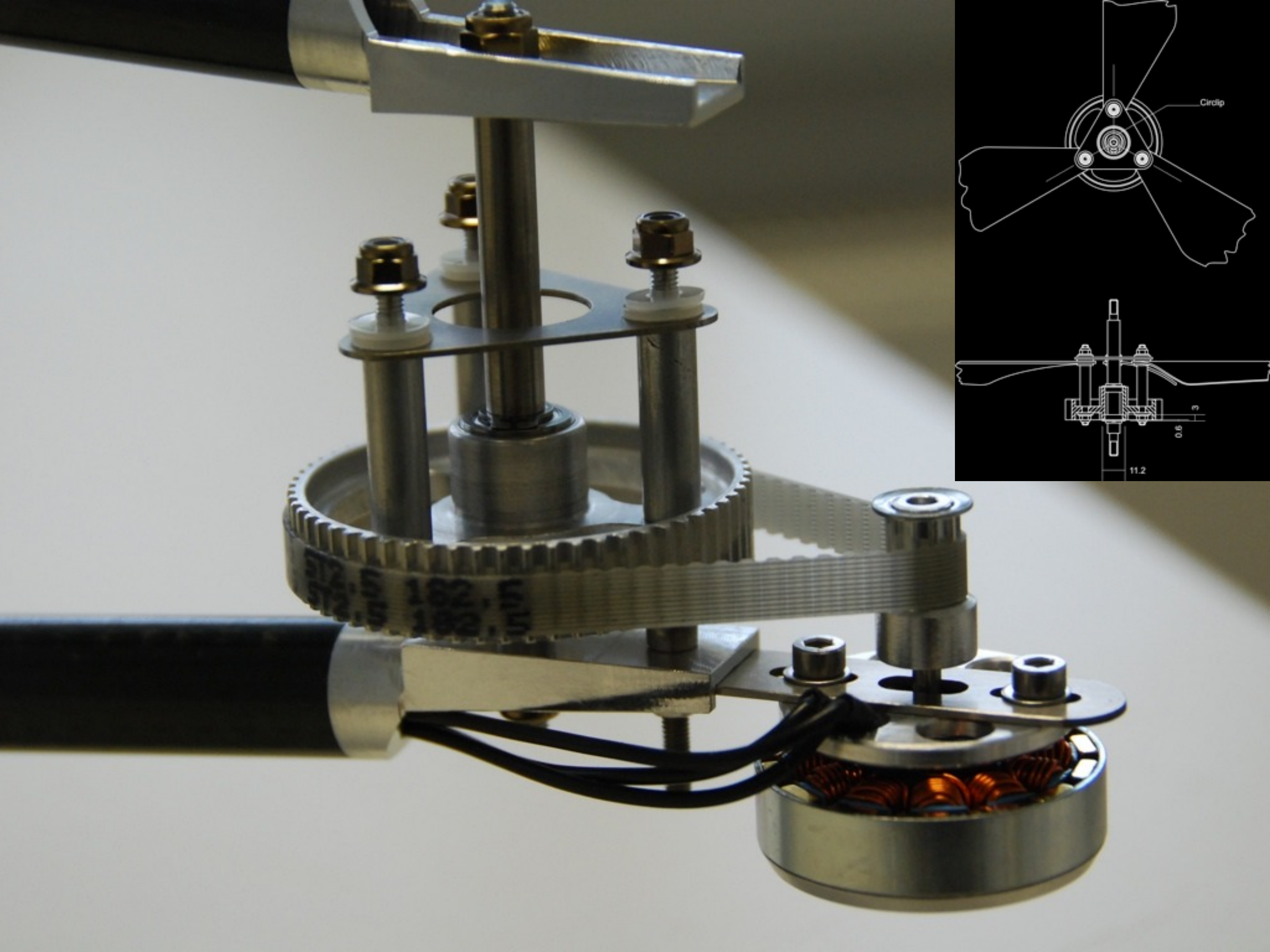
The JAviator

javiator.cs.uni-salzburg.at

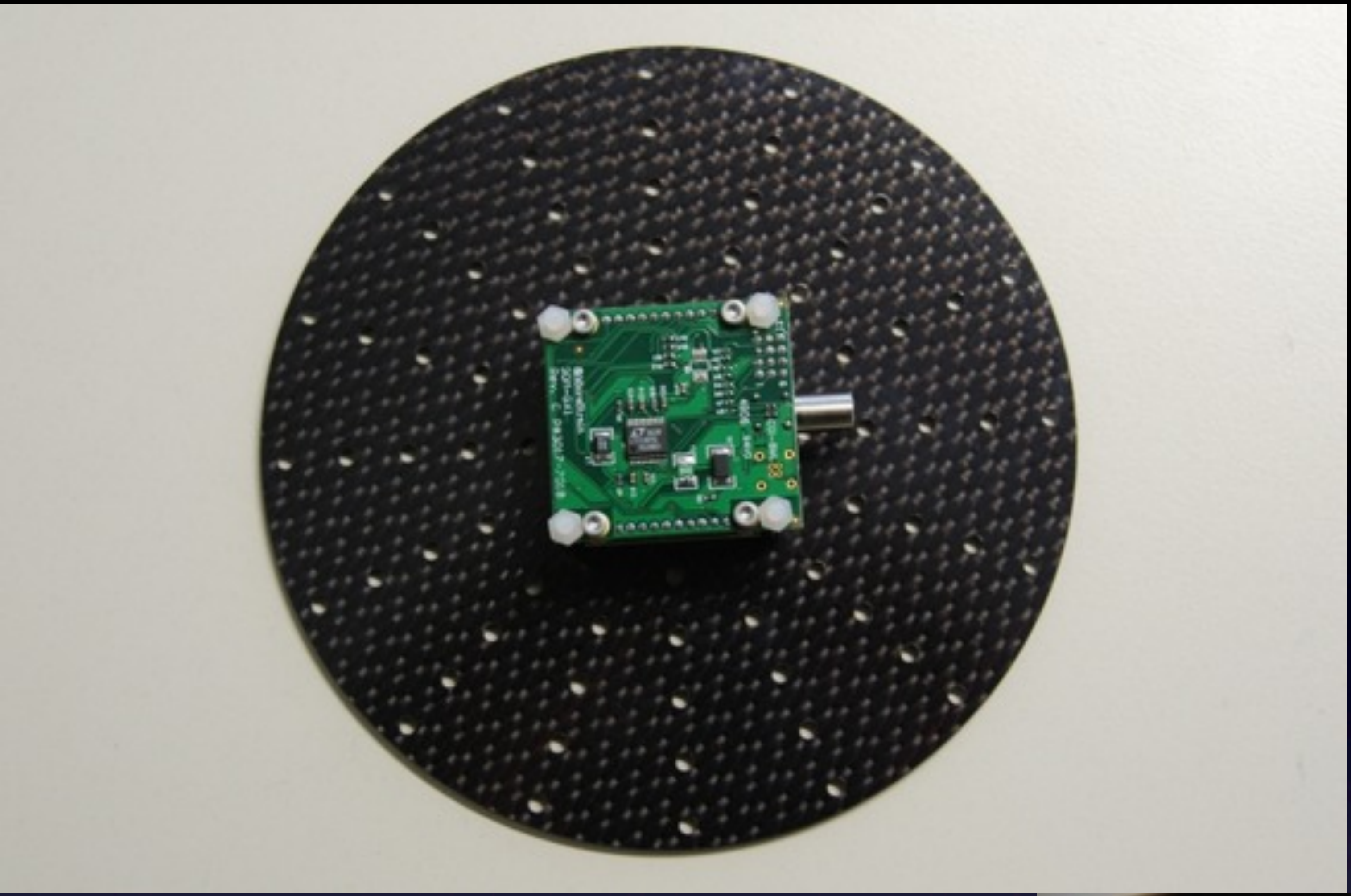
Quad-Rotor Helicopter




37°
4.7°
3.5
1.5
8
54.5
56

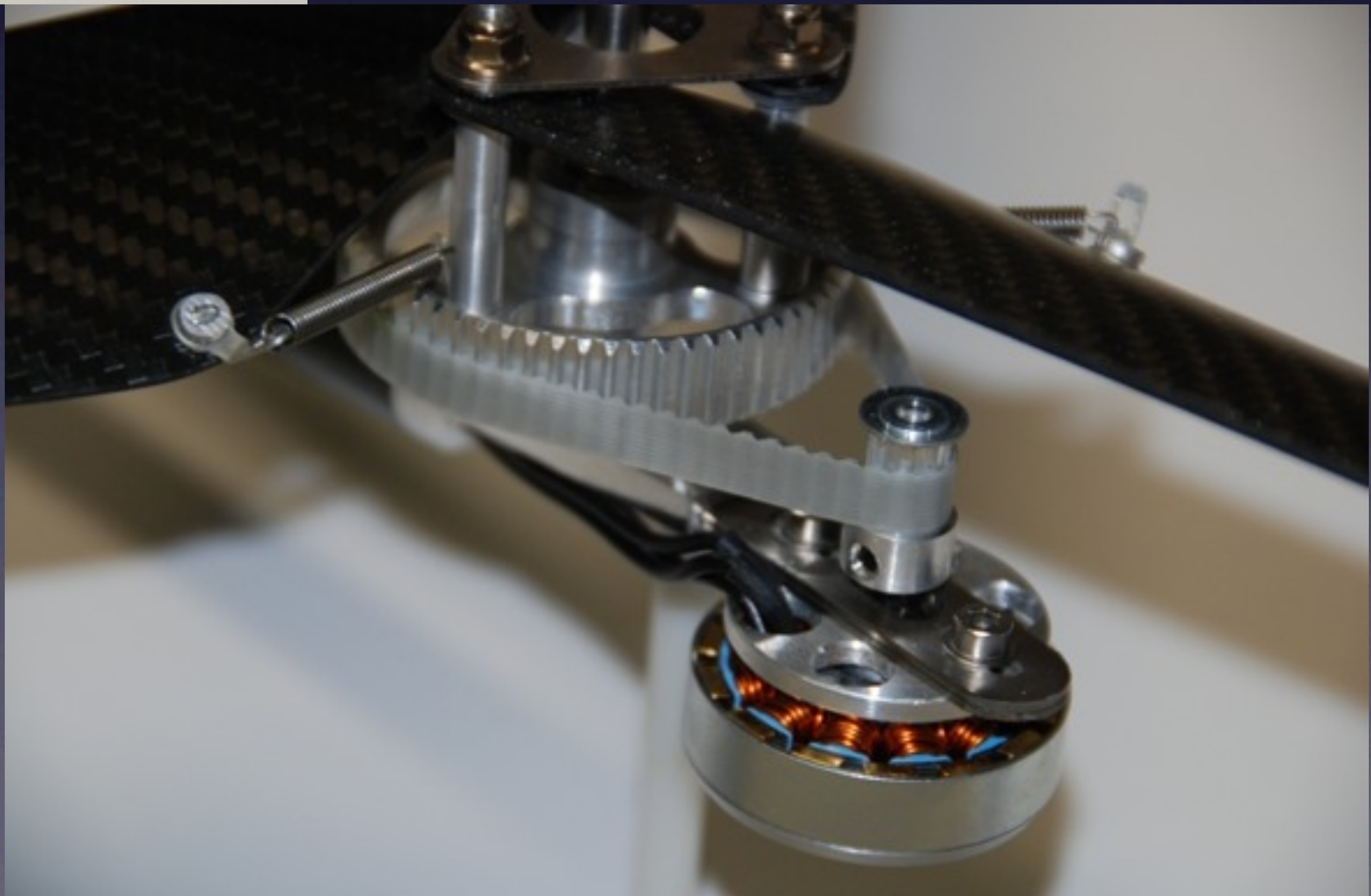

Circlip
3
0.6
11.2
ST2.5 182.5


25/12/2005

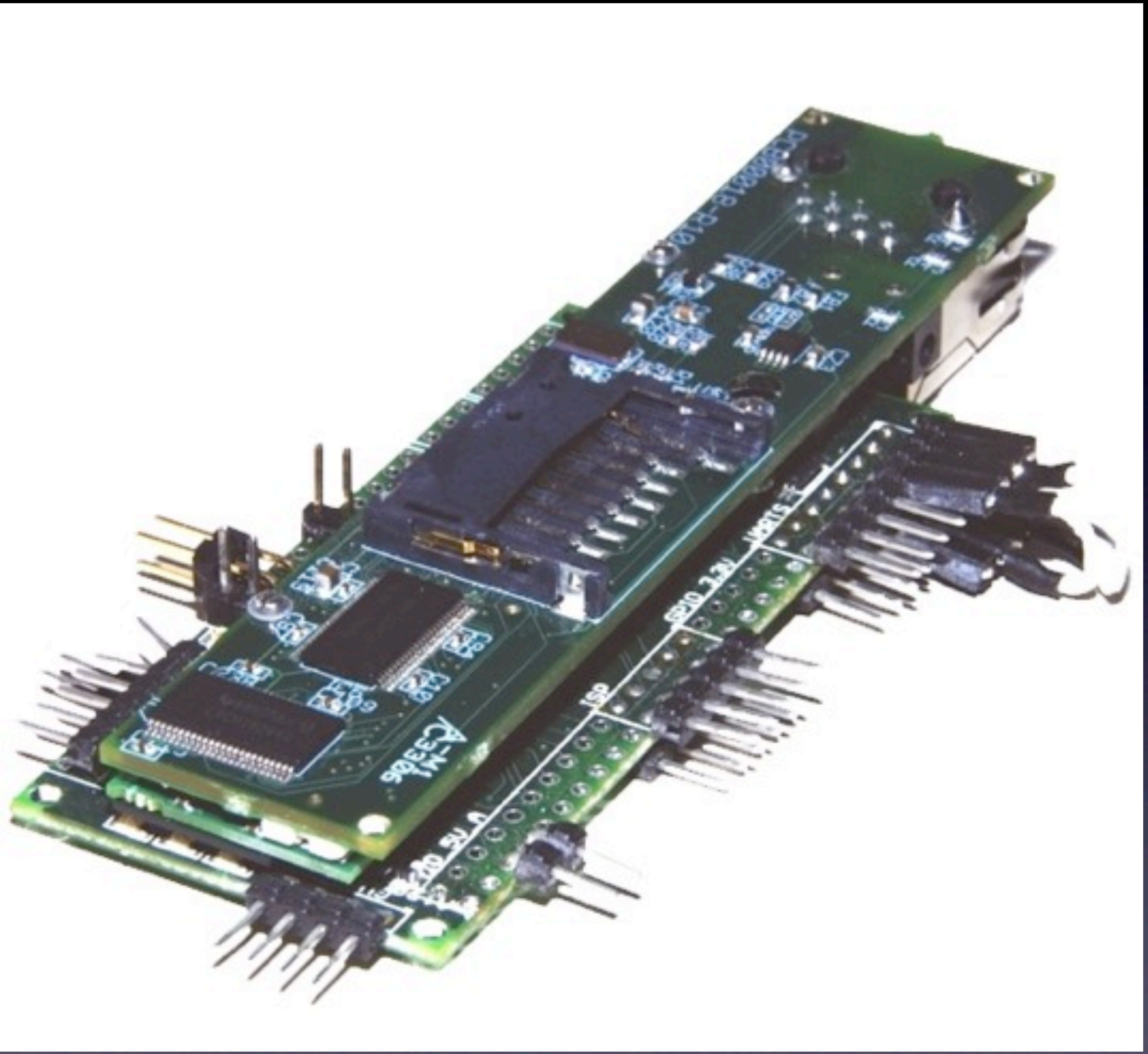


Gyro

Propulsion

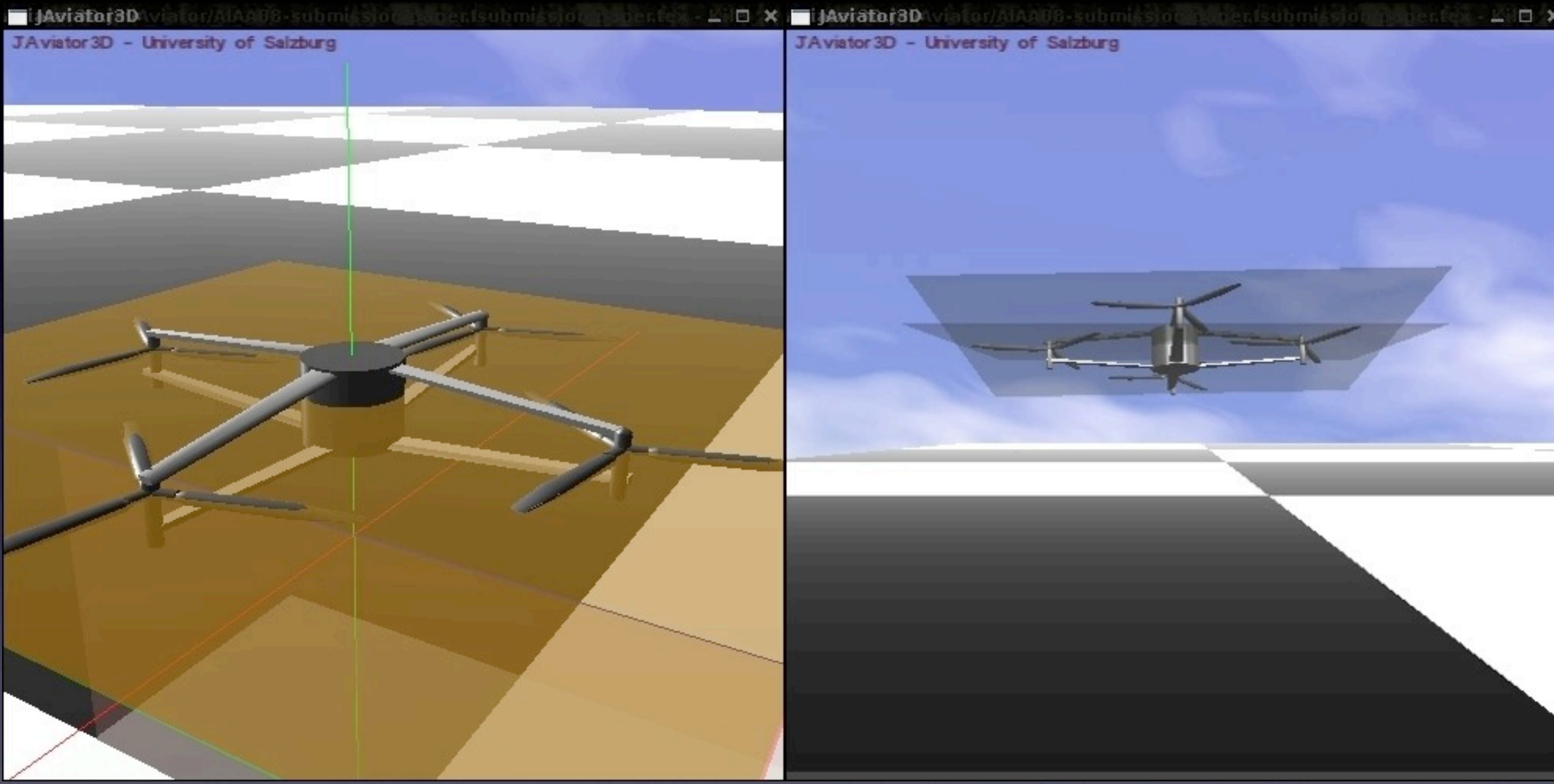


Gumstix



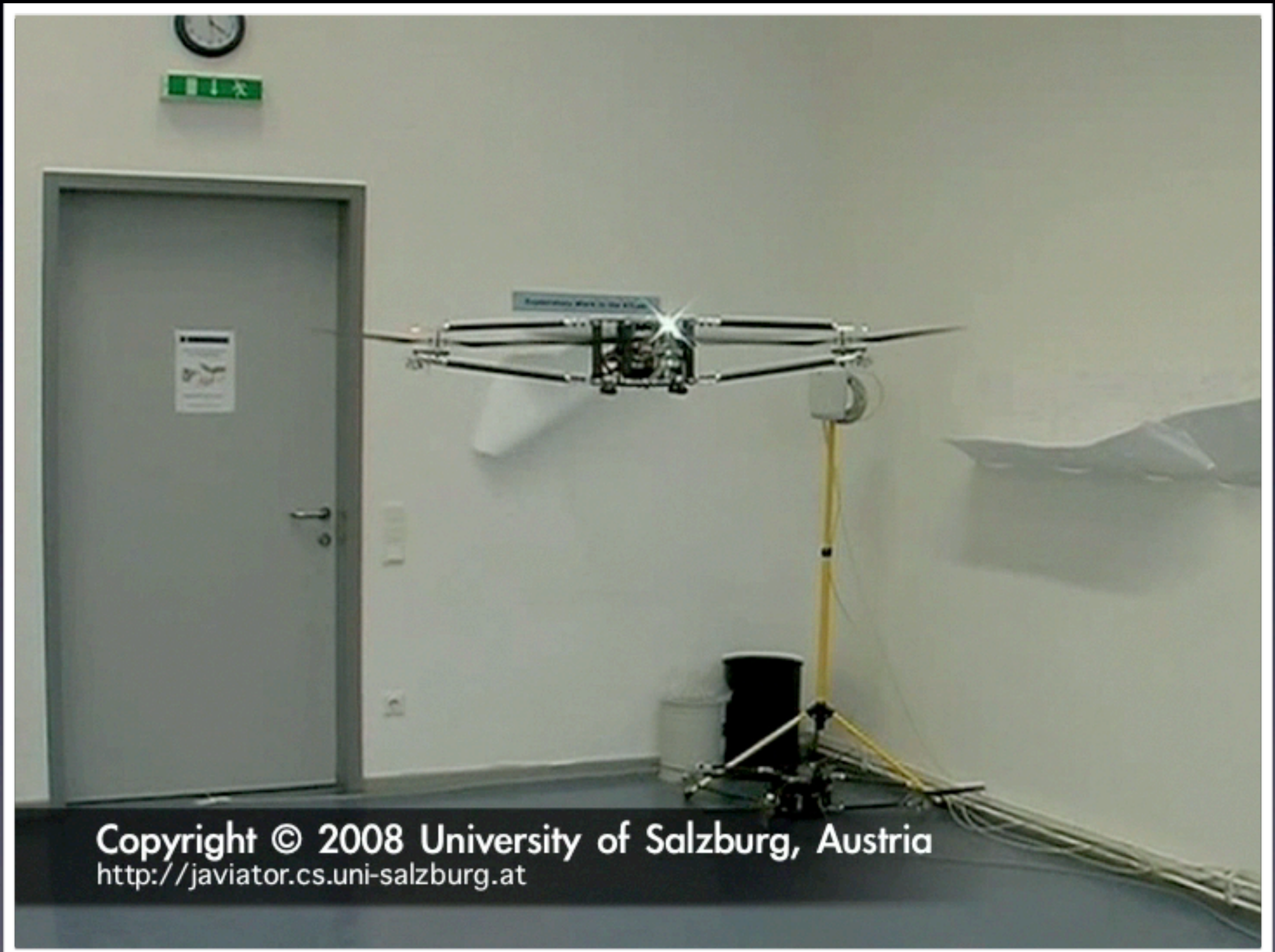
600MHz XScale, 128MB RAM, WLAN, Atmega uController


JAviator Control Terminal
– + Yaw Limit
Joystick Mode
Direct Connection
Connect to Heli
Start Test Mode
Start Logging
Roll Limit – +
Yaw
deg
Roll
deg
Front
Left
Hover
Right
Rear
-1
-1
0
1
Offsets
1
– + 33 Yaw Trim
Roll Trim 0 – +
0
-1
0
Altitude
cm
Pitch
deg
351
353
Signals
352
351
Pitch Trim 0 – +
– + Altitude Limit
Port Settings
Reset Needles
Shut down Heli
Key Assistance
About Terminal
Pitch Limit – +


JAviator3D
JAviator3D - University of Salzburg
JAviator3D
JAviator3D - University of Salzburg

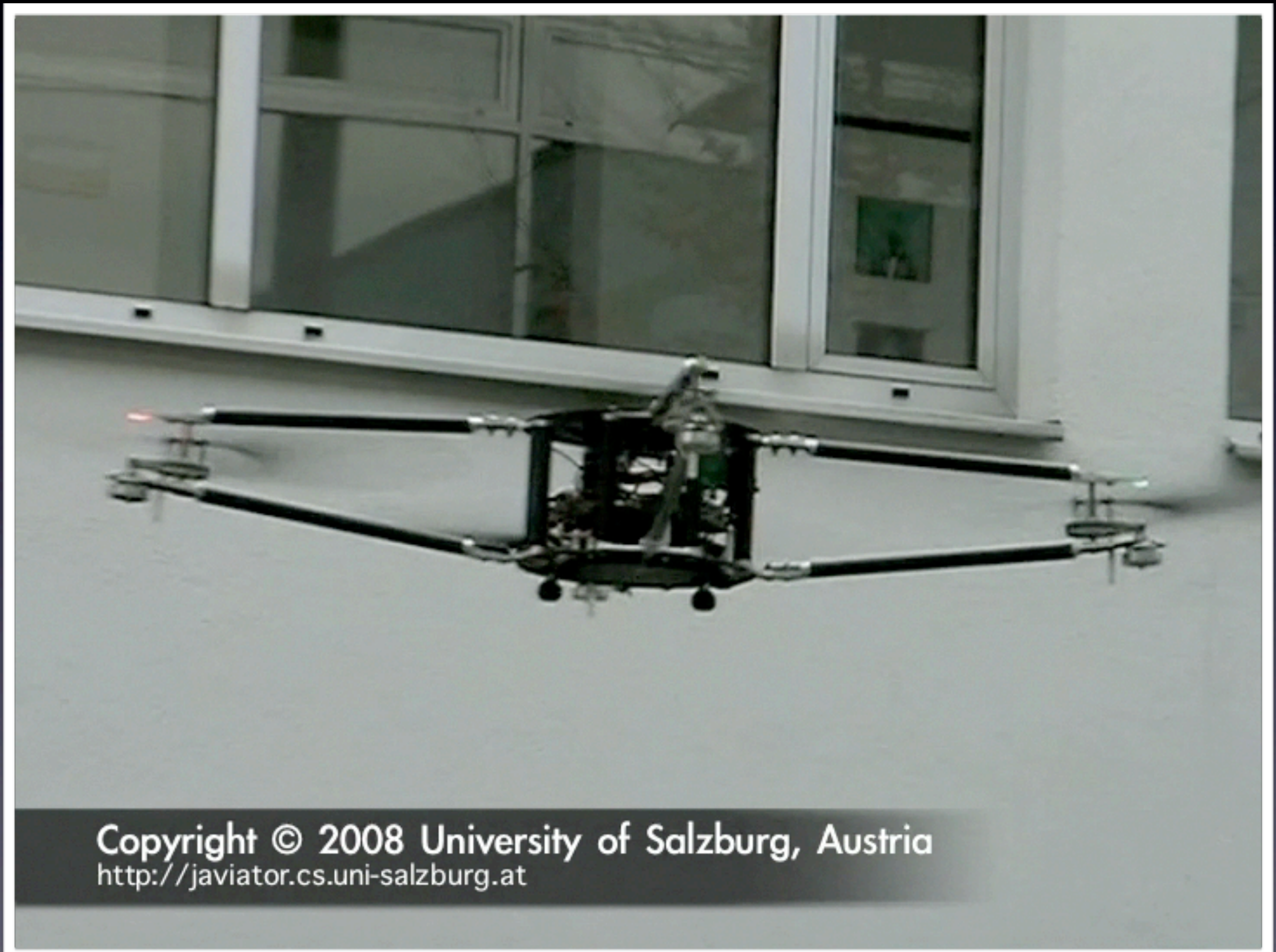
Indoor Flight
STARMAC Controller

Indoor Flight
STARMAC Controller



Outdoor Flight STARMAC Controller

Outdoor Flight
STARMAC Controller



Outdoor Flight
Salzburg Controller

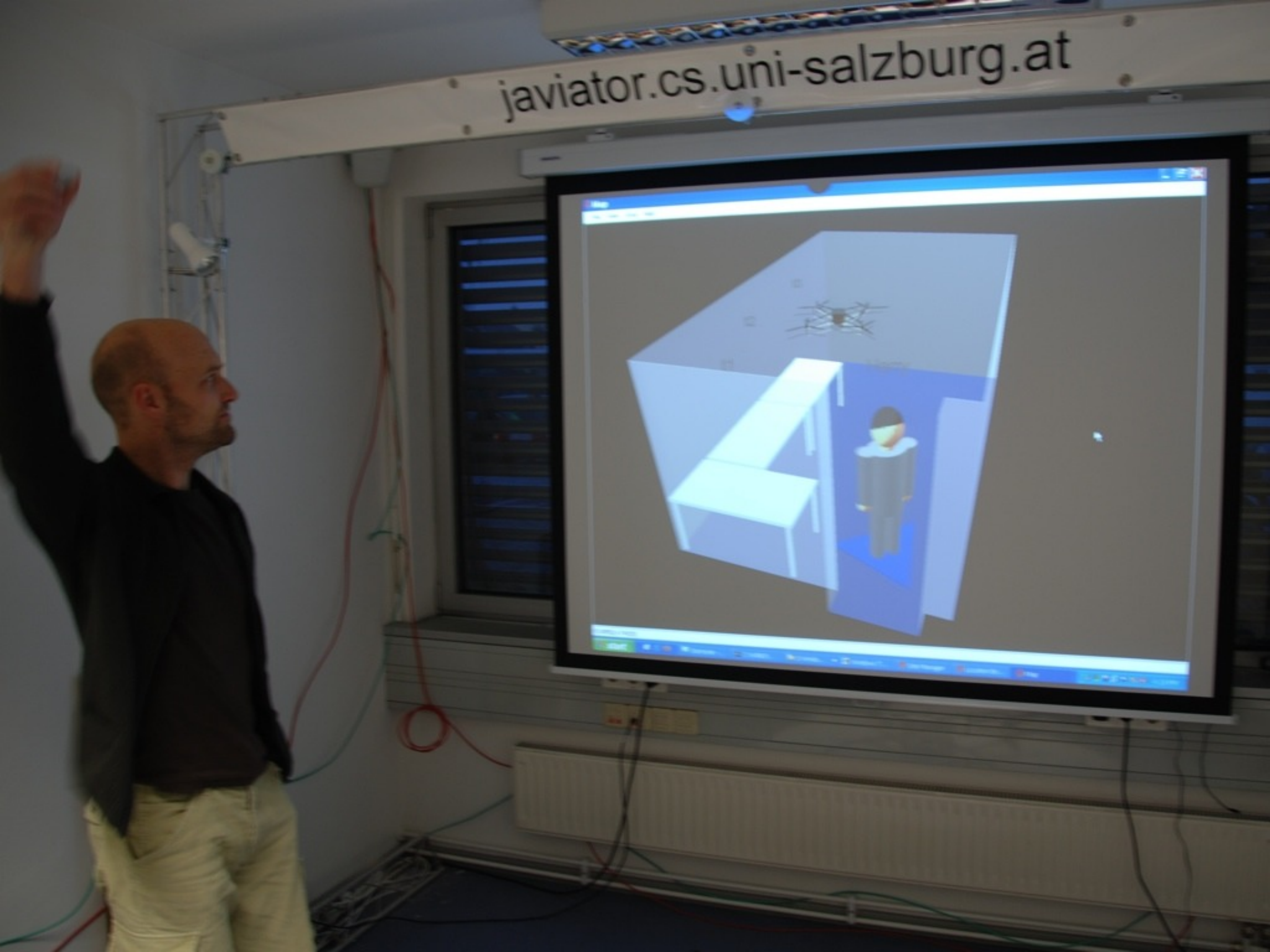
Outdoor Flight
Salzburg Controller

What's next?

- Autonomous single-vehicle flights
  - position controller
  - waypoint controller

What's next?

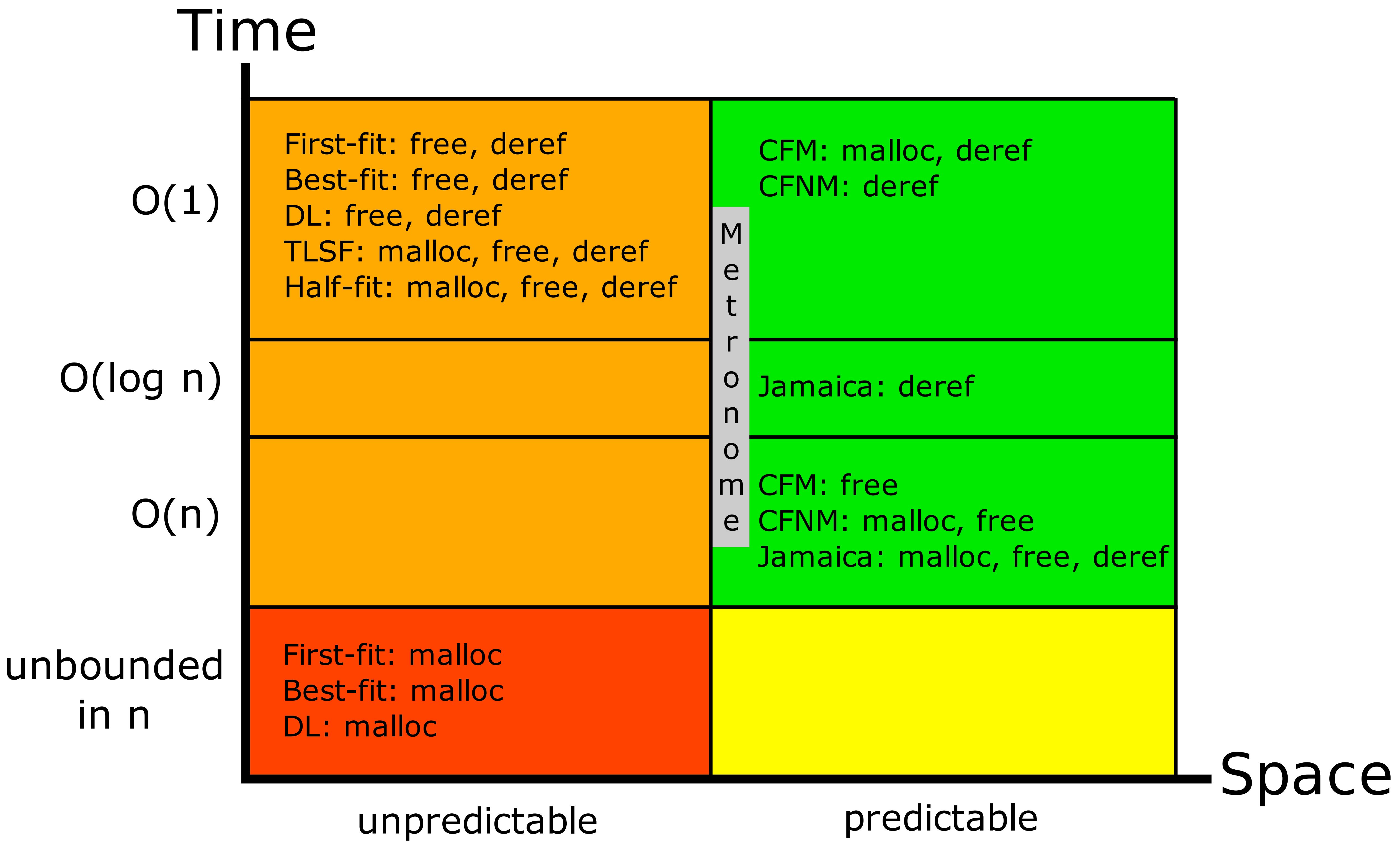
- Autonomous single-vehicle flights
  - position controller
  - waypoint controller
- Autonomous multi-vehicle flights
  - mission controller

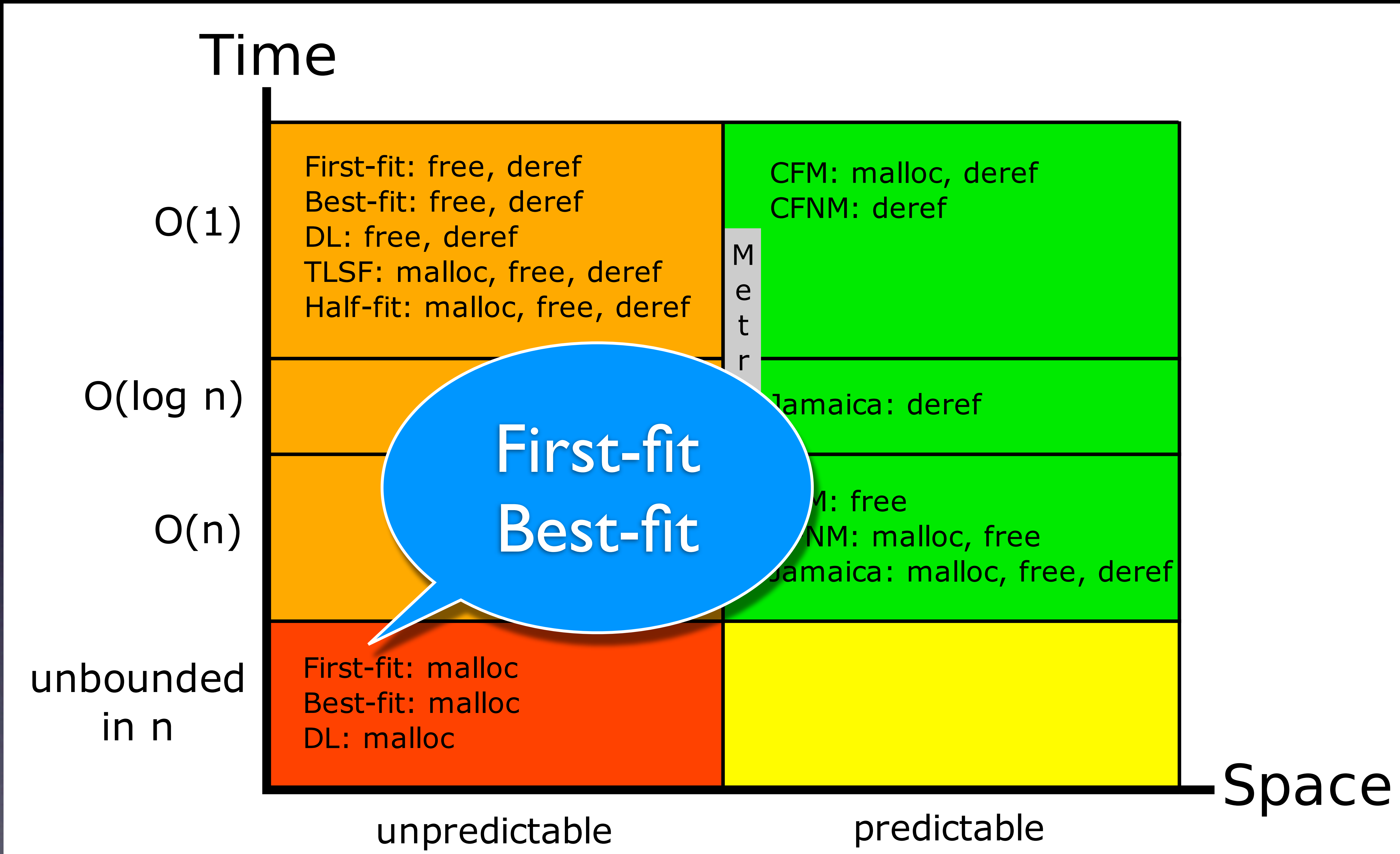

javiator.cs.uni-salzburg.at

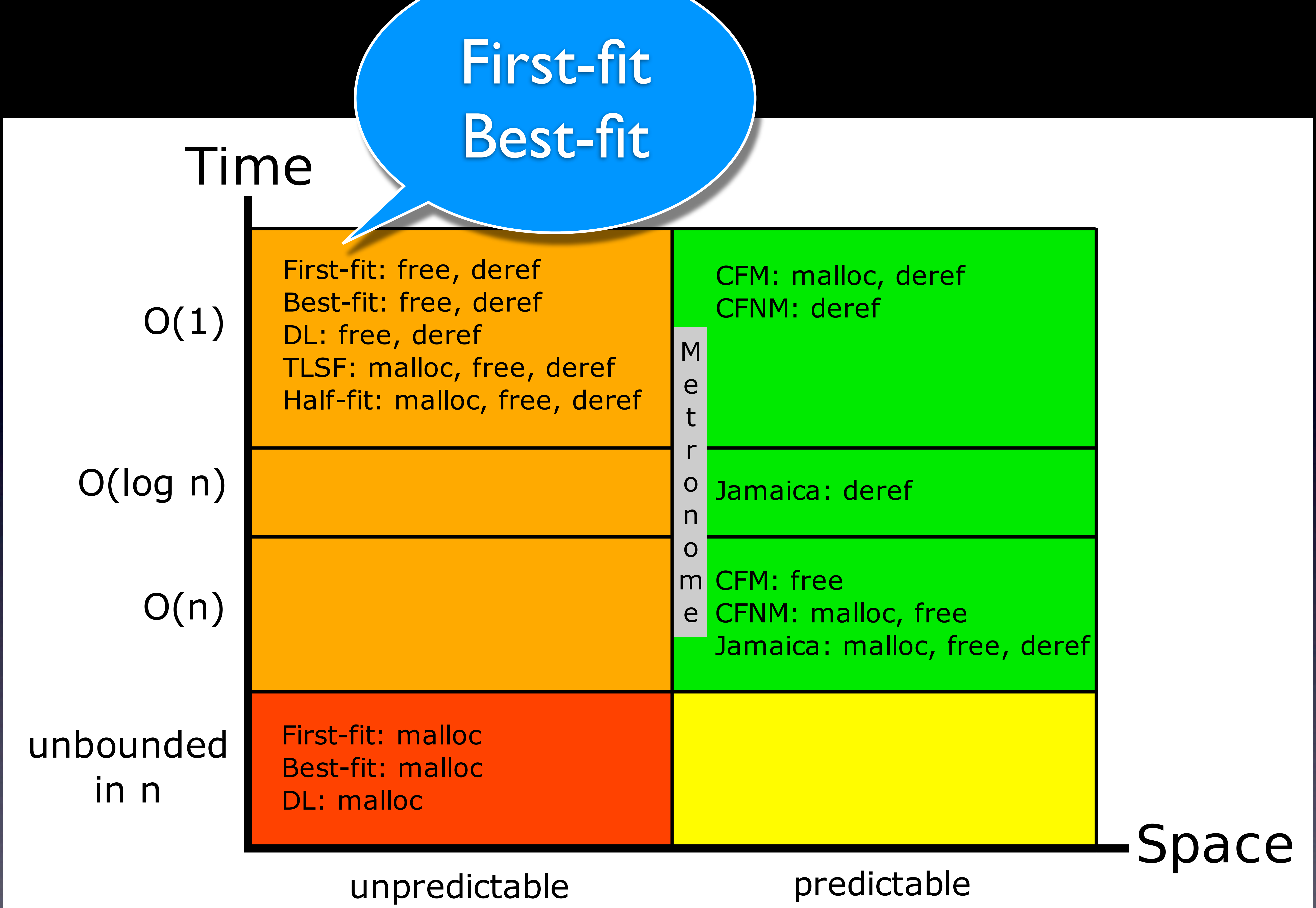
Salzburg Soft Walls Controller on JJ

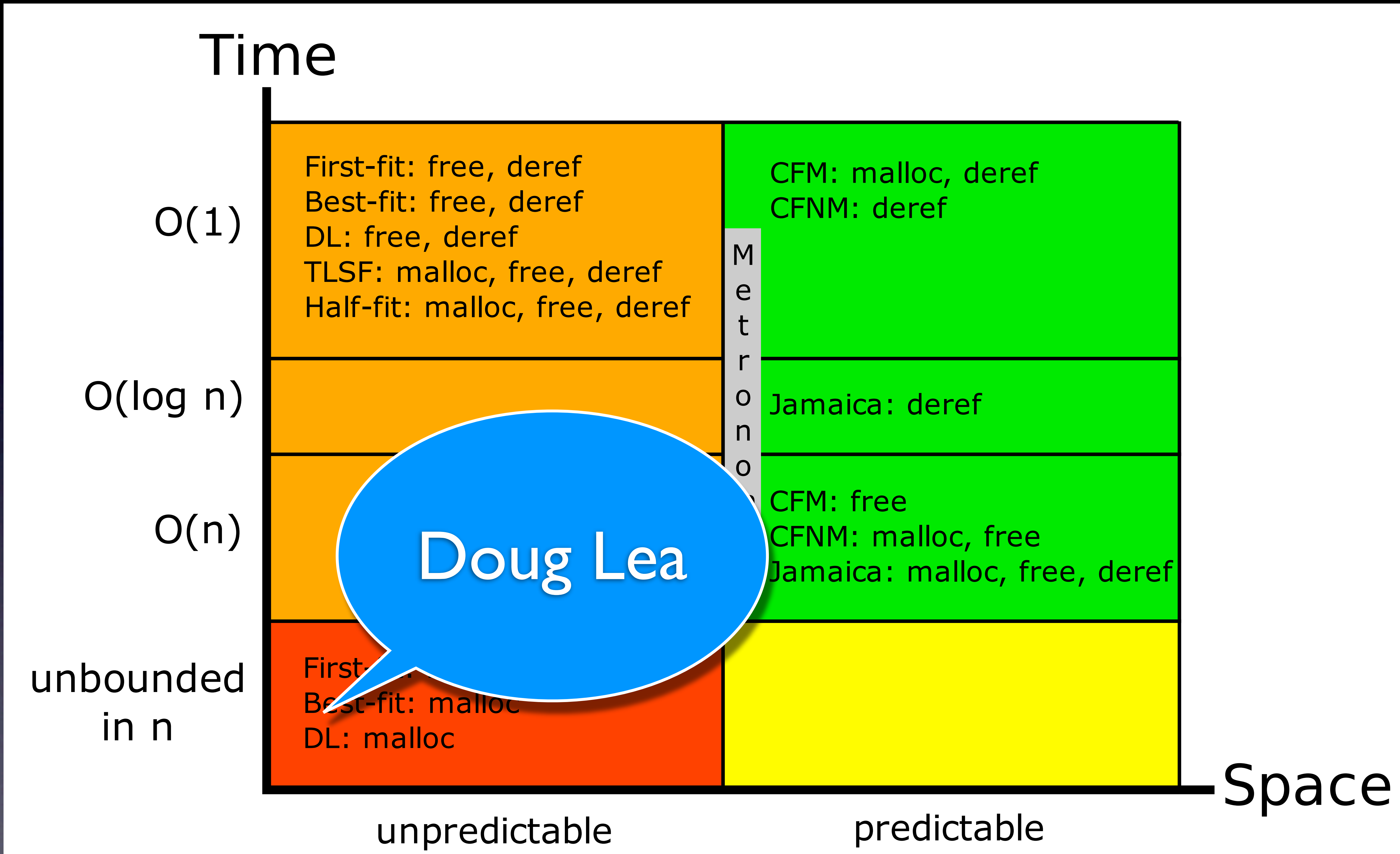
Salzburg Soft Walls Controller on JJ

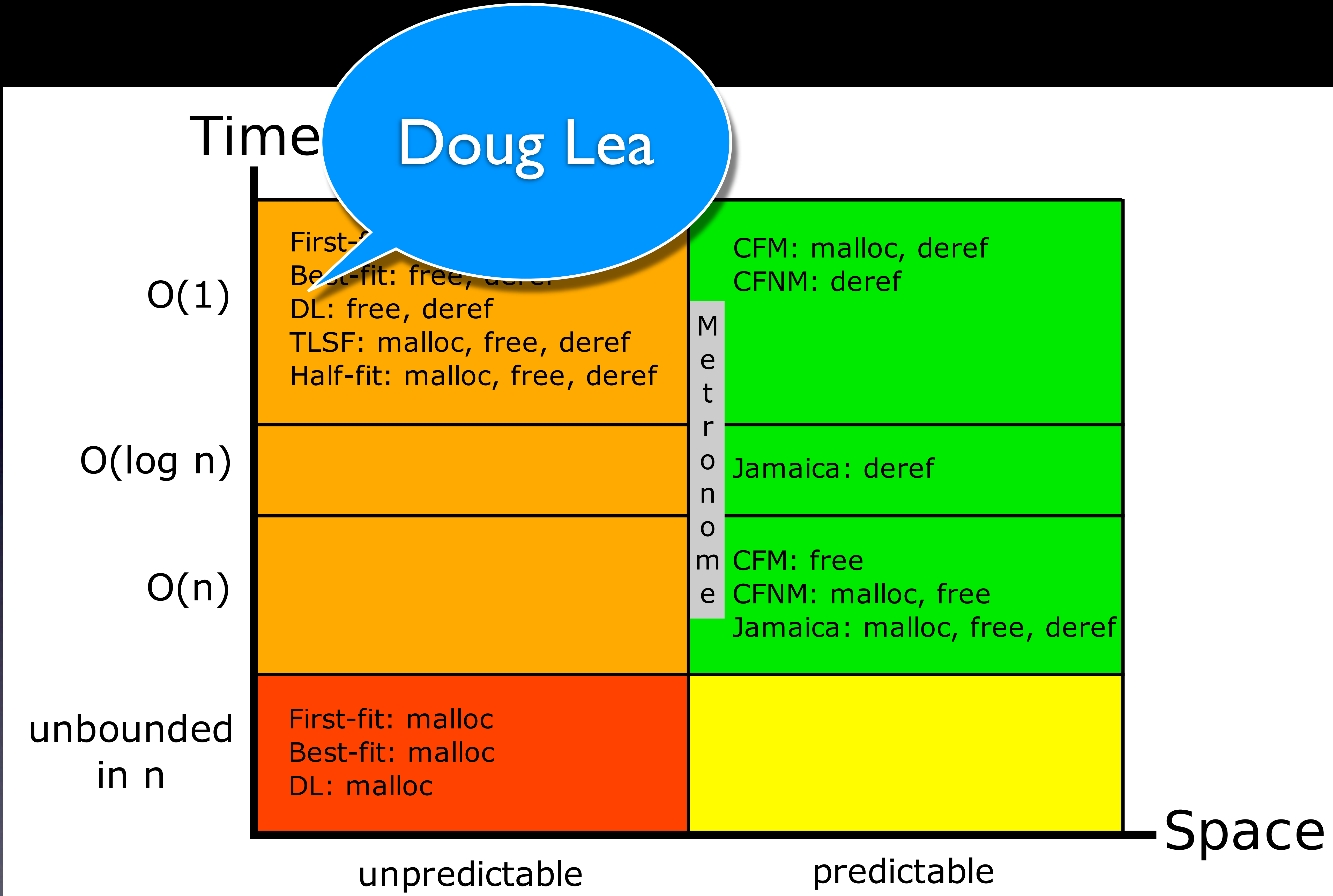
Memory Management Systems Overview

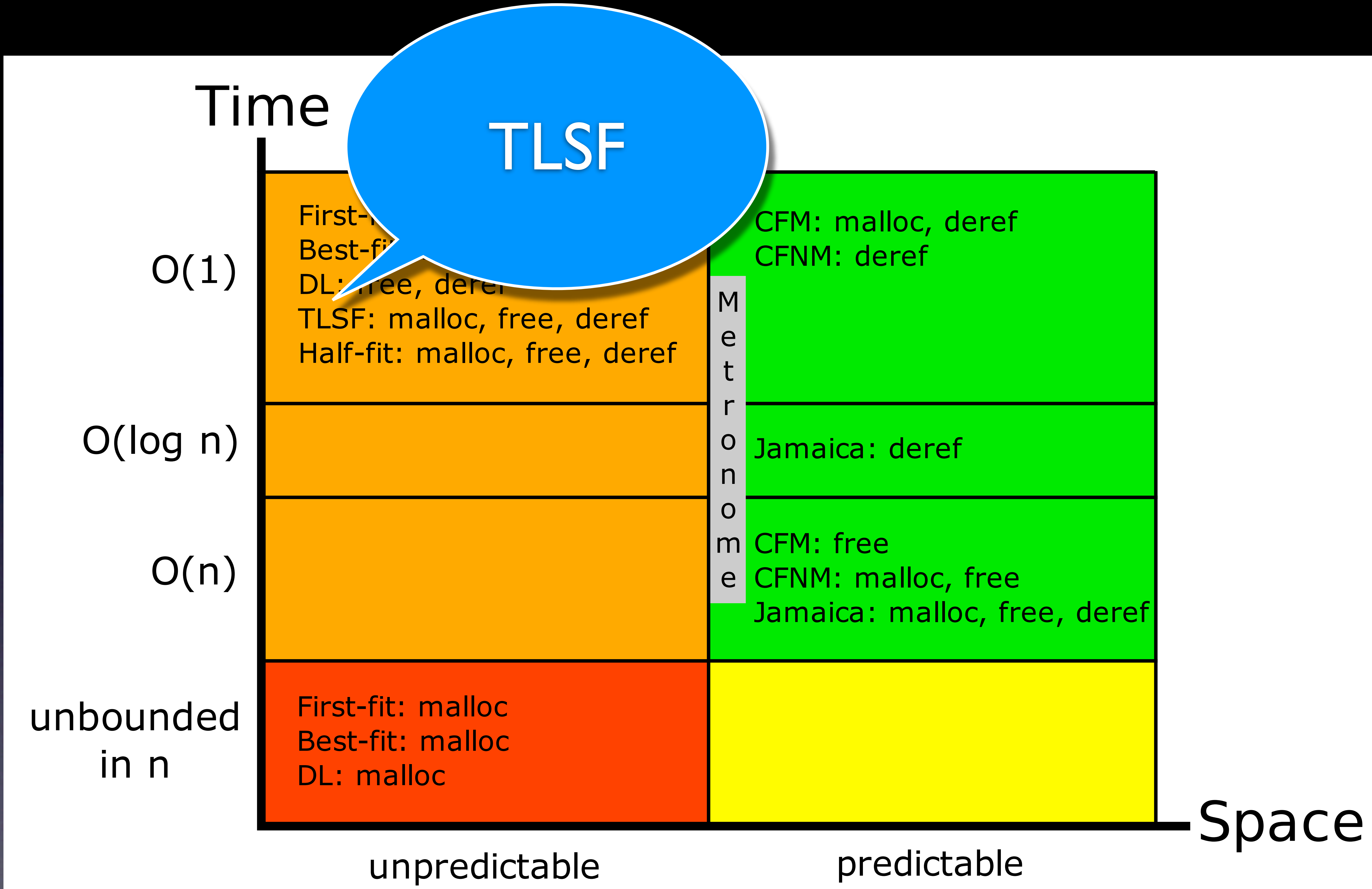

Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
unpredictable
predictable
Space

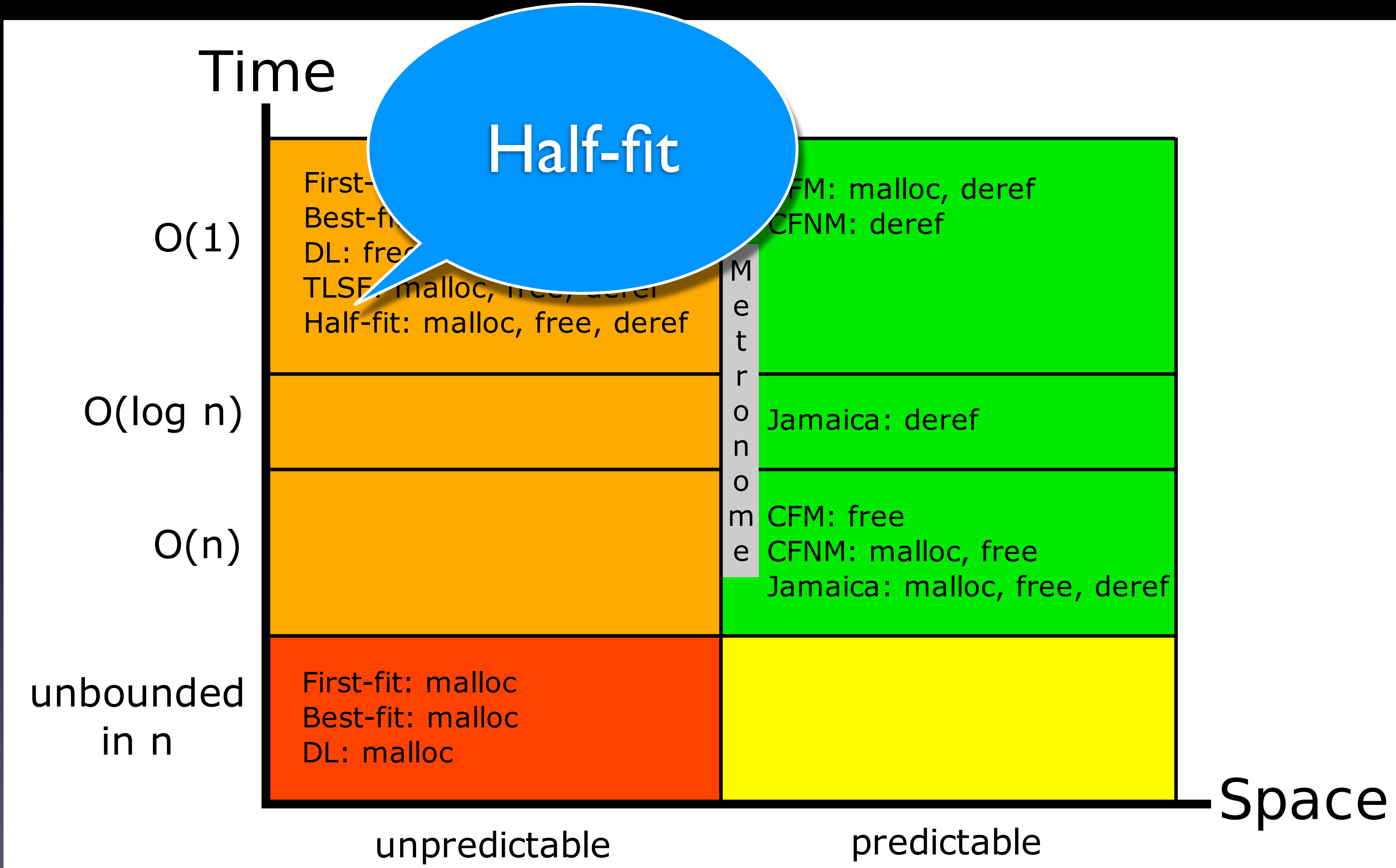

Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metr
amaica: deref
M: free
NM: malloc, free
amaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
First-fit
Best-fit
Space
unpredictable
predictable

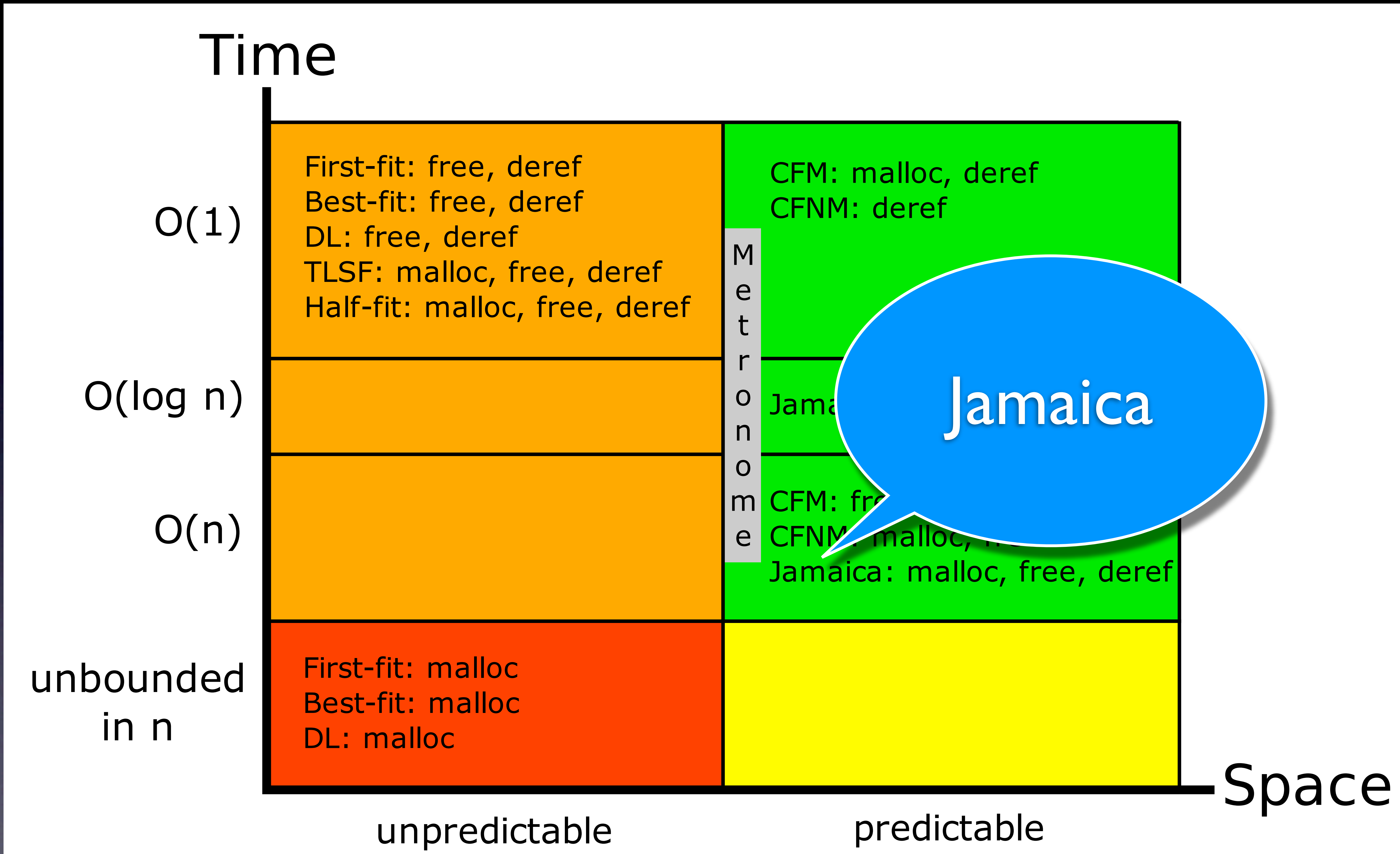

First-fit
Best-fit
Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
Metronome
CFM: malloc, deref
CFNM: deref
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

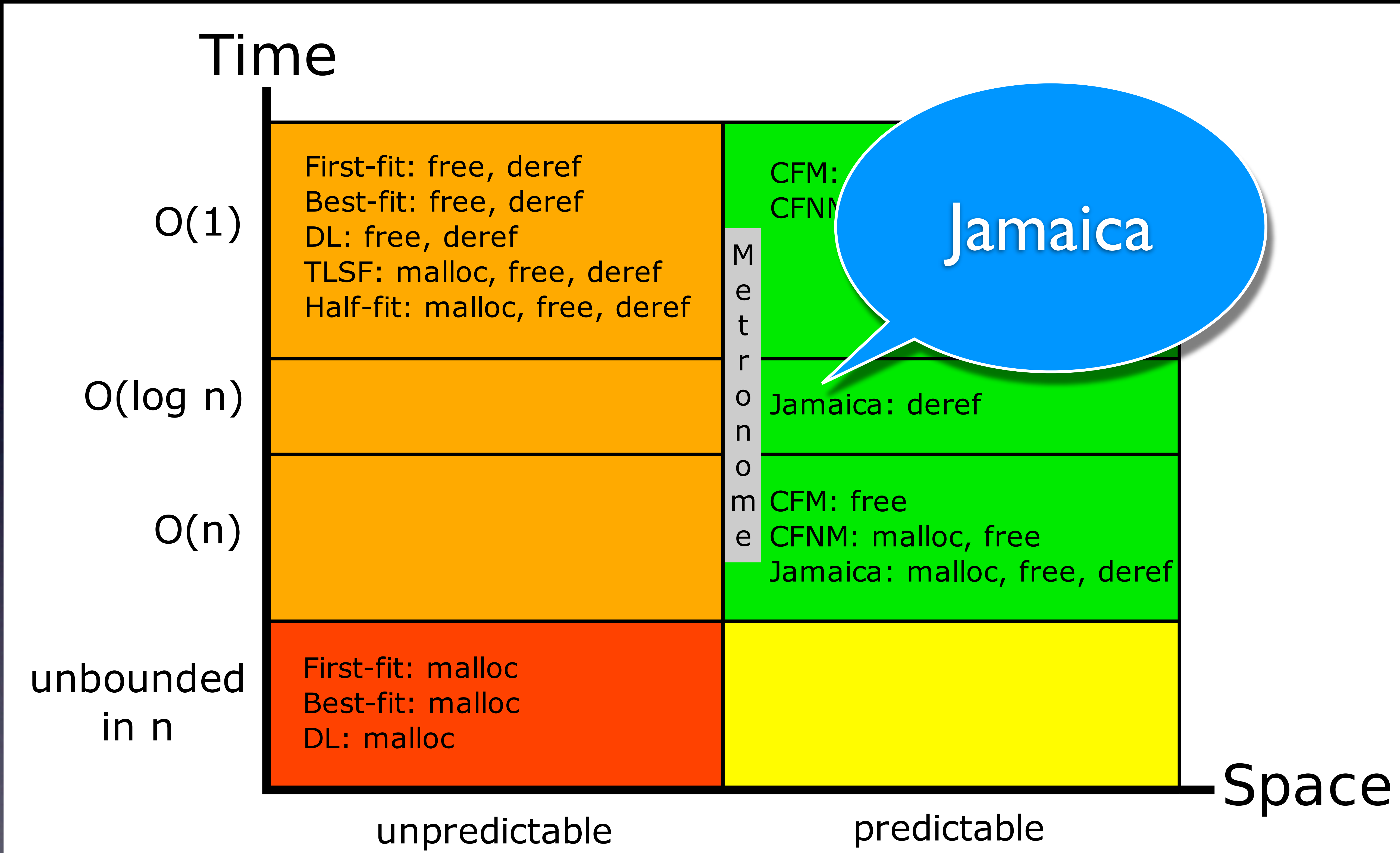

Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
DL: malloc
Doug Lea
unpredictable
predictable
Space

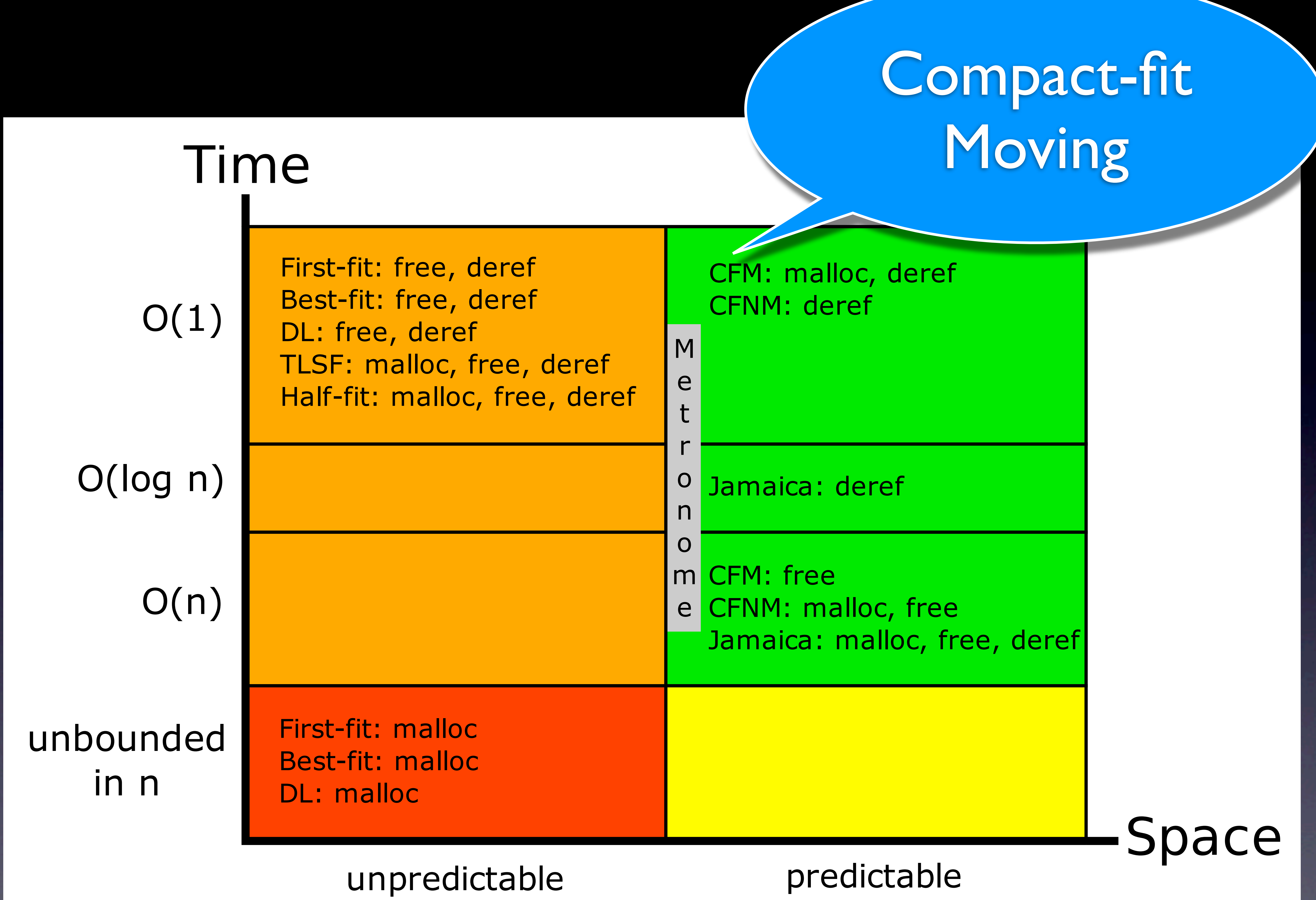

Doug Lea
Time
O(1)
O(log n)
O(n)
unbounded in n
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Metronome
CFM: malloc, deref
CFNM: deref
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
Space
unpredictable
predictable

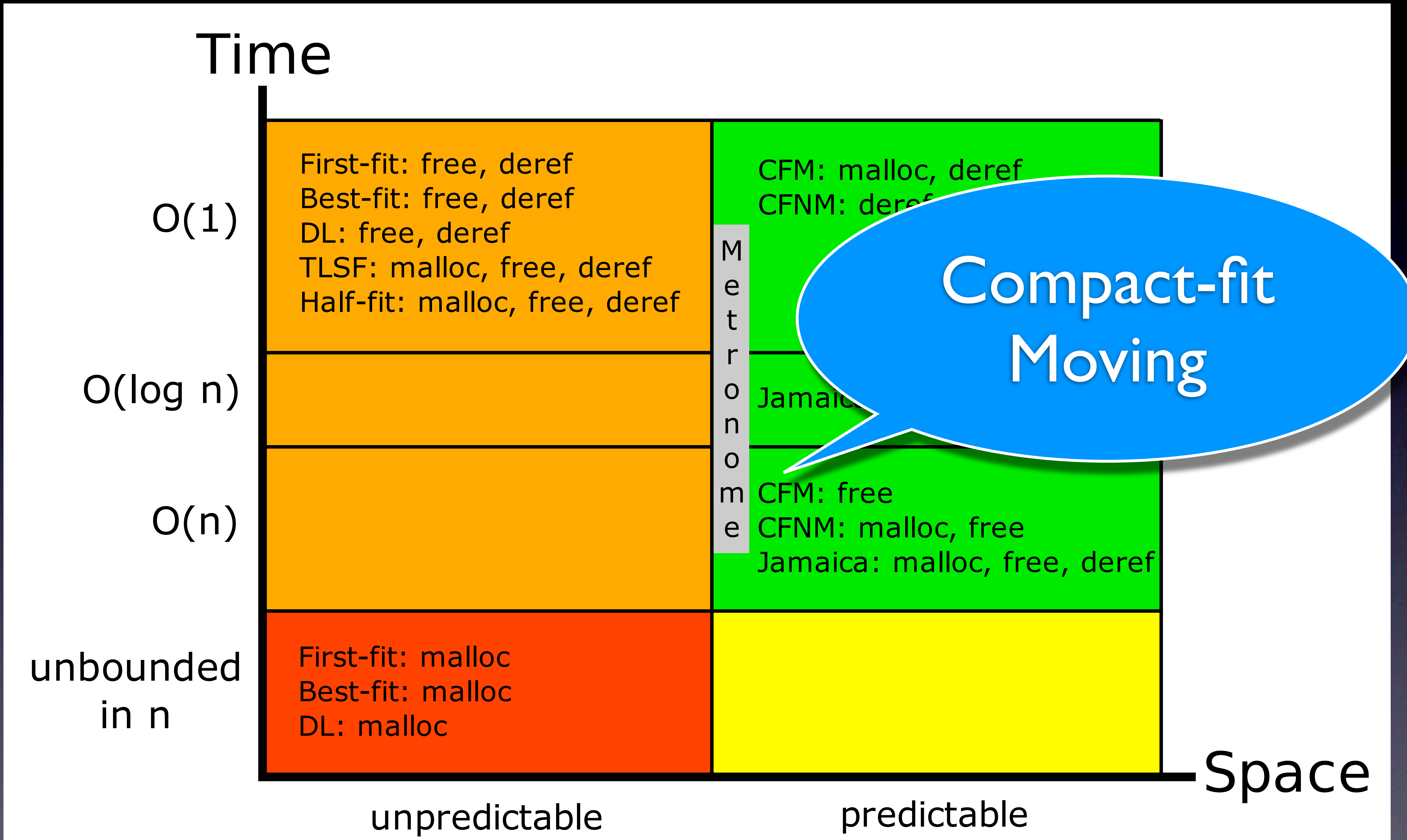

Time
TLSF
O(1)
O(log n)
O(n)
unbounded in n
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Metronome
CFM: malloc, deref
CFNM: deref
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
Space
unpredictable
predictable

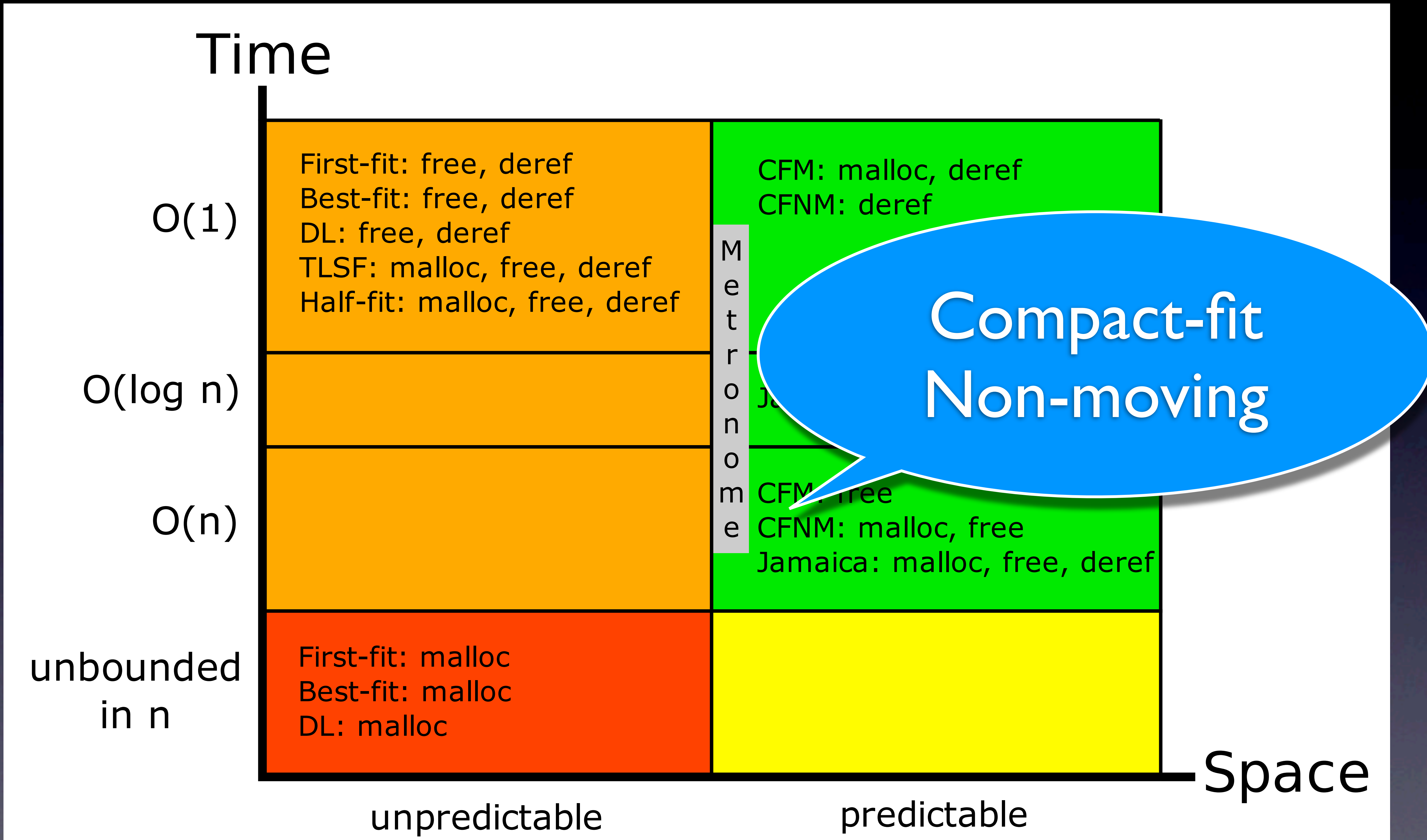

Time
Half-fit
First-
Best-fi
DL: fre
TLSF
Half-fit: malloc, free, deref
FM: malloc, deref
CFNM: deref
Metronome
O(1)
O(log n)
Jamaica: deref
O(n)
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
unbounded in n
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

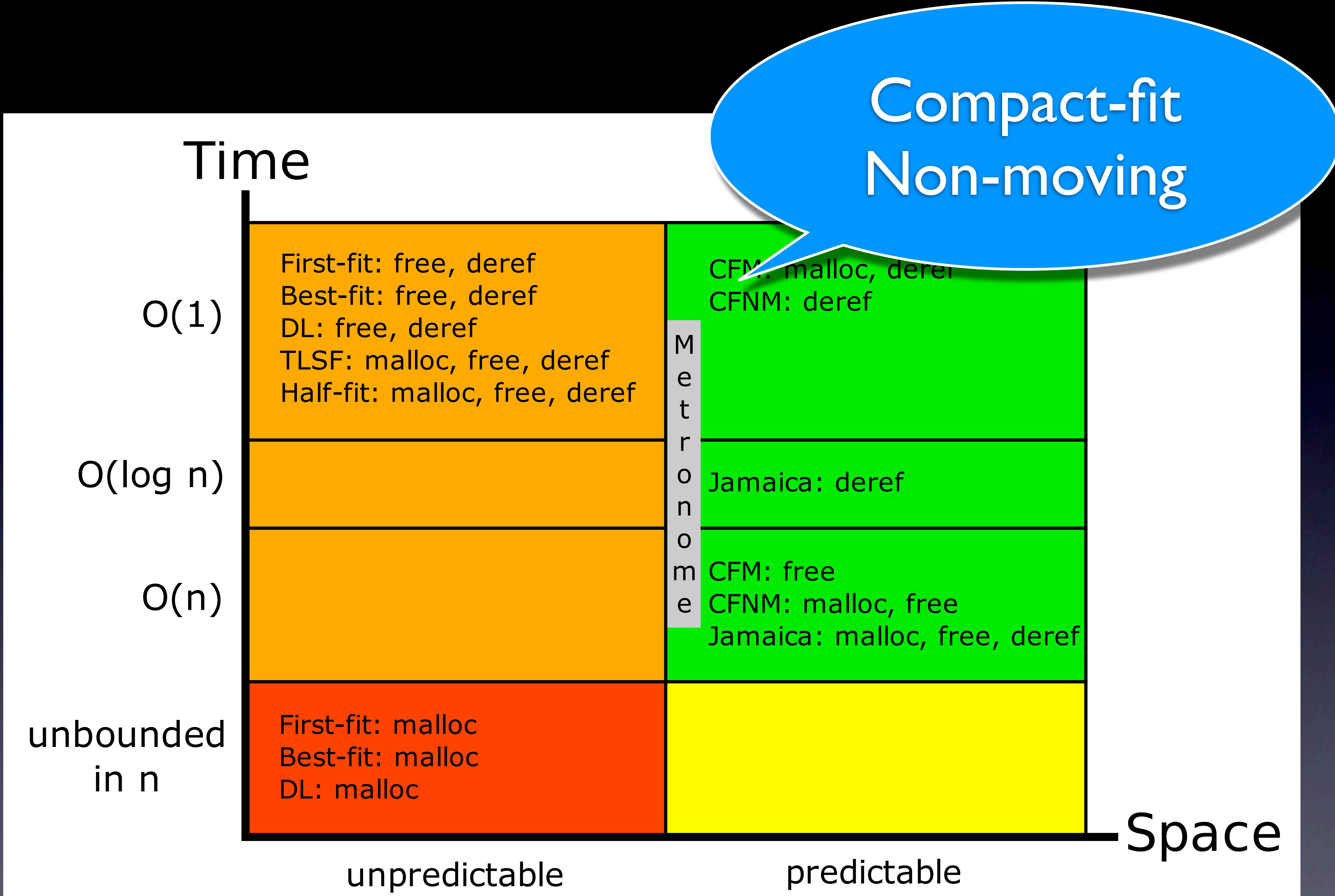

Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
Jama
CFM: fr
CFNM: malloc,
Jamaica: malloc, free, deref
Jamaica
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

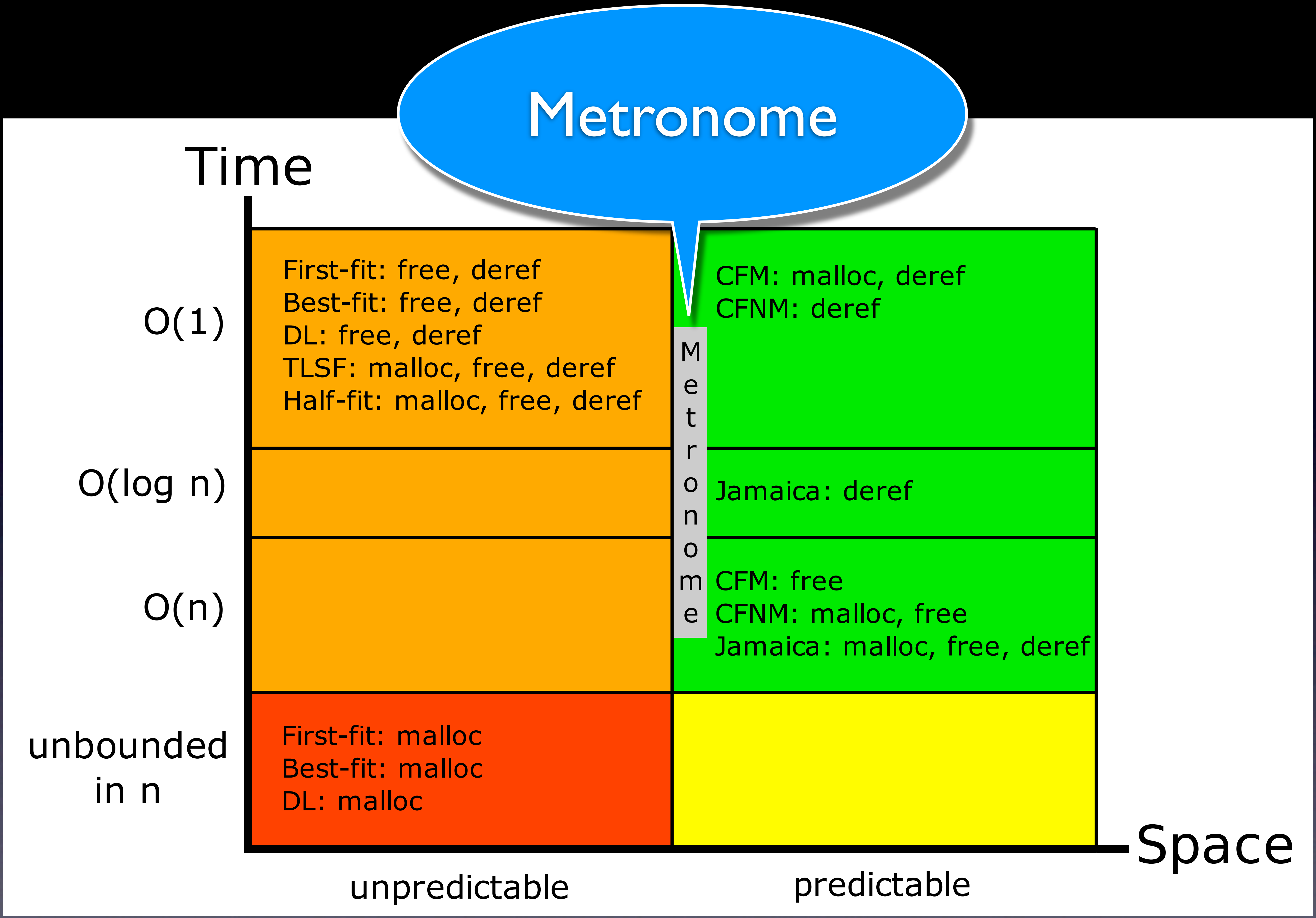

Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM:
CFN
Metronome
Jamaica
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
unpredictable
predictable
Space


Compact-fit
Moving
Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

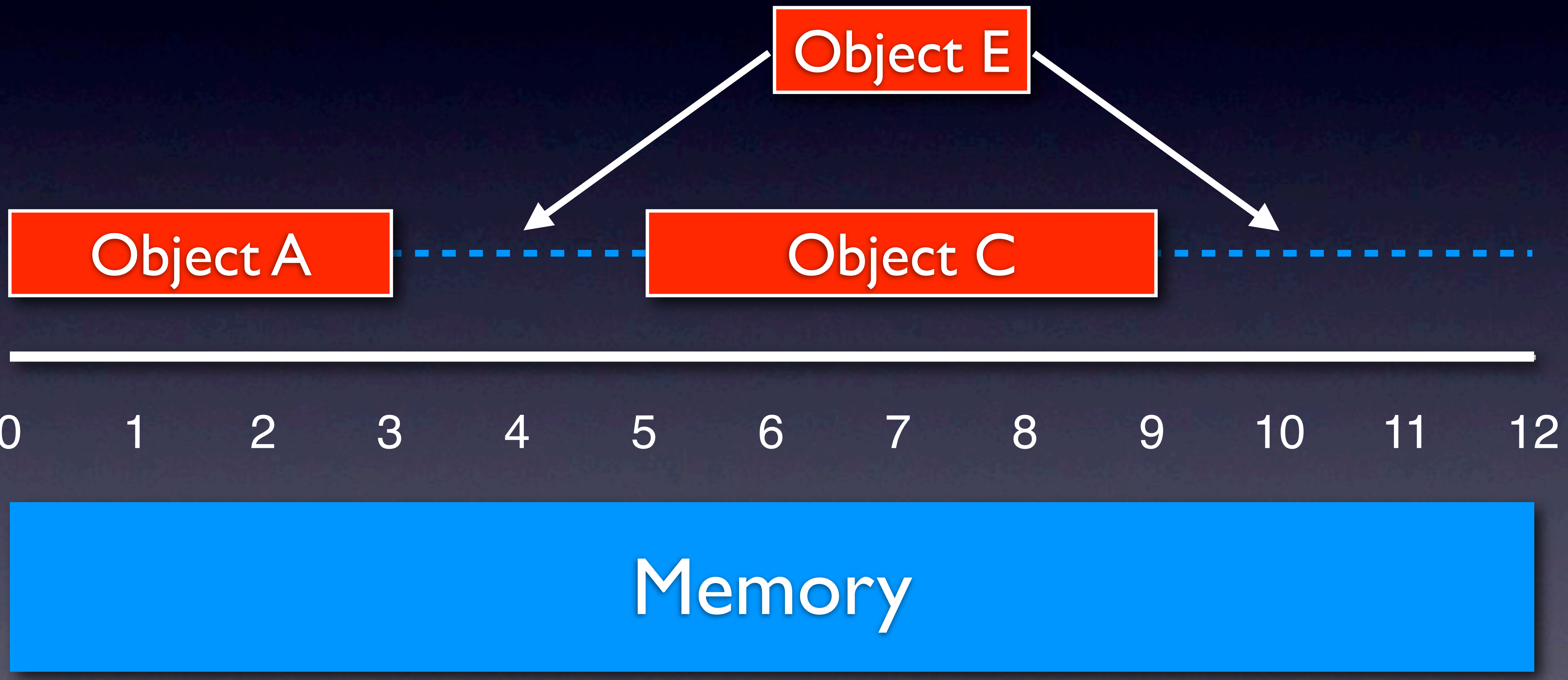

Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
Metronome
CFM: malloc, deref
CFNM: deref
Compact-fit
Moving
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable


Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Compact-fit
Non-moving
Space
unpredictable
predictable

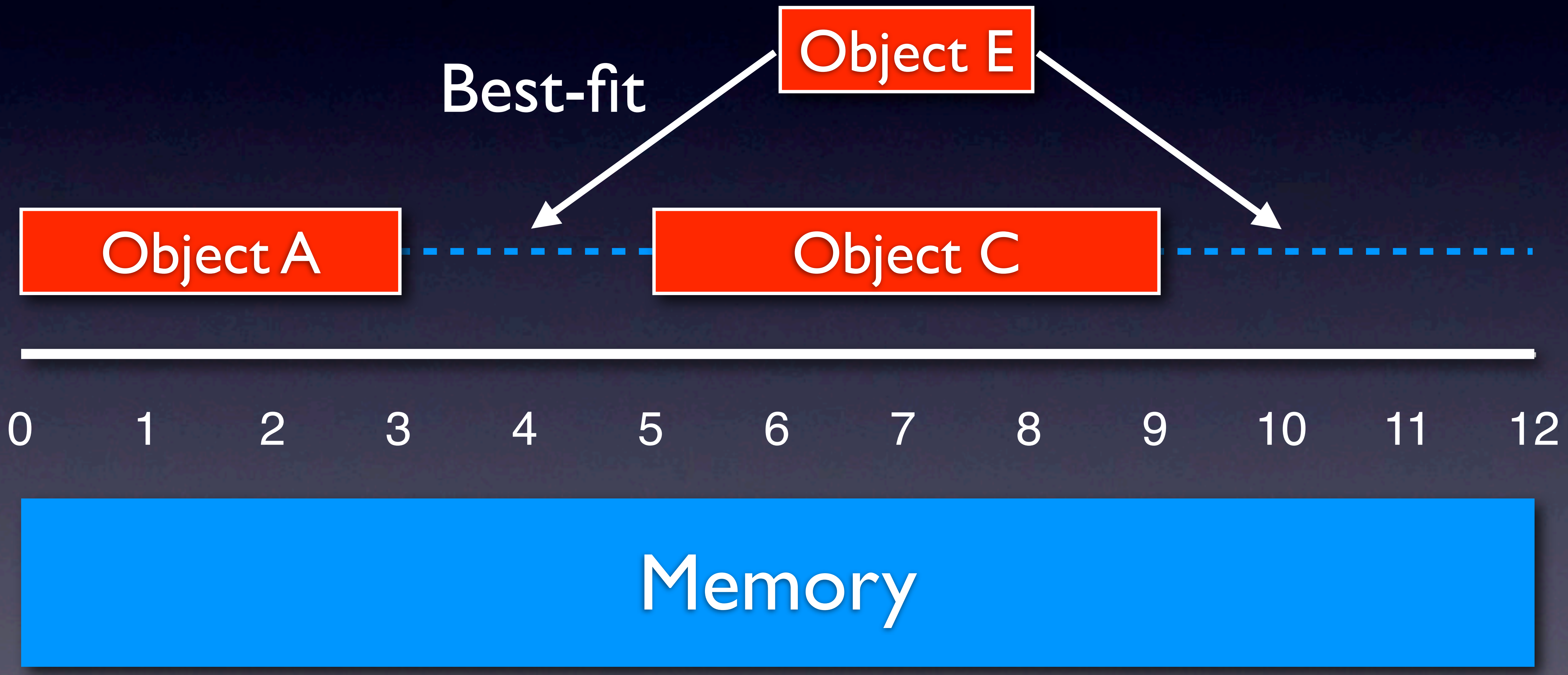

Compact-fit
Non-moving
Time
O(1)
O(log n)
O(n)
unbounded
in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFNM: deref
Metronome
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable


Metronome
Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

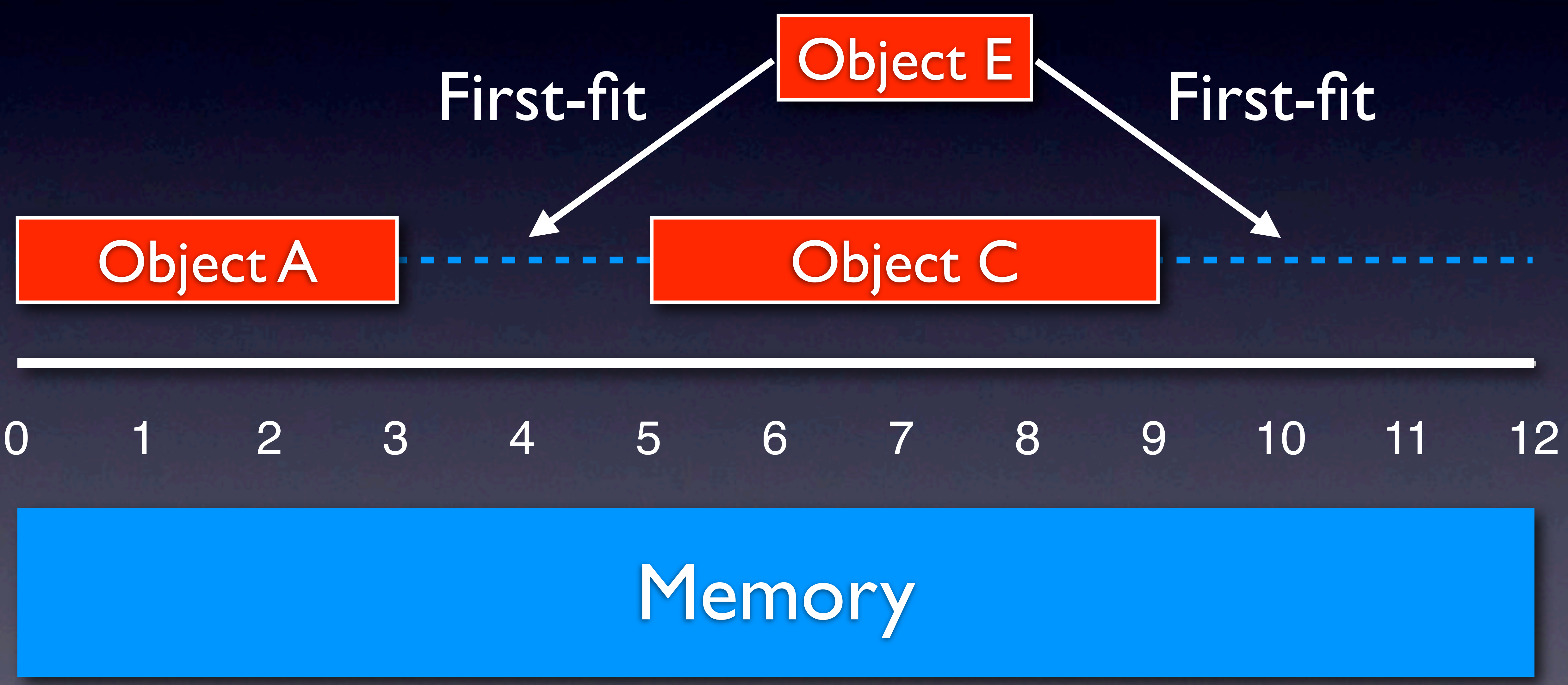
Best-fit versus First-fit



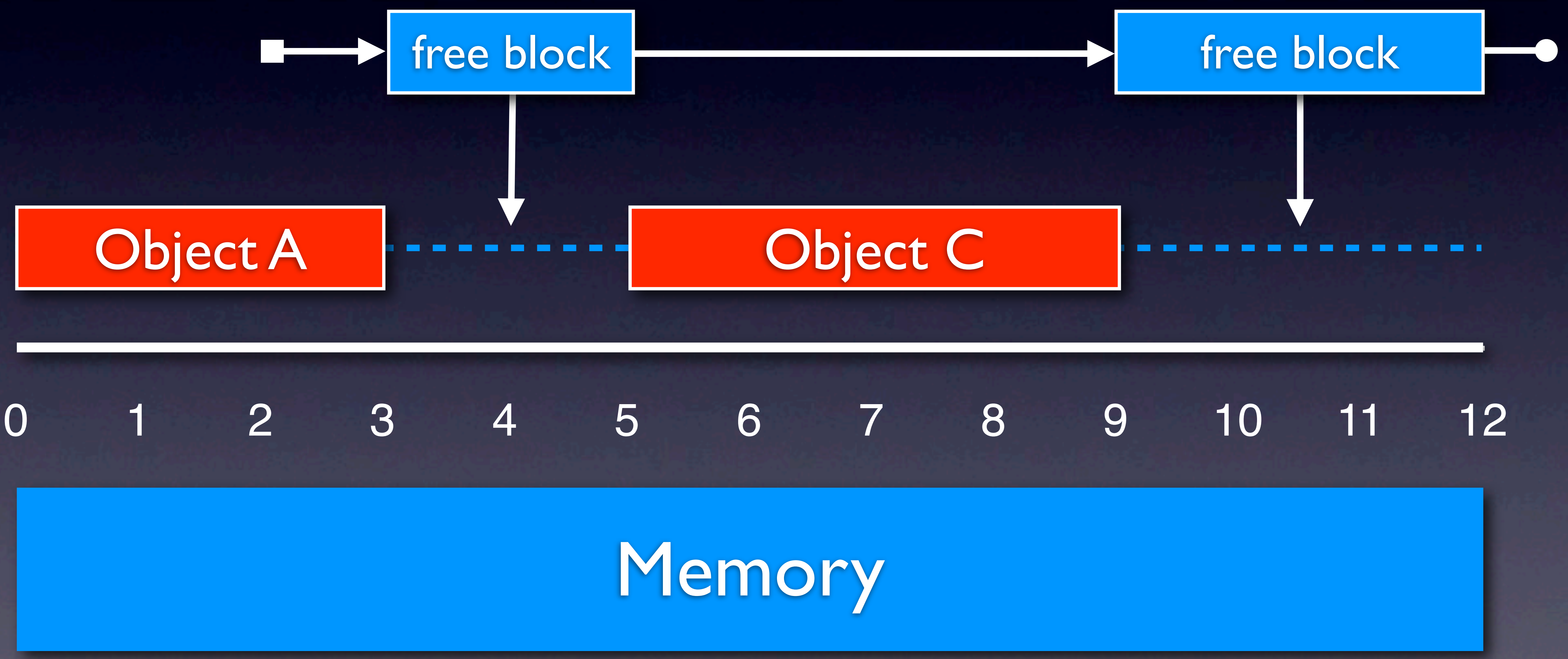
Best-fit versus First-fit



Best-fit versus First-fit



Free List


free block
free block
Object A
Object C
0
1
2
3
4
5
6
7
8
9
10
11
12
Memory

Best-fit, First-fit Complexity

- Allocation:
  - `malloc` may take time proportional to heap size

Best-fit, First-fit Complexity

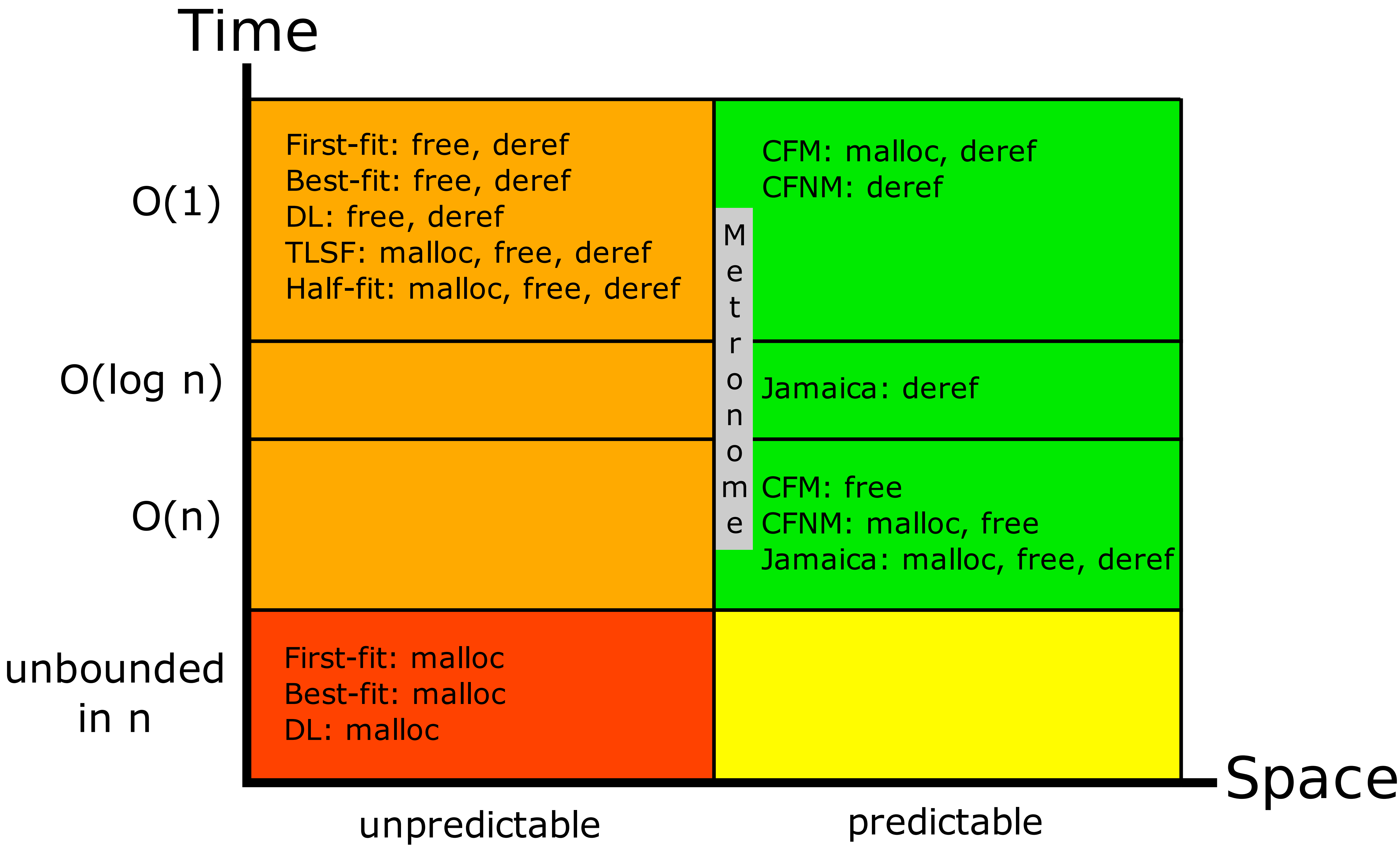
- Allocation:
  - `malloc` may take time proportional to heap size
- Deallocation:
  - `free` takes constant time

Best-fit, First-fit Complexity

- Allocation:
  - `malloc` may take time proportional to heap size
- Deallocation:
  - `free` takes constant time
- Access:
  - `read` and `write` take constant time

Best-fit, First-fit Complexity

- Allocation:
  - `malloc` may take time proportional to heap size
- Deallocation:
  - `free` takes constant time
- Access:
  - `read` and `write` take constant time
- Unpredictable fragmentation


Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
Metronome
CFM: malloc, deref
CFNM: deref
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

Free List Operations

- Select:
  - `malloc`

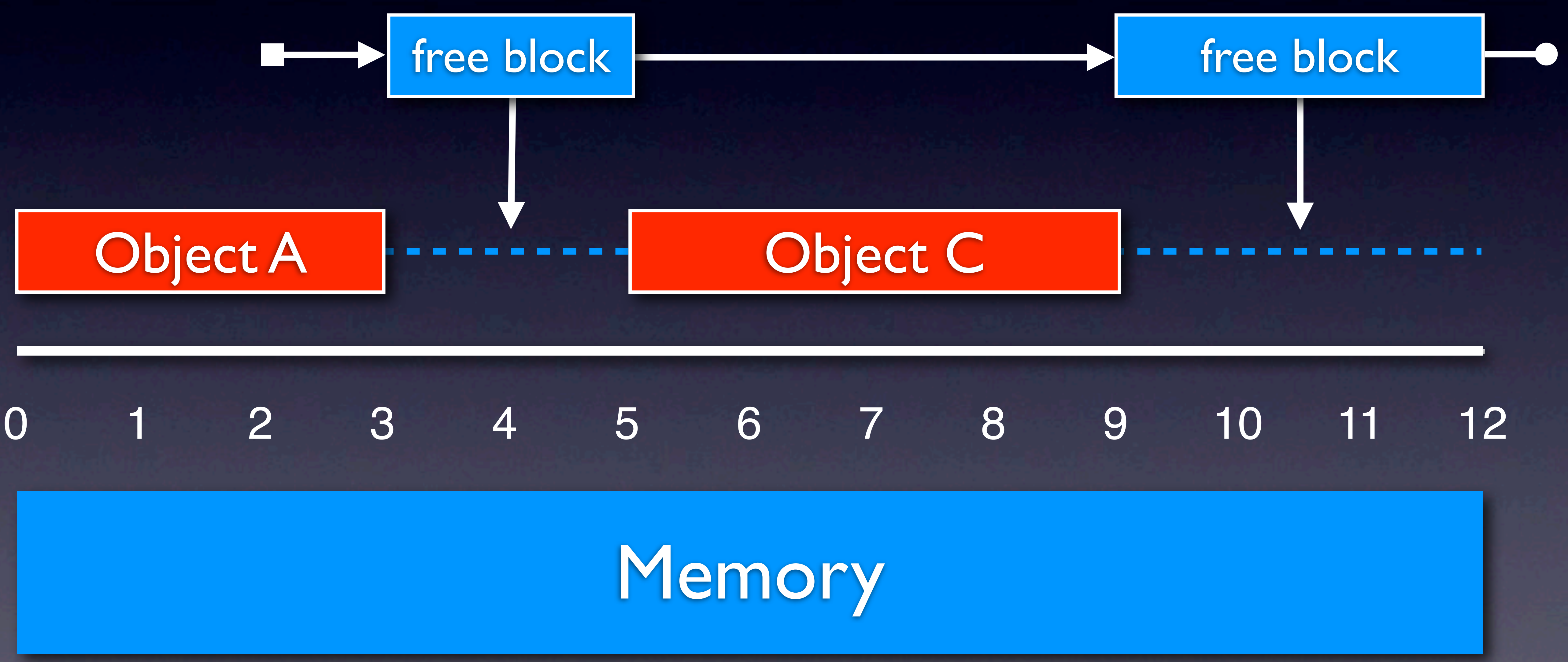
Free List Operations

- Select:
  - `malloc`
- Insert:
  - `free`

Free List Operations

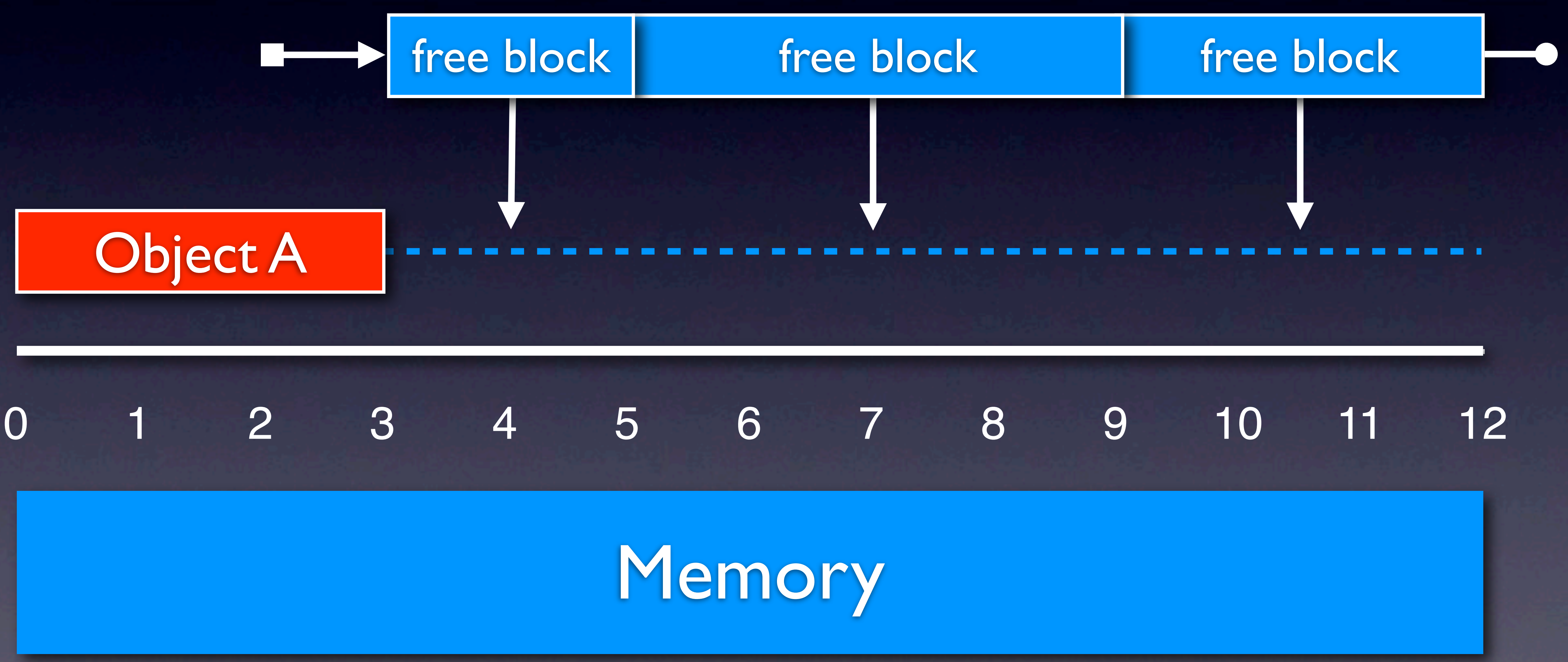
- Select:
  - `malloc`
- Insert:
  - `free`
- Delete:
  - coalescing

Coalescing

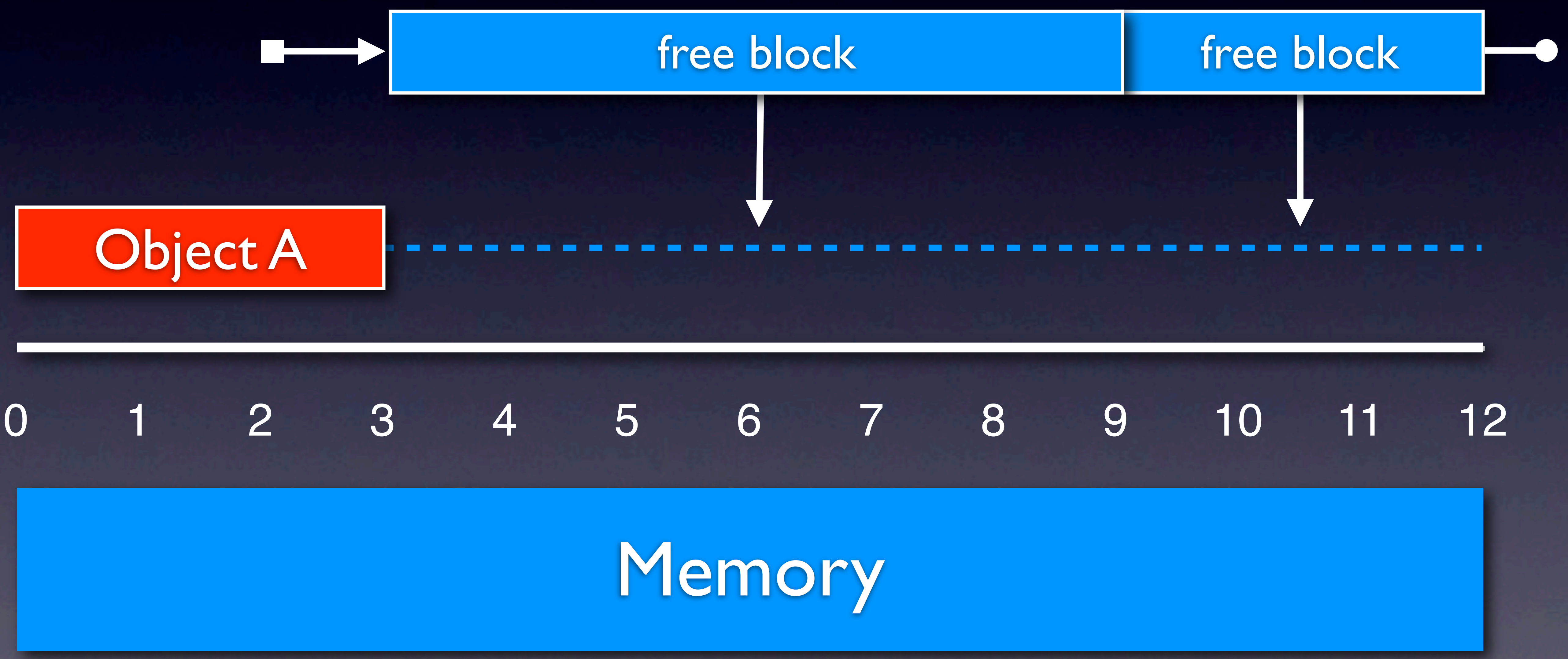


free block
free block
Object A
Object C
0
1
2
3
4
5
6
7
8
9
10
11
12
Memory

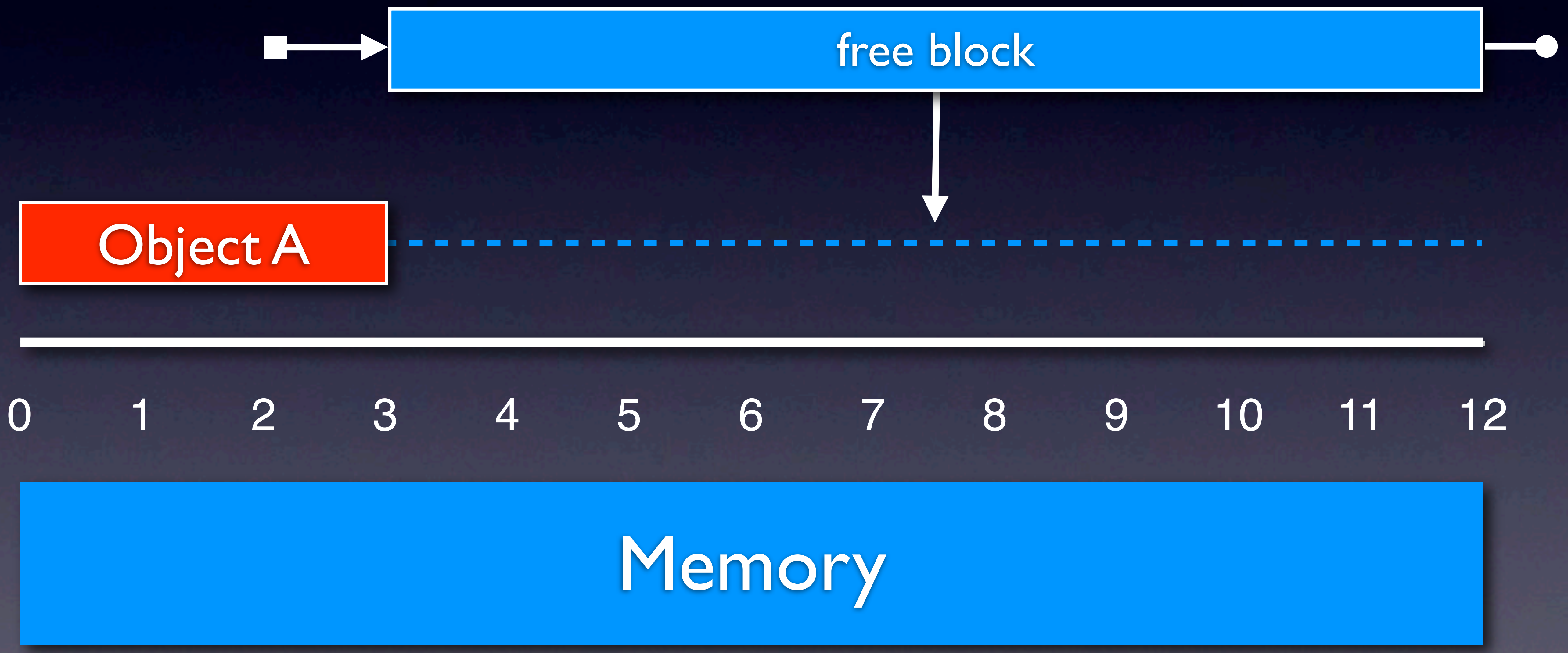
Coalescing


free block
free block
free block
Object A
0
1
2
3
4
5
6
7
8
9
10
11
12
Memory

Coalescing



Coalescing



List Representations

- List: singly-linked or doubly-linked (using boundary tags)

List Representations

- List: singly-linked or doubly-linked (using boundary tags)
- Segregated lists: array of lists for different sizes

List Representations

- List: singly-linked or doubly-linked (using boundary tags)
- Segregated lists: array of lists for different sizes
- Buddy systems: split blocks in powers of two (called buddies if same size)

List Representations

- List: singly-linked or doubly-linked (using boundary tags)
- Segregated lists: array of lists for different sizes
- Buddy systems: split blocks in powers of two (called buddies if same size)
- Indexed lists: trees, bitmaps

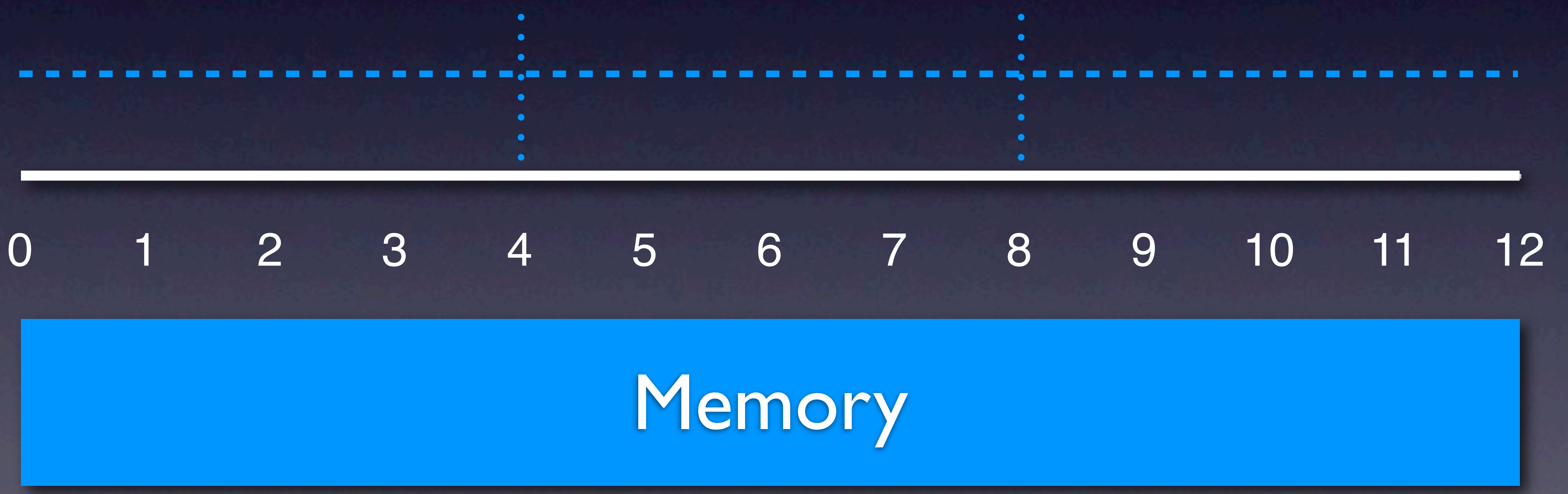
List Representations

- List: singly-linked or doubly-linked (using boundary tags)
- Segregated lists: array of lists for different sizes
- Buddy systems: split blocks in powers of two (called buddies if same size)
- Indexed lists: trees, bitmaps
- Hybrid: Doug Lea's allocator

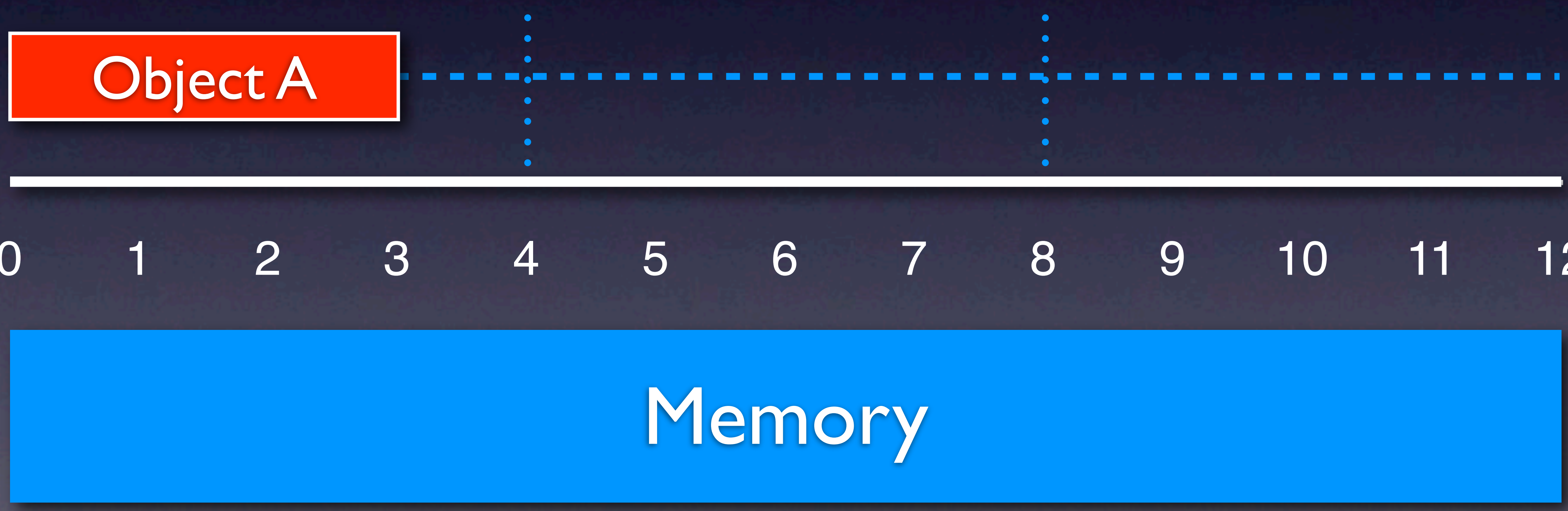
DL Complexity

- Allocation:
  - `malloc` may take time proportional to heap size
- Deallocation:
  - `free` takes constant time
- Access:
  - `read` and `write` take constant time
- Unpredictable fragmentation

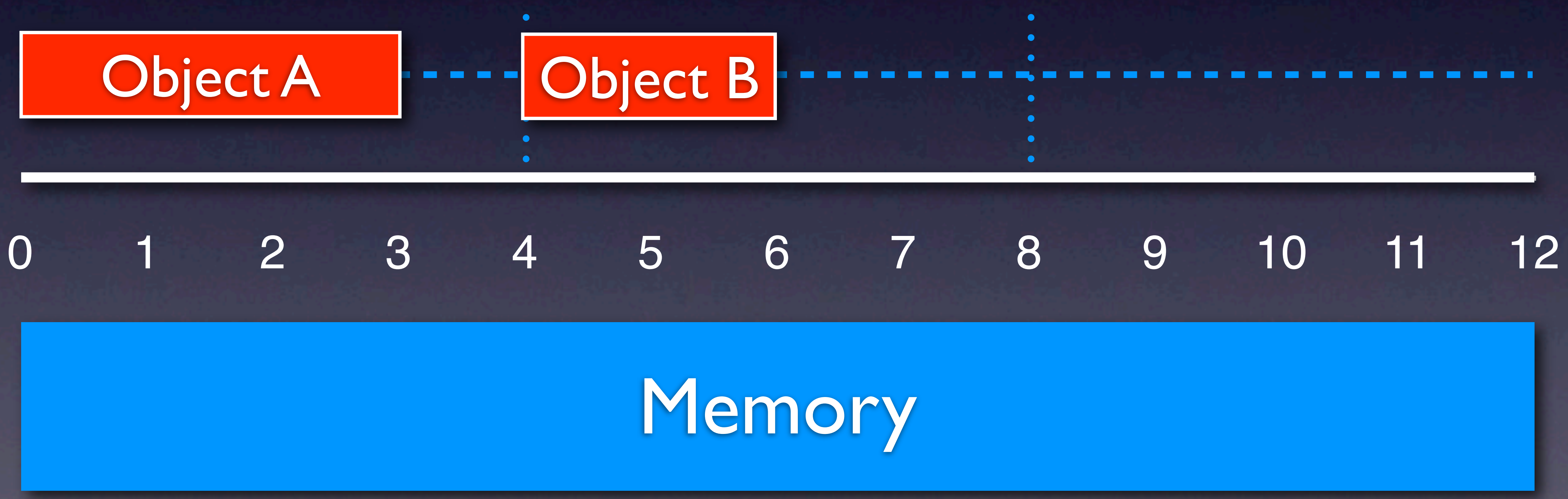
Partitioning



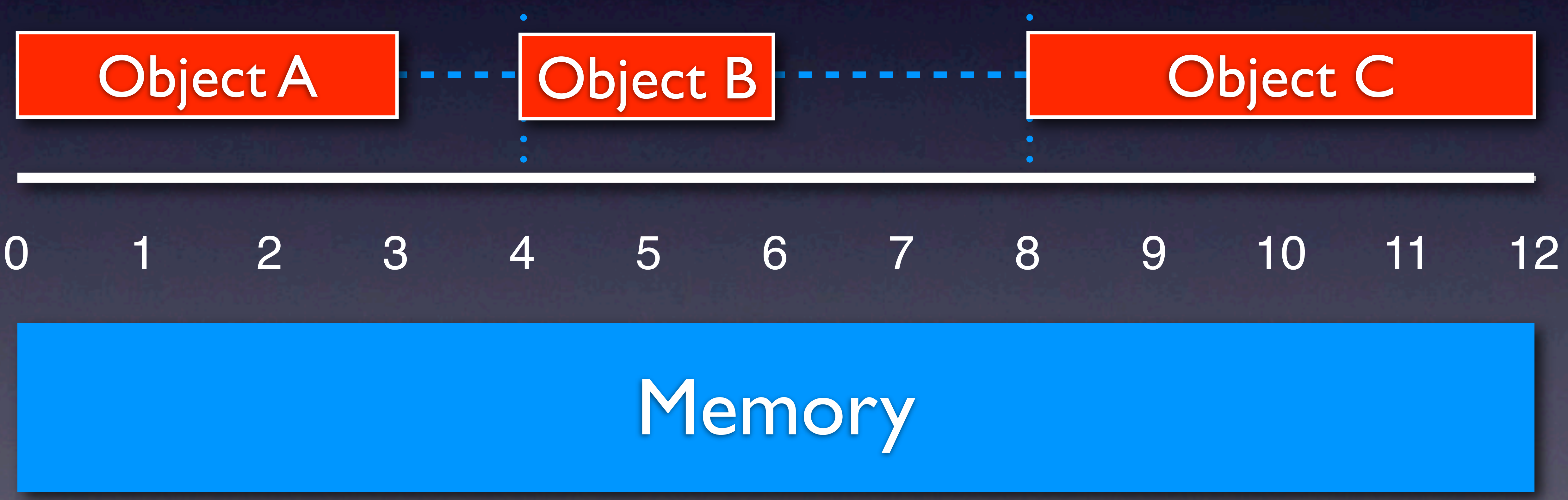
Partitioning



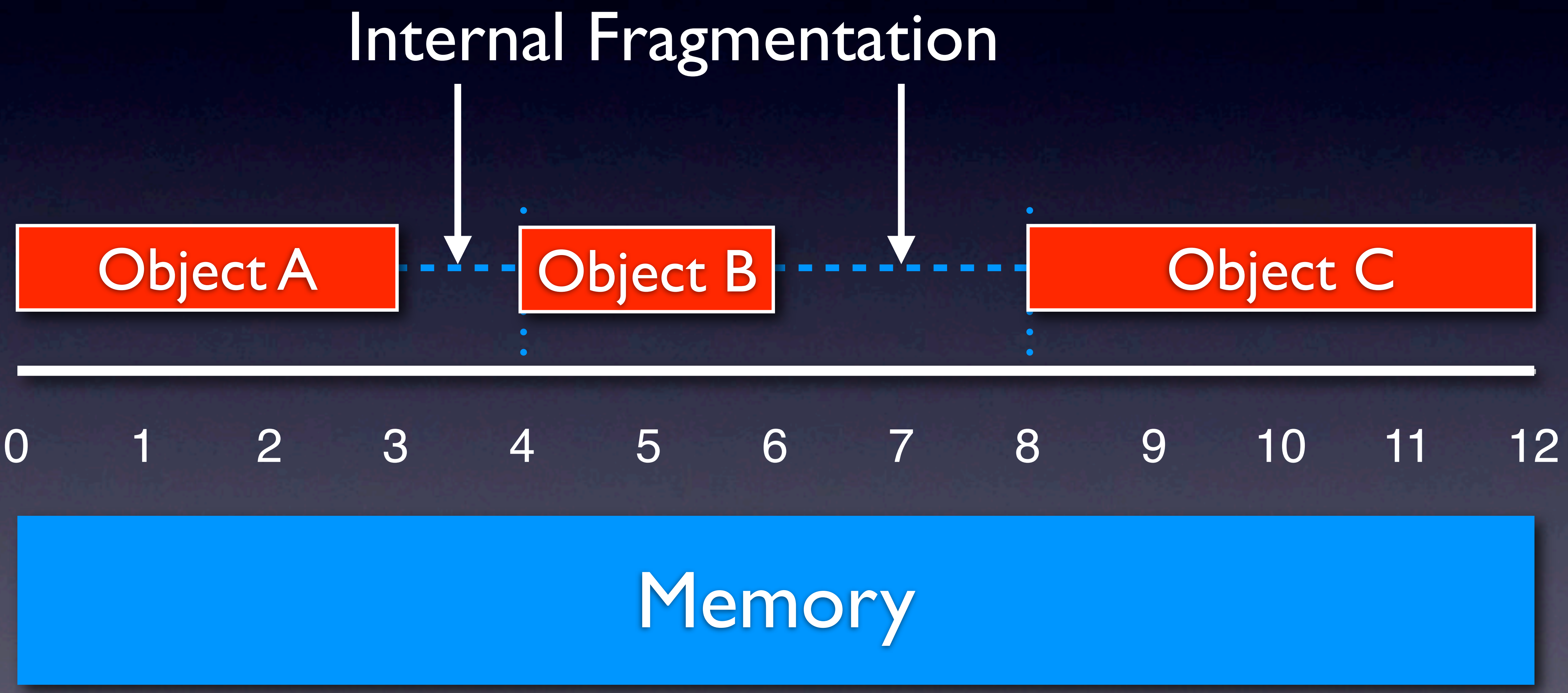
Partitioning



Partitioning

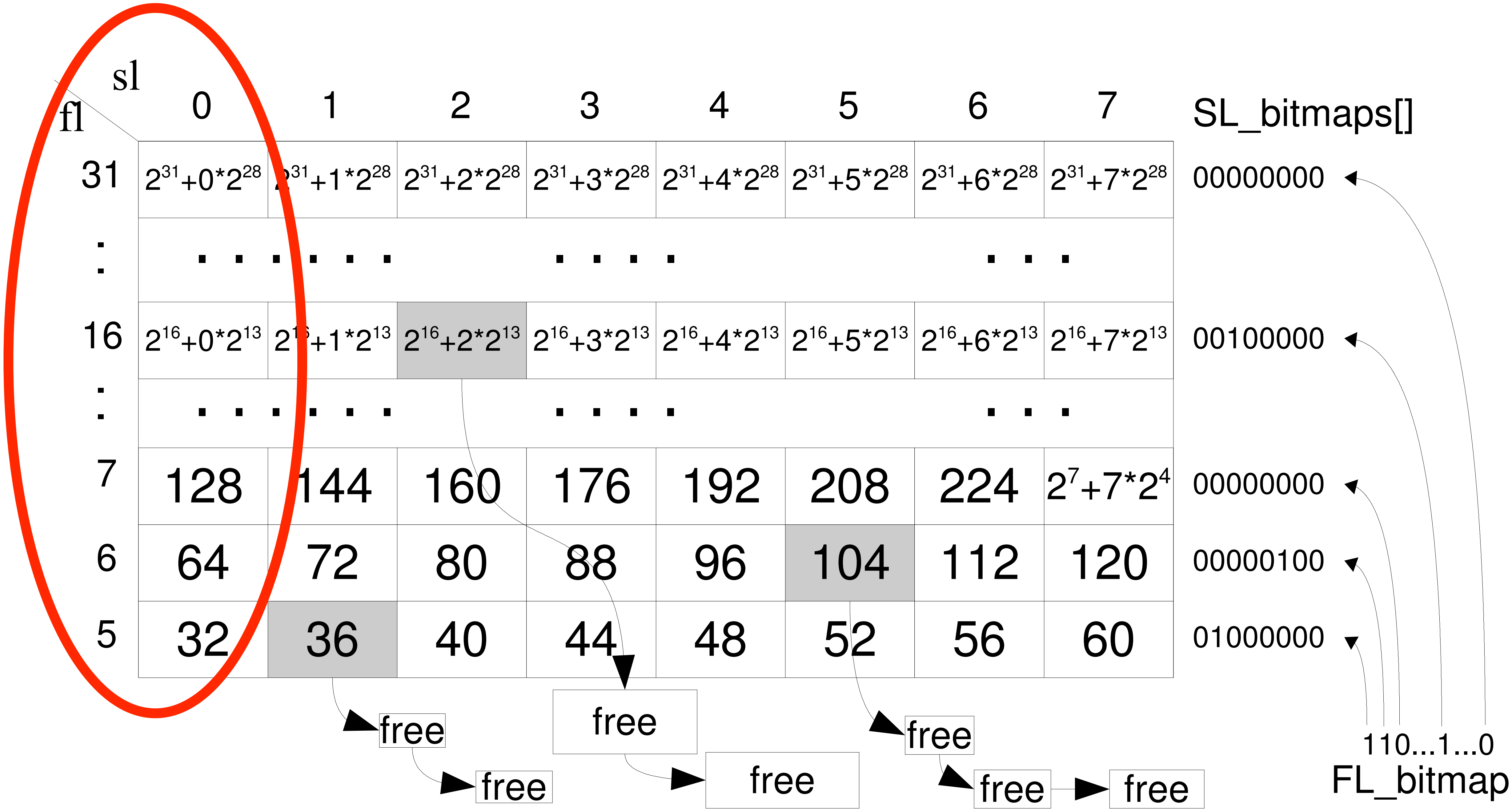


Partitioning



There is a trade-off between external and internal fragmentation

Half-fit

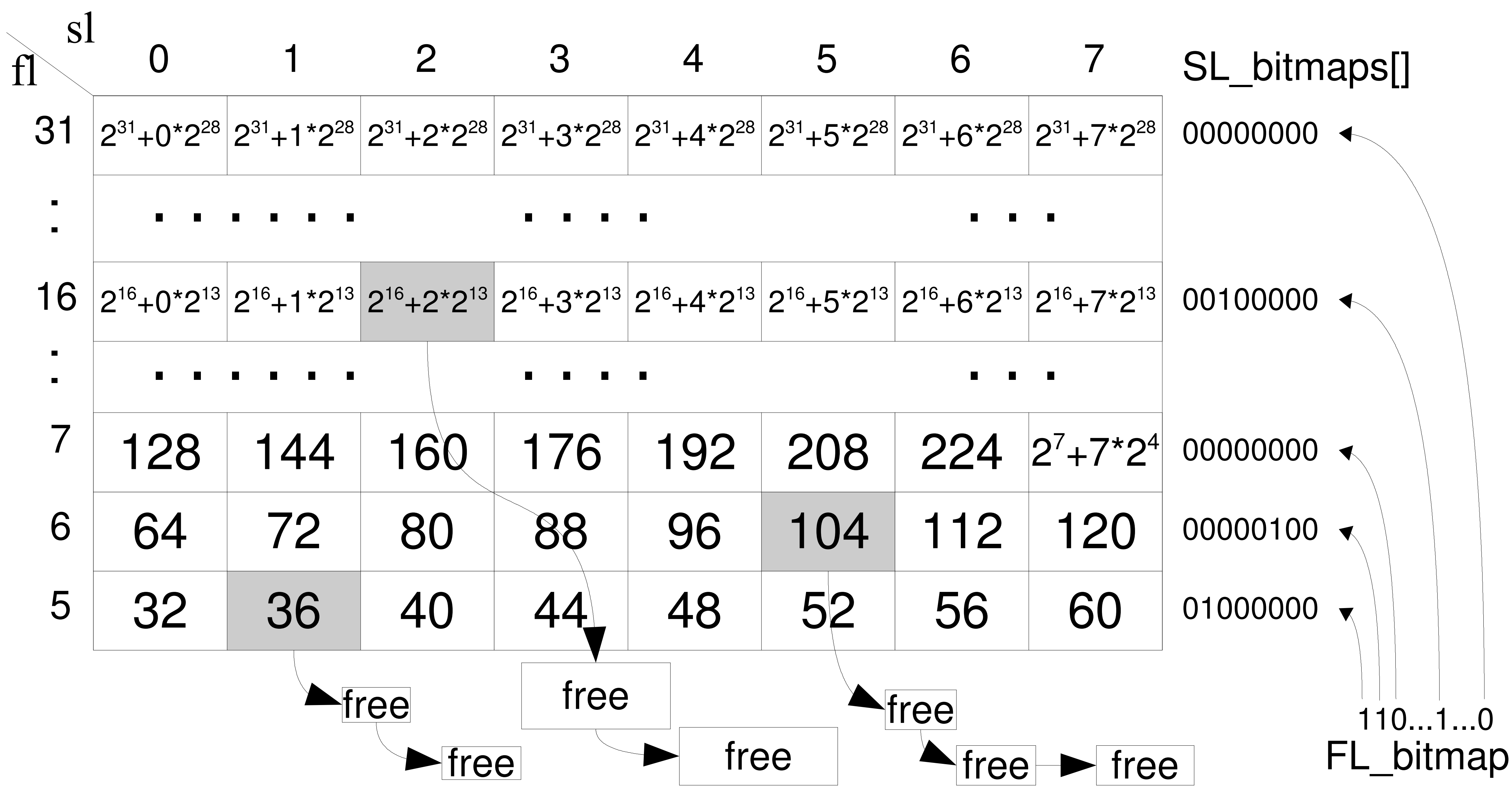


[Masmano et al., In J. of Real-Time Systems, 2008]

Half-fit Complexity

- Allocation:
  - `malloc` takes constant time
- Deallocation:
  - `free` takes constant time
- Access:
  - `read` and `write` take constant time
- <u>Unpredictable</u> fragmentation

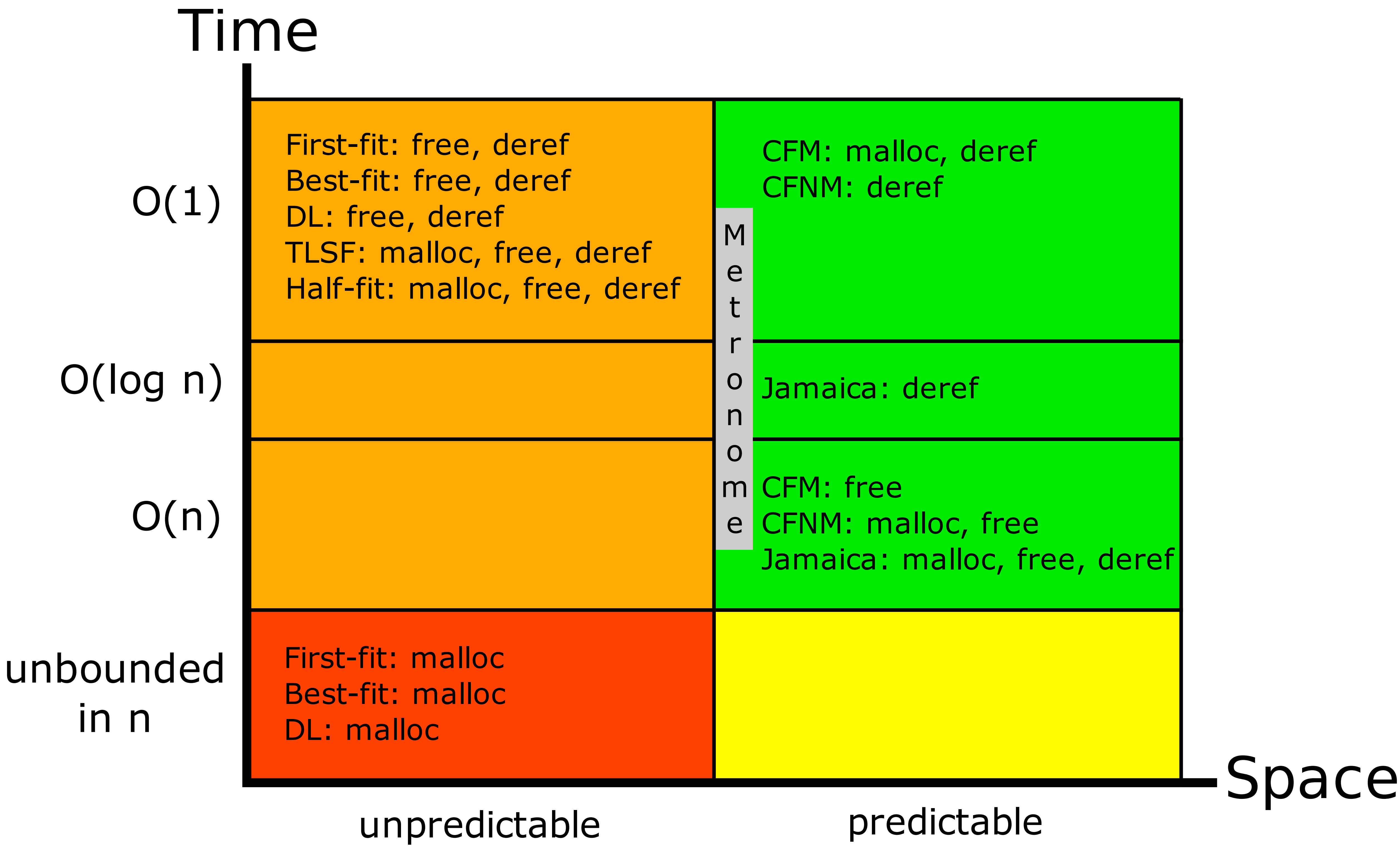
Two-level Segregated Fit (TLSF)



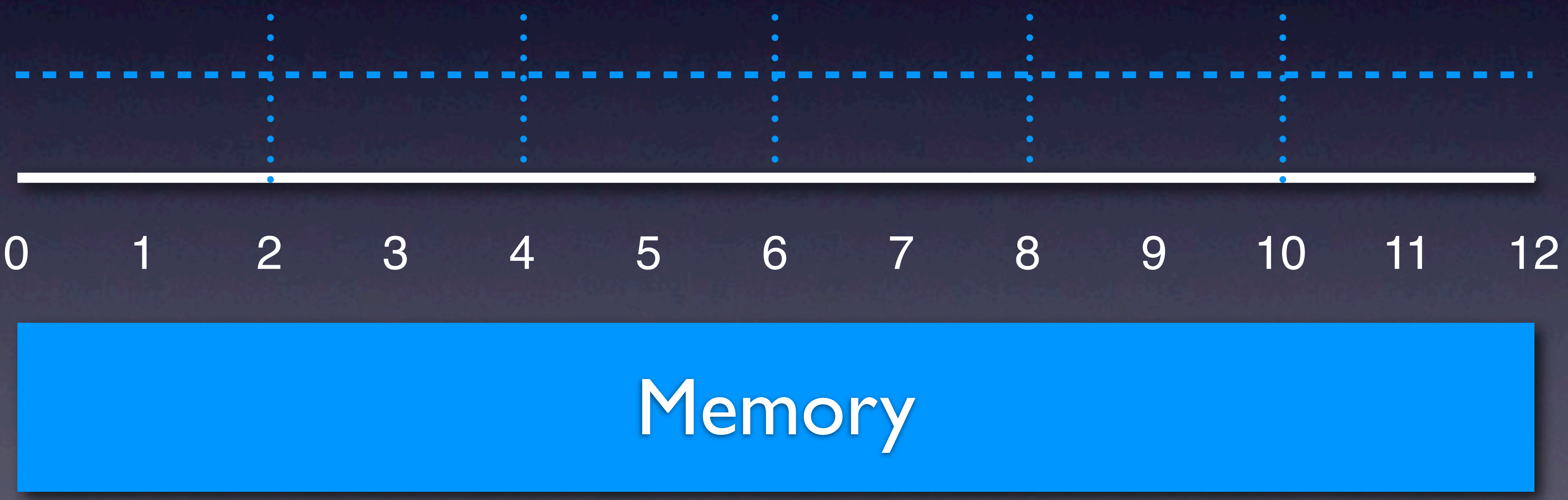
[Masmano et al., In J. of Real-Time Systems, 2008]

TLSF Complexity

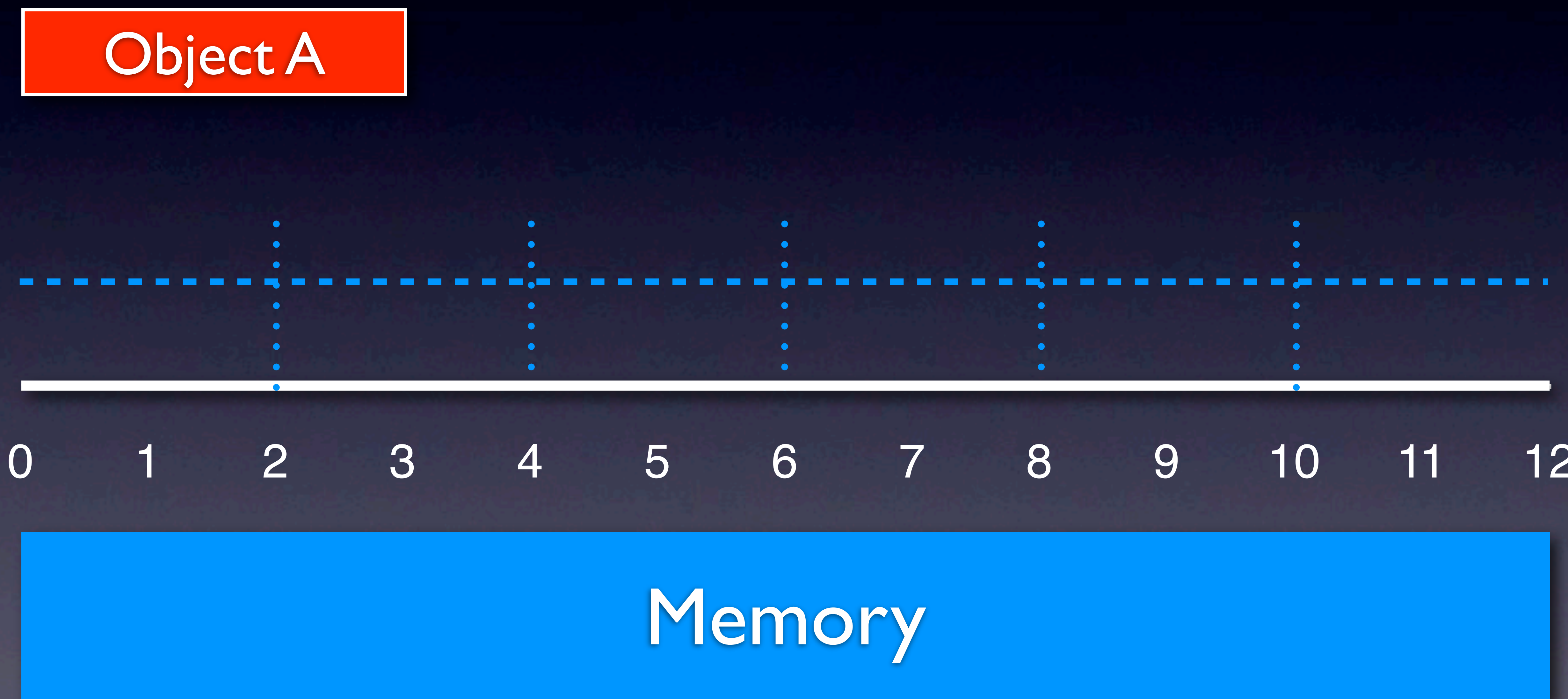
- Allocation:
  - `malloc` takes constant time
- Deallocation:
  - `free` takes constant time
- Access:
  - `read` and `write` take constant time
- Unpredictable fragmentation (yet better than HF)


Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

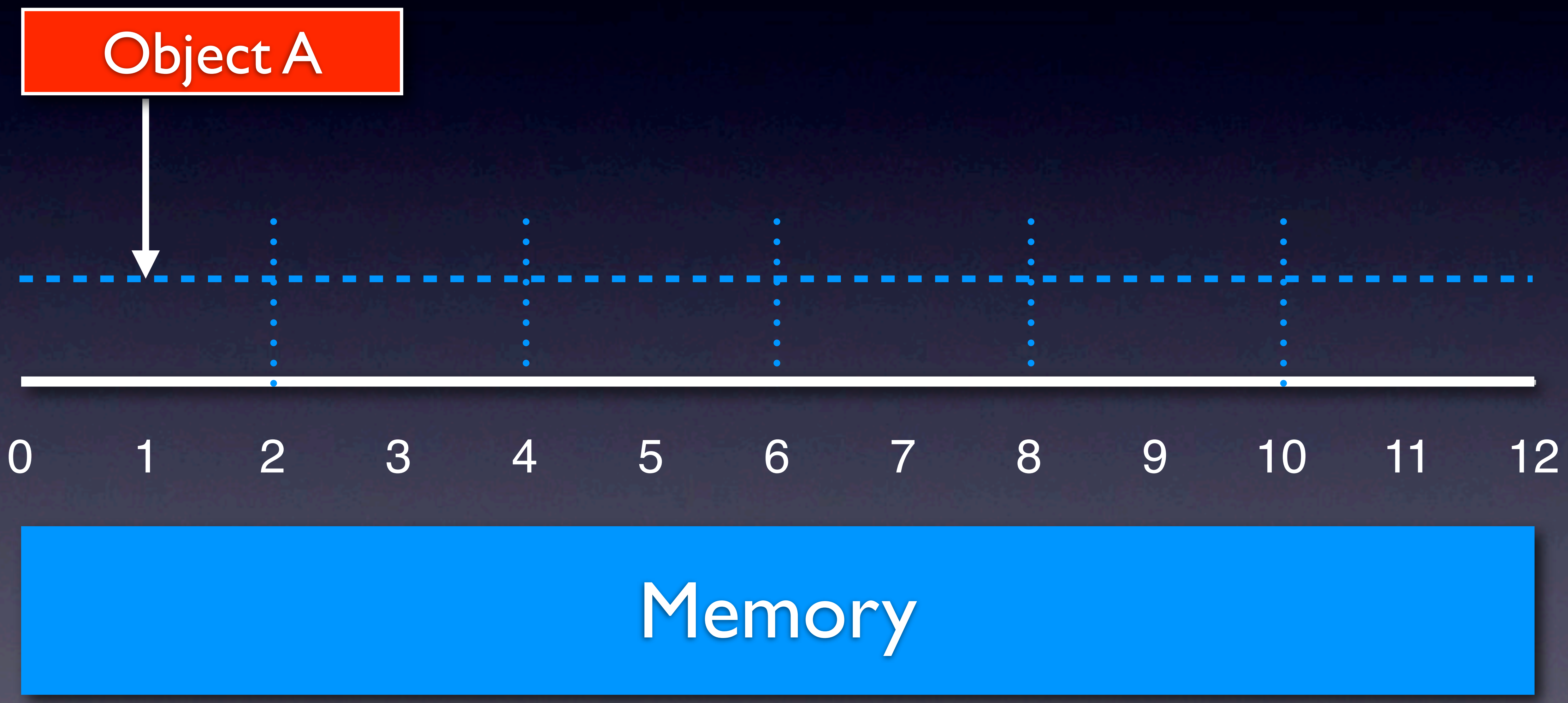
Jamaica



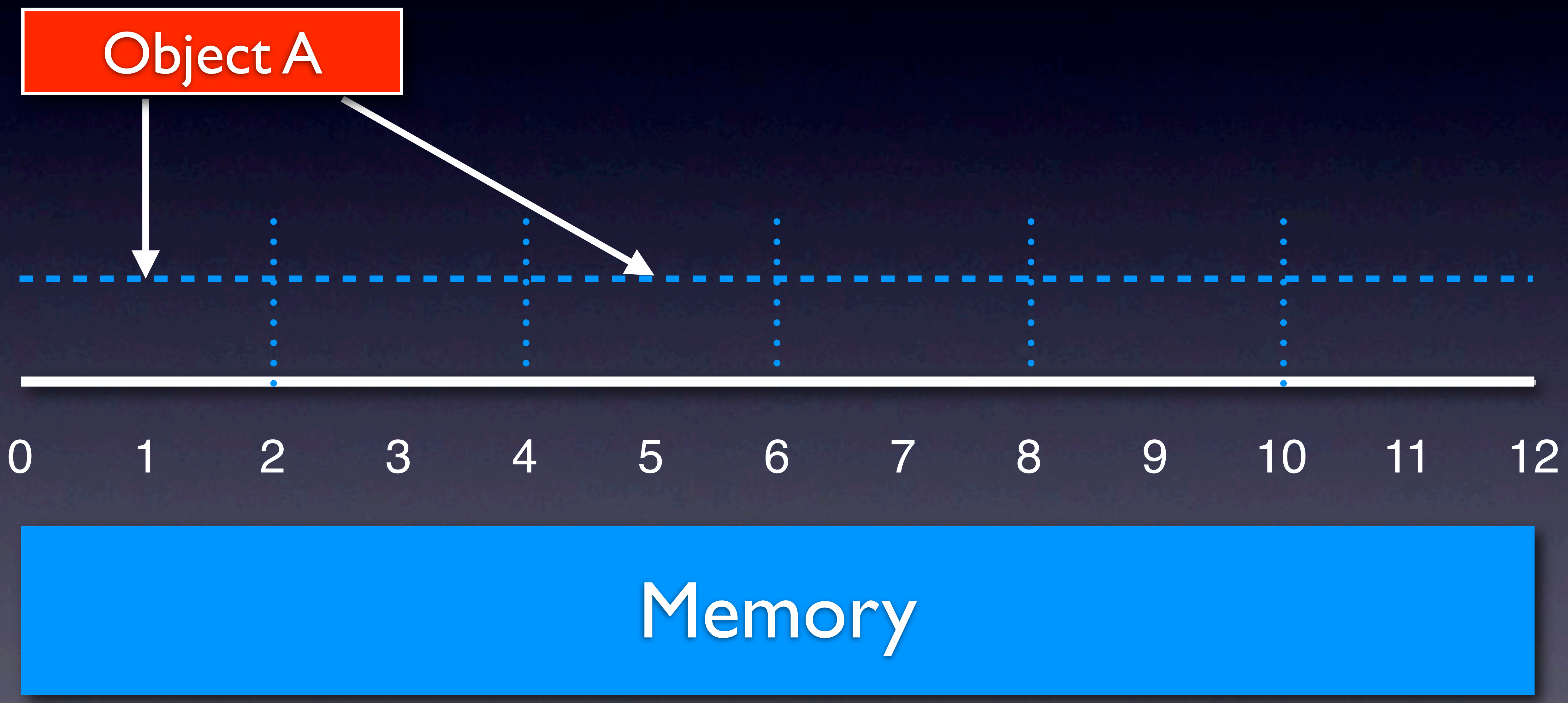
Jamaica



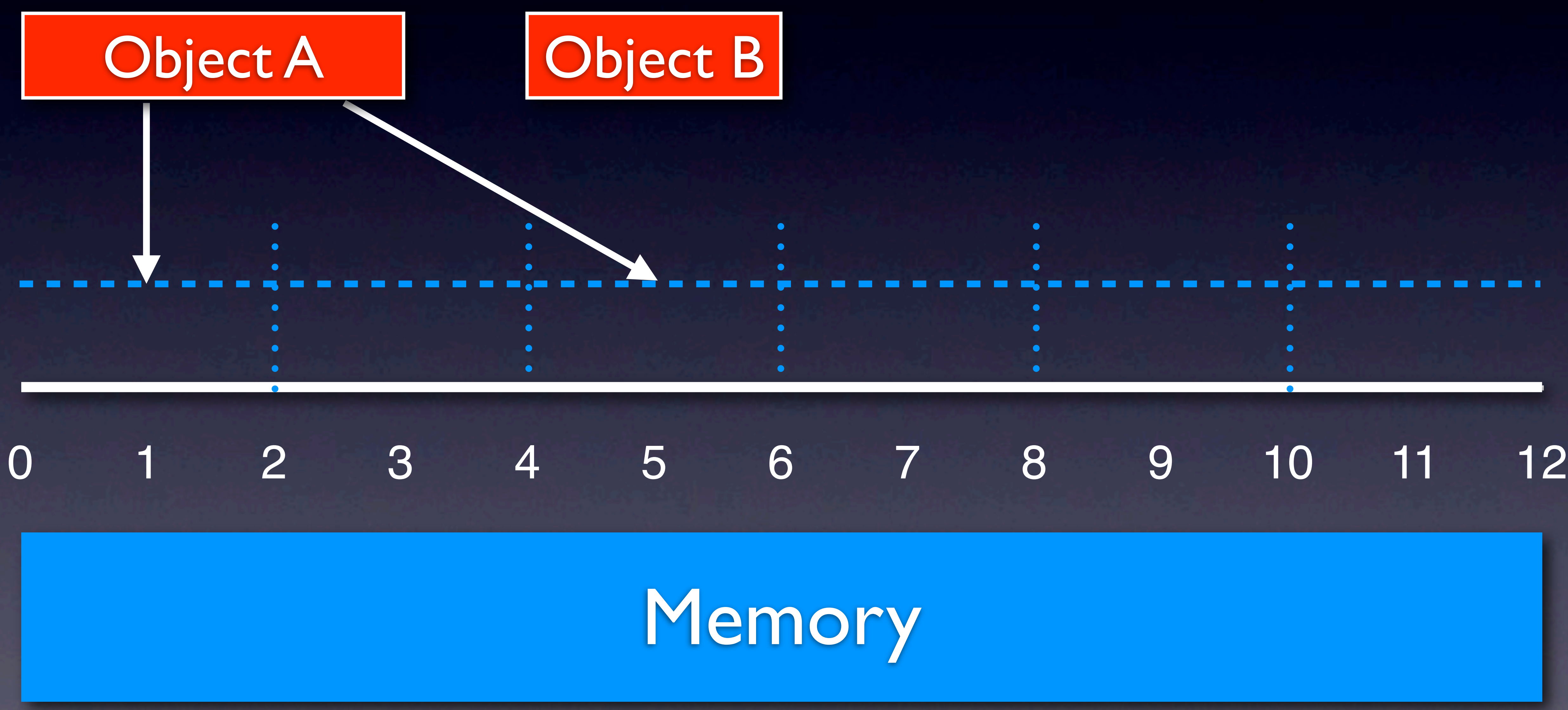
Jamaica



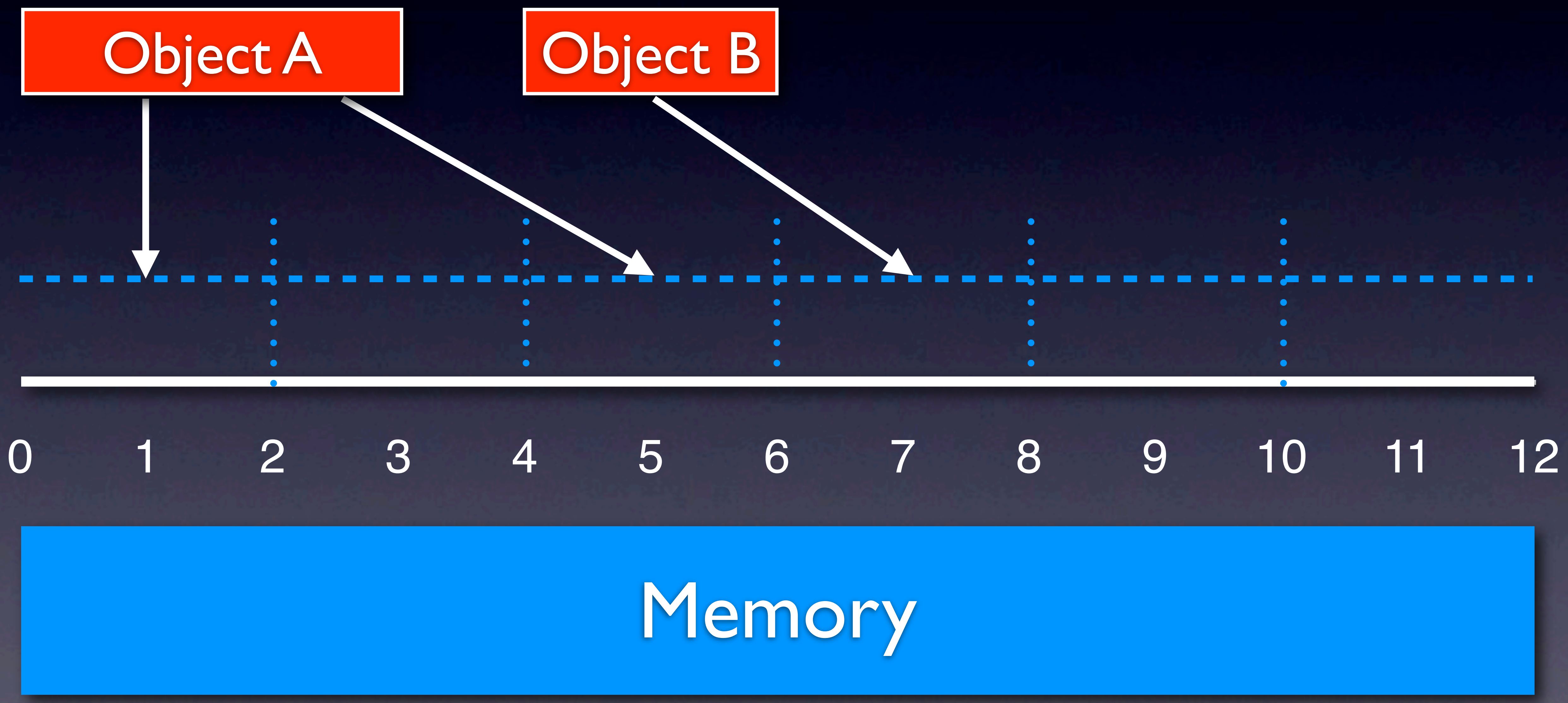
Jamaica



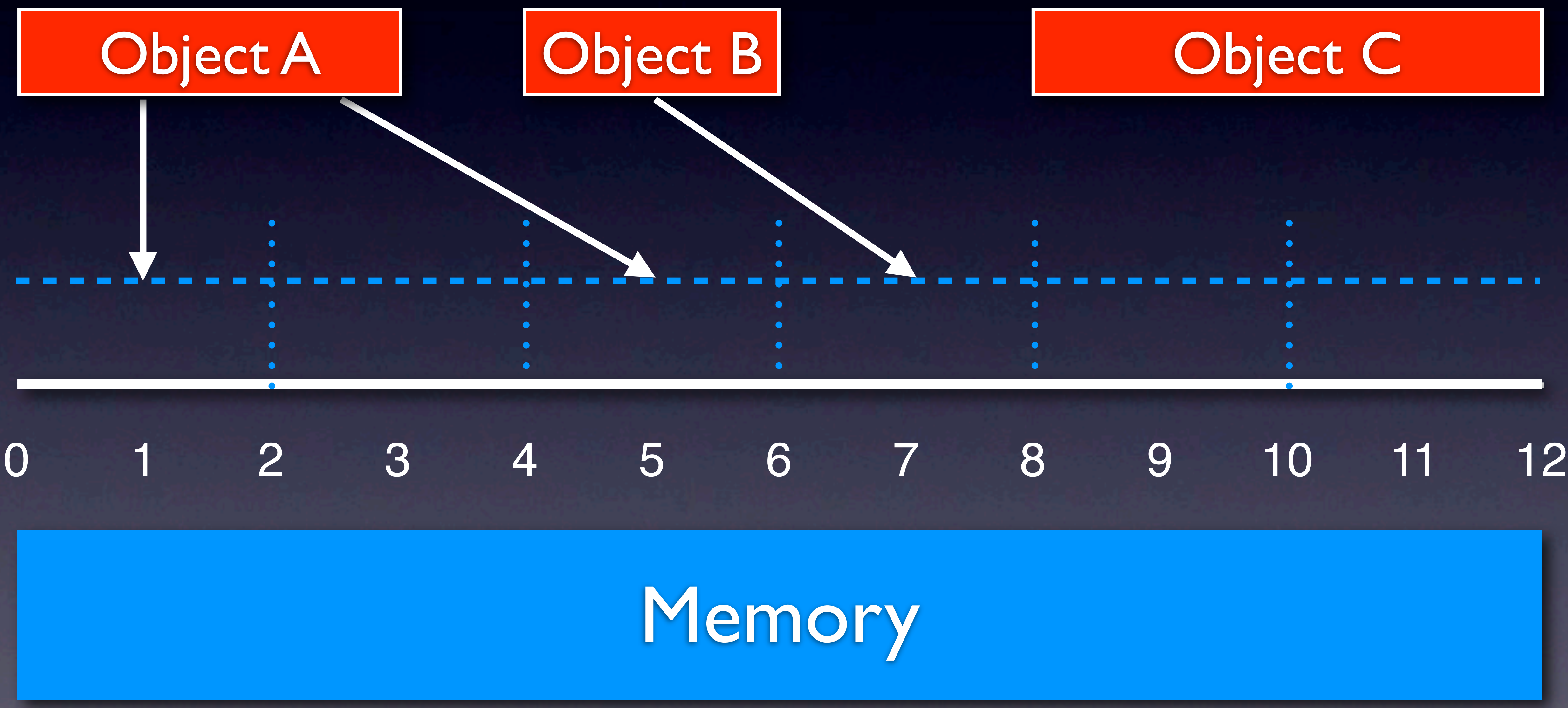
Jamaica



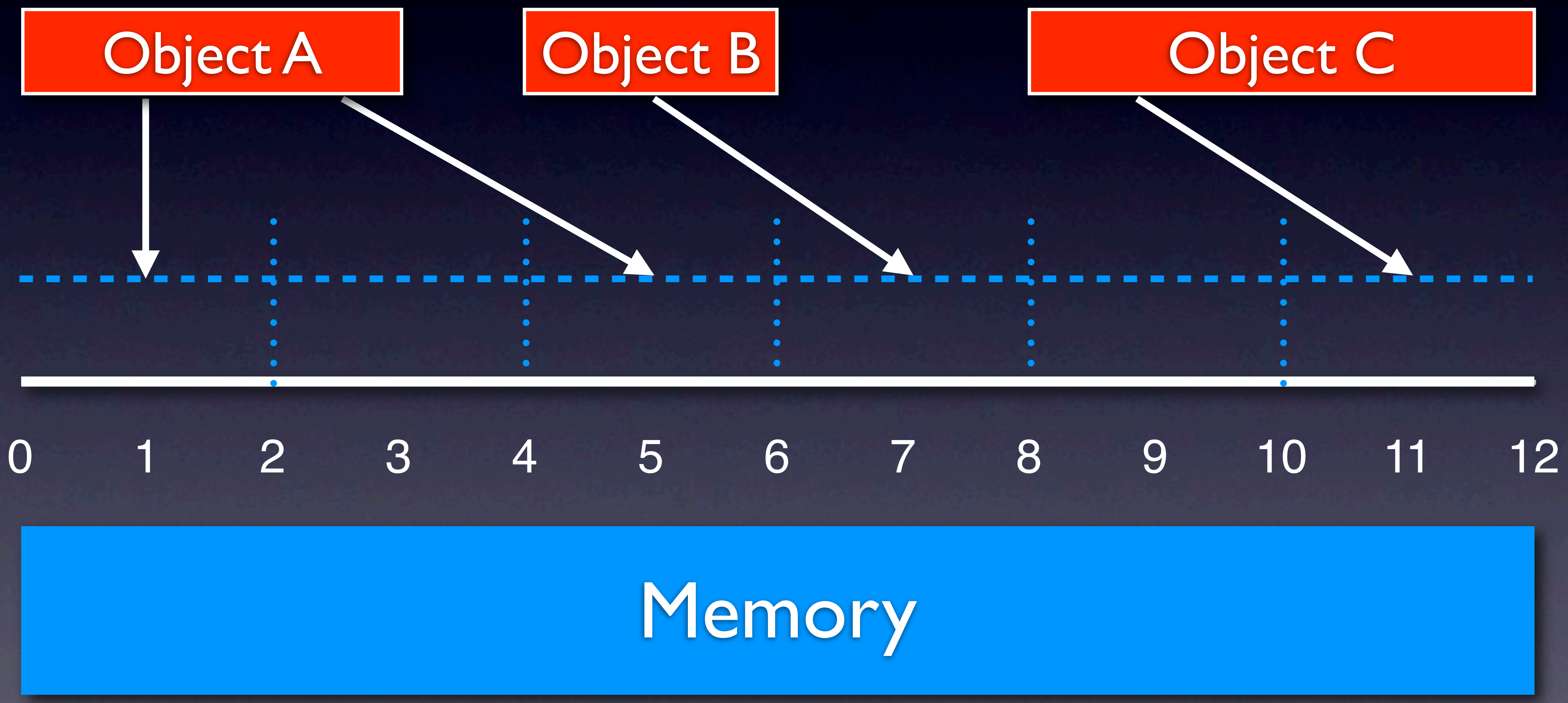
Jamaica



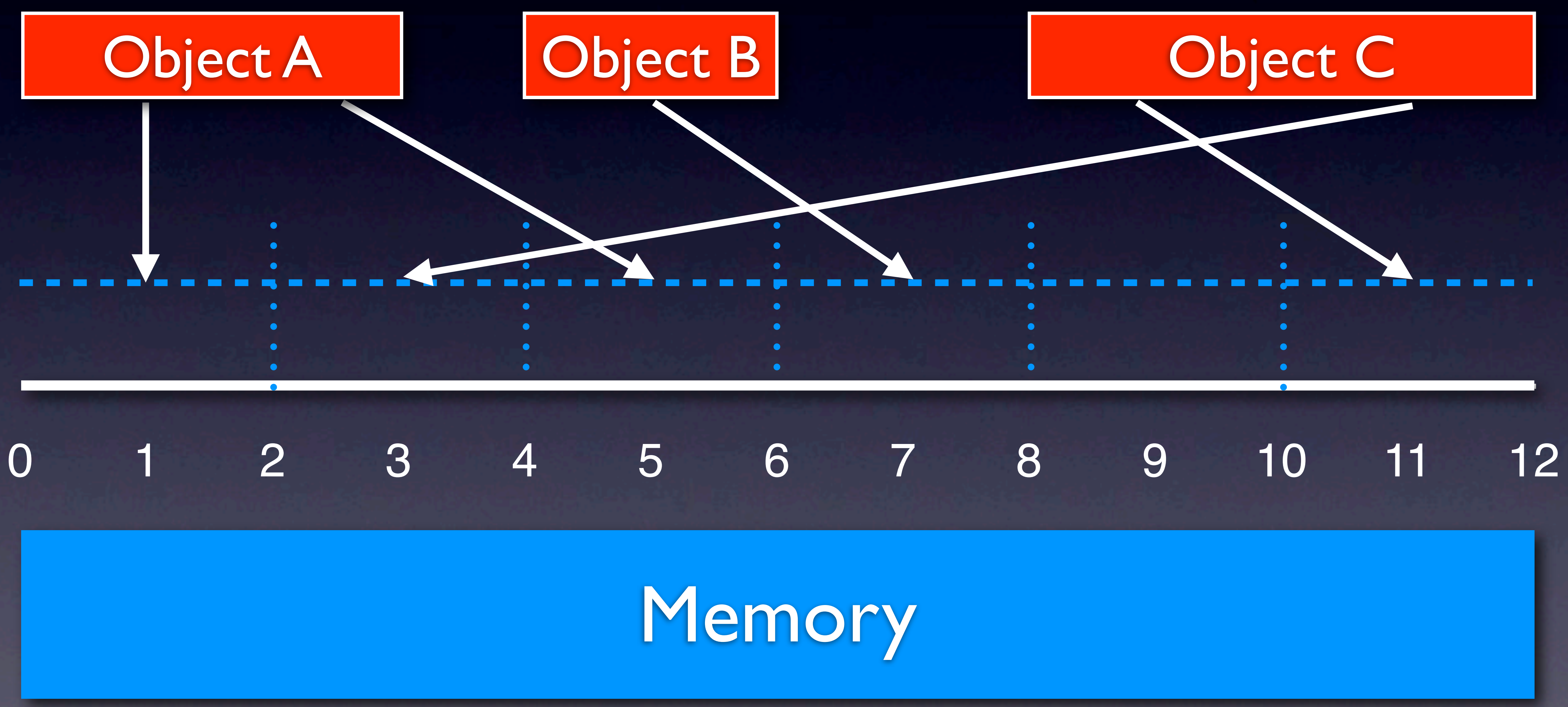
Jamaica



Jamaica

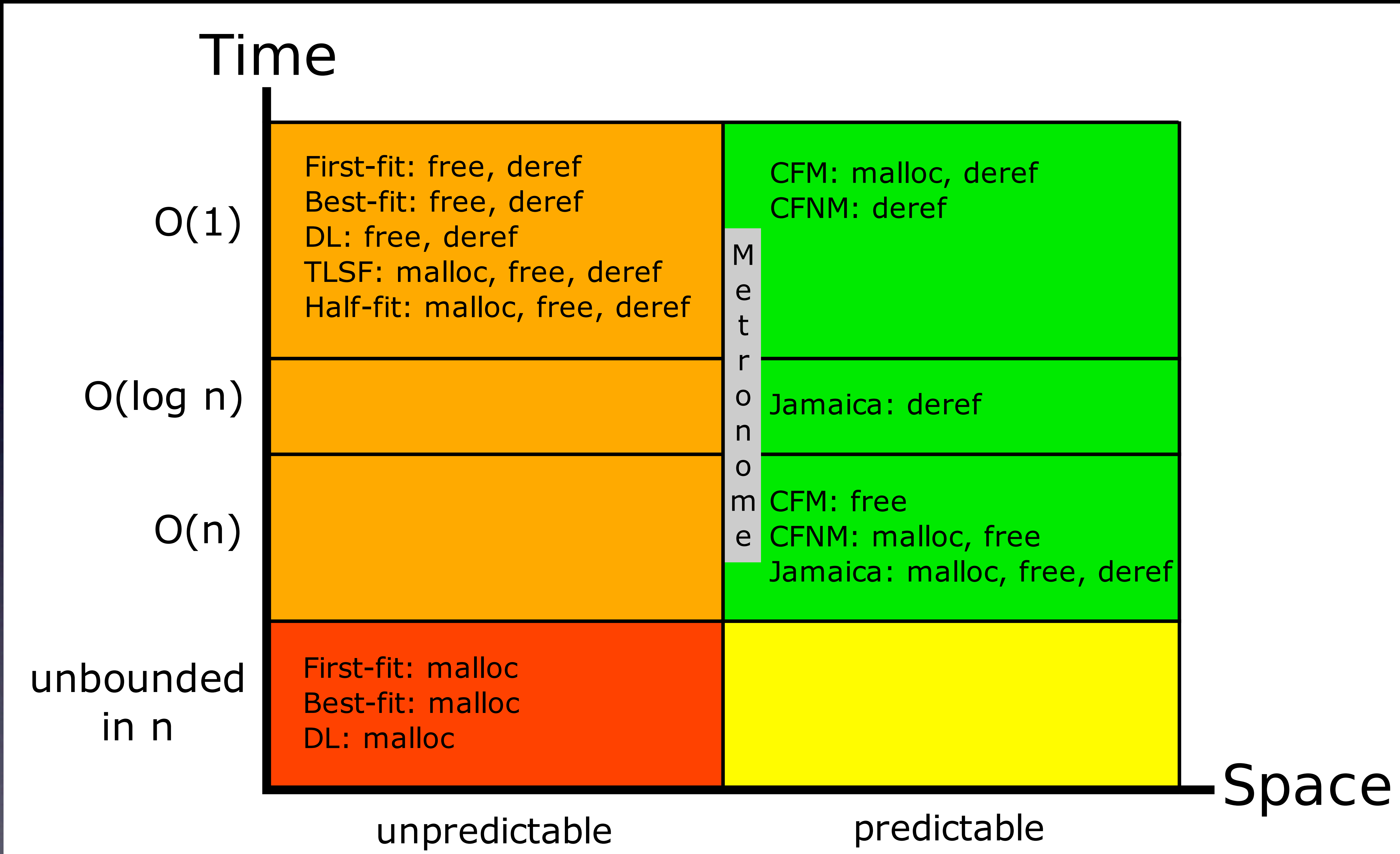


Jamaica



Jamaica Complexity

- Allocation:
  - `malloc(n)` takes time proportional to `n`
- Deallocation:
  - `free(n)` takes time proportional to `n`
- Access:
  - `read` and `write` take time <u>proportional</u> to `n`
- Predictable fragmentation


Time
O(1)
O(log n)
O(n)
unbounded in n
First-fit: free, deref
Best-fit: free, deref
DL: free, deref
TLSF: malloc, free, deref
Half-fit: malloc, free, deref
CFM: malloc, deref
CFNM: deref
Metronome
Jamaica: deref
CFM: free
CFNM: malloc, free
Jamaica: malloc, free, deref
First-fit: malloc
Best-fit: malloc
DL: malloc
Space
unpredictable
predictable

Introduction to Compact-fit

Concurrent
Compact-fit

Concurrency & Scalability versus Fragmentation & Compaction

Questions

- Does allocation/deallocation throughput scale with multiple processors?

Questions

- Does allocation/deallocation throughput scale with multiple processors?
- Which aspects influence scalability?

Questions

- Does allocation/deallocation throughput scale with multiple processors?
- Which aspects influence scalability?
- Does compaction of large objects harm system latency?

Questions

- Does allocation/deallocation throughput scale with multiple processors?
- Which aspects influence scalability?
- Does compaction of large objects harm system latency?
- Does concurrency and incrementality affect memory consumption?

Partial Compaction

- Per-size-class partial compaction bound κ bounds size-class fragmentation:
    - $\kappa = 1$: fully compacting
    - $1 < \kappa < \infty$: partially compacting
    - $\kappa = \infty$: non-compacting

Partial Compaction

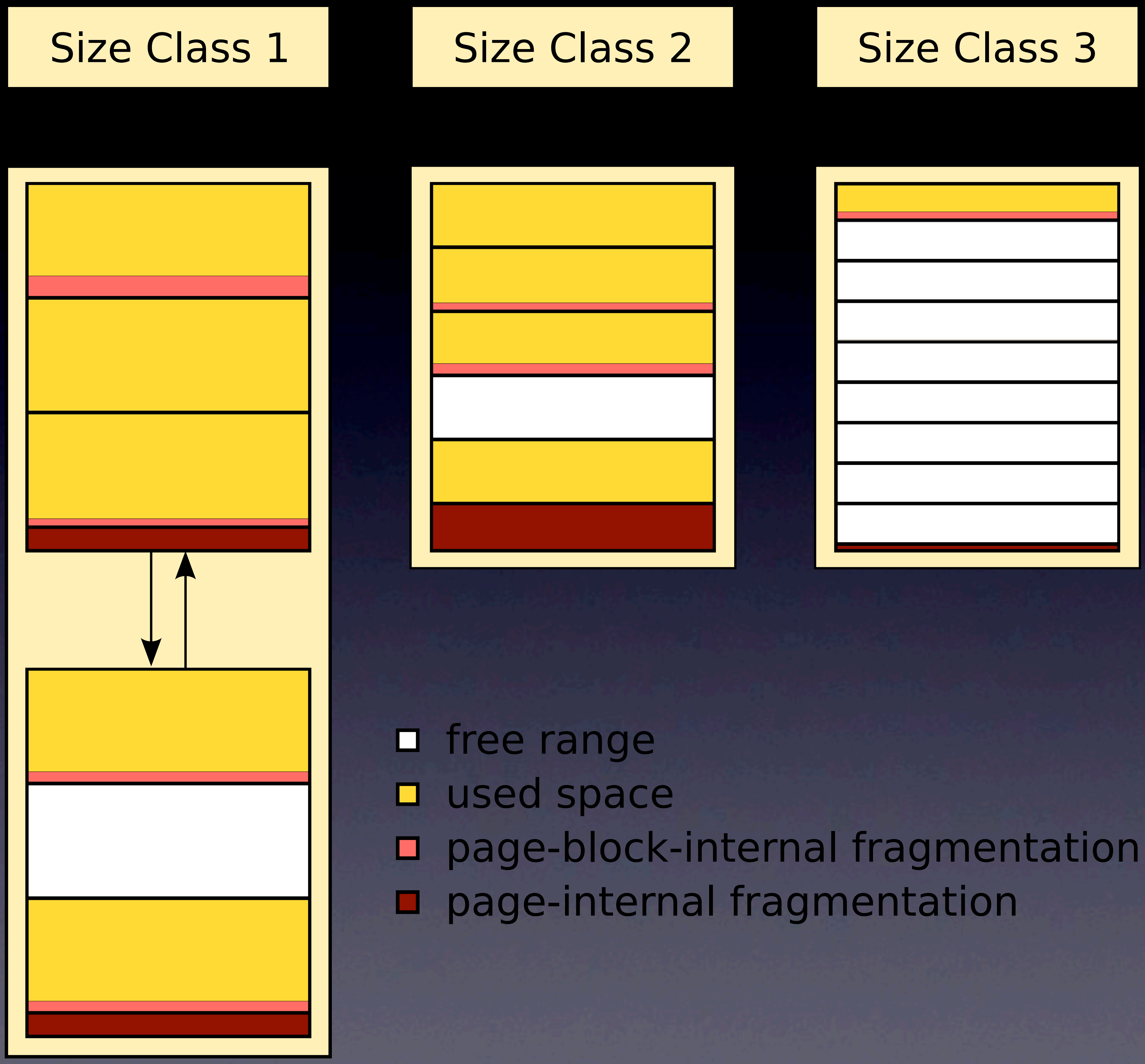
- Per-size-class partial compaction bound κ bounds size-class fragmentation:
  - $\kappa = 1$: fully compacting
  - $1 < \kappa < \infty$: partially compacting
  - $\kappa = \infty$: non-compacting
- Non-compacting CF can be optimized by not using abstract addresses

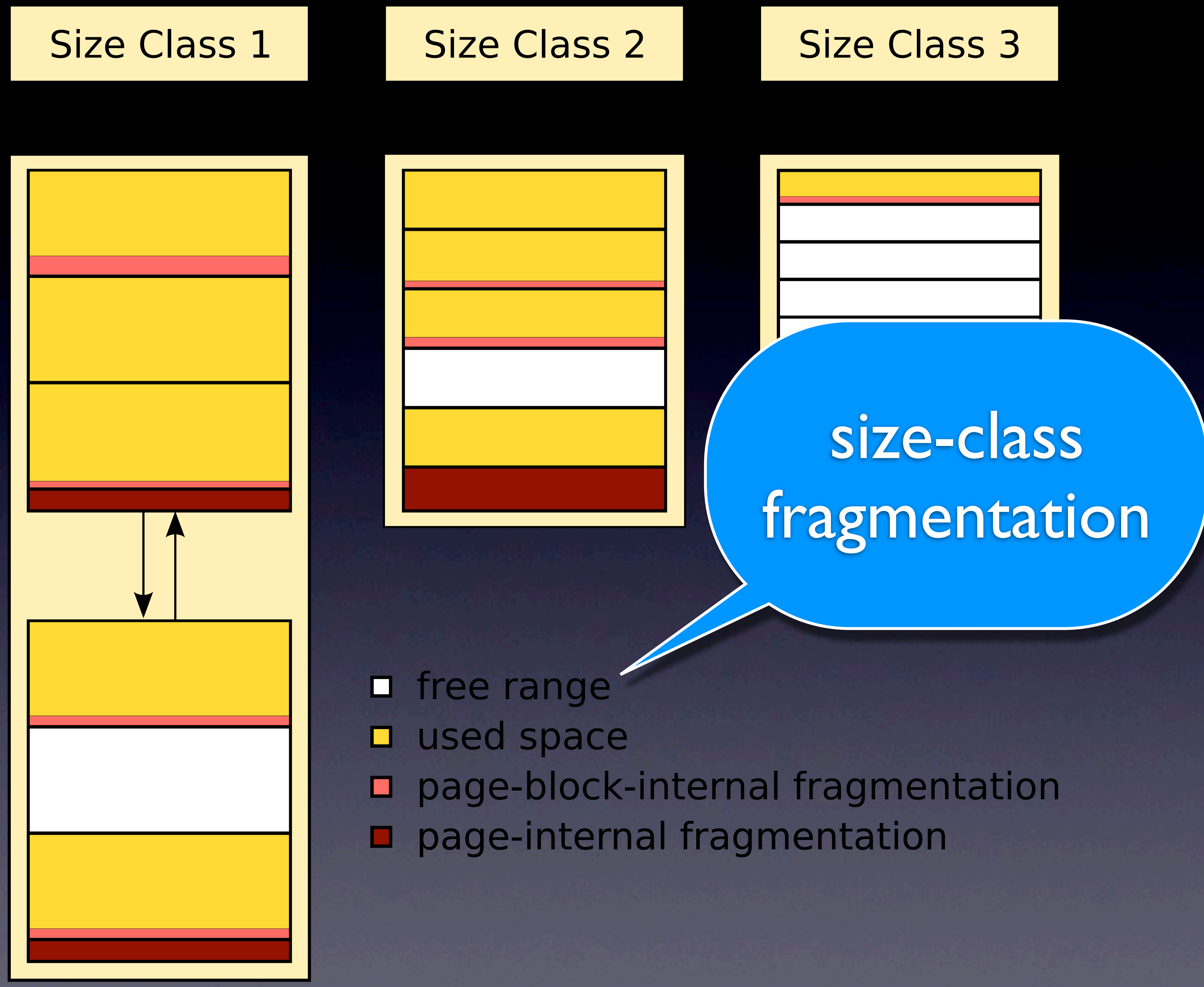
Fragmentation through Partitioning

- Fragmentation through partitioning is fixed at compile time and is not controlled by partial compaction:
  - Page-block-internal fragmentation
  - Page-internal fragmentation

Fragmentation through Partitioning

- Fragmentation through partitioning is fixed at compile time and is not controlled by partial compaction:
  - Page-block-internal fragmentation
  - Page-internal fragmentation
- May dominate overall fragmentation


Size Class 1
Size Class 2
Size Class 3
free range
used space
page-block-internal fragmentation
page-internal fragmentation


Size Class 1
Size Class 2
Size Class 3
size-class fragmentation
free range
used space
page-block-internal fragmentation
page-internal fragmentation

Incremental Compaction

- Global compaction increment ι bounds size of memory involved in any atomic compaction operation:
  - $1 < \iota < \infty$: incremental compaction of objects larger than ι
  - $\iota = \infty$: non-incremental compaction

Incremental Compaction

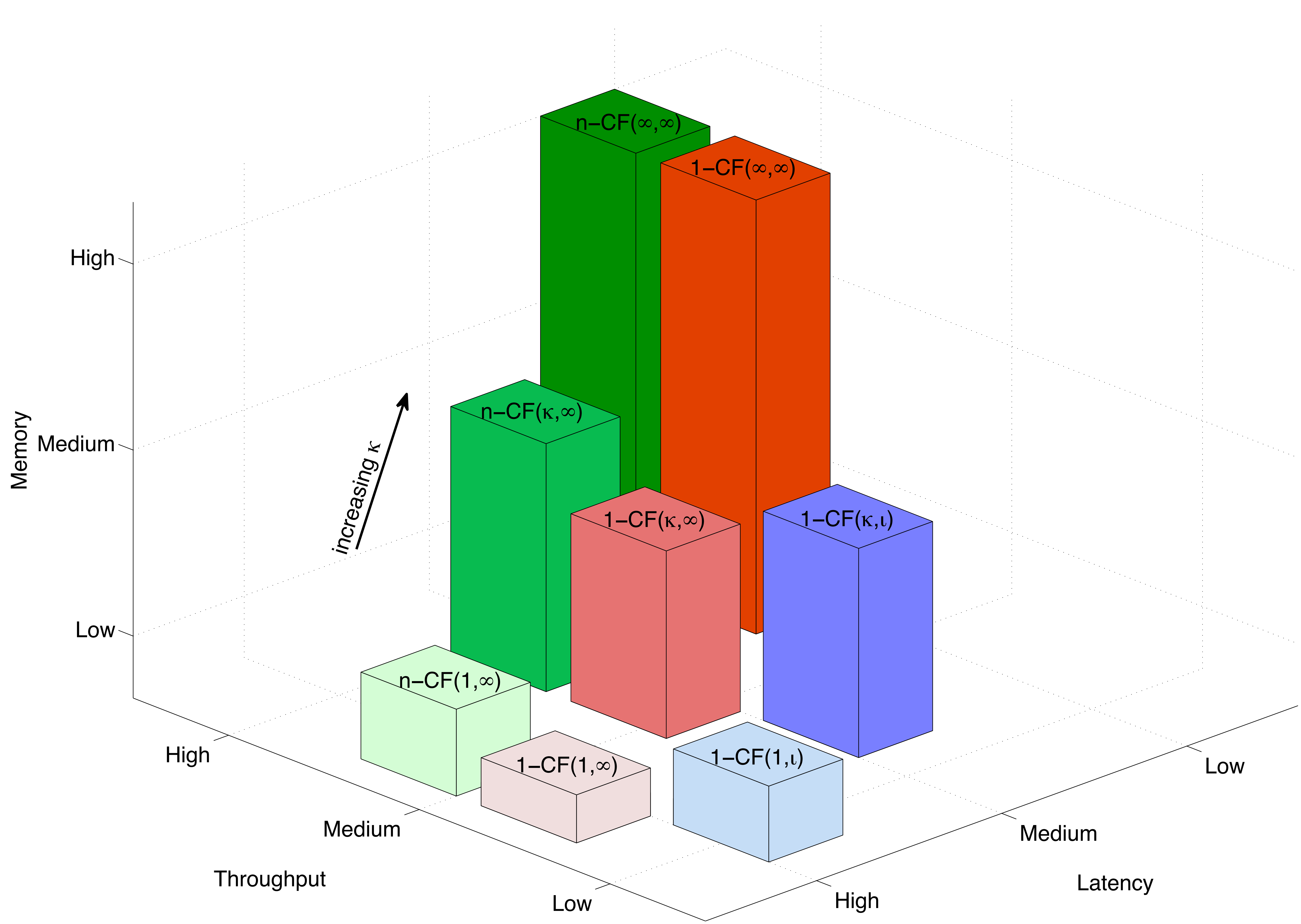
- <u>Global</u> compaction increment ι bounds size of memory involved in any atomic compaction operation:
  - $1 < \iota < \infty$: incremental compaction of objects larger than ι
  - $\iota = \infty$: non-incremental compaction
- Incremental compaction creates transient size-class fragmentation

CF Configurations

- 1-CF(κ, ι)
  - one CF instance for multiple threads
  - partial compaction bound κ
  - compaction increment ι

CF Configurations

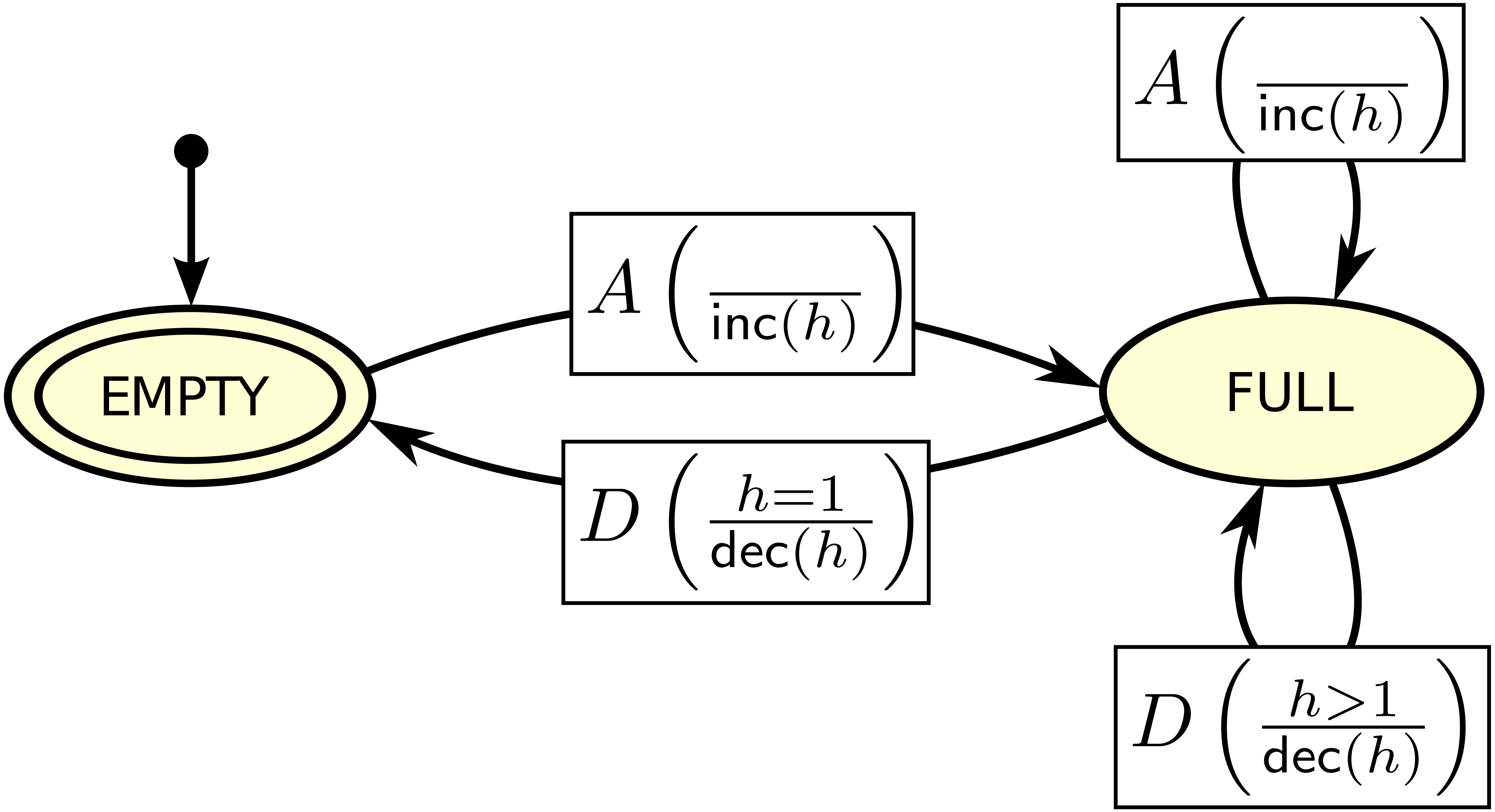
- 1-CF(κ, ι)
  - one CF instance for multiple threads
  - partial compaction bound κ
  - compaction increment ι
- n-CF(κ, ι)
  - n CF instances for n threads
  - allows to control degree of sharing



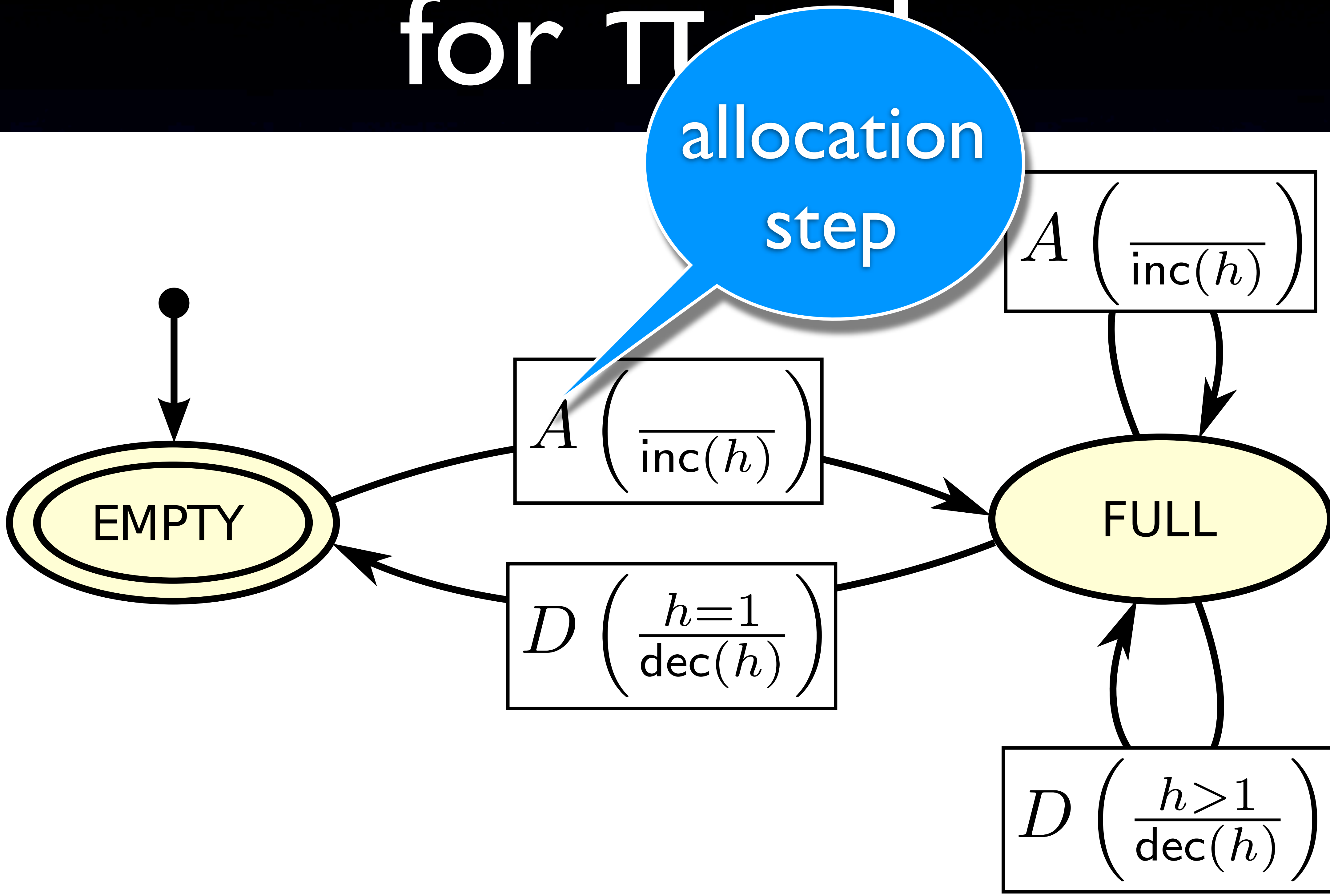
n–CF(∞,∞)
1–CF(∞,∞)
n–CF(κ,∞)
1–CF(κ,∞)
1–CF(κ,ι)
n–CF(1,∞)
1–CF(1,∞)
1–CF(1,ι)
increasing κ
Memory
High
Medium
Low
Throughput
High
Medium
Low
Latency
Low
Medium
High

To make CF
concurrent and incremental
we model the algorithm
as a
finite state machine
whose transitions
must be atomic!

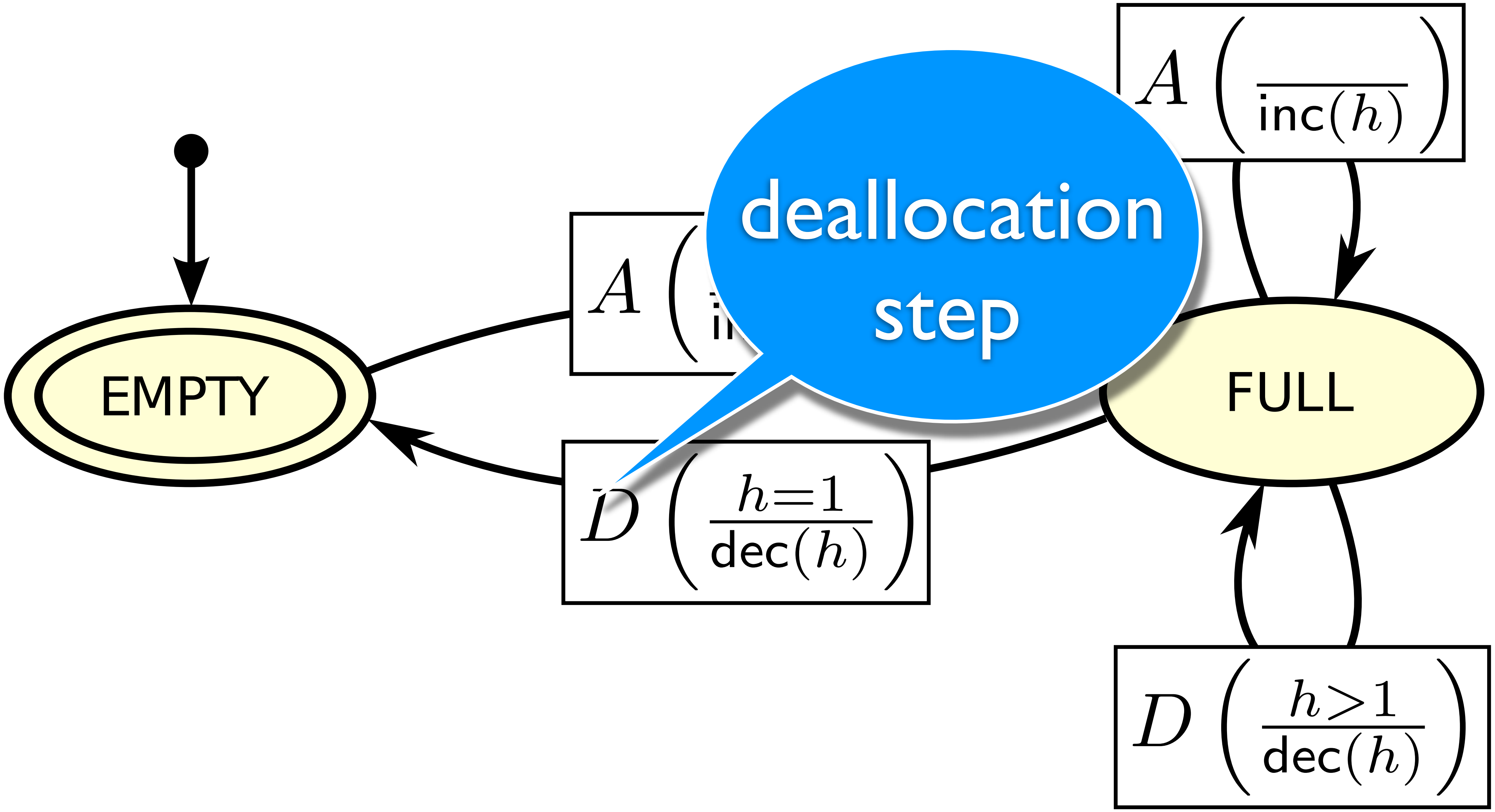
Size-Class Automaton for π = 1



Size-Class Automaton for TT



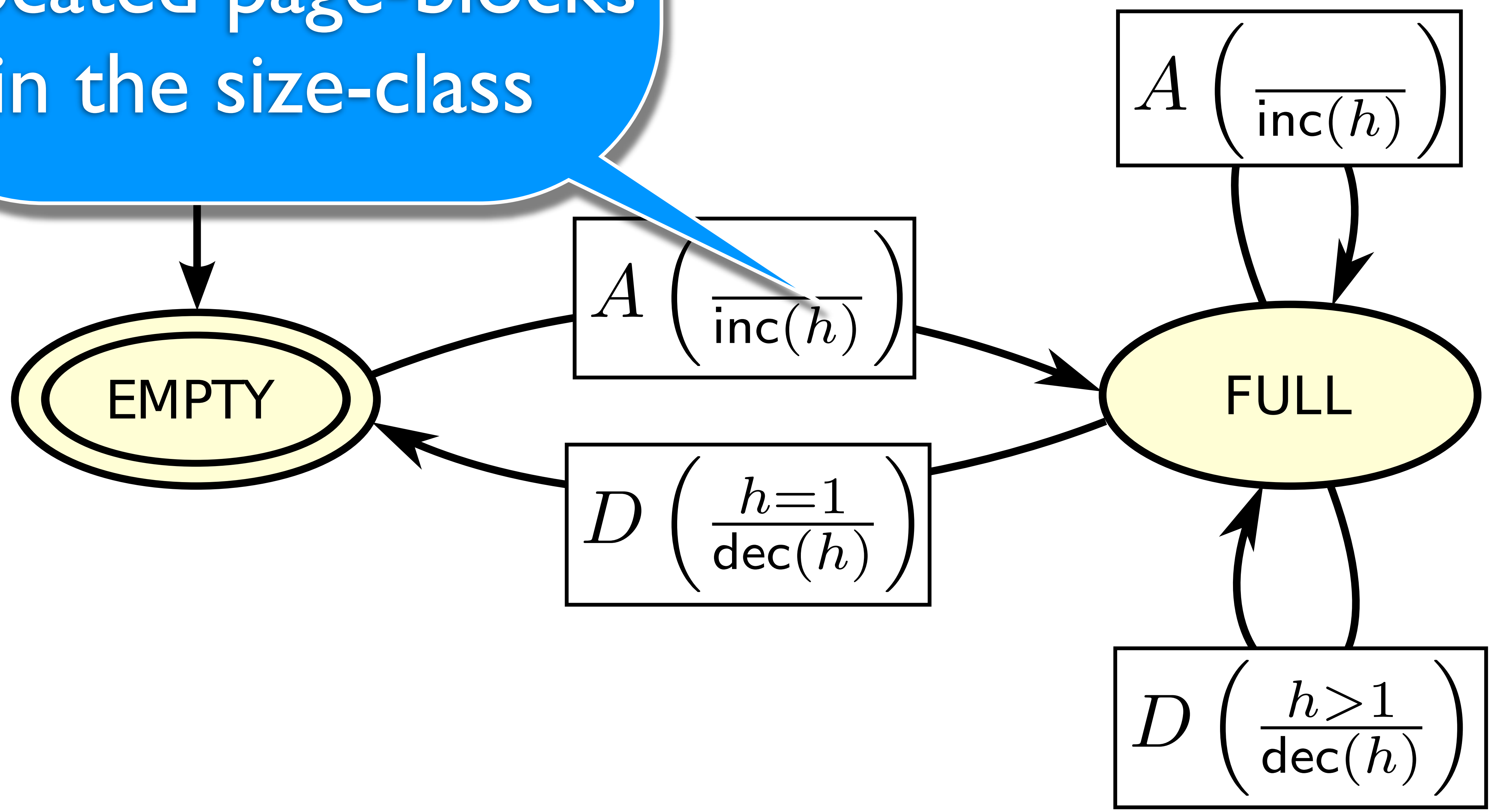
Size-Class Automaton for π = 1



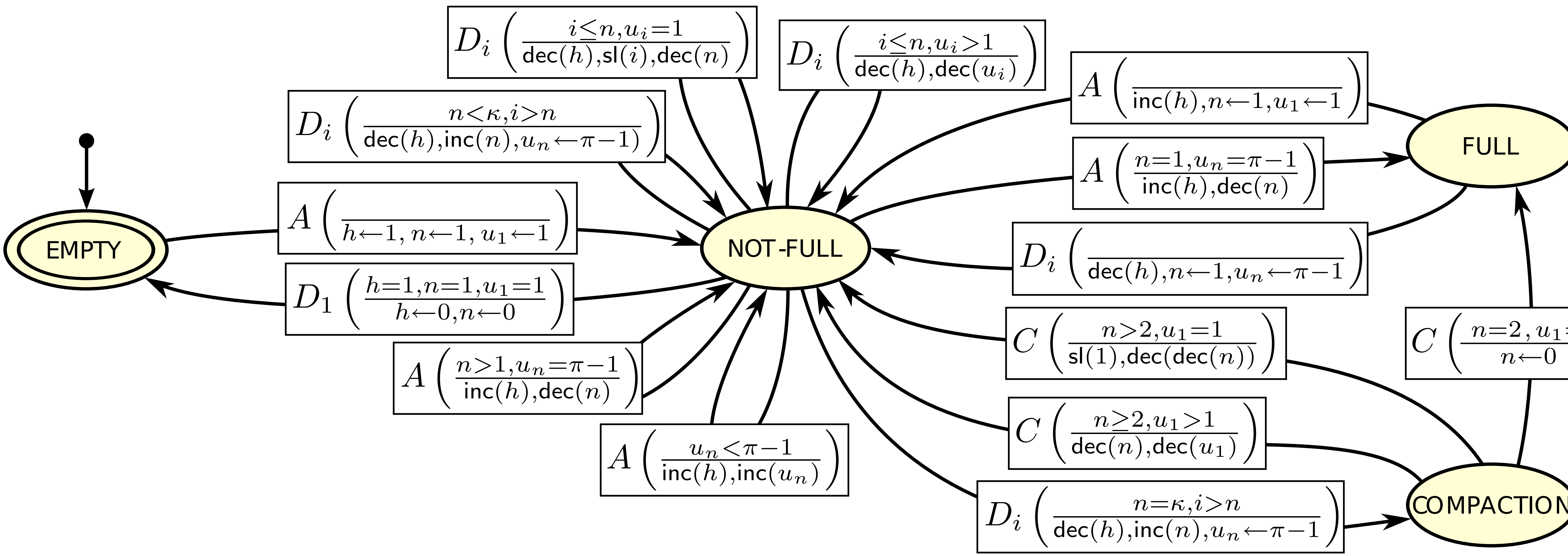
Size-Class Automaton

$\pi = 1$

h is the total # of allocated page-blocks in the size-class

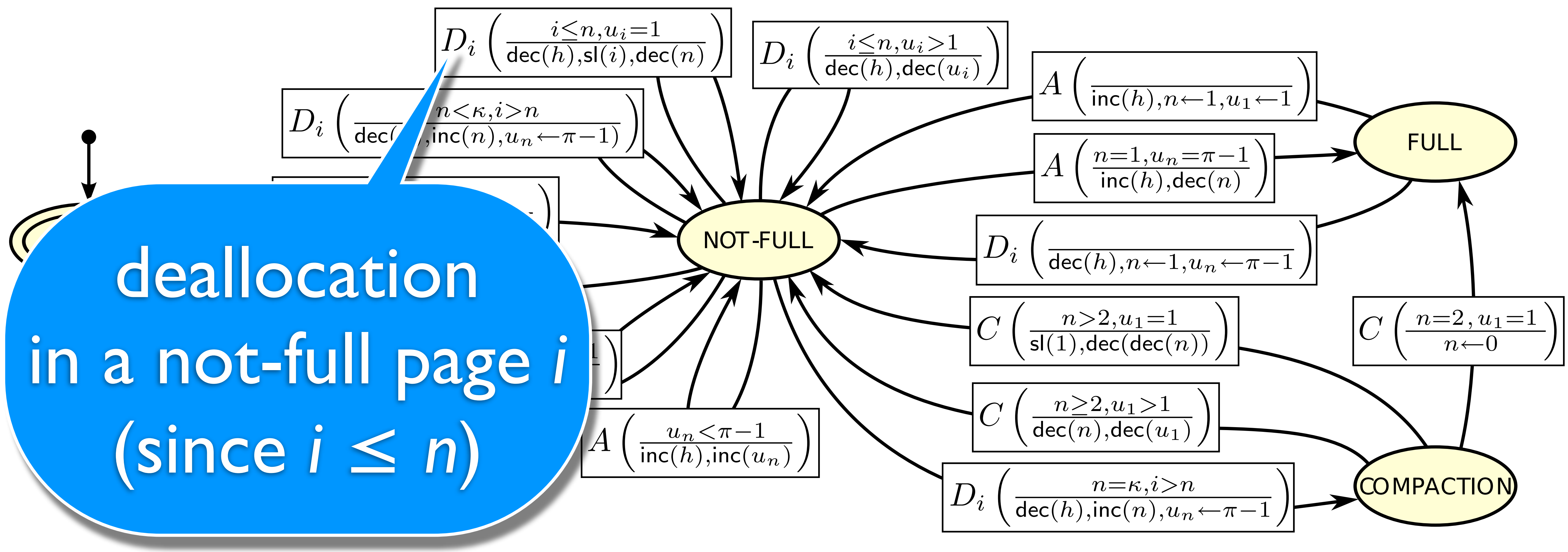


Size-Class Automaton for π > 1



h is the total # of allocated page-blocks in the size-class
n is the # of not-full pages
u_i is the # of used page-blocks in a not-full page *i*

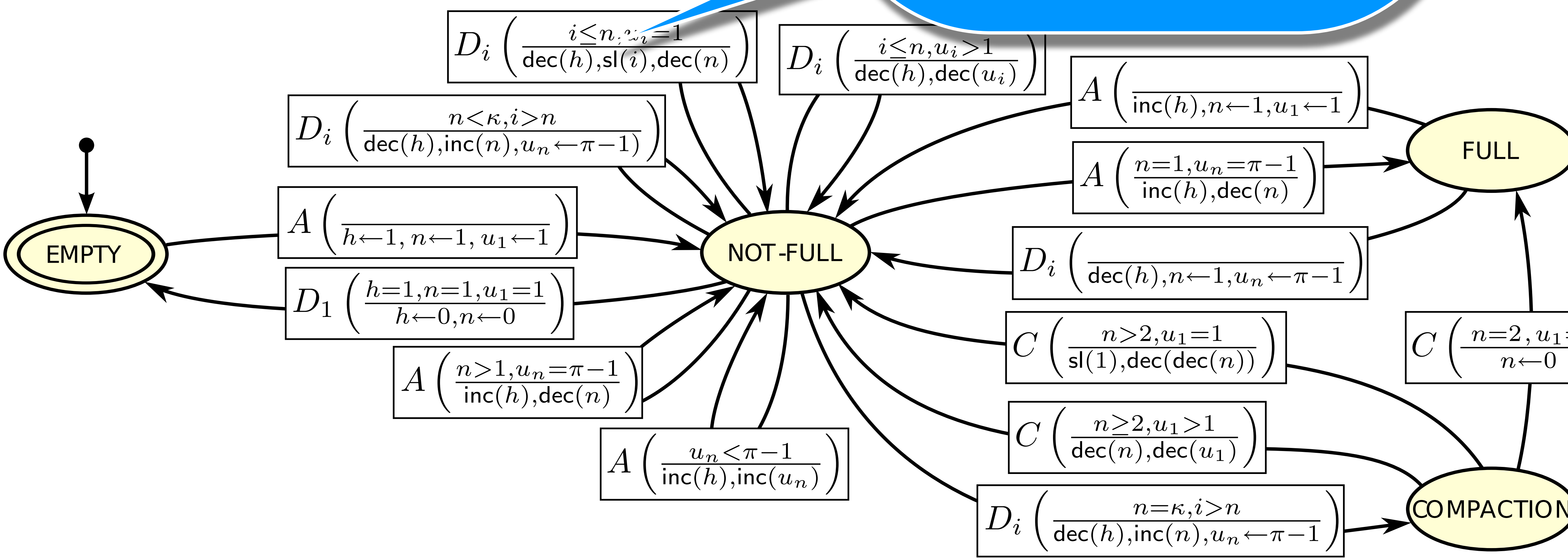
Size-Class Automaton for π > 1



h is the total # of allocated page-blocks in the size-class
n is the # of not-full pages
u_i is the # of used page-blocks in a not-full page *i*

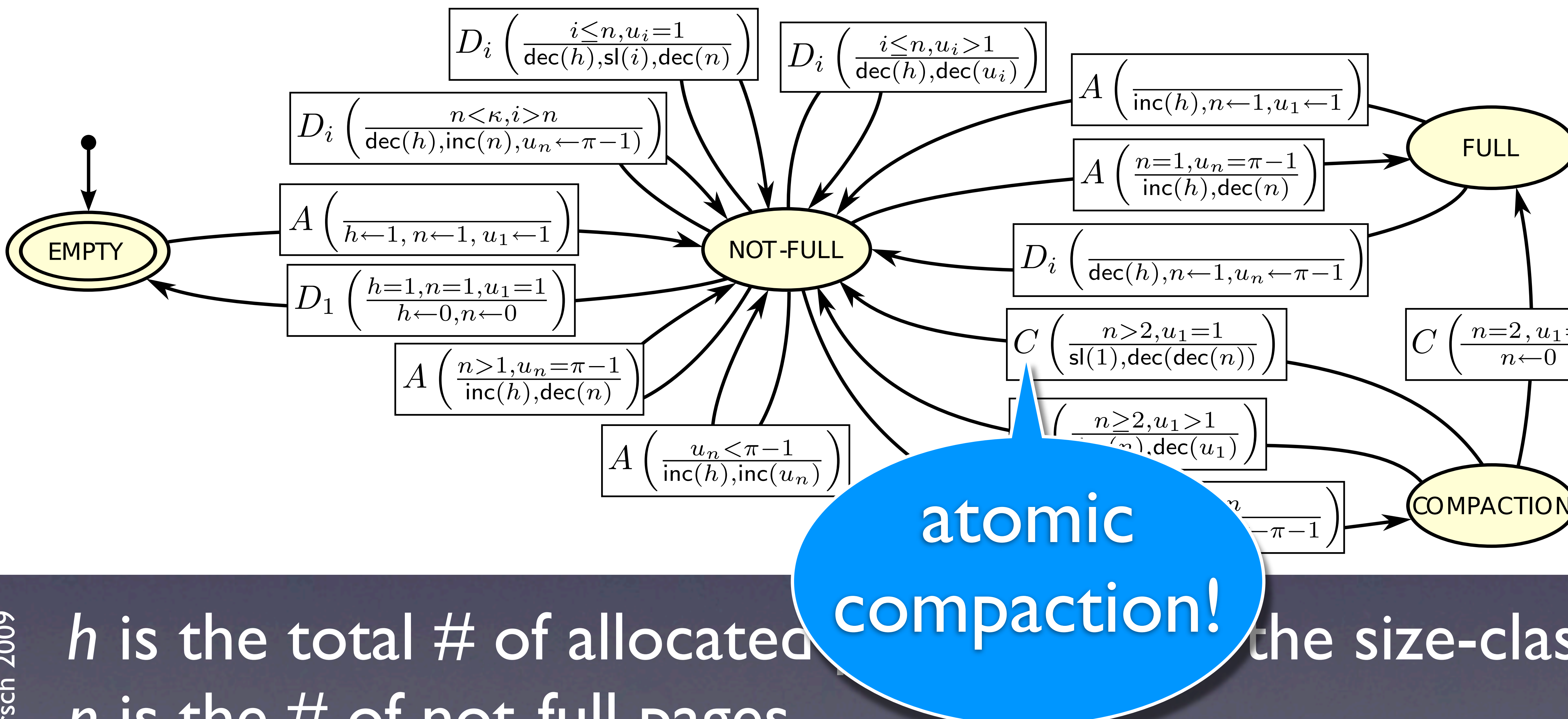
Size-Class A... for T...

remove page since it is now empty



h is the total # of allocated page-blocks in the size-class
n is the # of not-full pages
u_i is the # of used page-blocks in a not-full page i

Size-Class Automaton for π > 1



h is the total # of allocated [...] the size-class

n is the # of not-full pages

u_i is the # of used page-blocks in a not-full page i

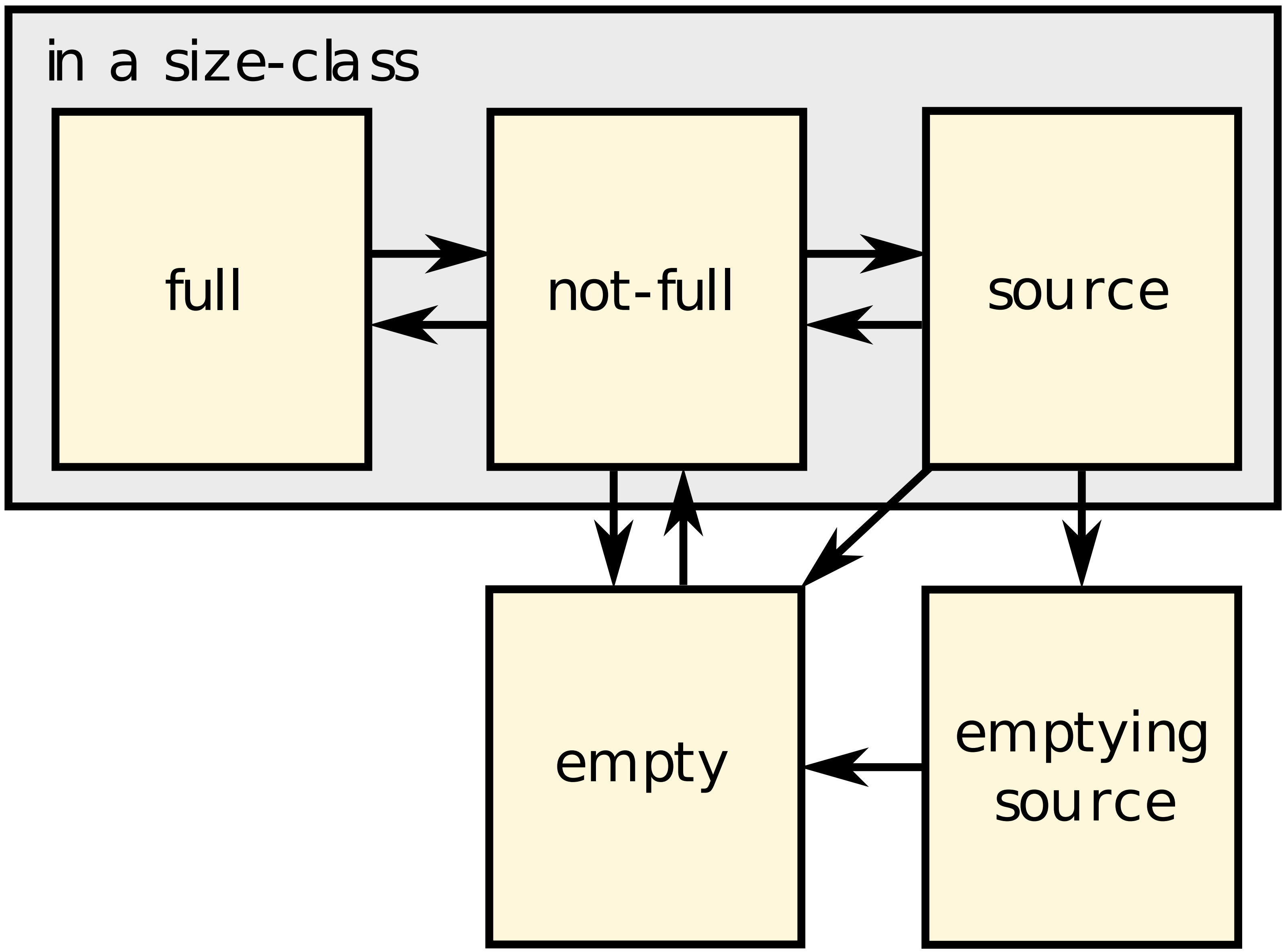
Incremental Compaction

- A page-block that is incrementally moved actually occupies two page-blocks:
  - source page-block
  - target page-block

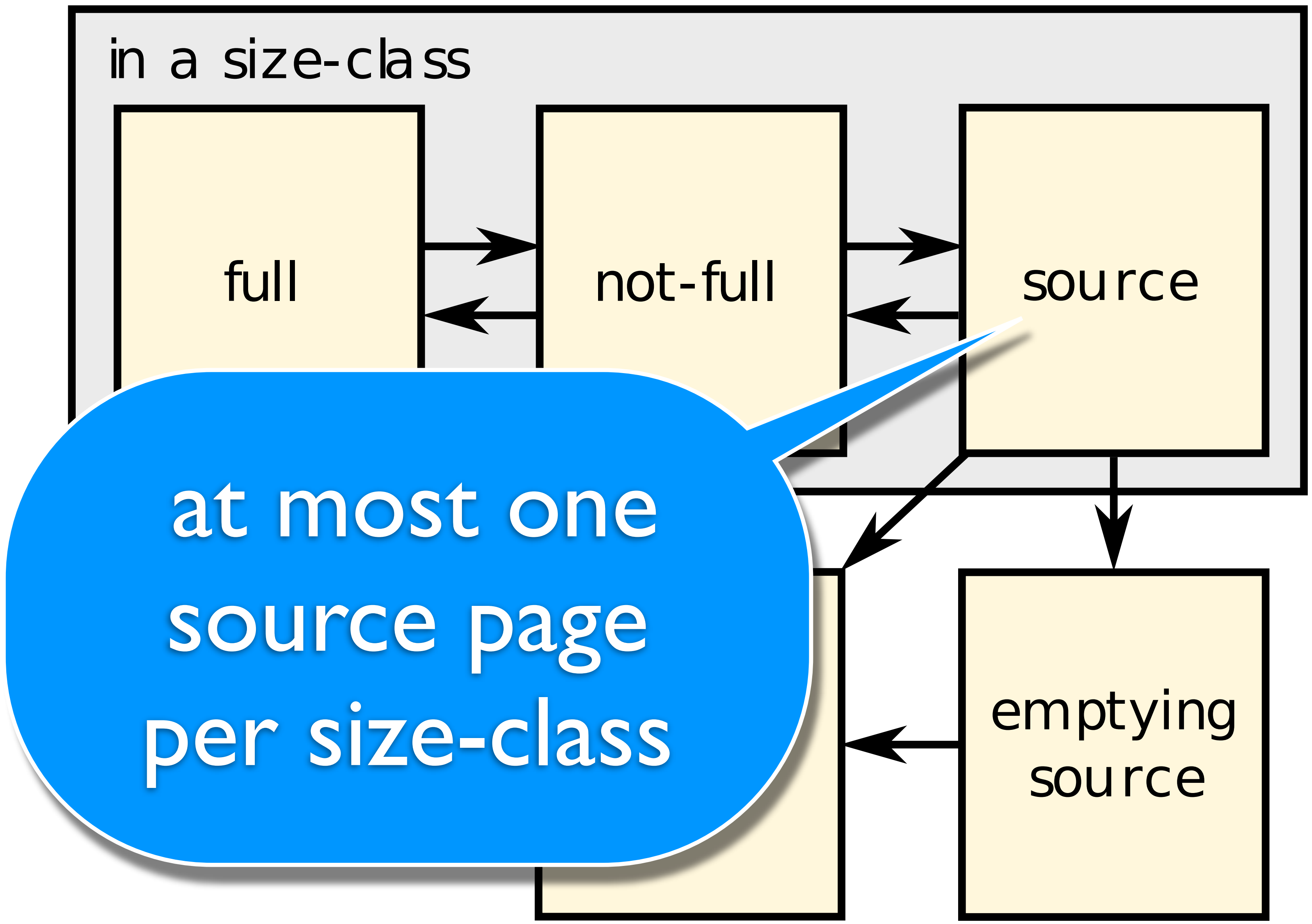
Incremental Compaction

- A page-block that is incrementally moved actually occupies two page-blocks:
  - source page-block
  - target page-block
- A page containing source page-blocks is called source page
  - may also contain used and free page-blocks

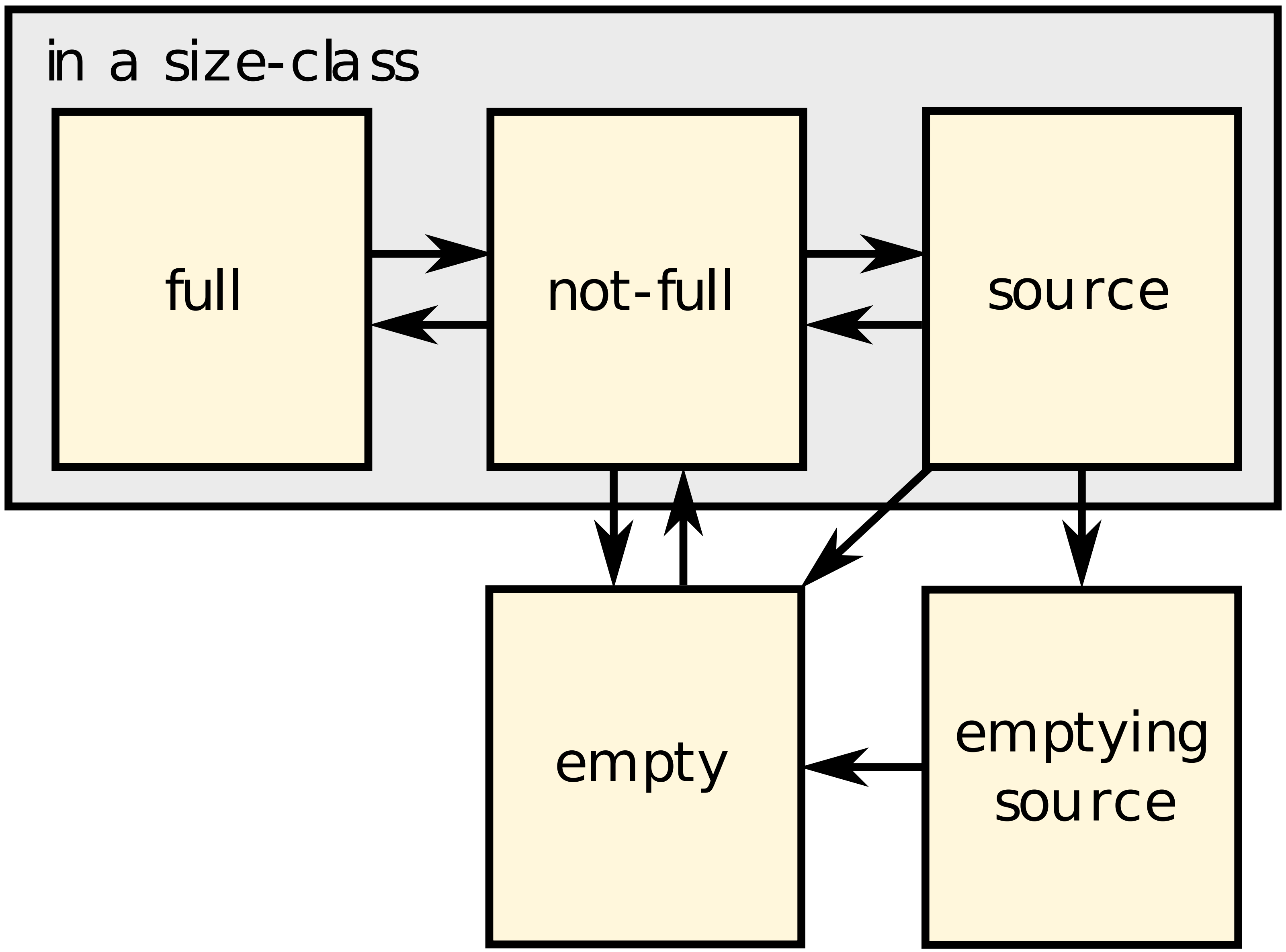
The Lifetime of a Page



The Lifetime of a Page

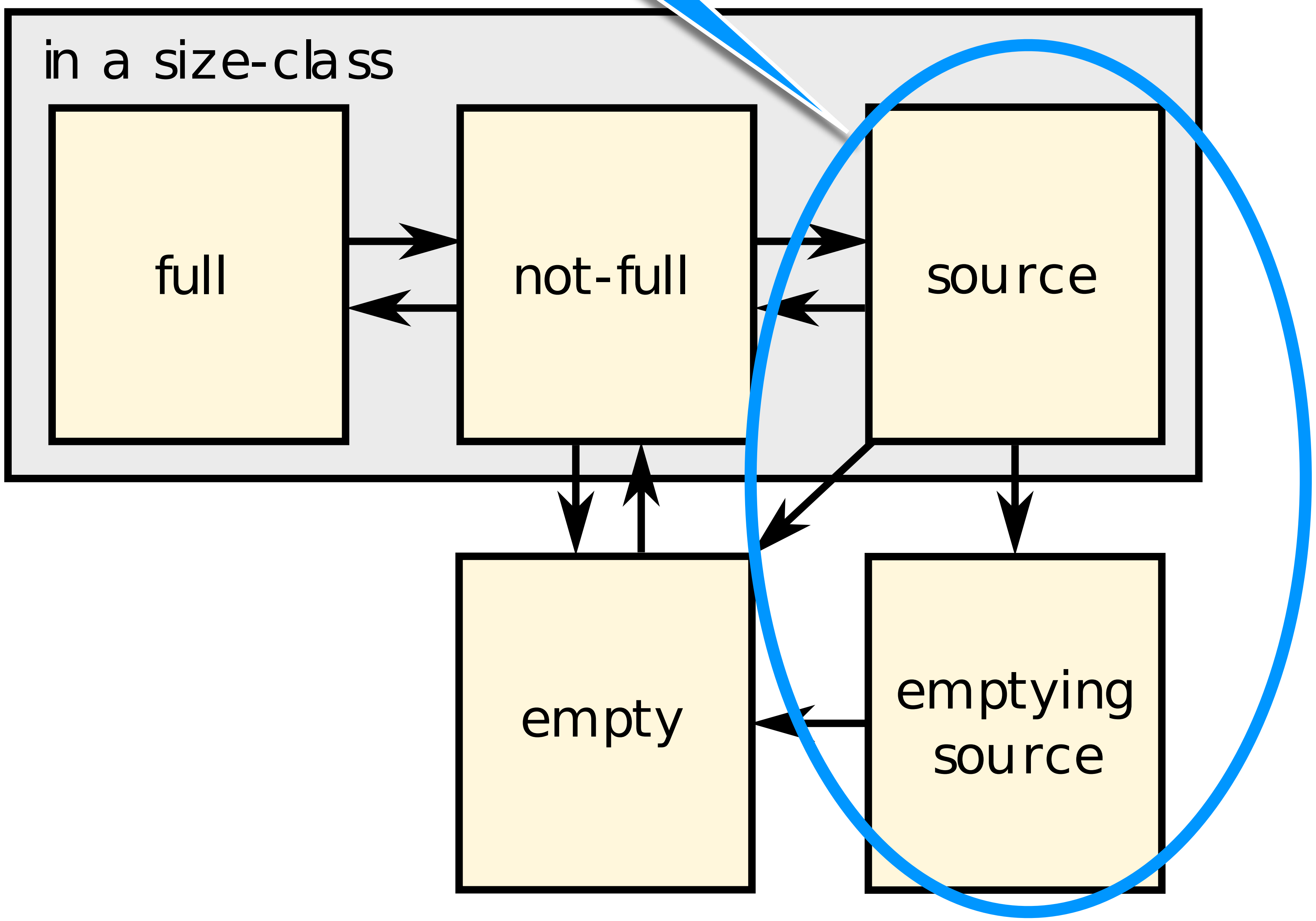


The Lifetime of a Page

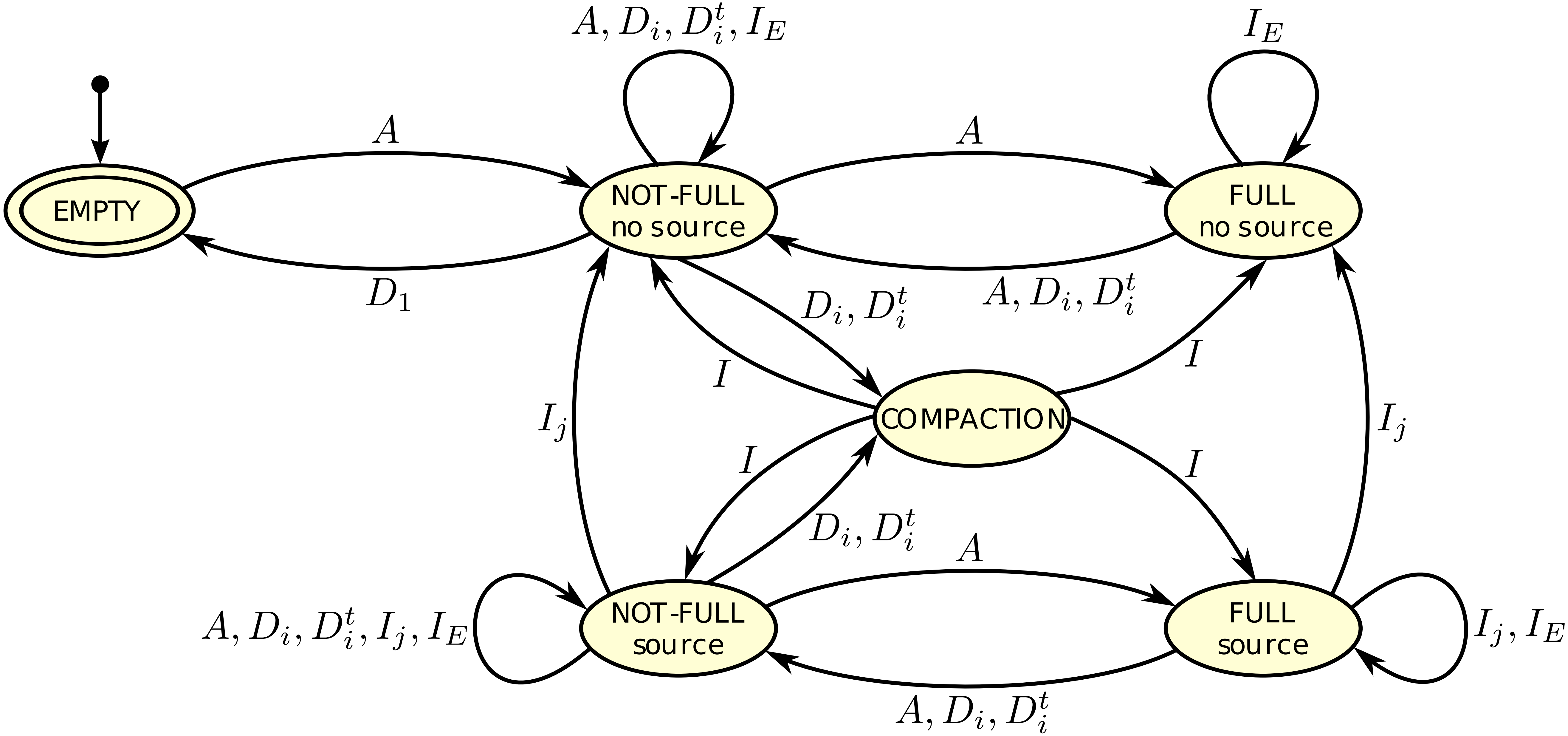


# of a Page

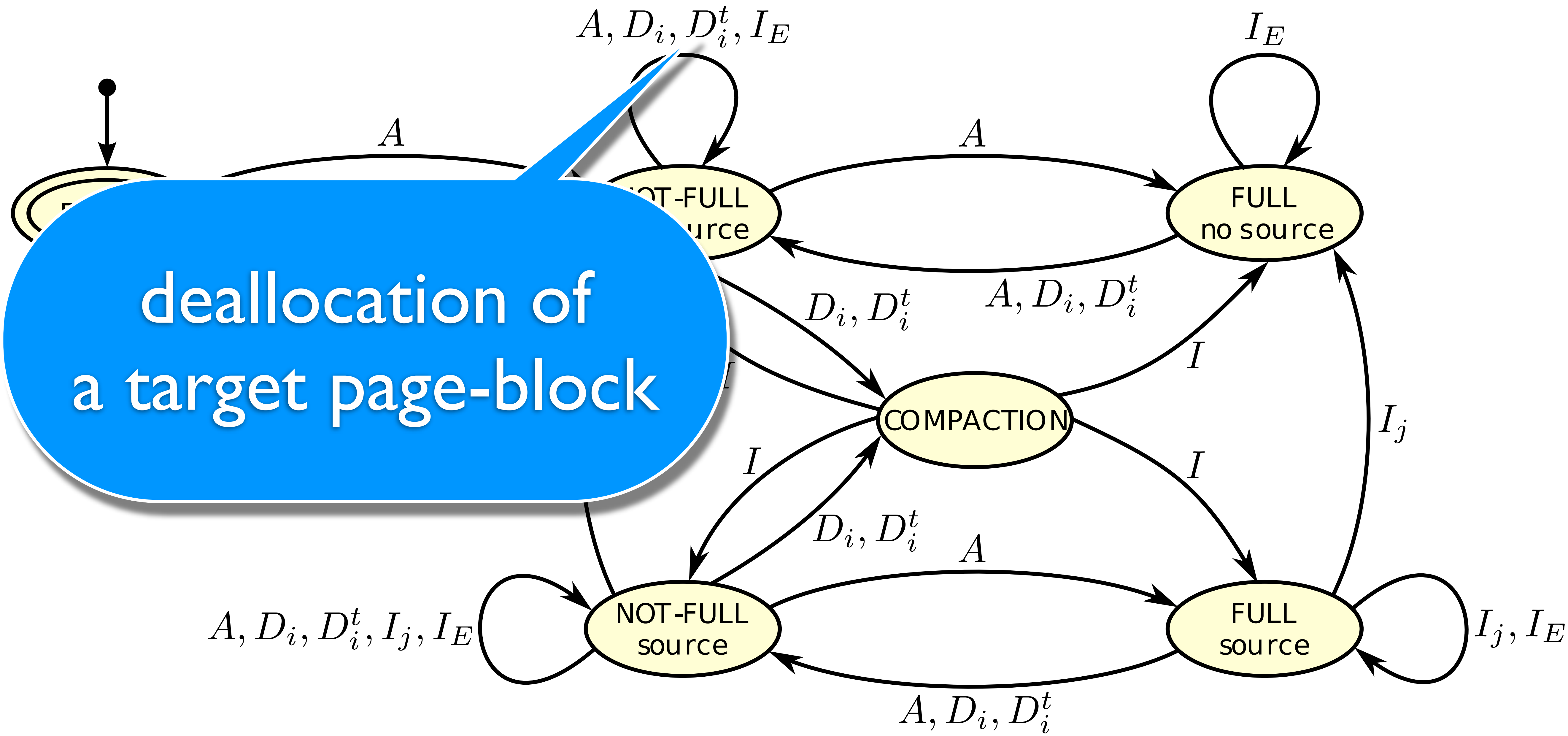
transient size-class fragmentation



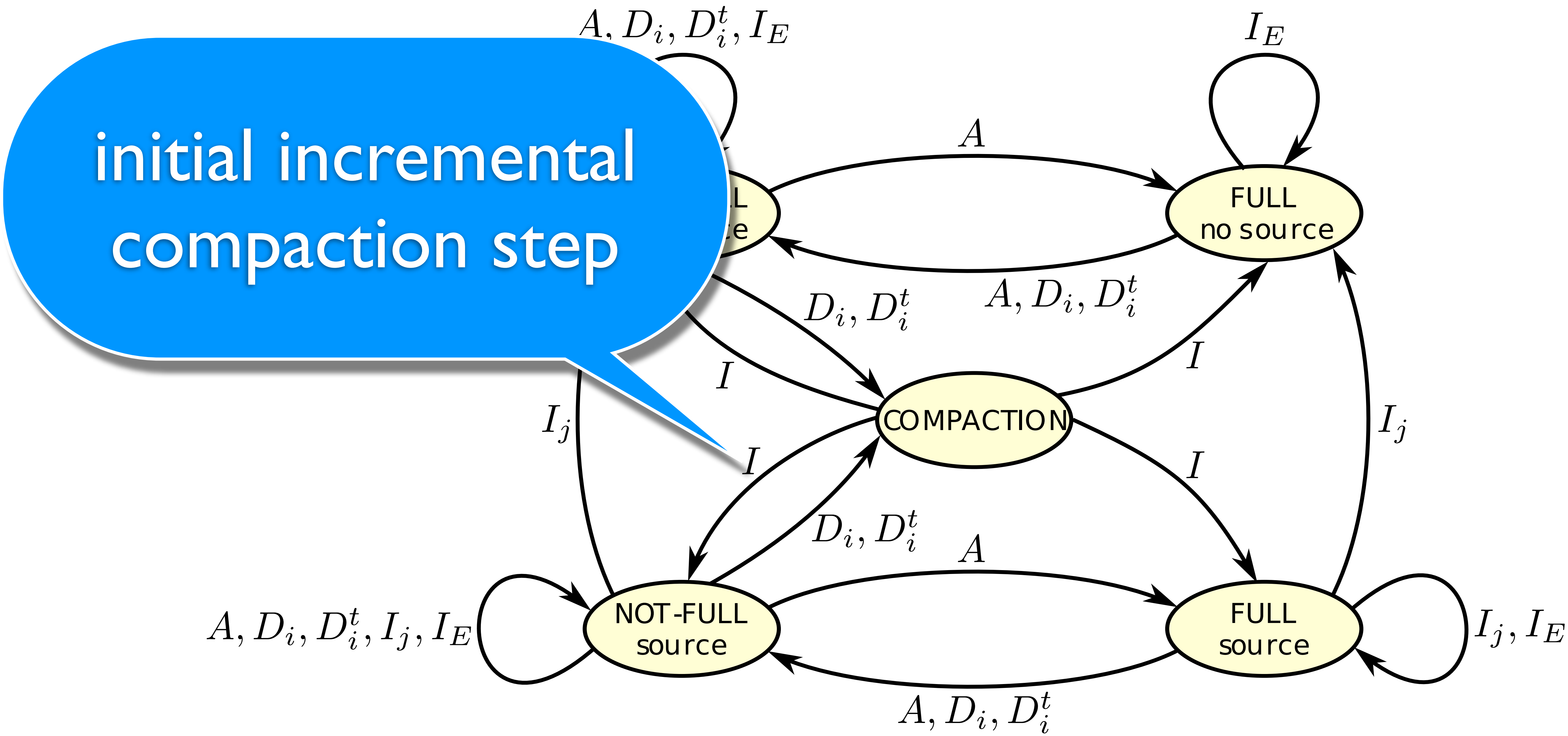
Incremental Size-Class Automaton for π > 1



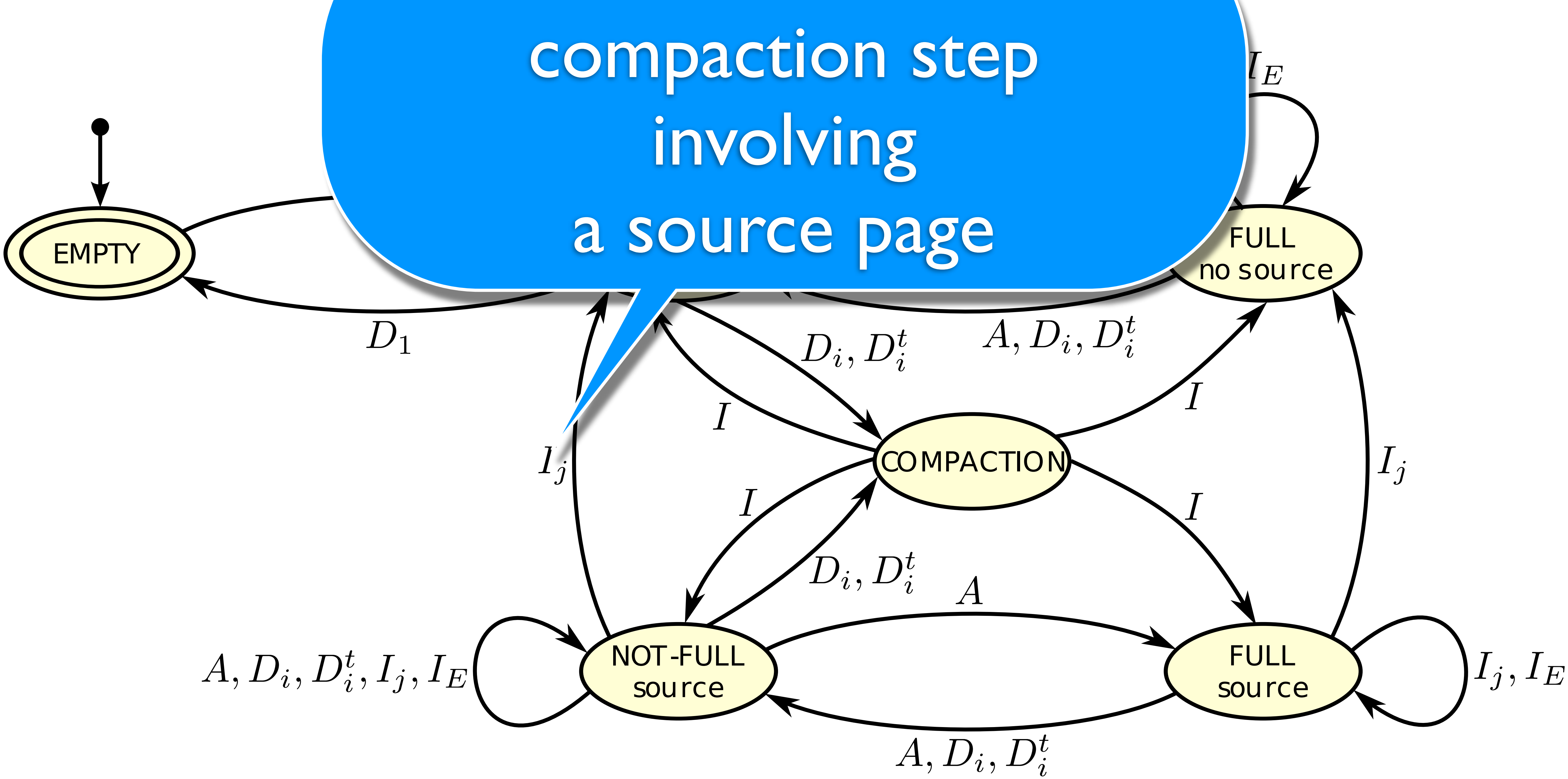
Incremental Size-Class Automaton for π > 1



Incremental Size-Class Automaton for π > 1

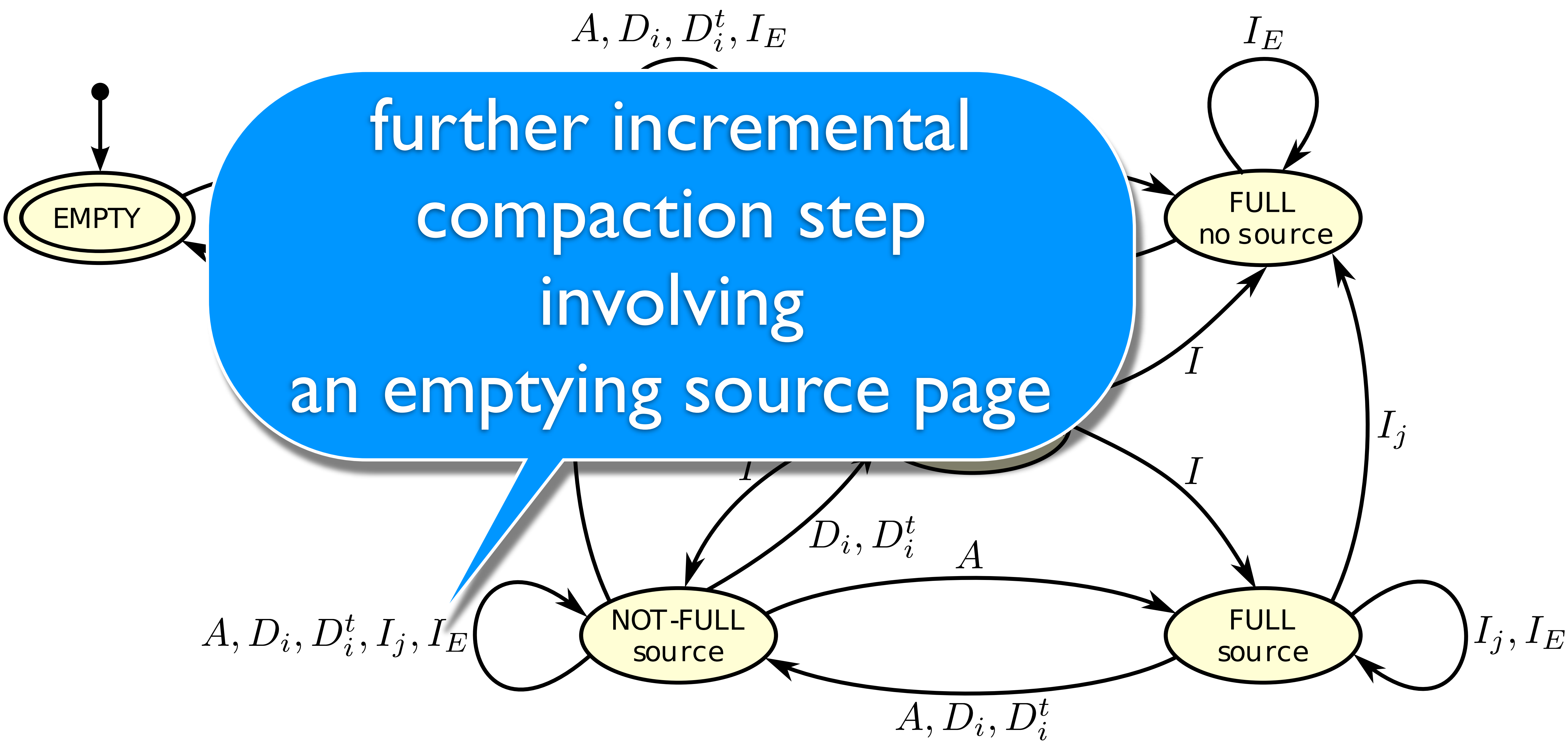


Incremental Size-Class Automaton for $\pi > 1$



further incremental compaction step involving a source page

Incremental Size-Class Automaton for π > 1



Temporal and Spatial Complexity per CF Configuration and Size-Class

	malloc	free	latency
$1\text{-CF}(\infty, \infty)$	$O(n)$	$O(n)$	$O(1)$
$1\text{-CF}(\kappa, \infty)$	$O(n)$	$O(n + \beta)$	$O(\beta)$
$n\text{-CF}(\infty, \infty)$	$O(1)$	$O(1)$	$O(1)$
$n\text{-CF}(\kappa, \infty)$	$O(1)$	$O(\beta)$	$O(\beta)$
$1\text{-CF}(\kappa, \iota)$	$O(n)$	$O(n + \beta + \lfloor \frac{\beta}{\iota} \rfloor)$	$O(\min(\beta, \iota))$

	memory size	size-class fragmentation
$1\text{-CF}(\infty, \infty)$	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
$1\text{-CF}(\kappa, \infty)$	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
$n\text{-CF}(\infty, \infty)$	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
$n\text{-CF}(\kappa, \infty)$	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
$1\text{-CF}(\kappa, \iota)$	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

Temporal and Spatial Complexity per CF Configuration and Size-Class

	malloc	free	latency
1-CF(∞, ∞)	$O(n)$	$O(n)$	$O(1)$
1-CF(κ, ∞)	$O(n)$	$O(n + \beta)$	$O(\beta)$
n-CF(∞, ∞)	$O(1)$	$O(1)$	$O(1)$
[illegible]	$O(1)$	$O(\beta)$	$O(\beta)$
[illegible]	[illegible]	[illegible]$(n + \beta + \lfloor \frac{\beta}{\iota} \rfloor)$	$O(\min(\beta, \iota))$

n is the # of threads

	[illegible] size	size-class fragmentation
1-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
1-CF(κ, ∞)	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
n-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
n-CF(κ, ∞)	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
1-CF(κ, ι)	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

Temporal and Spatial Complexity per CF Configuration and Size-Class

	malloc	free	latency
1-CF(∞, ∞)	$O(n)$	$O(n)$	$O(1)$
1-CF(κ, ∞)	$O(n)$	$O(n + \beta)$	$O(\beta)$
n-CF(∞, ∞)	$O(1)$	$O(1)$	$O(1)$
n-CF(κ, ∞)	$O(1)$	$O(\beta)$	$O(\beta)$
1-CF(κ, ι)	$O(n)$	$O(n + \beta + \lfloor \frac{\beta}{\iota} \rfloor)$	$O(\min(\beta, \iota))$

	memory size	size-class fragmentation
1-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
1-CF(κ, ∞)	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
n-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
n-CF(κ, ∞)	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
1-CF(κ, ι)	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

Temporal and Spatial Complexity per CF Configuration and Size-Class

	malloc	free	latency
1-CF(∞, ∞)	$O(n)$	$O(n)$	$O(1)$
1-CF(κ, ∞)	$O(n)$	$O(n + \beta)$	$O(\beta)$
n-CF(∞, ∞)	$O(1)$	$O(1)$	$O(1)$
n-CF(κ, ∞)	$O(1)$	$O(\beta)$	[illegible]
1-CF(κ, ι)	$O(n)$	O([illegible]	[illegible]

β is the page-block size

	memory size	[illegible]ion
1-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
1-CF(κ, ∞)	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
n-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
n-CF(κ, ∞)	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
1-CF(κ, ι)	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

Temporal and Spatial Complexity per CF Configuration and Size-Class

	malloc	free	latency
1-CF(∞, ∞)	$O(n)$	$O(n)$	$O(1)$
1-CF(κ, ∞)	$O(n)$	$O(n + \beta)$	$O(\beta)$
n-CF(∞, ∞)	$O(1)$	$O(1)$	$O(1)$
n-CF(κ, ∞)	$O(1)$	$O(\beta)$	$O(\beta)$
1-CF(κ, ι)	$O(n)$	$O(n + \beta + \lfloor \frac{\beta}{\iota} \rfloor)$	$O(\min(\beta, \iota))$

	memory size	size-class fragmentation
1-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
1-CF(κ, ∞)	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
n-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
n-CF(κ, ∞)	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
1-CF(κ, ι)	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

Temporal and Spatial Complexity per CF Configuration and Size-Class

		latency
1-CF		$O(1)$
1-C		$O(\beta)$
n-C		$O(1)$
n-CF		$O(\beta)$
1-CF(κ, ι)		$O(\min(\beta, \iota))$

m is the # of per-thread-allocated page-blocks

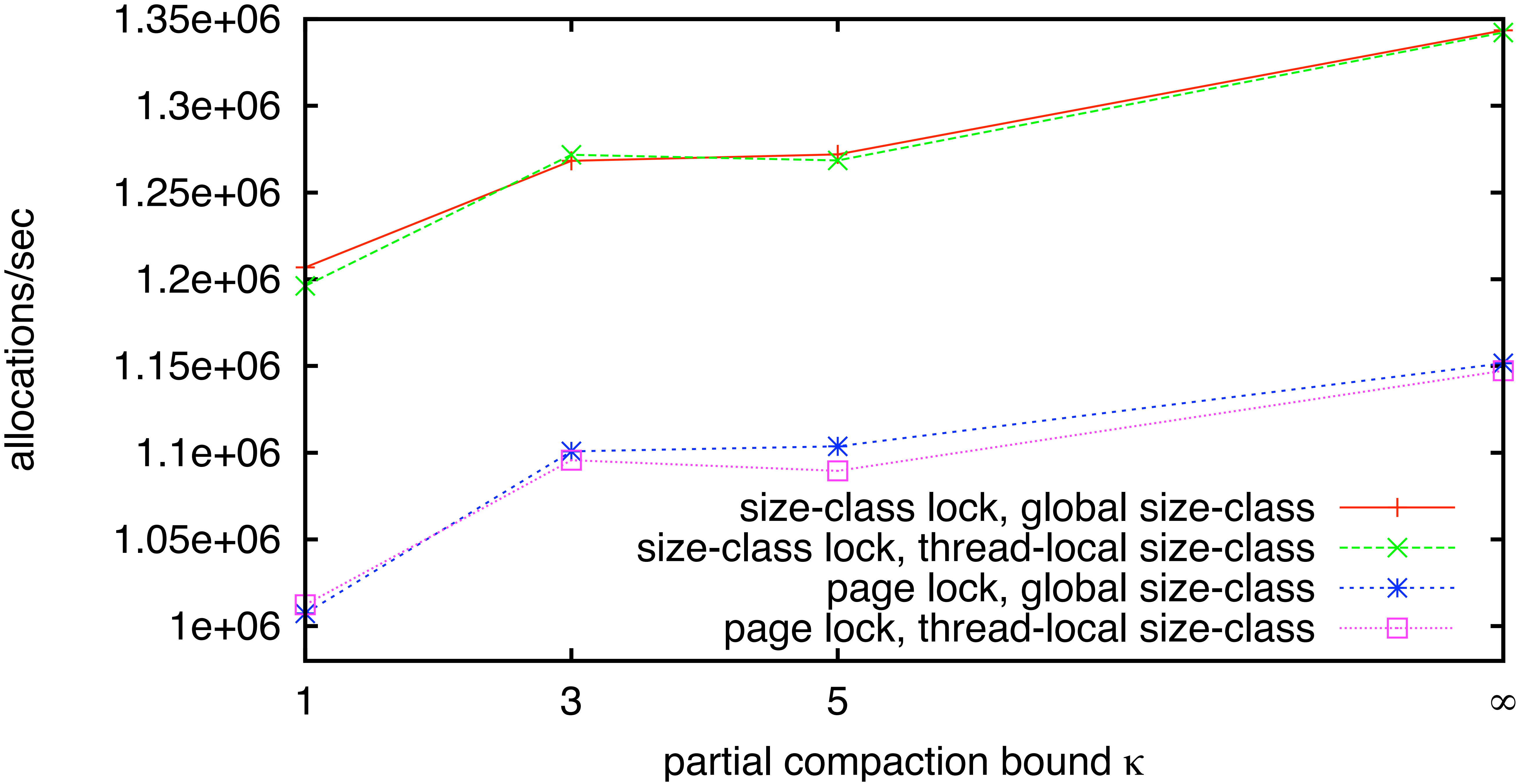
	memory size	size-class fragmentation
1-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
1-CF(κ, ∞)	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
n-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
n-CF(κ, ∞)	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
1-CF(κ, ι)	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

Temporal and Spatial Complexity per CF Configuration and Size-Class

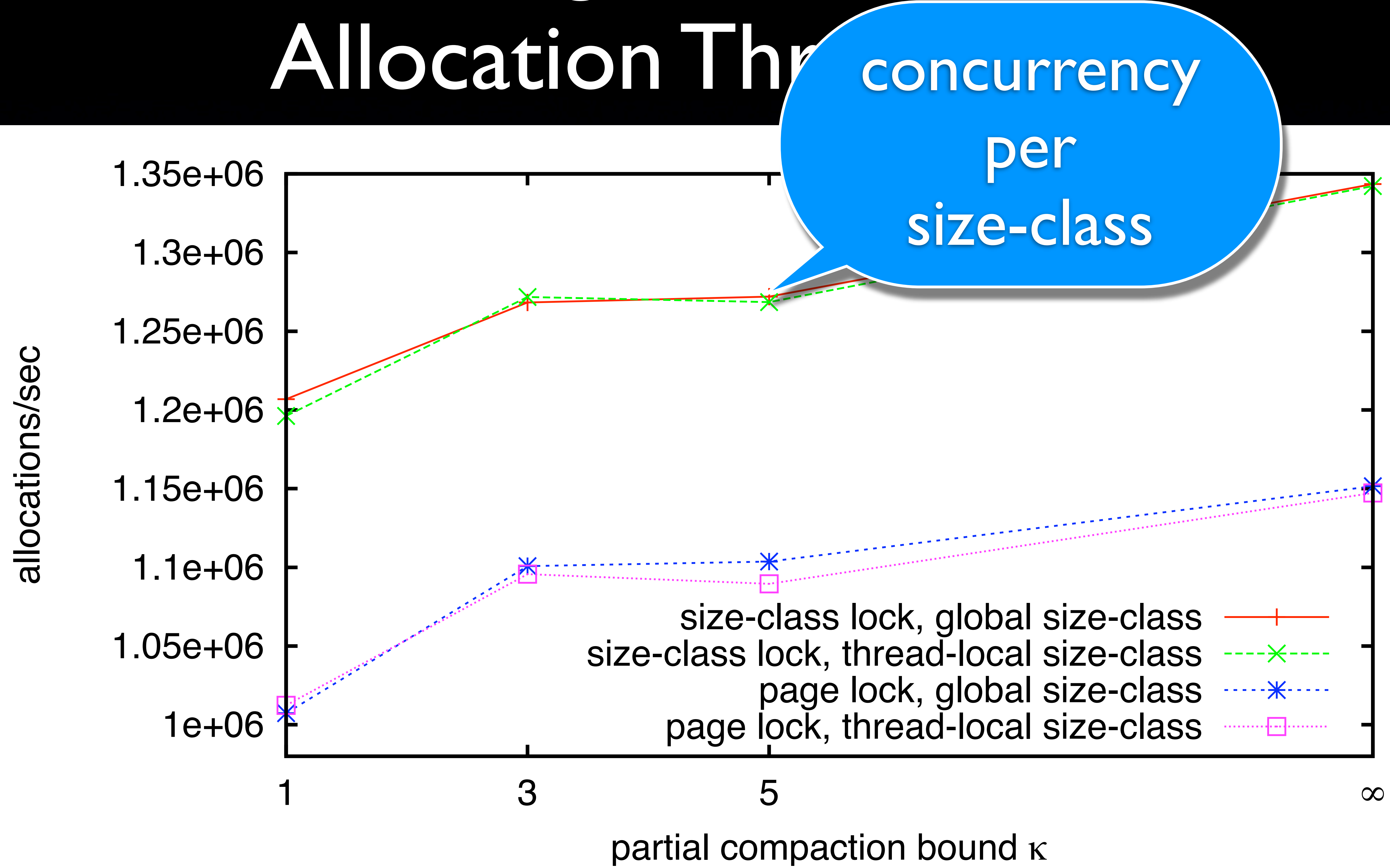
	malloc	free	latency
1-CF(∞, ∞)	$O(n)$	$O(n)$	$O(1)$
1-CF(κ, ∞)	$O(n)$	$O(n + \beta)$	$O(\beta)$
n-CF(∞, ∞)	$O(1)$	$O(1)$	$O(1)$
n-CF(κ, ∞)	$O(1)$	$O(\beta)$	$O(\beta)$
1-CF(κ, ι)	$O(n)$	$O(n + \beta + \lfloor \frac{\beta}{\iota} \rfloor)$	$O(\min(\beta, \iota))$

	memory size	size-class fragmentation
1-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
1-CF(κ, ∞)	$O((n * m + \kappa * (\pi - 1)) * \beta)$	$O(\kappa * (\pi - 1) * \beta)$
n-CF(∞, ∞)	$O(n * m * \pi * \beta)$	$O(n * m * (\pi - 1) * \beta)$
n-CF(κ, ∞)	$O(n * (m + \kappa * (\pi - 1)) * \beta)$	$O(n * \kappa * (\pi - 1) * \beta)$
1-CF(κ, ι)	$O((n * m + n * \pi + \kappa * (\pi - 1)) * \beta)$	$O((n * \pi + \kappa * (\pi - 1)) * \beta)$

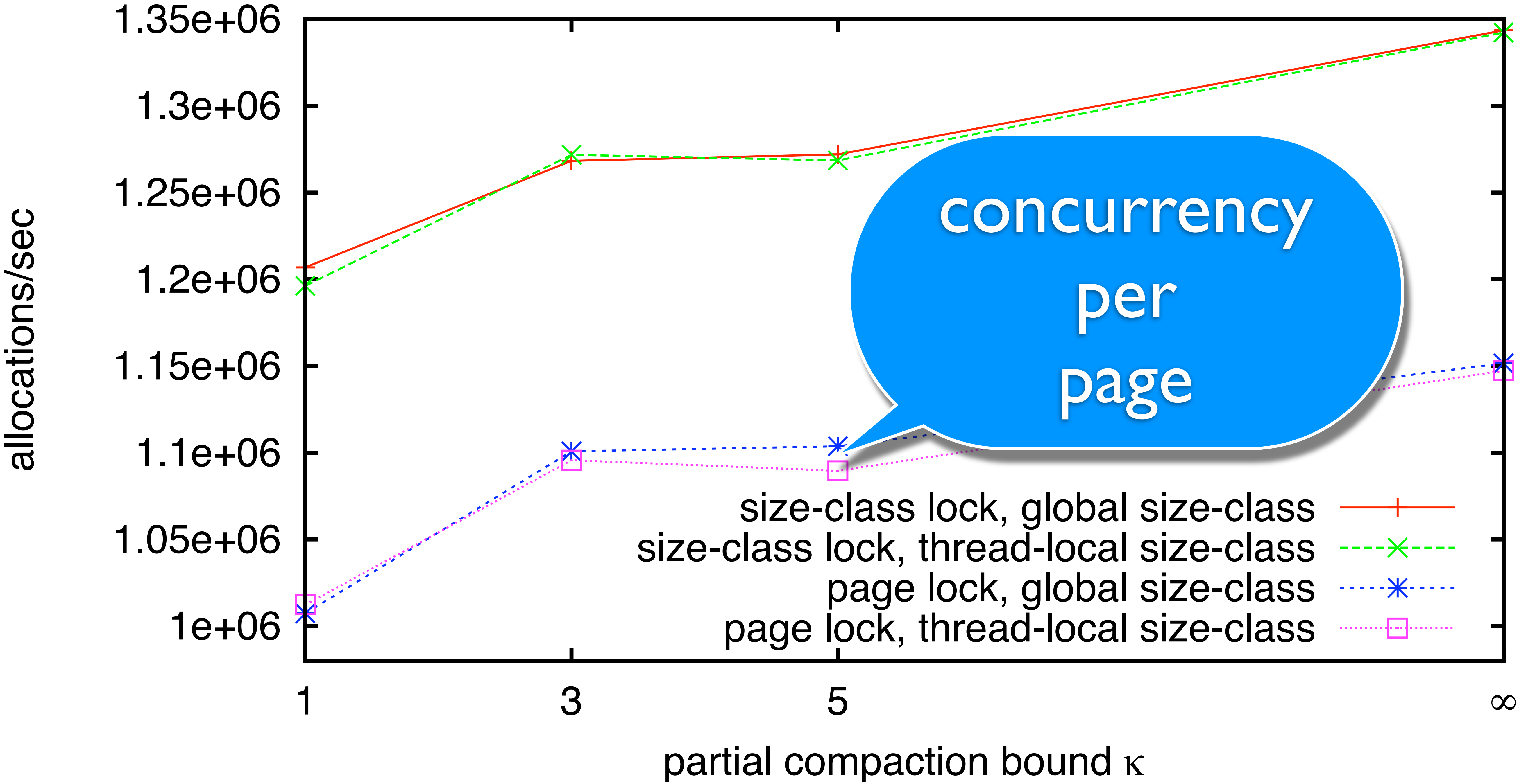
Single Thread Allocation Throughput



Single Thread Allocation Thr

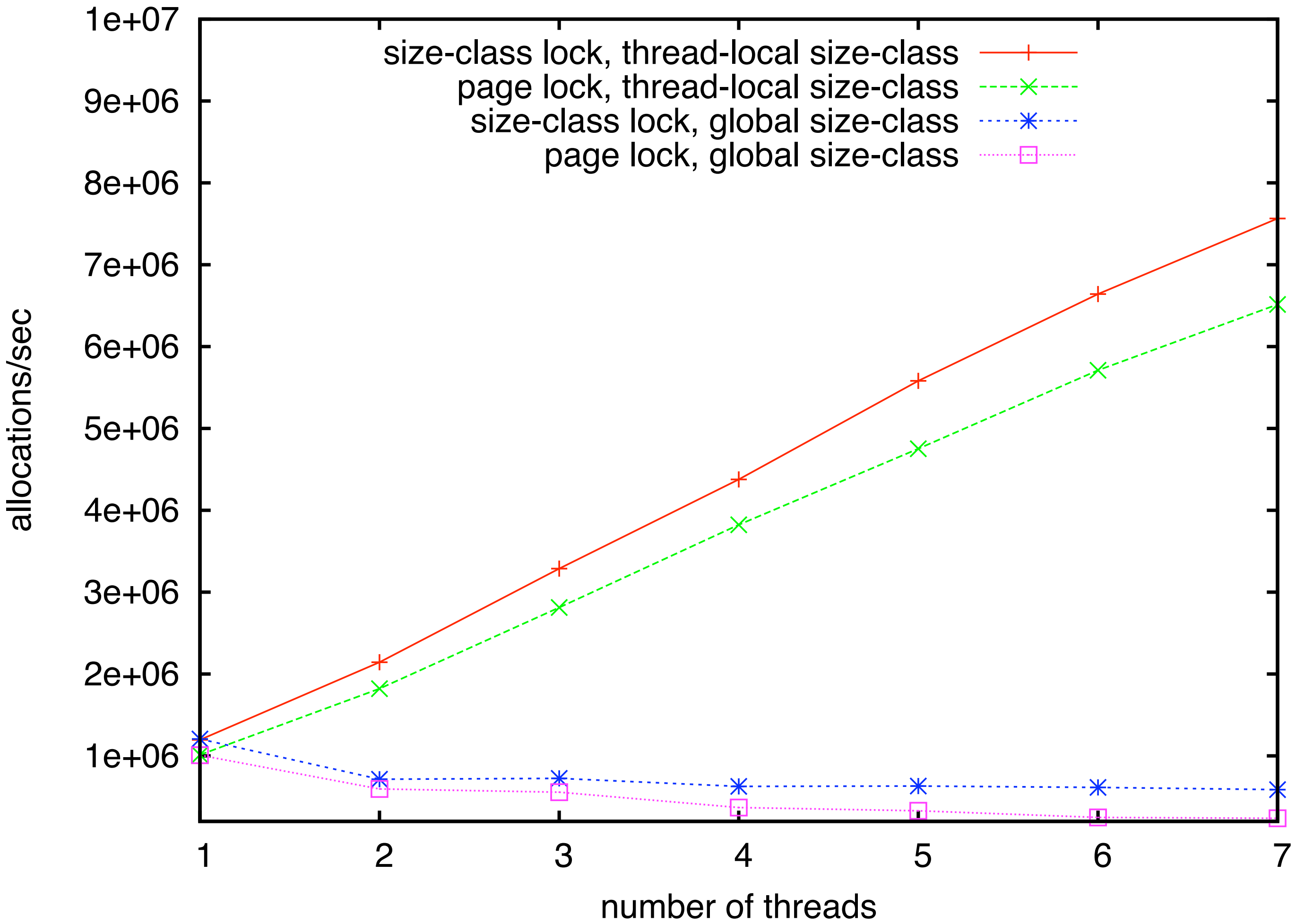


Single Thread Allocation Throughput

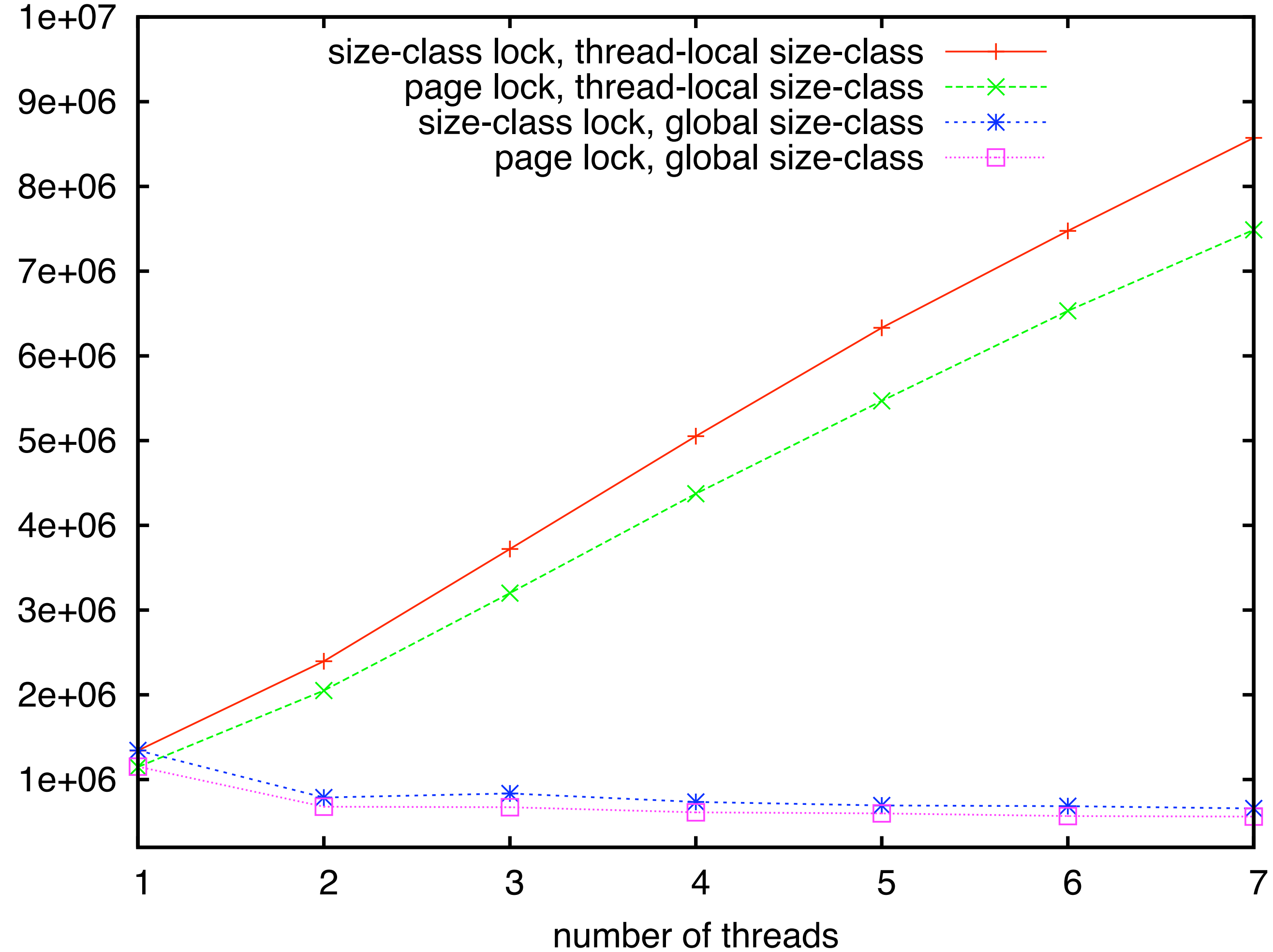


- less compaction may result in better allocation throughput
- size-class locks better than page locks

Scalability of Allocation Throughput

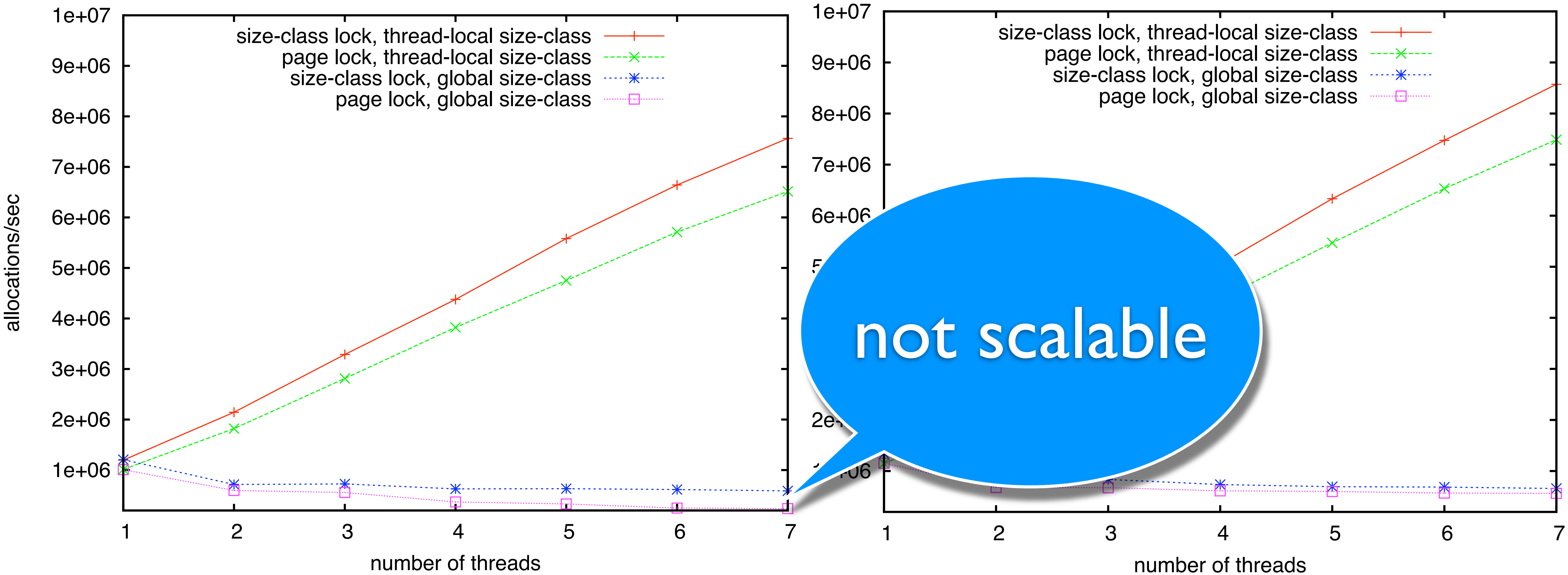


(a) full compaction



(b) optimized, non-compacting

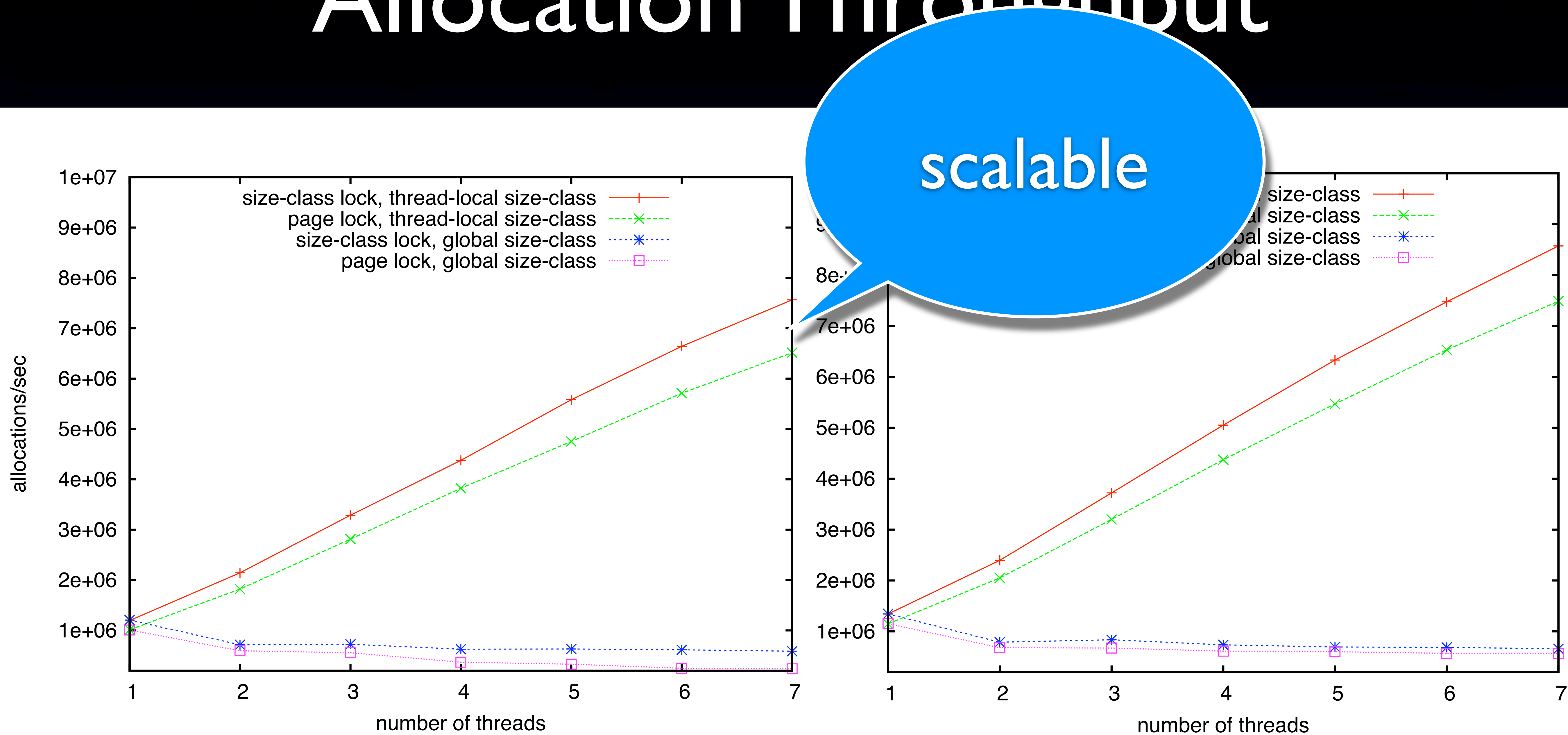
Scalability of Allocation Throughput



(a) full compaction

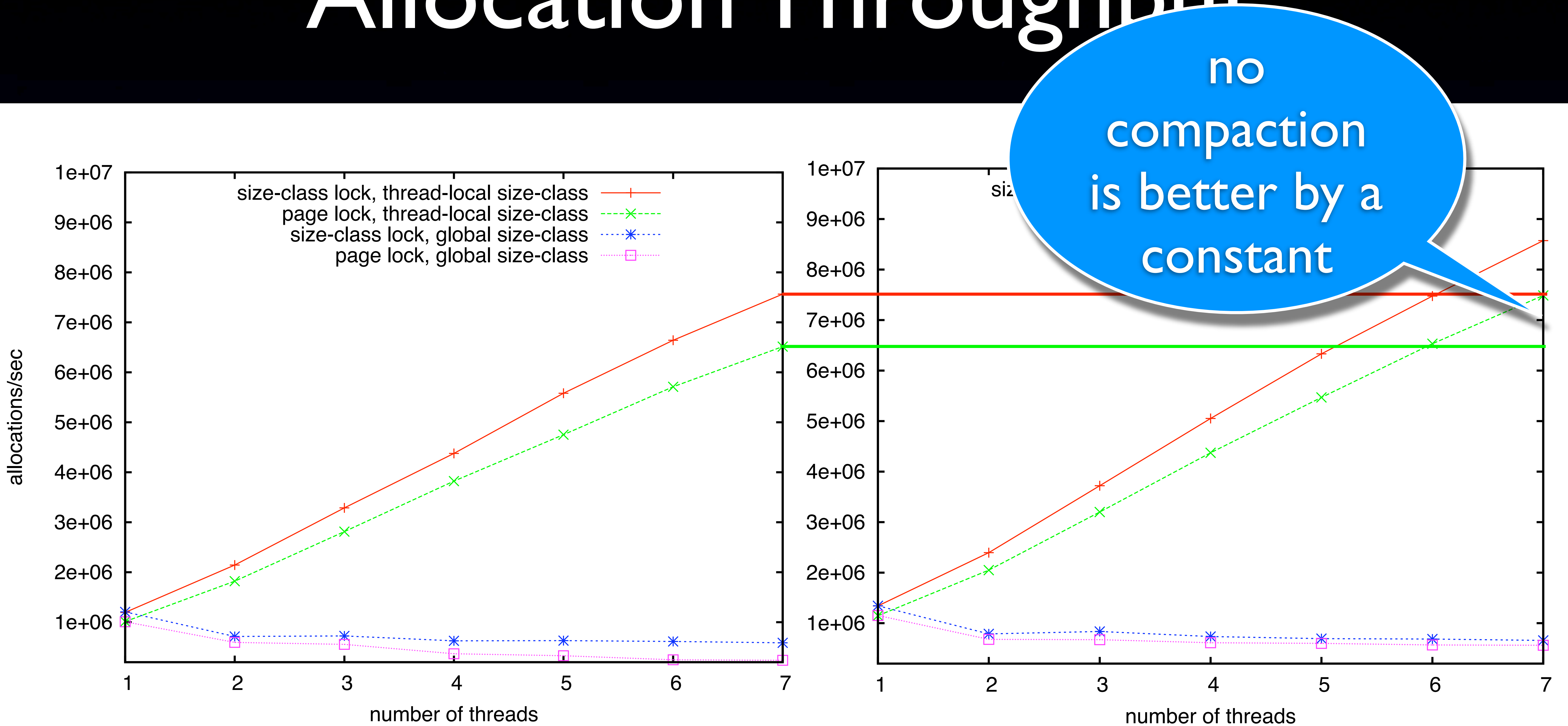
(b) optimized, non-compacting

Scalability of Allocation Throughput



(a) full compaction (b) optimized, non-compacting

Scalability of Allocation Throughput

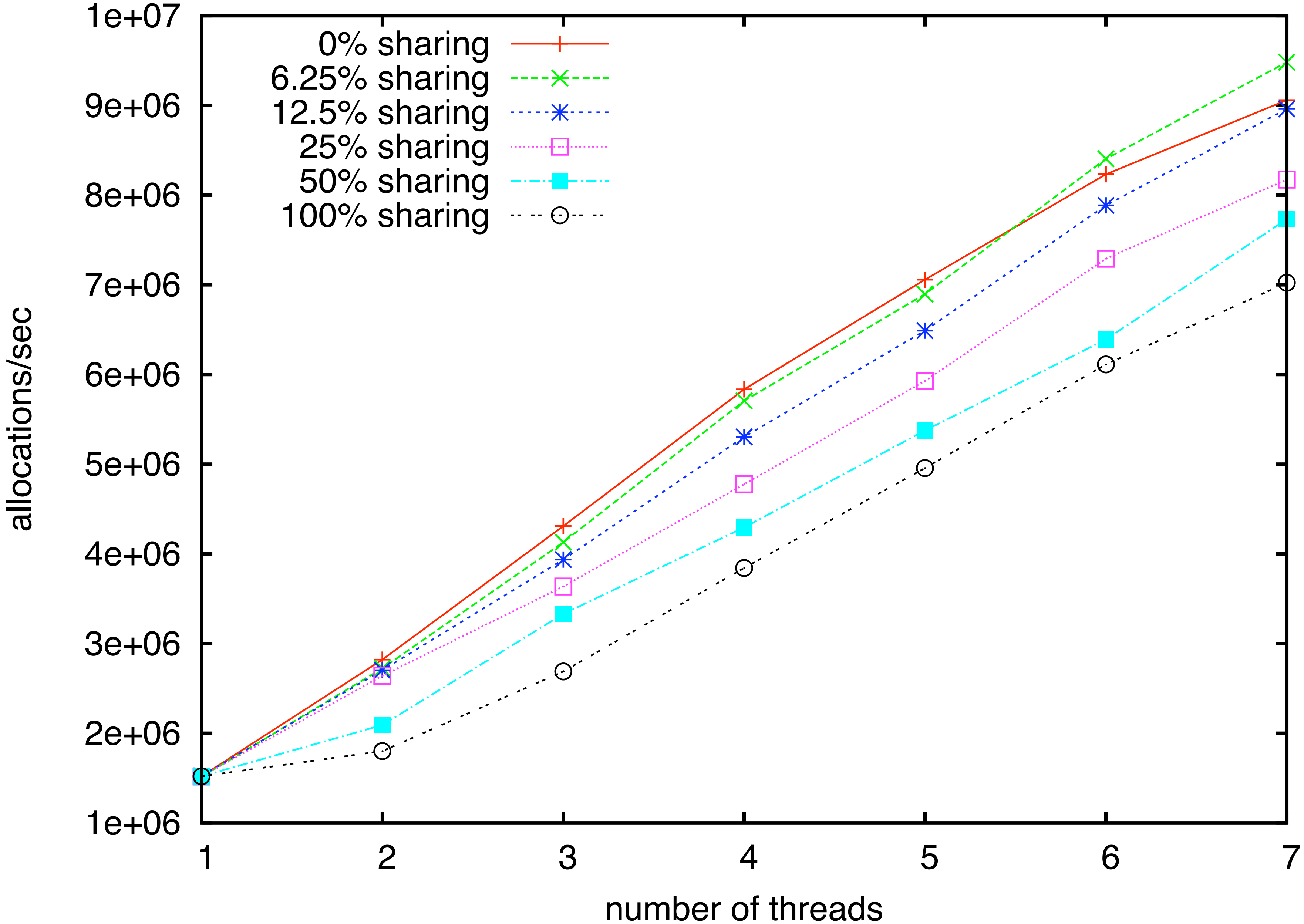


(a) full compaction

(b) optimized, non-compacting

- global size-class locks do not scale
- full compaction only requires constant factor

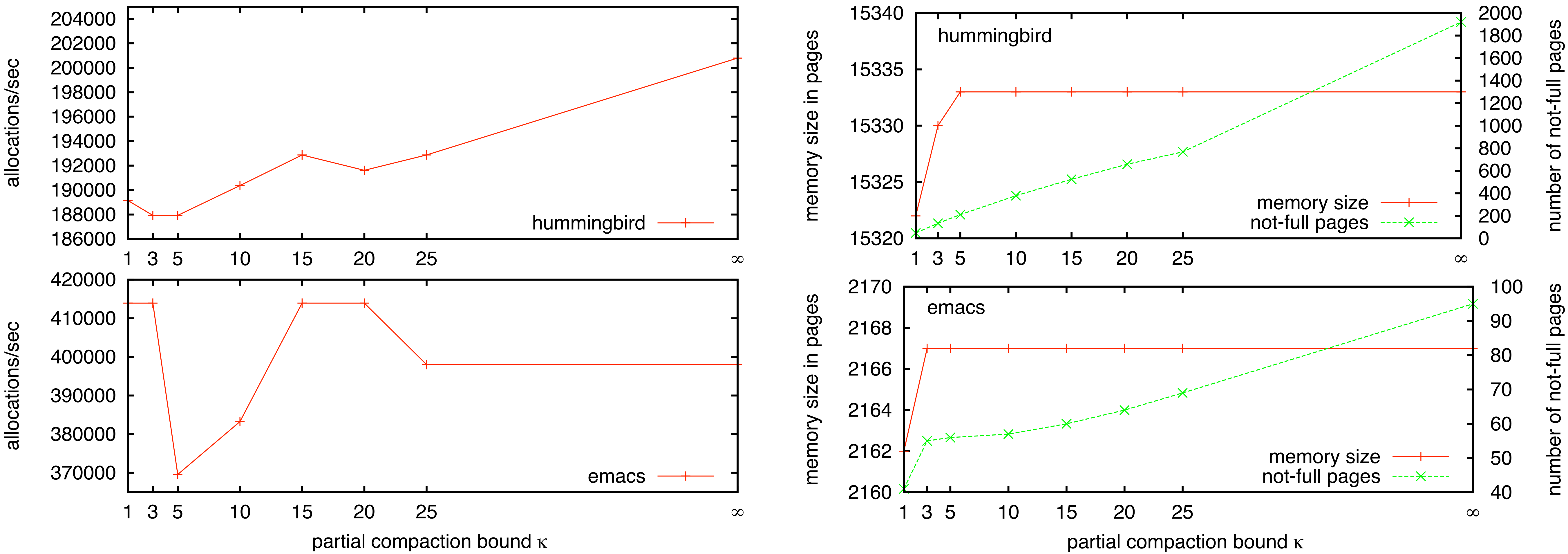
Scalability of Allocation Throughput



(c) opt., non-comp. with sharing

- level of sharing determines scalability

Real Application Performance



- less compaction may result in better allocation throughput
- size-class fragmentation increases with less compaction but total memory consumption may not

TLSF vs. opt., non-comp. CF Performance

	memory (in MB)				
	TLSF	CF (16B blocks)		CF (32B blocks)	
	memory size	memory size	size-class fragmentation	memory size	size-class fragmentation
Emacs	25.7	34.6	0.46	34.5	0.38
Hummingbird	203.7	245.3	8.3	245.9	11.4

	malloc (in clock ticks)				free (in clock ticks)			
	TLSF		CF		TLSF		CF	
	avg time	max time	avg time	max time	avg time	max time	avg time	max time
Emacs	228	93359	260	81662	153	71159	279	74798
Hummingbird	411	109079	529	98820	500	69192	574	79914

TLSF vs. opt., non-comp. CF Performance

only 1.3%
of the 35% more
memory

	memo[illegible]				
	TLSF	CF (16B bloc[illegible]		[illegible]cks)	
	memory size	memory size	size-cl[illegible] fragment[illegible]	[illegible]	[illegible]-class [illegible]gmentation
Emacs	25.7	34.6	0.46	34.5	0.38
Hummingbird	203.7	245.3	8.3	245.9	11.4

	malloc (in clock ticks)				free (in clock ticks)			
	TLSF		CF		TLSF		CF	
	avg time	max time	avg time	max time	avg time	max time	avg time	max time
Emacs	228	93359	260	81662	153	71159	279	74798
Hummingbird	411	109079	529	98820	500	69192	574	79914

TLSF vs. opt., non-comp. CF Performance

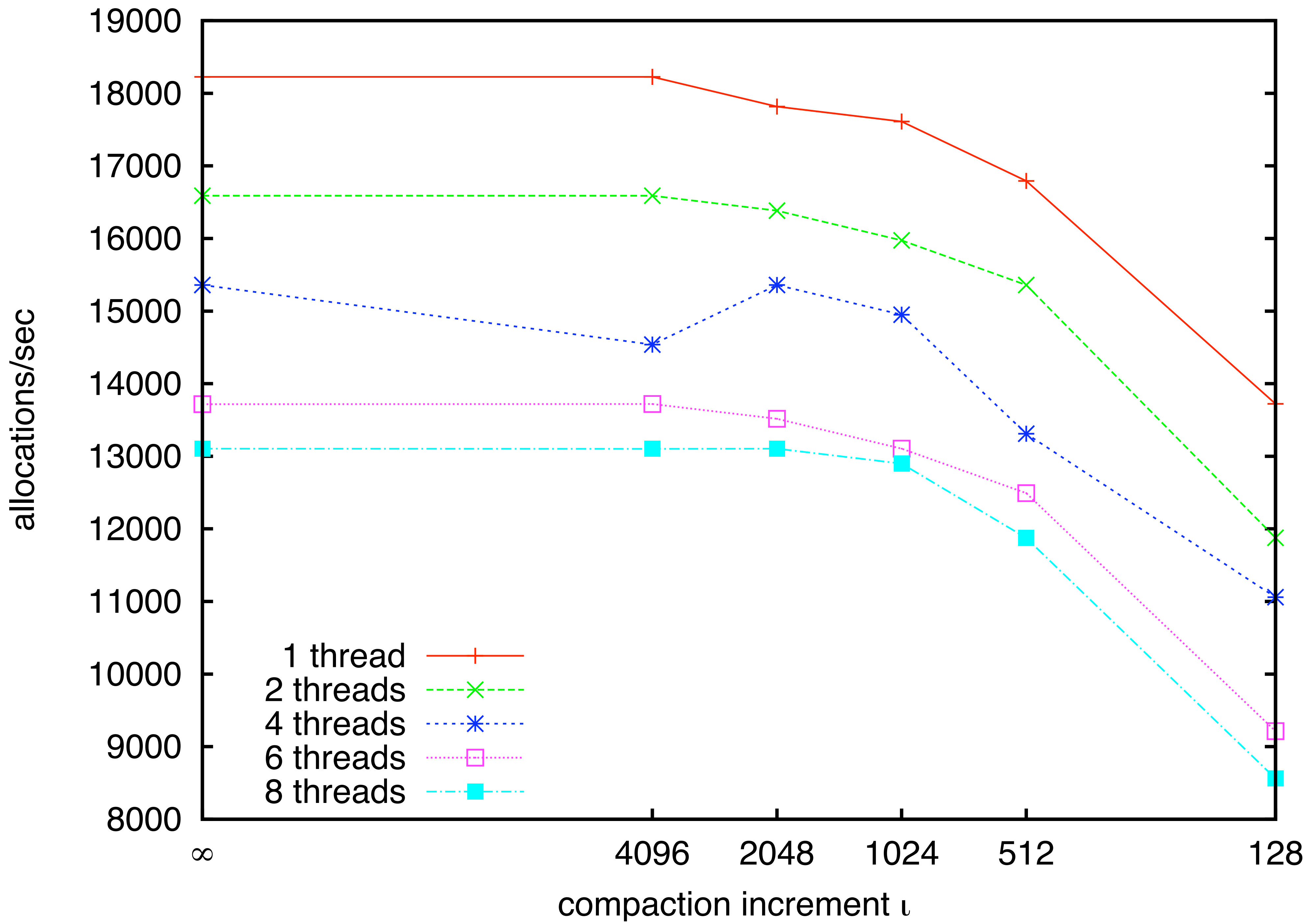
	memory (in MB)				
	TLSF	CF (16B blocks)		CF (32B blocks)	
	memory size	memory size	size-class fragmentation	memory size	size-class fragmentation
Emacs	25.7	34.6	0.46	34.5	0.38
Hummingbird	203.7	245.3	8.3	245.9	11.4

	malloc (in clock ticks)				ks)			
	TLSF		CF		[illegible]		[illegible]	
	avg time	max time	avg time	ma tim	[illegible]	[illegible]	[illegible]	max time
Emacs	228	93359	260	81662	153	71159	279	74798
Hummingbird	411	109079	529	98820	500	69192	574	79914

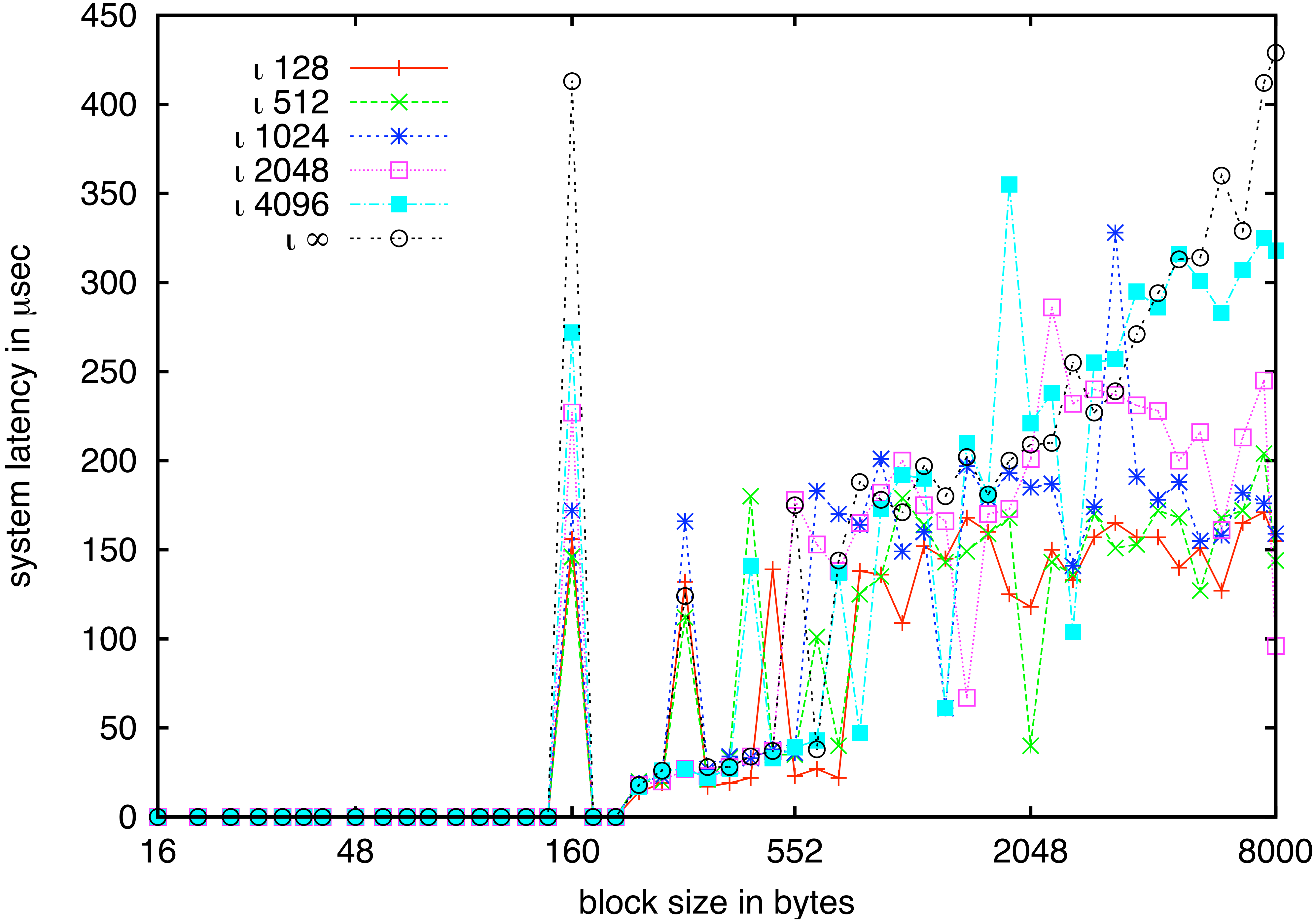
sometimes even better than TLSF

- fragmentation through partitioning dominates CF memory consumption
- opt., non-comp. CF only slightly slower than TLSF

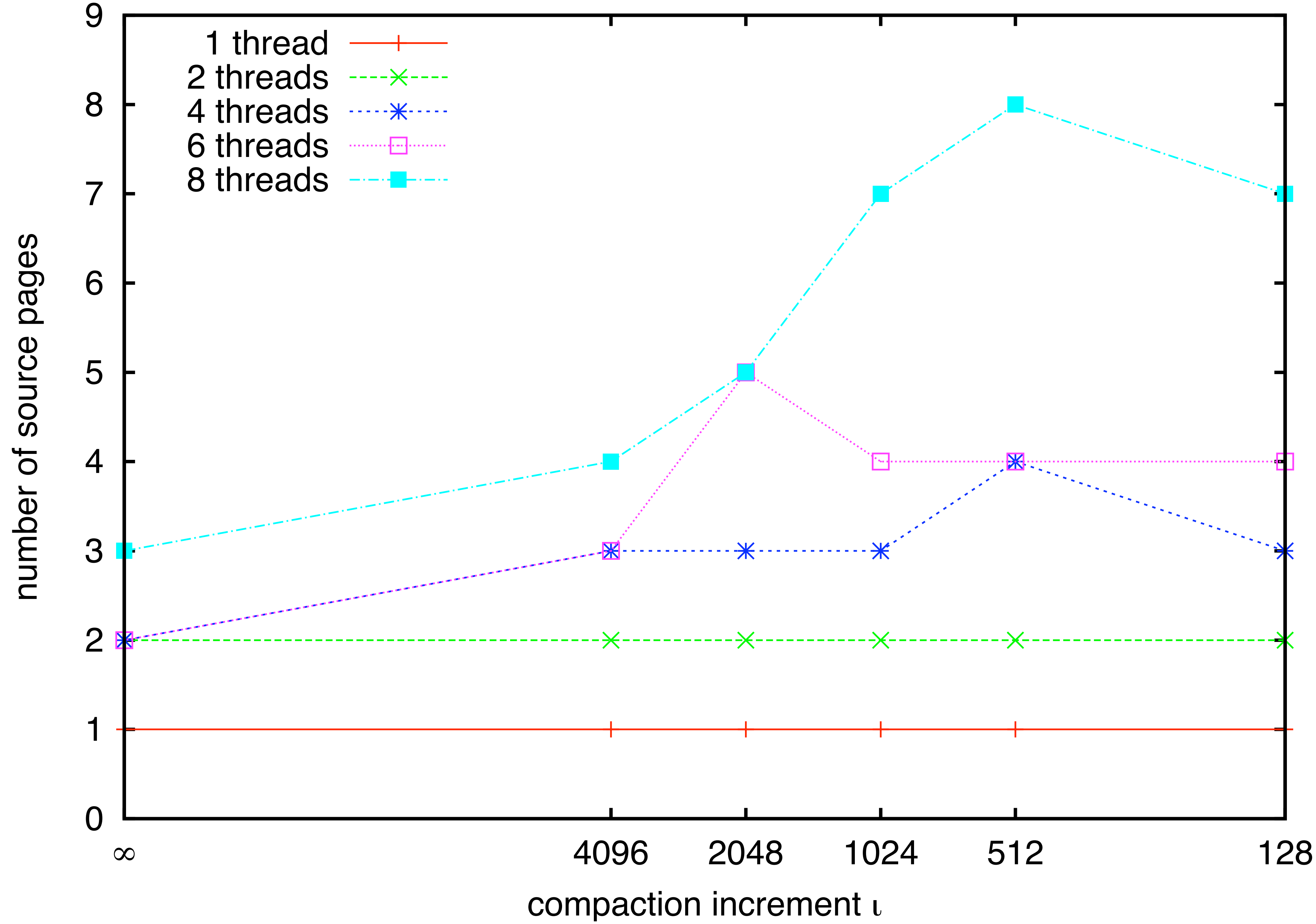
Allocation Throughput with Decreasing Compaction Increment



System Latency with 8 Threads and Increasing Block Size



Transient Size-Class Fragmentation with Decreasing Compaction Increment




Thank you