Compacting the Uncompactable!

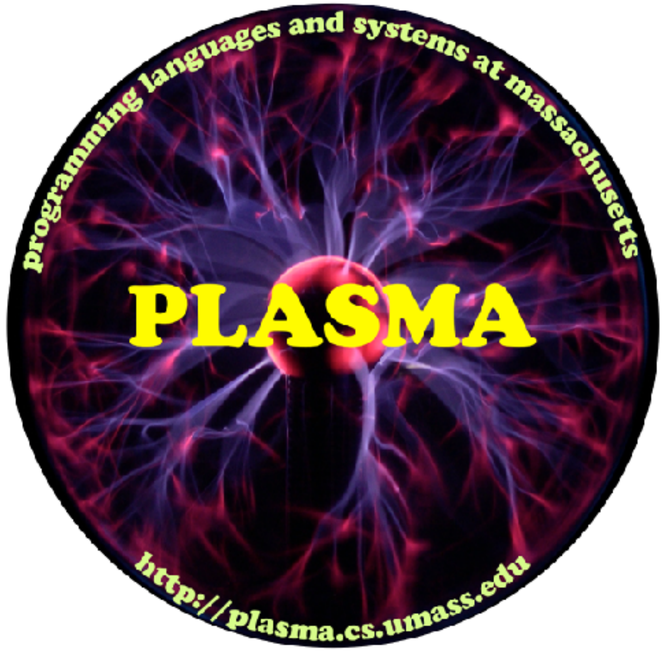
MESH

Automatically Compacting
Your C++ Application's Memory

Emery Berger
UMass Amherst

with Bobby Powers, David Tench, & Andrew McGregor
University of Massachusetts Amherst
http://libmesh.org

[PLDI 2019]



PIANO • VOCAL • GUITAR

The Impossible Dream
(The Quest)

Lyrics by JOE DARIEN
Music by MITCH LEIGH

PIANO · VOCAL · GUITAR

The Impossible Dream
(The Quest)

Lyrics by JOE DARIEN
Music by MITCH LEIGH

Reconquer
all of Spain!

PIANO · VOCAL · GUITAR

The Impossible Dream
(The Quest)

Lyrics by JOE DARIEN
Music by MITCH LEIGH

MALLOC

Reconquer
all of Spain!

MALLOC

~~Reconquer~~ Allocate all of Spain!

PIANO · VOCAL · GUITAR

The Impossible Dream
(The Quest)

Lyrics by JOE DARIEN
Music by MITCH LEIGH

MALLOC

PIANO · VOCAL · GUITAR

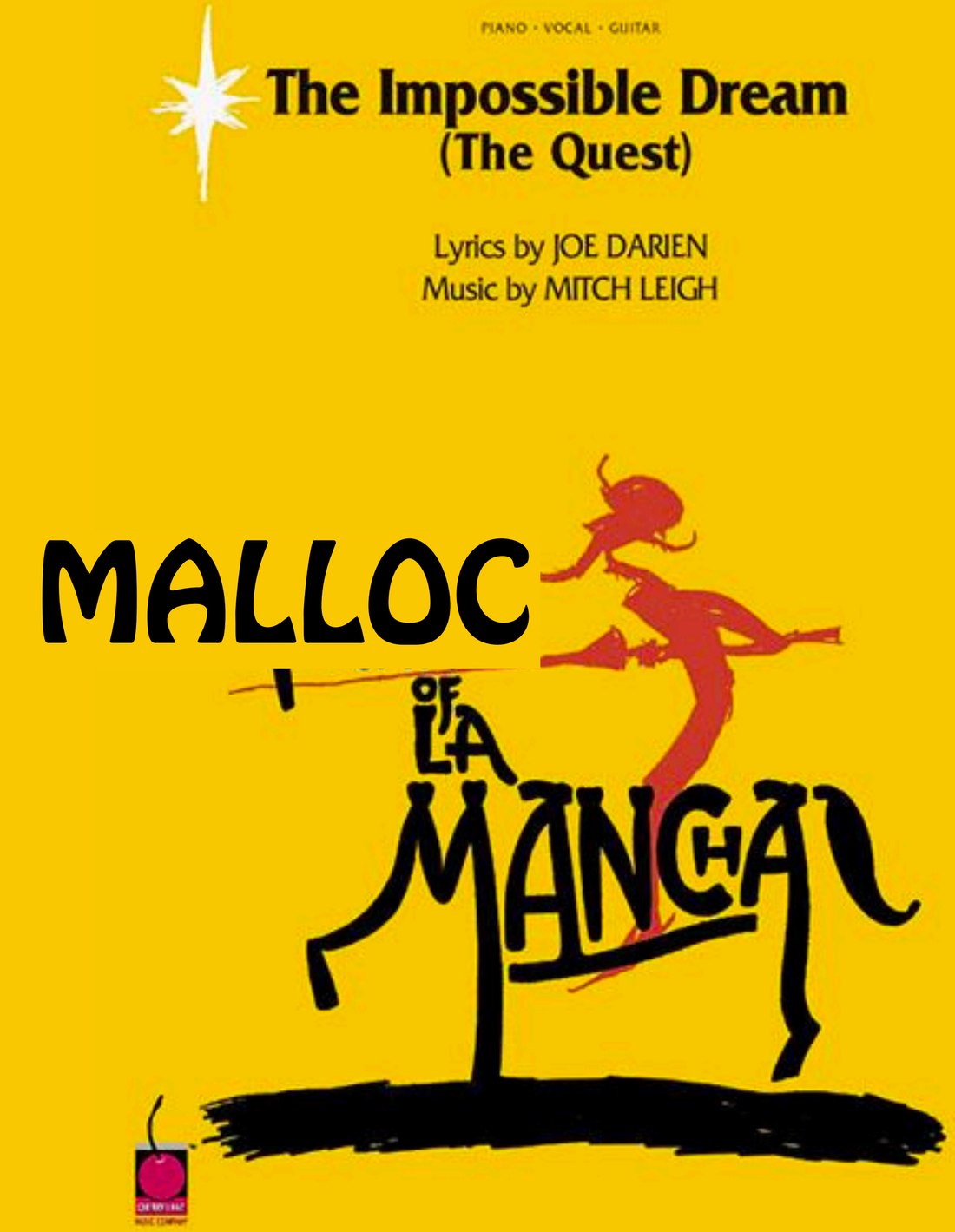
The Impossible Dream
(The Quest)

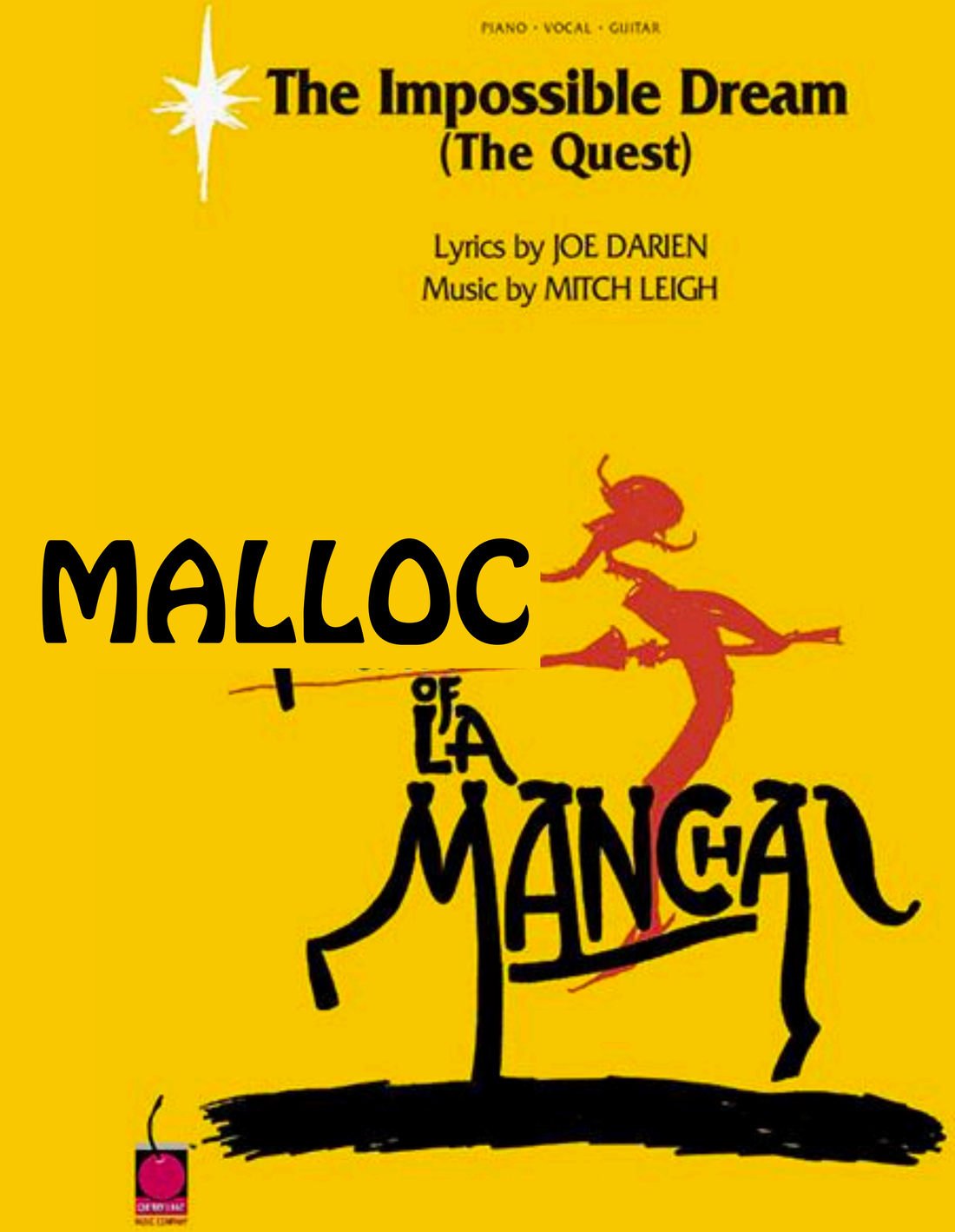
Lyrics by JOE DARION
Music by MITCH LEIGH

MALLOC


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MAN
OF
LA
MANCHA

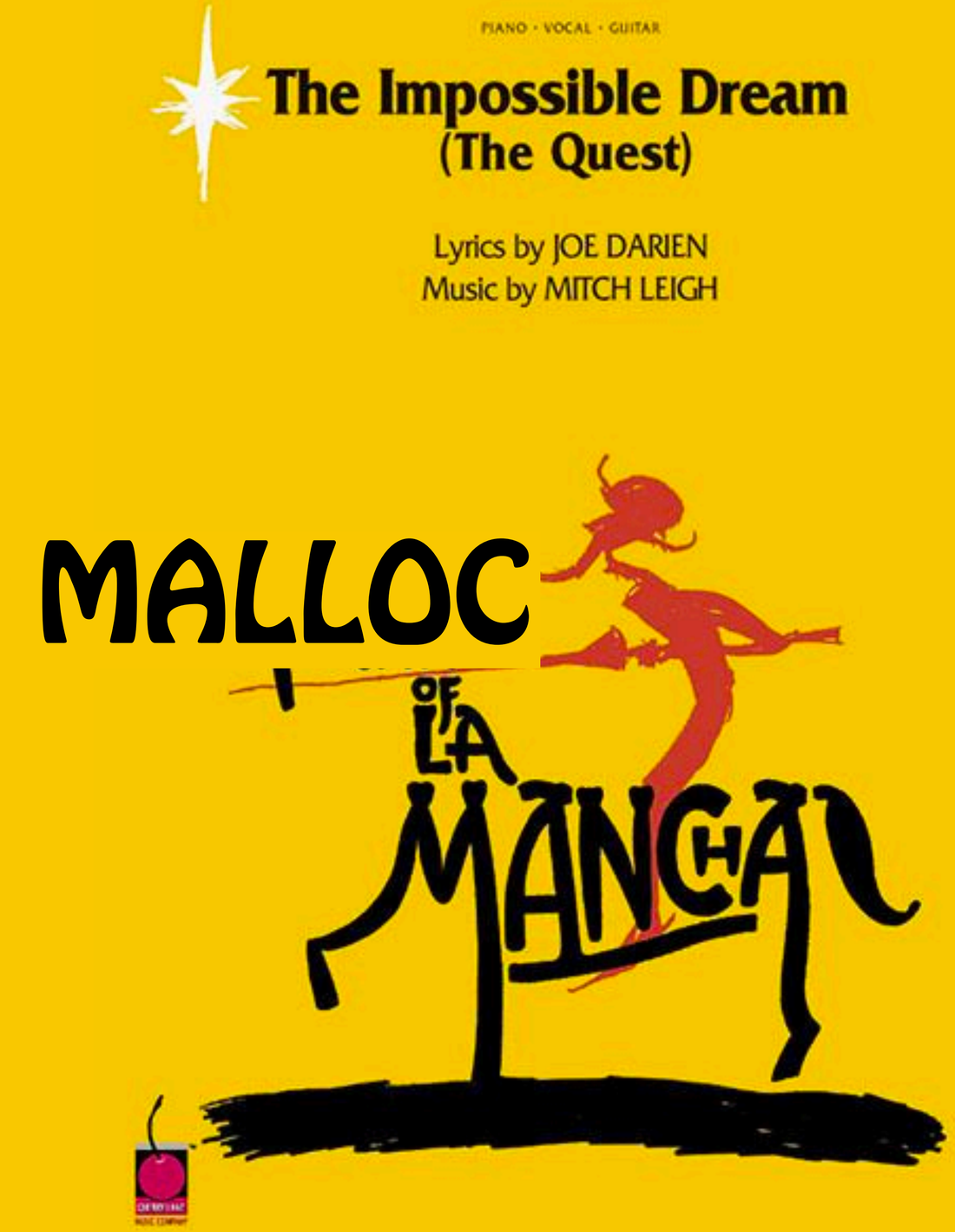
(malloc)

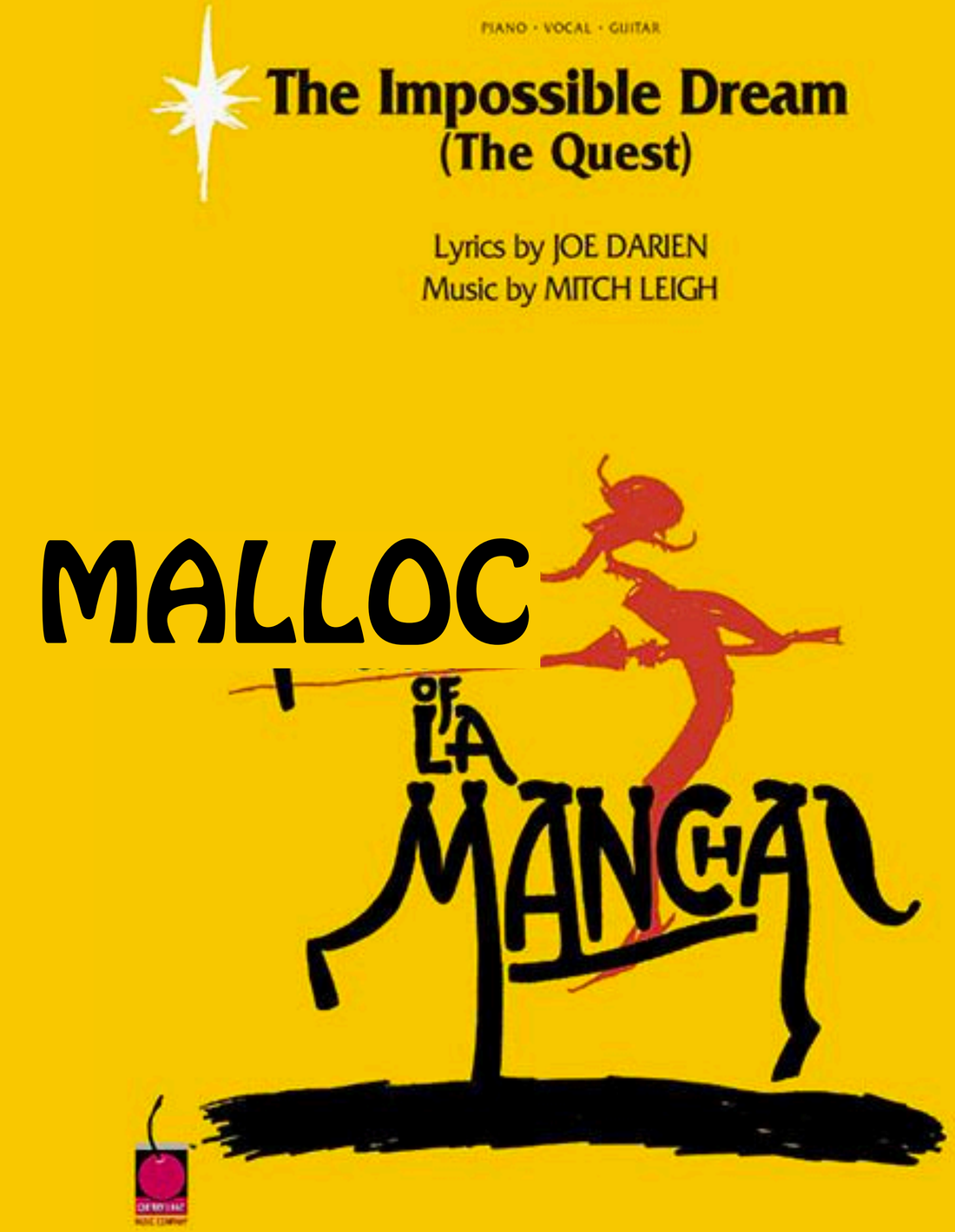

PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA

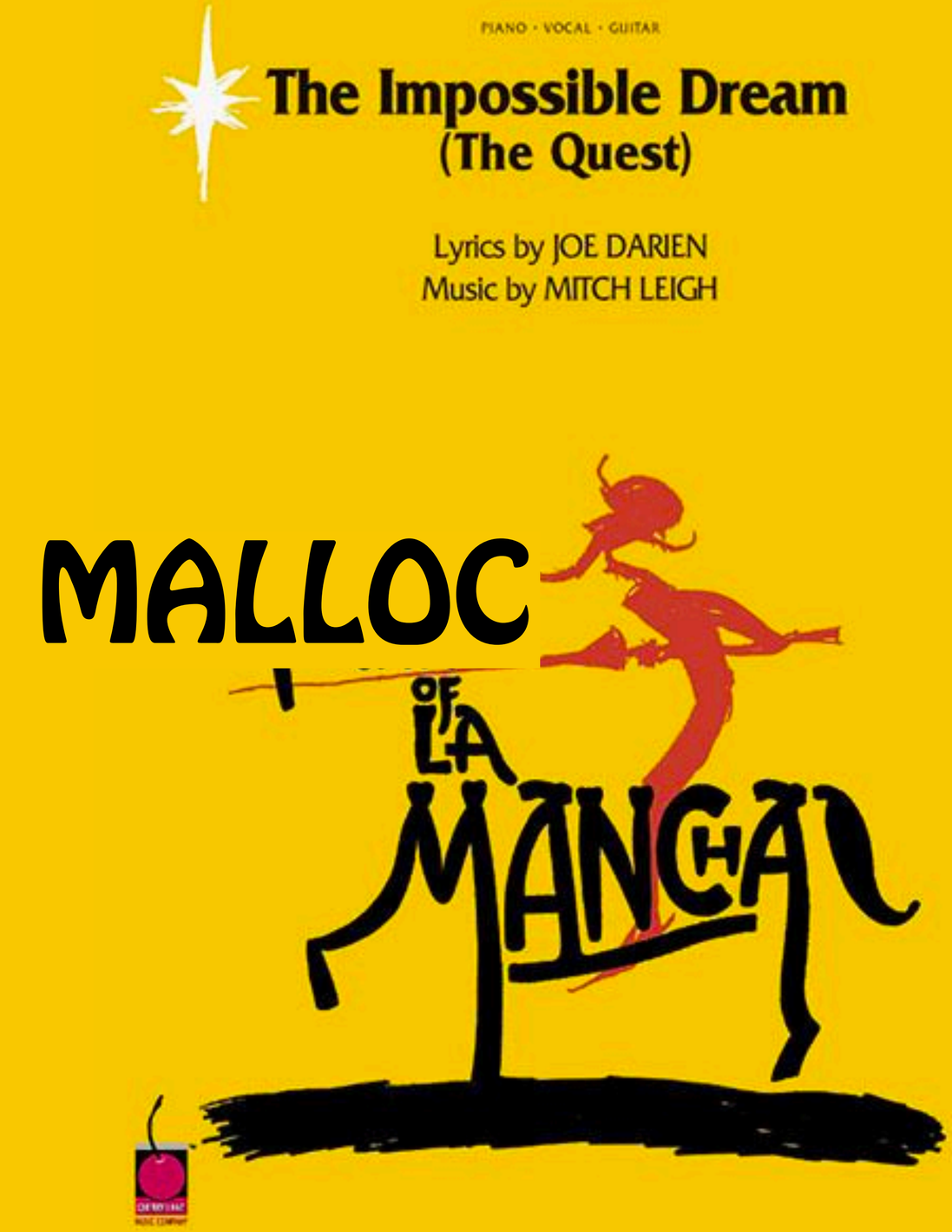

PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MAN OF LA MANCHA

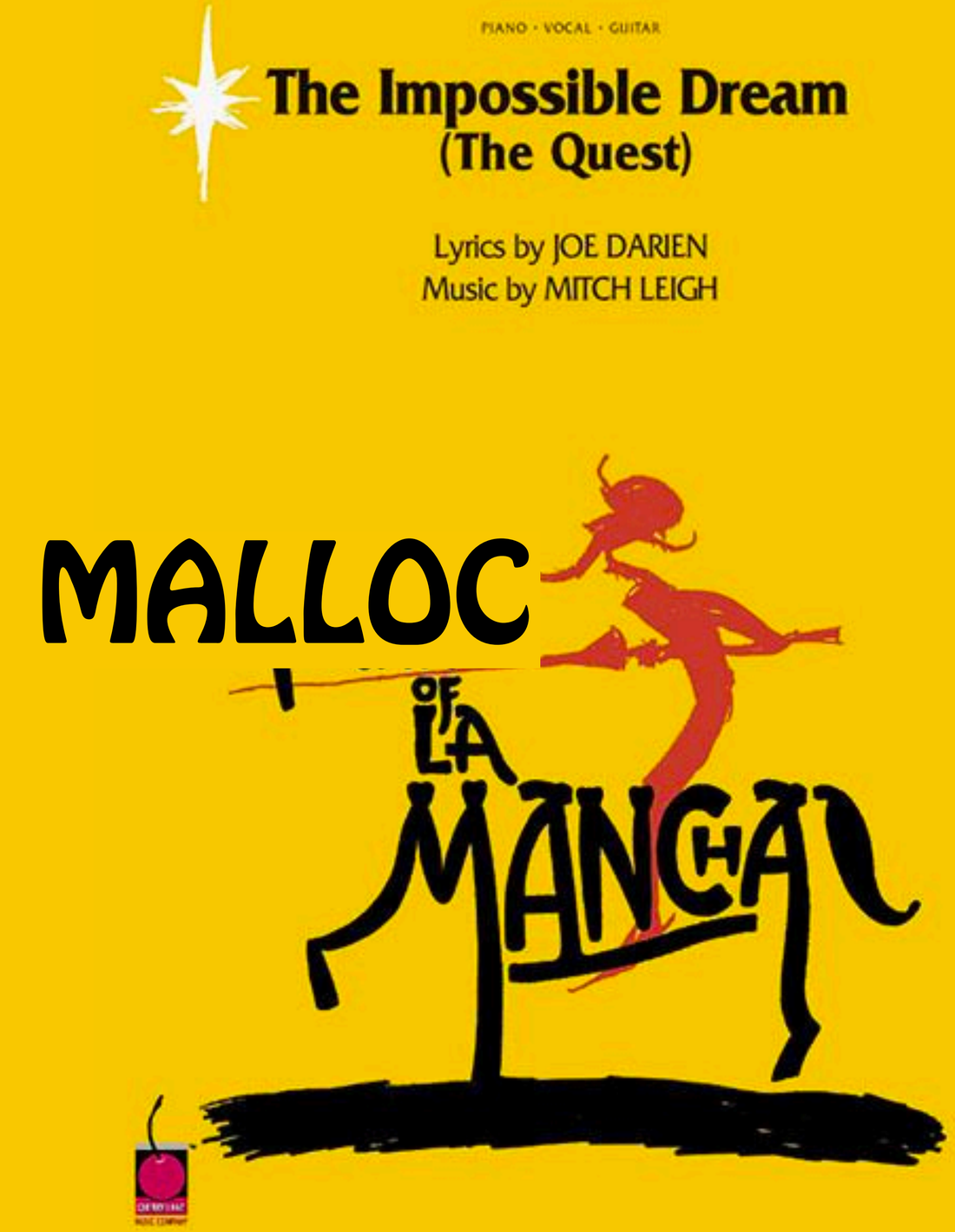
MALLOC


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA

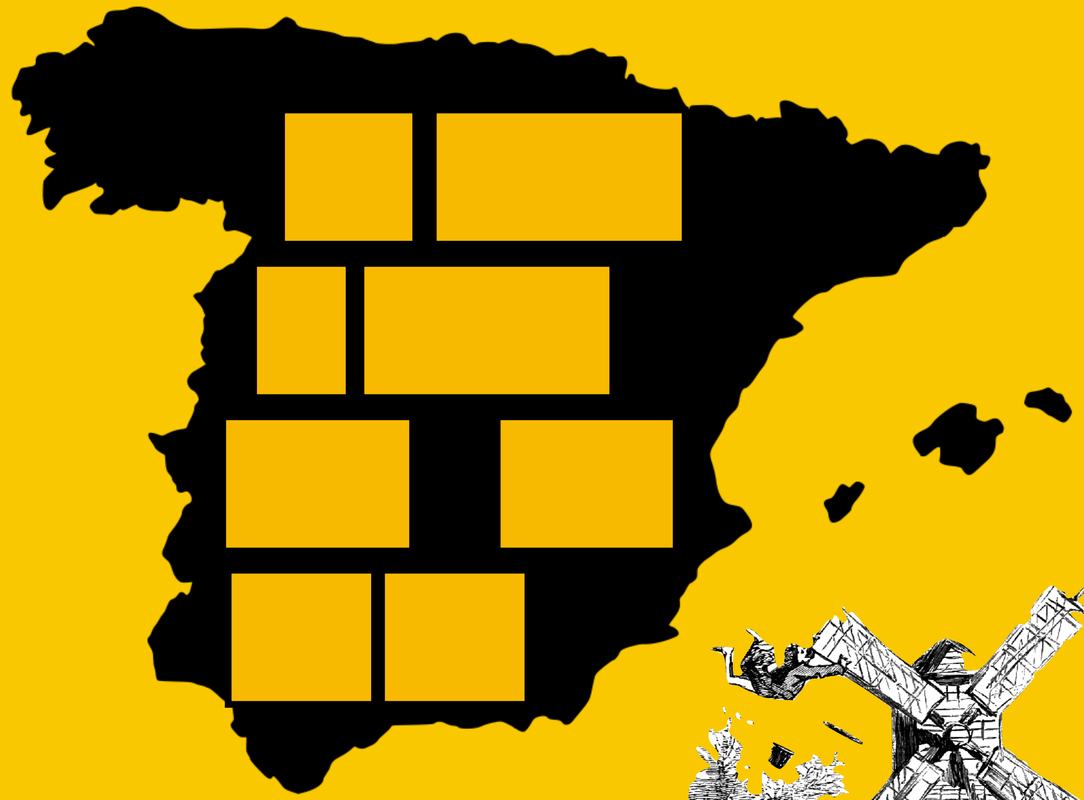
PIANO · VOCAL · GUITAR

The Impossible Dream
(The Quest)

Lyrics by JOE DARIEN
Music by MITCH LEIGH

MALLOC

OF LA MANCHA

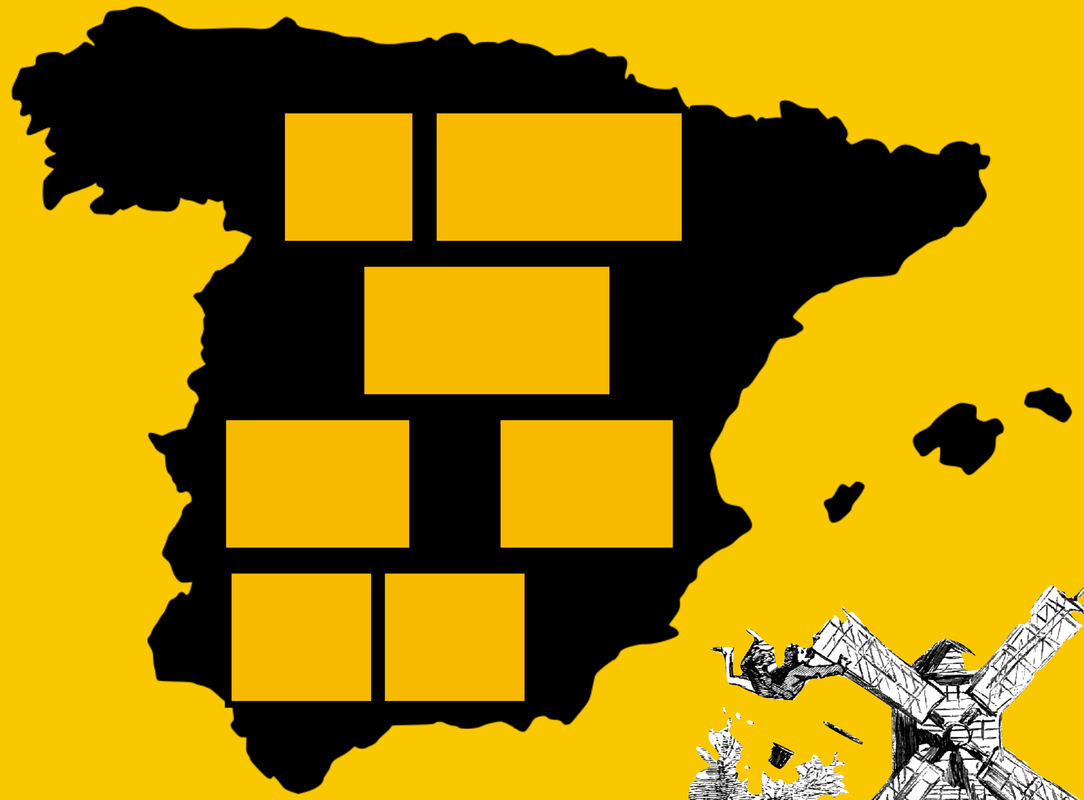


PIANO · VOCAL · GUITAR

The Impossible Dream
(The Quest)

Lyrics by JOE DARIEN
Music by MITCH LEIGH

MALLOC

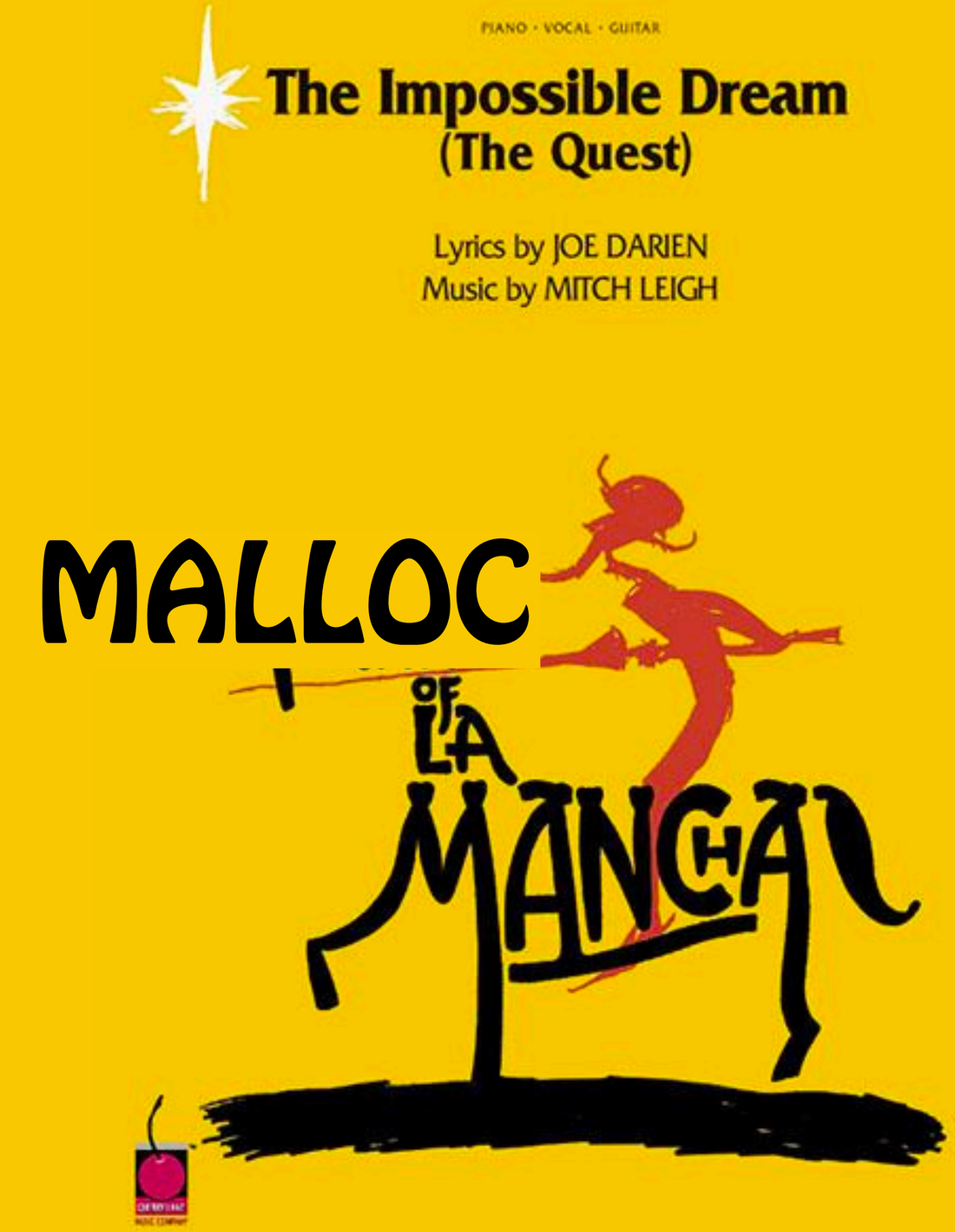


PIANO · VOCAL · GUITAR

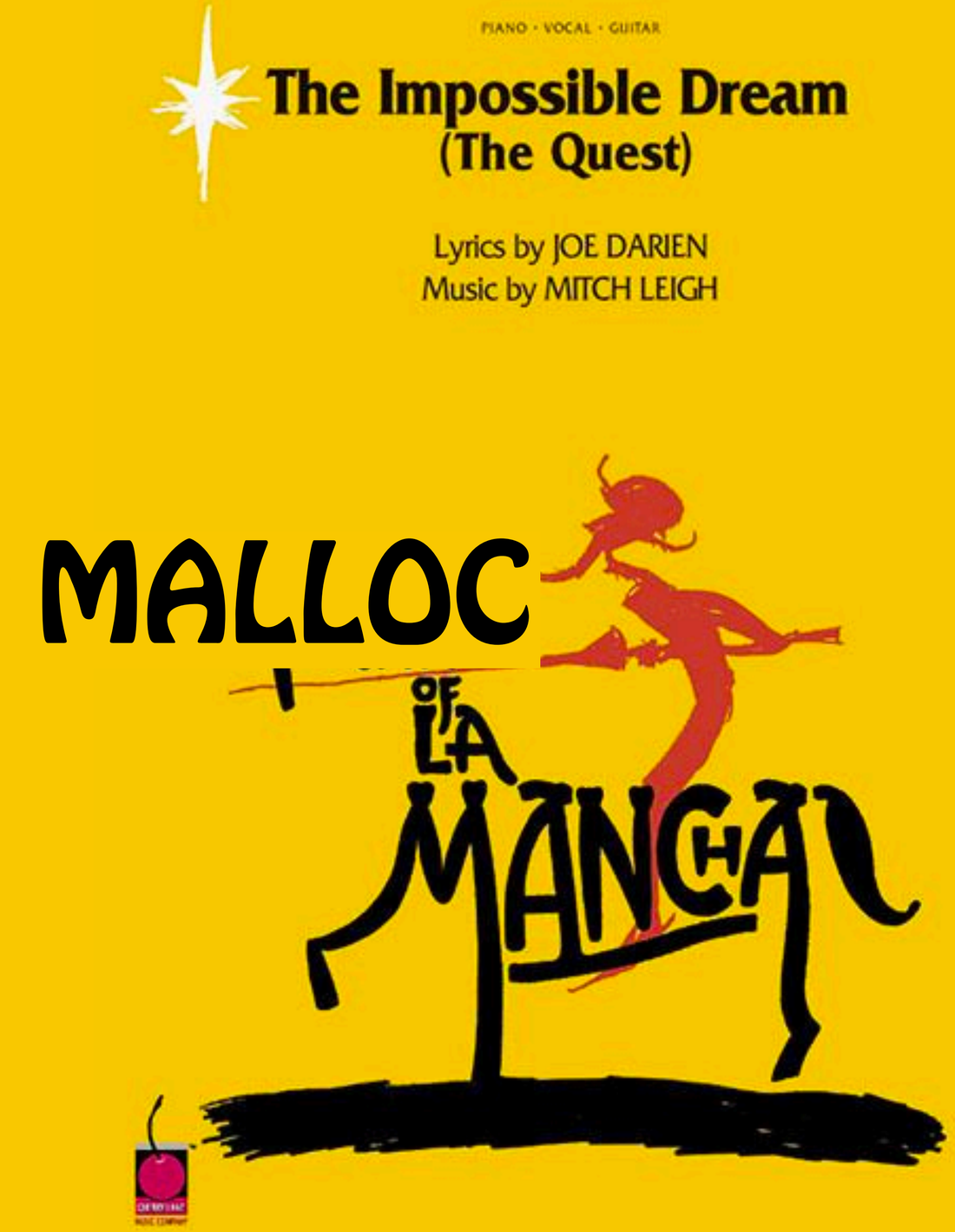
The Impossible Dream
(The Quest)

Lyrics by JOE DARION
Music by MITCH LEIGH

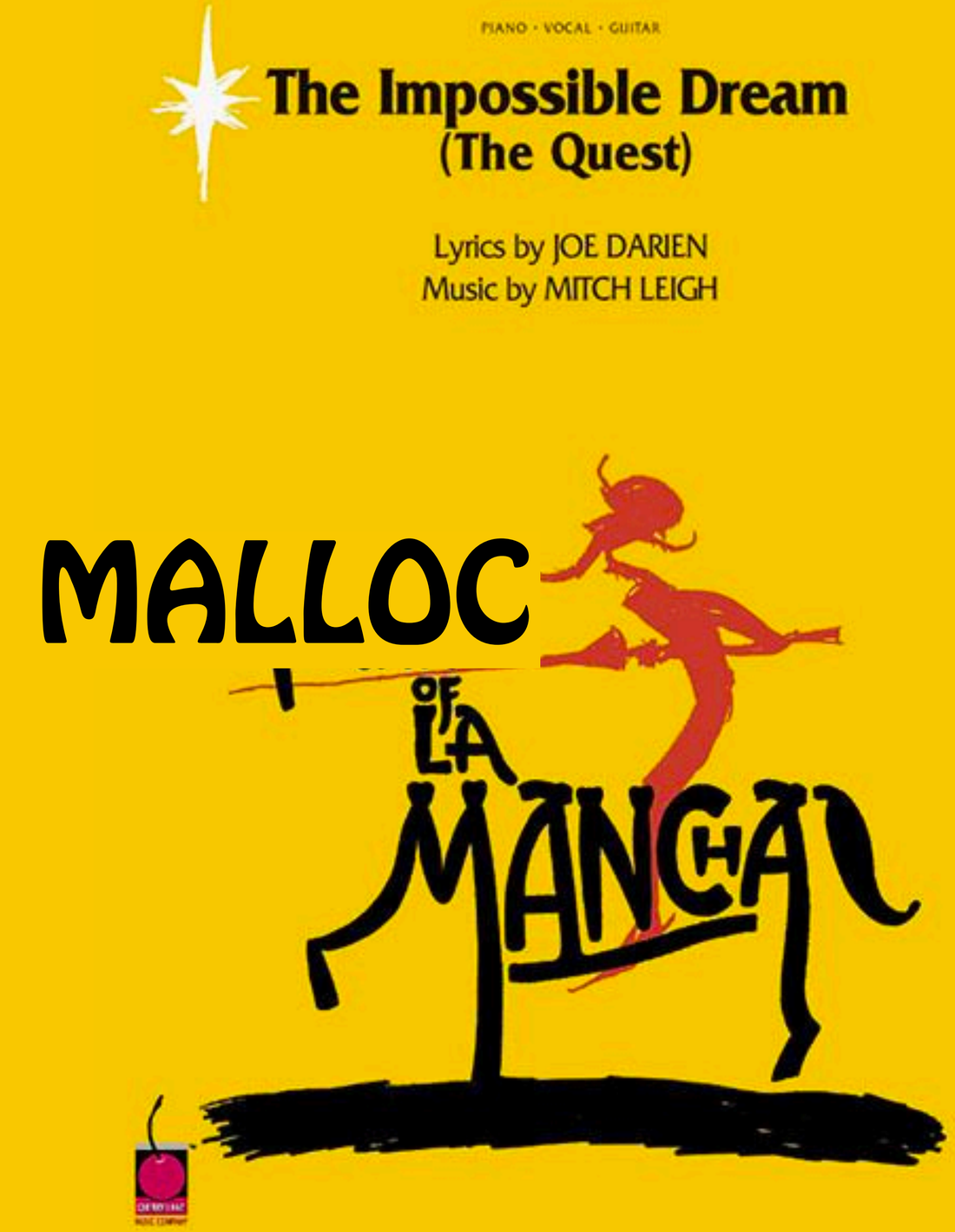
MALLOC


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA

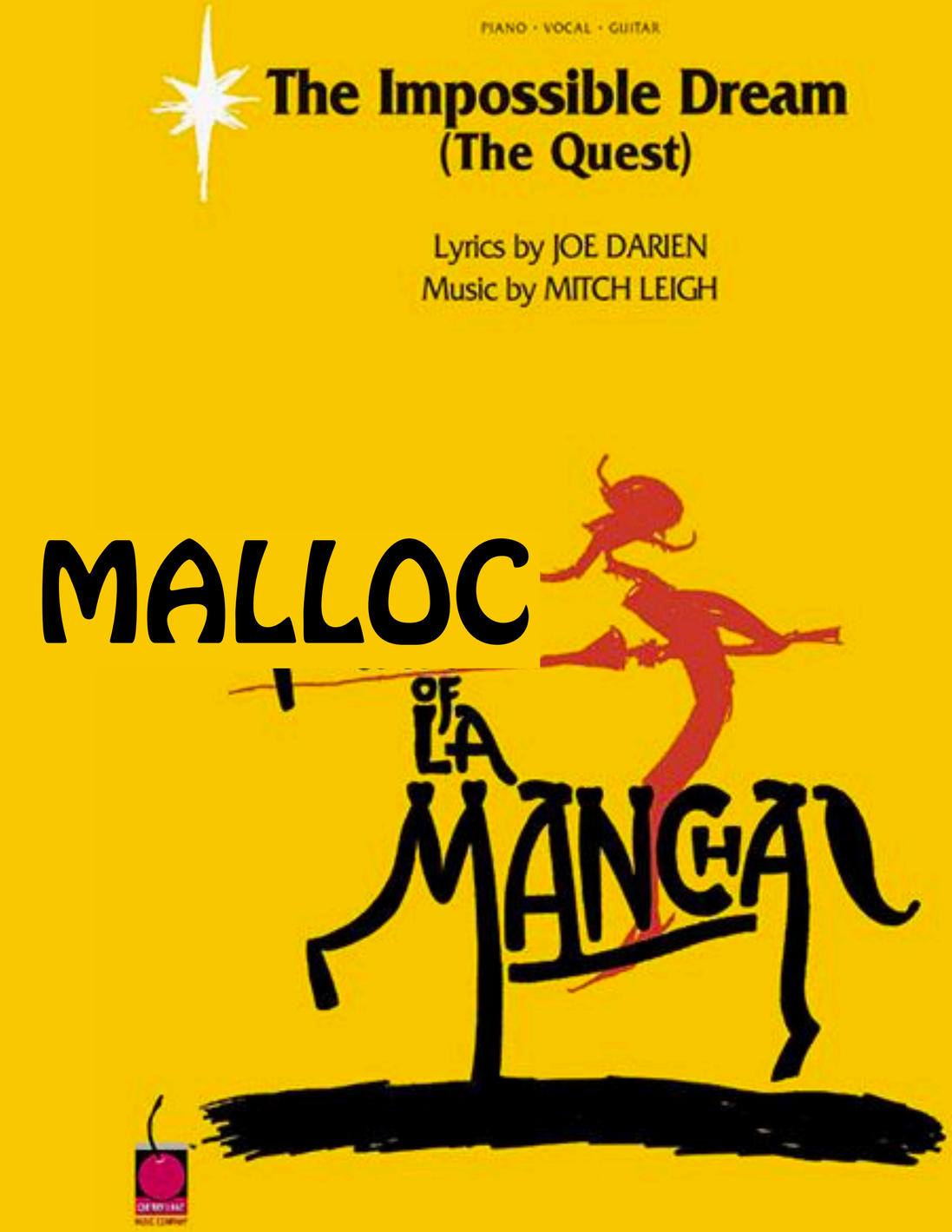

?


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA


?

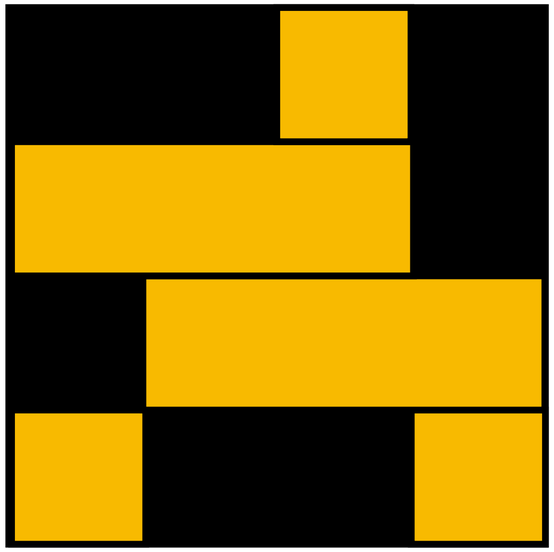

PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA

!


PIANO · VOCAL · GUITAR
The Impossible Dream
(The Quest)
Lyrics by JOE DARION
Music by MITCH LEIGH
MALLOC
OF LA MANCHA

¡!

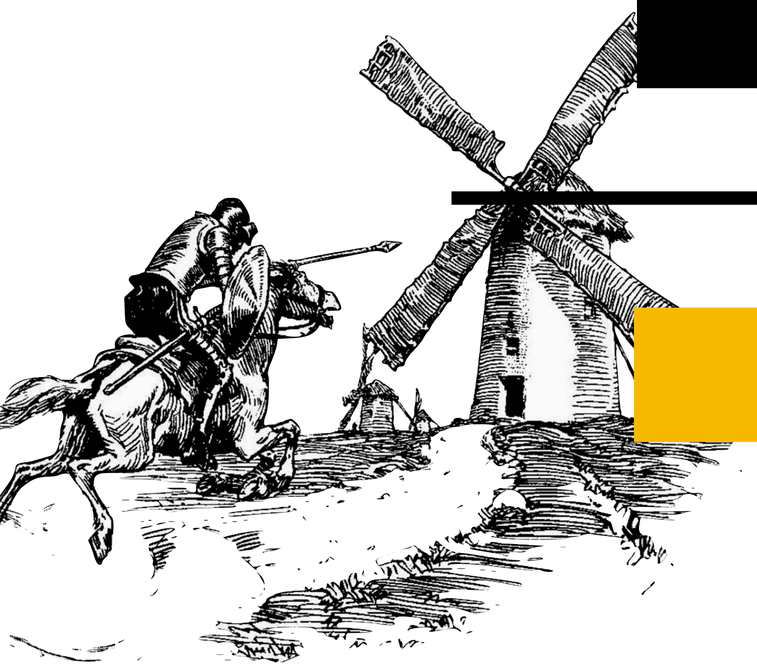
¡Fragmentación!

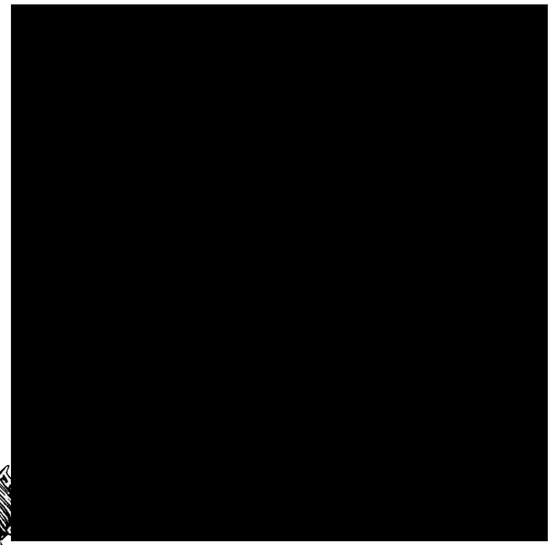


OS Page (“Spain”)

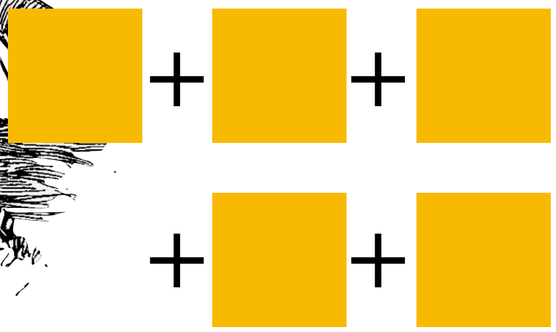
Live objects (malloc’d)

¡Fragmentación!



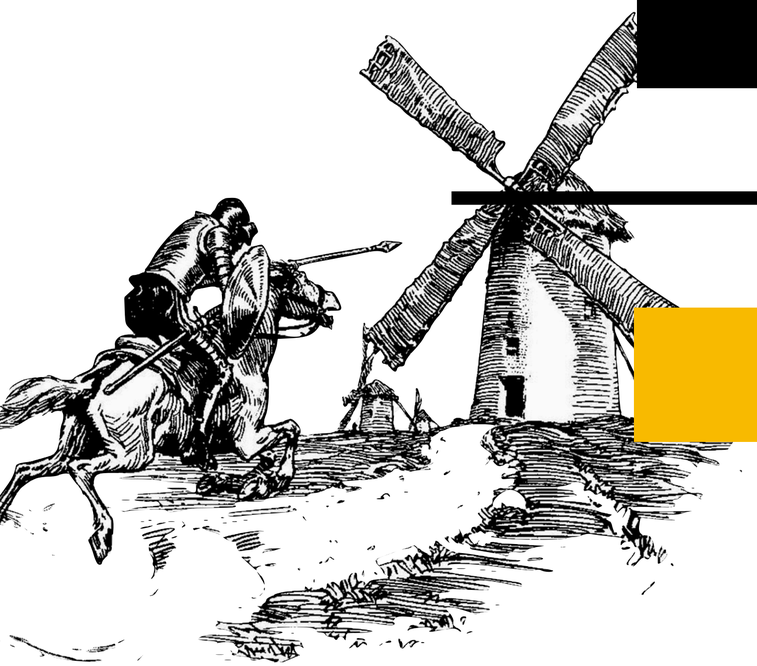


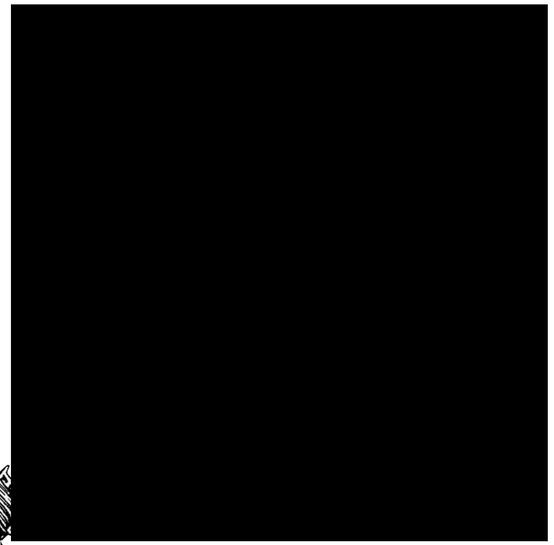
OS Pages



Live objects

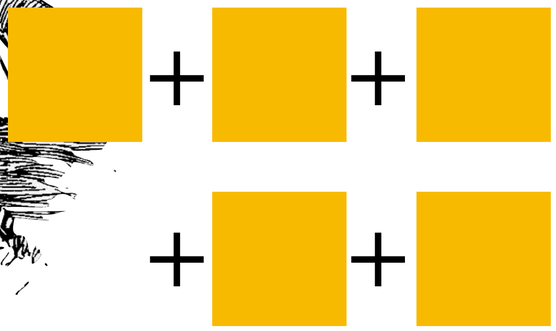
¡Fragmentación!





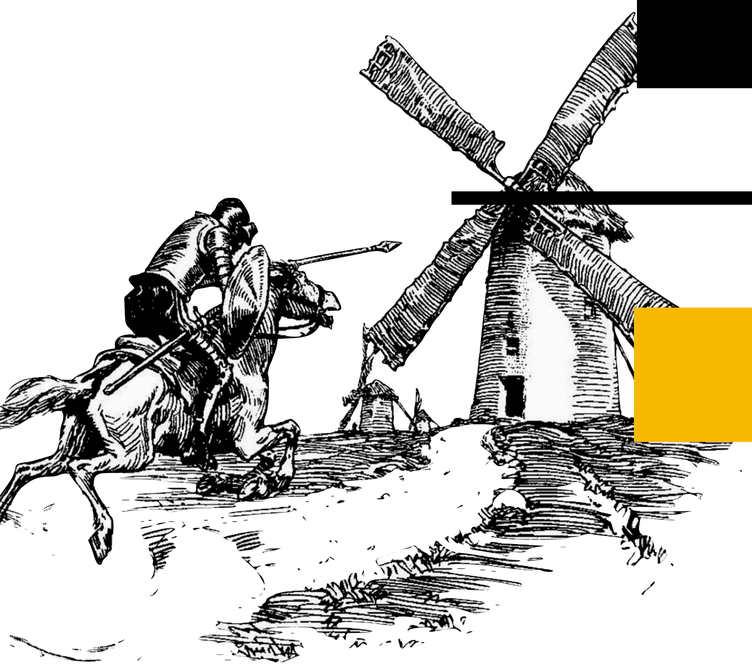
OS Pages

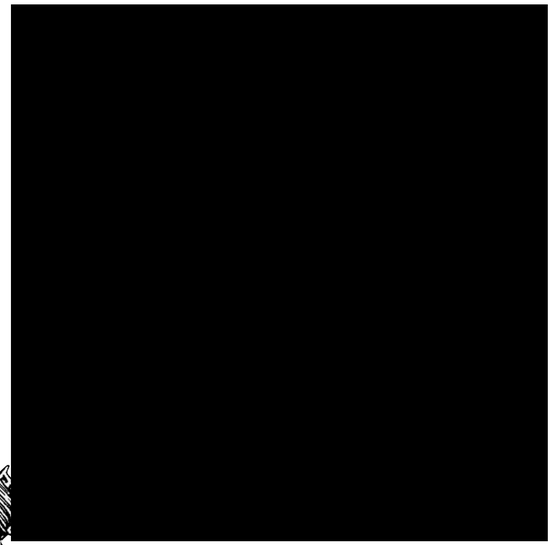
≫ 1



Live objects

¡Fragmentación!

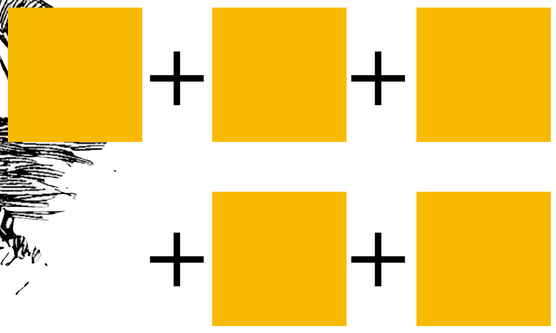




OS Pages

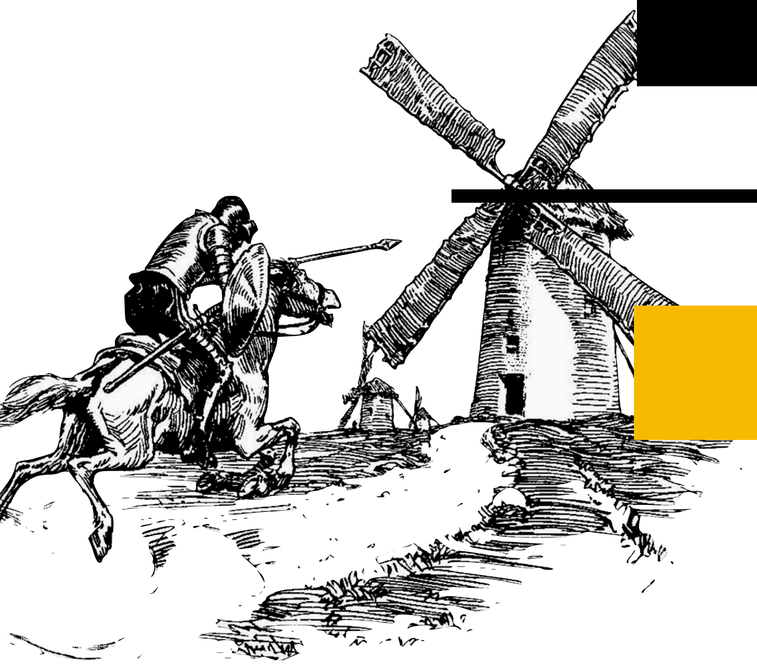
O(log ) 13x!

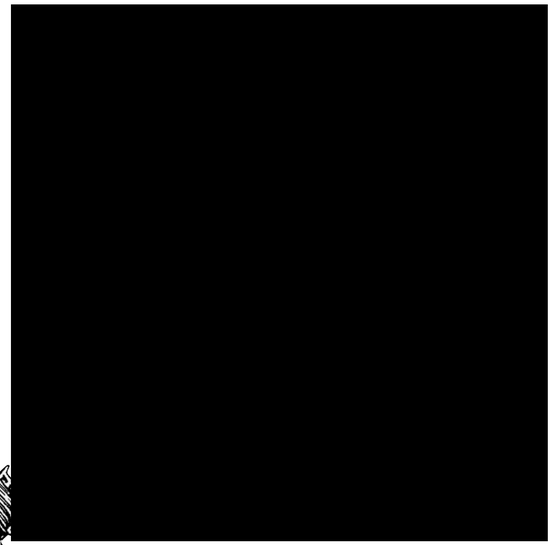
[Robson ’77]



Live objects

¿Compactación?

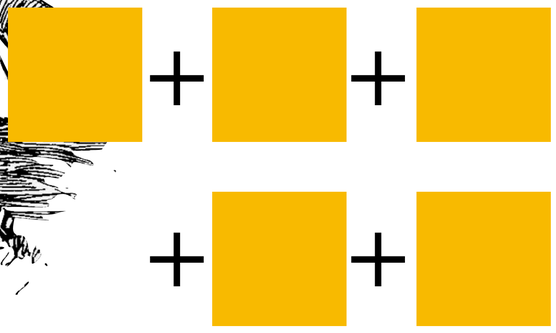




OS Pages

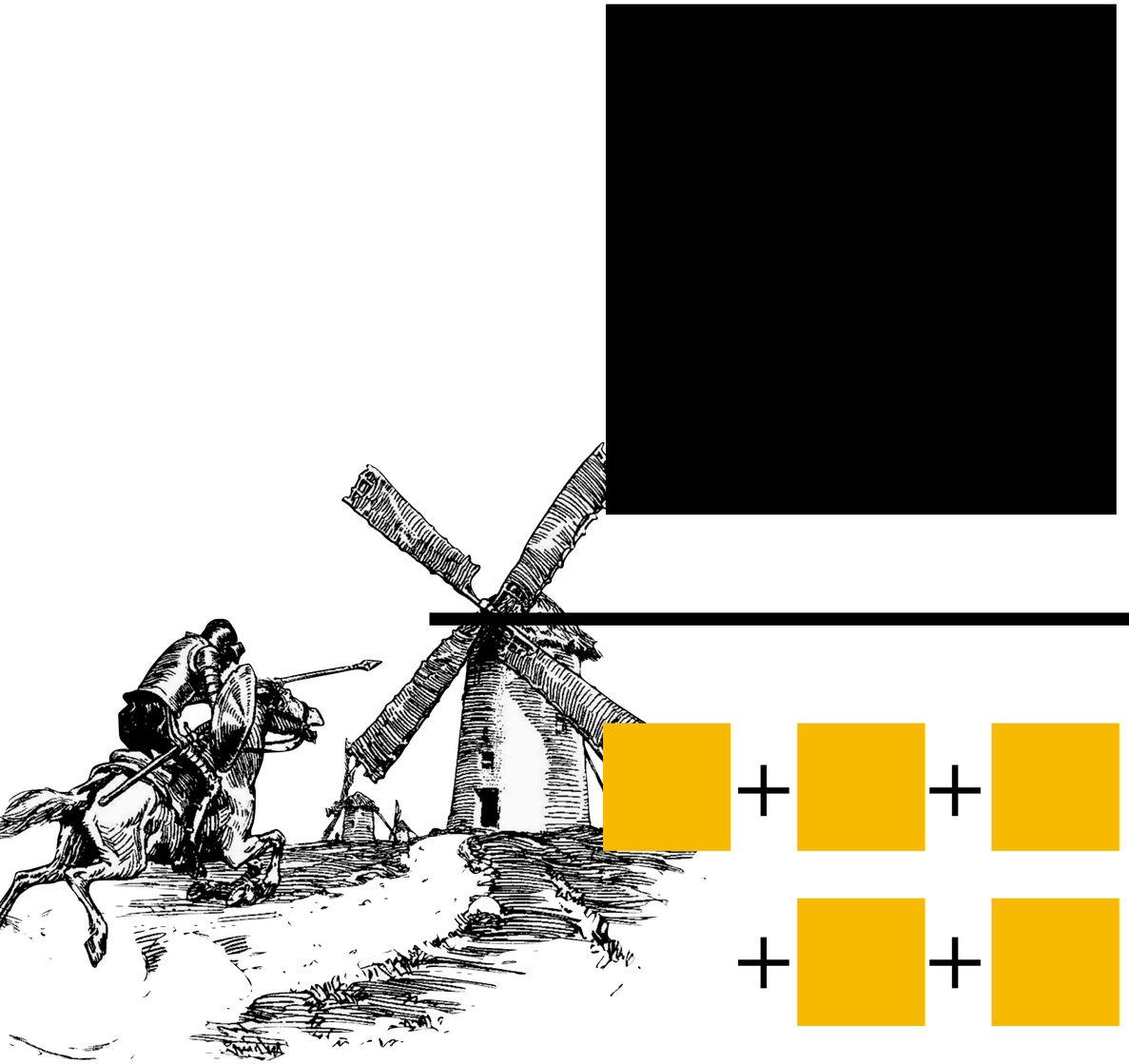
O(log ) 13x!

[Robson ’77]



Live objects

¿Compactación?



OS Pages

O(1)

Live objects

THE

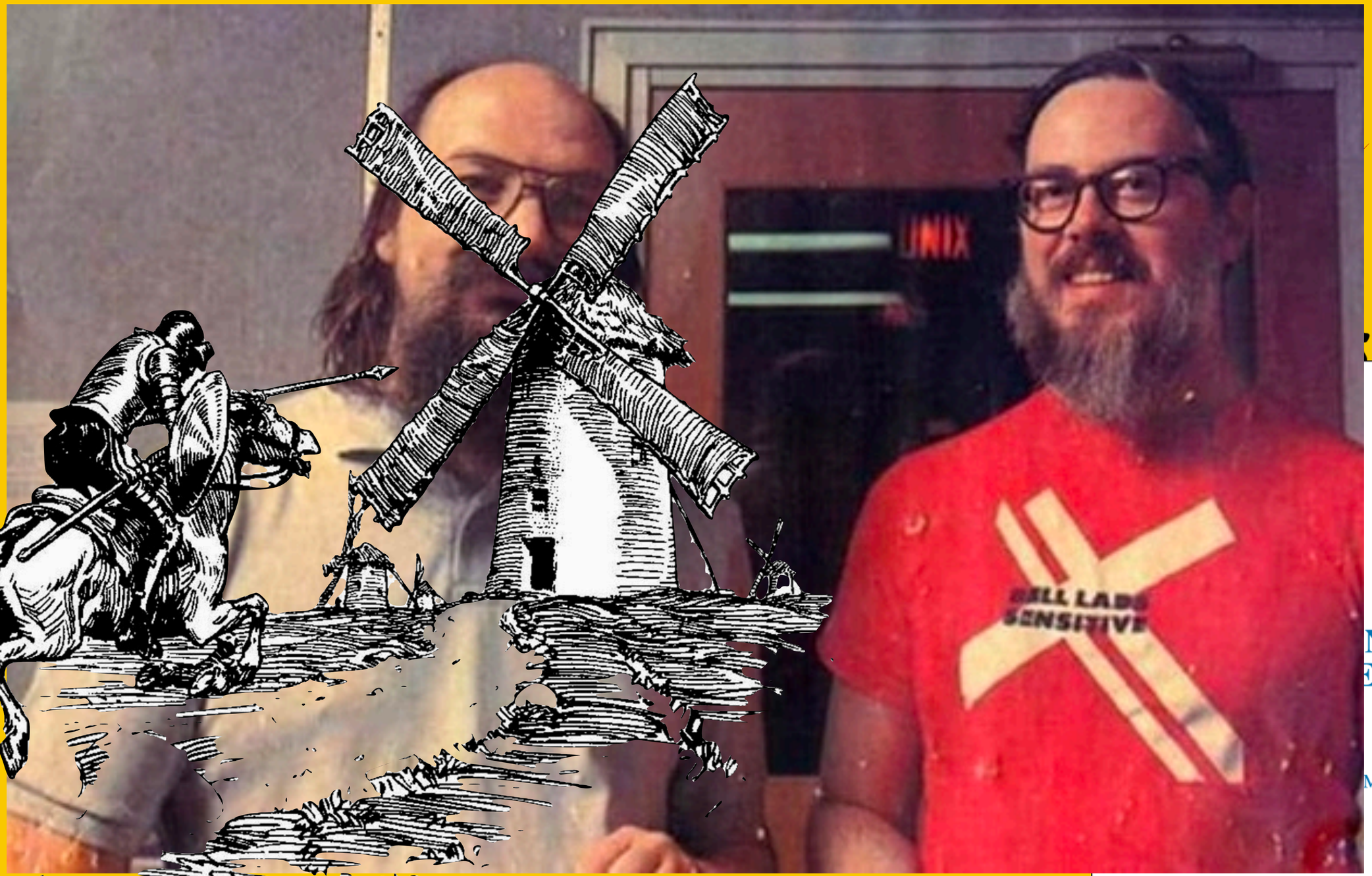
C

PROGRAMMING

LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

PRENTICE HALL SOFTWARE SERIES



NG
E

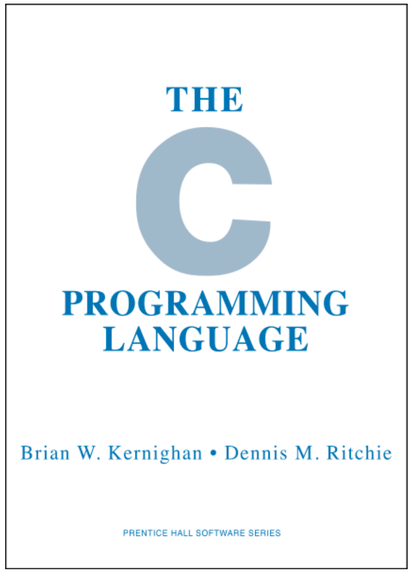
M. Ritchie

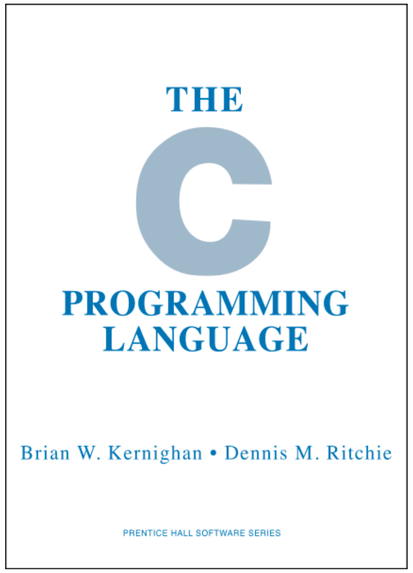
NG
E

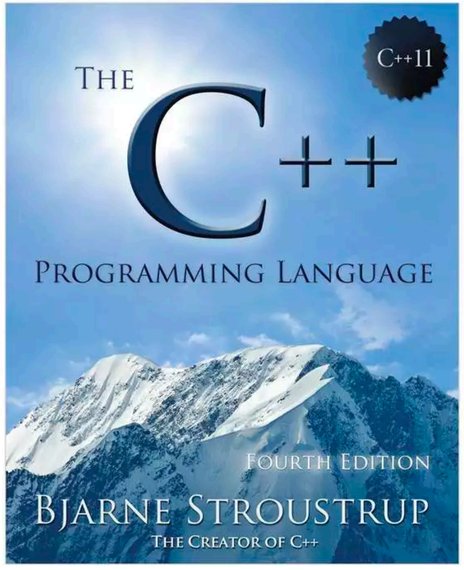
M. Ritchie

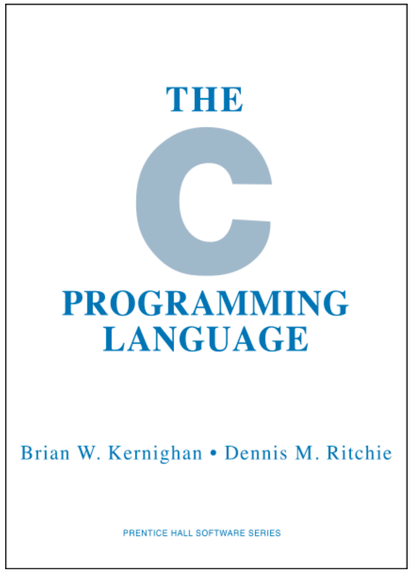
NG
E

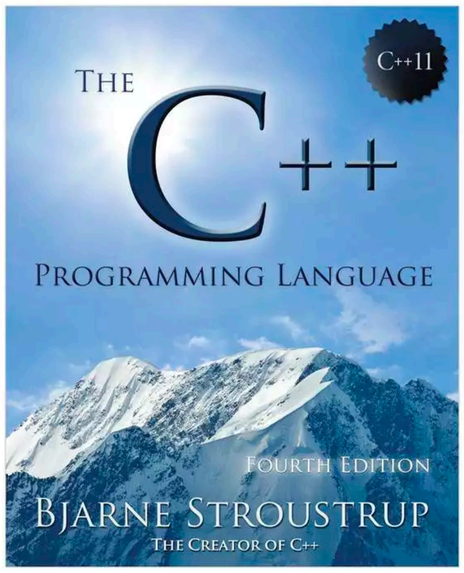
M. Ritchie


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES

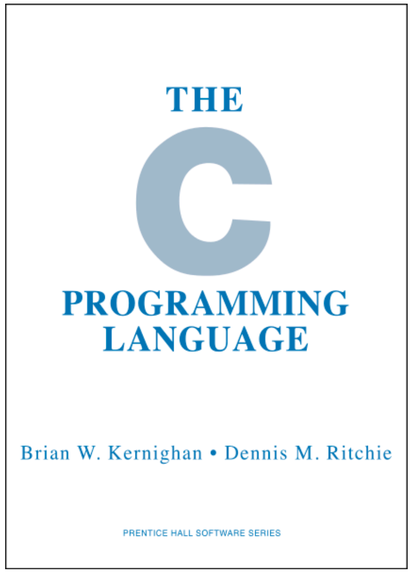

THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES


C++11
THE
C++
PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES


C++11
THE
C++
PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++


The Swift
Programming
Language
Swift 5 Edition



[Lattner, 2016]

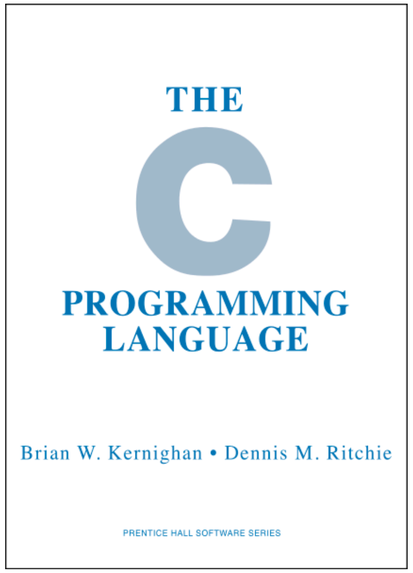
Why not a tracing GC?

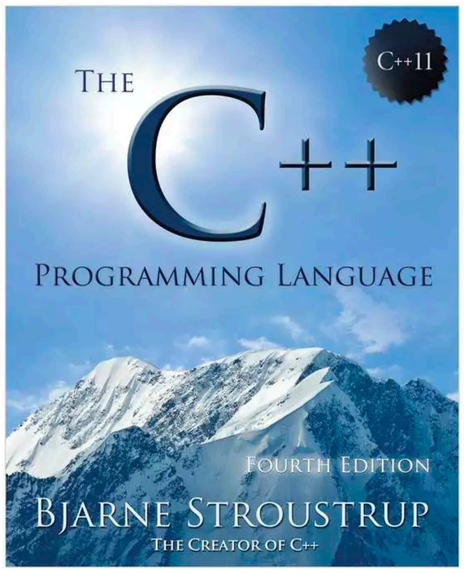
- Native interoperability with unmanaged code
- Deterministic destruction provides:
  - No "finalizer problems" like resurrection, threading, etc.
  - Deterministic performance: can test/debug performance stutters
- Performance:
  - GC use ~3-4x more memory than ARC to achieve good performance
  - Memory usage is very important for mobile and cloud apps
  - Incremental/concurrent GCs slow the mutator like ARC does

Quantifying the Performance of Garbage Collection vs. Explicit Memory Management
Matthew Hertz, Emery D. Berger. OOPSLA'05

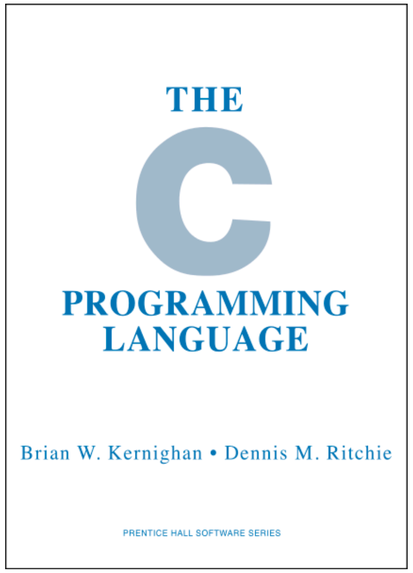
- Native interoperability with unmanaged code
- Deterministic destruction provides:
  - No "finalizer problems" like resurrection, threading, etc.
  - Deterministic performance: can test/debug performance stutters
- Performance:
  - GC use ~3-4x more memory than ARC to achieve good performance
  - Memory usage is very important for mobile and cloud apps
  - Incremental/concurrent GCs slow the mutator like ARC does

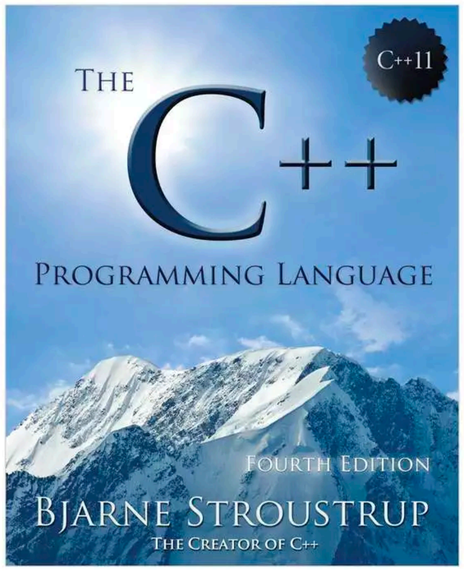
Quantifying the Performance of Garbage Collection vs. Explicit Memory Management
Matthew Hertz, Emery D. Berger. OOPSLA'05


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES

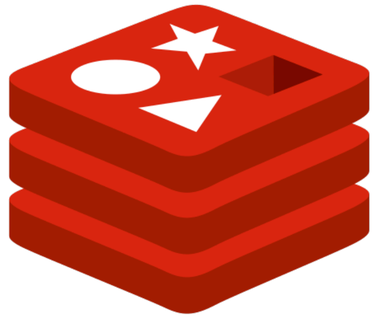

C++11
THE
C++
PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++


The Swift
Programming
Language
Swift 5 Edition


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES


THE
C++
C++11
PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++


The Swift
Programming
Language
Swift 5 Edition

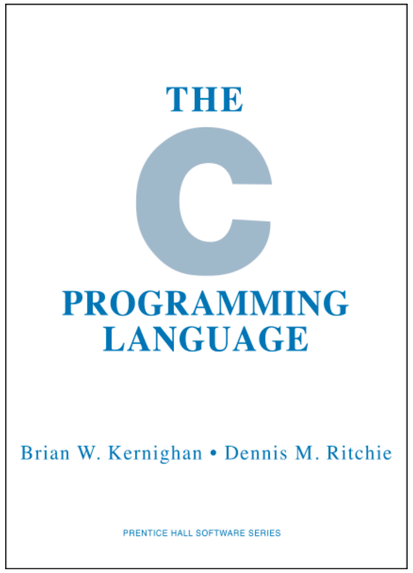


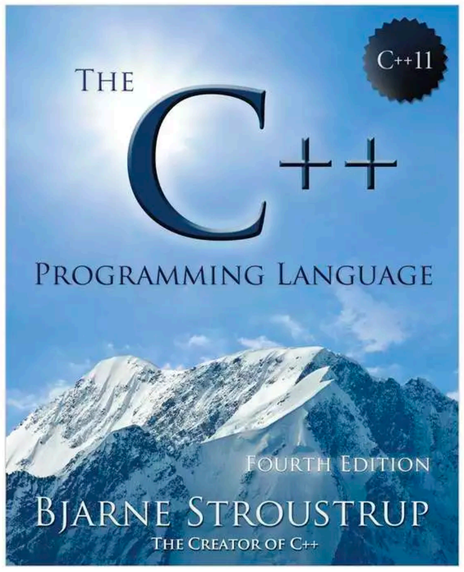
redis


MySQL™

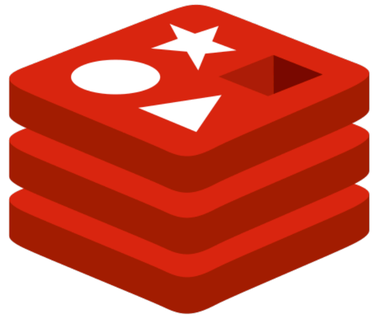

Office


W
P
X
N
O
P
A

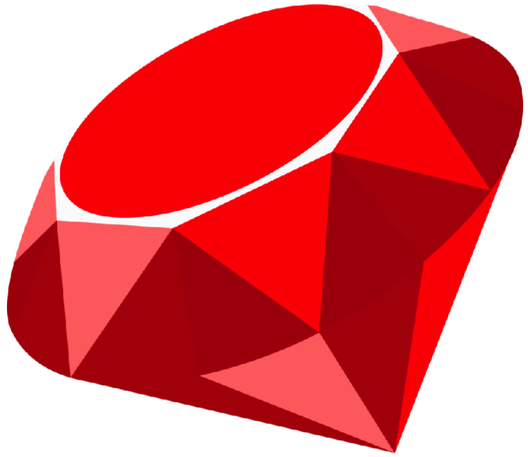

THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES


THE
C++
C++11
PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++


The Swift
Programming
Language
Swift 5 Edition



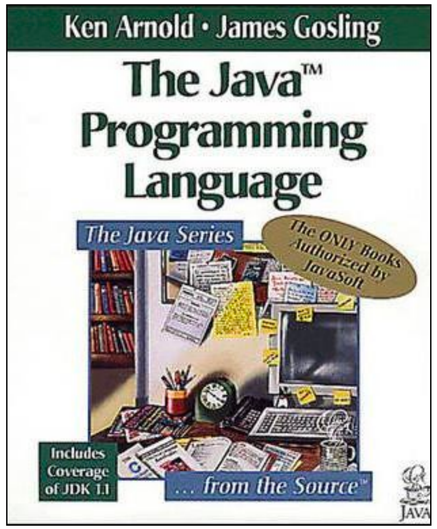
redis

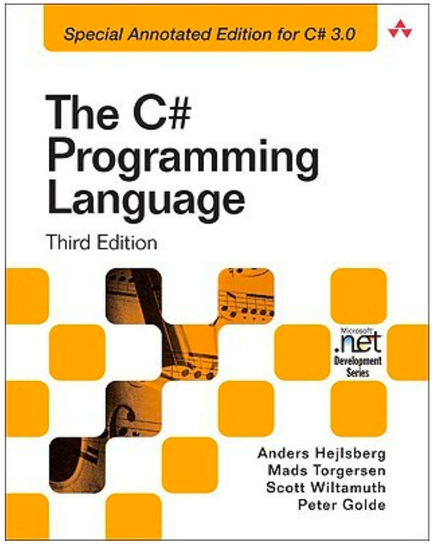


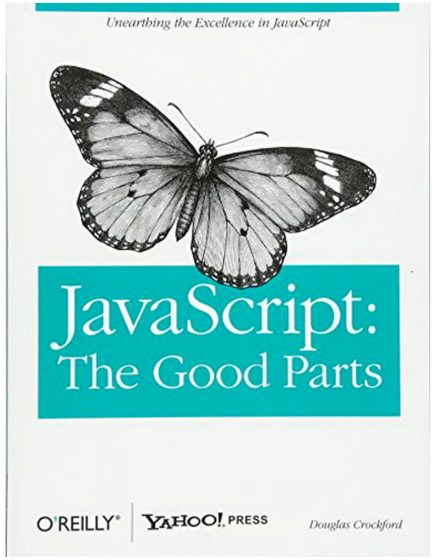

MySQL™

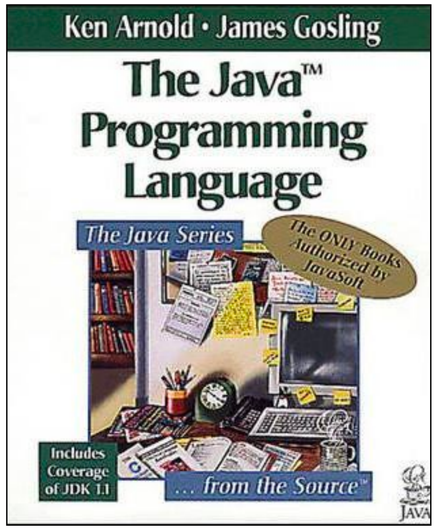

Office

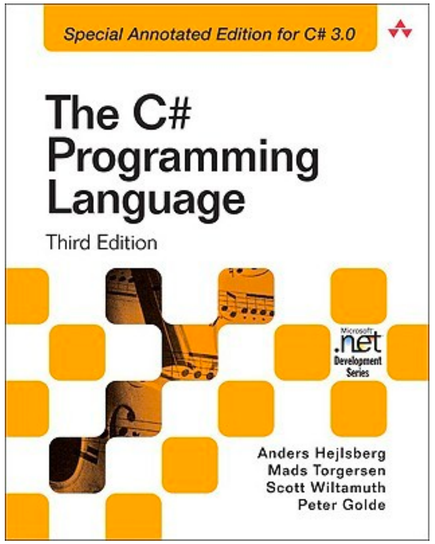

W
P
X
N
O
P
A

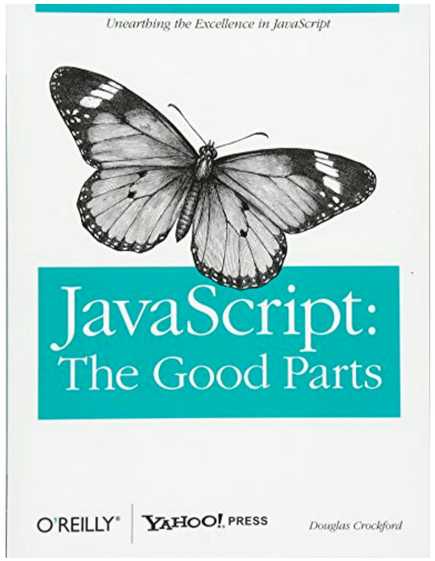

Ken Arnold • James Gosling
The Java™ Programming Language
The Java Series
The ONLY Books Authorized by JavaSoft
Includes Coverage of JDK 1.1
... from the Source™
JAVA


Special Annotated Edition for C# 3.0
The C# Programming Language
Third Edition
Microsoft .net Development Series
Anders Hejlsberg
Mads Torgersen
Scott Wiltamuth
Peter Golde


Unearthing the Excellence in JavaScript
JavaScript: The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford

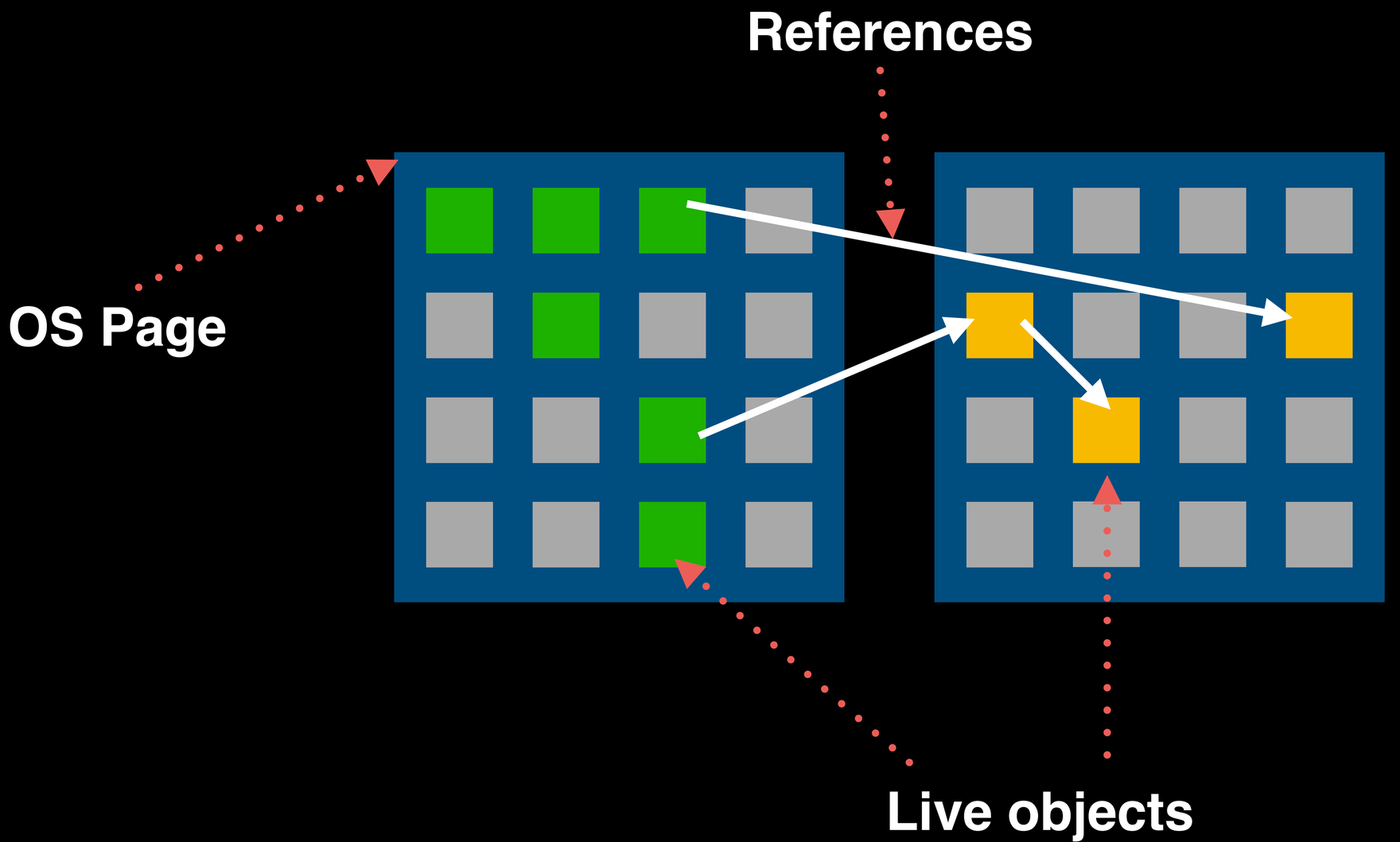


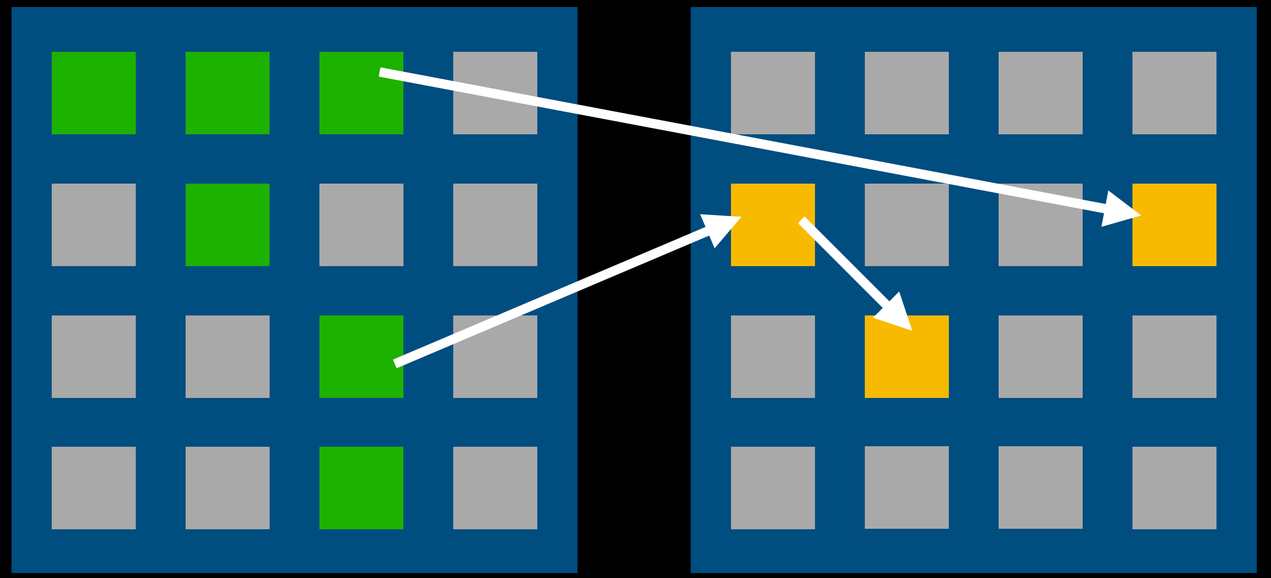


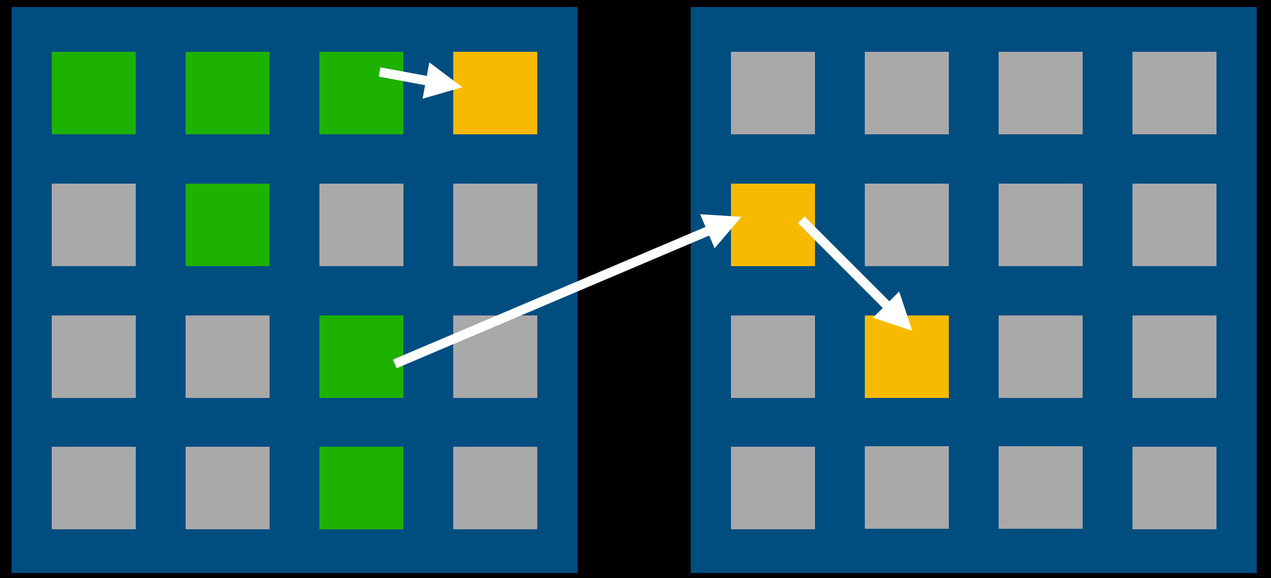


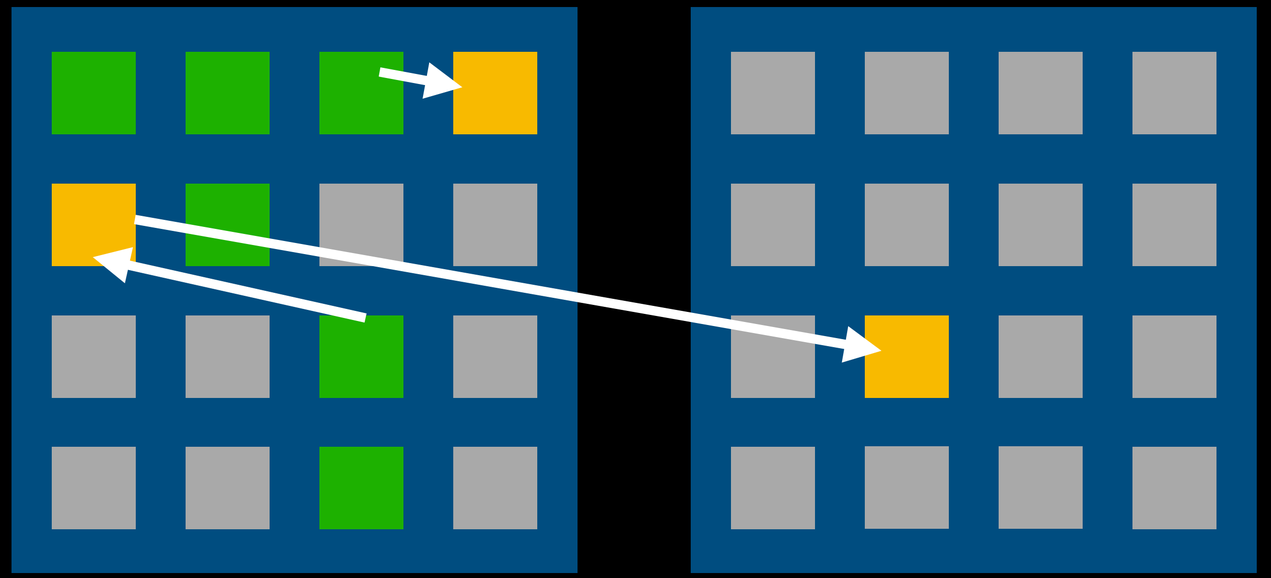
¡Compactación!

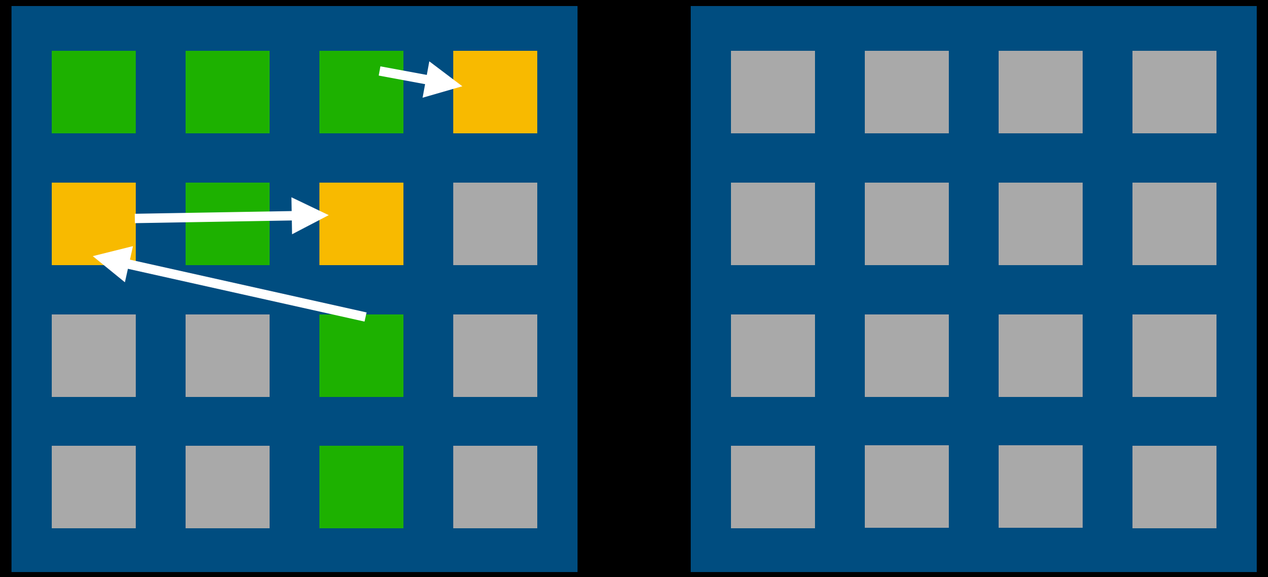
Compaction In Action

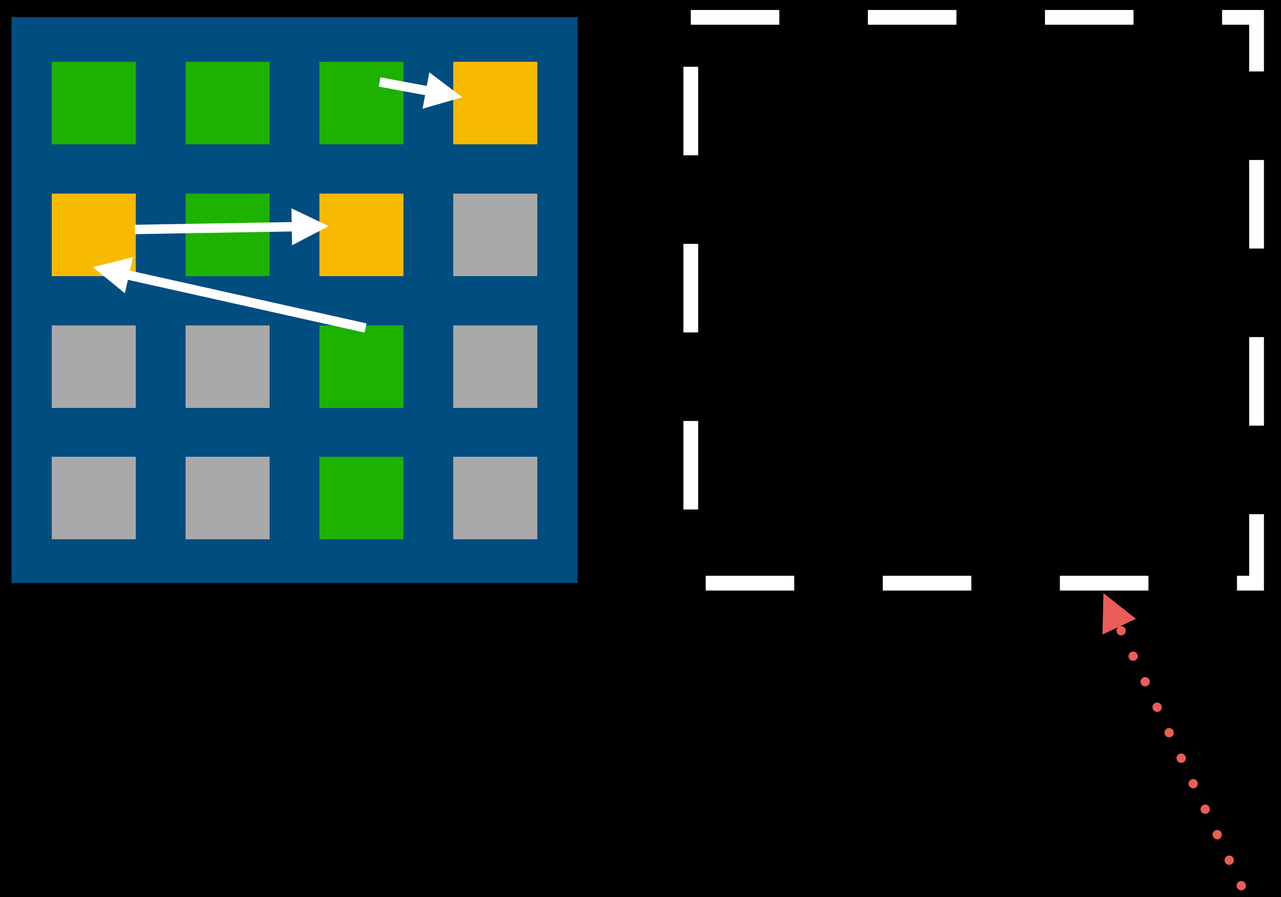












Page returned
to the OS

```
$ clang++ -o yolo main.cc
```

```
$ clang++ -o yolo main.cc
$ strip yolo
```

```
$ clang++ -o yolo main.cc
$ strip yolo
$ ./yolo
```

No way to precisely distinguish pointers from integers

```
$ clang++ -o yolo main.cc
$ strip yolo
$ ./yolo
```

0xDEADC000

No way to precisely distinguish pointers from integers

```
$ clang++ -o yolo main.cc
$ strip yolo
$ ./yolo
```

0xDEADC000

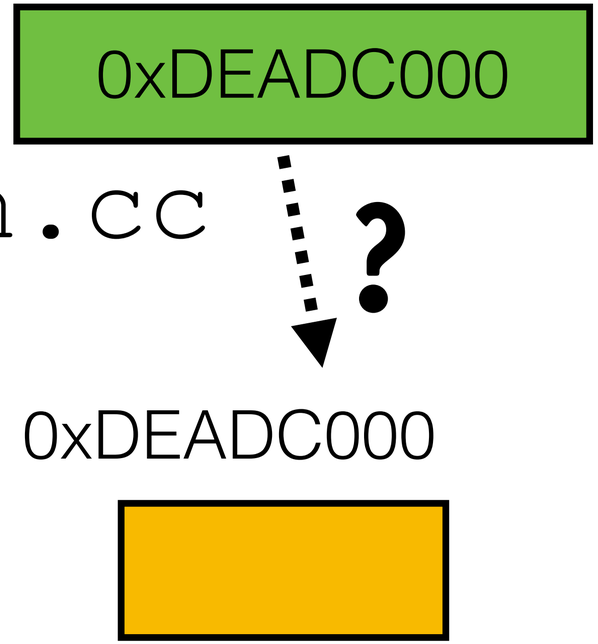
No way to precisely distinguish pointers from integers

0xBEEFC000

```
$ clang++ -o yolo main.cc
$ strip yolo
$ ./yolo
```

No way to precisely distinguish pointers from integers

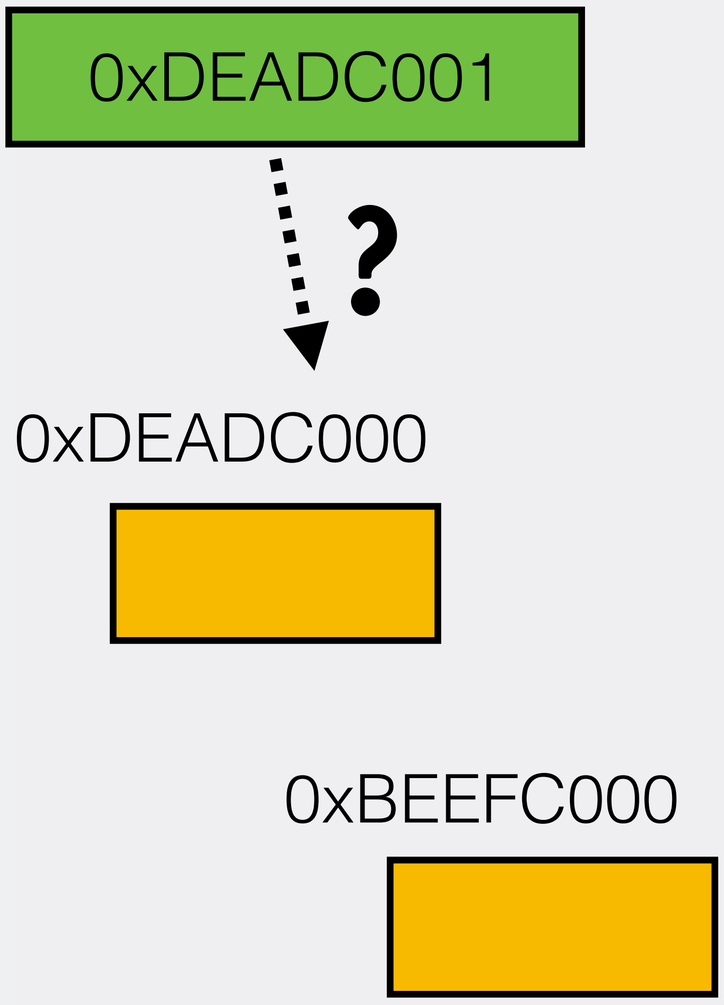
```
$ clang++ -o yolo main.cc
$ strip yolo
$ ./yolo
```



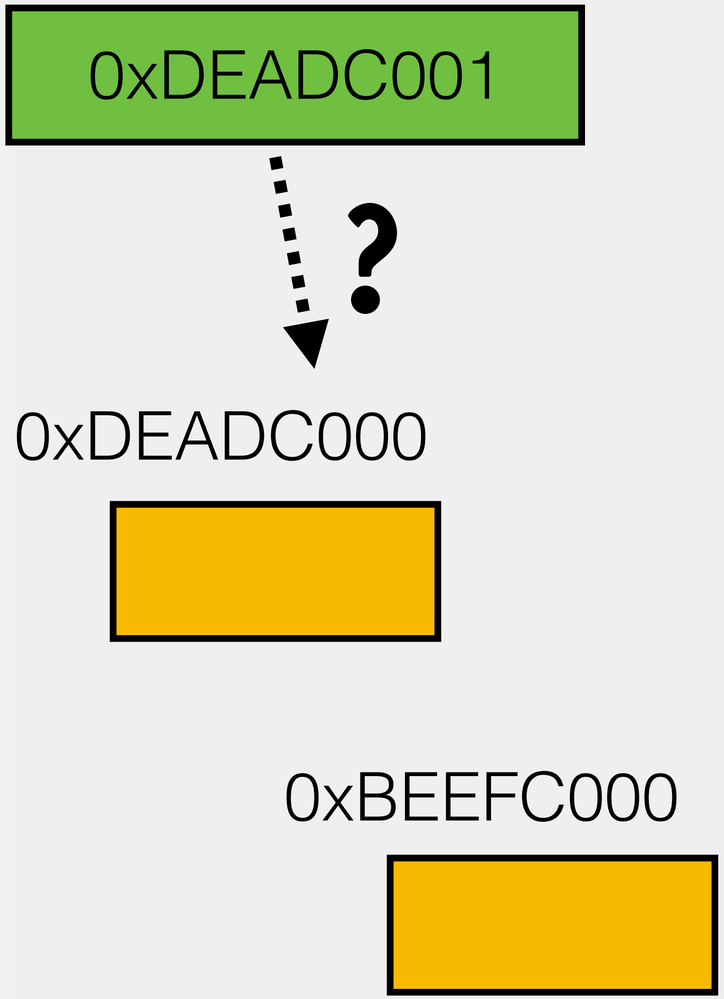
No way to precisely distinguish pointers from integers

```
union tiny
{
 int * ptr;
 uintptr_t flag;
};
tiny x;

// initialize
x.ptr = new int;

// set flag true
x.flag |= 1;
```

```
union tiny
{
 int * ptr;
 uintptr_t flag;
};
tiny x;

// initialize
x.ptr = new int;

// set flag true
x.flag |= 1;
```



```
union tiny
{
 int * ptr;
 uintptr_t flag;
};
tiny x;

// initialize
x.ptr = new int;

// set flag true
x.flag |= 1;
```



MESH

Compaction without Relocation for C/C++

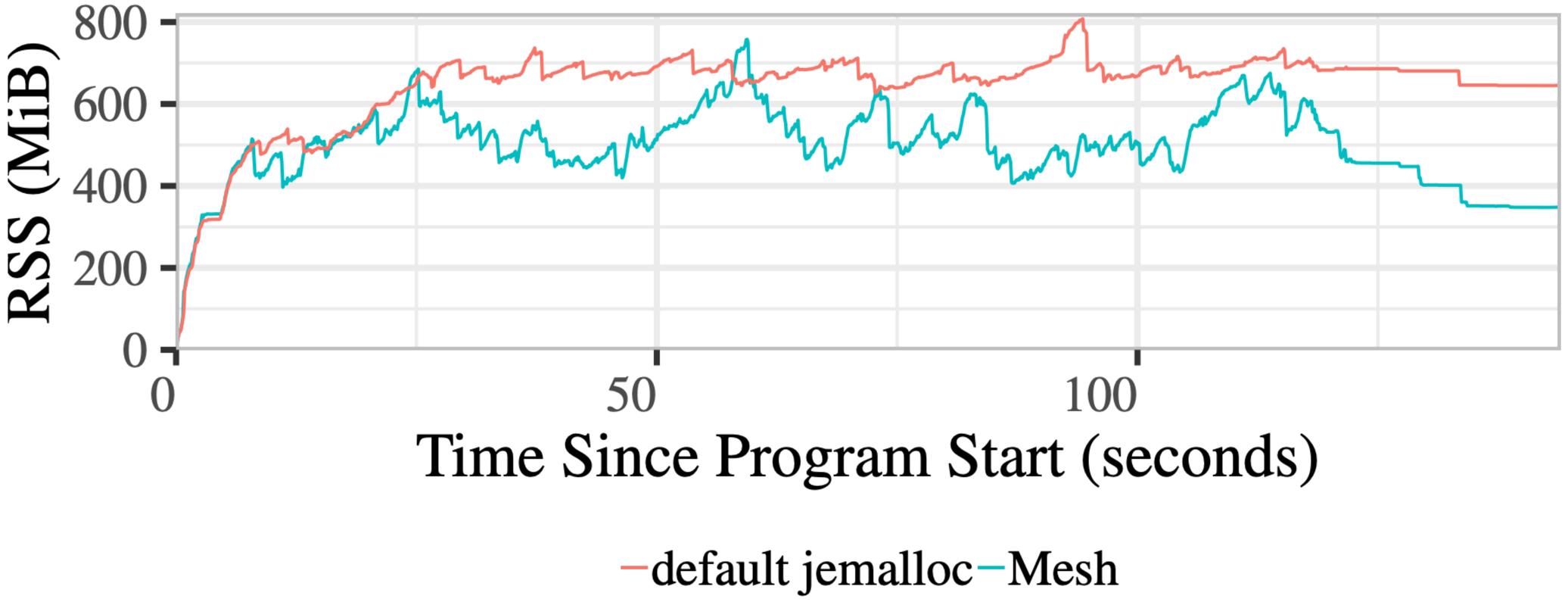
No code changes

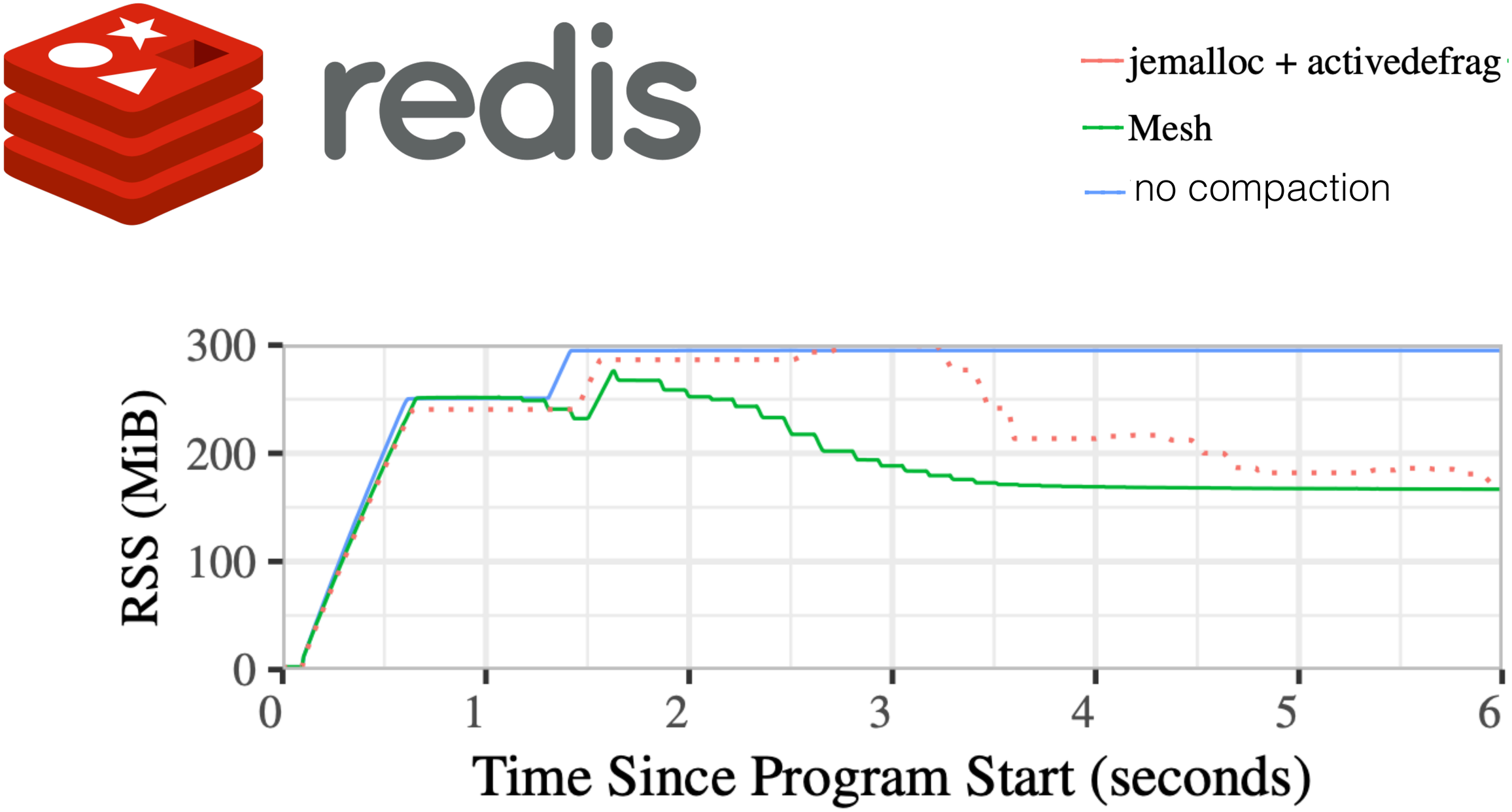
No recompilation

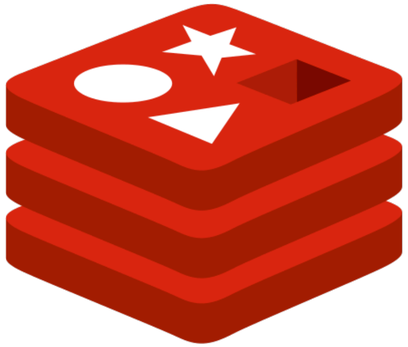
`LD_PRELOAD` and go

17% heap size reduction

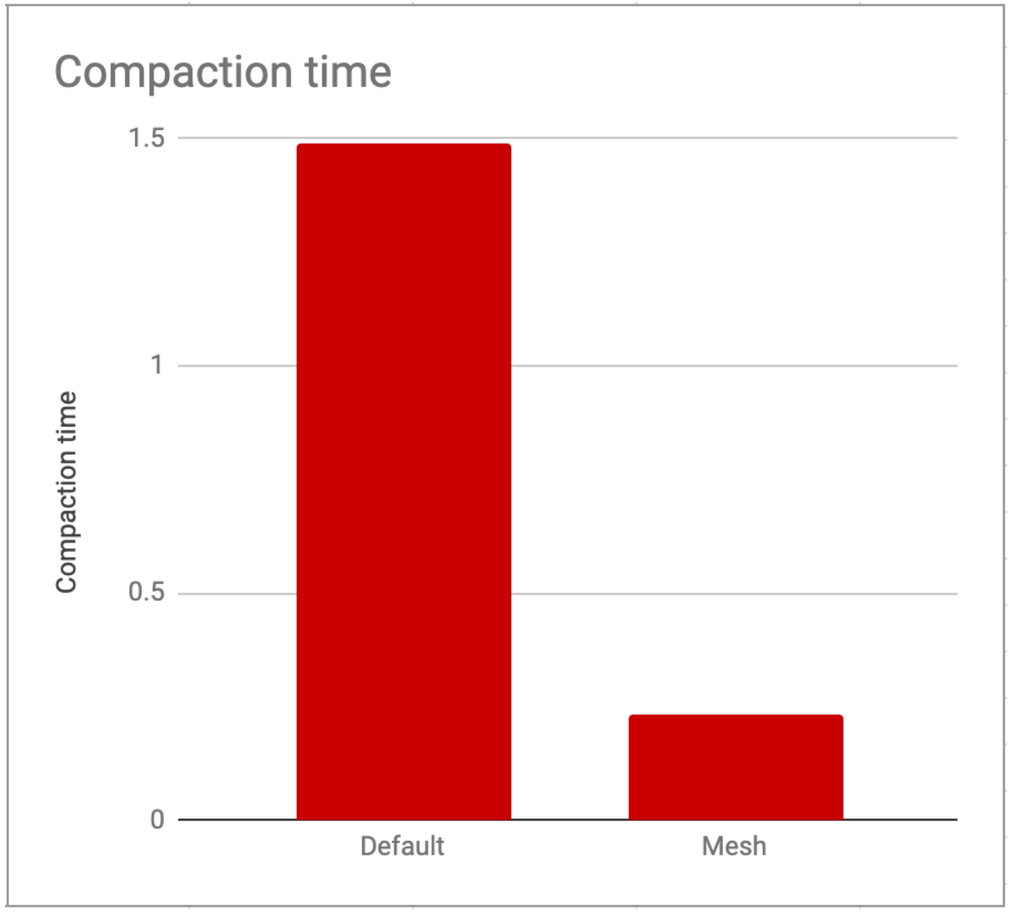
< 1% performance overhead




redis
jemalloc + activedefrag
Mesh
no compaction
RSS (MiB)
300
200
100
0
0
1
2
3
4
5
6
Time Since Program Start (seconds)

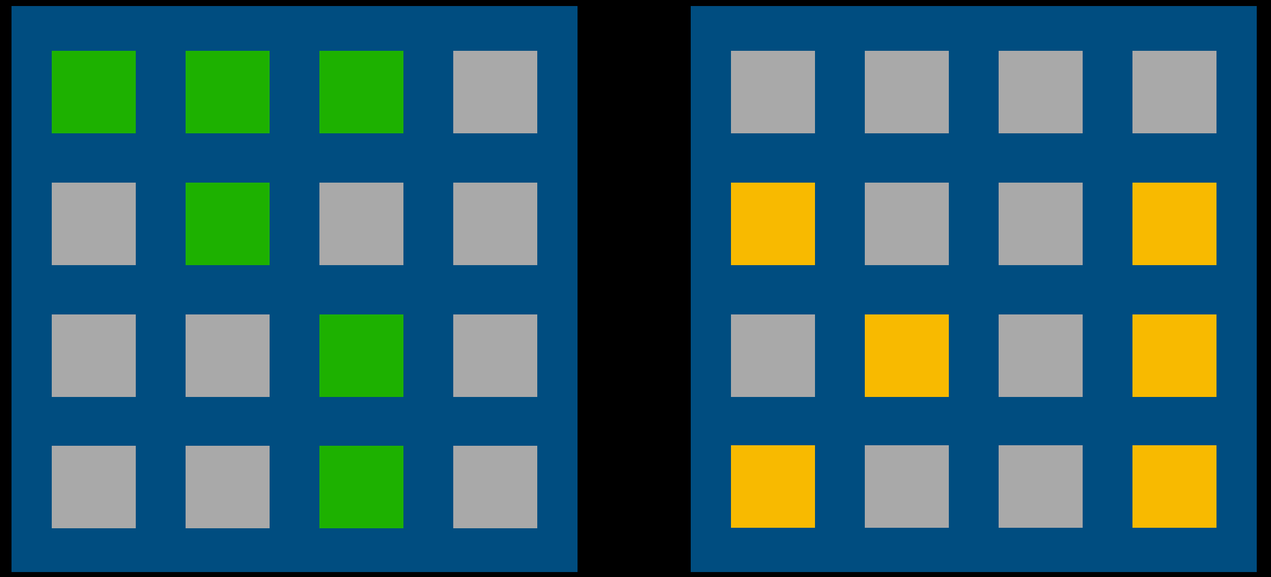


redis



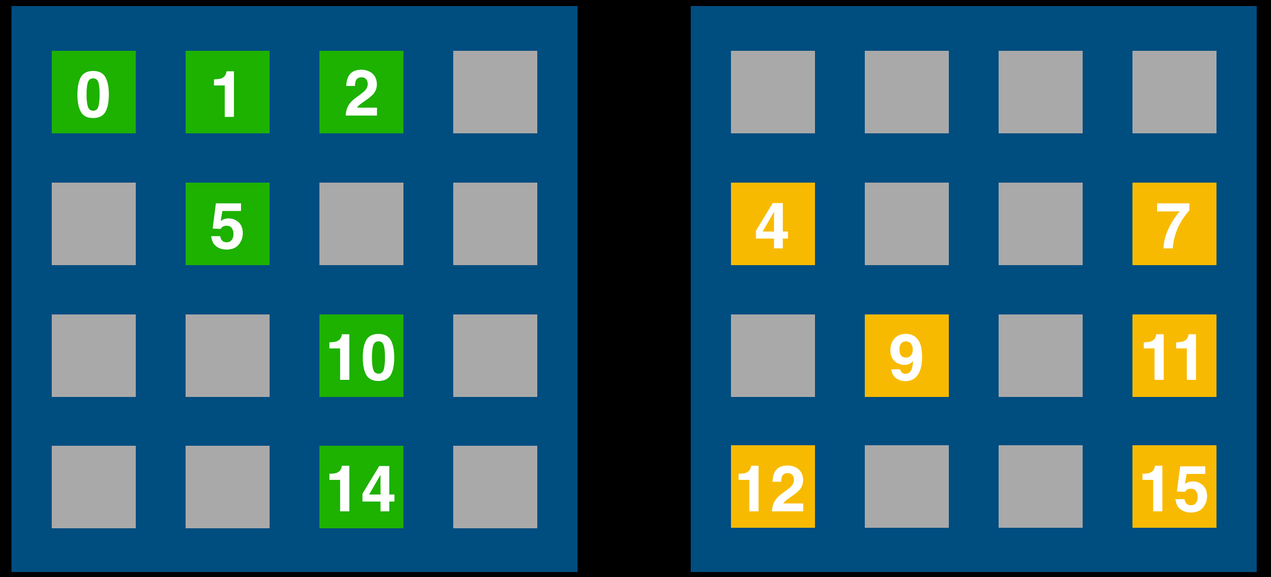
5x reduction in time spent compacting

Meshing: *compaction without (virtual!) relocation*



Pages are **Meshable** when they:

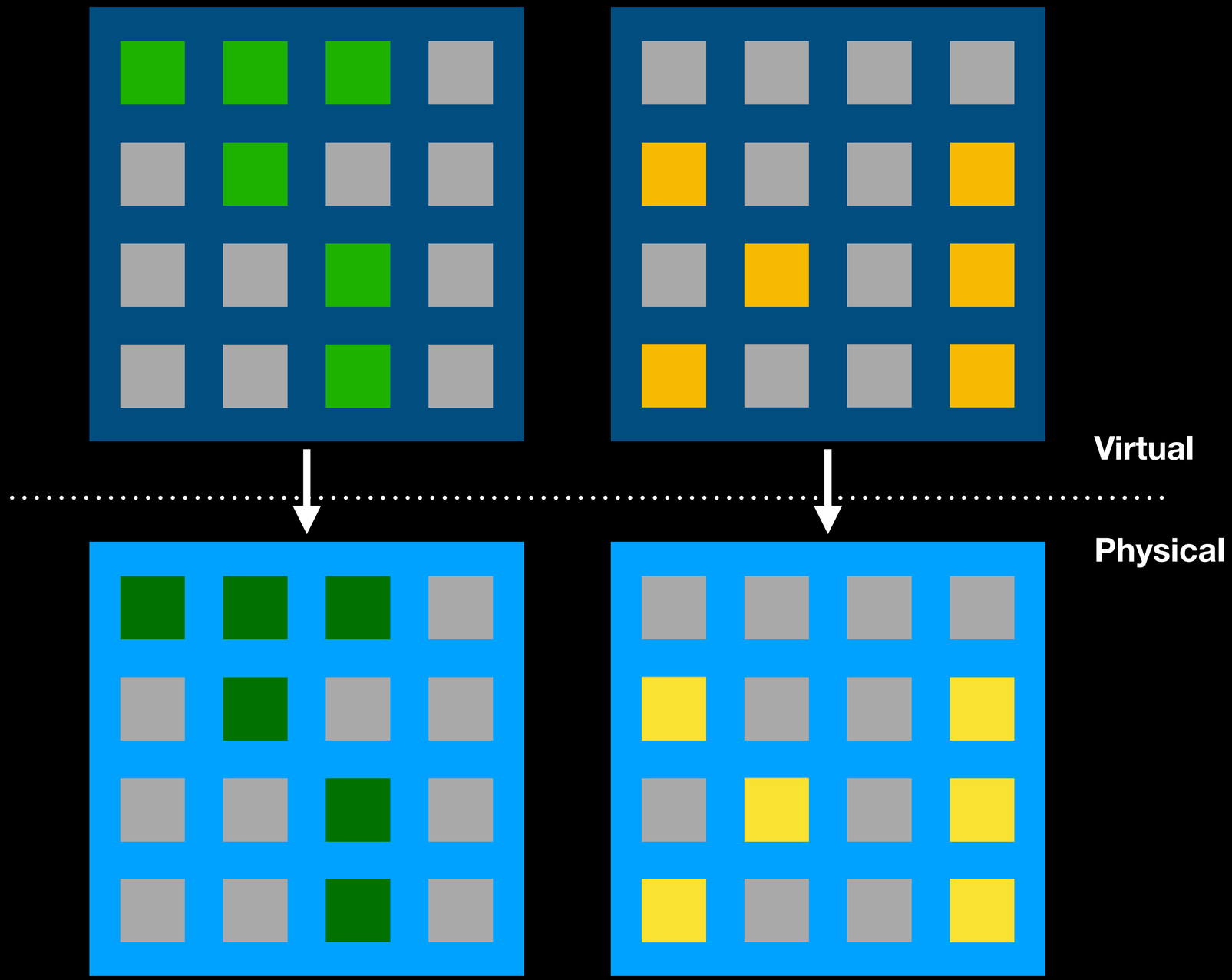
- Hold objects of the same size class



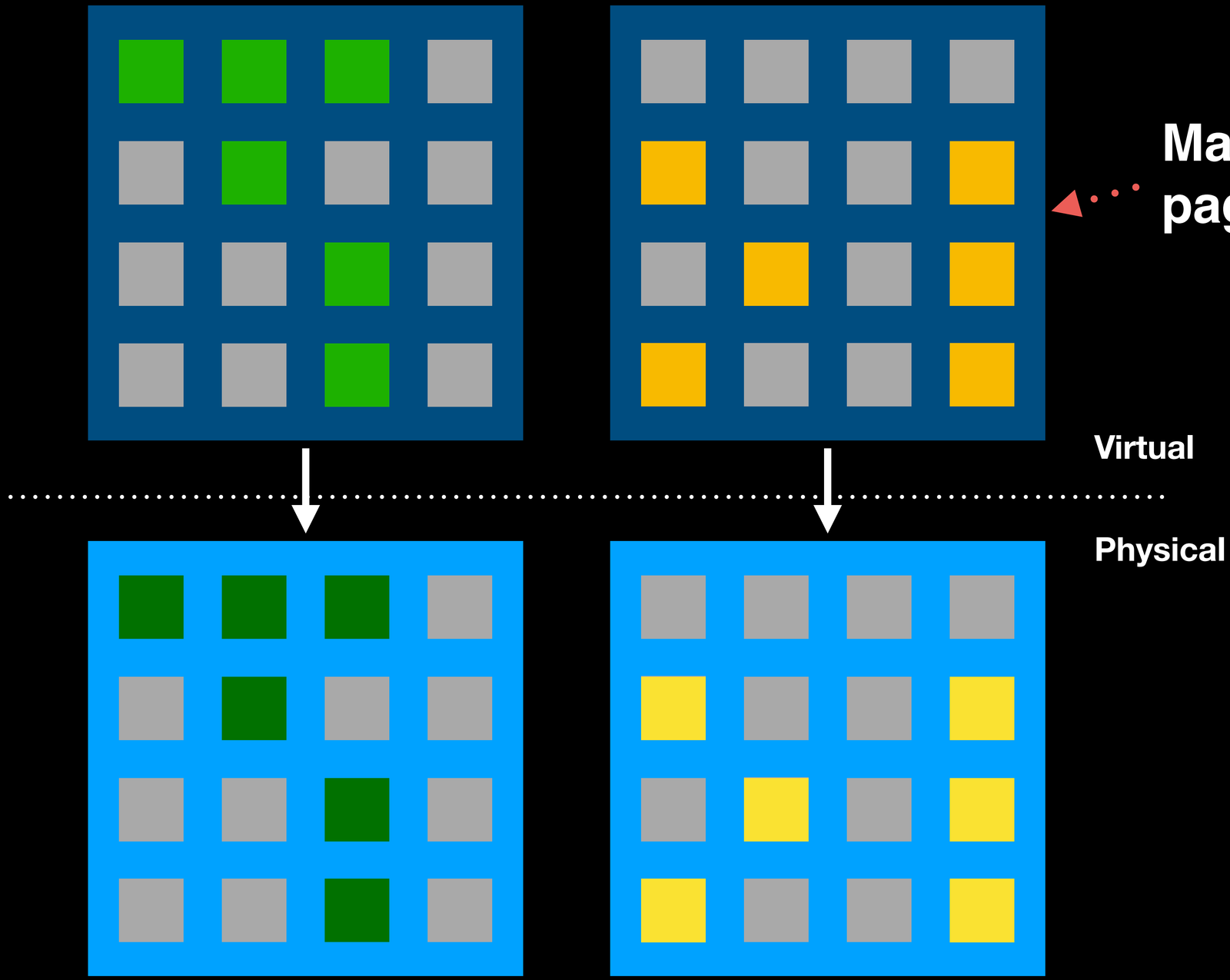
Pages are **Meshable** when they:

- Hold objects of the same size class
- Have non-overlapping object offsets

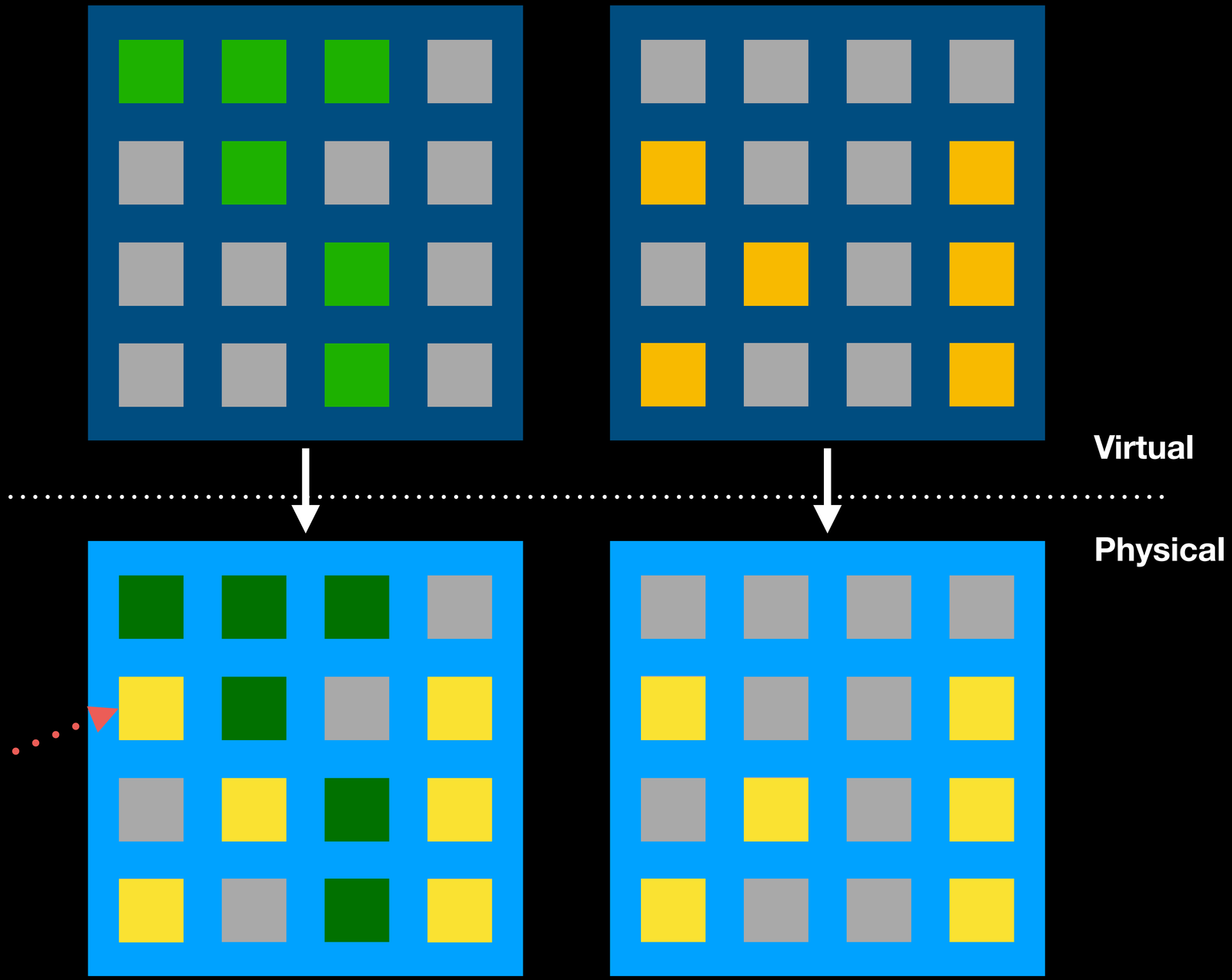
Meshing



Meshing

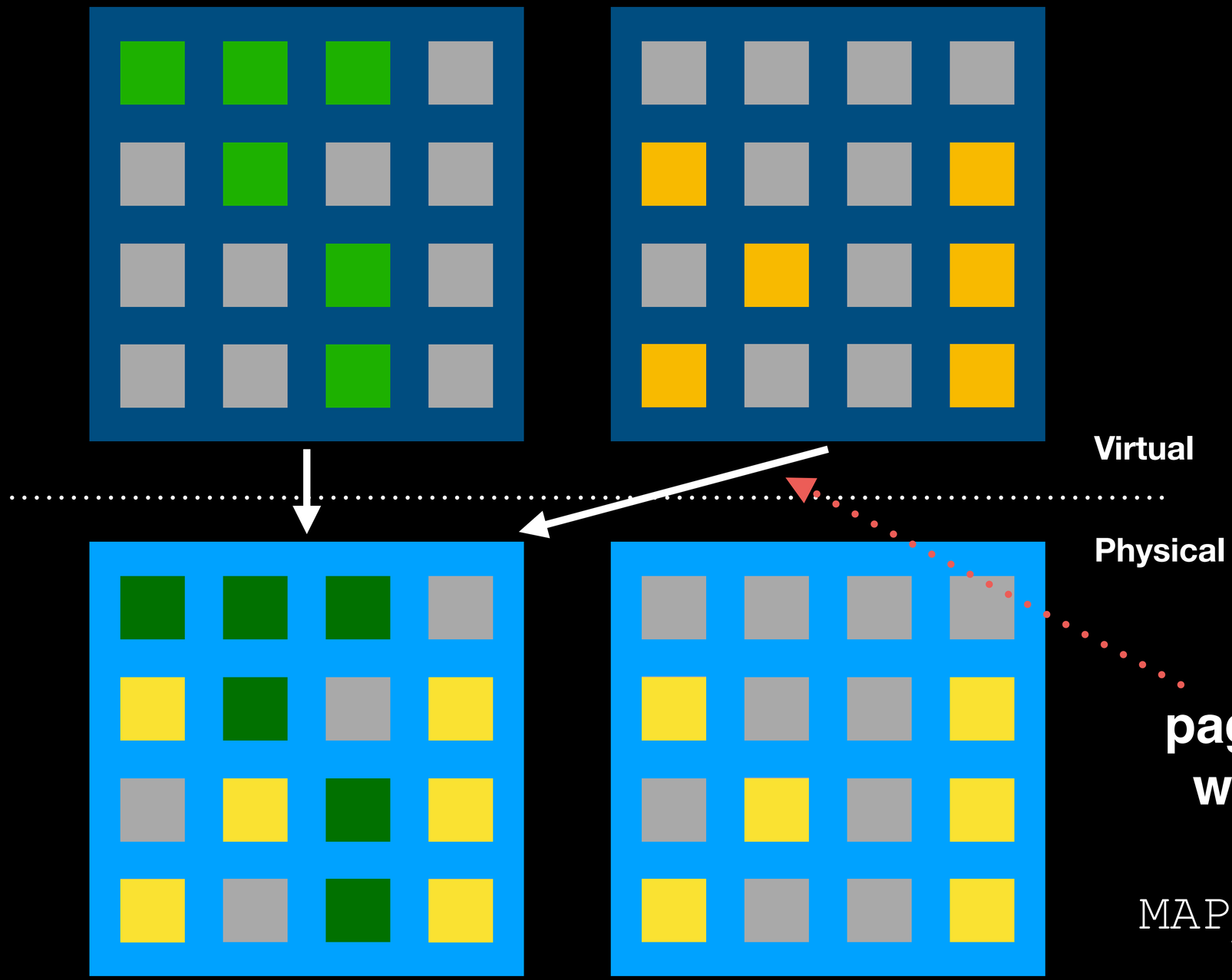


Meshing



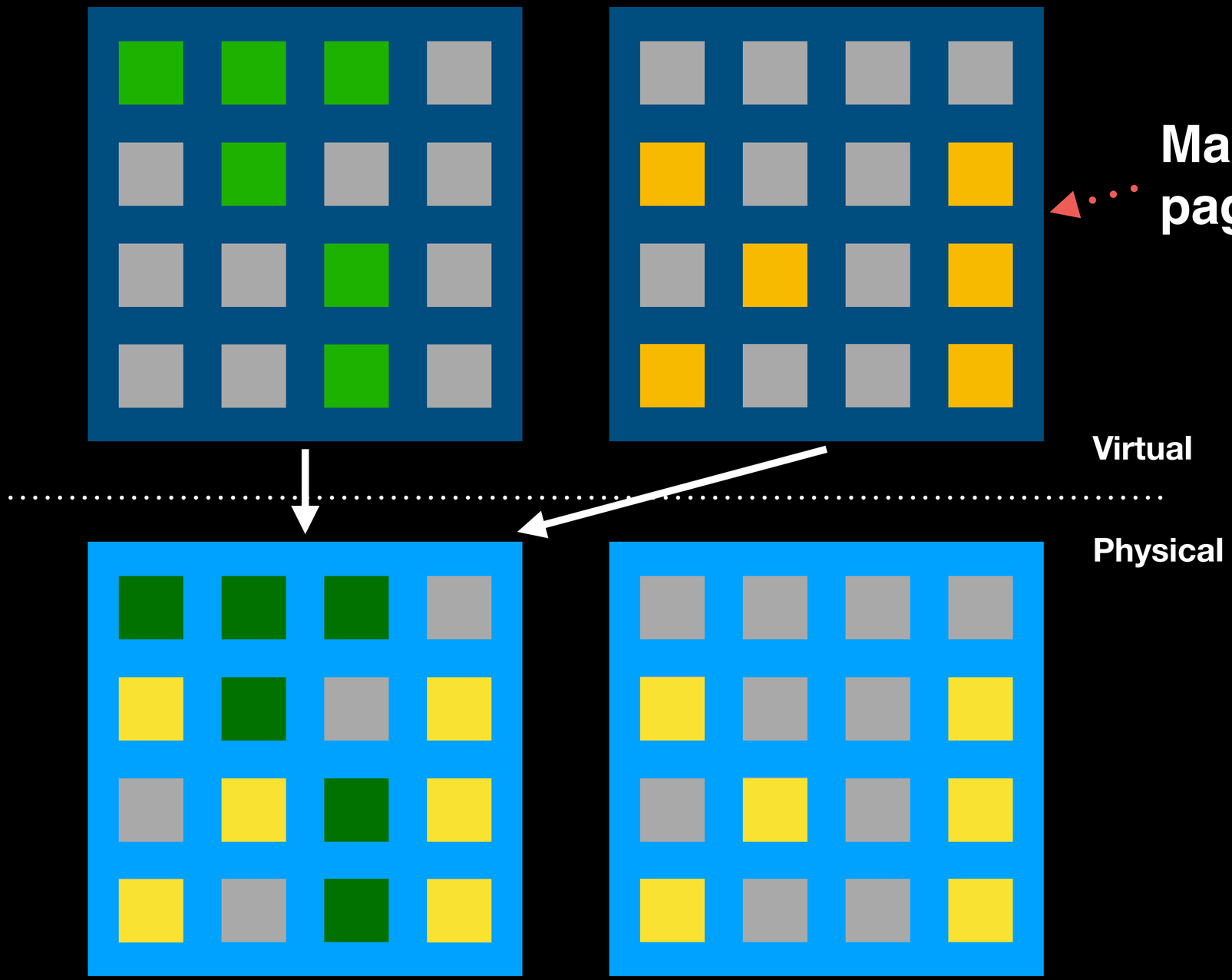
Copy (maintaining offsets)

Meshing

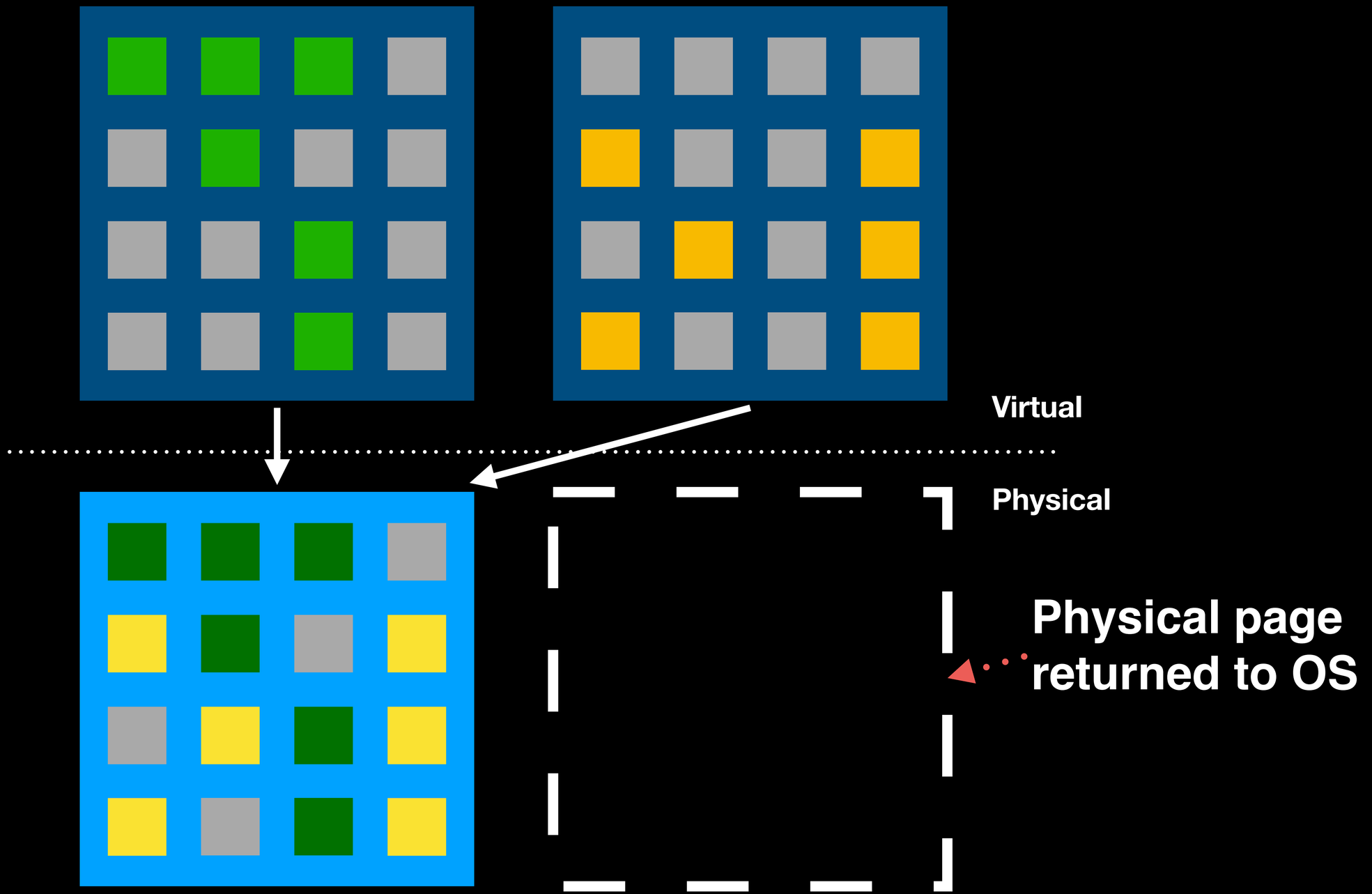


Update page tables with `mmap`, `memfd`, `MAP_SHARED`

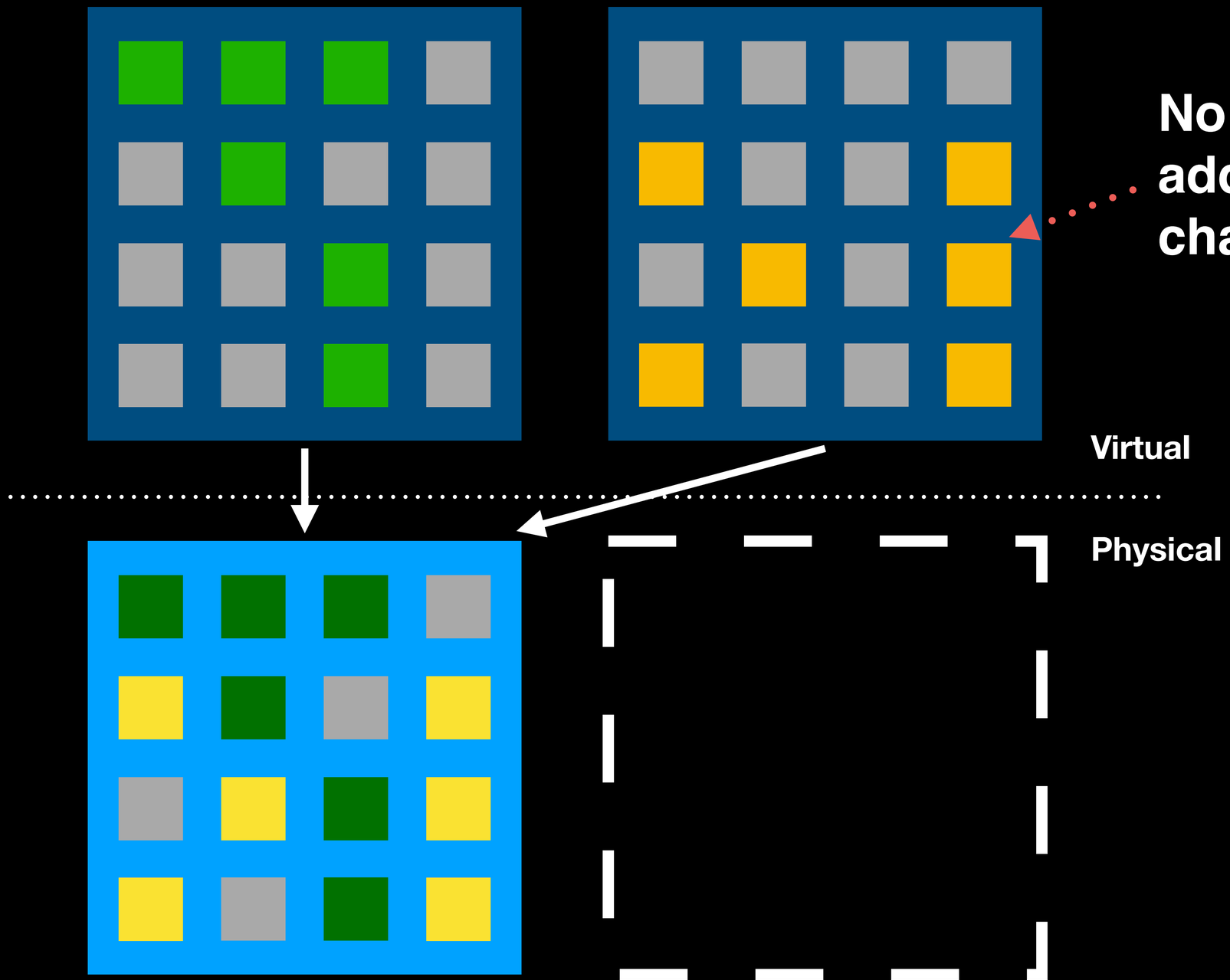
Meshing



Meshing



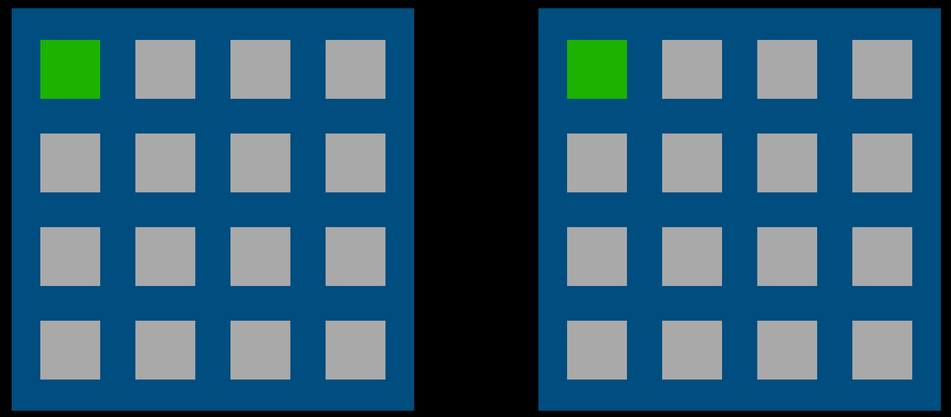
Meshing



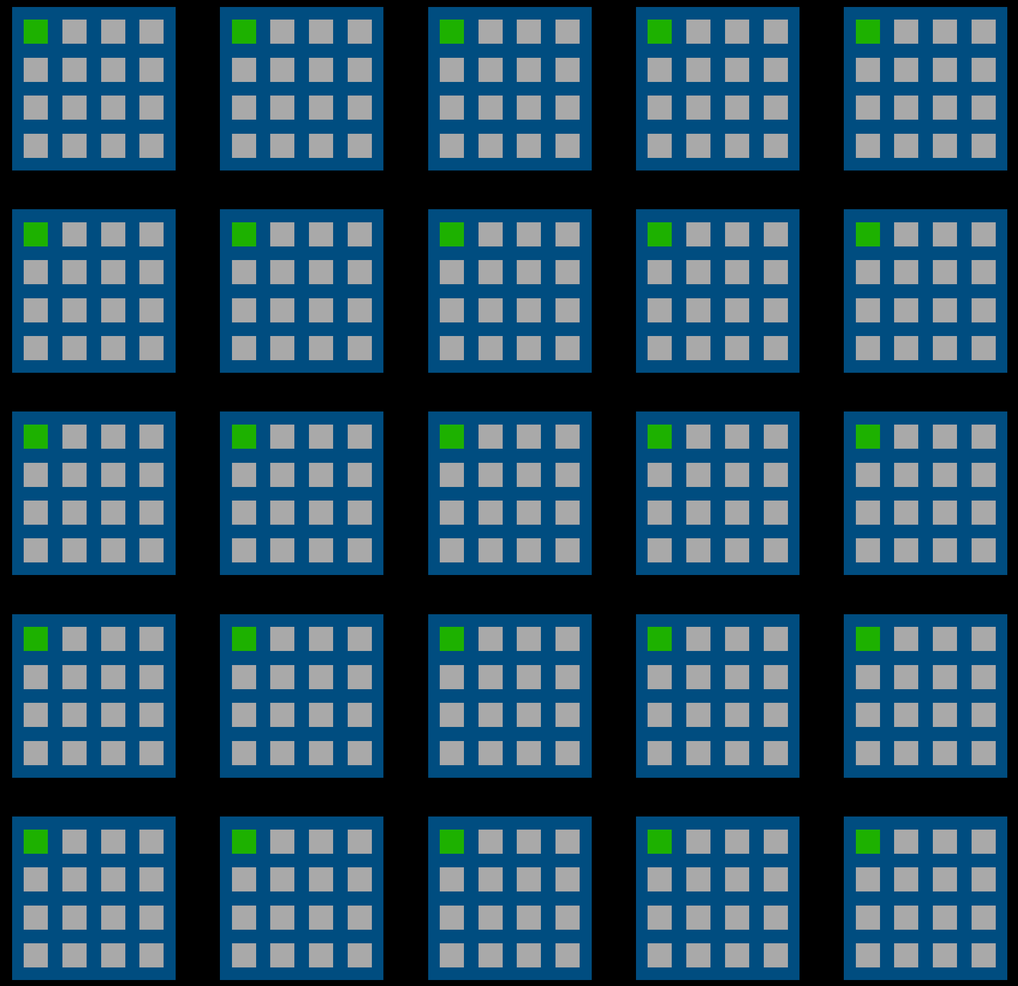
Worst Case:

Worst Case:

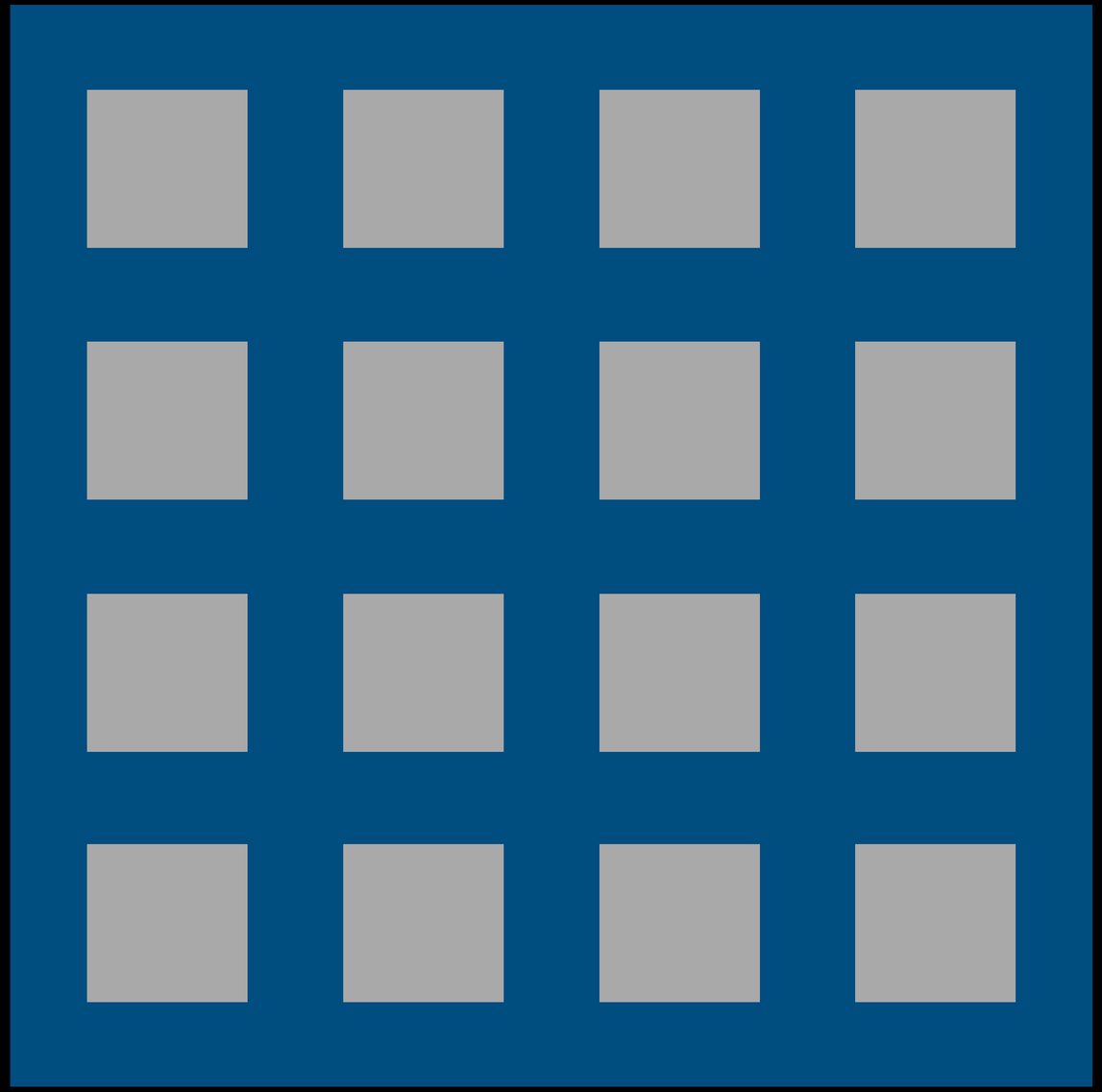
low occupancy, non-meshable pages



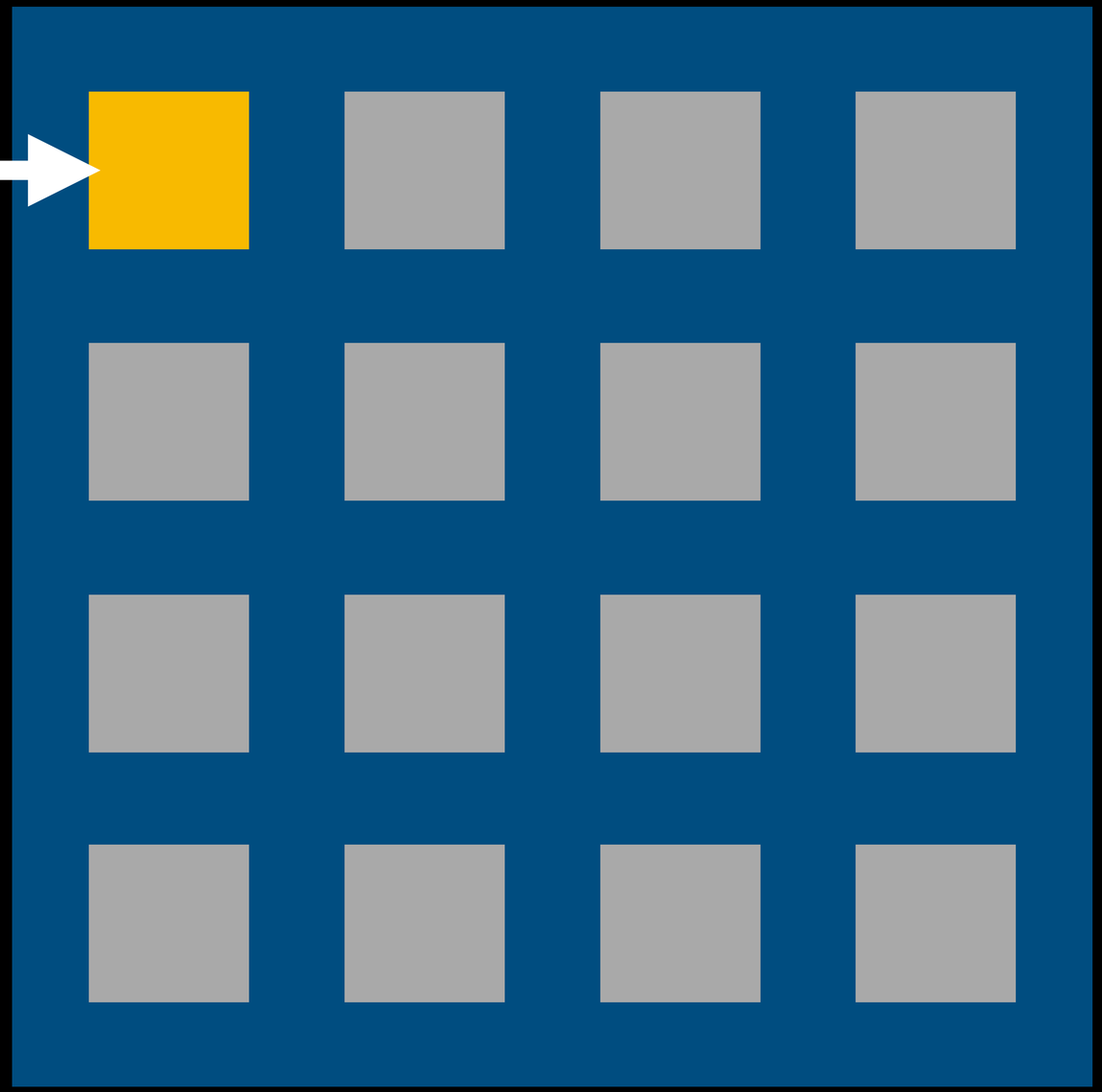
Worst Case:
many
low occupancy,
non-meshable
pages



Standard allocators

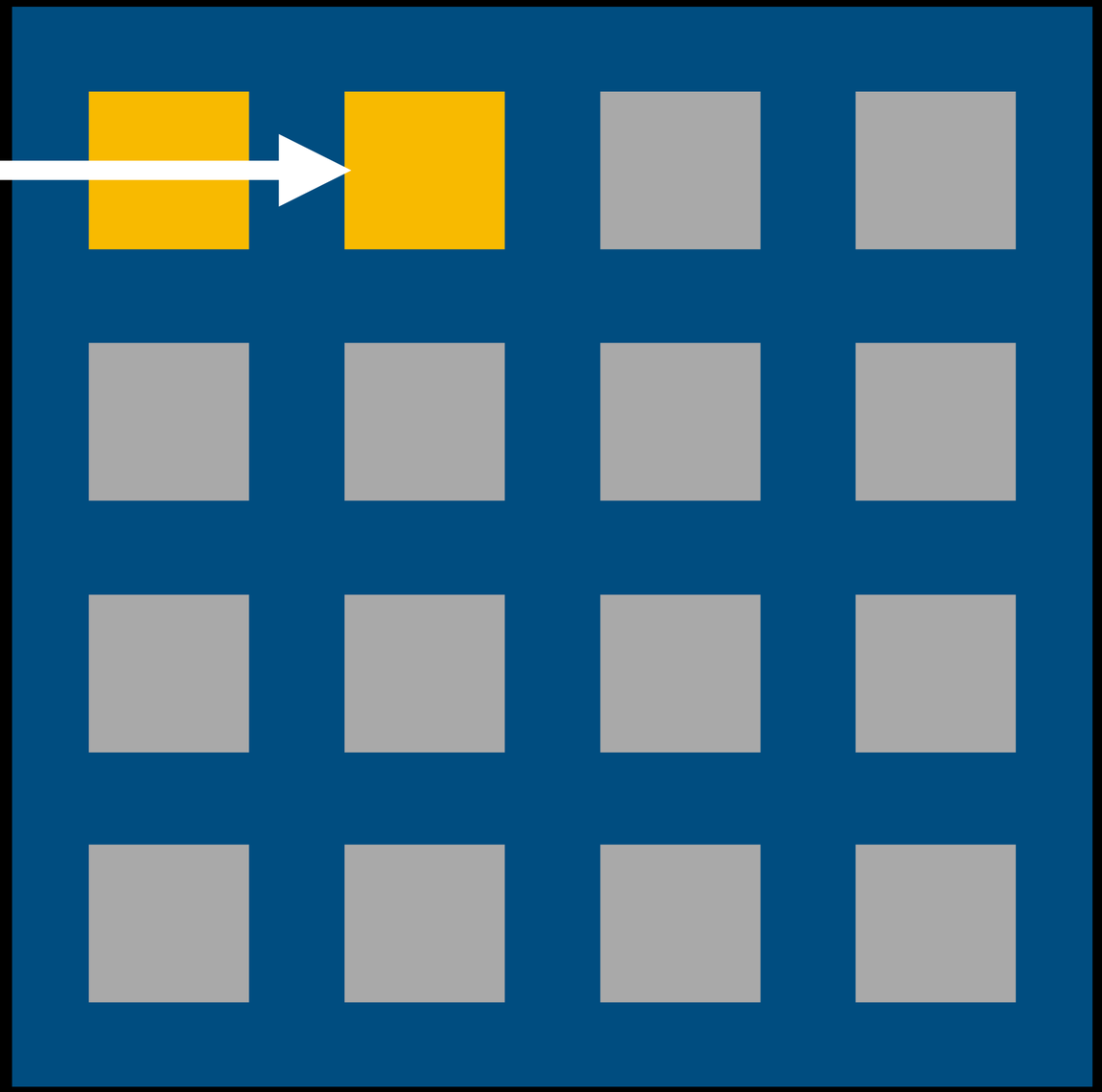


```
malloc(256)
```



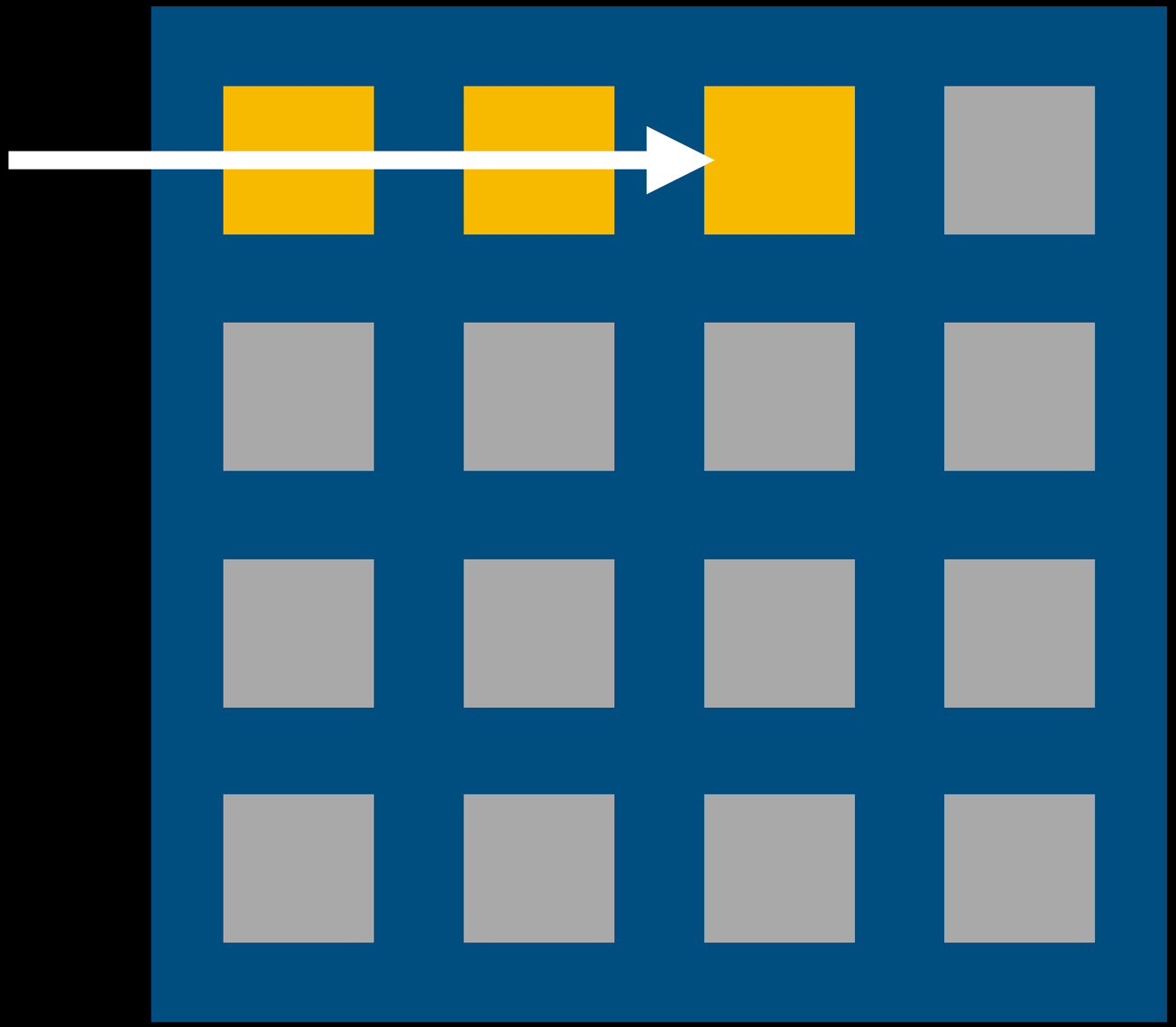
Standard allocators

```
malloc(256)
```



Standard allocators

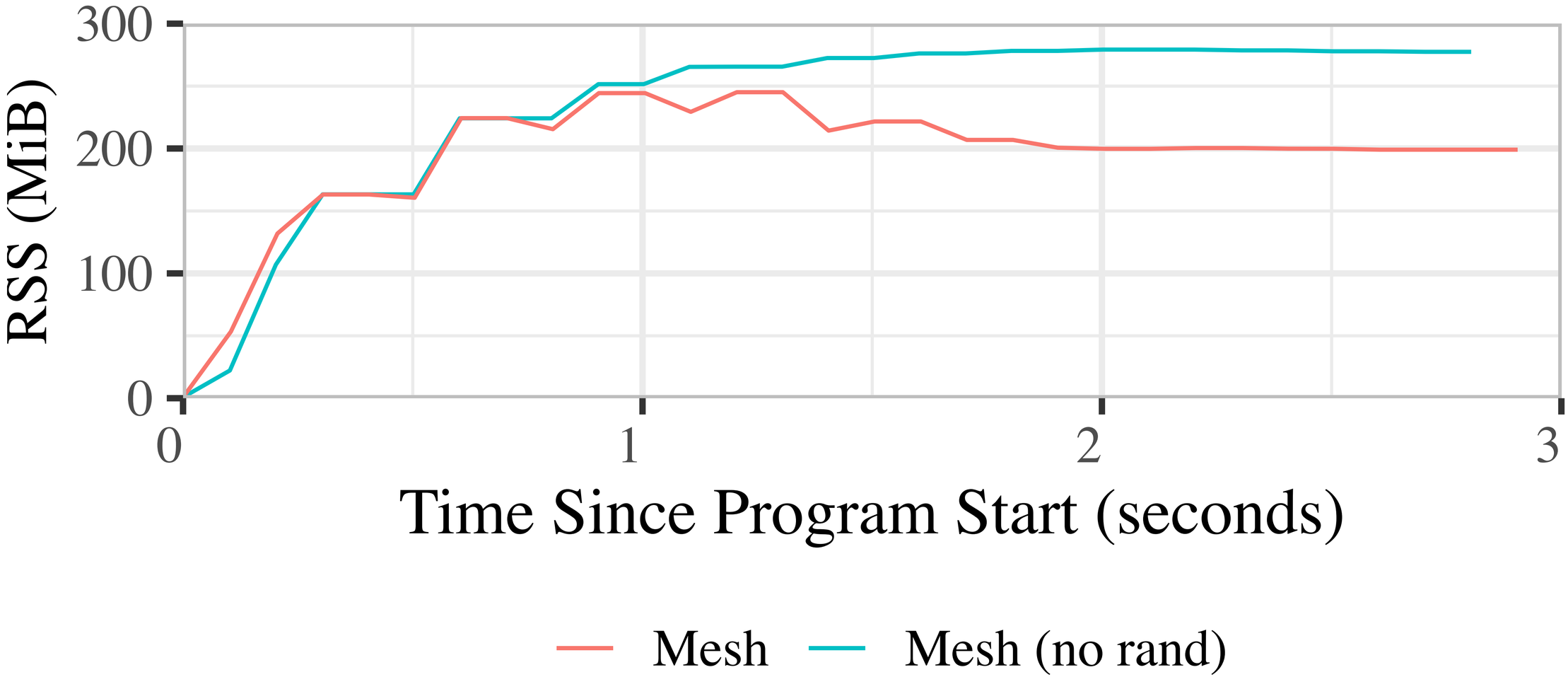
```
malloc(256)
```



Standard allocators

Mesh uses **randomization** to ensure live objects are uniformly distributed

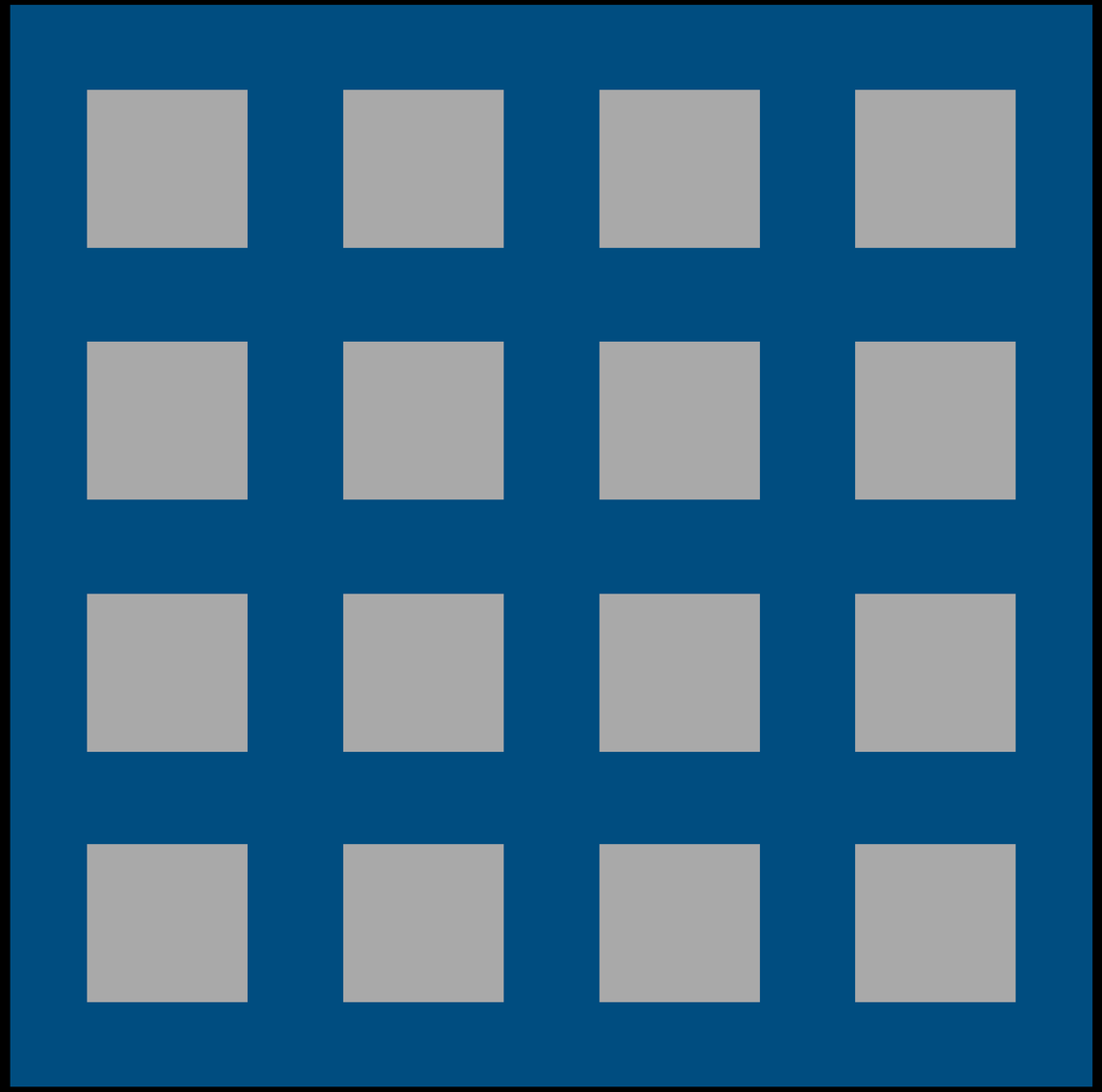
Regular allocation patterns are real



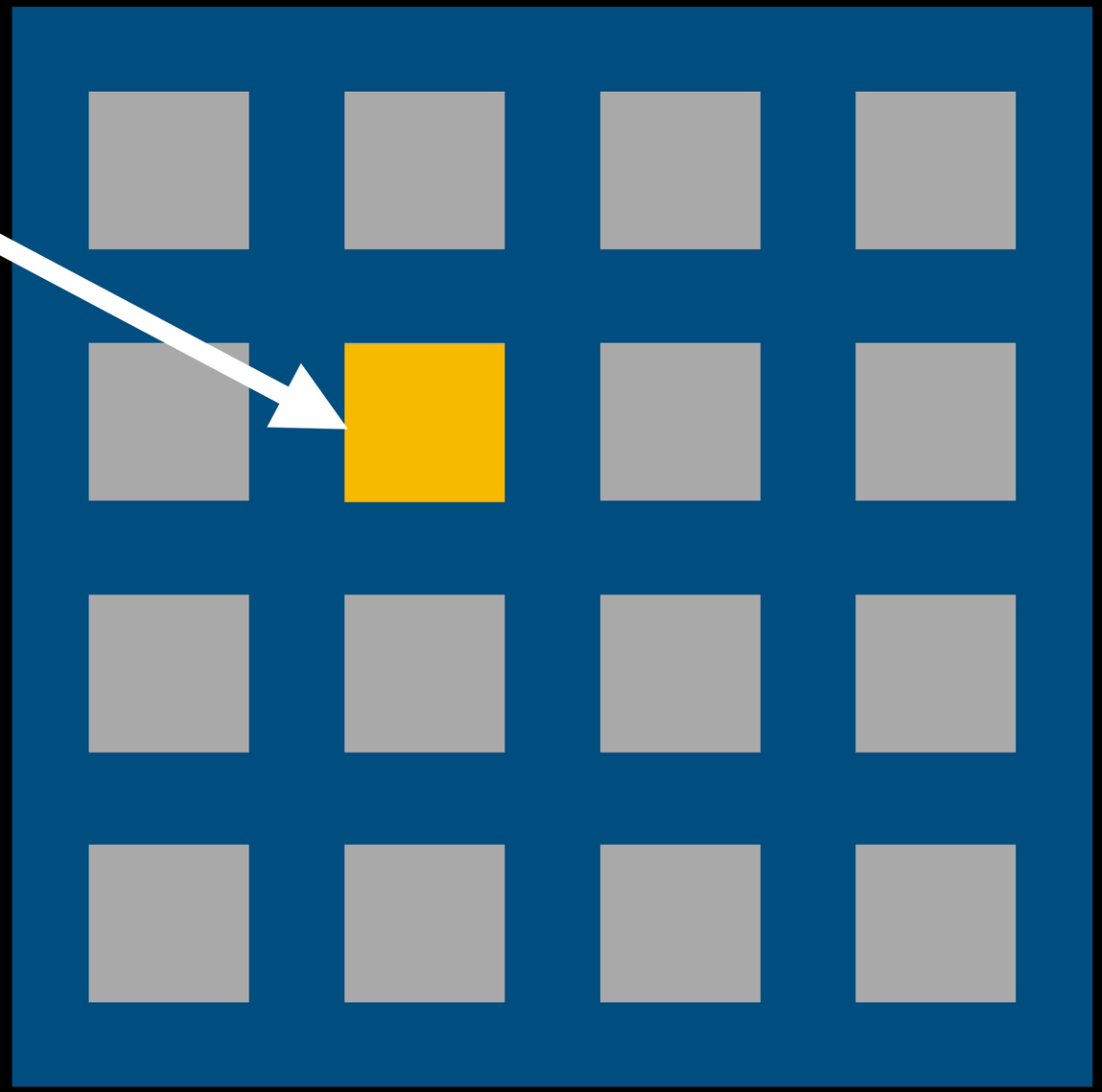
How to **randomize** allocation?

Random probing:

```
while true:
   if rand_off().is_free:
      return rand_off
```



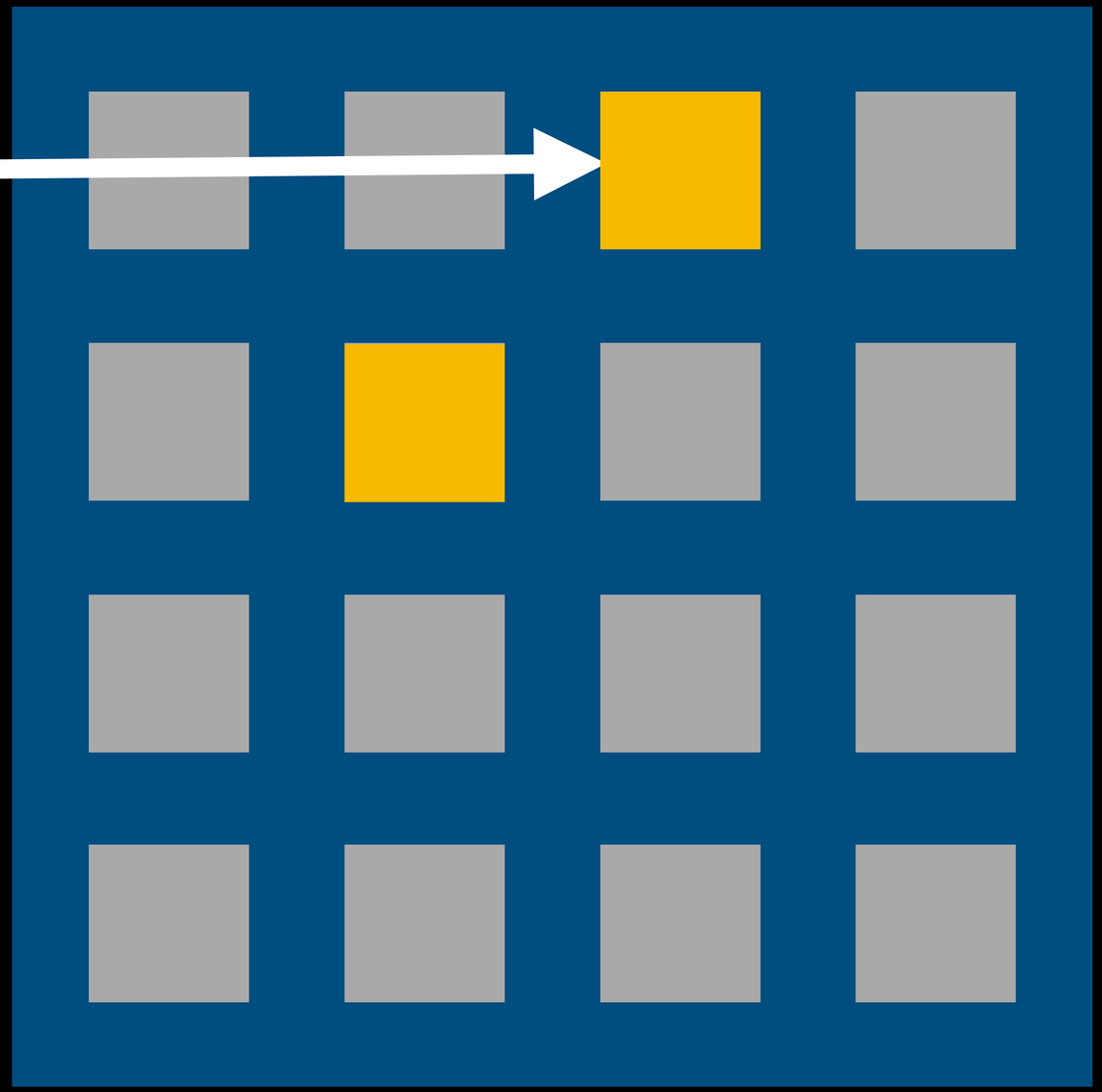
```
malloc(256)
```



Random probing:

```
while true:
   if rand_off().is_free:
      return rand_off
```

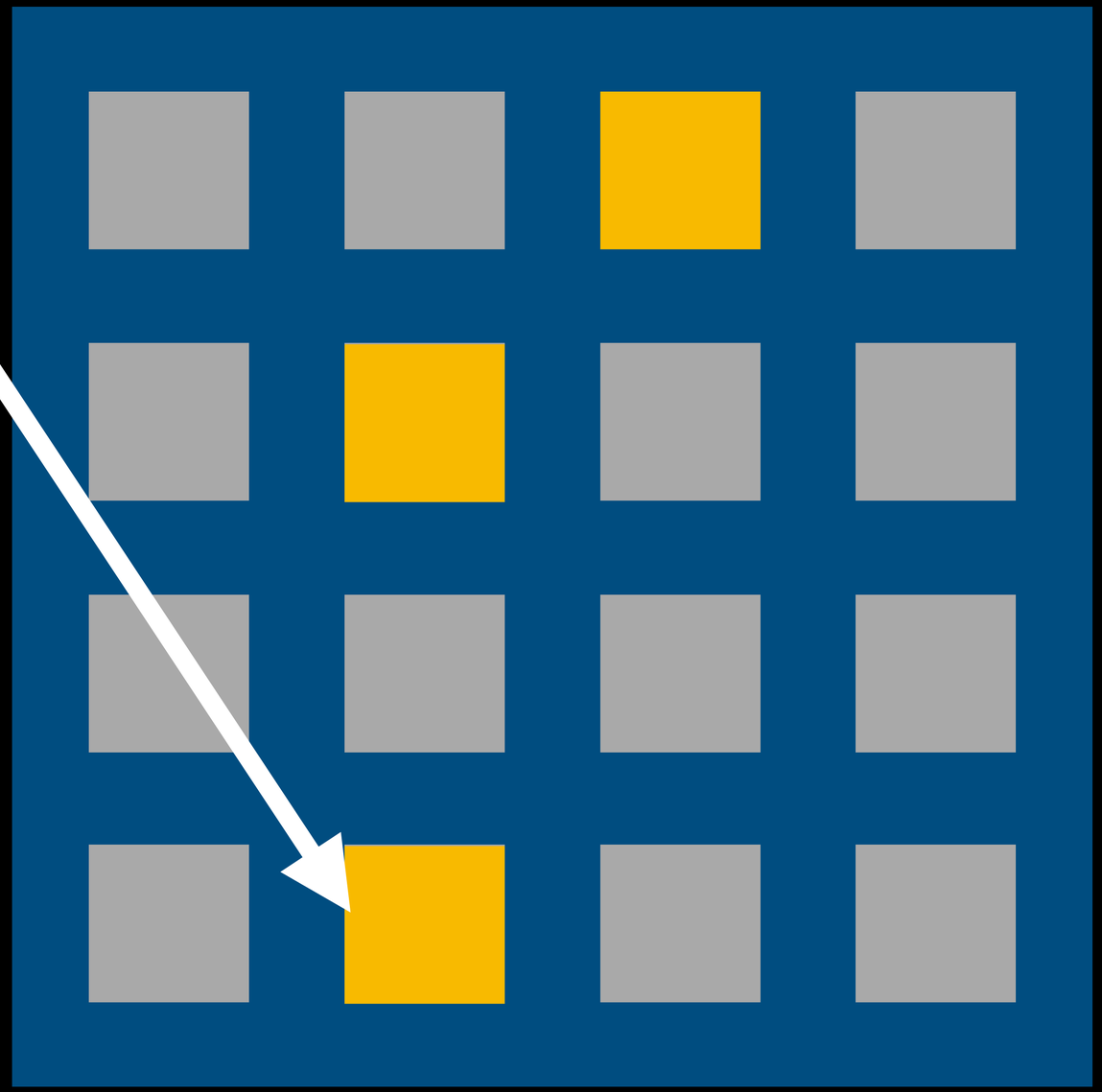
```
malloc(256)
```



Random probing:

```
while true:
   if rand_off().is_free:
      return rand_off
```

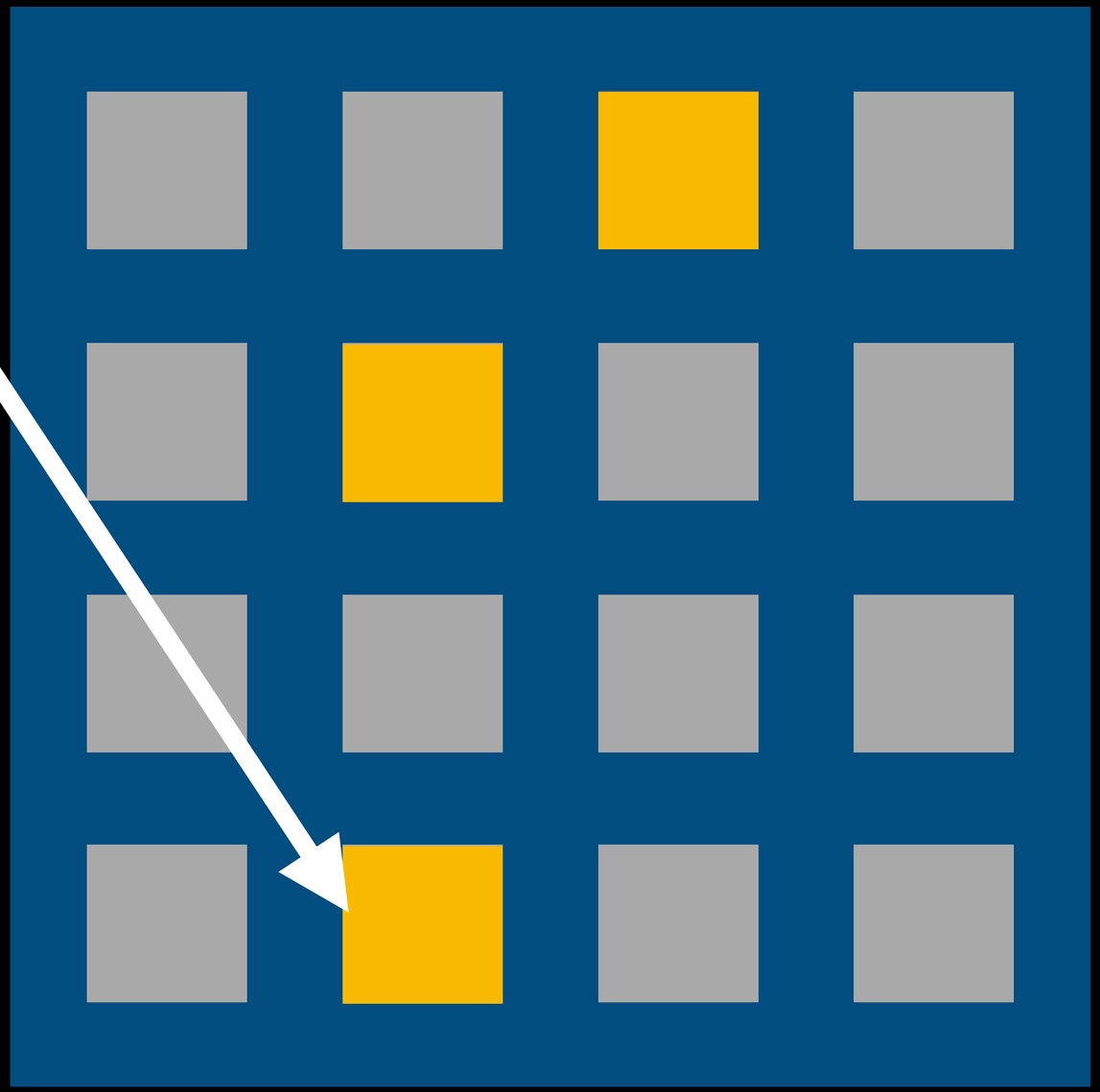
```
malloc(256)
```



Random probing:

```
while true:
   if rand_off().is_free:
      return rand_off
```

```
malloc(256)
```



Random probing:

```
while true:
   if rand_off().is_free:
      return rand_off
```

(DieHard [Berger & Zorn 2006])

Random probing fast in expectation *iff page occupancy is low*

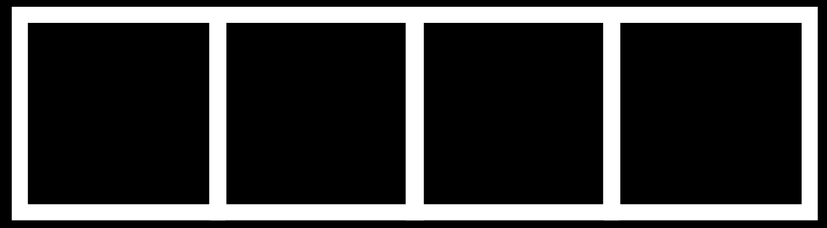
Random probing fast in expectation *iff page occupancy is low*

but this is at odds with minimizing heap size!

Shuffle Vector:
Fast randomized allocation + full page utilization

Shuffle Vector: Fast randomized allocation

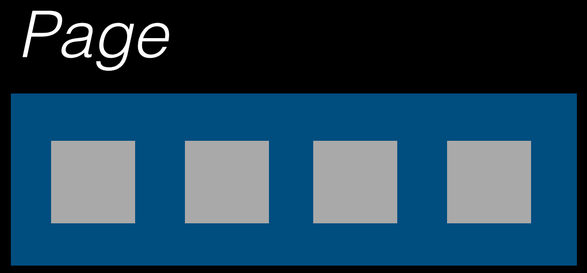
```
load
```

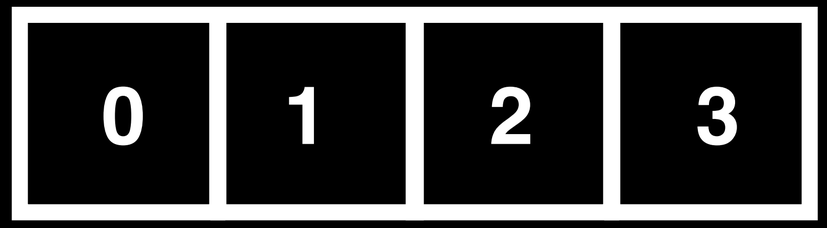


Thread-local shuffle vector

Shuffle Vector: Fast randomized allocation

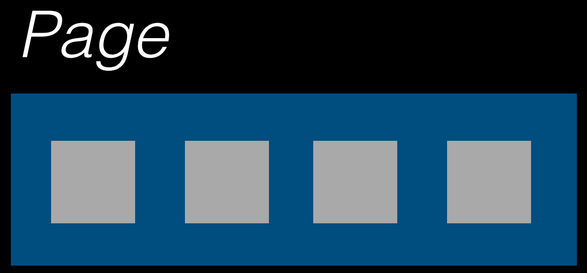
```
load
```

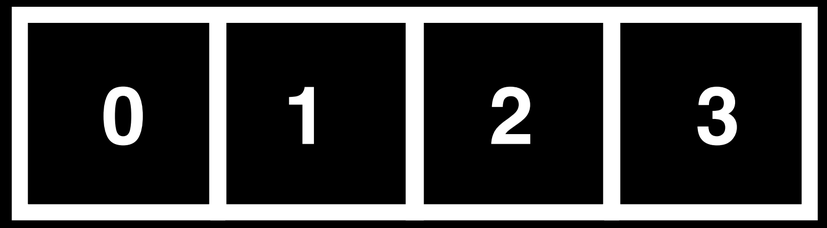




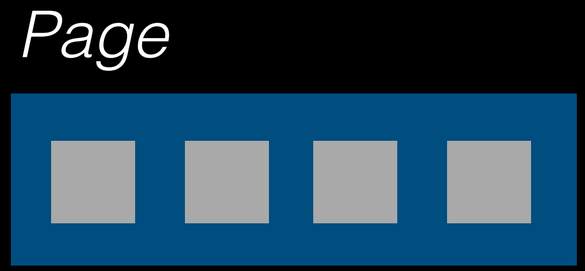
Thread-local shuffle vector

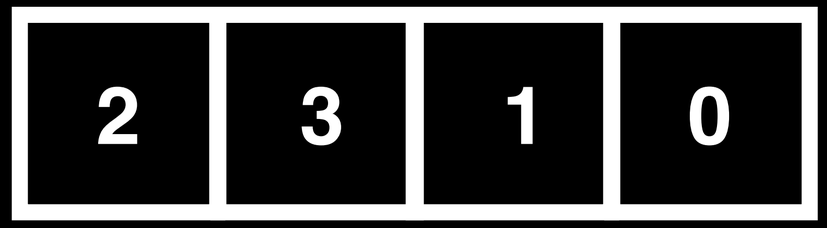
Shuffle Vector: Fast randomized allocation





Shuffle Vector: Fast randomized allocation

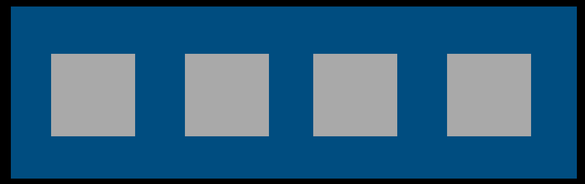


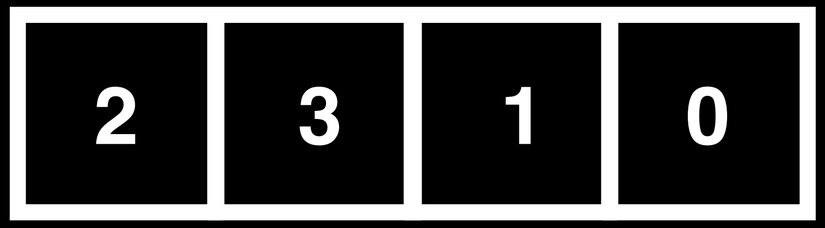


Shuffle Vector: Fast randomized allocation

`malloc()`

Page

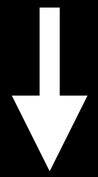




Thread-local shuffle vector

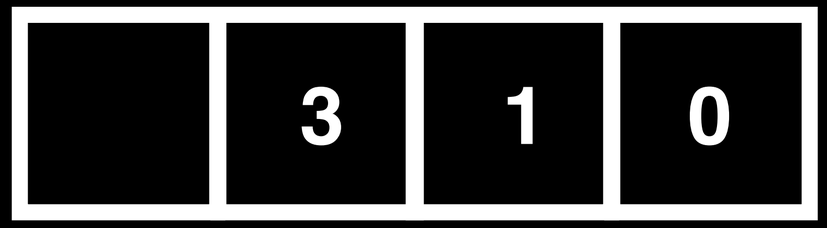
Shuffle Vector: Fast randomized allocation

```
malloc()
```



```
page_start +
2 * object_size
```

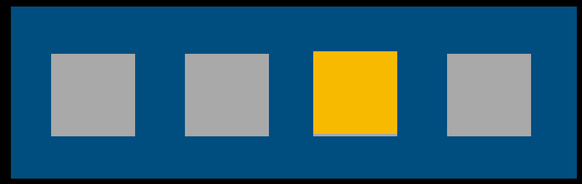
Page

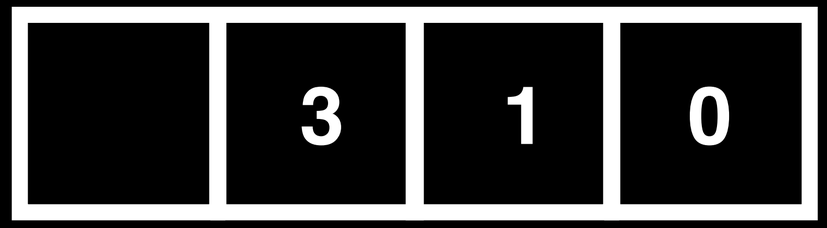


Thread-local shuffle vector

Shuffle Vector: Fast randomized allocation

```
free(2)
```

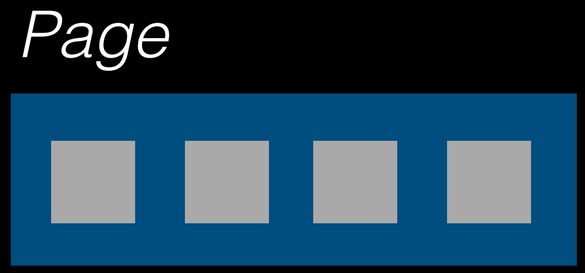


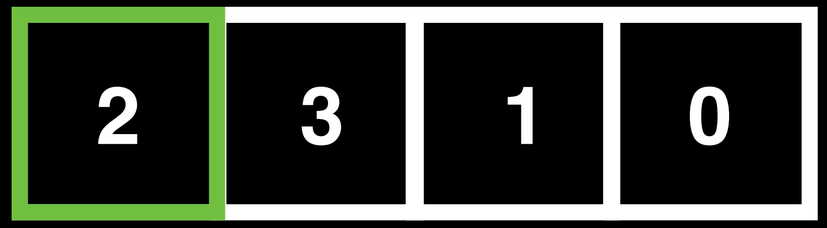


Thread-local shuffle vector

Shuffle Vector: Fast randomized allocation

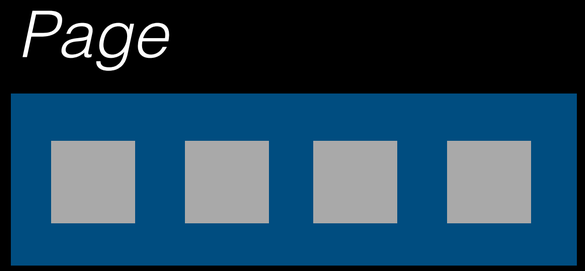
```
free( )
```



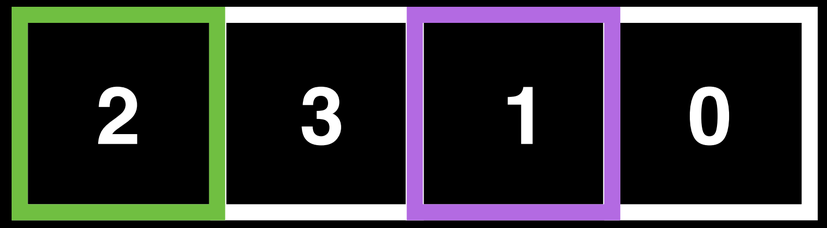


Thread-local shuffle vector

Shuffle Vector: Fast randomized allocation



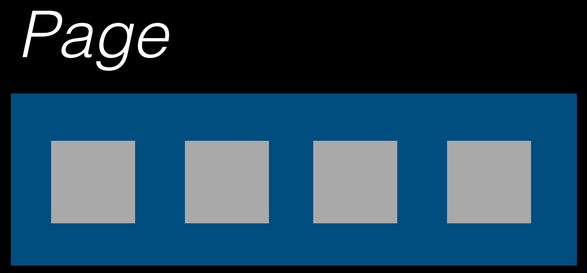

```
free( )

shuffle_one(
```



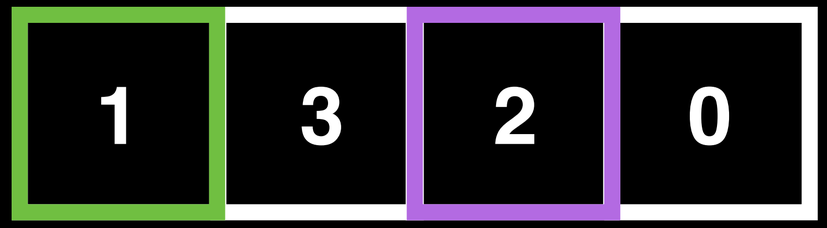

```
)
```

Thread-local shuffle vector

Shuffle Vector: Fast randomized allocation




```
free( )

shuffle_one(
```

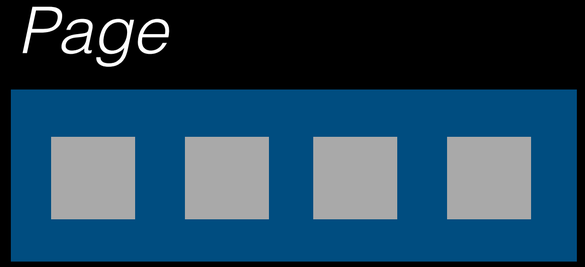


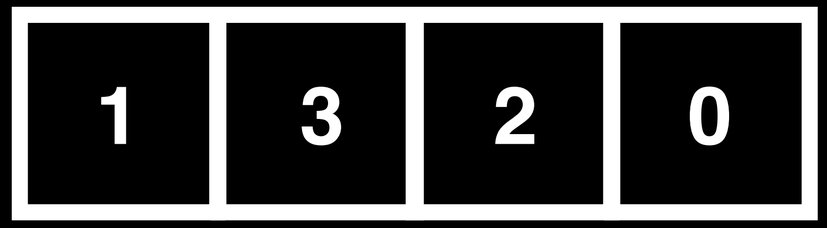

```
)
```

Thread-local shuffle vector

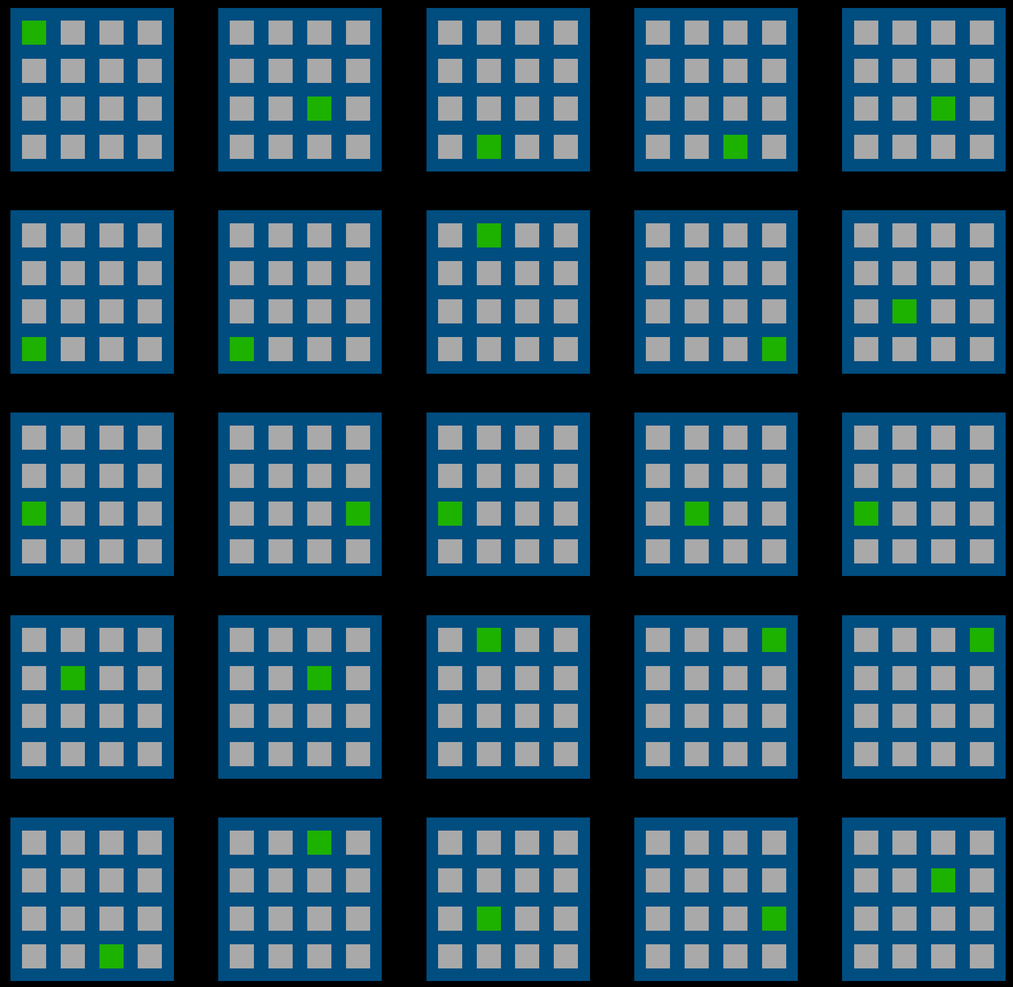
Shuffle Vector: Fast randomized allocation

```
free( )
```





Thread-local shuffle vector



All
Pages
Meshable

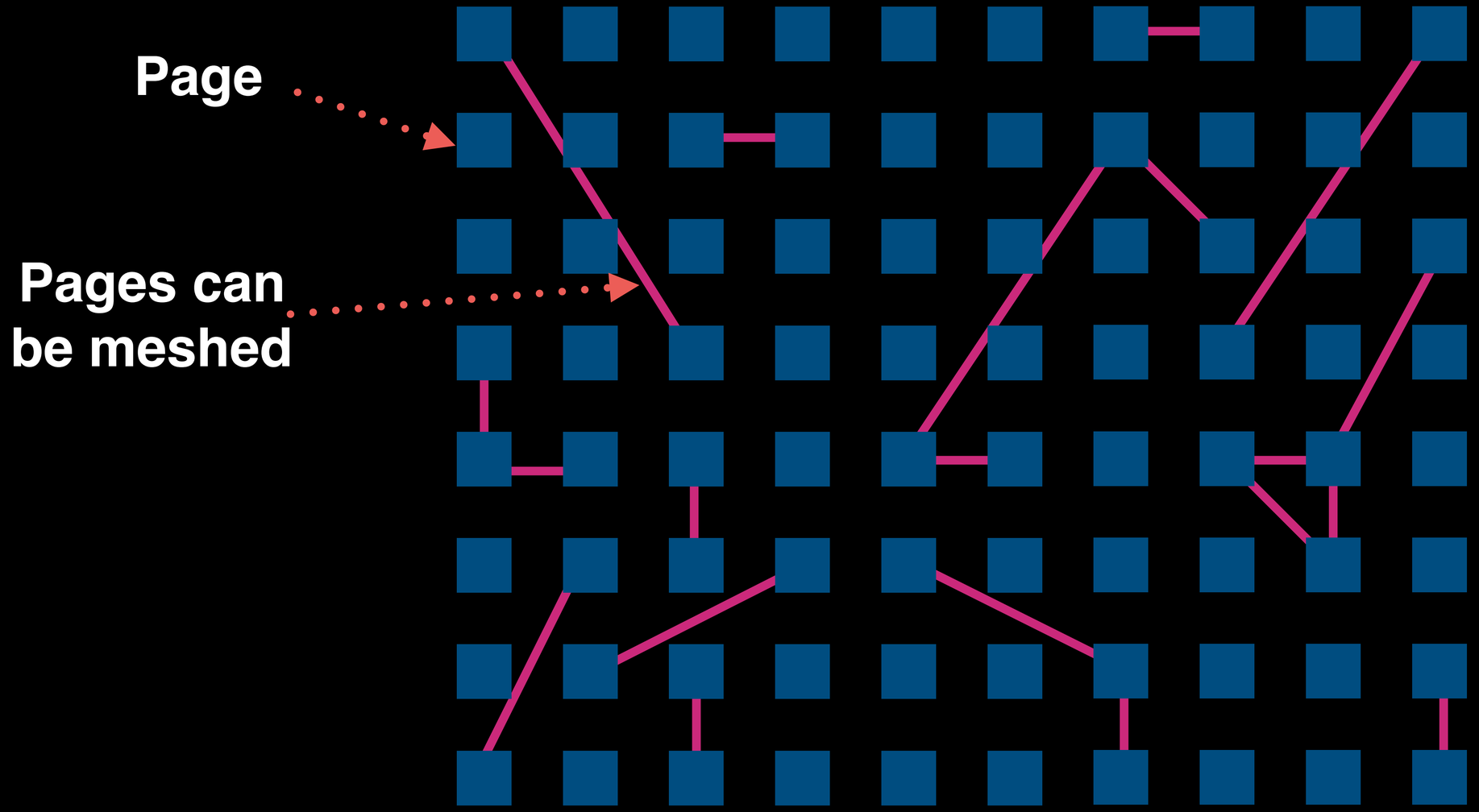
Finding pages to Mesh

Problem: Find meshing that releases maximum number of pages

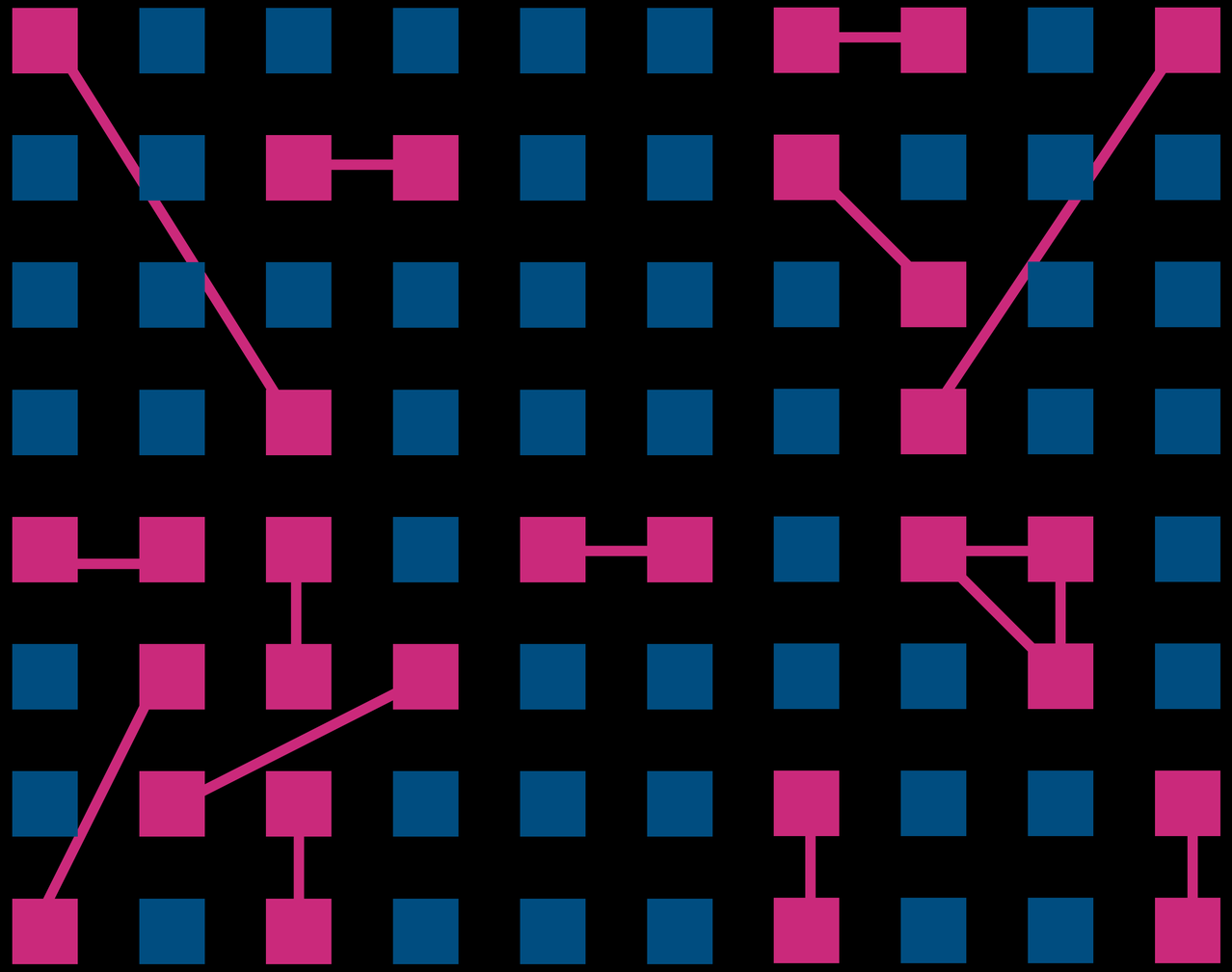
Problem: Find meshing that releases maximum number of pages

Run in the `free()` slowpath
At most once every 100 ms
Treat each size class independently

Problem: Find meshing that releases the maximum number of pages



MinCliqueCover



MinCliqueCover

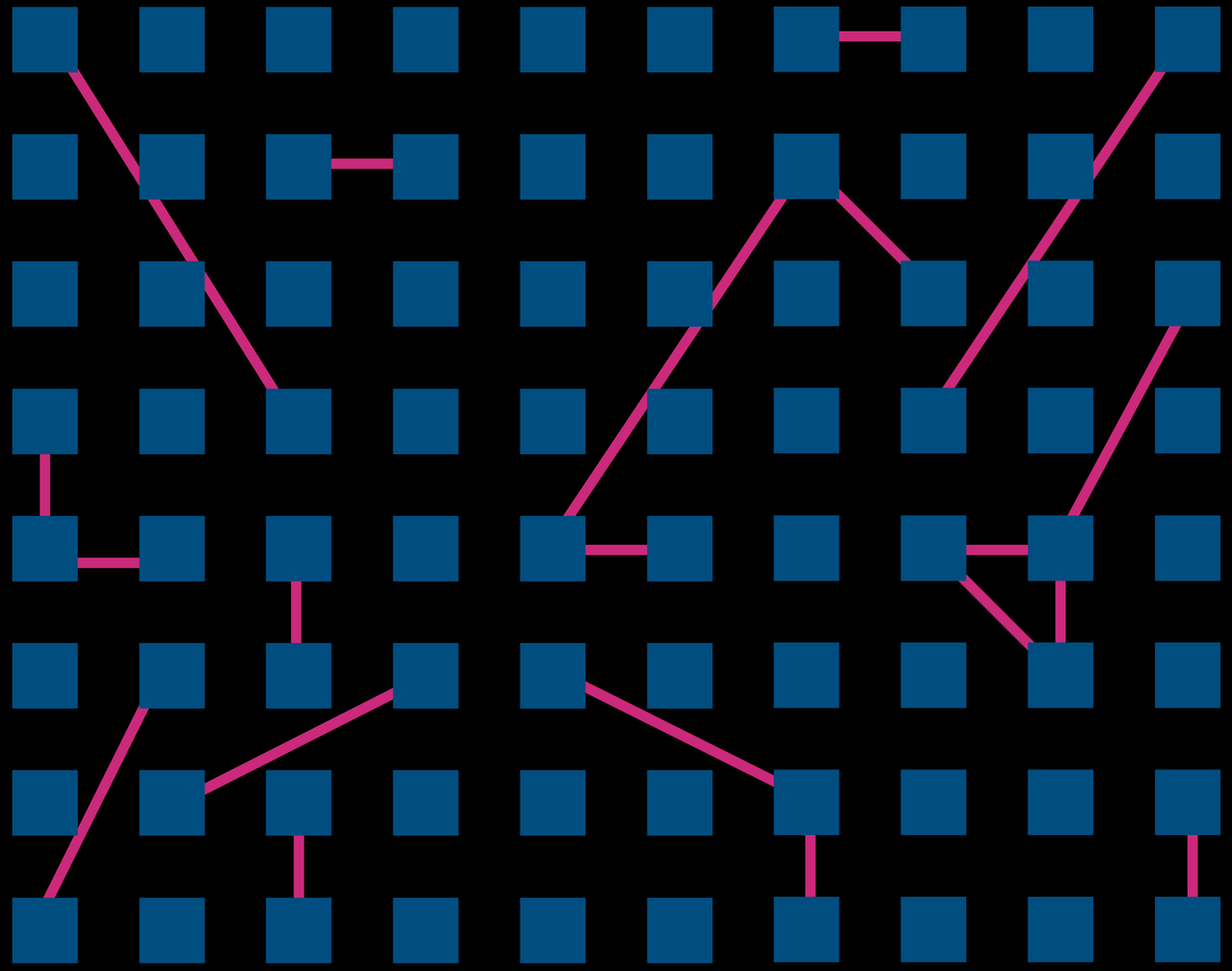
(NP-Complete)

MinCliqueCover

(NP-Complete)

BUT! Randomness ensures we can get away with solving simpler graph problem (Matching)

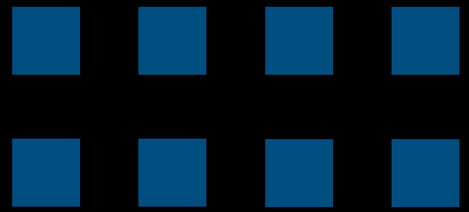
Wrinkle: building this graph would require RAM + time



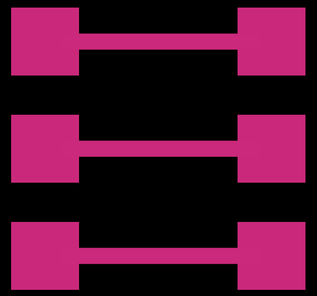
SplitMesher: approximates **Matching** without materializing meshing graph

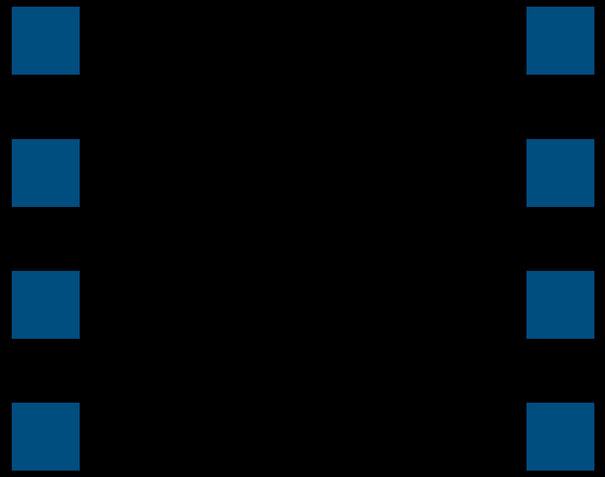
SplitMesher: approximates **Matching** without materializing meshing graph

Set of partially full pages

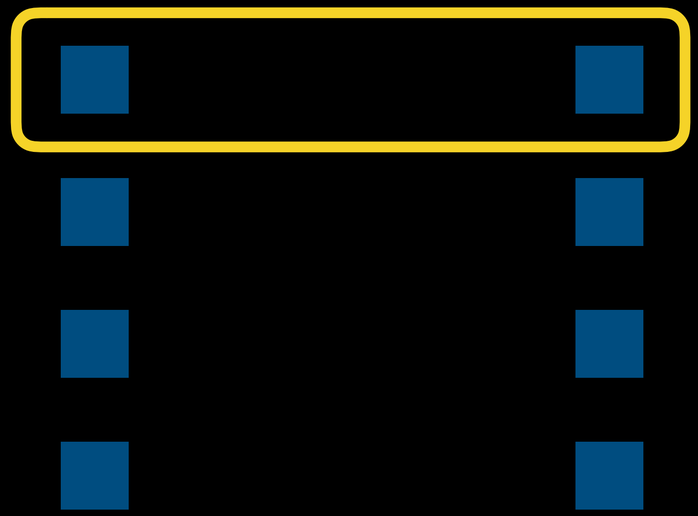


Pairs of meshable pages

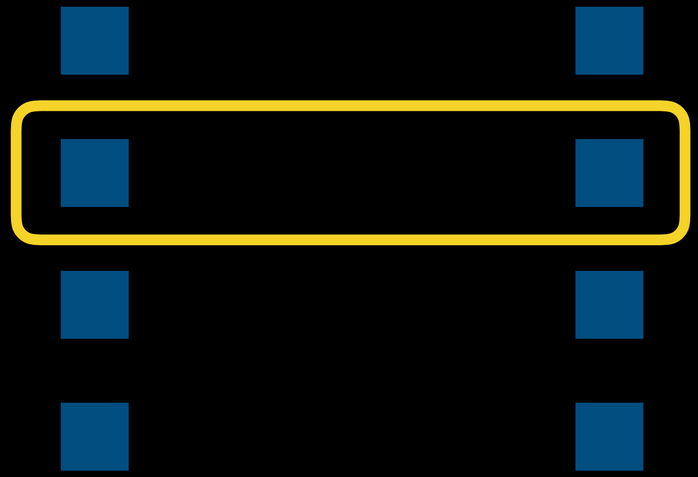




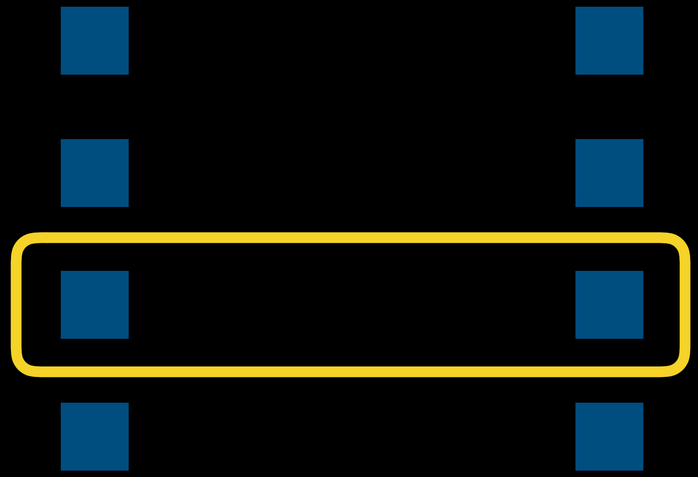
Iterate, comparing a[i] to b[i]



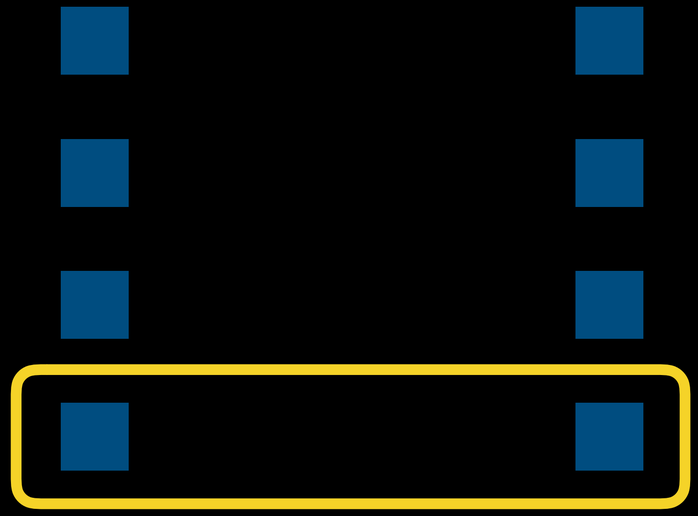
Iterate, comparing a[i] to b[i]



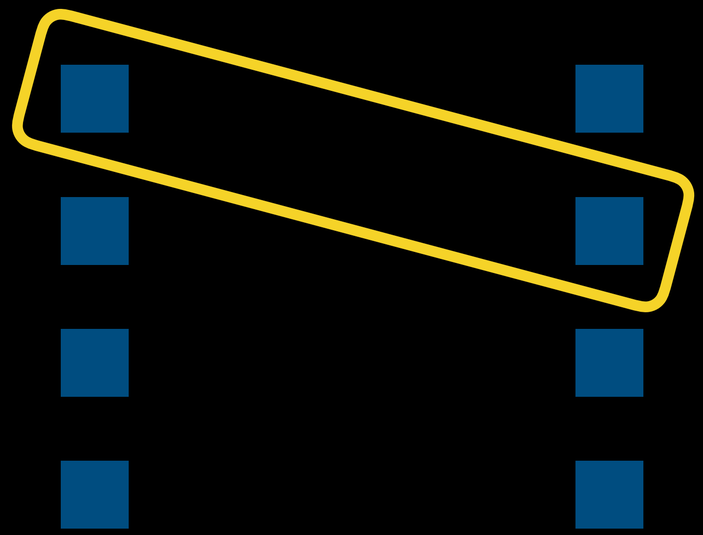
Iterate, comparing a[i] to b[i]



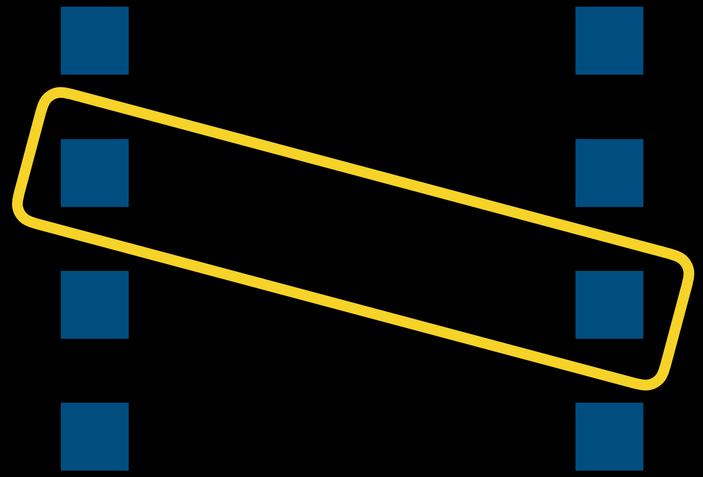
Iterate, comparing a[i] to b[i]

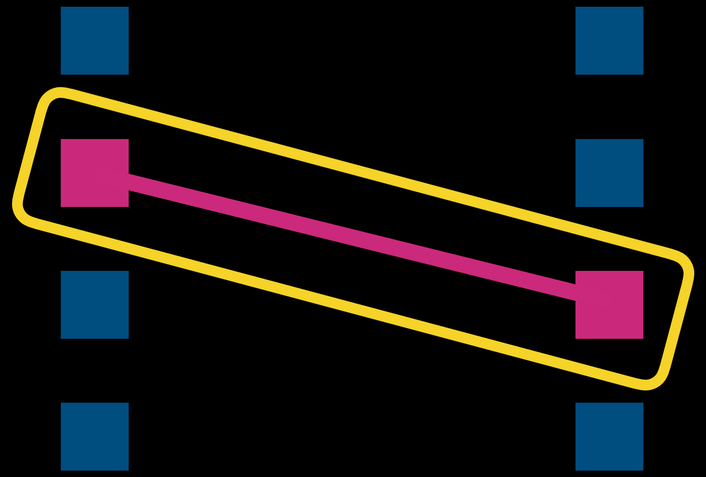


loop, comparing a[i] to b[(i+1)%len]

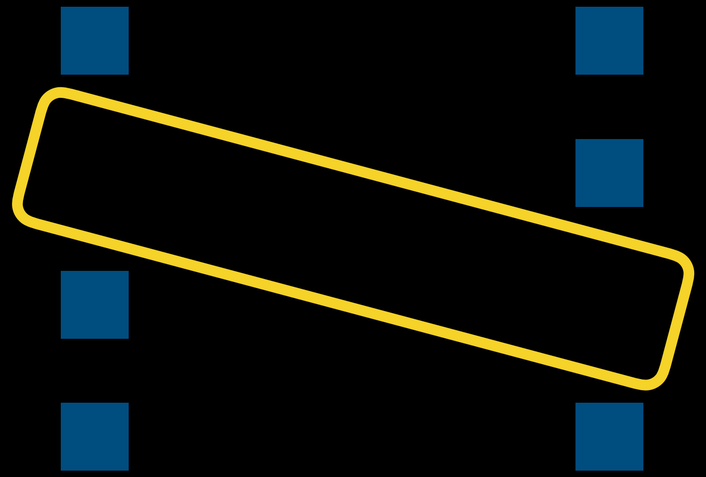


loop **fixed** number of times,
comparing a[i] to b[(i+1)%len]

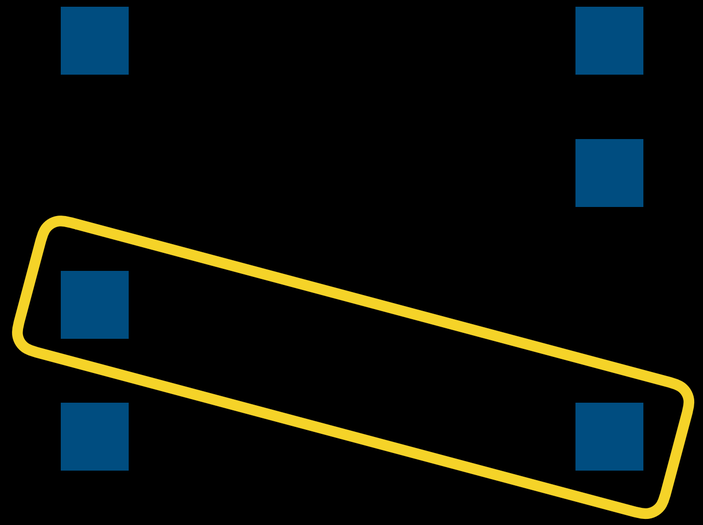




Remove found match



Continue



SplitMesher: approximates **Matching** without materializing meshing graph

SplitMesher: approximates **Matching** without materializing meshing graph

$O(n/q)$ time

(*q* is the global probability of spans meshing)

SplitMesher: approximates **Matching**
without materializing meshing graph

$O(n/q)$ time

(q is the global probability of spans meshing)

1/2* approximation w.h.p.

MESH *implementation details*

MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

fast framework for building custom allocators!

MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

< 5K SLOC, C++

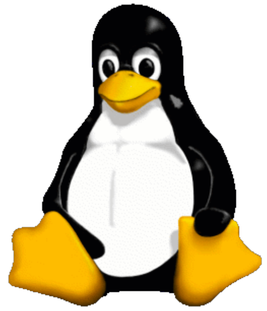
MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

< 5K SLOC, C++



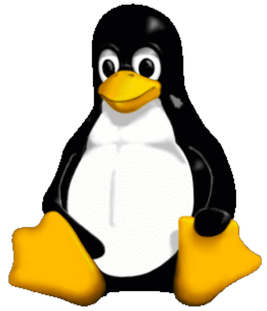
MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

< 5K SLOC, C++



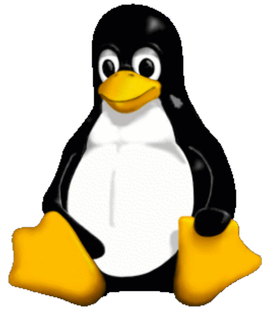
MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

< 5K SLOC, C++



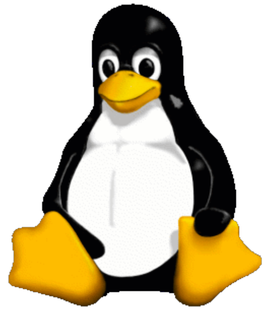
MESH *implementation details*

built using **Heap Layers**

github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

< 5K SLOC, C++



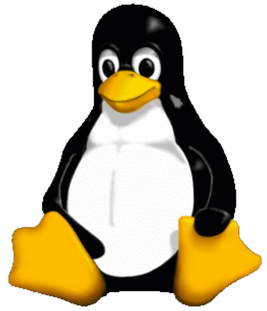
MESH *implementation details*

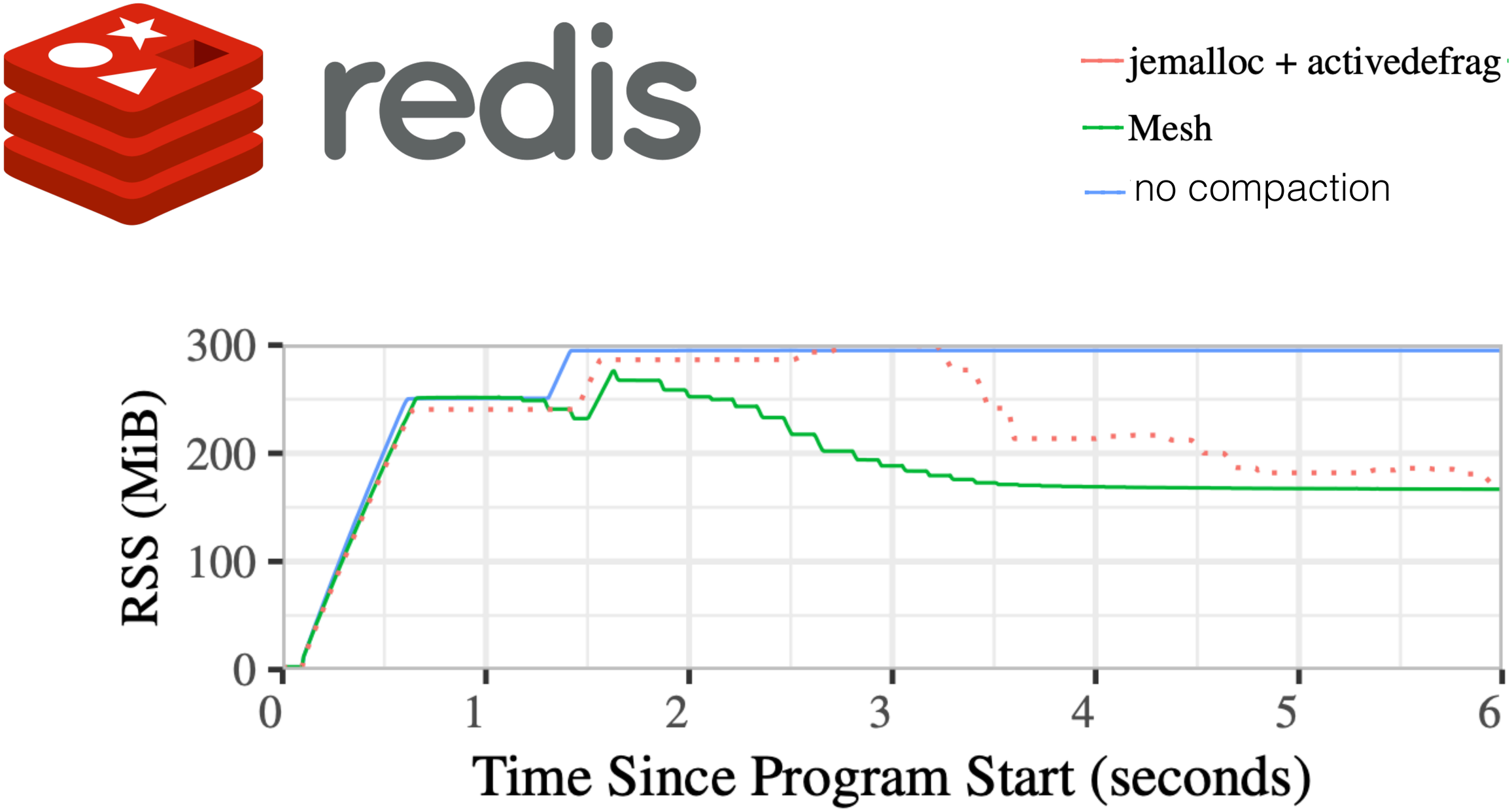
built using **Heap Layers**

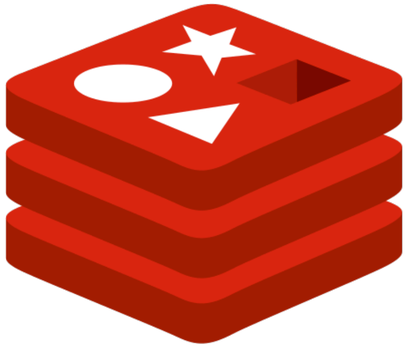
github.com/emeryberger/Heap-Layers

(used to build Hoard, DieHard, DieHarder…)

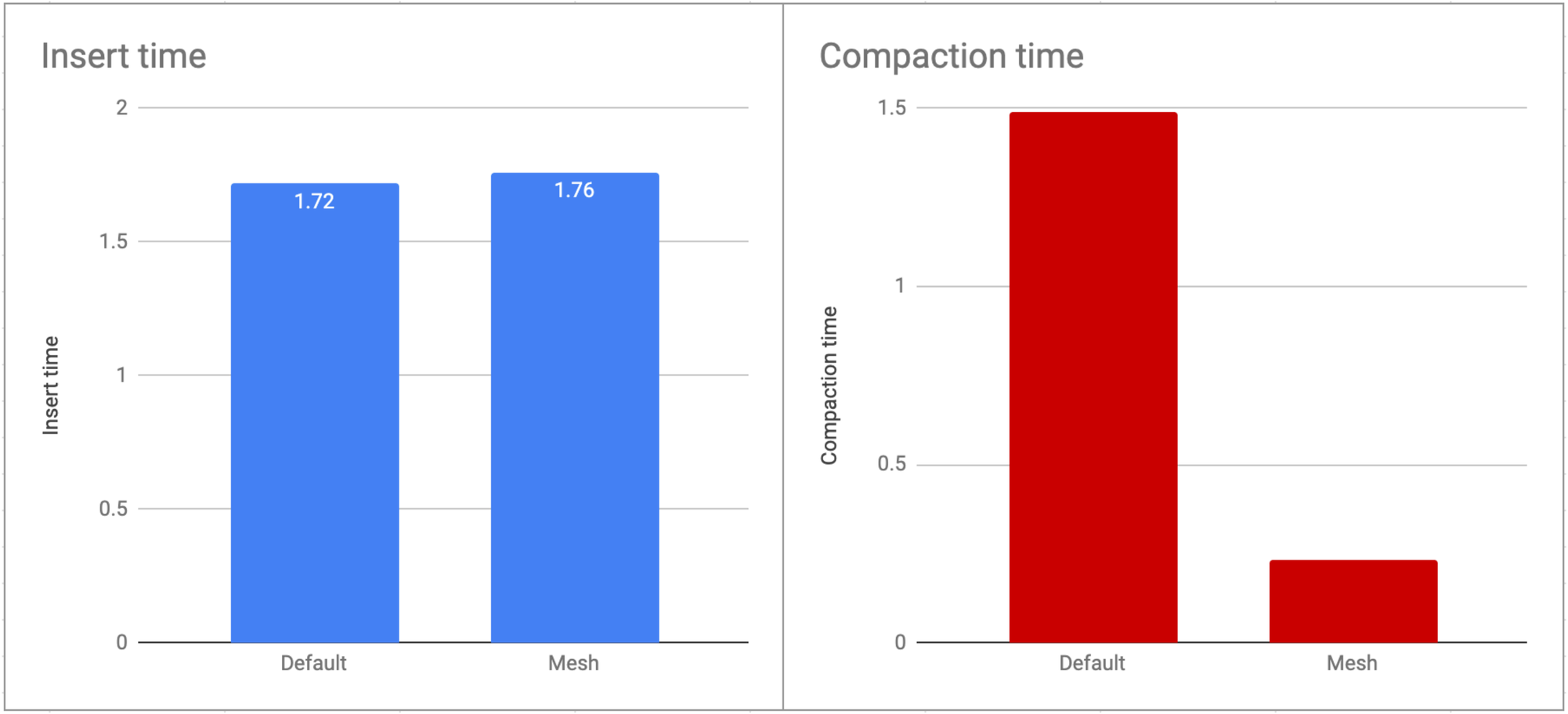
< 5K SLOC, C++





redis
jemalloc + activedefrag
Mesh
no compaction
RSS (MiB)
300
200
100
0
0
1
2
3
4
5
6
Time Since Program Start (seconds)

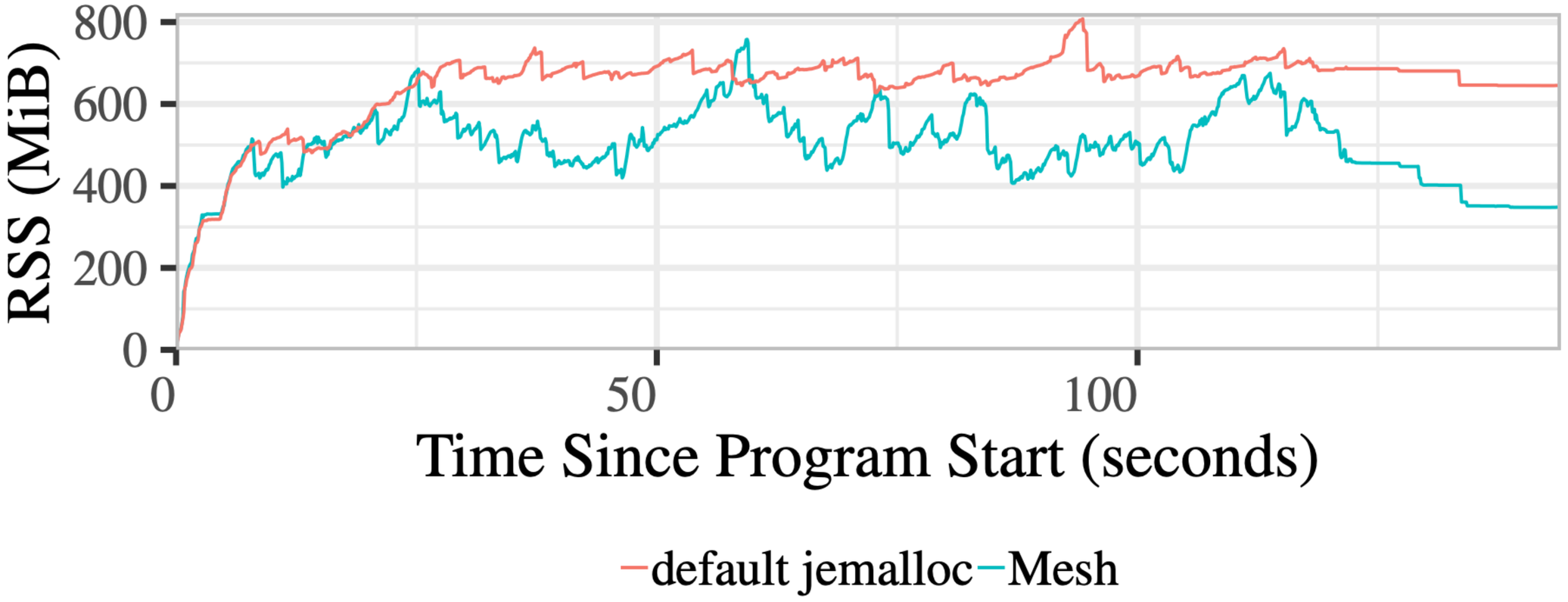


redis + MESH

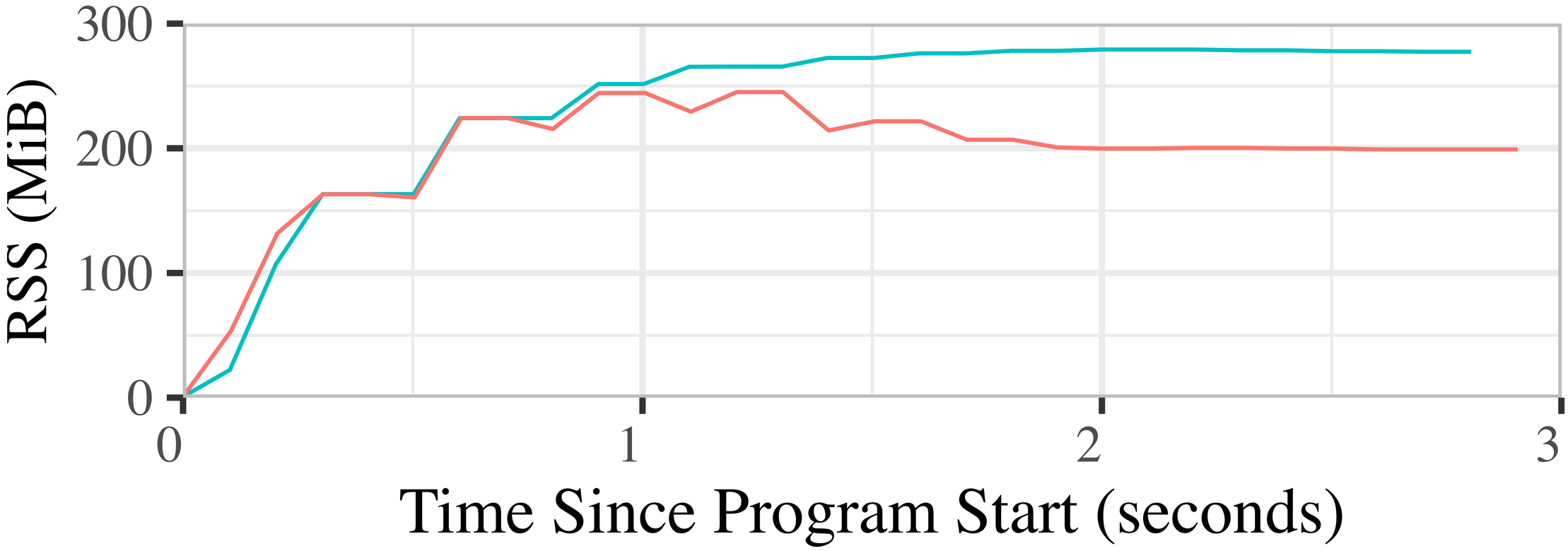

Insert time
2
1.5
1
0.5
0
Insert time
1.72
1.76
Default
Mesh
Compaction time
1.5
1
0.5
0
Compaction time
Default
Mesh

17% heap size reduction

< 1% performance overhead



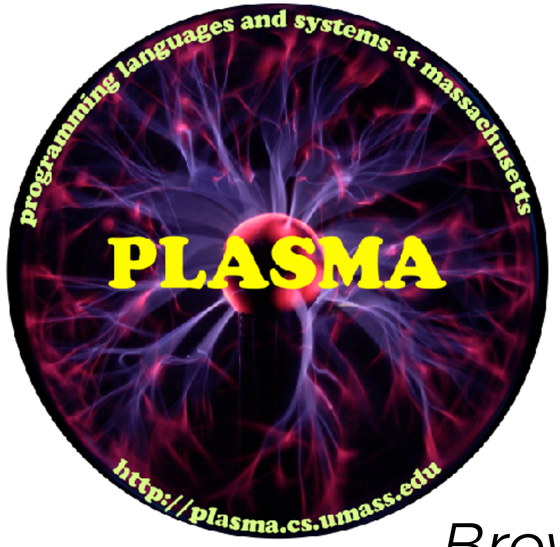
Ruby Compaction for Free



http:// **LIBMESH** .org

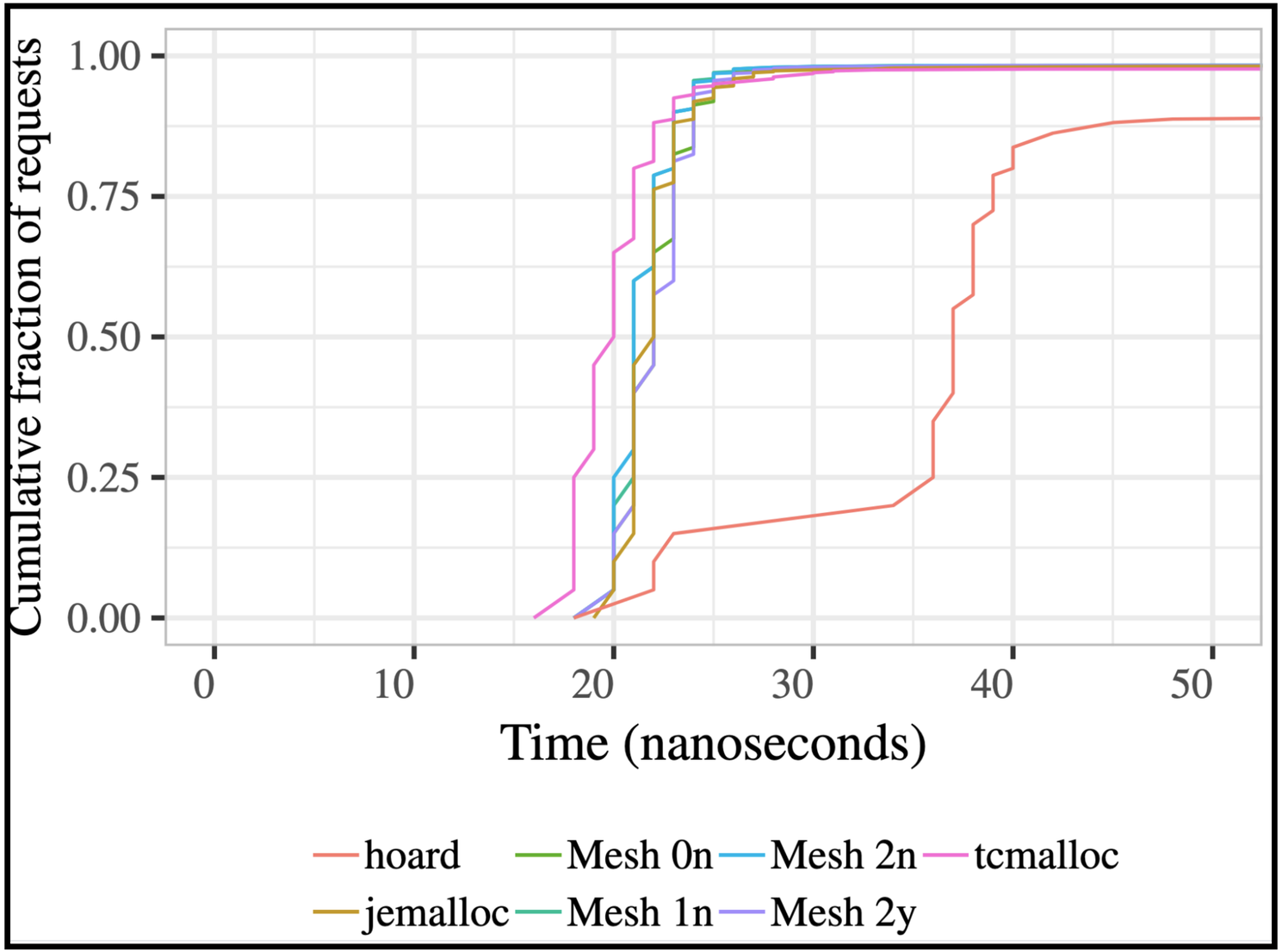
¡Compactación sin Relocación!

(compaction without relocation)



Browsix, Coz, DieHard, Hoard...

Latency: `malloc`



Latency: free