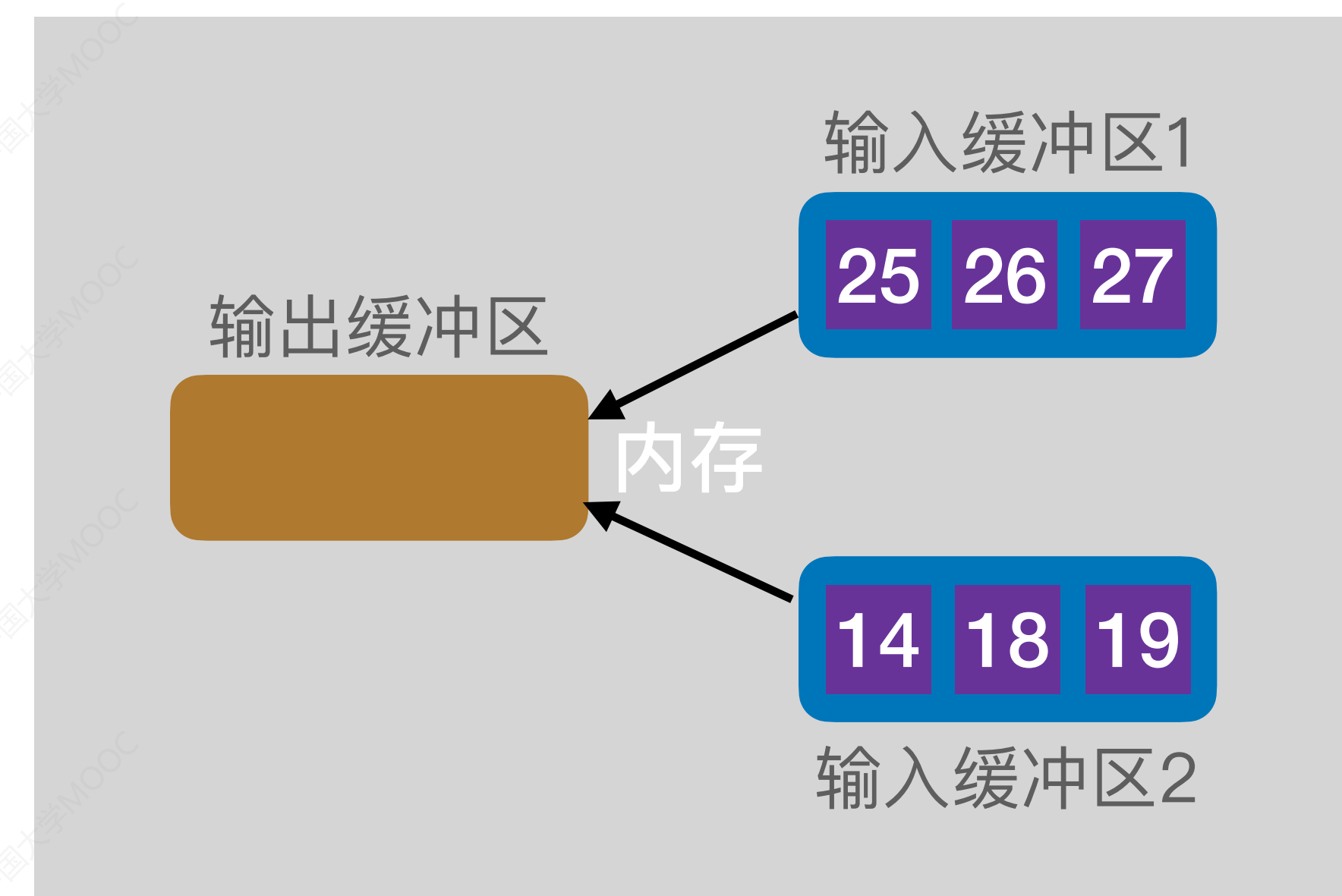
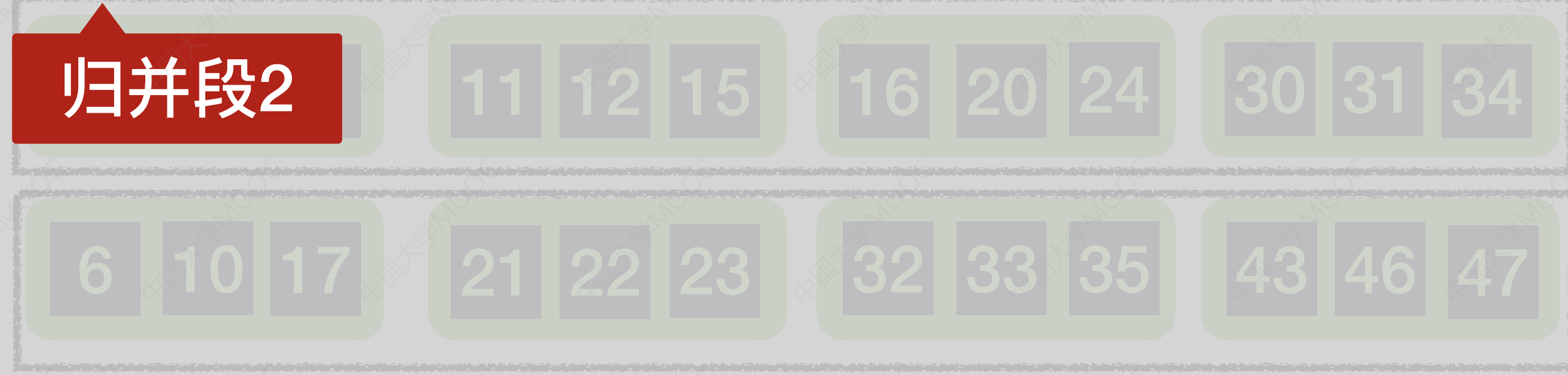
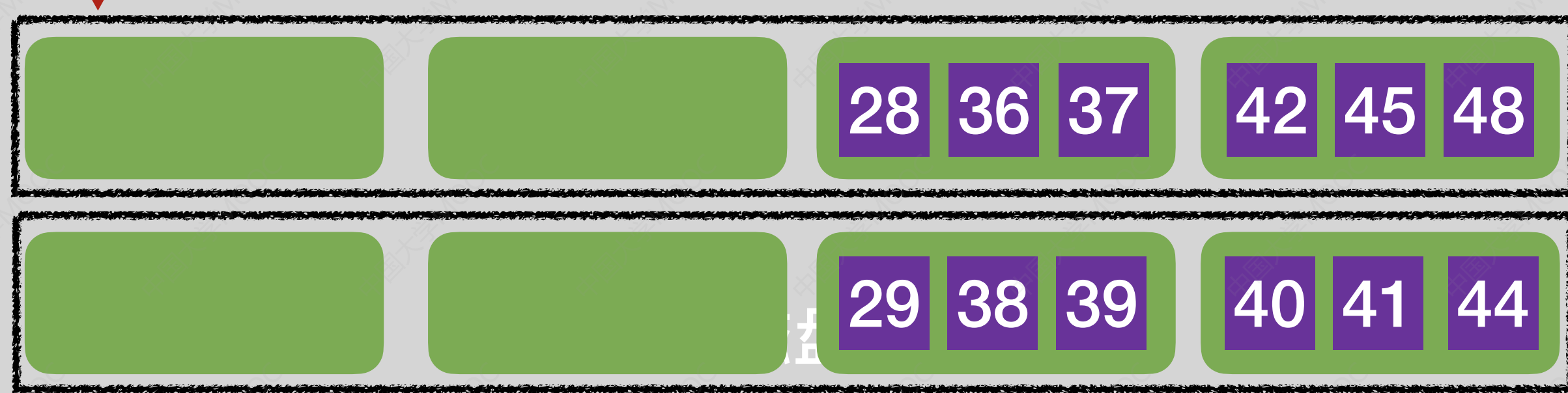
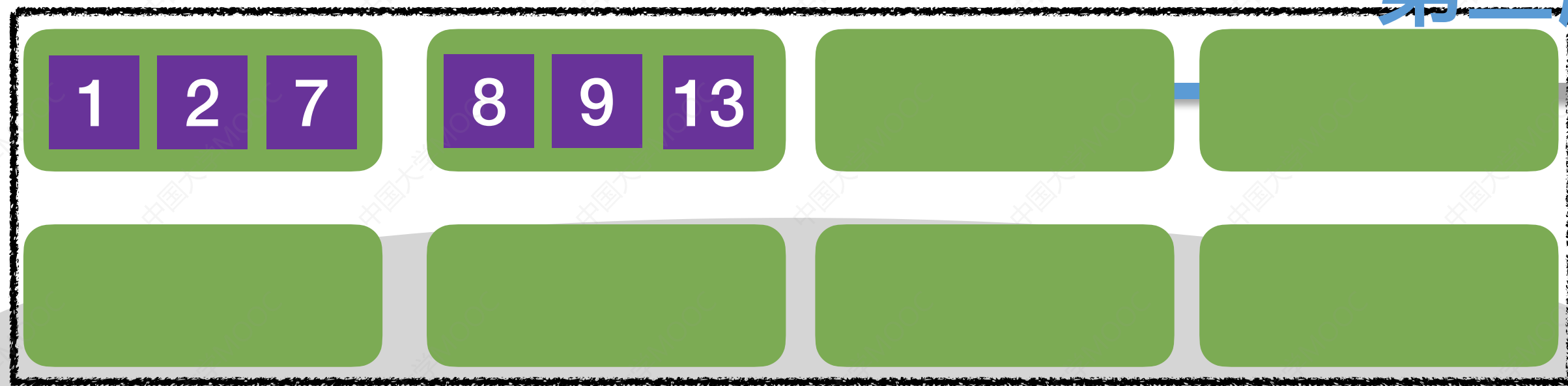
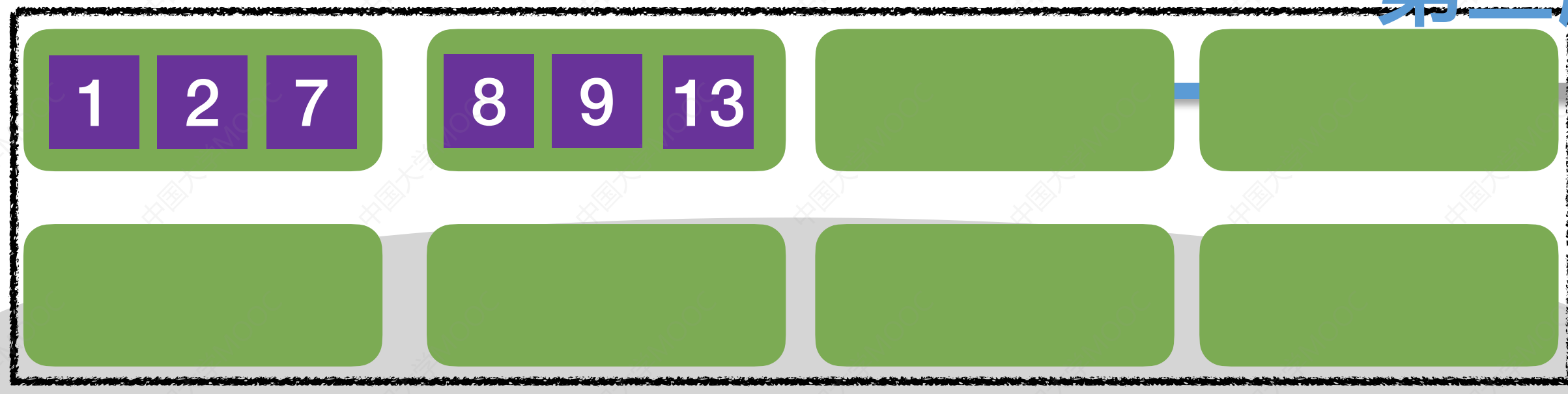


第二趟归并

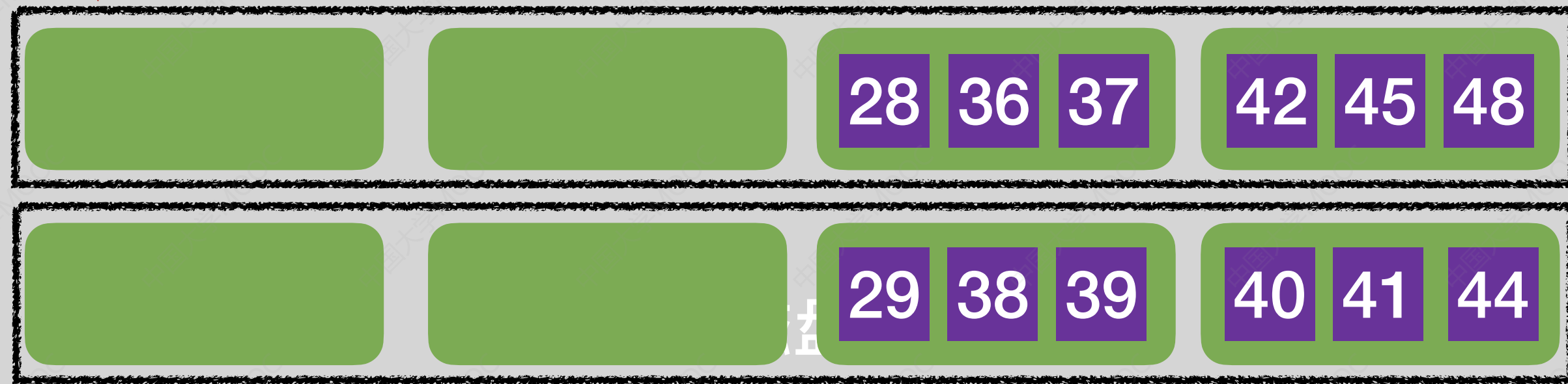


将两个有序归并段归并为一个

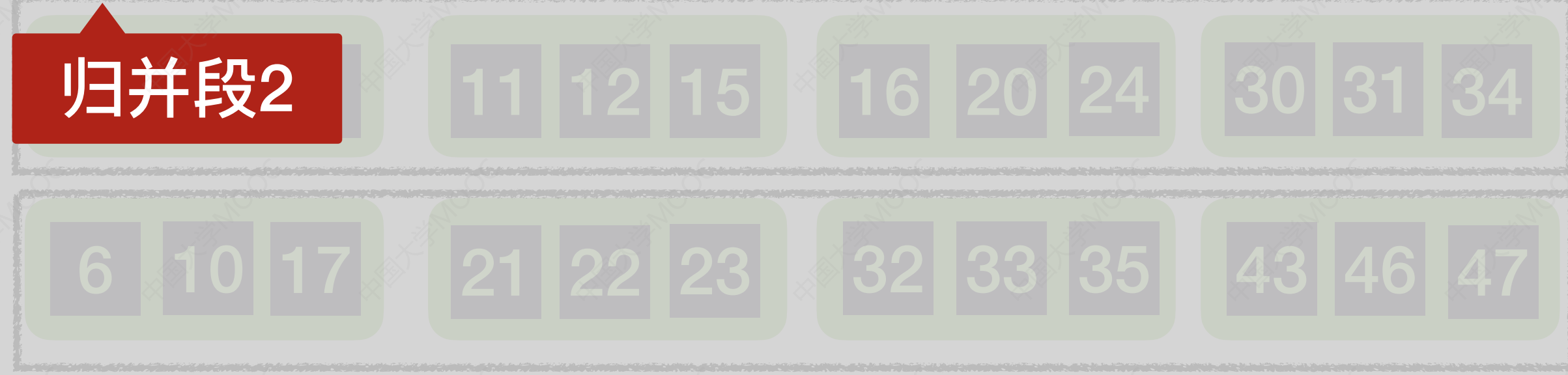
第二趟归并



归并段1

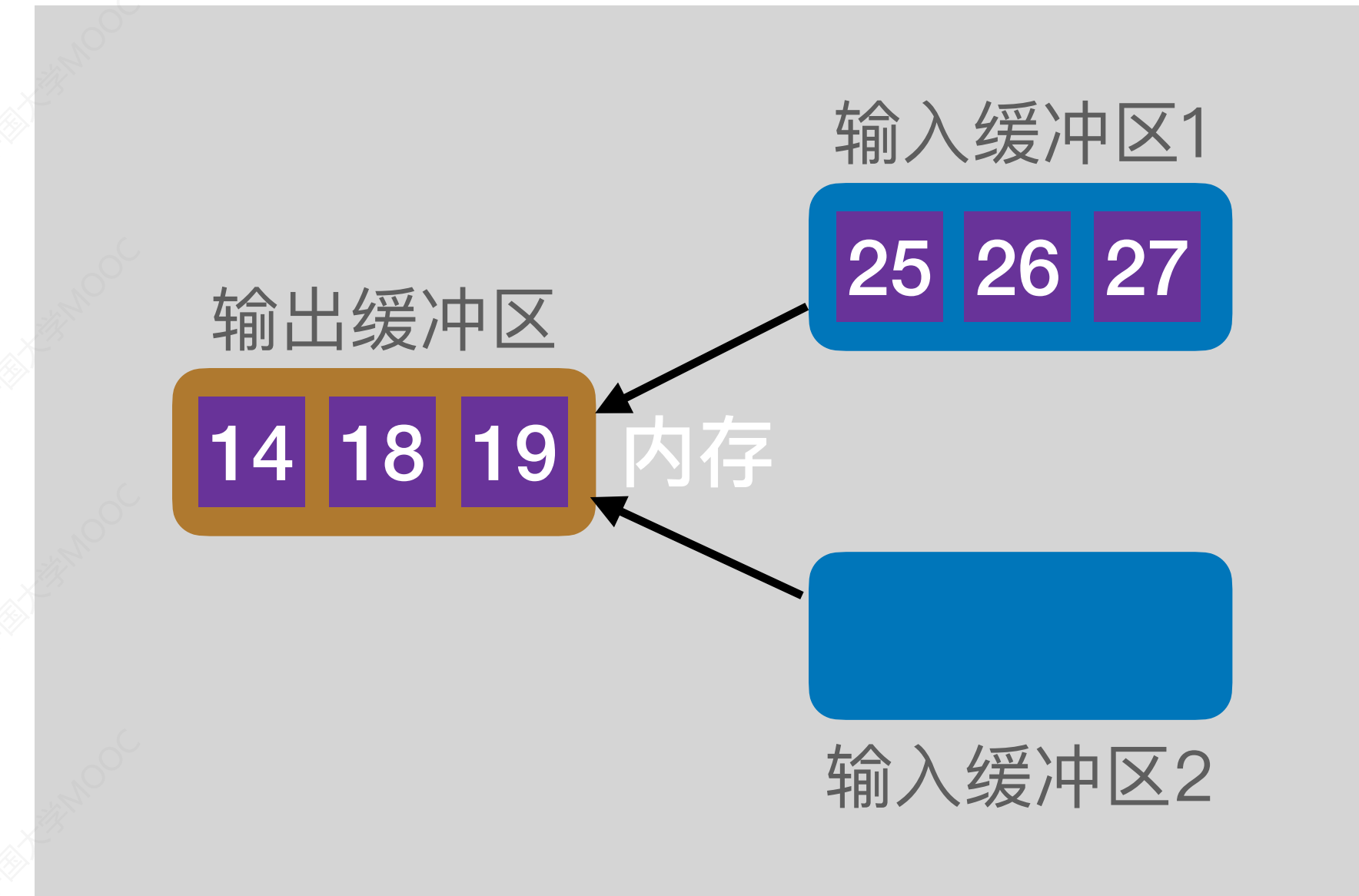


归并段2



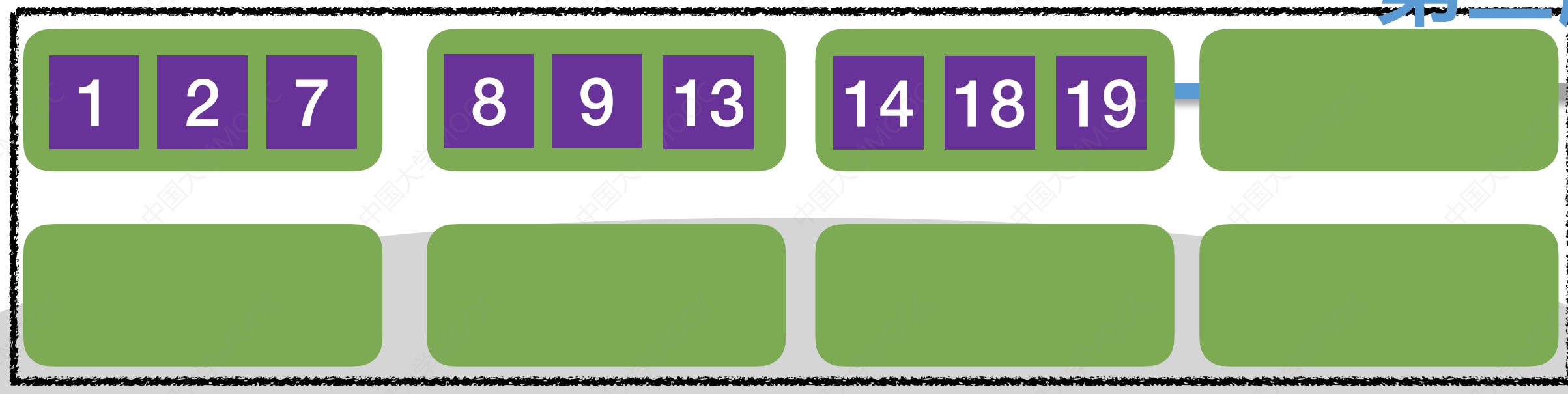
读磁盘

写磁盘

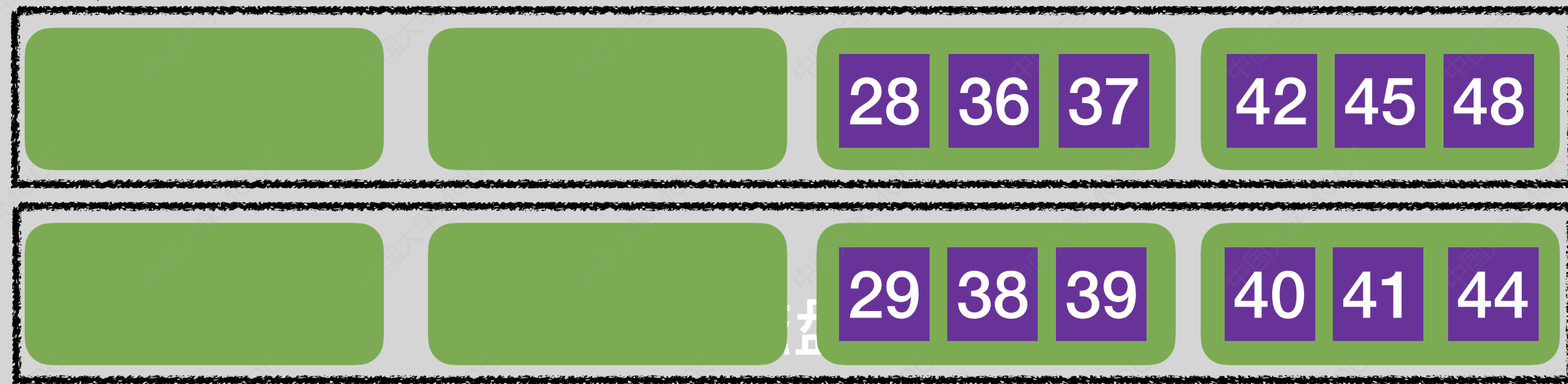


将两个有序归并段归并为一个

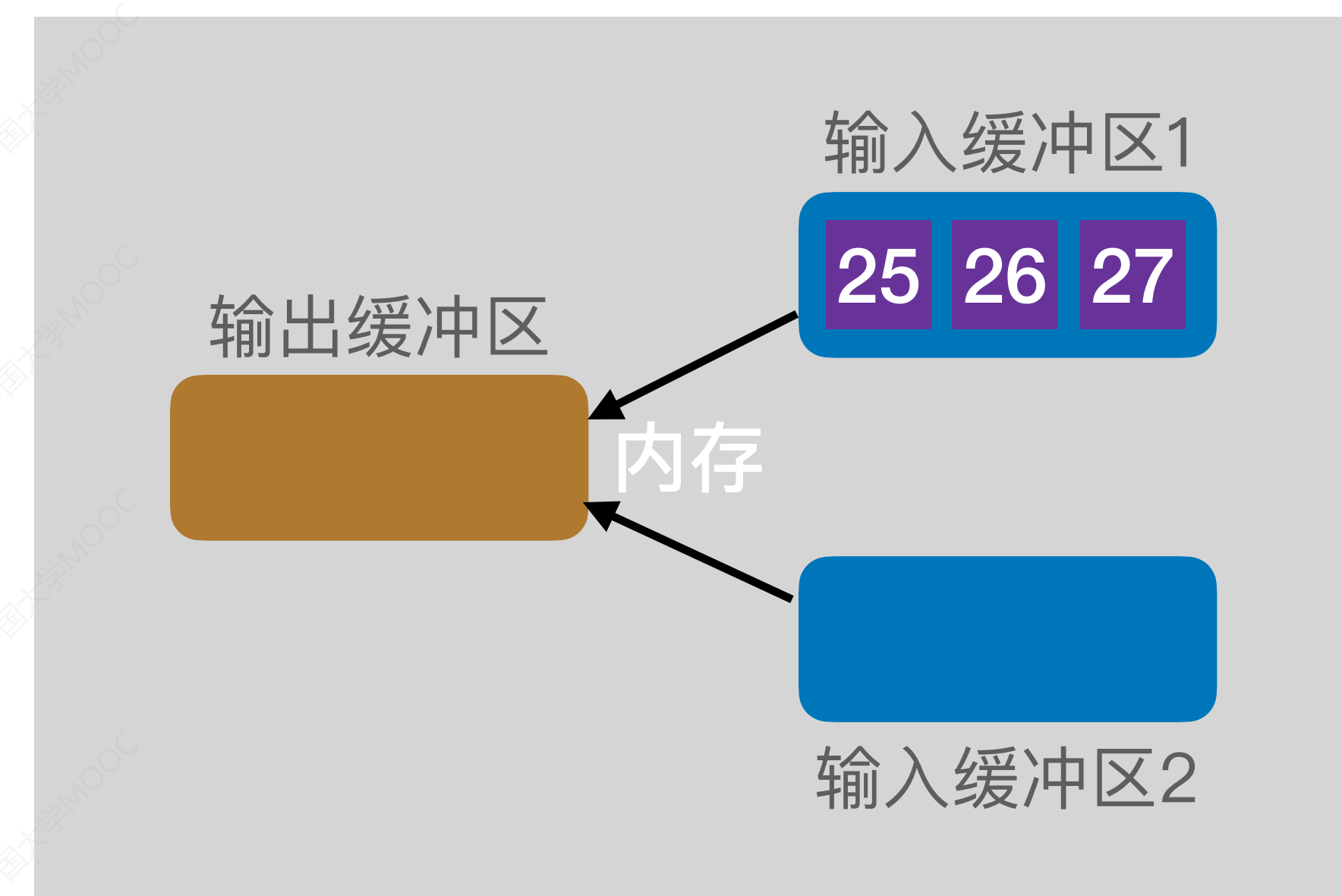
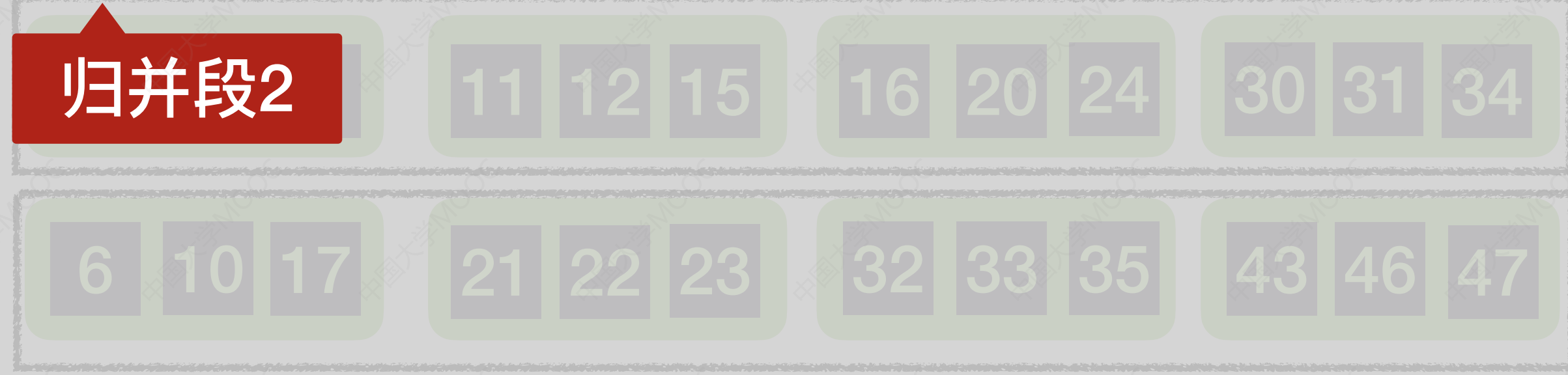
第二趟归并



归并段1

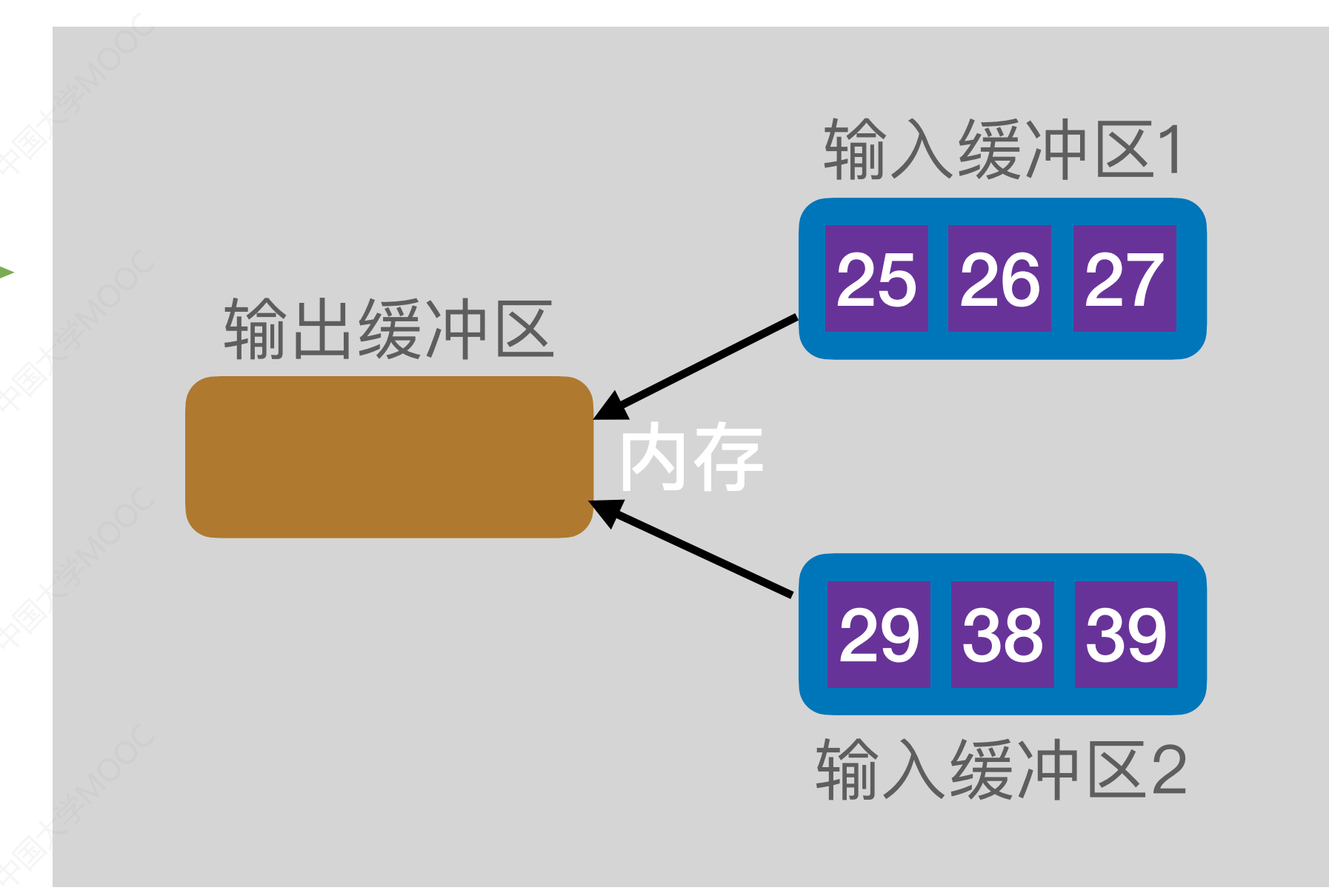
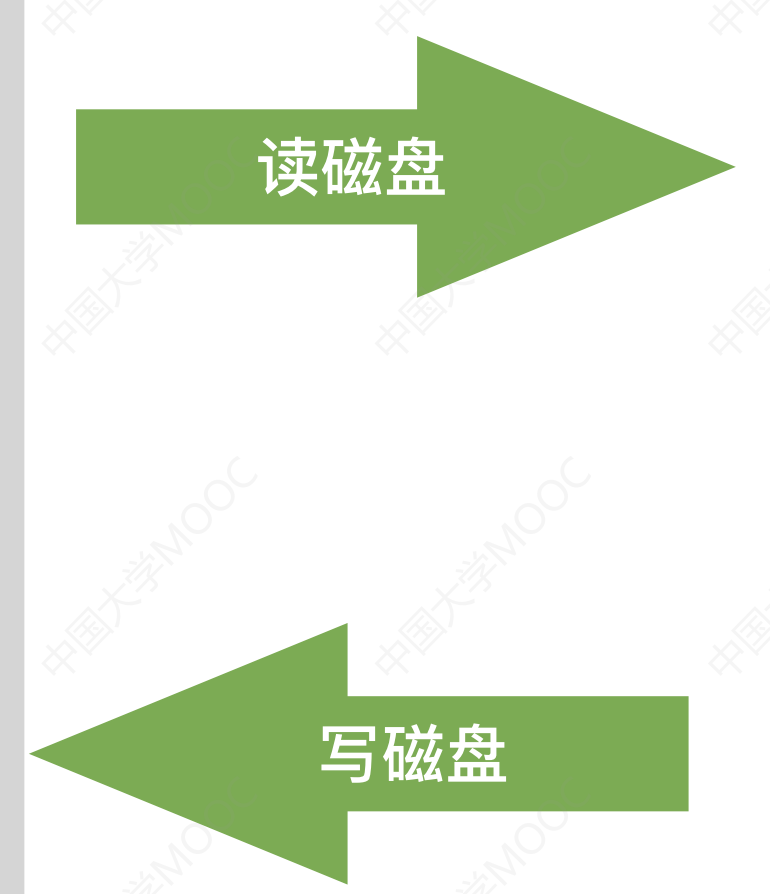
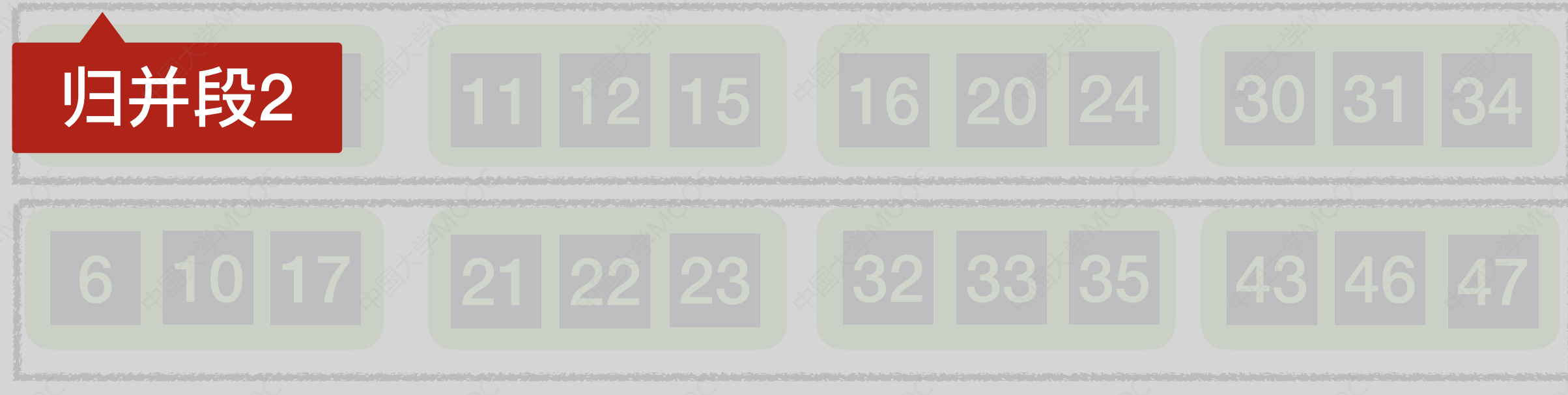
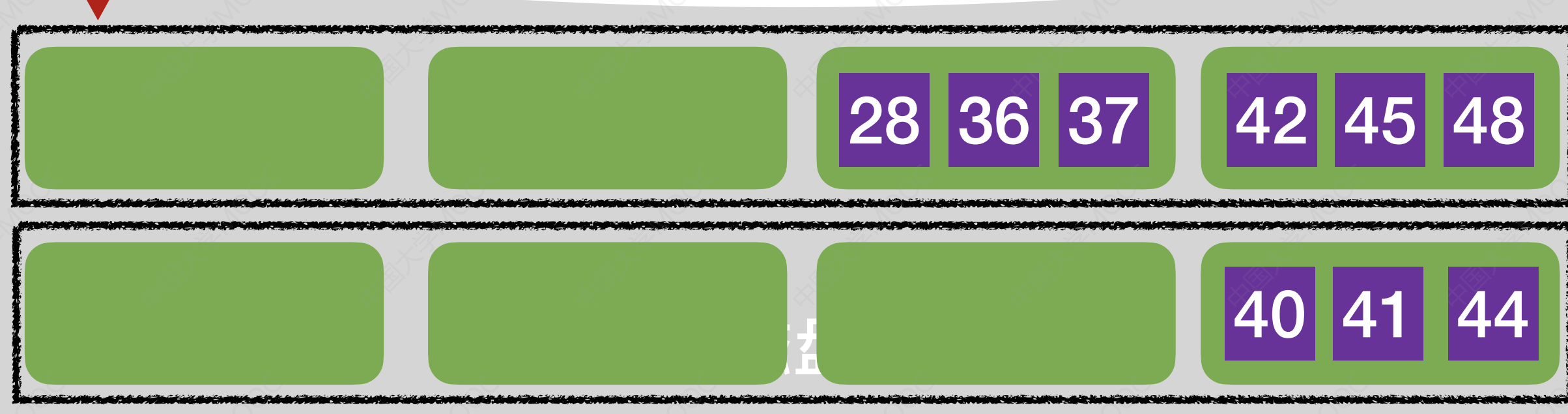
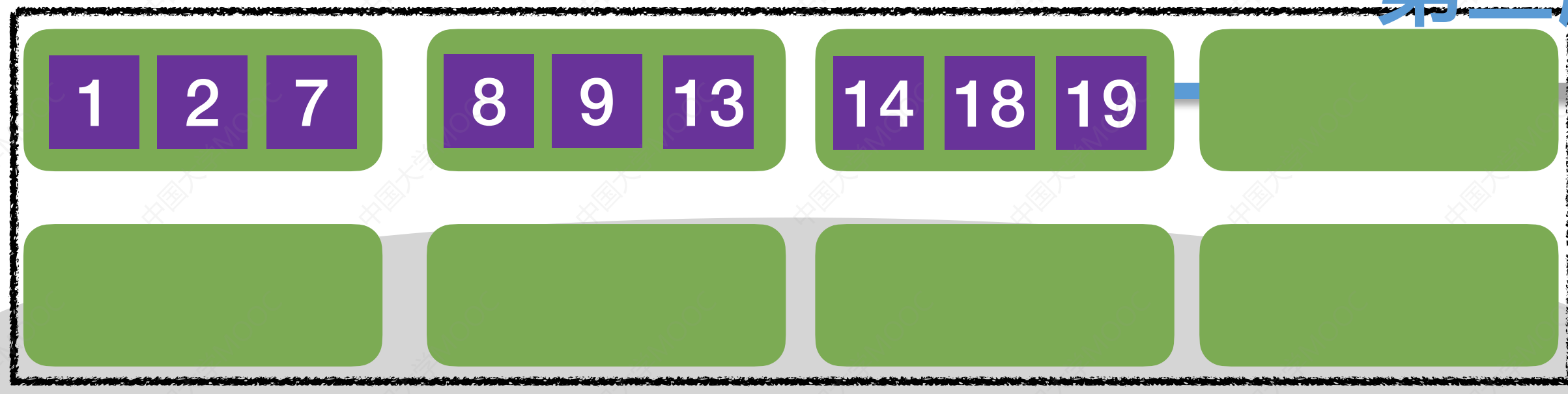


归并段2



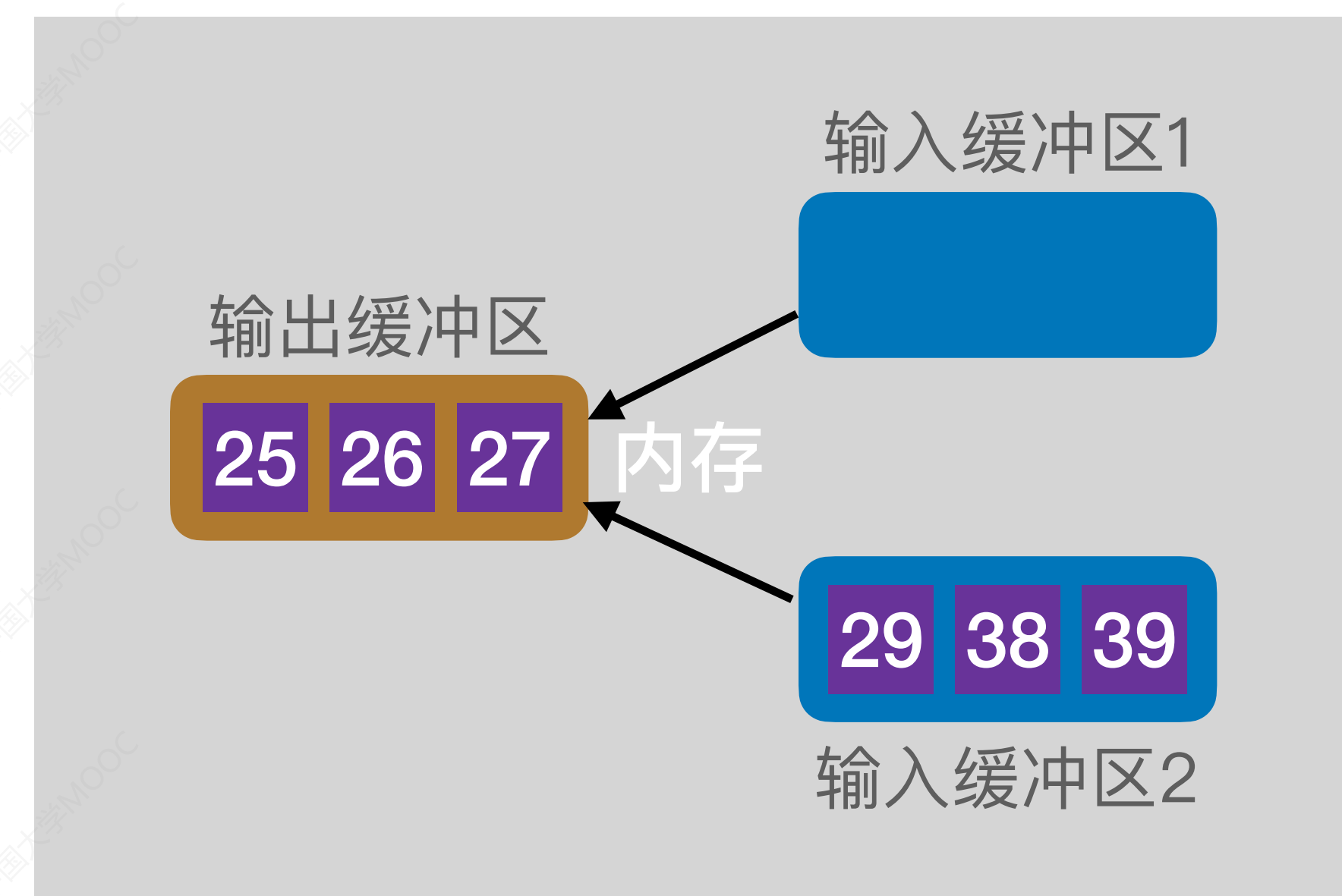
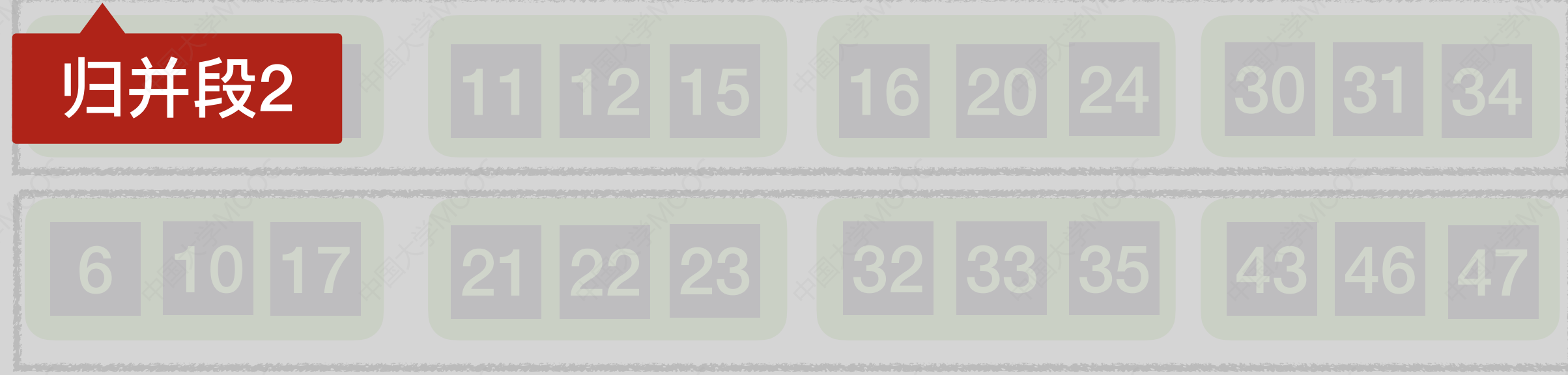
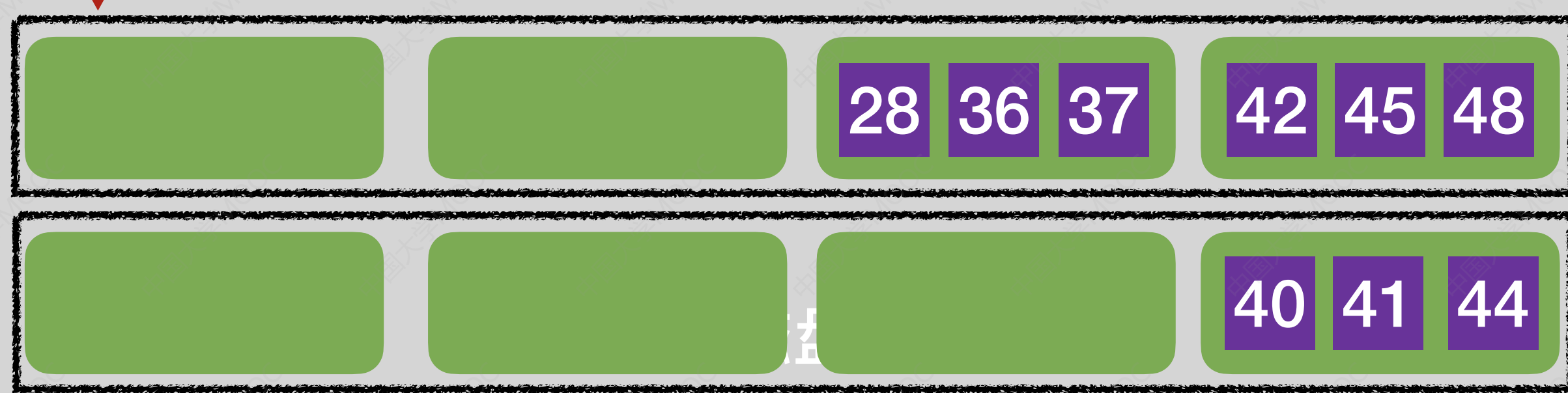
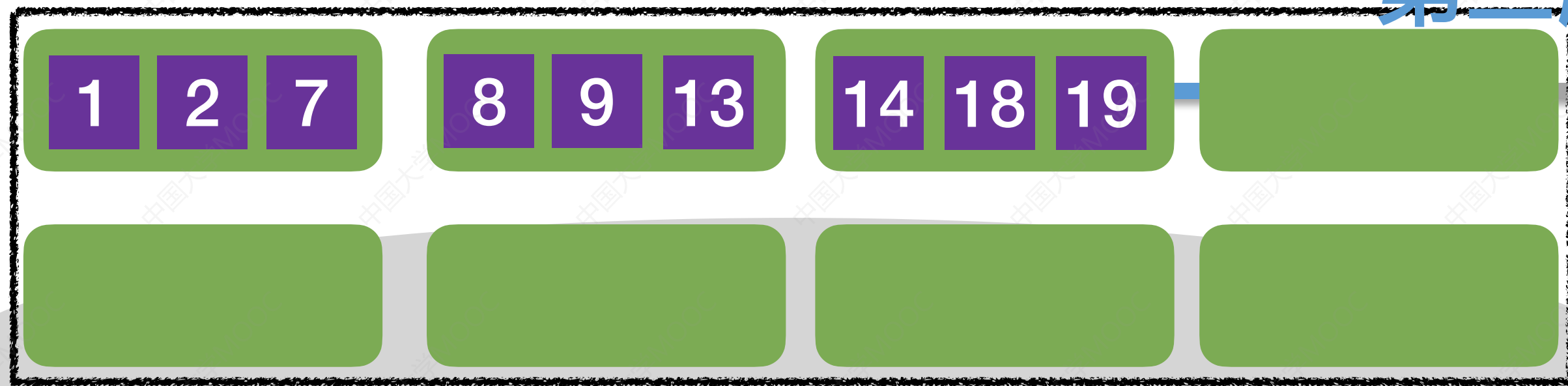
将两个有序归并段归并为一个

第二趟归并



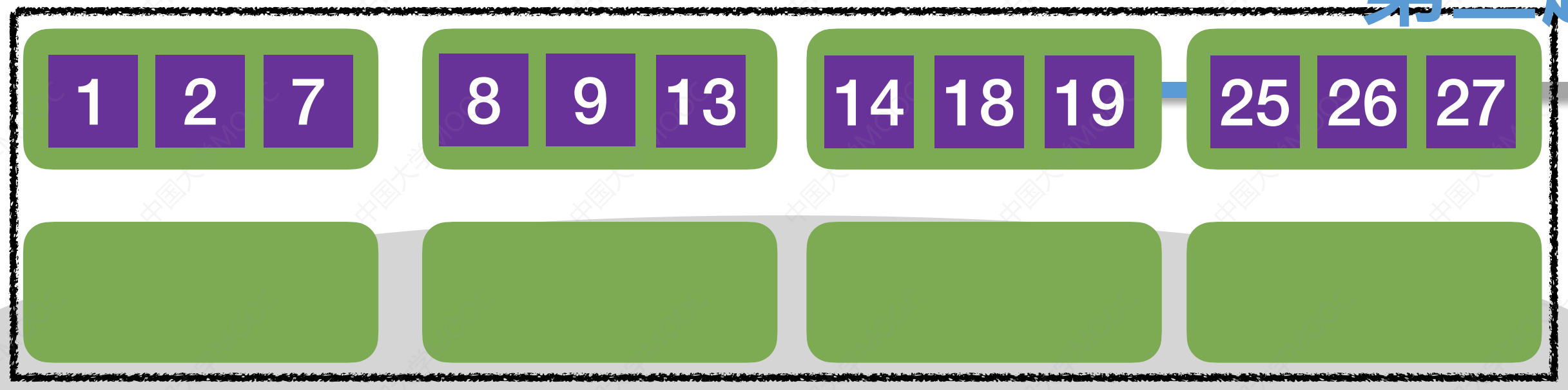
将两个有序归并段归并为一个

第二趟归并

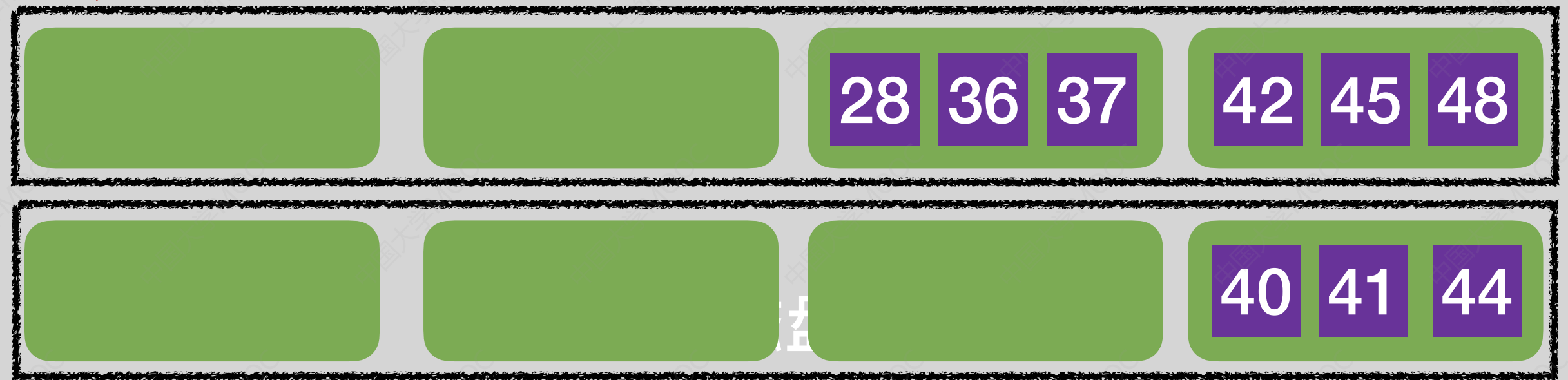


将两个有序归并段归并为一个

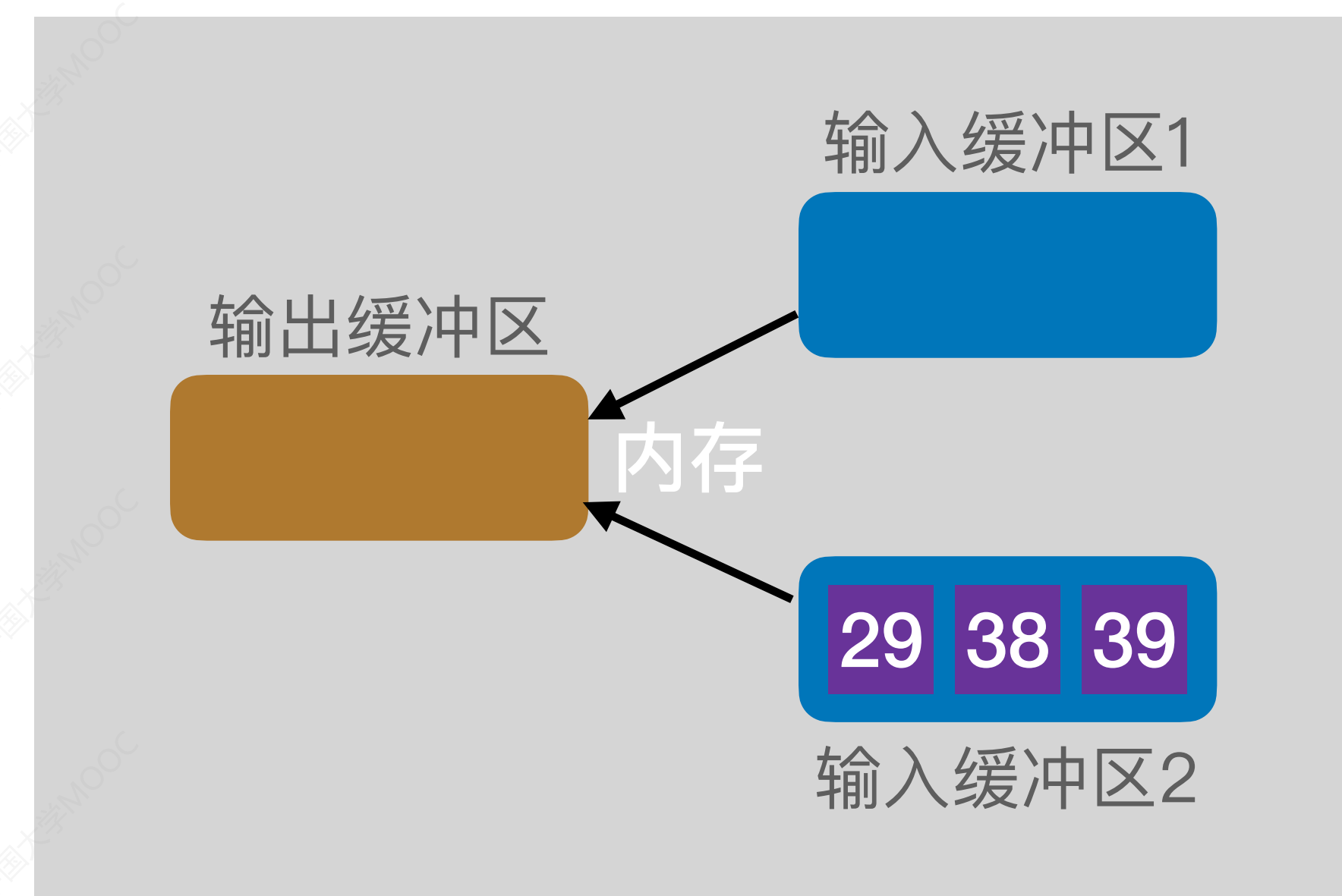
第二趟归并



归并段1

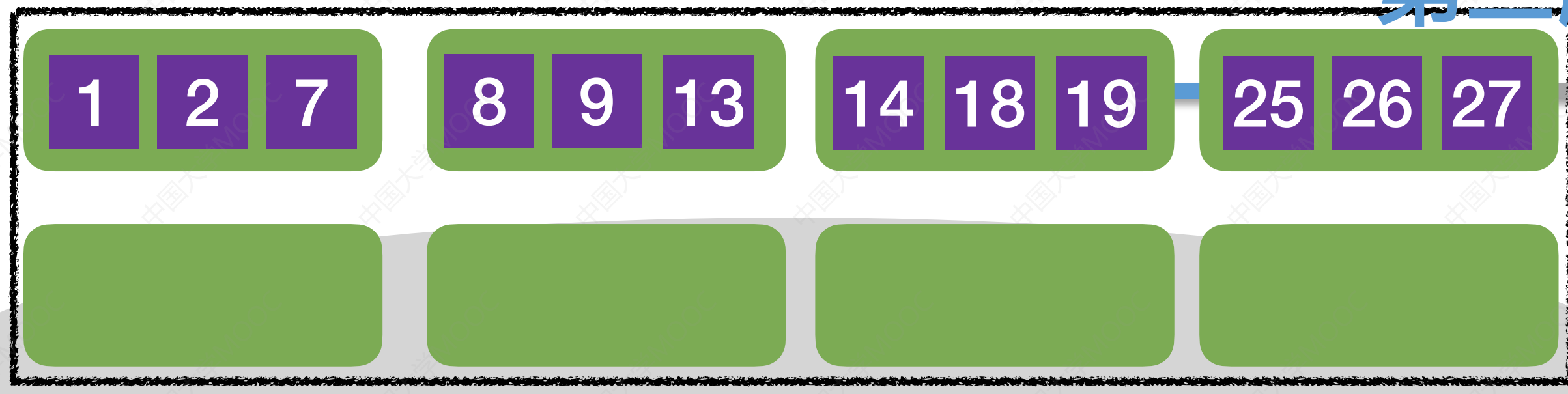


归并段2

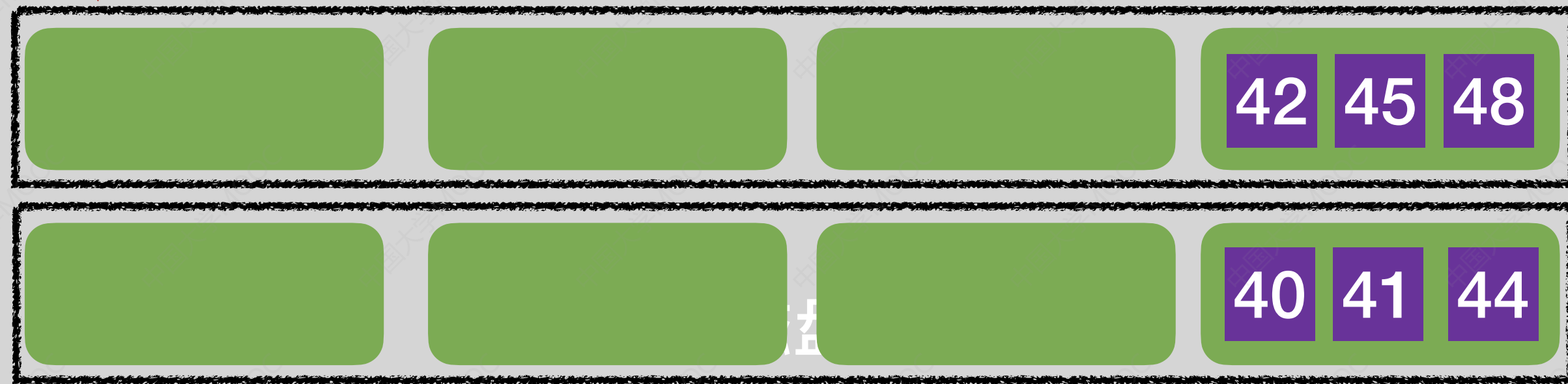


将两个有序归并段归并为一个

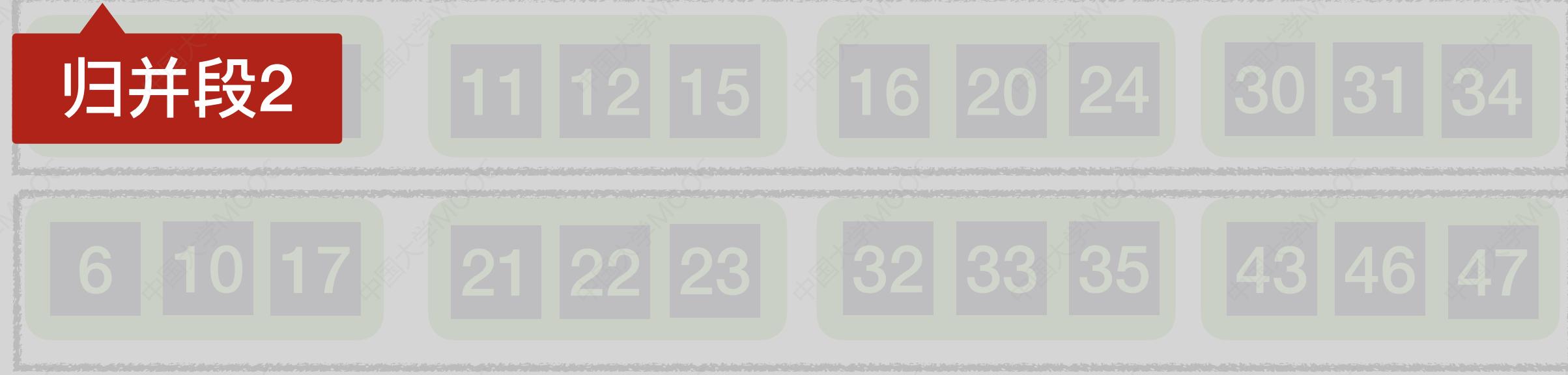
第二趟归并



归并段1

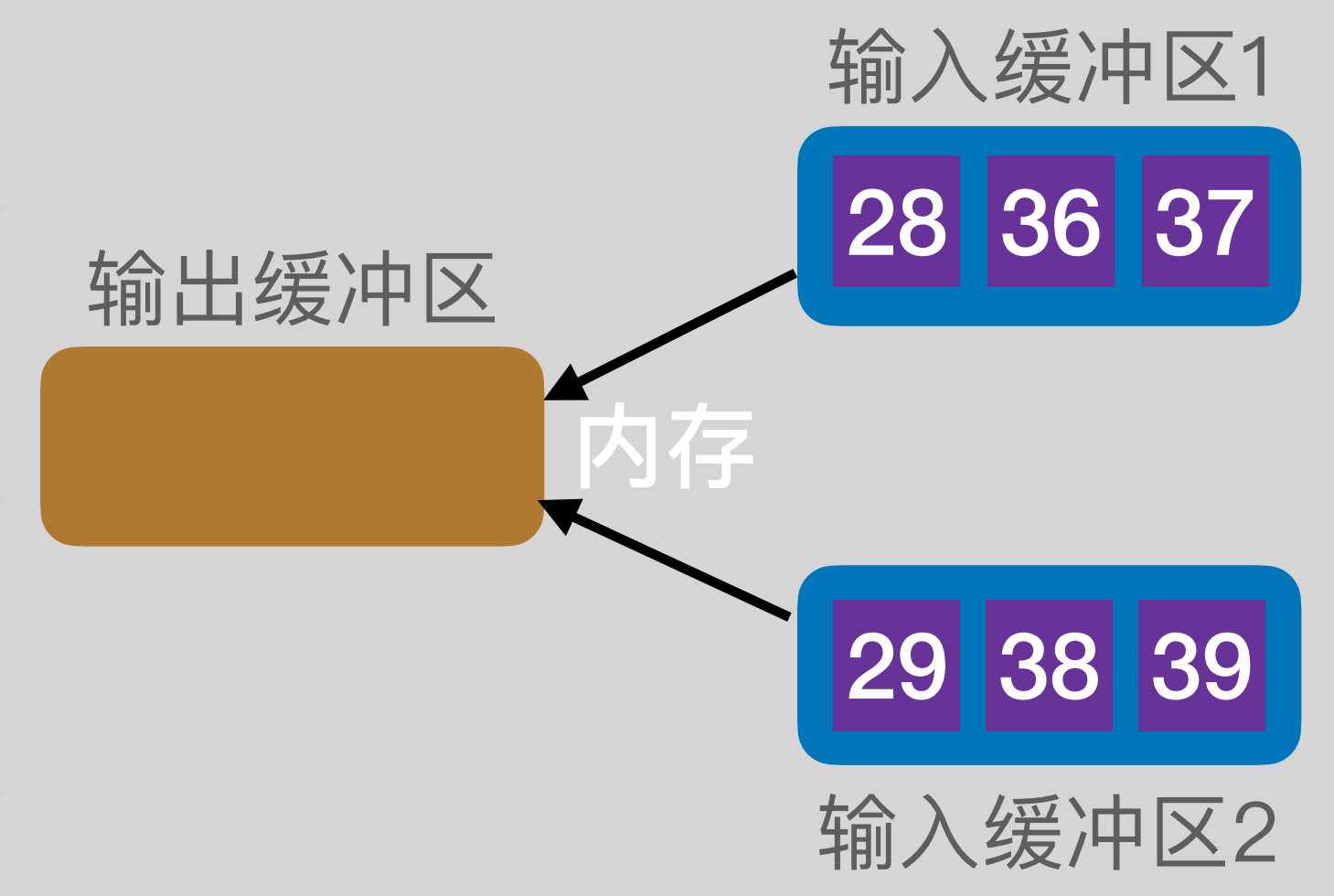


归并段2



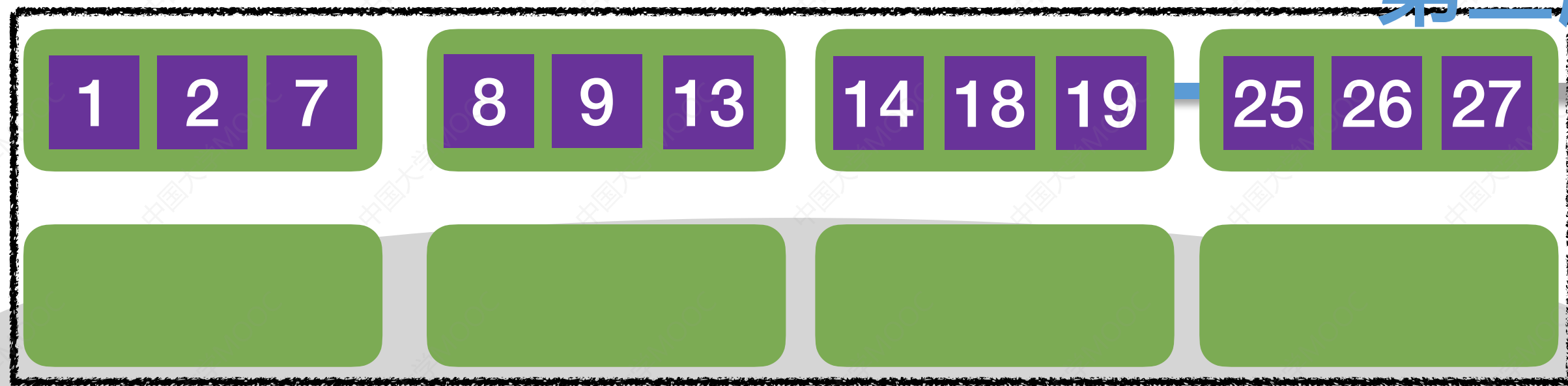
读磁盘

写磁盘

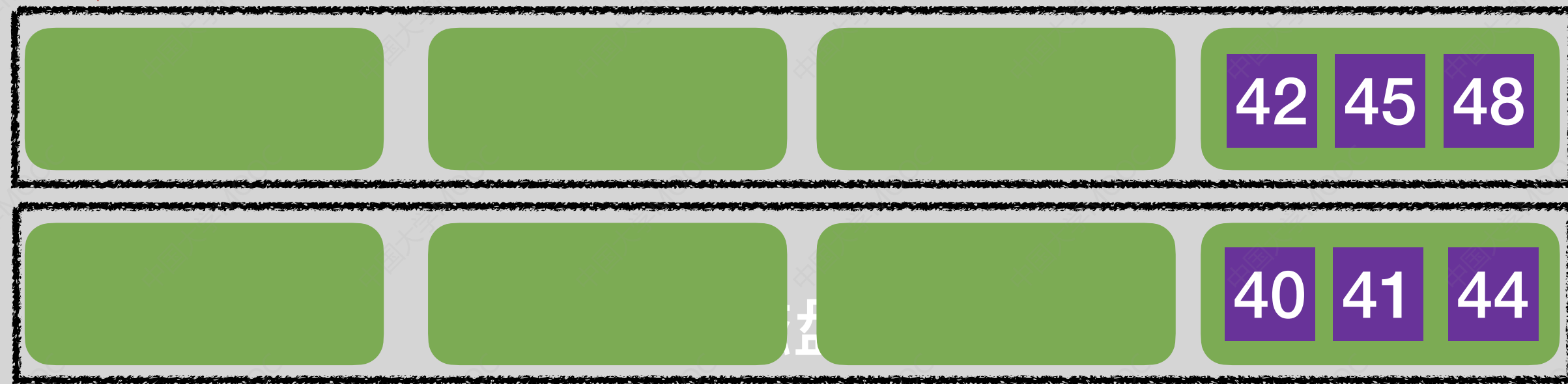


将两个有序归并段归并为一个

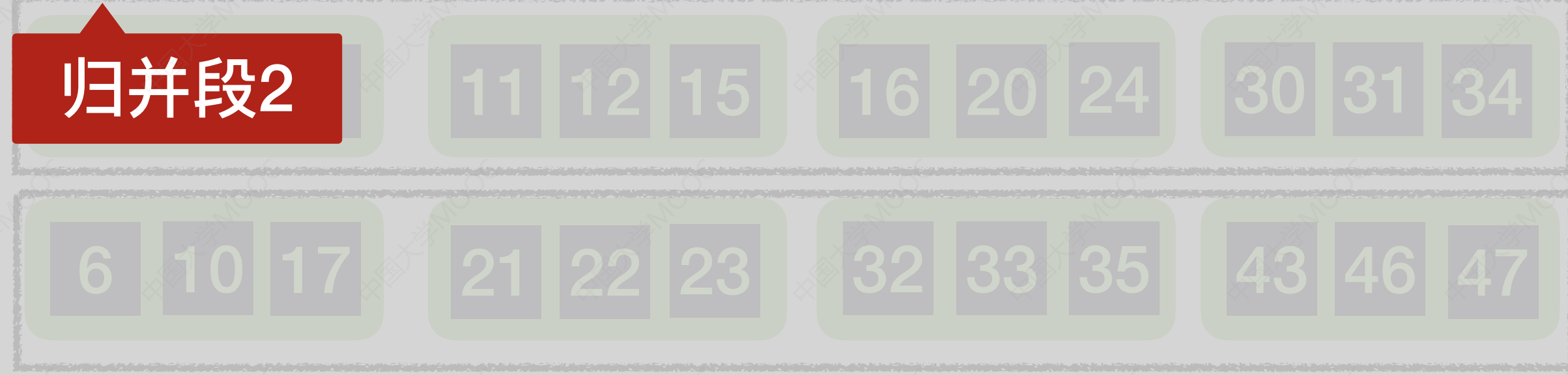
第二趟归并



归并段1

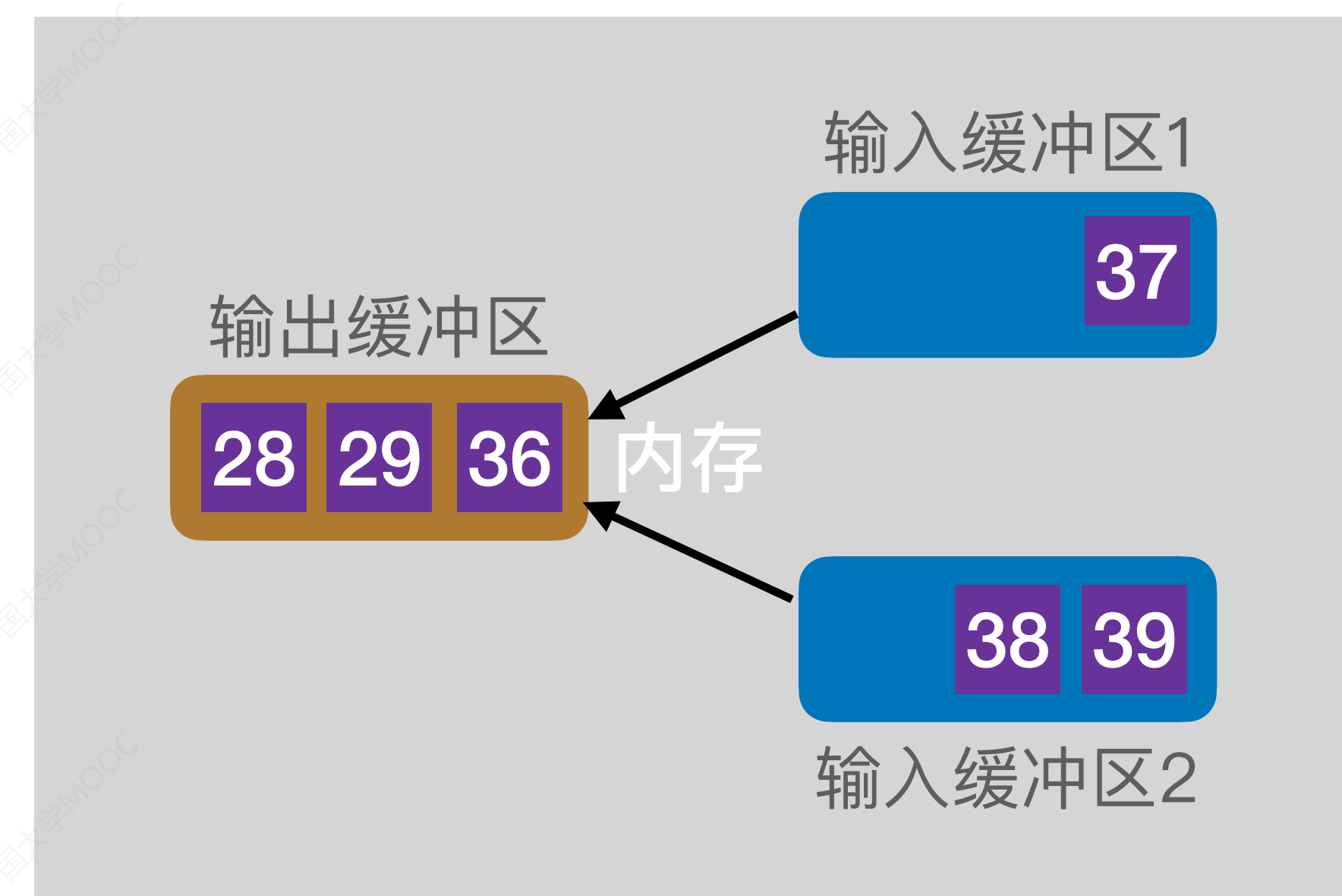


归并段2



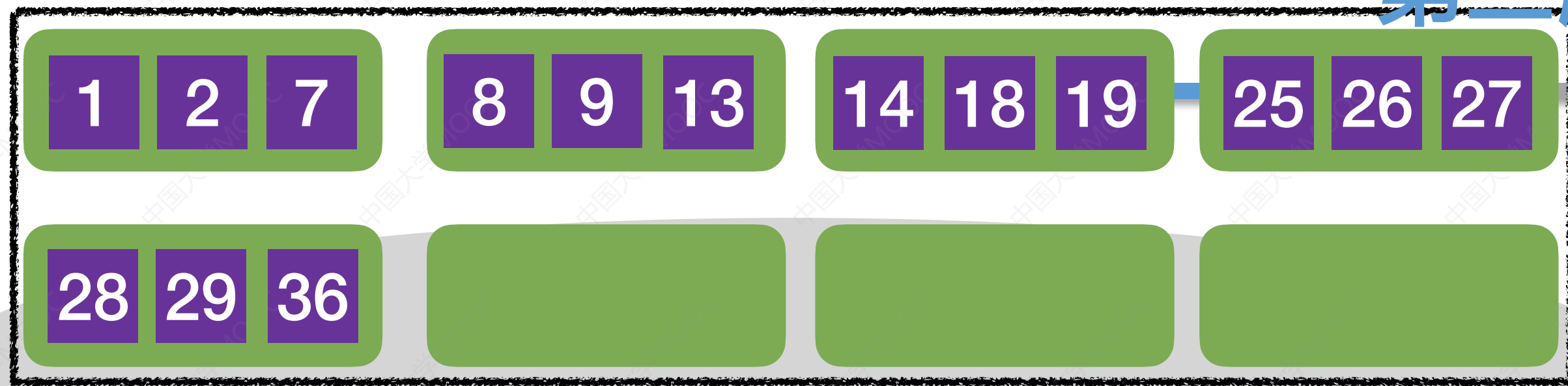
读磁盘

写磁盘

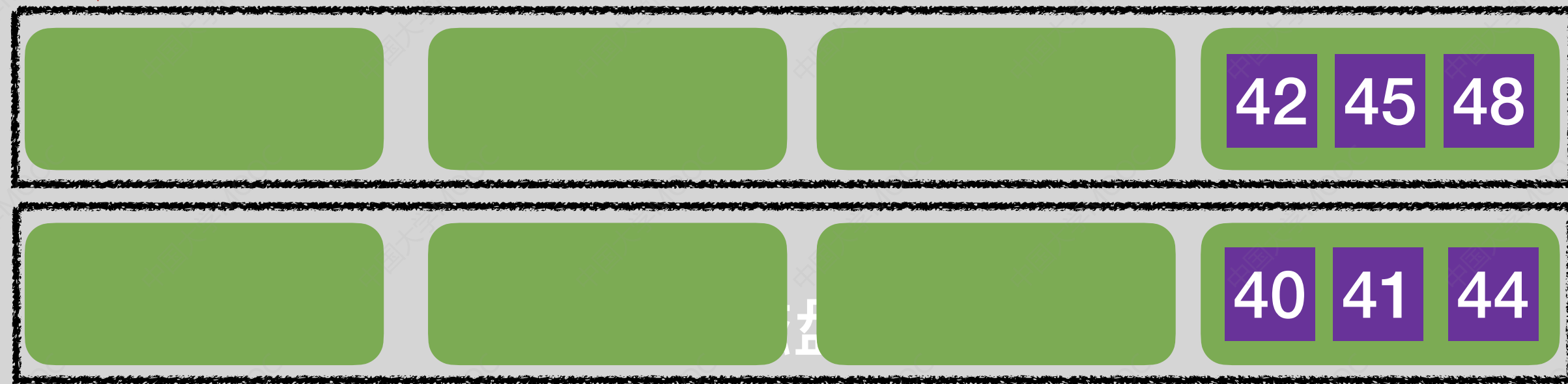


将两个有序归并段归并为一个

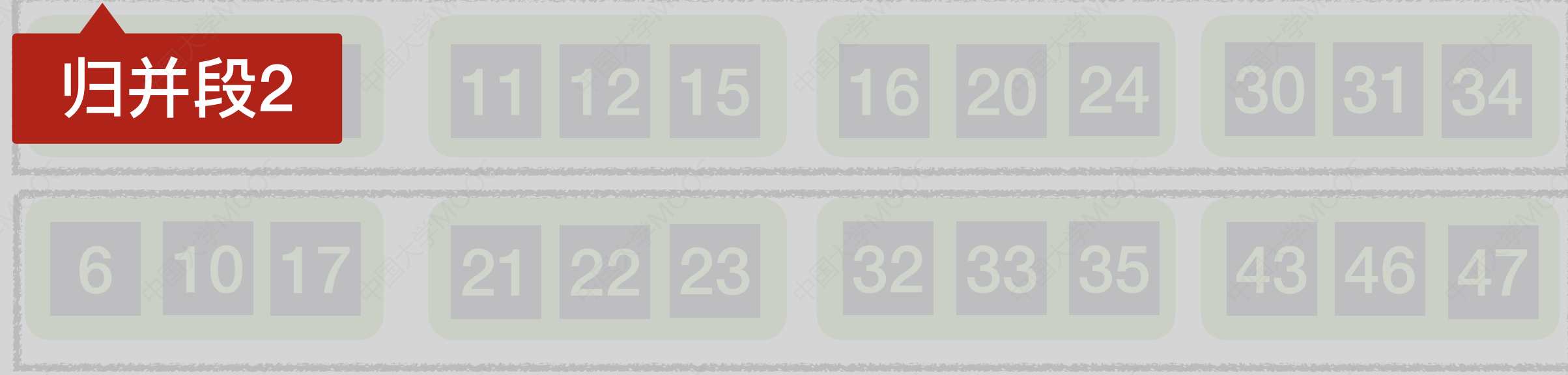
第二趟归并



归并段1

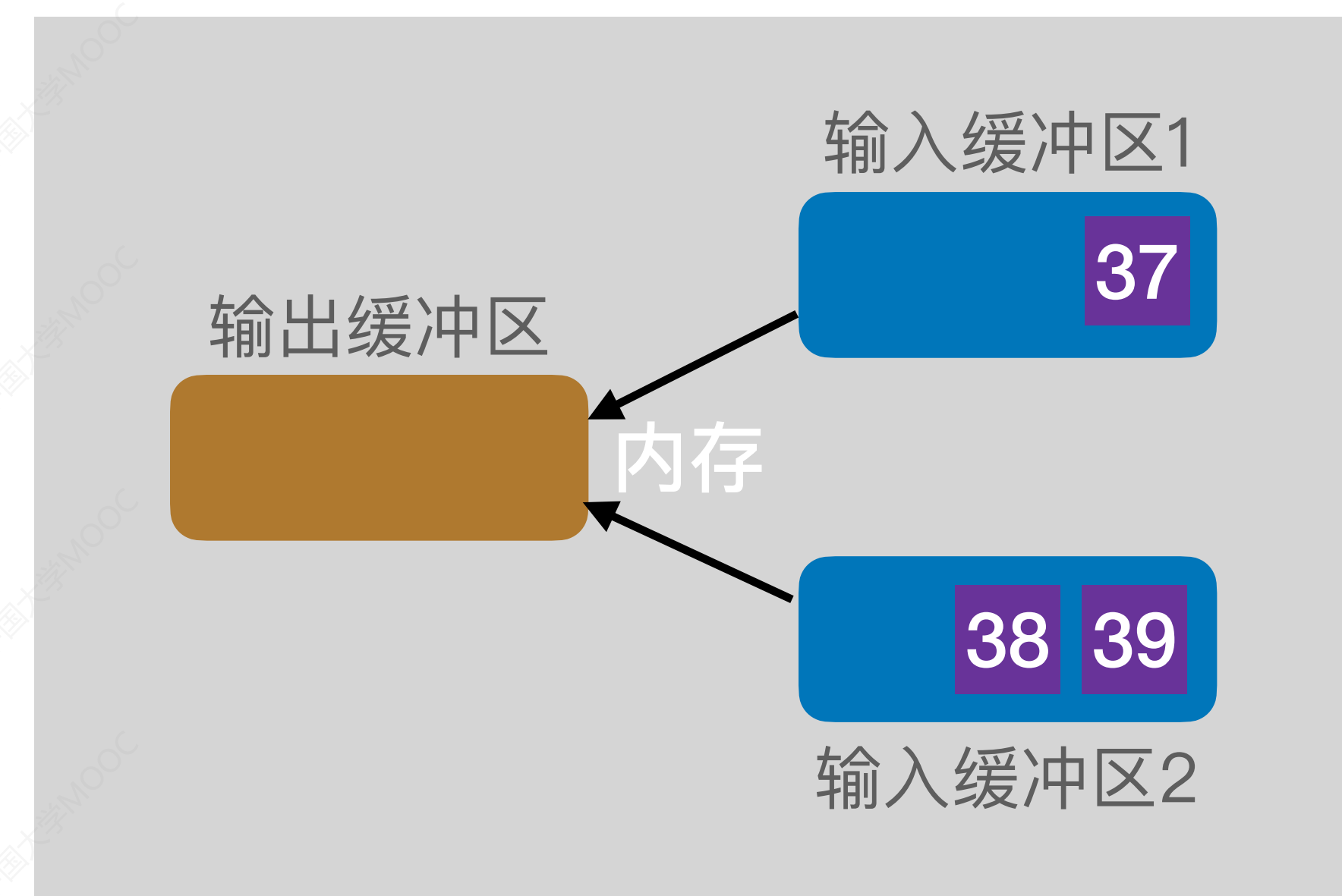


归并段2



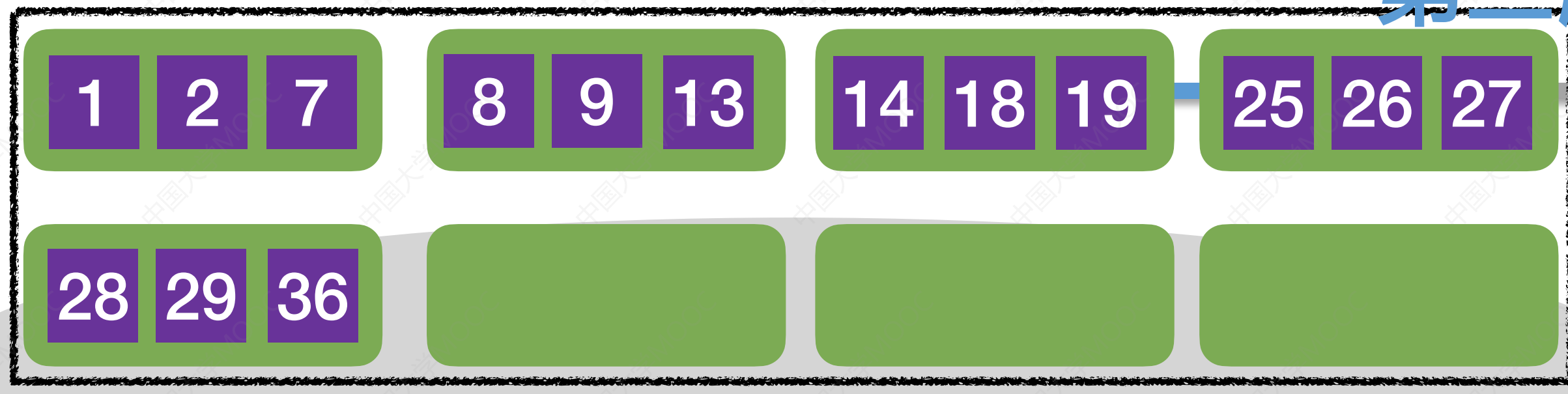
读磁盘

写磁盘

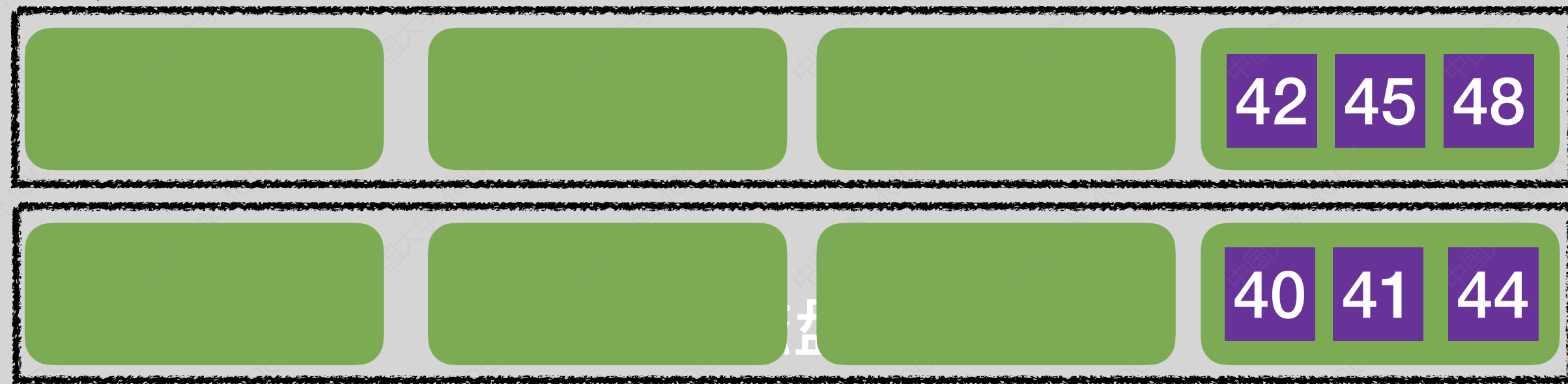


将两个有序归并段归并为一个

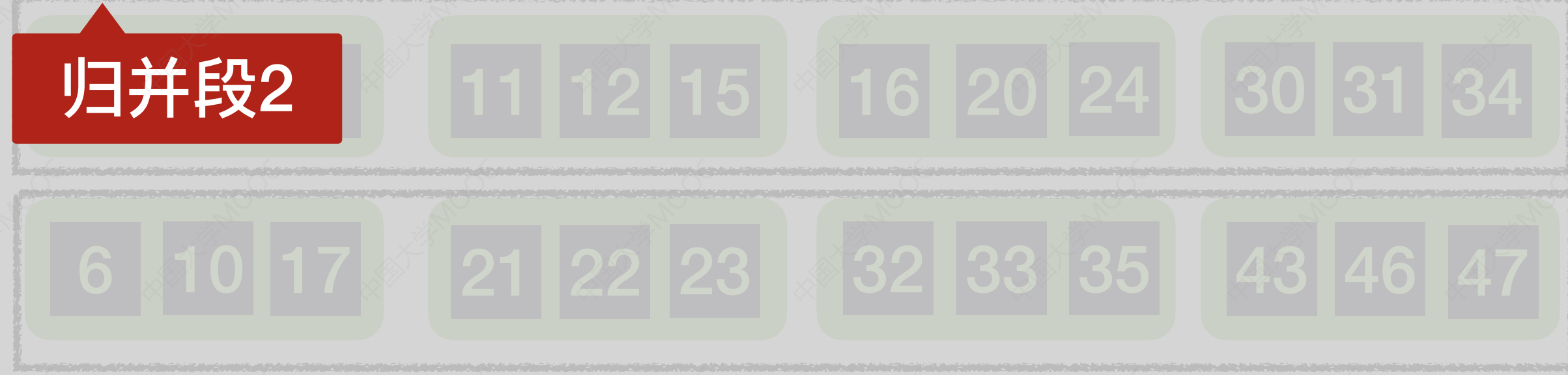
第二趟归并



归并段1

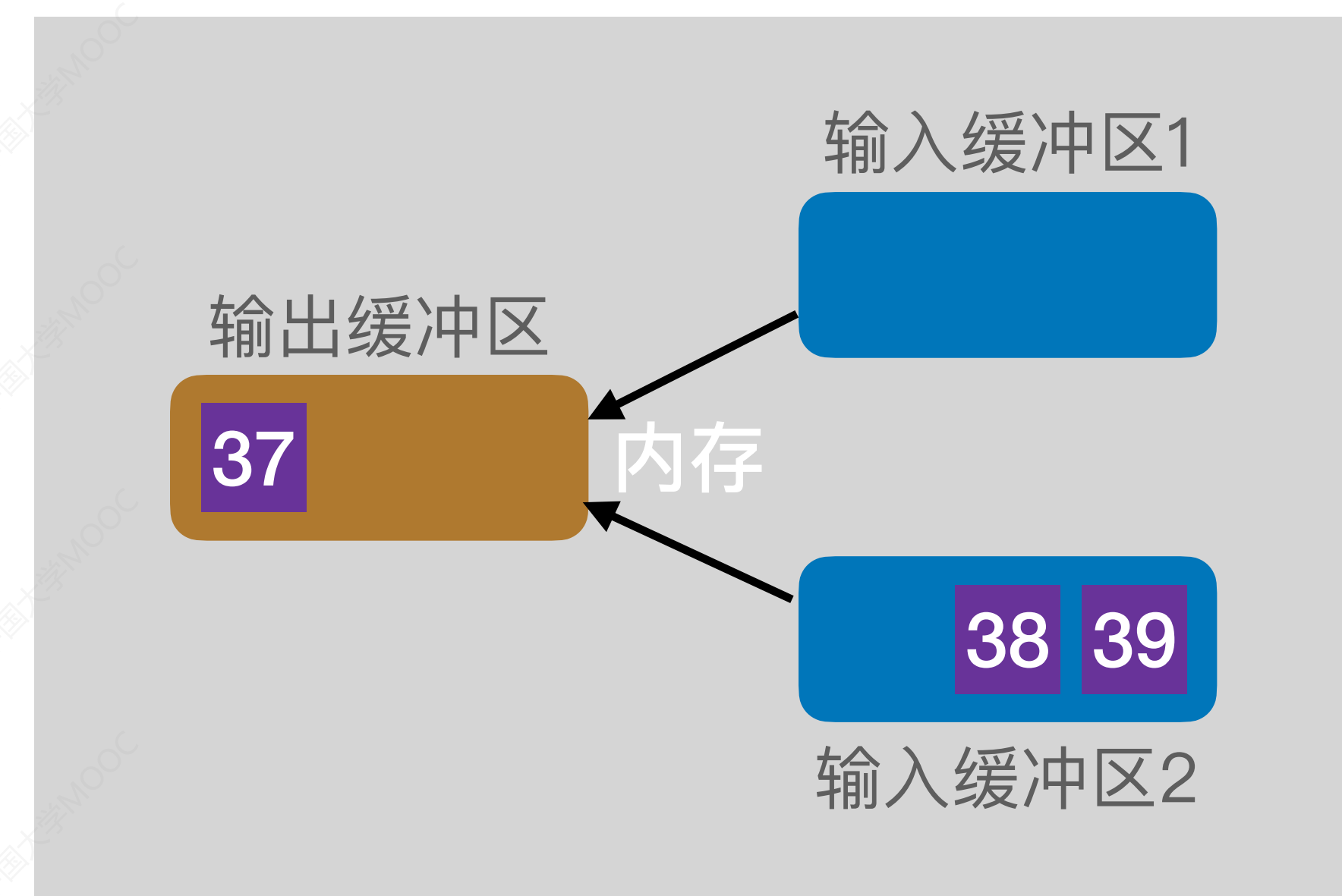


归并段2



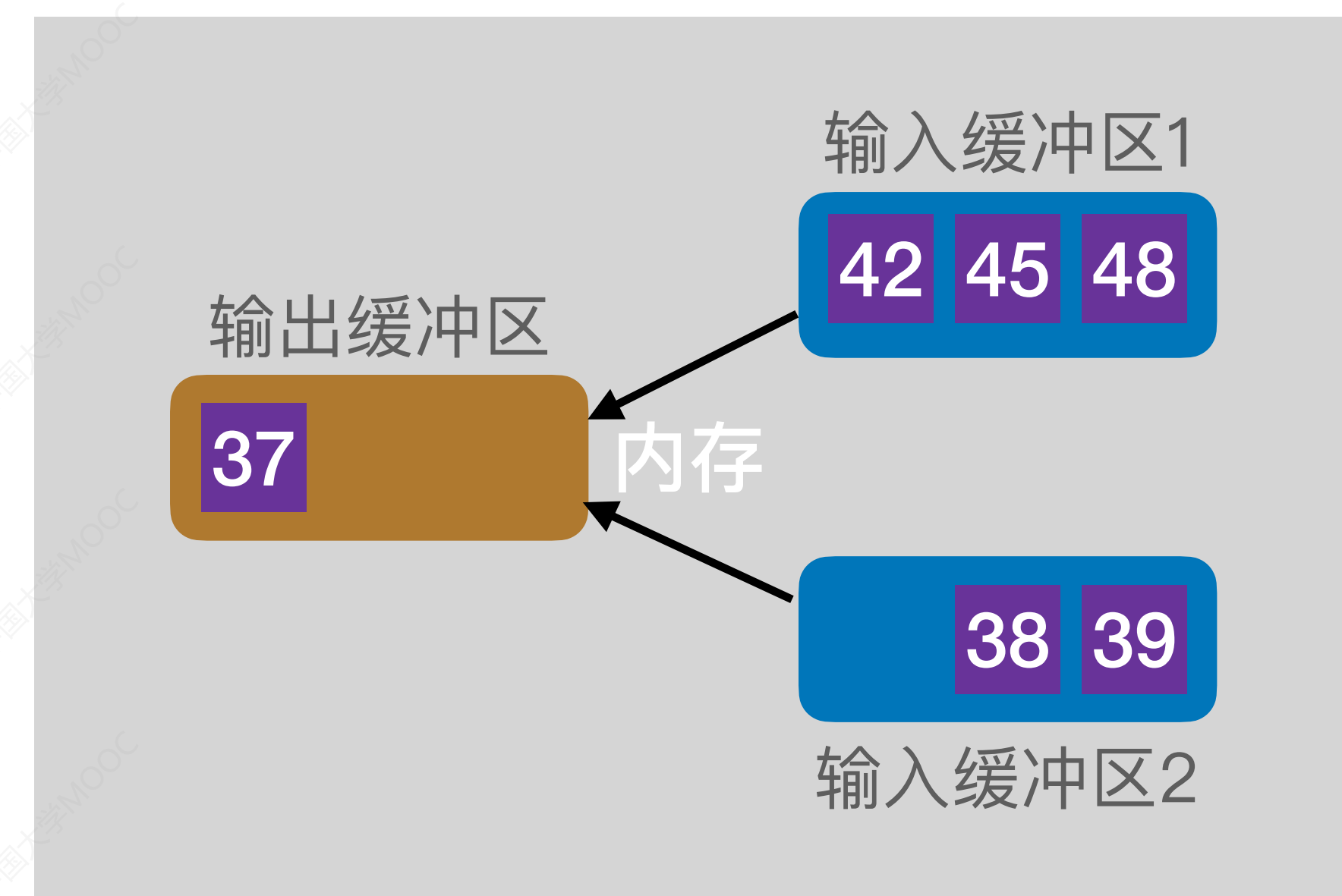
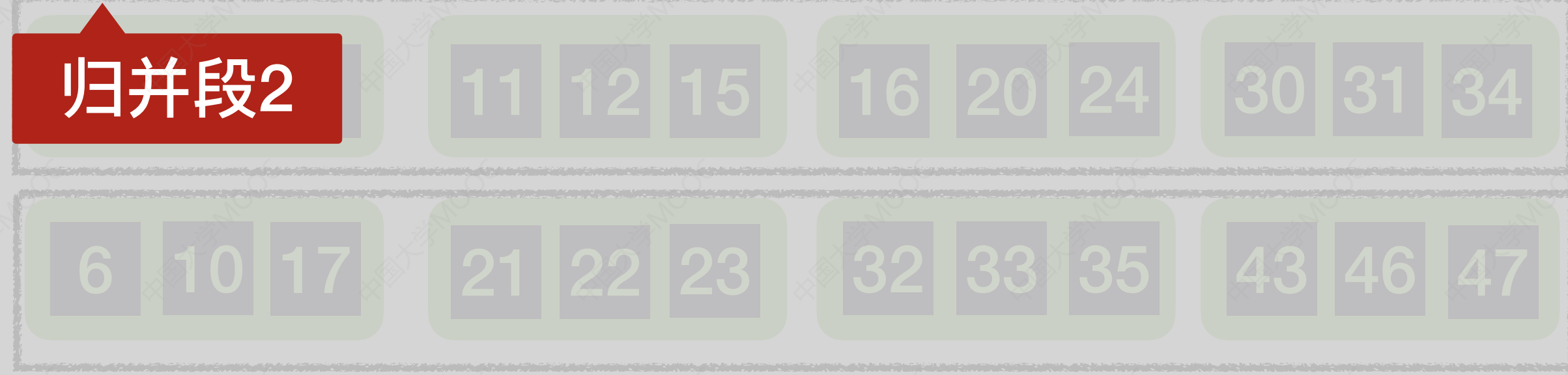
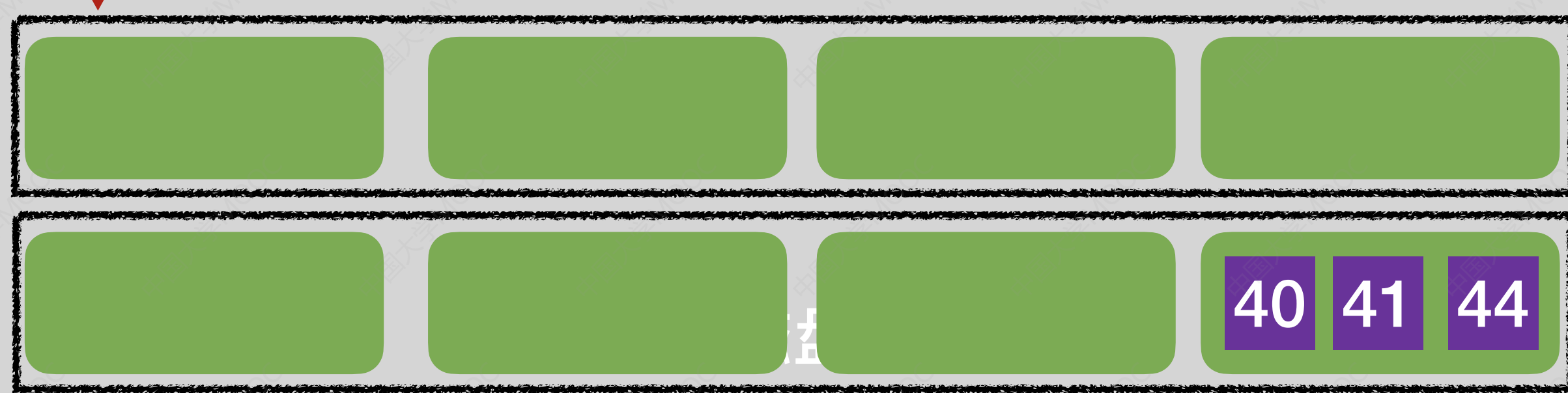
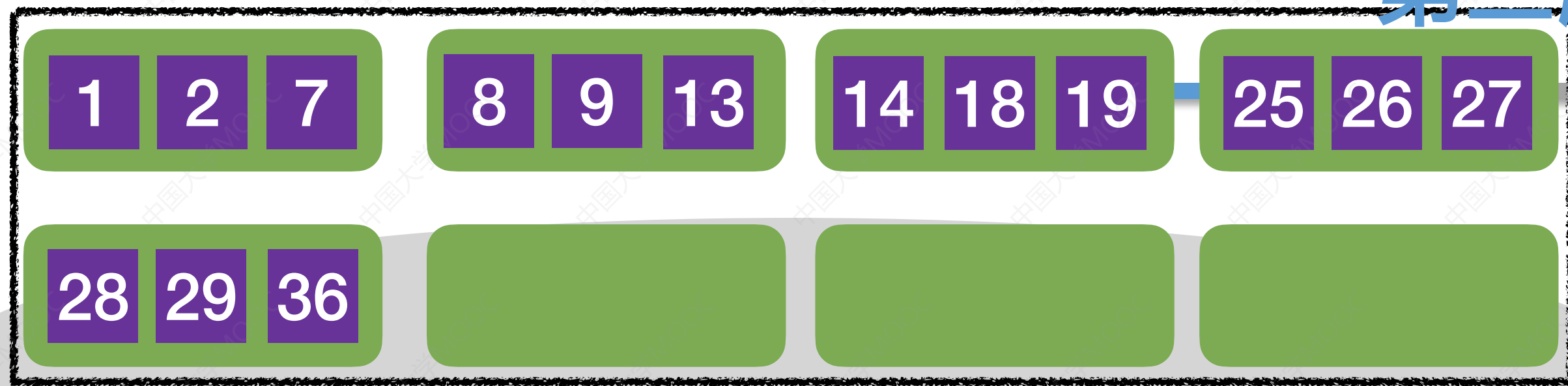
读磁盘

写磁盘



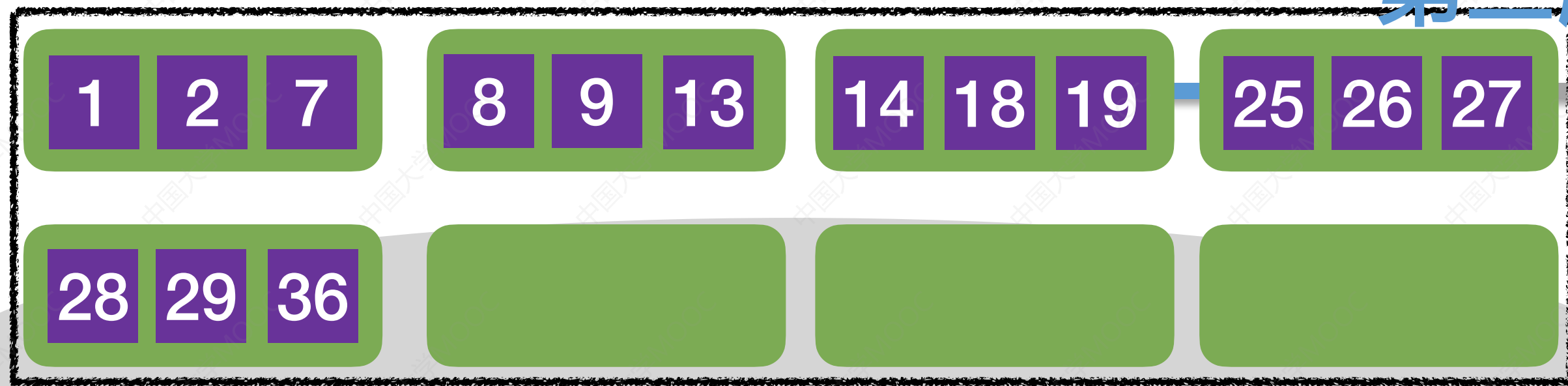
将两个有序归并段归并为一个

第二趟归并

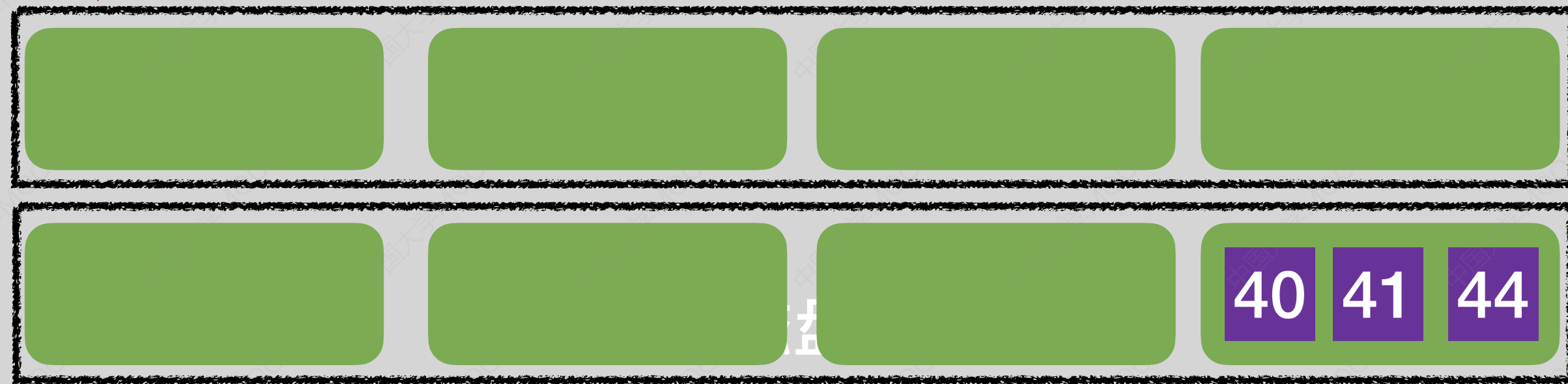


将两个有序归并段归并为一个

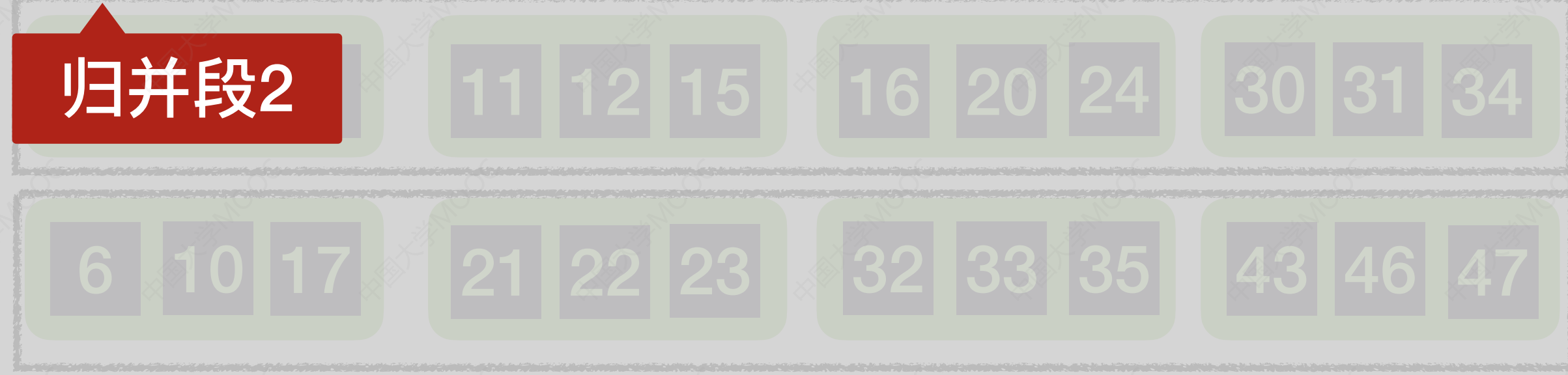
第二趟归并



归并段1

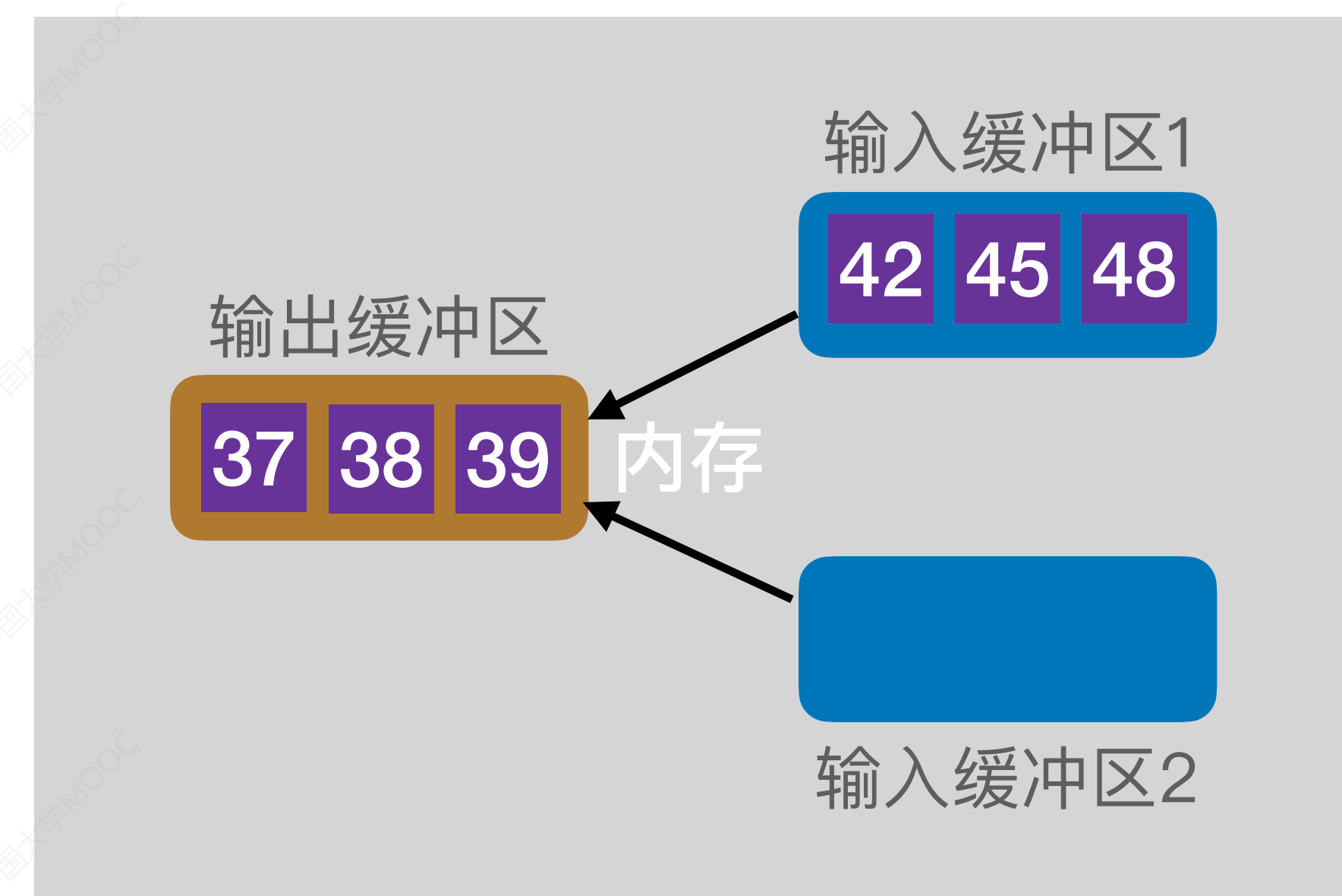


归并段2



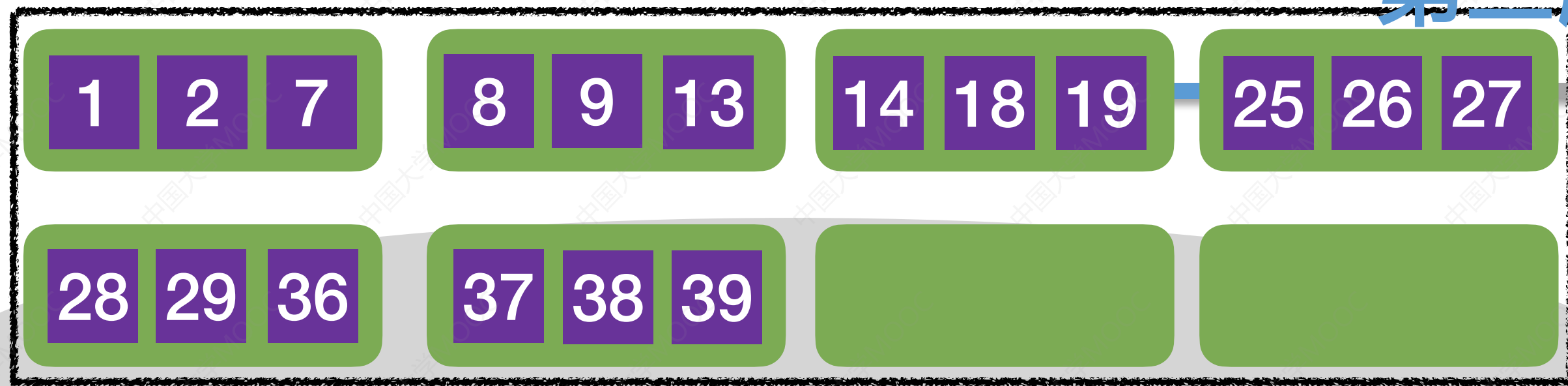
读磁盘

写磁盘

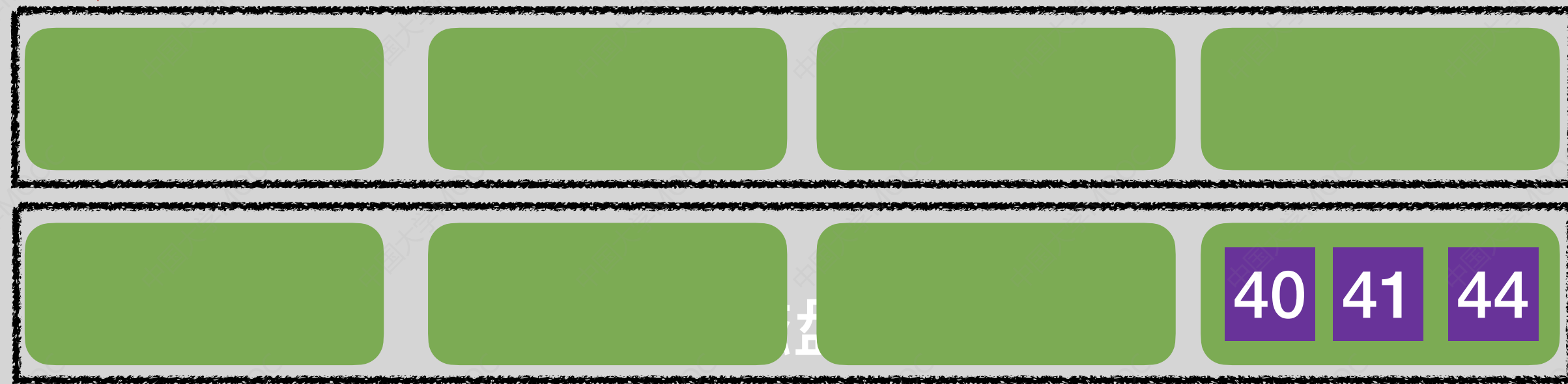


将两个有序归并段归并为一个

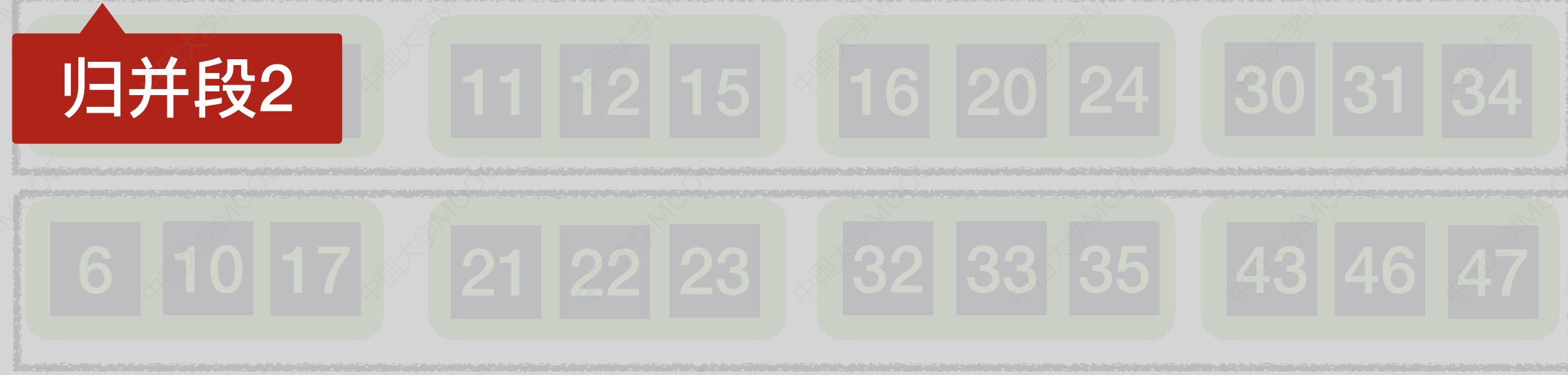
第二趟归并



归并段1

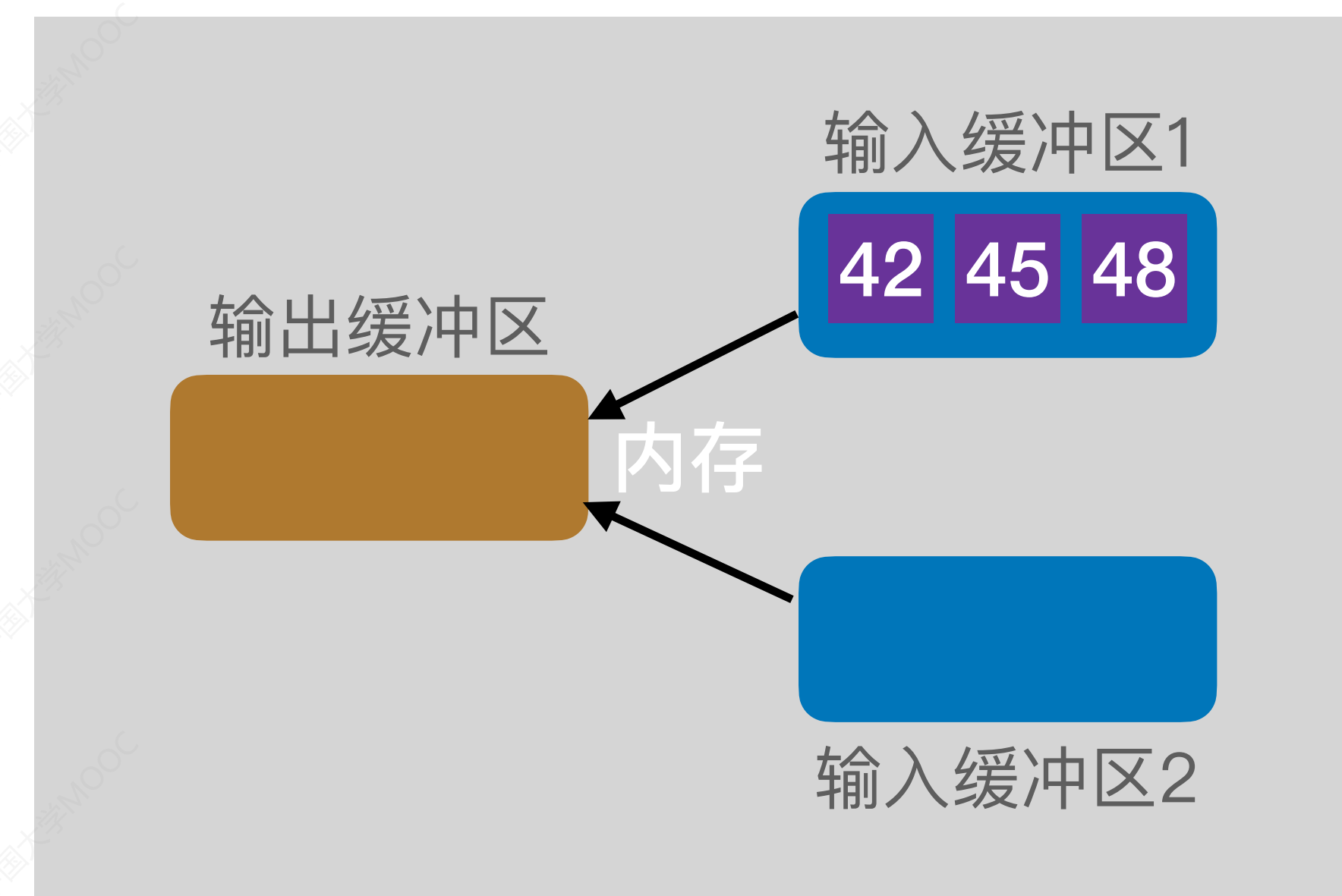


归并段2



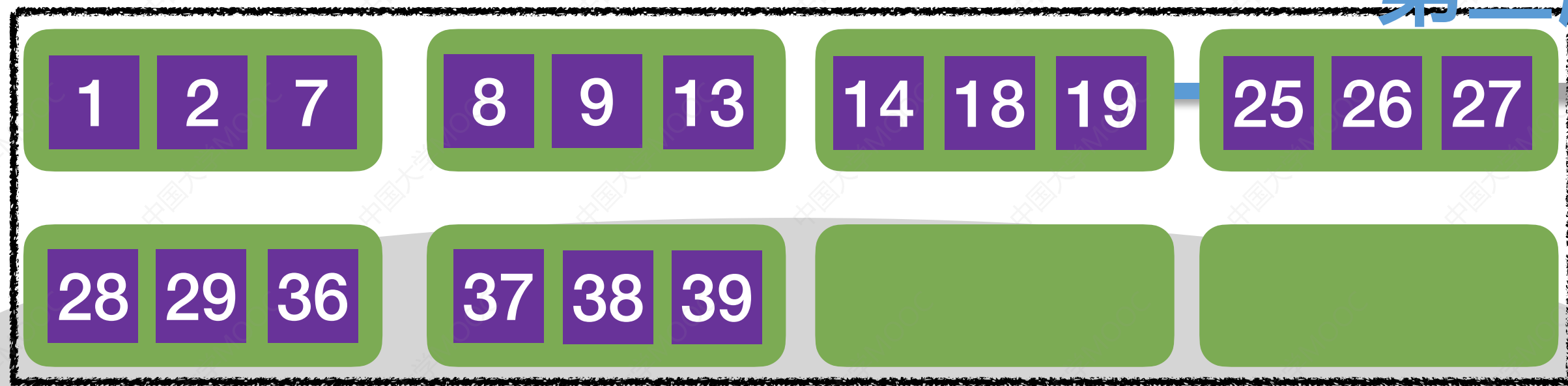
读磁盘

写磁盘

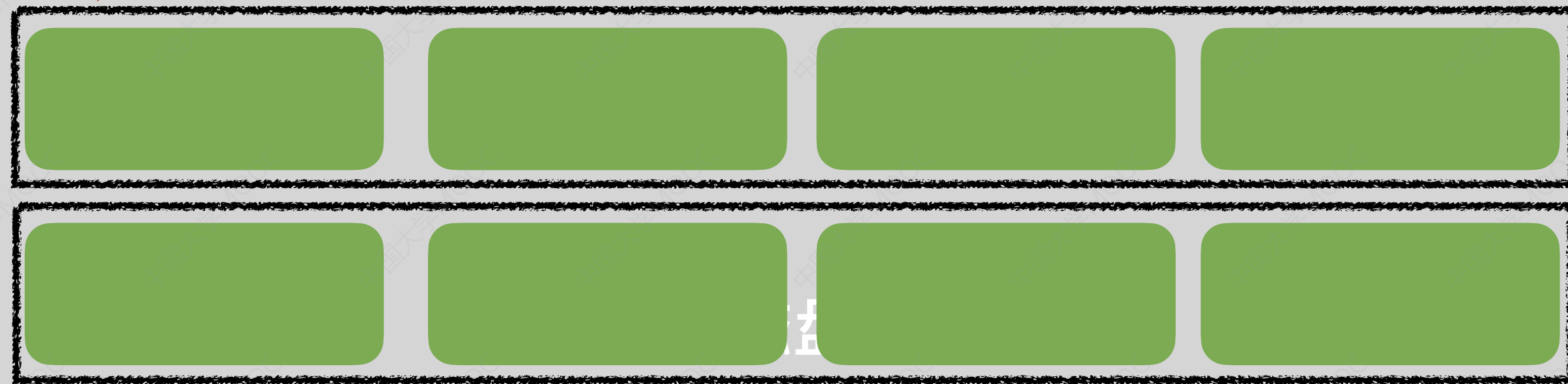


将两个有序归并段归并为一个

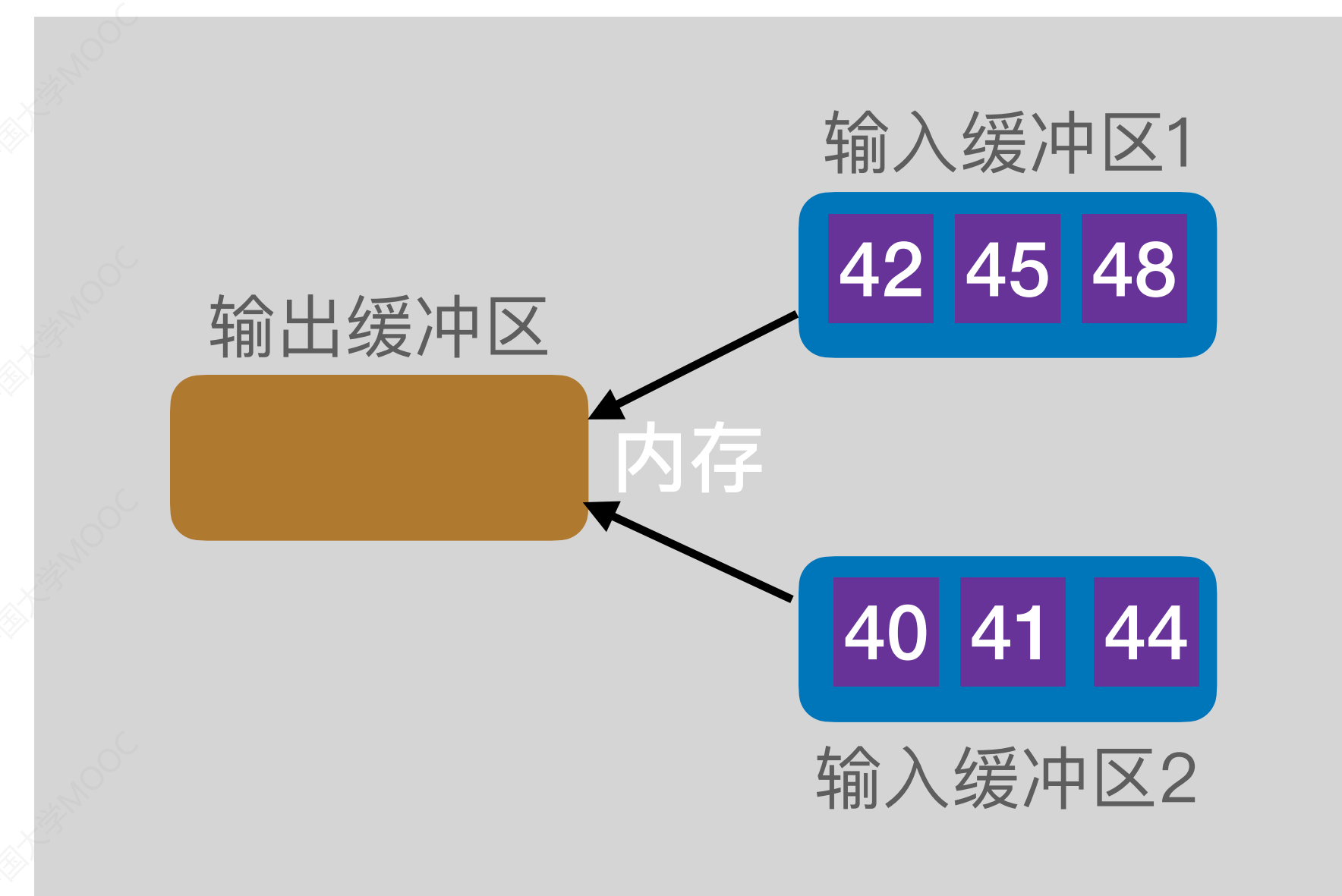
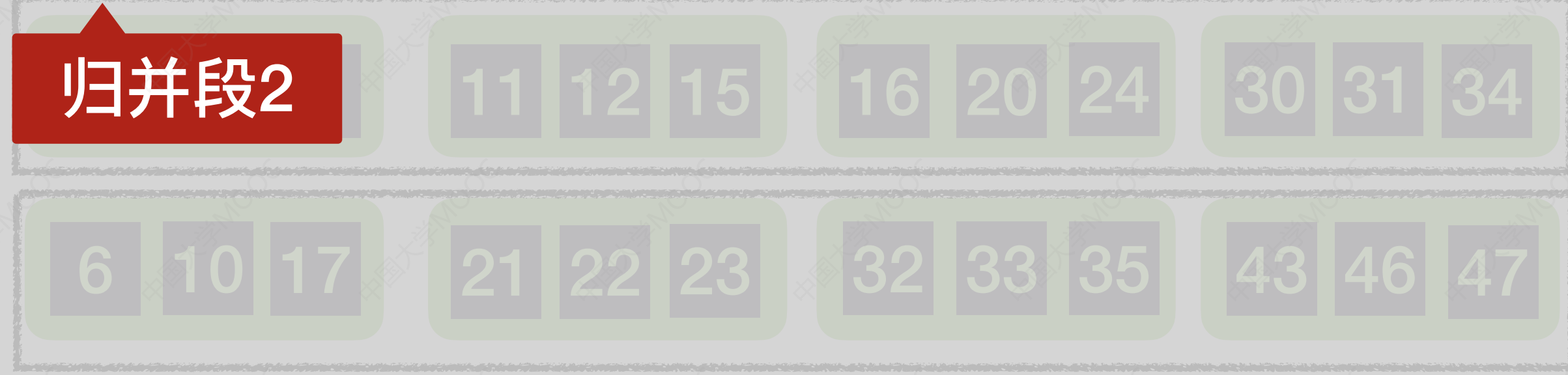
第二趟归并



归并段1

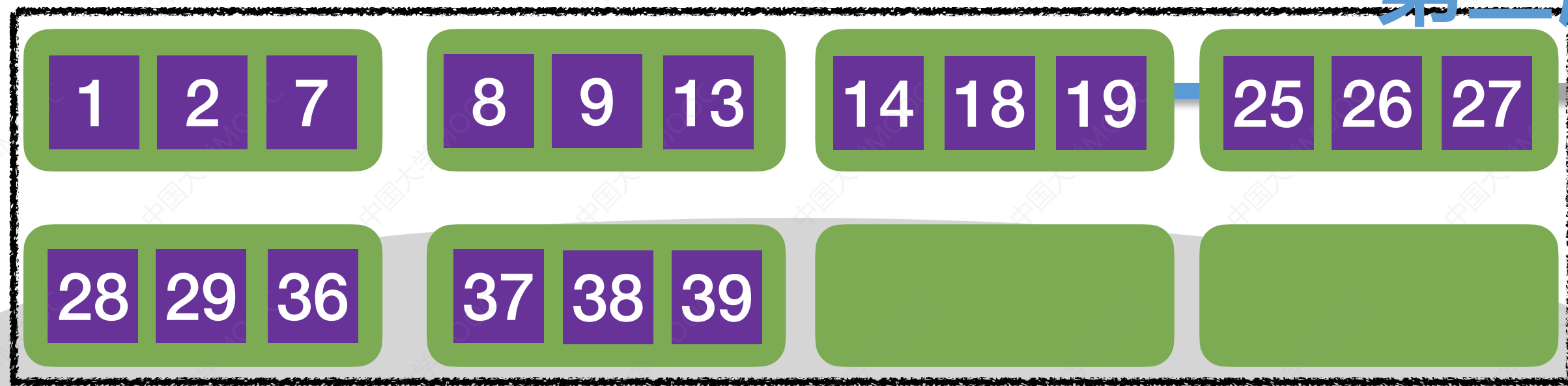


归并段2

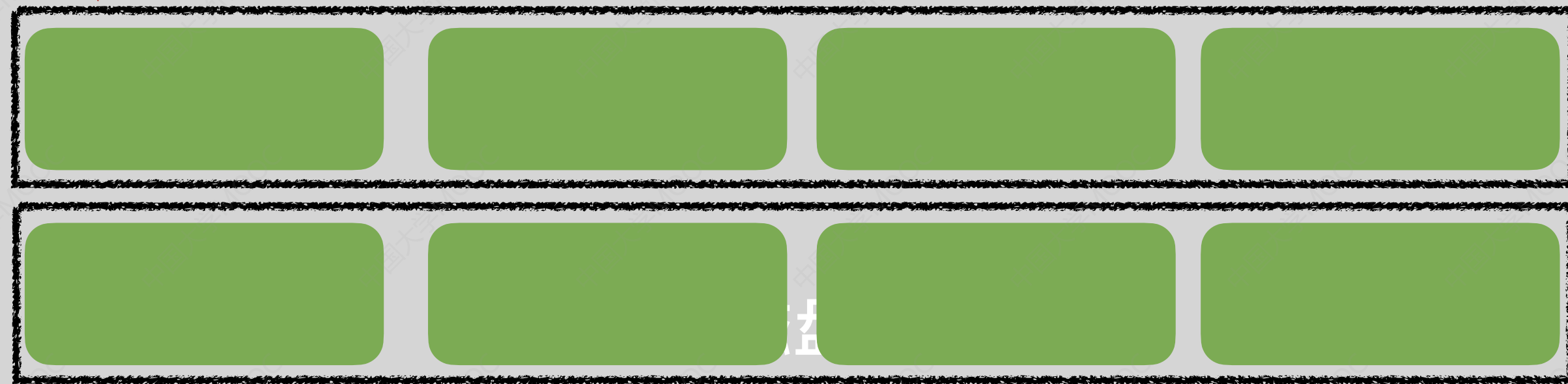


将两个有序归并段归并为一个

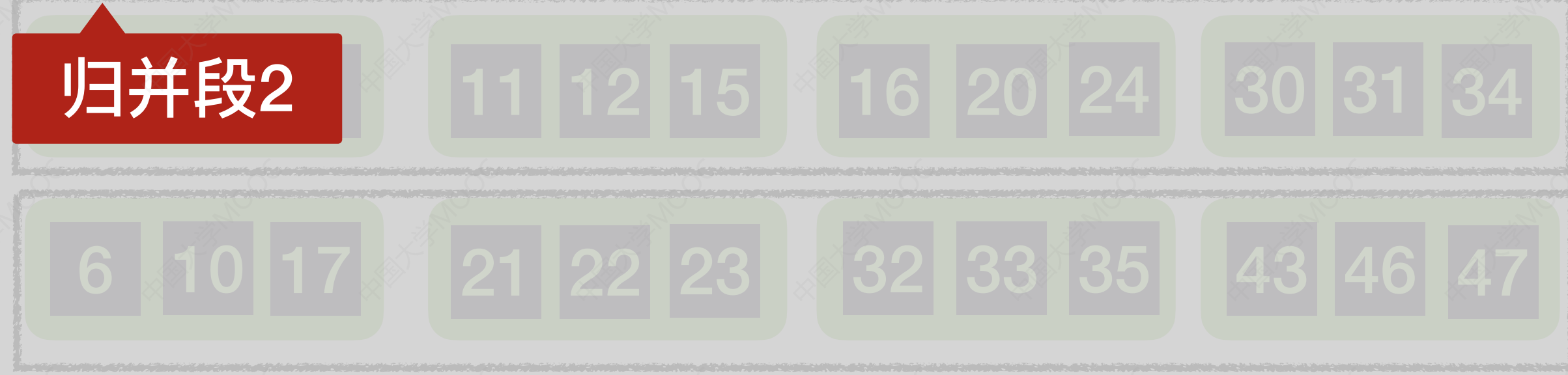
第二趟归并



归并段1

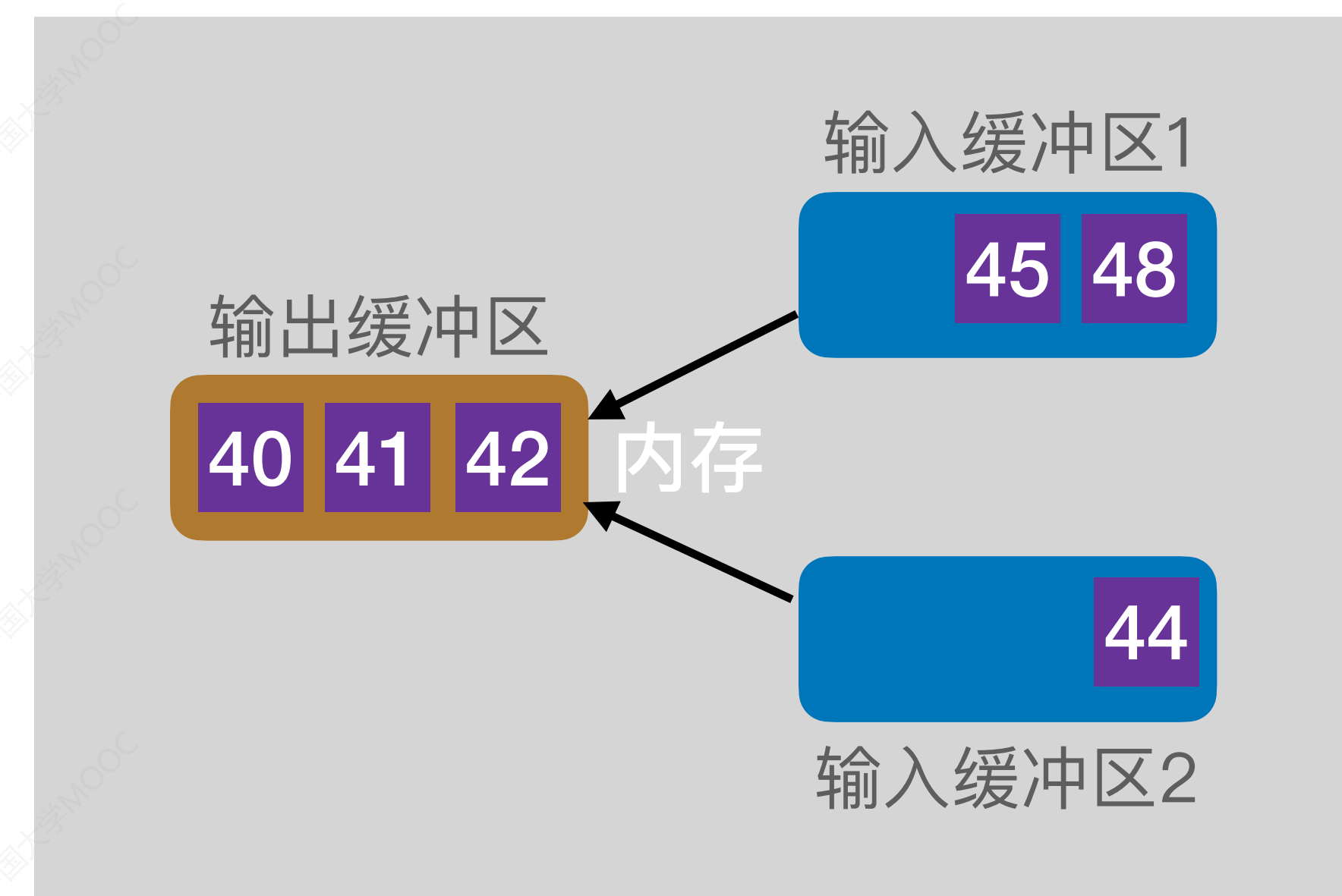


归并段2



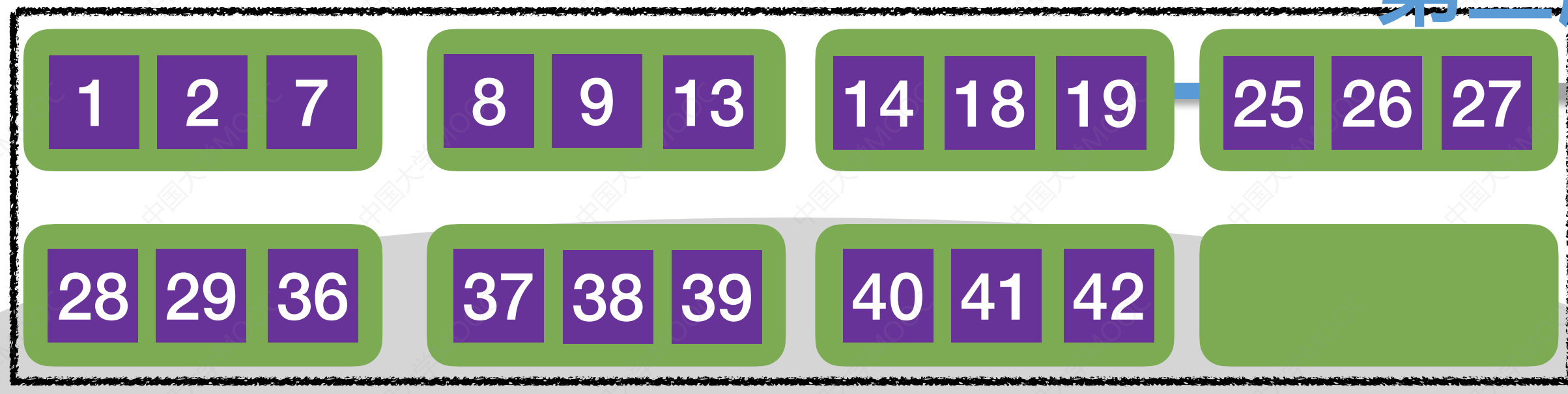
读磁盘

写磁盘

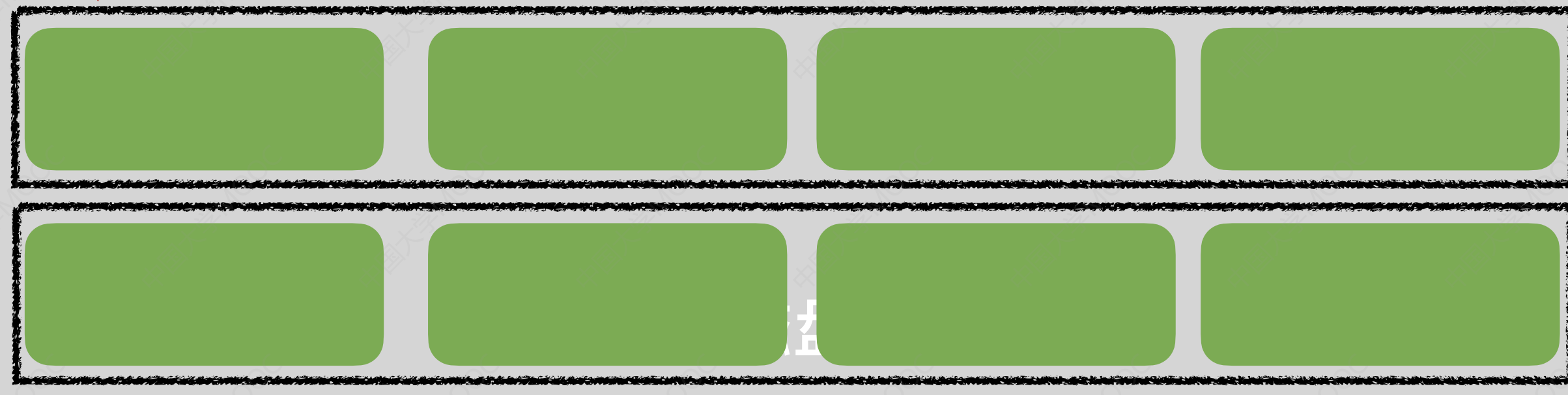


将两个有序归并段归并为一个

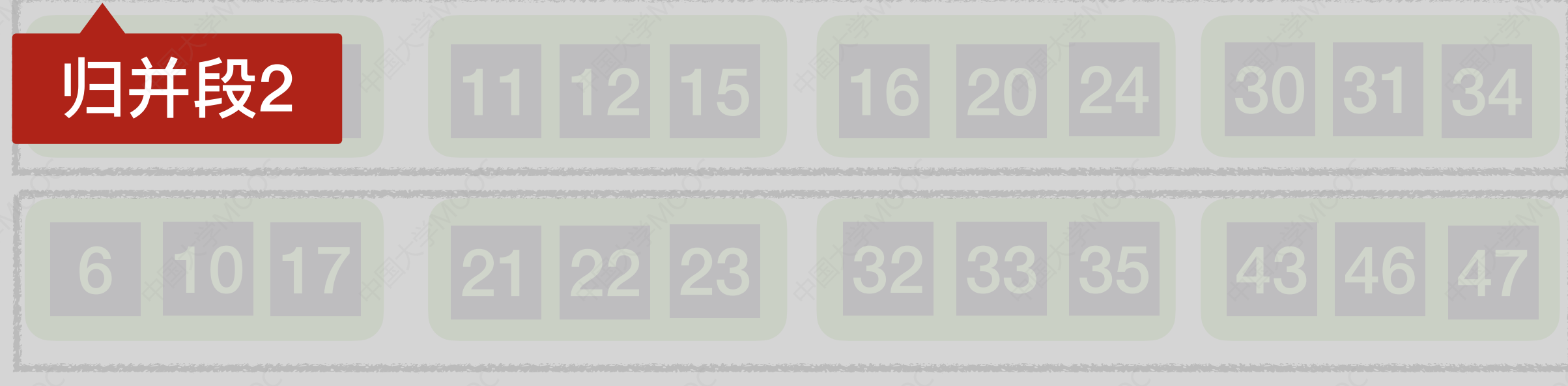
第二趟归并



归并段1

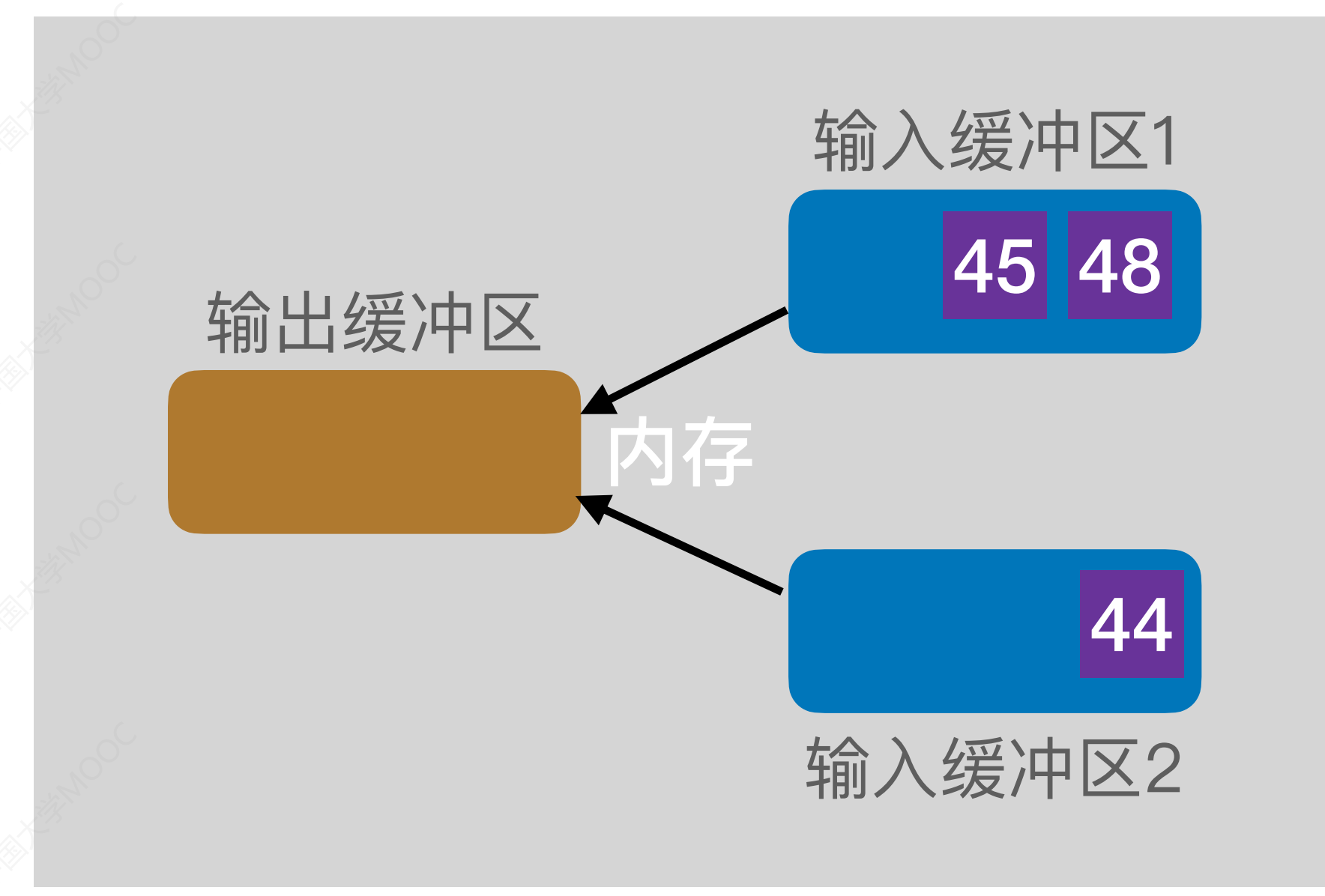


归并段2



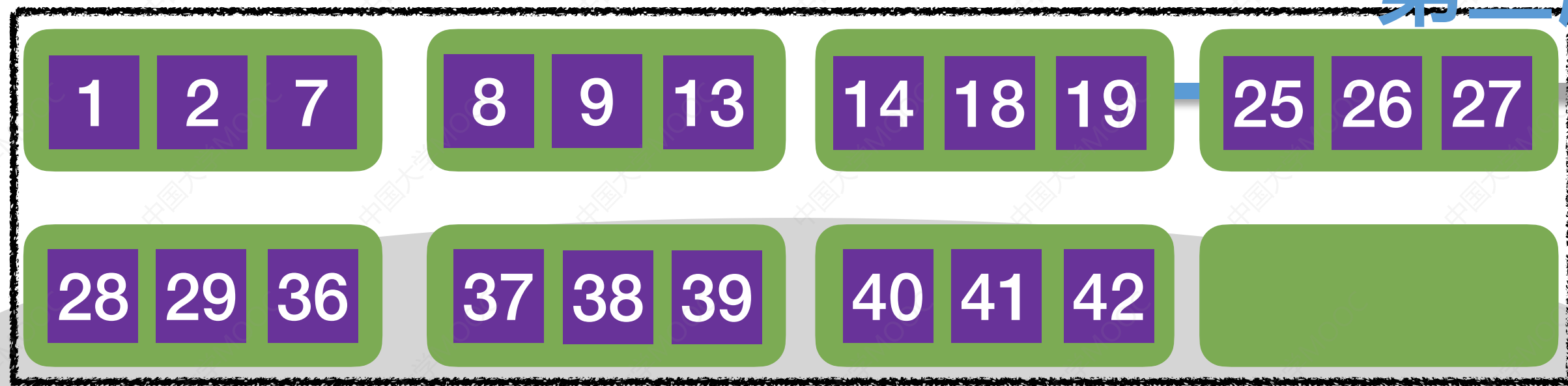
读磁盘

写磁盘

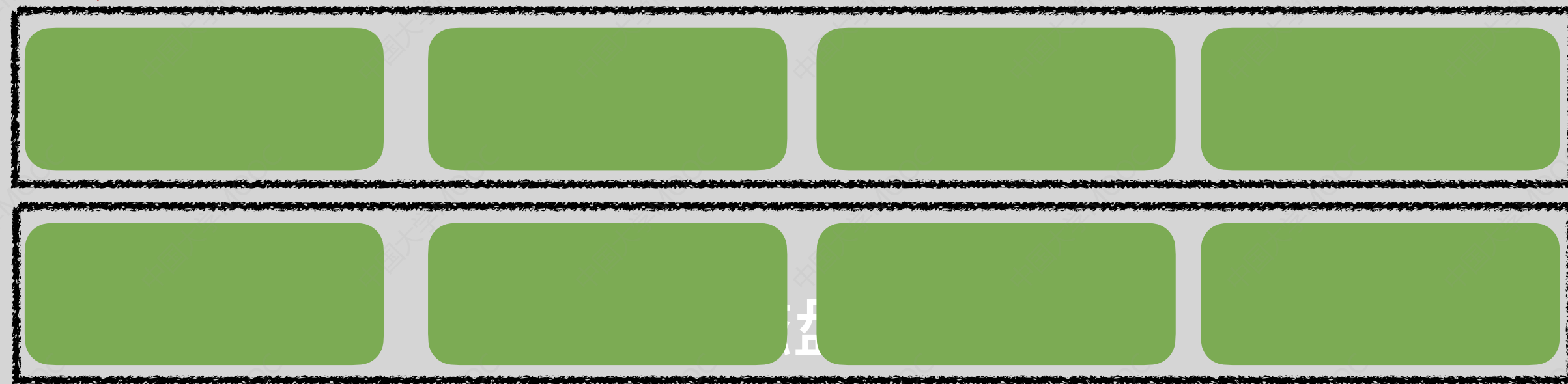


将两个有序归并段归并为一个

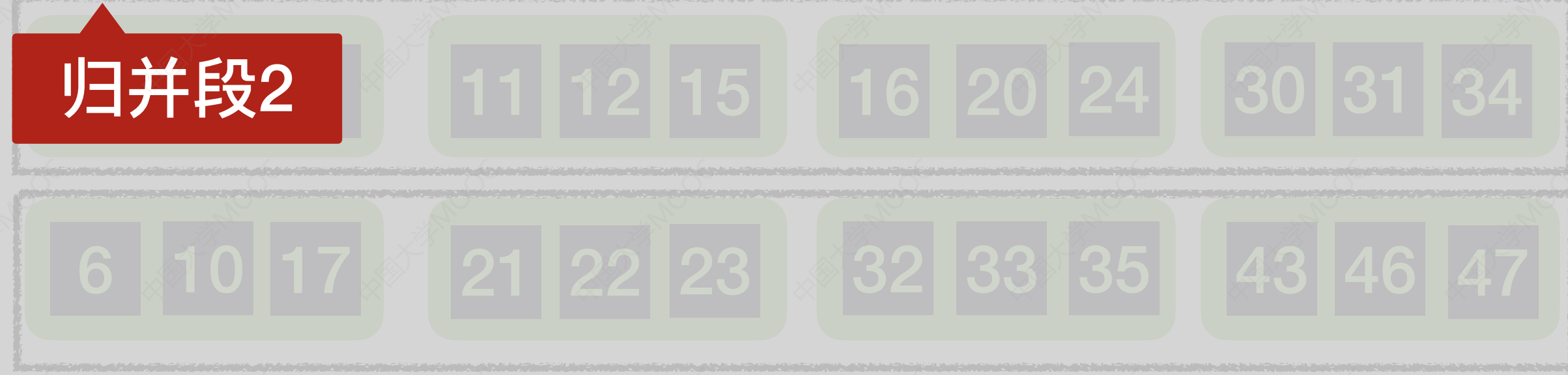
第二趟归并



归并段1

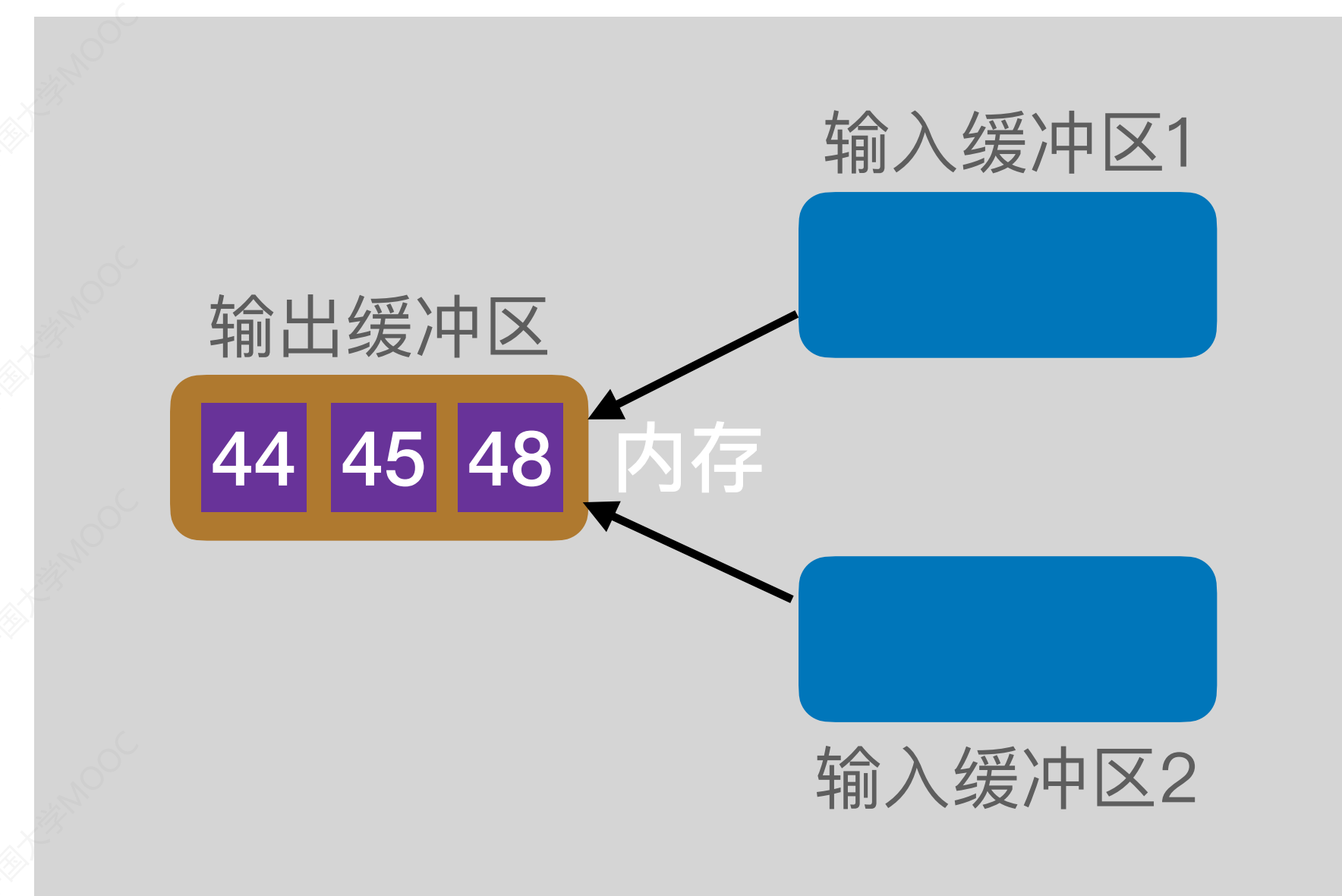


归并段2



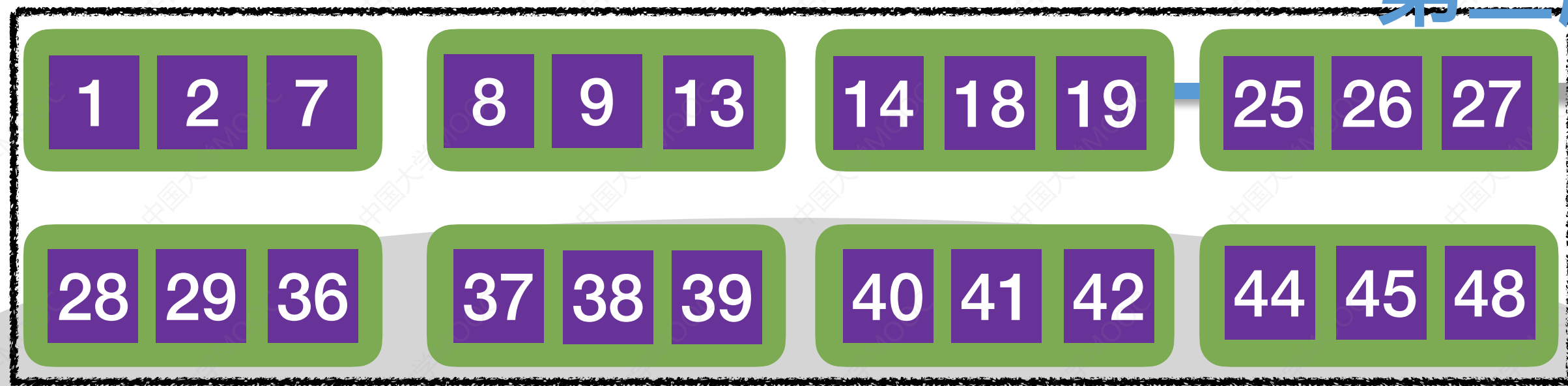
读磁盘

写磁盘

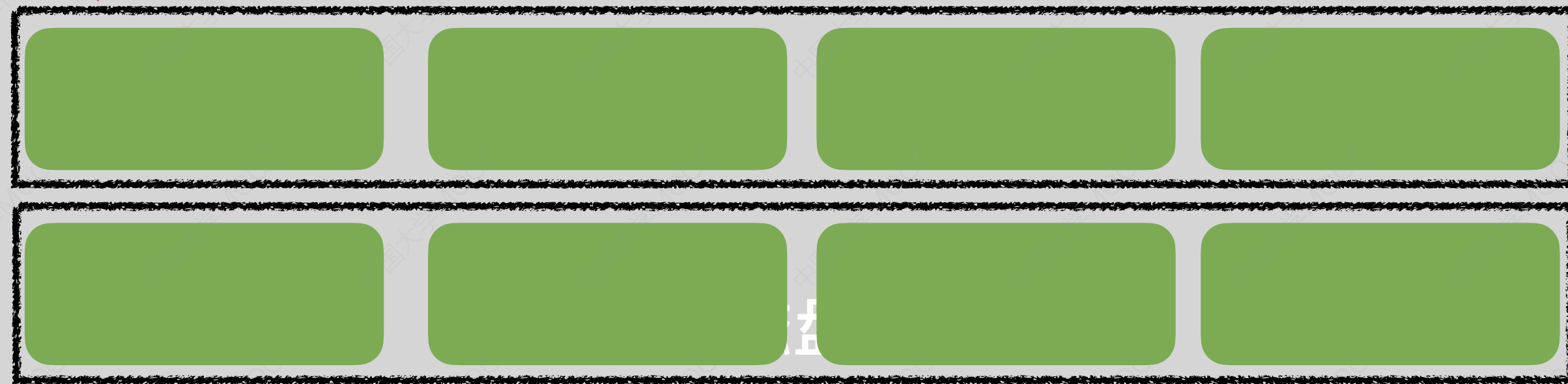


将两个有序归并段归并为一个

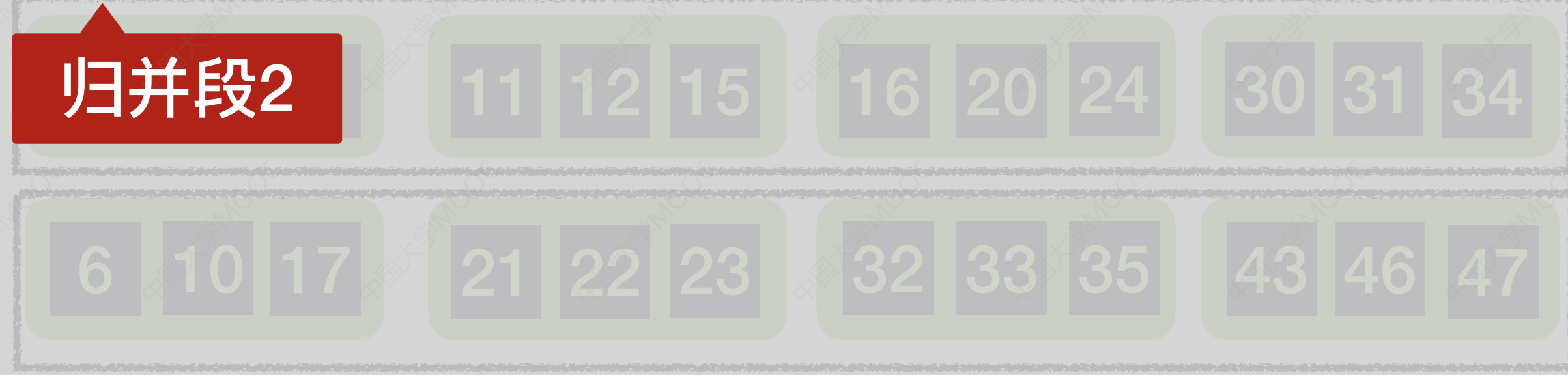
第二趟归并



归并段1

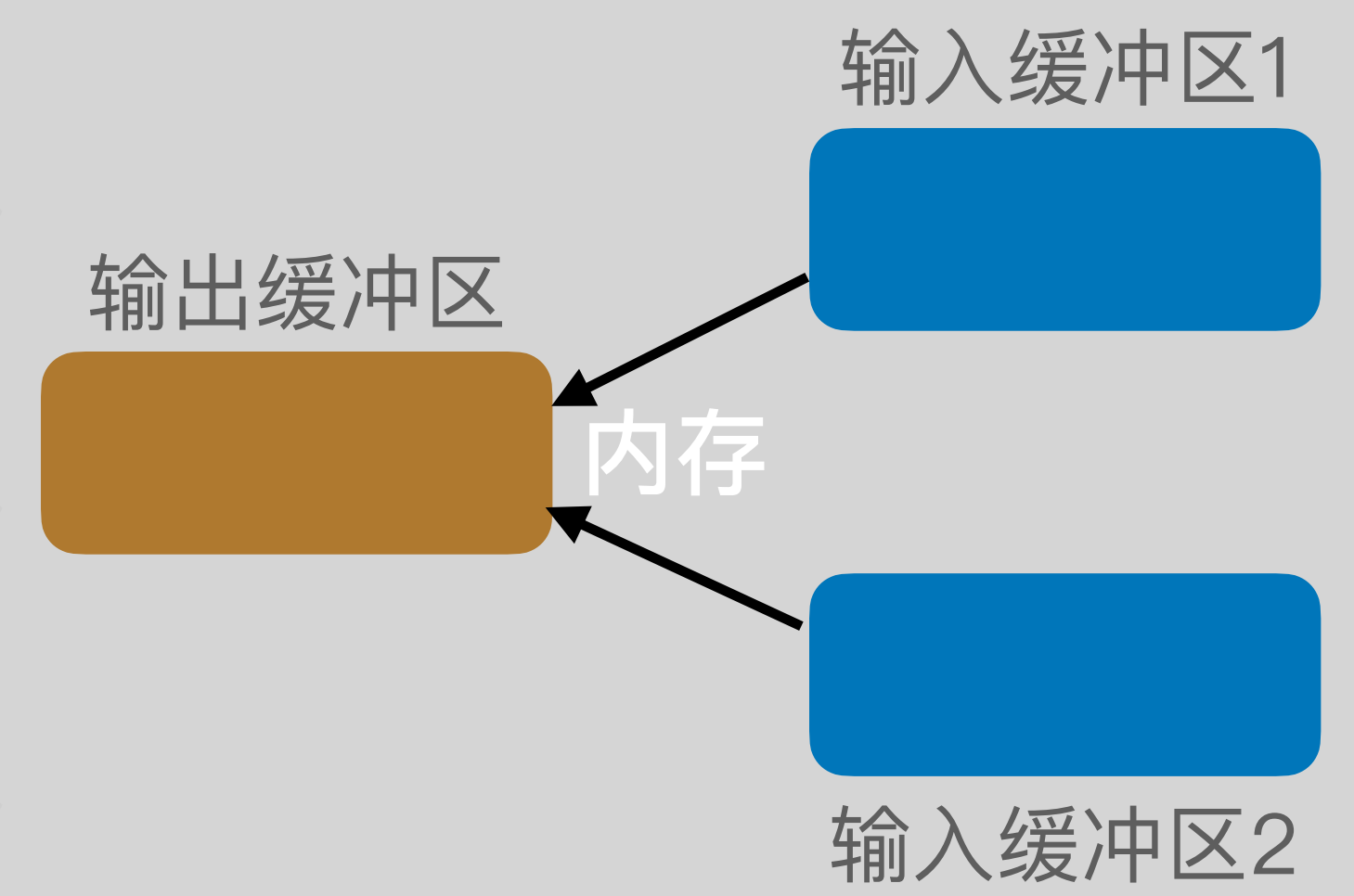


归并段2



读磁盘

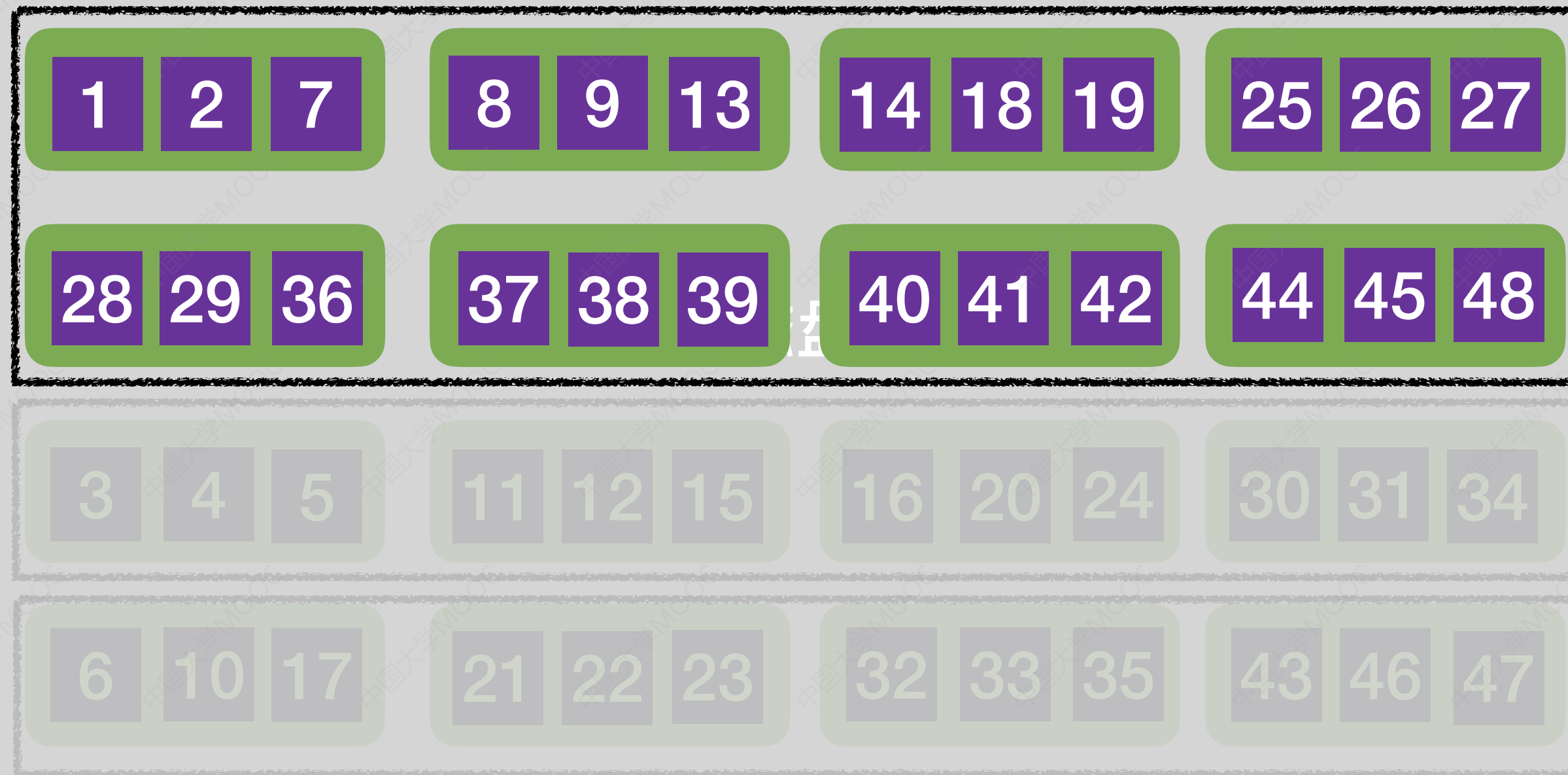
写磁盘



将两个有序归并段归并为一个

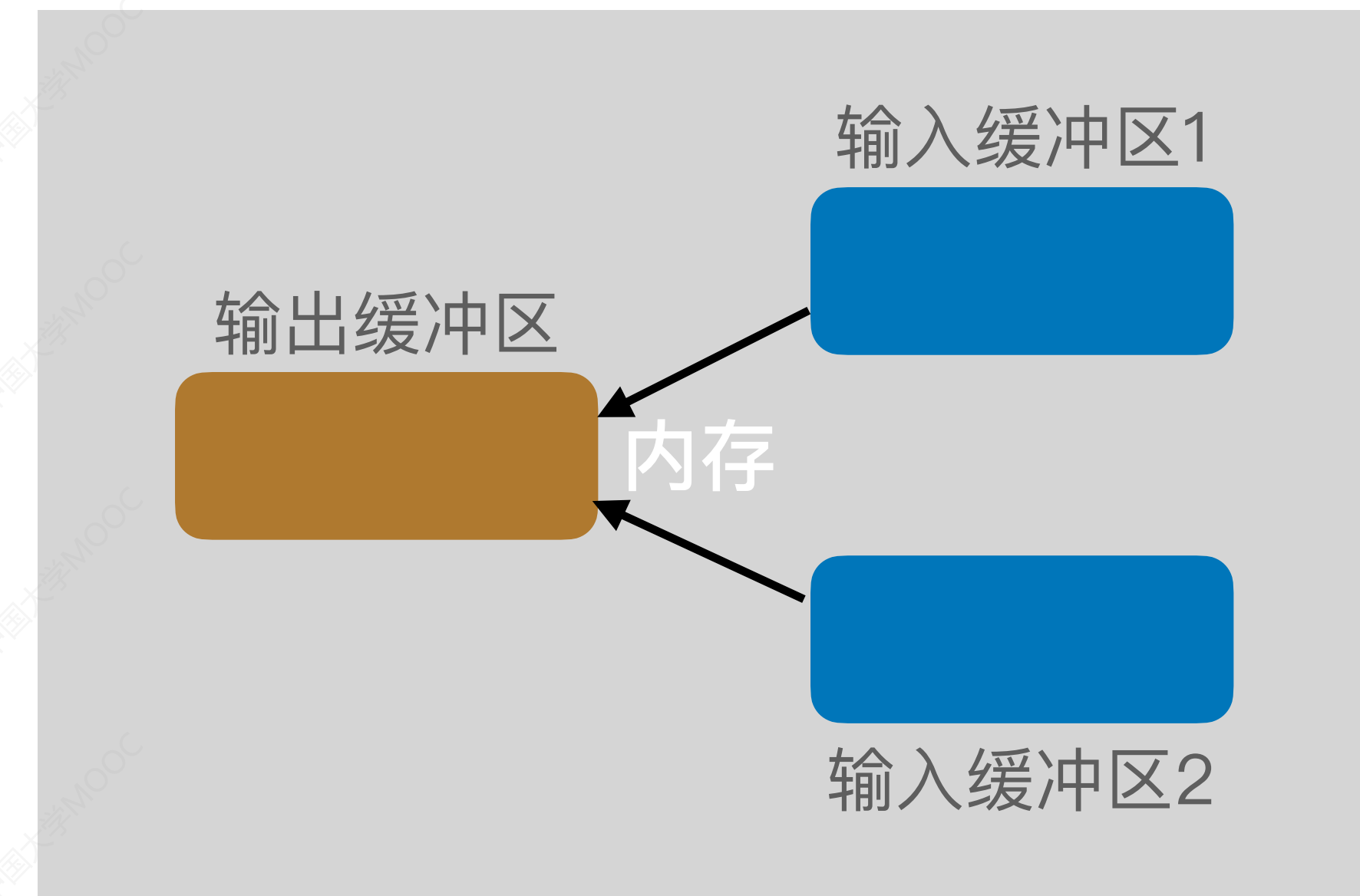
第二趟归并

归并为一个更长的有序序列



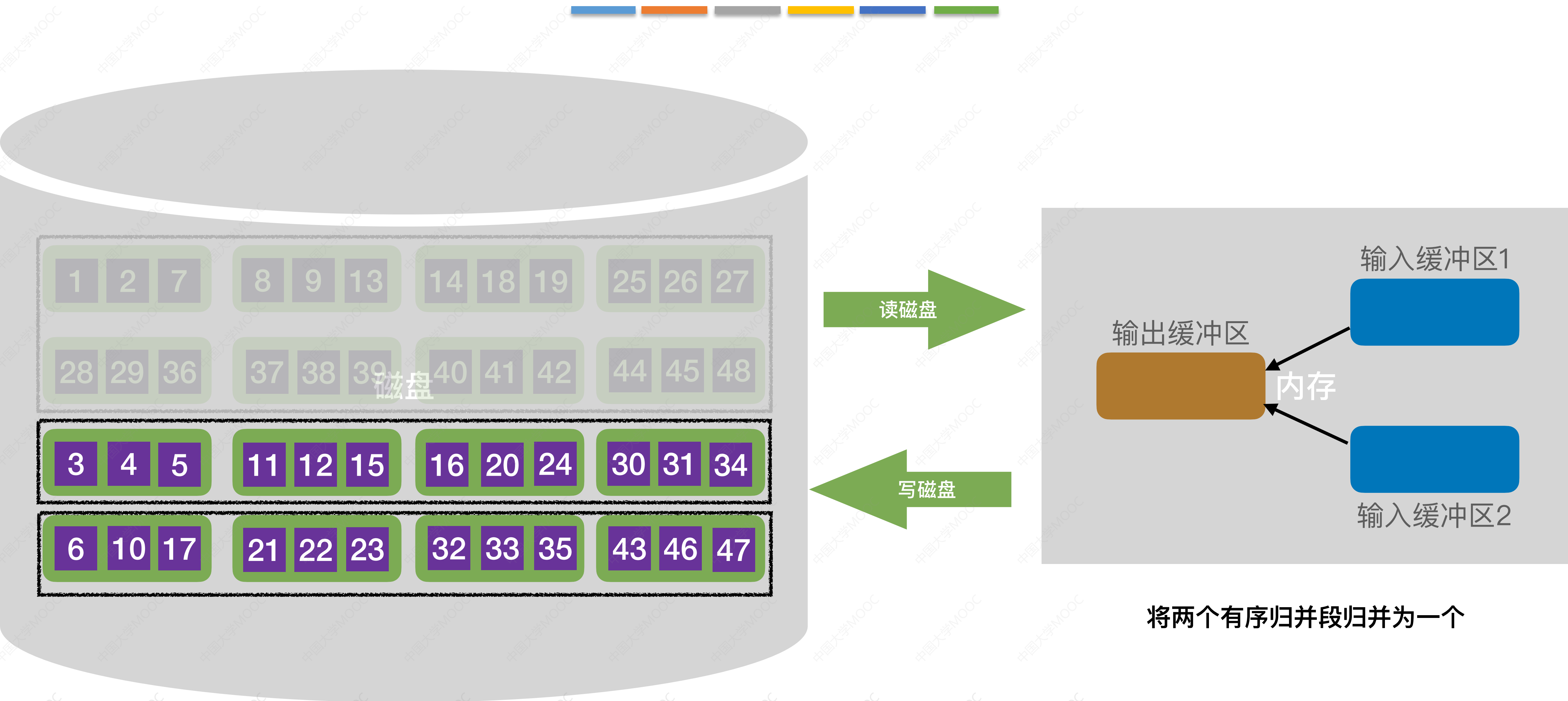
读磁盘

写磁盘

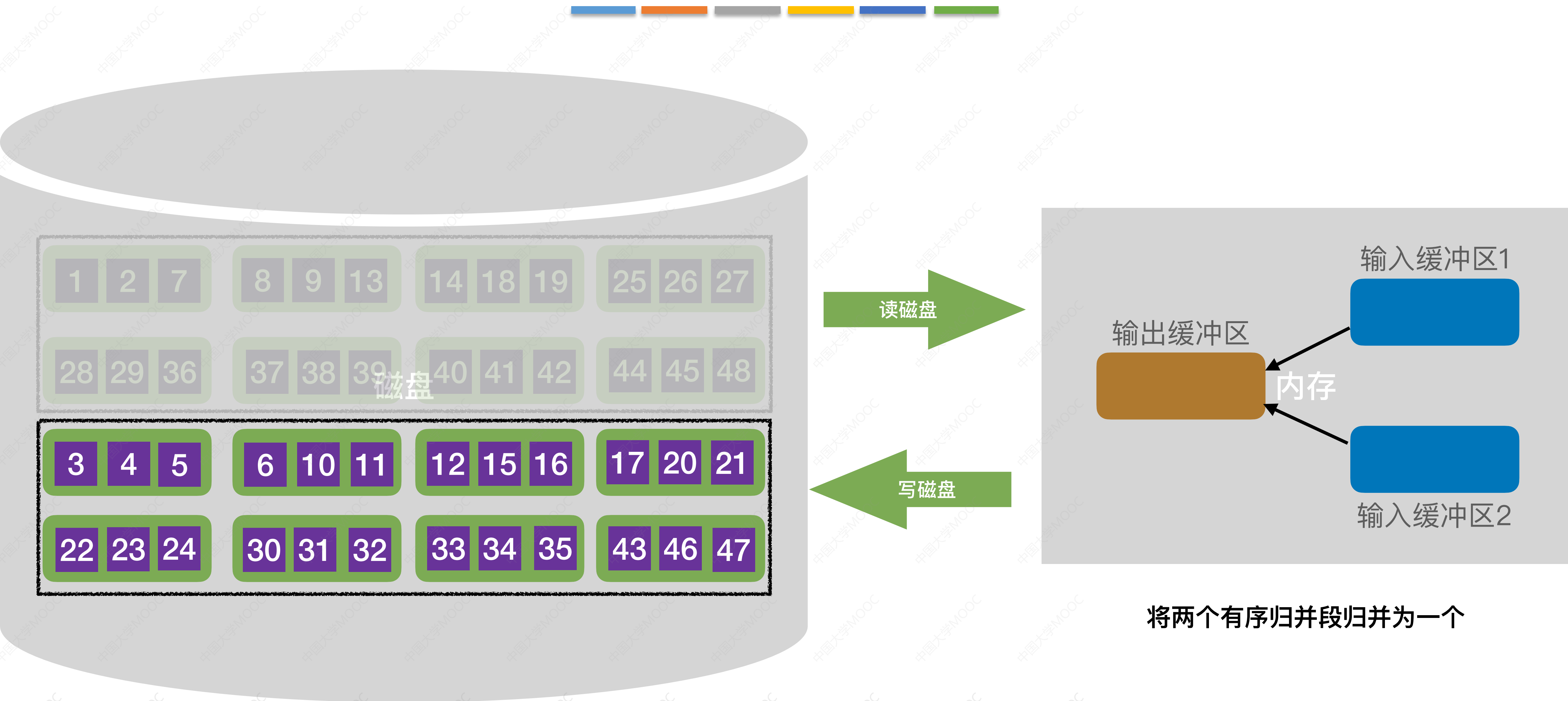


将两个有序归并段归并为一个

第二趟归并

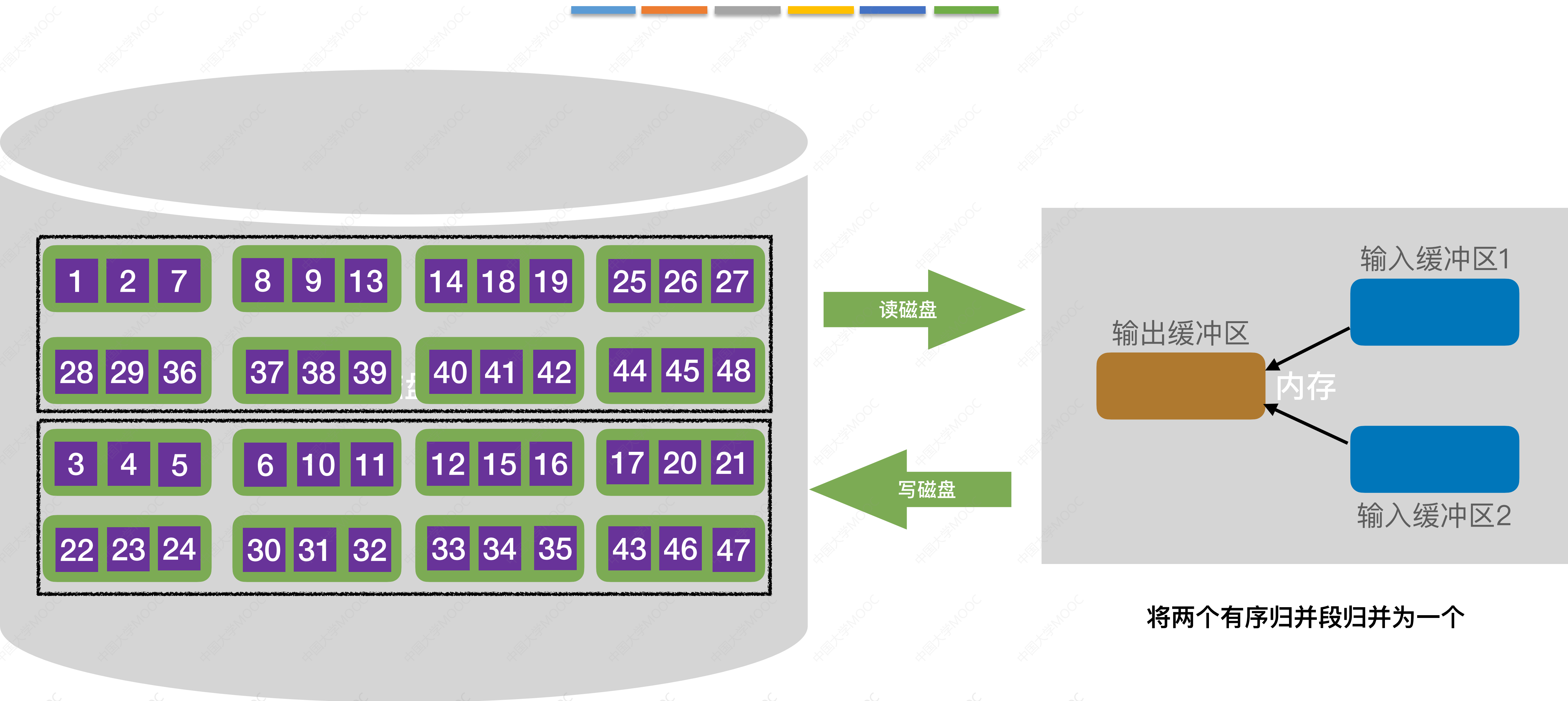


第二趟归并



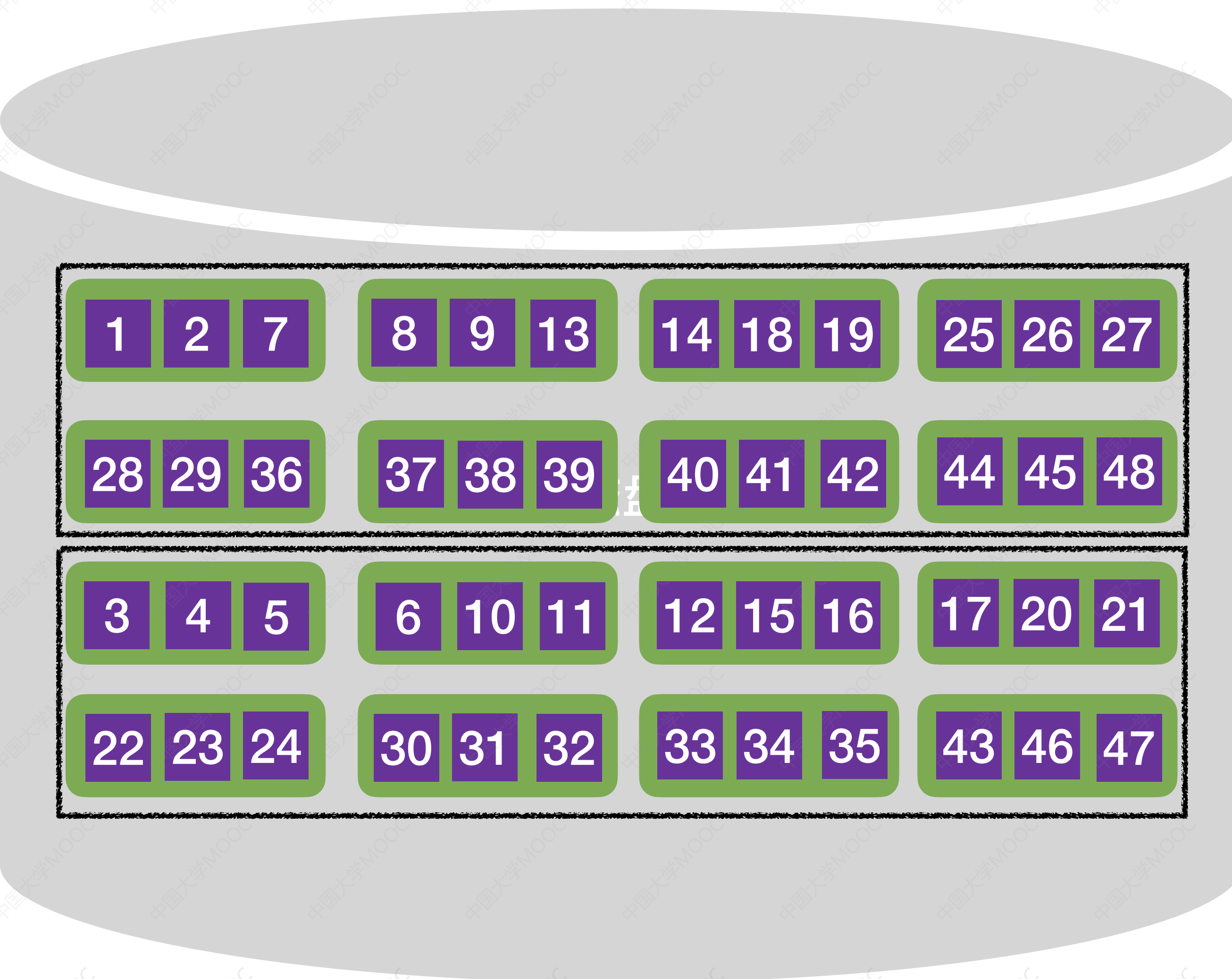
将两个有序归并段归并为一个

第二趟归并



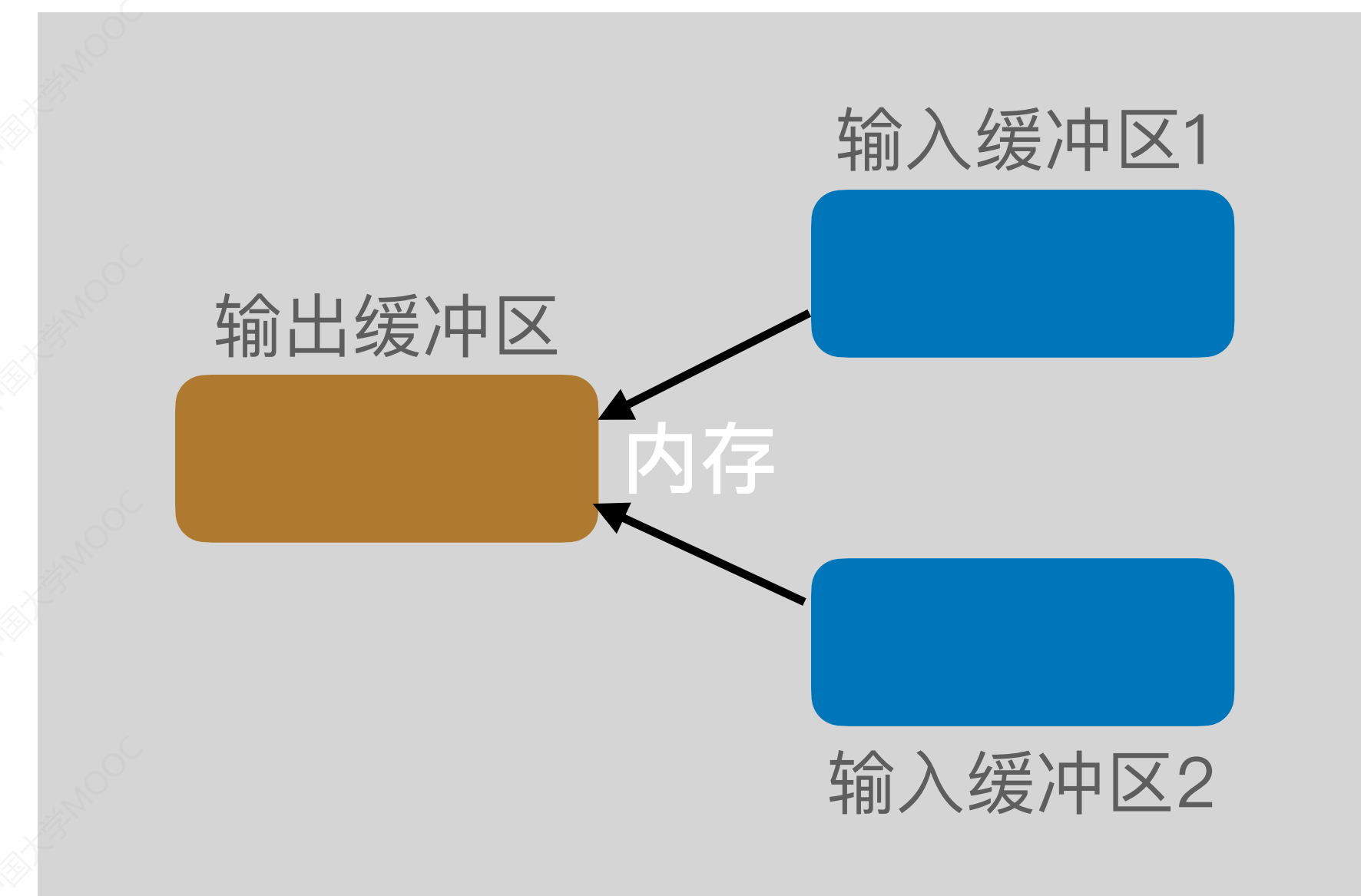
第三趟归并

把2个有序子序列
(归并段) 归并



读磁盘

写磁盘



将两个有序归并段归并为一个

第三趟归并



经过3趟归并，
整体有序

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48

读磁盘

写磁盘

输入缓冲区1

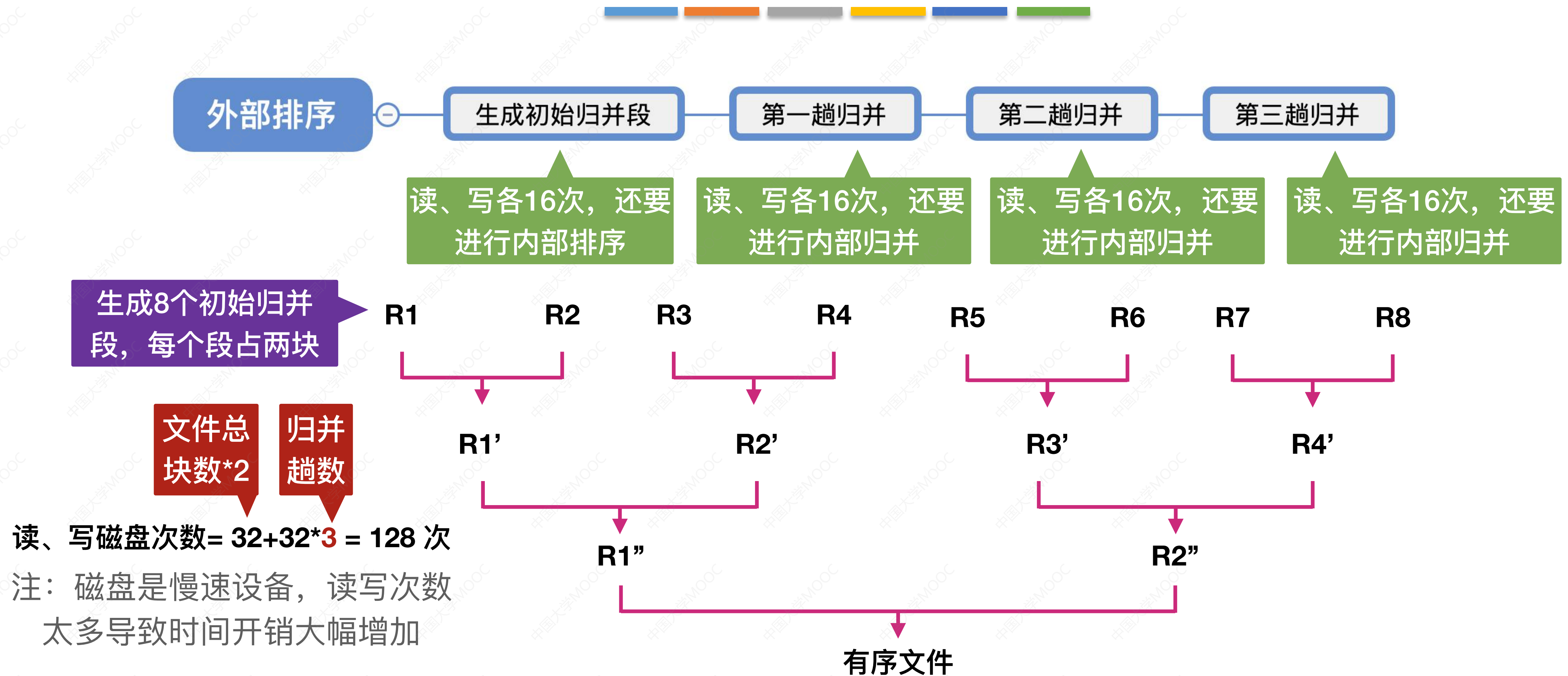
输出缓冲区

内存

输入缓冲区2

将两个有序归并段归并为一个

时间开销分析



外部排序时间开销=读写外存的时间+内部排序所需时间+内部归并所需时间

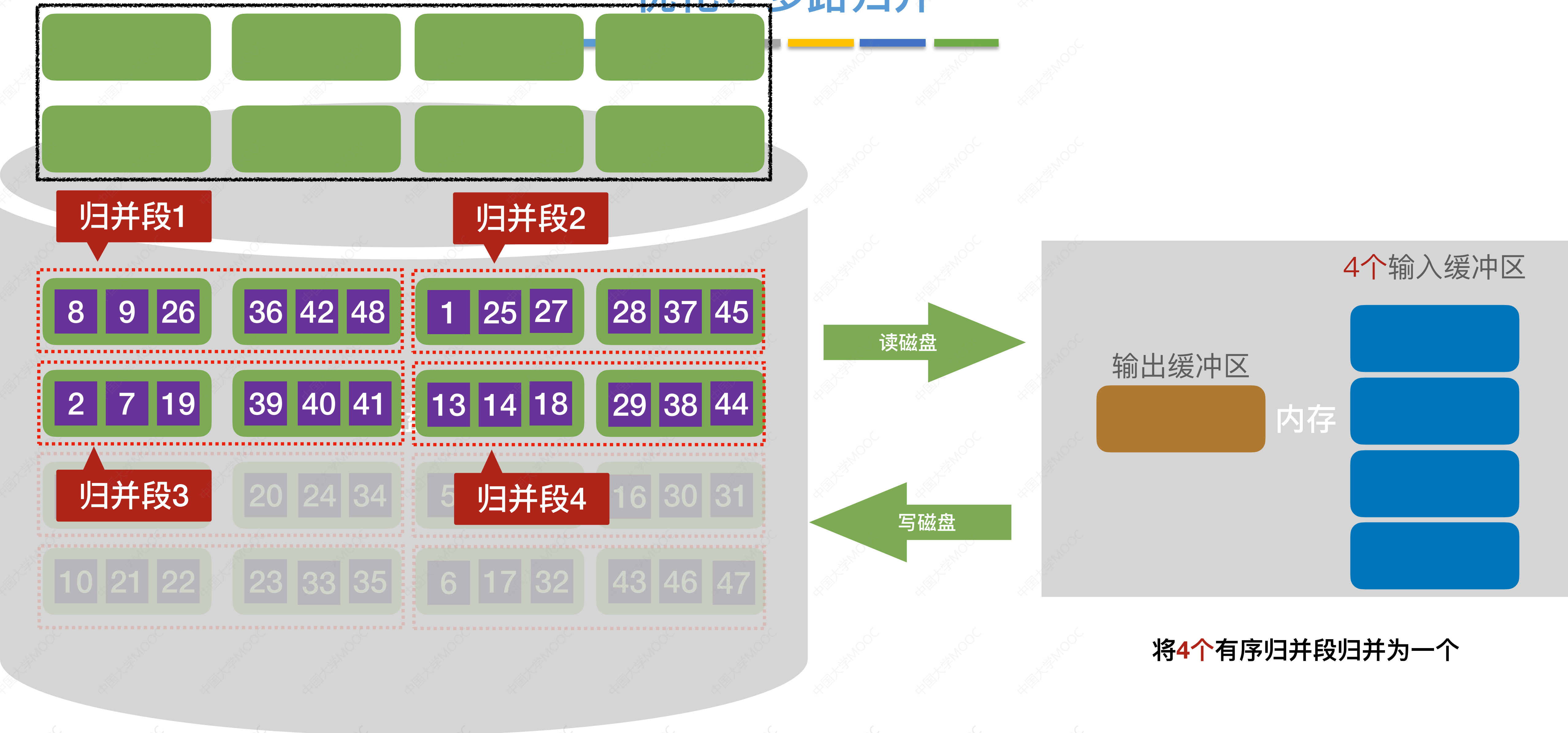
如何优化?



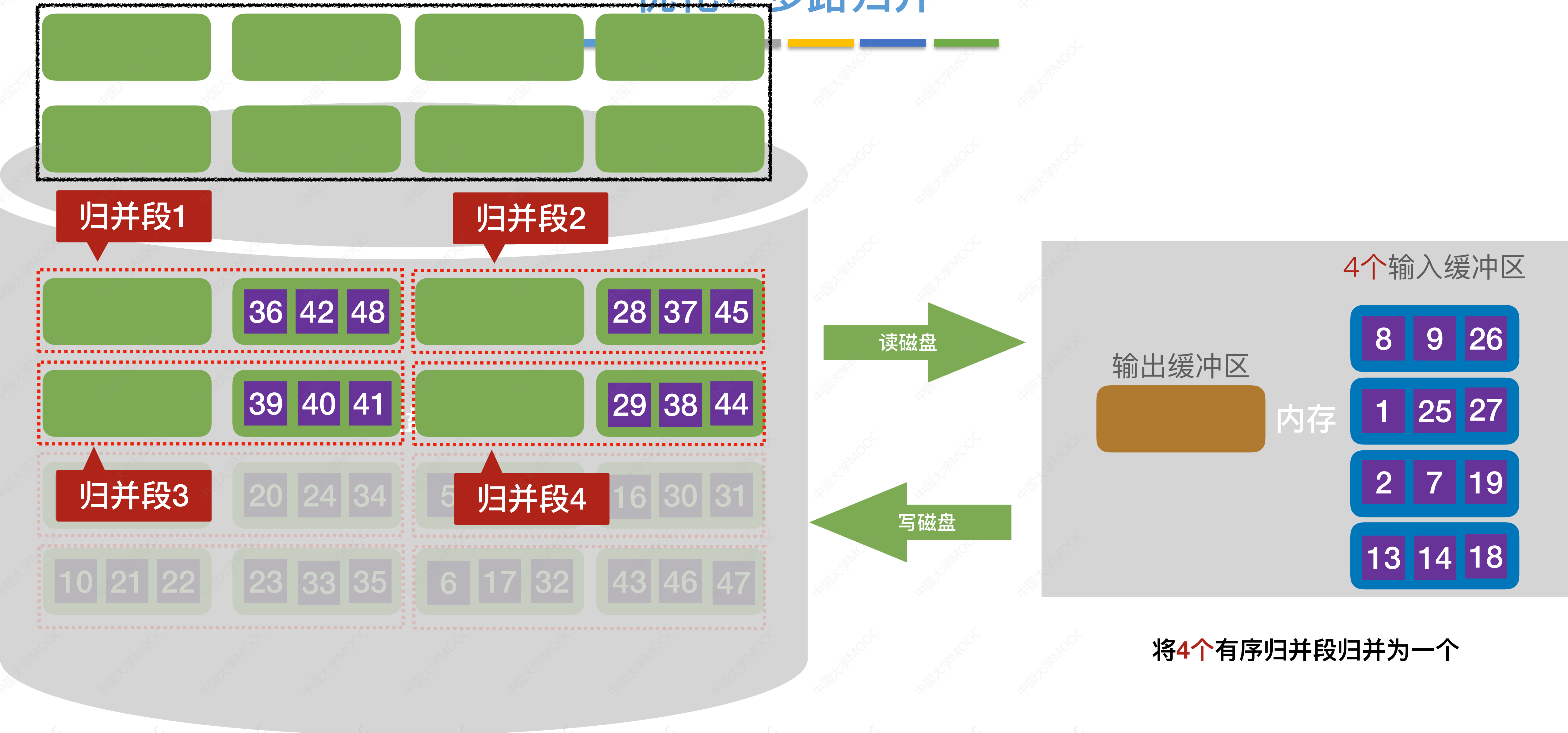
外部排序时间开销=读写外存的时间+内部排序所需时间+内部归并所需时间



优化：多路归并

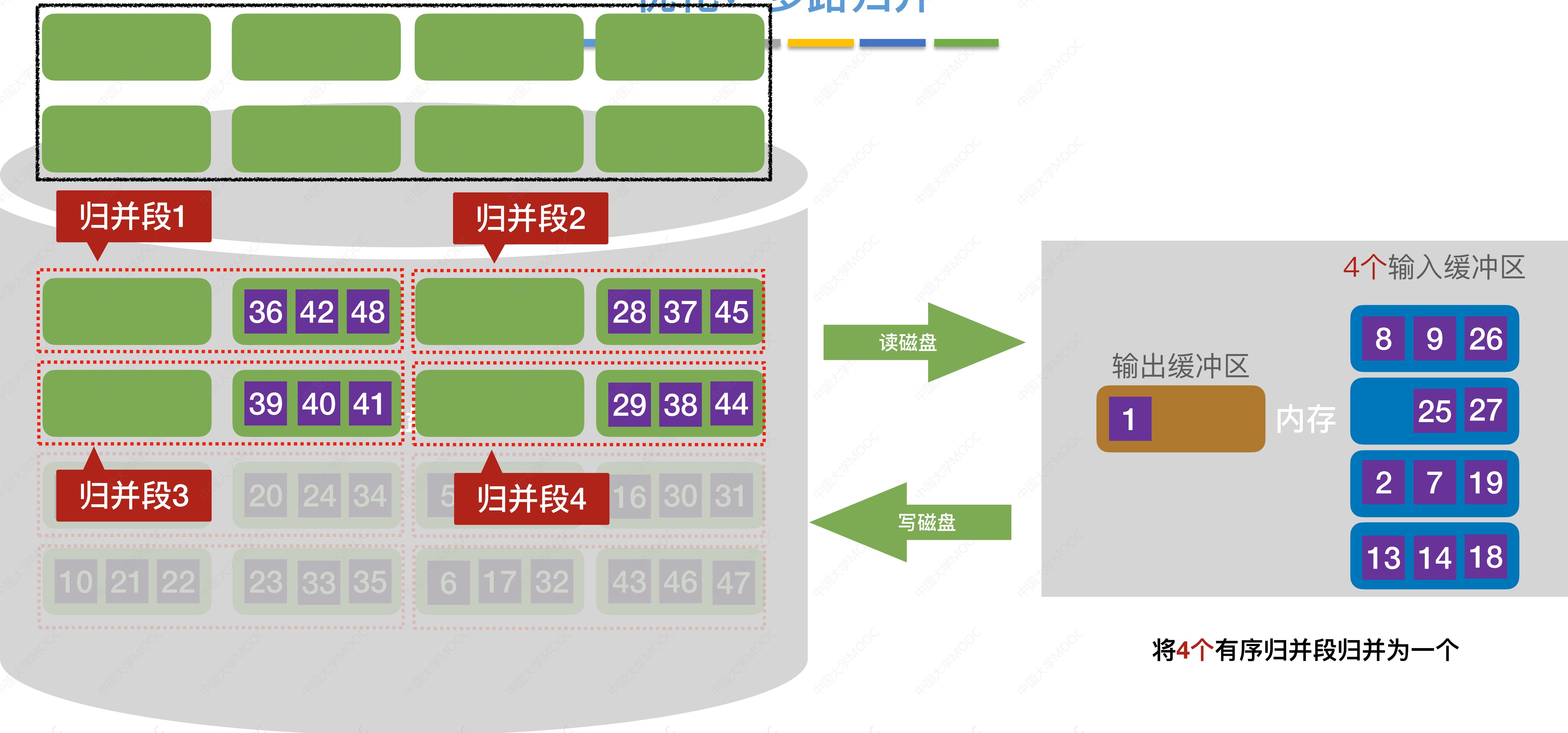


优化：多路归并



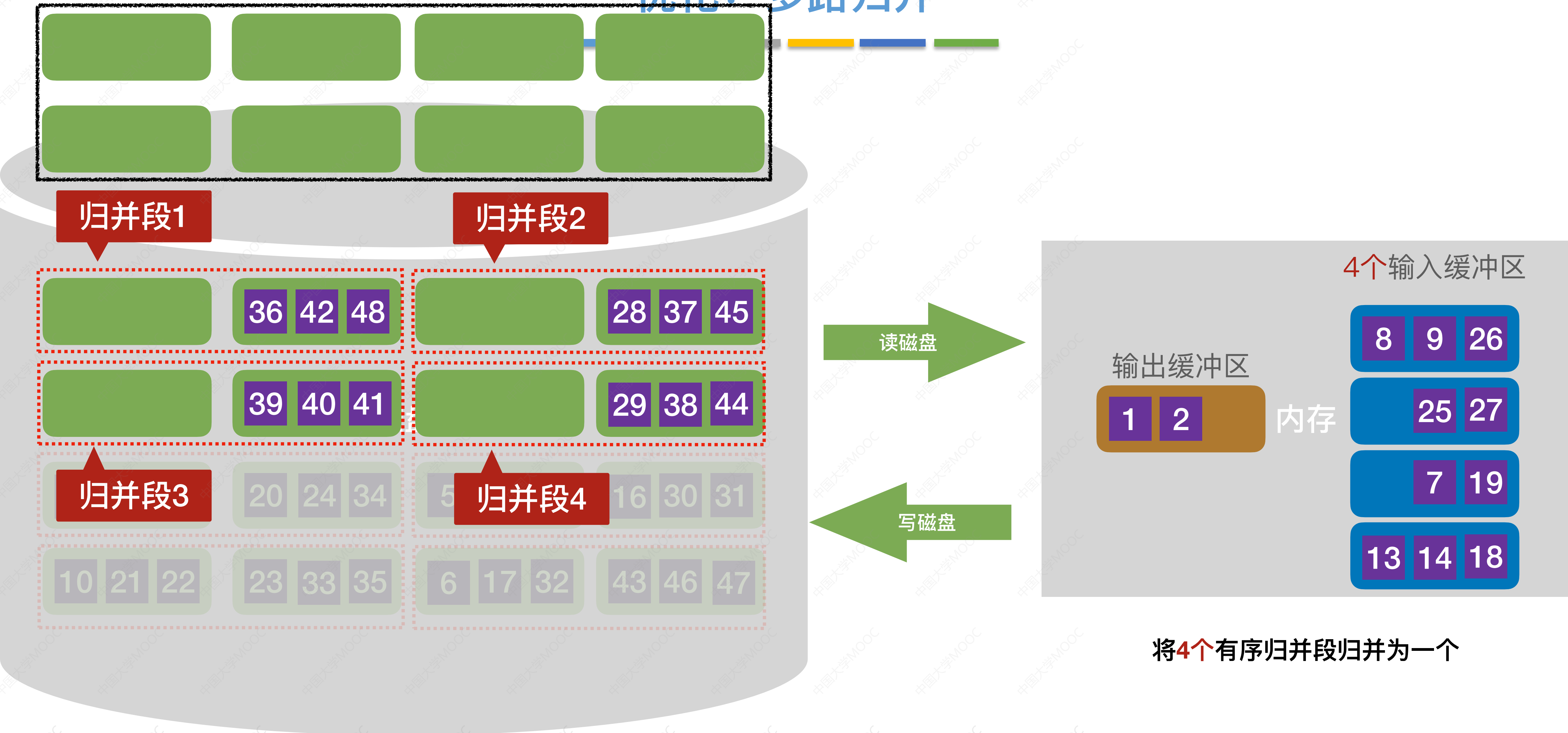
将4个有序归并段归并为一个

优化：多路归并

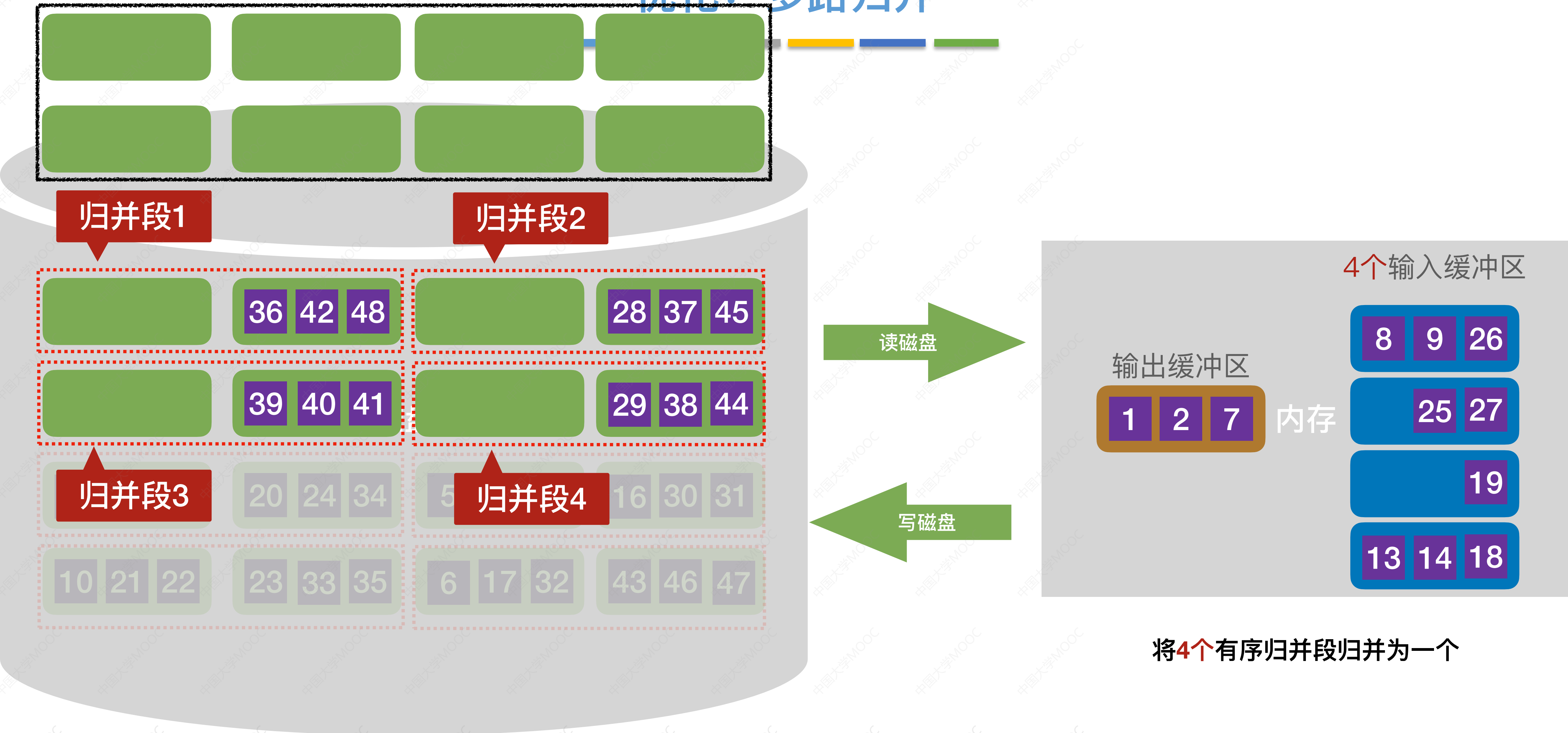


将4个有序归并段归并为一个

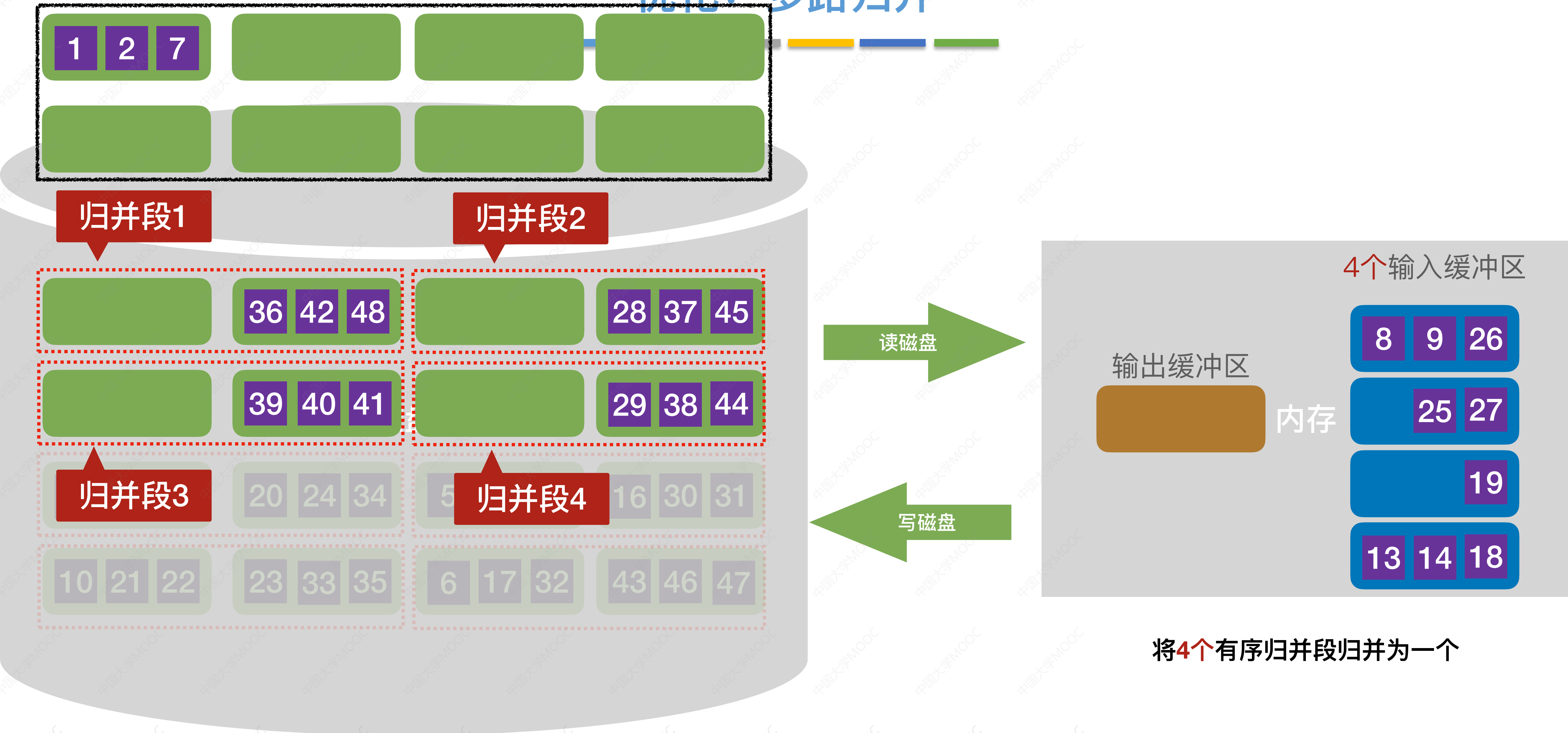
优化：多路归并



优化：多路归并

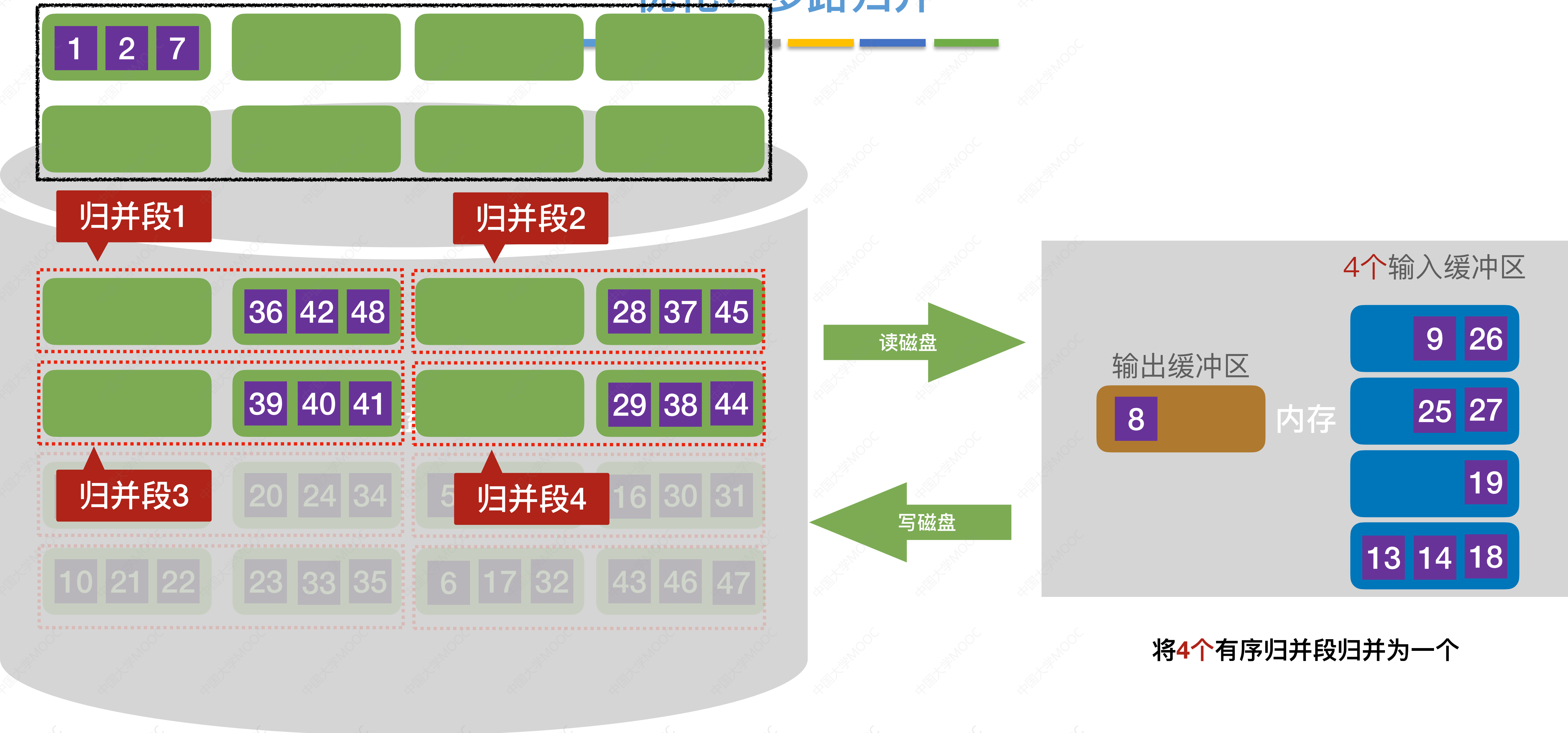


优化：多路归并

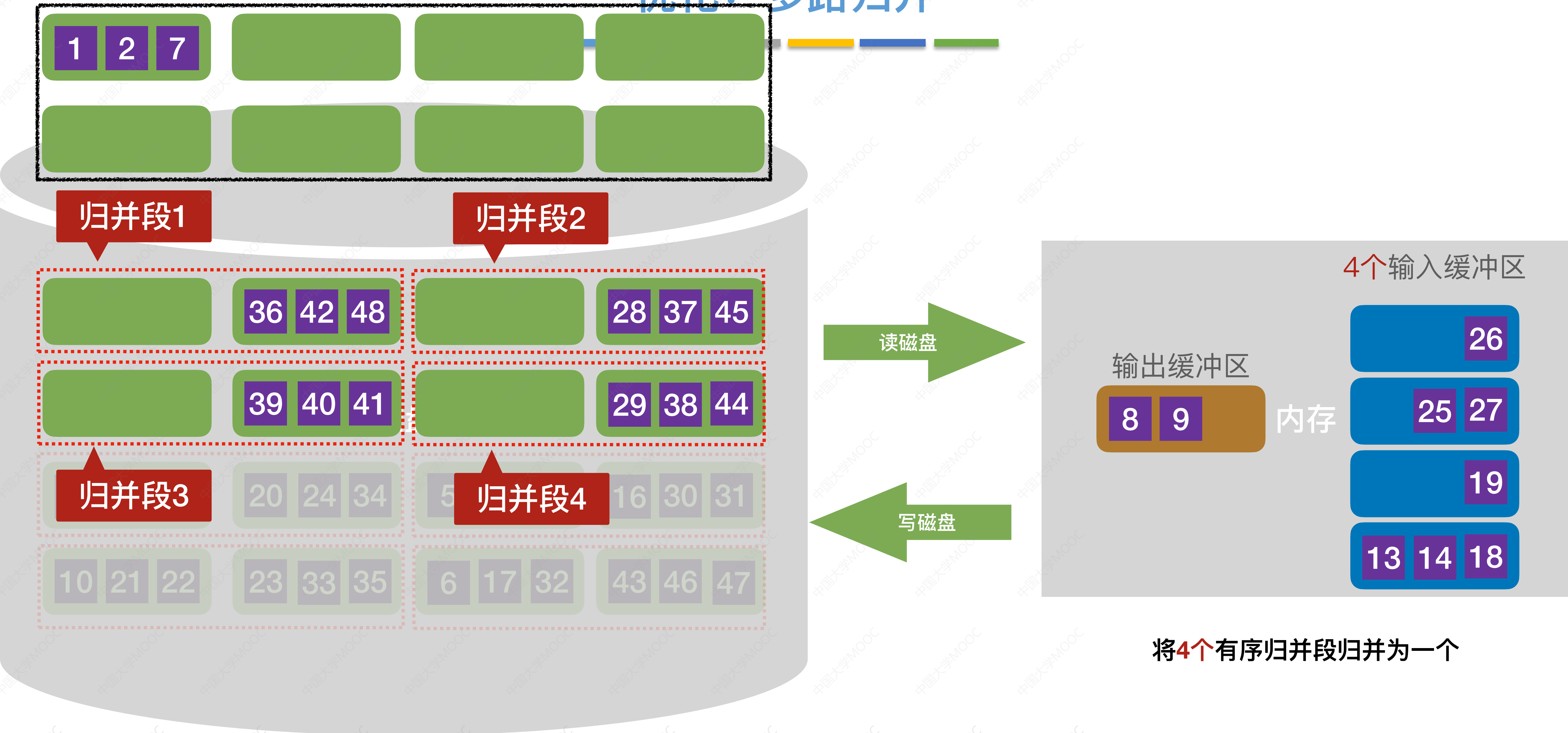


将4个有序归并段归并为一个

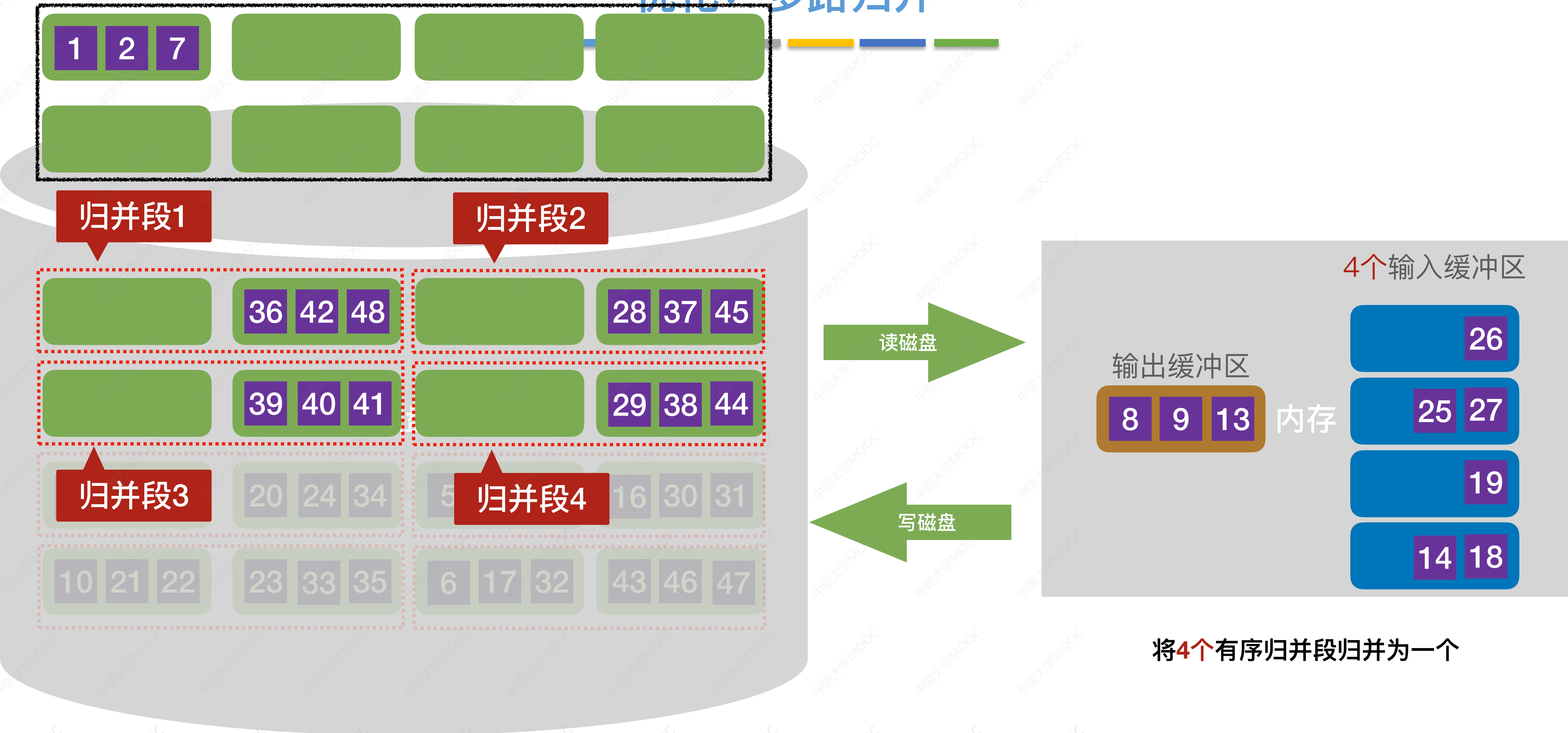
优化：多路归并



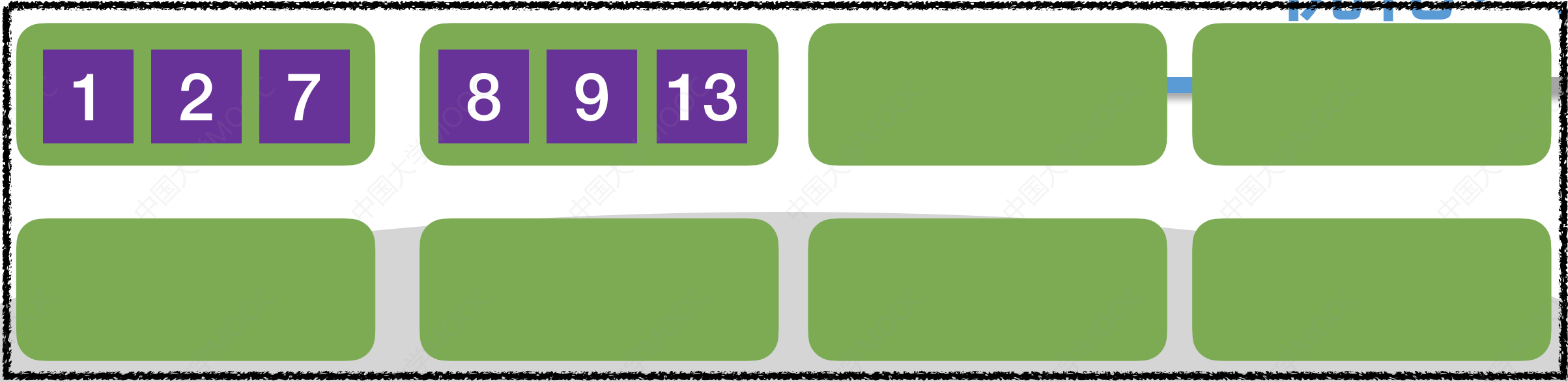
优化：多路归并



优化：多路归并



优化：多路归并



归并段1

归并段2



读磁盘

写磁盘

4个输入缓冲区

输出缓冲区

内存

26

25 27

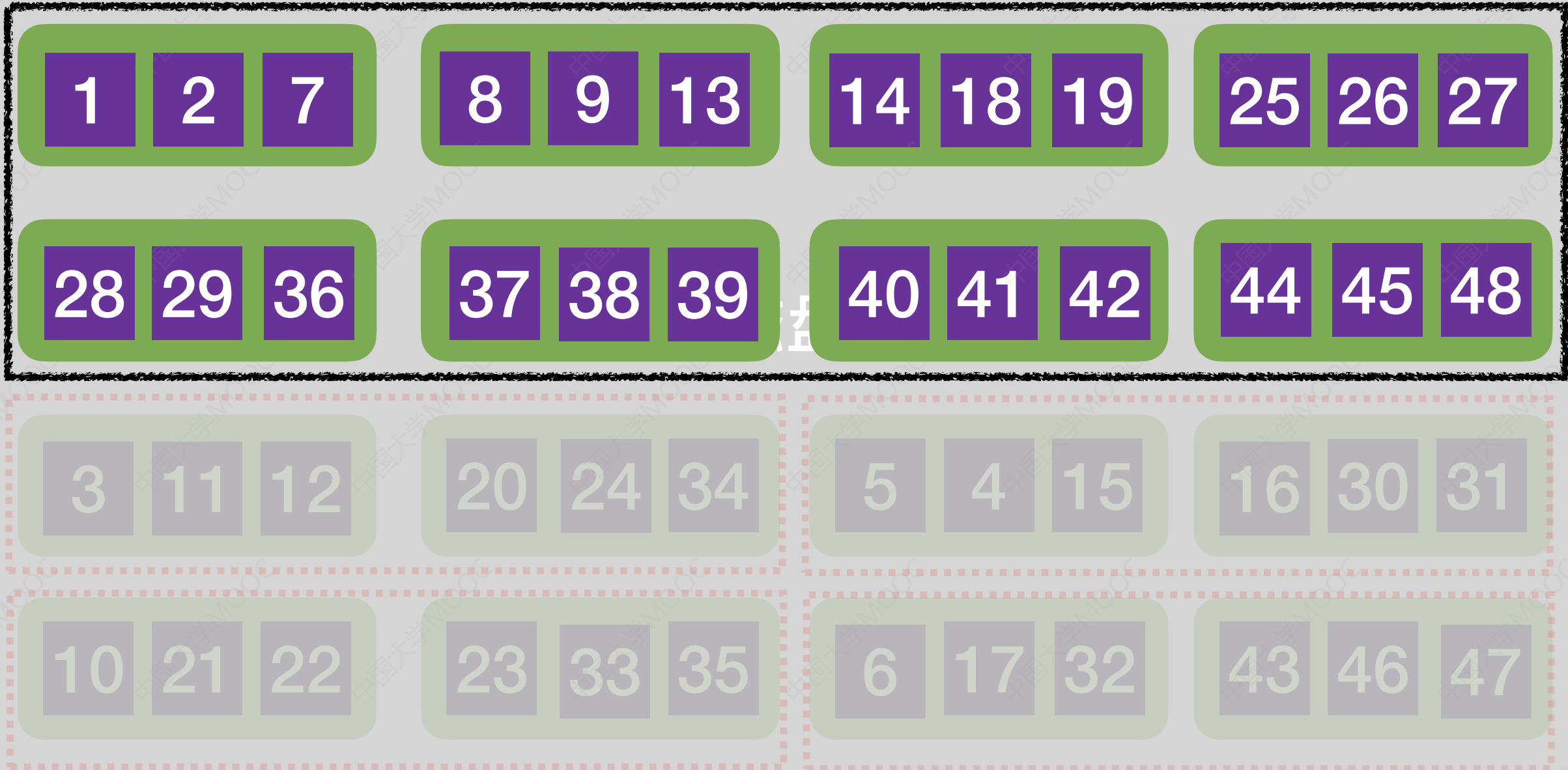
19

14 18

将4个有序归并段归并为一个

优化：多路归并

归并为一个更长的有序序列



读磁盘

写磁盘

4个输入缓冲区

输出缓冲区

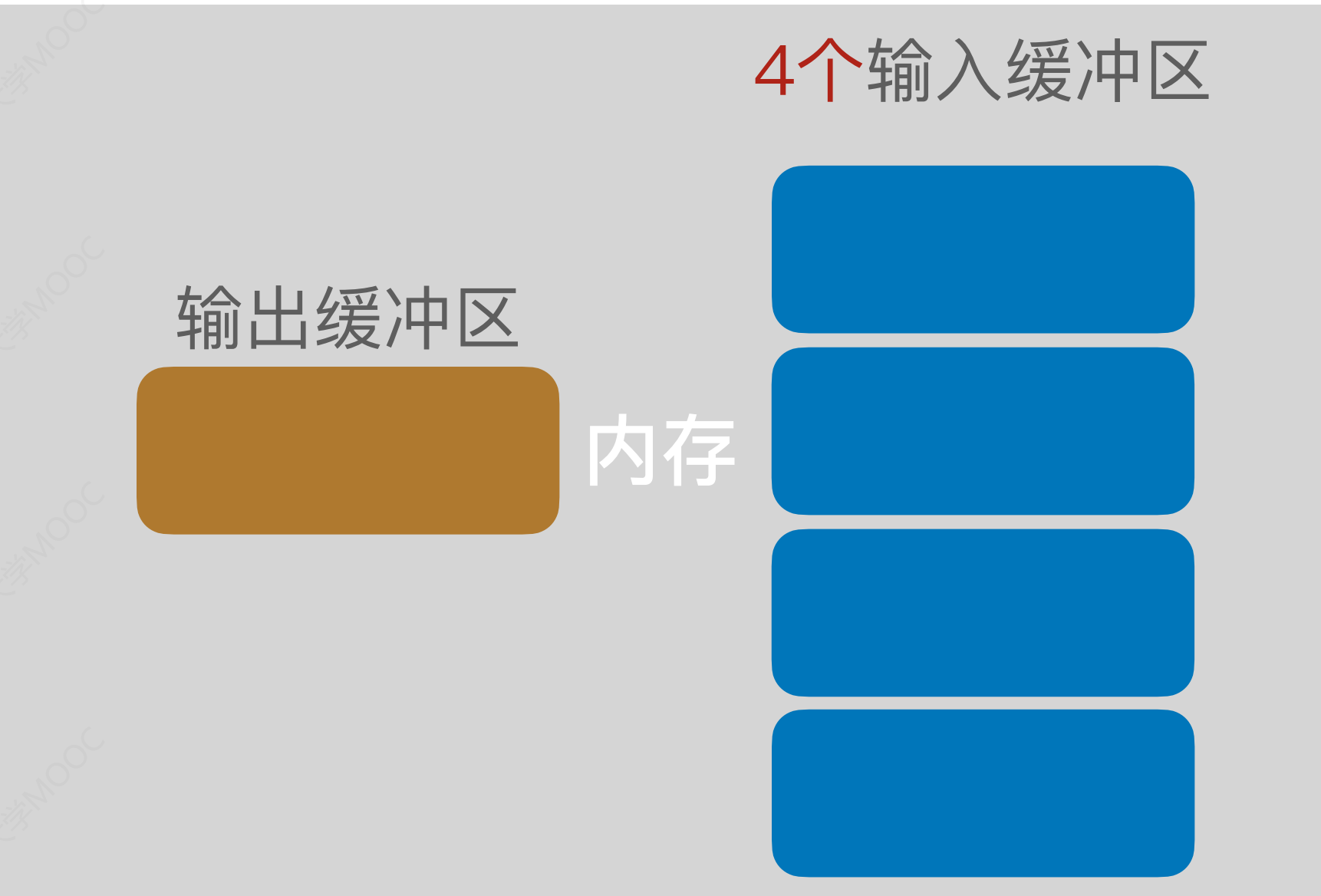
内存

将4个有序归并段归并为一个

优化：多路归并



1	2	7	8	9	13	14	18	19	25	26	27
28	29	36	37	38	39	40	41	42	44	45	48
3	4	5	6	10	11	12	15	16	17	20	21
22	23	24	30	31	32	33	34	35	43	46	47



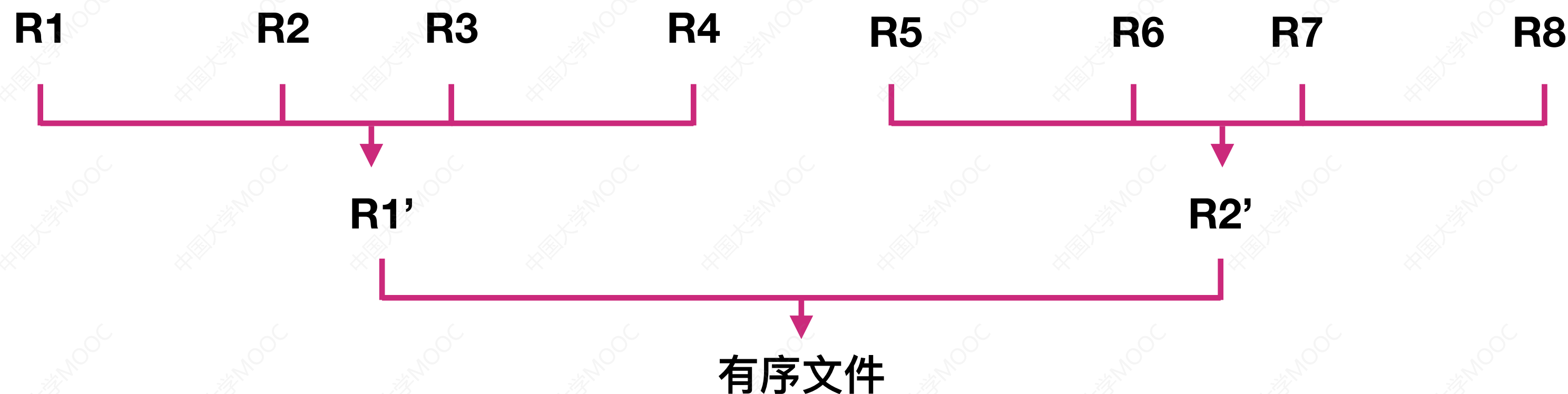
将4个有序归并段归并为一个

第一趟归并之后

优化：多路归并



外部排序时间开销=读写外存的时间+内部排序所需时间+内部归并所需时间



采用4路归并，只需进行两趟归并即可

读、写磁盘次数= $32+32*2 = 96$ 次

重要结论：采用多路归并可以减少归并趟数，从而减少磁盘I/O(读写)次数

对 r 个初始归并段，做 k 路归并，则归并树可用 k 叉树表示

若树高为 h ，则归并趟数 $= h-1 = \lceil \log_k r \rceil$

推导： k 叉树第 h 层最多有 k^{h-1} 个结点

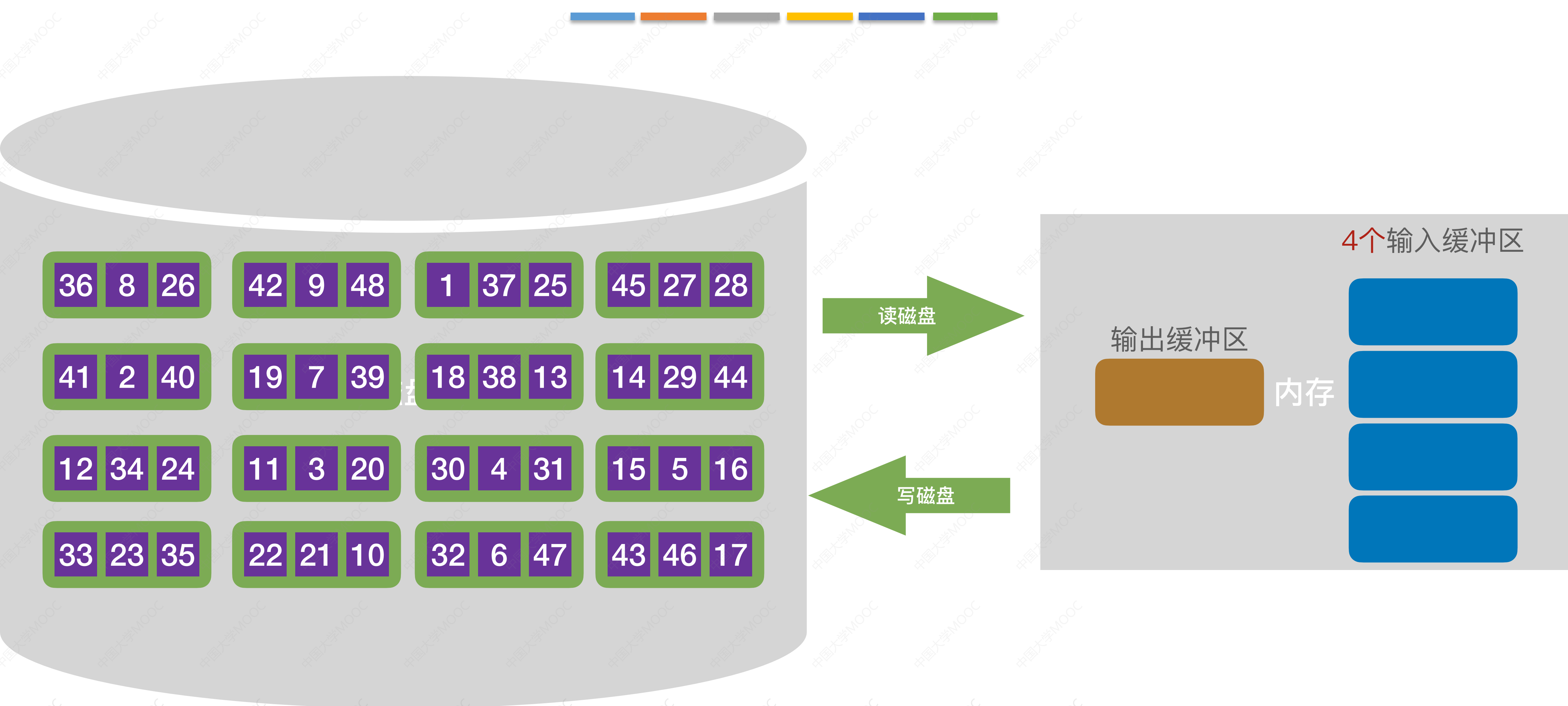
则 $r \leq k^{h-1}$ ， $(h-1)_{\text{最小}} = \lceil \log_k r \rceil$

k 越大， r 越小，归并趟数越少，读写磁盘次数越少

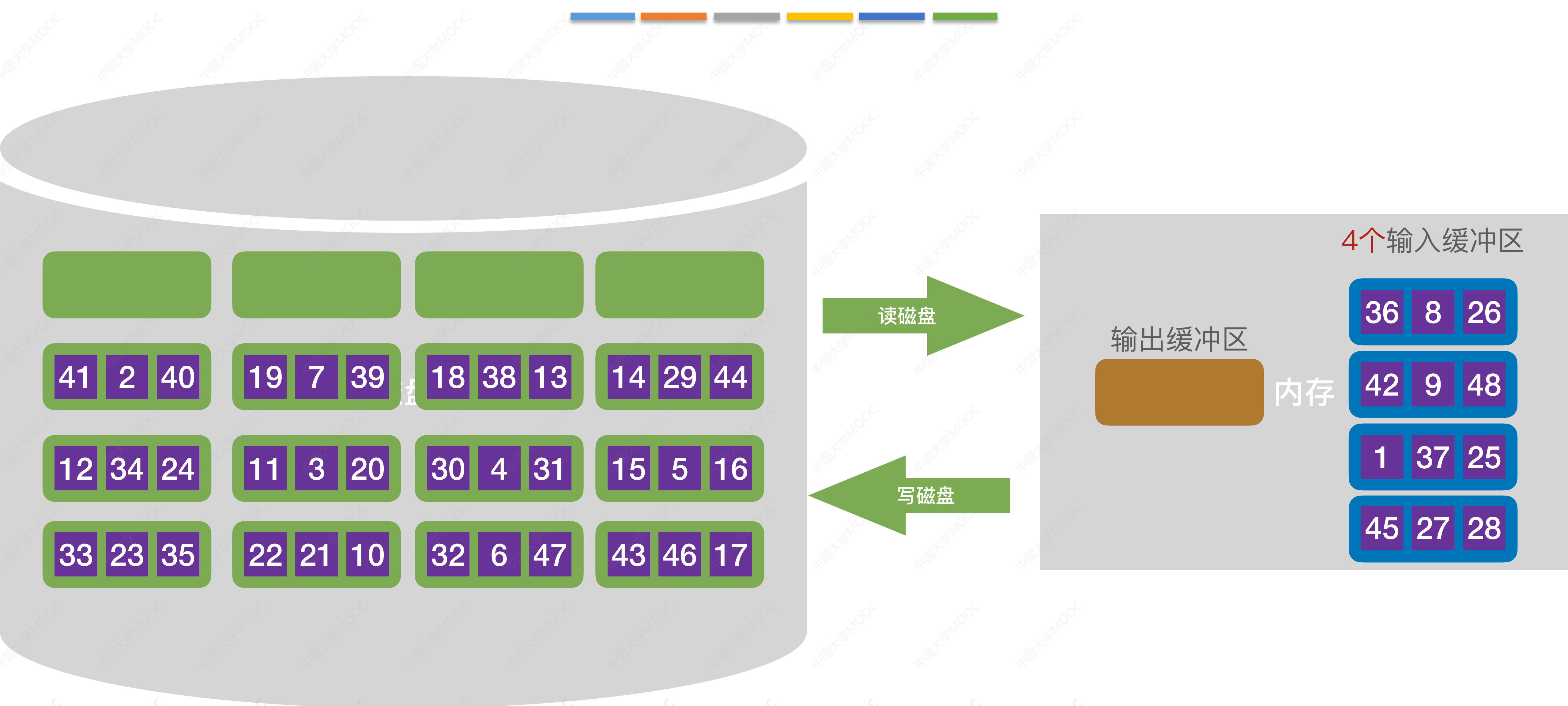
多路归并带来的负面影响：

- ① k 路归并时，需要开辟 k 个输入缓冲区，内存开销增加。
- ② 每挑选一个关键字需要对比关键字 $(k-1)$ 次，内部归并所需时间增加

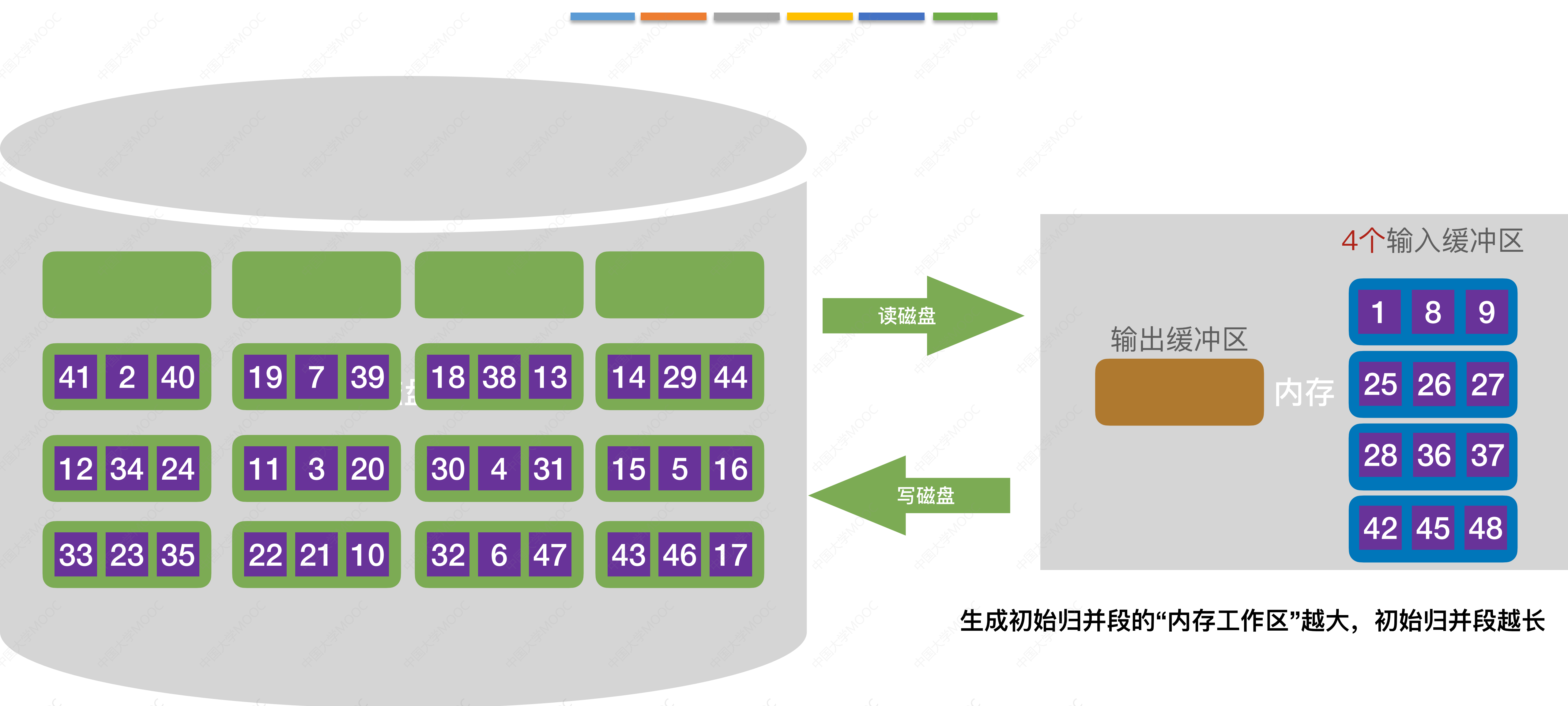
优化：减少初始归并段数量



优化：减少初始归并段数量

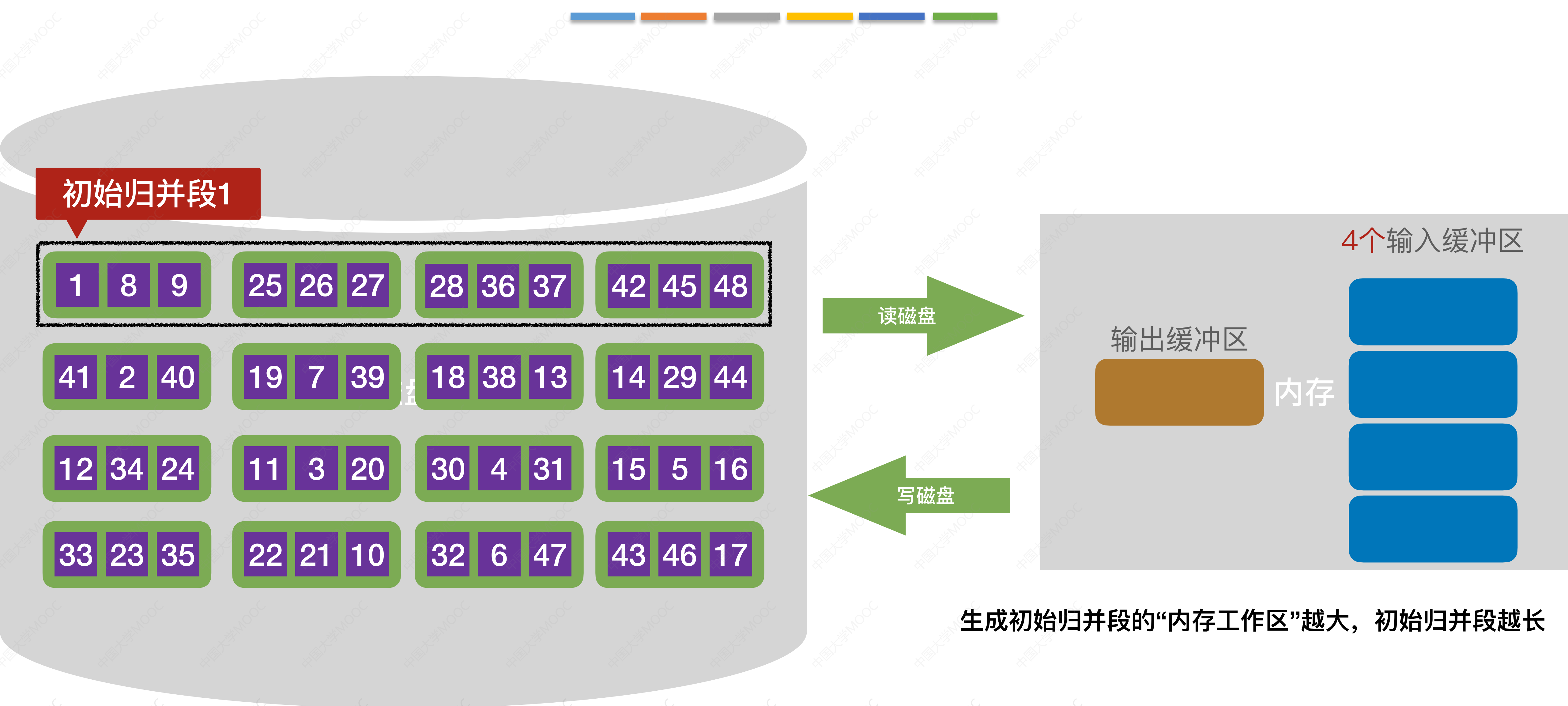


优化：减少初始归并段数量



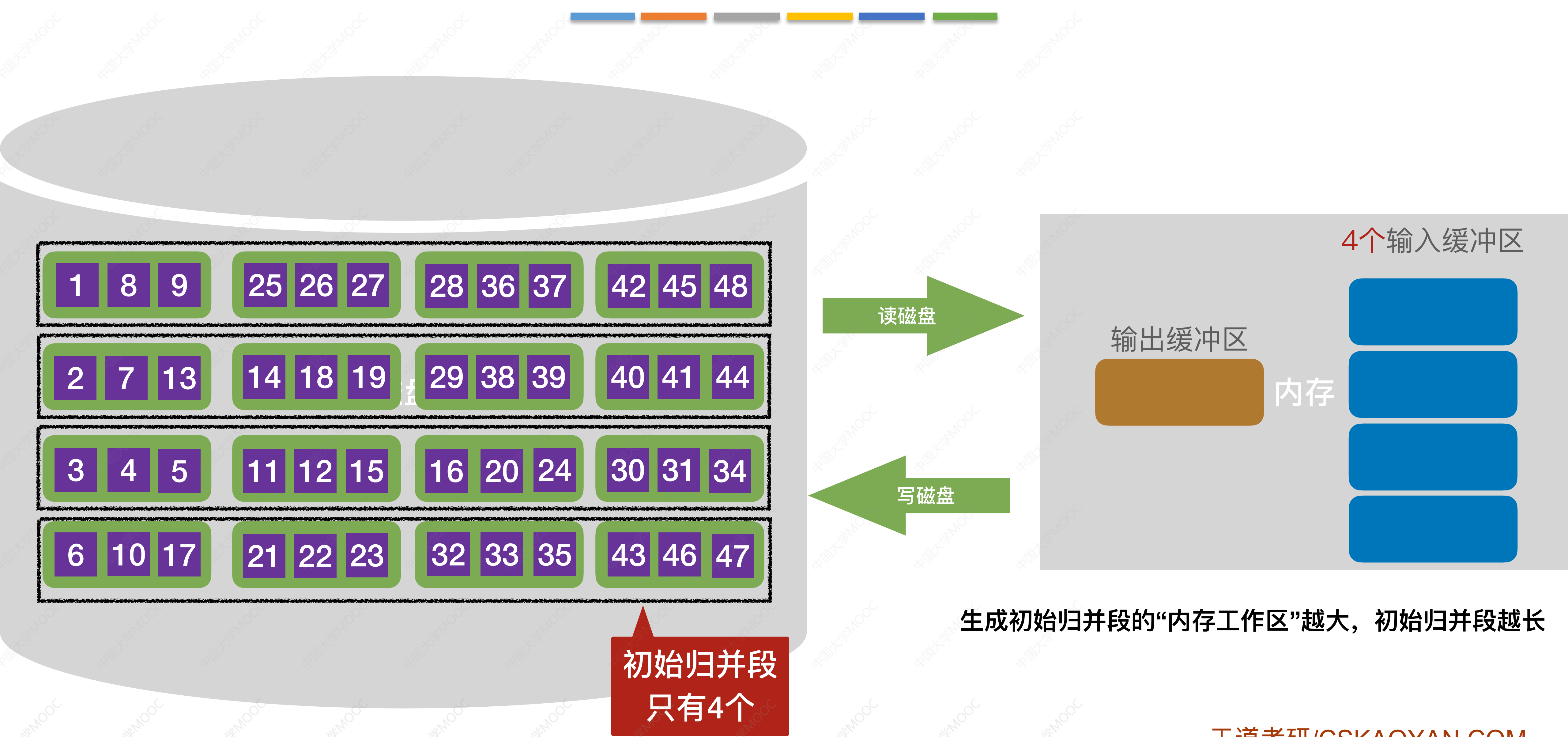
生成初始归并段的“内存工作区”越大，初始归并段越长

优化：减少初始归并段数量



生成初始归并段的“内存工作区”越大，初始归并段越长

优化：减少初始归并段数量



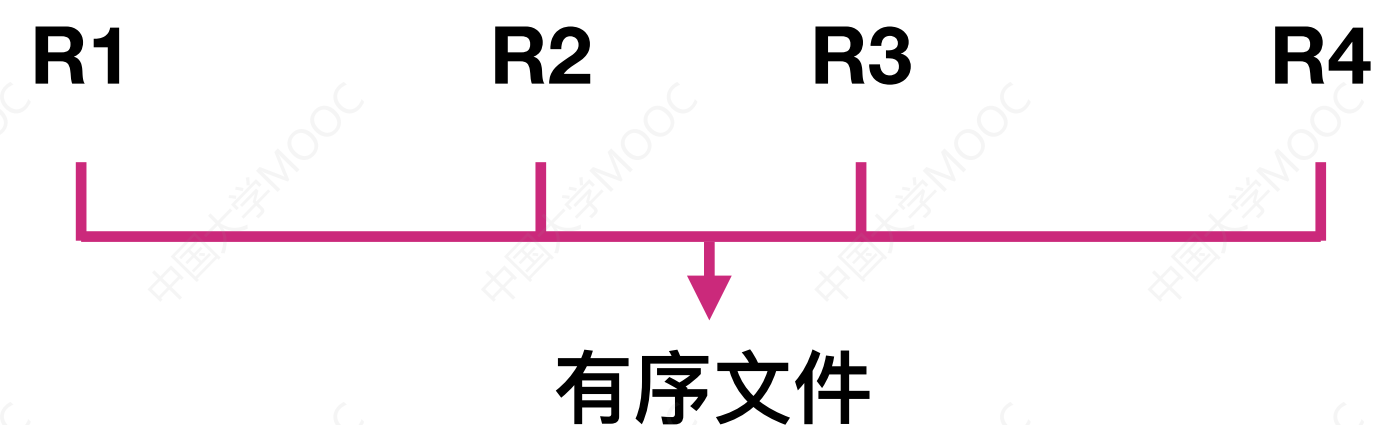
优化：减少初始归并段数量



对 r 个初始归并段，做 k 路归并，则归并树可用 k 叉树表示

若树高为 h ，则归并趟数 $= h-1 = \lceil \log_k r \rceil$

k 越大， r 越小，归并趟数越少，读写磁盘次数越少



结论：若能增加初始归并段的长度，则可减少初始归并段数量 r

知识回顾与重要考点

若要进行k路归并排序，则需要在内存中分配k个输入缓冲区和1个输出缓冲区

外部排序

步骤

- ①生成 r 个初始归并段（对 L 个记录进行内部排序，组成一个有序的初始归并段）
- ②进行 S 趟k路归并， $S = \lceil \log_k r \rceil$

如何进行k路归并

- 把k个归并段的块读入k个输入缓冲区
- 用“归并排序”的方法从k个归并段中选出几个最小记录暂存到输出缓冲区中
- 当输出缓冲区满时，写出外存

外部排序时间开销

读写外存的时间 + 内部排序所需时间 + 内部归并所需时间

优化

- 增加归并路数 k，进行多路平衡归并
 - 代价 1：需要增加相应的输入缓冲区
 - 代价 2：每次从k个归并段中选一个最小元素需要 (k-1) 次关键字对比
- 减少初始归并段数量 r

可用“败者树”减少关键字对比次数

可用“置换-选择排序”进一步减少初始归并段数量

注：按照本节介绍的方法生成的初始归并段，若共 N 个记录，内存工作区可以容纳 L 个记录，则初始归并段数量 $r = \lceil N/L \rceil$

纠正：什么是多路平衡归并？



误：对 r 个初始归并段，做 k 路平衡归并，归并树可用严格 k 叉树（即只有度为 k 与度为 0 的结点的 k 叉树）来表示。



知道错哪了不？

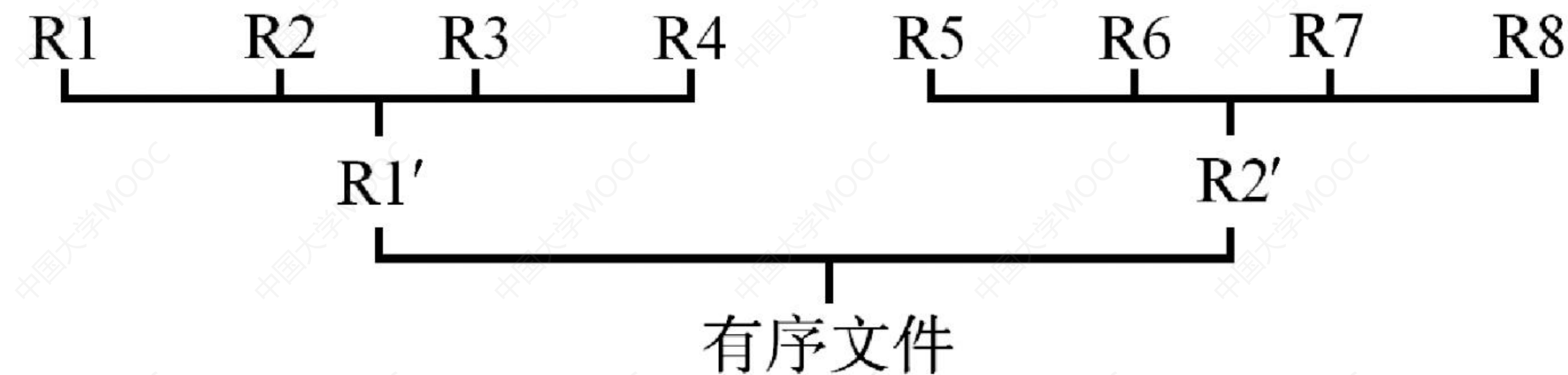


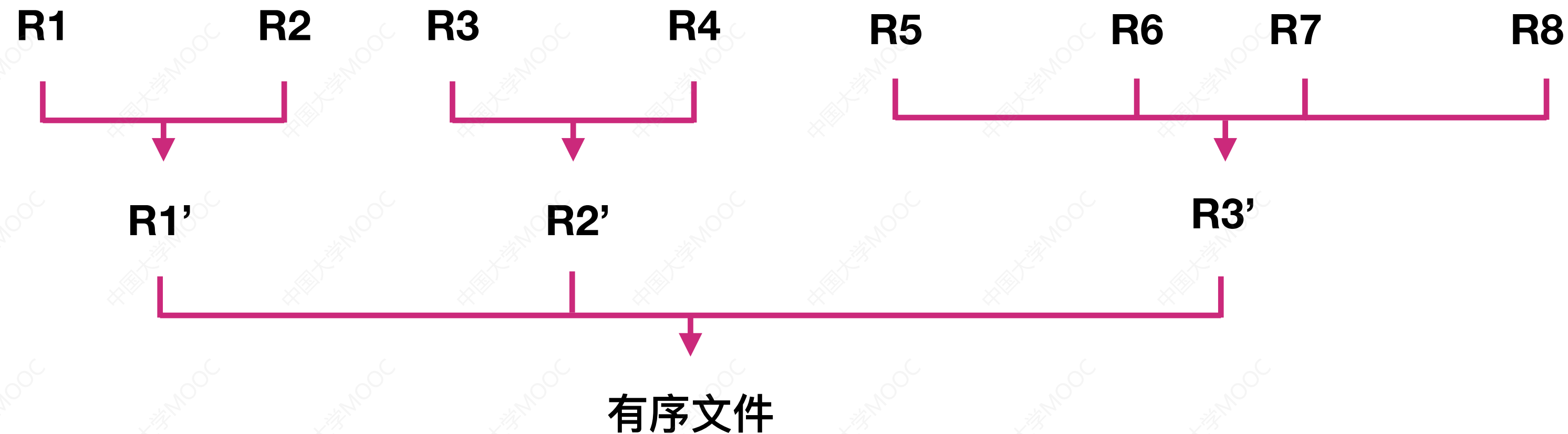
图 7.8 4 路平衡归并的排序过程

k路平衡归并：①最多只能有 k 个段归并为一个；

②每一趟归并中，若有 m 个归并段参与归并，则经过这一趟处理得到 $\lceil m/k \rceil$ 个新的归并段

纠正：什么是多路平衡归并？

8个归并段经过一趟处理后得到3个新的归并段



这个例子是不是4路归并排序？—— 是！

这个例子是不是4路平衡归并排序？—— 不是！

$$\lceil 8/4 \rceil = 2 \neq 3$$

k路平衡归并：①最多只能有k个段归并为一个；

②每一趟归并中，若有 m 个归并段参与归并，则经过这一趟处理得到 $\lceil m/k \rceil$ 个新的归并段

欢迎大家对本节视频进行评价~



学员评分：8.7.1+8.7.2...

扫一扫二维码打开或分享给好友



— 腾讯文档 —

可多人实时在线编辑，权限安全可控



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研