ZRAM那点事

ZRAM和ZSMALLOC简介及3个问题的改进和1个提高

朱辉
zhuhui@xiaomi.com
teawater.github.io

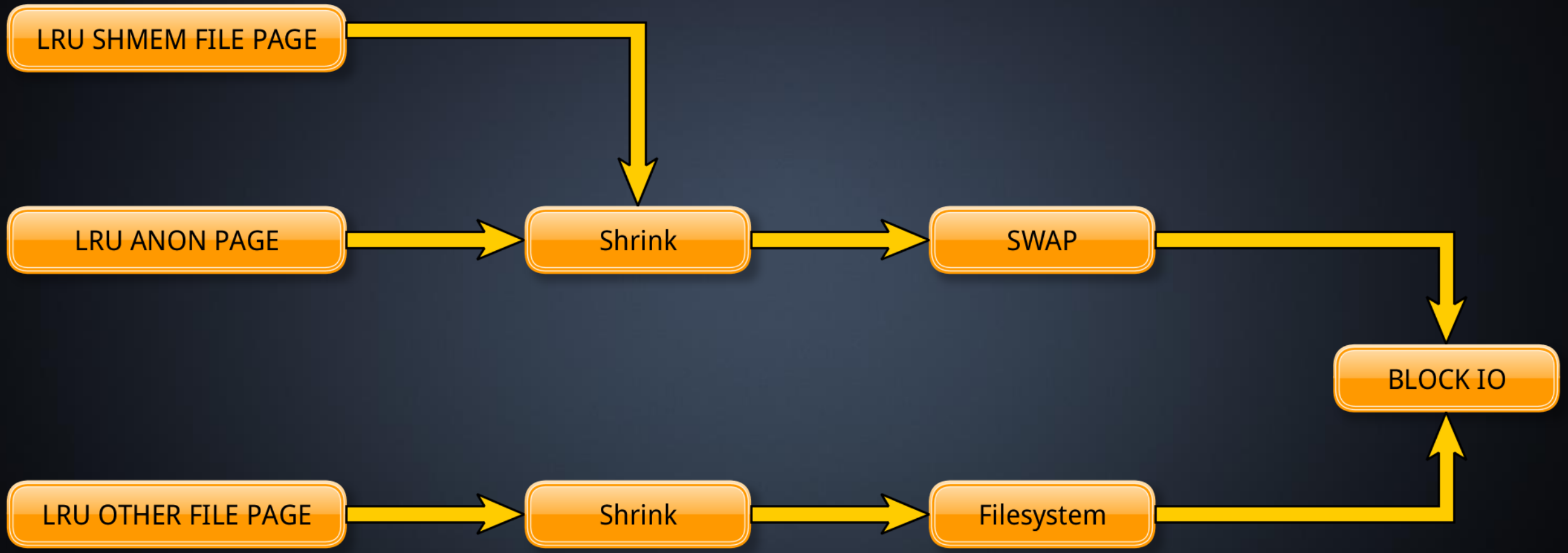
什么是ZRAM?（见下图）

- 虚拟磁盘。
- 将写入的页面压缩并分配内存存储在系统中。
- 主要用来作为SWAP设备。
- 常用在用闪存作为存储空间的设备上。

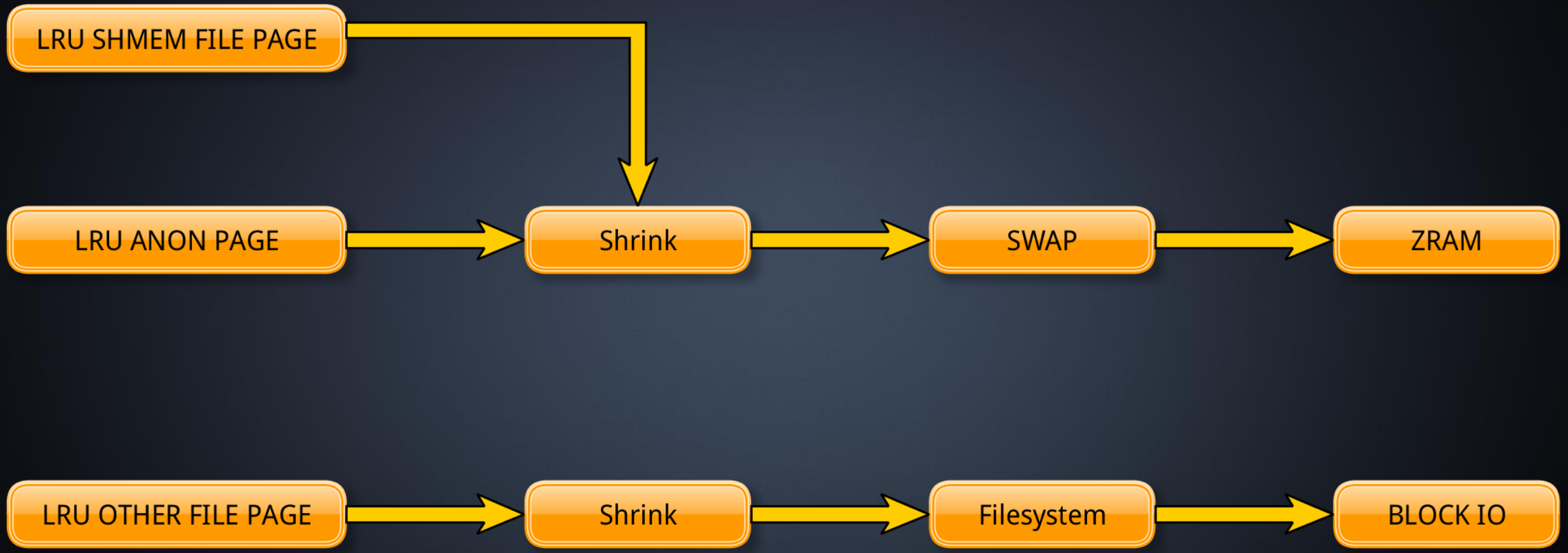
为什么用ZRAM?（见下图）

- 可以不再需要开其他的SWAP 设备。（区别于ZSWAP）
  节省了FLASH的寿命。
  节省了硬盘空间。
  节省了BLOCK IO。
- Android下可用来提高进程保活度。

LRU Shrinker



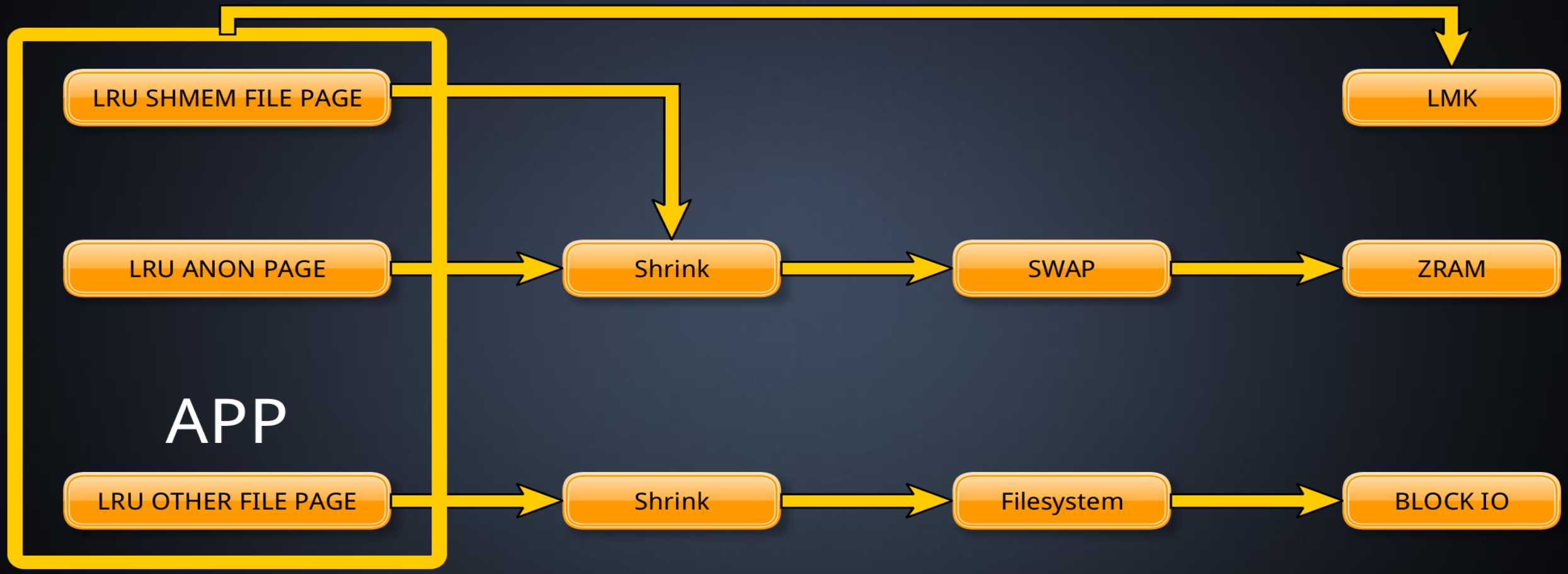
ZRAM


LRU SHMEM FILE PAGE
LRU ANON PAGE
Shrink
SWAP
ZRAM
LRU OTHER FILE PAGE
Shrink
Filesystem
BLOCK IO

Android 无SWAP

Android 无SWAP

Android有SWAP



什么是ZSMALLOC？（见下图）

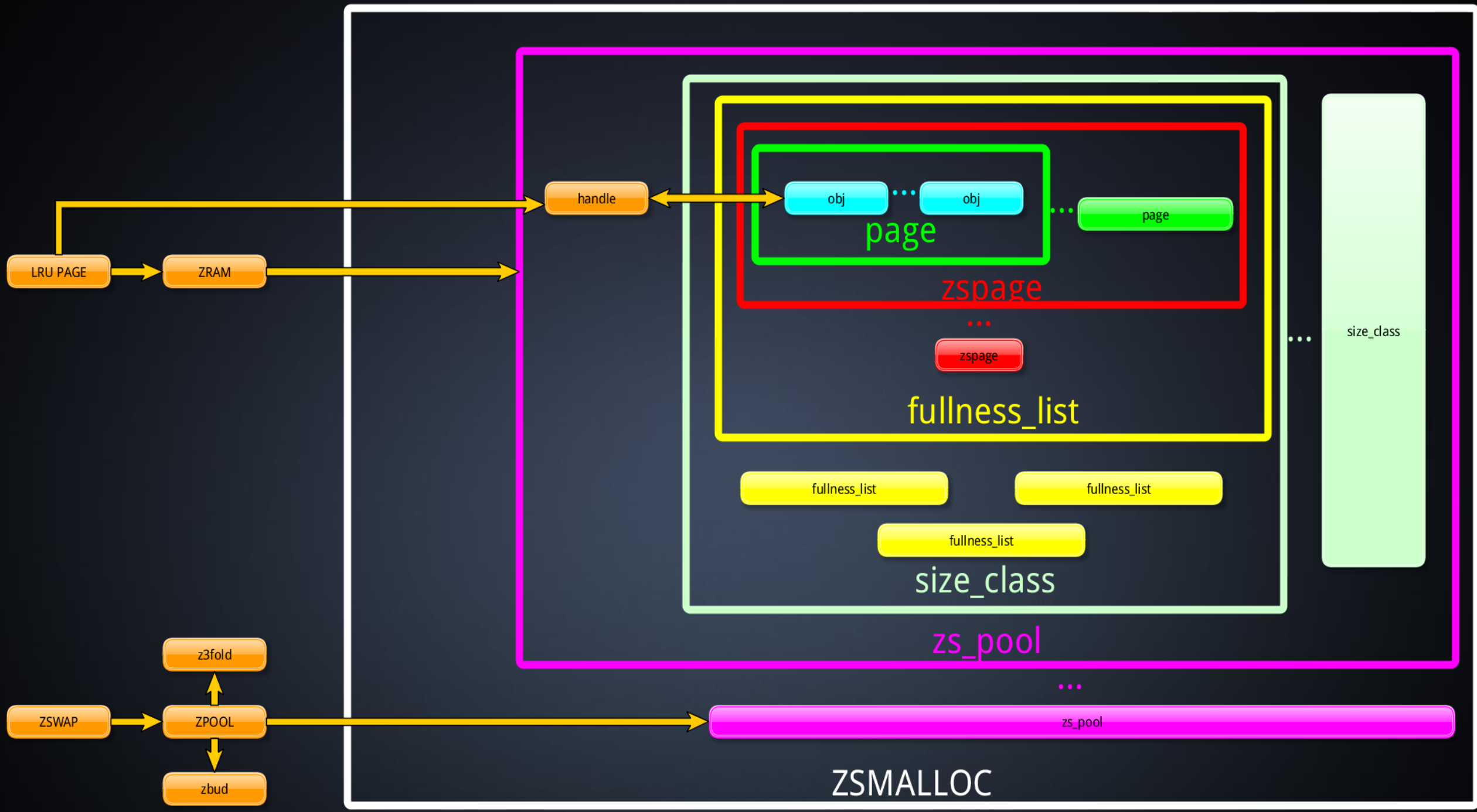
- 针对内存压缩场景设计的内存分配器。
  类似其的还有ZBUD和Z3FOLD。

- ZRAM直接使用ZSMALLOC。
  ZSWAP通过ZPOOL框架使用其。

ZSMALLOC内部结构（见下图）

- 每个对ZSMALLOC的使用者会申请一个zs_pool。
- 每个zs_pool根据存储数据的大小分出若干size_class。
- 每个size_class中根据元素占有状态分出4个fullness_list。
- 每个fullness_list保存若干zspage，每个zspage由一个到几个几个不连续page组成。
- 每个zspage(由最多4个非连续page组成)保存若干储存对象obj，ZRAM中每个page最终会被保存为一个obj。
- ZRAM通过handle以映射的形式访问某个obj，因为是映射式的访问，highzone page也是可以使用的。（今年还有修改，后面会提到）

- 实际观测ZSMALLOC内部结构可在配置内核中打开CONFIG_ZSMALLOC_STAT=y。访问/sys/kernel/debug/zsmalloc/zram0/classes就可查看ZSMALLOC中的内部信息。


handle
obj
obj
page
page
zspage
zspage
fullness_list
fullness_list
fullness_list
fullness_list
size_class
size_class
zs_pool
zs_pool
LRU PAGE
ZRAM
ZSWAP
ZPOOL
z3fold
zbud
ZSMALLOC

class	size	almost_full	almost_empty	obj_allocated	obj_used	pages_used	pages_per_zspage
0	32	0	0	0	0	0	1
1	48	3	1	2560	2434	30	3
2	64	2	1	1280	1221	20	1
3	80	1	1	663	637	13	1
4	96	0	1	640	600	15	3
5	112	0	1	365	326	10	2
6	128	0	1	352	342	11	1
7	144	0	1	425	385	15	3
8	160	1	1	357	327	14	2
9	176	0	1	372	317	16	4
10	192	1	0	448	434	21	3
11	208	0	1	390	360	20	2
12	224	1	0	292	281	16	4
13	240	1	1	272	264	16	1
14	256	0	1	224	212	14	1
15	272	0	1	225	219	15	1
16	288	0	1	224	214	16	1
17	304	0	1	240	210	18	3
18	320	0	1	204	173	16	4
19	336	0	1	168	165	14	1
20	352	0	1	207	191	18	2
21	368	0	1	275	269	25	1
22	384	0	1	192	172	18	3
23	400	1	0	140	139	14	1
24	416	1	0	156	153	16	4
25	432	0	1	168	146	18	3
26	448	0	1	153	145	17	1
27	464	0	1	175	143	20	4
28	480	0	1	187	172	22	2
29	496	1	0	132	127	16	4
30	512	0	1	152	146	19	1
31	528	0	1	186	170	24	4
32	544	1	1	150	142	20	2
33	560	0	2	174	153	24	4
34	576	1	0	168	167	24	1
35	592	0	2	162	140	24	4
36	608	0	1	160	153	24	3
37	624	0	1	130	122	20	2
38	640	0	1	152	136	24	3
40	672	1	0	318	317	53	1
42	704	0	1	322	310	56	4
43	720	0	1	170	158	30	3
44	736	0	1	165	162	30	2
46	768	0	1	288	276	54	3
49	816	0	0	425	425	85	1
51	848	1	0	323	320	68	4
52	864	2	0	154	151	33	3
54	896	0	1	315	311	70	2
57	944	0	1	455	450	105	3
58	960	0	0	136	136	32	4
62	1024	0	1	616	614	154	1
66	1088	0	1	720	714	192	4
67	1104	0	1	187	182	51	3
71	1168	1	1	805	798	230	2
74	1216	1	0	720	718	216	3
76	1248	0	1	468	459	144	4
83	1360	0	1	1503	1502	501	1
91	1488	0	1	1551	1544	564	4
94	1536	0	1	616	614	231	3
100	1632	0	0	1345	1345	538	2
107	1744	0	1	1596	1594	684	3
111	1808	0	1	765	758	340	4
126	2048	0	0	2426	2426	1213	1
144	2336	0	1	1960	1958	1120	4
151	2448	1	0	545	544	327	3
168	2720	0	1	1389	1387	926	2
190	3072	0	1	1564	1562	1173	3
202	3264	0	1	45	43	36	4
254	4096	0	0	23306	23306	23306	1
Total		22	54	58118	57191	33259	

注意查找彩蛋

ZRAM使用中出现3大问题

Minchan Kim于2015年提出：

- 外部碎片
- 不能移动的页
- 内部碎片

这三个问题都需要在ZSMALLOC中进行修复，并有一定的相关性。
从去年开始层层修复到今年在Upstream上全部修复完成。

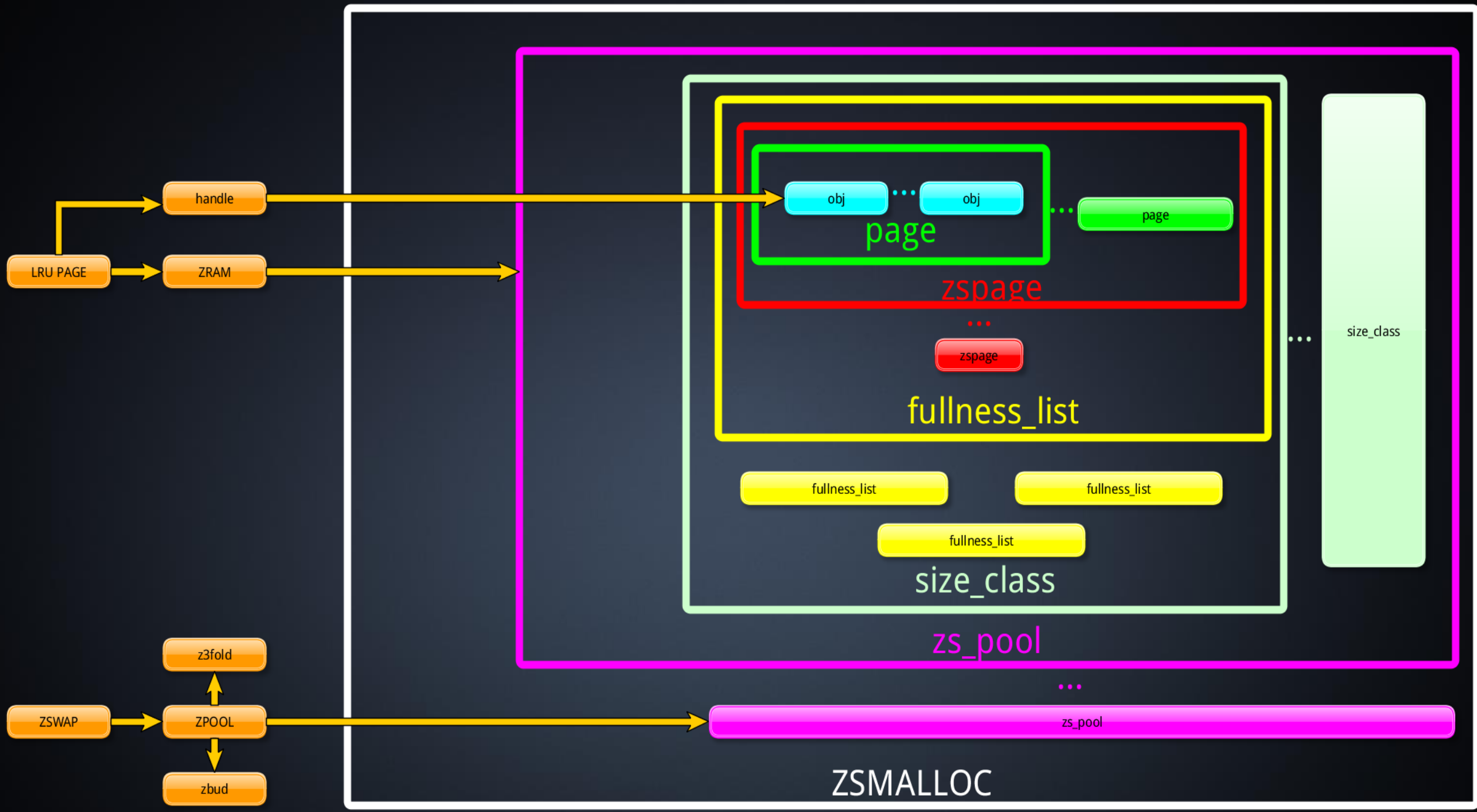
内部碎片（见下图）

- 一个理论上存在但是我没太观测到过的问题。
- 大概现象应该就是感觉开ZRAM时间长了以后感觉内存还是不够。杀掉一堆进程后又感觉好点了。
- 让人想起每个人的Android系统，有事没事鼓励用户杀APP。

内部碎片的成因（见下图）

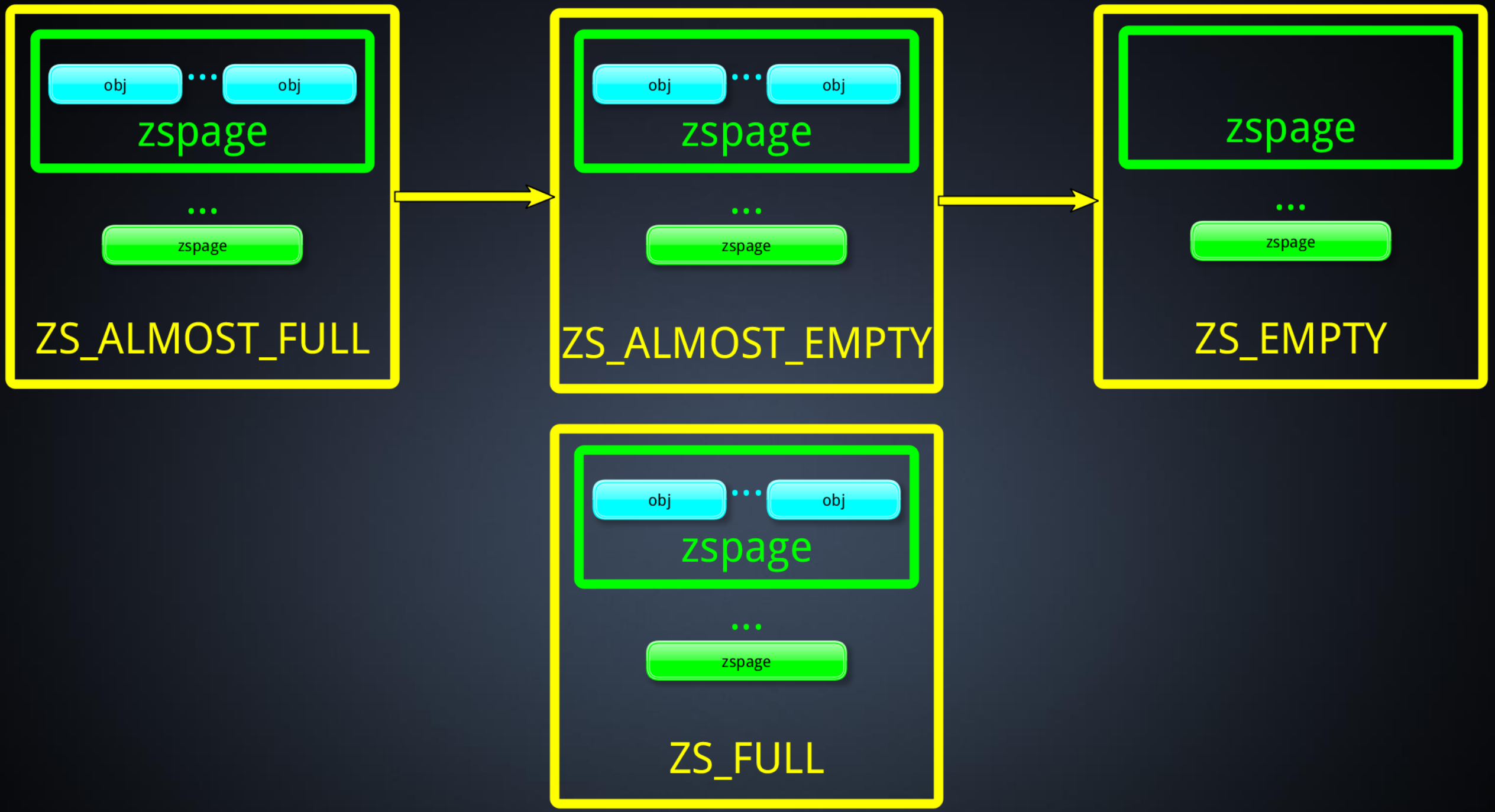
- 针对一个尺寸的size_class中实际存储数据的是obj。
- 而obj被存储在zspage中，随着使用会不断有分配和释放，如果没有相应碎片处理，则会出现很多碎片化的zspage。

- 注意这张图和前面的图不同，因为前面的图是根据修正后来做的。


handle
LRU PAGE
ZRAM
obj
obj
page
page
zspage
zspage
fullness_list
fullness_list
fullness_list
fullness_list
size_class
size_class
zs_pool
z3fold
ZSWAP
ZPOOL
zbud
zs_pool
ZSMALLOC

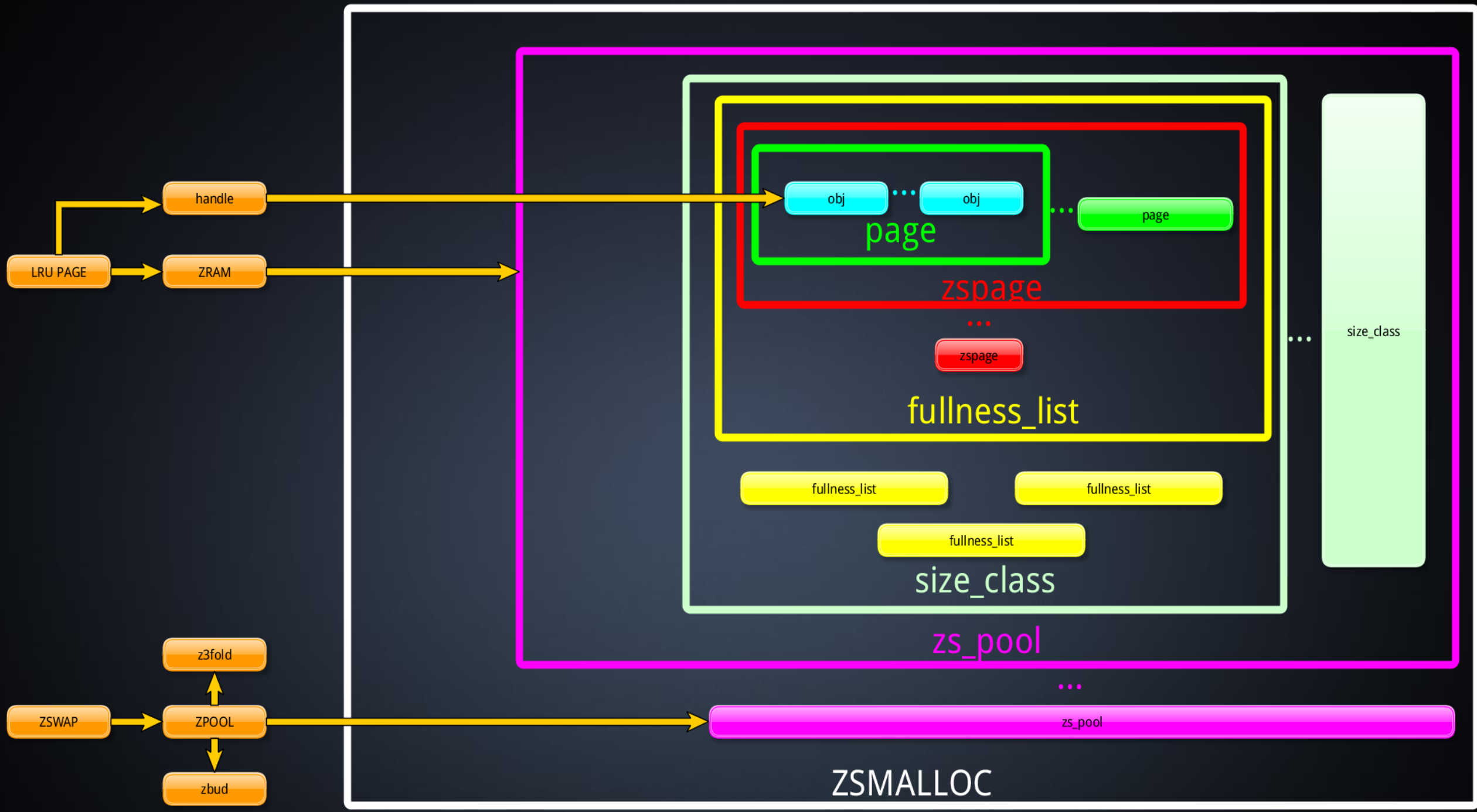
fullness_list最早的抗碎页机制（见下图）

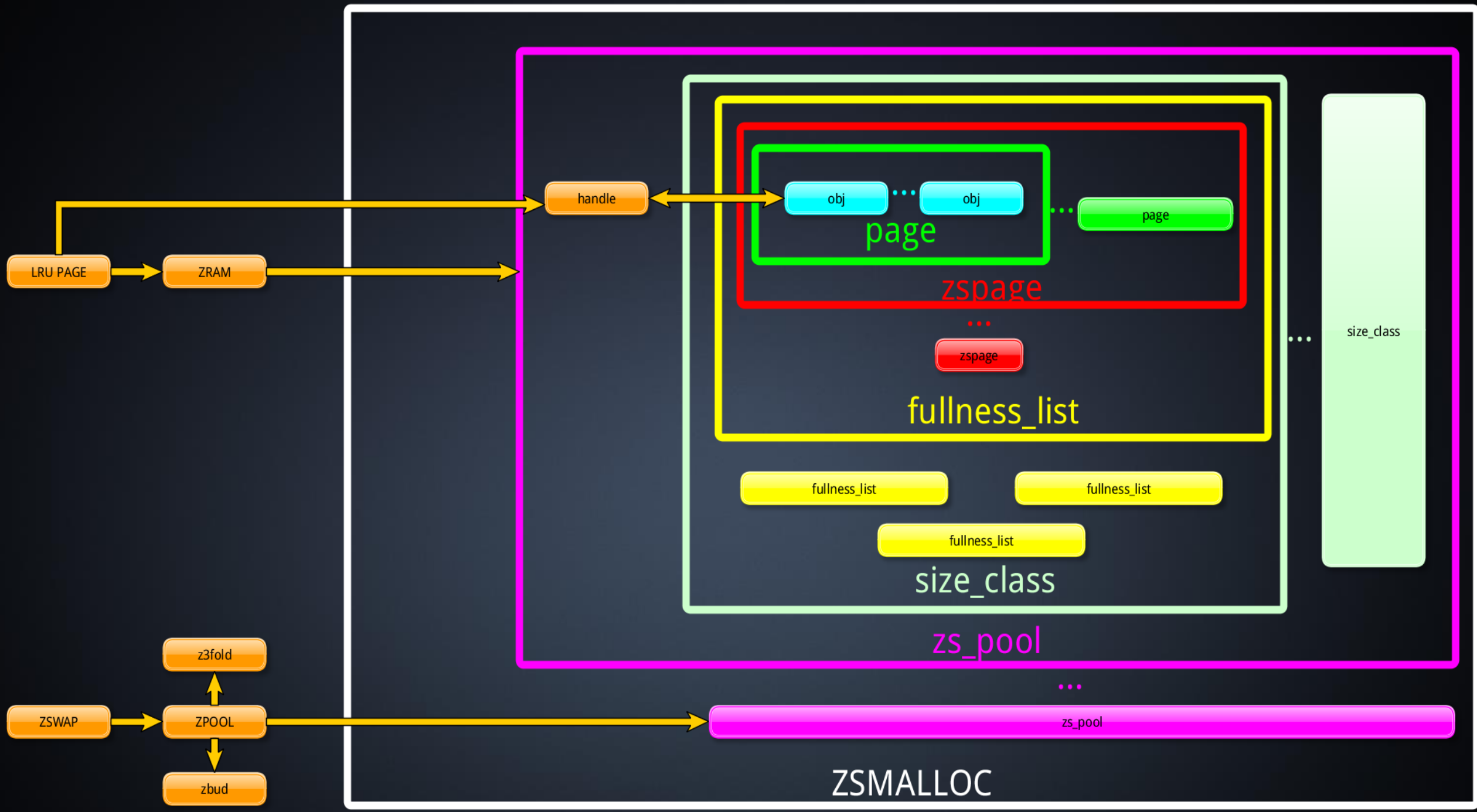
- ZS_EMPTY，ZS_ALMOST_EMPTY，ZS_ALMOST_FULL，ZS_FULL这4个fullness_list。每个LIST储存的ZSPAGE是其标记的所处状况。使用的时候优先从ZS_ALMOST_FULL中找ZSPAGE。
- 缺点：完全被动，虽然分配可以优化，但是释放还是碎片化的。


obj
obj
zspage
zspage
ZS_ALMOST_FULL
obj
obj
zspage
zspage
ZS_ALMOST_EMPTY
zspage
zspage
ZS_EMPTY
obj
obj
zspage
zspage
ZS_FULL

__zs_compact主动碎片清理(1)准备工作（见下图）

- 将handle从存储指向obj的数据改为指向obj的指针。可以根据obj位置的变化修改handle内容，ZRAM还可以找到某个页面对应的obj。


handle
LRU PAGE
ZRAM
obj
obj
page
page
zspage
zspage
fullness_list
fullness_list
fullness_list
fullness_list
size_class
size_class
zs_pool
z3fold
ZSWAP
ZPOOL
zbud
zs_pool
ZSMALLOC


handle
obj
obj
page
page
zspage
zspage
fullness_list
fullness_list
fullness_list
fullness_list
size_class
size_class
zs_pool
zs_pool
LRU PAGE
ZRAM
ZSWAP
ZPOOL
z3fold
zbud
ZSMALLOC

__zs_compact主动碎片清理(2)实际处理（见下图）

对一个size_class进行如下处理：

按照先ZS_ALMOST_EMPTY，后ZS_ALMOST_FULL的顺序抽出一个来源zspage。

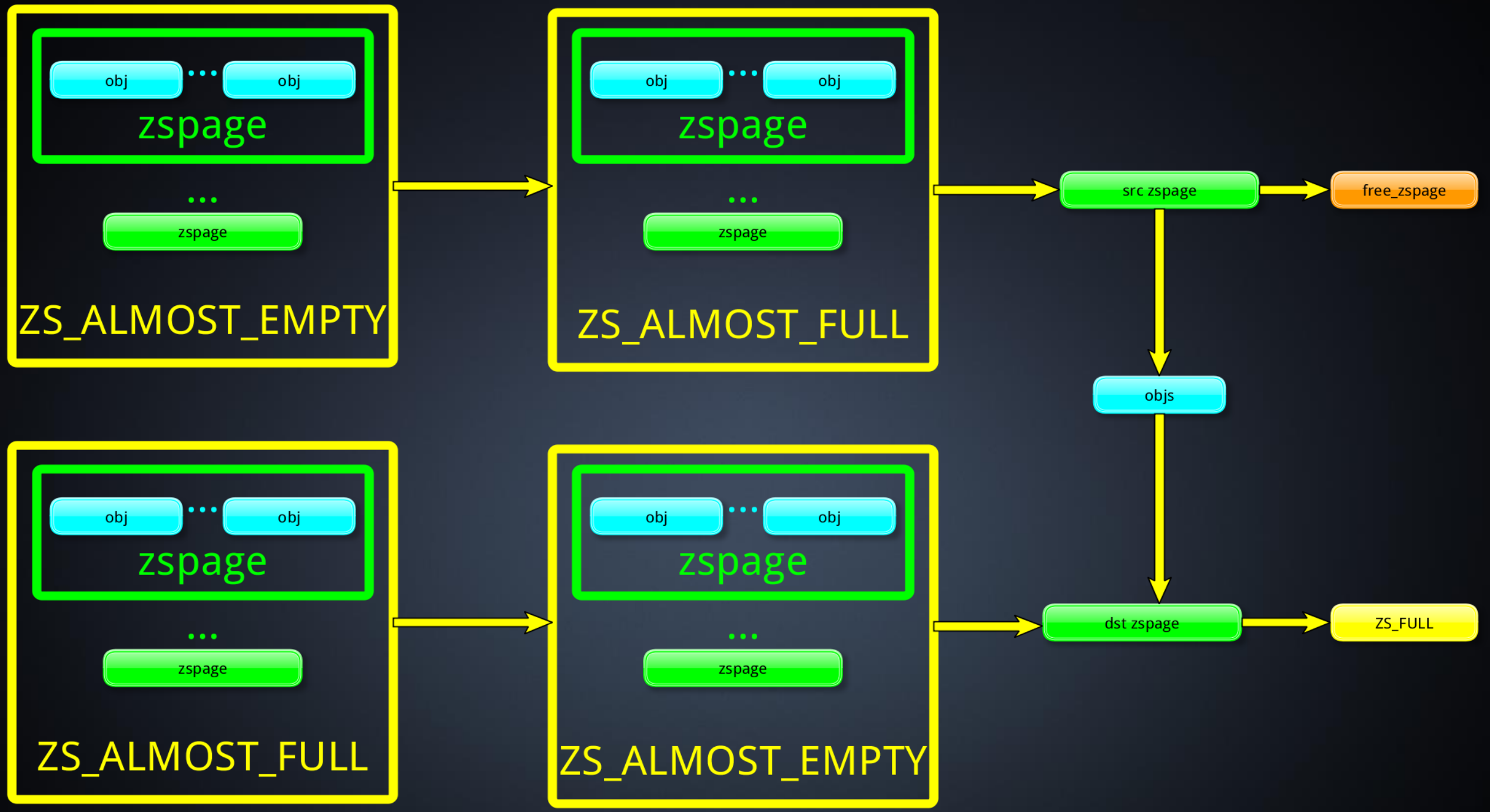
然后按照先ZS_ALMOST_FULL，后ZS_ALMOST_EMPTY的顺序抽出一个目标zspage。

把来源zspage中的obj依次移动到目标zspage中。

将已经空了的来源zspage释放掉，再按照刚才的方法取得一个来源zspage。

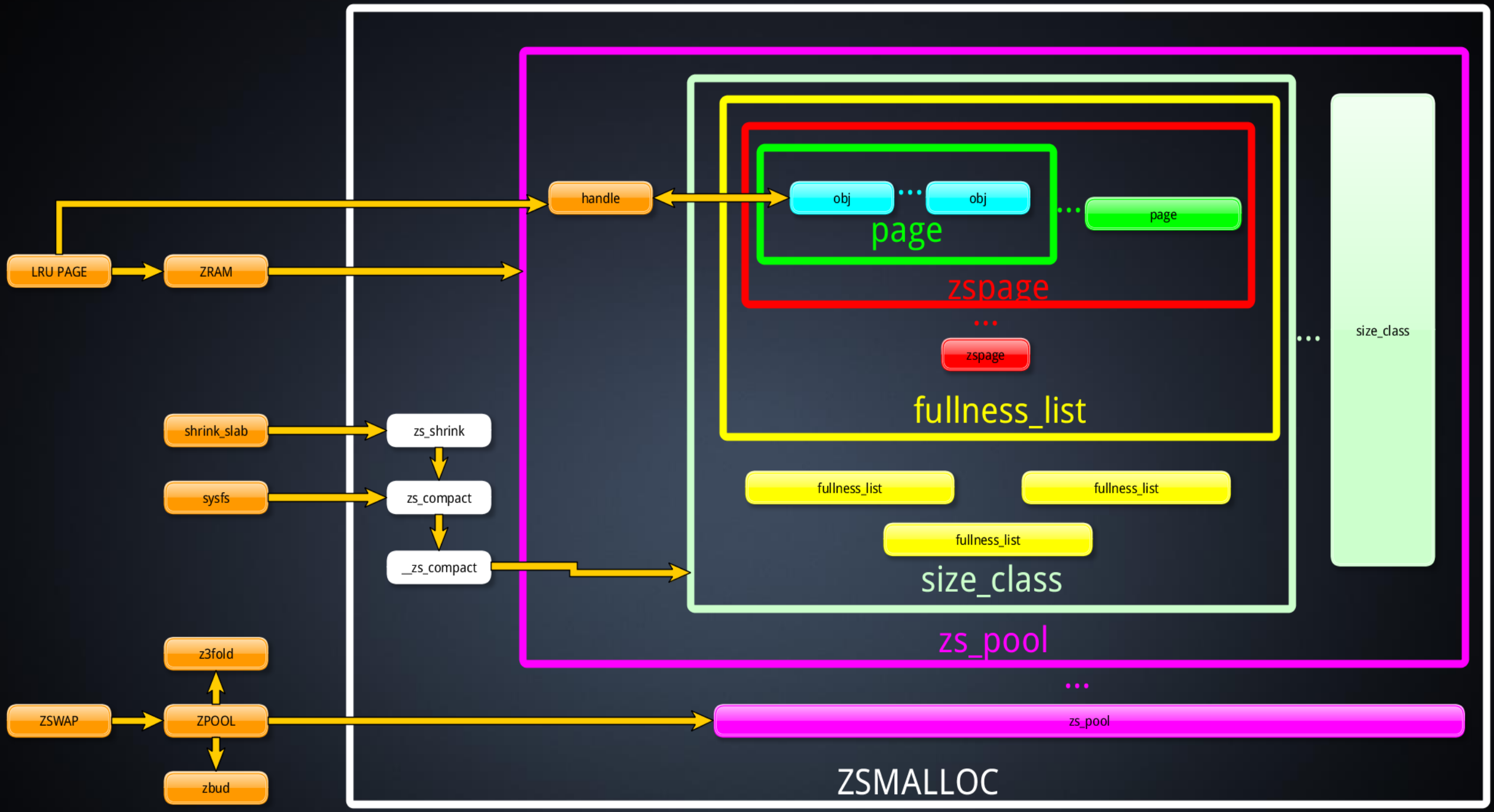
将满了的目标zspage加入ZS_FULL，再按照刚才的方法取得一个目标zspage。

如此循环，直到无法取得来源zspage或者目标zspage。


obj
obj
zspage
zspage
ZS_ALMOST_EMPTY
obj
obj
zspage
zspage
ZS_ALMOST_FULL
src zspage
free_zspage
objs
obj
obj
zspage
zspage
ZS_ALMOST_FULL
obj
obj
zspage
zspage
ZS_ALMOST_EMPTY
dst zspage
ZS_FULL

__zs_compact主动碎片清理(3)调用处理（见下图）

- zs_compact会依次调用__zs_compact对一个zs_pool中的每个size_class进行清理。
- 需要的时候可通过sysfs接口调用清理。
- 或者shrink_slab会调用zs_shrink，zs_shrink会调用zs_compact根据需要需要做清理。


handle
obj
obj
page
page
zspage
zspage
fullness_list
size_class
fullness_list
fullness_list
fullness_list
size_class
zs_pool
zs_pool
LRU PAGE
ZRAM
shrink_slab
sysfs
zs_shrink
zs_compact
__zs_compact
z3fold
ZSWAP
ZPOOL
zbud
ZSMALLOC

内部碎片问题的相关PATCH

- 相关的PATCH主要集中在zsmalloc.c，集成难度不大，建议内核不是很老的系统遇到相关问题可以打上尝试一下。

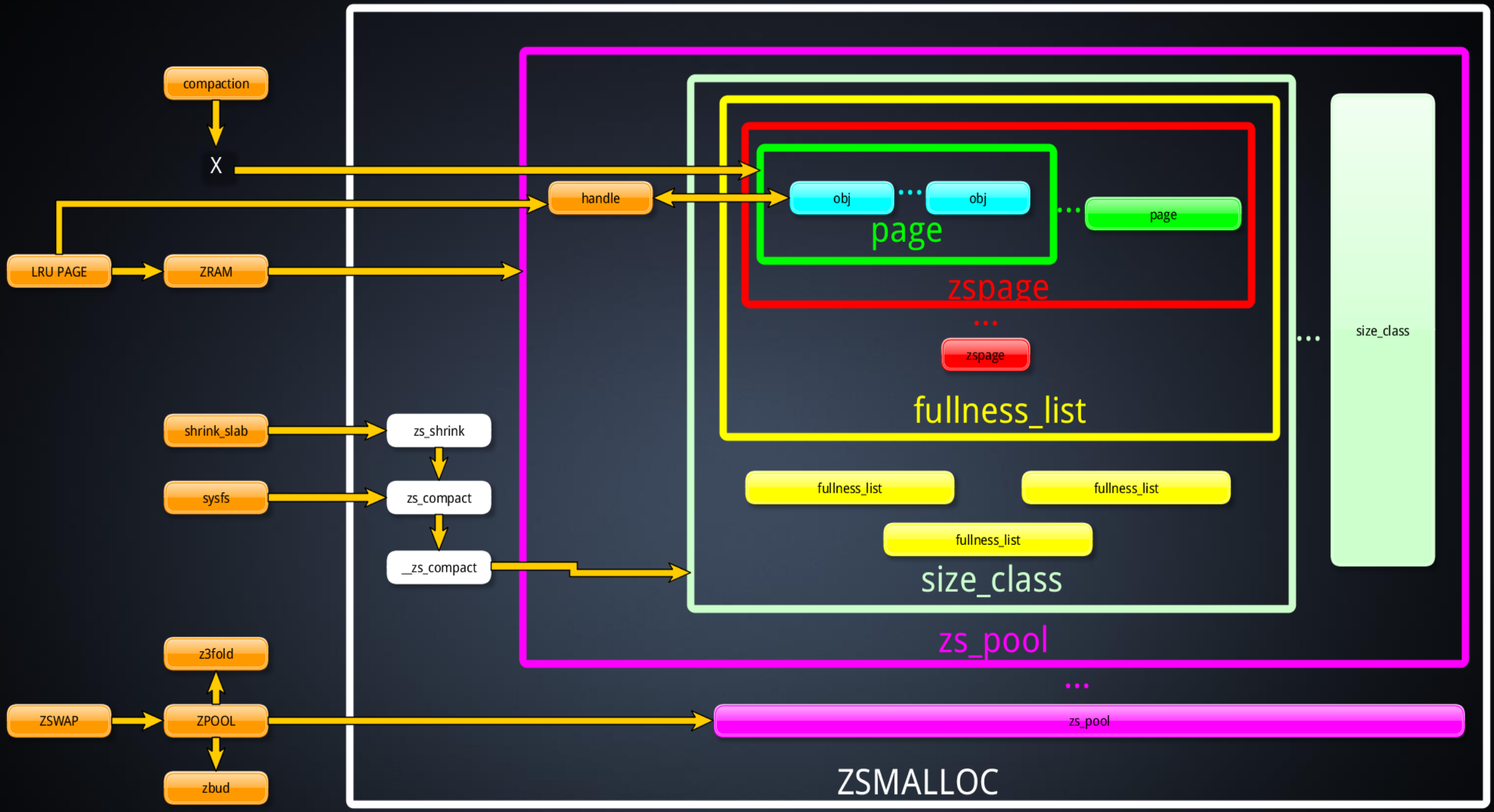
外部碎片和 不可移动页面（见下图）

- 因为一个设计引起的问题，所以放在一起介绍。
- 现象：
  1. 深度使用ZRAM后系统经常遇到连续页面分配失败的报错。
  2. 因为页面碎片增加引起的系统变慢。
  3. 在打上各种主动使用CMA相关的PATCH后(具体可见到2014年在CLK上关于CMA的演讲)，ZRAM中存放比较多内存后，还会遇到CMA页面剩下很多，普通内存剩下很少的情况。

外部碎片和不可移动页面的成因（见下图）

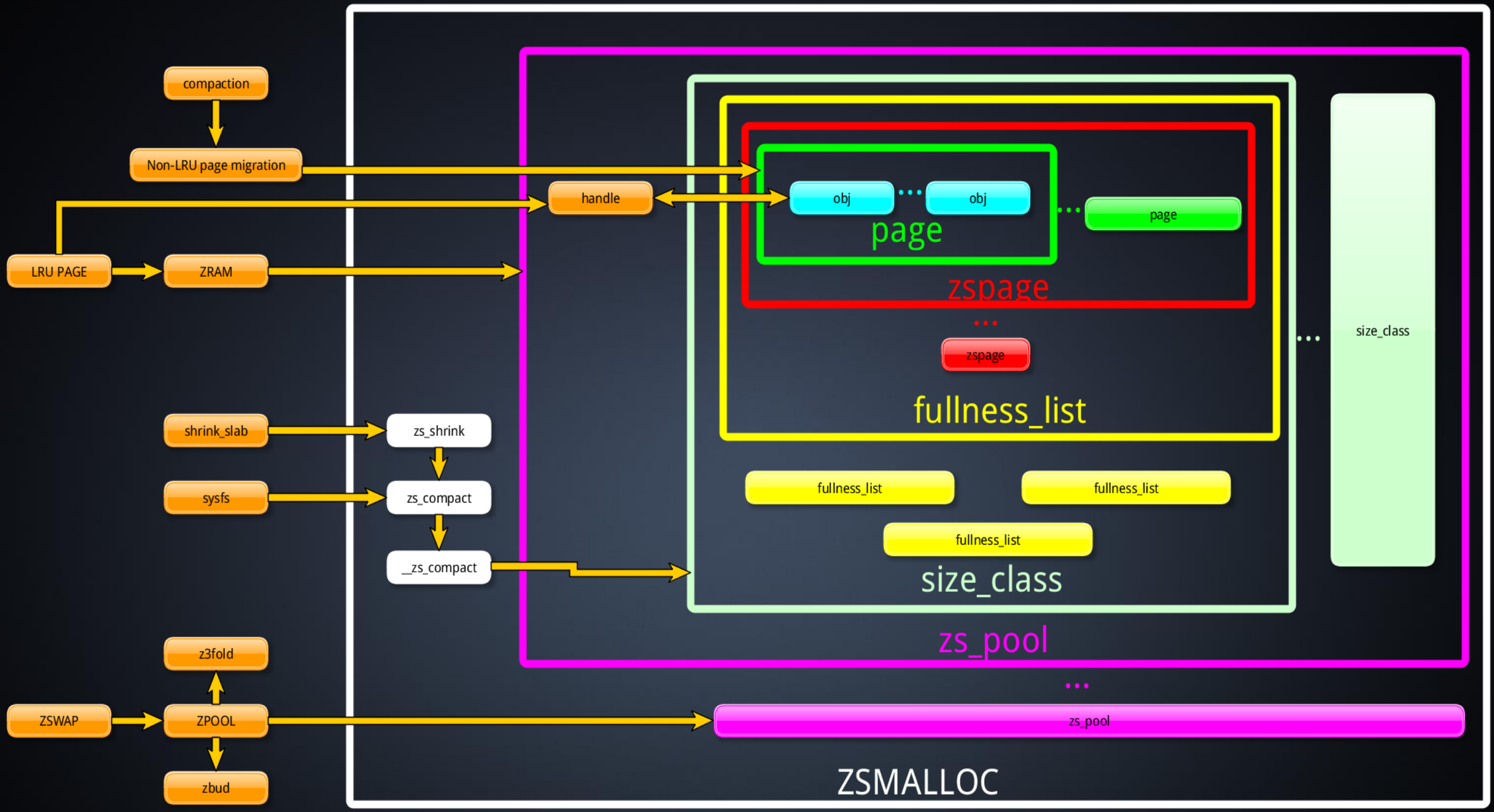
- ZSMALLOC分配的页面为不可移动页面。
- 被放入ZRAM的LRU页面(可移动)变成不可移动页面，这些页面不能被LINUX内核系统的除碎页功能处理，导致系统页面碎片化严重。
- 导致了现象1和现象2。
- CMA页面不能被当成不可移动页面分配(原理同样见我2014年在CLK的演讲)，导致了现象3。

- 题外话：碎片问题在A64上被放大因为A64的THREAD_SIZE比A32大。
- 除ZSMALLOC外，去年到今年还有若干相关提高。


compaction
X
handle
LRU PAGE
ZRAM
obj
obj
page
page
zspage
zspage
fullness_list
size_class
shrink_slab
sysfs
zs_shrink
zs_compact
__zs_compact
fullness_list
fullness_list
fullness_list
size_class
zs_pool
z3fold
ZSWAP
ZPOOL
zbud
zs_pool
ZSMALLOC

外部碎页解决方案：Non-LRU page migration（见下图）

- 没有条件就创造条件，不能migration就让它能migration。
- 在ZSMALLOC内部，handle已经在前面处理好可以保证obj迁移到其他页面，处理掉之前保存在page结构里面的ZSMALLOC相关信息，实现了相关接口就令ZSMALLOC中使用的页可迁移了。


compaction
Non-LRU page migration
handle
LRU PAGE
ZRAM
obj
obj
page
page
zspage
zspage
fullness_list
size_class
shrink_slab
zs_shrink
sysfs
zs_compact
__zs_compact
fullness_list
fullness_list
fullness_list
size_class
zs_pool
z3fold
ZSWAP
ZPOOL
zbud
zs_pool
ZSMALLOC

外部碎页问题的相关PATCH

- 因为要对页面移动相关代码进行修改，涉及到不少Linux内存管理相关代码，有一定难度。
- 依赖内部碎页的相关修正patch。

class	size	almost_full	almost_empty	obj_allocated	obj_used	pages_used	pages_per_zspage
0	32	0	0	0	0	0	1
1	48	3	1	2560	2434	30	3
2	64	2	1	1280	1221	20	1
3	80	1	1	663	637	13	1
4	96	0	1	640	600	15	3
5	112	0	1	365	326	10	2
6	128	0	1	352	342	11	1
7	144	0	1	425	385	15	3
8	160	1	1	357	327	14	2
9	176	0	1	372	317	16	4
10	192	1	0	448	434	21	3
11	208	0	1	390	360	20	2
12	224	1	0	292	281	16	4
13	240	1	1	272	264	16	1
14	256	0	1	224	212	14	1
15	272	0	1	225	219	15	1
16	288	0	1	224	214	16	1
17	304	0	1	240	210	18	3
18	320	0	1	204	173	16	4
19	336	0	1	168	165	14	1
20	352	0	1	207	191	18	2
21	368	0	1	275	269	25	1
22	384	0	1	192	172	18	3
23	400	1	0	140	139	14	1
24	416	1	0	156	153	16	4
25	432	0	1	168	146	18	3
26	448	0	1	153	145	17	1
27	464	0	1	175	143	20	4
28	480	0	1	187	172	22	2
29	496	1	0	132	127	16	4
30	512	0	1	152	146	19	1
31	528	0	1	186	170	24	4
32	544	1	1	150	142	20	2
33	560	0	2	174	153	24	4
34	576	1	0	168	167	24	1
35	592	0	2	162	140	24	4
36	608	0	1	160	153	24	3
37	624	0	1	130	122	20	2
38	640	0	1	152	136	24	3
40	672	1	0	318	317	53	1
42	704	0	1	322	310	56	4
43	720	0	1	170	158	30	3
44	736	0	1	165	162	30	2
46	768	0	1	288	276	54	3
49	816	0	0	425	425	85	1
51	848	1	0	323	320	68	4
52	864	2	0	154	151	33	3
54	896	0	1	315	311	70	2
57	944	0	1	455	450	105	3
58	960	0	0	136	136	32	4
62	1024	0	1	616	614	154	1
66	1088	0	1	720	714	192	4
67	1104	0	1	187	182	51	3
71	1168	1	1	805	798	230	2
74	1216	1	0	720	718	216	3
76	1248	0	1	468	459	144	4
83	1360	0	1	1503	1502	501	1
91	1488	0	1	1551	1544	564	4
94	1536	0	1	616	614	231	3
100	1632	0	0	1345	1345	538	2
107	1744	0	1	1596	1594	684	3
111	1808	0	1	765	758	340	4
126	2048	0	0	2426	2426	1213	1
144	2336	0	1	1960	1958	1120	4
151	2448	1	0	545	544	327	3
168	2720	0	1	1389	1387	926	2
190	3072	0	1	1564	1562	1173	3
202	3264	0	1	45	43	36	4
254	4096	0	0	23306	23306	23306	1
Total		22	54	58118	57191	33259	

再次查找彩蛋

ZRAM的低压缩率问题

190	3072	0	1	1564	1562	1173	3
202	3264	0	1	45	43	36	4
254	4096	0	0	23306	23306	23306	1
Total		22	54	58118	57191	33259	

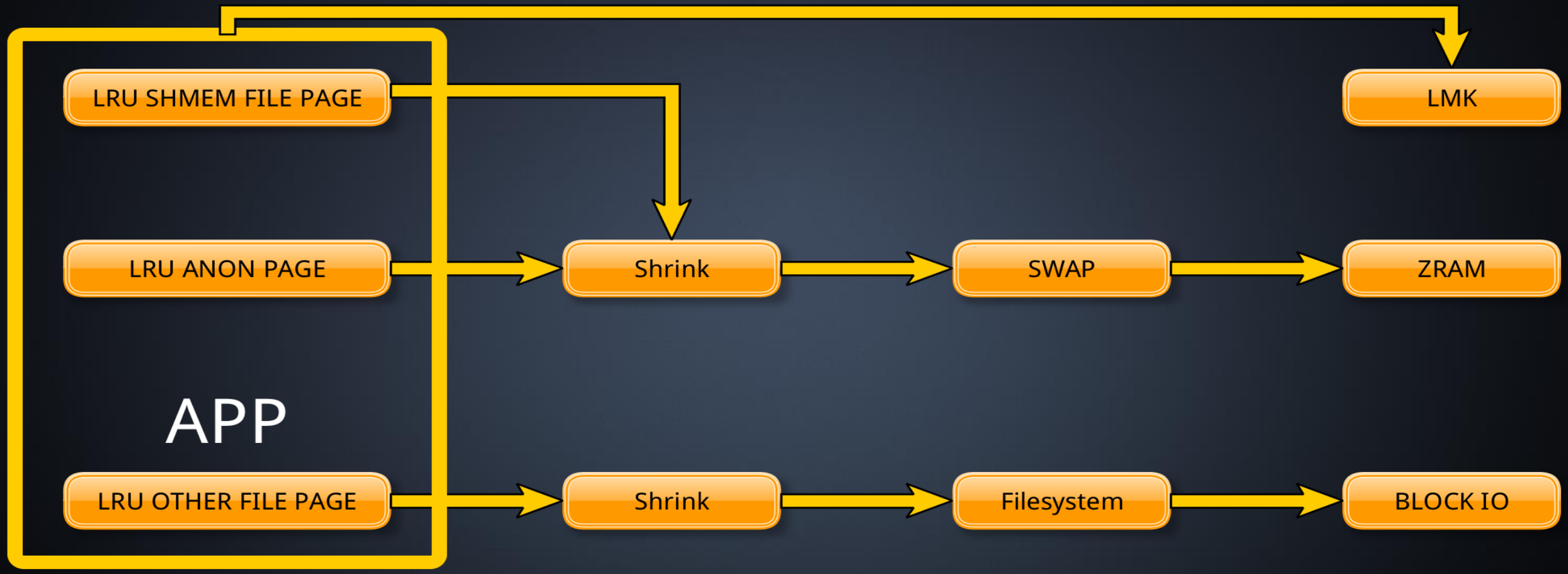
- 这是我看到的一个比较极端的例子，ZRAM中70%的页面起到的压缩作用是0。
- 实际情况可能没有这么糟糕，另外CONFIG_ZSMALLOC_STAT选择默认不打开，所以一般人可以做到眼不见心不烦。
- 这样的PAGE浪费了CPU的时间，浪费了ZRAM空间。
- 这时勉强能起到的作用只有把非HIGHMEM ZONE的页面保存到HIGHMEM ZONE上，在一定程度上节省了非HIGHMEM ZONE。
  但是64位时代到来，使用HIGHMEM减少，优势不再明显。


不给力呀老湿

ZRAM的本质，提高思路的来源（见下图）

- 用CPU和少量内存换取系统BLOCK IO和比较多数量内存。
- 存放到ZRAM中的页面压缩率越好，ZRAM的工作效果越好。
- 但是现在不能选择，只能照单全收。

Android有SWAP



Non-Swap（见下图）

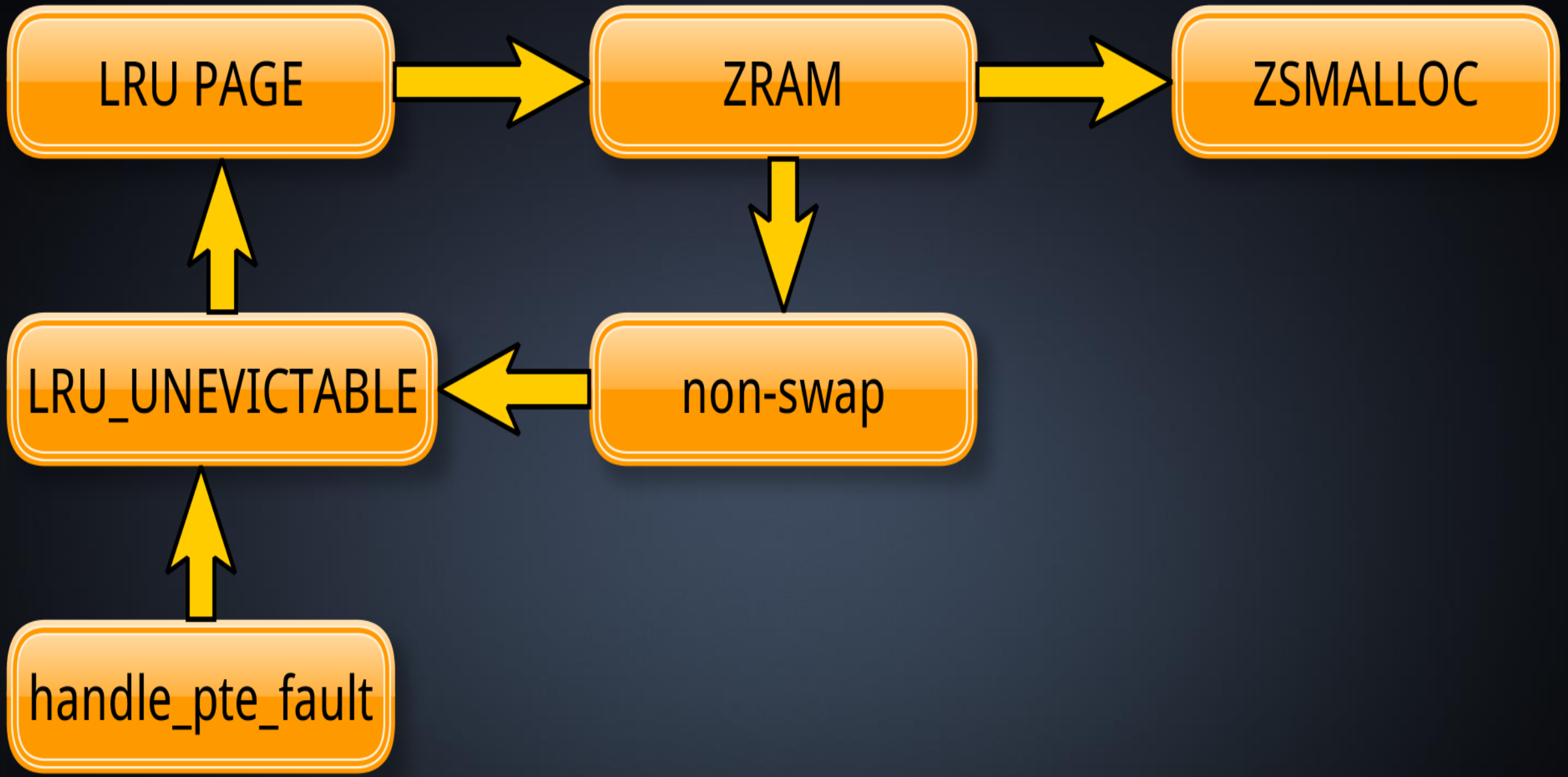
- 为了提高ZRAM压缩率。
- 其思路是把压缩率不高的页面不写入ZRAM，同时将其从LRU列表中抽出去放到单独的页列表保证其不会再次被写入ZRAM。
  当这些页面再次被写时，表明其有可能在压缩率上出现变化，将其再次放回LRU列表。

Non-Swap的实现（见下图）

- 根据设置放弃在ZRAM中存储压缩率不高的页。

- 具体步骤：
  shrink_page_list在发送PAGE到ZRAM之前把UNMAP PAGE改为把页面设置为只读。
  把页面发给ZRAM。
  ZRAM压缩页面，然后检查压缩率，压缩率不足的页面设置NON-SWAP标志。
  返回shrink_page_list，把没有NON-SWAP也没有在发送到ZRAM过程中被写入的页面UNMAP然后释放掉。
  把被设置为NON-SWAP的页面丢入UNEVICTABLE列表防止再次被shrink_page_list。
  一直到这个页面被写触发handle_pte_fault，在这里去掉NON-SWAP标志并踢回LRU PAGE列表。

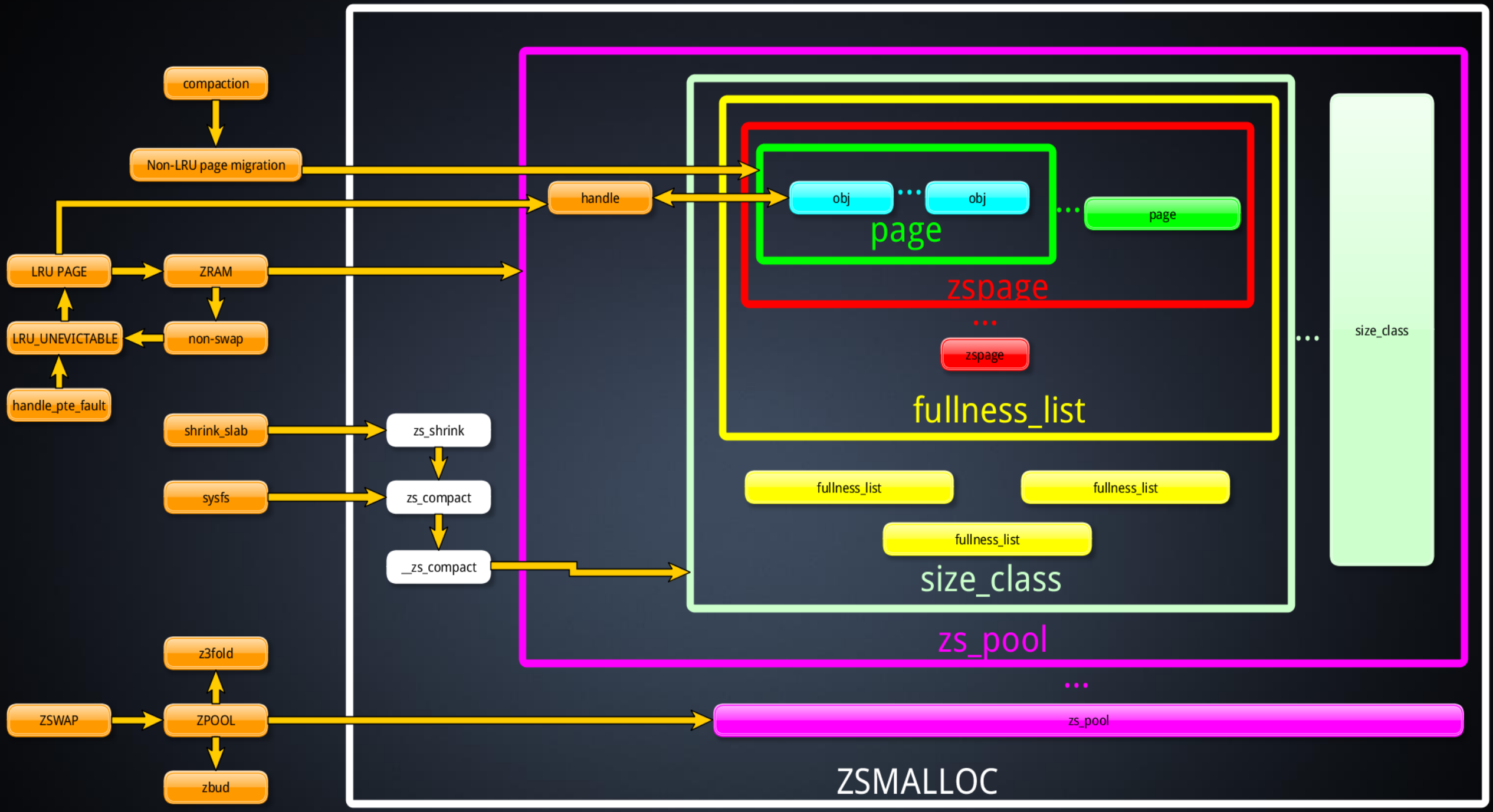
- https://lkml.org/lkml/2016/8/22/151


LRU PAGE
ZRAM
ZSMALLOC
LRU_UNEVICTABLE
non-swap
handle_pte_fault

Non-Swap的效果（见下图）

- 一个平台用内部稳定性测试。
  未打上Non-Swap功能PATCH和打开Non-Swap功能后对比结果，每次的LMK次数都降低50%以上。
  最好的一次降低了79%。
  能达到这点的原因我分析是因为当压缩率提高时，每做一组页面shrink都可以获得比低压缩率更高的内存释放。

- 另外就是在考虑如何处理Non-Swap页面的时候，想到了如何让系统支持任意多的watchpoint。
  不过超过本话题讨论范畴了。


compaction
Non-LRU page migration
handle
obj
obj
page
page
zspage
zspage
fullness_list
size_class
LRU PAGE
ZRAM
LRU_UNEVICTABLE
non-swap
handle_pte_fault
shrink_slab
zs_shrink
sysfs
zs_compact
__zs_compact
fullness_list
fullness_list
fullness_list
size_class
zs_pool
z3fold
ZSWAP
ZPOOL
zbud
zs_pool
ZSMALLOC

谢谢！ 问题？

- weibo: @teawater_z 欢迎在线吐槽