Page Replacement in Linux

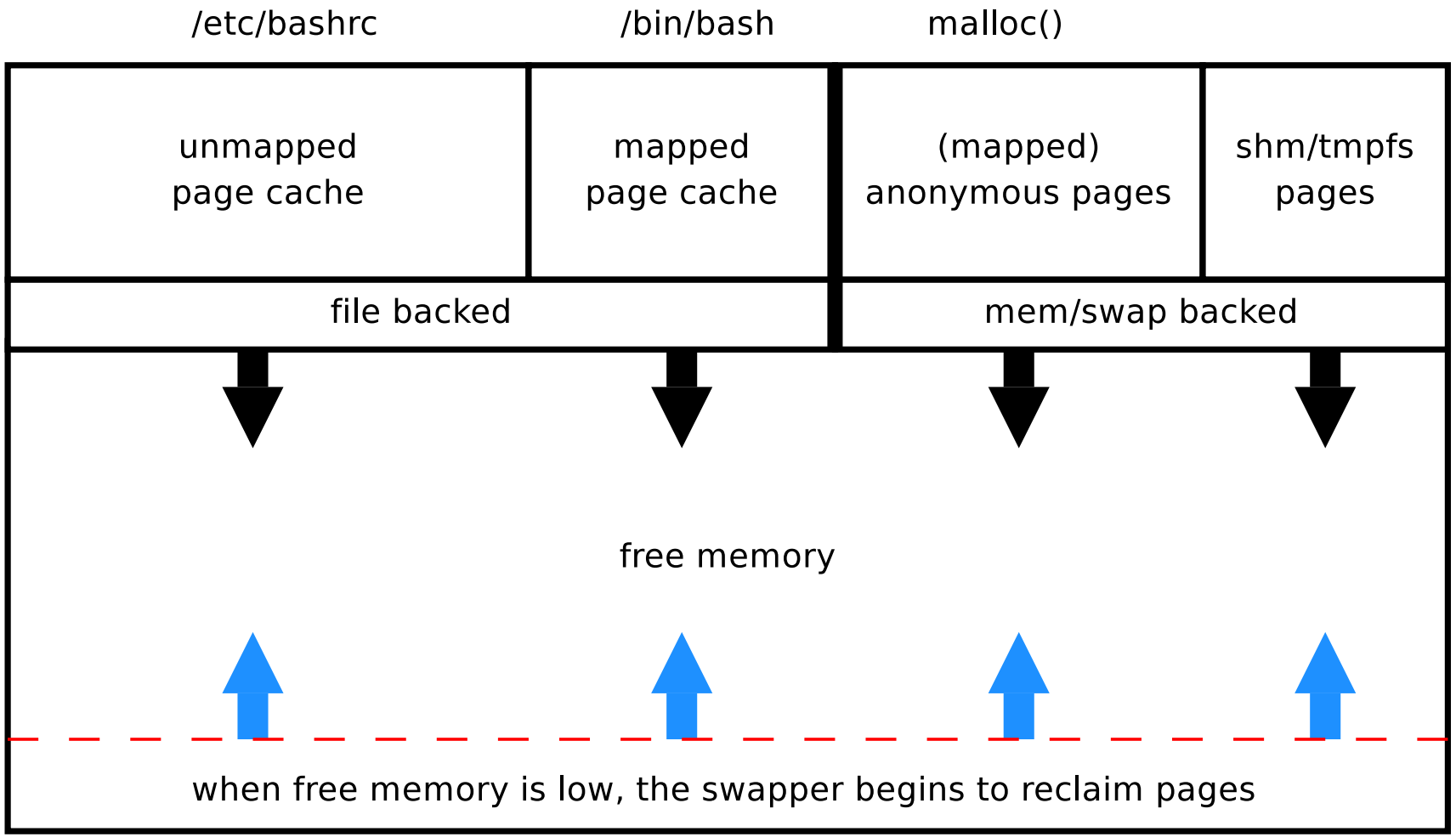
Wu Fengguang
<wfg@linux.intel.com>

October 22, 2009

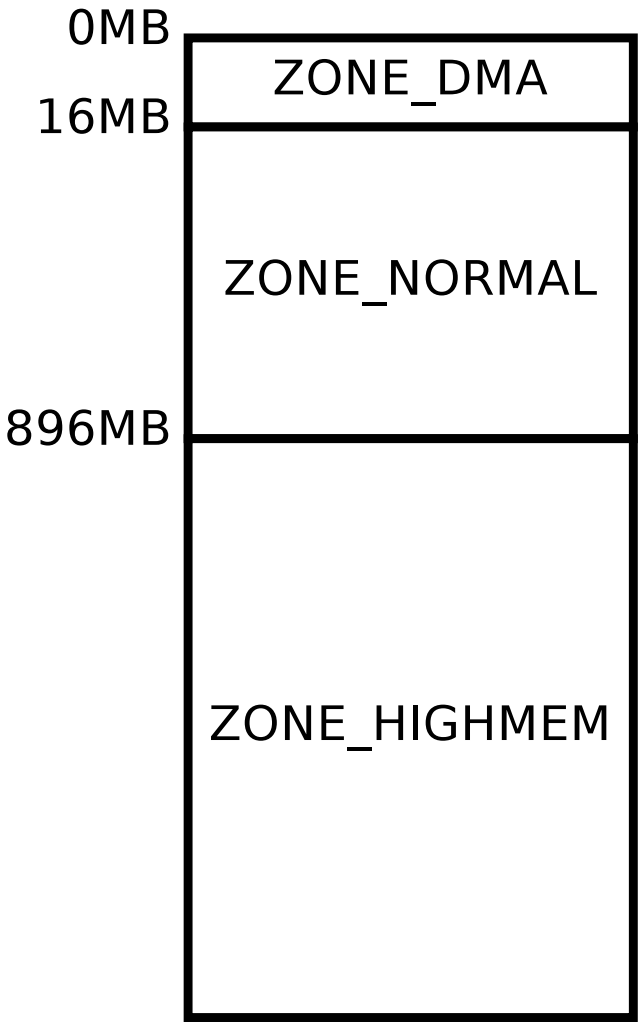
Outline

1. page types and organization
2. page reclaim logics

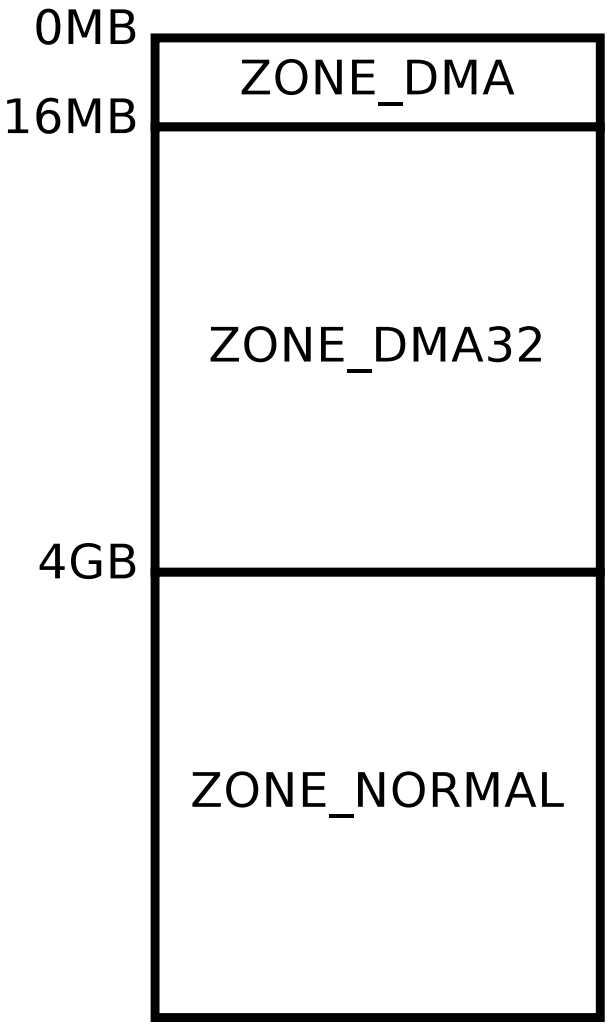
page reclaim



memory zones



x86_32



x86_64

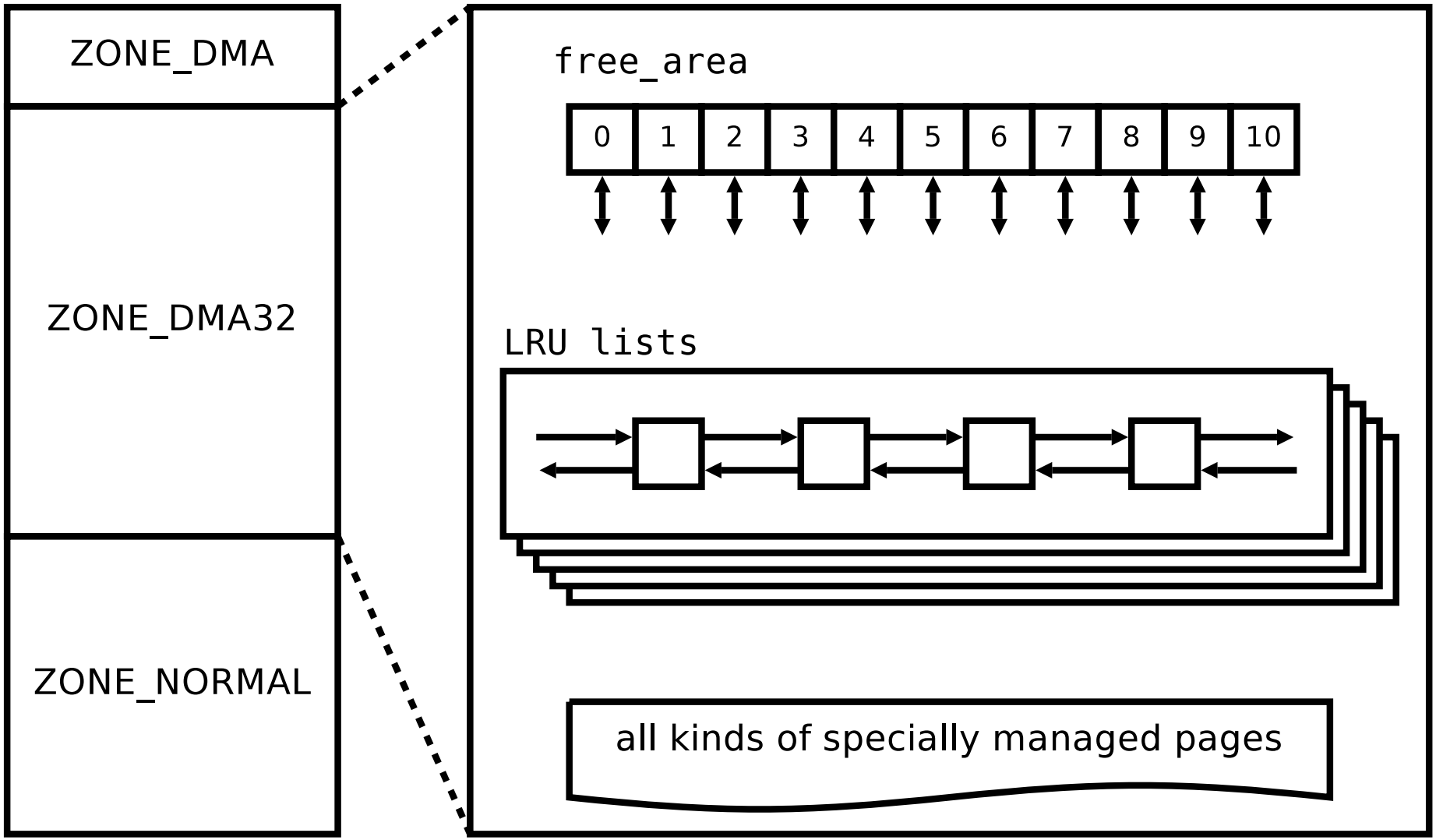
zoned page allocation

DMA	24-bit addressable
DMA32	32-bit addressable
NORMAL	direct addressable
HIGHMEM	addressable via (temp) mapping

__get_free_page(gfp_mask)

zone modifier	zone fallback order
__GFP_DMA	DMA
__GFP_DMA32	DMA32 => (DMA)
(unspecified)	NORMAL => DMA32 => (DMA)
__GFP_HIGHMEM	HIGHMEM => NORMAL => DMA32 => (DMA)

zone pages

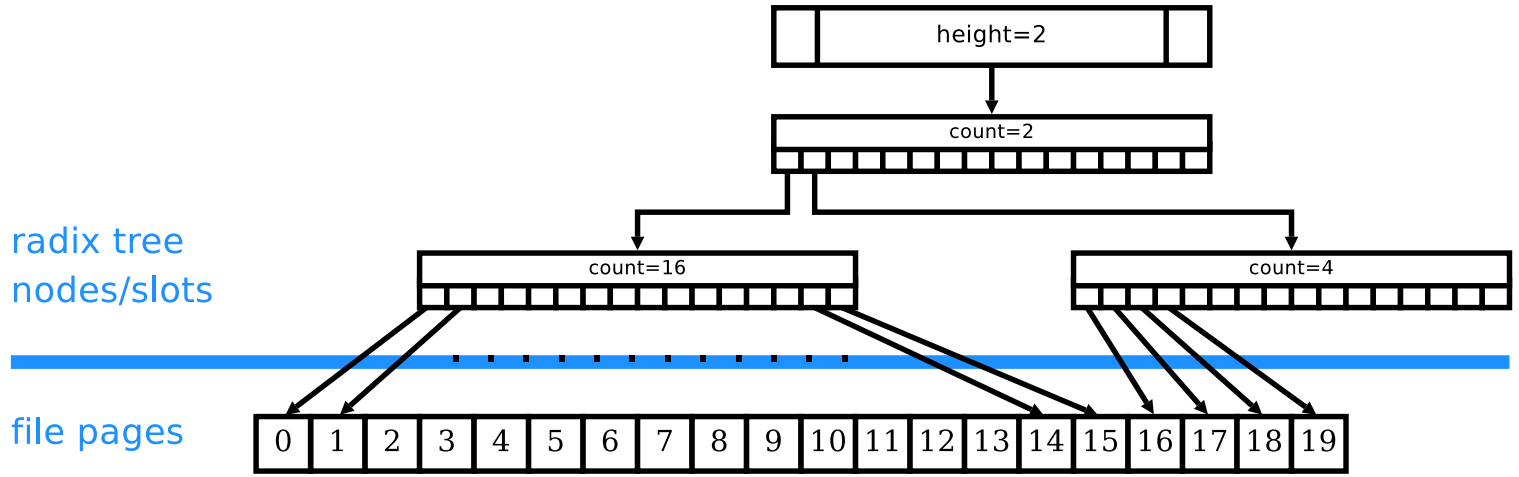

ZONE_DMA
ZONE_DMA32
ZONE_NORMAL
free_area
0
1
2
3
4
5
6
7
8
9
10
LRU lists
all kinds of specially managed pages

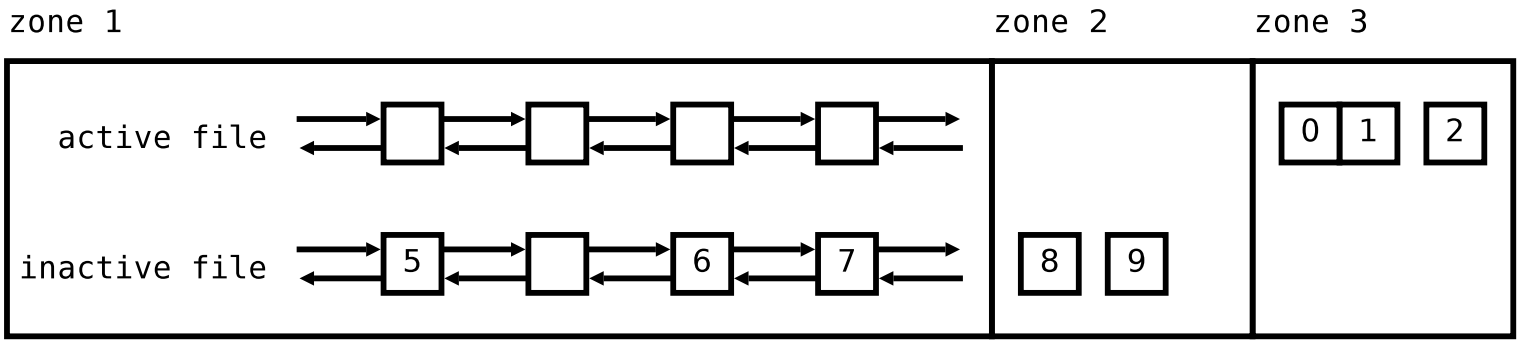
zone pages (cont.)

- **free pages**
- **LRU cache (in 5 lists)**
  - `file backed pages (active + inactive)`
  - `swap backed pages (active + inactive)`
  - `unevictable pages`
- **slab cache (reclaimable)**
  - `icache`
  - `dcache`
- **other pages**

file backed pages

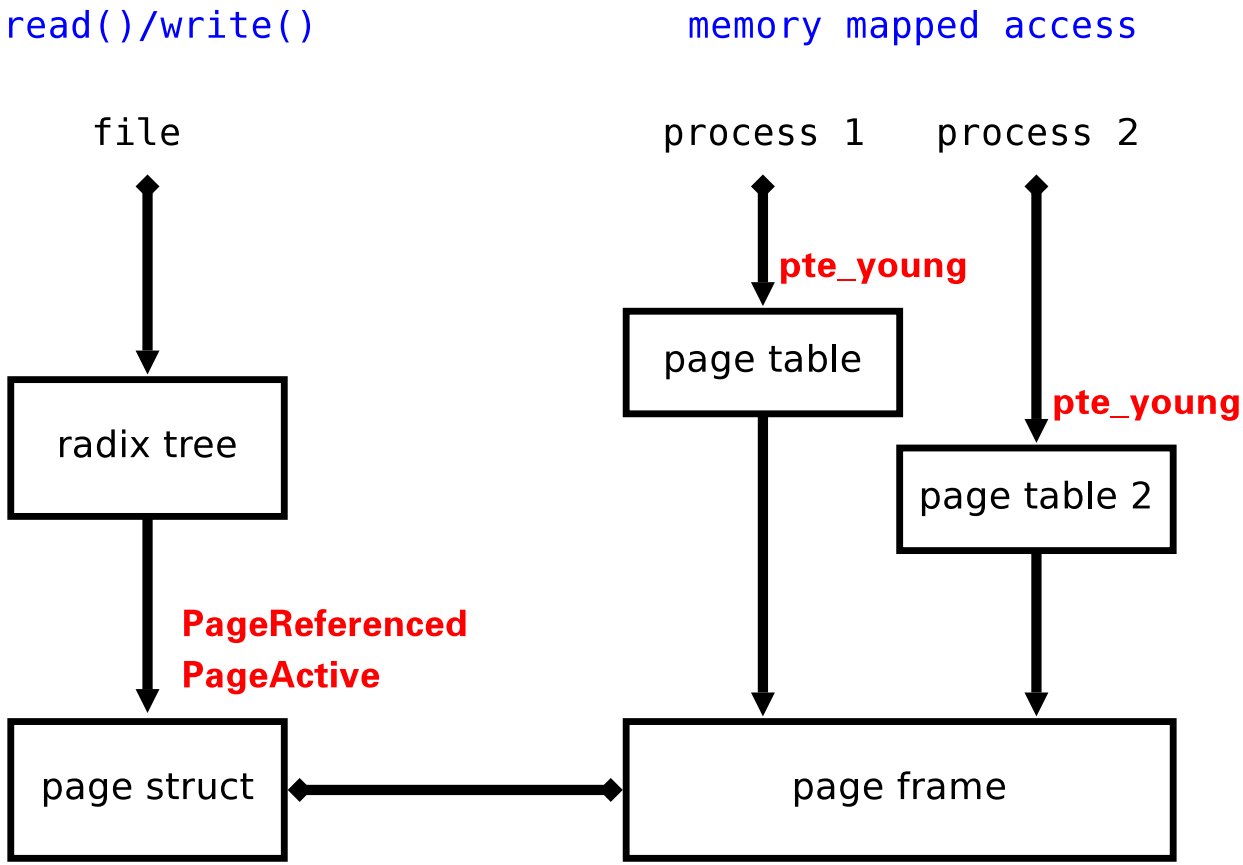
Managed in both **radix tree** and **file LRU lists**.





page referenced bits

- `read()` **sets** `PageReferenced` **or** `PageActive` **(activate it)**
- `mmap read` **sets** `pte_young`**, to be examined at reclaim time**

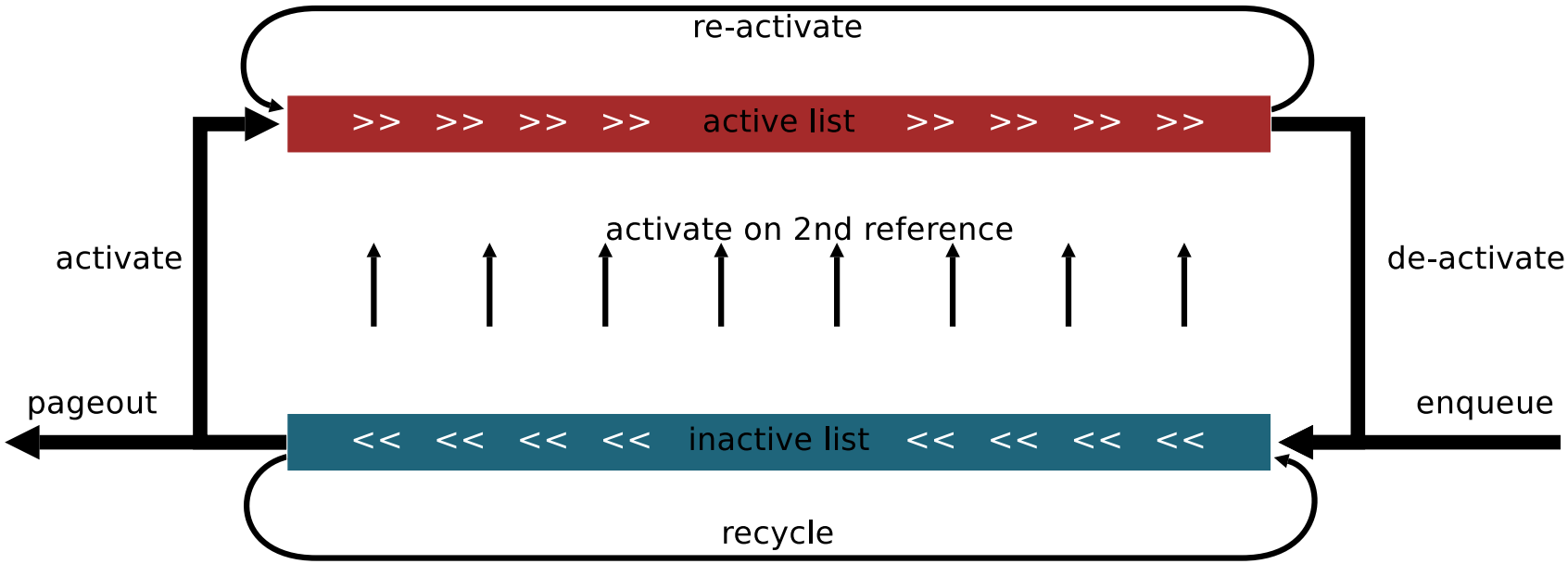


general cache rules

In order to stay in memory:

unmapped have to be referenced **2 times** when in **active+inactive** list

mapped have to be referenced **1 times** when in **inactive** list

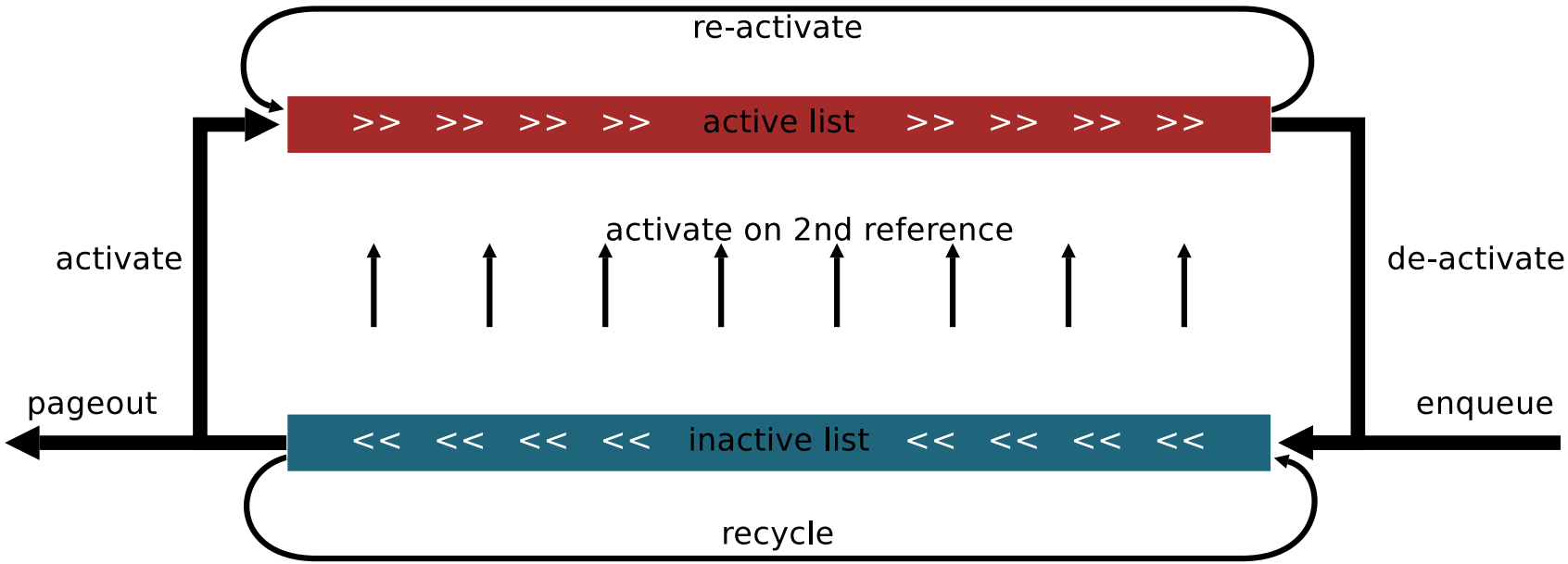


activate rules

activate on the 2nd `read()/write()` access

activate accessed mapped pages; some unfreeable pages

re-activate accessed program text

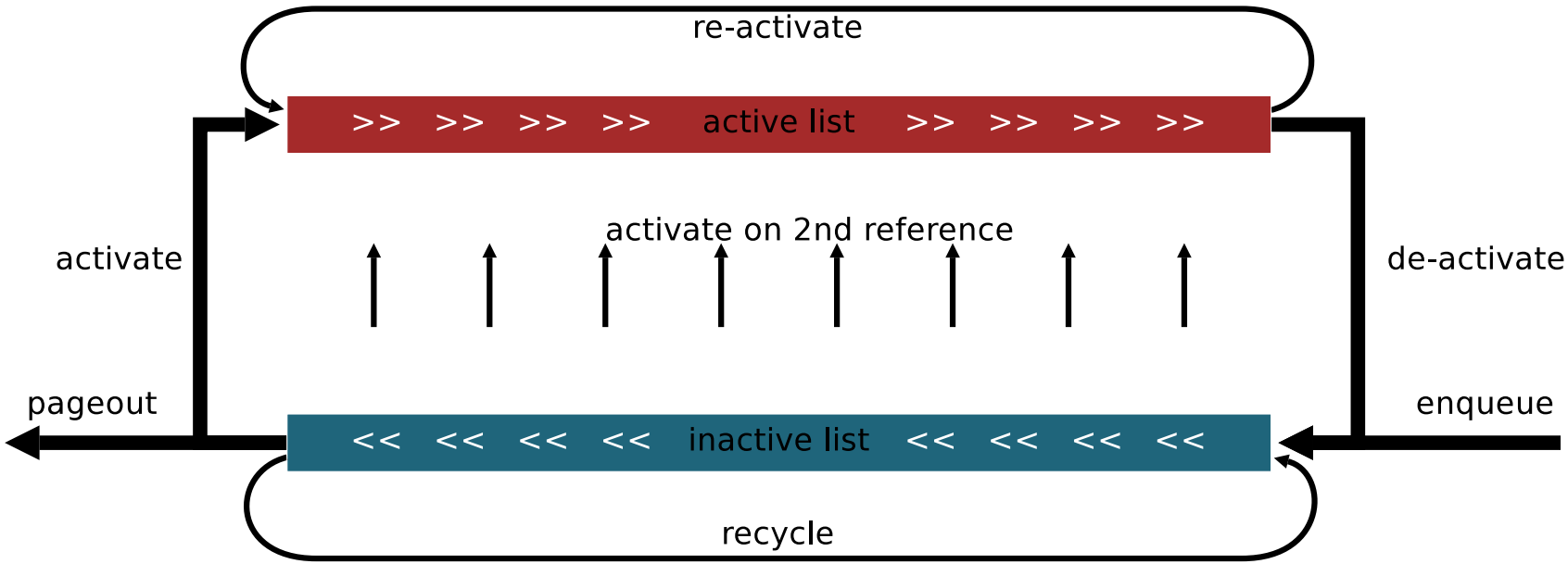


evict rules

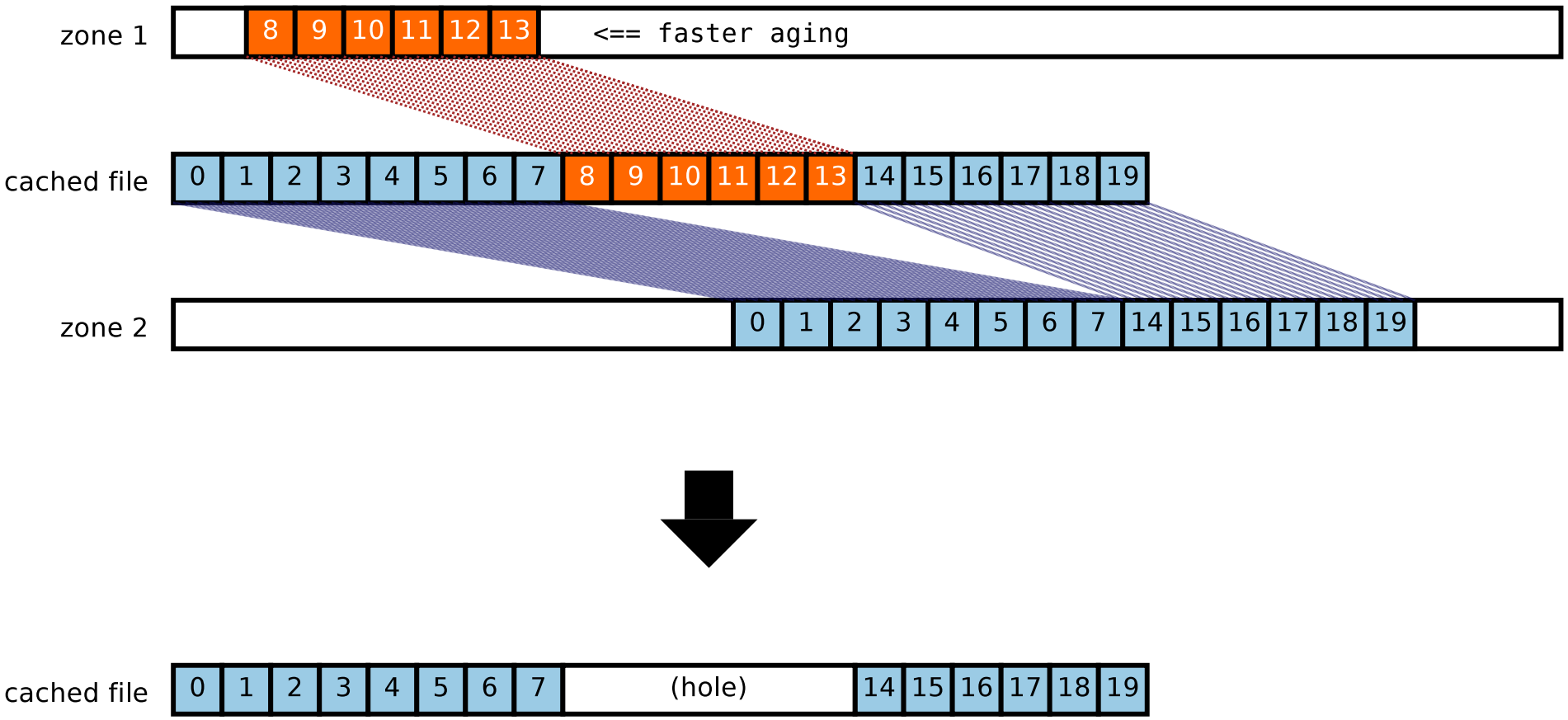
de-activate all unmapped pages; mapped data pages

pageout all unmapped pages; not accessed mapped pages

recycle pages put to writeback; some unreclaimable pages for now



balanced aging: why



On sequential read, need a small (24kb) I/O to fill the hole.

balanced aging: how

for ratio in $\frac{1}{4096}$, $\frac{1}{2048}$, $\frac{1}{1024}$, ..., $\frac{1}{2}$, 1

```
        for each zone
                scan (zone_nr_lru_pages * ratio) pages;

        if enough pages reclaimed
                break;
```

balanced aging: overscan of high zones

problemic scenario:

the 16MB DMA zone is cycled once per second

=>

the 500MB NORMAL zone is cycled once per second

=>

all pages not referenced within 1 second are evicted

=>

huge `nr_free_pages`

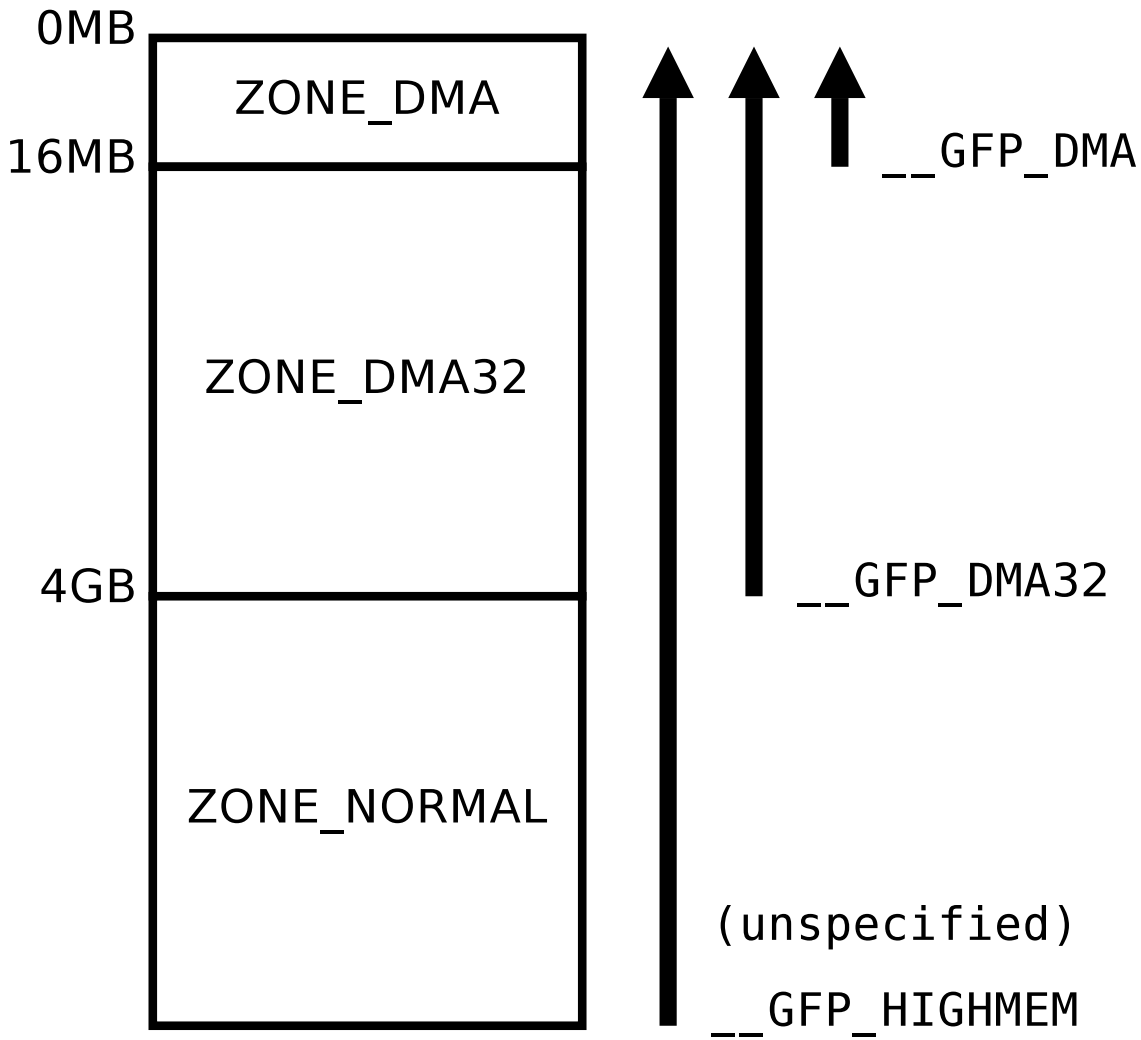
balanced aging: how 2

for ratio in $\frac{1}{4096}$, $\frac{1}{2048}$, $\frac{1}{1024}$, ..., $\frac{1}{2}$, 1

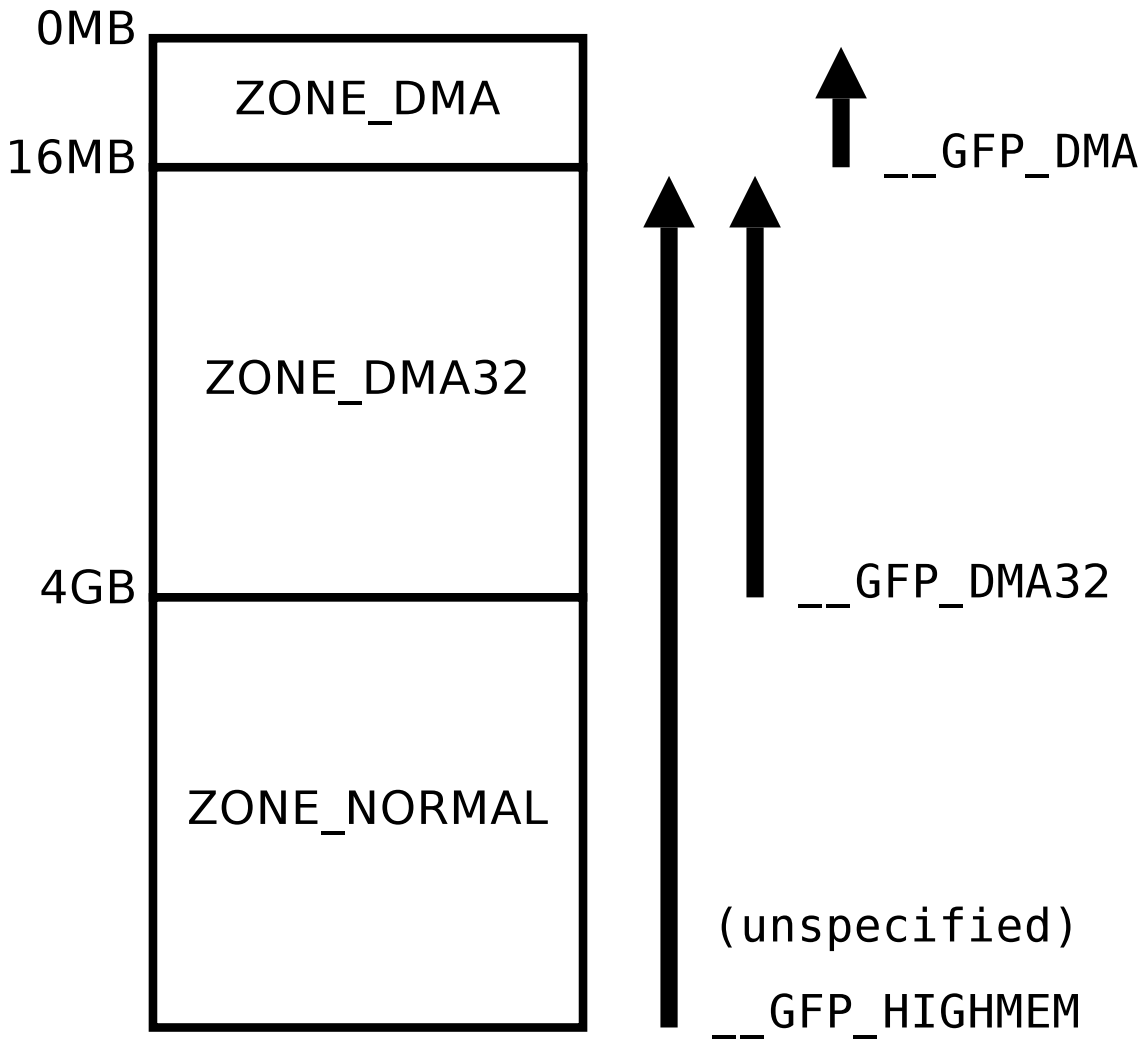
```
        for each zone from target zone to ZONE_DMA
                scan (zone_nr_lru_pages * ratio) pages;

        if enough pages reclaimed
                break;
```

balanced aging: overscan of lower zones



balanced aging: low mem reserve



balanced aging: over reclaim

for ratio in $\frac{1}{4096}$, $\frac{1}{2048}$, $\frac{1}{1024}$, ..., $\frac{1}{2}$, 1

```
        for each zone from target zone to lowest zone
                scan (zone_nr_lru_pages * ratio) pages

        if enough pages reclaimed
                break;
```

balanced aging: early break

So we reach the current 2.6.32 logic. Problems of the loops:

- **risks overscanning lower zones**
- **risks overscanning small zones**

for ratio in $\frac{1}{4096}$, $\frac{1}{2048}$, $\frac{1}{1024}$, ..., $\frac{1}{2}$, 1

```
        for each zone from target zone to lowest zone
                scan (zone_nr_lru_pages * ratio) pages,
                        until enough pages reclaimed && ...

        if enough pages reclaimed
                break;
```

balanced aging: NUMA nodes

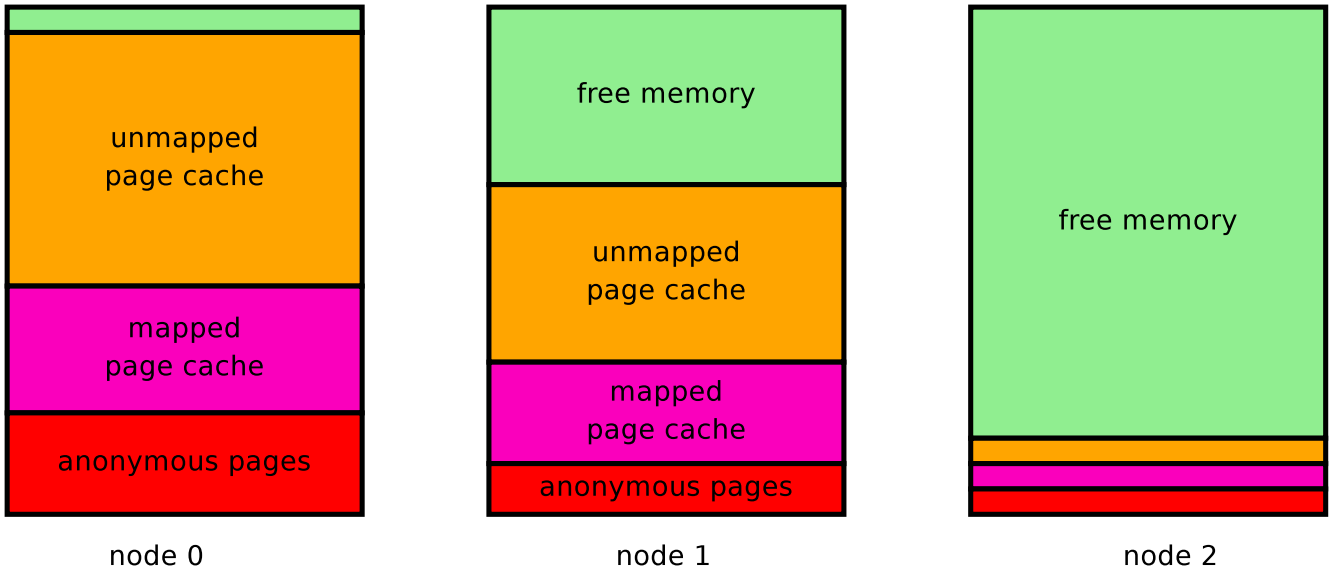
Primary goals:

- **avoid off node allocation, or**
- **spread NUMA interlink traffic evenly**

- **no explicit efforts for balanced scan of NUMA nodes**
- **aging rates are heavily impacted by page allocation policy**
- **NUMA memory policies: interleave, bind, preferred, default**

zone reclaim in a NUMA system

memory is exhausted locally => off node allocation of anonymous pages



source: *Local and Remote Memory: Memory in a Linux/NUMA System*, Christoph Lameter

zone reclaim:

start reclaiming (unmapped) pages if a zone's free memory is low

direct reclaim in a busy system

kswapd reclaim kswapd frees memory in the background, so that light page allocations can complete instantly;

direct reclaim when kswapd cannot keep up with the allocation rate, applications try to free memory for themselves.

- **helps throttle heavy allocation apps**
- **NUMA: focus reclaim on allocation heavy nodes**
- **may isolate too many LRU pages on fork bombs**

reclaim scalability: unevictable list

to avoiding scanning the unreclaimable pages:

- **pages pinned by** `mlock()` **or** `SHM_LOCK`
- **memory backed pages (ramfs, ramdisk)**

reclaim scalability: mapped pages

- **we used to avoid scanning and evicting mapped pages**
  - mapped pages are more costly to scan/reclaim/refault
  - mapped pages are more valuable to cache in typical desktop
- **now tend to de-activate mapped pages unconditionally**
  - long round trip time (eg. 1TB mem)
  - every page have `pte_young`
- **`VM_EXEC` mapped file pages are still protected as usual (for better desktop responsiveness)**

reclaim efficiency: lumpy reclaim

rational excessive reclaims for high-order page allocation

solution lumpy reclaim for high-order page:

1. **the first page is taken from LRU list (as before),**
2. **try to reclaim the (physically) surrounding pages (to form a high-order block)**

problem distort LRU; randomly punching holes in cached files

split file/anon LRU lists

`file LRU` file backed pages

`anon LRU` mem/swap backed pages

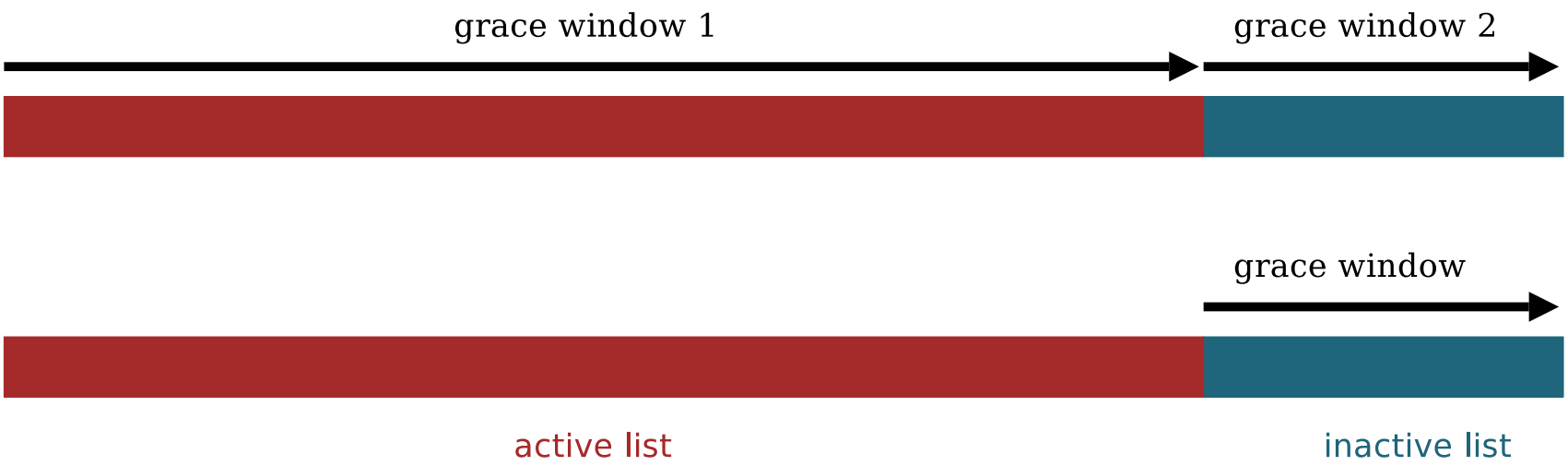
- **anonymous pages**
- **tmpfs pages**

`scan ratio` $\propto$ *recent_scanned* : *recent_referenced*
(more referenced, less scanned)

- **protects anonymous pages**
- **improves reclaim efficiency**

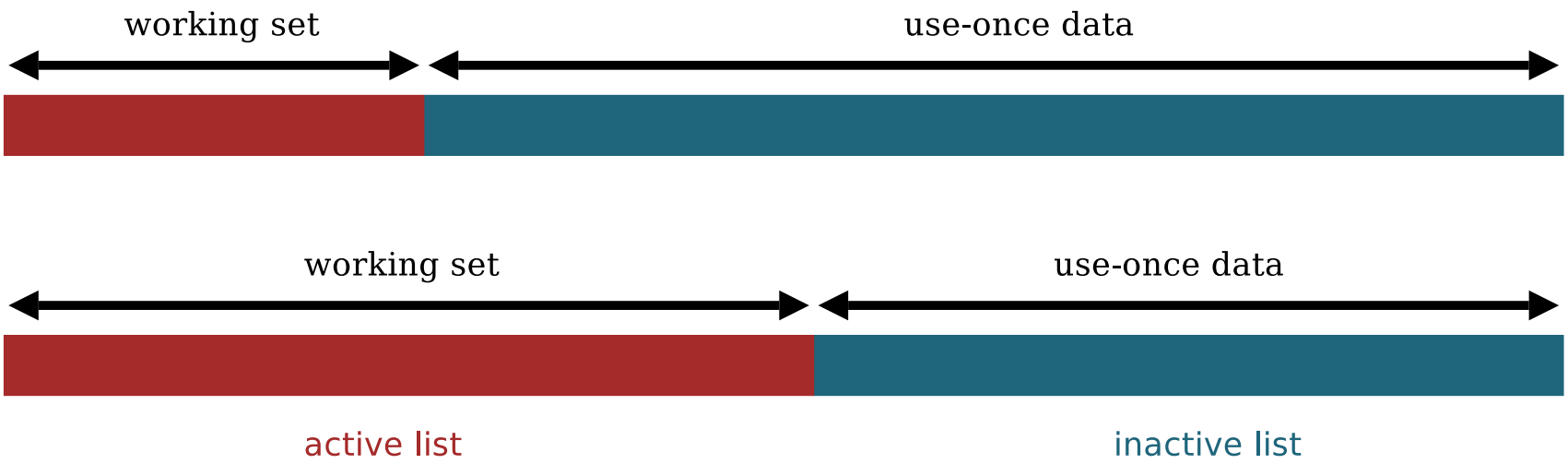
anon LRU active:inactive ratio

- **balanced scan in normal**
- **shrink active list iif inactive list is low**
  - inactive list serves as the only grace window for active references
  - note that when inactive list => 0, LRU reduce to FIFO
- `active:inactive => sqrt(10 * GB)`



file LRU active:inactive ratio

- **balanced scan in normal**
- **shrink active list only when inactive list is low**
  - active list: working set
  - inactive list: use-once data
  - protect working set from being flushed by use-once data
- `active:inactive => 1:1`



advanced page replacement

- **current logic: balanced but not optimal**
- **obvious optimization: streaming IO**
  - drop behind
  - use frequency instead of recency
- **available solutions**
  - CLOCK-Pro, CART, CAR, ARC, LIRS etc.
- **problems**
  - complexity
  - possible regressions

readings and resources

- **LinuxMM**
  `http://linux-mm.org/`
- **linux-mm mailing list**
  `http://marc.info/?l=linux-mm`
- **linux kernel source code**
  `/usr/src/linux $ git log mm`

book **Understanding The Linux Virtual Memory Manager**
`http://www.skynet.ie/~mel/projects/vm/guide/pdf/understand.pdf`

Thank you!