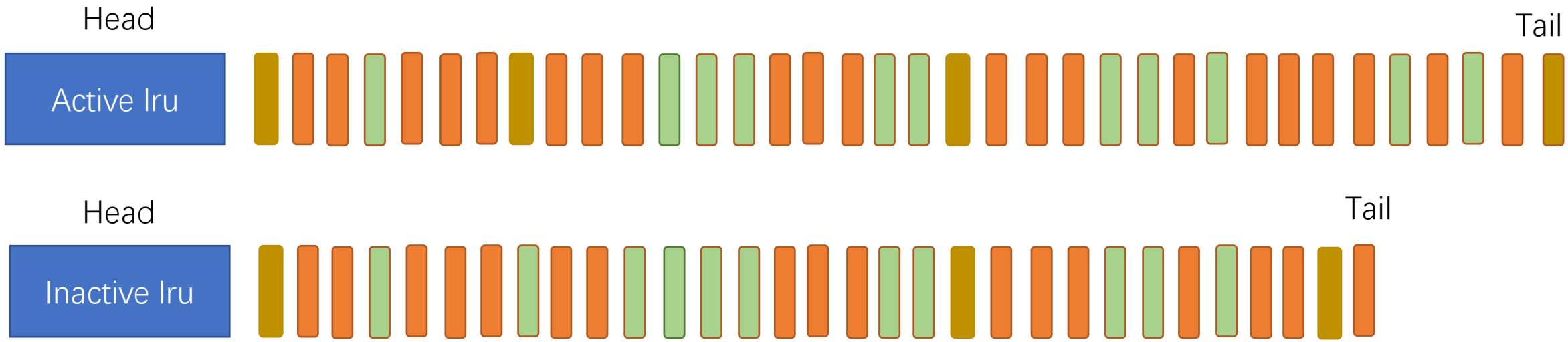
Workingset Based Memory Reclaim

DiDi Chuxing Kernel Team

Speaker: Xie Yongmei



Known issue

- **The time to trigger kswapd is late**
- **Lru aging is less frequent**

About Reclaim

- Global reclaim
  - Min/low/high
  - /proc interface
    - min_free_kbytes
    - watermark_scale_factor
    - watermark_boost_factor
- Cgroup reclaim

- The reality is
  - In K8S production, we saw lots of direct relaim from“sar –B”

Why Kswapd Not Catch Up

- Single thread vs multiple thread
  - Kswapd cannot produce enough free memory in a short period to handle burst memory allocation by multiple threads/processes
  - Ex.: new rich container created
    - the application process + all kinds of agents
- Direct reclaim does reclaime more memory than kswapd
  - shrink_lruvec
- What if kswapd produces enough memory
  - Ultimately, but not immediately

What If Trigger Kswap Ahead of Time

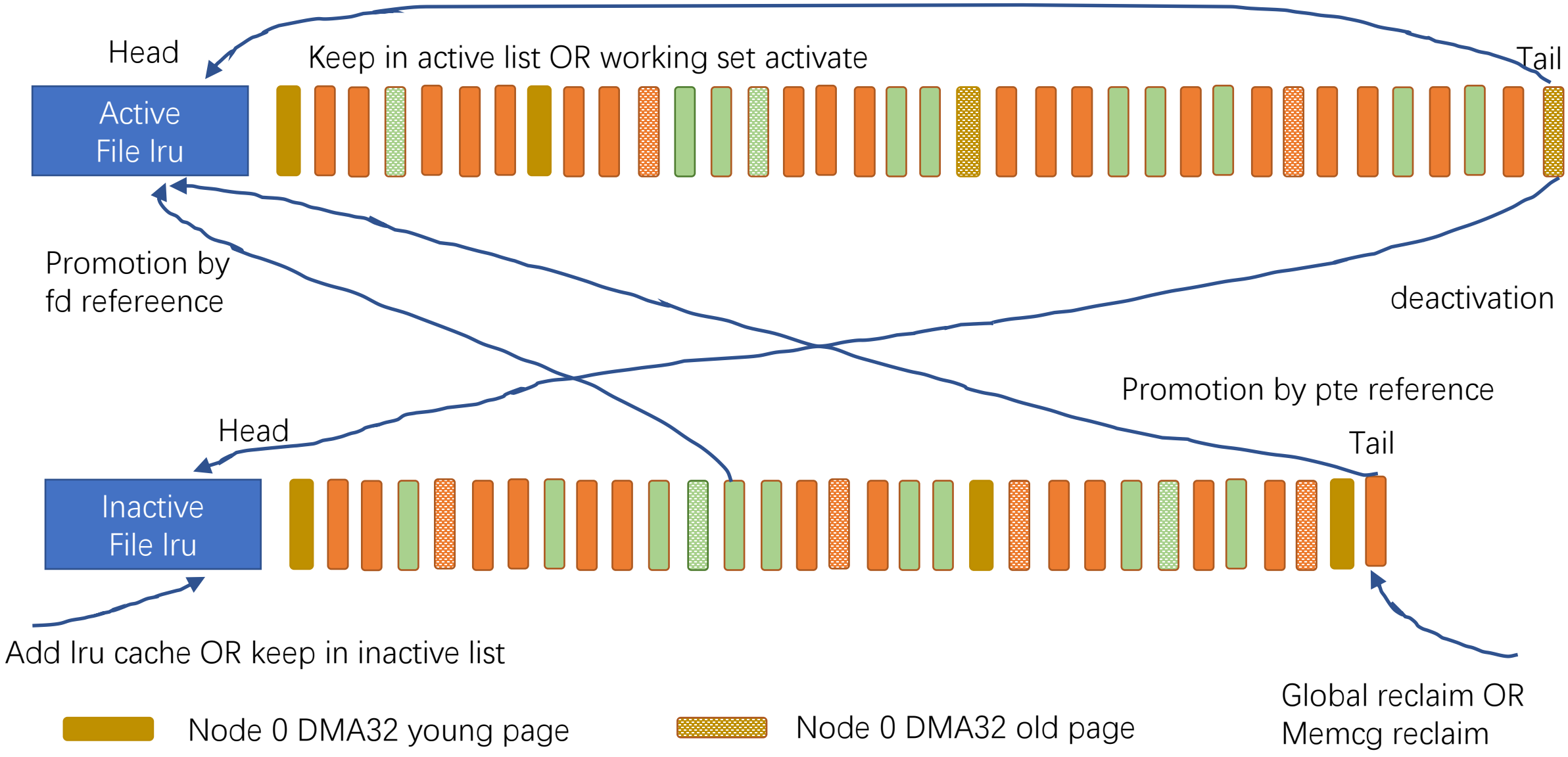
- It does make sense
- But active/inactive ratio is changed since commit # 59dc76b
  - “mm: vmscan: reduce size of inactive file list”
- Based on working set, evicted page can be promoted directly with regards to refault distance, so Rik van Riel utilizes the side effect of workingset activation to reduce inactive lru.
- Advantage
  - Friendly to filter out streaming I/O
- Disadvantage
  - Short window for candidate of activation

Current Situation

- If inactive tail has enough cold pages, we could safely reclaim them in proactive way. Otherwise, it costs to build new working set.
- How to tell the coldness of pages
  - Aging
    - Explicitly aging for anon pages
    - mark_page_accessed for fd access
    - Second chance for mapped pages
  - Aging happens
    - Shrink_inactive_list: second chance check
    - Shrink_active_list: deactivate page
    - Kswapd age anon pages

Our Solution

- Periodically aging thread
  - Age file pages in a fix duration
  - Based on idle page tracking:
    - use page_is_idle() to detect whether page was referenced since last round scan
- Periodically proactive reclaim thread
  - If page is cold enough, reclaim it from lru list
  - Grab lru lock to maintain lru list and add 1 extra refcnt on page
  - Grab page lock to reclaim it
    - Remove page refcnt
    - Remove the page from VFS page cache (AKA radix tree)
    - Uncharge from memcg
    - Add into pcp free list
  - Similar to shrink_inactive_list, but reclaim pages from several memcgs


Head
Keep in active list OR working set activate
Tail
Active
File lru
Promotion by
fd refereence
deactivation
Promotion by pte reference
Head
Tail
Inactive
File lru
Add lru cache OR keep in inactive list
Global reclaim OR
Memcg reclaim
Node 0 DMA32 young page
Node 0 DMA32 old page
Node 0 Normal young page
Node 0 Normal old page
Node 1 Normal young page
Node 1 Normal old page

High Level View

- Original lru is FIFO
- With this patch, lru could be handled as pseudo double linked list
  - Add in head
  - Delete from any position

Overhead & Benefit

- Overhead
  - About 1% cpu for aging
  - About 2% cpu for evict
- Benefit
  - Friendly to heavy page cache scenario
  - Direct reclaim almost disappears
- Limitation
  - Not work on mapped page, because rmap is expensive
  - This patchset is designed for 4.18 kernel

Future work

- Redesign solution for upstream
  - Memcg page aging & order
  - Debugfs interface to trigger proactive reclaim
- Will be ready in Nov.

NEXT

- What is next from DiDi Kernel team
  - More contributions to upstream
  - More effort on cloud native containers
- Stay tuned
- BTW: we are hiring

内核技术交流群