

What Is Linux Doing With My RAM?

Vlastimil Babka, SUSE Labs

Open Source Summit EU, Prague 2026

The textbook command: free (1)

- ```
  > free -m
                 total        used        free      shared  buff/cache   available
  Mem:           31372       10740        8444         596       13234       20631
  Swap:           2454           0        2454
  ```
- `used + free != total ? (19184 < 31372)` – what's going on here ?
- `used + available == total ? (31371 ~= 31372)` – this looks more promising
- Linux memory management principle: unused memory is wasted memory
  - some memory will be allocated directly by processes (`malloc()`, "`new`" → `mmap()`)
  - kernel will also cache all accessed file contents in memory, as long as it fits
  - once free memory is (almost) depleted, the cache is relatively easy to discard
  - *used* tries to estimate how much is occupied that can't easily be discarded
    - otherwise the number wouldn't be really useful (always close to total)
  ```

Kernel’s accounting view: /proc/meminfo

```
MemTotal:       32125696 kB
MemFree:         8627220 kB
MemAvailable:   21107232 kB
Buffers:          258820 kB
Cached:         12902952 kB
SwapCached:            0 kB
Active:          5870672 kB
Inactive:       16277940 kB
Active(anon):       8120 kB
Inactive(anon):  9589756 kB
Active(file):    5862552 kB
Inactive(file):  6688184 kB
Unevictable:          80 kB
Mlocked:              80 kB
SwapTotal:       2513056 kB
SwapFree:        2513056 kB
Zswap:                 0 kB
Zswapped:              0 kB
Dirty:               844 kB
Writeback:           196 kB
```

```
AnonPages:       8922944 kB
Mapped:          1683592 kB
Shmem:            611016 kB
KReclaimable:     390056 kB
Slab:             613748 kB
SReclaimable:     390056 kB
SUnreclaim:       223692 kB
KernelStack:       35568 kB
PageTables:        82500 kB
SecPageTables:      3248 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    18575904 kB
Committed_AS:   21267524 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       84632 kB
VmallocChunk:          0 kB
Percpu:             7680 kB
HardwareCorrupted:     0 kB
```

```
AnonHugePages:   1363968 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:   1062912 kB
FilePmdMapped:    684032 kB
CmaTotal:              0 kB
CmaFree:               0 kB
Unaccepted:            0 kB
Balloon:               0 kB
GPUActive:        149220 kB
GPUReclaim:       128140 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      894136 kB
DirectMap2M:    25239552 kB
DirectMap1G:     7340032 kB
```

```
MemTotal:       32125696 kB    SwapFree:        2513056 kB    CommitLimit:    18575904 kB
MemFree:         8627220 kB    Zswap:                 0 kB    Committed_AS:   21267524 kB
MemAvailable:   21107232 kB    Zswapped:              0 kB    VmallocUsed:       84632 kB
Buffers:          258820 kB    Dirty:               844 kB    Percpu:             7680 kB
Cached:         12902952 kB    Writeback:           196 kB    AnonHugePages:   1363968 kB
SwapCached:            0 kB    AnonPages:       8922944 kB    ShmemHugePages:        0 kB
Active:          5870672 kB    Mapped:          1683592 kB    ShmemPmdMapped:        0 kB
Inactive:       16277940 kB    Shmem:            611016 kB    FileHugePages:   1062912 kB
Active(anon):       8120 kB    KReclaimable:     390056 kB    FilePmdMapped:    684032 kB
Inactive(anon):  9589756 kB    Slab:             613748 kB    CmaTotal:              0 kB
Active(file):    5862552 kB    SReclaimable:     390056 kB    CmaFree:               0 kB
Inactive(file):  6688184 kB    SUnreclaim:       223692 kB    Balloon:               0 kB
Unevictable:           80 kB    KernelStack:       35568 kB    GPUActive:        149220 kB
Mlocked:               80 kB    PageTables:        82500 kB    GPUReclaim:       128140 kB
SwapTotal:       2513056 kB    SecPageTables:      3248 kB    Hugetlb:               0 kB
```

```
(free -m)       total    used    free    shared  buff/cache   available
Mem:            31372   10740    8444       596       13234       20631
Swap:            2454       0    2454
```

MemTotal:	32125696 kB	SwapFree:	2513056 kB	CommitLimit:	18575904 kB
MemFree:	8627220 kB	Zswap:	0 kB	Committed_AS:	21267524 kB
MemAvailable:	21107232 kB	Zswapped:	0 kB	VmallocUsed:	84632 kB
Buffers:	258820 kB	Dirty:	844 kB	Percpu:	7680 kB
Cached:	12902952 kB	Writeback:	196 kB	AnonHugePages:	1363968 kB
SwapCached:	0 kB	AnonPages:	8922944 kB	ShmemHugePages:	0 kB
Active:	5870672 kB	Mapped:	1683592 kB	ShmemPmdMapped:	0 kB
Inactive:	16277940 kB	Shmem:	611016 kB	FileHugePages:	1062912 kB
Active(anon):	8120 kB	KReclaimable:	390056 kB	FilePmdMapped:	684032 kB
Inactive(anon):	9589756 kB	Slab:	613748 kB	CmaTotal:	0 kB
Active(file):	5862552 kB	SReclaimable:	390056 kB	CmaFree:	0 kB
Inactive(file):	6688184 kB	SUnreclaim:	223692 kB	Balloon:	0 kB
Unevictable:	80 kB	KernelStack:	35568 kB	GPUActive:	149220 kB
Mlocked:	80 kB	PageTables:	82500 kB	GPUReclaim:	128140 kB
SwapTotal:	2513056 kB	SecPageTables:	3248 kB	Hugetlb:	0 kB

(free -m)	total	used	free	shared	buff/cache	available
Mem:	31372	10740	8444	596	13234	20631
Swap:	2454	0	2454			

Legend: Kernel Userspace *Redundant (Subset of another)* Not consumed memory

MemTotal:	32125696 kB	SwapFree:	2513056 kB	CommitLimit:	18575904 kB
MemFree:	8627220 kB	Zswap:	0 kB	Committed_AS:	21267524 kB
MemAvailable:	21107232 kB	Zswapped:	0 kB	VmallocUsed:	84632 kB
Buffers:	258820 kB	*Dirty:*	*844 kB*	Percpu:	7680 kB
Cached:	12902952 kB	*Writeback:*	*196 kB*	*AnonHugePages:*	*1363968 kB*
SwapCached:	*0 kB*	AnonPages:	8922944 kB	*ShmemHugePages:*	*0 kB*
Active:	*5870672 kB*	*Mapped:*	*1683592 kB*	*ShmemPmdMapped:*	*0 kB*
Inactive:	*16277940 kB*	*Shmem:*	*611016 kB*	*FileHugePages:*	*1062912 kB*
Active(anon):	*8120 kB*	*KReclaimable:*	*390056 kB*	*FilePmdMapped:*	*684032 kB*
Inactive(anon):	*9589756 kB*	Slab:	613748 kB	CmaTotal:	0 kB
Active(file):	*5862552 kB*	*SReclaimable:*	*390056 kB*	CmaFree:	0 kB
Inactive(file):	*6688184 kB*	*SUnreclaim:*	*223692 kB*	Balloon:	0 kB
Unevictable:	*80 kB*	KernelStack:	35568 kB	GPUActive:	149220 kB
Mlocked:	*80 kB*	PageTables:	82500 kB	GPUReclaim:	128140 kB
SwapTotal:	2513056 kB	SecPageTables:	3248 kB	Hugetlb:	0 kB

Anonymous memory vs page cache

- **Anonymous pages** – not backed by files, main counter: **`AnonPages`**
  - allocated mainly by faulting in `mmap(MAP_PRIVATE | MAP_ANONYMOUS)` areas
  - lifetime tied to the process(es) mapping them, can become shared during `fork()`
  - cannot be simply written back to a file and discarded, only swapped out to swap
- **Page cache** – pages backed by files on a FS, or a block device
  - main counter: **`Cached - Shmem`** (FS), **`Buffers`** (block device – FS metadata)
  - allocated by accessing files that are `mmap()`ed, or read/written to in other ways
  - lifetime independent of processes, can be discarded once not dirty, read back when needed again
- **Shared memory** (shmem, tmpfs) – a hybrid between the above – main counter: **`Shmem`**
  - a RAM only filesystem (`/dev/shm`) or `shmget()` or `mmap(MAP_SHARED | MAP_ANONYMOUS)`
  - behaves a lot like page cache – lifetime can independent of processes
  - but there are no persistent files backing it – must be swapped out like anonymous pages

Active and Inactive, (anon) and (file)

- Userspace pages are sorted by their “hotness” on least recently used (LRU) lists
  - divided to anon and file due to their different characteristics
  - divided to active and inactive parts for reclaim implementation purposes
    - this does not imply that the inactive part is just sitting there unacessed (cold)
- The related counters are an orthogonal way to see how much memory is used by the page cache and anonymous pages
  - With a twist: *Shmem* is in page cache (*Cached*) but on *anon* LRU (because it’s swap backed)
  - Unevictable/Mlocked may account both page cache and anonymous pages
- `AnonPages + Shmem = 8922944 + 611016 = 9533960 kB`
  `Active(anon) + Inactive(anon) = 9589756 + 8120 = 9597876 kB`
- `Cached + Buffers – Shmem = 12902952 + 258820 – 611016 = 12550756 kB`
  `Active(file) + Inactive(file) = 5862552 + 6688184 = 12550736 kB`
- Significant differences (e.g. larger active+inactive sum) may be due to a memory leak

Legend: Kernel Userspace *Redundant* Not consumed memory **Sums to MemTotal**

MemTotal:	32125696 kB	SwapFree:	2513056 kB	CommitLimit:	18575904 kB
MemFree:	**8627220 kB**	**Zswap:**	**0 kB**	Committed_AS:	21267524 kB
MemAvailable:	21107232 kB	Zswapped:	0 kB	**VmallocUsed:**	**84632 kB**
Buffers:	**258820 kB**	*Dirty:*	*844 kB*	**Percpu:**	**7680 kB**
Cached:	**12902952 kB**	*Writeback:*	*196 kB*	*AnonHugePages:*	*1363968 kB*
SwapCached:	*0 kB*	**AnonPages:**	**8922944 kB**	*ShmemHugePages:*	*0 kB*
Active:	*5870672 kB*	*Mapped:*	*1683592 kB*	*ShmemPmdMapped:*	*0 kB*
Inactive:	*16277940 kB*	*Shmem:*	*611016 kB*	*FileHugePages:*	*1062912 kB*
Active(anon):	*8120 kB*	*KReclaimable:*	*390056 kB*	*FilePmdMapped:*	*684032 kB*
Inactive(anon):	*9589756 kB*	**Slab:**	**613748 kB**	CmaTotal:	0 kB
Active(file):	*5862552 kB*	*SReclaimable:*	*390056 kB*	CmaFree:	0 kB
Inactive(file):	*6688184 kB*	*SUnreclaim:*	*223692 kB*	Balloon:	0 kB
Unevictable:	*80 kB*	**KernelStack:**	**35568 kB**	**GPUActive:**	**149220 kB**
Mlocked:	*80 kB*	**PageTables:**	**82500 kB**	**GPUReclaim:**	**128140 kB**
SwapTotal:	2513056 kB	**SecPageTables:**	**3248 kB**	**Hugetlb:**	**0 kB**

So which fields sum to MemTotal?

- `MemFree + Buffers + Cached + Zswap + AnonPages + Slab + KernelStack + PageTables + SecPageTables + VmallocUsed + Percpu + GPUActive + GPUReclaim + Hugetlb`
  `= 31816672 kB`
- `MemTotal = 32125696 kB`
  - `300 MB` not accounted for by any `/proc/meminfo` counter
  - Might be allocations by drivers, filesystems, networking… some expose the info elsewhere
  - `Unaccounted` field proposed as RFC in 2016 – mixed feedback
- So what exactly is MemAvailable then?
  - An estimate of how much is free or reclaimable (except anonymous)
  - `MemFree` (minus some reserve)
    `+ Active(file) + Inactive(file)` (minus some reserve)
    `+ KReclaimable` (minus some reserve)
    `= 21568012 kB` (reserves not subtracted)
  - `MemAvailable: 21107232 kB`

What to make of memory usage ?

- "`MemFree`" isn't a useful value, will converge to near-zero eventually on most systems
- "`MemAvailable`" and "*used*" (reported by `free (1)` as *total* minus *available*) make much more sense to report the system state at any given moment
- During the workload's steady operation, the exact values are rarely important
- Values may change on distro upgrades due to kernel/glibc implementation details
  - kernel/user memory usage – changes e.g. due to scalability, sizing of structures
  - active/inactive LRU sizes – reclaim algorithm may change
- Limits for memcgs / alerts determined on an older version may need updating
  - memcg kmem consumption – new kinds of kernel objects may get accounted

What to make of memory usage ?

- What matters more than exact values - trends over time or dynamic changes
  - a particular counter keeps growing over time - possible memory leak?
  - “available” memory exhausted, tiny page cache remaining, thrashing or OOM kills?
  - is this due to to kernel bug, userspace bug or my workload is simply too big?
- Looking at `/proc/vmstat` event counters can tell much more information and often explain the changes in memory counters
  - Shows what actions the kernel is doing to keep up with the memory demand

`/proc/vmstat` at a glance

- `/proc/meminfo` is curated and formatted for humans
  - kB, names, ordering, values calculated from multiple counters
- `/proc/vmstat` is more suitable for script consumption
  - Mostly a dump (202 lines) of simple counter names in their "`enum`" order and their raw values
- Some counters ("**`nr_`**") are for state, can increase and decrease, many used also in `/proc/meminfo`
  - Others (not "**`nr_`**") count events (since boot) and thus can only increment
  - Naming is not 100% consistent and there are exceptions in both directions
- Some are internally tracked per NUMA node or even zone (DMA, DMA32, Normal…) and summed up for `/proc/vmstat`
  - Node-specific subsets in `/sys/bus/node/devices/nodeX/vmstat`
  - Per-zone breakdown in `/proc/zoneinfo`

Events related to page faults

- **`pgfault`** – page fault was handled (for whatever reason – mapping doesn't exist, or is read-only and userspace wants to write), unit is per fault regardless of folio size
- **`pgmajfault`** – disk I/O (read from swap or file) was necessary to finish the page fault
  - readahead can prevent it even for a first fault for given part of a file
- **`pswpin`** – reads from swap (unit is base page)
  - **`swap_ra`** – swap folio readahead (unit is per folio), **`swap_ra_hit`** – first fault (thus minor) on a readahead swap folio
- **`pgpgin`** – reads via the block layer, the unit is kilobytes, includes other sources than page faults
- **`pgreuse`** – fault on a write-protected page table entry that can simply make the entry writable using the already mapped page
  - either shared file mapping or the last of parent/child writing to a copy on write (CoW) page

Events related to memory reclaim

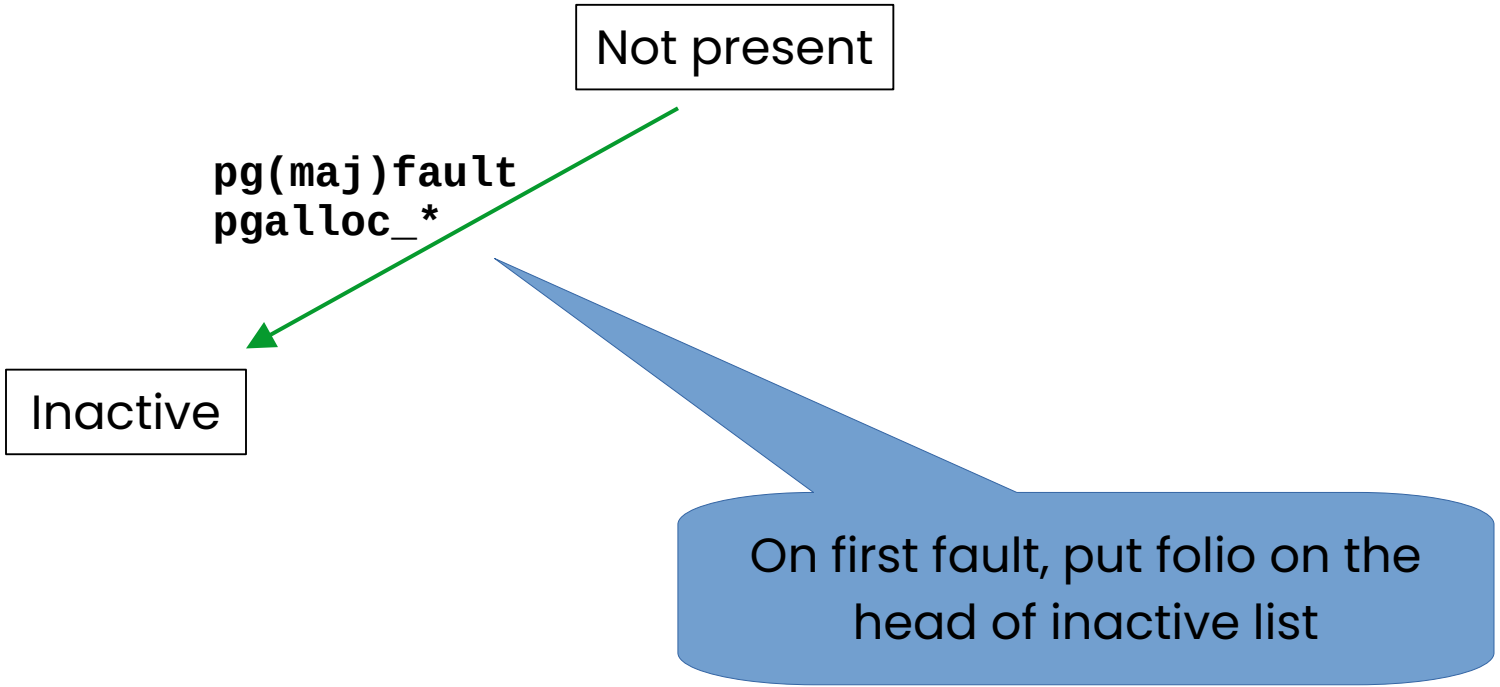
- Folios are organized in LRU lists, split into file/anon and active/inactive lists, more recently accessed folios closer to head of the list
  - The sizing of the active/inactive split is artificial, meant to hold most folios on the active list and focus reclaim on the smaller inactive list
    - Inactive ratio grows with square root of size of the LRU lists
    - With 1GB size, active:inactive is 3:1, with 10GB it's 10:1 etc…
- Reclaim calculates how many folios from each list to scan
  - To maintain the ratio, to account for `vm.swappiness`, recent reclaim "cost"…
  - Then scans the lists from the tail (where the coldest pages should be)
- The related events often distinguish **`_kswapd`**/**`_direct`**/**`_khugepaged`** (for deferred THP collapsing)/**`_proactive`** (/sys/…/nodeX/reclaim) and **`_anon`**/**`_file`**

Events related to memory reclaim

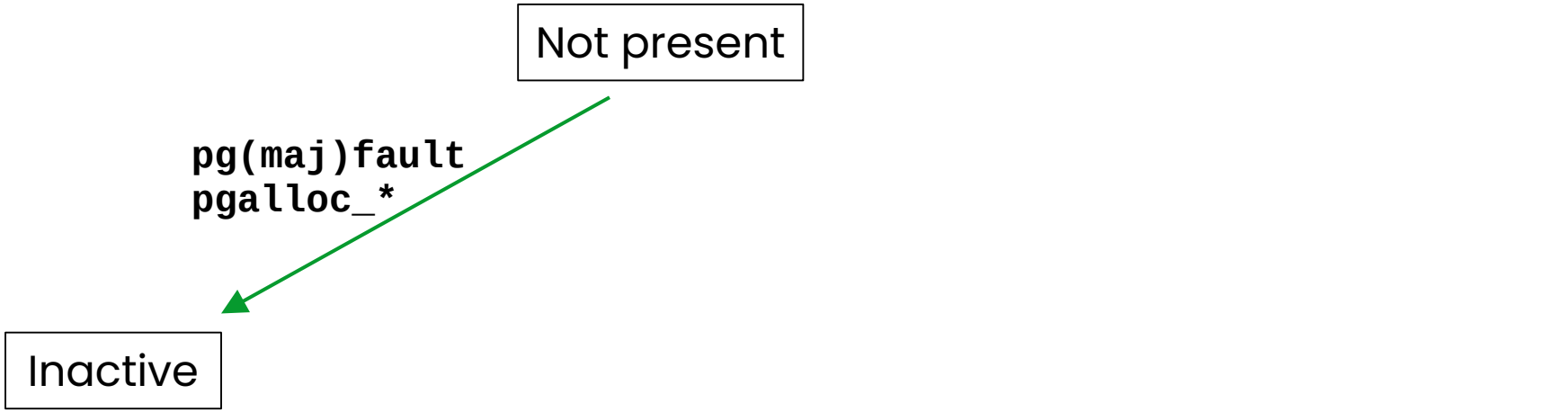
Not present

Events related to memory reclaim

promote

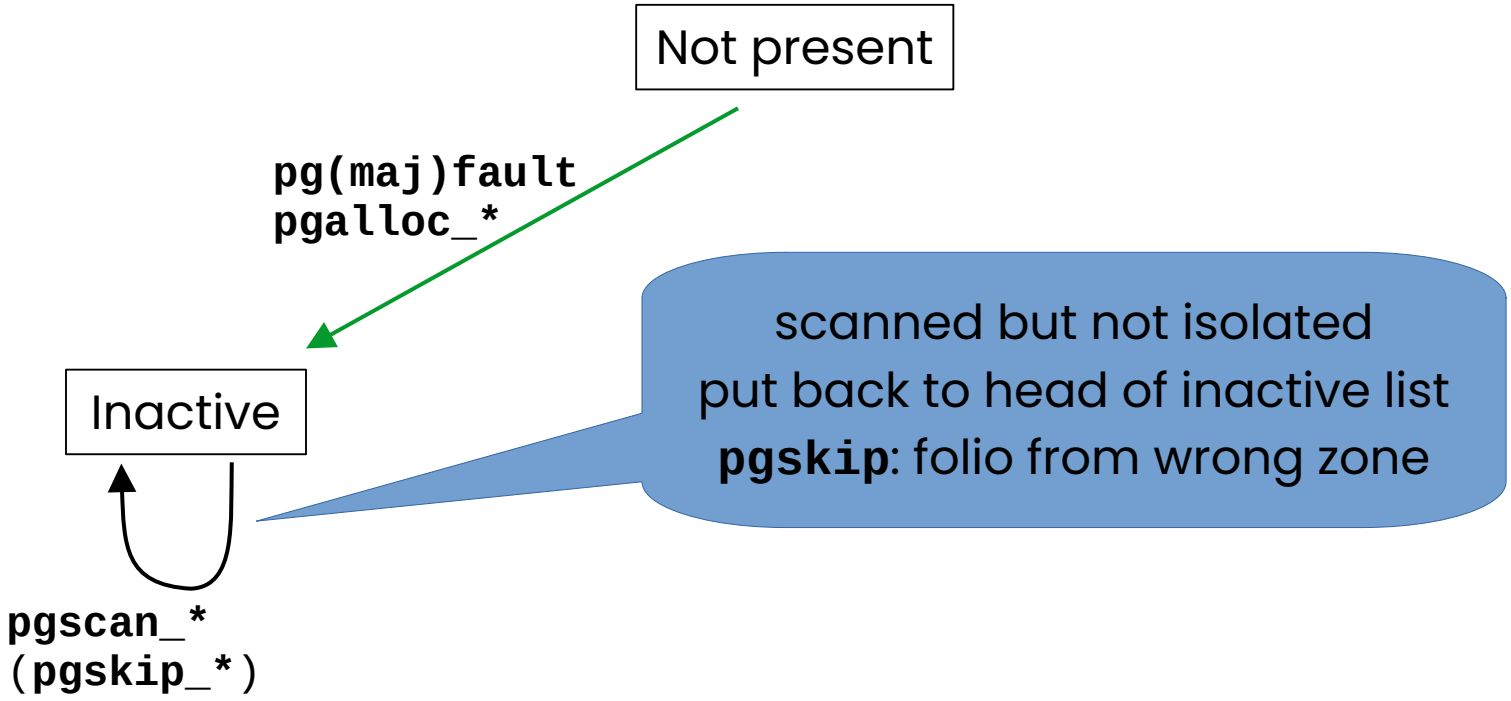


Events related to memory reclaim

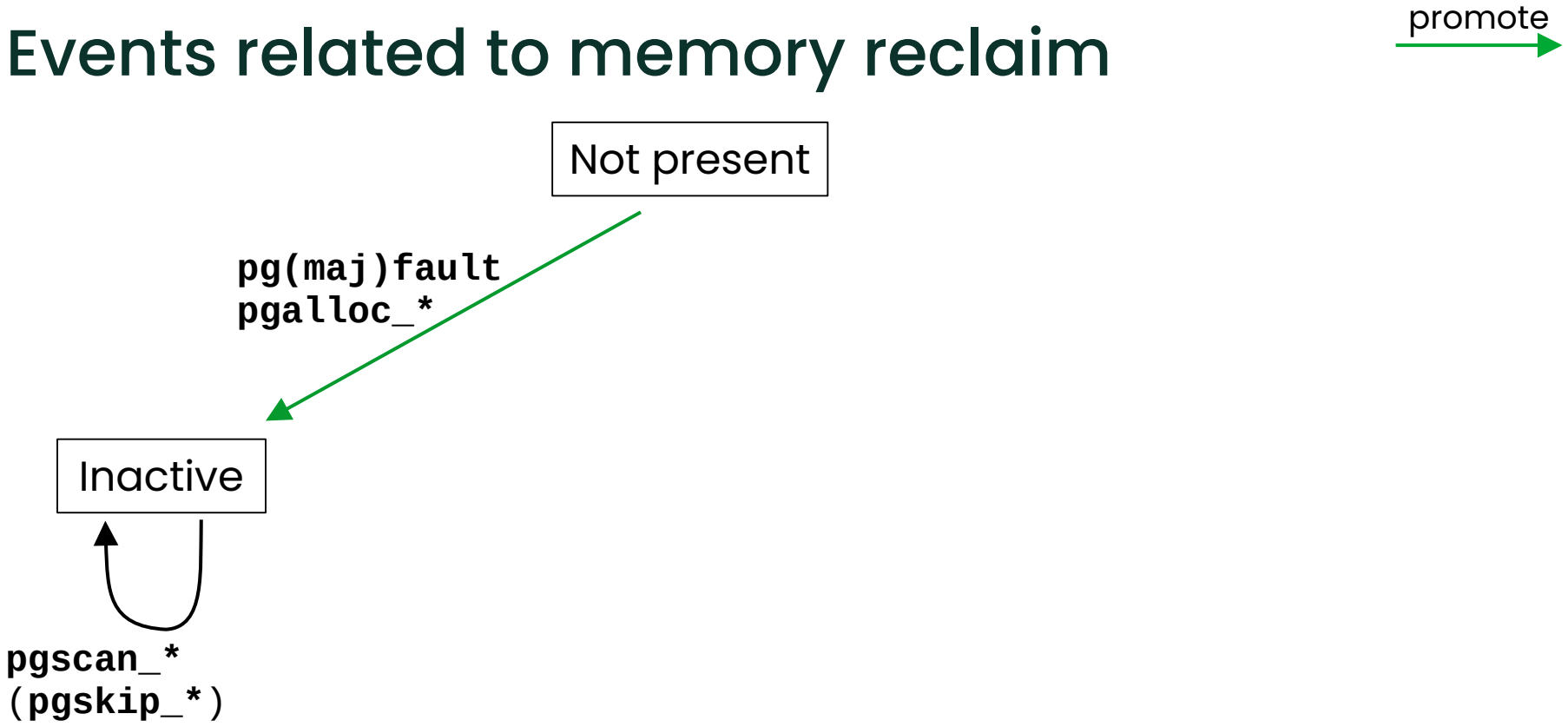


Events related to memory reclaim

promote

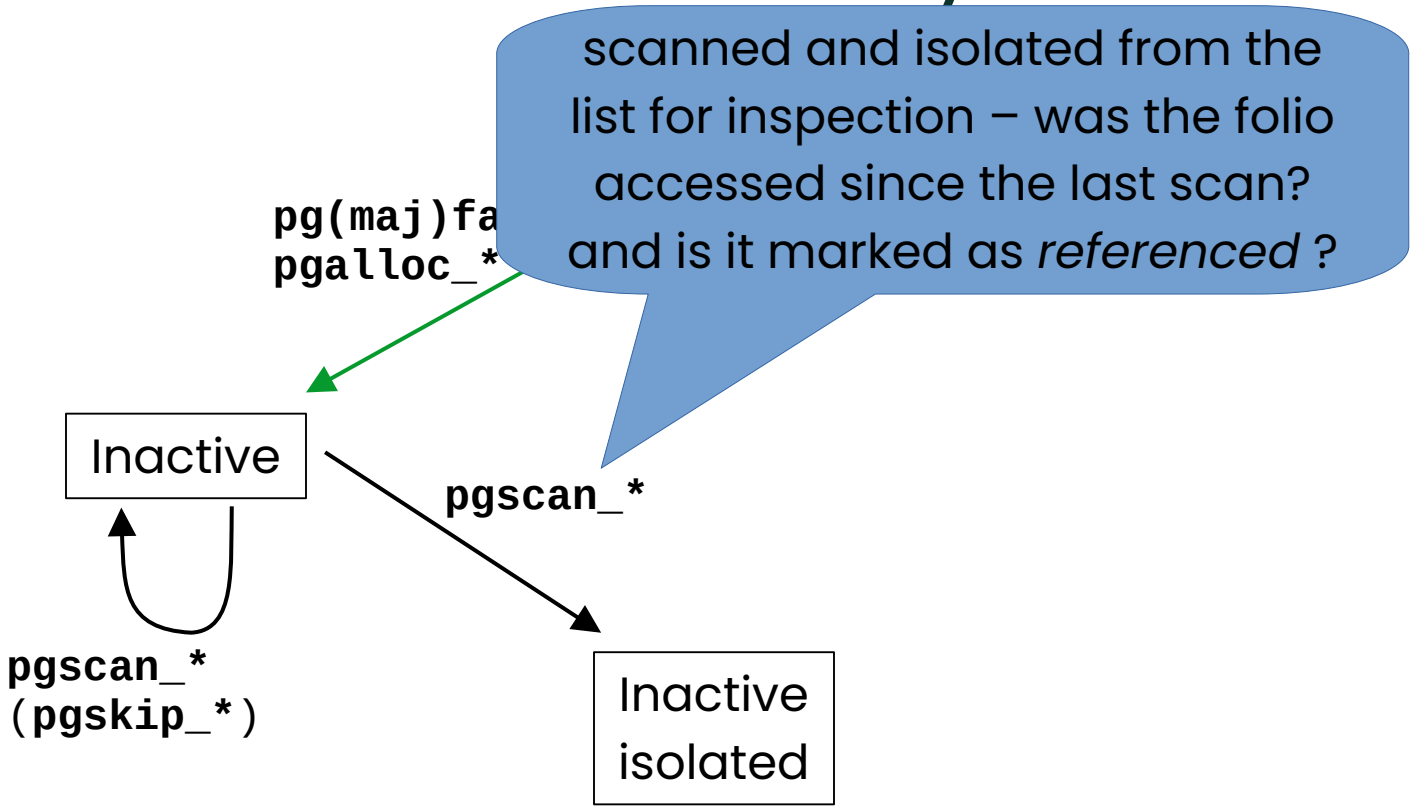


Events related to memory reclaim



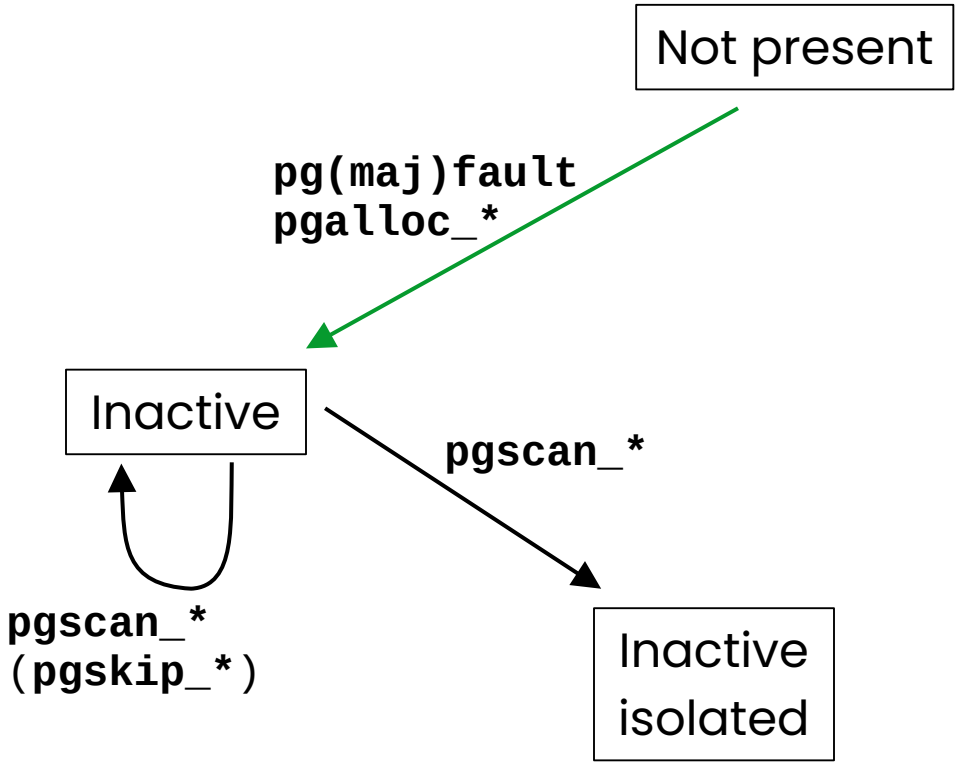
Events related to memory reclaim

promote

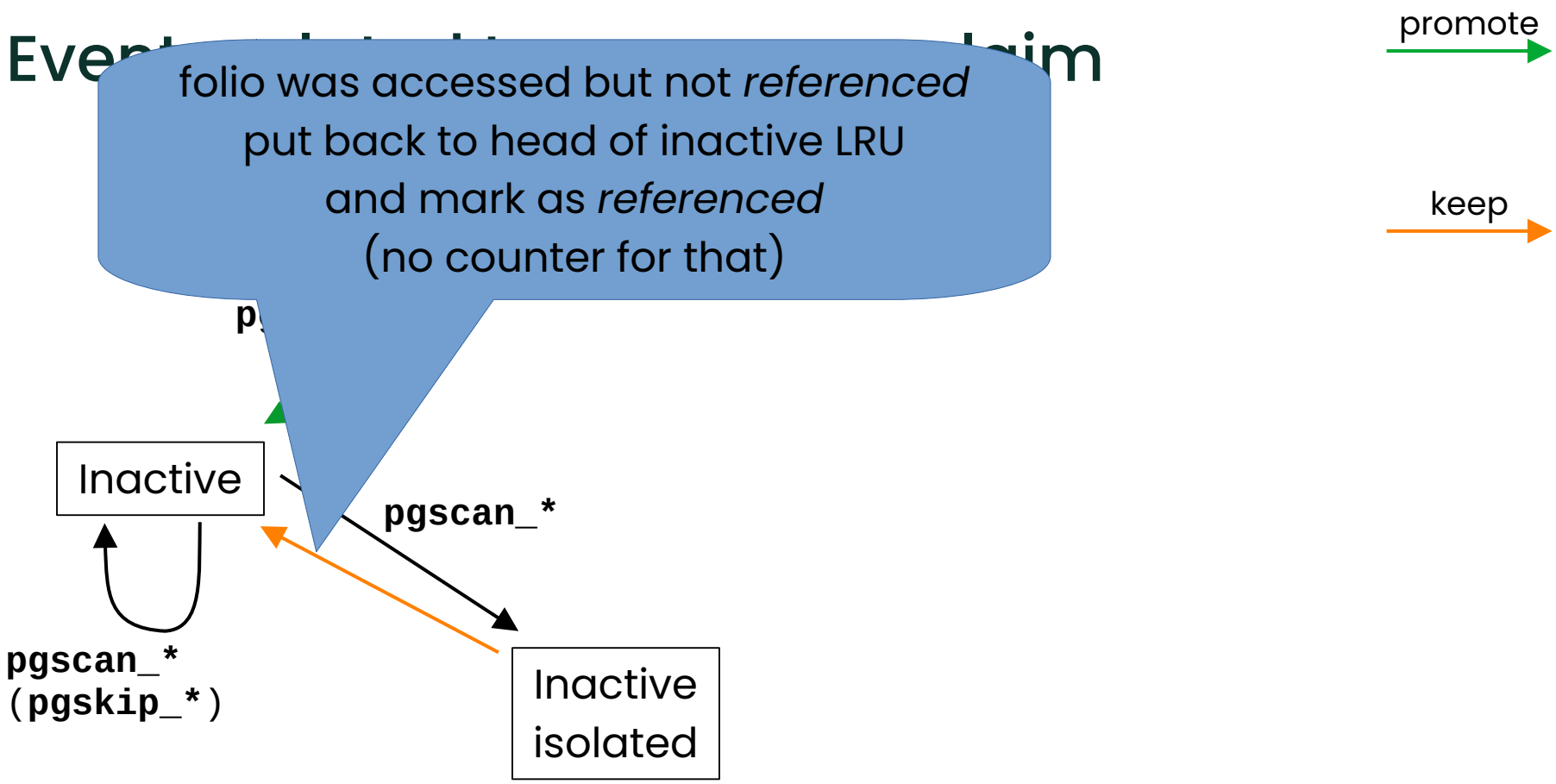


Events related to memory reclaim

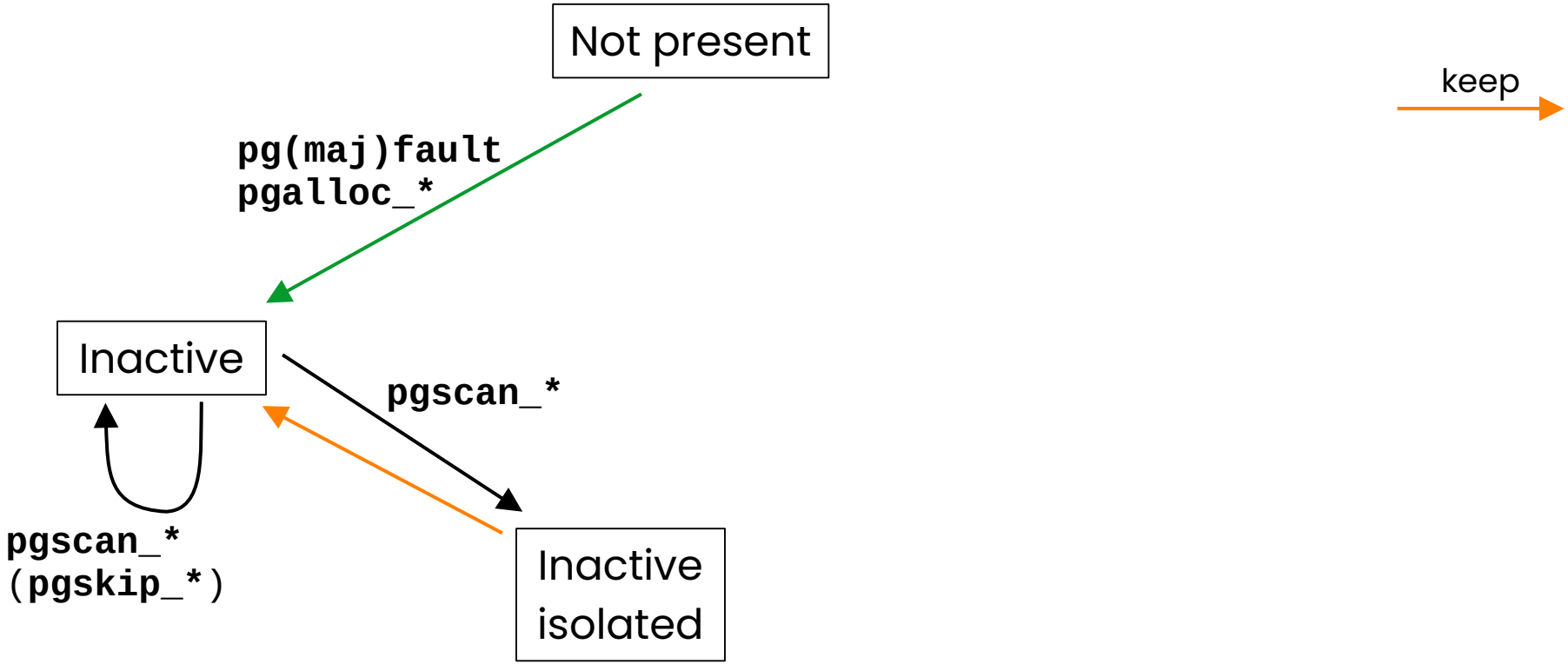
promote



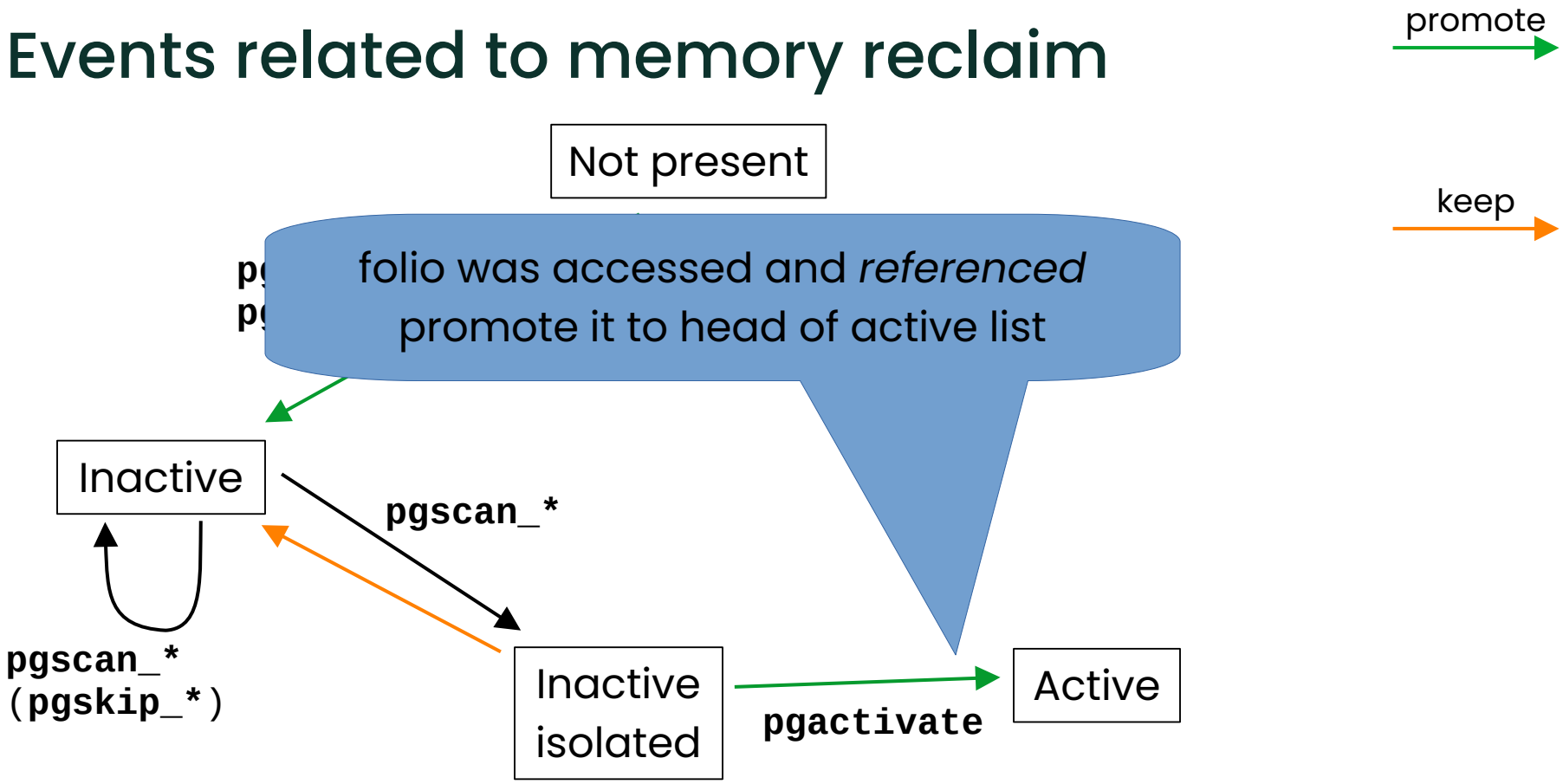
Even[illegible]aim


promote
keep
folio was accessed but not *referenced*
put back to head of inactive LRU
and mark as *referenced*
(no counter for that)
Inactive
pgscan_*
pgscan_*
(pgskip_*)
Inactive
isolated

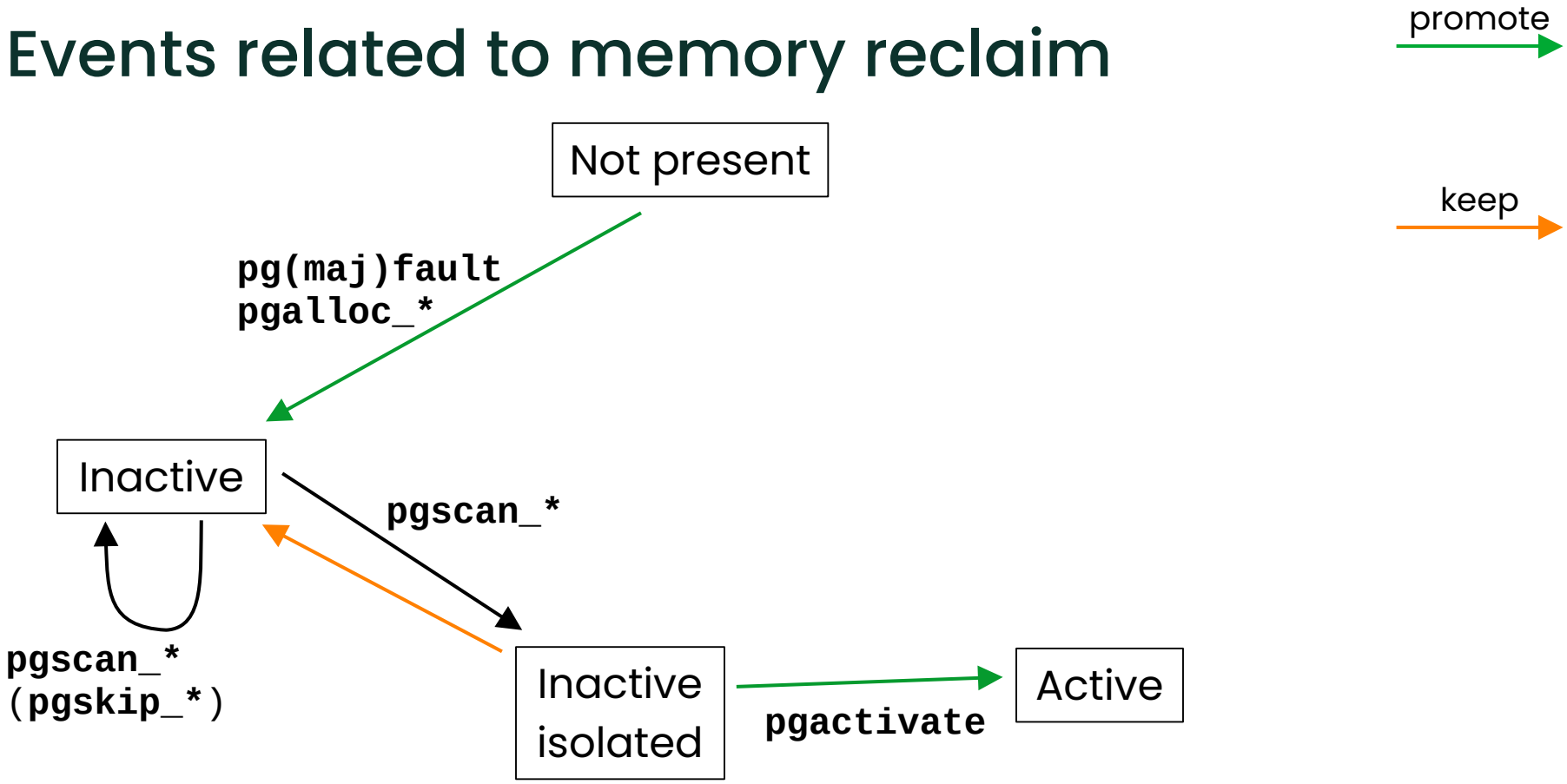
Events related to memory reclaim



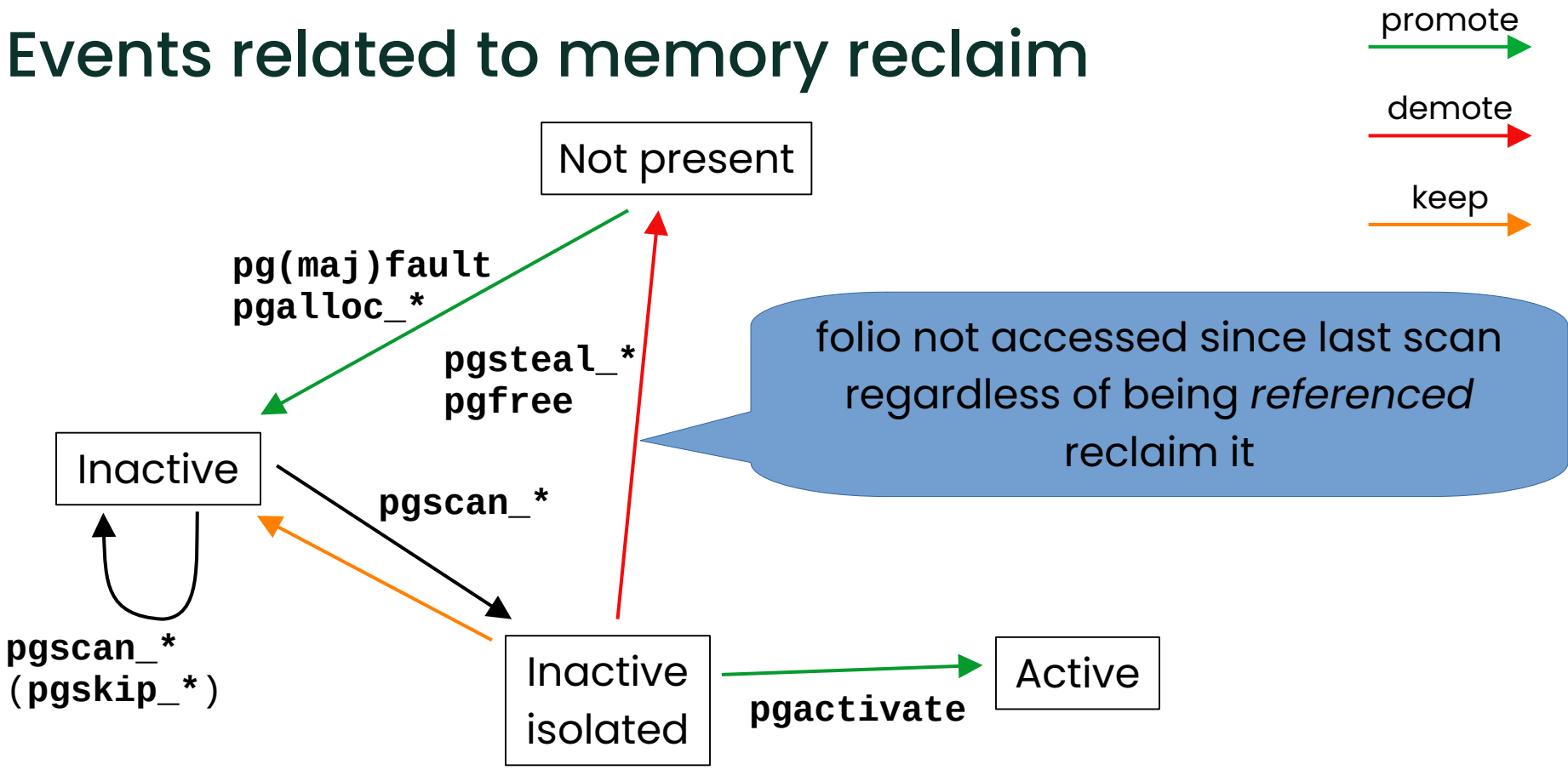
Events related to memory reclaim



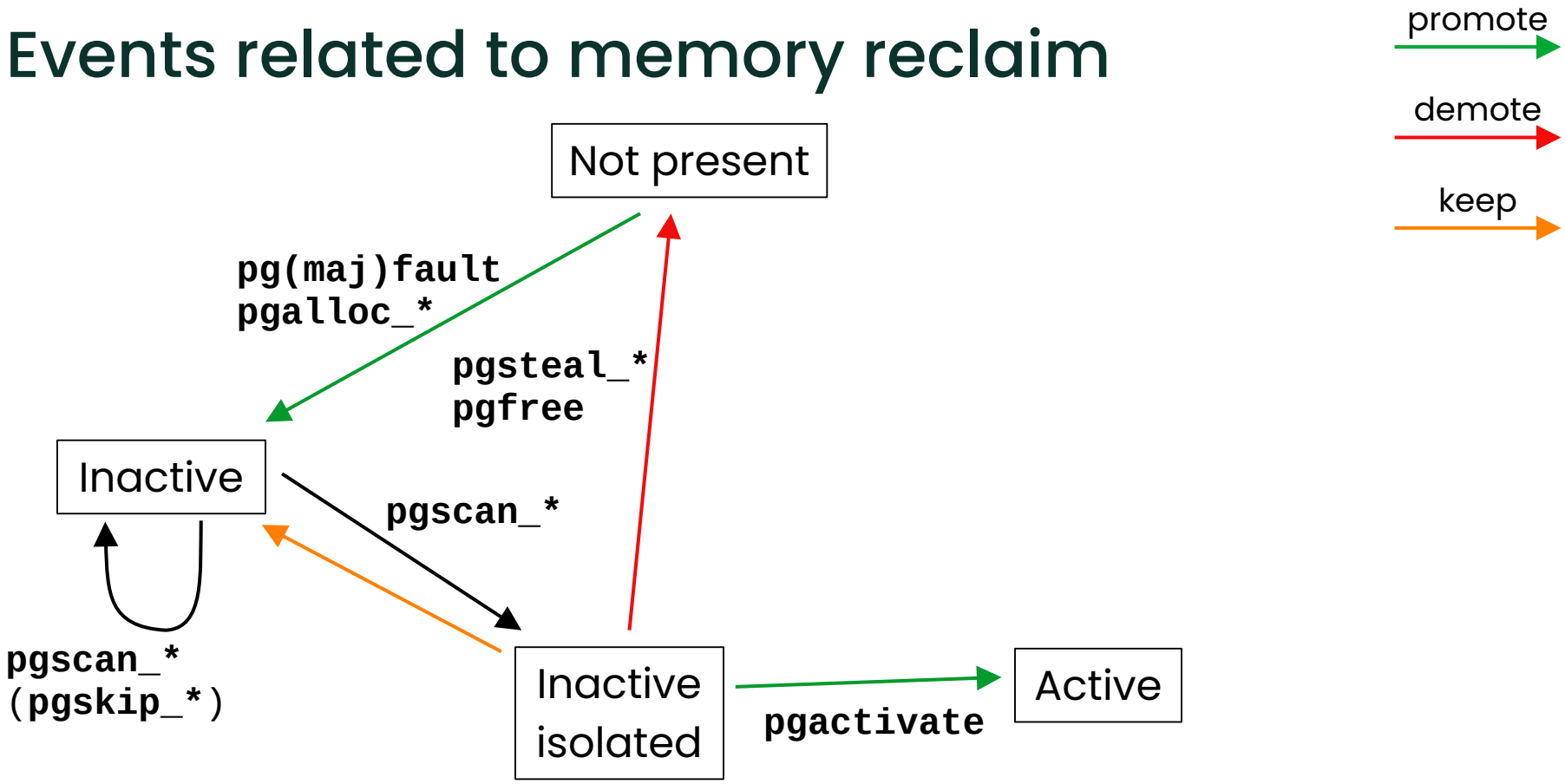
Events related to memory reclaim



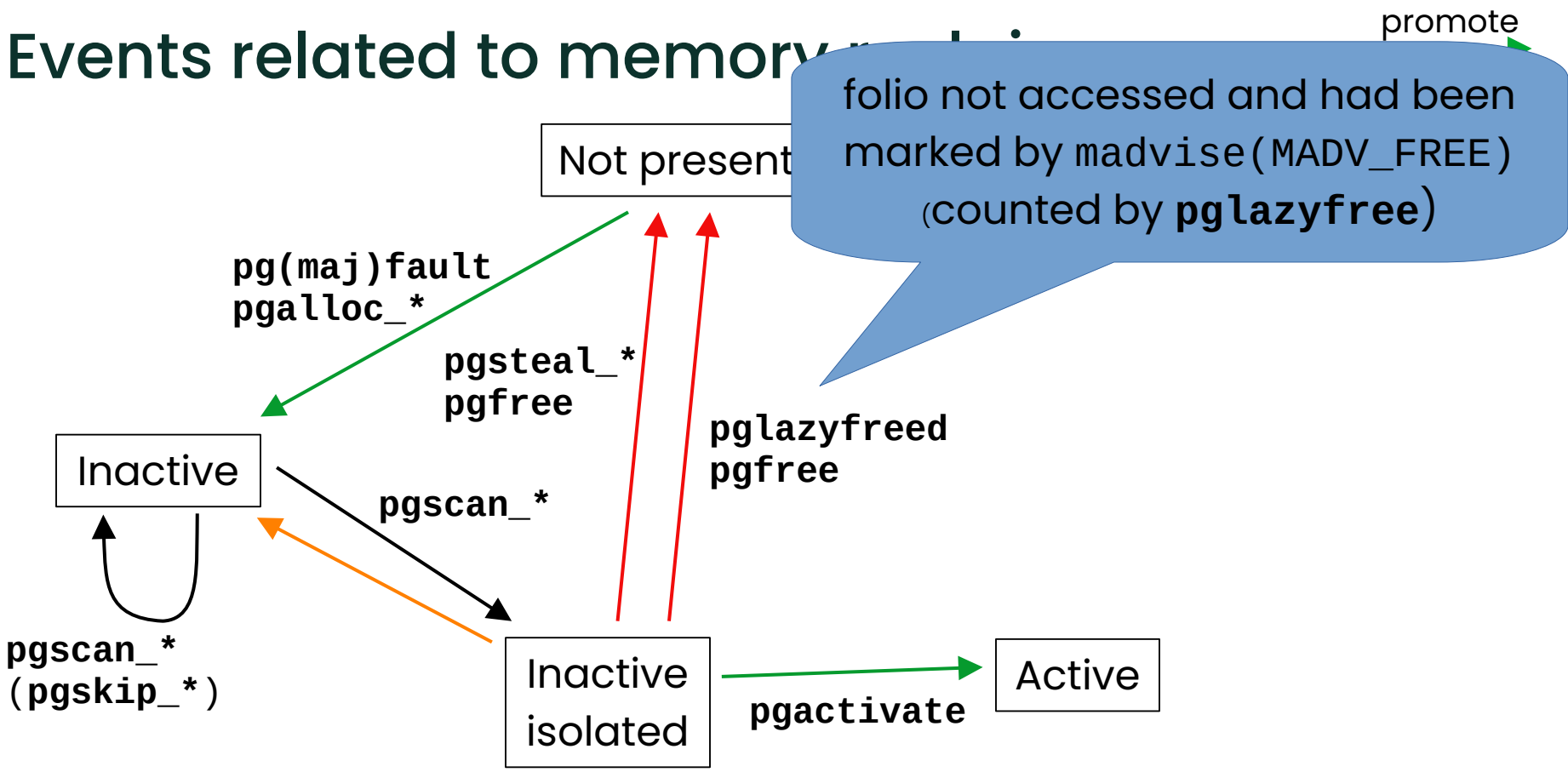
Events related to memory reclaim



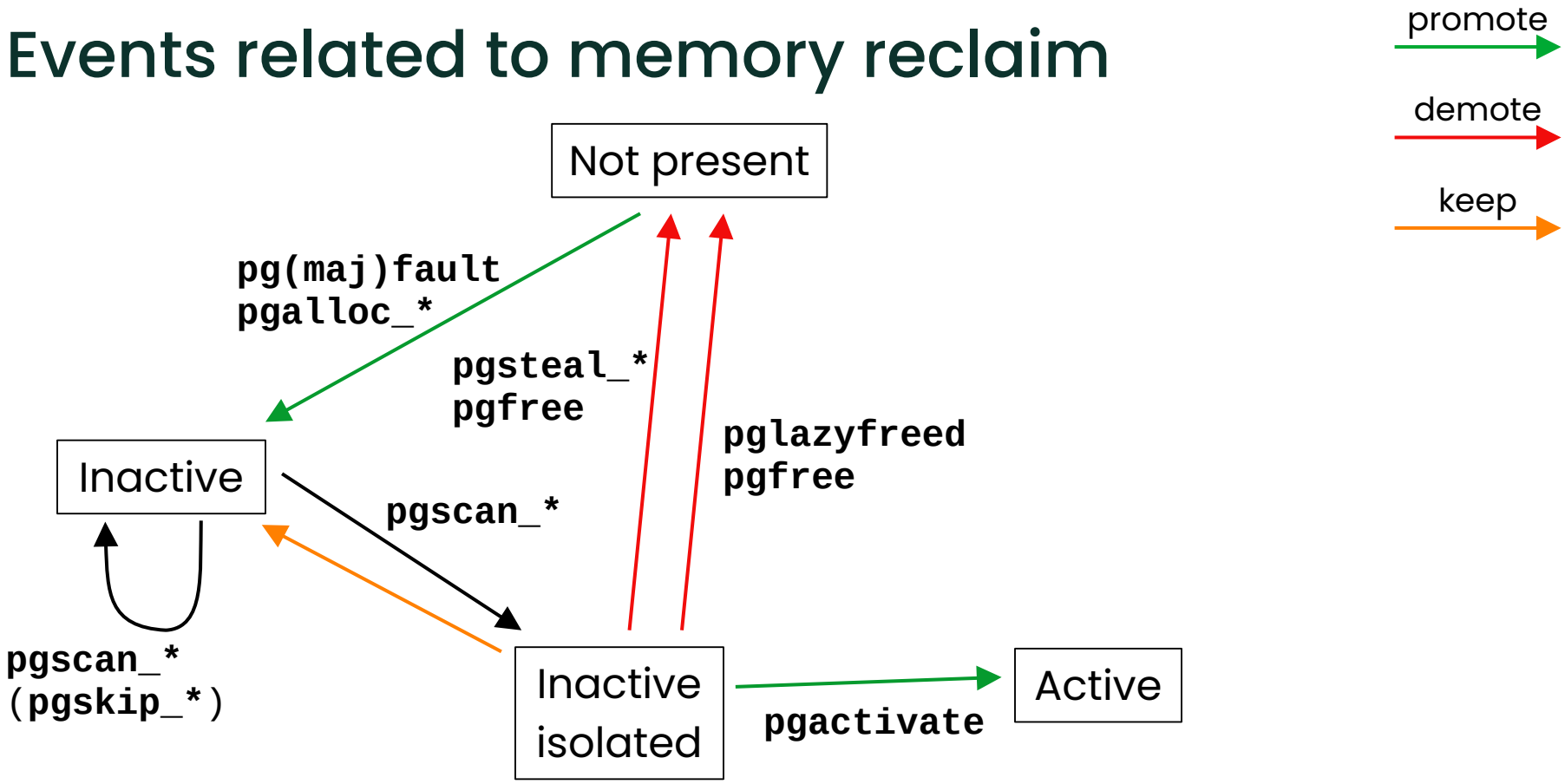
Events related to memory reclaim



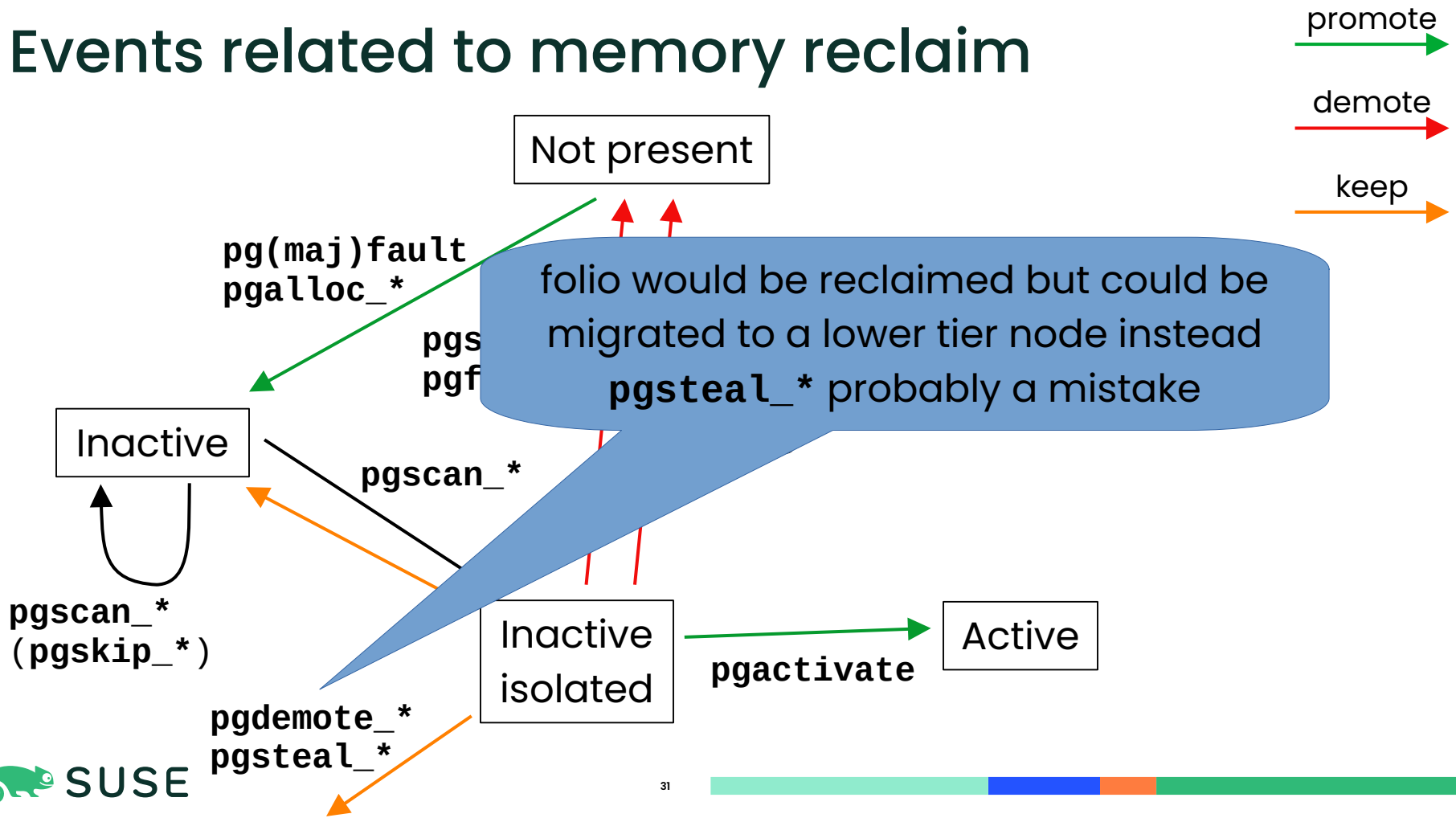
Events related to memory



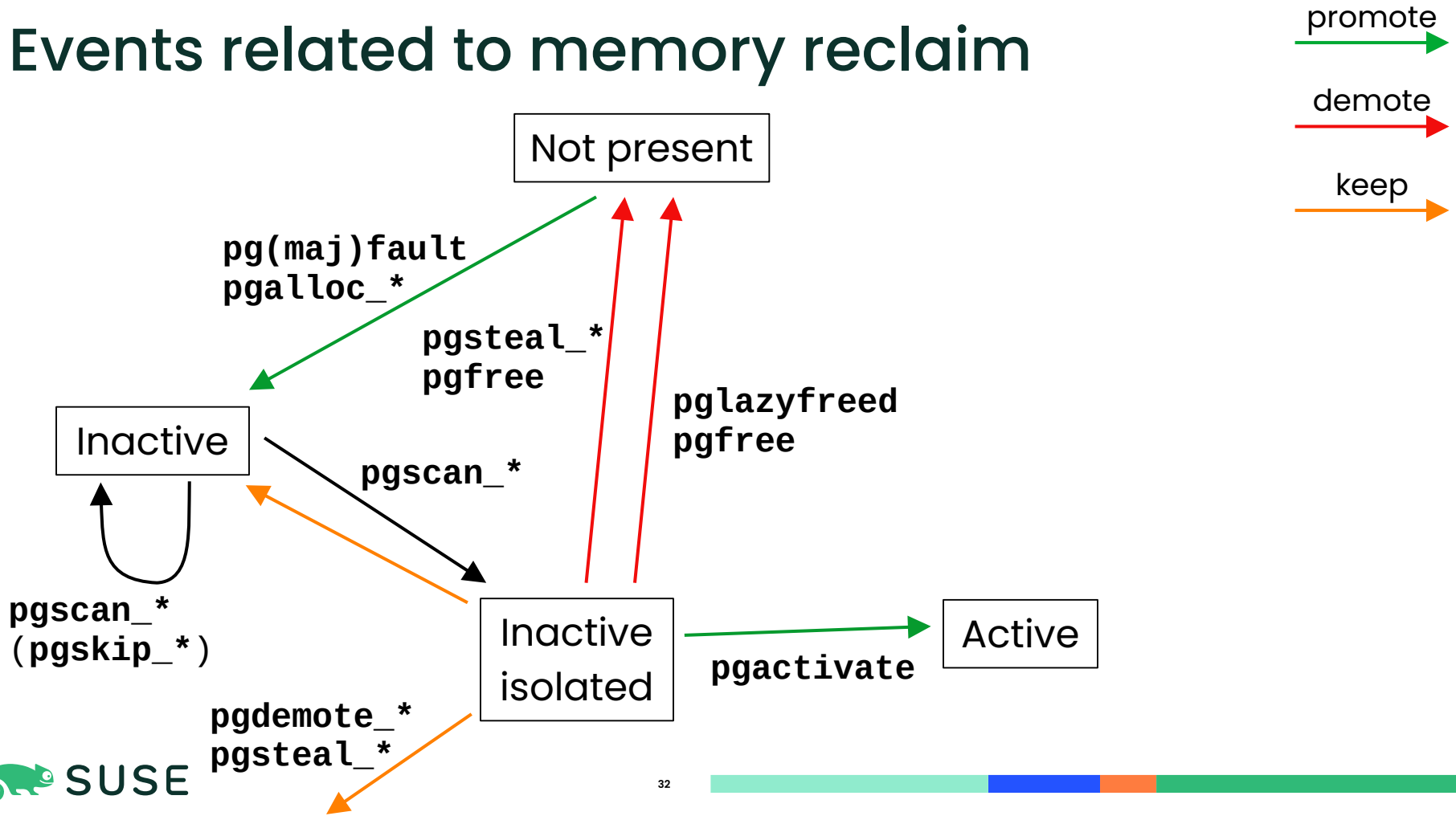
Events related to memory reclaim



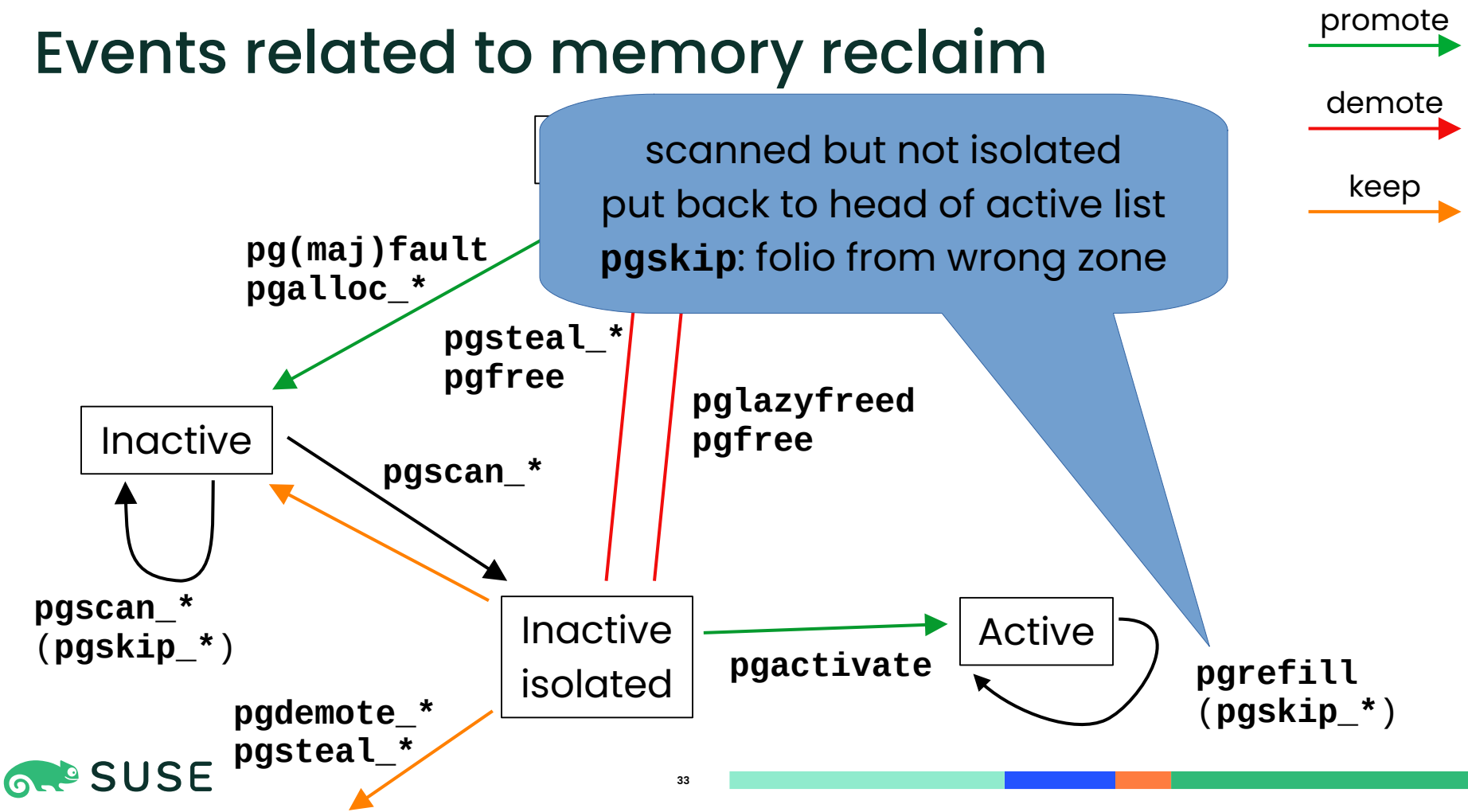
Events related to memory reclaim



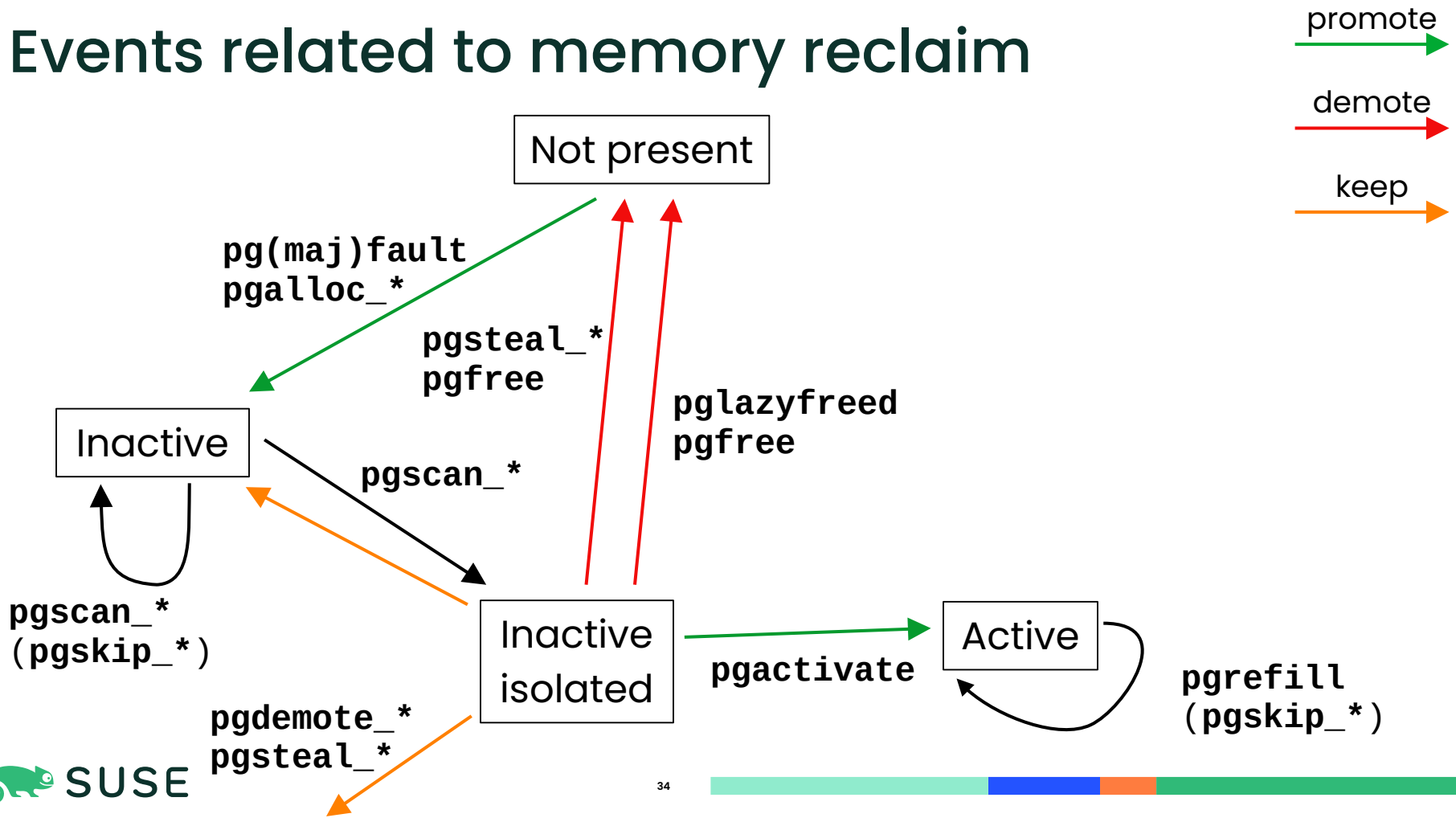
Events related to memory reclaim



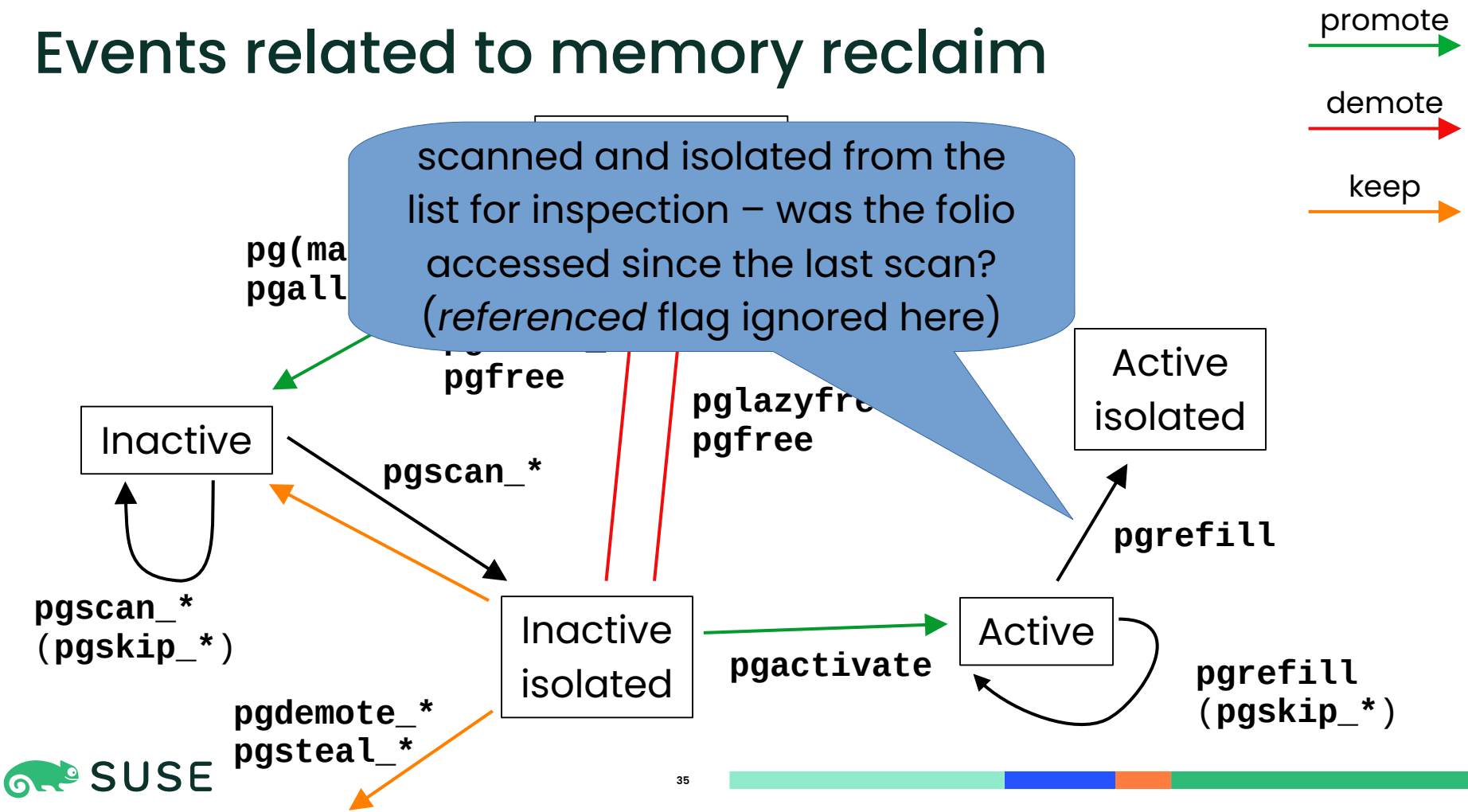
Events related to memory reclaim



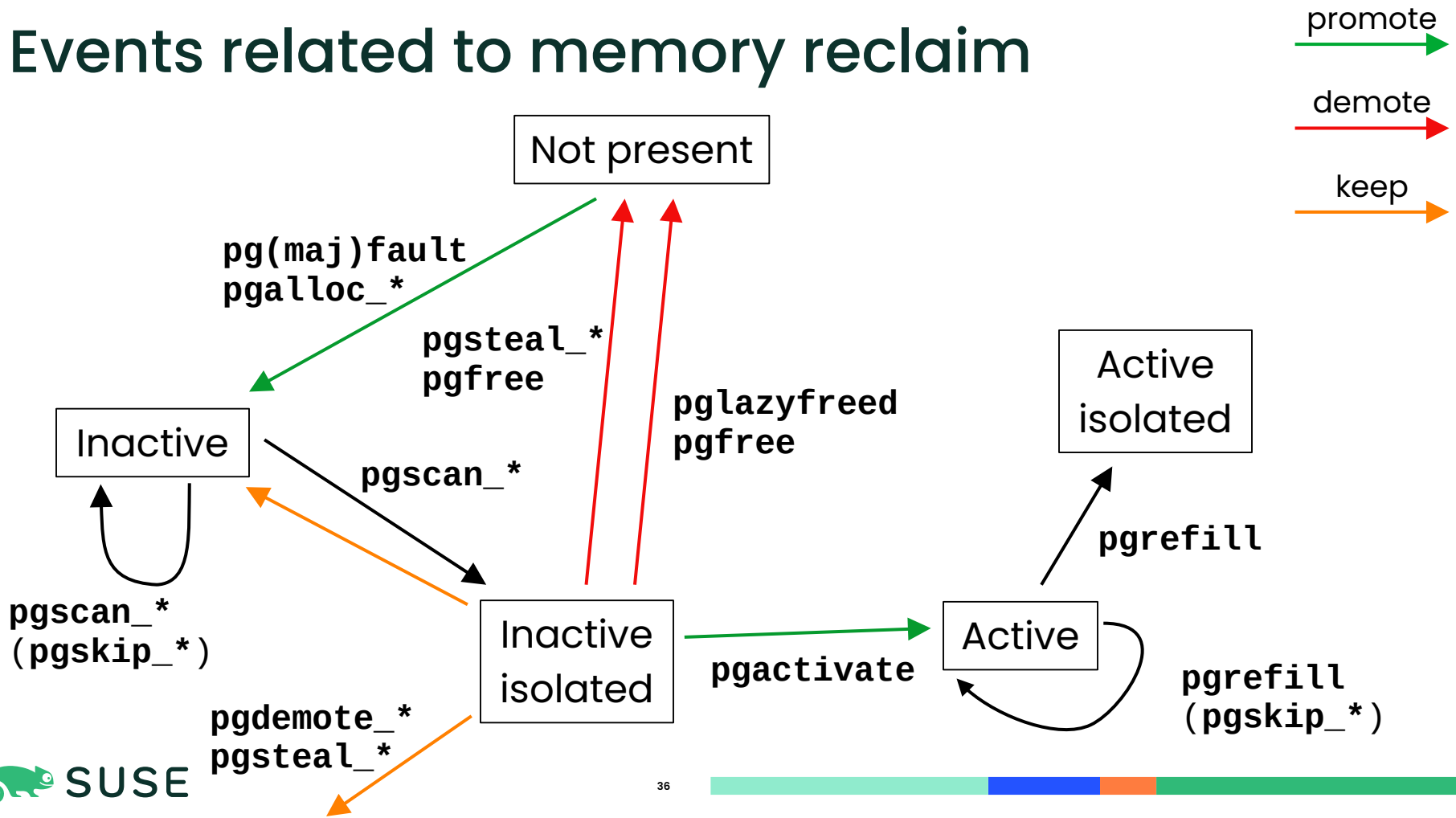
Events related to memory reclaim



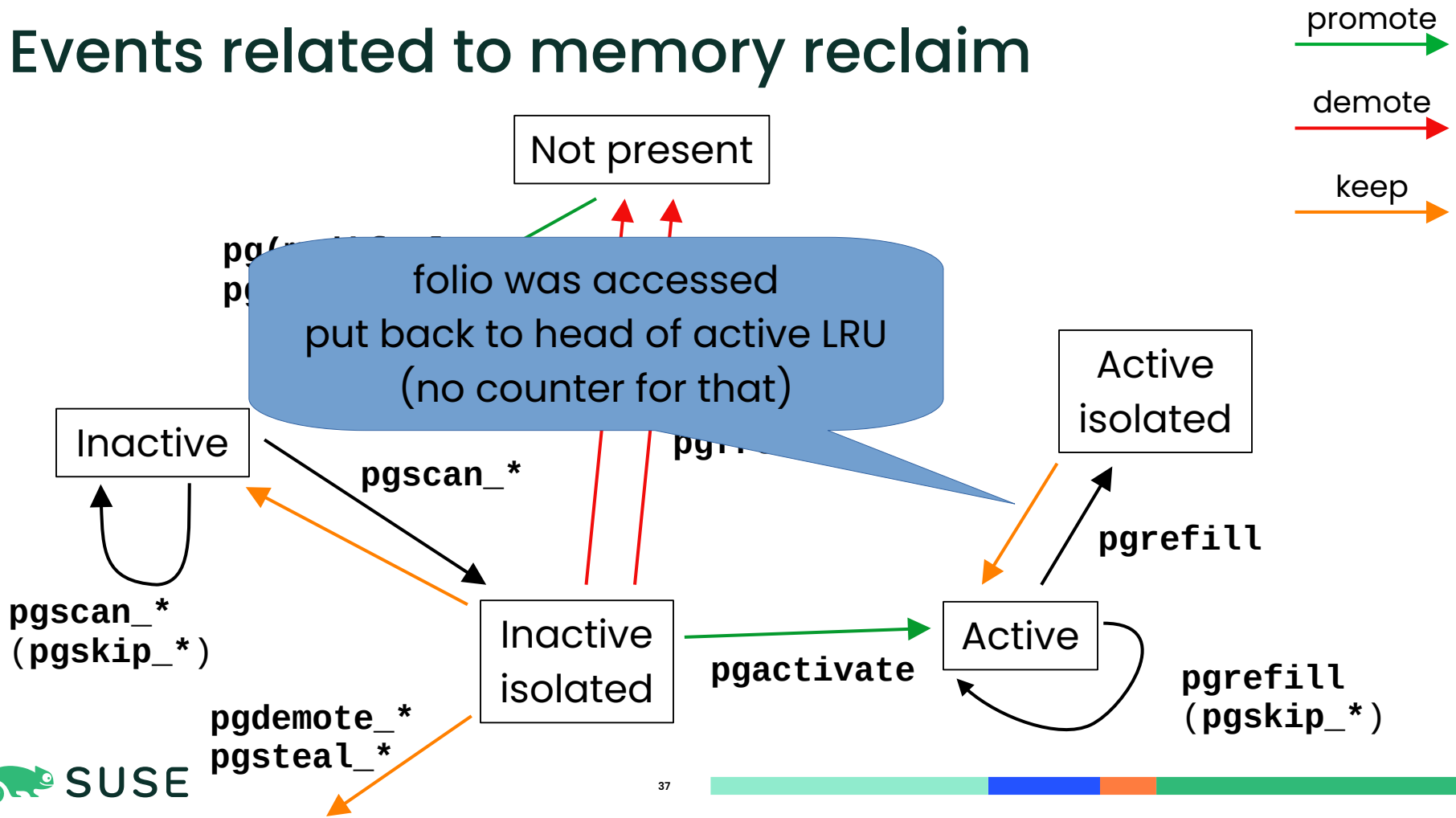
Events related to memory reclaim



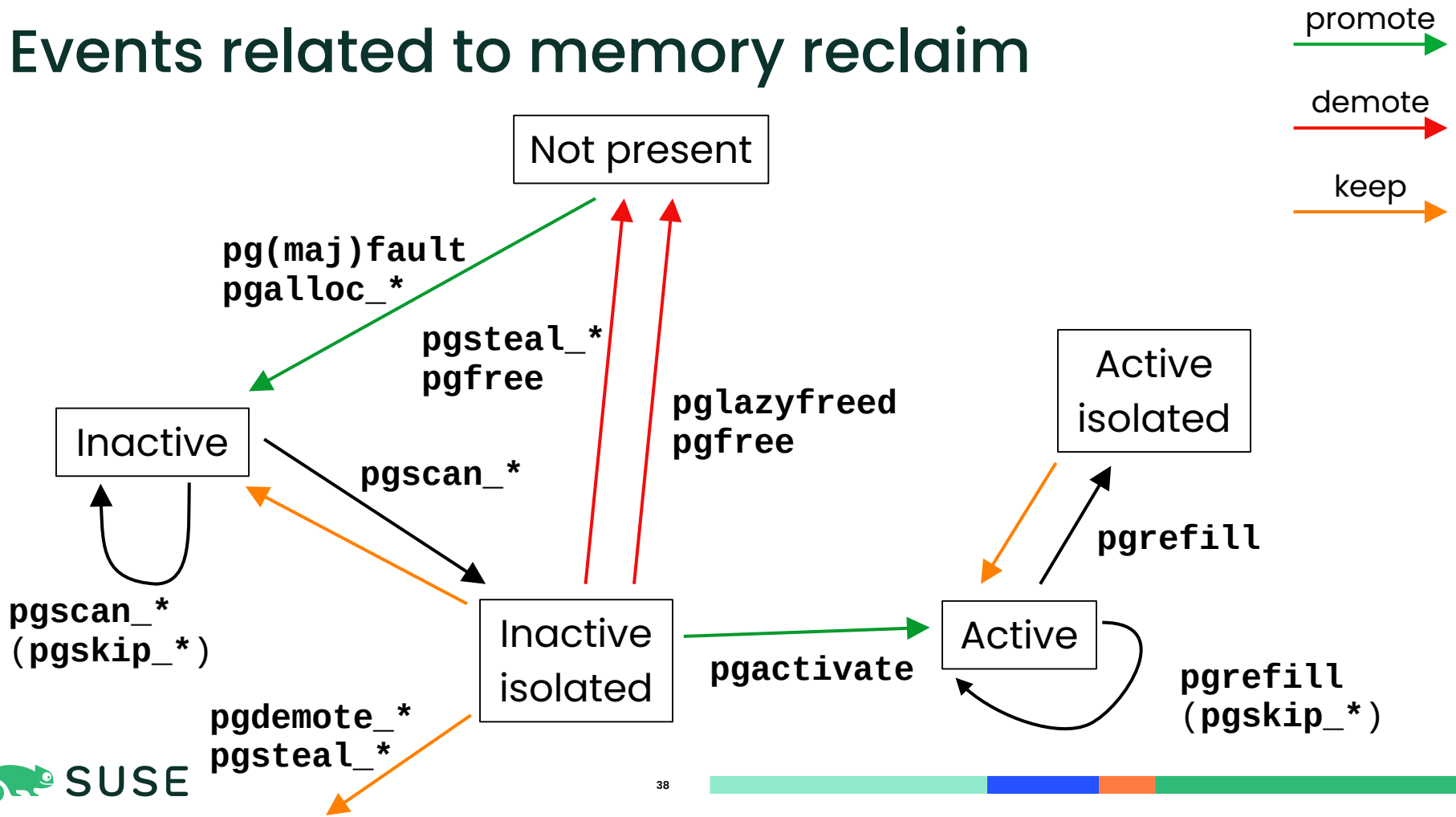
Events related to memory reclaim



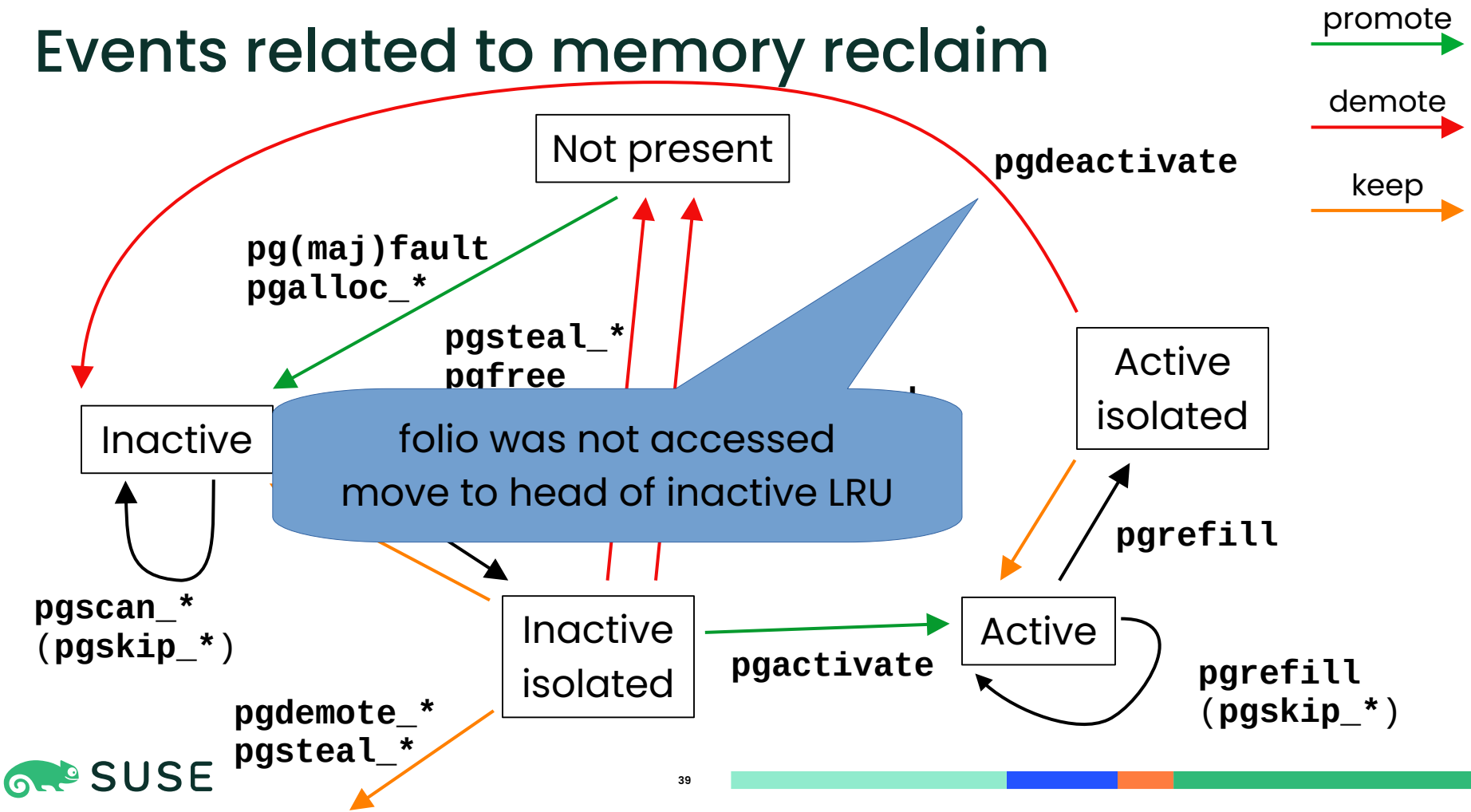
Events related to memory reclaim



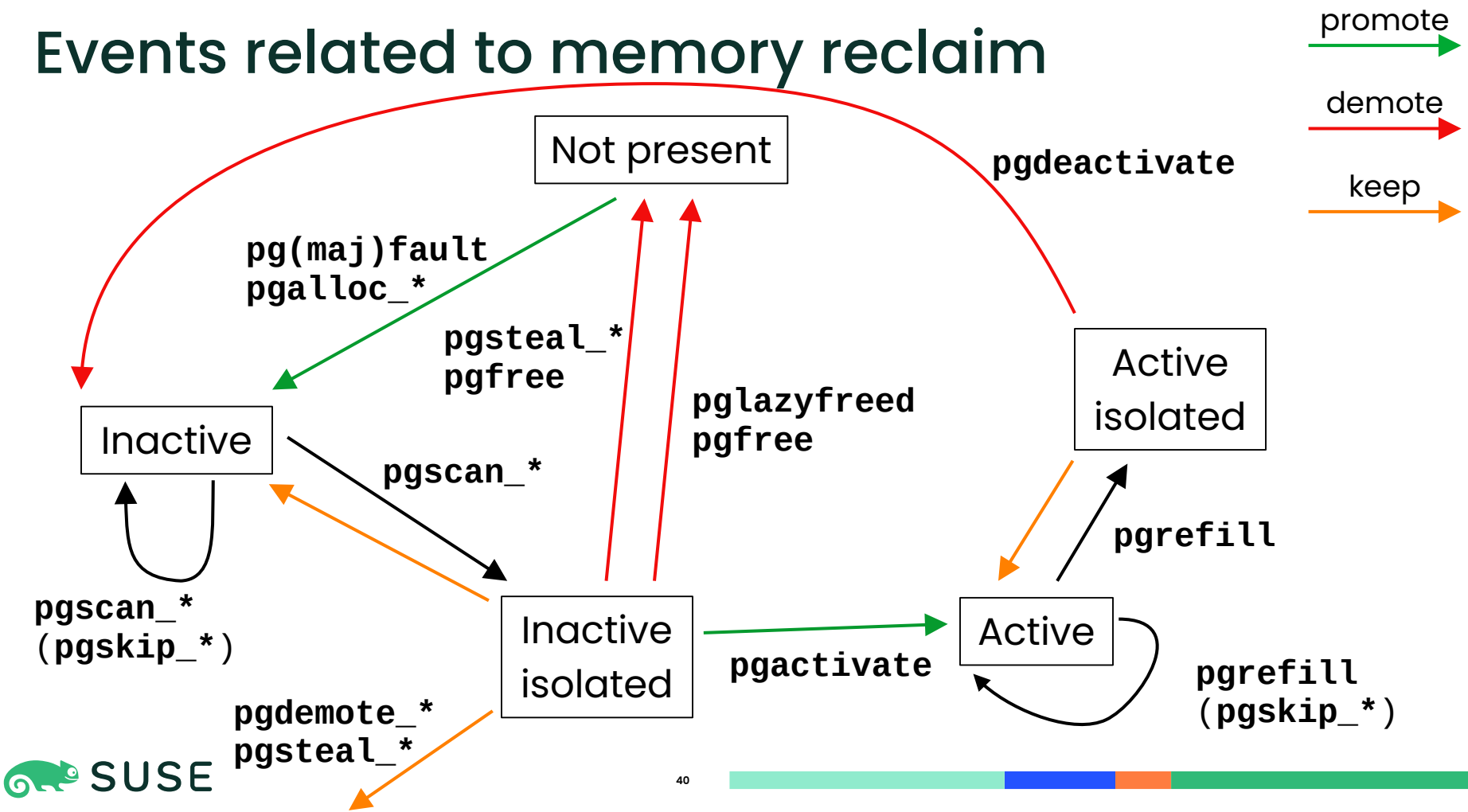
Events related to memory reclaim



Events related to memory reclaim



Events related to memory reclaim



Events related to reclaim/writeback

- anonymous and shmem folios go to swap and are written back immediately (if dirty)
  - **`pswpout`** – pages written to swap (unit is base pages) during reclaim
- dirty file folios are written back outside of reclaim (flusher kthreads) independently
  - reclaim will only mark dirty folios with a `reclaim` folio flag and put them on active list
    - so it doesn't encounter them again during another pass of inactive list
    - but this also increments the **`pgactivate`** counter! (bug?)
    - same after initiating with swap out - unless the write is synchronous
- **`pgrotated`** – incremented (unit is base pages) when folio writeback finishes (no longer dirty) and has the `reclaim` flag – it's moved to the tail of inactive list (immediately)
- **`pgpgout`** – all writes via the block layer, unit is kB, swap and writeback only part of it

Events related to workingset detection

- Motivation: workloads changing their working set (which memory they access) might be thrashing if folios can't be accessed often enough while on inactive list to have chance to be promoted to active list
  - Example: a system with only 4 or 8GB RAM running Firefox, now LibreOffice is started
  - Inactive list is intentionally small, the workload's new working set might be just larger
  - If a recently reclaimed folio is faulted in again, we don't know if it's new or thrashing
  - Meanwhile the folios on active list might be already idle, but we won't shrink it quickly enough
- Idea: remember when folios are reclaimed using "shadow nodes" that outlive them
  - Track certain events in reclaim to approximate "access distance" between two accesses to the same folio
  - Use this to estimate if the refaulted folio would have not been reclaimed in the first place, if the inactive list was larger on the expense of the active list, and also the other folio type's lists
    - I.e. for a file fault, would there have been reclaim+refault of this folio if the file inactive file list size would grow by the file active list size, and also both anon list sizes (if swapping is enabled)
    - If the answer is yes, put the page on active list instead of inactive list

Events related to workingset detection

- **`workingset_refault_anon/_file`** – a fault reads back a folio that was previously reclaimed (it has a shadow entry)
- **`workingset_activate_anon/_file`** – a refault happening soon enough after reclaim to consider the folio active – it will be put on active list immediately instead of inactive
- **`workingset_restore_anon/_file`** – an activation refault of a folio previously active
- **`workingset_nodes`** – current number of shadow nodes (not events)
- **`workingset_nodereclaim`** – shadow nodes reclaimed via slab shrinker

Conclusion

- /proc/vmstat events provide insight into what the mm subsystem is doing
- Helps for dealing with reports such as
  - “my system was slow and I think it was mm’s fault”
  - “kswapd/kcompactd takes too much cpu time”
  - “swap usage increased even though there was free memory”
- Often the absolute values are not interesting
  - Take a snapshot e.g. every 10 seconds (also with meminfo), post-process to calculate differences and then e.g. per-second rate of events and how it changes over time
  - Can’t offer exact debugging recipes here (experience helps a lot) but knowing what the counters report (the details matter) and how things work is a good start

Conclusion – broad example usage

- An example of things to look for in the “swap usage increased even though there was free memory” case
  - Do snapshots of nr_ counters confirm that memory went below watermarks?
  - Does it correlate with swap usage increase (only in meminfo) and **`pswpout`**
  - Is page cache becoming very low and anonymous memory usage high in the process?
    - swap out caused by genuine memory pressure due to anonymous memory
  - Not, but **`workingset_activate_file`** increases?
    - swap out caused by some anon memory being more idle than those files
    - none or negligible **`pswpin`** events confirms the anon memory was idle

Conclusion – need more details?

- vmstat events we didn't cover – compaction, migration, THP, `mlock(3)`, slab shrinking, hugetlb, CMA, KSM, zswap, direct mapping splits/collapses, ballooning
- What if things behave unexpectedly and we suspect real mm issue?
  - E.g. reclaim failures, unexplained high kcompactd cpu usage
- /proc/vmstat might not provide enough details for events and with coarse granularity
- The next level of getting details is observing trace events
  - e.g.: `trace_mm_migrate_pages(stats.nr_succeeded, stats.nr_failed_pages, stats.nr_thp_succeeded, stats.nr_thp_failed, stats.nr_thp_split, stats.nr_split, mode,reason);` - all with timestamp, process context…
  - Much more data to collect and postprocess (or process online using BPF hooks)
- Even more low-level – `drgn` debugger scripts (on live vmcore)

Related talk at OSS EU 2026

- Richard Weinberger: A surprisingly hard question: How much memory does my system use ?
  - Friday 16:30, Panorama Hall
  - Includes per-process memory metrics, memory cgroups and more

Reminder: non-zero swap usage is not a bug

- Consider a system that has enough memory to fit its workload and some page cache
- An overnight activity (backup, virus scanning) results in accessing many more files
- No swap usage with older kernels, all swap is free
- Suddenly, some swap becomes used after a kernel/distro upgrade – why?
  - Same workload, same amount of memory?
- The answer: kernel got better at tracking unused file vs anon memory around v5.9
  - Workingset change tracking expanded to anon memory
  - Previously reclaim very biased towards page cache, reluctant to swap out anon
  - Now able to detect anonymous page not touched *at all* by their processes
    - Vs file pages touched at least once a day due to the overnight activity
    - Swapping out is the correct thing to do - more memory for actually used page cache
- TID: https://www.suse.com/support/kb/doc/?id=000021196



SUSE

Thank you

Questions?

Events related to page allocation

- Physical (buddy) page allocations, for both userspace and kernel purposes
- Allocation success – **`pgalloc_normal`** (or **`_dma`**, **`_dma32`**, **`_movable`**, **`_device`**)
- Also on success – **`numa_`** events
  - **`numa_local`** from node of allocating cpu, otherwise **`numa_other`**
  - **`numa_hit`** when from preferred node (i.e. via mempolicy, implicitly local node)
    - **`numa_interleave`** when **`numa_hit`** and mempolicy is (weighted) interleave
  - **`numa_miss`** otherwise, and then also **`numa_foreign`** on the preferred node's zone
- Page freeing increments **`pgfree`** (no zone distinction)
- All of these count base pages (2^n per single allocation), except **`numa_interleave`** (bug?)

Events related to page allocation

- Allocating means the free memory decreases, eventually below a watermark (min → low → high)
- Drops below the low watermark – kswapd woken up for the NUMA node
  - Kswapd initiates memory reclaim on its node – increments **`pageoutrun`** (global counter) and reclaims until free memory goes above the high watermark
- Kswapd tries to sleep for 100ms only
  - It's woken up during that short sleep – increments **`kswapd_low_wmark_hit_quickly`**
  - Not woken up but free memory went below the high watermark during the short sleep – increments **`kswapd_high_wmark_hit_quickly`** and continues reclaming
  - Not woken up, still above high watermark – goes sleep with no further timeout
- Free memory drops below min watermark – allocation must perform direct reclaim – increments **`allocstall_normal`** (**`_dma`** etc. depending the zone where the allocation was requested)
  - May be counted repeatedly if the attempt fails, or else allocates the freed memory faster
  - Too many concurrent direct reclaimers can make them sleep instead - **`pgscan_direct_throttle`**