PETER XU <PETERX@REDHAT.COM>

POSTCOPY PREEMPTION

OUTLINES

- A quick re-cap on live migration
- Postcopy limitations, challenges
- Three optimizations
  - Channel separation, huge page, thread model
- Performance Results
- Future works

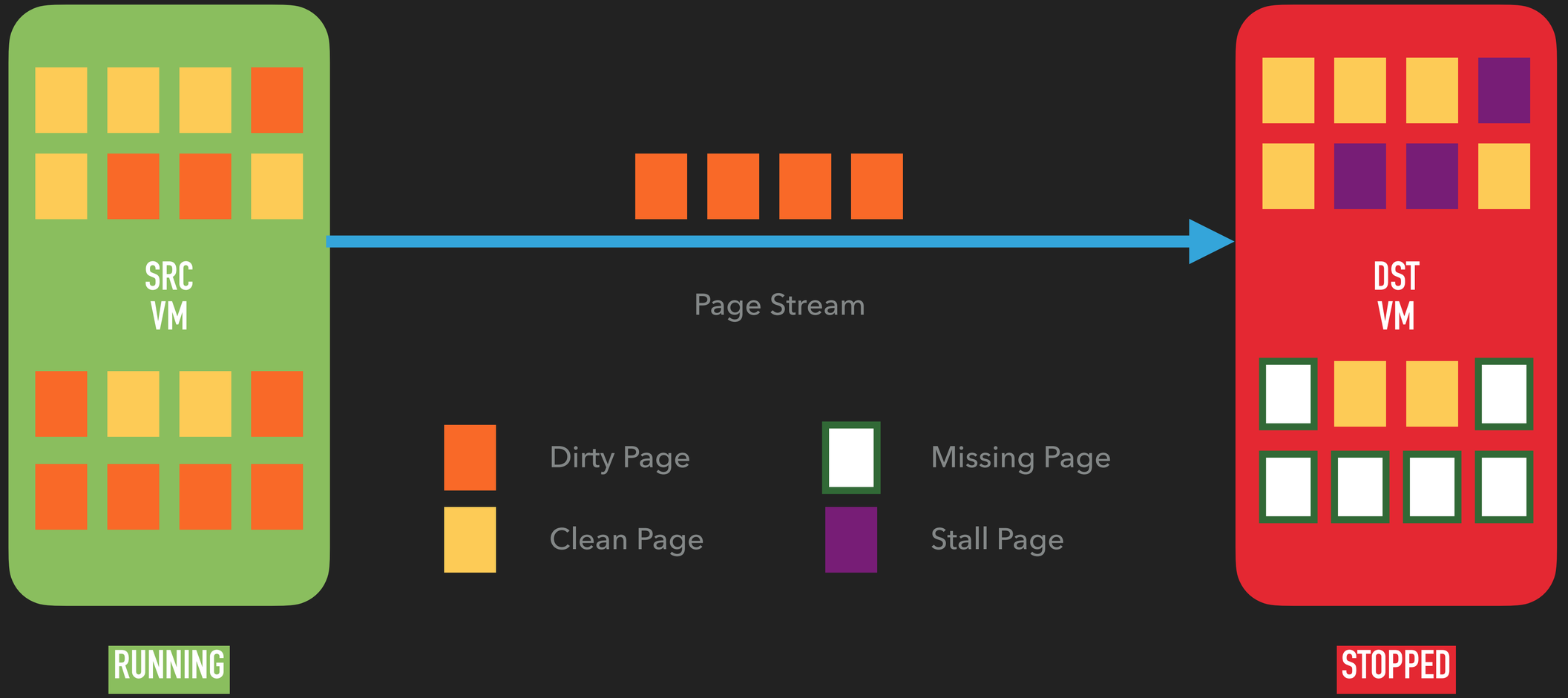
WHAT IS POSTCOPY?

- Allows the VM to start running with partial RAM (compared to precopy)
- Trap page faults when page missing (userfaultfd)
- Always converges

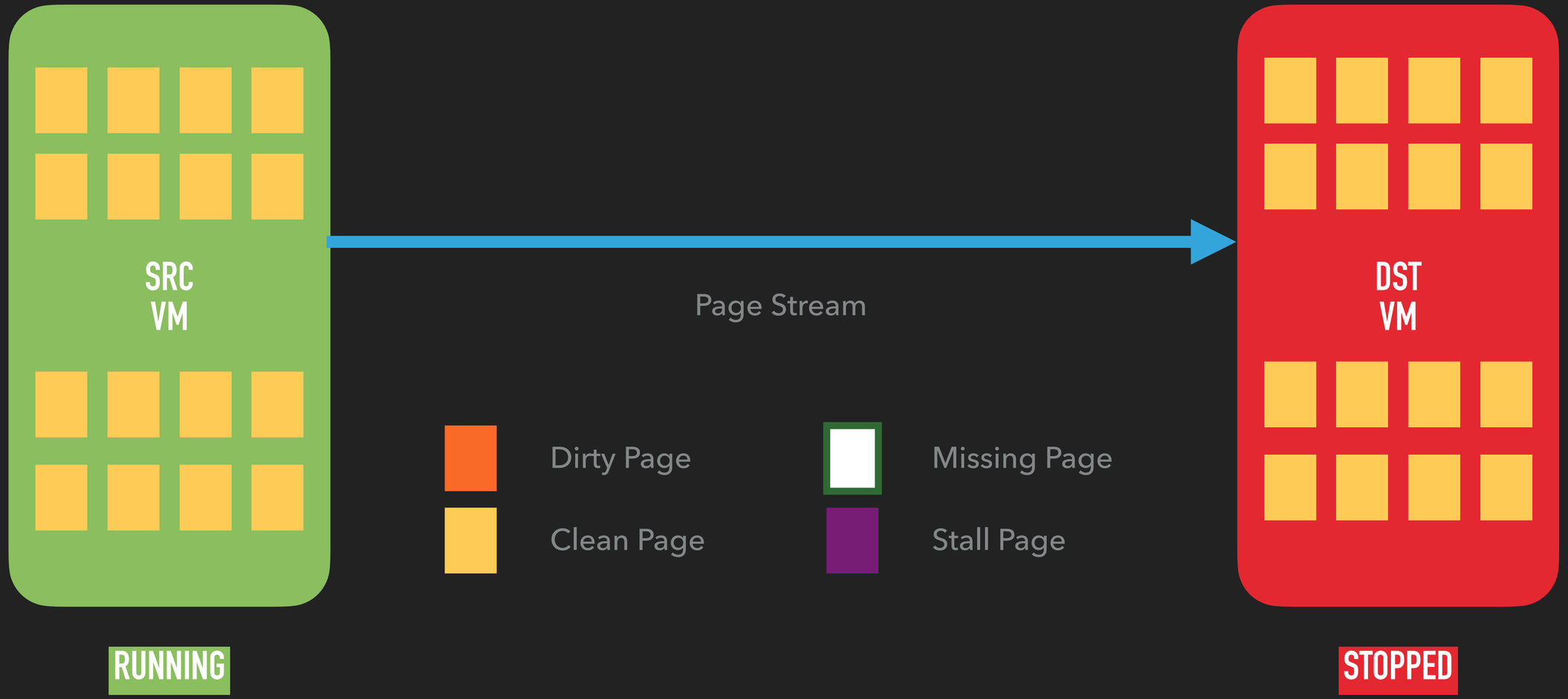
WHAT IS POSTCOPY PREEMPTION?

- A new capability (“postcopy-preempt”) introduced for postcopy-only
  - Need to be enabled on both src/dst QEMU
  - Not compatible with vanilla postcopy
  - No extra configuration needed
- Direct performance improvement on the speed of handling page faults
  - More test results at the end

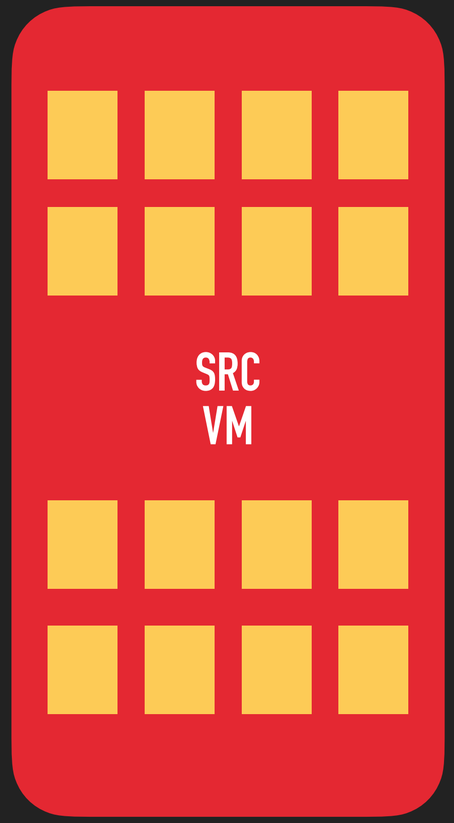
LIVE MIGRATION (PRECOPY)



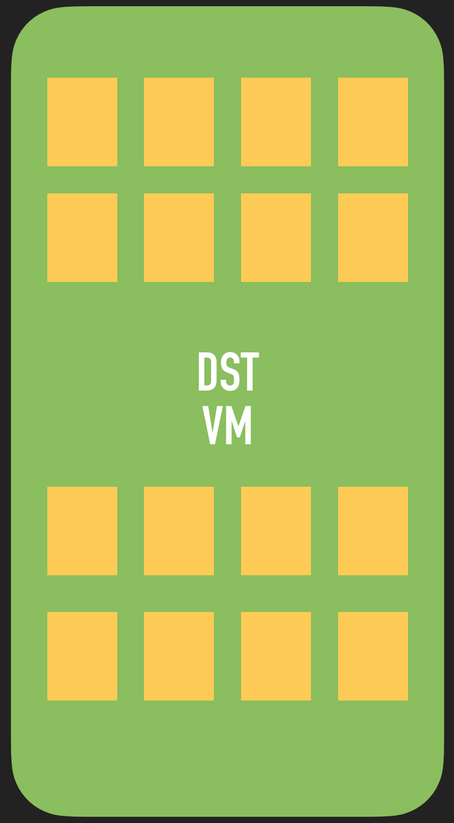
LIVE MIGRATION (PRECOPY COMPLETED)



LIVE MIGRATION (PRECOPY COMPLETED)

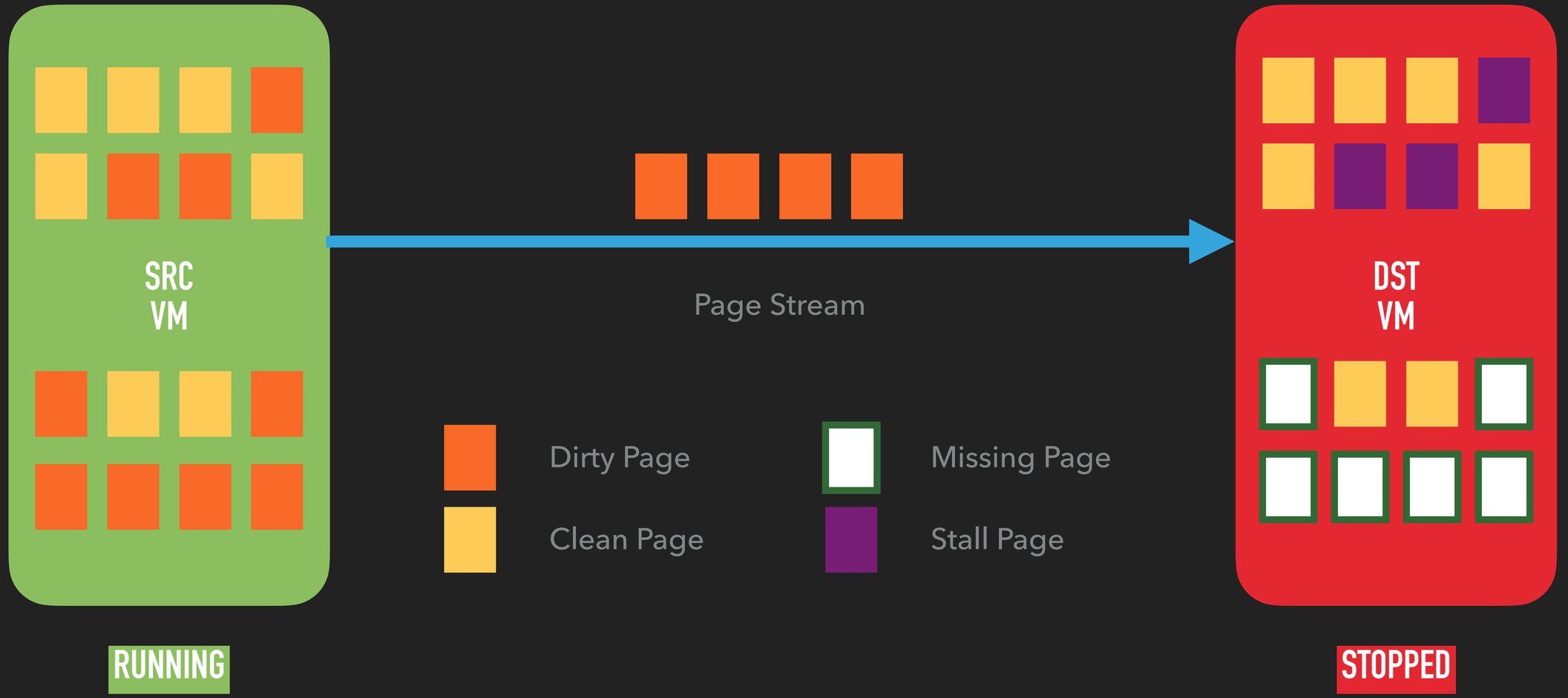


STOPPED

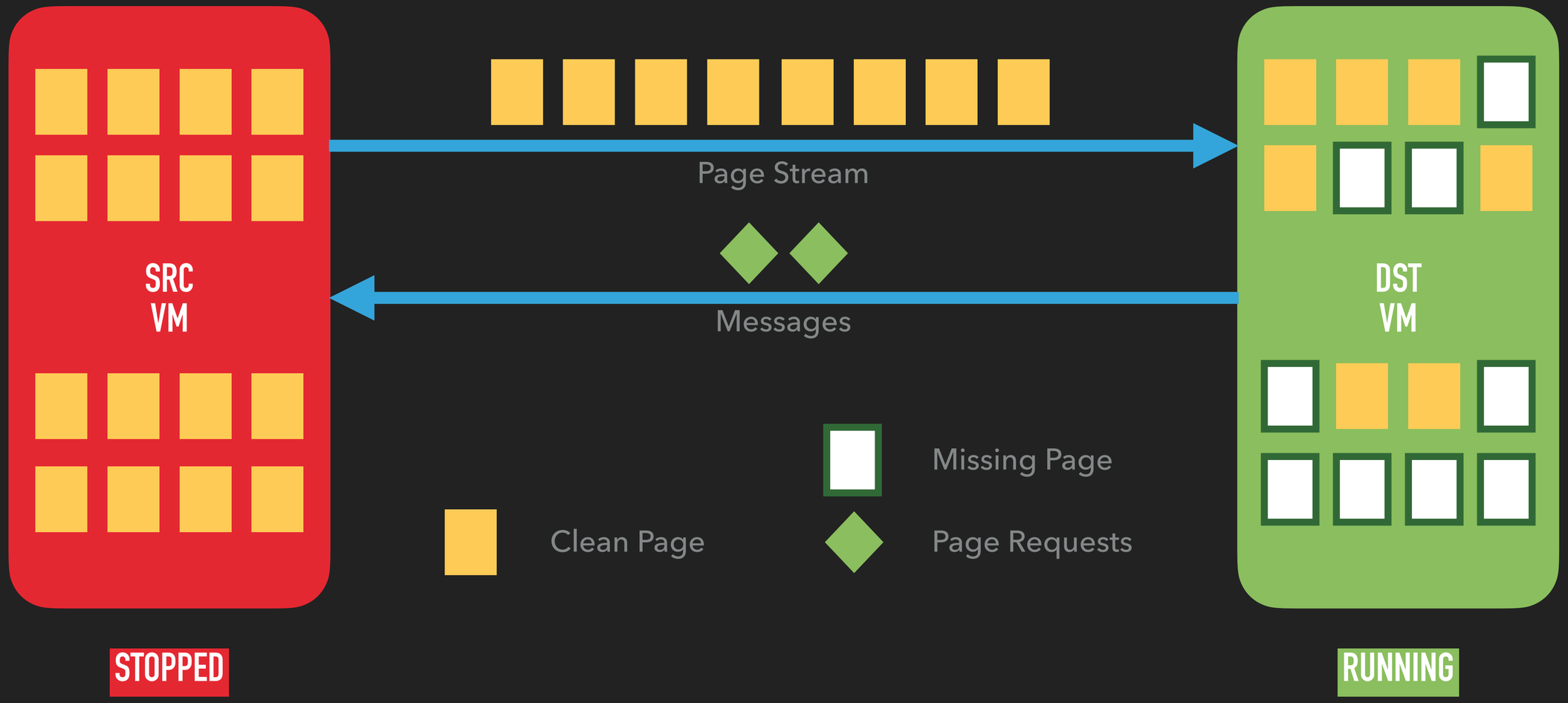


RUNNING

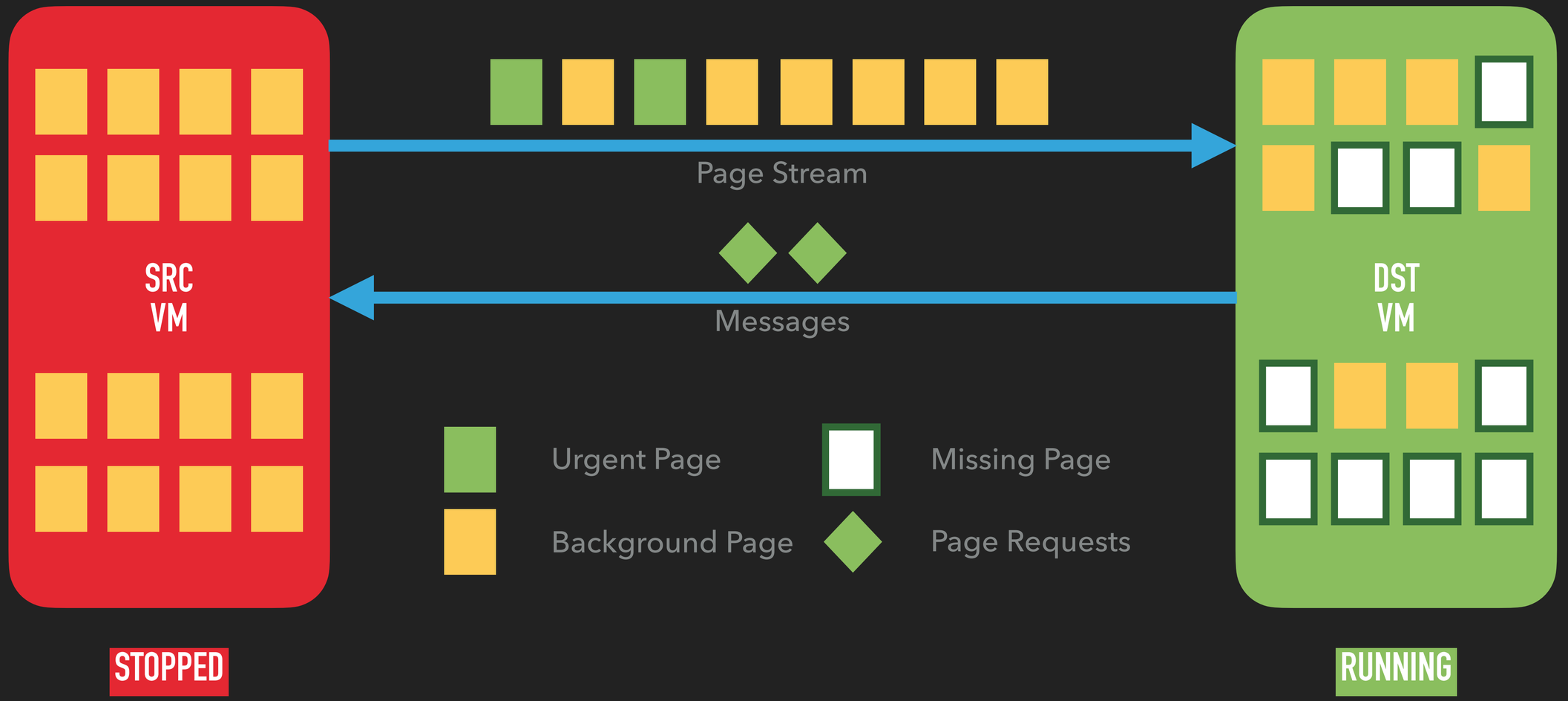
LIVE MIGRATION (PRECOPY, BUT RUN ON DST?)



LIVE MIGRATION (POSTCOPY)



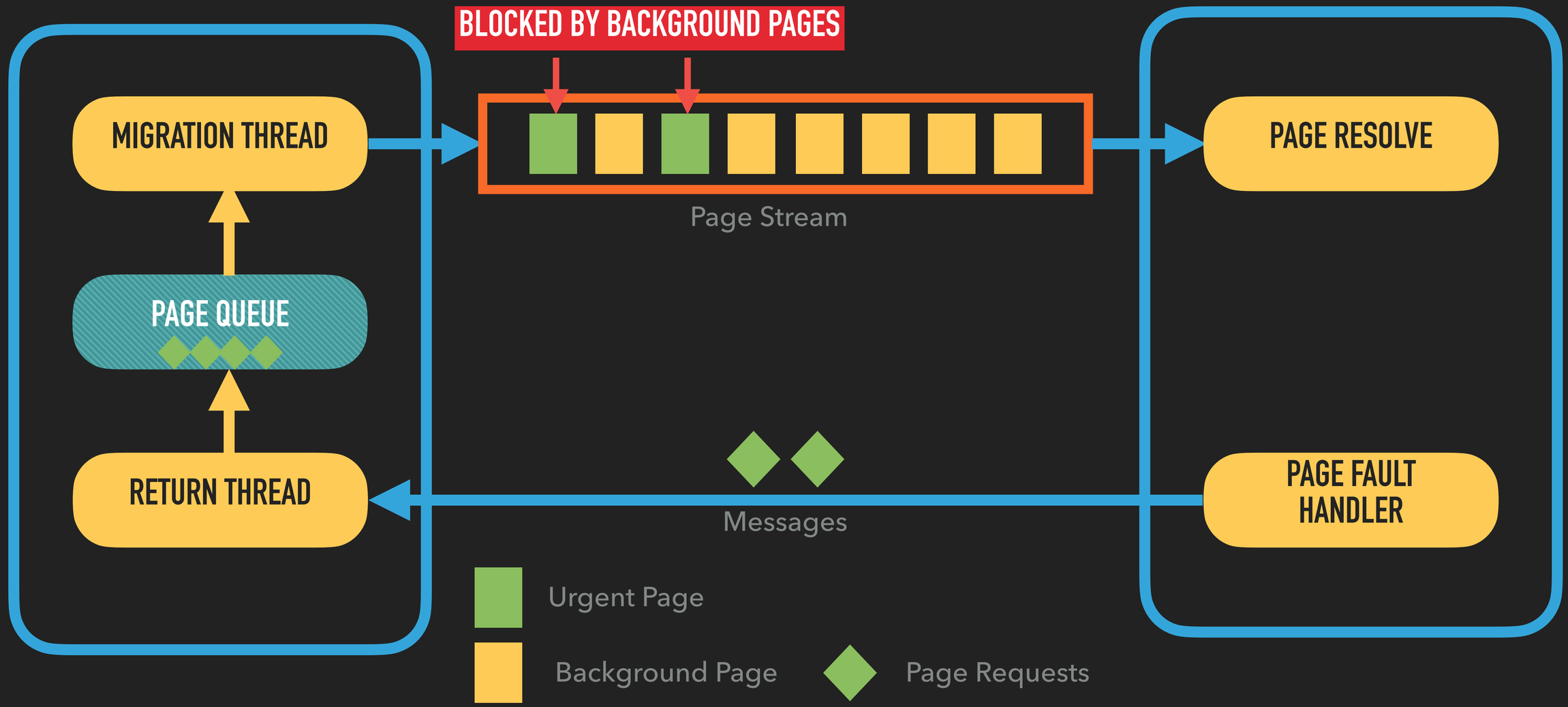
LIVE MIGRATION (POSTCOPY)



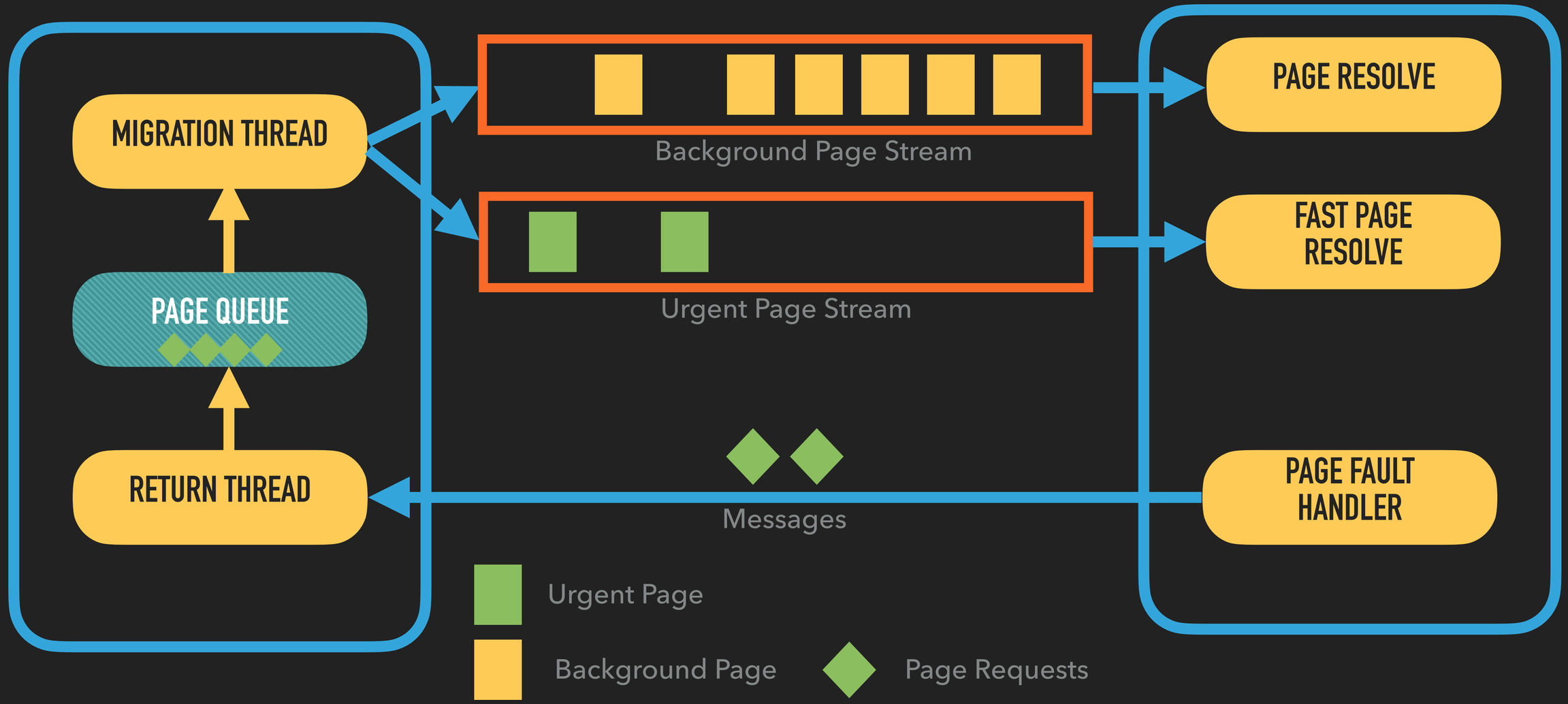
POSTCOPY LIMITATIONS

- Split brain, e.g. network failures during postcopy
  - Postcopy recovery (since QEMU v3.0.0)
- High page request latency
  - For huge pages…
    - Hugetlb double map allows page to be mapped in PAGE_SIZE
      https://lore.kernel.org/all/20220624173656.2033256-1-jthoughton@google.com/
  - Page transfers are slow even for small pages for QEMU
    - An average of 12ms on directly attached 10Gbps network for random access

ISSUE 1 - BACKGROUND PAGE FLUSH



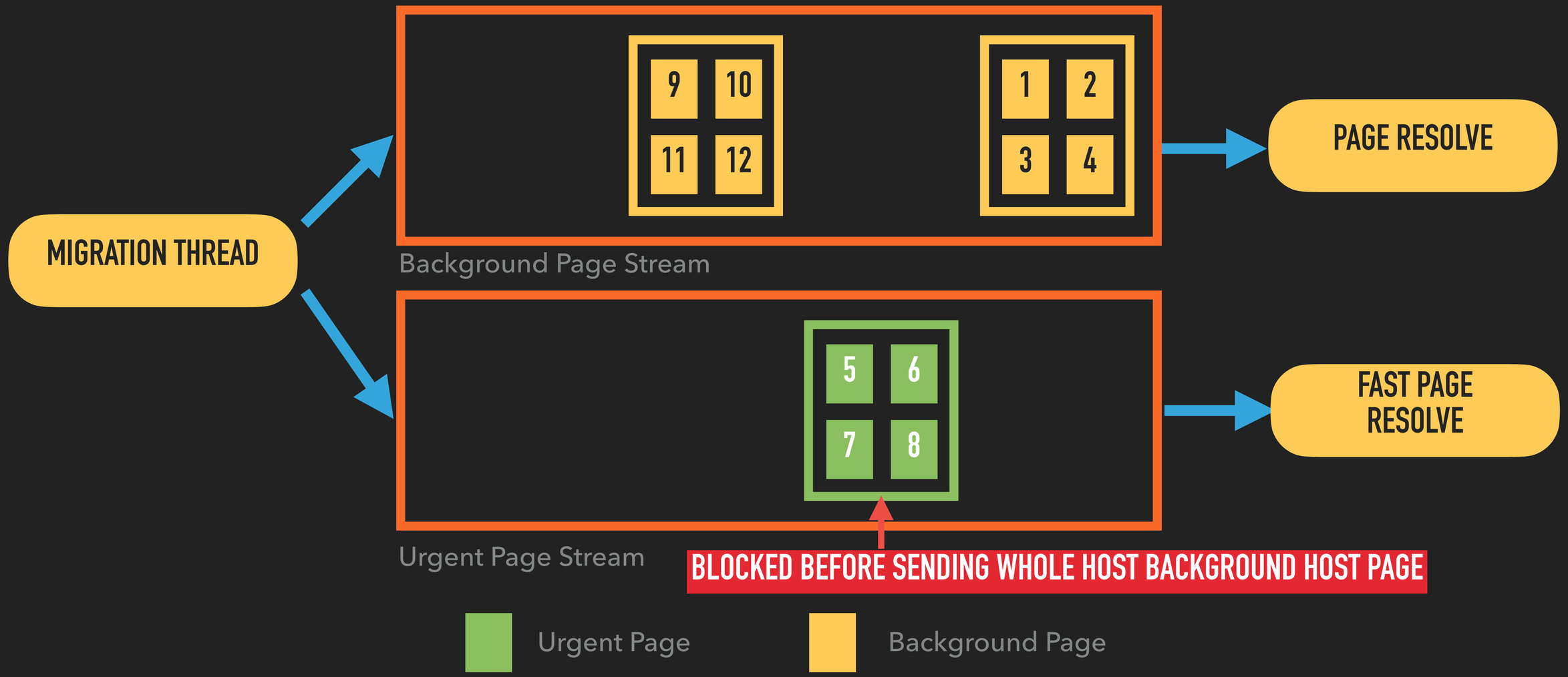
SOLUTION 1 - CHANNEL SEPARATION



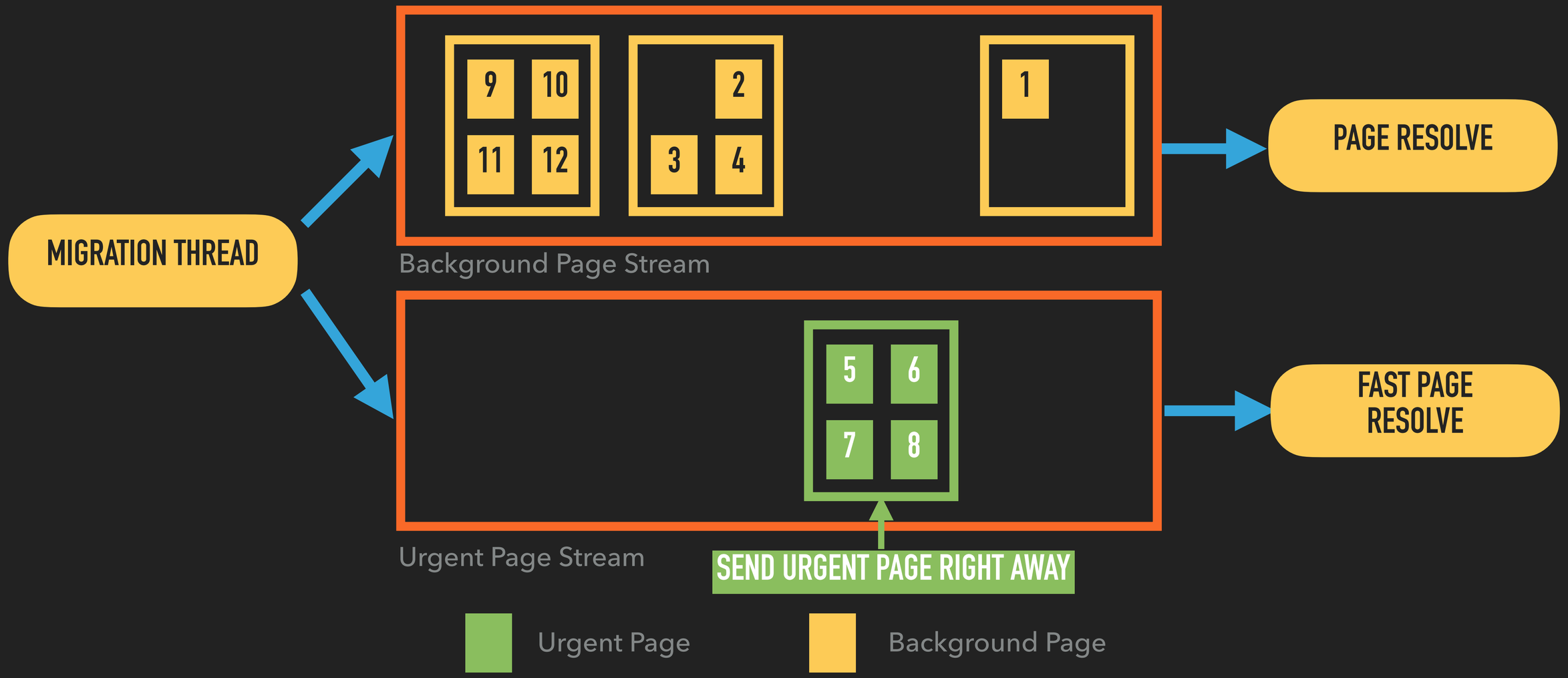
ISSUE 2 – HUGE PAGE GRANULE

- QEMU sends pages always in huge page granule
  - Before finish sending one huge page, we cannot send another page
  - An urgent page cannot preempt sending of a background huge page
- Why?
  - QEMU receiving page using temp huge page buffers, which are limited

ISSUE 2 – HUGE PAGE GRANULE



SOLUTION 2 – HUGE PAGE PREEMPTIONS



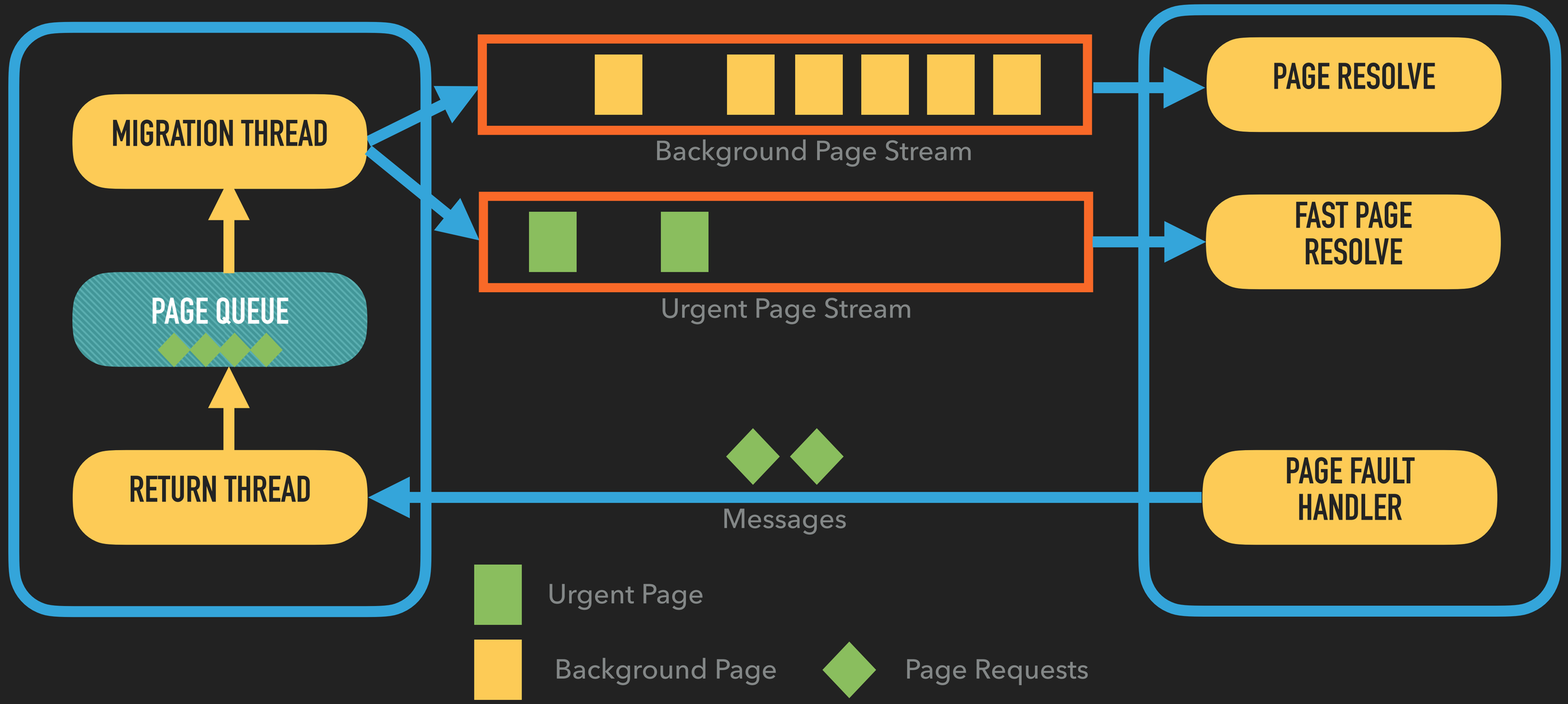
ISSUE 3 - MIGRATION THREAD ITSELF!

- “migration_thread” is the thread to save VM on src QEMU
  - Background sendmsg() blocks not only itself but all the rest (e.g. sending urgent page)
- The only thread to migrate a guest page, due to
  - Legacy state maintenances (RAMState, PageSearchState, bitmaps, etc.)
  - Required by all kinds of features (compression, XBZRLE, multifd, etc.)
    - Compression: distribute raw pages to compressor threads
    - XBZRLE: global xbzrle state maintenance
    - Multifd: entrance of page distributions

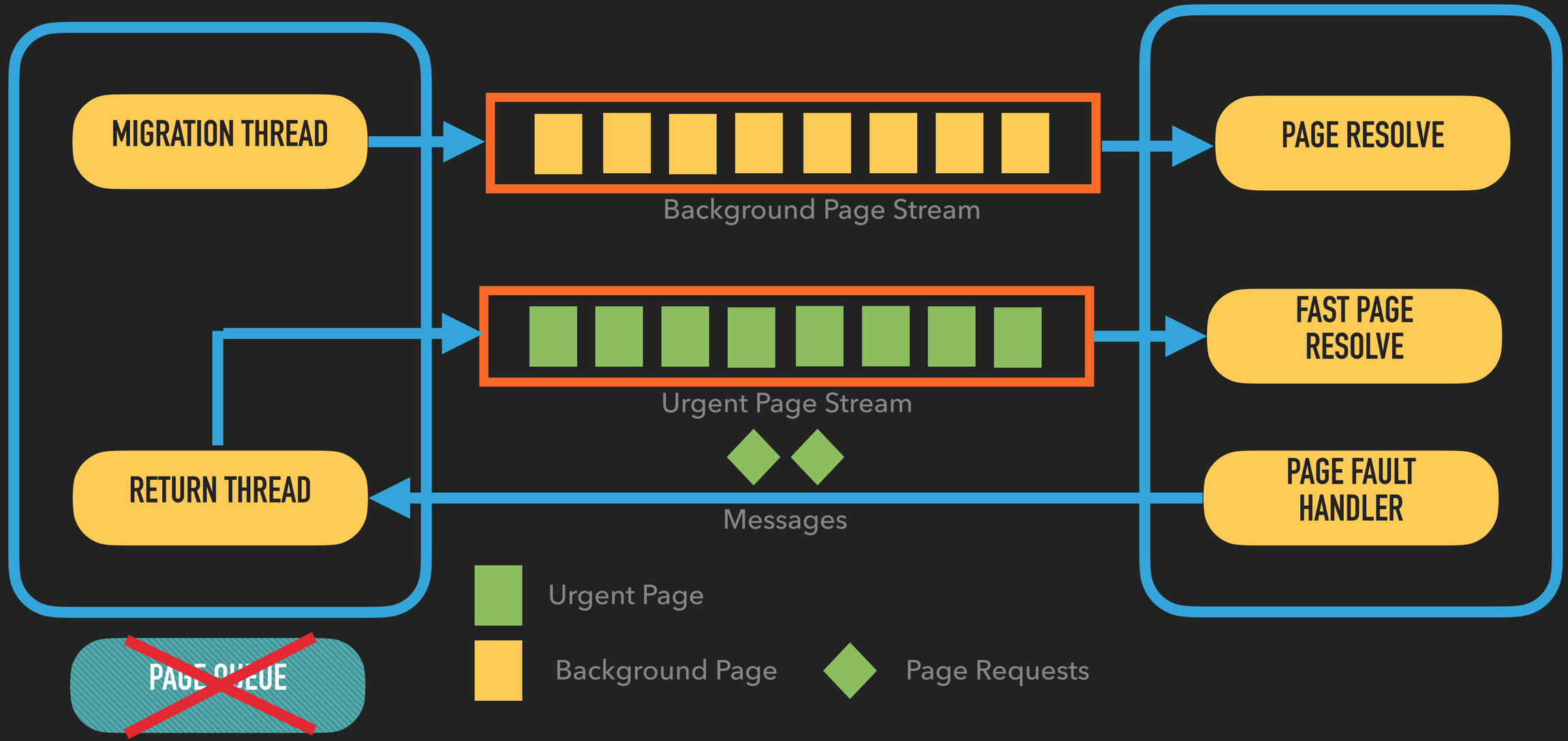
SOLUTION 3 – ???

- Refactor global states into per-channel ones
  - Turning PageSearchStatus into a per-channel structure, one for each channel
- Manage page ownerships, aka:
  When there are >1 threads sending, who should send which page?
  - Who took the bitmap bit (protected by bitmap_mutex)
  - Make sure to release any global lock during sending (e.g. sendmsg() could block)
- Send pages outside migration_thread
  - How about… the return thread????
- Drop page request queue, because we don’t need it anymore!

(A RECAP ON PREVIOUS…)



SOLUTION 3 – REUSE RETURN THREAD TO SEND PAGES



PERFORMANCE NUMBERS

- VM: 20 cpus, 20GB mem, 1 busy random write workload over 18GB
- Test program: mig_mon mm_dirty -m 18000 -p random
  https://github.com/xzpeter/mig_mon
- Measure average page fault latencies
  https://github.com/xzpeter/small-stuffs/blob/master/tools/huge_vm/uffd-latency.bpf
- Results (~50x speedup in 4K average page request latency)
  - Vanilla: 12093 (us)
  - Preempt Full (solution 1+2+3): 229 (us)

DISTRIBUTIONS OF LATENCIES

```
Vanilla
Average: 12093 (us)
@delay_us:
[1]                    1 |                                                    |
[2, 4)                 0 |                                                    |
[4, 8)                 0 |                                                    |
[8, 16)                0 |                                                    |
[16, 32)               1 |                                                    |
[32, 64)               8 |                                                    |
[64, 128)             11 |                                                    |
[128, 256)            14 |                                                    |
[256, 512)            19 |                                                    |
[512, 1K)             14 |                                                    |
[1K, 2K)              35 |                                                    |
[2K, 4K)              18 |                                                    |
[4K, 8K)              87 |@                                                   |
[8K, 16K)           2397 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[16K, 32K)             7 |                                                    |
[32K, 64K)             2 |                                                    |
[64K, 128K)           20 |                                                    |
[128K, 256K)           6 |                                                    |
```

```
Preempt Full
Average: 229 (us)
@delay_us:
[8, 16)                1 |                                                    |
[16, 32)               3 |                                                    |
[32, 64)               2 |                                                    |
[64, 128)          11956 |@@@@@@@@@@                                          |
[128, 256)         60403 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[256, 512)         15047 |@@@@@@@@@@@@                                        |
[512, 1K)            846 |                                                    |
[1K, 2K)              25 |                                                    |
[2K, 4K)              41 |                                                    |
[4K, 8K)             131 |                                                    |
[8K, 16K)             72 |                                                    |
[16K, 32K)             2 |                                                    |
[32K, 64K)             8 |                                                    |
[64K, 128K)            6 |                                                    |
```

FUTURE WORK

- Postcopy preempt part 1 merged in v7.1.0 (including solution 1+2)
  https://lore.kernel.org/qemu-devel/20220707185342.26794-1-peterx@redhat.com/
- Postcopy preempt part 2 RFC posted (including solution 3), during review
  https://lore.kernel.org/qemu-devel/20220829165659.96046-1-peterx@redhat.com/
- Comments welcomed