CS3223 Lecture 1
Data Storage

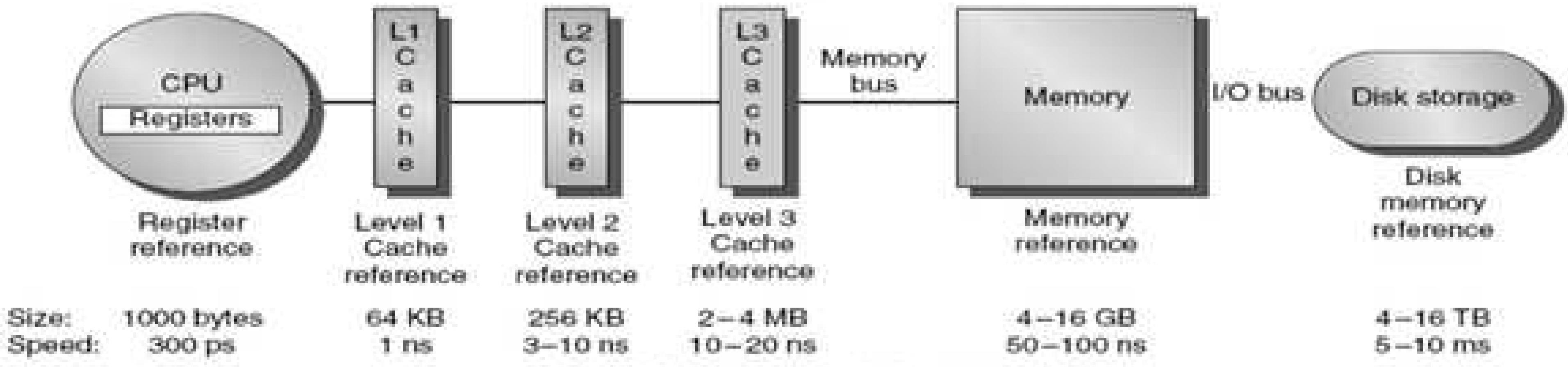
What does a DBMS store?

- Relations
- System catalog (a.k.a. data dictionary) stores metadata about relations
  - Relation schemas - structure of relations, constraints, triggers
  - View definitions
  - Indexes - derived information to speed up access to relations
  - Statistical information about relations for use by query optimizer
- Log files - information maintained for data recovery

Memory Hierarchy

- Primary memory: registers, static RAM (caches), dynamic RAM (physical memory)
- Secondary memory: magnetic disks (HDD), solid-state disks (SSD)
- Tertiary memory: optical disks, tapes
- Tradeoffs:
  - capacity
  - cost
  - access speed
  - volatile vs non-volatile

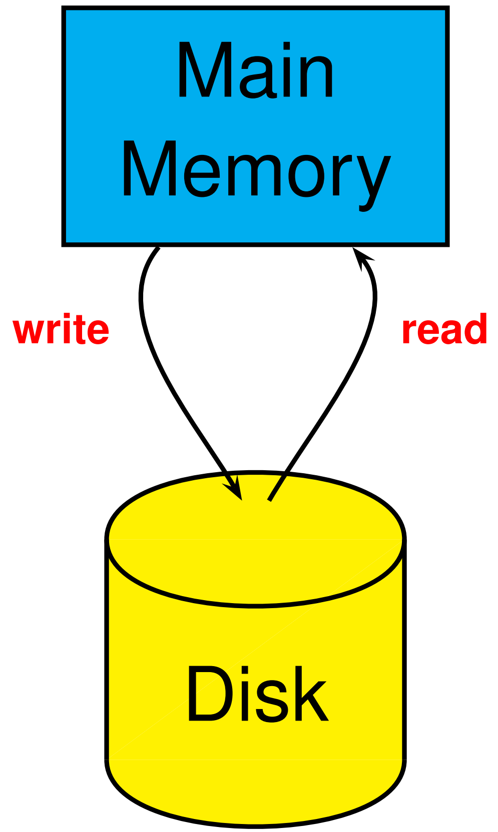
Memory Hierarchy: Tradeoffs



(a) Memory hierarchy for server

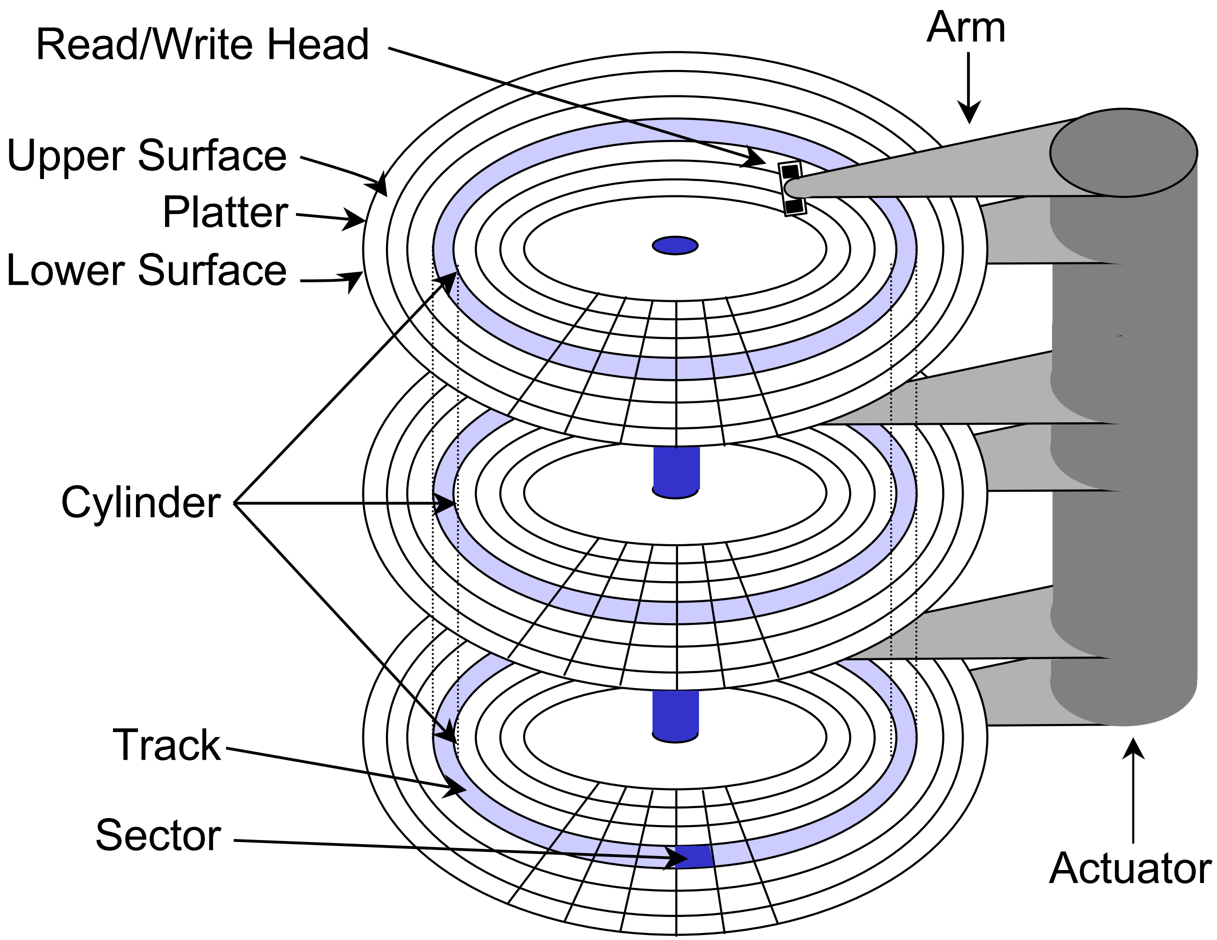
Source: Hennessy & Patterson's Computer Architecture: A Quantitative Approach

DBMS Storage



- DBMS stores data on non-volatile disk for persistence
- DBMS processes data in main memory (RAM)
- Disk access operations:
  - read: transfer data from disk to RAM
  - write: transfer data from RAM to disk

Magnetic Hard-Disk Drive (HDD)

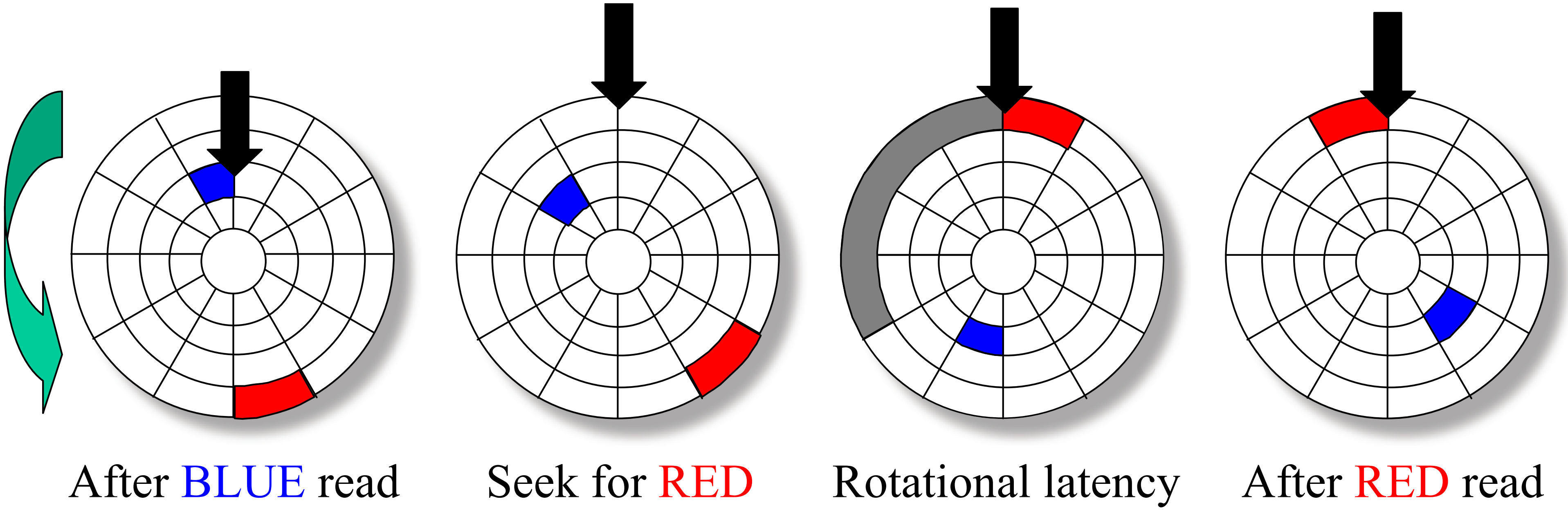


Source: R. Burns' slides on storage systems

Disk Access Time

- Disk access time:
  - command processing time: interpreting access command by disk controller
  - seek time: moving arms to position disk head on track
  - rotational delay: waiting for block to rotate under head
  - transfer time: actually moving data to/from disk surface
  - access time = seek time + rotational delay + transfer time (command processing time is considered negligible)
- Response time for disk access = queuing delay + access time

Components of Disk Access Time

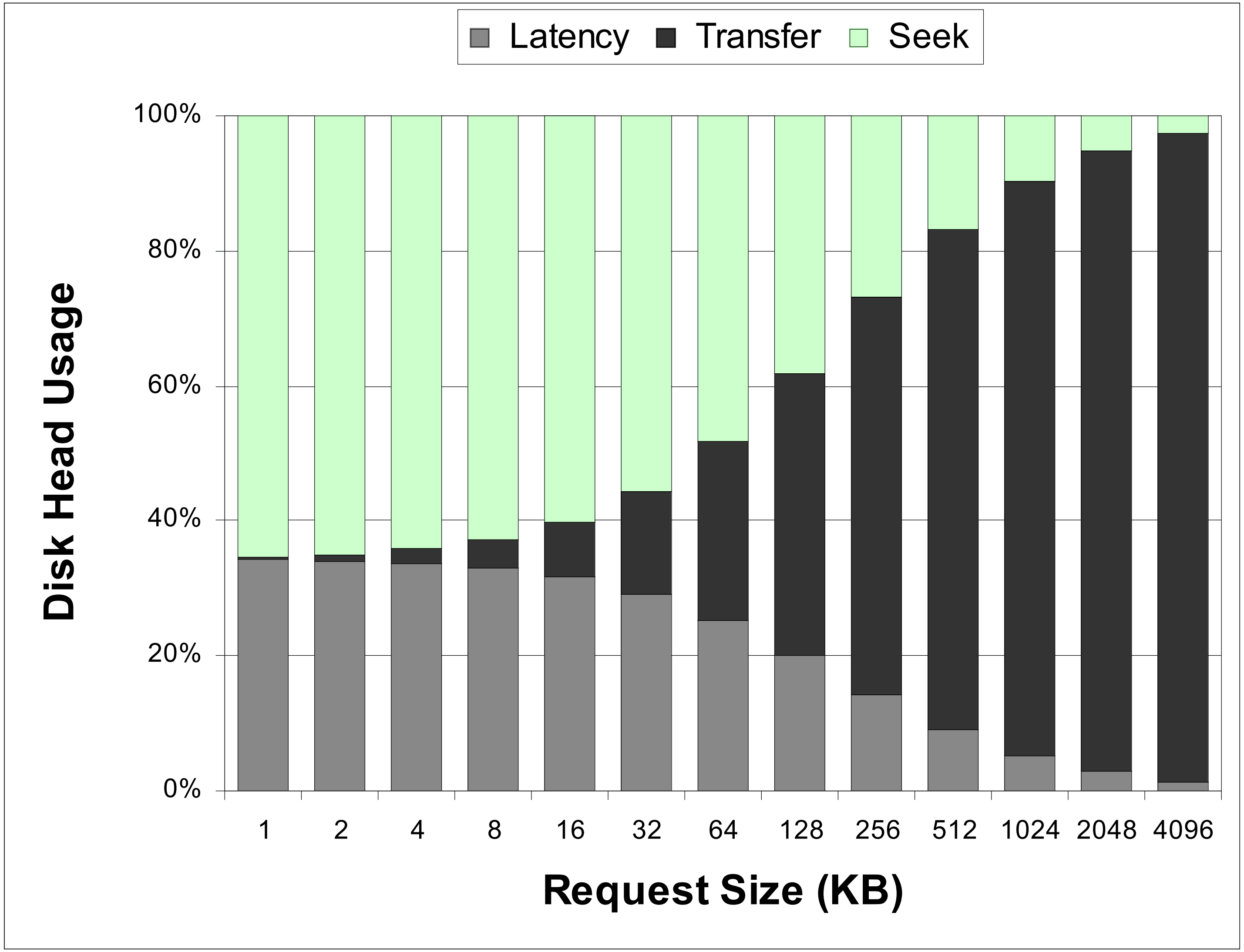


Source: R. Burns' slides on storage systems

Disk Access Time (cont.)

- Seek time
  - avg. seek time: 5-6 ms
- Rotational delay (or rotational latency)
  - Depends on rotation speed - measured in rotations per minute (RPM)
  - Average rotational delay = time for $\frac{1}{2}$ revolution
  - Example: For 10000 RPM, avg. rotational delay = 0.5 (60 / 10000) = 3 ms
- Transfer time
  - n = number of requested sectors on same track
  - transfer time = $n \times \frac{\text{time for one revolution}}{\text{number of sectors per track}}$
  - avg. sector transfer time: 100-200 μs
- Sequential vs random I/O

Where does the disk access time go?



Source: R. Burns' slides on storage systems

Solid-State Drive (SSD)

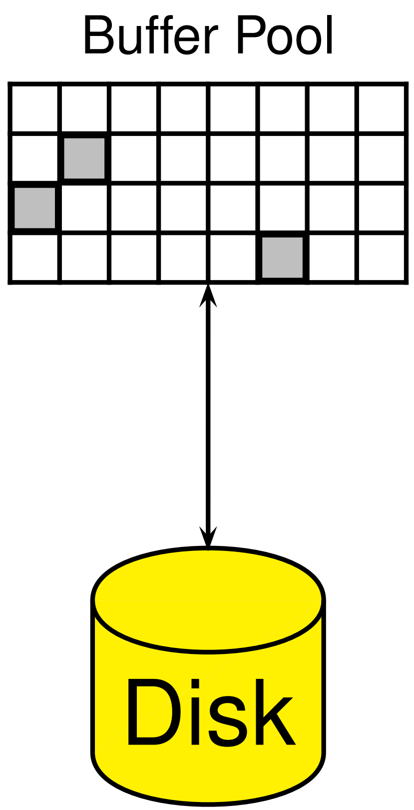
- Built with NAND flash memory without any mechanical/moving parts
- Faster random read: latency $\approx$ 20 - 100 μs
- Higher data transfer rate
  - SATA interface: $\approx$ 500 MB/s
  - NVMEe PCIe interface: up to 3 GB/s
- Lower power consumption
- Write latency $\approx$ 100 μs
- Disadvantages:
  - Update to a page requires erasure of multiple pages ($\approx$ 5 ms) before overwriting page
  - Limited number of times a page can be erased ($\approx 10^5$ - 10^6)

Storage Manager Components

file & access methods manager
buffer pool manager
disk space manager

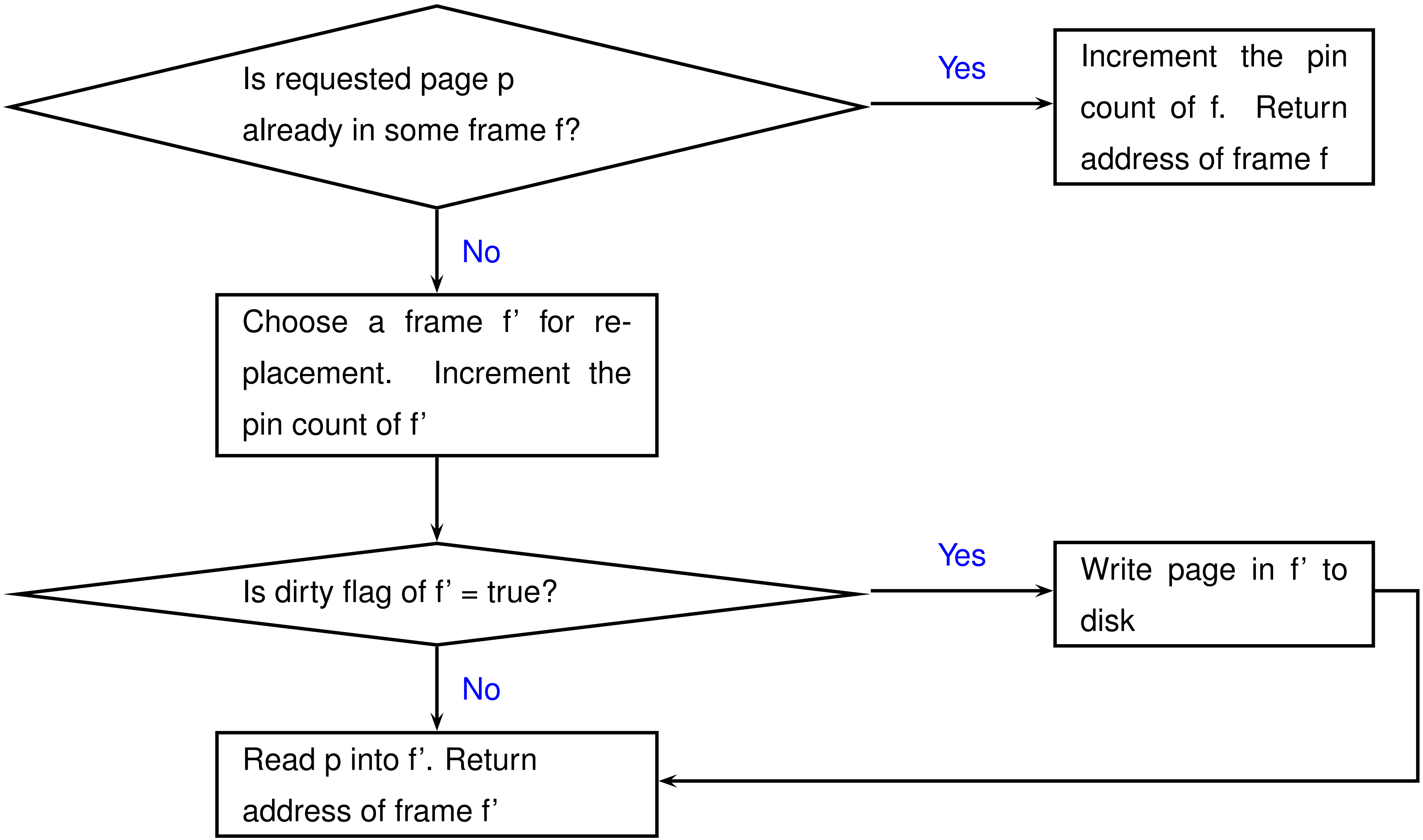
- Data is stored & retrieved in units called disk blocks (or pages)
  - Each block = sequence of one or more contiguous sectors
- Files & access methods layer (aka file layer) - deals with organization and retrieval of data
- Buffer Manager - controls reading/writing of disk pages
- Disk Space Manager - keeps track of pages used by file layer

Buffer Manager



- Buffer pool = main memory allocated for DBMS
- Buffer pool is partitioned into block-sized pages called frames
- Clients of buffer pool can
  - request for a disk page to be fetched into buffer pool
  - release a disk page in buffer pool
- A page in the buffer is dirty if it has been modified & not updated on disk
- Two variables are maintained for each frame in buffer pool:
  - pin count - number of clients using page (initialized to 0)
  - dirty flag - whether page is dirty (initialized to false)

Buffer Manager: Handling a request for page p



Buffer Manager (cont.)

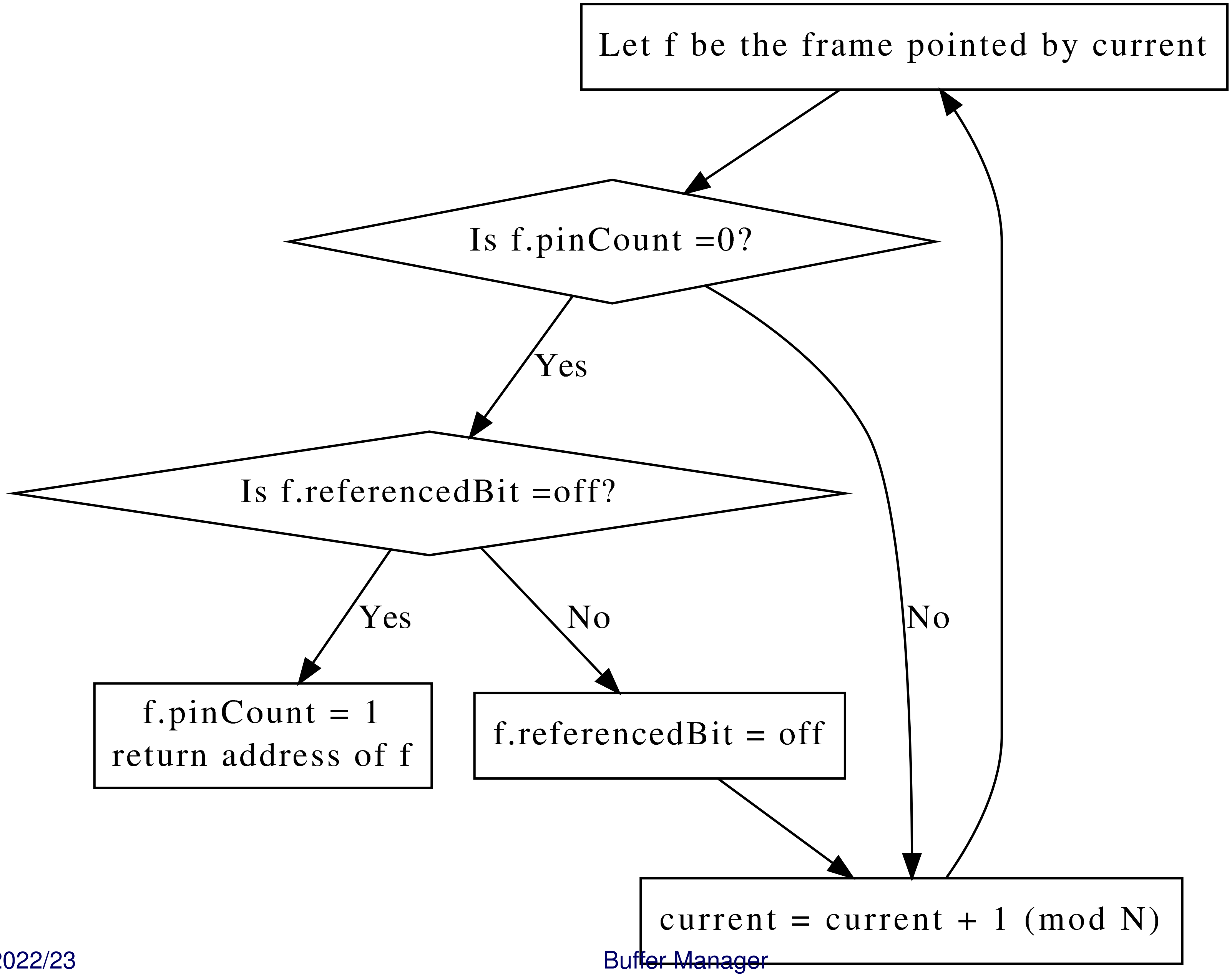
- Incrementing pin count is called pinning the requested page in its frame
- Decrementing the pin count is called unpinning the page
- When unpinning a page, its dirty flag should be updated to true if the page is dirty
- A page in buffer can be replaced only when its pin count is 0
- Before replacing a buffer page, it needs to be written back to disk if its dirty flag is true
- Buffer manager coordinates with transaction manager to ensure data correctness and recoverability

Buffer Manager: Replacement Policies

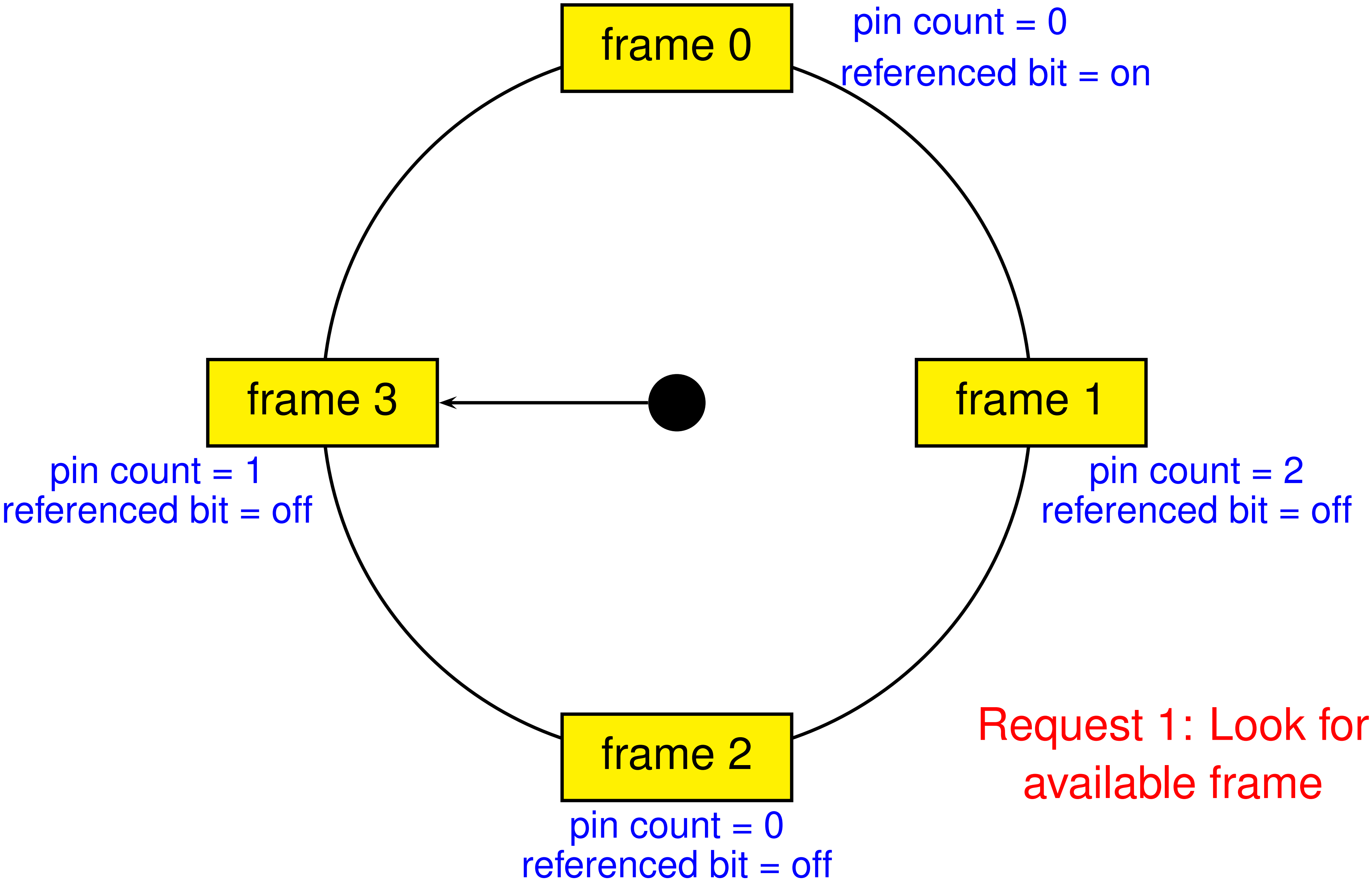
- Replacement Policy: decide which unpinned page to replace
  - Random
  - First In First Out (FIFO)
  - Most Recently Used (MRU)
  - Least Recently Used (LRU)
    - ★ Uses a queue of pointers to frames with pin count = 0
  - Clock - a variant of LRU
    - ★ current variable - points to some buffer frame
    - ★ Each frame has a referenced bit - turns on when its pin count becomes 0
    - ★ Replace a page that has referenced bit off & pin count = 0
  - etc.

Clock Replacement Policy

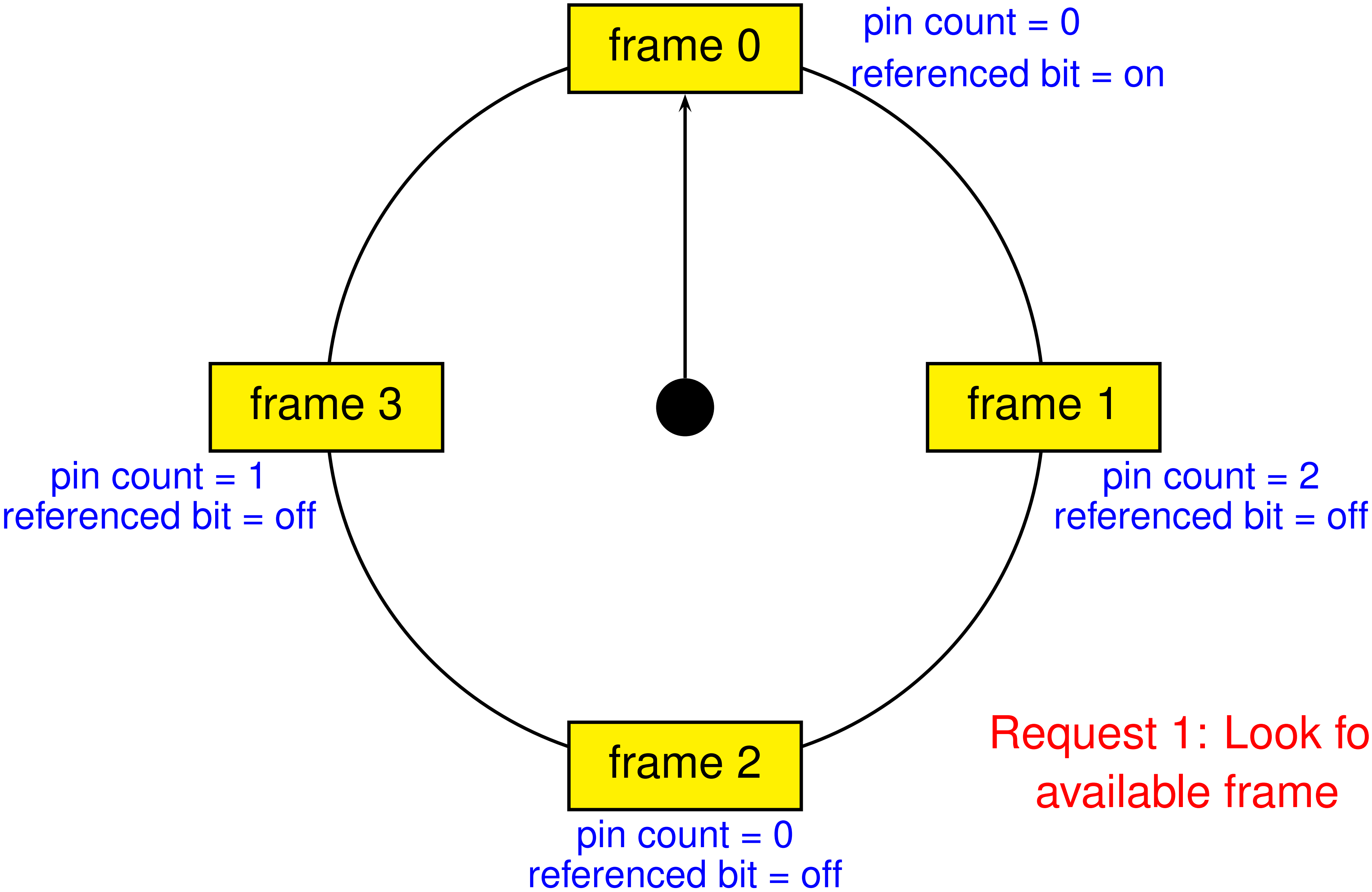
N = number of frames in buffer pool



Clock Replacement Policy: Example

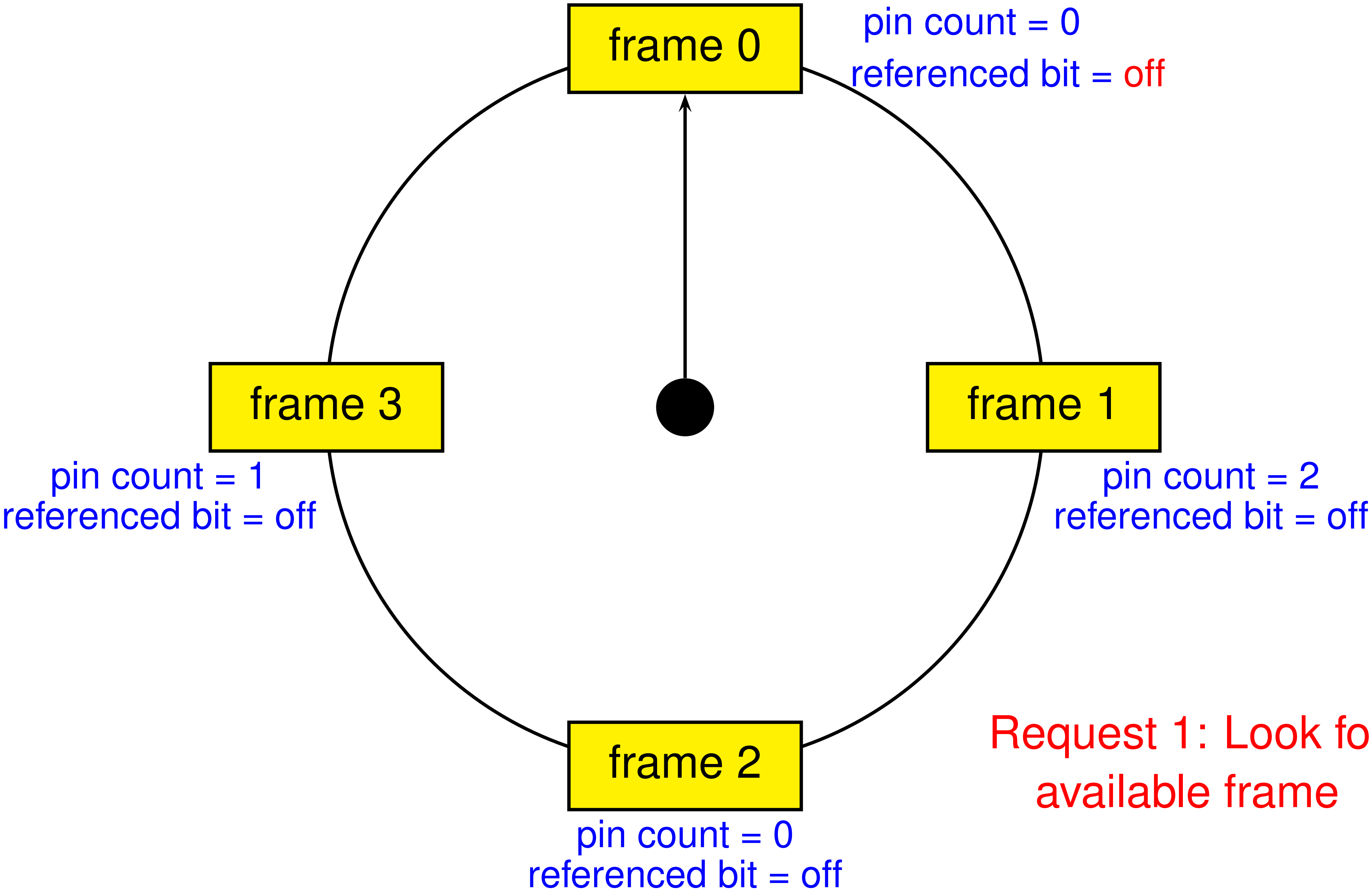


Clock Replacement Policy: Example



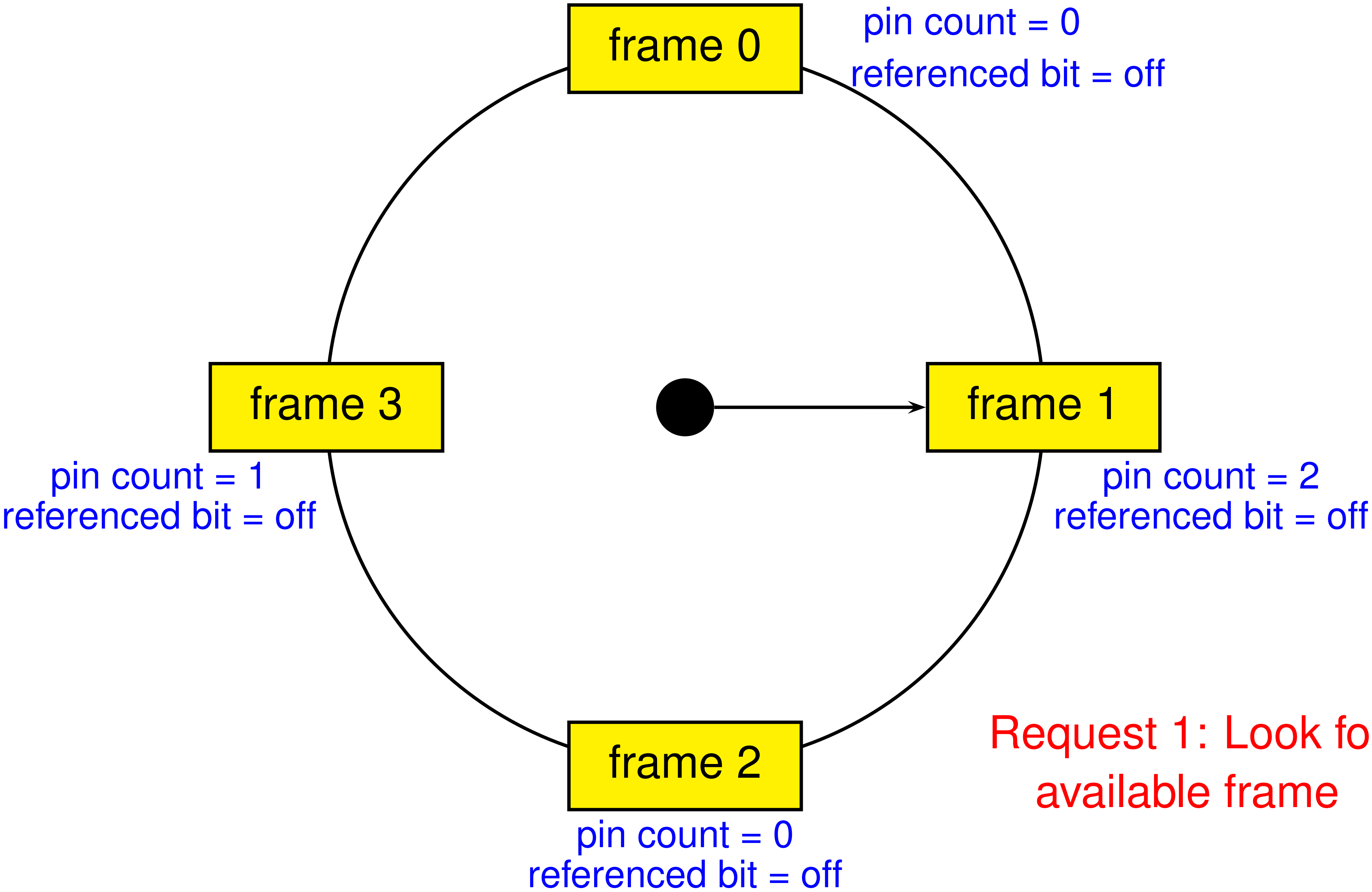
Request 1: Look for available frame

Clock Replacement Policy: Example



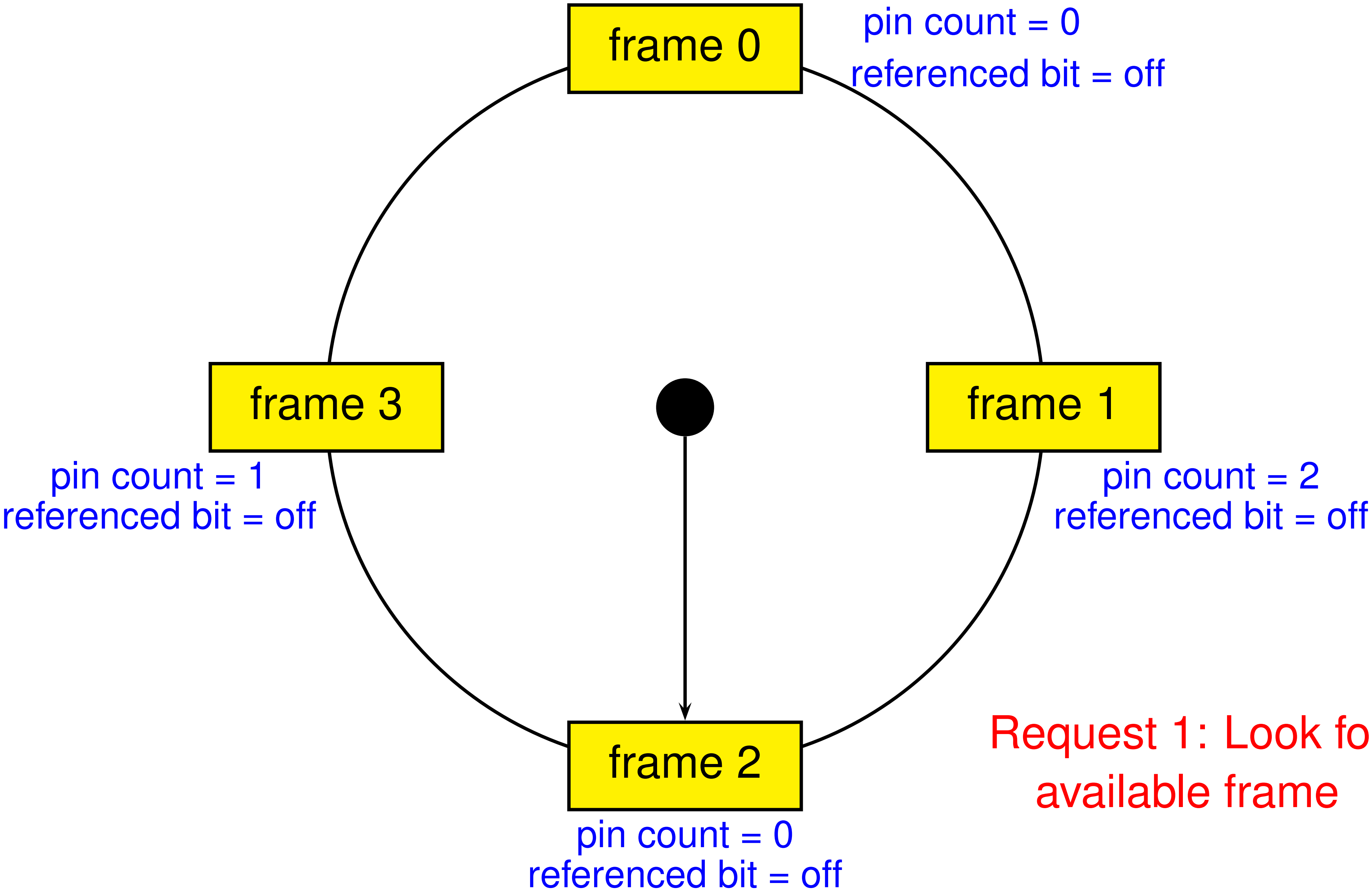
Request 1: Look for available frame

Clock Replacement Policy: Example



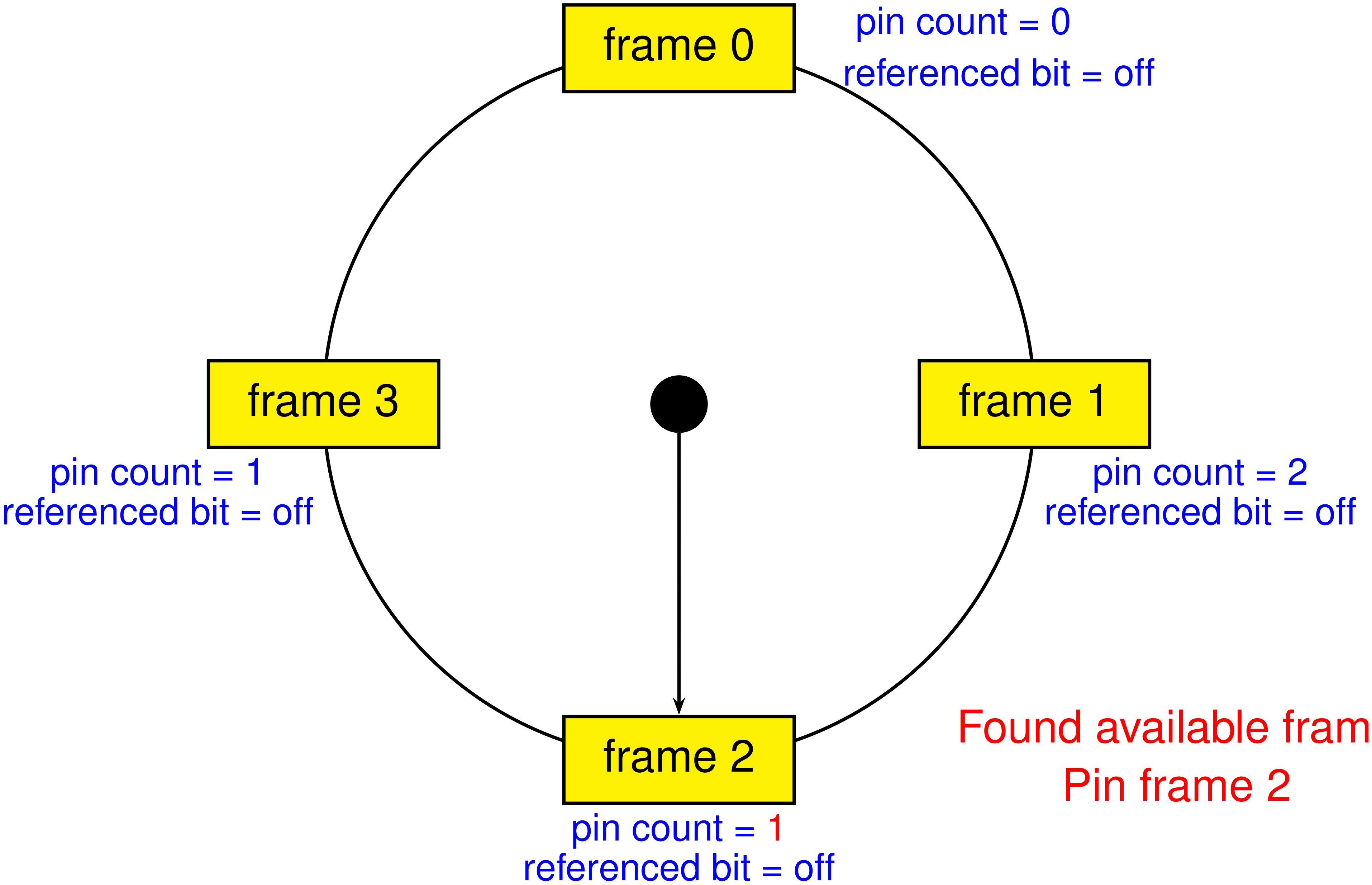
Request 1: Look for available frame

Clock Replacement Policy: Example



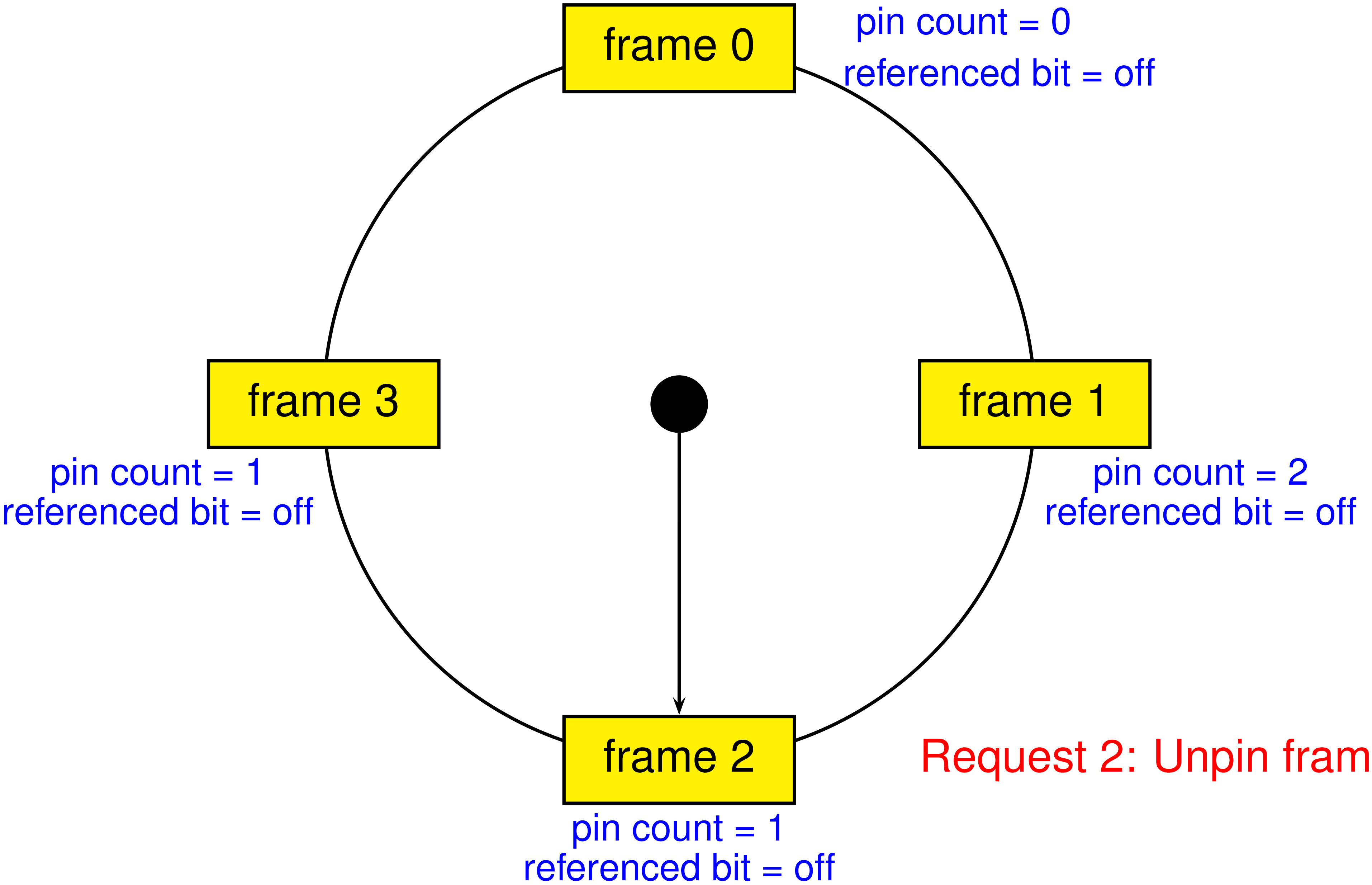
Request 1: Look for available frame

Clock Replacement Policy: Example



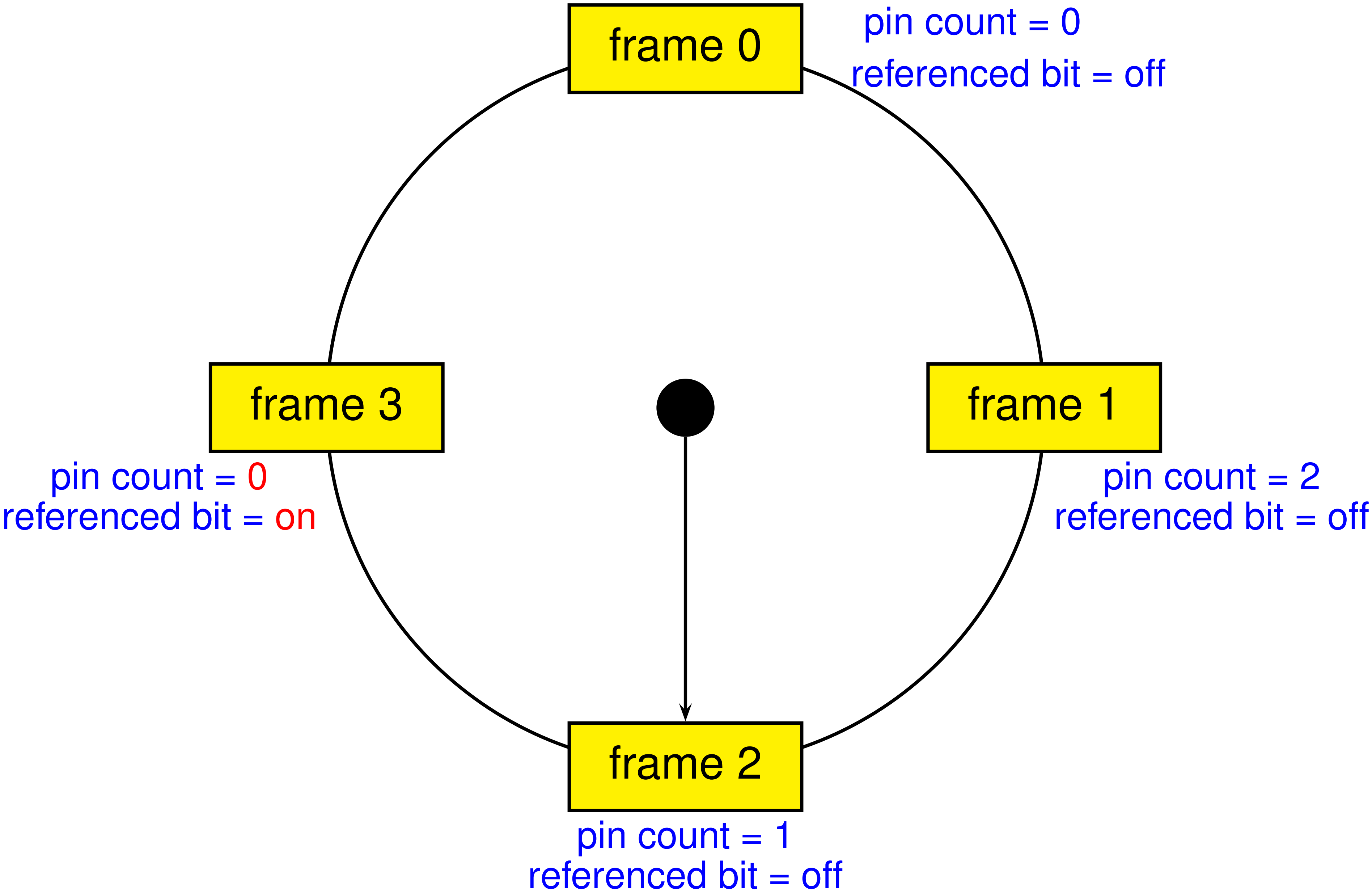
Found available frame
Pin frame 2

Clock Replacement Policy: Example

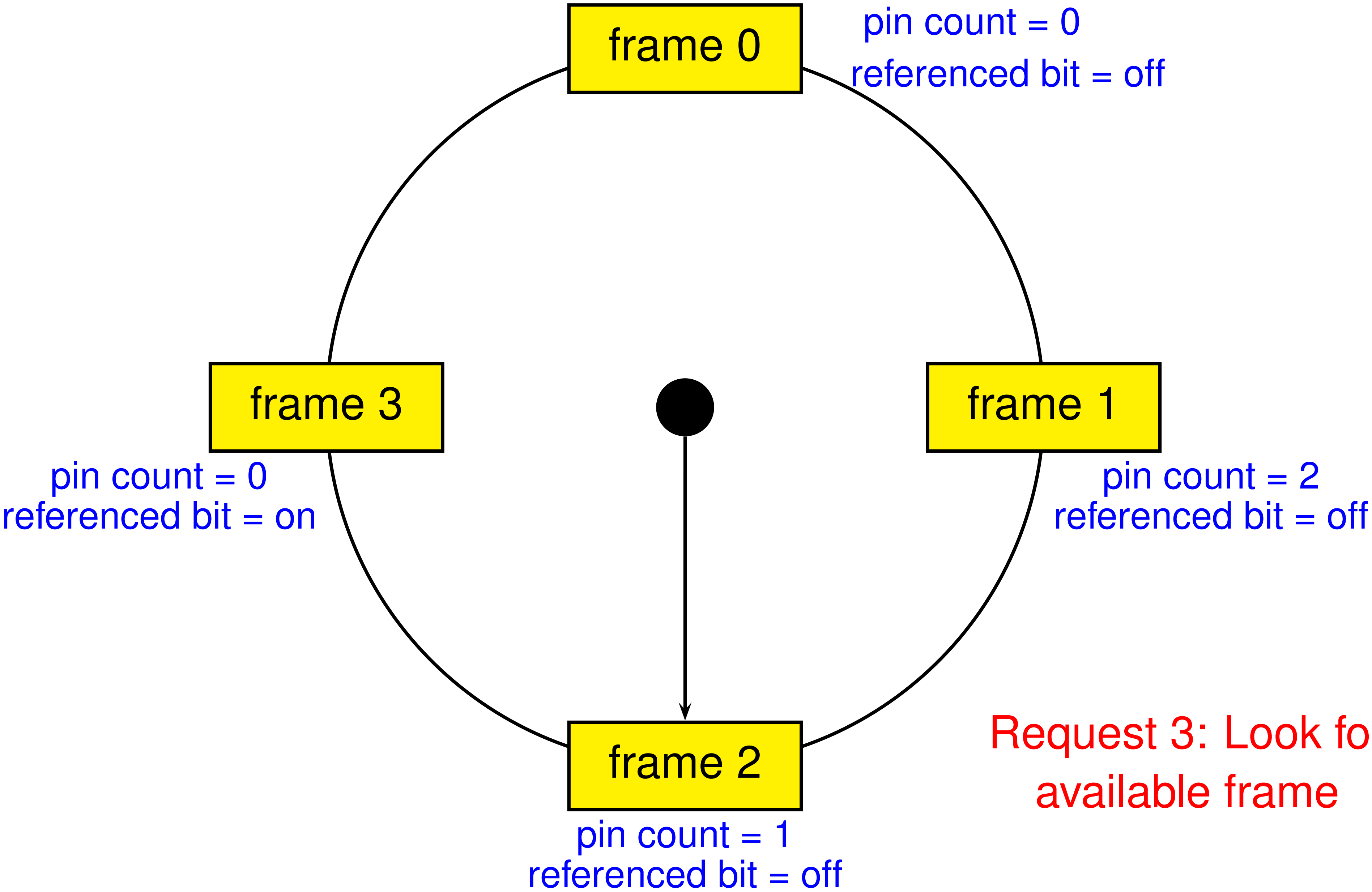


Request 2: Unpin frame 3

Clock Replacement Policy: Example

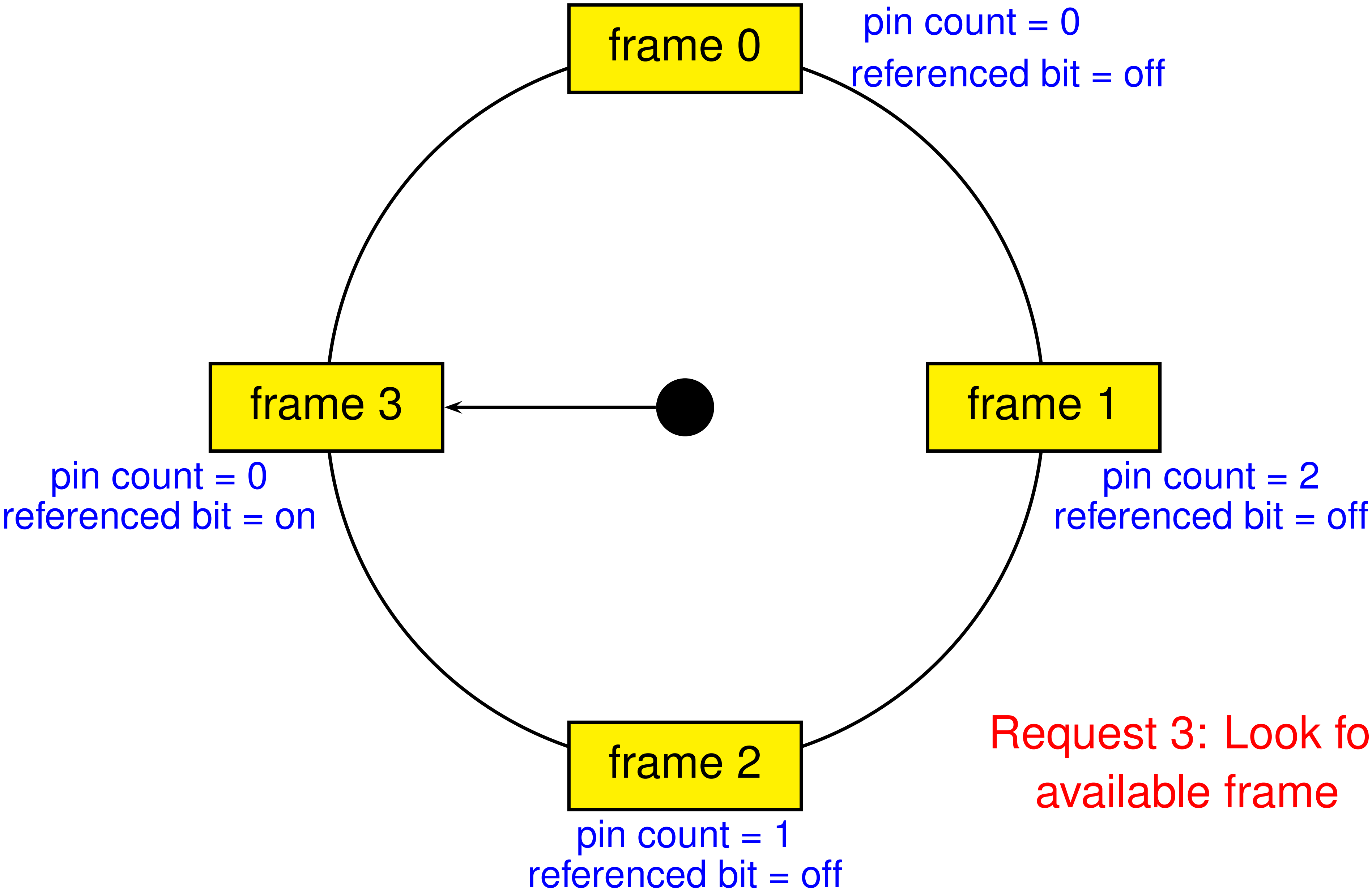


Clock Replacement Policy: Example



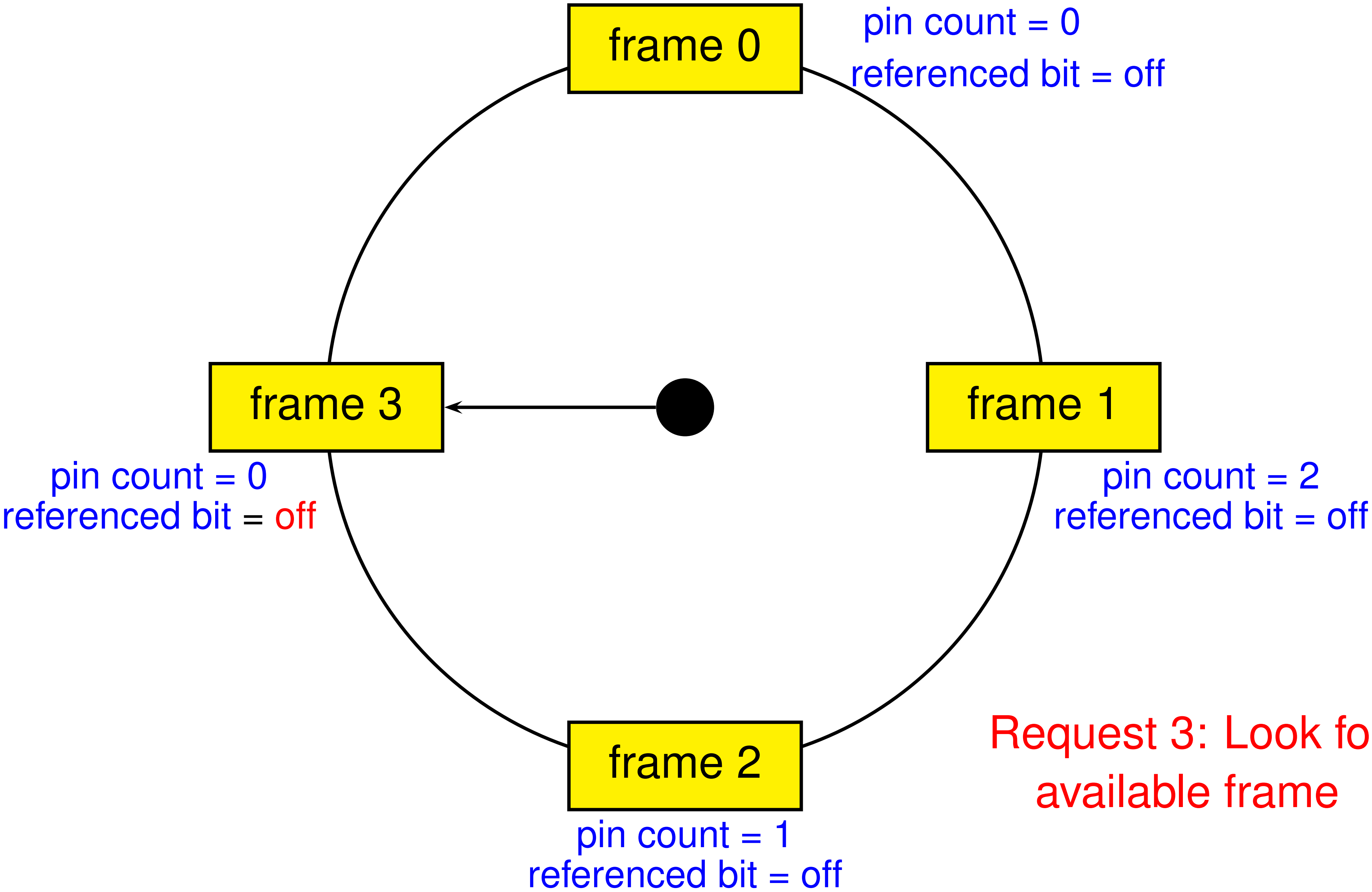
Request 3: Look for available frame

Clock Replacement Policy: Example



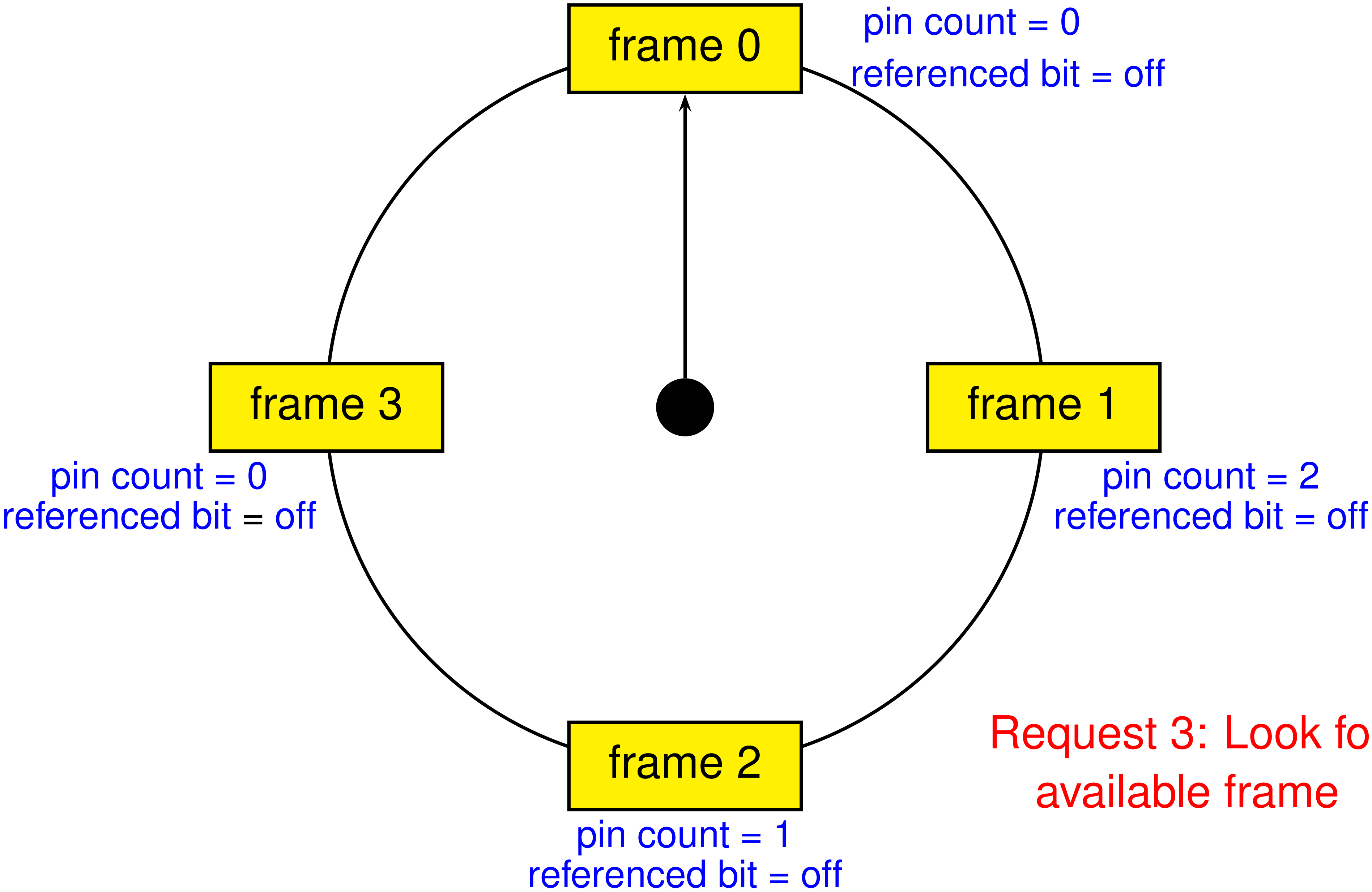
Request 3: Look for available frame

Clock Replacement Policy: Example



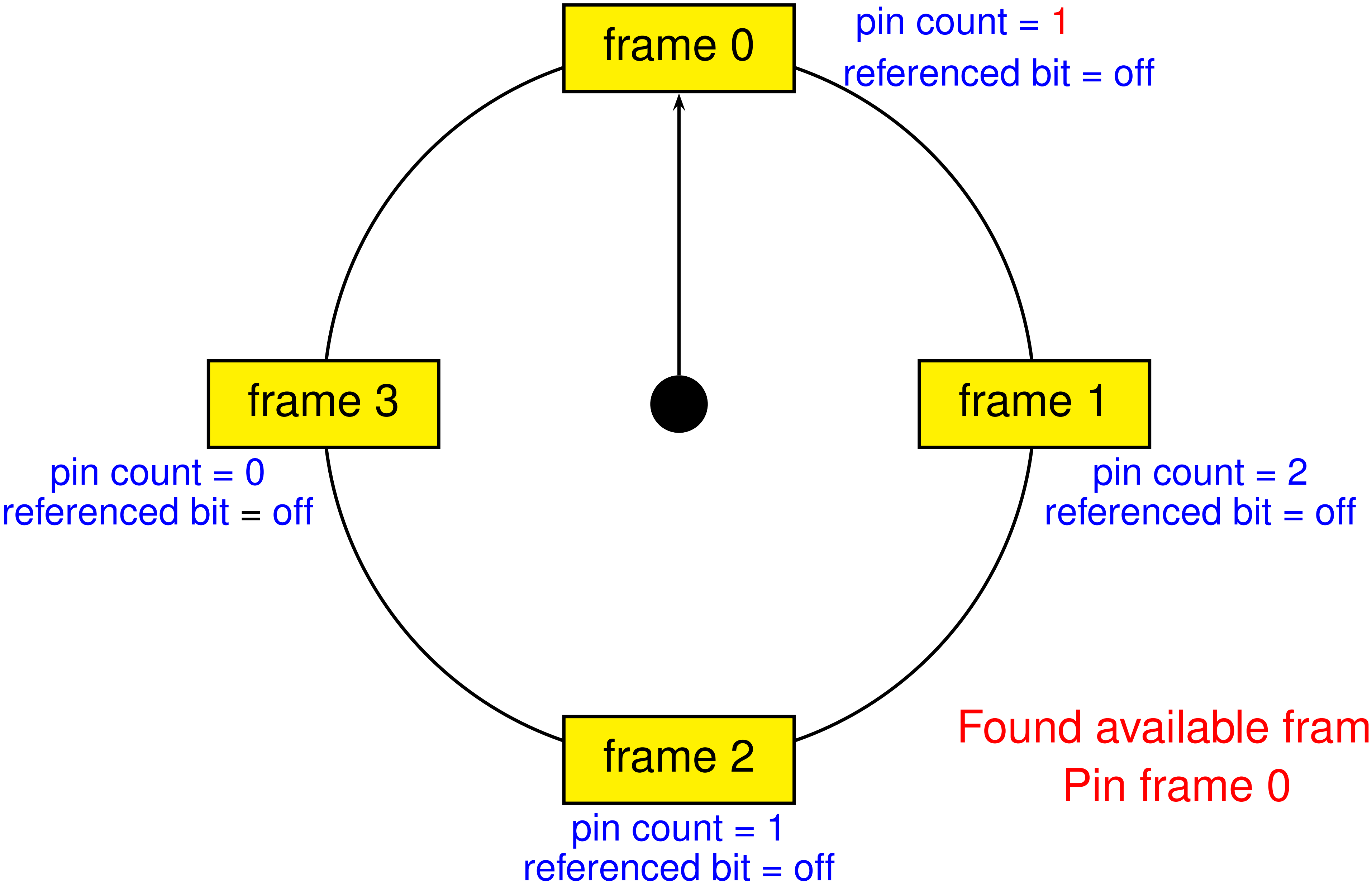
Request 3: Look for available frame

Clock Replacement Policy: Example



Request 3: Look for available frame

Clock Replacement Policy: Example

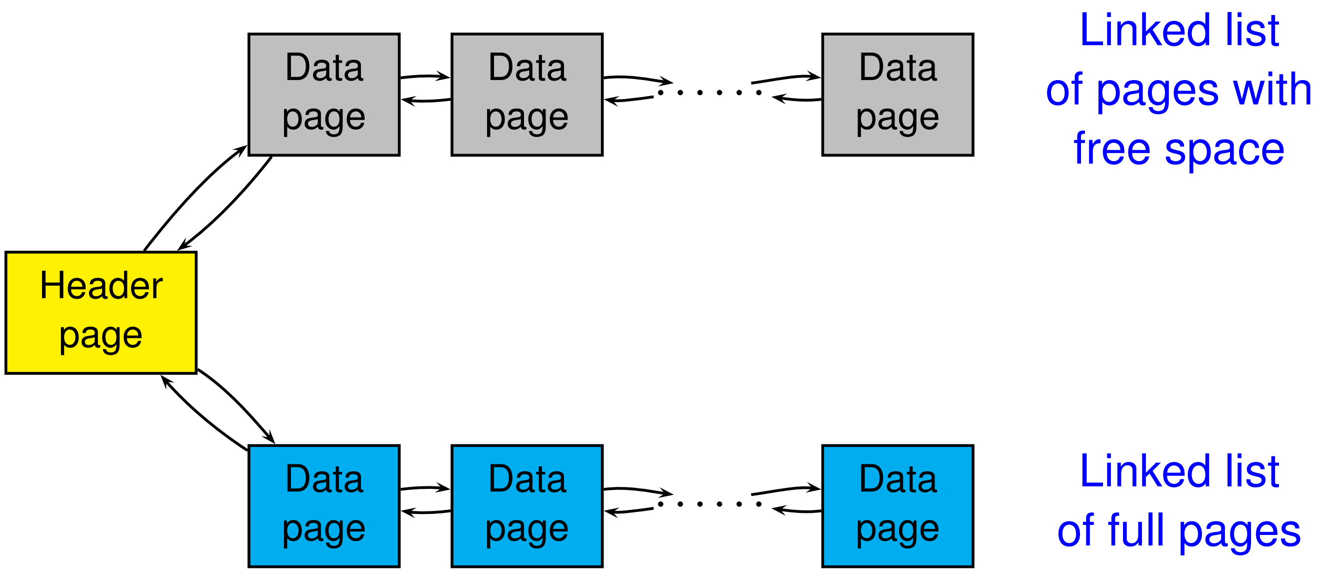


Found available frame
Pin frame 0

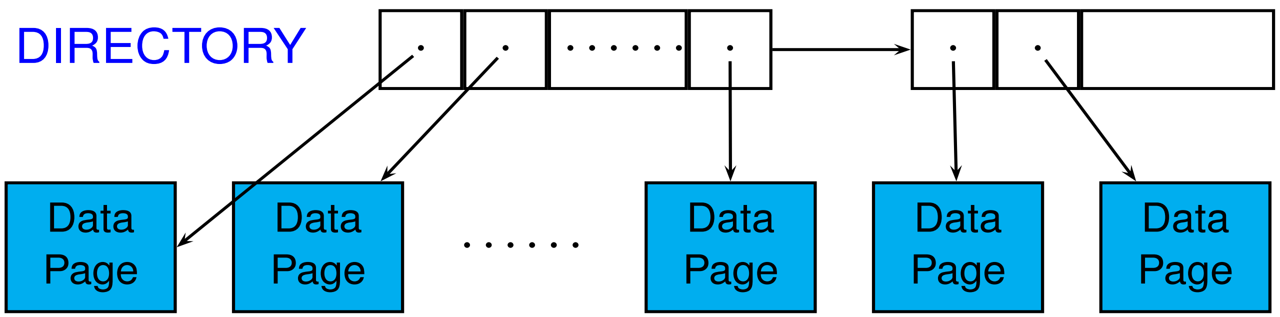
Files

- File abstraction
  - Each relation is a file of records
  - Each record has a unique record identifier called RID (or TID)
  - Common file operations:
    - ★ create a file
    - ★ delete a file
    - ★ insert a record
    - ★ delete a record with a given RID
    - ★ get a record with a given RID
    - ★ scan all records
- File organization = method of arranging data records in a file that is stored on disk
  - Heap file: unordered file
  - Sorted file records are ordered on some search key
  - Hashed file: records are located in blocks via a hash function

Heap File Implementations



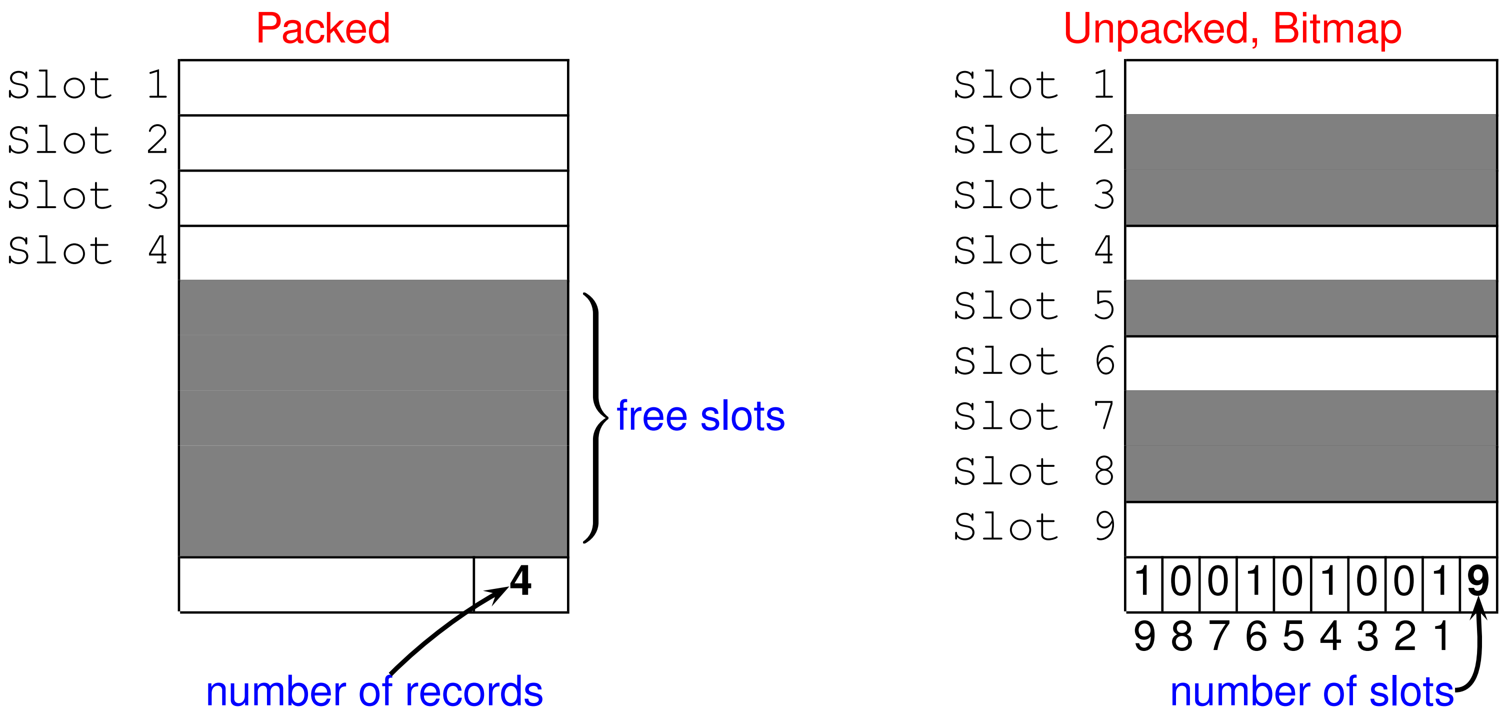
Linked List Implementation



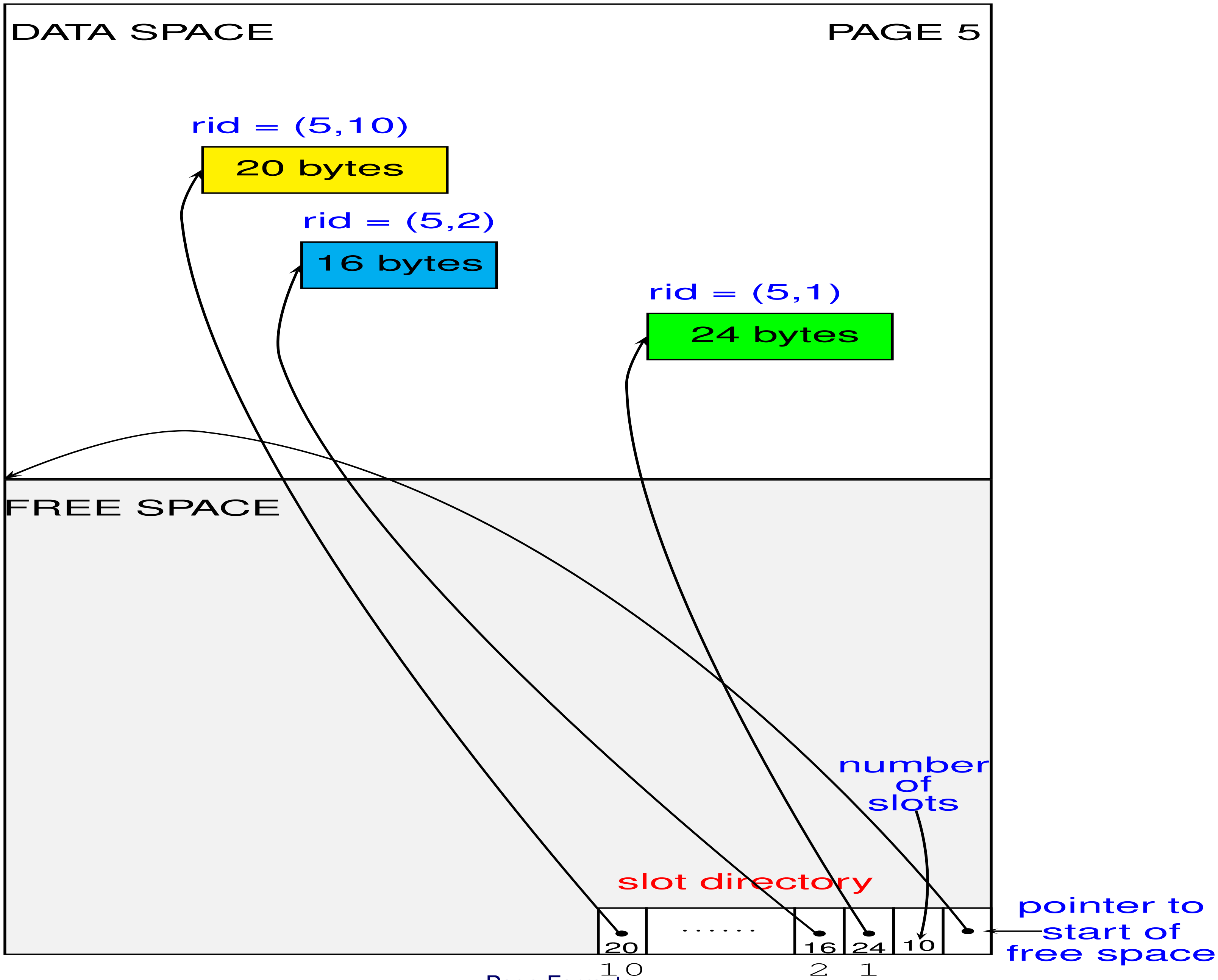
Page Directory Implementation

Page Formats

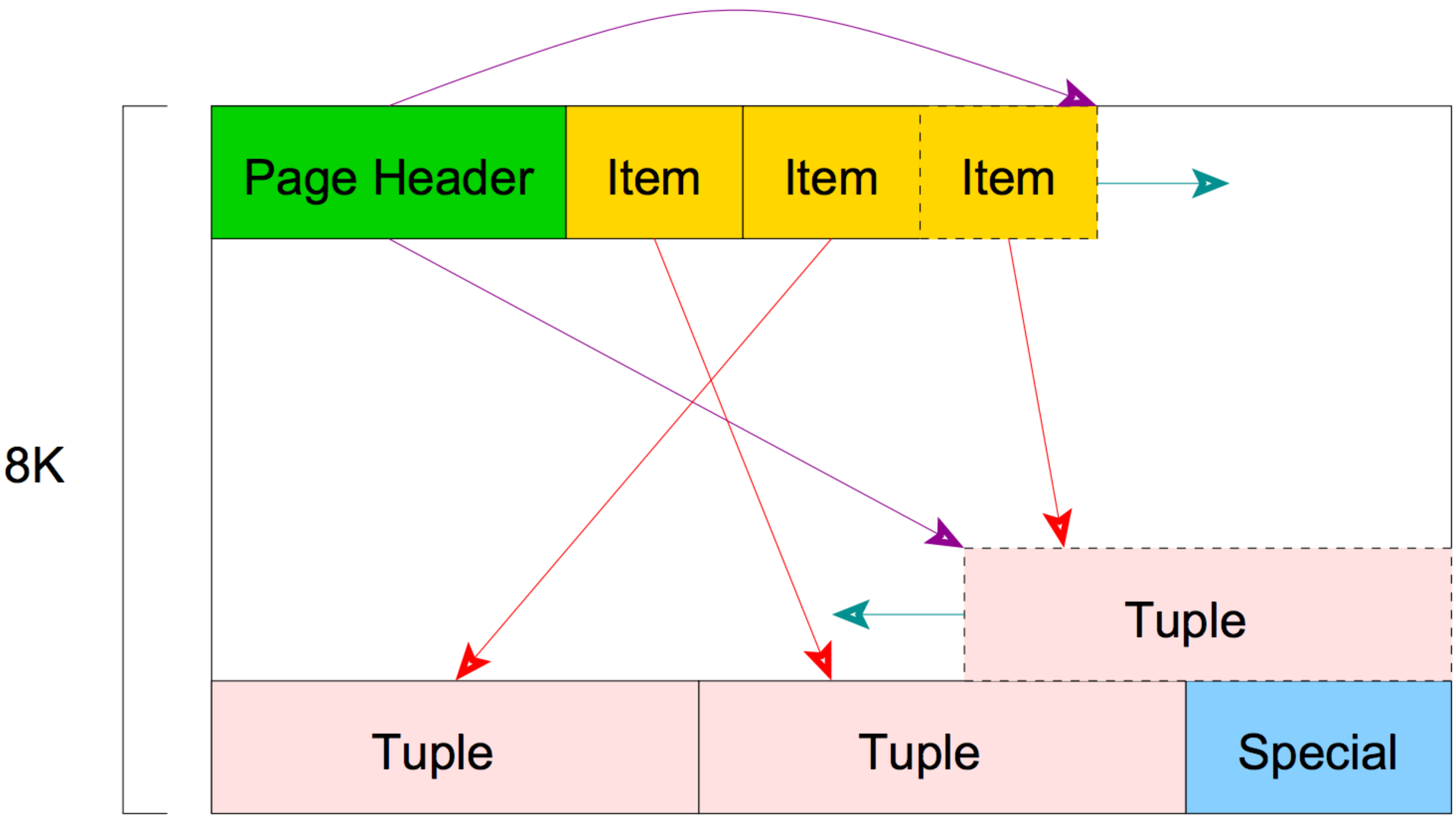
- How are records organized within a page?
- RID = (page id, slot number)
- Fixed-Length Records
  - Packed organization: Store records in contiguous slots
  - Unpacked organization: Uses a bit array to maintain free slots



Variable-Length Records: Slotted Page Organization



PostgreSQL's Slotted Page Organization



Source: B. Momjian's slides on PostgreSQL internals

Record Formats

- How to organize fields within a record?
- Fixed-Length Records
  - Fields are stored consecutively

F1	F2	F3	F4

- Variable-Length Records
  - Delimit fields with special symbols

F1	$	F2	$	F3	$	F4

  - Use an array of field offsets

o_1	o_2	o_3	o_4	F1	F2	F3	F4

Each o_i is an offset to beginning of field Fi