PRINCIPLES OF DATA ORGANISATION

Linear (Litwin) hashing

MOTIVATION

- Key, pointer pairs ~ index
- Hashed file organization
- Dynamic hashing
  - More records can be added
- Collapsing a trie
- Issue: Fagin's directory had to be doubled

LINEAR HASHING

- Litwin 1980, Enbody & Du 1988
- Directory-less scheme
  - No need to double the directory
  - No level of indirection
    - We need a continuous address space in the secondary memory
- Principal idea
  - Avoid doubling of the directory
  - Let us add one page after a pre-specified condition
    - E.g., overflow or given number of inserts (bucket load factor)
    - The space grows linearly – one page after another
  - If we find ourselves in ***i***-th step/iteration, then after $\mathbf{2}^{\boldsymbol{i}}$ insertions we get into ***i* + 1** iteration

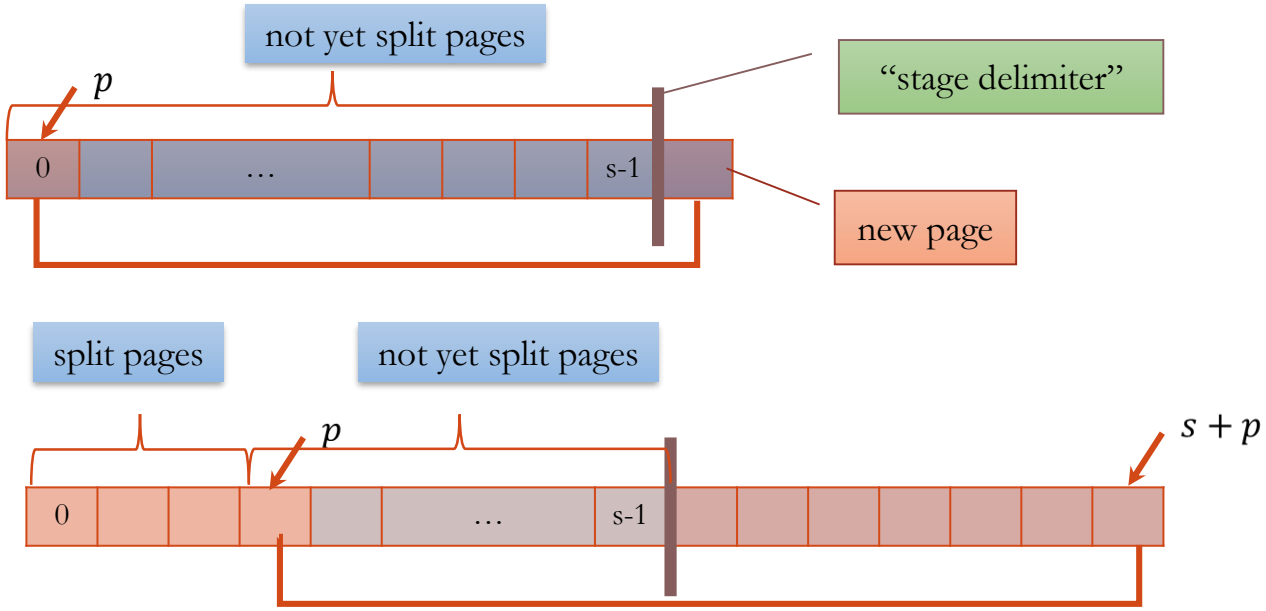
LINEAR HASHING

- Expensive expansion process is divided into stages
- Stage ***d*** starts when the number of pages is $s = 2^d$ and ends when the number reaches 2^{d+1}
  - 0. stage = 1 page
  - 1. stage = 2 pages
  - 2. stage = 4 pages
  - …
  - ***d*** = the number of bits to be used to address all pages in a given stage
    - 0 bits for 1 page
    - 1 bit for 2 pages
    - 2 bits for 4 pages
    - …
- A (split) pointer ***p*** is used to point to pages 0 … 2^d
  - The purpose of ***p*** is to identify the next page to be split

LINEAR HASHING

- At the beginning of stage d, p points to page **0**
- After every split operation it is increased by **1** (moves to the next page)
  - If a page overflows before it is its time to split, overflow pages need to be utilized
  - The growth of the primary file is linear
- When splitting, the new page will be at position $p+s$
- Records from page p (and overflow pages) will be distributed between p and $p+s$ using $h_{d+1}(k)$
  - We use one more bit to distribute the data
- At each stage we have two types of hash functions
  - for pages already split
  - for pages not yet split
- When we enter a new stage, we move pointer p to the start

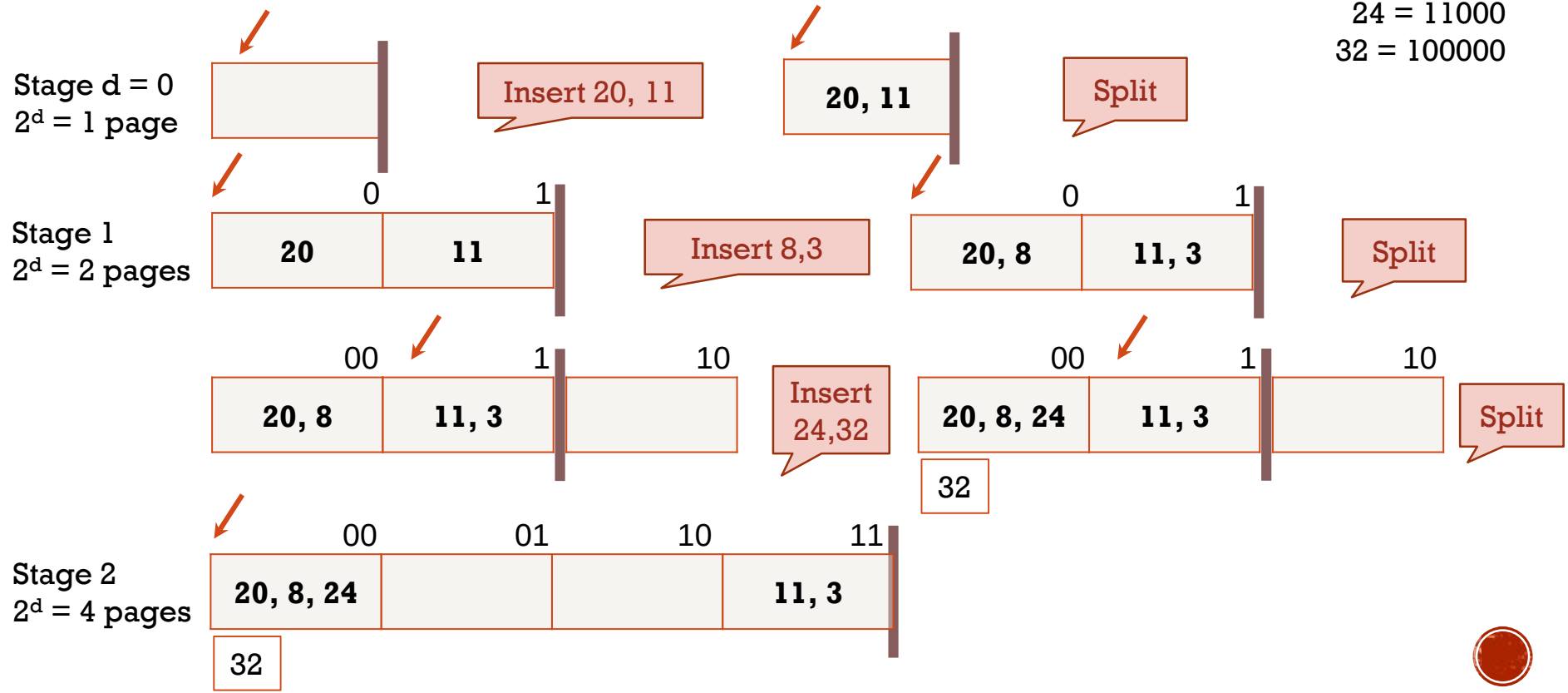
LINEAR HASHING – STRUCTURE



EXAMPLE

b = 3
split: after 2 inserts

20 = 10100
11 = 1011
8 = 1000
3 = 11
24 = 11000
32 = 100000



LINEAR HASHING – ADDRESSING

Unlike directory-based hashing, address of a record has to be computed.
Pages left of ***p*** are already split and therefore need one more bit for addressing than pages right of ***p***.

```
ADDR GetAddres ( KEY k, int cnt_pages ) {
  d = floor(log(cnt_pages, 2));
  s = exp(2, d);
  p = cnt_pages % s;

  addr = h(k) % s;
  if (addr < p) addr = h(k) % exp(2, d + 1);

  return addr;
}
```

LINEAR HASHING – SPLITTING

Uncontrolled splitting

- Page pointed to by ***p*** is split after a given number of insertions
- Page pointed to by ***p*** is split when any page overflows

Controlled splitting

- Splitting occurs when the utilization of page pointed to by **p** reaches a threshold, e.g. 80%

OVERFLOW HANDLING POLICY

- Problem: some pages may overflow, but we split some other page
- Overflow handling:
  - One global overflow area
  - One overflow page for each page
  - One buddy page for each page