



PERCONA

TECH DAYS

INNODB MERGE/SPLIT PAGES

Marco Tusa
September 2020

About Me

- Open source enthusiast
- MySQL tech lead; principal architect
- Working in DB world over 33 years
- Open source developer and community contributor



WHY WE SHOULD CARE?

- MySQL/InnoDB constantly performs SPLIT and MERGE operations
- We have very limited visibility of them.
- The sad story is there is also very little we can do to optimize this on the server side using parameters or some other magic.
- But the good news is there is A LOT that can be done at design time.
 - Use a proper Primary Key and design a secondary index, keeping in mind that you shouldn't abuse of them.
 - Plan proper maintenance windows on the tables that you know will have very high levels of inserts/deletes/updates.

DISCLAIMER

Mumbo

Jumbo

ahead



ADVANCED

THE MATRYOSHKA DOLL

```
[split]>select id, uuid,processid,points, date, active,time, substring(short,1,4) short,substring(longc,1,4) longc from tbinc limit 5 ;
```

id	uuid	processid	points	date	active	time	short	longc
1	1c0bc1cc-ef69-11ea-ad30-9eca1c2a7efa	127	2707	2020-04-01	1	2020-09-05 13:15:30	That	Whic
2	1c0bc1b8-ef69-11ea-ad30-9eca1c2a7efa	136	1810	2020-04-01	1	2020-09-05 13:15:30	That	All
3	1c0bc172-ef69-11ea-ad30-9eca1c2a7efa	163	4752	2020-04-01	0	2020-09-05 13:15:30	That	When
4	1c0bc17c-ef69-11ea-ad30-9eca1c2a7efa	71	3746	2020-04-01	1	2020-09-05 13:15:30	That	87F
5	1c5b0002-ef69-11ea-ad30-9eca1c2a7efa	14	2339	2020-04-01	0	2020-09-05 13:15:31	That	adve

THE MATRYOSHKA DOLL

A well-known statement
“Inside InnoDB all is an INDEX”

What it means?

In the examples we will use:

Schema: split

Tables:

tbinc

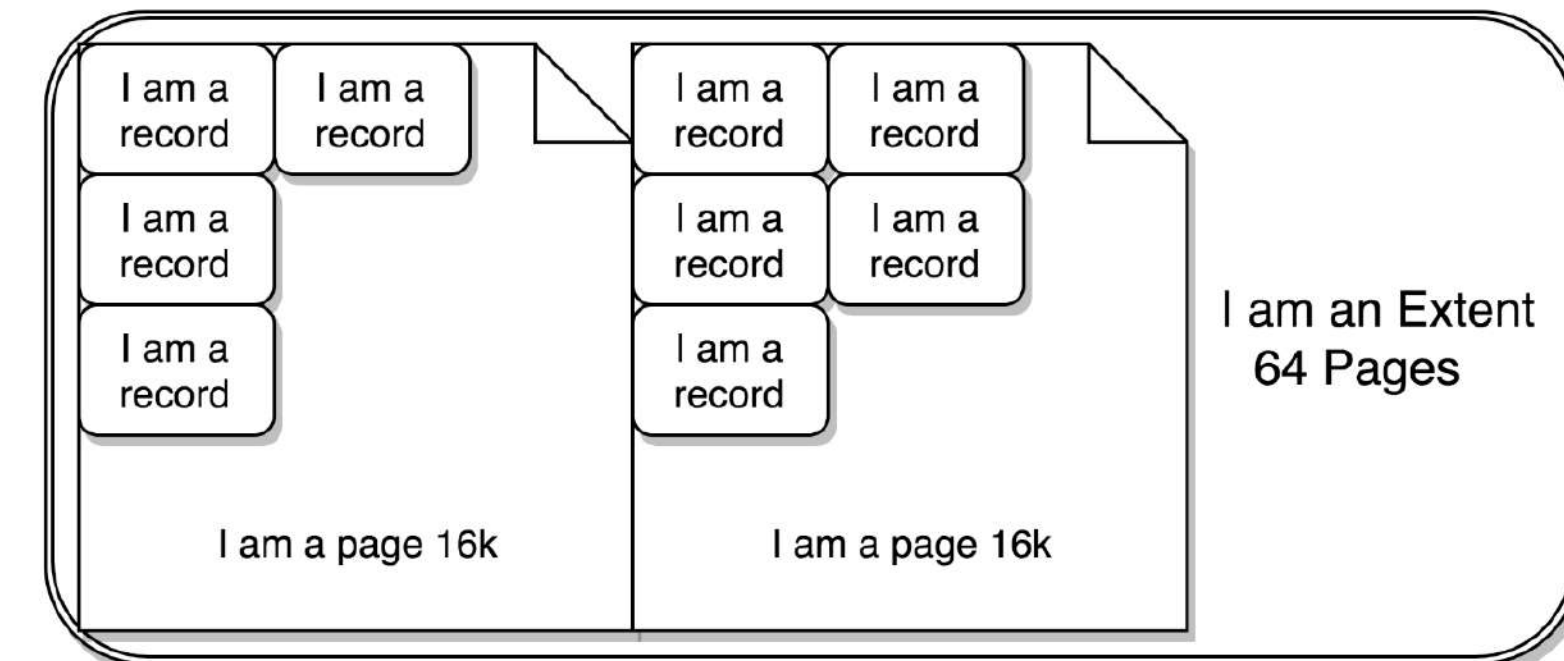
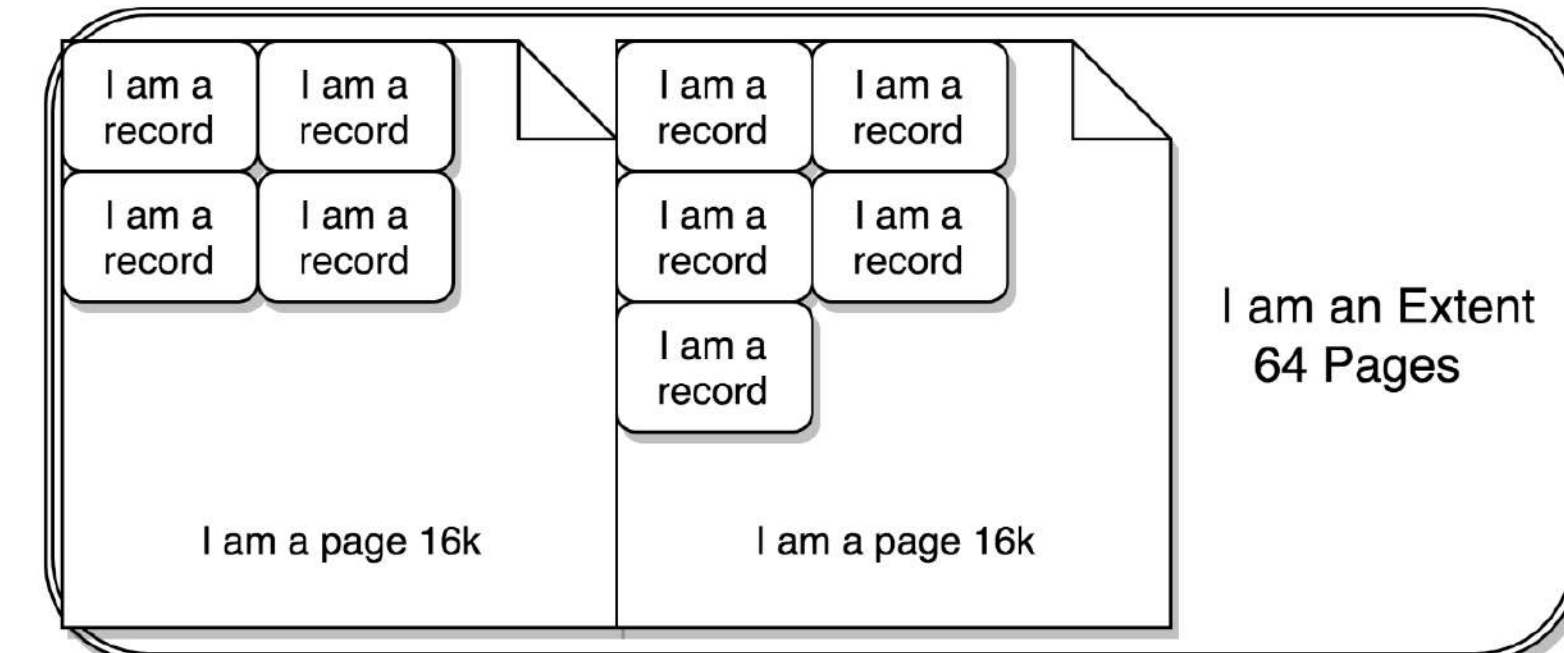
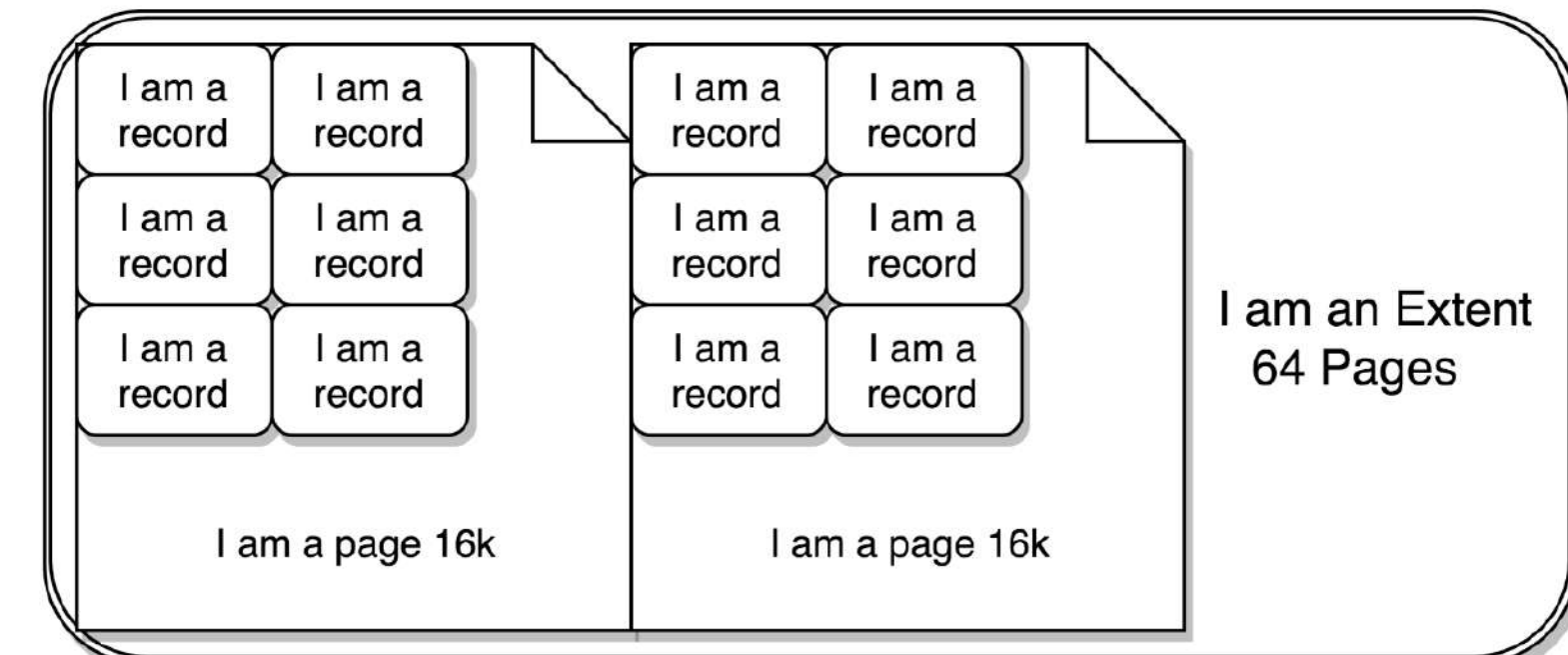
tbpcid

tbUUID

```
data/split/  
|-- db.opt  
|-- tbinc.frm  
|-- tbinc.ibd  
|-- tbpcid.frm  
|-- tbpcid.ibd  
|-- tbUUID.frm  
`-- tbUUID.ibd
```

THE MATRYOSHKA DOLL

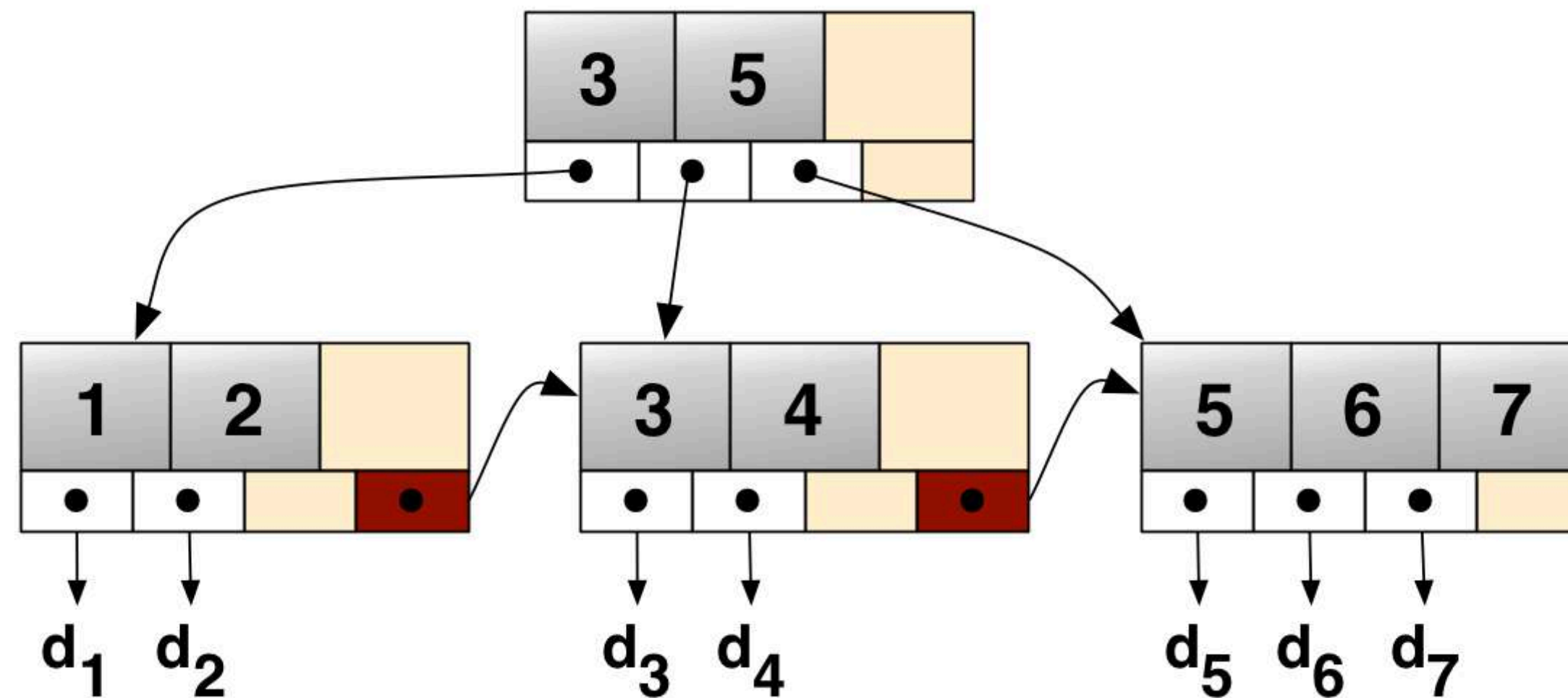
- data is in one segment and each associated index is in its own segment. Segments grow and shrink as data is inserted and deleted. When a segment needs more room, it is extended by one extent (1 megabyte) at a time.
- A group of pages within a tablespace. Extent size is always 1MB.
- A page can contain one or more rows, depending on how much data is in each row. If a row does not fit entirely into a single page, InnoDB sets up additional pointer-style data structures so that the information about the row can be stored in one page.
- Maximum row size for the default `innodb_page_size` of 16KB is about 8000 bytes



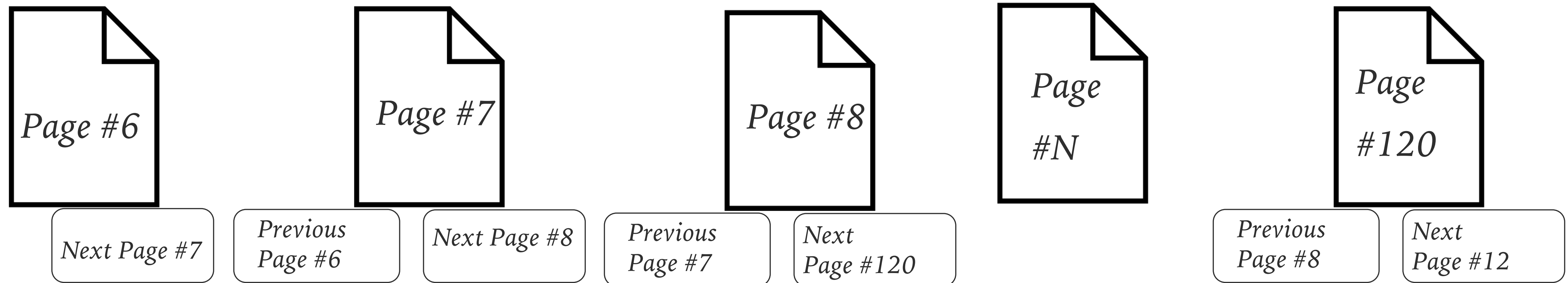
I am a Segment
each extent is 1 MB
I grow or shrink of one extent a time

INNODB B-TREE

- InnoDB uses B-trees to organize your data inside pages across extents, within segments.
- Roots, Branches, and Leaves
- Each page (leaf) contains 2-N rows(s) organized by the primary key. The tree has special pages to manage the different branch(es). These are known as internal nodes (INodes).



LEAVES AS LINKED LIST



- ▶ Pages are also referenced as linked list
- ▶ Linked list follow the logical order of the index (or Primary Key)
- ▶ Linked list order may not follow the physical order of the pages in the extents

A RECORD IN A PAGE - TBINC

```
CREATE TABLE `tbinc` (  
  `id` bigint(11) NOT NULL AUTO_INCREMENT,  
  `uuid` char(36) NOT NULL,  
  `processid` smallint(6) NOT NULL,  
  `points` int(11) NOT NULL,  
  `date` date NOT NULL,  
  `short` varchar(50) NOT NULL,  
  `longc` varchar(500) NOT NULL,  
  `active` tinyint(2) NOT NULL DEFAULT '1',  
  `time` timestamp NOT NULL DEFAULT  
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`),  
  KEY `IDX_processid` (`processid`, `active`),  
  KEY `IDX_active` (`id`, `active`)  
) ENGINE=InnoDB
```

```
ROOT NODE #3: 169 records, 2873 bytes  
  NODE POINTER RECORD ≥ (id=2) → #6  
  LEAF NODE #6: 12 records, 7560 bytes  
    RECORD: (id=2) →  
      (uuid="a993ee0c-e7a4-11ea-bde9-08002734ed50",  
       processid=14,  
       points=3531,  
       date="2020-04-01",  
       short="Art.. have",  
       longc="France.. To-mo",  
       active=1,  
       time="2020-08-26 14:01:39")  
    RECORD: (id=5) → ...  
  NODE POINTER RECORD ≥ (id=38) → #7  
  LEAF NODE #7: 24 records, 15120 bytes  
    RECORD: (id=38) →
```

A RECORD IN A PAGE - TBPCID

```
Create Table: CREATE TABLE `tbpcid` (  
  `id` bigint(11) NOT NULL AUTO_INCREMENT,  
  `uuid` char(36) NOT NULL,  
  `processid` smallint(6) NOT NULL,  
  `points` int(11) NOT NULL,  
  `date` date NOT NULL,  
  `short` varchar(50) NOT NULL,  
  `longc` varchar(500) NOT NULL,  
  `active` tinyint(2) NOT NULL DEFAULT '1',  
  `time` timestamp NOT NULL DEFAULT  
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  PRIMARY KEY (`processid`, `id`),  
  KEY `IDX_id` (`id`),  
  KEY `IDX_mid` (`processid`, `active`)  
) ENGINE=InnoDB
```

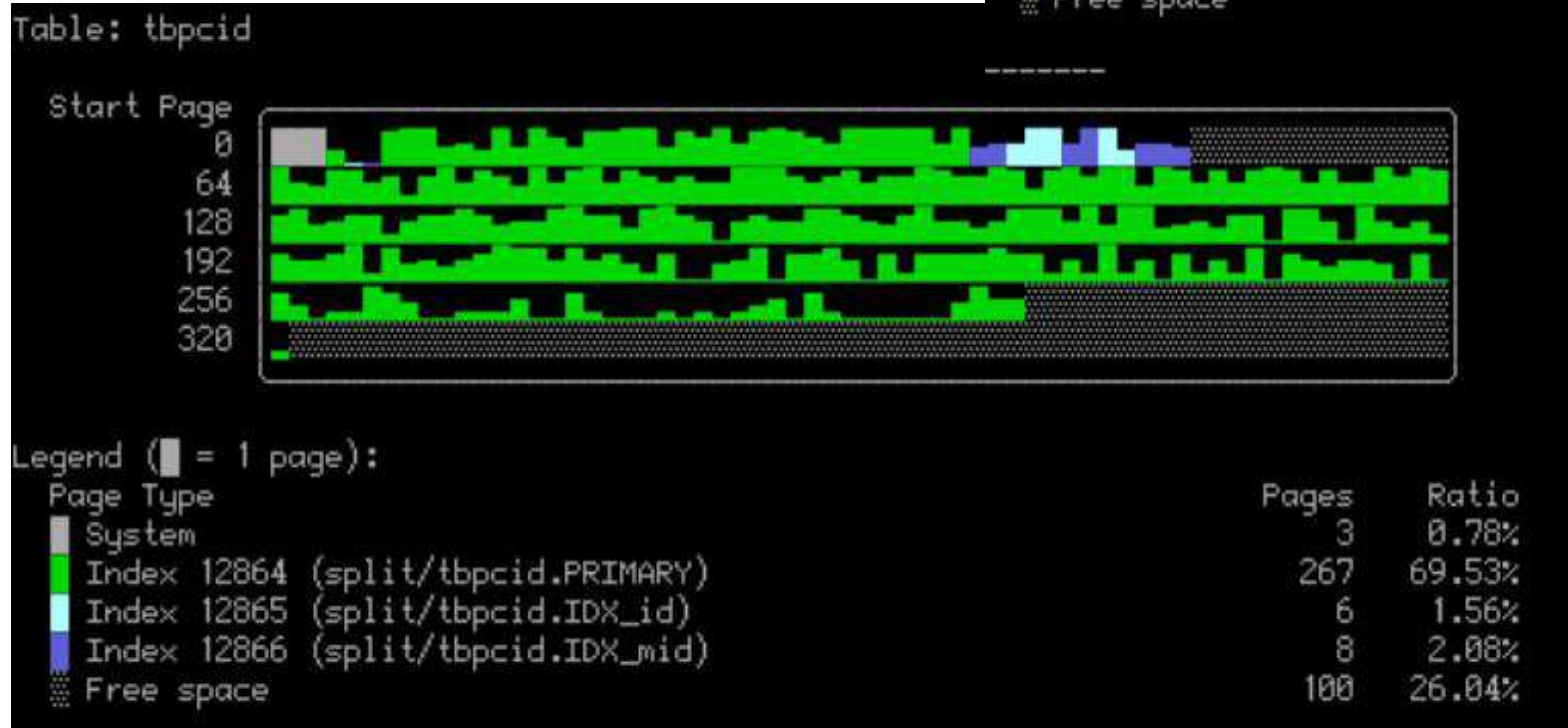
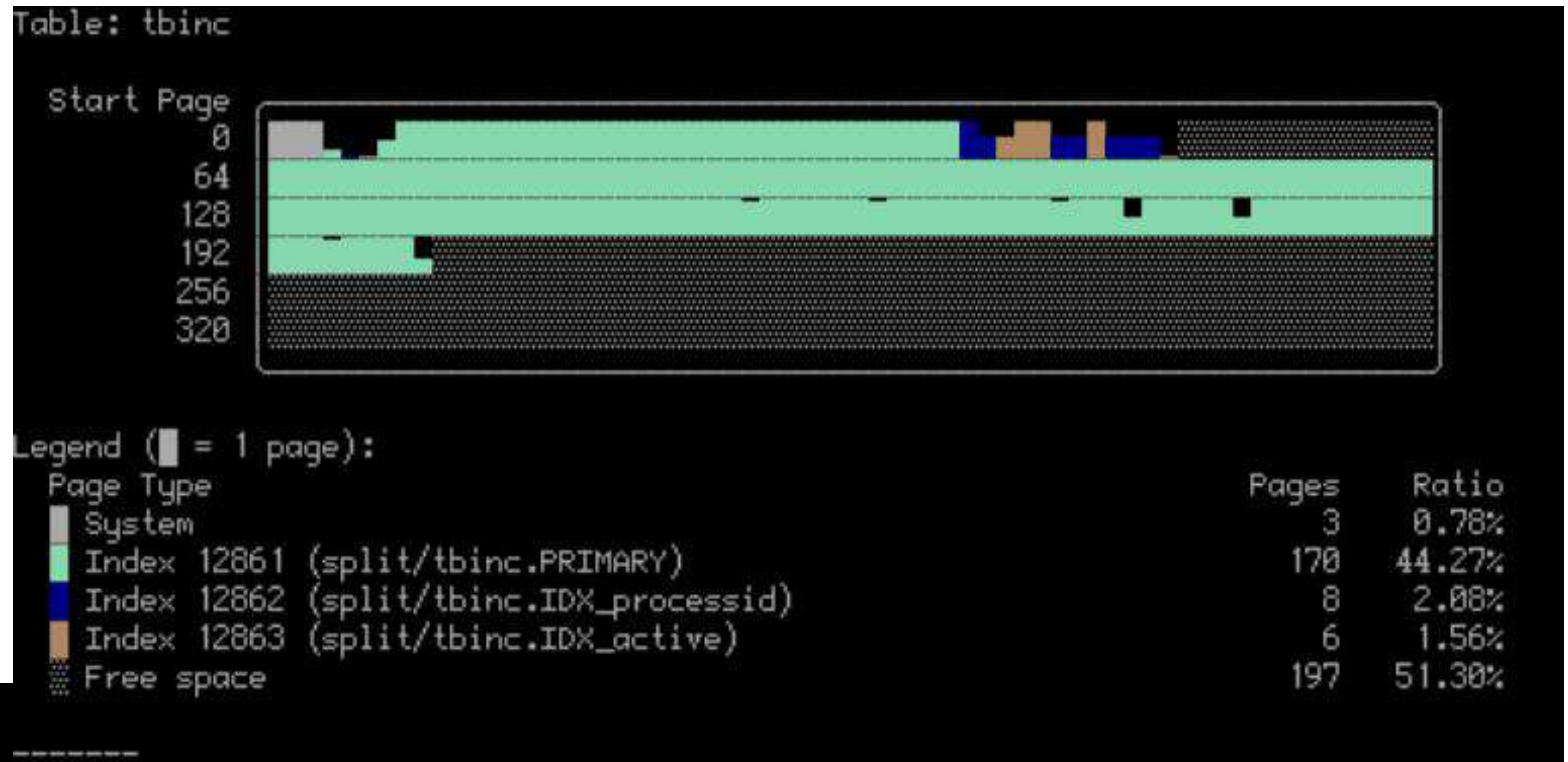
```
ROOT NODE #3: 270 records, 5130 bytes  
NODE POINTER RECORD ≥ (processid=7, id=17) → #6  
LEAF NODE #6: 24 records, 14901 bytes  
  RECORD: (processid=0, id=137) →  
    (uuid="ad1353d3-e7a4-11ea-bde9-08002734ed50",  
     points=3729,  
     date="2020-04-01",  
     short="Nor ... the",  
     longc="From ...",  
     active=1,  
     time="2020-08-26 14:01:45")  
NODE POINTER RECORD ≥ (processid=1, id=3827) →  
#209  
LEAF NODE #209: 15 records, 9450 bytes  
  RECORD: (processid=1, id=3827) →
```

A RECORD IN A PAGE - TBUUID

```
Create Table: CREATE TABLE `tbUUID` (  
  `id` bigint(11) NOT NULL AUTO_INCREMENT,  
  `uuid` char(36) NOT NULL,  
  `processid` smallint(6) NOT NULL,  
  `points` int(11) NOT NULL,  
  `date` date NOT NULL,  
  `short` varchar(50) NOT NULL,  
  `longc` varchar(500) NOT NULL,  
  `active` tinyint(2) NOT NULL DEFAULT '1',  
  `time` timestamp NOT NULL DEFAULT  
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  PRIMARY KEY (`uuid`, `processid`),  
  UNIQUE KEY `IDX_id` (`id`),  
  KEY `IDX_mid` (`processid`, `active`)  
) ENGINE=InnoDB
```

```
ROOT NODE #3: 235 records, 11280 bytes  
  NODE POINTER RECORD ≥ (uuid="05de43720080-9edb-  
ae11-268e-0d216866", processid=12) → #6  
  LEAF NODE #6: 24 records, 15120 bytes  
    RECORD:  
      (uuid="05de43720080-9edb-ae11-268e-00252077",  
processid=73) →  
      (id=632,  
points=3740,  
date="2020-01-10",  
short="So ... p",  
longc="The s.. w",  
active=0,  
time="2020-08-27 12:40:18")  
  NODE POINTER RECORD ≥ (uuid="05de43720080-9edb-  
ae11-268e-03b393c6", processid=192) → #136  
  LEAF NODE #136: 4 records, 2520 bytes  
    RECORD: (uuid="05de43720080-9edb-ae11-268e-  
03b393c6", processid=192) →
```

SEGMENTS, EXTENTS AND PAGES



Same number of rows.

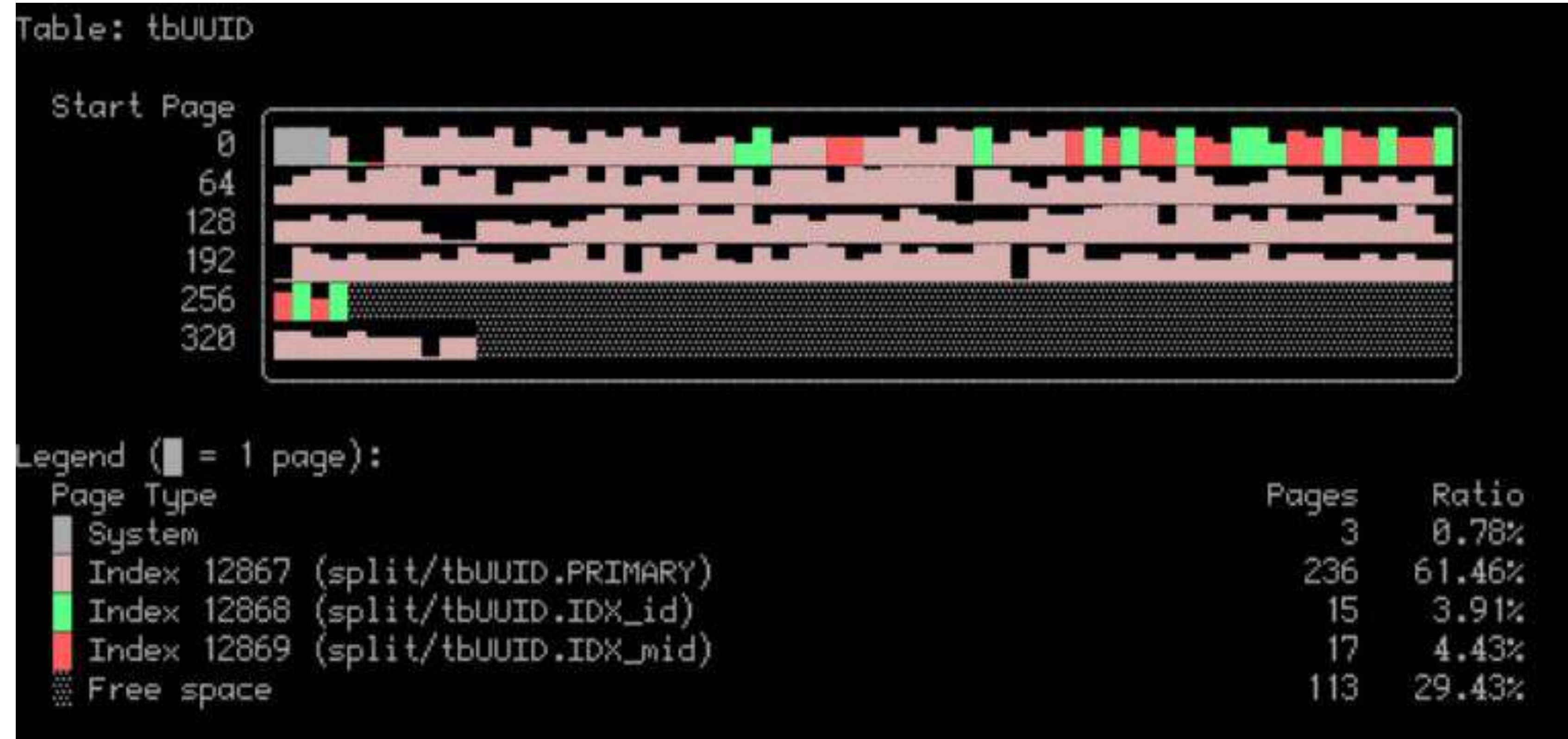
Different physical distribution:

PK tbinc: id autoinc

PK tbpcid: process id, id

PK tbUUID: reverse UUID,
process id

SEGMENTS, EXTENTS AND PAGES



Same number of rows.

Different physical distribution:

PK tbinc: id autoinc

PK tbpcid: process id, id

PK tbUUID: reverse UUID,
process id

REMINDER

The concept here is that while you organize your data in tables and rows, InnoDB organizes it in branches, pages, and records.

It is very important to keep in mind that InnoDB does not work on a single row basis.

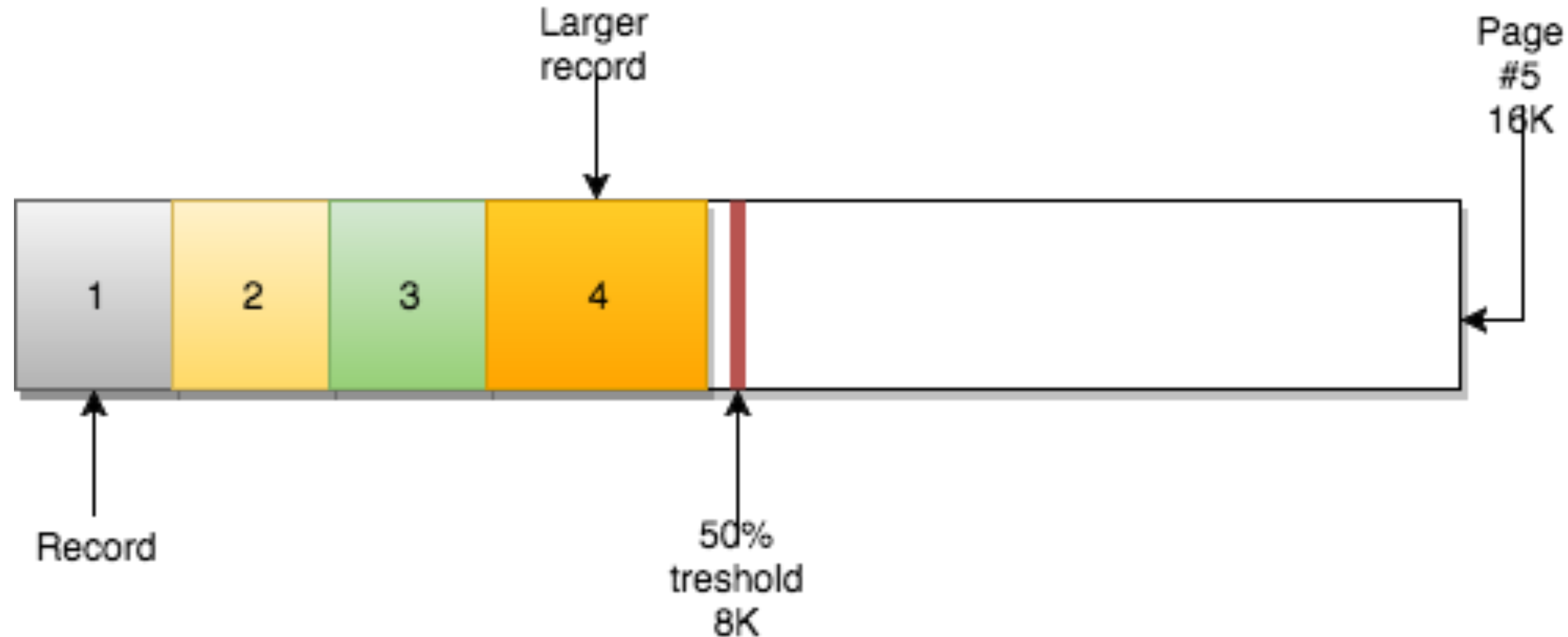
InnoDB always operates on pages.

Once a page is loaded, it will then scan the page for the requested row/record.

PAGE INTERNALS

A page can be empty or fully filled (100%).

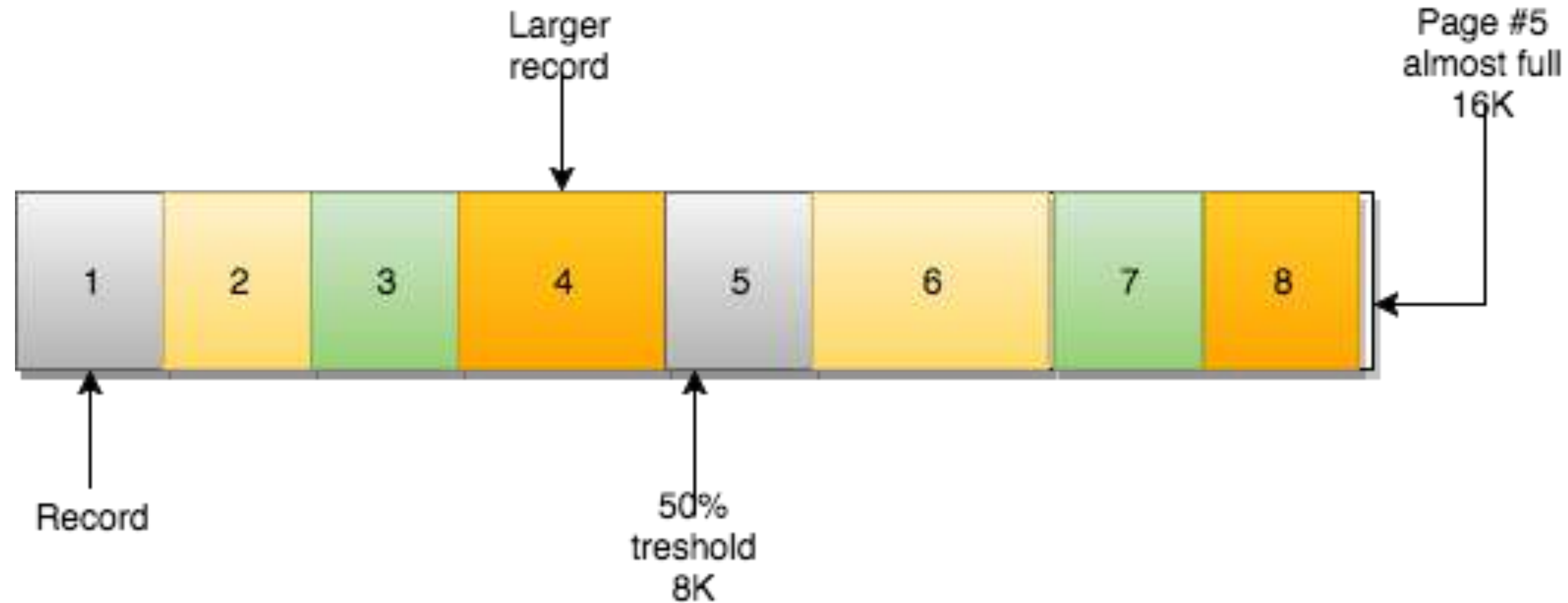
The row-records will be organized by PK. For example, if your table is using an *AUTO_INCREMENT*, you will have the sequence ID = 1, 2, 3, 4, etc.



A page also has another important attribute: `MERGE_THRESHOLD`. The default value of this parameter is 50% of the page, and it plays a very important role in InnoDB merge activity.

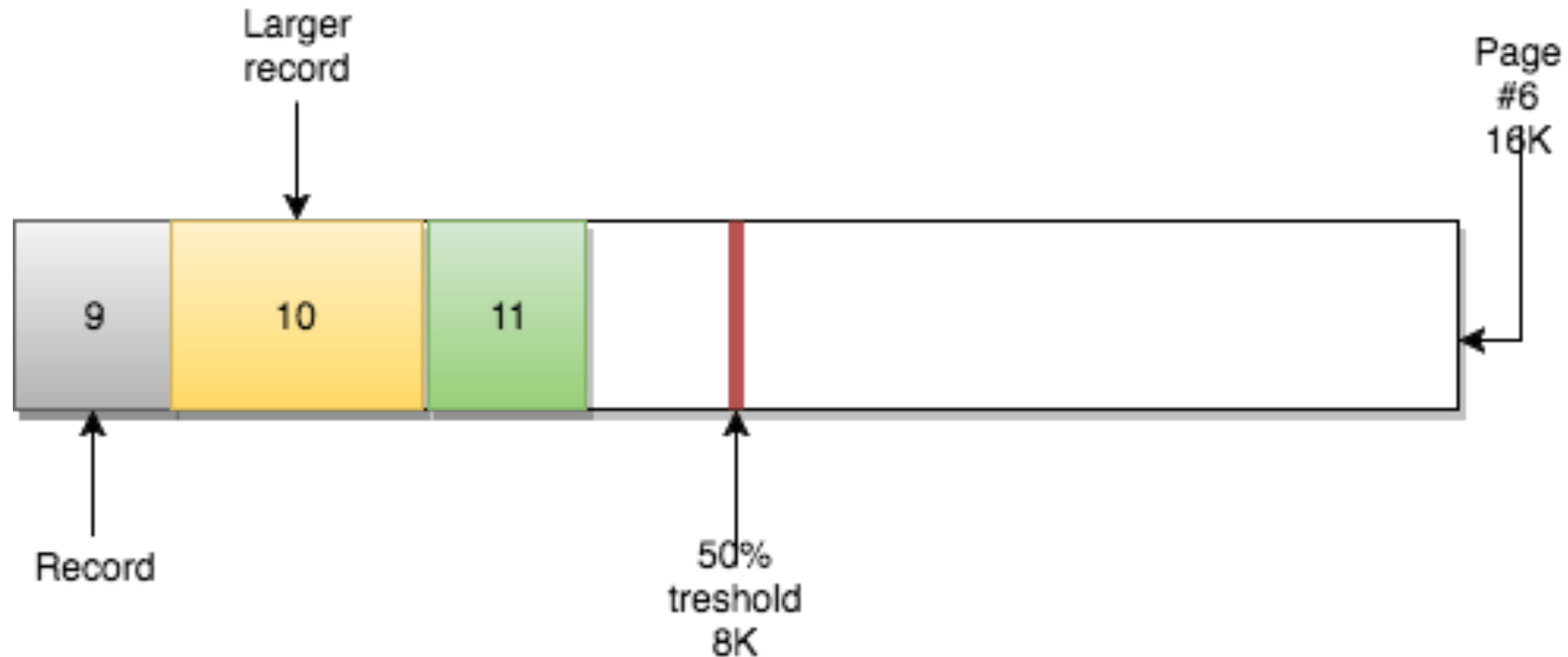
PAGE INTERNALS

While you insert data, the page is filled up sequentially if the incoming record can be accommodated inside the page.



PAGE INTERNALS

When a page is full, the next record will be inserted into the NEXT page (*SPLIT counter incremented*):

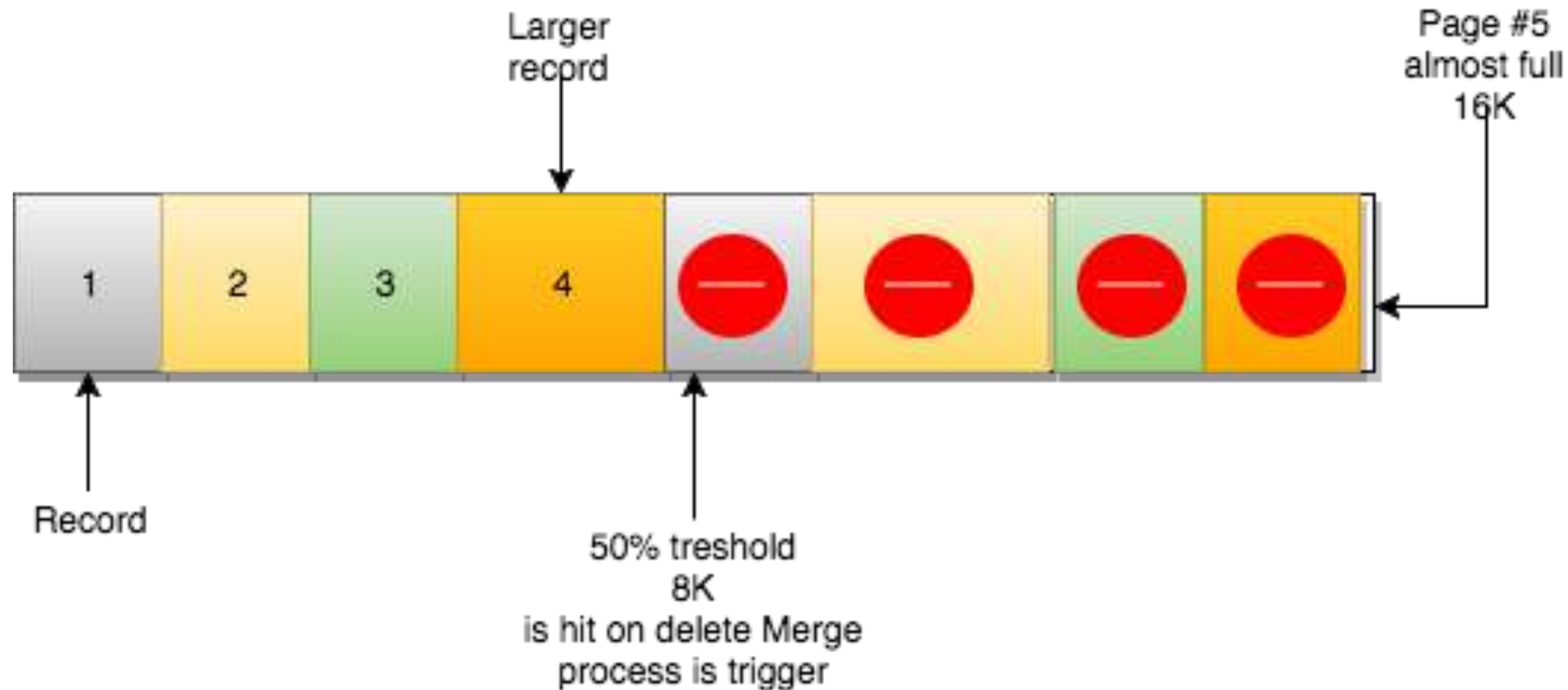


Browse is not only top down but also by linked list, page #6 refer to #5 and #7

But what happens if I start to delete values?

PAGE MERGES

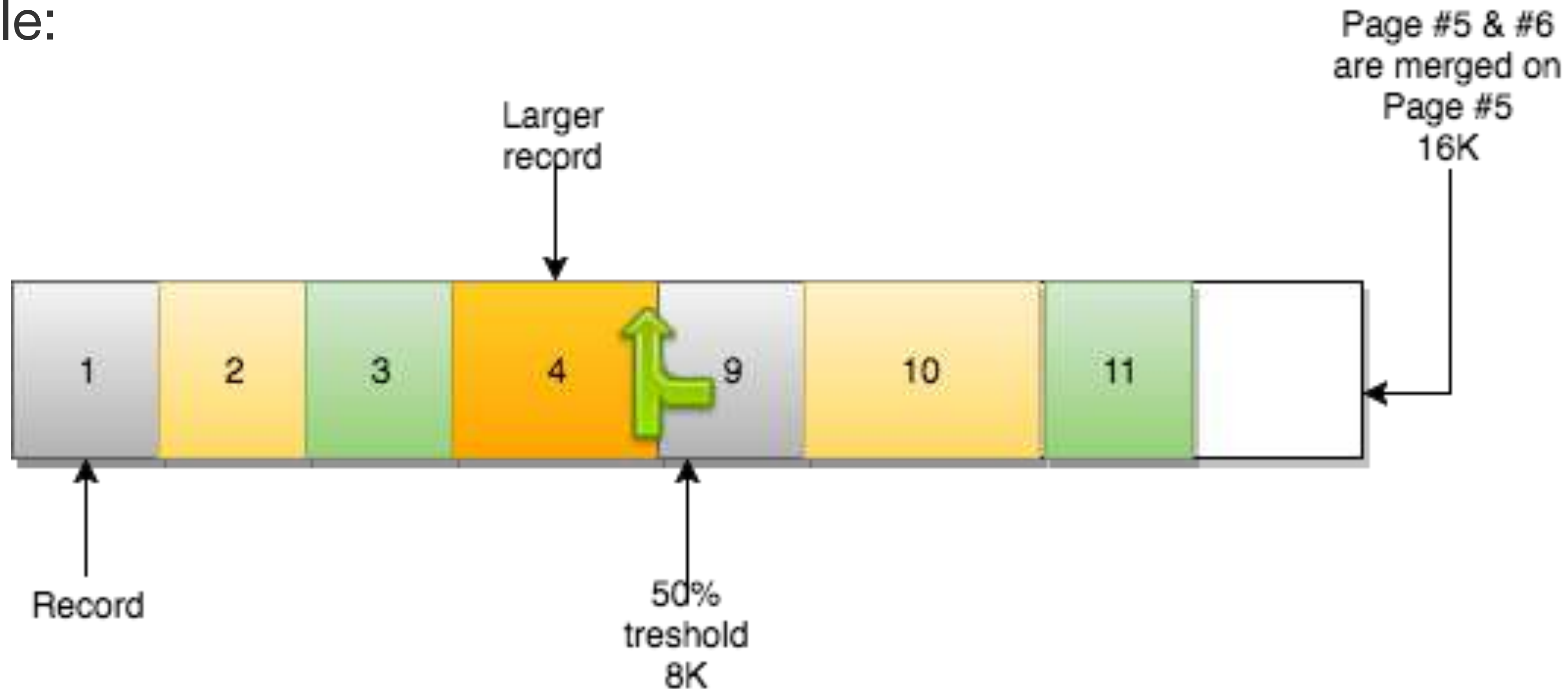
When you delete a record, the record is not physically deleted. Instead, it flags the record as deleted and the space it used becomes reclaimable.



When a page has received enough deletes to match the `MERGE_THRESHOLD` (50% of the page size by default), InnoDB starts to look to the closest pages (`NEXT` and `PREVIOUS`) to see if there is any chance to optimize the space utilization by merging the two pages.

PAGE INTERNALS

In this example, Page #6 is utilizing less than half of its space. Page #5 received many deletes and is also now less than 50% used. From InnoDB's perspective, they are mergeable:



The merge operation results in Page #5 containing its previous data plus the data from Page #6

Page #6 will be empty after the merge

THE TAKEOUT

The rule is: Merges happen on delete and update operations involving close linked pages.

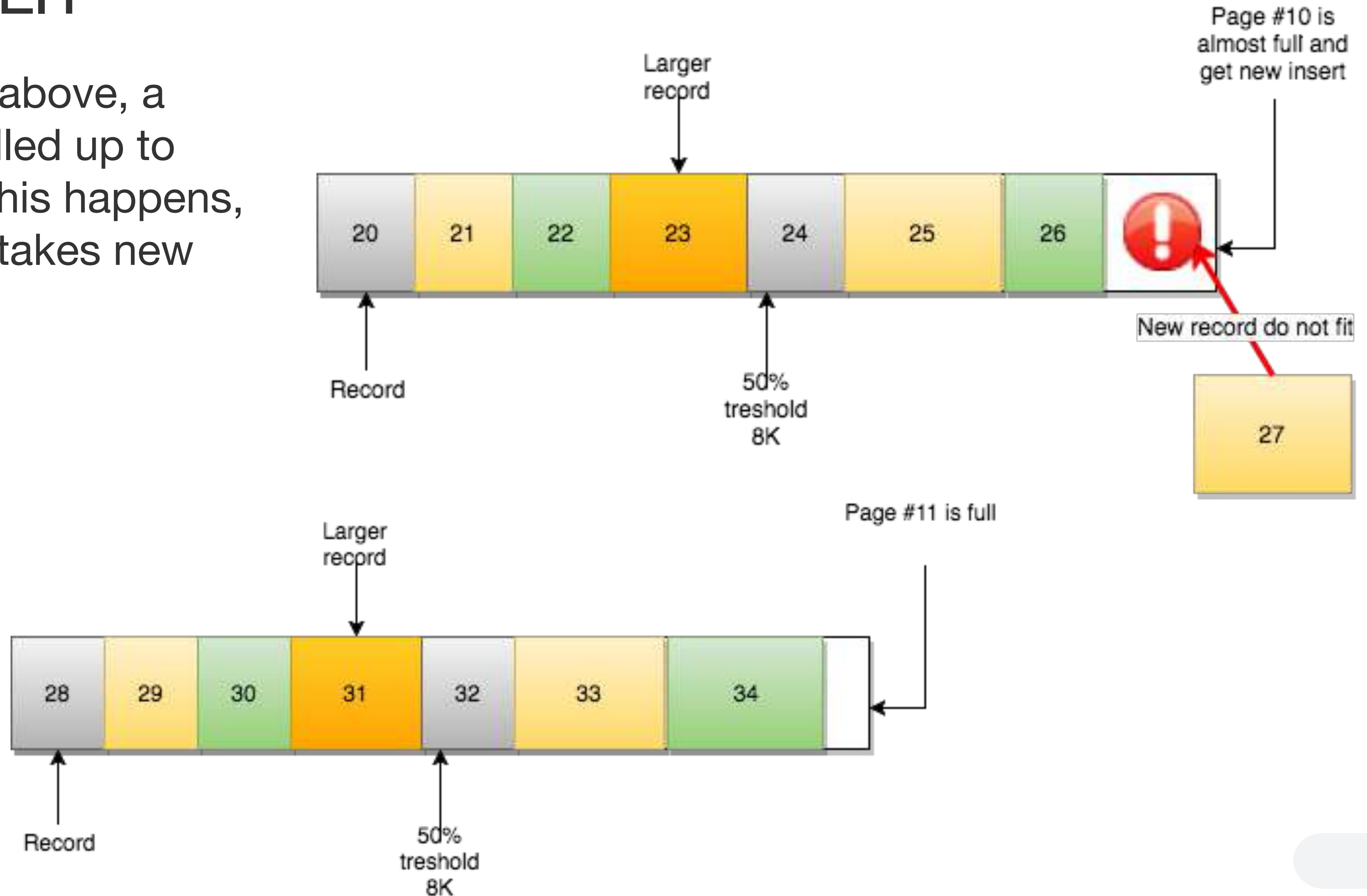
If a merge operation is successful, the `index_page_merge_successful` metric in `INFORMATION_SCHEMA.INNODB_METRICS` is incremented.

```
select name,count from INFORMATION_SCHEMA.INNODB_METRICS where name like 'index_page%';
```

name	count
index_page_splits	25391
index_page_merge_attempts	135259
index_page_merge_successful	15534
index_page_reorg_attempts	83949
index_page_reorg_successful	83949
index_page_discards	29

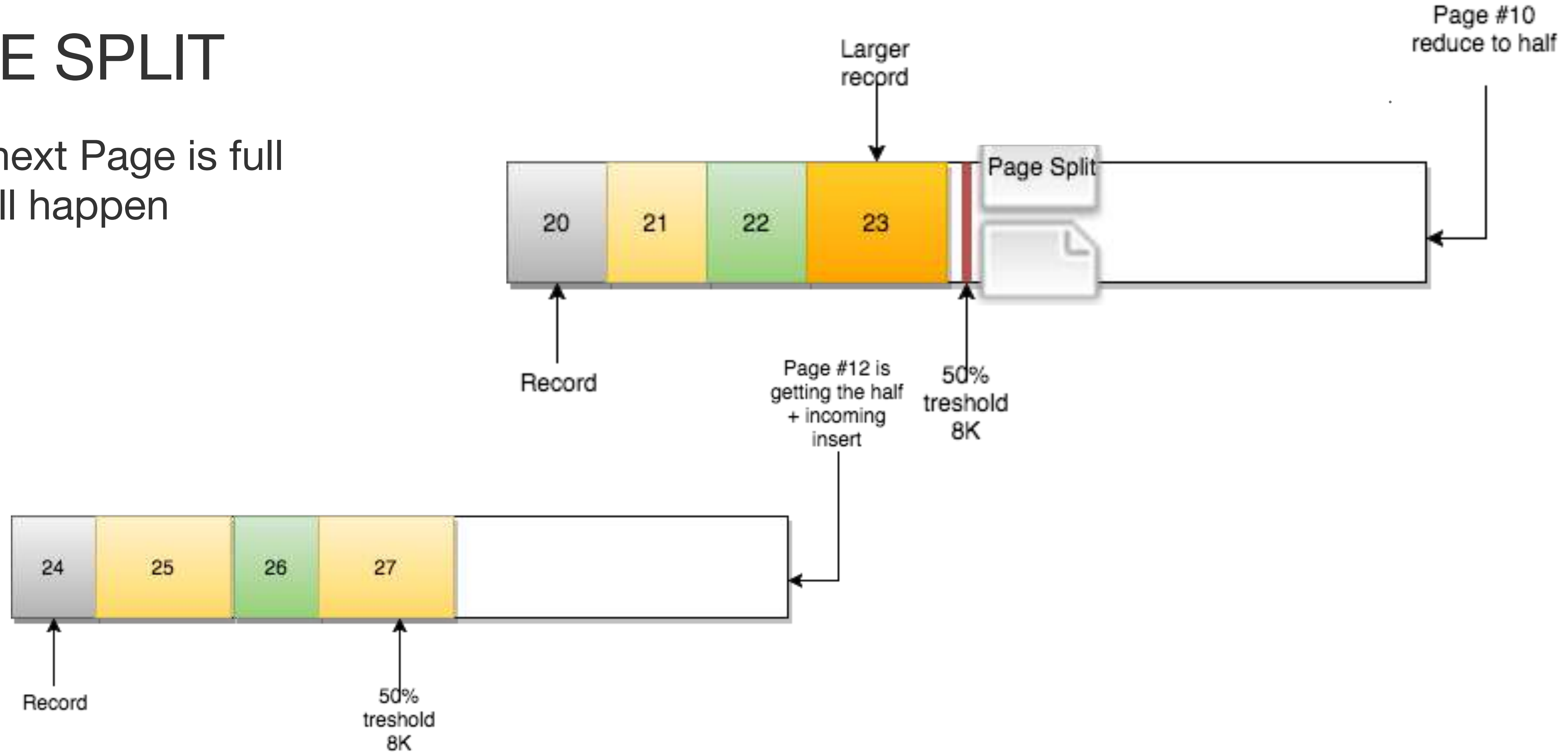
PAGE SPLIT

As mentioned above, a page can be filled up to 100%. When this happens, the next page takes new records.



PAGE SPLIT

When next Page is full split will happen



Records from page #10 will be moved to the split page #12 up to the merge threshold.

PAGE SPLIT

What InnoDB will do is (simplifying):

- Create a new page
- Identify where the original page (Page #10) can be split (at the record level)
- Move records
- Redefine the page relationships

Page #11 stays as it is.

The thing that changes is the relationship between the pages:

- Page #10 will have Prev=9 and Next=12
- Page #12 Prev=10 and Next=11
- Page #11 Prev=12 and Next=13

THE TAKEOUT

The rule is: Page splits happens on Insert or Update, and cause page dislocation (in many cases on different extents).

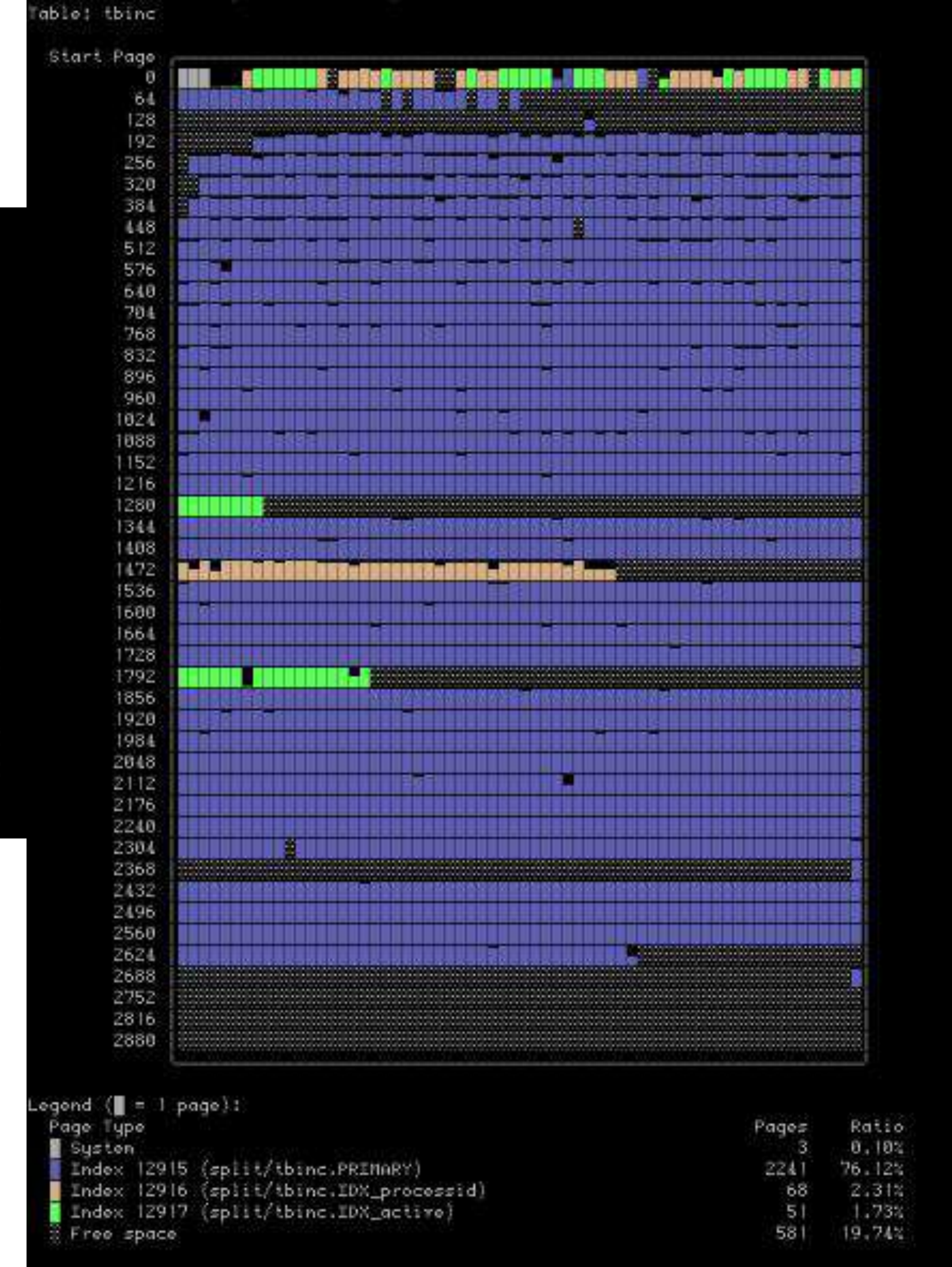
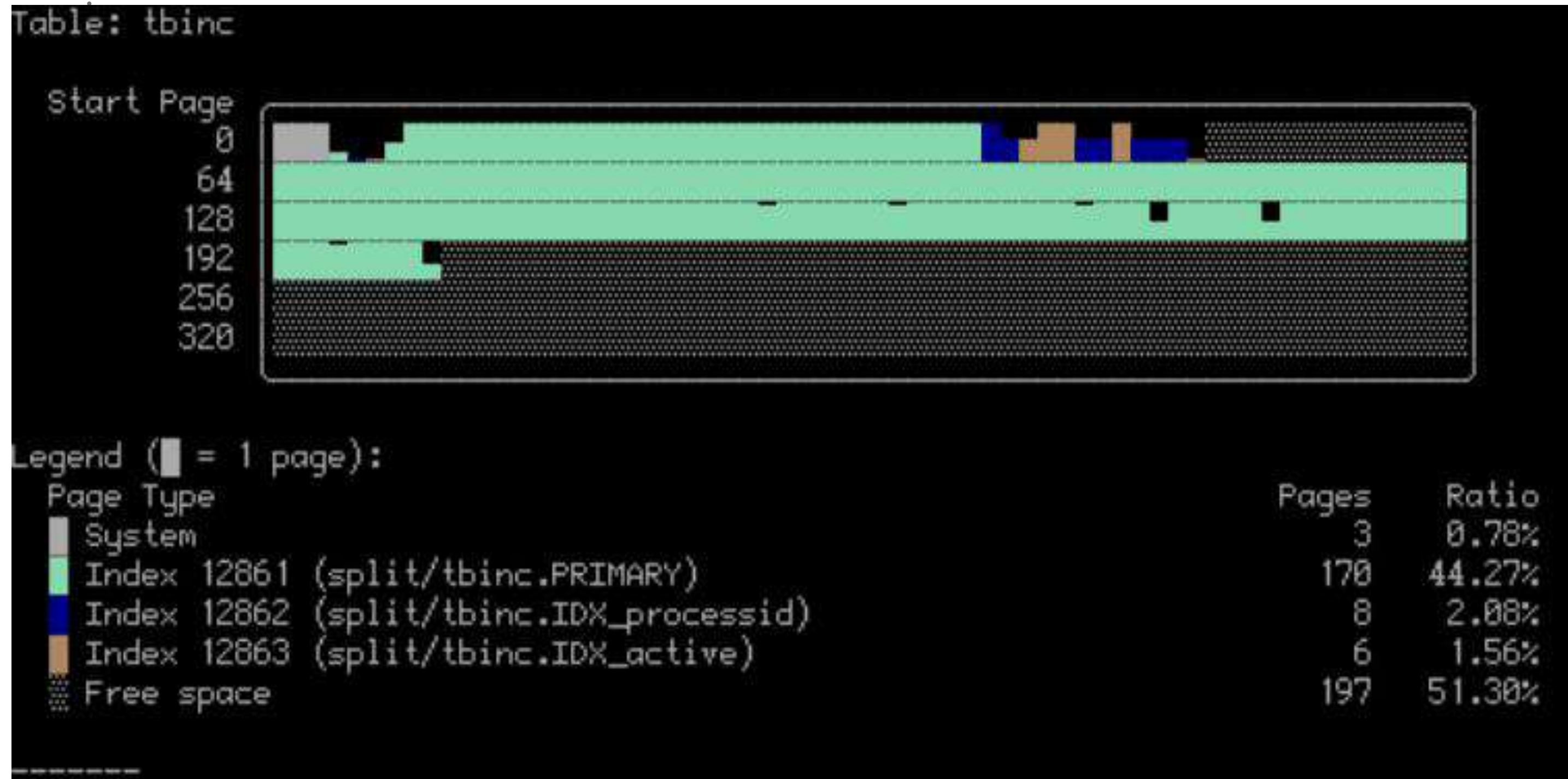
If a merge operation is successful, the `index_page_splits`, `index_page_reorg_attempts` / `successful` metrics in

`INFORMATION_SCHEMA.INNODB_METRICS` is incremented.

```
select name,count from INFORMATION_SCHEMA.INNODB_METRICS where name like 'index_page%';
```

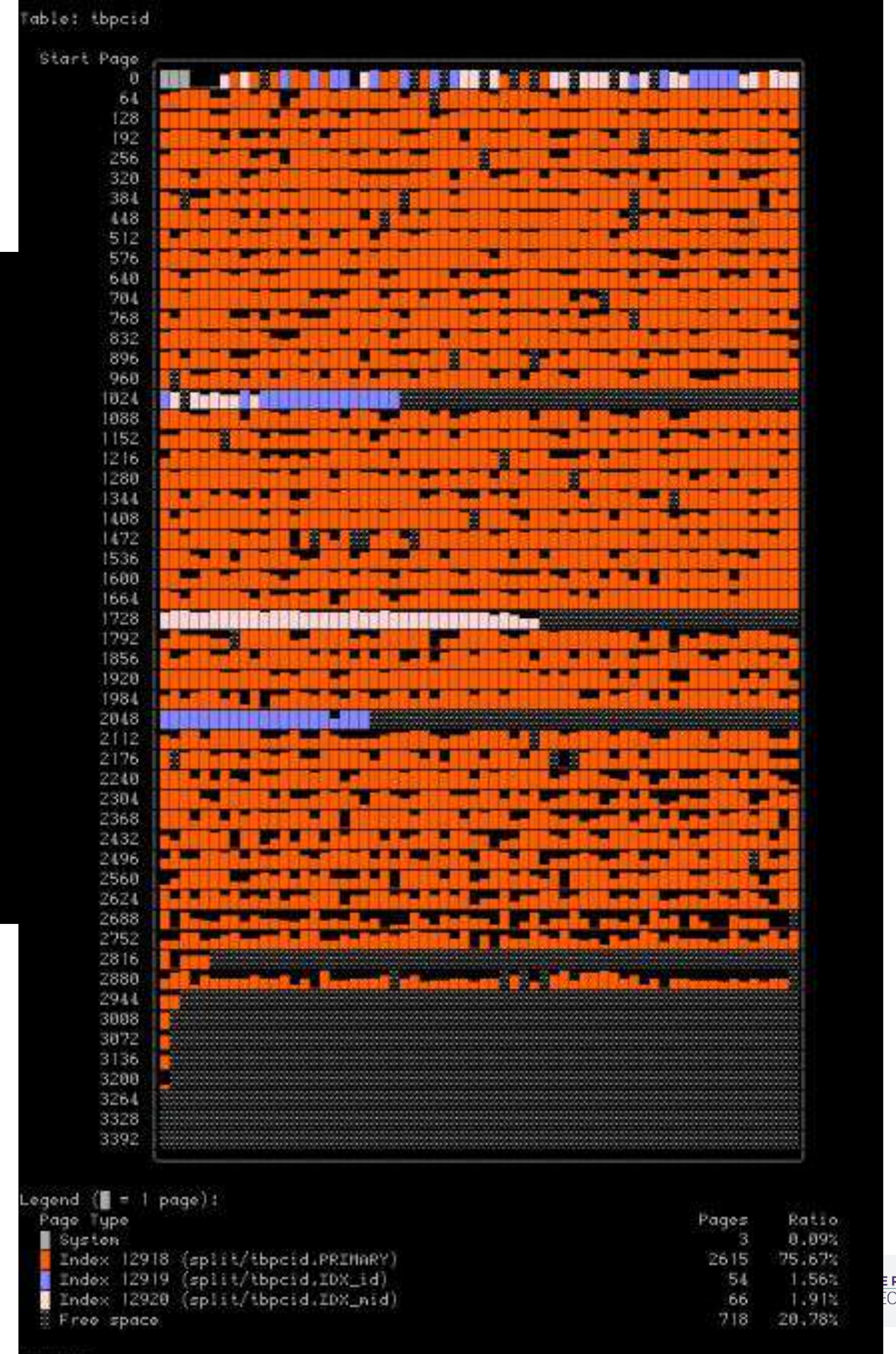
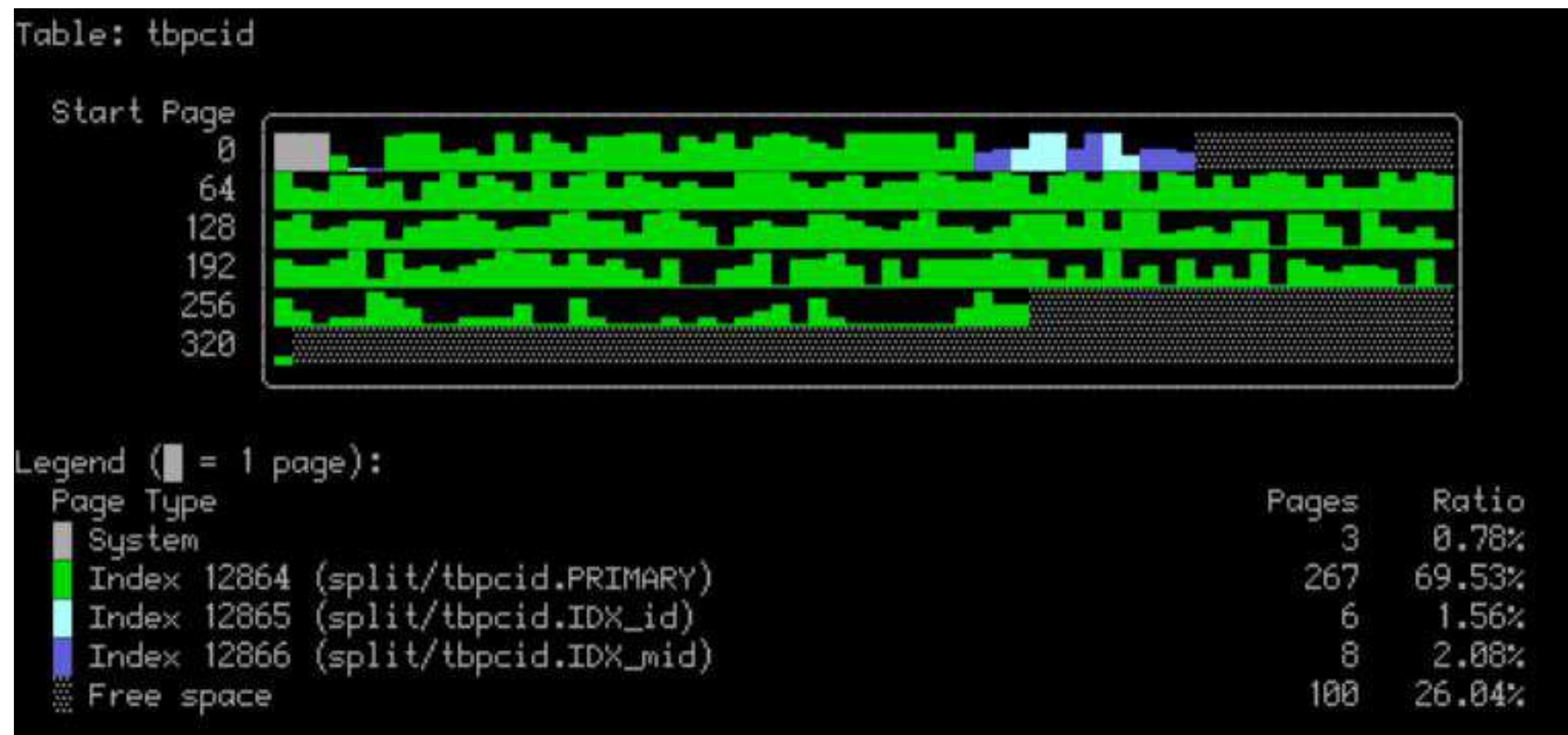
name	count
<code>index_page_splits</code>	25391
<code>index_page_merge_attempts</code>	135259
<code>index_page_merge_successful</code>	15534
<code>index_page_reorg_attempts</code>	83949
<code>index_page_reorg_successful</code>	83949
<code>index_page_discards</code>	29

LET US VISUALIZE - AUTOINCREMENT



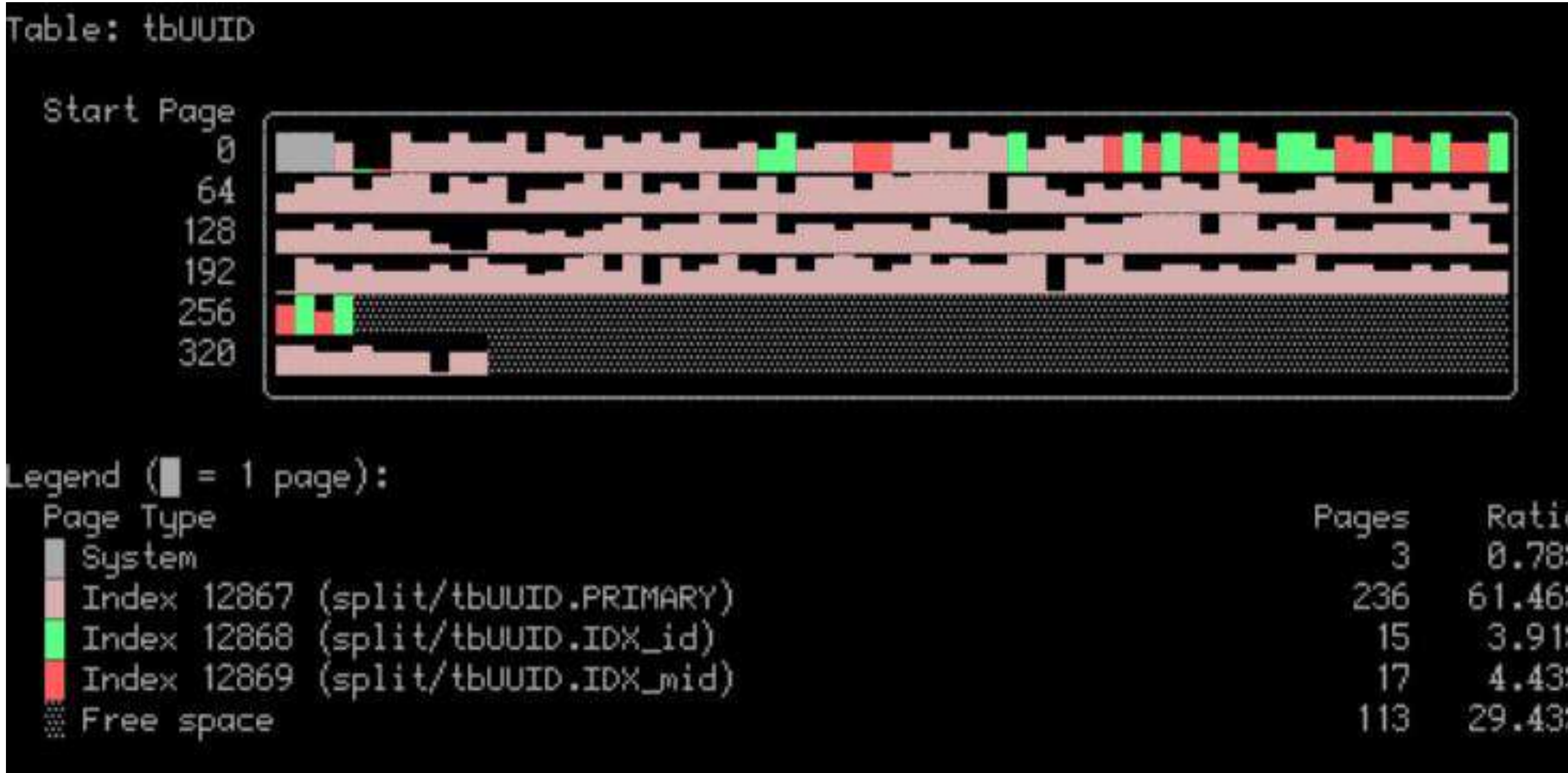
- Insert
- Update
- Delete

LET US VISUALIZE - PROCESSED - INC



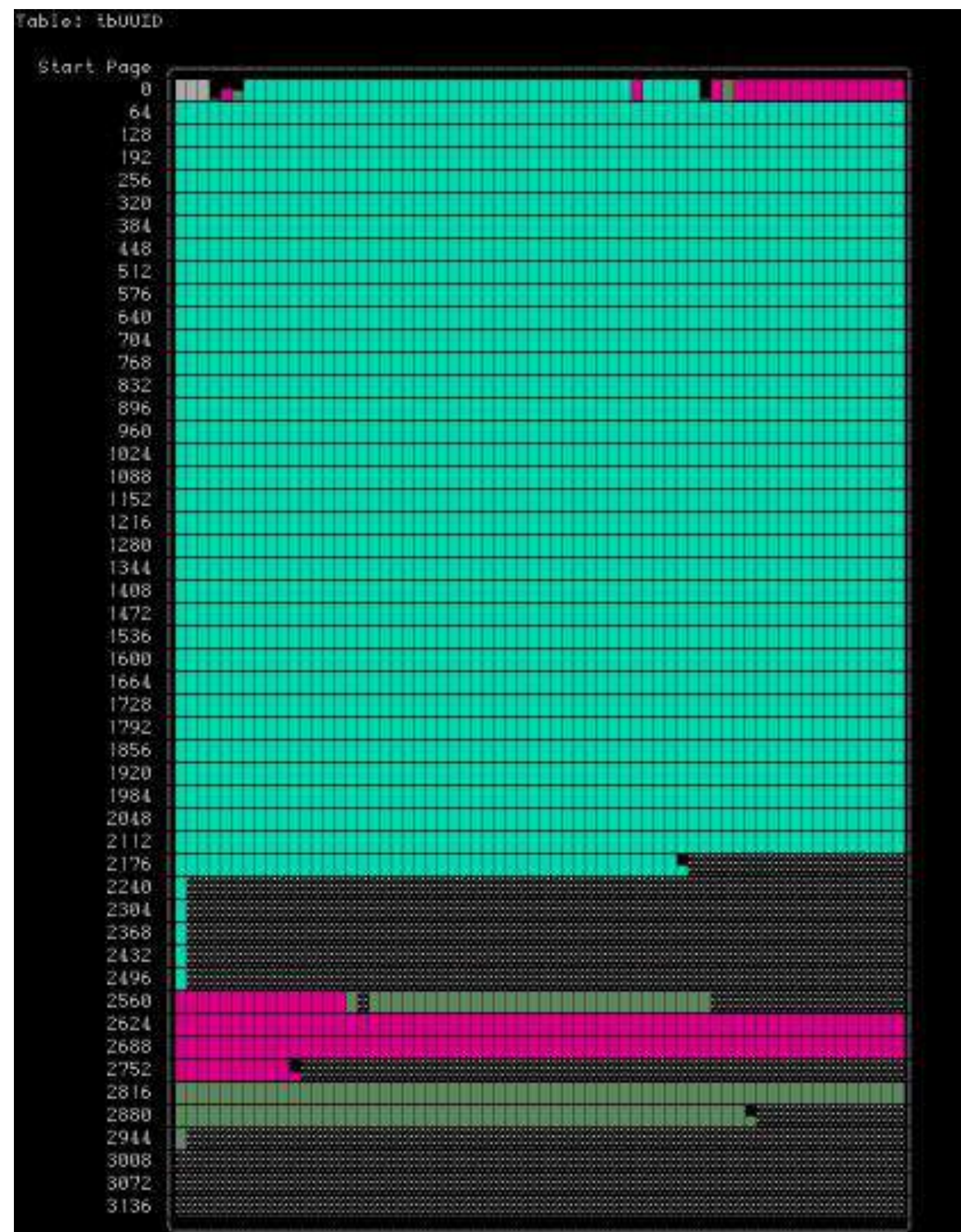
- Insert
- Update
- Delete

LET US VISUALIZE - REVERSE UUID



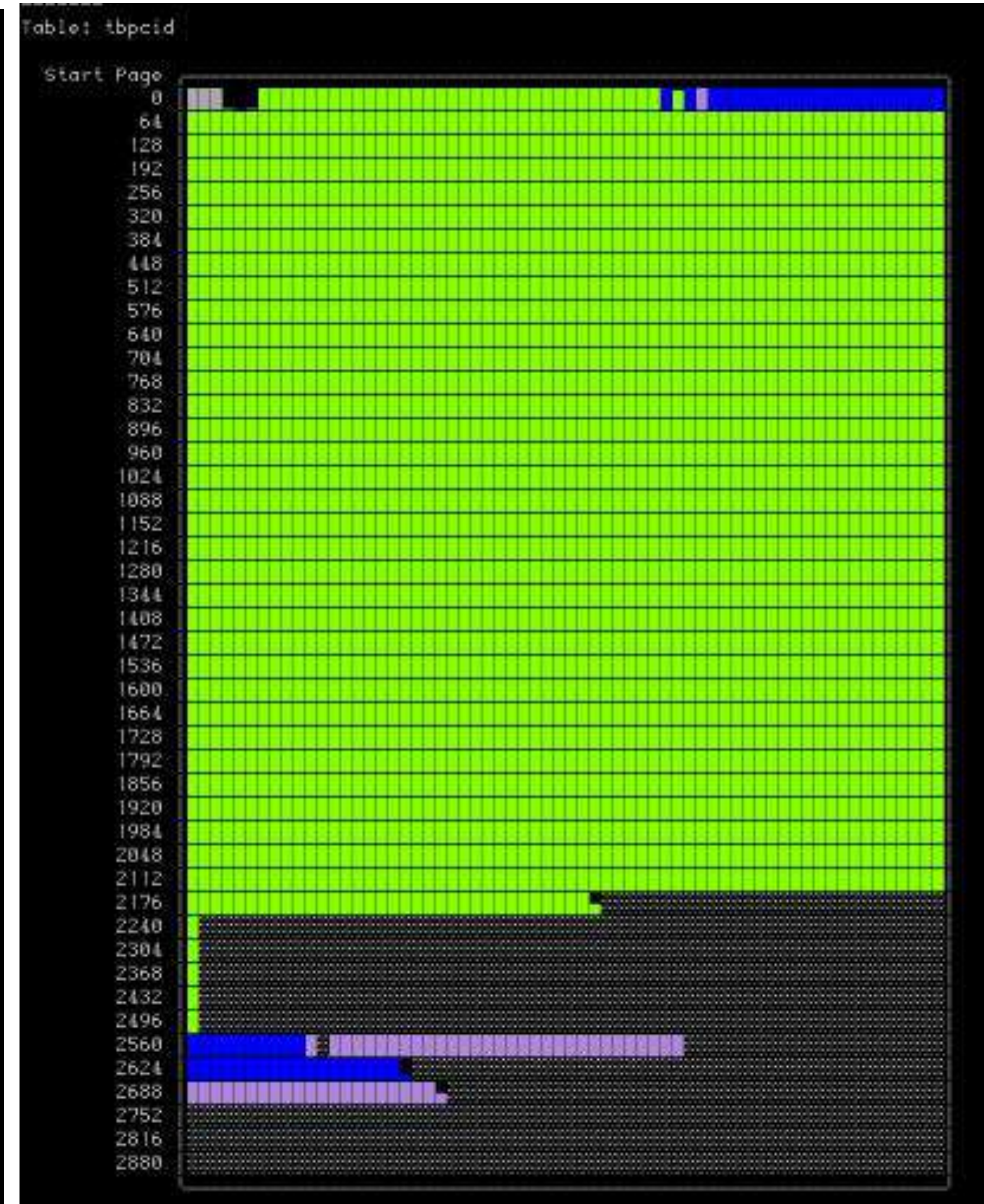
- Insert
- Update
- Delete

LET US VISUALIZE – AFTER OPTIMIZE



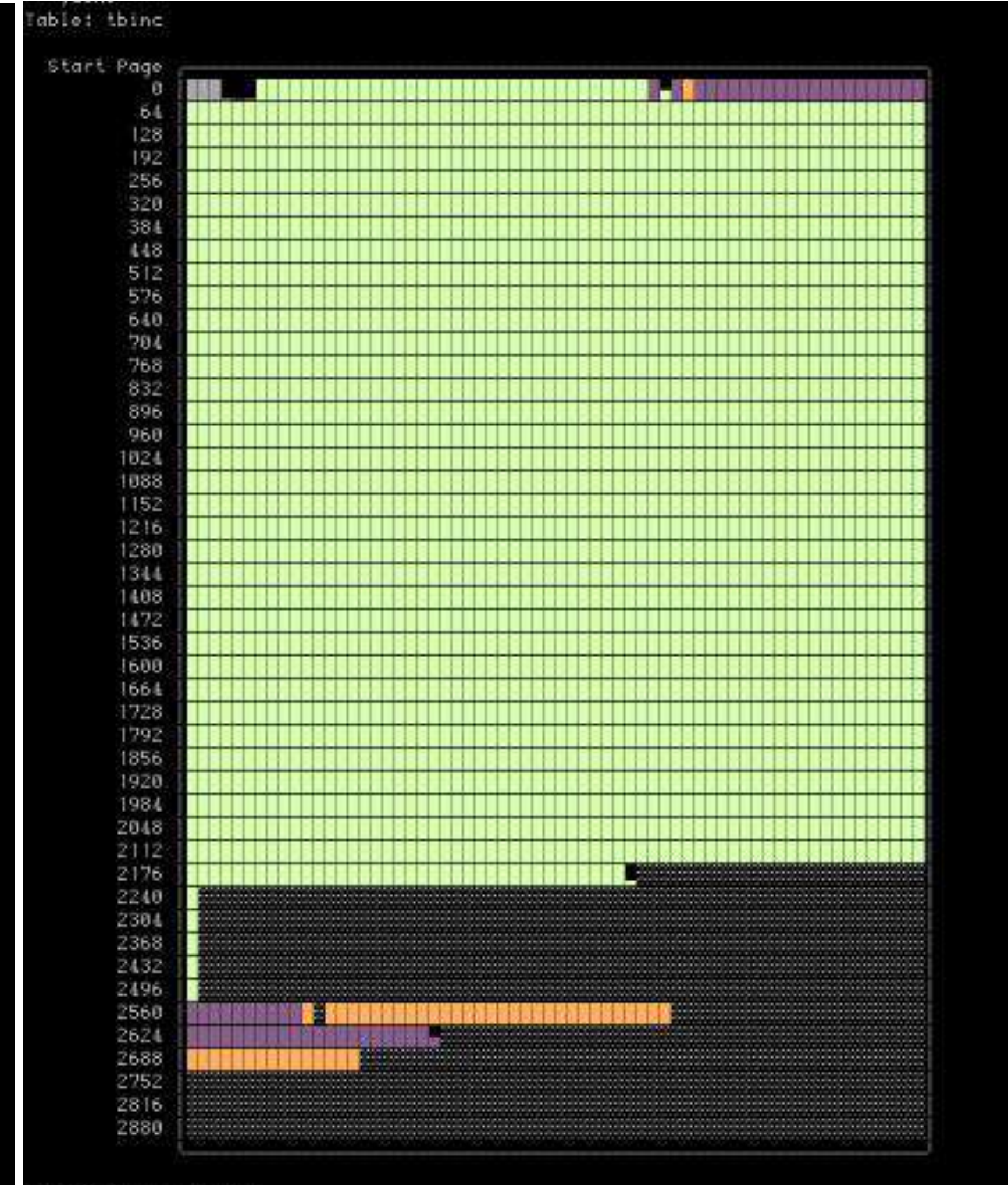
Legend (█ = 1 page):

Page Type	Pages	Ratio
System	3	0.09%
Index 12933 (split/tbuuid.PRIMARY)	2203	68.84%
Index 12934 (split/tbuuid.IDX_id)	172	5.38%
Index 12935 (split/tbuuid.IDX_nid)	149	4.66%
Free space	673	21.03%



Legend (█ = 1 page):

Page Type	Pages	Ratio
System	3	0.10%
Index 12930 (split/tbpcid.PRIMARY)	2188	74.32%
Index 12931 (split/tbpcid.IDX_id)	52	1.77%
Index 12932 (split/tbpcid.IDX_nid)	55	1.87%
Free space	646	21.94%

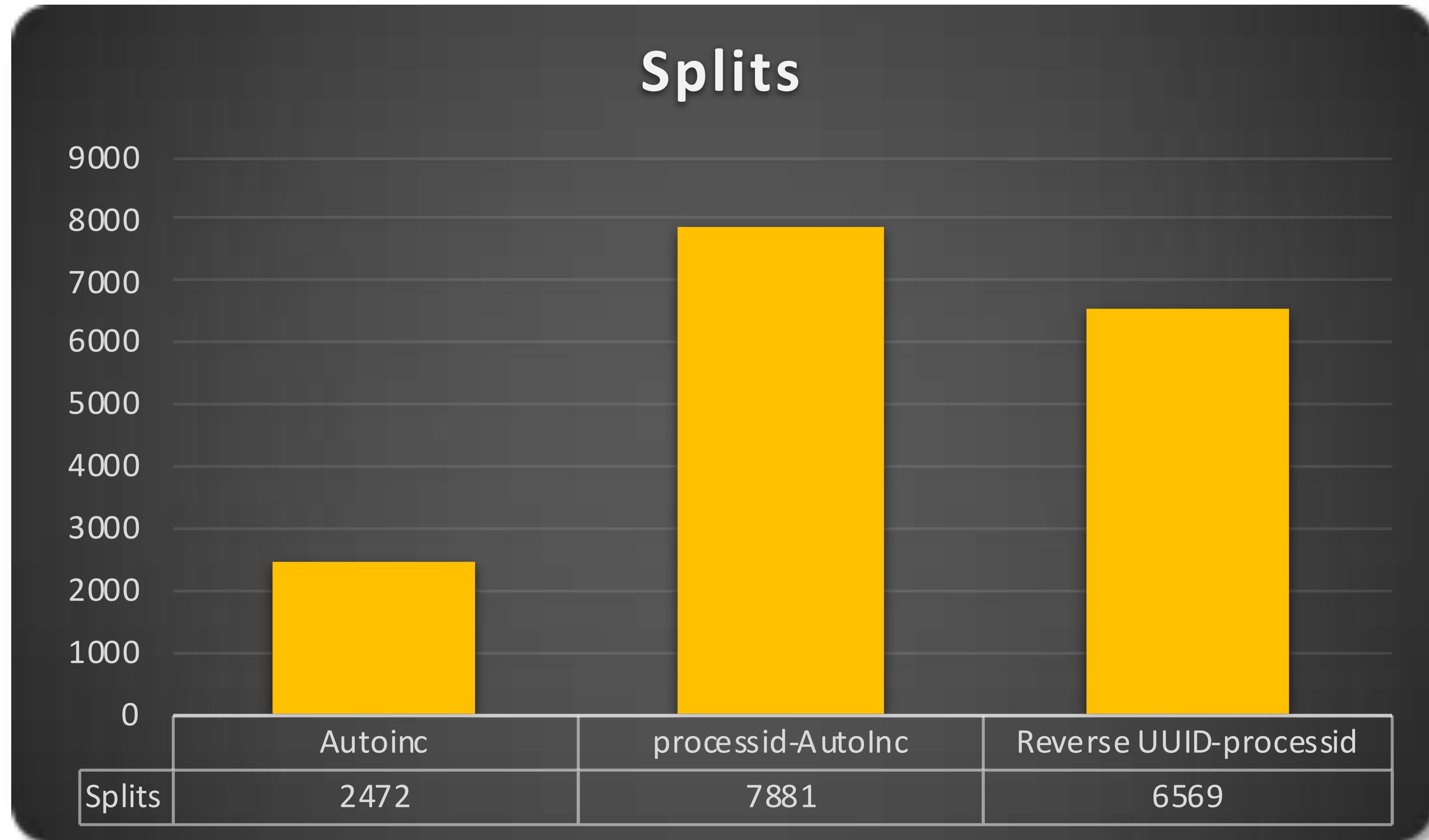


Legend (█ = 1 page):

Page Type	Pages	Ratio
System	3	0.10%
Index 12927 (split/tbinc.PRIMARY)	2192	74.46%
Index 12928 (split/tbinc.IDX_processid)	55	1.87%
Index 12929 (split/tbinc.IDX_active)	48	1.63%
Free space	646	21.94%

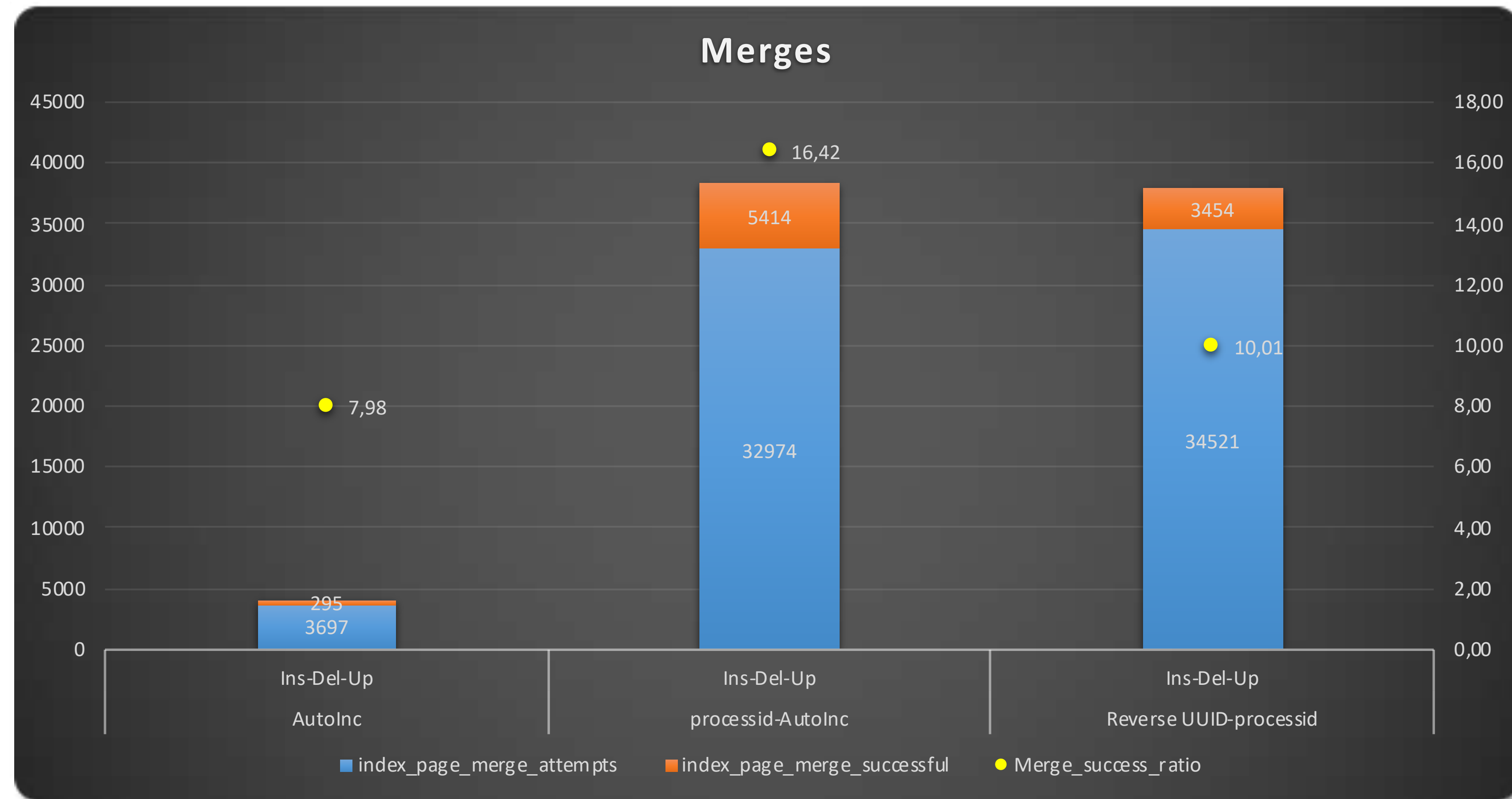
SOME GRAPHS AND NUMBERS – SPLITS

The first case will have more “compact” data distribution. This means it will also have better space utilization, while Processid with autoinc, and the semi-random nature of the UUID will cause a significant “sparse” page distribution (causing a higher number of pages and related split operations).



SOME GRAPHS AND NUMBERS – MERGES

auto-increment has less page merge attempts and success ratio than the other two types. The PK with Processis-autoinc (on the side other of the spectrum) has a higher number of merge attempts, but at the same time also a significantly higher success ratio at 16,42%, given that the “sparse” distribution left many pages partially empty.



SO WHAT?

During merge and split operations, InnoDB acquires an x-latch to the index tree.

On a busy system, this can easily become a source of concern.

This can cause index latch contention.

If no merges and splits (aka writes) touch only a single page, this is called an “optimistic” update in InnoDB, and the latch is only taken in S.

Merges and splits are called “pessimistic” updates, and take the latch in X.

CONCLUSIONS

- MySQL/InnoDB constantly performs these operations, and you have very limited visibility of them. But they can bite you, and bite hard, especially if using a spindle storage VS SSD (which have different issues, by the way).
- The sad story is there is also very little we can do to optimize this on the server side using parameters or some other magic. But the good news is there is A LOT that can be done at design time.
- Use a proper Primary Key and design a secondary index, keeping in mind that you shouldn't abuse of them. Plan proper maintenance windows on the tables that you know will have very high levels of inserts/deletes/updates.
- This is an important point to keep in mind. In InnoDB you cannot have fragmented records, but you can have a nightmare at the page-extent level. Ignoring table maintenance will cause more work at the IO level, memory and InnoDB buffer pool.
- You must rebuild some tables at regular intervals. Use whatever tricks it requires, including partitioning and external tools (pt-osc). Do not let a table to become gigantic and fully fragmented.
- Wasting disk space? Need to load three pages instead one to retrieve the record set you need? Each search causes significantly more reads?
That's your fault; there is no excuse for being sloppy!

Thanks!!!
Any question?





PERCONA

LIVE ONLINE

20-21 OCTOBER

2020