«Что у него внутри»
хранение данных на низком уровне

Николай Шаплов

www.postgrespro.ru

О докладчике

Николай Шаплов

- Сотрудник компании Postgres Professional
- Perl, C разработчик

Контакты

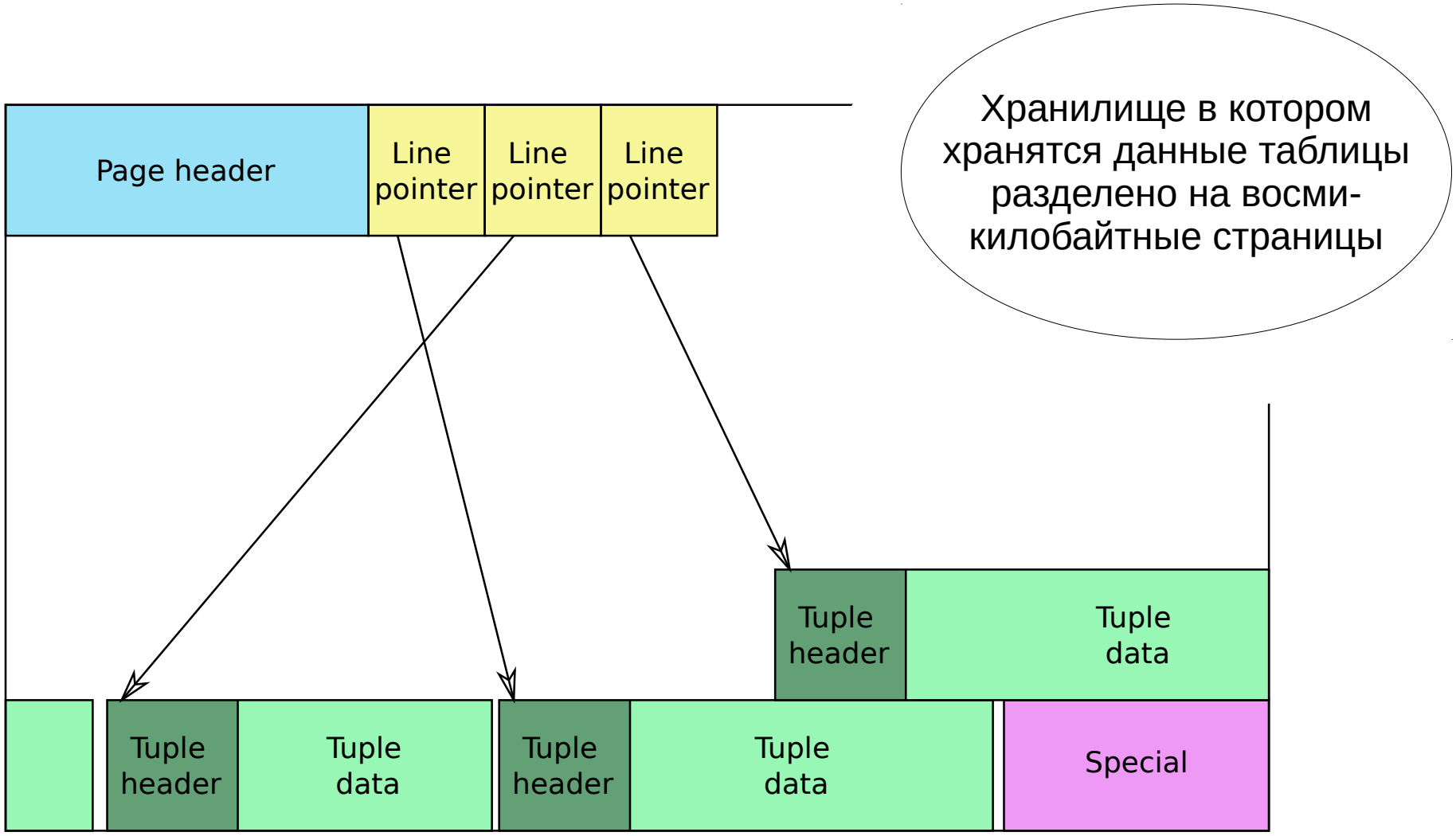
- E-mail: n.shaplov@postgrespro.ru
- Jabber: n.shaplov@postgrespro.ru

Postgres PROFESSIONAL

О докладе

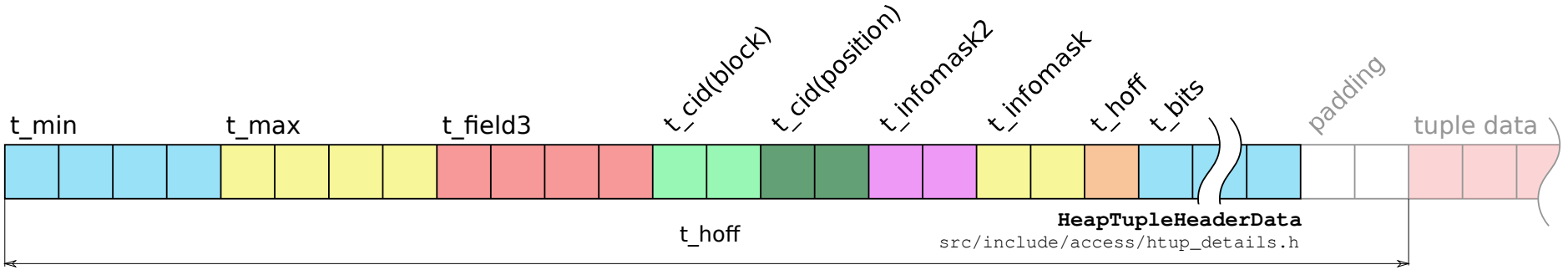
- Введение
- Атрибуты в записи
  - Атрибуты фиксированной длины
  - Атрибуты переменной длины
  - Значение NULL
- Выравнивание
- Большие атрибуты переменной длины
  - Сжатие
  - TOAST-хранилище
- Оптимизация
  - Уменьшение размера
  - Увеличение скорости

Хранение записей на странице



Хранилище в котором хранятся данные таблицы разделено на восми-килобайтные страницы

Заголовок записи / Tuple header



t_min
t_max
t_field3
} Видимость записи

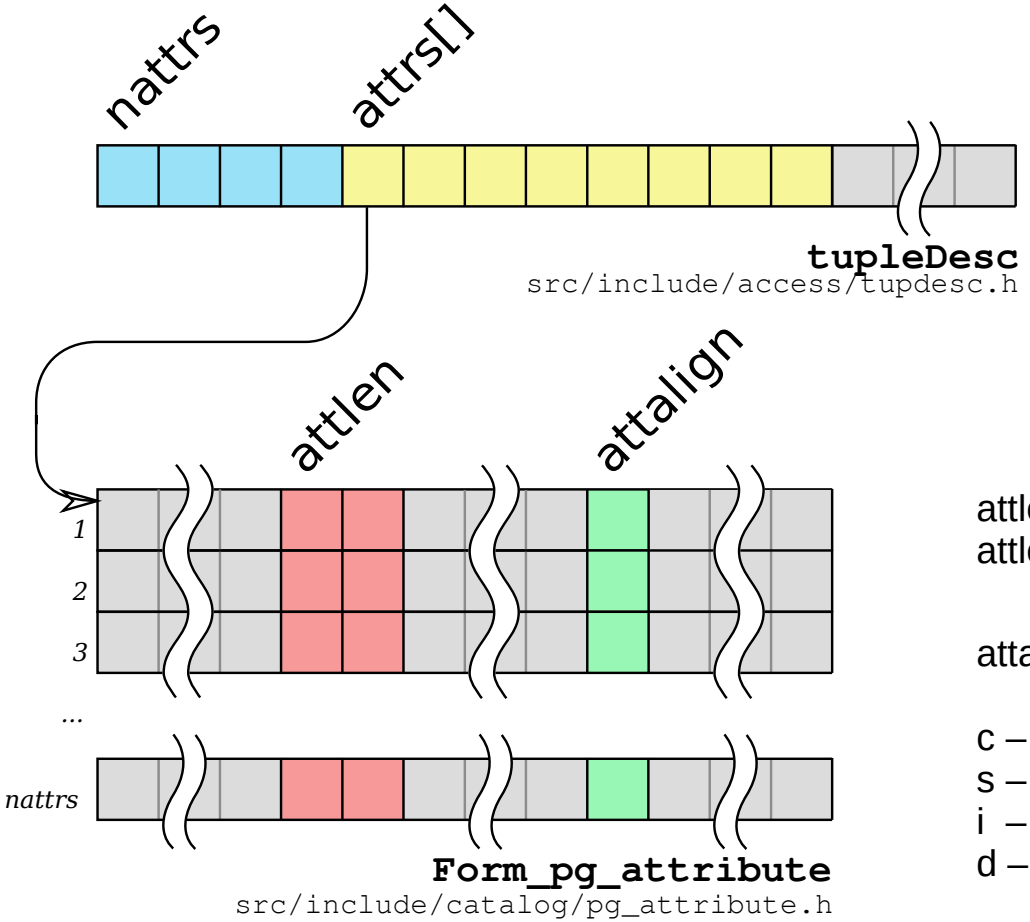
t_cid – положение записи в хранилище

t_infomask2
t_infomask
} Флаги + кол-во атрибутов

t_bits – маска NULL-значений
t_hoff – смещение области данных

Заголовок записи не содержит информации о том, как значения атрибутов размещены в области данных

Дескриптор записи / Tuple descriptor



Информация об атрибутах сохраненных в области данных находится в структуре tupleDesc, которую для каждой таблицы можно получить при помощи функции CreateTupleDescCopyConstr

attlen>0 – атрибут фиксированной длины
attlen=-1 – атрибут переменной длины

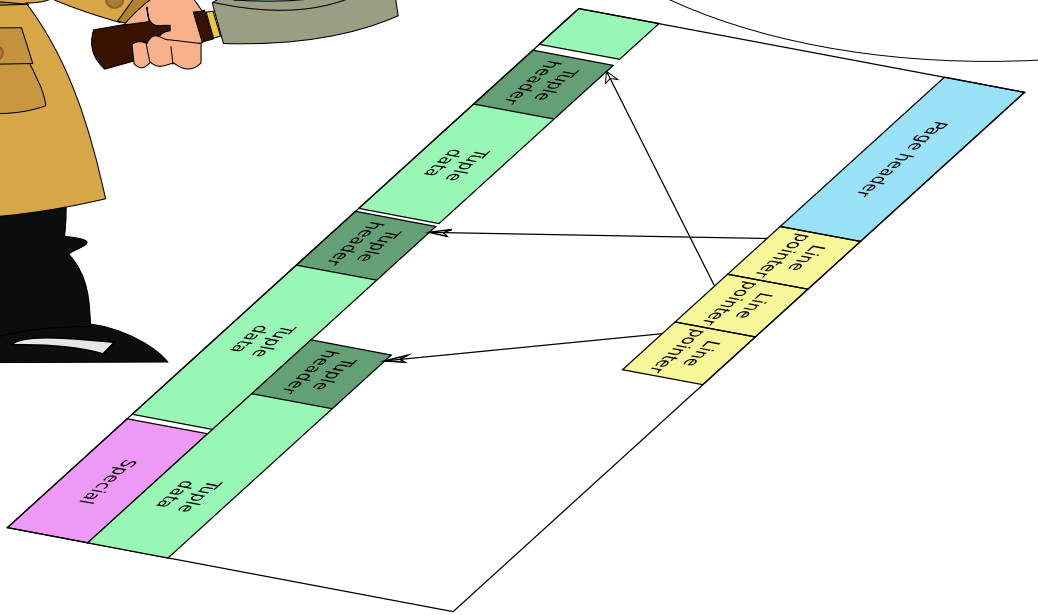
attalign:

c – без выравнивания
s – значение смещения округляется до кратности 2
i – значение смещения округляется до кратности 4
d – значение смещения округляется до кратности 4 или 8

Расширение pageinspect

Расширение pageinspect позволяет исследовать структуры данных сохраненных на диске

Патчи d6061f8 + 0271e27 позволяют просматривать не только заголовки, записей, но и область данных. Возможность будет представлена в postreSQL 9.6. Доступна в Postgres Pro 9.5 уже сейчас!



Хранение атрибутов фиксированной длины

```
test=# create table test (a int, b int, c int);
CREATE TABLE
test=# insert into test VALUES (1,2,3);
INSERT 0 1
test=# select lp, t_data from heap_page_items(get_raw_page('test', 0));
```

```
 lp | t_data
----+--------------------------------
1   | \x010000000200000003000000
```

Числа хранятся на диске как числа
Сюрприз :-)

```
test=# select * from heap_page_item_attrs(get_raw_page('test',0),'test'::regclass);
```

```
 lp | t_attrs
----+-----------------------------------------
1   | {"\\x01000000","\\x02000000","\\x03000000"}
```

Атрибуты фиксированной длины
Самостоятельная работа

Используя расширение pageinspect попробуйте выяснить как хранится атрибут типа date:

- Хранятся ли день, месяц и год как отдельные значения?
- Хранится ли дата как количество некоторых единиц относительно условного нуля?
- Что именно это за единицы?
- Что принято за ноль?

Шпаргалка:

```
create table test (d date);
insert into test values ('2016-02-13'),('2016-02-13'::date+2);
```

Атрибуты фиксированной длины
Самостоятельная работа

```
test=# create table test (s varchar);
CREATE TABLE
test=# insert into test VALUES (now()),('2000-01-01'),('2000-01-01'::date-1);
INSERT 0 1
test=# select lp, t_data from heap_page_items(get_raw_page('test', 0));
```

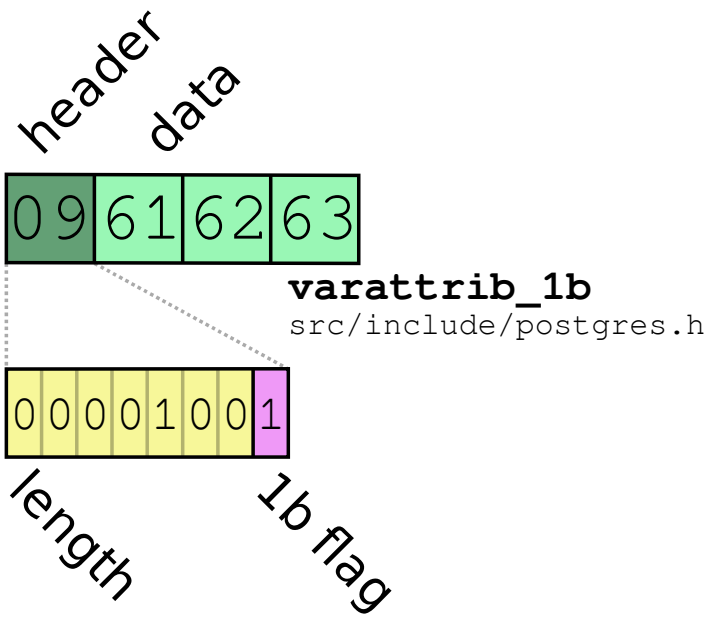
```
 lp | t_data
----+---------------
 1  | \xf4160000
 2  | \x00000000
 3  | \xffffffff
```

Хранение атрибутов переменной длины. Однобайтный заголовок

Однобайтный заголовок используется в некомпрессированных строках длиной до 126 символов

```
test=# create table test (s varchar);
CREATE TABLE
test=# insert into test VALUES ('abcd'),('abc');
INSERT 0 1
test=# select lp, t_data from
                    heap_page_items(get_raw_page('test', 0));
```

```
 lp  | t_data
-----+---------------
 1   | \x0b61626364
 2   | \x09616263
```

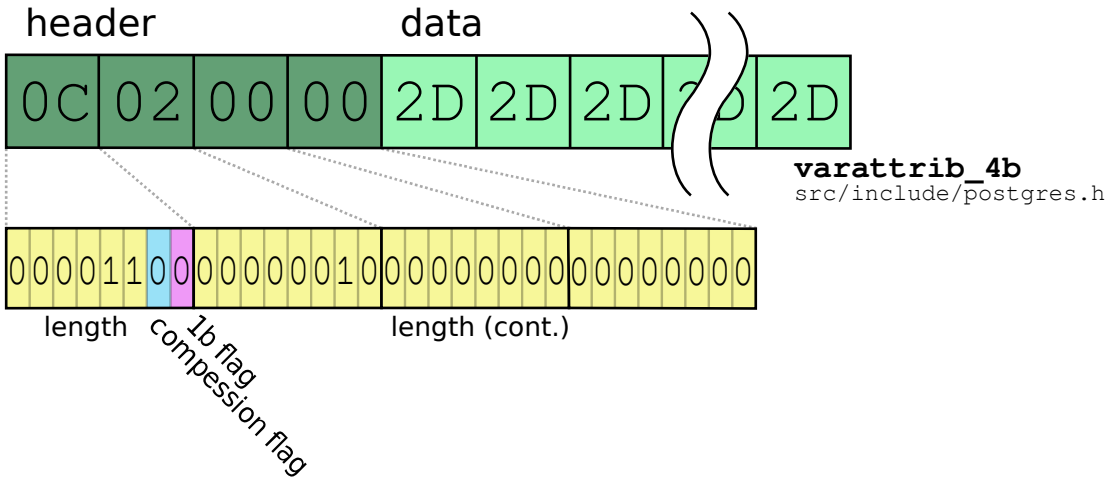


Хранение атрибутов переменной длины. Четырехбайтный заголовок

```
test=# create table test (a varchar);
CREATE TABLE
test=# insert into test VALUES (repeat('+',126)),(repeat('-',127));
INSERT 0 1
test=# select lp, t_data from
                heap_page_items(get_raw_page('test', 0));
```

Четырехбайтный заголовок используется для строк длиннее 126 байт, а так же для сжатых строк

```
 lp  |  t_data
-----+----------------------
   1 | \xff2b2b2b2b2b2b2b2b2b2b2b2b2b2b2b...
   2 | \x0c0200002d2d2d2d2d2d2d2d2d2d2d2d...
```



Атрибуты переменной длины
Самостоятельная работа

Представление массивов во внутренней структуре данных

- Убедитесь, что массивы хранятся как атрибут переменной длины;
- Убедитесь, что элементы массива хранятся в самой структуре данных;
- Найдите поле, в котором хранится размер массива;
- Хранится ли информация о типе данных элемента массива в каждой записи содержащий массив?
- Что еще интересного удалось узнать?

Шпаргалка:

```
create table test (a integer[]);
insert into test values ('{1,2,3,4}');
```

src/include/utils/array.h

```
/* A standard varlena array has the following internal structure:
 *    <vl_len_>     - standard varlena header word
 *    <ndim>        - number of dimensions of the array
 *    <dataoffset>  - offset to stored data, or 0 if no nulls bitmap
 *    <elemtype>    - element type OID
 *    <dimensions>  - length of each array axis (C array of int)
 *    <lower bnds>  - lower boundary of each dimension (C array of int)
 *    <null bitmap> - bitmap showing locations of nulls (OPTIONAL)
 *    <actual data> - whatever is the stored data */
```

Атрибуты переменной длины
Самостоятельная работа

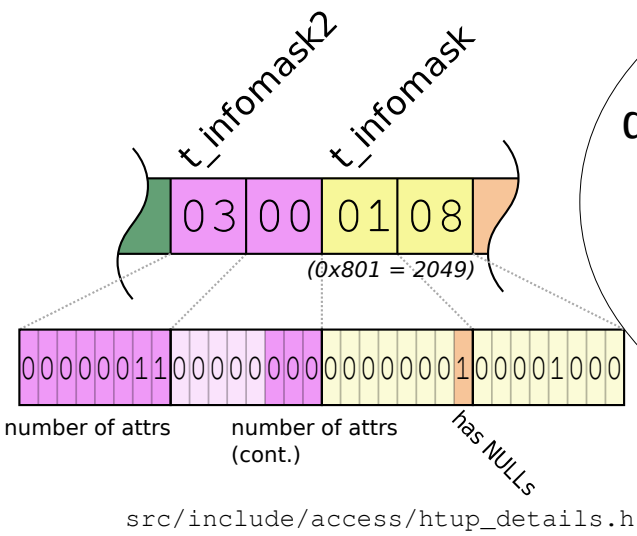
```
test=# create table test (a integer[]);
CREATE TABLE
test=# insert into test VALUES ('{255,127,63}'),('{255,127}'),('{255}');
INSERT 0 3
test=# select lp, t_data from heap_page_items(get_raw_page('test', 0));
```

```
 lp |                                t_data
----+-------------------------------------------------------------------
  1 | \x43010000000000000017000000030000000100000ff0000007f0000003f000000
  2 | \x3b010000000000000017000000020000000100000ff0000007f000000
  3 | \x33010000000000000017000000010000000100000ff000000
(3 rows)
```

Infomask и Infomask2

```
test=# create table test (a int, b int, c int);
CREATE TABLE
test=# insert into test VALUES (1,2,3),(1,null,3);
INSERT 0 2
test=# select lp, t_infomask2, t_infomask, t_bits, t_data from heap_page_items(get_raw_page('test', 0));
```

```
lp  | t_infomask2 | t_infomask | t_bits   | t_data
----+-------------+------------+----------+--------------------------------
1   |           3 |       2048 |          | \x0100000020000003000000
2   |           3 |       2049 | 10100000 | \x010000003000000
```



src/include/access/htup_details.h

t_infomask и t_infomask2 содержат массив битовых флагов, сообщающих о различных о свойствах хранимых в записи данных. Например флаг has NULLs указывающий есть ли среди атрибутов NULL-значения

Кроме того t_infomask2 хранит количество атрибутов в записи

NULL-значения

Если атрибут имеет значение NULL, то он вообще не представлен в области данных, зато для него установлен специальный бит в поле t_bits

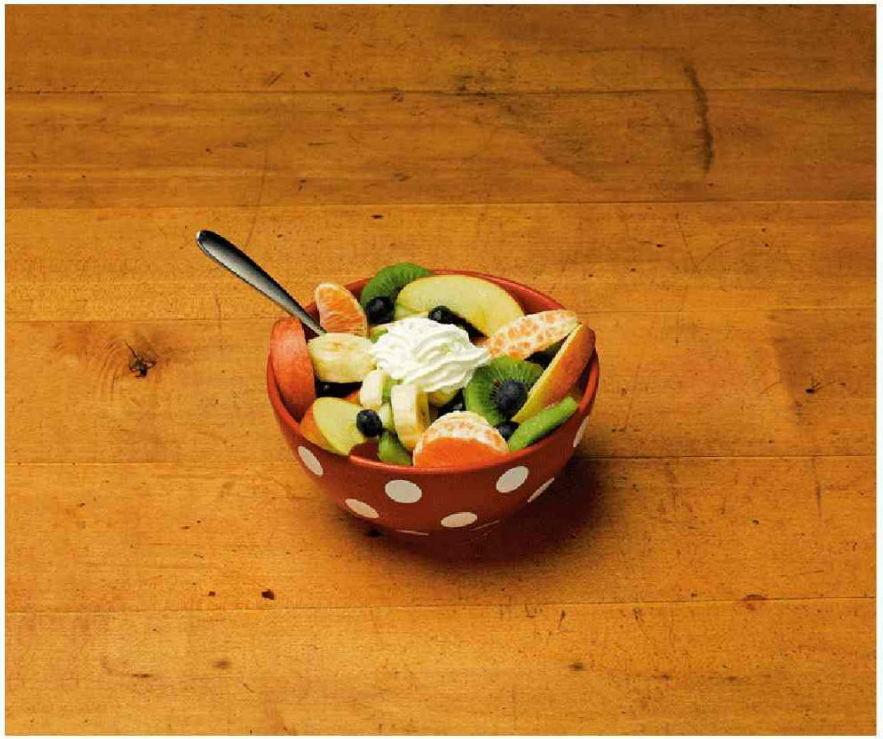
```
test=# create table test (a int, b int, c int);
CREATE TABLE
test=# insert into test VALUES (1,2,3),(1,null,3);
INSERT 0 2
test=# select lp, t_infomask, t_bits, t_data from
                        heap_page_items(get_raw_page('test', 0));
```

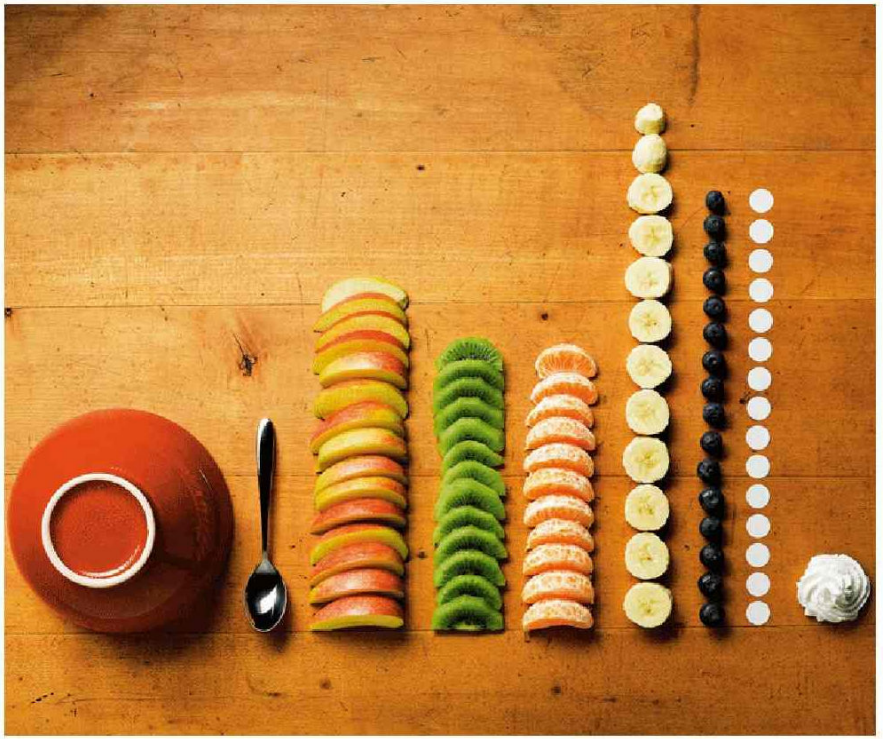
```
 lp  | t_infomask | t_bits   | t_data
-----+------------+----------+----------------------------
 1   |       2048 |          | \x010000000200000003000000
 2   |       2049 | 10100000 | \x0100000003000000
```

```
test=# select lp, t_infomask, t_bits, t_attrs from
                                heap_page_item_attrs(get_raw_page('test',0),'test'::regclass)
```

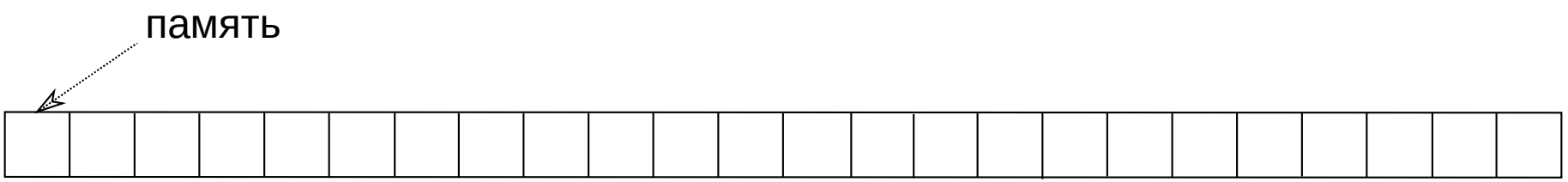
```
 lp | t_infomask | t_bits   | t_attrs
----+------------+----------+-----------------------------------------
 1  |       2048 |          |{"\\x01000000","\\x02000000","\\x03000000"}
 2  |       2049 |10100000  |{"\\x01000000",NULL,"\\x03000000"}
```

Alignment

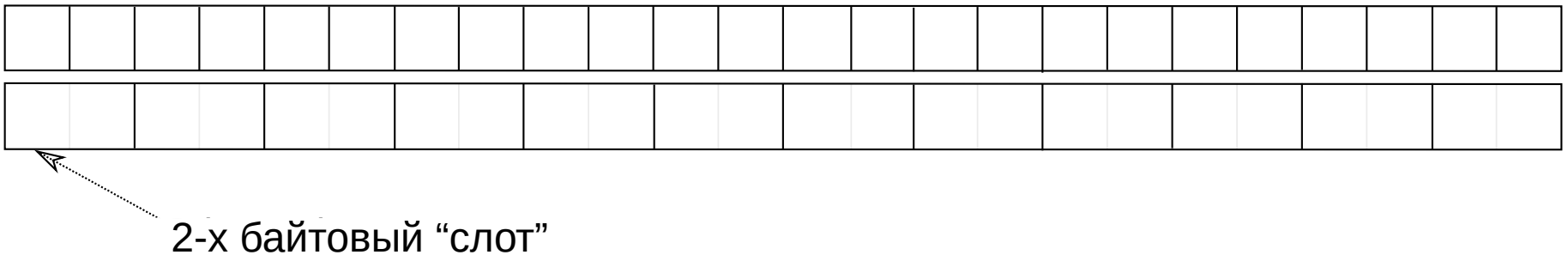




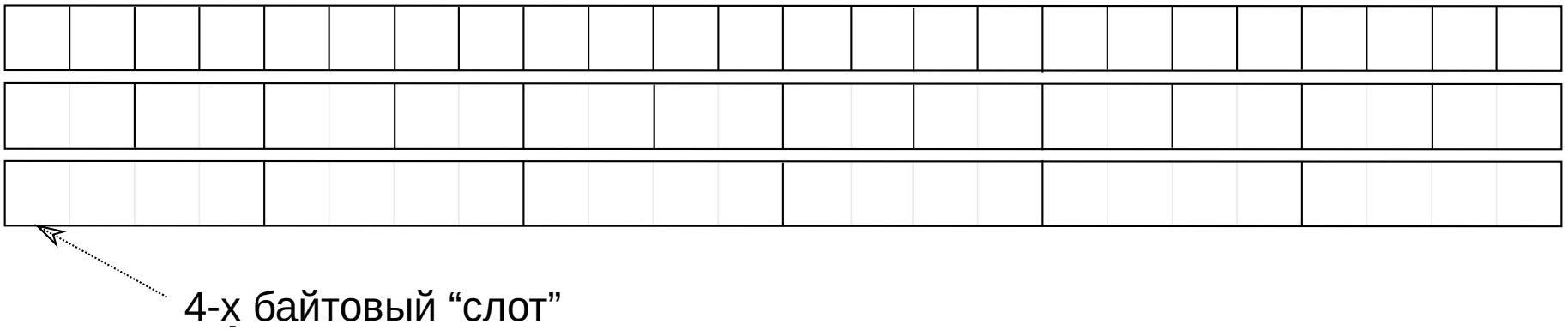
Выравнивание



Выравнивание

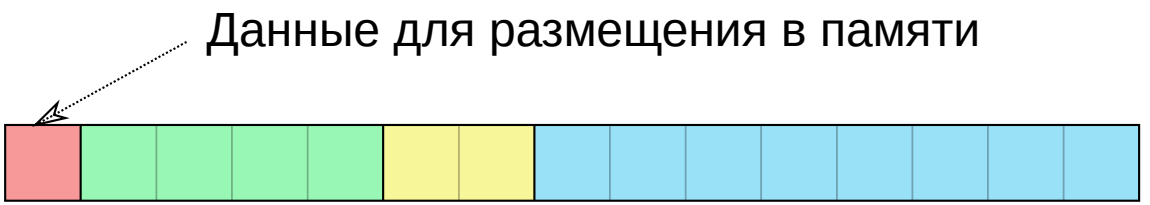


Выравнивание

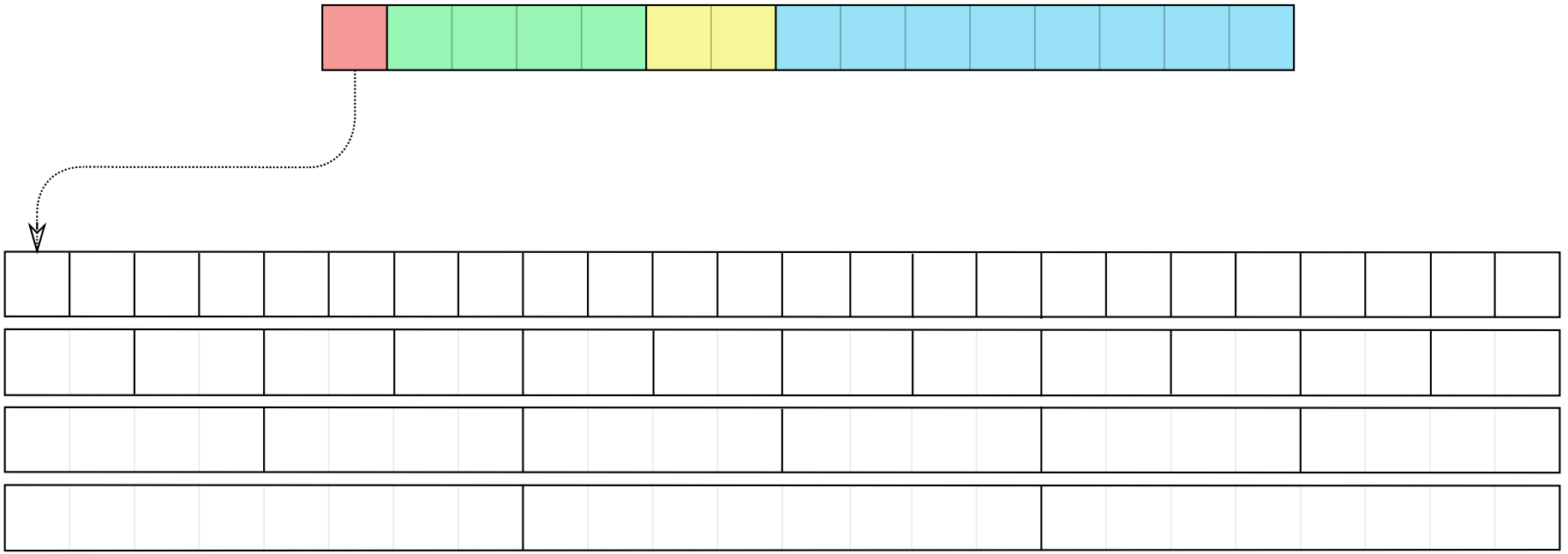


Выравнивание

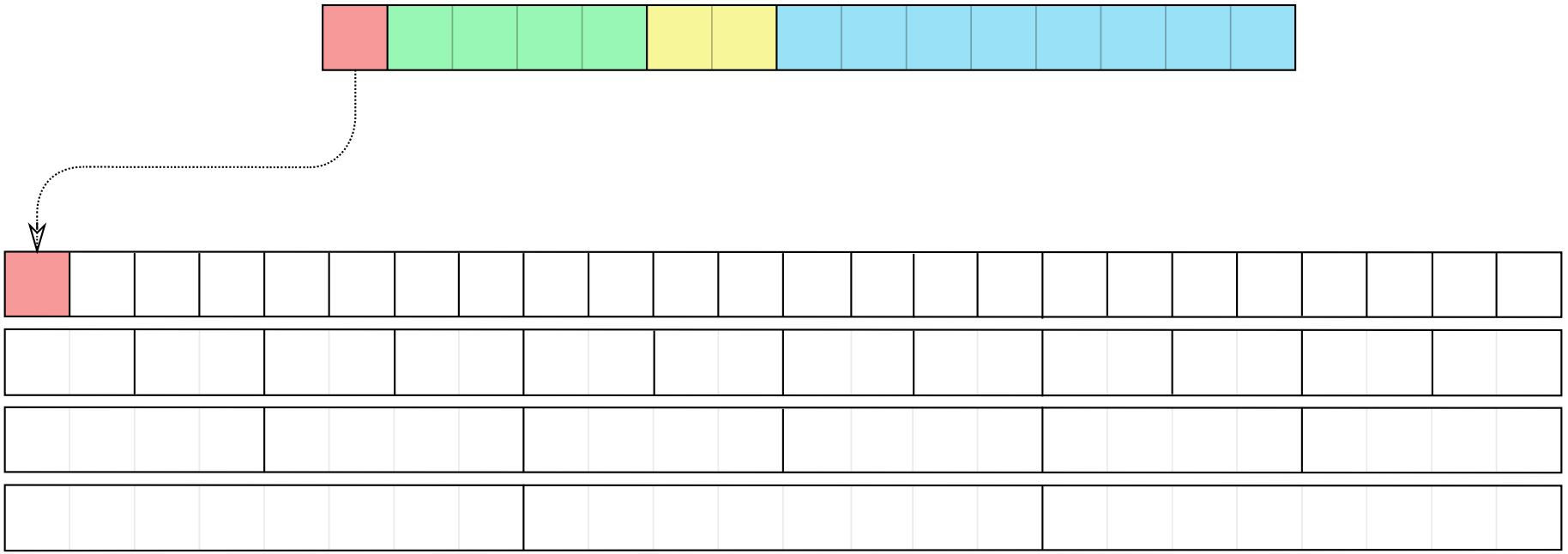
Выравнивание



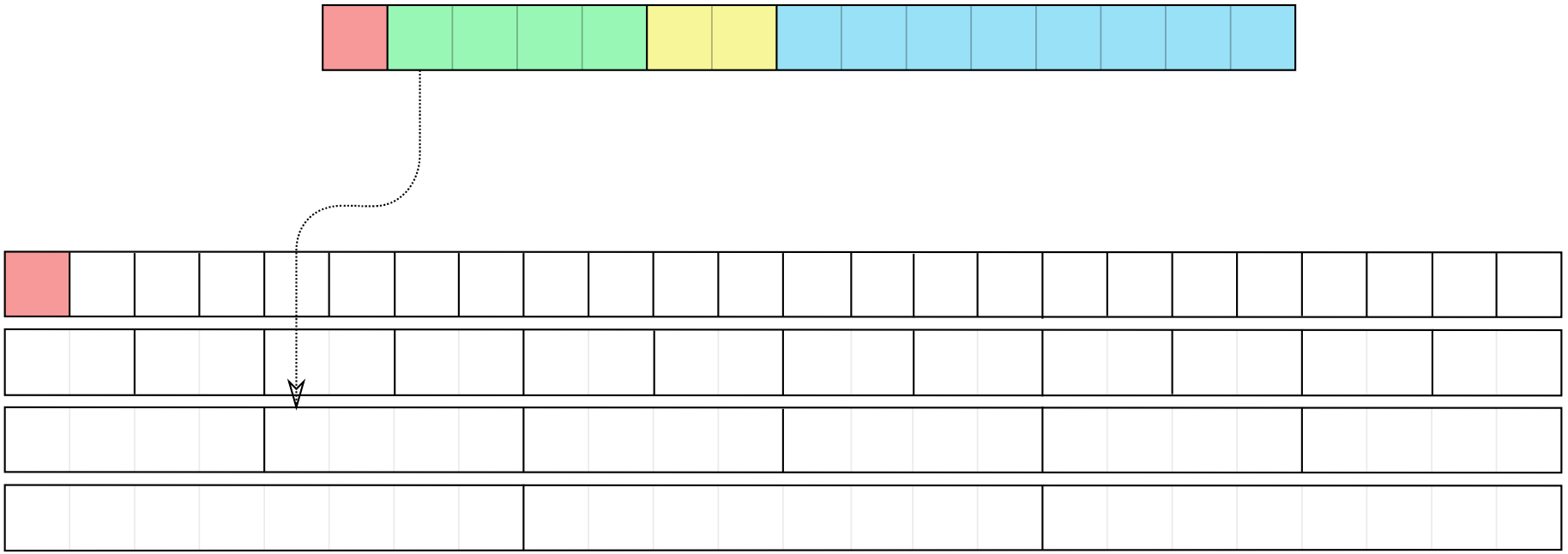
Выравнивание



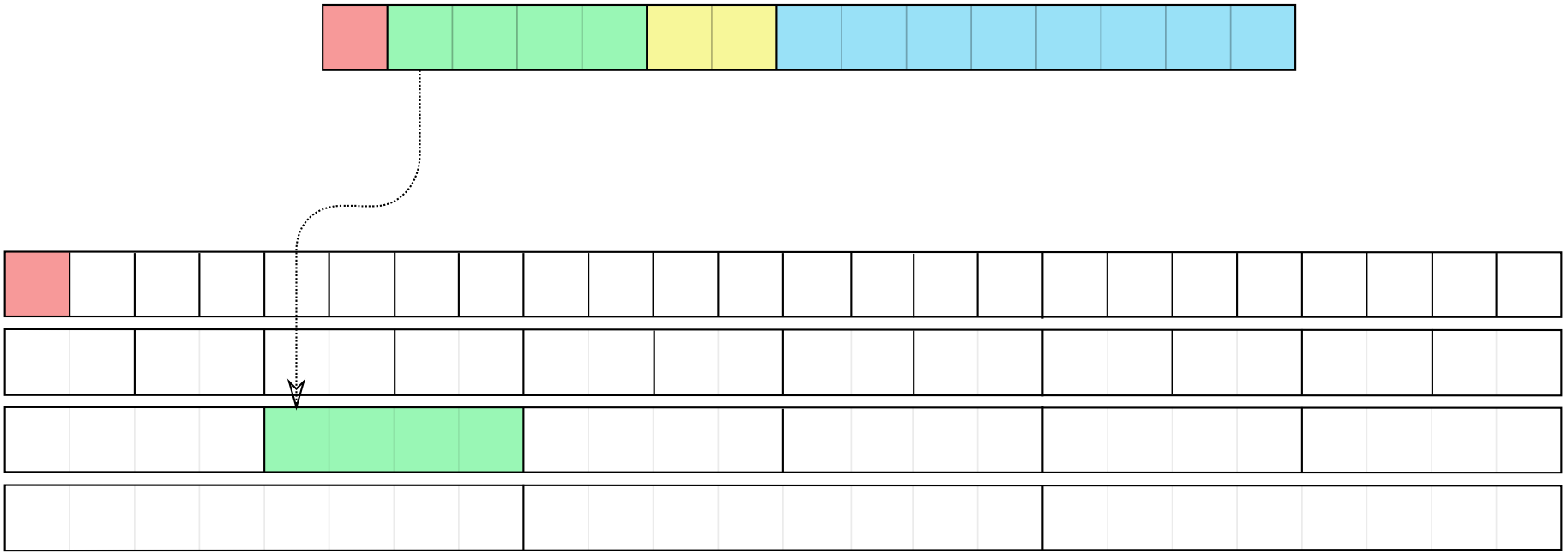
Выравнивание



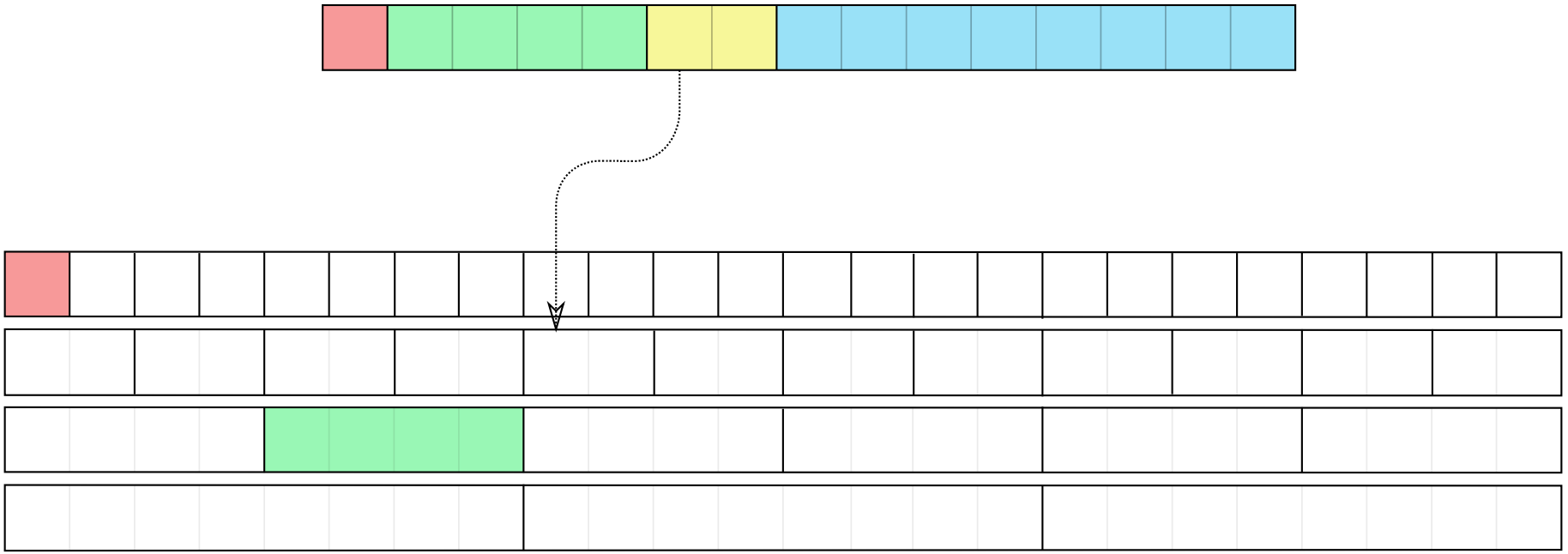
Выравнивание



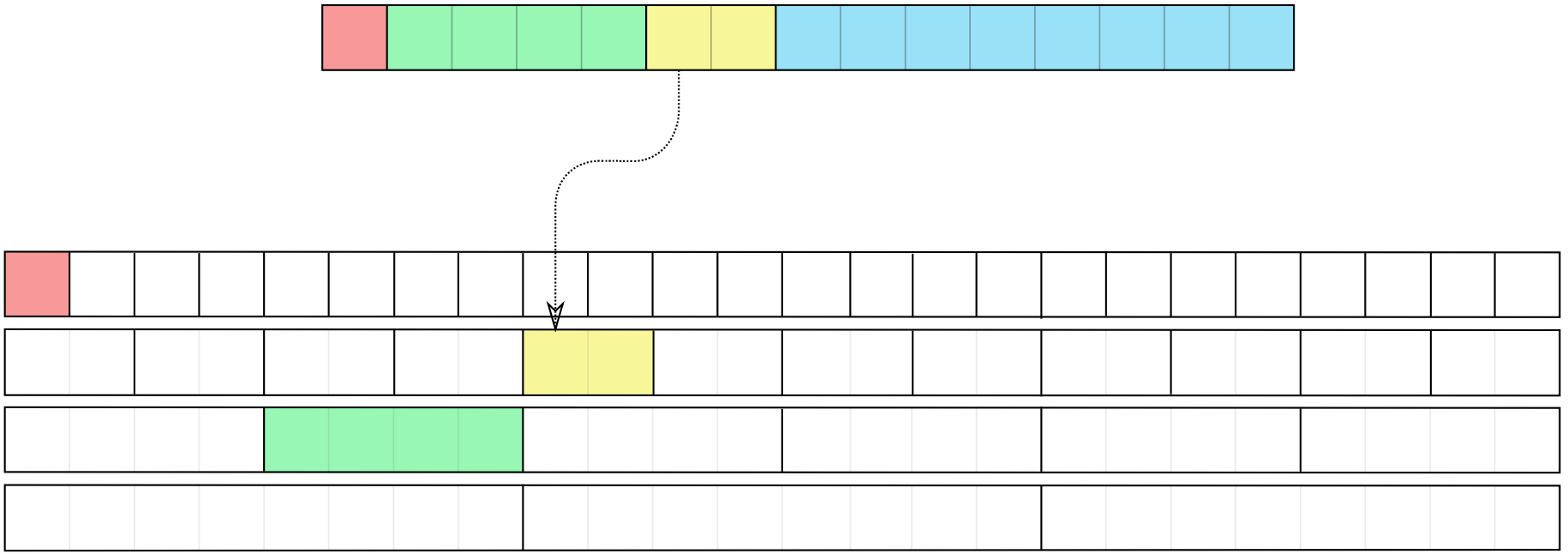
Выравнивание



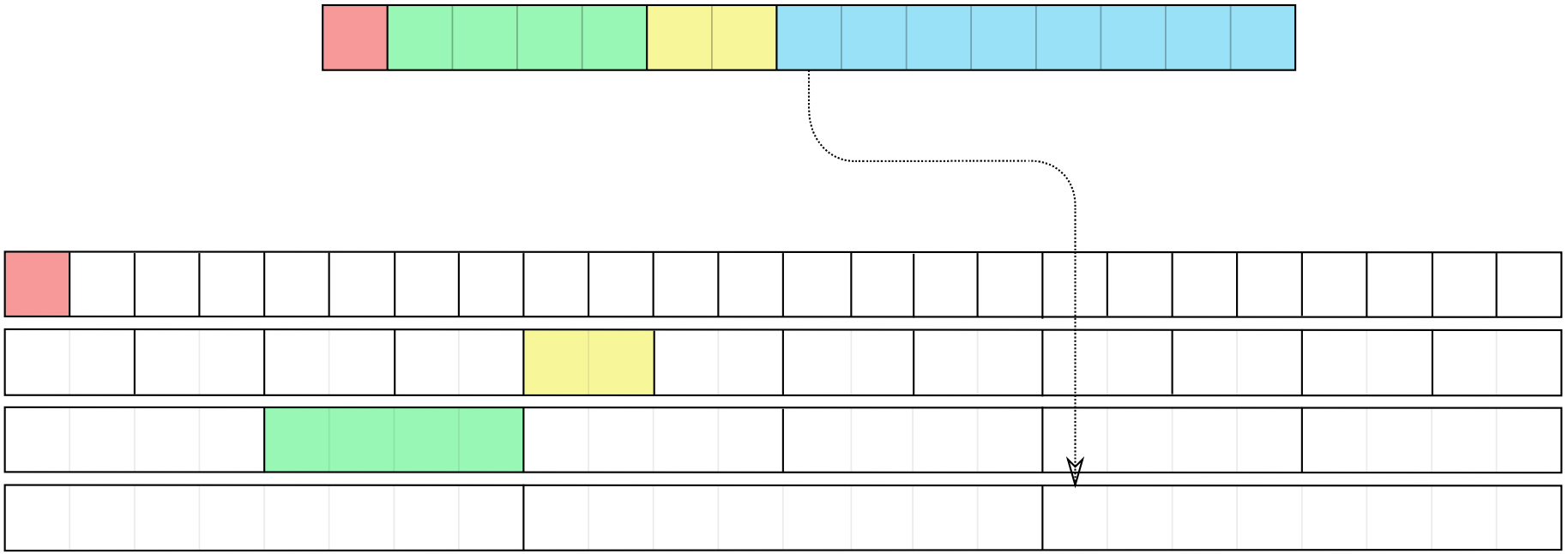
Выравнивание



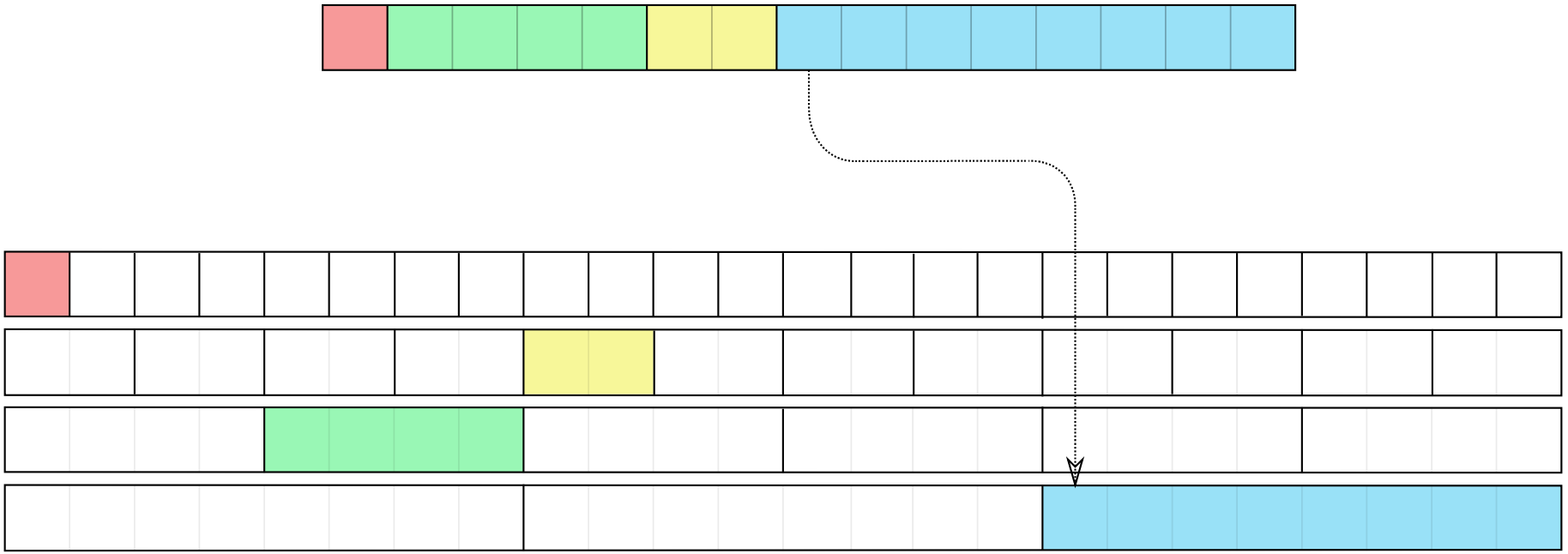
Выравнивание



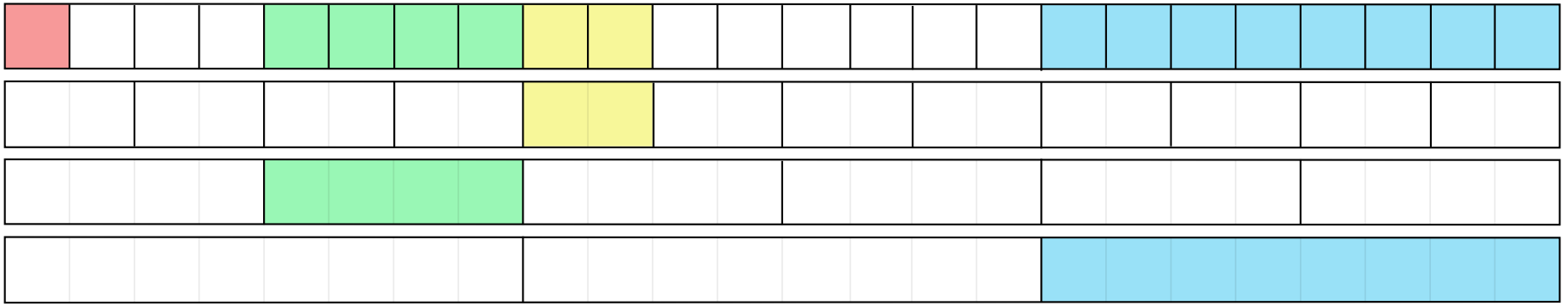
Выравнивание



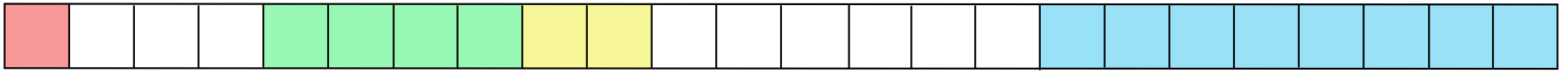
Выравнивание



Выравнивание



Выравнивание

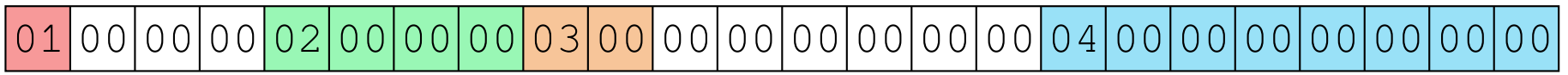


Выравнивание атрибутов фиксированной длины

```
test=# create table test (a boolean, b int, c smallint, d bigint);
CREATE TABLE
test=# insert into test VALUES ('t',2,3,4);
INSERT 0 1
test=# select lp, t_data from
                heap_page_items(get_raw_page('test', 0));
```

Атрибуты фиксированной длинны размещены в области данных ровно так, как показано в предыдущем примере

```
 lp |                          t_data
----+-----------------------------------------------------
  1 | \x01000000020000000300000000000000040000000000000000
```

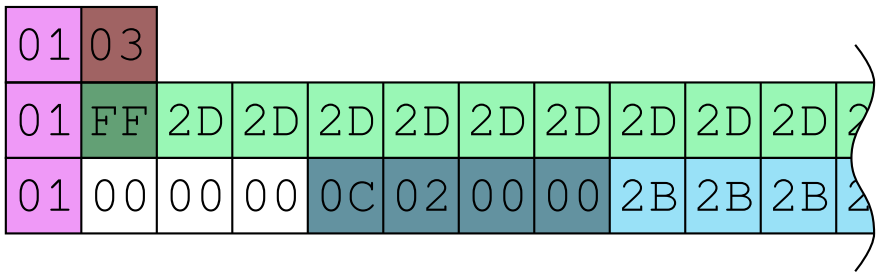


Выравнивание атрибутов переменной длины

```
test=# create table test (a boolean, b varchar);
CREATE TABLE
test=# insert into test VALUES ('t',''),
test-#                         ('t',repeat('-',126)),
test-#                         ('t',repeat('+',127));
INSERT 0 3
test=# select lp, t_data from heap_page_items(get_raw_page('test', 0));
```

```
lp | t_data
---+------------------------------
1  | \x0103
2  | \x01ff2d2d2d2d2d2d2d2d...
3  | \x010000000c020000 2b2b2b2b2b...
```

В случае атрибутов переменной длины фактически происходит выравнивание заголовка



Выравнивание на 64-х и 32-х битных платформах

```
# create table test (b int, c bigint);
CREATE TABLE
# insert into test VALUES (1,2);
INSERT 0 1
```

Для процессоров разной разрядности выравнивание работает по-разному

64bit

```
lp | t_data
---+-----------------------------------
1  | \x0100000000000000 0200000000000000
```

32bit

```
lp | t_data
---+-----------------------------
1  | \x0100000002000000 0000000000
```

"Недостача" атрибутов в записи

```
test=# create table test (a int, b int);
CREATE TABLE
test=# insert into test VALUES (1,10);
INSERT 0 2
test=# ALTER TABLE test ADD COLUMN c int;
ALTER TABLE
test=# insert into test VALUES (3,30,300);
INSERT 0 1
```

Если вы добавили новый атрибут со значением по умолчанию NULL, то postgres не будет обновлять записи в хранилище, а будет все недостающие атрибуты считать равными NULL

```
test=# select lp, t_infomask2, t_data from heap_page_items(get_raw_page('test', 0));

lp   |   t_infomask2   | t_data
----+-------------+----------------------------
1    | 2             | \x010000000a000000
2    | 3             | \x030000001e0000002c010000
```

```
test=# select lp, t_infomask2, t_attrs from
                   heap_page_item_attrs(get_raw_page('test',0),'test'::regclass);

lp  | t_infomask2 | t_attrs
---+----------+---------------------------------------
1   | 2         | {"\\x01000000","\\x0a000000",NULL}
2   | 3         | {"\\x03000000","\\x1e000000","\\x2c010000"}
```

Добавление и удаление атрибутов Самостоятельная работа

Попробуйте при помощи расширения pageinspect выяснить что происходит при:

- добавлении нового атрибута с непустым значением по умолчанию;
- удалении существующего атрибута;

Шпаргалка:

```
alter table test add column b integer default 12;

alter table test drop column a;
```

Подсказки:

- Поймать полное пересоздание хранилища можно по пропавшим из хранилища удаленным записям;
- Форсировать пересоздание хранилища можно командой vacuum full. Это приведет к пересозданию всех записей, и все следы оптимизаций будут очищены

Добавление атрибутов
Самостоятельная работа

```
test=# create table test (a integer);
CREATE TABLE
test=# insert into test values (1),(2),(3),(4);
INSERT 0 4
test=# delete from test where a=3;
DELETE 1
test=# SELECT lp, t_xmin, t_xmax, t_attrs FROM
heap_page_item_attrs(get_raw_page('test', 0), 'test'::regclass);

 lp | t_xmin | t_xmax |     t_attrs
----+--------+--------+-----------------
  1 |    826 |      0 | {"\\x01000000"}
  2 |    826 |      0 | {"\\x02000000"}
  3 |    826 |    827 | {"\\x03000000"}
  4 |    826 |      0 | {"\\x04000000"}
(4 rows)
```

При добавлении атрибута со значением по умолчанию хранилище просто пересоздается

```
test=# alter table test add column b integer default 12;
ALTER TABLE
test=# SELECT lp, t_xmin, t_xmax, t_attrs FROM
heap_page_item_attrs(get_raw_page('test', 0), 'test'::regclass);
 lp | t_xmin | t_xmax |            t_attrs
----+--------+--------+-------------------------------
  1 |    828 |      0 | {"\\x01000000","\\x0c000000"}
  2 |    828 |      0 | {"\\x02000000","\\x0c000000"}
  3 |    828 |      0 | {"\\x04000000","\\x0c000000"}
(3 rows)
```

Удаление атрибутов Самостоятельная работа

```
test=# create table test (a integer, b integer, c integer);
CREATE TABLE
test=# insert into test values (1,2,16),(2,4,32),(3,6,48);
INSERT 0 3
test=# alter table test drop column b;
ALTER TABLE

test=# SELECT lp, t_xmin, t_xmax, t_attrs FROM
heap_page_item_attrs(get_raw_page('test', 0), 'test'::regclass);

 lp | t_xmin | t_xmax |                    t_attrs
----+--------+--------+-----------------------------------------------
  1 |    835 |      0 | {"\\x01000000","\\x02000000","\\x10000000"}
  2 |    835 |      0 | {"\\x02000000","\\x04000000","\\x20000000"}
  3 |    835 |      0 | {"\\x03000000","\\x06000000","\\x30000000"}
(3 rows)

test=# insert into test values (4,64);
INSERT 0 1

test=# SELECT lp, t_xmin, t_xmax, t_attrs FROM
heap_page_item_attrs(get_raw_page('test', 0), 'test'::regclass);
 lp | t_xmin | t_xmax |                    t_attrs
----+--------+--------+-----------------------------------------------
  1 |    835 |      0 | {"\\x01000000","\\x02000000","\\x10000000"}
  2 |    835 |      0 | {"\\x02000000","\\x04000000","\\x20000000"}
  3 |    835 |      0 | {"\\x03000000","\\x06000000","\\x30000000"}
  4 |    837 |      0 | {"\\x04000000",NULL,"\\x40000000"}
(4 rows)
```

При удалении атрибута, данные из хранилища не удаляются, атрибут лишь помечается в tupleDescriptor как удаленный

Все новые записи вместо удаленного атрибута содержат значение NULL

Postgres PROFESSIONAL

Хранение больших данных

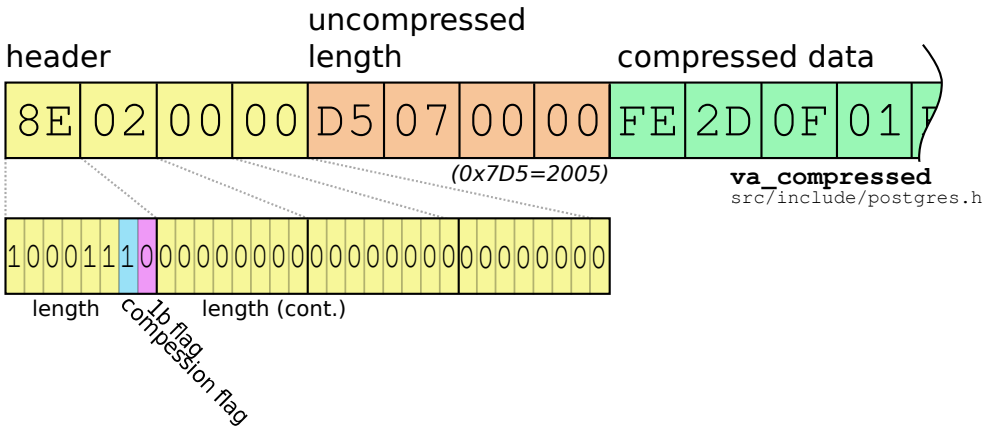
- Компрессия
- TOAST

Компрессия значений атрибутов

Если размер записи превышает ~2k postgres попытается сжать данные в атрибутах переменной длины (начиная с самого длинного)

```
test=# create table test (a varchar);
CREATE TABLE
test=# insert into test VALUES (repeat('-',2004)),
test-#                         (repeat('-',2005));
INSERT 0 2
test=# select lp, t_data from heap_page_items(get_raw_page('test', 0));

lp |    t_data
---+-----------------------------------------------------------------------------
1  | \x601f00002d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d2d.....
2  | \x8e000000d5070000fe2d0f01ff0f01ff0f01ff0f01ff0f01ff0f01ff0f01ff010f014b
```

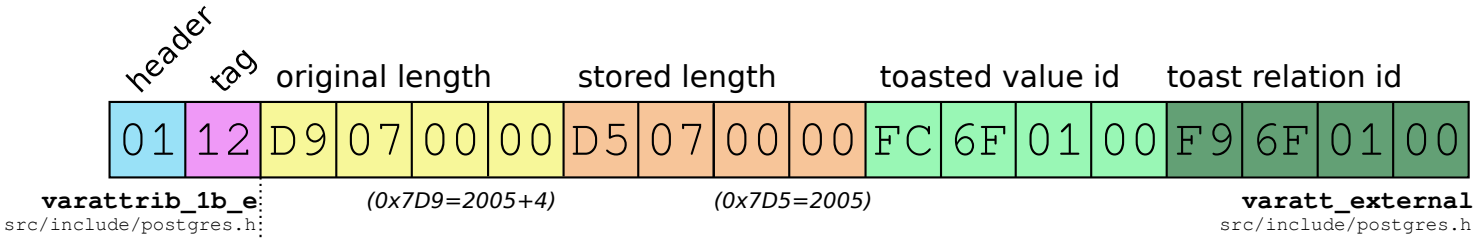


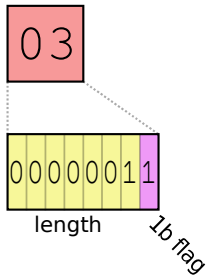
Вынос значение атрибута в TOAST-хранилище

```
test=# create table test (a varchar);
CREATE TABLE
test=# ALTER TABLE test ALTER a SET STORAGE EXTERNAL;
ALTER TABLE
test=# insert into test VALUES (repeat('-',2005),
test-#                          (''));
INSERT 0 2
test=# select lp, t_attrs from
                heap_page_items(get_raw_page('test', 0));
```

Если все атрибуты переменной длины сжаты, а запись все равно длиннее ~2k, postgres начинает выносить значения атрибутов в TOAST-хранилище (начиная с самого длинного)

```
lp  |  t_attrs
----+-----------------------------------------
1   | \x0112d9070000d5070000fc6f0100f96f0100
2   | \x03
```





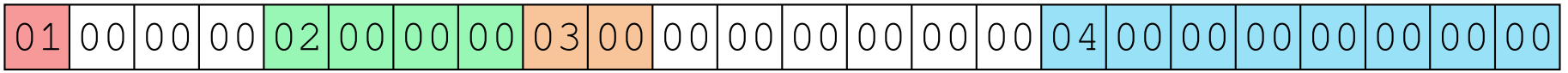
Минимально возможное значение однобайтного заголовка – 0x03. Поэтому 0x01 используется как маркер вынесенного в TOAST атрибута

Postgres PROFESSIONAL

Optimization

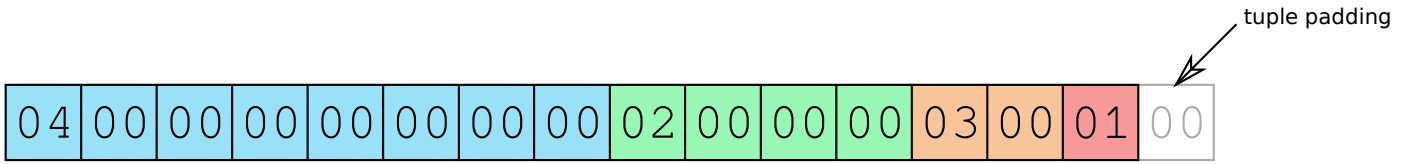
Оптимизация размера

```
test=# create table test (a boolean, b int, c smallint, d bigint);
```



Измените порядок атрибутов чтобы избежать пустот вызванных выравниванием

```
test=# create table test (d bigint, b int, c smallint, a boolean);
```



Оптимизация скорости доступа

```
create table test (a smallint, b int, c int);
insert into test VALUES (0,   1,  2),
                        (16,  17, NULL),
                        (NULL,33, 34),
                        (64,  65, 66);
```

00	00	00	00	01	00	00	00	02	00	00	00
10	00	00	00	11	00	00	00				
21	00	00	00	22	00	00	00				
40	00	00	00	41	00	00	00	42	00	00	00

Если в записи все атрибуты фиксированной длинны и ни один из них не NULL, то положение атрибута в области данных – неизменно

То же верно для атрибутов перед которыми нет NULL'ов и атрибутов переменной длинны

Оптимизация скорости доступа

Как разместить атрибуты в записи для увеличения скорости доступа:

- Первыми должны идти атрибуты фиксированной длины, не принимающие значения NULL
- За ними – атрибуты фиксированной длины принимающие значение NULL с малой вероятностью
- Все атрибуты переменной длины должны идти в конце записи

Оптимизация скорости доступа

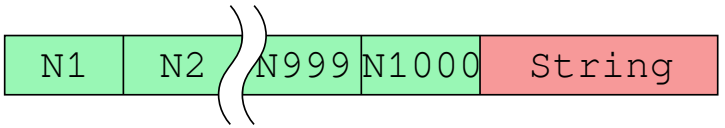
- 1 000 атрибутов типа bigint: N1..N1000
- Одна varchar строка в начале или конце записи
- 1 000 000 записей
- Запрос для тестирования: SELECT SUM(N1000)

Тестовое окружение
CPU: 4 socket Xeon V7, 15 Core per socket
Shared Memory: 100Gb

Параметры pgbench
clients: 100
jobs: 100
time: 360

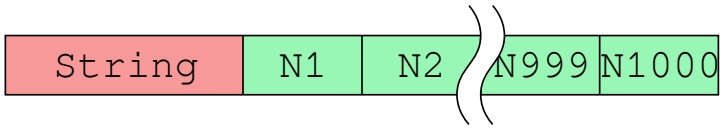
А какое ускорение может дать правильное расположение атрибутов? Попробуем это показать на вот таком синтетическом тесте

Строка в конце



tps = 64.373621

Строка в начале



tps = 50.387503

64.373621 / 50.387503 = **1.27**

Postgres PROFESSIONAL

Postgres PROFESSIONAL

Look for a newer versions at

https://github.com/dhyannataraj/tuple-internals-presentation