Яндекс

UUIDv7: стандарт и реализации

Андрей Бородин, PostgreSQL hacker

Про меня

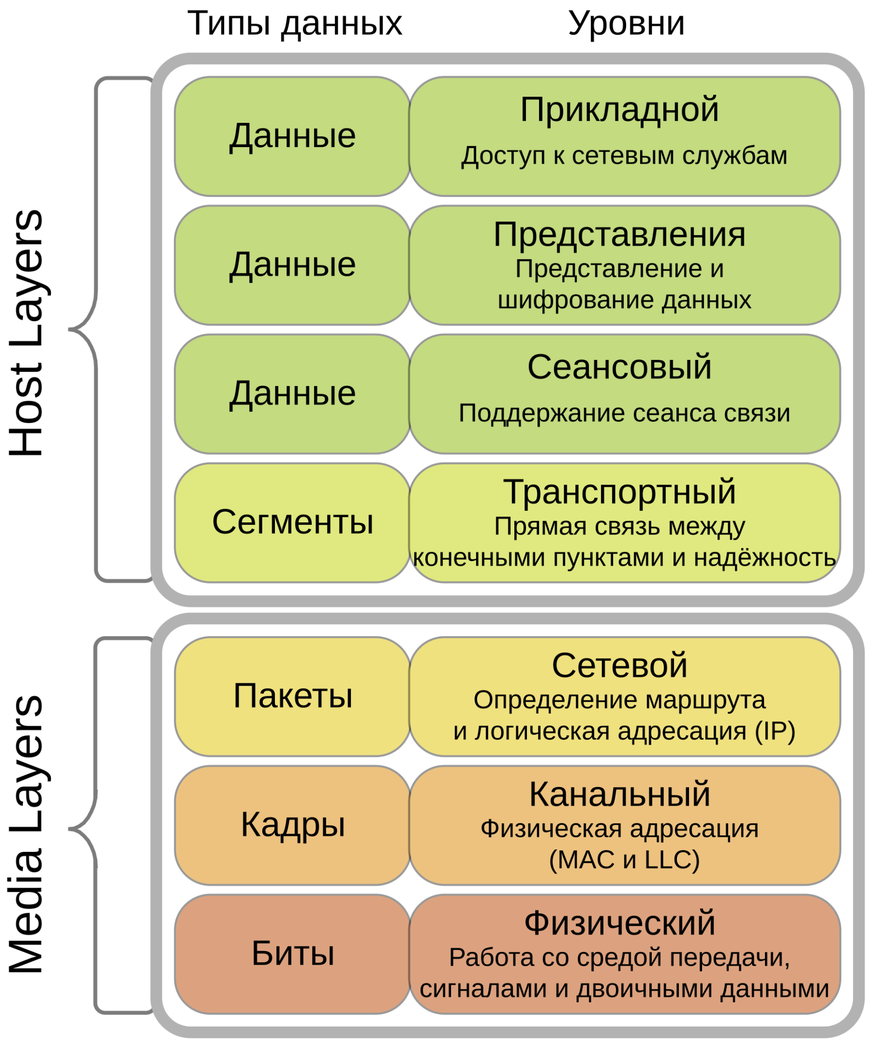
- Пишу Postgres и другие базы для Yandex Cloud
- Исследую управление данными с 2008 года
- Поддерживаю пачку open source-проектов


Open Source
Кирилл
Решке
Yandex Cloud
Денис
Волков
Yandex Cloud
КИРИЛЛ
ГАВРИЛОВ
Яндекс
Младший разработчик
HighLoad++
Яндекс
Участник
serverless
yandex cloud
YDB
TOM and JERRY

Идентификация

https://ru.wikipedia.org/wiki/Парадокс_Банаха_—_Тарского

OSI


Типы данных
Уровни
Host Layers
Media Layers
Данные
Прикладной
Доступ к сетевым службам
Данные
Представления
Представление и
шифрование данных
Данные
Сеансовый
Поддержание сеанса связи
Сегменты
Транспортный
Прямая связь между
конечными пунктами и надёжность
Пакеты
Сетевой
Определение маршрута
и логическая адресация (IP)
Кадры
Канальный
Физическая адресация
(MAC и LLC)
Биты
Физический
Работа со средой передачи,
сигналами и двоичными данными

В науке о вычислениях две сложные проблемы:

- Инвалидация кэша
- Наименование вещей
- Ошибки на единицу

Phil Carlton

Primary Key

```
CREATE TABLE employees (
   employee_id   NUMBER         NOT NULL,
   first_name    VARCHAR(1000) NOT NULL,
   last_name     VARCHAR(1000) NOT NULL,
   date_of_birth DATE          NOT NULL,
   phone_number  VARCHAR(1000) NOT NULL,
   CONSTRAINT employees_pk PRIMARY KEY (employee_id)
)
```

```
SELECT first_name, last_name
  FROM employees
 WHERE employee_id = 123
```

https://use-the-index-luke.com/sql/where-clause/the-equals-operator/primary-keys

Просто сделай SEQUENCE

Нормальный пользователь Postgres’а

Проблемы

- Sequence живёт в БД
- При этом не транзакционный

A Million Random Digits with 100,000 Normal Deviates

73735	45963	78134	63873
02965	58303	90708	20025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	20422	05720
15838	47174	76866	14330
89793	34378	08730	56522
78155	22466	81978	57323
16381	66207	11698	99314
75002	80827	53867	37797
99982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
30500	28220	12444	71840

RFC 4122: UUID

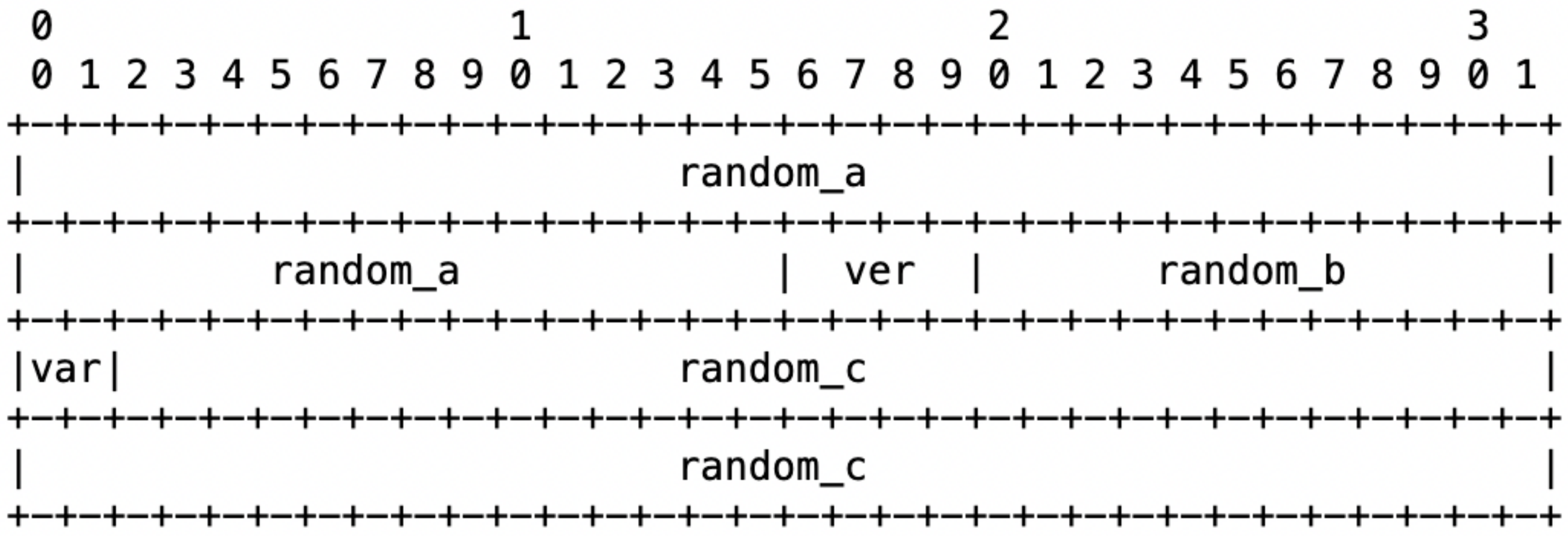
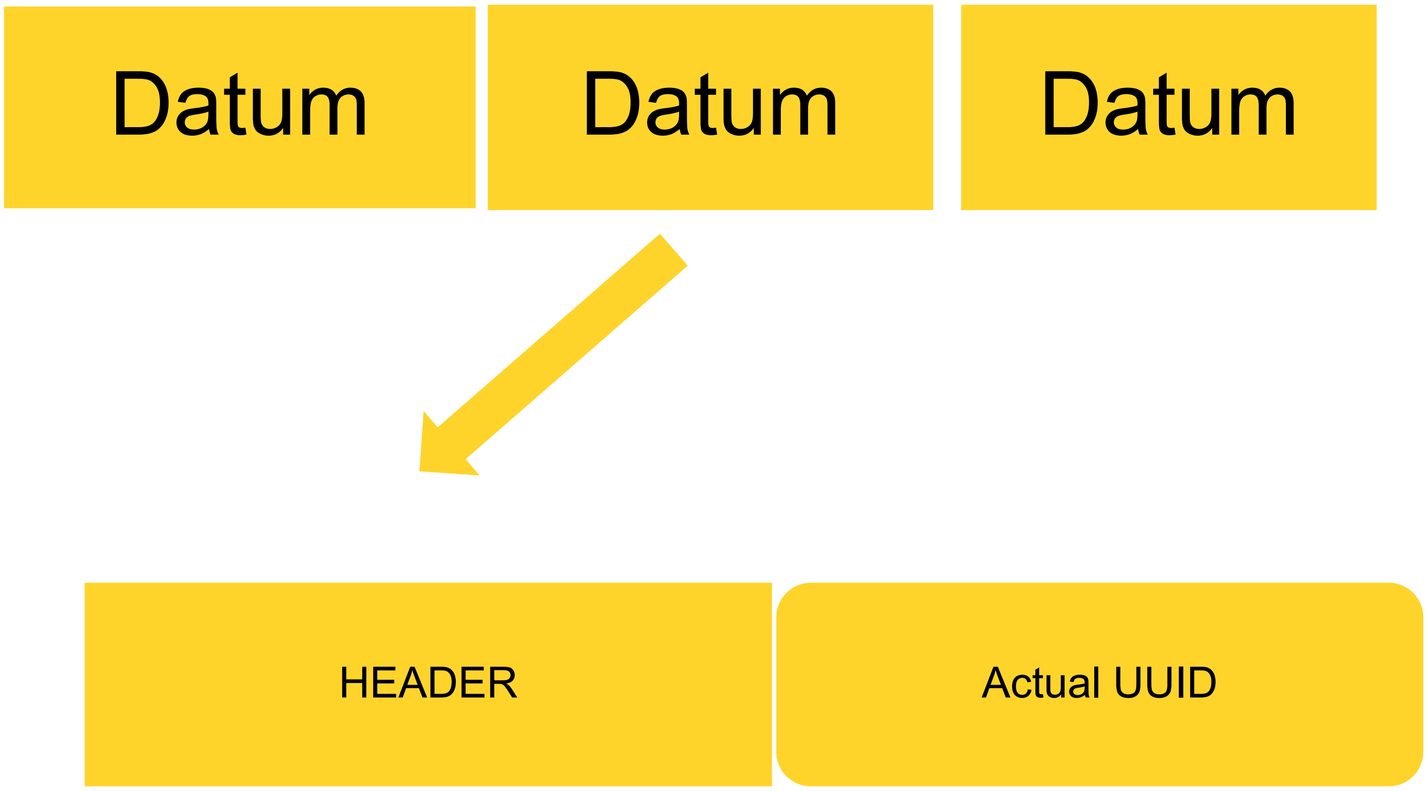


Figure 8: UUIDv4 Field and Bit Layout

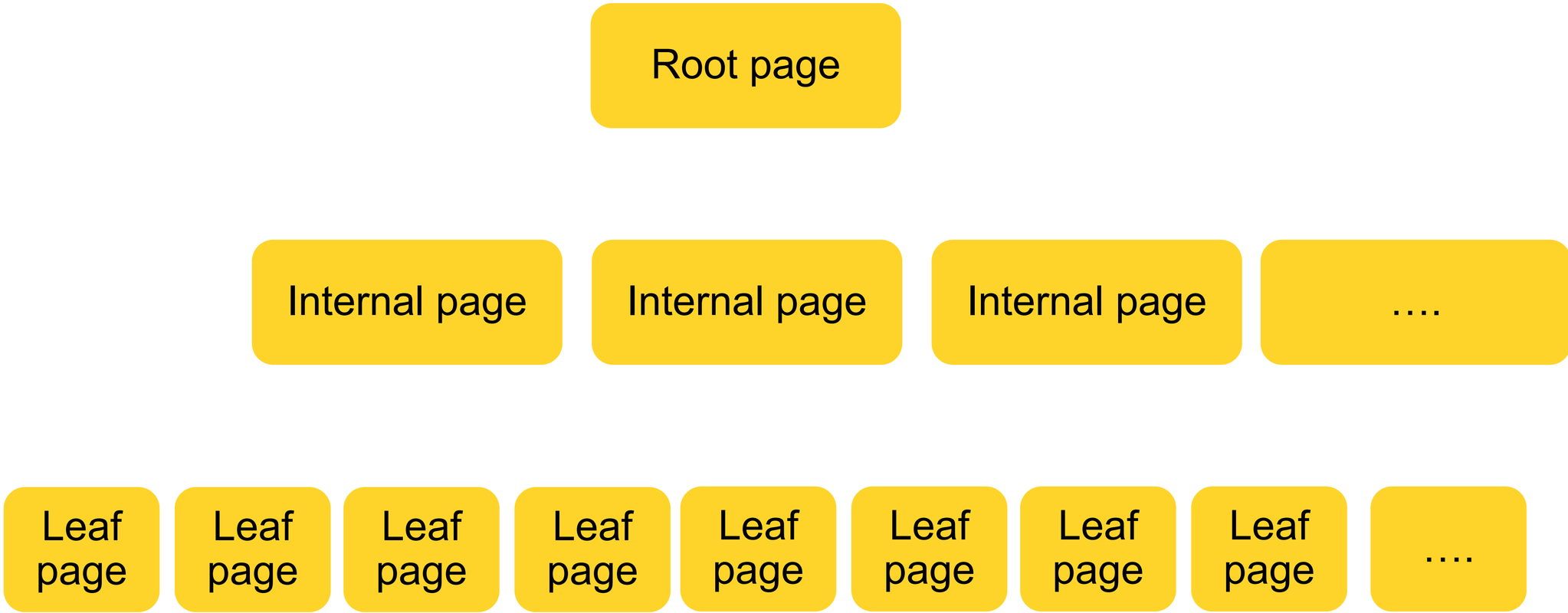
https://www.rfc-editor.org/rfc/rfc9562.txt

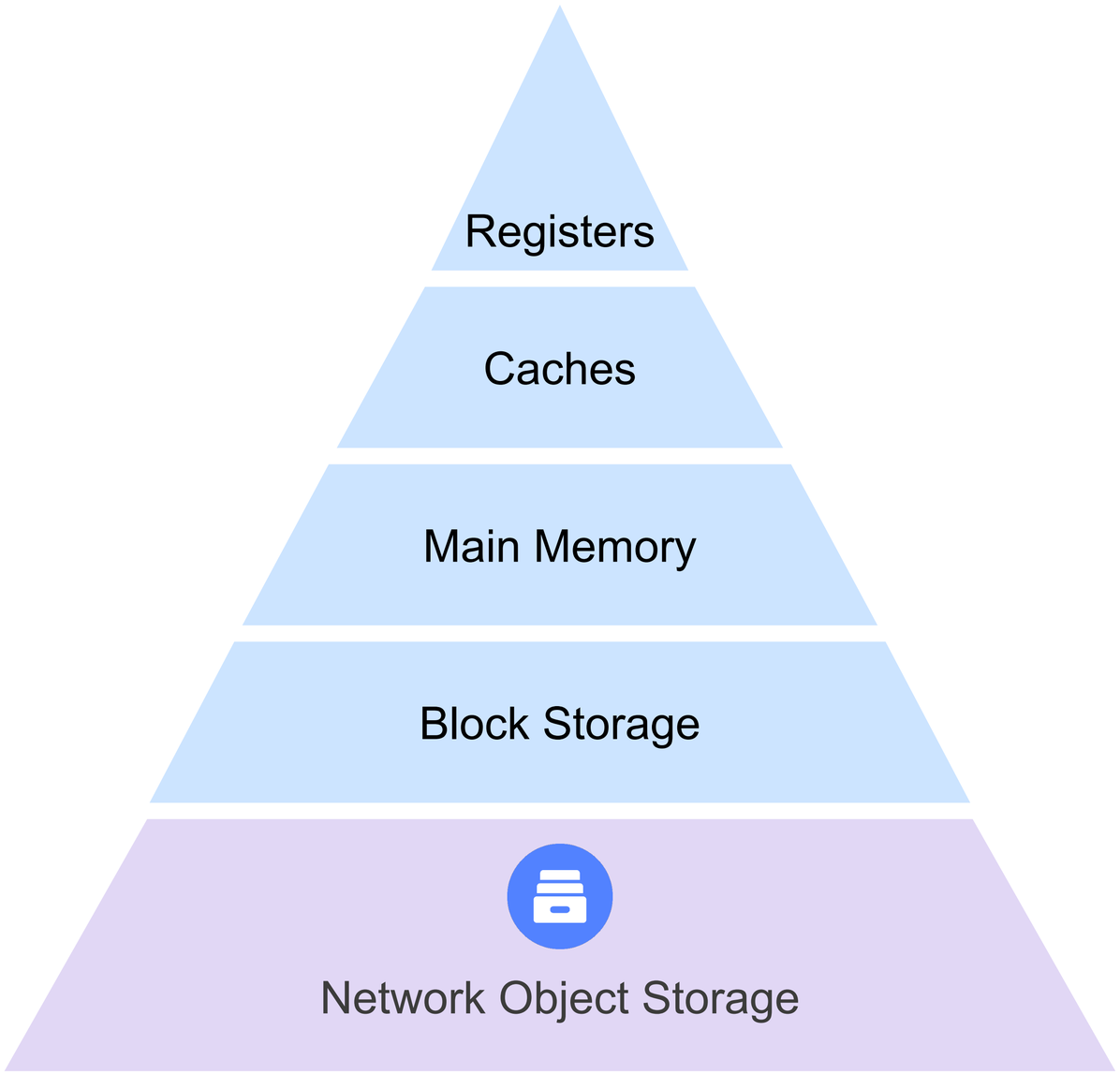
N	Вероятность хотя бы одной коллизии (p)									
	10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}	0,1 %	1 %	25 %	50 %	75 %
32	2	2	2	2,9	93	$2{,}9 \times 10^{3}$	$9{,}3 \times 10^{3}$	$5{,}0 \times 10^{4}$	$7{,}7 \times 10^{4}$	$1{,}1 \times 10^{5}$
64	6,1	$1{,}9 \times 10^{2}$	$6{,}1 \times 10^{3}$	$1{,}9 \times 10^{5}$	$6{,}1 \times 10^{6}$	$1{,}9 \times 10^{8}$	$6{,}1 \times 10^{8}$	$3{,}3 \times 10^{9}$	$5{,}1 \times 10^{9}$	$7{,}2 \times 10^{9}$
128	$2{,}6 \times 10^{10}$	$8{,}2 \times 10^{11}$	$2{,}6 \times 10^{13}$	$8{,}2 \times 10^{14}$	$2{,}6 \times 10^{16}$	$8{,}3 \times 10^{17}$	$2{,}6 \times 10^{18}$	$1{,}4 \times 10^{19}$	$2{,}2 \times 10^{19}$	$3{,}1 \times 10^{19}$
256	$4{,}8 \times 10^{29}$	$1{,}5 \times 10^{31}$	$4{,}8 \times 10^{32}$	$1{,}5 \times 10^{34}$	$4{,}8 \times 10^{35}$	$1{,}5 \times 10^{37}$	$4{,}8 \times 10^{37}$	$2{,}6 \times 10^{38}$	$4{,}0 \times 10^{38}$	$5{,}7 \times 10^{38}$
384	$8{,}9 \times 10^{48}$	$2{,}8 \times 10^{50}$	$8{,}9 \times 10^{51}$	$2{,}8 \times 10^{53}$	$8{,}9 \times 10^{54}$	$2{,}8 \times 10^{56}$	$8{,}9 \times 10^{56}$	$4{,}8 \times 10^{57}$	$7{,}4 \times 10^{57}$	$1{,}0 \times 10^{58}$
512	$1{,}6 \times 10^{68}$	$5{,}2 \times 10^{69}$	$1{,}6 \times 10^{71}$	$5{,}2 \times 10^{72}$	$1{,}6 \times 10^{74}$	$5{,}2 \times 10^{75}$	$1{,}6 \times 10^{76}$	$8{,}8 \times 10^{76}$	$1{,}4 \times 10^{77}$	$1{,}9 \times 10^{77}$

64-bit Datum[]



Вставка в B-дерево




Registers
Caches
Main Memory
Block Storage
Network Object Storage

Локальность

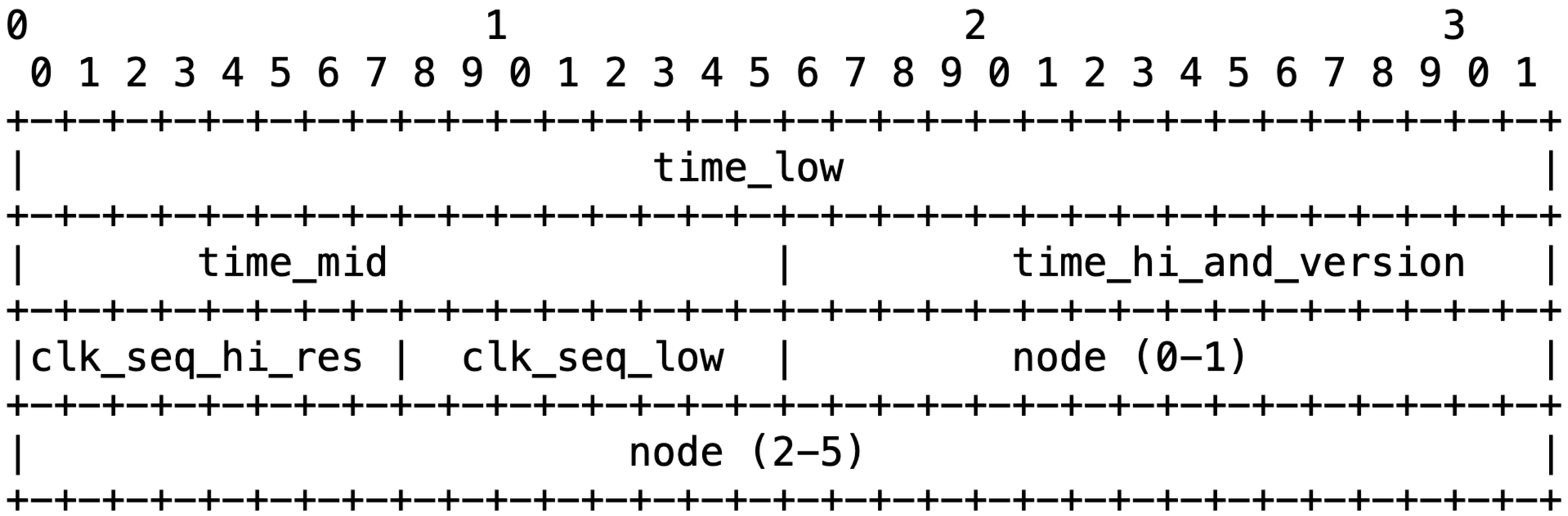
- Поля в структуре
- Datum'ы в кэш-линии
- Tuple'ы на странице

Используются обычно совместно

Монотонность

- Удобна базам данных
- Её не бывает в распределённых системах

UUID version 1



```
postgres=# create table t(i uuid, ix serial);

postgres=# insert into t values (uuid_generate_v1());
postgres=# insert into t values (uuid_generate_v1());
postgres=# insert into t values (uuid_generate_v1());
...


postgres=# select * from t order by i;
```

```
postgres=# select * from t order by i;
                  i                   | ix
--------------------------------------+----
 033d2acc-ef31-11ec-83aa-ee7894e15da8 |  8
 0e9ce376-ef31-11ec-83aa-ee7894e15da8 |  9
 106d73aa-ef31-11ec-83aa-ee7894e15da8 | 10
 aefe9540-ef30-11ec-83aa-ee7894e15da8 |  1
 afe594ae-ef30-11ec-83aa-ee7894e15da8 |  2
 b0351538-ef30-11ec-83aa-ee7894e15da8 |  3
 b0b3628a-ef30-11ec-83aa-ee7894e15da8 |  4
 b1171014-ef30-11ec-83aa-ee7894e15da8 |  5
 b15a19fe-ef30-11ec-83aa-ee7894e15da8 |  6
 c470db2c-ef30-11ec-83aa-ee7894e15da8 |  7
(10 rows)
```

Hacking hack

```
create extension uuid_v1_ops
```

This will add several operators and a `btree` operator class to the current schema in the `search_path`. The operator class is called `uuid_v1_ops` and it can be used to create an index on UUID columns like so:

```
create table v1(u uuid);
create index on v1(u uuid_v1_ops);
```

https://github.com/pgnickb/uuid_v1_ops

Hacking hack

```
static unsigned int order[UUID_LEN] = {6, 7, 4, 5, 0, 1, 2, 3, 8, 9, 10, 11, 12, 13, 14, 15};

static int
uuid_v1_internal_cmp(const pg_uuid_t *arg1, const pg_uuid_t *arg2)
{
        int                     res;

        for (int i = 0; i < UUID_LEN; ++i)
        {
                res = arg1->data[order[i]] - arg2->data[order[i]];
                if (res != 0)
                        break;
        }

        return res;
}
```

https://github.com/pgnickb/uuid_v1_ops

Альтернативы

1. [ULID] by A. Feerasta
2. [LexicalUUID] by Twitter
3. [Snowflake] by Twitter
4. [Flake] by Boundary
5. [ShardingID] by Instagram
6. [KSUID] by Segment
7. [Elasticflake] by P. Pearcy
8. [FlakeID] by T. Pawlak
9. [Sonyflake] by Sony
10. [orderedUuid] by IT. Cabrera

RFC 9562: UUID v7

```
postgres=# select uuidv7();
                uuidv7
--------------------------------------
 0194a78b-06da-797c-a58f-c811f9205a25
(1 row)
```

Сергей Прохоренко

```
  -- version
  SELECT uuid_extract_version('11111111-1111-5111-8111-111111111111');  -- 5
  SELECT uuid_extract_version(gen_random_uuid());  -- 4
  SELECT uuid_extract_version('11111111-1111-1111-1111-111111111111');  -- null
+ SELECT uuid_extract_version(uuidv4()); --4
+ SELECT uuid_extract_version(uuidv7()); --7
```

```
  -- timestamp
- SELECT uuid_extract_timestamp('C232AB00-9414-11EC-B3C8-9F6BDECED846') = 'Tuesday, February 22, 2022 2:22:22.00 PM GMT+05:00';
+ SELECT uuid_extract_timestamp('C232AB00-9414-11EC-B3C8-9F6BDECED846') = 'Tuesday, February 22, 2022 2:22:22.00 PM GMT+05:00';
+ SELECT uuid_extract_timestamp('1EC9414C-232A-6B00-B3C8-9F6BDECED846') = 'Tuesday, February 22, 2022 2:22:22.00 PM GMT+05:00';
+ SELECT uuid_extract_timestamp('017F22E2-79B0-7CC3-98C4-DC0C0C07398F') = 'Tuesday, February 22, 2022 2:22:22.00 PM GMT+05:00';
```

Бенчмарк

```
postgres=# create table table_for_uuidv4(id uuid primary key);

CREATE TABLE Time: 9.479 ms

postgres=# insert into table_for_uuidv4 select uuidv4() from generate_series(1,3e7);
INSERT 0 30000000 Time: 2003918.770 ms (33:23.919)

postgres=# create table table_for_uuidv7(id uuid primary key);

CREATE TABLE Time: 3.930 ms

postgres=# insert into table_for_uuidv7 select uuidv7() from generate_series(1,3e7);
INSERT 0 30000000 Time: 337001.315 ms (05:37.001)
```

Счетчик

```
440     static uint32_t sequence_counter;
441     static uint64_t previous_timestamp = 0;
```

Счетчик

```
510     Datum
511     uuidv7(PG_FUNCTION_ARGS)
512     {
513         pg_uuid_t  *uuid = palloc(UUID_LEN);
514         uint64_t    tms;
515         struct timeval tp;
516         bool        increment_counter;
517
518         gettimeofday(&tp, NULL);
519         tms = ((uint64_t) tp.tv_sec) * 1000 + (tp.tv_usec) / 1000;
520         /* time from clock is protected from backward leaps */
521         increment_counter = (tms <= previous_timestamp);
522
523         if (increment_counter)
524         {
525             /*
526              * Time did not advance from the previous generation, we must
527              * increment counter
528              */
529             ++sequence_counter;
530             if (sequence_counter > 0x3ffff)
531             {
532                 /* We only have 18-bit counter */
533                 sequence_counter = 0;
534                 previous_timestamp++;
535             }
536
537             /* protection from leap backward */
538             tms = previous_timestamp;
```

Bits

```
546             /* most significant 4 bits of 18-bit counter */
547             uuid->data[6] = (unsigned char) (sequence_counter >> 14);
548             /* next 8 bits */
549             uuid->data[7] = (unsigned char) (sequence_counter >> 6);
550             /* least significant 6 bits */
551             uuid->data[8] = (unsigned char) (sequence_counter);
```

Microseconds

```
Replace Leftmost Random Bits with Increased Clock Precision
(Method 3):
   For UUIDv7, which has millisecond timestamp precision, it is
   possible to use additional clock precision available on the system
   to substitute for up to 12 random bits immediately following the
   timestamp.  This can provide values that are time ordered with
   sub-millisecond precision, using however many bits are appropriate
   in the implementation environment.  With this method, the
   additional time precision bits MUST follow the timestamp as the
   next available bit in the rand_a field for UUIDv7.
```

```
x4mmm@night bin % ./pg_test_timing
Testing timing overhead for 3 seconds.
Per loop time including overhead: 19.51 ns
Histogram of timing durations:
  < us   % of total      count
     1     98.05201  150789646
     2      1.94748    2994936
     4      0.00009        135
     8      0.00026        399
    16      0.00016        243
    32      0.00001          8
    64      0.00000          3
```

Паттерны использования UUID v7

Партиционирование

- По uuid_extract_timestamp()

UUID необязательно генерировать в базе

Большинство ЯП уже поддерживают генерацию

- С разными уровнем следования RFC

> Слишком много чего угодно – это плохо. Однако слишком много виски – это едва достаточно.

Марк Твен

Когда локальности слишком много

```
SELECT uuidv7(now()+’10 years’);
SELECT uuidv7(now()+’20 years’);
SELECT uuidv7(now()+’30 years’);
SELECT uuidv7(now()+’40 years’);
```

UUIDv7 содержит время, в которое он был сгенерирован

Но это время можно скрыть

UUID v8

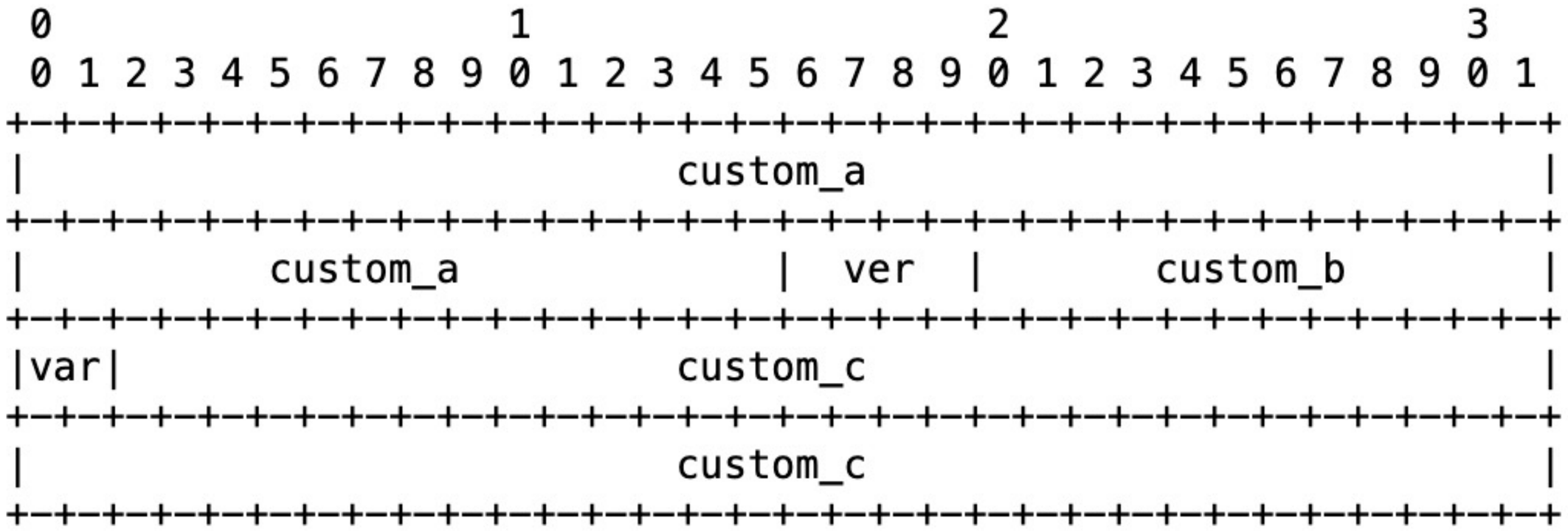


Figure 12: UUIDv8 Field and Bit Layout

https://www.rfc-editor.org/rfc/rfc9562#name-uuid-version-8

Что там в .NET?

[API Proposal]: Uuid data type #86084

Closed as not planned

vanbukin opened on May 11, 2023 · edited by vanbukin Edits Contributor •••

Background and motivation

There are 2 ways to organize the binary representation of Uuid:

1. as 16 separate octets
2. mixed-endian, which .NET inherited from COM/OLE.

Currently, the .NET Base Class Library only includes System.Guid. This is a data structure that implements the second method of binary representation.

If you use `System.Guid` as a unique identifier for objects, it is necessary to be extremely careful.
For example, if it is used as a parameter in a database query. Due to the implementation specifics of `System.Guid`, calling the method `System.Guid::ToByteArray()` and calling `System.Guid::ToString()` with subsequent conversion of the resulting hexadecimal string to a byte array will produce **different results**.

https://github.com/dotnet/runtime/issues/86084

```
304 +         public static Guid CreateVersion7(DateTimeOffset timestamp)
305 +         {
306 +             // This isn't the most optimal way, but we don't have an easy way to get
307 +             // secure random bytes in corelib without doing this since the secure rng
308 +             // is in a different layer.
309 +             Guid result = NewGuid();
```

```
317 +             long unix_ts_ms = timestamp.ToUnixTimeMilliseconds();
318 +             ArgumentOutOfRangeException.ThrowIfNegative(unix_ts_ms, nameof(timestamp));
319 +
320 +             Unsafe.AsRef(in result._a) = (int)(unix_ts_ms >> 16);
321 +             Unsafe.AsRef(in result._b) = (short)(unix_ts_ms);
322 +
323 +             Unsafe.AsRef(in result._c) = (short)((result._c & ~VersionMask) | Version7Value);
324 +             Unsafe.AsRef(in result._d) = (byte)((result._d & ~Variant10xxMask) | Variant10xxValue);
325 +
326 +             return result;
327 +         }
```

https://github.com/dotnet/runtime/pull/104124/files

Guid.CreateVersion7() - bug? #111503

Closed

sdrapkin opened on Jan 16

Description

`Guid.CreateVersion7()` explicitly states that it "Creates a new Guid according to RFC 9562, following the Version 7 format." (RFC link).

The RFC states that the v7-UUID uses a Unix millisecond-timestamp in the most significant 48 bits (i.e. first 6 bytes in network-order).
Quoting from the RFC:

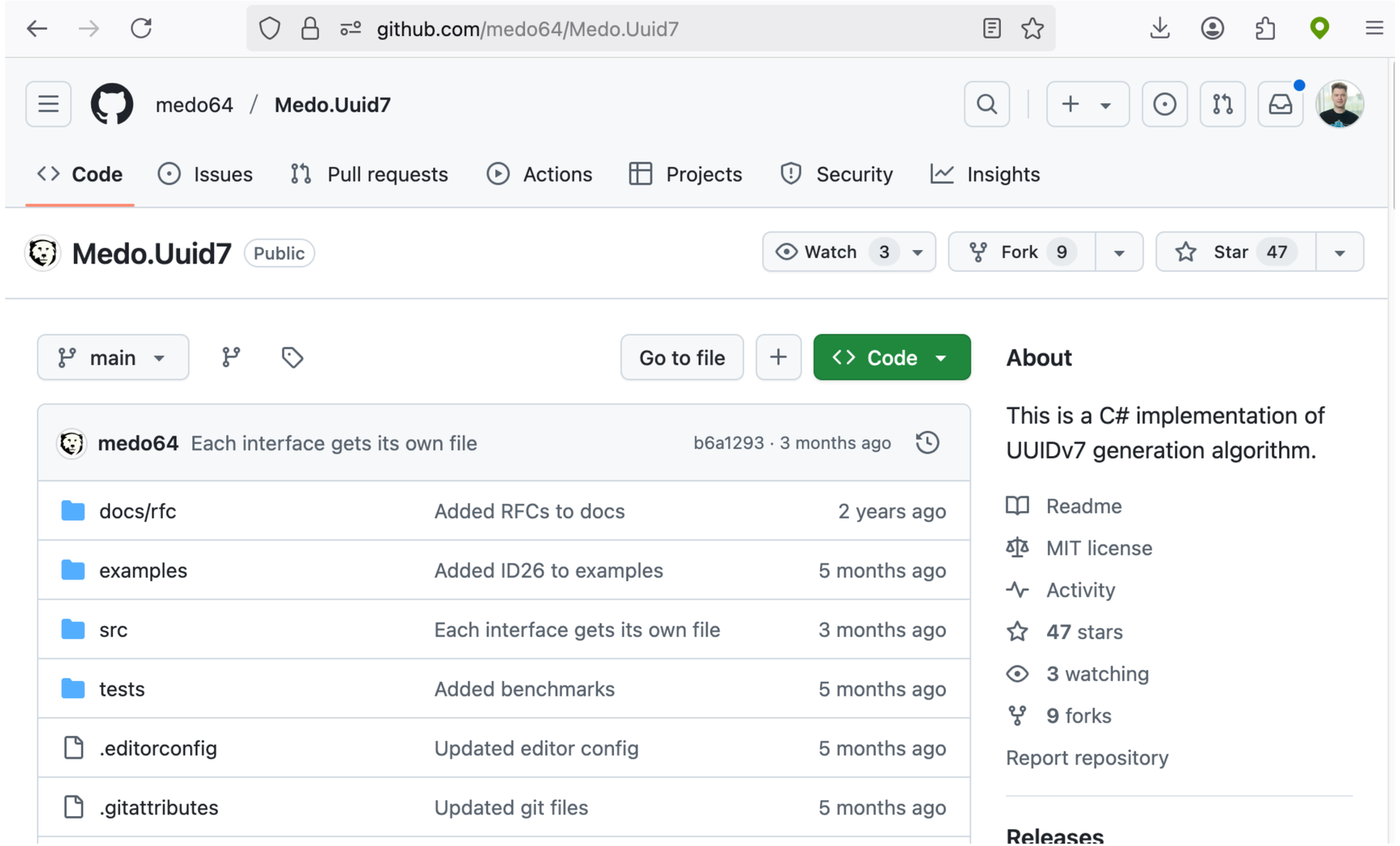
> unix_ts_ms:
> 48-bit big-endian unsigned number of the Unix Epoch timestamp in milliseconds as per Section 6.1. Occupies bits 0 through 47 (octets 0-5).

```
using System;
using UUIDNext;

// Creating a database friendly UUID for PostgreSQL (version 7) or MS SQL Server (Version 8)
Guid postgreSqlUuid = Uuid.NewDatabaseFriendly(Database.PostgreSql);
Console.WriteLine($"This is a PostgreSQL friendly UUID : {postgreSqlUuid}");

Guid sqlServerUuid = Uuid.NewDatabaseFriendly(Database.SqlServer);
Console.WriteLine($"This is a SQL Server friendly UUID : {sqlServerUuid}");
```

https://github.com/mareek/UUIDNext/tree/main

```
152 ••• public static Guid CreateUuidV7(long timestamp, Span<byte> followingBytes)
153     {
154         if (followingBytes.Length > 10)
155             throw new ArgumentException($"argument {nameof(followingBytes)} should have a size of 10 bytes or less", nameof(followingBytes));
156
157         // Extra 2 bytes in front to prepend timestamp data.
158         Span<byte> buffer = stackalloc byte[18];
159
160         // write the timestamp
161         Span<byte> timestampBytes = buffer.Slice(0, 8);
162         BinaryPrimitives.TryWriteInt64BigEndian(timestampBytes, timestamp);
163
164         // write the data provided by the caller
165         Span<byte> followingUuidBytes = buffer.Slice(8, followingBytes.Length);
166         followingBytes.CopyTo(followingUuidBytes);
167
168         // fill the remaining bytes with random data
169         Span<byte> randomBytes = buffer.Slice(8 + followingBytes.Length);
170         RandomNumberGeneratorPolyfill.Fill(randomBytes);
171
172         // cut the 2 highest bytes of the timestamp to keep only 48bits (see bit layout) and create the UUID
173         Span<byte> uuidBytes = buffer.Slice(2);
174         return CreateGuidFromBigEndianBytes(uuidBytes, 7);
175     }
```

https://github.com/mareek/UUIDNext/blob/main/Src/UUIDNext/Tools/UuidToolkit.cs#L152



https://github.com/medo64/Medo.Uuid7

```
46              // Timestamp
47              bytes[0] = (byte)(msCounter >> 40);
48              bytes[1] = (byte)(msCounter >> 32);
49              bytes[2] = (byte)(msCounter >> 24);
50              bytes[3] = (byte)(msCounter >> 16);
51              bytes[4] = (byte)(msCounter >> 8);
52              bytes[5] = (byte)msCounter;
```

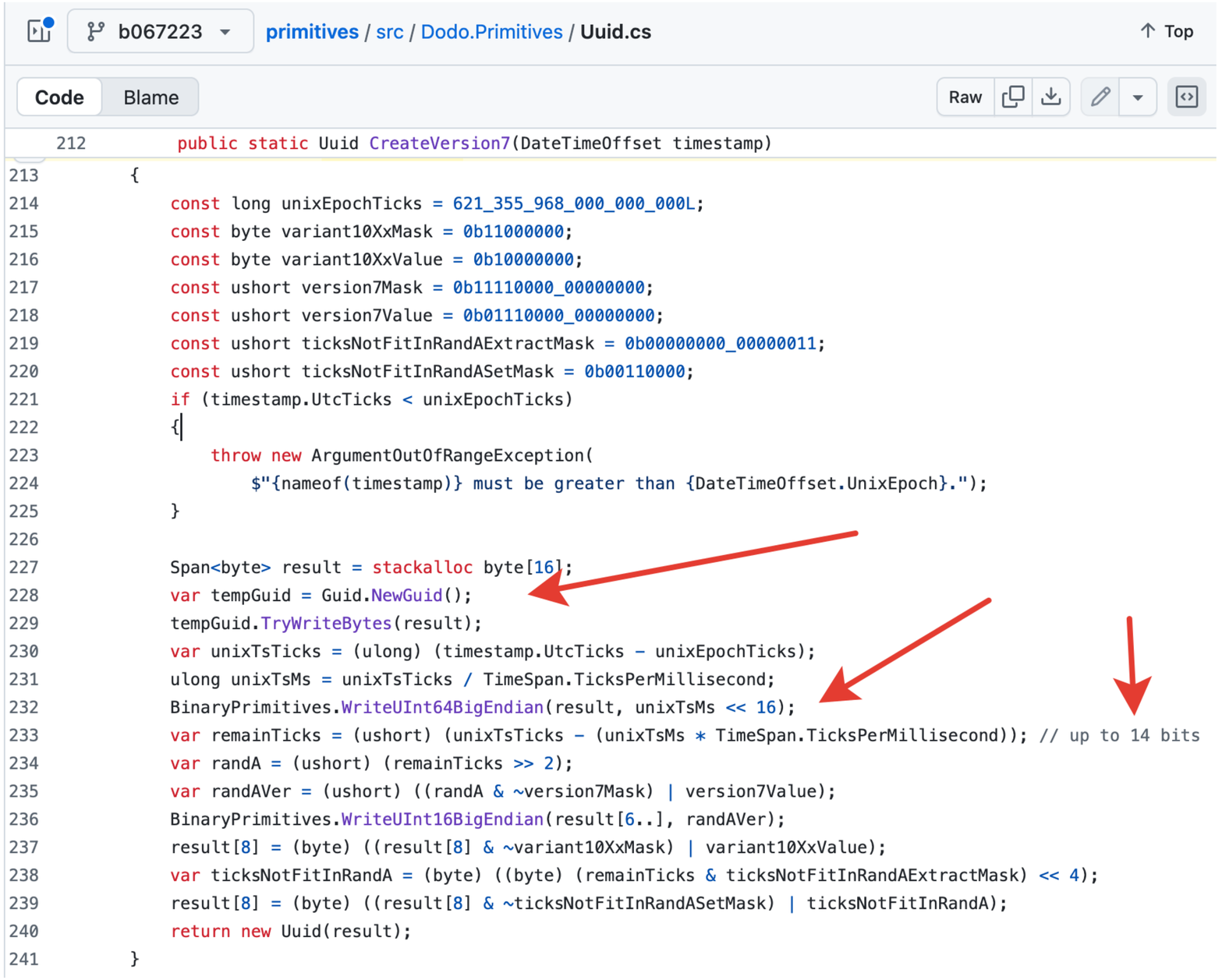
```
54              // Randomness
55              uint monoCounter;
56              if (newStep) {
57                  GetRandomBytes(ref bytes, 6, 10);
58                  monoCounter = (uint)(((bytes[6] & 0x07) <
59              } else {
60                  GetRandomBytes(ref bytes, 9, 7);
61                  monoCounter = unchecked(monotonicCounter
62                  bytes[7] = (byte)(monoCounter >> 14);
63                  bytes[9] = (byte)(monoCounter);
64              }
65              monotonicCounter = monoCounter;
```

Benchmarks

medo64 on Aug 20, 2023

I tried to follow **@bgrainger** example as close as possible. Since my code does both UUIDv7 and UUIDv4 generation, I expanded both table and code a bit.

Author	Repository or reference	Standard, version	Description	Language	Platform	Format	Write frequency, kHz	fre
Josip Medved	Medo.Uuid7	rfc4122bis-09	.NET library for generating v7 and v4 UUIDs	C#	.NET 7.0, .NET Standard 2.0	UUIDv7 (48-bit timestamp, 26-bit counter with rollover guard, 48-bit random)	13401.2	
Josip Medved	Medo.Uuid7	rfc4122bis-09	.NET library for generating v7 and v4 UUIDs	C#	.NET 7.0, .NET Standard 2.0	UUIDv4 (122-bit random)	23849.3	n/a




```
212         public static Uuid CreateVersion7(DateTimeOffset timestamp)
213         {
214             const long unixEpochTicks = 621_355_968_000_000_000L;
215             const byte variant10XxMask = 0b11000000;
216             const byte variant10XxValue = 0b10000000;
217             const ushort version7Mask = 0b11110000_00000000;
218             const ushort version7Value = 0b01110000_00000000;
219             const ushort ticksNotFitInRandAExtractMask = 0b00000000_00000011;
220             const ushort ticksNotFitInRandASetMask = 0b00110000;
221             if (timestamp.UtcTicks < unixEpochTicks)
222             {
223                 throw new ArgumentOutOfRangeException(
224                     $"{nameof(timestamp)} must be greater than {DateTimeOffset.UnixEpoch}.");
225             }
226
227             Span<byte> result = stackalloc byte[16];
228             var tempGuid = Guid.NewGuid();
229             tempGuid.TryWriteBytes(result);
230             var unixTsTicks = (ulong) (timestamp.UtcTicks - unixEpochTicks);
231             ulong unixTsMs = unixTsTicks / TimeSpan.TicksPerMillisecond;
232             BinaryPrimitives.WriteUInt64BigEndian(result, unixTsMs << 16);
233             var remainTicks = (ushort) (unixTsTicks - (unixTsMs * TimeSpan.TicksPerMillisecond)); // up to 14 bits
234             var randA = (ushort) (remainTicks >> 2);
235             var randAVer = (ushort) ((randA & ~version7Mask) | version7Value);
236             BinaryPrimitives.WriteUInt16BigEndian(result[6..], randAVer);
237             result[8] = (byte) ((result[8] & ~variant10XxMask) | variant10XxValue);
238             var ticksNotFitInRandA = (byte) ((byte) (remainTicks & ticksNotFitInRandAExtractMask) << 4);
239             result[8] = (byte) ((result[8] & ~ticksNotFitInRandASetMask) | ticksNotFitInRandA);
240             return new Uuid(result);
241         }
```

https://github.com/dodobrands/primitives/blob/main/src/Dodo.Primitives/Uuid.cs#L212

Используйте UUID v7 уже сейчас!

Буду рад ответить на вопросы ☺

Андрей Бородин

 x4mmm @yandex-team.ru

 x4mmm