

# **Сериализация: быстро, компактно, кроссплатформенно**

Владимир Озеров

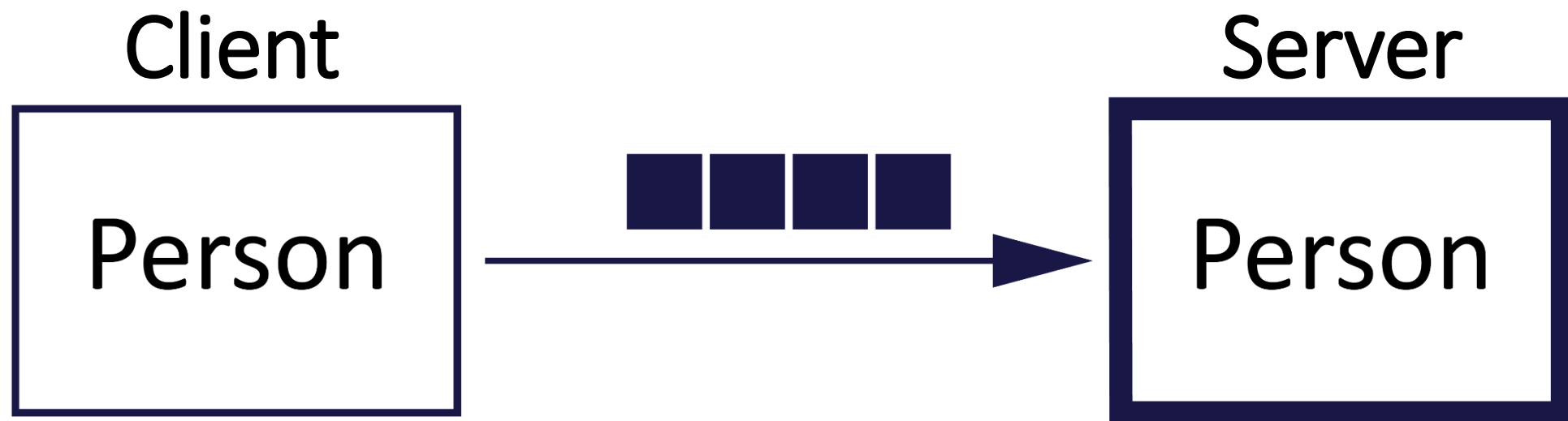
GridGain



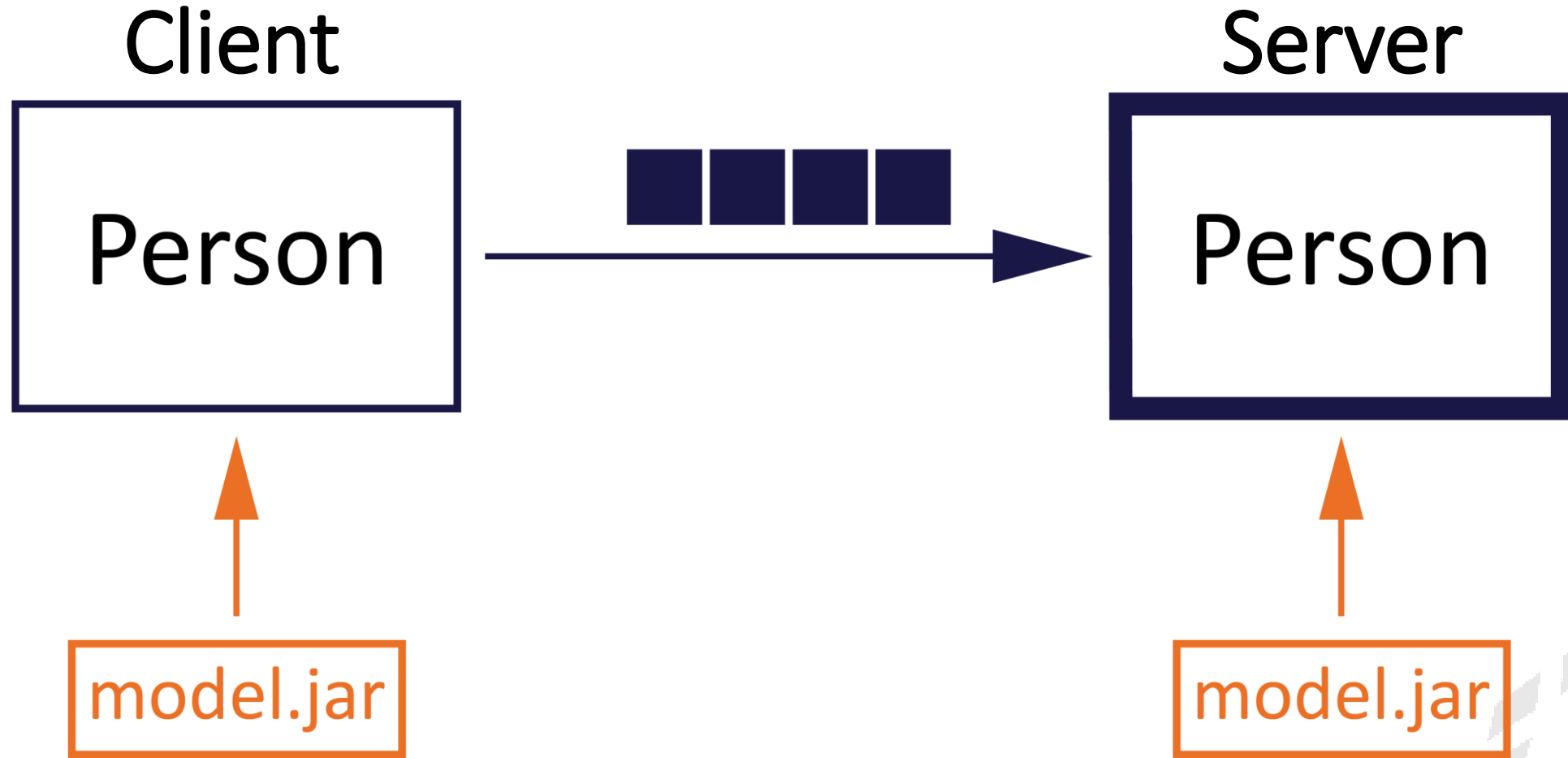
КТО?



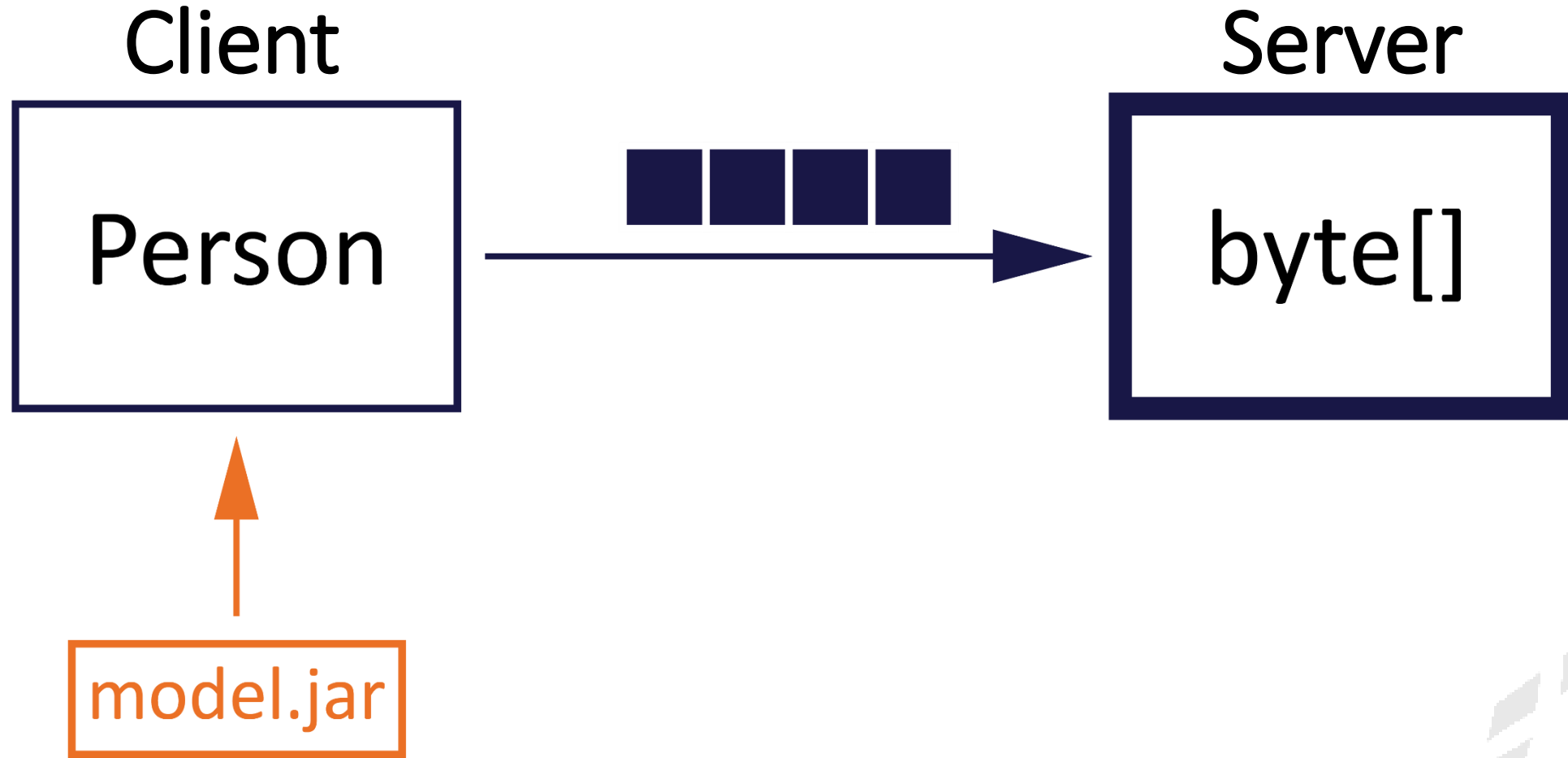
# Задача



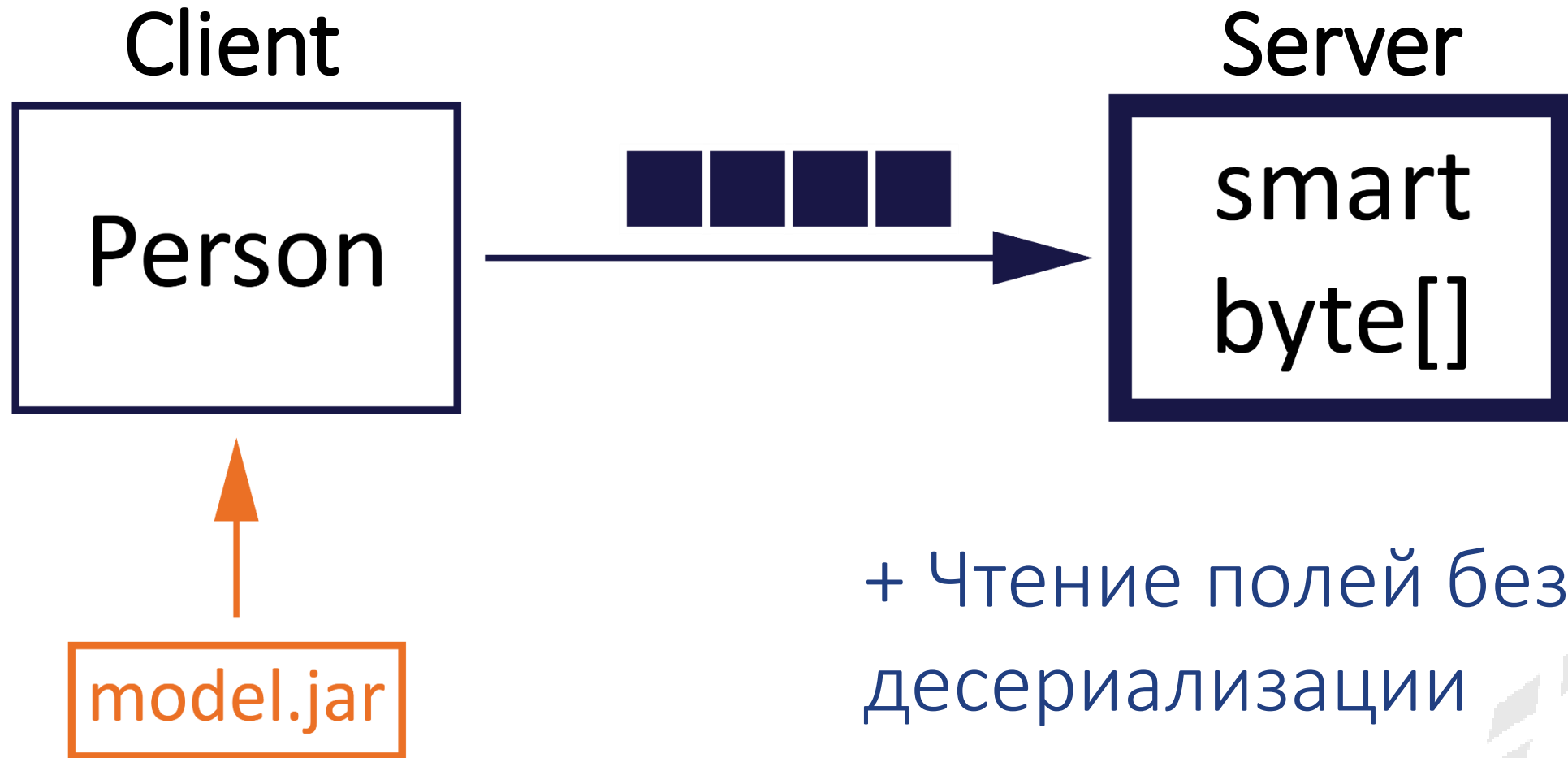
# Задача



# Задача



# Задача



# Задача

- Свой движок без внешних зависимостей
- Достаточно быстрый и компактный
- Кроссплатформенный
- Чтение полей без десериализации



# План

- Работа с типами
- Кроссплатформенность
- Метаданные
- Движок
- Чтение полей без десериализации





# План

- Работа с типами
- Кроссплатформенность
- Метаданные
- Движок
- Чтение полей без десериализации



# Джентельменский набор

- Целочисленные типы (int, long, etc.)
- Массивы
- Строки



# Джентельменский набор

- Целочисленные типы (int, long, etc.)
- Массивы
- Строки

Все остальное можно вывести!

Пишем int

```
int val = 256
```

Пишем int

```
int val = 256
```

00000000 00000000 00000001 00000000

Пишем int

```
int val = 256
```

00000000 00000000 00000001 00000000

# Varint

0000 00000000 00000000 0000010 00000000



# Varint

0000 00000000 00000000 0000010 00000000

00000000



# Varint

0000 00000000 00000000 0000010 00000000

1 | 00000000

# Varint

0000 00000000 00000000 0000010 00000000

0000010 1 | 00000000

# Varint

0000 00000000 00000000 0000010 00000000

0 | 0000010 1 | 00000000

## Varint: отрицательные значения

$< 2^7 \Rightarrow$  1 байт

$< 2^{14} \Rightarrow$  2 байта

$< 2^{21} \Rightarrow$  3 байта

$< 2^{28} \Rightarrow$  4 байта



## Varint: отрицательные значения

$< 2^7 \Rightarrow$  1 байт

$< 2^{14} \Rightarrow$  2 байта

$< 2^{21} \Rightarrow$  3 байта

$< 2^{28} \Rightarrow$  4 байта

$\geq 2^{28} \Rightarrow$  5 байт!



Varint: большие значения

```
UUID.randomUUID()
```



## Varint: большие значения

```
UUID.randomUUID()
```

```
most: 4733520965880989459
```

```
least: -9172651242694326537
```



## Varint: большие значения

```
UUID.randomUUID()
```

```
most: 4733520965880989459
```

```
least: -9172651242694326537
```

20 байт, вместо 16





## Varint: отрицательные значения

-1 => 0xFFFFFFFFFF => 5 байт

## Varint: отрицательные значения

$-1 \Rightarrow 0xFFFFFFFF \Rightarrow 5 \text{ байт}$

ZigZag encoding:

$$x = (x \ll 1) \wedge (x \gg 31)$$



## Varint: отрицательные значения

$-1 \Rightarrow 0xFFFFFFFF \Rightarrow 5 \text{ байт}$

ZigZag encoding:

$x = (x \ll 1) \wedge (x \gg 31)$

$-1 \Rightarrow 1 \Rightarrow 0x00000001 \Rightarrow 1 \text{ байт}$

# Varint в Apache Ignite

- Не используем для пользовательских данных по умолчанию
- Используем для малых значений (напр. длина строки)



## Запись других типов

- `float`       $\rightarrow$  `int`
- `double`     $\rightarrow$  `long`
- `UUID`        $\rightarrow$  `long + long`
- `Date`         $\rightarrow$  `long`



# Массивы

Формат: длина + содержимое

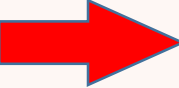


# Массивы

Формат: длина + содержимое

```
1: public void write(int[] arr, DataOutput out) {  
2:     out.writeInt(arr.length);  
3:  
4:     for (int i = 0; i < arr.length; i++)  
5:         out.writeInt(arr[i]);  
6: }
```

# Массивы: пишем быстро

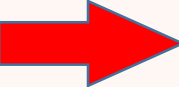
```
1: public void write(int[] arr, BinaryOutput out) {  
2:  out.writeInt(arr.length);  
3:  
4:     out.ensureCapacity(arr.length * 4);  
5:  
6:     UNSAFE.copyMemory(  
7:         arr, [SRC_OFFSET],  
8:         out.array(), [DEST_OFFSET],  
9:         arr.length * 4  
10:    );  
11:  
12:     out.shift(arr.length * 4);  
13: }
```



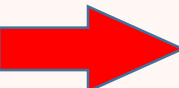
# Массивы: пишем быстро

```
1: public void write(int[] arr, BinaryOutput out) {
2:     out.writeInt(arr.length);
3:
4:     → out.ensureCapacity(arr.length * 4);
5:
6:     UNSAFE.copyMemory(
7:         arr, [SRC_OFFSET],
8:         out.array(), [DEST_OFFSET],
9:         arr.length * 4
10:    );
11:
12:    out.shift(arr.length * 4);
13: }
```

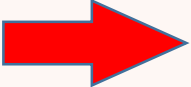
# Массивы: пишем быстро

```
1: public void write(int[] arr, BinaryOutput out) {
2:     out.writeInt(arr.length);
3:
4:     out.ensureCapacity(arr.length * 4);
5:
6:      UNSAFE.copyMemory(
7:         arr, [SRC_OFFSET],
8:         out.array(), [DEST_OFFSET],
9:         arr.length * 4
10:    );
11:
12:    out.shift(arr.length * 4);
13: }
```

# Массивы: пишем быстро

```
1: public void write(int[] arr, BinaryOutput out) {
2:     out.writeInt(arr.length);
3:
4:     out.ensureCapacity(arr.length * 4);
5:
6:     UNSAFE.copyMemory(
7:         arr, [SRC_OFFSET],
8:         out.array(), [DEST_OFFSET],
9:         arr.length * 4
10:    );
11:
12:  out.shift(arr.length * 4);
13: }
```

# Массивы: пишем быстро

```
1: public void write(int[] arr, BinaryOutput out) {
2:     out.writeInt(arr.length);
3:
4:     out.ensureCapacity(arr.length * 4);
5:
6:     UNSAFE.copyMemory(
7:         arr, [SRC_OFFSET],
8:          out.array(), [DEST_OFFSET],
9:         arr.length * 4
10:    );
11:
12:    out.shift(arr.length * 4);
13: }
```

# Строки

```
String s = "Hello"
```

# Строки

```
String s = "Hello"
```

**UTF-16:** 00 48 00 65 00 6c 00 6c 00 6f

# Строки

```
String s = "Hello"
```

**UTF-16:** 00 48 00 65 00 6c 00 6c 00 6f

# Строки

```
String s = "Hello"
```

**UTF-16:** 00 48 00 65 00 6c 00 6c 00 6f

**UTF-8:** 48 65 6c 6c 6f



# Строки

```
String s = "Привет"
```

UTF-8 : 12 байт

Ср1251: 6 байт

# Строки в Apache Ignite

- Пишем в UTF-8 по умолчанию
- Наращиваем другие кодировки
- Формат: длина (int) + массив



# План

- Работа с типами
- Кроссплатформенность
- Метаданные
- Движок
- Чтение полей без десериализации



# Источники проблем

- Железо
- Софт



# Железо: endianness

$x = 1$

Big endian:        00   00   00   **01**

Little endian:    **01**   00   00   00

# Endianness: пишем в little-endian

```
1: public void write_little(int val, BinaryOutput out) {  
2:     out.writeInt(val);  
3: }  
4:  
5:  
6:  
7:  
8:  
9:
```

# Endianness: пишем в little-endian

```
1: public void write_little(int val, BinaryOutput out) {  
2:     out.writeInt(val);  
3: }  
4:  
5: public void write_big(int val, BinaryOutput out) {  
6:     val = Integer.reverseBytes(val);  
7:  
8:     out.writeByte(val);  
9: }
```

## Если не угадали

- Дороже запись примитивов\*
- Дороже запись массивов\*

\* Кроме byte и byte[]





# Endianness в Apache Ignite

- Пишем всегда в little-endian, так как она встречается чаще
- Если потребуется – дадим возможность выбирать



# Железо: unaligned memory access

**IGNITE-1493:** Fatal exception is thrown during queue instantiation when using Ignite with an **HP-UX** machine

```
# Problematic frame:
```

```
# V [libjvm.so+0xc7c438] Unsafe_SetInt+0x14c
```

```
...
```

```
siginfo:si_signo=SIGBUS: si_errno=0, si_code=1  
(BUS_ADRLN), si_addr=0xfffffffffaf0e1a1a9
```

# Alignment в Apache Ignite

- Пишем словами только на x86
- Побайтовая запись на остальных платформах
- Слишком **пессимистично**, можно регулировать руками



# Софт: источники проблем

- МАРПИНГ ТИПОВ
- ФОРМАТ ТИПОВ



# Unsigned типы

Java

.NET

byte



sbyte

byte

short



short

ushort

int



int

uint

long



long

ulong

# Unsigned типы

Java

.NET

byte



sbyte

byte

short



short

ushort

int



int

uint

long



long

ulong

# Unsigned типы: решения

1) Не поддерживать unsigned (Thrift)



# Unsigned типы: решения

1) Не поддерживать unsigned (Thrift)

2) Мапить на signed тип (Ignite, Protobuf)

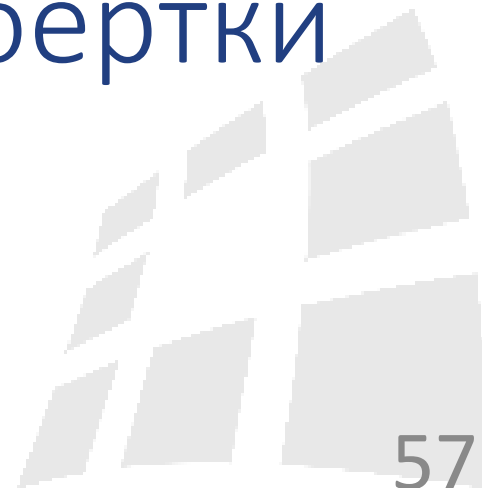
- `ushort -> short`
- `uint -> int`





## Unsigned типы: решения

- 1) Не поддерживать unsigned (Thrift)
- 2) Мапить на signed тип (Ignite, Protobuf)
  - `ushort -> short`
  - `uint -> int`
- 3) Создать дополнительные типы-обертки



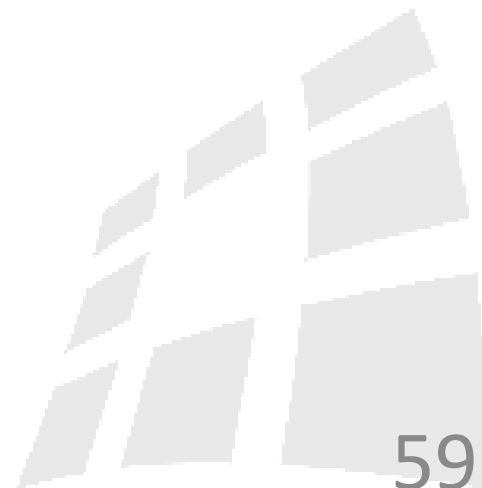
# Даты

`java.util.Date` – абсолютное время

# Даты

`java.util.Date` – **абсолютное** время

`System.DateTime` – время + **тип** (`DateTimeKind`)



# Даты

`java.util.Date` – **абсолютное** время

`System.DateTime` – время + **тип** (`DateTimeKind`)

- Абсолютное (`Utc`)

# Даты

`java.util.Date` – **абсолютное** время

`System.DateTime` – время + **тип** (`DateTimeKind`)

- Абсолютное (`Utc`)
- Локальное (`Local`)



# Даты

`java.util.Date` – **абсолютное** время

`System.DateTime` – время + **тип** (`DateTimeKind`)

- Абсолютное (`Utc`)
- Локальное (`Local`)
- **X3 какое** (`Unspecified*`)

\* **MSDN**: “The time represented is not specified as either local time or Coordinated Universal Time (UTC).”

# UUID

```
1: public void writeUUID(DataOutput out, UUID val) {  
2:     out.writeLong(val.getMostSignificantBits());  
3:     out.writeLong(val.getLeastSignificantBits());  
4: }
```

# UUID

## Java

A1 A2 B1 B2 C1 C2 C3 C4 | D1 D2 D3 D4 D5 D6 D7 D8





UUID

Java

A1 A2 B1 B2 C1 C2 C3 C4 | D1 D2 D3 D4 D5 D6 D7 D8

C1 C2 C3 C4 B1 B2 A1 A2 | D8 D7 D6 D5 D4 D3 D2 D1

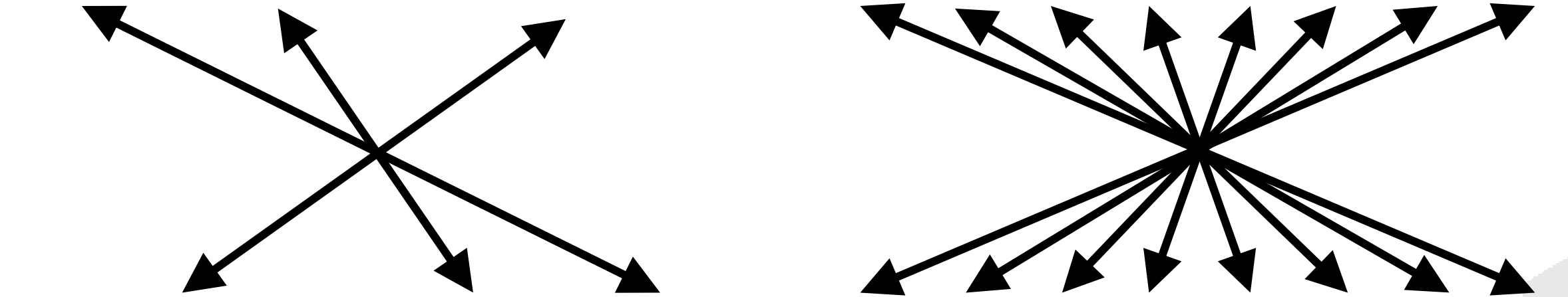
.NET



UUID

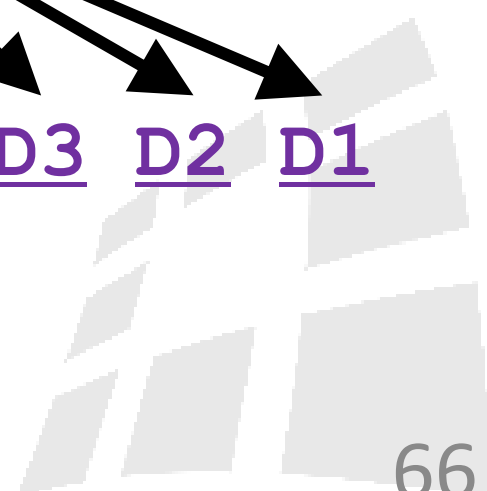
Java

A1 A2 B1 B2 C1 C2 C3 C4 | D1 D2 D3 D4 D5 D6 D7 D8



C1 C2 C3 C4 B1 B2 A1 A2 | D8 D7 D6 D5 D4 D3 D2 D1

.NET



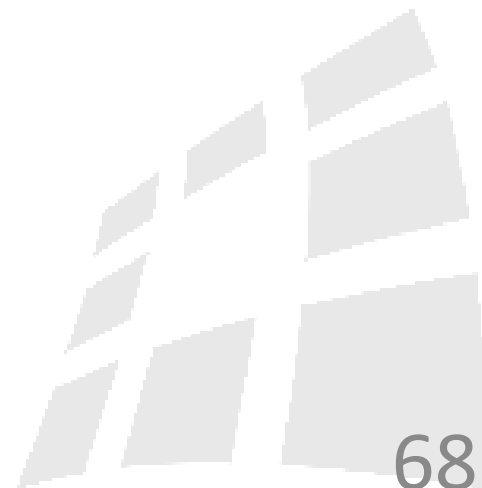
# UUID: решения

- Выбрать один формат, но какая-то платформа будет в проигрыше
- Или просто поддерживать несколько форматов!



# План

- Работа с типами
- Кроссплатформенность
- **Метаданные**
- Движок
- Чтение полей без десериализации



# Сериализуем через JDK

```
1: class MyClass {  
2:     int val;  
3: }
```

# Сериализуем через JDK

```
ObjectOutputStream.writeObject(new MyClass(1));
```



# Сериализуем через JDK

```
ObjectOutputStream.writeObject(new MyClass(1));
```

74 байта!



# Сериализуем через JDK

```
ObjectOutputStream.writeObject(new MyClass(1));
```

74 байта!

sr + org.devozerov.joker2017.\_01\_javaser.MyClassTm(I valxp



# Передаем имена классов и полей

- + Не требует вмешательства пользователя
- + Работает для Java всегда
- Не компактно!
- Не кроссплатформенно!



# Введем идентификаторы

```
MyClass => 42
```

```
MyClass.val1 => 1
```

```
MyClass.val2 => 2
```

# Задаем идентификаторы руками

```
1: @TypeId(value = 42)
1: class MyClass {
2:     int val1;
3:     int val2;
4:
5:     void read(Reader reader) {
6:         val1 = reader.readInt(1);
7:         val2 = reader.readInt(2);
8:     }
9:
10:    void write(Writer write) {
11:        write.writeInt(1, val1);
12:        write.writeInt(2, val2);
13:    }
14: }
```

# Задаем идентификаторы руками

- + Компактно

- + Кроссплатформенно

- Неудобно!



# Как сгенерировать ID в кластере?

- Руками – железобетонно, но страдает user experience



# Как сгенерировать ID в кластере?

- Руками – железобетонно, но страдает user experience
- Централизованно – сложно, нужен persistence



# Как сгенерировать ID в кластере?

- Руками – железобетонно, но страдает user experience
- Централизованно – сложно, нужен persistence
- Hashing – легко, но могут быть коллизии

# Как сгенерировать ID в кластере?

- Руками – железобетонно, но страдает user experience
- Централизованно – сложно, нужен persistence
- Hashing – легко, но могут быть коллизии



# Идентификаторы в Apache Ignite

**org.package.MyClass**

✓

Class.getName()

✓

String.hashCode()

✓

**[TYPE\_ID]**



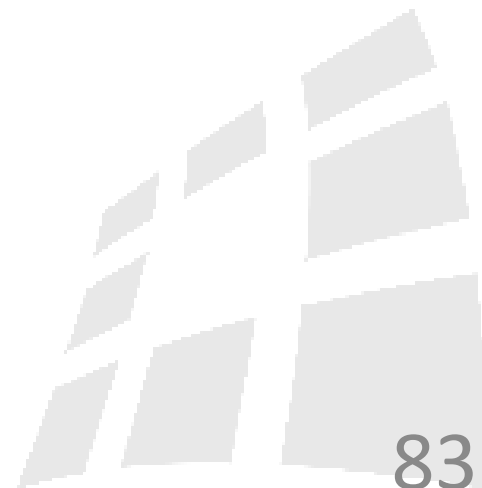
# Идентификаторы в Apache Ignite

- Используем `String.hashCode()`, буквально!
- Можно переопределить в случае коллизий
- Храним прямой и обратный mapping в кластере



# План

- Работа с типами
- Кроссплатформенность
- Метаданные
- **Движок**
- Чтение полей без десериализации



# Механизм сериализации

```
1: class Person implements Serializable {  
2:     private String name;  
3:     private int age;  
4: }
```

Как сериализовать данный класс?

# Reflection

```
1: class ClassDescriptor {  
2:     Collection<FieldDescriptor> fields;  
3:  
4:  
5:  
6:  
7:  
8:  
9:  
10:  
11:  
12:  
13:  
14:  
15: }
```

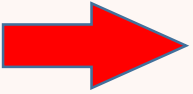
# Reflection

```
1: class ClassDescriptor {  
2:     Collection<FieldDescriptor> fields;  
3:  
4:     void write(Object obj, BinaryOutput out) {  
5:         for (FieldDescriptor field : fields) {  
6:  
7:  
8:  
9:  
10:  
11:  
12:  
13:     }  
14: }  
15: }
```

# Reflection

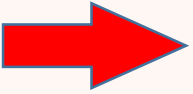
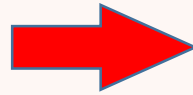
```
1: class ClassDescriptor {
2:     Collection<FieldDescriptor> fields;
3:
4:     void write(Object obj, BinaryOutput out) {
5:         for (FieldDescriptor field : fields) {
6:             switch (field.getType()) {
7:                 case INT:
8:                     int val = field.getInt(obj);
9:                     out.writeInt(val);
10:                    break;
11:                ...
12:            }
13:        }
14:    }
15: }
```

# Reflection

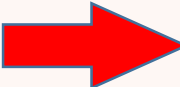
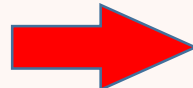
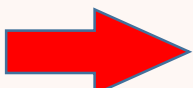
```
1: class ClassDescriptor {
2:     Collection<FieldDescriptor> fields;
3: Итерация!
4:     void write(Object obj, BinaryOutput out) {
5:          for (FieldDescriptor field : fields) {
6:             switch (field.getType()) {
7:                 case INT:
8:                     int val = field.getInt(obj);
9:                     out.writeInt(val);
10:                    break;
11:                ...
12:            }
13:        }
14:    }
15: }
```



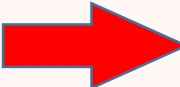
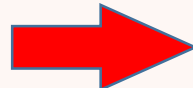
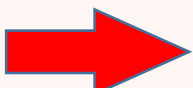
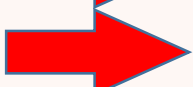
# Reflection

```
1: class ClassDescriptor {
2:     Collection<FieldDescriptor> fields;
3: Итерация!
4:     void write(Object obj, BinaryOutput out) {
5:          for (FieldDescriptor field : fields) {
6:              switch (field.getType()) {
7:                 case INT:
8:                     int val = field.getInt(obj);
9:                     Условный оператор!
10:                    out.writeInt(val);
11:                    break;
12:                ...
13:            }
14:        }
15:    }
```

# Reflection

```
1: class ClassDescriptor {
2:     Collection<FieldDescriptor> fields;
3: Итерация!
4:     void write(Object obj, BinaryOutput out) {
5:          for (FieldDescriptor field : fields) {
6:              switch (field.getType()) {
7:                 case INT:
8:                      int val = field.getInt(obj);
9:                     Условный оператор! out.writeInt(val);
10:                    break;
11:                ...
12:            }
13:        }
14:    }
15: }
```

# Reflection

```
1: class ClassDescriptor {
2:     Collection<FieldDescriptor> fields;
3: Итерация!
4:     void write(Object obj, BinaryOutput out) {
5:          for (FieldDescriptor field : fields) {
6:              switch (field.getType()) {
7:                 case INT:
8:                      int val = field.getInt(obj);
9:                      out.writeInt(val);
10:                    break;
11:                    ...
12:            }
13:        }
14:    }
15: }
```

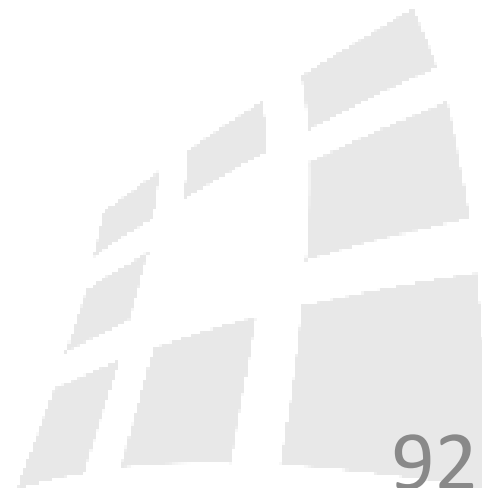
**Условный оператор!**

**Reflection!**

**Проверка границ!**

# Reflection

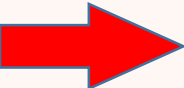
- + Никаких действий со стороны пользователя
- Не самый быстрый подход



# Сериализуем руками

```
1: class Person implements Serializable {  
2:     private String name;  
3:     private int age;  
3:  
3:     void writeBinary(BinaryWriter writer) {  
3:         writer.writeString(name);  
3:         writer.writeInt(age);  
3:     }  
4: }
```

# Сериализуем руками

```
1: class Person implements Serializable {  
2:     private String name;  
3:     private int age;  
3:  
3:     void writeBinary(BinaryWriter writer) {  
3:         writer.writeString(name);  
3:          writer.writeInt(age);  
3:     } Проверка границ!  
4: }
```

# Сериализуем руками

+ Быстрее

- Boilerplate
- Не всегда можно менять модель



# Кодогенерация

```
1: class MyClassDescriptor implements ClassDescriptor {  
2:     void write(Object obj, BinaryOutput out) {  
3:         MyClass obj0 = (MyClass)obj;  
4:  
5:         ➡ int maxLen = 4 * obj0.name.length() + 4 + 4;  
6:  
7:  
8:  
9:  
10:  
11:     }  
12: }
```

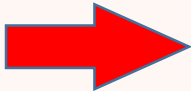


# Кодогенерация

```
1: class MyClassDescriptor implements ClassDescriptor {  
2:     void write(Object obj, BinaryOutput out) {  
3:         MyClass obj0 = (MyClass)obj;  
4:  
5:         int maxLen = 4 * obj0.name.length() + 4 + 4;  
6:  
7:         ➡ out.ensureCapacity(maxLen);  
8:  
9:  
10:  
11:     }  
12: }
```

# Кодогенерация

```
1: class MyClassDescriptor implements ClassDescriptor {  
2:     void write(Object obj, BinaryOutput out) {  
3:         MyClass obj0 = (MyClass)obj;  
4:  
5:         int maxLen = 4 * obj0.name.length() + 4 + 4;  
6:  
7:         out.ensureCapacity(maxLen);  
8:  
9:         out.writeStringUnsafe(obj0.name);  
10:        out.writeIntUnsafe(obj0.age);  
11:     }  
12: }
```



Пример: protobuf optimize\_for

Person [name, age]



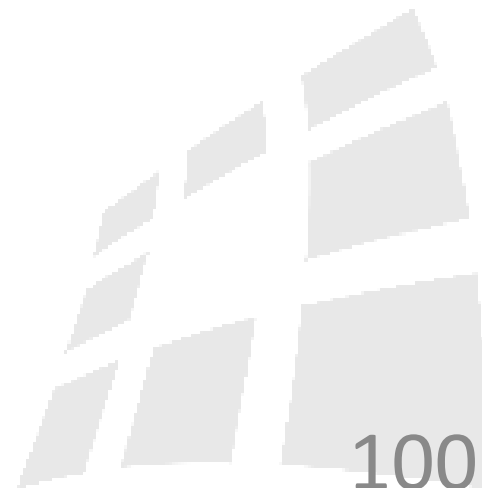
# Пример: protobuf и опция “optimize\_for”

Person [name, age]

CODE\_SIZE (reflection)

=> **2.7M** ops/sec (write)

=> **5.4M** ops/sec (read)



# Пример: protobuf и опция “optimize\_for”

Person [name, age]

CODE\_SIZE (reflection)

=> **2.7M** ops/sec (write)

=> **5.4M** ops/sec (read)

SPEED (codegen)

=> **13.5M** ops/sec (write)

=> **18.2M** ops/sec (read)



# Кодогенерация в Ignite

- **Compile time** – отказались, неудобно



# Кодогенерация в Ignite

- **Compile time** – отказались, неудобно
- **Runtime** – отказались, сложно

╱(ツ)╱



# Кодогенерация в Ignite

- **Compile time** – отказались, неудобно
- **Runtime** – отказались, ~~сложно~~ не нужно  
Текущей скорости хватает \_(ツ)\_/





# План

- Работа с типами
- Кроссплатформенность
- Метаданные
- Движок
- Чтение полей без десериализации



# Чтение полей без десериализации

## Нам нужна длина поля!



# Наивный формат

[ID | LENGTH | VAL]



# Наивный формат

[ **ID** | **LENGTH** | **VAL** ]

**ID** – идентификатор поля (4 байта)

**LENGTH** – длина поля (1-4 байта)



# Наивный формат

[**ID\_1** | **LENGTH\_1** | **VAL\_1**] [**ID\_2** | **LENGTH\_2** | **VAL\_2**]

**ID** – идентификатор поля (4 байта)

**LENGTH** – длина поля (1-4 байта)



# Наивный формат

[**ID\_1** | **LENGTH\_1** | **VAL\_1**] [**ID\_2** | **LENGTH\_2** | **VAL\_2**]

**ID** – идентификатор поля (4 байта)

**LENGTH** – длина поля (1-4 байта)

- Время поиска  $O(N)$ !
- Теряем место на ID (4 байта на поле)



# Cxema

[**SCHEMA**] [LENGTH\_1 | VAL\_1] [LENGTH\_2 | VAL\_2]

# Cxema

[**SCHEMA**] [**LENGTH\_1** | **VAL\_1**] [**LENGTH\_2** | **VAL\_2**]

SCHEMA:

"field1" -> 1

"field2" -> 2





# Схема

[**SCHEMA**] [LENGTH\_1 | VAL\_1] [LENGTH\_2 | VAL\_2]

SCHEMA:

"field1" -> 1

"field2" -> 2

- Описывает порядковые номера полей

# Схема

[**SCHEMA**] [**LENGTH\_1** | **VAL\_1**] [**LENGTH\_2** | **VAL\_2**]

SCHEMA:

"field1" -> 1

"field2" -> 2

- Описывает порядковые номера полей
- Храним схемы в кластере

# Схема

[**SCHEMA**] [LENGTH\_1 | VAL\_1] [LENGTH\_2 | VAL\_2]

SCHEMA:

"field1" -> 1

"field2" -> 2

- Описывает порядковые номера полей
- Храним схемы в кластере
- Время поиска осталось  $O(N)$ !



# Смещение вместо длины

**[SCHEMA]** **[OFFSET\_1 | OFFSET\_2]** **[VAL\_1 | VAL\_2]**

SCHEMA:

"field1" -> 1

"field2" -> 2



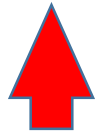
# Смещение вместо длины

[**SCHEMA**] [OFFSET\_1 | OFFSET\_2] [VAL\_1 | VAL\_2]

SCHEMA:

"field1" -> 1

"field2" -> 2



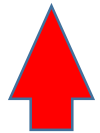
# Смещение вместо длины

[**SCHEMA**] [OFFSET\_1 | OFFSET\_2] [VAL\_1 | VAL\_2]

SCHEMA:

"field1" -> 1

"field2" -> 2



1) Получаем позицию поля ( $O(1)$ )



# Смещение вместо длины

[**SCHEMA**] [OFFSET\_1 | **OFFSET\_2**] [VAL\_1 | VAL\_2]



SCHEMA:

"field1" -> 1

"field2" -> 2



- 1) Получаем позицию поля ( $O(1)$ )
- 2) Путем арифметических операций узнаем смещение

# Смещение вместо длины

[**SCHEMA**] [OFFSET\_1 | OFFSET\_2] [VAL\_1 | VAL\_2]



SCHEMA:

"field1" -> 1

"field2" -> 2



- 1) Получаем позицию поля ( $O(1)$ )
- 2) Путем арифметических операций узнаем смещение
- 3) Переходим по смещению



# Выводы

- varint: экономит место ценой CPU, не всегда оправдано
- Строки: кодировка имеет значение, UTF8 как default
- Кроссплатформенность: баланс между платформами
- Метаданные: идентификаторы для компактности
- Кодогенерация: ускоряет, но и усложняет



# Контакты

@devozerov

<http://ignite.apache.org>

[https://github.com/devozerov/ozero\\_2017\\_joker](https://github.com/devozerov/ozero_2017_joker)

# Вопросы?

