2026

Yaroslav Tkachenko

MAKING KAFKA SERIALIZATION FAST AGAIN: STOP TRUSTING DEFAULTS

Yaroslav Tkachenko

My background

- → Started my career as a full-stack web developer, worked in a few startups
- → Technical Lead at **Activision**, **Shopify**
- → Founding Engineer at **Goldsky**
- → Data streaming tooling, consulting and training at **Irontools**

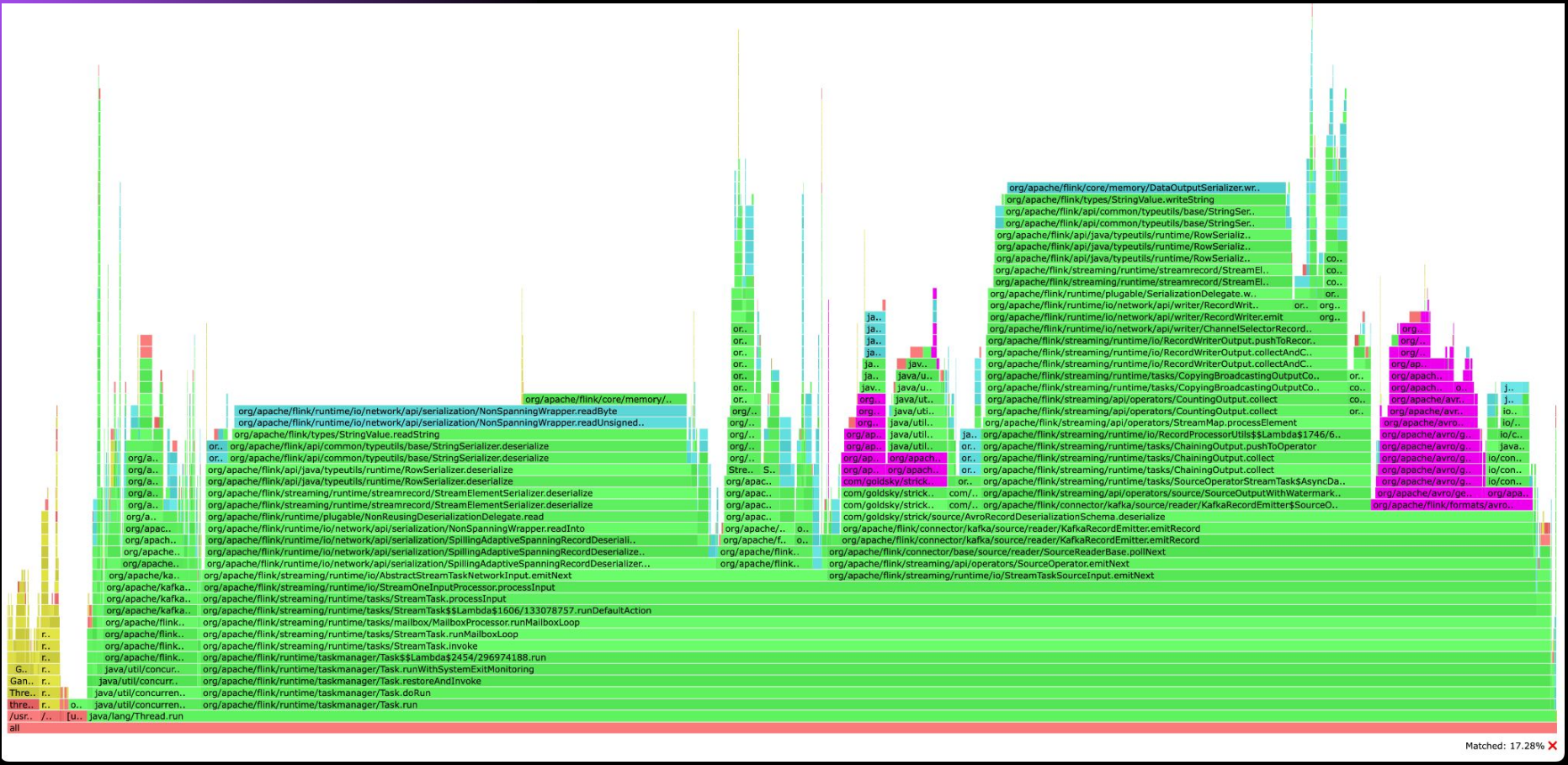
https://www.linkedin.com/in/sap1ens/

Your data serialization is slow

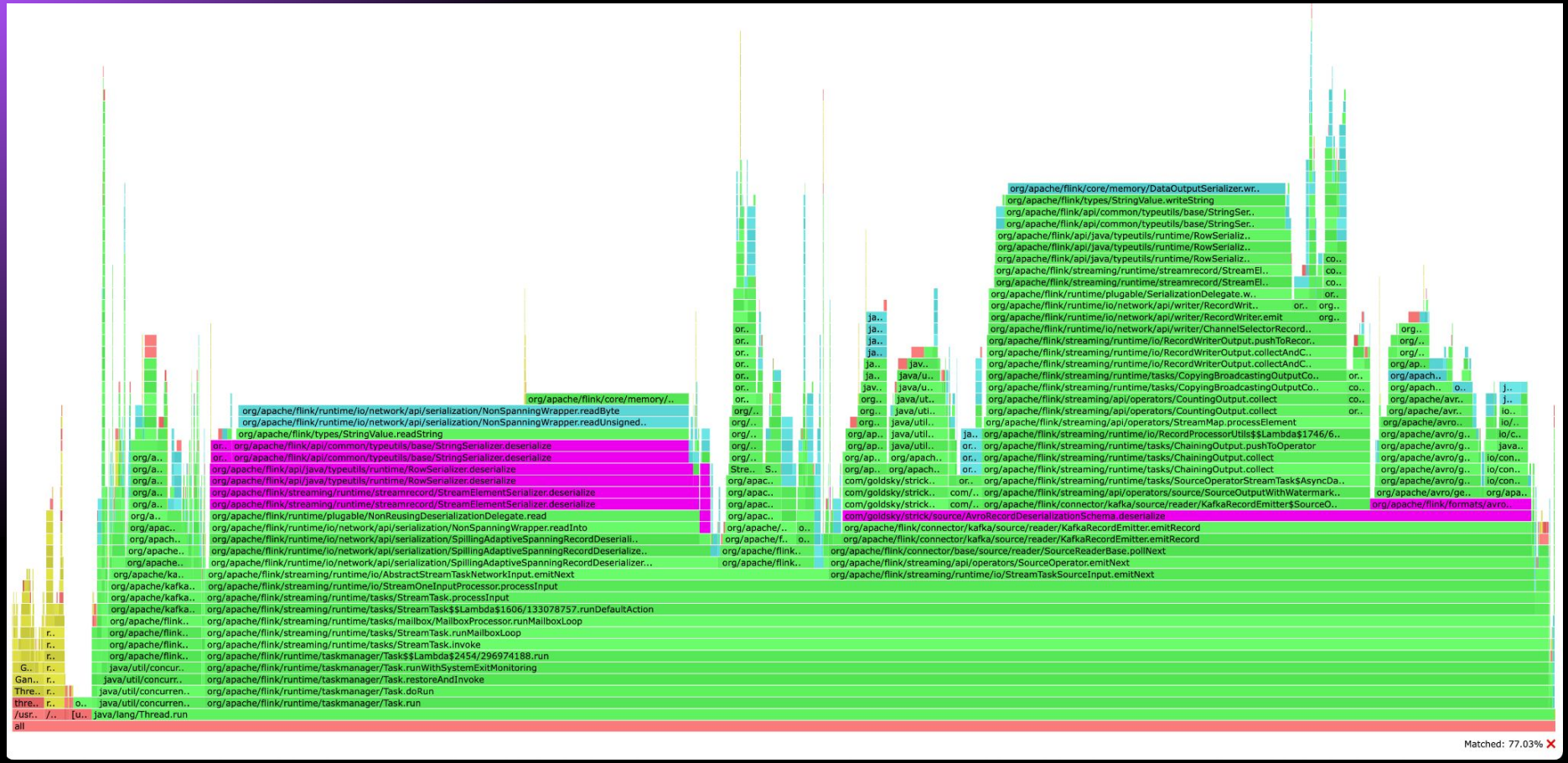
And likely can be 2x - 3x faster

OPINION

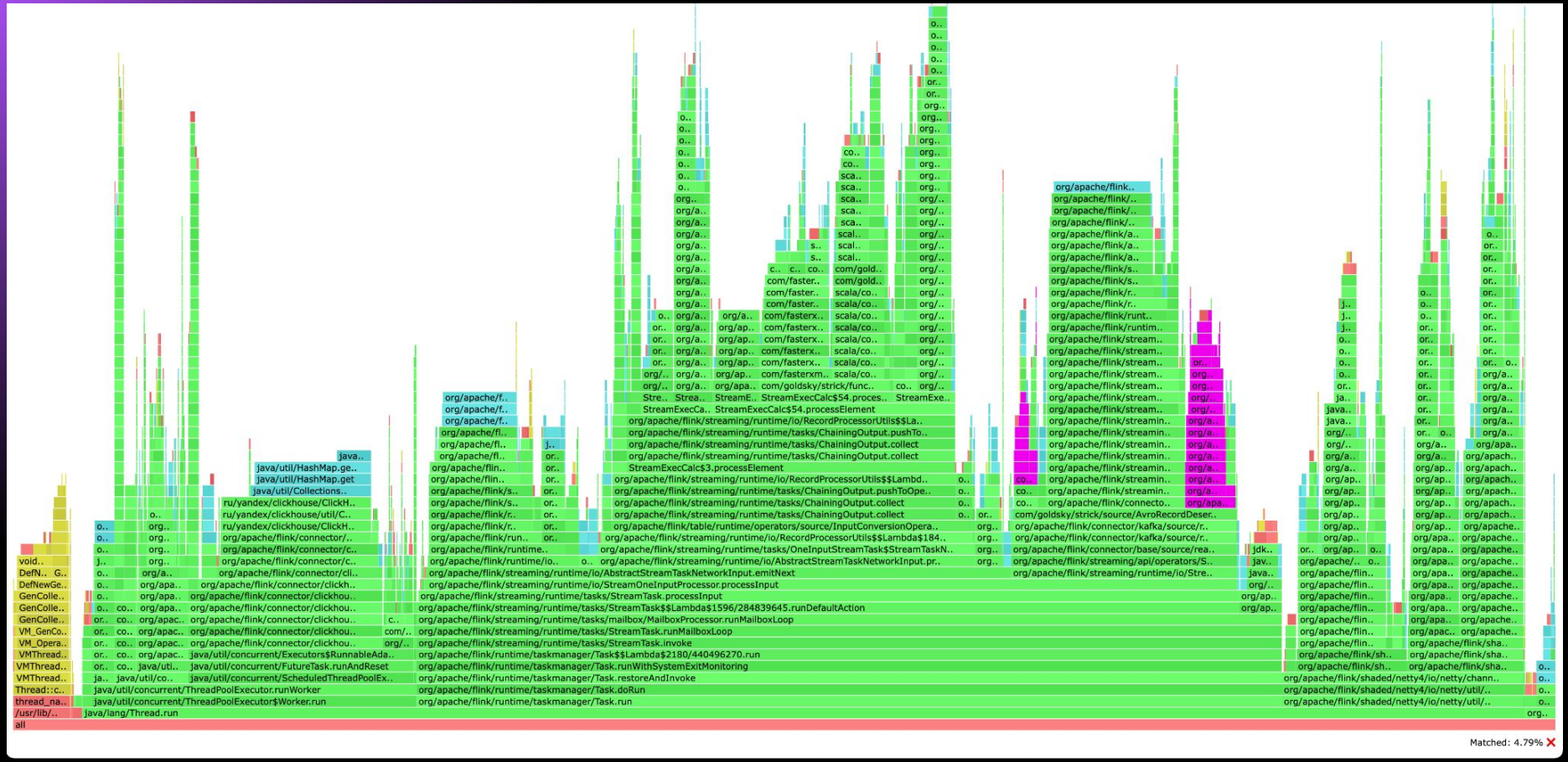
Serialization speed can be the biggest contributing factor to the performance of **stateless** streaming pipelines



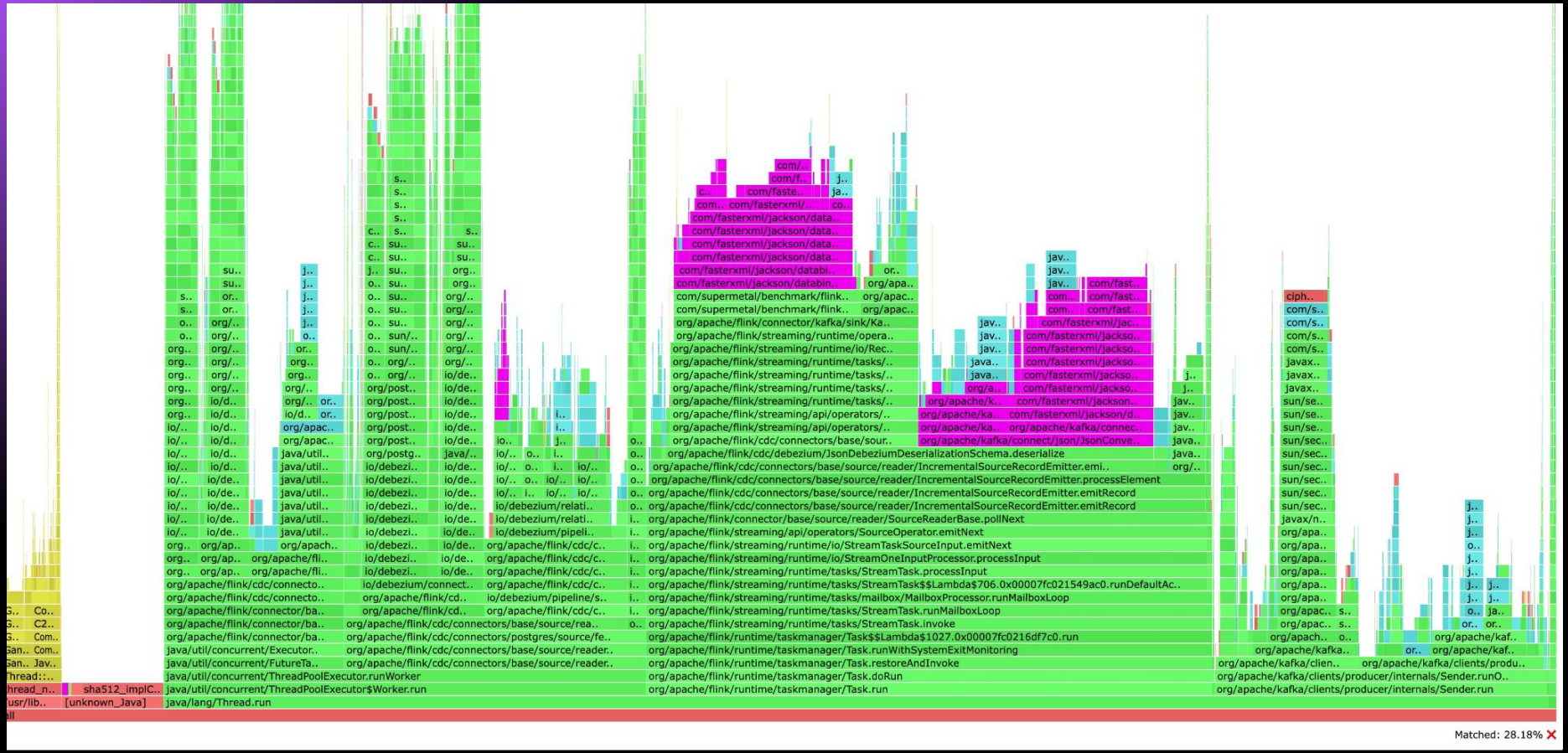
Kafka Avro Flink Source Deserializer



Kafka Avro Flink Source Deserializer



Kafka Avro Flink Source Deserializer



Flink CDC JSON Debezium Deserializer

THINGS CAN GET WORSE

- → Complex schemas
  - Arrays, nested objects, etc.
- → Large payloads
  - E.g. WarpStream recommends setting max.request.size = **64000000**
- → Number of Kafka topic partitions can be a bottleneck
  - Unless using non-standard multi-threading approaches for scaling

What can we do?

KEY OPTIMIZATIONS

Vectorization / SIMD

Leverage modern CPU instructions to process multiple data points at once.

→ Most modern CPUs support it.

→ Historically supported well in C++.

→ Java is lagging*

Code Specialization

Generate specialized code in runtime.

→ Highly specialized code can be very compact, eliminate branching and function calls.

→ JIT-friendly.

→ Works great with schemas and schema registries.

Columnar Formats

Organizing data in columns.

→ Columnar storage and processing have been used for many years in OLAP systems.

→ Enables CPU cache locality.

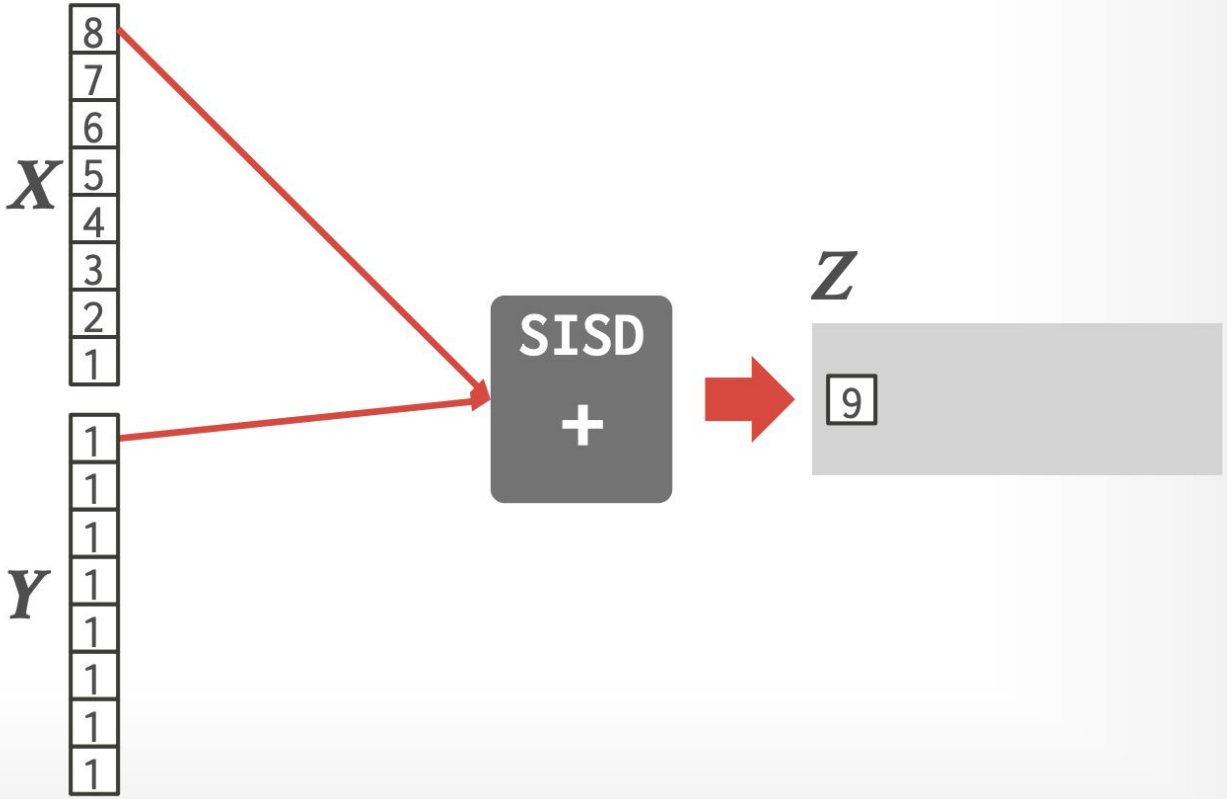
→ Enables efficient vectorization.

Vectorization / SIMD

$$X + Y = Z$$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1+y_1 \\ x_2+y_2 \\ \vdots \\ x_n+y_n \end{bmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```

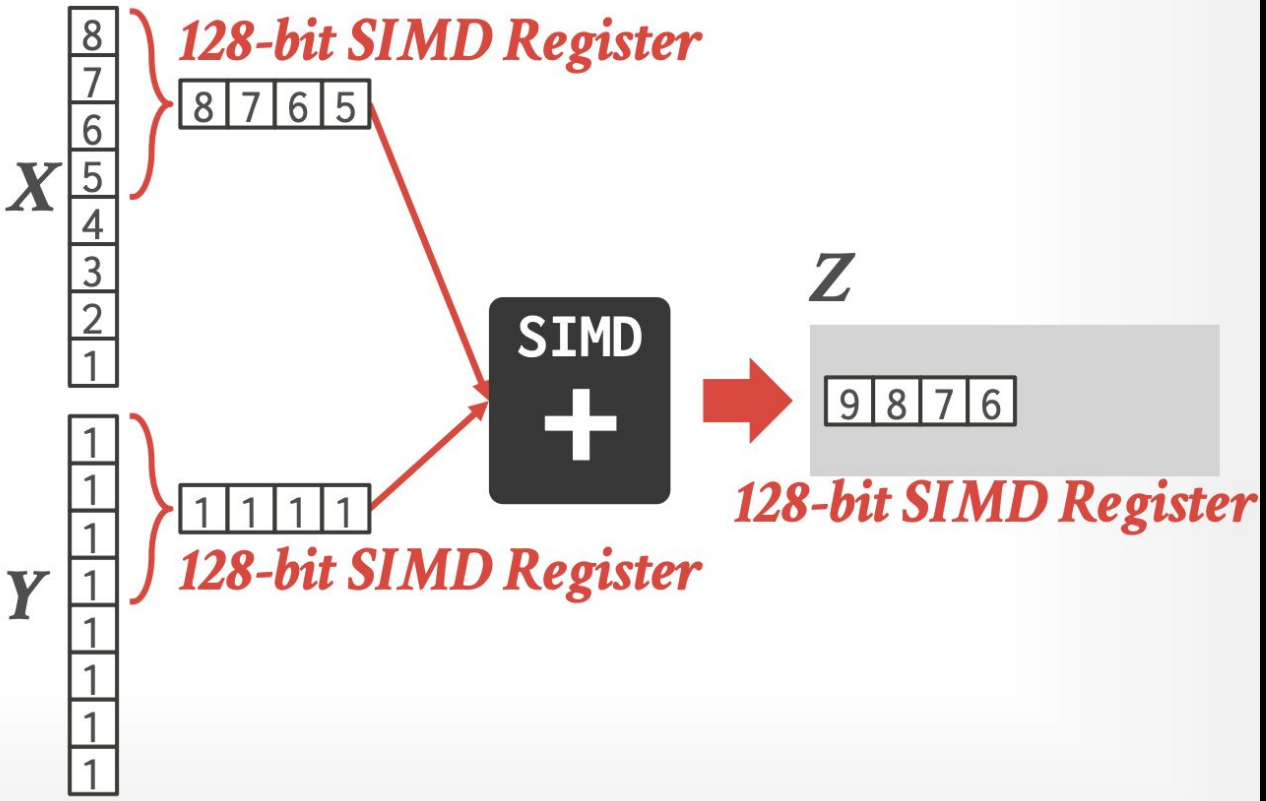


CMU·DB

$$X + Y = Z$$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1+y_1 \\ x_2+y_2 \\ \vdots \\ x_n+y_n \end{bmatrix}$$

```
for (i=0; i<n; i++) {
  Z[i] = X[i] + Y[i];
}
```



CMU·DB

MANUAL VECTORIZATION

Use CPU intrinsics to manually marshal data
between SIMD registers and execute vectorized instructions.

Probably not something most people enjoy. Look for libraries that implement this support.

```
void add(int *X,
         int *Y,
         int *Z) {
  __m128i *vecX = (__m128i*)X;
  __m128i *vecY = (__m128i*)Y;
  __m128i *vecZ = (__m128i*)Z;
  for (int i=0; i<MAX/4; i++) {
    _mm_store_si128(vecZ++,
      ↳_mm_add_epi32(*vecX++,
                     ↳*vecY++));
  }
}
```

Java Version	Status	Package	`ByteVector` availability
15	*Preview only* (very early)	`jdk.incubator.vector`	❌ Not usable yet (incomplete)
16	**1st incubator (JEP 338)**	`jdk.incubator.vector`	✅ Available
17 (LTS)	2nd incubator (JEP 414)	`jdk.incubator.vector`	✅ Available
18	3rd incubator (JEP 417)	`jdk.incubator.vector`	✅ Available
19	4th incubator (JEP 426)	`jdk.incubator.vector`	✅ Available
20	5th incubator (JEP 438)	`jdk.incubator.vector`	✅ Available
21 (LTS)	**Finalized Vector API**	`jdk.vector` (not incubator)	⚠️ Package moved
22+	Final	`jdk.vector`	

SIMD vs SWAR

SIMD

```
var vec =
ByteVector.fromArray(SPECIES_256,
array, i);

var cmp = vec.eq((byte)0);

if (cmp.anyTrue()) {
    // zero found
}
```

SWAR

```
long x = load64(ptr);

long tmp = (x -
0x0101010101010101L) & ~x &
0x8080808080808080L;

if (tmp != 0) {
    // zero byte present
}
```

Code Specialization

Can we perform the **same** amount of work with **fewer** CPU instructions?

```
Object read(Object obj, Schema schema) {
    switch(schema.type()) {
        case STRUCT:
            return readStruct(obj, schema);
        case STRING:
            return readString(obj, schema);
        // ... rest of the types
    }
}

Object readStruct(Object obj, Schema schema) {
    Struct struct = new Struct();
    for (Field field: schema.fields()) {
        struct.field(field.name(), read(field.value(), field.schema()));
    }
    return struct;
}
```

"Interpreted" Approach

```
User read(Object obj, Schema schema)  {
    User user = new User();
    Decoder decoder = new Decoder(obj, schema);

    user.put(0, decoder.readString());
    user.put(1, decoder.readString());
    user.put(2, decoder.readLong());

    Order order = new Order();
    order.put(0, decoder.readString());
    user.put(3, order);

    return user;
}
```

Compiled Approach

RUNTIME CODE SPECIALIZATION IN JVM

→ You can generate Java code, compile and instantiate it with a classloader. So you'll need:

- Code generation: via some DSL or templating
- Compiler: Janino, javac (!)
- A way to load / switch implementations

→ Or you can generate bytecode directly

- ASM is a very popular library for that

CODE SPECIALIZATION FOR DESERIALIZATION

For every new detected schema:

⟶ Create a specialized deserializer tailored to the schema

⟶ Use that instead of the default serializer

- Can still fallback to the default if needed
- If compilation takes a while, use the default until it's done, then switch

JSON

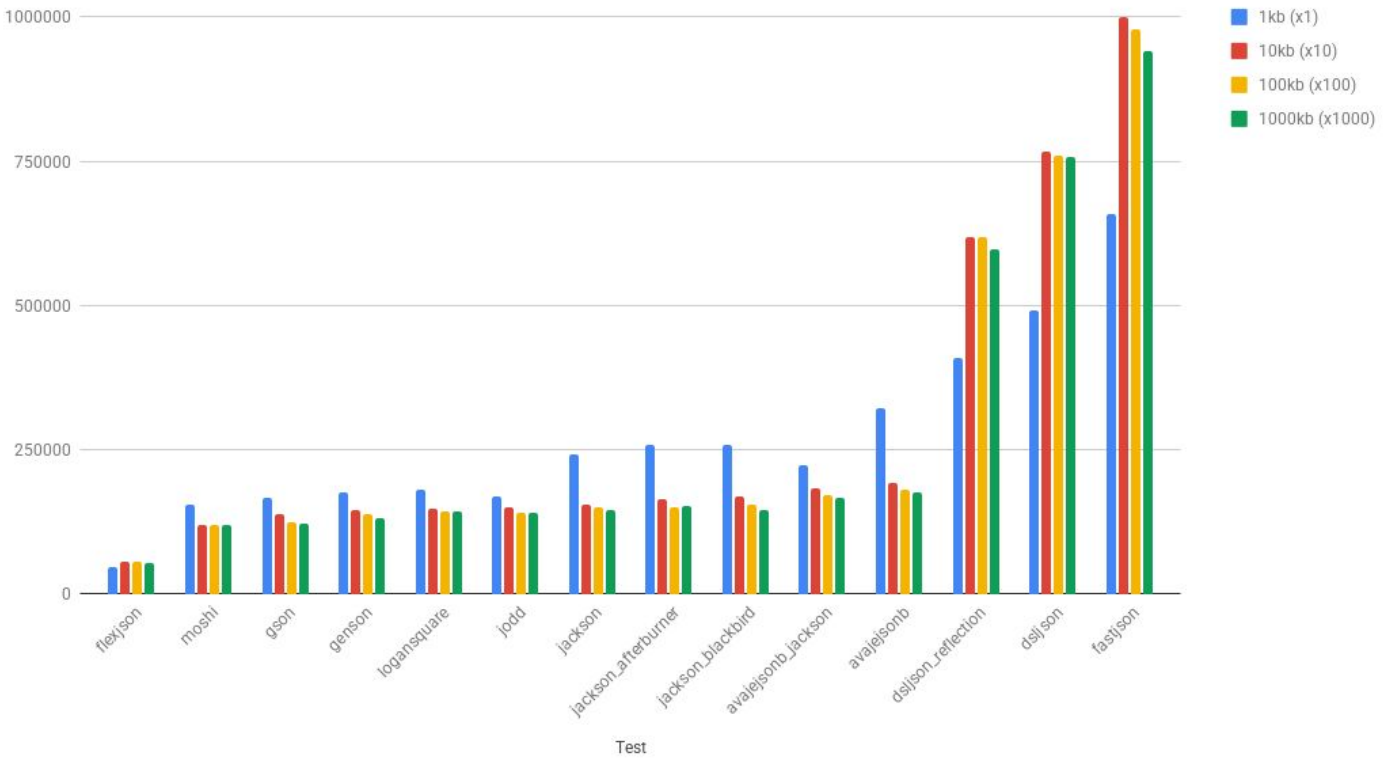
JSON DESERIALIZATION

When it comes to Java JSON serialization / deserialization, **Jackson** is everywhere

→ Confluent's KafkaJsonDeserializer

→ Spring's JsonDeserializer

→ Flink's JsonDeserializationSchema



github.com/fabienrenaud/java-json-benchmark

README Apache-2.0 license

simdjson-java

CI passing License Apache 2

A Java version of simdjson - a JSON parser using SIMD instructions, based on the paper Parsing Gigabytes of JSON per Second by Geoff Langdale and Daniel Lemire.

Code Sample

DOM Parser

```
byte[] json = loadTwitterJson();

SimdJsonParser parser = new SimdJsonParser();
JsonValue jsonValue = parser.parse(json, json.length);
Iterator<JsonValue> tweets = jsonValue.get("statuses").arrayIterator();
while (tweets.hasNext()) {
    JsonValue tweet = tweets.next();
    JsonValue user = tweet.get("user");
    if (user.get("default_profile").asBoolean()) {
        System.out.println(user.get("screen_name").asString());
    }
}
```

Schema-Based Parser

```
byte[] json = loadTwitterJson();

SimdJsonParser parser = new SimdJsonParser();
SimdJsonTwitter twitter = simdJsonParser.parse(buffer, buffer.length, SimdJsonTwitter.class);
for (SimdJsonStatus status : twitter.statuses()) {
    SimdJsonUser user = status.user();
    if (user.default_profile()) {
        System.out.println(user.screen_name());
    }
```

github.com/simdjson/simdjson-java

```java
public class SimdJsonDeserializer<T> implements Deserializer<T> {

  private final SimdJsonParser parser = new SimdJsonParser();
  private final Class<T> clazz;

  public SimdJsonDeserializer(Class<T> clazz) {
    this.clazz = clazz;
  }

  @Override
  public T deserialize(String topic, byte[] data) {
    if (data == null || data.length == 0) {
      return null;
    }
    return parser.parse(data, data.length, clazz);
  }
}
```

“We require Java 24 or better”

“IllegalArgumentException: Unsupported vector species: Species[byte, 16, S_128_BIT]”

BENCHMARK

Benchmark	Mode	Cnt	Score	Error	Units
Benchmark.jackson	avgt	10	565.766 ±	2.474	ns/op
Benchmark.simdJava	avgt	10	8878.502 ±	29.739	ns/op

README | Code of conduct | Contributing | Apache-2.0 license | Security

build passing | coverage 74% | maven-central v2.0.61.android8 | release v2.0.61 | Java 8+ | license Apache-2.0

gitpod ready-to-code | latest snapshot v2.0.58-SNAPSHOT | Stars 4.3k | Forks 551 | user repos 0 | contributors 111

English Documentation | 中文文档

The issues of fastjson will be also posted on Alibaba Cloud Developer Community

FASTJSON v2

`FASTJSON v2` is an upgrade of the `FASTJSON`, with the goal of providing a highly optimized `JSON` library for the next ten years.

- Supports the JSON and JSONB Protocols.
- Supports full parsing and partial parsing.
- Supports Java servers and Android Clients, and has big data applications.
- Supports Kotlin https://alibaba.github.io/fastjson2/Kotlin/kotlin_en
- Supports Android 8+
- Supports `JSON Schema` https://alibaba.github.io/fastjson2/JSONSchema/json_schema_en

github.com/alibaba/fastjson2

fastjson2 uses code specialization and SWAR (and other optimizations)

```java
public class IronJsonDeserializer<T> implements Deserializer<T> {

  private final Class<T> clazz;
  private final ObjectReader<T> objectReader;
  private final JSONReader.Context jsonContext;

  public IronJsonDeserializer(Class<T> clazz) {
    this.clazz = clazz;
    ObjectReaderProvider provider = JSONFactory.getDefaultObjectReaderProvider();
    this.jsonContext = new JSONReader.Context(provider);
    this.objectReader = provider.getObjectReader(clazz,false);
  }

  @Override
  public T deserialize(String topic, byte[] data) {
    if (data == null || data.length == 0) {
      return null;
    }

    try (JSONReader reader = JSONReader.of(data, jsonContext)) {
      return objectReader.readObject(reader, clazz,null, 0);
    }
  }
}
```

```
    // Create class writer
    ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS | ClassWriter.COMPUTE_FRAMES);

    // Define class
    cw.visit(
        Opcodes.V1_8,
        Opcodes.ACC_PUBLIC,
        internalName,
        null,
        "java/lang/Object",
        null
    );

    // Add public fields
    for (int i = 0; i < rowType.getFieldCount(); i++) {
      String fieldName = rowType.getFieldNames().get(i);
      LogicalType fieldType = rowType.getTypeAt(i);
      String typeDescriptor = getTypeDescriptor(fieldType);
      String signature = getTypeSignature(fieldType);

      // Add public field
      FieldVisitor fv = cw.visitField(
          Opcodes.ACC_PUBLIC,
          fieldName,
          typeDescriptor,
          signature,
          null
      );

      // Add JSONField annotation to field
      AnnotationVisitor av = fv.visitAnnotation("Lcom/alibaba/fastjson2/annotation/JSONField;", true);
      av.visit("name", fieldName);
      av.visitEnd();
      fv.visitEnd();
    }
```

```
1.  switch (type.getTypeRoot()) {
2.    case CHAR:
3.    case VARCHAR:
4.      return "Ljava/lang/String;";
5.    case BOOLEAN:
6.      return "Ljava/lang/Boolean;";
7.    case TINYINT:
8.      return "Ljava/lang/Byte;";
9.    case SMALLINT:
10.     return "Ljava/lang/Short;";
11.   case INTEGER:
12.     return "Ljava/lang/Integer;";
13.   // ...
14. }
```

BENCHMARK

Benchmark	Mode	Cnt	Score	Error	Units
Benchmark.fastjson2	avgt	10	173.572 ±	0.913	ns/op
Benchmark.jackson	avgt	10	565.766 ±	2.474	ns/op
Benchmark.simdJava	avgt	10	8878.502 ±	29.739	ns/op

Avro

AVRO DESERIALIZATION

When it comes to Java Avro serialization / deserialization, almost always the standard **org.apache.avro** library is used.

→ Confluent's KafkaAvroDeserializer

→ Flink's AvroDeserializationSchema

→ ...

README | Contributing | BSD-2-Clause license

Avro-Util

push flow passing | latest 0.4.38

A collection of utilities and libraries to allow java projects to better work with avro.

How To Use

Artifacts are published to artifactory. To use them (say in a gradle build) you'd need to add that repository to your build.gradle, and then add a dependency on the module(s) you wish to use:

```
repositories {
   maven {
      url  "https://linkedin.jfrog.io/artifactory/avro-util/"
   }
}
...
dependencies {
   compile "com.linkedin.avroutil1:helper-all:<latest version>"
}
```

Background

Apache Avro is a widely used serialization format.

Unfortunately it is not always possible to write java code that "just works" across multiple versions of avro, as there have been breaking changes to both the API and wire format. This project provides utility code to enable java developers to write code that is compatible across a wide range of avro versions

Furthermore, this project enables Avro-based projects to achieve better performance.

github.com/linkedin/avro-util

avro-fastserde uses code specialization, it starts compilation for a new schema asynchronously and performs a swap when it’s ready

```
[INFO ] 2026-02-28 15:45:56.586 [avro-fastserde-compile-thread-1]  FastSerdeBase - Starting compilation for the generated source file: User_GenericDeserializer_739236478_739236478.java

[INFO ] 2026-02-28 15:45:56.868 [avro-fastserde-compile-thread-1]  FastSerdeBase - Successfully compiled class User_GenericDeserializer_739236478_739236478 defined at source file User_GenericDeserializer_739236478_739236478.java

[DEBUG] 2026-02-28 15:45:56.870 [avro-fastserde-compile-thread-1]  FastSerdeCache - Generated classes dir: /var/folders/jl/60lw5h01623gd51ffm22hzrr0000gn/T/generated8742529887734165862 and generation of generic FastDeserializer is done for writer schema of type: example.avro.User with fingerprint: {
  "type" : "record",
  "name" : "User",
  "namespace" : "example.avro",
  "fields" : [ {
    "name" : "id",
    "type" : "int"
  },

[DEBUG] 2026-02-28 15:45:57.553 [main] FastGenericDatumReader -  FastGenericDeserializer was generated and cached for reader schema:
[{"type":"record","name":"User","namespace":"example.avro","fields":[{"name":"id","type":"int"},
```

```
1.  public IndexedRecord deserialize(IndexedRecord reuse, Decoder decoder,
    DatumReaderCustomization customization)  throws IOException {
2.      return deserializeUser0( null, (decoder), (customization));
3.  }
4.
5.  public IndexedRecord deserializeUser0(Object reuse, Decoder decoder,
    DatumReaderCustomization customization)  throws IOException {
6.      IndexedRecord User;
7.      User = new org.apache.avro.generic.GenericData.Record(readerSchema);
8.      User.put( 0, (decoder.readInt()));
9.      Utf8 charSequence0 = (decoder).readString( null);
10.     User.put( 1, charSequence0);
11.     Utf8 charSequence1 = (decoder).readString( null);
12.     User.put( 2, charSequence1);
13.     // ..
14. }
```

BENCHMARK

Benchmark	Mode	Cnt	Score	Error	Units
Benchmark.avroDeserializationComplex	avgt	10	610.802	± 10.620	ns/op
Benchmark.avroDeserializationStandard	avgt	10	384.341	± 4.519	ns/op
Benchmark.avroDeserializationTrivial	avgt	10	131.794	± 4.027	ns/op
Benchmark.ironAvroDeserializationComplex	avgt	10	224.669	± 7.045	ns/op
Benchmark.ironAvroDeserializationStandard	avgt	10	134.520	± 3.171	ns/op
Benchmark.ironAvroDeserializationTrivial	avgt	10	44.185	± 0.609	ns/op

SUMMARY

VECTORIZATION / SIMD

⟶ Leverage modern CPU instructions to process multiple data points at once.

⟶ Check various SIMD JSON libraries, e.g. simdjson-java.

⟶ fastjson2 uses SIMD within a register (SWAR) technique, not specialized SIMD instructions*.

CODE SPECIALIZATION / COMPILATION

⟶ Specialized compiled code with minimum instructions is always faster.

⟶ Both fastjson2 and avro-fastserde compile specialized serializer / deserializer classes for a given schema (or class) in runtime.

⟶ Still compatible with Schema Registries.

OPINION

Most importantly, do not trust the defaults: measure things yourself.

QUESTIONS?

NEWSLETTER!

https://streamingdata.tech