TDT4225
Chapter 4 – Encoding and Evolution

Svein Erik Bratsberg
Department of Computer Science (IDI), NTNU

Evolvability - Basics

- SQL databases assume a schema – may be changed with ALTER TABLE
- Schema-on-read databases don’t enforce a schema – a database may contain older and newer formats
- Server-side applications – rolling upgrades
- Client-side applications – at the mercy of the user who may not upgrade for some time?
- *Backward compatibility*: Newer code can read old data
- *Forward compatibility*: Older code can read new data. Tricky.

Formats for encoding data

- In memory, data is kept in objects, structs, lists, arrays, hash tables, trees, etc. Use pointers.
- When writing to a file or sending it over the network: Self-contained sequence of bytes. No pointers.
- Encoding/decoding, marshalling/unmarshalling, serialization/deserialization: sending and receiving data
- Bulit-in language support
  - Java has java.io.Serializable
  - Ruby has Marshal
  - Python has pickle
- Tied to a particular programming language
- Decoding may cause security problems due to arbitrary code has to be installed
- Versioning?
- Performance?

Data formatting languages (textual)

- JSON – Supported by browsers and Javascript
- XML – complex data formats
- CSV – simple data format
- Ambigiuity in encoding numbers/strings and int/float
- Support for UNICODE, but no support for binary strings (encode binary streams using text)
- Schema support for JSON and XML. Used in XML, but skipped in JSON. Hard-coded interpretation in the apps.
- CSV has no schema support
- JSON, XML, and CSV are good enough for many purposes
- “The difficulty in getting organizations to agree on anything, outweights most others concerns.”

Binary encoding

- Used for internal data between programs
- Compacter and faster than textual encoding
- Binary encodings for JSON: MessagePack, BSON, BJSON, UBJSON, BISON, and Smile.
- For XML: WBXML and Fast Infoset.

Example 4-1. Example record which we will encode in several binary formats in this chapter

```
{
    "userName": "Martin",
    "favoriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

MessagePack

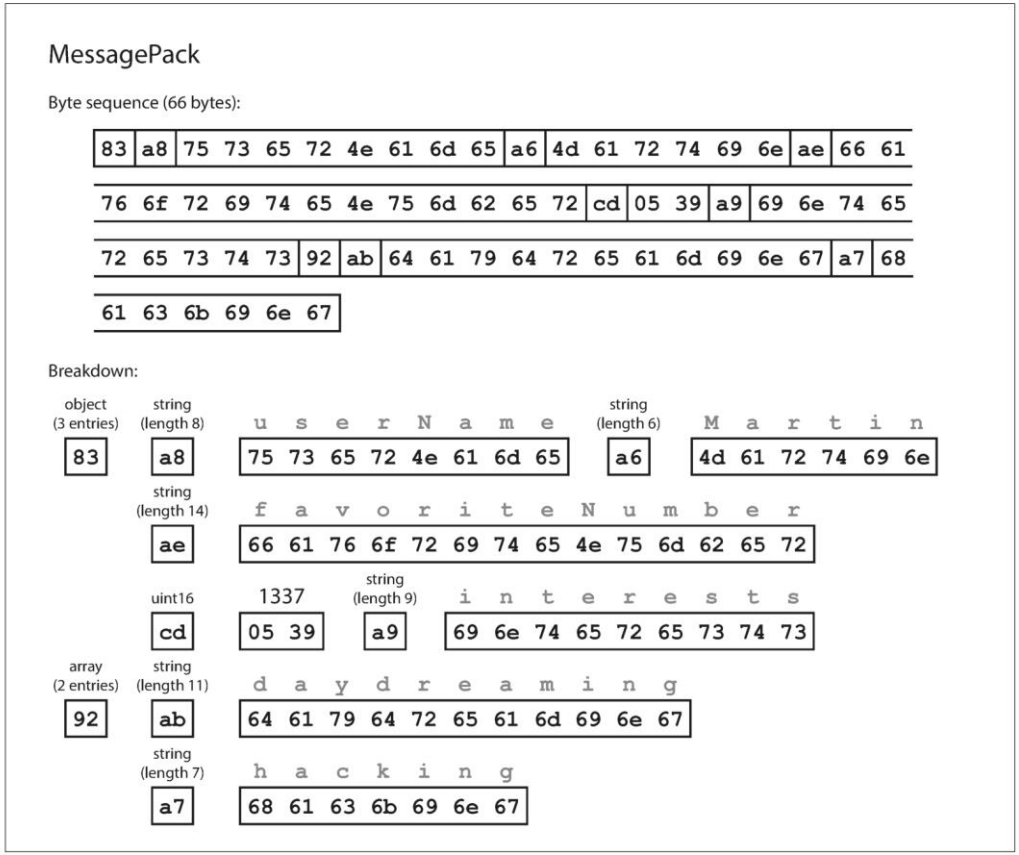

Figure 4-1. Example record (Example 4-1) encoded using MessagePack.

Thrift and Protocol Buffers

- Apache Thrift (Facebook) and Protocol Buffers (Google) are binary encodings
- Both require a schema, with tools to generate code

```
struct Person {
  1: required string       userName,
  2: optional i64          favoriteNumber,
  3: optional list<string> interests
}
```

```
message Person {
    required string user_name       = 1;
    optional int64  favorite_number = 2;
    repeated string interests       = 3;
}
```

Thrift's BinaryProtocol

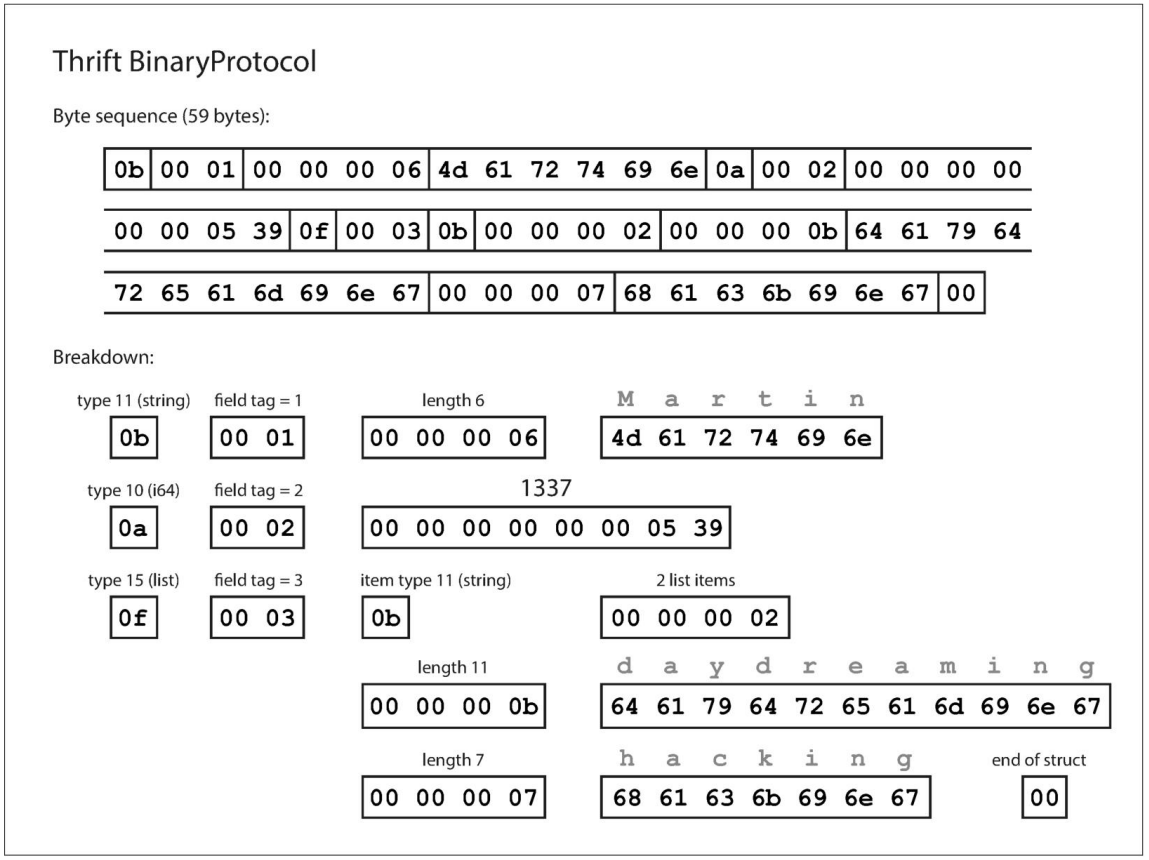

Figure 4-2. Example record encoded using Thrift's BinaryProtocol.

Thrift CompactProtocol

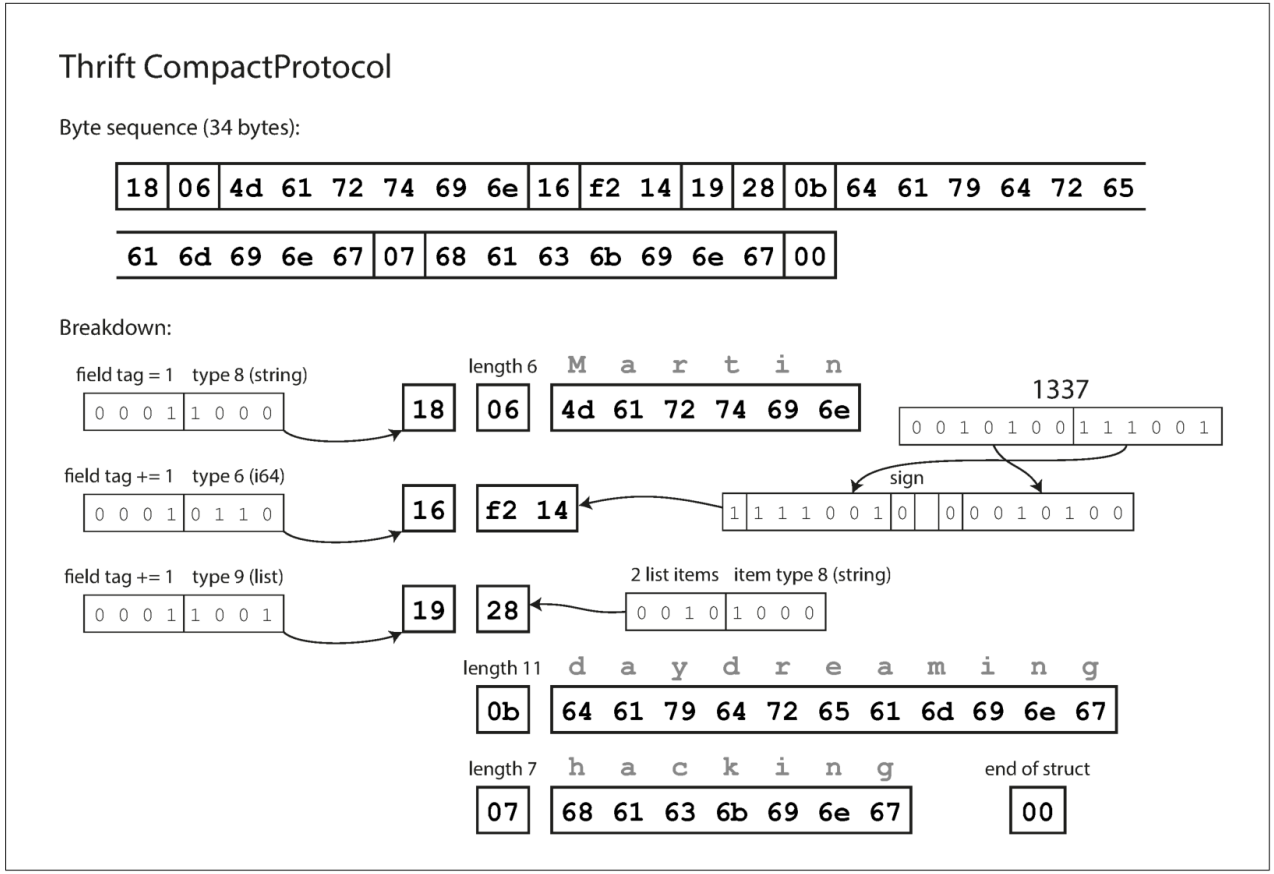

Figure 4-3. Example record encoded using Thrift's CompactProtocol.

Protocol Buffers

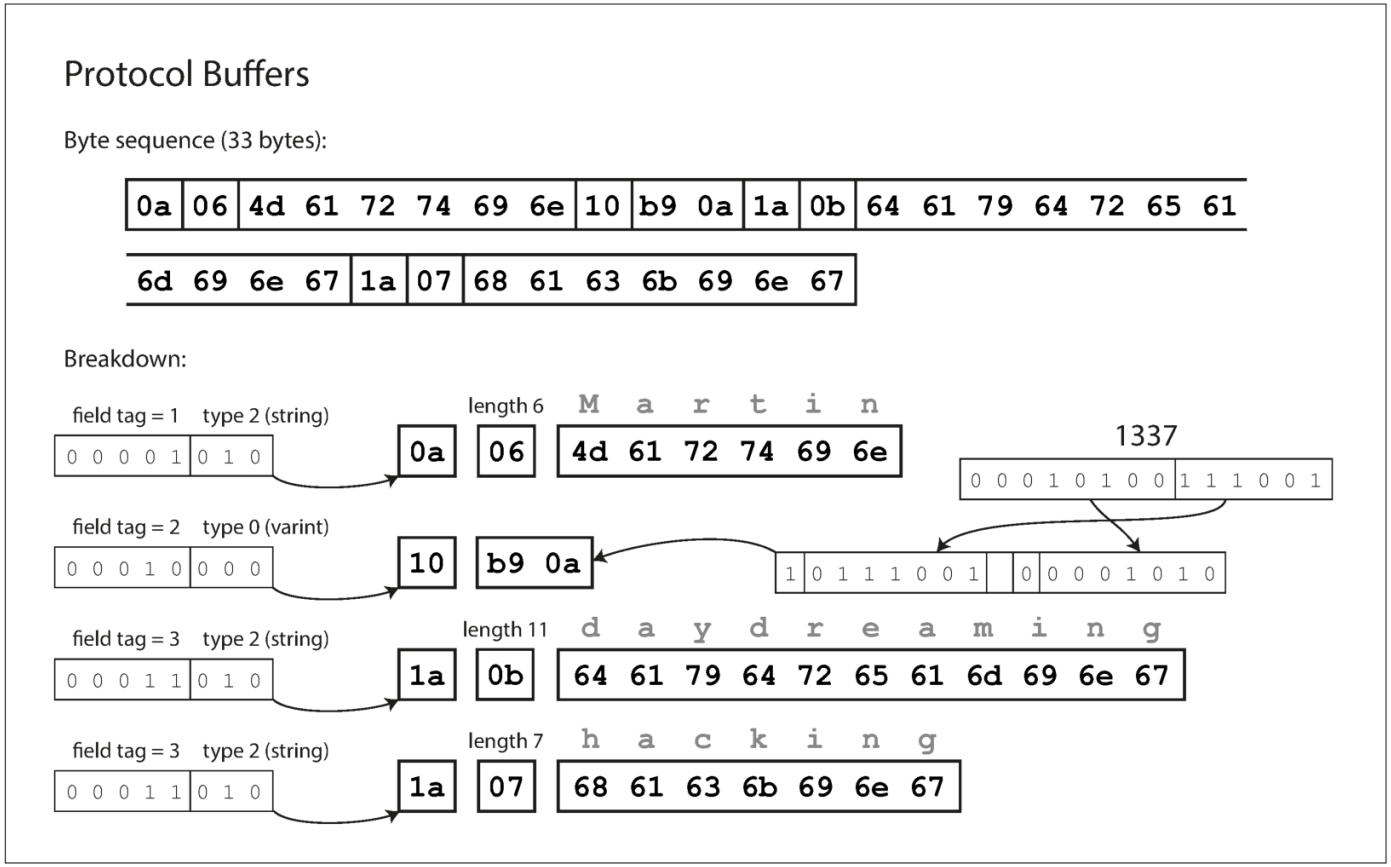

Figure 4-4. Example record encoded using Protocol Buffers.

Schema evolution

- Thrift/Protocol Buffers use schemas and fields are identified with tag numbers
- Field names may change and new fields may be added, but they must be optional or have a default value
- Optional fields may be removed
- Changing datatypes may be possible, check the documentation
- Protocol buffers has a repeated field instead of arrays

Avro

- Apache Avro is another binary encoding format that is interestingly different

```
record Person {
    string               userName;
    union { null, long } favoriteNumber = null;
    array<string>        interests;
}
```

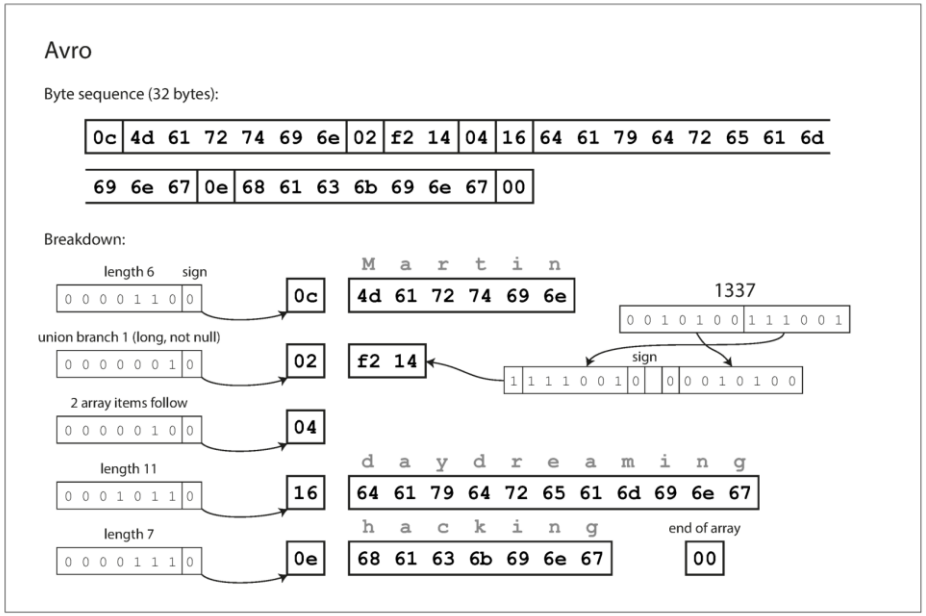

Figure 4-5. Example record encoded using Avro.

Reader’s and writer’s schema

- Avro resolves differences automatically. When reading you know both schemas.

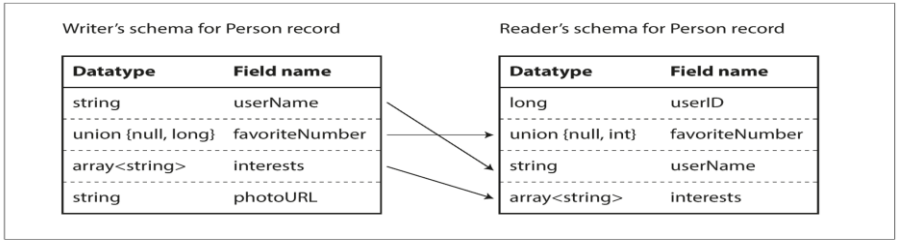

Figure 4-6. An Avro reader resolves differences between the writer’s schema and the reader’s schema.

- You may add or remove fields which have default values
- Schemas stored with large files
- In databases schema version number included with each record
- On network messages, schema versions are negotiated at startup
- Avro is friendlier to dynamically generated schemas
- Also better for dynamically typed language since the data is self-describing (in contrast to statically typed languages)

Advantages of using a schema

- More compact since they don’t use field names
- Schemas are documentation
- Database of schemas keep version history for backward and forward compatibility
- For users of statically typed languages, the use of schemas make compile-time type checking possible

Dataflow through databases

- Store data in databases for later use or exchange
- Data outlives code: old data may exist
- Some problems may appear

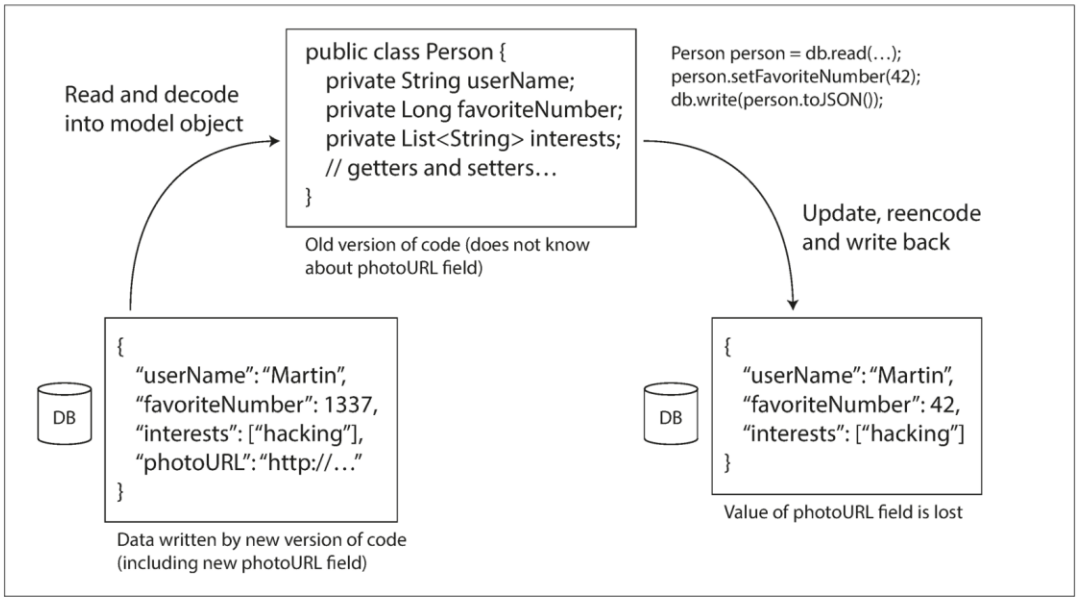

Figure 4-7. When an older version of the application updates data previously written by a newer version of the application, data may be lost if you're not careful.

Dataflow through services: REST / RPC

- Clients and servers: Servers expose services using API
- Web browser and servers:
  - GET to download HTML, CSS, JavaScript, images, etc.
  - POST to submit data
- API uses standard formats: HTTP, URLs, SSL/TLS, HTML, etc
- Applications may also use these protocols and standards (e.g. AJAX)
- Microservice architectures: Decompose a large application into smaller services by area of functionality

Web services

- Is when HTTP is used as the underlying protocol for talking to the service.
  - An app uses HTTP to access a server over the Internet
  - Services within an organization uses HTTP to communicate
  - Services used between organizations, e.g. credit card processing
- REST – a design philosophy -- RESTful interfaces
  - Using HTTP for cache control, authentication, etc.
  - URLs to describe resources
  - Focuses on code generation and use of tools
- SOAP – based on XML and its API is described by WSDL
  - Complex and not human-readable, thus tools are used
  - Used in and between large organizations

Problems with RPCs

- RPC – Remote procedure calls
- EJB – Enterprise Java Beans, Java RMI, Corba, DCOM, Sun RPC, etc.
- Make a request to a remote network service look the same as calling a function or method
  - Unpredictable behavior – error handling necessary
  - Timeouts (no answer)
  - Idempotence? Retries.
  - Response time widely variable
  - Parameters must be marshalled and complex objects are problematic
  - Different programming languages at both ends? Problems with data types?

The future for RPC

- Thrift, Avro, gRPC (Protcol Buffers), Finagle, Rest.li are examples of RPCs.
- Using *futures* simplifies error handling and parallel requests
- gRPC uses *streams (multiple calls and replies)*
- Custom RPC using binary encoding may achieve good performance
- JSON over REST may provide good flexibility and is supported by many tools and programming languages

Message passing dataflow

- Asynchronous message-passing systems
- Message broker / message queue / message-oriented middleware
- Buffer – more reliable communication
- Redeliver message to failed servers
- The sender does not need to know the addresses of receivers
- Send to more recipients
- Decouples senders and receivers
- Asynchronous – send and forget
- One-way
- Systems: TIBCO, WebSphere, Kafka, NATS, RabbitMQ

Distributed actor frameworks

- Programming model with independent actors and messages
- Message may be lost and automatically resent
- The same messages used internally or distributed
- Akka (java), Erlang OTP, Orleans
- Must solve schema evolution without much help from the system.

Summary

- Data encoding should support rolling upgrades
- Must ensure backward compatibility
- Programming language solutions – efficient, but locked
- Textual formats like JSON, XML and CSV support schemas, but are vague about datatypes
- Binary schema–driven formats like Thrift, Protocol Buffers, and Avro allow compact, efficient encoding
- Modes of dataflow
  - Through databases
  - RPC and REST API
  - Message passing