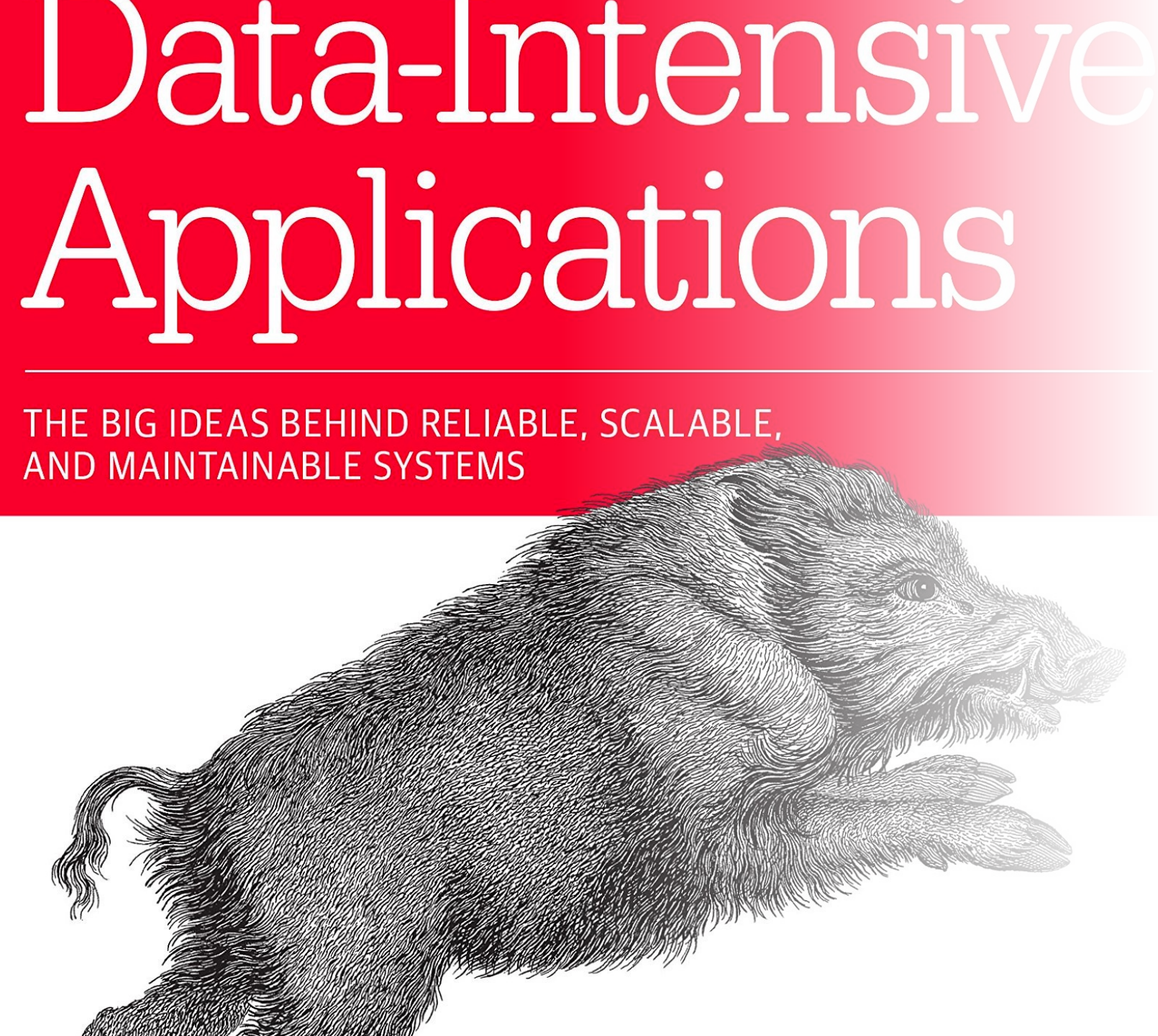

Chapter 4 : Encoding and Evolution

Why Encoding and Evolution?

- Systems evolve over the time.
  - New business requirements
  - New Features gets added
  - Business circumstances change
- New Data needs to be stored or data type gets added in DB.
- Application code also needs to handle this schema
  - On server side applications - Rolling upgrade(aka staged rollout)
  - Client side applications – depends on user
- Old and New versions of code exist at same time, so old and new formats of data can coexist in the system.
- We have to maintain compatibility
  - Backward compatibility – Newer code can read data that was written by older code
  - Forward compatibility – Older code can read data that was written by new code

What is encoding and decoding?

- Programs work with data in at least two different representations:
  - Memory
    - Lists
    - Structs
    - Arrays etc..
- Writing data to file or transferring over the network
  - Self contained sequence of bytes
- Thus translation between two representations, the translation from the in-memory representation to a byte sequence is called encoding(also serialization or marshalling).
- Reverse is called decoding(also parsing, deserialization or unmarshalling).

Examples of Language Specific Formats and Their problems

- Built in support for encoding in memory objects into bytes of sequences.
  - Java - java.io.Serializable
  - Ruby - Marshal
  - Python – pickle
- They are convenient but come with additional problems.
  - Bound to use current programming language
  - Security issues
  - Efficiency

JSON and XML

- JSON and XML:
  - verbose and unnecessary complicated.
  - Ambiguity around encoding of number:
- XML and CSV – cannot distinguish between a number and string.
- JSON – doesn’t distinguish integers and floating point numbers and doesn’t specify a precision.
- JSON and XML – support Unicode characters, but not binary strings(sequence of bytes without a character encoding).
  - So generally encoded into Base64.
  - Bulky – increases the data size by 33%
  - Hacky
- Optional schema in JSON and XML
- CSV does not have schema
- Binary encodings for JSON – like MessagePack, BSON, BJSON and for XML – WBXML, Fast Infoset

Binary Variants

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

- Binary encodings for JSON – like Message Pack, BSON, BJSON and for XML – WBXML, Fast Infoset}
- Binary encoding using Message Pack = 66 bytes
- Textual JSON encoding = 81 bytes
- Thrift = 34 bytes
- Buffer protocols = 33 bytes

How data flows between processes?

- Databases
- Service calls
- Asynchronous messages

Dataflow through Processes

- Backward compatibility
- Forward compatibility
- Data outlives code
- Rewriting(migration) to new schema

Dataflow through Services: REST and RPC

- Clients and Servers
- API exposed by servers – service
- SOA – Service Oriented Architecture
- Microservices Architecture
- Web Service – when HTTP is used as the underlying protocol for talking to service
  - A client application running on user's device
  - Middleware services
  - Services owned by different organizations
- Two approaches – REST and SOAP

REST vs SOAP

- REST is not a protocol rather a design philosophy
- Emphasizes simple data formats, using URLs for identifying resources, and uses HTTP features like cache control, authentication and content type negotiation
- Less code generation, automated tooling
- OpenAPI formats - like swagger used for documentation for RESTful APIs.
- SOAP – XML based protocol
- Complex multitude of related standards (web service framework- WS*_
- XML based language- WSDL (Web services description language)
- Not human readable, complex to use, relies on tool support, code generation tools

RPC – Remote Procedure Call

- RPC model - request to remote network service – calling a function or method in your programming language within the same process – location transparency
- Network request is different from a local function call:
  - Local function call is predictable, network request is unpredictable
  - Local function call returns either a response, or exception or never returns but network request might return a timeout,
  - Will have to take care of idempotency in network requests
  - Network requests latency is variable
  - In local call – we can efficiently use references of objects whereas in network we need to depend on sequence of bytes

Dataflow through Message Passing

- Message Broker (Message queue or Message Oriented Middleware)
- Advantages compared to RPC:
  - Act as buffer if recipient is unavailable – improving reliability
  - Can redeliver messages
  - One message to several recipients
  - Decouples sender from receiver

Summary

- Importance of Encoding and Evolution
- What is Encoding and Decoding?
- Encoding Formats
- Data Flows between processes
- Data flow through Databases
- Data flow through Services
- Data flow through Message passing

Thank You!