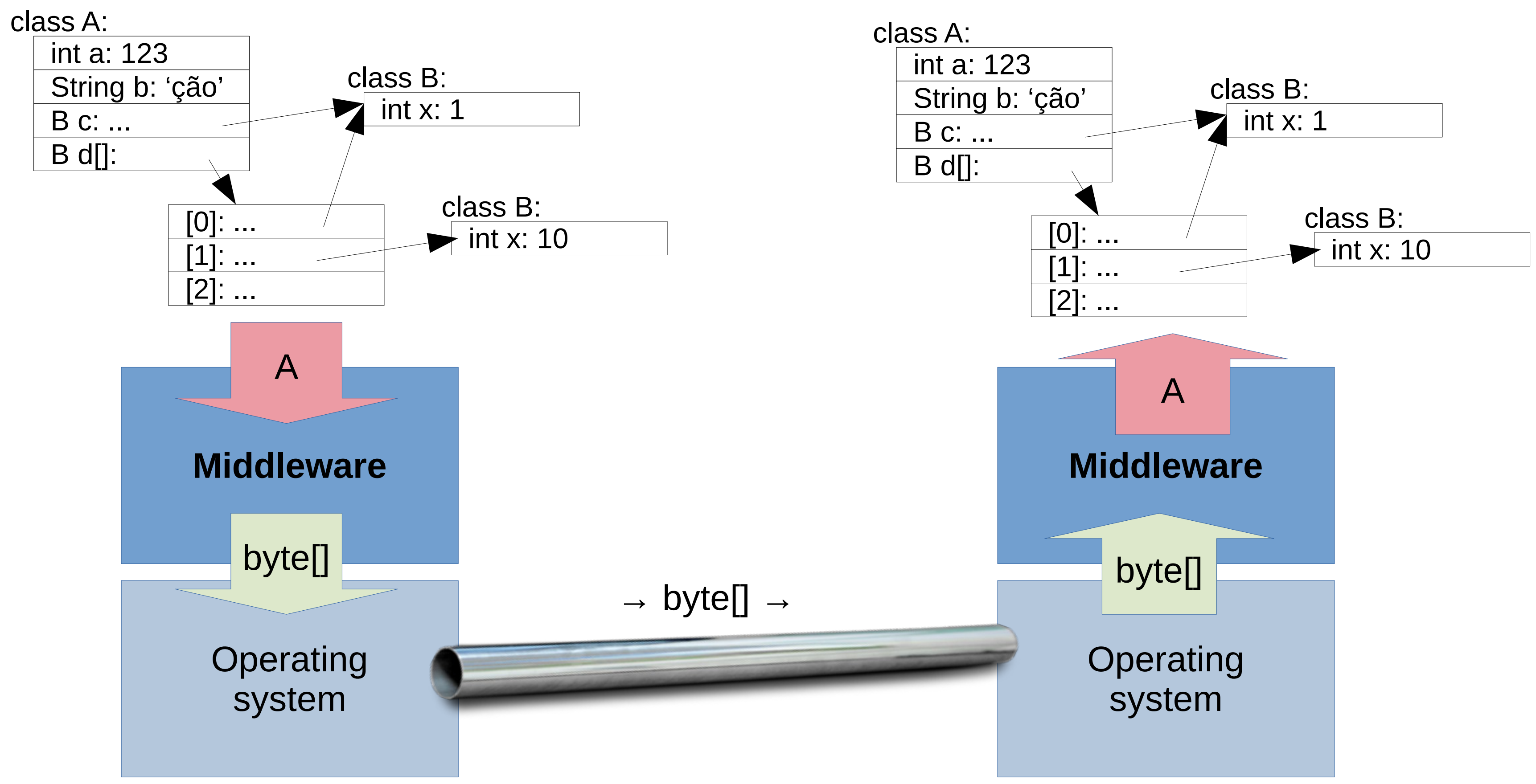
Sistemas Distribuídos

José Orlando Pereira

Departamento de Informática
Universidade do Minho

Serialization / Marshaling



Motivation

- Abstraction:
  - Messages as general purpose data structures
- Heterogeneity:
  - In space:
    - Different hardware
    - Different language / platform
    - Different middleware
  - In time:
    - Evolution of middleware and different versions co-existing in the same system

Roadmap

- Representation of basic data types
- Representation of composite data types
- Conversion code

Design issue: Text vs Binary

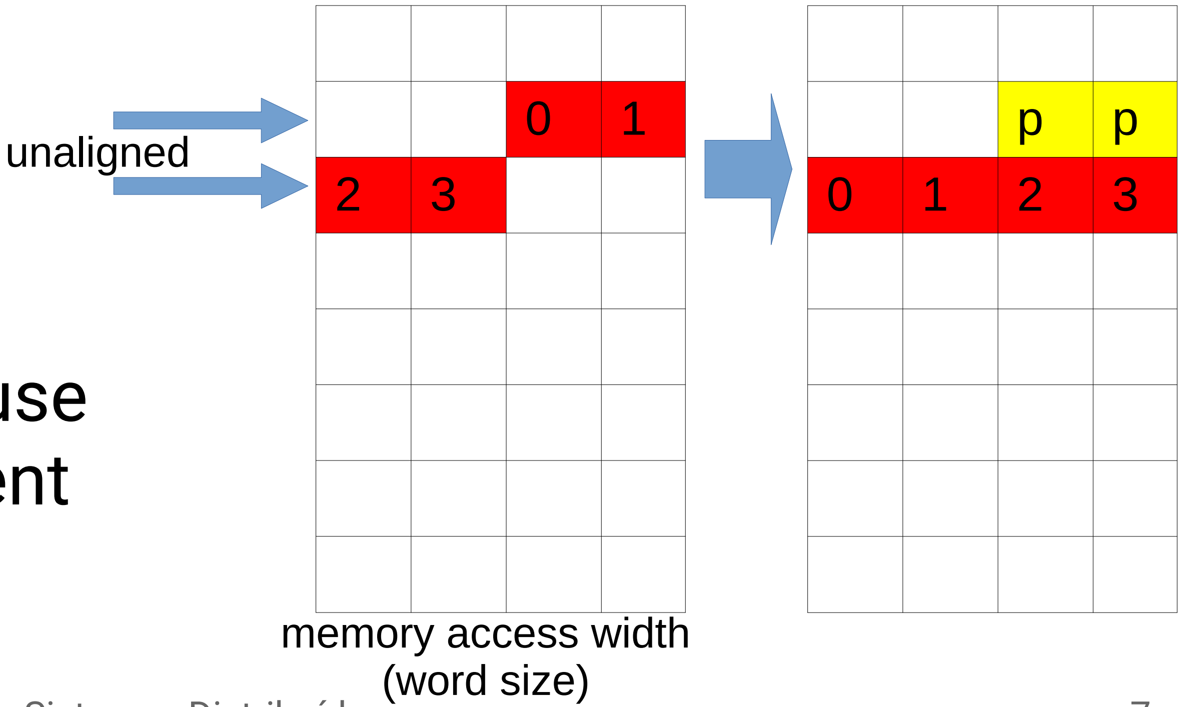
- Text formats:
  - Human readable and robust
  - Redundant and slower to parse
  - Examples: plain text, HTTP1.x, JSON
    - https://json.org/example.html
- Binary formats:
  - Compact and efficient
  - Opaque (harder to debug) and brittle
  - Examples: Java Data*Stream streams

Binary formats: Representation

- Endianess
  - An integer: 3735928559 / 0xDEADBEEF
  - Big endian bytes: { 0xDE, 0xAD, 0xBE, 0xEF }
  - Little endian bytes: { 0xEF, 0xBE, 0xAD, 0xDE }
- Character encoding
  - A string: “ção”
  - UTF8: { 0xC3, 0xA7, 0xC3, 0xA3, 0x6F }
  - Latin1: { 0xE3, 0xE7, 0x6F }

Binary formats: Alignement and padding

- Memory is addressed at byte offsets
- But accessed as multi-byte words
- Unaligned accesses are:
  - Slower; or
  - not allowed
- Binary formats may use <u>padding</u> for alignement



Binary formats: Example

- Example in Java:
  - java.io.DataOutput/DataInput

```
os.writeInt(123);
os.writeUTF(“ção”);
…
```

- Uses a common representation
  - Big endian
  - Modified UTF8 strings
  - IEEE standard floating point

Design issue: Implicit vs Explicit

- Explicit formats describe their own content (types and/or item names):

```
<file>
    <format>mp3</format>
    <tags><tag>jazz</tag><tag>modern</tag></tags>
    <size>5443236</size>
</file>
```

- Implicit formats depend on custom code to read them

```
mp3\0\0x2jazz\0modern\0\0x31\0x24...
```

Design issue: Single or Multiple

- Agree on a common representation (aka “network byte order”):
  - Convert when sending
  - Convert when receiving
    - Even if sender and receiver are identical!
- Use sender representation:
  - Send with a tag
  - Depending on tag, convert when receiving

Composite types

- Non-contiguous in memory
  - Lists, trees, …
- Contain additional information, not needed or meaningless over the network
  - Pointers, locks, ...

Composite types

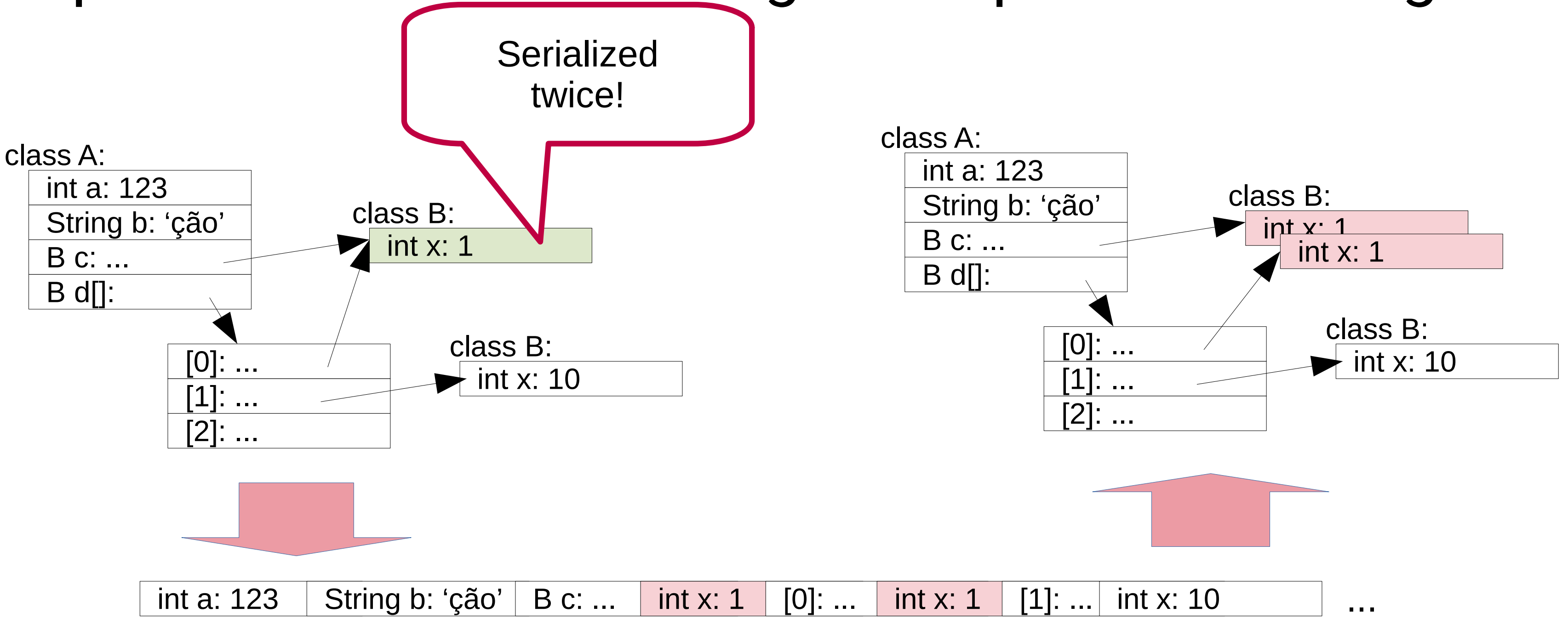
- Records
  - Enumerate each of the components
  - Include optional padding
- Objects (with subclassing) and unions:
  - Prefix content with a tag that identifies the actual option used
  - Use the tag to determine what to deserialize
- Optional items (e.g., nullable fields)
  - Prefix with a boolean indicating if present

Composite types: Collections

- Arrays, lists, sets, and maps
- First option:
  - Prefix with the number of components, then each of the components
  - Common in binary representations
  - Better if the size can be determined easily
- Second option:
  - Each of the components, then a special terminator value
  - Common in text-based representations
  - Better if the data structure can grow dynamically

Composite types: Graphs

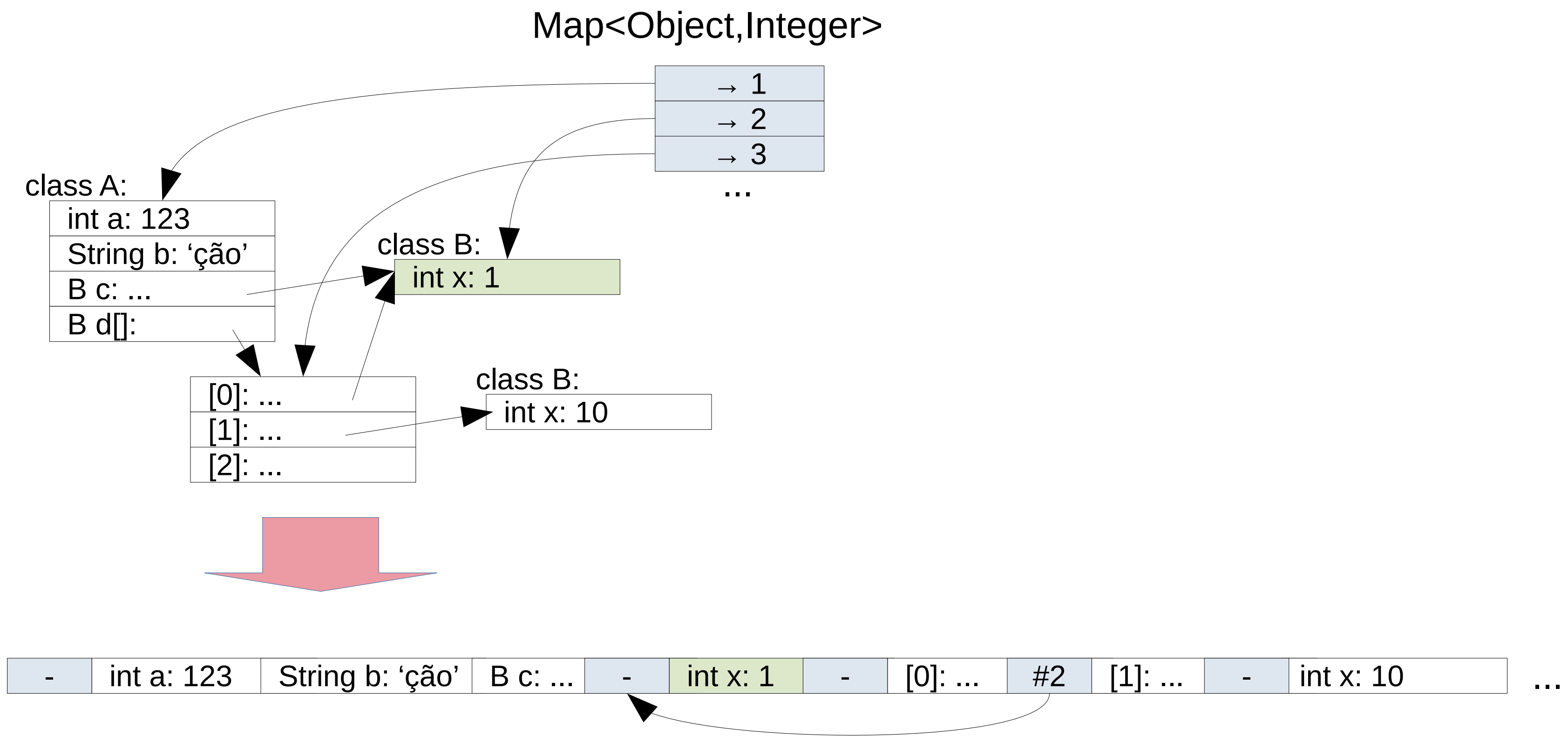
- Simple traversal is not enough with pointer aliasing:



- In fact, with cycles, simple traversal does not terminate (and generates unbounded data...)

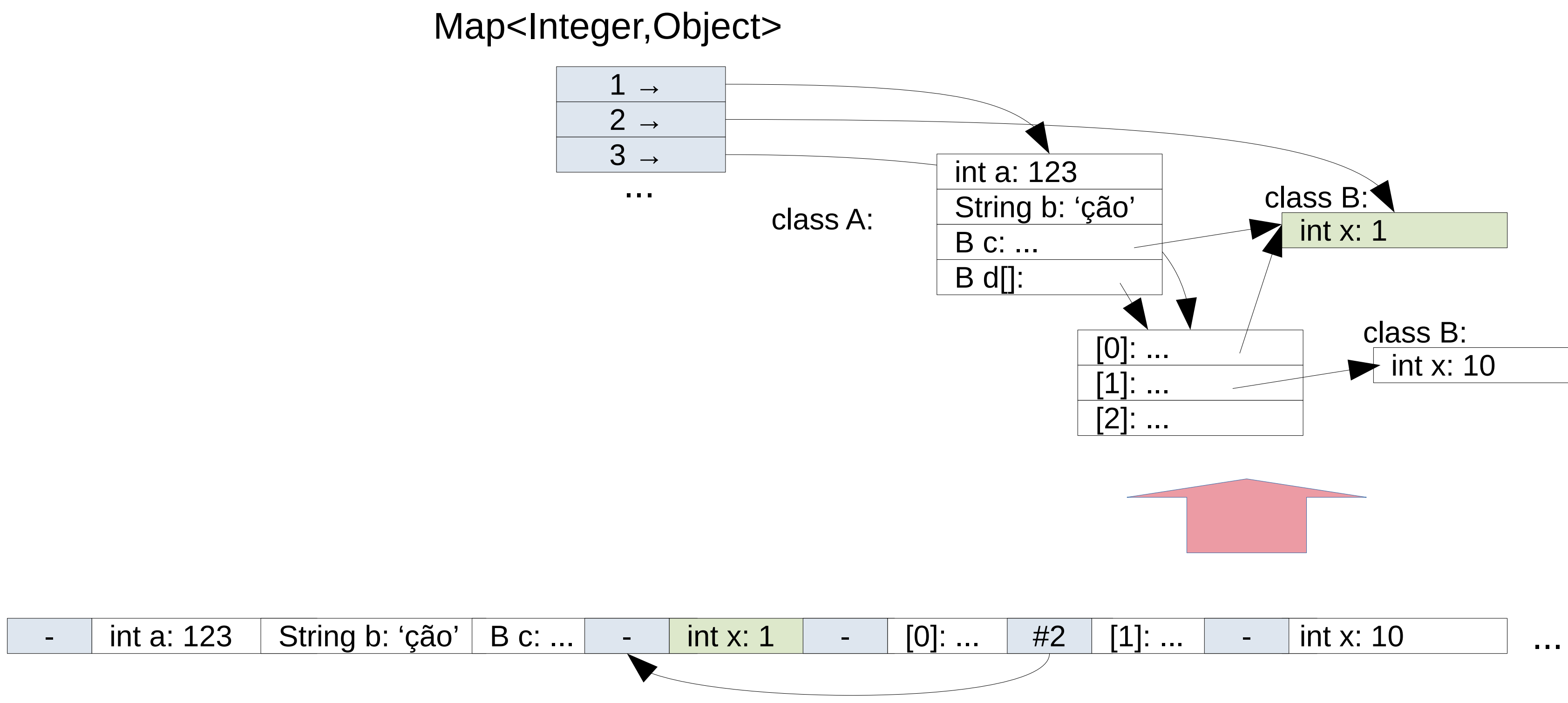
Composite types: Graphs

- Use tags and an auxiliary map while serializing:



Composite types: Graphs

- When deserializing, keep track of objects to restore pointers:



Composite types and graphs

- Example in Java:
  - java.io.ObjectOutput/ObjectInput
  - Uses DataOutput/DataInput for basic types
  - Recreates object graphs
- Very inefficient...

Design issue: Program vs Data first

- Program first: Data format inferred from an existing program
  - Convenient for development
  - Tied to a single language
  - Example: Java Object*Stream
- Data first: Program generated from an abstract description of data
  - Language and middleware independence
  - Forces developer to use new tools
  - Example: Protobuf

Conversion code

- Enumerate components to be written / read by recursively traversing data structures
- Filter methods
  - Tedious and error-prone
- Reflection
  - Allow overriding for transient data (Locks, caches, ...)
- Generated code
  - Interface language and compiler

Design issue: Versioning

- If the data structure changes, the receiver will break / decode corrupt data
- Allow a data structure to be modified with new versions of the program
- Make individual items optional and/or provide defaults
  - Example: Protobuf
- Allow versioning of the data structure as a whole
  - Example: Java Object*Stream

Design issue: Streaming vs Object model

- Streaming:
  - Data directly copied from / to external representation
  - Internal and external layouts exactly the same
  - Better efficiency: data copied only once
- Object model:
  - An intermediate model is built in memory
  - The model can be queried and traversed
  - Some applications might actually use the intermediate model directly

Common alternatives

- Simple text lines
- **Java Data*Stream**
- Java Serialization (Object*Stream)
- Kryo
- Protobuf
- JSON
- XML

Summary

- Heterogeneity as a key property of distributed systems
- Standard data representation and serialization/marshaling are needed to address it
- Tedious and error prone programming task that should be automated
- Key issue in access transparency