arnaud.nauwynck@gmail.com

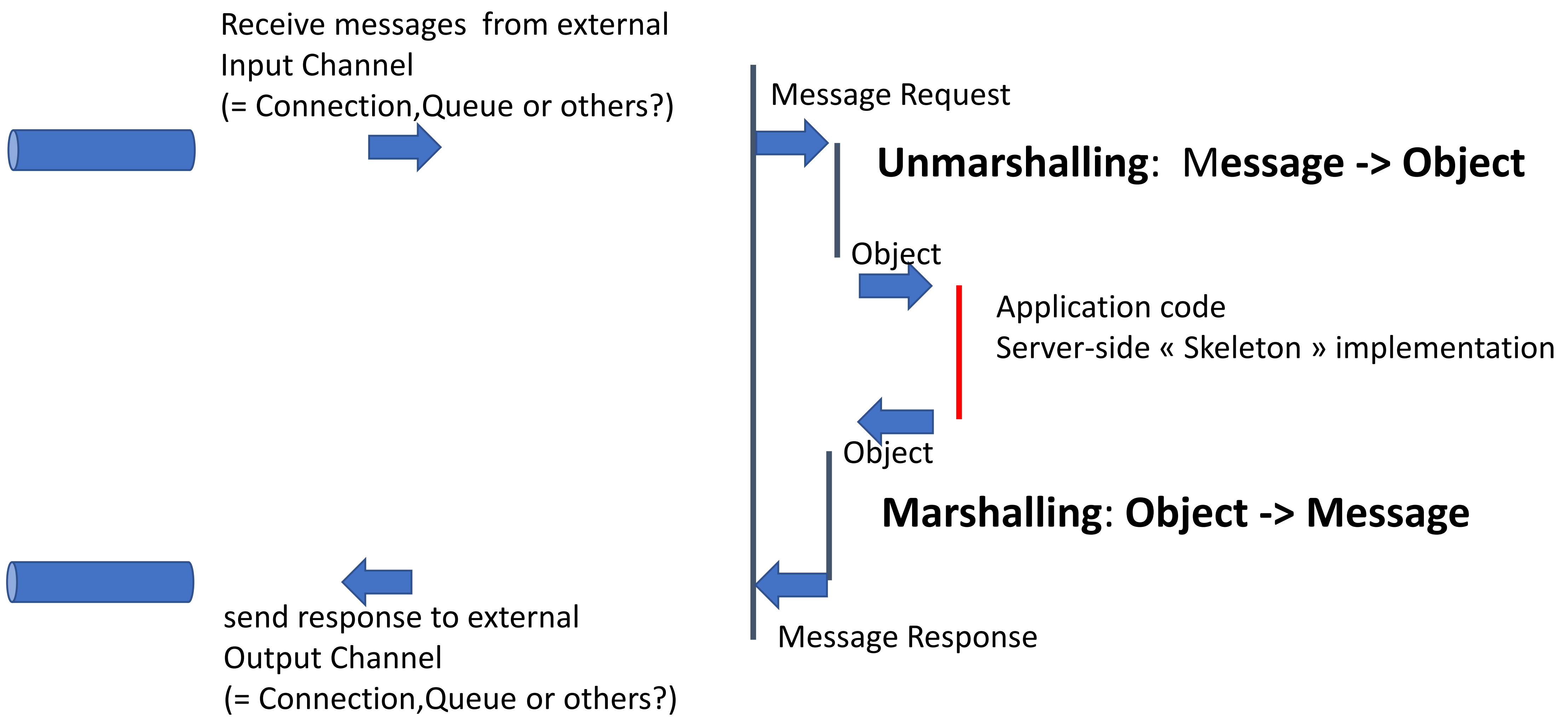
Architecture Design

Part 3 : DTO

Entity – DTO, external APIs, Protocol Marshalling

This document:
http://github.com/arnaud-nauwynck/Presentations/java/
Architecture-Design-part3-DTO.pdf

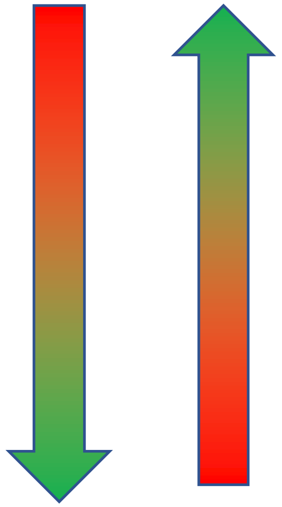
External Protocol
… Marshalling/Unmarshalling to Internal Langage


Receive messages from external
Input Channel
(= Connection,Queue or others?)
Message Request
Unmarshalling: Message -> Object
Object
Application code
Server-side « Skeleton » implementation
Object
Marshalling: Object -> Message
Message Response
send response to external
Output Channel
(= Connection,Queue or others?)

Message Encoding : Text / Binary

explained

verbose



compressed

opaque

```
<xml? schema:=« ..xsd » namespace=« ns1:..... » >
        <a> <ns1:b> some value</ns1:b> </a>
```

+++ most powerfull
+++ Self-explained / extensible
--- most verbose

```
<xml?>   <a> <b> some value</b> </a>
```

```
JSON: { « a »: { « b »: « some value » } }
```

Simplest
Compromise for Web

```
CSV:   a.b; \n
       some value
```

```
Binary: 0101010101101010101010
```

Obscure
+++ compressed encoding
+++ efficient int32, long64, float32, ..
CPU representation

Schema … Code-Generator

Text
(No code generator)

- <?xml> ← xsd schema
- { json} ← ?? No standard (json-schema,OpenApi,Swagger)
- CSV ← Header line … Tabular, Not tree

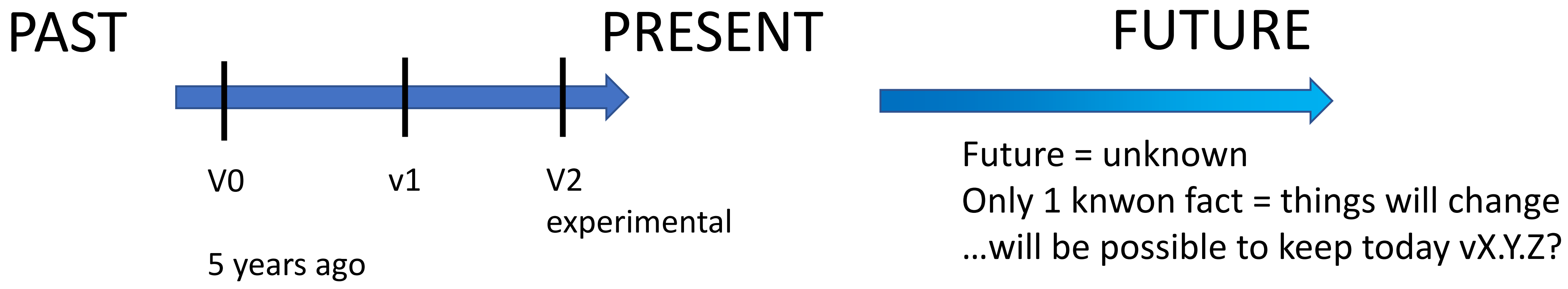
Binary
+ code generator
per-langage

- Corba,RPC,XRD,.. ← .idl
- Avro ← .avsc : Avro-Schema (in json)
- Thrift ← .idl : Thrift Schema
- Protobuf,gRPC ← .proto : Protobuf Schema

Schema: Fixed / support version upgrade

Backward-compatibility is mandatory

Still evolutivity possible ? Better if true



Future = unknown
Only 1 knwon fact = things will change
…will be possible to keep today vX.Y.Z?

Dynamic Schema: Protobuf, gRPC compiled Versioned Fields

In schema, all field are numbered (unique ID)

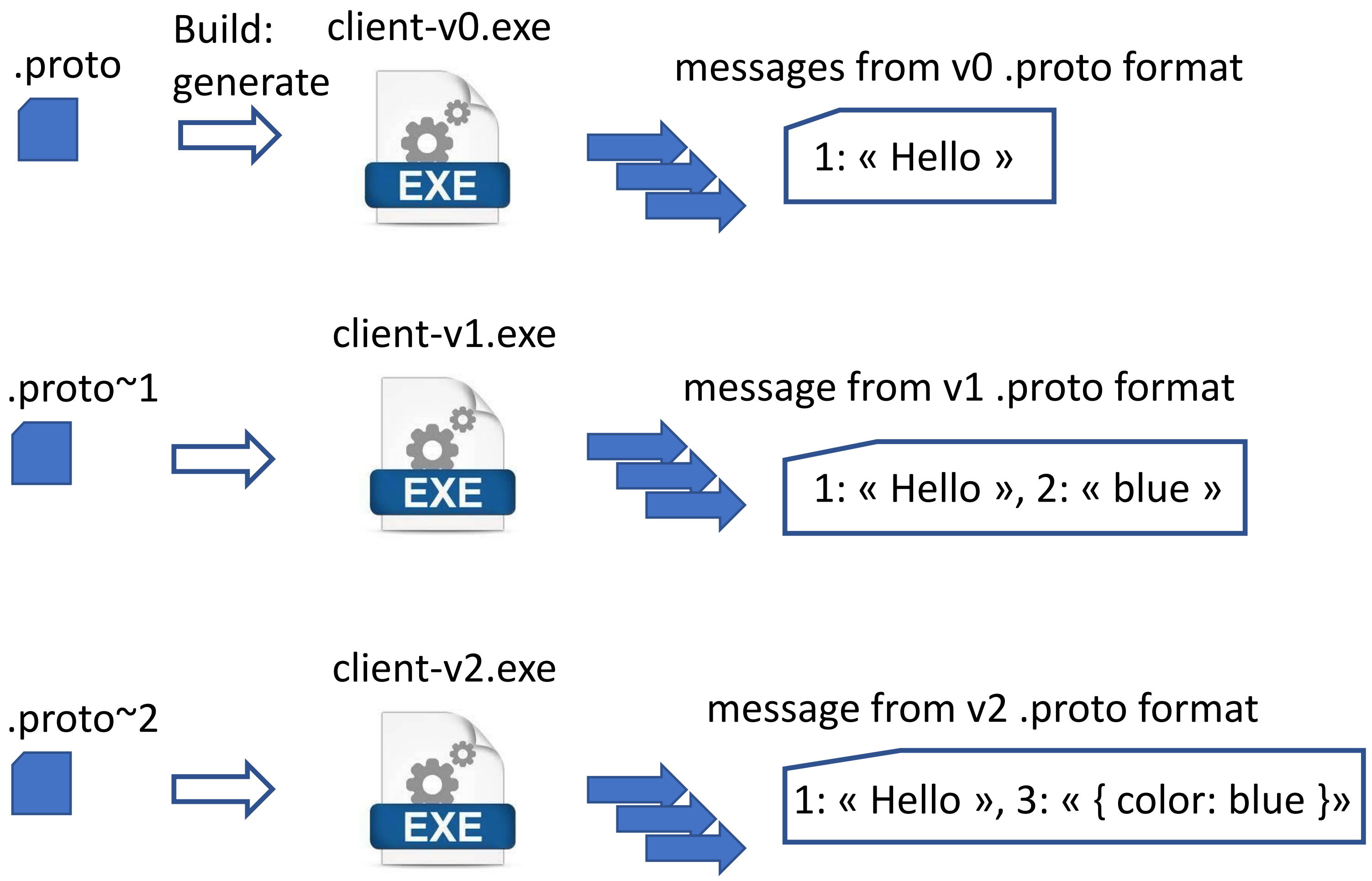
```
message HelloRequest {
  string name = 1;
}


message HelloRequest {
  string name = 1;
  string color = 2; // added in version 2
}

message HelloRequest {
  string name = 1;
  string color = 2; // deprecated, unused in version >=3 .. replaced by cssStyle
  string cssStyle = 3;
}
```

Dynamic Messages, Schema-compatible

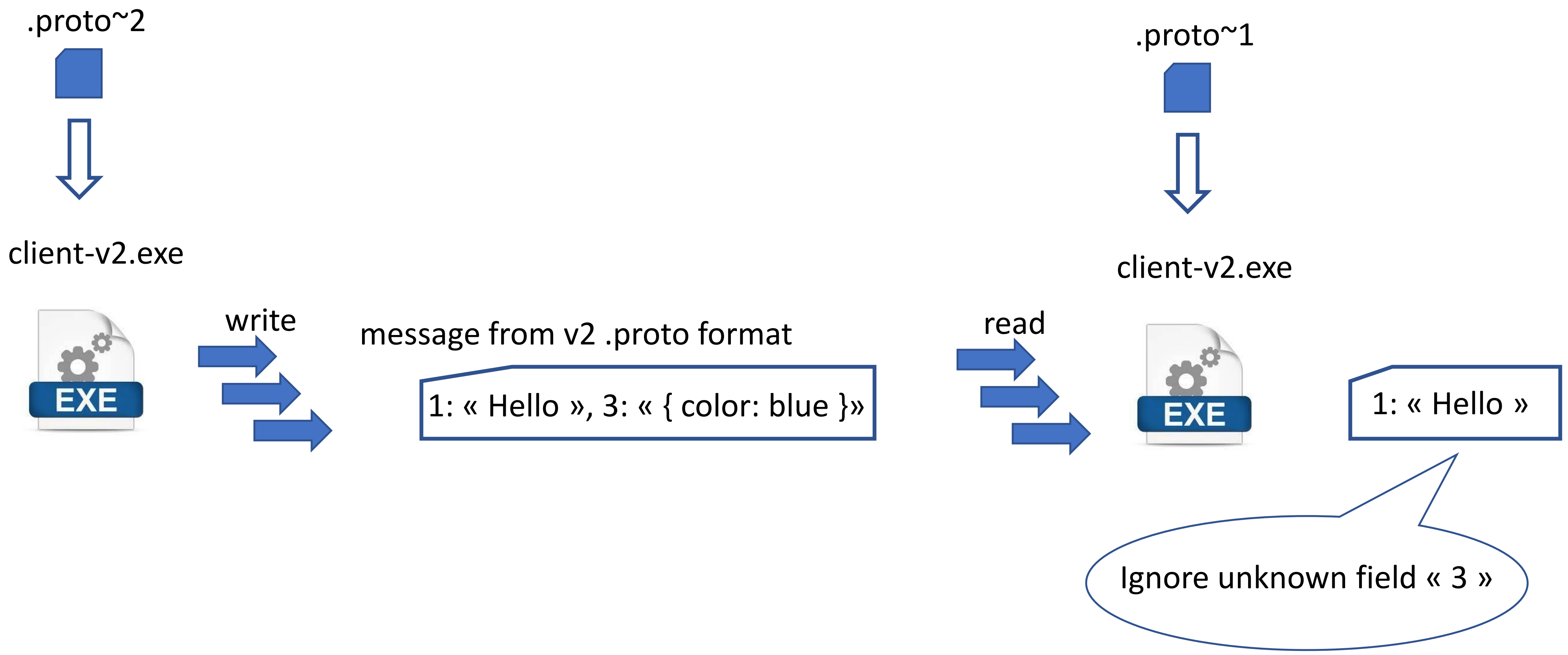
In Messages
« Map<FieldVersion, Value> »

all Field ids + values encoded

Small extra cost
… Great compatibility

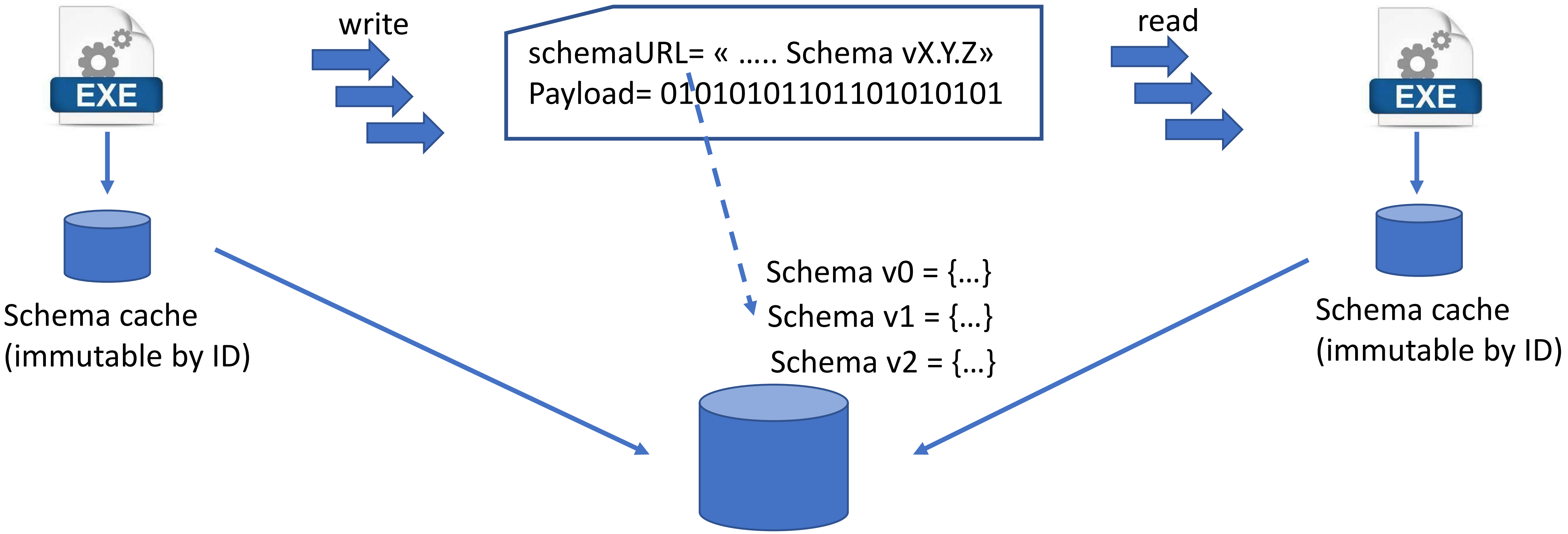

.proto
Build:
generate
client-v0.exe
EXE
messages from v0 .proto format
1: « Hello »
.proto~1
client-v1.exe
EXE
message from v1 .proto format
1: « Hello », 2: « blue »
.proto~2
client-v2.exe
EXE
message from v2 .proto format
1: « Hello », 3: « { color: blue }»

Reading « Future » Message / Backward Client … Ignore Unknown Fields

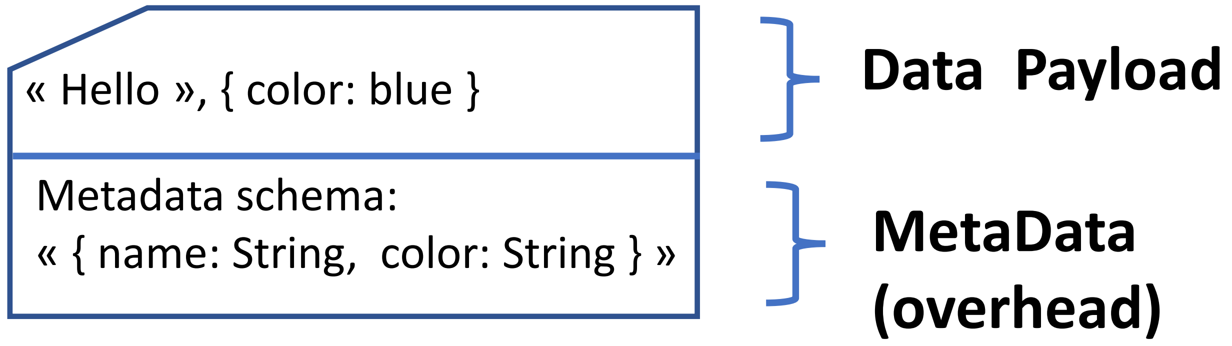


.proto~2
client-v2.exe
EXE
write
message from v2 .proto format
1: « Hello », 3: « { color: blue }»
read
.proto~1
client-v2.exe
EXE
1: « Hello »
Ignore unknown field « 3 »

Schema Registry



Schema Contained in Messages


« Hello », { color: blue }
Metadata schema:
« { name: String, color: String } »
Data Payload
MetaData
(overhead)

Examples :

Kafka Messages, Pulsar Messages

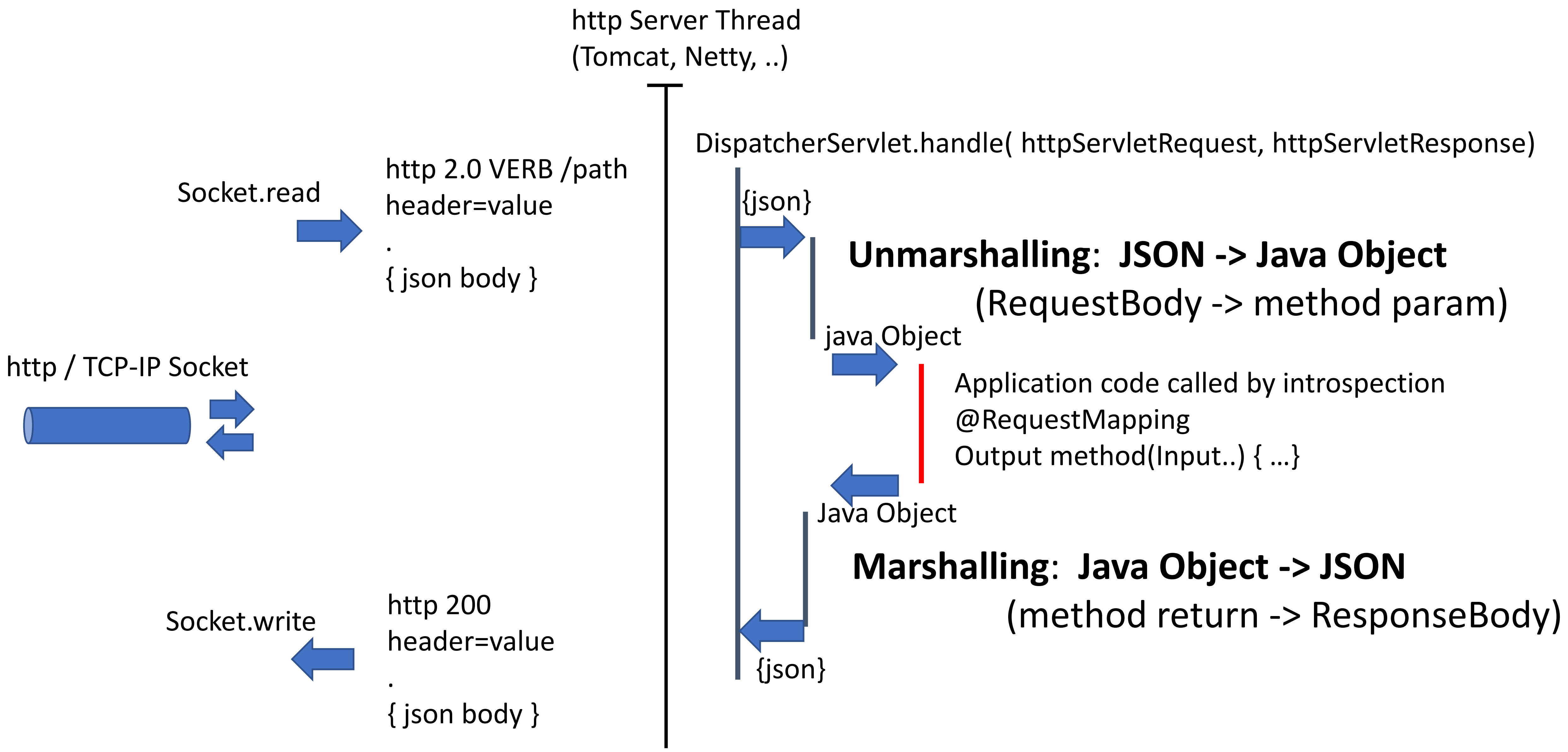
Avro Message, Avro Data-File

PARQUET File — for ULTRA compression of millions of rows (dictionnary, incremental, filter..)

java.io.Serializable — Contains serialVersionUID + fully-qualified Names + field name / types

Use « Kryo » instead of « java.io. » for typed / schema-less messages !
(better performances)

Http Json Request <-> Java Object method


http Server Thread
(Tomcat, Netty, ..)
DispatcherServlet.handle(httpServletRequest, httpServletResponse)
Socket.read
http 2.0 VERB /path
header=value
.
{ json body }
{json}
Unmarshalling: **JSON -> Java Object**
(RequestBody -> method param)
java Object
http / TCP-IP Socket
Application code called by introspection
@RequestMapping
Output method(Input..) { …}
Java Object
Marshalling: **Java Object -> JSON**
(method return -> ResponseBody)
Socket.write
http 200
header=value
.
{ json body }
{json}

JavaScript <-> Json (JavaScript Object Notation) <-> Java

Script (untyped interpreter)

```
let object = {
    name: 'arnaud',
    skills: [ 'IT', 'math']
};
console.log(`Hello ${object.name}`);
```

DATA format (no schema)

```
{
    "name": "arnaud",
    "skills": [ "IT", "math"]
}
```

Langage (typed)

```
public class User {
    public String name;
    public List<String> skills;
}
```

TypeScript

JavaScript

{JSON}

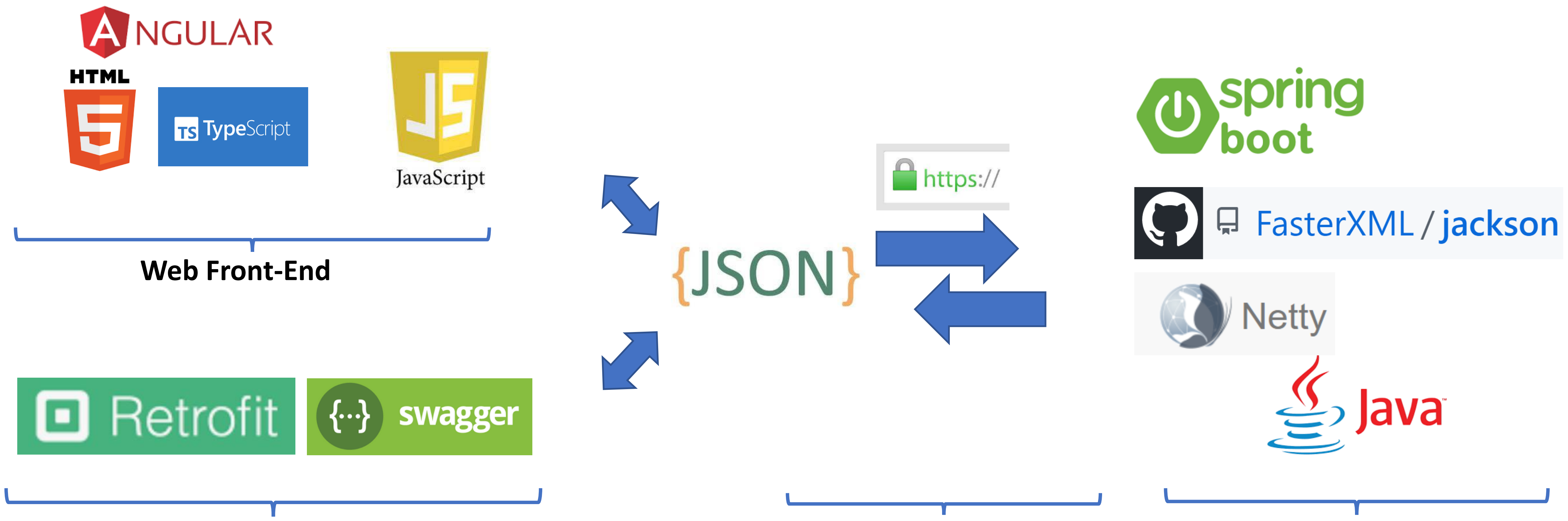
Java

Web Front-End

Network encoding

Back-end

http JSON : Open & DeFacto Standard for Portability, Simplicity, Frameworks



Web Front-End

API Front-End (app-to-app)

Network
connection+encoding

Back-end

Http Request <-> Spring Java Mappings @RequestMapping, @{Get|Post|..}Mapping, @RequestBody ..

```java
@PostMapping
public TodoDTO postTodo(
        @RequestBody TodoDTO req // => from outside, spring dispatcher...
                                 // request body as json text, is converted to java Object using Jackson
        ) {
    Log.info("http POST /api/todo");
    TodoDTO res = service.createTodo(req);
    return res;
}



@GetMapping("/{id}")
public TodoDTO get(@PathVariable("id") int id) {
    TodoDTO res = service.get(id);
    return res;
} // => outside, spring dispatcher... return java Object is converted to json using Jackson
```

Equivalent Explicit Json Unmarshalling

```
@PostMapping
public TodoDTO postTodo(
        @RequestBody TodoDTO req) {
    log.info("http POST /api/todo");
    TodoDTO res = service.createTodo(req);
    return res;
}

@Autowired
ObjectMapper jsonMapper;

// equivalent
@PostMapping(consumes = "application/json")
public TodoDTO postTodo2(
        @RequestBody byte[] reqBodyContent) throws Exception {
    log.info("http POST /api/todo");
    TodoDTO req = jsonMapper.readValue(reqBodyContent, TodoDTO.class);
    TodoDTO res = service.createTodo(req);
    return res;
}
```

Equivalent Explicit Json Marshalling

```
@GetMapping("/{id}")
public TodoDTO get(@PathVariable("id") int id) {
    return service.get(id);
}
```

```
@Autowired
ObjectMapper jsonMapper;

// implicit equivalent..
@GetMapping(path = "/equivalent1/{id}",
        produces = "application/json")
public byte[] get1(@PathVariable("id") int id) throws JsonProcessingException {
    TodoDTO res = service.get(id);
    return jsonMapper.writeValueAsBytes(res);
}
```

Explicit Equivalent, with http Status + Headers

```java
// implicit equivalent.. with extra header
@GetMapping(path = "/equivalent2/{id}",
        produces = "application/json")
public ResponseEntity<byte[]> get2(@PathVariable("id") int id) throws JsonProcessingException {
    TodoDTO res = service.get(id);
    byte[] content = jsonMapper.writeValueAsBytes(res);
    return ResponseEntity.status(HttpStatus.OK)
            .header("some-response-header", "value")
            .body(content);
}

// implicit equivalent.. with extra header
@GetMapping(path = "/equivalent2/{id}",
        produces = "application/json")
public void get2(@PathVariable("id") int id,
        HttpServletResponse serlvetResponse
        ) throws IOException {
    TodoDTO res = service.get(id);
    byte[] content = jsonMapper.writeValueAsBytes(res);
    serlvetResponse.setStatus(200);
    serlvetResponse.addHeader("some-response-header", "value");
    serlvetResponse.getOutputStream().write(content);
}
```

Naive (Not working) @Entity to JSON

```
@RestController
@RequestMapping("/api/not-working/todo")
@Transactional
public class NaiveStupidBuggedRestController {

    @Autowired
    private TodoRepository repository;

    @GetMapping("/{id}")
    public TodoEntity get(@PathVariable("id") int id) {
        TodoEntity entity = repository.getById(id);
        return entity;
    }
}
```

Entity class maybe
not JSON compatible

Entity object invalid
After transaction commit
(managed lifecycle)

Why Not using 1 class Entity = DTO ?

1/ internal database is PRIVATE, implementation specific
    != external API is publicly specified

2/ Decoupling helps evolutivity

3/ Mandatory for complex Entities graph
   with cyclic dependencies parent-child (@ManyToOne(mappedBy..))

4/ several DTO classes/APIs for a single Entity class

5/ Do not use HACKS like @JsonIgnore… (see 1/, 2/, 3/, 4/)

Example Entity class .. Do not export all

```
@Entity
@Getter @Setter
public class UserEntity {

    @Id
    private String email;

    private String firstName;
    private String lastName;

    @Column(unique = true)
    private String pseudo;

    private String password;

    private byte[] photo;

}
```

Personnal data
Do not publish

Critical Security data
never publish !!
(Except for owner)

High volume data
export only explicitly demanded

1 Entity class <-> Several specific DTO classes

```java
@Entity
@Getter @Setter
public class UserEntity {

    @Id
    private String email;

    private String firstName;
    private String lastName;

    @Column(unique = true)
    private String pseudo;

    private String password;

    private byte[] photo;

}
```

Default mapping

```java
@Data
public class UserLightDTO {

    public String firstName;
    public String lastName;
    public String pseudo;

}
```

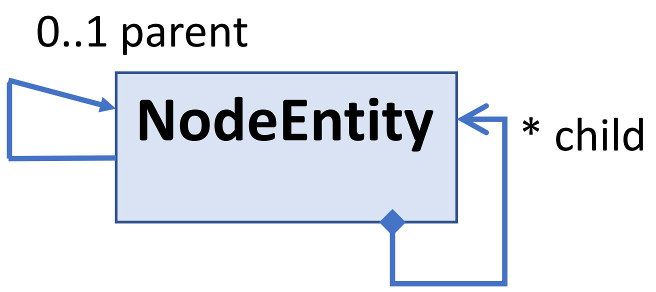
Owner personal view

```java
@Data
public class SecuredUserDetailDTO {
    public String email;
    public String firstName;
    public String lastName;
    public String pseudo;

    // computed from group,settings, ...
    public List<String> grantedPermissions;
}
```

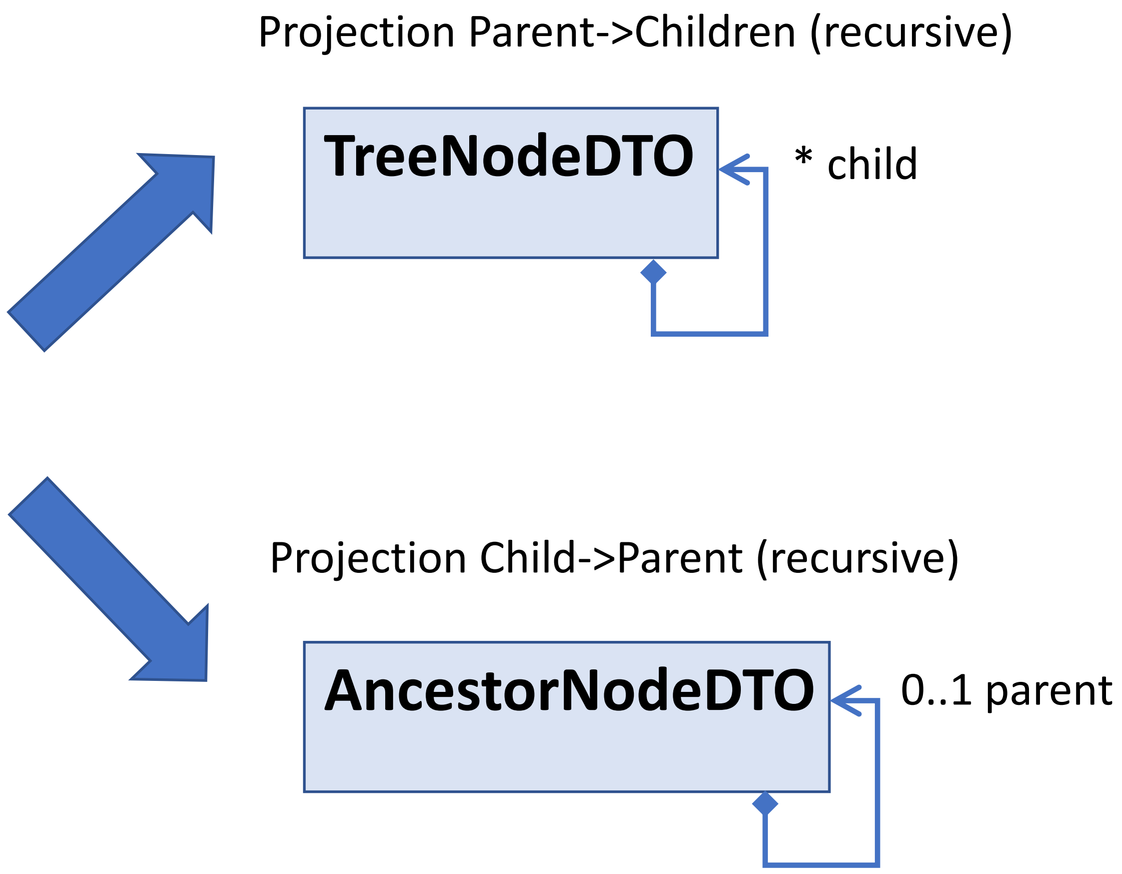
Public detailed view (high data volume)

```java
@Data
public class SecuredUserDetailDTO {
    public String pseudo;
    public byte[] photo;
}
```

Entity -> Projection DTOs = « Cutting » relations

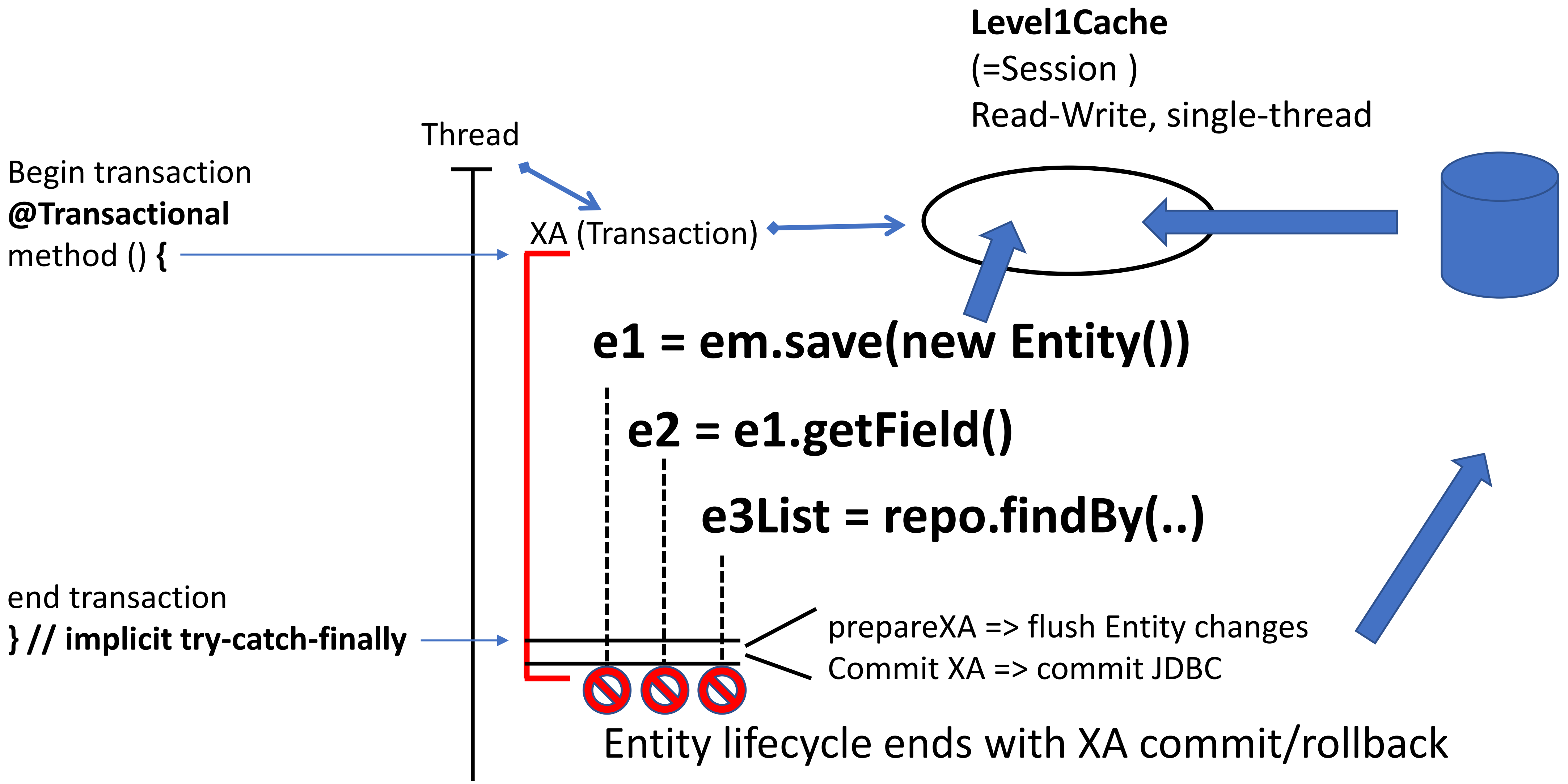


```java
@Entity
@Getter @Setter
public class NodeEntity {

    @Id
    private String path;

    // parent -> child relationship
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "parent")
    private List<NodeEntity> child;

    // child -> parent (inverse) relationship
    @ManyToOne(fetch = FetchType.LAZY)
    private NodeEntity parent;

}
```

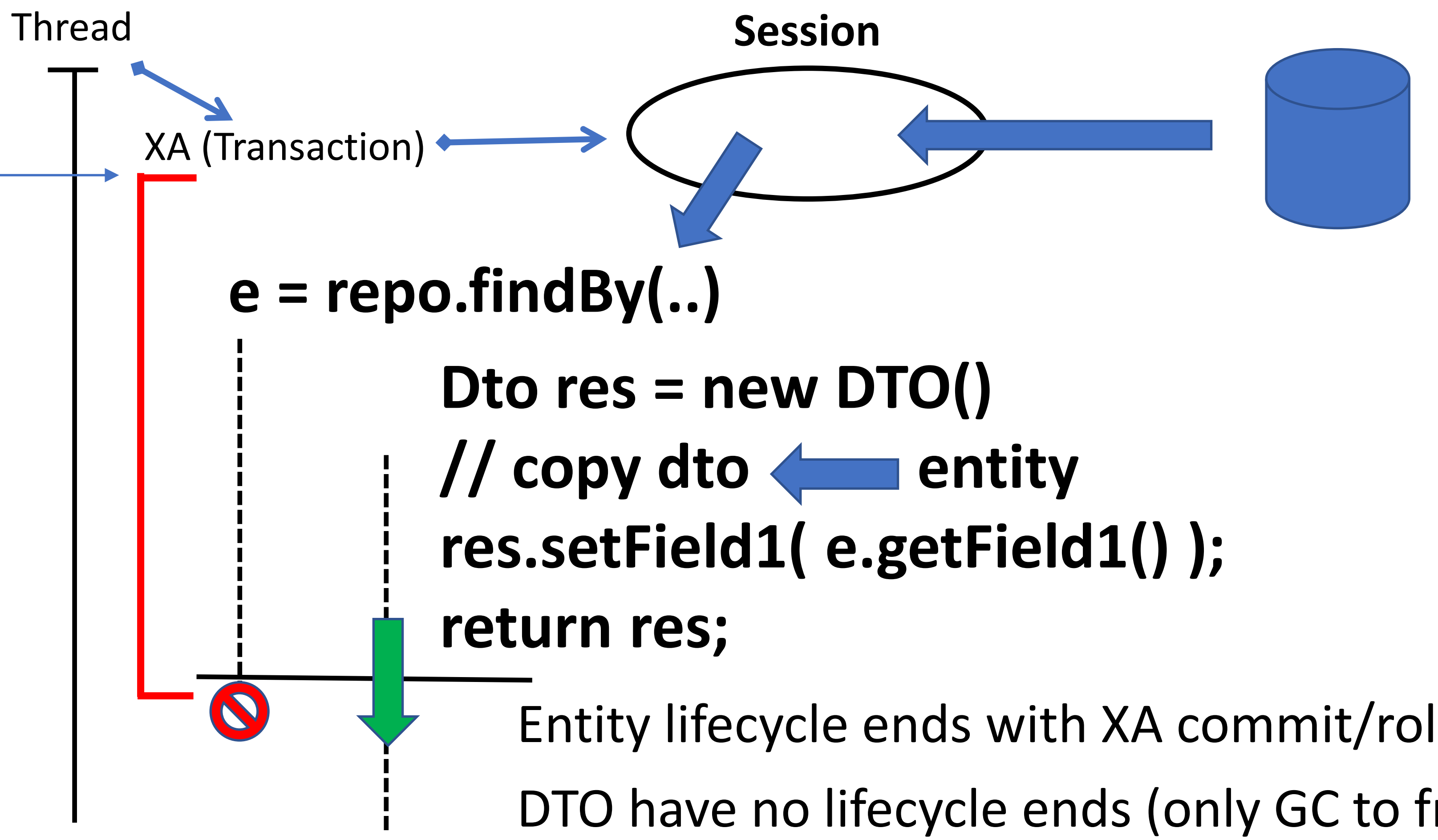

Projection Parent->Children (recursive)
TreeNodeDTO
* child
Projection Child->Parent (recursive)
AncestorNodeDTO
0..1 parent

Reminder Part 2 … Entity lifecycle in Session

Entity : Lifecycle managed by Session (Transaction)


Level1Cache
(=Session)
Read-Write, single-thread
Thread
Begin transaction
@Transactional
method () **{**
XA (Transaction)
e1 = em.save(new Entity())
e2 = e1.getField()
e3List = repo.findBy(..)
end transaction
} // implicit try-catch-finally
prepareXA => flush Entity changes
Commit XA => commit JDBC
Entity lifecycle ends with XA commit/rollback

Copy Entity data to Transfer before Commit



Begin transaction
@Transactional
method () **{**

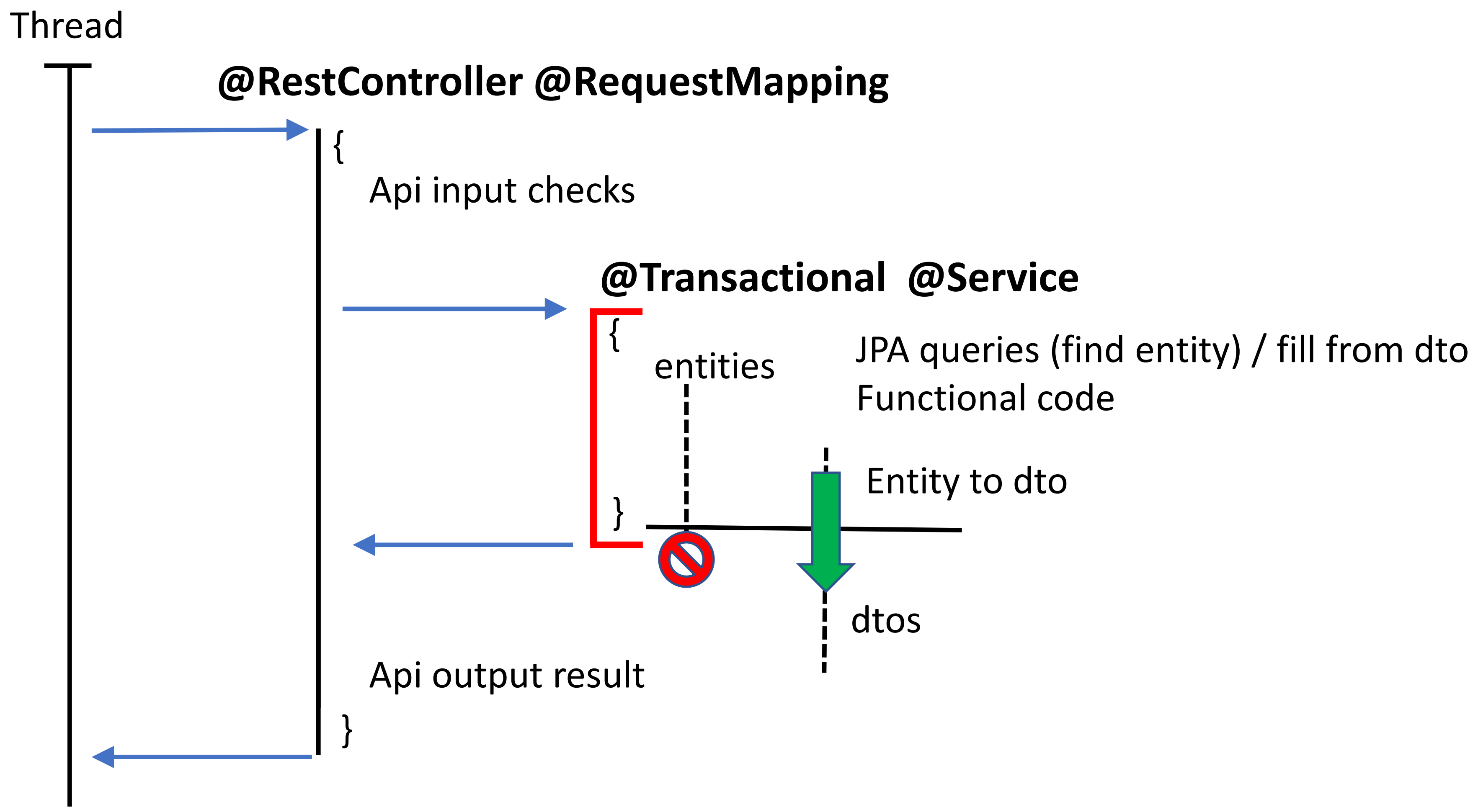
```
e = repo.findBy(..)
        Dto res = new DTO()
        // copy dto ⇐ entity
        res.setField1( e.getField1() );
        return res;
```

end transaction
} // implicit try-catch-finally

Entity lifecycle ends with XA commit/rollback

DTO have no lifecycle ends (only GC to free)

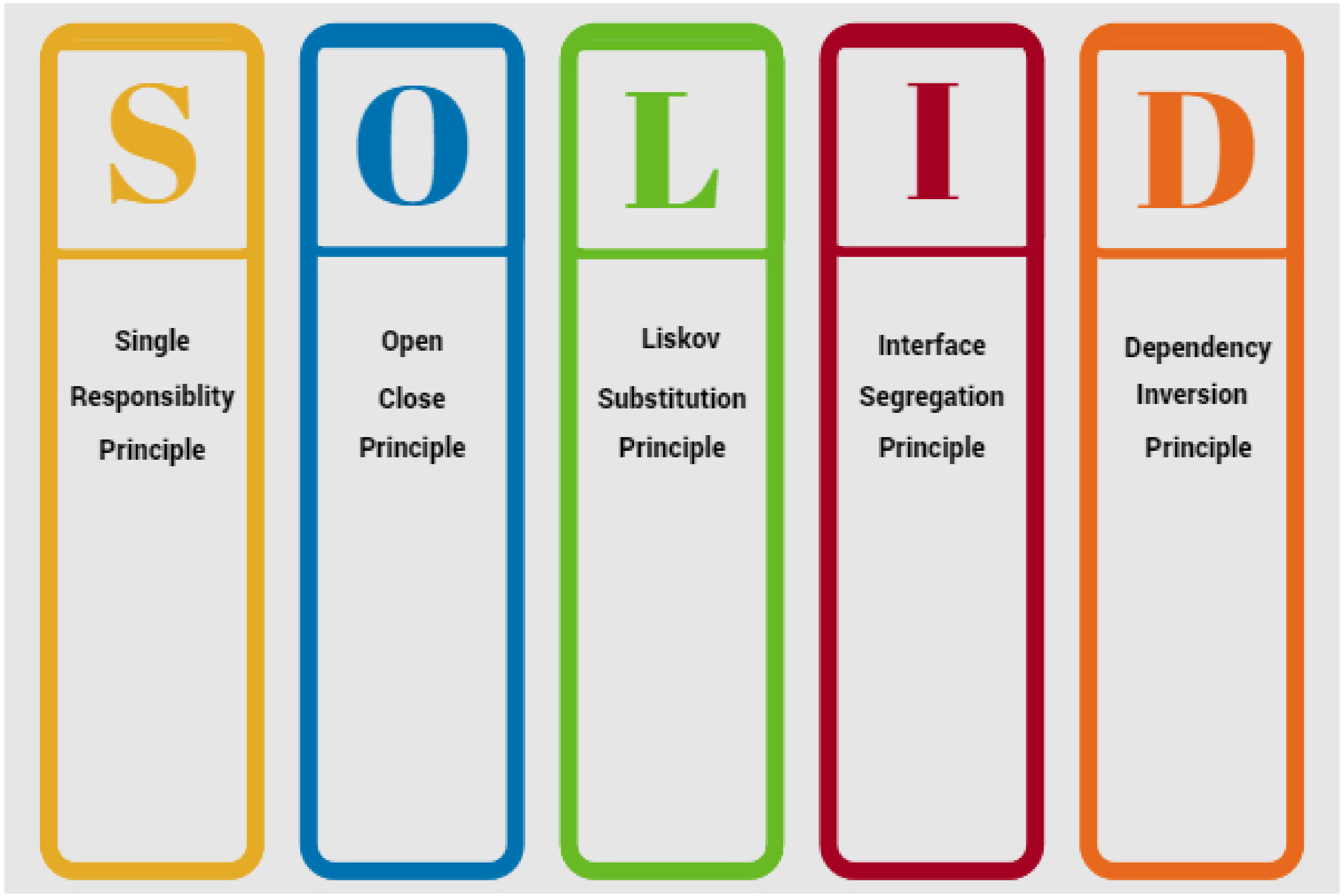
2 Rules : a/ use DTOs != Entities b/ use RestController != (Transactional) Service



Controller Method Pattern
{ unmarshal / delegate / marshall }

```
@PostMapping
public ResponseDTO postTodo(
        @RequestBody TodoDTO req) {
    // step 1/3: unmarshall, check inputs, convert, logs...
    long start = System.currentTimeMillis();
    log.info("http POST /api/todo");

    // step 2/3: delegate to service
    TodoDTO res = service.createTodo(req);

    // step 3/3: convert, format output, logs...
    log.info(".. done http POST, took " + (System.currentTimeMillis()-start) + " ms");
    return new ResponseDTO(res.id, res.label);
}
```

« solid » principles


S
O
L
I
D
Single Responsiblity Principle
Open Close Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle

SOLID RestController .... S = Single

A RestController does only 1 thing :

controls (maps) http Rest requests to Java methods

… delegate all others things to injected Service

Typical CRUD Rest Controller

```java
@RestController
@RequestMapping("/api/todo")
@Slf4j
public class TodoRestController {

    @Autowired
    private TodoService service;

    @GetMapping()
    public List<TodoDTO> list() {
        List<TodoDTO> res = service.list();
        return res;
    }

    @GetMapping("/{id}")
    public TodoDTO get(@PathVariable("id") int id) {
        TodoDTO res = service.get(id);
        return res;
    }

@PostMapping
public TodoDTO postTodo(@RequestBody TodoDTO req) {
    Log.info("http POST /api/todo");
    TodoDTO res = service.createTodo(req);
    return res;
}

@PutMapping
public TodoDTO putTodo(@RequestBody TodoDTO req) {
    Log.info("http PUT /api/todo");
    TodoDTO res = service.updateTodo(req);
    return res;
}

@DeleteMapping("/{id}")
public TodoDTO deleteTodo(@PathVariable("id") int id) {
    Log.info("http DELETE /api/todo");
    TodoDTO res = service.deleteTodo(id);
    return res;
}

}
```

Typical CRUD Transactional Service (1/2)

```java
@Service
@Transactional
public class TodoService {

    @Autowired
    private TodoRepository repository;

    @Autowired
    private DtoConverter dtoConverter;

    public List<TodoDTO> list() {
        List<TodoEntity> entities =
                repository.findAll();
        return entity2Dtos(entities);
    }

    public TodoDTO get(int id) {
        TodoEntity entity = repository.getById(id);
        return entity2Dto(entity);
    }
```

```java
public TodoDTO createTodo(TodoDTO req) {
    TodoEntity res = repository.save(dto2Entity(req));
    return entity2Dto(res);
}

public TodoDTO updateTodo(TodoDTO req) {
    TodoEntity entity = repository.getById(req.id);
    entity.setLabel(req.label);
    entity.setPriority(req.priority);
    return entity2Dto(entity);
}

public TodoDTO deleteTodo(int id) {
    TodoEntity entity = repository.getById(id);
    repository.delete(entity);
    return entity2Dto(entity);
}
```

Typical CRUD Service (2/2)
entity2Dto / dto2Entity

```
public TodoDTO entity2Dto(TodoEntity src) {
    TodoDTO res = new TodoDTO();
    res.id = src.getId();
    res.label = src.getLabel();
    res.priority = src.getPriority();
    // other fields...
    return res;
}

public TodoEntity dto2Entity(TodoDTO src) {
    TodoEntity res = new TodoEntity();
    res.label = src.label;
    res.priority = src.priority;
    // other fields...
    return res;
}

public List<TodoDTO> entity2Dtos(Collection<TodoEntity> src) {
    return src.stream().map(e -> entity2Dto(e)).collect(Collectors.toList());
}
```

entity2Dto / dto2Entity using generic signature and « .class »

```
protected TodoDTO entity2Dto(TodoEntity src) {
    return dtoConverter.map(src, TodoDTO.class);
}

protected List<TodoDTO> entity2Dtos(Collection<TodoEntity> src) {
    return dtoConverter.mapAsList(src, TodoDTO.class);
}

protected TodoEntity dto2Entity(TodoDTO src) {
    return dtoConverter.map(src, TodoEntity.class);
}
```

using Orika MapperFacade

```
@Component
public class DtoConverter {

    private MapperFacade mapper = createMapper();

    private MapperFacade createMapper() {
        MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();
        return mapperFactory.getMapperFacade();
    }

    public <S, D> D map(S sourceObject, Class<D> destinationClass) {
        return mapper.map(sourceObject, destinationClass);
    }

    public <S, D> List<D> mapAsList(Iterable<S> source, Class<D> destinationClass) {
        return mapper.mapAsList(source, destinationClass);
    }

}
```