V. Client/Server-Interaction

Important: pg. 42 - 55 on SOA details is an Add-On and NOT part of the core module content.

Message passing too closely coupled for modular systems $\Longrightarrow$ higher level of de-coupling needed: **Remote Procedure Calls** Nelson 1981

- **Basic Model:** simple role-based architecture
  - *Server* offers services
  - *Client* requests and uses services

  Examples: `Email`, file server, time server, name server, . . .
- **Well-known programming paradigm:** Procedure Calls at the time

  $\Longrightarrow$ simple interaction protocol
- **Moderate coupling**

  Server $\approx$ no global knowledge besides own state

  Client $\approx$ service signatures of Services (Service-Broker)

  Substitution Test: new Clients may use running Server
  new Server may offer the same/new services

Remark: really old concept `www.ietf.org/rfc/rfc707.txt` 1976

Client-Server-Interaction – Roles

Roles allow for a suitable level of **abstraction**:

- Independent from specific **kind of Request**
  - compute function: Compute server replicated
  - read/write data: File system server replicated

  $\Longrightarrow$ **Transparency w.r.t. active and passive components**
- Role may **change due to point of view**

  $\Longrightarrow$ context-sensitive roles

  $\Longrightarrow$ same object may play different roles

  Example: User-Programm reads data from file

$$\overbrace{\text{Program}}^{\text{Client}} \longleftrightarrow \underbrace{\overbrace{\textbf{file server}}^{\text{Server}}}_{\text{Client}} \longleftrightarrow \underbrace{\text{Authentification/Time Server}}_{\text{Server}}$$

Remark: C/S model is even more general than RPC interaction

C/S-Paradigm Development over 45+ Years

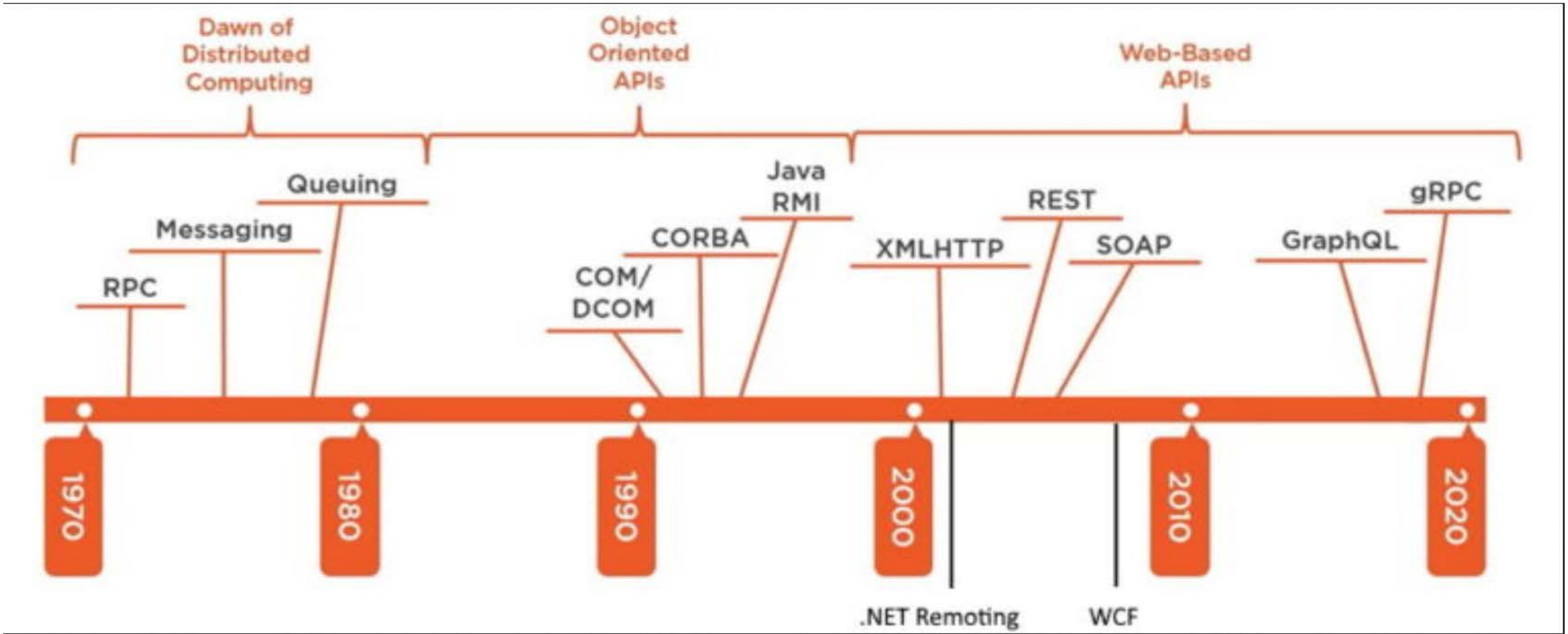


Fig.: S. Wildermuth www.pluralsight.com/courses/aspnet-core-grpc

- OS/Network: Stream I/O message exchange using `TCP`-Sockets
  $\Longrightarrow$ C/S role determines which kind of `socket` used in program

  $\ldots \Longrightarrow \ldots$
- All kinds of distributed models for **Remote Procedure Call**
  Description, Addressing, Protocols, Data Formats, Sync, ...

Different Approaches for the C/S-Paradigm

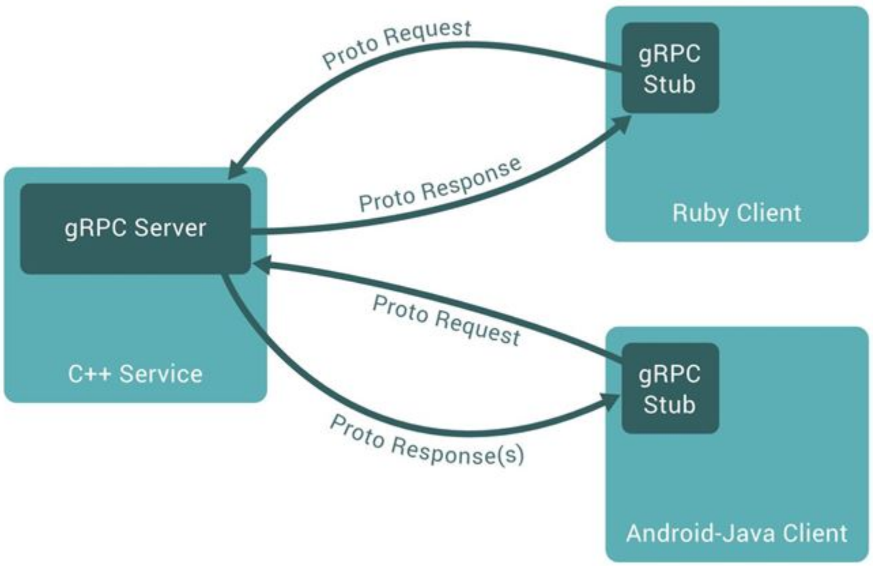
- 'Classical' **Remote Procedure Call** Envs $\Longrightarrow$ **Characteristics**
  - Distributed Computing Environment (`DCE`) approx. 1990
  - Microsoft MSRPC; COM; COM+; DCOM; ... .NET Remoting
  - **C**ommon **O**bject **R**equest **B**roker **A**rchitecture, e.g., in `C` based on C-like **I**nterface **D**efinition **L**anguage (IDL)

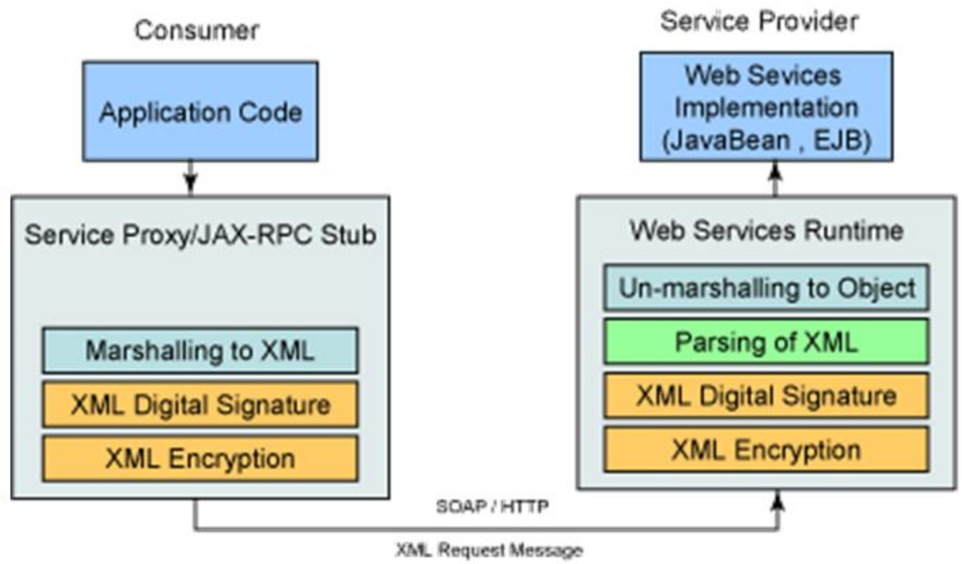
- **Object-Oriented C/S Implementations**
  - Java Remote Method Invocation (RMI)
  - Using `CORBA` and `IDL` in a non-Java OO Environment, e.g., `C++`

- **Client/Server on the Internet** – Services
  - Google g`RPC` re-inventing IDLs in `JSON` (`grpc.io`, 2015) V.3
  - Heavy-weighted `XML/SOAP`-based Web Services (SOA) (V.5)
  - Light-weighted `http`-based APIs using a `REST` Architecture (**RE**presentational **S**tate **T**ransfer; Roy Fielding 1994/2000) V.4

Preview – 1: Google RPC vs. Webservice

grpc.io.docs.guides





www.ibm.com/developerworks/library/ws-best9 index.html

- Left: gRPC setting for multi-language clients
  `protocol buffers` as Interface Definition Language — IDL
- Right: IBM Webservice using 'RPC' technology
  `XML` Ecosystem as common Interface Definition Language

Preview – 2: SOAP vs. REST APIs

Fig.: www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis

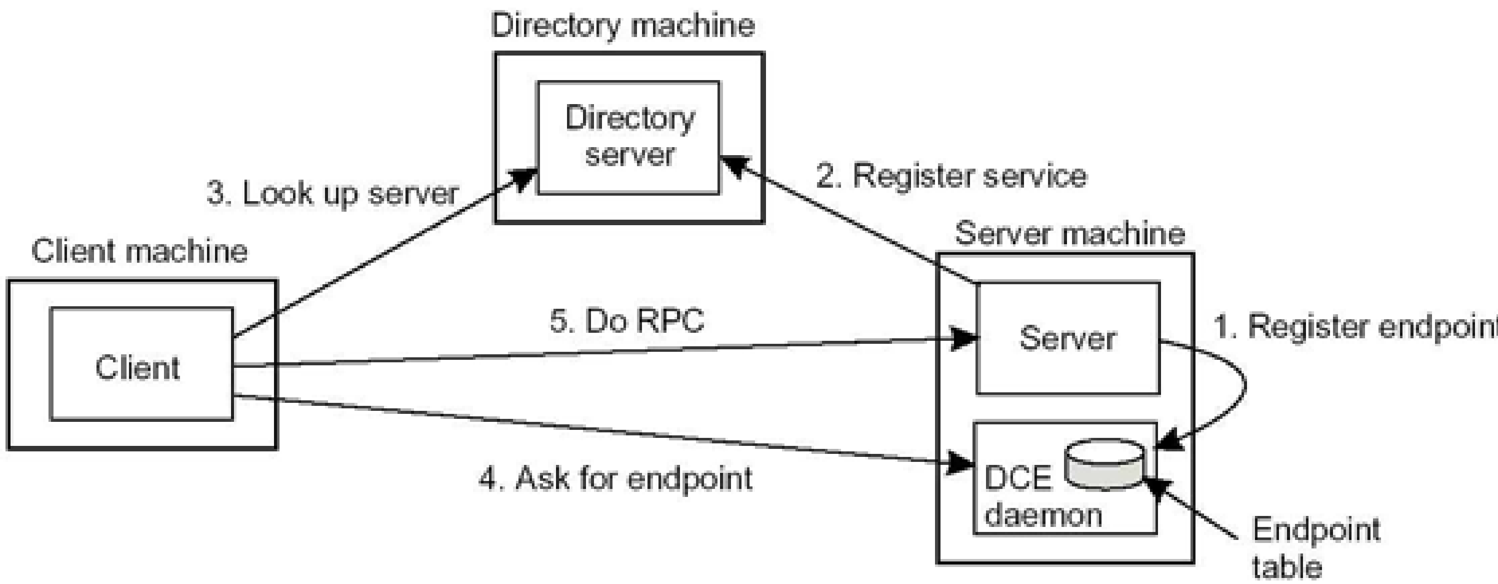
Question of Architectural Philosophy vs. 'plain' Efficiency:

- Interface-based service environments requiring heavy messages
- Efficiency-based API 'style' using light-weighted `http` 'only'

V.1 Remote Procedure Call Characteristics

- Client program needs procedure/function signature for call
- Client process needs address of a server 'serving' the procedure

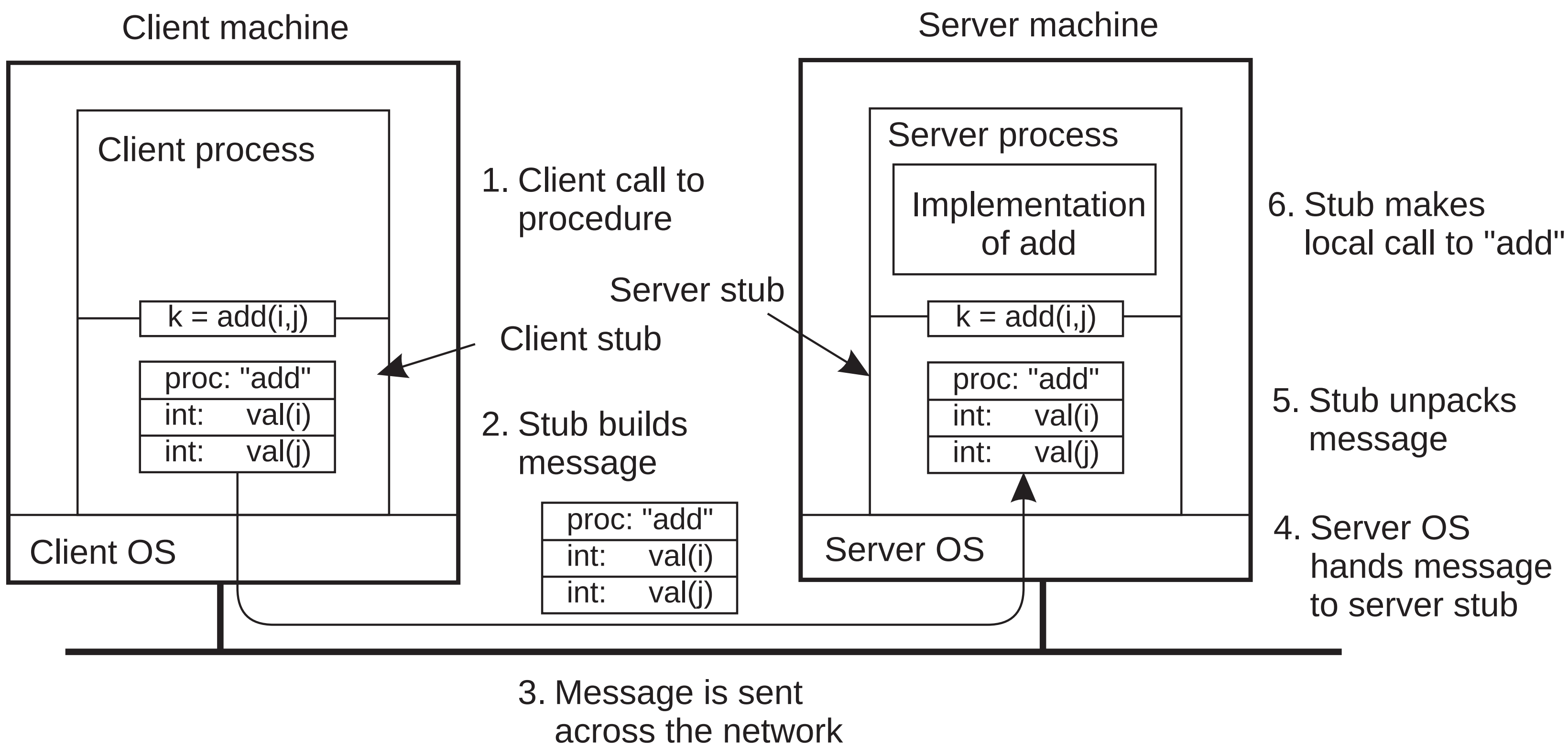
Tanen baum Distri- buted Systems Fig.2.15 DCE



- Server needs a **registry**/directory to publish procedures
  infrastructure to accept remote calls as local calls
- Client needs an initial address for a **registry** or directory
  lookup functionality to find published procedures
- Client and Server need transport protocol(s) to interact at all

Internals of an RP Call with already known Server

Tanen baum Distri buted Systems Fig.2.8



- Steps 1. and 6. constitute a 'classical' local procedure call
- Client calls a *client stub* that mimics exactly a local call
  internal steps 2./3. handle remote aspects and interaction
- Server side remote aspects are hidden by a *server stub*
  - ∗ 4./5. transform message back into call and call local procedure
  - ∗ takes result and sends it back to client stub (not shown)

RPC – Levels to 'guarantee' Transparency

Procedure call using `in/out`-Parameters **implemented** as:

▷ Procedure locally available $\Longrightarrow$ standard local *Library*-Call

▷ Procedure only remotely available $\Longrightarrow$ Stubs (Proxy procedures)

1. **Caller** (client): `call` (User)
   Server lookup; marshal call; WAIT ... (client stub) c.f. pg. V-8
   Usage of directory or meta-servers (*Trader*/*Matchmaker*)
   `SendRequest` with call; (Client Network IF) c.f. pg. V-7
2. **Callee** (server): ... wait using a `GetRequest`
   receive `Request`; (Server Network IF)
   un-marshal request; (server stub)
   local procedure-`call` ... WORK ... `return` result;
   marshal result (server stub)
   `SendReply` with result; ... (Server Network IF)
3. **Caller** (client): WAIT ...;
   receive `Reply` with result; (Client Network IF)
   un-marshal result; (client stub)
   `return` result (User)

Problems w.r.t. RPC–Transparency

- **Performance**: Remote Calls imply lots of Overhead
  - ◁ transfer from (to) programming language MEM via serializing/ packing (deserializing/unpacking) to (from) network
  - ◁ Operating System(s): process handling on both sides
  - ◁ Network: routing, varying load, errors, re-transmit, . . .

  $\Longrightarrow$ Local Call should be much faster.

- **Infrastructure:** *Server* has to be identified and contacted
  - ∗ **naming**: symbolic interface names, e.g., ports
  - ∗ **locating**: fixed network address vs. inquiry via *broadcast* more flexible: Meta-Server manages Server registrations
  - ∗ **binding**: decide on concrete Server for runtime or a single call

  $\Longrightarrow$ Wide Spectrum of Supporting Environments:

  - fixed known `URI`, registry, . . ., Service Eco Systems
  - basic functionality . . . advanced **S**ervice-**L**evel **A**greements SLAs

Problems w.r.t. RPC–Transparency – cont'd.

- **Parameter Handling**: has to respect distributed memory
  - ▷ *call-by-value* easy via copy and marshalling
  - ▷ *simple value results* easy via copy and marshalling
  - ◀ **Problems:** 'Remote Addresses' in Distributed Memory?
    - ∗ *call-by-reference*: reference is useless on server side
    - ∗ Global variables or implicitly used local references in code
    - ∗ Additional resources, e.g., open file pointers
  - ▷ **'Solutions':**
    - ▷ Objects (references) may be marshalled and migrated to Callee
    - ▷ Call-by-visit/move: temporary vs. 'permanent' migration

      Remark: Object-oriented concepts reduce these problems!

    Emer-ald

    - ◀ Files do not fit into the 'Remote' Computing Model

      $\Longrightarrow$ Store data in **Database Servers**

Problems w.r.t. RPC–Transparency – cont'd.

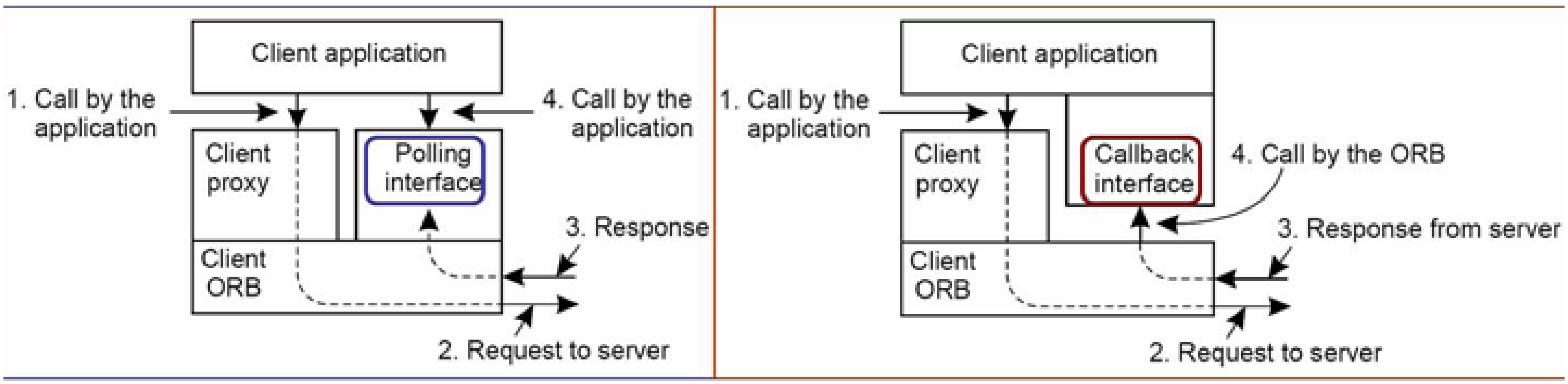
- **Synchronization** (Caller)
  - ▶ **Standard**: Caller blocks until `return`-value is available exactly as in a local Procedure Call $\Longrightarrow$ Transparency
    Trade-Off: simple control vs. idle time in Client-Processor
  - ▶ **Optimizations** possible but forfeit transparency advantage
    - ▷ **Asynchronous Calls without waiting for results** $\Longrightarrow$ Request-Protocol without Reply, e.g., `SunRPC` (timeout 0)
    - ▶ **Asynchronous Calls with delayed waiting for results**
      Client control structures for 'postponed wait' required
      Concept: promises $\approx$ proxy for expected result; Client waits via `claim`-Operation which is `blocked` until result arrives — Liskov 1988
      Implementation: `Callable` $\approx$ `Runnable` with return value — DSG-PKS-B
      `Future` $\longrightarrow$ send to `Executor`; compute asynchronously, etc.
      Example.: `java.lang.*/java.util.concurrent` since `Java 1.5`

Relaxed RPC Model: Callback vs. Polling

Tanenbaum Distributed Systems Fig. 9.7/8

Example: CORBA `Common Object Request Broker Architecture`

- Client issues RPC (via stub) and resumes local work asynchronously
- Server gets requests and computes procedure
- Asynchronous model less transparent but much more flexible



New Issue: *Who triggers delivery of result?* Client vs. Server

- **Polling:** *Client* 'asks' explicitly for result when needed — left
- **Callback:** *Server* calls callback interface provided at client side — right
  responsibility reverses Client/Server role for result

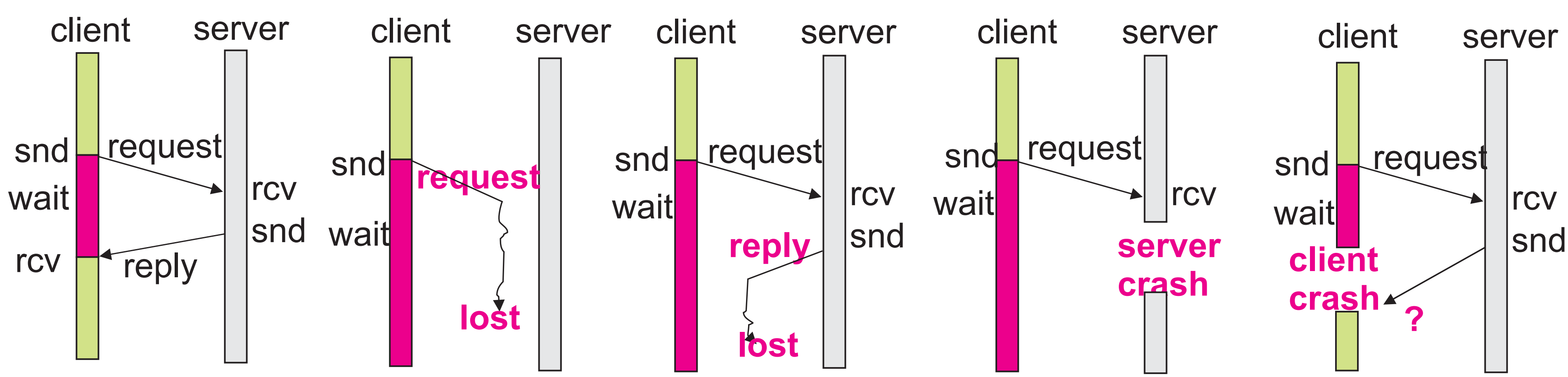
V.2 Client-Server-Interaction: Handling of Failures

c.f. chapt. I.3

Reasons: communication channels and/or compute nodes

- Messages: request or reply lost
- Server unable to accept or process request, e.g., due to a crash
- Server takes unexpectedly long to process request due to high load
- Client is crashed and not able to accept result

$\Longrightarrow$ Errors not easy to distinguish on client side



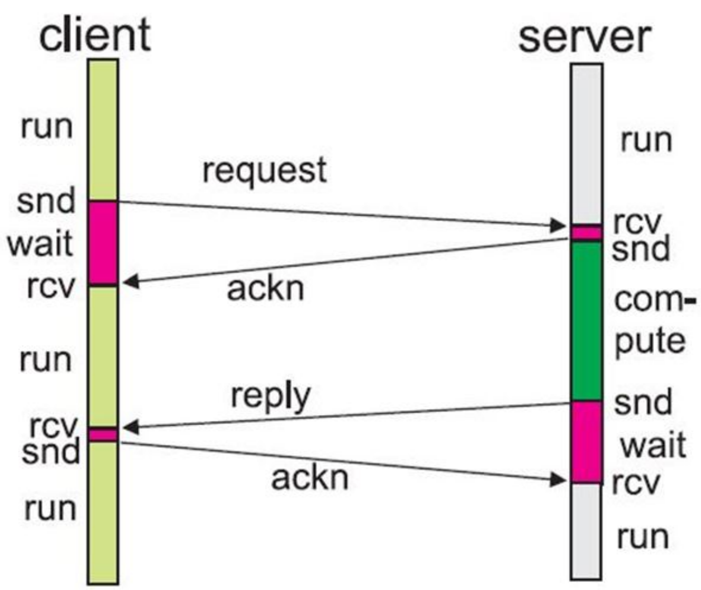
- Time-Outs and re-sends are critical to ensure successful interaction.
- Different levels of robustness can be achieved by different communication protocols for implementing RPCs.

C/S-Interaction Protocols: 4–Message–Protocol

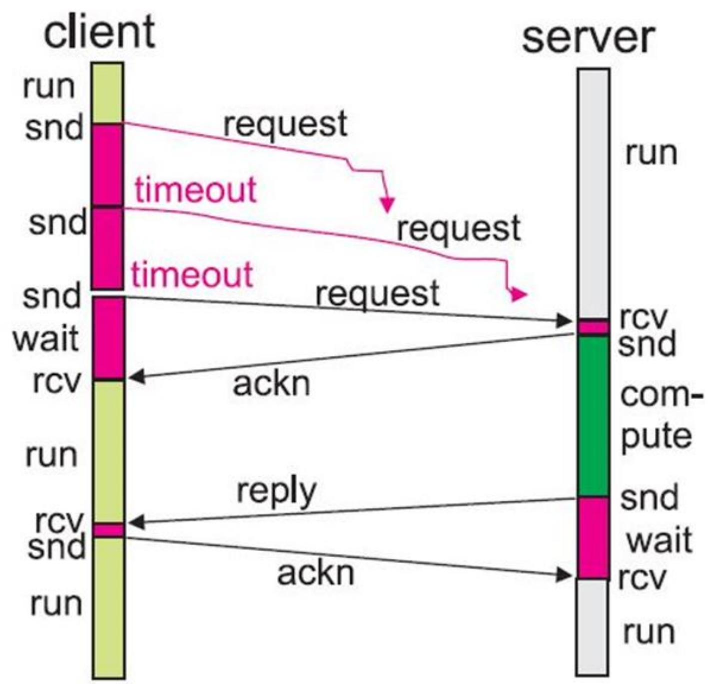
`rcv` uses *Time-out* > transfer time + expected msg handling time

Time-out: Transfer time Client ⟶ Server + Server ⟶ Client

`C.snd(req); C.rcv(ackn); ... C.rcv(reply); C.snd(ackn);`



4-msg protocol: no errors



4-msg protocol: lost requests

Advantages:

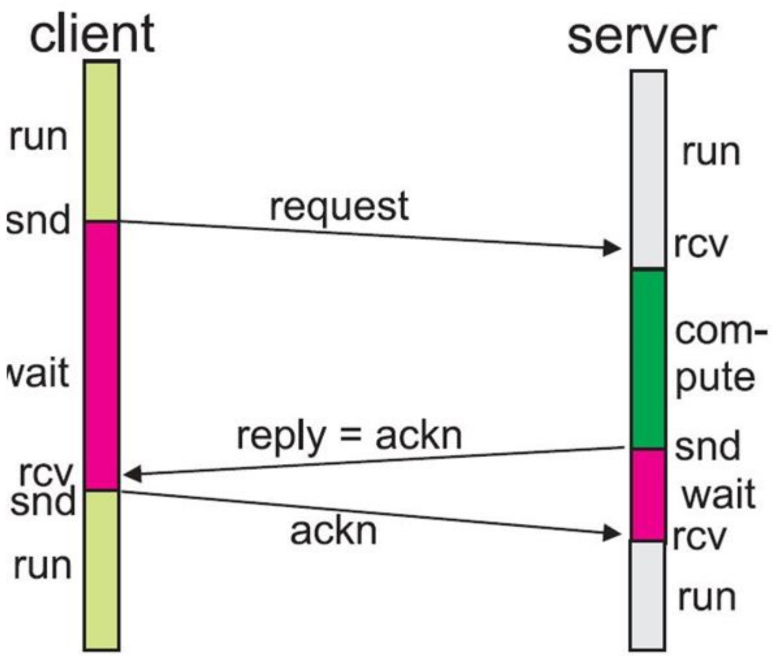
▷ Client is not blocked during entire `Request` processing

▶ Time-out is independent from 'job' dependent compute time

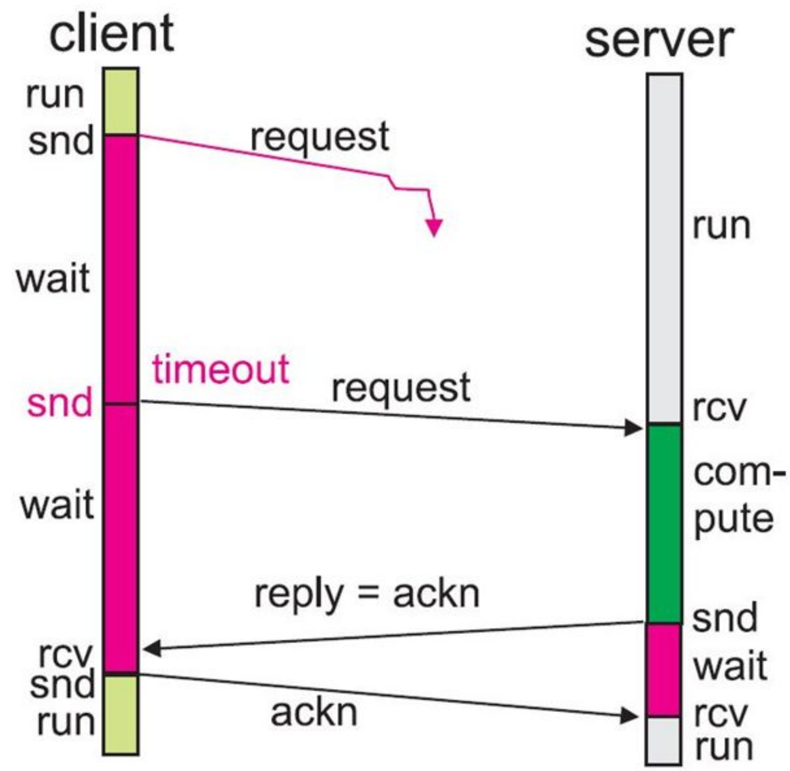
C/S-Interaction Protocols: 3–Message–Protocol

`Request` is not explicitly acknowledged;
`Reply` is used as implicit `acknowledgement`

`C.snd(req); C.rcv(reply); C.snd(ackn); ...`



3-msg protocol: no errors



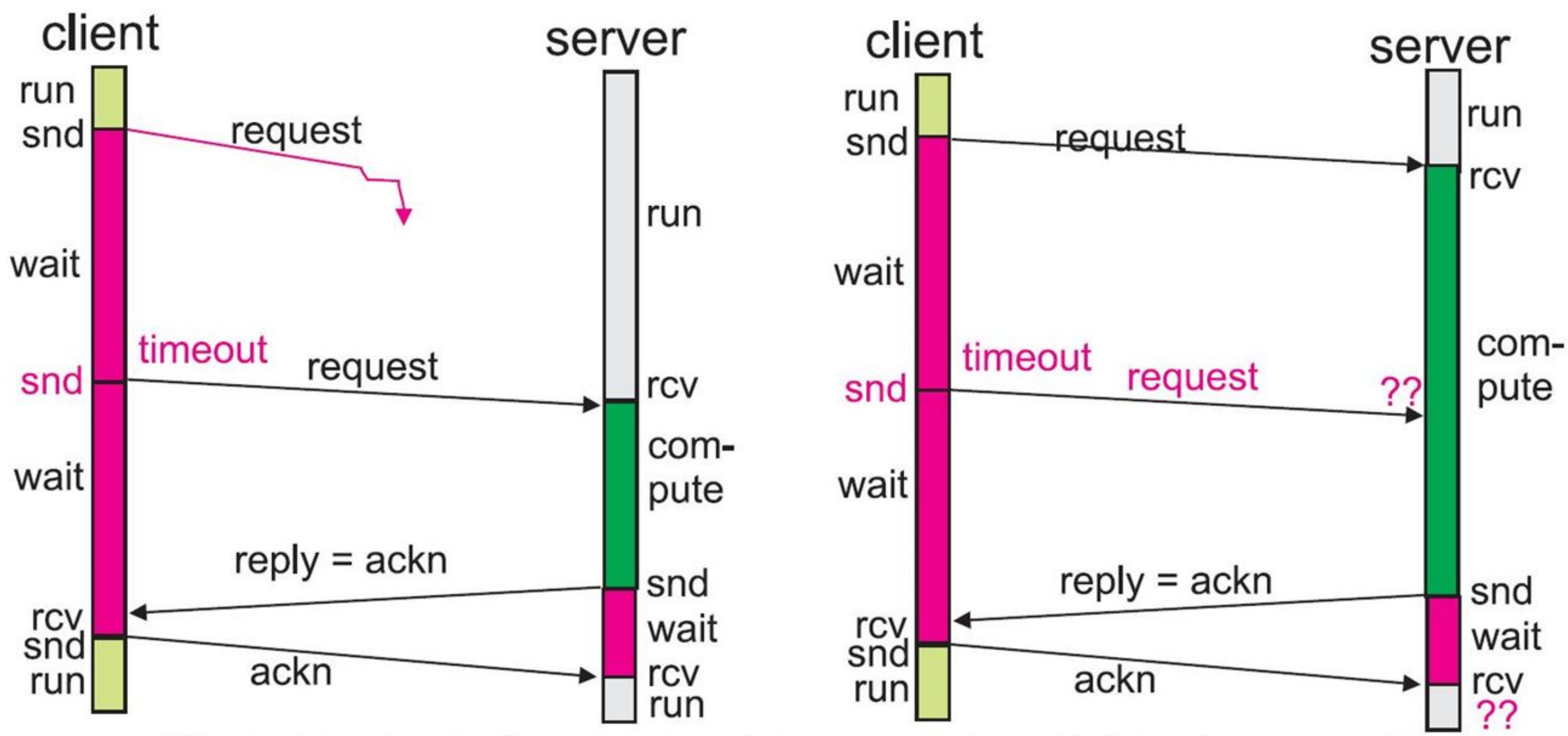
3-msg protocol: lost requests

Drawbacks: Client blocks for entire time of `Req` processing
'long' timeouts have to respect compute time

Problem: Setting adequate Time-outs is critical

◀ too short: long compute times lead to repeated re-sends

◀ too long: errors are detected after long blocking times only



Client-side view in 3-msg protocol: lost requests or high load on server?

▷ Compromise for Time-Out on client side

▶ Server should always use short timeouts for `ackn`: avoids blocking.

C/S-Interaction Protocols: 2–Message–Protocol

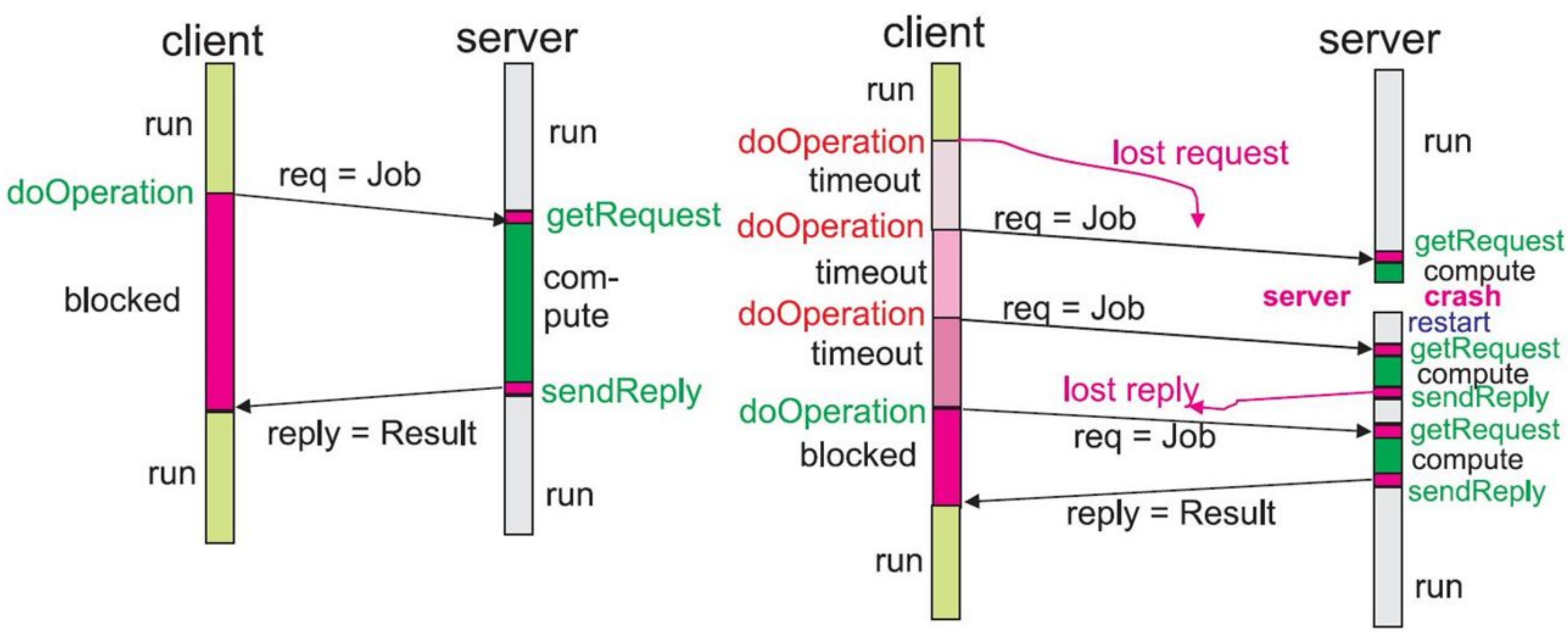
Concept: Do not use any explicit acknowledgments: optimal w.r.t msg-count

`C.snd(req); C.rcv(reply)`

If `reply` is lost, simply re-send the `request` as 'new'

$\Longrightarrow$ Protocol consists of 3 operations only:

`C.doOperation(...); S.getRequest(...); S.sendReply(...)`



2-msg protocol: no errors 2-msg protocol: Error handling

Transparency: Client waits *exactly as in a local Procedure Call*

Semantic Models w.r.t. Failure Handling

Problem: *How to handle partial failures?*

- local calls: Caller/Callee on the same machine $\Longrightarrow$ both crash
- remote calls: typically only one side crashes
  lost messages as an additional source of failures

Critical: non-idempotent operations causing side-effects c.f. pg. V-20

Example: `read(i)` vs. `increment(i)`; repeated money transfer

Hierarchy of Models:

(a) **Maybe/best effort**: No guarantees $\Longrightarrow$ hard to use
simple and efficient; Call is executed either once or not at all

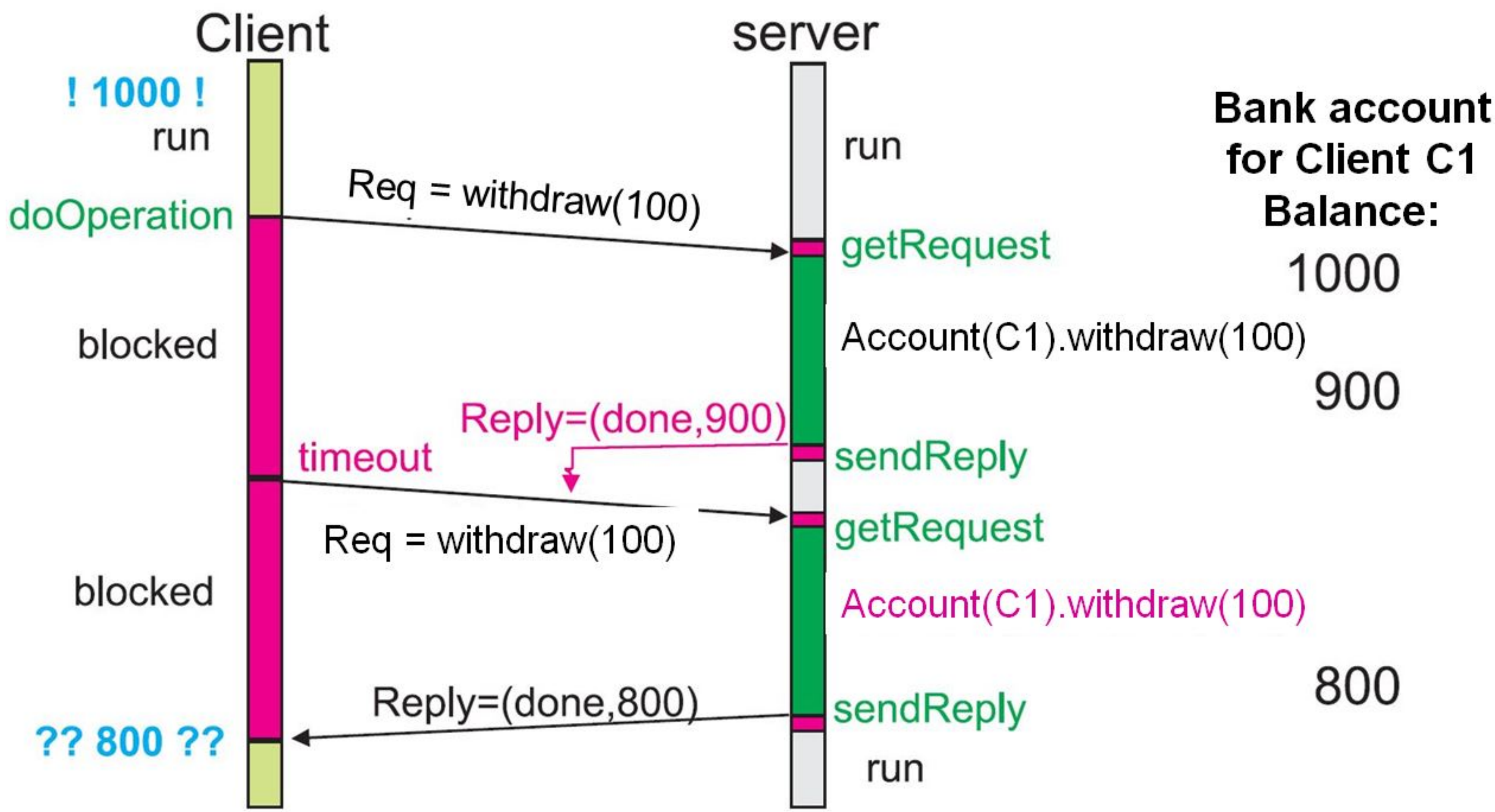
(b) **At-least-once**: System ensures execution of call

- Client: repeats requests (`doOperation`) until executed c.f. pg. V-18
- Server: doesn't check for *duplicates* $\Longrightarrow$ repeated executions?

$\Longrightarrow$ high load levels combined with short Time-outs are critical

$\Longrightarrow$ **multiple side-effects lead to unwanted system state**

Problem: Lost Msg and non-idempotent Operation



Time
t
↓

◁ Lost replies as well as high loads may lead to multiple re-sends

◀ Server state gets corrupted due to unexpected side effects

Semantic Models w.r.t. Failure Handling – cont'd

Hierarchy of Models (cont'd):

(c) Last-of-many: Result from last Call is accepted only, but nested RPCs may lead to so-called **Orphans** for intermediate results

(d) Last-one: last-of many plus Orphan-Handling for nested calls
Technique: version numbers for calls; detects/discards Orphans

(e) **At-most-once**: Server detects and handles duplicate `Requests`
Technique: Server stores `Request` and `Replies` (for re-send)
$\Longrightarrow$ **works well for non-idempotent calls**

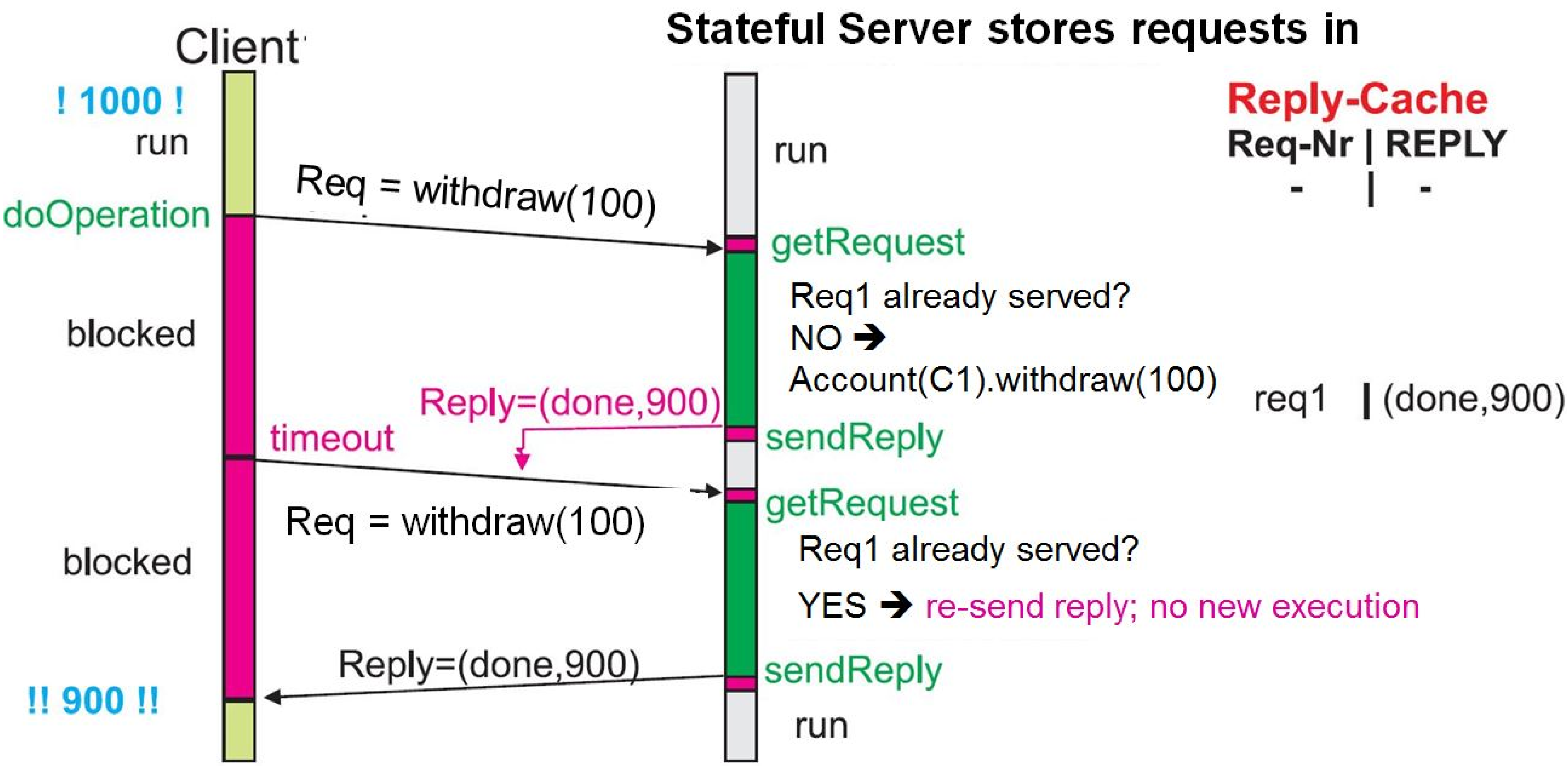
c.f. pg. V-22

(f) **Exactly-once**: Ensures execution also in the presence of crashes
Implementation: at-most-once plus *persistence* techniques
`Req`/`Reply` Caching detects duplicates
Problem: high overhead for global snapshot/rollback algorithms

c.f. chapter. VI

Trade-Off: *High Quality-of-Service vs. Implementation Overhead*

Example: Server Recognizes Duplicate Calls



- Request cache and request numbers needed for all clients
- Request can be flushed from cache when acknowledgement arrives

Communication Protocols to implement RPCs

RPCs are a general mechanism to implement DS:

- wide range of requirements based on actual application
- different Quality-of-Service models with varying overhead costs

$\Longrightarrow$ **Support for different failure models helps:**

1. **R**-Protocol (Request): asynchronous RPC without return value
   Optimal efficiency in case of lots of repeated calls
   Examples: Screen updates; time signals of a 'master clock'
   c.f. pg. V-12
2. **RR**-Protocol (Request,Reply): optimistic standard
   $\texttt{Reply}_i$ acknowledges $\texttt{Request}_i$; $\texttt{Request}_{i+1}$ acknowledges $\texttt{Reply}_i$
   Facilitates at-least-once implementation without much overhead
   c.f. pg. V-18
3. **RRA**-Protocol (Request,Reply, Acknowledge-Reply)
   Facilitates exactly-once on Server: store reply only until `Ackn`
   Client should store order of Requests and acknowledgements
   c.f. pg. V-16

End V.2

Summary: Client/Server using classical RPCs

▶ Interaction model supporting moderate de-coupling of roles
▷ Synchronous version close to traditional procedure call
▶ Frequently used in DS: `DCE`; `SunRPC`; `Corba`; `RMI`; ...

◀ No transparency w.r.t. call-by-reference and resources
◀ No transparency w.r.t. infrastructure for servers etc.
◁ comfortable levels of failure transparency are costly

◀ Optimizations on client side lose transparency advantage
◁ No integrated concept of 'server state', e.g. for handling situations where calls cannot be executed due to program logic.

Modern Variant(s):

- *Light-weight RPC* models: `XML-RPC`; `JSON-RPC`; `google gRPC`
- *Services* based on Web Service Stack or REST-based APIs

V.3 Google gRPC

- 'Classical' RPC implementation based on an IDL
- Result of several steps of internal RPC implementations at `google`
- 'Incubating Project' of the *Cloud Native Computing Foundation* (c.f. `www.cncf.io/projects/`)

Characteristics:

▷ Designed for Interaction on the Internet via `HTTP/2` transport

▷ Multi-Language Support (> 10 languages): `protocol buffer` as IDL, but other formats possible, e.g. `JSON`

▷ **Advanced Interaction Styles** between a single Client and Server:

- ∗ Single Message: Request-Reply
- ∗ Stream-based models: 1-n/n-1/n-m Message Stream Flow
- ∗ Synchronous and Asynchronous Interaction (Deadlines/Cancelling)
- ∗ Advanced Error Handling (easily mapped to HTTP Error Msgs)

IDL: Protocol Buffers specify Payload and Services

- `message` declarations for payload structures for de/serialization
- `service` used to bundle one or more `rpcs`
- Single RPC: Keyword `rpc` plus Parameterlist and Result

  `rpc` <Name>(<Parmlist_Msg_Types>) `returns` (<Result_Msg_Type>)

Interaction Styles specified via Parameter/Result:

∗ `message` type names only ⟹ single Request and/or Reply

∗ Keyword `stream` used for Parameter or Result

⟹ Server-/Client-/Bidirectional Streaming

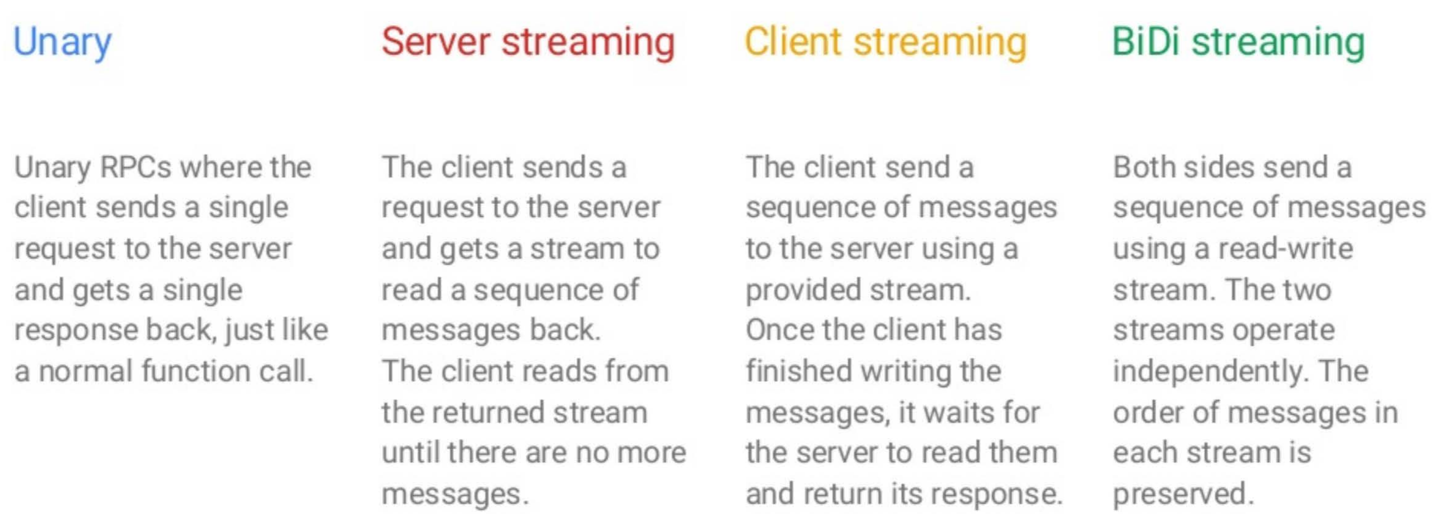


Fig. taken from: www.slideshare.net/borisovalex/enabling-googley-microservices-with-http2-and-grpc?next_slideshow=1 pg.86

gRPC Interaction Styles and Synchronisation

- `newStub` $\Longrightarrow$ **asynchronous** calls from client
  extended RPC model supports all interaction variants on both sides
  - ∗ streams are not 'synchronized' between Client and Server
  - ∗ interaction uses 'Observers': `onNext`; `onCompleted` ...
- `blockingStub`: $\Longrightarrow$ **synchronous** calls from client
  'classical' RPC model; no streaming on client side possible
- `newFutureStub` $\Longrightarrow$ **synchronizes** explicitly when result retrieved
  'lazy evaluation' model; no streaming calls possible

Additional Synchronization:

- *Timeout*: how long waits a client for RPC result
- *Deadline*: fixed time when RPC result should be received
- *Cancelling*: always possible on client and server side

depends on lang uage

Problem: Inconsistent views w.r.t. 'Success' on client vs. server, e.g. deadline matched on server side BUT deadline exceeded on client caused by network transfer on reply

gRPC Error Handling Issues

- **Standard Error Model:** `OK` iff success, Error-Codes otherwise
  - supported by all languages, implementations, stubs
  - 16 general **gRPC Error Codes** supported by any model, e.g.

    `OK`, `UNKNOWN`, `INVALID_ARGUMENT`, `DEADLINE_EXCEEDED`, ... `PERMISSION_DENIED`, `UNAUTHENTICATED`, `UNIMPLEMENTED`, `RESOURCE_EXHAUSTED` , `UNAVAILABLE`, `Data_LOSS`

- **'Rich' Model**: detailed reasons and/or how to overcome an error specified in `status/error_details.proto` files:
  - `BadRequest` info w.r.t. wrong parameter types or ranges
  - `RetryInfo` for suitable timeout; `QuotaFailure`; ...
  - Stack trace, meta-level info for request, documentation

Remark: codes can be matched to `HTTP/2` error codes (`github.com/googleapis/googleapis/blob/master/google/rpc/code.proto`)

Developing a gRPC Multi-Language Application

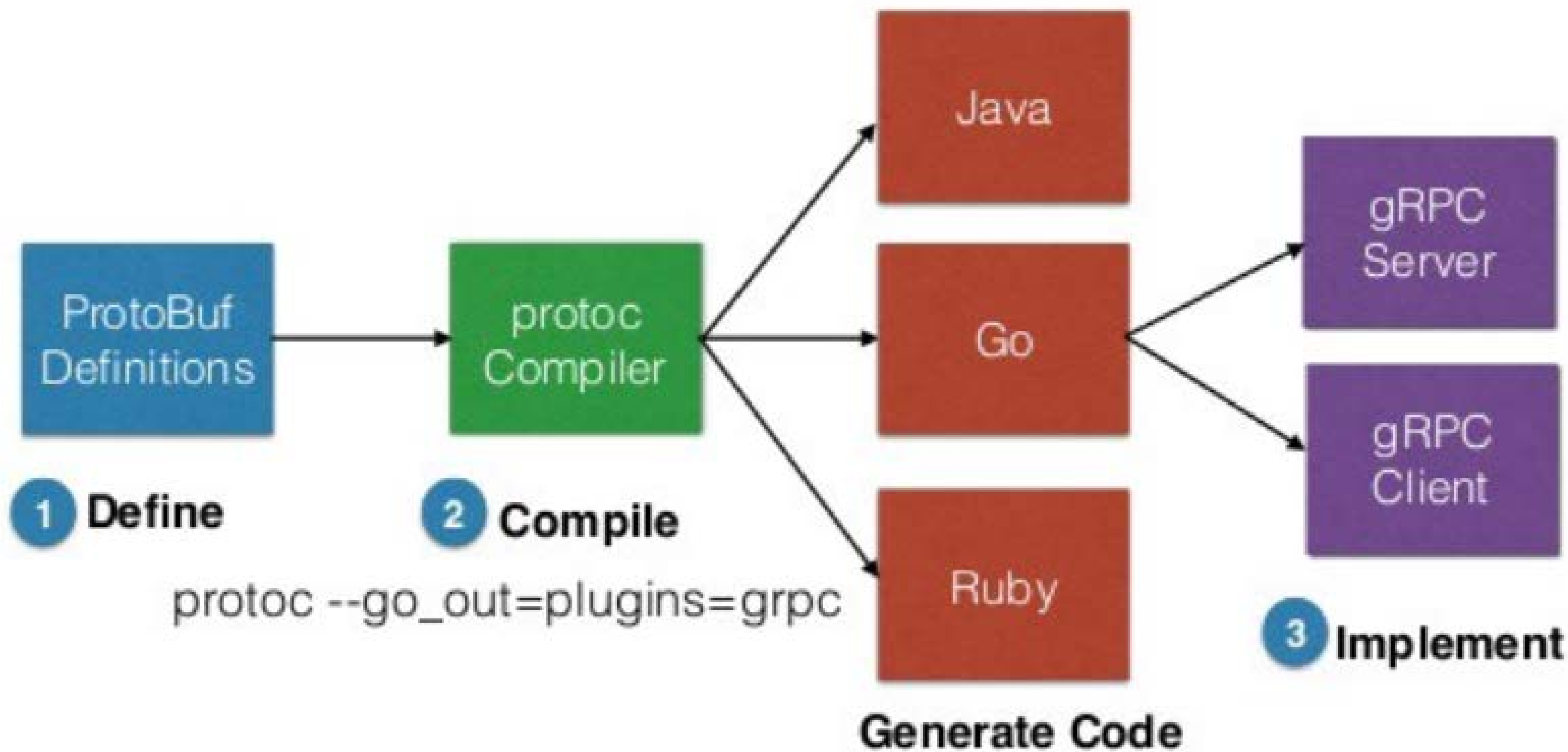


Fig.: medium.com/@akshitjain_74512/inter-service-communication-with-grpc-d815a561e3a1

- Specify the `service` interface
- Implement the server code
- *Tooling*: Generate stubs needed for your language(s) for both sides
- Implement your Client

Szenario: Combine Multi-Language Services

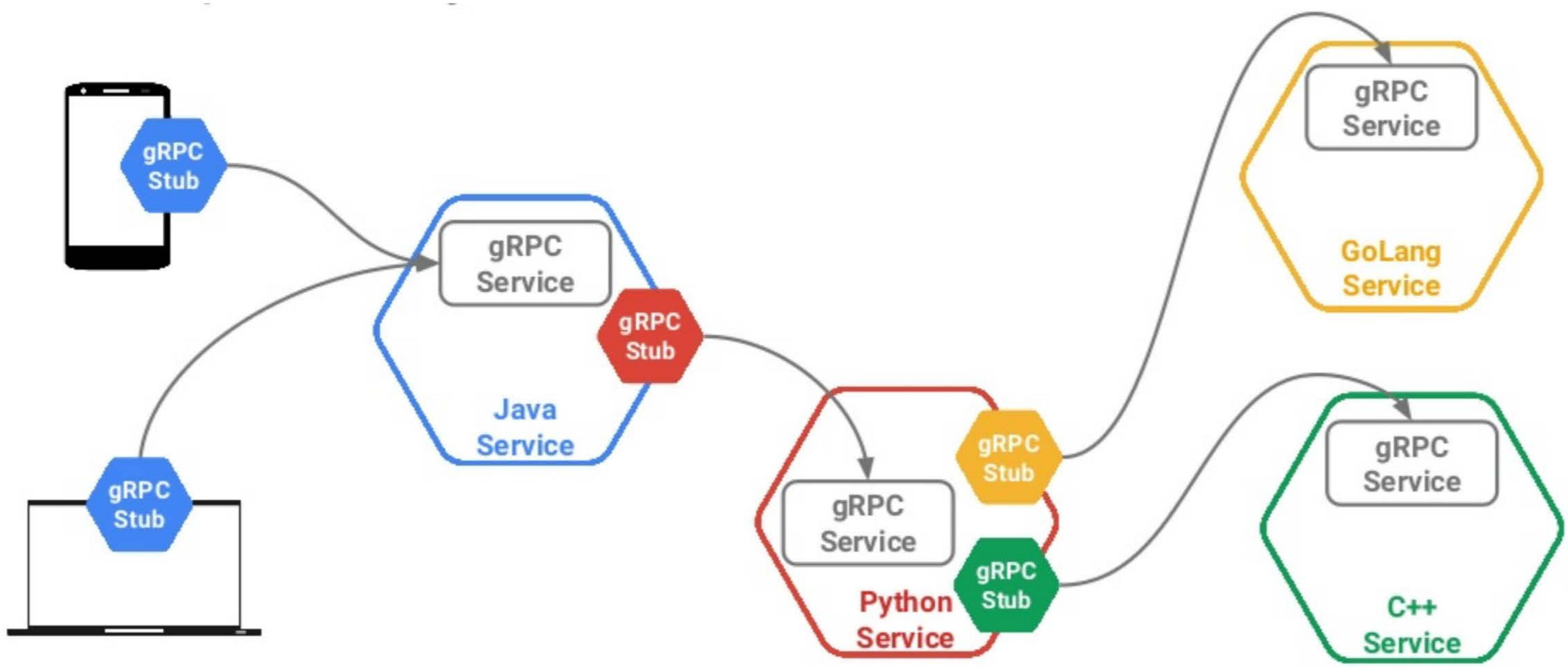


Fig. taken from: www.slideshare.net/borisovalex/enabling-googley-microservices-with-http2-and-grpc?next_slideshow=1 pg.99

- IDL as a special 'extra' language guarantees independence
  `proto files` describe Msg formats $\Longrightarrow$ Transport Independence
  `proto files` describe Service Interface $\Longrightarrow$ Call Independence
- IDL plus Stub-Compilers allow for easy language support

Behind the Scenes – 1: HTTP/2-based RPC

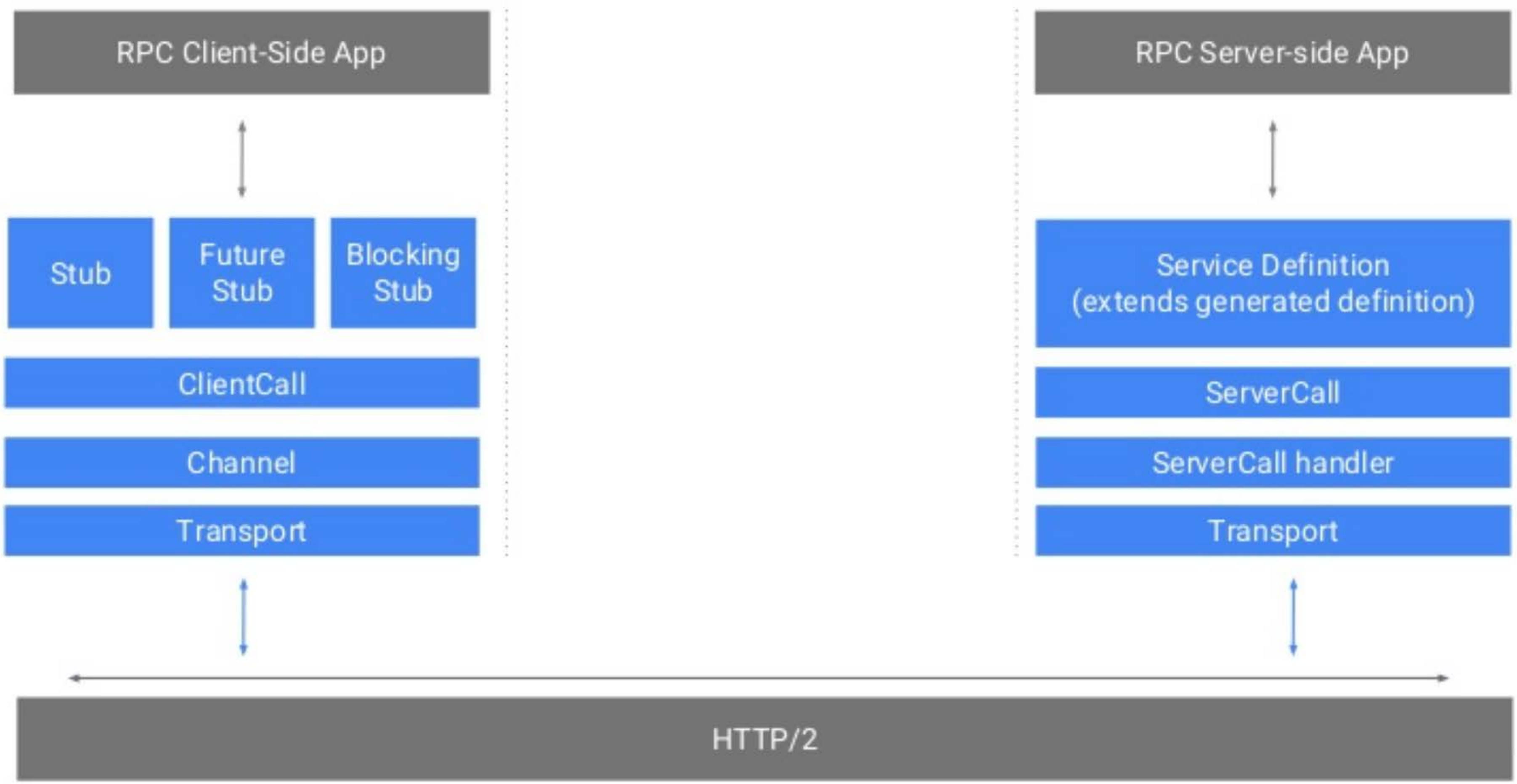


Fig. taken from: www.slideshare.net/borisovalex/enabling-googley-microservices-with-http2-and-grpc?next_slideshow=1 pg.165

- Classical RPC implementation (see `DCE`)
- Stubs for handling a/synchronous calls and synchronization
- `Channel`: `HTTP/2` connection between client stub and server

c.f. pg. V-8

Behind the Scenes – 2: Multi-Channel Support

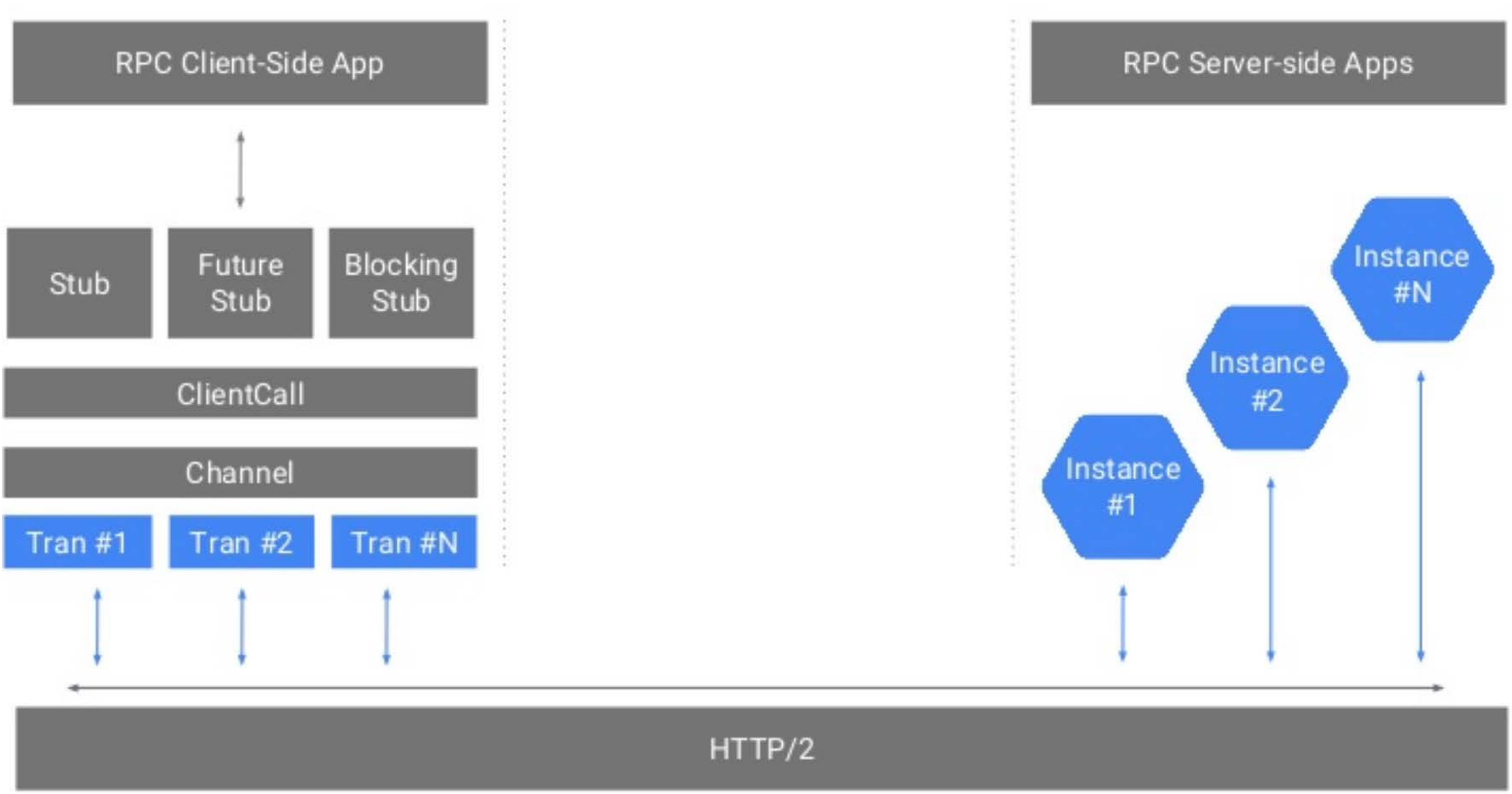


Fig. taken from: www.slideshare.net/borisovalex/enabling-googley-microservices-with-http2-and-grpc?next_slideshow=1 pg.167

- `service` may be implemented by many servers ('instances')
- Transport may be duplicated in order to gain performance

Behind the Scenes – 3: Channel Loadbalancing

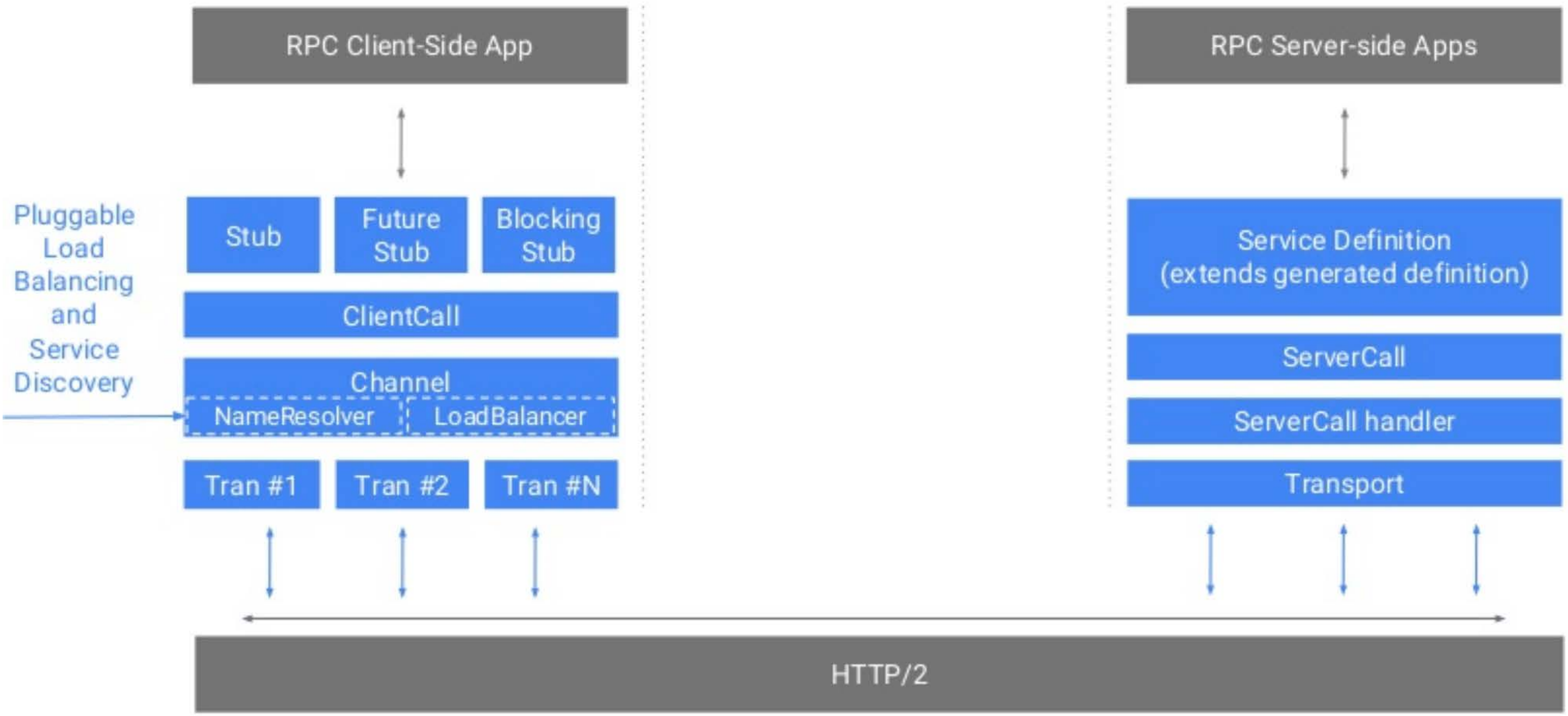


Fig. taken from: www.slideshare.net/borisovalex/enabling-googley-microservices-with-http2-and-grpc?next_slideshow=1 pg.172

- Higher-level techniques usable through 'Interceptors' as 'plugins'
- Finding suitable servers based on Discovery Service
- Include 'Load balancing' for optimizing channel usage etc.

End of V.3

Combining gRPC with REST-based Environments

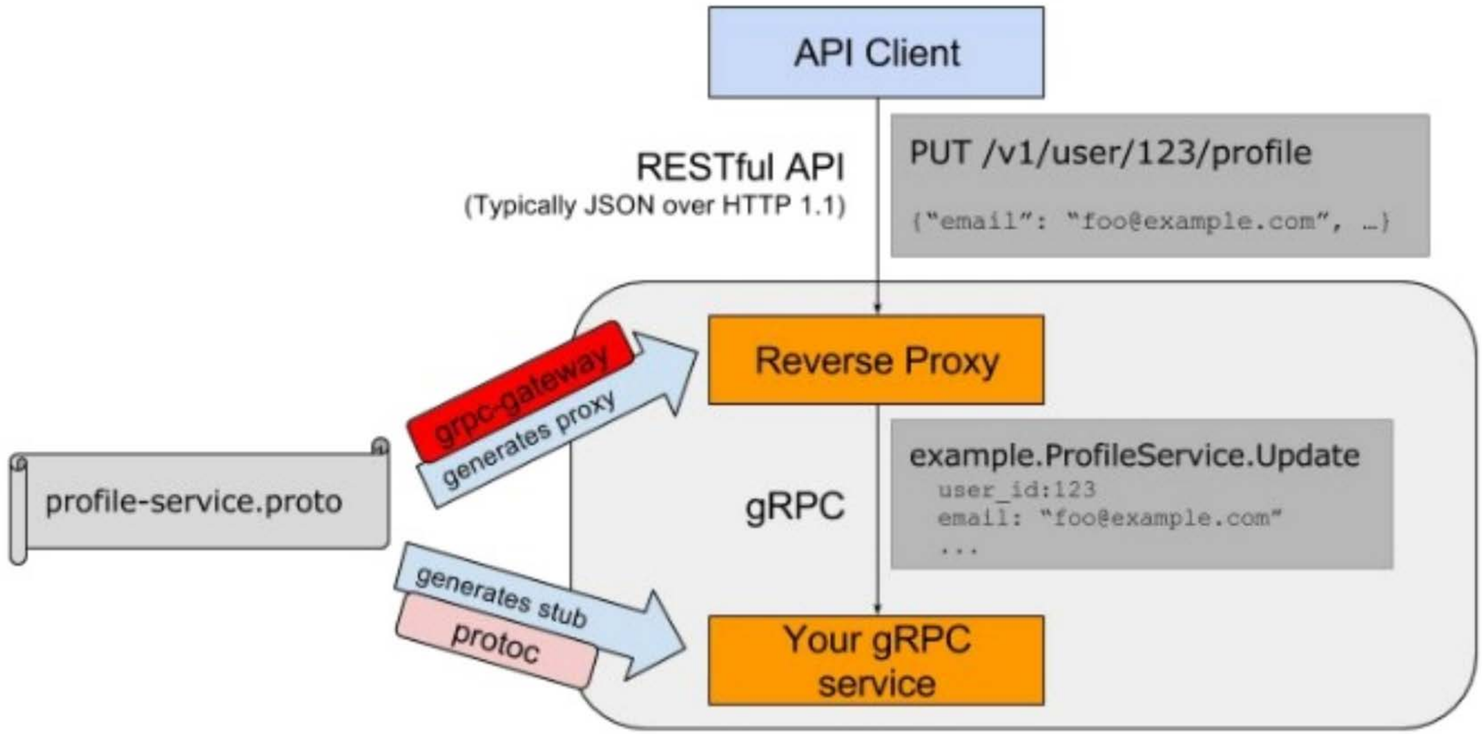


Fig. taken from: www.slideshare.net/borisovalex/enabling-googley-microservices-with-http2-and-grpc?next_slideshow=1 pg.193

- ▷ gRPC as part of `Google` eco-system $\Longrightarrow$ Interoperability
- ▷ Interfacing other popular techniques, esp. REST-based APIs is done in a way that a gRPC service is reachable and usable by:
  - ∗ 'ordinary' gRPC using the generated stub
  - ∗ gateway to a RESTful API via generated 'Reverse Proxy'

V.4 REST-based Services

[Roy Fielding, Diss. 2000]: '*The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system.* [...] *It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application.*' www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

REST: **RE**presentatial **S**tate **T**ransfer

- Architectural Style, not a concrete technology $\Longrightarrow$ 'very abstract'
- inspired by/dedicated to the World Wide Web
- Handling of Web resources, esp. hypermedia content of all kind
- developed 1994 – 2000 'in parallel' to `URI` and `HTTP/1.1`

Remark: *We discuss only an overview of REST in general and focus on the C/S (RPC) aspect in interpreting* **RESTful services** *as* `HTTP`*-based RPCs. There is much more to the REST architectural style when it comes to the detailed architecture guidelines and the Web in general. A more concise treatment is offerend in the master DSG-SOA-M module.* ♦

REST Characteristics – Resources and Addresses

tools.ietf.org/html/rfc2396 (1998) rfc3986 (2005)

'A resource can be anything that has identity.'

'[. . .] abstract concepts can be resources.'

- 'Everything is a **Resource**' as the basic concept:
  - documents, images, video/audio, (structured) objects, **services**
  - works on single as well as multiple entities (collections)
  - resources may be **linked** together and navigated like **Hypertext**
  - single resource may have various different representations/formats
- **Addressing: U**niform **R**esource **I**dentifier (`URI`)
  - different types of 'Identification' : **UR N**ame vs. **UR L**ocation
  - extensible syntax based on `schemes`, e.g. `https`, `mailto`, . . .

    `scheme://[userinfo@]host[:port] path [?query] [#fragment]`

    `https:// en.wikipedia.org /wiki/Uniform_Resource_Identifier#URLs_and_URNs`
  - direct access or dynamically navigating through **links**

$\Longrightarrow$ **general and universally applicable concept**.

REST Characteristics cont'd – Interaction

- **Interaction**: 'Messages' based on `HTTP` interaction model
  - **uniform** requests: `POST`, `GET`, `PUT`, `DELETE`, ...
  - **uniform** responses via HTTP-defined formats and codes
  - typical API style: **C***reate* **R***ead* **U***pdate* **D***elete* (`CRUD`)

c.f. pg. V-59

Method	REST Operation	Description
POST	CREATE (INSERT)	Create or update
GET	READ (QUERY)	Query about the resource
PUT	UPDATE (CHANGE)	Update
DELETE	DELETE (DELETE)	I want to delete what-ever-it-is ....
HEAD		I'm something like 'GET' [1]
OPTIONS		JAX-RS mumbles something about me.

Fig.: de.slideshare.net/imcinstitute/java-web-services-15-rest-and-jaxrs

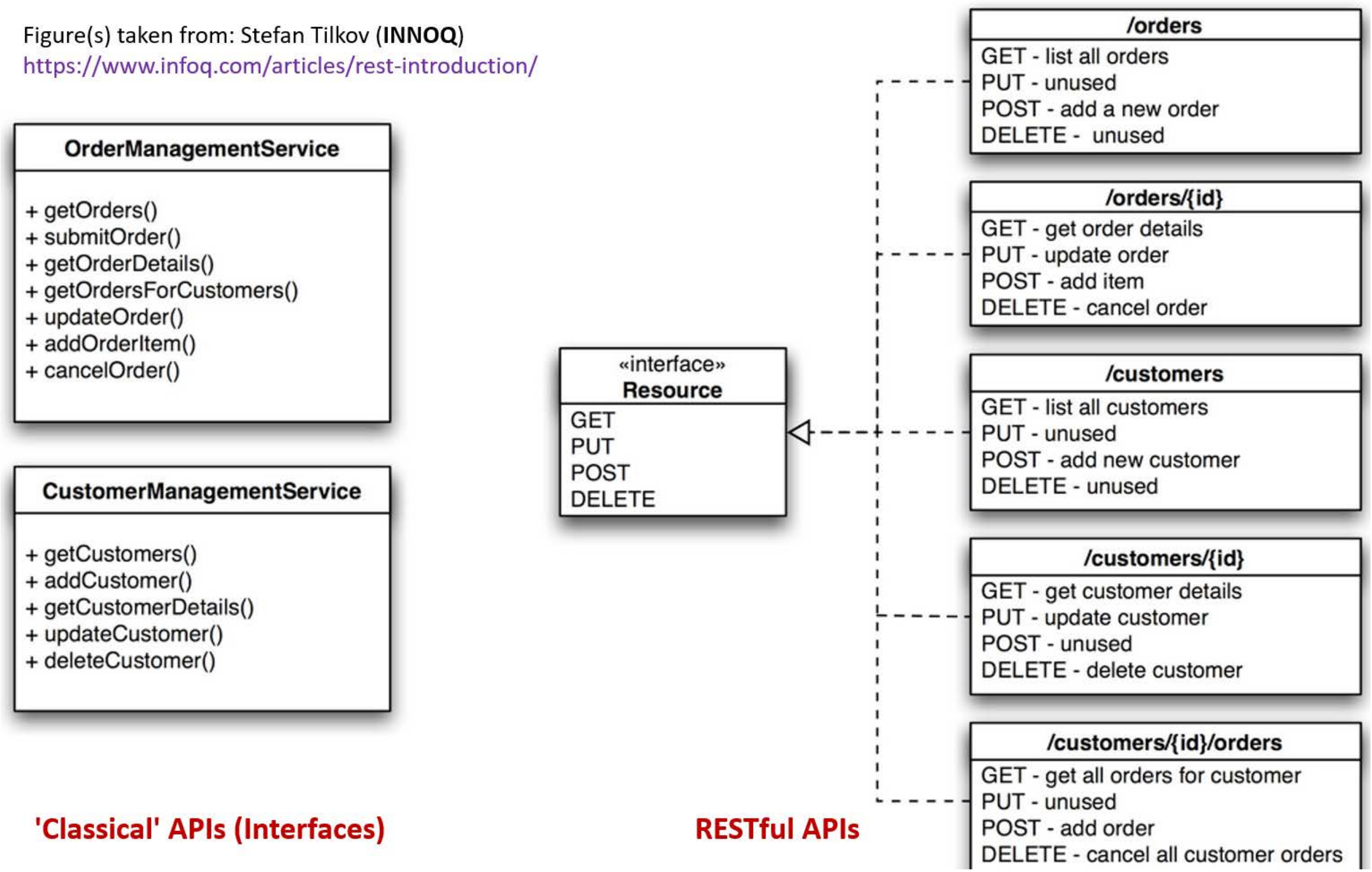
▶ Service APIs are always syntactically uniform

▶ No fixed API (syntactical service spec) needed **before** contacting a service $\Longrightarrow$ **highly flexible and supports dynamic change!**

▶ Service Semantics in (more) resources and operation parameters

Example – Classical vs. RESTful API Style

Figure(s) taken from: Stefan Tilkov (**INNOQ**)
https://www.infoq.com/articles/rest-introduction/



REST Characteristics cont'd – Stateless Services

- **Stateless Services** as an architectural principle
  - ◀ C/S applications without state are mostly useless
  - ▶ state should **not** be kept 'implicitly' on the server side, but
    - ▷ store server-side aspects of the state explicitly in resource state
    - ▷ keep everything else on the client side
    - ◀ avoid client-specific server states like, e.g., sessions, cookies
  - $\Longrightarrow$ **scaling, load balancing, replacing resources much easier**

- **Messages** (Request/Replies):
  - ∗ have to carry always all relevant information
  - ∗ transfer *Representations of Resource States*
  - ∗ may be *cacheable* on Client side for performance reasons
  - ∗ trigger effects on states by sending changed representations
  - $\Longrightarrow$ **flexible and easy for distributed systems**

hence the Name

see: coupling levels pg. III-2

Taking Restful Services serious – HATEOAS

H*ypertext* **A***s* **T***he* **E***ngine* **O***f* **A***pplication* **S***tate*

Term coined by Leonard Richardson (2008)

www.crummy.com/writing/speaking/2008-QCon/act3.html

▷ Enhancing C/S architectures step by step towards 'solid REST'

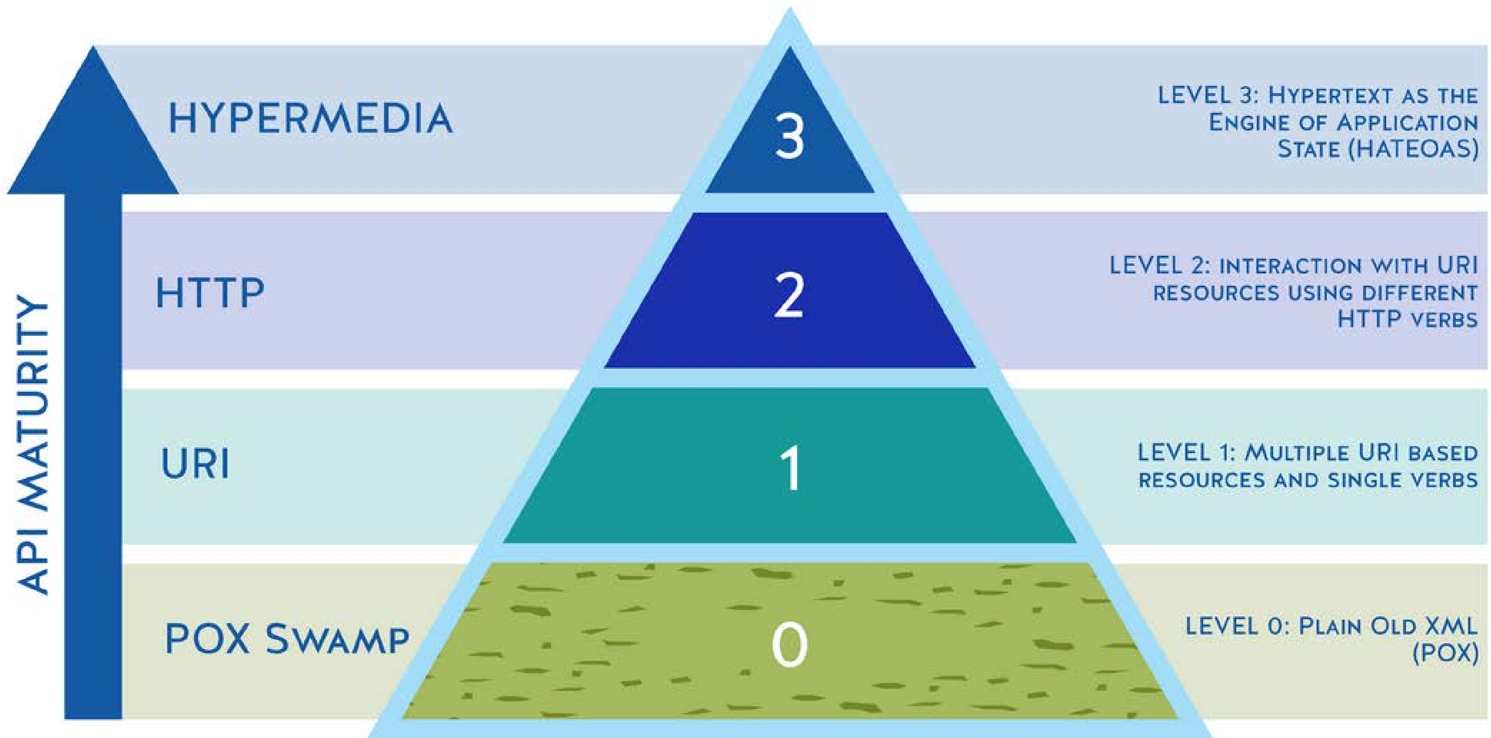


Fig.: nordicapis.com/what-is-the-richardson-maturity-model/

◀ Level 3: only RESTful level in the sense of R. Fielding's definition

Representation Formats and Java-'Binding'

- Level 1 up: No need for 'classical' operation interface specification
- Needed: format(s)/structure definition for representations (msgs)

$\Longrightarrow$ *Description Languages* for RESTful APIs?

◁ Web Application Description Language (WADL) $\approx$ WSDL4REST?

▷ RESTful API Modeling Language (RAML): YAML-based

▶ OpenAPI (Swagger): resource structure specifications

▶ needed: structure/format of resource data, e.g. `JSON`, `XML`

Java API for RESTful Webservices `JAX-RS`: (*Intro 2nd Assignment*)

- Code-First approach using `Java`-annotations: `@PATH`, `@POST`, `@GET` ...
- `JSON` is used for msg content specification
- Java-based tooling (`Jersey` as reference implementation)

End of V.4

V.5 Webservices and SOA

Note: This part is an Add-On for completing the overview of RPC technologies (for 'historical' reasons) - it is NOT part of the regular core module content - but may be interesting for you anyway.

▷ **Client/Server**-paradigm uses '*Service*' notion implicitly

▶ **Service-Oriented Computing** (SOC)

- centers around an explicitly defined notion of 'Service'
- provides 'the' C/S implementation from approx. 2000-2015
- fosters a *description-oriented style* of programming

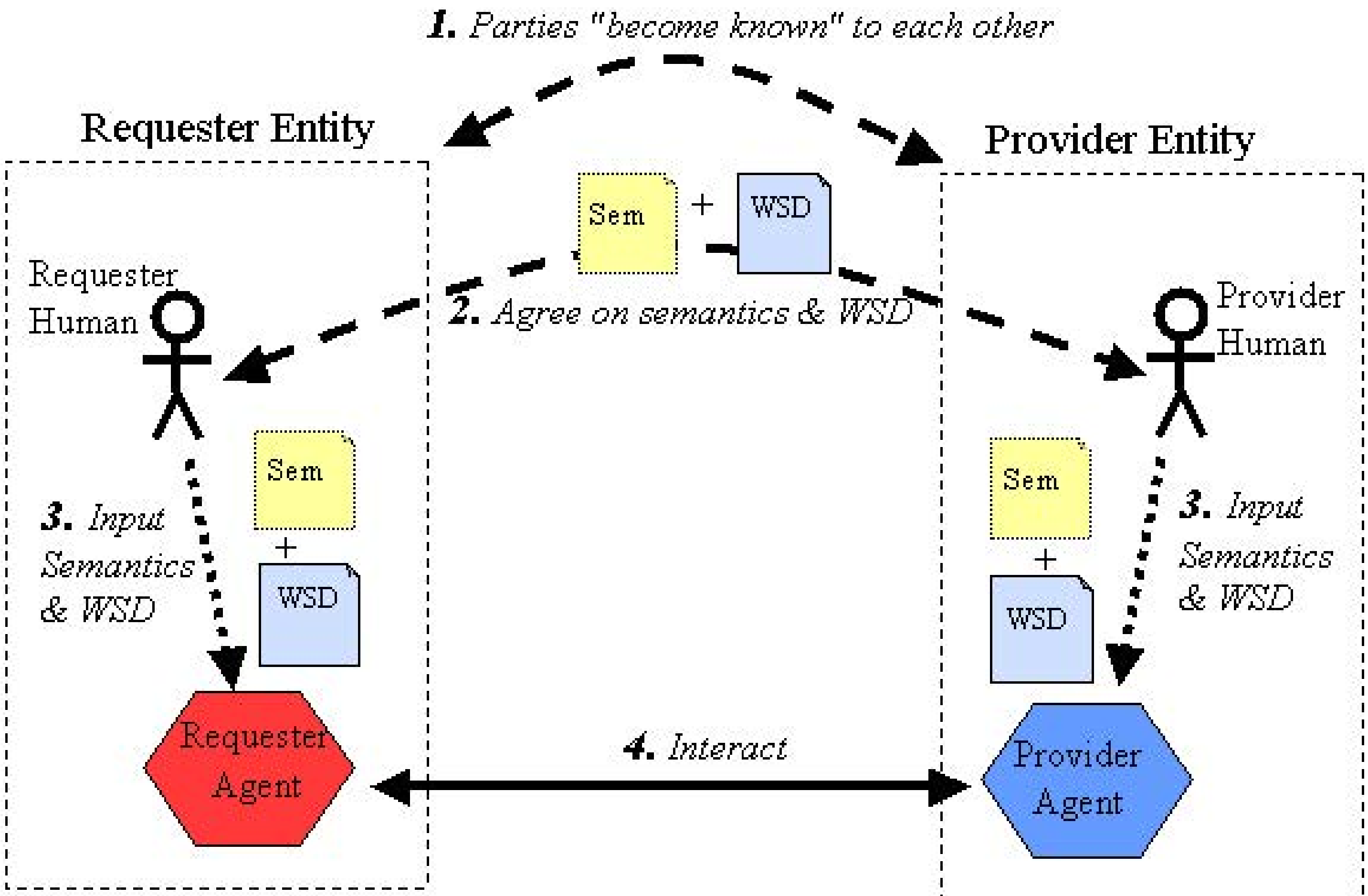
▶ **Driving force:** 'Business Needs' for **EAI** and **B2Bi**

Definition V.1: (W3C WSA Group 2004)

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. ♦

Webservice Model: Generic Usage Scenario

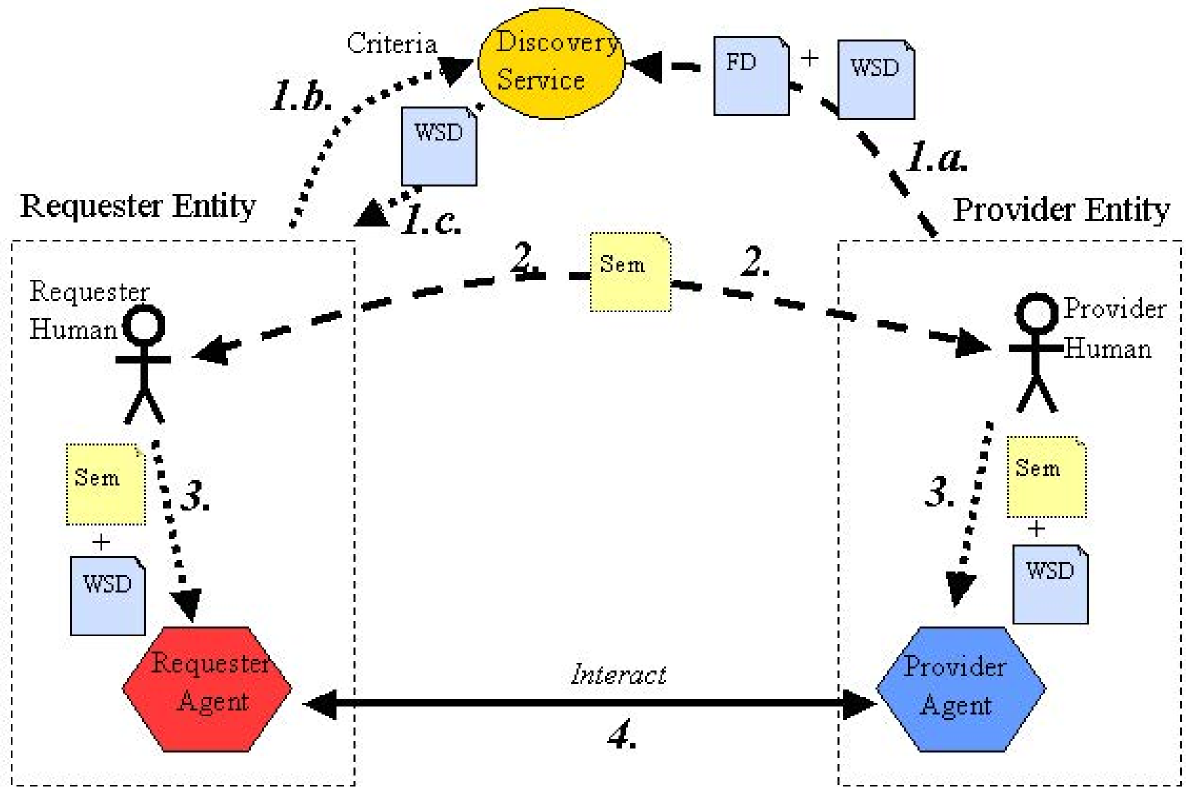
www.
w3.org/
TR/2004
NOTE-
ws-arch-
2004
0211
Fig 1.1



▷ **Roles:** Requester vs. Provider for a single interaction

◁ generic paradigm, esp. regarding implementation of step 1.

Practical Usage Scenario using Discovery Service



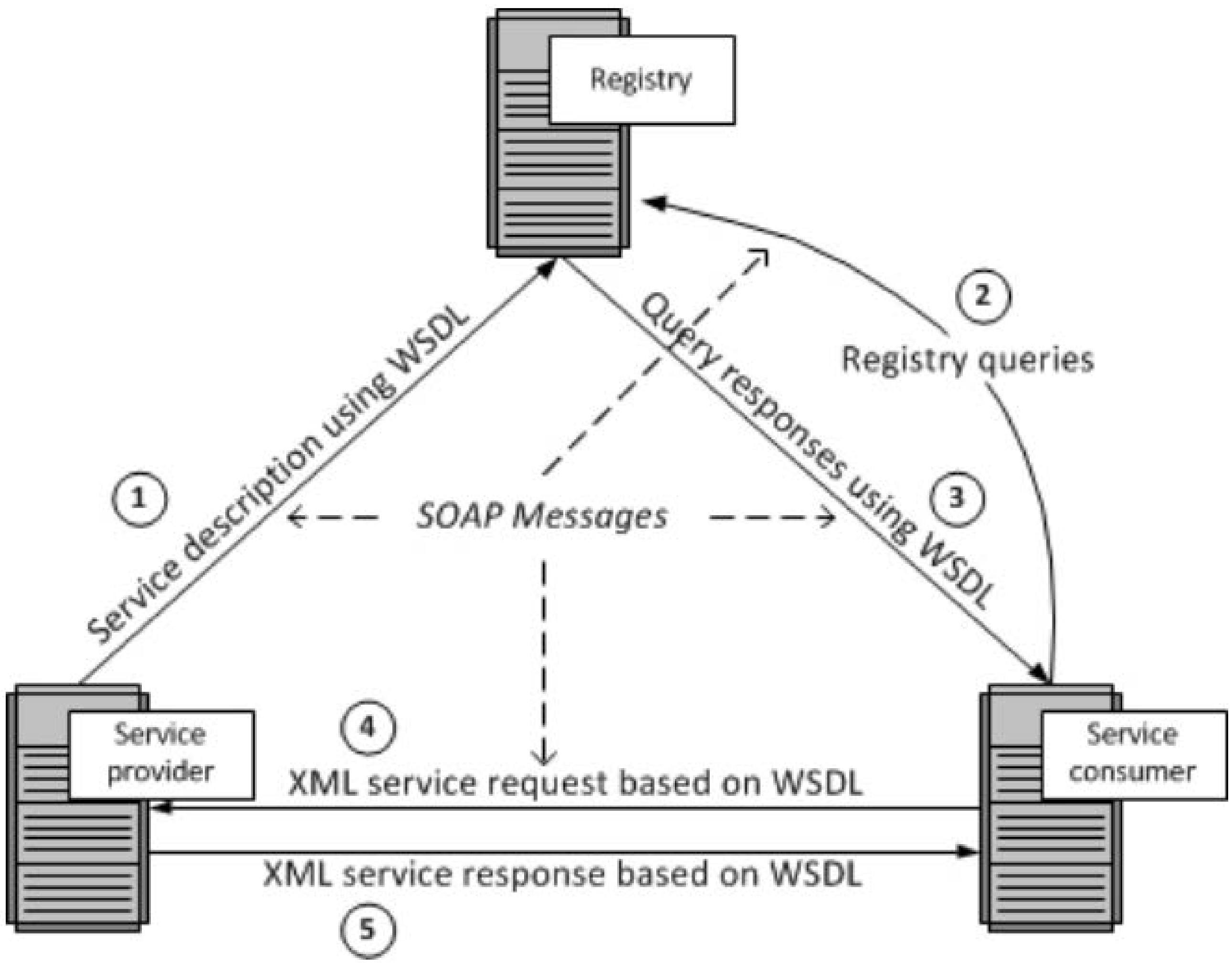
www.w3.org/TR/2004/NOTE-ws-arch-20040211 Fig.3.2

- 1.a: Provider exports functional description FD for DS
- 1.b: Requester looks up criteria in DS
- 1.c: Requester gets description(s) from DS
- 2: Requester and Provider negotiate w.r.t. service usage
- 3./4.: Requester and Provider interact to execute Webservice

c.f. pg. V-43

WS Reality: Participants, Protocols and Formats

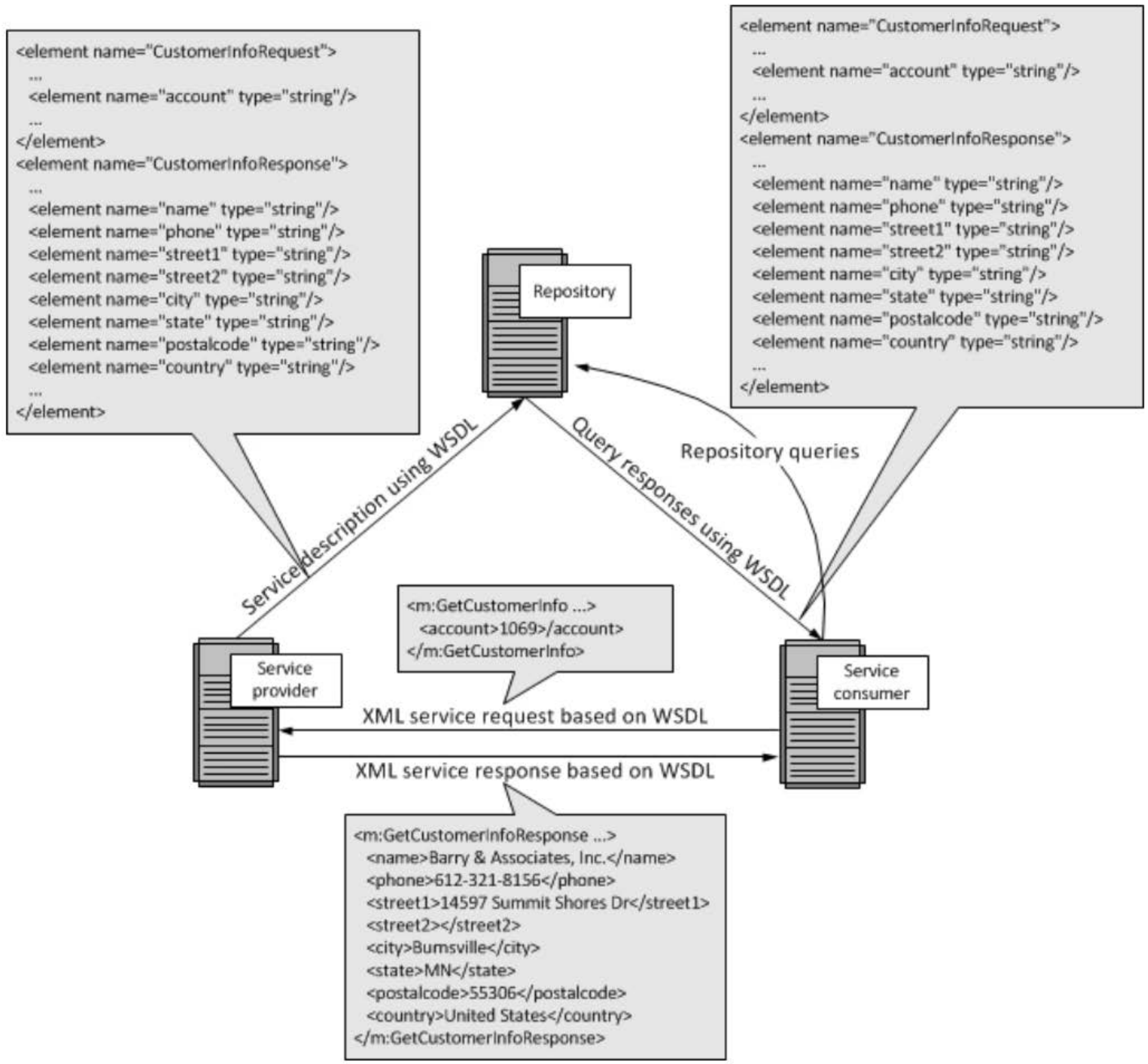
www.service-architect.com/web-services/articles/web_services_explained.html



- ▷ How to find your service' address? Directory/Registry Service for service publishing and finding vs. 'known' address, e.g. (IP, port)
- ▷ C/S interaction using a Remote Procedure Call via SOAP

Example: Interaction Information in a WSDL file

www.service-architect.com/web-services/articles/web_services_explained.html



excerpts only

Service-Oriented Architecture

Service-Oriented Architecture (SOA): more than Webservices

- Complete architectural paradigm for distributed systems
- Ranges from business processes to detailed implementation
- Extends to '*Service Landscapes*' or '*Service Eco Systems*'

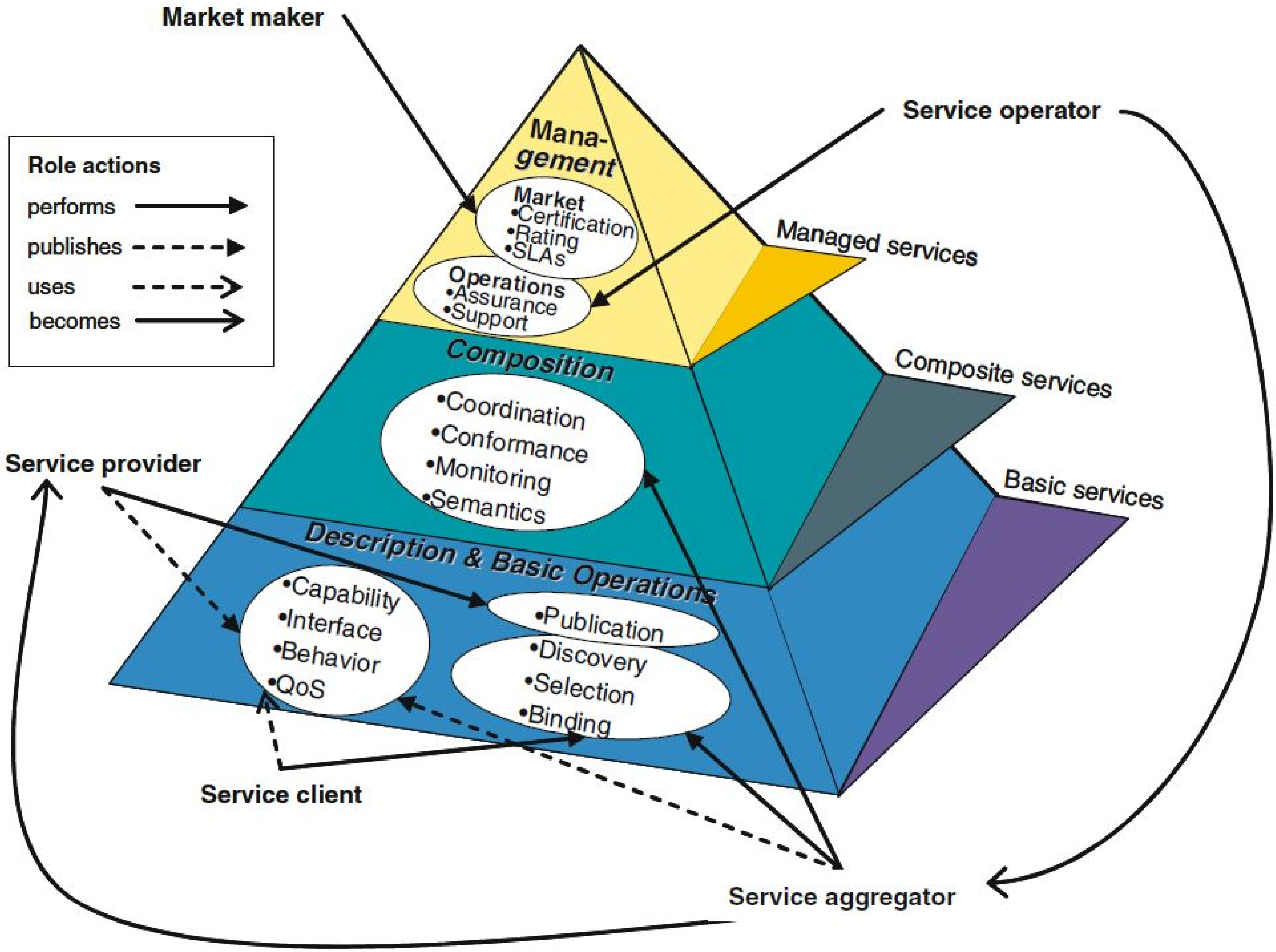
From the point of view of:	SOA is
Business executive and business analyst	A set of services that constitutes IT assets (capabilities) and can be used for building solutions and exposing them to customers and partners
Enterprise architect	A set of architectural principles and patterns addressing overall characteristics of solutions: modularity, encapsulation, loose coupling, separation of concerns, reuse, composability, and so on
Project manager	A development approach supporting massive parallel development
Tester or quality assurance engineer	A way to modularize, and consequently simplify, overall system testing
Software developer	A programming model complete with standards, tools, and technologies, such as Web services

Note: *Boris Lublinsky: Defining SOA as an architectural style.*
`www.ibm.com/developerworks/architecture/library/ar-soastyle/`

c.f. vc docs

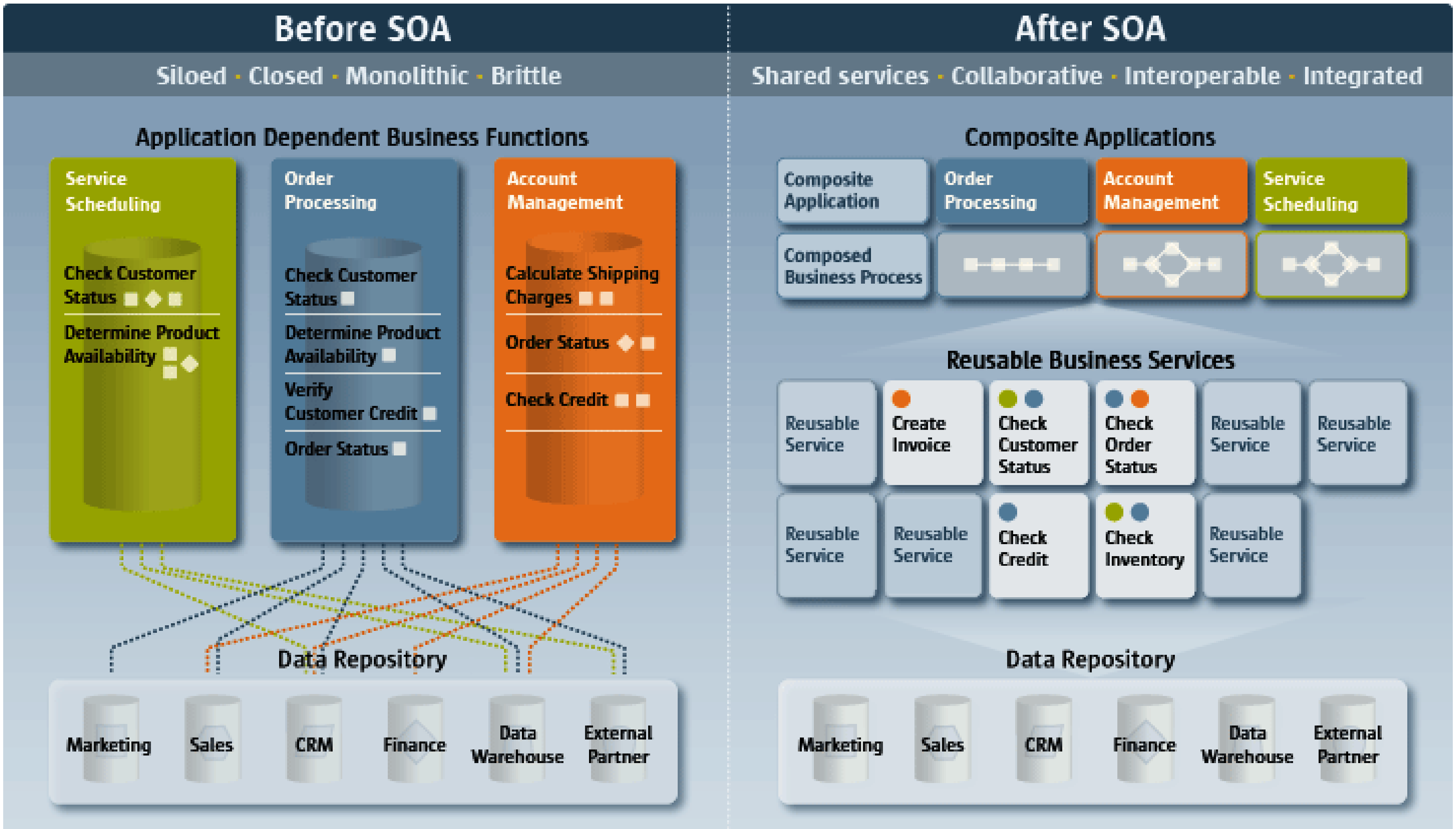
A 'holistic' view on SOA

Papa zoglu van Heuvel VLDB Journ. 2007



Vision – 1: Expected Impact: SOA-fication on Enterprise IT

www.
the
server
side.
de



Vision – 2: Service Landscapes and Eco Systems

'**Service Eco Systems** *are electronic market places and emerge as a result of the shift toward service economies. The aim of service ecosystems is to trade services over the internet. The vision of service ecosystems is an evolution of service orientation and takes services from merely integration purposes to the next level by making them available as tradable products on service delivery platforms.'*

c.f. Barros, Dumas: The Rise of Web Service Ecosyste in: IT Professional, 8(5), 2006

SES infrastructure must provide support for service

1. discovery based on service descriptions
2. selection by comparing and evaluating descriptions
3. contracting, esp., *Service-Level-Agreements* (SLAs)
4. consumption by calling and running a service call
5. monitoring at runtime, esp., w.r.t. SLA fulfillment
6. profiling combines 4.-5. for further service evaluation

c.f. Scheit hauer, Augustin Wirtz: Describing Services for Service Ecosystems. in: ESBE 2008

Benefits of SOA vs. Status of Webservices?

- ▶ Holistic support from business level down to technological details
- ▶ Explicit integration of **contracts** into the core concepts
- ▶ Clean designed paradigm that
  - ∗ demands explicitly defined **interfaces** and type definition
  - ∗ demands explicitly defined **interaction protocols** up front
  - ∗ distinguishes between specification and implementation
  - ∗ de-couples program logic from transport issues

Status of Webservices and SOA in Practice

- ▶ SOA as an architectural style still important
- ▷ Webservice technology stack still in use, esp. enterprise integration
- ◀ Transport neutral text-based `SOAP/XML` is costly (heavy-weighted)
- ◁ WebService stack is complicated and not successfully standardized
- ◁ 'Holistic' approach and benefits are still hard to achieve
- ◀ Interface-based design via `WSDL` specification not 'mainstream'?

End of V.5

V.6. Interoperability and Standards

See Appendix V-7.1 and V-7.2

▷ **HTTP/1.1/HTTP/2** internet transport protocol (1999/2015)
more efficient interaction for webservers (pushing data) etc.

▷ **XML 1.1** (29.09.2006, `http://www.w3.org/TR/xml11/`)

- self-describing, machine-processable documents
- standardized formats for structured documents and schemata
- transformation rules, sophisticated navigation, ...

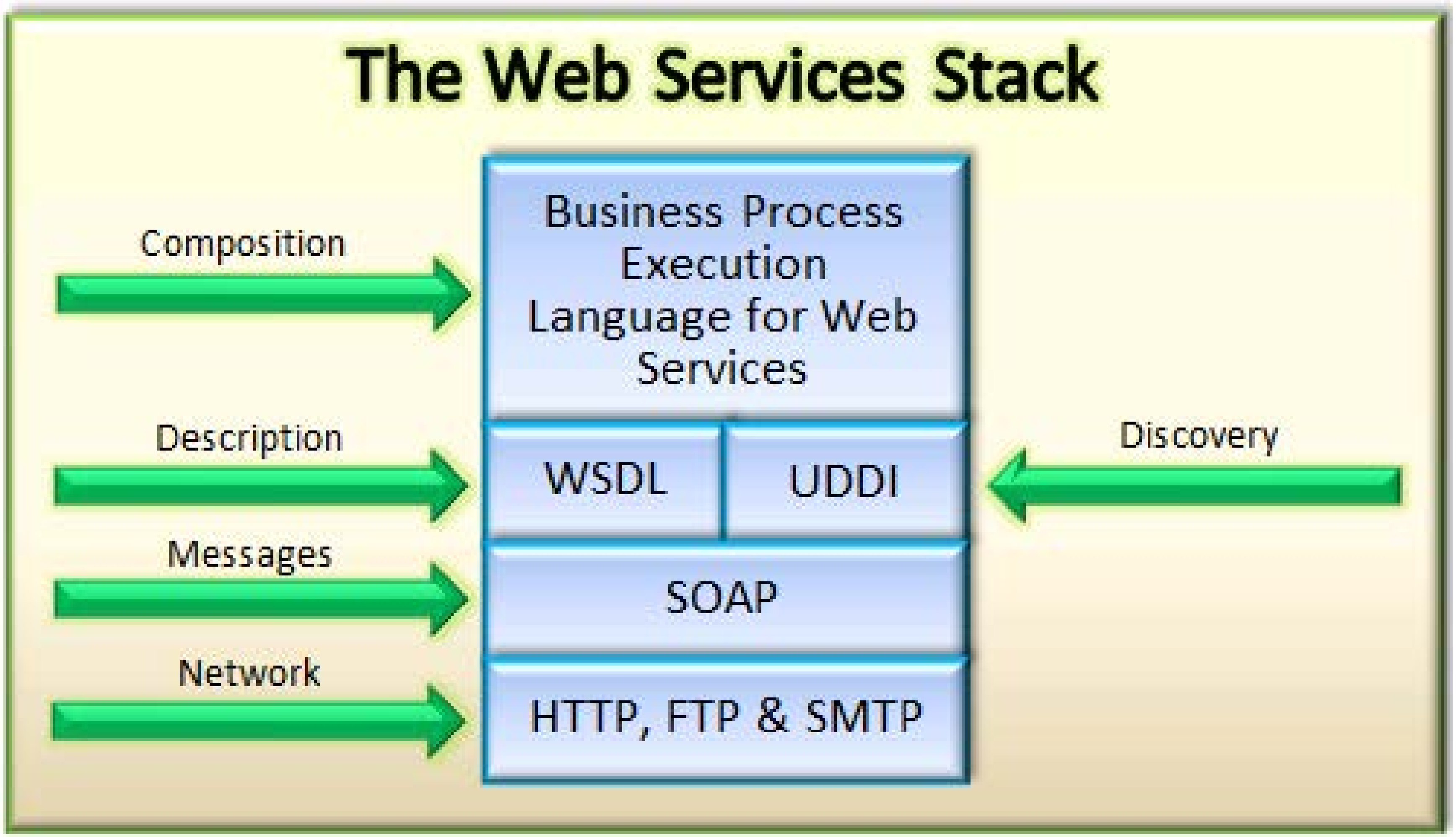
▶ SOAP 1.2 (27.04.2007, `http://www.w3.org/TR/soap/`)
Note: lot's of tools still work with version `1.1` today

- message structure and XML mappings
- Nodes and roles: Sender → Intermediary → Receiver
- Interaction protocols and predefined '*msg exchange pattern*'

▶ WSDL 1.2/2.0 (26.06.2007, `http://www.w3.org/TR/wsdl20/`)
Note: lot's of tools still work with version `1.1/1.2`

- defines data/msg types, interfaces, ops, mapping to XML

Webservice Stack – Overview: Lower Layers



projectraja.com/images/pages/WebServicesStack.png

Webservice Standardization: A Lesson how to fail

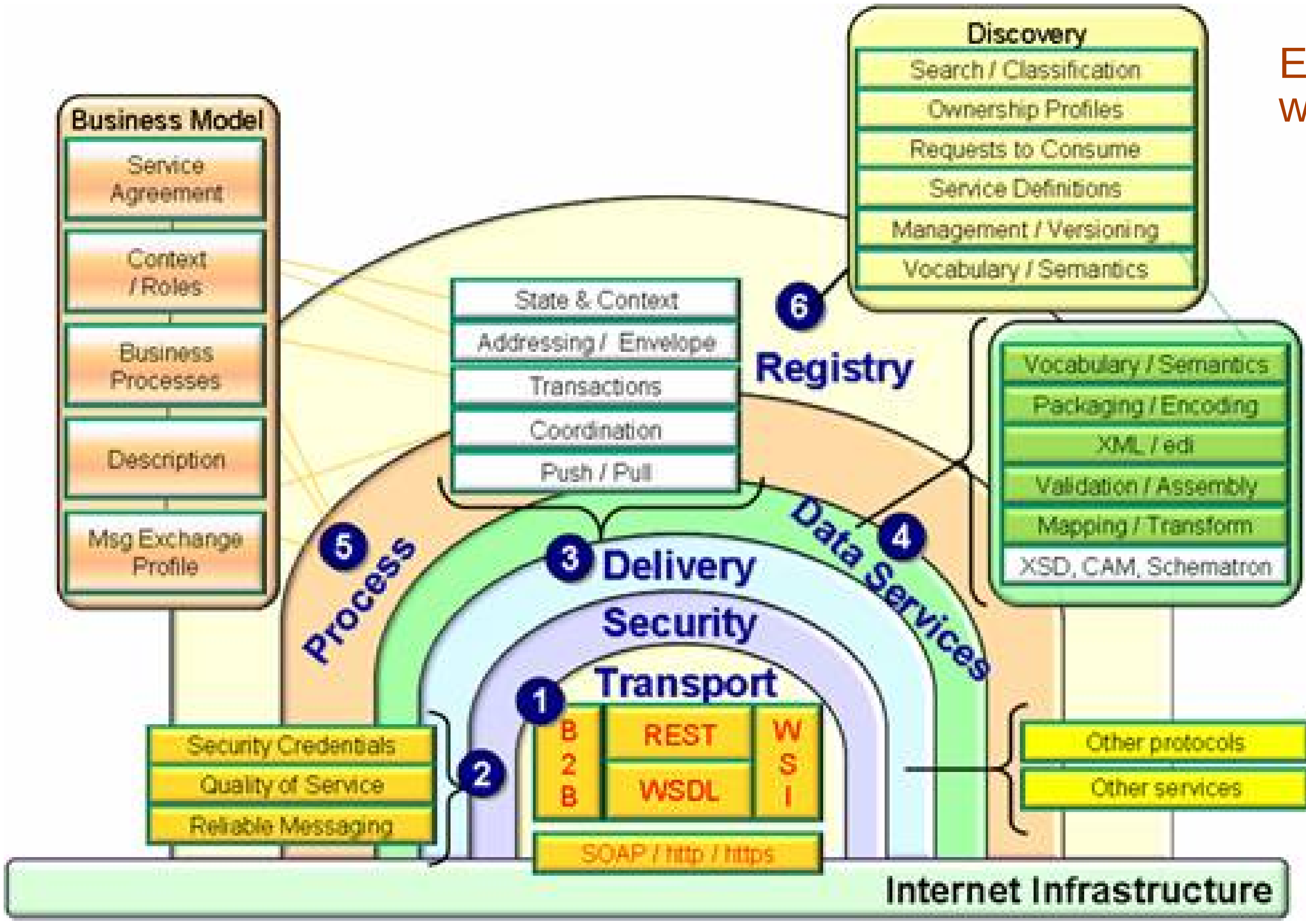
www.ws-i.org

www.oasis-ws-i.org/

Web **S**ervice **I**nteroperability Profiles 1.2/2.0 09.11.2010

- ▷ standard compliance between standards of different organizations
- ▷ Additional rules eliminating standard inconsistencies etc.
- ▷ Use-Cases and test-suites for checking conformance
- ◀ Initiative was cancelled in 12/2017

Reasons for Standardization Failure: Complexity



End of ADD-On w.r.t. SOA details

▷ Big Picture: Handling lots of different architectural aspects . . .

◀ . . . requires lots of different inter-operability standards?

End of V.6

Summary: How to build Modern C/S Systems?

NOTE: This slide is discussed now at the end of section V.4 (end of REST)

When to use which technology? No one-fits-all answer!

- ▷ **WSDL**-based Services
  - ∗ contract-based, high overhead, complex msg and environment
  - ∗ still ok for EAI in 'closed', 'statically planned' environments
- ▶ **RESTful** Services:
  - ∗ dynamic 'resources', overhead $<$ WS, msg still 'clumsy text'
  - ∗ based on Web standards without too much a priori customization
- ▶ **gRPC**-based Services
  - ∗ contract-based, faster than REST due to binary msgs
  - ∗ more flexible interaction styles (esp. `stream`-handling)
  - ∗ higher requirements w.r.t. environment: `HTTP/2`, `SSL`, ...

First Humble `gRPC` $\Longrightarrow$ *performance, inter-application, clouds*
Guess: `REST` $\Longrightarrow$ *inter-op., loose coupling, rapid change*

End of V

Complex Standards - The Way of the World?

fox deploy. files. word press. com/ 2014 /11/ stan dards-1. png

V.7.1 Appendix: *Basic Standards – HTTP*

Note: http and XML are also discussed in 2 videos as they are also important for REST etc.

HyperText Transport Protocol 1.1/**2.0**

tools.ietf.org/html/rfc7540 2015

- **Adressing** based on standardized rules

  `http://<host> [ :<port> ] [ abs_path [ ? <query> ]]`

- **Data formats and encodings** for transmitting data
  - ∗ ASCII text and compressed sequences of bytes
  - ∗ **M**ultipurpose **I**nternet **M**ail **E**xtensions: `text/plain`; `image/gif` etc.

- Supported **interactions steps** and permitted formats
  - ∗ Request–Msgs allow for 9 pre-defined methods/formats
  - ∗ Response–Msgs: information (1xx), success (2xx), redirection (3xx), client errors (4xx), server errors (5xx)

www.restapitutorial.com/httpstatuscodes.html

- **Connection Model:** (*'philosophy' vs. 'efficiency'*)
  - ◁ `HTTP 1.0` connections-less: new connection for each message
  - ▷ `HTTP 1.1/2.0`: `persistent` connection; better performance default

HTTP Command 'Verbs'

`GET`
: The `GET` method requests a representation of the specified resource. Requests using `GET` should only retrieve data.

`HEAD`
: The `HEAD` method asks for a response identical to that of a `GET` request, but without the response body.

`POST`
: The `POST` method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

`PUT`
: The `PUT` method replaces all current representations of the target resource with the request payload.

`DELETE`
: The `DELETE` method deletes the specified resource.

`CONNECT`
: The `CONNECT` method establishes a tunnel to the server identified by the target resource.

`OPTIONS`
: The `OPTIONS` method is used to describe the communication options for the target resource.

`TRACE`
: The `TRACE` method performs a message loop-back test along the path to the target resource.

`PATCH`
: The `PATCH` method is used to apply partial modifications to a resource.

HTTP – Properties of Command 'Verbs'

Fig.: en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

HTTP method	RFC	Request has Body	Response has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	Optional	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	Optional	Yes	No	Yes	No
CONNECT	RFC 7231	Optional	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

- **Safe**: method is (by definition) guaranteed to have no side-effects on server side, i.e. reads or 'retrieval' only
- **Idempotent**: multiple *identical* requests have only one effect, i.e. repetition does not change server side
- *Cacheable*: if still valid, avoid re-transmit etc. (detailed control)

Main Issues: HTTP/2 ⟷ HTTP/1.1

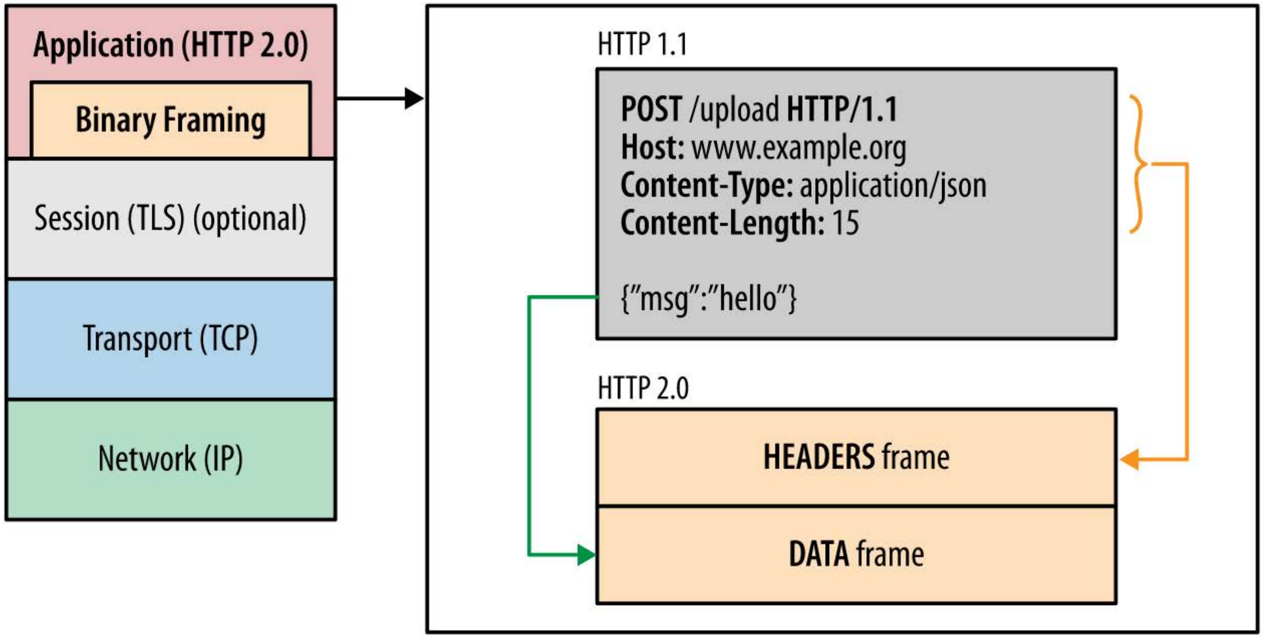


Fig.: developers.google.com/web/fundamentals/performance/http2

- plain text vs. binary encoding
- multiplexing multiple streams on a single TCP connection in parallel vs. multiple TCP connections in case of multiple streams
- security overhead using TLS/SSL much less for single connection
- streams may be prioritized based on dependencies
- advanced *flow control* in order to avoid buffer overflows

Remark: Details also important for gRPC *and esp.* REST

HTTP/2 – Example Request/Response using Get

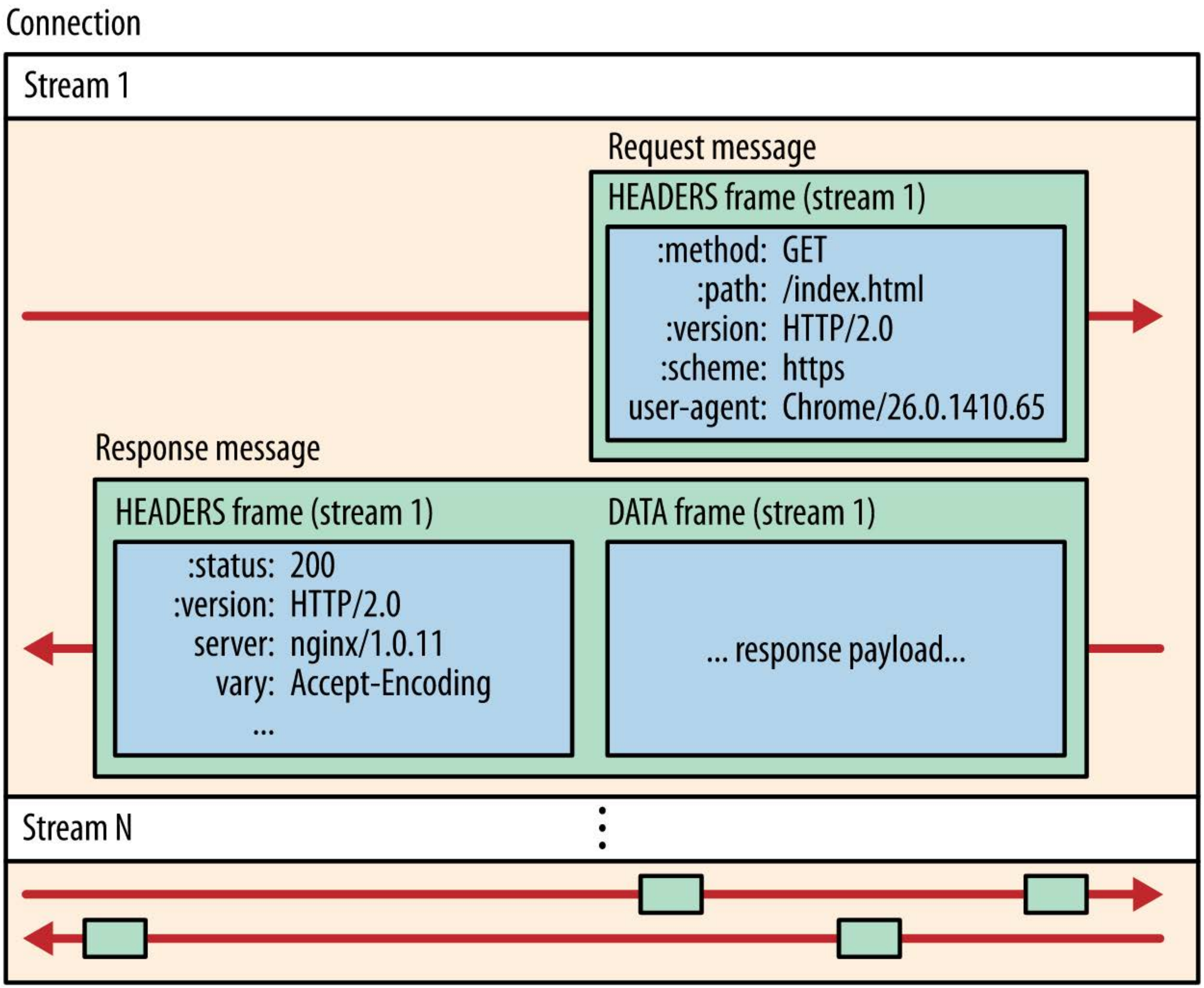


Fig.: developers.google.com/web/fundamentals/performance/http2

V.7.2 *Basic Standards – XML*

eXtensible **M**arkup **L**anguage: text-based language

- **Basic Concepts** ...
  - self-describing document structures holding meta-information
  - separation of structure, content and form

  ... **ease**

  - ▷ consistency checks: document vs. structural description
  - ▷ automatic processing (parsing) based on format definitions
  - ▷ handling the same content in different representations

- **Background:** long tradition of **hypertext** documents Nelson 1950
  - extends `HTML` (1989) by self-defined Tags
  - Restricts the `SGML` ISO standard (1986)
  - lots of additional standards and tools for XML processing

  `XML 1.0/1` standard defined by **W3C** still in use

  (`XML 1.0 1st 1998/4th 2006; XML 1.1 29.09.06`)

XML Basis for structured messages

- **XML Vocabulary**: predefined set of elements/structures
  - nested complex structures via open/close `markups` $\Longrightarrow$ **Tree**(s) c.f. pg. V-65
  - **Structure** may be specified using different techniques:
    - ◁ **D**ocument **T**ype **D**efinitions (DTDs) similar to grammars and restricted to structure definitions (*out-dated*)
    - ▷ **XML Schemata**: structure specification in XML; supports type definitions, (nested) name spaces; extensibility c.f. pg. V-66

    $\Longrightarrow$ **Documents can be evaluated with respect to:**

    - ∗ **well-formed-ness:** comprise to general **XML** specification
    - ∗ **validity:** document matches predefined definition or schema
- Powerful **XML processing tools**:
  - **S**imple **A**PI for **X**ML/**D**oc **O**bj **M**odel/**St**reaming **A**PI for **X**ML
  - `JAXB`: un/marshall Objects $\leftrightarrow$ XML and `JAX-WS` DSG-AJP-B
    Annotations in `Java`-Code define attributes, . . ., webservices
  - **Transformation:** e**X**tensible **S**tylesheet **L**anguage/**T**ransformation

XML File complying to an XML schema definition

c.f. www.w3schools.com/schema/schema_example.asp

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

The corresponding XML schema definition

c.f. www.w3schools.com/schema/schema_example.asp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

End V.7

IT sucks . . . for Management and Financing

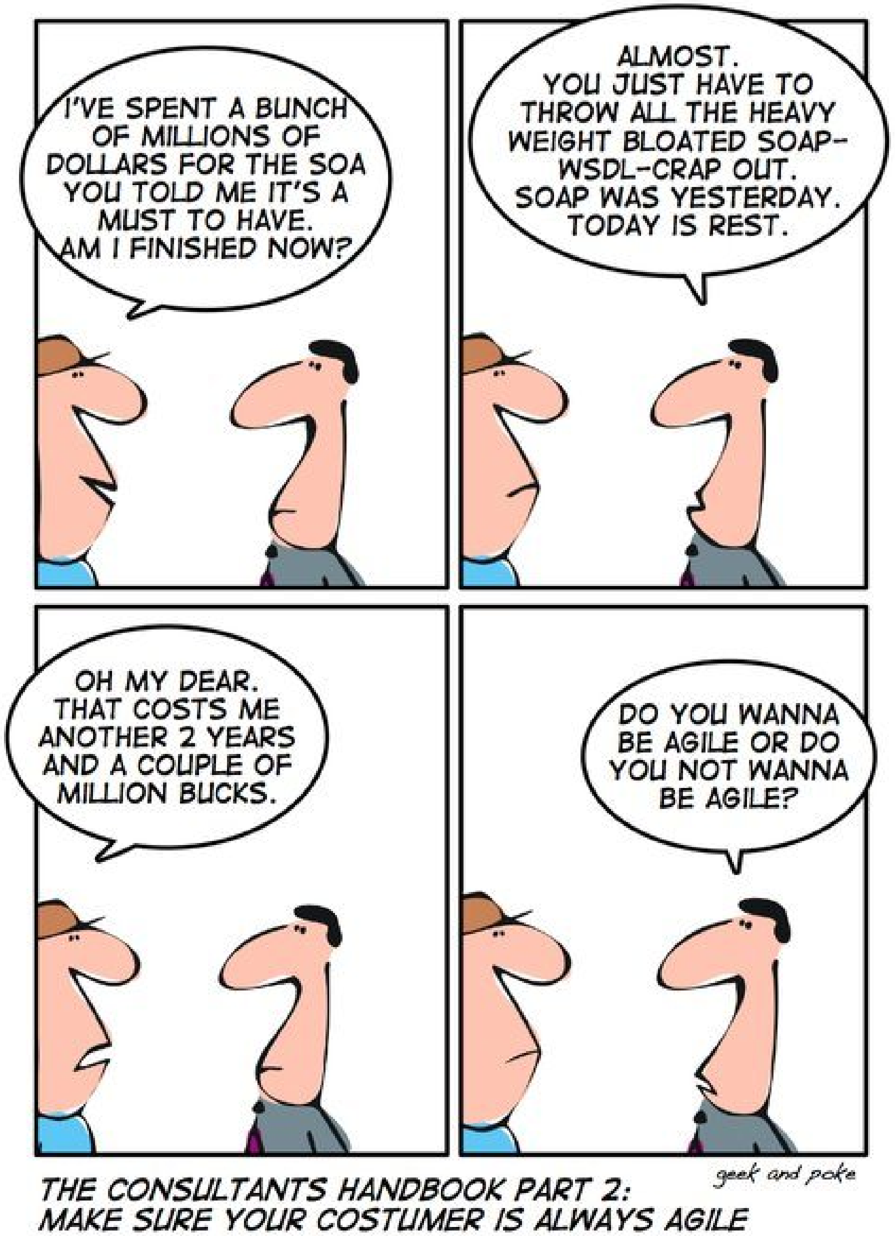


Fig.: cdn.crunchify.com/wp-content/uploads/2013/07/REST-Over-SOAP-Protocol.jpg