Mestrado Integrado em Engenharia Informática
Dependable Distributed Systems

Confiabilidade de Sistemas Distribuídos

Project Assignment #2

2021/2022, 2nd SEM

Intrusion-Tolerant Decentralized Ledger Platform based on a Permissionless Blockchain Solution

BITDL

or … Blockchained ITDL

PA#2 Delivery

- Initial Presentation: 6/May/2022
- Delivery: Submission Period (Open Form): 9 to 16 /June 2022
  - Note: Test 2 Date: 14/Jun/2022, Room 128-II, 14h00
- Submission Deadline: 16/June/2022
  - Github Repo (shared w/ “henriquejoaolopesdomingos”
  - Submission Process: Similar to PA#1
    - Individual Delivery Form (Google) access by URL to be announced
      - Submission materials
        - » Github ProjectRepo (URL) w/ source codes or runtime components, ready for cloning, installation and evaluation
        - » README with all necessary indications for configs, installation and deployment
      - Characterization of the delivered solution and checkout of the implemented requirements vs. specified requirements
      - Possible Quick quiz/initial clarification on concrete implementation and involved background

PA#2 Workgroups

- Initial groups of 2 students (as in PA#1)
- Individual students
  - Possible merging process w/ groups w/ 2 students
  - Fixed from FRI, 6/May, until WED, 11/May

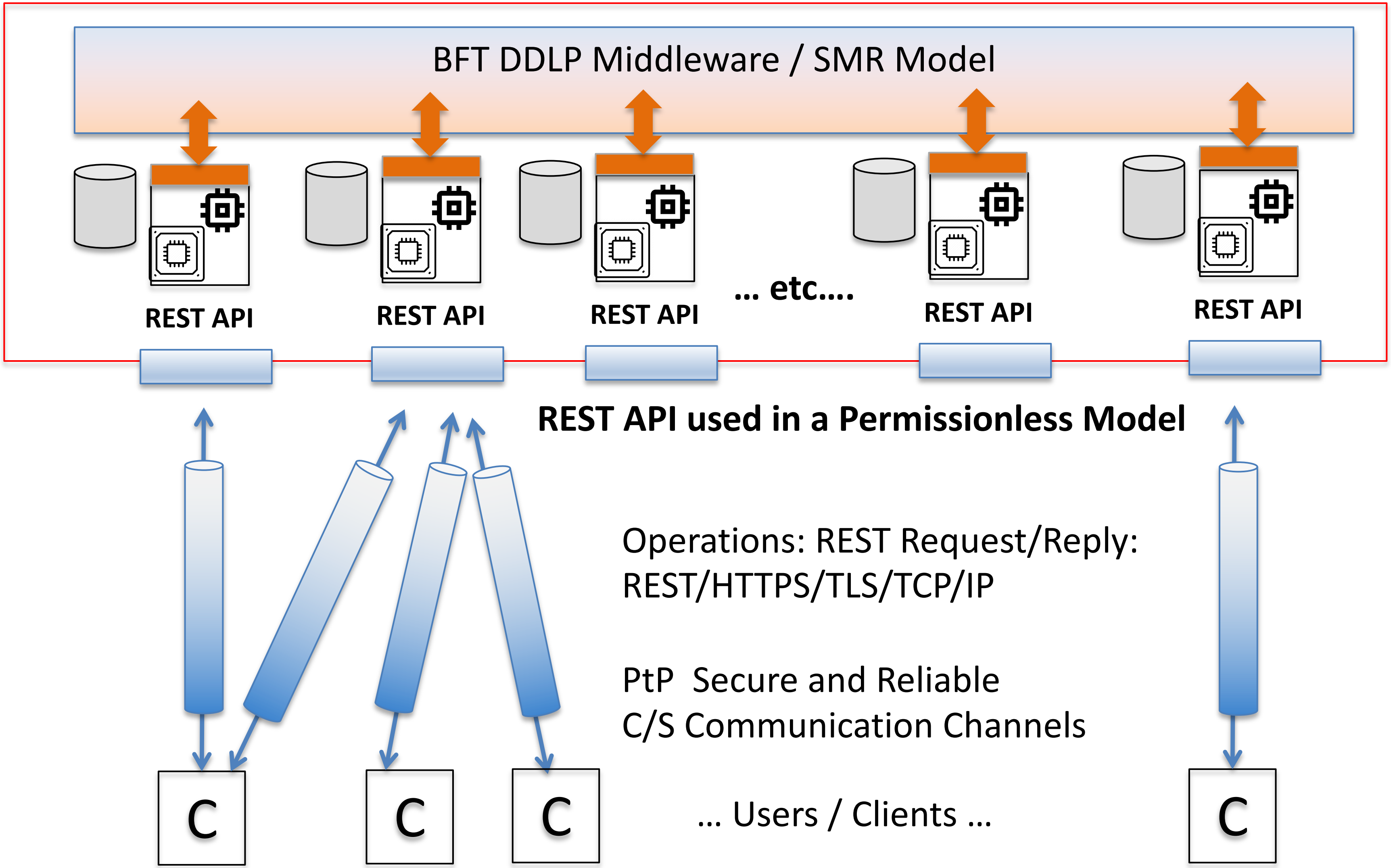
PA#2: Pre-Requirements

- Sequence of PA#1 Goals and (possible) consolidation
- Evaluation Reference for Practical Evaluation:
  - PA#1 (max, 40%) => 8/20
  - PA#2 (max. 60%) => 12/20
- Extension of PA#1 with new requirements proposed in two groups:
  - M: Mandatory (Common Requirements): 8/20
  - OV: Optional/Valorative: 4/20

- **Overall (after PA#2): 8 (PA#1) + 8 (PA#2-M) + 4 (PA#2-OV)**

Architecture (PA#1 reference review)

Consistency and Byzantine Fault Tolerance for Intrusion Tolerance



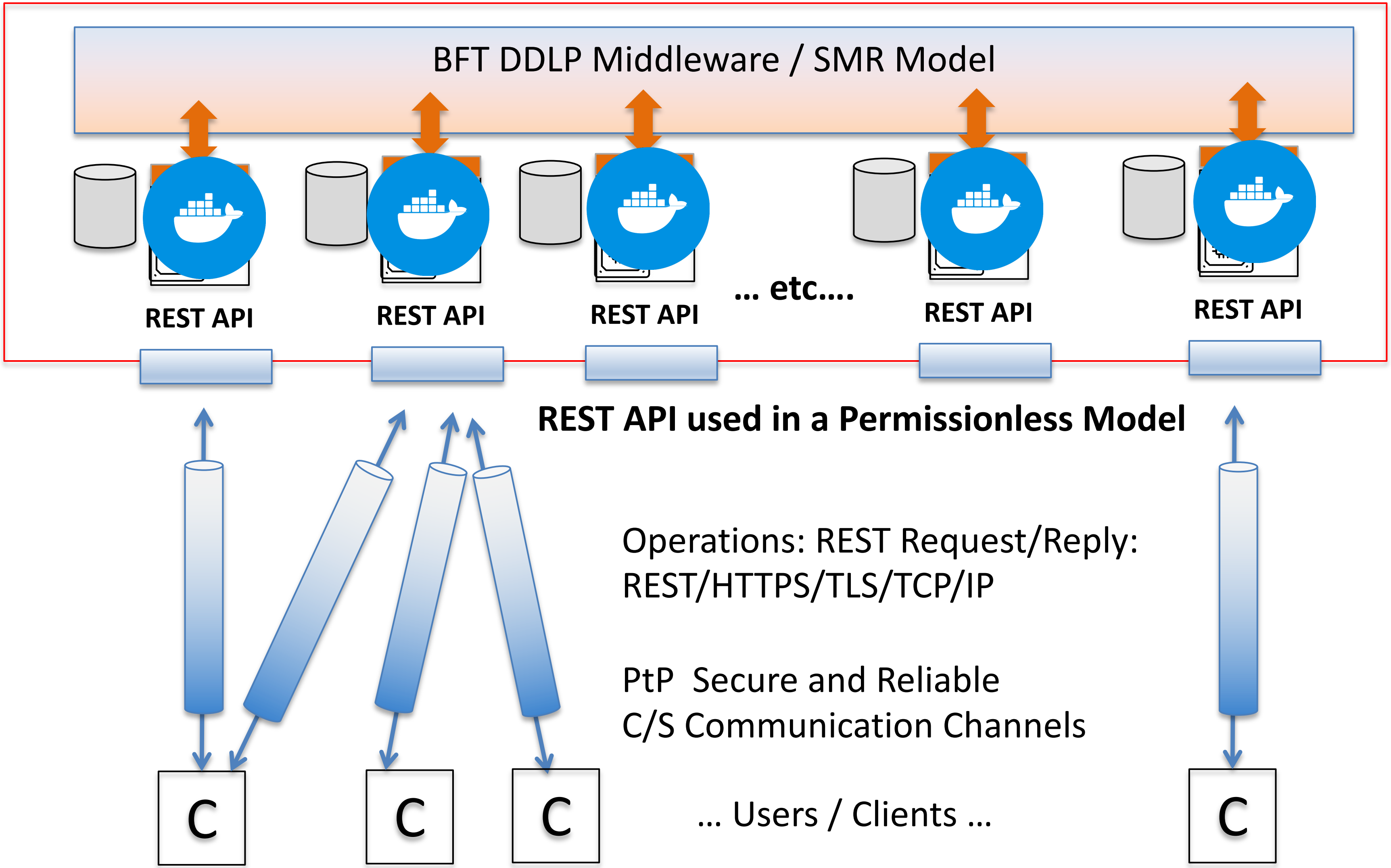
REST API used in a Permissionless Model

Operations: REST Request/Reply:
REST/HTTPS/TLS/TCP/IP

PtP Secure and Reliable
C/S Communication Channels

... Users / Clients ...

Containerized Service Solution (Docker Enabled Design and Implementation)



BFT DDLP Middleware / SMR Model

REST API

REST API

REST API

… etc….

REST API

REST API

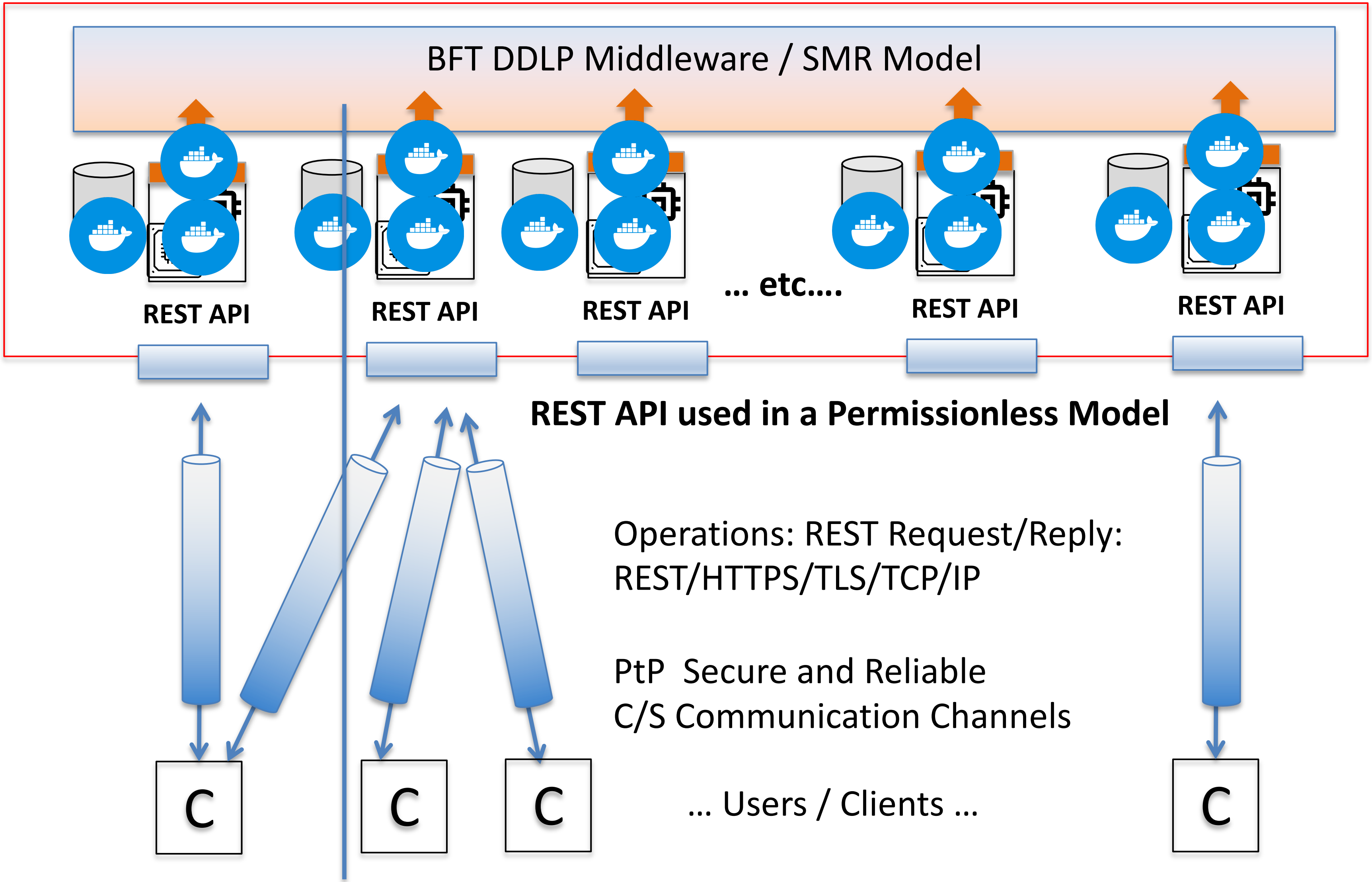
REST API used in a Permissionless Model

Operations: REST Request/Reply:
REST/HTTPS/TLS/TCP/IP

PtP Secure and Reliable
C/S Communication Channels

… Users / Clients …

Composition of Containers (Docker Enabled Design and Implementation)


BFT DDLP Middleware / SMR Model
REST API
REST API
REST API
... etc....
REST API
REST API
REST API used in a Permissionless Model
C
C
C
C

Operations: REST Request/Reply:
REST/HTTPS/TLS/TCP/IP

PtP Secure and Reliable
C/S Communication Channels

... Users / Clients ...

PA#1 Review:
Open Issues,, Limitations and Drawbacks
as “Input Ideas” for PA#2

Discussion: PA#1 Open Problems (1)

1) Servers implementing the API for client-operations: No BFT Assumptions

2) No durability guarantees of servers (Fail-recovery mechanisms and required state-transfer and recovery of the ledger correct state)

3) Insufficient analysis on latency, throughput of operations, with a better benchmark evaluation of client-operations (workloads) and service behavior (w/ different observations), ex:

> # Clients, Parallel Workloads
> Scale conditions (# BFT replicas)
> Fault conditions (BFT SMaRt, Regular operation, Fail Stop conditions, Fail-Stop & Recovery conditions, BFT behavior)
> Comparative performance of order vs. unorder operations
> What more ? (ex., cryptographic operations, BFT SMaRt configs)

Discussion: PA#1 Open Problems (2)

4) No Blockchained Solution (limitations in current ledger implementation – in-memory vs persistency, w/ no solution to deal with possible attacks on reversibility, tampering, deleting of “ledgered” operations

Imply that state-verifications must be supportd by the BC-Ledgeres

5) No observation of possible impact or advantages in using “asynchronous” operations (on the BFT SMaRt environment, as an option for possible invocations of operations)

6) Packaging of the solution modularized “as a service”: Docker-Composed solutions of service components: Servers, BFT Smart Service Replicas and Persistent Ledger

7) Lack of appropriate support for modularization and possible isolation of the main service components: servers, BFT-SMaRt process-replicas, ledger/storage, in TRUSTED EXECUTION ENVIRONMENTS (ex., Hardware-Backed)

Discussion: Other PA#1 Limitations: possible space for interesting improvements

8) Smart-Contracts: can we support operations (ex., transactions) with implicit “smart-contract” for validation, at execution time ?

9) Support for Edged-P2P Transactions (Client-to-Client), to support P2P Transfers

10) Support for “privacy-enhanced transactions) in supporting P2P transfers (as 11)

11) Better scalability conditions with an improved Ledger supported by the Blockmess Approach

12) Some other idea(s) / Improvement Highlight(s)?

How to address the challenges, open problems enhancements and solutions’ complexity

DISCUSSION ON THE REQUIREMENTS:

	Less Difficult? More commonly Addressable ?	Extended or Complex ?
1) …	**X**	
2) …		**X** (complex) !!!!
3) …	**X**	
4) …	**X**	
5) …		**X** (design requires 1, 3 and 4)
6) …	**X (**after 1, 3, 4)	
7) …		**X** (w/ Isol. Comp and TEE)
8) …		**X**
9) …		**X**
10) …		**X** (requires homomorphic enc.)
11) …		**X** (Requires Blockmess Integ.)
12) … ??		

Commonly Addressable Requirements
(for the common addressable challenges: 1, 3, 4, 6) ?

1) … a) Async. Invocations, Multiple (2f+1) signed responses, verif. by the client
or
b) Client operations on a byzantine quorum set of servers (no so easy ? Why ?)

3) … Support for benchmarking and analysis of experimental evaluations

4) … Requires the extension of the API, a designed solution for Blocks and Blockchains
Block-Structure: Ex., inspired on Blockchain Blocks ad in Bitcoin with
Block Header elements and "Tree-Structure" aggregating Transactions
Block-Finalization: Client-Mined Blocks and Proof-of-Work based consensus
Blockchained Ledger: can be operated "in memory" for performance and w/
Persistency (as base assumption for possible Fault&Recovery model of Servers

6) Requires a docker-composed solution, ready for deployment (example in a Cloud-Enabled node) and experimental analysis conducted on such environment

PA#2 Approach and Requirements

Rational for PA#2

- Commonly addressable requirements as mandatory PA#2 requirements (MRs)
  - 4 Requirements
    - See before: 1), 3), 4) and 6)

- Extended requirements as optional/valorative requirements for PA#2 (VRs)
  - Must choose to address 2 of the proposed requirements
    - See before: 2), 5), 7, 8), 9) 10, 11

PA#2: Requirements

- Extension of PA#1 with new requirements proposed in two groups:
  - MR: Mandatory (Common Requirements): 8/20
    - 4 Extensions w/ Related Requirements
    - 2 Points per Designed/Implemented/Demonstrated Requirement
  - VRs: Optional/Valorative: 4/20
    - 2 Points per Designed/Implemented/Demonstrated Requirement

- **Overall (after PA#2): 8 (Consolidated PA#1) + 8 (PA#2-MR) + 4 (PA#2-OV)**

PA#2 MR Requirements

Note: For all PA#2 Mandatory Requirements you don’t
need additional or
Particular new Tools, because all the requirements can be
addressed as extensions
or refinements of the previous PA#1 requirements

PA#2: Mandatory (or Common) Requirements

- **Mandatory: 1), 3), 4) and 6):**
  - **1): Support for Byzantine-Servers**: Inclusion of support for prevention of Byzantine Servers, extending the Server API to support Asynchronous Invocations and Responses with multiple authentication evidences of BFT SMaRt replicas
  - **3): Blockchained Ledger**: Extension of Server API allowing clients to manage and mine Blocks of Transactions and Extension of the current Ledger to address a Blockchained Ledger Structure
  - **4): Benchmarking Analysis** (Throughut vs. Latency, on Client Workloads) w/ comparative impact between Regular Operation, Fail-Stop&Recovery of BFT Replica and Dynamic Join of new replicas
  - **6): Solution as a Service** (Docker-Composable Solution)

1): **Support for Byzantine-Servers (1)**

- Design and implementation support for prevention of Byzantine Servers
  - Requires that the Server API must provide for clients variants of API operations, to be executed asynchronously, with asynchronous invocations on BFT SMaRt
  - These Invocations allow for the server to capture Multiple Authenticated Responses from the BFT SMaRt replicas involved
  - Responses can be controlled by the final clients as a valid ser of quorum-responses
  - The extension can involve only as required set of operations (in the API leveraged from the PA#1), ex:

What operations make sense to consider in the initial ref. API?

- createaccount()
- getbsmoney()
- sendtransaction() X
- getbalance() X
- getextract()
- gettotalvalue()
- getglobalvalue()
- getledger()
- (what else in your API ?)

3): **Blockchained Ledger**

- Requires:
  - Extension of Server API allowing clients to ask servers for a Block (A block structure of transactions to be mined)
  - Clients will mine Blocks with a PoW model
  - Clients proposed the mined Blocks to the used server
    - Incentivization for Clients: Verified/Finalized Mined Blocks transfers an amount for the respective Client Account
  - Server must validate the mined block and the mined block must be replicated to the other servers, to include the block in a blockchain structure
  - Guidelines for the implementation: Blocks, Challenges/PoW for Block mining and Blockchain structure inspired in the Bitcoin case (Block Header and Block Body with an Hash/Merkle tree of included transactions)

At least two more operations are required in the Service API for Clients

- Block = getBlockToMine()

  Obtain from the server a Block, to be mined w/

  PoW by the client. The Bock structure is free but must be considered a structure inspired in the Block structure of Bitcoin/Blockchain Blocks, with required Block-Header elements and a Tree-Based Contained Transactions and their Hashes

  Ex., Consider a fixed number of transactions, ex., 8 or 16, for example

- verificationStatus= proposeMinedBlock(Block)

  Propose mined block for server verification, replication and finalization in the blockchained ledger (in-memory vs. persistent leger on all servers)

Block and Blockchain (as in Bitcoin/Bockchain)

- See also materials (discussion) from Lectures
- Bitcoin Block specificatioin
  - https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch07.html
  - Note some simplifications:
    - 1 transaction is 1 transaction (not a set of input and output transactions as in a Bitcoin Block)
    - You don'tneed the version number (or you can use a version number = 0x0001, by default
  - You can maintain (probably) the Transaction (Operations) Structure, as you have in PA#1
    - But you can change it if you want

4): **Benchmarking Observations**

- Requires:
  - Benchmarking tool that can submit workloads of operations through the API provided by a server
  - Observations
    - Latency vs. Throughput Observations (see the experimental analysis of BFT SMaRt paper as reference)
      - Servers always perform correctly
      - Reference workload: (write vs. read reference operation related to the provided API)
    - Comparatives (variation on observations):
      - Workload Regular operation (non failures)
      - Operation on failstop conditions: BFT SMaRt replicas
        - » 3f+1, 1 fail stop !
      - Operation on fail&recovery settings (BFT SMaRt replicas)
        - » 3f+1, 1 fail-recover (will include the BFT SmArt state transfer impact)
      - Operation on dynamic setting (BFT SMaRt):
        - » 3f+1, 1 fail, 1 added (will include the BFT SmArt state transfer impact)

Note: we will not consider BFT Assumptions for the Benchmark. Complicated ?

6) **Packaging & Deployment for a Solution aaS**

- Requires:
  - – Docker-Composable Components
  - – Separate/Isolate the main Service components:
    - Servers (Still, possible configurations outside)
    - BFT Replicas (Still, configurations outside)
    - Persistent Blockchained Ledger (possible configs outside)

VR Options

Note: To address some (not all) Valorative Requirements, you need additional tools (see the next slide)

VR Options

- First of all, choose your valorative options
- Analyze the requirements, clarify and discuss (in lab/class) what is involved, how to address the challenge and the research/design/implementation materials you need to research and use

Note:

2) Don’t need new particular tools, but probably complex !

5) , 8), 9) – don’t require new particular support

For 7), 10 and 11) you need additional tools (ask for them):

- 7) Need (Sugg): https://sconedocs.github.io/
- 10): Need: SJ HomoLib (A Java Library w/ Implementation of Partial Homomorphic Encryption Alg.)
- 11): Need a Blockmess Platform and enabled Externalized APIs, similar as you have for BFT Smart in your PA#1 Implementation

Questions ?