Teaclave: A Universal Secure Computing Platform

https://teaclave.apache.org/

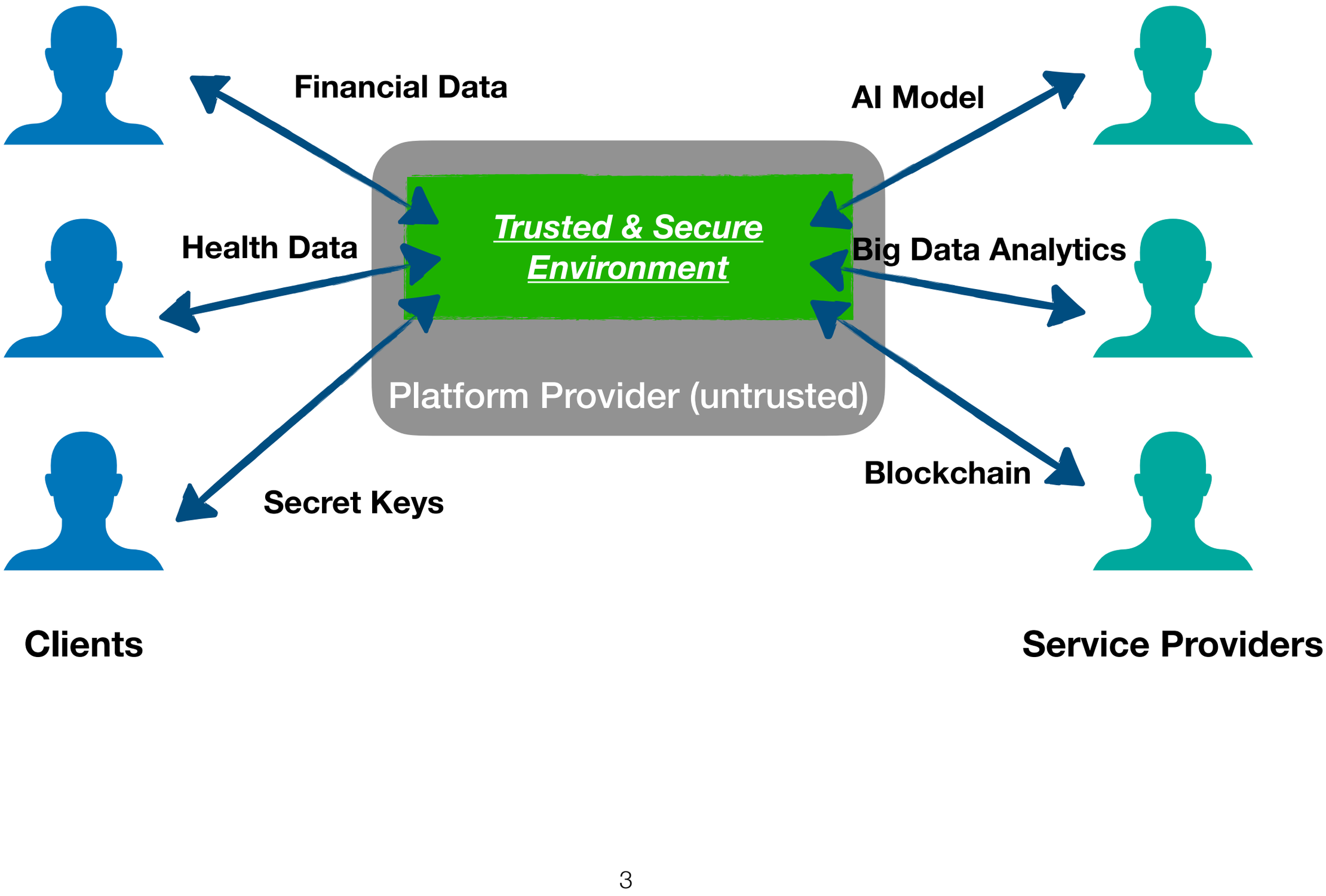
Mingshen Sun

Baidu, Apache Teaclave (incubating) PPMC

Data Privacy

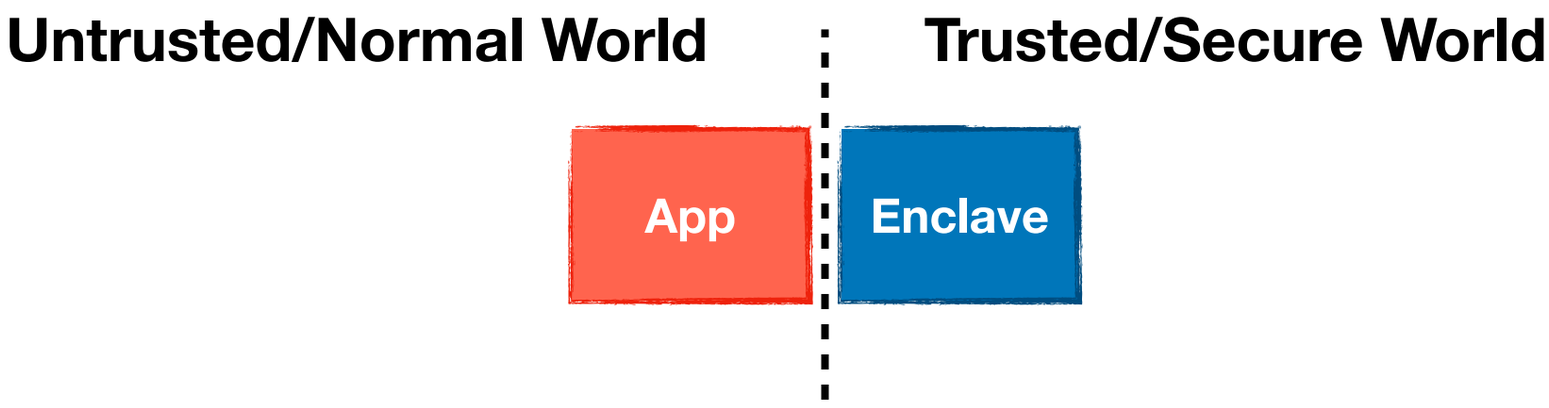
- big data analytics, machine learning, cloud/edge computing, and blockchain
- public cloud infrastructure
- intellectual property such as models and algorithms

Secure Computing



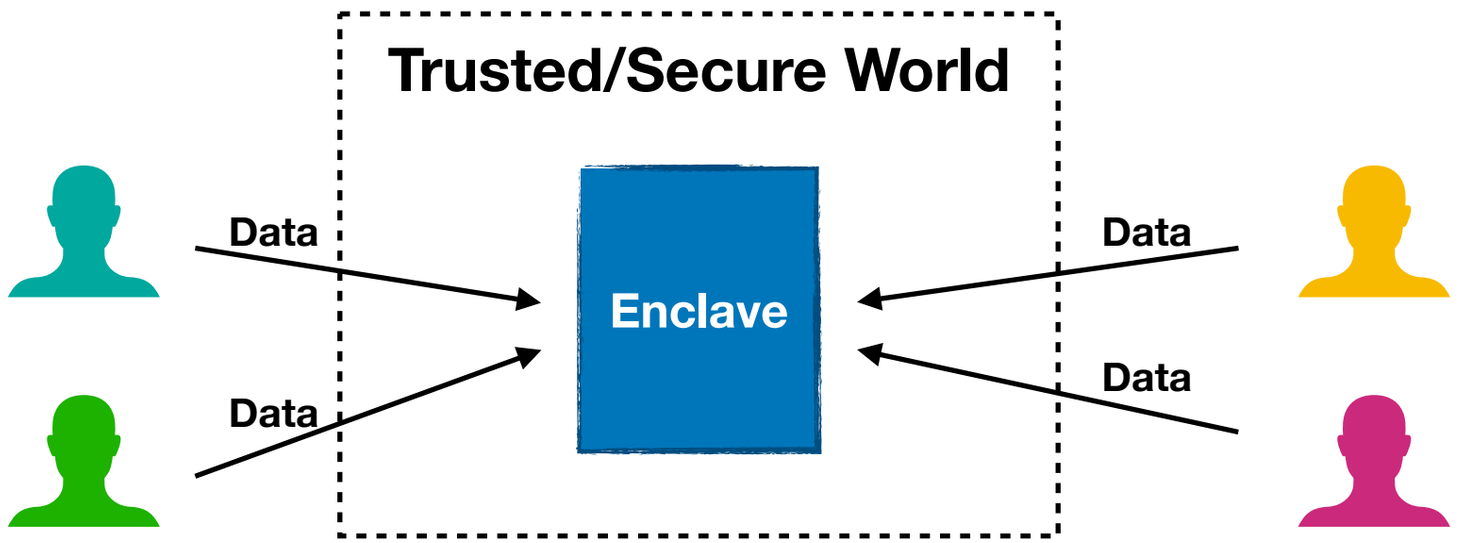
Secure Computing

- Private Computation: private set intersection
- Private Machine Learning: multi-party model training
- Homomorphic encryption/Oblivious transfer MPC
- Hardware-based isolation, memory encryption and attestation: Intel SGX, ARM TrustZone
- Platform providers: Microsoft Azure, Google Cloud, IBM Cloud



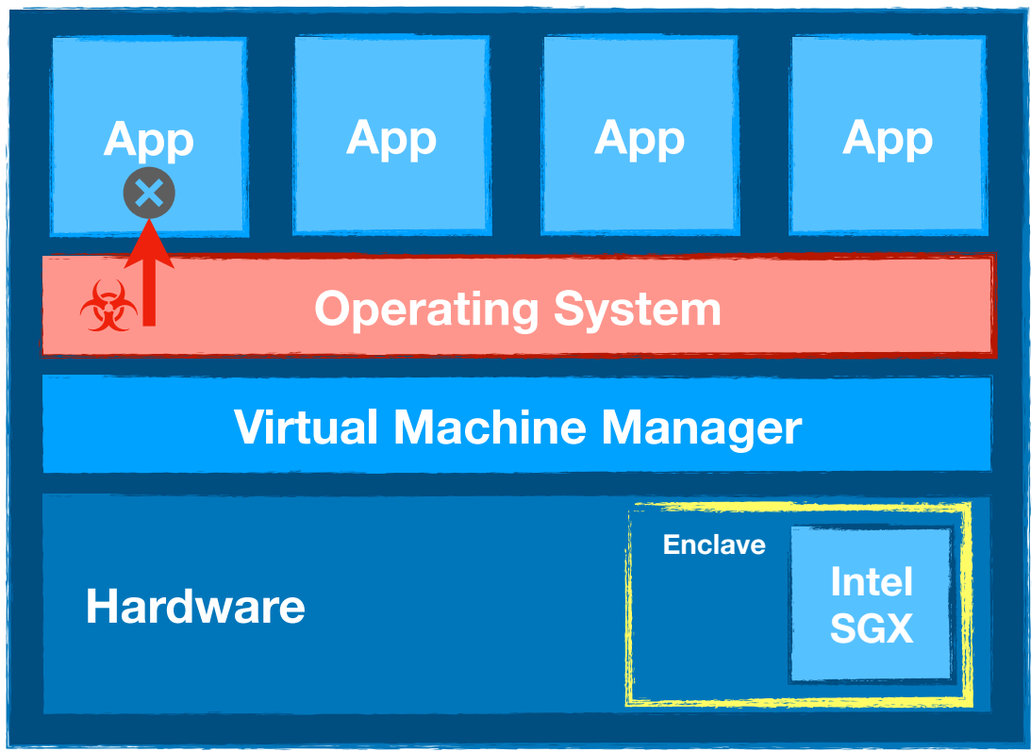
Secure Computing

- Private Computation: private set intersection
- Private Machine Learning: multi-party model training
- Homomorphic encryption/Oblivious transfer MPC
- Hardware-based isolation, memory encryption and attestation: Intel SGX, ARM TrustZone
- Platform providers: Microsoft Azure, Google Cloud, IBM Cloud

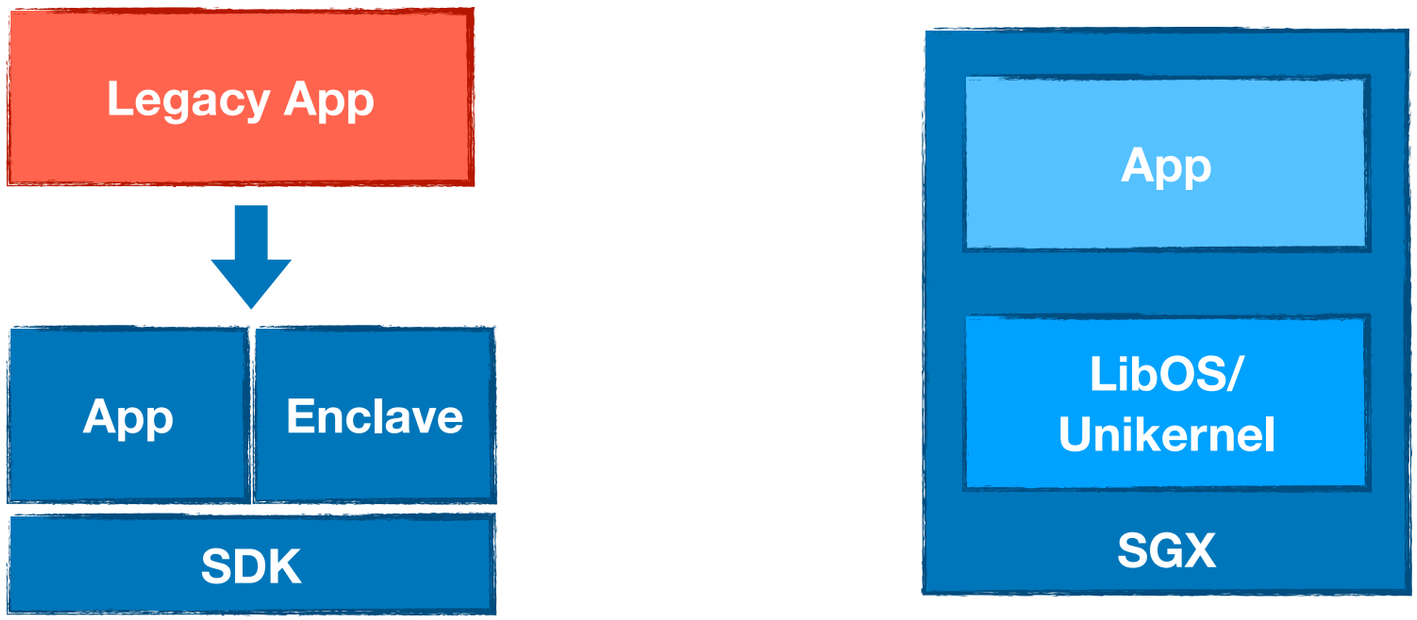


Intel SGX

- Intel® Software Guard Extensions
- Hardware-based isolation and memory encryption provides more code protection to help you develop and deliver more secure solutions.

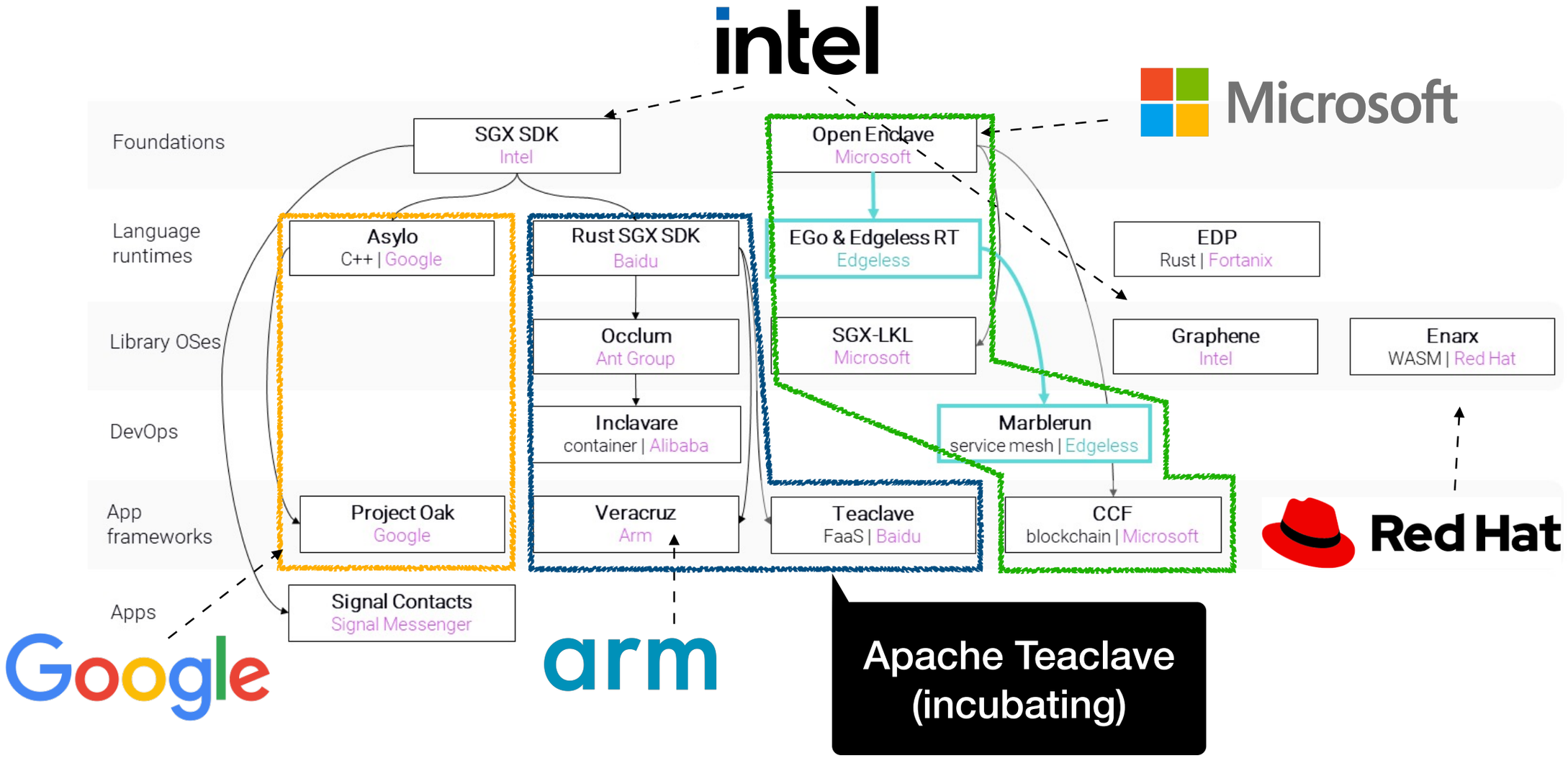


SGX Ecosystem: Now and Next



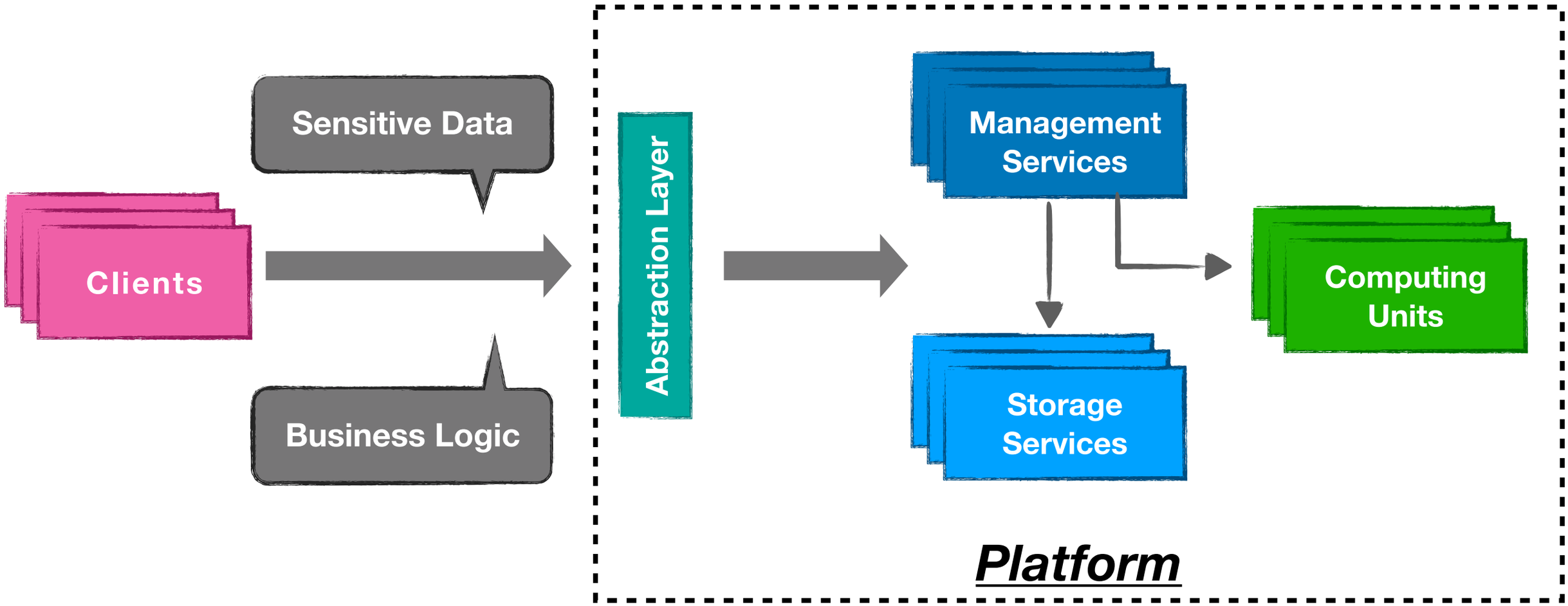
- Today we can build SGX application with SDK
- or we can deploy legacy application in containerized SGX environment based on LibOS and Unikernel concepts
- Still, lots of effort for developers

SGX Ecosystem Landscape



https://blog.edgeless.systems/the-open-source-landscape-of-confidential-computing-in-2021-7f847ebfc0a9

SGX Ecosystem: Now and Next



- We need a **framework or platform** that allow the programmer to **concentrate on the business logic** and automates more protection of their code and data without worrying about technical details of different TEE implementations.

Teaclave

- **Apache Teaclave (incubating) is an open source universal secure computing platform, <u>making computation on privacy-sensitive data safe and simple</u>.**
  - Originally developed at Baidu called MesaTEE/Rust SGX SDK, open-source in July 2019
  - Entered Apache Incubator on August 2019, using Teaclave as the project name
  - Open source in <u>The Apache Way</u>
  - Homepage: <u>https://teaclave.apache.org/</u>

- Repositories (sub-projects)
  - **Teaclave**: <u>https://github.com/apache/incubator-teaclave</u>
  - **Teaclave SGX SDK**: <u>https://github.com/apache/incubator-teaclave-sgx-sdk</u>
  - **Teaclave TrustZone SDK**: <u>https://github.com/apache/incubator-teaclave-trustzone-sdk</u>

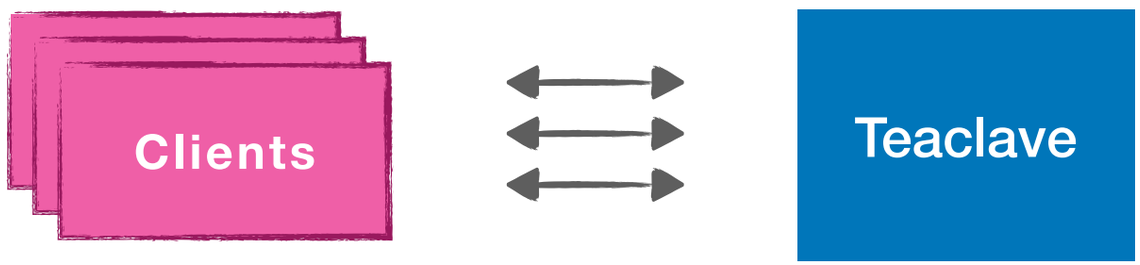
Highlights

- **Function-as-a-Service**
  - **function-as-a-service** interfaces
  - built-in functions and Python executors
- **Secure and Attestable**
  - Intel SGX: **hardware-based** isolation, memory encryption and attestation
  - Rust: fast, **memory-safe**, system programming language
- **Ease of Use**
  - deployment on the cloud infrastructure
  - API, SDK, CLI, SGX tool, etc
- **Flexible**
  - attestation, RPC, functions, binder

Workflow

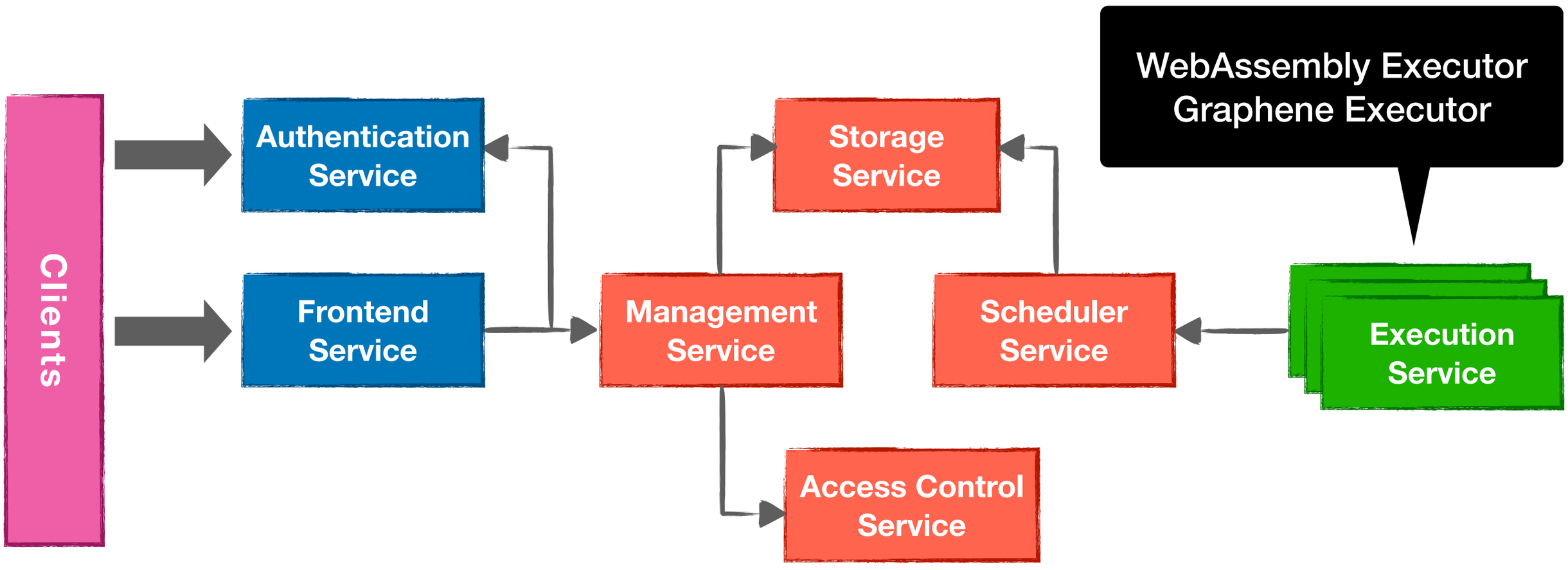
- **FaaS interface**
  - function: business logic
  - data: sensitive data
  - participants: parties involved in a task
- **Workflow of a task in Teaclave**

1. register sensitive data into the platform
2. register a function you want to execute with the data
3. create a task
4. run the task and get results



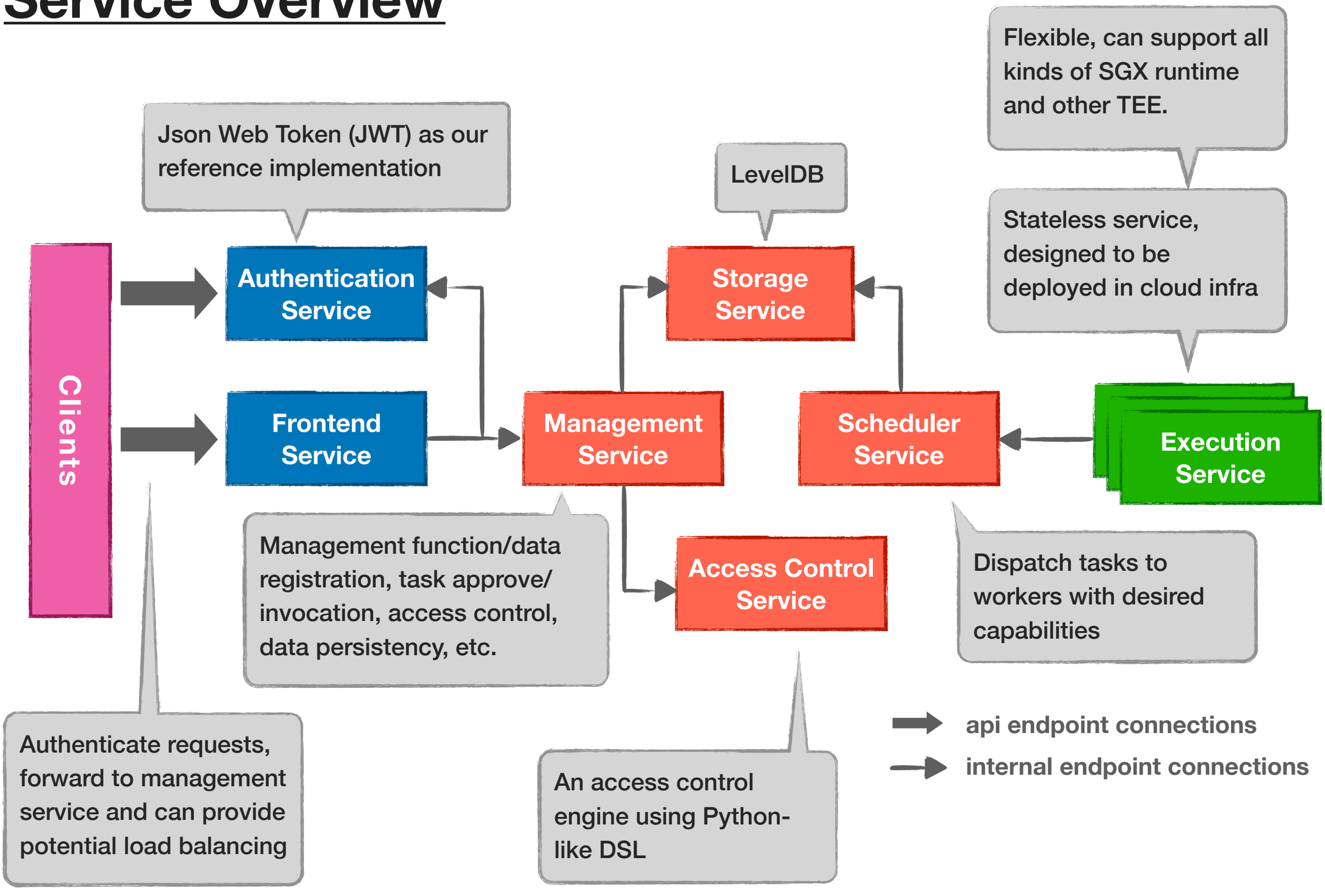
Teaclave Design

Architecture

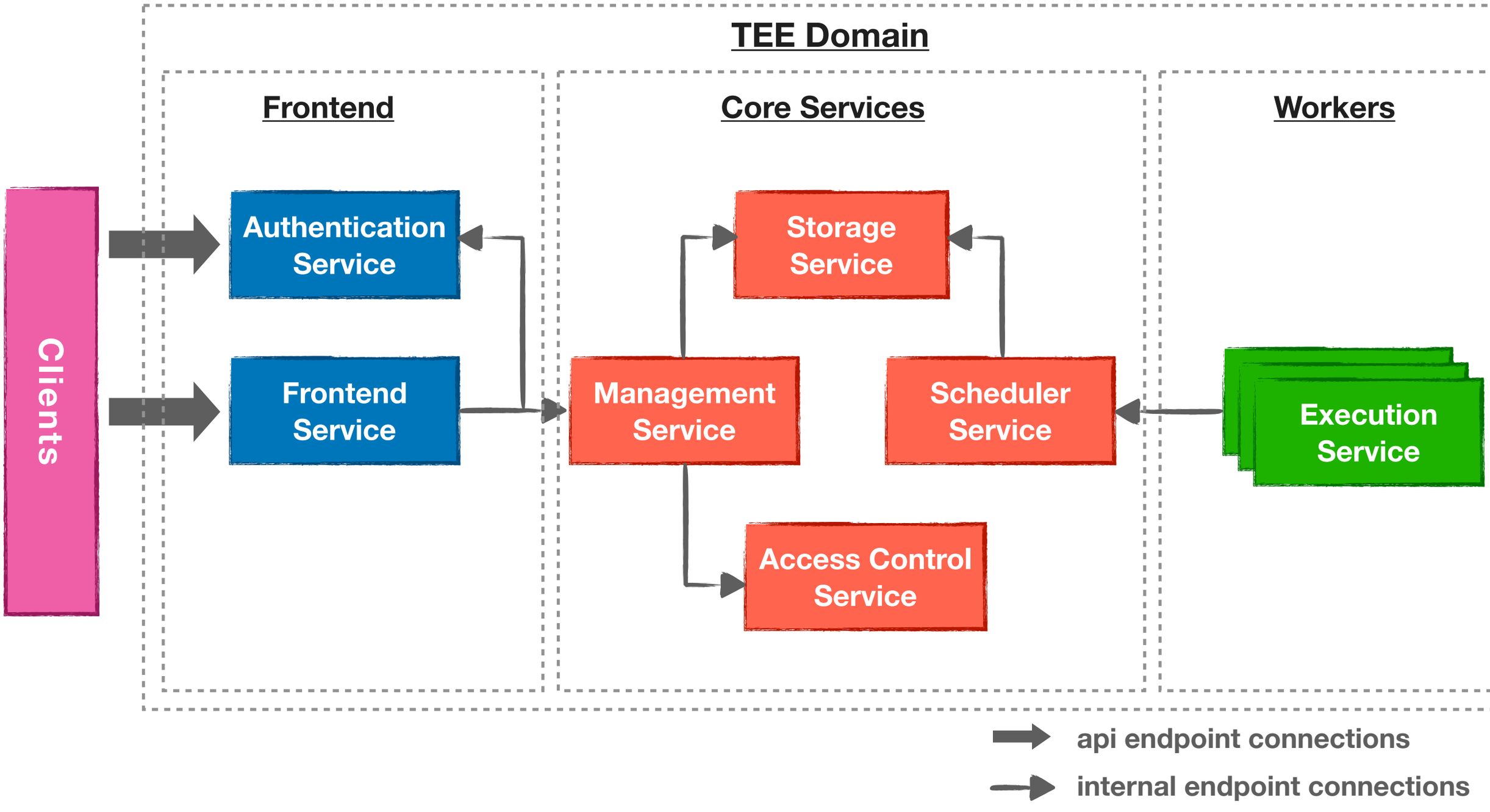


Teaclave Architecture

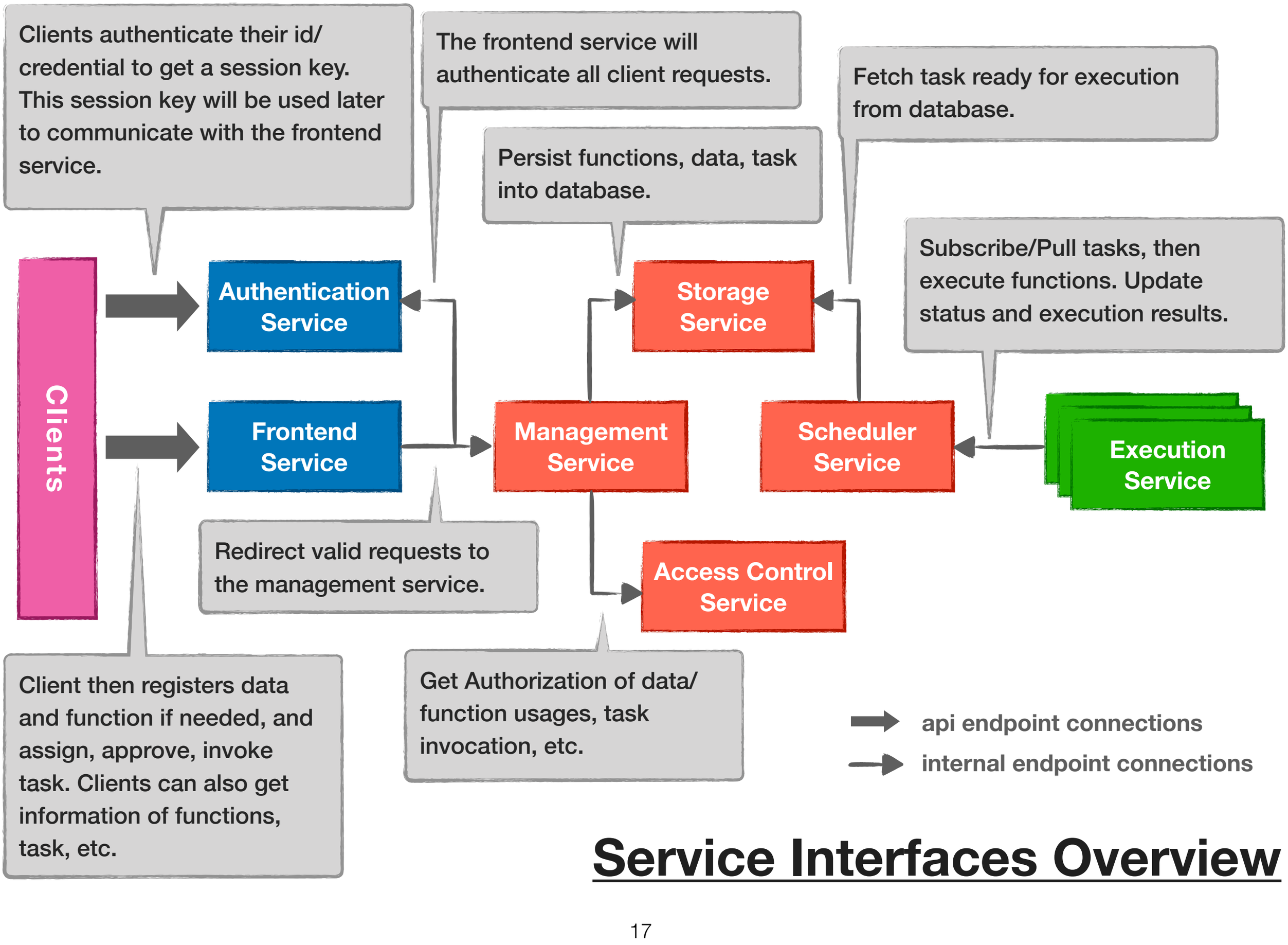
Service Overview



Domains


TEE Domain
Frontend
Core Services
Workers
Clients
Authentication Service
Frontend Service
Storage Service
Management Service
Scheduler Service
Execution Service
Access Control Service
api endpoint connections
internal endpoint connections

Service Interfaces Overview



RPC Interfaces

```
service TeaclaveStorage {
  rpc Get
  rpc Put
  rpc Delete
  rpc Enqueue
  rpc Dequeue
}
```

```
service TeaclaveAuthenticationApi {
  rpc UserRegister
  rpc UserLogin
}

service TeaclaveAuthenticationInternal {
  rpc UserAuthenticate
}
```

```
service TeaclaveScheduler {
  rpc PublishTask
  rpc Subscribe
  rpc PullTask
  rpc UpdateTaskStatus
  rpc UpdateTaskResult
}
```

```
service TeaclaveManagement {
  rpc RegisterInputFile
  rpc RegisterOutputFile
  rpc RegisterFusionOutput
  rpc RegisterInputFromOutput
  rpc GetOutputFile
  rpc GetInputFile
  rpc RegisterFunction
  rpc GetFunction
  rpc CreateTask
  rpc GetTask
  rpc AssignData
  rpc ApproveTask
  rpc InvokeTask
}
```

```
service TeaclaveExecution {
}
```

```
service TeaclaveFrontend {
  rpc RegisterInputFile
  rpc RegisterOutputFile
  rpc RegisterFusionOutput
  rpc RegisterInputFromOutput
  rpc GetOutputFile
  rpc GetInputFile
  rpc RegisterFunction
  rpc GetFunction
  rpc CreateTask
  rpc GetTask
  rpc AssignData
  rpc ApproveTask
  rpc InvokeTask
}
```

```
service TeaclaveAccessControl {
  rpc AuthorizeData
  rpc AuthorizeFunction
  rpc AuthorizeTask
  rpc AuthorizeStagedTask
}
```

RPC
`/rpc`

service impl
`/services`

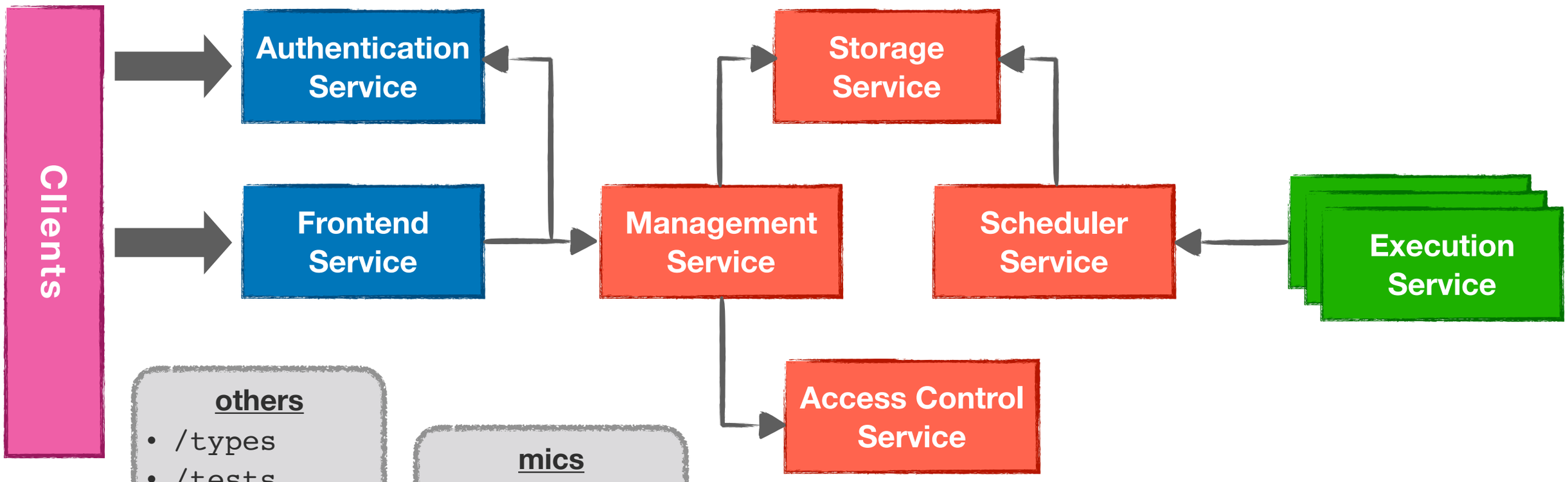
function impl
`/worker`

app/enclave communication
`/binder`

service protocol def
`/services/proto`

attestation
`/attestation, /rpc, /dcap`

build/runtime configuration
`/config`



others
- `/types`
- `/tests`
- `/examples`
- `/sdk`
- `/file_agent`
- `/common`
- `/utils`
- `/docker`
- `/cli`
- `/crypto`

mics
- `/third_party`
- `/licenses`
- `/docs`
- `/keys`

api endpoint connections

internal endpoint connections

Service Protocol Definition

Service Protocol Definition

services/proto/src/proto/teaclave_authentication_service.proto

```
13 message UserLoginRequest {
14   string id = 1;
15   string password = 2;
16 }
17
18 message UserLoginResponse {
19   string token = 1;
20 }
21
22 message UserAuthenticateRequest {
23   teaclave_common_proto.UserCredential credential = 1;
24 }
25
26 message UserAuthenticateResponse {
27   bool accept = 1;
28 }
29
30 service TeaclaveAuthenticationApi {
31   rpc UserRegister(UserRegisterRequest) returns (UserRegisterResponse);
32   rpc UserLogin (UserLoginRequest) returns (UserLoginResponse);
33 }
34
35 service TeaclaveAuthenticationInternal {
36   rpc UserAuthenticate (UserAuthenticateRequest) returns (UserAuthenticateResponse);
37 }
```

Service Protocol Definition

```rust
#[derive(Clone, PartialEq, ::prost::Message)]
#[derive(serde::Serialize, serde::Deserialize)]
pub struct AuthorizeStagedTaskRequest {
    #[prost(string, tag="1")]
    pub subject_task_id: std::string::String,
    #[prost(string, tag="2")]
    pub object_function_id: std::string::String,
    #[prost(string, repeated, tag="3")]
    pub object_input_data_id_list: ::std::vec::Vec<std::string::String>,
    #[prost(string, repeated, tag="4")]
    pub object_output_data_id_list: ::std::vec::Vec<std::string::String>,
}
#[derive(Clone, PartialEq, ::prost::Message)]
#[derive(serde::Serialize, serde::Deserialize)]
pub struct AuthorizeStagedTaskResponse {
    #[prost(bool, tag="1")]
    pub accept: bool,
}
#[derive(Clone, serde::Serialize, serde::Deserialize, Debug)]
#[serde(tag = "request", rename_all = "snake_case")]
pub enum TeaclaveAccessControlRequest {
    AuthorizeData(AuthorizeDataRequest),
    AuthorizeFunction(AuthorizeFunctionRequest),
    AuthorizeTask(AuthorizeTaskRequest),
    AuthorizeStagedTask(AuthorizeStagedTaskRequest),
}

#[allow(clippy::large_enum_variant)]
#[derive(Clone, serde::Serialize, serde::Deserialize, Debug)]
#[serde(tag = "response", rename_all = "snake_case")]
pub enum TeaclaveAccessControlResponse {
    AuthorizeData(AuthorizeDataResponse),
    AuthorizeFunction(AuthorizeFunctionResponse),
    AuthorizeTask(AuthorizeTaskResponse),
    AuthorizeStagedTask(AuthorizeStagedTaskResponse),
}

pub trait TeaclaveAccessControl {
    fn authorize_data(
        &self,
        request: teaclave_rpc::Request<crate::teaclave_access_control_service::AuthorizeDataRequest>
    ) -> teaclave_types::TeaclaveServiceResponseResult<crate::teaclave_access_control_service::AuthorizeDataResponse>;
```

Generated request/response message in Rust

Generated trait definition of RPC interfaces

Service Protocol Definition


```rust
#[into_request(TeaclaveAuthenticationApiRequest::UserRegister)]
#[derive(Debug)]
pub struct UserRegisterRequest {
    pub id: std::string::String,
    pub password: std::string::String,
}

impl UserRegisterRequest {
    pub fn new(id: impl Into<String>, password: impl Into<String>) -> Self {
        Self {
            id: id.into(),
            password: password.into(),
        }
    }
}

#[into_request(TeaclaveAuthenticationApiResponse::UserRegister)]
#[derive(Debug, Default)]
pub struct UserRegisterResponse;

#[into_request(TeaclaveAuthenticationApiRequest::UserLogin)]
#[derive(Debug)]
pub struct UserLoginRequest {
    pub id: std::string::String,
    pub password: std::string::String,
}
```

services/proto/src/teaclave_authentication_service.rs

Service Implementation

- **App part**
  - Read runtime configuration file
  - Invoke service enclave and pass configuration to the enclave
- **Enclave entrypoint**
  - Get attestation report
  - Launch service to serve connections
- **Enclave service part**
  - Register struct as a service implementation
  - Implement traits defined in the protocol

Service Implementation

- App part
  - Read runtime configuration file
  - Invoke service enclave and pass configuration to the enclave

```rust
fn start_enclave_service(tee: Arc<TeeBinder>, config: RuntimeConfig) {
    let input = StartServiceInput::new(config);
    let command = ECallCommand::StartService;
    match tee.invoke::<StartServiceInput, TeeServiceResult<StartServiceOutput>>(command, input) {...}

    unsafe { libc::raise(signal_hook::SIGTERM) };
}

fn main() -> Result<()> {
    env_logger::init();

    let tee = Arc::new(TeeBinder::new(PACKAGE_NAME).context("Failed to new the enclave.")?);
    let config = teaclave_config::RuntimeConfig::from_toml("runtime.config.toml")
        .context("Failed to load config file.")?;

    let tee_ref = tee.clone();
    std::thread::spawn(move || {
        start_enclave_service(tee_ref, config);
    });
```

Service Implementation

- **Enclave entrypoint**
  - Get attestation report, enclave info, and create server config
  - Launch service to serve connections

Service Implementation

```rust
fn start_service(config: &RuntimeConfig) -> anyhow::Result<()> {
    let listen_address = config.internal_endpoints.access_control.listen_address;
    let as_config = &config.attestation;
    let attestation_config = AttestationConfig::new(...);
    let attested_tls_config = RemoteAttestation::new()
        .config(attestation_config)
        .generate_and_endorse()
        .unwrap()
        .attested_tls_config()
        .unwrap();
    let enclave_info = EnclaveInfo::verify_and_new(...)?;
    let accepted_enclave_attrs: Vec<teaclave_types::EnclaveAttr> = INBOUND_SERVICES
        .iter()
        .map(|service| {...})
        .collect();
    let server_config = SgxTrustedTlsServerConfig::from_attested_tls_config(attested_tls_config)
        .unwrap()
        .attestation_report_verifier(...)
        .unwrap();

    acs::init_acs().unwrap();
    let mut server = SgxTrustedTlsServer::<...>::new(listen_address, server_config);
    let service = service::TeaclaveAccessControlService::new();
    match server.start(service) {...}
    Ok(())
}
```

Get listen address

Construct attestation config
Do remote attestation
Get attested TLS config

Load and verify enclave info

Get accepted enclave attributes used for verifier of TLS attestation

Construct trusted TLS server config with attested TLS config

Init access control service

Create a trusted TLS server

Construct a service

Start the service with trusted TLS Server

Service Implementation

- Register struct as a service implementation

Proc macro will generate impl of TeaclaveService for the struct, need to provide create name, trait name, and error struct

```rust
#[teaclave_service(
    teaclave_authentication_service,
    TeaclaveAuthenticationApi,
    TeaclaveAuthenticationError
)]
#[derive(Clone)]
pub(crate) struct TeaclaveAuthenticationApiService {
    db_client: DbClient,
    jwt_secret: Vec<u8>,
}
```

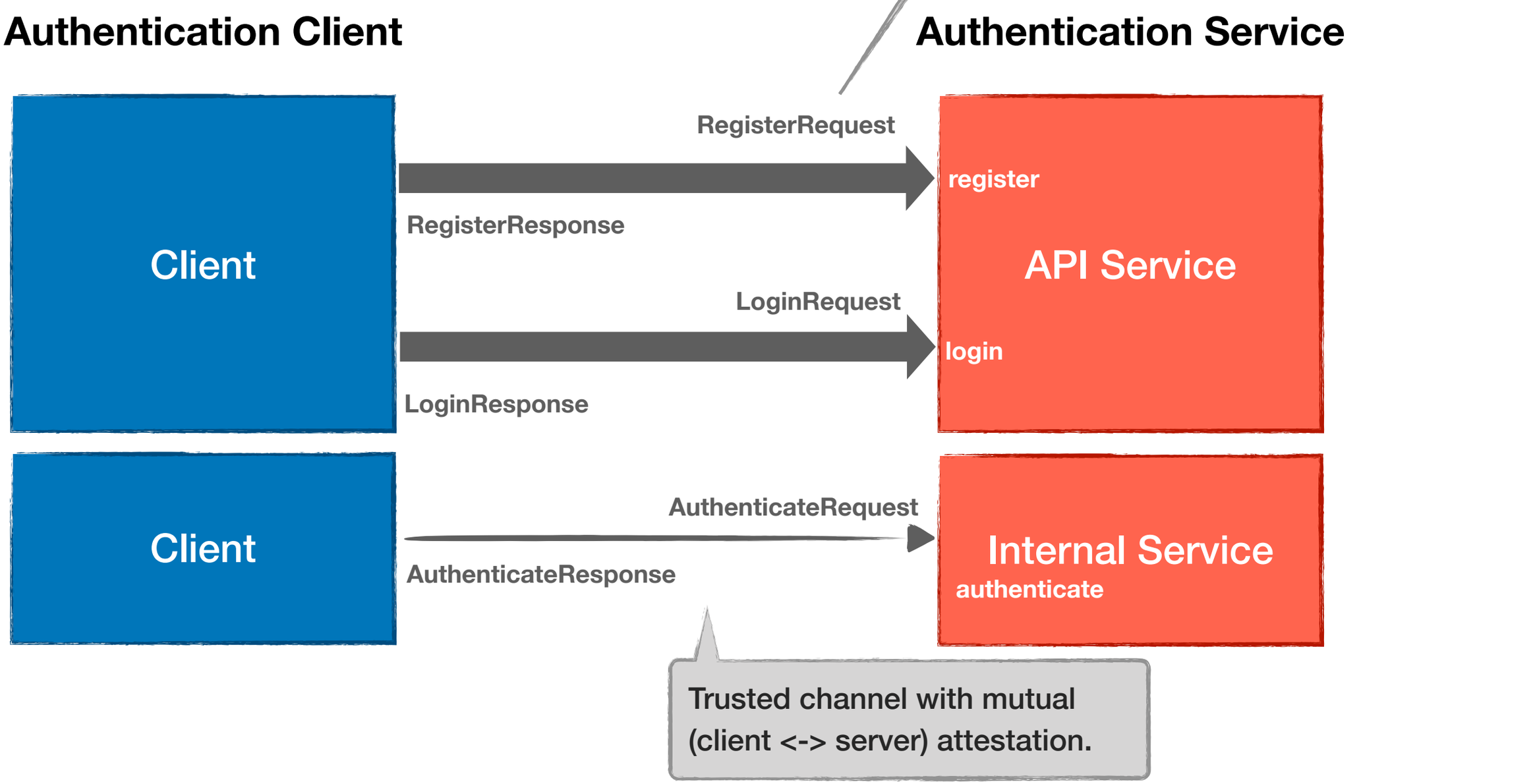
Service Implementation

- Implement traits defined in the protocol

```
53  impl TeaclaveAuthenticationApi for TeaclaveAuthenticationApiService {
54      fn user_register(
55          &self,
56          request: Request<UserRegisterRequest>,
57      ) -> TeaclaveServiceResponseResult<UserRegisterResponse> {...}
72
73      fn user_login(
74          &self,
75          request: Request<UserLoginRequest>,
76      ) -> TeaclaveServiceResponseResult<UserLoginResponse> {...}
101 }
```

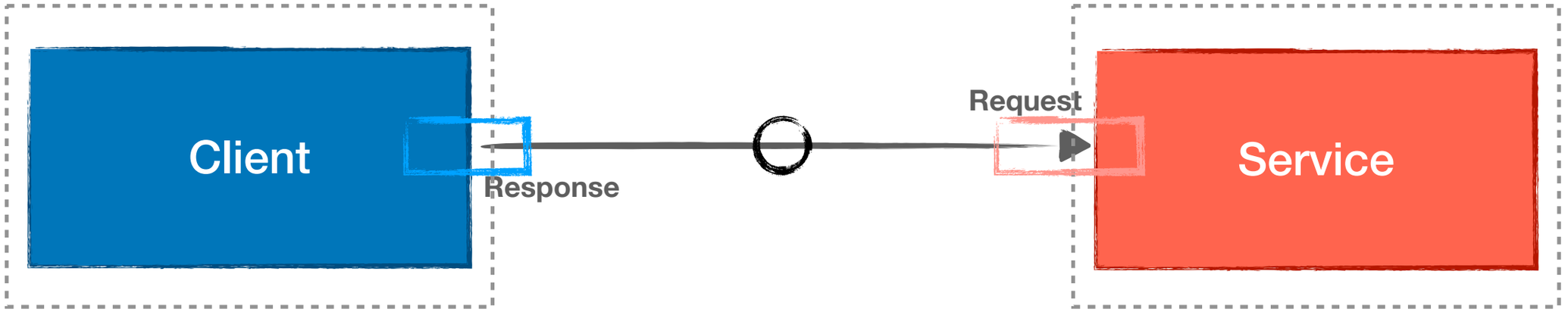
RPC

- Communication between client/service
- Support trusted channel (attested TLS channel)



RPC

- Client, Endpoint
- SgxTrustedTlsChannel, SgxTrustedTlsServer
- JsonProtocol



Request (JsonProtocol)

length (big endian)

```
{
  "request": "user_login",
  "id": "user_id",
  "password": "user_password",
}
```

Response (JsonProtocol)

length (big endian)

```
{
  "result": "ok",
  "response": "user_login",
  "token": "eyJ0eXAiO...",
}
```

RPC

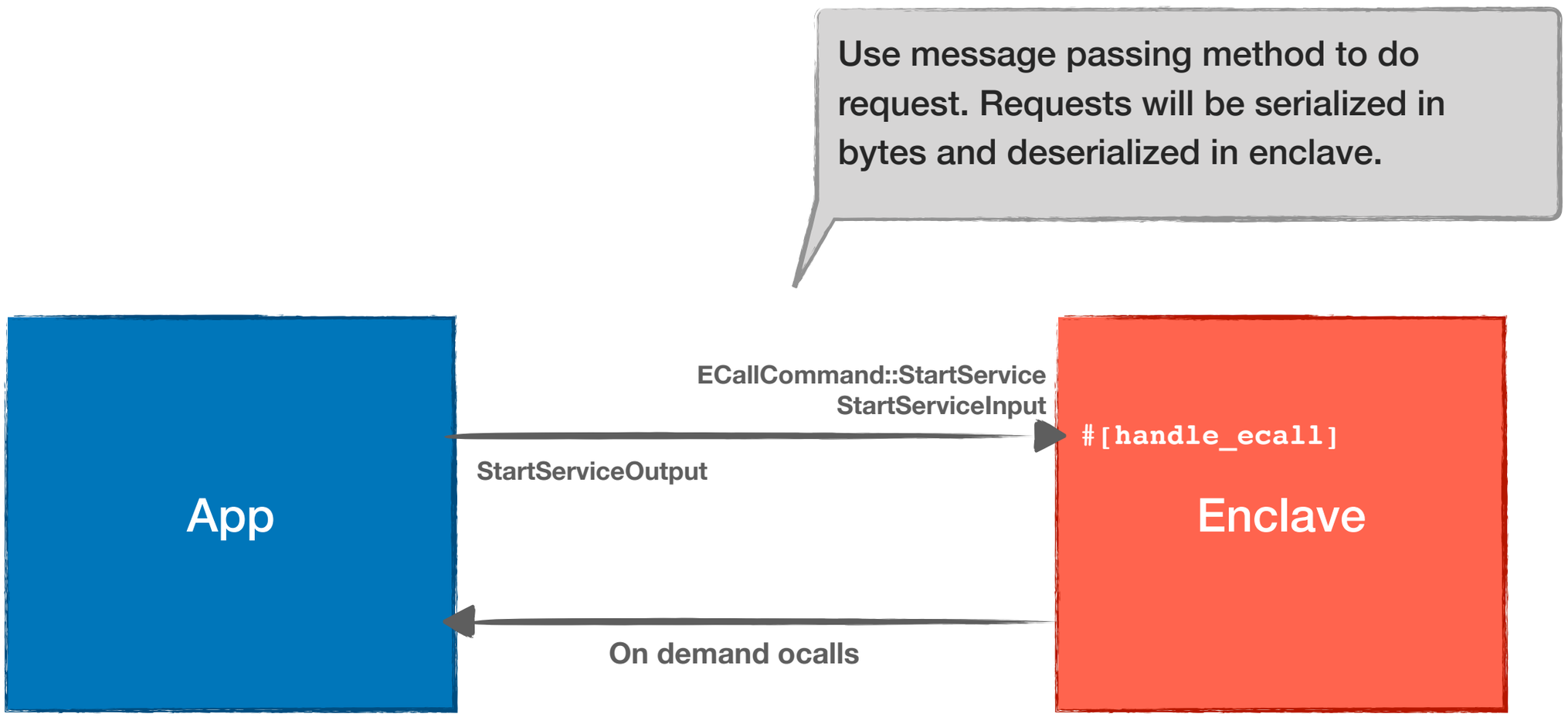
- Echo example

```rust
fn start_echo_service() {
    use super::*;
    use std::thread;
    use std::time::Duration;
    thread::spawn(move || {
        let cert = pemfile::certs(&mut io::BufReader::new(
            fs::File::open(END_FULLCHAIN).unwrap(),
        ))
        .unwrap();
        let private_key =
            &pemfile::pkcs8_private_keys(&mut io::BufReader::new(fs::File::open(END_KEY).unwrap()))
                .unwrap()[0];
        let addr = "127.0.0.1:12345".parse().unwrap();
        let config = SgxTrustedTlsServerConfig::new()
            .server_cert(&cert[0].as_ref(), &private_key.0)
            .unwrap();
        let mut server = SgxTrustedTlsServer::<EchoResponse, EchoRequest>::new(addr, config);
        server.start(EchoService).unwrap();
    });
    thread::sleep(Duration::from_secs(3));
}
```

```rust
fn echo_success() {
    use super::*;

    let channel = Endpoint::new("localhost:12345").connect().unwrap();
    let mut client = EchoClient::new(channel).unwrap();
    let request = SayRequest {
        message: "Hello, World!".to_string(),
    };
    let response_result = client.say(request);
    info!("{:?}", response_result);

    assert!(response_result.is_ok());
    assert!(response_result.unwrap().message == "Hello, World!");
}
```

App/Enclave Communication

- **ecall**: message passing communication, use binder to send request similar with RPC
- **ocall**: define ocall on demand



App/Enclave Communication

- App

Construct an input of start service ecall with the runtime config

```
48 fn start_enclave_service(tee: Arc<TeeBinder>, config: RuntimeConfig) {
49     let input = StartServiceInput::new(config);
50     let command = ECallCommand::StartService;
51     match tee.invoke::<StartServiceInput, TeeServiceResult<StartServiceOutput>>(command, input) {...}
62
63     unsafe { libc::raise(signal_hook::SIGTERM) };
64 }
```

Define command as start service

Use TeeBinder ref to invoke command with input

App/Enclave Communication

- Enclave

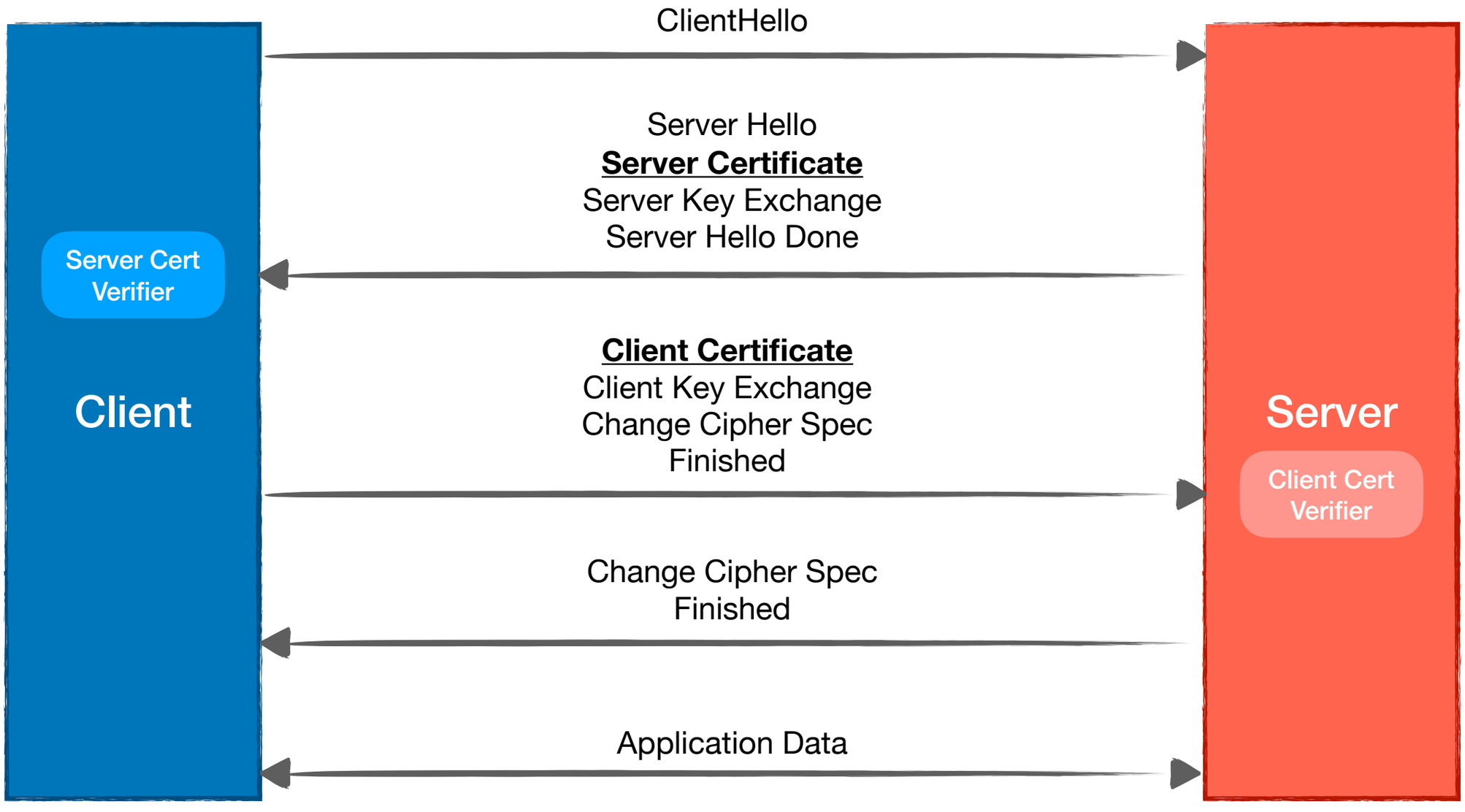
Define function (i.e., entrypoint) to handle a ecall

```
185 #[handle_ecall]
186 fn handle_start_service(input: &StartServiceInput) → TeeServiceResult<StartServiceOutput> {...}
190
191 #[handle_ecall]
192 fn handle_init_enclave(_: &InitEnclaveInput) → TeeServiceResult<InitEnclaveOutput> {...}
196
197 #[handle_ecall]
198 fn handle_finalize_enclave(_: &FinalizeEnclaveInput) → TeeServiceResult<FinalizeEnclaveOutput> {...}
202
203 register_ecall_handler!(
204     type ECallCommand,
205     (ECallCommand::StartService, StartServiceInput, StartServiceOutput),
206     (ECallCommand::InitEnclave, InitEnclaveInput, InitEnclaveOutput),
207     (ECallCommand::FinalizeEnclave, FinalizeEnclaveInput, FinalizeEnclaveOutput),
208 );
```

Register handlers for various ecall command

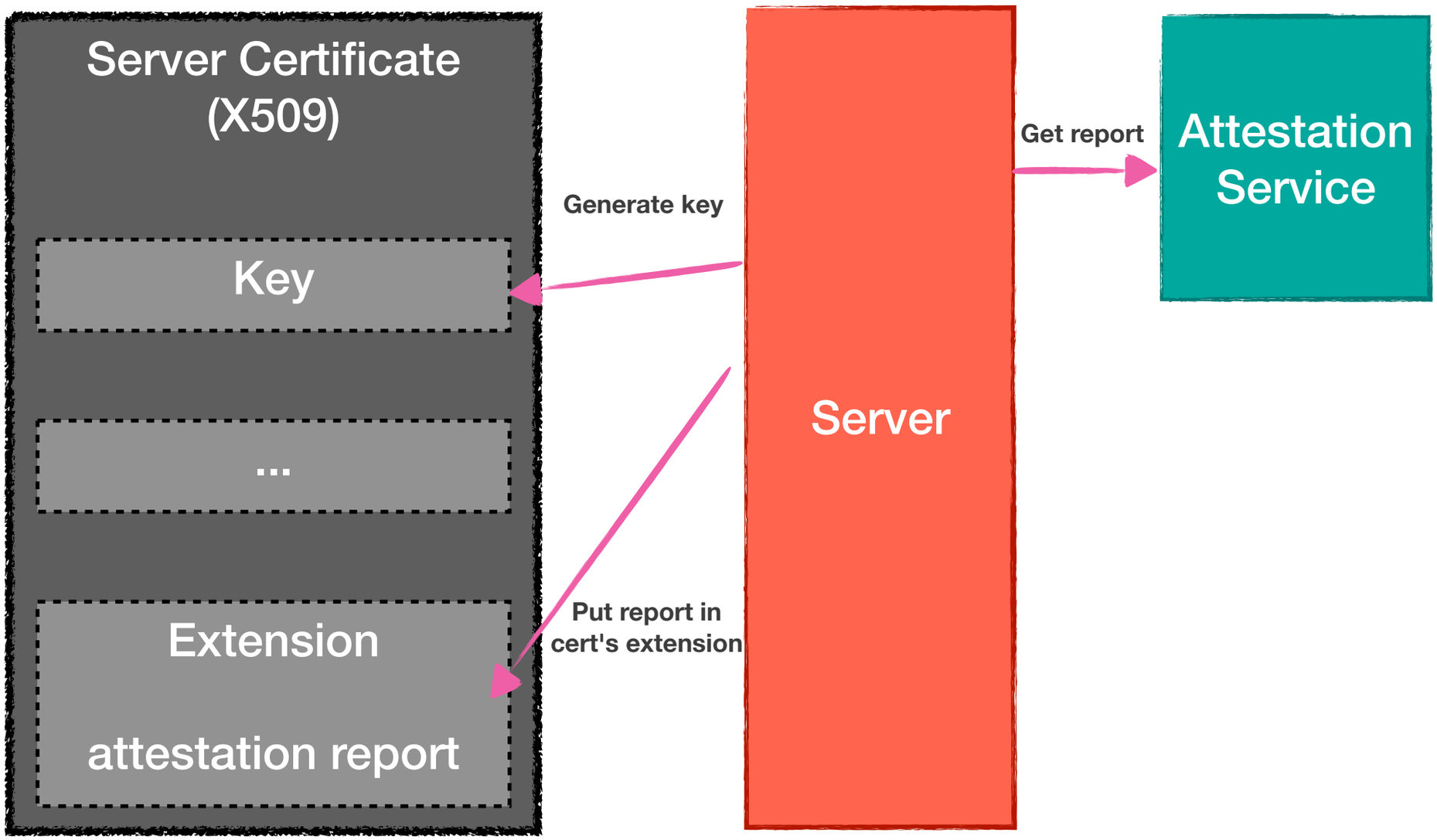
Attestation

- Attestation in TLS handshake



Attestation

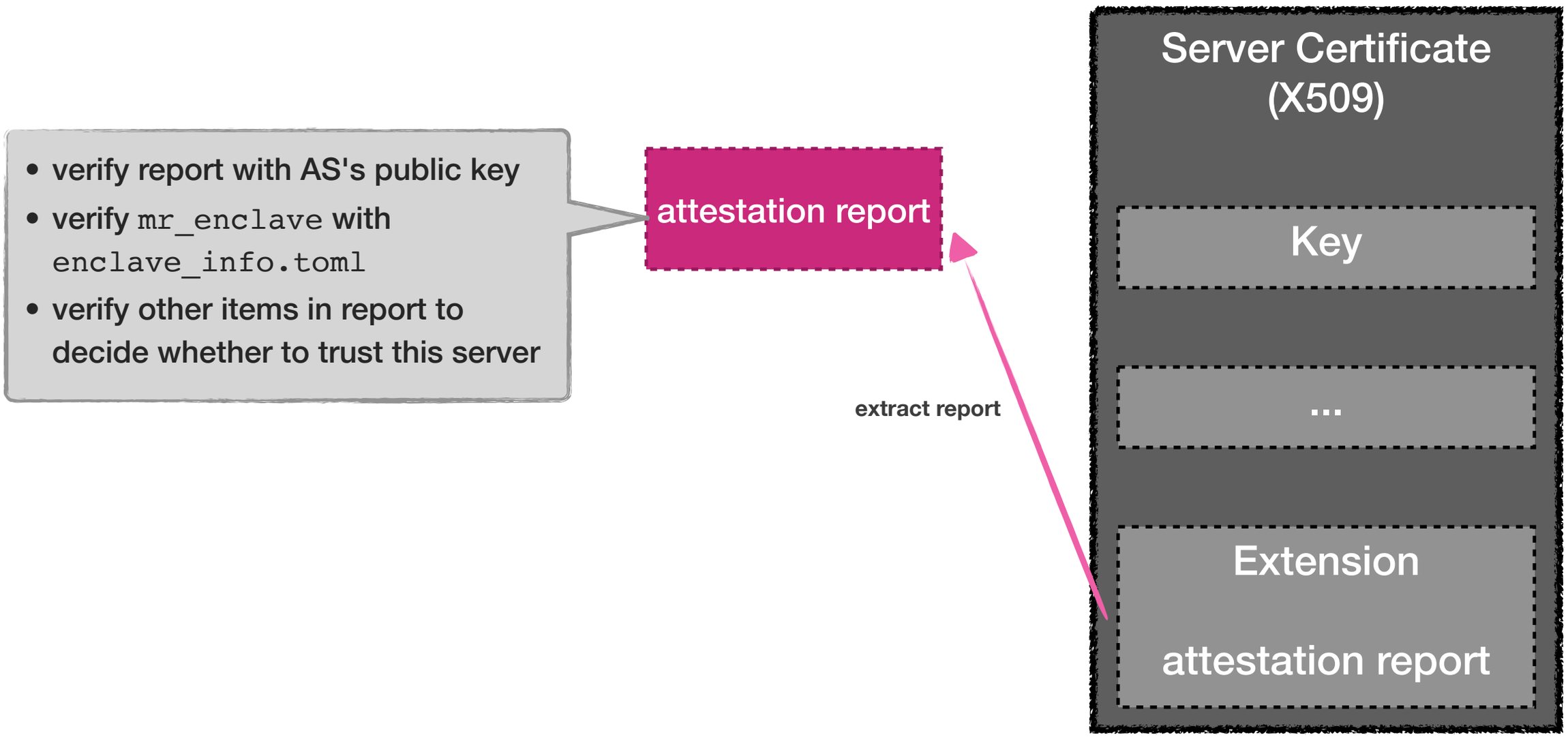
- Server certificate (same for client certificate)



- Attestation report will be refreshed after a period of time.
- The server certificate will be also updated automatically.

Attestation

- Server certificate verifier



Attestation

- Server certifica

```toml
[teaclave_access_control_service]
mr_enclave = "41860a711767332f80cb49a0c31ab0d597e79296597383ed7b47aa93f4eb942a"
mr_signer  = "83d719e77deaca1470f6baf62a4d774303c899db69020f9c70ee1dfc08c7ce9e"
[teaclave_authentication_service]
mr_enclave = "d5350aad6065e6bcfee0a5753af4e180ca95cd726053f3d9cff914a83ebbd40d"
mr_signer  = "83d719e77deaca1470f6baf62a4d774303c899db69020f9c70ee1dfc08c7ce9e"
[teaclave_execution_service]
mr_enclave = "6ff6f7a2bd452b1602042b13876ecfdb6f7f8b5285fb5d080c656e14d49b08fc"
mr_signer  = "83d719e77deaca1470f6baf62a4d774303c899db69020f9c70ee1dfc08c7ce9e"
```

- verify report with AS's public key
- verify `mr_enclave` with `enclave_info.toml`
- verify other items in report to decide whether to trust this server

attestation report

```
→  auditors git:(develop) tree
.
├── albus_dumbledore
│   ├── albus_dumbledore.public.pem
│   └── albus_dumbledore.sign.sha256
├── godzilla
│   ├── godzilla.public.pem
│   └── godzilla.sign.sha256
└── optimus_prime
    ├── optimus_prime.public.pem
    └── optimus_prime.sign.sha256
```

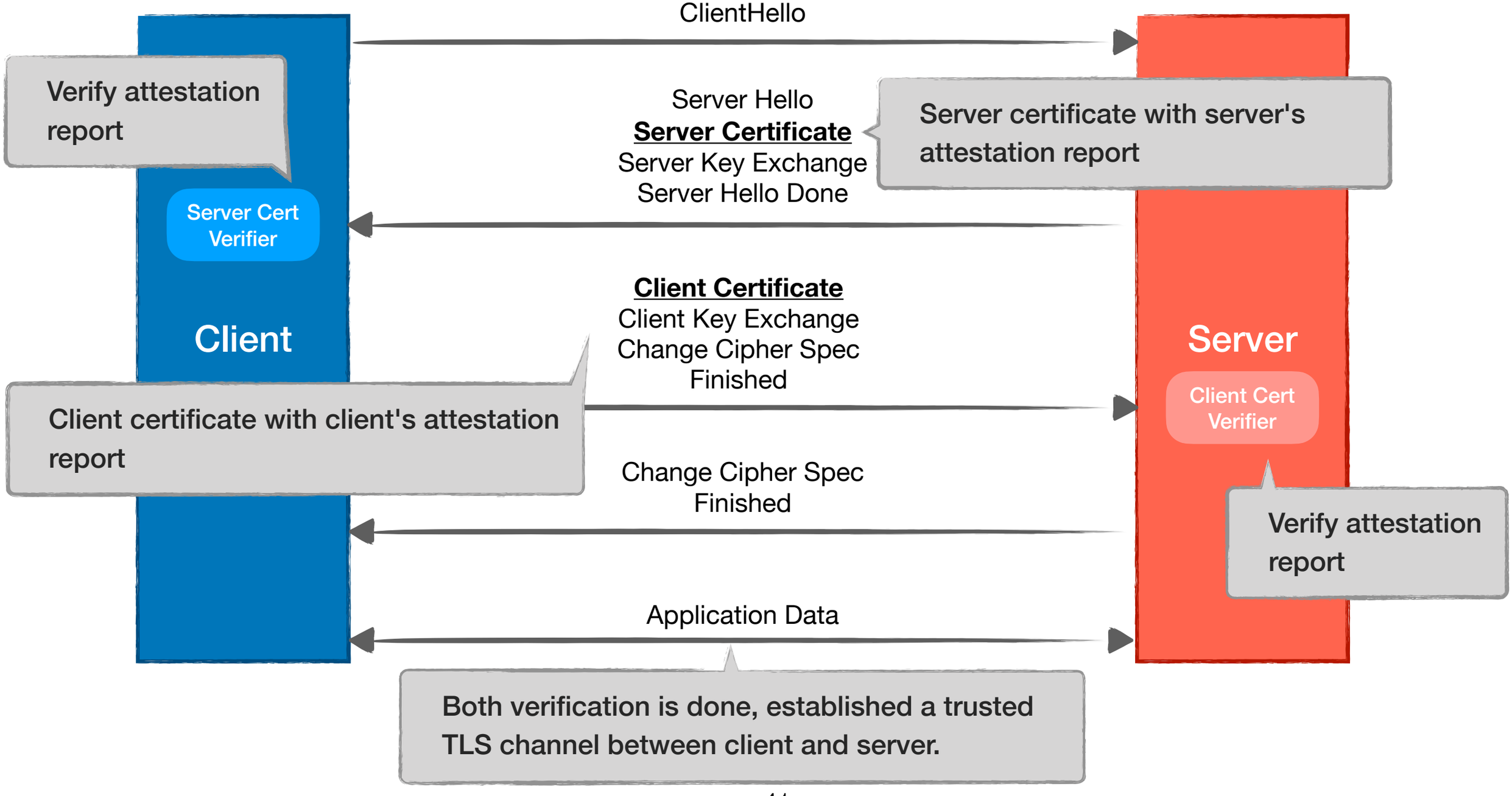
extract repo

attestation report

- `enclave_info.toml` is generated at build time containing information like `mr_signer` and `mr_enclave` of all enclaves.
- `enclave_info.toml` should be signed by all auditors and will be verify at the startup of a service.

Attestation

- Attestation in TLS handshake



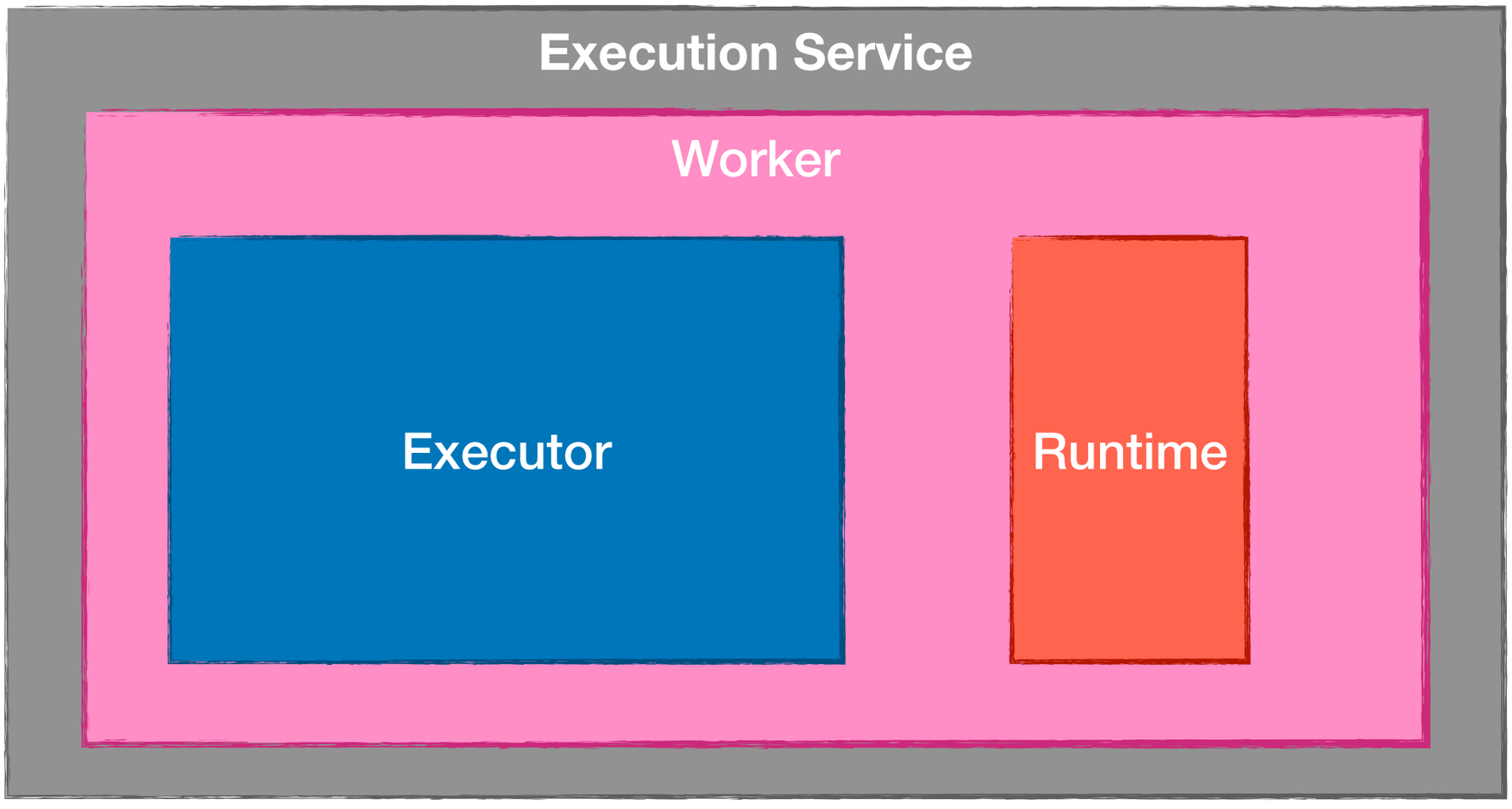
Runtime/Build Configuration

- Build configuration: build-time configuration, i.e., hard-coded in the sources, which may be security-related config.
- Runtime configuration: loaded at runtime, e.g., listen addr.

```
# Teaclave Build Config

# Intel Attestation Service root CA certificate to verify attestation report
as_root_ca_cert = { path = "../keys/ias_root_ca_cert.pem" }
# For DCAP, use the following cert
# as_root_ca_cert = { path = "../keys/dcap_root_ca_cert.pem" }

# Auditors' public keys to verify their endorsement signatures
auditor_public_keys = [
    { path = "../keys/auditors/godzilla/godzilla.public.pem" },
    { path = "../keys/auditors/optimus_prime/optimus_prime.public.pem" },
    { path = "../keys/auditors/albus_dumbledore/albus_dumbledore.public.pem"},
]

# RPC max message size
rpc_max_message_size = 409600

# Validity in seconds for a remote attestation report and endorsed attested TLS config
attestation_validity_secs = 3600

# Specify accepted inbound services to enforce incoming connections via mutual
# attestation. Below figure illustrates current topology of Teaclave services.
#
# client ⇒ authentication ←+       +----> storage <-----+
#                          |       |                    |
# client ⇒ frontend ----------> management            scheduler <-- execution
#                              |
#                              +—→ access_control
#
#                                          ⇒      api endpoint connections
#                                          → internal endpoint connections
[inbound]
access_control = ["teaclave_management_service"]
authentication = ["teaclave_frontend_service"]
storage        = ["teaclave_management_service", "teaclave_scheduler_service"]
management     = ["teaclave_frontend_service"]
scheduler      = ["teaclave_execution_service"]
```

```
# Teaclave Runtime Config
#
# Note that this config is loaded at running time. We don't have to trust the
# content though. Maliciously crafted config from this file will not break data
# confidentiality/integrity.

[api_endpoints]
authentication = { listen_address = "0.0.0.0:7776" }
frontend       = { listen_address = "0.0.0.0:7777" }

[internal_endpoints]
authentication = { listen_address = "0.0.0.0:17776", advertised_address = "localhost:17776" }
management     = { listen_address = "0.0.0.0:17777", advertised_address = "localhost:17777" }
storage        = { listen_address = "0.0.0.0:17778", advertised_address = "localhost:17778" }
access_control = { listen_address = "0.0.0.0:17779", advertised_address = "localhost:17779" }
execution      = { listen_address = "0.0.0.0:17770", advertised_address = "localhost:17770" }
scheduler      = { listen_address = "0.0.0.0:17780", advertised_address = "localhost:17780" }

[audit]
enclave_info = { path = "enclave_info.toml" }
auditor_signatures = [
    { path = "auditors/godzilla/godzilla.sign.sha256" },
    { path = "auditors/optimus_prime/optimus_prime.sign.sha256" },
    { path = "auditors/albus_dumbledore/albus_dumbledore.sign.sha256" },
]

[attestation]
algorithm = "sgx_epid"
url = "https://api.trustedservices.intel.com:443"
key = "00000000000000000000000000000000"
spid = "00000000000000000000000000000000"
```

Worker



Function Implementation

- Native executor
- Define struct and implement trait
- Register functions

Function Implementation

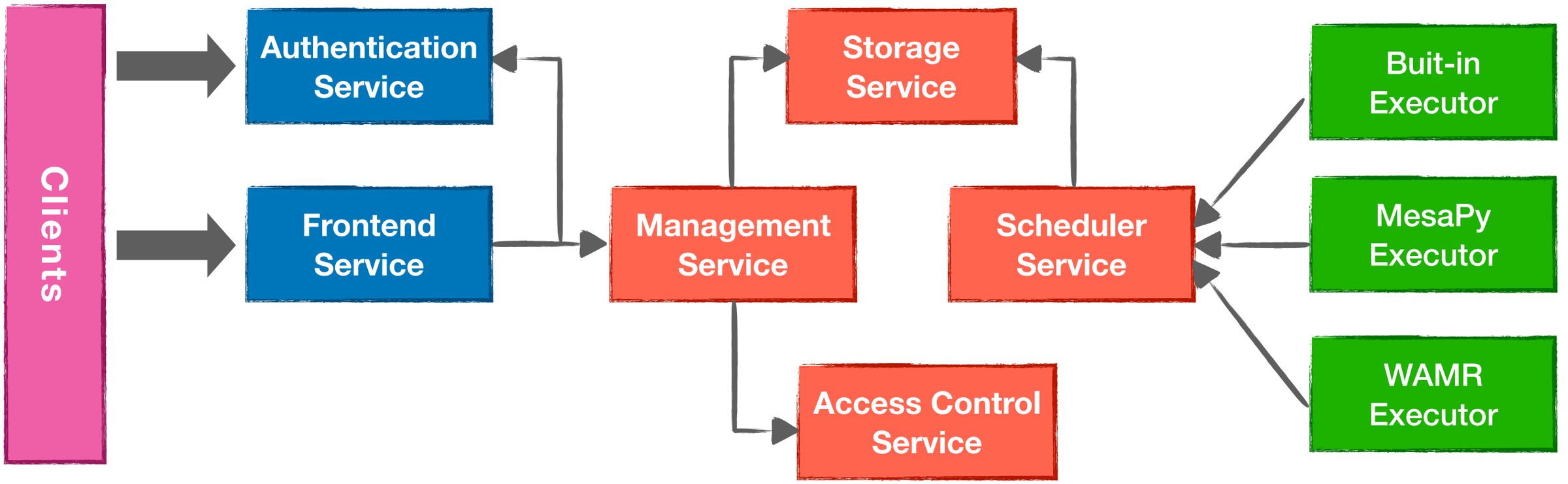
- Define struct and implement trait

```rust
impl TeaclaveFunction for GbdtPrediction {
    fn execute(
        &self,
        runtime: Box<dyn TeaclaveRuntime + Send + Sync>,
        _args: TeaclaveFunctionArguments,
    ) -> anyhow::Result<String> {
        let mut json_model = String::new();
        let mut f = runtime.open_input(IN_MODEL)?;
        f.read_to_string(&mut json_model)?;

        let model: GBDT = serde_json::from_str(&json_model)?;

        let in_data = runtime.open_input(IN_DATA)?;
        let test_data = parse_test_data(in_data)?;

        let predict_set = model.predict(&test_data);

        let mut of_result = runtime.create_output(OUT_RESULT)?;
        for predict_value in predict_set.iter() {
            writeln!(&mut of_result, "{:.10}", predict_value)?
        }

        let summary = format!("Predict result has {} lines of data.", predict_set.len());
        Ok(summary)
    }
}
```

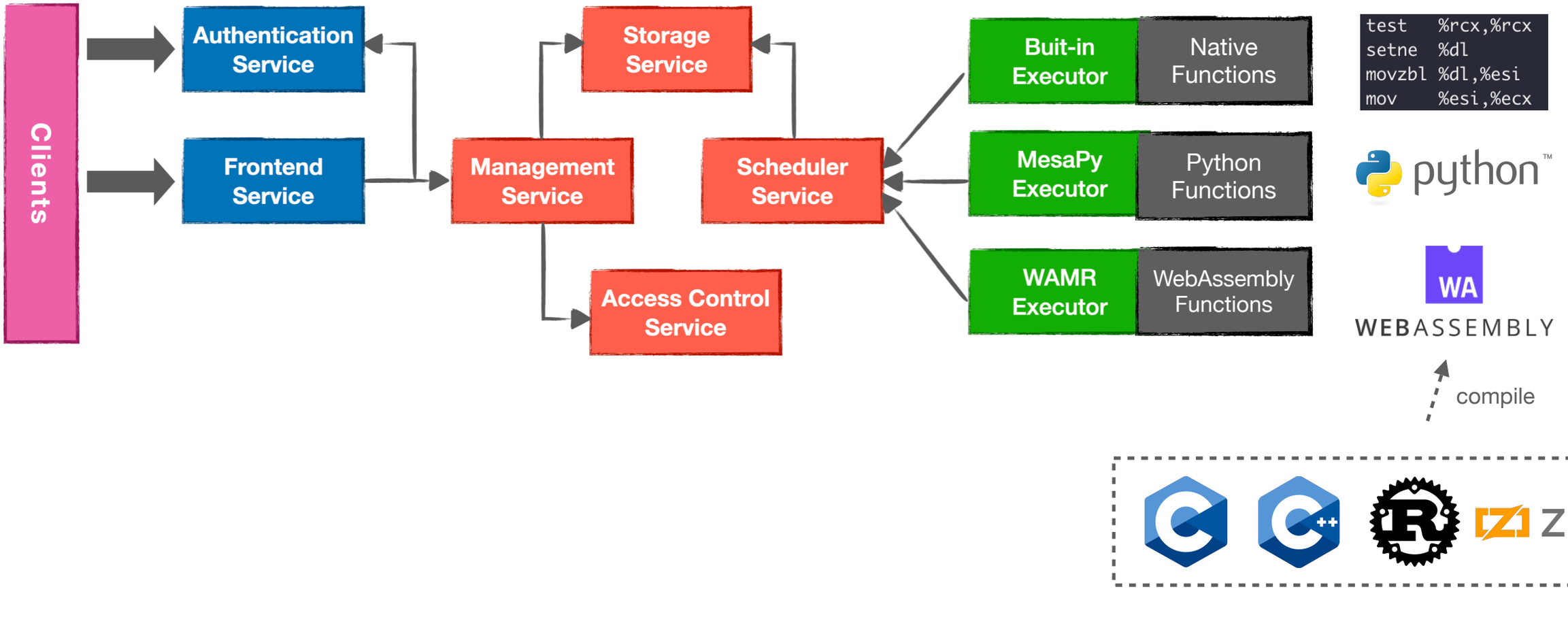
Functions Executors

- **Builtin Executor**
  - functions written in Rust and natively executed
- **MesaPy Executor**
  - functions written in Python and run in MesaPy for SGX
- **WAMR Executor**
  - WebAssembly bytecodes

Function Executors



Function Executors



Functions

Currently, we have these built-in functions:

- `builtin-echo`: Return the original input message.
- `builtin-gbdt-train`: Use input data to train a GBDT model.
- `builtin-gbdt-predict`: GBDT prediction with input model and input test data.
- `bulitin-logistic-regression-train`: Use input data to train a LR model.
- `builtin-logistic-regression-predict`: LR prediction with input model and input test data.
- `builtin-private-join-and-compute`: Find intersection of muti-parties' input data and compute sum of the common items.
- `builtin-ordered-set-intersect`: Allow two parties to compute the intersection of their ordered sets without revealing anything except for the elements in the intersection. Users should calculate hash values of each item and upload them as a sorted list.
- `builtin-rsa-sign`: Signing data with RSA key.
- `builtin-face-detection`: An implementation of Funnel-Structured cascade, which is designed for real-time multi-view face detection.
- `builtin-principal-components-analysis`: Example to calculate PCA.
- `builtin-password-check`: Given a password, check whether it is in the exposed password list.

https://teaclave.apache.org/docs/codebase/function/

WebAssembly Examples

Machine Learning Models

Apache TVM

WebAssembly Bytecode

Privacy-preserving Machine Learning

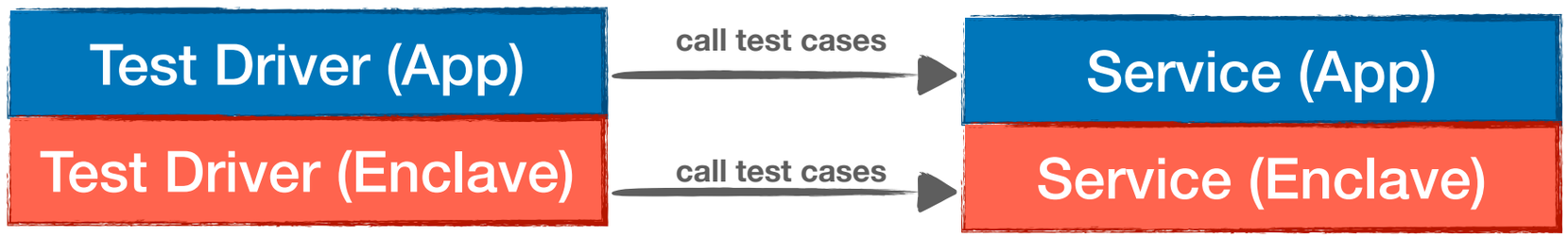
Teaclave

Test

- **unit test**: small and internal tests
- **integration test**: public API tests
- **functional test**: protocol-level tests

Test

- Unit tests in enclave




```
235 #[cfg(feature = "enclave_unit_test")]
236 pub mod tests {
237     use super::*;
238     use std::sync::mpsc::channel;
239     use teaclave_rpc::IntoRequest;
240
241     fn get_mock_service() -> TeaclaveStorageService {...}
254
255     pub fn test_get_key() {
256         let service = get_mock_service();
257         let request = GetRequest::new("test_get_key").into_request();
258         assert!(service.get(request).is_ok());
259     }
260
261     pub fn test_put_key() {...}
268
269     pub fn test_delete_key() {...}
276
277     pub fn test_enqueue() {...}
284
285     pub fn test_dequeue() {...}
298 }
```

Test

- Integration tests in enclave

```rust
pub fn run_tests() -> bool {
    use teaclave_test_utils::*;

    start_echo_service();

    run_tests!(echo_success)
}

fn start_echo_service() {...}

fn echo_success() {
    use super::*;

    let channel = Endpoint::new("localhost:12345").connect().unwrap();
    let mut client = EchoClient::new(channel).unwrap();
    let request = SayRequest {
        message: "Hello, World!".to_string(),
    };
    let response_result = client.say(request);
    info!("{:?}", response_result);

    assert!(response_result.is_ok());
    assert!(response_result.unwrap().message == "Hello, World!");
}
```

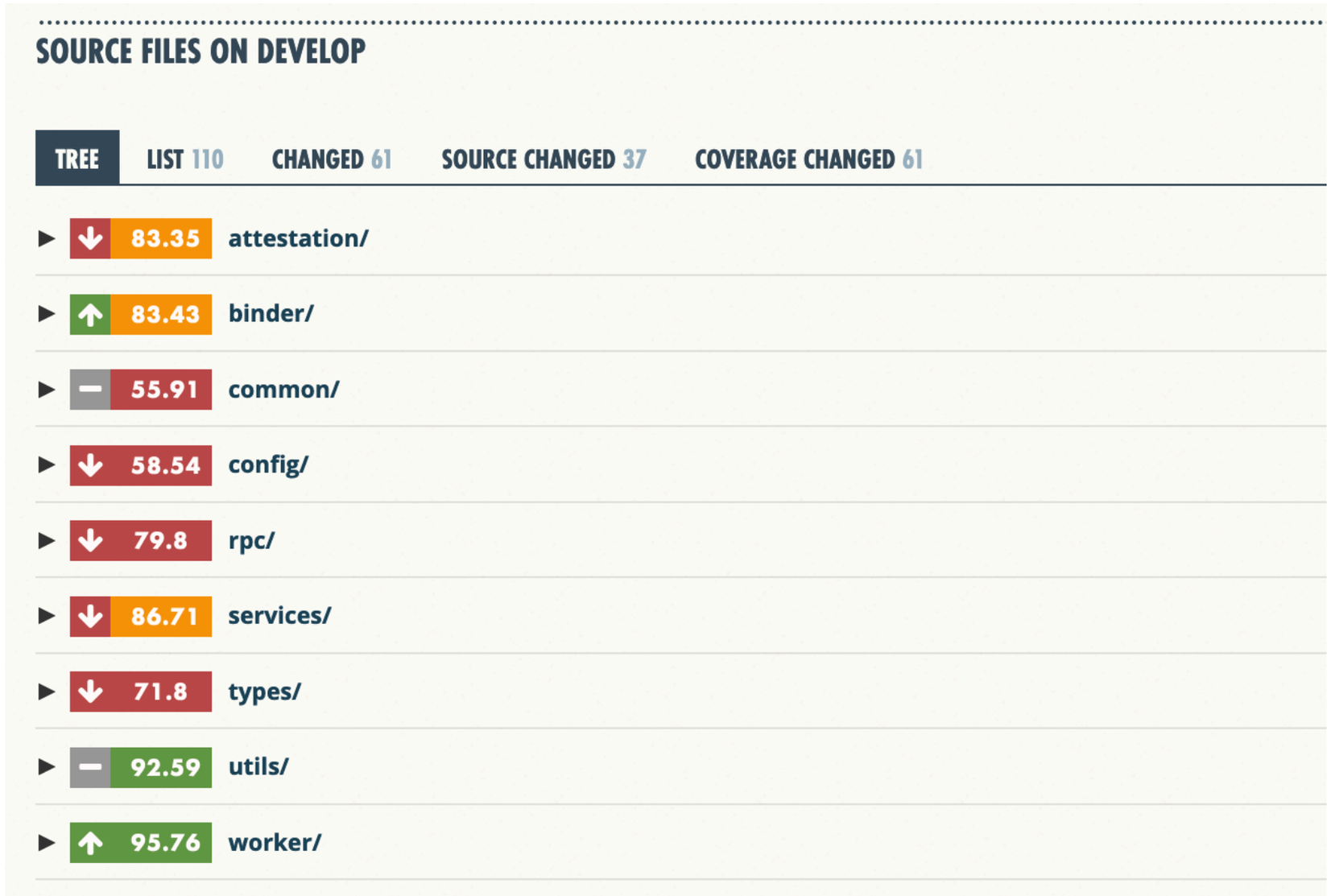
Test

- Functional tests in enclave

```
11 pub fn run_tests() → bool {
12     use teaclave_test_utils::*;
13
14     run_tests!(
15         test_login_success,
16         test_login_fail,
17         test_authenticate_success,
18         test_authenticate_fail,
19         test_register_success,
20         test_register_fail,
21     )
22 }
23
24 fn get_api_client() → TeaclaveAuthenticationApiClient {...}
43
44 fn get_internal_client() → TeaclaveAuthenticationInternalClient {...}
68
69 fn test_login_success() {...}
80
81 fn test_login_fail() {...}
92
93 fn test_authenticate_success() {...}
109
110 fn test_authenticate_fail() {...}
124
125 fn test_register_success() {...}
132
133 fn test_register_fail() {...}
```

Test

- Coverage



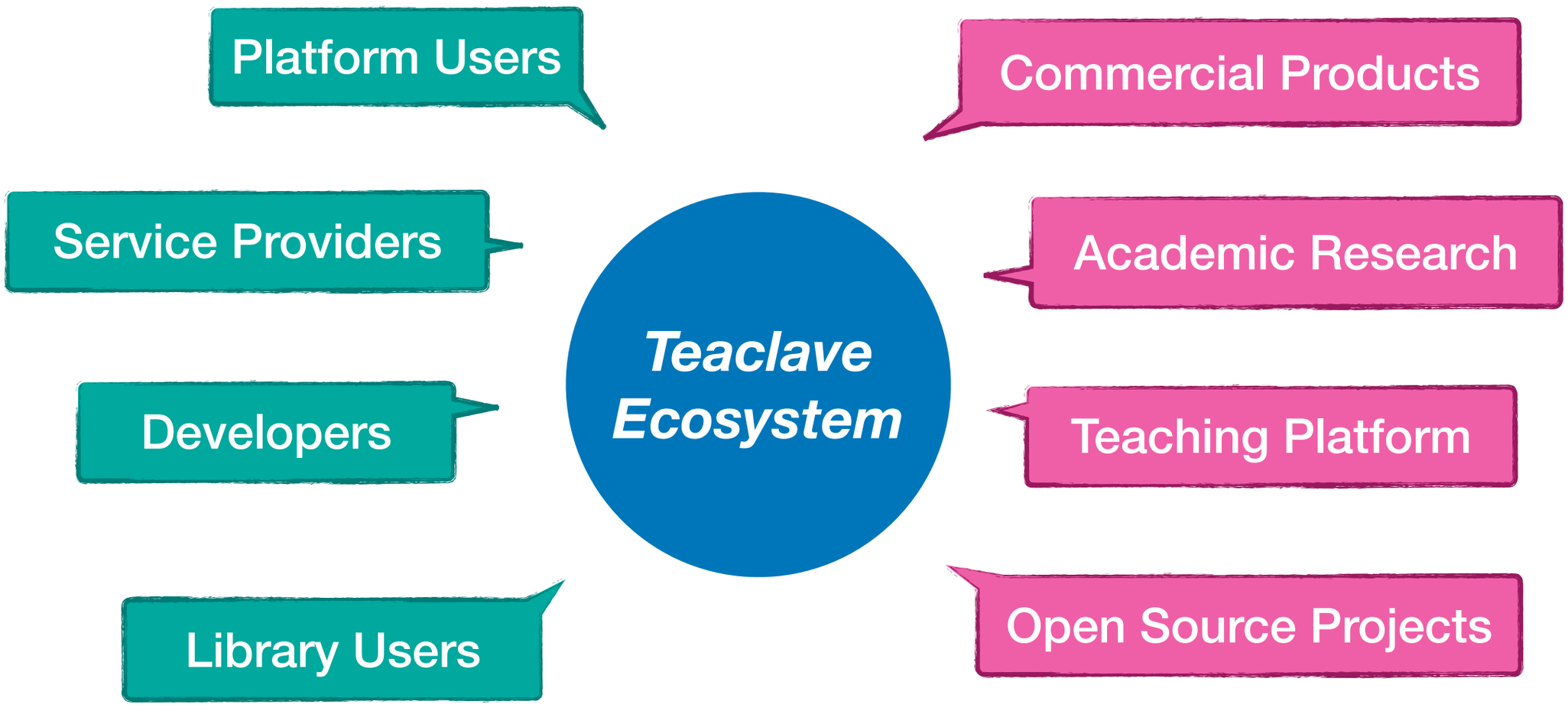
Getting Started

- **Try**
  - My First Function
  - Write Functions in Python
  - How to Add Built-in Functions
- **Design**
  - Threat Model
  - Mutual Attestation: Why and How
  - Access Control
  - Build System
  - Teaclave Service Internals
- **Contribute**
  - Rust Development Guideline
  - Development Tips
- **Codebase**

Documentation

https://teaclave.apache.org/docs/

Teaclave Community



Projects Powered by Teaclave

ADVANCA

Baidu 百度

PHALA NETWORK

substraTEE by SGS

Occlum

LayerX

enigma

Anonify

crypto.com

CONFIDENTIAL COMPUTING CONSORTIUM | Veracruz

secret network

INCLAVARE CONTAINERS

OP-TEE .org

Alibaba Group

蚂蚁集团 ANT GROUP

ByteDance 字节跳动

https://teaclave.apache.org/powered-by/

Projects Powered by Teaclave

Organizations

- Alibaba
- Ant Group
- Baidu
- ByteDance
- Enigma
- LayerX

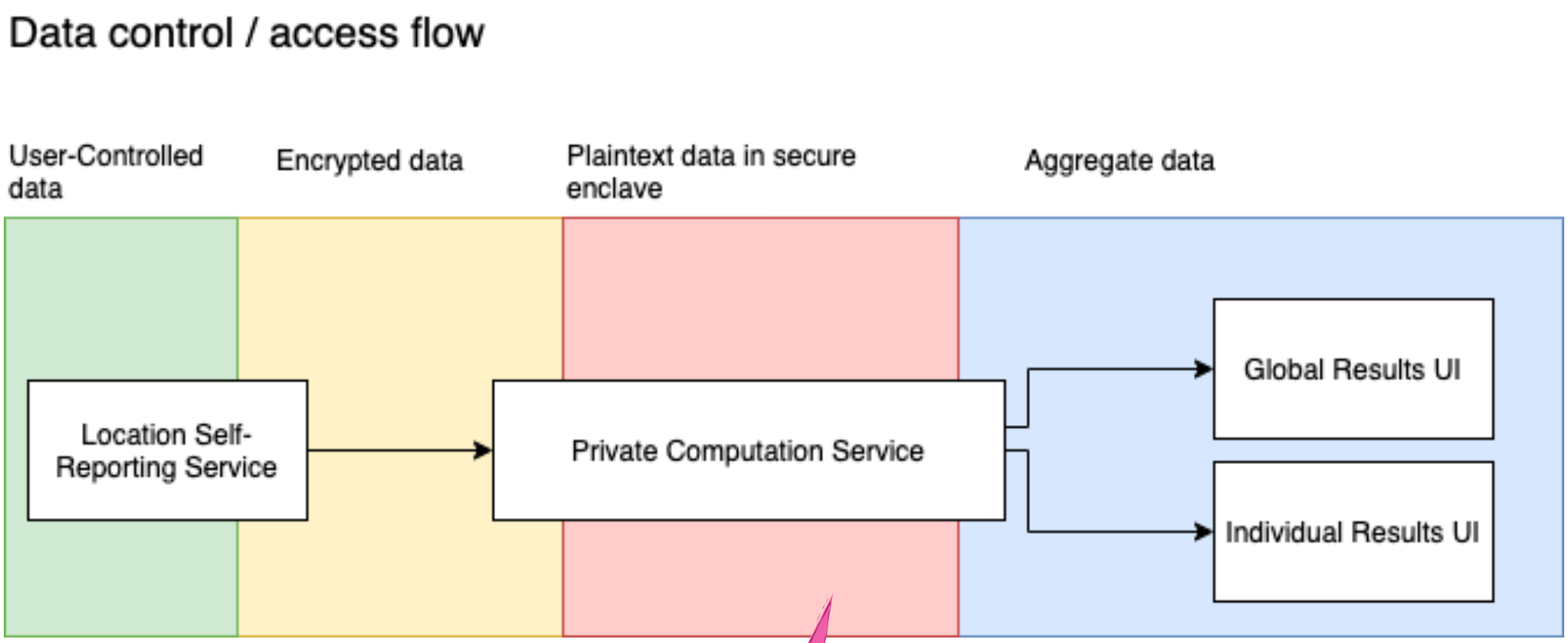
Projects

- Advanca: A privacy-preserving general-purpose compute/storage infrastructure for Dapps.
- Anonify: A blockchain-agnostic execution environment with privacy and auditability based on TEE.
- Eigen Network: Eigen is an end-to-end privacy computation network for a better digital economy based on hybrid privacy computation protocols and AI federated machine learning.
- Enigma Core: Enigma Core library. The domain: Trusted and Untrusted App in Rust.
- Crust Network: A decentralized storage blockchain of Web3.0 ecosystem based on TEE and Substrate.
- Crypto.com Chain: Alpha version prototype of Crypto.com Chain.
- Inclavare Containers: A novel container runtime, aka confidential container, for cloud-native confidential computing and enclave runtime ecosystem.
- Occlum: Occlum is a memory-safe, multi-process library OS for Intel SGX.
- Phala Network: A TEE-Blockchain hybrid architecture implementing Confidential Contract on Polkadot.
- SafeTrace: Privacy preserving voluntary COVID-19 self-reporting platform for contact tracing.
- Secret Network: A blockchain-based, open-source protocol that lets anyone perform computations on encrypted data, bringing privacy to smart contracts and public blockchains.
- substraTEE: Trusted Off-Chain Compute Framework for substrate blockchains.
- Veracruz: Veracruz is a framework for defining and deploying collaborative, privacy-preserving computations amongst a group of mutually mistrusting individuals.

Case Study - MesaTEE by Baidu

Case Study - SafeTrace: COVID-19 Self-reporting with Privacy

- Developed by Enigma
- a privacy-preserving data sharing and analytics platform



Powerd by Teaclave SGX SDK

Thank you!

- **Join us on our mailing list**: https://lists.apache.org/list.html?dev@teaclave.apache.org
- **Visit our homepage**: https://teaclave.apache.org/
- **Follow us at** @ApacheTeaclave
- **Discord**: https://discord.gg/ynECXsxm5P
- **Checkout our code**: https://github.com/apache/incubator-teaclave
- **Contributors**: https://teaclave.apache.org/contributors/
- **Call for collaborations and contributors!**

Thanks!