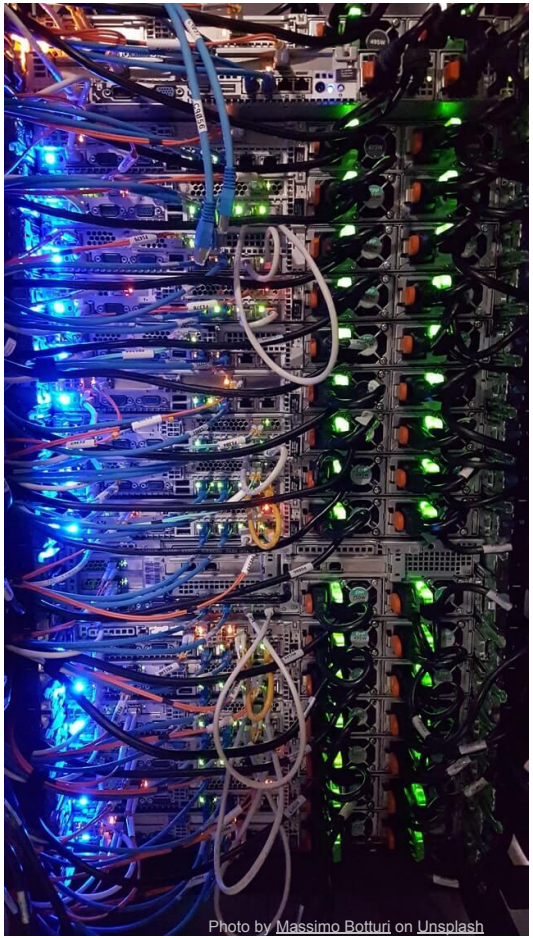
HYPERLEDGER CACTI

A Framework for Integrating Distributed Ledgers

Peter Somogyvari,
Technology Architect, *Accenture*

What is it?

- A pluggable, enterprise-grade framework to transact on multiple distributed ledgers without introducing yet another competing blockchain
- An SDK of SDKs



Why?

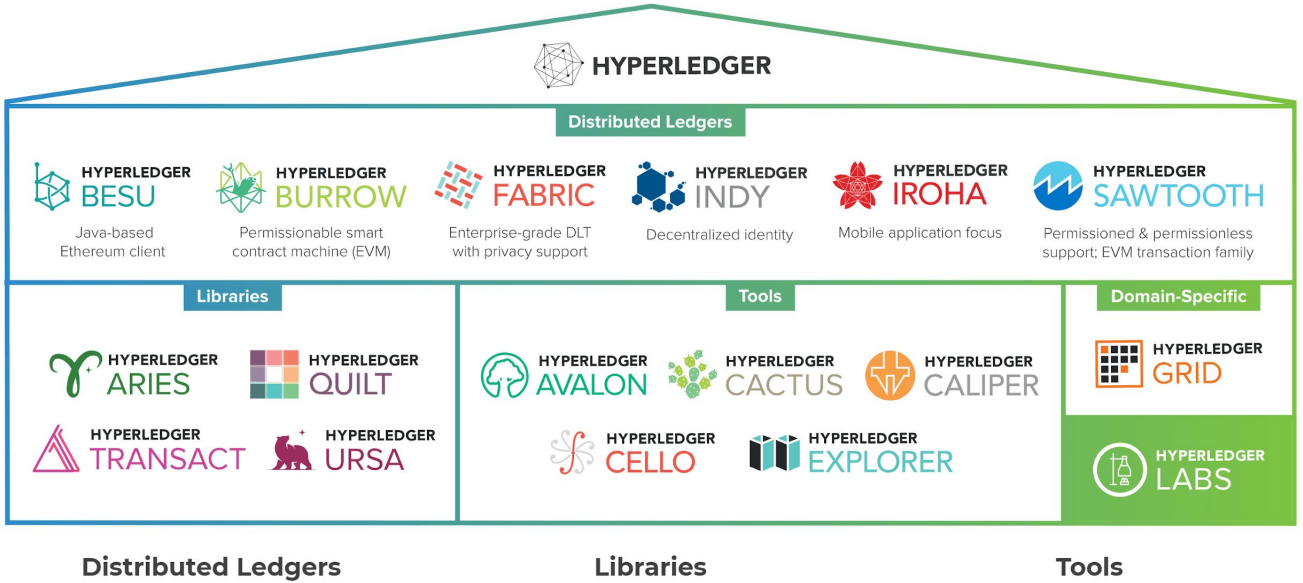
1. To address blockchain/DLT fragmentation
2. Save (distributed) app. developers from re-inventing the wheel
3. Lower risk of adopting distributed ledgers by businesses

How bad is fragmentation?

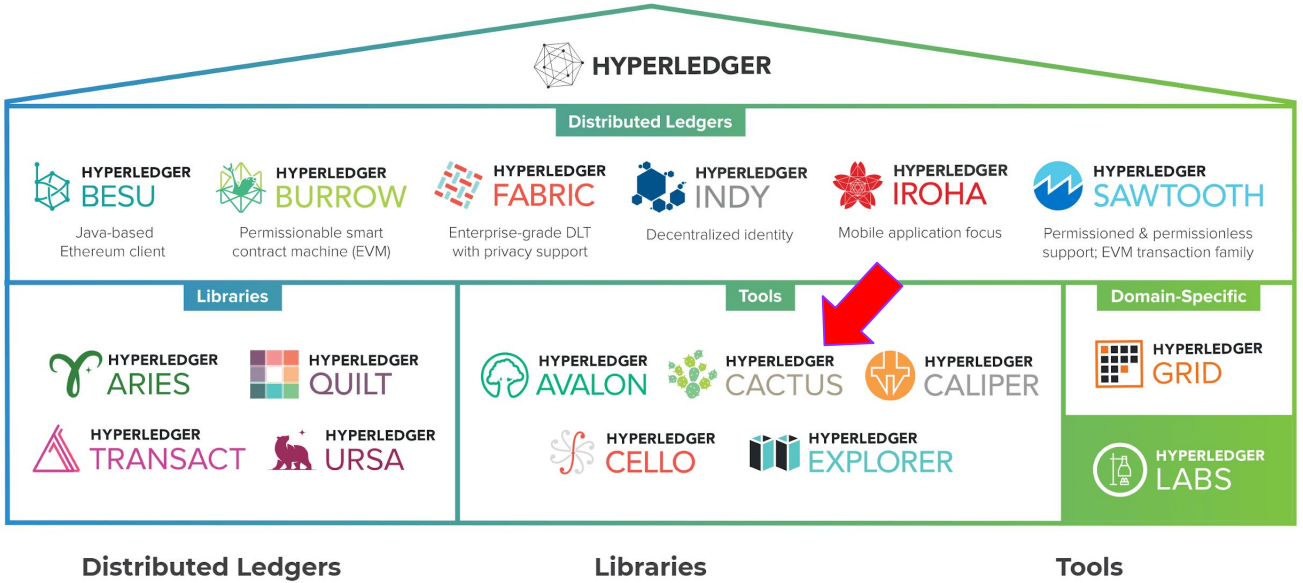
- Integrations required to have a fully connected network (graph) of ledgers (not nodes) grows quadratically with the number of ledgers.
- Not quite as bad as the exponential growth, but still pretty bad…

- **n**: The number of ledgers in existence
- **c**: The possible number of integrations between ledgers
  - `c = n * (n-1) / 2`
  - 100 ledgers => 5k integration scenarios

Position in the Hyperledger Greenhouse

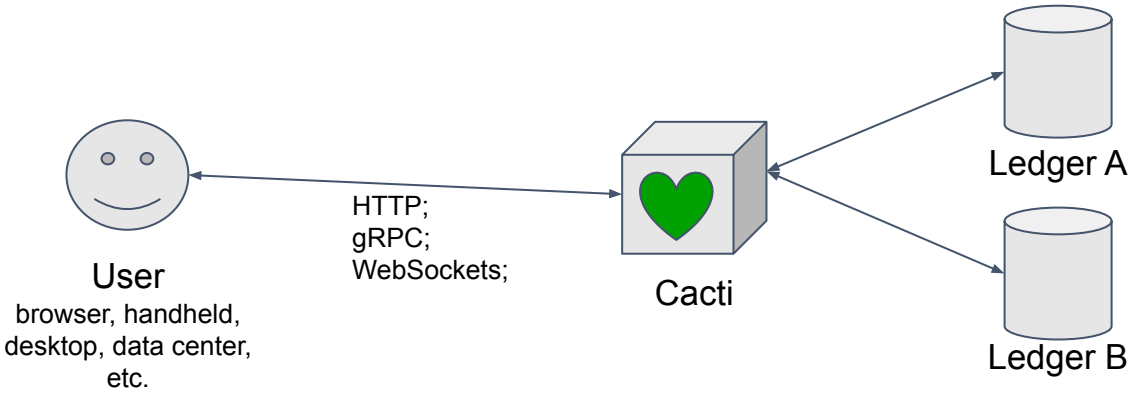


Position in the Hyperledger Greenhouse



Generic Use Case

1. Business case where multiple ledgers are involved
2. Application adds value by somehow making said ledgers data/assets work together (e.g. transactions have to happen on both ledgers)



Key Hyperledger Cacti Design Principles

Plugin Architecture

Maximize **flexibility** and **future-proofing** through plug-in architecture

Secure by Default

Avoid needing explicit action from users to have a secure Cacti deployment.

Toll Free

Users should not be required to use tokens for transactions & **Operators** should not be required to take a cut of individual transactions

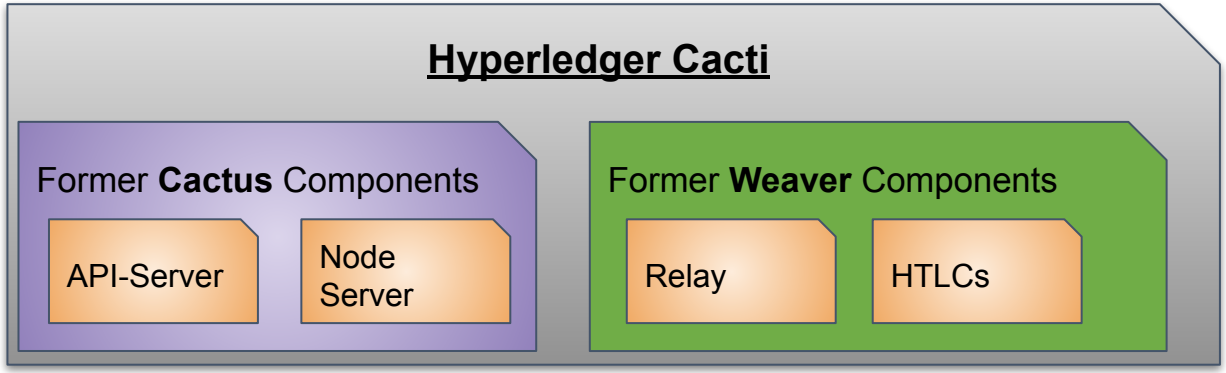
Low Impact Deployment

Do not interfere with or impede **existing** network requirements

Additional Design Principles

- Wide Support
- Prevent Double Spending (where applicable)
- Preserving Ledger Features
- Horizontal Scalability
- (Complete list in the whitepaper)

Hyperledger Cacti Architecture



Architecture Decisions

- The code is written mostly in Typescript and Rust
- We use Lerna for managing a mono-repo where packages can be:
  - Server-side: NodeJS only
  - Client-side: Browser only
  - Cross-platform/Universal: NodeJS and Browser as well
  - Any other language such as Rust, go, Kotlin/Java, etc.
- Test **Automation**

Plugin Architecture

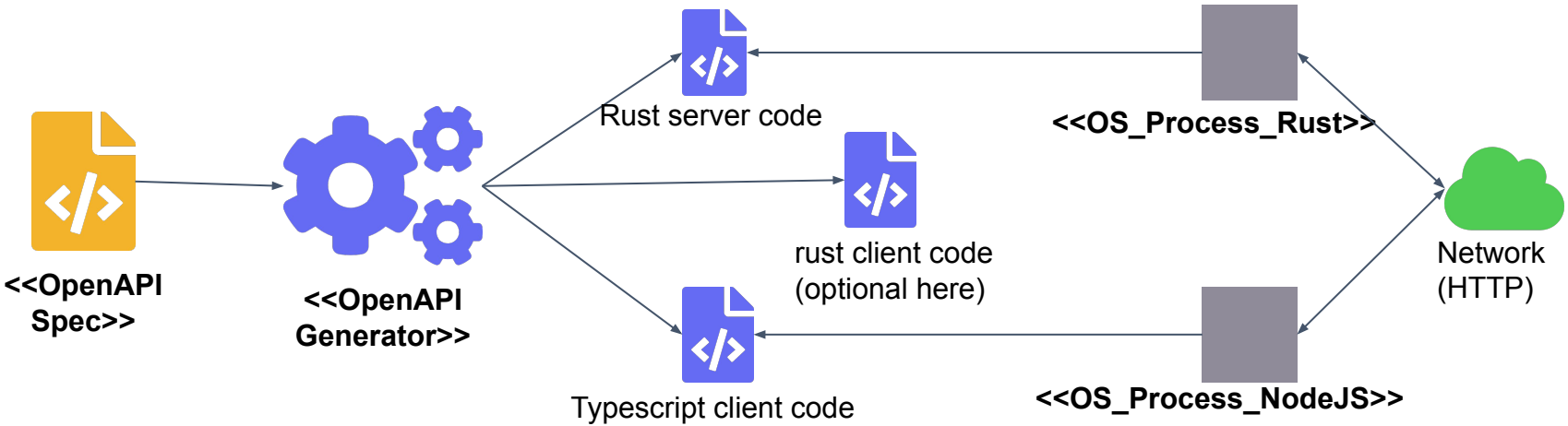
- Nobody knows the future
- We can prepare for it with software that bends not breaks when major technological shifts occur.

Plugin Development - Governance Model

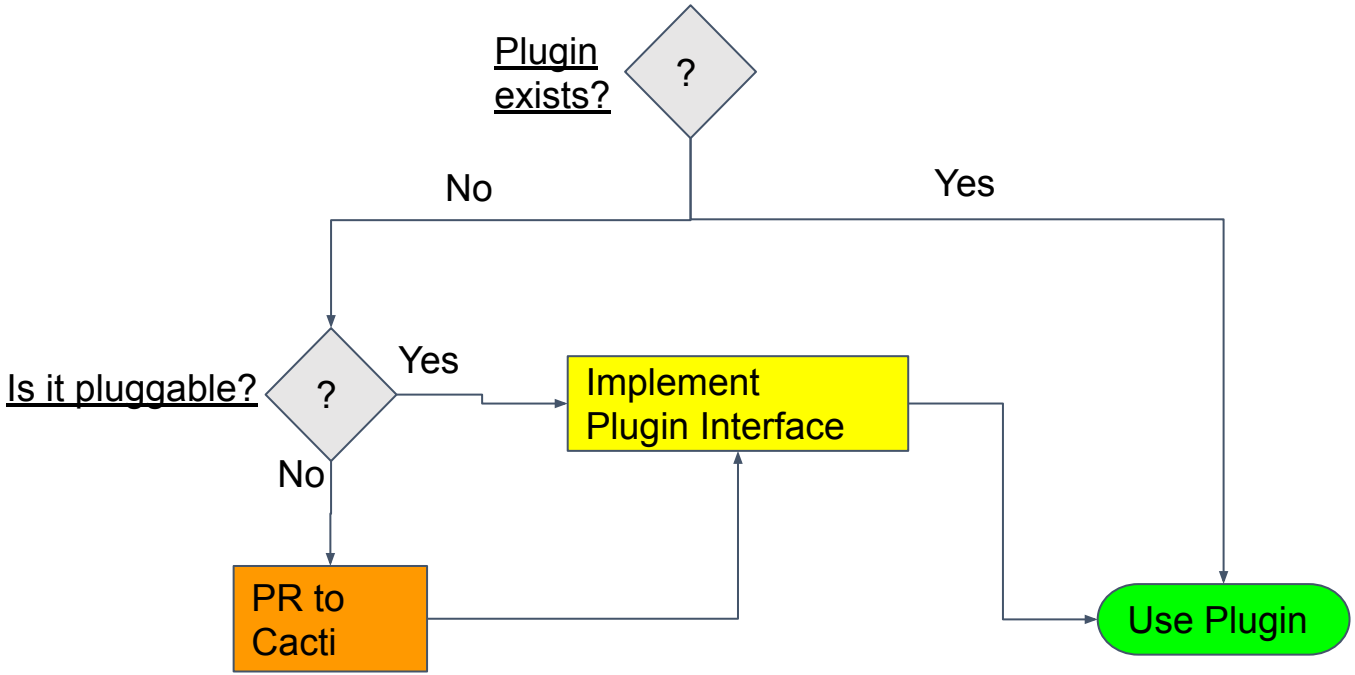
- Plugin projects can live outside of the main Cacti repo
- Anyone can create a new plugin and use/promote it independently
- There is no need to have your plugin implementation approved by the maintainers
  - Just implement the published interfaces and publish your code, done.

Language Agnostic Plugin Development

- You can write Cacti plugins in any language!



What About Supporting X?



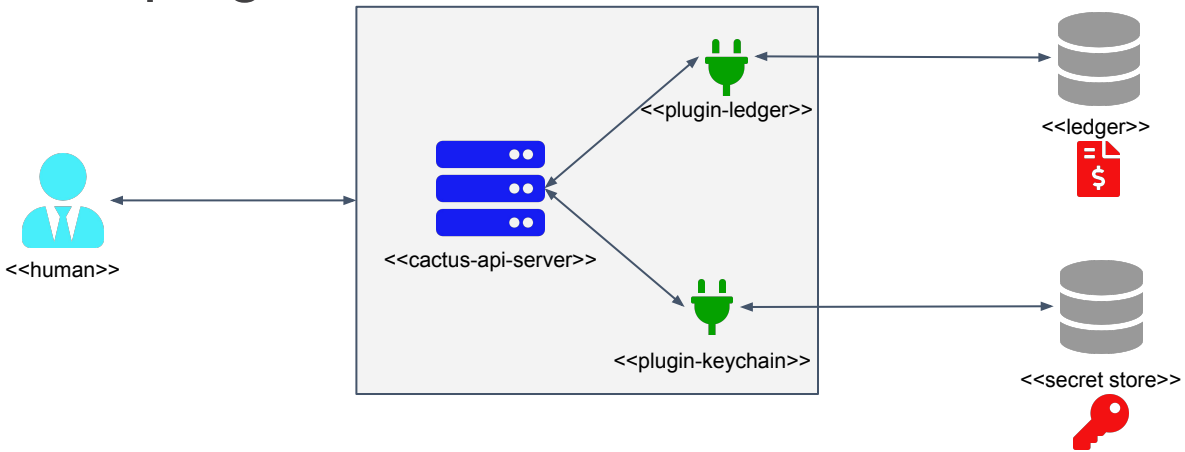
Plugin Example - Keychain

- The interface is meant to be really simple:

```
export interface IPluginKeychain extends ICactusPlugin {
    getKeychainId(): string;
    has(key: string): Promise<boolean>;
    get<T>(key: string): Promise<T>;
    set<T>(key: string, value: T): Promise<void>;
    delete<T>(key: string): Promise<void>;
}
```

Plugin Example - Keychain

- Store secrets that other plugins can retrieve
- Role Based Access Control is available on a per Endpoint Basis for all plugins



Preserve Ledger Features

- Do not limit a transaction to the intersection of features of two participating ledgers
  - Example: If both ledgers have private transactions, enable it through Cacti
  - No unexpected behaviour from either side of a transaction: If only one side supports private transactions there should be no expectation of privacy in the scope of the transaction

Performance and Scalability

- Bottleneck should always be the ledger, not Cacti
- Coming soon:
  - horizontal scalability for the REST API server and the validator node components.
  - Published benchmarks with performance characteristics broken down by
    - Supported ledgers (Fabric, Quorum, Besu, Corda) and
    - Supported tasks (Read/Write Ledger, Sign Transaction, Verify Transaction)

Supply Chain App (Quorum+Besu+Fabric) Demo

```
$ docker run \
      --rm \
      --privileged \
      -p 3000:3000 \
      -p 3100:3100 \
      -p 3200:3200 \
      -p 4000:4000 \
      -p 4100:4100 \
      -p 4200:4200 \
      hyperledger/cactus-example-supply-chain-app:2021-03-24-feat-362
```

Visit http://localhost:3100 for the demo GUI

(might take a several minutes for the container to boot up)

Roadmap (*Subject to Change*)

- Finalize 2.0 Design and Technical Architecture
- Perform Merge of Codebases from Weaver & Cactus
    - Issue 2.0.0-GA release
        - Use-cases and documentation are up to date with the latest architectural changes and the Weaver+Cactus codebases can be part of the same deployment
- Identity
    - Bootstrapping with different Trust Anchors through plugins (JSON Web Signatures, X509, Indy, DID, DIF)
- Implementation
    - Corda v5.0 Support
    - Corda Flow Invocation JSON DSL Simplified (currently functional, but cumbersome due to JVM type system)
    - Cross-ledger Atomic Swaps - HTLC

Roadmap (*Subject to Change*)

- Documentation
  - Central Bank Digital Currency Example
  - Reference Architecture Documented
  - First public test deployment
    - Transact Permissioned & Permissionless stablecoins
    - How to use with common wallets
- Performance
  - Performance benchmarks with regularly updated and published results
  - “One-click” scripts to run full benchmark suite on provided infrastructure provider (cloud vendor)

Join the Cacti Community!

https://chat.hyperledger.org/channel/cacti

https://github.com/hyperledger/cacti

https://wiki.hyperledger.org/display/cactus