HYPERLEDGER

BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Hyperledger Sawtooth Blockchain Technical Overview

Dan Anderson, Intel
(based on slides from Dan Middleton)

November 2018

Agenda

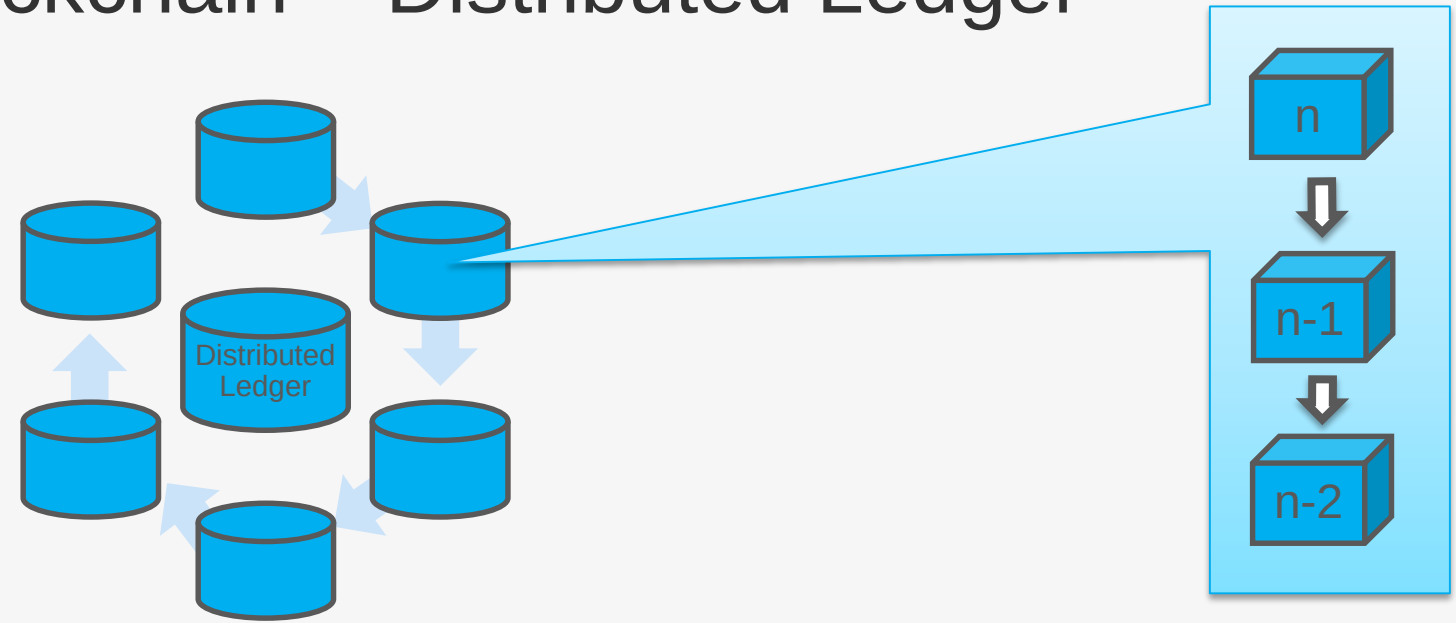
Blockchain Basics

Sawtooth Design Motivations

Sawtooth Architecture and Features

Sawtooth Application Development

Blockchain = Distributed Ledger

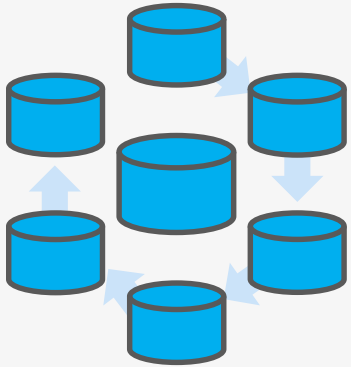


Each node is an instance of a database (ledger) managed by all participants.

Within each database, blocks of transactions are cryptographically chained in order.

Why Blockchain?

Mutually distrusting organizations (“fremenies”) that update the same distributed ledger



Immutable transaction history

- blocks are never deleted

High availability

- Crash fault tolerant (CFT)
- Byzantine fault tolerant (BFT) – protects against bad actors
- Liveness – nodes eventually agree (finite loop)

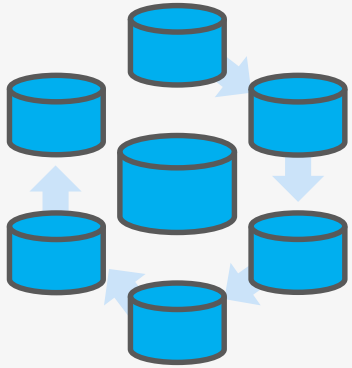
Why Not Blockchain?

Active Research Areas:

- Throughput
- “Private” Transactions

Wrong Usage Model:

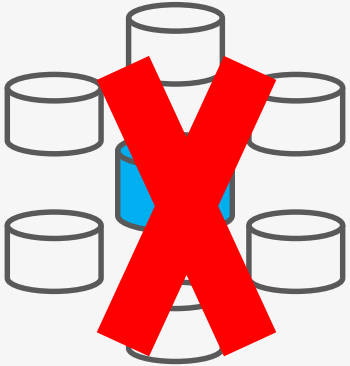
- Internal-only Business Process



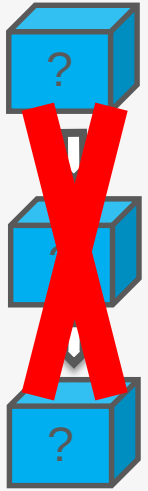
When people talk about *blockchain security*, they mostly mean availability and integrity guarantees.
Confidentiality is open research.

Bad Enterprise Blockchain Shortcuts

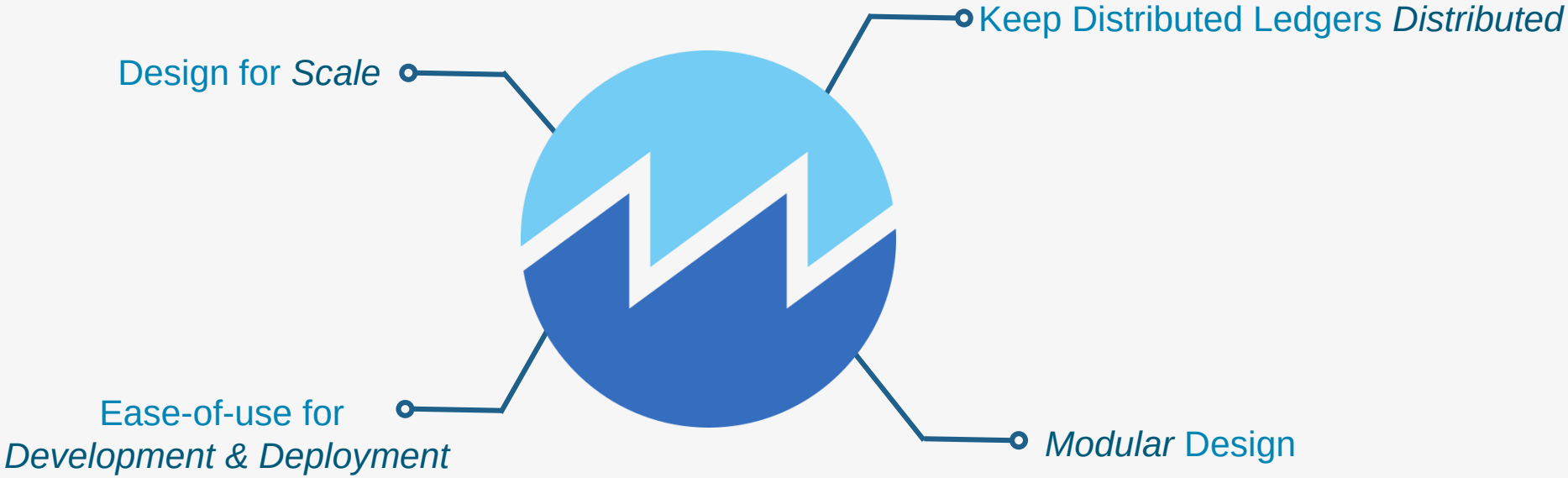
- Centralized Architectures – Removes the main value of a distributed ledger



- No Ledger State (database fields) – Committing only transaction receipts, not the data itself, turns the blockchain into a opaque event log



Sawtooth Design Philosophy



HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Hyperledger Sawtooth 1.0 Architecture & Features

1.0 Released January 2018

v1.0 Highlighted New Features

Advanced Transaction Execution

- Parallel Execution
- Multi-Language Support
  - Build apps in your language of choice

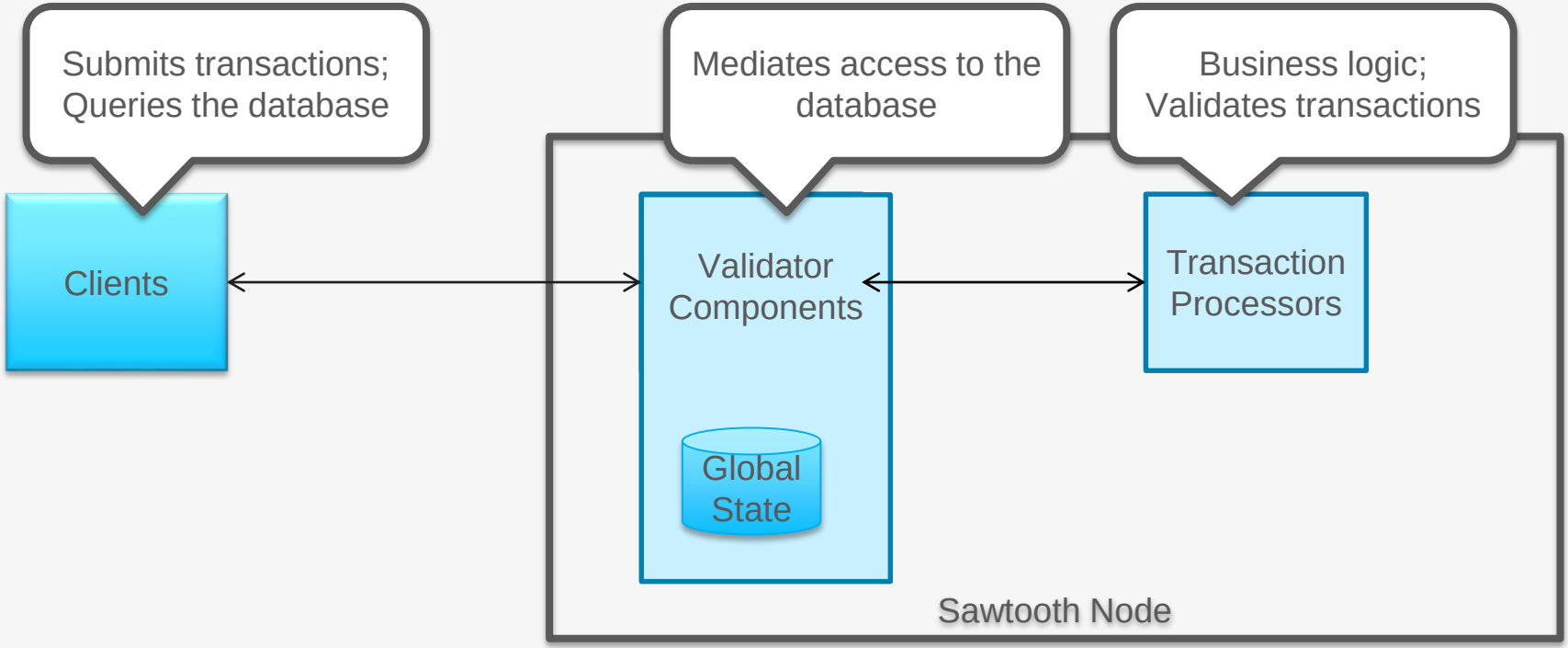
On-chain Governance

- Dynamic Consensus
  - Proof of Elapsed Time (PoET)
- New Permissioning Features

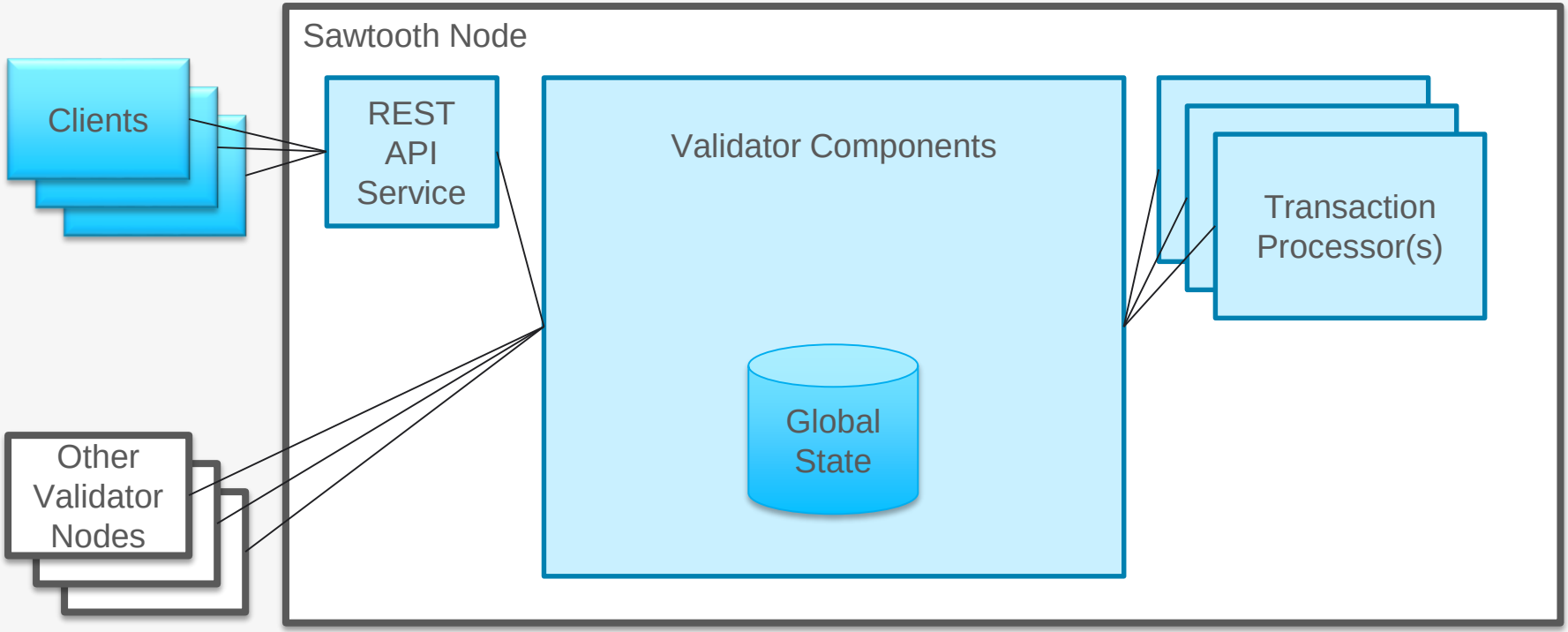
Distributed Applications

- Seth
  - Ethereum on Sawtooth
  - Run Solidity smart contracts
- Supply Chain
  - Provenance of goods
  - Telemetry / tracking

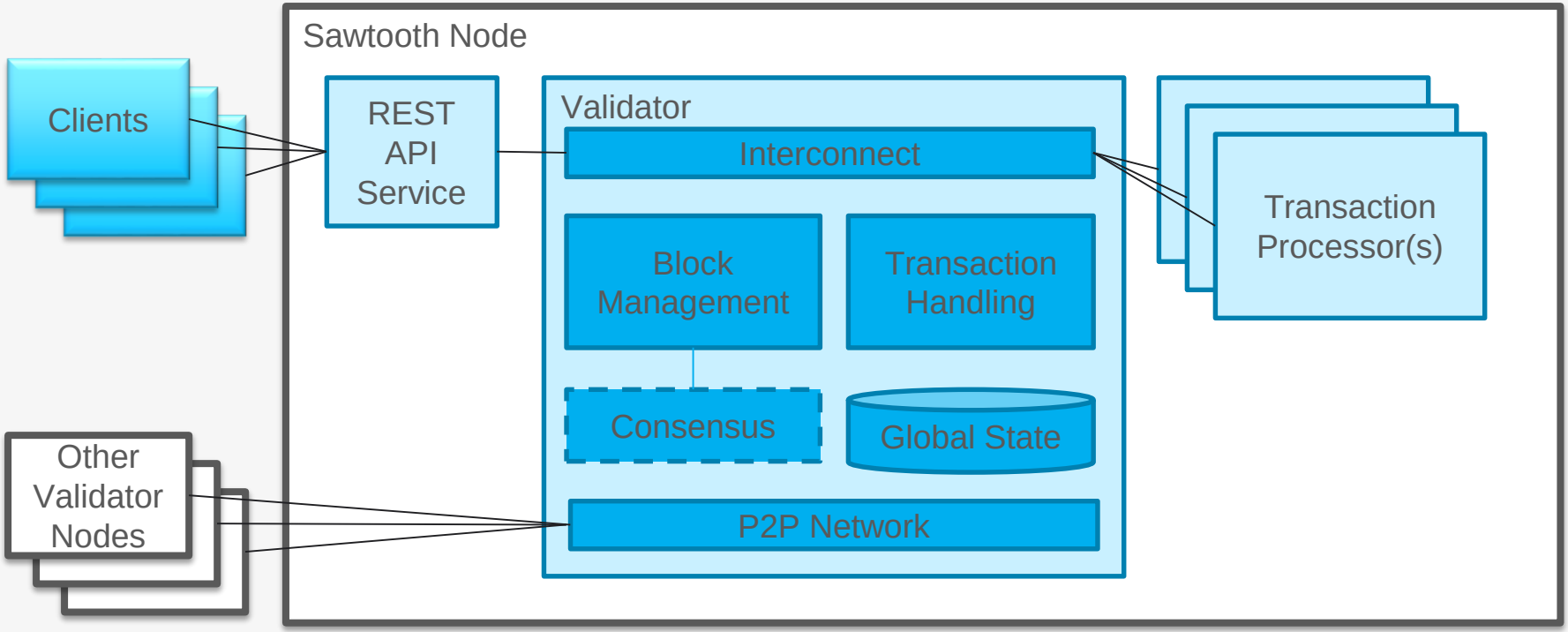
Basic Sawtooth Concept



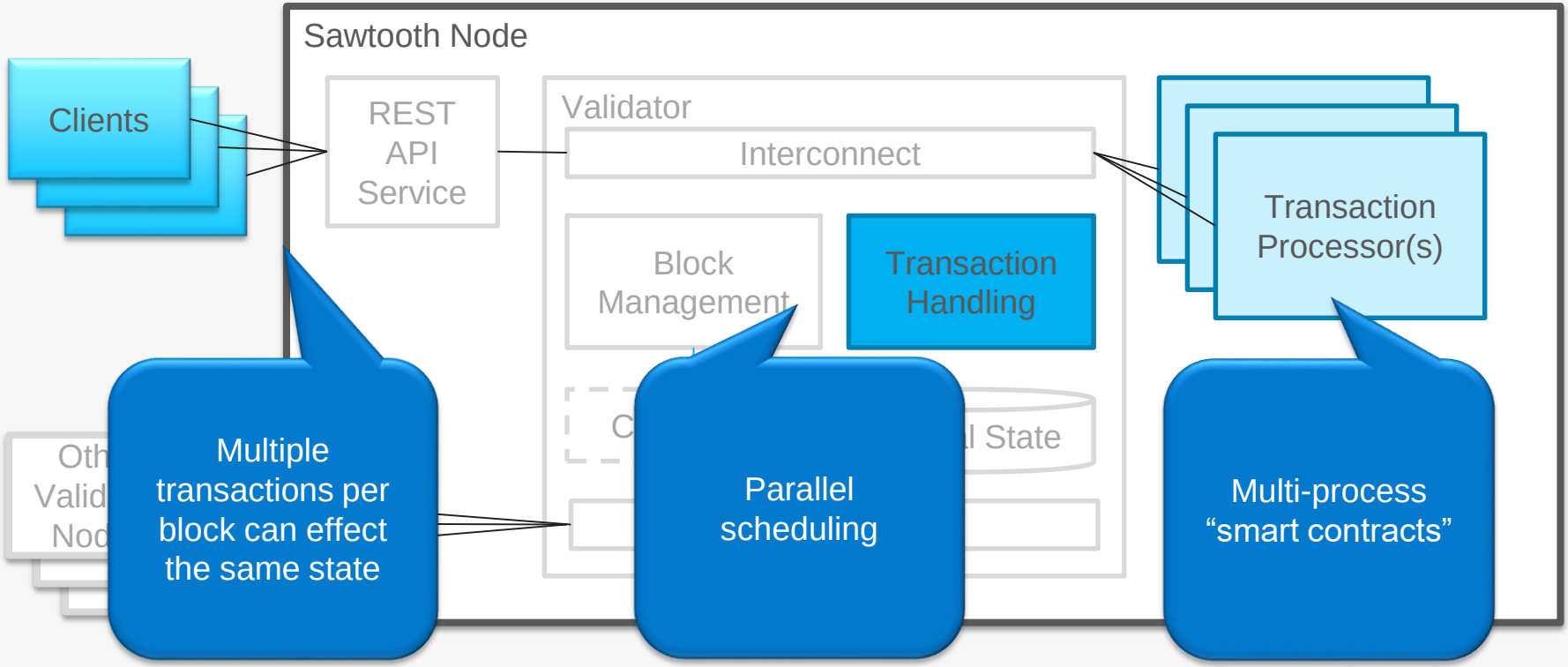
A Couple More Pieces



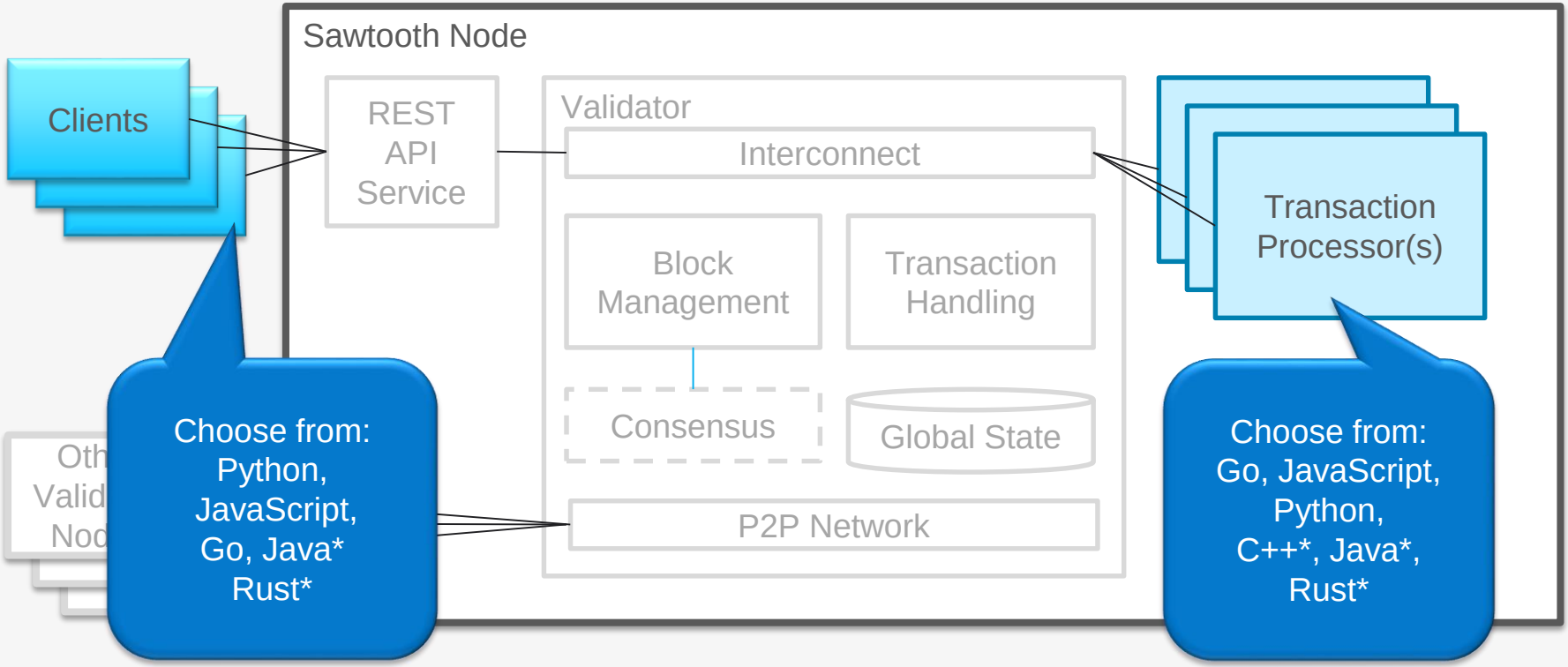
High-level Sawtooth Architecture



Transaction Processing: Parallel Execution

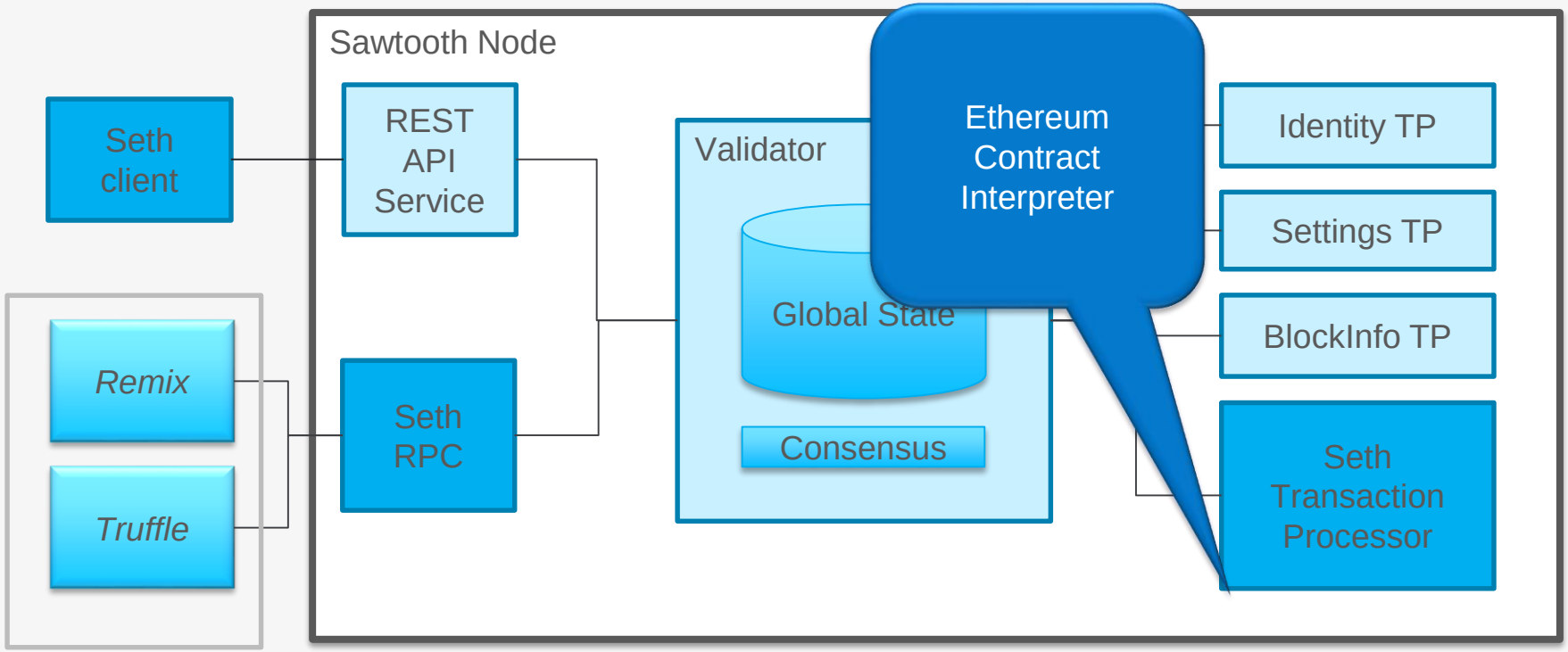


SDK: Multi-Language Support



*Not all features are available in these languages in v1.0.5

Seth: Ethereum Transaction Processor



On-chain Blockchain Governance

Control the blockchain on the blockchain

Settings Transaction Family enables participants to agree on network policies

For example, vote on changing consensus parameters using registered public keys of consortia members.

Settings are extensible – they can be added after genesis.

Setting (Examples)	Value
sawtooth.poet.target_wait_time	5
sawtooth.validator.max_transactions_per_block	100000

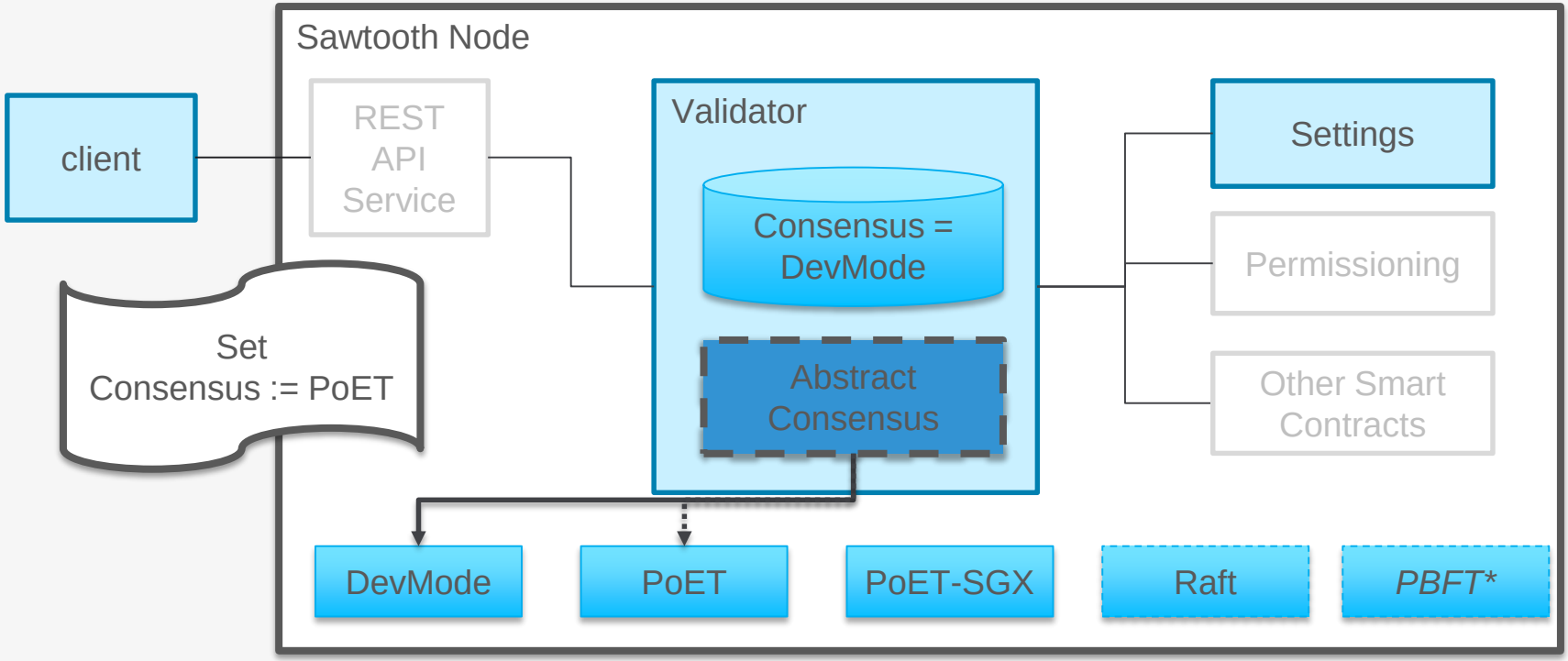
sawtooth.validator.transaction_families	[{
	"family": "intkey",
	"version": "1.0"
	},
	{
	"family": "xo",
	"version": "1.0"
	}]

Consensus Algorithms

Leader is who can add a block to the blockchain
Consensus is agreement among nodes on leader
Consensus Types:
Byzantine Fault Tolerance (BFT) vs. *Crash Fault Tolerance* (CFT)
Classical Consensus (election) vs. *Nakamoto Consensus* (lottery

Fault Tolerance	Type	Consensus Algorithm
BFT	Lottery	Proof of Work (PoW) – classic Bitcoin/Ethereum mining (energy waste)
BFT	Lottery	Proof of Elapsed Time (PoET) – SGX (Sawtooth). Uses a random timer
CFT	Lottery	PoET CFT– PoET without SGX (simulator)
BFT	Election	Practical Byzantine Fault Tolerance (PBFT). Used in DB replication. Does not scale, $O(n^2)$. In development
CFT	Election	Raft–uses an elected leader; fast

Dynamic Consensus Algorithm



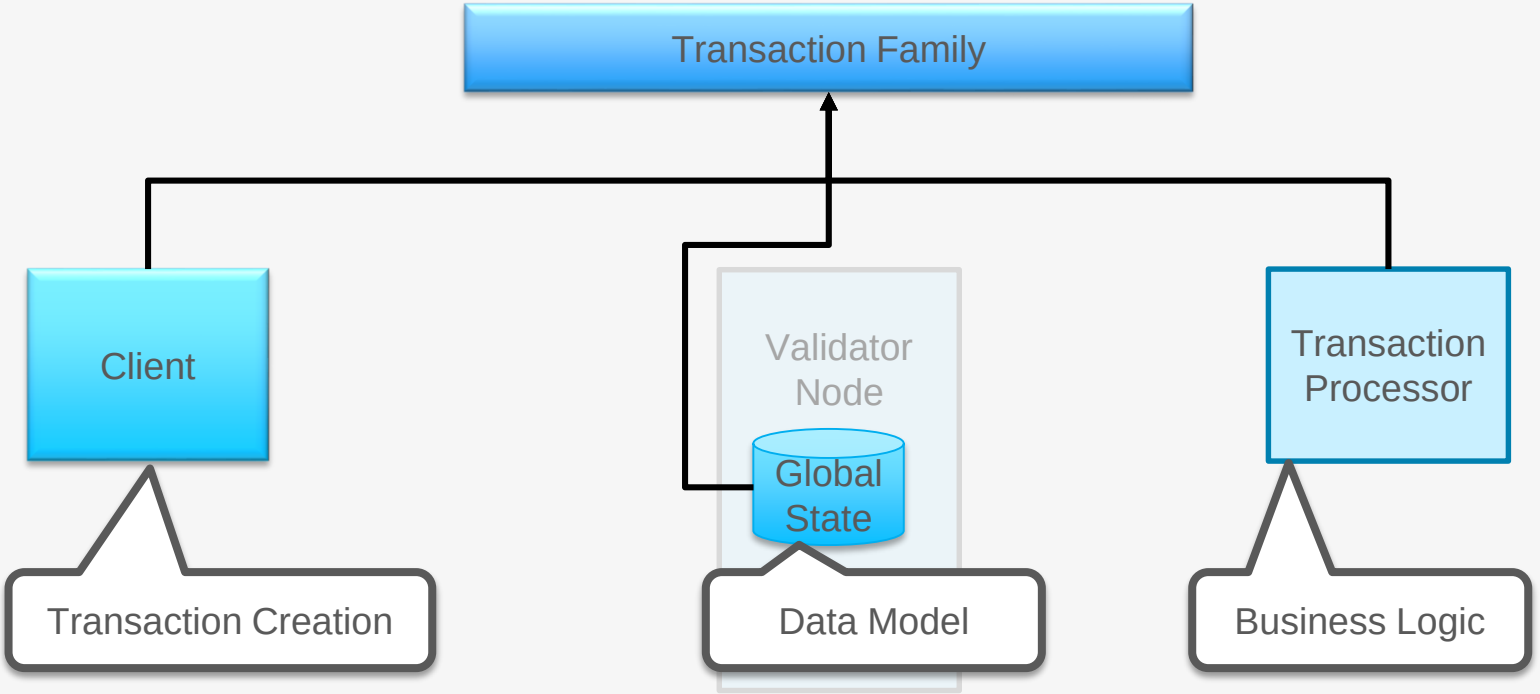
*Future consensus options

HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Hyperledger Sawtooth 1.0 Application Development

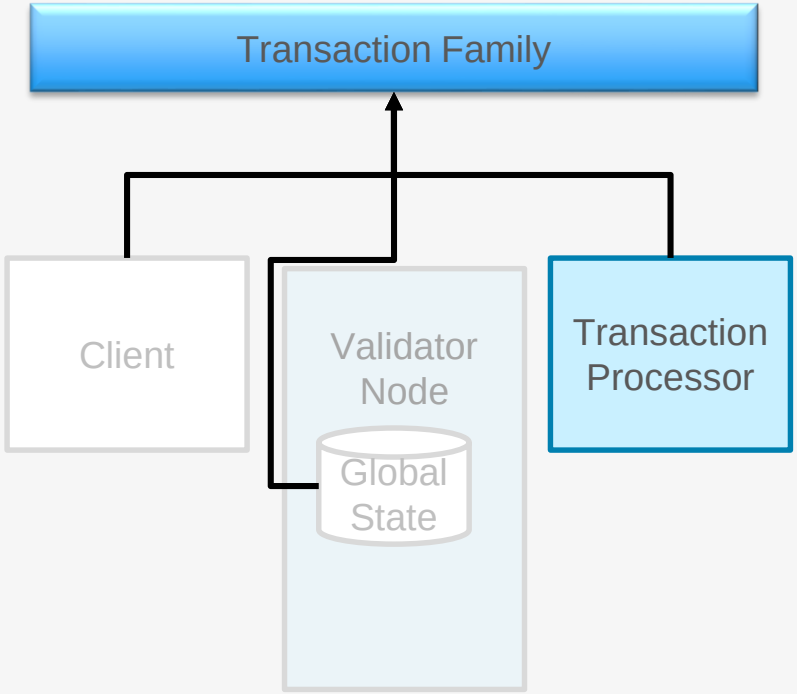
1.0 Released January 2018

Application Development: Transaction Family



Application is separated from complexities of blockchain management

Transaction Processor ≈ Smart Contracts



Transaction Families **encapsulate business logic** on Sawtooth.

A Transaction Family can be as simple as a single transaction format, with associated validity and state update logic…

…or as complex as a VM with opcode accounting and bytecode stored in state – 'smart contracts'.

The *choice* is up to the developer.

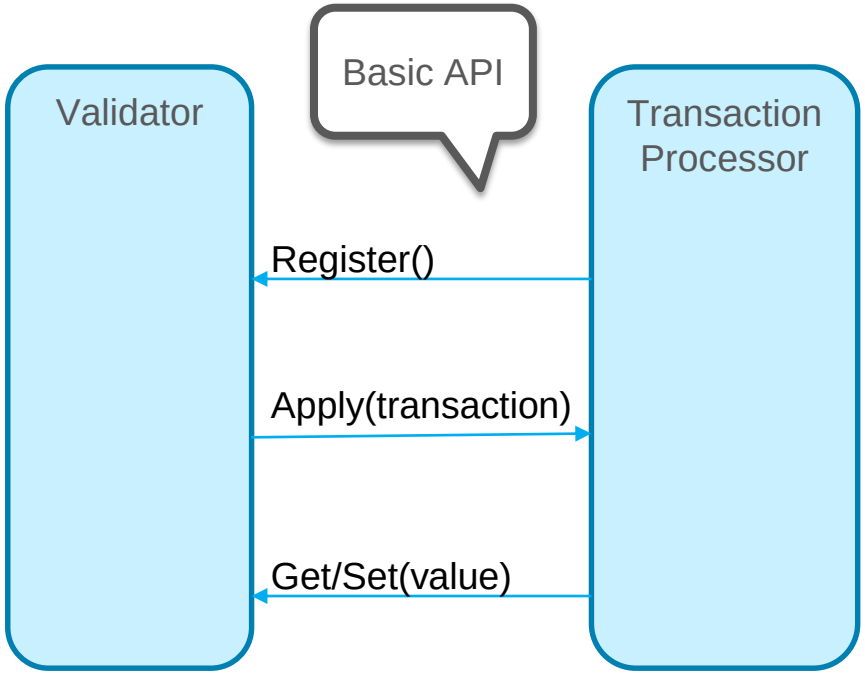
Sawtooth allows these concepts to **coexist** in the same instance of the blockchain – same blocks, same global state.

Transaction Families: The Transaction Processor

All validators in the network run every authorized transaction processor.

On receipt of a transaction the validator will call the TP's Apply() method.

Business logic written in Apply(), which calls Get()s and Set()s state as needed.



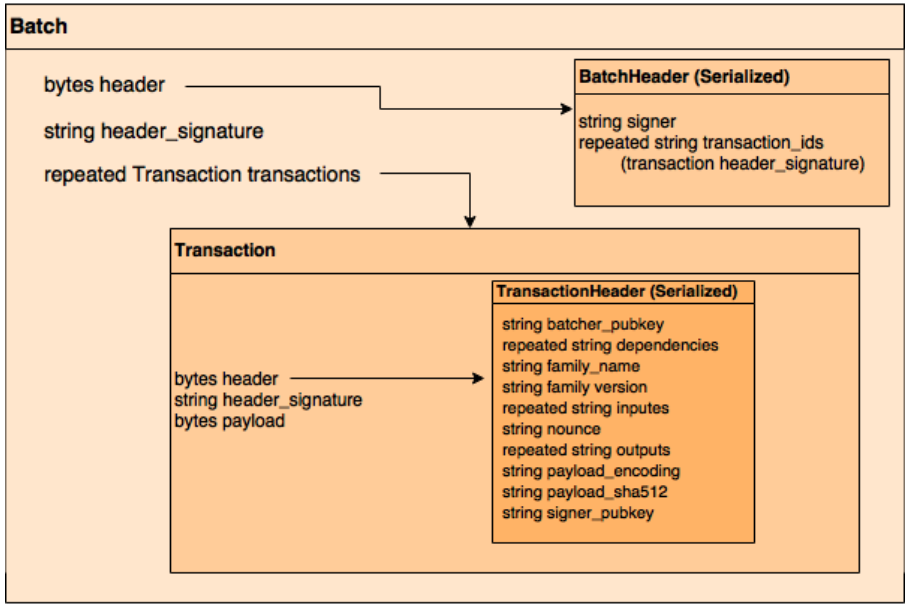
Transaction Families: The Client

Clients can be browser apps, CLIs, GUIs, BUIs, server, etc.

Client’s job is to package and sign transactions and batches

Clients post batches to node through REST API

Transactions, Batches, Batch Lists, and Blocks



Transactions are wrapped in batches which provide an *atomic* unit of commit for multiple transactions (which can span transaction families).

Atomic means process all or none of the transactions in a batch.

Batches are wrapped in batch lists.

Transactions *declare input and output addresses* (including wildcards) to allow for state access isolation calculations (topological sort on DAG) in the scheduler.

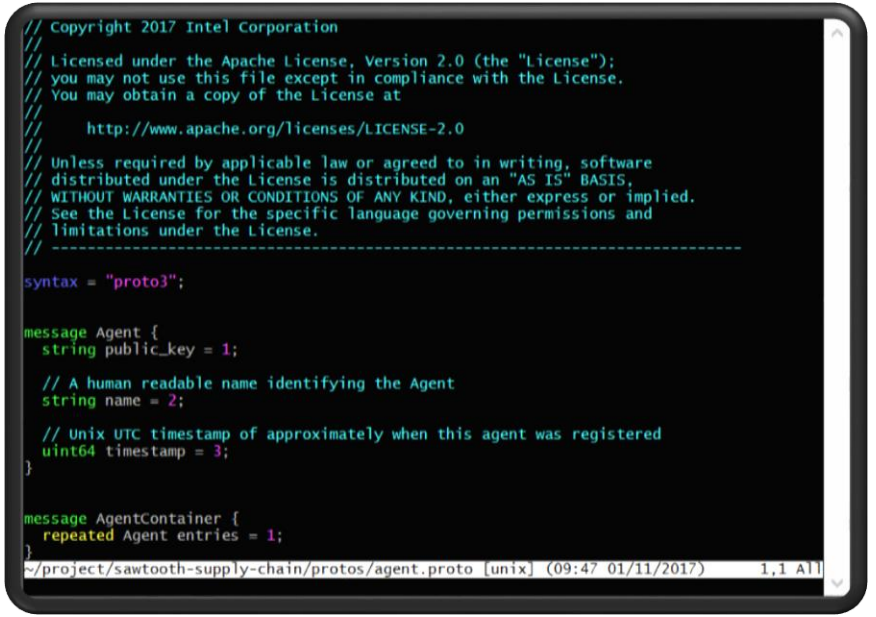
The inputs and outputs lists are enforced by the Context Manager on the context established for the transaction.

This allows parallel validation and execution across a potentially large number of transactions (and across blocks).

Transaction Families: The Data Model

Both Client and Transaction Processor must use the same…

- Data model (transaction and state)
- Serialization
  (CSV, CBOR, Protobuf, …)
- Addressing scheme into state
  (6 char prefix +
  64 char address)



```
// Copyright 2017 Intel Corporation
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// ------------------------------------------------------------------------------

syntax = "proto3";


message Agent {
  string public_key = 1;

  // A human readable name identifying the Agent
  string name = 2;

  // Unix UTC timestamp of approximately when this agent was registered
  uint64 timestamp = 3;
}


message AgentContainer {
  repeated Agent entries = 1;
}
~/project/sawtooth-supply-chain/protos/agent.proto [unix] (09:47 01/11/2017)      1,1 All
```

HYPERLEDGER SAWTOOTH

Example Application Code and Links

Simple Standalone Examples:
https://github.com/danintel/sawtooth-cookiejar
https://github.com/askmish/sawtooth-simplewallet
Supply Chain:
https://github.com/hyperledger/education-sawtooth-simple-supply
https://github.com/hyperledger/sawtooth-supply-chain
Marketplace:
https://github.com/hyperledger/sawtooth-marketplace

Application Developers Guide:
https://sawtooth.hyperledger.org/docs/core/releases/latest/app_developers_guide.html

FAQ: https://sawtooth.hyperledger.org/faq/

Check It Out!

Give Sawtooth a try

Work through the tutorials in the docs

Build your own transaction family to explore use cases

Become a contributor

Help with docs, code, answering chat questions

More Information

Code: https://github.com/hyperledger/sawtooth-core

Docs: https://sawtooth.hyperledger.org/docs/

Chat: https://chat.hyperledger.org/channel/sawtooth

Mailing List: https://lists.hyperledger.org/g/sawtooth/topics

FAQ: https://sawtooth.hyperledger.org/faq/

Thanks!