BLEEP : A general framework for implementing and testing versatile blockchains

Yonggon Kim

KAIST

KAIST 한국과학기술원

BLEEP

- **BLockchain Emulation and Evaluation Platform**
- 모든 종류의 블록체인을 동작시키고 일괄적으로 평가할 수 있는 플랫폼
- 현재 **BLEEP**의 목표
  - 제대로 동작하는 블록체인을 누구나 쉽게 구현해보고 동작시켜보고 발생하는 문제점을 빠르게 파악할 수 있도록 도와주는 플랫폼
  - ***기초적인*** 블록체인 알고리즘을 쉽게 구현하고 테스트할 수 있도록 하는 것이 목표

BLEEP's goal (in this course)

- 하지만 블록체인이 종류가 워낙 다양하다
  - PoW, PoS, DPoS, PBFT, …
  - Private blockchain, public blockchain, …
  - 2000 개가 넘는 암호화폐와 다양한 알고리즘들…
- 이 모든 블록체인에 관심있는 모든 사람들이 도움을 얻을 수 있는 플랫폼이 가능할까?
- 어느 정도의 수준으로 가능할지는 모르지만, 어쨌든 그것이 우리의 목표
  - 다양한 블록체인을 쉽게 구현할 수 있는 구현 템플릿 제공
  - 구현한 블록체인을 쉽게 테스트해볼 수 있는 평가 플랫폼 제공

BLEEP's goal in this course

- **목표 1. 블록체인 구현 템플릿 및 라이브러리 제공**
  - 모든 블록체인이 공통적으로 가지는 형태를 파악하여, 대다수 블록체인을 쉽게 구현해볼 수 있는 템플릿을 제공하는 것이 첫번째 목표
  - 또한 대다수 블록체인이 사용할 수 있는 공통 library 모듈을 제공하는 것이 목표
    - Data 모듈, 네트워크 통신, p2p 시스템, 등등

- **목표 2. 블록체인 테스트 플랫폼 제공**
  - 구현한 블록체인을 쉽게 테스트해보고 검증할 수 있는 방법을 제공하는 것이 두번째 목표
  - 여러가지 fault 를 생성해보고, 블록체인의 동작 결과를 검증
  - 벤치마크 형태로 테스트를 제공하며, 블록체인 구현체에 independent 한 자동화된 테스트가 가능하도록 테스트 프레임워크를 제공

BLEEP : Library for developing blockchain

- **Today's Topic**
  - BLEEP 목표 1. 블록체인 구현 템플릿 및 라이브러리 제공
- 좋은 라이브러리란?
  - 범용성 (generality)
  - 사용성 (easy-to-use)
- 대다수의 블록체인을 개발하는데 똑같이 적용하여 사용할 수 있는 범용성
- 또한 개발자가 **library** 를 쉽게 이해하고 쉽게 사용할 수 있어야 함

Abstraction for blockchain

- 범용성과 사용성을 가진 블록체인 **library** 를 제작하기 위해 어떤 방법을 사용해야 하는가**?** 무엇이 필요한가**?**
- **Solution key : A good abstraction for blockchain system**
- 블록체인의 좋은 추상화 모델이 존재한다면**,** 추상화 모델에 대한 라이브러리를 제공할 수 있다
  - 이를 통해, 대다수의 블록체인에 적용되는 라이브러리를 제공할 수 있다
- 좋은 추상화 모델은 디테일한 부분을 생략하며**,** 핵심 컴포넌트에 집중할 수 있게 도와준다
  - 블록체인에 대한 수많은 디테일한 부분을 전부 이해하지 않아도 핵심 컴포넌트에 대한 라이브러리를 이용해 블록체인 개발할 수 있게 된다
  - 핵심 컴포넌트에 집중함으로서 단순하고 사용하기 쉬운 라이브러리를 제공

BLEEP: 오늘의 주제

- 즉, 좋은 블록체인 구현 템플릿 및 라이브러리의 디자인을 위해서는 우선적으로 일반적인 블록체인 시스템을 추상화하여 이해할 필요성이 있다
- 블록체인 시스템을 어떻게 **formal** 하게 정의할 수 있을까**?**
- 대다수의 블록체인 시스템을 어떤 형태로 단순화하여 이해할 수 있을까**?**
- 그리고 추상화된 블록체인 모델을 기반으로 어떻게 범용적인 블록체인 개발 프레임워크를 제공할 수 있을까**?**

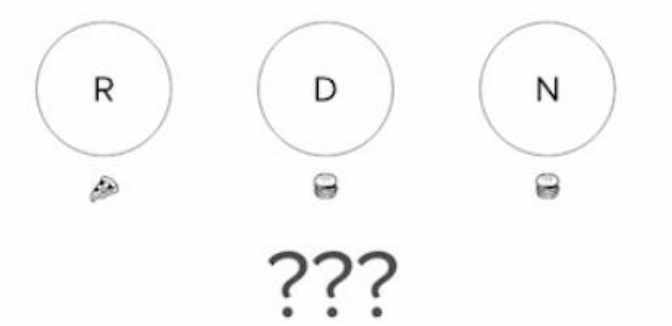
Outline

- **Background**
  - Distributed system approach
  - Formal system definition
- **Abstract model for blockchain system**
  - Replicated state machine model
  - Abstraction of recent blockchain paper
  - Our blockchain system abstraction model
- **BLEEP : Design & Implementation**
  - 구현 이슈
  - Overall design & components
  - Examples

Background : Distributed system

- **Blockchain system 은 우선 distributed system**
  - 다행히도, Distributed system에 대한 시스템 모델 및 formalization 은 오래 기간 연구되어 왔다

- **Simple analogy : 점심시간에 뭐먹을지 결정하는 문제 역시 넓은 의미의 distributed system**
  - Majority opinion 을 파악하고, agreement 를 이루는 문제



Background : Origin of distributed system

- **Origin of distributed system**
  - Super dependable computers pioneered by aircraft manufacturers

- 비행기 시스템은 매우 취약한 환경에서 동작함
  - 희박한 대기, 태양광 등에 의해 bit 가 튀는 현상
  - 만일 연료 시스템, 오토 파일럿 등이 잘못 동작하게 된다면?

- 이를 막기 위해, 여러 대의 컴퓨터를 동작시키고, 일부가 잘못 동작해도 제대로 된 상태를 유지하고 시스템을 정상 동작시킬 수 있는 **distributed system** 이 개발됨

Background : Blockchain and Distributed System

- 블록체인 역시 비슷하다
  - 네트워크가 이상하게 동작하거나, malicious 한 개체들이 블록체인의 동작에 영향을 주는 상황에서도 전체 시스템 동작을 안전하게 유지하는 것이 목적

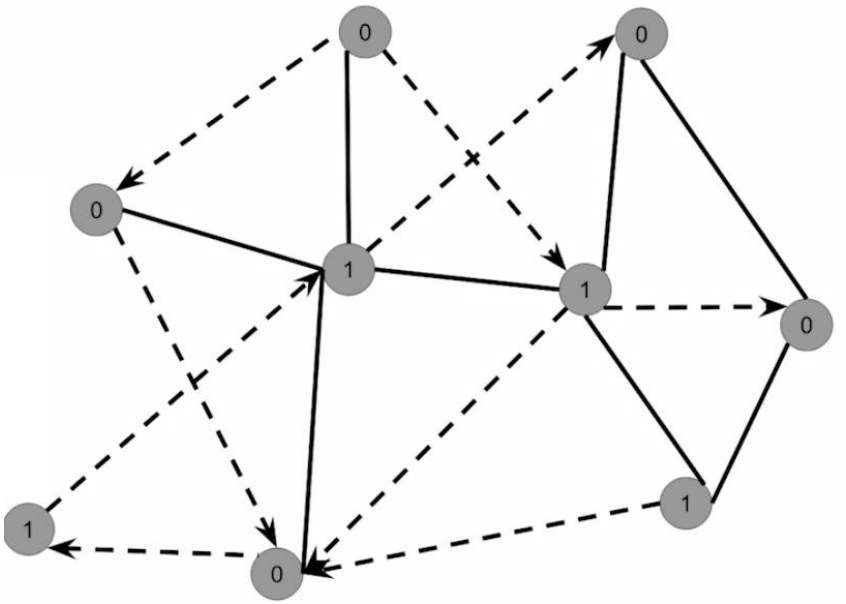
- **Distributed system : Trust without trust**
  - Distributed system 에 참여하는 개체들의 reliability 나 trust 에 의존하지 않고, protocol 및 기반하는 math 에 의존하여 시스템의 trust 를 유지하는 것

Background : Distributed System's formal definition

- **Distributed system 은 어떻게 정의할 수 있는가**
- **Distributed system 은 두가지 컴포넌트로 구성된다**
  - Nodes : 전체 시스템을 구성하는 머신들 (혹은 process 들)
  - message passing : 머신 간에 정보를 주고받을 수 있는 체계
- **Distributed system 의 세가지 중요한 특징**
  - Node 들이 Concurrent 하게 동작하는 시스템
  - 각각의 node 들은 제각각의 time 을 가지고 있음
  - 시스템 어디에서든 Failure 가 발생할 수 있음
- **또 어떤 것들이 필요한가? 일단 예제를 보자**

Background : Distributed system example

- **Binary consensus (classical distributed system example)**
  - 각각의 노드들은 0 혹은 1 두가지 input 만을 가진다
  - 전체 system의 output 역시 0 혹은 1 두가지를 가진다



- **Distributed system 의 목적 (및 요구사항) 은 어떻게 정의할 수 있나?**

Background : Definition of distributed system

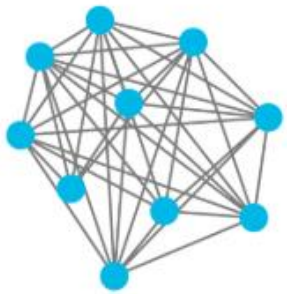
- **Distributed system** 은 여러 **node** 와 **node** 간의 **message passing** 으로 구성되며, 결국 **distributed system** 은 주어진 **input** 에 대해 노드들이 합의를 하여 공통된 **output** 을 내놓는 시스템

- 각 노드는 서로 다른 **input** 을 받을 수 있다
- 노드들이 합의를 통해 하나의 올바른 **output** 을 생성한다
- 합의하는 과정을 우리는 일반적으로 **consensus algorithm** 이라고 얘기한다

Background : Distributed system example

- **(distributed) blockchain system**
  - 각각의 node 는 외부로부터 input 을 받는다.
  - 네트워크를 구성하는 모든 node 가 서로 통신을 통해 correct 한 output 을 제공하는 것이 blockchain system 의 목적
  - 이 과정에서 다양한 failure 가 존재함에도, blockchain system 이 제대로 동작할 수 있어야 한다.

- **distributed system 이 제대로 동작한다는 것은 무엇일까?**
  - 제대로 동작한다는 것은 무엇일까?
  - 시스템이 내놓는 output 은 언제 correct 하다고 얘기할 수 있는가?
  - 어떤 property 들을 만족시켜야 제대로 된 distributed system 인가?



- **Distributed system 의 correctness 에 대한 formal definition?**

Background : Safety & liveness

- **Leslie Lamport (legendary distributed systems researcher)**
  - A system is correct if two things are true:
  - it doesn't do bad things
  - And it eventually does good things

- **More formally**
  - it doesn't do bad things : **safety**
  - it eventually does good things : **liveness**

Background : Safety & liveness for distributed system

- Consensus algorithms in the context of safety and liveness
- What must happen, and what cannot happen, in order for a system to come to consensus?

- The three requirements of any correct consensus algorithm

Agreement: all non-faulty processes must agree on the same value

Validity: any value decided upon must be proposed by one of the processes

Termination: all non-faulty nodes eventually decide

Background : Terminology for formal blockchain system

- **Node : system** 을 구성하는 단위
- **Message passing : node** 들 간의 정보 교환

- **Input :** 각 **node** 에 주어지는 값들
- **Output :** 모든 **node** 가 합의하는 결과 값들
- **Consensus algorithm : input** 을 **output** 으로 합의하는 과정

- **Correctness (based on safety & liveness)**
  - Agreement : 모든 node 가 같은 output 을 가지는가
  - Validity : 유효한 input 으로 들어온 값으로 output 을 만들어 냈는가
  - Termination : output 을 eventually 만들어낼 수 있는가

ABSTRACT MODEL FOR BLOCKCHAIN SYSTEM

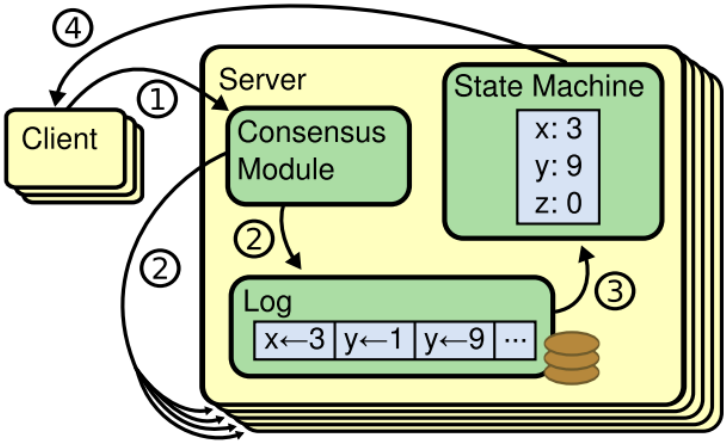
Abstract Model for Blockchain System

- **Based on distributed system formalization…**
  - What is the proper model (abstraction) for blockchain system?
- **Replicated State Machine Model**
  - Classical distributed system model
- **Abstractions of recent blockchain paper**
- **Our suggestion of blockchain system model in BLEEP**

Raft & Replicated state machine model

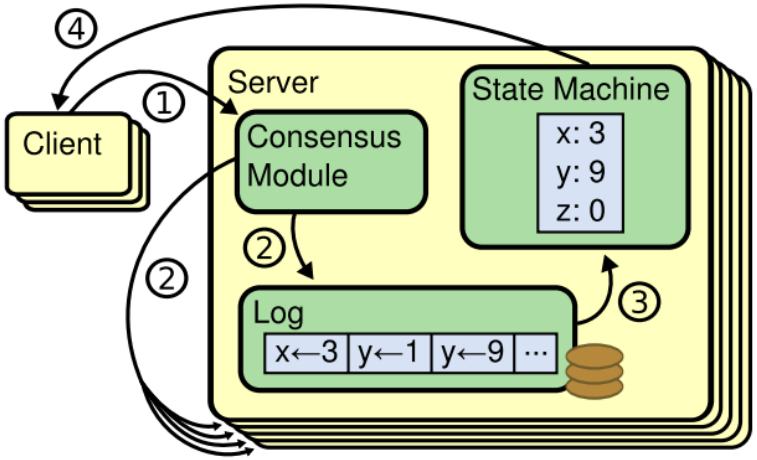
- **Raft (Usenix ATC 2014, Best paper award)**

- Raft and Replicated state machine
  - 기존 분산 시스템에서 많이 사용하는 모델
  - 각각의 서버는 동일한 state 를 가지며, 몇몇 서버가 죽거나 해도 계속해서 시스템이 valid 한 state 를 유지하며 동작할 수 있음
  - Fault 가 발생했을 때 안전하게 data 를 관리하는 문제를 풀기위해 사용



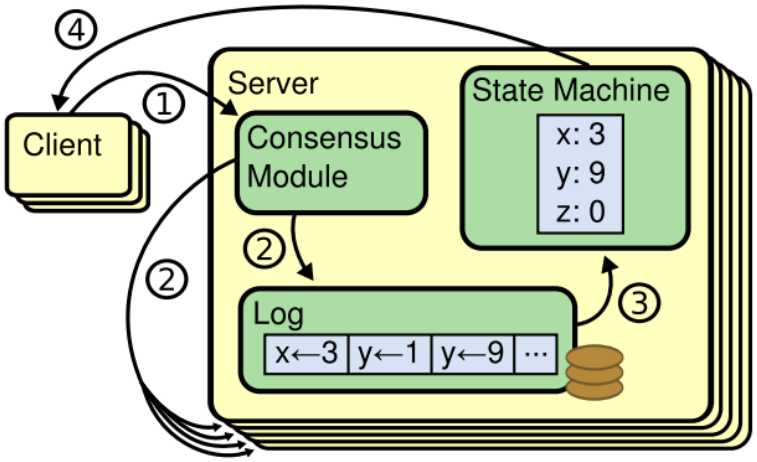
Replicated state machine model

- 분산 시스템에 들어온 로그를 input으로 생각하며, 이에 대한 합의를 통해 똑같이 복제된 log 를 만들고, 각각의 서버는 이렇게 복제된 log 를 이용해 동일한 state machine 을 동작시키는 방식
- (1) 각 서버는 서버로 들어온 명령을 받고,
  (2) 합의하여 순서대로 로그로 남기고,
  (3) 로그에 따라서 State machine을 수행하여,
  (4) State machine 수행 결과를 client에게 output으로 보내주게 됨
- (2)에서 나온 합의라는 것이 바로 모든 서버에 대해서 로그를 일관성있게 유지한다는 것이며, 합의 알고리즘(Consensus algorithm)이 여기서 로그를 일관적으로 유지하는 알고리즘으로 사용됨.



Replicated state machine model & blockchain

- 하지만 이 모델은 일반적인 블록체인 모델에 적용하기는 어렵다
- 블록체인을 구성하는 모든 **node** 에 들어오는 **input** 에 대해 **consensus** 가 일어나야 함
  - 즉, Scalability 가 매우 떨어짐

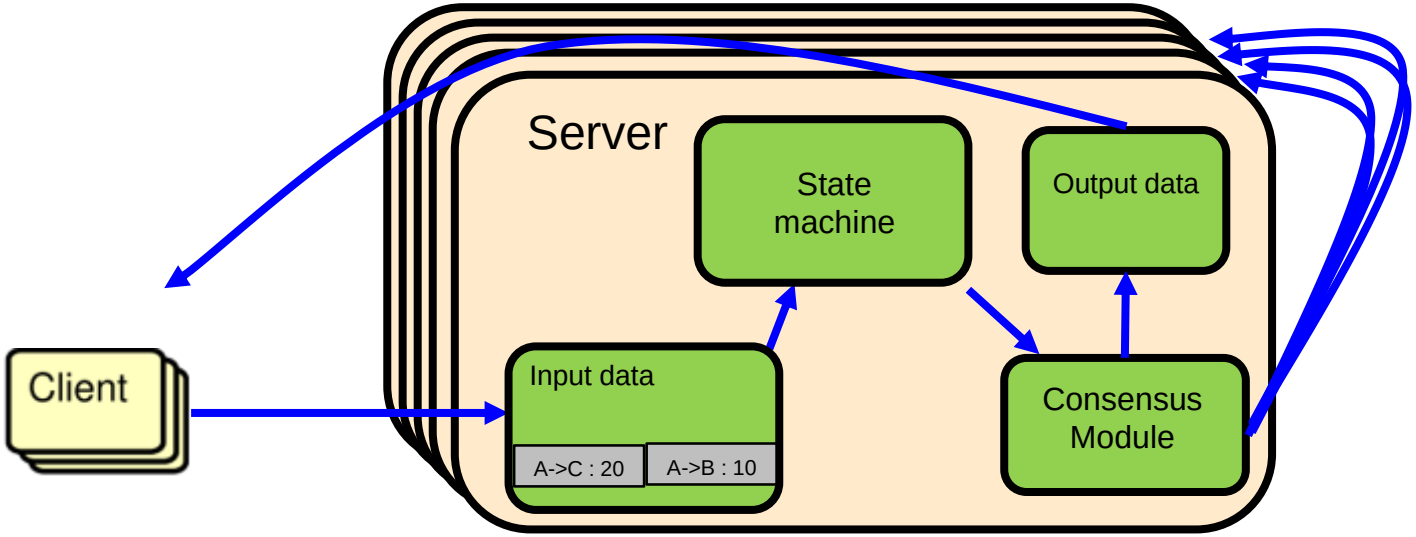


Recent abstractions of blockchain system

- **Not many …**
  - Usually focus on specific portion of the blockchain (smart contract, economy, crypto)
- **Hawk [S&P 16]**
- **Abstract abstract abstract model of blockchain**

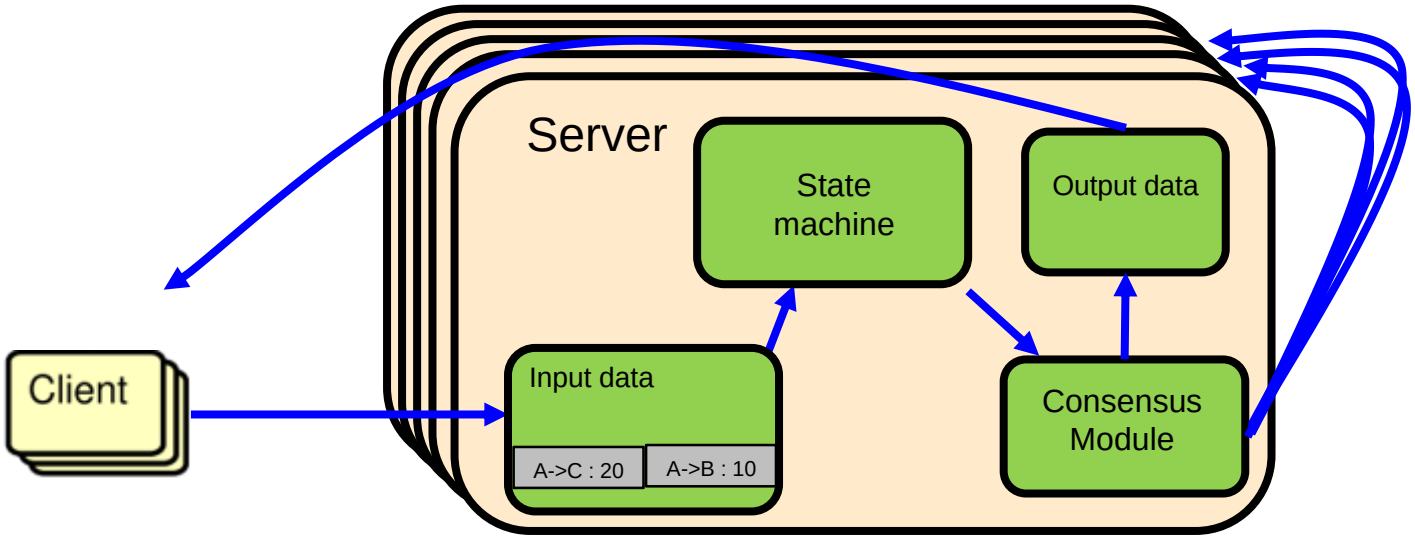
- **Blockchain 은 data 와 computation 에 대한 consensus 를 통해, up-to-date state**를 유지한다
- 블록체인은 **input** 에 대한 **consensus** 가 아니라, **data** 및 **computation** 에 대한 **consensus** 를 목표로 함
  - 하지만 너무 광범위한 정의

BLEEP's blockchain system model



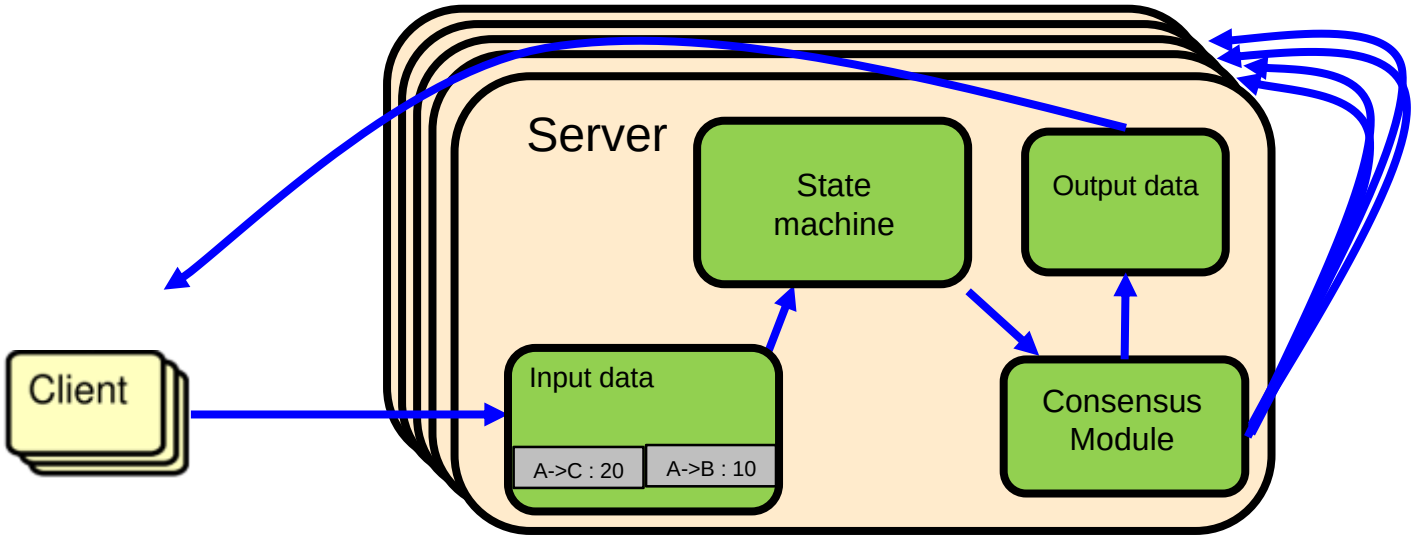
BLEEP's blockchain system model

- **RSM 모델과의 차이**
  - Input 이 들어오면, 이에 대해 합의를 진행하는 것이 아니라 State machine 에 전달하게 됨
  - 각 node 들은 독립적으로 State machine 을 동작시키며, 각기 독립적인 상태를 가짐
  - Consensus algorithm 은 input 이 아닌, state machine 의 중간 값에 대해 이루어짐



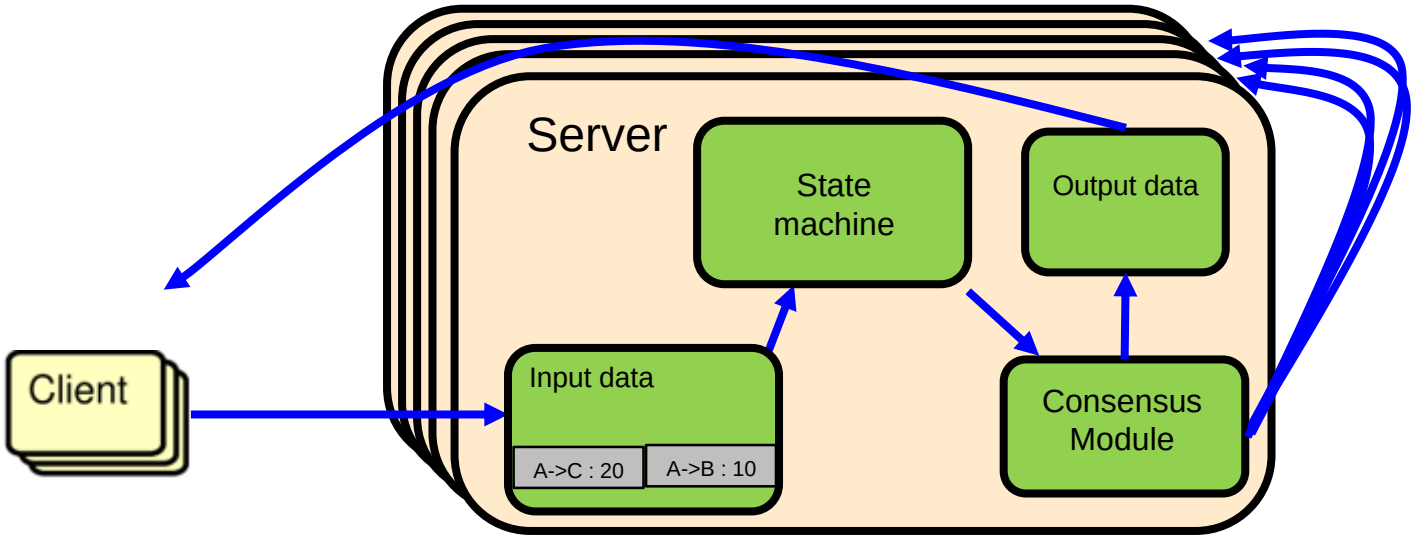
BLEEP's blockchain system model

- 중간값은 무엇인가?
  - Set of input
- **Consensus** 결과는 무엇인가?
  - 역시 set of input 이며, 이를 block 이라고 정의할 수 있다.
- **Output**
  - Set of block. 즉 blockchain



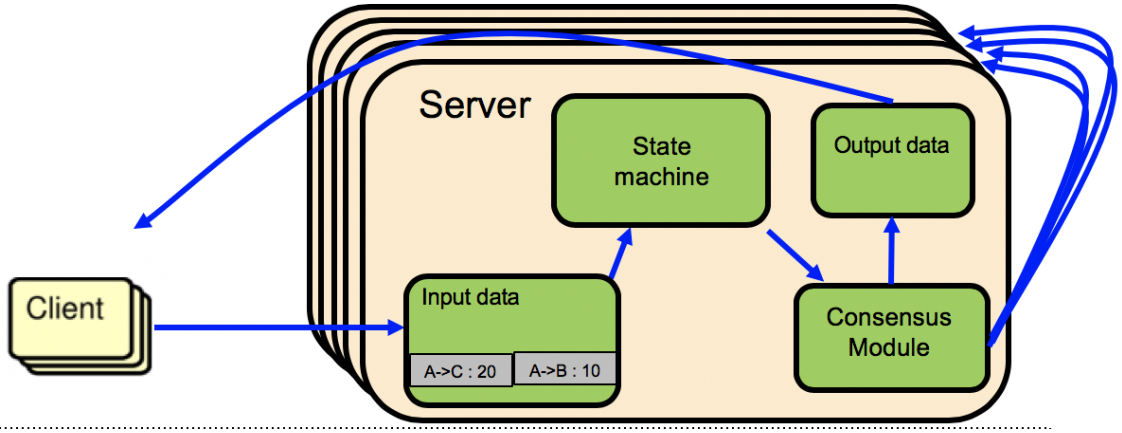
BLEEP's blockchain system model

- 예를 들어 비트코인은 **transaction** 을 **input** 으로 받아서 각 **node** 별로 **candidate block** 을 만든 이후에, 각각의 **block** 에 대해 **consensus** 를 요청하게 됨 **(**즉, **block** 에 대한 **mining** 작업 요청**)**
- 이 중에서, **mining** 에 성공한 **block** 은 모든 노드에 전파되며, **output data** 에 포함되게 됨



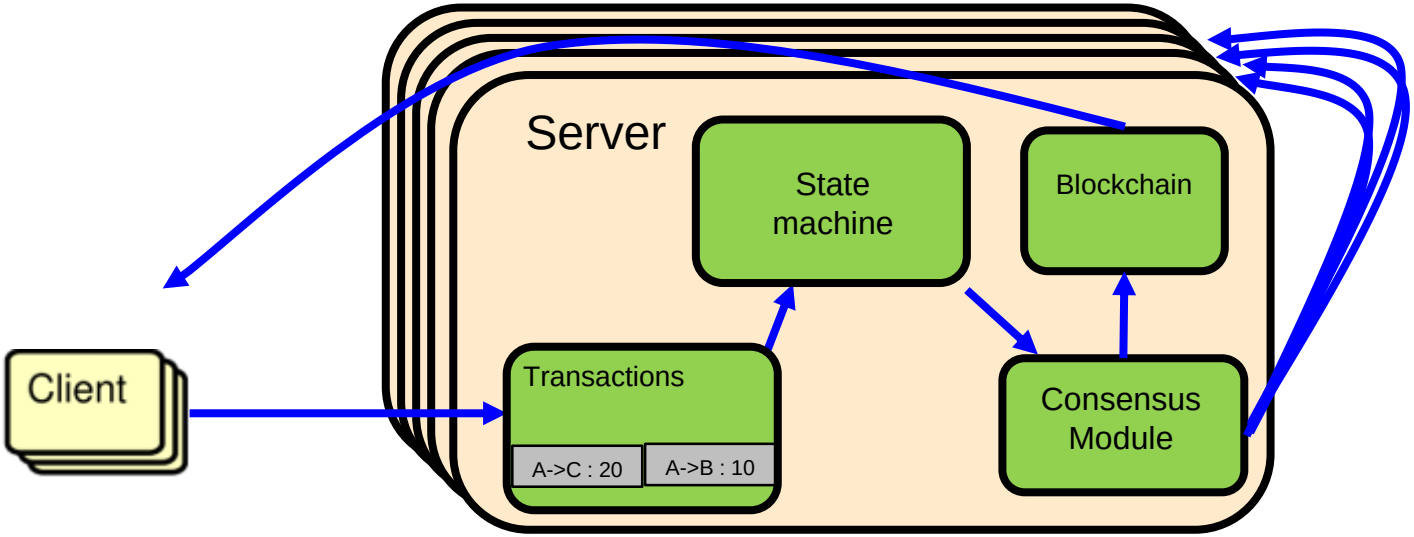
BLEEP's Blockchain system model

- **consensus** 의 결과로 **produce** 되는 **output** 이 최종적인 합의 결과가 아닐 수 있다
  - Ex) longest-chain policy
  - 예를 들어 비트코인 같은 경우, consensus 의 결과로 만들어진 valid block 에 대해 확실한 consensus 가 이루어졌다고 결정하는 finalize 가 없으며, 전체 네트워크에서 longest chain 이 바뀐다면, valid block은 버려질 수 있다
  - 반면 RSM 모델에서는, consensus 결과가 나오는 fix 되며, 따라서 state machine 을 거쳐 나오는 output도 deterministic 하다

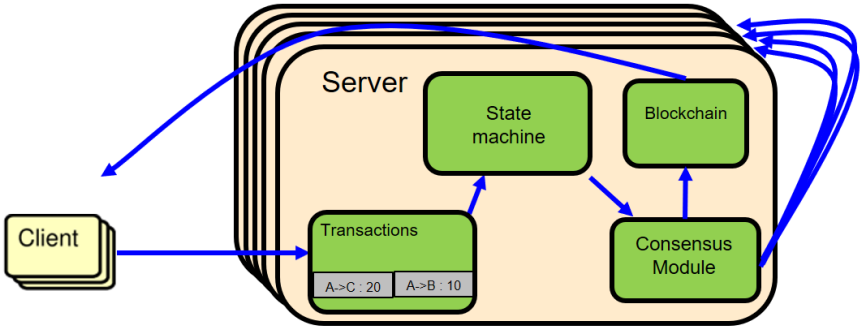


Blockchain system model : input & output

- 블록체인 시스템에 대한 **Input (즉, log),** 그리고 **output (output data)**에 대한 **format** 역시 블록체인에서 쓰이는 일반적인 **data structure** 인 **transaction** 과 **blockchain (ledger)** 으로 정의할 수 있음
  - Input : Transaction
  - Output : Blockchain



Definition : Blockchain system model



- **Client 로부터 transaction 들을 받고, 각 node 에서 독립적인 state machine 들이 동작하면서 중간 값들에 대해 consensus 를 진행하고, 합의의 결과로 blockchain 을 생성하는 구조**
- **Definition of correctness of blockchain system**
  - 네트워크를 구성하는 모든 node 는 동일한 blockchain 을 가진다 (agreement)
  - Blockchain 은 각 node 가 받은 transaction 중에서 선택된다 (validity)
  - Blockchain 을 만드는 과정에서 다양한 failure 가 존재해도, eventually 생성할 수 있어야 한다 (termination)

BLEEP : DESIGN & IMPLEMENTATION

Recap : Abstraction for blockchain

- 범용성과 사용성을 가진 블록체인 **library (**개발 플랫폼**)**를 제작하기 위해 어떤 방법을 사용해야 하는가**?** 무엇이 필요한가**?**
- **Solution key : A good abstraction for blockchain system**
- 블록체인의 좋은 추상화 모델이 존재한다면, 추상화 모델에 대한 라이브러리를 제공할 수 있다
  - 이를 통해, 대다수의 블록체인에 적용되는 라이브러리를 제공할 수 있다
- 좋은 추상화 모델은 디테일한 부분을 생략하며, 핵심 컴포넌트에 집중할 수 있게 도와준다
  - 블록체인에 대한 수많은 디테일한 부분을 전부 이해하지 않아도 핵심 컴포넌트에 대한 라이브러리를 이용해 블록체인 개발할 수 있게 된다
  - 핵심 컴포넌트에 집중함으로서 단순하고 사용하기 쉬운 라이브러리를 제공

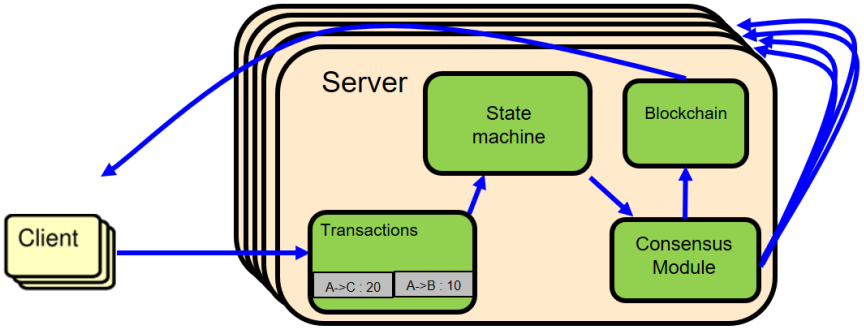
Recap Background : Terminology for formal blockchain system

- **Node : system** 을 구성하는 단위
- **Message passing : node** 들 간의 정보 교환

- **Input :** 각 **node** 에 주어지는 값들
- **Output :** 모든 **node** 가 합의하는 결과 값들
- **Consensus algorithm : input** 을 **output** 으로 합의하는 과정

- **Correctness (based on safety & liveness)**
  - Agreement : 모든 node 가 같은 output 을 가지는가
  - Validity : 유효한 input 으로 들어온 값으로 output 을 만들어 냈는가
  - Termination : output 을 eventually 만들어낼 수 있는가

Recap : Blockchain system model definition



- **Client 로부터 transaction 들을 받고, 각 node 에서 독립적인 state machine 들이 동작하면서 중간 값들에 대해 consensus 를 진행하고, 합의의 결과로 blockchain 을 생성하는 구조**
- **Definition of correctness of blockchain system**
  - 네트워크를 구성하는 모든 node 는 동일한 blockchain 을 가진다 (agreement)
  - Blockchain 은 각 node 가 받은 transaction 중에서 선택된다 (validity)
  - Blockchain 을 만드는 과정에서 다양한 failure 가 존재해도, eventually 생성할 수 있어야 한다 (termination)

BLEEP : Implementation challenges

- 좋은 라이브러리란?
  - 범용성 (generality), 사용성 (easy-to-use)
  - 이를 위한 추상화된 블록체인 모델
- 앞서 제시한 모델은 **conceptual** 한 블록체인 모델로서, 실제 구현에는 또 다른 이슈가 존재
- **Okay,** 구현을 위해 또 어떤 부분들이 중요할까?
  - 모듈화, 유지보수, 성능, 에러 핸들링, 테스팅 등등..
- 게다가 블록체인 특성상 라이브러리 개발에 있어 어려운 난제들이 존재함
- 모델에선 단순화 시켰으나 사실 블록체인은 수많은 컴포넌트로 구성되어 있음
  - P2P, cryptography, consensus, socket, transaction management, blockchain management, etc
  - 각각의 컴포넌트가 상호작용하여 동작하게 됨
  - 만일 특정 컴포넌트를 업데이트 해야할 때, 다른 모든 컴포넌트가 동작에 영향을 받는다면?

BLEEP : Implementation challenges

- 블록체인에 대한 공통의 요구사항이 명확히 정해지지 않았다
  - 수많은 컴포넌트로 이루어지고, 컴포넌트에 대한 요구사항이 블록체인마다 제각각
  - 2천개에 이르는 암호화폐의 모든 요구사항을 분석하고 정리해야하나? 현실적으로 불가능
  - (수요일에 다루겠지만, 그래서 BLEEP이 다루고자 하는 핵심 요구사항은 테스트)

- 블록체인은 계속해서 발전하고 있으며, 블록체인에 대한 요구사항 자체가 지속적으로 변화하고 있음
  - ex) PoW -> PoS -> dPoS -> ??
  - ex) EoS 의 bootstrapping 이슈 -> 새로운 요구사항

BLEEP : Library Requirements

- 이처럼 블록체인의 특징에서 비롯된 핵심 이슈들을 잘 처리해야함
  - 많은 컴포넌트
  - Requirement 의 지속적인 변화

- 블록체인 라이브러리가 가져야 하는 **requirements**
  - Component 독립성
  - Easy-to-update
  - Backward-compatibility

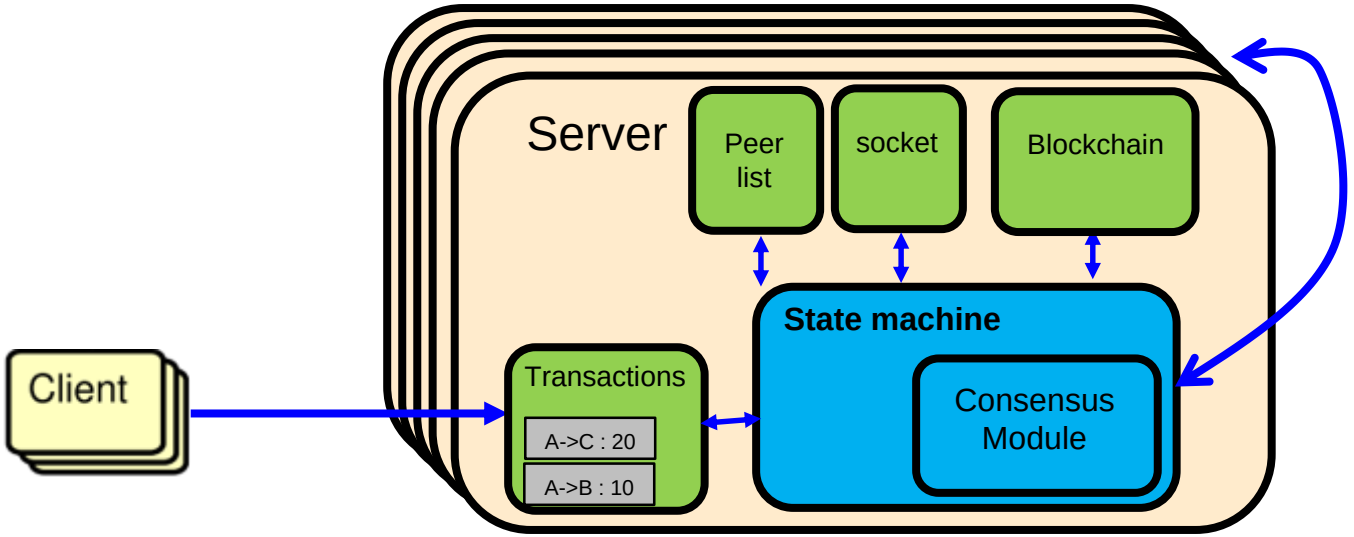
- 컴포넌트의 독립성
  - 각각의 컴포넌트들이 서로 독립적으로 개발되고 동작할 수 있도록 하여, 수많은 블록체인 컴포넌트 간의 불필요한 상호작용을 줄이고 독립성을 유지할 수 있어야함
- **Easy-to-update, backward-compatibility**
  - 새로운 요소, 기능을 추가하기 위한 라이브러리 업데이트가 쉬워야 함
  - 계속해서 변화하는 블록체인을 지원하는 동시에, 이전에 개발된 모든 블록체인들을 함께 지원할 수 있어야 함

BLEEP : Design principles

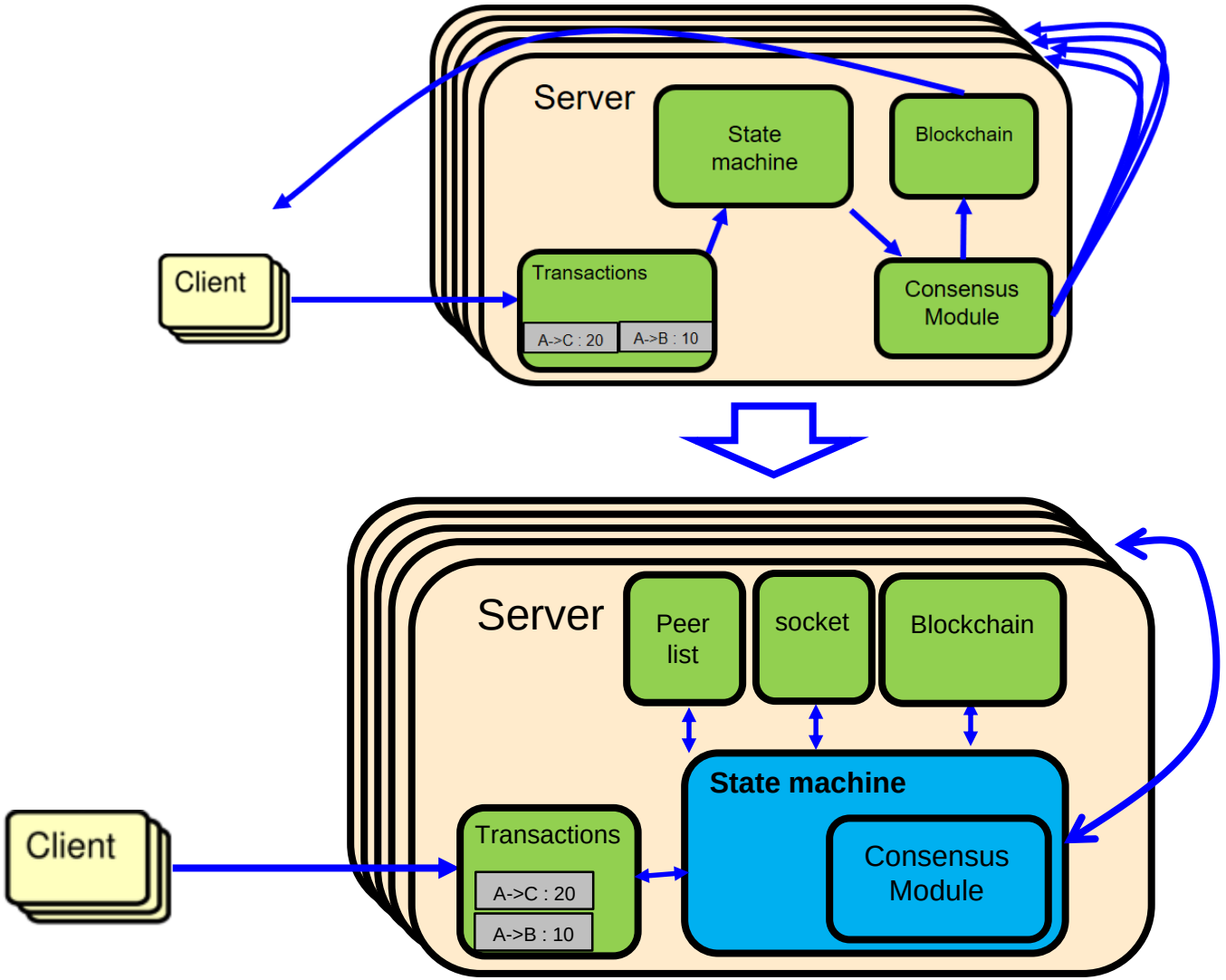
- **Focus on main functionality**
  - 현실적으로, 블록체인의 수많은 이슈, 요구사항들을 모두 다루는 것은 불가능
  - Our main focus is “Universal tests for blockchains” (수요일), especially for consensus
- **Suggestion of (independent, backward-compatible) modules for blockchain data**
  - 블록체인의 다양한 기능과 알고리즘에 비해, 사용하는 핵심 data 구조는 비교적 정형화 될 수 있기 때문에, library component 로 구현 가능함
- **Flexible (easy-to-update) template for developing versatile blockchain algorithms**
  - 수많은 블록체인 알고리즘 (p2p, consensus, tx 등) 들을 유연하게 구현할 수 있는 템플릿 및 예제 제공
- 유지보수가 가능하면서도, 대다수의 블록체인을 쉽게 구현하고 테스트할 수 있는 범용적인 플랫폼

BLEEP : Overall design

- **Blockchain Library & Template**
  - Library : Data 및 management module
    - data 의 저장 및 관리 담당
  - Template : State machine
    - state machine 의 동작, consensus algorithm 의 로직 구현 담당
- **Consensus** 알고리즘을 포함하는 **State machine** 과 이 **state machine** 이 참조하는 **data** 모듈들로 구성

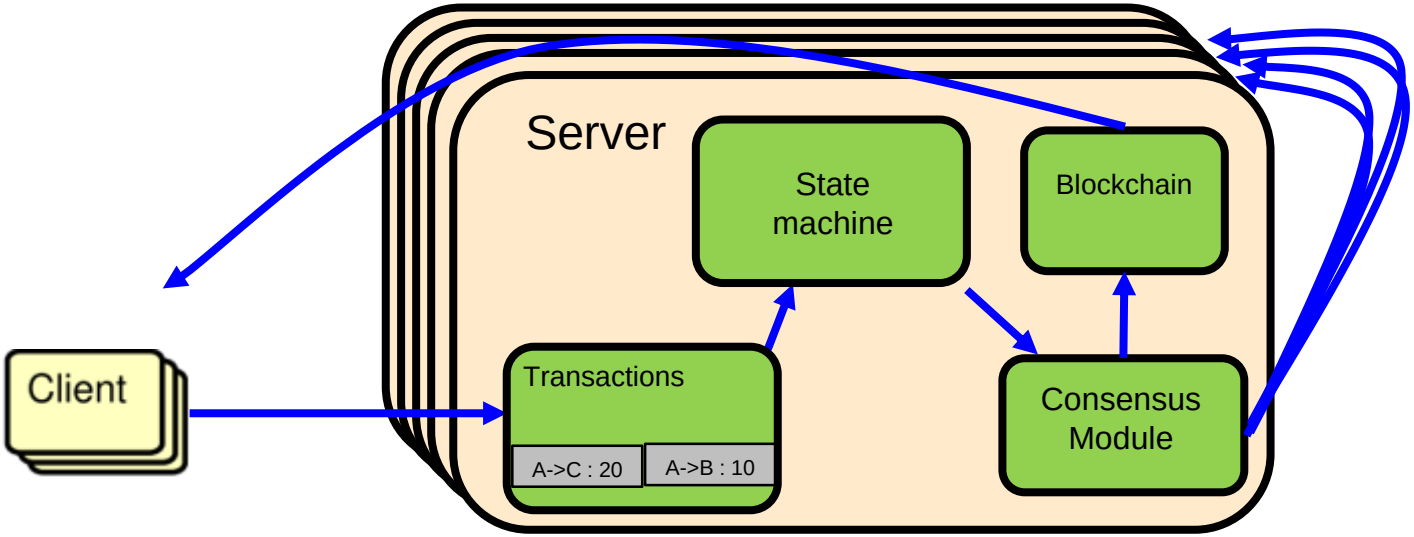


BLEEP : Overall design (why?)



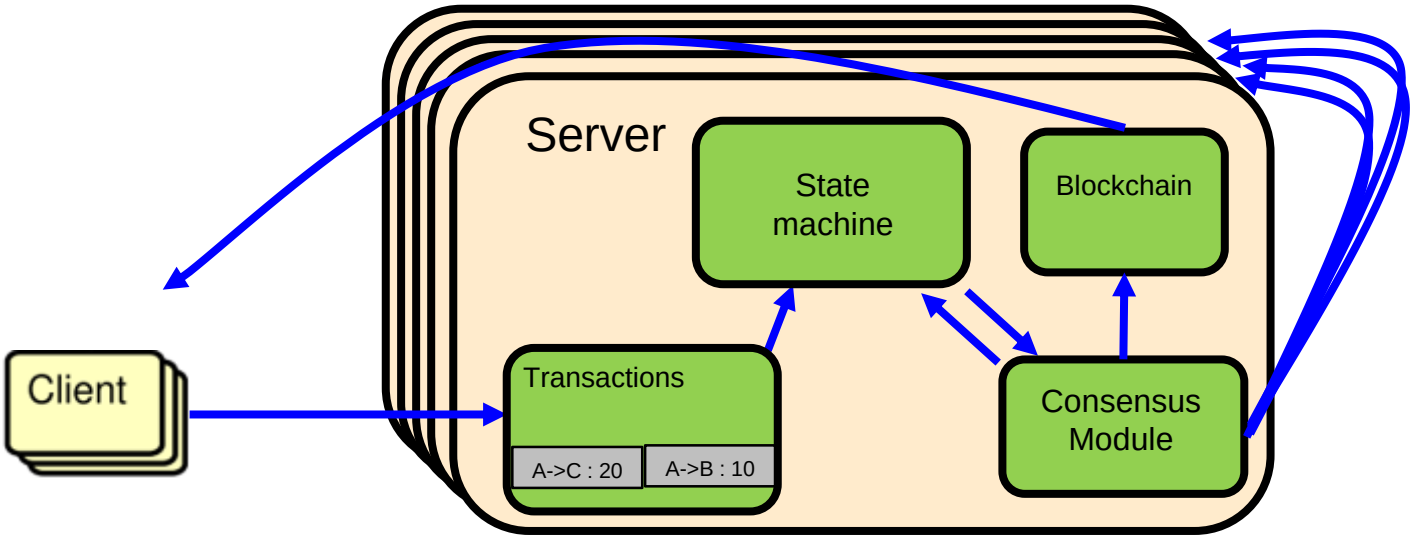
Abstract blockchain system model

- **Client** 로부터 **input** 들을 받고, 각 **node** 에서 독립적인 **state machine** 들이 동작하면서 중간 값들에 대해 **consensus** 를 진행하고, 합의의 결과로 **output** 을 생성하는 구조
  - Input : Transaction
  - Output : Blockchain



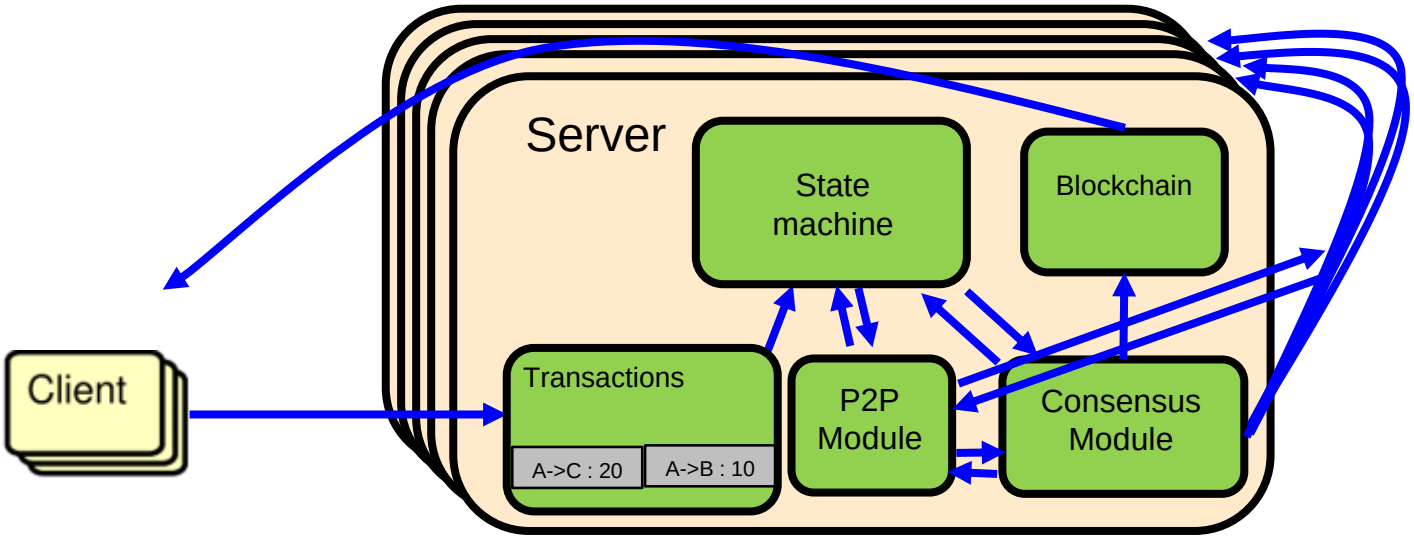
More realistic blockchain system model

- 이해를 위해, 단순화된 추상화된 모델을 설명했지만. **But,** 엄밀하게 따지자면, 실제 블록체인들이 동작하는 방식을 반영하기 위해서는 추가적으로 고려해야하는 부분이 존재
- **State machine** 은 **consensus module** 에게 중간 값을 전달할 뿐, **consensus** 의 결과를 반영하지 못하는가**?**
  - 아니다, 비트코인의 예를 들어보자.
  - 비트코인은 state machine 이 충분한 input 을 넘겨 받아 candidate block 을 생성하게 되더라도, 만일 이미 이전에 다른 candidate block 을 consensus module 에게 넘겨 준 상태라면, 또다시 새로운 마이닝(consensus)을 요구하지 않는게 일반적이다
  - 하지만, 다른 노드로부터 valid 한 block 을 받게 된다면? 그렇다면, 이제 다시 mining 을 요구할 수 있는 상황이 된 것이고, 두개의 candidate block 에서 어떤 block 에 대해 consensus 를 동작시킬지 결정해야하고, 혹은 둘 다 아니고 새로운 candidtate block 을 생성해야 할수도 있음



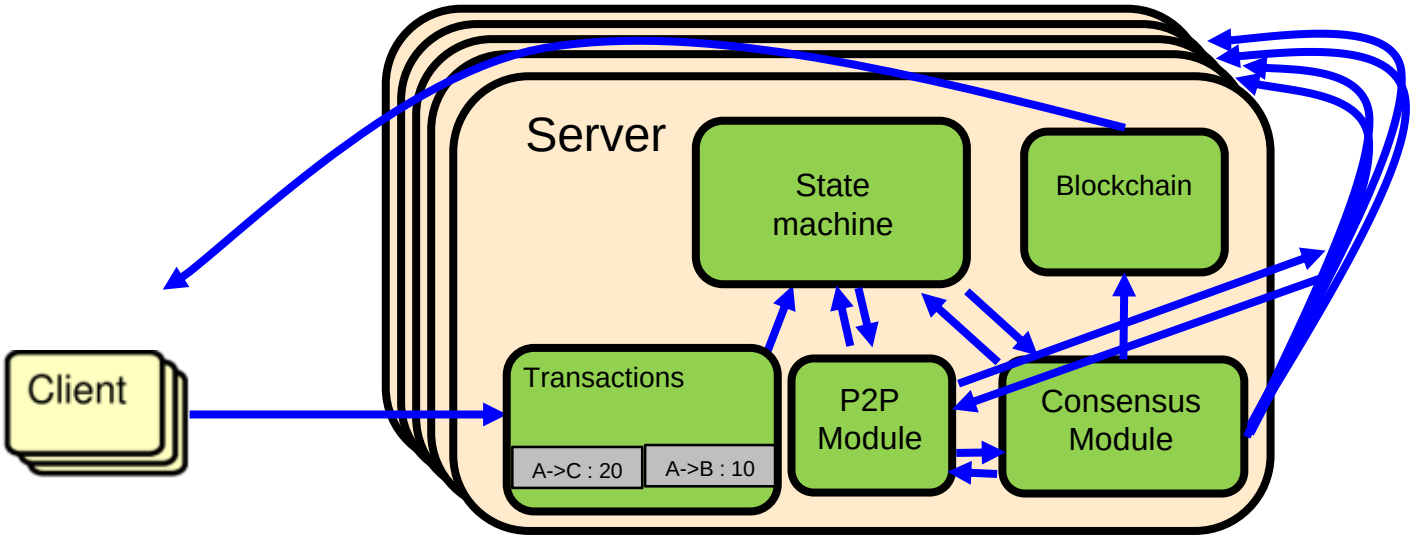
Implementing blockchain system model

- **Consensus module 만이 다른 node 들과 커뮤니케이션을 하게 되는가?**
  - 아니다, public 블록체인이 동작하는 large-scale P2P 시스템을 생각해보자
  - Large-scale p2p 환경에서 message passing 을 구현하기 위해서는 peer management, gossiping algorithm 등의 개발이 필요하며, 이들 알고리즘은 자체적으로 효과적인 broadcasting 을 위해 별도의 data structure (gossip tree)를 유지하며, 이를 위한 별도의 커뮤니케이션을 하게 된다
  - P2P module 의 구현이 필요하며, 이를 위한 message passing 도 구현해주어야 함
  - 또한, consensus module 은 필요에 따라 P2P (broadcast) module 을 사용하게 됨



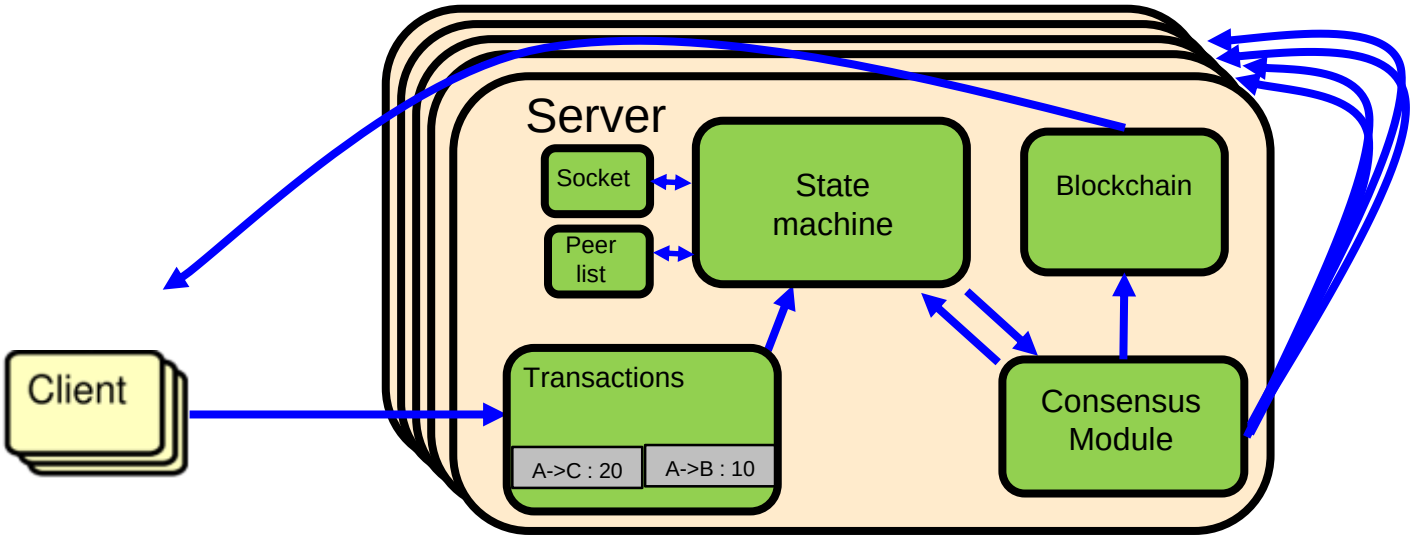
Implementing blockchain system model

- 하지만, 일단 당장은 **P2P module** 은 현재 **BLEEP library module** 이 아니다
  - 이유 1. P2P module is Not necessary component for blockchain
    - PBFT 와 같이, private 한 소규모 환경을 기반으로 하는 blockchain 은 별도의 gossip 알고리즘이 필요하지 않음
  - 이유 2. 구현 및 디버깅의 어려움
    - P2P algorithm 은 환경에 따라 다양하게 존재할 수 있음 (정형화 되어 있지 않음)
    - Consensus module 과 상호작용하기 위한 인터페이스를 디자인해야하며, P2P 모듈 자체적으로도 message passing 이 가능해야함. 즉, 인터페이스 및 채널이 늘어남에 따라 유지보수와 디버깅의 이슈가 있음
  - 이유 3. 성능 상의 이슈
    - 간단한 gossip protocol (plumtree) 를 구현해보았으나, gossip data structure 의 update를 위해 주고받은 메시지 때문에 실험 상 성능 저하가 꽤 존재하며, large scale 네트워크를 테스트할 때 성능 이슈가 존재



Implementing blockchain system model

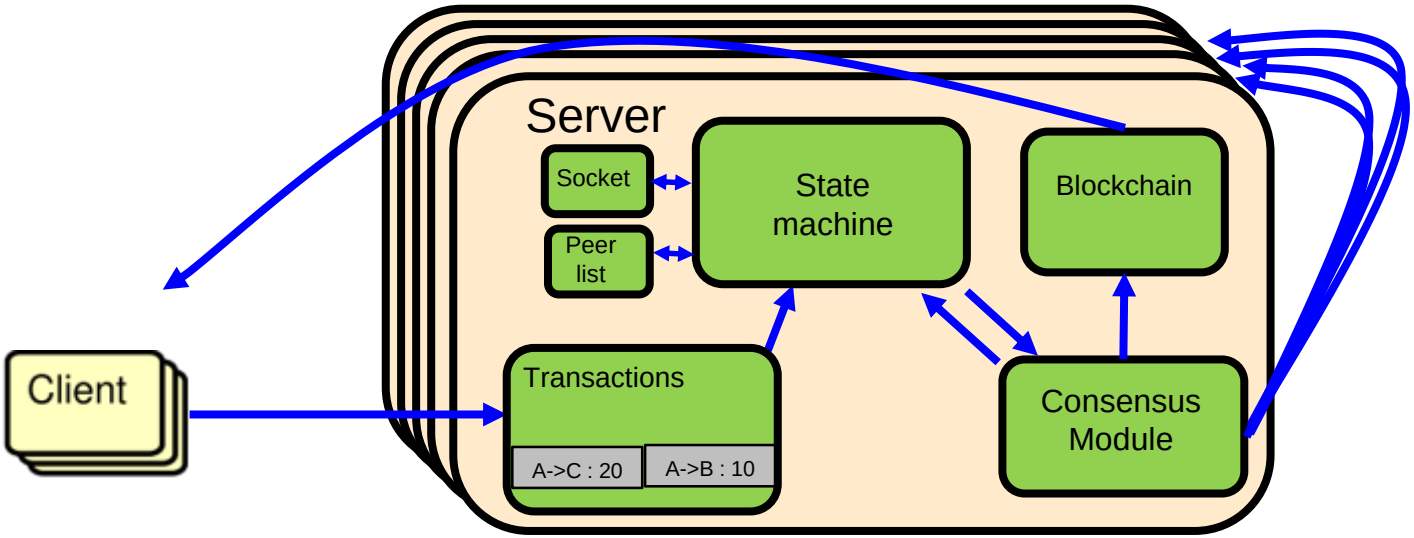
- **Remind our principle**
  - Focus on main component (즉, consensus 와 universal 테스팅)
- **Okay, then… How to implement message passing without P2P module?**
  - Socket 통신을 위한 socket data 모듈을 제공
  - Peer 를 관리하기 위한 data 모듈을 제공
- **이를 통해 BLEEP 구조를 단순화 하고, 핵심 컴포넌트 (consensus) 에 집중**
  - 예를 들어, Small static network, 1-hop message passing 을 가정한 상태에서의 consensus 구현 및 테스트부터
  - 물론 더 복잡한 message passing 로직을 구현하고 테스트하고 싶을 경우에도, 기본 data 모듈을 이용해서 직접 구현 가능하다
- **또한 빠른 시일 내에, gossip 을 위한 P2P 모듈을 라이브러리화 하여 지원할 계획**
  - 이미 구현된 구 버전이 있으며, 별도 branch 에서 테스트 진행 중



Implementing blockchain system model

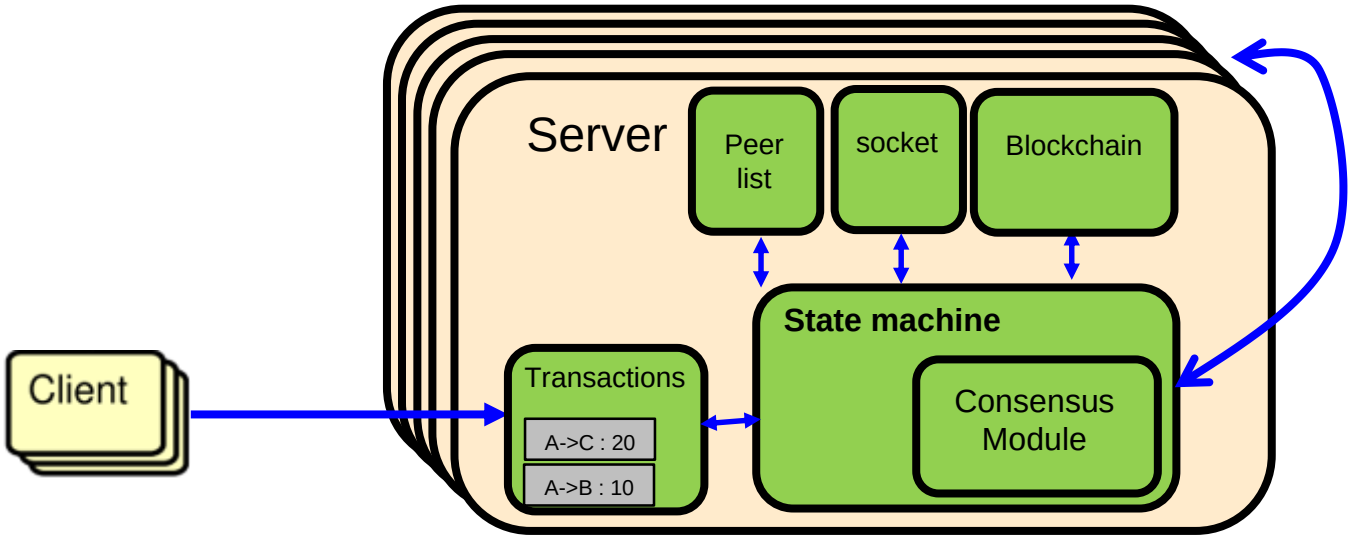
- **Consensus module** 이 **socket, peer** 에 대한 직접적인 접근이 불가?
- **Consensus module** 은 **message passing** 을 구현하기 위한 주체
  - 즉, message passing 을 위해 socket, peer 등에 대한 접근이 필요하다

- 여기서 두가지 다른 접근방법이 가능하고 이에 대한 **design decision** 이 필요하게 된다



Implementing blockchain system model

- **Consensus** 모듈을 **state machine, (**즉 **node** 의 상태 및 동작을 결정하는 컴포넌트**)** 의 일부로 포함시킬 것인가

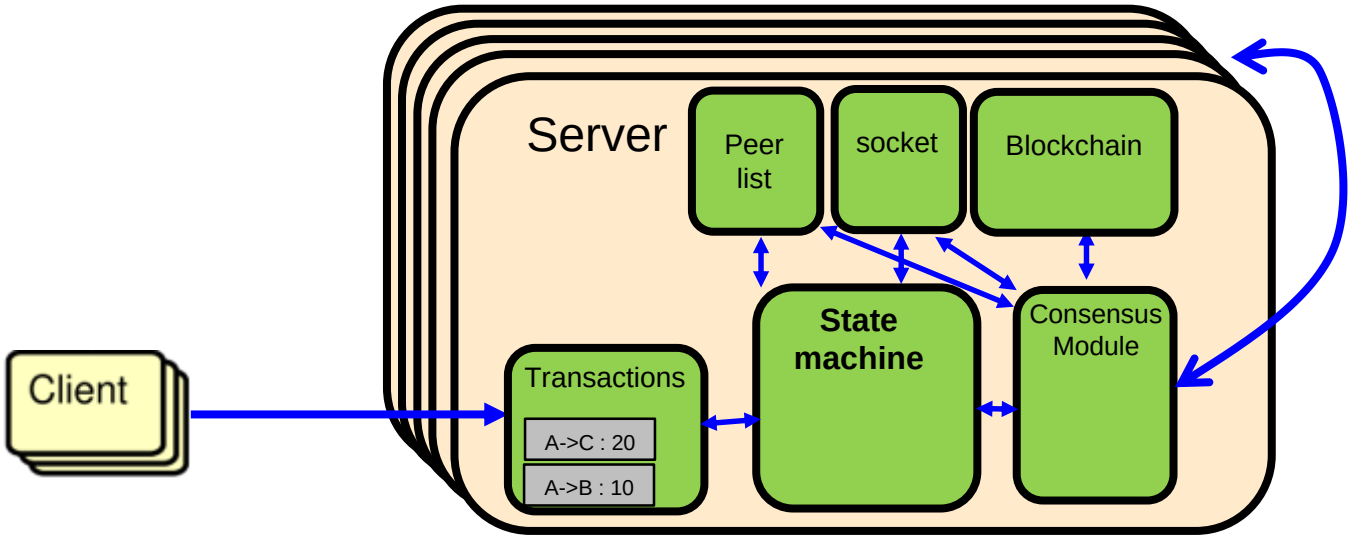


Implementing blockchain system model

- **Consensus** 모듈을 **state machine, (**즉 **node** 의 상태 및 동작을 결정하는 컴포넌트**)** 의 일부로 포함시킬 것인가
- 아니면 **independent** 한 **interface** 를 가진 별도의 모듈로 **library** 화 할 것인가**?**

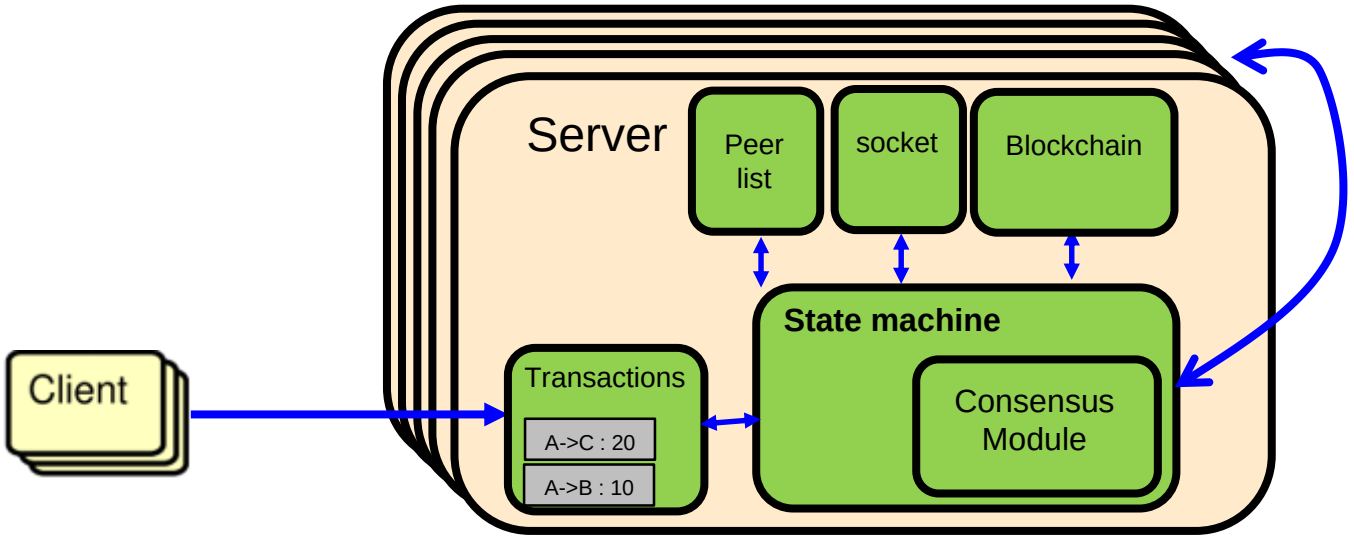
- 다시 한번 말하지만**, design** 이슈라고 할 수 있다 **(**즉**,** 정답이 있는 것이 아니다**)**

- **BLEEP** 은 첫번째 디자인을 선택하였다



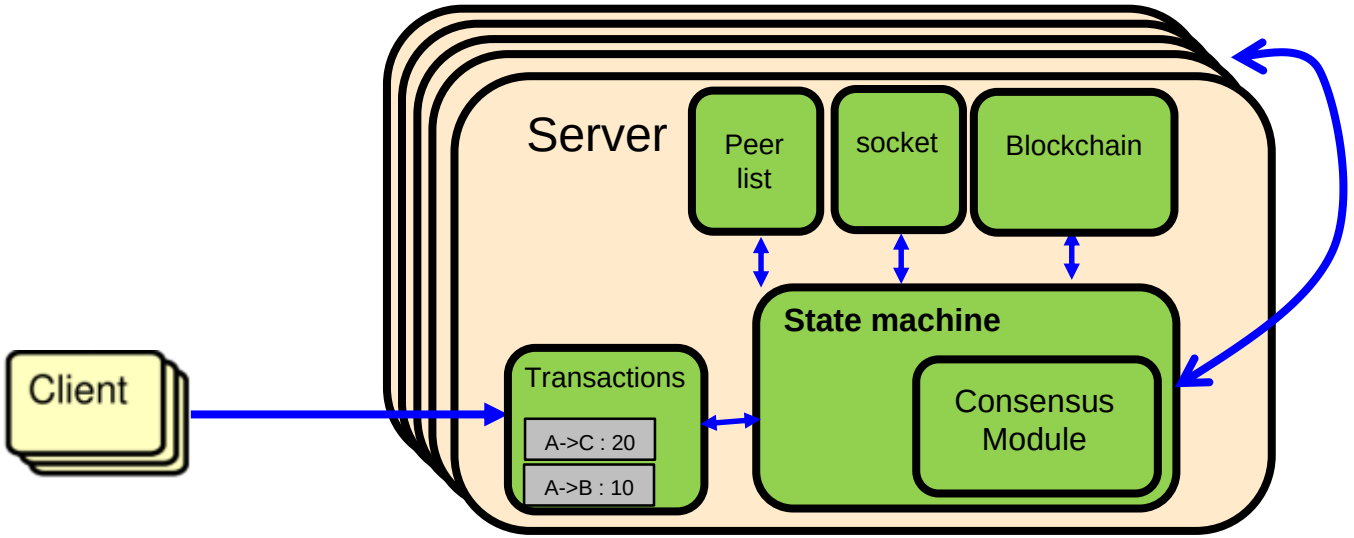
Implementing blockchain system model

- **Consensus 모듈을 state machine, (즉 node 의 상태 및 동작을 결정하는 컴포넌트) 의 일부로 포함시킴**
  - 즉, consensus module 은 library 화 하지 않음
- **Again, remind our principle**
  - 다양한 기능과 알고리즘이 존재할 수 있는 consensus 보다는, 비교적 정형화하기 쉬운 데이터 구조를 component 화 하는데 초점을 맞춤
  - 이를 통해 Easy-to-update, Backward-compatibility 를 만족하는 library 의 구현 가능
- **결과적으로, data 모듈들과, 로직을 구현하는 컴포넌트를 완전히 구분.**



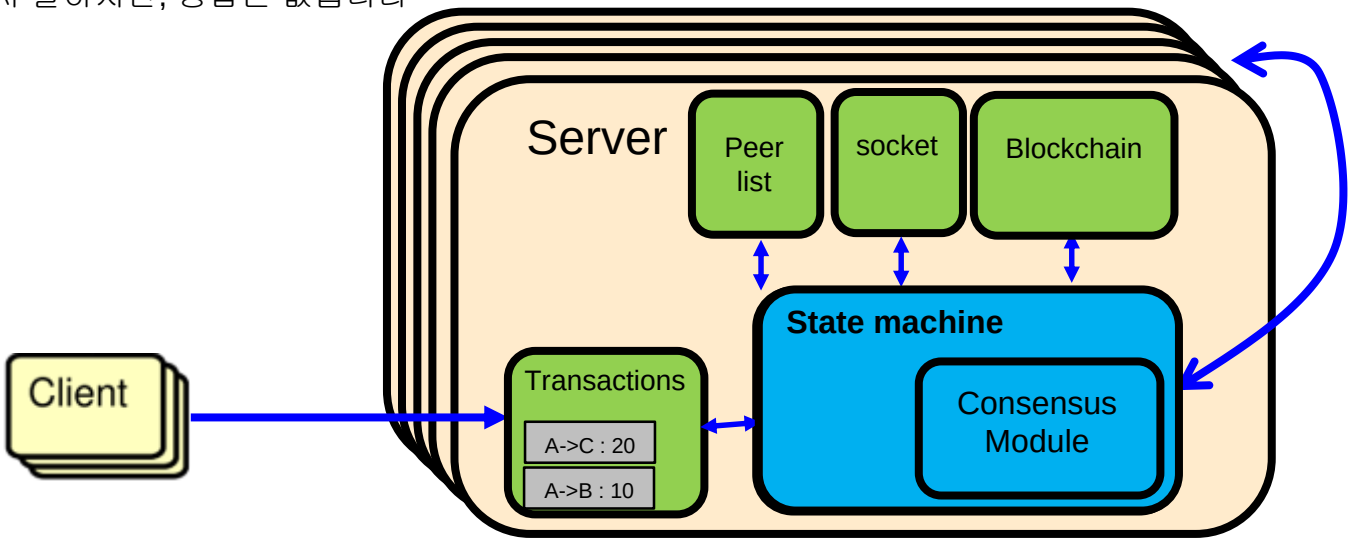
Implementing blockchain system model

- 하지만, **Consensus** 에서 공통된 부분은 **library** 화 할 수 있지 않겠는가?
  - 물론 가능할 것. 이 역시, 우리의 궁극적인 목표.
  - 장기적으로 P2P 모듈과 함께 consensus 모듈도 개발할 예정



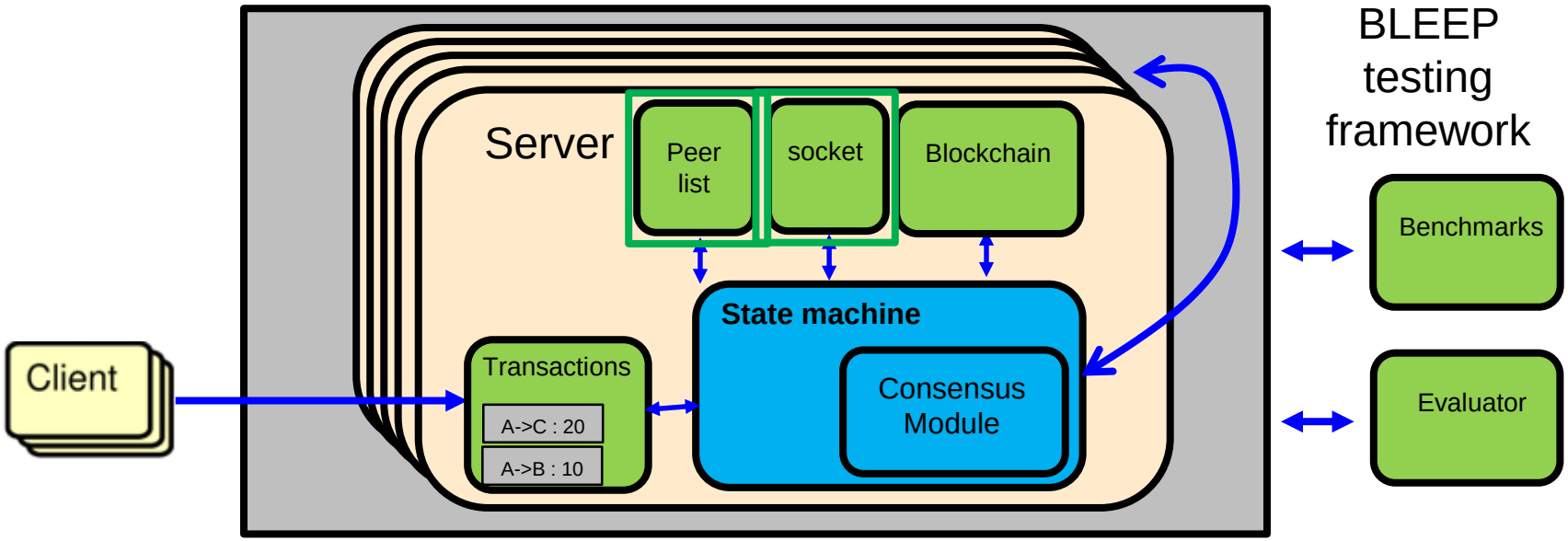
Implementing blockchain system model

- 마지막으로, 블록체인의 추상적 모델을 위해 정의한 **state machine** 의 실구현?
  - node 의 상태 및 동작을 결정하는 컴포넌트
  - 로직 및 알고리즘의 동작을 담당
- 이 개념적인 **State machine** 을 실제 구현에서는 어떻게 디자인하고 구현할 것인가?
- 정말 **state machine** 으로 구현함
- 이 역시 **design choice**
  - 다시 말하지만, 정답은 없습니다



BLEEP : Implementing blockchain system

- **(Transaction, Blockchain 모듈) : node** 에 대한 **input**과 **consensus** 의 결과로 나오는 **output** 을 정의하고 관리하기 위한 모듈
- **node** 간의 통신**(message passing)**을 위한 **socket** 관리 모듈
- **node** 간의 통신**(message passing)**을 위한 **peerlist** 관리 모듈
- 또한, **Correctness** 를 검증할 수 있는 외부 메커니즘 제공 **(수요일 강의)**



BLEEP : blockchain system components

Provided as template (skeleton code)

Algorithms & Logics

StateMachine

mainloop & state transition control

StateHandler

Function handlers for each state.
Implement consensus algorithm

Provided as library

Data Management Modules

Data Management Modules

Transaction Pool | Ledger Management | Peer list Management | etc

Data Classes

Data Classes

Transaction | Block | Ledger | Socket | Peer | Pipe

BLEEP : state machine components

- **StateMachine, StateHandler**
  - 가장 기본이 되는 component
  - Blockchain 이 동작하는 모든 로직을 state machine 형태로 control 함
  - 구현과 디버깅을 쉽게 하기 위해 single thread 로 구현
  - Idle state 에서 event 에 대한 blocking (i/o handling)
  - 새로운 event 가 발생할 경우, state transition 이 이루어지며, 해당하는 state handler 가 로직을 처리함
- **Skeleton 형태로 모듈이 제공되며, 개발자는 원하는 블록체인의 형태에 맞게 statemachine 와 statehandler 를 개발**
  - 현재 구현 예제로서, simple-idle-exit-machine, singlenode-blockchain-machine, doublenode-blockchain-machine 가 있으며, 계속 추가될 예정

StateMachine

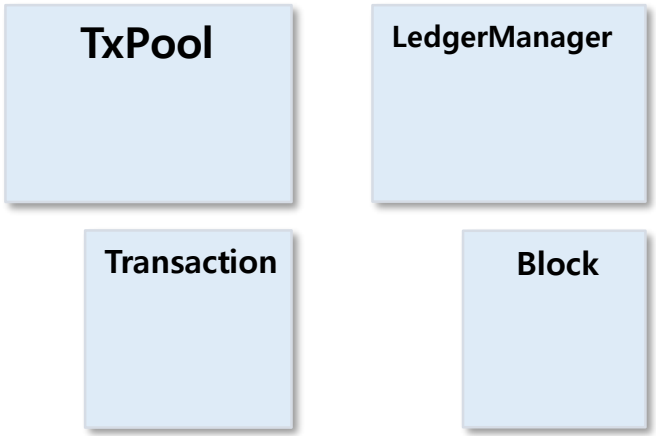
mainloop & state transition control

StateHandler

Function handlers for each state

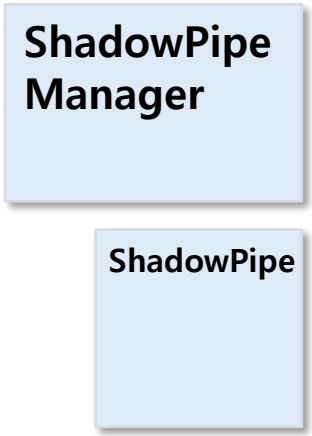
BLEEP : data module library

- **Transaction, TxPool (input management)**
  - Blockchain system 의 input 에 해당하는 Transaction class
  - Transaction 들을 pool 로 관리하기 위한 TxPool class
- **Block, LedgerManager (output management)**
  - Blockchain system 의 output 인 blockchain 의 기본 단위인 Block class
  - Block 은 id 와 포함하고 있는 transaction list 를 가짐
  - Blockchain 에 block을 추가/삭제 등을 할 수 있는 ledgerManager class



BLEEP : data module library

- **ShadowPipe, ShadowPipeManager (client emulation)**
  - Clients 가 blockchain system 을 control 하기 위해 요구하는 다양한 명령들을 에뮬레이션 하기 위한 컴포넌트들
    - 예 : Client 가 새로운 transaction 을 생성하는 상황
    - 예 : 새로운 node 를 distributed network에 참여 시키는 상황 (P2P churn)
  - ShadowPipeManager 를 이용해, ShadowPipe 를 생성할 수 있고, 생성한 ShadowPipe 를 통해 에뮬레이션된 명령이 들어오면, event 가 세팅되어 어떤 명령이 들어온 것인지 확인할 수 있다
  - 자세한 내용은 수요일에 다룰 예정이며, 사용방법은 singlenode-blockchain-machine 예제를 참고

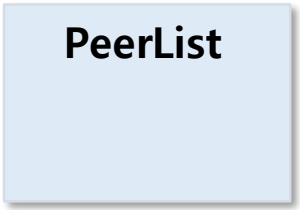


BLEEP : data module library

- 다음으로, 여러 노드 간의 **message passing** 을 구현하기 위한 모듈들을 제공
- **Socket, SocketManager**
  - Socket 을 생성하고 관리
  - Message 를 send 및 recv 하는 동작 처리
- **Peer**
  - 각 node 의 identity 를 위한 모듈
  - 메시지내 source, destination 의 타입
- **Message**
  - Node 간에 서로 information 을 주고 받는 단위
  - Source, Destination, payloadtype, (serialized)payload

BLEEP : data module library

- **PeerList**
  - 앞서 정의한 Peer 들의 집합을 관리하기 위한 data 모듈
  - 특정 peer 를 추가/삭제 및 검색 할 수 있음



- **message passing 과정이 P2P 환경에서 진행될 때.**
  - 향후 구현 예정인 P2P 모듈 역시 위의 PeerList 모듈을 기반으로 membership control, gossiping algorithm 등을 구현할 예정

Current implementation & Schedule

- 현재, **transaction(input), block, blockchain (output), shadowpipe** 까지 구현되어 있음
- 그리고 금주 내로, 노드 간의 **message passing** 을 위한 **socket, Message, peer** 모듈까지 포팅을 완료할 예정
- 다음주 내로, **p2p** 환경 지원까지 고려하여 **peerlist** 모듈을 포팅할 예정

BLEEP advantages

- **Why use BLEEP library**
  - Based on abstract blockchain model (distributed system formalization)
    - Applicable to various blockchain system, easy to understand
  - Easy to implement
  - Test your own implementation (수요일)
    - 여러 노드에 Input 을 임의로 발생시켜서 동작시키기
    - 노드 간의 socket 연결 과정 디버깅
    - 주고 받는 message 에 대한 디버깅/분석 및 시각화
    - 생성하는 output (blockchain) 에 대한 correctness 검증
    - 이 모든 테스트들을 별도의 추가 구현 없이, 블록체인 구현 시에 BLEEP library 를 사용하기만 하면 자동적으로 이루어 질수 있도록 함

BLEEP usage example

- Singlenode consensus
- Raft consensus
- Nakamoto consensus (PoW)

Singlenode consensus

- Let's say we have a single node system

thesecretlivesofdata.com/raft/

The Secret Lives of Data

Singlenode consensus

- For this example, you can think of the node as a database server that stores blockchain.

thesecretlivesofdata.com/raft/

The Secret Lives of Data

Singlenode consensus

- We also have a client that can send a value (transaction) to the server.

thesecretlivesofdata.com/raft/

The Secret Lives of Data

Singlenode consensus

- We also have a client that can send a value (transaction) to the server.

thesecretlivesofdata.com/raft/

The Secret Lives of Data

Singlenode consensus

- We also have a client that can send a value (transaction) to the server.

thesecretlivesofdata.com/raft/

The Secret Lives of Data

Singlenode consensus

- We also have a client that can send a value (transaction) to the server.

thesecretlivesofdata.com/raft/

The Secret Lives of Data

Singlenode consensus

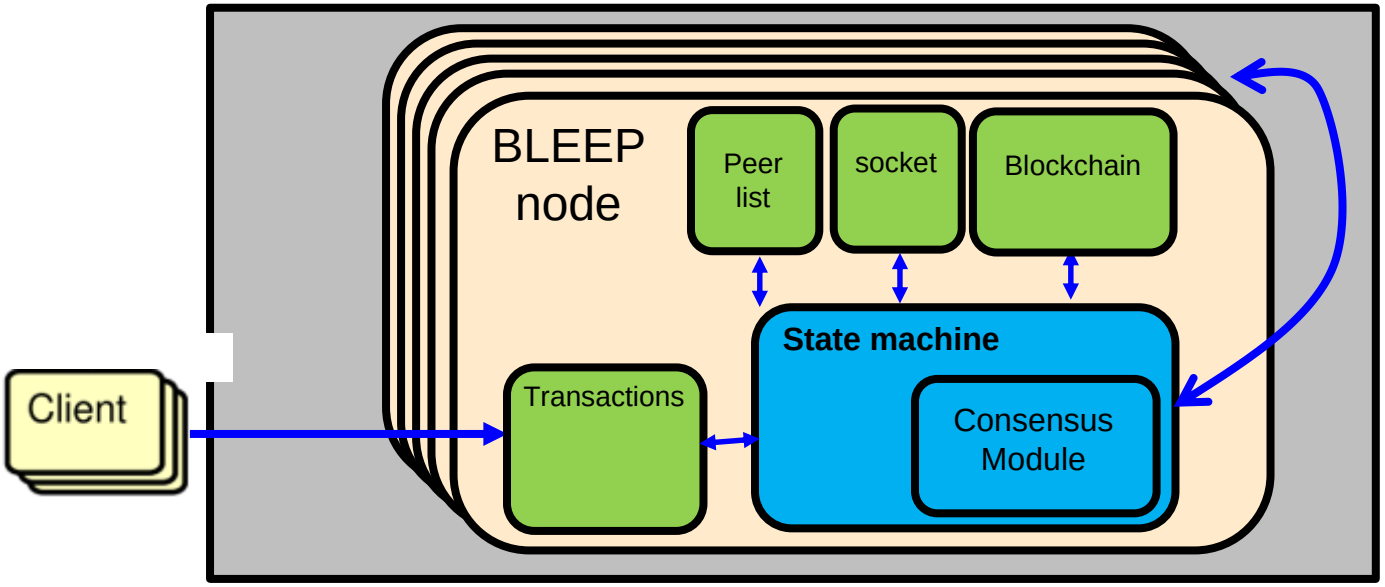
- We also have a client that can send a value (transaction) to the server.
- Coming to agreement, or *consensus*, on that value is easy with one node.

thesecretlivesofdata.com/raft/

The Secret Lives of Data

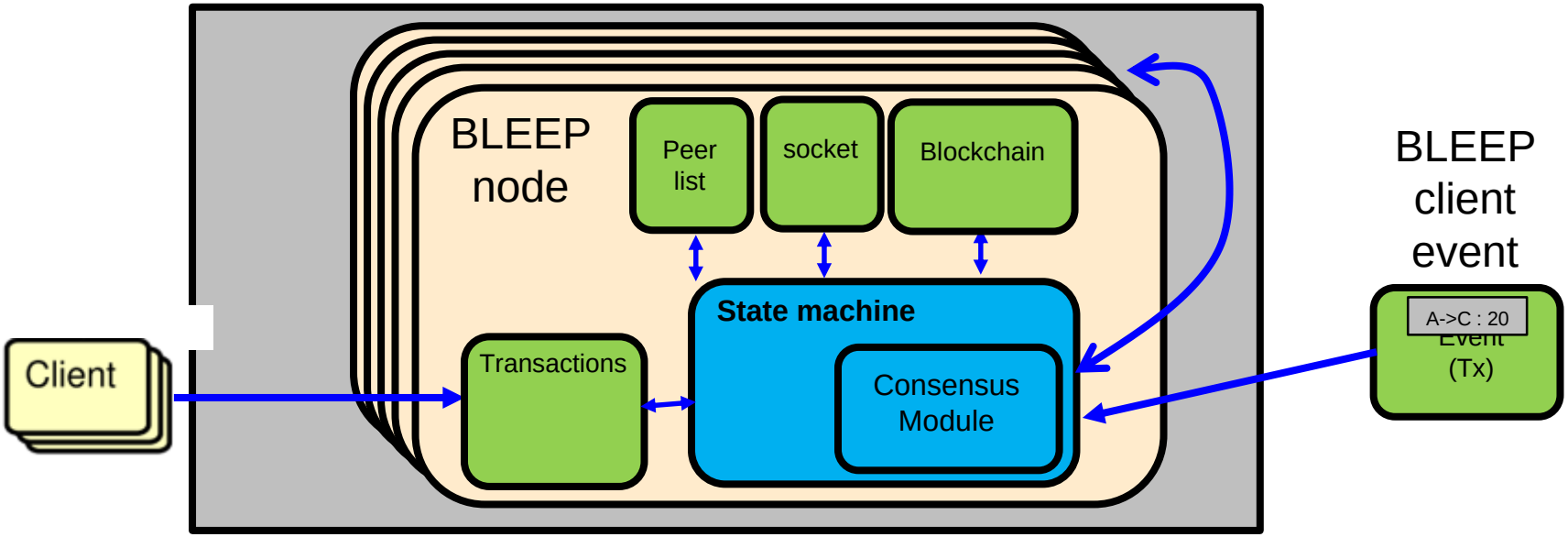
Singlenode consensus – client behavior?

- 실제 시스템에서는 **client** 가 실제로 동작하겠지만, 테스트를 위한 **BLEEP** 환경에서는 어떻게 **client** 의 행동을 구현할 수 있는가**?**
- 시뮬레이터 상에서 동작하는 **BLEEP** 에 **client** 의 행동을 에뮬레이션 하려면**?**



Singlenode consensus

- **BLEEP** 에서는 **Client** 의 동작을 **BLEEP event** 를 발생시키고, **event** 에 대한 핸들러를 **state machine** 에 구현하는 방식으로 에뮬레이션 하였다
  - 모든 BLEEP 노드들은 시뮬레이션 환경에서 동작하게 되며, BLEEP은 임의의 virtual clock 에 특정 이벤트를 특정 노드로 발생 시킬 수 있다. 이를 통해, client 가 node 에 transaction 을 input 으로 넣는 작업을 에뮬레이션 할 수 있다
- **BLEEP node** 의 **state machine** 이 **event** 를 받으면, 이벤트 핸들러를 동작시키게 되고, 핸들러는 실제 **transaction** 을 생성하여 **txpool** 에 **push** 하게 된다



Singlenode consensus - 구현하기

- **Implement StateHandlers**

```
namespace singlenode_blockchain_machine {

    void RegisterStateHandlers();

    StateEnum idleStateHandler();
    StateEnum libevEventTriggeredStateHandler();
    StateEnum shadowPipeEventNotifiedStateHandler();
    StateEnum appendBlockStateHandler();
    StateEnum exitStateHandler();

};
```

```
StateEnum singlenode_blockchain_machine::appendBlockStateHandler() {
    std::cout << "appendBlock state handler executed!" << "\n";
    // This state is for appending a new block.
    M_Assert(gStateMachine.txPool.GetPendingTxNum() >= block_tx_num, "requires enough txs");

    std::shared_ptr<Block> newBlock(new Block("", gStateMachine.txPool.GetTxs(block_tx_num)));
    gStateMachine.ledgerManager.AppendBlock(newBlock);
    gStateMachine.txPool.RemoveTxs(newBlock->GetTransactions());

    std::cout << utility::GetGlobalClock() << ":Block is appended" << "\n";
    std::cout << *newBlock << "\n";

    return StateEnum::idle;
}
```

Singlenode consensus

- **Build statemachine**

```
#ifndef STATE_MACHINE_H
#define STATE_MACHINE_H

#include "./examples/singlenode-blockchain-machine/State.h"
#include "./examples/singlenode-blockchain-machine/StateHandler.h"
using namespace singlenode_blockchain_machine;
```

```
set(SRCS
  utility/GlobalClock.cpp
  utility/NodeInfo.cpp
  utility/ArgsManager.cpp
  crypto/SHA256.cpp
  # statemachine/examples/simple-idle-exit-machine/State.cpp
  # statemachine/examples/simple-idle-exit-machine/StateHandler.cpp
  statemachine/examples/singlenode-blockchain-machine/State.cpp
  statemachine/examples/singlenode-blockchain-machine/StateHandler.cpp
```

Singlenode consensus

- Configuring experiment

```
<node id="bleep1">
  <application plugin="PEER" time="13" arguments="" />
  <command id="generateTx" starttime="30" arguments="1 2 30" />
  <command id="generateTx" starttime="30" arguments="1 2 30" />
  <command id="generateTx" starttime="30" arguments="1 2 30" />
  <command id="generateTx" starttime="30" arguments="1 2 30" />
  <command id="generateTx" starttime="30" arguments="1 2 30" />
```

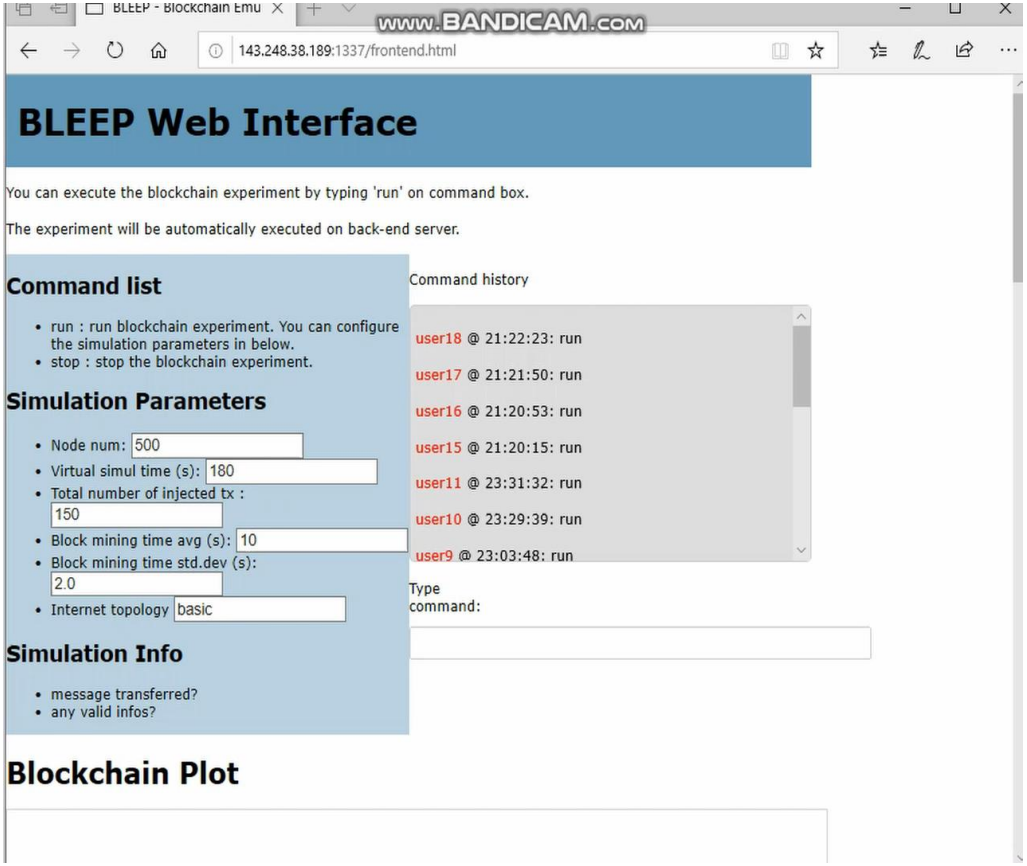
Singlenode consensus

- **Results**

```
Testing node up
[StateMachineLog::Start blockchain statemachine] initial state is idleState
idle state handler executed!
shadowpipe iocallback called!
[StateMachineLog::State transition]: idleState to libevEventTriggeredState
[StateMachineLog::State transition]: libevEventTriggeredState to shadowPipeEventNotifiedState
Successfully received shadow-generated command (generateTx:1 2 30)
[StateMachineLog::State transition]: shadowPipeEventNotifiedState to idleState
idle state handler executed!
shadowpipe iocallback called!
[StateMachineLog::State transition]: idleState to libevEventTriggeredState
[StateMachineLog::State transition]: libevEventTriggeredState to shadowPipeEventNotifiedState
Successfully received shadow-generated command (generateTx:1 2 30)
[StateMachineLog::State transition]: shadowPipeEventNotifiedState to idleState
idle state handler executed!
shadowpipe iocallback called!
[StateMachineLog::State transition]: idleState to libevEventTriggeredState
[StateMachineLog::State transition]: libevEventTriggeredState to shadowPipeEventNotifiedState
Successfully received shadow-generated command (generateTx:1 2 30)
[StateMachineLog::State transition]: shadowPipeEventNotifiedState to idleState
idle state handler executed!
shadowpipe iocallback called!
[StateMachineLog::State transition]: idleState to libevEventTriggeredState
[StateMachineLog::State transition]: libevEventTriggeredState to shadowPipeEventNotifiedState
Successfully received shadow-generated command (generateTx:1 2 30)
[StateMachineLog::State transition]: shadowPipeEventNotifiedState to idleState
idle state handler executed!
shadowpipe iocallback called!
[StateMachineLog::State transition]: idleState to libevEventTriggeredState
[StateMachineLog::State transition]: libevEventTriggeredState to shadowPipeEventNotifiedState
Successfully received shadow-generated command (generateTx:1 2 30)
[StateMachineLog::State transition]: shadowPipeEventNotifiedState to appendBlockState
appendBlock state handler executed!
17:Block is appended
:Block has following transactions
:0 sends 30 to 2
:0 sends 30 to 2
:0 sends 30 to 2
:0 sends 30 to 2
:0 sends 30 to 2
```

Singlenode consensus

- Visualization



BLEEP blockchain example

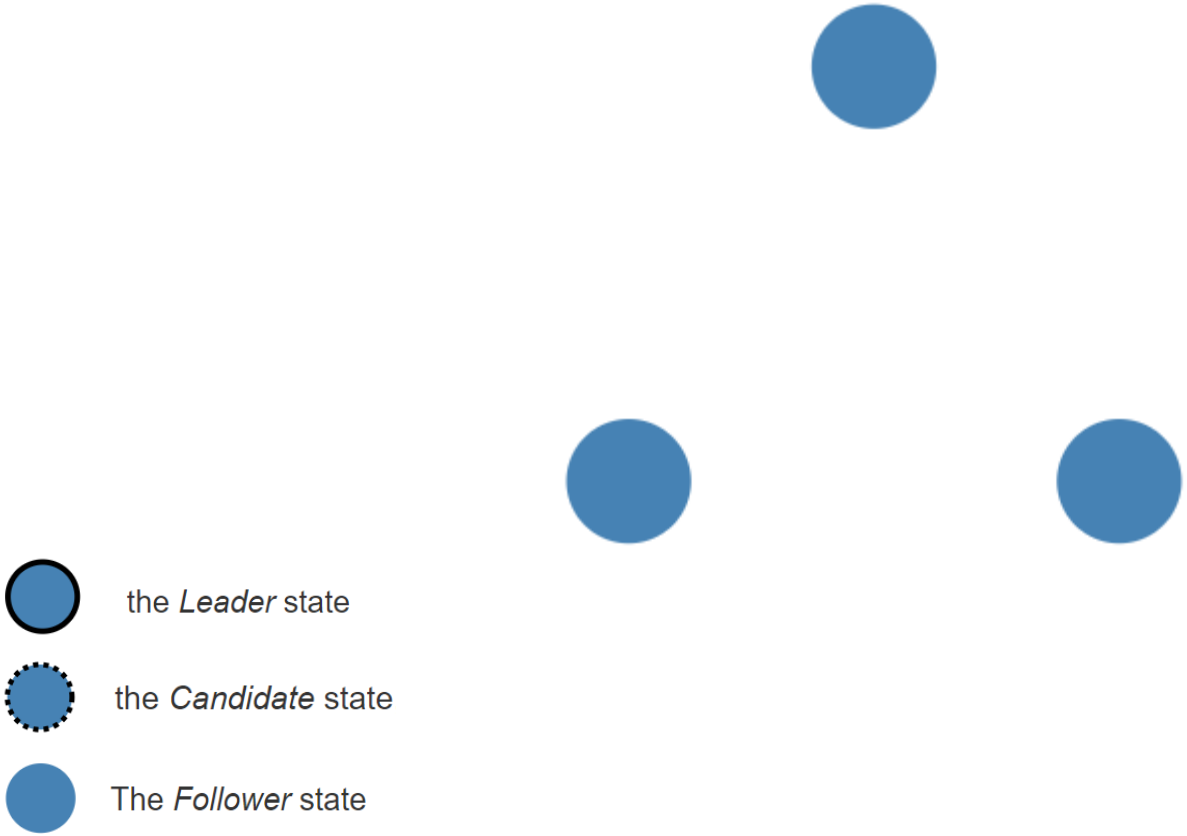
- Singlenode consensus
- **Raft consensus**
- Nakamoto consensus (PoW)

Raft consensus

- **Raft consensus**
  - Consensus algorithm designed to be more understandable than Paxos
  - Leader-based approach
  - USENIX ATC 2014 best paper
    - “In Search of an Understandable Consensus Algorithm”, D. Ongaro et al.
  - Tolerant only for fail-stop fault, not for byzantine fault

Raft Consensus

- We need modules for implementing ‘message passing’
- So.. We will implement this example later



Nakamoto consensus

- **PoW** 에 대해 구현이 되어 있으나, 새로운 모듈화된 구조에 대해서는 아직 포팅이 진행되지 않았음
- **PeerList** 모듈을 구현한 뒤에, 포팅을 완료하고 예제로 업로드할 예정

Summary

- **BLEEP : Blockchain Emulation and Evaluation Platform**
  - 모든 종류의 블록체인을 동작시키고 평가할 수 있는 플랫폼
  - 블록체인 구현 템플릿 및 라이브러리 제공 (Today's topic)
  - 블록체인 테스트 플랫폼 제공 (수요일 topic)
- **블록체인 추상화 모델**
  - Distributed System formalization 에 기반한 system 모델 제시
  - Blockchain system 의 동작 및 correctness 를 formalization
- **BLEEP : blockchain library & template**
  - Independent, backward-compatible 한 블록체인 데이터 모듈 디자인
  - Flexible, easy-to-update 한 블록체인 알고리즘 구현 모듈 디자인
- **BLEEP advantages**
  - 추상화된 모델을 이용해 블록체인을 쉽게 이해하고, 모듈들을 사용해 직접 구현해 볼 수 있고, 테스트 기능을 통해 쉽게 테스트할 수 있음