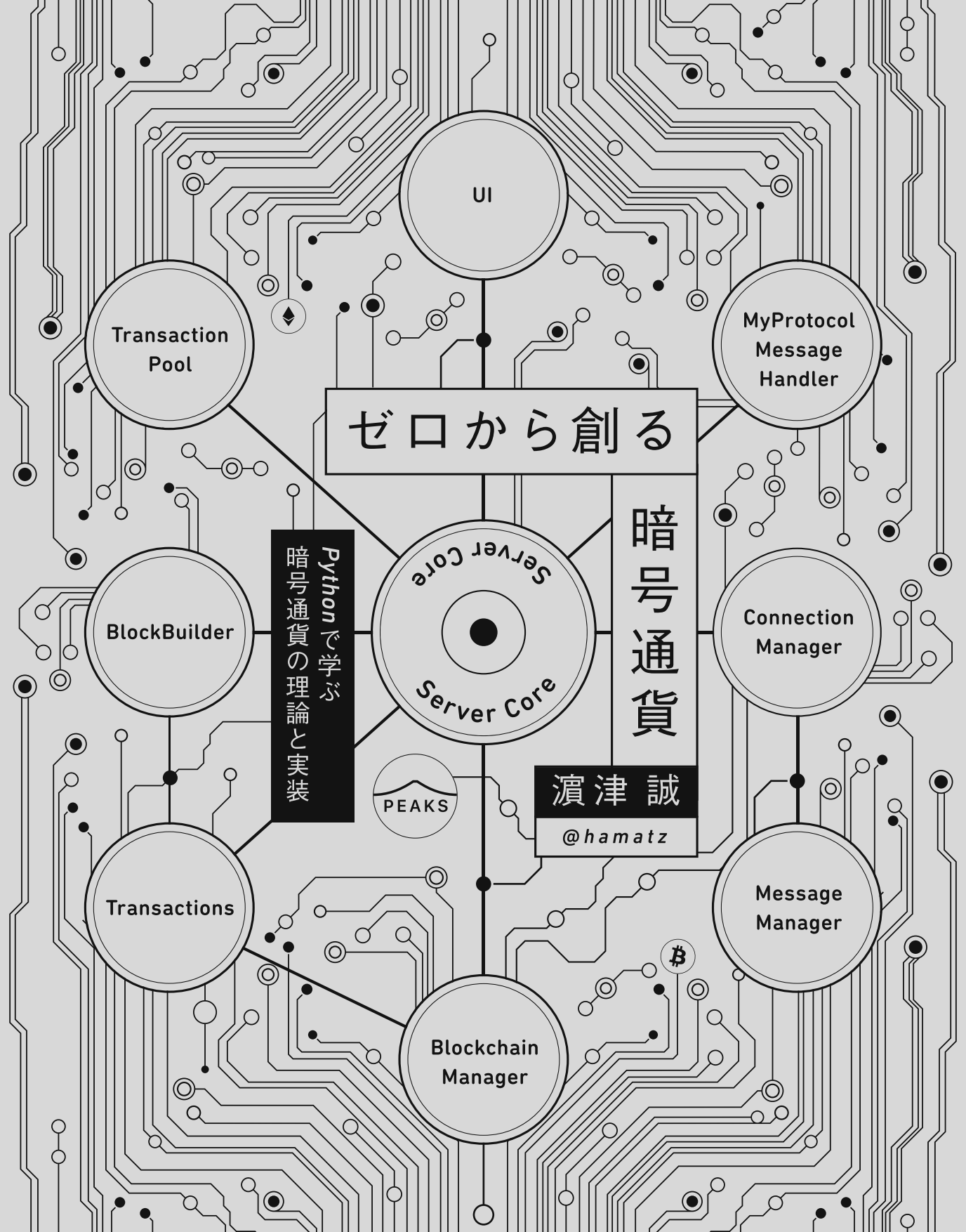

UI
Transaction Pool
MyProtocol Message Handler
ゼロから創る
暗号通貨
Python で学ぶ
暗号通貨の理論と実装
Server Core
Server Core
BlockBuilder
Connection Manager
PEAKS
濵津 誠
@hamatz
Transactions
Message Manager
Blockchain Manager

はじめに

私は「こういうものです。覚えなさい」と教えられるのが昔から大嫌いでした。たとえば高校の数学の授業。黒板に書かれた問題に対する解説をただただノートにとり、それを暗記する。たとえそれが連綿と続く人類の歴史の積み重ねの中で獲られた知恵の結晶だとしても、私にとっては苦痛以外の何ものでもない「作業」でした。その性分は今でも変わっておらず、紙に書かれた何かをただ記憶する、という作業がとても苦手です。

さて、そんな性分を抱える私が、職務上のいろいろな出会いを通じて「暗号通貨に関連する技術を学ぼう」ということになり、有名どころの本を揃えたりTwitterで話題の記事などを漁ったりしていたところ、ある1つの問題にぶつかりました。すなわち、

- **「ビットコインの仕組みはこのようになっています」**
- **「ブロックチェーンとはこういうものです」**

といった「事実の紹介」しかしてくれない教材ばかりが目につき、「なぜそのような形に至ったのか？」という肝心な「思考の過程」を伝えてくれるものに出会えなかった、ということです。

- **「こんなことを実現したいなら、こういうふうに作ればよいのでは？」**
- **「ああ、なるほど。こういう問題が起こるのか……」**

あえて不完全なところからスタートし、失敗や反省を繰り返しながら「創る」ことで完成品に至るまでの思考の過程をシェアする。私が欲しかったのはそういうテキストでした。本書は、私のような性分を持った人たちに対して「あのときこういう本があれば良かった」と筆者自らが思えるものを提供することを大きな目的としています。

このところ、ブロックチェーンという技術は、ネット上でもその他の媒体でも用語を目にしない日はないのではないかというほどポピュラーになりつつあります。ですが、その中身をしっかりと理解し、必要に応じて自分の用途に応じてカスタマイズできる人となると、まだまだ僅か。学生から現役のエンジニアまで、ブロックチェーンの技術を学ぼうとする方々の動機はさまざまでしょう。本書では、暗号通貨技術の本質を掴むために、細かな部分を削ぎ落としてデザインされた**SimpleBitcoin**というプラットフォームをゼロから創造していきます。

SimpleBitcoinの完成までの過程には、当然、失敗もあります。「学ぶ」という目的を外れない程度のシンプルさを保つために、あえて不完全なまま終わる部分もあります。とはいえ、不完全であると

いうことさえ理解できていれば、「今回は解決を見送ったこの問題は、たとえば本家のビットコインではどう対処しているのだろう？他の暗号通貨ではどうしているんだろう？」といった、暗号通貨技術全般に対する勘所は身につくはずです。

本書の解説、そしてSimpleBitcoinの実装には、コードの可読性の高さや近年の流行を鑑み、プログラミング言語**Python**（バージョン3系）を採用しました。読み進める上でPythonに対する深い知識は前提としていないので安心してください。ただし、Pythonの解説書ではありませんので、環境のセットアップをはじめとするプログラミング言語Pythonの詳細な解説は含めてありません。必要であれば他の書籍などで適宜補完してください。

繰り返しになりますが、本書で創っていくSimpleBitcoinのゴールは、既存の実装の完全なコピー実装ではありません。暗号通貨のエッセンスをつかみ取れることがSimpleBitcoinの主眼です。本書を通じ、既存の仕組みの仕様を単純に受け入れるのではなく、より良いものにするためにどんな工夫が考えられるかを常に意識できるようなブロックチェーン・エンジニアが一人でも多く誕生してくれれば幸いです。

サンプルコード

次のリポジトリに本書のサンプルコードが掲載されています。

- https://github.com/peaks-cc/cryptocurrency-samplecode

各章でサンプルコードの指定がある場合は、そちらも合わせて確認ください。

クラウドファンディングとPEAKS

本書は技術書クラウドファンディング・サービスである「PEAKS」のプロジェクトとして開始され、795人の支援者のサポートによって作られました。出資者特典である「アーリーアクセス」「ベータ版公開」でいただいた多くのコメント、ご意見も反映されております。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません

目次

第 1 章

Simple Bitcoin：価値の移転を記録するプロトコル

本章では、これから本書を通じて学んでいく暗号通貨の実装について最終的なゴールを明らかにし、そこまでの大まかな道筋を示します。プログラミングに入る前に、暗号通貨の全体像を把握することが目的です。次のような点に着目しながら読み進めてみてください。

1. SimpleBitcoinで実現しようとしているもの。できること、できないこと
2. SimpleBitcoinを実現するために必要な構成要素
3. 実装を進める順序とその理由

「兄さん、僕、暗号通貨のことをもっとよく知りたいんだ」

春の週末の昼下がり、自室で論文作成に集中していた**晃（あきら）**は、闖入者である**樹（いつき）**の姿にチラと目をやった。だが、そのまま再び作業中の画面に視線を戻す。

「そうかい」

「いや、だから暗号通貨やりたいんだよ。ブロックチェーン！」

晃は心底意外であるかのような表情を樹に向ける。

「愚弟よ、どうしたね？何か悪いものでも拾い食いしたかい？」

「なんでだよっ！」

完全なる上から目線に樹は思わず声を荒げてしまう。

「(ぐぬぬ……。兄さんと話すといつもこうなるな。なかなか自分のペースに持っていけない……)」

「それなら、急にどうしたというんだい？まさか研究に目覚めたというわけでもないだろうに」

このままお前と話を続ける価値があるのか？縁なしのメガネの奥からそんな問いかけを含んだ視線を弟に投げかける晃。集中の邪魔をされたせいか、どうやら彼はあまり機嫌がよいとはいえないようだ。

「ぼ、僕が勉強しちゃ悪いのかよっ！」

「もちろん悪くないさ。悪くはない。だが、動機が気になってね」

樹には、晃の目が「どうせ大した目的ではないのだろう」と語っているように見えた。

「周りのやつらが、インターンとか始めてさ。それで、僕も何かやんなきゃって気になって……」

「ほう。それで？」

晃の声のトーンが一段低くなった気がした。

「ほ、ほら。ブロックチェーンとか流行ってるし、それで何かサービスを自分で作れないかなーって。」

「なるほど？」

ここから先、言葉を間違えると部屋から叩き出される。そんな予感に内心で冷や汗をかきつつ、頭をフル回転させる樹。

「話題の実装をForkして、それを改造して通貨を作って実験してみようと思ったんだ。でも、ネットで調べたパラメータをいじってbuildしただけだと、さっぱり動かなくて」

「ほう？」

樹の話に少し興味を持ったのか、意外にも晃の声のトーンが若干変わる。

「それで思ったんだ。もしパラメータをいじって、こいつがなんとか動くようになったとしても、これだと完全にブラックボックスだよなって。そんなものを使って、自分で何が作れるんだろうって」

「へえ」

珍しい光景を見ているかのように樹に目を向ける晃。どことなく嬉しそうにも見える。樹は言葉を続けた。

「動きはしたけど、その中身が何もわからないままでは、自分で付け足した機能の過不足もわからない。お金のようなものを扱おうってのに、それってどうかと思ったんだ」

「うむ。愚弟かとばかり思っていたが、いつの間にか、ずいぶんとまともなことを言うようになったじゃないか」

「だから、暗号通貨のメカニズムをちゃんと把握したいんだ。暗号通貨をゼロから勉強したい！そう思ってビットコインの実装を読んでみたりしたんだけど、いまいちよくわからないんだ……」

「ソースを読もうとしたのか」

「だから兄さん。なんかこう、ソースを読むだけで暗号通貨のすべてがわかるような、それでいてソースが読みやすい最高の教材、そういうのあったら教えてくれよ！」

「そんな都合のいいものがあったら、とっくにWeb検索でヒットしているとは思わなかったのかい？」

「それはそうなんだけど……」

常識で考えればそんな都合のいいものはない。そんなことはわかっている。しかし、次に踏み出す一歩が欲しいのだ。

「結論から言うと、そんなものはないだろう。しかし、そうだな……」

ふと何かを思いついたのか、そこで次の言葉を探す晃。樹は、メガネの奥で細くなった晃の目と弧を描く唇に気がついて怖気づく。

「(あ！これ、アレだ、何か企んでるときの顔だ……。ヤバい予感。逃げ……)」

「創ればいいんだよ。君自身で、そういう教材を」

「えっ？僕が？」

楽をして、いい感じのサービスを立ち上げて、周りにデカい顔をしたい。樹が欲しかったのはそのための近道だったのだが、何やら面倒な話になってしまった。

「自慢したいだろう？周りの連中に、『ライブラリを叩くアプリを作ってるだけのお前らとは違うんだぜ、オレは！』って？」

「うっ…！」

「自慢したいだろう？周りの連中に、『ブロックチェーン完全に理解したとか言ってるけど、お前ら実装したことあんの？』って？」

「ううううっ…！」

図星だった。完全に読まれている。

「まぁ、承認欲求は悪いものではない。」

そこで樹と真っ直ぐ目を合わせ、晃は言葉を続ける。

「ただ、君は楽な方法で得た優越感で満足できるのかい？同じ近道を見つけた誰かに追い抜かれる日がくるのに怯えながら、自分は特別なんだって吹聴し続けるのかい？」

「うううううううっ……」

みんなに褒められたい。しかし、できれば頑張りたくはない。そんな想いの狭間で悩む樹。

「仮に、君が望むような簡単なコードがあったとして、人のコードからその試行錯誤の歴史を読み取るのは難しい。本当に何かを理解したいのなら、自らの手でゼロからそれを創るべきだとは思わないかい？」

「いや、別にそこまで本格的じゃなくても……」

「そう思わないのかい？」

「あ、はい、そう思います……」

「君にとって残念なことに、写経すれば動くような完成品はない。しかし、幸いなことに、そのレシピならば私の頭の中にある」

「えっ？マジで！」

「そのレシピを通して、君は暗号通貨についての学びと道しるべを得られるだろう。私のほうも、いまは自分の頭にしかないレシピが、はたして君にも咀嚼可能なレベルまでブレークダウンできているかどうかを確認できる。だから、これは完全なWin-Winだ。素晴らしい」

一人、合点がいったかのようにうなずく晃。

「僕はまだ、やると決めたわけじゃないんだけど……」

「ということは、世界のどこかで誰かが君のために使える教材を作ってくれる日がくるのを、ただただ座して待つってことかい？」

「いや、そういうつもりでもないんだけど……」

「はっきりしないな。**ゼロから創る暗号通貨**、やるのかい？それともやめとくかい？」

「や、やるよ！やってやりますよぉぉ！」

こうして樹の暗号通貨学習がスタートしたのである。

1.1 これから作ろうとするものは何か？あるいは何ではないか？

「まずは、これから作る暗号通貨の名前を決めておこうか」

「名前って、つまり『ビットコイン（Bitcoin）』とか『イーサリアム（Ethereum）』みたいな？」

「そうだ。学習用のシンプルな実装であることを強調するために、**SimpleBitcoin**という名前にしようと思う」

「ちょっと弱そうだね……」

「うむ……。それじゃあ、はじめにはっきりと言っておこう。これから作るSimpleBitcoinは、暗号通貨のメカニズムを学ぶための学習用サンプルだ。間違っても『完成したらそのまま運用してICO(Initial Coin Offering)で大金ゲットやー！』というものではない。そんな夢が叶うことはないので、そこは勘違いしないこと」

「えっ、そうだったの？」

「当然だ。君は読みやすいソースコードがお望みなんだろう？詳細な作り込みを省く以上、たとえばセキュリティに穴ができないわけがない」

「そういわれてみれば……」

「そして、SimpleBitcoinは既存の暗号通貨はベースにしない。したがって、便利な開発用ライブラリなどは一切存在しない。暗号化や通信処理などはライブラリに頼るが、そうした基礎的な部分を除けば、すべてスクラッチから『創る』ことになる。お望みどおり『ゼロから』というわけだ」

「な、なんかすごそう……。僕にそんな大それたものが作れるのかな……？」

「まぁ、できないんじゃないかな？」

「えっ？」

「何もいわずにできてしまうのなら、今ここに君はいないだろう？できないから、君は私のところにきた。つまづけばいい。大いに間違えればいいのさ」

「なるほど。間違えていたり、見落としがあったりしたら、兄さんが助けてくれるってこと？」

「私は正解に向かうためのヒントを出すだけさ。君が自分の力で考え、悩むからこそ、実になるってもんだからね」

「それが兄さんのいう『レシピ』ってやつか……」

「そのとおり」

「わかった。じゃあ、何から手をつければいいのかな？」

「まずはSimpleBitcoinの仕様から考えていこう。何を実現するのか、そして何を実現しないのか、すべてはそこを決めないと始まらない」

「実現する機能かぁ……。暗号通貨というからには、まずはやっぱりユーザー間でお金のやり取りはしたいよね」

「お金？君はSimpleBitcoinに、我々が日常的に使っているお金に換算可能な価値を持たせるつもりかい？」

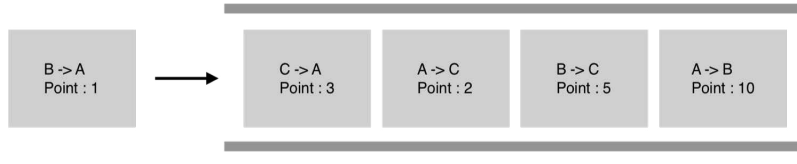
「えっ、だって『通貨』でしょ？」

「もちろん君の気持ちはわかる。しかし、**ビットコインだってリアルなお金との交換可能性なんて仕様上は保証していない**。誰かが勝手にビットコインに価値があると判断し、その時価が交換所などを通じて決められてるだけだ」

「そうか……。つまり兄さんが言いたいのは、リアルなお金の移動はSimpleBitcoinとは関係ない、ということ？」

「そのとおり。**何らかの価値が、『誰かから誰か』あるいは『何かから何か』に移動したことを、時系列をもって記録する**のがSimpleBitcoinの役割だ。**SimpleBitcoinでできることは、それ以上でも以下でもない**と考えるべきだろうね」

「なるほどー」



● **図 1.1　SimpleBitcoin は価値の移動を時系列で記録するプロトコル**

「余談になるが、君が先ほど名前を出したイーサリアムなどの暗号通貨では、スマートコントラクト[注1)]のようなさらに高度な機能も導入されている。そうした高度な機能については、SimpleBitcoinという名前から程遠くなりそうなので、次のステップで実現することにして今は考えないことにする」

「たしかに、今はあれこれ手を広げるタイミングじゃないよね。つまり、SimpleBitcoinとして作るのは、『価値の移動を記録する』仕組みということになるのかな」

「手を広げるべきでないのはそのとおり。しかし、『価値の移動を記録する』という機能だけでは

注 1)　契約を機械的に自動実行する仕組み。ある条件を満たしたときに自動的に支払いが行われる、といったことを実現したいときに利用される。概念としては、1994 年に Nick Szabo という法学者・暗号学者によって提唱された。そこでは代表例として自動販売機が挙げられている。イーサリアムの登場によって、暗号通貨の代表的な活用法として認知されるに至る

暗号通貨という名前から程遠いぞ」

「えっ、どういうこと？」

「『価値の移動を記録する』というだけなら、サーバーでも立てて、ユーザー間でポイント交換可能なサービスでも作ればそれで済む話じゃないかね？」

「あ、そうか！『価値の移動を記録する』という要件だと暗号通貨を使う必然性がどこにもない。そういうこと？」

「そのとおり」

「そういわれてみると、暗号通貨の特徴というか、**暗号通貨でないとできないこと**ってなんだろう……？」

「いくつか挙げられると思うけど、特に注目すべきなのは、**特定の管理主体を持たずに成立する**という点かな。通貨という点に注目すると、**中央集権的な通貨の発行元が存在しない**ことだといえる」

「そうか！だから、どこかの企業がSimpleBitcoinのネットワークに参加しているとして、そこが何らかの事情でSimpleBitcoinのネットワークから撤退してしまったとしても、SimpleBitcoinでやり取りする通貨とかポイントみたいなものは消滅しないってことだね。SimpleBitcoinに参加している人たちがほかにも残っている限り、自分が持っている通貨とかポイントみたいなものが継続して利用できるってことか」

「それじゃあ、これから創るものの機能を要約してごらん」

「『特定の企業や誰かに依存しないポイント移動の管理システム』ってところかな」

「上出来だ。それじゃあ、次は、そのようなシステムに参加してくれる有志のモチベーションについて考えてみよう」

「え、モチベーション？」

「『特定の企業や誰かに依存しないポイント移動の管理システム』に協力してサーバーを提供してもらうのに、まさかボランティアを前提にはできないよね？」

「なるほど、そういうことか。サーバーを提供して価値の移動を記録する仕組みを支える対価として、税金とか手数料みたいな形で少しずつお礼のポイントが受け取れる、っていうのではだめ？SimpleBitcoinのユーザー同士は、何らかのポイントをやり取りする。それを支える人たちは、そのポイントの一部をもらえる」

「すでに指摘したとおり、SimpleBitcoinではリアルなお金と連動した価値の移動は担保しないんだぞ」

「あ、そうだった。ポイントを換金できるわけじゃないから、そのままだと嬉しさがないか」

「お金ではないけれど、数えることが可能で、もらえると嬉しいもの、何かないかい？」

「Twitterのリツイートとか『いいね！』は、数が増えると嬉しいよなあ……、あ！」

「おや？何か思いついたようだね？」

「そうか、**『いいね！』**だ！『いいね！』だよ兄さん！」

「『いいね！』をどうするんだい？」

「誰かが面白かったり役に立つことをツイートしたときに、『座布団一枚！』みたいな感覚でユーザー間で自由にポイントをあげられるようにする。そして、それを支えるシステムの提供者は、貢献者として一定のポイントがもらえるようにする。みんなから集めた『お前スゲー！』を可視化するんだ！これだ、これだよ兄さん！承認の数はプライスレス！」

「ふむ、つまりサーバー提供者は、『お前スゲー！』という賞賛を受ける数がサーバーを運用するコスパに合わないと判断したら、SimpleBitcoinのネットワークから離脱するということだね？『お前スゲー！』というポイントを獲るとことには、実際のところ、どれくらいの価値があるんだろう？」

「そうだなー。獲得ポイント数のランキングが見える、っていうのはどうだろう？承認の数を競い合うゲームにするんだよ！地域別とか年代別とかでランキングを細かく分けたりして、分野ごとの上位をゲットしやすくすれば、SimpleBitcoinに参加し続けてくれるんじゃないかな」

「これからSimpleBitcoinのブロックチェーンを実装するわけだが、そのチェーンにはすべてのポイント移動情報が残される。だから、個々人が獲得したポイントが確認可能になるハズ、ということか」

「そうそう、よくわからないけど、そういう感じ！」

「だが、Twitterの『いいね！』は無制限にできる。暗号通貨のポイントとして他人に『いいね！』をあげるということは、自分が獲得した承認ポイントをわざわざ他人にあげるということになる。それにはどういう意味があるんだい？自分のランキングが下がるわけだよね？」

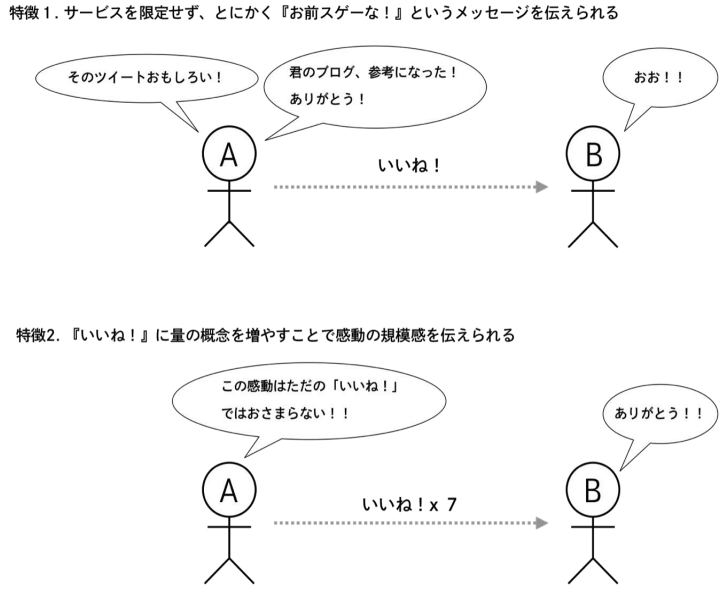
「『コイツはスゲーから、もっとみんなに知られるべき！』という意思表示、かな……。直接的にポイントを送るのであれ、手数料として間接的にサーバー提供者に渡すのであれ」

「なるほど。言い換えると、自分がそれまでに得た『いいね！』の総量がSimpleBitcoinのネットワークに対する影響力として捉えられるというわけか。自分のポイントを消費することで誰かの影響力を高める、そういうイメージかな？」

「うん。SimpleBitcoinのポイントは、『コミュニティの中で貢献している度合いを示すもの』と考えるほうがいいかもね」

「Twitterのように無制限に送れる賞賛ではないからこその嬉しさ、みたいなものもあるかもしれないね」

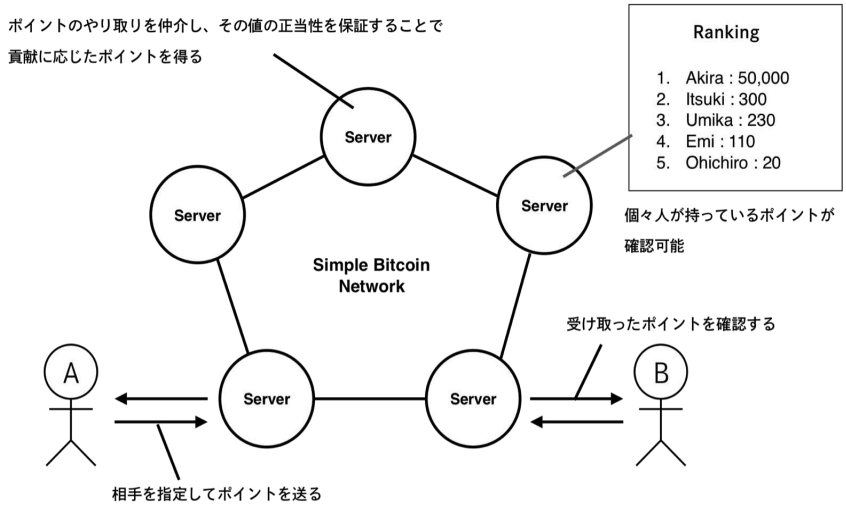
「あると思う。それに、世にある大半のSNSでは、常に1ポイントしか『いいね！』をあげられないでしょ？でも、『いいね！』にも程度の差があるじゃん？『いいね！ 100回押したい！』みたいな」



● 図 1.2　サービスを横断する形で、量的な概念を持った「いいね！」を提供する

「うん、悪くないアイデアだ。それじゃあ、ここまでの話を整理するために、図にまとめて描いてごらん」

「うーん……。図1.3みたいな感じかな？」



● 図 1.3　SimpleBitcoin サービスイメージ

「なるほど。次の3つの行為が可能な仕組みとしてSimpleBitcoinを作る、というわけだ」

1. 指定したユーザーに対してポイントを送ることができる
2. 自分が保持しているポイント残高を確認することができる
3. ランキングを確認することで上位ユーザーの獲得ポイント数を閲覧することができる

● SimpleBitcoin でできること

「よーし、作りたいものが見えてきたぞ」

「ランキングを確認できる人をSimpleBitcoinのユーザーに限定する必要もないから、厳密には1と2がこれから実装するメインと考えてよいだろうね」

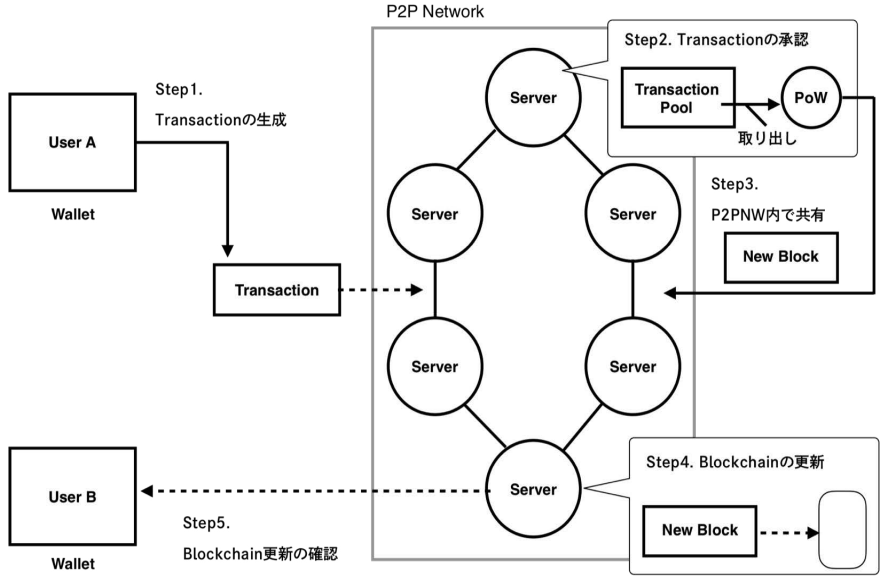
「ああ、そうだね。『こいつスゲーんだよ！』を広く知らしめるという目的からしても、ランキングを見られる人は多いほうがいいに決まってるもんな」

「いずれにしても、これから君が創るすべての部品は、この3つの仕組みを実現するためのものになる。よく頭に入れておくように」

「わかったよ、兄さん」

1.2 全体像を俯瞰する

「それでは次に、SimpleBitcoinの機能をもう少し細かく分解して見ていこう。図1.4を見てほしい」



● 図 1.4　SimpleBitcoin におけるデータの流れ

「えっと、これは？」

「クライアントとサーバーの役割分担という観点で、AさんからBさんにポイントを送付する際にどこでどんな処理が行われるのかを簡単に概念図としてまとめたものだ。左側にある**Wallet**と書かれたボックスがクライアントに相当すると思ってくれればいい。ちなみに、Walletは英語で財布のことだ。そのWalletを起点として、時計回りにデータの流れを図示してある。厳密な話をすればStep4はもっと複雑なんだが、全体のイメージをつかむという意味で、今はこのくらいの理解で十分だろう」

「**Transaction**とか、はじめて出てきたんですけど……」

「これから作ろうとしているSimpleBitcoinでは、『誰から誰に何ポイント移動するか？』といった、ポイント送付に関する情報をひとまとめにしたものが1つのTransactionになると思えばいい」

「つまり、まずStep1で、AさんからBさんへのポイント移動に関する情報が**P2Pネットワーク**上にあるサーバー群に通知されるってことか」

「そう。P2Pネットワーク上にあるサーバーのそれぞれでは、Transactionを受け取ると、それらを積み重ねる形で順番に保存していく。その保存場所のことを、Transactionをためておく場所とい

う意味で、**Transaction Pool**と呼ぼう。Step2では、このTransaction Poolから定期的に一定量のTransactionを取り出して、新規のブロックを生成する」

「ブロック……。ひょっとして、それが『ブロックチェーン』におけるブロックになるの？」

「そう考えていい。新規に生成されたブロックは、P2Pネットワークに参加しているすべてのサーバーで共有される（Step3）。個々のサーバーでは、自分が保持するブロックチェーンの中に共有したブロックを取り込む（Step4）。こうして、P2Pネットワーク上の全サーバー上のブロックチェーンが更新されるわけだ。ちなみに、**PoW**と書かれている部分の中身については後で説明するから今はあまり気にしないでくれて構わない」

「あのー……。Transactionがすべてのサーバーに届いてるなら、サーバーそれぞれで独立して新規ブロックを作れるんじゃないの？新規ブロックを共有するStep3って、意味あるのかな？」

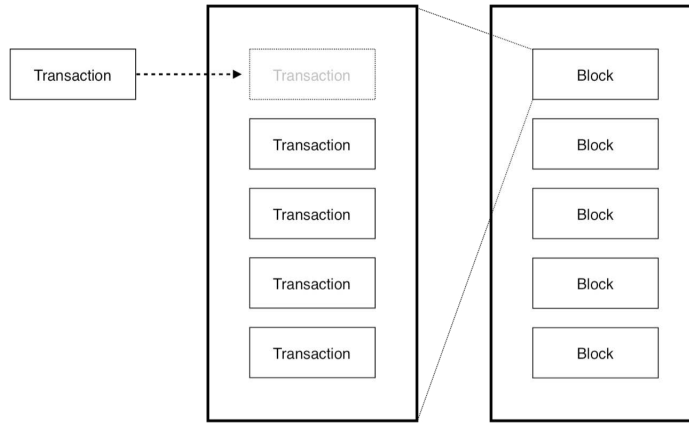
「個々のサーバーで勝手にブロックを作り、それを相互にチェックせずに放っておくような仕組みになっていたら、それぞれが勝手に自分の都合がいいようにポイントの移動データを改ざんできてしまう。P2Pネットワーク上のどのサーバーにあるブロックチェーンが本当に正しいものなのか判断できなくなる、という可能性は理解できるかい？」

「ああ、そういうことか。不正をするヤツがいる前提で考える必要があるってことだね」

「そう。ブロックチェーンの仕組みを使ってサービスを作るときは、**何か（あるいは誰か）を無条件で信用する必要のないシステム**という視点がとても重要になるので覚えておいてほしい。そのような性質のことを**トラストレス**なんていう言葉で表現することもある」

「トラストレス……。文字どおり『信用をおかない』いう意味か」

「不正防止のための工夫といった細かい話は、また後で詳しく説明する。ここで意識しておいてほしいのは、複数のTransactionが集まってブロックが作られ、そのブロックを時系列で順番に保存することでブロックチェーンが構成される、という考え方だ」



● **図 1.5　Transaction とブロック、そしてブロックチェーン**

「ということは、まず、Transactionをどんな形にするかを考えればいいのかな」

「そうじゃない。ブロックの中に入れるTransactionの中身についての仕様が確定していなくても、ブロックを作るためのロジックは考えられる。さらに、そのブロックを時系列で保存するだけだから、Transactionの仕様が確定していない状態でもブロックチェーンを作れる。図1.4を言い換えると、そういうことさ」

「そういうことか！つまり、図1.4の各Stepは、あくまでも利用シーンにおけるデータの流れを示しているってことだね。開発については、図1.4の順番に進めるわけではない、と」

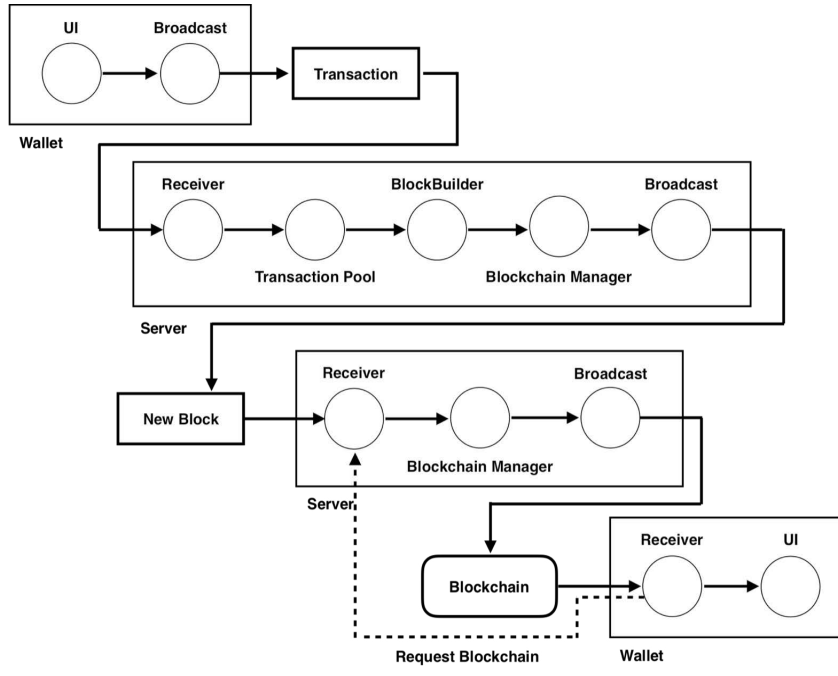
「そうだ。もっと具体的にいえば、Step1で全体に共有するデータなんて、最初のうちはダミーでも開発が進められる」

「なるほどなるほど」

「図1.4から読み取ってほしいのは、SimpleBitcoinの全体像だけではない。どこを先に作る必要があり、どこは後回しにしても大丈夫なのか、そういうイメージを感覚でいいので把握しておいてほしい」

1.3 これからの話の流れ

「では次に図1.6を見てほしい」



● 図 1.6　SimpleBitcoin の構成要素とデータの流れ

「左上のWalletがTransactionを送信するところを起点として、2つのサーバーを通じて新規ブロックがブロックチェーンに保存され、最後にまた別のWalletからサーバーに接続してブロックチェーンの更新分を確認する。見た目はちょっと違うけど、書かれている中身は図1.4に似ているね」

「そのとおり。図1.6には、図1.4で示した処理を実行するにあたってクライアントとサーバーそれぞれにどのような機能が備わっている必要があるか、そして、それらがどのように連携しているかを示してある。まだ抽象度の高い書き方をしているから、これから先に創るものと構造が完全に一致するわけではないが、それでも実装についてイメージするには十分だろう」

「メッセージを扱うフェーズに応じてクライアントとサーバーに求められる機能が違うから、そのときそのときで使われている機能にだけスポットを当てた図ってことだよね？もちろん物理的には離れた別物なのだろうけど」

「そう。クライアントとサーバーでそれぞれ2種類、合計4種類のソフトウェアがあるというわけではない。あくまでも、あるタイミングでどの機能が使われるのかに着目した図だ。この図1.6を

ベースにして、これからの開発の進め方、つまり、私が用意したレシピのカリキュラムを説明していこうと思う。ということで、このプリントを見てほしい」

『ゼロから創る暗号通貨』カリキュラム

1st Step（第 2 章）

目的は、インフラとなるP2Pネットワークを構築すること。図1.6で「Broadcast」や「Receiver」と示されている機能ブロックを作る。

Phase1

クライアントとサーバー、サーバーとサーバーの両方がどのような関係で接続し合っているかを考えながら、Coreノードと Edgeノードという概念を理解する

Phase2

P2Pネットワーク上で必要となるメッセージについて整理し、そのメッセージを管理するクラスを作成する。たとえば、新しくP2Pネットワークに参加したい場合にどのようなやり取りをすればよいか、逆に離脱したい場合にはどうするか、具体例に沿って考える

Phase3

Phase2で規定したメッセージを使って実際に動作するクライアントとサーバーを作り、接続してみる

Phase4

P2Pネットワークに拡張性を持たせることで、SimpleBitcoin以外の用途にも簡単に流用できるような設計について考え、既存のコードを改良する

2nd Step（第 3 章）

目的は、実装を通じてブロックチェーンの仕組みと考え方を学ぶこと。図1.6で「Transaction Pool」、「Block Builder」、「Blockchain Manager」、「New Block」と示されている機能ブロックを作る。

Phase1

基本構造を学ぶため、Transactionを受け取るたびにブロックを追加していくような簡単なブロックチェーンを作ってみる

Phase2

効率よくブロックチェーンを伸ばしていくためにはブロック生成のタイミングが重要であることを確認しながら、一般的な暗号通貨が採用している工夫を取り込み、Phase1の成果を改善する

Phase3

複数のサーバーでブロックを作る際に採用されるルール（PoW、Proof of Work）のアイデアについて説明し、それを実装に取り込んでみる。また、PoWを利用しても複数のサーバーで異なるチェーンが保持される可能性が存在することについて確認し、それを解決するためのアイデアとして一般的な暗号通貨で採用されているコンセンサスというルールについて説明する。そのメカニズムを、1st Stepで実装したP2Pネットワーク機能を組み込む形で、サーバーの実装に取り込む

3rd Step（第 4 章）

目的は、Walletの実装を通じて暗号通貨において肝となるTransactionの考え方を学ぶこと。図1.6で「Wallet」、「Transaction」と示されている機能ブロックを作る。

Phase1

通貨を送付する相手をSimpleBitcoin上で特定する仕組みについて学び、Walletの原型を実装する

Phase2

それまでの説明ではあえて触れてこなかった「やり取りする通貨はどこから出てくるのか？」という問題に着目し、持ってもいない通貨を無限に送付できてしまわないために一般の暗号通貨で採用されている工夫（UTXO、Unspent Transaction Output）について学ぶ

Phase3

自分のTransactionをサーバー側でブロックに取り込んでもらうためには手数料が必要となることについて学ぶ

Phase4

Phase2やPhase3で確認した内容を生かし、Transactionのデータ構造を考えながら実装してみる

Phase5

主にUI面からWalletに必要となる機能を整理し、それを実装してみる

Phase6

Phase4で実装したTransactionを、Phase5で実装したWalletで利用できるよう組み込む

4th Step（第5章）

目的は、これまでに作ってきた部品をすべて繋ぎ合わせることで、SimpleBitcoinを暗号通貨として実際に動作させること。

Phase1

WalletにP2Pネットワーク機能を組み込むことで、SimpleBitcoin上で動作可能なクライアントを完成させる

Phase2

2nd Stepで実装したサーバーに3rd Stepで実装したTransactionを組み込むことで、SimpleBitcoin上で動作可能なサーバーを完成させる

Phase3

通貨の送付が実装できているか、実際に動かして確認する

5th Step（第6章）

目的は、現在の実装に残された課題を認識すること。ここまでの実装では、簡易化を優先するために、一般的な暗号通貨よりもセキュリティという観点で弱い箇所がある。それらを確認していく。

6th Step（第7章）

目的は、SimpleBitcoinの暗号通貨以外への利用方法を考えてみること。SimpleBitcoinの可能性についてディスカッションする。

「最初にP2Pネットワークから作るんだ……」

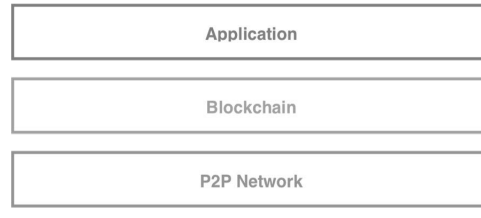
「そうだ。汎用性の高い機能を切り出して先に作る。その際には、上に載せるアプリケーションを差し替えることで容易に他の機能を実現しやすいものにする。これは、そういう思想に基づいたカリキュラムになっている」

「暗号通貨だけを作るわけじゃないってこと？」

「完成までの道のりはそこそこ長いからね。最後までコードとにらめっこしながらコツコツ作るより、節目節目で完成したところまで動作させて、どんなふうに使えるのか見られるほうがヤル気が続くだろう？それを考慮した順番が、いま見せたカリキュラムの方針ということさ」

「おお、なるほど！」

「こういう順番に作っていけば、ブロックチェーンを使った他のアプリケーションにも発展させやすいしね。図1.7のブロック図の関係性でいうと、下から上に向かって作っていくようなイメージになる」



● **図 1.7　SimpleBitcoin を構成する 3 つの機能ブロック**

「んー、最初に作るP2Pネットワークを使って暗号通貨とはまったく違うものも実現できる、ということ？なんかまだちょっとピンとこないけど……」

「たとえば、特定の相手に対してテキストを送るだけのメッセンジャーアプリなんかは、最初に作ることになるP2Pネットワークをベースにしてワリと簡単に実装できるだろう。そのようなアプリであれば、必ずしもブロックチェーンは必要ない。もちろん、LINEのように高機能なものを作ろうとすれば大変になるけれど」

「なるほど、『汎用性の高い機能』っていうのは、そういう意味か！順番的に先に作るものほどSimpleBitcoin以外での使い道があり、そういうものから順に作っていくから、そのつど単体で意味がある動作確認もしやすい、ということだね。そうやって少しずつ完成度の高まりを実感しながら進められるからヤル気が持続する、それが兄さんが用意してくれたカリキュラムというわけだね！」

「そうだ。ヤル気が持続するかどうかについては、私の仮説という域を出ないけれど、私自身はこの進め方が有効であると確信している」

「なるほどー！なんかやれそうな気がしてきたよ、兄さん！」

「では、さっそく始めていくとしよう！」

第 2 章

P2Pネットワーク：基盤作りから始めよう

本章では、SimpleBitcoinの基盤となるP2Pネットワークを実装します。ここで実装するP2Pネットワークでは、前章で登場したWalletとServerの機能を、EdgeノードおよびCoreノードという2種類のノードに割り当てます。2種類のノードの接続形態として考慮が必要になるのは、CoreノードとCoreノードの接続、および、CoreノードとEdgeノードの接続です。それぞれの接続を実現するプログラムを順番に書いていくので、ぜひ主人公たちと一緒にコードを書き、その動作をお手元のPC上で確認しながら読み進めてみてください。

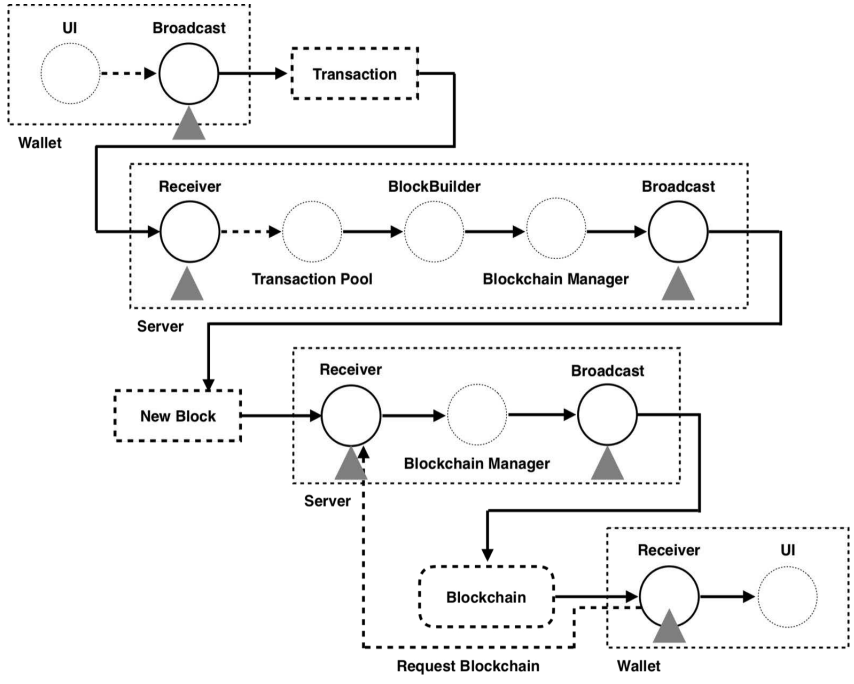
本章を読み進める際には次のようなポイントを意識してみてください。

1. CoreノードとEdgeノードの違い
2. P2Pネットワークが拡大していくステップ
3. SimpleBitcoinにおけるP2Pネットワークの役割範囲

「それでは、お待ちかねの開発に取り掛かろう。予告どおり、まずは暗号通貨に直接は関係しないP2Pネットワークを作ってもらうよ」

「なんでP2Pネットワークを作る必要があるんだっけ」

「暗号通貨の基盤となるのがブロックチェーンであり、そのブロックチェーンは中心が存在しない複数のサーバーが連携して、はじめてうまく機能する。そのようなサーバー間の連携は、P2Pネットワークにほかならない。全体像と照らし合わせると図2.1で示す箇所になる」



● 図 2.1　今回作る機能ブロックについて

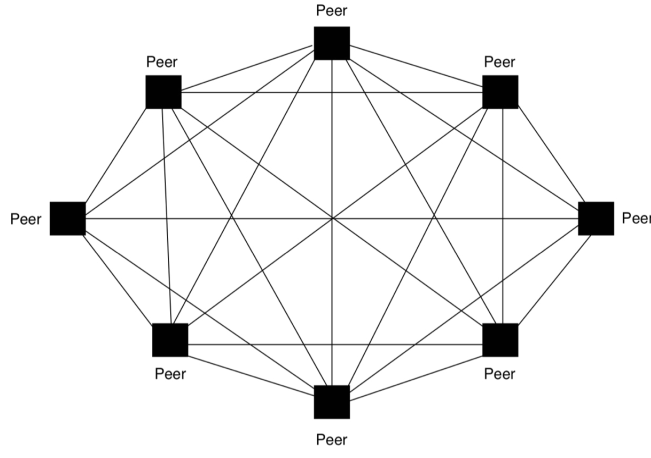
「いきなりP2Pネットワークを実装しろとかいわれても、あんまイメージわかないんだよなぁ」

「当然そうだろう。ここまでの説明では、ブロックチェーンにはWalletとServerという2つの登場人物が存在する、という話しかしていないからね。これから作りたいブロックチェーンを機能させるためのP2Pネットワークにおける両者の関係性や、連携方法については、まったく触れていない。だから、まずはそれらをはっきりさせる必要がある。それがこのカリキュラムのスタートポイントだ」

「よかった……。コードを書く前にもうちょっと説明があるんだ」

2.1 P2Pネットワーク

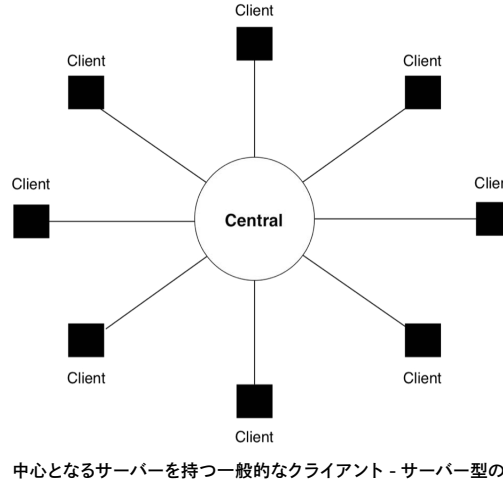
「念のために、P2Pの基本からおさらいしておこう。P2Pは、peer-to-peerの意味だ。**peer**には、『(能力などが) 同等の人』みたいな意味がある。相互に接続されているpeer同士がそれぞれ通信したり各自でデータを保持したりする図2.2のような形態のネットワークだ」



● **図 2.2　P2P ネットワーク**

「わざわざP2Pなんて名前がついているってことは、他の形態のネットワークもあるっていうことだよね」

「ある。というか、中心となるサーバーに他のクライアントが接続する図2.3のような形態のほうが一般によく見られる。こういう形態は、クライアント - サーバー型のネットワークと呼ばれている」



● **図 2.3　中心となるサーバーを持つ一般的なクライアント - サーバー型のネットワーク**

「うん。それはさすがに知ってた。暗号通貨では、中央サーバーがないP2Pが選択されている、ってことだよね」

「そういうことになる。どこかのpeerが離脱したり、あるいは新しいpeerが増えたり、そういう変化に柔軟に対応できるようにすることで、特定のプレイヤーに依存することなく成立し続けるネットワークが暗号通貨には必要不可欠だからね」

「なんか、P2Pネットワークに繋がるpeerのことをサーバーって言っているけど、暗号通貨の登場人物はWalletとServerの2つだったよね。Walletはどういう位置づけになるの？」

「いい質問だ。P2Pネットワークにおいて、いわゆるサーバーにもクライアントにもなり得るものを『peer』と呼ぶことにする。その定義からするとWalletはクライアントの役割しか果たさないわけだから、『peer』ではない。Walletは、新しいTransactionを生み出したときにだけ、Server群にそれを伝えられれば十分。そう考えると、常時P2Pネットワークに接続しておく必要はない」

「なるほどー。Walletは『P2Pで繋がっているものではない』なら、しばらくは忘れていてもいいのかな」

「単にP2Pネットワークを実装することが目標なら、その認識で正しい。しかし、最終的に作りたいものはP2Pネットワークそのものではなく、暗号通貨の基盤となるブロックチェーンだ。WalletからServerに対してTransactionが送信されるというイベントがあって、はじめてブロックチェーンに更新が発生するのだから、相互接続するServerだけ作っても暗号通貨としては機能しない」

「たしかに」

「よって『peer』とはいえないWalletについてもP2Pネットワークの一部としてまとめて考えるほうがいいだろう。とはいえ、WalletとServerは同格の存在ではなく、役割が異なる。そのことを明確にしたほうがいいので、便宜上、Walletを**Edgeノード**、Serverを**Coreノード**と呼び分けることにしよう」

「あえて名前を変えて呼ぶのはなんで？」

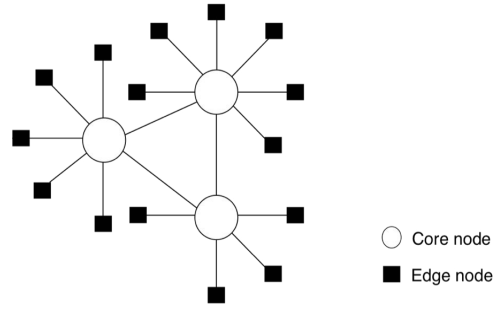
「P2Pネットワークは**暗号通貨以外の用途でも利用が可能**だよね？ WalletとServerは暗号通貨用途を実現するアプリケーションとしての名前であって、基盤としてのP2Pネットワークを実現するための呼び名としては適切ではない」

「ああ、なるほど。通信に特化した部分を切り出して別に名前を与える、ってことだね」

「そのとおり。次は、このCoreノードとEdgeノードについて詳しく見ていくことにしよう」

2.2 CoreノードとEdgeノード

「CoreノードとEdgeノードの関係を図にすると、図2.4のようになる」



● 図 2.4　Core ノードと Edge ノードの関係

「Coreノード同士がそれぞれ相互に接続してて、各Coreノードにぶら下がるような形で複数のEdgeノード群が接続されている、って感じだね」

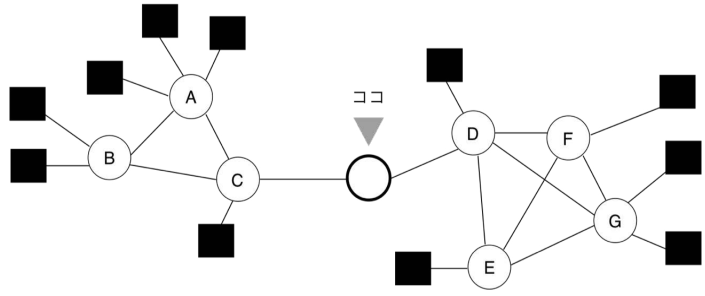
「そう。EdgeノードはP2Pネットワークに対し、必要なときにだけ新規のTransactionを送信する。自身に関連する値が変更されているかを定期的に確認するだけでよいので、いずれかのCoreノードを選んで接続するだけだ」

「この図を見るまで、Transaction ってのは、Edgeノードから全部のCoreノードに対して送信されるものかと思ってた」

「そういう形にするのもアリだとは思う。けれど、Coreノードの数が3桁〜4桁と増えていった将来のことを考えると、全部のCoreノードに対してEdgeノードからTransactionを一括で送るのは効率が悪そうだよね」

「そりゃそうか」

「余談だけど、その意味では、CoreノードがP2Pネットワークに参加している全部のCoreノードを記憶している、というのも効率が悪い。そこで一般的には、図2.5のように、Coreノードのネットワーク同士を繋ぎ合うような、『リレーノード』などと呼ばれるものを用意する」



● 図 2.5　リレーノード

「ふむふむ、なるほど。リレーノードか。また面倒そうなものが増えたな……」

「まあ、SimpleBitcoinでは話を単純化するために、今回はCoreノードとEdgeノードの２種類だけで話を進めていこう」

2.2.1　接続するCoreノードの選択

「とはいえ、いくら話を単純化するといっても、Edgeノードの接続先を１か所に絞るような構成にすれば、すべてのEdgeノードが１つのCoreノードに集中してしまうよね。それはさすがにあんまりだ。Edgeノードの接続先が１か所に集中することなくいい感じにバラけるようにしたいけど、それにはどうすればいいかなー」

「ちゃんとやるなら、Coreノード側というかP2Pネットワーク側で、接続されているEdgeノードの数といった何らかの基準に応じて、登録を要請してきたEdgeノードの接続先を割り振るような機能があるほうがいいんじゃないかな」

「たしかに。そっちのほうがちゃんとしてる。でも、けっこう複雑になりそうだし、ここは日和ってEdgeノード側で接続したいCoreノードを選択する程度にしておきたい……」

「それは悪くない判断だよ。では、ユーザーが自分で接続先のCoreノードを選択するとして、その候補はどうやって入手するとよいだろうか？」

「気持ち的には手動で、と言いたいところだけど、さすがにCoreノードの一覧をダウンロードしてきてユーザーに選ばせるくらいのことはしたほうがいいかなぁ」

「その機能を入れるにしても、肝心のCoreノードの一覧を取得する先は手動で入力するか、あるいは、あらかじめソースコードの中に固定値を入れておく必要はあるよね」

「あー、たしかに。だったら、もう最初からCoreノードの候補をいくつか仕込んどけばよくない？」

「SimpleBitcoinはゼロからの立ち上げだから、最初からがっつりCoreノードが揃ってる前提はないよ」

「そうだった。となると、どこか特定の場所にCoreノードを登録してもらうことにして、そこから一覧をもらって接続先のCoreノードを選ぶような方式にするか……」

「せっかく『特定のプレイヤーに依存しない』ようなブロックチェーンを作ろうとしてるのに、

いきなり特定の何かに依存する前提にするのは、ちょっと悲しくないかい？」

「うわー、たしかにそのとおりだー。あー、どうすりゃいいんだー」

「あんまり考えすぎるとドツボにハマるし、導入初期はシンプルにしておいて、それなりにネットワークが立ち上がった後でCoreノードの選択方法を変える、くらいの気持ちでもいいんじゃないかな」

「つまりどういうこと？」

「接続先の入力は、とりあえず手動でいいんじゃないか、ってこと。ユーザーが接続のために必要な情報を事前に入手する感じだよ」

「えっ？そんなんでいいの？」

「まぁ、完全なビットコインではなく、SimpleBitcoinだしね。Edgeノードが利用可能な接続先情報を管理するための中央サーバーを前提にして、そのサーバーを運用し続ける負担を背負うことに比べれば、ずっとマシなんじゃないかな？」

「いわれてみれば、そりゃそうか」

2.2.2　接続先の Core ノードを信用できるか

「でも、Edgeノードの接続先って、本当に1つのCoreノードでいいのかな？」

「なんでそう思うんだい？」

「前に『トラストレス』って話があったよね。トラストレスという観点で考えると、接続先のCoreノードを無条件に信頼するのはダメなんじゃないかって気がするんだけど」

「いい視点だ。セキュリティの話だね。間違いなくいい視点ではあるんだが……」

「……だが、なに？」

「それを真面目に解決しようとすると、まったくシンプルではなくなってしまうんだよね。たとえば、Edgeノードが接続先のCoreノードから取得できるブロックチェーンの情報に、嘘が混じっている可能性について心配するとしよう。君が疑問に思ったとおり、接続先が1つのCoreノードしかないと、この可能性を排除できない。複数のCoreノードから情報をもらう形にして、それらを比較できるようにすれば、ひとつの解決策になる。でも、その複数のCoreノードの情報ってのは、どうやって入手するんだろう？」

「Coreノードが信頼できない前提なら、そのCoreノードからもらうP2Pネットワーク上の他のノードの情報だって、やはり信用しちゃいけないわけか。かといって、たとえばどこかの掲示板で複数のCoreノードの情報を探してきたとして、それが信用できるかといえば、そんなわけもない、と」

「そのとおり」

「だったら、そもそも、CoreノードにEdgeノードが接続するっていう方式を取らなければいいのでは。参加者が自分でCoreノードを用意する前提にしてしまうのがいいんじゃない？」

「一見すると、そう思えるよね」

「その言い方だと、それじゃダメってこと？」

「さすがに、自分で立てているCoreノードなら信頼できる。それは正しい。ただ、自分が立てるCoreノードは、どこか他のCoreノードに接続することで、はじめてP2Pネットワークに参加できる。じゃあ、そのときに接続する先のCoreノードの情報は、どうやって信用するに値すると判断できるだろう。自分が立てるCoreノードにしたって、結局はP2Pネットワークに接続するにあたって同じような問題に直面するだろう？」

「たしかに、Coreノードを自前にしたからといって、何も解決してないな」

「そう。いまはとりあえず、**『Coreノードとして信頼可能なものの一覧が、コミュニティなど何らかの方法で定期的にメンテナンスされている』**といった前提を置くことにしよう。何らかの方法でメンテナンスされた情報を信頼してるという時点で、厳密にいうとこれは『トラストレス』じゃない。しかし、SimpleBitcoinではシンプルさを優先しよう。実をいうと、オリジナルのビットコインにしたって、やっていることはこれと大差ない」

「そうだね。Coreノード同士で認証し合うみたいな話になると、どんどん複雑化していって、いつまでたっても暗号通貨の話にまで進まなそうだ」

「Coreノードが別のCoreノードを何らかの方法で信頼できるという前提にしておけば、Edgeノードが信頼できるCoreノードがないから自分でCoreノードを立てる、といった議論にも意味がなくなる」

「そりゃそうだね」

「念のため強調しておくと、各参加者が自分でCoreノードを運用すること自体に意味がないわけではない。セキュリティを理由にCoreノードを立てることに意味がないといっているだけだから、そこは間違えないように」

「Coreノードを立てる意味がない、なんて話になったら、Serverを提供する人にとって価値が見出せるという、SimpleBitcoinのそもそもの前提が成立しなくなるものね。ちゃんと覚えてるよ」

2.2.3 Coreノードと Edgeノードの仕様まとめ

この節では、本書で実装する暗号通貨SimpleBitcoinの基盤となるP2Pネットワークについて、次のような仕様を確認しました。

1. EdgeノードはCoreノードにぶら下がる形で存在する
2. 1つのCoreノードに対して複数のEdgeノードが接続可能とする
3. Edgeノードが接続する先のCoreノードは、あらかじめ何らかの方法で用意されたCoreノードの一覧から、Edgeノードのユーザーが自分で選択して入力する
4. Coreノードを新規にP2Pネットワークに接続したい場合も、3と同じCoreノードの一覧から、Coreノードのユーザーが自分で選択して入力する

次節では、これらの仕様を前提として、CoreノードとEdgeノードを実装していきます。

2.3 P2Pネットワークの実装

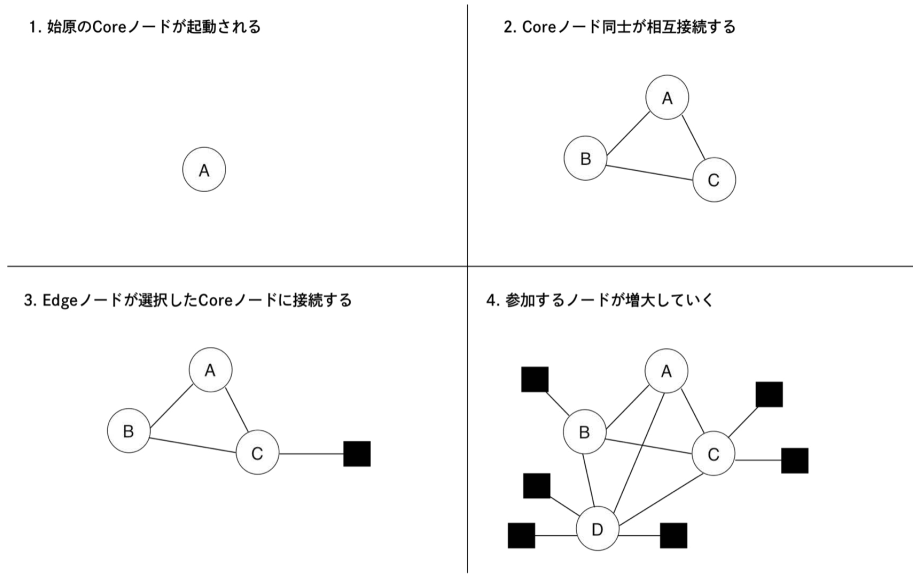
2.3.1 プロトコルを考える

「実装にあたり、まずは、Coreノードと Edgeノードが接続されて図2.4のようなP2Pネットワークの形になるまでに、どういうステップを踏めばいいかを考えてみよう」

「えっと、とりあえず、拠り所となるCoreノードが存在していることを前提とするね。そのCoreノードに、別のCoreノードが接続する。そういう接続が次々に起こって、Coreノード同士のネットワークができる。その中で信用可能と思われるCoreノードをEdgeノードが選択し、そこに接続する」

「そうだね。じゃあ、それを絵に描いてみよう」

「図2.6のような流れになると思う」



● 図 2.6 P2P Network の成長過程

1. 拠り所となるCoreノードを起動する
2. 1で起動したCoreノードに対し、別に起動したCoreノードが接続していくことで、Coreノードのネットワークが生まれる
3. 2で形成されたネットワークから、Coreノードを1つ選択し、Edgeノードが接続する
4. 2および3を繰り返すことで、CoreノードおよびEdgeノードからなるネットワークがどんどん大きくなっていく

「いいんじゃないかな。では、この図2.6を見ながら、CoreノードとEdgeノードがそれぞれどんな機能を持っていればいいかを考えてみよう。まず、2つめのステップで必要になる機能は何かな？」

「すでに起動済みのCoreノードに対して接続を依頼し、それを受け付けてもらう必要があると思う」

「それだけ？」

「えっと、Coreノードは接続する相手がどんどん増えていくから、最初のCoreノードに接続が完了したタイミングで、そこに接続されている既存の他のCoreノード群の情報を教えてもらう必要があるかな」

「厳密にいえば、初回接続時だけじゃなく、Coreノードが追加されるタイミングで常に通知してもらう必要があるけどね」

「なるほどたしかに」

「ほかにはどうだろう？」

「あとは、接続依頼と対になる形で、ネットワークから離脱したい場合にそれをネットワークに伝える機能も必要になると思う。離脱するときにお行儀よく離脱メッセージをくれる相手ばかりとも限らないだろうから、定期的に接続状況の確認ができるような機能も必要かなぁ。Ping的なやつ」

「Pingに返事がなかったらP2Pネットワークから離脱したものとして扱うことにしよう、というわけか」

「そうそう」

「じゃあ、次は3つめのステップ、Edgeノードの追加について考えてみようか」

「これは簡単だね。CoreとEdgeの区別をつける必要はあるけど、基本的にはCoreノードとほぼ同じで、接続要求と離脱要求が処理できればいい。他のCoreノードに接続されているEdgeノードについては情報を知る必要ないから、それくらいかな。念のため、Edgeノード自身が接続しているCoreノードの生存確認と、死んでしまっているときに他のCoreノードにぶら下がり先を引っ越せるようにCoreノードの一覧は持てるようにしておいたほうがいいかも」

「Coreノードの生存確認の方法と、Coreノードの一覧をもらう方法については、CoreとEdgeで区別をつける必要はないだろうね」

「うん。以上をまとめると、CoreノードとEdgeノードに必要な機能はこんな感じになると思う」

1. Coreノードに対する接続リクエストと、それに応じたCoreノード一覧への登録処理
2. 離脱リクエストとCoreノード一覧からの削除処理
3. Coreノード一覧のリクエストと、それに対するレスポンス
4. P2P Network上のCoreノードの接続状況の確認
5. Edgeノードからの接続リクエストの処理と、それに対するレスポンスとしてCoreノード一覧の送信
6. Edgeノードからの離脱リクエストの処理

● **Core ノードに必要な機能**

1. Coreノードに対する接続リクエストの送信
2. Coreノードへの離脱リクエストの送信
3. Coreノードから送信されたCoreノード一覧の処理

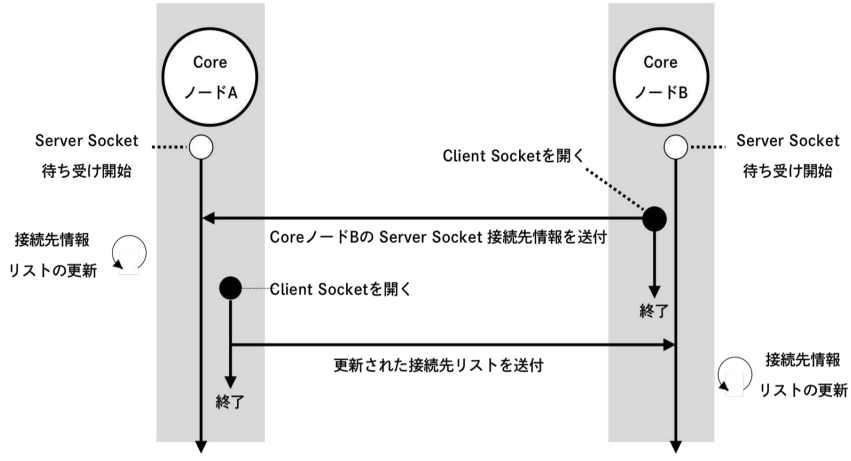
● **Edge ノードに必要な機能**

「いいね。もちろん、接続した後でメッセージを送信しあうためにP2Pネットワークを構成するわけだから、このほかにメッセージ送受信機能が必要になる。とはいえ、接続管理に関するものとしては、ここにまとめた機能で十分だろう。次は、これらの機能を使ってノードをどう実装すればいいか、少し詳しく考えてみよう」

2.3.2 Core ノードの動作

「Core ノードの動作から考えるね。まずは、始原となる Core ノードに自分の接続先情報を送信し、それを登録してもらうことによって、他のノードからも接続可能な Core ノードとなる。それから、Edge ノードなり Core ノードなりが接続してくるのを待って、そのときがきたら送られてきた情報に応じて処理を実行して返す、という感じかな」

「だいたいあってる。Core ノードの動作を絵としてまとめると、図2.7のようになる」



● 図 2.7　Core ノード同士が接続するまでの流れ

「ん？ソケット？」

「そう。各ノード間のデータのやり取りにはソケット通信を使うことにする。**Server Socket**と**Client Socket**、この2つのイメージを掴むために、まずは簡単な通信プログラムを書いてみるところから始めてごらん」

「図2.7によると、Server Socketは、いったん開いたら他のノードからのデータを受け取るためにずっと口を開けておくんだね。一方、Client Socketは、送信が必要なデータがあるときだけ暫定的に開き、送信が終わって役目を果たしたら使い捨てられる。そんなイメージであってる？」

「そのとおり。というわけで、まずはServer Socketを開いて待ち受ける側の簡易サーバーと、その待ち受けているServer SocketにClient Socketを使って接続してデータを送信する簡易クライアントを作ってみよう」

「開いているソケットにデータを送信すればいいだけだから、クライアント側は簡単だね！せいぜい気をつけることがあるとすると、文字列の場合そのまま送信はできないからUTF-8でエンコードすることくらいかな……」

● リスト 2.1　client.py

```
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# ここは環境に合わせた接続先を入れる
my_socket.connect(('10.1.1.27', 50030))
my_text = "Hello! This is test message from my sample client!"
my_socket.sendall(my_text.encode('utf-8'))
```

「次はサーバーのほうを考えてみよう」

「うーん、クライアントから接続するのだから、クライアントにわかる接続先の名前が必要だよな。ソケットプログラムの解説にあるgethostname()を使えばいいのかな」

「利用するユーザー側の環境制限が少ないほうがいいから、IPアドレスで接続できるようにしたほうがいいんじゃないかな。ちなみに、自分で自分のIPアドレスの確認ができる人には無用の機能だが、Googleの提供しているパブリックDNSサーバーを指定してs.connect(("8.8.8.8", 80))のようにすると、s.getsockname()[0]でIPアドレスが取得できる」

「なるほど。これから先、複数のクライアントが接続してくることも考慮して作っとくか」

「実験目的で会社の中といった閉じられた環境で使いたい場合など、Googleのサーバーに接続できない場合も考慮して、IPアドレスの手動入力も可能にはしておくべきだろうね」

「たしかにそれはそうかも」

● リスト2.2　server.py

```
from concurrent.futures import ThreadPoolExecutor
import socket
import os


def __handle_message(args_tuple):

    conn, addr, data_sum = args_tuple
    while True:
        data = conn.recv(1024)
        data_sum = data_sum + data.decode('utf-8')

        if not data:
            break

    if data_sum != '':
        print(data_sum)


def __get_myip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(('8.8.8.8', 80))
    return s.getsockname()[0]


def main():

    # AF_INET : IPv4 ベースのアドレス体系を使うということ
```