RPCアプリの作成

@DG Lab - Karl-Johan Alm

このセッションについて

0104. RPCアプリの作成：注意点とベストプラクティス（前半）

0202. RPCのコードを触ろう

0302. RPCアプリの作成：注意点とベストプラクティス（後半）

ダウンロード版：http://bc-2.jp/materials/0103_RPCアプリ-na.pdf

Slackのチャネルについて

RPCアプリ作成の相談や質問の為のチャネル：**#rpc-app**

① CHANNELSをクリック

② **#rpc-app**をクリック

③ Join Channelをクリック

Agenda

- 環境
- Confirmations, reorgs (reorganizations)
- 脅威モデル
- RPCアプリの流れ（invoice、支払、問題点）
- 演習（Task 1～3）

環境

ポート

Bitcoin Core：

① P2P（main netは8333、BC2では8444）

② RPC（main netは8332、BC2では18442）

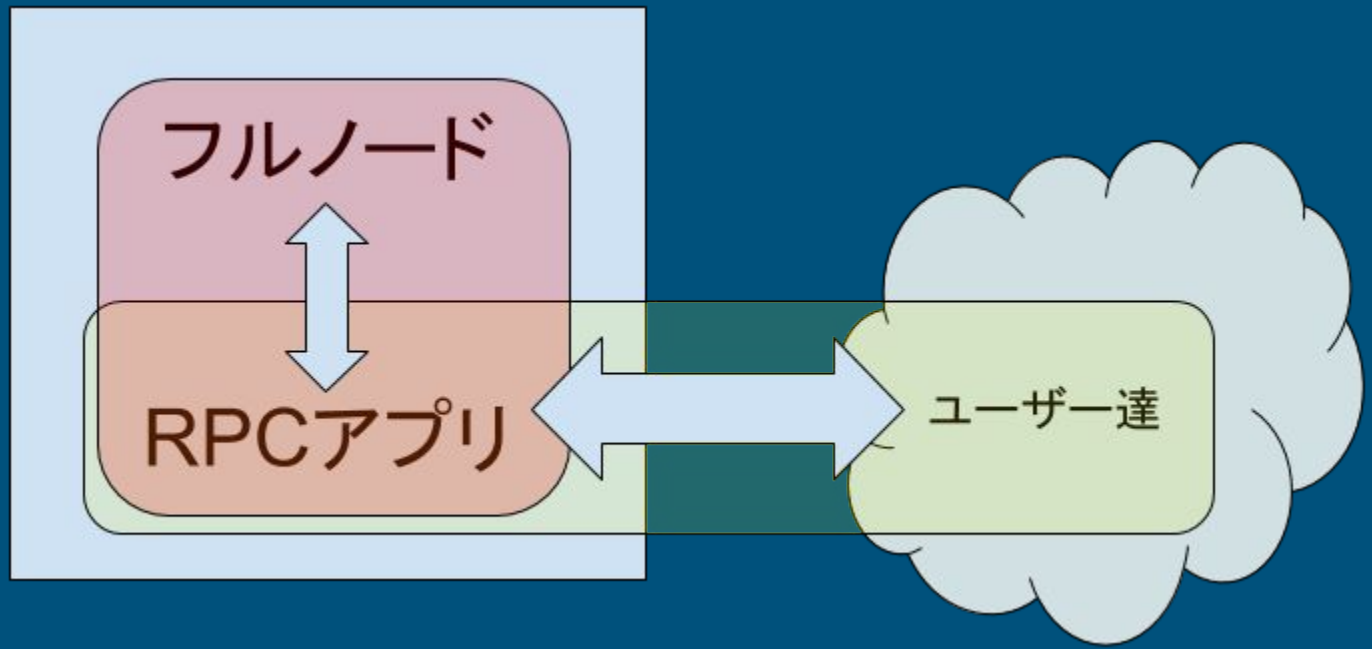
RPCとP2Pとの違い

RPCアクセス＝ウォレットに無制限のアクセス

アクセスを与える＝自分の財布を相手に渡す！

「自分のアプリをApp Storeに出して、アプリからRPCで繋いで…」という考えは、無論、駄目！

正しいRPCアプリの構成法



正しいRPCアプリの構成法

① フルノードを安全な状態に設定する。

② そのノードと接続するRPCアプリを作成し、
ノードと同じサーバーに入れる。

③ そのRPCアプリにサーバー外と接続する機能を追加する。

④ ユーザーが使うアプリを③を通して作成する。

正しいRPCアプリの構成法

① フルノードを安全な状態に設定する。

② **そのノードと接続するRPCアプリを作成し、ノードと同じサーバーに入れる。**

③ そのRPCアプリにサーバー外と接続する機能を追加する。

④ ユーザーが使うアプリを③を通して作成する。

RPCの使用制限

・原則、localhost以外のホストからのアクセスは却下。

・RPCアプリが認証情報を指定する方法は以下の二つ：

① 設定ファイル（bitcoin.conf）に記述して、
そのディレクトリを指定する：`-datadir=X`

② 直接指定する：`-rpcuser=Y -rpcpassword=Z`

RPCアプリ

ここでは②、つまりrpcuser/rpcpasswordを使う。

そのため、bitcoin.confを作る必要がある。

※再掲（デフォルト）

Mac: ~/Library/Application Support/Bitcoin/bitcoin.conf

*nix: ~/.bitcoin/bitcoin.conf

RPCアプリ

bitcoin.confの内容：

```
rpcuser=user
rpcpassword=password
```

注意：このイベントでは「password」というパスワードを使うが、
通常は**安全なパスワードにしよう**。

RPCアプリ

保存したら、bitcoindを再起動する必要がある。

① ./bitcoindを実行したターミナルで**^C**（Ctrl+**C**）を押す

　見つからなかったらkillall -9 bitcoindで全てのbitcoindが消える。マイニングしていると停止するまでに時間が掛かる可能性がある

② ./bitcoind -printtoconsoleをもう一度入力する

RPCアプリ

シンプルなCLIオンリーのシステム。

実装する機能：

・invoiceの作成

・支払の確認

・ダブルスペンド、reorgなどの対策

RPCアプリ

・環境はnode.js

・bcrpcというbitcoin-RPCラッパーを使う

・bitcointestというbitcoinテストスイートを使う

・あまり時間がないので、bitcoinに関係ないところを全て入れてあるバージョンを以下に用意した：

bc2/conference/rpc-cli

RPCアプリ

必要なライブラリのインストールとRPCアプリの起動:

```
$ cd bc2/conference/rpc-cli
$ npm install
$ ./rpc-cli
Available commands:
[...]
```

RPCアプリの開発ツール（npm test）

テストスイートの起動：

```
$ npm test
> rpc-cli@1.0.0 test ../bc2/conference/rpc-cli
> mocha tests.js
  db
    ✓ can be connected to
  bitcoind
    ✓ can be found
  bitcointest
    ✓ configures
```

…

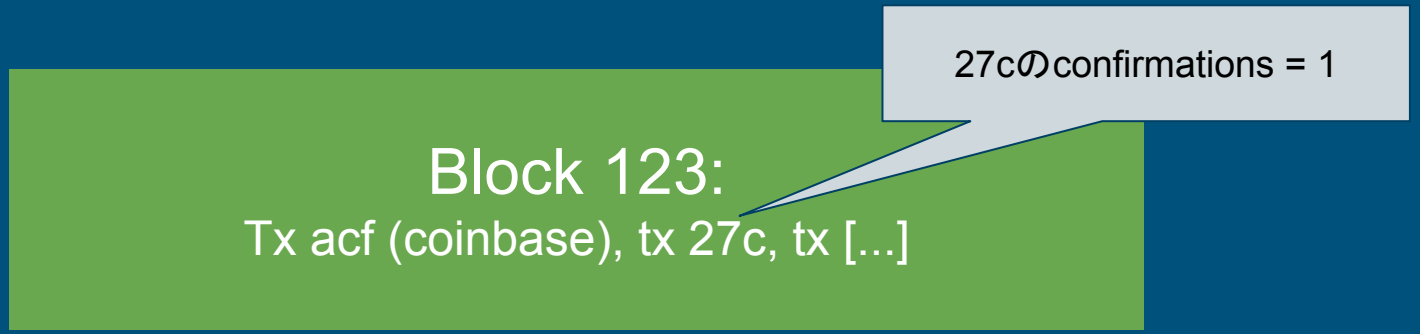
Confirmations, reorgs

Confirmations

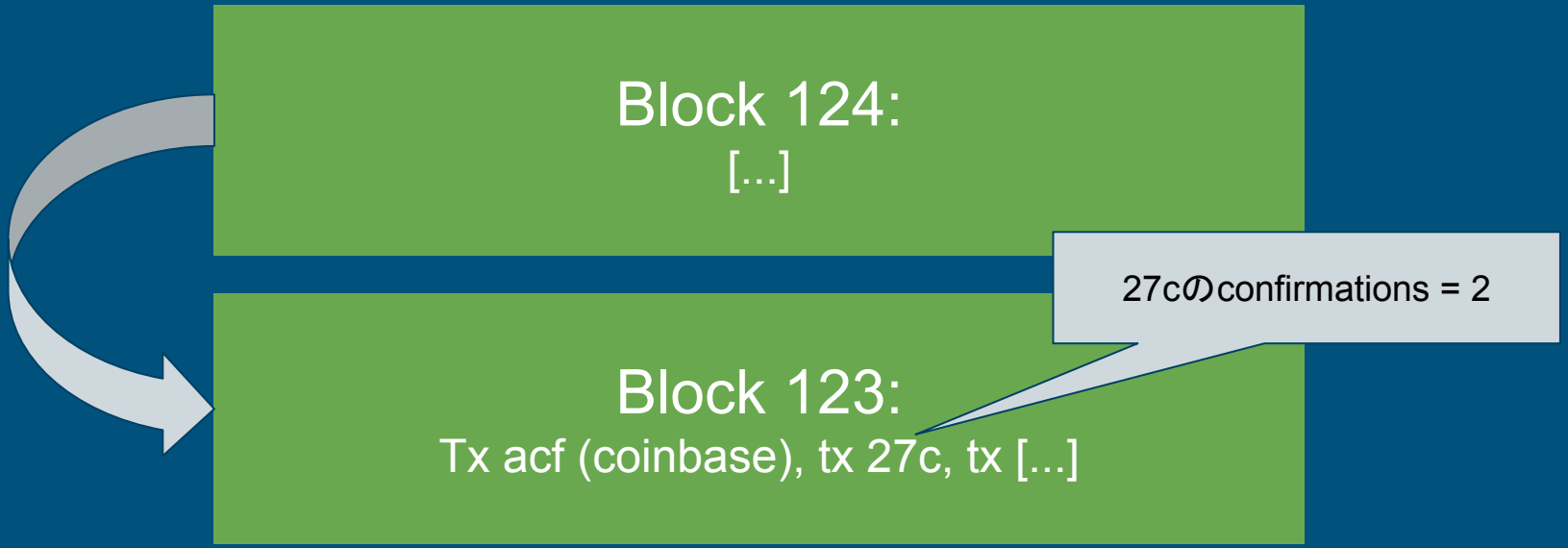
Block 123:

Tx acf (coinbase), tx 27c, tx [...]

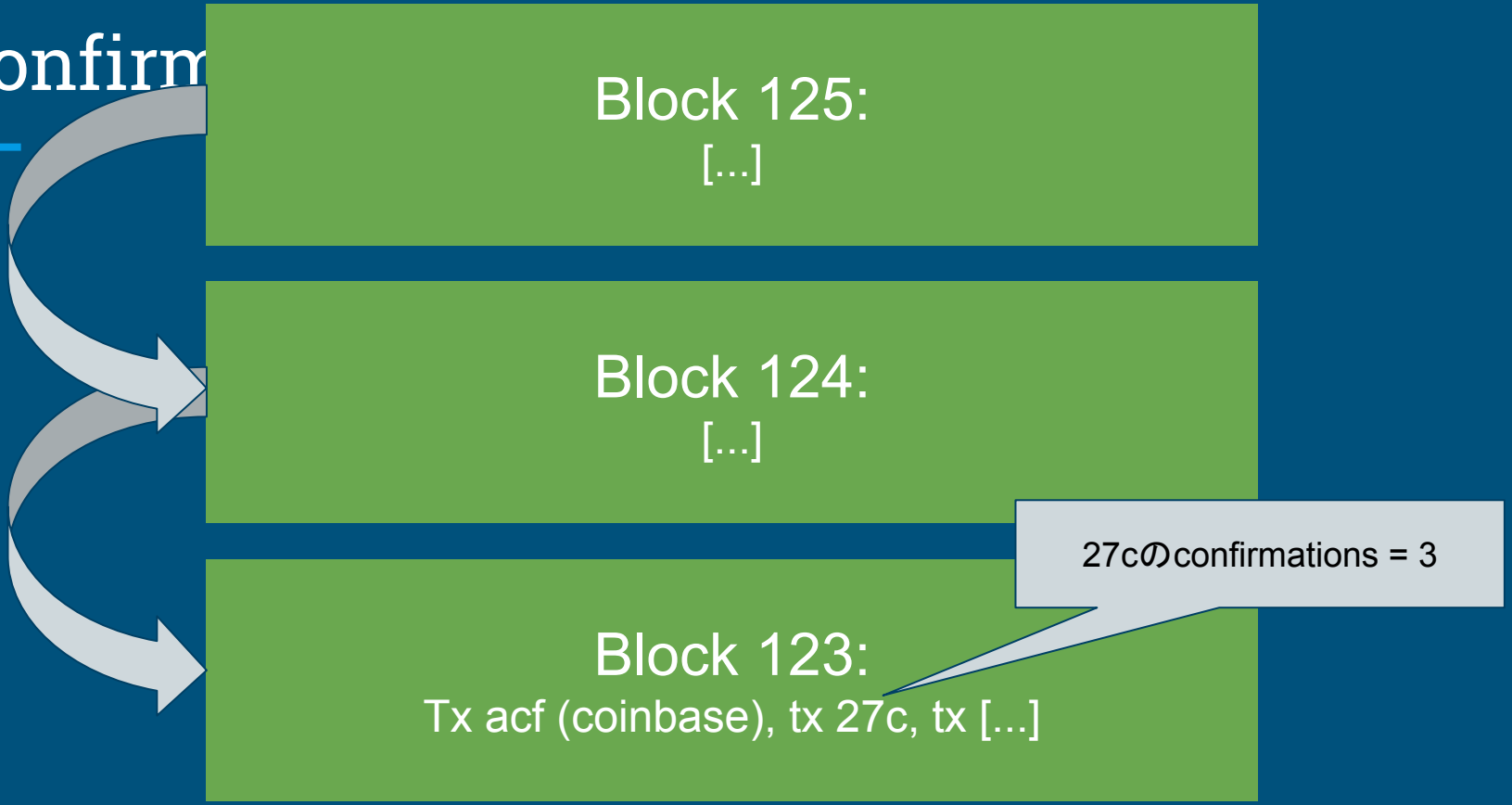
Confirmations



Confirmations

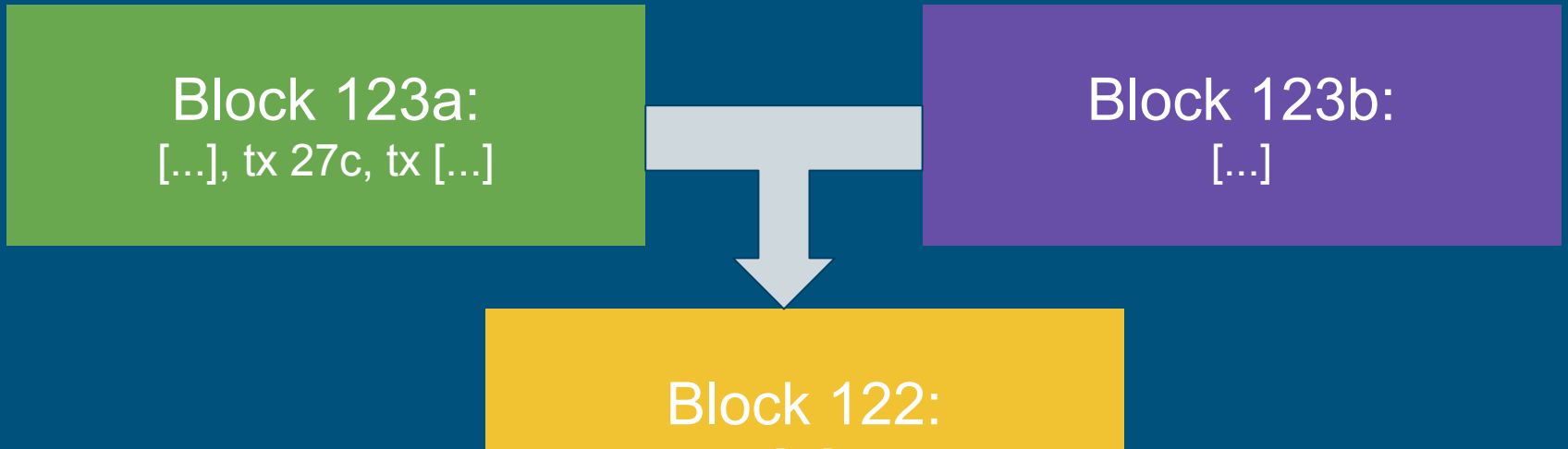


Confirm

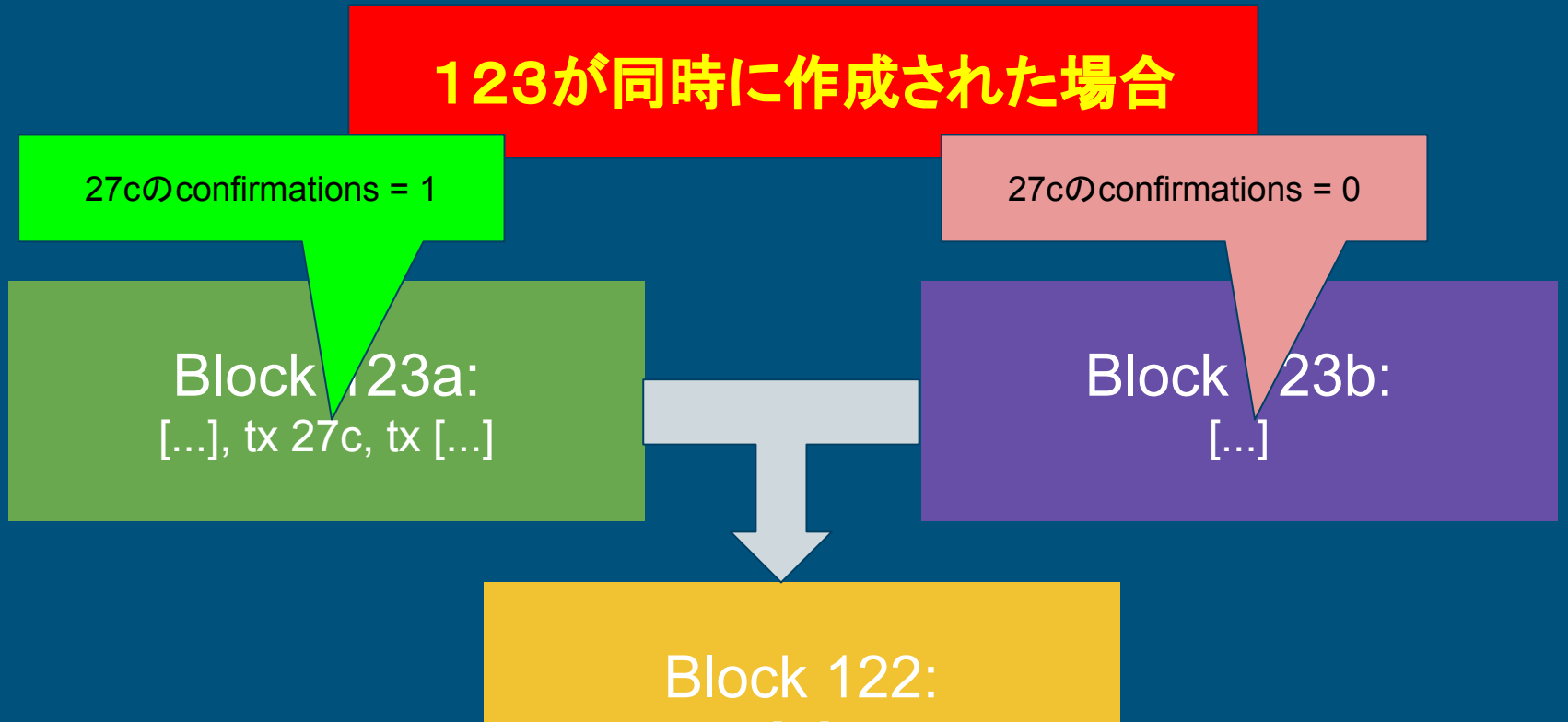

Block 125:
[...]
Block 124:
[...]
Block 123:
Tx acf (coinbase), tx 27c, tx [...]
27cのconfirmations = 3

Reorg

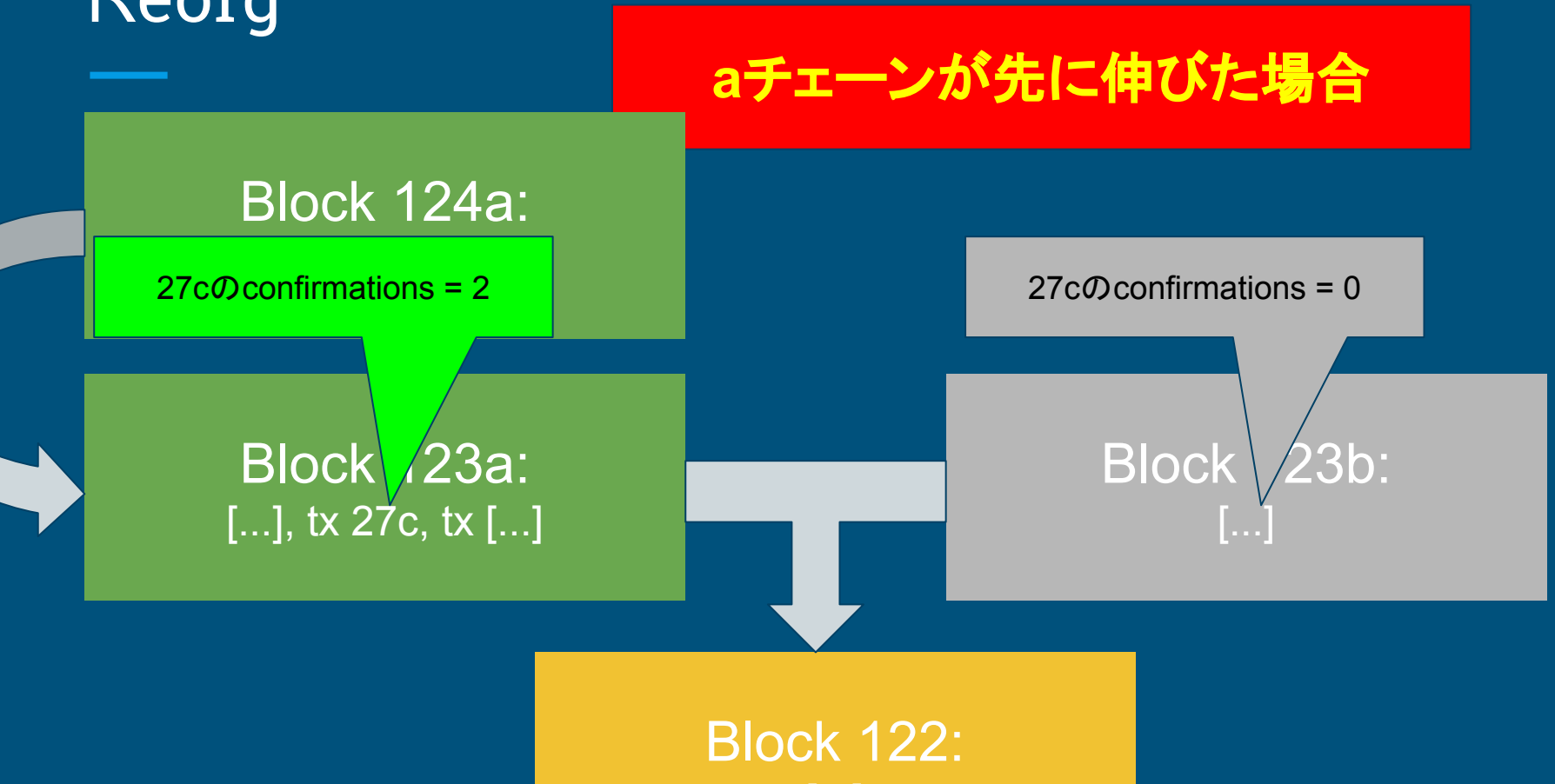
123が同時に作成された場合



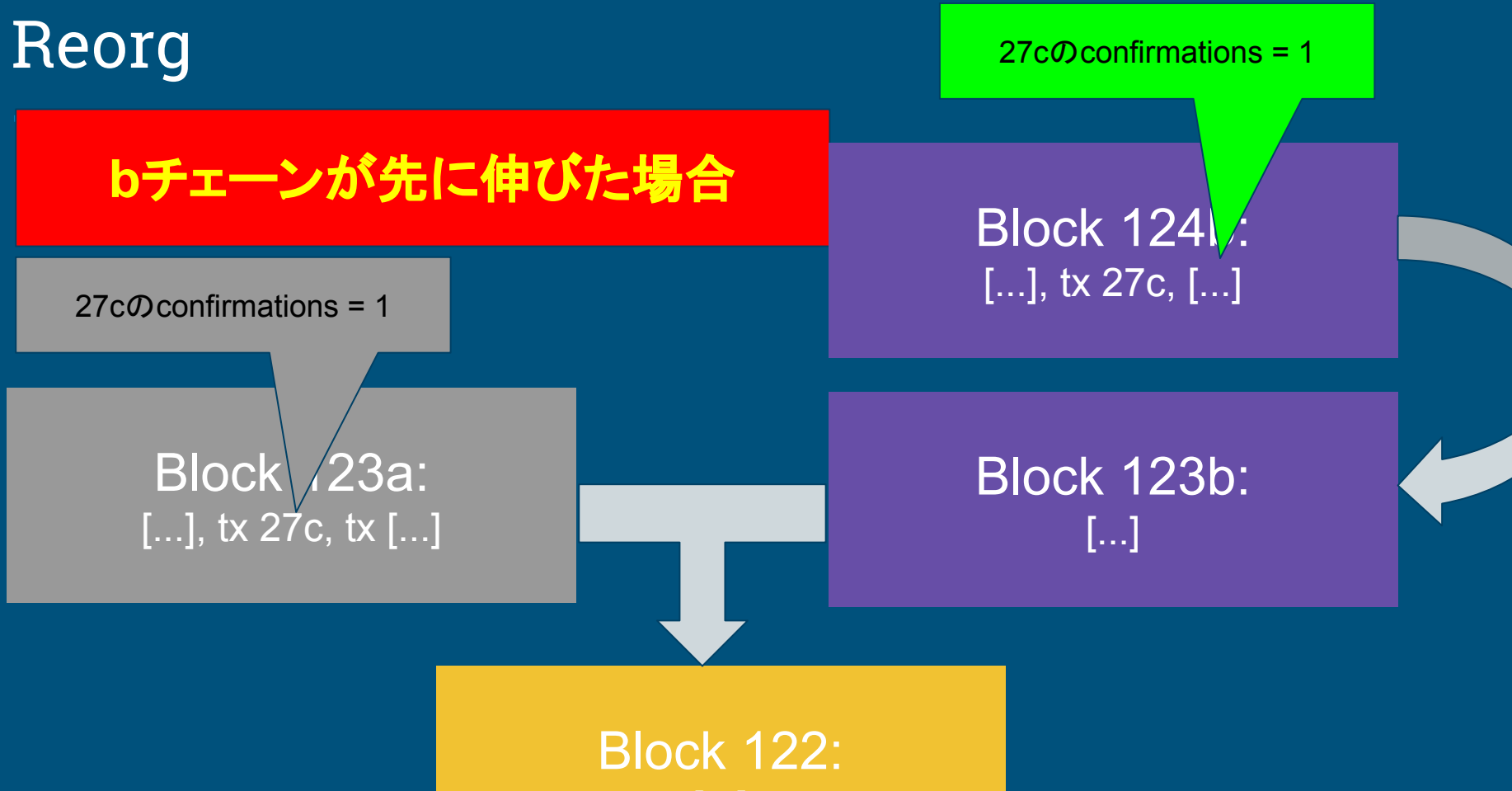
Reorg


123が同時に作成された場合
27cのconfirmations = 1
27cのconfirmations = 0
Block 123a:
[...], tx 27c, tx [...]
Block 123b:
[...]
Block 122:

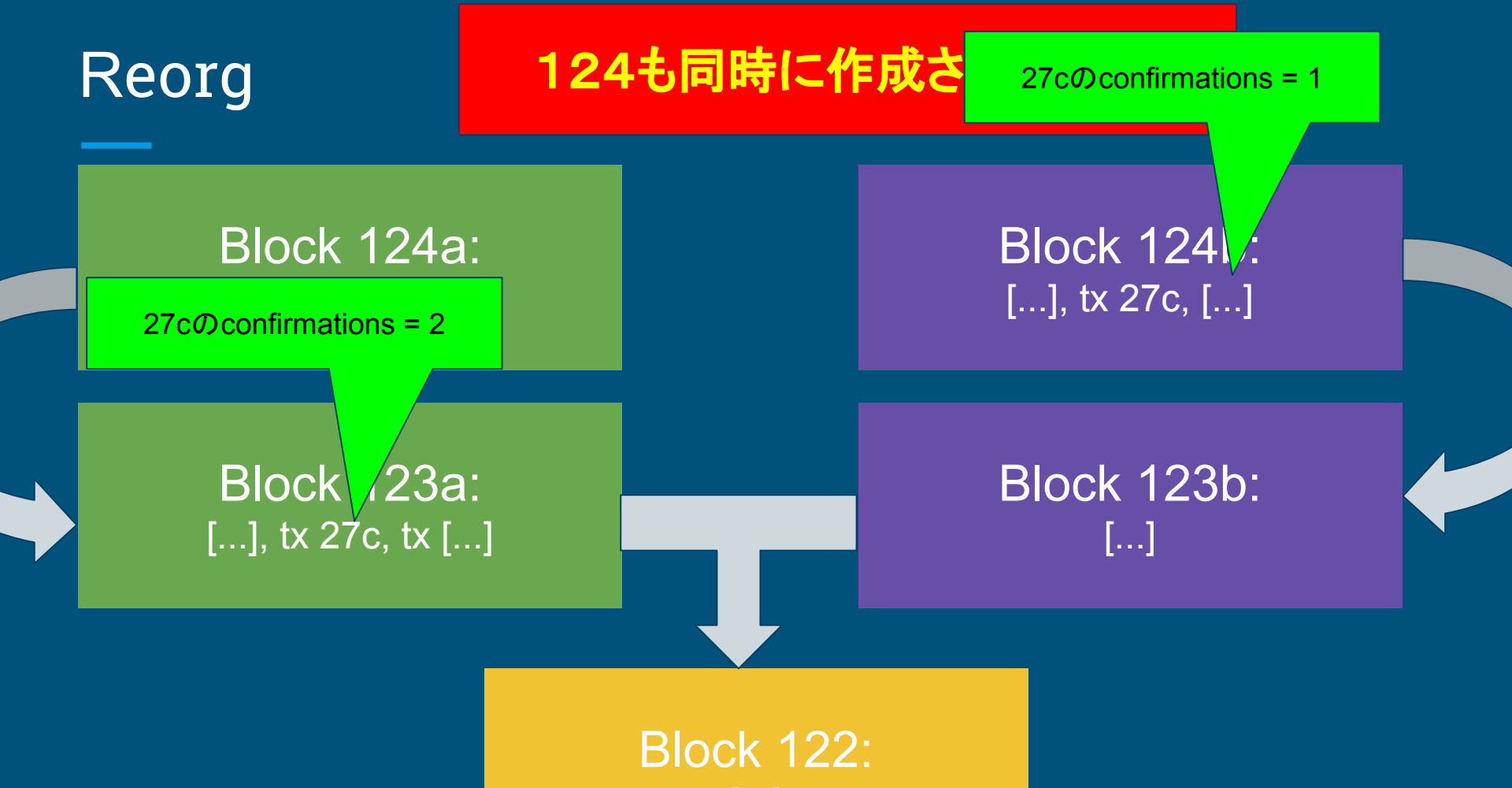
Reorg


aチェーンが先に伸びた場合
Block 124a:
27cのconfirmations = 2
27cのconfirmations = 0
Block 123a:
[...], tx 27c, tx [...]
Block 123b:
[...]
Block 122:

Reorg


bチェーンが先に伸びた場合
27cのconfirmations = 1
27cのconfirmations = 1
Block 124b:
[...], tx 27c, [...]
Block 123a:
[...], tx 27c, tx [...]
Block 123b:
[...]
Block 122:

Reorg

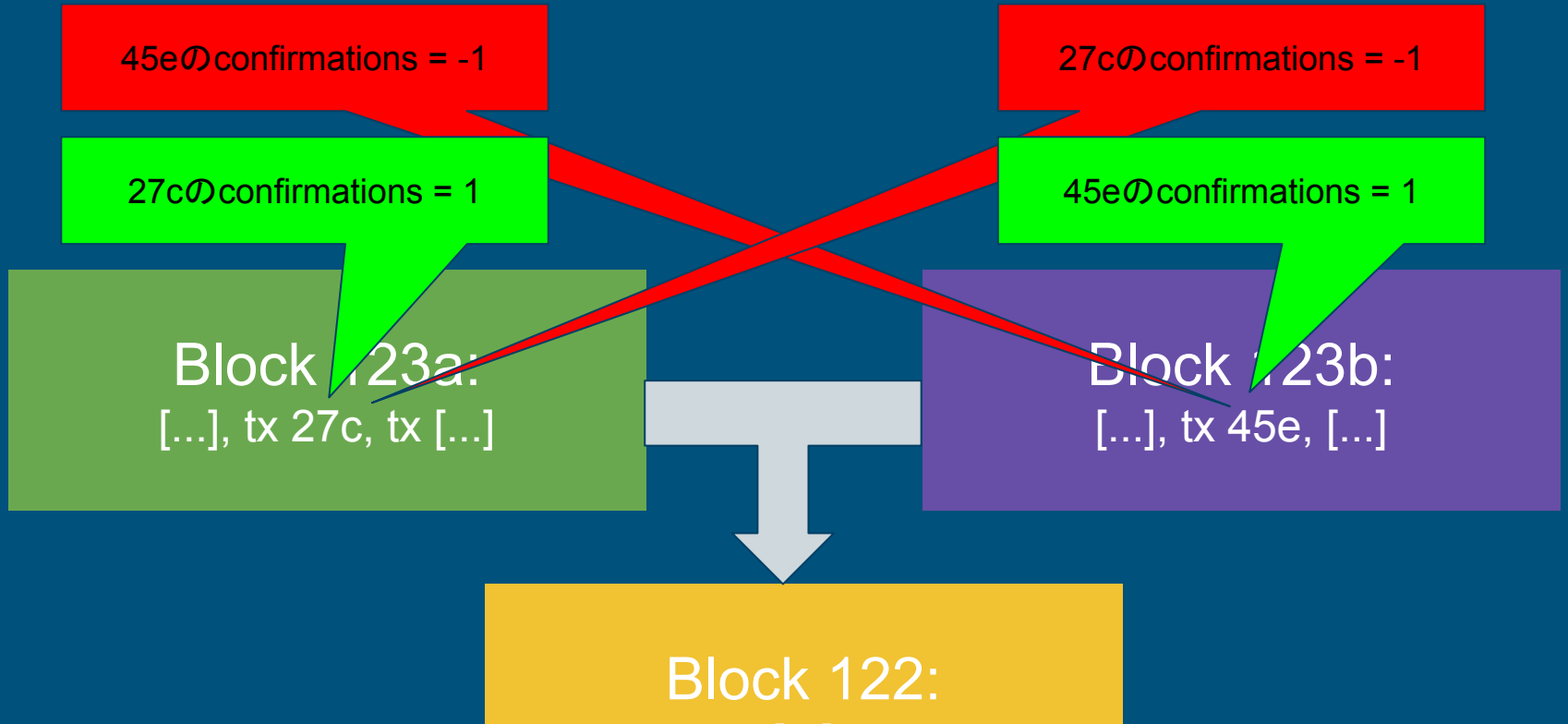

124も同時に作成さ
27cのconfirmations = 1
Block 124a:
27cのconfirmations = 2
Block 124b:
[...], tx 27c, [...]
Block 123a:
[...], tx 27c, tx [...]
Block 123b:
[...]
Block 122:

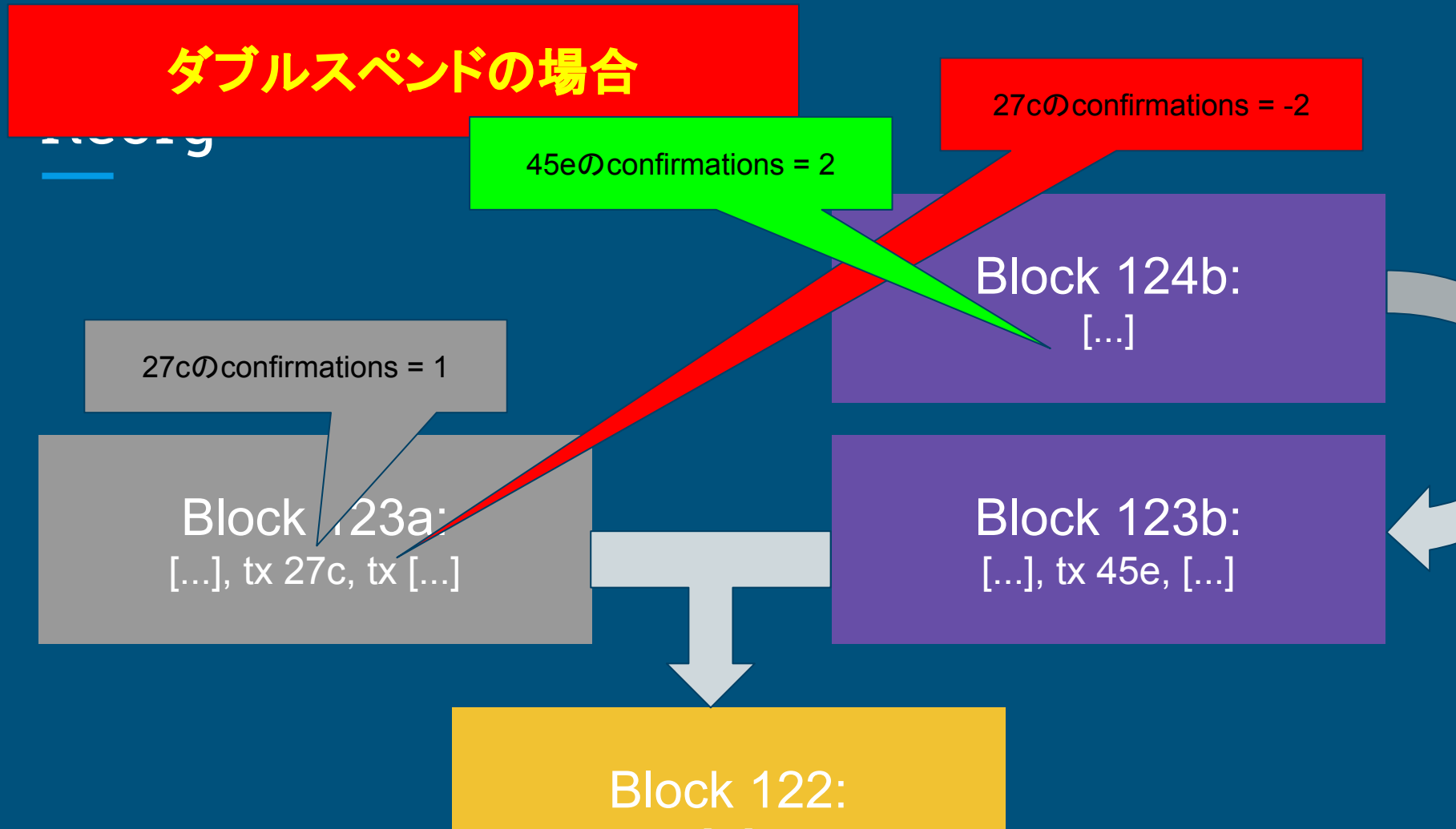
Reorg

TX 27c:
- **txin:**
  - **hash = 356**
  - **index = 0**
- txout:
  - addr = 1abc

TX 45e:
- **txin:**
  - **hash = 356**
  - **index = 0**
- txout:
  - addr = 1def

ダブルスペンドの場合


45eのconfirmations = -1
27cのconfirmations = -1
27cのconfirmations = 1
45eのconfirmations = 1
Block 123a:
[...], tx 27c, tx [...]
Block 123b:
[...], tx 45e, [...]
Block 122:


ダブルスペンドの場合
27cのconfirmations = -2
45eのconfirmations = 2
Block 124b:
[...]
27cのconfirmations = 1
Block 123a:
[...], tx 27c, tx [...]
Block 123b:
[...], tx 45e, [...]
Block 122:

脅威モデル

とりあえず、敵を知れ

- POCやプロトタイプの開発ではあまり気にしない方が良い。
- どこが駄目で、どう解決すれば良いか分かれば充分。
- 最初からあらゆる問題を対応しようとするのは無理！
- とりあえず、敵を知る事。

環境の例え

カジノでコイントスをやるか決める：

・表なら2万円をWIN、裏なら1万円をLOSE。

国が造幣した硬貨か、カジノが造幣したコインかで変わる。

ビットコイン開発が似たような状況である。

ブロックチェーン上の脅威モデル

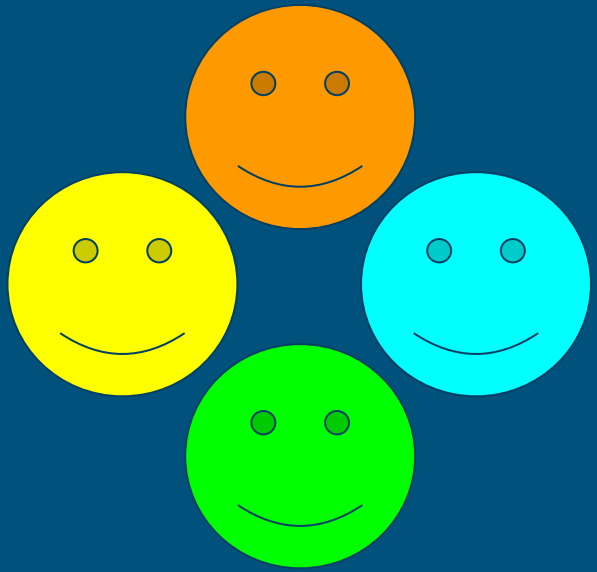
- 一般のソフト開発と異なる部分がいくつかある
- きちんと理解せずに開発すると
  問題が起きる可能性が充分にある
- 「万一」というレベルの話**ではない**
- 今現在全く**安全ではない**ソフトが世の中に溢れている！

脅威モデル

シビルアタック (Sybil Attack)

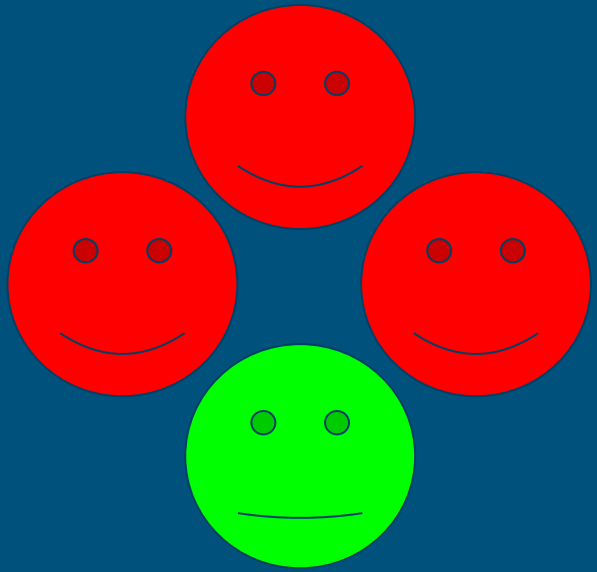
カジノでポーカーに参加する

普通：ディーラー以外の参加者は自分の為に行動を取る。

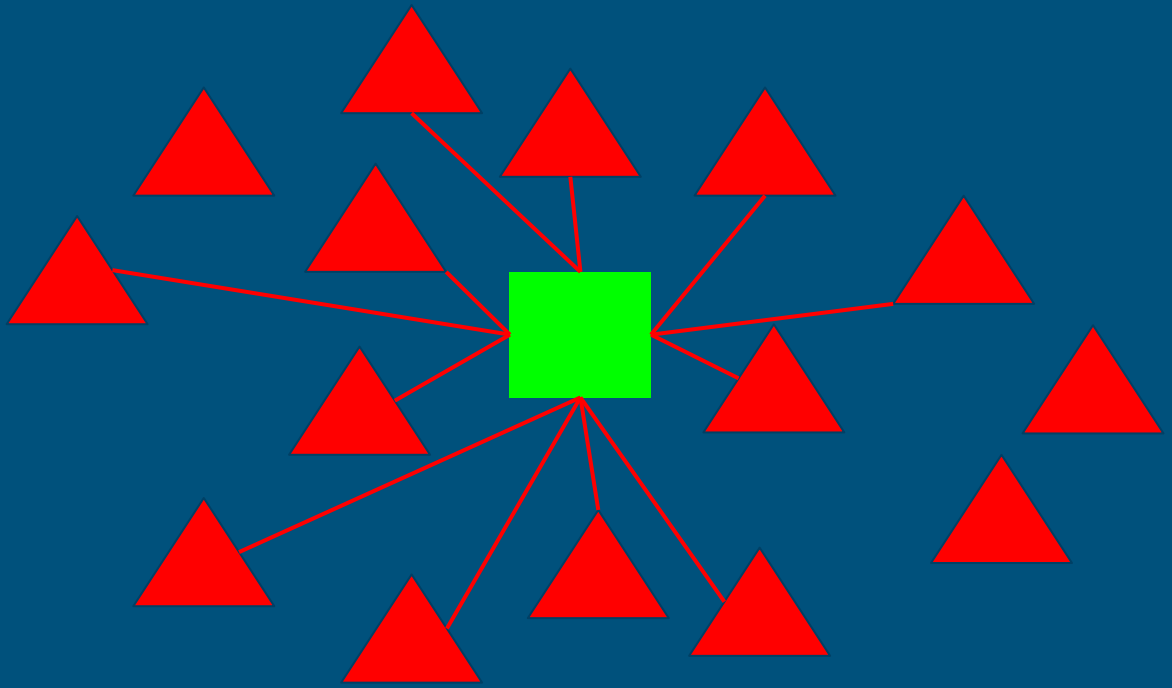


カジノでポーカーに参加する

但し、自分以外の人は皆、カジノに雇われた人なら？



ビットコインネットの場合



シビル攻撃（Sybil Attack）、種類

・一つのノードを攻撃する

・ネットを完全に分割させる

・ネットを部分的に分割させる

シビル攻撃（Sybil Attack）、方法

- スロットを使い切る
- 大量のノードを建てる
- MITM（Man In The Middle）
- ファイバーの接続切断

シビル攻撃（Sybil Attack）、防止

・networkhashpsが下がると怪しい

・複数ノードを複数の場所に建てて、信用するノードとしか接続しない

　　・出来ればVPNを通して

・BIP-150 (Peer Authentication)

脅威モデル

ダブルスペンド

Cheque（小切手）に例える

3万円の小切手を2万円が入っている口座に送信する依頼を入れる。

併せて5万円を持っていると思う。

すぐ後3万円を引き出すが、その後小切手が却下されて口座のバランスが**ー1万円**に。

Cheque（小切手）に例える

小切手が確認されてから使えば問題ない。

ビットコインも同じ。

ダブルスペンド

BTCを貰った様にみえるのに、実は貰っていない。

例：低い目のfeeのtxと、高い目のfeeのtxを作成。

前半をターゲットに送信。確認された時点で後半をネットに送信。

（実にあった例）

ダブルスペンド

展性によるダブルスペンド:

・tx1 → tx2 → tx3（繋がっている、0-conf使用）

・tx1のハッシュを変えるとtx2、tx3が駄目になる

・誰でも出来る（マイナー、リレーノード、...）

（segwitが解決する）

ダブルスペンド

ユーザーのウォレットソフトがクラッシュ。

既に使われたUTXOをもう一度使おうとすることがあり得る。

RBF（Replace By Fee）

結局：txの価値によって、安全程度を決める事。

ダブルスペンド(シビルアタック下)

- 本物のtxをターゲットに送信しない。
- 本物のtxが入っているブロックを送信しない。
- 偽物txをターゲットにしか見せない。

脅威モデル

ビジネスロジックの問題

ビジネスロジックの問題

安全じゃない環境

　　秘密鍵のアクセス（Bitfinexウォレット）

・不可分（atomic）実行

　　Web walletのSubmit問題

・プログラマーエラー

　　ランダムではないランダム（nonceを二回以上使う等）

脅威モデル

確認・信用

確認・信用

- SPVウォレット
- ブロックエクスプローラー

脅威モデル

暗号化の破綻

暗号化の破綻

・ECの弱点

・Address reuse → やばい

・ECが完全に破綻しても、pub key hashしかなければ、それでもbitcoinを安全にさせる方法がある。（hash algoが破られない限り（低可能性））

脅威モデル

コンセンサス問題

コンセンサス問題

・ネットがフォークしてしまう時の話。例：BIP-66（PPCOIN）

・複数のノードを複数の場所に置き、賛成する事を確認すれば問題ない。複数バージョンを同時に使う。

・フォークが起きた場合、両方のフォークの上で全てのtxを同時に送信。

RPCアプリの流れ

RPCアプリの流れ（invoice）

「invoiceシステム」ノードを作成する。

会社がアプリで以下の情報を含むinvoiceを発行する。

・invoice number
・bitcoin address（ここに払ってもらう）
・amount（払ってもらう金額）
・content（払ってもらう理由-品目など）

RPCアプリの流れ（invoice）

具体的には：

① bitcoin.getNewAddressからaddrを入手する。

② dbのinvoiceにaddr, amount, contentをinsertする。

RPCアプリの流れ（支払）

Invoiceに付いているbitcoin address（addr）に誰かがbitcoinを送った時に、そのtxを記録する。

これをpaymentとして、DBに追加する。

```
db.payment.insert({txid,addr,invoice,amount})
```

既に記録しているtxidの場合は更新する。

RPCアプリの流れ（支払）

Paymentに変化があった場合（create/update）、invoiceのステータス（未入金、一部入金など）を更新する。

- if sum(payment.amount) = 0 → unpaid
- if sum(payment.amount) = invoice.amount → paid
- if sum(payment.amount) < invoice.amount → partial
- if sum(payment.amount) > invoice.amount → overpaid

...

RPCアプリの流れ

ここまでのおさらい：

① invoiceにbitcoin address (addr) を付ける

② 支払先がaddrのtxを見つけたらpaymentとして記録する

③ sum(payment.amount) = invoice.amountだとpaid（入金済）として認める

RPCアプリの流れ

これだけだと、問題が複数ある。

どんな問題が起き得るか1分考えよう。

Task 1

Task 1

場所：payments.js, 25行目（createInvoice）

内容：model.invoiceの作成

詳細：関数の足りない部分を補う（具体的には、bitcoin addressを獲得し、それをinvoiceのDBに登録する）

ヒント：非同期プログラミングを調べるとやり易くなる（http://bc-2.jp/materials/0103_追加書類.pdfを参考に）

Task 1

リファレンス実装では5行（4ステップ）。

テスト（npm test）でPASSすること（13/40→22/40）:

```
Invoice ✓ can be created
reorg unconfirm then... / double spend (replace...):
  ✓ can create an invoice
  ✓ can pay an invoice
double spend replace (payment to same address):
  ✓ submits invoice
```

Task 1

Task 1が完了すれば、rpc-cliというコマンドでinvoiceを作る事ができるようになる。

```
$ ./rpc-cli create 5 "靴下"
$ ./rpc-cli list
invoice:                         amount: status:         conf:      pending:
587ca8535a1e4536248f5df9         5       ???             0 BTC      0 BTC
```

でもrpc-cli info 587ca…はまだできない...。

Task 2

Task 2

場所：payments.js, ~100行目（updateInvoice）

内容：paymentの対応

詳細：paymentの確認と

total/final/disabledAmountの計算

（上の方に変数がある）

Task 2

リファレンス実装では25行（13ステップ）。

テスト（npm test）でPASSすること（22/40→25/40）

ヒント1：reorgが発生した場合、以下が必要：

```
model.payment.setStatus(payment._id, 'reorg', (err) => {
  model.history.create(invoice._id, payment._id, {
    action: 'reorg',
  …
```

Task 2

ヒント2：reorgが発生したかどうかは、txがブロックに入っているかどうかで分かる。

そして、transaction.blockhashがnullであれば、そのtxはブロックに入っていないということになる！

Task 3

ヒント：（過不足なく）丁度支払ったという事を

```
Math.abs(支払った金額 - 要求した金額) < config.thresholdSatoshi
```

というように確かめる。

Task 3

Task 3

場所：payments.js, ~140行目（updateInvoice）

内容：invoiceのアップデート

詳細：ここまでに確認・集計したpaymentの情報を元に、invoiceをアップデートする

Task 3

リファレンス実装では23行（21ステップ）。

テスト（npm test）でPASSすること（25/40→40/40）

bitcointestについて

bitcointestはノードを作ったり消したりする。

大抵うまく動作するが、稀にプロセスが残ることもある。

bitcointestというテストが駄目だったりしたら

```
$ killall -9 bitcoind
```

で直るが、…

bitcointestについて

…これではすべてのbitcoindのプロセスが殺される。
bitcointestのプロセスを個別に指定するなら:

```
$ ps -A | grep bitcoind
```

でリストを見て、

```
11453 ... ../../src//bitcoind -regtest
-datadir=/tmp/bitcointest//22011 -rpcuser=user…
```

というようなノードのPID(11453)をkill -9に渡す。

ヒント等

全てのスライドはhttp://bc-2.jp/materials/で落とせる。

node.jsに自信の無い方は**0103_追加書類.pdf**を参考にすると良い。

チームを組んでやることをおすすめする。

ヒント等

npm testが実行するテストのソースコード（tests.js）を見れば、何が駄目か明確になるかもしれない。

`TASK=1` `npm test`とするとTask 1のテストだけが実行される。

同じく1を2, 3にするとTASK 2, 3になる。

ヒント等

V=1 `npm test`ではテストが出力する情報が増える。
DEBUG=bc2 `npm test`ではRPCアプリのdebug()が表示される。これらは同時に指定できる:

```
$ V=1 DEBUG=bc2 npm test
```

DBをリセットしたい時(**全てが消える！！！**):

```
$ mongo --eval "db.dropDatabase()"
```

ヒント等

はまったら遠慮無く#rpc-appにて質問をしたり、私たちに直接聞いたりして下さい。

頑張って下さい！

@DG Lab - Karl-Johan Alm