

第19回ICN研究会ワークショップ

Cefore/Cefpycoを用いたICNによる ネットワーク機能呼び出し実装

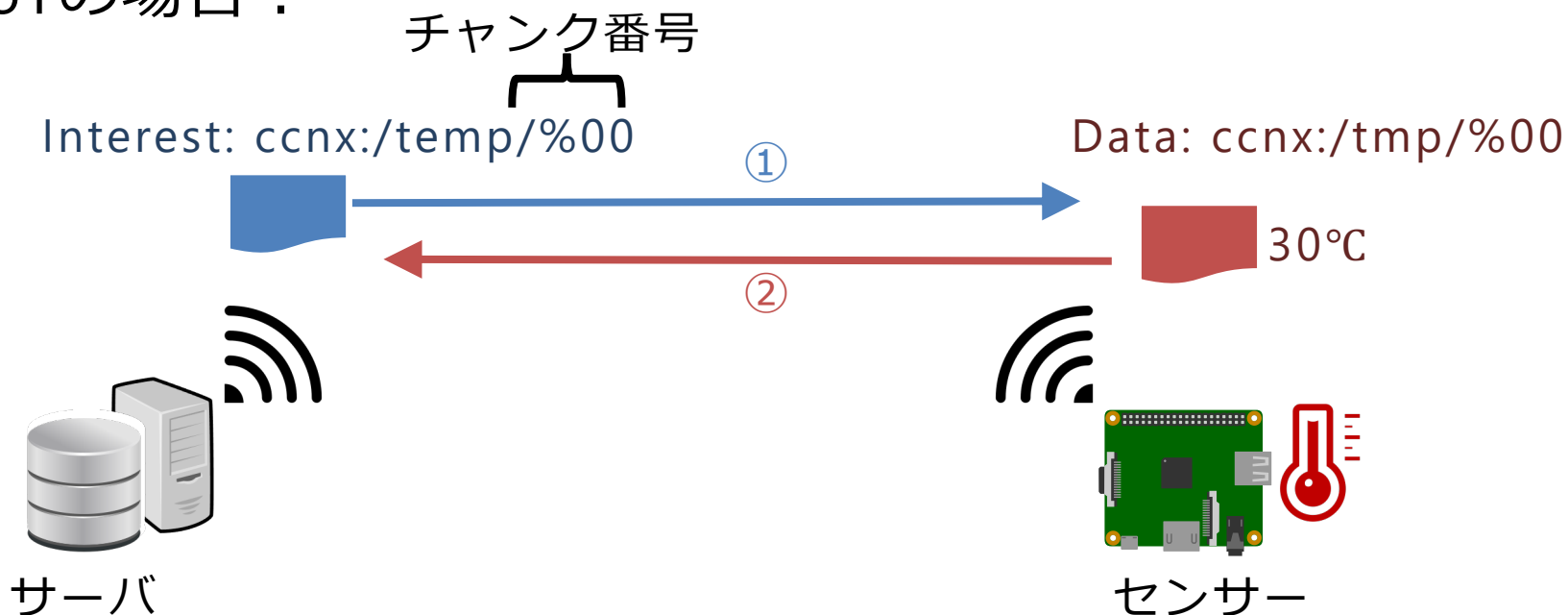
2021年8月26,27日

■ ICNの基本原則

● Pull型通信：

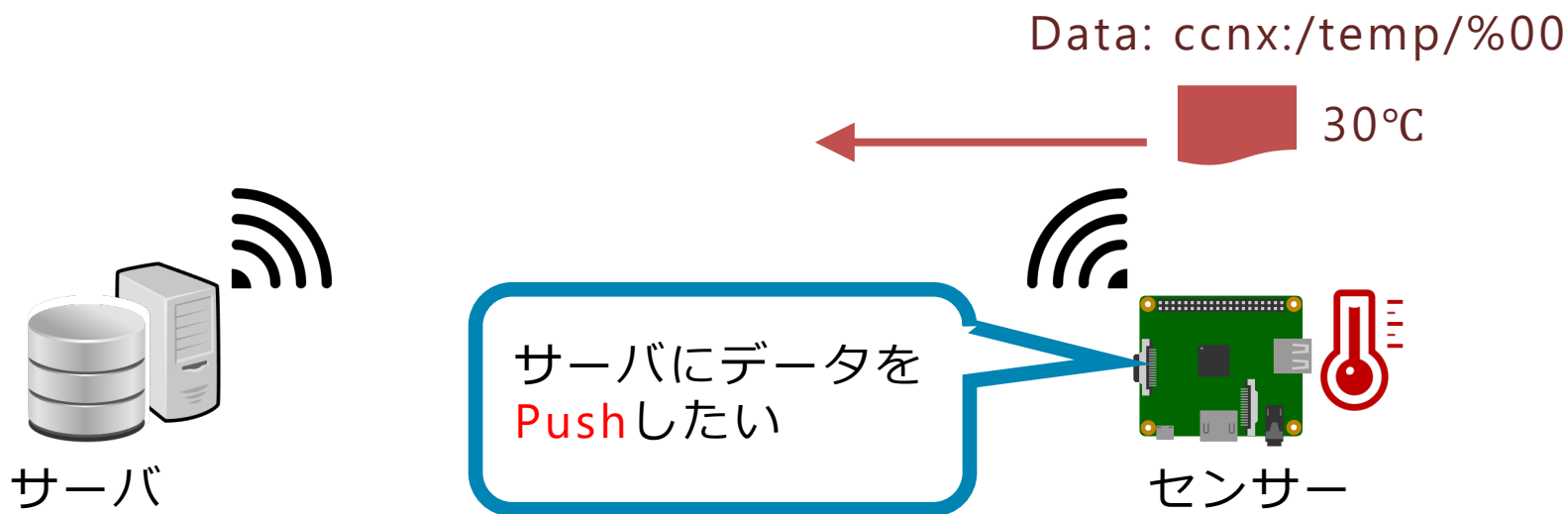
- 1) 取得したい情報/コンテンツの名前を指定したInterestを送信
- 2) ネットワーク(のキャッシュ)から当該Dataを取得

■ IoTの場合：



NICT ICNによるPush型通信

- IoT環境では、Push型通信もしばしば必要
 - どうする？

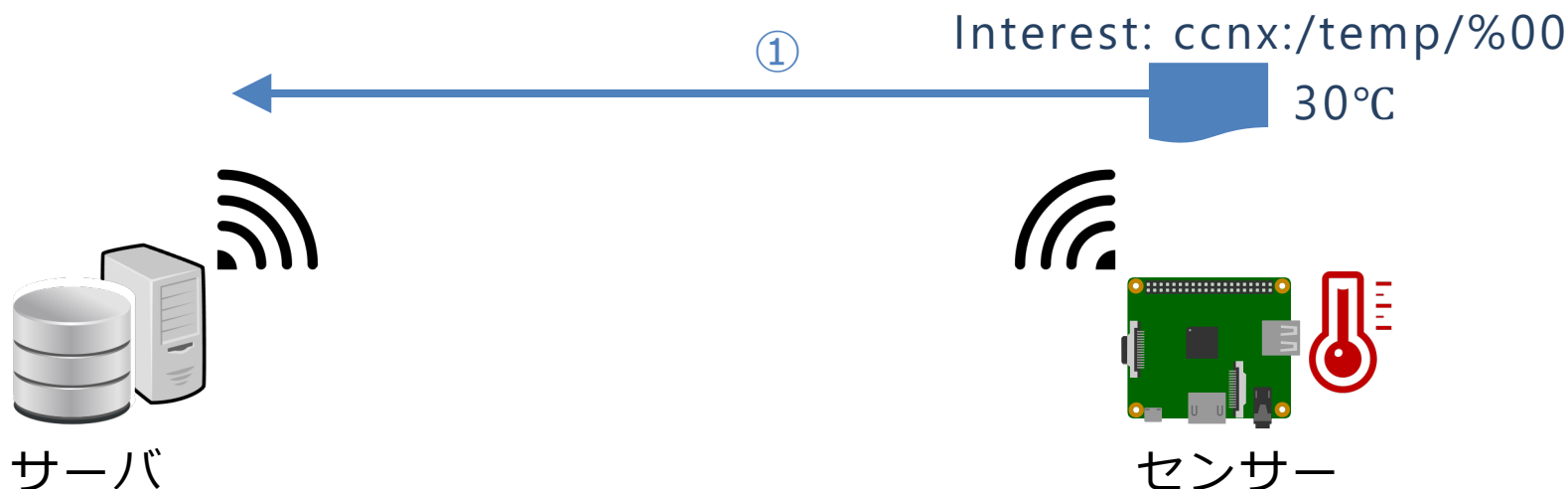


■方法1：

- センサーがInterestに情報を載せてサーバに送る
 - CeforeではPayload TLVが定義されている(参照: CCNx(RFC8609))

■方法1の特徴

- メリット：シンプルで分かりやすいが、
- デメリット：一つのInterestで一つのデータをネットワークから引き出すというICNの原則から乖離

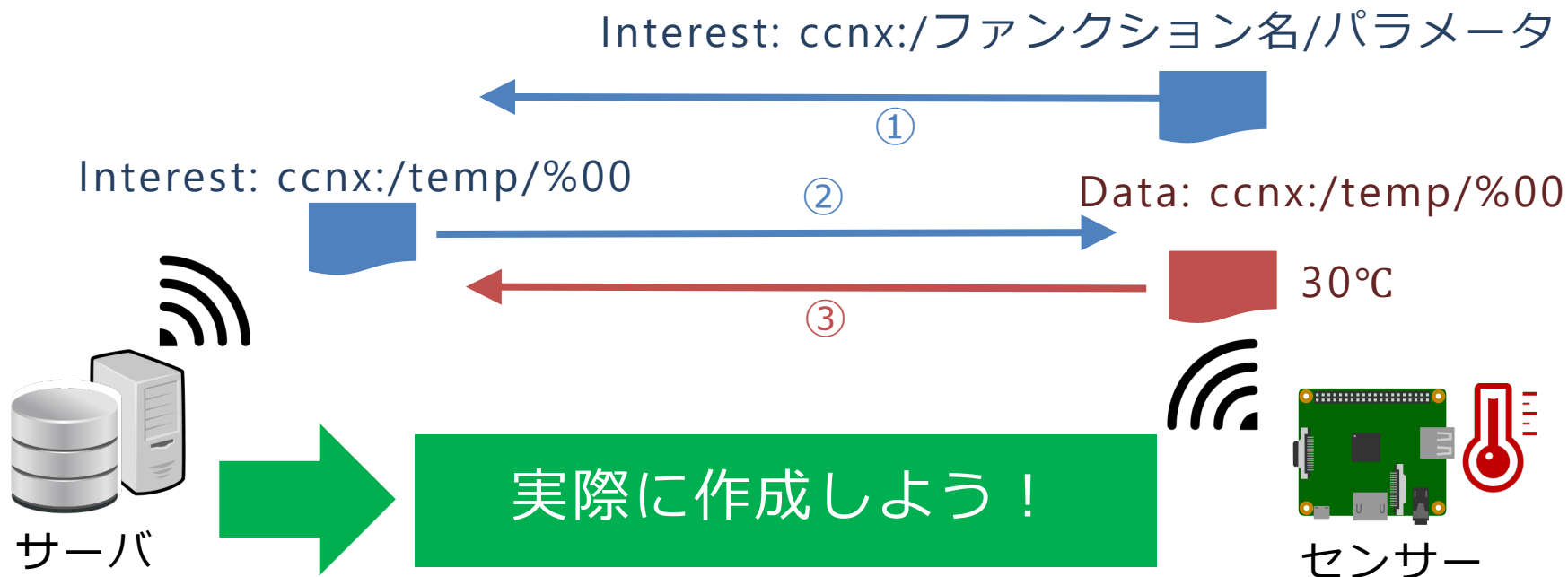


■情報の名前を指定したPull型通信

- ネットワーク内のキャッシュからデータを引き出す
ネットワーク機能と捉えることができる

■方法2：

- ネットワークにデータをキャッシュさせるPush機能を
ネットワーク機能として定義し名前呼び出す



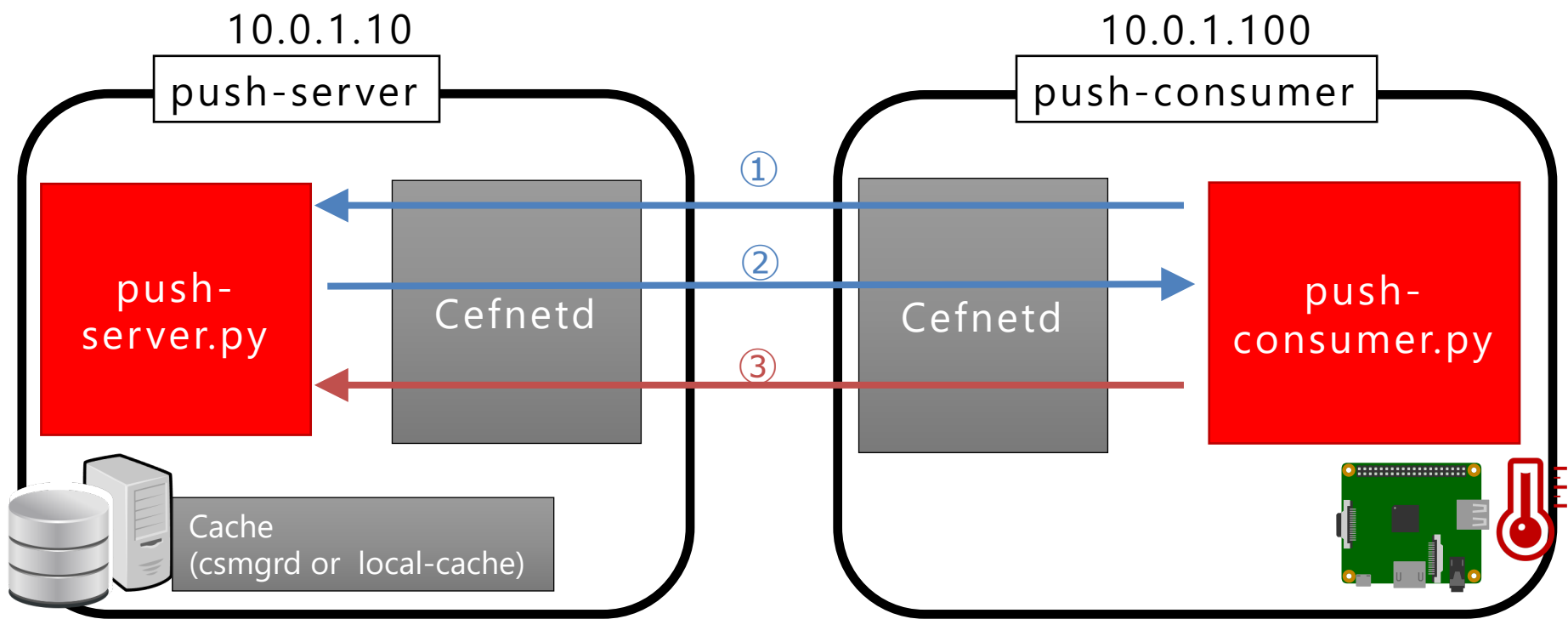
ネットワーク機能呼び出し Practice (1)

Push型通信基本編

ICNによるPush型通信: 全体図(方法 2)

■ゴール：

- ① センサー(push-consumer)は、PUSHイベントをネットワーク機能として呼び出すためのInterestを送信し、
- ② サーバー(push-server)は、指定された名前を使ってInterestをセンサーに送信し、
- ③ センサーは、該当するデータ(文字列: 30 degree celsius)をサーバに送信する



- ccnx:/_SF_abcdef_/K/NAME/%00
- ① ② ③

- `push-consumer.py`及び`push-server.py`にて定義している箇所を確認してみよう

- `push-consumer.py`で定義している箇所を確認してみよう

- `push-consumer.py`にて定義している箇所を確認してみよう

- ccnx:/_SF_abcdef_/5/Current-Temp/%00



ICNによるPush型通信: 実行準備

■ 前準備

- “push-consumer”コンテナ、及び“push-server”コンテナを起動
 - `$./start-C-1.bash`
- ターミナルから“push-consumer”にログイン
 - `$ docker exec -it push-consumer bash`
- 別のターミナルから“push-server”にログイン
 - `$ docker exec -it push-server bash`

ICNによるPush型通信: 実行準備

■ 前準備

● センサー(**push-consumer**)

- FIB設定を行う。以下のコマンドを実行。

```
$ cefroute add ccnx:/_SF_abcdef_ udp 10.0.1.10
```

● サーバー(**push-server**)

- FIB設定を行う。以下のコマンドを実行

```
$ cefroute add ccnx:/Current-Temp udp 10.0.1.100
```

- キャッシュを行う必要があるため、cefstatusコマンドにて、Cache ModeがLocalcacheになっているか確かめよう

- ```
$ cefstatus
```

- キャッシュを行う必要があるため、CS\_MODE=1 (Local-Cacheを有効化)でceforeが起動している必要あり。(または、CS\_MODE=2 (use csmgrd)でCsmgrdを使って実行しても良い)。

- Local-Cacheを有効にする手順

- 1) /usr/local/cefore/cefnetd.confを編集し、CS\_MODEを以下にする
  - CS\_MODE=1
- 2) cefnetdの再起動
  - ```
$ cefnetdstop
```
 - ```
$ cefnetdstart
```

# ICNによるPush型通信: 実行準備

## ■ 実行する前に確かめよう

- センサー(**push-consumer**)のFIB設定はできているか？
  - コマンド"cefstatus"を実行して、下記が表示されるか？
    - faceid = XX : address = 10.0.1.10:9896 (udp)
    - FIB: 1
      - ccnx:/\_SF\_abcdef\_
        - Faces: XX (-s-)
- サーバー(**push-server**)のFIB設定は出来ているか？
  - コマンド"cefstatus"を実行して、下記が表示されるか？
    - faceid = YY : address = 10.0.1.100:9896 (udp)
    - FIB: 1
      - ccnx:/Current-Temp
        - Faces: YY (-s-)
- サーバーのキャッシュモードは、LocalCache)になっているか？
  - コマンド"cefstatus"を実行して、下記が表示されるか？
    - Cache Mode : LocalCache

# ICNによるPush型通信: 実行

- 1) サーバー(**push-server**)で下記コマンドを実行  
\$ python3 bin/push-server.py
- 2) センサー(**push-consumer**)で下記コマンドを実行  
\$ python3 bin/push-consumer.py
- 3) サーバー(**push-server**)で適切にキャッシュされているか確認してみよう
  - (Ctrl+c を入力し、push-server.pyを停止し、下記を実行)
  - \$ cefgetfile ccnx:/Current-Temp -f ./cache-data.txt
  - \$ cat cache-data.txt
    - ("30 degree celsius"という内容がチャンク数分だけ表示されるはず!!)
- 再度、このシナリオを実行したい場合
  - サーバーにキャッシュ(ccnx:/Current-Temp)が残っているので、キャッシュを消すために、サーバーにてcefnetdを一旦終了し、再起動しよう
    - \$ cefnetdstop
    - \$ cefnetdstart
    - \$ python3 push-server.py

※ push-server.pyなどのプログラムを強制終了したい場合、Ctrl+c を入力

# ICNによるPush型通信: 試してみよう

## ■ Push機能呼び出す名前とPushするデータ名を自分で決めてみよう

## ■ 下記は、2つのサンプルプログラム(Push-Server.py, Push-Consumer.py)から抜粋

- ccnx:/\_SF\_abcdef\_/K/Name/%00
- ①                      ②                      ③

### ① ネットワーク機能名

- "\_abcdef\_"の部分を自分で定義して、push-consumer.pyの該当箇所を変更してみよう

### ② チャンク数

- チャンク数は一つでも良いので、push-consumer.pyの該当箇所"k"を1に変更してみよう。

### ③ キャッシュさせたいデータの名前

- データの名前を決めて、2つのサンプルプログラムの該当箇所"Name"を変更してみよう。

センサーとサーバーのFIB設定を適切に反映することを忘れずに

## ■ PUSH機能呼び出すInterestの名前例:

- ccnx:/\_SF\_PUSH\_/1/Current-Temp-MySensor/%00

## ■色々考えてみよう

- サーバー側のFIB設定に関して
  - ・ サーバーがセンサーからPUSH機能要求を受信した際に、自動的にFIB設定するにはどうしたら良いだろうか？
- 中継ノード(ルータ)を介してPushを行う場合、ルータはどのような機能が必要か？
  - ・ (サーバ)<--> (中継ノード(ルータ))<--> (センサー)
- 様々な場所に温度センサー等が散らばっている状況で、複数センサーが現在の温度データを同一サーバにPushする場合、
  - ・ どのようなPUSH通信を行う必要があるだろうか？
  - ・ また、どんなメリットやデメリットがあり得るだろうか。

# ネットワーク機能呼び出し Practice (2)

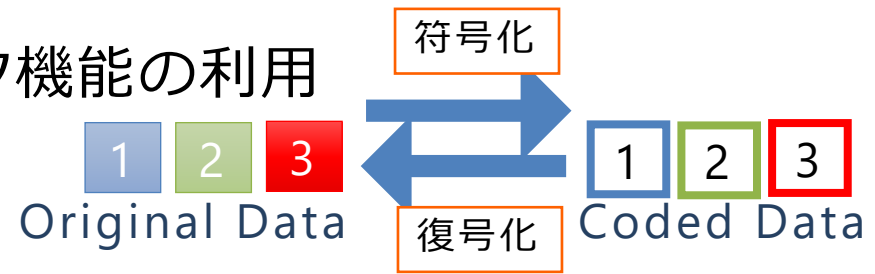
---

Push型通信応用編

# ICNによるPush型通信: 応用編

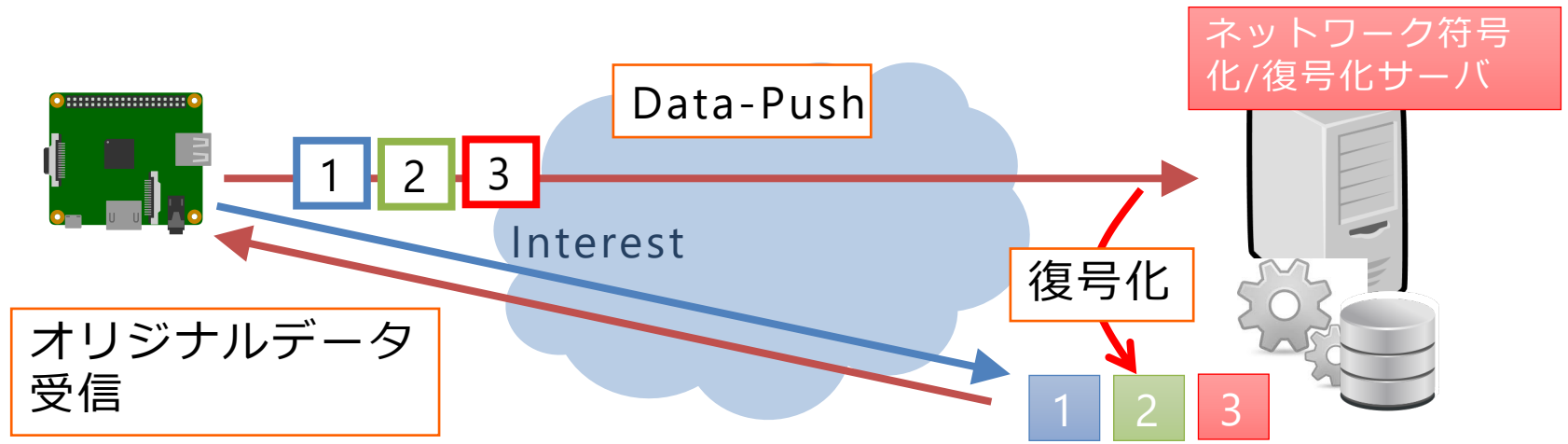
## ■ PUSH型通信利用いたネットワーク機能の利用

## ■ 例えばネットワーク符号化(NC)



### ● シナリオ例:

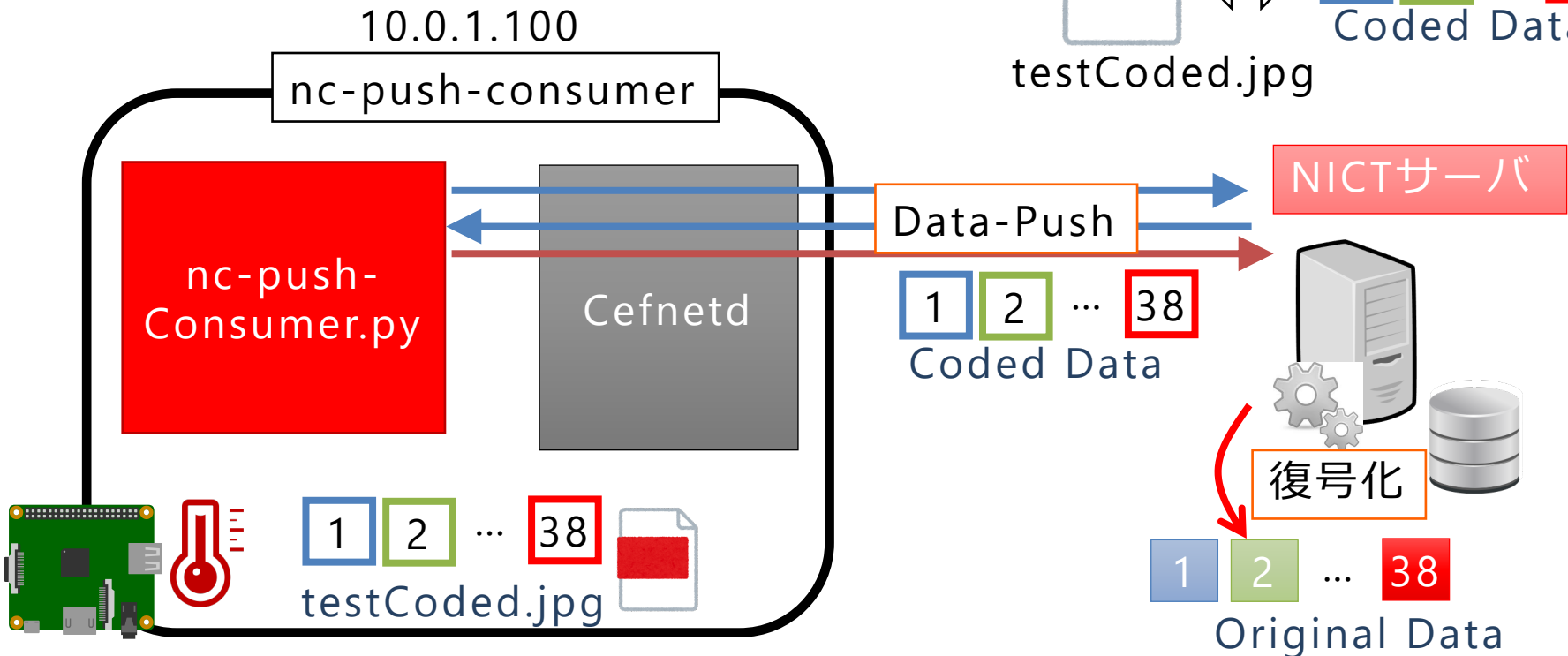
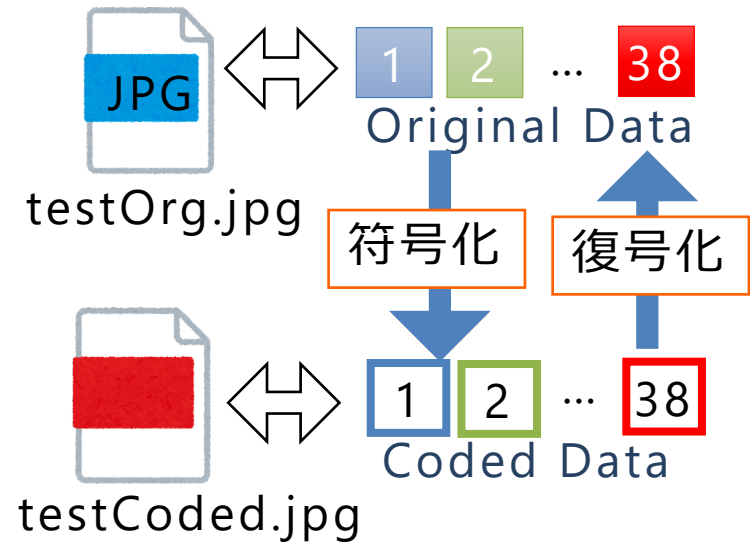
- 1) 計算機性能が乏しいノード、あるいは復号化機能が無いノードが符号化データK個受信。
  - 符号化データ: オリジナルデータ(K個)を元に作成され、オリジナルデータとは異なるデータ
  - 復号化: 符号化データ(K個)を元にオリジナルデータ(K個)を生成
- 2) 符号化データをサーバに渡し、復号化・キャッシュしてもらう
- 3) サーバにキャッシュされてあるオリジナルデータを受信





# ICNによるPush型通信: 応用編

- 手順1: "testCoded.jpg"(ファイル)をNICTサーバにPushして、復号化・キャッシュしてもらう
  - "testCoded.jpg"は"push-consumer"のホームディレクトリにあります。
- 手順2: cefgetfileで復号化したオリジナルファイルを取得



(前準備)

- ホストの“practice-C”ディレクトリにて、
  - 既に起動してあるコンテナを終了させよう
    - `$ ./teardown.bash`
  - 設定ファイルを変更するために、下記を実行
    - `$ cp docker-compose.yml_global docker-compose.yml`
  - “start-C-2.bash”を編集
    - 3行目の “#mkdir -p /tmp/log” のコメントアウト“#”を削除
      - → “mkdir -p /tmp/log”
  - “nc-push-consumer”コンテナを起動
    - `$ ./start-C-2.bash`
- ターミナルから“nc-push-consumer”にログイン
  - `$ docker exec -it nc-push-consumer bash`

# ICNによるPush型通信: 応用編

## ■ “nc-push-consumer”にて、符号化データをサーバに送信し復号化を要求

### ● Fibを追加

- \$ cefroute add ccnx:/\_SF\_tcp [NICTサーバのIPアドレス]

### ● nc-push-consumer.pyを実行

- \$ python3 bin/nc-push-consumer.py bin/testCoded.jpg 自身の苗字

- 例: \$ python3 bin/nc-push-consumer.py bin/testCoded.jpg Matsuzono

上手くサーバからInterestを受信できない場合、“cefstatus”コマンドで、FIBが適切に設定されているか確かめてみよう。どうしても動かない/動かなくなった場合、cefnetdを再起動しよう

### ● 実行内容

- 符号化データをPushするためのInterest送信

- 名前: ccnx:/\_SF/\_CS.STORE/\_K.38/\_uid.Matsuzono\_

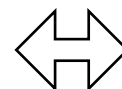
- NICTサーバは、consumerに38個のInterestを送信してくれます

- 名前: ccnx:/\_uid.Matsuzono\_/chunk=0,...,37

- testCoded.jpgファイルを1024byteづつ読み込み、サーバに符号化データを38個送信(PUSH)

- 名前: ccnx:/\_uid.Matsuzono\_/chunk=0,...,37

testCoded.jpg

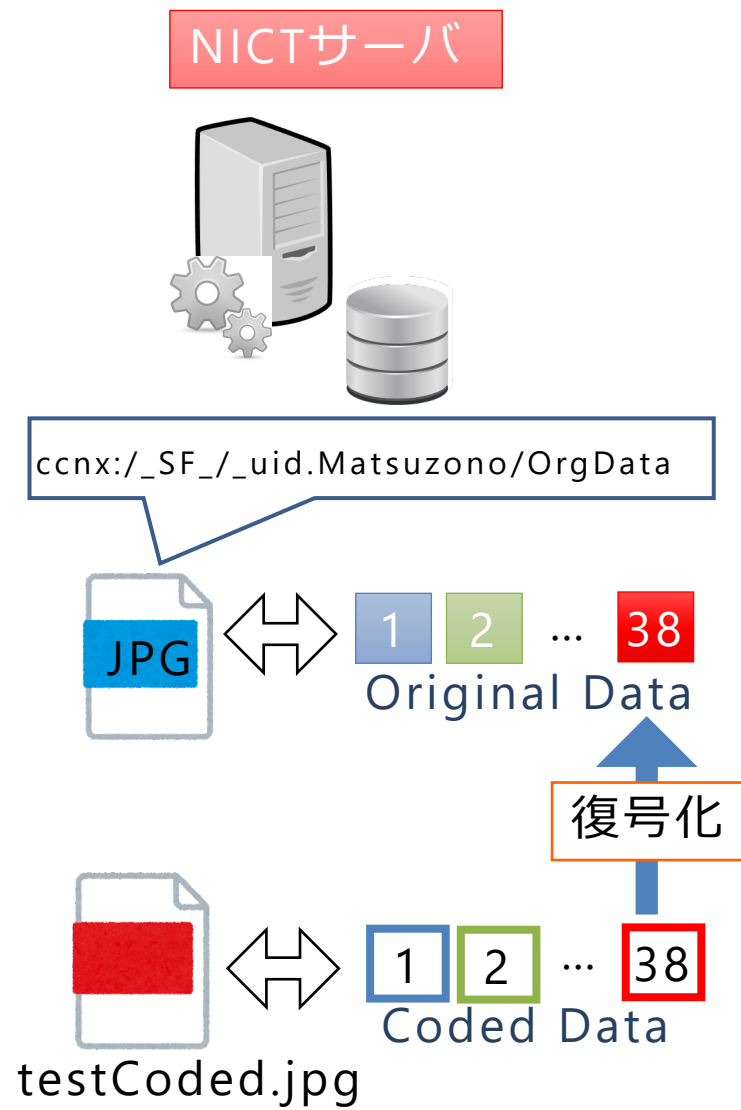


1 2 ... 38  
Coded Data

# ICNによるPush型通信: 応用編

## ■ サーバ側の実行内容：

- NICTサーバは、consumerに38個のInterestを送信
  - ・ 名前:  
ccnx:/\_uid.Matsuzono\_/chunk=0,...,37
  - ・ このために、“cefroute”コマンドを自動で実行し、適切なFIBを作成する
- 38個全ての符号化データを受信した後、オリジナルデータを復元し、以下のファイル名でキャッシュする
  - ・ ファイル名：  
ccnx:/\_SF/\_uid.Matsuzono\_/OrgData
- その際、チャンクデータをいくつ受信したかをステータス情報として、以下のファイル名でキャッシュ
  - ・ ファイル名：  
ccnx:/\_SF/\_uid.Matsuzono\_/Status



# ICNによるPush型通信: 応用編

- 復号化されたオリジナルデータをNICTサーバから受信しよう
  - cefgetfileの利用
    - 38個のオリジナルデータを受信し、“testOrg.jpg”というファイル名で保存
    - `$ cefgetfile ccnx:/_SF/_uid.Matsuzono_/OrgData -f /tmp/log/testOrg.jpg`
  - 保存したtestOrg.jpgが適切に開けるか見てみよう
    - ホストの“/tmp/log”ディレクトリのtestOrg.jpgを開こう
  - うまくtestOrg.jpgが受信できない場合、サーバのステータスを調べてみよう。
    - `$ cefgetfile ccnx:/_SF/_uid.Matsuzono/Status -f ./Status`
    - `$ cat ./Status`
- 課題：
  - “push-server”コンテナを使って、サーバプログラムを自作し、“push-consumer”コンテナを使って、独自の機能を実装してみよう。