Source 型DAQコンポーネント開発

KEK

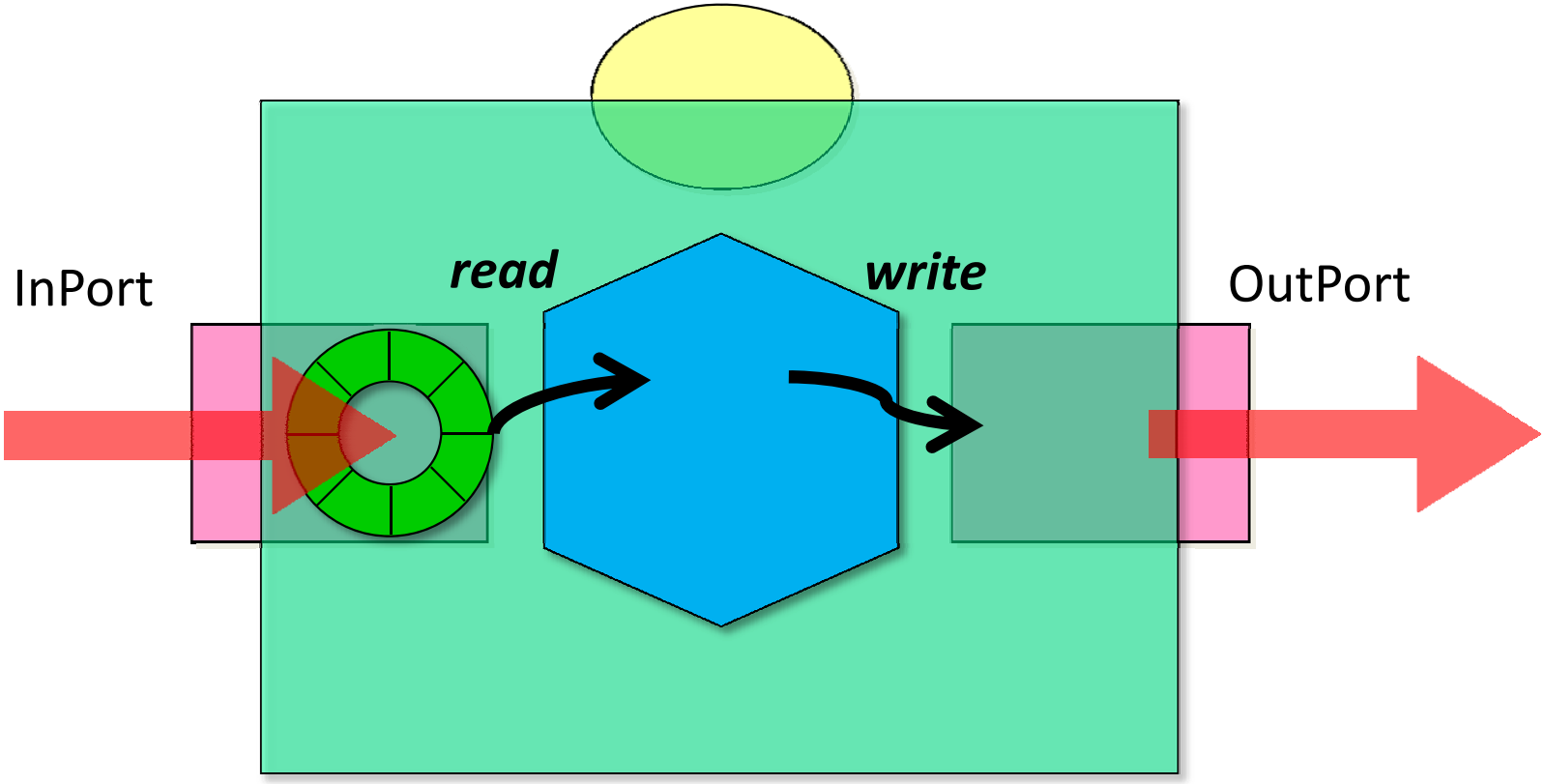
Kazuo Nakayoshi

内容

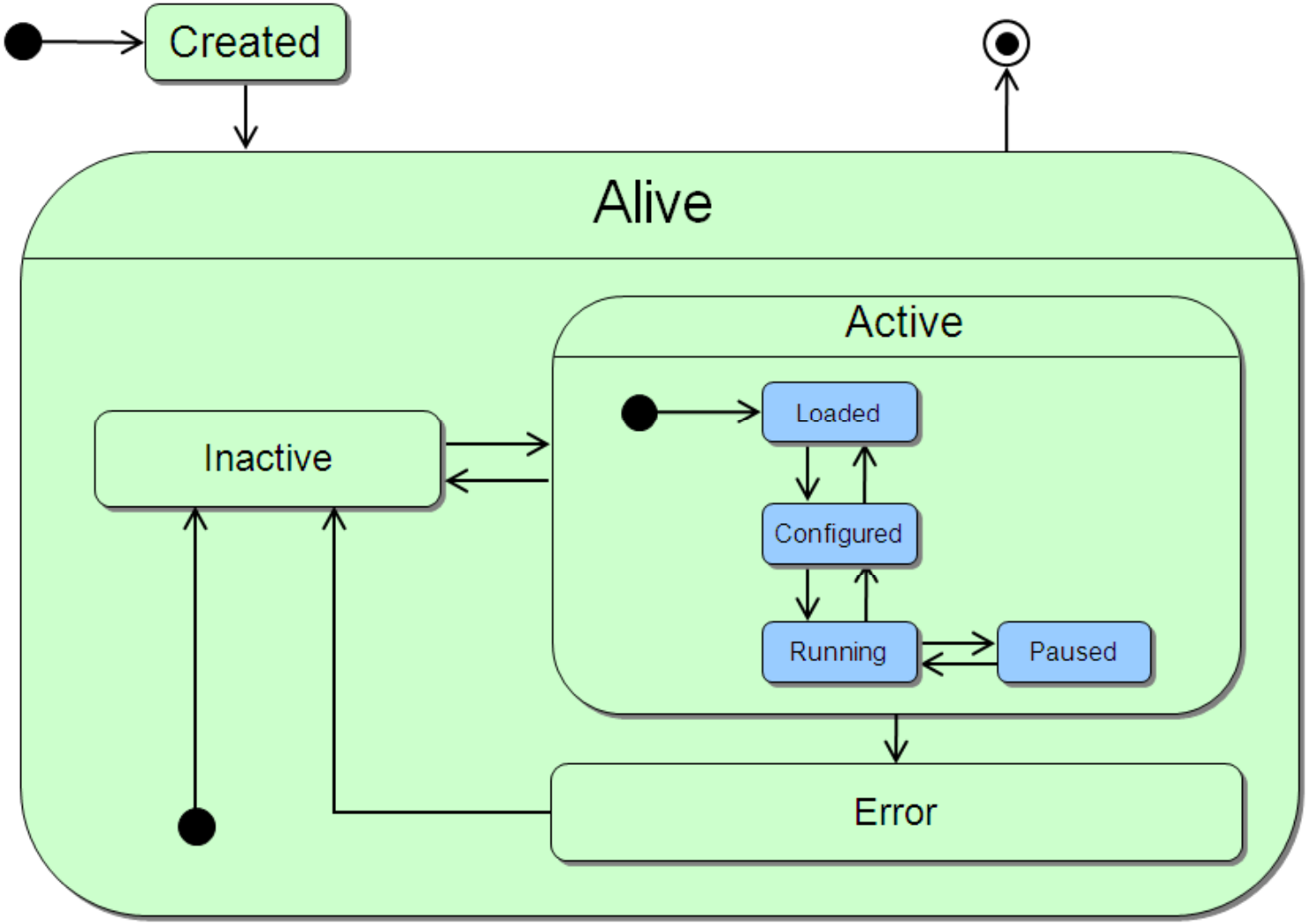
- DAQコンポーネントのデータポート
- DAQコンポーネントのステート
- DAQコンポーネントのタイプ
- Source型コンポーネント開発の手順

DAQコンポーネントのInPort/OutPort

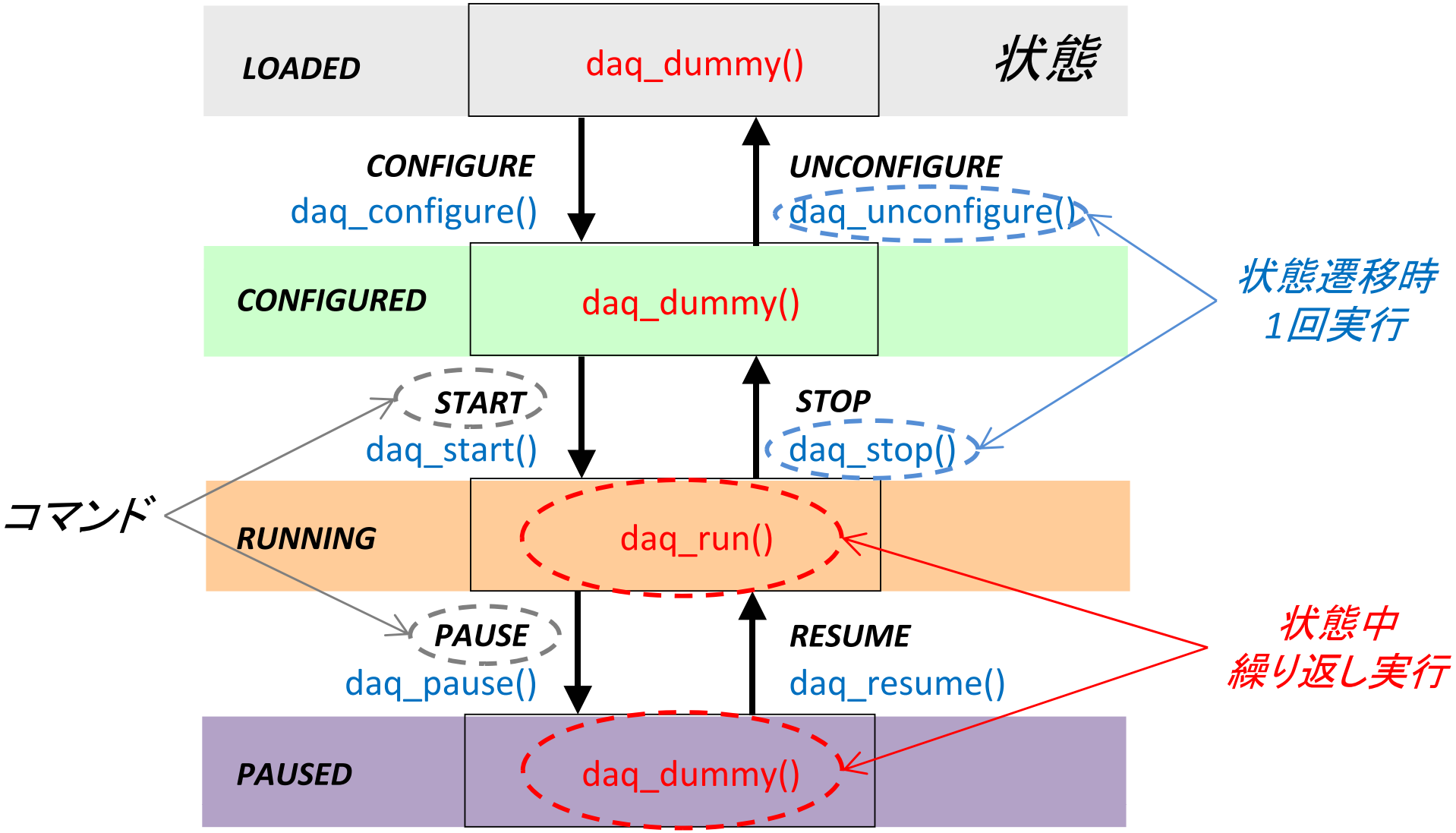
- InPortから受信したデータは、バッファに格納される。
- OutPortのバッファへ書いたデータは、直ちに転送される



DAQコンポーネントのステートチャート



各Stateの実装


LOADED
daq_dummy()
状態
CONFIGURE
daq_configure()
UNCONFIGURE
daq_unconfigure()
CONFIGURED
daq_dummy()
状態遷移時
1回実行
START
daq_start()
STOP
daq_stop()
コマンド
RUNNING
daq_run()
PAUSE
daq_pause()
RESUME
daq_resume()
状態中
繰り返し実行
PAUSED
daq_dummy()

DAQコンポーネントのコマンドループ

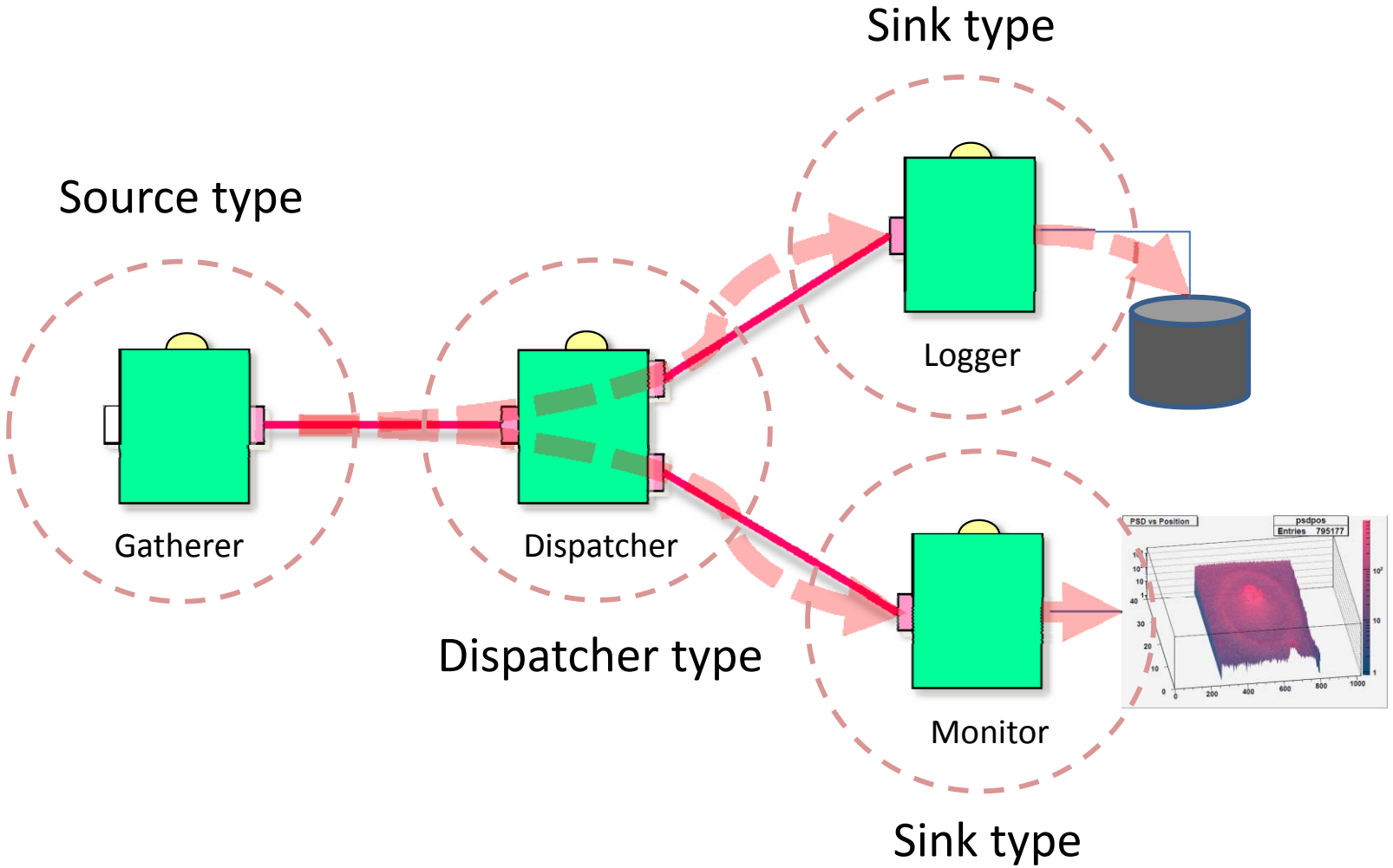
- コンポーネントは、*Active*状態ではonExecute() が定期的に呼び出される
- onExecute()からdaq_do()が呼び出される
  ↖コマンドループ処理
  - CONFIGURE状態で、Startコマンドを受けるとdaq_run()が呼ばれ（RUNNING状態）、StopコマンドかPauseコマンドを受けるまで繰り返し呼ばれる
  - その他の状態では アイドル状態を実現する daq_dummy() が呼ばれる

daq_run(), daq_dummy()の中では、タイムアウトなしでブロックするような処理を行ってはいけない

DAQコンポーネントのタイプ

- コンポーネント間のデータストリームに対して
  - Source型
    - データストリームの起点となるコンポーネント
    - OutPortをもつ。InPortをもたない
    - Gathererコンポーネント（MLF）
  - Sink型
    - データストリームの終点となるコンポーネント
    - InPortをもつ。OutPortをもたない
    - Logger, Monitor（MLF）

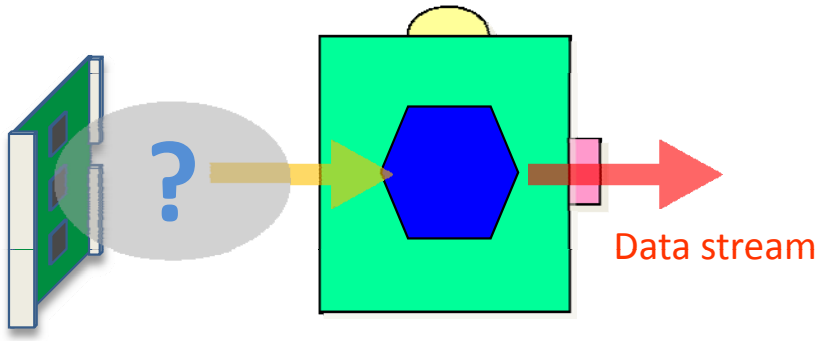
DAQコンポーネントのタイプ



Source型DAQコンポーネントの開発

- Source型の代表的なコンポーネントはリードアウトモジュールからデータを読み出すもの
- リードアウトモジュールの種類・読み出し方法は多様であり一般化は困難

ユーザによるデータ読み出しの実装が必要



参考文献

- Source型コンポーネント開発マニュアル
  - http://greentea.kek.jp/daqm/docs/source-comp.pdf
  - リードアウト・ハードウェアなしでデータを取得し、後段のコンポーネントへ送信

EchoReaderコンポーネント

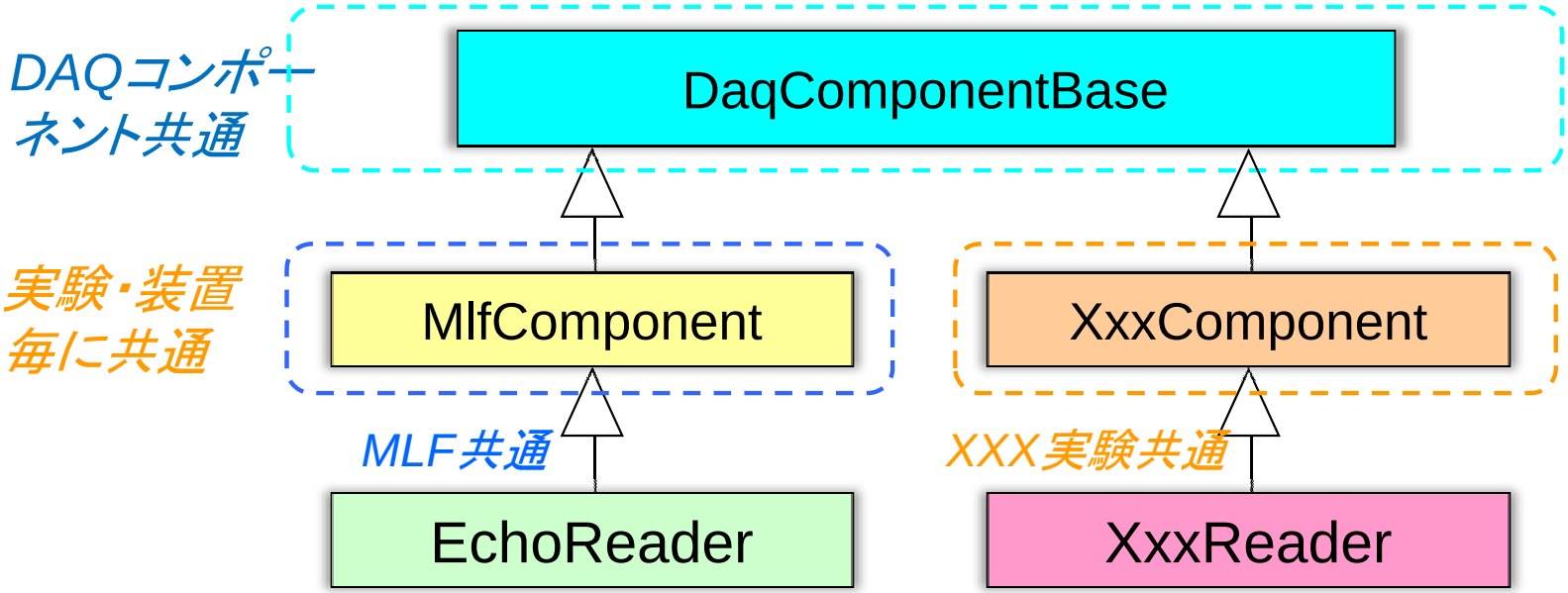
- ハードウェアがなしでSource型DAQコンポーネントの開発をするための例題
- echo server へ接続してデータを送信し、データを受信して、OutPortから送信する

※TCP ベースのエコーサービス:

エコーサービスは TCP に基づくアプリケーションの接続として定義される。サーバは TCP 接続のため TCP ポート 7 で待機する。いったん接続が確立したら受信したどんなデータも送り返す。これは呼び出したユーザが接続を終了するまで続ける。

EchoReaderコンポーネント仕様

- MLF中性子用DAQ-Middleware2009.07版の中のSkeletonコンポーネントを利用して、EchoReaderコンポーネントを開発する



EchoReaderコンポーネント仕様

- 正規分布データを生成する。100event(4byte/event)をechoサーバへ送信
  - 一様乱数から生成（平均値60、標準偏差10）
- Echoサーバからデータ（400byte）を受信
- 受信したデータにヘッダ、フッタをつけOutPortのバッファへコピーする
- OutPortから送信する
- OutPortの送信ステータスを調べる
- エラーの場合はDAQオペレータへ通知する

開発手法

- DAQ-Middleware for MLF 2009.07に入っているSkeletonコンポーネント（Skeleton.h, Skeleton.cpp, SkeletonComp.cpp）をもとに必要な機能を追加する
- Gathererコンポーネントから不要な機能を取り除く

今回はSkeletonコンポーネントをもとにした開発手順を紹介する

開発手順(1)

- DAQ-Middleware for MLF 2009.07版の
  - Skeleton.{h, cpp}　⇒ *ヘッダファイル、実装*
  - SkeletonComp.cpp ⇒ *コンポーネントの生成を行う*
  - Makefile.Skeleton ⇒ *コンポーネントのMakefile*
- 名前を変更する
  - EchoReader.{h, cpp}
  - EchoReaderComp.cpp
  - Makefile.EchoReader

4種類のDAQコンポーネント開発に必要なファイル

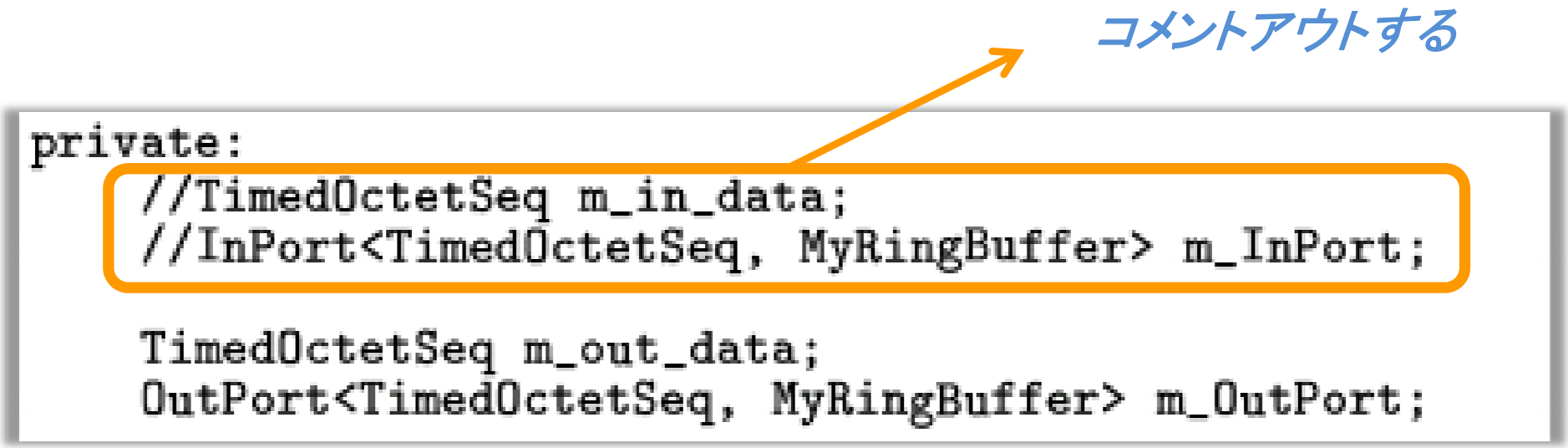
開発手順(2)

- 次のファイル中の文字列を置換する
  - EchoReader.{h, cpp}
  - EchoReaderComp.cpp
  - Makefile.EchoReader

```
$ for i in *Skeleton*; do
    sed -i.bak -e 's/Skeleton/EchoReader/g'
               -e 's/skeleton/echoReader/g'
               -e 's/SKELETON/ECHOREADER/g'  $i;
    done
```

開発手順(3)

- InPortをdisableにする
  - EchoReader.h



開発手順(4)

- InPortをdisableにする
  - EchoReader.cpp

```
EchoReader::EchoReader(RTC::Manager* manager)
    : DAQMW::MlfComponent(manager),
      ///m_InPort("echoReader_in",   m_in_data),
      m_OutPort("echoReader_out", m_out_data),

      ///m_in_status(BUF_SUCCESS),
      m_out_status(BUF_SUCCESS),

      m_debug(false)
{
    // Registration: InPort/OutPort/Service

    // Set InPort buffers
    ///registerInPort ("echoReader_in",   m_InPort);
    registerOutPort("echoReader_out", m_OutPort);

    init_command_port();
    init_state_table( );
    set_comp_name(COMP_NONAME);
}
```

開発手順(5)

- ここでビルドを行う

```
$ cd your_directory/DaqComponents
$ make -f Makefile.EchoReader
```

- コンパイルが成功すれば、メインロジックが空のEchoReaderが完成
- 安心してメインロジックを実装する

開発手順(6)

- メインロジックは空だが、コマンドによる状態遷移は動作する
- マニュアルのP.8のようにコマンドラインからコマンドを発行して状態が変化するか確認する
- EchoReader を使用するためのコンフィグレーション・ファイルを用意する(次スライド)

開発手順(7)

```
<?xml version="1.0"?>
<configInfo>
  <daqOperator>
    <hostAddr>127.0.0/1</hostAddr>
  </daqOperator>
  <daqGroups>
    <daqGroup gid="group0">
      <components>
        <component cid="EchoReader0">
          <hostAddr>127.0.0.1</hostAddr>
          <hostPort>50000</hostPort>
          <instName>EchoReader0.rtc</instName>
          <execPath>/home/daq/DaqComponents/bin/EchoReaderComp</execPath>
          <confFile>/home/daq/DaqComponents/rtc.conf</confFile>
          <startOrd>1</startOrd>
          <inPorts>
          </inPorts>
          <outPorts>
          </outPorts>
          <params>
          </params>
        </component>
      </components>
    </daqGroup>
  </daqGroups>
</configInfo>
```

メインロジックの実装

```
int EchoReader::daq_run()
{
    send_data_to_echoServer();
    recv_data_from_echoServer();

    set_data_to_OutPortBuf(m_loop);
    m_out_status = m_OutPort.write(m_out_data); /// send data to next
    if (check_outPort_status(m_out_status) == -1) {
        std::cerr << "### EchoReader: OutPort.write(): FATAL ERROR\n";
        fatal_error_report(USER_ERROR3, -1);
        return 0;
    }

    if (check_trans_lock() ) {  /// got stop command
        set_trans_unlock();
        return 0;
    }

    usleep(100000); /// sleep for 100ms to adjust histogram update time
    return 0;
}
```

set_data_to_OutPortBuf()

- OutPort のバッファにデータをコピーする

```
int EchoReader::set_data_to_OutPortBuf( unsigned int seq_num )
{
    unsigned int addr  = 0;
    unsigned int daqId = 0;
    unsigned int modNo = 0;

    unsigned char header[8];
    unsigned char footer[8];

    set_header(header, daqId, modNo, m_dataByteSize);
    set_footer(footer, addr, seq_num);

    ///set OutPort buffer length
    m_out_data.data.length(m_dataByteSize + HEADER_BYTE_SIZE + FOOTER_BYTE_SIZE);

    memcpy( &(m_out_data.data[0]), header, HEADER_BYTE_SIZE);
    memcpy( &(m_out_data.data[HEADER_BYTE_SIZE]), &m_rdata[0], m_dataByteSize);
    memcpy( &(m_out_data.data[HEADER_BYTE_SIZE + m_dataByteSize]), footer, FOOTER_BYTE_SIZE);

    return 0;
}
```

check_outPort_status()

- 送信ステータスのチェック

```
int EchoReader::check_outPort_status(int status)
{
    int ret = 0;

    if (status == BUF_SUCCESS) {
        m_loop++;
        m_total_event += (m_dataByteSize/m_eventByteSize);
        if (m_debug) {
            if (m_loop%100 == 0) {
                std::cerr << "EchoReader: m_loop = " << m_loop << std::endl;
                std::cerr << "\033[A\r";
            }
        }
    }
    else if (status == BUF_TIMEOUT) {
        std::cerr << "EchoReader::Time Out occurred..." << std::endl;
    }
    else if (status == BUF_FATAL) {
        std::cerr << "### EchoReader::Fatal error occurred..." << std::endl;
        ret = -1;
    }
    return ret;
}
```

Fatal_error_report()

```
int fatal_error_report(CompFatalTypes types, int code)
```

- DaqComponentBase.h
- Fatal エラーをDAQオペレータへ通知して、コンポーネントは待機状態になる
- Fatalエラーを受けた上位システム、ユーザが状況を判断して対処する(ラン停止、システムの再立ち上げ、ハードウェアのチェック)

Fatal_error_report()

- Fatal エラーのタイプは DAQService.idl 中で定義している
- ユーザが利用できるのは、USER_ERROR{1..4}
- 今後、柔軟に追加できるように検討中

```
enum CompFatalTypes
{
    HEADER_DATA_MISMATCH,
    FOOTER_DATA_MISMATCH,
    SEQUENCE_NUM_MISMATCH,

    USER_ERROR1,
    USER_ERROR2,
    USER_ERROR3,
    USER_ERROR4,
    UNKNOWN_FATAL_ERROR
};
```

まとめ

- 「Source型コンポーネント開発マニュアル」をもとに重要と思われる項目を説明しました
- 実際に上記マニュアルを見ながら、自分でEchoReaderコンポーネントを作って動かしてみてください
- マニュアルの不備や質問、要望があればお知らせください

予備

DAQコンポーネントのInPort/OutPort

- InPort<DataType, Buffer>
  - read(): DataPortから値を読み出す
- OutPort<DataType, Buffer>
  - write(const DataType &value): データ書込み
- BufferBase<DataType>
  - virtual long int length(): バッファの長さの取得

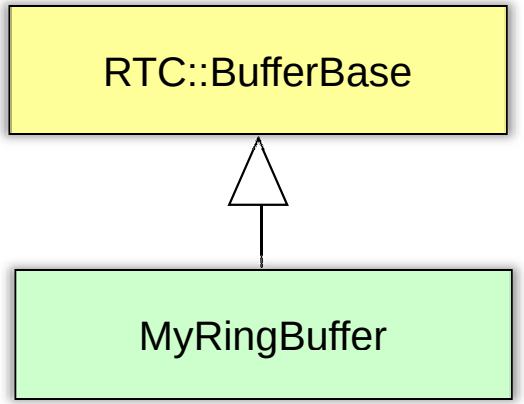
InPort/OutPortへのバッファの指定

- コンポーネント・ヘッダファイル

```
private:
    //TimedOctetSeq m_in_data;
    //InPort<TimedOctetSeq, MyRingBuffer> m_InPort;

    TimedOctetSeq m_out_data;
    OutPort<TimedOctetSeq, MyRingBuffer> m_OutPort;
```

rtm/idl/BasicDataType.idl で
定義されている標準型
（8bit 配列型）



バッファの長さ

- コンストラクタで初期化の際に指定

```
EchoReader::EchoReader(RTC::Manager* manager)
    : DAQMW::MlfComponent(manager),
      ///m_InPort("echoReader_in",   m_in_data),
      m_OutPort("echoReader_out", m_out_data),

      ///m_in_status(BUF_SUCCESS),
      m_out_status(BUF_SUCCESS),

      m_debug(false)
```

```
m_InPort("echoReader_in", m_in_data, 64)
```

※MLF用2009.07版では、InPortのバッファの数は128