DAQ 実習
-DAQ のからくりと実装-

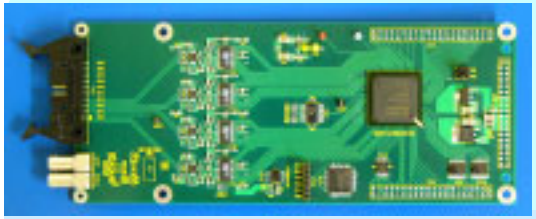
五十嵐 洋一
KEK

概要

- 実習0) Hardware を理解する。
  - Hardware(COPPER)概説
  - DAQ Software の構成
- 実習1) DATA を読んでファイルに書き出す。
  - OS から hardware の read-out
- 実習2) Network を通した data の読み込み
  - 実習1) を network server 化する。
  - Network client
- 実習3) DATA のデコードと表示
  - DATA format を理解する。
  - ROOT を使い file を読んで DATA を表示する。
- 実習4) オンラインディスプレイ
  - 実習3) をオンラインディスプレイ化する。
- 実習5) 製作したものを統合して動かす
- 補講
  - 制御について
  - Network based DAQ software の紹介

実習 0
ハードウエアを理解する
KEK-VME COPPER 概説

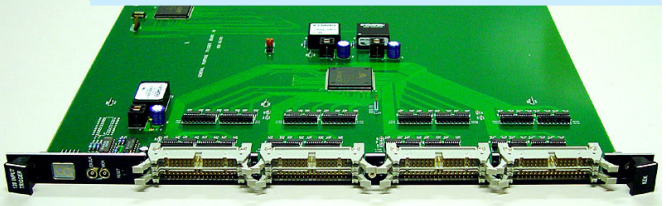
KEK-VME System Families

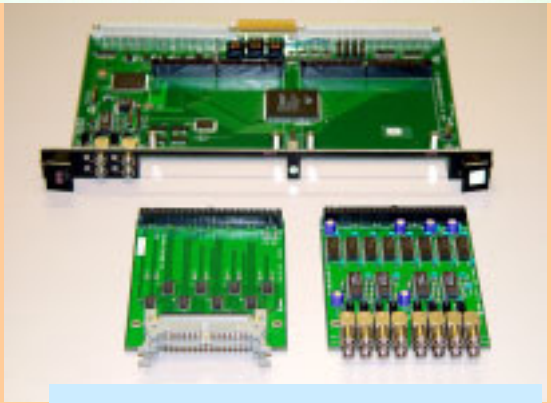


Front-end daughter card
- 500MHz, 8bit FADC
- 65MHz, 12bit FADC
- TMC
- ...

KEK-VME crate
VME crate with
extended power supply

128 input Trigger module





KEK-VME general I/O
(NIM/ECL/LVDS)

Read-Out module

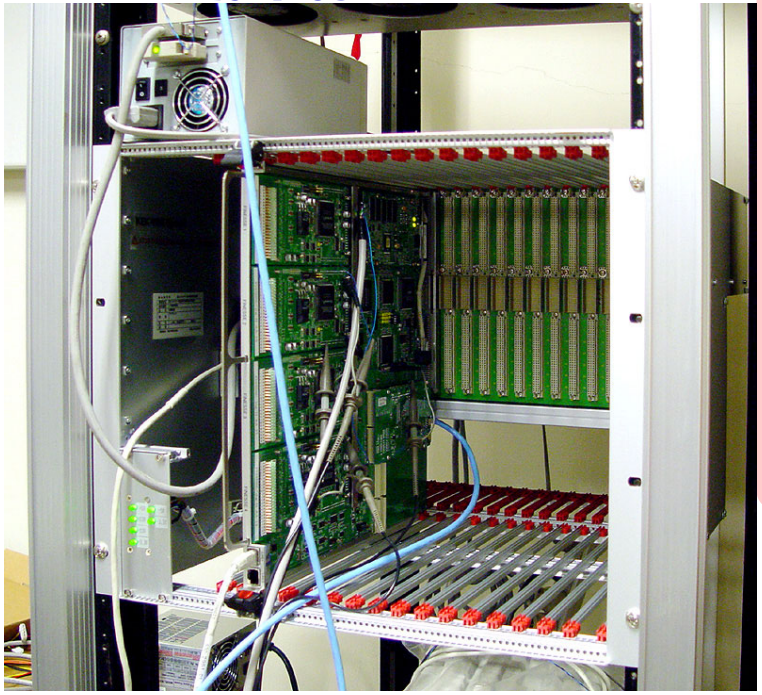
KEK-VME G.G/C.G.

Read-out part

Trigger part
(Replace for NIM module)

Crate and Power Supply

- Euro card/crate
  - 9U and 6U
  - VME-32 bus
- J0 Connector for Power Supply
  - Treat front-end analog to digital conversion devices



Pin assignment of J0

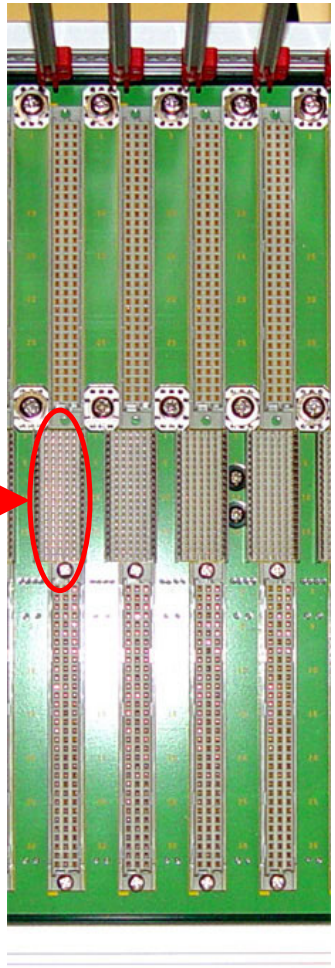
Pos.	z	a	b	c	d	e	f
1	GND	GND	GND	GND	GND	GND	GND
2	GND	GND	GND	GND	GND	GND	GND
3	GND	GND	GND	GND	GND	GND	GND
4	GND	+3.3V	+3.3V	+3.3V	+3.3V	+3.3V	GND
5	GND	+3.3V	+3.3V	+3.3V	+3.3V	+3.3V	GND
6	GND	+3.3V	+3.3V	+3.3V	+3.3V	+3.3V	GND
7	GND	+3.3V	+3.3V	GND	GND	GND	GND
8	GND	GND	GND	GND	GND	GND	GND
9	GND	GND	GND	GND	GND	GND	GND
10	GND	GND	GND	GND	-3.3V	-3.3V	GND
11	GND	-3.3V	-3.3V	-3.3V	-3.3V	-3.3V	GND
12	GND	-3.3V	-3.3V	-3.3V	-3.3V	-3.3V	GND
13	GND	GND	GND	GND	GND	GND	GND
14	GND	-5V	-5V	-5V	-5V	-5V	GND
15	GND	GND	GND	GND	GND	GND	GND
16	GND	S1+	S1-	GND	S2+	S2-	GND
17	GND	S3+	S3-	GND	S4+	S4-	GND
18	GND	S5+	S5-	GND	S6+	S6-	GND
19	GND	S7+	S7-	GND	C1	C2	GND

+3.3V

-3.3V

-5.0V

Voltage	-5.0V	-3.3V	+3.3V
Total Max Current	100A	320A	200A

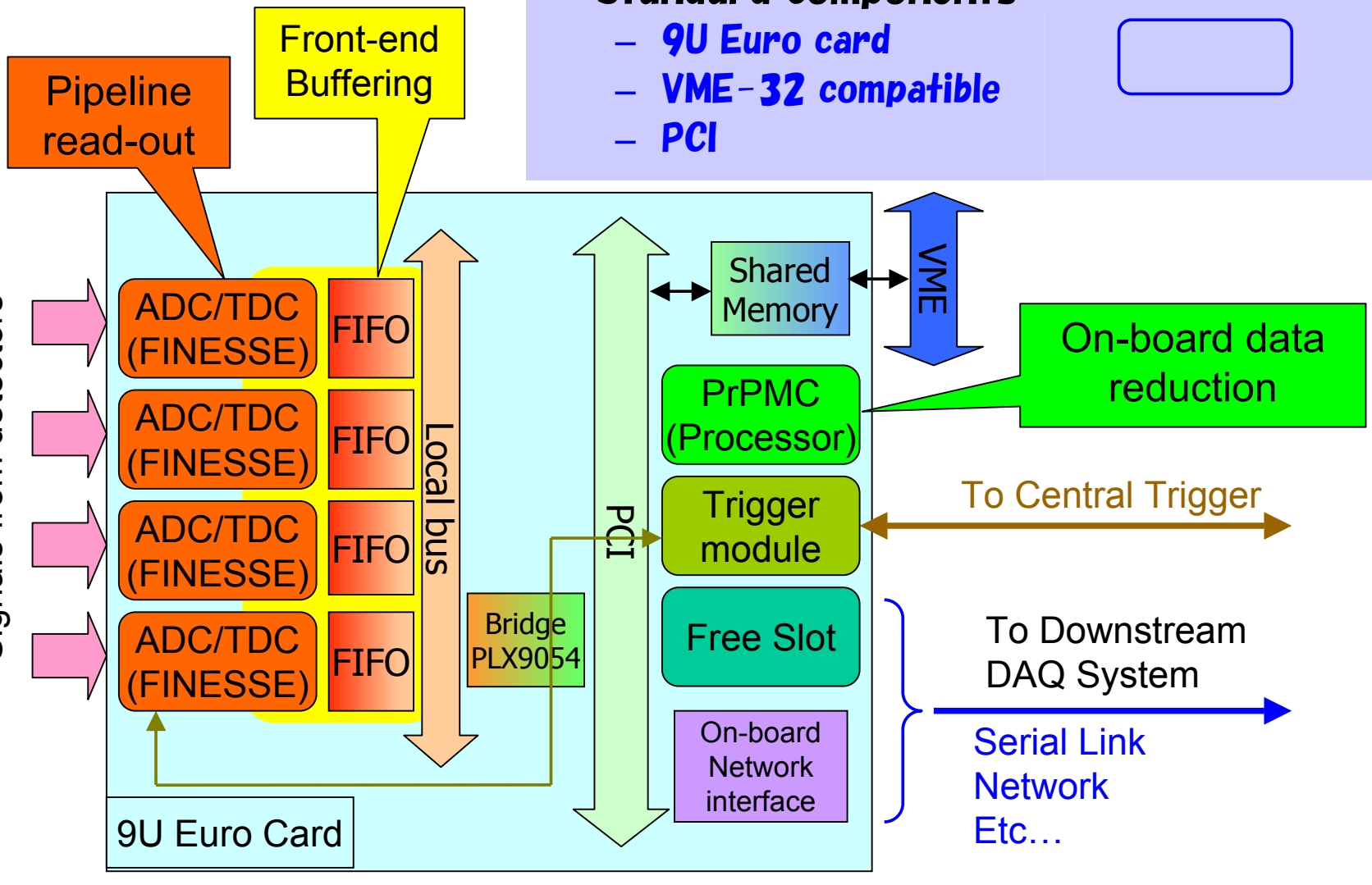


A low noise power supply was developed. (1/10 less than standard VME power supply)

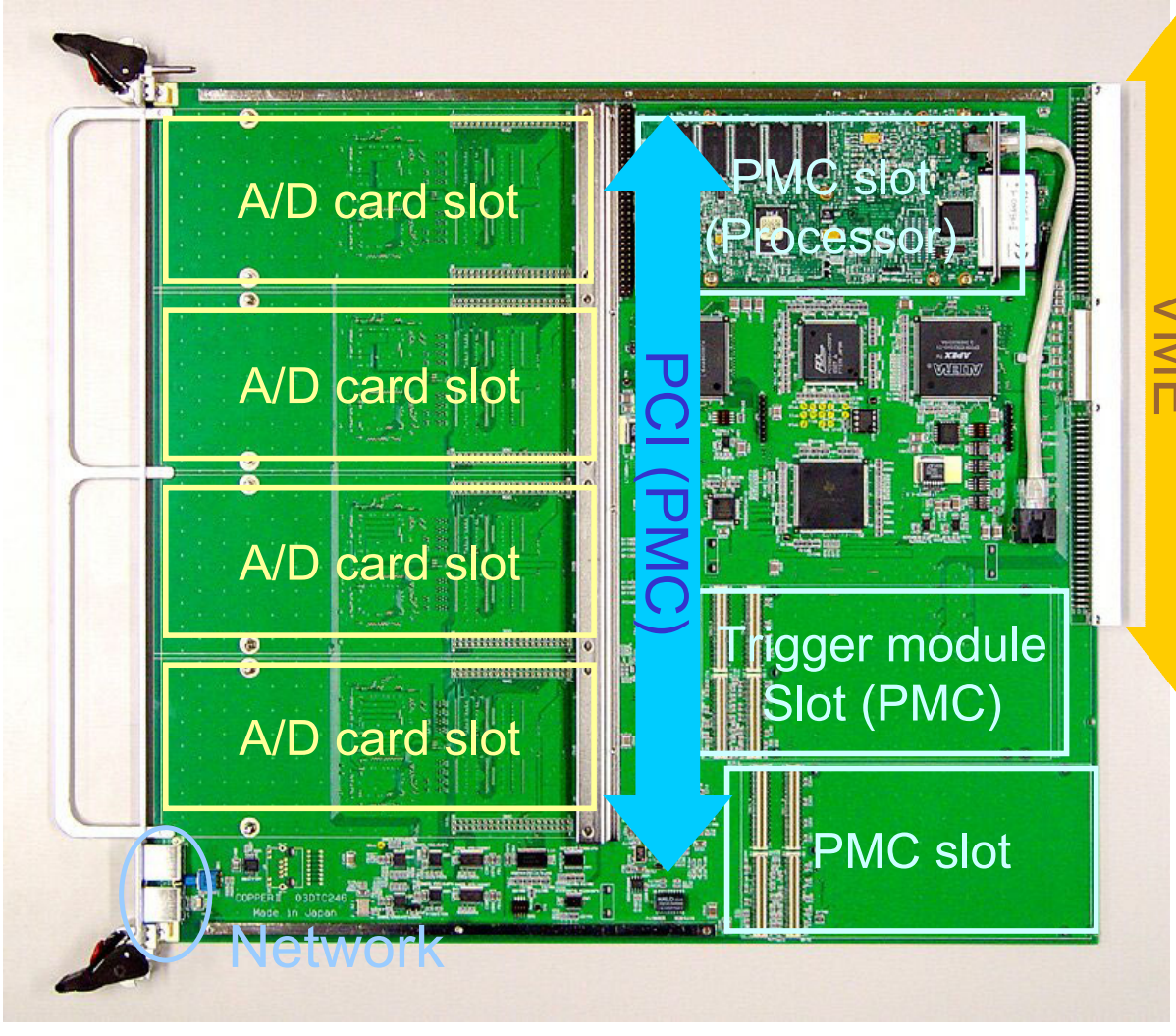
Schematic view of a read-out module

- **Standard components**
  - **9U Euro card**
  - **VME-32 compatible**
  - **PCI**

•Module structure



Read-out module (COPPER)



- 9U euro card(VME)
- 4 Front-end A/D card slot
- Processor PMC slot
- Trigger module slot
- general PMC slot
- VME-32 interface
- 1MB x 4 FIFO
- 32bit 33MHz PCI bus
- 2 network interface
  - Processor module
  - On-board NIC

PCI/PMC

- PCI Mezzanine Card
  - PCI compliant
  - Several modules are available at a reasonable price.
    - Processor (PPC/x86/…), 100Base/Gigabit Ethernet, IEEE1394, Memory, Etc….
- PrPMC EPC
  - PC architecture
  - Standard Linux 2.4

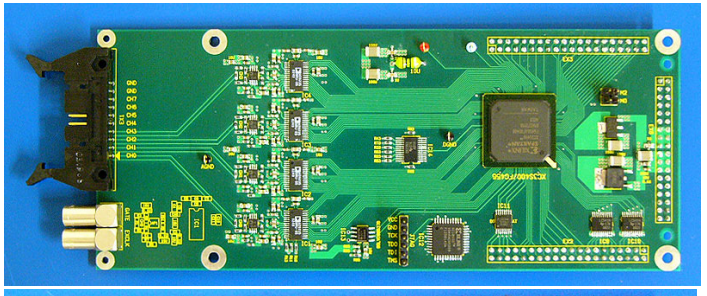
Ramix PMC610
4port Ethernet card

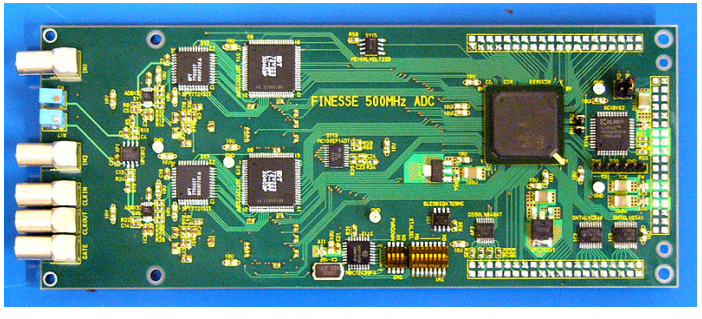
Radisys EPC-6315

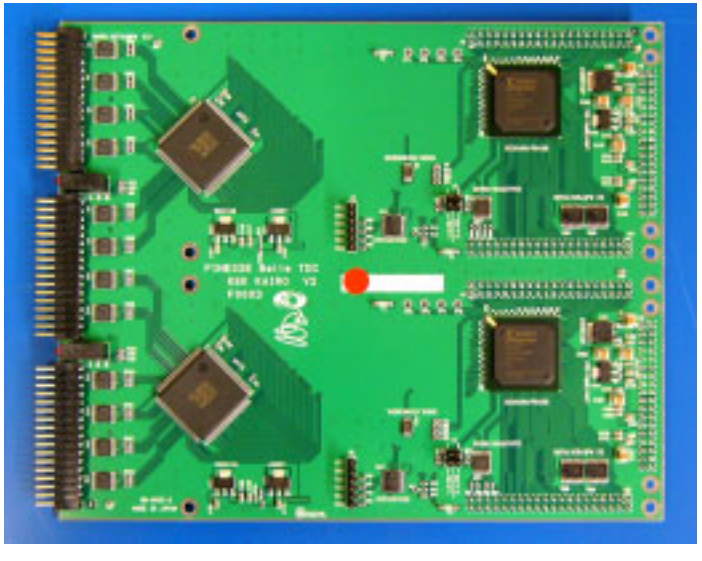
- 800 MHz Pentium IIIm
- Up to 512 MB SDRAM with ECC.
- 10/100 BaseT Ethernet port
- On-board Compact Flash socket.
- 32-bit 33/66 MHz PCI bus interface.

Front-end daughter card (FINESSE)

- AMT based Pipeline TDC
  - Time Memory Cell : AMT3
  - 24ch or 48ch(2 slot) LVDS signal
  - Timing resolution : 0.78 ns/bit (40MHz clock)
  - Dynamic range :17 bit
- FPGA based Pipeline TDC
  - 16 ch nomal pitch connector version
  - 32 ch high density connector version
    - 1 GHz counting
    - 256 depth
    - Dynamic range :16 bit
- Flash ADC
  - 8bit/500MHz sample, 2ch
    - Signal range : ±0.5V
    - 250MHz ADC x 2 x 2
  - 12bit/65MHz sample, 8ch
    - Differential input(±1V)
- Sampling encoder for MWPC
  - 32 ch high density connector
  - 100MHz sampling



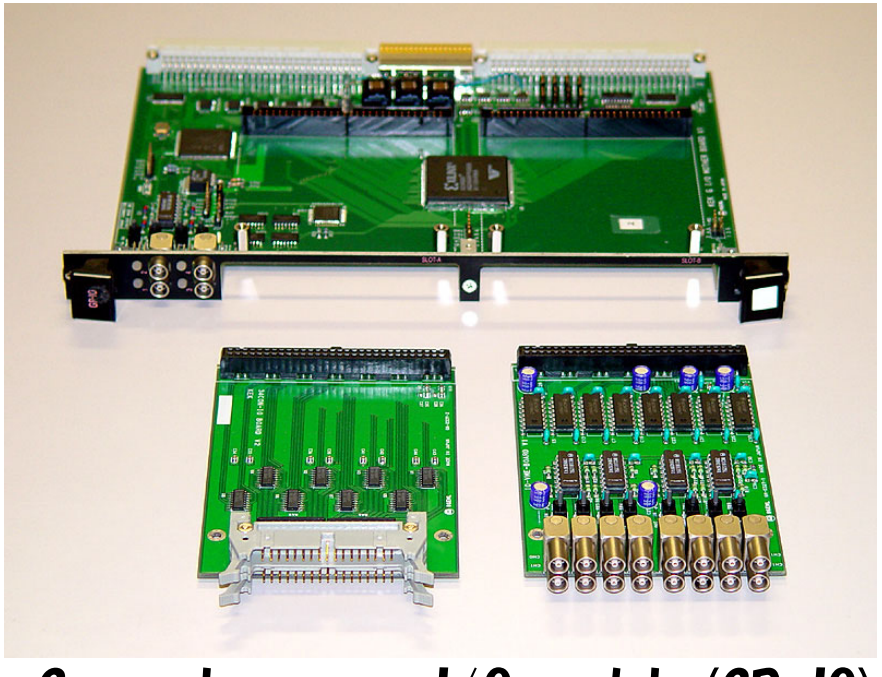




KEK-VME General Purpose FPGA board

- Successors of NIM logic modules

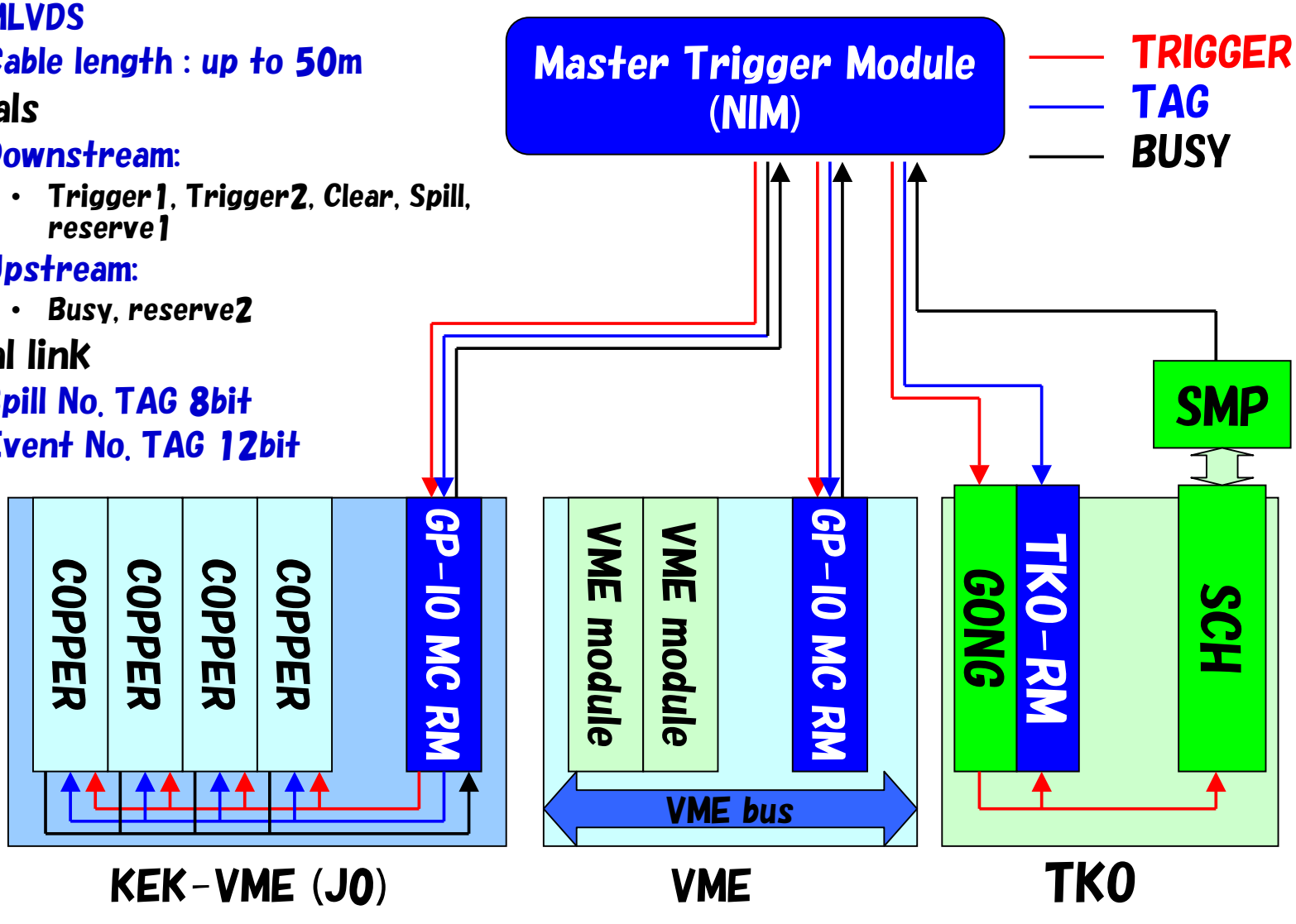
- GP-IO Mezzanine Cards
  - NIM driver card
  - LVDS/ECL/TTL driver card
  - 65MHz ADC card
  - 65MHz DAC card
  - Optical TX/RX card
  - Trigger Receiver card



General purpose I/O module (GP-IO)

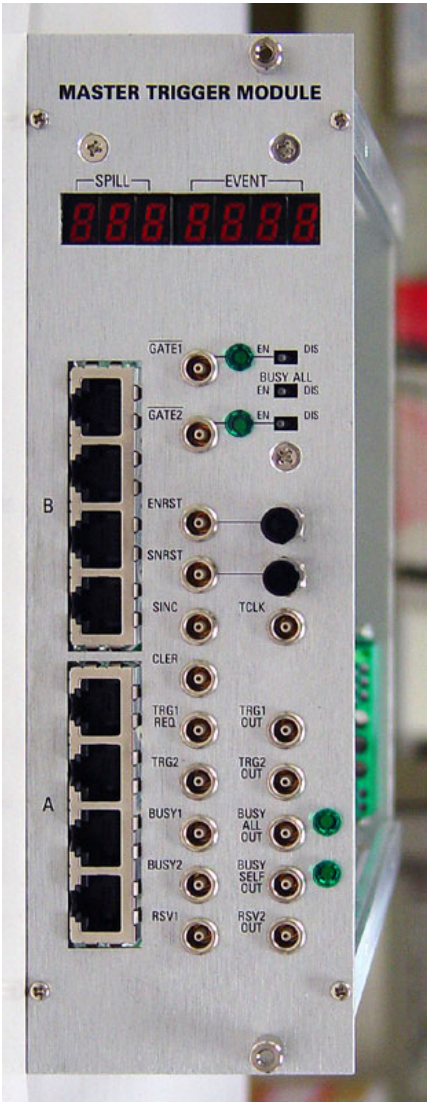
A Trigger/Tag Distribution System

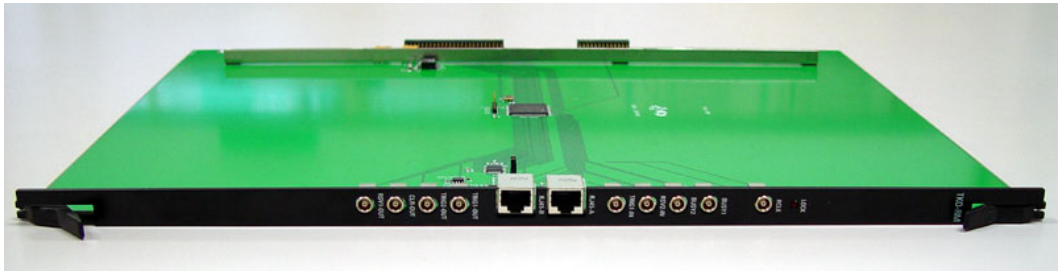
- Category 5 cable x 2
  - MLVDS
  - Cable length : up to 50m
- Signals
  - Downstream:
    - Trigger1, Trigger2, Clear, Spill, reserve1
  - Upstream:
    - Busy, reserve2
- Serial link
  - Spill No. TAG 8bit
  - Event No. TAG 12bit



Trigger/Tag Distribution Modules

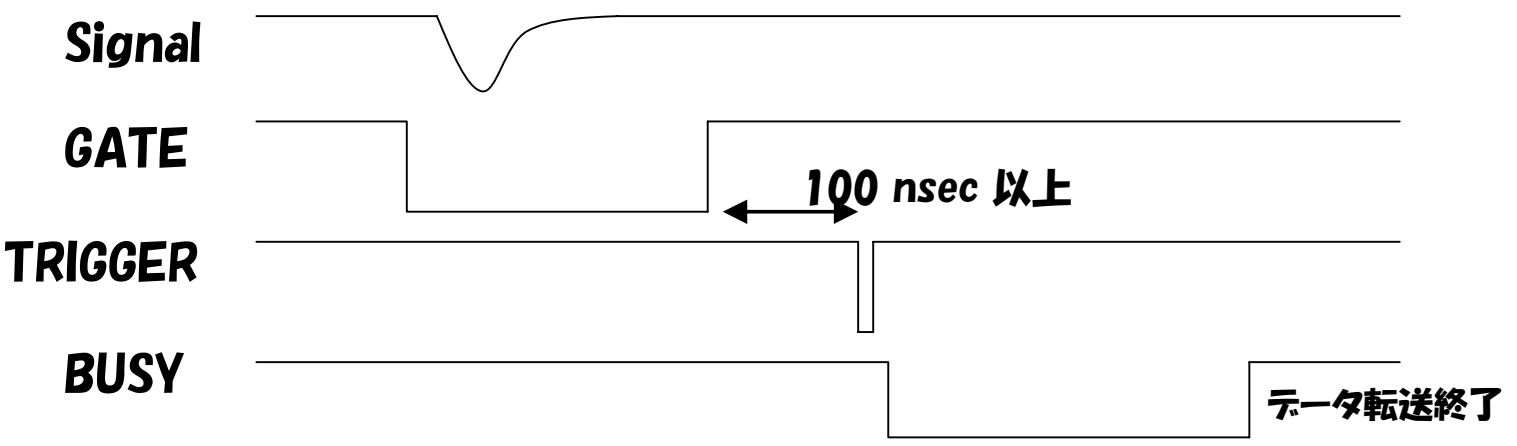
- NIM Master Trigger Module
  - 4 Transfer port
    - 2 RJ-45 port for 1 RM
  - 8bit Spill Counter
  - 12bit Event Counter
- GP-IO Receiver Module
  - Receive Trigger/TAG from MTM
  - Send BUSY signal to MTM
  - Export TAG info. to COPPERs
- TKO Receiver Module
  - Receive Trigger/TAG from MTM
  - Send BUSY signal to MTM
  - Export TAG info. to TKO bus





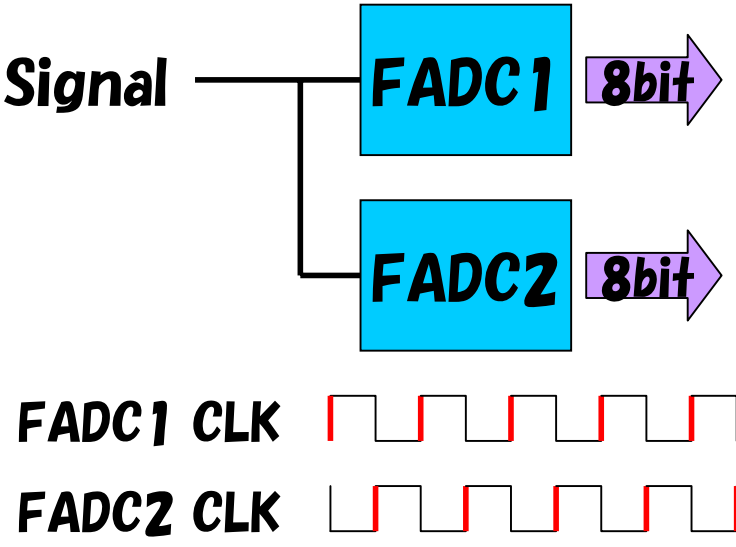
FINESSE の動作

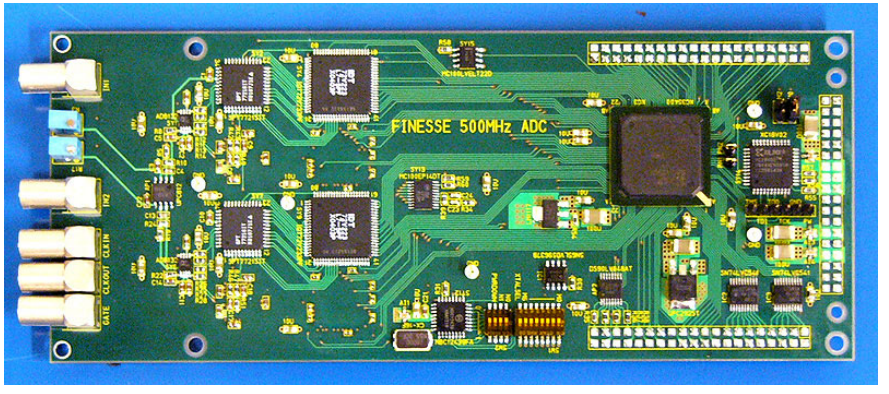
- FINESSE 基本
  - プロセッサ部とは独立にデータを収集している。
  - COPPER から TRIGGER がくると 溜め込んだデータを COPPER の FIFO に 1 event のデータとしてデータを書き込む
- 500 MHz FADC の動作
  - フロントパネルの GATE がアクティブの間 FADC のデータを内部 FIFO に溜め込む。
  - COPPER から供給される TRIGGER 信号で内部 FIFO のデータを COPPER の FIFO に転送する。



500 MHz FADC

- 波形の電位をそのまま ADC で A/D 変換してサンプルしていく。
- 500MHz sample, Dynamic range 8bit
  - 250MHz sample FADC を半周期位相をずらして使用。
- 2ch, 50ohm, ±1 V





COPPER の動作

- FINESSE とのインターフェースをとる2種類の FIFO
  - EVENT FIFO : Data 入っている。
  - LENGTH FIFO : 1 Event の長さが入っている。
  - FINESSE SLOT ごとに 4 対ある。
- 2 種類の FIFO の状態により割り込みが発生させることが可能。
- 割り込みが発生すると DMA をかけてデータをプロセッサのメモリに転送を行う。
  - この部分はデバイスドライバが受け持つ

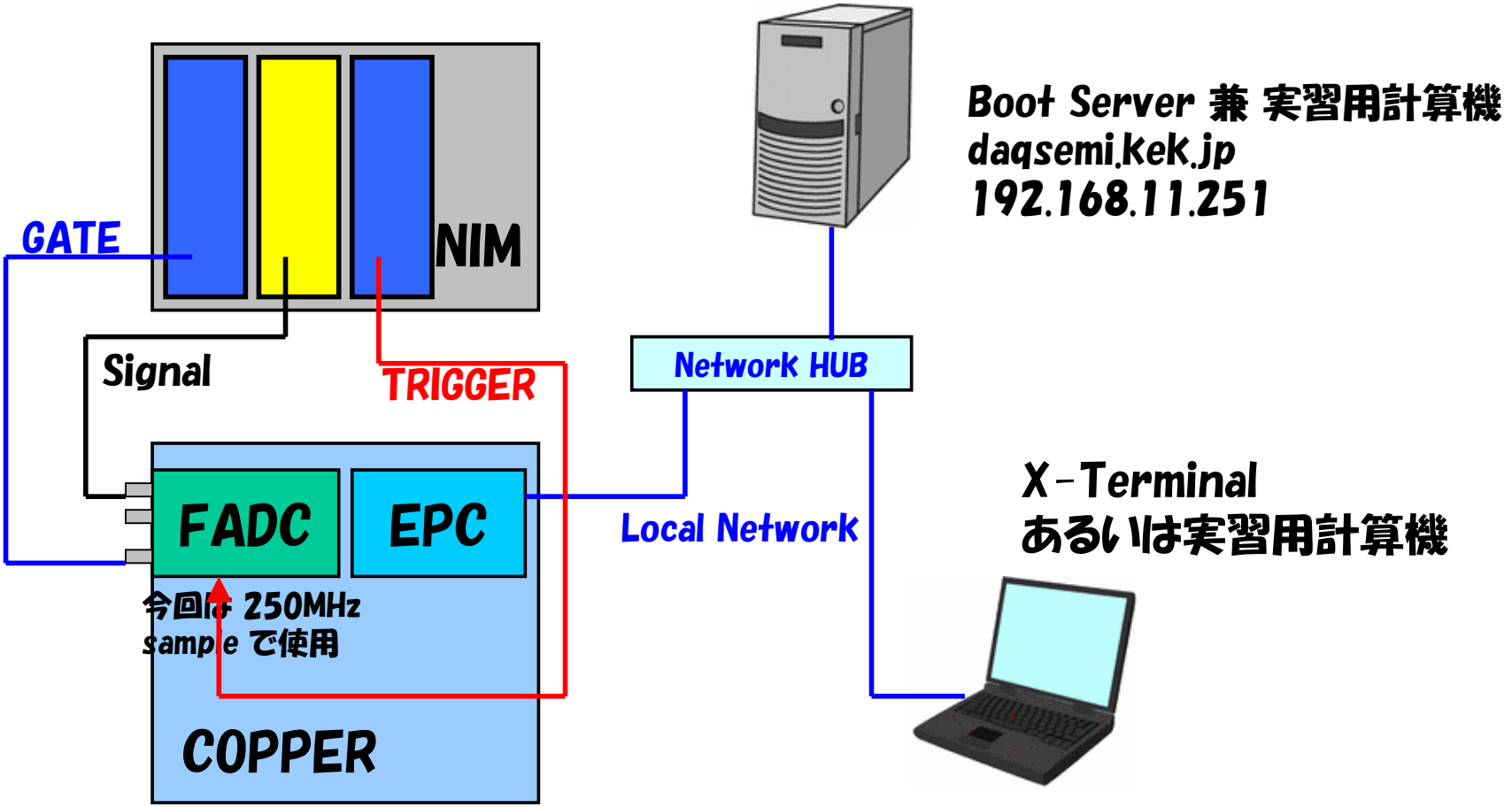
  → ユーザプログラムが受け取る。

COPPER のデータ転送開始条件

- EVENT FIFO FULL
- EVENT FIFO ALMOST FULL
- EVENT FIFO がある値を超えたとき
- LENGTH FIFO FULL
- LENGTH FIFO ALMOST FULL
- LENGTH FIFO がある値を超えたとき
- 以上の組み合わせで割り込みをかけられる。

- 通常は LENTH FIFO 1 以上でデータ転送開始で OK
- 実習環境ではドライバが設定してくれている。

実習のセットアップ


GATE
NIM
Signal
TRIGGER
Network HUB
Boot Server 兼 実習用計算機
daqsemi.kek.jp
192.168.11.251
FADC
EPC
Local Network
X-Terminal
あるいは実習用計算機
今回は 250MHz
sample で使用
COPPER

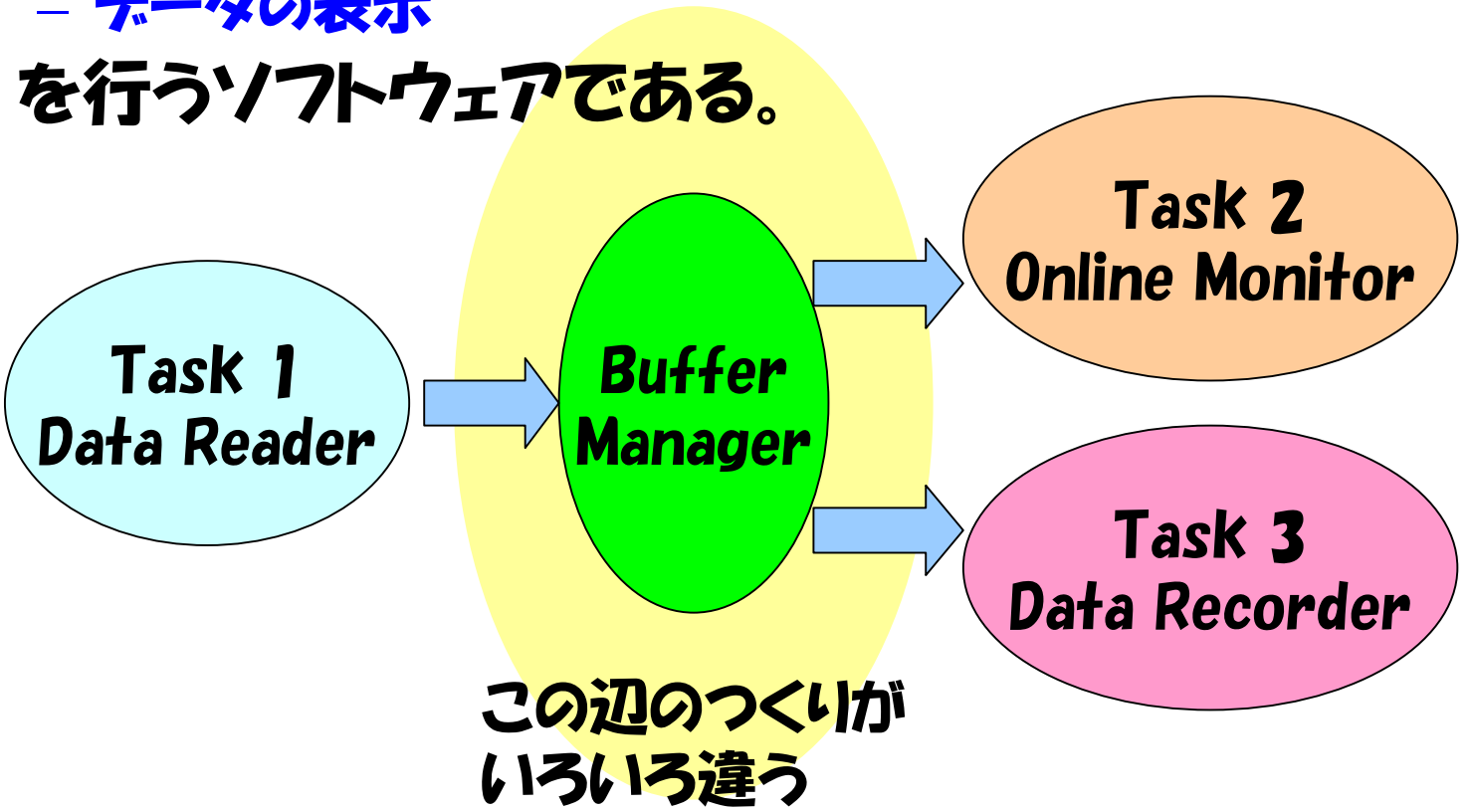
粒子線計測における DAQ の動作について

- 最近では必ずしも当てはまらないが大まかには次のシーケンスをとる。
  1. Event の発生 → Trigger の出力
     - Trigger 回路, NIM 標準モジュールなどで作る。
  2. Front-end に Gate やタイミング等の入力
     - 主に Trigger 回路 から Event 発生時のタイミングをもらう。
  3. Read-out module がデータ収集を開始
     - ハードウェアに合せた方法で開始する。
  4. 計算機がデータを読み込む

DAQ Software の構成

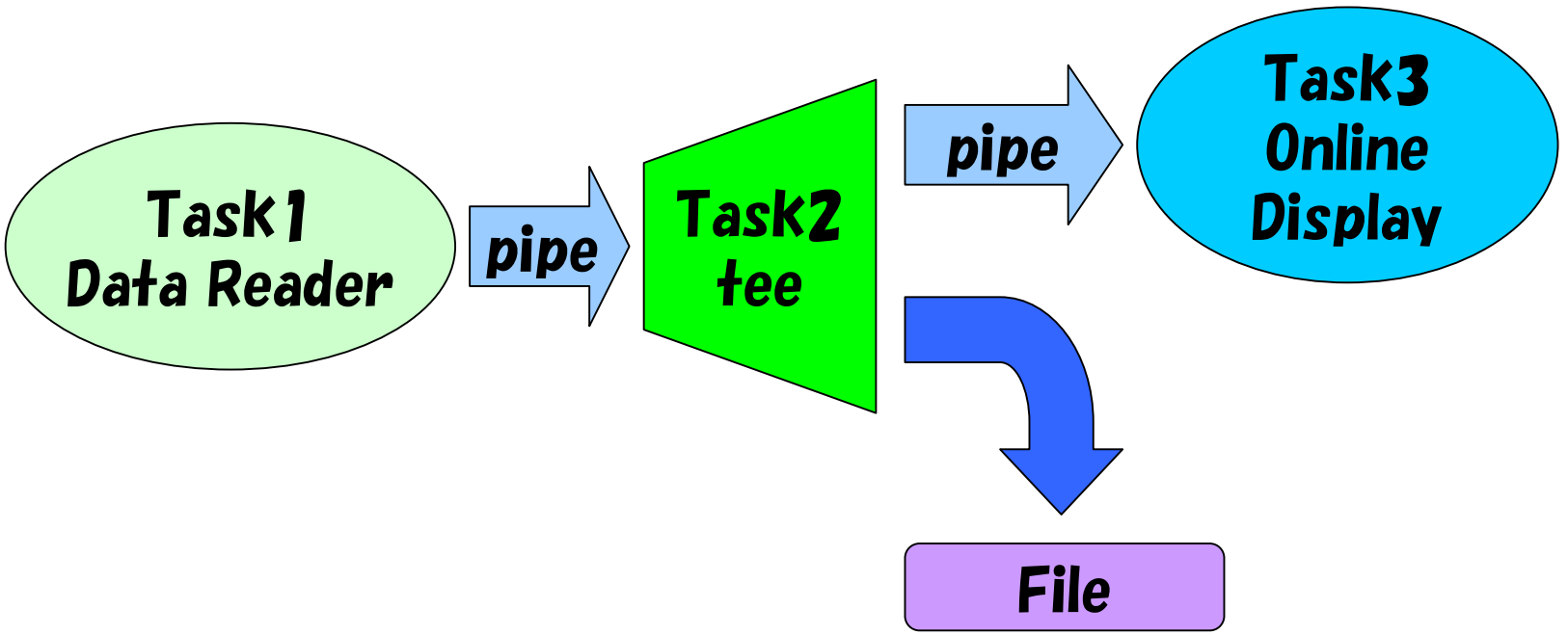
- DAQ Software とは？
  - 機器からのデータの読み込み
  - データの記録
  - データの表示

  を行うソフトウェアである。



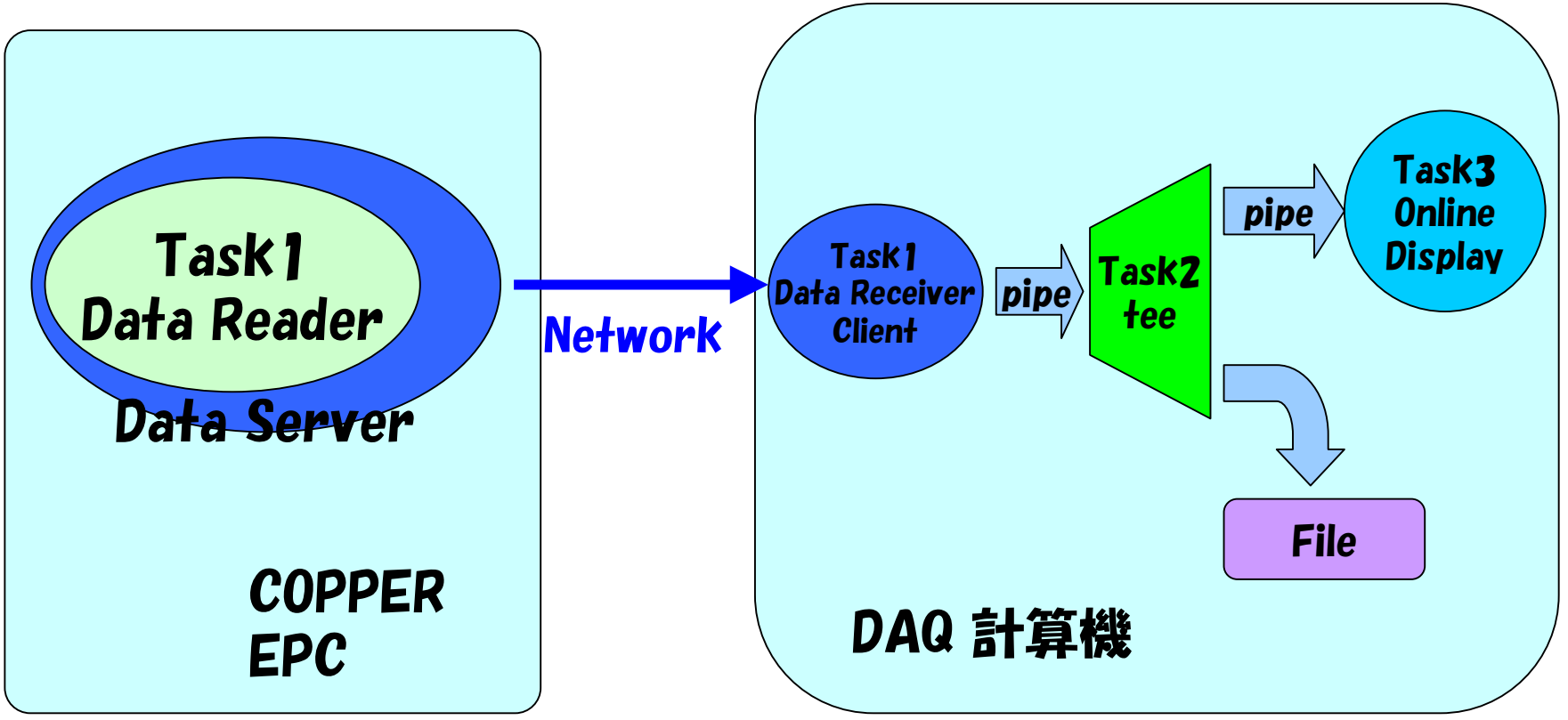
Minimum DAQ Software の構成

- 最小のコードで DAQ software を書いてみる。
  - Process 間通信には 標準入出力+ pipe を使用。
    - エラーメッセージなどを出したいときは標準エラー出力を利用する。
  - バッファリングには 標準入出力のものを使用する。
  - データの記録は Redirection を利用する。
  - データのオンラインモニタと記録双方への振り分け → tee



今回の DAQ Software の構成

- COPPER と DAQ 計算機の間でネットワークによる通信がある。



使用している計算機環境の確認

- どの環境を使用するか決めよう。
- Helloworld のコンパイルは出来ますか?

```
#include <stdio.h>
int main()
{
    printf("Hello World¥n");
    return 0;
}
```

- Linux, MacOSX
  - ターミナルを開いて 作業ディレクトリに cd
  - $ cc helloworld.c
  - $ ./a.out
- Windows
  - Start Menu から Visual Studio の menu のなかにある コマンドプロンプトを起動する。
  - 作業ディレクトリに cd
  - >cl helloworld.c
  - >helloworld
- 環境変数 ROOTSYS の確認

Local Network IP address

- daqsemi : 192.168.11.251, daqsemi.kek.jp
- copper2 : 192.168.11.4
- copper14 : 192.168.11.16
- copper6 : 192.168.11.8
- copper8 : 192.168.11.10
- copper9 : 192.168.11.11
- copper10 : 192.168.11.12
- copper11 : 192.168.11.13
- copper12 : 192.168.11.14
- copper13 : 192.168.11.15

- Local Network では name service が動いていません。IP アドレスをじかに使用してください。
- 残念なことに COPPER は9台しかありません。
- 2人で一つの COPPER を譲り合って使ってください。

実習 0.5 遊んでみよう

- 全体を把握するためにサンプルを動かす。
- COPPER に slogin
  - ワークディレクトリを作りそこに移動
  - $ wget http://192.168.11.251/~igarashi/oncopper.tar.gz
  - $ zcat oncopper.tar.gz
  - $ cd oncopper
  - $ ./cserver
  - ペアの人と譲り合って実行してください。
- DAQ 計算機で (daqsemi` あるいは皆さんのノートパソコン)
  - ワークディレクトリを作りそこに移動
  - http://192.168.11.251/~igarashi/ondisplay.zip を持ってくる。
  - $ unzip ondisplay.zip
  - $ make –f Makefile.linux (nmake –f Makefile.win32 など)
  - データを記録する。
    - $./cli 192.168.11.xx 8888 > datafile
  - データをオンラインで表示する。
    - $./cli 192.168.11.xx 8888 | ./onladc
  - データを記録しながらオンライン表示を行う。
    - $./cli 192.168.11.xx 8888 | tee datafile | ./onladc

- うまく動いたでしょうか？

- なんとなくイメージがつかめたらファイルを消して 実習 1 に臨みましょう。

進め方

- Hello world が速やかに出来た人は出来るだけスライド及び補足資料をみてプログラムを作成してください。
- 時間がかかったひとはサンプルプログラムを見たり改造したりしてみてください。
- どうしても時間が足り無そうな場合はサンプルプログラムを make しましょう。(Windows の場合は nmake)
- Makefile はそれなりに面倒なのでサンプルを見ながら作ったほうが良いと思います。
- サンプルがダサいと思った人は自由に作って OK です。特に表示するものは自分で考えて好きなものを表示しましょう。
- ROOT は思わぬ挙動を示すことがあります。メモリマネージメントは特に気をつけましょう。
- http://192.168.11.251/~igarashi/ にサンプルがあります。必要に応じてコピーして使用してください。

実習 1
Data を読んで file に書き出す。

- COPPER 上でプログラムを書く
  - Device driver の読み出し
  - ファイルの書き込み (stdout)

- 課題: COPPER からデータを読んでヘッダつきのイベントデータとして標準出力に出力する。

UNIX 系 OS での Device の触り方

1. Device file を open する。
2. Device の初期化、設定
   - ioctl でレジスタの設定
   - Linux の場合は /proc/ .... で設定する場合もある。
3. Data のアクセス
   - Data の read/write を行う。
   - mmap をしてメモリアクセスをする。
   - Ioctl で付加情報(レジスタ)のアクセスを行う場合もある。
4. Device の close
   - mmap した場合は munmap もする。

Event header をつけよう

- データの read をやりやすくする。
- データが何であるかわかるようにする。
- 例

```
struct event_header {
        unsigned int magic; (0x45564e54)
        unsigned int size;
        unsigned int event_number;
        unsigned int run_number;
        unsigned int node_id;
        unsigned int type;
        unsigned int nblock;
        unsigned int reserve;
};
```

main()

```
int main()
{
変数初期化
    status = init_device();
    while (1) {
        status = wait_device(); ← 今回のドライバでは何もしない。
        status = read_device(data, &nread);
        dsize = nread + sizeof(struct event_header)/sizeof(unsigned
    int);
        header->event_number = event_number;
        header->size = dsize;
        status  = fwrite(buf, sizeof(unsigned int), dsize, stdout);
        if (status < nread) {
                perror("fwrite");
                break;
        }
        fprintf(stderr, "¥r e: %d      ", event_number);
        event_number++;
    }
    finalize_device();
    return 0;
}
```

Init()

```
void reset_fifo_and_finesse(int cprfd)
{
        int val;
        val = 0x1F; ← 全てのFINEESEとCOPPER に Reset
        ioctl(cprfd, CPRIOSET_FF_RST, &val, sizeof(val));
        val = 0;
        ioctl(cprfd, CPRIOSET_FF_RST, &val, sizeof(val));
}
#define DEVCOPPER "/dev/copper/copper"
int init_device()
{
        int status;
        fd = open(DEVCOPPER, O_RDONLY);
        if (fd == -1) {
                perror("open_device");
                return -1;
        }
        status = ioctl(fd, CPRIO_INIT_RUN, 0); ← ドライバの初期化
        reset_fifo_and_finesse(fd);

        return status;
}
```

read()

```
int read_device(unsigned int *data, int *len)
{
        int status;

        status = read(fd, (char *)data, MAX_DATASIZE * sizeof(int));
        if (status == 0) {
                perror("read_device, got EOF");
                return 0;
        }
        if (status < 0) {
                perror("read_device");
                return status;
        }
        *len = status/sizeof(int);

        return status;
}
```

finalize()

```
int finalize_device()
{
        ioctl(fd, CPRIO_END_RUN, 0);
        close(fd);
        return 0;
}
```

ioctl のフラグはデバイスドライバのインクルードファイル
　copper.h に定義されています。
コピーしてインクルードしてください。

動かしてみよう

- Terminal を開いて COPPER に slogin
- プログラムを作成する。
- make する。

- $ ./creader > datafile
- $ od –t x4z | less

- $ ./creader | od –t x4z | less

- ヘッダやデータのようなものが見えるでしょうか?
- 実行するときはペアの人と譲り合ってください。

実習 2
Network を通した dataの読み込み

- 実習1 で作ったものを server 化 する。
- Network Client を作成。
- 藤井さんの講義を参考に
- 課題:
  - 実習1 のプログラムを network server 化する。
  - Network からデータを読んで標準出力に出力する。

Network Server 化

- 実習1 で作成したデータを読むプログラムを network server にする。
- それなりに面倒なので ライブラリを利用する。
  - Kolc library (http://www-online.kek.jp/~keibun/netprog/tcplib.htm)
    - 手間を減らすだけの単純なライブラリなのでわかりやすく使いやすい。
- 手順
  1. 受付 socket を作る (とりあえず port 番号は 8888)
  2. 受付 socket に接続が来たら転送 socket をつくり受付 socket に来た接続に繋げる。
  3. 今回は接続は1つか受け付けないので受付 socket は close する。
  4. データ読んで標準出力の代わりに転送 socket にデータを書き込む
  5. 書き込みに失敗したら終了

main()

```
int main(int argc, char* argv[])
{
    変数初期化
    SOCKET ss;
    SOCKET sa;
    netlibstart();
    ss = tcpsocket();
    tcplisten(ss, PORT, 5);
    if ((sa = accept(ss, 0, 0)) != INVALID_SOCKET) {
        tcpclose(ss);
        status = init_device();
        while (1) {
            status = wait_device();
            status = read_device(data, &nread);
            dsize = nread + sizeof(struct event_header)/sizeof(unsigned int);
            header->event_number = event_number;
            header->size = dsize;
            status  = tcpwrite(sa, (char *)buf, sizeof(unsigned int) * dsize);
            if (status < dsize * sizeof(unsigned int)) {
                perror("tcpwrite");
                break;
            }
            fprintf(stderr, "¥r e: %d     ", event_number);
            event_number++;
        }
        tcpclose(sa);
    }
    finalize_device();
    netlibstop();
    return 0;
}
```

Server を立ち上げ続ける Script

```
#!/bin/sh
while true
do
  ./cserver
  sleep 1
done
```

Run ごとにプログラムを終了させると OS が資源の解放を行ってくれる。

Network Client

- ネットワークからデータを読んで標準出力に出力する。
- Linux や MacOSX では nc (net cat)がある。
- 実習であるし Windows などのために一応 cli.c を作ってみた。
  - Windows の場合の注意
    - Windows では file open の mode に binary がある。
    - Default では text モードのため binary data を流すと開業コードなどでデータ化けが発生する。
      - 標準入出力に _setmode() をかけてモードを変更する。
    - Windows の network socket インターフェースは WinSock と呼ばれかない方言がある。
- kolc library のサンプルをみれば簡単に作れる。

Network Client を作る。

- kolc から exam02.c をコピーする。
- テキストにダンプしているところを
- fwrite(...,stdout) などに書き換える。

exam02.c

```
netlibstart();
s = tcpsocket();
tcpopen(s, argv[1], port);
while((n = tcpread(s, buf, MYBUFSIZE)) > 0)
{
        buf[n] = 0;          ← このへん
        printf("%s", buf);   ←
}
tcpclose(s);
netlibstop();
```

動かしてみよう

- COPPER で server を動かす。
- $ ./cli 192.168.11.xx 8888 | od –t x4z | less
- データの保存
- $./cli 192.168.11.xx 8888 > datafile.dat

- ヘッダやデータのようなものが見えるでしょうか?

実習3 Data file を読んで表示を行う。

- 山形さんの講義を参考にしよう。
- 課題:
  1. 標準入力を読み込んで data をdump
  2. 標準入力を読み込んで data のデコード
  3. ROOT を使い、標準入力を読み込んでヒストグラムの表示
  - 初日の方々がここで体力を消耗したので2日目の課題はサンプルを利用します。
    1. decoder.cxx をコンパイルして動かす
    2. ヒストグラムに工夫を加えて別の情報を書き出そう
       - 例: 波形を積算して電荷情報のヒストグラムをつくるのは?

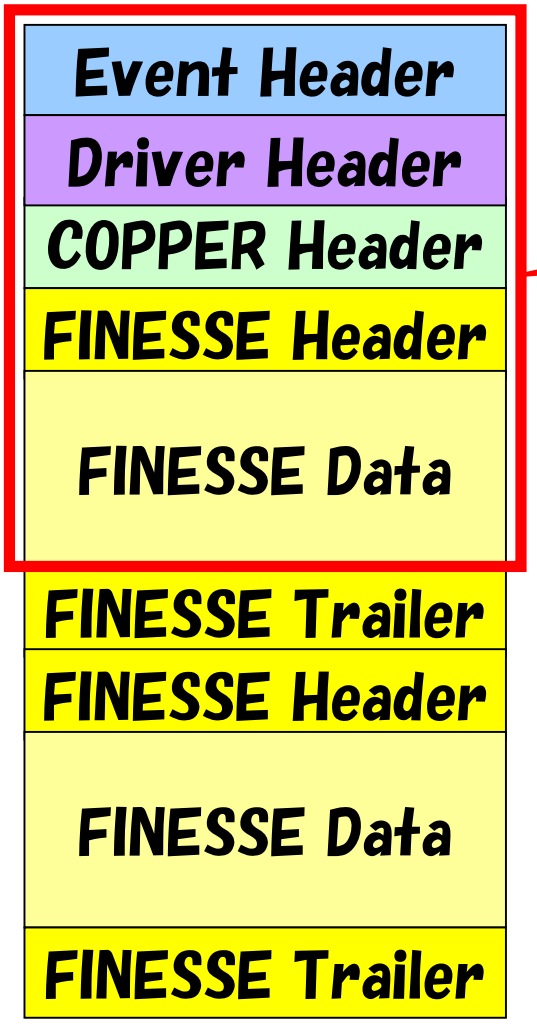
データの読み込み方の例

1. std::cin.read で固定長 header を読み込む。
2. header からデータ長を読み取り、残りのデータを読み込む。
3. データをデコード
4. ヒストグラムにフィルする。
5. データを読み終わったらデータを表示してみる。

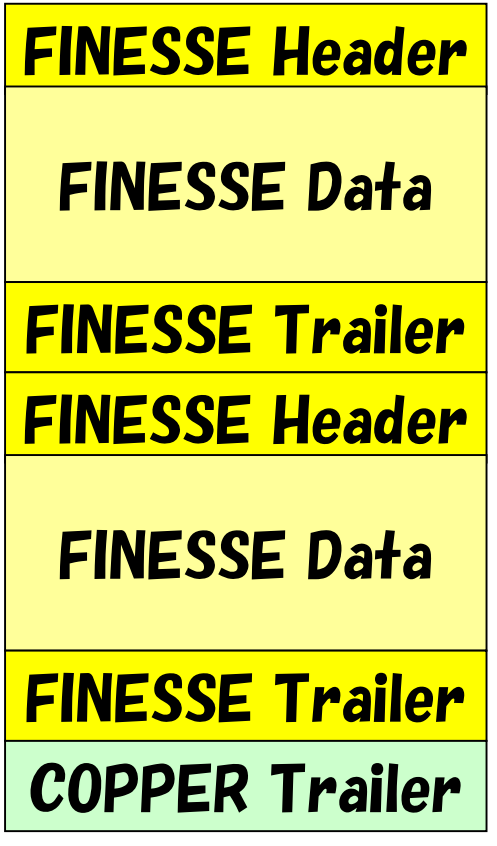
標準入力の読み込み

```
while (true) {
    std::cin.read(cbuf, sizeof(struct event_header));
    if (std::cin.eof()) break;
    int nread = eheader->size * sizeof(unsigned int) -
    sizeof(struct event_header);
    assert(eheader->magic == 0x45564e54);
    std::cin.read(cbody, nread);
    if (std::cin.eof()) break;
    if (std::cin.gcount() < nread) break;
    decodefadc500(cbuf, data, &ndata);
    std::cout << "#DD " << std::dec << ndata << " :";
    for (int i = 0 ; i < ndata ; i++) std::cout << " " << data[i];
    std::cout << std::endl;
}
```

Data format



例ではここだけデコード



- 今回は FINESSE は1枚なので FINESSE の block はひとつ。
- デコードに関しては資料と山形さんの講習を参照

Software Headers

EVENT header

```
struct event_header {
        unsigned int magic: (0x45564e54)
        unsigned int size;
        unsigned int event_number;
        unsigned int run_number;
        unsigned int node_id;
        unsigned int type;
        unsigned int nblock;
        unsigned int reserve;
};
```

Device Driver Header

```
struct cpr_header {
        unsigned int magic;
        unsigned int event_number;
        unsigned int subsys;
        unsigned int crate;
        unsigned int slot;
        unsigned int ttrx[2];
};

struct cpr_footer {
        unsigned int chksum_xor;
        unsigned int magic;
};
```

Module headers

COPPER header

```
Struct copper_header {
    Unsigned int Magic: (0xfffffafa)
    Unsigned int Totalsize:
    Unsigned int Finesse_size[4]:
}
Struct copper_footer {
    Unsigned int footer: (0xfffff5f5)
}
```

FINESSE header

```
Struct finesse_header {
        unsigned int magic: (0xffaa0000)
        unsigned int tag:
}
Struct finesse_footer {
        unsigned int footer: (0xff55xxxx)
}
```

データにヘッダー構造体を貼っていく

```
char *cbody = buf + sizeof(struct event_header);
cprheader *cheader = reinterpret_cast<cprheader *>(cbody);
finheader *fheader = reinterpret_cast<finheader *>(cbody +
    sizeof(cprheader));
fadc500data *fdata = reinterpret_cast<fadc500data *>(cbody
        + sizeof(cprheader) + sizeof(finheader));
unsigned int *rawdata = reinterpret_cast<unsigned int *>(cbody
        + sizeof(cprheader) + sizeof(finheader));

assert(cheader->hw_magic == 0xfffffafa);  ← magic number check
assert(fheader->magic == 0xffaa0000); ←
```

時間がある人は trailer もチェックしよう

デコード

```
int ndata = (cheader->fin_words[0] - 3) / 2;
*nsample = ndata * 4;
assert(*nsample < 1024);
for (int i = 0 ; i < ndata ; i++) {
    data[i*4+0] = fdata->ch0data0;
    data[i*4+1] = fdata->ch0data1;
    data[i*4+2] = fdata->ch0data2;
    data[i*4+3] = fdata->ch0data3;
    fdata++;
}
```

この例は ch0 だけデコード。時間がある人は ch1 にも対応できるように拡張しよう。

ヒストグラム

- ROOT でヒストグラムを作るには ？
  - TH1F hist1("name1", "title1", nbin, min, max); ← 1次元ヒストグラム
  - TH2F hist2("name2", "title2", xnbin, xmin, xmax, ynbin, ymin, ymax); ← 2次元ヒストグラム
- ヒストグラムに値を積むには
  - hist1.Fill(x);
  - hist2.Fill(x, y);

ヒストグラムをフィル

大域で

```
static TApplication *app = new TApplication("App", NULL, NULL);
static TStyle *t1 = new TStyle();
static TCanvas *c1 = new TCanvas("c1", "Online Display");
static TH2F *fadc[2];
```

Main で初期化

```
fadc[0] = new TH2F("FADC00", "FADC00", 256, 0, 256, 256, 0, 256);
fadc[1] = new TH2F("FADC01", "FADC01", 256, 0, 256, 256, 0, 256);
t1->SetPalette(1);
c1->Divide(1, 2);
c1->cd(1);
fadc[0]->Draw("col");
c1->cd(2);
fadc[1]->Draw("col");
c1->Update();
c1->Flush();
```

データをデコードした場所で

```
for (int i = 0 ; i < ndata ; i++)  fadc[0]->Fill(i, data[i], 1);
```

ファイルを読み終わったら UI を実行

```
c1->cd(1)->Modified();
c1->cd(2)->Modified();
c1->Update();
app->Run(); ← TApplication
```

Sample decoder の使い方

- 推奨はしませんが，時間がかかりそうな場合sample decoder を使うことも出来ます。

```
#define UNPACKER_NOMAIN
#include "unpacker.cxx"

DCopper copper1;
DFinesseFadc500 adcfinesse;
copper1.set_finesse(&adcfinesse, NULL, NULL, NULL);
```

- データの読み込み
- Event_header をスキップして COPPER data の先頭をポイント

```
copper1.set_data(copperdata);

unsignd int *data, ndata;
data =copper1.get_data(slot, ch, &ndata);
```

- でスロット slot チャンネル ch のデータ配列の先頭ポインタと要素数 ndata がもらえます。

動かしてみよう

- 標準入力からデータを入れる。
- $ ./cat datafile | ./decoder

実習4 オンラインディプレイ

- 課題:
  - 実習3 で作ったプログラムをオンラインディスプレイで使用できるようにする。
    - データを読んでヒストグラムに積む。
    - 描画
    - メニューなど UI

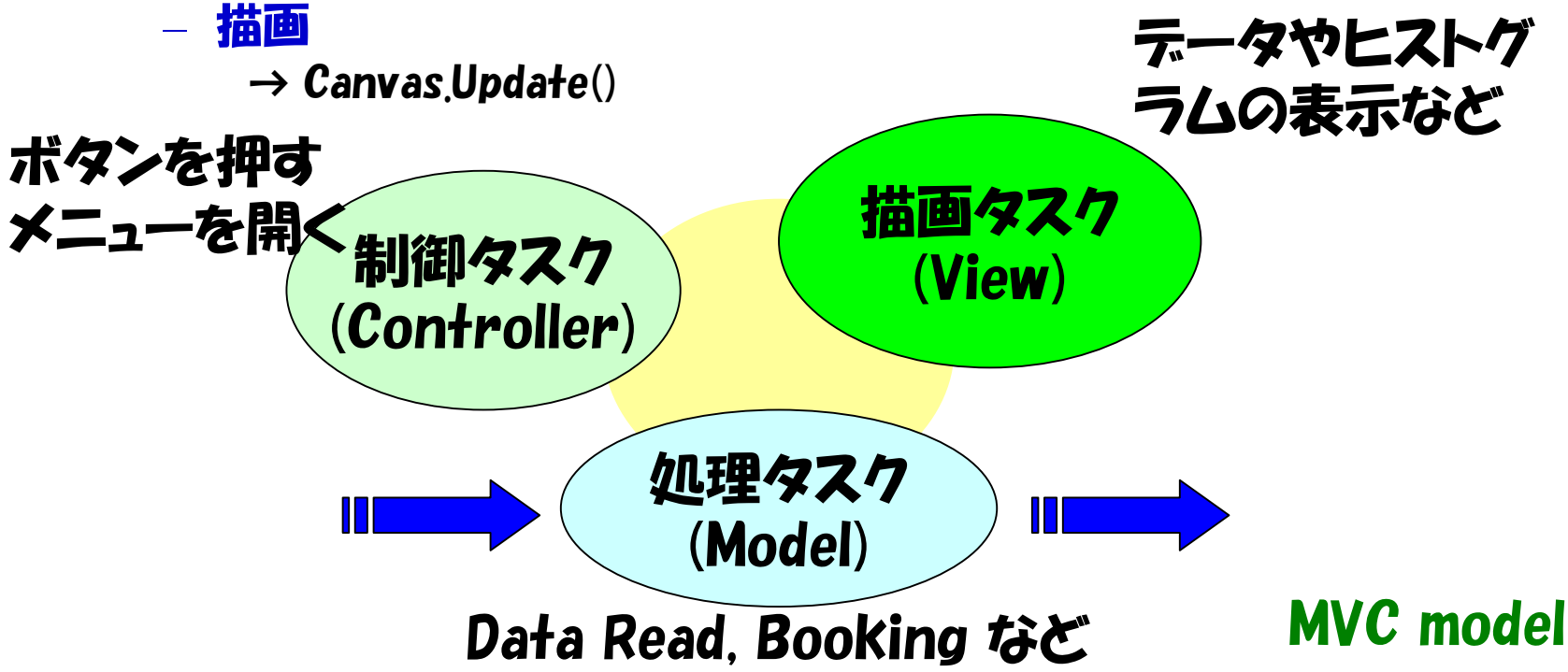
    を同時に動くようにする。

Online Display

- Online Display
  - ヒストグラムナビゲートとヒストグラムのフィルを同時に行う。
  - 長時間にわたり動き続ける必要がある。
    - 一定資源(特にメモリ)で動き続けなければいけない。
    - Tree は不向き 使用する場合は資源を消費しない使い方をすること。
    - とりあえず固定サイズのヒストグラムを固定数作る
      - FADC なので 2次元ヒストグラムに積んでいくのではどうか?
      - 数秒に一回くらいイベントディスプレイもよいかも?

Online Display の構成

- 制御と処理は非同期に行われる。
  - UI と 処理部
    - 処理の最中に UI が止まってしまうとよろしくない。
- ROOT では
  - ボタンを押すメニューを開く
    → TApprication
  - 描画
    → Canvas.Update()



Online Display main()

```
Thread 間共有資源の宣言
int main()
{
    初期化, オブジェクトの生成
    画面を作る
    Reader Thread を実行
    描画 Thread を実行
    TApplication.Run()
    return 0;
}
```

Online display reader()

```
void reader()
{
    初期化
    while (true) {
        stdin から data を読む
            EOF で終了
        Data のデコード
        ヒストグラムの Fill
    }
}
```

Online display updater()

```
void updater()
{
    while (true) {
        TThread::Sleep(1, 0);
        canvas->cd(1)->Modified();
        canvas->cd(2)->Modified();
        canvas->Update();
        }
}
```

動かしてみよう

- COPPER で server を動かす。
- $ ./cli 192.168.11.xx 8888 | ./onlinedisplay
- 記録したデータを見る場合
- $ cat datafile | ./onlinedisplay

実習5 製作したものを統合する。

- 課題:
  - データを記録とオンラインディスプレイに振り分ける。
  - 今まで作ってきたものを統合して動かし
    - データの収集
    - データの記録
    - データの表示

    を行う。

実習5 製作したものを統合する。

- 標準入力のデータを標準出力とファイルに振り分けるもの → tee
  - とうぜん UNIX 系の OS では標準にあります。
- 1), 4), tee, 2) を使い DAQ software として動かす。

データを記録とオンラインディスプレイに振り分ける

```
int main(int argc, char* argv[])
{
        int dat;
        FILE *fd;
        if ((fd = fopen(argv[1], "w")) == NULL) {
                perror("file open err.");
                return 1;
        }
        while ((dat = getchar()) != EOF) {
                if (putchar(dat) == EOF) break;
                if (fputc(dat, fd) == EOF) break;
        }
        fclose(fd);
        return 0;
}
```

動かしてみよう

- COPPER で server を動かす。
- $ ./cli 192.168.11.xx 8888 | tee filename | ./onlinedisplay

- データの読み込み、記録、表示ができたでしょうか？
- 時間がある人は便利なスクリプトを書いてみよう。

まとめ

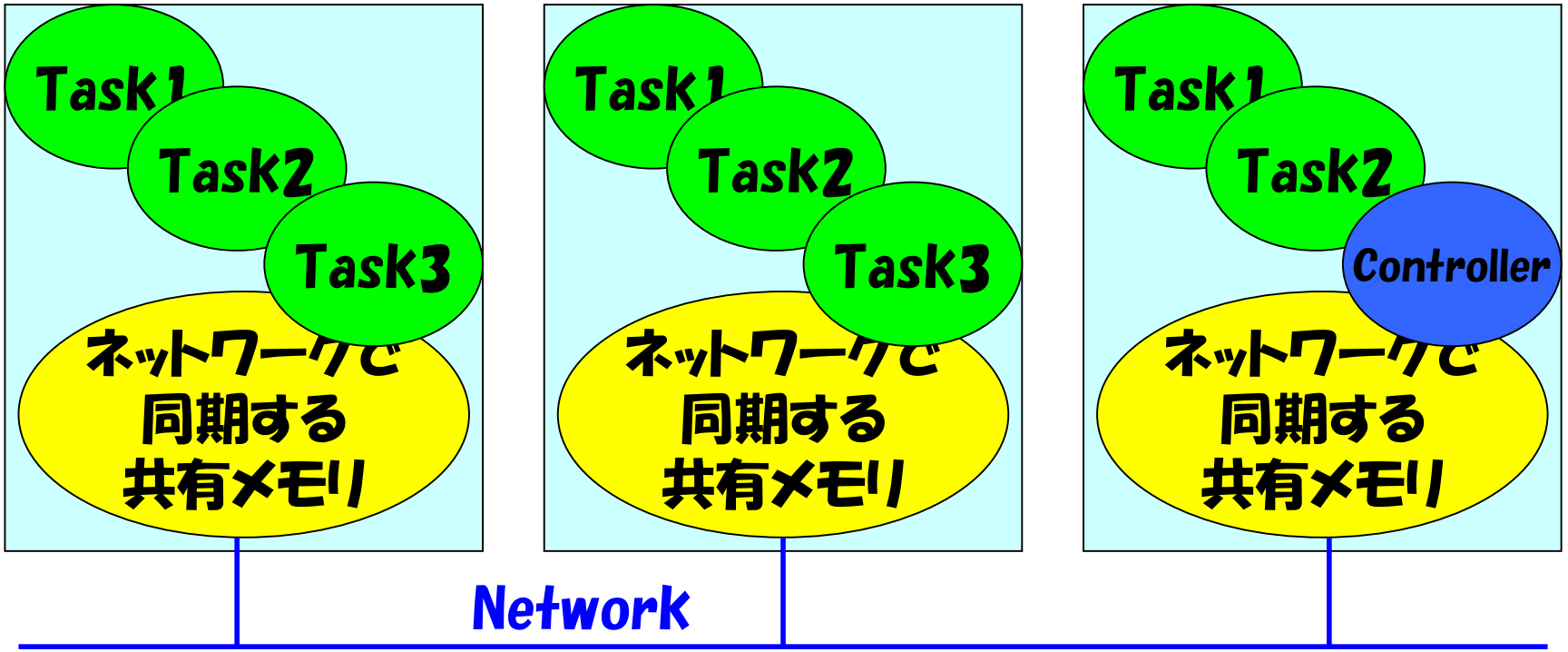
- OS の機能を利用して DAQ software の基本を体験
- データ収集モジュールが何であれの読み出しが出来ればテスト実験程度の DAQ は作れるはずです。

補講 制御について

- 実はたくさんのタスクを制御するのは難しい。
- コントローラーからネットワーク分散したタスクを制御する必要がある。
  - ネットワークで同期する共有メモリ
  - ネットワーク分散型データベース
  - ネットワークを利用したメッセージ
  - …

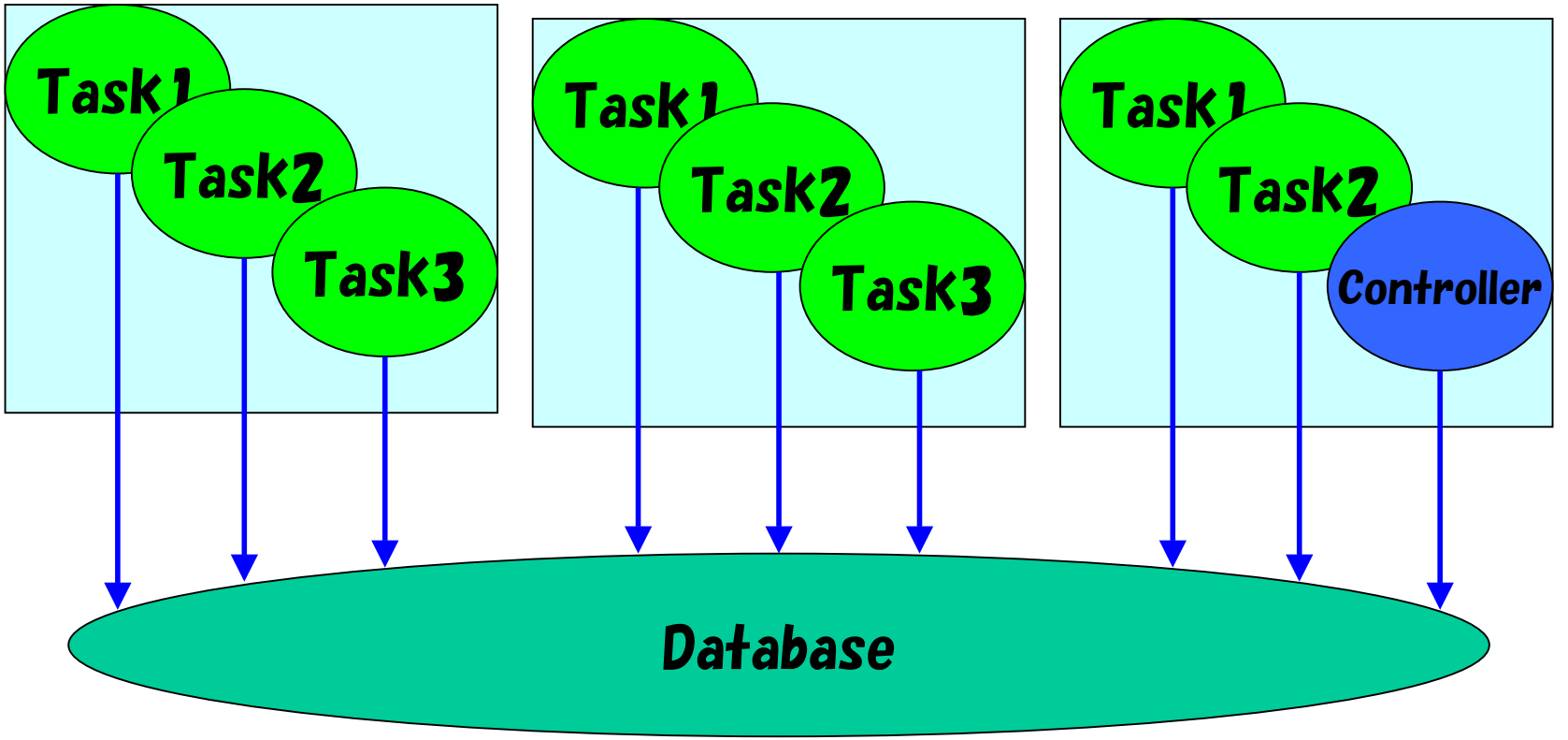
ネットワークで同期する共有メモリ

- リフレクティブメモリをネットワークで実現したもの



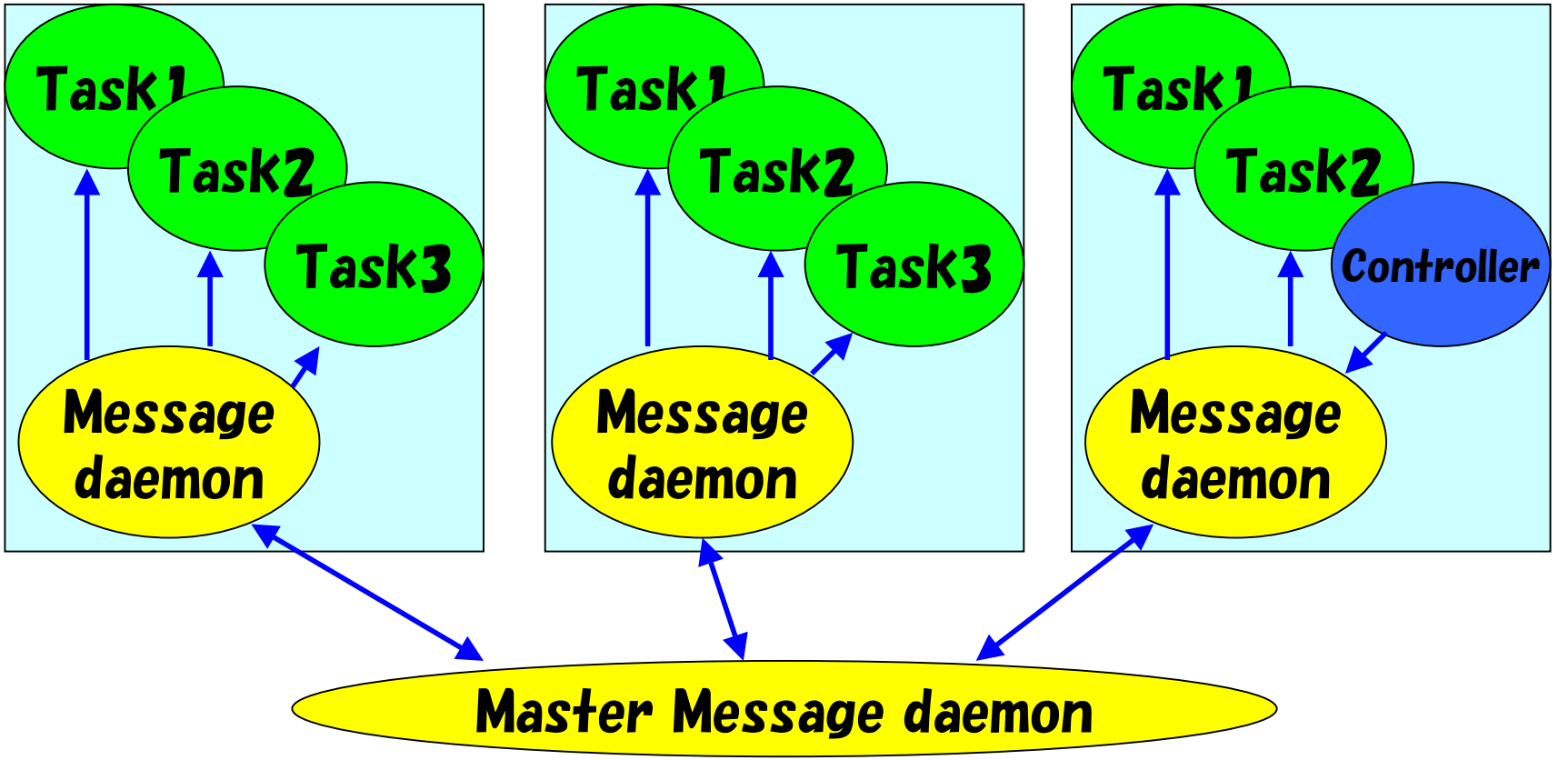
データベースを参照

- Task が共通のデータベースを参照しながら動く



Message を投げる

- Controller からの message が各タスクまで伝えられる。



Message を利用した制御の DAQ software の紹介

ネットワーク分散型データ収集ソフトウェアの開発

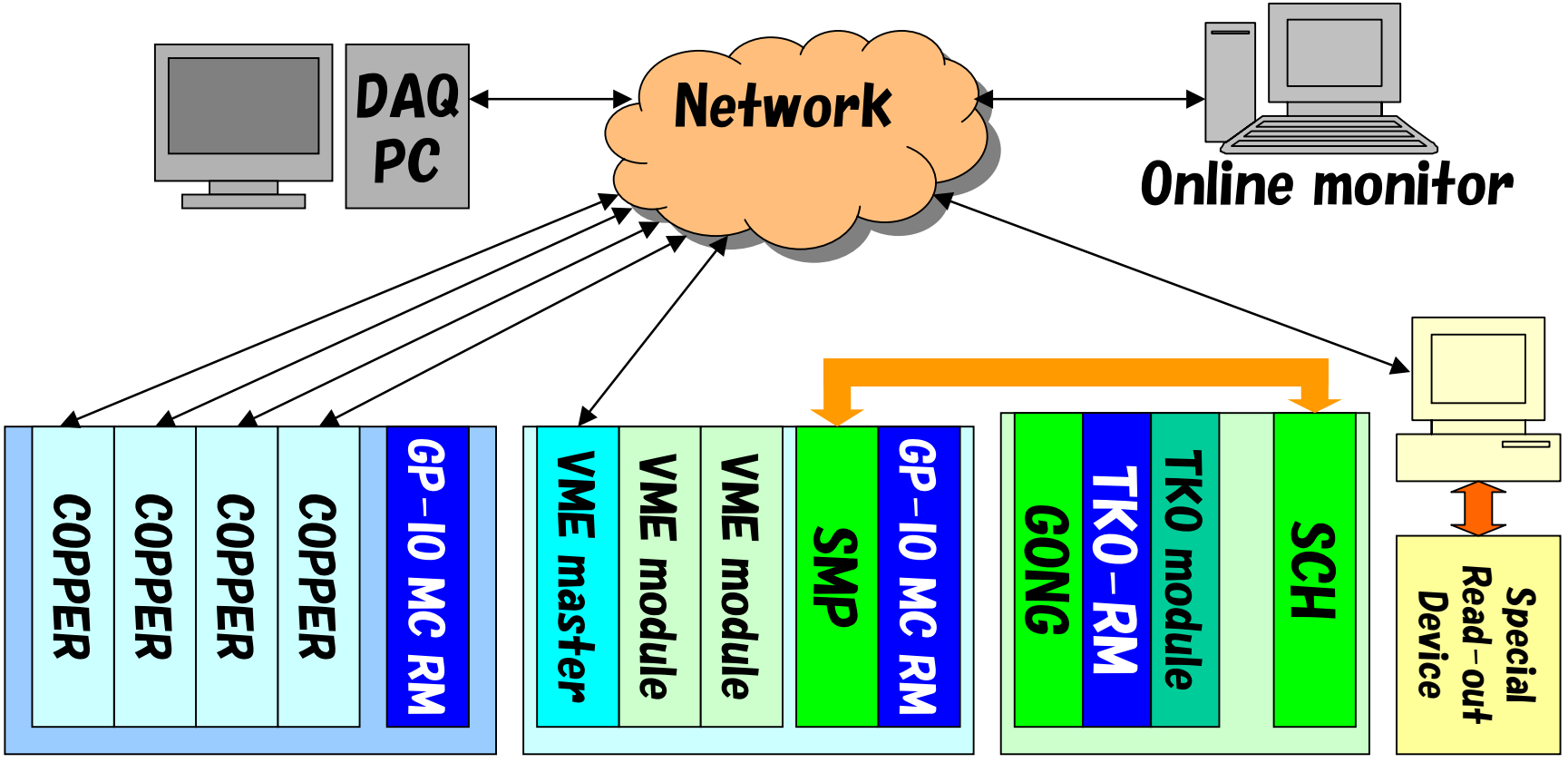
五十嵐 洋一[A]

藤井 啓文[A], 仲吉 一男[A], 長坂 康史[B], 細見 健二[C],
高橋 智則[D], KEK エレクトロシステムグループ

高エネ機構[A], 広工大[B], 東北大[C], 東大[D]

動機

- ネットワークにつながった多数の計算機に A/D のハードがつながっている環境でデータを収集したい。
- Trigger は共通にネットワークとは別に掛かるものとする。

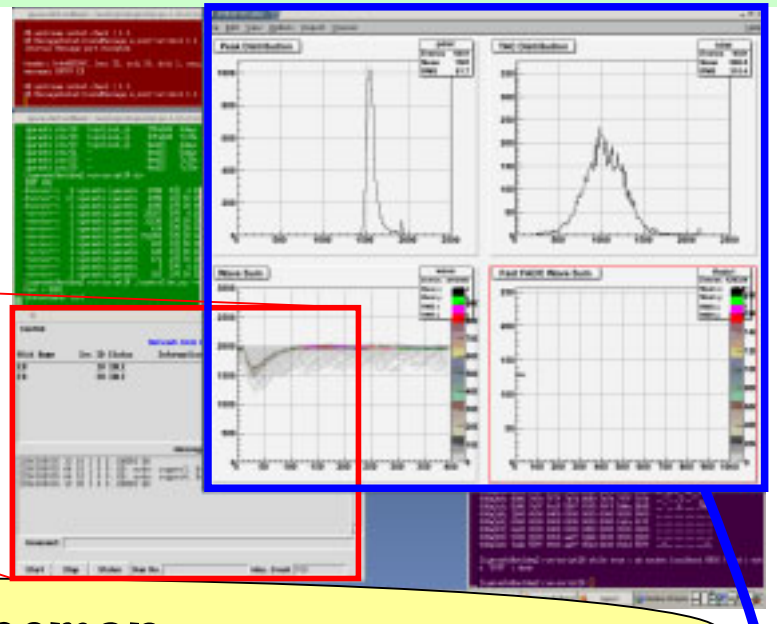


Network based DAQ Software (名前はまだ無い)

- たくさんの Front-end 計算機と Event builder 計算機、オンラインモニター用計算機がネットワーク上にあることを仮定。
- 小機能の複数のプロセスによる協調動作
  - Front-end node, Event Builder, Event distributor, Recorder, message daemon, controller, …
  - 機能の追加はデータフローにプロセスを挟むことで行える。
  - 各々の出力は同じ構造の data header をもつ。
- IPC は全て TCP/IP
- 自分で保守するために理解できる程度のコンパクトな実装
- 環境のバージョンアップに対応できるようにベース部分には ISO/POSIX, C, C++ のみ使用。
  - ただし UI, オンライン解析部は労力最小に
    - Controller は python, analyzer は ROOT が基本

Network based DAQ software processes

- Data が流れる data path
- 制御のための通信を行う message path



controller

Control /monitor

Message daemon

Front-end

Event Builder

Event Distributor

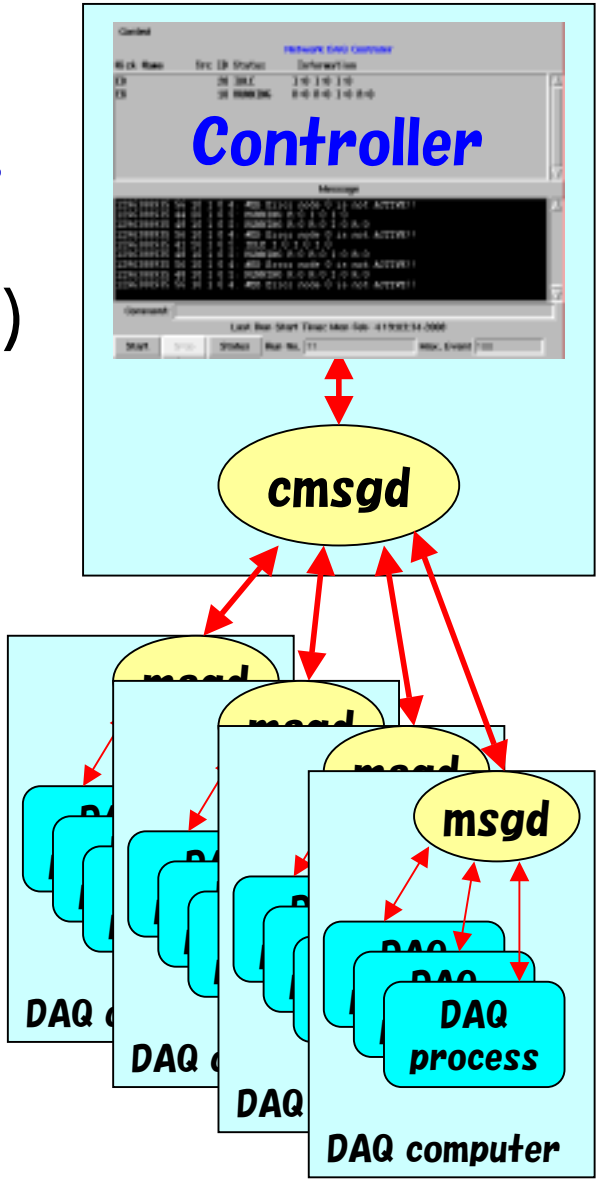
Recorder

Online analysis

Data flow

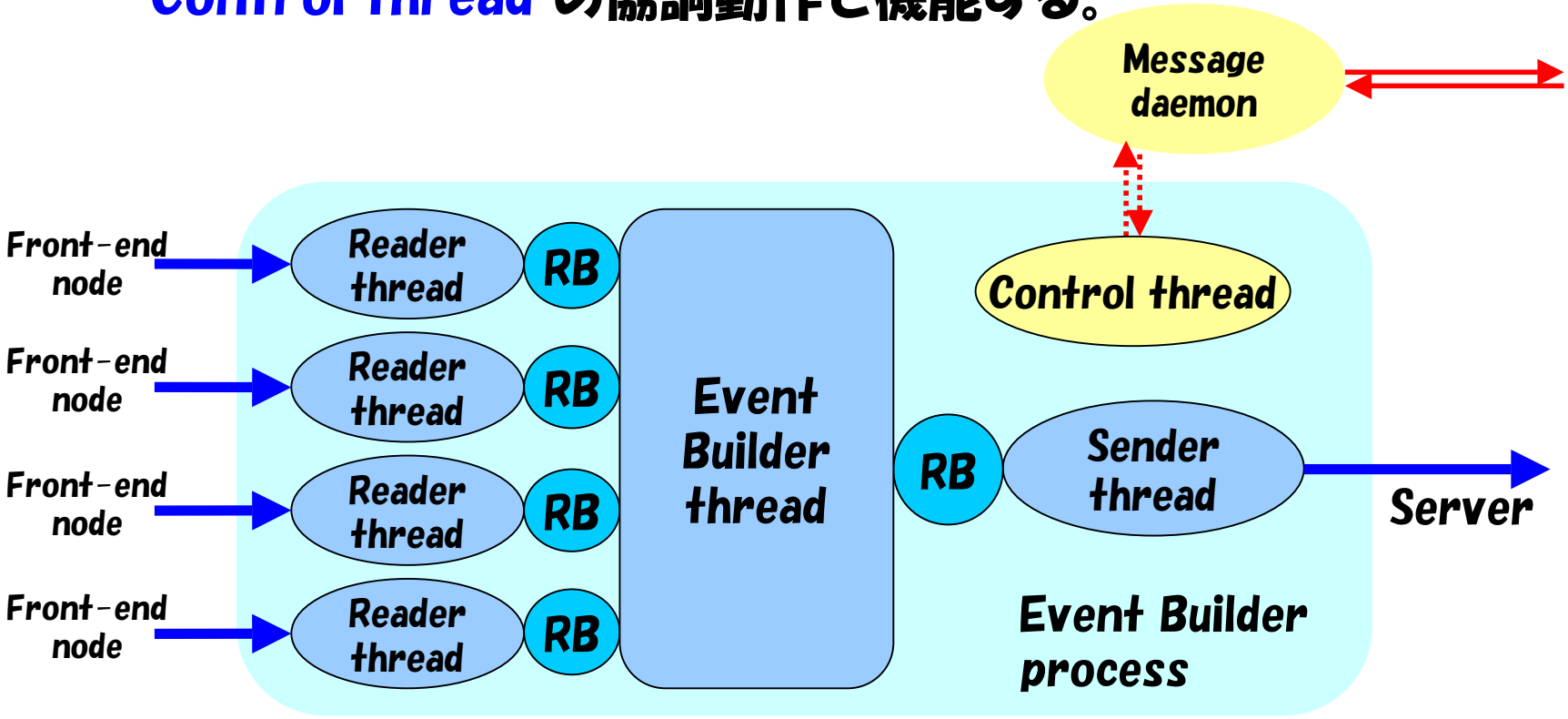
Message path

- Contoller
  - cmsgd(msgd) に command を伝える。
  - Status message をみて制御を行う。
- cmsgd (central message daemon)
  - システムにひとつだけ動いている。
    - おのおのの計算機で動いている msgd に command を配る。
    - msgd から status を受け取って controller に渡す。
- msgd (message daemon)
  - 各計算機の上で動いている
    - Command を cmsgd から受けておのおのの process に伝える。
    - Status を おのおののプロセスから受けて cmsgd に伝える。



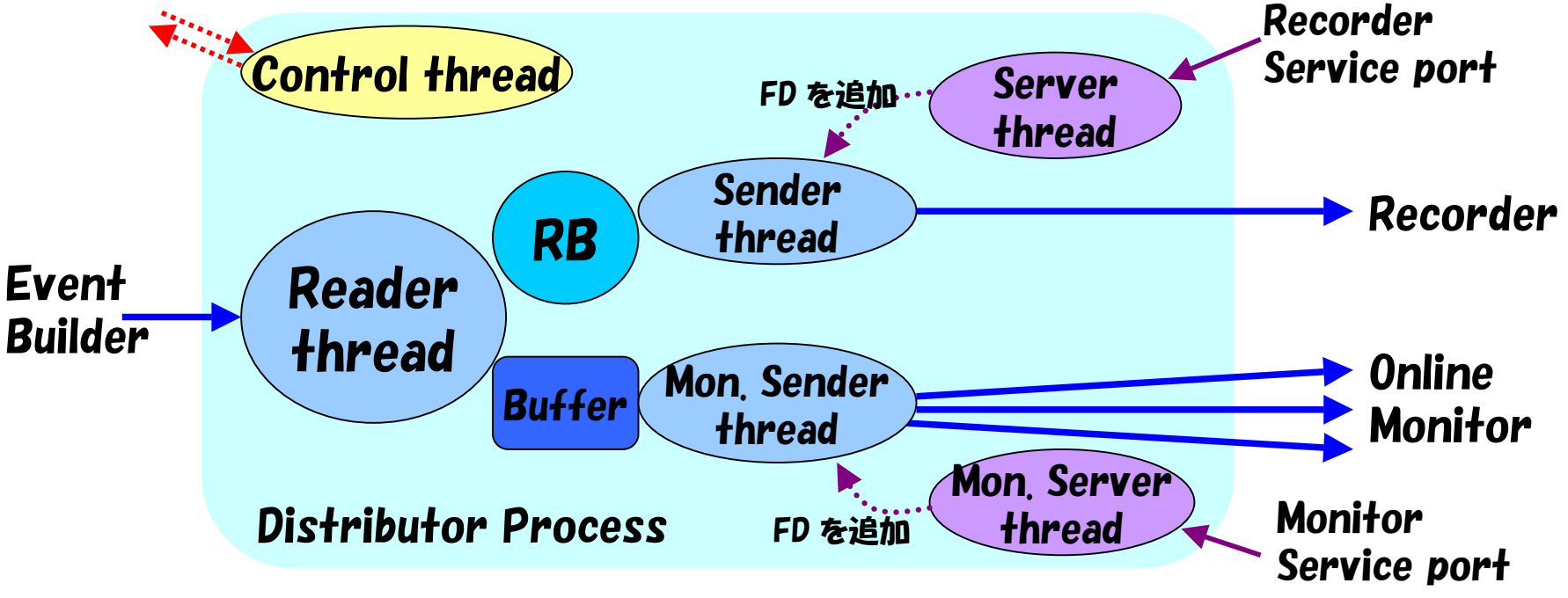
Event Builder

- Network 上の Front-end node から data を集めて Event Build を行う。
- Event Build の時点で Event Tag のチェック
- 複数の Reader thread, Builder thread, Sender thread, Control thread の協調動作で機能する。



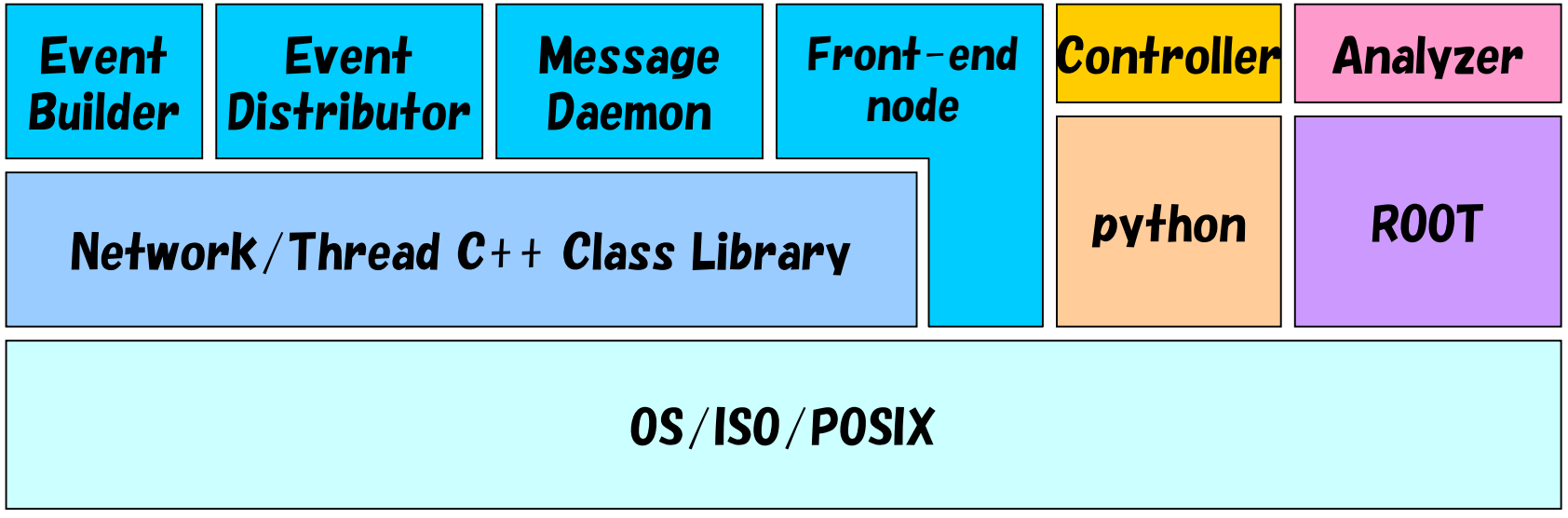
Event Distributor

- Event Builder から Event Data をもらい、Recorder processes, Monitor processes に Data を振り分ける。
- Reader thread, 2つの Server thread, 二種類の Sender thread (Recorder Sender/Monitor Sender), Control thread の協調動作で機能する。
- Recorder は全ての Event を読むため、止まるとシステムが停止するが、Monitor が止まっても Event がスキップされるだけ。



システム構成

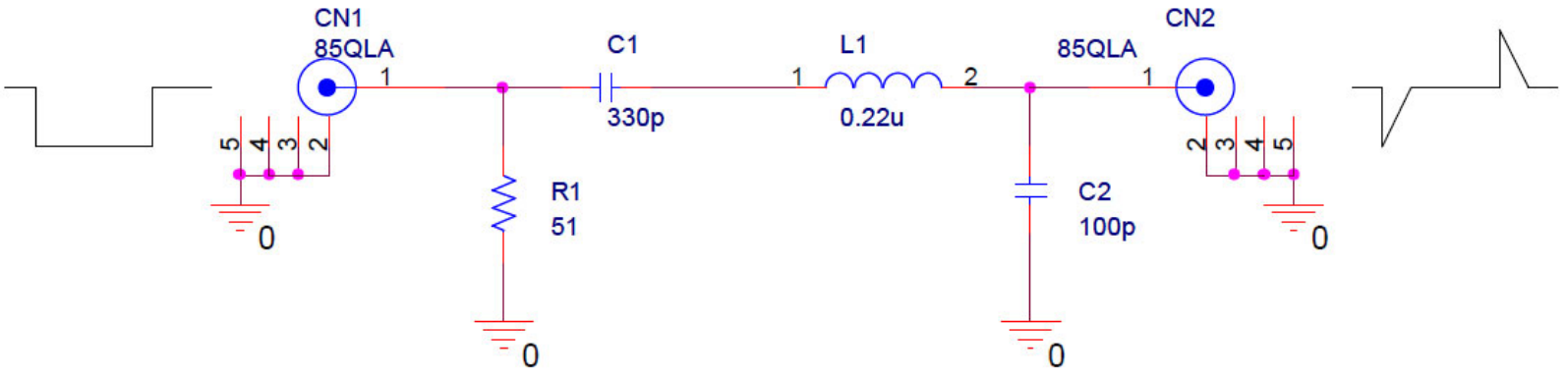
- 主要コンポーネントは DAQ 用 Network/Thread C++ Class Library の上に実装
- Controller は python で作成
- Analyzer は基本的には ROOT の上で開発



Supplement

今回の信号源

- NIM logic 信号をフィルタで整形
  - 微分+時定数をもって落ちいく回路。



Network Client 手順

- cli.c
  - Socket を作る。
  - gethostbyname でホスト名から ip アドレスを引いてくる。
  - sockaddr_in 構造体を埋める。
  - connect をかけ server と接続を行う。
  - エラーがでるまで socket から recv でデータを受け取りながら標準出力に受け取ったデータを書き込む。