
# Введение в ССД
# Программное обеспечение

О.Соловьянов (на основе материалов E. Pasqualucci INFN Roma / CERN)

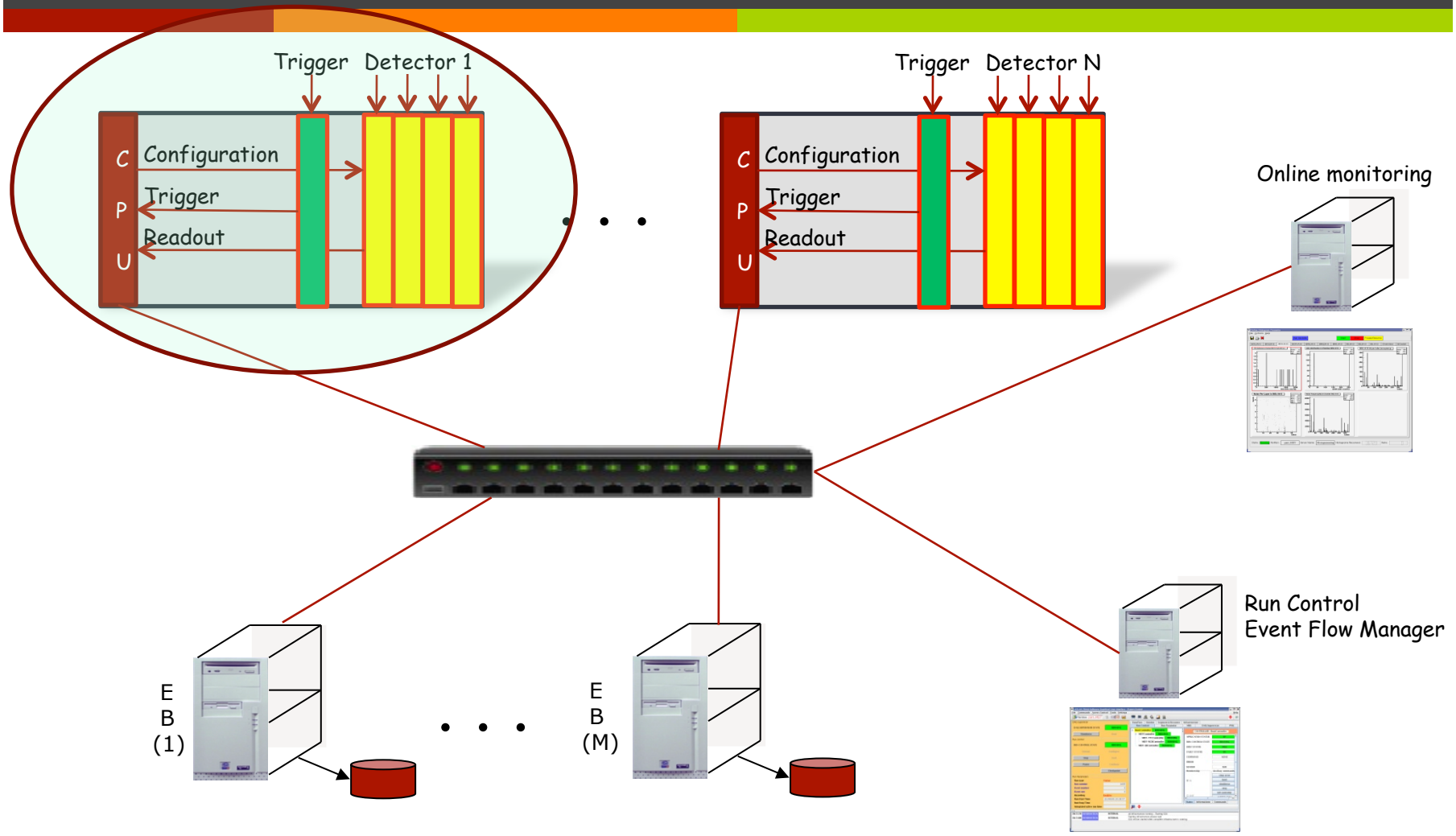
# Введение

- В этой лекции
  - обзор ССД среднего размера
  - анализ её компонентов
  - основные концепции программного обеспечения ССД
    - «Строительные материалы»
    - псевдо-код для объяснения

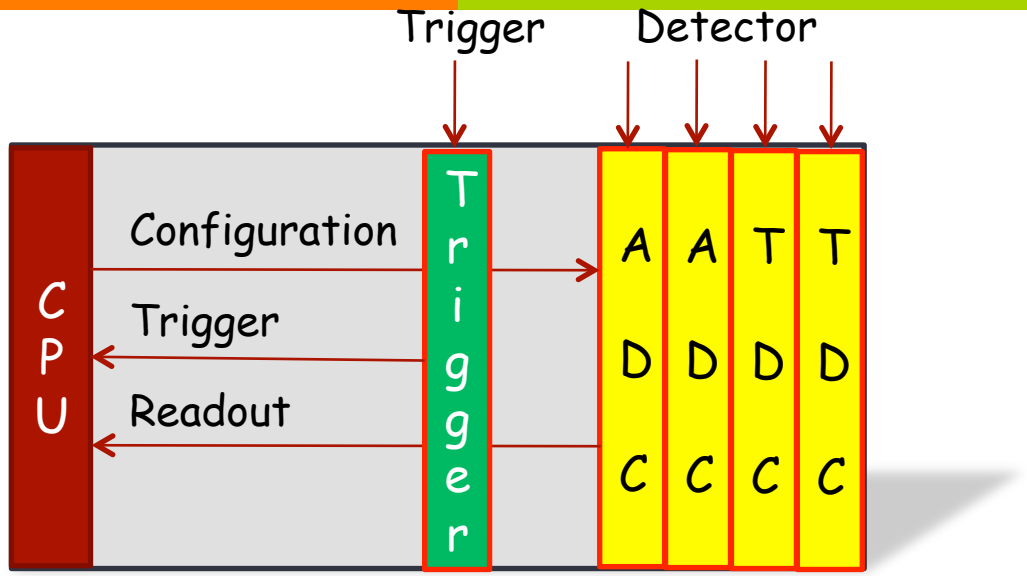
# Программные компоненты

- Управление триггером
- Чтение данных
- Форматирование и буферизация
- Передача данных
- Построение событий и запись
- Управление и наблюдение за системой
- Мониторинг и проверка качества данных

# Система из нескольких крейтов



# Чтение данных (пример)




- Данные оцифровываются модулями VME (ADC и TDC)
- Триггерный модуль получает триггерный сигнал
  - Регистр ввода/вывода или генератор прерываний
- Данные вычитываются одноплатным компьютером (SBC)

# Есть ли новое событие?

- Как узнать если новые данные (событие)?
  - Прерывание
    - Прерывание посылается модулем электроники
    - Далее прерывание
      - Преобразуется в программный сигнал
      - Детектируется программой сбора данных
        - Начинается чтение данных
  - Опрос
    - Постоянно читается некий регистр в электронном модуле
    - Чтение данных начинается когда в нем есть нужный признак
- Синхронная система
  - Триггер должен выставить сигнал «занят»
  - Процесс чтения должен снять этот сигнал по окончании чтения события

# Программирование в «реальном масштабе времени»

- Должно строго удовлетворять временным требованиям оборудования
  - Подразумевает взятие на себя типичных обязанностей операционной системы
    - Например, управление процессами
  - ОС реального времени предоставляет эти возможности
- Наиболее важное свойство – предсказуемость
  - Производительность менее важна нежели предсказуемость
- Обычно применяется при следующих требованиях
  - Время реакции на прерывание должно лежать в определенном интервале
  - Полный контроль за распределением времени на процессы

# Так ли нужна система «реального времени»?

- В некоторых случаях абсолютно необходима
  - Важна для управления ускорителем
    - Когда время реакции является существенным
    - Когда необходимы сложные вычисления
- В настоящее время мало применяется в ССД
  - Большие системы обычно асинхронны
    - События буферизуются в электронике или в памяти программ
      - Производительность улучшается при использовании ПДП(DMA)
    - Основной поток данных идет не по шинам
  - В маленьких системах мёртвое время тоже мало
- Недостатки отказа от систем реального времени
  - Теряется полный контроль за мёртвым временем
    - Скорость реакции на событие и передачу данных определяется операционной системой
  - Увеличение задержки из-за буферизации
    - Увеличение размера буфера
  - Тем не менее, всё это не является проблемой для ССД

# Опрос

- Чтение в цикле регистра, содержащего признак триггера

```
while (end_loop == 0)
{
  uint16_t *pointer;
  volatile uint16_t trigger;

  pointer = (uint16_t *) (base + 0x80);
  trigger = *pointer;

  if (trigger & 0x200) // look for a bit in the trigger mask
  {
    ... Read event ...
    ... Remove busy ...
  }
  else
    sched_yield (); // if in a multi-process/thread environment
}
```

# Опрос или прерывание?

- Какой метод предпочтительнее?
- Зависит от частоты событий
  - Прерывание
    - Длительное время отклика
      - Обычно (O (1 μs))
    - Подходит для редких событий
      - Нет необходимости в постоянной проверке
      - Плата может также использовать прерывания для ошибок
  - Опрос
    - Подходит для событий с высокой частотой
      - Когда вероятность события велика
    - Не влияет на другие процессы при хорошем планировщике

# Простейшая ССД

- Синхронное чтение:
  - Триггер
    - Авто-блокируемый («занят» выставляет сам триггер)
    - Явно разрешается по завершении чтения события
- Дополнительное мёртвое время из-за передачи данных

```
// VME interrupt is mapped to SYSUSR1
//
static int event = FALSE;
const int event_available = SIGUSR1;

// Signal Handler

void sig_handler (int s)
{
  if (s == event_available)
    event = TRUE;
}
```

```
event_loop ()
{
  while (end_loop == 0) {
    if (event) {
      size += read_data (*p);
      write (fd, ptr, size);
      busy_reset ();
      event = FALSE;
    }
  }
}
```

# Буферизация данных

- Почему буферизация?
  - Создание «внутренних» разравнивателей
    - Оптимизация использования внешних каналов
      - Диск
      - Сеть
    - Позволяет избегать заторов при неравномерных интервалах между событиями
  - Внимание!
    - Следует избегать множественного копирования
      - Копирования памяти это длительная операция
      - Надо передавать указатели

# Простой пример…

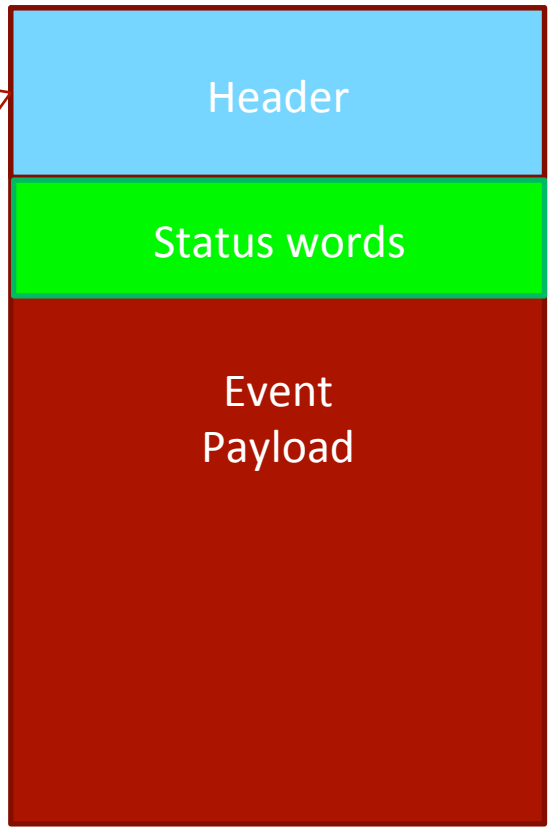
- Кольцевой буфер (имитирует FIFO)
  - Буфер создается в памяти
    - Можно использовать «разделяемую память» (shared memory)
    - «Ведущий» создает/уничтожает память и семафор
    - «Ведомы» присоединяется/отсоединяется к памяти
  - События («пакеты»)
    - Пишется в буфер процессом записи
    - Читаются процессом чтения
  - Работает в много-процессной среде
  - Важно
    - Избегать излишнего копирования
    - Если возможно, форматировать события непосредственно в памяти

# «Обрамление» данных (framing)

- Заголовок/концевик фрагмента события
- Определяет фрагменты события и их характеристики
  - Полезно для цепочки последовательных процессов ССД
    - Построитель событий, мониторирование
  - Легко определяется происхождение фрагмента
    - Помогает определить источник проблем
  - Может (должен) содержать номер триггера (для построителя событий)
  - Может (должен) содержать статусное слово
- Общее оформление фрагмента
  - Даёт общую информацию о событии
- Важно с сетевых системах

# Пример обрамления

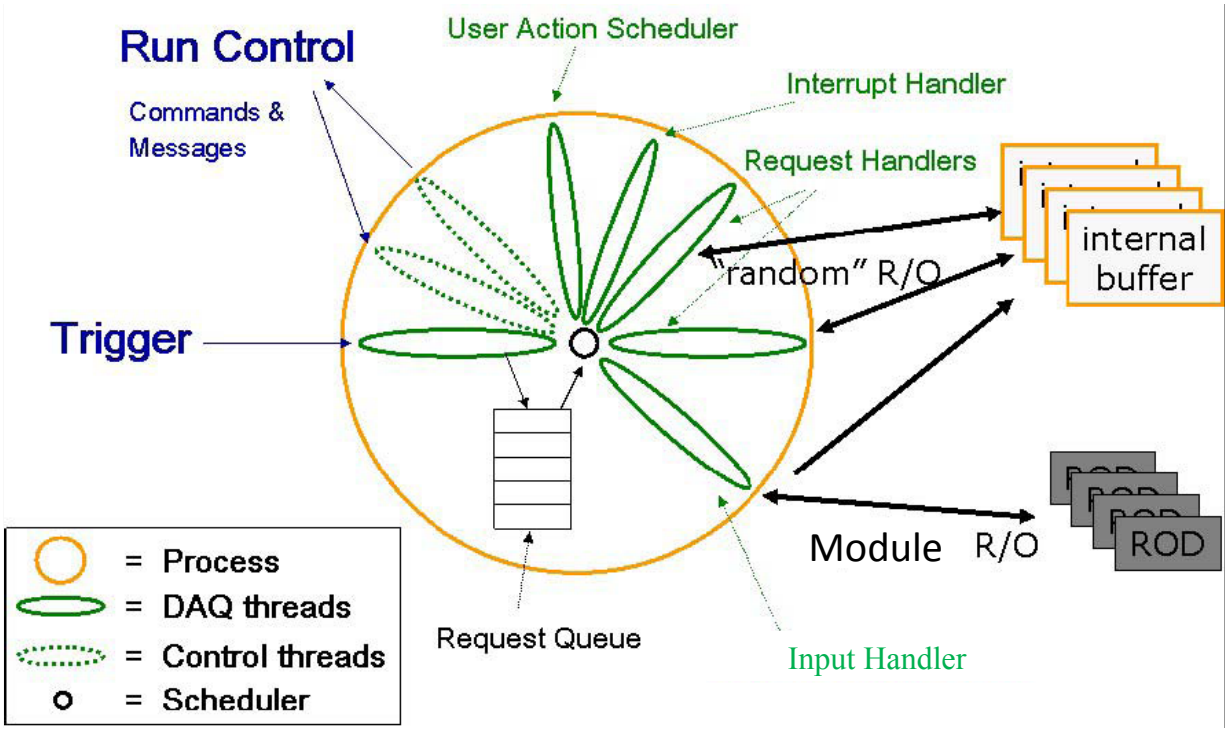
```
typedef struct
    {
      u_int startOfHeaderMarker;
      u_int totalFragmentsize;
      u_int headerSize;
      u_int formatVersionNumber;
      u_int sourceIdentifier;
      u_int numberOfStatusElements;
    } GenericHeader;
```



# Более общий буферный менеджер

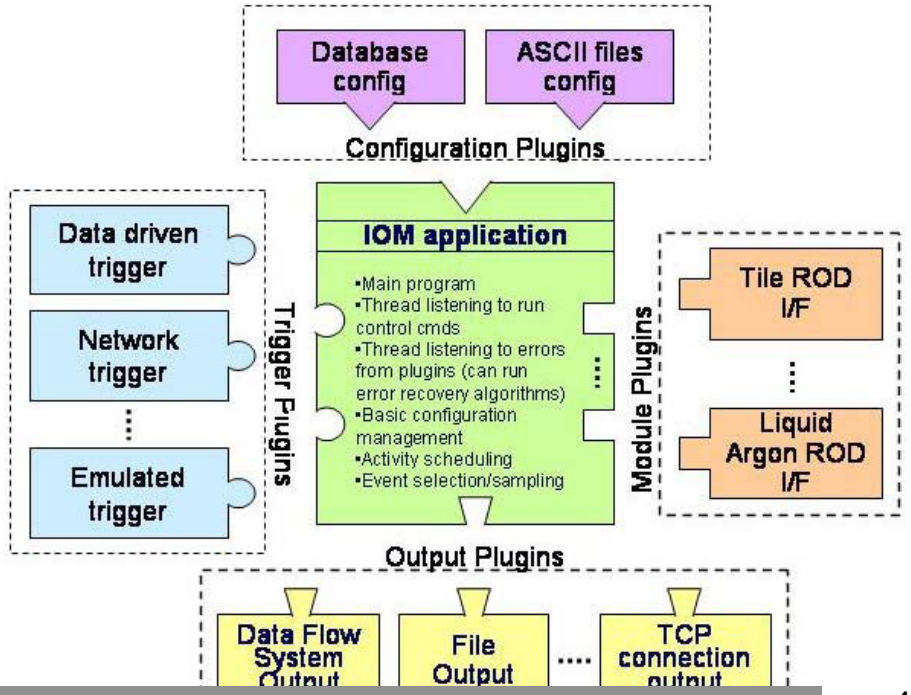
- Простая идея
  - Использование выделенного «пула» памяти для передачи данных
- Страничная (paged) память
  - Для минимизации вычислений указателей
  - Удобно в случае событий примерно одинакового размера
    - За счёт некоторого объема памяти
- Буферные описатели
  - Хранятся в специально отведённой памяти
  - Указатели на описатели хранятся в очереди
- Позволяет иметь любое число входных и выходных потоков

# Обобщённая программа сбора данных



# Настраиваемые приложения

- Амбициозная идея
  - Поддерживать все системы (детекторы) с помощью одного единственного приложения
    - С помощью механизма динамически загружаемых библиотек
    - Требует механизма конфигурации

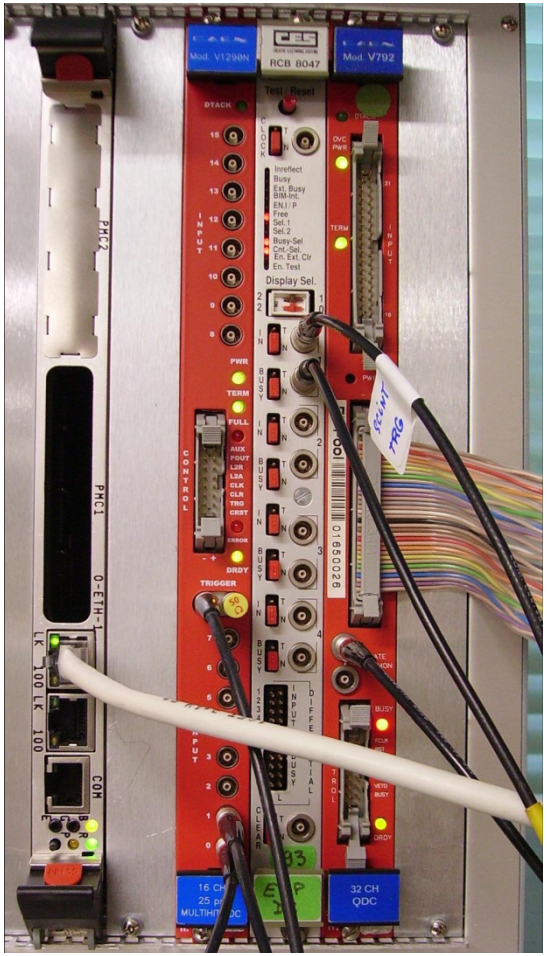


# Основные компоненты

- Взаимодействие между процессами (IPC)
  - Сигналы
  - Общая (разделяемая) память
  - Семафоры
  - Очереди сообщений
- Стандартные концепции ССД
  - Триггер, «занято» и «обратное давление»
  - Синхронные и асинхронные системы
  - Опрос и прерывания
  - Системы реального времени
  - Обрамление данных
  - Управление памятью

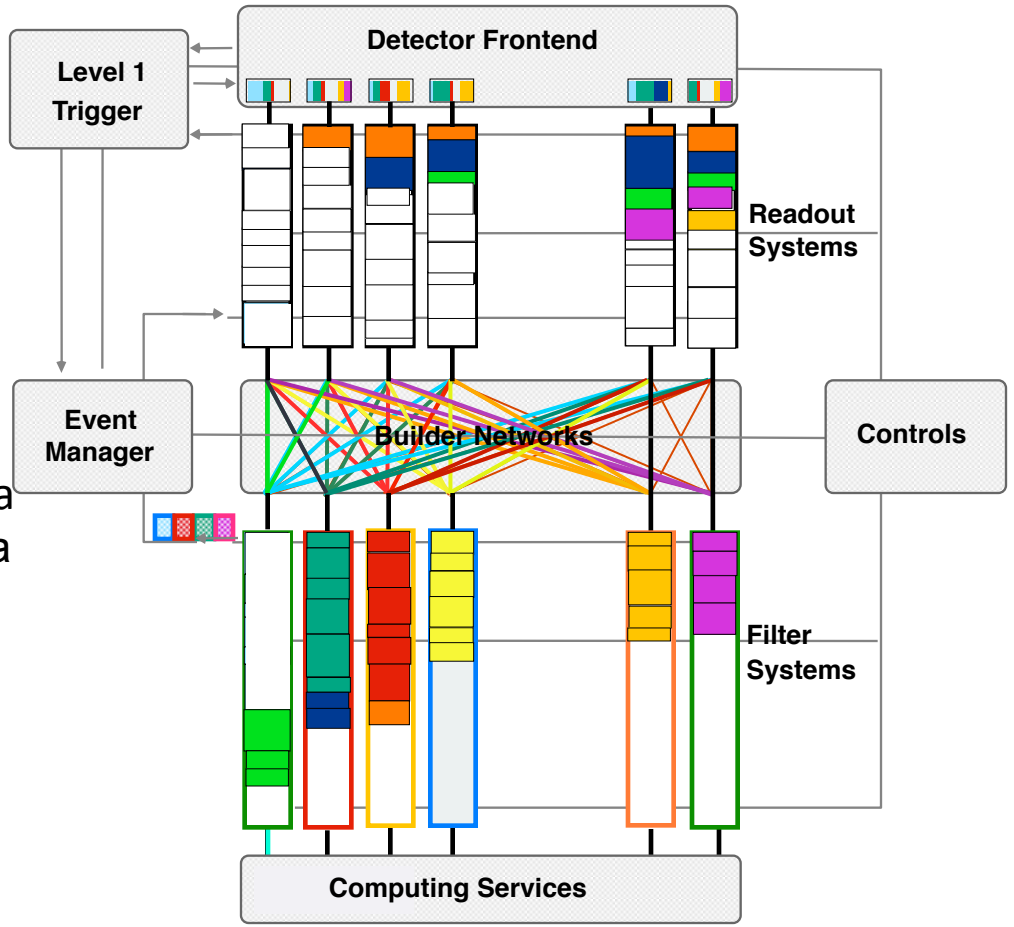
# Пример системы сбора данных

- Простая ССД
  - Контроллер крейта VME
  - Триггерный модуль
    - По приходу сигнала триггера
      - Ставит сигнал «занят»
      - Посылает прерывание VME
      - Запоминает триггер в регистре
  - QDC
  - TDC



# Построение событий

- Большие детекторы
  - Данные в под-детекторах собираются параллельно
    - Сети для чтения
    - Быстрые линии связи
  - События собираются компоновщиками событий
    - Из соответствующих фрагментов данных
  - Специализированные устройства
    - Регистрирующая электроника
    - Триггер нижнего уровня
  - Устройства общего назначения
    - Триггер высокого уровня
    - Сеть компоновщика событий
- Система сбора данных
  - Управление потоками данных
  - Распределённая и асинхронная



# Сети передачи данных и протоколы

- Передача данных
  - Фрагменты должны быть посланы компоновщику (построителю)
    - Одному или нескольким
  - Обычно делается через коммутируемые сети
- Протоколы уровня пользователя
  - Предоставляют необходимый уровень абстракции
    - Можно игнорировать детали используемого «железа»
    - И оптимизации предоставляемые операционной системой (не всегда)
- Наиболее часто используемые
  - Набор протоколов TCP/IP
    - UDP (User Datagram Protocol)
      - Без установления соединения
    - TCP (Transmission Control Protocol)
      - С установлением соединения
      - Включает в себя подтверждение приёма и повторную передачу

# Пример клиент/сервер TCP

```
struct sockaddr_in sinhim;
sinhim.sin_family      = AF_INET;
sinhim.sin_addr.s_addr = inet_addr (this_host);
sinhim.sin_port = htons (port);

if (fd = socket (AF_INET, SOCK_STREAM, 0) < 0)
{ ; // Error ! }
if (connect (fd, (struct sockaddr *)&sinhim,
             sizeof (sinhim)) < 0)
{ ; // Error ! }
```

```
while (running) {
  memcpy ((char *) &wait, (char *) &timeout,
          sizeof (struct timeval));
  if ((nsel = select (nfds, 0, &wfds,
                      0, &wait)) < 0)
  { ; // Error ! }
  else if (nsel) {
    if ((BIT_ISSET (destination, wfds))) {
      count = write (destination, buf, buflen);
      // test count…
      // > 0 (has everything been sent ?)
      // == 0 (error)
      // < 0 we had an interrupt or
      // peer closed connection
    }
  }
}
```

```
struct sockaddr_in sinme;
sinme.sin_family      = AF_INET;
sinme.sin_addr.s_addr = INADDR_ANY;
sinme.sin_port        = htons(ask_var->port);

fd = socket (AF_INET, SOCK_STREAM, 0);
bind (fd0, (struct sockaddr *) &sinme,
      sizeof(sinme));
listen (fd0, 5);

while (n < ns) { // we expect ns connections
  int val = sizeof(this->sinhim);
  if ((fd = accept (fd0,
      (struct sockaddr *) &sinhim, &val)) >0) {
    FD_SET (fd, &fds);
    ++ns;
  }
}
```

```
while (running) {
  if ((nsel = select( nfds, (fd_set *) &fds,
       0, 0, &wait)) [
    count = read (fd, buf_ptr, buflen);
    if (count == 0) {
      close (fd);
      // set FD bit to 0
    }
  }
```

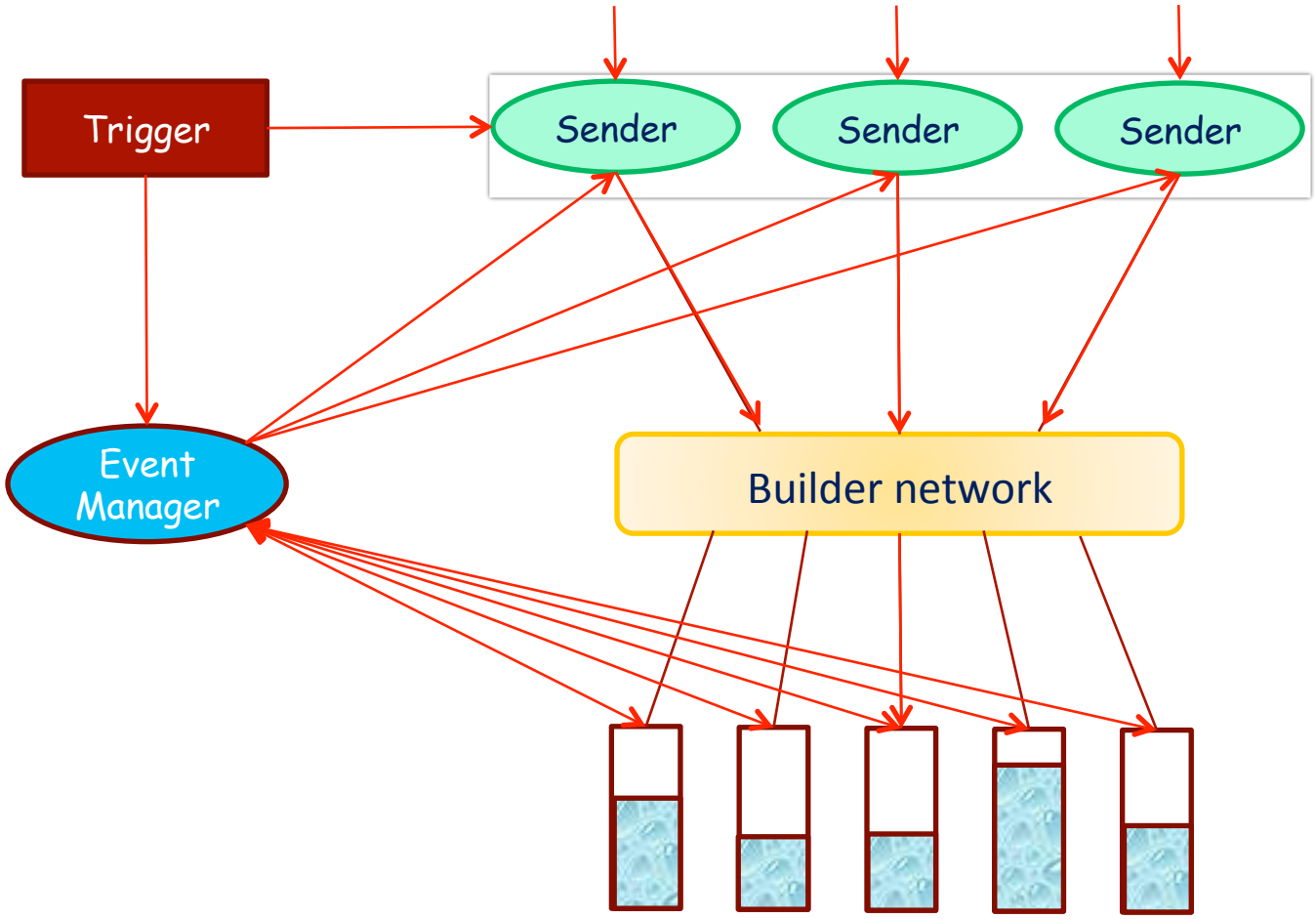
# Оптимизация передачи данных

- При отправке данные копируются в системный буфер
- На системном уровне
  - Каждый узел приёма-передачи данных использует буфер
  - TCP перестаёт посылать данные когда нет свободного места
    - Возникает «обратное давление» (back-pressure)
  - При использовании UDP данные теряются при отсутствии места
  - Если не хватает места в буфере:
    - Увеличить размер сетевого буфера (обычно доступно до 8 MB)
  - Слишком большие буфера могут привести к проблемам с производительностью (поиск, очистка)

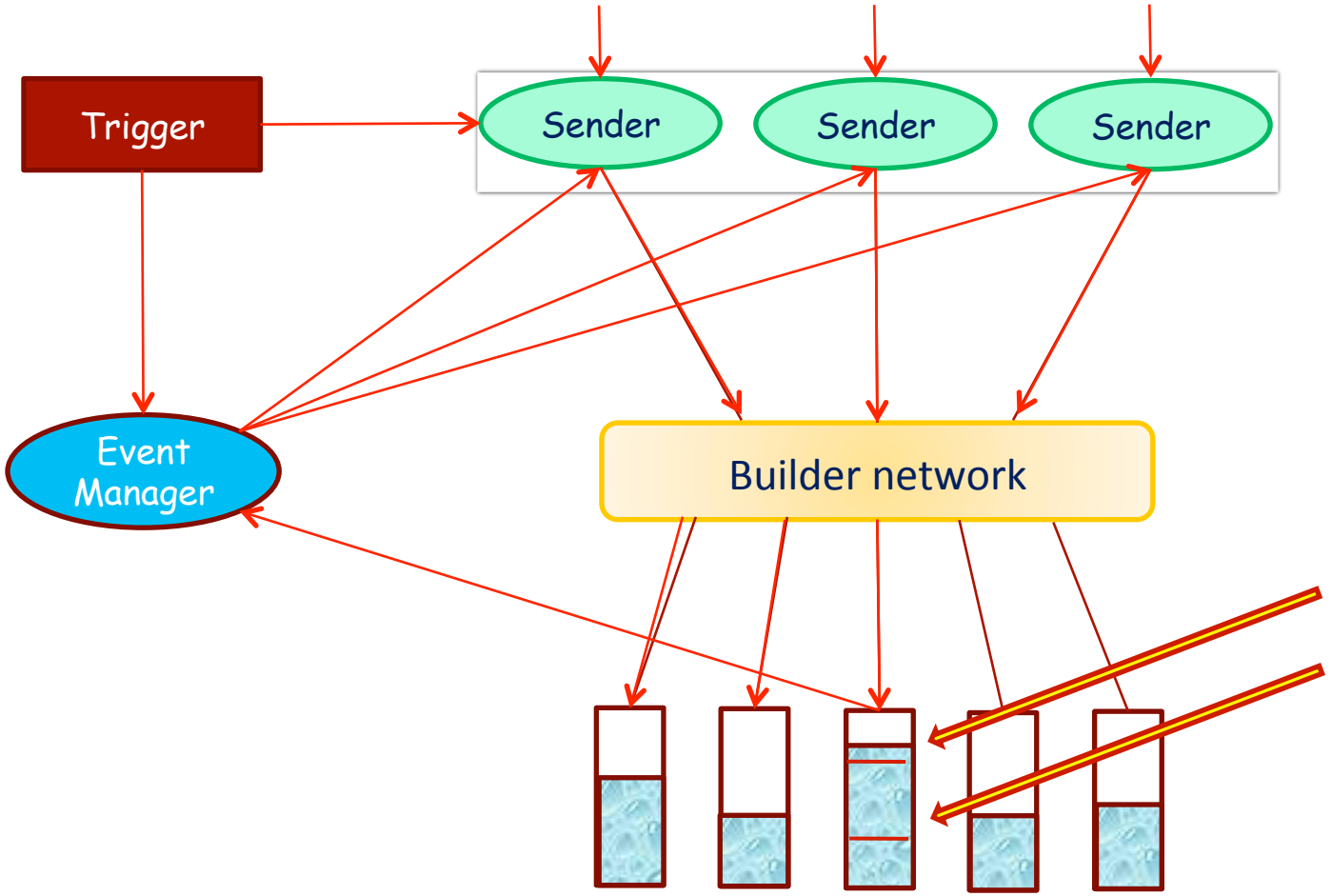
# Управление потоками данных

- Оптимизация производительности
- Следует избегать мёртвого времени из за back-pressure
  - Увеличивая число точек приёма
  - Требует знания их состояний
- Архитектуры компоновщика (построителя) событий
  - Толкай
    - События посылаются отправителем сразу после получения
      - Отправитель знает куда посылать данные
      - Самый простой алгоритм распределения – по кругу
  - Тяни
    - События запрашиваются получателем
      - Требует менеджера событий
        - Хотя можно обойтись и без него

# Пример «тяни» (pull)



# Пример «толкай» (push)

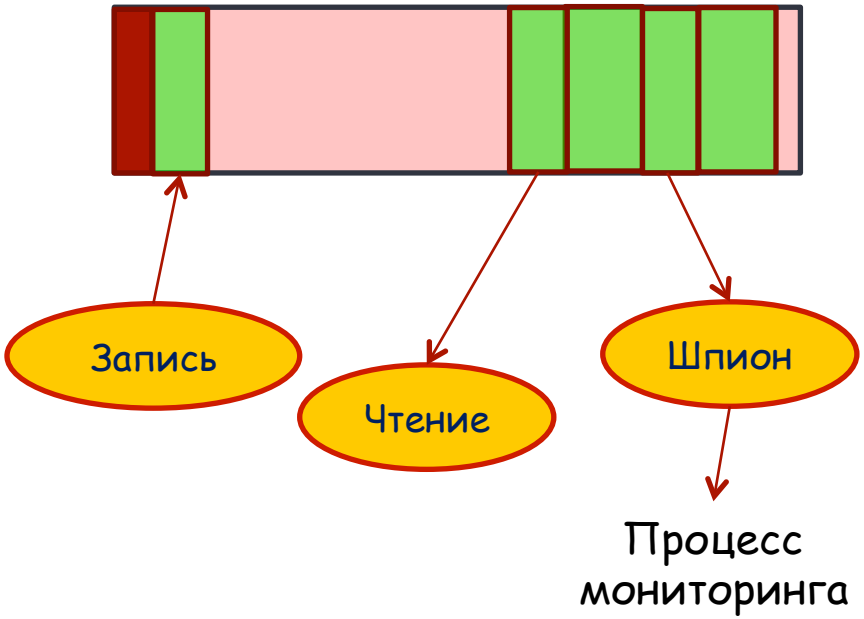


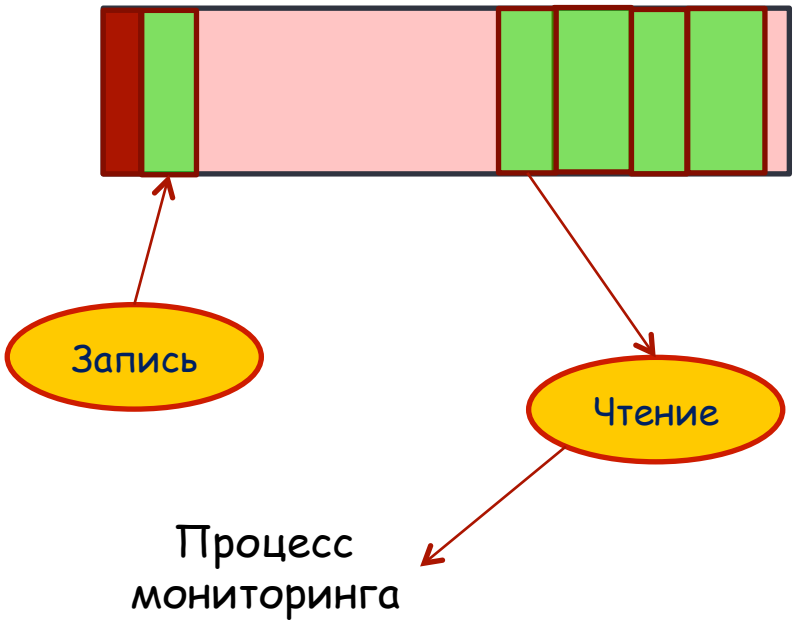
# Мониторинг системы

- Два основных аспекта
  - Мониторинг работы системы
    - Обмен переменными, отражающими производительность системы
  - Мониторинг данных
    - Выборка данных для мониторинга
    - Обмен графиками и гистограммами
    - Просмотр графиков и гистограмм

# Пример выборки данных

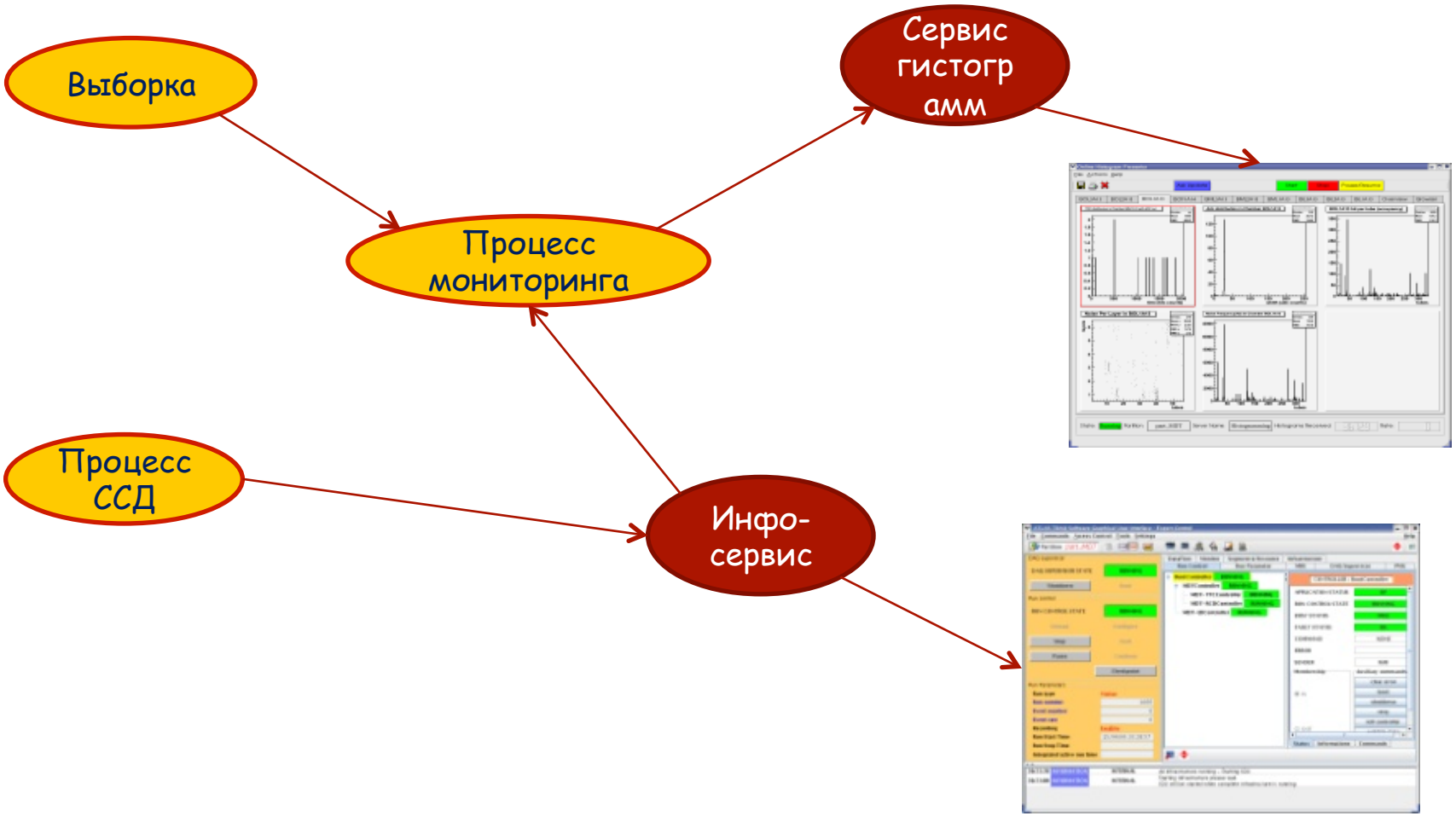
- Выборка из буфера
- На входе и выходе



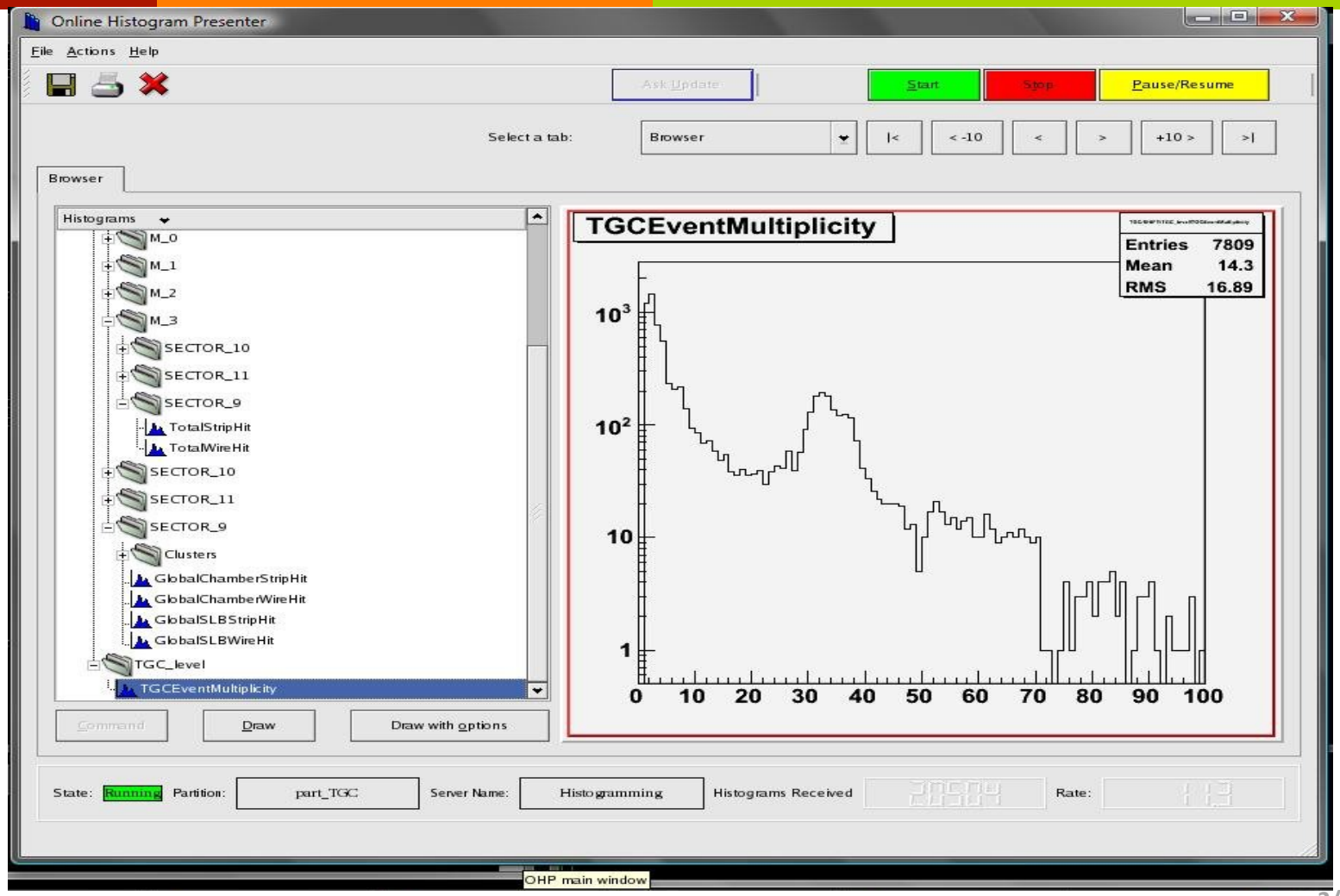




**Выборка не должна тормозить передачу данных**

# Обмен переменными и гистограммами



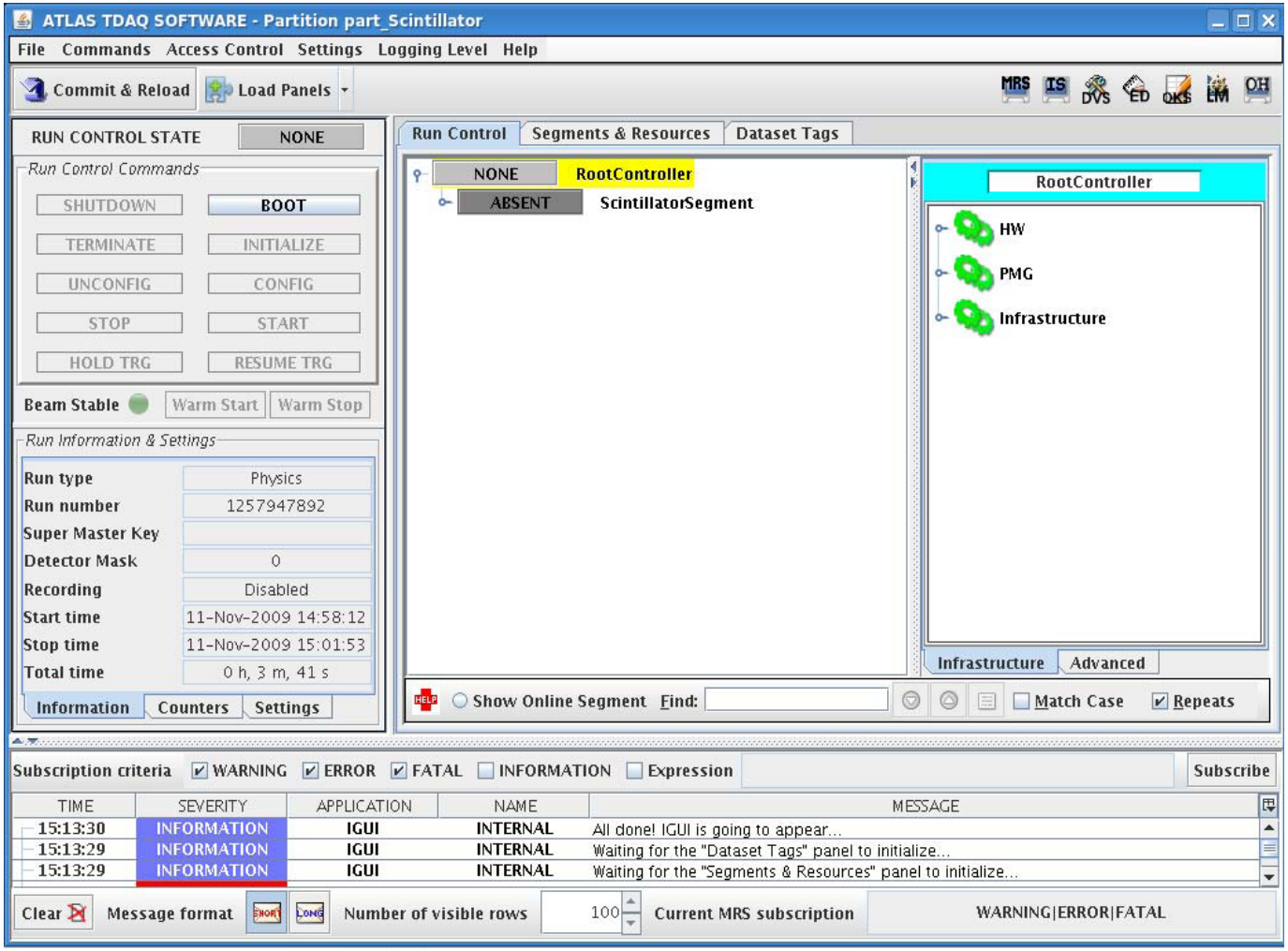
# Просмотр гистограмм



# Управление системой

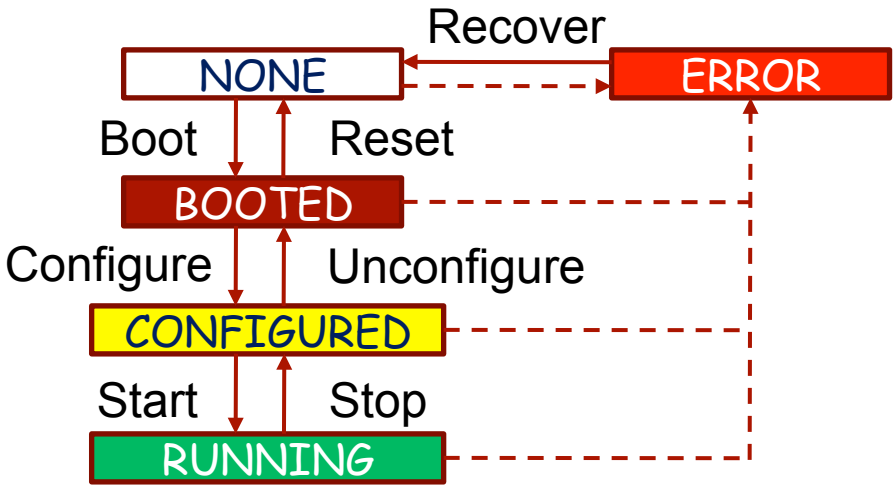
- Каждый компонент ССД должен иметь
  - Набор строго определенных состояний
  - Набор правил перехода из одного состояния в другое
  - ⇒ Конечный автомат (finite state machine = FSM)
- Системой управляет центральный процесс
  - Управление работой (run control)
    - Осуществляет конечный автомат
    - Запускает смену состояния и отслеживает состояние компонентов
      - Для масштабирования используются древесные структуры процессов (иерархическая группировка)
- Графический интерфейс пользователя
  - Всякие системные средства и утилиты

# Пример графического интерфейса пользователя (GUI)



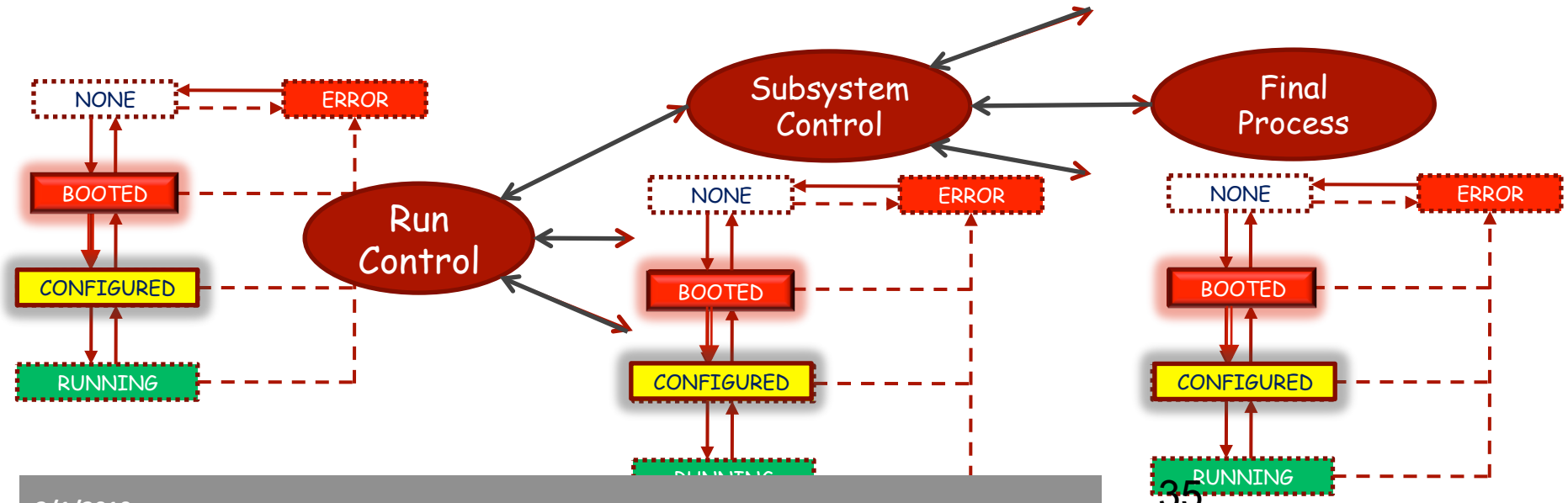
# Конечный автомат (FSM)

- Моделирует поведение системы с помощью конечного набора состояний системы и компонентов
- Конечный автомат состоит из 4-х основных элементов
  - Состояния, которые задают поведение и порождают действия
  - Переходы между состояниями
  - Правила и условия разрешающие определённые переходы
  - Входные события, внешние и внутренние, которые могут активизировать правила и привести к смене состояния



# Распространение состояний

- Каждый компонент или подсистема выполнена как конечный автомат
  - Смена состояния компонента завершена только в случае перехода всех под-компонентов в нужное состояние
  - Переходы между состояниями запускаются командами, передаваемыми системой сообщений



# Реализация конечного автомата

- Концепция состояния хорошо ложится на концепцию состояния объекта
  - Объектно-ориентированное программирование хорошо подходит для реализации конечных автоматов
- Смена состояний
  - Обычно реализуется в виде обратных вызовов (callbacks)
    - В ответ на сообщения
- Важно:
  - Каждое состояние должно быть строго определено
  - Переменные, определяющие состояние, должны иметь те же самые значения
    - Независимо от направления перехода

# Система передачи сообщений

- Обычно сетевая
- Множество реализаций
  - От обычных TCP пакетов…
  - … включая (достаточно экзотическую) SNMP …
  - … до объектно-ориентированных Object Request Broker (ORB)

# В заключение

- Нет единственно правильного способа
  - Разные системы требуют разных подходов и реализации
- Параметры системы должны задавать архитектуру программного обеспечения
  - Примеры:
    - Компоновщик событий может использовать динамическое выделение памяти операционной системой
      - Это затратно
      - Годится в случае ограничения пропускной способности скоростью передачи данных по сети
    - Реагировать на прерывания или непрерывно опрашивать
      - Зависит от ожидаемой частоты событий
    - Обрамление событий важно
      - Главное не переборщить