

The CBM FLES Input Interface

Dirk Hutter

hutter@compeng.uni-frankfurt.de

Prof. Dr. Volker Lindenstruth

FIAS Frankfurt Institute for Advanced Studies

Goethe-Universität Frankfurt am Main,
Germany

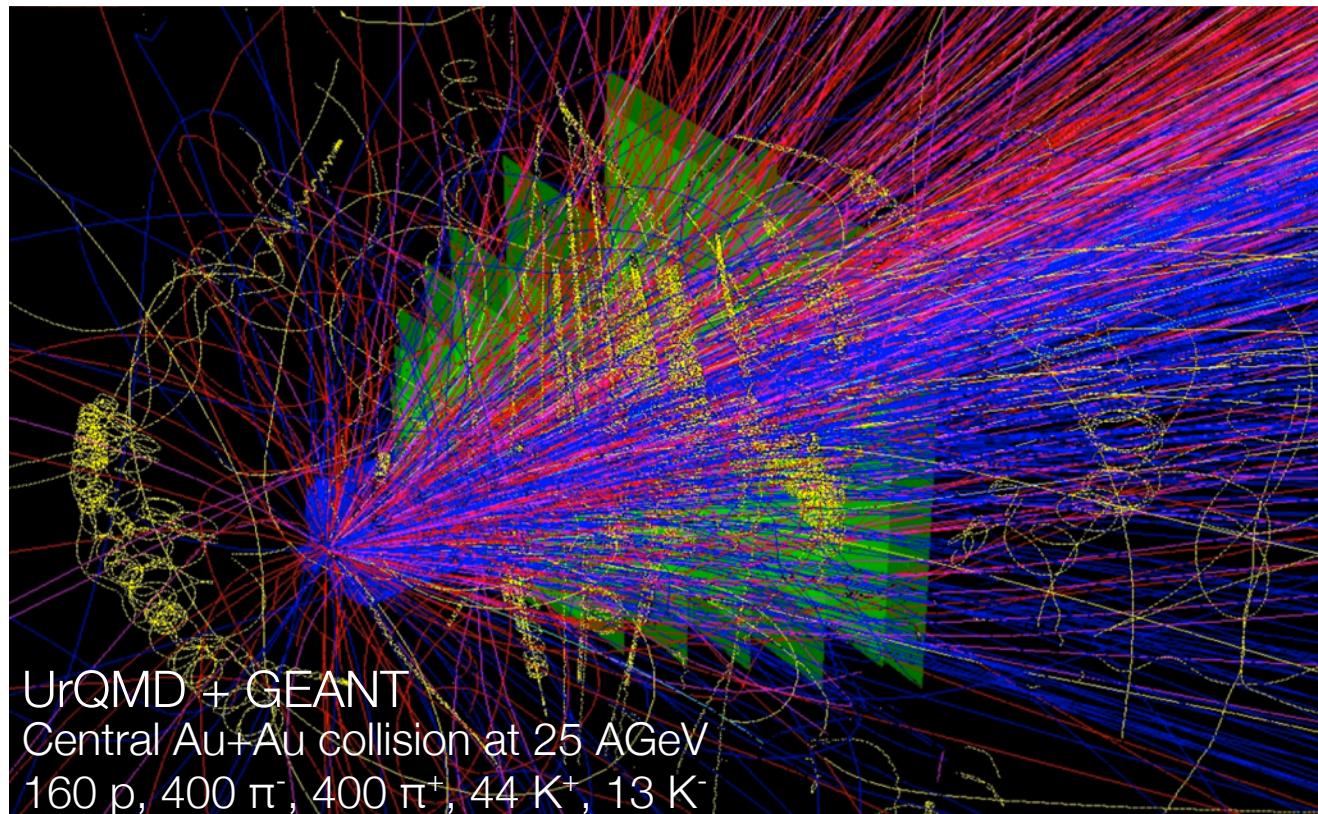
<http://compeng.uni-frankfurt.de>



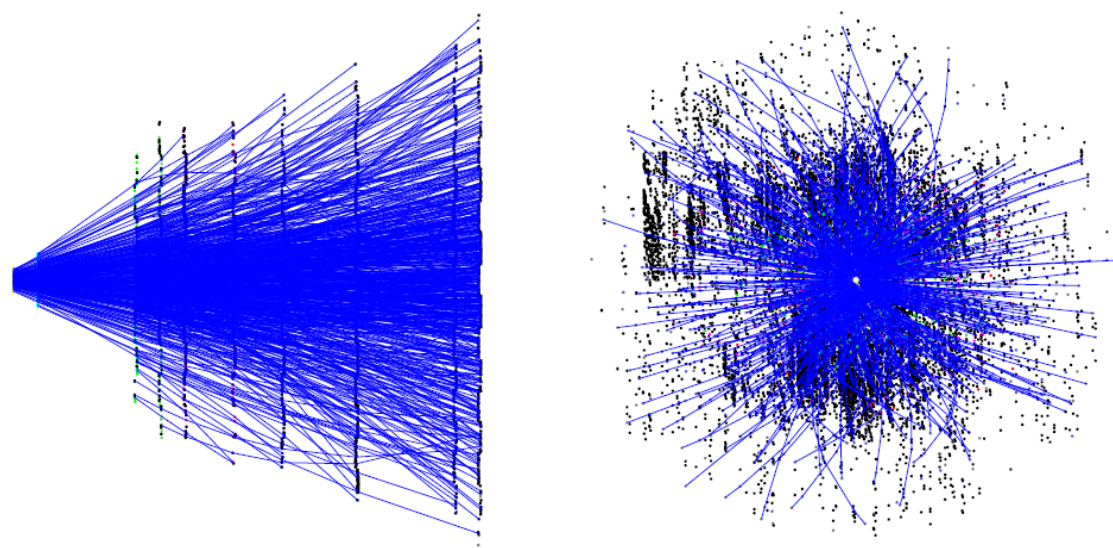
DPG Spring Meeting Darmstadt

2016-03-14

The CBM Experiment



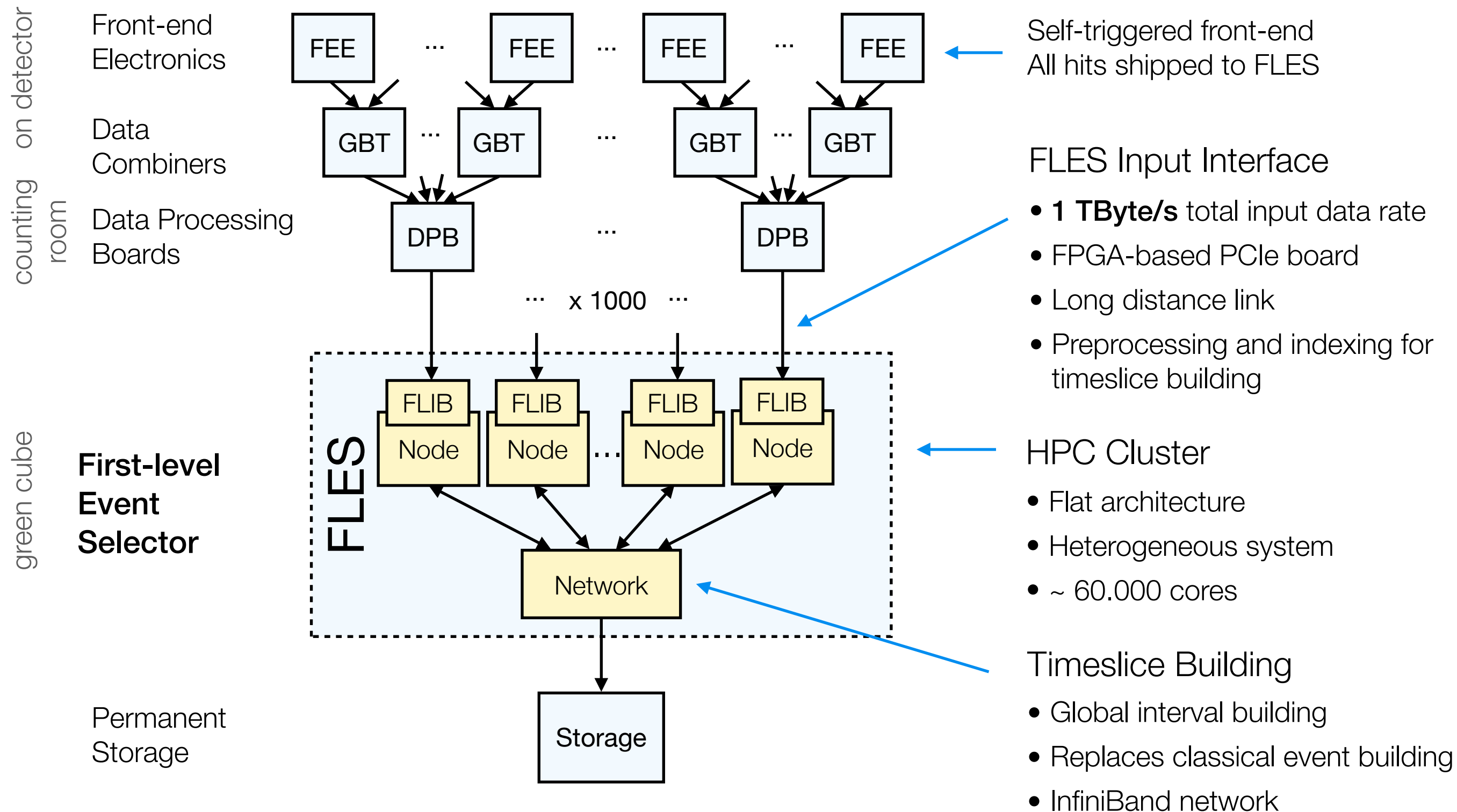
- Fixed target heavy ion experiment at FAIR
- Extreme reaction rates of 10 MHz and track densities of 1000 tracks in aperture
- Conventional trigger architecture not feasible



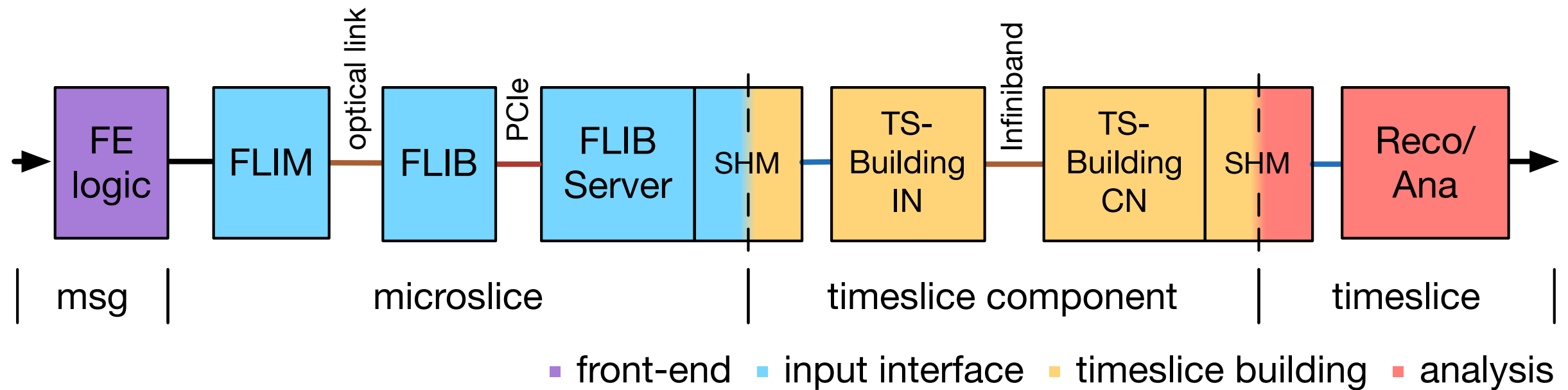
reconstructed event in STS

- ➔ Self-triggering free-streaming readout electronics
- ➔ Event selection exclusively done in an HPC cluster

First-Level Event Selector (FLES)



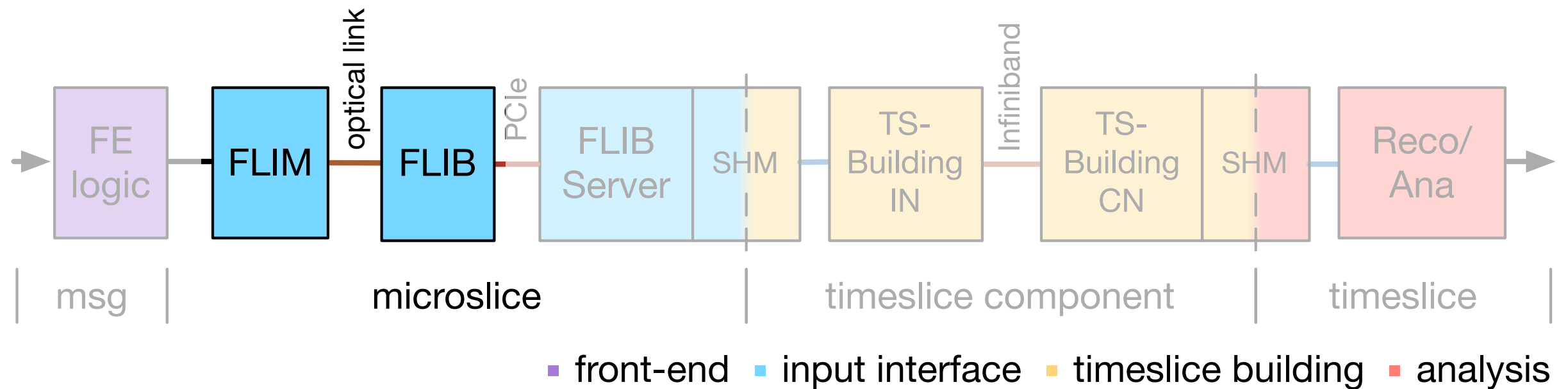
FLES Data Flow



This Talk: Input Interface

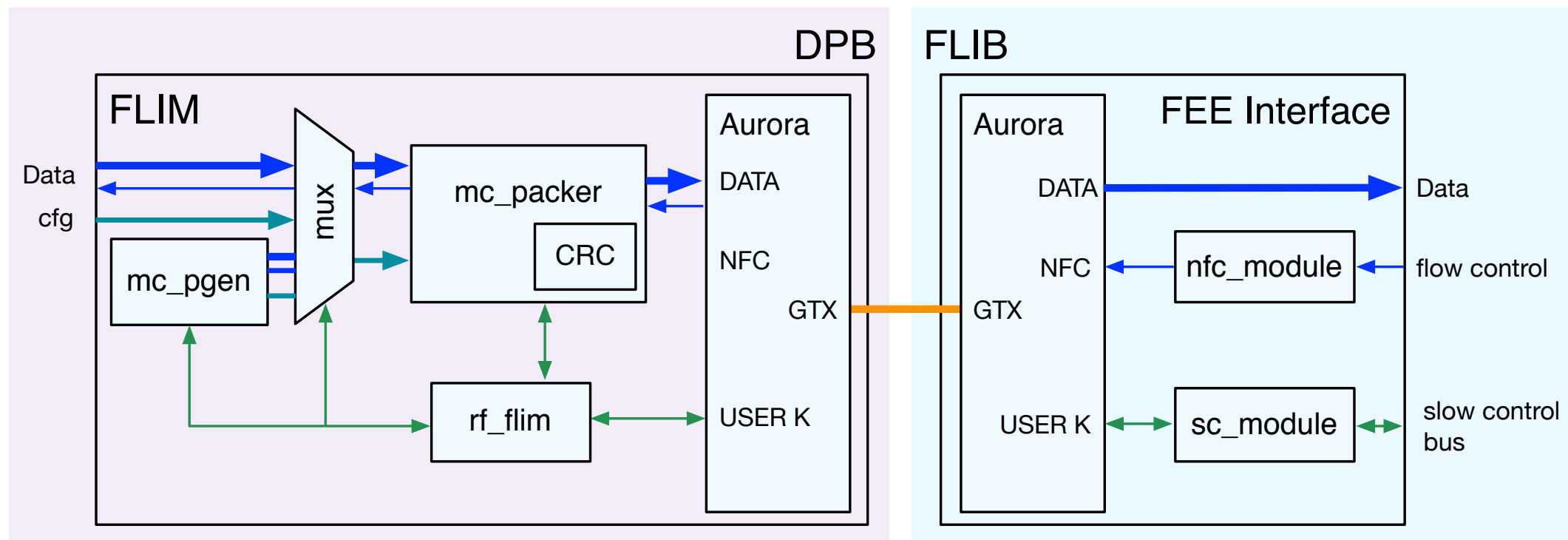
HK 54.5 Timeslice Building

FLES Data Flow



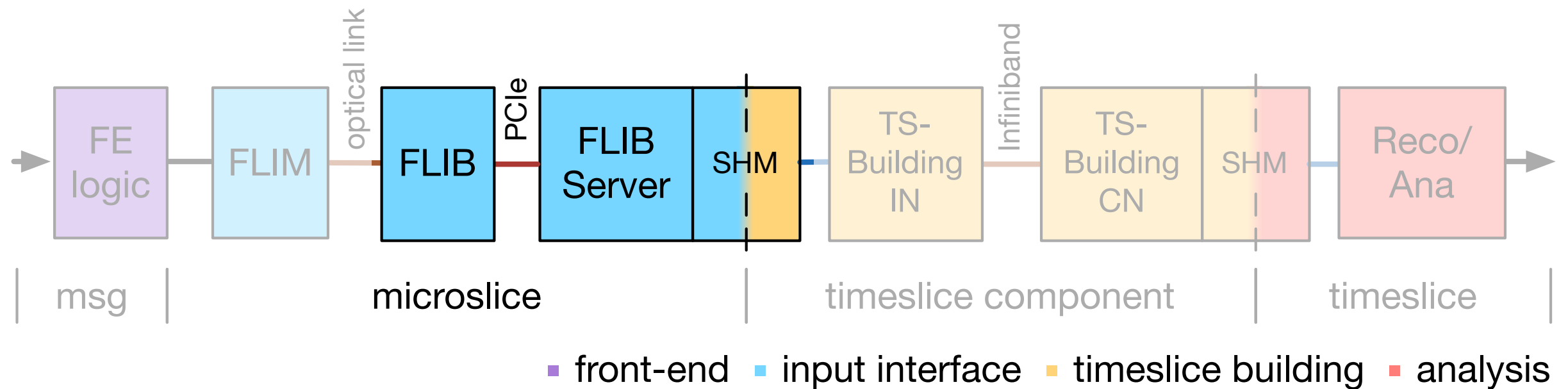
Frontend Interface

Frontend Interface



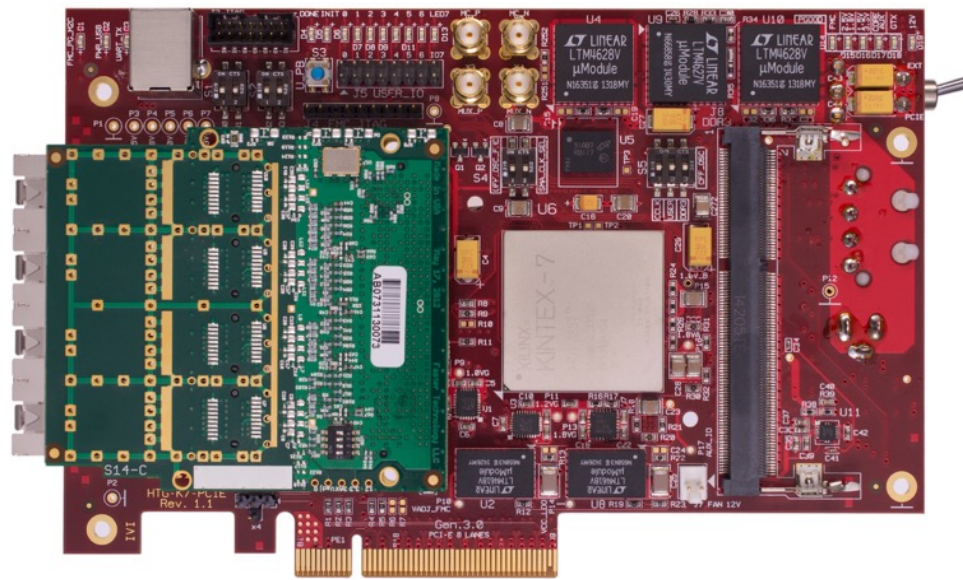
- 10 Gbit/s optical link
 - 64B/66B Aurora transport layer
 - CRC-32C (Castagnoli) end-to-end checksum on data content
- FLES Interface Module (FLIM)
 - HDL module implementing interface to front-end logic and link protocol
 - Runtime configuration via FLES link
 - Internal pattern generator for test and validation
- Bit error rates $< 10^{-16}$ per link measured in stability tests

FLES Data Flow

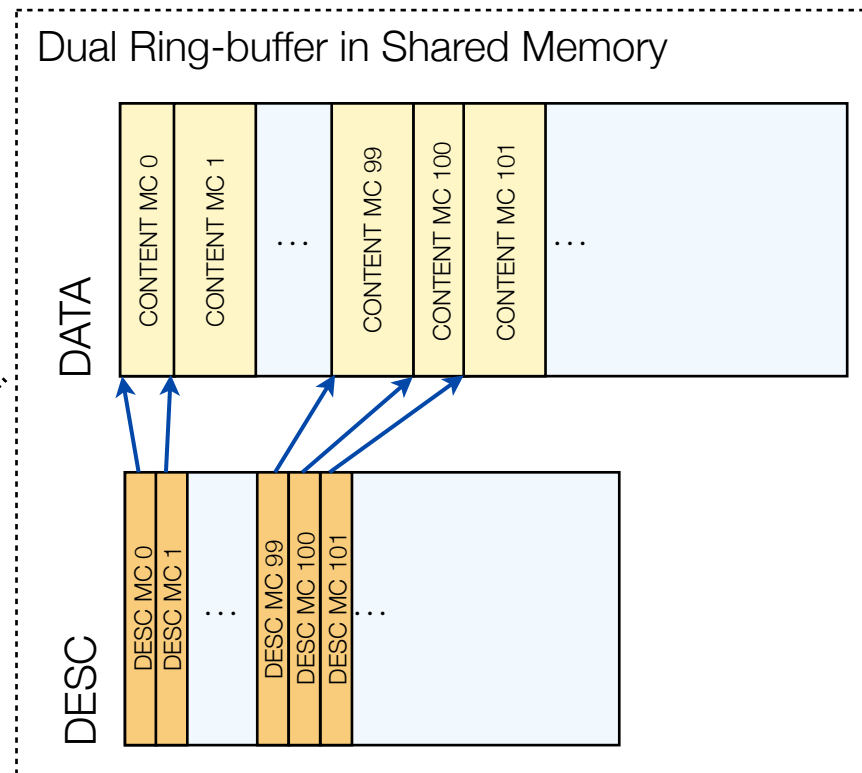
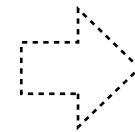


Host Interface

Host Interface

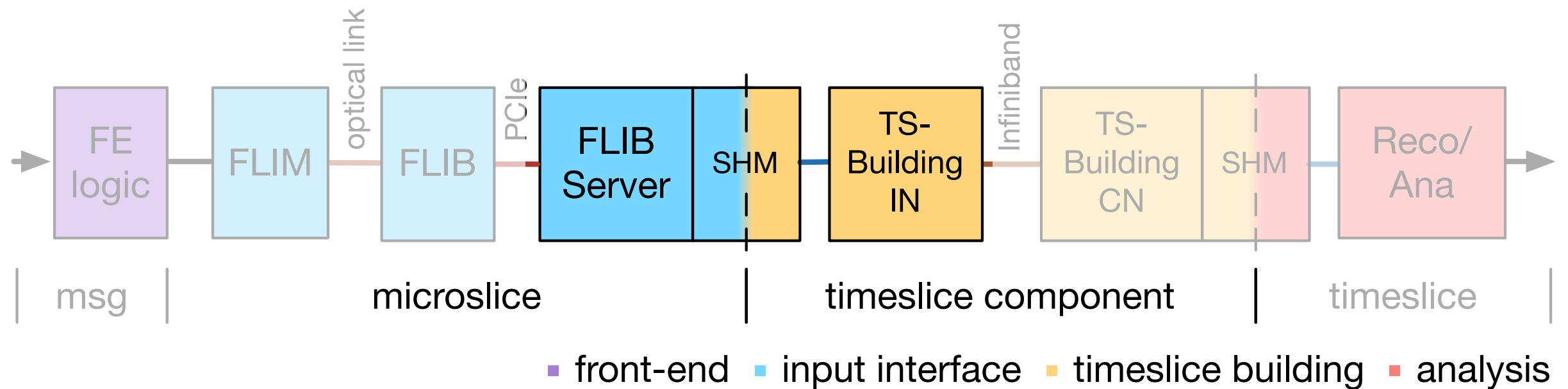


FLES Interface Board (FLIB)



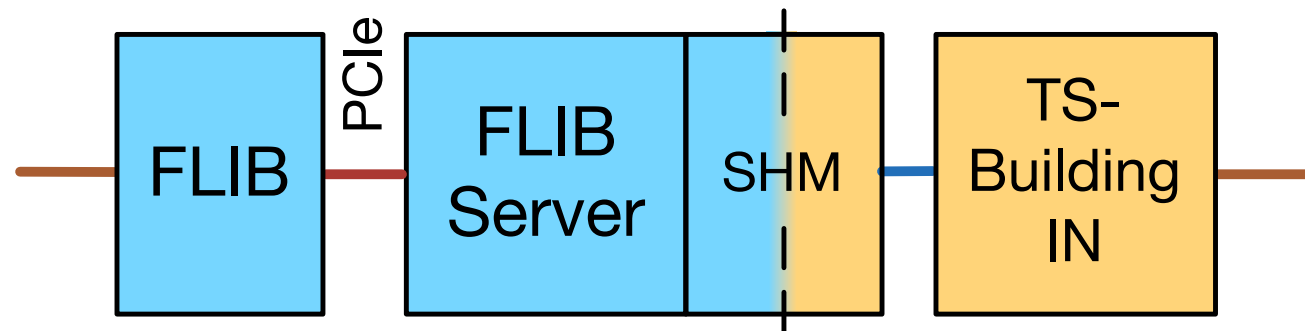
- PCIe FPGA board with custom HDL design
- Full offload DMA engine manages dual ring-buffer structure
 - Data buffer for microslice data content
 - Descriptor buffer for index table and microslice meta data
- Random access to data via index tables
- Device driver allows posix shared memory as DMA buffer

FLES Data Flow



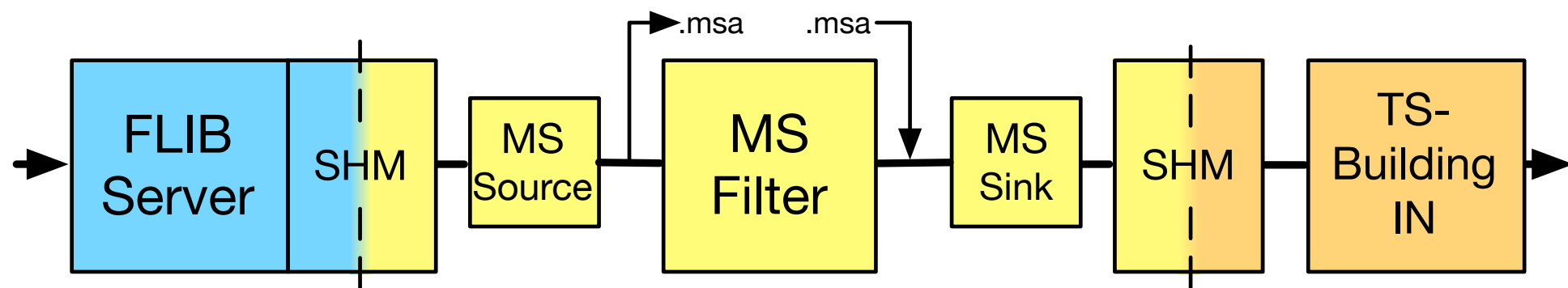
FLIB Server and Application Interface

Interface to Timeslice Building



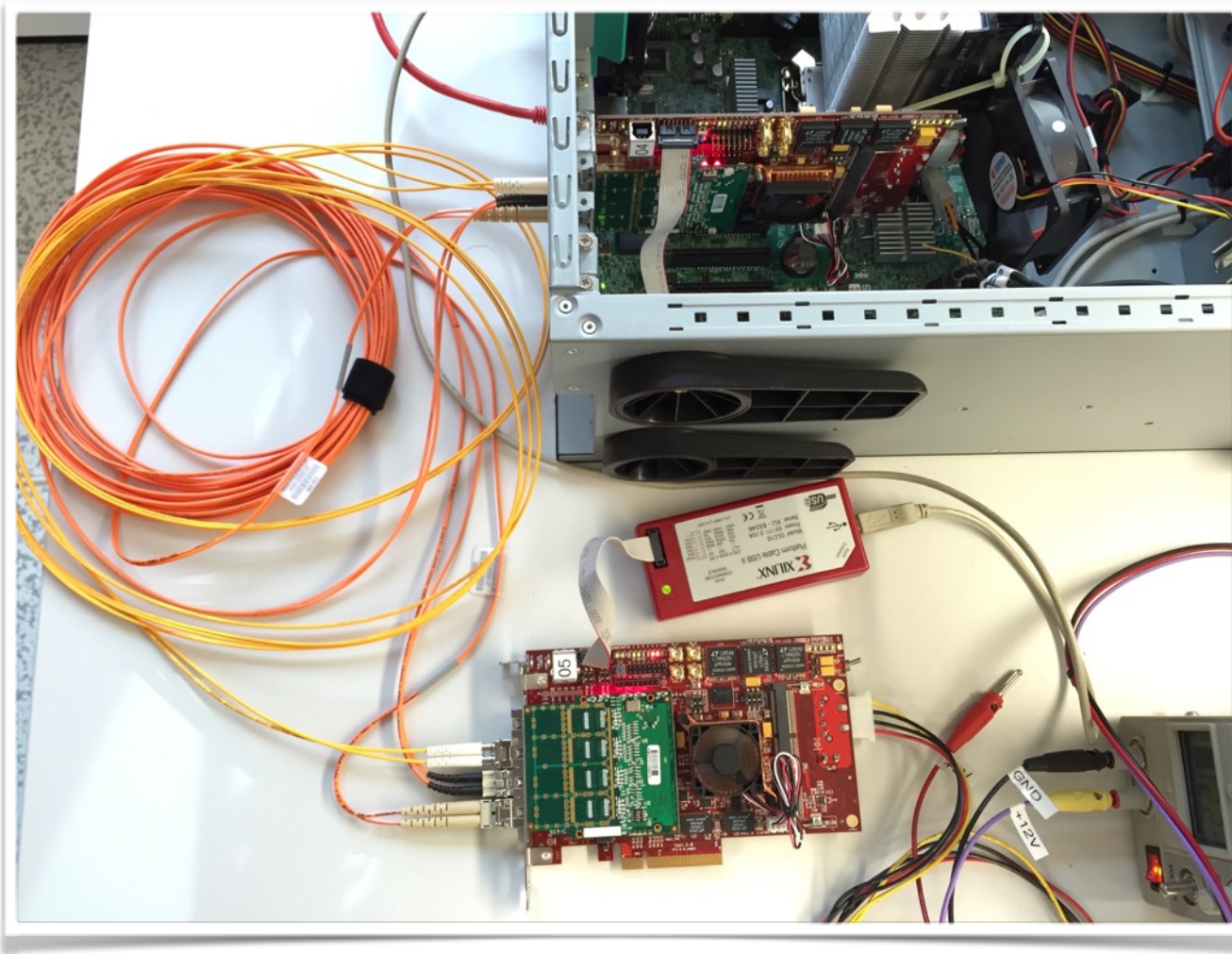
- Low-level dual ring-buffer interface
 - Raw access to DMA ring buffers in shared memory
 - Read/writer pointer updates towards FLIB/consumer
- Shared memory decouples input interface from TS building
- Buffers are directly used for Infiniband RDMA
- FLIB Server application manages communication to hardware
 - Multiple consumer possible

Microslice Interface



- High level access to microslice objects
 - Encapsulates low-level dual ring-buffer interface
 - Source to get microslices, sink to write microslices
- Allows to implement filters before timeslice building
- Possibility to store and load microslices to/from disk

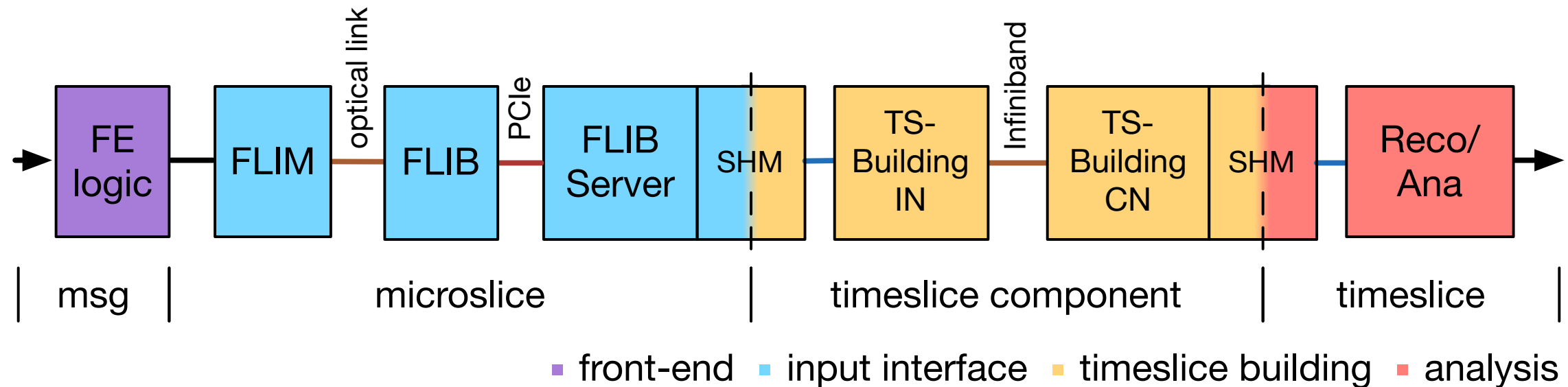
Status



Lab setup using DPB mockup

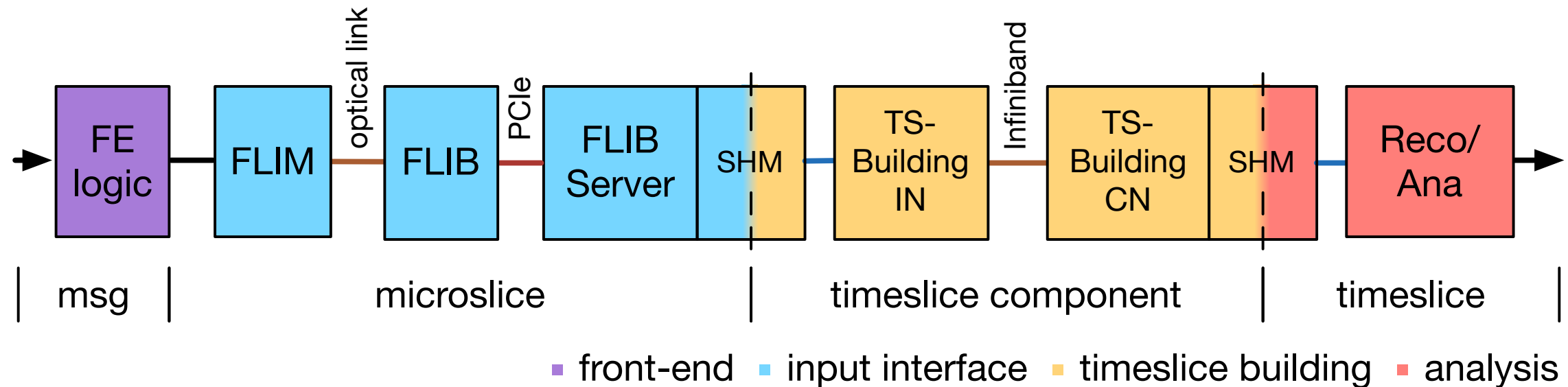
- Prototypes of all components are developed and in active use
- Fully integrated into flesnet software framework
- Integration of FLIM into DPB design started
- Intended to be used to readout the next CBM testbeam

Full System Startup Studies



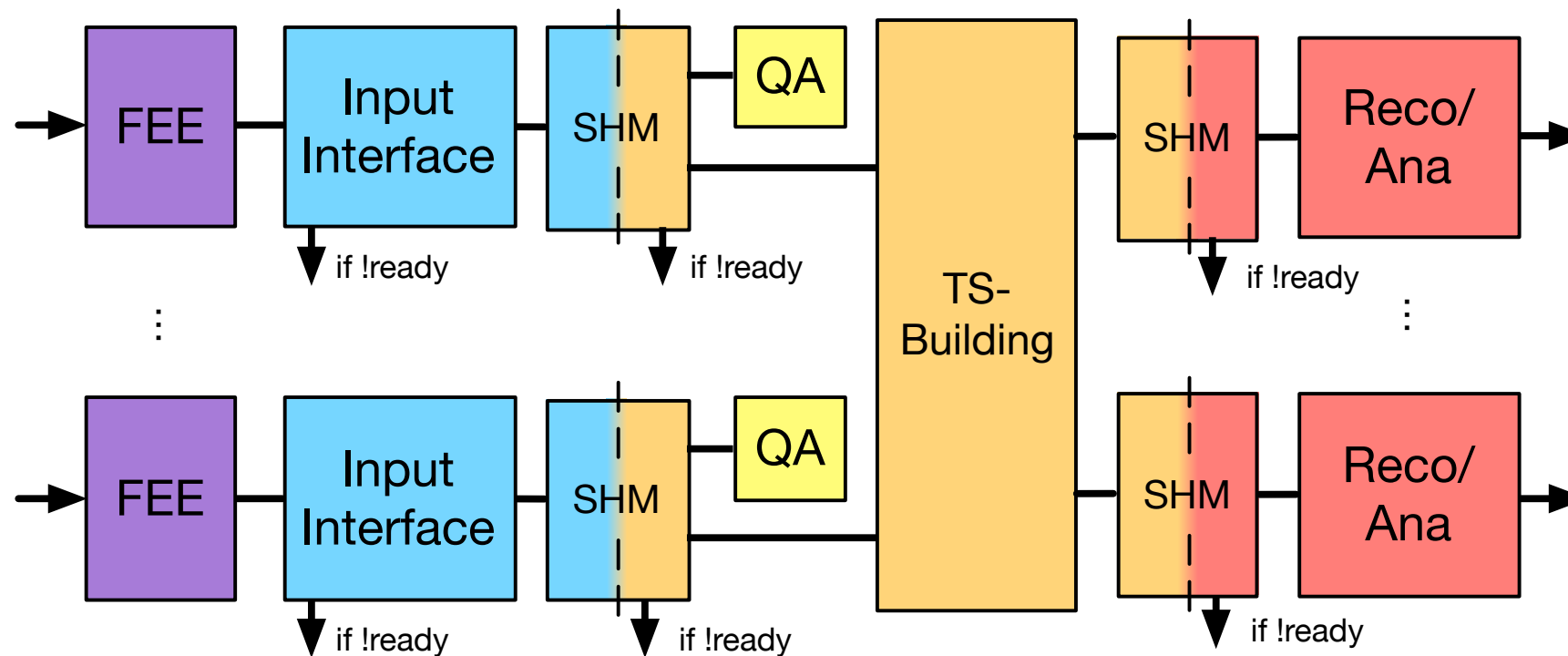
- Reminder:
 - CBM readout is a free-streaming data push architecture
 - FEE is mostly independent and asynchronous
 - „First event“ isn't defined until full analysis
- Synchronous startup leads to unnecessary complexity
 - Difficult to check readiness of all components before startup
 - Error probabilities for all components sum up
 - Inflexible in case of failing components

Asynchronous Startup



- Exploit asynchronous nature of the FEE
- Keep independent components independent
 - One link may produce data while another is still offline
 - Components may be restarted w/o restarting all components
- Decouple producers from consumers
 - Producers push data if ready, data is dropped if no consumer is available
 - Producers don't need to know states of consumers

Asynchronous startup in practice

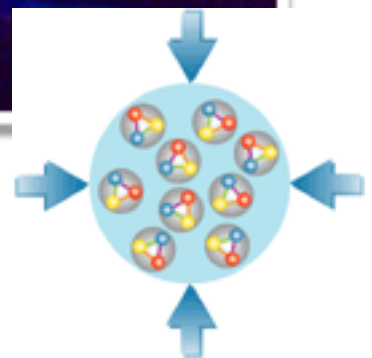
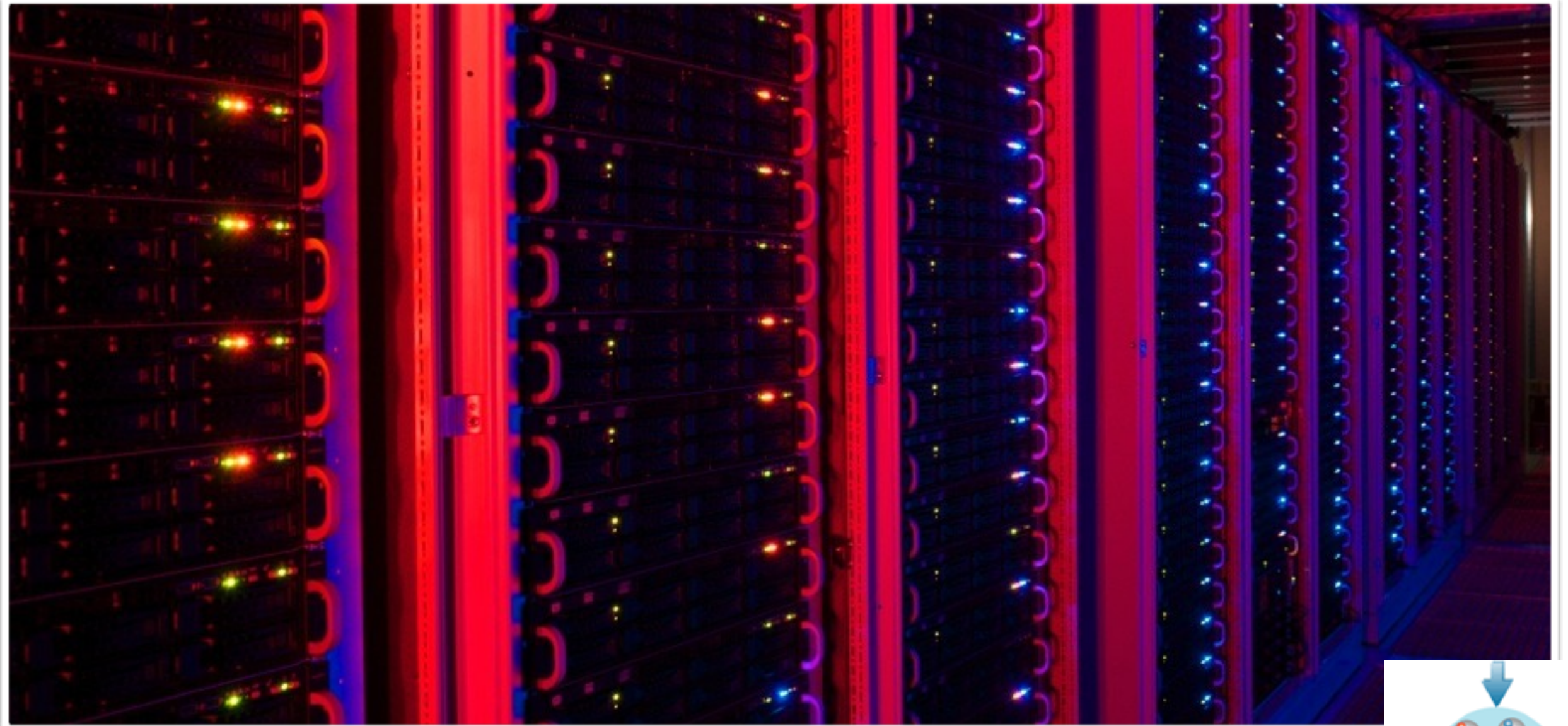


- FEE and input interface instances are started independent for all links
- QA processes check link health and mark links as ready
- Single components can be restarted individually if needed
- Timeslice building synchronizes using microslice time information

Summary

- Input interface implementation finalized and operational
→ will be used in the upcoming testbeams
- Dual ring-buffer shared memory interface
→ high-performance & good separation
- Asynchronous system startup can allow flexible error handling

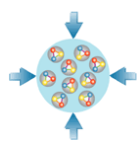
Thanks for your attention



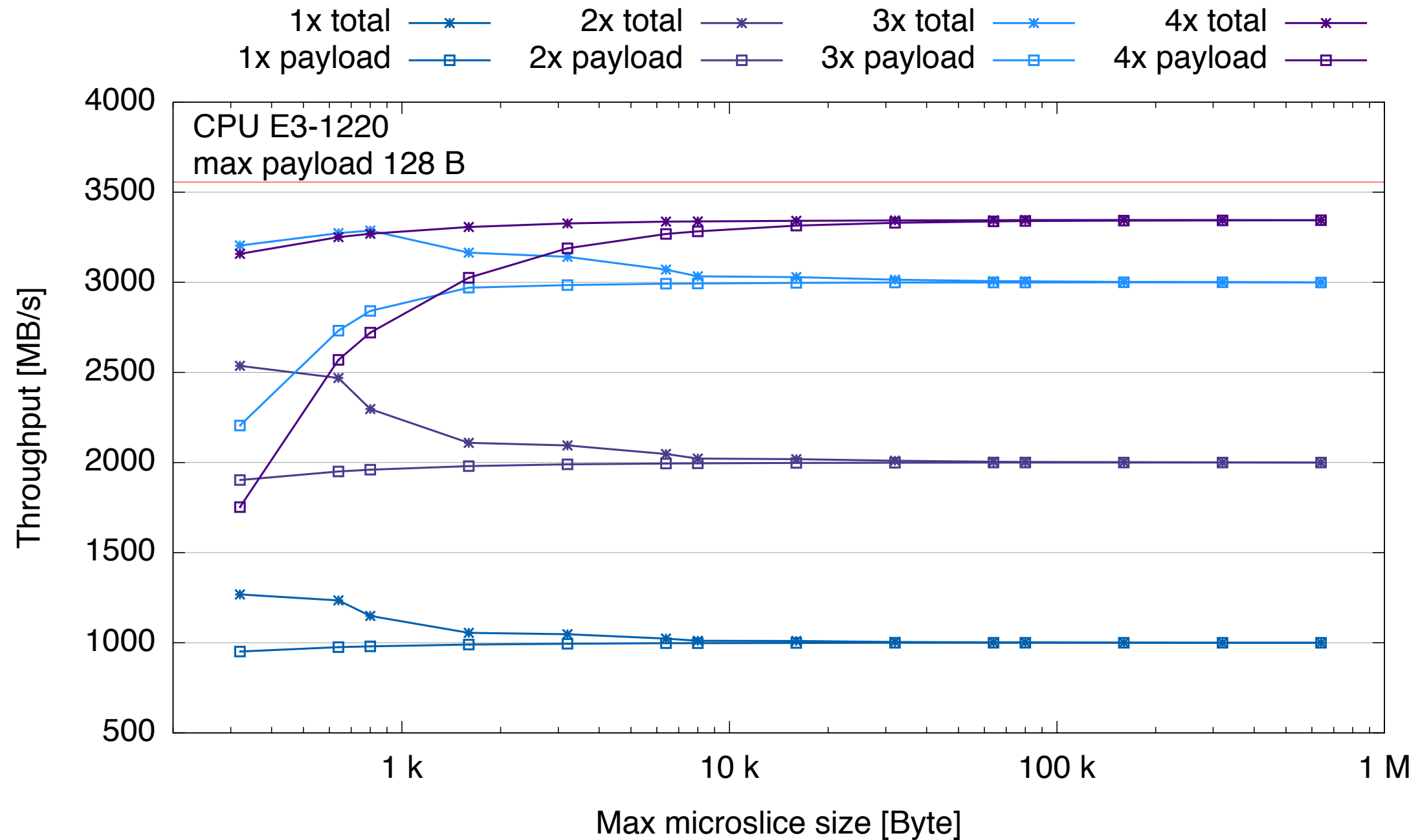
CBM

Dirk Hutter

hutter@compeng.uni-frankfurt.de

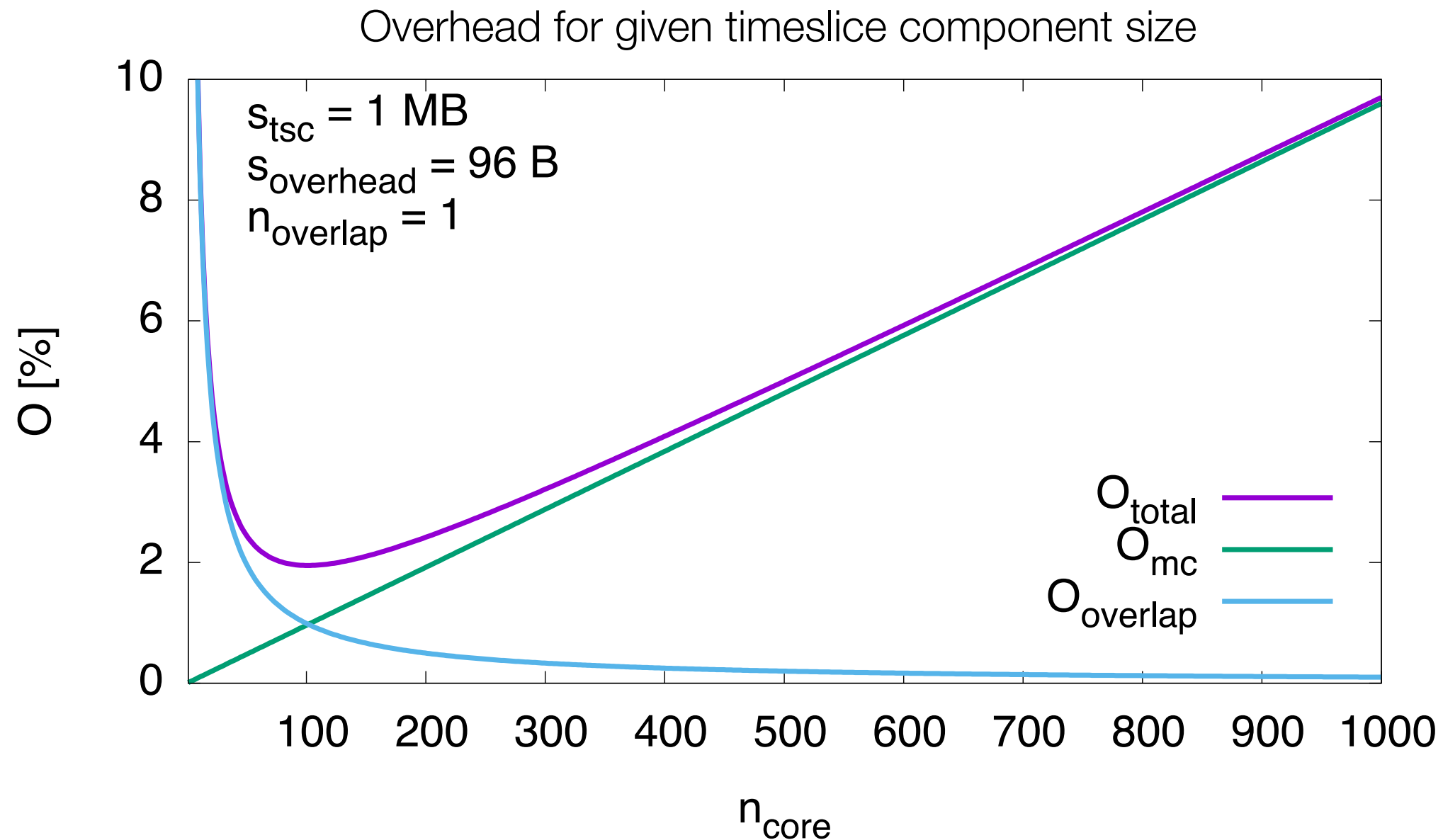


Readout Performance



- Data is generated by 10 GBits/s pattern generators and partitioned into microslices
- Throughput only limited by host system
- CPU load during transfer is negligible

Timeslice Overhead



O_{mc} Overhead from packaging of microslices
(packaging + alignment to PCIe max payload size)

O_{overlap} Overhead from duplication of microslices for overlap region

The Microslice Descriptor

```
struct MicrosliceDescriptor
{
    uint8_t    hdr_id;    // Header format identifier (0xDD)
    uint8_t    hdr_ver;   // Header format version (0x01)
    uint16_t    eq_id;     // Equipment identifier
    uint16_t    flags;     // Status and error flags
    uint8_t    sys_id;    // Subsystem identifier
    uint8_t    sys_ver;   // Subsystem format version
    uint64_t    idx;       // Microslice index
    uint32_t    crc;       // CRC32 checksum
    uint32_t    size;      // Content size (bytes)
    uint64_t    offset;    // Offset in event buffer (bytes)
};
```

- Microslice descriptor (32 bytes) per microslice
- Contains:
 - Meta information on data
 - Access information to the data content (size, offset)

API Example: The Timeslice Class

```
class Timeslice
{
public:
    uint64_t index() const;           // Timeslice index
    uint64_t num_components() const;  // Number of contributing inputs
    uint64_t num_core_microslices() const; // Number of core microslices
    uint64_t num_microslices(uint64_t component) const; // Total number of
                                                    // microslices

    // Retrieve the descriptor of a given microslice
    const MicrosliceDescriptor& descriptor(uint64_t component,
                                           uint64_t microslice) const;

    // Retrieve a pointer to the data content of a given microslice
    const void* content(uint64_t component, uint64_t microslice) const;
};
```

- The Timeslice base class provides access to the microslices of a single timeslice indexed by component and microslice numbers