Intermediate Network

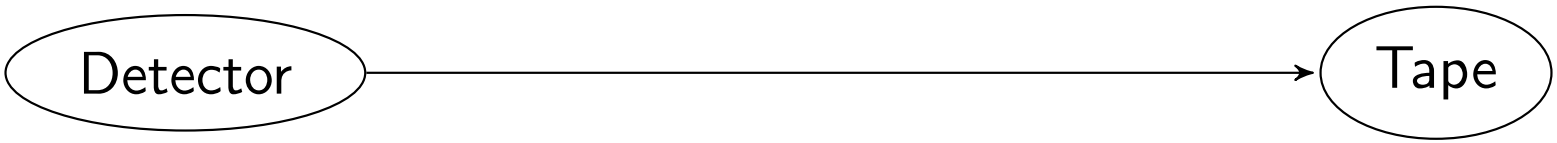
Jan C. Bernauer

EIC Streaming Readout Meeting Dec 2018

RBRC
RIKEN BNL Research Center

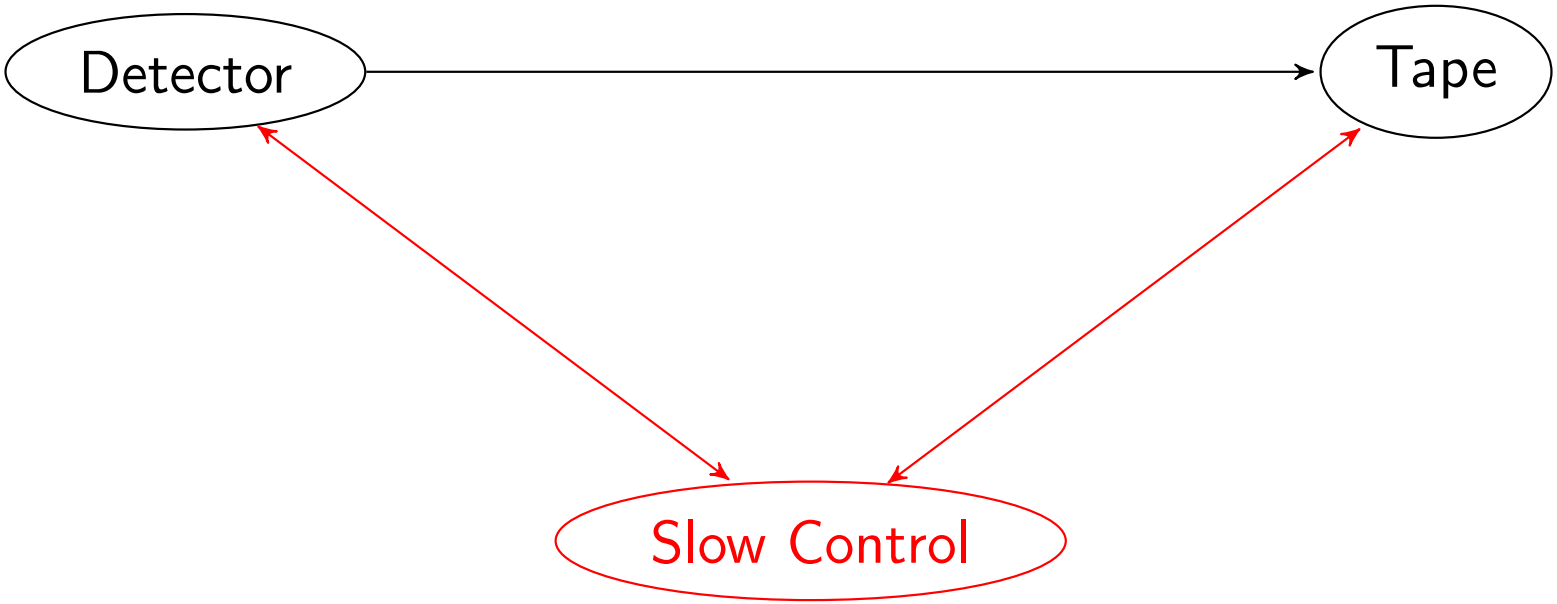
Stony Brook
University

What are we building: The simplest case



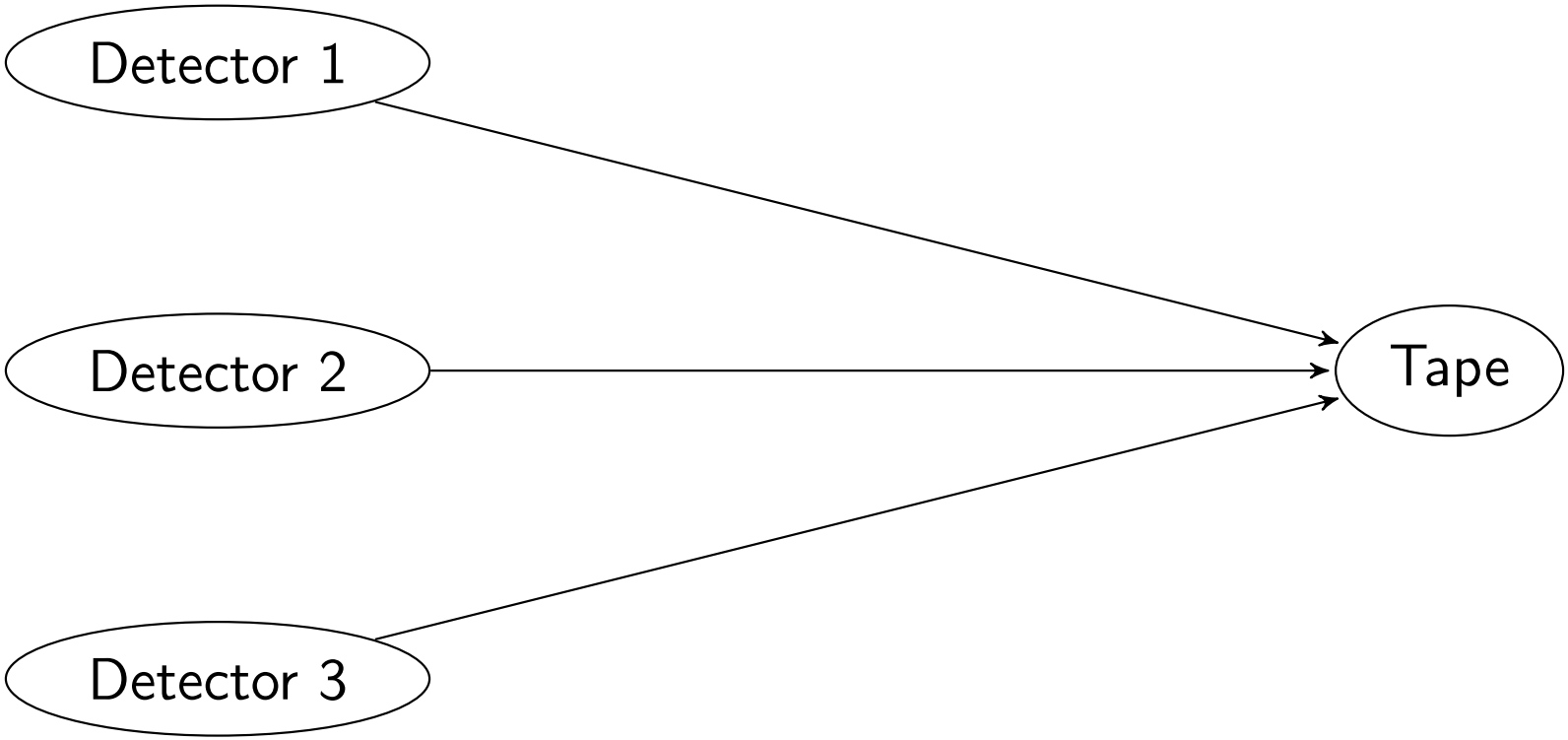
Arrows represent logical data streams, not necessarily physical connections.

What are we building: The simplest case



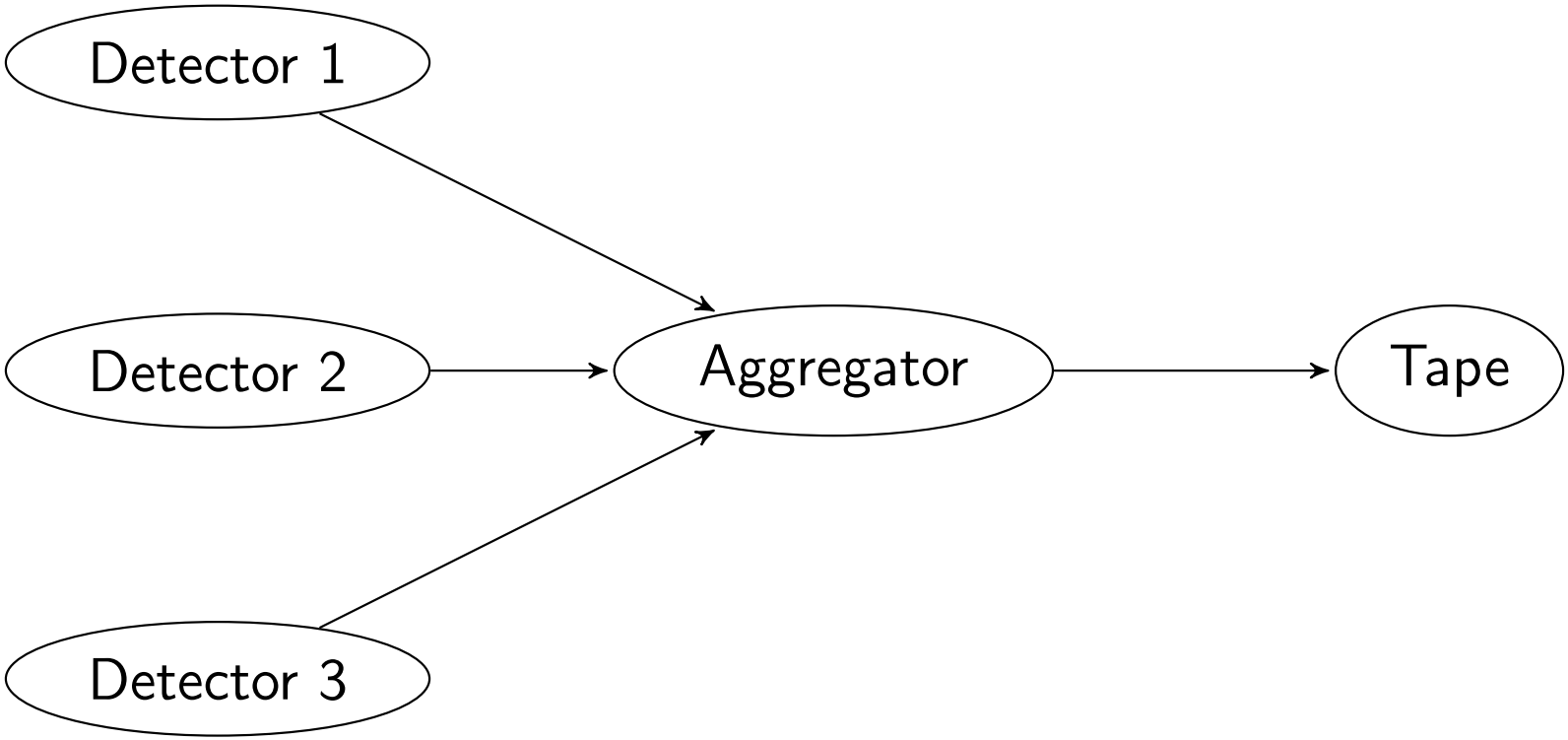
Arrows represent logical data streams, not necessarily physical connections.

What are we building: let's do more



Arrows represent logical data streams, not necessarily physical connections.

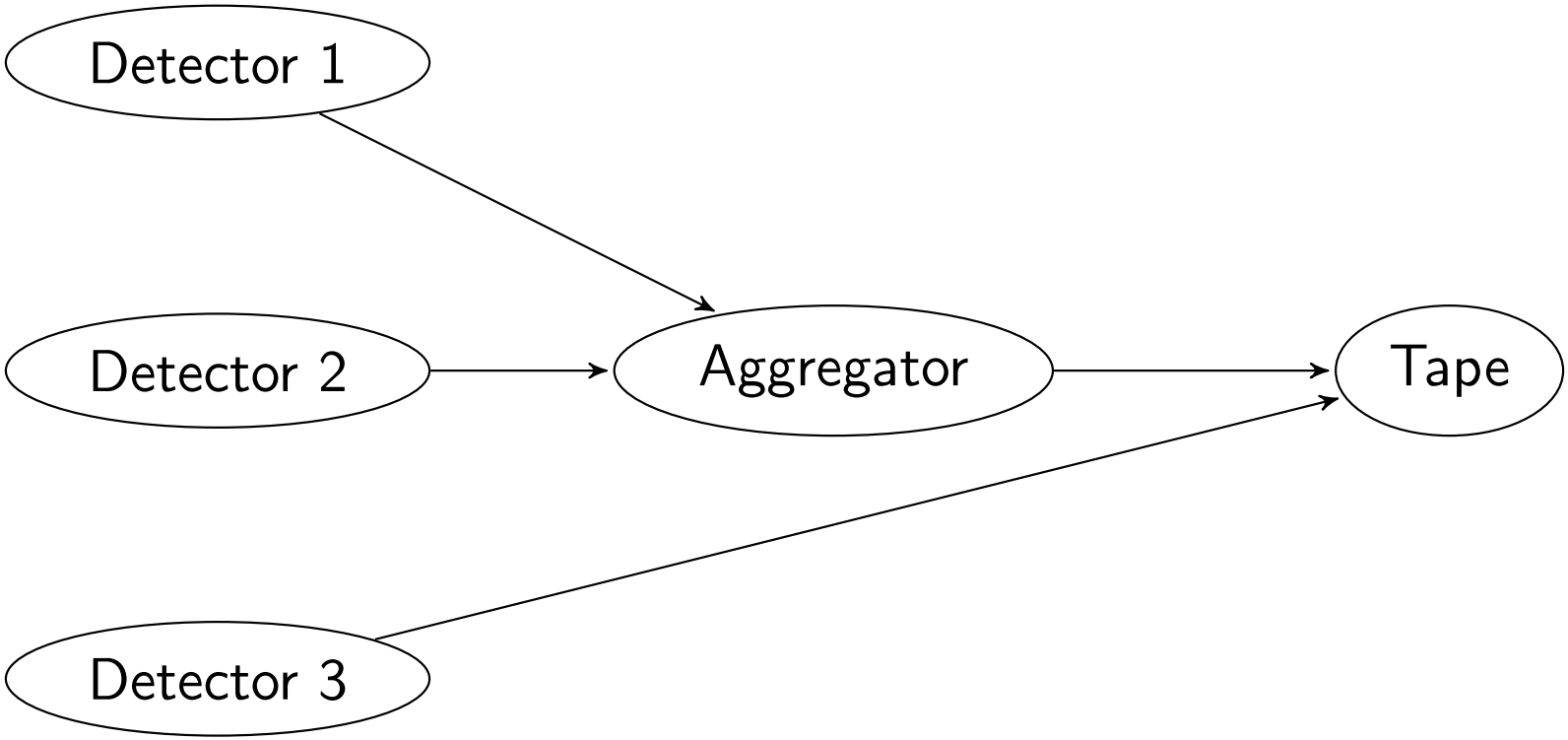
What are we building: let's do more



Arrows represent logical data streams, not necessarily physical connections.

Need to send data at the same time. This essentially rules out Circuit switching. We want a packet network.

What are we building: let's do more



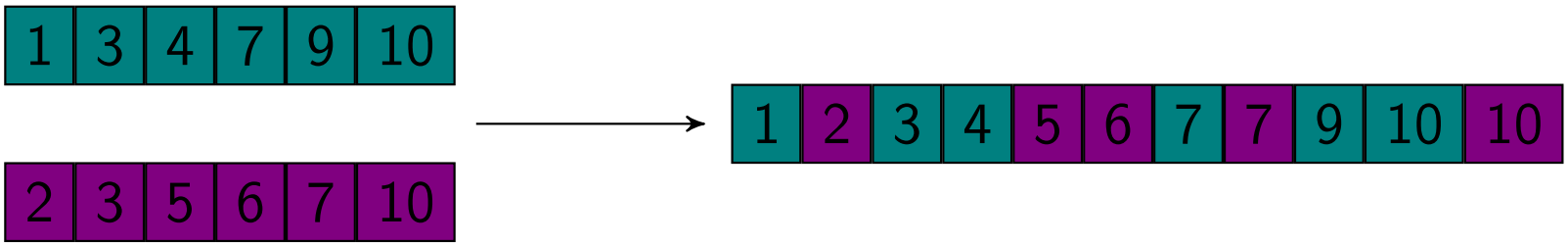
Arrows represent logical data streams, not necessarily physical connections.

Need to send data at the same time. This essentially rules out Circuit switching. We want a packet network.

Let's look at that aggregator

Takes N data streams in, multiplexes them to 1 output stream:
Each input stream i has time-ordered data words, $(t_0.d_0)_i$, $(t_1.d_1)_i$,...
Two possibilities:

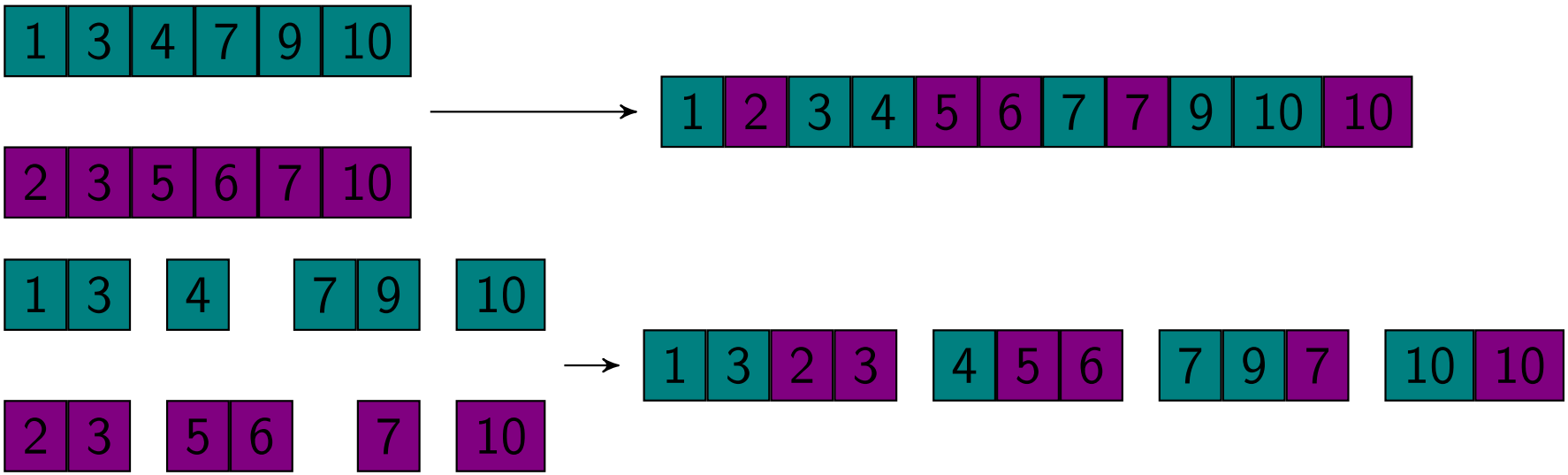
- Full time-ordered output: $t_k < t_{k+1}$



Let's look at that aggregator

Takes N data streams in, multiplexes them to 1 output stream:
Each input stream i has time-ordered data words, $(t_0.d_0)_i$, $(t_1.d_1)_i$,...
Two possibilities:

- Full time-ordered output: $t_k < t_{k+1}$
- Frame-ordered. Pick time buckets of duration T, then
  - copy all data from channel 1 where $nT \leq t < (n+1)T$,
  - copy all data from channel 2 where ...



Multiplexer

Fully time ordered:

- Computational expensive on the multiplexer
- Consumer has advantage if looking at many channels and interesting time spans $\ll T$

Frame-ordered:

- Computational less expensive on the multiplexer, mainly straight copies
- Consumer has advantag if looking at few channels (skip ahead), or time spans $\gtrsim T$
- Worse case: Complexity moved from multiplexer to consumer

Filters

The aggregator:

- consumes N input streams
- does something
- produces 1 output stream

Let's generalize:

A filter:

- consumes N input streams
- does something
- produces M output streams

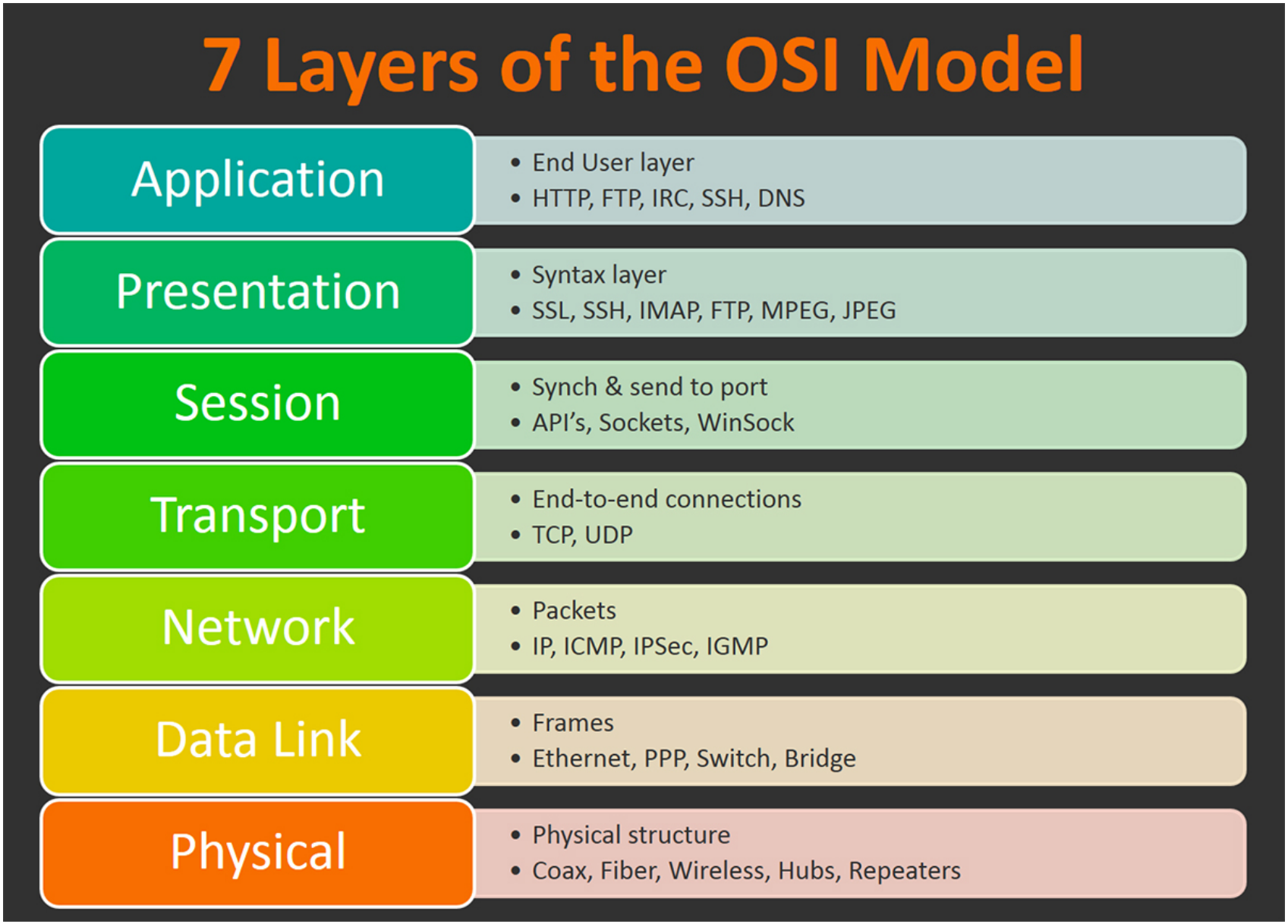
Concepts

- Sources / Producers: 0 inputs, M outputs
  - Detectors
  - Slow control info
  - Replayed data
- Sinks/Consumers: N inputs, 0 outputs
  - long term storage
  - displays
- Filters: N inputs, M outputs
  - Aggregators
  - Event builder
  - (online) trackers
  - Data selectors

Opinion 1

- With these three concepts, we can build a network describing any streaming readout solution.
- The physical readout network will resemble this network.
- For composition: Good if off-device physical interface is "the same".
- It makes sense that the analysis software, online and offline, is organized the same way.

Open Systems Interconnection layers



1) Physical

- Will have multiple solutions
- Some of them might be on-chip!
  - Think multi-channel streaming ADC. Each channel will be a produce, and there is a on-fpga mixer.
- Local area: Probably what ethernet can use, because next layer

2) Data Link

- Will have multiple solutions.
- On-chip: Wishbone, AXI ...
- Likely with data frames, but doesn't have to be.
- Hardware addresses
- Local area:
  - Ethernet
  - Roll-your-own
  - ATM, Frame Relay

Opinion 2: Why Ethernet

- Ethernet is dirt cheap.
- IP available.
- Don't have to design infrastructure. Switches are cheap.
- All computers have an ethernet interface.
- Backward- and forward compatible. Better than PCIe, better than USB.
- Slow control back channel for free.
- Some more overhead, wasted bandwidth.
- I don't think we can get time synchronization to the level we need

3) Network

- Packet routing between segments
- Logical addresses
  - Easy to provision/replace hardware
- Could be skipped for higher efficiency
- Essentially two options:
  - IP
  - Roll your own

Opinion 3: Why IP

- Allow WAN type of networking. Stream data to data center.
- Need some form to jump from ethernet segment to segment.
- SOO much hardware and software exists.
- Does give addition overhead, wasted bandwidth.

4) Transport

- On a packet network, two options
  - Connection-oriented protocols: TCP
  - Connection-less protocols: UDP
- But RTP, QUIC=http/3 uses UDP to create connection-oriented protocol.

Open question: TCP or UDP

- Layers below do NOT guarantee in-order delivery. Might drop packages!
- UDP will pass this onward. UDP can boardcast/multicast
- TCP guarantees in-order delivery, no loss
  - Will request resend of packages which are not received in time
  - This means source needs memory to preserve already sent data

Open question: TCP or UDP

- Layers below do NOT guarantee in-order delivery. Might drop packages!
- UDP will pass this onward. UDP can boardcast/multicast
- TCP guarantees in-order delivery, no loss
  - Will request resend of packages which are not received in time
  - This means source needs memory to preserve already sent data
- My gut feeling: TCP probably worth it.
- Most alternative implementations of layer 1-4 are fiber-optical serial links.

5) Session

Standard: Socket API

Also: User-space network stacks!

- If we stay with the usual net stack, all is provided.
- If not, merge with upper layers?

6) Presentation Layer

This is where the software people will work on!

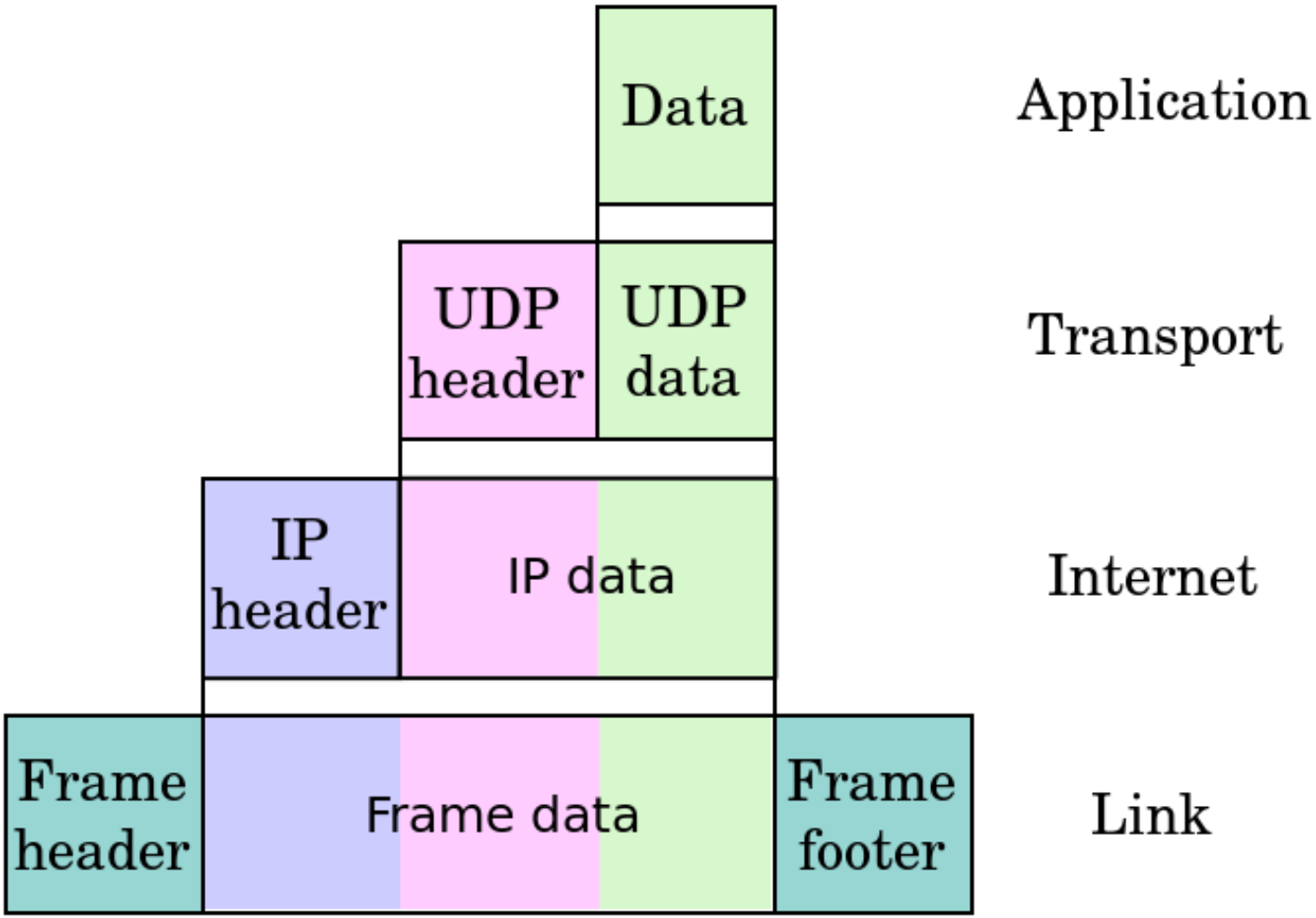
- Define representation of data in structures instead of datagrams
- (De)-Serialization of data
- If we do it right, mostly independent of layers 1-5
- See below

7) Application layer

This is not the application!

- Actual protocol spoken between network partners
- Think HTTP, DNS etc..

How bad is the overhead, really?



Overhead in numbers

- Standard Ethernet frame size: 1538 octets (=bytes), up to 1500 bytes data: 97.5% efficiency
- Jumbo frames up to 9000 bytes payload, so 99.54% efficiency
- IPv4 header 20 bytes, IPv6: 40 Bytes: Worst case 94.9% efficiency
- UDP header: 8 Bytes. 94.4% efficiency. TCP header: 20 Bytes. 93.6% efficiency
- Is a custom solution less than 7% more expensive?

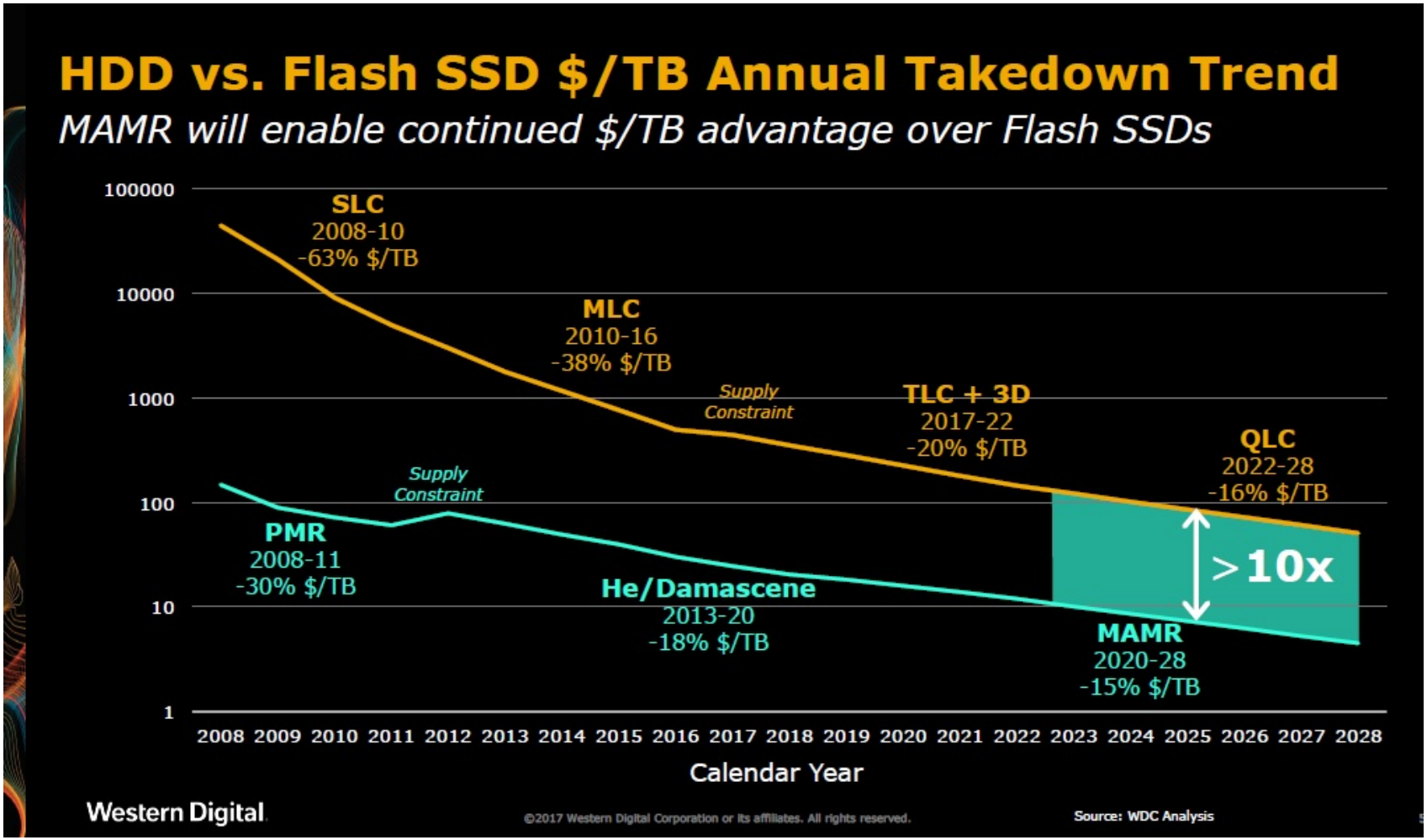
Alternatives to TCP and UDP

- Can not realistically go "below" UDP. No software support in OS.
- But TCP provides things we might not need:
  - Ordered data $\leftarrow$ we need that, but also true for UDP and simple networks
  - Reliable transmission $\leftarrow$ we might not need that
  - Congestion control $\leftarrow$ probably not what we need
- Might be able to push what we need into application layer. Easier on FPGA

Scaling of Ethernet

- 1000Base-T Ethernet is standard. 125 MByte/s
  - Virtually all computers, FPGAs....
- 10GBase-XX is commonly available COTS. 1.25 GByte/s
  - Copper: Switch port ∼$50, network card ∼$100
  - Available in all bigger FPGAs
- 40GbE. 5 GByte/s
  - 32 port switch for ∼1000$, network card ∼$150
  - Cables expensive, short :)
  - Available in latest gen FPGAs
- 100GbE, 12.5 GByte/s
  - switch $200/port, network card ∼$500
  - That's ∼4 Bytes/cycle on a modern CPU.

Side note: Data storage



Looks like \$8/TByte. Let's assume \$20/TByte

Let's assume 1 mio\$/year. That's 50 PByte/year, 136 TByte/day, or 1.6 GByte/s.

Opinions:

- Software stack
  - Highest level gets data on frame level
  - Next lowest level should assume FIFO semantics
  - Plugable support for TCP, UDP etc. Start with TCP
- Network mostly based on ethernet, IP
  - Can use DHCP, BOOTP etc for bring-up!

Wish list for Application Layer

- Need to organize which source connects to which sink.
  - Round-Robin or job request for load-balancing?
- Send data to more than one node, for selected T-bins, for monitoring
- Diagnose flow limits, dead time etc

Wish list for Presentation Layer

- Efficient implementation on CPU and FPGA
- Easy to add/drop substreams, parts of streams.
- Must decode enough without configuration
- Empty channels need 0 bytes
- Flexible for wide range of data types
- Self-documenting
- Bindings to many languages

A look at MPEG/DVB/ATSC

- Digital video broadcast is a stream of multiple channels, video and audio, time-synchronized
- Widely distributed, IP available?

A look at MPEG/DVB/ATSC

- Digital video broadcast is a stream of multiple channels, video and audio, time-synchronized
- Widely distributed, IP available?Only some.
- Three types of streams:
  - Transport streamProgram clock reference
  - Program stream
  - Packetized elementary stream

MPEG Transport stream

- Combines a number of substreams, mostly packetized elementary streams
- Sequence of packets, 188 bytes long
- Each packet has data from one substream
- Not every packet has timing information.
- Interesting fields/items
  - Random Access indicator: Data from here on "makes sense"
  - Priority fields. Out-of-band data?
  - Contains multiple programs (group of PES) which can have different time base
  - Always has length field.

Program Stream

- Similar to TS
  - Designed for "Reasonably reliable media". Less protection against data loss.
  - Only one time base

Packetized elementary stream

- Packetized version of an elementary stream
- Can be of any length (for video)

Features present?

- Efficient implementation on CPU and FPGA
- Easy to add/drop substreams, parts of streams.
- Must decode enough without configuration
- Empty channels need 0 bytes
- Flexible for wide range of data types
- Self-documenting
- Bindings to many languages

Take away

- Protocol itself not suitable for us
- But some ideas interesting:
  - PES packets don't need to match TS packets
  - Does "different time bases" help us?
  - Sync words to find headers in stream

Close out