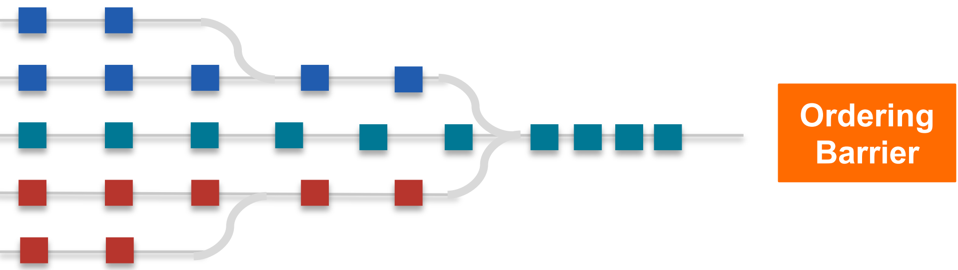
ETH zürich

Triggerless Data Acquisition for Online Reconstruction in High-Rate Experiments

Ring-CAM timestamp resequencing in FPGA

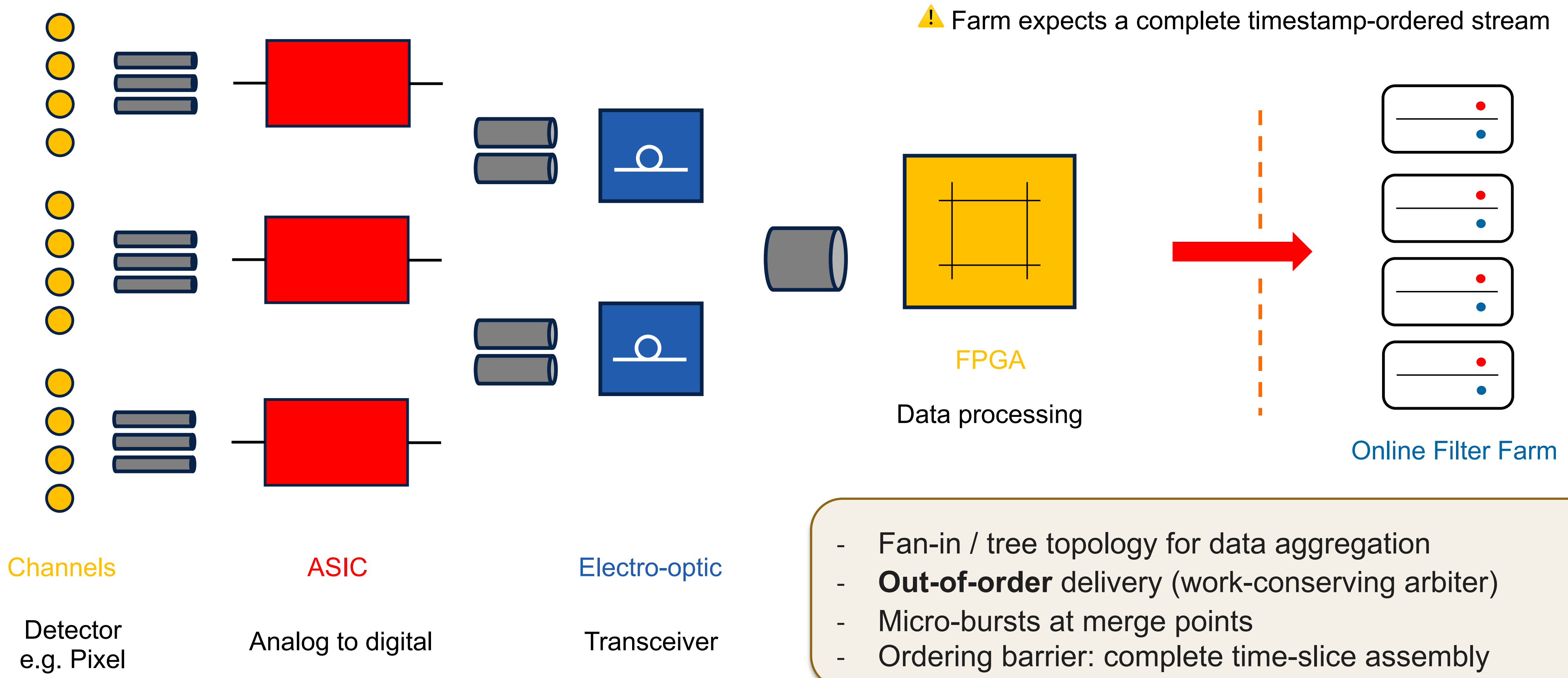


16-min talk · Data Acquisition and Trigger Architectures Session

Yifeng Wang (ETH Zürich)

25th IEEE Real Time Conference
La Biodola, Elba, Italy

Triggerless DAQ: Fan-In Creates an Ordering Barrier



- Fan-in / tree topology for data aggregation
- **Out-of-order** delivery (work-conserving arbiter)
- Micro-bursts at merge points
- Ordering barrier: complete time-slice assembly

A detector time slice is distributed across many readout lanes but requires synchronization at the last stage

ETH zürich

HEP Bottleneck: Time-Slice Assembly, Not Average Bandwidth

Future Physics Demands

- High data rate, low physics loss
- Smart-trigger and triggerless readout modes
- Online filtering is the prevailing strategy
- Traffic driven by physics occupancy, not a fixed trigger accept rate

Engineering blockers

- ✅ Faster transceivers
- ✅ Better integration methods, e.g. chiplets and silicon-photonic
- Remaining blocker: deterministic time-slice *(frame)* transport across many lanes
- ***frame*** = *fixed-width time slice.*

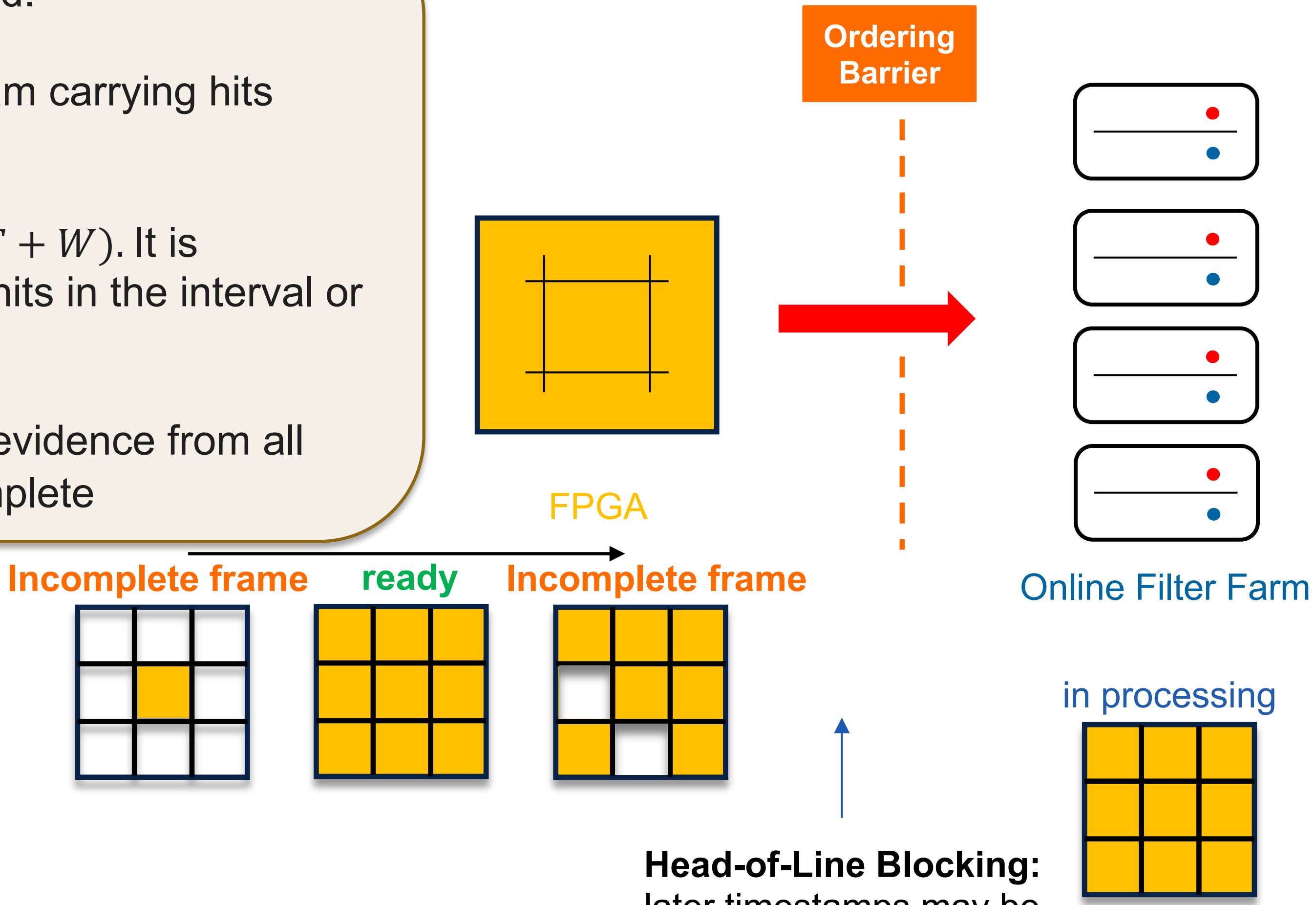
At high rate, the critical path shifts from average bandwidth to bounded lane skew and complete time-slice *(frame)* delivery

ETH zürich

Time-Slice Assembly Waits for the Slowest Lane

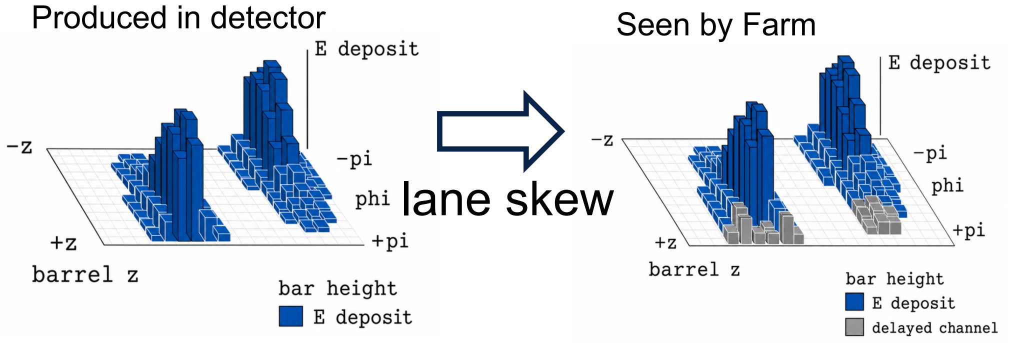
- ***Hit:*** a timestamped detector readout word.
- ***Lane/link:*** an independent readout stream carrying hits and/or progress markers.
- ***Time slice***: fixed timestamp interval $[T, T+W)$. It is complete when all lanes have delivered hits in the interval or advanced a watermark beyond it.
- A frame needs hits or empty/watermark evidence from all lanes; lane skew makes the frame incomplete

The point where downstream processing cannot safely close or process timestamp interval *T* until every input lane has either delivered all hits for *T* or advanced a watermark beyond *T*.



Head-of-Line Blocking: later timestamps may be present but cannot be released before the incomplete earlier frame

Example Event Display of Sparse/Zero-suppressed Detector
a lane may have no hit

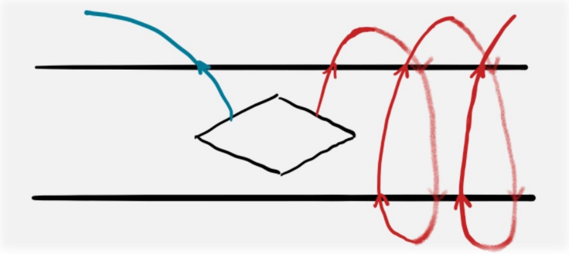


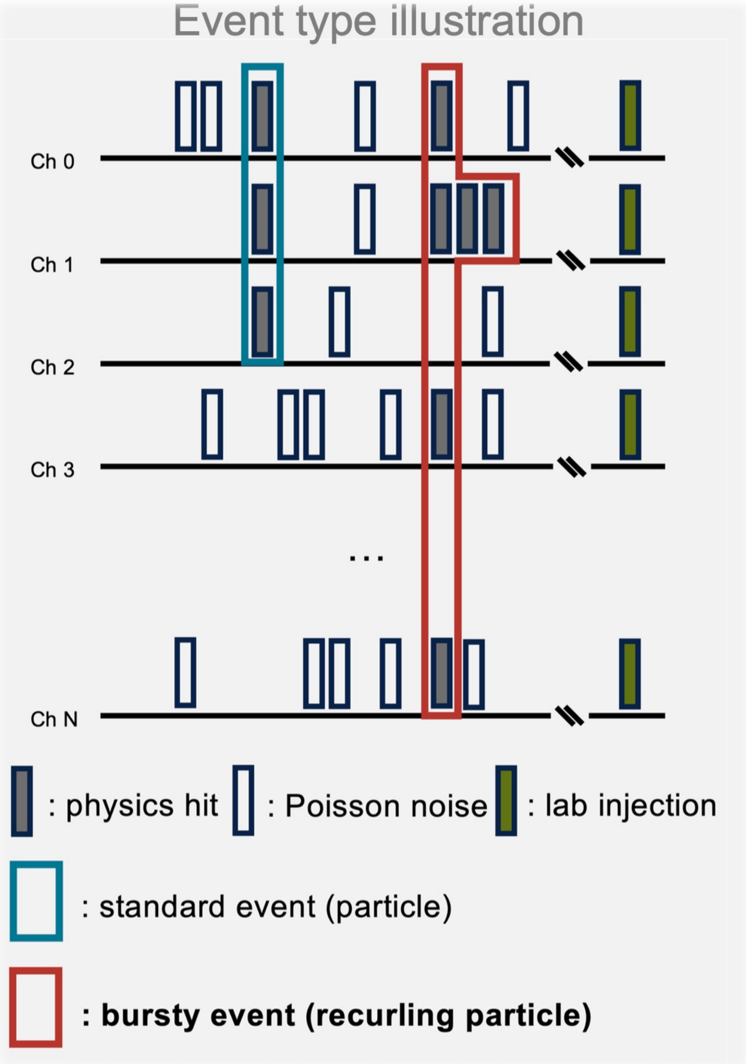
Physics Events Create Short Bursts

- ✅ Average link bandwidth can still be sufficient
- ❗ Physics-rich events concentrate hits in the same time window

Examples

- Fan-in concentrates simultaneous hits from many lanes into local bursts.
- A charged-particle track is already bursty at the source: correlated hits share nearby timestamps.



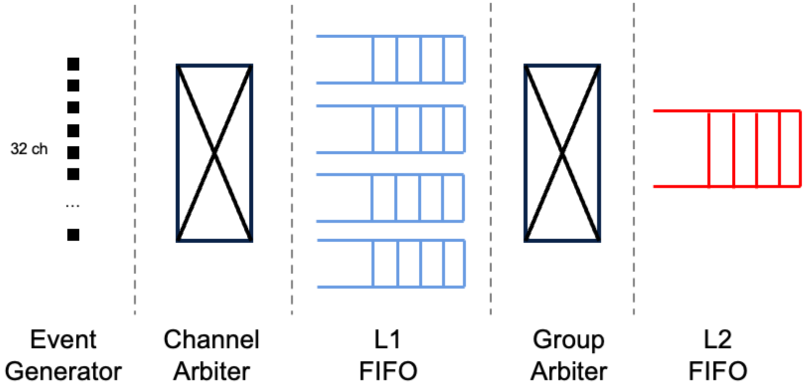


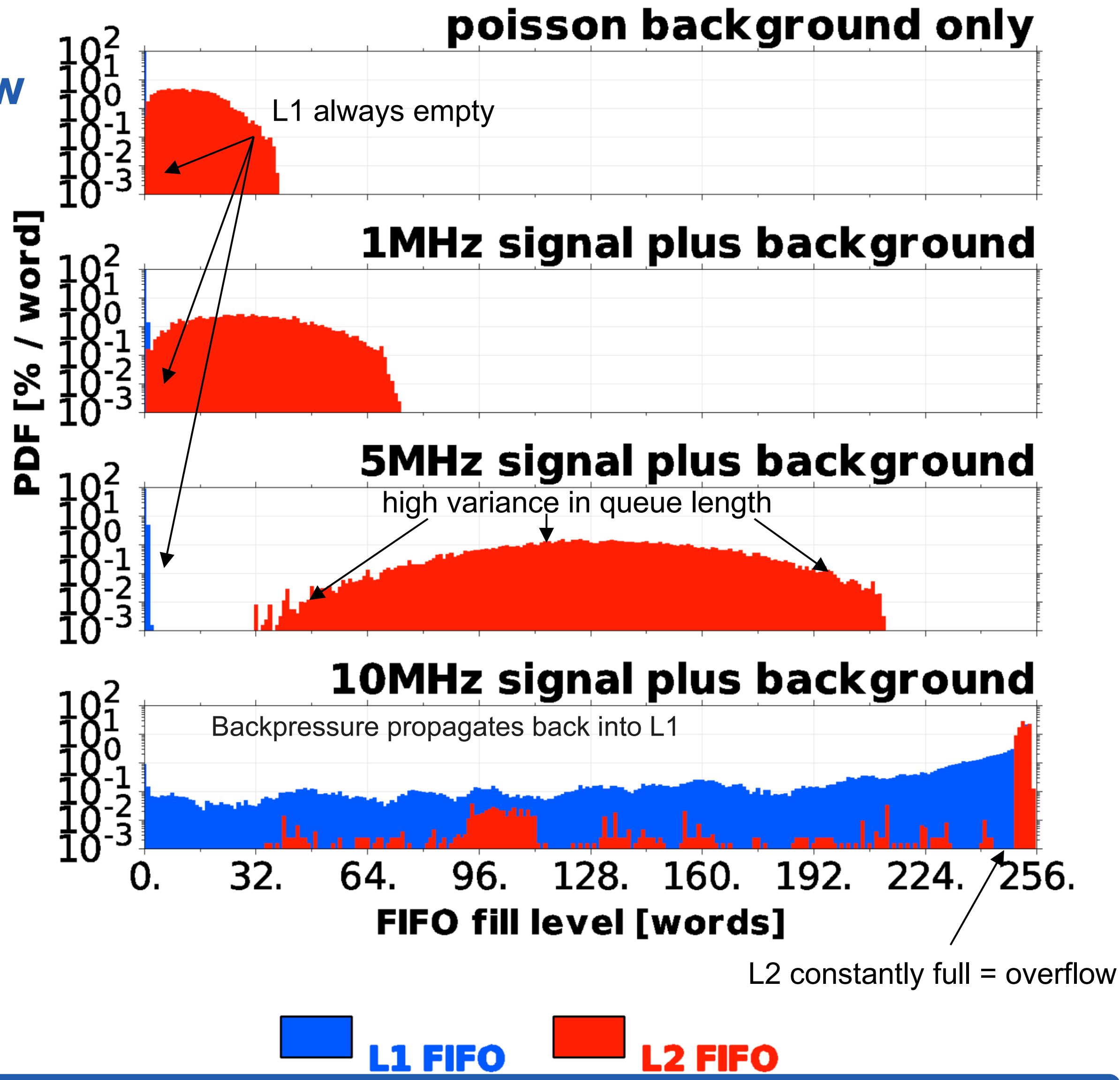
Bursts may contain high-value physics, so local loss is not an acceptable mitigation

Local FIFO Pile-Up Creates Loss and Lane Skew

- ✅ No persistent bandwidth bottleneck required
- ❗ Short physics bursts fill local FIFOs
- ❌ FIFO pile-up becomes overflow and lane skew
- ❌ Too fast for trigger feedback; needs local mitigation

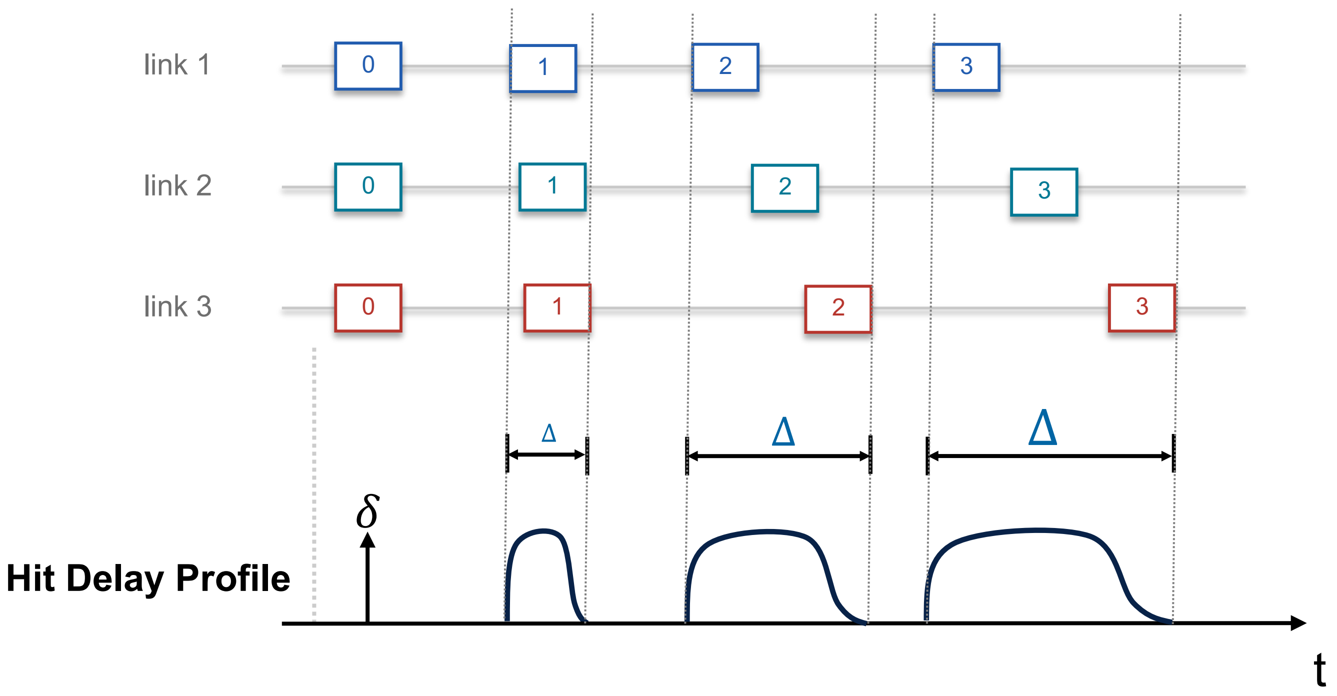
Example ASIC Architecture





FIFO fill level maps directly to residence time; unequal residence time across lanes becomes lane skew

Lane Skew Must Be Reset Locally

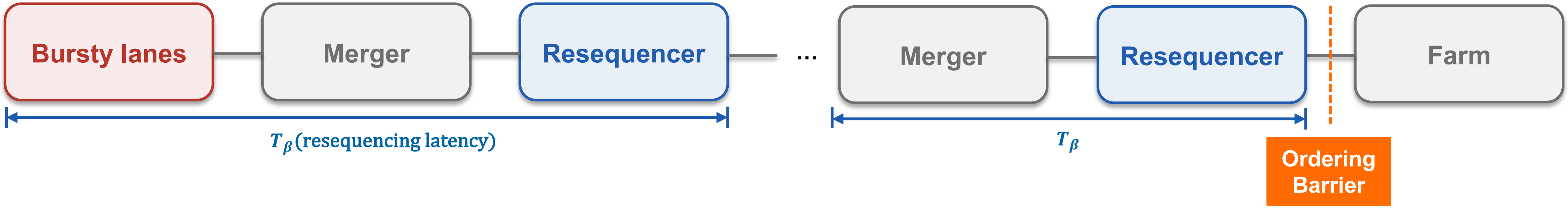


- ***Hit delay***: $d_i = t_i^{arrival} - t_i^{hit}$
- ***Lane Skew***: $\Delta = \max_i d_i - \min_i d_i$, for hits or watermarks belonging to the same timestamp/frame. It is also known as *delay spread*.
- ***Skew Bound***: $\Delta_{\max}$ is the configured worst-case bound used for resequencer sizing.

Skew accumulation across stages, if no local reset

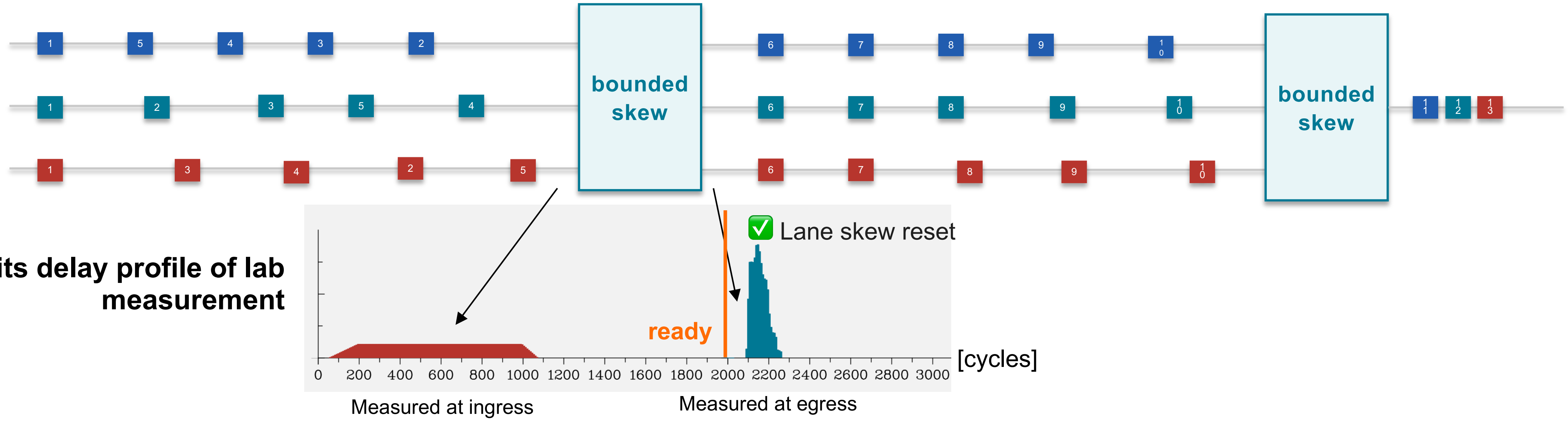
FPGA Resequencing Resets Skew Before the Farm

Sort by timestamp in FPGA at each aggregation boundary. The farm receives an ordered stream; no latency-critical farm-side ordering repair



Before resequencing

After resequencing

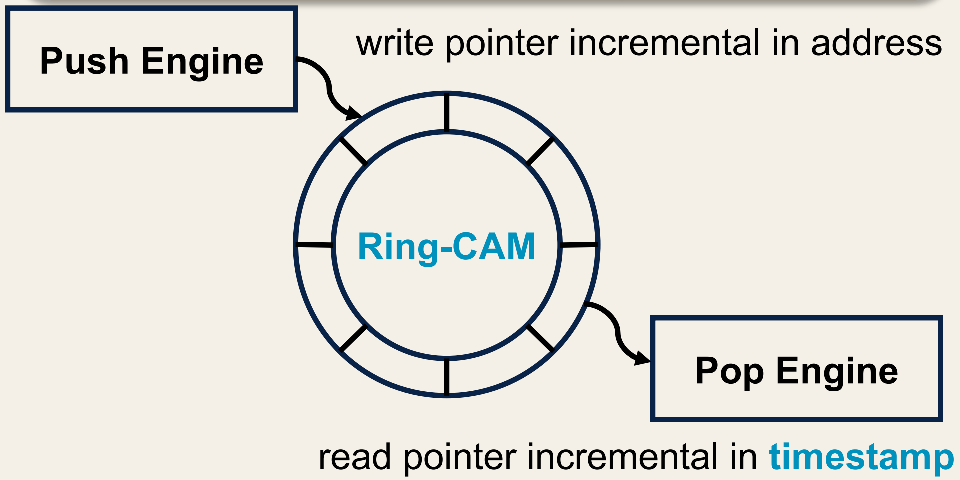


Hits delay profile of lab measurement

Ring-CAM: Write by Arrival, Read by Timestamp

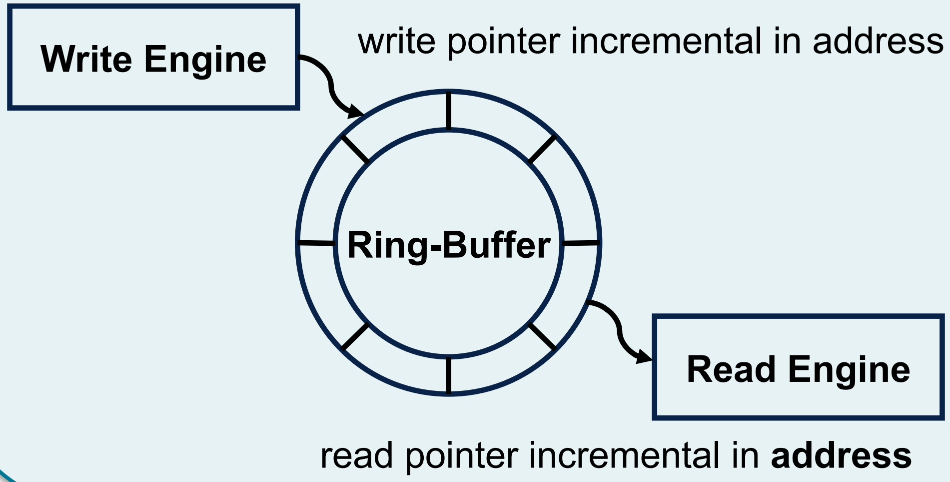
Ring-CAM (resequencer)

- Write hits as they arrive; read the earliest timestamp that is ready. Effective sorted in timestamp.
- **Ready:** timestamp τ is ready when τ is older than the current time plus delay of skew window, $\Delta_{\max} + \mathrm{t}$.
- Fixed push/pop path; depth is sized by the skew window



Compare with normal Ring-Buffer

- Normal FIFO: storage order = arrival order
- Ring-CAM: storage order = arrival order, output order = timestamp order



CAM = Content Addressable Memory
Compare to **RAM**: write op is the same, read is content or address indexed

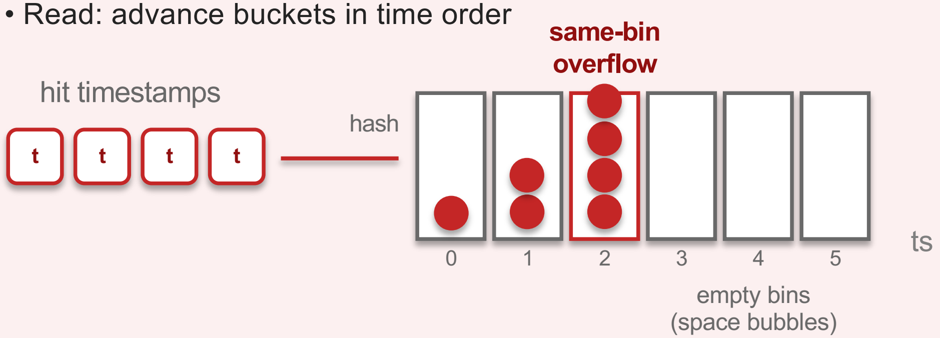
Source code available:
RTL Ring-CAM resequencer
VHDL and SystemVerilog versions

Why Ring-CAM Instead of a Calendar Queue?

Both produce timestamp-ordered output; the difference is how FPGA storage is allocated under bursts.

Calendar Queue — classic bucketed sorter

- Priority queue introduced for simulation event sets (Brown, CACM 1988)
- Write: bucket = floor(timestamp / W) mod N; W=bucket width, N=total bins
- Read: advance buckets in time order

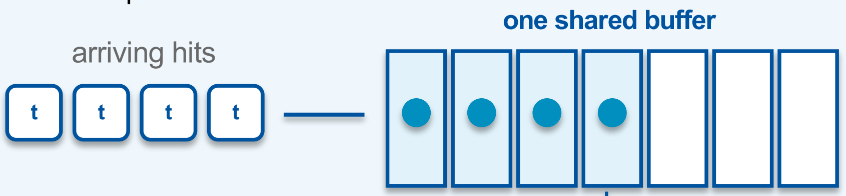


DAQ issue: a hardware calendar queue gives each timestamp bin bounded capacity. Bursty hits can overflow one bin while total memory is still available elsewhere.

Calendar Queue: Randy Brown, CACM 1988

Ring-CAM — shared storage + timestamp lookup

- Write: next free slot in ring RAM (arrival order)
- Index: CAM records timestamp → slot address
- **Read (CAM lookup):** timestamp → list/bitmap of valid slots. Multiple hits with the same timestamp are emitted before advancing to the next timestamp.



CAM lookup: timestamp → stored address

All timestamps share the full depth. Size the buffer by input rate × maximum lane skew window, plus margin.

Takeaway: Calendar Queue sorts by placing hits into time bins; Ring-CAM sorts by looking up timestamps in shared storage.

ETH zürich

Sizing Rule: Depth ≥ Rate × Skew Window

For lossless resequencing:

$$D \geq R_{in,peak} \times \Delta_{max} + \text{margin}$$

$$D \geq \sup_t [A(t) - A(t - \Delta_{max})] + \text{margin}$$

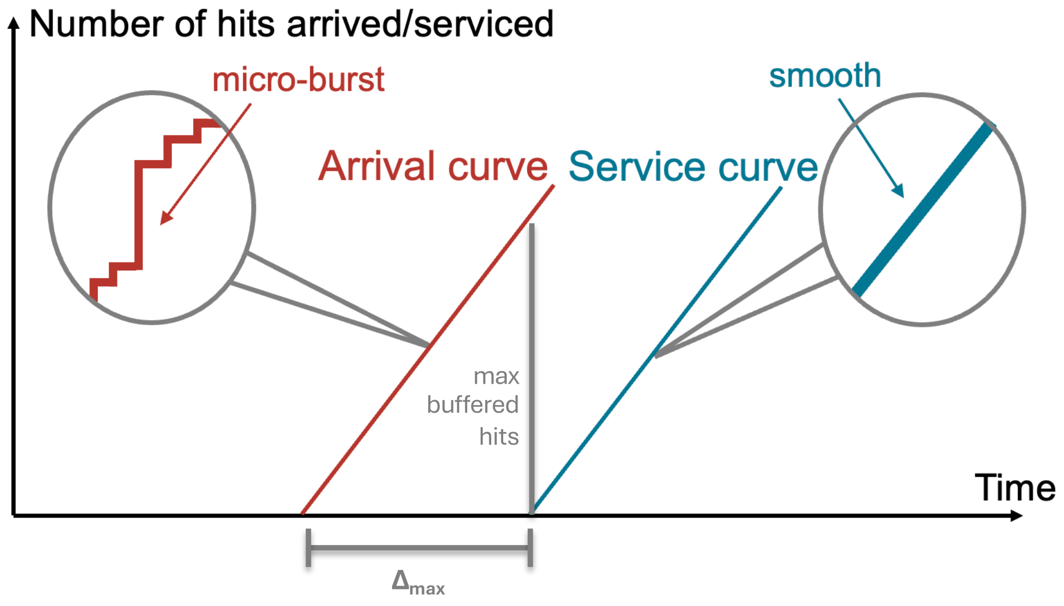
D: required Ring-CAM hit depth

$R_{in,peak}$: worst-case accepted aggregate rate over the skew window, not long-term average rate.

Δ_{max}: max lane-arrival skew for same timestamp

margin: clocking, burst granularity, etc

$A(t)$: cumulative accepted input-hit arrivals.



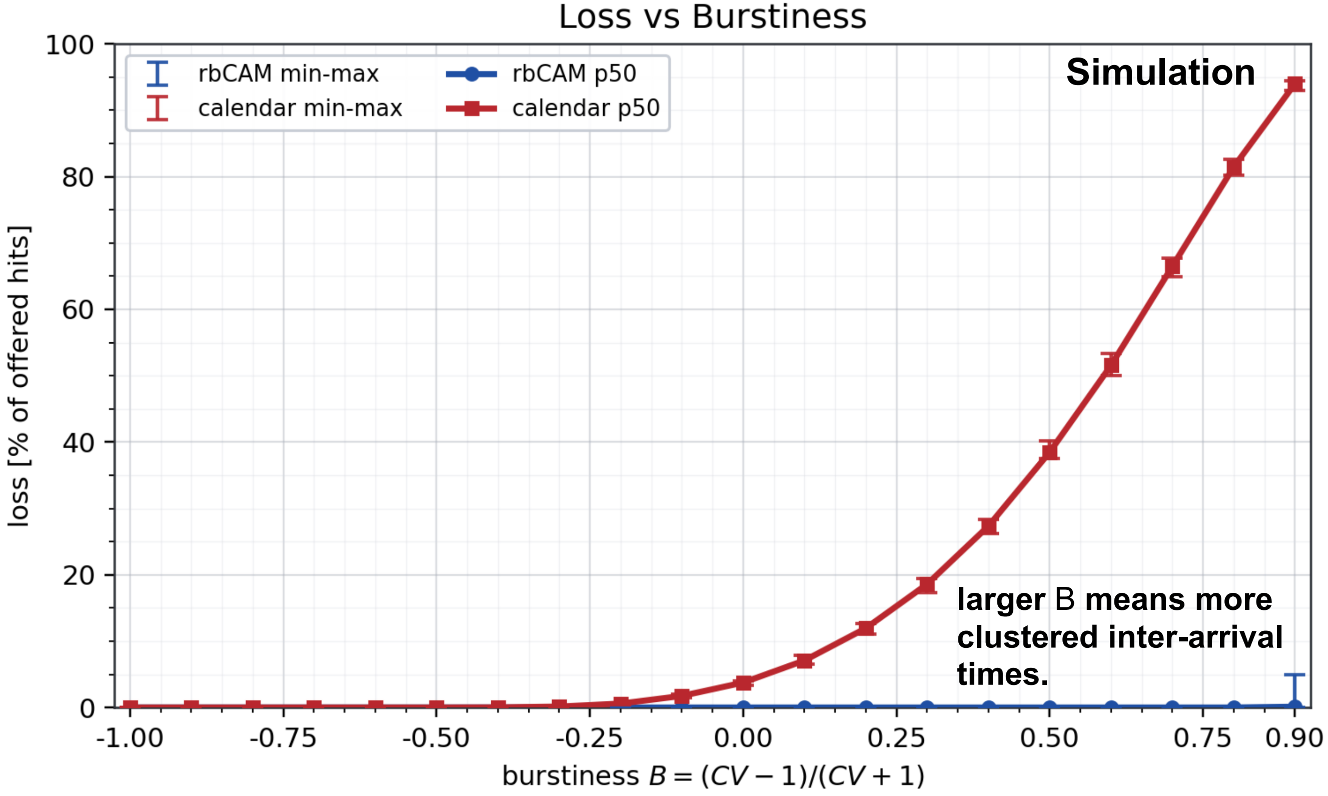
Depth is set by the aggregate number of accepted hits that can accumulate during the skew window, not by fixed per-timestamp bucket occupancy.

Assumption: *output service capacity is sufficient; the depth rule is for skew-induced backlog, not persistent overload.*

Hardware interpretation: one shared buffer absorbs the skew window.

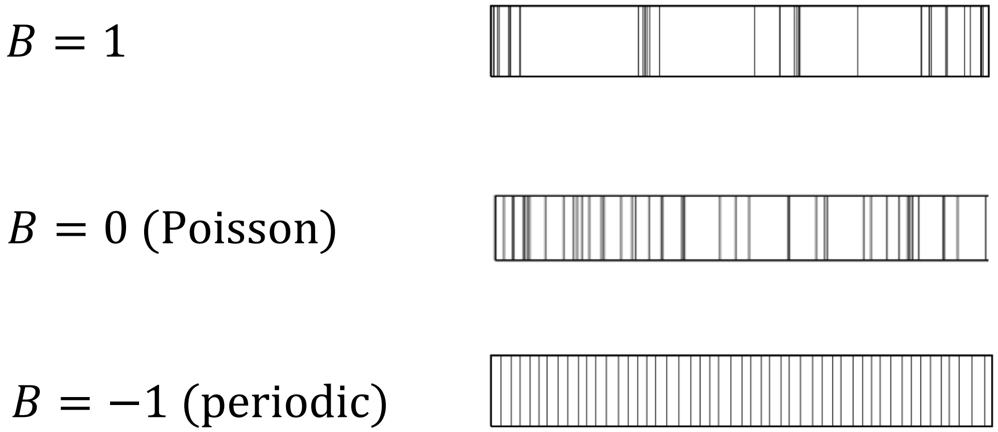
Benchmark 1: Ring-CAM Stays Lossless Under Bursts

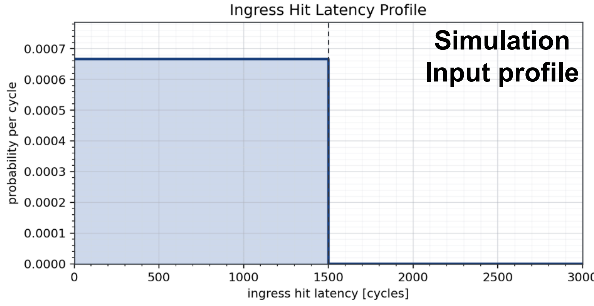
Calendar queue loss rises with burstiness; Ring-CAM remains at zero loss in this configured sweep.



SV-ingress rbCAM vs calendar 256x4; both depth=1024; push=0.125 hit/cyc (64 KHz)

Example of Burstiness (B)





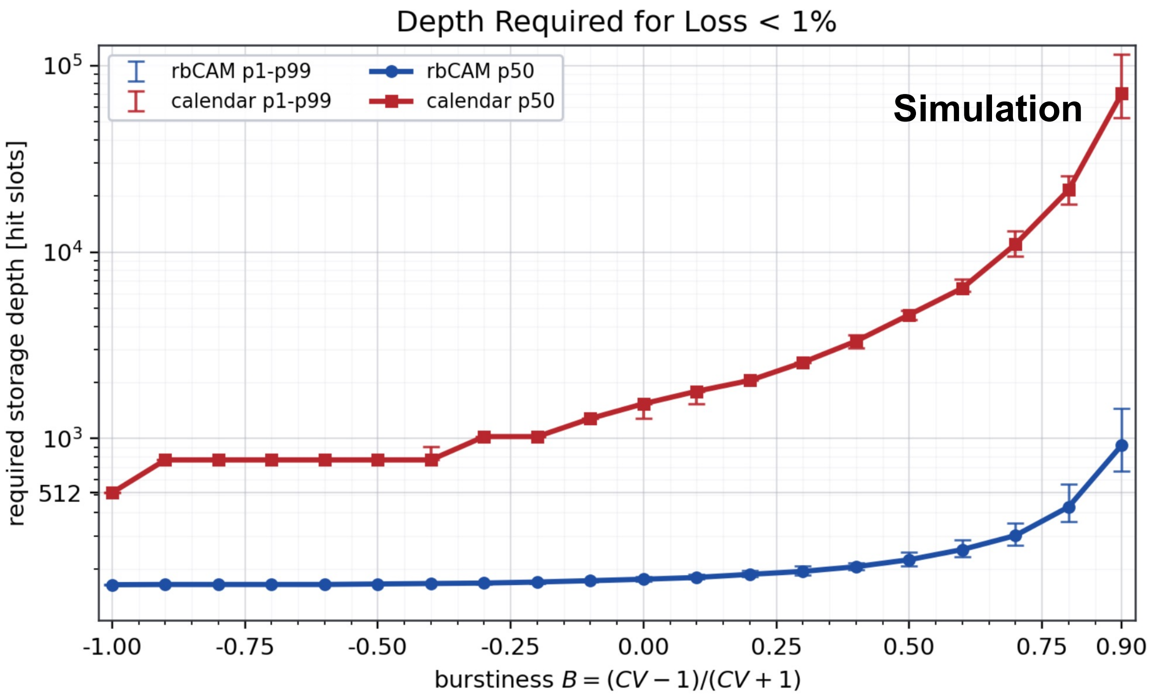
integer-uniform latency: 0..1500 cycles; plotted over 0..3000

Δ_{max} =2000 cycles

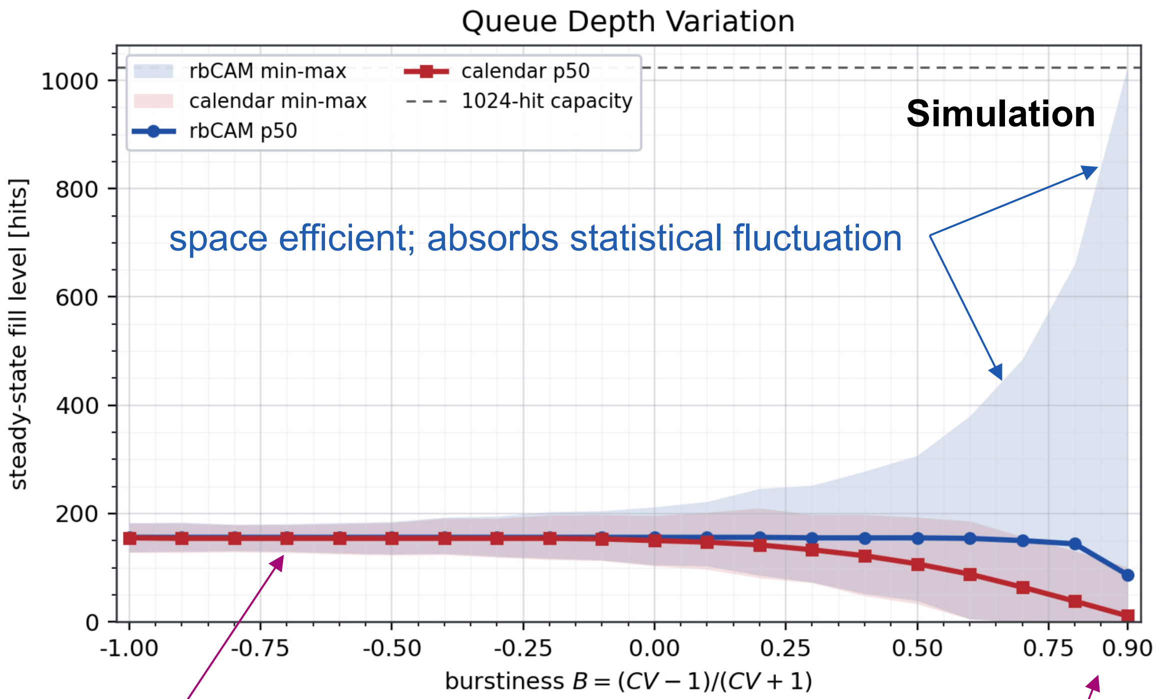
[Burstiness Formal Definition]
https://arxiv.org/abs/physics/0610233

ETH zürich

Benchmark 2: Ring-CAM Needs Less Depth Under Bursts



rbCAM: SV-ingress ideal TLM; calendar: 256 bins x cap; push=0.125 hit/cycle; bars=p1-p99



input=2 hits/bin, push=0.125 hit/cyc, mean residence=1250 cyc; bands=min-max

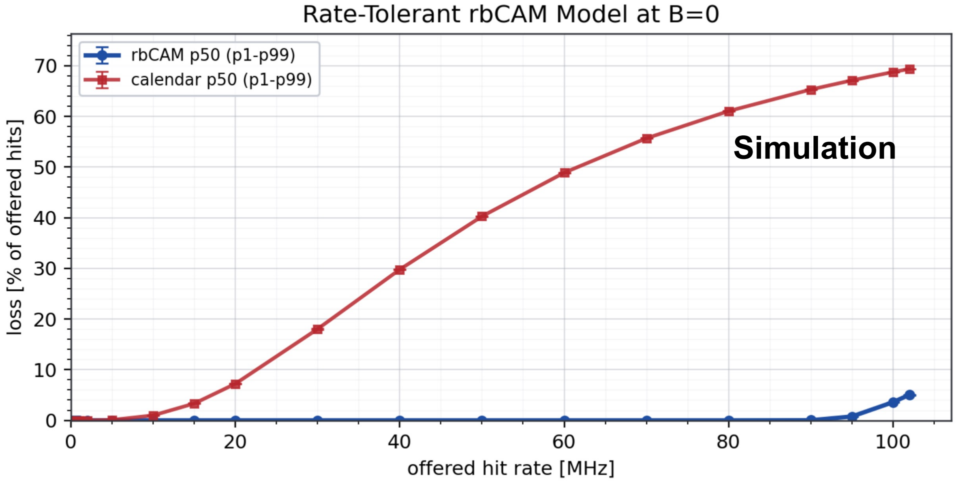
Wastes space

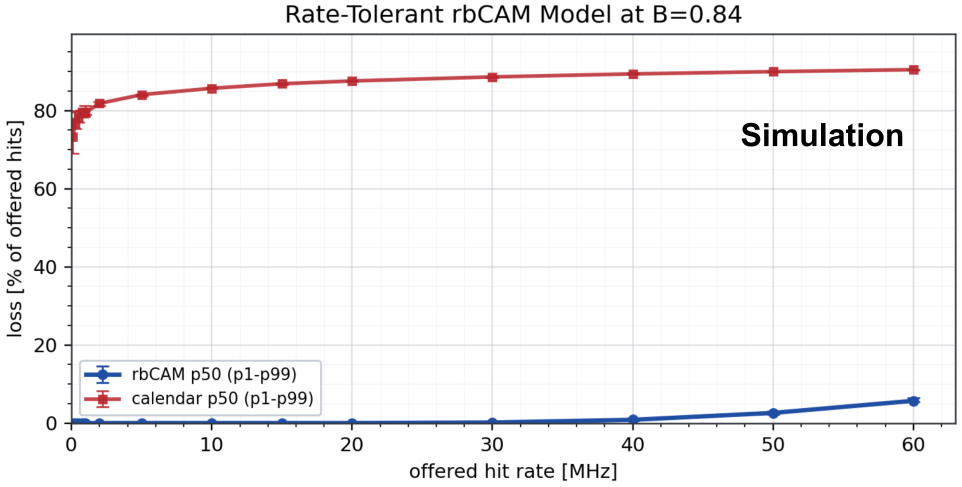
always overflows at high burstiness

Ring-CAM shares storage across timestamps; calendar queues strand memory in empty bins

Benchmark 3: Full-Rate Pressure Test

- Rate bottleneck of ring-CAM is **only** in the pop engine
- **Load balancing** with simple routing thanks to its burstiness tolerance



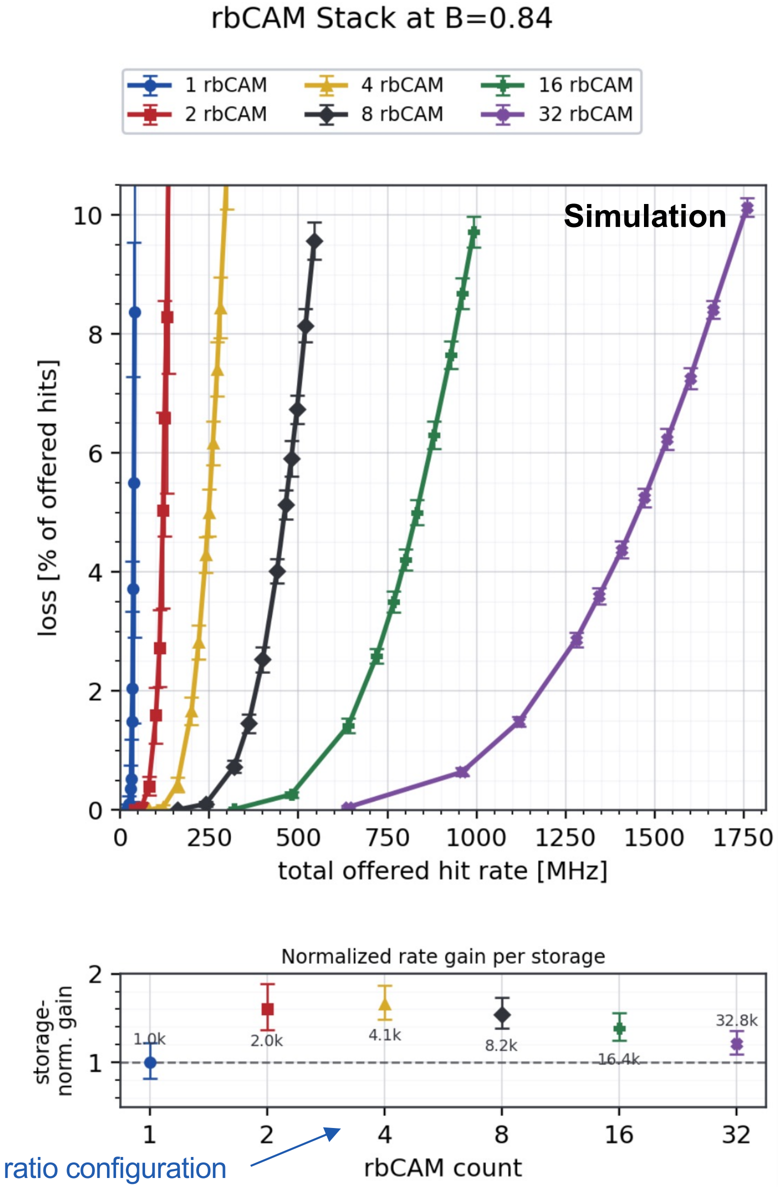


Single ring-CAM:

near-lossless until service-rate limit

Calendar queue:

bucket overflow under bursty events



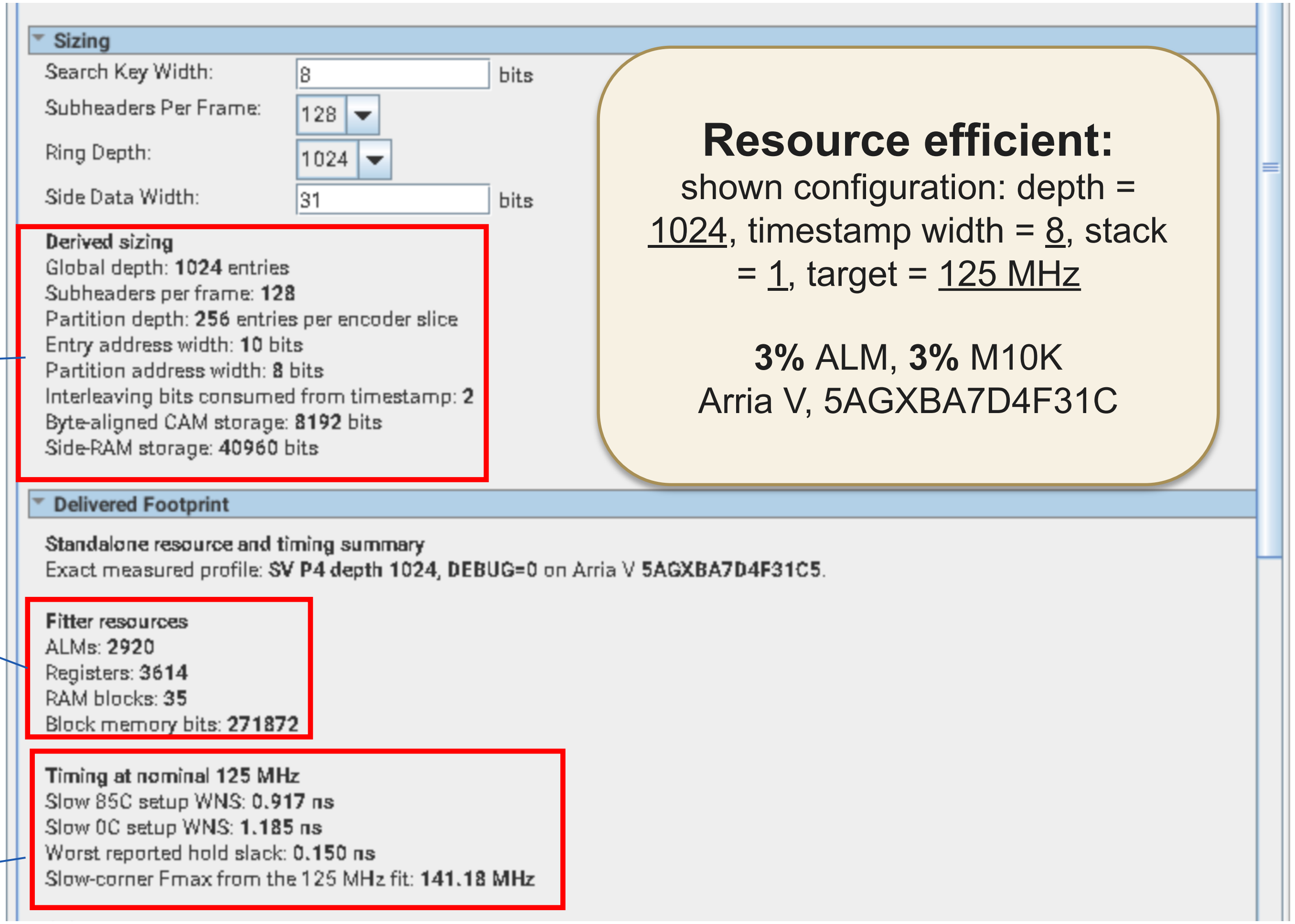
highest $\frac{\text{throughput}}{\text{resource}}$ ratio configuration

upper: 1024 slots/rbCAM; lower normalized by total storage N x depth
Gaussian mu +/- 1 sigma over 80 seeded trials; lower bars propagated at 5%

Stack ring-CAM: throughput scales with stack count with **even better** efficiency of space.

Stacking: K Ring-CAMs run in parallel; hits are routed by timestamp interleaving/mod; a final scheduler preserves timestamp order.

FPGA Implementation: Fully Synthesizable and Timing-Closed



Auto-pipeline insertion for target clock and configuration

Resource
ALM 2,920
M10K 35

Timing
Fmax 141.18 MHz
$F_{\text{sign-off}}$ 125 MHz

Resource efficient:
shown configuration: depth = 1024, timestamp width = 8, stack = 1, target = 125 MHz

3% ALM, **3%** M10K
Arria V, 5AGXBA7D4F31C

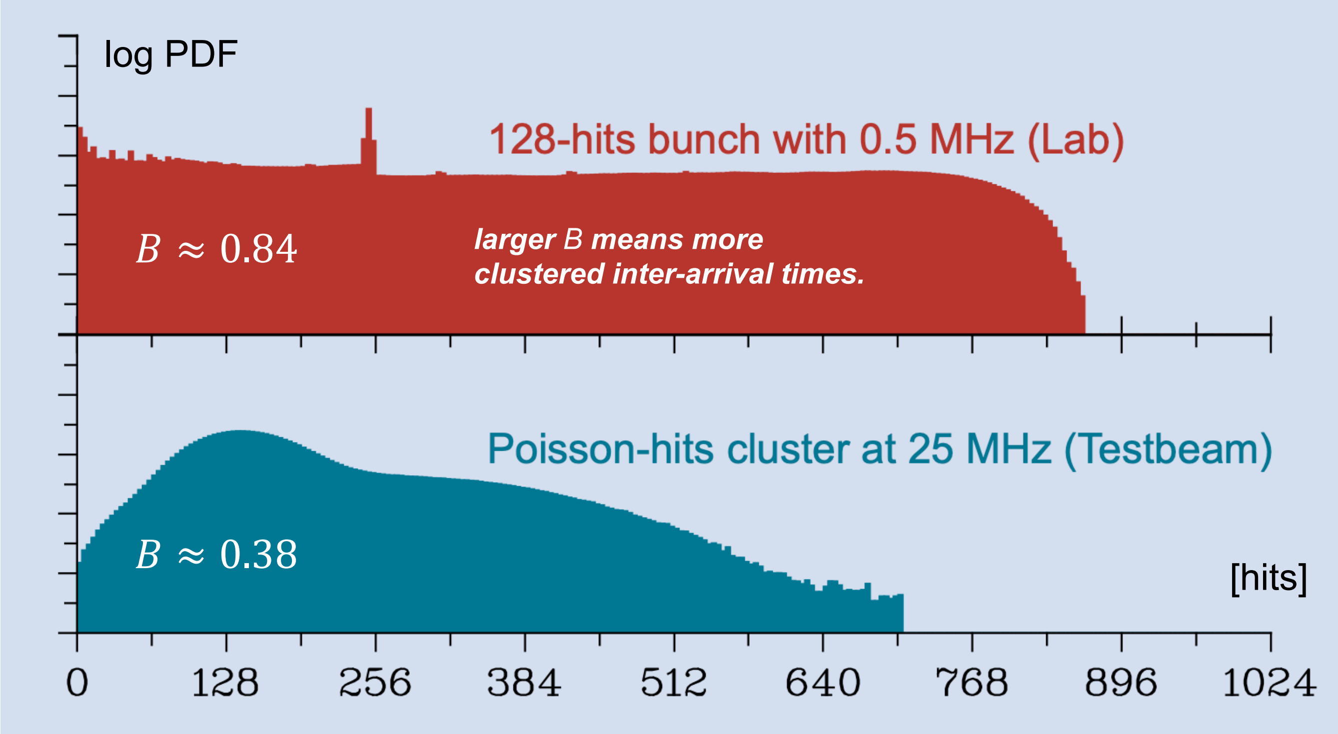
- Synthesized Ring-CAM IP
- Configuration-aware resource and timing estimate
- Easy to integrate

Source code available:
RTL Ring-CAM resequencer
VHDL and SystemVerilog versions

Hardware-in-the-Loop Testbeam: 1024-Hit Depth Remains Lossless

- Ring-CAM depth is set by the aggregate accepted hits accumulated during the skew window; **shared storage avoids fixed per-timestamp bucket overflow**.
- In PSI PiM1 / Mu3e SciFi testbeam online monitoring, fill level always stays below the configured 1024-hit depth.
- Configured service capacity is 25 MHz per stack; measured fill-level distributions remain well below the 1024-hit depth, **leaving large overflow margin**.

$D \geq R_{in,peak} \times \Delta_{max}$ + margin → lossless



Real physics data from PSI PiM1 testbeam with Mu3e SciFi detector

Verification Signoff: No Lost, Ghost, or Misordered Hits

buckets of scenarios

status	bucket	catalog_planned	promoted
✅	BASIC	158	158
⚠️	EDGE	140	140
⚠️	PROF	137	137
⚠️	ERROR	138	138

code coverage

status	metric	pct	target
✅	stmt	96.58	95.0
⚠️	branch	89.81	90.0
ℹ️	cond	68.90	-
ℹ️	expr	82.26	-
✅	fsm_state	100.00	95.0
⚠️	fsm_trans	55.56	90.0
✅	toggle	82.11	80.0

signoff nightly runs

status	run_id	kind	build	seq	txns	cross_pct
✅	CROSS-008	anchored_hybrid	default_p2_pipe4	nightly promoted X117 anchor: GOOD(2048) → ERROR(64) → FLUSH → GOOD(2048)	2048	88.8
✅	CROSS-009	anchored_hybrid	default_p2_pipe4	nightly promoted X118 anchor: GOOD(2048) → TERM → IDLE → RUN_PREPARE → RUN → GOOD(2048)	2048	88.8
✅	CROSS-010	anchored_hybrid	default_p2_pipe4	overwrite-pressure GOOD(pool=1, λ=1.0, 10k) → X119 anchor → recovery GOOD(random, 10k)	784	80.9
✅	CROSS-015	anchored_hybrid	default_p2_pipe4	nightly curated all-bucket mix: B005/B006, E002, P001-style random push-pop, X117-style error+flush recovery, plus overwrite-pressure windows, all separated by random idle gaps	2048	88.8
✅	CROSS-076	seed_sweep	default_p2_pipe4	nightly hotspot overwrite soak derived from P111, 131072 txn same-ts pressure	131072	88.8

- ✅ Functional and code coverage
- ✅ Nightly soak runs with real physics stimuli
- *Caveats*: ⚠️ = waived unreachable or configuration-infeasible coverage bins.

Signoff focus: no ghost hits, no lost hits, timestamp order preserved.

lost = input hit not emitted**;**
ghost = output hit not in input;
misordered = timestamp decreases at output

Summary: Bounded-Skew Timestamp Resequencing

Main claim: reset lane skew locally in FPGA so triggerless DAQ can deliver complete timestamp-ordered streams to the farm.

Problem

Fan-in creates micro-bursts

FIFO delay time differs by lane

The farm waits for the slowest lane (**tail latency**)

Ring-CAM mechanism

Write hits in arrival order

Index timestamp -> slot list/bitmap

Emit all slots for earliest ready timestamp

Measured evidence

Simulated burst sweeps: lossless for Ring-CAM

Resource/timing and UVM signoff complete

Physics testbeam proven lossless and promising margin

Design rule: $D \geq R_{in,peak} \times \Delta_{max}$ + margin

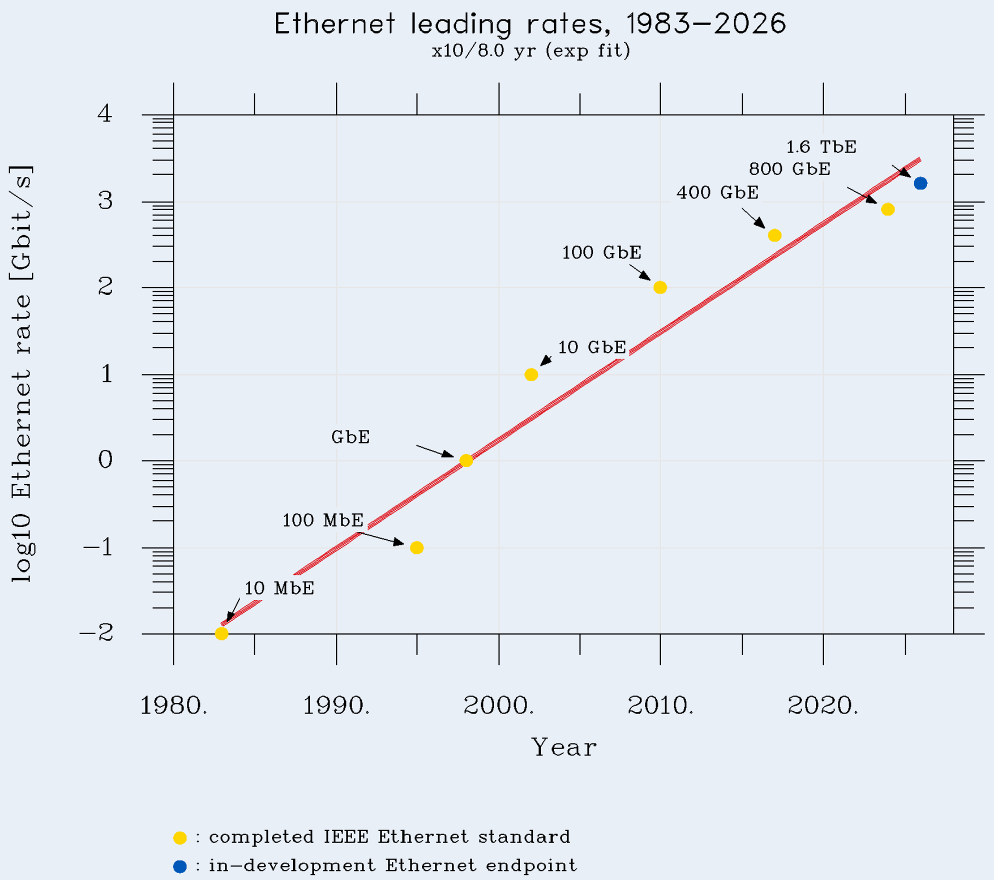
Contribution: a bounded-skew triggerless DAQ architecture plus validated Ring-CAM FPGA resequencer IP.

BACKUP

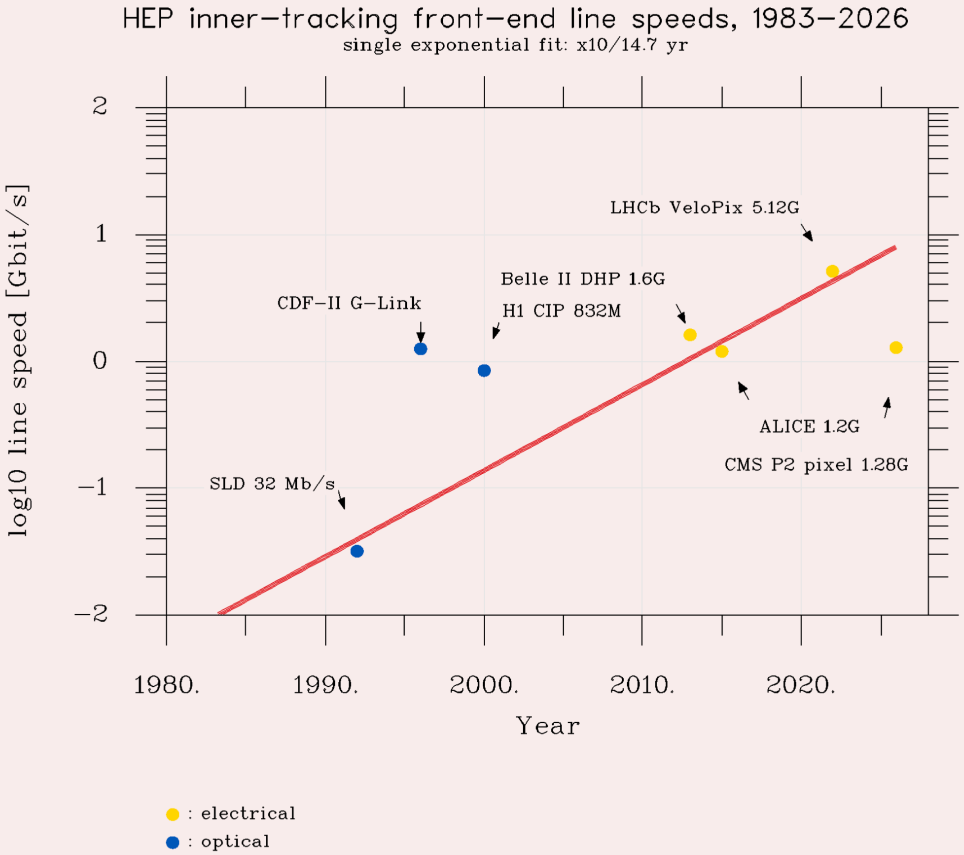
Main talk ends here. **Thank you for your attention**.
Following slides are for technical questions.

Readout Bandwidth Is Scaling Up

Industrial

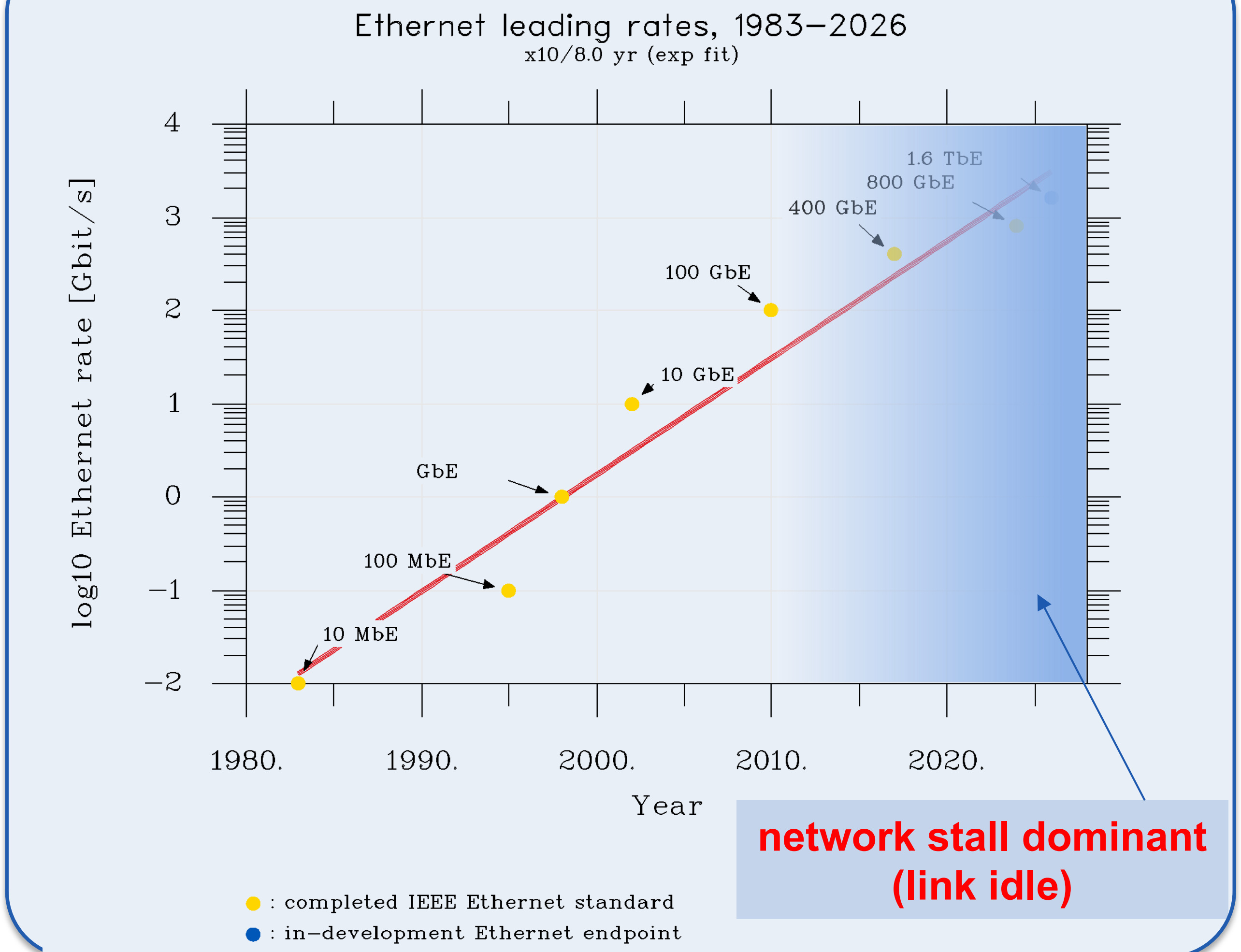


Academia



Faster links reduce the average-bandwidth problem, but do not remove timestamp synchronization and lane-skew closure

Different domain, same synchronization pattern: progress is gated by the slowest participant



External analogy: synchronization waits for the slowest transfer

The duration of each communications round is determined by the ***slowest transfer… Network failures***… become expensive in lost GPU time.

[OpenAI/Microsoft] https://cdn.openai.com/pdf/resilient-ai-supercomputer-networking-using-mrc-and-srv6.pdf

All-to-All communication can account for ***50%-60% of the total time …*** the primary communication bottleneck arises from the ***mandatory data synchronizations (network)***.

[USENIX] https://dl.acm.org/doi/abs/10.5555/3768039.3768101

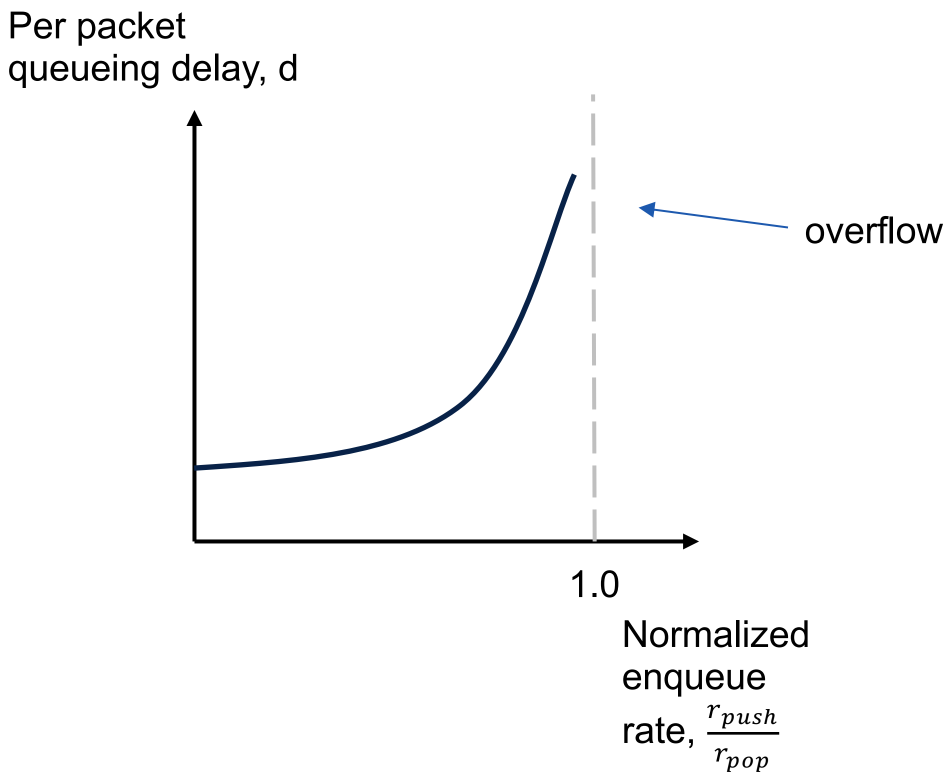
HEP analogue: fan-in waits for a complete time slice

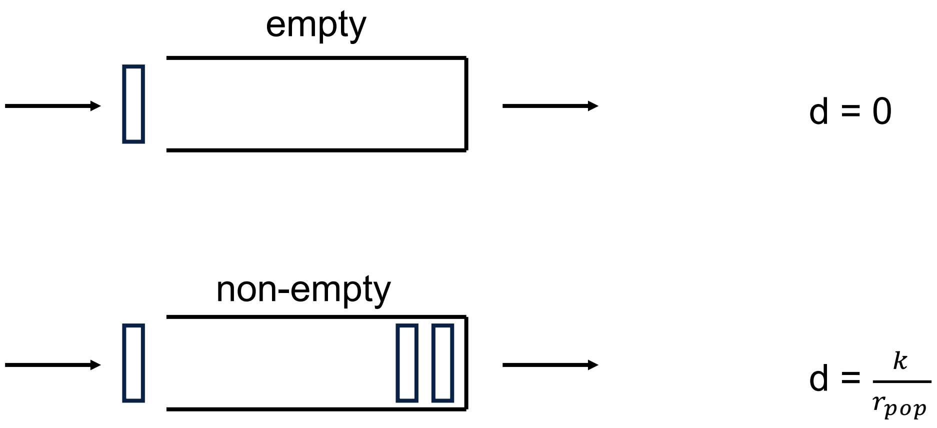
ETH zürich

Backup: FIFO Residence Time Varies with Occupancy

FIFO residence-time intuition / Work-conserving FIFO model

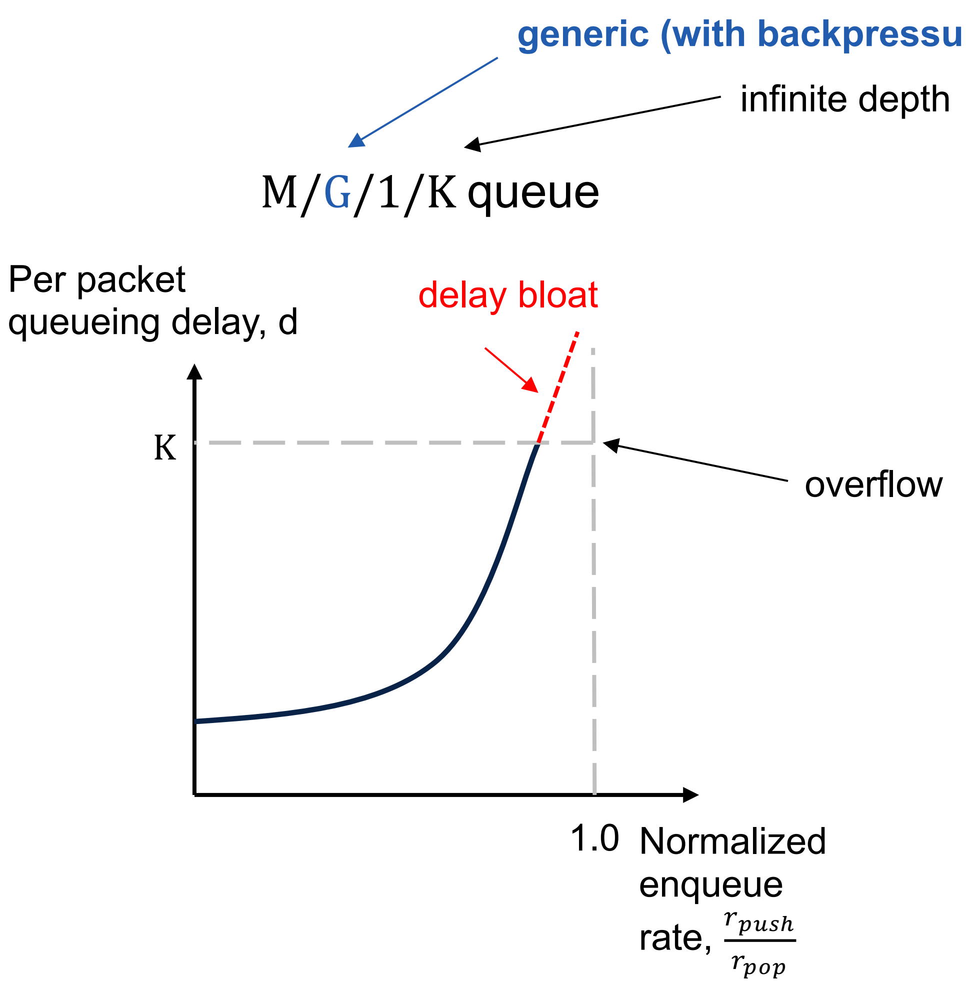
$\mathrm{M/M/1/\infty}$ queue

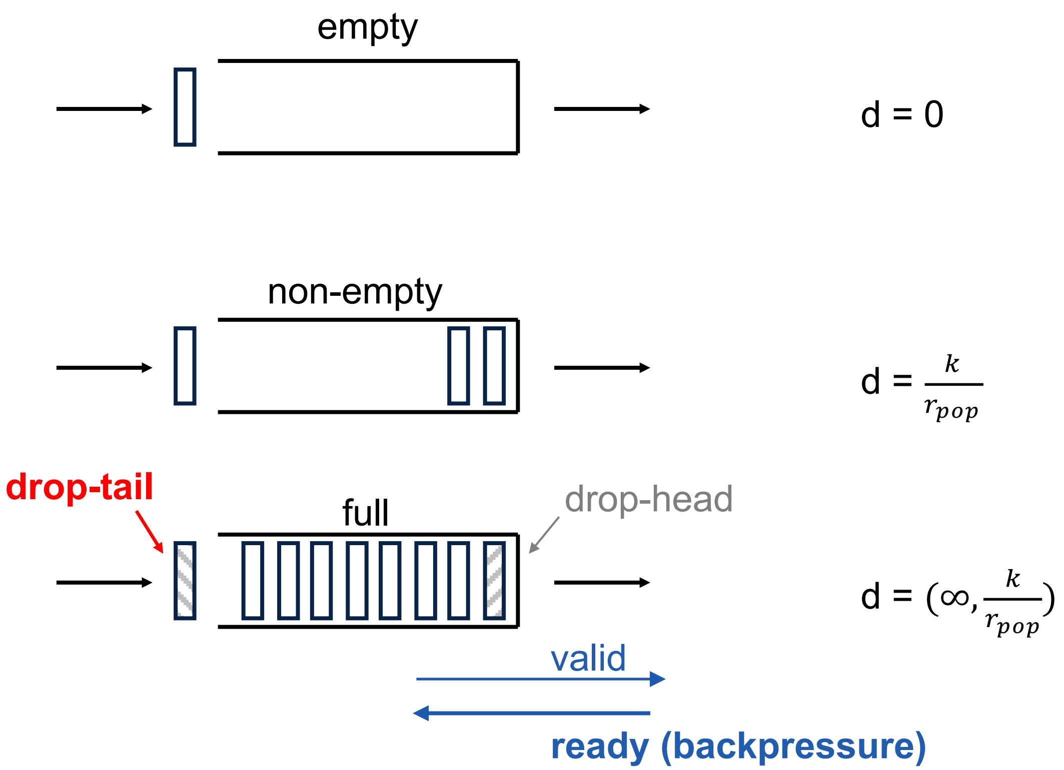




- At each merge stage, FIFOs absorb short bursts
- ⚠️ FIFO occupancy becomes variable residence time
- ⚠️ More depth prevents immediate loss but increases worst-lane delay

Backup: Backpressure Converts Burst Pressure into Delay



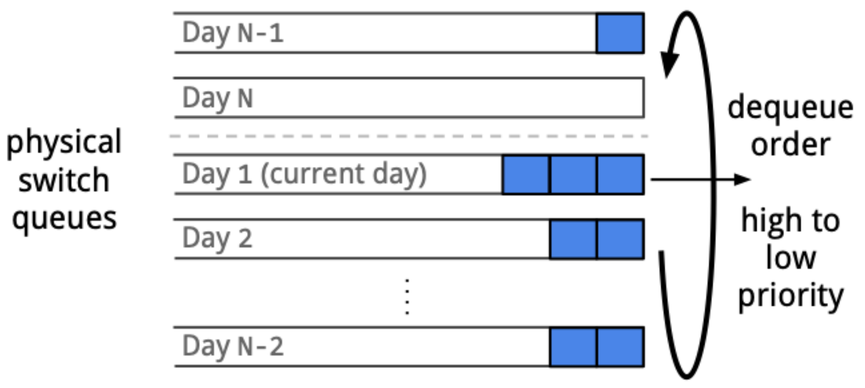


- ⚠️ Backpressure avoids immediate loss but can grow residence time
- ⚠️ Because congestion is local, lanes accumulate different delay

FIFO residence-time spread becomes lane skew

Backup: Fixed-capacity hardware bucketed sorter

Calendar Queue



Sort hits by timestamp using fixed buckets

Workflow

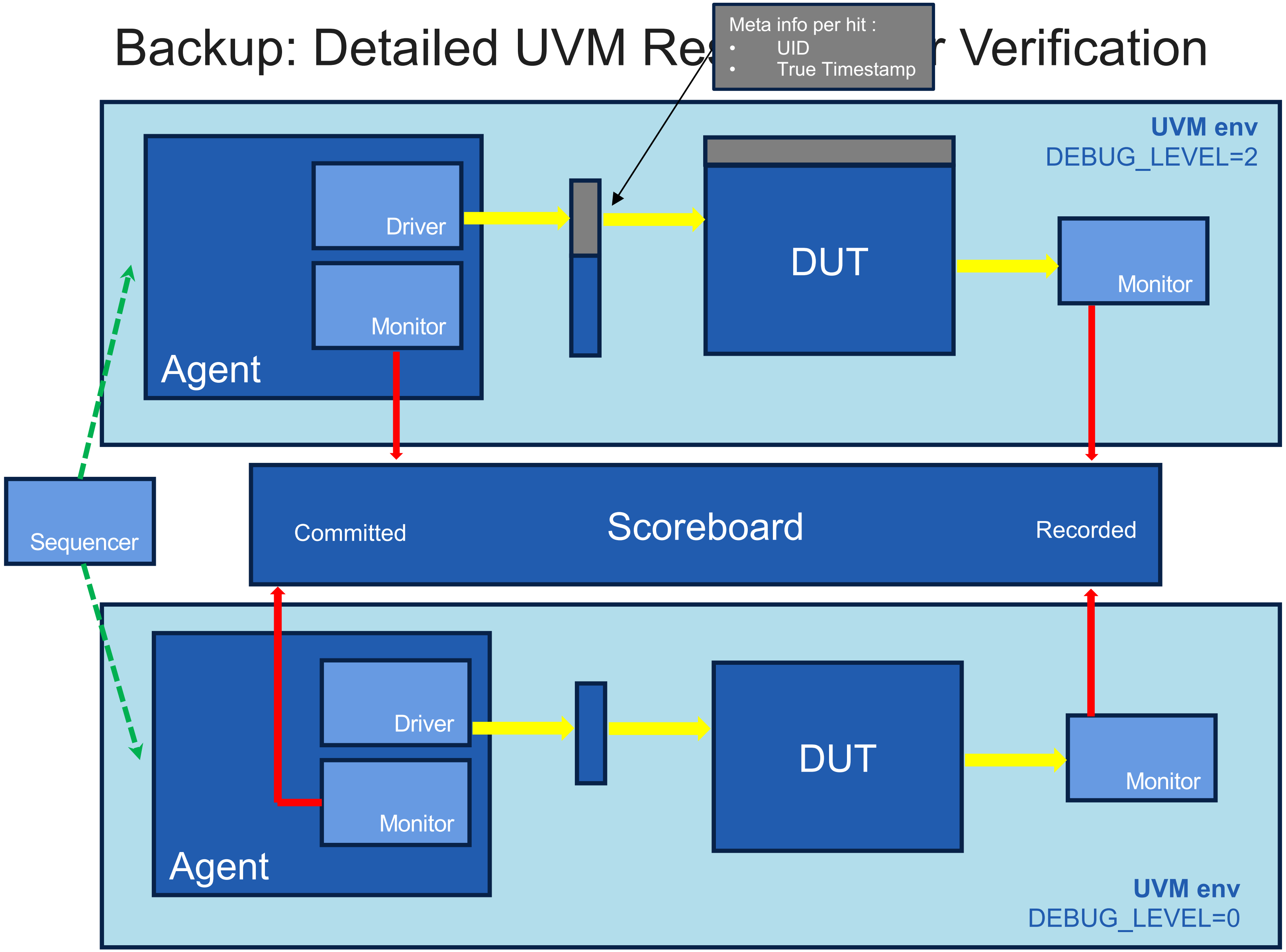
- Fixed bucket size and running admission window
- Write hits to the bucket associated with timestamp
- Read hits from the earliest bucket

Drawback

- Fixed bucket size can waste storage under bursty occupancy

[NSDI] https://www.usenix.org/system/files/nsdi20-paper-sharma.pdf
[NSDI] https://www.usenix.org/conference/nsdi25/presentation/alcoz

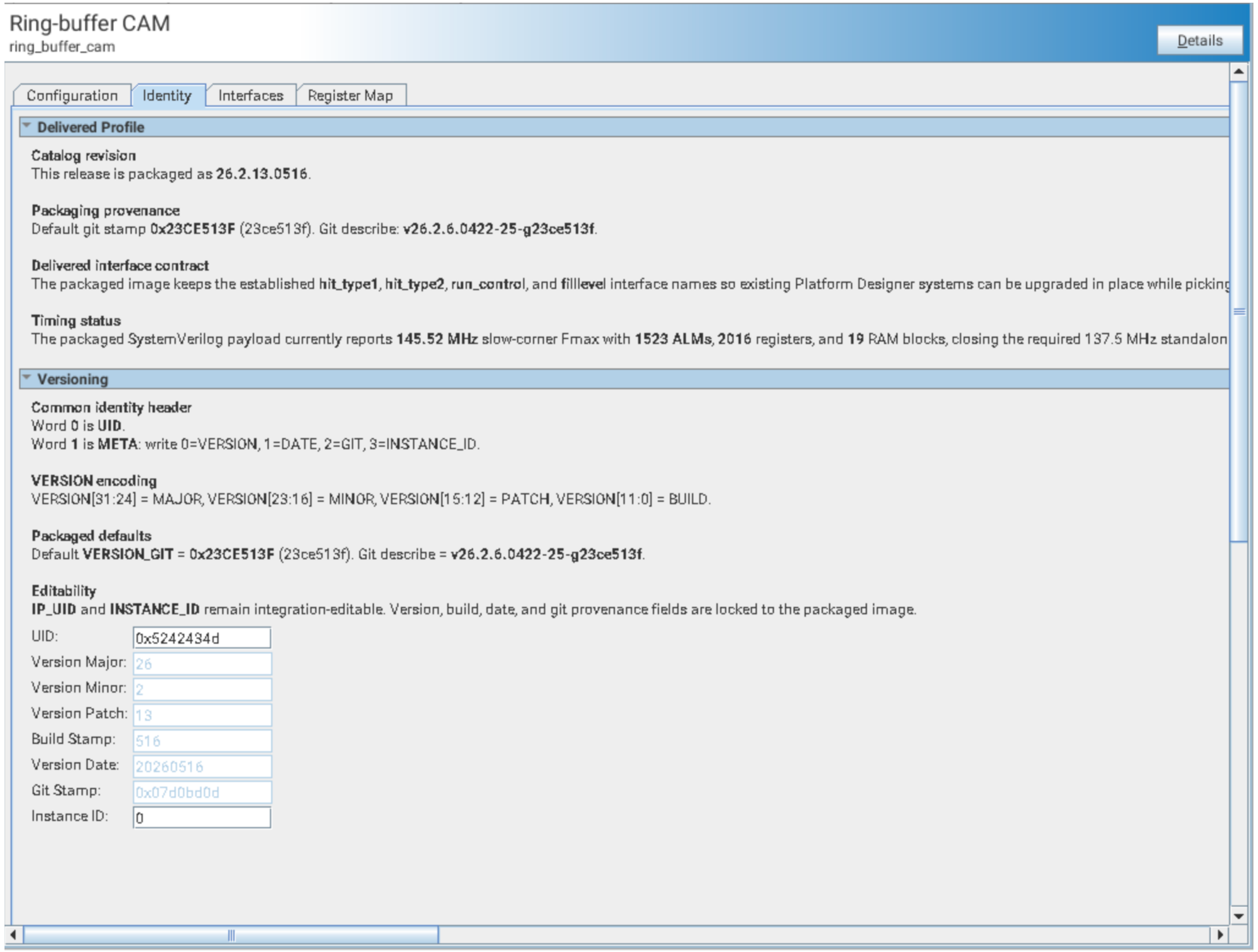
Backup: Detailed UVM Re r Verification



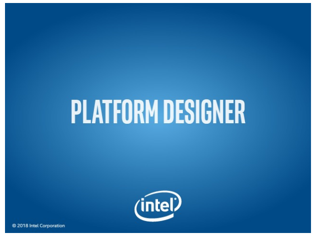
Challenge: complex gray-box DUT with memory; underflow/overflow risks.

Mitigation: dual UVM environments, per-hit UID tracking, directed regressions, randomized soak tests with physics stimuli.

Resequencer IP-core



Four version-identification fields: release, Git hash, date, build



ETH zürich

Resequencer IP-core

Running monitoring with control plane

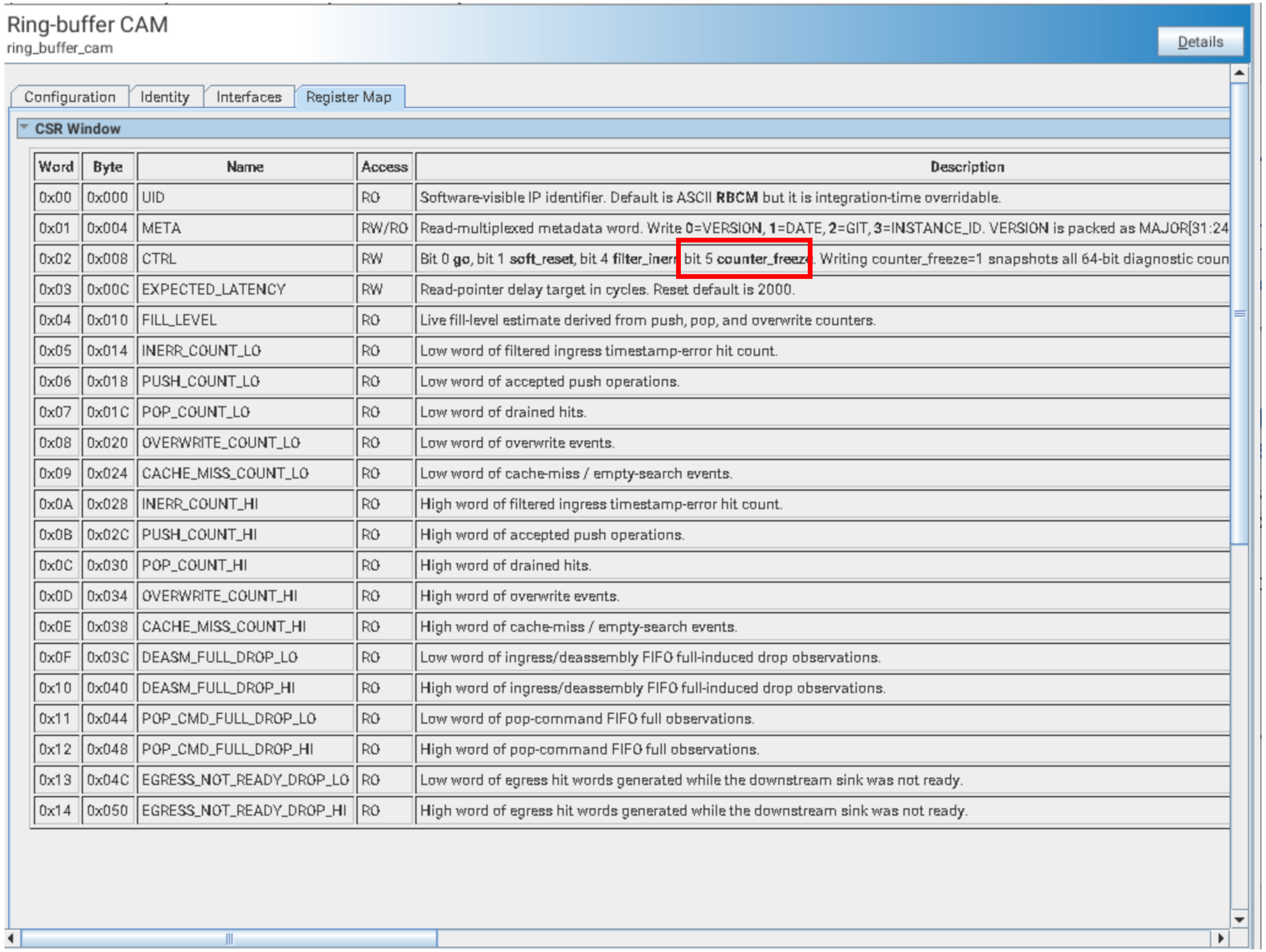
counter freeze = 1

Reading counters

Reading counters

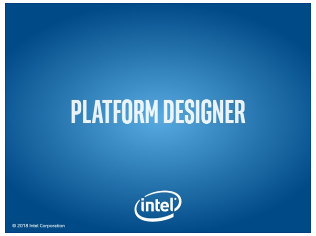
…

counter freeze = 0

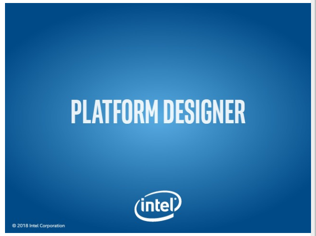


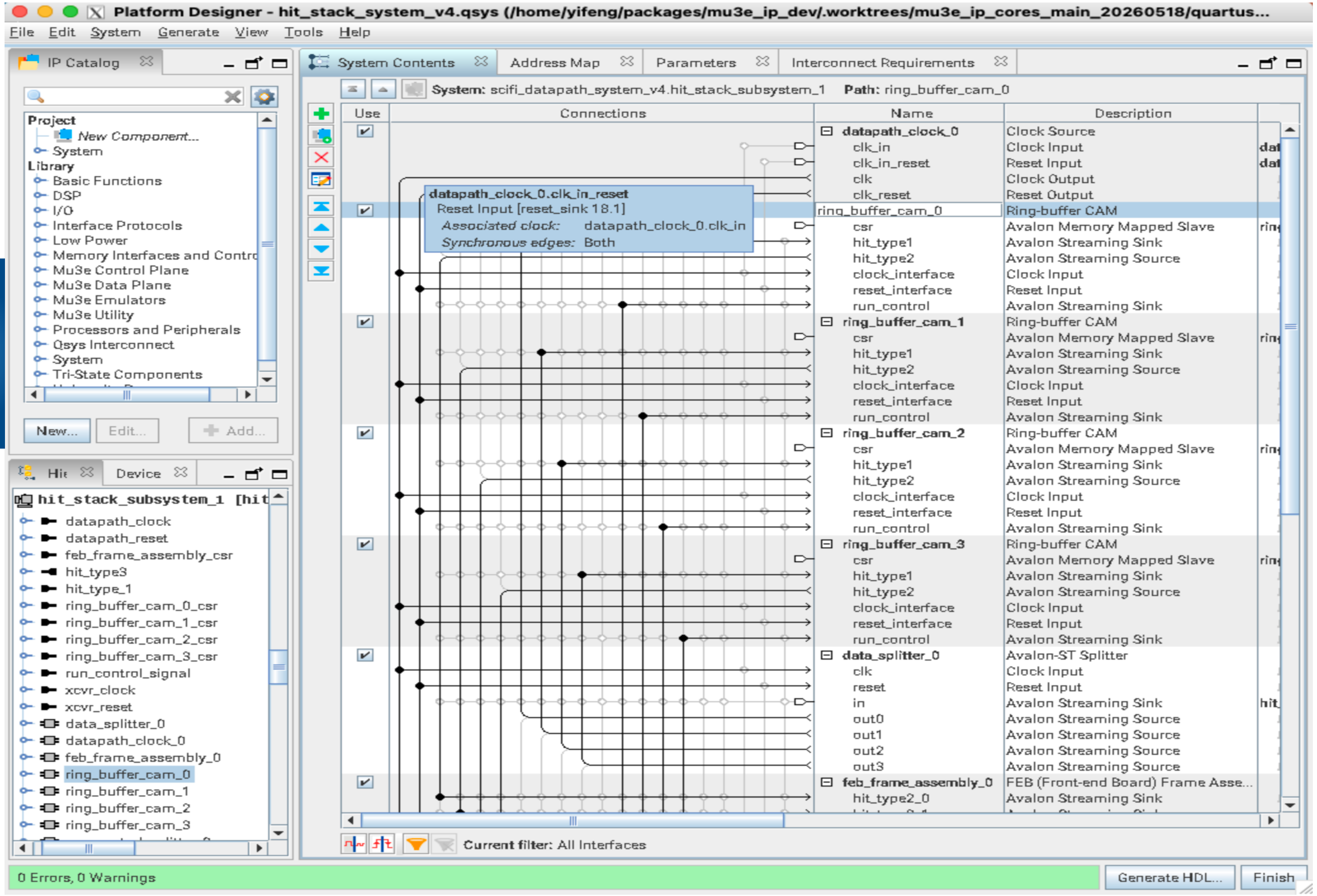
Word	Byte	Name	Access	Description
0x00	0x000	UID	RO	Software-visible IP identifier. Default is ASCII **RBCM** but it is integration-time overridable.
0x01	0x004	META	RW/RO	Read-multiplexed metadata word. Write **0**=VERSION, **1**=DATE, **2**=GIT, **3**=INSTANCE_ID. VERSION is packed as MAJOR[31:24
0x02	0x008	CTRL	RW	Bit 0 **go**, bit 1 **soft_reset**, bit 4 **filter_iner** bit 5 **counter_freeze**. Writing counter_freeze=1 snapshots all 64-bit diagnostic coun
0x03	0x00C	EXPECTED_LATENCY	RW	Read-pointer delay target in cycles. Reset default is 2000.
0x04	0x010	FILL_LEVEL	RO	Live fill-level estimate derived from push, pop, and overwrite counters.
0x05	0x014	INERR_COUNT_LO	RO	Low word of filtered ingress timestamp-error hit count.
0x06	0x018	PUSH_COUNT_LO	RO	Low word of accepted push operations.
0x07	0x01C	POP_COUNT_LO	RO	Low word of drained hits.
0x08	0x020	OVERWRITE_COUNT_LO	RO	Low word of overwrite events.
0x09	0x024	CACHE_MISS_COUNT_LO	RO	Low word of cache-miss / empty-search events.
0x0A	0x028	INERR_COUNT_HI	RO	High word of filtered ingress timestamp-error hit count.
0x0B	0x02C	PUSH_COUNT_HI	RO	High word of accepted push operations.
0x0C	0x030	POP_COUNT_HI	RO	High word of drained hits.
0x0D	0x034	OVERWRITE_COUNT_HI	RO	High word of overwrite events.
0x0E	0x038	CACHE_MISS_COUNT_HI	RO	High word of cache-miss / empty-search events.
0x0F	0x03C	DEASM_FULL_DROP_LO	RO	Low word of ingress/deassembly FIFO full-induced drop observations.
0x10	0x040	DEASM_FULL_DROP_HI	RO	High word of ingress/deassembly FIFO full-induced drop observations.
0x11	0x044	POP_CMD_FULL_DROP_LO	RO	Low word of pop-command FIFO full observations.
0x12	0x048	POP_CMD_FULL_DROP_HI	RO	High word of pop-command FIFO full observations.
0x13	0x04C	EGRESS_NOT_READY_DROP_LO	RO	Low word of egress hit words generated while the downstream sink was not ready.
0x14	0x050	EGRESS_NOT_READY_DROP_HI	RO	High word of egress hit words generated while the downstream sink was not ready.

- CSR (control and status register) read back of counters for observability



Resequencer System Integration

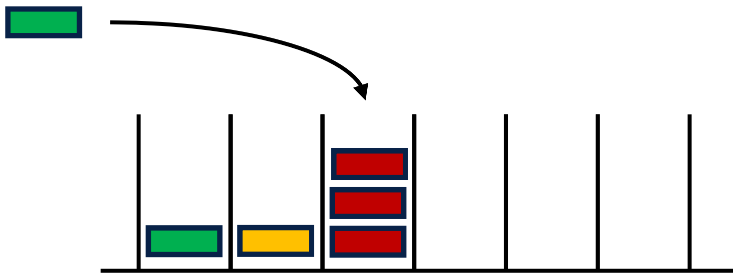




Reduces integration effort from custom RTL work to GUI-level system integration

Sorting algo

- **Calendar queue** $\Leftarrow \text{state}-\text{of}-\text{the}-\text{art}$
- Ring-CAM



Workflow:
- Write to bin addressed by TS

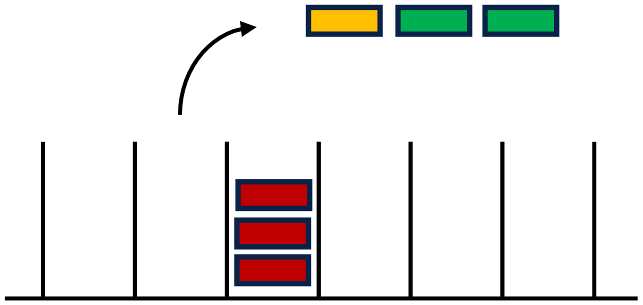
Write:
- Hash(TS) = address

ETH zürich

Sorting algo

- **Calendar queue** $\Leftarrow \text{state}-\text{of}-\text{the}-\text{art}$
- Ring-CAM

Workflow:
- Write to bin based on TS
- Read from bin sequentially

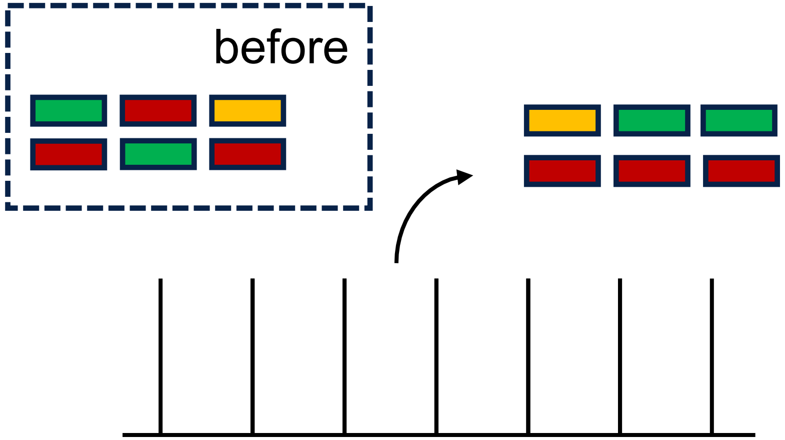


Read:
- If (empty), then address = address + 1

Sorting algo

- **Calendar queue** $\Leftarrow$ state−of−the−art
- Ring-CAM

Workflow:
- Write to bin based on TS
- Read from bin sequentially



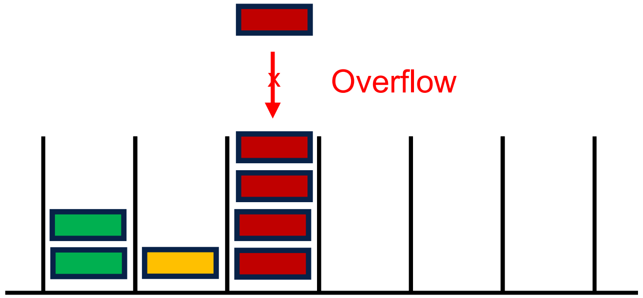
Read:
- If (empty), then address = address + 1

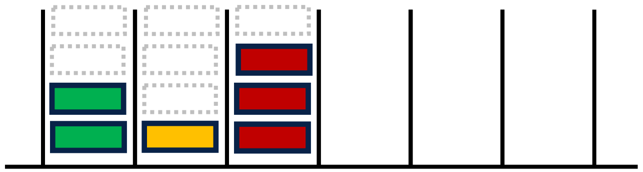
Sorting algo

- **Calendar queue** $\Leftarrow \text{state}-\text{of}-\text{the}-\text{art}$
- Ring-CAM

Problem of calendar queue:

- Fixed bin size, not O(1) space

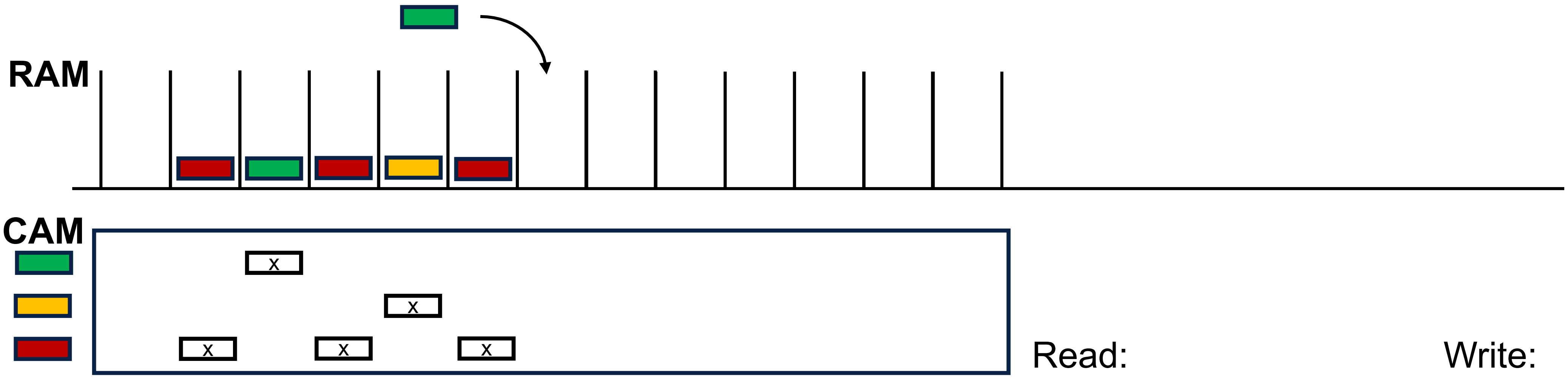




Sorting algo

- Calendar queue $\Leftarrow \text{state}-\text{of}-\text{the}-\text{art}$
- **Ring-CAM**

Workflow:
- Write sequentially



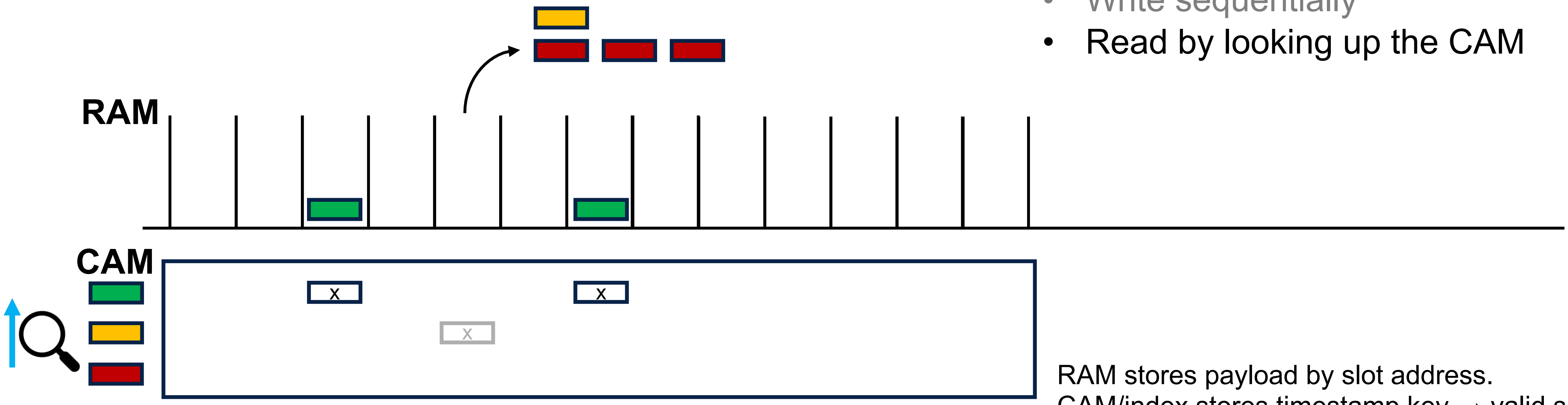
Read:
- RAM(address) = data
- CAM(data) = address

Write:
- RAM(address) = data
- CAM(address) = data

Sorting algo

- Calendar queue $\Leftarrow \text{state}-\text{of}-\text{the}-\text{art}$
- **Ring-CAM**

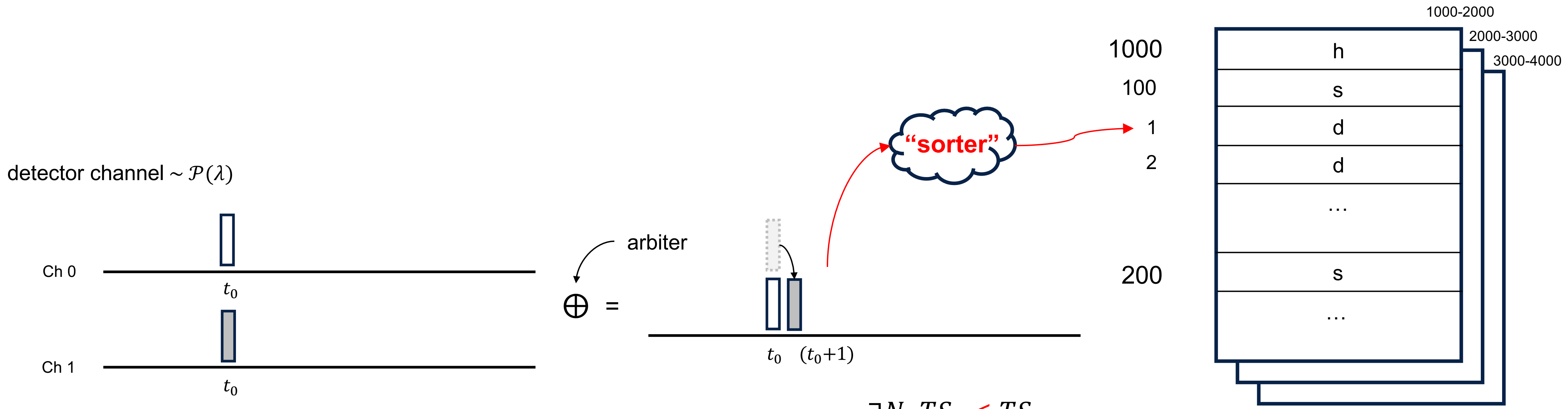
Workflow:
- Write sequentially
- Read by looking up the CAM



RAM stores payload by slot address.
CAM/index stores timestamp key → valid slot address set

Problem formulation

Q: What ruins the event **ordering**?



$\exists N, TS_N < TS_{N-1}$

non-linear modulation by arbiter

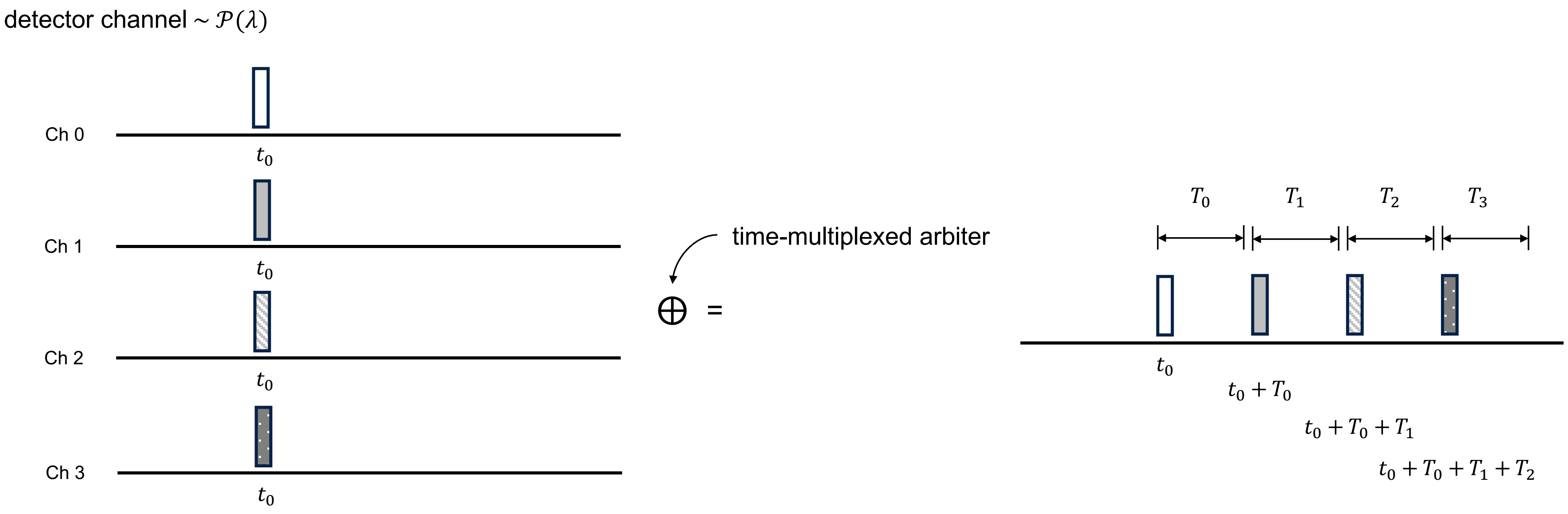
Time-Sorted Frames

Timestamp of hit := h + s + d (sum of offsets)

Problem formulation (cont.)

Q: What ruins the event **ordering**? (cont.)

Arbiter choice 1: **time multiplexed**



Example: JESD 204B/C protocol for ADC/DAC

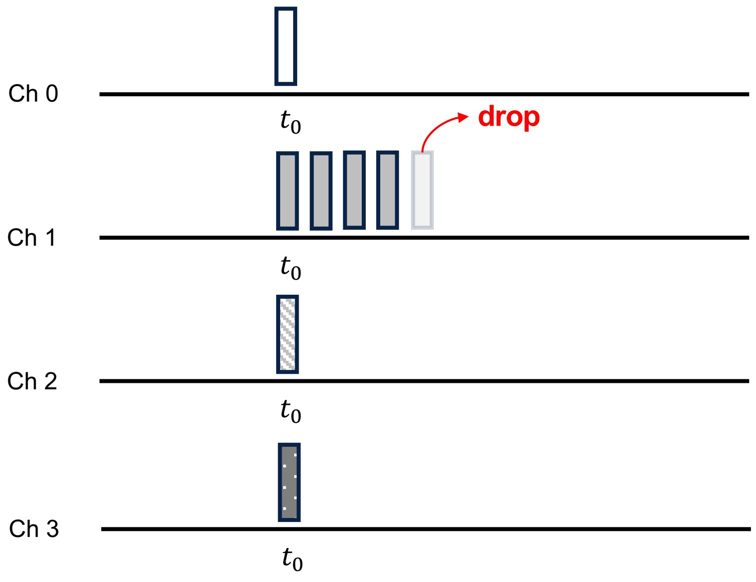
Problem formulation (cont.)

Q: What ruins the event **ordering**? (cont.)

- Q: What is the issue with time-multiplexing? – **fixed bandwidth** and deterministic latency

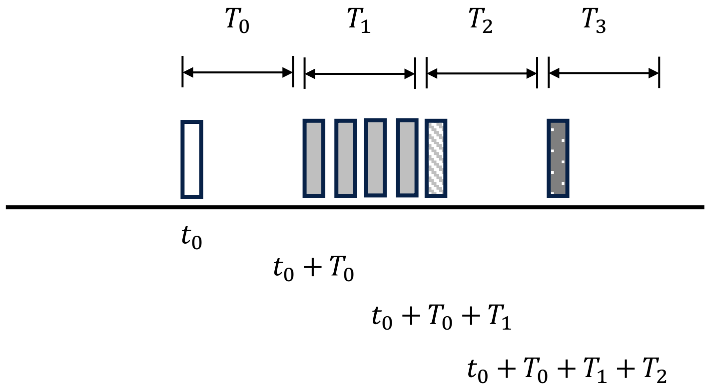
not suitable for event-driven

detector channel $\sim \mathcal{P}(\lambda)$



time-multiplexed arbiter

$\oplus$ =



Example: JESD 204B/C protocol for ADC/DAC

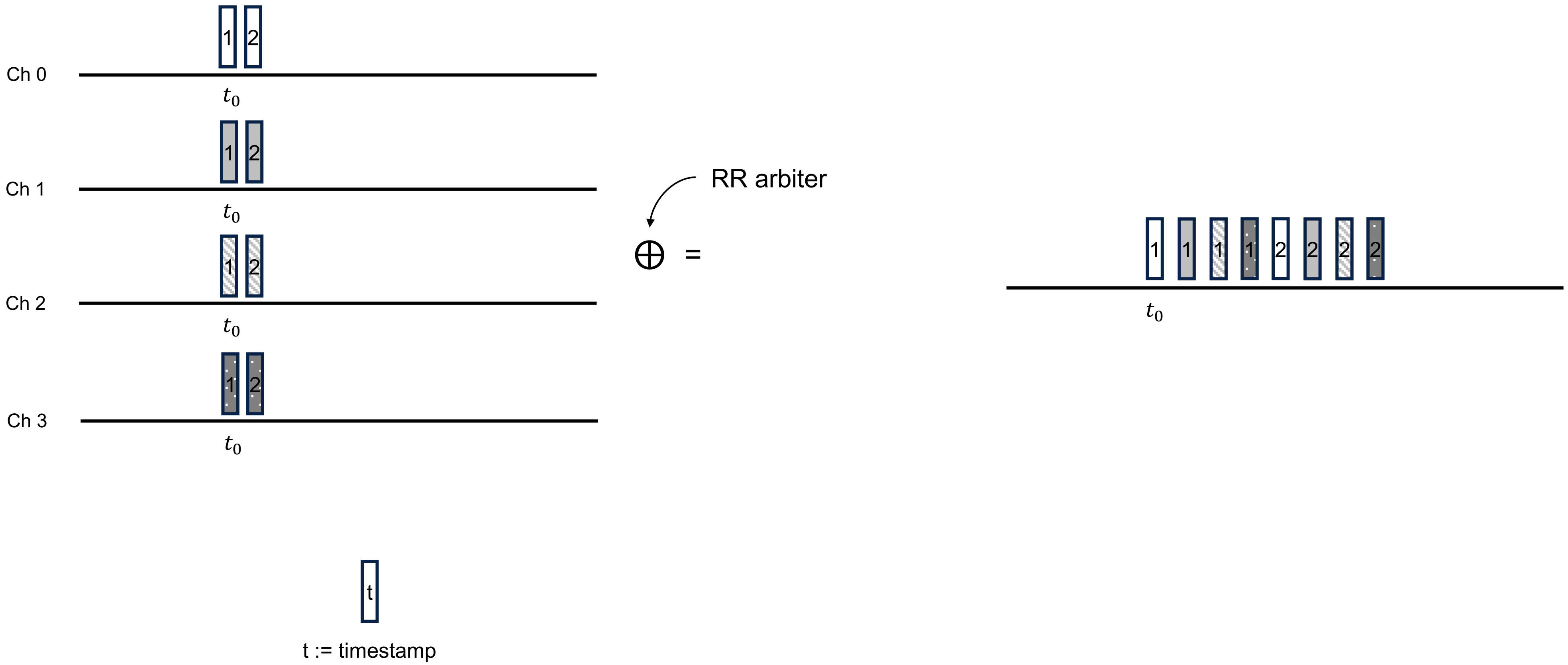
ETH zürich

Problem formulation (cont.)

Q: What ruins the event **ordering**? (cont.)

Arbiter choice 2: **rotating priority** (Round-Robin)

detector channel $\sim \mathcal{P}(\lambda)$



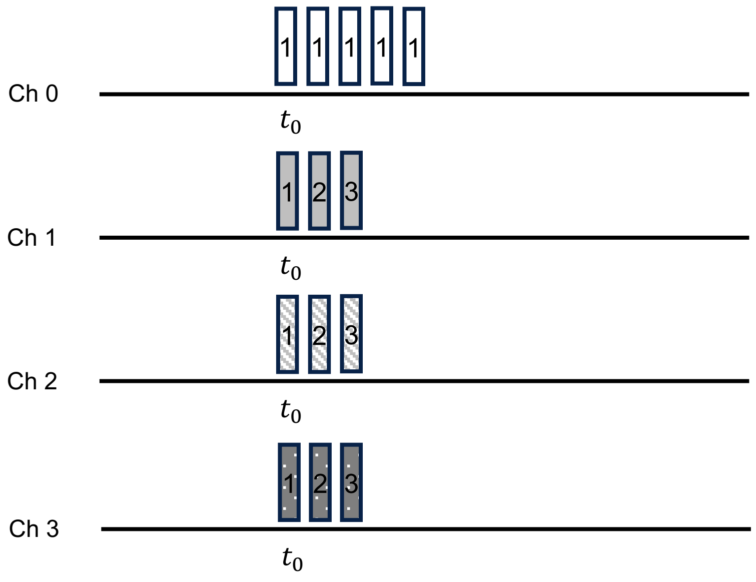
Problem formulation (cont.)

Q: What ruins the event **ordering**? (cont.)

Arbiter choice 2: **rotating priority** (Round-Robin)

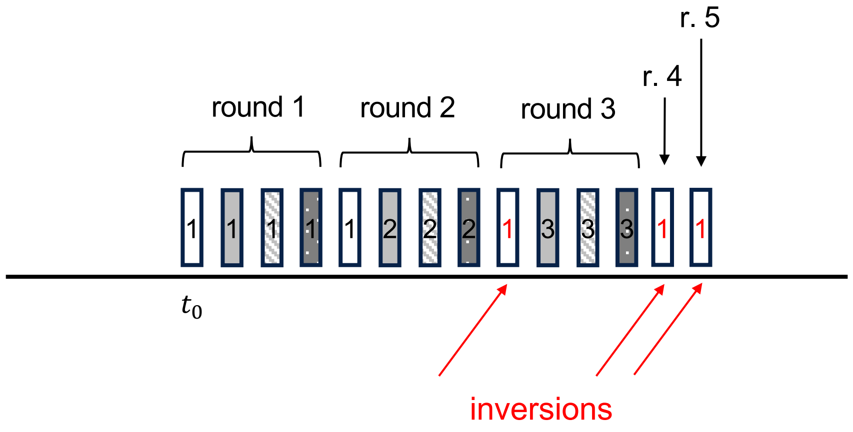
detector channel $\sim \mathcal{P}(\lambda)$

- Q: What is the issue with RR? – bandwidth "fair" sharing but **inversions**



RR arbiter

$\oplus$ =



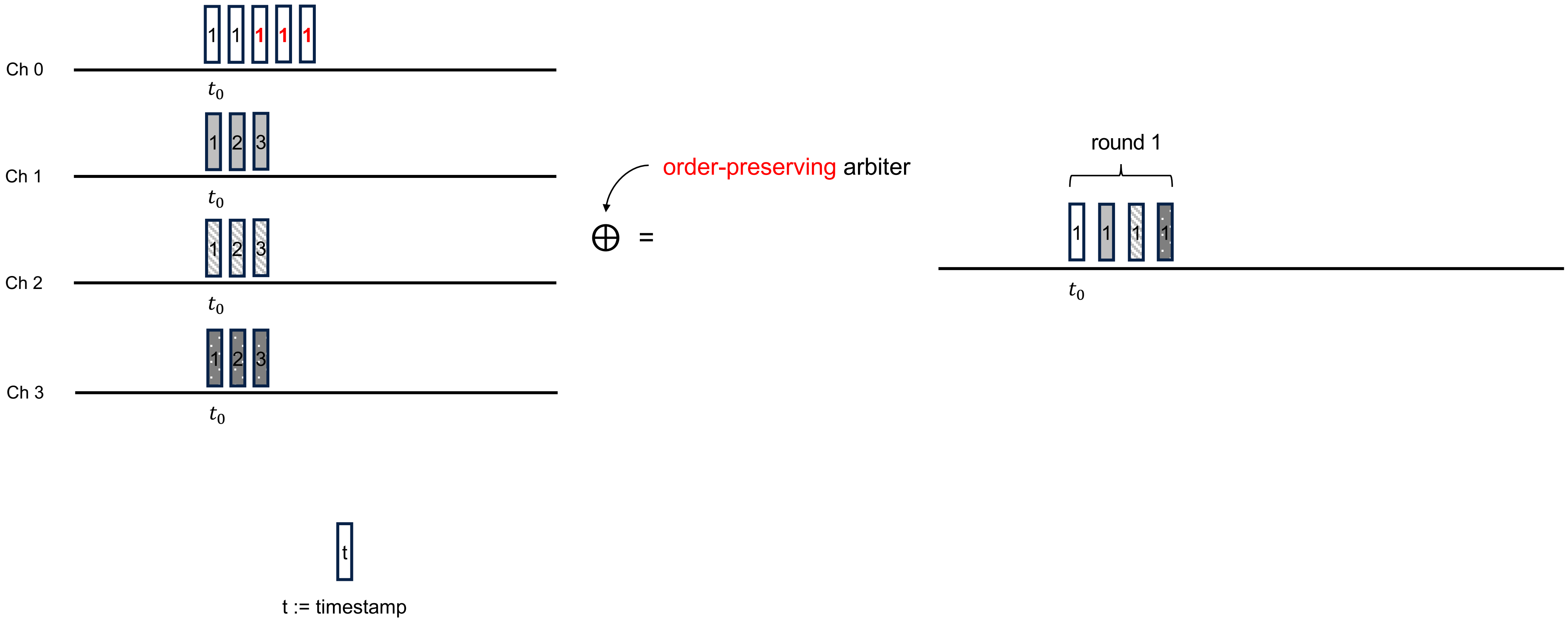
t := timestamp

Problem formulation (cont.)

Q: What ruins the event **ordering**? (cont.)

Arbiter choice 3: **order-preserving** (granting the smallest Head-of-Line)

detector channel $\sim \mathcal{P}(\lambda)$

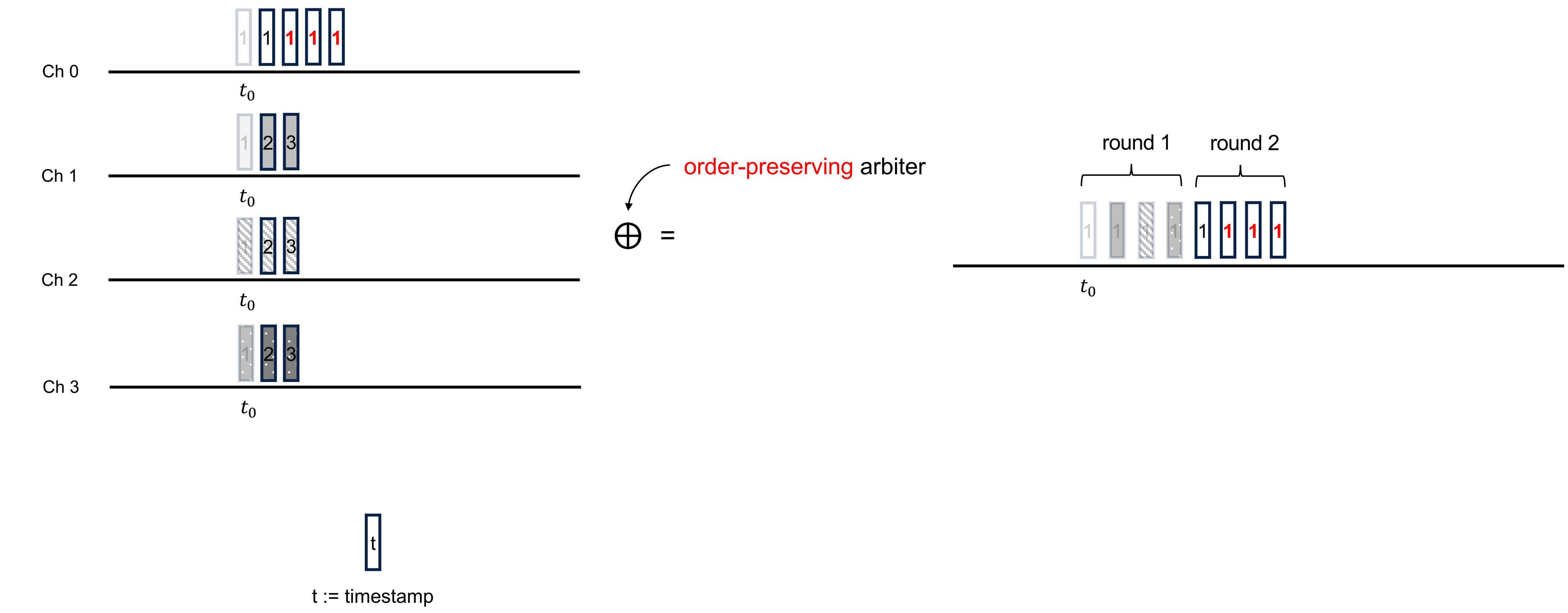


Problem formulation (cont.)

Q: What ruins the event **ordering**? (cont.)

Arbiter choice 3: **order-preserving** (granting the smallest HOL)

detector channel $\sim \mathcal{P}(\lambda)$



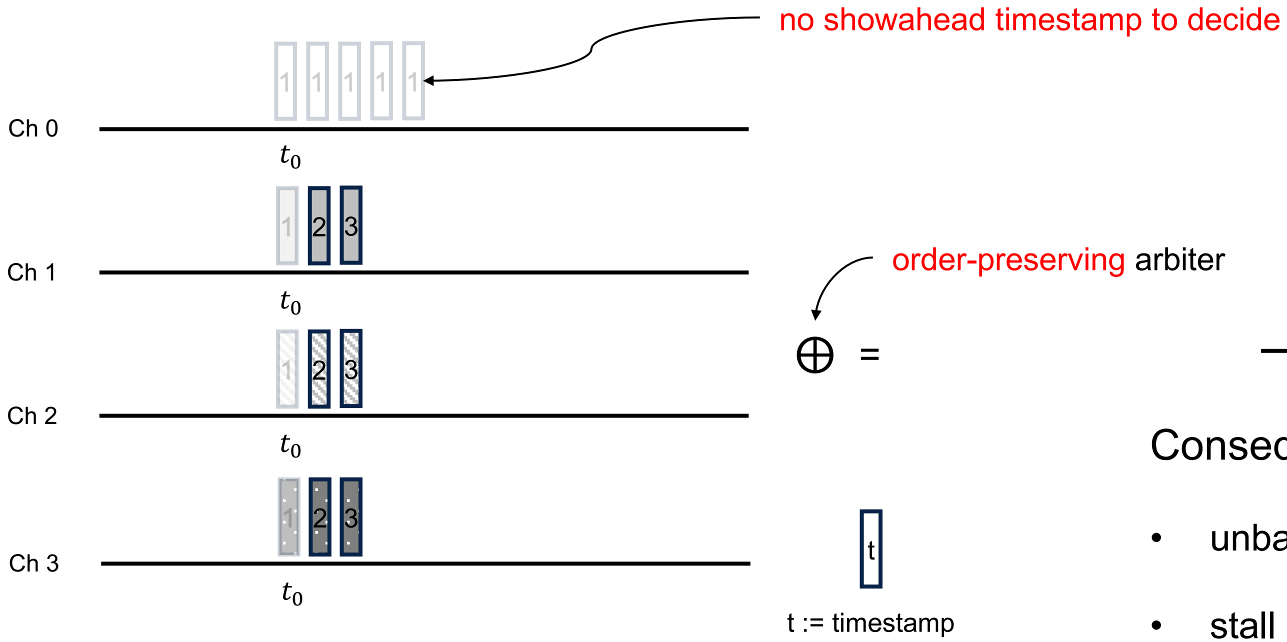
Problem formulation (cont.)

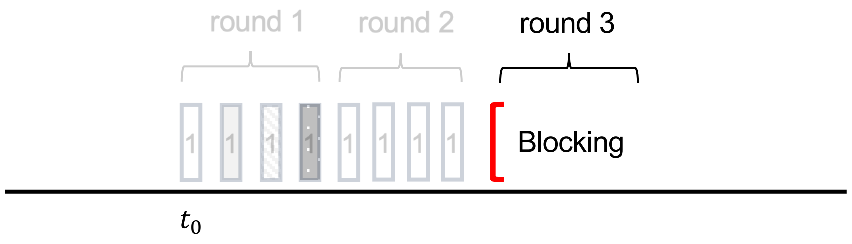
Q: What ruins the event **ordering**? (cont.)

Arbiter choice 3: **order-preserving** (granting the smallest HOL)

➢ Q: What is the issue with order-preserving? – complexity and stall (**delay jitter**)

detector channel $\sim \mathcal{P}(\lambda)$





Consequence of non-work-conserving:

- unbalanced channel rate induces blocking
- stall time can **accumulate** across stages
- insert beacon signal can help, but merely bound the stall time.
- if packet loss and overflow? **Deadlock** (infinite HOL block)
- **delay jitter depends on the input timestamp distribution**

Out-of-order arrival is inevitable. No single solution of arbiter can solve the unsortiness problem. Sorter is needed.

Re-sequencing buffer (“sorter”) implementation

Q: How to sort?

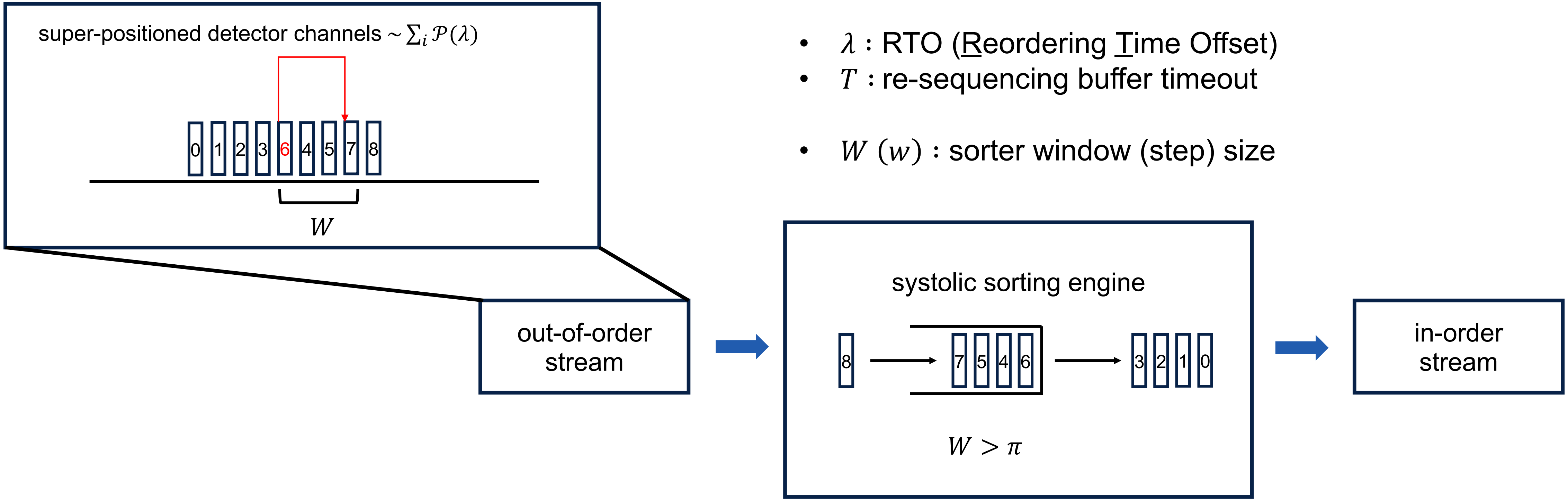
Sorter choice 1: **systolic array**

How to quantify the **timestamp reordering** in terms of displacement in space and time

- π : **RBO** (Reordering Byte Offset)
- B : re-sequencing buffer size

- λ : RTO (Reordering Time Offset)
- T : re-sequencing buffer timeout

- W (w) : sorter window (step) size



E. Mohammadpour and J. -Y. Le Boudec, "**On Packet Reordering in Time-Sensitive Networks**," in IEEE/ACM Transactions on Networking, vol. 30, no. 3, pp. 1045-1057, June 2022, doi: 10.1109/TNET.2021.3129590

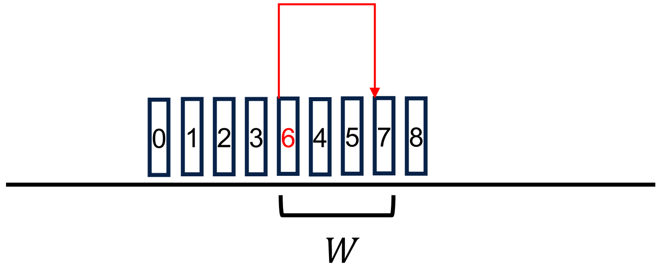
Re-sequencing buffer ("sorter") implementation

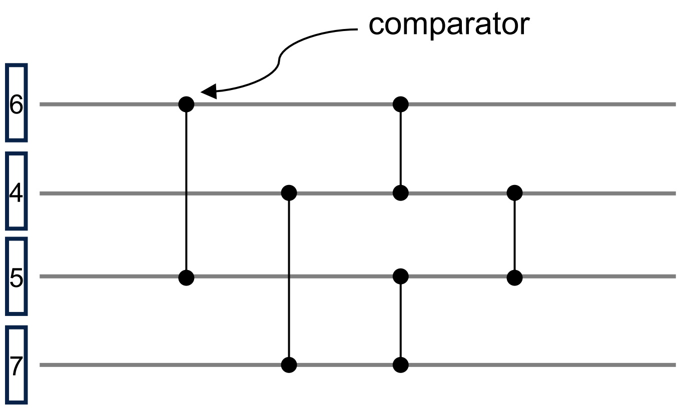
Q: How to sort?

Sorter choice 1: **systolic array** (cont.)

- π : **RBO** (Reordering Byte Offset)
- λ : RTO (Reordering Time Offset)
- W (w) : sorter window (step) size

super-positioned detector channels $\sim \sum_i \mathcal{P}(\lambda)$





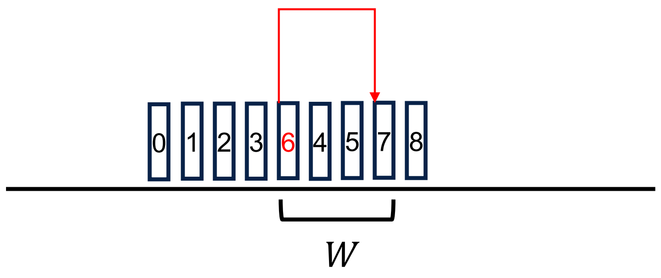
Re-sequencing buffer ("sorter") implementation

Q: How to sort?

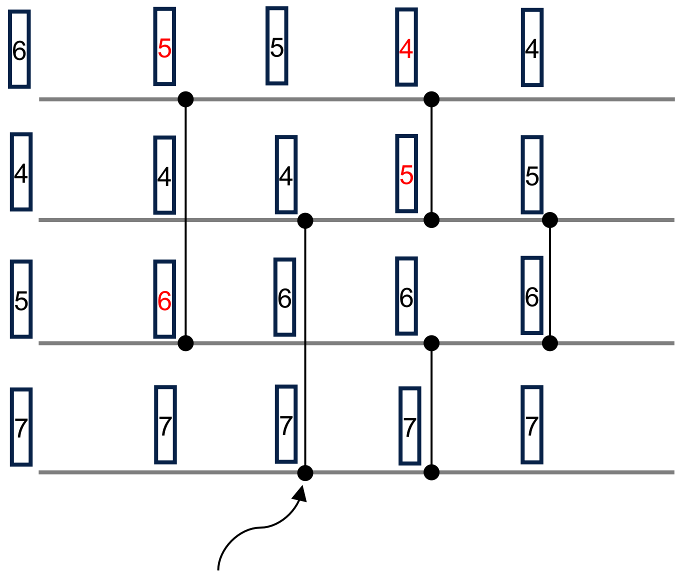
Sorter choice 1: **systolic array** (cont.)

- π : **RBO** (Reordering Byte Offset)
- λ : RTO (Reordering Time Offset)
- W (w) : sorter window (step) size

super-positioned detector channels $\sim \sum_i \mathcal{P}(\lambda)$



output: smallest on top



Re-sequencing buffer (“sorter”) implementation

ETH zürich

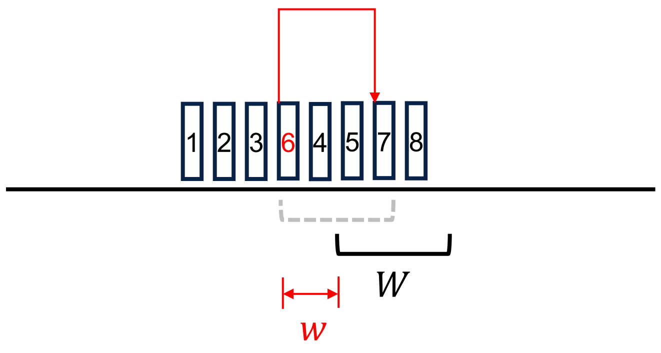
Q: How to sort?

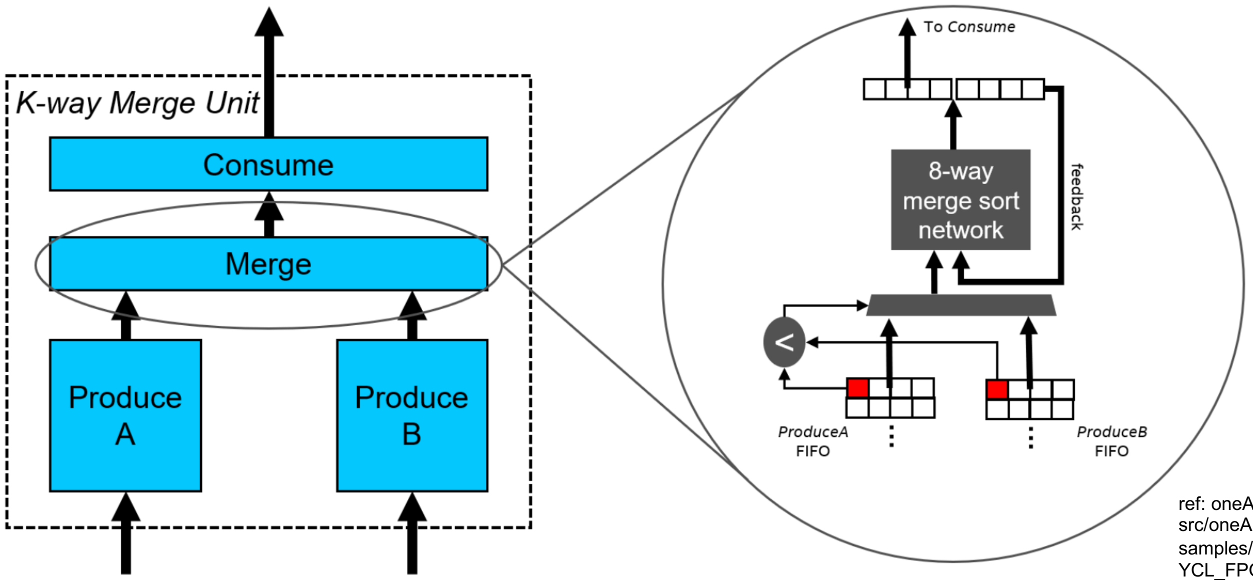
Sorter choice 1: **systolic array** (cont.)

- Q: What is the issue with systolic array?

- α : arrival curve
- V : delay jitter bound
- B : re-sequencing buffer size
- T : re-sequencing buffer timeout

super-positioned detector channels $\sim \sum_i \mathcal{P}(\lambda)$





ref: oneAPI (https://github.com/oneapi-src/oneAPI-samples/tree/master/DirectProgramming/C++SYCL_FPGA/ReferenceDesigns/merge_sort)

Require :

$W > \pi$

$w \to 1$

RBO bound :

$\pi < \alpha(V) - 1$ Packet arrival in time window of jitter

Buffer size bound :

$B > \max(\pi, \alpha(V + T))$ Packet arrival in time window of jitter

Issues :

- for RR : π (RBO) is loosely bounded
- Scalability is poor, space O(N^2) – O(N^3), W < 64
- TS dist. limits $\min(w)$, time > O(1)

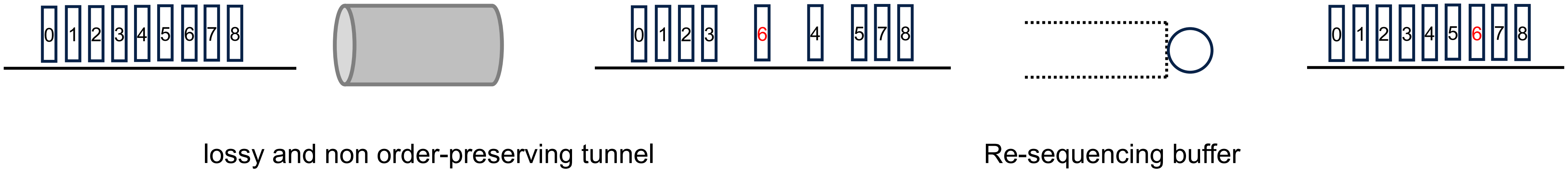
Re-sequencing buffer ("sorter") implementation-2

Q: How to sort?

Sorter choice 2: **???**

- Q: How to bound the RTO?

- π : RBO (Reordering Byte Offset)
- λ : **RTO** (Reordering Time Offset)



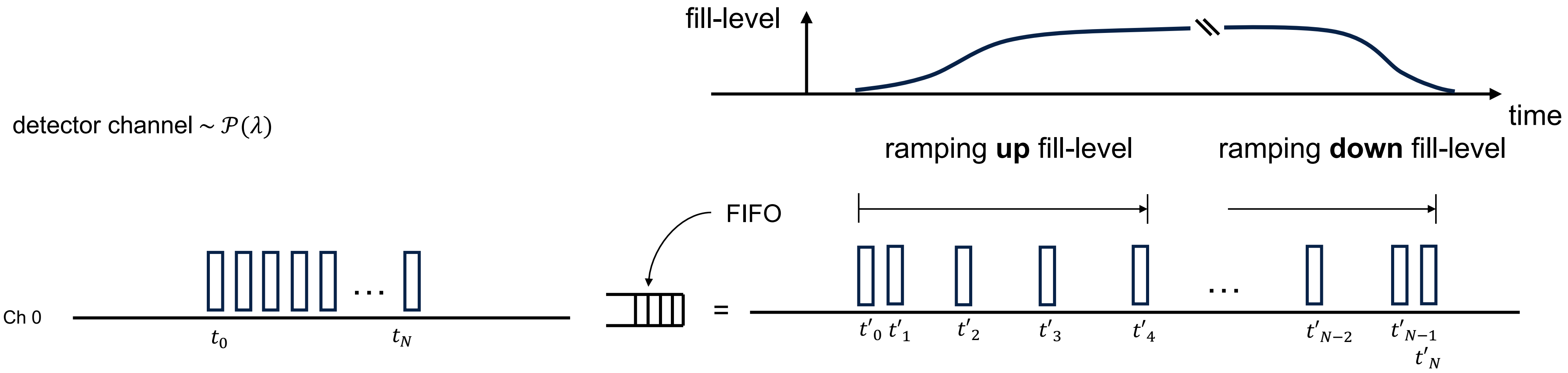
RTO : time offset of the inverted packet w.r.t its **assumed arrival time**

ETH zürich

Delay of a non memoryless system

FIFO ruins events arrival-time displacement (delay spread / introduce delay jitter)

detector channel $\sim \mathcal{P}(\lambda)$



time-variant
modulation by FIFO (stateful FIFO delay)

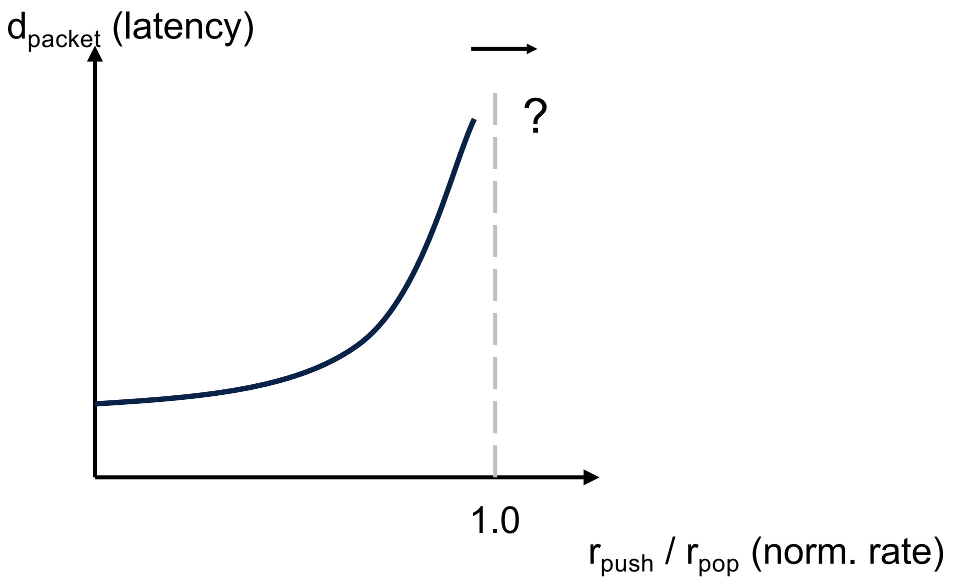
d_N : Latency of packet N := $t'_N - t_N$

Can RTO be bounded from above? $\exists\lambda \quad d_N < \lambda \quad \forall N$

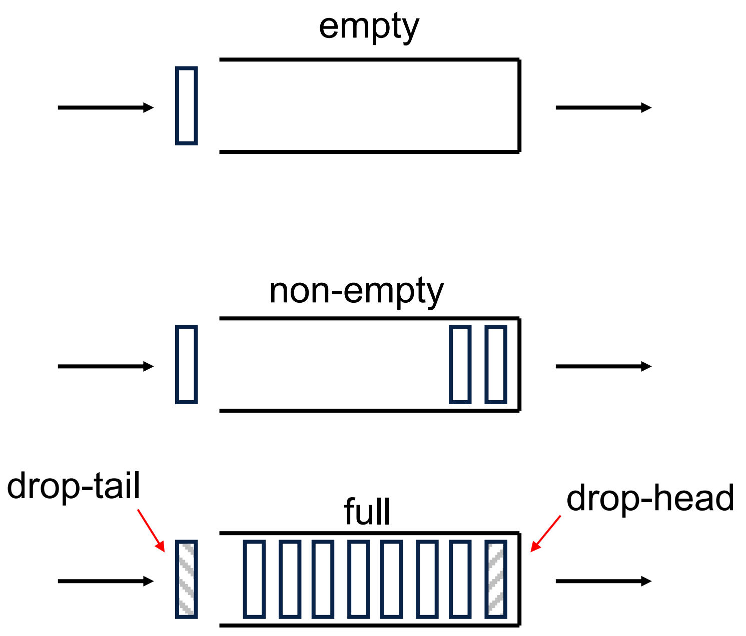
Delay of a non memoryless system (cont.)

Latency by FIFO (fill-level)

Macroscopic



Microscopic



$d = 0$

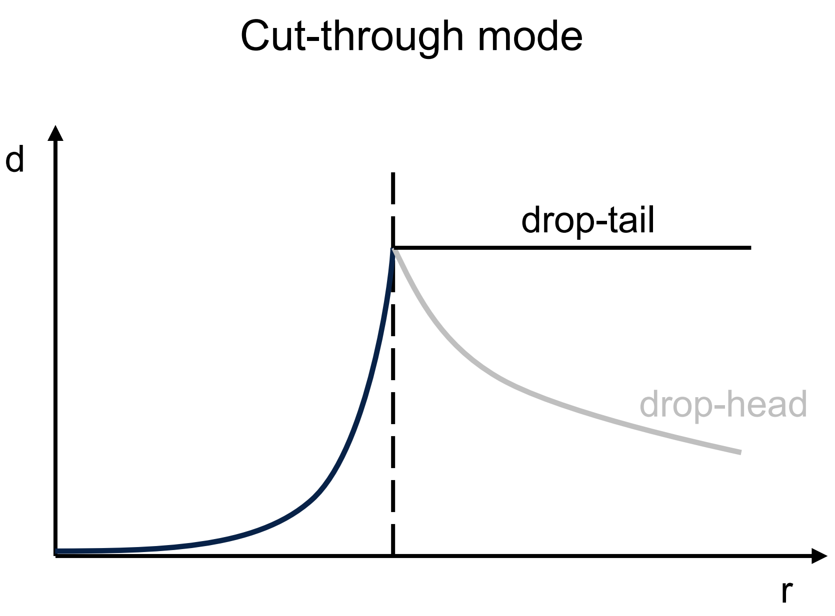
$d = \frac{n}{r_{pop}}$

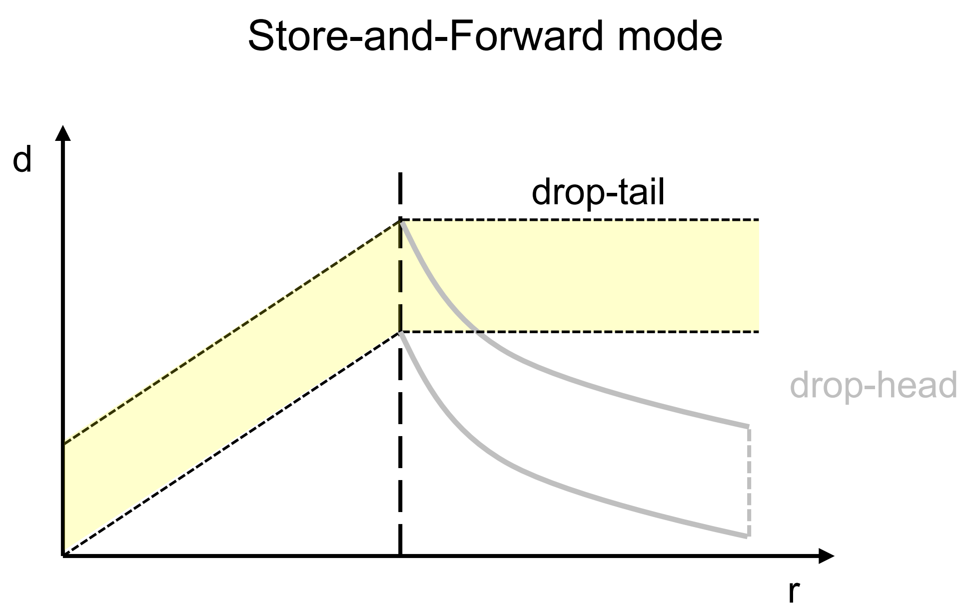
$d = (\infty, \frac{n}{r_{pop}})$

n : instantaneous fill-level
r : push/pop rate

Delay of a non memoryless system (cont.)

stateful FIFO delay (operation mode)





drop-head : possible recovery from deadlock
drop-tail : easy to implement

ETH zürich

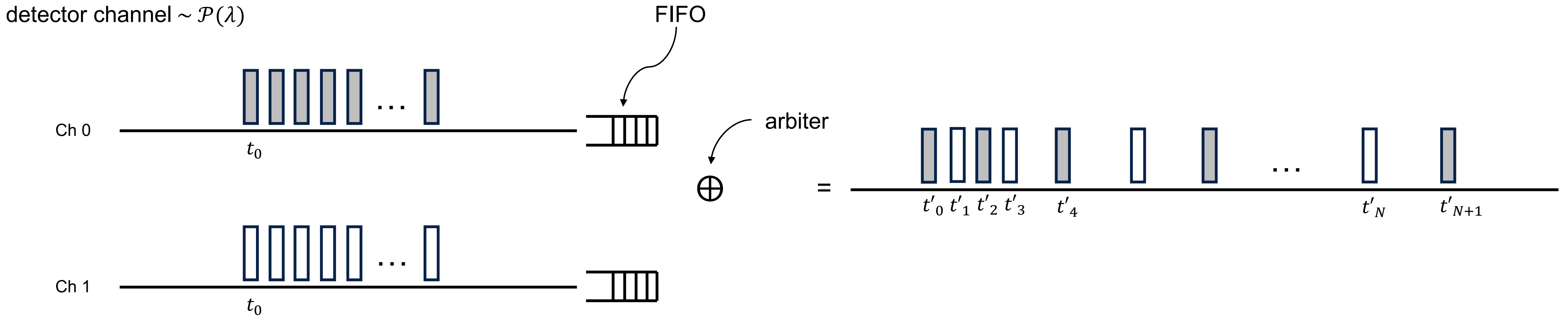
How to bound RTO in the system?

Sorter choice 2: ???

Q: How to bound the RTO?

➢ A: need to estimate the **delay jitter**

stateful FIFO delay and lane-skew accumulation



time-variant and **non-linear** **modulation by FIFO and arbiter**

d_N : Latency of packet N := $t'_N - t_N$

Out-of-Order: There exist output positions i < j such that $TS_i > TS_j$.