Next Generation Real Time Computers at ESO

Marcos Suárez Valles

Historical Context

Starting Point for the ELT RTC Discussion

2016: legacy VLT RTC platform not reusable for the ELT

- Majority of VLT AO systems running on an in-house RTC platform (**SPARTA**):
  - Hybrid FPGA / DSP / CPU core **obsolete** – w/ last RTC instance still ongoing
  - Successor technology does not scale well (at a reasonable cost) to the ELT
  - Linux cluster healthy and maintainable, but not aligned with ELT standards
- **Expertise gap** at ESO wrt. to new, fast-developing HPC technology

- ELT **resources constrained**: unclear way forward for an ELT RTC platform
- Promising **community projects** exploring RTCs with HPC technology
- Comparable size telescopes moving to mainstream server technologies

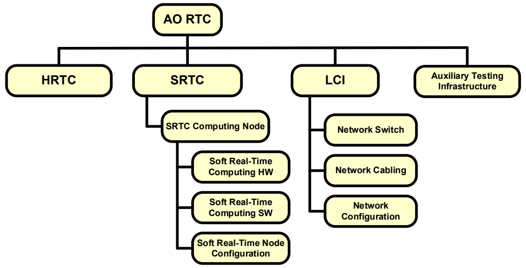
IDENTIFY VIABLE STRATEGY FOR FIRST-LIGHT RTC

Supporting Strategy for ELT First-Light RTCs

No fully-fledged ELT RTC platform to be produced by ESO!

- **Standardize** RTC architecture (incl. networking):
  - Functional and product breakdown largely obsolescence-driven
  - Building blocks with clear function and well-defined I/F

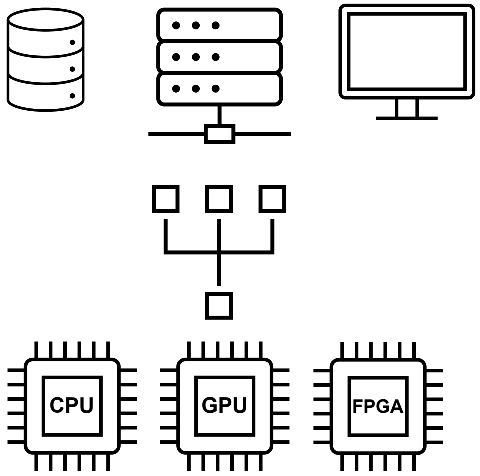
ESO-287079



- **Concentrate** long-term **resources** on cluster infrastructure:
  - Long-lived, likely to benefit from incremental evolution of ESO standards
  - Produce RTC-specific software framework on top of ELT software
  - Focus on maintainability and conservative **upgrade path**

- **Isolate** performance-critical, core **AO loops** behind network I/Fs:
  - Enable realizations with different technologies – depending on Instrument
  - Focus on maintainability and facilitate end-of-life **replaceability**

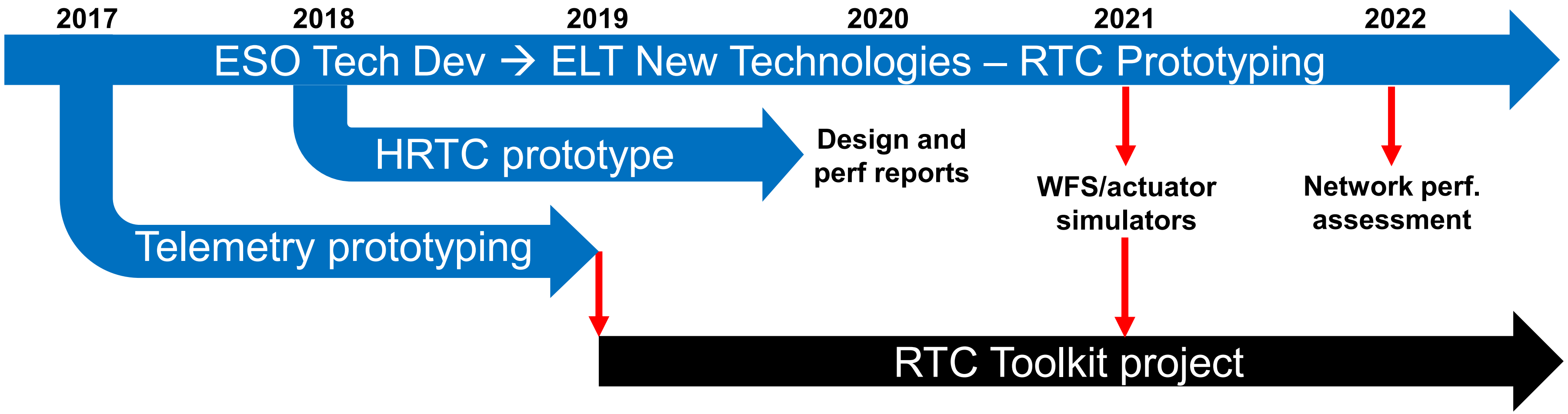
Cluster: long-term ESO standard



AO loops: industry standard evolving faster

ELT RTC Risk Mitigation Measures

Long-term RTC prototyping activity to de-risk ELT First-Light RTCs



- Early focus on Telemetry distribution → splinters (scope extended) into **RTC Toolkit** project
- Hard real-time performance with mainstream technology → **HRTC Prototype** external contract
- Initial deterministic WFS / actuator **simulators** → further evolution taken over by RTC Toolkit
- Long-term focus on deterministic **networking** and network performance verification

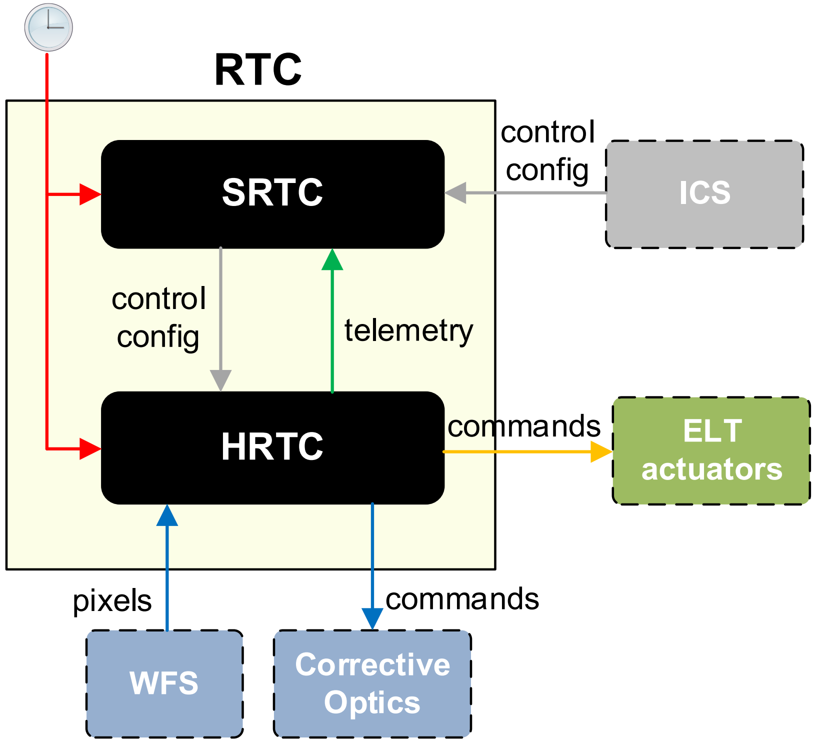
The RTC Standard Architecture

Data Classification: ESO PUBLIC

RTC Computational Building Blocks

Physical separation of functions by technology roadmap

- Hard Real-Time Core (**HRTC**):
  - Performance-critical AO loops; direct sensor/actuator I/F
  - May require 3rd party software – license constrained by ESO
  - May need to diverge from ESO server hardware standards
  - May require dedicated spares; replaceable as a whole
- Soft Real-Time Cluster (**SRTC**):
  - Telemetry-based optimization, coordination, recording, etc.
  - Based on RTC Toolkit – baseline: no 3rd party software
  - Based on ESO server hardware standards – incl. GPU
  - Follows upgrade path of mainstream server technologies

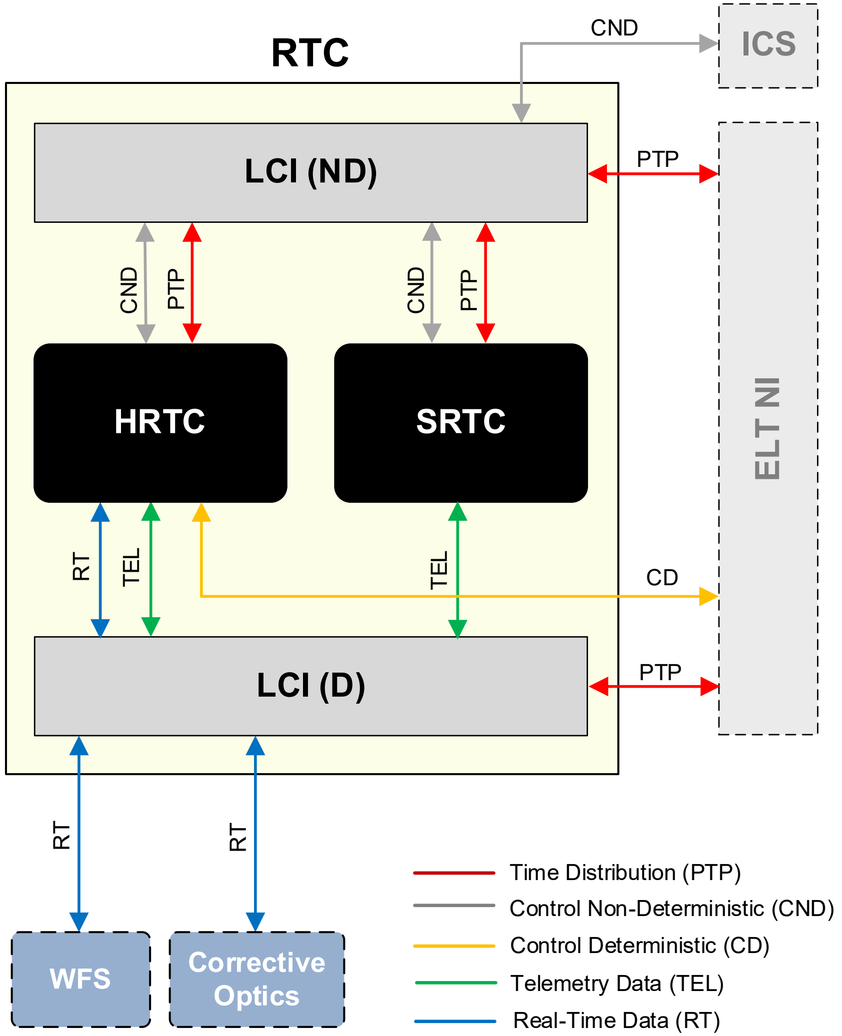


Data Classification: ESO PUBLIC

RTC Local Communications Infrastructure (LCI)

Acknowledge central role of networking in ELT

- Physically separate deterministic (D) vs non-deterministic (ND) LCI:
  - LCI (D): minimum latency / jitter – LCI (ND): best effort
  - Deterministic amount of switch shared resources where required
  - Platform-specific LCI (D) switch optimizations confined
- Partitioning into physically separate subnets by function:
  - Reduces background traffic / endpoints; increases flexibility
  - Not necessarily L2 design; low-impact L3 routing possible
  - System-wide PTP synchronization – switch as boundary clock
- Based on ELT NI standard hardware – where feasible:
  - Follows upgrade path of mainstream Ethernet technology



The RTC Toolkit

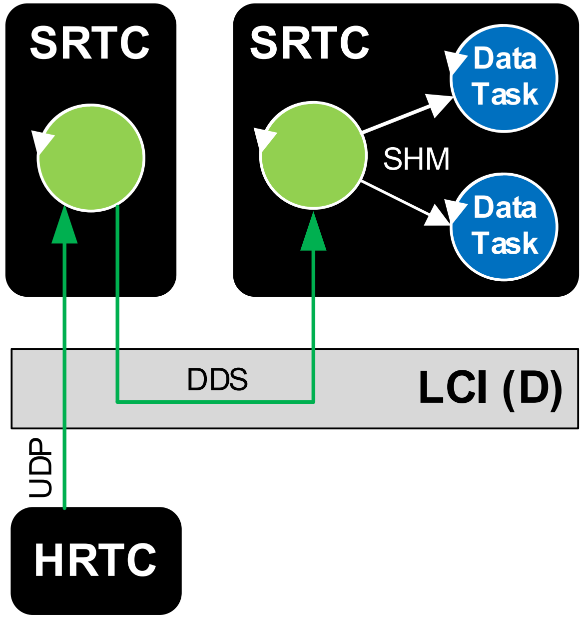
Related talks:

- See Day 1, 10:20 - *“RTC Toolkit – Overview and Status”, B. Jeram*

Key RTC Toolkit Design Goals

Common framework for SRTC applications with lessons learnt from VLT

- True **interoperability** with the ELT infrastructure and Instrument software:
  - Native integration for database, command / reply, events, visualization, etc.
- Scale / adapt successful VLT patterns to the ELT; address known pitfalls
- Improve telemetry distribution to *"data tasks"*:
  - Avoid retransmissions by HRTC, *"all-data-out"* within loop cycle – UDP
  - Keep reliable multicast delivery amongst SRTC nodes – DDS
  - Address VLT scalability limits: less DDS endpoints per node + shared memory
  - Prevent VLT failure modes: avoid buffer pressure back-propagating to DDS
- Enable **GPU acceleration** for *"data tasks"* – *"conservative"* upgrade path
- Avoid VLT *"data task"* contention: enable **resource-aware** deployment



Data Classification: ESO PUBLIC

The HTRC Prototype

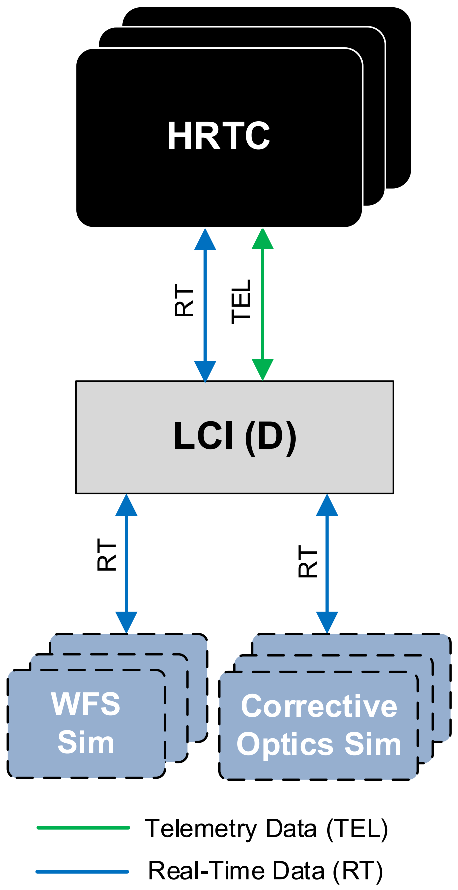
Related talks:

- **Day 1, 12:40** - *"HRTCp: ELT-sized hard real time core on common-off-the-shelf hardware", N.H. Pedersen, P.H. Kampf*

HRTC Prototype Motivation and Goals

Why embarking in prototyping with no client project? What to prototype?

- ESO does not develop any ELT Instrument RTC – but will maintain all of them:
  - Confirm that *“maintainable”* solutions are within reach of ELT consortia
  - Generate knowledgebase for future internal projects / maintenance
- Explore performance envelope under the standard RTC architecture:
  - Restrict *“fully compliant”* scope to HRTC, LCI (D) and critical interfaces
  - Address deterministic networking, robustness of UDP communication
- Target CPU-based, multi-core, multi-socket, HPC-class server technology:
  - Maintainability synergy: aligned with ELT control system and ESO staffing plans
  - Community projects already exploring GPU accelerators extensively
  - Prioritize maintainability over compactity, cost and marginal performance gains

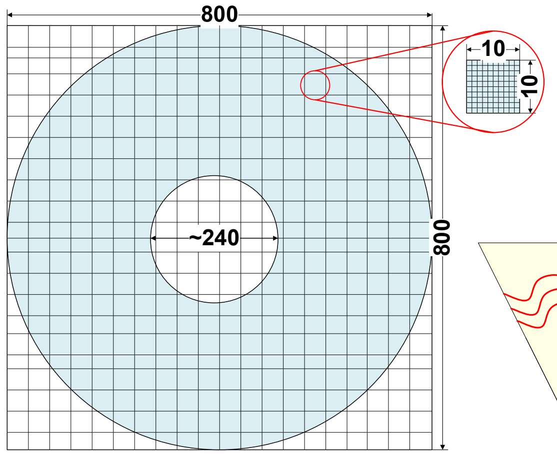


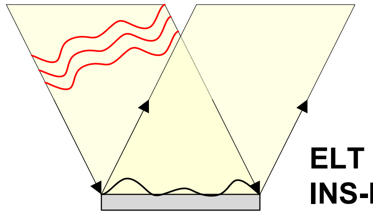
HRTC Prototype Performance Envelope

Size and performance a "maintainable" RTC for the ELT problem size?

Prototype dimensioning: (early) ELT MCAO size

- 6 x LGS WFS, simulated ESO LISA camera
- Simulated ELT M4 / M5 + 2 x INS DM
- POL control: 6,316 x 55,392 MVM reconstruction
- Basic IIR time-filtering with HO / TT separation





Measured performance:

- End-to-end latency **~250 µs** avg., **< 3 µs** std. dev.
- Trading latency for **sub-µs** jitter (!) possible

KEY DESIGN PARADIGMS CONTRIBUTING TO PERFORMANCE IDENTIFIED

Data Classification: ESO PUBLIC

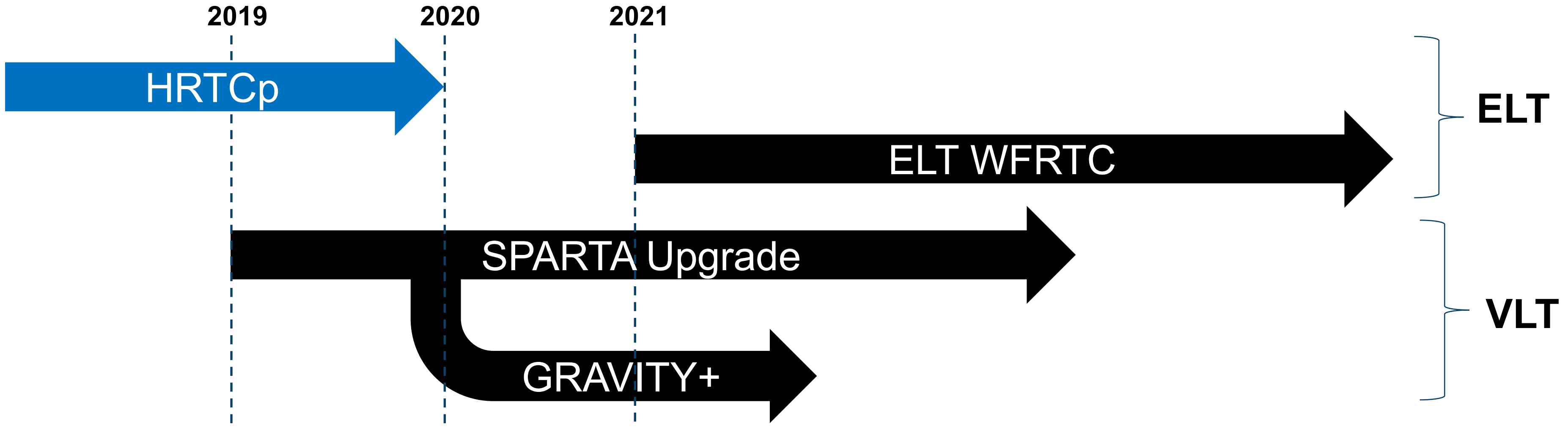
The HRTC Prototype Heritage

Related talks:

- **Day 1, 15:30** – *“SPARTA Upgrade”, P. Shchekaturov*
- **Day 1, 15:50** – *“GRAVITY+ RTC Design”, R. Dembet*
- **Day 3, 14:00** – *“ELT WFRTC - Software patterns and library solutions for low latency multithreading”, C. Rosenquist*

ESO Reuse of HRTC Prototype Knowledgebase

(Partial) adoption of HRTCp paradigms by ELT and VLT internal projects



- HRTCp is ***not*** a platform → useful paradigms are re-implemented independently by the projects
- SPARTA Upgrade: early adoption, constrained by legacy sensor/actuator I/F and cluster APIs
- GRAVITY+: re-uses SPARTA upgrade code base, adding support for Ethernet sensor/actuator I/F
- ELT WFRTC: ELT-native RTC; largely generalizes and maps key concepts to supporting libraries

Useful HRTC Prototype Paradigms (I)

Initialize only the hardware that you need, decouple HRTC upgrade path

Controlled initialization of HRTC hardware and services:

- HRTC boots custom `initrd` image and kernel over the network (PXE) → no hard-disk
- Only network cards and essential services initialized using `init.d` → no `systemd`

Monolithic HRTC application, (mostly) statically built:

- Minimum dependencies with `initrd` image – kernel headers in image same version or newer
- ELT WFRTC links dynamically `glibc`, `libstdc++` to enable VDSO – faster clock access

HRTC application built from within VLTSW / ELT Dev Env:

- Dedicated vs standard toolchain depending on compiler support for HRTC hardware; dedicated toolchain in the long term enables decoupling HRTC and VLTSW / ELT Dev Env upgrade paths
- `initrd` image generated using `Buildroot` – also builds toolchain as by-product

Useful HRTC Prototype Paradigms (II)

Exploit visibility within process and resource locality

Multiple threads within single process space:

- Direct visibility of shared data controlled during construction: no shared memory IPC overheads

NUMA-awareness and thread pinning:

- Enabling in BIOS of fine-grain NUMA node configurations – dedicated RAM, PCIe root
- Control of CPU affinity and memory policies at thread creation – incl. migration of thread's stack
- NUMA-aware memory allocations – use of `std::pmr::memory_resource` by ELT WFRTC

Locality to network interfaces:

- IRQ affinity, co-location of Rx/Tx threads in NUMA node of corresponding PCIe root

Useful HRTC Prototype Paradigms (III)

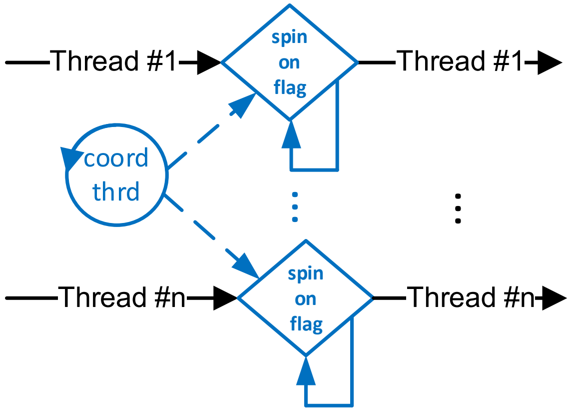
"Quiet" CPU cores for improved determinism, use lockless synchronization

Latency- / jitter-critical threads spinning in *"quiet"* CPU cores:

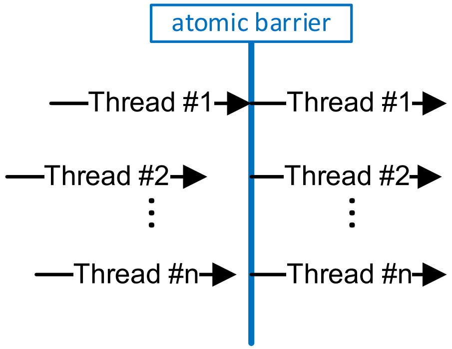
- Isolate from OS scheduler, omit scheduling ticks, disable RCU call-backs, remove IRQs

Lockless synchronization:

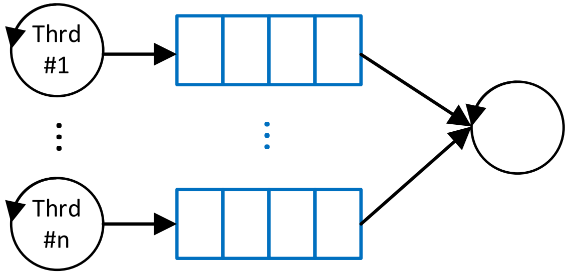
- Inter-thread coordination based on wait-free atomic flags and original counter-tracker setup
- ELT WFRTC introduces atomic signals, `std`-like wait-free barriers and lockless queues



Dedicated supervisory thread signals flags



Supervision executed by selectable thread in the warp



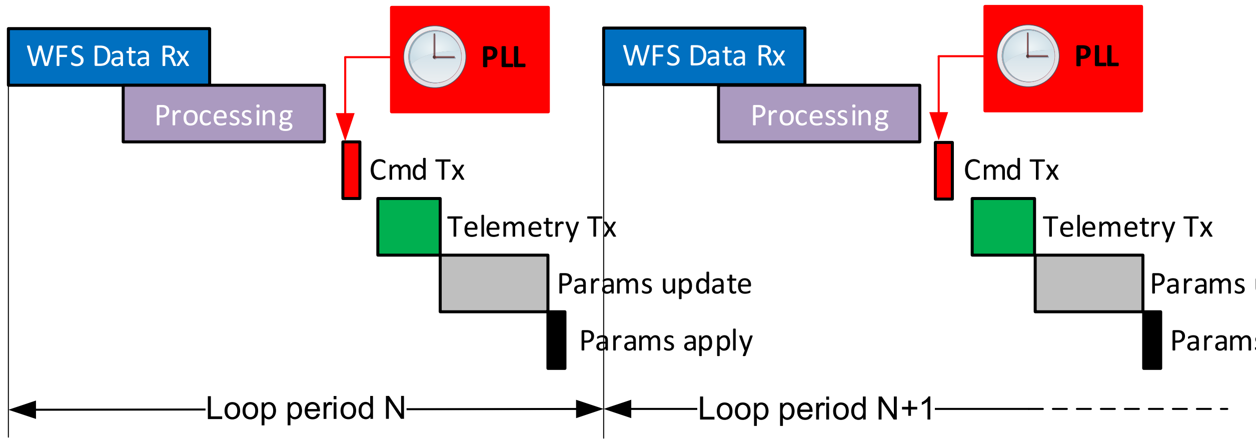
Threads enqueue work and continue processing

Useful HRTC Prototype Paradigms (IV)

Time-separation of critical processing, re-synchronize for better jitter

Time-slicing of AO loop cycle:

- Schedule operations subject to performance cross-talk in distinct time slots
- Piece-wise copy large parameters to pipeline incrementally over several loop cycles



Jitter reduction by PLL synchronization:

- Phase-lock command transmission to *"processing complete"* plus delay – to shadow jitter
- Trades off excess latency for potentially sub-*μ*s jitter in jitter-critical applications

Data Classification: ESO PUBLIC

RTC Performance Verification

Related talks:

- **Day 3, 14:00** – *“Ethernet packet time stamping techniques for real time performance assessment“, T. Grudzien*

PTP-based RTC Performance Observability

Make RTC performance observable during operation

- **VLT**: limited RTC performance observability once in operation:
  - Fine-grained, absolute time-stamping costly → not integrated in operational metrics
  - Latency / jitter measurements require I/O signals and *ad hoc* **oscilloscope** setup
- **ELT**: PTP enables RTC performance trending and *"on-the-wire"* verification:
  - Cheap access to corrected clock in most HRTCs → performance metrics *"always on"*
  - Oscilloscope replaced by (possibly integrated) Ethernet **packet capture** hardware
- Current packet capture strategies at ESO:
  - Dedicated *"sniffer"* cards – various generations tested at ESO with good results
  - Regular network cards with *HW_TIMESTAMP_ALL*, synchronized to system (corrected) clock
  - PTP time stamping **inside network switch** and propagation of captured packets via ERSPAN

Thank you!

Marcos Suárez Valles
msuarez@eso.org

@ESOAstronomy
@esoastronomy
@ESO
european-southern-observatory
@ESOobservatory