LGWR Beyond The Docs

What Oracle Didn't Write To Disk Part 2

SOUG Day 01/2026
March 11th, 2026
Bern

Christoph Lutz
Swisscom

Blog: https://www.0xchris.dev
Twitter: @chris_skyflier
Bluesky: @christophlutz.bsky.social
Github: Christoph-Lutz

swisscom

Before we start …

This is a low-level technical presentation about internal and undocumented behavior

Beware that:

- Things can change across different versions and patch levels
- My observations, findings and interpretations may be inaccurate or wrong
- Some of the techniques shown in this presentation are dangerous – use them at your own risk!
- Examples and demos were tested in the following environments:
  - **Oracle 19.26** (Exadata 25.2.6, OEL 8.10, kernel 5.15.0-308.179.6.16, bpftrace 0.16)

Note: all examples and demos are available on Github

LGWR Features – Timeline

9i	**Log File Sync Timeout** **Log Parallelism** (multiple public redo strands)	PART 1
10g	**Private Redo Strands & In-Memory Undo**	
11g	**Adaptive Log File Sync**	
12c	**Adaptive Scalable LGWR (multiple LG worker processes)**	PART 2
19c	**Exadata X8M and X9M only (PMEM): Fast Log File Sync (Fast Sync)s**	PART 1
26ai	**Exadata only: Pipelined Log Writes (backported to 19.22+)**	PART 3

There's lots of innovation in LGWR, but unfortunately, all these features are mostly undocumented ...

LGWR & Redo Log Buffer

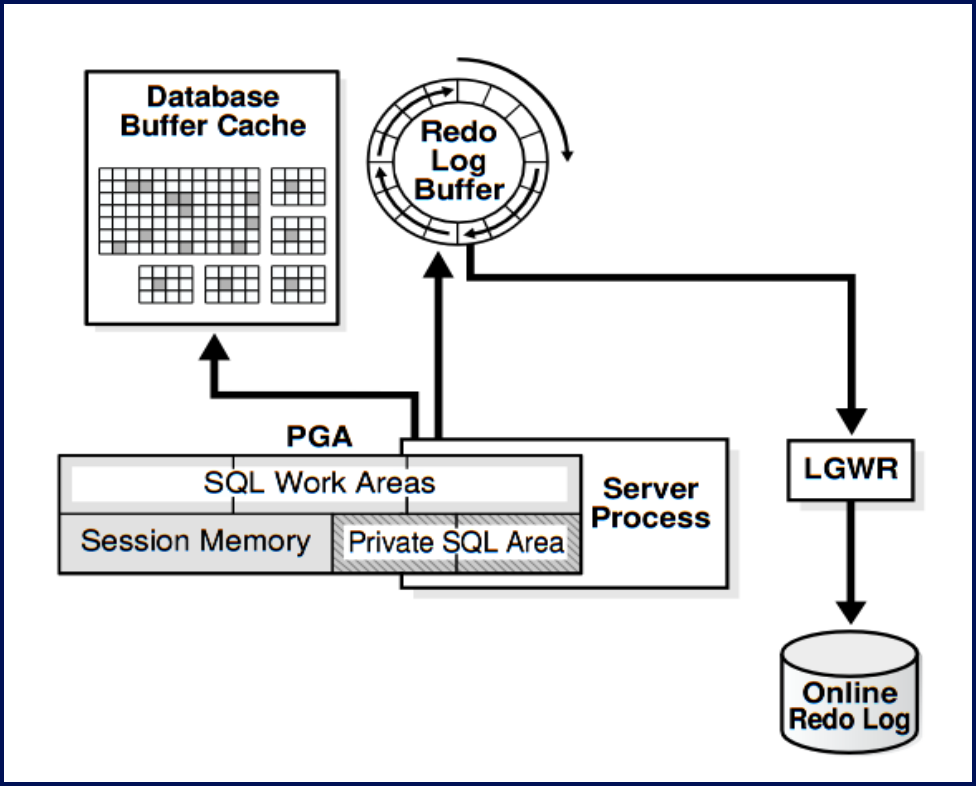


Figure source: Oracle Corp., Database 19c, Database Concepts, Section 15-21: Redo Log Buffer

The redo log buffer is a **circular buffer** in the SGA that stores redo entries describing changes made to the database.

[...]

The background process log writer process (LGWR) writes the redo log buffer to the active online redo log group on disk.

This is not wrong, but it leaves out an awful lot of important details ...

LGWR Fundamentals

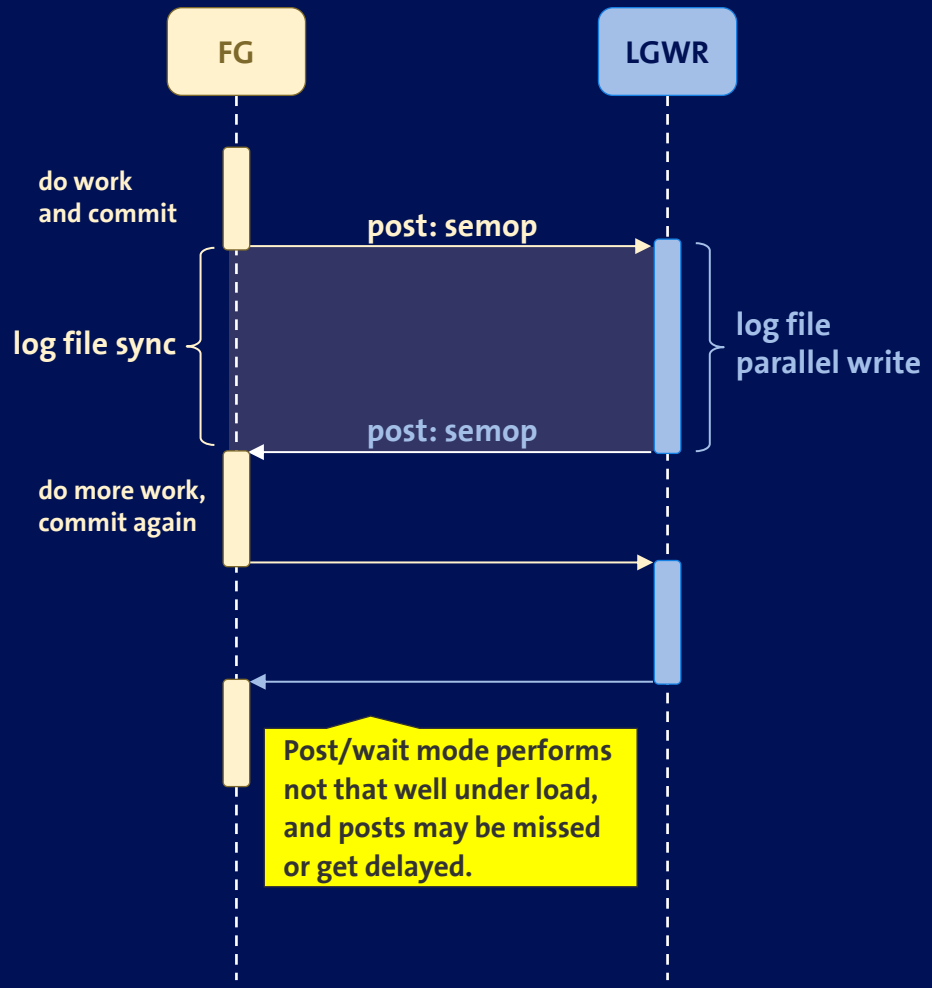
Commit Performance – Key Factors

Commit performance depends on:

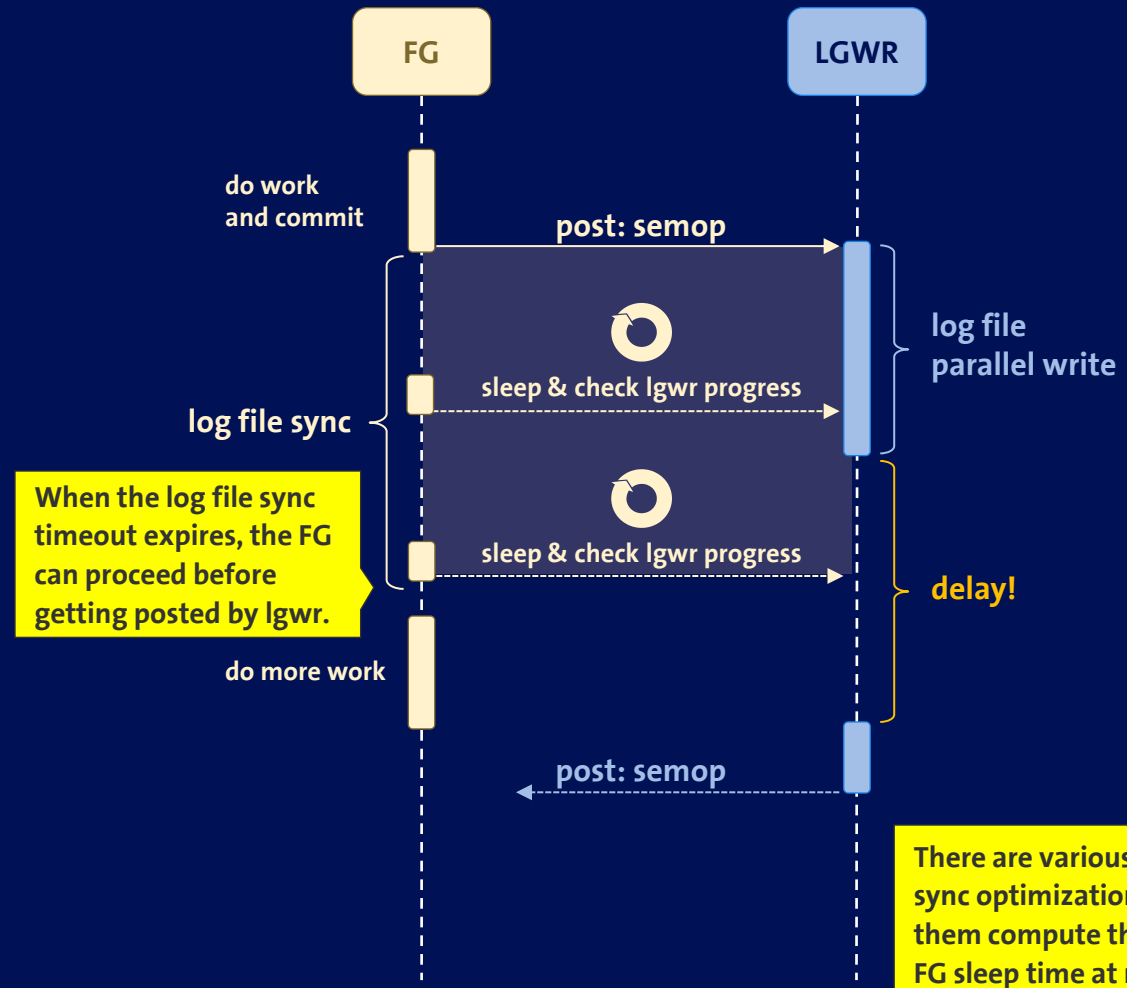
- the **speed of writing** redo records from memory to permanent storage
- the delay of **receiving confirmation** that the changes are safely persisted

Log File Sync Timeout

In the days of yore – Oracle 8 or before



Modern Oracle Versions

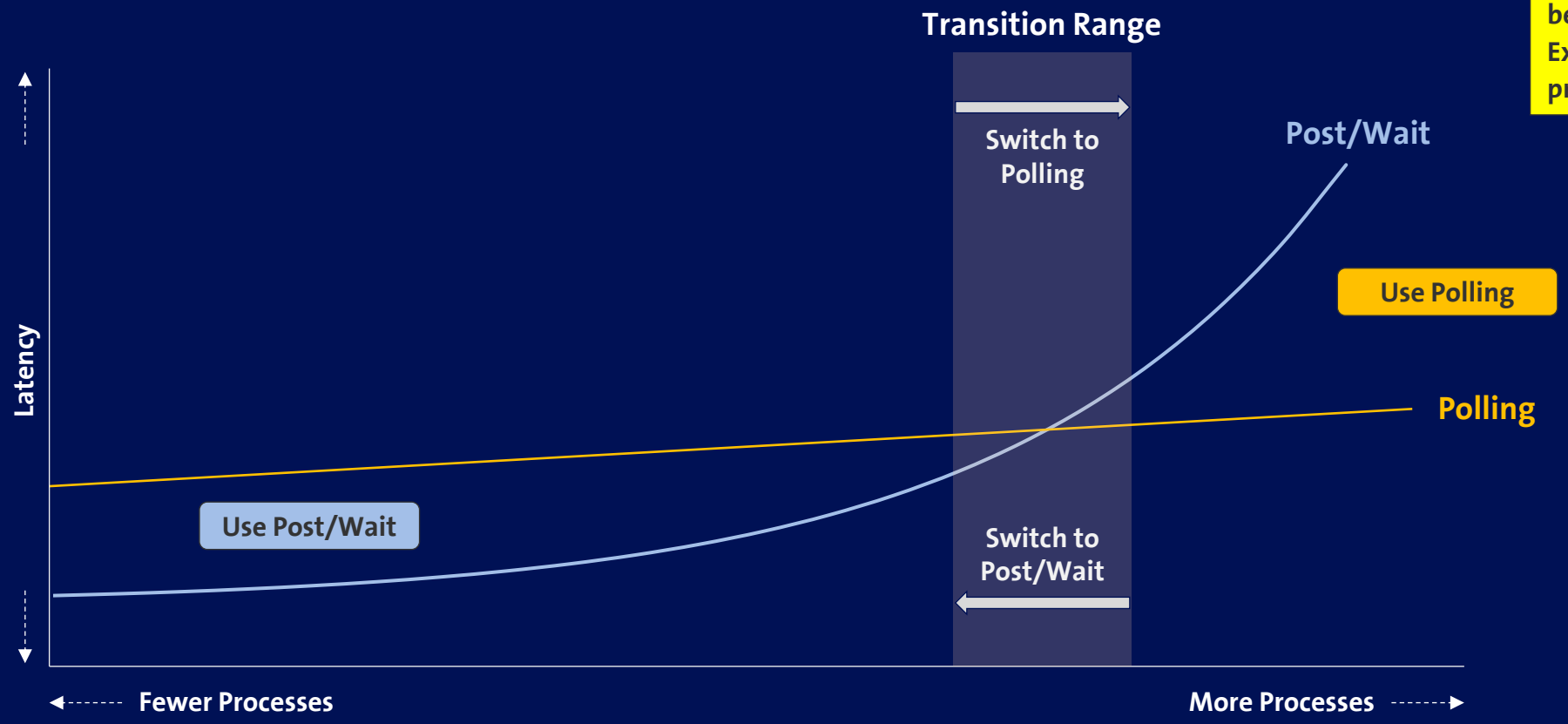


Log File Sync – Optimizations

Optimization	Platforms	Sleep Duration	Spin Duration	Oracle Version
Log File Sync Timeout	All platforms	10 cs (default)	n/a	11.2.0.1
Adaptive Log File Sync	All platforms	Adaptive	n/a	11.2.0.3
Fast Log File Sync	Exadata only?	Adaptive	Configurable	21c (backported to 19c)

Adaptive Log File Sync (ALFS) – Post/Wait vs Polling Mode

In Oracle 19c, ALFS has been enhanced to support Exadata systems with pmemlog (Fast Sync).

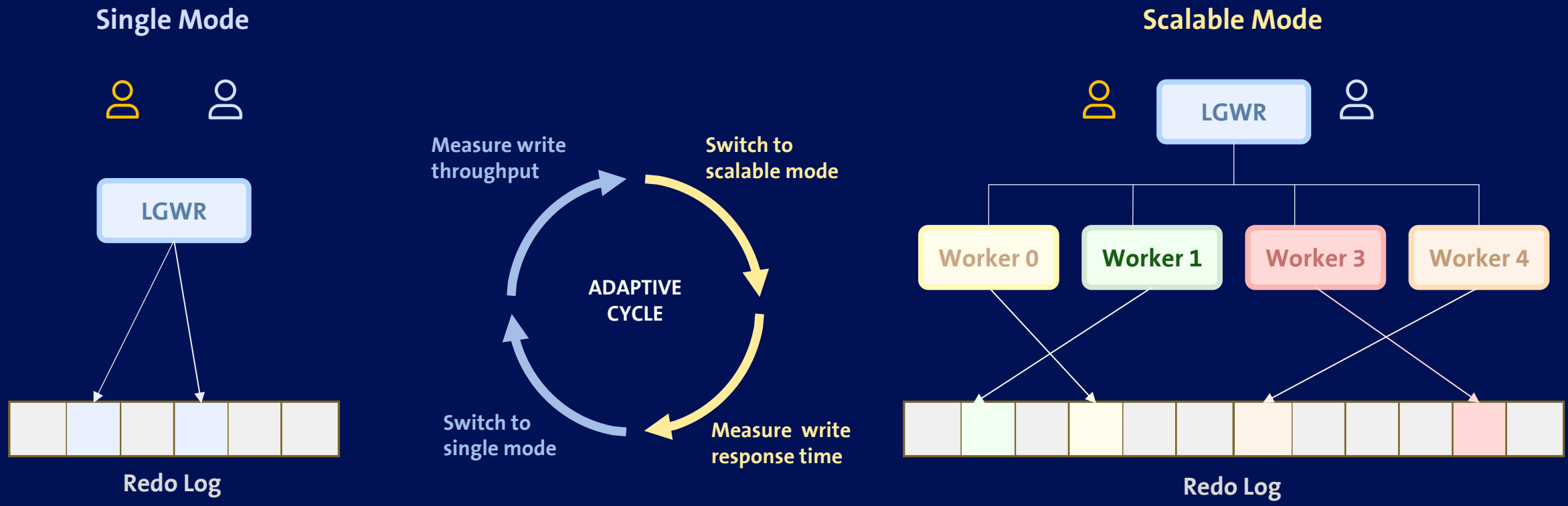


ALFS: Dynamically switch between Post/Wait and Polling mode at runtime!

Figure modified from source: Chi Cao Minh, et. al., United States Patent, 9418129 B2, Adaptive High Performance Database Redo Log Synchronization, August 2016

Adaptive Scalable LGWR

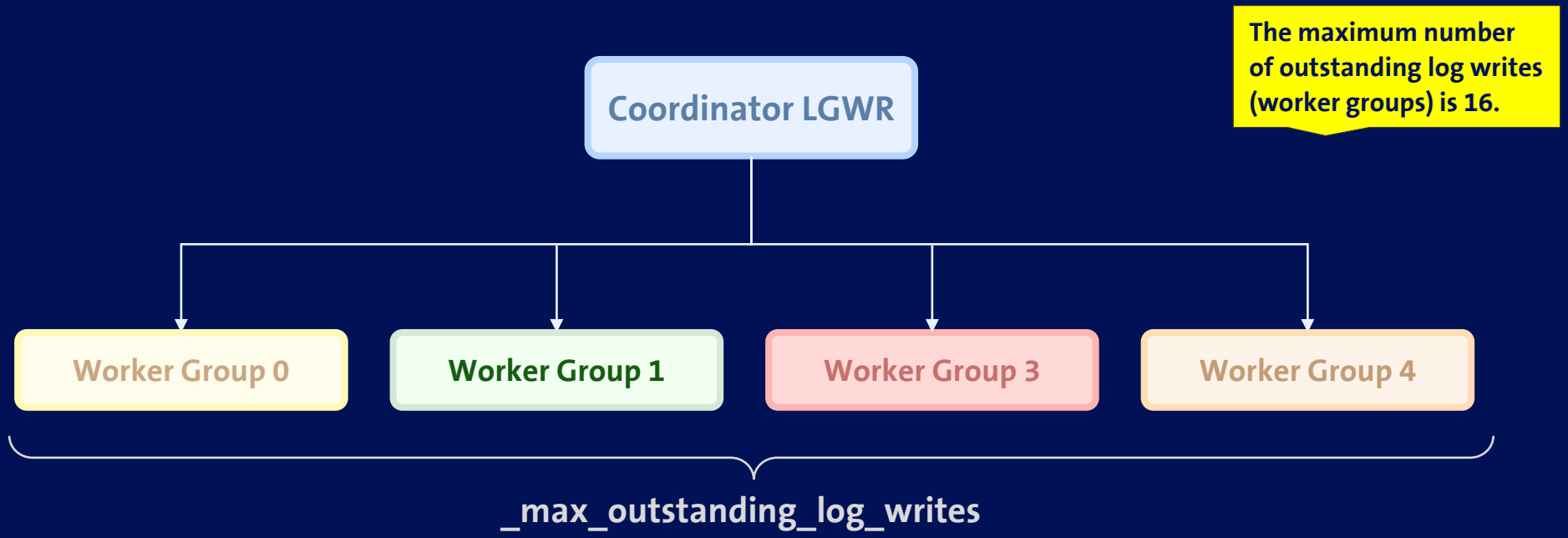
Adaptive Scalable LGWR – Overview



+ Responsive under moderate load (no coordination overhead)
– Limited write throughput (single process)

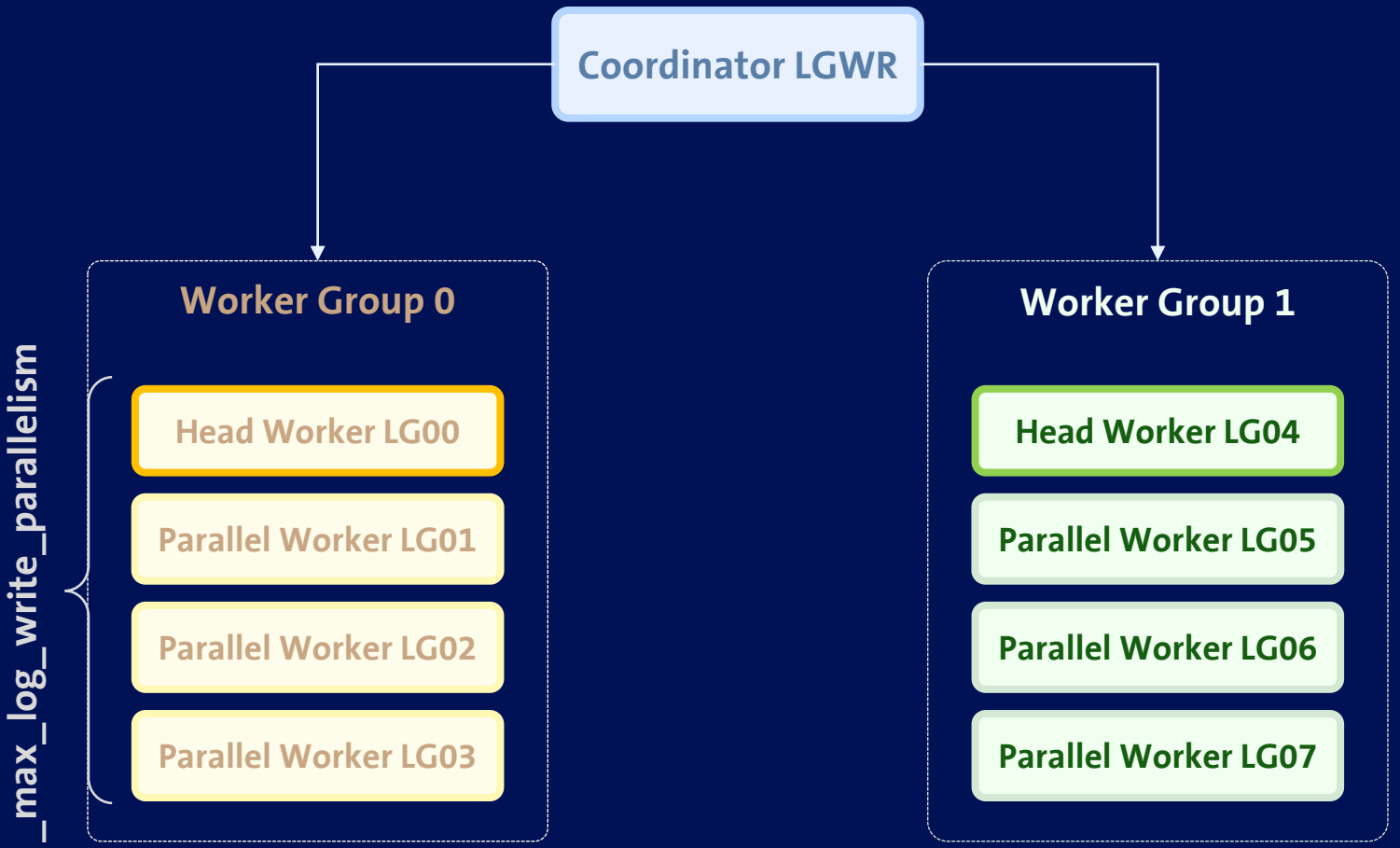
+ High write throughput (parallel writes)
– Less responsive due to coordination overhead

Adaptive Scalable LGWR – Concepts



The maximum number of outstanding log writes (worker groups) is 16.

Adaptive Scalable LGWR – Concepts



Head Workers:
Serial + parallel operations

Parallel Workers:
Parallel operations only

Serial operation:
Intra-Group Coordination
Advance on-disk SCN

Parallel operation:
Write

Adaptive Scalable LGWR – Concepts

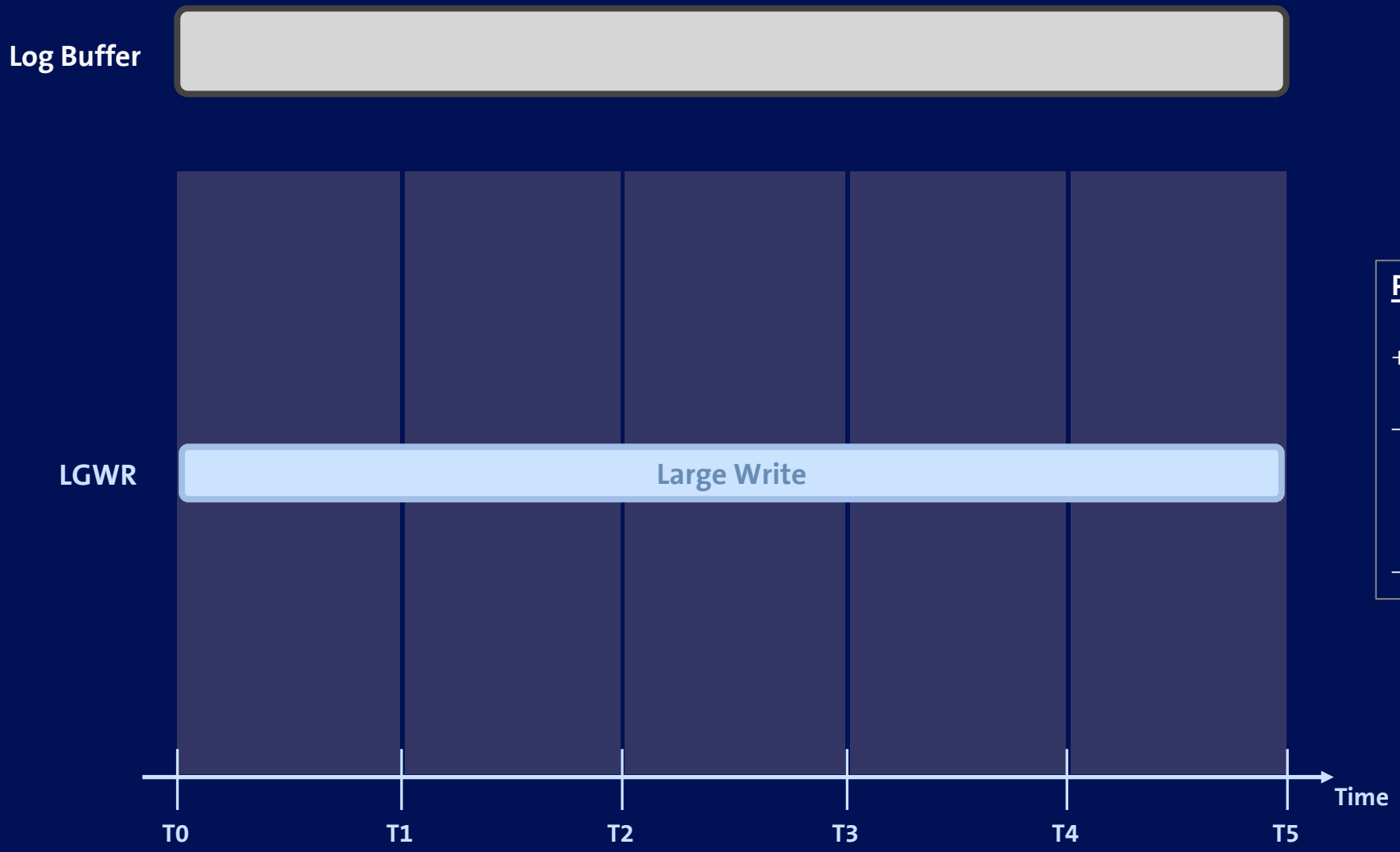
Nr of LG Worker Processes per Instance =

_max_outstanding_log_writes x _max_log_write_parallelism

Defaults:

- _max_outstanding_log_writes: 2 (hardcoded in the binary: struct ksptii)
- _max_log_write_parallelism: ceil(_log_parallelism_max / 4)

Single LGWR Write Patterns – Large Write



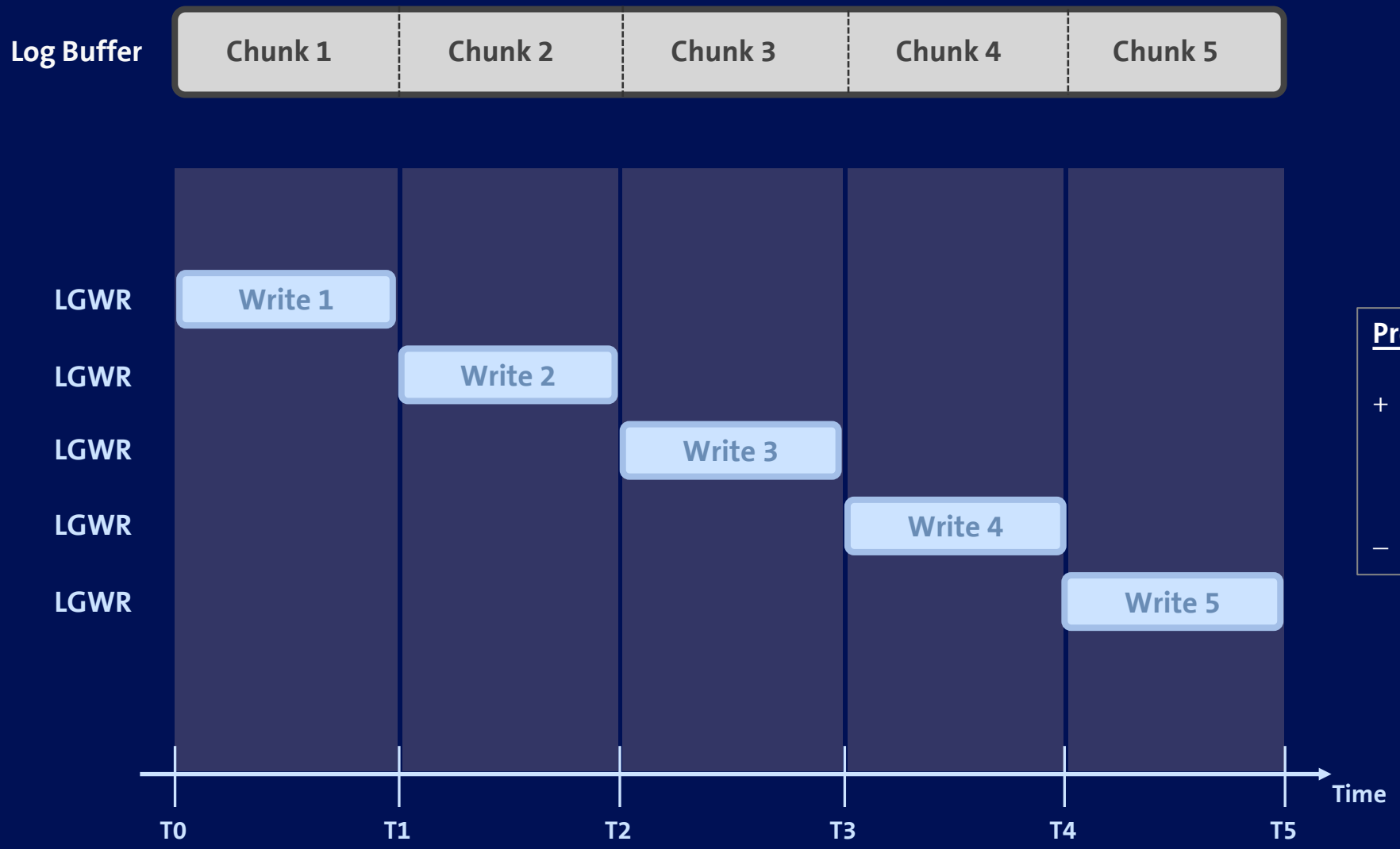
Pros & Cons

\+ Few I/O requests

– Large I/O requests are more susceptible to response time outliers than smaller requests

– Responsiveness

Single LGWR Write Patterns – Serial Writes

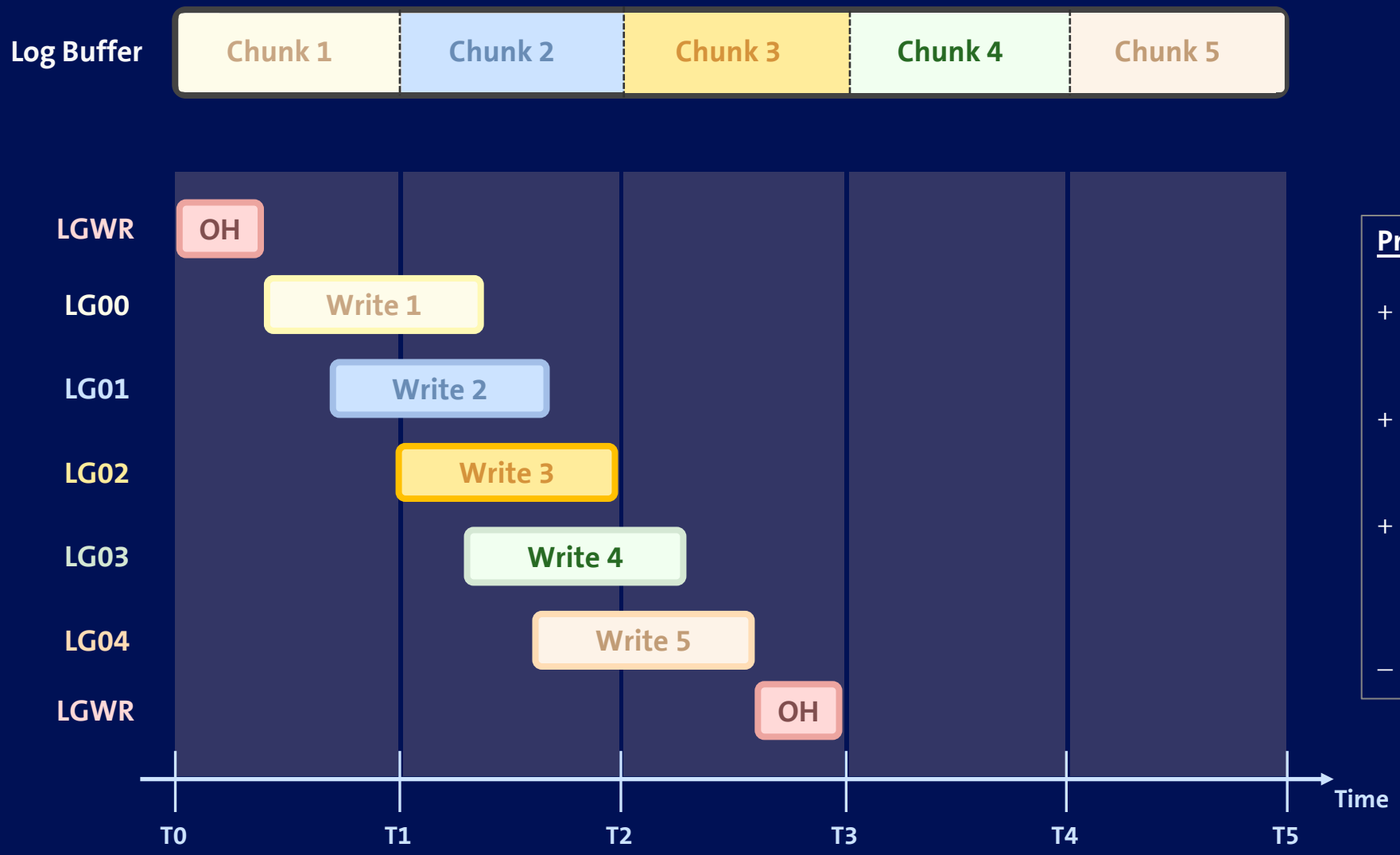


Pros & Cons

\+ Small I/O requests are less susceptible to write response time outliers than large requests

– Responsiveness

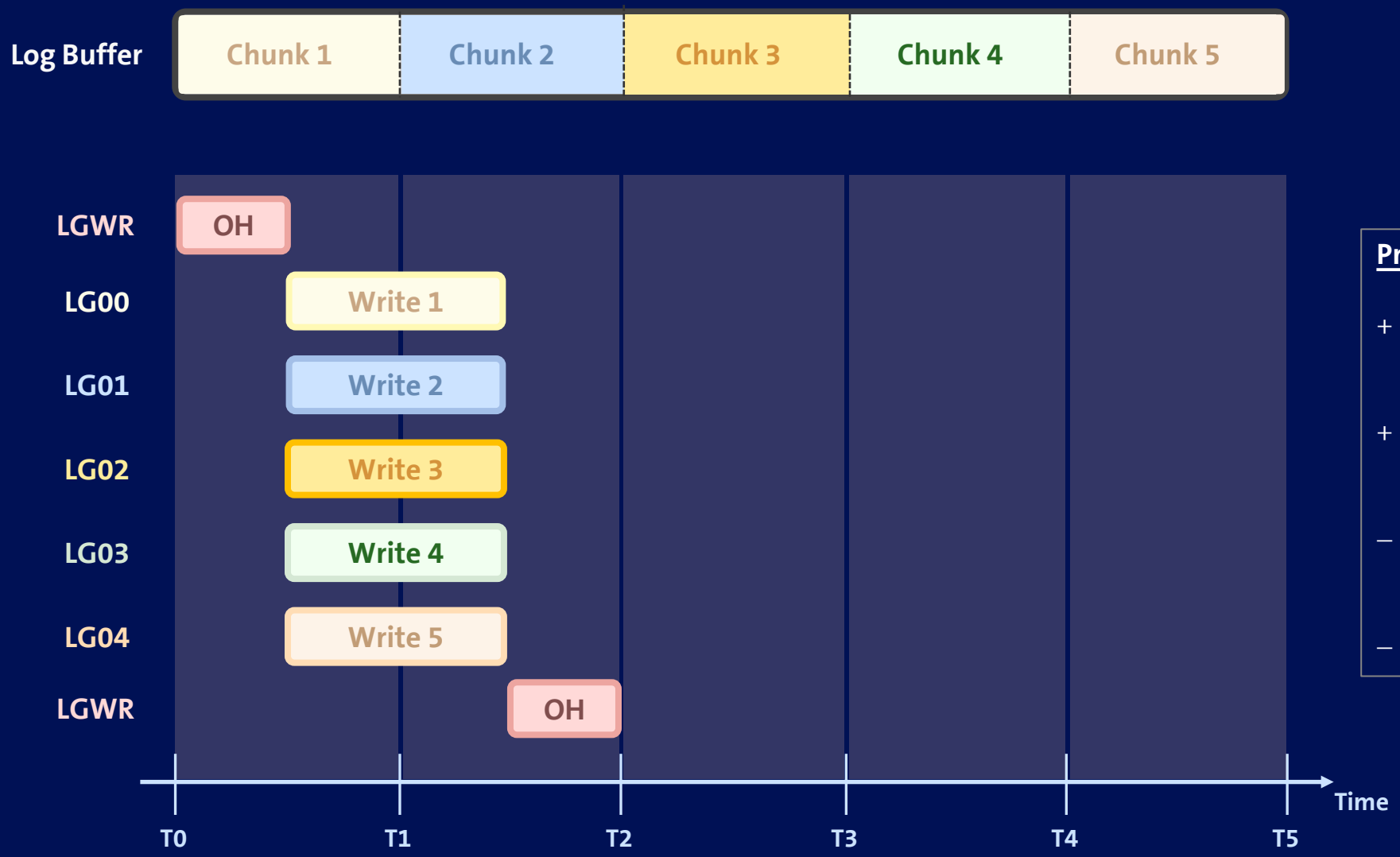
Scalable LGWR Write Patterns – Pipelined (Overlapped) Writes



Pros & Cons

+ Higher write throughput than serial requests
+ Faster write completion than serial requests
+ Responsiveness (provided a worker is immediately available when needed)
– Coordination overhead

Scalable LGWR Write Patterns – Parallel Write



Pros & Cons

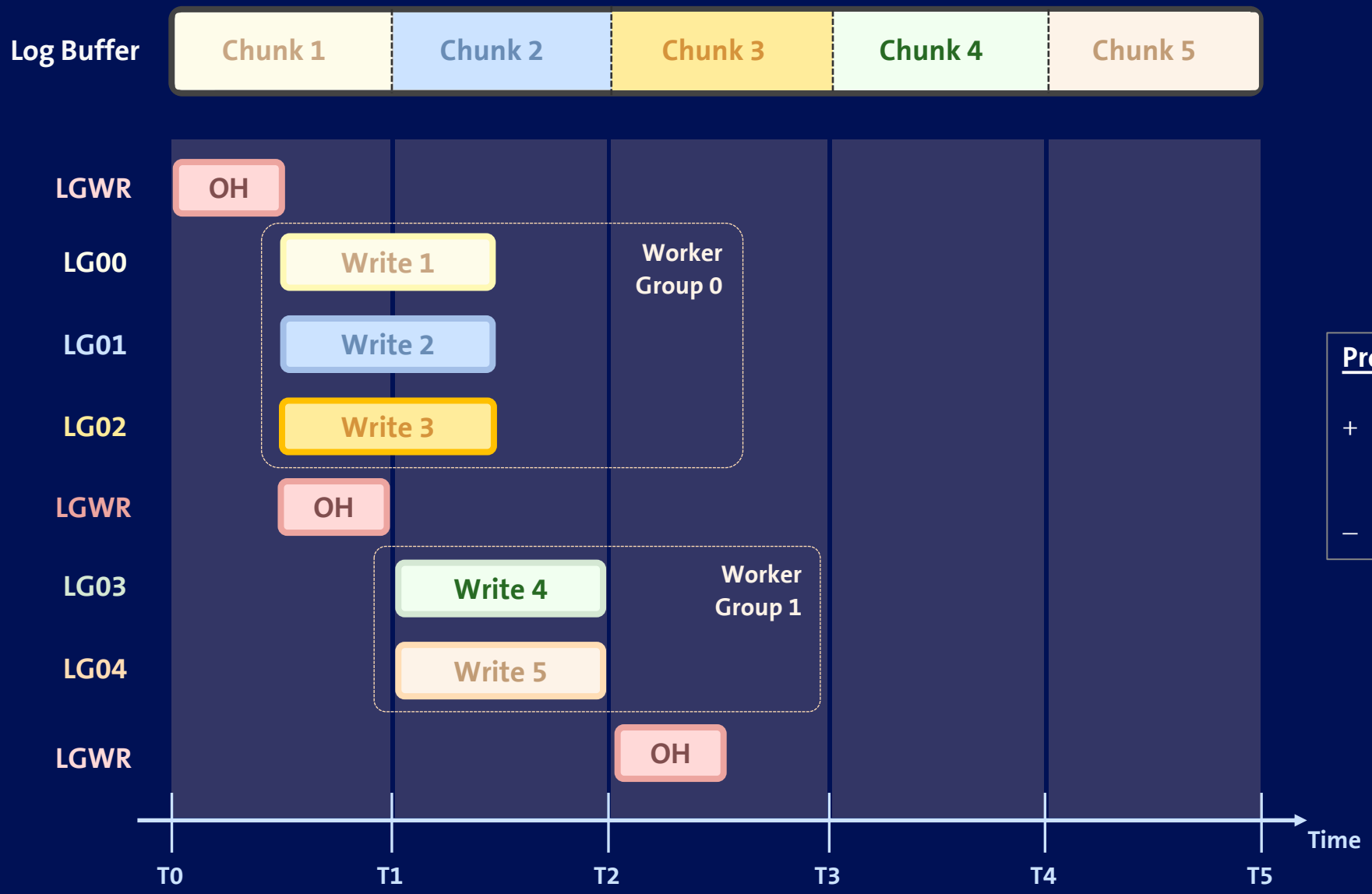
\+ Higher write throughput than serial requests

\+ Faster write completion than serial requests

– Responsiveness (in situations where all workers busy)

– Coordination overhead

Scalable LGWR Write Patterns – Pipelined Parallel Writes

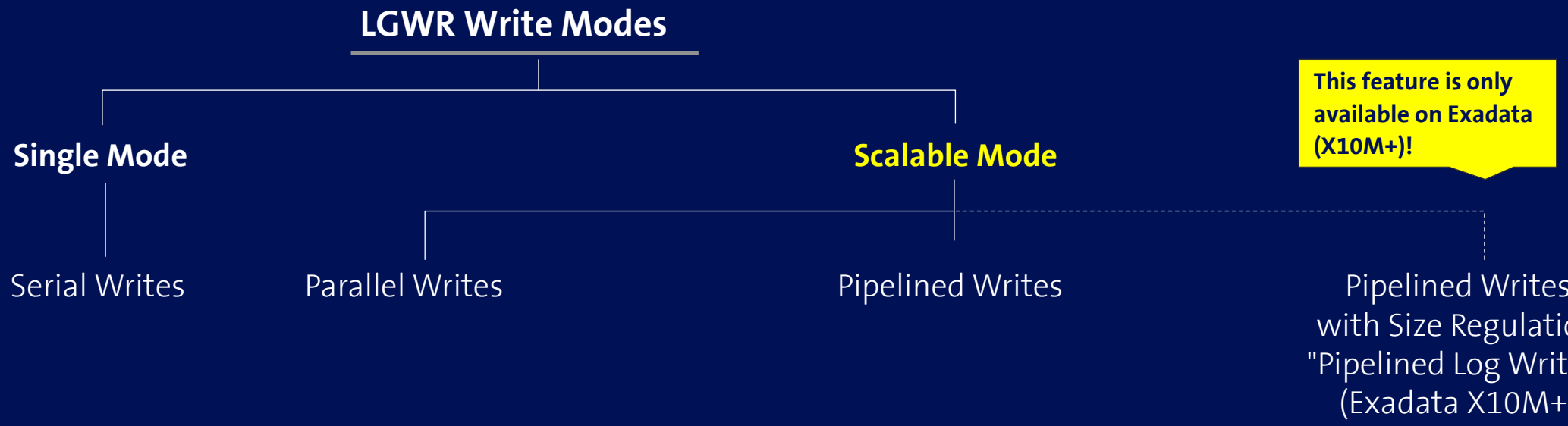


Pros & Cons

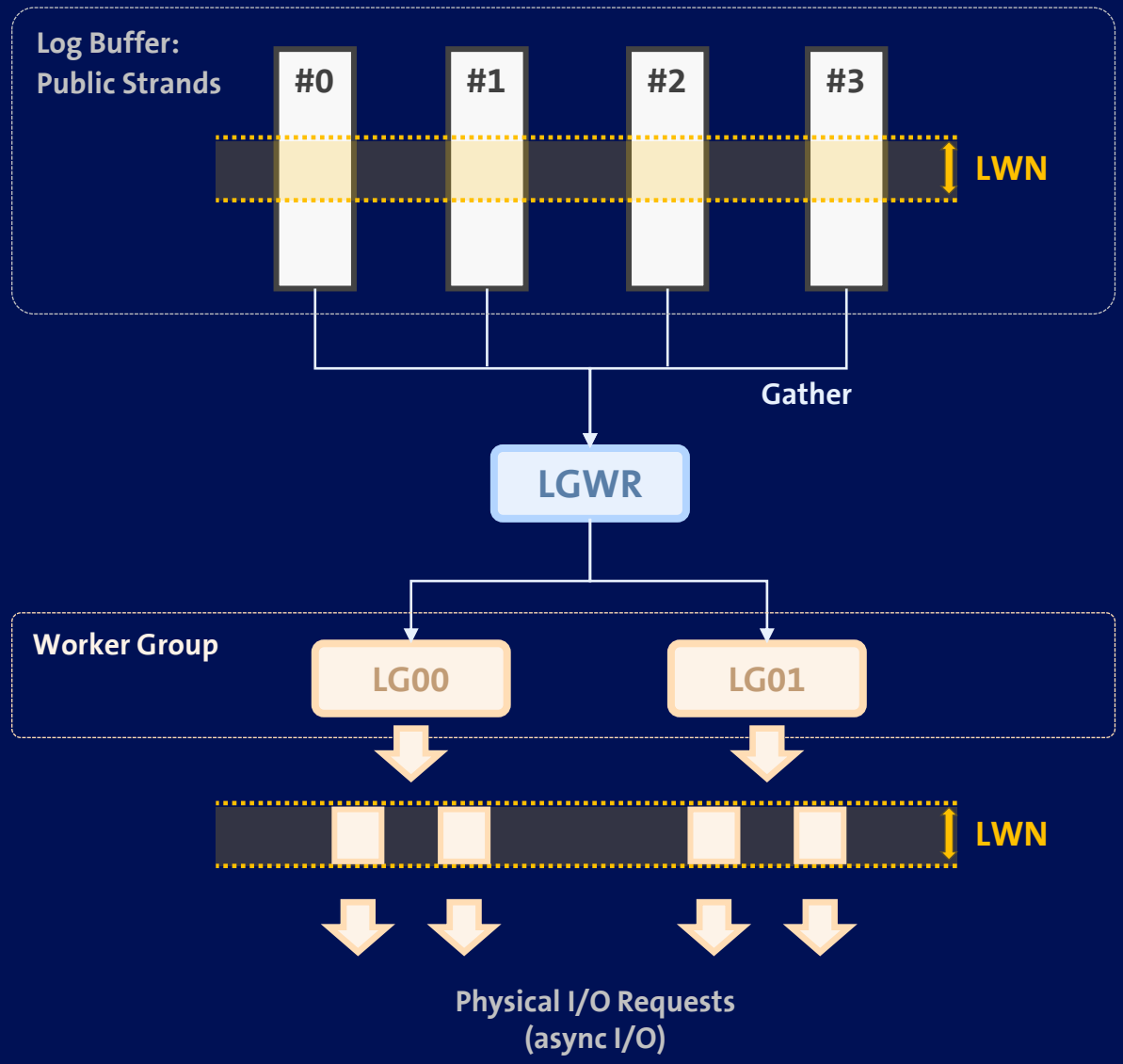
\+ Faster write completion than serial requests

– Coordination overhead

Adaptive Scalable LGWR – Write Modes



Adaptive Scalable LGWR – Parallel Write



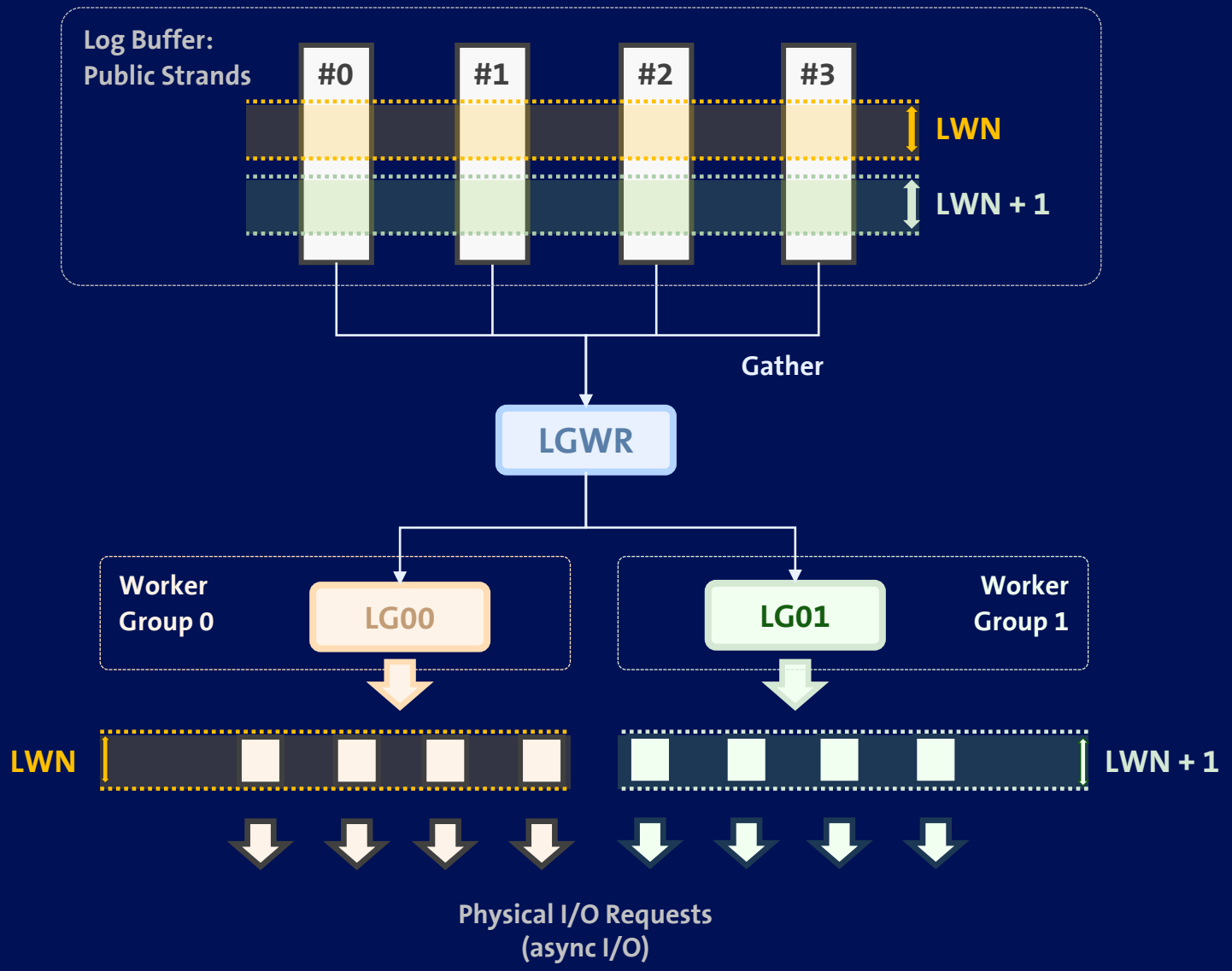
Think of the **LWN** as a "logical" write request, which can be split into one or more physical writes, handled by one or multiple LGWR workers.

Parallel Write
Multiple LG workers process a single logical redo write (LWN) in parallel.

Public Redo Strands:
LGWR assigns one or more public redo strands to each LG worker process.

Physical I/O Requests:
The LG workers issue at least one physical I/O request per strand assigned to them.

Adaptive Scalable LGWR – Pipelined Write

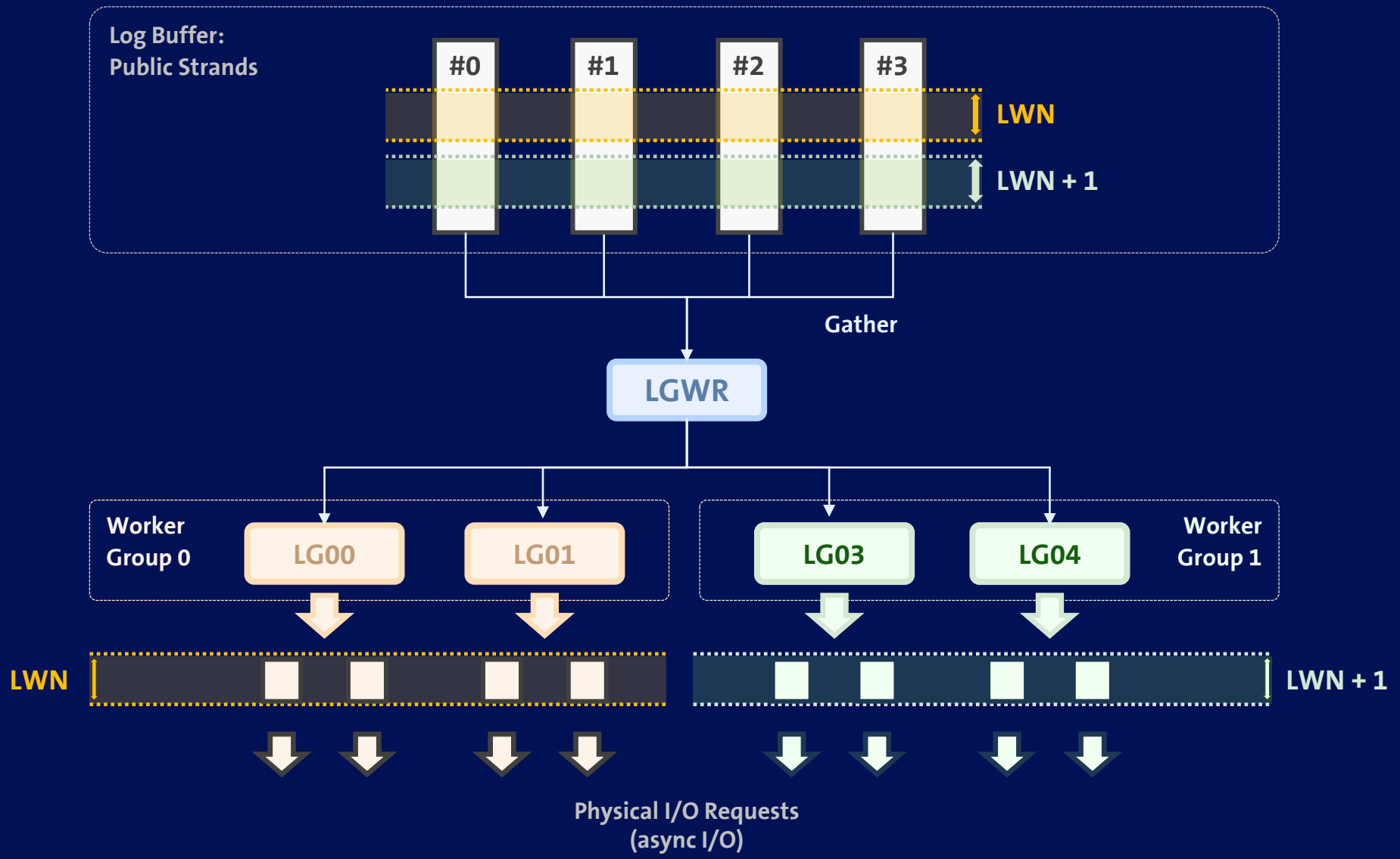


Pipelined Write
Multiple LG workers process different redo writes (LWNs) concurrently.

Public Redo Strands:
LGWR assigns one or more public redo strands to each LG worker process.

Physical I/O Requests:
The LG workers issue at least one physical I/O request per strand assigned to them.

Adaptive Scalable LGWR – Pipelined Parallel Write



Pipelined Parallel Write
Multiple LG workers process different parallel redo writes (LWNs) concurrently.

Public Redo Strands:
LGWR assigns one or more public redo strands to each LG worker process.

Physical I/O Requests:
The LG workers issue at least one physical I/O request per strand assigned to them.

Adaptive Scalable LGWR – Considerations

1. When does LGWR switch from **Single to Scalable** mode?
2. When does LGWR switch back from **Scalable to Single** mode?
3. How does LGWR prevent overly **frequent mode switching**?
4. When does LGWR **pipeline (overlap) and parallelize** redo write requests in Scalable mode?

When does LGWR switch from Single to Scalable mode?

redorate > switch * 2

redorate
Moving average of the system statistic "redo size" computed by CKPT every 3 sec.

switch
The value of redorate at the time when LGWR last switched back from Scalable to Single mode.

Redorate Moving Average

Redorate Moving Avg

```
delta = curr - prev                 # "redo size" delta since last check (3 sec)

deviation =  delta - redorate       # deviation from the redorate average (can be negative)

redorate += deviation * 2/129       # new redorate average adjusted by smoothing factor
```

Smoothing Factor: 2/129 ≈ 0.0155 (~ 1.55%)

Determines the weight of the new deviation in the updated average, smoothing out short-term fluctuations.

Special Case – Exadata X8M and X9M using PMEM Log

redorate > switch * 4

redorate
Moving average of the system statistic "redo size" computed by CKPT every 3 sec.

switch
The value of redorate at the time when LGWR last switched back from Scalable to Single mode.

Adaptive Scalable LGWR – Switch Threshold: Single->Scalable

Switch Threshold

adj = max_log_write_parallelism == 1 ? 2 : 1

redorate > (switch * enable_worker_threshold/100 * adj)

redorate
Moving average of the system statistic "redo size", computed by CKPT every 3 sec (in kcrfw_slave_adaptive_cronjob).

switch
In steady state, switch is the value of the "redorate" at the time when LGWR last switched from scalable to single mode.

enable_worker_threshold
The value of parameter _adaptive_scalable_log_writer_enable_worker_threshold (default: 200)

Note:
The Fast Log File Sync mechanism on Exadata X8M and X9M automatically sets _max_log_write_parallelism to 1 at instance startup, which doubles the threshold defined by _adaptive_scalable_log_writer_enable_worker_thresh!

Adaptive Scalable LGWR – Switch Threshold: Single->Scalable

Note: On Exadata X8M and X9M, these redorate and switch values would not trigger a mode switch!

LGWR Trace

```
kcrfw_slave_adaptive_updatemode: single->scalable redorate=29381531 switch=14315077

*** 2025-07-04T00:10:00.147531+02:00 (CDB$ROOT(1))
Adaptive scalable LGWR enabling workers
```

Example Calculation:

redorate > (switch * 200 / 100) =

redorate > (switch * 2) =

29381531 > (14315077 * 2) =

29381531 > 28630154

=> Enable workers!

When does LGWR switch back from Scalable to Single mode?

single <= scalable

single
Redo write time with a single LGWR process.

scalable
Estimated (modelled) redo write time with writes distributed across multiple LG workers, adjusted for pipelining efficiencies.

Switch if a single LGWR writes faster than multiple LG workers.

Adaptive Scalable LGWR – Switch Threshold: Scalable -> Single

LGWR uses the effective write time when log writes can be parallelized; otherwise, it relies on the average redo write time

_max_log_write_parallelism = 1

singlerw = rw_avg - slave_delay_avg

_write_parallelism > 1

singlerw = rw

single = _max_outstanding_log_writes * singlerw

single represents the time to complete all outstanding log writes using a single LGWR

Adaptive Scalable LGWR – _max_log_write_parallelism on Exadata X8M and X9M

```
# BPFTRACE_CACHE_USER_SYMBOLS=1 ./kcrfw_dax_is_smart_log-max-lwp.bt <ORACLE_SID>

Attaching 6 probes...

Tracing ... Hit Ctrl + C to stop.

301172 ora_lgwr_tctest -> kcrf_determine_storage: 0x600218c8, 0x1

301172 ora_lgwr_tctest -> kcrfw_dax_is_smart_log: 0x7ffc9c8dc8b8
301172 ora_lgwr_tctest    max_log_write_parallelism: start=4, end=1
301172 ora_lgwr_tctest <- kcrfw_dax_is_smart_log: 1

301172 ora_lgwr_tctest <- kcrf_determine_storage: 0x0 (0)
```

With the PMEM log enabled, kcrfw_dax_is_smart_log fixes the max_log_write_parallelism to 1!

Script source: Christoph Lutz, oracle Github Repository, kcrfw_dax_is_smart_log.bt

Adaptive Scalable LGWR – Redo Write Time Moving Average

CKPT computes the moving average every 3 sec (kcrfw_slave_adaptive_cronjob).

Redo Write Time Moving Avg

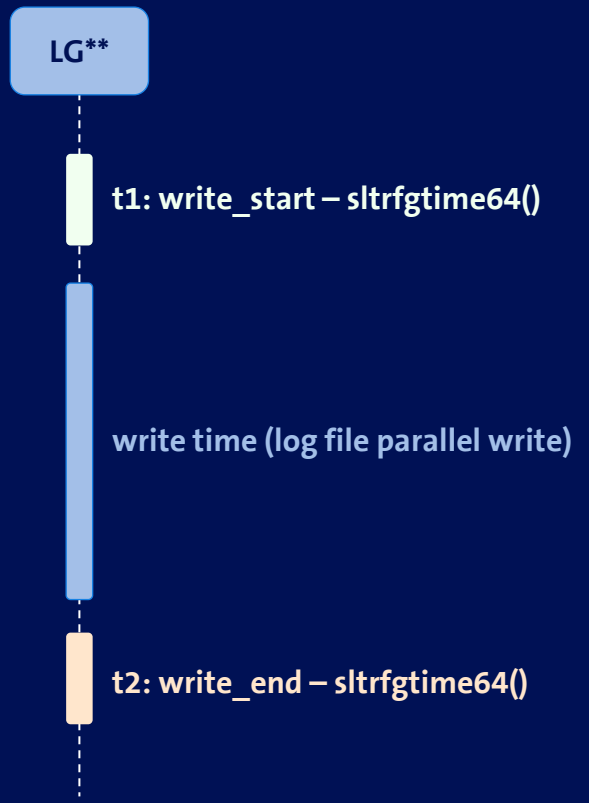
write_time = t2 - t1

deviation = write_time - rw_avg

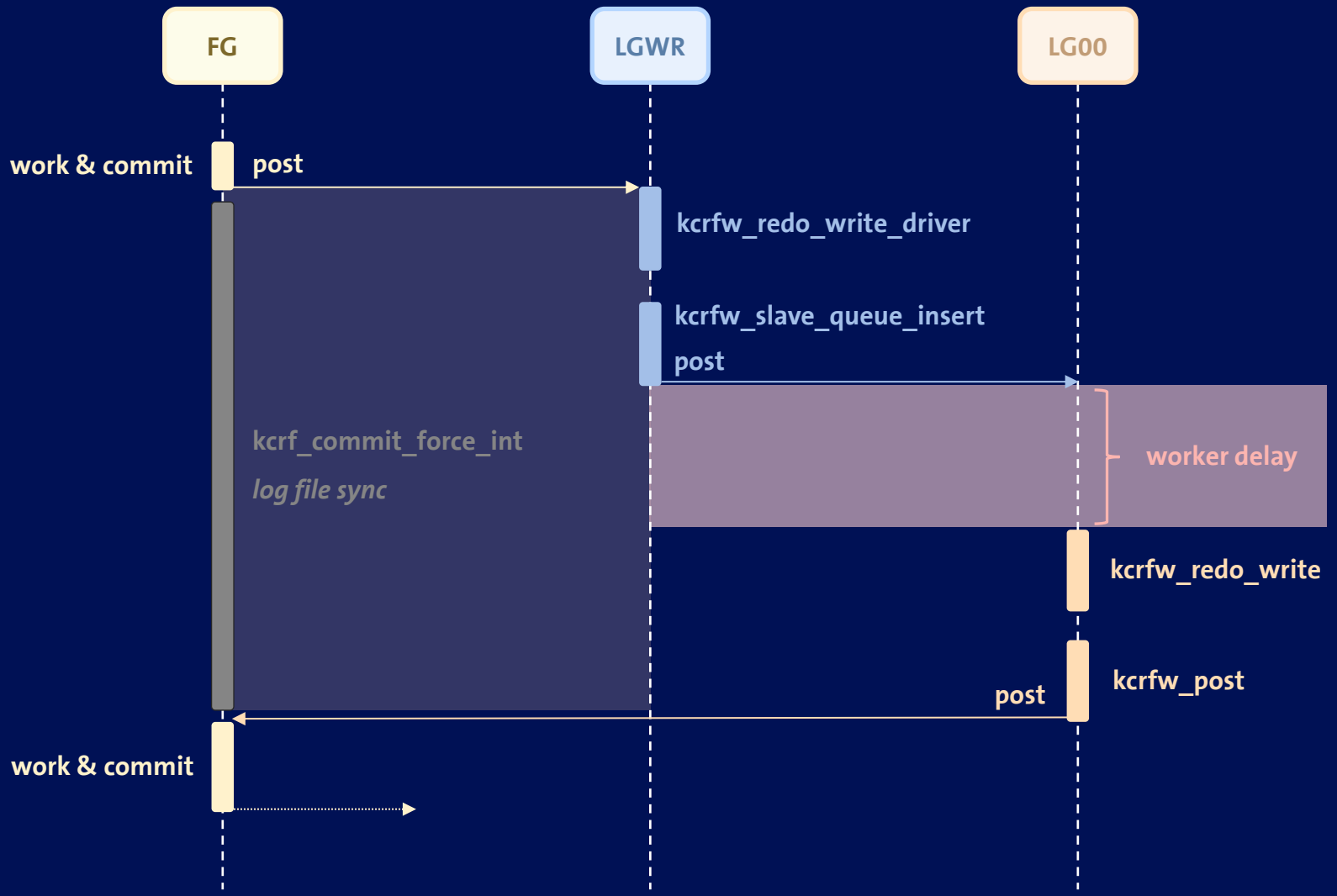
rw_avg += deviation * 2/129

Smoothing Factor: 2/129 ≈ 0.0155 (~ 1.55%)

Determines the weight of the new deviation in the updated average, smoothing out short-term fluctuations.



Adaptive Scalable LGWR – Redo Write Worker Delay



Adaptive Scalable LGWR – Redo Write Worker Delay Moving Average

CKPT computes the moving average every 3 sec (kcrfw_slave_adaptive_cronjob).

Worker Delay Moving Average

```
delta = curr - prev                 # delta "redo write worker delay (usec)" since last check (3 sec)

deviation =  delta - delay          # deviation from the average delay (can be negative)

delay += deviation * 2/129          # new average delay adjusted by smoothing factor
```

Smoothing Factor: 2/129 ≈ 0.0155 (~ 1.55%)

Determines the weight of the new deviation in the updated average, smoothing out short-term fluctuations.

Adaptive Scalable LGWR – Switch Threshold: Scalable -> Single

1. **Redo write time assuming sequential redo writes (using the average redo write time)**

   scalable_nopipe = max_outstanding_log_writes * rw_avg

2. **Redo write time with pipelined writes (using 50% pipelined writes by default)**

   scalable_pipe = rw_avg + rw_avg x (max_outstanding_log_writes - 1) x (100 - disable_worker_thresh) / 100

3. **Interpolate sequential and pipelined redo write times, considering the time saved by pipelined writes**

$$\text{scalable} = \text{scalable\_nopipe} + \frac{(\text{all} - \text{group0})}{\text{group 0}} \times (\text{scalable\_pipe} - \text{scalable\_nopipe})$$

Adaptive Scalable LGWR – Switch Threshold: Scalable -> Single (Summary)

Single LGWR

rw = (max_log_write_parallelism > 1) ? rw : rw_avg - slave_delay_avg

single = max_outstanding_log_writes * rw

Multiple LG Worker Processes

scalable_nopipe = max_outstanding_log_writes * rw_avg

scalable_pipe = rw_avg + rw_avg * (max_outstanding_log_writes - 1) * (100 - disable_worker_thresh) / 100

scalable = scalable_nopipe + ((all - group0) / group0) * (scalable_pipe - scalable_nopipe)

Adaptive Scalable LGWR – Switch Threshold: Scalable -> Single

rw
Current redo write time.

rw_avg
Redo write time moving average.

all
Total number of redo writes across all groups.

group0
Number of redo writes in group 0.

max_outstanding_log_writes
Maximum number of outstanding redo log writes (parameter _max_outstanding_log_writes).

slave_delay_avg
Time delay between LGWR preparing a redo I/O request for a log writer worker and the moment the worker actually issues the I/O.

disable_worker_thresh
Value of parameter _adaptive_scalable_log_writer _disable_worker_threshold (default: 50).

max_log_write_parallelism
Maximum parallelism within a log write (parameter _max_log_write_parallelism).

The parallelism can change at runtime (on Exadata with PMEM log, Fast Sync sets this to a fixed value of 1 in kcrfw_dax_is_smart_log).

Adaptive Scalable LGWR – Switch Threshold: Scalable->Single

LGWR Trace

```
kcrfw_slave_adaptive_updatemode: scalable->single group0=13468210 all=15882687
rw=1700 single=1722 scalable_nopipe=3400 scalable_pipe=2550 scalable=3247

*** 2025-07-17T14:34:21.003313+02:00 (CDB$ROOT(1))
Adaptive scalable LGWR disabling workers
```

Example

scalable =
scalable_nopipe +
((all - group0) / group0) *
(scalable_pipe - scalable_nopipe) =

3400 +
((15882687 - 13468210) / 13468210) *
(2550 - 3400) =

3400 + 0.17927 * (-850) = 3247.6

single <= scalable

1722 <= 3247

=> Disable workers!

Adaptive Scalable LGWR – Mode Switch Threshold Aging

The switch value is updated when a mode switch evaluation is performed (s. next slide).

switch = (switch * worker_aging / 1000000)

worker_aging
The value of parameter _adaptive_scalable_log_writer_enable_worker_aging (default: 999900)

Worker Aging decreases ("ages") the switch threshold by 0.01%

Adaptive Scalable LGWR – Mode Switch Frequency Guardrails

Oracle uses safety mechanisms to prevent overly frequent mode transitions:

With these defaults, a mode switch can occur at most once every 9 sec!

```
sampling_time  > 3      &&
sampling_count > 128    &&
arbiter == 0
```

A mode switch is evaluated only if:

- **more than 3 seconds have passed since last evaluation**
- **more than 128 redo writes since last evaluation**
- **the arbiter reaches 0**

sampling_time (default: 3 sec)
Minimum time in sec before a mode switch is evaluated again
(defined by initialization parameter _adaptive_scalable_log_writer_evaluation_interval).

sampling_count (default: 128)
Minimum number of redo writes before a mode switch is evaluated again
(defined by initialization parameter_adaptive_scalable_log_writer_sampling_count).

arbiter (default 3)
Every time the switch threshold is exceeded during evaluation, the arbiter is decremented by one.
A mode switch is only performed when the arbiter reaches 0 (the value of 3 is hard-coded in kcrfw_slave_adaptive_updatemode).

Adaptive Scalable LGWR – Redo Write Parallelization

In Adaptive Scalable mode, LGWR can assign multiple LG workers to process a redo write in parallel.

Preconditions:

- redo write size > _max_io_size (usually 1 MB or 2048 redo blocks of 512 bytes)
- Number of active public redo strands > 1
- _max_log_write_parallelism > 1

On Exadata X8M and X9M, _max_log_write_parallelism is set to a fixed value of 1, which disables parallel redo writes!

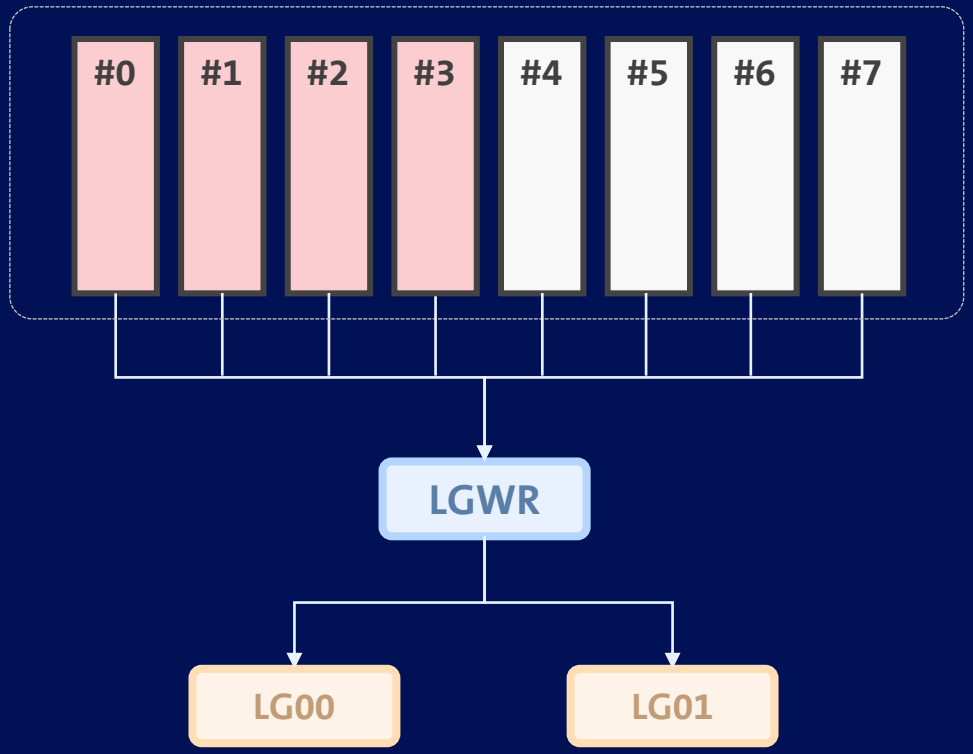
Formula:

strands_per_worker = ceil(max_strands / max_log_write_parallelism)

active_strands = highest_active_strand + 1

workers = ceil(active_strands/ strands_per_worker)

Adaptive Scalable LGWR – Workers per Parallel Write: 8 Strands Example



Example

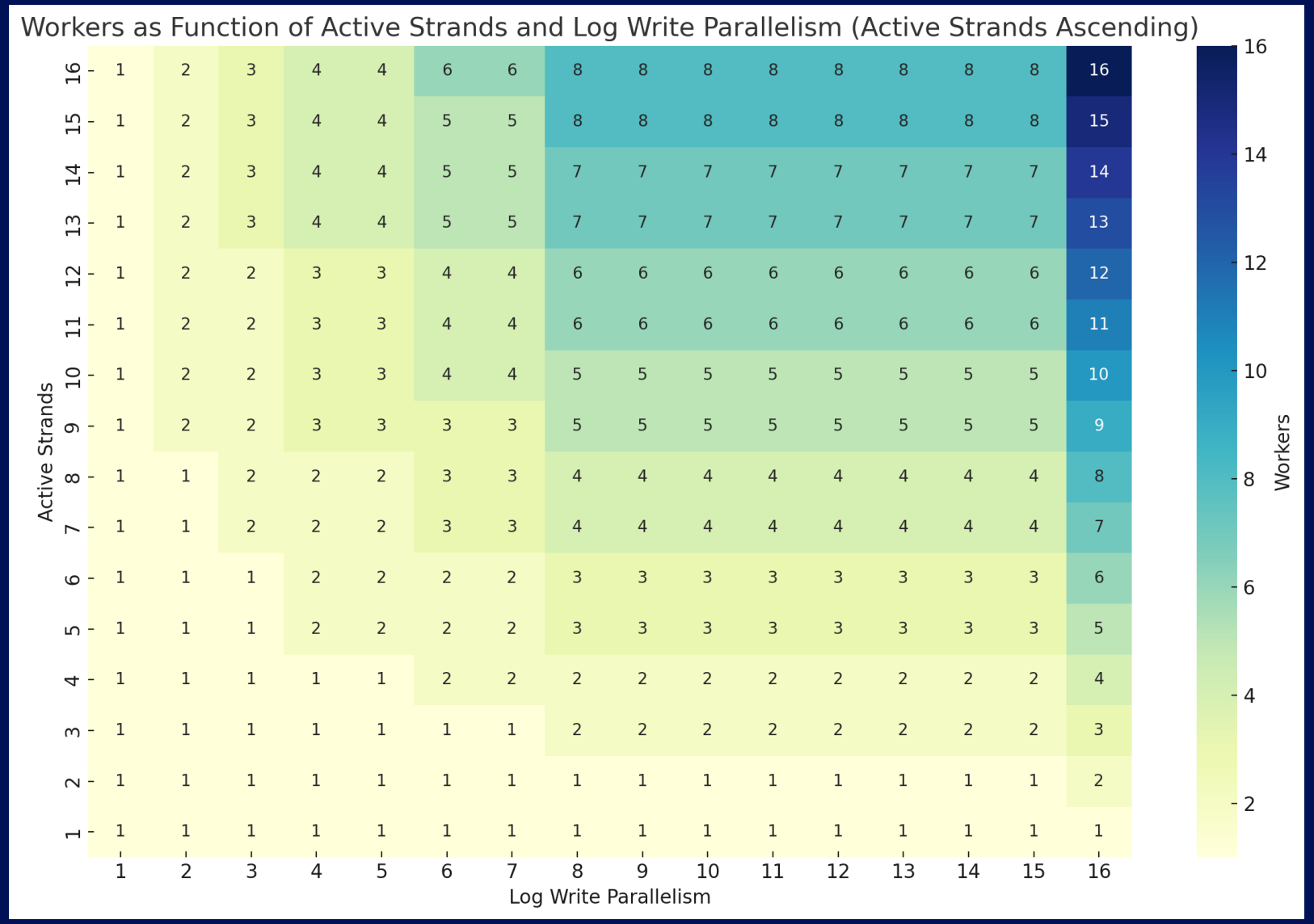
max_strands = 8
max_log_write_parallelism = 4
highest_active_strand = 3

strands_per_worker = ceil(max_strands / max_log_write_parallelism) = ceil(8 / 4) = 2

active_strands = highest_active_strand + 1 = 3 + 1 = 4

workers = ceil(active_strands / strands_per_worker) = (4 / 2) = 2

Adaptive Scalable LGWR – Workers per Parallel Write: 16 Strands Example



With _max_log_write_ parallelism set to 1, LGWR will not parallelize redo writes (kinda obvious ☺)!

Adaptive Scalable LGWR – Sysstat Counters

redo writes adaptive worker:

Total number of redo writes in Adaptive Scalable mode.

redo writes adaptive all:

Total number of redo writes in Single + Adaptive Scalable mode.

Note:

The difference between "redo writes adaptive all" and "redo writes adaptive worker" represents the redo writes issued in Single mode.

Adaptive Scalable LGWR – Sysstat Counters

redo writes (group n):

Total number of redo writes in group <n> (where n = 0 - 7).

redo blocks written (group n)

Total number of redo blocks written by group <n> (where n = 0 - 7).

Adaptive Scalable LGWR – Sysstat Counters

redo write worker delay (usec):

The "worker delay" is the time between LGWR preparing a redo write for an LG worker and the moment the worker schedules the I/O.

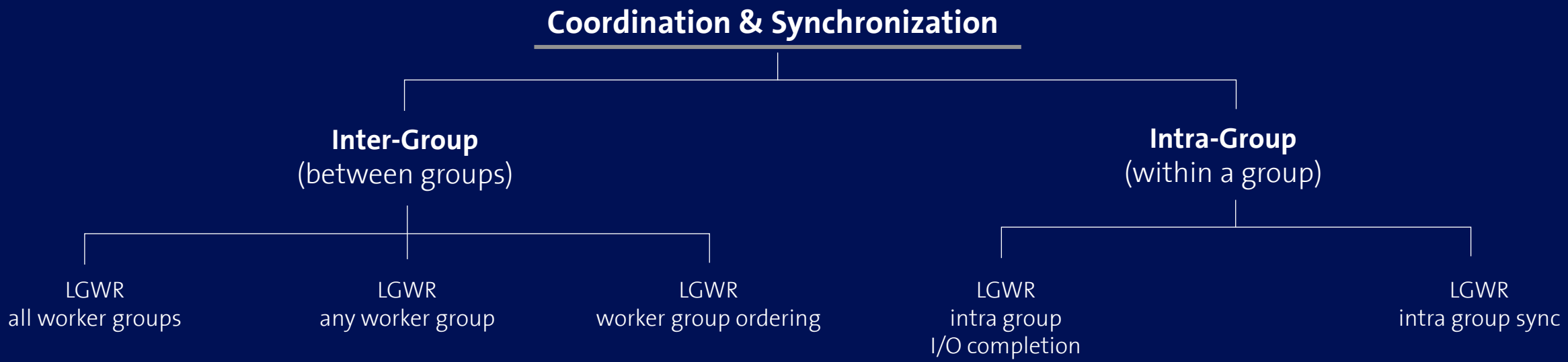
This counter tracks the total delay time (in usec).

redo write worker delay count

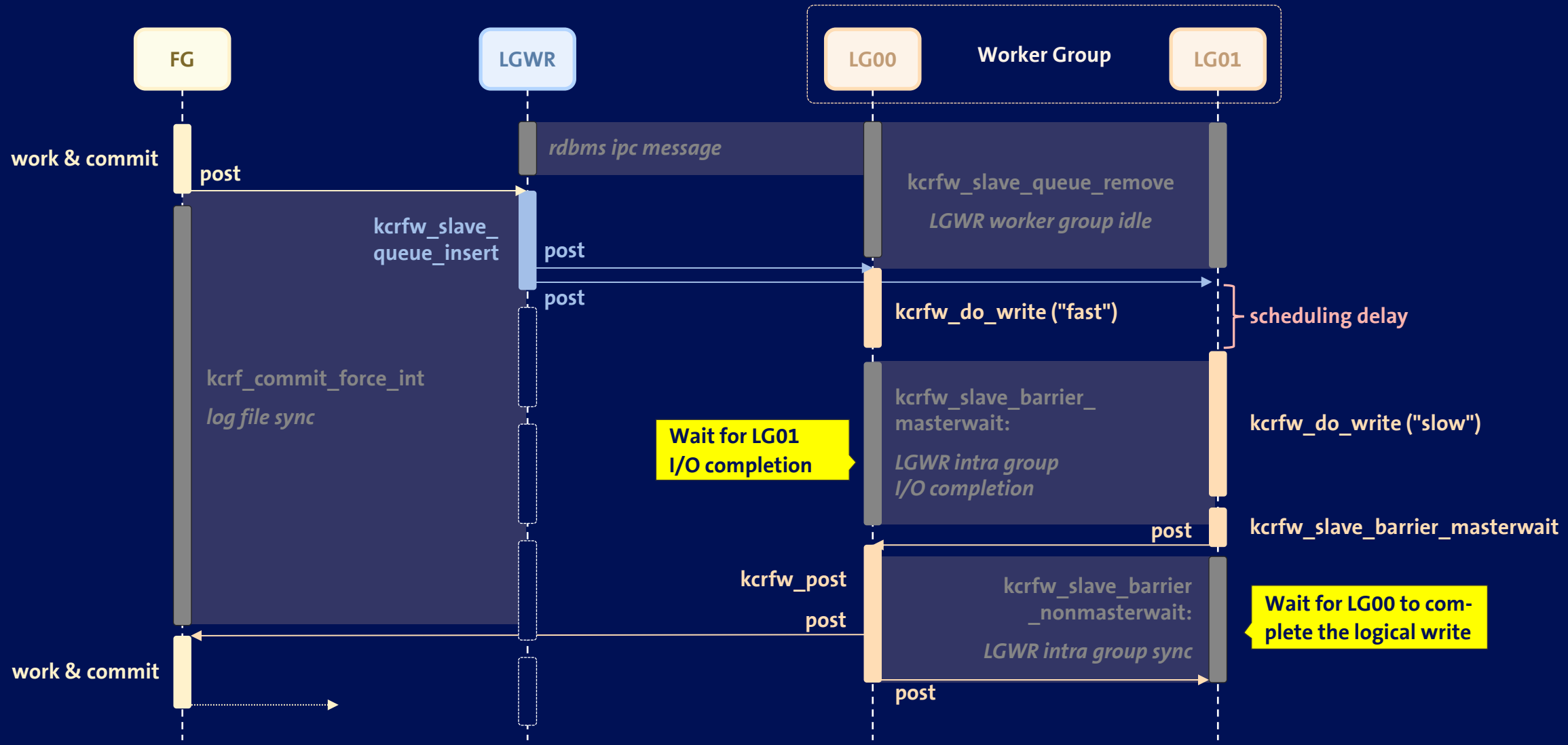
This counter is incremented every time a LG worker process is posted by LGWR to perform a redo write

This counter correlates with the counter "redo writes adaptive worker".

Adaptive Scalable LGWR – Worker Coordination & Synchronization



Adaptive Scalable LGWR – Parallel Write



Adaptive Scalable LGWR – Drawback: Heuristics

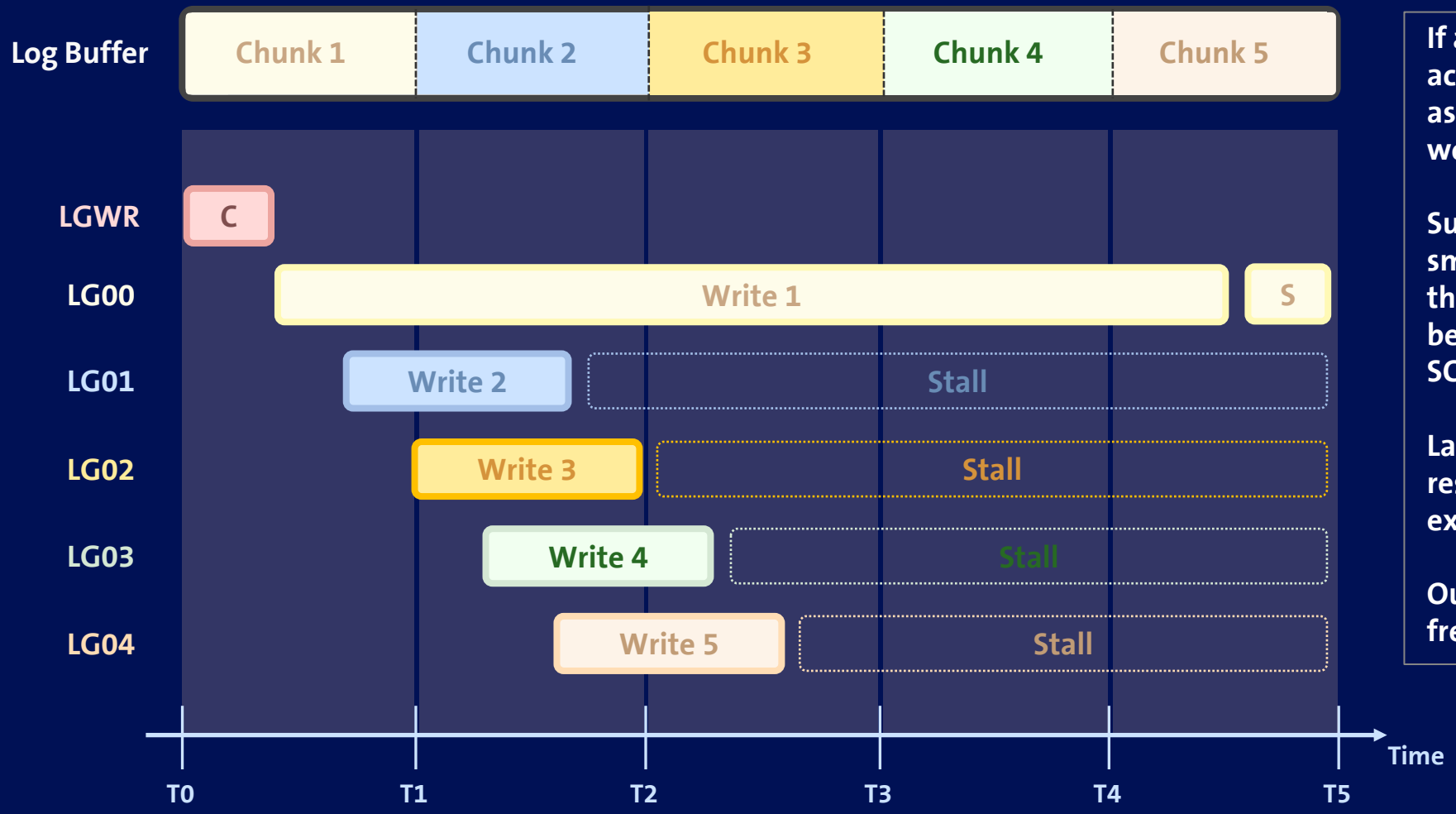
The LGWR decision making is **based on heuristics**.

The heuristics are based on factors that **do not always reflect current conditions**:

- Historical switch sample as a reference point to switch from single to scalable mode
- Assume 50% overlapped redo writes in the decision to switch back to single mode

The heuristics **react slowly to changes** in workload!

Adaptive Scalable LGWR – Drawback: Pipeline Stalls



If a large amount of redo data accumulates in the log buffer, LGWR assigns a large logical write to a worker.

Subsequent redo writes may be much smaller and therefore complete before the large write – the pipeline stalls because Oracle can't advance the on-disk SCN unless the large write completes !

Large writes are susceptible to response time outliers, further exacerbating the problem!

Outlier redo writes tend to increase in frequency with larger redo writes.

Exadata Only – Pipelined Log Writes (X10M and Newer Models)

Pipelined Log Writes allow a new logical write only if:

1. No logical write currently in progress
2. New logical write has reached a minimum size threshold
3. A timeout is reached while waiting for the logical write to become larger

LGWR spins on CPU until either the write size or timeout threshold has been reached!

LGWR dynamically regulates the write size at runtime!

Adaptive Scalable LGWR & Other Features – SYNC Mode Data Guard

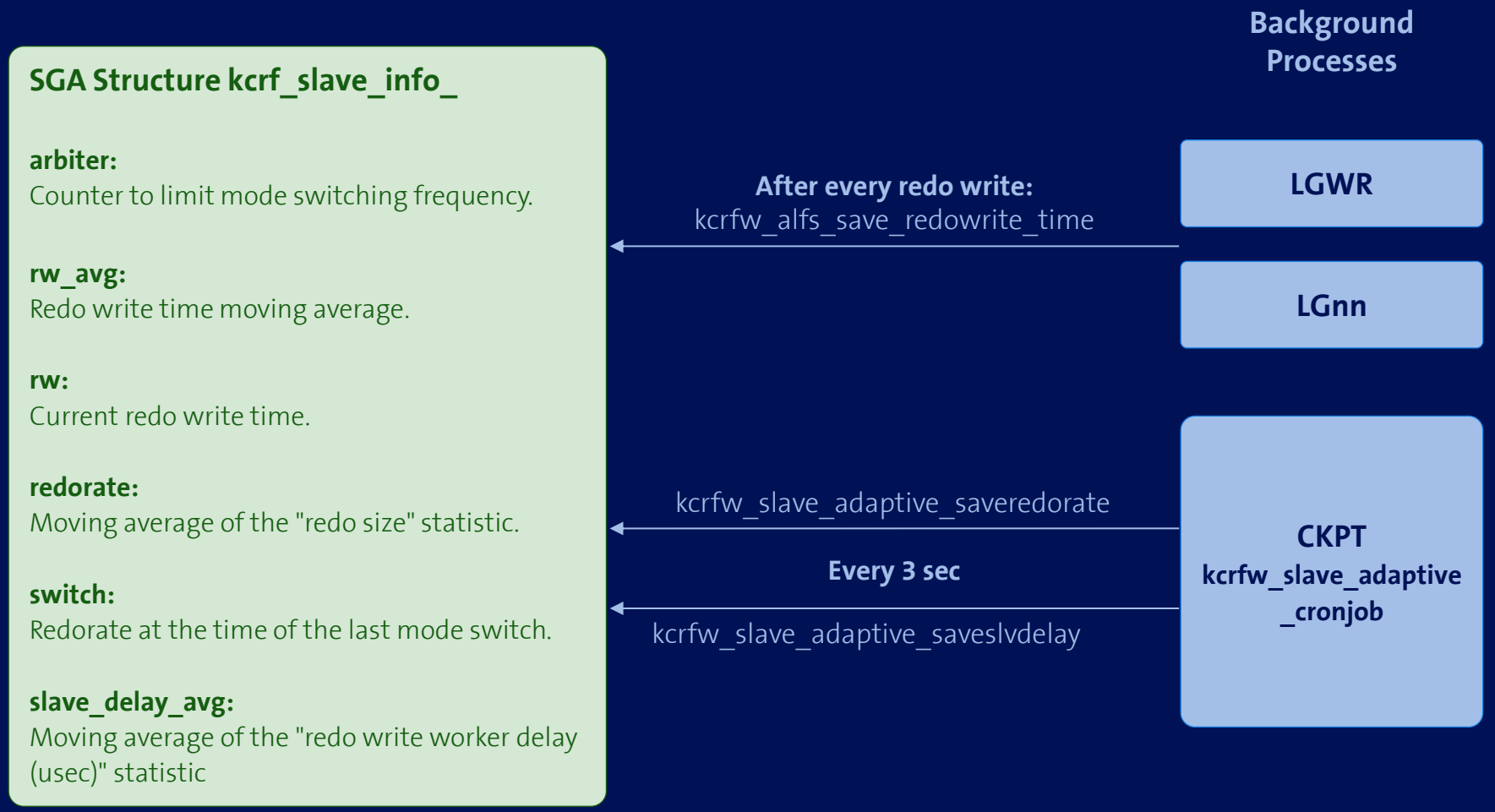
When SYNC LogXptMode is enabled, all LG workers are put into "standby mode".

Standby mode is indicated by a flag in the SGA structure kcrf_slave_info_.

This means, the LGnn processes keep on running, but LGWR doesn't send them any redo write requests as long as the "standby mode" flag is set.

The "standby mode" flag is toggled on an off in function kcrfw_slave_pool_setstandbymode.

Adaptive Scalable LGWR – Infrastructure



Demo & LGWR Hacking – If Time Permits :-)

```
# gdb -q -x ./lgwr-control.gdb -p <pid>

(gdb) show_lgwr_mode

----- Show LGWR mode -----
  lgwr mode is                  : serial
  adaptive scalable mode is     : heuristic
  alwe is                       : disabled
  olrw is                       : disabled (mode: thin)
  kcrfwslv arbiter is           : 3
  slave pool stdby mode is      : disabled
  fast sync is                  : enabled
  alfs polling mode is          : disabled
  alfs arbiter is               : 3
  max log write parallelism is  : 1
  nr of active redo strands is  : 1
  max nr of redo strands is     : 16
  cpu_count (startup) is        : 16
--------------------------
```

Script source: Christoph Lutz, oracle Github Repository, lgwr-control.gdb

Thank you for attending!

SYMPOSIUM42

Thank you for attending!

Questions, feedback, comments?
I look forward to hearing from you!

https://www.0xchris.dev

christoph.lutz@swisscom.com

@chris_skyflier

@christophlutz.bsky.social

Christoph-Lutz

References (1/2)

Chi Cao Minh et al., United States Patent 9418129B2, Adaptive High Performance Database Redo Log Synchronization, 2016-08-16

Chi Cao Minh et al., United States Patent 10706009B2, Techniques to parallelize CPU and IO work of log writes, 2020-07-07

Christoph Lutz, oracle Github Repository, July 2025

Christian Craft, Persistent Memory Primer, 2020-05-20

Christian Craft, Persistent Memory in Exadata X8M, 2020-05-21

Frits Hoogland, A look into Oracle redo, part 4: the log writer null write, 2018-02-20

Graham Ivy et al., United States Patent 12346311B2, Adaptively overlapping the writing of redo log records, 2025-07-01

My Oracle Support, Bug 14823372 - Adaptive "log file sync" picks inaccurate polling interval on RAC (Doc ID 14823372.8), 2022-03-13

Nikolay Savvinov, High log file sync waits? Check log parallelism!, 2014-19-22

Oracle Corp., Database 19c, Database Concepts, E96138-09, July 2025

Oracle Corp., Oracle® Exadata System Software, User's Guide 25.2, F29254-45, July 2025

Oracle Corp., Oracle 9i Database Reference Release 2 (9.2), Parameter LOG_PARALLELISM, A96536-02

References (2/2)

Oracle Corp., Oracle 19c Database Reference, E96196-38, July 2025

Tanel Poder, Understanding LGWR, Log File Sync Waits, and Commit Performance (via Scribd), undated

Tanel Poder, Testing Oracle's Use of Optane Persistent Memory, Part 1 - Low Latency Commits, 2022-04-28

Tanel Poder, Testing Oracle's Use of Optane Persistent Memory, Part 2 - Fast Log File Sync Waits, 2022-07-15

Tanel Poder, Oracle Performance Troubleshooting Without OS Access, Part 1: Identifying CPU Scheduling Latency, 2020-02-10

Wei Ming Hu et al., United States Patent 7039773B2, Method and Mechanism for Efficient Implementation of Ordered Records, 2006-05-02

Yuhong Zhong, XRP: In-Kernel Storage Functions with eBPF (Youtube Video eBPF Summit 2022), 2022-10-04