ORACLE

NoCOUG Virtual Spring Conference 2020

Maintaining Availability and Restoring Performance After the Glitch Has Gone

Michael Hallas
Real-World Performance Team
Oracle Database Development

ORACLE
REAL-WORLD PERFORMANCE

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

ORACLE

A Confusion of Connections The Problem of Processes Struggling with Sessions

Michael Hallas

Real-World Performance

Who We Are

Part of Oracle Database Development

Team members at HQ and in the USA, Europe and Asia

Over three hundred years of experience combined

What We Do

Use the product as designed

Aim for the best performance

Apply data-driven analysis

Avoid guesswork

Share what we learn

Maintaining Availability and Restoring Performance

Connection Primer

Anatomy of a Glitch

Guidelines for Most Workloads

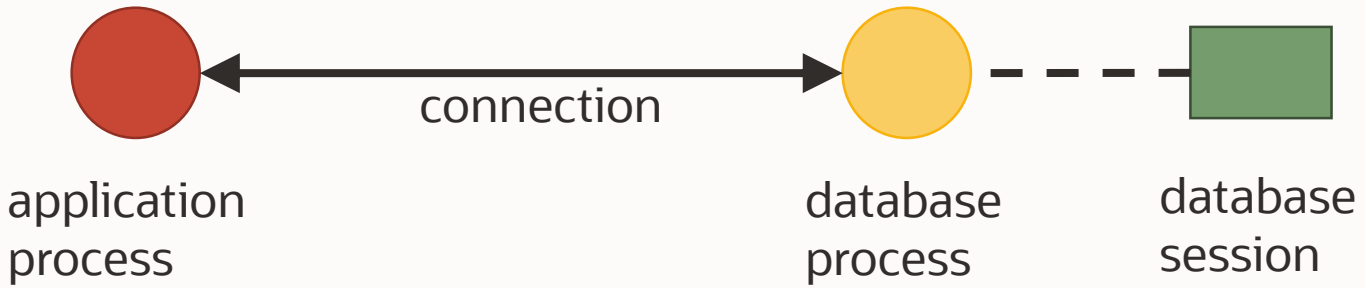
Something New

Connection Primer

Getting work done in Oracle Database requires a **session**

In the simplest case, for example SQL*Plus

- A **connection** is established between my SQL*Plus process and one foreground **process**
- One **session** is authenticated using my credentials
- Away I go



Connection Primer
For OLTP

A foreground process will run on one CPU when there is work to be done

- This works for me
- What about everybody else?

Oracle relies on multiple active foreground processes to exploit multiple CPUs

Connection Primer

QUESTION

So the more connections, the better?

ANSWER

Up to a point

- Most workloads need multiple connections

Not too many

- Wastes resources

Not too few

- May be a bottleneck

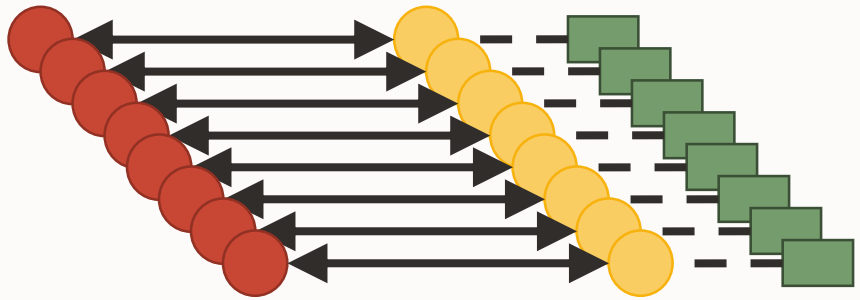
Just right!

Just Right with a Connection Pool?

Start with a minimum number of connections

If the workload increases

- If all existing connections are being used
  - If the number of connections is below the maximum
    - add a connection



If the workload decreases

- If some connections have been idle for some time
  - If the number of connections is above the minimum
    - remove a connection

Feedback loops that are mostly harmless

Connection Primer

QUESTION

My server has lots of memory

My server has lots of CPU

So what's the problem?

ANSWER

In normal circumstances?

- Nothing much

When a glitch occurs

- Glitches may escalate to outages

Anatomy of a Glitch

QUESTION

So what is a glitch?

ANSWER

A glitch is an unexpected transient delay to normal processing

Some possible examples

- Delay in redo log write I/O
- Delay in network write I/O
- Invalidation or Recompilation
- Reconfiguration

Anatomy of a Glitch

QUESTION

Sounds bad. How do I eliminate glitches?

ANSWER

Careful choices of hardware, software, people and processes can reduce the incidence of glitches

Even the best systems suffer from occasional glitches

We call this the real-world!

Anatomy of a Glitch

QUESTION	ANSWER
So what happens during a glitch?	Database work is delayed
Wasn't my connection pool just right?	Work continues to arrive
	Existing connections and sessions become active
	The existing connections in the connection pool will soon be exhausted
	The number of connections rises rapidly

Avoiding a Thousand Words

Videos from Real-World Performance

- Large Dynamic Connection Pools - Part 1
- Large Dynamic Connection Pools - Part 2

Guidelines for Most OLTP Workloads

Eliminate dynamic resizing of connection pools

- Establishing a new connection is expensive
- Especially when the system is already under pressure

Aim for about right

Ten processes per CPU is almost always more than enough

- Enough to make the server busy
- But not so busy that performance suffers

Something New

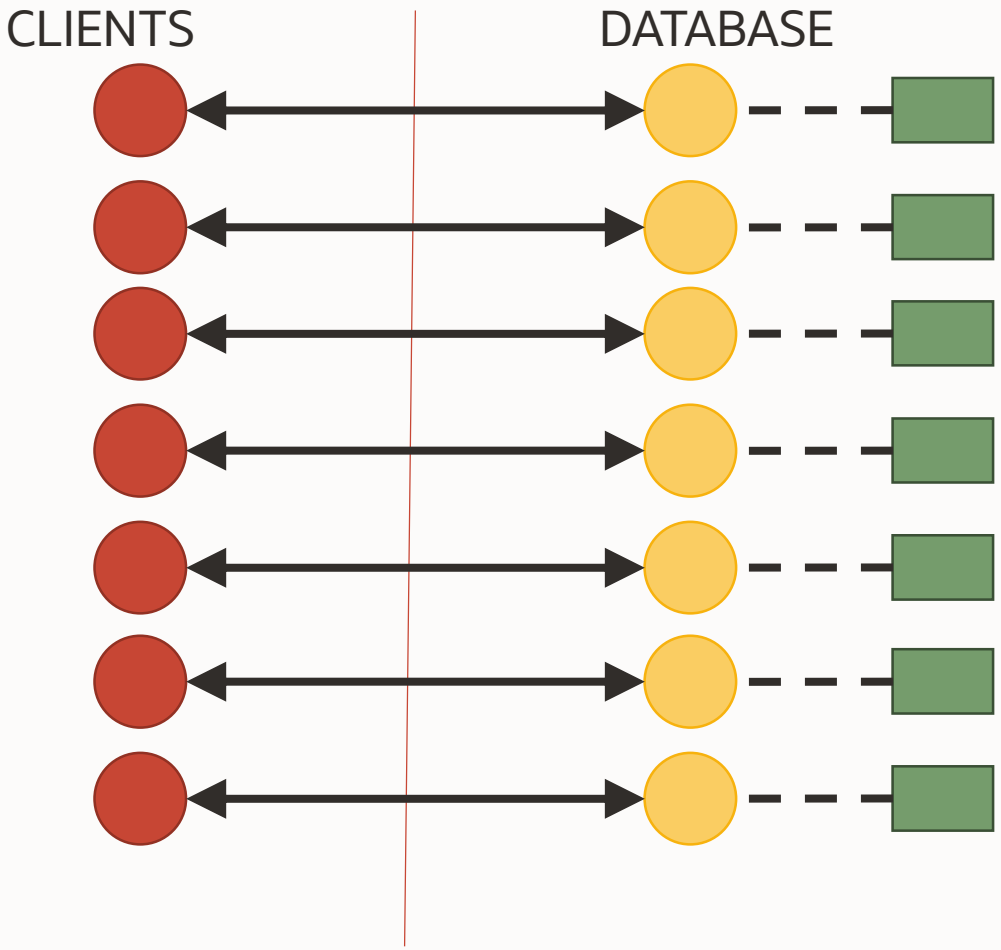
Modern application often scale horizontally

- More containers
- More application server instances
- More connection pools

Sometimes it is hard to meet the guidelines

Enter Proxy Resident Connection Pool

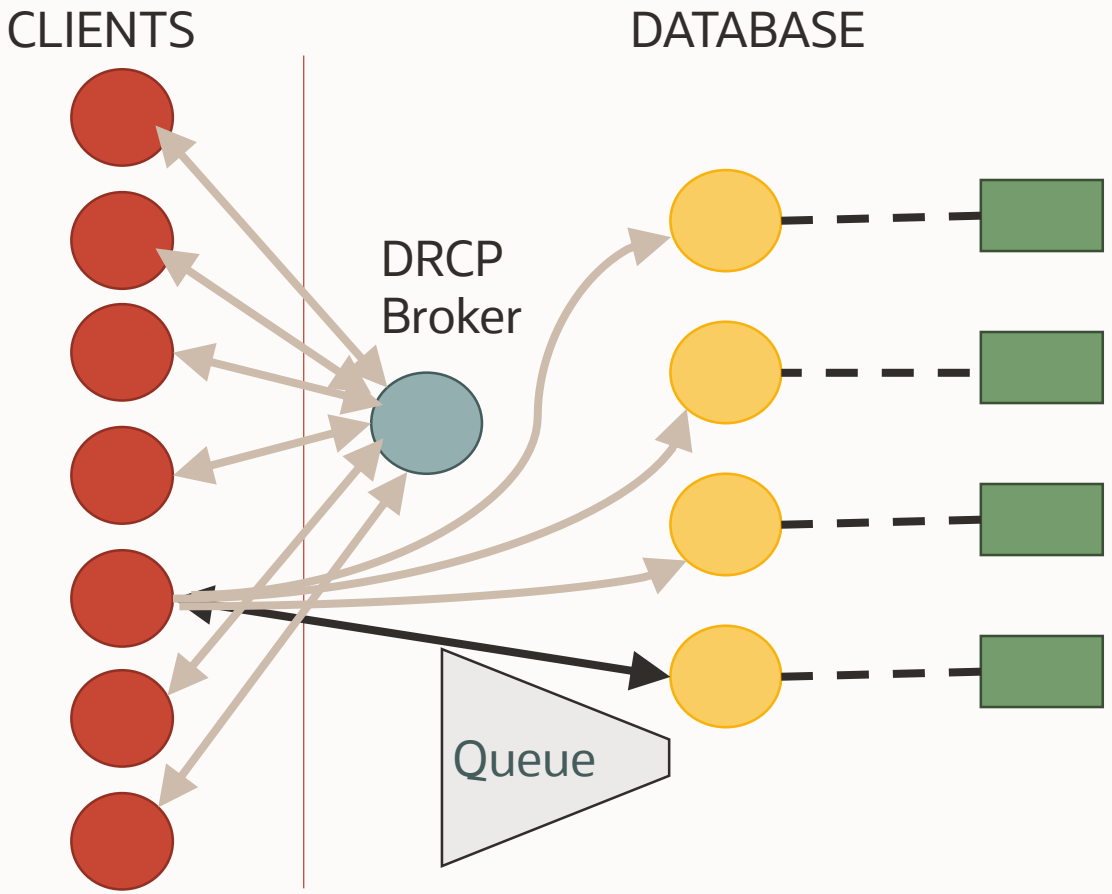
First Something Old



Each client is connected to a separate database process and is associated with separate database sesssions.

This allows for the programming style with mixing of user interface and database calls.

Let the database do the pooling



Using DRCP – Database Resident Connection Pool – the pooling between sessions now take place on the database side.

This allows for pooling using single-threaded applications.

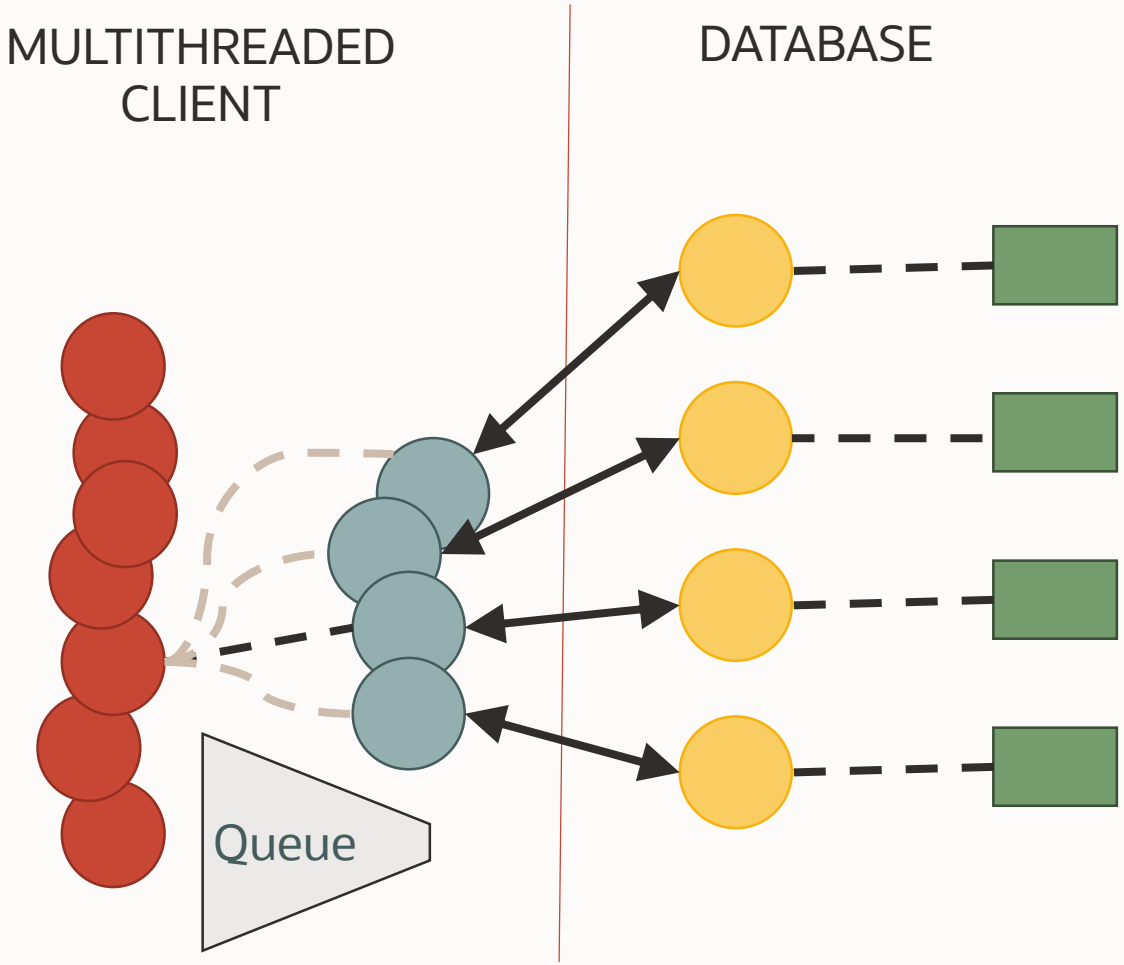
The programming model with separation of user interface and database calls is *required.*

The actual code (Java, Python, C) is almost the same as for multi-threaded clients.

Again, clients will have to queue when requesting a session, if none is available

Support in ODP.NET as of 18c only.

Typical current model



The client side now has worker threads that are responsible for user interface and has a pool of threads that are connected to the database.

This *requires* the programming model with separation of user interface and database calls.

- Java does this via UCP
- Python does this via cx_Oracle.SessionPool
- ODP.NET pool attibutes of OracleConnection
- OCI has OCISessionPool

A multi-threaded client is *required*

From the database's perspective, this is the same as legacy, so the database doesn't "know" requests are from different clients.

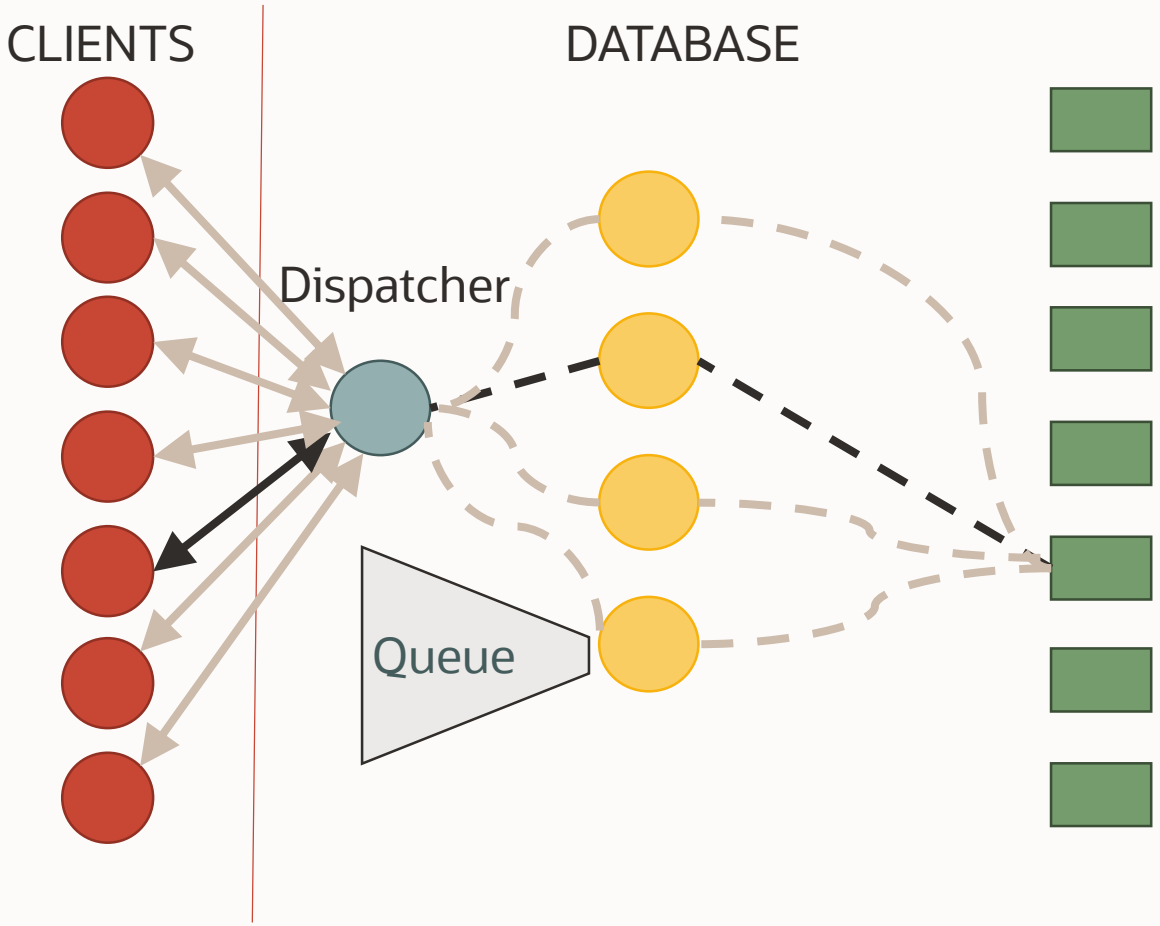
If no session is available when requested, the worker thread will have to queue.

Session pooling core points

- The application identifies itself with the database *once*, up front.
- This allows the pool (be it threads in the client or the DRCP broker) to establish connections to and sessions in the database.
- When needed, i.e. when all data is available to process a complete business transaction, the application acquires a session from the pool.
- As soon as the business transaction is complete, the session is returned to the pool.
- The call to acquire a session may have to queue for one to be available (or a new one to be started)
- No database state can be kept after releasing the session back to the pool.

```
identify();
loop ...
  user_interface();
  user_interface();
  get_session();
    sql1;
    sql2;
    commit;
  release_session();
end loop;
leave();
```

What about shared server?



Shared server is *very different* from the pools mentioned so far.

Each client has a *session* in the database; the session is retained during user think time

No session get/release calls

During processing of one *database call* (such as one SQL statement, a commit, etc), some database process does the work.

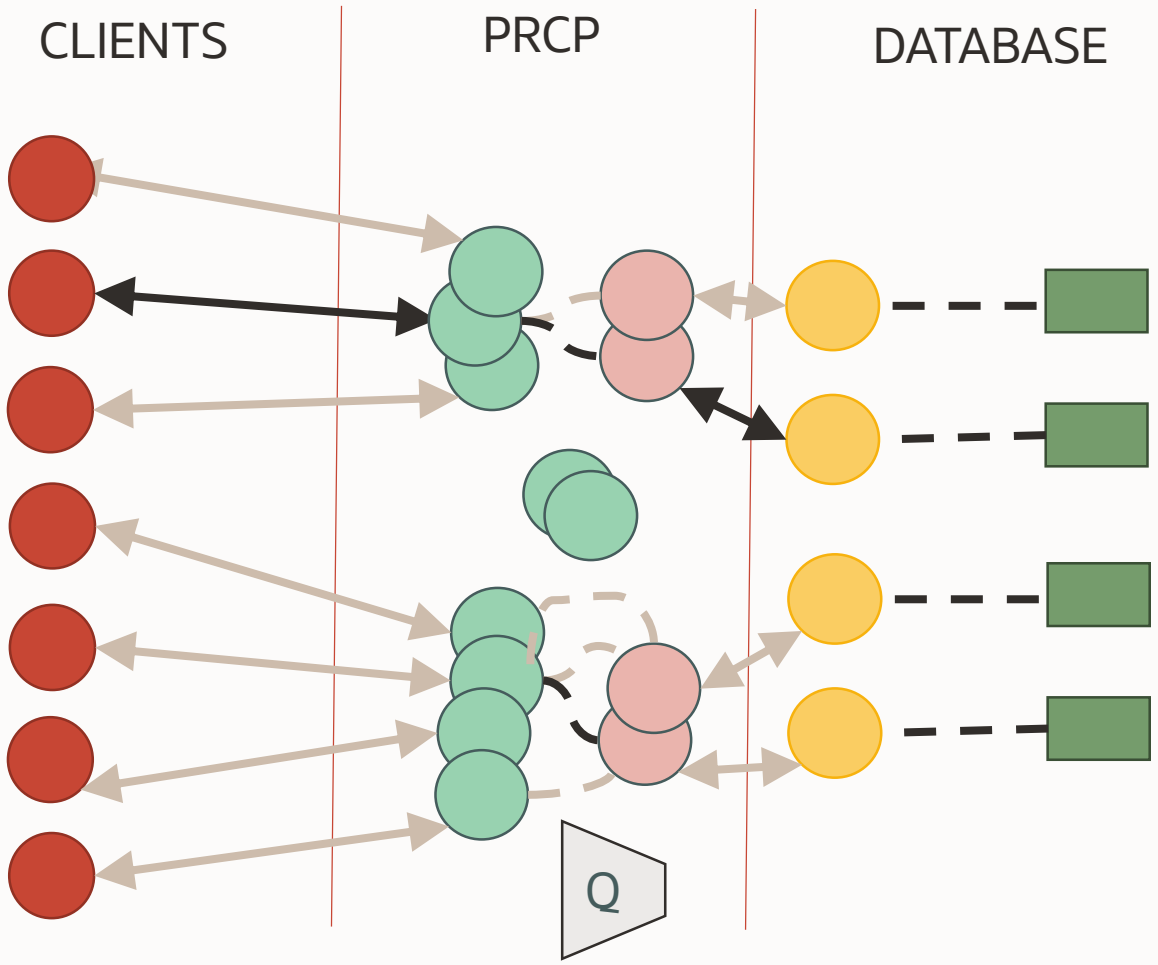
When no database call is in progress, the client is not associated with a process; however, the session (with state) is still there.

Clients will potentially queue to get an available process; this may happen for every database call.

Comparing the models

	Direct connection	Client pool	DRCP	Shared server
Application style	Any	session get/release required	session get/release required	Any
Programming model	Any	Multi-threaded only	Any	Any
Benefits	Supports legacy applications	Very scalable, database resources only spent doing database work	Somewhat scalable	Somewhat scalabile, supports legacy application style
Drawbacks	Not scalable, high risk of database contention	Multi-threading required	Database resources spent pooling	Database resources spent pooling, complex database configuration

PRCP – Proxy Resident Connection Pool



PRCP runs on a separate server

It has a pool of in-coming connections, that are one-to-one connected to client processes.

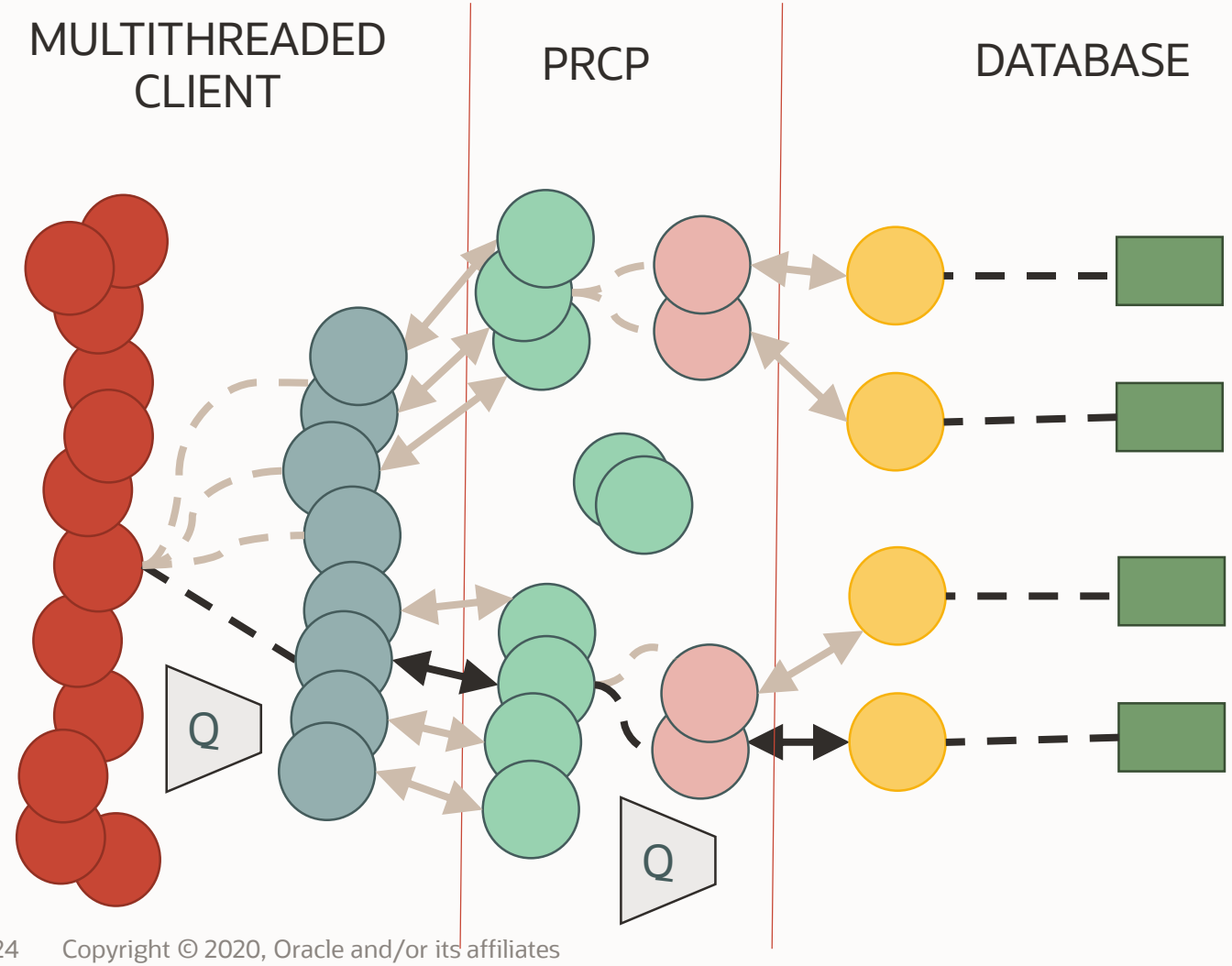
It has a pool of out-going sessions that are connected to database processes and associated with database sessions.

The database does (almost) not “know” its connections and sessions are from PRCP

PRCP allows pooling and provides a funnel *without* database side overhead and *without* the need for clients to be multi-threaded.

Clients may queue if no session is available.

PRCP and client pool together



Pooling in the client exactly as with ordinary pools.

When worker thread gets a session, it is first gotten from the client pool, then from the out-going pool in PRCP. Again, this implies two potential places for queueing.

Provides double funneling without database side overhead.

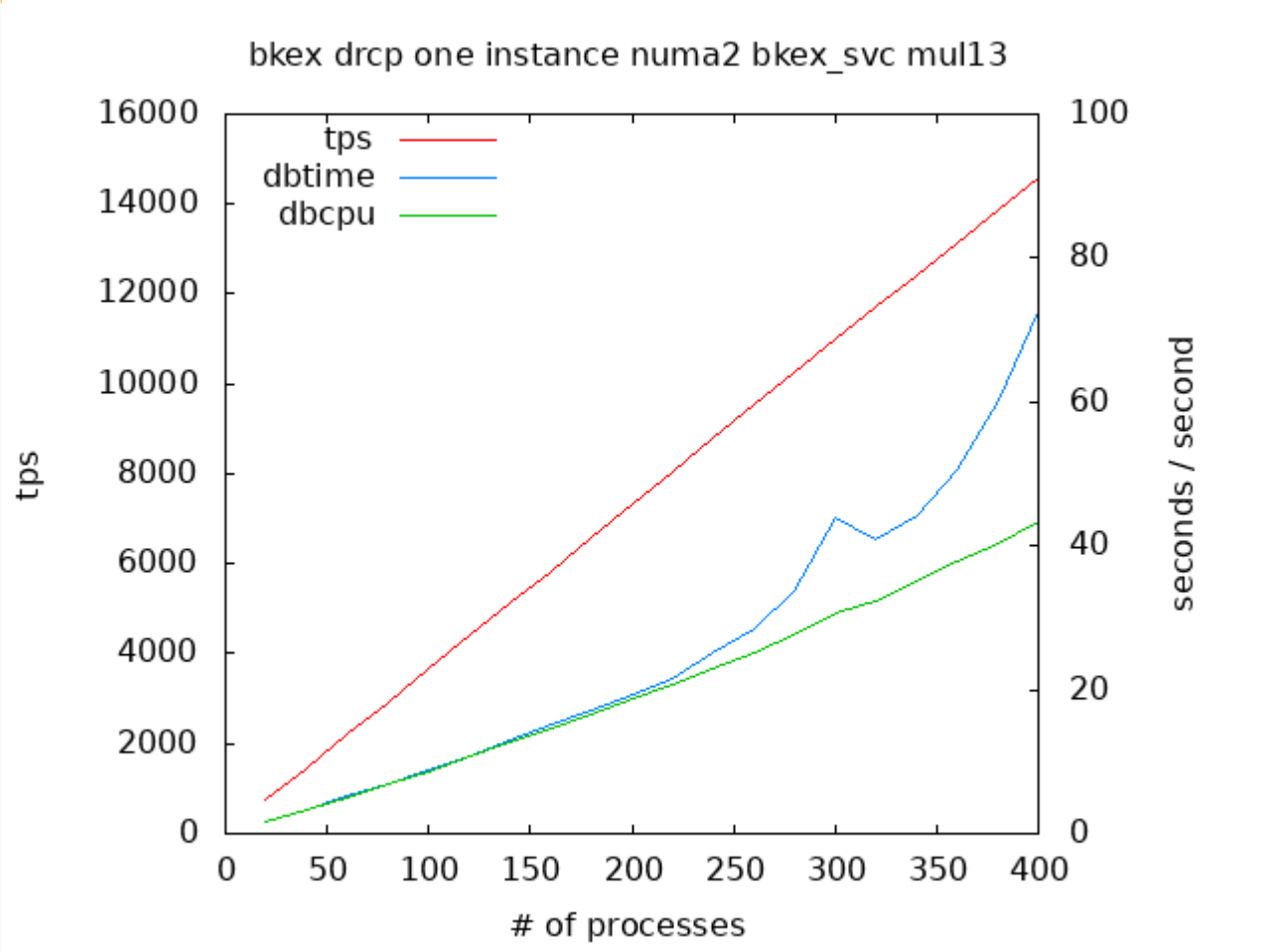
The “price” is two network hops.

Brings the X*Y*Z<N*C goal nearer.

Comparing the models

	Client pool	DRCP	PRCP	Client pool and PRCP
Application style	session get/release required in all cases			
Programming model	Multi-threaded only	Any	Any	Multi-threaded only
Benefits	Very scalable, pooling done outside database	Somewhat scalable	Very scalabile, pooling done outside database; different pools possible	Extremely scalable, pooling done outside database
Drawbacks	Multi-threading required	Database resources spent pooling; single pool for all tenants etc	Extra network hop, one funnel	Multi-threading required, extra network hop

Application pool and DRCP

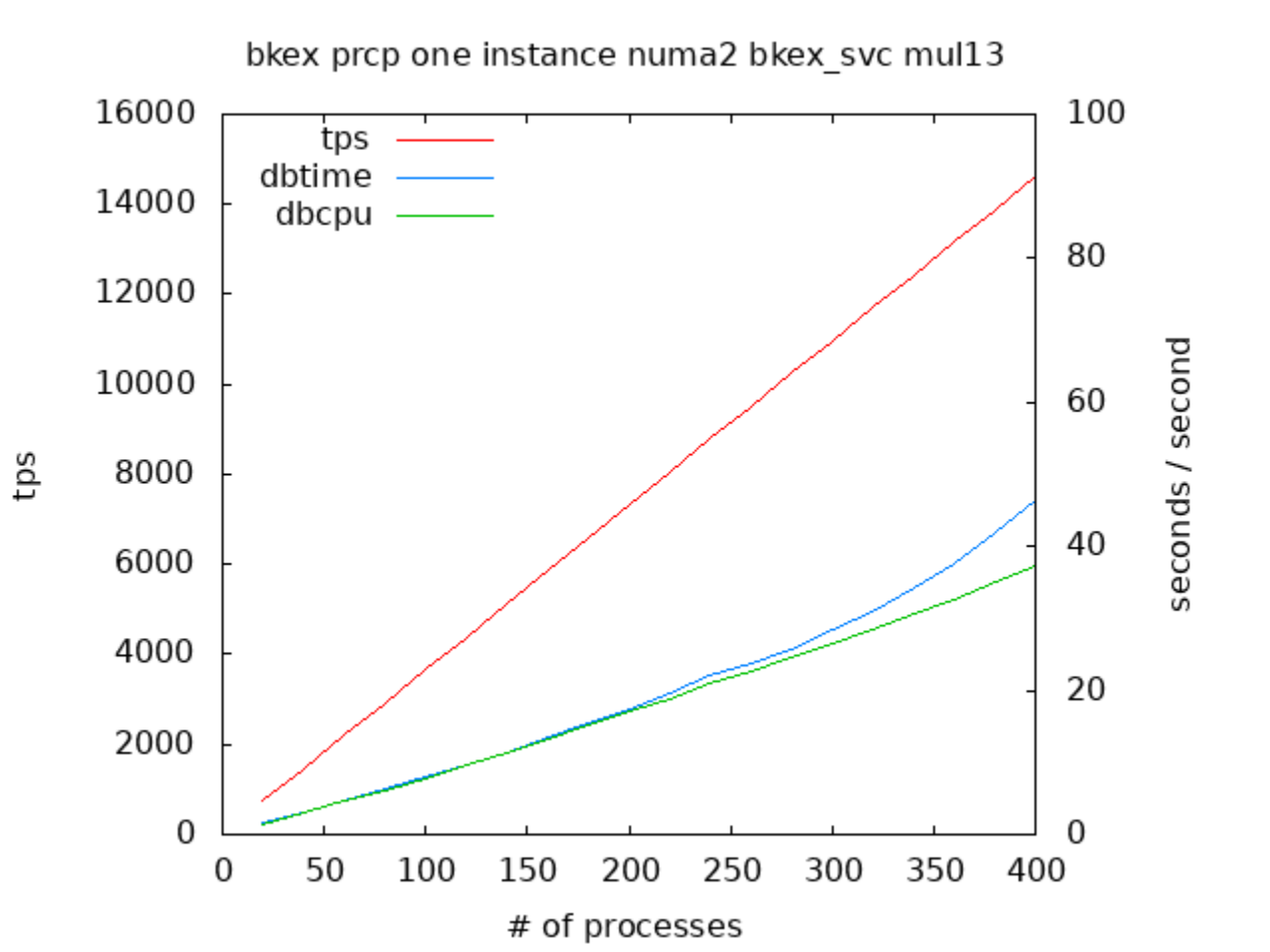


Some waits at high workloads

DB CPU effectively as with direct connection

No errors

Application pool and PRCP



Database CPU clearly reduced

Only little wait, mostly log file sync

No errors

Summary
What did we learn?

For most workloads, the established rules continue to apply

Ten processes per CPU is almost always more than enough

PRCP is very good at protecting the database when used with ordinary application pools

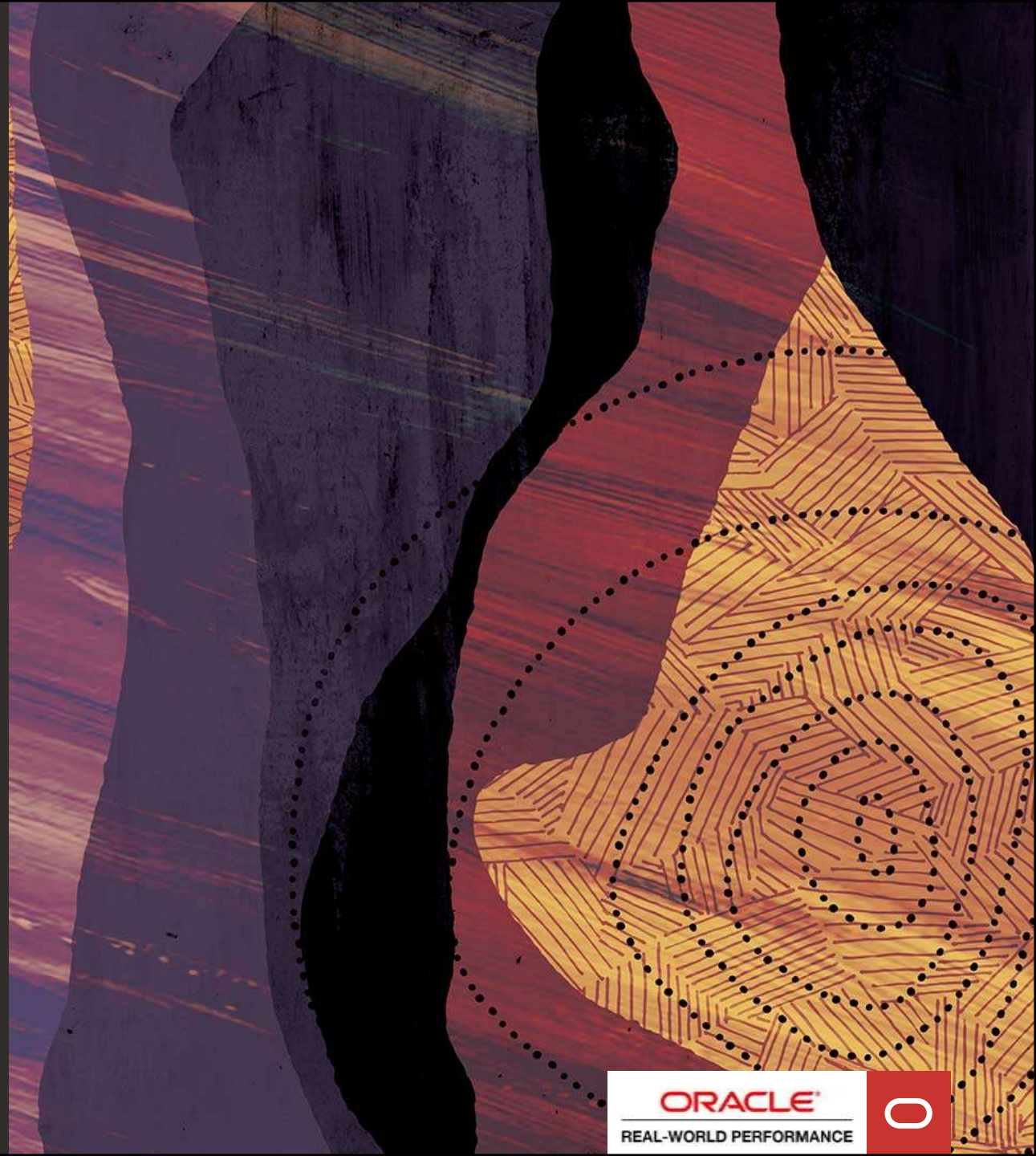
There is no such thing as a free lunch

- PRCP introduces an extra network hop
- The extra hop increases latency for lightweight SQL

If you are interested in PRCP, please contact Real-World Performance

Thank you

Michael Hallas
Real-World Performance Team
Oracle Database Development



ORACLE