kindred

Connection Pooling Demystified

Priit Piipuu

WHOAMI: Priit Piipuu

- Started as a DBA 20++ years ago
- Currently database performance engineer at Kindred Group
- Main job is to help developers use Oracle technologies in best way possible
- Blog: https://priitp.wordpress.com
- twitter: @PPiipuu

DBAs run the world

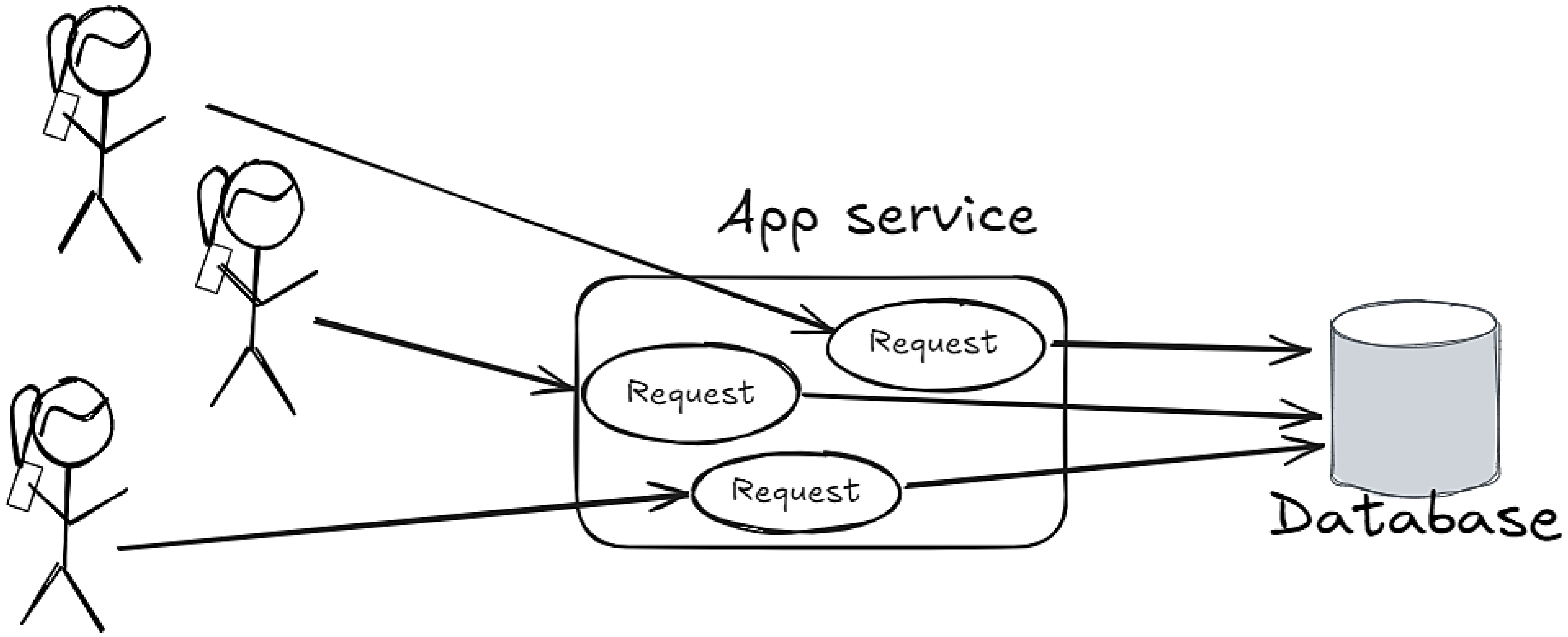
strava.com/clubs/dbasruntheworld

In this presentation…

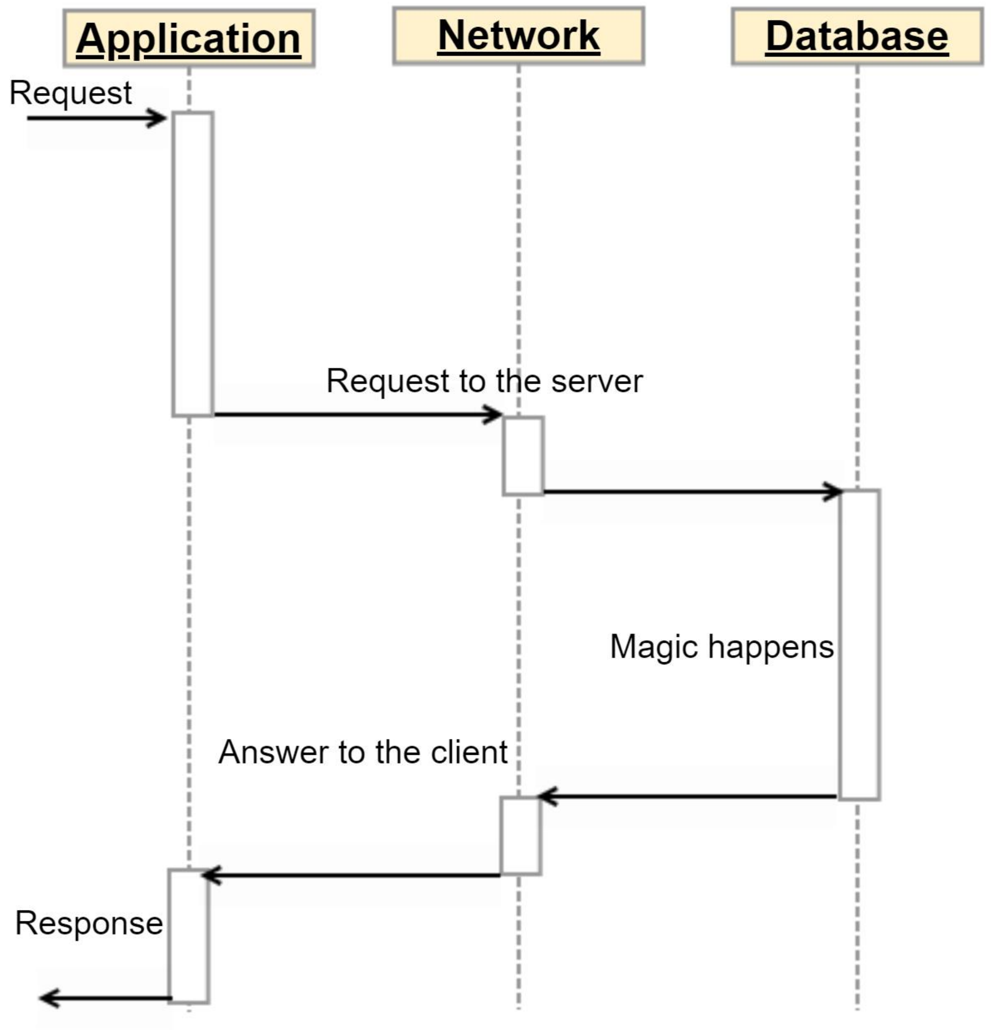
- Why connection pooling
  - Problems it solves
  - Problems it creates
- Extra features in connection pools
  - Oracle Universal Connection Pool
  - Database Resident Connection Pool
- Connection pool sizing & configuration

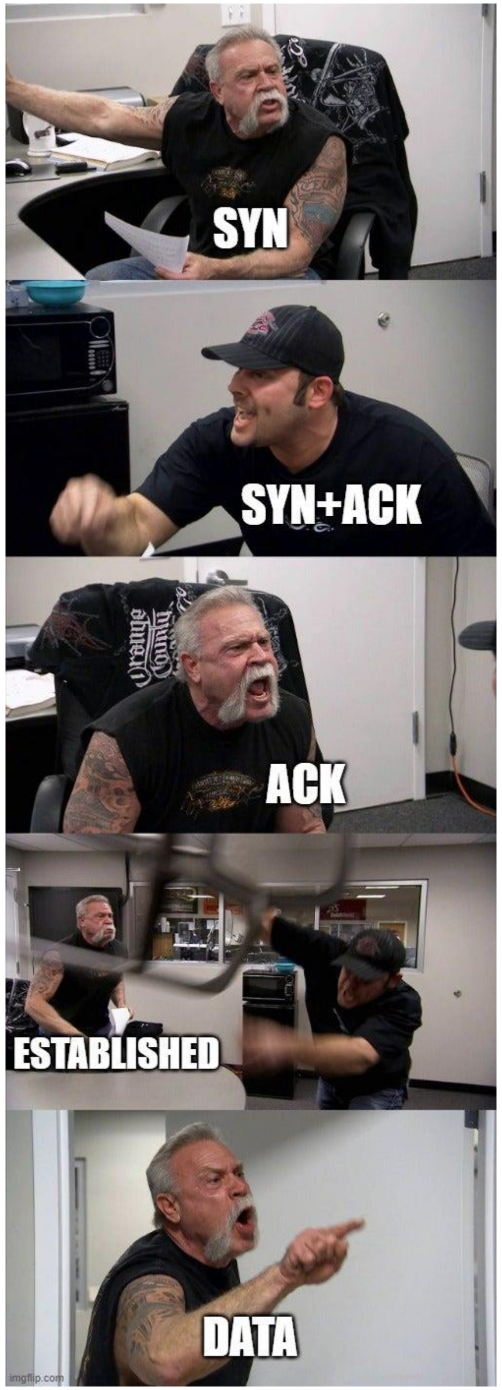
Modern application architecture

- Request-response model
  - One of the basic ways how computers communicate
  - Has been around for a while
  - One computer sends a request while second computer answers to that request
  - In simple cases communication is syncronous: think of typical web service

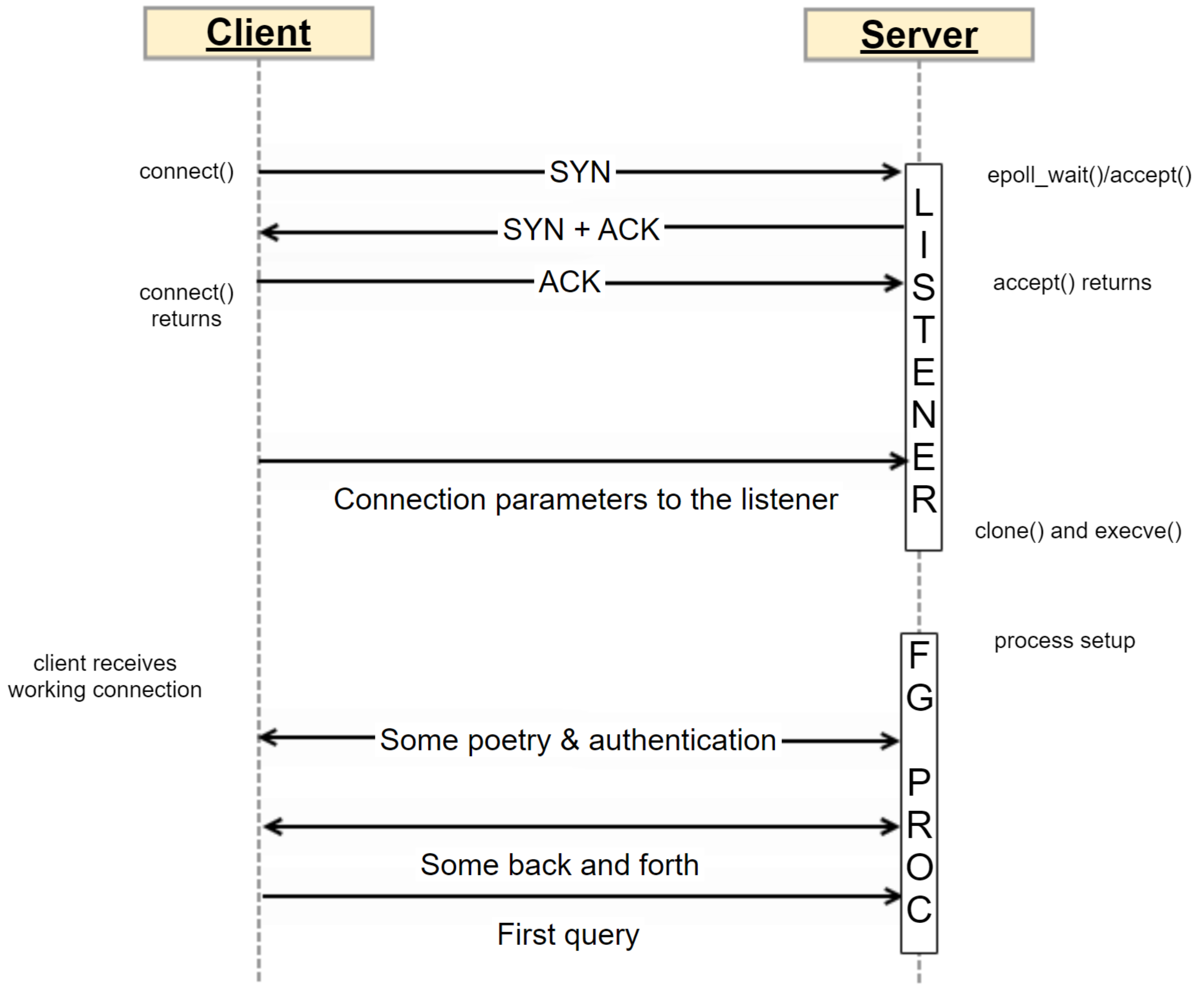

App service
Request
Request
Request
Database

Request-response model




SYN
SYN+ACK
ACK
ESTABLISHED
DATA
imgflip.com

Birth of a connection



Is it worth it?

		Median	p99
Borrowing connection from UCP	Without connection validation	2.6us	13.5us
	With connection validation	562us	990us
Creating new connection	Local	46ms	56ms
	Remote	49ms	62ms
	Local with DRCP	19.9ms	28ms
	Remote with DRCP	22ms	33ms

Is it worth it?

		Median	p99
Connection close	UCP	9us	34us
	Local	780us	1ms
	Remote	1.1ms	2.5ms

- Test case
  - Single threaded, talks to single database instance and local listener
  - Private cloud, vm's in different networks
  - Opens and closes connection, builds histograms on response times
  - N = 1m

Long-running sessions (I)

- Good
  - Reduces network connections
  - Predictable response times for fast queries
  - Cursor caching in the foreground process
  - Statement caching in application

Long-running sessions (II)

- Bad
  - Network isn’t reliable
  - RAC: Troubles with planned downtimes, and no connection time load balancing

Dead connection detection (I)

- Lightweight connection validation in Oracle JDBC
  - Sends empty packet to the database
  - Does not wait for reply
  - Available since 18c, specific to JDBC thin
  - Connection.isValid method has timeout

Dead connection detection (II)

- In UCP
  - Connection validation is by default off
  - When enabled, default is lightweight validation
    - Toggled with PoolDataSource.setValidateConnectionOnBorrow method
  - Lightweight validation is disabled when SQL statement for validation is set
    - PoolDataSource.setSQLForValidateConnection method
  - No timeout for lightweight connection validation

Dead connection detection (III)

- Fast Application Notification
  - Based on Oracle Notification Service
  - Notifies subscribers of service configuration and status changes
  - Needs Clusterware, in case of multiple clusters messages can be forwarded by GDS
  - Starting with UCP 12.2 FAN and FCF are enabled automatically

Dead connection detection (IV)

- Fast Connection Failover
  - Planned outages: affected connections are closed gracefully
  - Unplanned events: affected connections are closed immediately

Connection pool from developer's perspective (I)

- Similar API compared to vanilla JDBC or OCI
- Single point of configuration
  - Number of connections
  - Statement caching

Connection pool from developer’s perspective (II)

- Automates some of the connection management tasks
  - Dead connection detection
  - Handling stale connections

Connection pool from developer's perspective (III)

- Extra features in UCP:
  - Support for Fast Application Notification and Fast Connection Failover
  - Runtime connection rebalancing and connection affinity
  - Application continuity

Connection pool as a circuit breaker

- With microservices problematic requests will propagate to the upstream services
  - Worst case: waiting on database or connection pool introduces cascading failure
- Query timeout for the rescue
  - Sets timeout on java.sql.Statement object
  - Timeout is in full seconds

Connection pool as a circuit breaker

- Not everything is query
  – Query timeout does not affect commits, metadata operations, connection creation
  – These can be handled by setting timeout on socket read (through java.sql.Connection)
  – Only way to achieve timeouts less than one second

Connection pool sizing(I)

- Connection pool limits max number of connections
  - Initial pool size: number of connections initially created, default is 0
  - Minimum pool size: minimum number of connections pool maintains
  - Maximum pool size: upper limit for the number of connections
  - Connection pool always tries to reach minimum pool size, unless minimum pool size is not reached

Connection pool sizing(II)

- Limits are per connection pool instance.
- What about autoscaling?

Connection pool sizing(III)

- If max number of connections is reached, new requests will wait for connections to become available
  - Connection wait timeout: how many seconds application request waits for a new connection
  - Default is 3 seconds

Dynamically sized connection pools

- Initial pool size/minimum pool size is set low
- Maximum pool size is huge
- During the rush hour application will create large number of new connections
- Result is lousy response time and extra strain on systems

Dynamically sized connection pools

- Solution: use static connection pools
  - Initial pool size = minimum pool size = max pool size

Connection pool sizing: Little's law

- Little's law:
  - $L = \lambda W$
  - L -- average customers in stationary system,
  - λ -- average arrival rate
  - W -- is average response time

Little’s law: example

- Example:
  - Web service is expected to handle 600 requests per second on average (λ)
  - Average response time for database query is 15ms (W)
  - So on average there’s 600*0.015s = 9 active database sessions
- Gives either minimum pool size or a good starting point for further tuning

Little’s law and the real world

- Everything is a long term average
- It describes stationary process
  - Average and variance do not change over time
- All requests are assumed independent
  - No extra variance due to the contention on common resources
- What if your load pattern has seasonality?

Mitigation: Resource Manager

- Can guarantee limited amount of CPU for important sessions
- Active session pools: limits number of active sessions allowed for specific useg group
- Limits amount of time session can be idle

Mitigation: Oracle Connection Manager

- Oracle Connection Manager (cman)
  - Session multiplexing can reduce number of idle sessions

Mitigation: Database Resident Connection Pool

- Database Resident Connection Pool (DRCP)
  - Can work with cman and UCP

Mitigation: shared servers and threads

- Esoterics
  - Shared servers
  - Threaded execution

DRCP on the client side (I)

- For the developer
  - On client side connection type must be specified as POOLED
    - Example: jdbc:oracle:thin:@test.example.com:1521/testpdb.dbs:POOLED

- New connection in JDBC:
  - Reserve and release the connection manually

- New connection with UCP?
  - It will happen automatically

DRCP on the client side (II)

- Connection class name is set through oracle.jdbc.DRCPConnectionClass property
- Server processes can be tagged, application can attach to the tagged session
  - Application can control if DRCP can reuse the connection or not
  - oracle.jdbc.OracleConnection.attachServerConnection and detachServerConnection methods

DRCP on the client side (III)

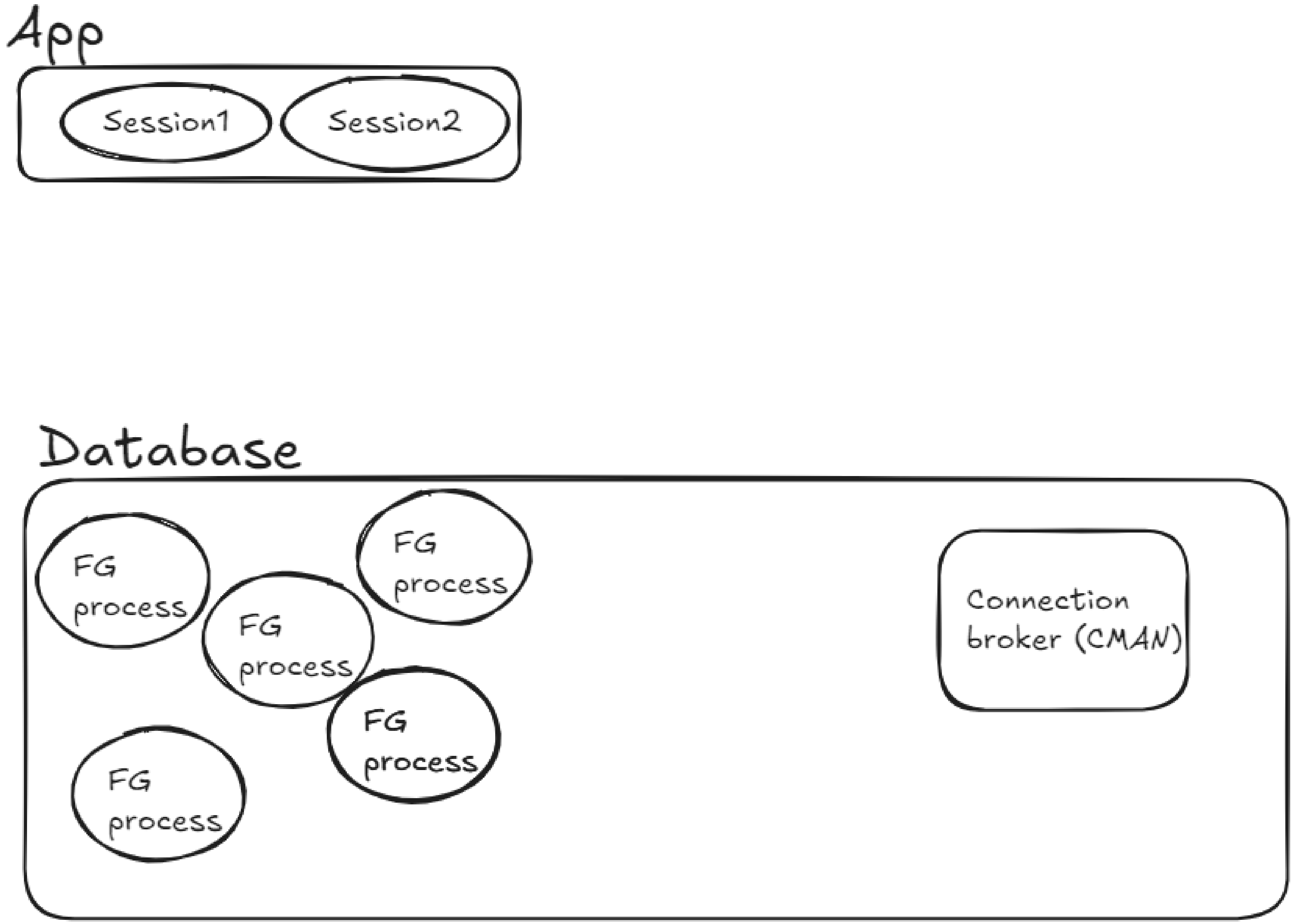
- Application can specify if it wants brand new session or reuse old one
  - Reuse is good
- Callback to fix database session state

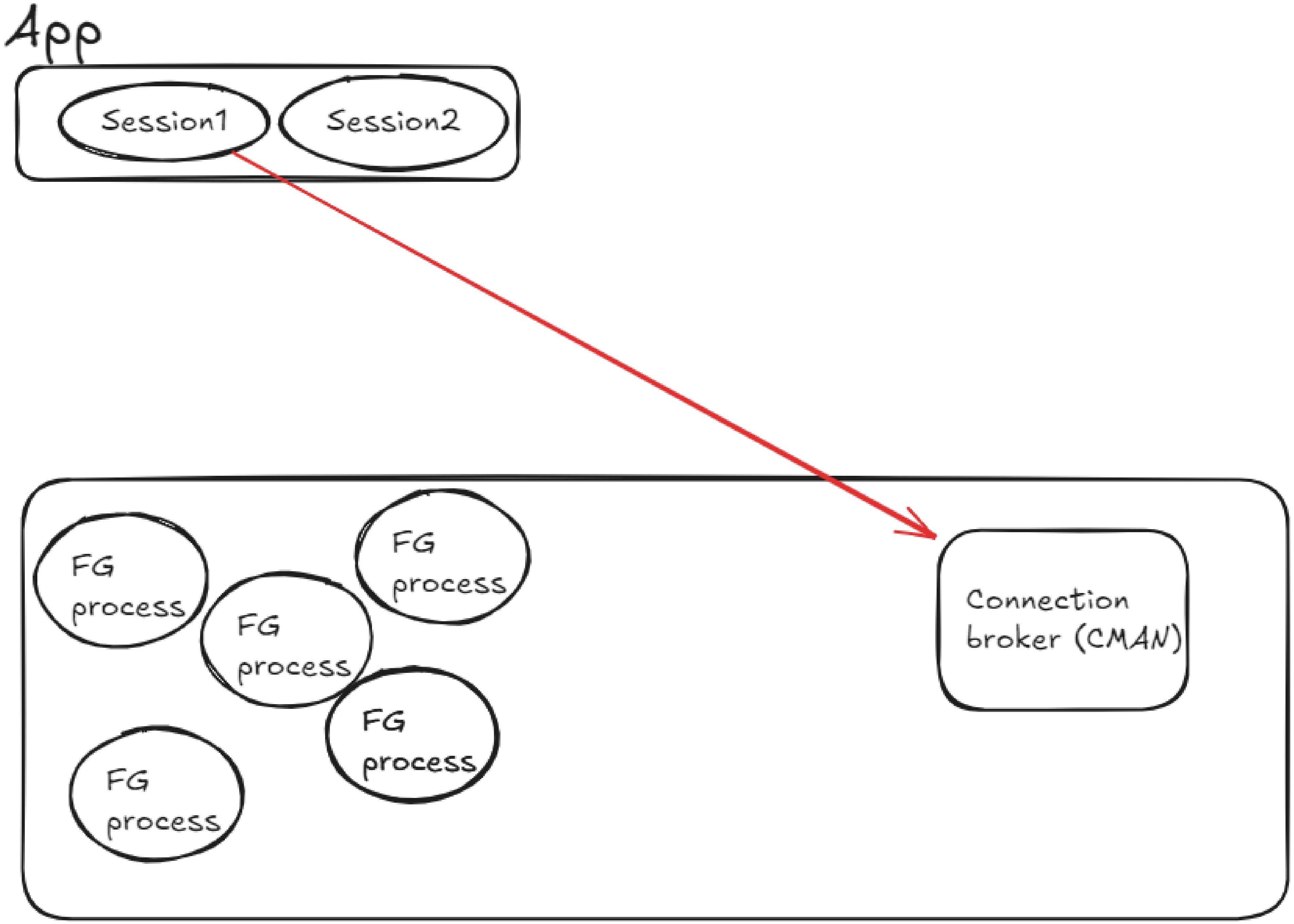
DRCP on server side (I)

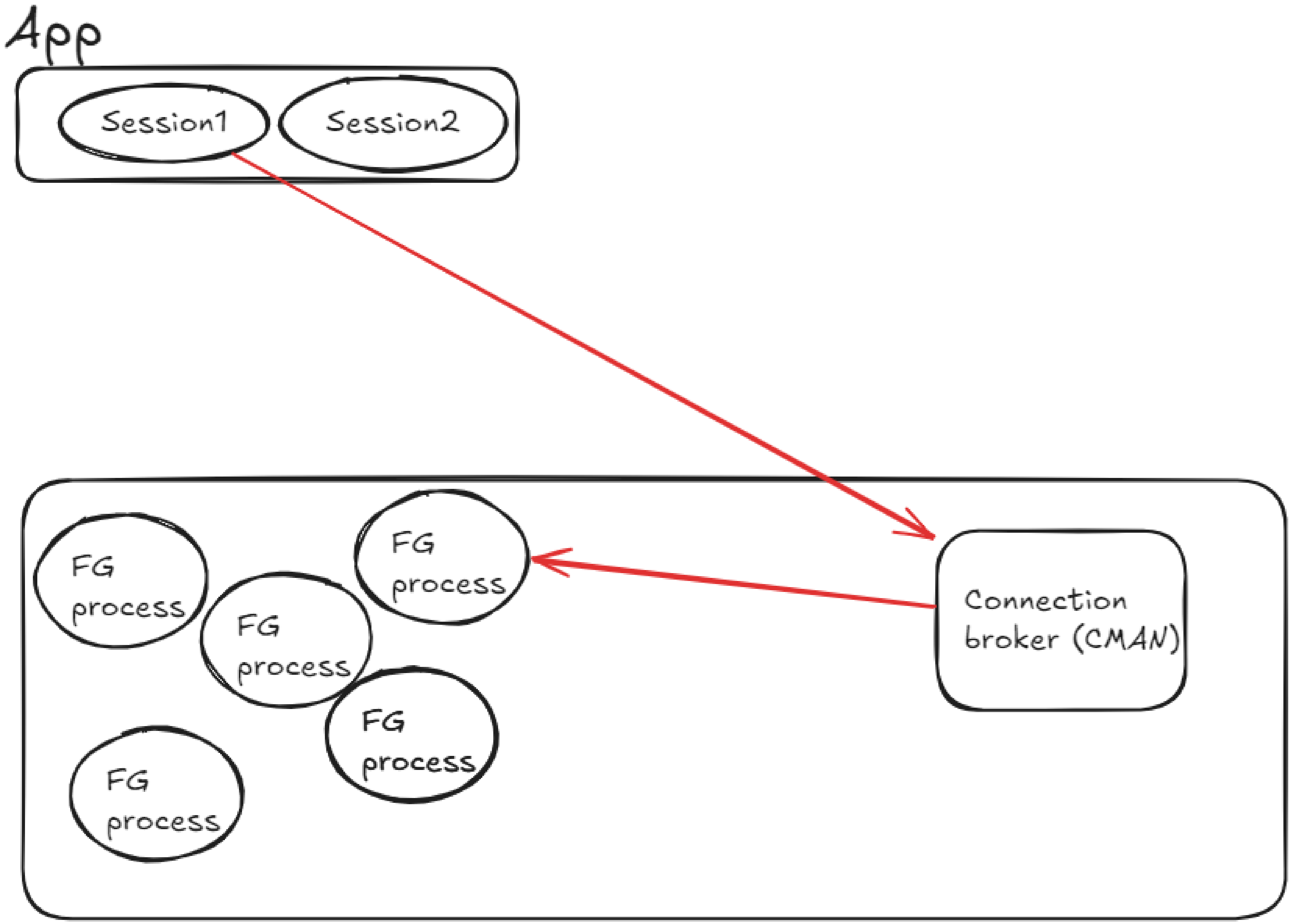
- Default connection pool is created, but not started
- Managed through DBMS_CONNECTION_POOL package
- By default connection pools are created in CDB level
  - Can be managed on PDB level since 21c
- Does not support TCPS

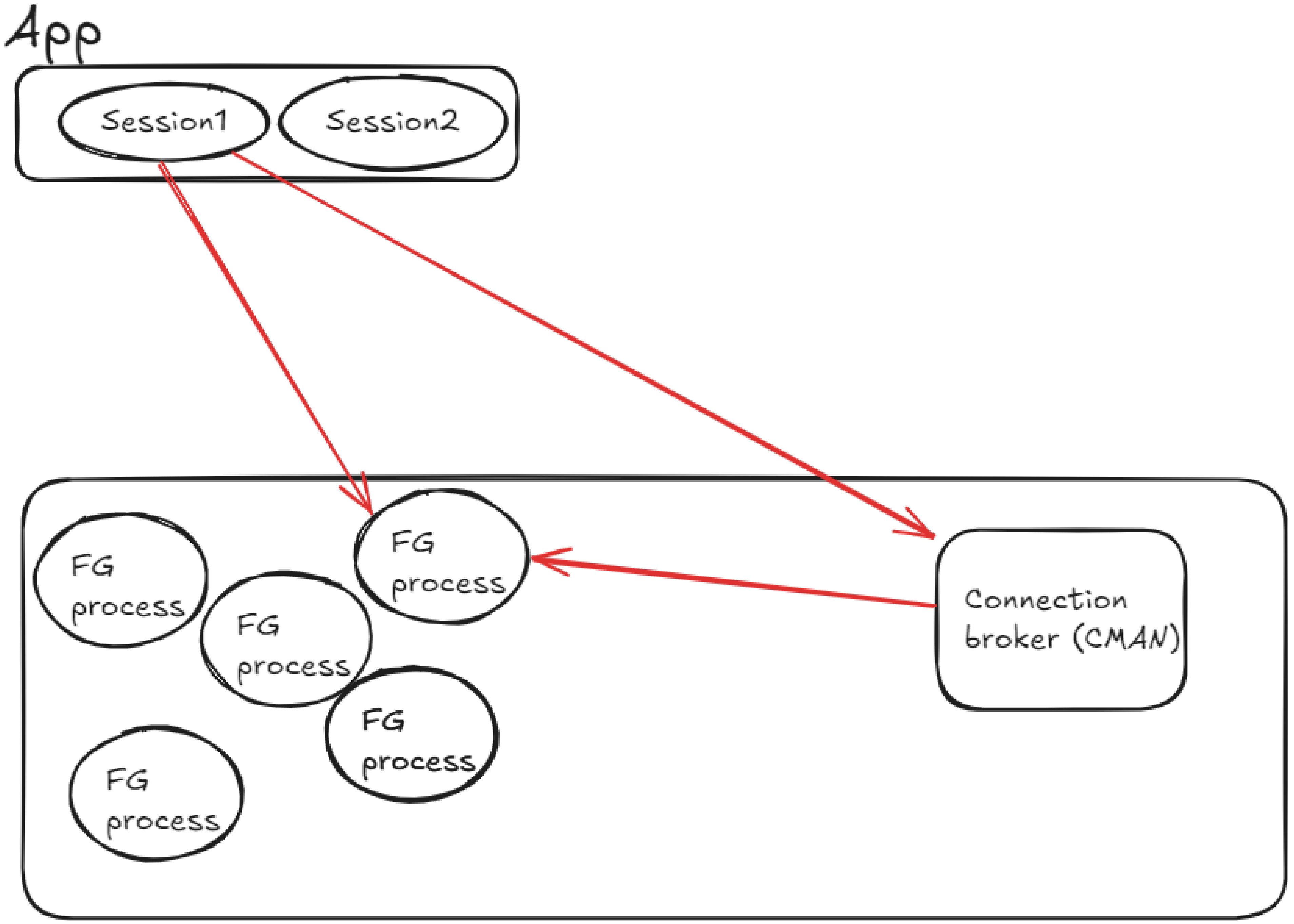
DRCP on server side (II)

- Has usual set of configuration parameters
  - MINSIZE and MAXSIZE
  - SESSION_CACHED_CURSORS -- # of cursors to cache in pooled session
  - INACTIVITY_TIMEOUT – how long pooled server can stay idle before terminated
  - MAX_THINK_TIME – how long client can stay idle before client connection is terminated
  - MAX_TXN_THINK_TIME – how long client with open transaction can stay idle

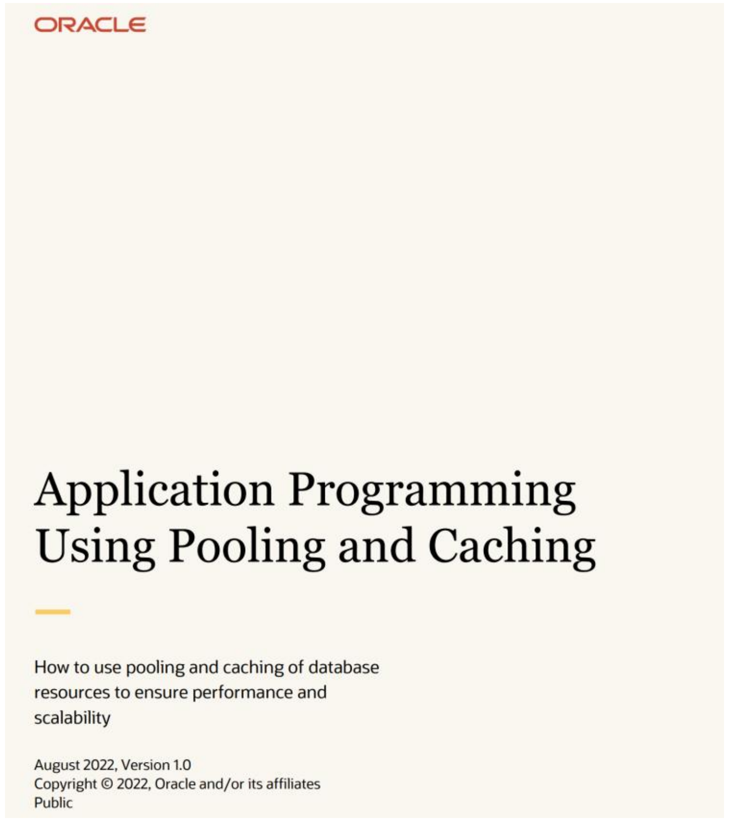

App
Session1
Session2
Database
FG process
FG process
FG process
FG process
FG process
Connection broker (CMAN)


App
Session1
Session2
FG process
FG process
FG process
FG process
FG process
Connection broker (CMAN)


App
Session1
Session2
FG process
FG process
FG process
FG process
FG process
Connection broker (CMAN)


App
Session1
Session2
FG process
FG process
FG process
FG process
FG process
Connection broker (CMAN)

Application Programming Using Pooling and Caching


ORACLE

Application Programming Using Pooling and Caching

How to use pooling and caching of database resources to ensure performance and scalability

August 2022, Version 1.0
Copyright © 2022, Oracle and/or its affiliates
Public

- https://download.oracle.com/ocomdocs/global/Application_Programming_Using_Pooling.pdf

NL.OUG
NL ORACLE USER GROUP

Please fill in your evaluations

Connection pooling demystified-Priit Piipuu 141 kindred