Spring Boot Microservices

Beginner to Guru

Datasource Connection Pooling

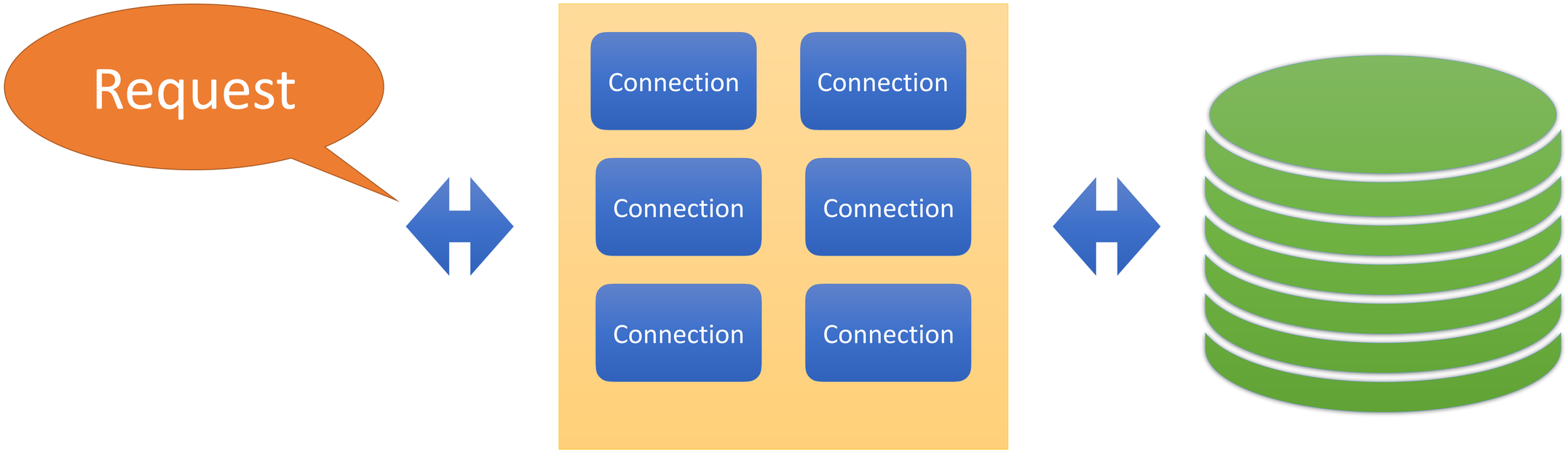
Datasource Connections

- Establishing a Database Connection is an expensive operation
  - Call out to Database Server to get authenticated
  - Database Server need to authenticate credentials
  - Establish a connection
  - Establish a session - ie allocate memory and resources

Datasource Optimizations

- Prepared Statements: SQL Statements with placeholders for variables
  - Saves server from having to parse and optimize execution plan
  - HUGE COST SAVINGS
  - Avoid SQL Injection attacks
- Optimizations within a single datasource connection:
  - Ability to cache prepared statements (may be at the server level too)
  - Use server side prepared statements
  - Statement Batching

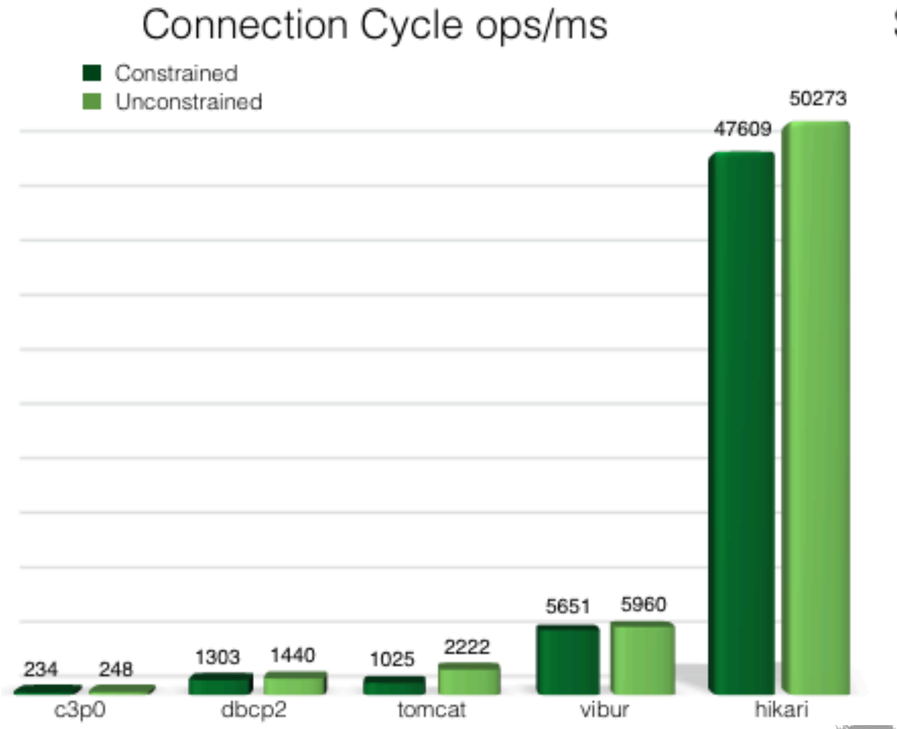
Datasource Connection Pooling

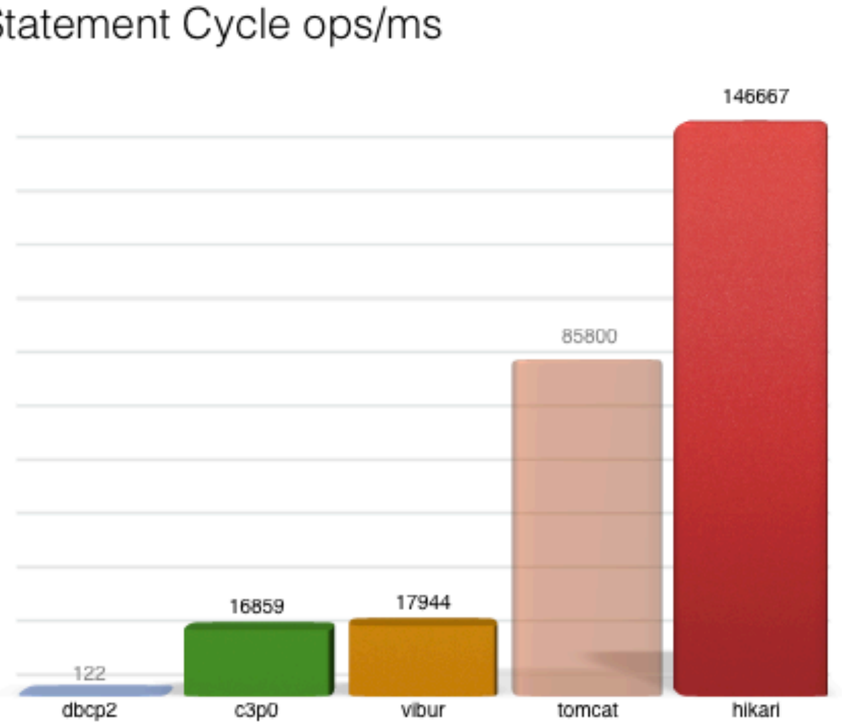


Datasource Connection Pooling

- Spring Boot 1.x used Tomcat
- Spring Boot 2.x moved to HikariCP
  - HikariCP is very light weight
  - Very high performance!
- Hikari has a number of configuration options

HikariCP Performance

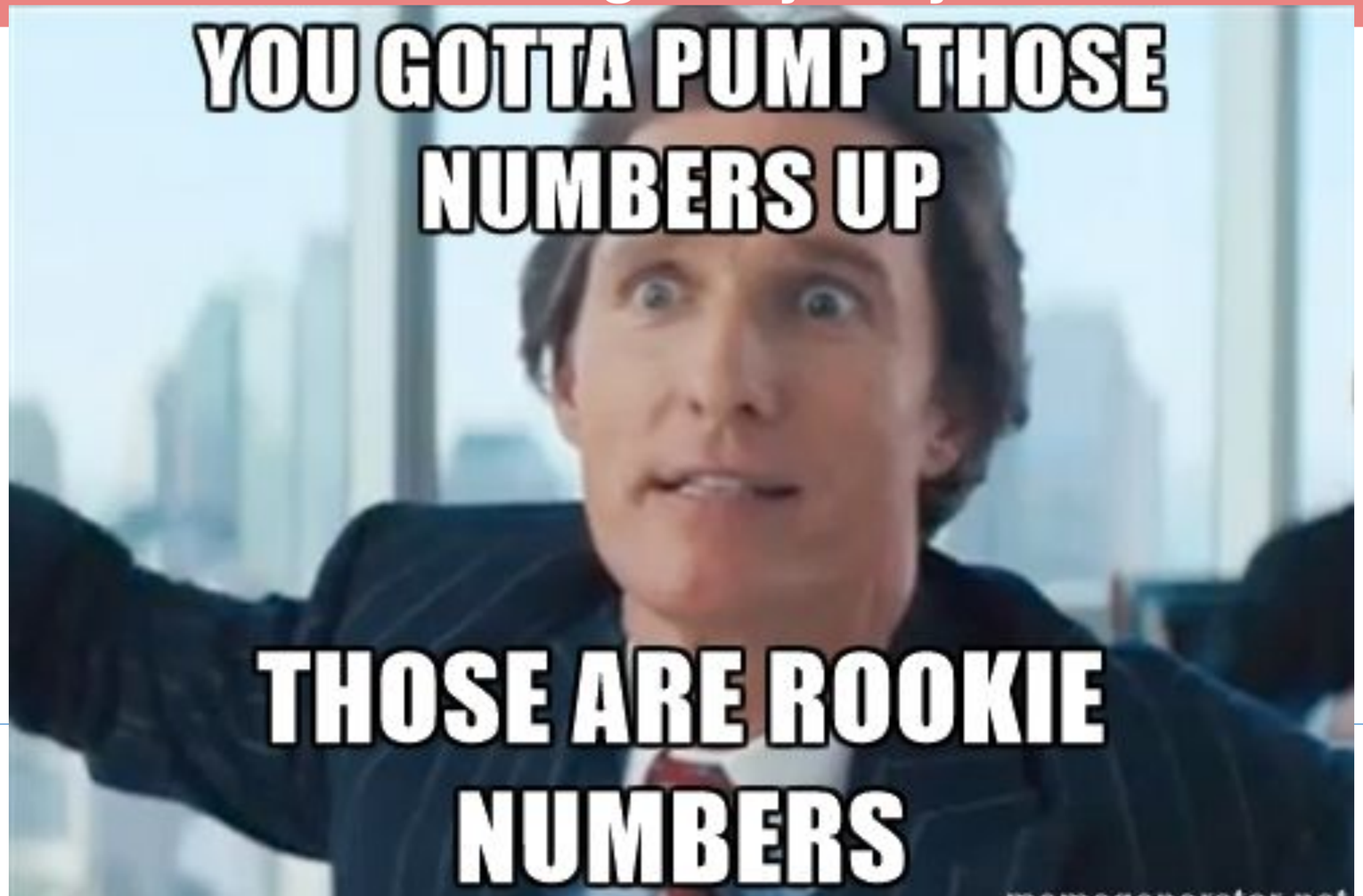




Hackers Guide to Connection Pool Tuning

- Every RDMS will accept a max number of connections - each connection has a cost!
- If running multiple instances of your microservice, keep number of pool connections lower
  - If fewer, can go to a higher of connections per instance
- MySQL defaults to a limit 151 connections
  - Can be adjusted to much higher - depending on the hardware running MySQL
- Statement caching is good
  - BUT - does consume memory on the server
- Disabling autocommit can help improve performance

Your Milage May Vary!!!



MySQL Recommended Settings

- HikariCP Recommended settings????
- Next video

SPRING FRAMEWORK

GURU