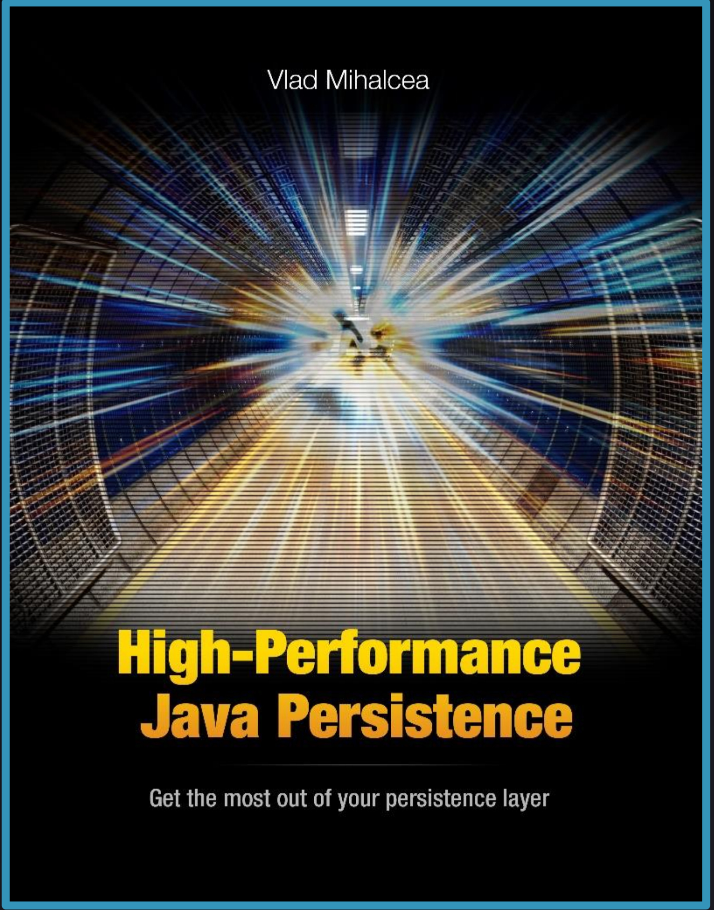
High-Performance Hibernate

VLAD MIHALCEA

About me

- @Hibernate Developer
- vladmihalcea.com
- @vlad_mihalcea
- vladmihalcea


Vlad Mihalcea
High-Performance Java Persistence
Get the most out of your persistence layer

Agenda

- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

Performance Facts

“More than half of application performance bottlenecks originate in the database”

AppDynamics - http://www.appdynamics.com/database/

Google Ranking

“Like us, our users place a lot of value in speed — that's why we've decided to take site speed into account in our search rankings.”

https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html

Performance and Revenue

"It has been reported that every 100ms of latency costs Amazon 1% of profit."

http://radar.oreilly.com/2008/08/radar-theme-web-ops.html

Response Time and Throughput

- n - number of completed transactions
- t - time interval

$$T_{avg} = \frac{t}{n} = \frac{1s}{100} = 10\ ms$$

$$X = \frac{n}{t} = \frac{100}{1s} = 100\ TPS$$

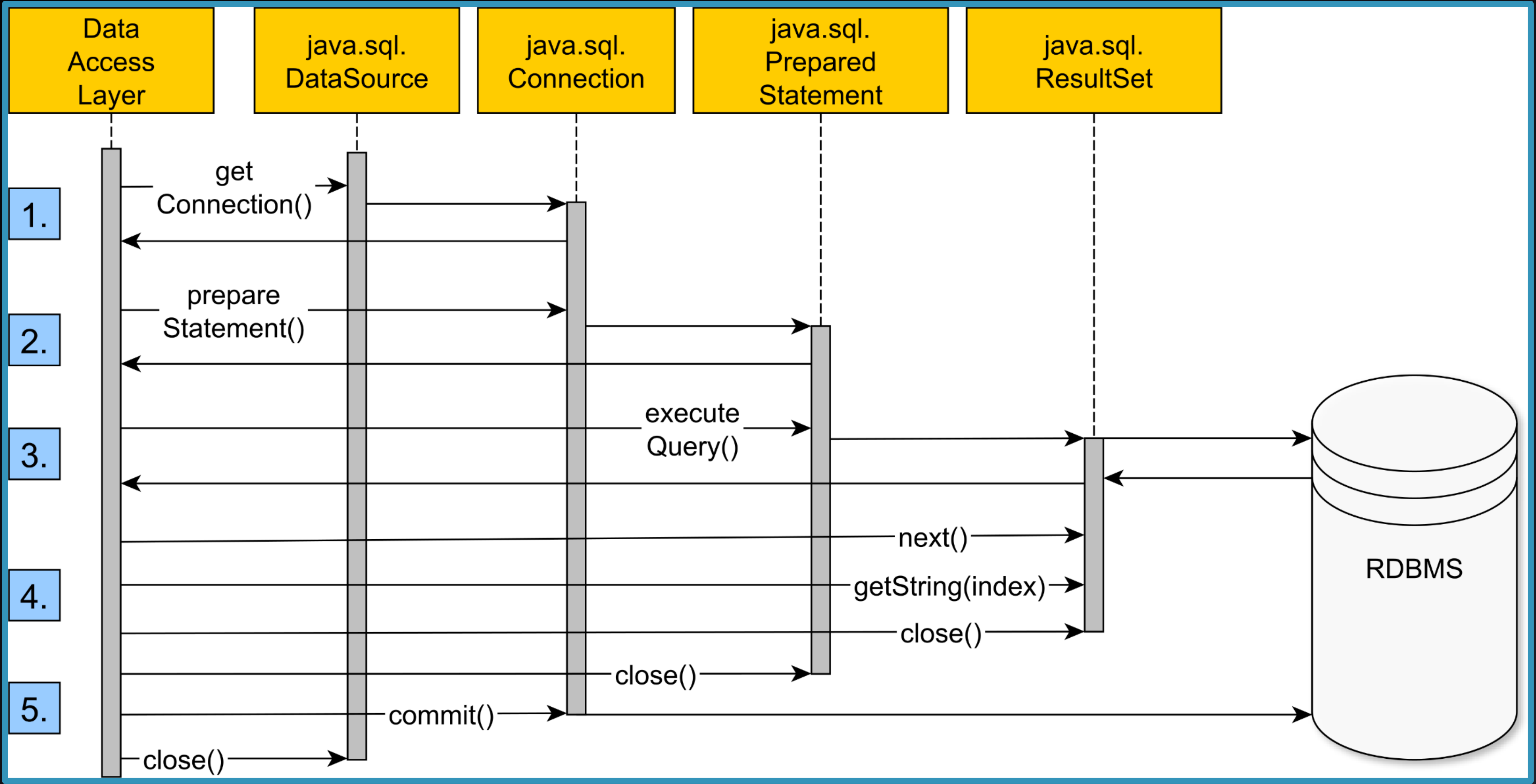
Response Time and Throughput

$$X = \frac{1}{T_{avg}}$$

"The lower the Response Time,

The higher the Throughput"

The anatomy of a database transaction



Response Time

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

- connection acquisition time
- statement submit time
- statement execution time
- result set fetching time
- idle time prior to releasing database connection

Agenda

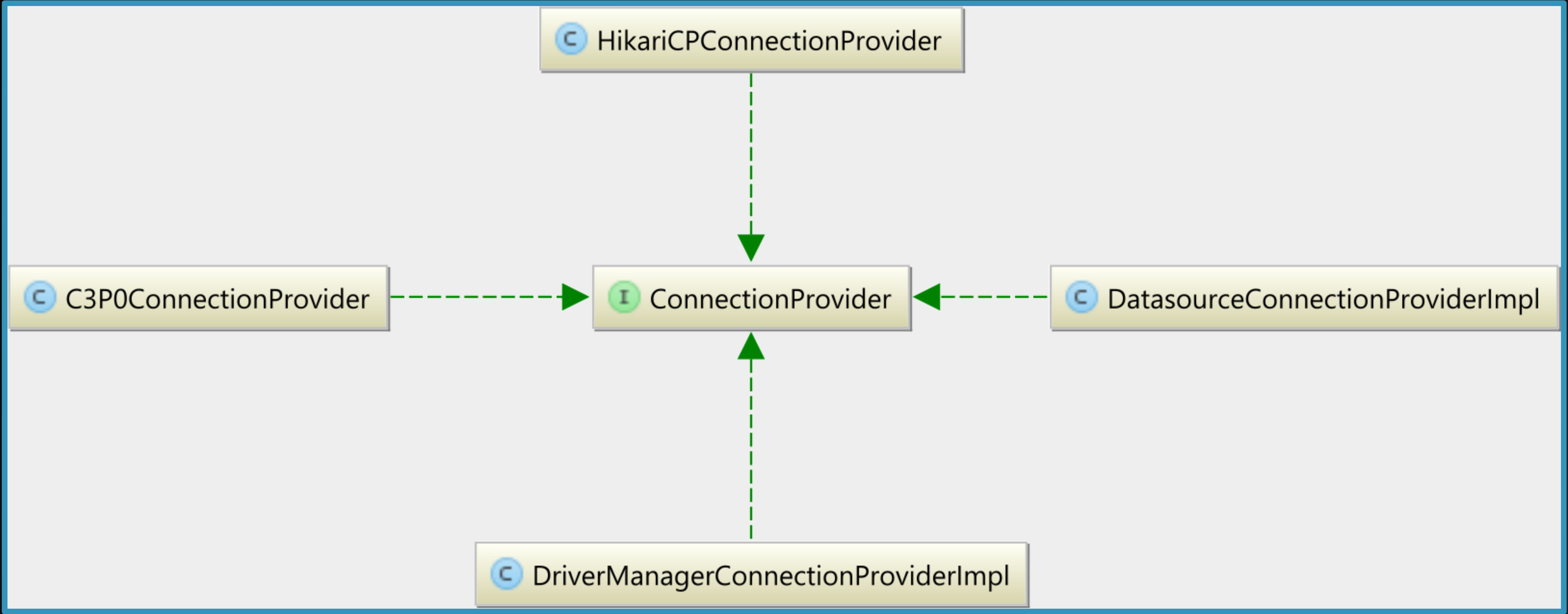
- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

Connection Management

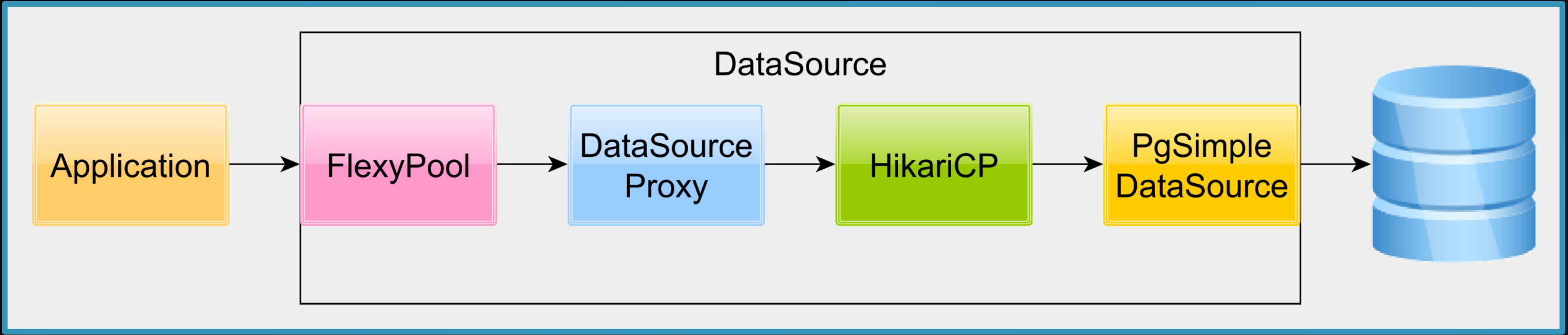
$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

Metric	DB_A (ms)	DB_B (ms)	DB_C (ms)	DB_D (ms)	HikariCP (ms)
min	11.174	5.441	24.468	0.860	0.001230
max	129.400	26.110	74.634	74.313	1.014051
mean	13.829	6.477	28.910	1.590	0.003458
p99	20.432	9.944	54.952	3.022	0.010263

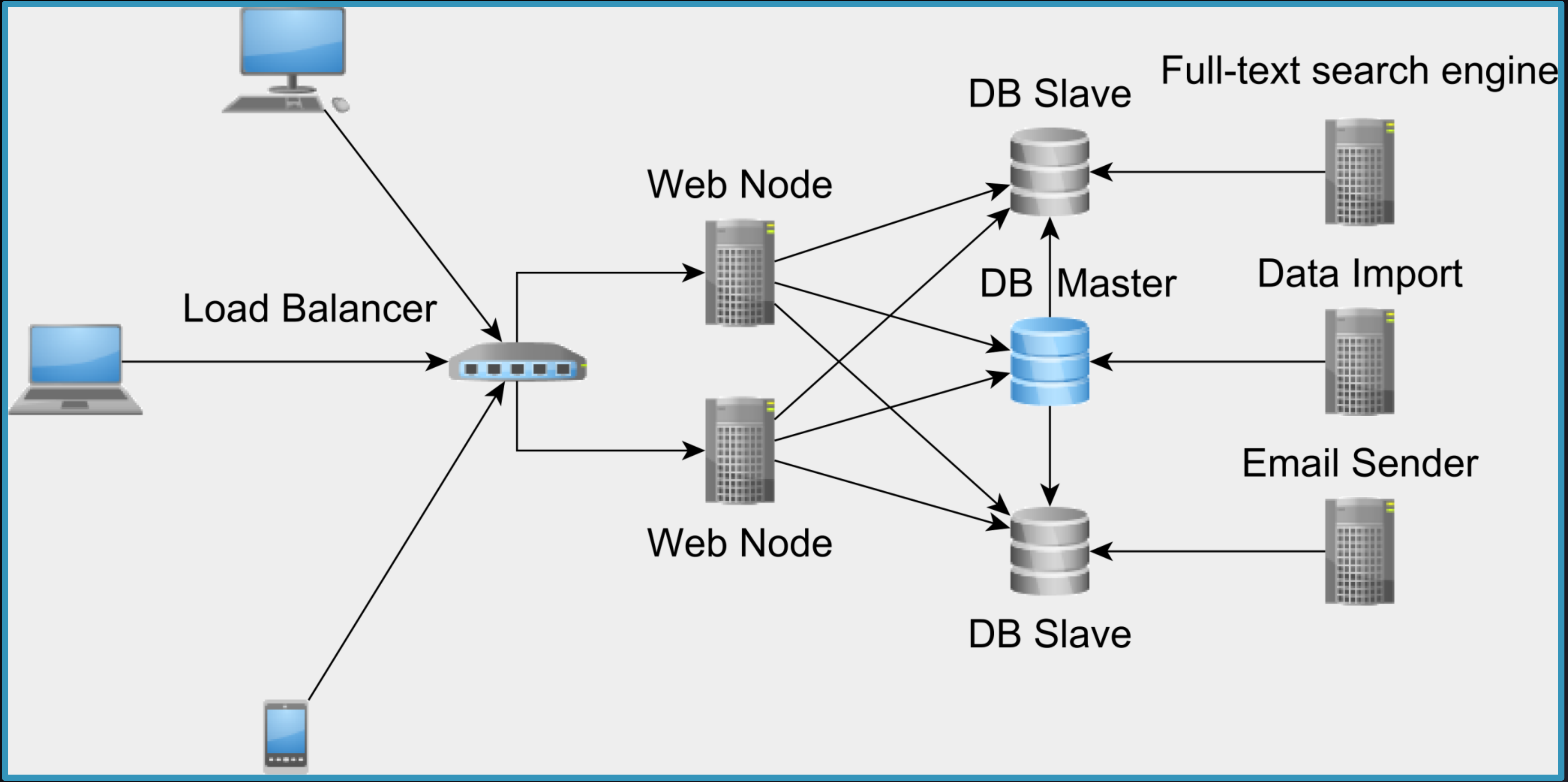
Connection Providers


HikariCPConnectionProvider
C3P0ConnectionProvider
ConnectionProvider
DatasourceConnectionProviderImpl
DriverManagerConnectionProviderImpl

DataSourceConnectionProvider



Connection Provisioning



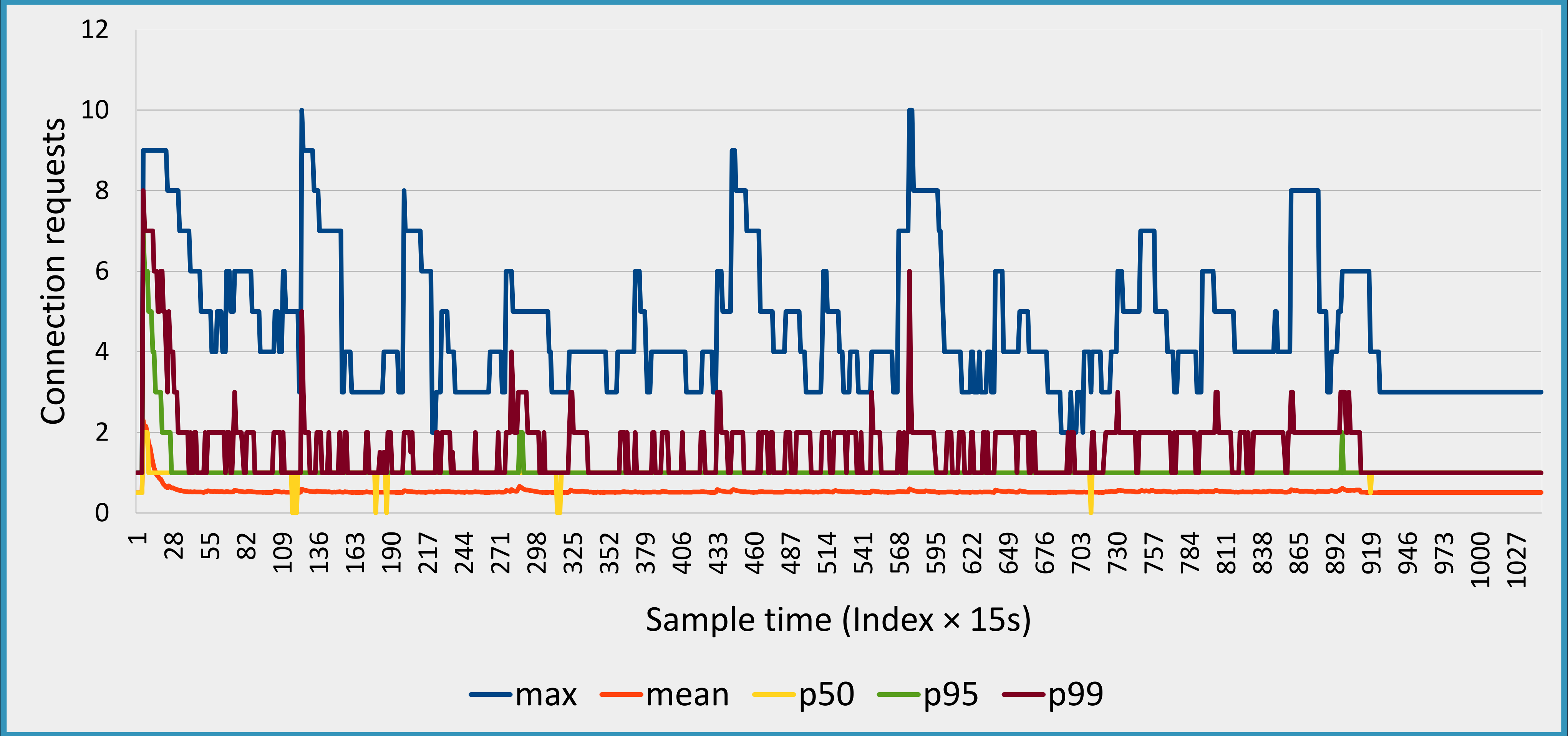
FlexyPool

https://github.com/vladmihalcea/flexy-pool

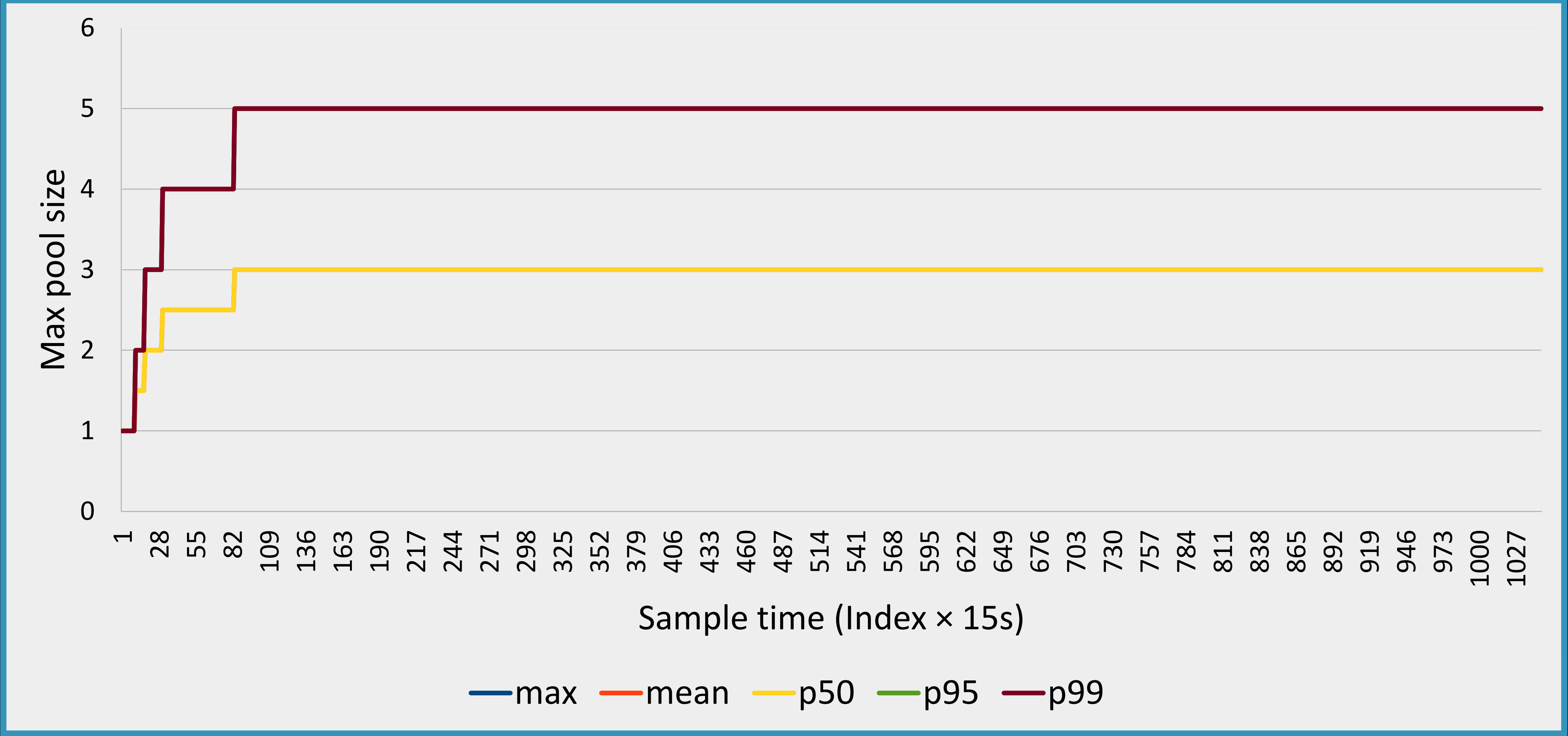
- Java EE
- Bitronix / Atomikos
- Apache DBCP / DBCP2
- C3P0
- BoneCP
- HikariCP
- Tomcat CP
- Vibur DBCP

- concurrent connections
- concurrent connection requests
- connection acquisition time
- connection lease time histogram
- maximum pool size
- overflow pool size
- retries attempts
- total connection acquisition time

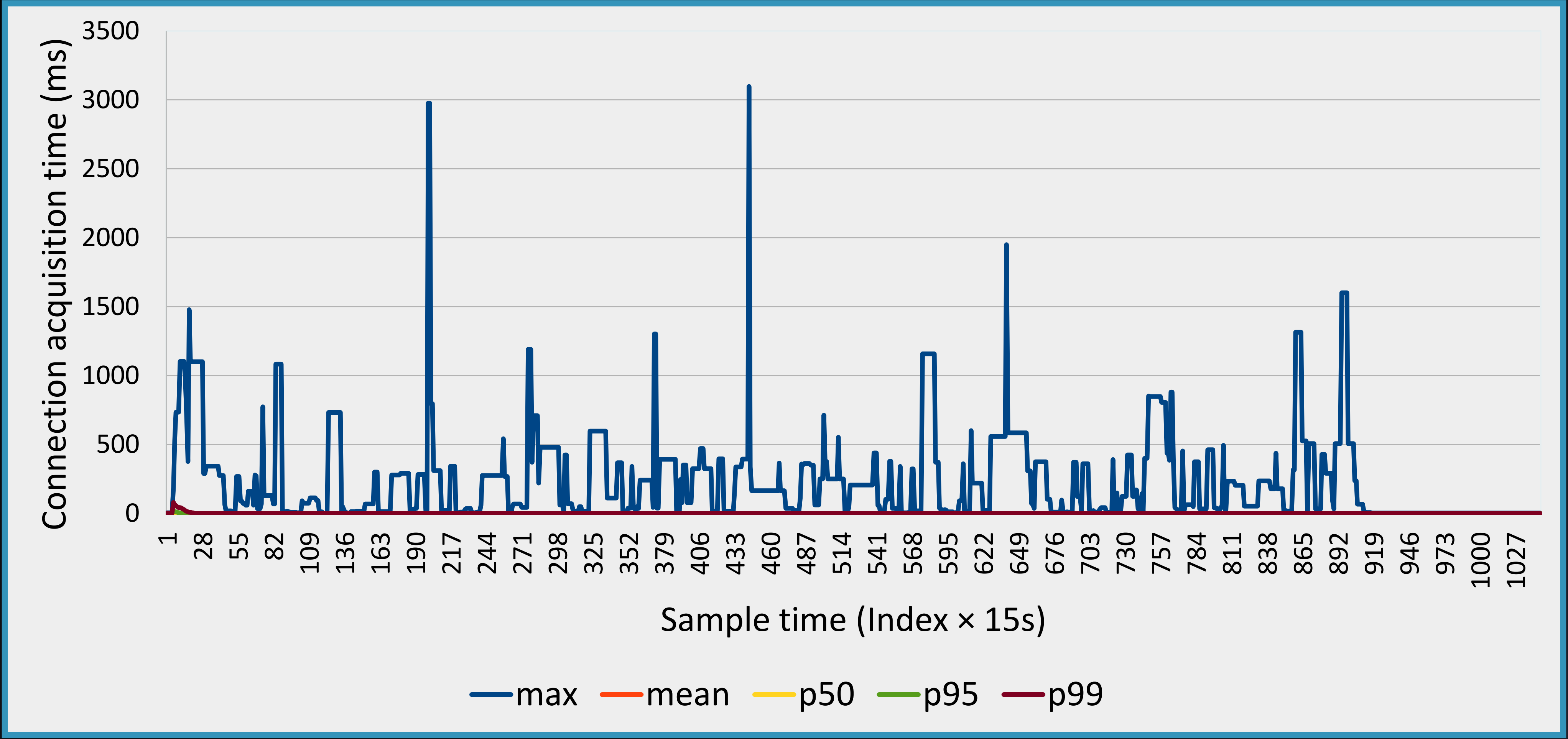
FlexyPool – Concurrent connection requests



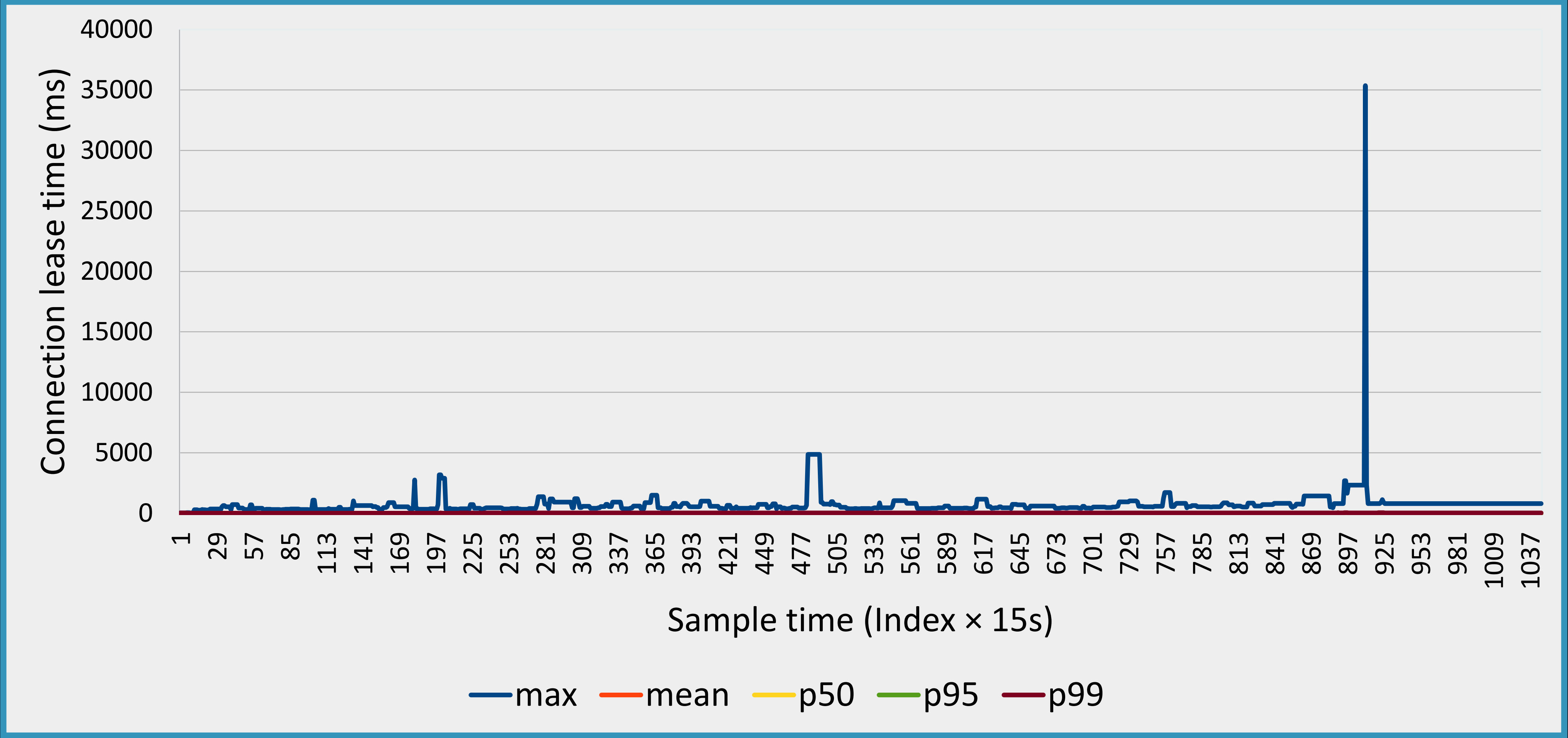
FlexyPool – Pool size growth



FlexyPool – Connection acquisition time



FlexyPool – Connection lease time



Agenda

- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

JPA Identifier Generators

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

- IDENTITY
- SEQUENCE
- TABLE
- AUTO

IDENTITY

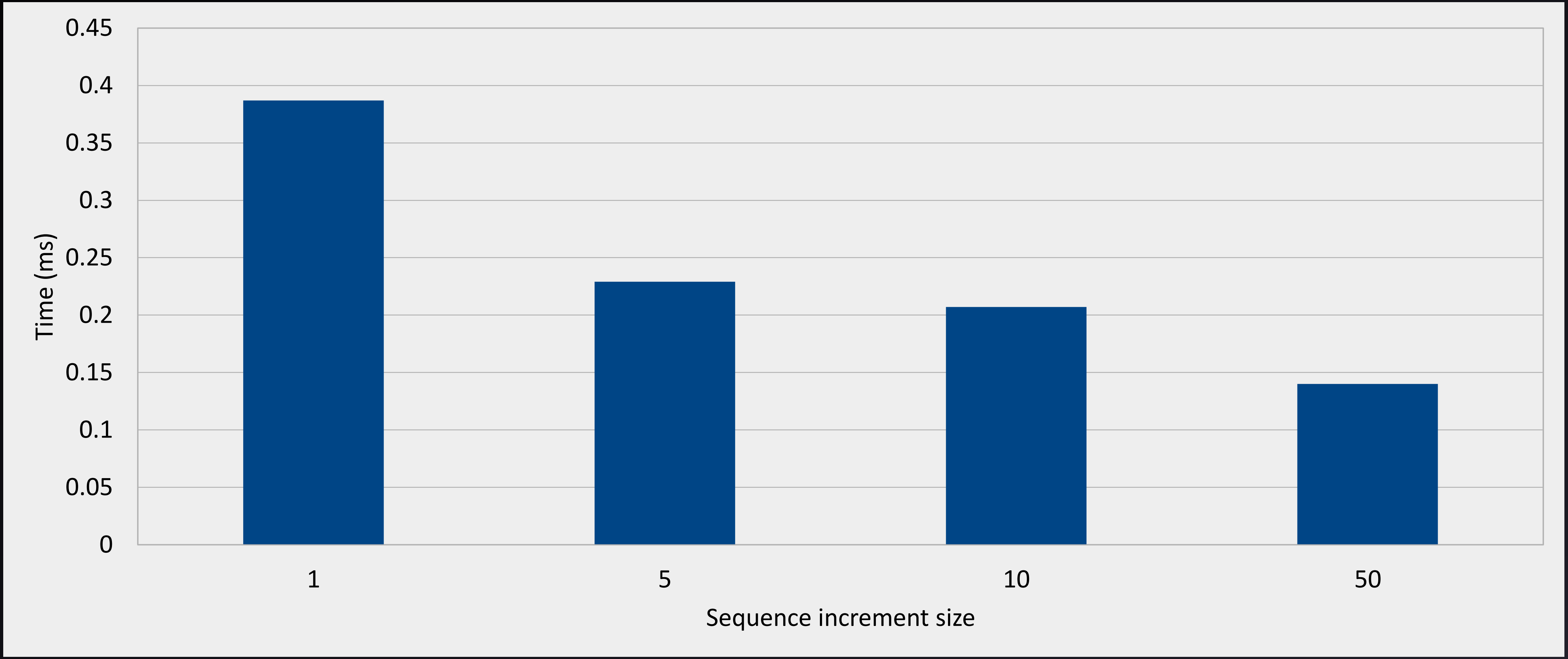
- In Hibernate, IDENTITY generator disables JDBC batch inserts
- MySQL 5.7 does not offer support for database SEQUENCE

SEQUENCE

- Oracle, PostgreSQL, and even SQL Server 2012
- May use roundtrip optimizers: hi/lo, pooled, pooled-lo
- By default, Hibernate 5 uses the enhanced sequence generators

```
<property
    name="hibernate.id.new_generator_mappings"
    value="true"/>
```

SEQUENCE - Pooled optimizer (50 rows)

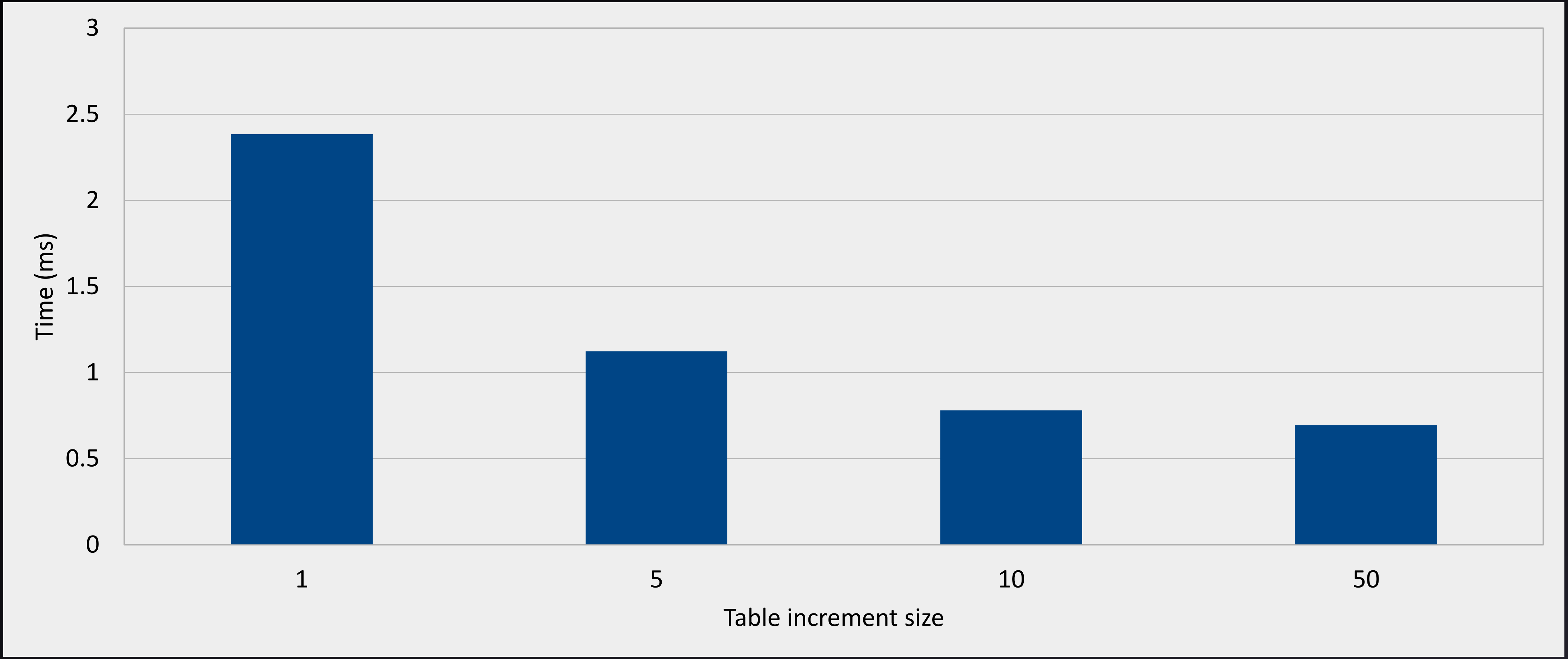


TABLE

- Uses row-level locks and a separate transaction/connection
- May use roundtrip optimizers: hi/lo, pooled, pooled-lo
- By default, Hibernate 5 uses the enhanced sequence generators

```
<property
    name="hibernate.id.new_generator_mappings"
    value="true"/>
```

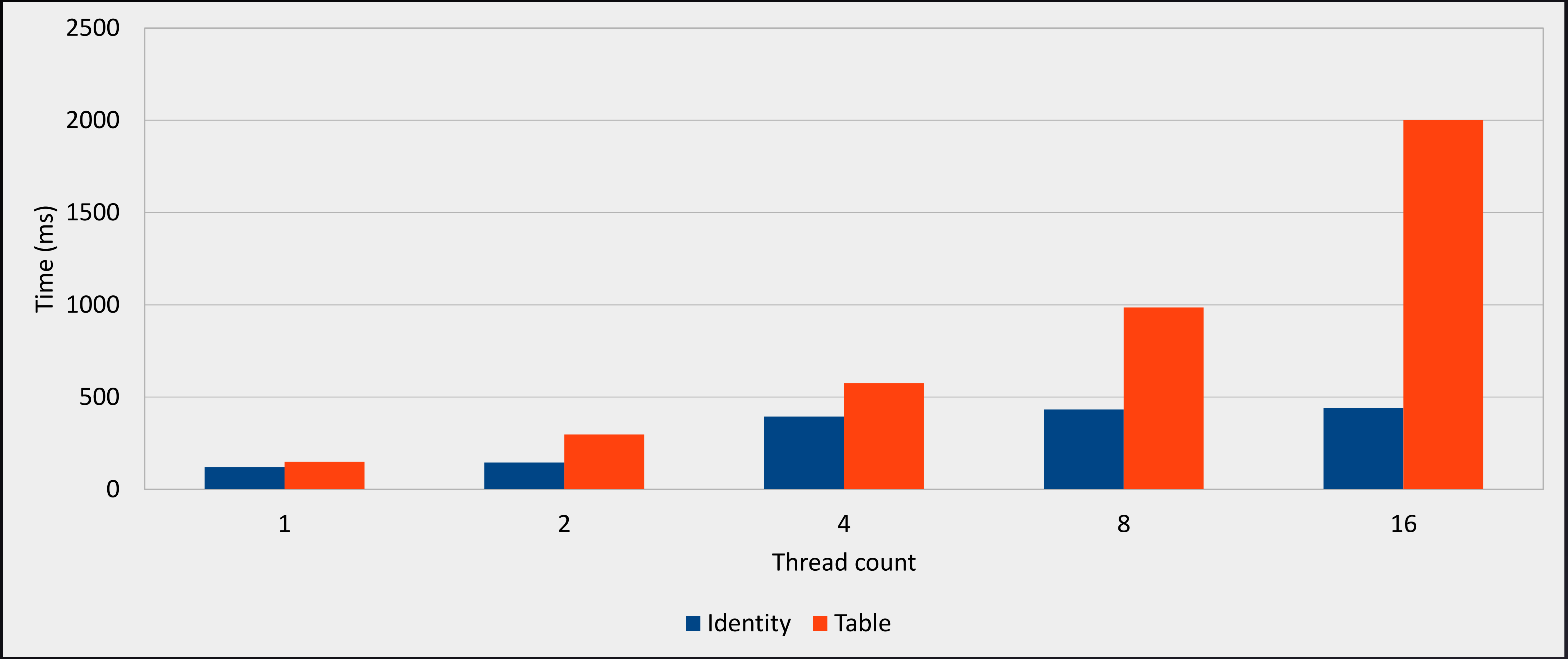
TABLE - Pooled optimizer (50 rows)



IDENTITY vs TABLE (100 rows)

- IDENTITY makes no use of batch inserts
- TABLE generator using a pooled optimizer with an increment size of 100

IDENTITY vs TABLE (100 rows)



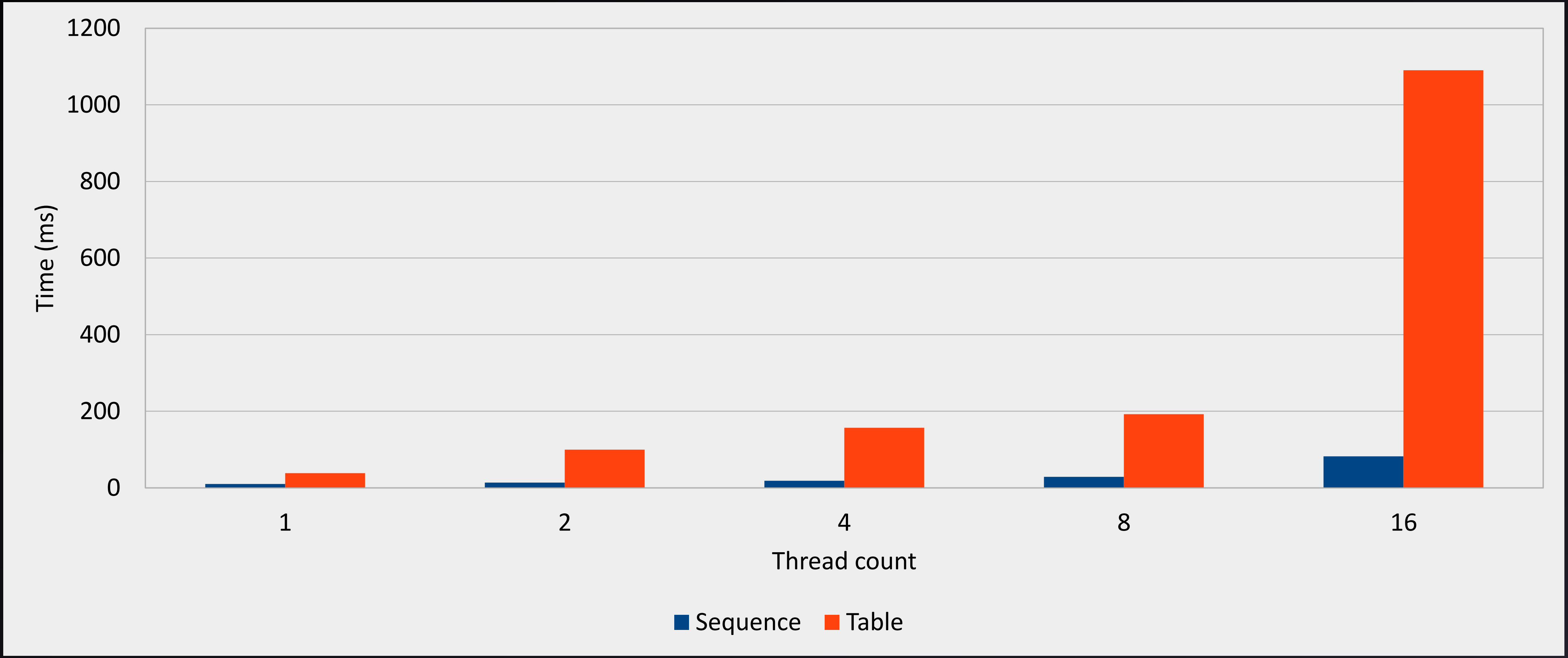
AUTO: IDENTITY vs TABLE?

- Prior to Hibernate 5, AUTO would resolve to IDENTITY if the database supports such a feature
- Hibernate 5 uses TABLE generator if the database does not support sequences

SEQUENCE vs TABLE (100 rows)

- Both benefiting from JDBC batch inserts
- Both using a pooled optimizer with an increment size of 100

SEQUENCE vs TABLE (100 rows)



Agenda

- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

Relationships

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

	Efficient		Less efficient			Least efficient
one-to-many	@ManyToOne	@OneToMany (mappedBy=...)	@OneToMany @JoinColumn	@OneToMany Set<Post>	@OneToMany @OrderColumn (name = ...)	@OneToMany List<Post>
one-to-one	@OneToOne @MapsId	@OneToOne (mappedBy=...) BE	@OneToOne (mappedBy=...)			
many-to-many	@ManyToMany Set<Post>	@ManyToOne @OneToMany	@ManyToMany @OrderColumn(name = ...) List<Post>			@ManyToMany List<Post>

Agenda

- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

Batching

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

- SessionFactory setting
- Session-level configuration since Hibernate 5.2

Batching - SessionFactory

- Switching from non-batching to batching

```
<property
    name="hibernate.jdbc.batch_size"
    value="5"/>
```

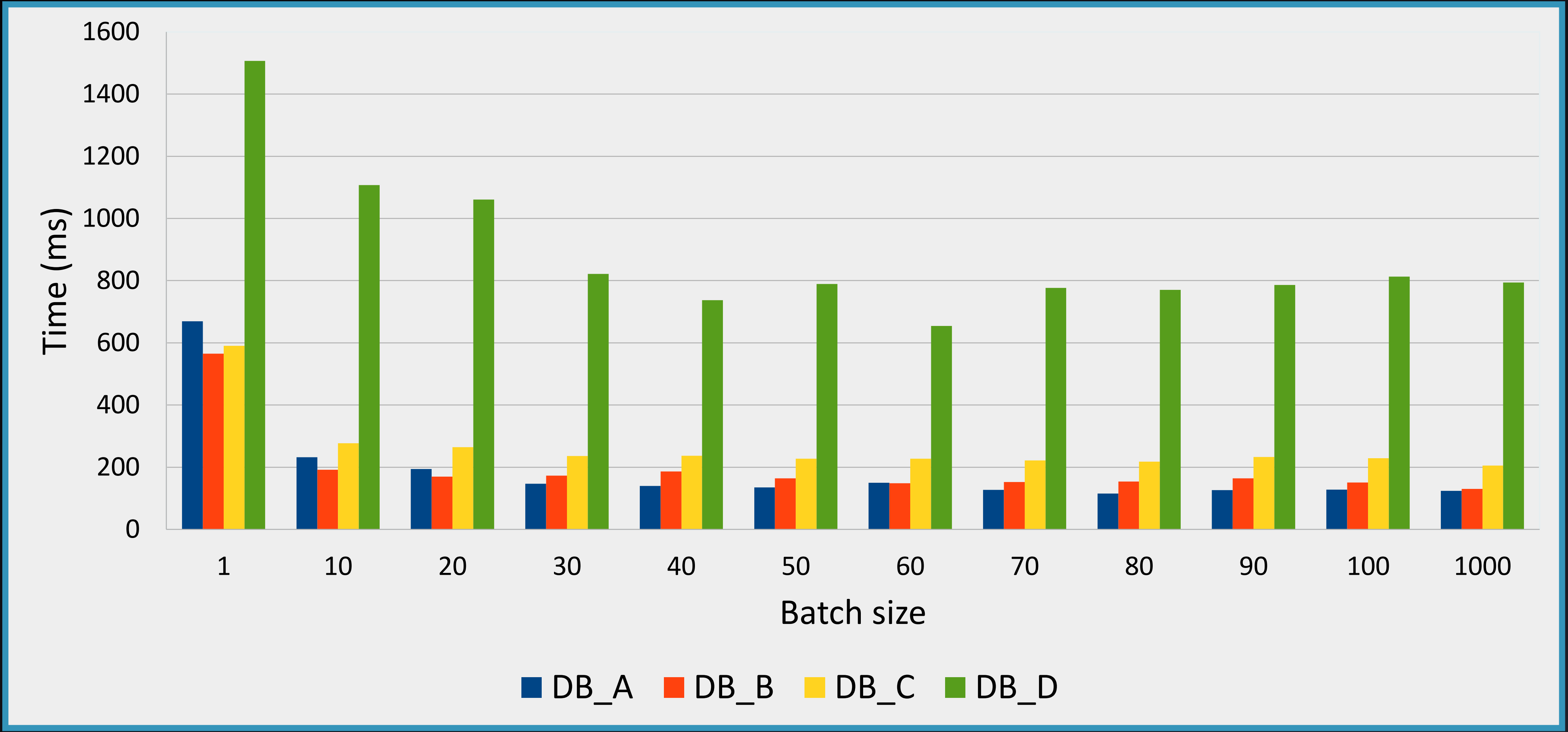
Batching - Session

```
doInJPA( this::entityManagerFactory, entityManager -> {

   entityManager.unwrap( Session.class )
      .setJdbcBatchSize( 10 );

   for ( long i = 0; i < entityCount; ++i ) {
      Person = new Person( i, String.format( "Person %d", i ) );
      entityManager.persist( person );

      if ( i % batchSize == 0 ) {
         entityManager.flush();
         entityManager.clear();
      }
   }
} );
```

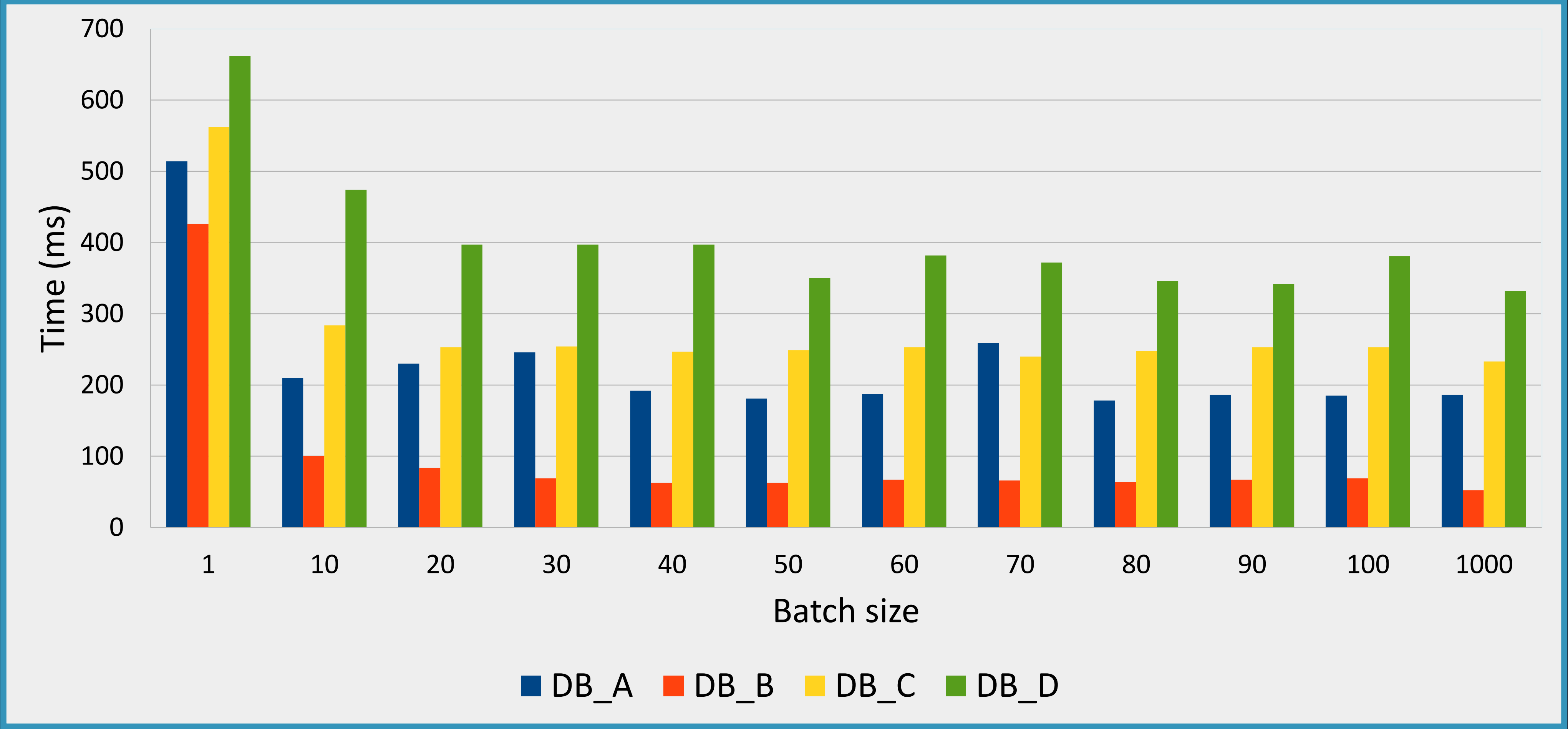
Batching

```
DEBUG [main]: n.t.d.l.SLF4JQueryLoggingListener -
Name:DATA_SOURCE_PROXY,
Time:1,
Success:True,
Type:Prepared,
Batch:True,
QuerySize:1,
BatchSize:10,
Query: ["insert into Person (name, id) values (?, ?)"],
Params:[
(Person 1, 1), (Person 2, 2), (Person 3, 3), (Person 4, 4), (Person 5, 5),
(Person 6, 6), (Person 7, 7), (Person 8, 8), (Person 9, 9), (Person 10, 10)
]
```

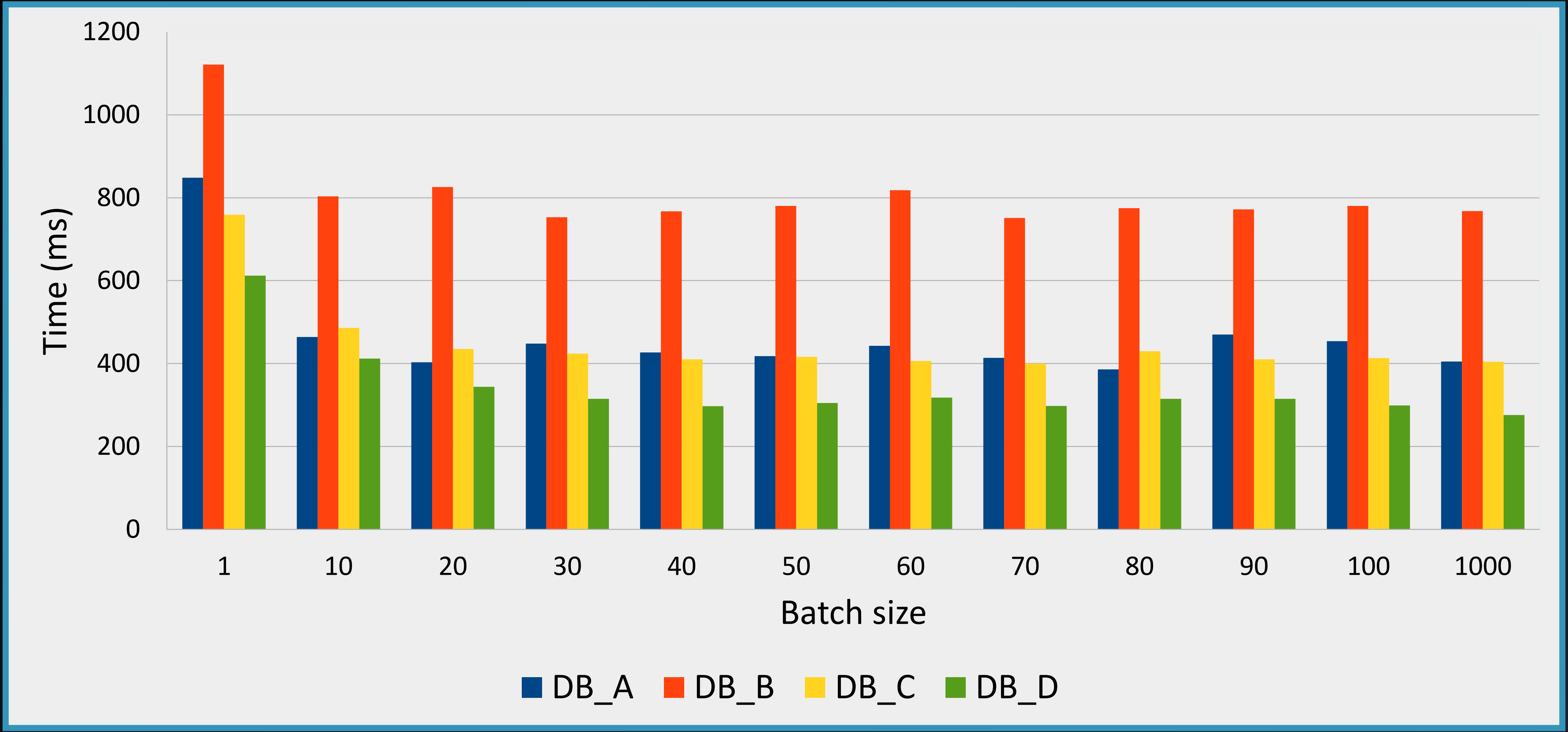
Insert PreparedStatement batching (5k rows)



Update PreparedStatement batching (5k rows)



Delete PreparedStatement batching (5k rows)



Batching - Cascading

```xml
<property
    name="hibernate.order_inserts"
    value="true"/>
```

```xml
<property
    name="hibernate.order_updates"
    value="true"/>
```

Batching – @Version

```
<property
    name="hibernate.jdbc.batch_versioned_data"
    value="true"/>
```

- Enabled by default in Hibernate 5
- Disabled in Hibernate 3.x, 4.x, and for Oracle 8i, 9i, and 10g dialects

Agenda

- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

Fetching

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

- JDBC fetch size
- JDBC `ResultSet` size
- DTO vs Entity queries
- Fetching relationships

Fetching – JDBC Fetch Size

- Oracle – Default fetch size is 10
- SQL Server – Adaptive buffering
- PostgreSQL, MySQL – Fetch the whole `ResultSet` at once
- `SessionFactory` setting:

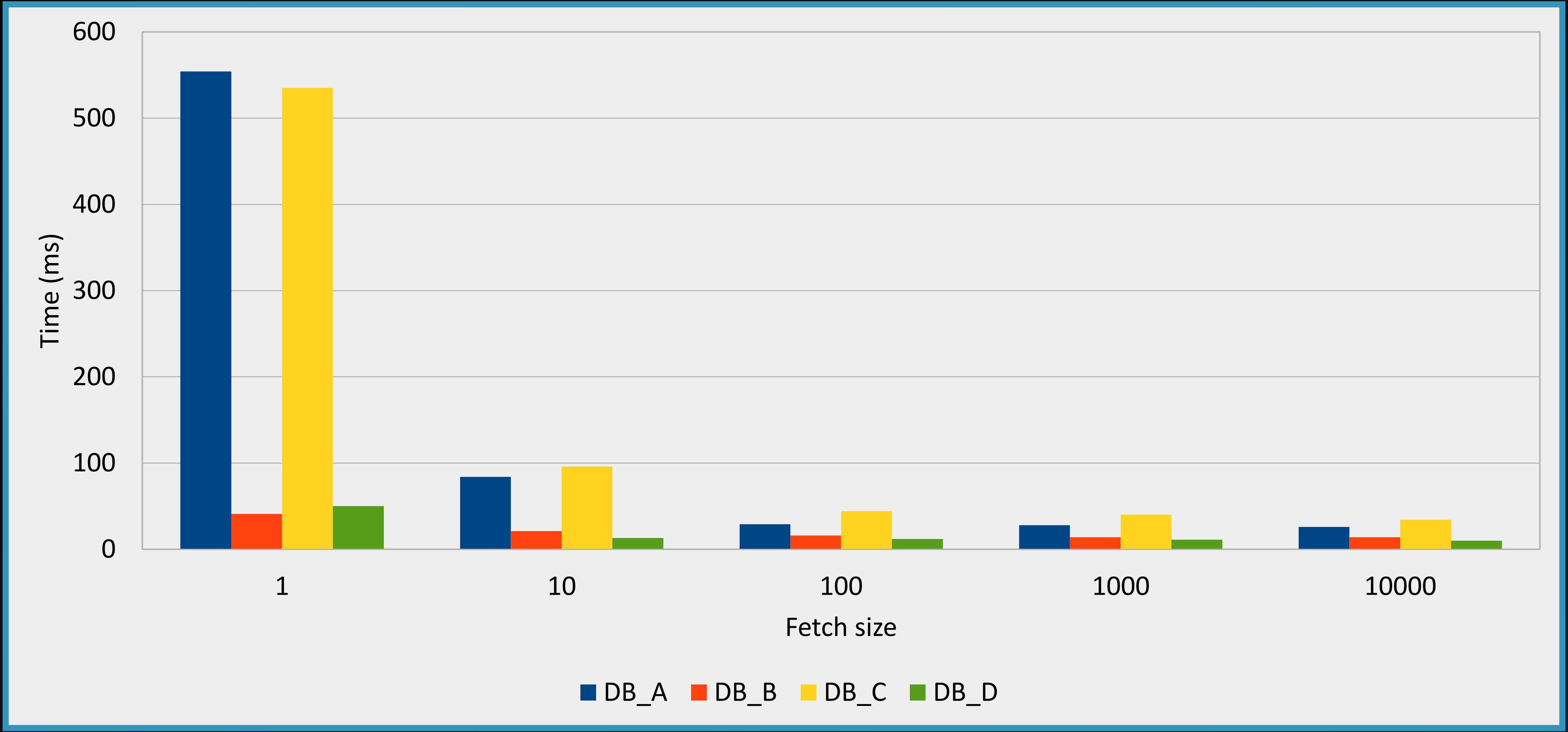
```
<property
    name="hibernate.jdbc.fetch_size"
    value="100"/>
```

Fetching - JDBC fetch size

- Query-level hint:

```
List<PostCommentSummary> summaries =
entityManager.createQuery(
    "select new PostCommentSummary( " +
    "   p.id, p.title, c.review ) " +
    "from PostComment c " +
    "join c.post p")
.setHint(QueryHints.HINT_FETCH_SIZE, fetchSize)
.getResultList();
```

Fetching – JDBC Fetch Size (10k rows)

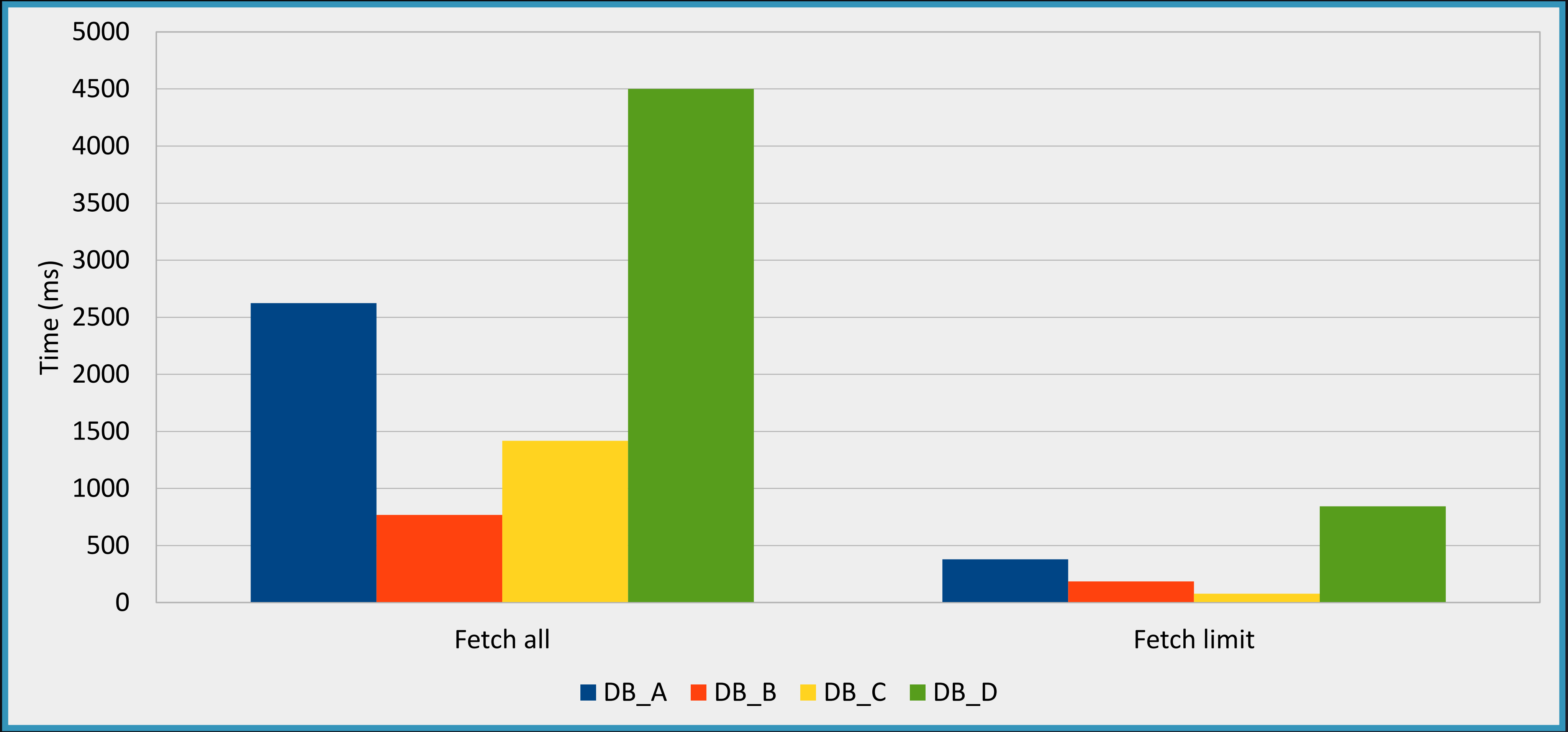


Fetching – Pagination

- JPA / Hibernate API works for both entity and native queries

```
List<PostCommentSummary> summaries =
entityManager.createQuery(
    "select new PostCommentSummary( " +
    "   p.id, p.title, c.review ) " +
    "from PostComment c " +
    "join c.post p")
.setFirstResult(pageStart)
.setMaxResults(pageSize)
.getResultList();
```

Fetching – 100k vs 100 rows



Fetching – Pagination

- Hibernate uses OFFSET pagination
- Keyset pagination scales better when navigating large result sets
- http://use-the-index-luke.com/no-offset

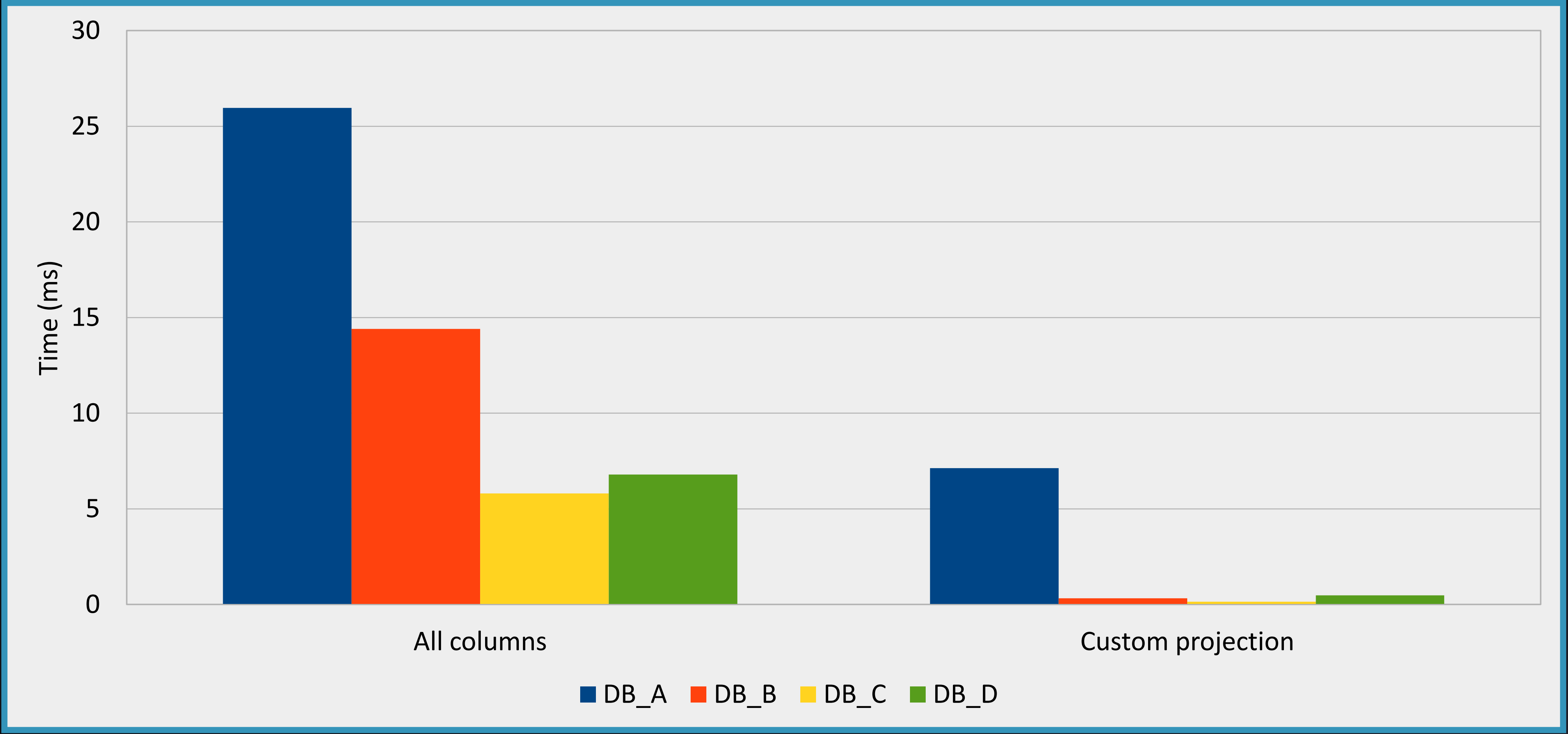
Fetching – Entity vs Projection

- Selecting all columns vs a custom projection

```sql
SELECT *
FROM post_comment pc
INNER JOIN post p ON p.id = pc.post_id
INNER JOIN post_details pd ON p.id = pd.id
```

```sql
SELECT pc.version
FROM post_comment pc
INNER JOIN post p ON p.id = pc.post_id
INNER JOIN post_details pd ON p.id = pd.id
```

Fetching – Entity vs Projection



Fetching – DTO Projections

- Read-only views
- Tree structures (Recursive CTE)
- Paginated Tables
- Analytics (Window functions)

Fetching – Entity Queries

- Writing data
- Web flows / Multi-request logical transactions
- Application-level repeatable reads
- Detached entities / `PersistenceContextType.EXTENDED`
- Optimistic concurrency control (e.g. version, dirty properties)

Fetching – Relationships

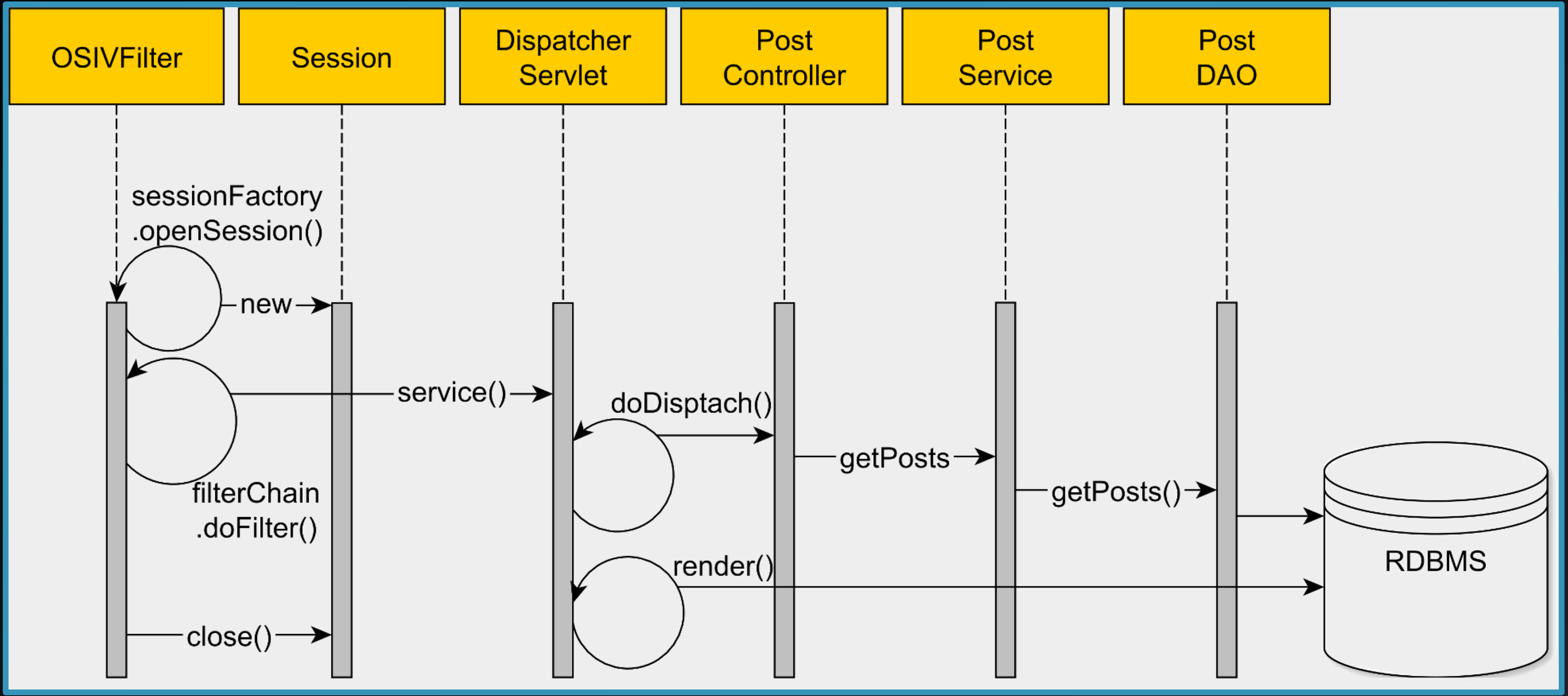
Association	FetchType
@ManyToOne	EAGER
@OneToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

- LAZY associations can be fetched eagerly
- EAGER associations cannot be fetched lazily

Fetching – Best Practices

- Default to `FetchType.LAZY`
- Fetch directive in JPQL/Criteria API queries
- Entity graphs / `@FetchProfile`
- `LazyInitializationException`

Fetching – Open Session in View Anti-Pattern



Fetching – Temporary Session Anti-Pattern

- “Band aid” for LazyInitializationException
- One temporary Session/Connection for every lazily fetched association

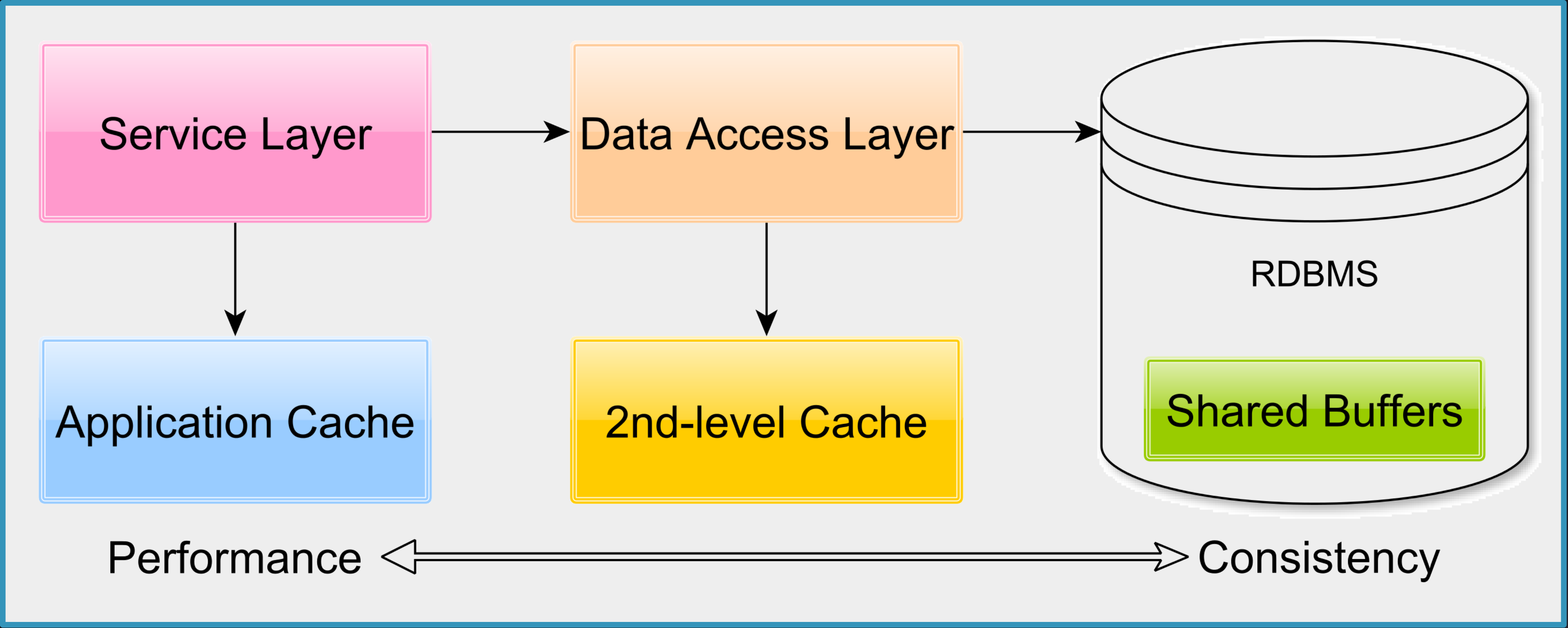
```
<property
    name="hibernate.enable_lazy_load_no_trans"
    value="true"/>
```

Agenda

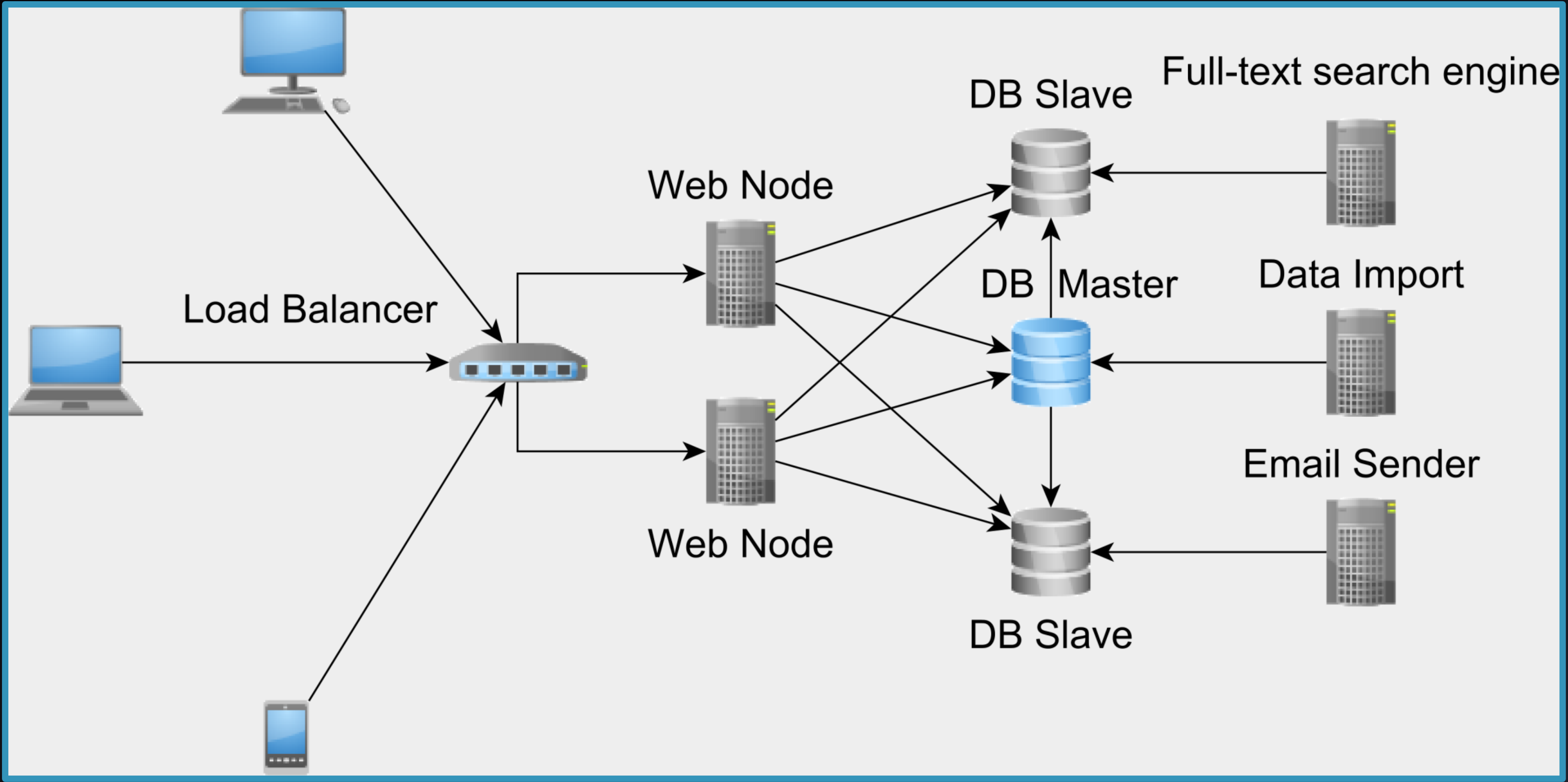
- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching

Caching

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$



Caching – Why 2nd - Level Caching



Caching – Why 2nd - Level Caching

“There are only two hard things in Computer Science: cache invalidation and naming things.”

Phil Karlton

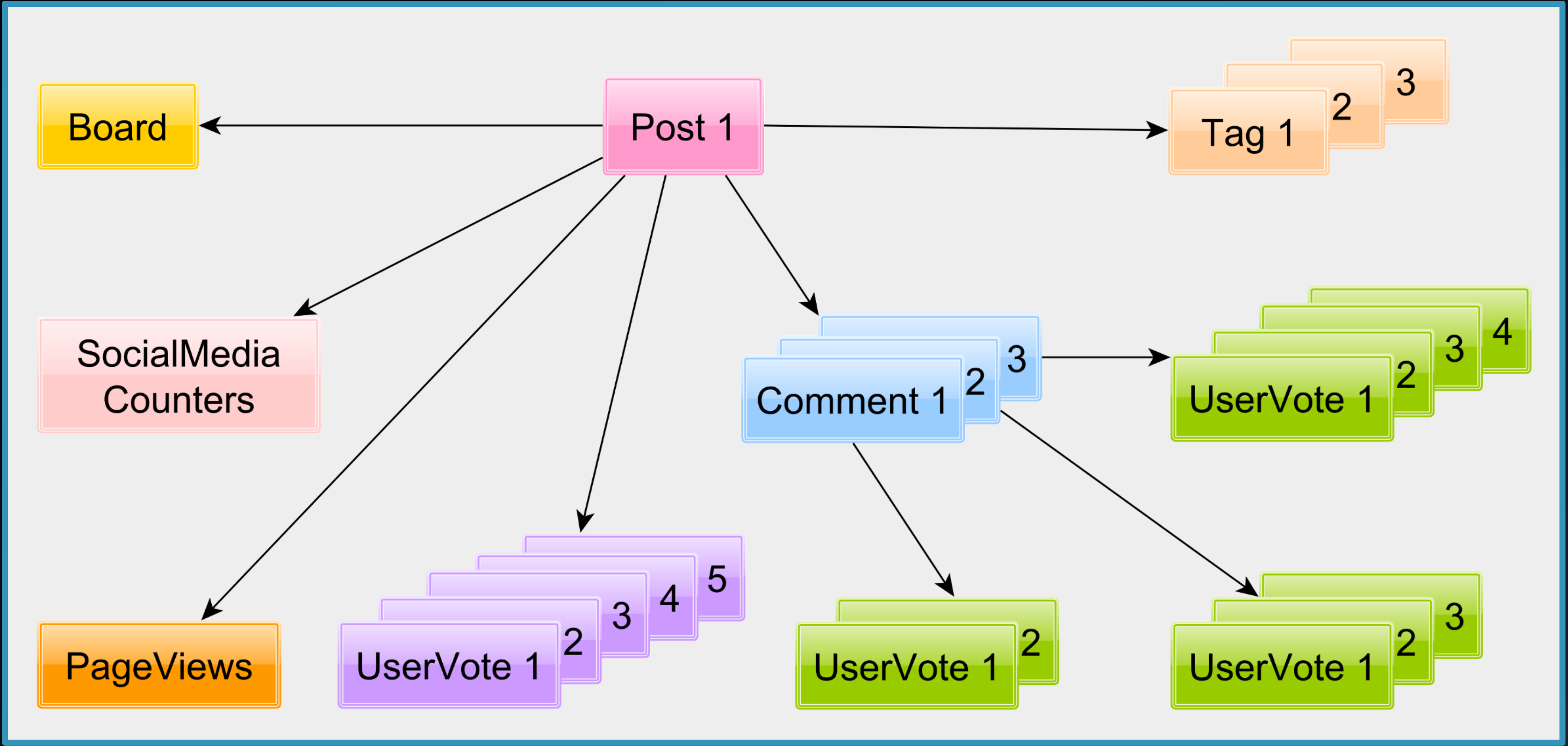
Caching – Strategies

Strategy	Cache type	Particularity
READ_ONLY	READ-THROUGH	Immutable
NONSTRICT_READ_WRITE	READ-THROUGH	Invalidation/ Inconsistency risk
READ_WRITE	WRITE-THROUGH	Soft Locks
TRANSACTIONAL	WRITE-THROUGH	JTA

Caching – Collection Cache

- It complement entity caching
- It stores only entity identifiers
- Read-Through
- Invalidation-based (Consistency over Performance)

Caching – Read - Write Aggregates



See you at Voxxed Days

VOXXED DAYS ZÜRICH

23rd of February 2017

Sihlcity Cinema

amazing speakers!

workshops available

290 CHF

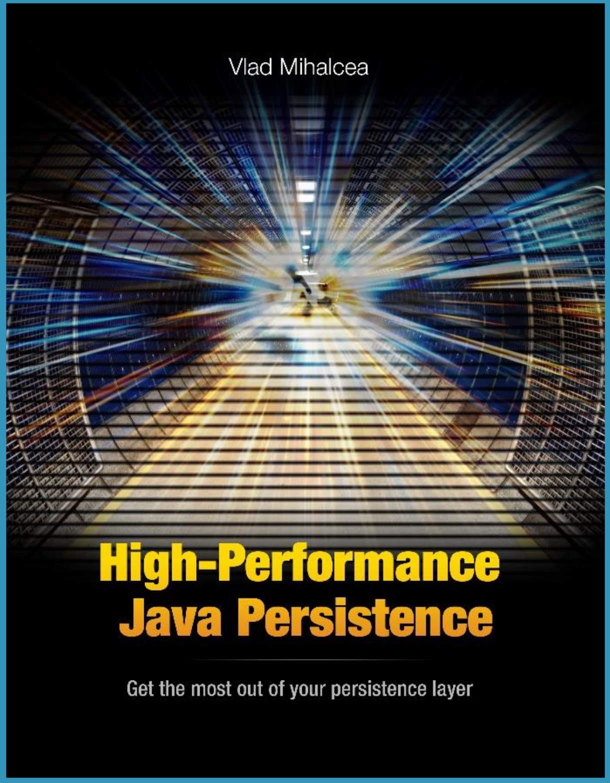
-15% discount using **VDZ17_JUG_CH** code

https://zurich.voxxeddays.com

Questions and Answers

$$T = t_{acq} + t_{req} + t_{exec} + t_{res} + t_{idle}$$

- Performance and Scaling
- Connection providers
- Identifier generators
- Relationships
- Batching
- Fetching
- Caching


Vlad Mihalcea
High-Performance Java Persistence
Get the most out of your persistence layer