accenture

High performance. Delivered.

Parallel processing with Spring Batch

Lessons learned

• Consulting • Technology • Outsourcing

accenture High performance. Delivered.

Morten Andersen-Gott

- Manager at Accenture Norway
- 30 years old
- Been using Spring Batch since 1.0M2 (2007)
- Member of JavaZone program committee
  - http://tinyurl.com/javaever
  - http://tinyurl.com/thestreaming
  - http://tinyurl.com/ladyjava

mortenag

www.github.com/magott

www.andersen-gott.com

A BoF focusing on the problems

- Functional background for the batch
- Short introduction to Spring Batch
- Even shorter on Hibernate
- The problems
- The problems
- The problems

accenture

High performance. Delivered.

Norwegian Public Service Pension Fund (SPK)

- Norway’s main provider of public occupational pensions
- Also provide housing loans and insurance schemes
- Membership of the Norwegian Public Service Pension Fund is obligatory for government employees
- Stats
  - 950,000 members across 1600 organisations
  - Approx 138,000 receive a retirement pension
  - Approx 58,000 receive a disability pension
  - 950,000 members have total accrued pension entitlements in the Norwegian Public Service Pension Fund of 339 billion kroner.

accenture

High performance. Delivered.

Background

- Every year the parliament sets the basic amount of the national insurance
- This amount is a constant used in calculation of all benefits
- When the basic amount is changed, all benefits must be recalculated
- It’s more complex than a constant in an algorithm
  - Rules are result of political games the last 50 years
  - Complex rules
  - Lots of exemptions

accenture

High performance. Delivered.

Execution time requirements

- SPK’s recalculation batch must run
  - After the basic amount is set
  - After the Labour and Welfare Administration has done it’s calculation
  - Before the pensions are due next month
- Window of 1 week
  - Will ideally only run during weekends
  - Can not run while case workers are doing their job

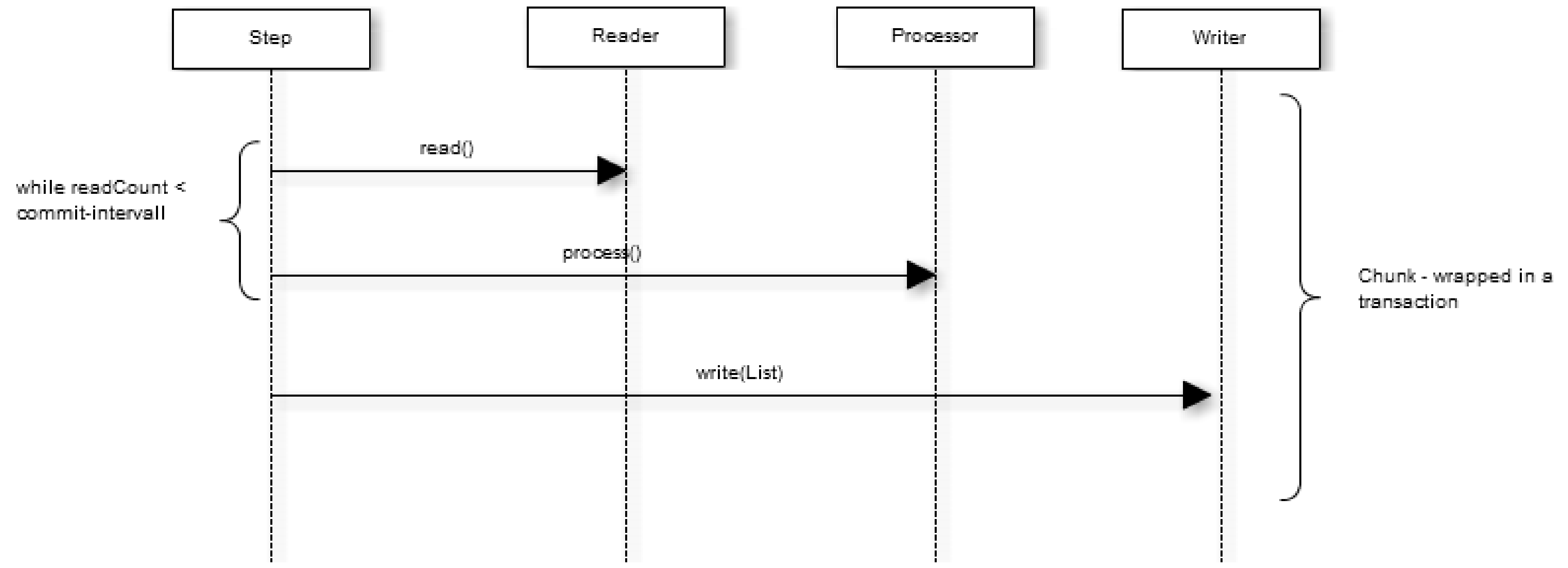
Spring Batch

- Framework for developing batch applications
- Implements batch semantics
  - Steps, Chunks, Stop, Restart, Skips, Retries
  - Partitioning, Multithreading, Parallel steps

accenture

High performance. Delivered.

A step…


Step
Reader
Processor
Writer
read()
process()
write(List)
while readCount < commit-intervall
Chunk - wrapped in a transaction

accenture

High performance. Delivered.

The components

- ItemReader
  - `read()` returns one row at the time
  - Step is completed once `read()` returns `null`
- ItemProcessor
  - `process(item)` item is return value from `read()`
  - Business logic goes here
  - Items can be filtered out by returning `null`
- ItemWriter
  - `Write(list)` list of items returned from `process()`

High performance. Delivered.

The code

```
<job id=”foo”>
    <step id="fileImport">
      <tasklet>
           <chunk commit-interval="10"
             reader=“reader"
             processor=“processor"
             writer=“writer“/>
      </tasklet>
    </step>
</job>
<bean id=“reader" class="...">
<bean id=“processor" class="...">
<bean id=“writer" class="...">
```

accenture

High performance. Delivered.

Chunk

- A chunk is a unit of work
  - Executes within a transaction
  - Size is defined by number of items read
- A step is divided into chunks by the framework
- When x is the chunk size
  - read() is called x times
  - process() is called x times
  - write is called once with a list where list.size()==x (minus filtered items)

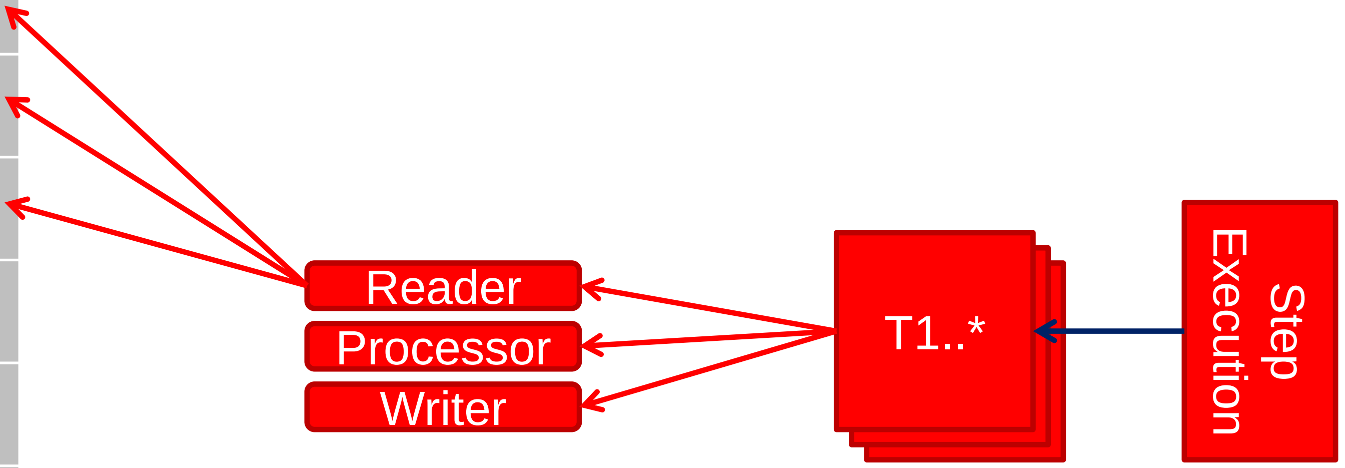
High performance. Delivered.

Scaling

High performance. Delivered.

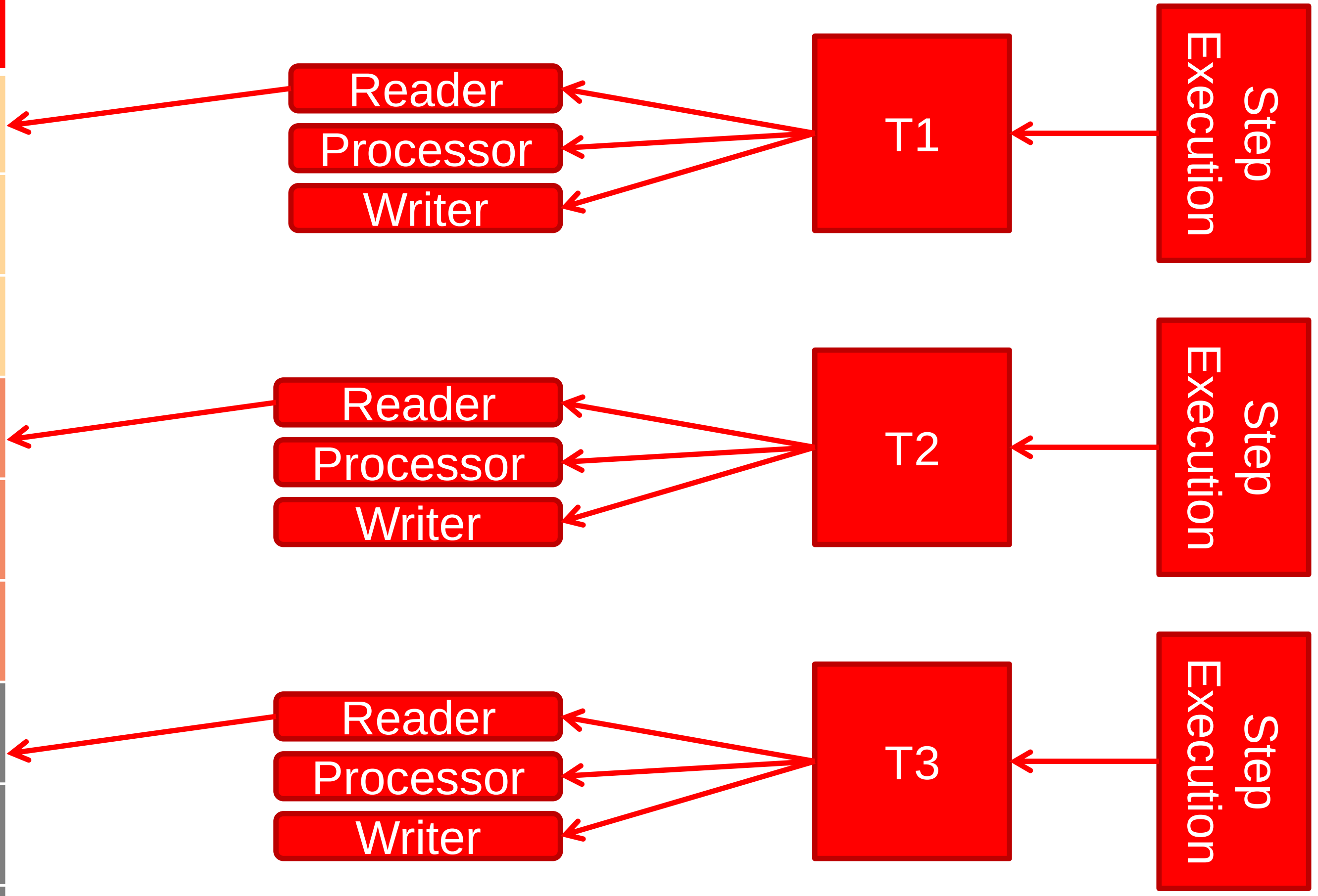
Multi-threaded step

Id	Name
1	Paul
2	John
3	Lisa
4	Simon
5	Rick
6	Julia
7	Olivia
8	Scott
9	Martin



Partitioned Step

Id	Name
1	Paul
2	John
3	Lisa
4	Simon
5	Rick
6	Julia
7	Olivia
8	Scott
9	Martin



accenture
High performance. Delivered.

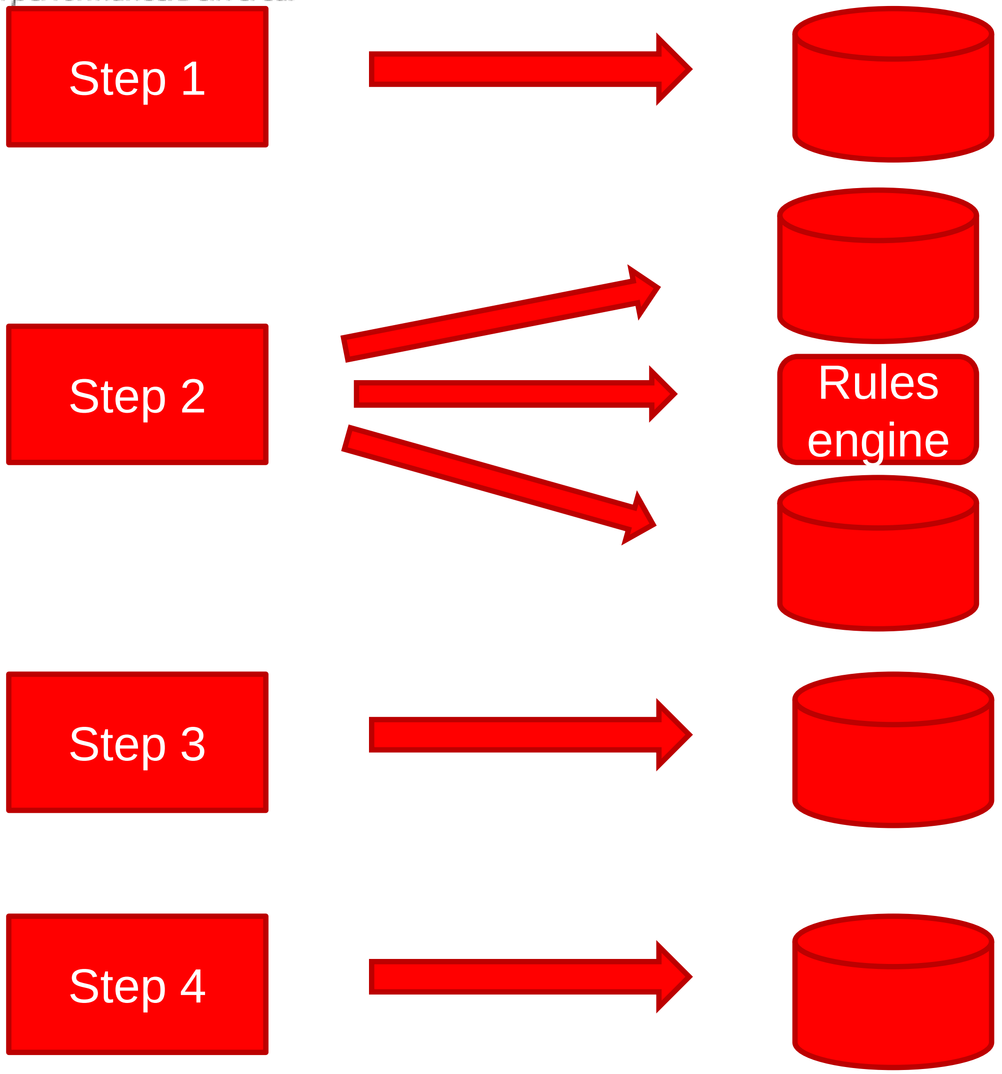
Hibernate 101

- Proxy
- Session cache
- Flushing
  - Queries
  - Commit

accenture

High performance. Delivered.

The pension recalculation batch



- Stored procedure
Populate staging table

- Read from staging
- Fetch additional data
- Call out to rules engine
- Store result as pending pensions

- Loop through pending pensions and activate

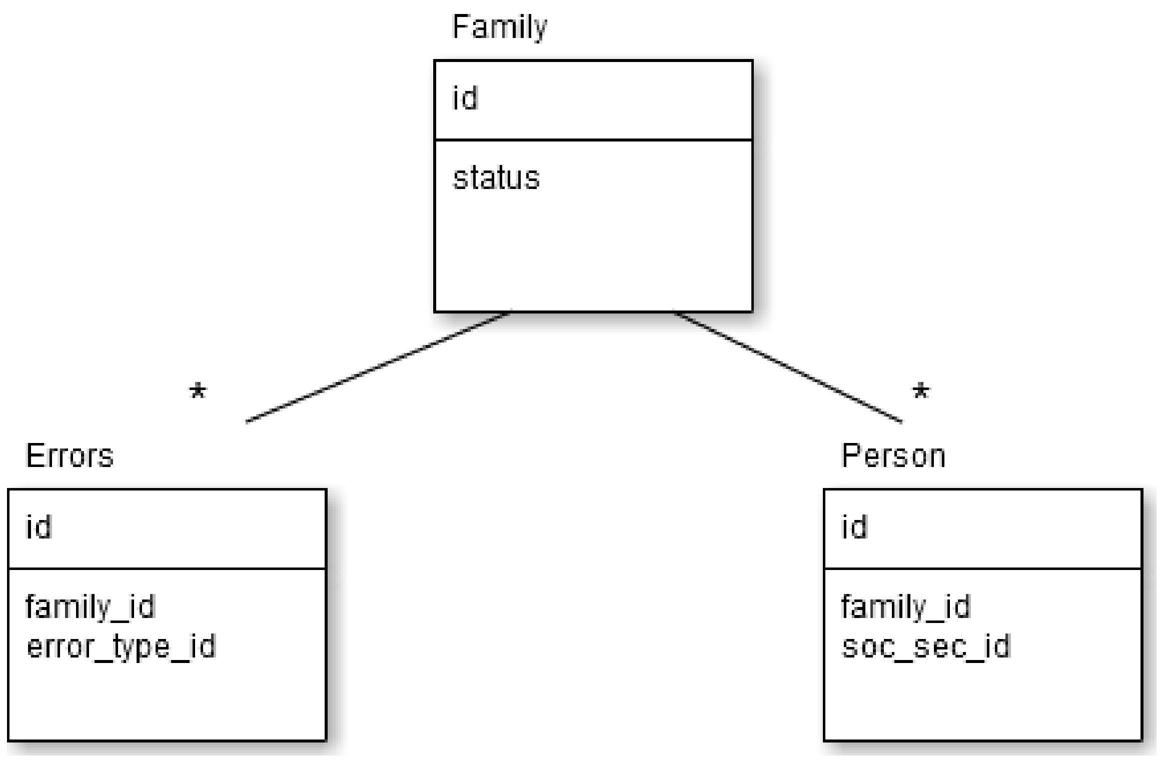
- Create manual tasks for failed items

High performance. Delivered.

Staging table

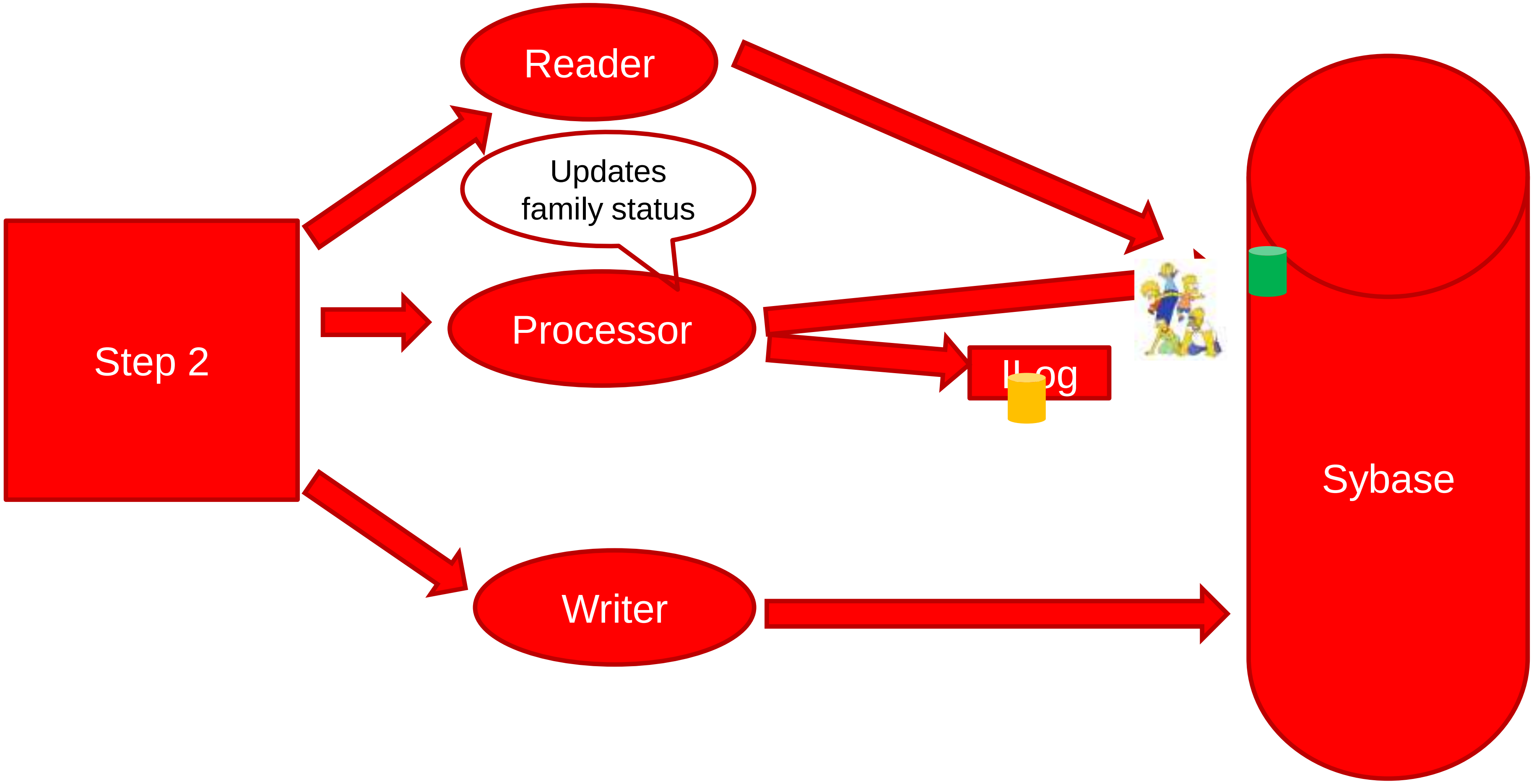
- A batch processing pattern
- A simple table with a identity column, functional key and processing status
- Restart is easy
  - Select soc_sec from staging where processed=false
- An easy way to get parallel processing capabilities

Our staging table(s)



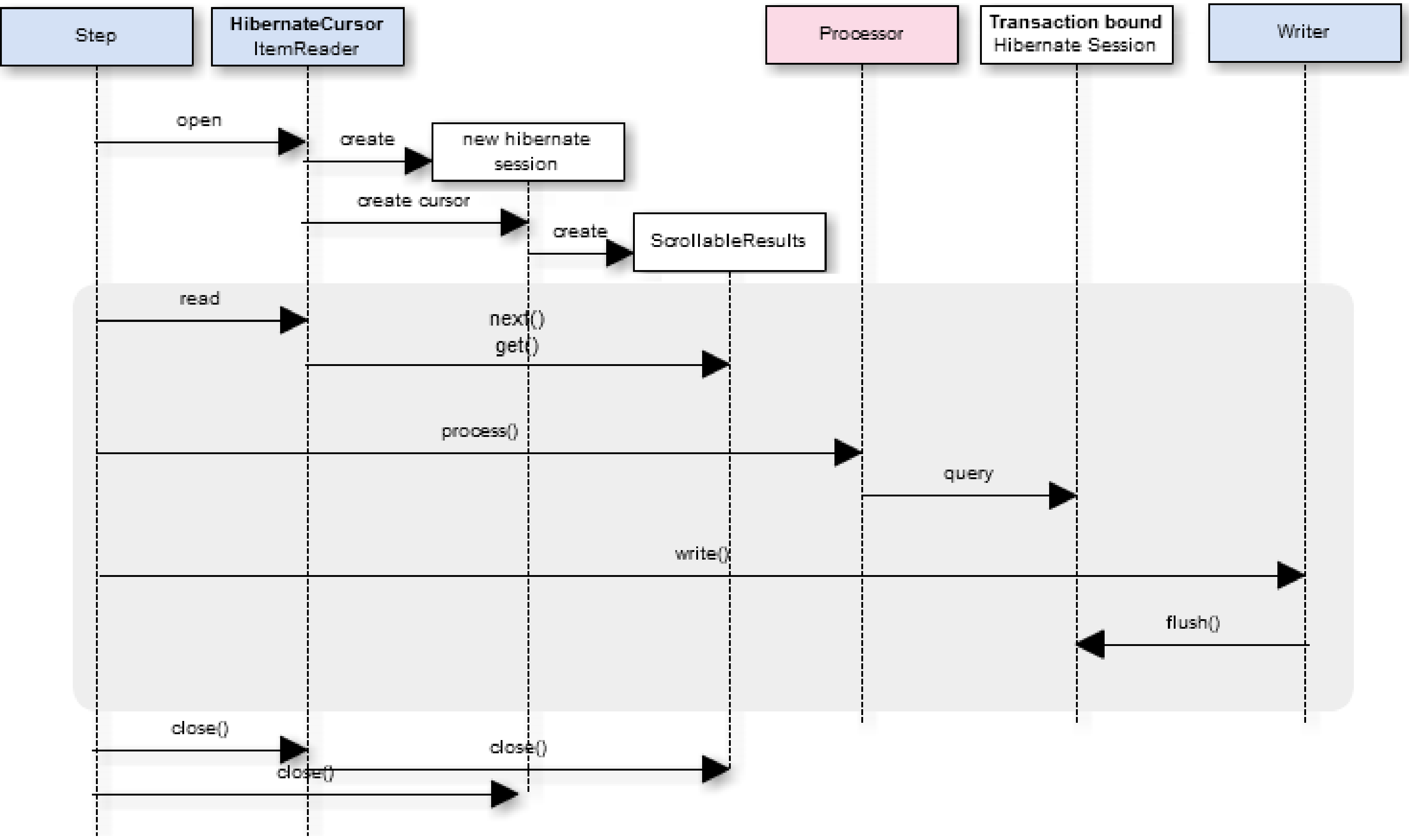
accenture

High performance. Delivered.



Hibernate in the reader

- Using a separate statefull session in reader
  - Not bound to active transaction
  - Is never flushed
  - clear() is called before commit
- Entities are written to database using a (different) transaction bound session
  - Used by processor and writer


Step
HibernateCursor ItemReader
Processor
Transaction bound Hibernate Session
Writer
open
create
new hibernate session
create cursor
create
ScrollableResults
read
next()
get()
process()
query
write()
flush()
close()
close()
close()

High performance. Delivered.

Problems?

accenture

High performance. Delivered.

What happened?

org.hibernate.HibernateException: Illegal attempt to associate a collection with two open sessions

Why

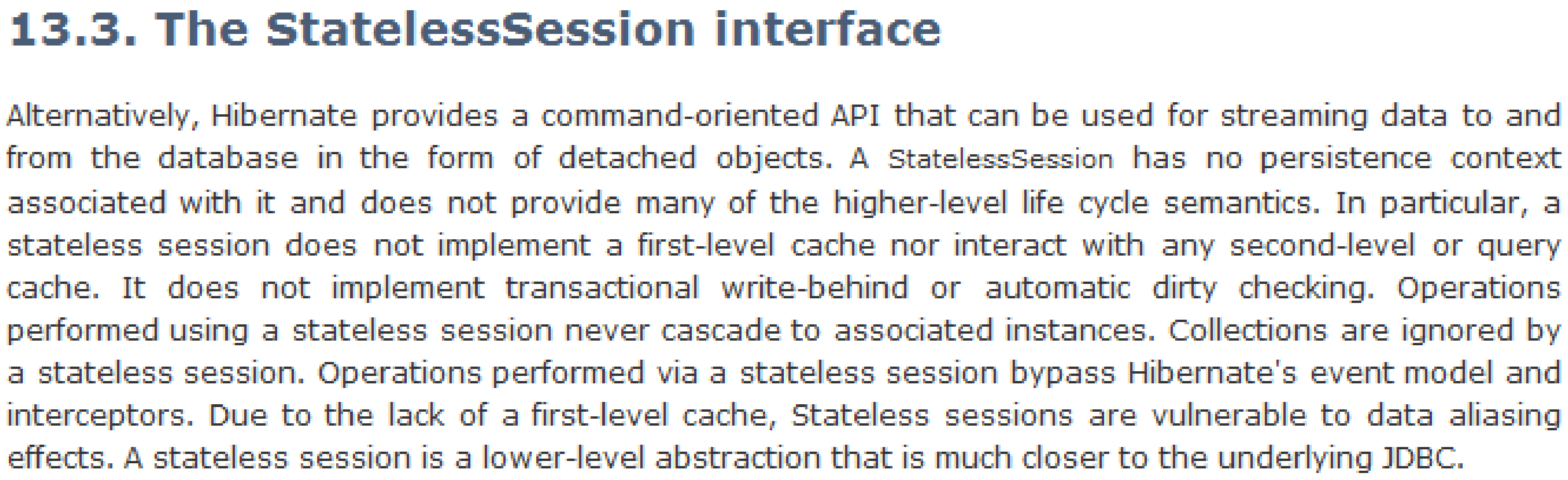
- Family.errors and Family.persons are attached to reader’s session
- Attempting to attach them to transaction bound session
- Hibernate will have non of that!

accenture

High performance. Delivered.

The solution?

Stateless sessions



13.3. The StatelessSession interface

Alternatively, Hibernate provides a command-oriented API that can be used for streaming data to and from the database in the form of detached objects. A `StatelessSession` has no persistence context associated with it and does not provide many of the higher-level life cycle semantics. In particular, a stateless session does not implement a first-level cache nor interact with any second-level or query cache. It does not implement transactional write-behind or automatic dirty checking. Operations performed using a stateless session never cascade to associated instances. Collections are ignored by a stateless session. Operations performed via a stateless session bypass Hibernate's event model and interceptors. Due to the lack of a first-level cache, Stateless sessions are vulnerable to data aliasing effects. A stateless session is a lower-level abstraction that is much closer to the underlying JDBC.

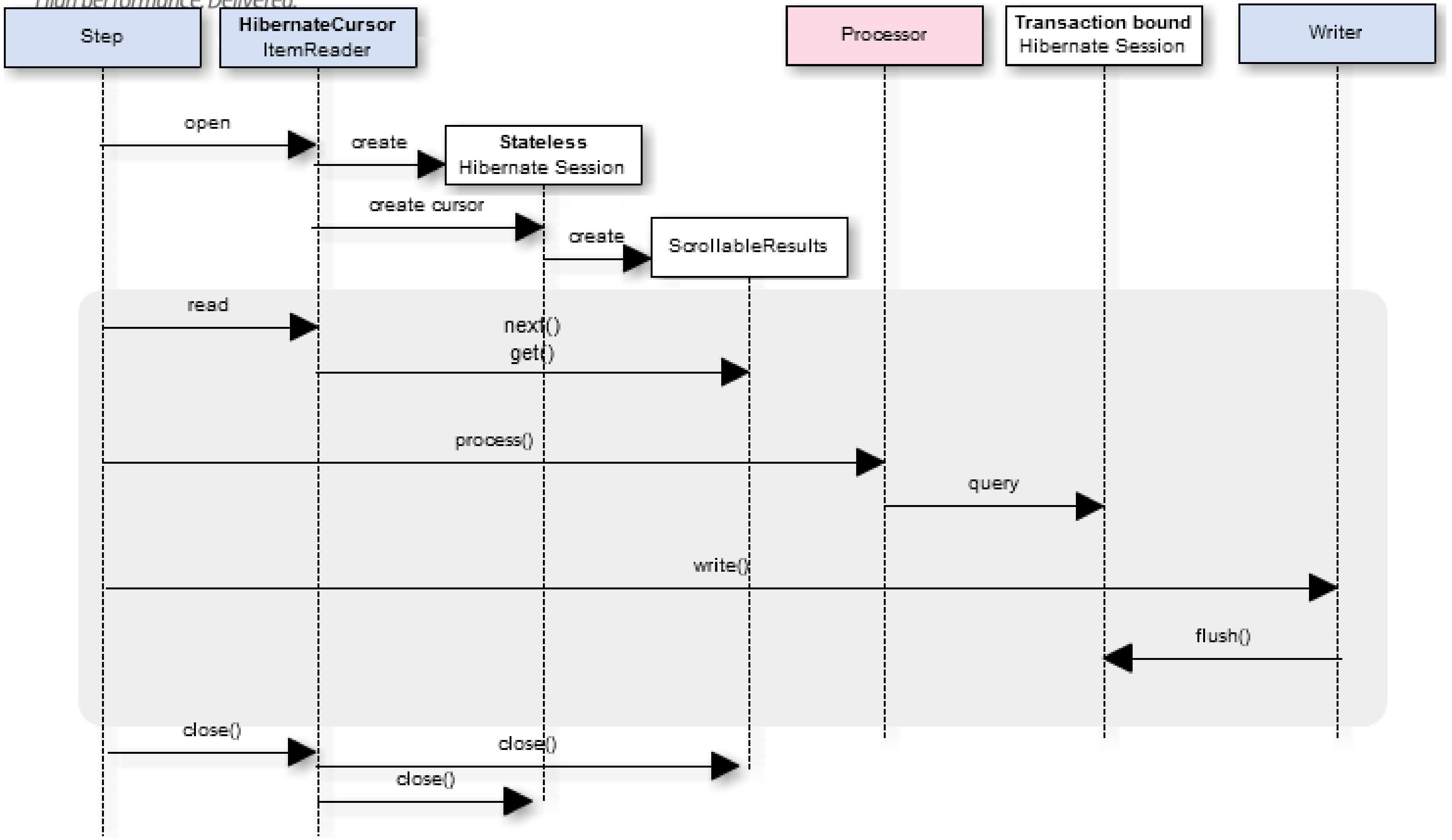
Hibernate in the reader
(Second attempt)

accenture

High performance. Delivered.

- Let’s try stateless session
- Default behavior in Spring Batch’s Hibernate ItemReaders
  - useSateless=”true”
- LEFT JOIN FETCH for eager loading collections
  - Avoiding LazyLoadingExceptions

Hibernate in the reader (Second attempt)



High performance. Delivered.

Problems?

What happened?

- org.hibernate.HibernateException: cannot simultaneously fetch multiple bags

accenture

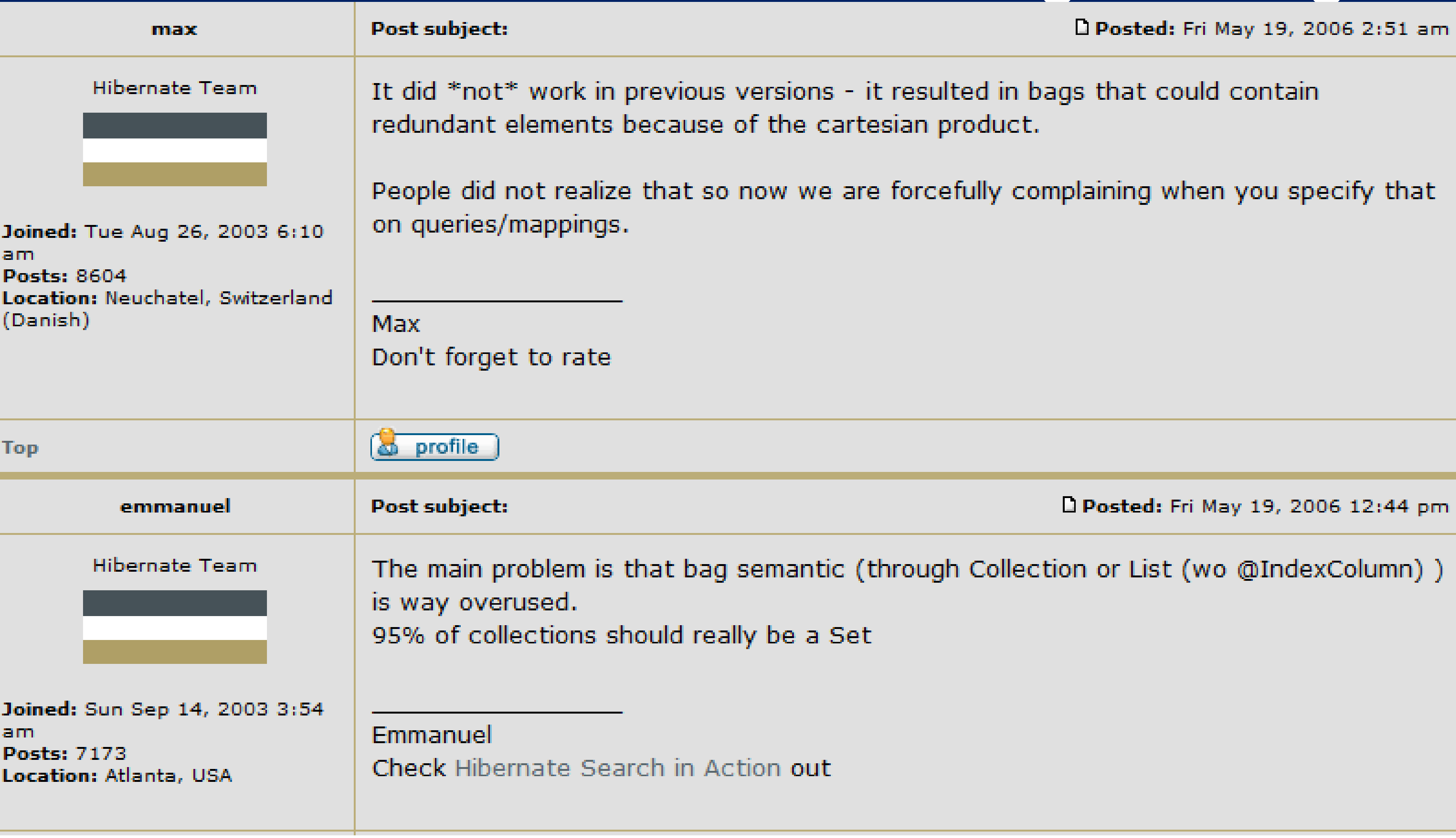
High performance. Delivered.

Why

- Hibernate is unable to resolve the Cartesian product
  – Throws exception to avoid duplicates

accenture

You are using it wrong!



max

Hibernate Team

Joined: Tue Aug 26, 2003 6:10 am
Posts: 8604
Location: Neuchatel, Switzerland (Danish)

Post subject: **Posted:** Fri May 19, 2006 2:51 am

It did *not* work in previous versions - it resulted in bags that could contain redundant elements because of the cartesian product.

People did not realize that so now we are forcefully complaining when you specify that on queries/mappings.

Max
Don't forget to rate

Top profile

emmanuel

Hibernate Team

Joined: Sun Sep 14, 2003 3:54 am
Posts: 7173
Location: Atlanta, USA

Post subject: **Posted:** Fri May 19, 2006 12:44 pm

The main problem is that bag semantic (through Collection or List (wo @IndexColumn)) is way overused.
95% of collections should really be a Set

Emmanuel
Check Hibernate Search in Action out

accenture

High performance. Delivered.

Curing the *you are using it wrong* syndrome

Hibernate in the reader (The third attempt)

- Examine the object graph
- Replace List with Set
  - Only one eagerly loaded collection may be of type list List
- This works…
  - ..for a while
  - We'll revisit Hibernate later…

accenture

High performance. Delivered.

Exception resilience

- New demands sneak in
  - The batch should not abort
    - Not under any circumstance
- The batch should deal with
  - Missing or functionally corrupt data
  - Programming errors
  - ...

Pseudo code

```
try{
  //do data and business operations
}catch(Exception e){
   //Add error to staging & continue
   family.addError(createError(e));
}
```

accenture

High performance. Delivered.

Problems?

Overzealous exception handling

an assertion failure occurred (this may indicate a bug in Hibernate, but is more likely due to unsafe use of the session)

Overzealous exception handeling
The solution

High performance. Delivered.

- Some exception MUST result in a rollback
  - Ex. StaleStateException
- Configure the framework to do retry/skip for these
- Only catch exceptions you **know** you can handle in a meaningful way
  - Nothing new here
  - Do not succumb to crazy requirements

Time to turn on parallelization

- We chose partitioned over mutli-threaded step
  - No need for a thread safe reader
    - Step scope
    - Each partition get a new instance of reader
  - Page lock contentions are less likely
    - Row 1 in partition 1 not adjacent to row 1 in partition 2

Deadlocks

- Legacy database using page locking
- Normalized database
- Relevant data for one person is spread across a number of tables
- Different threads **will** access same data pages
- Deadlocks **will** occur

High performance. Delivered.

Page locking

ID	NAME
1	Paul T1
2	John T2 waiting
3	Simon
4	Scott T1 waiting
5	Lisa T2
6	Jack
7	Nina
8	Linda T3

accenture

High performance. Delivered.

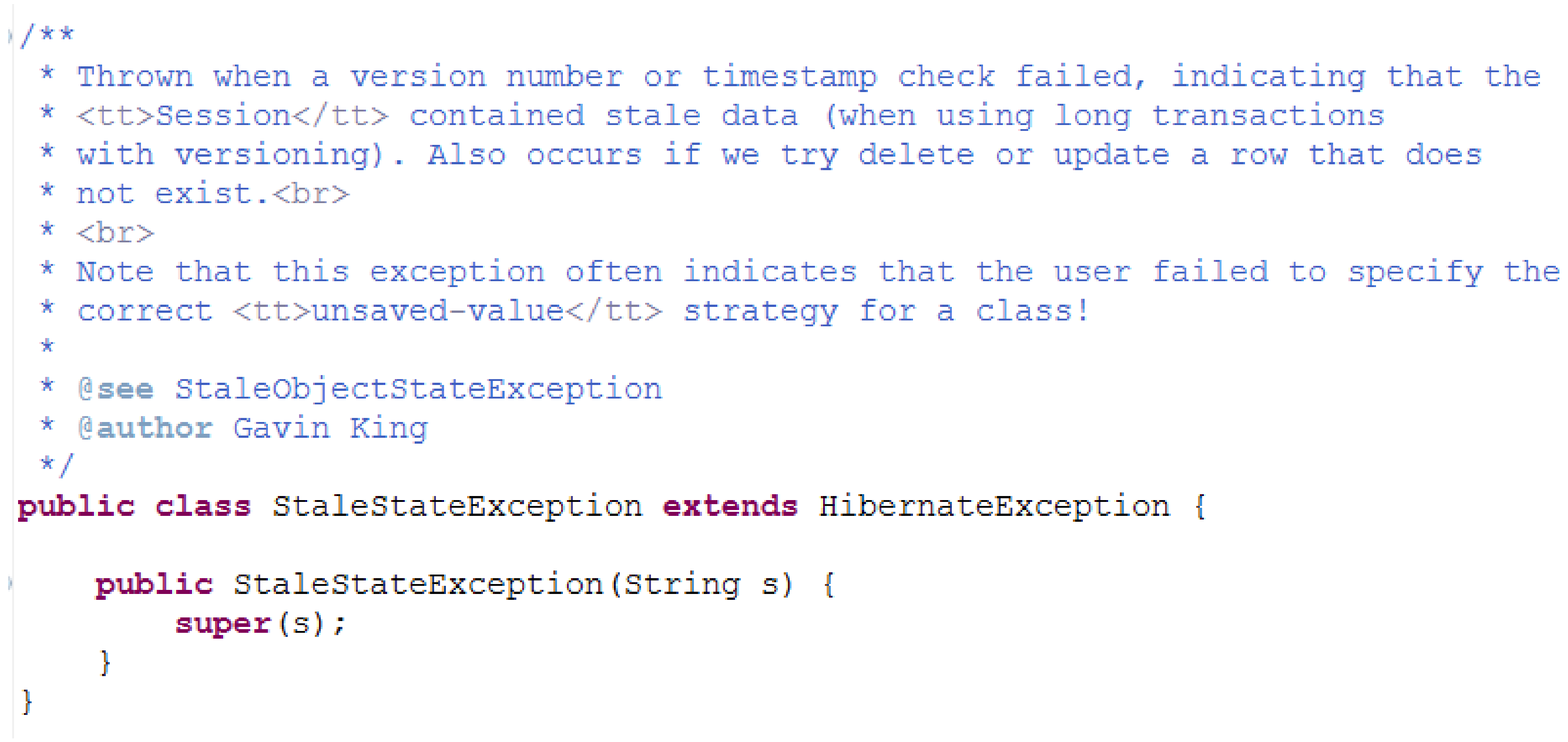
DeadlockLoserDataAccessException

High performance. Delivered.

Retry to the rescue

```
<step id="step2">
 <tasklet>
   <chunk reader="reader" processor="processor" writer="writer"
      commit-interval="10"  retry-limit="10">
         <retryable-exception-classes>
           <include class="…DeadlockLoserDataAccessException"/>
         </retryable-exception-classes>
   </chunk>
  </tasklet>
</step>
```

The mystery exception

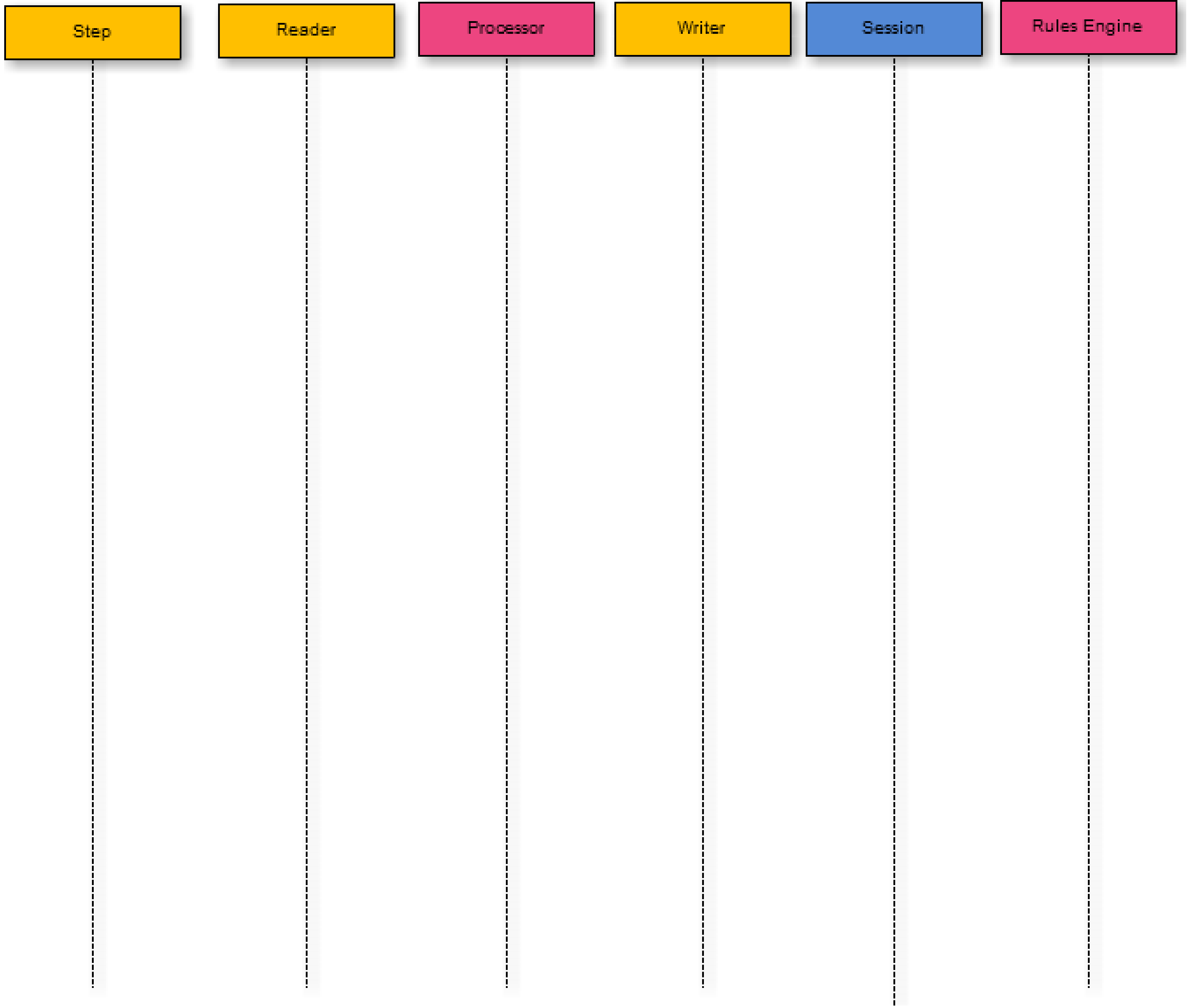


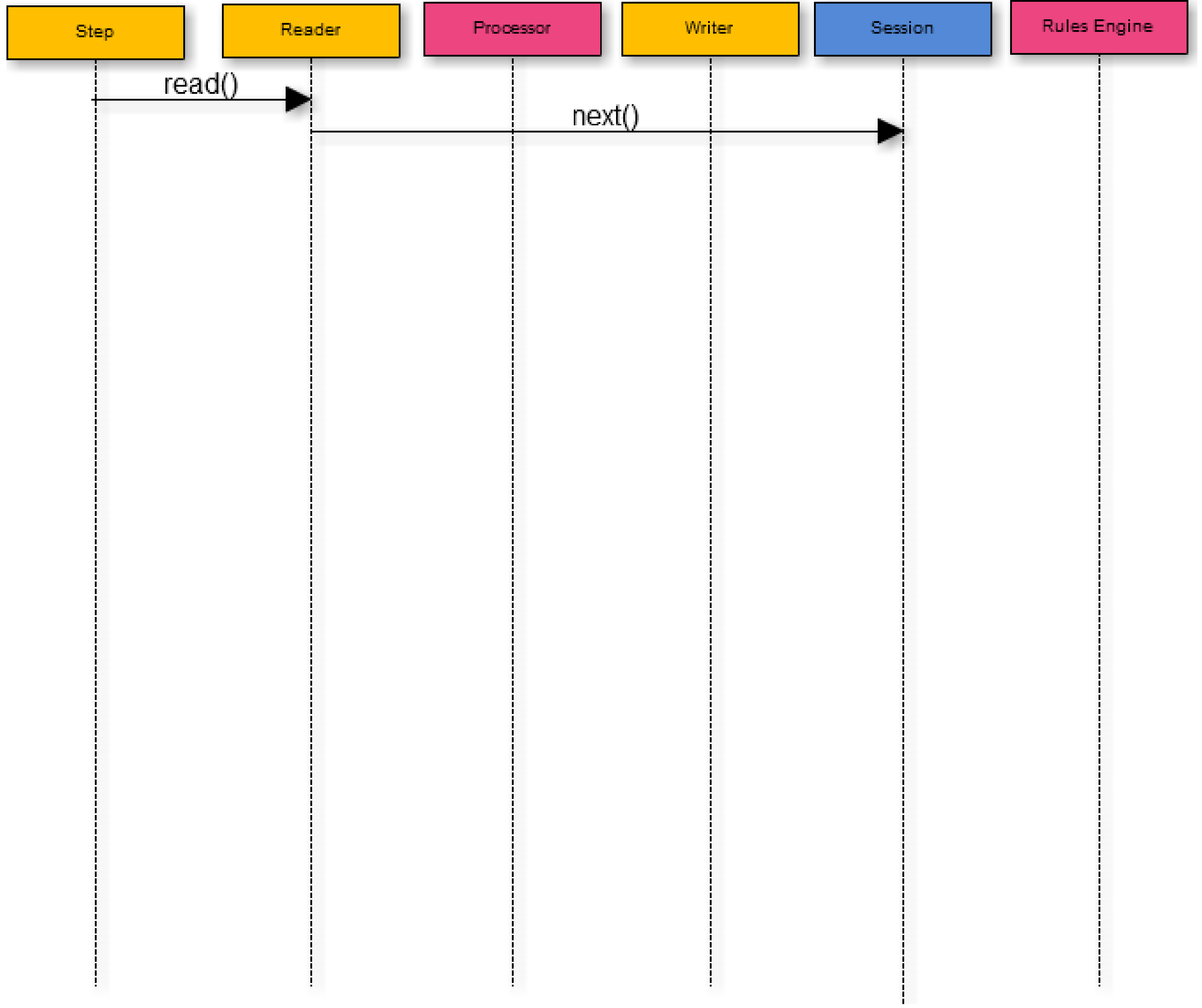
```
/**
 * Thrown when a version number or timestamp check failed, indicating that the
 * <tt>Session</tt> contained stale data (when using long transactions
 * with versioning). Also occurs if we try delete or update a row that does
 * not exist.<br>
 * <br>
 * Note that this exception often indicates that the user failed to specify the
 * correct <tt>unsaved-value</tt> strategy for a class!
 *
 * @see StaleObjectStateException
 * @author Gavin King
 */
public class StaleStateException extends HibernateException {

    public StaleStateException(String s) {
        super(s);
    }
}
```

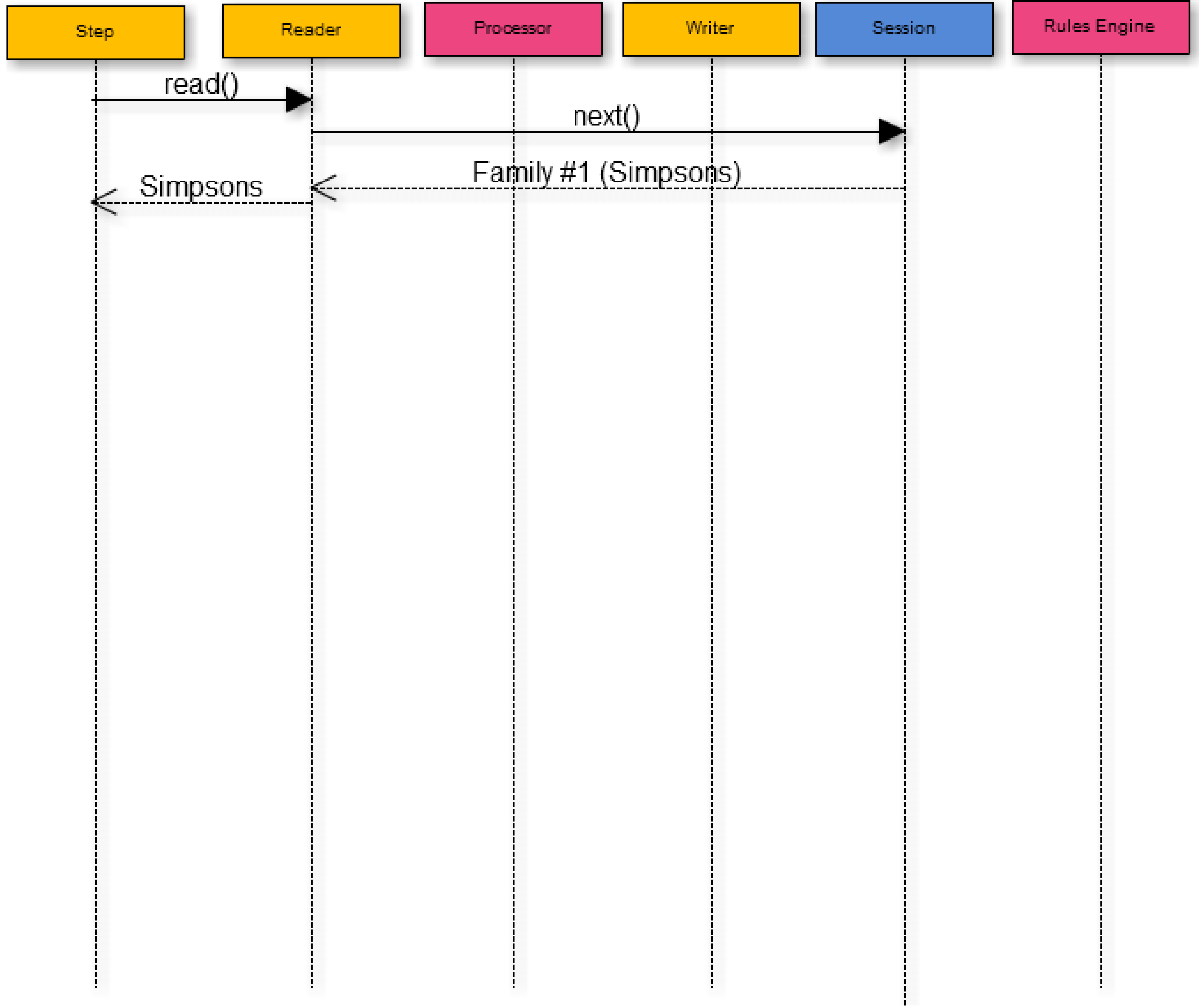
accenture

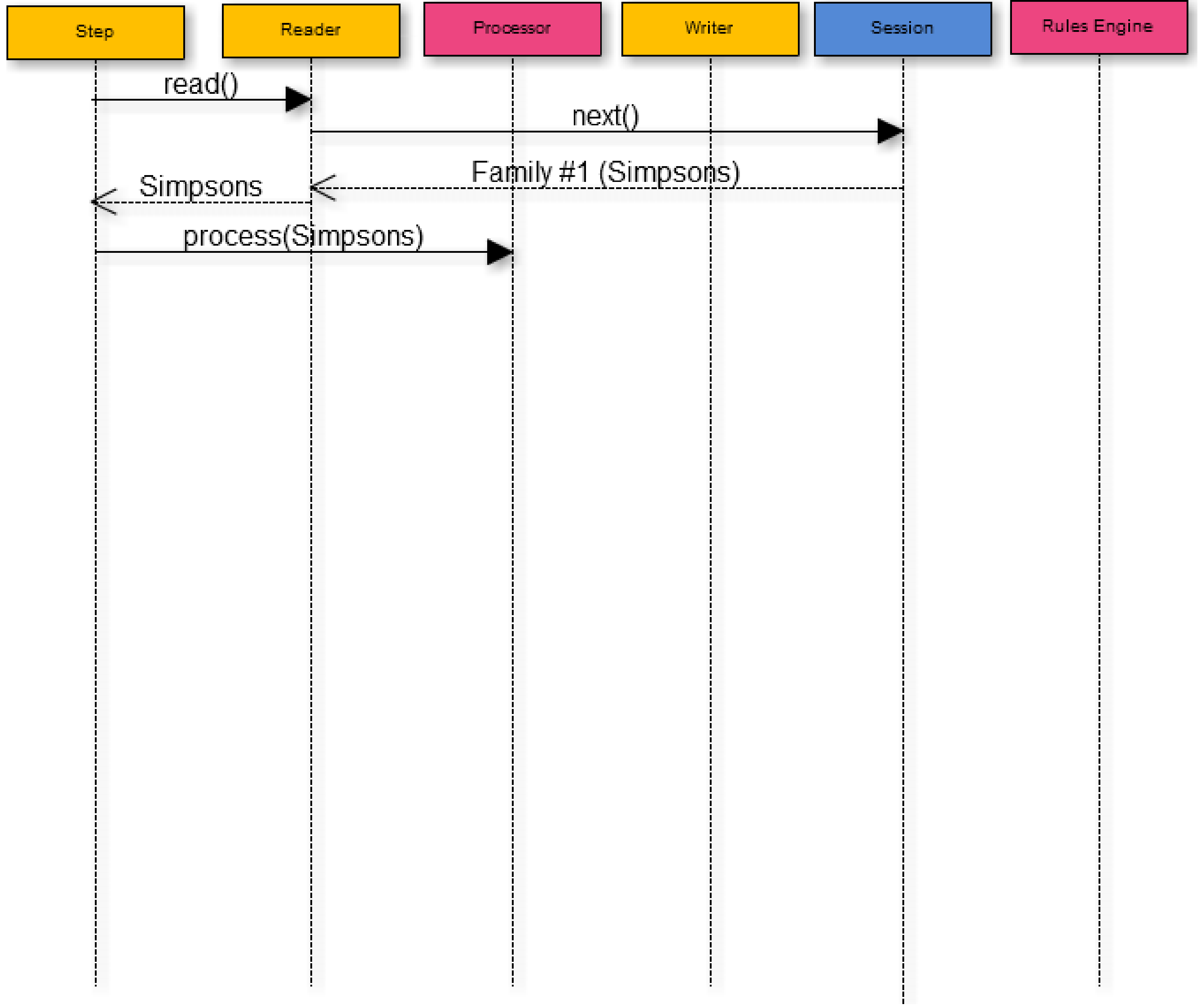
High performance. Delivered.

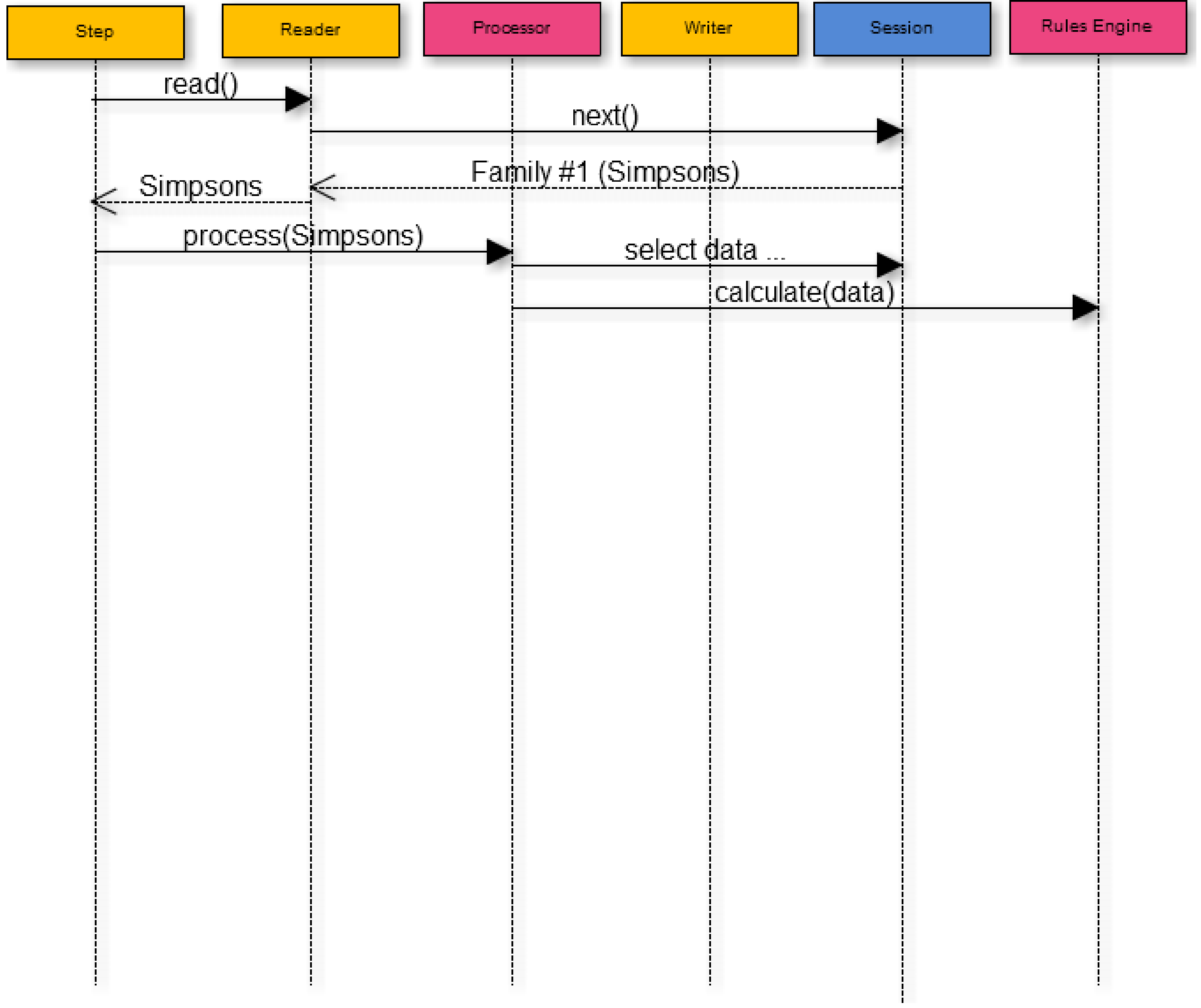
Two weeks later..

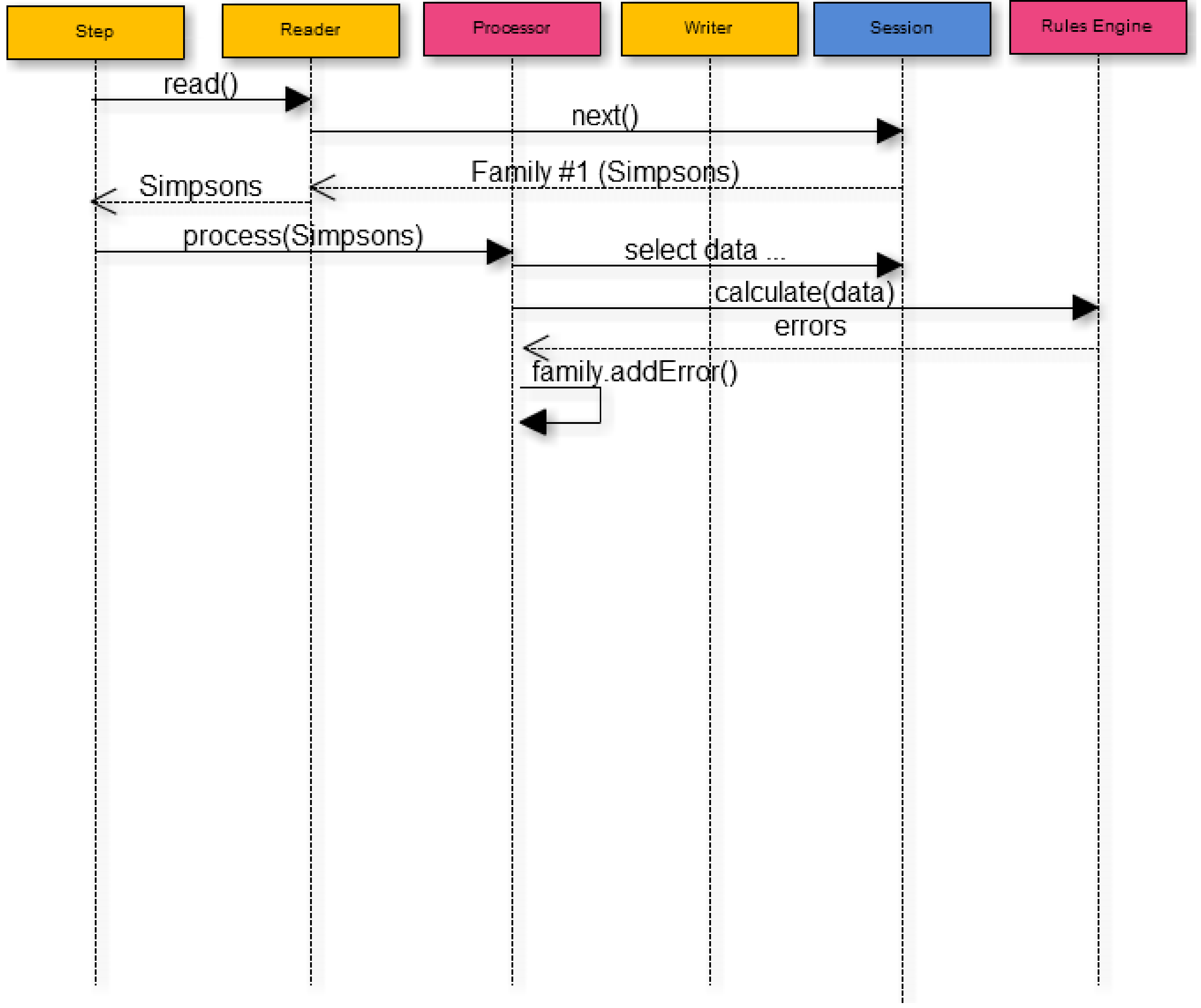

Step
Reader
Processor
Writer
Session
Rules Engine

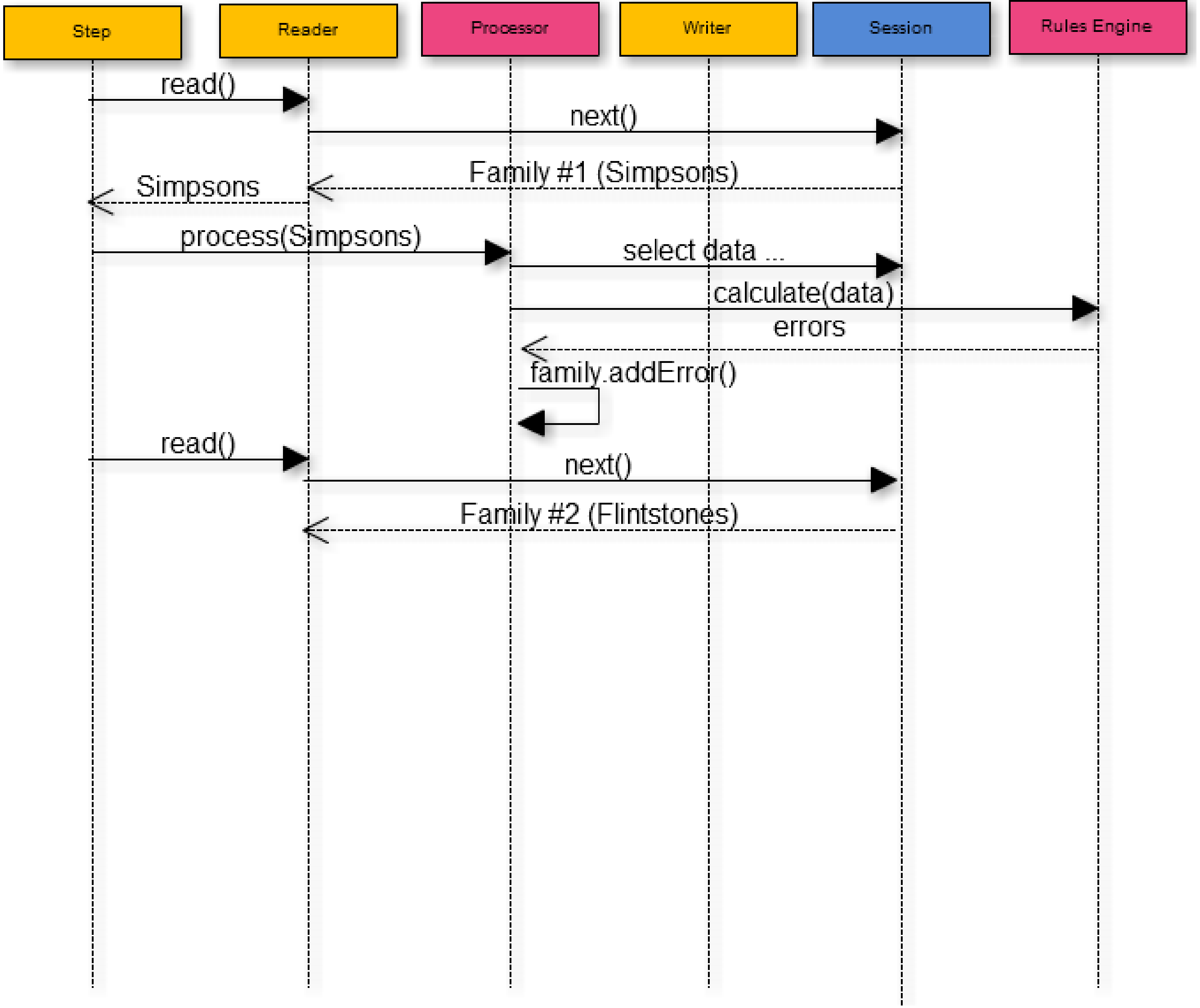

Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()

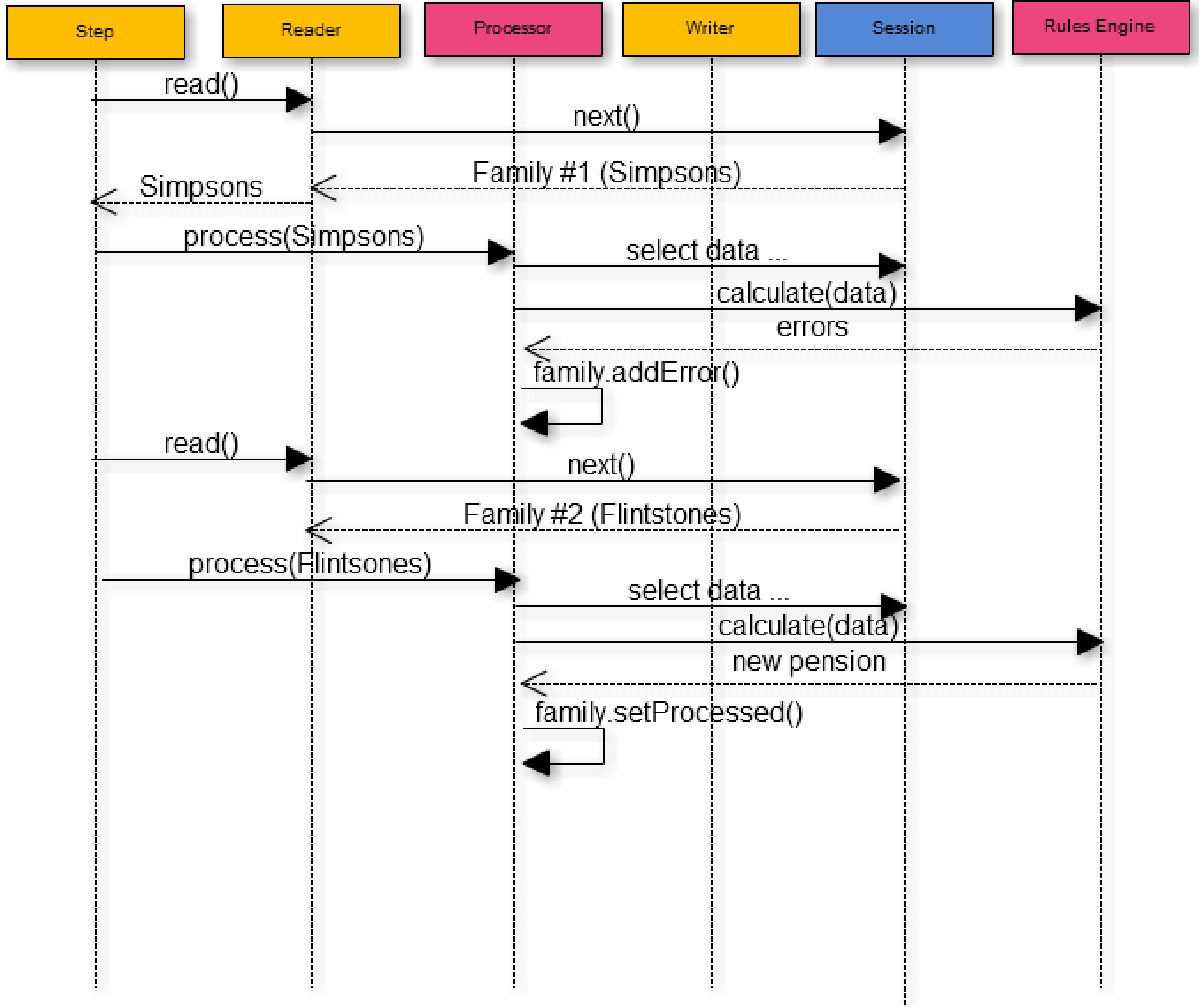

Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons

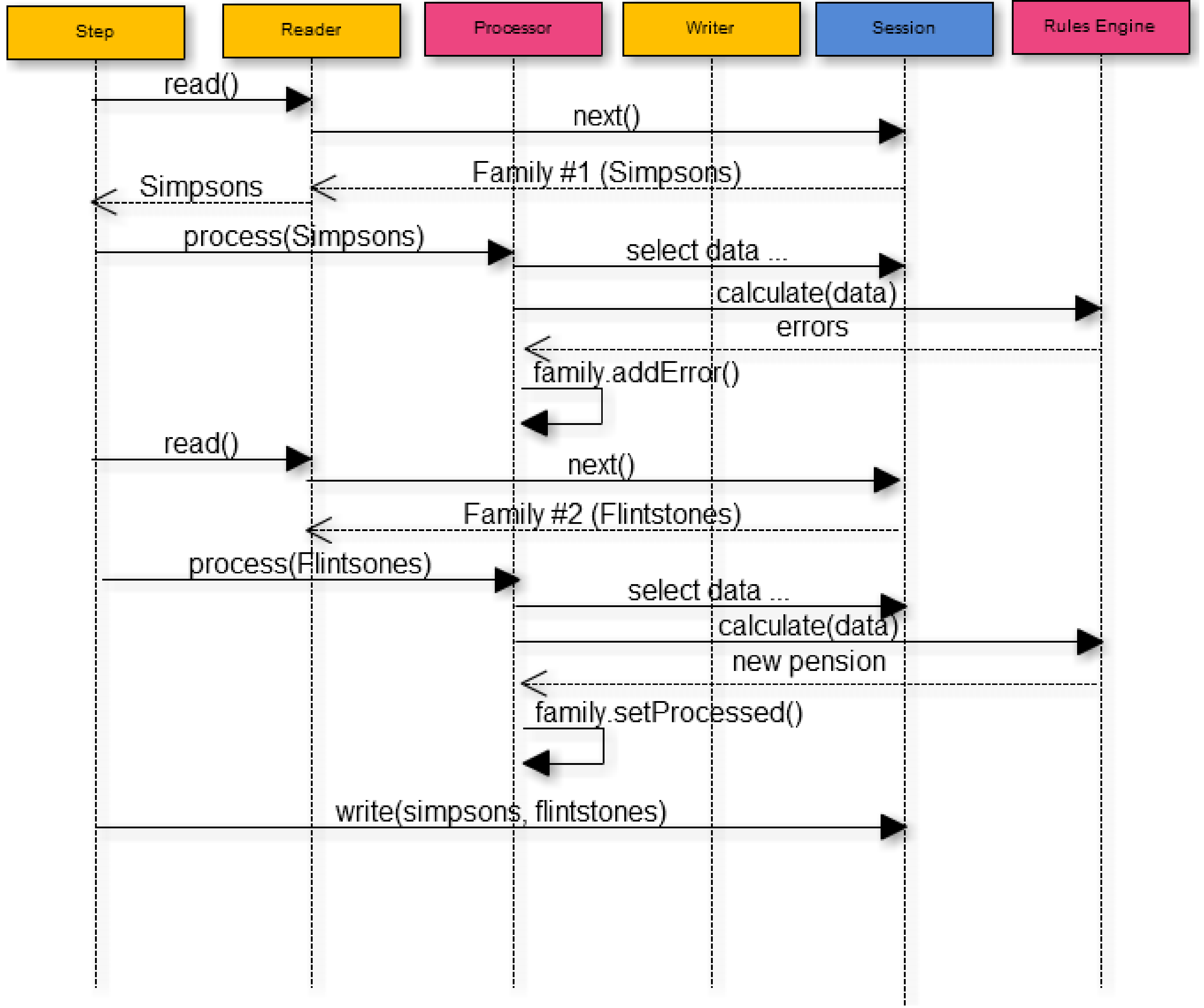

Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)

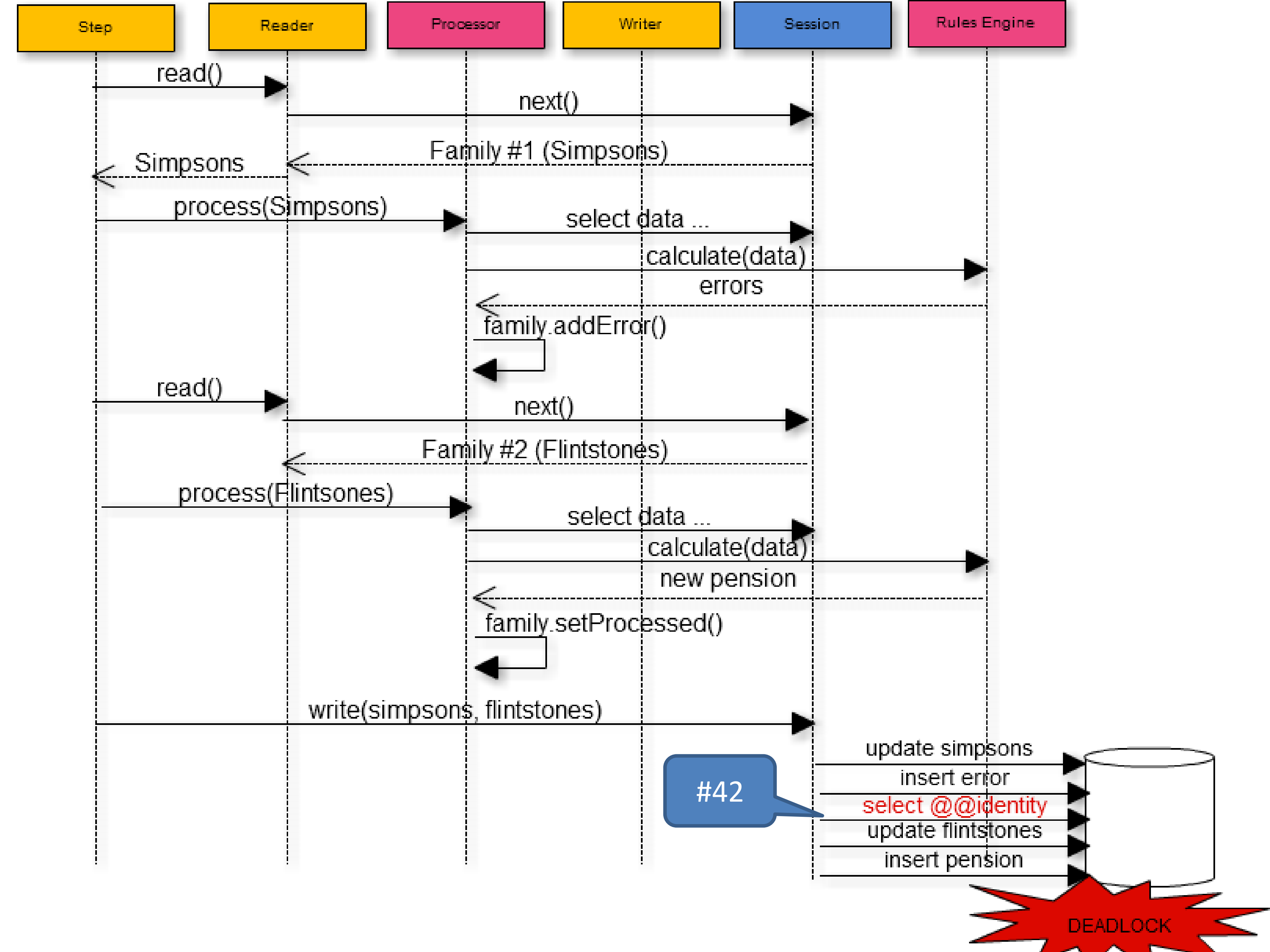

Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)
select data ...
calculate(data)


Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()


Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
read()
next()
Family #2 (Flintstones)


Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
read()
next()
Family #2 (Flintstones)
process(Flintsones)
select data ...
calculate(data)
new pension
family.setProcessed()

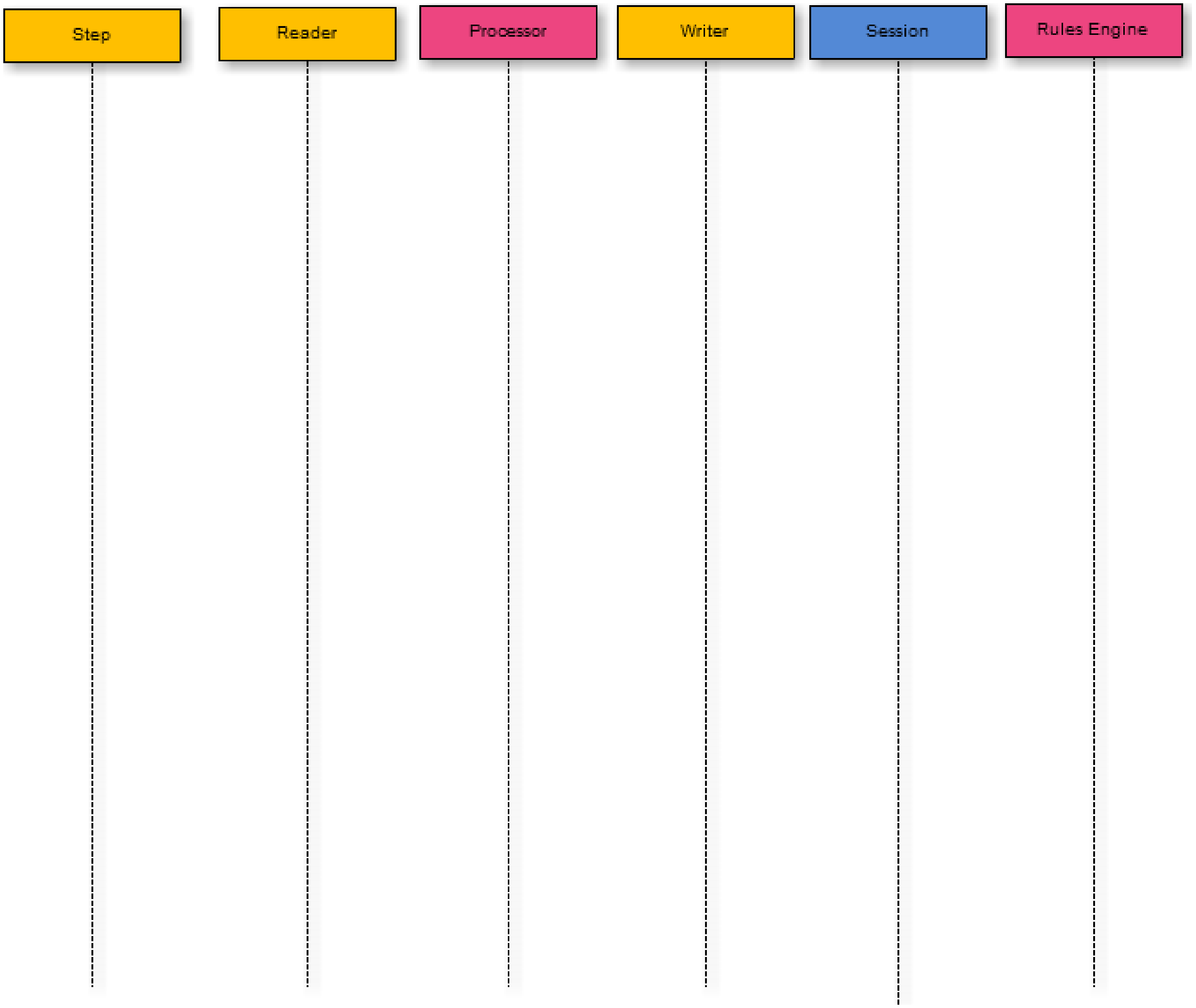

Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
read()
next()
Family #2 (Flintstones)
process(Flintsones)
select data ...
calculate(data)
new pension
family.setProcessed()
write(simpsons, flintstones)

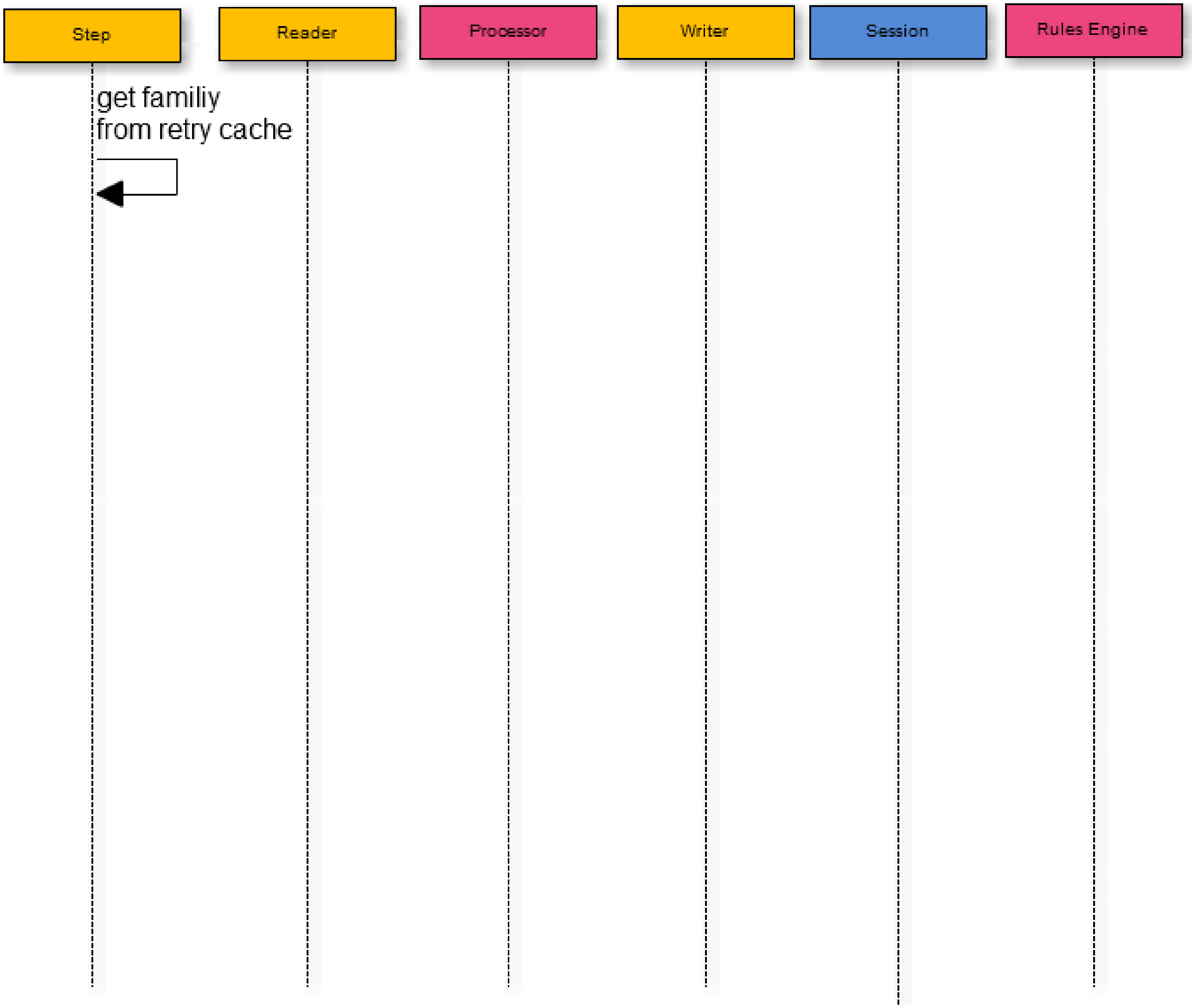

Step
Reader
Processor
Writer
Session
Rules Engine
read()
next()
Family #1 (Simpsons)
Simpsons
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
read()
next()
Family #2 (Flintstones)
process(Flintsones)
select data ...
calculate(data)
new pension
family.setProcessed()
write(simpsons, flintstones)
update simpsons
insert error
select @@identity
update flintstones
insert pension
#42
DEADLOCK

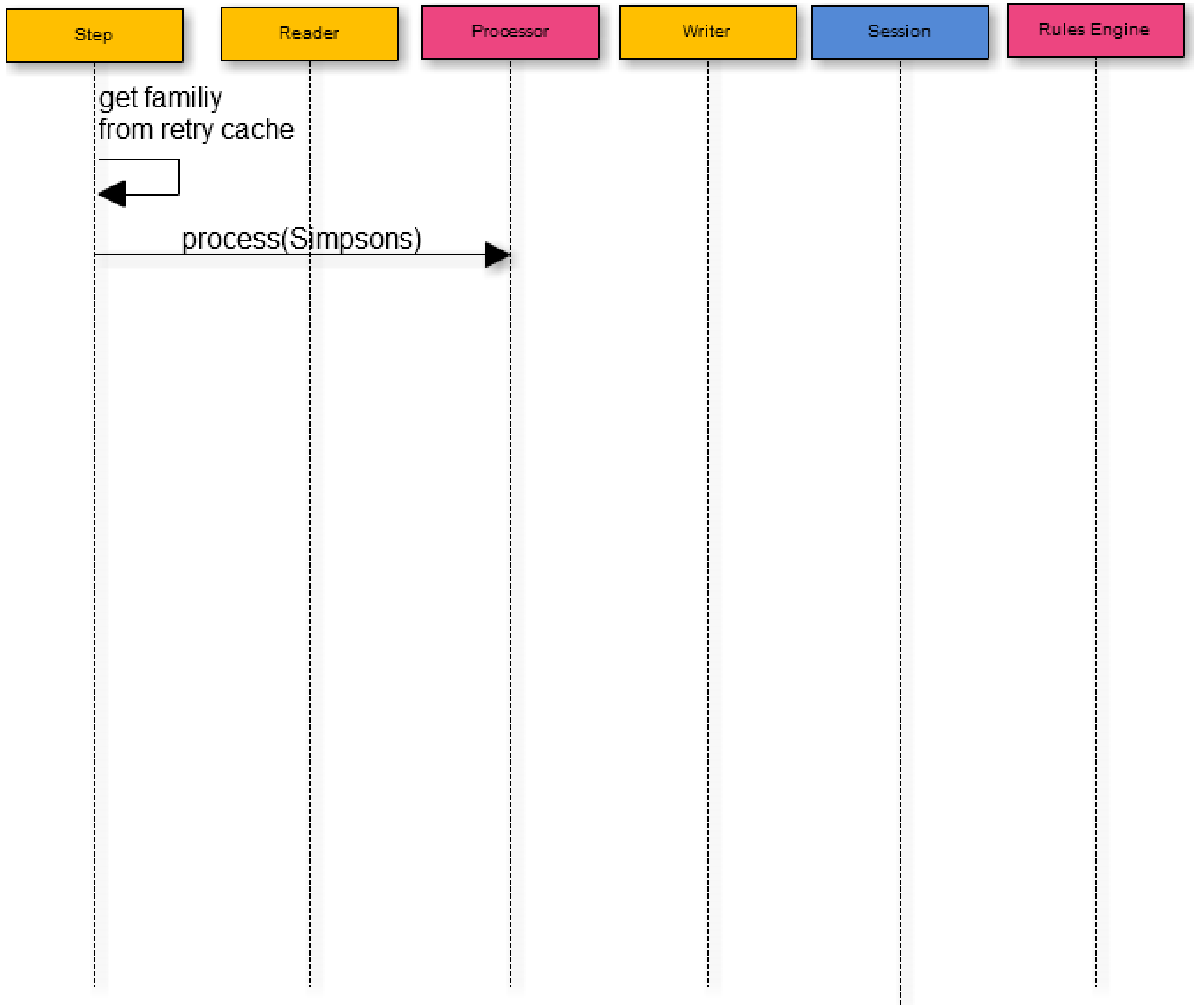
accenture

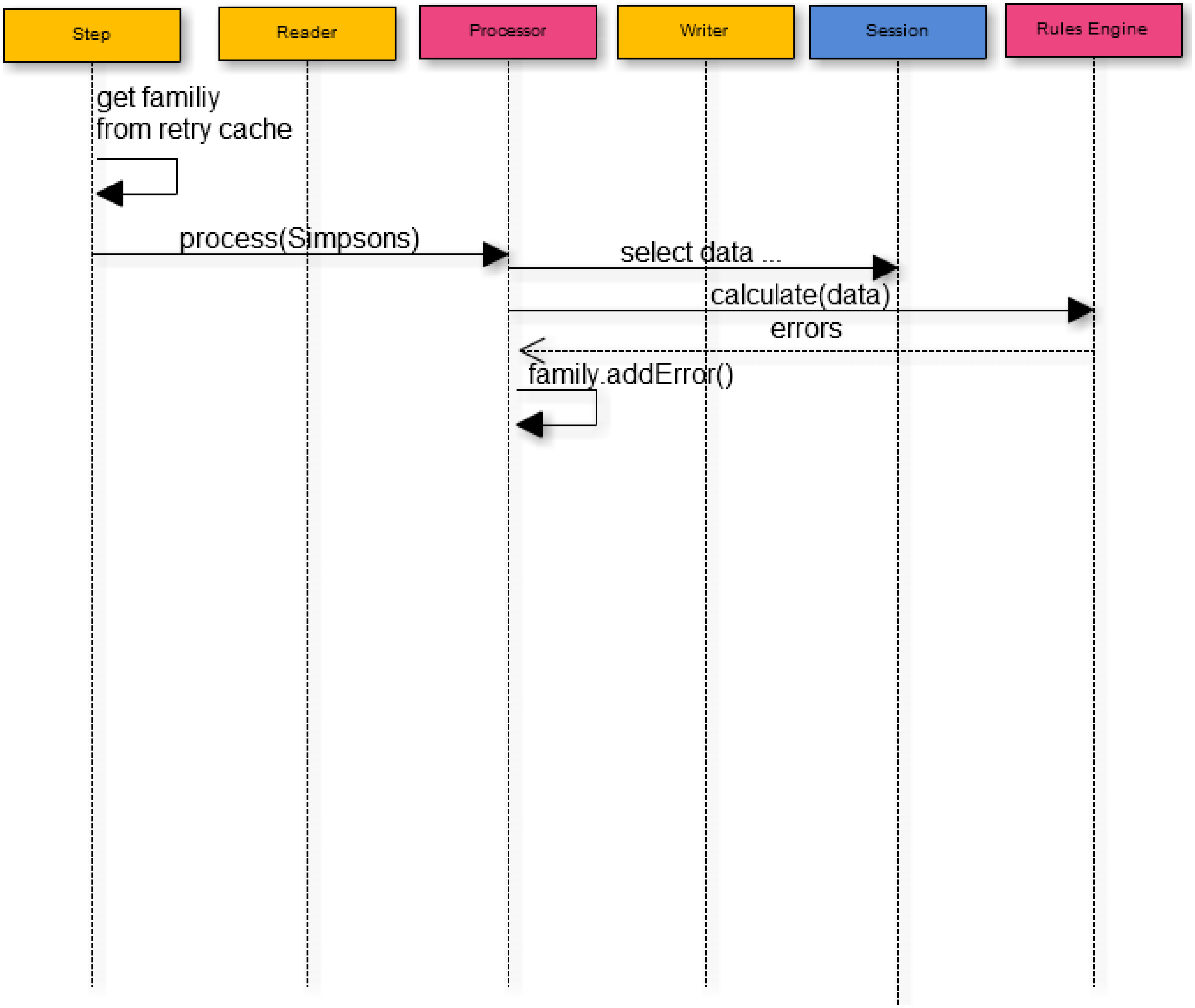
High performance. Delivered.

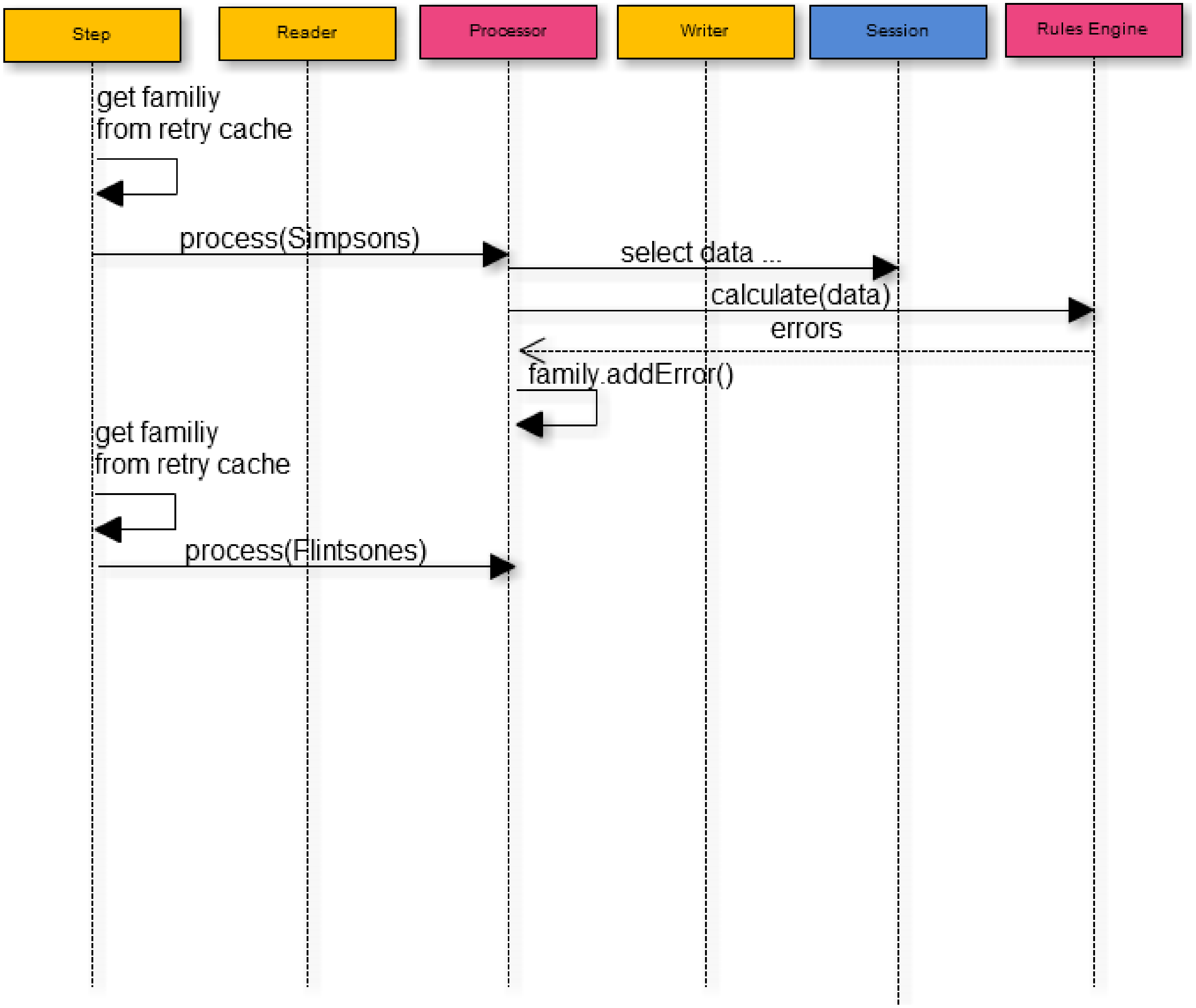
Retry

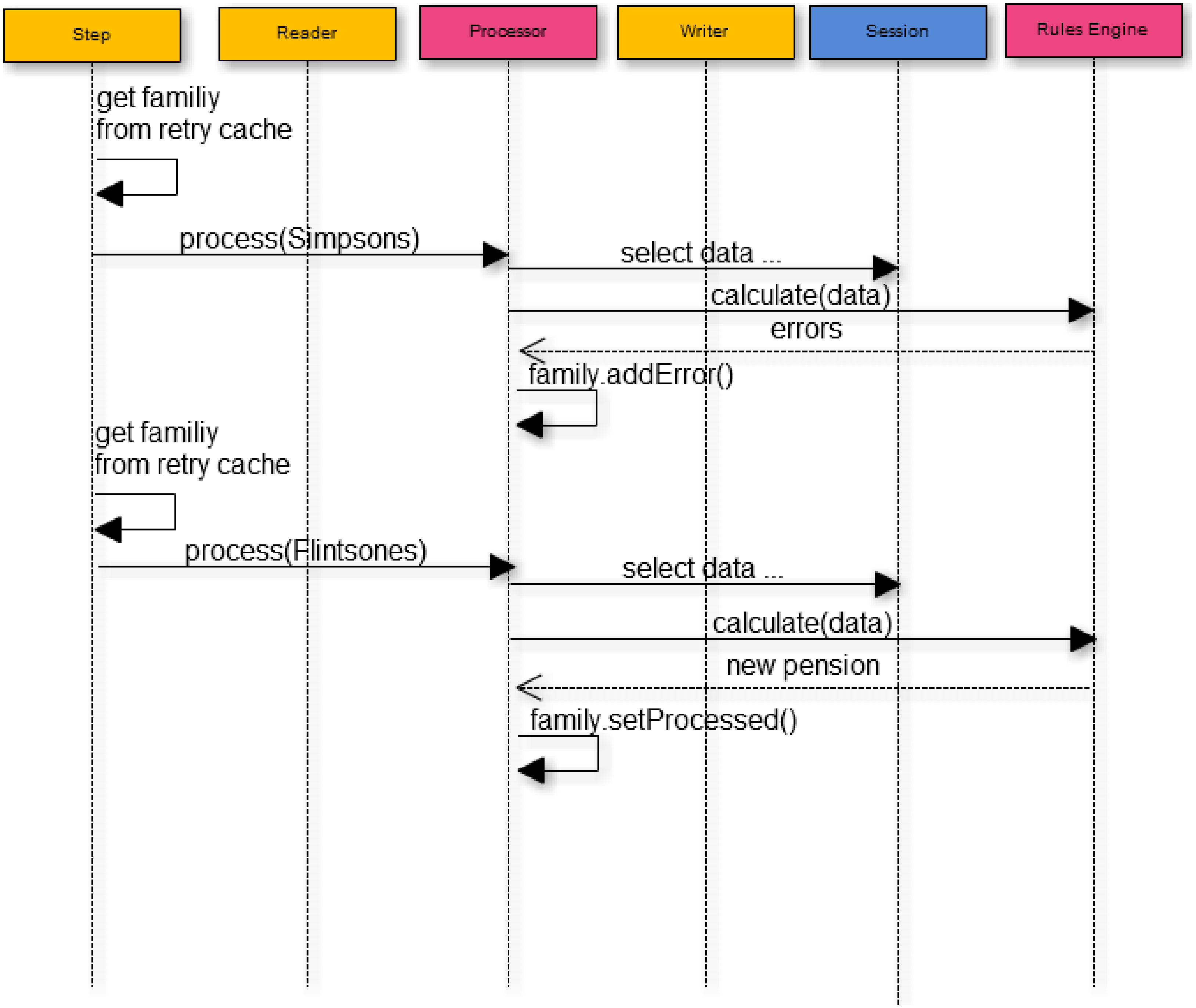

Step
Reader
Processor
Writer
Session
Rules Engine

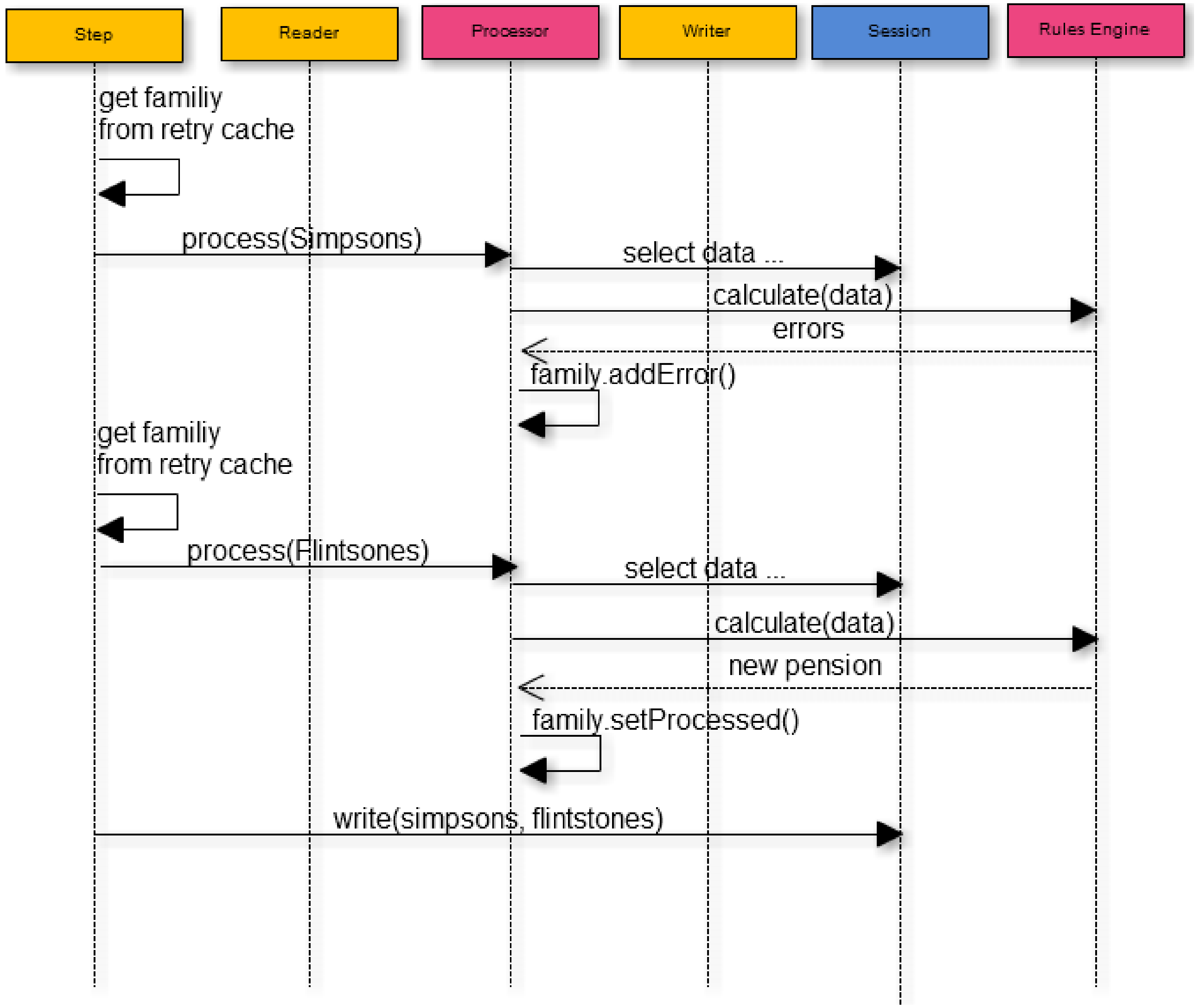

Step
Reader
Processor
Writer
Session
Rules Engine
get familiy
from retry cache

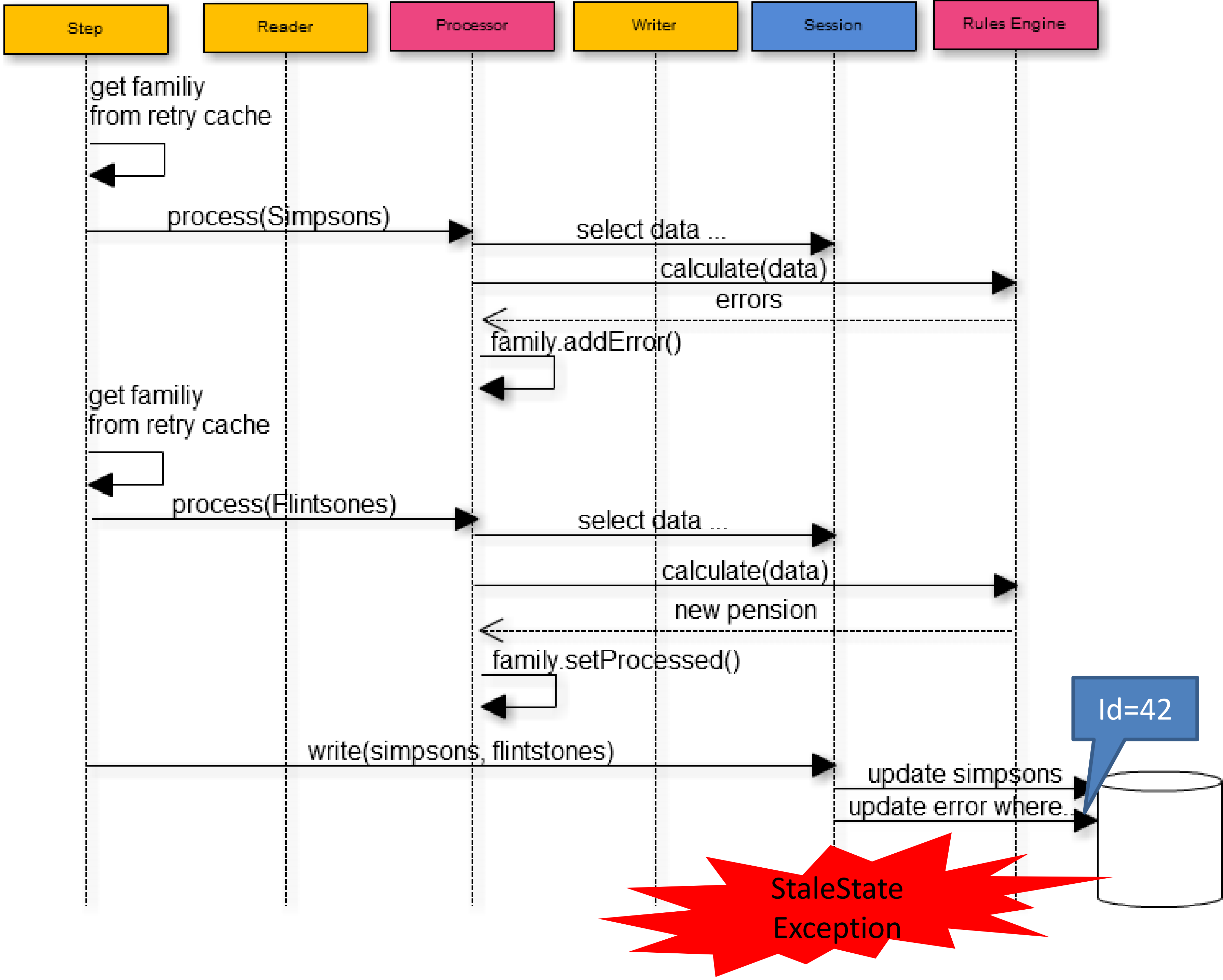

Step
Reader
Processor
Writer
Session
Rules Engine
get familiy
from retry cache
process(Simpsons)


Step
Reader
Processor
Writer
Session
Rules Engine
get familiy
from retry cache
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()


Step
Reader
Processor
Writer
Session
Rules Engine
get familiy from retry cache
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
get familiy from retry cache
process(Flintsones)


Step
Reader
Processor
Writer
Session
Rules Engine
get familiy
from retry cache
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
get familiy
from retry cache
process(Flintsones)
select data ...
calculate(data)
new pension
family.setProcessed()


Step
Reader
Processor
Writer
Session
Rules Engine
get familiy from retry cache
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
get familiy from retry cache
process(Flintsones)
select data ...
calculate(data)
new pension
family.setProcessed()
write(simpsons, flintstones)


Step
Reader
Processor
Writer
Session
Rules Engine
get familiy from retry cache
process(Simpsons)
select data ...
calculate(data)
errors
family.addError()
get familiy from retry cache
process(Flintsones)
select data ...
calculate(data)
new pension
family.setProcessed()
write(simpsons, flintstones)
update simpsons
update error where..
Id=42
StaleState Exception

accenture

High performance. Delivered.

What do we do?

accenture

High performance. Delivered.

What we should have done weeks ago

We ditch Hibernate

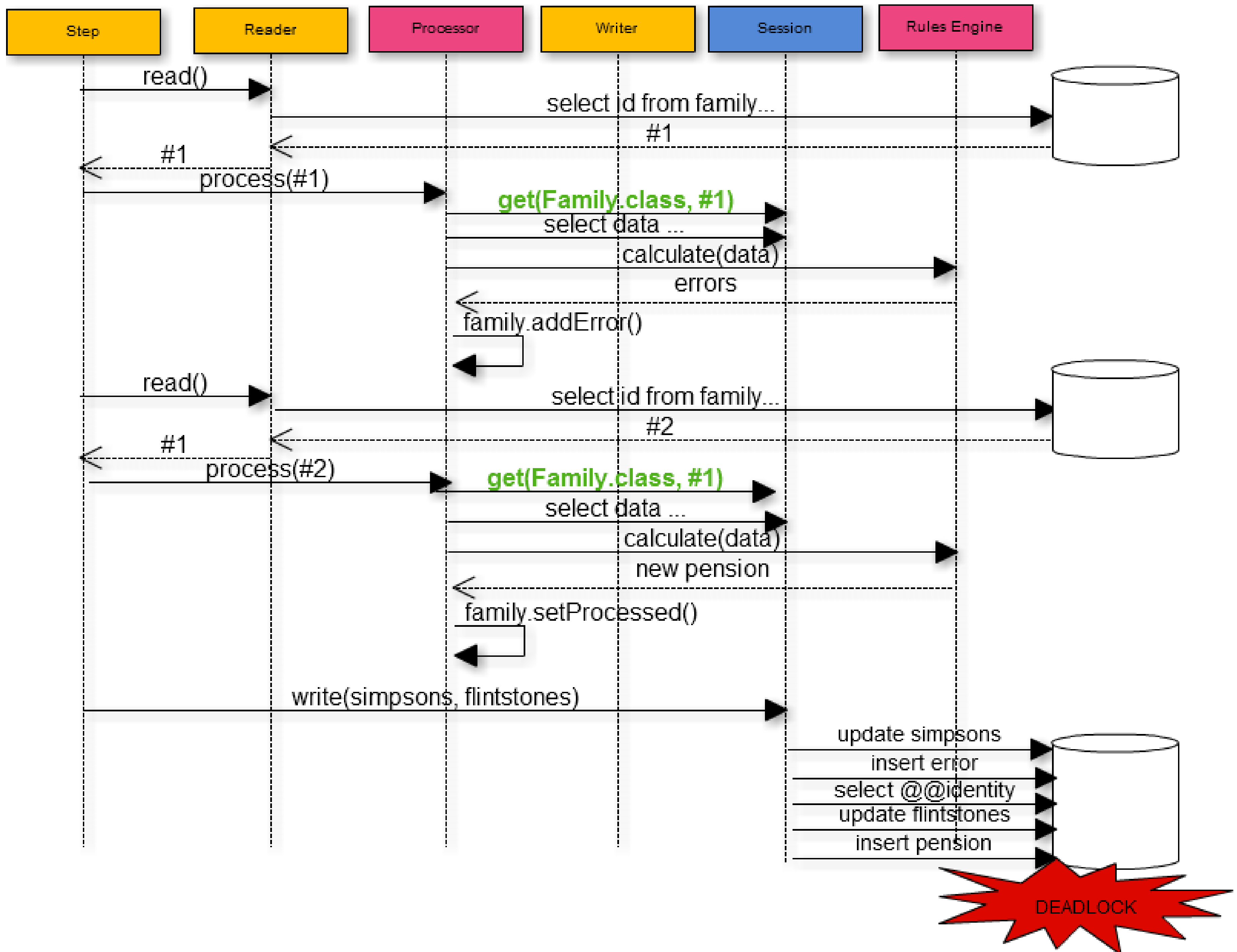
…well, almost anyway

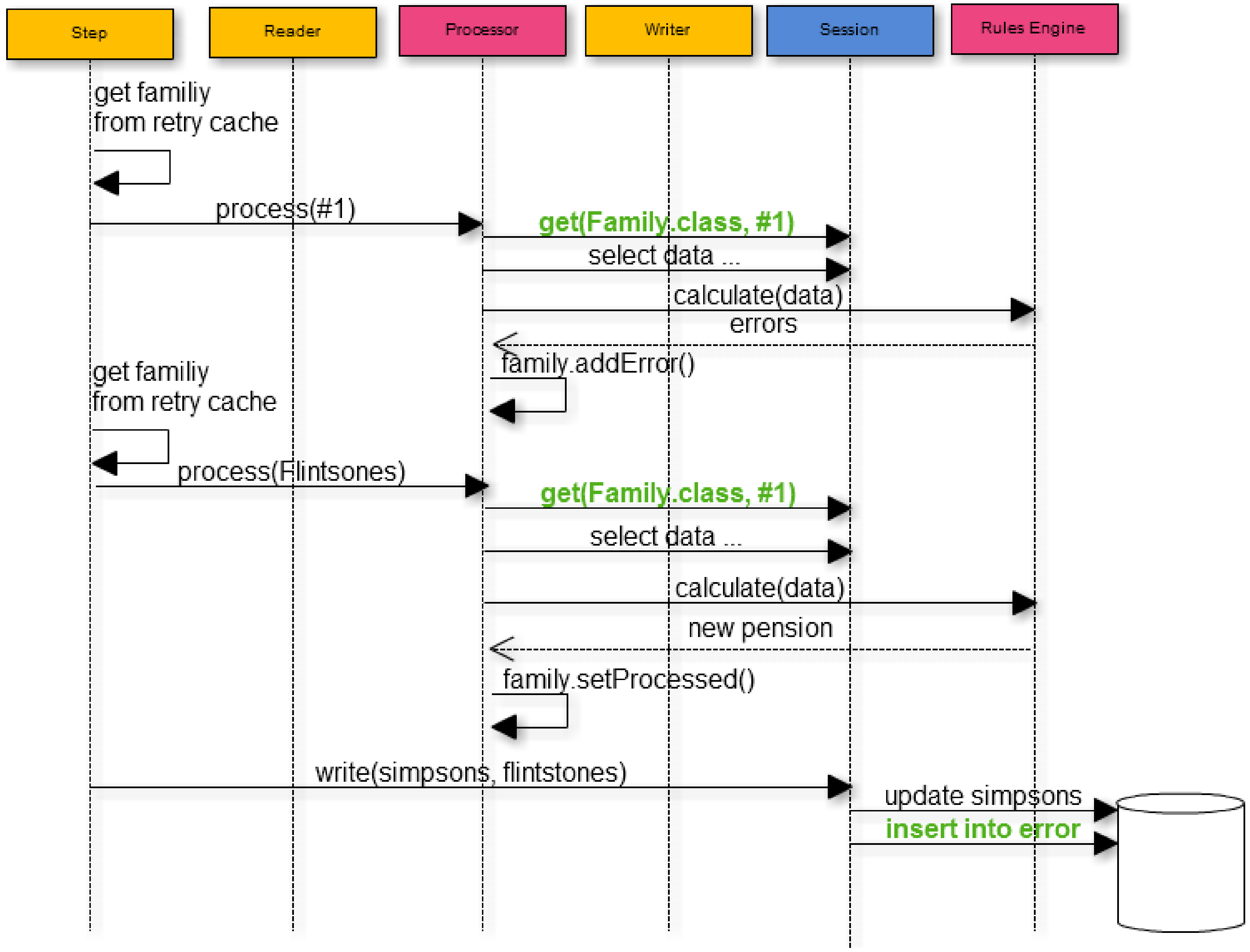
accenture

High performance. Delivered.

Removing Hibernate from reader

- ItemReader is re-configured to use JDBC
- Fetches primary key from family staging table
- ItemProcesser fetches staging object graph
  - Uses primary key to fetch graph with hibernate
- Primary keys are immutable and stateless


Step
Reader
Processor
Writer
Session
Rules Engine
read()
select id from family...
#1
#1
process(#1)
get(Family.class, #1)
select data ...
calculate(data)
errors
family.addError()
read()
select id from family...
#2
#1
process(#2)
get(Family.class, #1)
select data ...
calculate(data)
new pension
family.setProcessed()
write(simpsons, flintstones)
update simpsons
insert error
select @@identity
update flintstones
insert pension
DEADLOCK


Step
Reader
Processor
Writer
Session
Rules Engine
get familiy from retry cache
process(#1)
get(Family.class, #1)
select data ...
calculate(data)
errors
family.addError()
get familiy from retry cache
process(Flintsones)
get(Family.class, #1)
select data ...
calculate(data)
new pension
family.setProcessed()
write(simpsons, flintstones)
update simpsons
insert into error

End result

- Performance requirement was 48 hrs
- Completed in 16 hrs
- Used 12 threads
- C-version used 1 thread and ran for 1 week
  – Stopped each morning, started each evening
- A batch that scales with the infrastructure
  – Number of threads is configurable in .properties

What we would have done differently

- Switched from partioned to multi-threaded step
  - All work is shared among threads
  - All threads will run until batch completes
  - Avoid idle threads towards the end
  - With partitioning some partitions finished well before others

Recommendations

- Do not use Hibernate in the ItemReader
- Test parallelization early
- Monitor your SQLs
  - Frequently called
  - Long running
- Become friends with your DBA
- There is no reason to let Java be the bottle neck
  - Increase thread count until DB becomes the bottle neck

accenture

High performance. Delivered.

Thanks!

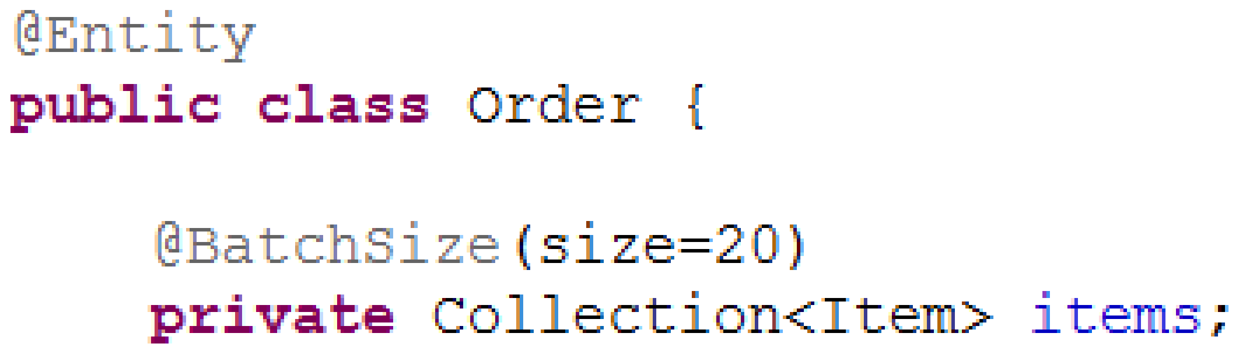
@mortenag

www.github.com/magott

www.andersen-gott.com

Pro tip: @BatchSize

- If one lazily loaded entity is fetched, they are all fetched – in one query



```
@Entity
public class Order {

    @BatchSize(size=20)
    private Collection<Item> items;
```