Concurrency and Thread-Safe Data Processing in Background Tasks

Kieran Kelleher
Green Island Consulting LLC
SmartleadsUSA LLC
SmartMix Technologies LLC

Why Background Threads?

- “Background” = Not in the Request-Response thread (WOWorkerThread)
  - We can execute long-running logic asynchronously in a background thread.
- An action MAY be always or sometimes too long
  - Examples
    - importing a mailing list of 50 records versus a list of 100,000 records.
    - Generating a report off a selection of 50 EOs versus 100,000 EOs.
    - your app interacting with remote web services (connection timeouts, etc.)
    - interacting with a merchant payment gateway (Credit Card processing)
    - Administration & Bulk Data Processing Operations

Objectives

- How to run simple background tasks with the least effort on your behalf.
- Start an asynchronous task
- Use a long response page to start a task, monitor and handle a task result
    - with basic user feedback (busy)
    - with better user feedback (progress, status)
- How to work with EOF in the context of background threads.
- How to pass arguments and return results (including EOs) from asynchronous tasks.
- How to run a multi-threaded bulk data processing asynchronous task to get more done faster.
- Introduce convenient classes in Wonder that are related to running tasks in background threads.

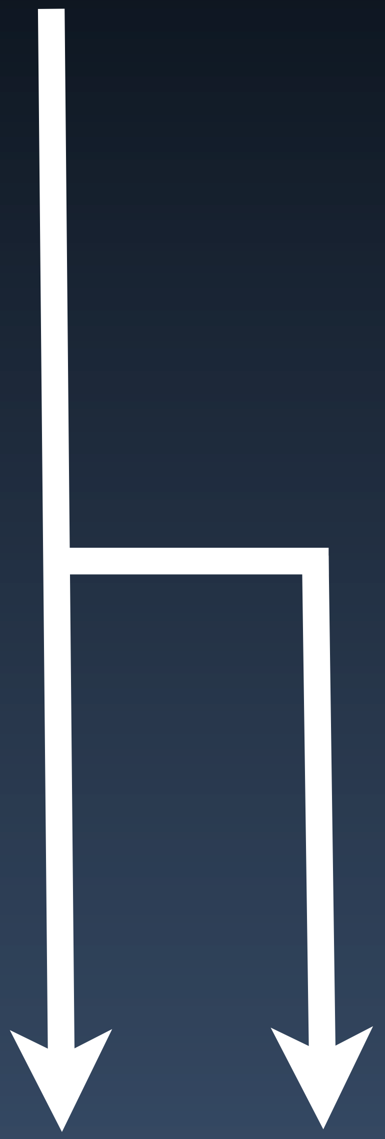
Anatomy of a “Task”

```
public class SimpleRunnable implements Runnable {

     public void run() {

          //Do a whole lot of stuff here

     }
}
```

```
public class SimpleCallable implements Callable<ResultType> {

     public ResultType call() {

          //Do a whole lot of stuff here

           return _someResult;
     }
}
```

Executing a Task

- Use java.util.concurrent.*
  - Convenient, easy to use.
- In summary: Basic concurrency classes to understand
  - Runnable, Callable, ExecutorService, Future
- java.util.concurrent.ExecutorService interface
  - executorService.execute(task);
  - Future future = executorService.submit(task);
- java.util.concurrent.Future interface
  - future.get() or future.get(timeout, timeUnit)
  - future.isDone()



Executing a Task

```
// Creating an ExecutorService in plain Java
ExecutorService executorService = Executors.newCachedThreadPool();
```

```
// Getting a reference to the WebObjects-friendly singleton ExecutorService
ExecutorService executorService = ERXExecutorService.executorService()
```

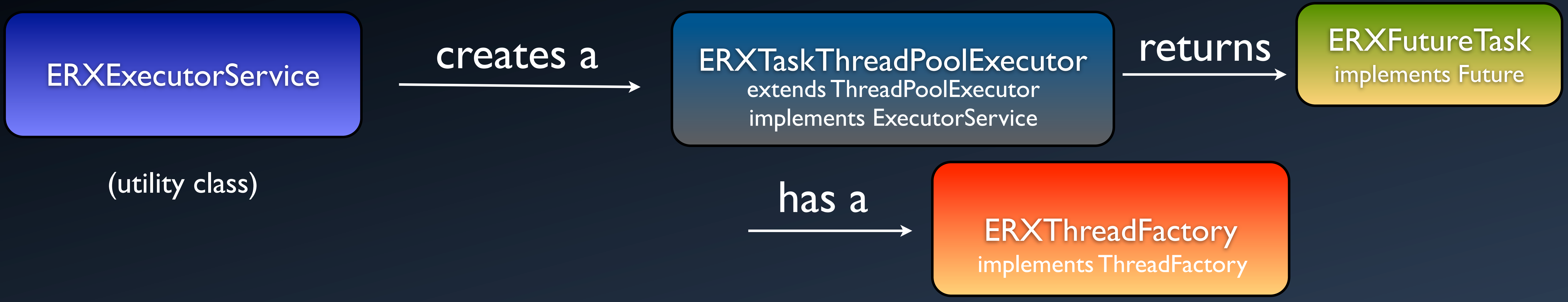
```
// Starting an asynchronous task (aka run it in another thread)
Runnable task = new MyRunnableTask();
executorService.execute( task );
```

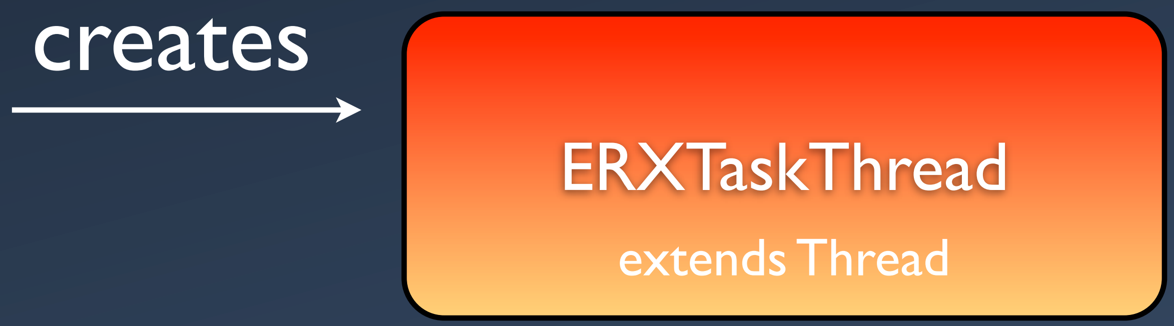
```
// Starting an asynchronous task and keeping an eye on it
Future future = executorService.submit( task );
```

```
// Starting a task in a WebObjects action
public WOActionResults dispatchBackgroundTask() {
    MyTask task = new MyTask();
    ERXExecutorService.executorService().execute(task);
    return null;
}
```

Demo 1

ERXExecutorService





- Usage
  - ExecutorService es = ERXExecutorService.executorService();
  - es.execute(runnable);
  - Future future = es.submit(task);
  - You generally don't need to directly use the stuff to the right :-)

Benefits of using ExecutorService instances returned by ERXExecutorService

- Allows for loosely coupled plain Runnable and Callable tasks.
    - No subclassing necessary to get WO integration!
- EC Safety Net: Automatically unlocks all EC's at the end of task execution (no subclassing needed)
    - NOT a reason to ignore locking!
- TODO: ERXThreadStorage "safe" cloning.
- By the way...
    - ERXTaskThread subclass of Thread used
    - supports use of ERXExecutionStateTransition interface in your task.

The "Ideal" Long Response Page?

- Provides feedback to the user
- Simple UI and easy to understand user experience
- Can be reusable and easy (even pleasant) to implement
  - ~~! WOLongResponsePage~~
- Controls the user by making them wait
  - May not be a good idea for very long tasks

Demo 2

CCAjaxLongResponsePage

- Resides in ERCoolComponents framework
- Easy to use ... really!
- CSS styleable
- Customizable via Properties

CCAjaxLongResponsePage

```
// Basic usage: run a task in long response page and return to the same page

     public WOActionResults dispatchBackgroundTaskInLongResponsePage() {
          Runnable task = new MyRunnableTask();

          CCAjaxLongResponsePage nextPage = pageWithName(CCAjaxLongResponsePage.class);
          nextPage.setTask(task);

          return nextPage;
     }
```

```
#####################################################################
# Optional configuration properties
#####################################################################
```

```
# A default status message to display if the long running task does not implement ERXStatusInterface
er.coolcomponents.CCAjaxLongResponsePage.defaultStatus=Please wait...
```

```
# Stylesheet for CCAjaxLongResponsePage
er.coolcomponents.CCAjaxLongResponsePage.stylesheet.framework = ERCoolComponents
er.coolcomponents.CCAjaxLongResponsePage.stylesheet.filename = CCAjaxLongResponsePage.css
```

```
# Useful for developing a custom CSS style-sheet. When set to true, this flag prevents AJAX refresh on all containers
# on the CCAjaxLongResponsePage and keeps the page open indefinitely even after the task has completed.
er.coolcomponents.CCAjaxLongResponsePage.stayOnLongResponsePageIndefinitely = false
```

```
# Default refresh interval for CCAjaxLongResponsePage
#er.coolcomponents.CCAjaxLongResponsePage.refreshInterval = 2
```

```
#Defines a default controller class, other than the hard-coded default, for handling task errors for the application
er.coolcomponents.CCAjaxLongResponsePage.nextPageForErrorResultControllerClassName=com.myproject.MyErrorController
```

Monitoring & Controlling Tasks

- ERXStatusInterface
  - public String status();
- ERXTaskPercentComplete
  - public Double percentComplete();
- IERXStoppable
  - public void stop();
- Only ONE method in each interface!! :-)

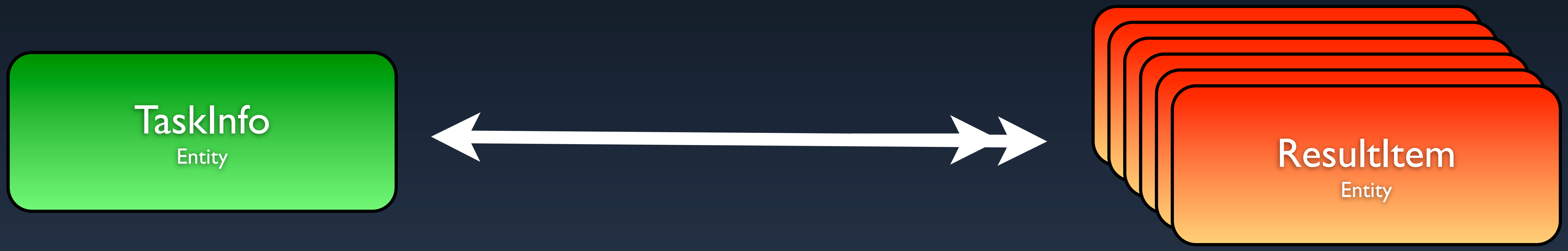
Demo 3

EOF Background Tasks

- Good Practices
  - Only pass EOGlobalIDs (or raw rows) <u>between</u> threads
  - Manual lock/unlock. --- EC lock() / try / finally / unlock()
  - Avoid using default EOObjectStoreCoordinator
- Things to remember
  - Task constructor code does not run in the task thread.
    - Pass in an EO in constructor - convert to EOGlobalID

FYI, About the Demo EOModel

(Number Crunching for the sake of it)

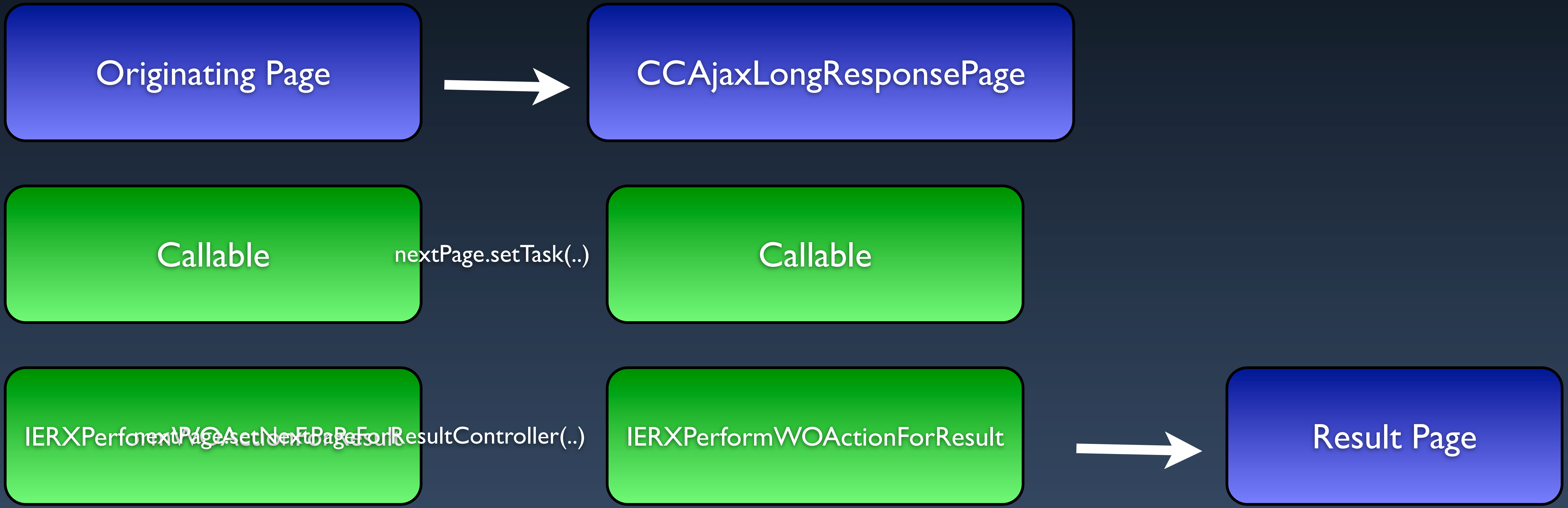


Represents a single execution of a task

Represents result of one loop iteration.
1) Checking if a number if Prime
2) Checking if a Prime is a Factorial Prime

Demo 4

How It Works


Originating Page
CCAjaxLongResponsePage
Callable
nextPage.setTask(..)
Callable
IERXPerformWOActionForResult
nextPage.setNextPageForResultController(..)
IERXPerformWOActionForResult
Result Page

IERXPerformWOActionForResult

- Interface
  - public WOActionResults performAction();
  - public void setResult(Object result);
- Utility Implementation
  - ERXNextPageForResultWOAction
  - new ERXNextPageForResultWOAction(resultPage, "resultKey");

Customize end of task WOActionResults behavior

```
// Example of action in originating page to return a long response page

public WOActionResults performTaskWithCustomResultController() {

    // Create the controller for handling the result and returning the next page after the task is done
    IERXPerformWOActionForResult controller = new ERXNextPageForResultWOAction(pageWithName(MyResultPage.class), "resultKeyInPage");

    // Create the task
    Callable<EOGlobalID> task = new MyCallableTask();

    // Create the CCAjaxLongResponsePage instance
    CCAjaxLongResponsePage nextPage = pageWithName(CCAjaxLongResponsePage.class);

    // Push controller and the task into CCAjaxLongResponsePage
    nextPage.setNextPageForResultController(controller);
    nextPage.setTask(task);

    // Return the CCAjaxLongResponsePage instance
    return nextPage;
}
```

Avoiding the default EOObjectStoreCoordinator in your task

```
// Using an OSC from a pool of OSC dedicated to background tasks. Pool size configurable.
EOObjectStoreCoordinator osc = ERXTaskObjectStoreCoordinatorPool.objectStoreCoordinator();
```

```
EOEditingContext ec = ERXEC.newEditingContext( osc );
ec.lock();
try {
    // Do stuff

} finally {
    ec.unlock();
}
```

```
# Configure the default OSC pool size for background tasks with property
er.extensions.concurrency.ERXTaskObjectStoreCoordinatorPool.maxCoordinators = 4
```

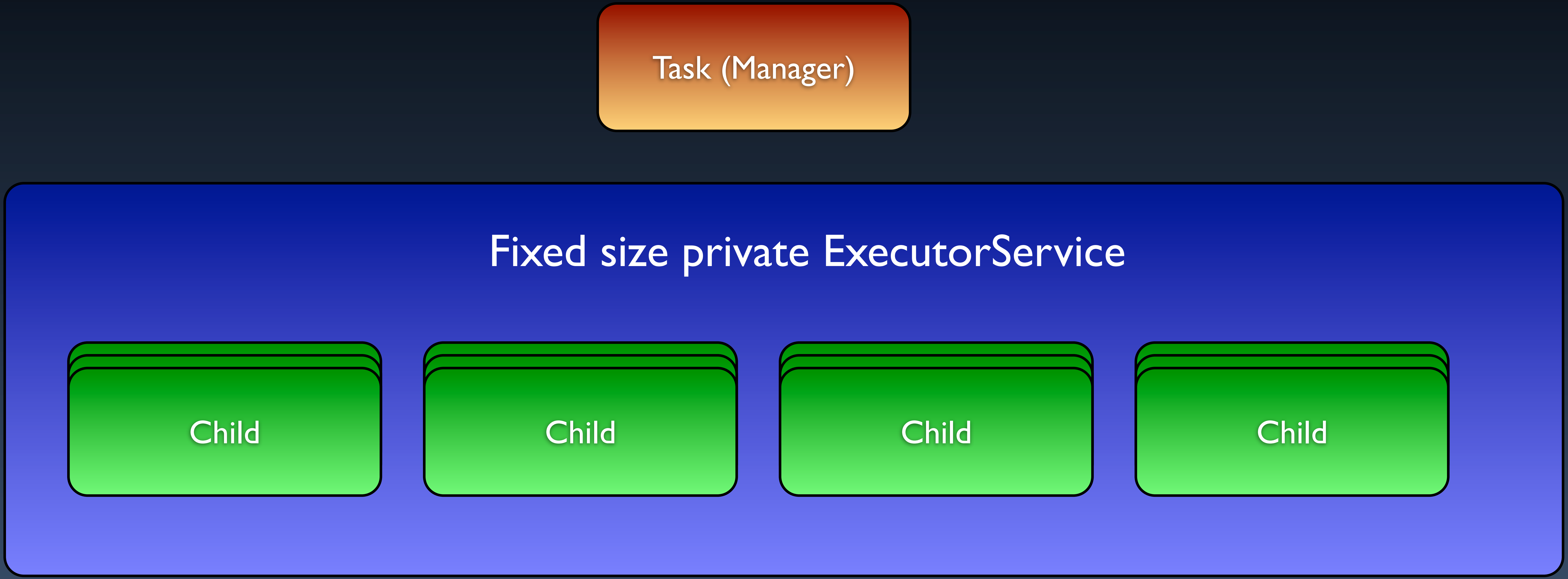
```
// Convenience class ERXAbstractTask
// @see ERXAbstractTask#newEditingContext()

public class MyRunnable extends ERXAbstractTask
```

Just extend ERXAbstractTask and call newEditingContext() !

Handling the return object

```
// If you implement your own IERXPerformWOActionForResult and result is an EOGlobalID


if (_result instanceof EOGlobalID) {

    // Create a new EC
    EOEditingContext ec = ERXEC.newEditingContext();

    // Let's ensure fresh ec since we are likely coming out of a background task
    ec.setFetchTimestamp(System.currentTimeMillis());

    _result = ec.faultForGlobalID((EOGlobalID) _result, ec);

}

_nextPage.takeValueForKey(_result, _nextPageResultKey);
```

Multi-threaded Tasks



Demo 5

Multi-threaded task notes

- Parent task can delegate batches of work to child tasks
  - Concept - many child tasks serviced by finite thread count ExecutorService
- Use EOGlobalIDs (or raw rows) as parameters to the child tasks.
  - DO NOT pass EOs directly to child tasks.
- Fixed thread pool
  - static var: pool is shared between all instances of the task (resource conservative)
  - instance var: pool is dedicated to each task (careful)
- Take care to correctly size the ERXTaskObjectStoreCoordinatorPool
  - er.extensions.concurrency.ERXTaskObjectStoreCoordinatorPool.maxCoordinators = n
- Use Futures to track child task completion. Don't exit until all child tasks have completed.

Working with Fixed Thread Pool ExecutorService

```
ExecutorService es = ERXExecutorService.newFiniteThreadPool(4);
boolean isRejected = true;
while ( isRejected ) {
    try {
        Future<?> future = es.submit(childTask);
    } catch (RejectedExecutionException e) {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e1) {
            // Handle that
        }
    }
}
```

Passing Parameters to a Task

```
 private final EOGlobalID _myObjectID

 // Example Task Constructor
public MyUpdateTask(MyEntityClass eo) {

     if (eo.isNewObject()) {
          throw new IllegalArgumentException("MyEntityClass cannot be a new unsaved object");
     }

     // Grab GID reference before the task is started.
     _myObjectID = eo.editingContext().globalIDForObject(eo);

}
```

Problems and Solutions (1/2)

- Memory when Processing Huge Data Sets
  - Allocate More Memory
  - Force garbage collection (for example if above a % usage)
  - Recycle EOEditingContexts periodically (see demo T06xxx.java)
- Large toMany Relationships
  - Remove the toMany from the EOModel and use ERXUnmodeledToManyRelationship
    - @see example usage in 'BackgroundTasks' demo:
      - TaskInfo.resultItems relationship.

Problems and Solutions (2/2)

- Prevent bottleneck in default EOObjectStoreCoordinator
  - ERXTaskObjectStoreCoordinatorPool.objectStoreCoordinator()
  - er.extensions.concurrency.ERXTaskObjectStoreCoordinatorPool.maxCoordinators = n
  - Implement IERXRefreshPage interface on result page to prevent stale EO result.
    - in refresh() call ERXEOControlUtilities.refreshObject(eo);
    - @see demo TaskInfoPage and logic in ERXNextPageForResultController
- Use Fresh Data in your background tasks
  - ec.setFetchTimestamp(System.currentTimeMillis());
  - ERXEOControlUtilities.refreshObject(eo);

More Information

- Demo App: wonder/Examples/Misc/BackgroundTasks
- Java API Docs
  - Runnable, Callable, ExecutorService, Future, Executors
- Wonder classes and javadoc
  - CCAjaxLongResponsePage
  - IERXPerformWOActionForResult / ERXNextPageForResultWOAction, IERXRefreshPage
  - ERXStatusInterface, ERXTaskPercentComplete, IERXStoppable
  - ERXExecutorService
  - ERXAbstractTask, ERXTaskObjectStoreCoordinatorPool
- Effective Java, 2nd Edition by Joshua Bloch, chapter 10

Q&A

Concurrency and Thread-Safe Data Processing in Background Tasks

Kieran Kelleher