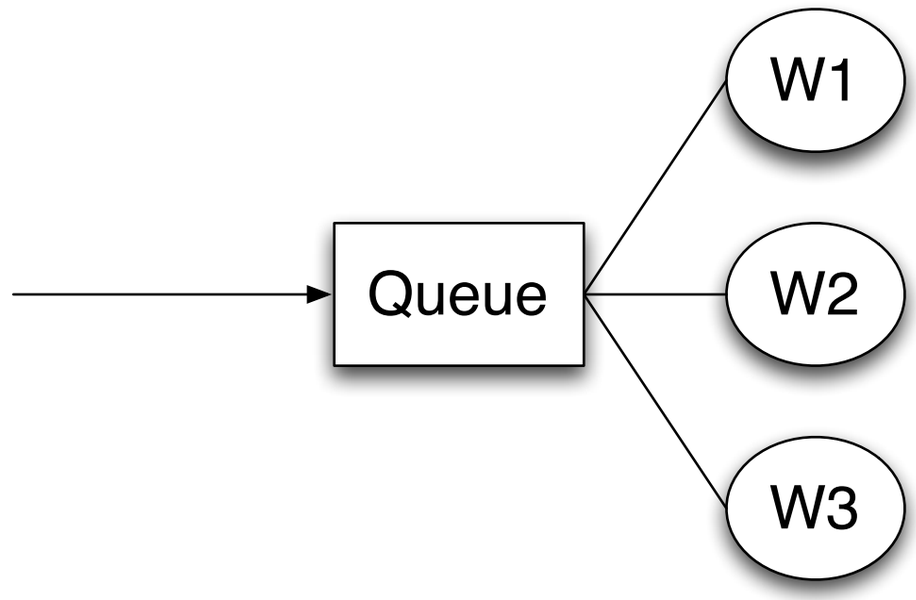
CS 683 Emerging Technologies
Fall Semester, 2008
Doc 9 Worker Pool
Sept 25 2008

Worker Pool



Worker

```
-module (worker).
-export ([start/2]).

start(Name, Queue) -> spawn(fun() -> loop(Name, Queue) end).

loop(Name, Queue) ->
      io:format("~w starting loop~n", [Name]),
   % Queue ! {worker, self()},
      erlang:send_after(Name*5000, Queue, {worker, self()}),
    receive
          {work, Request} ->
        perform_work(Name, Request),
        loop(Name, Queue)
    end.

perform_work (Name, Work) ->
     io:format("~w performing ~w~n", [Name,Work]).
```


erlang:send_after is used to help simulate the worker taking some time to perform the job.

Work Queue

```
-module (work_queue).
-export ([start/1, add_work/1, stop/0]).

start(WorkSize) ->
    register(work_queue, spawn(fun() -> loop([]) end)),
      add_workers(WorkSize).

stop() ->
      work_queue ! {stop}.

add_workers(0) -> void;
add_workers(N) ->
      worker:start(N, work_queue),
      add_workers(N-1).

add_work(Work) ->
      work_queue ! {work,Work}.
```

Work Queue

```
loop(Workers) ->
      io:format("queue starting loop~n"),
   receive
          {work, Request} when Workers /= []->
                io:format("work ~w with workers~n", [Request]),
                [Worker|OtherWorkers] = Workers,
                Worker ! {work, Request},
         loop(OtherWorkers);
          {worker, WorkerId} ->
                io:format("~w worker no work, # of workers ~w~n", [WorkerId, length(Workers) +
1]),
                loop([WorkerId|Workers]);
          {stop} ->
                io:format("Stoping~n")
   end.
```