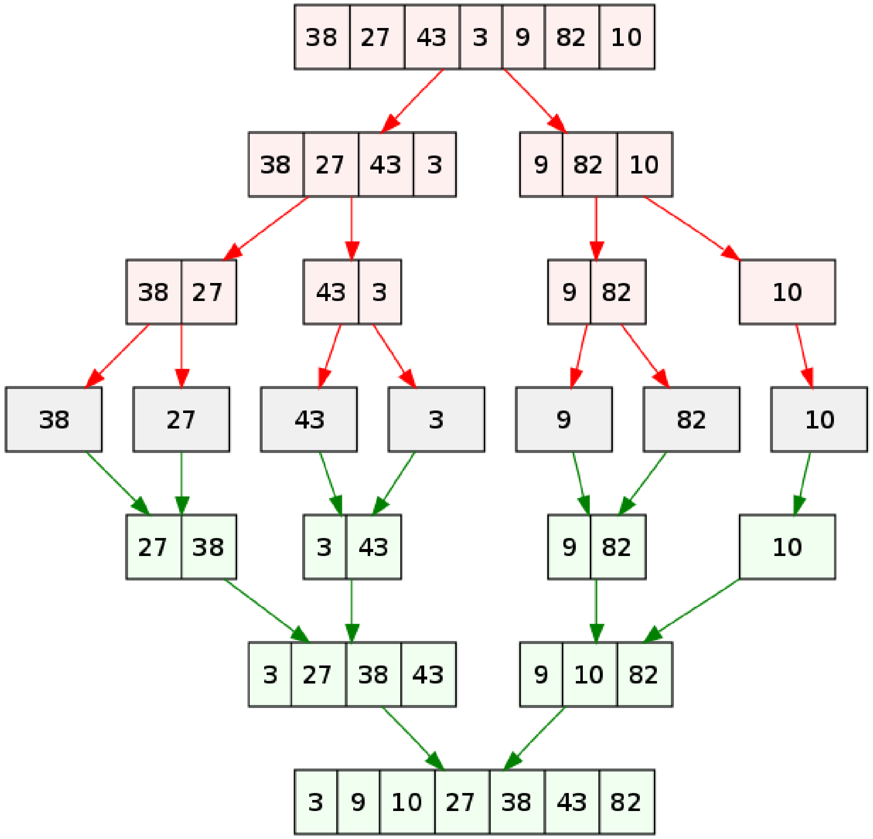
Parallel design patterns ARCHER course

Practical three: Divide and conquer with a process pool (master worker) for mergesort

Reusing this material

Mergesort

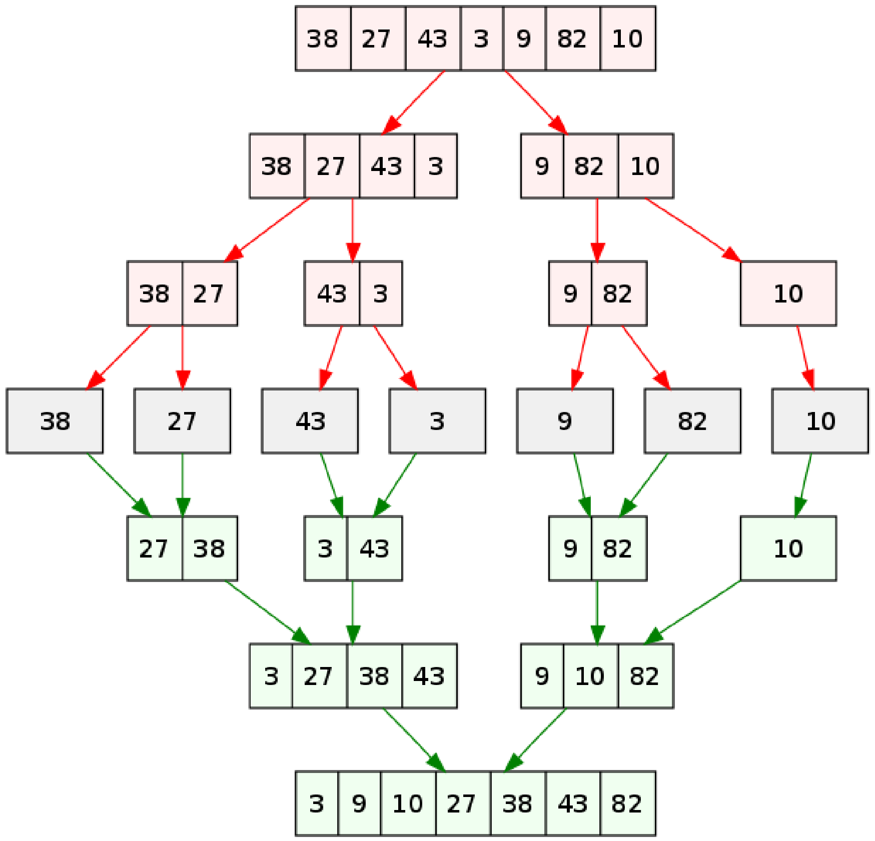
- Starting from some randomly generated, unsorted data.



- Repeatedly divide the data (problem) up until it is trivial to solve
- Then merge the small answers together to form the overall sorted list of numbers

- Maps very well to D&C pattern

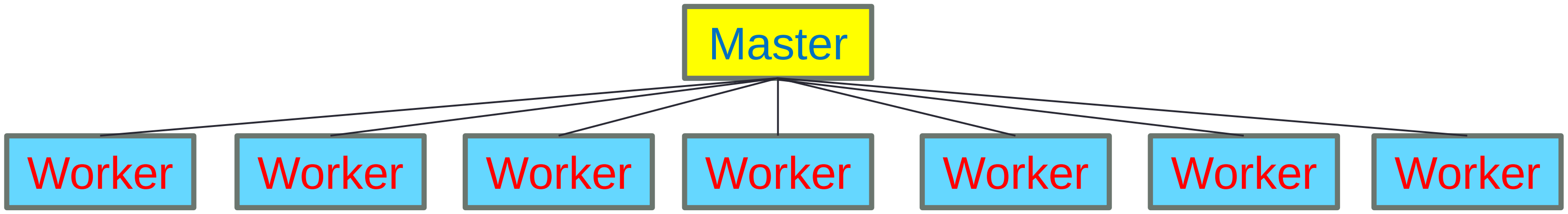
Parallel mergesort



- Each division is a task, working down to some serial threshold (in the image this is 1, but in reality you probably want it to be higher than this.)
- Remember from the lecture, only create one task for the first half of the data and use the existing task for the second half

Skeleton code is provided, your task is to hook it all up with MPI!

How to do the task generation?

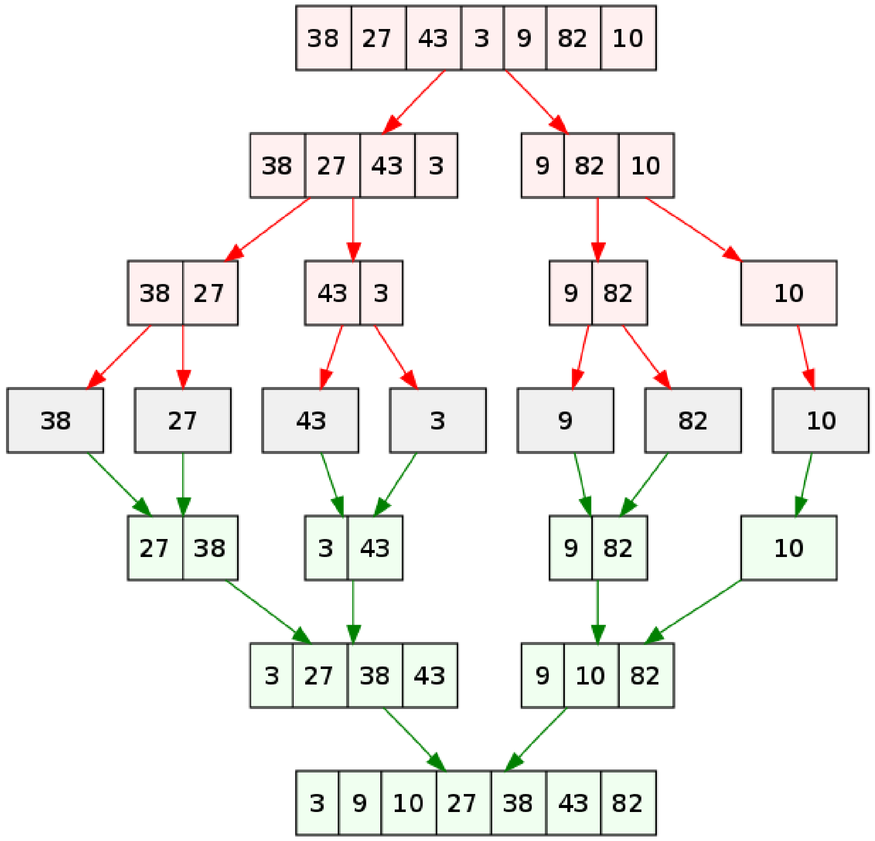


- Provide you with a process pool which implements the master worker pattern
  - The master keeps track of which worker UEs are currently busy
  - Workers sit there and wait for a command from the master to start
  - When a task requests a new worker from the master, the master sends back the rank of this new worker. The new worker is provided with the rank of its parent when it is started and from this the two UEs can communicate
    - i.e. the parent can tell the new worker what data it needs to process

Function	Description
int processPoolInit()	Initialises the process pool (1=worker, 2=master)
void processPoolFinalise()	Finalises and process pool (called from all)
int masterPoll()	Master polls to determine whether to continue or not
int workerSleep()	Worker waits for new task (1=new task, 0=stop)
int startWorkerProcess()	Starts a new worker task and returns the rank of this
int getCommandData()	Retrieves the rank of the task created this one
void shutdownPool()	Called by anyone to shut down the pool

Wash up

- Sample solutions are available
- The fact that the existing worker is *reused* for half of the data is really important here
  - As otherwise workers would be sitting idle waiting for their children to complete



Computation vs overhead

Data size	Serial threshold	Number workers	Task start up overhead	Comm time (s)	Compute Time (s)	Runtime (s)
100	10	16	6e-6	3.6e-5	2.0e-6	0.00125
1000	100	16	6e-6	4.0e-5	8.0e-6	0.00131
10000	2000	8	5e-6	1.2e-4	5.7e-4	0.00228
10000	1000	16	6e-6	8.4e-5	5.24e-4	0.00215
10000	100	128	1e-3	5.6e-3	2.2e-5	0.01559
1000000	100000	16	7e-6	3.1e-3	1.1e-2	0.05768
1000000	50000	32	5.7e-4	9.3e-3	5.6e-3	0.07008
1000000	10000	128	6.2e-4	1.3e-2	2.3e-3	0.08194
100000000	10000000	16	2.0e-5	0.34	1.50	6.27
100000000	1000000	128	5.8e-5	0.083	0.187	5.45